ESP32-P4

技术参考手册 预发布 v0.7

PRELIMINARY

ESPRESSIF

关于本文档

ESP32-P4 技术参考手册面向使用 ESP32-P4 系列产品进行底层软件开发的人员，介绍了 ESP32-P4 系列产品中内置的硬件模块，包括概述、功能列表、硬件架构、编程指南、寄存器列表等信息。

本文档中的跳转

在本文档中实现跳转，请参考以下建议：

- 发布进度速览（下一页）罗列了本文档中的所有章节，您可以从这里快速跳转至某个具体章节。
- 您还可以通过文档左侧的**书签**，从文中的任何位置直接跳转至另一个章节。注意，本文档已设置默认打开**书签**功能，但一些 PDF 阅读器或浏览器会忽略此设置。因此，如果您无法找到**书签**功能，请尝试以下方法：
  - 在您的浏览器中安装 PDF 阅读器拓展；
  - 下载本文档，使用本地 PDF 阅读器进行浏览；
  - 配置您的 PDF 阅读器，使其默认打开**书签**功能。
- 大多数 PDF 阅读器均支持跳转功能，允许您借助按钮、菜单选项或快捷键进行跳转（**向上**、**向下**、**向前**、**向后**、**后退**、**前进**及**前往页面**）等。
- 此外，您还可以使用本文档内置的 **GoBack** 按钮（每页右上角）快速后退至跳转之前的位置。注意，本功能仅适用于 Acrobat 系列的 PDF 阅读器（比如 Acrobat Reader 和 Adobe DC）以及内置 Acrobat 系列 PDF 阅读器或拓展的浏览器（比如 Firefox）。

发布进度速览

注意本文档尚未全部完成，具体发布进度见下表：

No.	章节	最新进度
第 I 卷：微处理器和主机		
1	总线架构	已发布
2	高性能处理器	已发布
3	RISC-V 追踪编码器 (TRACE)	已发布
4	处理器指令扩展 (PIE)	已发布
5	低功耗处理器	已发布
第 II 卷：系统 DMA		
6	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	已发布
7	VDMA 控制器 (VDMA)	已发布
8	2D-DMA 控制器 (2D-DMA)	已发布
第 III 卷：存储器组织结构		
9	系统和存储器	已发布
10	eFuse 控制器 (EFUSE)	已发布
第 IV 卷：系统组件		
11	GPIO 交换矩阵和 IO MUX	已发布
12	复位和时钟	已发布
13	芯片 Boot 控制	已发布
14	中断矩阵	已发布
15	事件任务矩阵 (ETM)	已发布
16	低功耗管理	已发布
17	系统定时器	已发布
18	定时器组 (TIMG)	已发布
19	看门狗定时器 (WDT)	已发布
20	实时时钟定时器	已发布
21	权限控制 (PMS)	已发布
22	系统寄存器 (SYSREG)	已发布
23	辅助调试	已发布
24	AXI 性能监测器	已发布
25	LP 信箱控制器	已发布
26	欠压检测器	已发布
27	电源检测器 (PSDET)	已发布
第 V 卷：加密和安全组件		
28	AES 加速器 (AES)	已发布
29	ECC 加速器 (ECC)	已发布
30	HMAC 加速器 (HMAC)	已发布
31	RSA 加速器 (RSA)	已发布
32	SHA 加速器 (SHA)	已发布
33	RSA 数字签名外设 (RSA_DS)	已发布
34	ECDSA 数字签名外设 (ECDSA_DS)	已发布
35	片外存储器加密与解密 (XTS_AES)	已发布

No.	章节	最新进度
36	随机数发生器 (RNG)	已发布
37	密钥管理器	已发布
第 VI 卷：图像和语音处理		
38	JPEG 图像编解码器	已发布
39	图像信号处理器 (ISP)	已发布
40	像素处理加速器 (PPA)	已发布
41	LCD 与 Camera 控制器 (LCD_CAM)	已发布
42	H264 编码器	已发布
43	MIPI CSI	已发布
44	MIPI DSI	已发布
45	语音活动检测 (VAD)	已发布
第 VII 卷：通讯接口		
46	UART 控制器 (UART)	已发布
47	SPI 控制器 (SPI)	已发布
48	I2C 控制器 (I2C)	已发布
49	模拟 I2C 控制器	已发布
50	I3C 控制器 [to be added later]	86%
51	I2S 控制器 (I2S)	已发布
52	LP I2S 控制器	已发布
53	脉冲计数控制器 (PCNT)	已发布
54	USB 2.0 高速 OTG	已发布
55	USB 2.0 全速 OTG	已发布
56	USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)	已发布
57	以太网介质访问控制器 (EMAC)	已发布
58	双线汽车接口 (TWAI)	已发布
59	SD/MMC 主机控制器 (SDHOST)	已发布
60	LED PWM 控制器 (LEDC)	已发布
61	电机控制脉宽调制器 (MCPWM)	已发布
62	红外遥控 (RMT)	已发布
63	并行 IO 控制器 (PARLIO)	已发布
64	比特调节器	已发布
第 VIII 卷：模拟信号处理		
65	触摸传感器 (TOUCH)	已发布
66	温度传感器 (TSENS)	已发布
67	ADC 控制器 (ADC)	已发布
68	模拟电压比较器	已发布

说明：

点击链接或扫描二维码确保您使用的是最新版本的文档：
https://www.espressif.com/documentation/esp32-p4_technical_reference_manual_cn.pdf

目录

I 微处理器和主机 64

1 总线架构 65

2 高性能处理器 68
2.1 概述 68
2.2 特性 68
2.3 术语 70
2.4 地址分布 70
2.5 配置和状态寄存器 (CSRs) 71
2.5.1 寄存器列表 71
2.5.2 寄存器描述 74
2.6 RISC-V 标准 ISA 扩展支持 110
2.6.1 M 扩展 110
2.6.1.1 概述 110
2.6.1.2 功能描述 110
2.6.2 C 扩展 110
2.6.2.1 概述 110
2.6.2.2 功能描述 110
2.6.3 F 扩展 111
2.6.3.1 概述 111
2.6.3.2 功能描述 111
2.6.3.3 初始化 111
2.6.4 Zc (Z) 扩展 112
2.6.4.1 概述 112
2.6.4.2 功能描述 112
2.6.4.3 Zc 指令 113
2.6.4.4 限制 114
2.6.5 位操作 (Zb) 扩展 115
2.6.5.1 概述 115
2.6.5.2 功能描述 115
2.6.6 A 扩展 116
2.6.6.1 概述 116
2.6.6.2 功能描述 116
2.6.6.3 加载保留字 (LR.W) 指令 116
2.6.6.4 条件存入字 (SC.W) 指令 116
2.6.6.5 AMO 指令 117
2.7 自定义指令扩展 118
2.7.1 硬件循环 118
2.7.1.1 概述 118
2.7.1.2 特性 118
2.7.1.3 功能描述 118

2.7.1.4 HWLP 中支持的指令/操作/模式 118
2.7.1.5 HWLP 限制 119
2.7.1.6 寄存器列表 119
2.7.1.7 寄存器描述 119
2.7.1.8 HWLP 指令 124
2.7.2 处理器指令扩展 125
2.7.2.1 概述 125
2.7.2.2 功能描述 125
2.7.2.3 初始化、上下文切换和异常 125
2.8 存储器映射寄存器 127
2.8.1 概述 127
2.8.2 特性 127
2.8.3 功能描述 127
2.9 中断控制器 128
2.9.1 概述 128
2.9.2 CLIC 128
2.9.2.1 概述 128
2.9.2.2 CLIC CSRs 128
2.9.2.3 中断映射 128
2.9.2.4 CLIC 参数 129
2.9.2.5 中断级别和优先级编码 129
2.9.2.6 CLIC 存储器映射寄存器列表 130
2.9.2.7 CLIC 存储器映射寄存器描述 131
2.9.3 核心本地中断 (CLINT) 134
2.9.3.1 概述 134
2.9.3.2 特性 134
2.9.3.3 软件中断 134
2.9.3.4 定时器计数器和中断 134
2.9.3.5 寄存器列表 134
2.9.3.6 寄存器描述 135
2.10 存储器保护单元 139
2.10.1 标准物理存储器保护 139
2.10.1.1 概述 139
2.10.1.2 特性 139
2.10.1.3 功能描述 139
2.10.1.4 寄存器列表 140
2.10.1.5 寄存器描述 140
2.10.2 自定义物理存储器属性 (PMA) 检查器 144
2.10.2.1 概述 144
2.10.2.2 特性 144
2.10.2.3 功能描述 144
2.10.2.4 寄存器列表 145
2.10.2.5 寄存器 146
2.11 调试和追踪 148
2.11.1 调试 148
2.11.1.1 概述 148

2.11.1.2 功能 149
2.11.1.3 功能描述 149
2.11.1.4 JTAG 控制 149
2.11.1.5 寄存器 150
2.11.1.6 寄存器描述 151
2.11.2 调试暂停组 155
2.11.2.1 概述 155
2.11.2.2 特性 155
2.11.2.3 功能描述 155
2.11.2.4 寄存器列表 156
2.11.2.5 寄存器描述 156
2.11.3 硬件触发器 157
2.11.3.1 概述 157
2.11.3.2 特性 157
2.11.3.3 功能描述 157
2.11.3.4 触发执行流程 158
2.11.3.5 寄存器列表 158
2.11.3.6 寄存器描述 159
2.11.4 追踪 164
2.11.4.1 概述 164
2.11.4.2 特性 164
2.11.4.3 功能描述 164
2.12 性能 165
2.12.1 分支预测 165
2.12.1.1 概述 165
2.12.1.2 特性 165
2.12.1.3 功能描述 165
2.12.2 RAS 166
2.12.2.1 概述 166
2.12.2.2 特性 166
2.12.2.3 功能描述 166
2.12.3 性能配置控制状态寄存器 167
2.13 自定义特性 168
2.13.1 总线错误响应 168
2.13.1.1 概述 168
2.13.1.2 功能描述 168
2.13.1.3 控制状态寄存器 168
2.13.2 专用 IO 170
2.13.2.1 概述 170
2.13.2.2 特性 170
2.13.2.3 功能描述 170
2.13.2.4 寄存器列表 171
2.13.2.5 寄存器 171
2.13.3 运行暂停 (RunStall) 173
2.13.3.1 概述 173
2.13.3.2 功能描述 173

2.13.3.3 寄存器列表 173
2.13.4 调试辅助信息 174
2.13.5 核心锁定 174
2.13.5.1 概述 174
2.13.5.2 功能描述 174
2.13.5.3 寄存器列表 174
2.13.5.4 寄存器描述 174
2.13.6 外部唤醒 174
2.13.6.1 概述 174
2.13.6.2 功能描述 174

3 RISC-V 追踪编码器 (TRACE) 175
3.1 术语 175
3.2 介绍 175
3.3 特性 176
3.4 架构概览 178
3.5 功能描述 178
3.5.1 同步 178
3.5.2 地址模式 178
3.5.3 可选边带信号 179
3.5.4 过滤 179
3.5.5 锚定 180
3.5.6 写存储器模式 180
3.5.7 自动重启 180
3.6 编码器输出数据包 180
3.6.1 头部 180
3.6.2 索引 181
3.6.3 有效载荷 181
3.6.3.1 格式 3 181
3.6.3.2 格式 2 183
3.6.3.3 格式 1 183
3.7 中断 185
3.8 编程流程 185
3.8.1 配置编码器选项 185
3.8.2 配置过滤器 186
3.8.3 使能编码器 187
3.8.4 关闭编码器 188
3.8.5 通知 188
3.8.6 解码数据包 189
3.8.7 AHB 配置 189
3.8.8 软件保留 189
3.9 寄存器列表 190
3.10 寄存器 191

4 处理器指令扩展 (PIE) 202
4.1 概述 202

4.2 主要特性 202
4.3 结构概述 202
4.3.1 内部结构示意图 202
4.3.2 向量寄存器组 203
4.3.3 运算逻辑单元 (ALU) 203
4.3.4 乘法累加 (MAC) 单元 204
4.3.4.1 QACC 累加寄存器 204
4.3.4.2 XACC 累加寄存器 204
4.3.5 地址单元 204
4.4 扩展指令集组件 204
4.4.1 寄存器 204
4.4.1.1 通用寄存器 205
4.4.1.1.1 AR 205
4.4.1.2 向量寄存器 205
4.4.1.2.1 QR 205
4.4.1.3 专用寄存器 205
4.4.1.3.1 SAR 206
4.4.1.3.2 SAR_BYTE 206
4.4.1.3.3 FFT_BIT_WIDTH 206
4.4.1.3.4 UA_STATE 206
4.4.1.3.5 内核配置与状态寄存器 207
4.4.2 数据格式及对齐 207
4.4.3 非对齐访存处理 207
4.4.3.1 强制对齐模式 208
4.4.3.2 硬件非对齐模式 208
4.4.4 数据溢出与饱和 208
4.4.5 数据舍入 208
4.5 扩展指令列表 209
4.5.1 读内存指令 212
4.5.2 写内存指令 213
4.5.3 数据交换指令 214
4.5.4 运算指令 215
4.5.4.1 向量加法指令 215
4.5.4.2 向量乘法指令 215
4.5.4.3 向量复数乘法指令 216
4.5.4.4 向量乘累加指令 216
4.5.4.5 向量和标量乘累加指令 217
4.5.4.6 其他指令 217
4.5.5 比较指令 218
4.5.6 按位逻辑操作指令 219
4.5.7 移位指令 219
4.5.8 快速傅里叶变换 (FFT) 指令 220
4.5.8.1 蝶形运算指令 220
4.5.8.2 比特反转指令 220
4.5.8.3 实数 FFT 指令 220
4.6 指令性能 220

4.6.1 数据冒险 221
4.6.2 硬件资源冒险 222
4.6.3 控制冒险 222
4.7 扩展指令功能描述 223
4.7.1 指令符号与域定义 223
4.7.2 指令列表 223

5 低功耗处理器 616
5.1 概述 616
5.2 特性 616
5.3 配置与状态寄存器 (CSR) 617
5.3.1 寄存器列表 617
5.3.2 寄存器 618
5.4 中断和异常 626
5.4.1 中断 626
5.4.2 中断处理 627
5.4.3 异常 627
5.5 调试 627
5.5.1 特性 628
5.5.2 功能描述 628
5.5.3 寄存器列表 628
5.5.4 寄存器 629
5.6 硬件触发器 630
5.6.1 特性 630
5.6.2 功能描述 630
5.6.3 触发执行流程 631
5.6.4 寄存器列表 631
5.6.5 寄存器 631
5.7 性能计数器 634
5.8 系统访问 635
5.8.1 存储器访问 635
5.8.2 外设访问 635
5.9 事件任务矩阵功能 635
5.10 睡眠和唤醒流程 636
5.10.1 特性 636
5.10.2 流程 636
5.10.3 唤醒源 637
5.10.4 拒绝睡眠 638

II 系统 DMA 639

6 通用 DMA 控制器 (GDMA-AHB, GDMA-AXI) 640
6.1 概述 640
6.2 特性 641
6.3 架构 642
6.4 功能描述 642

6.4.1 链表 643
6.4.2 外设到存储及存储到外设的数据传输 644
6.4.3 存储到存储数据传输 645
6.4.4 启动 GDMA 645
6.4.5 读链表 646
6.4.6 GDMA-AXI 切换链表的中断响应机制 646
6.4.7 数据传输结束标志 646
6.4.8 访问存储 647
6.4.9 仲裁 648
6.4.10 总线错误响应信息记录 648
6.4.11 自动时钟门控 649
6.4.12 CRC 校验计算功能 649
6.5 事件任务矩阵功能 650
6.6 中断 651
6.7 编程流程 652
6.7.1 GDMA TX 通道配置流程 653
6.7.2 GDMA RX 通道配置流程 653
6.7.3 GDMA 存储器到存储器配置流程 653
6.7.4 通道优先级和权重配置流程 653
6.7.5 CRC 计算配置流程 654
6.8 寄存器列表 655
6.8.1 GDMA-AHB 寄存器列表 655
6.8.2 GDMA-AXI 寄存器列表 662
6.9 寄存器 669
6.9.1 GDMA-AHB 寄存器 669
6.9.2 GDMA-AXI 寄存器 708

7 VDMA 控制器 (VDMA) 750
7.1 概述 750
7.2 术语 750
7.3 特性 750
7.4 结构概览 751
7.5 功能描述 753
7.5.1 传输层次 753
7.5.2 仲裁机制 755
7.5.2.1 读仲裁器 755
7.5.2.2 写仲裁器 756
7.5.3 握手接口 756
7.5.4 传输控制 757
7.5.4.1 单块传输 757
7.5.4.2 多块传输 757
7.5.5 流控制器 762
7.5.6 通道挂起和恢复 762
7.5.7 通道禁用 762
7.5.7.1 传输完成前禁用被挂起的通道 762
7.5.7.2 传输完成前禁用未被挂起的通道 762

7.5.8 低功耗技术 763
7.5.8.1 DMA 通道低功耗技术 763
7.5.8.2 从机总线接口上的低功耗技术 763
7.5.8.3 AXI 主机接口通道上的低功耗技术 763
7.5.8.4 全局低功耗技术 764
7.6 中断 764
7.7 配置流程 765
7.7.1 通用配置流程 765
7.7.2 配置基于影子寄存器的多块传输 765
7.7.3 配置基于链表的多块传输 766
7.7.4 配置单块传输 767
7.8 寄存器列表 768
7.9 寄存器 771

8 2D-DMA 控制器 (2D-DMA) 800
8.1 概述 800
8.2 特性 800
8.3 架构 800
8.4 功能描述 801
8.4.1 传输模式 801
8.4.2 链表 802
8.4.3 2D-MOD1 模式和 DSCR-PORT 模式的补数方式 804
8.4.4 外设到存储及存储到外设的数据传输 806
8.4.5 存储到存储数据传输 806
8.4.6 宏块重排功能 806
8.4.6.1 发送方向宏块重排功能 806
8.4.6.2 接收方向宏块重排功能 807
8.4.7 颜色空间转换功能 808
8.4.8 启动 2D-DMA 811
8.4.9 读链表 811
8.4.10 数据传输结束标志 811
8.4.11 访问存储 812
8.4.12 仲裁 812
8.5 事件任务矩阵功能 813
8.6 中断 814
8.7 编程流程 815
8.7.1 2D-DMA 通用配置 815
8.7.2 2D-DMA 模式专属配置 815
8.7.3 2D-DMA TX 通道配置流程 816
8.7.4 2D-DMA RX 通道配置流程 816
8.7.5 2D-DMA 存储器到存储器配置流程 816
8.7.6 通道优先级和权重配置流程 817
8.7.7 工作状态下复位流程 817
8.8 寄存器列表 818
8.9 寄存器 825

III 存储器组织结构 868

9 系统和存储器 869
9.1 概述 869
9.2 主要特性 869
9.3 功能描述 870
9.3.1 地址映射 870
9.3.2 内部存储器 873
9.3.3 外部存储器 877
9.3.3.1 外部存储器地址映射 877
9.3.3.2 cache 877
9.3.3.3 cache 操作 878
9.3.4 DMA 地址空间 879
9.3.5 模块/外设地址映射 880
9.4 eFuse 控制器 (EFUSE) 884
9.4.1 概述 884
9.4.2 主要特性 884
9.4.3 功能描述 884
9.4.3.1 结构 884
9.4.3.1.1 EFUSE_WR_DIS 893
9.4.3.1.2 EFUSE_RD_DIS 893
9.4.3.1.3 数据存储方式 893
9.4.3.2 烧写参数 894
9.4.3.3 用户读取参数 896
9.4.3.4 eFuse VDDQ 时序 897
9.4.3.5 硬件模块使用参数 898
9.4.4 中断 898
9.4.5 寄存器列表 899
9.4.6 寄存器 906

IV 系统组件 943

10 GPIO 交换矩阵和 IO MUX 944
10.1 概述 944
10.2 主要特性 944
10.2.1 HP GPIO 交换矩阵和 HP IO MUX 944
10.2.2 LP GPIO 交换矩阵和 LP IO MUX 945
10.3 结构概览 945
10.4 通过 GPIO 交换矩阵的外设输入 947
10.4.1 概述 948
10.4.2 信号同步 948
10.4.3 GPIO 滤波器 949
10.4.4 毛刺滤波器 949
10.4.5 简单 GPIO 输入 949
10.4.6 GPIO 唤醒 950
10.4.6.1 HP GPIO 唤醒 950

10.4.6.2 LP GPIO 唤醒 950
10.4.7 编程流程 951
10.4.7.1 HP GPIO 交换矩阵 951
10.4.7.2 LP GPIO 交换矩阵 952
10.5 通过 GPIO 交换矩阵的外设输出 952
10.5.1 概述 952
10.5.2 简单 GPIO 输出 953
10.5.3 Sigma Delta 调制输出 (SDM) 953
10.5.3.1 功能描述 953
10.5.3.2 配置方法 954
10.5.4 编程流程 954
10.5.4.1 HP GPIO 交换矩阵 954
10.5.4.2 LP GPIO 交换矩阵 955
10.6 IO MUX 的直接输入输出功能 955
10.6.1 概述 955
10.6.2 功能描述 955
10.6.2.1 HP IO MUX 956
10.6.2.2 LP IO MUX 956
10.7 模拟功能描述 956
10.7.1 概述 956
10.7.2 模拟功能描述 956
10.8 Light-sleep 模式管脚功能 957
10.9 GPIO 管脚的 Hold 特性 957
10.10 GPIO 管脚的迟滞特性 959
10.11 GPIO 管脚供电和电源管理 960
10.11.1 GPIO 管脚供电 960
10.11.2 电源管理 960
10.12 HP 外设信号列表 960
10.13 LP 外设信号列表 971
10.14 HP IO MUX 管脚功能列表 973
10.15 LP IO MUX 管脚功能列表 976
10.16 GPIO 管脚模拟功能列表 977
10.17 事件任务矩阵 (ETM) 功能 978
10.18 中断 979
10.19 寄存器列表 980
10.19.1 HP GPIO 交换矩阵寄存器列表 980
10.19.2 HP IO MUX 寄存器列表 982
10.19.3 GPIO EXT 寄存器列表 983
10.19.4 LP GPIO 交换矩阵寄存器列表 984
10.19.5 LP IO MUX 寄存器 986
10.20 寄存器 987
10.20.1 HP GPIO 交换矩阵寄存器 987
10.20.2 HP IO MUX 寄存器 1007
10.20.3 GPIO EXT 寄存器 1010
10.20.4 LP GPIO 矩阵寄存器 1019
10.20.5 LP IO MUX 寄存器 1028

11 复位和时钟 1033
11.1 复位 1033
11.1.1 概述 1033
11.1.2 结构图 1033
11.1.3 特性 1033
11.1.4 功能描述 1034
11.1.5 外设复位 1035
11.2 时钟 1036
11.2.1 概述 1036
11.2.2 结构图 1037
11.2.3 特性 1037
11.2.4 功能描述 1038
11.2.4.1 高性能系统时钟 1038
11.2.4.2 低功耗系统时钟 1039
11.2.4.3 外设时钟 1040
11.3 配置流程 1044
11.3.1 高性能系统时钟配置 1044
11.3.2 低功耗系统时钟配置 1044
11.3.3 外设时钟复位配置 1045
11.4 寄存器列表 1047
11.4.1 复位与时钟 (HP_SYS_CLKRST) 寄存器列表 1047
11.4.2 LP 常开时钟与复位 (LP_CLKRST) 寄存器列表 1048
11.4.3 LP 外设时钟与复位 (LPPERI) 寄存器列表 1049
11.5 寄存器 1050
11.5.1 复位与时钟 (HP_SYS_CLKRST) 寄存器 1050
11.5.2 LP 常开时钟与复位 (LP_CLKRST) 寄存器 1133
11.5.3 LP 外设时钟与复位 (LPPERI) 寄存器 1153

12 芯片 Boot 控制 1162
12.1 概述 1162
12.2 功能描述 1162
12.2.1 默认配置 1162
12.2.2 Boot 模式控制 1163
12.2.3 ROM 代码日志打印控制 1165
12.2.4 JTAG 信号源控制 1165

13 中断矩阵 1167
13.1 概述 1167
13.2 ESP32-P4 中断术语 1167
13.2.1 中断 1167
13.2.2 中断信号/中断源 1167
13.2.3 ESP32-P4 中断流 1168
13.3 特性 1168
13.4 功能描述 1169
13.4.1 外部中断源 1169
13.4.2 分配外部中断源至 HP CPU 外部中断 1175

13.4.2.1 分配一个外部中断源 *SOURCE* 至 HP CPU*x* 外部中断 1175
13.4.2.2 分配多个外部中断源 *SOURCE* 至 HP CPU*x* 外部中断 1175
13.4.2.3 关闭 HP CPU*x* 外部中断源 *SOURCE* 1175
13.4.3 查询 HP CPU*x* 外部中断源当前的中断状态 1175
13.4.4 中断重映射 1175
13.5 寄存器列表 1177
13.5.1 HP CPU0 中断矩阵寄存器列表 1177
13.5.2 HP CPU1 中断矩阵寄存器列表 1182
13.6 寄存器 1189
13.6.1 HP CPU0 中断矩阵寄存器 1189
13.6.2 HP CPU1 中断矩阵寄存器 1199

14 事件任务矩阵 (ETM) 1210
14.1 概述 1210
14.2 特性 1210
14.3 功能描述 1210
14.3.1 架构 1210
14.3.2 事件 1211
14.3.3 任务 1216
14.3.4 事件任务触发状态 1221
14.3.5 时序考虑因素 1221
14.3.6 通道控制 1222
14.4 寄存器列表 1224
14.5 寄存器 1228

15 低功耗管理 1360
15.1 概述 1360
15.2 术语 1360
15.3 主要特性 1360
15.4 功能描述 1361
15.4.1 电源结构 1361
15.4.1.1 调压器 1362
15.4.1.2 数字电源域 1362
15.4.1.3 模拟电源域 1363
15.4.1.4 电池电源域 1363
15.4.2 PMU 1363
15.4.2.1 PMU 主状态机 1364
15.4.2.2 睡眠与唤醒控制器 1365
15.4.2.3 模拟电源控制器 1366
15.4.2.4 数字电源控制器 1367
15.4.2.5 时钟控制器 1368
15.4.2.6 数据备份控制器 1370
15.4.2.7 系统控制器 1371
15.4.2.8 电池供电控制器 1371
15.4.2.9 输出调压器控制 1371
15.4.2.9.1 调压器特性 1371

15.4.2.9.2 板级用法示例 1373
15.4.2.9.3 软件配置步骤 1374
15.5 功耗模式 1374
15.6 事件任务矩阵功能 1375
15.7 中断 1375
15.8 寄存器列表 1377
15.9 寄存器 1379

16 系统定时器 1439
16.1 概述 1439
16.2 主要特性 1439
16.3 系统定时器架构 1440
16.4 时钟源选择 1440
16.5 功能描述 1441
16.5.1 计数器 1441
16.5.2 比较器和报警 1442
16.5.3 事件任务矩阵 1443
16.5.4 同步操作 1443
16.6 中断 1443
16.7 编程示例 1444
16.7.1 读取当前计数器的值 1444
16.7.2 在单次报警模式下配置一次性报警 1444
16.7.3 在周期报警模式下配置周期性报警 1445
16.7.4 唤醒后时间补偿 1445
16.8 寄存器列表 1446
16.9 寄存器 1448

17 定时器组 (TIMG) 1463
17.1 概述 1463
17.2 主要特性 1463
17.3 功能描述 1464
17.3.1 16 位预分频器与时钟选择器 1464
17.3.2 54 位时基计数器 1464
17.3.3 警报生成 1465
17.3.4 定时器重新加载 1466
17.3.5 事件任务矩阵功能 1466
17.3.6 TIMGO_CALI_CLK 频率计算 1467
17.3.7 中断 1467
17.4 配置与使用 1468
17.4.1 定时器用作简单时钟 1468
17.4.2 定时器用于单次报警 1468
17.4.3 通过 APB 设置定时器用于周期性报警 1468
17.4.4 通过 ETM 设置定时器用于周期性报警 1469
17.4.5 RTC_SLOW_CLK 频率计算 1470
17.5 寄存器列表 1471
17.6 寄存器 1473

18 看门狗定时器 (WDT) 1485
18.1 概述 1485
18.2 数字看门狗定时器 1486
18.2.1 主要特性 1486
18.2.2 功能描述 1487
18.2.2.1 时钟源与 32 位计数器 1487
18.2.2.2 阶段与超时动作 1488
18.2.2.3 写保护 1488
18.2.2.4 flash 引导保护 1489
18.3 超级看门狗定时器 1489
18.3.1 主要特性 1489
18.3.2 SWD 控制器 1489
18.3.2.1 结构 1490
18.3.2.2 工作流程 1490
18.4 中断 1490
18.5 寄存器列表 1491
18.6 寄存器 1492

19 实时时钟定时器 1500
19.1 概述 1500
19.2 主要特性 1500
19.3 功能描述 1500
19.4 事件任务矩阵功能 1501
19.5 中断 1502
19.6 寄存器列表 1503
19.7 寄存器 1504

20 权限控制 (PMS) 1512
20.1 概述 1512
20.2 主要特性 1513
20.3 功能描述 1514
20.3.1 结构概览 1514
20.3.2 地址区间及访问权限 1516
20.3.2.1 HP APM 和 LP APM 管理的地址区间及访问权限 1516
20.3.2.2 DMA APM 管理的地址区间及访问权限 1516
20.4 配置流程 1517
20.4.1 配置 HP CPU0/1 的访问权限 1517
20.4.2 配置 LP CPU 的访问权限 1517
20.4.3 配置其他 DMA 主机的访问权限 1518
20.5 越权访问与中断 1518
20.6 寄存器列表 1519
20.6.1 HP_DMA_PMS_REG 1519
20.6.2 HP_PERI_PMS_REG 1522
20.6.3 LP2HP_PERI_PMS_REG 1523
20.6.4 LP_PERI_PMS_REG 1523
20.6.5 HP2LP_PERI_PMS_REG 1524

20.7 寄存器 1525
20.7.1 HP_DMA_PMS_REG 1525
20.7.2 HP_PERI_PMS_REG 1552
20.7.3 LP2HP_PERI_PMS_REG 1575
20.7.4 LP_PERI_PMS_REG 1587
20.7.5 HP2LP_PERI_PMS_REG 1595

21 系统寄存器 (SYSREG) 1605
21.1 概述 1605
21.2 功能描述 1605
21.2.1 HP 系统寄存器 1605
21.2.1.1 外部存储器加密/解密配置 1605
21.2.1.2 HP 外设时钟配置与电源控制 1605
21.2.1.3 HP Cache 时钟与复位配置 1605
21.2.1.4 HP SPM 与 HP L2MEM 时钟配置 1606
21.2.1.5 HP SPM 奇偶校验配置 1606
21.2.1.6 HP L2MEM ECC 校验配置 1606
21.2.1.7 比特调节器 (BitScrambler) 配置 1607
21.2.1.8 以太网 MAC 控制 1607
21.2.1.9 USB 2.0 高速 OTG 控制 1607
21.2.1.10 CPU 控制与记录 1608
21.2.1.11 HP 到 LP 中断与唤醒控制 1608
21.2.1.12 HP GPIO 保持控制 1608
21.2.1.13 HP GPIO 输出控制 1608
21.2.1.14 非法访问与未授权访问 1608
21.2.1.15 HP CPU 总线超时保护 1609
21.2.1.16 AXI 总线矩阵 1609
21.2.1.17 加速写 1610
21.2.2 LP 系统寄存器 1611
21.2.2.1 LP 定时器 Stall 控制 1611
21.2.2.2 聚焦离子束 (FIB) 控制 1611
21.2.2.3 启动模式 1611
21.2.2.4 LP CPU 控制 1611
21.2.2.5 复位控制 1612
21.2.2.6 唤醒控制 1613
21.2.2.7 Pad 保持控制 1613
21.2.2.8 放电控制 1613
21.2.2.9 LPROM 和 LP SPM 时钟控制 1613
21.2.2.10 模拟电压比较器控制 1613
21.2.2.11 非法访问与未授权访问 1614
21.2.2.12 LP CPU 总线超时保护 1614
21.2.2.13 随机数生成器控制 1615
21.3 中断 1615
21.4 寄存器列表 1617
21.4.1 HP 寄存器列表 1617
21.4.2 ICM 寄存器列表 1619

21.4.3 LP 寄存器列表 1620
21.5 寄存器 1622
21.5.1 HP 寄存器 1622
21.5.2 ICM Register 1674
21.5.3 LP 寄存器 1684

22 辅助调试 1709
22.1 概述 1709
22.2 主要特性 1709
22.3 功能描述 1709
22.3.1 区域读写监测 1709
22.3.2 栈指针监测 1709
22.3.3 PC 记录 1710
22.3.4 CPU/DMA 总线访问记录 1710
22.4 中断 1710
22.5 工作流程 1711
22.5.1 区域监测和栈指针监测配置 1711
22.5.2 PC 记录配置 1713
22.5.3 CPU/DMA 总线访问记录配置 1714
22.5.3.1 HP CPUO/1 总线访问记录配置 1714
22.5.3.2 DMA 总线访问记录配置 1716
22.6 寄存器列表 1719
22.6.1 HP CPU 总线记录配置寄存器列表 1719
22.6.2 DMA 总线记录配置寄存器列表 1719
22.6.3 其它寄存器列表 1720
22.7 寄存器 1723
22.7.1 HP CPU 总线记录配置寄存器 1723
22.7.2 DMA 总线记录配置寄存器 1730
22.7.3 其它寄存器 1737

23 AXI 性能监测器 1763
23.1 概述 1763
23.2 术语 1763
23.3 主要特性 1764
23.4 结构概览 1764
23.5 功能描述 1765
23.5.1 时钟 1765
23.5.2 复位 1765
23.5.3 传输 ID 过滤 1765
23.5.4 监测指标 1766
23.5.4.1 读/写地址通道传输事务最大 outstanding 数量 1767
23.5.4.2 读/写地址通道延迟 1768
23.5.4.3 读/写数据通道延迟 1768
23.5.4.4 写数据通道 WVALID 空隙 1768
23.5.4.5 读/写事务传输延迟 1768
23.5.4.6 平均带宽测量 1769

23.5.4.7 瞬时带宽测量 1769
23.6 中断 1770
23.7 配置流程 1770
23.7.1 使用 AXI Perf Mon 测量 outstanding 1770
23.7.2 使用 AXI Perf Mon 测量读平均带宽 1771
23.7.3 使用 AXI Perf Mon 测量读瞬时带宽 1771
23.8 寄存器列表 1773
23.9 寄存器 1779

24 LP 信箱控制器 1801
24.1 概述 1801
24.2 特性 1801
24.3 功能描述 1802
24.3.1 信息寄存器 1802
24.3.2 中断 1802
24.3.3 核间通信 1803
24.4 寄存器列表 1803
24.5 寄存器 1805

25 欠压检测器 1818
25.1 概述 1818
25.2 主要特性 1818
25.3 结构概览 1819
25.4 功能描述 1819
25.5 中断 1822
25.6 配置流程 1822
25.7 寄存器列表 1823
25.8 寄存器 1824

26 电源检测器 (PSDET) 1832
26.1 概述 1832
26.2 主要特性 1832
26.3 功能描述 1832

V 加密和安全组件 1833

27 AES 加速器 (AES) 1834
27.1 概述 1834
27.2 主要特性 1834
27.3 时钟和复位 1834
27.4 工作模式简介 1835
27.5 Typical AES 工作模式 1836
27.5.1 密钥、明文、密文 1836
27.5.2 字节序 1836
27.5.3 Typical AES 工作流程 1839
27.6 DMA-AES 工作模式 1839

27.6.1 密钥、明文、密文 1840
27.6.2 字节序 1840
27.6.3 标准增量函数 1841
27.6.4 块个数 1841
27.6.5 初始向量 1841
27.6.6 块工作流程 1841
27.6.7 GCM 工作流程 1842
27.7 GCM 算法 1843
27.7.1 哈希子密钥 1844
27.7.2 J_0 1844
27.7.3 认证标签 1844
27.7.4 AAD 块个数 1844
27.7.5 不完整块的有效位数 1844
27.8 伪轮次抗 DPA 功能 1845
27.9 中断 1845
27.10 存储器列表 1846
27.11 寄存器列表 1847
27.12 寄存器 1848

28 ECC 加速器 (ECC) 1855
28.1 概述 1855
28.2 主要特性 1855
28.3 ECC 背景知识 1855
28.3.1 椭圆曲线与曲线上的点 1855
28.3.2 仿射坐标系与 Jacobian 坐标系 1855
28.3.3 内存块 1856
28.3.4 数据与数据块 1856
28.3.5 数据存储 1857
28.3.6 数据读取 1857
28.3.7 标准运算与 Jacobian 运算 1857
28.4 功能描述 1857
28.4.1 曲线模式 1857
28.4.2 工作模式 1858
28.4.2.1 标准点乘模式 1858
28.4.2.2 标准点验证模式 1859
28.4.2.3 标准点验证 + 标准点乘模式 1859
28.4.2.4 Jacobian 点乘模式 1859
28.4.2.5 点加模式 1859
28.4.2.6 Jacobian 点验证模式 1860
28.4.2.7 标准点验证 + Jacobian 点乘模式 1860
28.4.2.8 模加模式 1860
28.4.2.9 模减模式 1861
28.4.2.10 模乘模式 1861
28.4.2.11 模除模式 1861
28.4.3 提升抗攻击性能 1862
28.5 时钟与复位 1862

28.6 中断 1862
28.7 配置流程 1863
28.8 寄存器列表 1864
28.9 寄存器 1865

29 HMAC 加速器 (HMAC) 1869
29.1 主要特性 1869
29.2 功能描述 1869
29.2.1 上行模式 1869
29.2.2 下行模式 - JTAG 启动功能 1870
29.2.3 下行模式 - RSA_DS 密钥派生功能 1870
29.2.4 HMAC eFuse 配置 1870
29.2.5 调用 HMAC 流程（详细说明） 1872
29.2.5.1 启动 HMAC 模块 1872
29.2.5.2 配置 HMAC 密钥和密钥功能 1872
29.2.5.3 下行模式流程 1872
29.2.5.4 上行模式流程 1872
29.3 HMAC 算法细节 1873
29.3.1 附加填充位 1873
29.3.2 HMAC 算法结构 1874
29.4 寄存器列表 1876
29.5 寄存器 1878

30 RSA 加速器 (RSA) 1885
30.1 概述 1885
30.2 主要特性 1885
30.3 功能描述 1885
30.3.1 定义与表达 1886
30.3.2 大数模幂运算 1886
30.3.3 大数模乘运算 1887
30.3.4 大数乘法运算 1887
30.3.5 控制加速 1888
30.4 中断 1889
30.5 存储器列表 1891
30.6 寄存器列表 1891
30.7 寄存器 1892

31 SHA 加速器 (SHA) 1896
31.1 概述 1896
31.2 主要特性 1896
31.3 工作模式简介 1896
31.4 功能描述 1897
31.4.1 信息预处理 1897
31.4.1.1 附加填充比特 1897
31.4.1.2 信息解析 1898
31.4.1.3 哈希初始值 (Initial Hash Value) 1898

31.4.2 哈希运算流程 1899
31.4.2.1 Typical SHA 模式下的运算流程 1899
31.4.2.2 DMA-SHA 模式下的运算流程 1901
31.4.3 信息摘要存储 1903
31.5 中断 1903
31.6 寄存器列表 1905
31.7 寄存器 1907

32 RSA 数字签名外设 (RSA_DS) 1911
32.1 概述 1911
32.2 主要特性 1911
32.3 功能描述 1911
32.3.1 概述 1911
32.3.2 私钥运算子 1912
32.3.3 软件准备工作 1912
32.3.4 RSA_DS 硬件工作流程 1913
32.3.5 RSA_DS 软件工作流程 1914
32.4 存储器列表 1916
32.5 寄存器列表 1917
32.6 寄存器 1918

33 ECDSA 数字签名外设 (ECDSA_DS) 1921
33.1 概述 1921
33.2 主要特性 1921
33.3 ECDSA 背景知识 1921
33.3.1 域参数 1921
33.3.2 密钥生成 1922
33.3.3 签名生成 1922
33.3.4 签名验证 1922
33.4 功能描述 1923
33.4.1 ECDSA_DS 工作模式 1923
33.4.2 数据和数据块 1924
33.4.2.1 数据写入 1924
33.4.2.2 数据读取 1924
33.4.2.3 消息填充 1925
33.4.2.4 解析消息 1925
33.4.3 安全功能 1925
33.4.3.1 抗攻击性能 1925
33.4.3.2 基于状态的寄存器访问控制 1925
33.4.3.3 硬件占用 1926
33.5 配置流程 1926
33.5.1 ECDSA_DS 流程 1926
33.5.1.1 IDLE 阶段 1927
33.5.1.2 PREP 阶段 1927
33.5.1.3 LOAD 阶段 1928
33.5.1.4 PROC 阶段 1928

33.5.1.5 POST 阶段 1928
33.5.1.6 ECDSA_DS SHA 接口 1928
33.5.2 时钟和复位 1929
33.6 中断 1929
33.7 存储器块 1930
33.8 寄存器列表 1931
33.9 寄存器 1932

34 片外存储器加密与解密 (XTS_AES) 1938
34.1 概述 1938
34.2 主要特性 1938
34.3 模块结构 1938
34.4 功能描述 1939
34.4.1 XTS 算法 1939
34.4.2 密钥 1940
34.4.3 目标空间 1940
34.4.4 数据写入 1941
34.4.5 手动加密模块 1941
34.4.6 自动加密模块 1942
34.4.7 自动解密模块 1942
34.5 软件流程 1943
34.6 抗 DPA 攻击 1943
34.7 寄存器列表 1945
34.8 寄存器 1946

35 随机数发生器 (RNG) 1951
35.1 概述 1951
35.2 主要特性 1951
35.3 功能描述 1951
35.4 编程指南 1952
35.5 寄存器列表 1953
35.6 寄存器 1954

36 密钥管理器 1955
36.1 概述 1955
36.2 密钥管理背景知识 1955
36.3 术语 1955
36.4 主要特性 1957
36.5 结构概览 1957
36.6 功能描述 1958
36.6.1 HUK 生成器 1958
36.6.1.1 HUK 生成过程 1958
36.6.1.2 HUK 状态检查 1959
36.6.2 密钥管理器 1960
36.6.2.1 随机部署模式 1960
36.6.2.2 AES 部署方式 1960

36.6.2.3 ECDH0 部署模式 1962
36.6.2.4 ECDH1 部署模式 1963
36.6.2.5 私钥恢复模式 1965
36.6.2.6 key_info 导出模式 1965
36.7 编程流程 1965
36.7.1 HUK 生成器 1965
36.7.1.1 IDLE 阶段 1966
36.7.1.2 PREP 阶段 1966
36.7.1.3 LOAD 阶段 1966
36.7.1.4 PROC 阶段 1966
36.7.1.5 GAIN 阶段 1966
36.7.1.6 POST 阶段 1967
36.7.2 密钥管理器 1967
36.7.2.1 IDLE 阶段 1967
36.7.2.2 PREP 阶段 1970
36.7.2.3 LOAD 阶段 1970
36.7.2.4 PROC 阶段 1971
36.7.2.5 GAIN 阶段 1971
36.7.2.6 POST 阶段 1971
36.8 编程示例 1972
36.8.1 芯片私钥部署阶段的软件流程示例 1972
36.8.2 芯片正常启动阶段的软件流程示例 1972
36.9 中断 1973
36.10 内存块 1973
36.11 寄存器列表 1974
36.11.1 HUK 寄存器列表 1974
36.11.2 密钥管理器寄存器列表 1974
36.12 寄存器 1976
36.12.1 HUK 寄存器 1976
36.12.2 密钥管理器寄存器 1980

VI 图像和语音处理 1989

37 JPEG 图像编解码器 1990
37.1 术语 1990
37.2 概述 1991
37.3 主要特性 1992
37.4 架构概述 1993
37.5 功能描述 1994
37.5.1 JPEG 编码器 1994
37.5.1.1 暂停 1994
37.5.1.2 颜色空间转换 1994
37.5.1.3 可配置量化系数表 1995
37.5.1.4 填充零字节 1997
37.5.1.5 EOI 标记 1997
37.5.2 JPEG 解码器 1997

37.5.2.1 多个色度分量 1997
37.5.2.2 解析 RST 标记 1998
37.5.2.3 可配置量化系数表 1998
37.5.2.4 可配置哈夫曼表 1999
37.5.2.5 超时检测 2000
37.6 中断 2000
37.7 编程流程 2002
37.7.1 JPEG 编码器 2002
37.7.2 JPEG 解码器 2004
37.7.3 复位 2008
37.8 寄存器列表 2009
37.9 寄存器 2010

38 图像信号处理器 (ISP) 2031
38.1 概述 2031
38.2 术语 2031
38.3 主要特性 2032
38.4 系统架构 2033
38.5 功能描述 2033
38.5.1 ISP_Header 2033
38.5.2 ISP_Pipeline 2034
38.5.2.1 黑电平矫正 (BLC) 2034
38.5.2.2 坏点矫正 (DPC) 2034
38.5.2.3 拜耳域降噪 (BF) 2035
38.5.2.4 镜头阴影矫正 (LSC) 2035
38.5.2.5 去马赛克 2036
38.5.2.6 白平衡增益 (WBG) 2036
38.5.2.7 颜色矫正矩阵 (CCM) 2036
38.5.2.8 gamma 矫正 2036
38.5.2.9 RGB 转 YUV (RGB2YUV) 2037
38.5.2.10 锐化 (sharpen) 2037
38.5.2.11 对比度 / 色相 / 饱和度 / 亮度 (COLOR) 2037
38.5.2.12 YUV 范围调整及 YUV 转 RGB (YUV2RGB) 2038
38.5.2.13 裁剪 (CROP) 2038
38.5.2.14 自动曝光统计 (AE) 2038
38.5.2.15 自动对焦统计 (AF) 2038
38.5.2.16 自动白平衡统计 (AWB) 2039
38.5.2.17 直方图统计 (HIST) 2040
38.5.3 ISP_Tail 2040
38.5.4 时序控制 2040
38.5.5 影子寄存器 2040
38.5.6 CSI_CM 2041
38.5.7 CSI_Bridge 2041
38.5.8 像素格式及字节排序 2041
38.5.8.1 输出像素排列格式 2042
38.5.8.2 字节排序 2042

38.6 中断 2043
38.7 配置流程 2045
38.7.1 ISP 时钟复位配置 2045
38.7.2 CSI_Bridge 配置 2045
38.7.3 VDMA 配置 2045
38.7.4 ISP 通用配置 2046
38.7.5 从 MIPI-CSI 获取图像时打开 ISP 2046
38.7.6 从 DVP 获取图像时打开 ISP 2046
38.7.7 从 VDMA 获取图像时打开 ISP 2047
38.7.8 LSC LUT 配置 2047
38.7.9 AE 配置 2047
38.7.10 AF 配置 2048
38.7.11 AWB 配置 2049
38.7.12 HIST 配置 2050
38.7.13 更新 ISP_Pipeline 模块配置 2051
38.8 寄存器列表 2052
38.8.1 ISP 寄存器列表 2052
38.8.2 MIPI CSI_Bridge 寄存器列表 2055
38.9 寄存器 2057
38.9.1 ISP 寄存器 2057
38.9.2 MIPI CSI_Bridge 寄存器 2125

39 像素处理加速器 (PPA) 2136
39.1 概述 2136
39.2 术语 2136
39.3 主要特性 2136
39.4 结构概览 2137
39.5 功能描述 2138
39.5.1 PPA 颜色空间 2138
39.5.1.1 SRM 颜色空间 2139
39.5.1.2 BLEND 颜色空间 2141
39.5.2 PPA 2D-DMA 链表配置 2141
39.5.2.1 SRM 特殊配置 2143
39.5.3 旋转 - 缩放 - 镜像 (SRM) 2143
39.5.3.1 基本功能 2143
39.5.3.2 像素块重排 2144
39.5.4 图层叠加 (BLEND) 2145
39.5.4.1 基本功能 2145
39.5.4.2 填充图像输出 2146
39.6 中断 2146
39.7 配置流程 2147
39.7.1 PPA 时钟复位配置流程 2147
39.7.2 SRM 配置流程 2147
39.7.3 BLEND CLUT 配置流程 2148
39.7.4 BLEND 配置流程 2148
39.7.5 BLEND 图像填充配置流程 2149

39.7.6 错误处理流程 2149
39.8 寄存器列表 2150
39.9 寄存器 2152

40 LCD 与 Camera 控制器 (LCD_CAM) 2175
40.1 概述 2175
40.2 特性 2175
40.3 功能描述 2175
40.3.1 功能框图 2176
40.3.2 信号描述 2176
40.3.3 LCD_CAM 模块时钟 2177
40.3.3.1 LCD 时钟 2177
40.3.3.2 Camera 时钟 2178
40.3.4 LCD_CAM 模块复位 2179
40.3.5 LCD_CAM 数据格式控制 2180
40.3.5.1 LCD 数据格式控制 2180
40.3.5.2 Camera 数据格式控制 2182
40.3.6 YUV-RGB 数据格式转换 2183
40.3.6.1 YUV 数据流格式 2183
40.3.6.2 格式转换模块配置流程 2184
40.3.7 LCD_CAM 时序 2185
40.3.7.1 LCD 时序 (RGB 格式) 2185
40.3.7.2 LCD 时序（I8080/MOTO6800 格式） 2187
40.4 中断 2187
40.5 软件配置流程 2187
40.5.1 软件配置 LCD（RGB 格式）发送流程 2188
40.5.2 软件配置 LCD（I8080/MOTO6800 格式）发送流程 2188
40.5.3 软件配置 Camera 接收流程 2189
40.6 寄存器列表 2191
40.7 寄存器 2192

41 H264 编码器 2210
41.1 概述 2210
41.2 术语 2210
41.3 特性 2210
41.4 架构 2211
41.5 功能描述 2212
41.5.1 编码器算法核 (ENC_CORE) 2213
41.5.1.1 架构 2213
41.5.1.2 原始视频色彩空间转换 2214
41.5.1.3 量化结果降采样 2214
41.5.1.4 运动矢量 (MV) 合并 2215
41.5.1.5 宏块级码率控制 2216
41.5.1.6 感兴趣区域 (ROI) 2217
41.5.1.7 调试用数据替换 2217
41.5.2 H264 专用 DMA 2218

41.5.2.1 架构 2218
41.5.2.2 链表描述符 2220
41.5.2.3 传输启动 2221
41.5.2.4 传输复位 2222
41.5.2.5 调试用数据替换 2222
41.5.2.6 通道配置 2223
41.6 中断 2224
41.7 配置流程 2226
41.7.1 GOP 模式 2226
41.7.2 双流模式 2229
41.7.3 软重置 2232
41.8 寄存器列表 2233
41.8.1 H264 编码器寄存器列表 2233
41.8.2 H264 DMA 寄存器列表 2235
41.9 寄存器 2243
41.9.1 H264 编码器寄存器 2243
41.9.2 H264 DMA 寄存器 2274

42 MIPI CSI 2444
42.1 概述 2444
42.2 术语 2444
42.3 主要特性 2445
42.4 结构概览 2445
42.5 功能描述 2446
42.5.1 通道状态 2446
42.5.2 MIPI RX D-PHY 操作 2447
42.5.2.1 断电状态 2448
42.5.2.2 关断状态 2448
42.5.2.3 AFE 初始化 2450
42.5.2.4 控制模式 2450
42.5.2.5 高速数据接收模式 2450
42.5.2.6 Escape 模式 2450
42.5.3 MIPI RX D-PHY 配置 2451
42.5.3.1 MIPI RX D-PHY 编码接口 2452
42.5.3.2 MIPI RX D-PHY Test Code 2453
42.5.4 解扰器 2454
42.5.5 错误管理 2454
42.6 中断 2455
42.7 配置流程 2457
42.7.1 开始高速数据接收 2457
42.7.1.1 时钟与复位 2457
42.7.1.2 MIPI RX D-PHY 和 MIPI 主机初始化 2458
42.7.2 停止高速数据接收 2458
42.8 寄存器列表 2459
42.9 寄存器 2461

43 MIPI DSI 2482

43.1 介绍 2482

43.2 功能列表 2482

43.3 架构概述 2483

43.4 功能描述 2484

43.4.1 DSI 桥接器 2484

43.4.2 DSI 主机 2485

43.4.2.1 视频传输 2485

43.4.2.2 命令传输 2486

43.4.2.2.1 LP 模式下的命令传输 2487

43.4.2.2.2HS 模式下的命令传输 2488

43.4.2.3 视频图案生成器 2489

43.4.2.3.1 色带图案 2489

43.4.2.3.2BER 测试图案 2490

43.4.2.4 错误报告 2490

43.4.3 D-PHY 2491

43.4.3.1 D-PHY 操作模式 2491

43.4.3.1.1 无电源状态 (No-Power) 2491

43.4.3.1.2 关闭状态 (Shutdown) 2491

43.4.3.1.3 PLL 锁定和 AFE 初始化 2493

43.4.3.1.4 控制模式 (Control) 2493

43.4.3.1.5 高速数据传输 2493

43.4.3.1.6 逃逸模式 (Escape) 2494

43.4.3.1.7 转向 2494

43.4.3.2 PLL 参数 2494

43.4.3.3 测试代码 2495

43.4.3.3.1 正常运行 2495

43.4.3.3.2PLL 偏置电流选择器 / 滤波电容控制 / VCO 控制 2496

43.4.3.3.3用于初始化和超低功耗模式的 PLL CP 控制/PLL 锁定旁路 2496

43.4.3.3.4PLL LPF 和 CP 控制 2497

43.4.3.3.5PLL 输入分频比 2498

43.4.3.3.6PLL 循环分频比 2499

43.4.3.3.7 PLL 输入和循环分频比控制 2499

43.4.3.3.8AFE/BIAS/Bandgap 模拟可编程性 2500

43.4.3.3.9HS RX 控制通道 0 2502

43.4.4 时钟和显示分辨率 2502

43.5 中断 2505

43.6 配置过程 2507

43.6.1 DSI 桥接配置 2507

43.6.2 D-PHY 配置 2507

43.6.3 MIPI DSI 通用配置 2508

43.6.4 系统内存数据输出流配置 2508

43.6.5 视频模式输出流配置 2508

43.6.6 使用 DCS 命令操作显示寄存器 2508

43.7 寄存器列表 2509

43.7.1 MIPI DSI 桥接寄存器列表 2509

43.7.2 MIPI DSI 主机寄存器列表 2510
43.8 寄存器 2512
43.8.1 MIPI DSI 桥接寄存器 2512
43.8.2 MIPI DSI 主机寄存器 2531

44 语音活动检测 (VAD) 2567
44.1 概述 2567
44.2 主要特性 2567
44.3 系统架构 2567
44.4 功能描述 2568
44.4.1 算法参数配置 2568
44.4.2 唤醒源配置 2570
44.5 中断 2570
44.6 配置流程 2570
44.6.1 自动运算模式 2570
44.6.2 手动运算模式 2570
44.7 寄存器列表 2571
44.8 寄存器 2572

VII 通讯接口 2575

45 UART 控制器 (UART) 2576
45.1 概述 2576
45.2 主要特性 2576
45.3 UART 架构 2578
45.4 功能描述 2579
45.4.1 时钟与复位 2579
45.4.2 UART FIFO 2579
45.4.3 波特率产生与检测 2579
45.4.3.1 波特率产生 2579
45.4.3.2 波特率检测 2580
45.4.4 UART 数据帧 2581
45.4.5 AT_CMD 字符格式 2582
45.4.6 RS485 2582
45.4.6.1 驱动控制 2582
45.4.6.2 转换延时 2583
45.4.6.3 总线侦听 2583
45.4.7 IrDA 2583
45.4.8 唤醒 2584
45.4.9 流控 2585
45.4.9.1 硬件流控 2585
45.4.9.2 软件流控 2586
45.4.10 GDMA 模式 2587
45.5 中断 2587
45.6 编程流程 2589
45.6.1 寄存器类型 2589

45.6.2 具体步骤 2590
45.6.2.1 UART*n* 模块初始化 2590
45.6.2.2 UART*n* 通信配置 2590
45.6.2.3 启动 UART*n* 2591
45.7 寄存器列表 2592
45.7.1 UART 寄存器列表 2592
45.7.2 LP UART 寄存器列表 2593
45.7.3 UHCI 寄存器列表 2594
45.8 寄存器 2596
45.8.1 UART 寄存器 2596
45.8.2 LP UART 寄存器 2618
45.8.3 UHCI 寄存器 2636

46 SPI 控制器 (SPI) 2655
46.1 概述 2655
46.2 术语 2655
46.3 特性 2657
46.4 架构概览 2659
46.5 功能描述 2659
46.5.1 数据模式 2660
46.5.2 总线信号描述 2660
46.5.3 数据位读/写顺序控制 2663
46.5.4 非字节对齐传输 2665
46.5.5 传输类型 2665
46.5.6 CPU 控制的数据传输 2665
46.5.6.1 CPU 控制的主机传输 2666
46.5.6.2 CPU 控制的从机传输 2668
46.5.7 DMA 控制的数据传输 2668
46.5.7.1 DMA 配置 2669
46.5.7.2 DMA TX/RX Buffer 长度控制 2670
46.5.8 主机和从机数据流控制（以 GP-SPI 为例） 2670
46.5.8.1 GP-SPI 功能块图 2671
46.5.8.2 GP-SPI 用作主机时的数据流控制 2672
46.5.8.3 GP-SPI 用作从机时的数据流控制 2672
46.5.9 用作主机（以 GP-SPI 为例） 2673
46.5.9.1 主机状态机 2673
46.5.9.2 状态控制和位模式控制寄存器 2676
46.5.9.3 主机全双工通信（仅支持 1-bit 模式） 2680
46.5.9.4 主机半双工通信（支持 1/2/4/(8)-bit 模式） 2681
46.5.9.5 DMA 控制的分段配置传输 2683
46.5.10 用作从机（以 GP-SPI 为例） 2686
46.5.10.1 可配置的通信格式 2686
46.5.10.2 半双工通信支持的 CMD 值 2687
46.5.10.3 从机单次传输和从机连读传输 2690
46.5.10.4 配置从机单次传输 2690
46.5.10.5 配置半双工通信下从机连续传输 2691

46.5.10.6 配置全双工通信下从机连续传输 2691
46.6 CS 建立时间和保持时间控制 2692
46.7 时钟控制 2693
46.7.1 GP-SPI 时钟控制 2693
46.7.2 LP-SPI 时钟控制 2694
46.7.3 时钟相位和极性 2694
46.7.4 主机时钟控制 2695
46.7.5 从机时钟控制 2696
46.8 GP-SPI 时序补偿 2696
46.9 LP-SPI 唤醒 2698
46.10 GP-SPI2 与 GP-SPI3 和 LP-SPI 之间的差异 2698
46.10.1 功能差异 2698
46.10.2 寄存器差异 2699
46.10.3 中断差异 2702
46.11 中断 2703
46.11.1 中断描述 2703
46.11.2 主机中断和从机中断（以 GP-SPI 为例） 2704
46.12 寄存器列表 2706
46.12.1 GP-SPI2 寄存器列表 2706
46.12.2 GP-SPI3 寄存器列表 2707
46.12.3 LP-SPI 寄存器列表 2708
46.13 寄存器 2710
46.13.1 GP-SPI2 寄存器 2710
46.13.2 GP-SPI3 寄存器 2751
46.13.3 LP-SPI 寄存器 2783

47 I2C 控制器 (I2C) 2809
47.1 概述 2809
47.2 主要特性 2809
47.3 I2C 架构 2810
47.4 功能描述 2812
47.4.1 时钟配置 2812
47.4.2 滤除 SCL 和 SDA 噪声 2813
47.4.3 SCL 时钟拉伸 2813
47.4.4 SCL 空闲时产生 SCL 脉冲 2813
47.4.5 同步 2814
47.4.6 漏级开路输出 2815
47.4.7 时序参数配置 2815
47.4.8 超时控制 2817
47.4.9 指令配置 2817
47.4.10 TX/RX RAM 数据存储 2818
47.4.11 数据转换 2819
47.4.12 寻址模式 2819
47.4.13 10 位寻址的读写标志位检查 2819
47.4.14 启动控制器 2820
47.5 LP_I2C 和 I2C 的功能差异 2820

47.6 编程示例 2820
47.6.1 I2C 主机写入从机，7 位寻址，单次命令序列 2821
47.6.1.1 场景介绍 2821
47.6.1.2 配置示例 2821
47.6.2 I2C 主机写入从机，10 位寻址，单次命令序列 2822
47.6.2.1 场景介绍 2822
47.6.2.2 配置示例 2823
47.6.3 I2C 主机写入从机，7 位双地址寻址，单次命令序列 2824
47.6.3.1 场景介绍 2824
47.6.3.2 配置示例 2824
47.6.4 I2C 主机写入从机，7 位寻址，多次命令序列 2825
47.6.4.1 场景介绍 2825
47.6.4.2 配置示例 2826
47.6.5 I2C 主机读取从机，7 位寻址，单次命令序列 2827
47.6.5.1 场景介绍 2827
47.6.5.2 配置示例 2828
47.6.6 I2C 主机读取从机，10 位寻址，单次命令序列 2829
47.6.6.1 场景介绍 2829
47.6.6.2 配置示例 2829
47.6.7 I2C 主机读取从机，7 位双寻址，单次命令序列 2831
47.6.7.1 场景介绍 2831
47.6.7.2 配置示例 2831
47.6.8 I2C 主机读取从机，7 位寻址，多次命令序列 2832
47.6.8.1 场景介绍 2833
47.6.8.2 配置示例 2834
47.7 中断 2835
47.8 寄存器列表 2837
47.8.1 I2C 寄存器列表 2837
47.8.2 LP_I2C 寄存器列表 2838
47.9 寄存器 2840
47.9.1 I2C 寄存器 2840
47.9.2 LP_I2C 寄存器 2863

48 模拟 I2C 控制器 2882
48.1 概述 2882
48.2 主要特性 2882
48.3 结构概览 2882
48.4 功能描述 2883
48.5 配置流程 2884
48.6 寄存器列表 2885
48.7 寄存器 2886

49 I2S 控制器 (I2S) 2890
49.1 概述 2890
49.2 术语 2890
49.3 特性 2891

49.4 系统架构 2892
49.5 I2S 模块支持的音频协议 2894
49.5.1 TDM Philips 标准模式 2895
49.5.2 TDM MSB 对齐标准模式 2895
49.5.3 TDM PCM 标准模式 2896
49.5.4 PDM 标准模式 2896
49.6 I2S TX/RX 模块时钟 2897
49.7 I2S 模块复位 2899
49.8 I2S 主/从机模式 2899
49.8.1 主/从机发送模式 2899
49.8.2 主/从机接收模式 2900
49.9 发送数据 2900
49.9.1 数据格式控制 2900
49.9.1.1 通道有效数据位宽 2901
49.9.1.2 通道有效数据字节序 2901
49.9.1.3 A 律/μ 律压缩/解压缩 2901
49.9.1.4 通道发送数据位宽 2902
49.9.1.5 通道数据比特顺序 2902
49.9.2 通道模式控制 2903
49.9.2.1 TDM 输出模式 2903
49.9.2.2 PDM 输出模式 2904
49.9.3 同步计数器 2906
49.10 接收数据 2907
49.10.1 通道模式控制 2907
49.10.1.1 TDM 输入模式 2907
49.10.1.2 PDM 输入模式 2908
49.10.2 数据格式控制 2909
49.10.2.1 通道数据比特顺序 2909
49.10.2.2 通道储存数据位宽 2909
49.10.2.3 通道接收数据位宽 2909
49.10.2.4 通道储存数据字节序 2909
49.10.2.5 A 律/μ 律压缩/解压缩 2910
49.11 事件任务矩阵功能 2910
49.12 I2S 中断 2911
49.13 软件配置流程 2911
49.13.1 软件配置 I2S 发送流程 2911
49.13.2 软件配置 I2S 接收流程 2912
49.14 寄存器列表 2913
49.15 寄存器 2915

50 LP I2S 控制器 2934
50.1 概况 2934
50.2 主要特性 2934
50.3 架构概览 2935
50.4 支持的音频协议 2936
50.4.1 TDM Philips 标准模式 2936

50.4.2 TDM MSB 对齐标准模式 2937
50.4.3 TDM PCM 标准模式 2937
50.4.4 PDM 标准模式 2938
50.5 RX 时钟 2938
50.6 复位 2940
50.7 主/从机接收模式 2940
50.8 接收数据 2941
50.8.1 通道模式控制 2941
50.8.1.1 TDM 输入模式 2941
50.8.1.2 PDM 输入模式 2941
50.8.2 数据格式控制 2942
50.8.2.1 通道数据比特顺序 2942
50.8.2.2 通道储存数据位宽 2942
50.8.2.3 通道接收数据位宽 2942
50.8.2.4 通道储存数据字节序 2943
50.8.3 内部存储器 2943
50.9 中断 2943
50.10 寄存器列表 2945
50.11 寄存器 2946

51 脉冲计数控制器 (PCNT) 2959
51.1 概述 2959
51.2 主要特性 2959
51.3 系统架构 2960
51.4 功能描述 2960
51.5 中断 2962
51.6 编程指南 2963
51.6.1 通道 0 独自递增计数 2963
51.6.2 通道 0 独自递减计数 2963
51.6.3 通道 0 和通道 1 同时递增计数 2964
51.7 寄存器列表 2965
51.8 寄存器 2966

52 USB 2.0 高速 OTG 2974
52.1 概述 2974
52.2 术语 2974
52.3 特性 2974
52.3.1 通用特性 2975
52.3.2 设备模式特性 2975
52.3.3 主机模式特性 2975
52.4 功能描述 2975
52.4.1 控制器内核与接口 2976
52.4.2 存储器布局 2976
52.4.2.1 控制 & 状态寄存器 (CSR) 2977
52.4.2.2 FIFO 访问 2977
52.4.3 FIFO 和队列组织 2978

52.4.3.1 主机模式 FIFO 和队列 2978
52.4.3.2 设备模式 FIFO 2979
52.4.4 中断层次结构 2980
52.4.5 DMA 模式和 Slave 模式 2981
52.4.5.1 Slave 模式 2981
52.4.5.2 Buffer DMA 模式 2981
52.4.5.3 Scatter/Gather DMA 模式 2981
52.4.6 事务和传输级操作 2982
52.4.6.1 DMA 模式下的事务和传输级操作 2982
52.4.6.2 Slave 模式下的事务和传输级操作 2982
52.4.7 OTG 2983
52.5 寄存器 2983

53 USB 2.0 全速 OTG 2984

53.1 概述 2984
53.2 术语 2984
53.3 特性 2985
53.3.1 通用特性 2985
53.3.2 设备模式 (Device mode) 特性 2985
53.3.3 主机模式 (Host mode) 特性 2985
53.4 功能描述 2985
53.4.1 控制器内核与接口 2986
53.4.2 存储器布局 2987
53.4.2.1 控制 & 状态寄存器 (CSR) 2987
53.4.2.2 FIFO 访问 2988
53.4.3 FIFO 和队列组织 2988
53.4.3.1 主机模式 FIFO 和队列 2988
53.4.3.2 设备模式 FIFO 2989
53.4.4 中断层次结构 2990
53.4.5 Slave 模式和 DMA 模式 2991
53.4.5.1 Slave 模式 2992
53.4.5.2 Buffer DMA 模式 2992
53.4.5.3 Scatter/Gather DMA 模式 2992
53.4.6 事务和传输级操作 2993
53.4.6.1 DMA 模式下的事务和传输级操作 2993
53.4.6.2 Slave 模式下的事务和传输级操作 2993
53.5 OTG 2994
53.5.1 OTG 接口 2994
53.5.2 ID 管脚检测 2995
53.5.3 会话请求协议 (SRP) 2995
53.5.3.1 A 设备 SRP 2995
53.5.3.2 B 设备 SRP 2996
53.5.4 主机协商协议 (HNP) 2997
53.5.4.1 A 设备 HNP 2997
53.5.4.2 B 设备 HNP 2998
53.6 寄存器 2999

54 USB 串口/JTAG 控制器 (USB_SERIAL_JTAG) 3000
54.1 概述 3000
54.2 特性 3000
54.3 功能描述 3002
54.3.1 CDC-ACM USB 接口描述 3002
54.3.2 CDC-ACM 固件接口描述 3003
54.3.3 USB-JTAG 接口：JTAG 命令处理器 3004
54.3.4 USB-JTAG 接口：CMD_REP 使用示例 3005
54.3.5 USB-JTAG 接口：响应捕捉单元 3005
54.3.6 USB-JTAG 接口：控制传输请求 3005
54.4 操作建议 3006
54.5 中断 3007
54.6 寄存器列表 3009
54.7 寄存器 3010

55 以太网介质访问控制器 (EMAC) 3034
55.1 概述 3034
55.2 主要特性 3034
55.3 EMAC 架构 3036
55.4 功能描述 3036
55.4.1 EMAC_CORE 3036
55.4.1.1 传输操作 3036
55.4.1.2 接收操作 3037
55.4.2 EMAC_MTL（MAC 传输层） 3039
55.4.3 EMAC_DMA 3039
55.4.3.1 发送描述符 3039
55.4.3.2 接收描述符 3043
55.4.4 PHY 接口 3047
55.4.4.1 MII（介质独立接口） 3047
55.4.4.2 RMII（精简介质独立接口） 3049
55.4.4.3 Station Management Agent (SMA) 接口 3051
55.4.5 MAC 地址的过滤 3052
55.4.5.1 单播目标地址过滤 3052
55.4.5.2 多播目标地址过滤 3052
55.4.5.3 广播地址过滤 3052
55.4.5.4 单播源地址过滤 3052
55.4.5.5 反向过滤操作 3052
55.4.6 Energy Efficient Ethernet（EEE） 3053
55.4.6.1 传输方向 3054
55.4.6.2 接收方向 3054
55.4.7 源地址、VLAN 和 CRC 的控制 3054
55.4.7.1 源地址控制 3054
55.4.7.2 VLAN 控制 3054
55.4.7.3 CRC 控制 3054
55.4.8 时间戳 3054
55.4.8.1 传输通路 3055

55.4.8.2 接收通路 3055
55.4.9 远端唤醒 3055
55.4.10 好的传输帧与接收帧 3055
55.5 编程步骤 3056
55.5.1 MAC 系统层配置 3056
55.5.2 EMAC 初始配置 3057
55.5.3 启动传输 3057
55.5.4 启动接收 3058
55.5.5 发送端进退 LPI 状态 3058
55.5.6 接收端进退 LPI 状态 3058
55.6 中断 3058
55.7 寄存器列表 3060
55.8 寄存器 3062

56 双线汽车接口（TWAI） 3108
56.1 主要特性 3108
56.2 协议概述 3109
56.2.1 TWAI 特性 3109
56.2.2 TWAI 报文 3109
56.2.2.1 数据帧和远程帧 3110
56.2.2.2 错误帧和过载帧 3111
56.2.2.3 帧间距 3113
56.2.3 TWAI 错误 3113
56.2.3.1 错误类型 3113
56.2.3.2 错误状态 3114
56.2.3.3 错误状态跳转 3114
56.2.3.4 错误计数规则 3114
56.2.4 TWAI 位时序 3115
56.2.4.1 标称位 3115
56.2.4.2 硬同步与再同步 3116
56.3 结构概述 3117
56.3.1 位时序逻辑 3117
56.3.2 位流处理器 3117
56.3.3 接收过滤器 3118
56.3.4 接收 FIFO 3118
56.3.5 错误管理逻辑 3118
56.3.6 寄存器模块 3118
56.4 功能描述 3119
56.4.1 模式 3119
56.4.1.1 复位模式 3119
56.4.1.2 操作模式 3119
56.4.2 位时序 3119
56.4.3 发送缓冲器与接收缓冲器 3120
56.4.3.1 缓冲器概述 3120
56.4.3.2 帧信息 3121
56.4.3.3 帧标识符 3121

56.4.3.4 帧数据 3122
56.4.4 接收 FIFO 和数据溢出 3122
56.4.5 接收过滤器 3123
56.4.5.1 单过滤模式 3124
56.4.5.2 双过滤模式 3125
56.4.6 错误管理 3126
56.4.6.1 错误报警限制 3126
56.4.6.2 被动错误 3126
56.4.6.3 离线状态与离线恢复 3126
56.4.7 错误捕捉 3127
56.4.8 仲裁丢失捕捉 3128
56.4.9 收发器自动待机 3128
56.5 中断 3129
56.6 寄存器列表 3132
56.7 寄存器 3134

57 SD/MMC 主机控制器 (SDHOST) 3151
57.1 概述 3151
57.2 主要特性 3151
57.3 SD/MMC 外部接口信号 3152
57.4 功能描述 3152
57.4.1 SD/MMC 主机控制器结构 3152
57.4.1.1 总线接口单元 (BIU) 3153
57.4.1.2 卡接口单元 (CIU) 3153
57.4.2 命令通路 3154
57.4.3 数据通路 3154
57.4.3.1 数据发送 3154
57.4.3.2 数据接收 3155
57.5 CIU 操作的软件限制 3155
57.6 收发数据 RAM 3156
57.6.1 TX RAM 模块 3157
57.6.2 RX RAM 模块 3157
57.7 DMA 链表 3157
57.8 DMA 描述符结构 3157
57.9 操作流程 3159
57.9.1 寄存器初始化流程 3159
57.9.2 命令发送流程 3160
57.9.3 DMA 控制器初始化流程 3160
57.9.4 DMA 控制器数据发送流程 3161
57.9.5 DMA 控制器数据接收流程 3161
57.10 时钟相位选择 3162
57.11 中断 3162
57.12 寄存器列表 3164
57.13 寄存器 3166

58 LED PWM 控制器 (LEDC) 3194

58.1 概述 3194
58.2 特性 3194
58.3 架构概览 3194
58.4 功能描述 3195
58.4.1 定时器 3195
58.4.1.1 时钟源 3196
58.4.1.2 时钟分频器配置 3196
58.4.1.3 20 位计数器 3197
58.4.2 PWM 生成器 3198
58.4.3 占空比渐变 3199
58.4.3.1 线性占空比渐变 3199
58.4.3.2 伽马曲线渐变 3200
58.4.3.3 占空比渐变暂停和恢复 3201
58.4.4 事件任务矩阵功能 3201
58.5 中断 3203
58.6 配置流程 3203
58.7 存储空间 3205
58.8 寄存器列表 3206
58.9 寄存器 3208

59 电机控制脉宽调制器 (MCPWM) 3220
59.1 概述 3220
59.2 主要特性 3220
59.3 模块 3223
59.3.1 概述 3223
59.3.1.1 定时器模块 3223
59.3.1.2 操作器模块 3223
59.3.1.3 捕获模块 3224
59.3.1.4 ETM 模块 3224
59.3.2 PWM 定时器模块 3224
59.3.2.1 PWM 定时器模块的配置 3224
59.3.2.2 PWM 定时器工作模式和定时事件生成 3225
59.3.2.3 PWM 定时器影子寄存器 3229
59.3.2.4 PWM 定时器同步和锁相 3229
59.3.3 PWM 操作器模块 3229
59.3.3.1 PWM 生成器模块 3230
59.3.3.2 死区生成器模块 3241
59.3.3.3 PWM 载波模块 3245
59.3.3.4 故障检测模块 3248
59.3.4 捕获模块 3249
59.3.4.1 介绍 3249
59.3.4.2 捕获定时器 3249
59.3.4.3 捕获通道 3250
59.3.5 ETM 模块 3250
59.3.5.1 介绍 3250
59.3.5.2 MCPWM 可产生的 ETM 事件 3250

59.3.5.3 MCPWM 可接收的 ETM 任务 3251
59.4 中断 3251
59.5 寄存器列表 3253
59.6 寄存器 3256

60 红外遥控 (RMT) 3302
60.1 概述 3302
60.2 主要特性 3302
60.3 功能描述 3303
60.3.1 RMT 架构 3303
60.3.2 RAM 3304
60.3.2.1 RAM 结构 3304
60.3.2.2 RAM 使用说明 3304
60.3.2.3 RAM 访问方式 3305
60.3.3 时钟 3306
60.3.4 发射器 3306
60.3.4.1 普通发送模式 3306
60.3.4.2 乒乓发送模式 3306
60.3.4.3 发送加载波 3307
60.3.4.4 持续发送模式 3307
60.3.4.5 多通道同时发送 3307
60.3.5 接收器 3307
60.3.5.1 普通接收模式 3307
60.3.5.2 乒乓接收模式 3308
60.3.5.3 接收滤波 3308
60.3.5.4 接收去载波 3308
60.3.6 配置参数更新 3308
60.4 中断 3309
60.5 寄存器列表 3311
60.6 寄存器 3313

61 并行 IO 控制器 (PARLIO) 3327
61.1 概况 3327
61.2 术语 3327
61.3 特性 3327
61.4 系统架构 3328
61.5 功能描述 3329
61.5.1 时钟生成器 3329
61.5.2 时钟复位使用限制 3330
61.5.3 主从机模式 3331
61.5.4 RX 模块接收模式 3331
61.5.4.1 电平使能模式 3332
61.5.4.2 脉冲使能模式 3332
61.5.4.3 软件使能模式 3333
61.5.5 RX 模块 GDMA SUC EOF 信号生成 3333
61.5.6 TX 模块 EOF 信号生成 3334

61.5.7 RX 模块超时 3334
61.5.8 TX 模块无限制长度数据传输 3334
61.5.9 TX 片选功能 3334
61.5.10 TX 模块输出时钟门控 3334
61.5.11 TX 模块总线空闲值 3335
61.5.12 单帧数据传输 3335
61.5.13 字节范围内数据顺序翻转 3335
61.6 中断 3335
61.7 配置流程 3336
61.7.1 数据接收操作流程 3336
61.7.2 数据发送操作流程 3336
61.8 应用示例 3337
61.8.1 与 SPI 进行数据传输 3337
61.8.2 与 I2S 进行数据传输 3339
61.8.3 与 LCD 进行数据传输 3339
61.9 寄存器列表 3341
61.10 寄存器 3342

62 比特调节器 3353
62.1 概述 3353
62.2 特性 3353
62.3 架构 3354
62.3.1 数据路径 3354
62.3.2 控制路径 3355
62.4 功能描述 3356
62.4.1 指令 3356
62.4.2 配置寄存器 3362
62.5 配置流程 3364
62.6 寄存器列表 3365
62.7 寄存器 3366

VIII 模拟信号处理 3378

63 触摸传感器 (TOUCH) 3379
63.1 术语 3379
63.2 特性 3379
63.3 系统架构 3380
63.3.1 触摸板 3380
63.3.2 电容触摸管脚 3380
63.3.3 触摸传感器 3381
63.4 功能描述 3382
63.4.1 Touch FSM 3382
63.4.2 采样信号预处理 3383
63.4.2.1 测量过程 3384
63.4.2.2 测量操作的触发源 3385
63.4.2.3 扫描模式 3386

63.4.2.4 跳频模式 3386
63.4.3 触摸检测 3387
63.4.3.1 采样值 3388
63.4.3.2 硬件触摸检测 3389
63.4.4 接近模式 3389
63.4.5 休眠模式 3390
63.4.6 防潮功能 3390
63.4.7 遇水保护功能 3391
63.5 中断 3391
63.6 寄存器列表 3392
63.6.1 中断与状态寄存器列表 3392
63.6.2 配置寄存器列表 3393
63.7 寄存器 3396
63.7.1 中断与状态寄存器 3396
63.7.2 配置寄存器 3403

64 温度传感器 (TSENS) 3418
64.1 介绍 3418
64.2 特性 3418
64.3 结构概览 3418
64.4 功能描述 3419
64.4.1 温度传感器上电 3419
64.4.2 时钟管理 3419
64.4.3 自动监测唤醒模式 3419
64.4.4 温度测量范围与温度偏移 3420
64.4.5 数据转换 3420
64.5 事件任务矩阵功能 3420
64.6 中断 3421
64.7 配置流程 3421
64.8 寄存器列表 3422
64.9 寄存器 3423

65 ADC 控制器 (ADC) 3428
65.1 概述 3428
65.2 术语表 3429
65.3 主要特性 3429
65.4 结构概览 3430
65.5 功能描述 3431
65.5.1 SAR ADC 上电 3431
65.5.2 SAR ADC 通道 3431
65.5.3 SAR ADC 时钟 3431
65.5.4 SAR ADC 转换和衰减 3432
65.5.5 HP ADC FSM 3432
65.5.6 HP ADC 样式表 3433
65.5.7 HP ADC 多通道采样配置示例 3435
65.5.8 双 HP ADC 采样控制 3435

65.5.9 HP ADC 滤波器 3436
65.5.10 HP ADC 阈值监控器 3436
65.5.11 HP ADC 的 GDMA 支持 3436
65.5.12 LP ADC 自动监测唤醒模式 3437
65.6 事件任务矩阵功能 3437
65.7 中断 3437
65.8 配置流程 3438
65.8.1 HP ADC 多通道采样配置 3438
65.8.2 LP ADC 单次采样配置 3438
65.8.3 LP ADC 自动监测唤醒配置 3439
65.9 寄存器列表 3440
65.9.1 HP ADC 寄存器列表 3440
65.9.2 LP ADC 寄存器列表 3440
65.10 寄存器 3442
65.10.1 HP ADC 寄存器 3442
65.10.2 LP ADC 寄存器 3453

66 模拟电压比较器 3468
66.1 概述 3468
66.2 主要特性 3468
66.3 系统架构 3468
66.4 功能描述 3469
66.5 事件任务矩阵 3470
66.6 中断 3470
66.7 配置流程 3471

IX 附录 3472

相关文档和资源 3473

词汇列表 3474
外设相关词汇 3474
寄存器相关缩写 3475
寄存器的访问类型 3476

如何配置寄存器的保留域 3478
概述 3478
如何配置保留域 3478

中断配置寄存器 3479

修订历史 3480

表格

2.4-1 CPU 地址分布 70
2.7-2 HWLP 指令 124
2.8-1 存储器映射寄存器访问的地址解码 127
2.8-2 内部存储器地址映射 127
2.9-1 CLIC 中断映射 129
2.9-2 CLIC 级别和优先级编码 (CLICINTCTLBITS=3) 129
2.9-4 核心本地中断源 134
2.11-1 JTAG 控制 150
2.11-3 进入调试模式时 DPC 中的虚拟地址 153
2.11-5 NAPOT 编码的 maddress 157

3.3-1 追踪编码器参数 177
3.5-1 可选边带信号 179
3.6-1 头部格式 180
3.6-2 索引格式 181
3.6-3 格式 3 子格式 0 181
3.6-4 格式 3 子格式 1 182
3.6-5 格式 3 子格式 3 182
3.6-6 格式 2 183
3.6-7 格式 1，有地址 184
3.6-8 格式 1，无地址 185
3.7-1 CORE*n*_TRACE's 的内部中断源 185
3.8-1 调试模块的触发支持（mcontrol 寄存器的 action 字段） 186
3.8-2 追踪状态 188

4.4-1 ESP32-P4 扩展指令集寄存器集合 204
4.4-2 数据格式及对齐 207
4.4-3 舍入模式配置 208
4.4-4 舍入操作结果 209
4.5-1 指令简介 210
4.5-2 读内存指令 213
4.5-3 写内存指令 213
4.5-4 数据交换指令 214
4.5-5 向量加法运算指令 215
4.5-6 运算指令 215
4.5-7 运算指令 216
4.5-8 运算指令 216
4.5-9 向量与标量乘累加指令 217
4.5-10 其他指令 218
4.5-11 比较指令 218
4.5-12 按位逻辑操作指令 219
4.5-13 移位指令 219
4.5-14 蝶形运算指令 220
4.5-15 比特反转指令 220

4.5-16 实数 FFT 指令 220
4.6-1 ESP32-P4 处理器五级流水线 221
4.7-1 指令域名及含义 223

5.4-1 LP CPU 中断映射 626
5.4-2 LP CPU 异常原因 627
5.7-1 性能计数器 634
5.10-1 唤醒源 637

6.4-1 GDMA-AHB 配置寄存器与外设选择关系表 644
6.4-2 GDMA-AXI 配置寄存器与外设选择关系表 644

7.4-1 握手接口与外设映射表 752
7.5-1 多块传输寄存器更新方法 757

8.4-1 描述符字段说明 802
8.4-2 2D-DMA TX 方向配置寄存器与外设选择关系表 806
8.4-3 2D-DMA RX 方向配置寄存器与外设选择关系表 806
8.4-4 2D-DMA 发送方向宏块重排推荐配置表 807
8.4-5 2D-DMA 接收方向宏块重排推荐配置表（不做颜色格式转换） 807
8.4-6 TX 方向颜色格式转换参数配置 809
8.4-7 RX 方向颜色格式转换参数配置 809
8.4-8 RGBB888 的存储格式 810
8.4-9 YUV444 的存储格式 810
8.4-10 YUV422 的存储格式 810

9.3-1 地址映射 872
9.3-2 模块/外设地址空间映射表 880
9.4-1 eFuse BLOCK0 中的参数 886
9.4-2 密钥用途数值对应的含义 891
9.4-3 BLOCK1~BLOCK10 中的参数 892
9.4-4 用户读取与烧写寄存器 896
9.4-5 VDDQ 默认时序参数配置 898
9.4-6 eFuse 控制器的内部中断源 898

10.4-1 HP GPIO 唤醒信号触发条件和清除条件 950
10.4-2 LP GPIO 唤醒信号触发条件和清除条件 950
10.8-1 IO MUX Light-sleep 管脚功能控制寄存器 957
10.12-1 HP GPIO 交换矩阵外设信号 961
10.13-1 LP GPIO 交换矩阵外设信号 972
10.14-1 HP IO MUX 管脚功能 973
10.15-1 LP IO MUX 管脚功能 976
10.16-1 GPIO 管脚模拟功能 977
10.18-1 HP IO MUX 和 HP GPIO 交换矩阵内部中断源 979
10.18-2 LP IO MUX 和 LP GPIO 交换矩阵内部中断源 980

11.1-1 HP CPU 复位源 1034
11.1-2 LP CPU 复位源 1035

11.2-1 ROOT_CLK 时钟源选择 1038
11.2-2 LP_SLOW_CLK 时钟源选择 1039
11.2-3 LP_FAST_CLK 时钟源选择 1039
11.2-4 衍生高性能时钟源 1041
11.2-5 高性能外设时钟 1041
11.2-6 衍生低功耗时钟源 1042
11.2-7 低功耗系统外设时钟 1043

12.2-1 管脚默认上拉/下拉 1163
12.2-2 系统 Boot 模式 1163
12.2-3 ROM 日志打印 1165
12.2-4 JTAG 信号源控制 1166

13.4-1 CPU 外部中断源映射寄存器、外部中断状态寄存器、外部中断源 1170

14.3-1 事件任务矩阵通道*n* 可选事件 1211
14.3-2 事件任务矩阵通道*n* 可映射任务 1217

15.4-1 唤醒源 1366
15.4-2 高性能系统外设功能时钟 1369
15.4-3 VO2/3/4 寄存器对应关系 1372
15.4-4 调压器开关控制 1373
15.4-5 限流保护控制 1373
15.4-6 调压器等待计数器 1373
15.5-1 预设的功耗模式 1374

16.5-1 UNIT*n* 配置控制位 1441
16.5-2 报警触发条件 1442
16.5-3 同步操作 1443
16.6-1 系统定时器的内部中断源 1444

17.3-1 可逆计数器向上计数时的报警场景 1465
17.3-2 可逆计数器向下计数时的报警场景 1465
17.3-3 TIMG*n* 的内部中断源 1467

18.2-1 超时动作 1488

19.3-1 配置 RTC Timer 记录特定事件的发生时刻 1501
19.3-2 RTC Timer 目标时刻配置 1501
19.5-1 RTC Timer 的内部中断源 1502

20.1-1 PMP 和 APM 的管控区域 1512
20.5-1 APM 内部中断源 1518

21.3-1 系统寄存器的内部中断源 1615

22.5-1 HP CPU0/1 包格式 1715
22.5-2 LOST 包格式 1715
22.5-3 DMA_2 包格式 1717
22.5-4 DMA_3 包格式 1717

22.5-5 DMA LOST 包格式 1718

23.5-1 指标选择寄存器中的测量指标配置 1766
23.5-2 指标选择寄存器中的区间统计指标配置 1767
23.5-3 agent 选择寄存器配置值与实际 agent 对应关系表 1767
23.6-1 AXI PERF MON 的内部中断源 1770

24.3-1 LP 信箱控制器的内部中断源 1803

27.4-1 工作模式 1835
27.4-2 密钥长度和加解密方向 1835
27.5-1 状态返回值 1836
27.5-2 Typical AES 文本字节序 1836
27.5-3 AES-128 密钥字节序 1837
27.5-4 AES-256 密钥字节序 1838
27.6-1 块模式选择 1839
27.6-2 状态返回值 1840
27.6-3 TEXT-PADDING 1840
27.6-4 DMA-AES 存储字节序 1841
27.9-1 AES 的内部中断源 1846

28.3-1 ECC 硬件加速器内存块 1856
28.4-1 ECC 加速器曲线模式控制 1858
28.4-2 ECC 硬件加速器工作模式控制 1858

29.2-1 HMAC 功能及配置数值 1871

30.3-1 加速效果 1889
30.4-1 RSA 加速器的内部中断源 1889

31.3-1 工作模式选择 1897
31.3-2 运算标准选择 1897
31.4-4 不同运算标准信息摘要的寄存器占用情况 1903
31.5-1 SHA 加速器的内部中断源 1903

33.4-1 ECDSA_DS 椭圆曲线选择 1923
33.4-2 ECDSA_DS SHA 算法 1923
33.4-3 ECDSA_DS 工作状态 1924
33.7-1 ECDSA_DS 存储器块 1930

34.4-1 根据 Key_A、Key_B 及 Key_C 生成的 Key 值 1940
34.4-2 目标空间与寄存器堆的映射关系 1941

36.6-1 HUK 生成器工作模式 1959
36.9-1 密钥管理器和 HUK 生成器内部中断源 1973
36.10-1 密钥管理器内存块 1973
36.10-2 HUK 生成器内存块 1974

37.1-2 标记码描述 1991
37.2-1 不同图像格式 MCU 包含的数据单位 1991

37.5-1 颜色空间转换支持的图像格式 1995
37.5-2 量化系数表中系数的位置编号（示例） 1996
37.5-3 推荐用于亮度分量的 8 位精度量化系数表 1996
37.5-4 推荐用于色度分量的 8 位精度量化系数表 1996
37.5-5 采样因子和图像格式 1997
37.7-1 JPEG 编码器 hb 和 vb 配置 2003
37.7-2 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* 配置 2004
37.7-3 JPEG 解码器 JPEG_HA、JPEG_VA 的配置 2005
37.7-4 JPEG 解码器 hb 和 vb 配置 2005
37.7-5 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* 配置 2006
37.7-6 2D DMA 颜色空间转换配置 2007

38.5-1 LSC LUT 增益存储顺序 2036
38.5-2 ISP 输出像素排列格式 2042
38.6-1 ISP 内部中断源 2043

39.5-1 像素存储顺序 2138
39.5-2 PPA SRM Alpha 配置效果 2140
39.5-3 BLEND CLUT 地址 2141
39.5-4 PPA 链表配置 2142
39.5-5 自动补全块大小配置 2143
39.5-6 SRM 输出像素块重排数量 2145
39.6-1 SRM 参数错误类型 2146
39.6-2 BLEND 参数错误类型 2147

40.3-1 信号描述 2176
40.3-2 LCD 数据格式控制 - 阶段一 2181
40.3-3 LCD 输出数据重排列功能模式 2182
40.3-4 Camera 数据格式控制 2183
40.3-5 24 位输出时的转换模式配置 2184
40.3-6 非 24 位输出时的转换模式配置 2185

42.5-1 线路状态描述 2447
42.5-2 MIPI RX D-PHY 支持频率范围 2449
42.5-3 通道 0 HS RX 控制下的测试数据 (Test Data) 2453
42.5-4 通道 0 HS RX 控制下的测试数据输出 (Test Dout) 2454
42.5-5 通道 1 和通道 2 加扰器 PRBS 初始种子值 2454
42.5-6 MIPI CSI-2 主机可检测到的错误 2454

43.4-1 输入像素排列 2484
43.4-2 DPI 位排列 2485
43.4-3 色带编码 2490
43.4-4 MIPI BD D-PHY 支持的频率范围 2492
43.4-5 M 和 N 2494
43.4-6 VCO 的范围 2494
43.4-7 PLL CP 和 LPF 控制位 2495
43.4-8 0x10 的测试数据 2496
43.4-9 0x10 的测试数据输出 2496

43.4-10 0x11 的测试数据 2497
43.4-11 0x11 的测试数据输出 2497
43.4-12 0x12 的测试数据 2498
43.4-13 0x12 的测试数据输出 2498
43.4-14 0x17 的测试数据 2498
43.4-15 0x17 的测试数据输出 2499
43.4-16 0x18 的测试数据 2499
43.4-17 0x18 的测试数据输出 2499
43.4-18 0x18 的测试数据输出 2499
43.4-19 0x19 的测试数据 2500
43.4-20 0x19 的测试数据输出 2500
43.4-21 0x22 的测试数据 2500
43.4-22 0x22 的测试数据输出 2502
43.4-23 0x22 的测试数据输出 2502
43.4-24 0x44 的测试数据 2502
43.4-25 0x44 的测试数据输出 2502
43.5-1 DSI 桥中断 2505
43.5-2 DSI 主机中断 2505

45.2-1 UART 和 LP UART 特性区分 2576
45.4-1 UART_CHAR_WAKEUP 模式配置 2585

46.5-1 GP-SPI 和 LP-SPI 支持的数据模式 2660
46.5-2 SPI2/SPI3 总线信号和 LP_SPI 总线信号功能描述 2660
46.5-3 各种 SPI 模式下使用到的 SPI2 总线信号 2661
46.5-4 各种 SPI 模式下使用到的 SPI3 总线信号 2662
46.5-5 各种 SPI 模式下使用到的 LP_SPI 总线信号 2663
46.5-6 GP-SPI 用作主机和从机时的数据位控制（以 GP-SPI2 为例） 2664
46.5-7 LP-SPI 用作主机和从机时的数据位控制 2664
46.5-8 GP-SPI 用作主机或从机时支持的传输类型 2665
46.5-9 GP-SPI 用作从机时数据传输的中断触发条件 2670
46.5-10 1/2/4/8-bit 模式下状态控制寄存器 2677
46.5-11 命令值的发送顺序 2679
46.5-12 地址值的发送顺序 2679
46.5-13 CONF 阶段 BM 位图 2685
46.5-14 传输事务 *i* 中 CONF buffer*i* 配置示例 2685
46.5-15 BM 位图与待更新的寄存器 2686
46.5-16 GP-SPI 从机 SPI 模式支持的 CMD 值 2688
46.5-16 GP-SPI 从机 SPI 模式支持的 CMD 值 2689
46.5-17 GP-SPI 从机 QPI 支持的 CMD 值 2689
46.5-17 GP-SPI 从机 QPI 支持的 CMD 值 2690
46.7-1 主机时钟相位和极性配置 2695
46.7-2 从机时钟相位和极性配置 2696
46.9-1 LP-SPI 从机唤醒模式配置 2698
46.10-1 GP-SPI2 与 GP-SPI3 和 LP-SPI 的功能差异 2698
46.10-2 寄存器差异 2699
46.10-3 中断差异 2702

46.10-3 中断差异 2703
46.11-1 LP-SPI 和 GP-SPI 的内部中断源 2703
46.11-2 GP-SPI 主机中断表 2705
46.11-3 GP-SPI 从机中断表 2705

47.3-1 I2C 时序参数（引自 The I2C-bus specification Version 2.1 Table5） 2811
47.4-1 I2C 同步寄存器 2814

49.4-1 模块信号描述 2894
49.9-1 通道有效数据位宽控制 2901
49.9-2 通道有效数据字节序控制 2901
49.9-3 PDM 输出模式下 I2S*n* 取数逻辑 2904
49.9-4 原始 PDM 数据的 I2S*n* 通道模式控制 2905
49.9-5 I2S*0* PCM-to-PDM 数据输出 2905
49.10-1 PDM-to-PCM 数据输入 2908
49.10-2 通道储存数据位宽控制 2909
49.10-3 通道储存数据字节序控制 2910

50.3-1 模块信号描述 2936
50.8-1 PDM-to-PCM 数据输入 2942
50.8-2 通道储存数据字节序控制 2943

51.4-1 控制信号为低电平时输入脉冲信号上升沿的计数模式 2961
51.4-2 控制信号为高电平时输入脉冲信号上升沿的计数模式 2961
51.4-3 控制信号为低电平时输入脉冲信号下降沿的计数模式 2961
51.4-4 控制信号为高电平时输入脉冲信号下降沿的计数模式 2962
51.5-1 PCNT 的内部中断源 2962

52.4-1 Slave 模式下的 IN 和 OUT 事务级操作 2982

53.4-1 Slave 模式下的 IN 和 OUT 事务级操作 2993
53.5-1 UTMI+ OTG 接口 2994

54.3-1 标准 CDC-ACM 控制请求 3002
54.3-2 CDC-ACM 中 RTS 和 DTR 的设置 3003
54.3-3 半字节中的命令 3004
54.3-4 USB-JTAG 控制请求 3006
54.3-5 JTAG 功能描述符 3006
54.4-1 复位芯片进入下载模式 3007
54.4-2 复位 SoC 从 flash 启动 3007

55.4-1 发送描述符 0 (TDES0) 3039
55.4-2 发送描述符 1 (TDES1) 3042
55.4-3 发送描述符 2 (TDES2) 3042
55.4-4 发送描述符 3 (TDES3) 3042
55.4-5 发送描述符 6 (TDES6) 3043
55.4-6 发送描述符 7 (TDES7) 3043
55.4-7 接收描述符 0 (RDES0) 3043
55.4-8 接收描述符 1 (RDES1) 3045

55.4-9 接收描述符 2 (RDES2) 3045
55.4-10 接收描述符 3 (RDES3) 3046
55.4-11 接收描述符 4 (RDES4) 3046
55.4-12 接收描述符 6 (RDES6) 3047
55.4-13 接收描述符 7 (RDES7) 3047
55.4-14 目标地址过滤 3053
55.4-15 源地址过滤 3053

56.2-1 不同帧类型、帧格式下的域及子域信息 3111
56.2-2 错误帧中的位域信息 3112
56.2-3 过载帧中的位域信息 3112
56.2-4 帧间距中的域信息 3113
56.2-5 名义位时序中包含的段 3115
56.4-1 TWAI_BUS_TIMING_0_REG 的位信息 (0x18) 3119
56.4-2 TWAI_BUS_TIMING_1_REG 的位 (0x1c) 3120
56.4-3 SFF 与 EFF 的缓冲器布局 3120
56.4-4 TX/RX 帧信息 (SFF/EFF)； TWAI 地址 0x40 3121
56.4-5 TX/RX 标识符 1 (SFF); TWAI 地址 0x44 3121
56.4-6 TX/RX 标识符 2 (SFF); TWAI 地址 0x48 3121
56.4-7 TX/RX 标识符 1 (EFF); TWAI 地址 0x44 3122
56.4-8 TX/RX 标识符 2 (EFF); TWAI 地址 0x48 3122
56.4-9 TX/RX 标识符 3 (EFF); TWAI 地址 0x4c 3122
56.4-10 TX/RX 标识符 4 (EFF); TWAI 地址 0x50 3122
56.4-11 TWAI_ERR_CODE_CAP_REG 中的位信息 (0x30) 3127
56.4-12 SEG.4 - SEG.0 的位信息 3127
56.4-13 TWAI_ARB LOST CAP_REG 中的位信息 (0x2c) 3128
56.5-1 TWAI 的内部中断源 3129

57.3-1 SD/MMC 信号描述 3152
57.8-1 DES0 单元描述 3158
57.8-2 DES1 单元描述 3159
57.8-3 DES2 单元描述 3159
57.8-4 DES3 单元描述 3159

58.4-1 常用配置频率及精度 3198
58.7-1 LEDC 存储空间 3205

59.3-1 操作器模块包含的各个子模块功能 3224
59.3-2 PWM 生成器中的所有定时事件 3231
59.3-3 PWM 定时器递增计数时，定时事件的优先级 3232
59.3-4 PWM 定时器递减计数时，定时事件的优先级 3232
59.3-5 控制死区时间生成器开关的字段 3242
59.3-6 死区生成器的典型操作模式 3243
59.3-7 MCPWM 可产生的 ETM 事件 3250
59.3-8 MCPWM 可接收的 ETM 任务 3251

60.3-1 更新配置参数 3308

61.5-1 时钟限制下复位 AXI 时钟域操作步骤 3330
61.5-2 时钟限制下 TX 模块从机模式操作要求 3331
61.5-3 时钟限制下 RX 模块从机模式操作要求 3331
61.6-1 PARLIO 的内部中断源 3335

62.4-1 指令结构 3357
62.4-2 当 CTL_MUX_REL 为 1 时 CTL_MUX_SEL_n 生效的值 3358
62.4-3 CTL_MUX_SEL 的值 3359
62.4-4 指令操作码 3360
62.4-5 LOOP 操作码 3360
62.4-6 ADD 操作码 3361
62.4-7 IF 操作码 3361
62.4-8 IFN 操作码 3361
62.4-9 LDCTD 操作码 3362
62.4-10 LDCTI 操作码 3362
62.4-11 EOF 模式设置 3363

63.3-1 ESP32-P4 电容式触摸传感器的管脚 3380
63.3-2 电容式触摸传感器输出信号 3382
63.4-1 平滑触摸数据计算 3388
63.4-2 基准数据算法 3388
63.4-2 基准数据算法 3389
63.5-1 触摸传感器的内部中断源 3391

64.4-1 温度测量范围与温度偏移 3420

65.5-1 SAR ADC 通道及芯片管脚 3431

66.4-1 模拟电压比较器 PAD 和 GPIO 之间的映射关系 3469
66.6-1 模拟电压比较器的内部中断源 3470

66.7-4 ENA/RAW/ST 寄存器的配置 3479

插图

1.0-1 ESP32-P4 总线架构 66

2.1-1 CPU 核心系统框图 68
2.11-1 调试系统概述 148
2.12-1 2 位饱和计数器 166
2.13-1 使用 RunStall 时 CPU 核心状态的切换 173

3.0-1 追踪编码器概述图 175
3.2-1 指令追踪流程概览 176

4.3-1 Xespv2p2 内部结构 (MAC) 203
4.6-1 操作数依赖导致的内部锁 221
4.6-2 硬件资源冒险 222
4.6-3 控制冒险 222

5.1-1 LP CPU 概览 616
5.10-1 LP CPU 唤醒和睡眠流程 636

6.1-1 共用 GDMA-AHB 通道的模块 640
6.1-2 共用 GDMA-AXI 通道的模块 641
6.3-1 GDMA 控制器的架构 642
6.4-1 链表结构图 643
6.4-2 链表关系图 646
6.4-3 CRC 计算矩阵示意图 650

7.4-1 VDMA 架构 752
7.5-1 外设的传输层次结构 754
7.5-2 存储器的传输层次结构 754
7.5-3 VDMA 读仲裁器 755
7.5-4 VDMA 写仲裁器 756
7.5-5 VDMA 链表项 760
7.5-6 CH*n*_LLP_STATUS 回写字段 761

8.3-1 2D-DMA 基本架构图 801
8.4-1 链表结构图 802
8.4-2 补齐示意图（整体） 805
8.4-3 补齐示意图（细节） 805
8.4-4 JPEG 解码输出数据宏块排布 807
8.4-5 颜色空间转换示意图 808

9.3-1 系统结构和地址映射 872
9.3-2 HP L2MEM 地址布局 875
9.3-3 cache 结构 878
9.4-1 eFuse 系统数据通路 884
9.4-2 移位寄存器电路图（前 32 字节） 894
9.4-3 移位寄存器电路图（后 12 字节） 894

10.3-1 HP GPIO 交换矩阵、HP IO MUX 、LP GPIO 交换矩阵和 LP IO MUX 框图 946
10.3-2 焊盘内部结构 947
10.4-1 GPIO 输入经 HP IO MUX 运行时钟上升沿或下降沿同步 949
10.4-2 GPIO 输入信号滤波时序图 949
10.4-3 毛刺滤波器时序示例 951
10.10-1 未使能迟滞功能时 PAD 上的电平翻转示例 959
10.10-2 使能迟滞功能时 PAD 上的电平翻转示例 960

11.1-1 四种复位等级 1033
11.2-1 HP/LP 系统时钟 1037
11.3-1 时钟配置示例 1045

12.2-1 芯片 Boot 流程 1164

13.2-1 ESP32-P4 的中断流 1168
13.3-1 中断矩阵结构图 1168

14.3-1 事件任务矩阵通道*n* 架构 1211
14.3-2 事件任务矩阵时钟结构 1222

15.4-1 ESP32-P4 电源结构 1362
15.4-2 PMU 的主要工作流程 1364
15.4-3 VO1 模式选择控制图 1372
15.4-4 VO2/3/4 模式选择控制图 1372

16.3-1 系统定时器结构图 1440
16.5-1 系统定时器生成报警 1441

17.1-1 定时器组概览 1463
17.3-1 定时器组架构 1464

18.1-1 看门狗定时器概览 1485
18.2-1 ESP32-P4 的数字看门狗定时器 1487
18.3-1 SWD 控制器结构 1490

20.1-1 PMP 和 APM 的管控关系 1513
20.3-1 APM 控制器结构图 1515

23.4-1 AXI Perf Mon 的架构 1765

24.3-1 LP 信箱控制器系统框图 1802

25.3-1 欠压检测模块结构图 1819
25.4-1 欠压检测模块结构图 1821

26.3-1 ESP32-P4 电源检测器结构 1832

27.7-1 GCM 加密过程 1843

29.3-1 HMAC 附加填充位示意图 1874
29.3-2 HMAC 结构示意图 1874

32.3-1 软件准备工作与硬件工作流程 1912

33.5-1 ECDSA_DS 流程 1927

34.3-1 片外存储器加解密结构 1939

35.3-1 噪声源 1951

36.5-1 密钥管理器 & HUK 生成器架构 1958
36.6-1 AES 部署模式 1961
36.6-2 ECDH0 部署模式 1962
36.6-3 ECDH1 部署模式 1964

37.1-1 JPEG 码流格式 1991
37.2-1 JPEG 图像编解码器的系统连接框图 1992
37.4-1 JPEG 编码器架构 1993
37.4-2 JPEG 解码器架构 1994

38.1-1 ESP32-P4 ISP SoC 架构 2031
38.4-1 ISP IP 架构 2033
38.5-1 gamma 曲线示意 2037
38.5-2 子窗口编号示意图 2038
38.5-3 AWB 窗口示意图 2039
38.5-4 ESP32-P4 CSI CM 结构 2041
38.5-5 字节顺序 2043

39.1-1 PPA 2D-DMA 连接关系 2136
39.4-1 PPA 系统架构 2137
39.5-1 PPA SRM 输入像素处理 2139
39.5-2 PPA SRM Alpha 配置逻辑 2139
39.5-3 2D-DMA 链表 2142
39.5-4 SRM 步骤示例 2144

40.3-1 LCD_CAM 功能框图 2176
40.3-2 LCD 时钟 2178
40.3-3 Camera 时钟 2179
40.3-4 LCD 视频帧结构 2186
40.3-5 LCD 时序（RGB 格式） 2186
40.3-6 LCD 时序（I8080/MOTO6800 格式） 2187

41.4-1 H264 编码器架构 2212
41.5-1 ENC_CORE 架构 2213
41.5-2 H264_DMA 架构 2219
41.5-3 H264_DMA 链表描述符 2220
41.7-1 GOP 模式下的图像编码顺序 2226
41.7-2 双流模式下图像编码的顺序 2229

42.4-1 MIPI CSI 架构 2446
42.5-1 MIPI 通道线路电平 2447

42.5-2 MIPI RX D-PHY 操作状态和模式 2448
42.5-3 MIPI CSI HS 数据接收序列 2450
42.5-4 时钟和数据通道 ULPS 序列 2451
42.5-5 控制接口时序图 2453
42.6-1 MIPI CSI 中断原理 2456

43.3-1 MIPI DSI 架构 2483
43.4-1 命令传输周期 2487
43.4-2 非突发带同步脉冲的 outvact_lpcmd_time 2487
43.4-3 突发或非突发带同步事件的 outvact_lpcmd_time 2488
43.4-4 非突发带同步脉冲的 invact_lpcmd_time 2488
43.4-5 突发或非突发带同步事件的 invact_lpcmd_time 2488
43.4-6 垂直色带 2489
43.4-7 水平色带 2489
43.4-8 RGB888 BER 测试图案 2490
43.4-9 PHY 运行模式 2491
43.4-10 MIPI DSI HS 数据传输序列 2493
43.4-11 MIPI DSI 时钟架构 2503
43.4-12 视频时序 2504

44.3-1 ESP32-P4 VAD 系统架构 2568

45.3-1 UART 基本架构图 2578
45.4-1 UART 控制器分频 2580
45.4-2 UART 信号下降沿较差时序图 2581
45.4-3 UART 数据帧结构 2581
45.4-4 AT_CMD 字符格式 2582
45.4-5 RS485 模式驱动控制结构图 2583
45.4-6 SIR 模式编解码时序图 2584
45.4-7 IrDA 编解码结构图 2584
45.4-8 硬件流控图 2585
45.4-9 硬件流控信号连接图 2586
45.4-10 GDMA 模式数据传输 2587
45.6-1 UART 编程流程 2590

46.4-1 SPI 模块概览 2659
46.5-1 CPU 控制的传输中使用的数据 Buffer 2666
46.5-2 GP-SPI 功能块图 2671
46.5-3 GP-SPI 用作主机时的数据流控制 2672
46.5-4 GP-SPI 用作从机时的数据流控制 2672
46.5-5 GP-SPI 主机状态机 2675
46.5-6 GP-SPI2 主机使用全双工与 SPI 从机通信 2680
46.5-7 4-bit 模式下 GP-SPI2 与 Flash 以及外部 RAM 的连接方式 2682
46.5-8 GP-SPI2 发送到 Flash 的 SPI Quad I/O Read 命令序列 2683
46.5-9 DAM 控制的分段配置传输 2683
46.6-1 GP-SPI 访问外部 RAM 时推荐的 CS 时序配置 2692
46.6-2 GP-SPI 访问 Flash 时推荐的 CS 时序配置 2693

46.7-1 SPI 时钟模式 0 和时钟模式 2 2694
46.7-2 SPI 时钟模式 1 和时钟模式 3 2695
46.8-1 GP-SPI 主机时序补偿控制图 2696
46.8-2 GP-SPI2 主机模式下时序补偿 2697

47.3-1 I2C 主机基本架构 2810
47.3-2 I2C 从机基本架构 2810
47.3-3 I2C 协议时序（引自 The I2C-bus specification Version 2.1 Fig.31） 2811
47.4-1 I2C 时序图 2815
47.4-2 I2C 命令寄存器结构 2817
47.6-1 I2C 主机写 7 位寻址的从机 2821
47.6-2 I2C 主机写 10 位寻址的从机 2822
47.6-3 I2C 主机写 7 位双地址寻址从机 2824
47.6-4 I2C 主机分段写 7 位寻址的从机 2825
47.6-5 I2C 主机读 7 位寻址的从机 2827
47.6-6 I2C 主机读 10 位寻址的从机 2829
47.6-7 I2C 主机从 7 位双寻址从机的 M 地址读取 N 个数据 2831
47.6-8 I2C 主机分段读 7 位寻址的从机 2833

48.3-1 模拟 I2C 主机模块结构图 2883
48.4-1 信号相位图 2883

49.4-1 ESP32-P4 I2S 系统框图 2892
49.5-1 时序图 – TDM Philips 标准 2895
49.5-2 时序图 – TDM MSB 对齐标准 2896
49.5-3 时序图 – TDM PCM 标准 2896
49.5-4 时序图 – PDM 标准 2897
49.6-1 I2S*n* 时钟分频器 2897
49.9-1 TX 数据格式控制 2903
49.9-2 TDM 通道控制 2904
49.9-3 PDM 通道控制 2906

50.3-1 ESP32-P4 LP I2S 系统架构 2935
50.4-1 时序图 – TDM Philips 标准 2937
50.4-2 时序图 – TDM MSB 对齐标准 2937
50.4-3 时序图 – TDM PCM 标准 2938
50.4-4 时序图 – PDM 标准 2938
50.5-1 LP I2S 时钟分频器 2939

51.1-1 PCNT 框图 2959
51.3-1 PCNT 单元基本架构图 2960
51.6-1 通道 0 递增计数图 2963
51.6-2 通道 0 递减计数图 2964
51.6-3 双通道递增计数图 2964

52.4-1 OTG_HS 系统架构 2976
52.4-2 内核地址映射 2977
52.4-3 主机模式 FIFO 2978

52.4-4 设备模式 FIFO 2979
52.4-5 OTG_HS 中断层次结构图 2980
52.4-6 Scatter/Gather DMA 链表结构 2981

53.4-1 OTG_FS 系统架构 2986
53.4-2 内核地址映射 2987
53.4-3 主机模式 FIFO 2989
53.4-4 设备模式 FIFO 2990
53.4-5 OTG_FS 中断层次结构图 2991
53.4-6 Scatter/Gather DMA 链表结构 2992
53.5-1 A 设备 SRP 2995
53.5-2 B 设备 SRP 2996
53.5-3 A 设备 HNP 2997
53.5-4 B 设备 HNP 2998

54.2-1 USB 串口/JTAG 控制器高层框图 3001
54.2-2 USB 串口/JTAG 控制器框图 3002

55.1-1 Ethernet MAC 功能概述 3034
55.3-1 EMAC 功能框图 3036
55.4-1 发送描述符 3039
55.4-2 接收链表结构 3043
55.4-3 MII 接口 3048
55.4-4 MII 时钟 3049
55.4-5 RMII 接口 3050
55.4-6 RMII 时钟 3051

56.2-1 数据帧和远程帧中的位域 3110
56.2-2 错误帧中的位域 3112
56.2-3 过载帧中的位域 3112
56.2-4 帧间距中的域（* 关于被动错误） 3113
56.2-5 标称位构成 3115
56.3-1 TWAI 概略图 3117
56.4-1 接收过滤器 3123
56.4-2 单过滤模式 3124
56.4-3 双过滤模式 3125
56.4-4 错误状态变化 3126
56.4-5 丢失仲裁的 bit 位置 3129

57.2-1 SD/MMC 控制器连接的拓扑结构 3151
57.3-1 SD/MMC 控制器外部接口信号 3152
57.4-1 SDIO 主机结构框图 3153
57.4-2 命令通路状态机 3154
57.4-3 数据发送状态机 3155
57.4-4 数据接收状态机 3155
57.7-1 链表结构 3157
57.8-1 描述符结构 3158

58.3-1 LED PWM 控制器架构 3195
58.3-2 定时器和 PWM 生成器功能框图 3195
58.4-1 LEDC_CLK_DIV 为非整数时的分频 3197
58.4-2 计数器和 PWM 信号精度关系 3197
58.4-3 LED PWM 输出信号图 3199
58.4-4 输出信号占空比线性渐变图 3200
58.4-5 输出信号占空比伽马曲线渐变图 3201

59.2-1 MCPWM 外设概览 3221
59.3-1 定时器模块 3223
59.3-2 操作器模块 3223
59.3-3 捕获模块 3224
59.3-4 ETM 模块 3224
59.3-5 递增计数模式波形 3225
59.3-6 递减计数模式波形 3226
59.3-7 递增递减循环模式波形，同步事件后递减 3226
59.3-8 递增递减循环模式波形，同步事件后递增 3226
59.3-9 递增模式中生成的 UTEP 和 UTEZ 3227
59.3-10 递减模式中生成的 DTEP 和 DTEZ 3228
59.3-11 递增递减模式中生成的 DTEP 和 UTEZ 3228
59.3-12 PWM 操作器的子模块 3230
59.3-13 递增递减模式下的对称波形 3233
59.3-14 递增计数模式，单边不对称波形，PWM*x*A 和 PWM*x*B 独立调制–高电平 3234
59.3-15 递增计数模式，脉冲位置不对称波形，PWM*x*A 独立调制 3235
59.3-16 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–高电平有效 3236
59.3-17 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–互补 3237
59.3-18 递增递减循环计数模式，错误或同步事件，在 PWM*x*A 和 PWM*x*B 上相同调制 3238
59.3-19 NCI 在 PWM*x*A 输出上软件强制事件示例 3239
59.3-20 CNTU 在 PWM*x*B 输出上软件强制事件示例 3240
59.3-21 死区模块的开关拓扑 3242
59.3-22 高电平有效互补 (AHC) 死区波形 3243
59.3-23 低电平有效互补 (ALC) 死区波形 3244
59.3-24 高电平有效 (AH) 死区波形 3244
59.3-25 低电平有效 (AL) 死区波形 3245
59.3-26 PWM 载波操作的波形示例 3246
59.3-27 载波模块的第一个脉冲和之后持续的脉冲示例 3247
59.3-28 PWM 载波模块中持续脉冲的 7 种占空比设置 3248

60.3-1 RMT 结构框图 3303
60.3-2 RAM 中脉冲编码结构 3304

61.4-1 PARLIO 系统框图 3328
61.5-1 PARLIO 时钟生成 3329
61.5-2 主机时钟正向波形 3331
61.5-3 主机时钟负向波形 3331
61.5-4 RX 模块电平使能模式 3332
61.5-5 RX 模块脉冲使能子模式 3333

61.5-6 RX 模块软件使能模式 3333

62.1-1 比特调节器系统架构图 3353
62.3-1 比特调节器数据路径图 3354
62.3-2 比特调节器控制路径图 3356

63.3-1 触摸传感器系统 3380
63.3-2 触摸传感器内部结构 3381
63.4-1 Touch FSM 内部结构和工作原理 3383
63.4-2 TOUCH_OUT 信号预处理 3384
63.4-3 扫描模式时序图 3386
63.4-4 触摸传感器跳频时序图 3387
63.4-5 感测面积示意图 3389

64.3-1 温度传感器结构框图 3419

65.1-1 SAR ADC 结构概图 3428
65.4-1 SAR ADC 结构图 3430
65.5-1 HP ADC FSM 概况 3433
65.5-2 ADC_SAR*x*_PATT_TAB1_REG 包含样式 0 - 3 3434
65.5-3 ADC_SAR*x*_PATT_TAB2_REG 包含样式 4 - 7 3434
65.5-4 ADC_SAR*x*_PATT_TAB3_REG 包含样式 8 - 11 3434
65.5-5 ADC_SAR*x*_PATT_TAB4_REG 包含样式 12 - 15 3434
65.5-6 样式结构 3434
65.5-7 HP ADC1 cmd0 配置示例 3435
65.5-8 HP ADC1 cmd1 配置示例 3435
65.5-9 GDMA 数据格式 3436

66.3-1 模拟电压比较器系统框图 3469

第 I 卷

微处理器和主机

该部分介绍构成芯片系统的基本要素，包括高性能 CPU 和低功耗 CPU 的微处理器架构、CPU 的扩展指令、用于调试的跟踪编码等。

第 1 章

总线架构

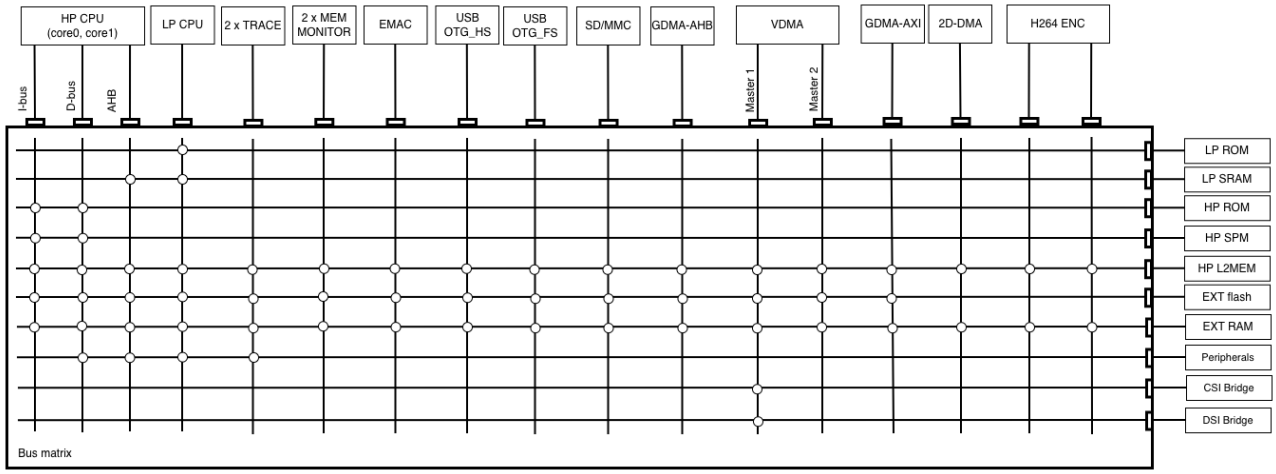
ESP32-P4 SoC 采用多层总线矩阵作为通信架构的核心，通过高效互连的结构连接多个主设备和从设备。该架构支持并发访问，有效缓解访问冲突，使高性能和低功耗的 RISC-V 核心、DMA 控制器、调试接口及各类外设能够在竞争最小的情况下同时运行。通过优化数据流与系统性能，即使多个高速外设同时工作，总线矩阵也能够确保系统高效运转。

ESP32-P4 SoC 的主系统由以下部分组成：

- 主设备 (master)：
  - 高性能 (HP) 32 位 RISC-V 处理器 0 的 I-bus、D-bus 和 AHB 总线
  - 高性能 (HP) 32 位 RISC-V 处理器 1 的 I-bus、D-bus 和 AHB 总线
  - 低功耗 (LP) 32 位 RISC-V 处理器
  - TRACE0（Trace 接口 0）
  - TRACE1（Trace 接口 1）
  - HP L2MEM MONITOR （HP L2MEM 监测器）
  - HP SPM MONITOR （HP SPM 监测器）
  - EMAC
  - USB OTG_HS（USB 2.0 高速 OTG）
  - USB OTG_FS（USB 2.0 全速 OTG）
  - SD/MMC
  - GDMA-AHB
  - VDMA
  - GDMA-AXI
  - 2D-DMA
  - H264 ENC
- 从设备 (slave)：
  - LP ROM
  - LP SRAM
  - HP ROM
  - HP SPM
  - HP L2MEM

- 外部 flash
- 外部 RAM
- 外设（LP & HP & HP CPU 外设）
- CSI Bridge
- DSI Bridge

ESP32-P4 的总线架构如下图所示。



注：

1. 图中交叉点处的圆圈表示对应列的总线可以访问对应行的从设备。
2. 图中 Peripherals 包括 LP、HP 以及 HP CPU 外设。
3. 图中 CPU 代表高性能 RISC-V 处理器 0 和处理器 1，两个处理器核心有独立的 I-bus, D-bus 和 AHB 总线接口。
4. 图中 2x MEM MONITOR 分别为 HP L2MEM MONITOR 和 HP SPM MONITOR。

图 1.0-1. ESP32-P4 总线架构

图 1.0-1 *ESP32-P4 总线架构* 中的总线说明如下：

- **HP CPU I-bus 总线**：连接高性能 (HP) 32 位 RISC-V 处理器的指令总线和总线矩阵，用于处理器取指。直接访问 HP SPM 或者通过 cache 间接访问其他包含代码的存储器，包括 HP ROM、HP L2MEM 以及外部 flash 和外部 RAM；通常情况下，cache 对用户而言是透明的。
- **HP CPU D-bus 总线**：连接高性能 (HP) 32 位 RISC-V 处理器的数据总线和总线矩阵，用于字面量加载和调试访问。直接访问 HP SPM 或者通过 cache 访问其它存储代码或数据的存储器（如 HP ROM、或外部 flash 和外部 RAM），还可以直接访问外设寄存器，但仅限于 HP CPU 外设；通常情况下，cache 对用户而言是透明的。
- **HP CPU AHB 总线**：连接高性能 (HP) 32 位 RISC-V 处理器的 AHB 总线和总线矩阵，用于访问外设或存储器中的数据。通过 AHB 接口可访问 LP 与 HP 外设，并可通过地址映射配置直接访问 HP L2MEM、HP ROM、LP SRAM 以及外部 flash 和外部 RAM （地址映射的详细配置请参阅章节 9 *系统和存储器*）。
- **LP CPU 总线**：连接低功耗 (LP) 32 位 RISC-V 处理器和总线矩阵，供 LP CPU 访问外设或存储器。可访问对象包括 LP SRAM, LP ROM, HP L2MEM、HP ROM 和外部 flash，外部 RAM 以及 LP & HP & HP CPU 外设。
- **TRACE 总线**：连接 TRACE0 和 TRACE1 模块和总线矩阵，用于执行调试操作。可访问对象包括 HP L2MEM 以及外部 RAM。

- **MEM MONITOR 总线**：连接 HP L2MEM MONITOR 和 HP SPM MONITOR 与总线矩阵，用于将各自监控的内存区域的监控数据写入存储器。两个 MEM MONITOR 接口访问通路相同，可访问对象包括 HP L2MEM 以及外部 RAM。
- **EMAC 总线**：连接以太网 MAC 控制器和总线矩阵，用于高速网络数据传输。可访问对象包括 HP L2MEM 以及外部 RAM。
- **USB OTG_HS 总线**：连接 USB 2.0 高速 OTG 控制器和总线矩阵，用于 USB 2.0 高速数据传输。可访问对象包括 HP L2MEM 以及外部 RAM。
- **USB OTG_FS 总线**：连接 USB 2.0 全速 OTG 控制器和总线矩阵，用于 USB 2.0 全速数据传输。可访问对象包括 HP L2MEM 以及外部 RAM。
- **SD/MMC 总线**：连接 SD/MMC 控制器和总线矩阵，用于 SD/MMC 卡的数据读写。可访问对象包括 HP L2MEM 以及外部 RAM，支持通过 DMA 进行高速数据传输。
- **GDMA 总线**：芯片包含 2 种 GDMA 控制器和 1 个 VDMA 控制器用于高速数据搬运。
  - **GDMA-AHB**：基于 AHB 总线协议的 DMA 控制器
  - **GDMA-AXI**：基于 AXI 总线协议的 DMA 控制器
  - **VDMA**：基于 AXI 总线协议的 Video DMA 控制器

  DMA 均提供一条连接至总线矩阵的线路和一条连接外部存储器的线路。GDMA 通过这些总线进行高速数据传输，包括外设到内存、内存到外设、外设到外设以及内存到内存的传输。可访问对象包括 HP L2MEM, 以及外部 flash 和外部 RAM。针对 DMA 访问外部存储器的场景，为确保数据吞吐率，建议尽可能设置较长的 burst 长度。
- **2D-DMA 总线**：连接 DMA2D 图形加速器和总线矩阵，用于 2D 图形数据搬运和处理。可访问对象包括 HP L2MEM 以及外部 RAM。
- **H264 ENC 总线**：连接 H.264 视频编码器和总线矩阵，包含两个 AXI 接口，访问通路相同。用于视频编码过程中的数据访问。可访问对象包括 HP L2MEM 以及外部 RAM。

第 2 章

高性能处理器

2.1 概述

ESP32-P4 采用了乐鑫第二代 32 位 RISC-V 指令集架构 (ISA) 的 CPU 核心系统。核心系统由双 RISC-V CPU 核心组成，配有专用的核心本地中断控制器 (CLIC)、调试模块以及核心本地中断 (CLINT) 定时器。图 2.1 为 CPU 核心系统的框图。

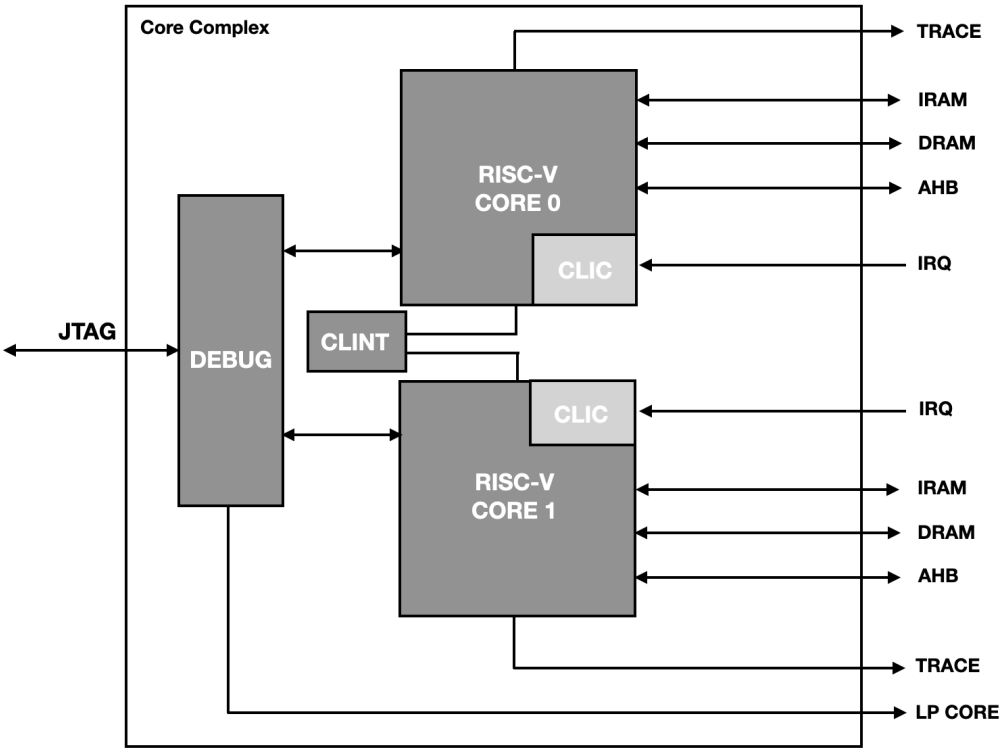


图 2.1-1. CPU 核心系统框图

在 CPU 核心系统中，每个 RISC-V 核心均使用 5 级顺序标量流水线架构，针对面积、功耗和性能进行了优化，内置专用的核心本地中断控制器 (CLIC) 和系统总线 (SYS BUS) 接口，用于访问存储器和外设。CLINT 定时器由两个核心共享。调试模块将 JTAG 接口连接至双核心，也支持 LP 核心调试。每个核心还提供用于离线调试的追踪接口。

2.2 特性

CPU 核心系统中的每个 RISC-V CPU 核心均实现了以下标准 RISC-V 扩展：

- 必须实现的基本整数 (I)
- 乘法/除法 (M)
- 原子 (A)
- 浮点 (F)
- 压缩 (C)
- 代码体积压缩 (Zc)
- 位操作 (Zb)

除了上述标准扩展之外，每个 RISC-V CPU 核心还实现了自定义指令 (X)，大致可分为以下两类：

- 加速 AI 算法的自定义指令
- 自定义硬件循环指令，进一步提高性能并减少 AI 算法重复执行时的代码大小

除了上述 RISC-V 指令扩展，每个 CPU 核心还支持以下特性：

- RISC-V RV32IMAFCX ISA，具有五级流水线，支持高达 360 MHz 的工作时钟频率
- 兼容《RISC-V 指令集手册 v2.2 第一卷 “RISC-V 用户级 ISA”》(RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2)
- 兼容《RISC-V 指令集手册 v1.10 第二卷 “特权架构”》(RISC-V Instruction Set Manual, Volume II: RISC-V Privileged Architecture, Version 1.10)
- 兼容 2023/10/10 版《核心本地中断控制器 (CLIC) RISC-V 特权架构扩展》(Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extensions, 2023/10/10)
- 兼容《RISC-V 外部调试支持规范 v0.13.2》(RISC-V External Debug Support Version 0.13.2)
- 通过 IRAM/DRAM 接口零等待周期访问片上 SRAM 和缓存中的程序和数据
- 32 位 AHB 系统总线用于访问外设
- 专用于各个特权模式的核心本地中断 (CLINT)
- 支持用户 (U) 特权模式
- 支持用户模式中断委派
- 总线错误响应
- 标准物理存储器保护 (PMP) 最多可配置 32 个区域，自定义物理存储器属性 (PMA) 最多可配置 16 个区域，以确保安全
- 分支目标缓冲器 (BTB) 具有 BHT 和 RAS 以提高性能
- 调试模块 (DM) 符合《RISC-V 外部调试支持规范 v0.13.2》，支持通过行业标准的 JTAG/USB 端口连接外部调试器
- 调试器通过 Core 0 以及系统总线 (SBA) 直接访问存储器和外设
- 支持指令追踪以进行离线调试
- 硬件触发器符合《RISC-V 调试规范 v0.13.2》(RISC-V External Debug Support Version 0.13.2)，具有多达 3 个断点/观察点
- 支持多达 8 个快速 GPIO

- 可配置的核心性能指标事件

2.3 术语

为了更好地说明高性能处理器的功能，本章使用了以下术语。

ABI	应用二进制接口 (application binary interface)
分支 (branch)	在一定条件下改变执行流程的指令
CLIC	核心本地中断控制器 (core-local interrupt controller)
CLINT	核心本地中断 (core-local interrupt)
CSR	控制和状态寄存器 (control and status register)
delta	指令在存储器中的存储地址不连续时，程序计数器的地址偏移
DTM	调试传输模块 (debug transport module)
FPR	浮点寄存器 (floating-point register)
GPIO	通用输入/输出 (general-purpose input/output)
GPR	通用寄存器 (general-purpose register)
HWLP	硬件循环 (hardware loop)
hart	RISC-V 硬件线程
LSB	最低有效位 (least significant bit)
LR	加载保留 (load reserved)
MSB	最高有效位 (most significant bit)
PMP	物理存储器保护 (physical memory protection)
退役 (retire)	机器状态更新时，执行指令的最后一个阶段
RO	只读域 (read-only)
R/W	可读写域 (readable and writable)
SC	条件存储 (store conditional)
SoC	片上系统 (system on chip)
WO	只写域 (write-only)
WARL	写入任何读取合法 (write any read legal)
W1	写入 1 (write 1)

2.4 地址分布

下表列出了 CPU 可访问的指令地址空间、数据地址空间和通过系统总线访问的外设地址空间。

表 2.4-1. CPU 地址分布

名称	描述	起始地址	结束地址	访问类型
CPU	CLIC/CLINT 寄存器	0x2000_0000	0x2FFF_FFFF	R/W
IRAM/DRAM	指令/数据区域	0x3000_0000	0x3FEF_FFFF	R/W
CPU 外设		0x3FF0_0000	0x3FF1_FFFF	R/W
IRAM/DRAM	指令/数据区域	0x3FF2_0000	0x4FFF_FFFF	R/W
CPU 外设	HP APB 外设	0x5000_0000	0x500F_FFFF	R/W
LP RAM/ROM		0x5010_0000	0x5010_FFFF	R/W

LP APB 外设		0x5011_0000	0x5012_FFFF	R/W
AHB	AHB 外设区域	* 默认	* 默认	R/W

*** 默认：** 上述地址范围以外的地址空间通过 AHB 总线访问。

2.5 配置和状态寄存器 (CSRs)

2.5.1 寄存器列表

下表为每个 HP CPU 核心实现的 CSR 列表。所有已实现的 CSR 都遵循《RISC-V 指令集手册 v1.10 第二卷 "特权架构"》中所述的位域标准映射，但是，由于 CPU 并未实现所有的功能子集，所以某些标准 CSR 中的部分位域并未实现。有关详细的 CSR 寄存器描述，请参阅下一小节。

名称	描述	地址	访问
机器信息 CSR			
mvendorid	机器供应商 ID	0xF11	RO
marchid	机器架构 ID	0xF12	RO
mimpid	机器实现 ID	0xF13	RO
mhartid	机器 hart ID	0xF14	RO
机器陷阱设置 CSR			
mstatus	机器模式状态	0x300	R/W
misa [1]	机器 ISA	0x301	R/W
mideleg	机器中断委托寄存器（在 CLIC 模式下无效）	0x303	RO
mie	机器中断使能寄存器（在 CLIC 模式下无效）	0x304	RO
mtvec [2]	机器陷阱向量	0x305	R/W
mcounteren	机器计数器使能	0x306	R/W
mtvt	机器向量中断基地址（参见 CLIC 规范）	0x307	R/W
机器陷阱处理 CSR			
mscratch	机器暂存	0x340	R/W
mepc	机器陷阱程序计数器	0x341	R/W
mcause [3]	机器陷阱原因	0x342	R/W
mtval	机器陷阱值	0x343	R/W
mip	机器中断挂起（在 CLIC 模式下无效）	0x344	RO
mnxti	中断处理程序地址和使能（参见 CLIC 规范）	0x345	R/W
mintstatus	当前中断级别（参见 CLIC 规范）	0xFB1	RO
mintthresh	中断阈值（参见 CLIC 规范）	0x347	R/W
mscratchcsw	特权模式改变时的条件暂存交换（参见 CLIC 规范）	0x348	R/W
mscratchcswl	级别改变时的条件暂存交换（参见 CLIC 规范）	0x349	R/W
mclicbase	机器模式中断控制器基地址寄存器（自定义）（参见 CLIC 规范）	0x350	RO
用户陷阱设置 CSR			

[1] 尽管 misa 具有读/写属性，但由于它的域是硬连线的，所以写操作无效。在 RISC-V 术语中称为 WARL（写入任意数值读取合法数值）。
[2] mtvec 保存了陷阱向量配置，包括向量基地址和向量模式。
[3] mcause 中反映的外部中断 ID 也包括 RISC-V 标准为核心内部中断源预留的 ID。

名称	描述	地址	访问
ustatus	用户模式状态	0x000	R/W
utvec [4]	用户陷阱向量	0x005	R/W
utvt	用户向量中断基地址（参见 CLIC 规范）	0x007	R/W
用户陷阱处理 CSR			
uscratch	用户暂存寄存器	0x040	R/W
uepc	用户陷阱程序计数器	0x041	R/W
ucause [5]	用户陷阱原因	0x042	R/W
unxti	中断处理程序地址和使能寄存器（参见 CLIC 规范）	0x045	R/W
uintthresh	中断阈值（参见 CLIC 规范）	0x047	R/W
uclicbase	用户模式中断控制器基地址寄存器（自定义）（参见 CLIC 规范）	0x050	RO
uintstatus	当前中断级别（参见 CLIC 规范）	0xCB1	RO
物理存储器保护 (PMP) CSR			
pmpcfg0	物理存储器保护配置	0x3A0	R/W
pmpcfg1	物理存储器保护配置	0x3A1	R/W
pmpcfg2	物理存储器保护配置	0x3A2	R/W
pmpcfg3	物理存储器保护配置	0x3A3	R/W
pmpcfg4	物理存储器保护配置	0x3A4	R/W
pmpcfg5	物理存储器保护配置	0x3A5	R/W
pmpcfg6	物理存储器保护配置	0x3A6	R/W
pmpcfg7	物理存储器保护配置	0x3A7	R/W
pmpaddr*n* (*n*: 0-31)	物理存储器保护地址寄存器	0x3B0+0x1**n*	R/W
触发器模块 CSR（与调试模式共享）			
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
tdata3	触发器抽象数据寄存器 3	0x7A3	R/W
tinfo	触发器信息寄存器	0x7A4	R/W
tcontrol	全局触发器控制	0x7A5	R/W
mcontext	机器模式上下文寄存器	0x7A8	R/W
调试模式 CSR（仅在调试模式下可访问）			
dcsr	调试控制和状态	0x7B0	R/W
dpc	调试 PC	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试暂存寄存器 1	0x7B3	R/W
dmcs2	调试模式状态控制寄存器 2	0x32	R/W
机器计数器设置			
mcountinhibit	机器计数器控制寄存器	0x320	R/W
mhpmevent8	机器性能监控事件选择器	0x308	R/W
mhpmevent9	机器性能监控事件选择器	0x329	R/W
mhpmevent13	机器性能监控事件选择器	0x32D	R/W

[4] utvec 保存陷阱向量配置信息，包括向量基地址和向量模式。

[5] mcause 中反映的外部中断 ID 也包括 RISC-V 标准为核心内部中断源预留的 ID。

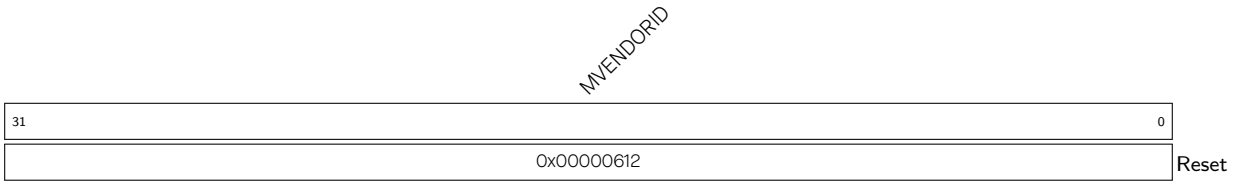
名称	描述	地址	访问
机器计数器/定时器			
mcycle	机器周期计数器	0xB00	R/W
minstret	机器指令退役计数器	0xB02	R/W
mhpmcounter8	机器性能监控计数器	0xB08	R/W
mhpmcounter9	机器性能监控计数器	0xB09	R/W
mhpmcounter13	机器性能监控计数器	0xB0D	R/W
mcycleh	mcycle 的高 32 位	0xB80	R/W
minstreth	minstret 的高 32 位	0xB82	R/W
mhpmcounter8h	mhpmcounter8 的高 32 位	0xB88	R/W
mhpmcounter9h	mhpmcounter9 的高 32 位	0xB89	R/W
mhpmcounter13h	mhpmcounter13 的高 32 位	0xB8D	R/W
非特权/用户模式浮点 CSR			
fflags	浮点累积异常	0x001	R/W
frm	浮点动态舍入模式	0x002	R/W
fcsr	浮点控制和状态寄存器 (frm + fflags)	0x003	R/W
非特权/用户模式计数器/定时器			
cycle	RDCYCLE 指令的周期计数器	0xC00	RO
time	RDTIME 指令的定时器	0xC01	RO
instret	RDINSTRET 指令的退役计数器	0xC02	RO
hpmcounter8	性能监控计数器	0xC08	RO
hpmcounter9	性能监控计数器	0xC09	RO
hpmcounter13	性能监控计数器	0xC0D	RO
cycleh	cycle 的高 32 位	0xC80	RO
instreth	instret 的高 32 位	0xC82	RO
hpmcounter8h	hpmcounter8 的高 32 位	0xC88	RO
hpmcounter9h	hpmcounter9 的高 32 位	0xC89	RO
hpmcounter13h	hpmcounter13 的高 32 位	0xC8D	RO
自定义系统 CSR（自定义）			
浮点用户模式 CSR（自定义）			
fxcr	浮点扩展控制寄存器	0x800	R/W
GPIO 访问 CSR（自定义）			
cpu_gpio_oen	GPIO 输出使能	0x803	R/W
cpu_gpio_in	GPIO 输入值	0x804	RO
cpu_gpio_out	GPIO 输出值	0x805	R/W
扩展控制和状态 CSR（自定义）			
mext_ill_reg	机器扩展非法寄存器	0x7F0	R/W
mhwloop_state_reg	机器模式 HWLP 扩展状态寄存器	0x7F1	R/W
mext_pie_status	机器模式 PIE 扩展状态寄存器	0x7F2	R/W
Zc 扩展 CSR			
jvt	机器向量跳转表基地址和控制寄存器	0x017	R/W
机器扩展状态 CSR（自定义）			
mexstatus	机器扩展控制和状态寄存器	0x7E1	R/W
mhint	机器隐式操作寄存器	0x7C5	R/W

名称	描述	地址	访问
物理存储器属性检查器 (PMAC) CSR（自定义）			
pma_cfg*n* (*n*: 0-15)	物理存储器属性配置寄存器	0xBC0+0x1**n*	R/W
pma_addr*n* (*n*: 0-15)	物理存储器属性地址寄存器	0xBD0+0x1**n*	R/W
机器硬件循环 CSR（自定义）			
mhloop0_start_addr	循环 0 的 32 位起始地址	0x7C6	R/W
mhloop0_end_addr	循环 0 的 32 位结束地址	0x7C7	R/W
mhloop0_count	硬件循环 0 的循环计数	0x7C8	R/W
mhloop1_start_addr	循环 1 的 32 位起始地址	0x7C9	R/W
mhloop1_end_addr	循环 1 的 32 位结束地址	0x7CA	R/W
mhloop1_count	硬件循环 1 的循环计数	0x7CB	R/W
用户硬件循环 CSR（自定义）			
uhloop0_start_addr	循环 0 的 32 位起始地址	0x8C0	R/W
uhloop0_end_addr	循环 0 的 32 位结束地址	0x8C1	R/W
uhloop0_count	硬件循环 0 的循环计数	0x8C2	R/W
uhloop1_start_addr	循环 1 的 32 位起始地址	0x8C3	R/W
uhloop1_end_addr	循环 1 的 32 位结束地址	0x8C4	R/W
uhloop1_count	硬件循环 1 的循环计数	0x8C5	R/W
uhwloop_state_reg	用户硬件循环状态	0x8C6	R/W
总线错误异常上下文 CSR（自定义）			
ldpc0	加载总线错误 PC 寄存器 0	0xBE0	R/W
ldpc1	加载总线错误 PC 寄存器 1	0xBE1	R/W
ldtval0	加载总线错误访问地址寄存器 0	0xBE8	R/W
ldtval1	加载总线错误访问地址寄存器 1	0xBE9	R/W
stpc0	存储总线错误 PC 寄存器 0	0xBF0	R/W
stpc1	存储总线错误 PC 寄存器 1	0xBF1	R/W
stpc2	存储总线错误 PC 寄存器 2	0xBF2	R/W
sttval0	存储总线错误访问地址寄存器 0	0xBF8	R/W
sttval1	存储总线错误访问地址寄存器 1	0xBF9	R/W
sttval2	存储总线错误访问地址寄存器 2	0xBFA	R/W

请注意，如果对上表中只读属性的任何 CSR 尝试执行写入/置位/清除操作，CPU 将生成非法指令异常。

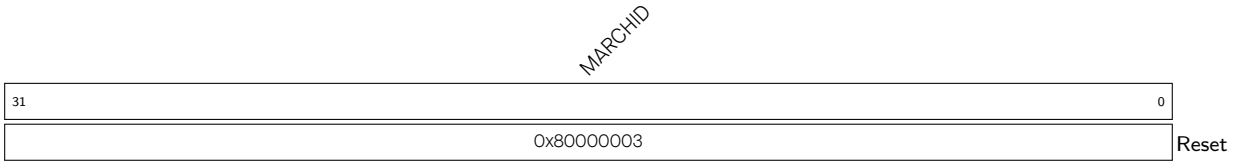
2.5.2 寄存器描述

Register 2.1. mvendorid (0xF11)



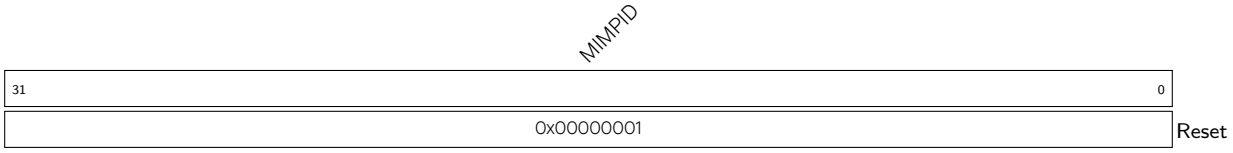
MVENDORID 表示供应商 ID。(RO)

Register 2.2. marchid (0xF12)



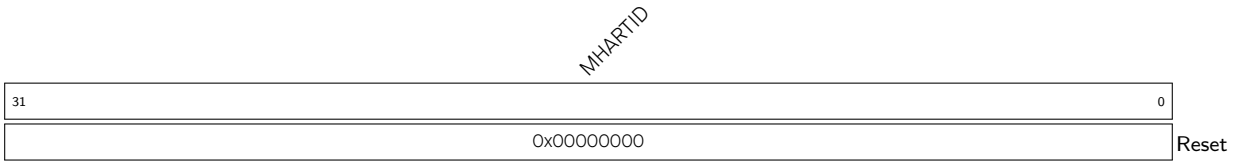
MARCHID 表示架构 ID。(RO)

Register 2.3. mimpid (0xF13)



MIMPID 表示实现 ID。(RO)

Register 2.4. mhartid (0xF14)



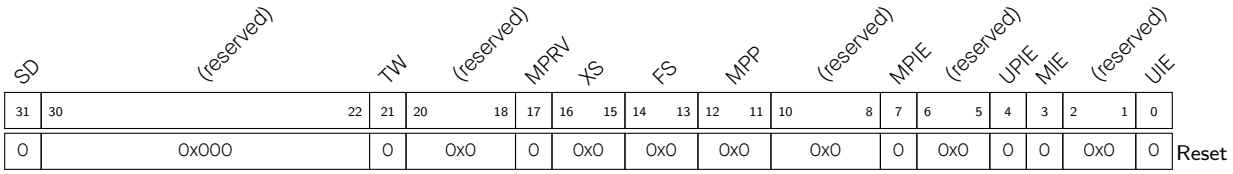
MHARTID 表示 hart ID。

0：hart 0

1：hart 1

(RO)

Register 2.5. mstatus (0x300)



SD	(reserved)	TW	(reserved)	MPRV	XS	FS	MPP	(reserved)	MPIE	(reserved)	UPIE	MIE	(reserved)	UIE	
31	30 22	21	20 18	17	16 15	14 13	12 11	10 8	7	6 5	4	3	2 1	0	
0	0x000	0	0x0	0	0x0	0x0	0x0	0x0	0	0x0	0	0	0x0	0	Reset

SD 当 FS 或 XS 域表示存在脏状态并需要将扩展上下文保存到存储器时，自动设置。(RO)

TW 配置 WFI（等待中断）指令是否可以在较低特权级模式下执行。

0：WFI 可在较低特权级模式下执行。

1：WFI 仅可在机器模式下执行，如果在较低特权级模式下执行，将触发非法指令异常。

(R/W)

MPRV 配置是否将 mstatus.MPP 作为有效的特权模式，用于执行加载和存储，而不是 CPU 实际执行的特权模式。

0：不作为

1：作为

注意，此位不影响指令保护。

(R/W)

XS 表示 HWLP 和 PIE 自定义扩展的状态。

0x0：所有扩展关闭

0x1：没有扩展处于脏 (dirty) 或干净 (clean) 状态，一些扩展已被启用

0x2：没有扩展处于脏状态，一些扩展处于干净状态

0x3：一些扩展处于脏状态

注意，要修改这些扩展的状态，必须使用相应的 mext_hwlp_status 和 mext_pie_status CSR。

(RO)

FS 表示浮点单元的状态，包括浮点寄存器 f0–f31 和 fcsr、frm 和 fflags CSR。

0x0：关闭

0x1：初始

0x2：干净

0x3：脏

(R/W)

MPP 配置机器之前的特权模式（陷阱前）。

0x0：用户模式

0x3：机器模式

注意：只有低位可写。对高位的任何写入都将被忽略，因为它直接与低位相连。

(R/W)

见下页……

Register 2.5. mstatus (0x300)

接上页……

MPIE 配置是否使能（陷阱）之前的机器中断。

0：禁用

1：使能

(R/W)

UPIE 配置是否使能（陷阱）之前的用户中断。

0：禁用

1：使能

(R/W)

MIE 配置是否使能全局机器中断。

0：禁用

1：使能

(R/W)

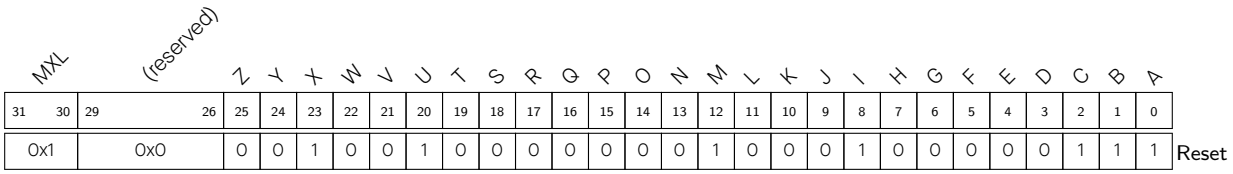
UIE 配置是否使能全局用户中断。

0：禁用

1：使能

(R/W)

Register 2.6. misa (0x301)



MXL	(reserved)	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
31 30	29 26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1	0x0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	1	1

Reset

MXL 机器 XLEN = 1（32 位）。(RO)

Z 保留 = 0。(RO)

Y 保留 = 0。(RO)

X 非标准扩展 = 1。(RO)

W 保留 = 0。(RO)

V 保留 = 0。(RO)

U 实现用户模式 = 1。(RO)

T 保留 = 0。(RO)

S 实现监督模式 = 0。(RO)

R 保留 = 0。(RO)

Q 四精度浮点扩展 = 0。(RO)

P 保留 = 0。(RO)

O 保留 = 0。(RO)

N 支持用户级别中断 = 0。(RO)

M 整数乘除法标准扩展 = 1。(RO)

L 保留 = 0。(RO)

K 保留 = 0。(RO)

J 保留 = 0。(RO)

I RV32I 基本 ISA = 1。(RO)

H 虚拟机管理程序扩展 = 0。(RO)

G 其他标准扩展 = 0。(RO)

F 单精度浮点扩展 = 0。(RO)

E RV32E 基本 ISA = 0。(RO)

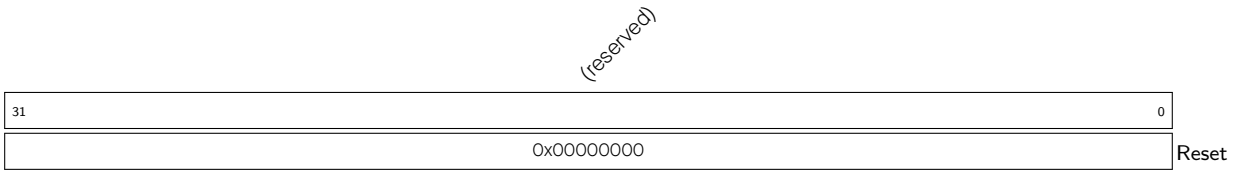
D 双精度浮点扩展 = 0。(RO)

C 压缩标准扩展 = 1。(RO)

B 位操作 = 1。(RO)

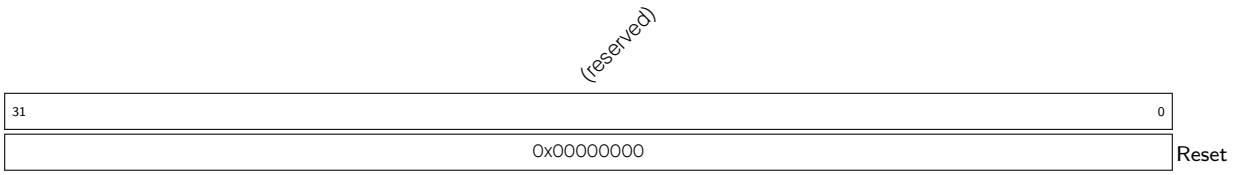
A 原子标准扩展 = 1。(RO)

Register 2.7. mideleg (0x303)



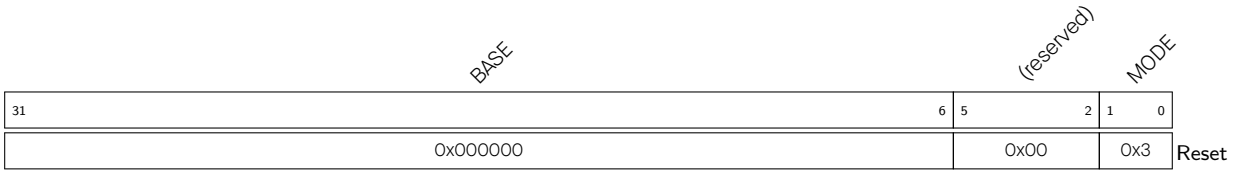
mideleg 所有位都硬连线为 0，因为仅支持 CLIC 模式。(RO)

Register 2.8. mie (0x304)



mie 所有位都硬连线为 0，因为仅支持 CLIC 模式。(RO)

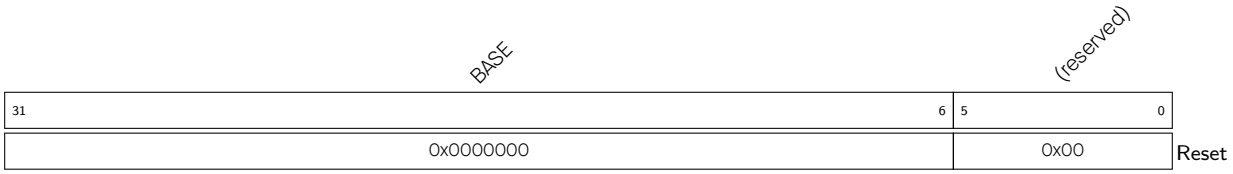
Register 2.9. mtvec (0x305)



BASE 配置异常和非向量机器模式中断基地址的高 26 位，地址 64 字节对齐。(R/W)

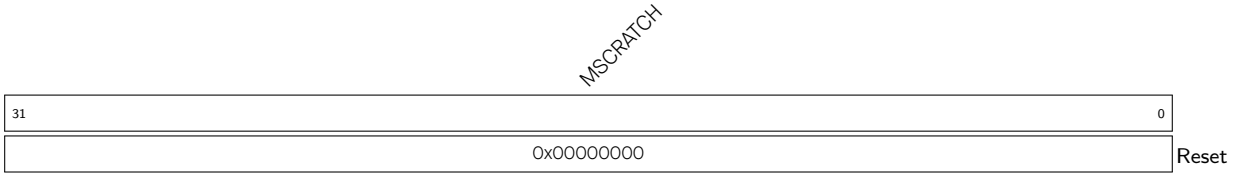
MODE 表示机器模式中断在 CLIC 模式还是向量/非向量 CLINT 模式运行。仅支持 CLIC 模式 **0x3**。(RO)

Register 2.10. mtvt (0x307)



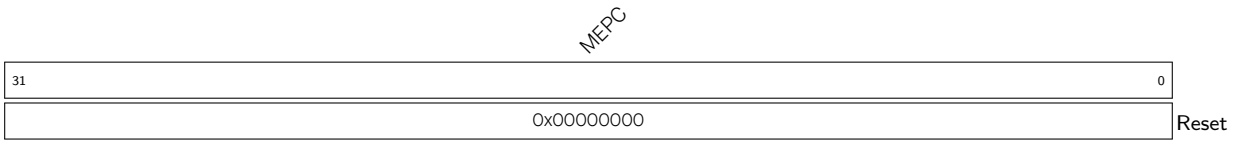
BASE 配置 CLIC 机器模式中断向量基地址的高 26 位，地址 64 字节对齐。(R/W)

Register 2.11. mscratch (0x340)



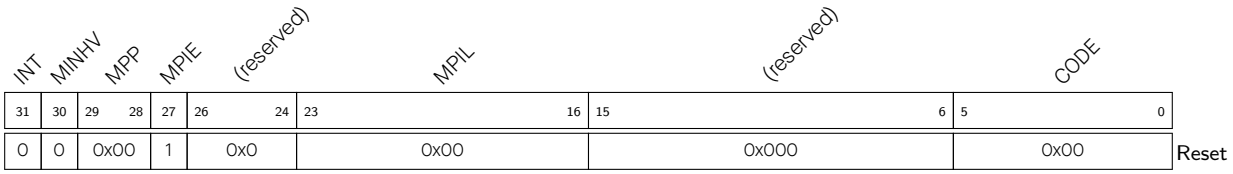
MSCRATCH 配置机器自定义的暂存信息。(R/W)

Register 2.12. mepc (0x341)



MEPC 配置机器陷阱/异常程序计数器。当 CPU 遇到异常时，此域将自动更新为 CPU 将要执行的指令的地址。(R/W)

Register 2.13. mcause (0x342)



INT 表示 CPU 是否因中断进入陷阱。

0：因异常进入陷阱

1：因中断进入陷阱

(R/W)

MINHV 表示 mepc 是指令地址还是向量表表项地址。

1：向量表表项地址

0：指令地址

(R/W)

MPP 表示之前的特权模式，与 mstatus.MPP 相同。(R/W)

MPIE 表示之前的中断使能，与 mstatus.MPIE 相同。(R/W)

MPIL 表示之前的 8 位中断级别。(R/W)

CODE CPU 进入异常时，此域将自动更新为最近的异常或中断的唯一 ID。可能的异常 ID：

0x01：指令访问错误

0x02：非法指令

0x03：硬件断点/观察点或 EBREAK

0x05：读存储器访问错误

0x06：写存储器地址或 AMO 地址未对齐

0x07：写存储器访问错误

0x08：用户模式环境调用 (ECALL)

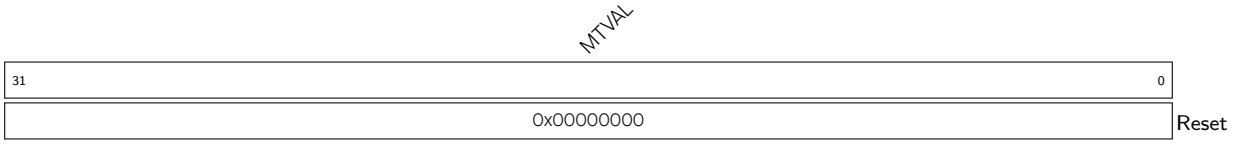
0x0b：机器模式环境调用

0x1f：PIE 指令非法其他值：保留

注意：异常 ID 0x0（指令地址非对齐）不存在，因为 CPU 在取指时始终屏蔽地址的最低位。

(R/W)

Register 2.14. mtval (0x343)



MTVAL 表示机器陷阱值。将自动更新为与异常有关的数据，该数据可能有助于处理该异常。

根据异常编号有以下解读：

0x1：指令虚拟地址错误

0x2：指令操作码错误

0x5：存储器读操作的数据地址错误

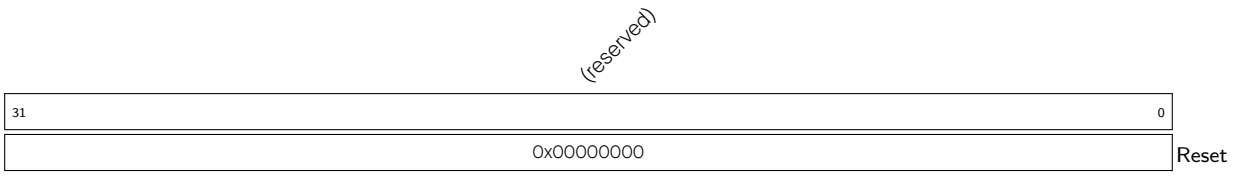
0x7：存储器写操作的数据地址错误

0x1F：PIE 非法指令操作码错误

注意：该寄存器不支持其他异常 ID 和中断。

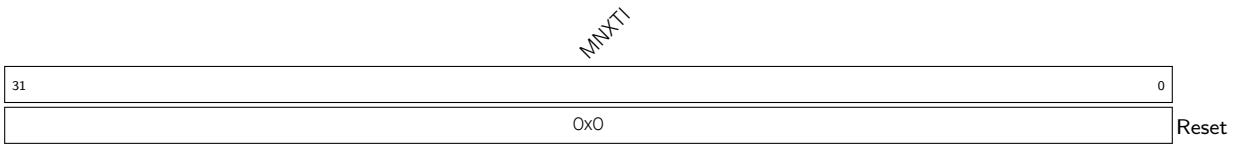
(R/W)

Register 2.15. mip (0x344)



所有位都硬连线为 0，因为仅支持 CLIC 模式。(RO)

Register 2.16. mnxti (0x345)



MNXTI 表示下一个机器模式中断入口地址，并配置中断使能状态。

软件可以使用该 CSR 处理下一个特权模式相同的水平中断，但该中断级别必须高于保存在 mcause.MPIL 中的中断上下文，也须高于相应特权模式的中断阈值，如此便减少了刷新中断流水线和保存/恢复上下文的开销。

此 CSR 使用 CSRRSI/CSRRCI 指令访问，读取的值是陷阱处理程序表中表项的指针，写回的是中断使能状态。此外，写入 mnxti 会产生副作用：更新中断上下文状态。

使用 CSRR 读取 mnxti CSR 时，返回零表示没有合适的中断可处理或系统不在 CLIC 模式；返回陷阱处理程序表中表项的非零地址，用于软件陷阱向量处理。

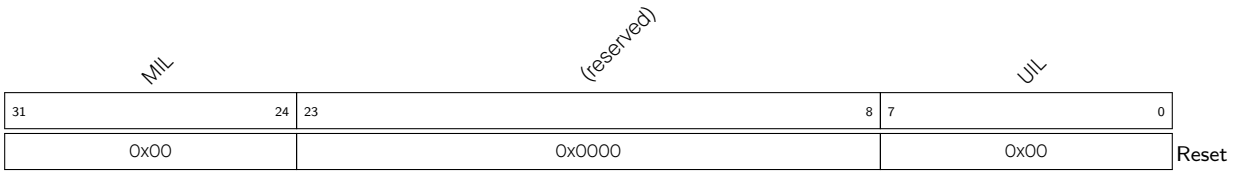
如果访问 mnxti 的 CSR 指令包含写操作，则用于读-改-写操作的 CSR 是 mstatus CSR，而 mcause 寄存器的 exccode 域和 mintstatus.MIL 域也可以更新为新的中断 ID 和级别。如果中断是边沿触发，则挂起位也会被清零。

当初始中断被接受 (taken) 后，mcause 和 mepc 寄存器会更新为中断上下文和中断 ID，此时，中断处理程序会使用 mnxti CSR。

如果挂起的中断是边沿触发的，当访问 mnxti 的 CSR 指令包含写操作时，硬件将自动清除相应的挂起位。然而，如果 CSR 指令不包含写操作，则不会对任何 CSR 进行状态更新，因此中断挂起位不会被清零。这种行为允许软件使用 mnxti 优化中断的选择和执行。

(R/W)

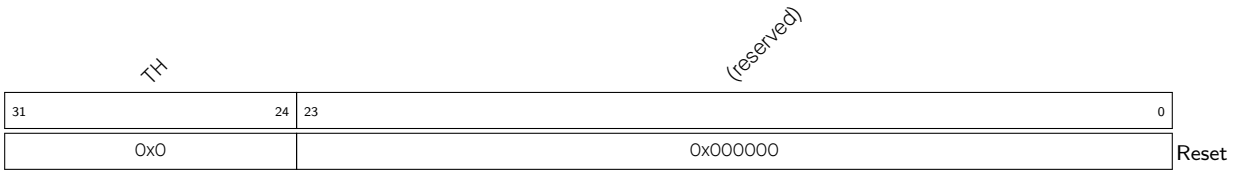
Register 2.17. mintstatus (0xFB1)



MIL 表示当前机器模式中断有效的 8 位中断级别。(RO)

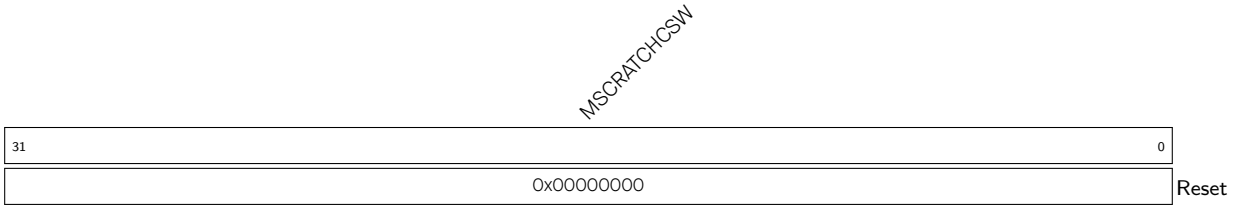
UIL 硬连线为 0x0，因为不支持用户模式中断。(RO)

Register 2.18. mintthresh (0x347)



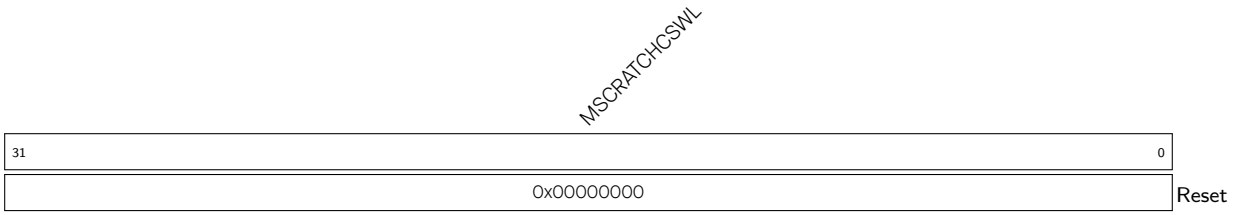
TH 配置机器模式中断的 8 位级别阈值。
注意，机器模式中断的有效阈值级别是 mintthresh.TH 和 mintstatus.MIL 的较大值。如果机器模式挂起中断的级别小于或等于有效阈值级别，则无法抢占执行。
(R/W)

Register 2.19. mscratchcsw (0x348)



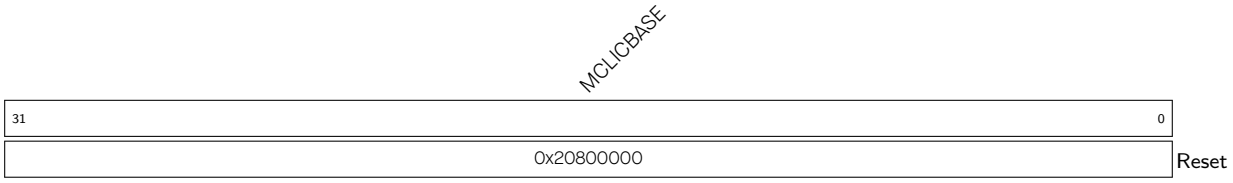
MSCRATCHCSW 配置 MSCRATCH 的值，MSCRATCH 的值取决于当前特权模式和 MPP 中之前的特权模式是否不同，如果不同，交换其 MSCRATCH 和 RS1 的值。
MSCRATCHCSW 是条件暂存交换 CSR，可以在单条指令中根据特权模式是否变化来决定是否交换暂存值。
使用 CSRRW 指令访问此 MSCRATCHCSW 时，如果 MPP 与当前特权模式不同，将 MSCRATCH 的值写入 RD，将 RS1 原来的值写入 MSCRATCH；如果相同，将 RS1 的值写入 RD。
(R/W)

Register 2.20. mscratchcswl (0x349)



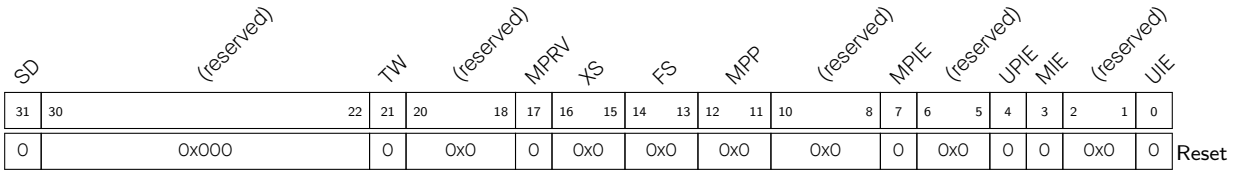
MSCRATCHCSWL 配置 MSCRATCH 的值，MSCRATCH 的值取决于 mintstatus.MIL 中的当前中断级别和 mcause.MPIL 中的之前中断级别是否不同，如果不同，交换 MSCRATCH 与 RS1 的值。
MSCRATCHCSWL 是条件暂存交换 CSR，可以在单条指令中根据中断级别是否变化来决定是否交换暂存值。
使用 CSRRW 指令访问 MSCRATCHCSWL 时，如果 mcause.MPIL 和 mintstatus.MIL 中只有一个等于 0，将 MSCRATCH 的值写入 RD，将 RS1 原来的值写入 MSCRATCH；如果是其他情况，将 RS1 写入 RD。
(R/W)

Register 2.21. mclicbase (0x350)



MCLICBASE 表示机器模式下 CLIC 存储器映射寄存器的基地址。(RO)

Register 2.22. ustatus (0x000)



SD 当 FS 或 XS 域表示存在脏状态并需要将扩展上下文保存到存储器时，自动设置。(RO)

TW 表示 WFI（等待中断）指令是否可以在较低特权级模式下执行。

0：WFI 可在较低特权级模式下执行。

1：WFI 仅可在机器模式下执行，如果在较低特权级模式下执行，将触发非法指令异常。

(RO)

MPRV 配置是否将 ustatus.MPP 作为有效的特权模式，用于执行加载和存储，而不是 CPU 实际执行的特权模式。

0：不作为

1：作为

注意，此位不影响指令保护。

(RO)

XS 表示 HWLP 和 PIE 自定义扩展的状态。

0x0：所有扩展关闭

0x1：没有扩展处于脏 (dirty) 或干净 (clean) 状态，一些扩展已被启用

0x2：没有扩展处于脏状态，一些扩展处于干净状态

0x3：一些扩展处于脏状态

注意，要修改这些扩展的状态，必须使用相应的 mext_hwlp_status 和 mext_pie_status CSR。

(RO)

FS 配置浮点单元的状态，包括浮点寄存器 f0–f31 和 fcsr、frm 和 fflags CSR。

0x0：关闭

0x1：初始

0x2：干净

0x3：脏

(R/W)

MPP 表示机器之前的特权模式（陷阱前）。

0x0：用户模式

0x3：机器模式

注意：只有低位可写。对高位的任何写入都将被忽略，因为它直接与低位相连。

(RO)

见下页……

Register 2.22. ustatus (0x000)

接上页……

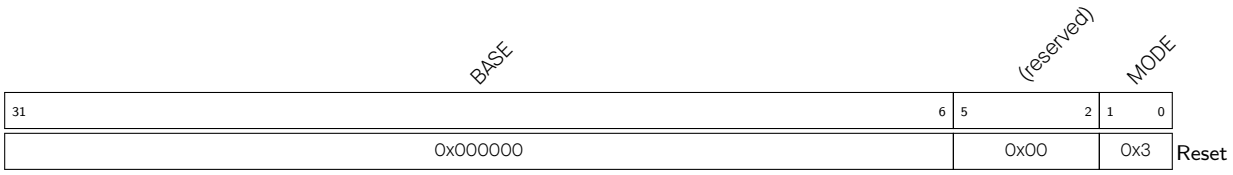
MPIE 表示（陷阱）之前机器中断使能状态。
0：禁用
1：使能
(RO)

UPIE 表示（陷阱）之前用户中断使能状态。
0：禁用
1：使能
(RO)

MIE 表示全局机器模式中断使能状态。
0：禁用
1：使能
(RO)

UIE 表示全局用户模式中断使能状态。
0：禁用
1：使能
(RO)

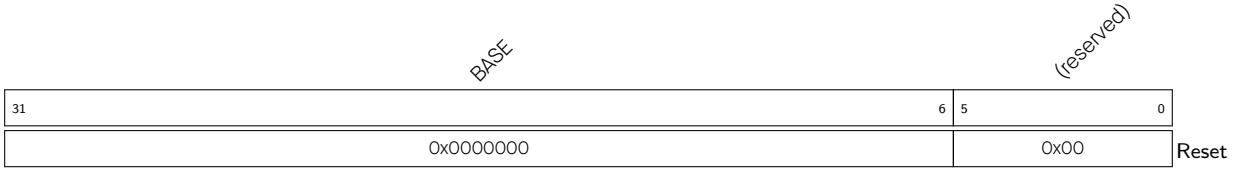
Register 2.23. utvec (0x005)



BASE 配置异常和非向量用户模式中断基地址的高 26 位，地址 64 字节对齐。(R/W)

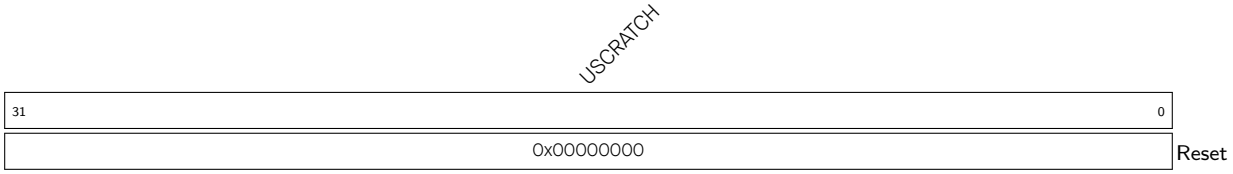
MODE 表示用户模式中断在 CLIC 模式还是向量/非向量 CLINT 模式运行。仅支持 CLIC 模式 **0x3**。
(RO)

Register 2.24. utvt (0x007)



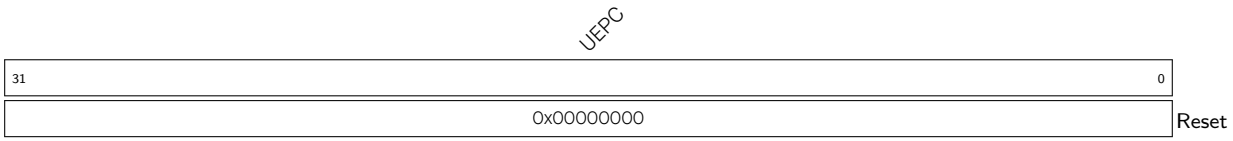
BASE 配置 CLIC 用户模式中断向量基地址的高 26 位，地址 64 字节对齐。(R/W)

Register 2.25. uscratch (0x040)



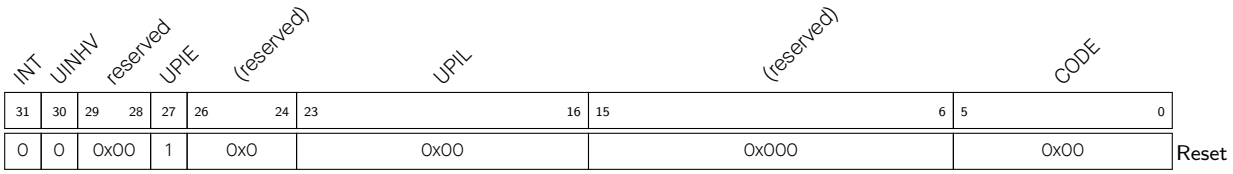
USCRATCH 配置用户自定义的暂存信息。(R/W)

Register 2.26. uepc (0x041)



UEPC 配置用户陷阱/异常程序计数器。当 CPU 遇到异常时，此域将自动更新为 CPU 将要执行的指令的地址。(R/W)

Register 2.27. ucause (0x042)



INT 表示 CPU 是否因中断进入陷阱。

0：因异常进入陷阱

1：因中断进入陷阱

(R/W)

UINHV 表示 uepc 是指令地址还是向量表表项地址。

1：向量表表项地址

0：指令地址

(R/W)

UPIE 表示之前的中断使能，与 ustatus.UPIE 相同。(R/W)

UPIL 表示之前的 8 位中断级别。(R/W)

CODE CPU 进入异常时，此域将自动更新为最近的异常或中断的唯一 ID。可能的异常 ID：

0x01：指令访问错误

0x02：非法指令

0x03：硬件断点/观察点或 EBREAK

0x05：读存储器访问错误

0x06：写存储器地址或 AMO 地址未对齐

0x07：写存储器访问错误

0x08：用户模式环境调用 (ECALL)

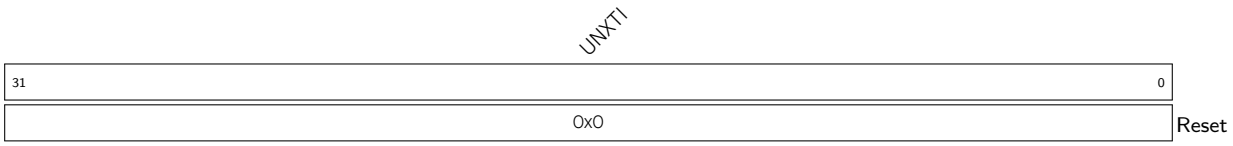
0x0b：机器模式环境调用

0x1f：PIE 指令非法其他值：保留

注意：异常 ID 0x0（指令地址非对齐）不存在，因为 CPU 在取指时始终屏蔽地址的最低位。

(R/W)

Register 2.28. unxti (0x045)



UNXTI 表示下一个用户模式中断入口地址，并配置中断使能状态。

软件可以使用该 CSR 处理下一个特权模式相同的水平中断，但该中断级别必须高于保存在 ucause.UPIL 中的中断上下文，也须高于相应特权模式的中断阈值，如此便减少了刷新中断流水线和保存/恢复上下文的开销。

此 CSR 使用 CSRRSI/CSRRCI 指令访问，读取的值是陷阱处理程序表中表项的指针，写回的是中断使能状态。此外，写入 unxti 会产生副作用：更新中断上下文状态。

使用 CSRR 读取 unxti CSR 时，返回零表示没有合适的中断可处理或系统不在 CLIC 模式；返回陷阱处理程序表中表项的非零地址，用于软件陷阱向量处理。

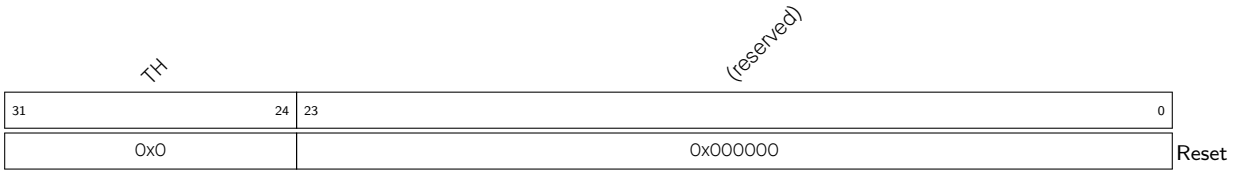
如果访问 unxti 的 CSR 指令包含写操作，则用于读-改-写操作的 CSR 是 ustatus CSR，而 ucause 寄存器的 exccode 域和 uintstatus.UIL 域也可以更新为新的中断 ID 和级别。如果中断是边沿触发，则挂起位也会被清零。

当初始中断被接受 (taken) 后，ucause 和 uepc 寄存器会更新为中断上下文和中断 ID，此时，中断处理程序会使用 unxti CSR。

如果挂起的中断是边沿触发的，当访问 unxti 的 CSR 指令包含写操作时，硬件将自动清除相应的挂起位。然而，如果 CSR 指令不包含写操作，则不会对任何 CSR 进行状态更新，因此中断挂起位不会被清零。这种行为允许软件使用 unxti 优化中断的选择和执行。

(R/W)

Register 2.29. uintthresh (0x047)

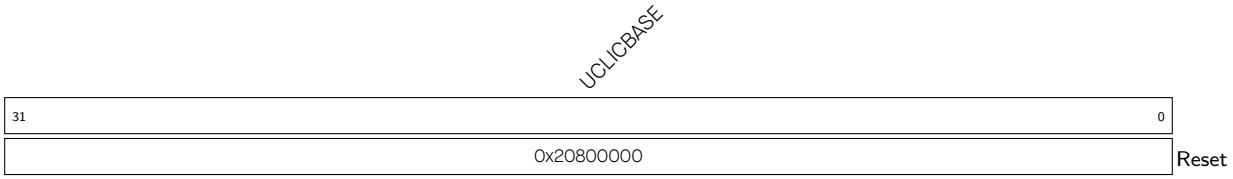


TH 配置用户模式中断的 8 位级别阈值。

请注意，用户模式中断的有效阈值级别是 uintthresh.TH 和 uintstatus.UIL 的较大值。若用户模式挂起的中断级别小于或等于有效阈值级别，则不允许抢占执行。

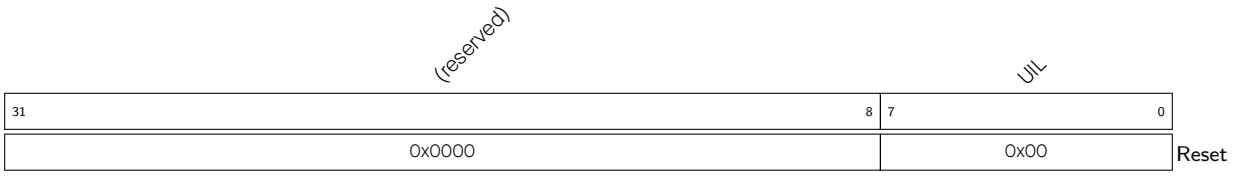
(R/W)

Register 2.30. uclicbase (0x050)



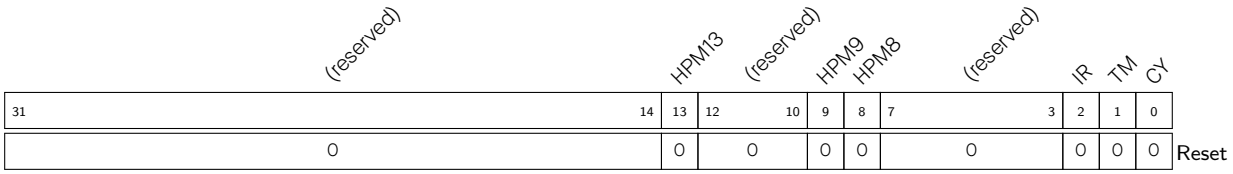
UCLICBASE 表示用户模式下 CLIC 存储器映射寄存器的基地址。(RO)

Register 2.31. uintstatus (0xCB1)



UIL 表示当前用户模式中断有效的 8 位中断级别。(RO)

Register 2.32. mcounteren (0x306)



HPM13 配置是否从用户模式访问 hpmcounter13。

0：从用户模式访问 hpmcounter13 会导致非法指令异常

1：从用户模式访问有效

(R/W)

HPM9 配置是否从用户模式访问 hpmcounter9。

0：从用户模式访问 hpmcounter9 会导致非法指令异常

1：从用户模式访问有效

(R/W)

HPM8 配置是否从用户模式访问 hpmcounter8。

0：从用户模式访问 hpmcounter8 会导致非法指令异常

1：从用户模式访问有效

(R/W)

CY 配置从用户模式访问 cycle 计数器。

0：在用户模式下访问 cycle CSR 将生成非法指令异常

1：在用户模式下访问 cycle CSR 是有效的

(R/W)

TM 配置从用户模式访问 time 计数器。

0：在用户模式下访问 time CSR 将生成非法指令异常

1：在用户模式下访问 time CSR 是有效的

(R/W)

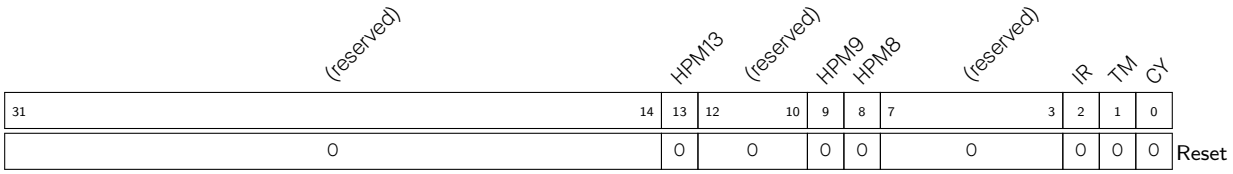
IR 配置从用户模式访问 instret 计数器。

0：在用户模式下访问 instret CSR 将生成非法指令异常

1：在用户模式下访问 instret CSR 是有效的

(R/W)

Register 2.33. mcountinhibit (0x320)



HPM13 配置 mhpmcounter13 的递增。

0：继续递增 mhpmcounter13

1：停止递增 mhpmcounter13

(R/W)

HPM9 配置 mhpmcounter9 的递增。

0：继续递增 mhpmcounter9

1：停止递增 mhpmcounter9

(R/W)

HPM8 配置 mhpmcounter8 的递增。

0：继续递增 mhpmcounter8

1：停止递增 mhpmcounter8

(R/W)

IR 配置 instret 计数器的递增。

0：继续递增 instret 计数器

1：停止递增 instret 计数器

(R/W)

TM 配置 time 计数器的递增。

0：继续递增 time 计数器

1：停止递增 time 计数器

(R/W)

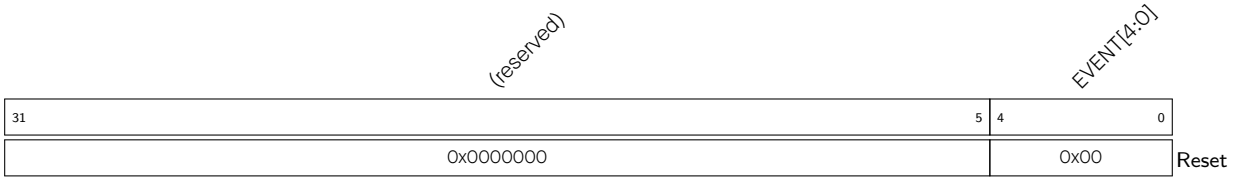
CY 配置 cycle 计数器的递增。

0：继续递增 cycle 计数器

1：停止递增 cycle 计数器

(R/W)

Register 2.34. mhpmevent8 (0x328)

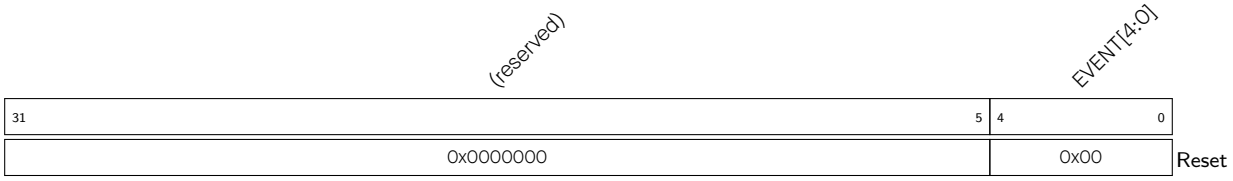


EVENT[4:0] mhpmcounter8 的事件选择器。

此计数器的唯一有效事件值为 0x6，用于条件分支预测错误。

(R/W)

Register 2.35. mhpmevent9 (0x329)

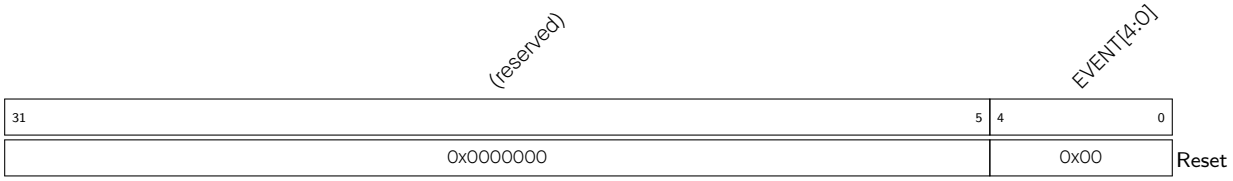


EVENT[4:0] mhpmcounter9 的事件选择器。

此计数器的唯一有效事件值为 0x7，用于条件分支指令。

(R/W)

Register 2.36. mhpmevent13 (0x32D)

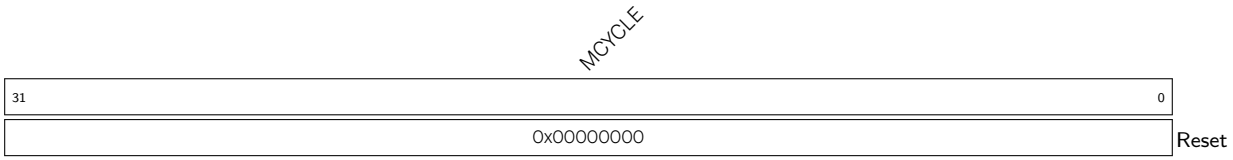


EVENT[4:0] mhpmcounter13 的事件选择器。

此计数器的唯一有效事件值为 0xB，用于存储指令。

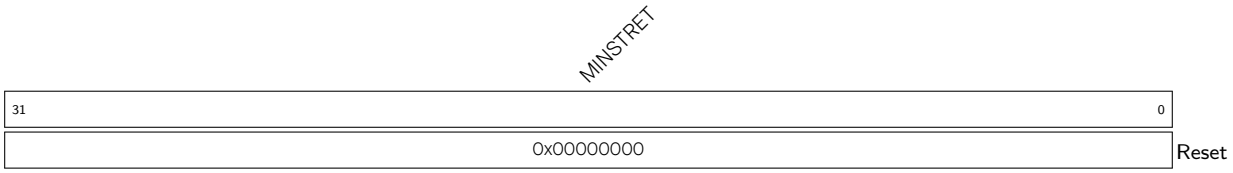
(R/W)

Register 2.37. mcycle (0xB00)



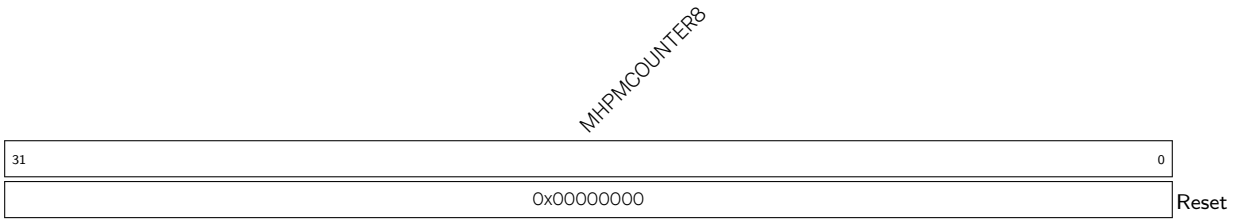
MCYCLE 表示机器模式周期计数器的低 32 位。(R/W)

Register 2.38. minstret (0xB02)



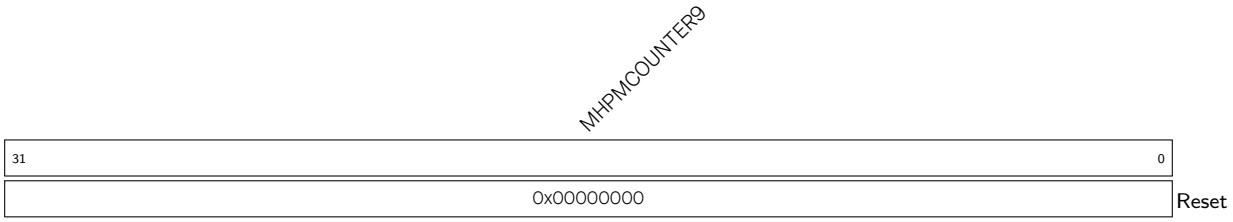
MINSTRET 表示退役机器指令计数器的低 32 位。(R/W)

Register 2.39. mhpmcounter8 (0xB08)



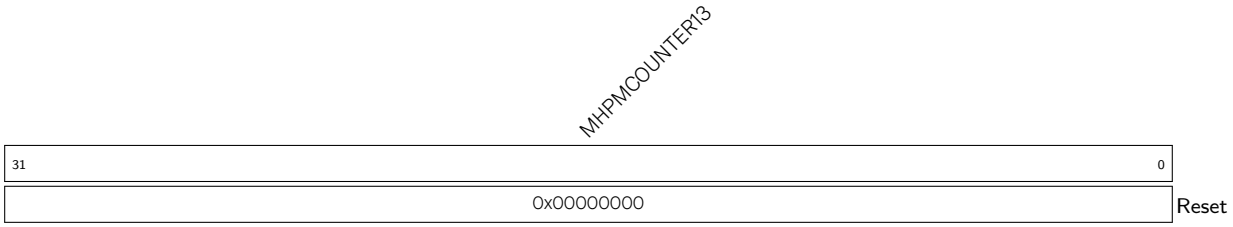
MHPMCOUNTER8 表示机器性能监控计数器 8 的低 32 位。(R/W)

Register 2.40. mhpmcounter9 (0xB09)



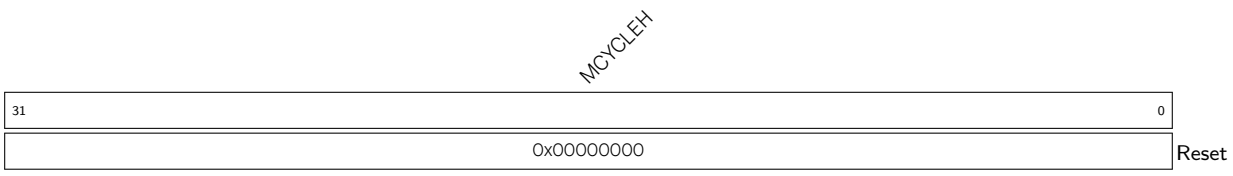
MHPMCOUNTER9 表示机器性能监控计数器 9 的低 32 位。(R/W)

Register 2.41. mhpmcounter13 (0xB0D)



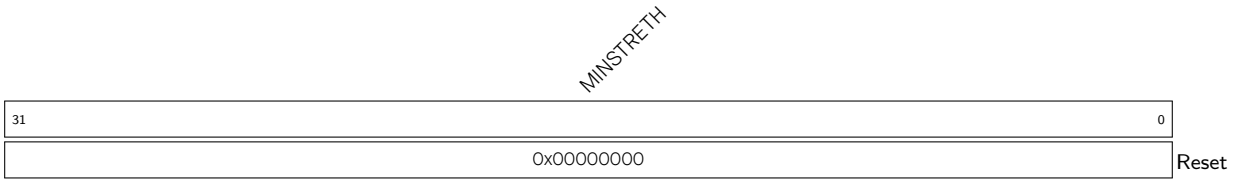
MHPMCOUNTER13 表示机器性能监控计数器 13 的低 32 位。(R/W)

Register 2.42. mcycleh (0xB80)



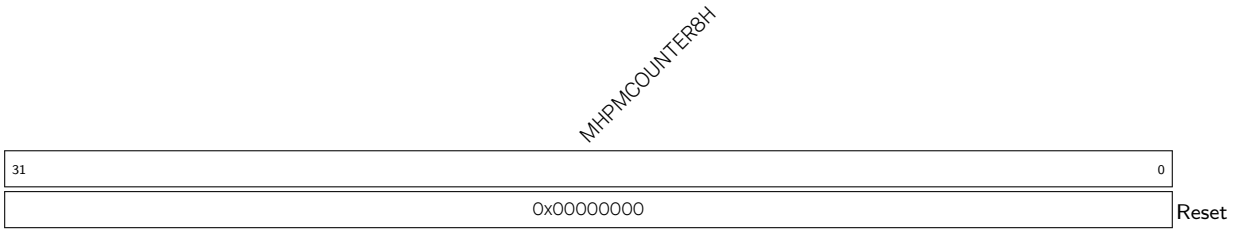
MCYCLEH 表示 mcycle 计数器的高 32 位。(R/W)

Register 2.43. minstreth (0xB82)



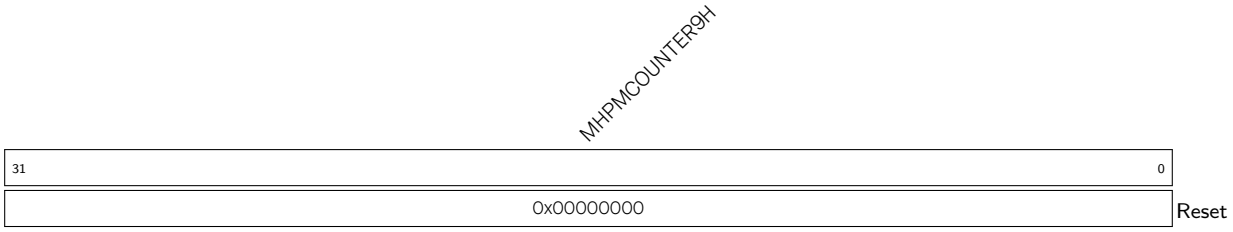
MINSTRETH 表示 minstret 计数器的高 32 位。(R/W)

Register 2.44. mhpmcounter8h (0xB88)



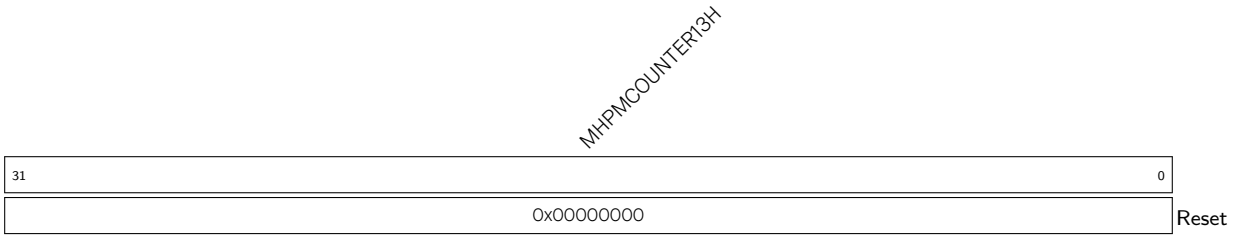
MHPMCOUNTER8H 表示 mhpmcounter8 计数器的高 32 位。(R/W)

Register 2.45. mhpmcounter9h (0xB89)



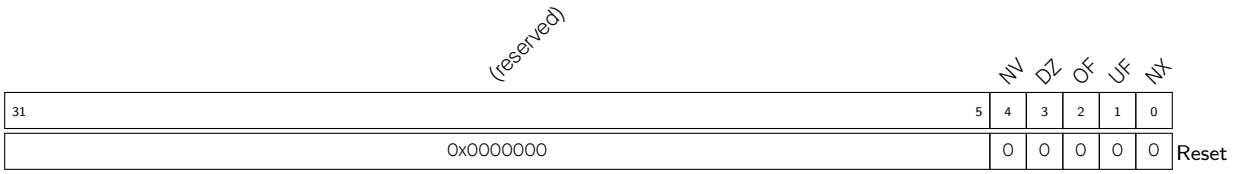
MHPMCOUNTER9H 表示 mhpmcounter9 计数器的高 32 位。(R/W)

Register 2.46. mhpmcounter13h (0xB8D)



MHPMCOUNTER13H 表示 mhpmcounter13 计数器的高 32 位。(R/W)

Register 2.47. fflags (0x001)



NV 表示浮点无效操作标志。(R/W)

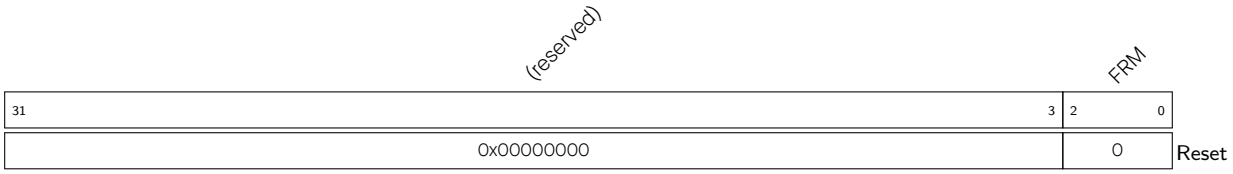
DZ 表示浮点除以零标志。(R/W)

OF 表示浮点上溢标志。(R/W)

UF 表示浮点下溢标志。(R/W)

NX 表示浮点不精确标志。(R/W)

Register 2.48. frm (0x002)



FRM 配置浮点舍入模式。

有效的舍入模式值:

0x0：RNE（舍入到最近，偶数优先）

0x1：RTZ（向零舍入）

0x2：RDN（向负无穷大舍入）

0x3：RUP（向正无穷大舍入）

0x4：RMM（舍入到最近，绝对值最大优先）

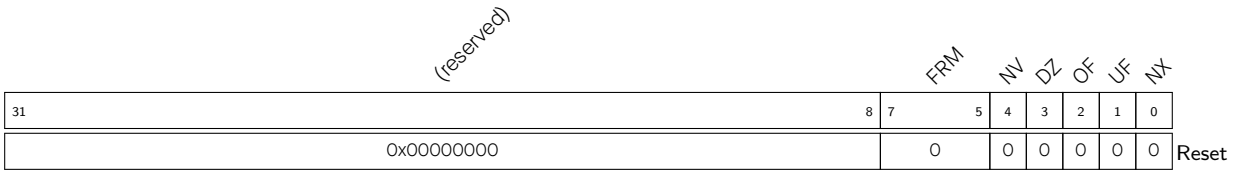
0x5：保留（无效，预留未来使用）

0x6：保留（无效，预留未来使用）

0x7：DYN（在指令的 rm 字段中，选择动态舍入模式；在舍入模式寄存器中无效）

(R/W)

Register 2.49. fcsr (0x003)



FRM 配置浮点舍入模式。

有效的舍入模式值:

0x0：RNE（舍入到最近，偶数优先）

0x1：RTZ（向零舍入）

0x2：RDN（向负无穷大舍入）

0x3：RUP（向正无穷大舍入）

0x4：RMM（舍入到最近，绝对值最大优先）

0x5：保留（无效，预留未来使用）

0x6：保留（无效，预留未来使用）

0x7：DYN（在指令的 rm 字段中，选择动态舍入模式；在舍入模式寄存器中无效）

(R/W)

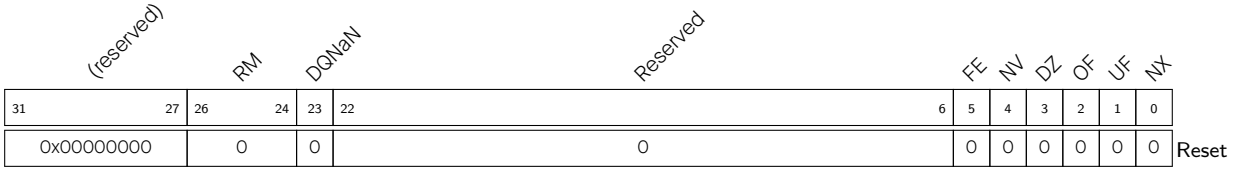
DZ 表示浮点除零标志。(R/W)

OF 表示浮点上溢标志。(R/W)

UF 表示浮点下溢标志。(R/W)

NX 表示浮点不精确标志。(R/W)

Register 2.50. fxsr (0x800)



RM 映射到 frm 位。对该字段的写操作将同时反映到 fcsr 寄存器。(R/W)

DQNaN 配置 QNaN 输出的计算值。

0：QNaN 输出的计算值为 RISC-V 指定的固定值，即 0x7FC00000

1：QNaN 输出的计算值与 IEEE 754 标准一致

(R/W)

FE 浮点异常累计位。映射到 FE 位。对该字段的写操作将同时反映到 fcsr 寄存器。

0：未发生浮点异常

1：发生浮点异常

(R/W)

NV 表示无效操作数异常标志。映射到 NV 位。对该字段的写操作将同时反映到 fcsr 寄存器。(R/W)

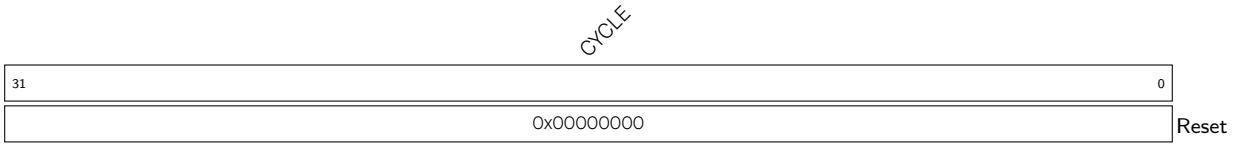
DZ 表示被零除异常标志。映射到 DZ 位。对该字段的写操作将同时反映到 fcsr 寄存器。(R/W)

OF 表示溢出异常标志。映射到 OF 位。对该字段的写操作将同时反映到 fcsr 寄存器。(R/W)

UF 表示下溢异常标志。映射到 UF 位。对该字段的写操作将同时反映到 fcsr 寄存器。(R/W)

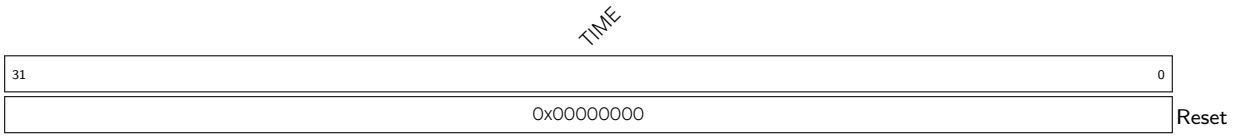
NX 表示不精确异常标志。映射到 NX 位。对该字段的写操作将同时反映到 fcsr 寄存器。(R/W)

Register 2.51. cycle (0xC00)



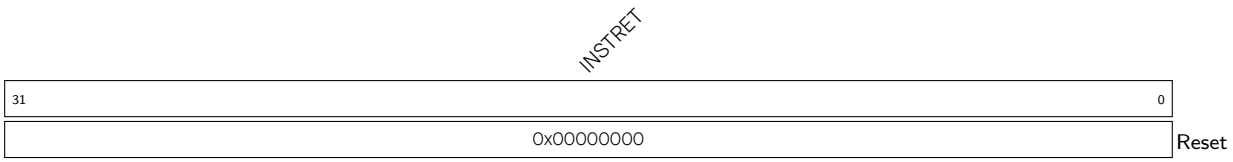
CYCLE mcycle 的只读镜像。(RO)

Register 2.52. time (0xC01)



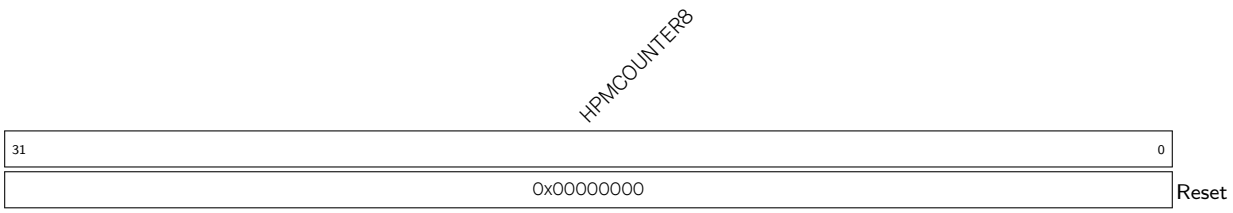
TIME 表示 RDTIME 指令的定时器值。(RO)

Register 2.53. instret (0xC02)



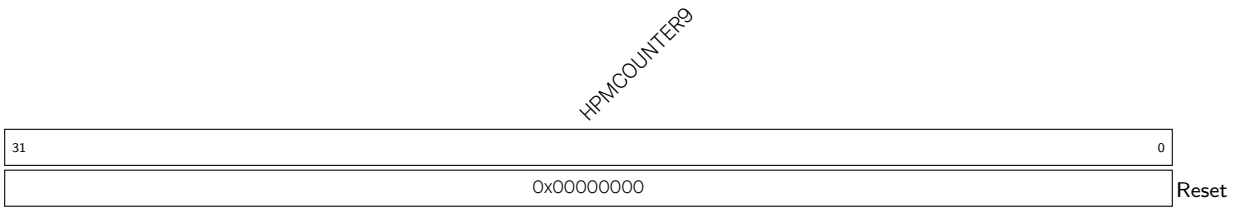
INSTRET minstret 的只读镜像。(RO)

Register 2.54. hpmcounter8 (0xC08)



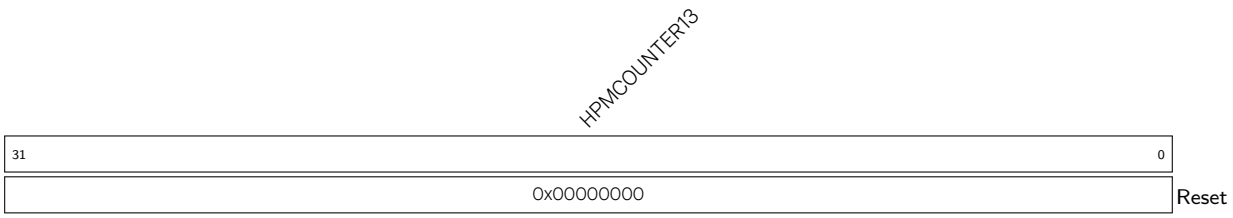
HPMCOUNTER8 mhpmcounter8 的只读镜像。(RO)

Register 2.55. hpmcounter9 (0xC09)



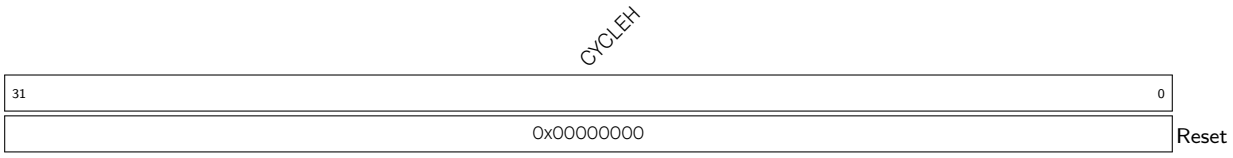
HPMCOUNTER9 mhpmcounter9 的只读镜像。(RO)

Register 2.56. hpmcounter13 (0xC0D)



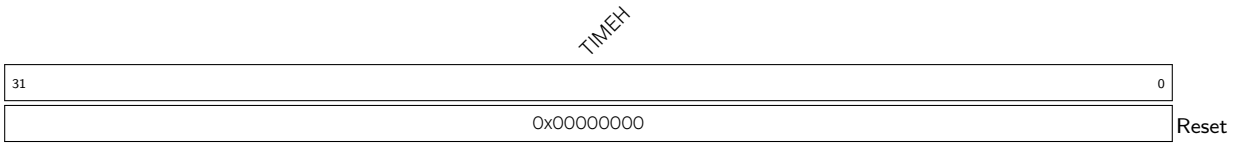
HPMCOUNTER13 mhpmcounter13 的只读镜像。(RO)

Register 2.57. cycleh (0xC80)



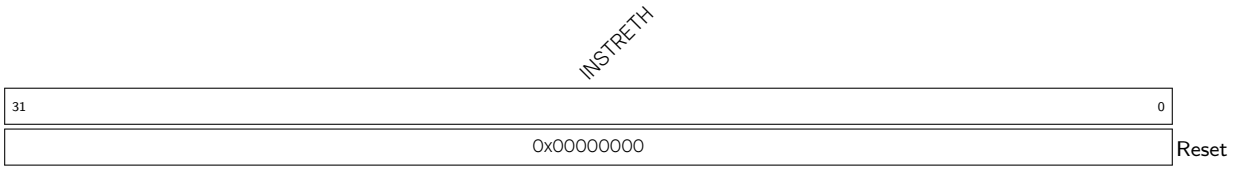
CYCLEH mcycleh 的只读镜像。(RO)

Register 2.58. timeh (0xC81)



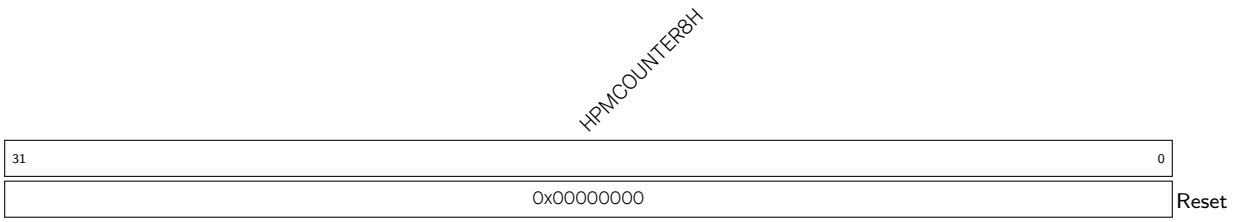
TIMEH 表示 RDTIME 指令定时器的高 32 位。(RO)

Register 2.59. instreth (0xC82)



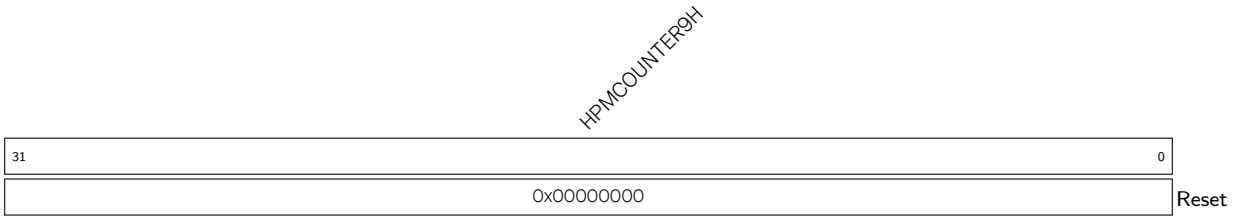
INSTRETH minstreth 的只读镜像。(RO)

Register 2.60. hpmcounter8h (0xC88)



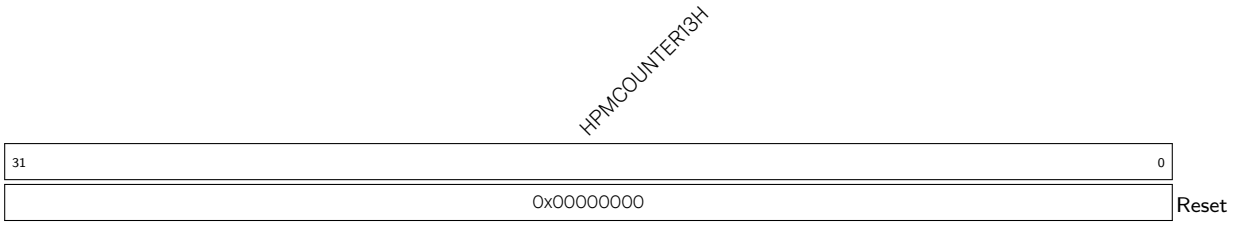
HPMCOUNTER8H mhpmcounter8h 的只读镜像。(RO)

Register 2.61. hpmcounter9h (0xC89)



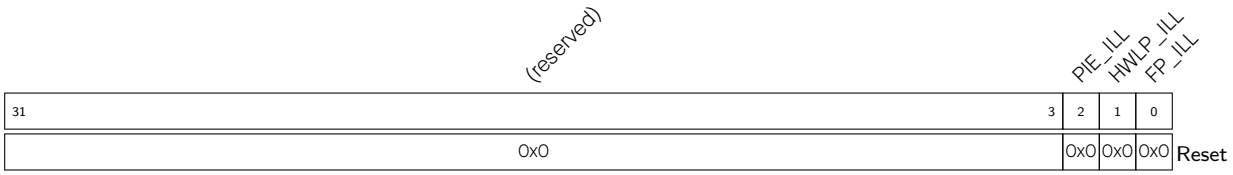
HPMCOUNTER9H mhpmcounter9h 的只读镜像。(RO)

Register 2.62. hpmcounter13h (0xC8D)



HPMCOUNTER13H mhpmcounter13h 的只读镜像。(RO)

Register 2.63. mext_ill (0x7F0)

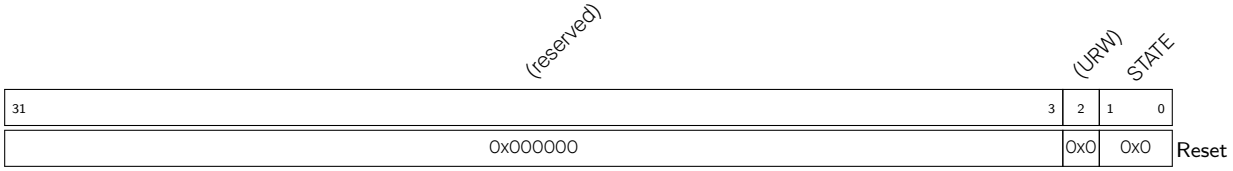


PIE_ILL 表示造成非法指令异常的原因是否为 CPU 执行禁用的 PIE 扩展指令。(R/W)

HWLP_ILL 表示造成非法指令异常的原因是否为 CPU 执行禁用的 HWLP 扩展指令。(R/W)

FP_ILL 表示造成非法指令异常的原因是否为 CPU 执行禁用的 RV32F 扩展指令，即 mstatus.FS=0x0。(R/W)

Register 2.64. mext_hwlp_status (0x7F1)



URW 表示是否启用用户模式访问用户模式的硬件循环 CSR。

0：禁用访问

1：启用访问

(R/W)

STATE 配置 HWLP 扩展状态：

0x0：关闭（访问 HWLP 指令/CSR 将触发非法指令异常）

0x1：初始

0x2：干净

0x3：脏

假设 HWLP 状态不是关闭的，每当 HWLP CSR 被修改或 HWLP 指令被执行时，状态将更新为脏。软件可以根据上下文将状态设置为初始或干净。根据 RISCV 标准：当扩展的状态设置为关闭时，任何试图读取或写入相应状态的指令将导致非法指令异常。

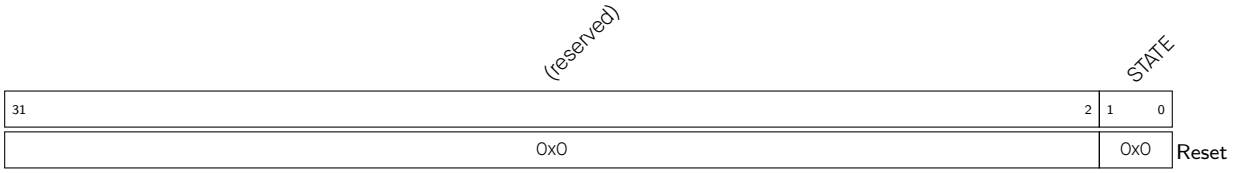
当状态为初始时，相应状态应具有初始常量值。

当状态为干净时，相应状态可能不是初始值，而是上下文切换时存储的最后一个值。

当 HWLP 的第一次循环完成时，状态变为脏。当状态为脏时，则意味着自上次上下文保存后，相应状态可能已被修改。

(R/W)

Register 2.65. mext_pie_status (0x7F2)



STATE 配置 PIE 扩展状态：

0x0：关闭（执行 PIE 指令将触发非法指令异常）

0x1：初始

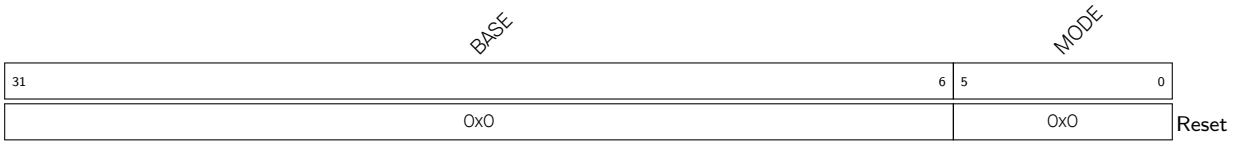
0x2：干净

0x3：脏

这是 PIE 扩展的状态位，与 RV32F 的 mstatus.FS 位有类似的功能。有关 STATE 值的标准解释，请参阅《RISC-V 指令集手册第二卷 “特权架构”》。

(R/W)

Register 2.66. jvt (0x017)



BASE 表示跳转表基地址。(R/W)

MODE 表示可配置的模式。此域用于定义此寄存器的用途，目前仅定义了一种用途，即跳转表，更多用途保留供将来使用。

0x0：JTV_BASE 用作跳转表。

其他：保留，供将来使用

(R/W)

Register 2.67. mexstatus (0x7E1)



EXPT_VLD 进入陷阱处理程序时，自动设置此位。

此位仅可从调试模式写入。

(RO)

LOCKUP 当两个异常发生时，自动设置此位。

一旦锁定，核心将暂停运行，并等待调试器处理锁定。

此位仅可从调试模式写入。

(RO)

BUS_ERR 由于总线错误异常进入加载/存储错误陷阱时，自动设置此位。

一旦进入陷阱处理程序，必须读取此位，确定加载/存储错误是否由总线错误造成。

若 mexstatus.NMFT 已启用，执行 MRET 指令时，此位立即自动清除。

mexstatus.NMFT 禁用且 mexstatus.PBEXE 启用时，必须在恢复执行前手动清除此位，否则可能会重新触发挂起的总线错误异常。

(R/W)

PBEXE 配置是否启用挂起的总线错误异常。

0：禁用

1：启用

当 mexstatus.PBEXE 启用且 mexstatus.NMFT 禁用时，设置 mexstatus.BUS_ERR 会生成同步异常。

如果 mexstatus.NMFT 启用，则 mexstatus.PBEXE 无效。

进入陷阱时，mexstatus.PBEXE 的值被写入 mexstatus.PPBEXE，并且 mexstatus.PBEXE 自动设置为 0。

通过 MRET 退出陷阱时，mexstatus.PPBEXE 的值被写回 mexstatus.PBEXE。

(R/W)

PPBEXE 配置是否启用之前挂起的总线错误异常。

0：禁用

1：启用

进入陷阱时，mexstatus.PBEXE 的值写入 mexstatus.PPBEXE。

通过 MRET 退出陷阱时，mexstatus.PPBEXE 的值被写回 mexstatus.PBEXE，并且 mexstatus.PPBEXE 自动设置为 1。

(R/W)

见下页……

Register 2.67. mexstatus (0x7E1)

接上页……

PMP_ERR 由于 PMP/PMA 异常进入加载/存储错误陷阱时，自动设置此位。

一旦进入陷阱处理程序，必须读取此位，确定加载/存储错误是否由 PMP/PMA 错误造成。

MRET 时此位自动清除。

(R/W)

FETCH_ERR 由于总线错误异常进入取指错误陷阱时，自动设置此位。

一旦进入陷阱处理程序，必须读取此位，确定取指错误是否由总线错误造成。

通过 MRET 退出陷阱时，此位自动清除。

(R/W)

NMFT 配置在进入陷阱后是否禁用存储器屏障。

1：CPU 在异常/中断时，直接进入陷阱处理程序，不等待挂起的存储器加载/存储操作完成（默认）。

0：CPU 等待挂起的存储器加载/存储操作完成后，再进入陷阱处理程序。

(R/W)

CLIC_INHV 配置是否启用恢复错误向量表取指。

0：禁用

1：启用（默认）。启用时，mcause.minhv 决定 mepc 保存的是指令地址还是向量表地址。

(R/W)

Register 2.68. mhint (0x7C5)

(reserved)	SBE	(reserved)	
31 21	20	19 0	
0x000	0	0x00000	Reset

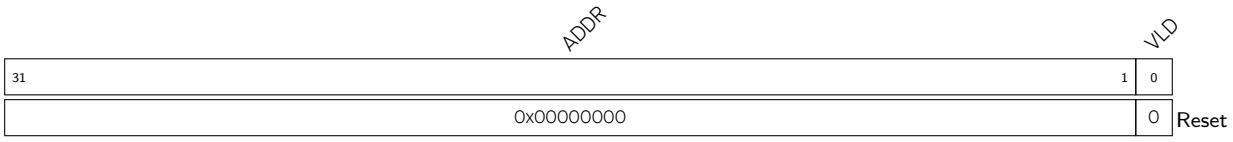
SBE 配置是否启用同步加载/存储总线错误异常。

0：禁用

1：启用

(R/W)

Register 2.69. ldpc0 (0xBE0)



VLD 配置此 CSR 中程序计数器地址值对于最近的加载总线错误异常是否有效。

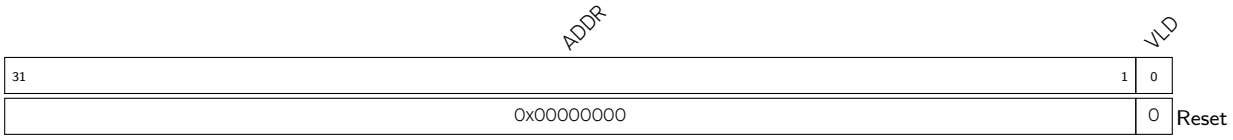
0：无效

1：有效

(R/W)

ADDR 配置与加载总线错误异常对应的程序计数器的高 31 位，数值应半字对齐。(R/W)

Register 2.70. ldpc1 (0xBE1)



VLD 配置此 CSR 中程序计数器地址值对于最近的加载总线错误异常是否有效。

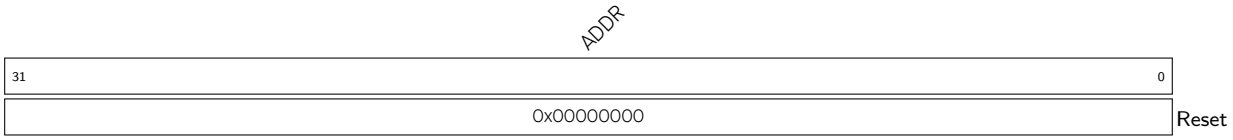
0：无效

1：有效

(R/W)

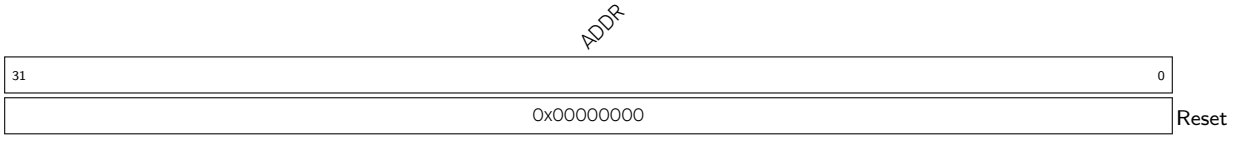
ADDR 配置与加载总线错误异常对应的程序计数器的高 31 位，数值应半字对齐。(R/W)

Register 2.71. ldtval0 (0xBE8)



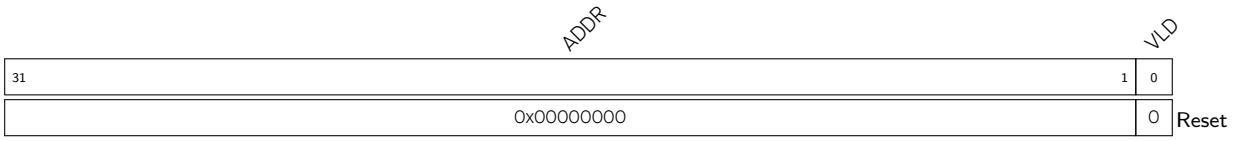
ADDR 配置与 ldpc0 中记录的加载总线错误异常对应的访问地址。(R/W)

Register 2.72. ldtval1 (0xBE9)



ADDR 配置与 ldpc1 中记录的加载总线错误异常对应的访问地址。(R/W)

Register 2.73. stpc0 (0xBF0)



VLD 配置此 CSR 中程序计数器地址值对于最近的存储总线错误异常是否有效。

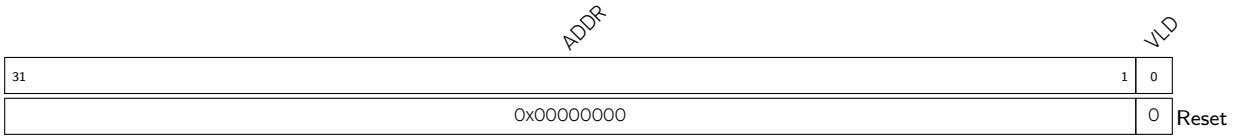
0：无效

1：有效

(R/W)

ADDR 配置与存储总线错误异常对应的程序计数器的高 31 位，数值应半字对齐。(R/W)

Register 2.74. stpc1 (0xBF1)



VLD 配置此 CSR 中程序计数器地址值对于最近的存储总线错误异常是否有效。

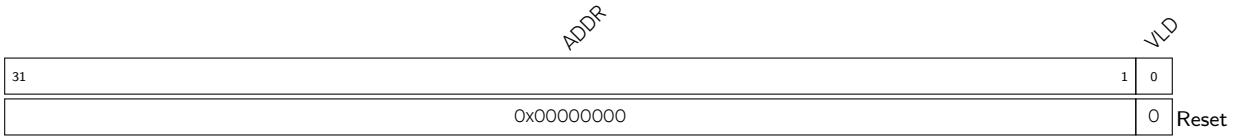
0：无效

1：有效

(R/W)

ADDR 配置与存储总线错误异常对应的程序计数器的高 31 位，数值应半字对齐。(R/W)

Register 2.75. stpc2 (0xBF2)



VLD 配置此 CSR 中程序计数器地址值对于最近的存储总线错误异常是否有效。

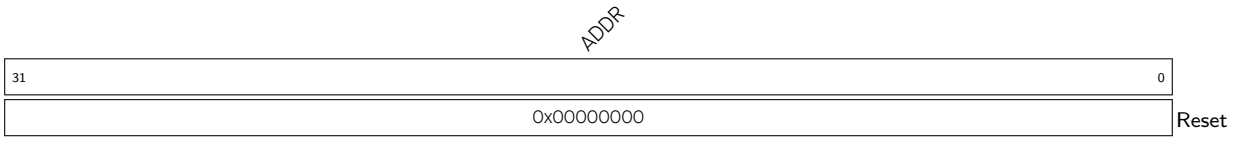
0：无效

1：有效

(R/W)

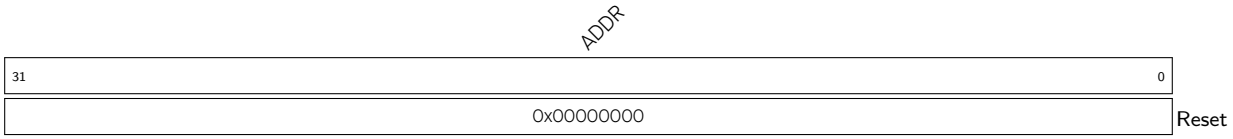
ADDR 配置与存储总线错误异常对应的程序计数器的高 31 位，数值应半字对齐。(R/W)

Register 2.76. sttval0 (0xBF8)



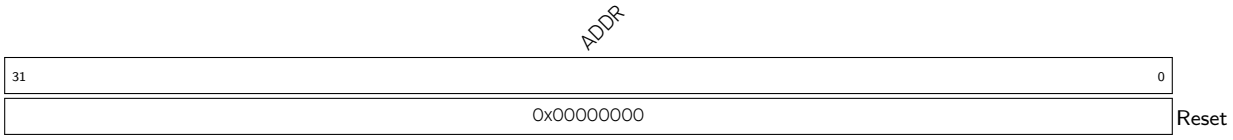
ADDR 配置与 stpc0 中记录的存储总线错误异常对应的访问地址。(R/W)

Register 2.77. sttval1 (0xBF9)



ADDR 配置与 stpc1 中记录的存储总线错误异常对应的访问地址。(R/W)

Register 2.78. sttval2 (0xBFA)



ADDR 配置与 stpc2 中记录的存储总线错误异常对应的访问地址。(R/W)

2.6 RISC-V 标准 ISA 扩展支持

每个 HP 核心实现了基本整数 ISA，并符合 RV32I 基本整数指令集的 2.0 版本，详见《RISC-V 指令集手册 v2.2 第一卷 “RISC-V 用户级 ISA”》(RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2)。

2.6.1 M 扩展

2.6.1.1 概述

除了基本的 RV32I 指令集外，每个 HP 核心还实现了标准整数乘法和除法扩展，并完全符合《RISC-V 指令集手册 v2.2 第一卷 “RISC-V 用户级 ISA”》中指定的 M 标准扩展的 2.0 版本。

2.6.1.2 功能描述

M 扩展中的指令对两个整数寄存器中的值进行乘法和除法操作。以下是乘法和除法指令在流水线中没有数据冒险的情况下所需的 CPU 周期。

- MUL 操作需要 2 个周期
- DIV 操作需要 1-19 个周期，具体取决于操作数的情况

2.6.2 C 扩展

2.6.2.1 概述

每个 HP 核心还实现了 RISC-V 标准压缩 (C) 指令集扩展，并完全符合《RISC-V 指令集手册 v2.2 第一卷 “RISC-V 用户级 ISA”》中 C 标准扩展的 2.0 版本。RVC 是 RISC-V 压缩扩展的简称。RVC 可以为常见操作添加较短的 16 位指令编码，减少静态和动态代码大小。通常，程序中 50-60% 的 RISC-V 指令可以替换为 RVC 指令，从而可以减少 25-35% 的代码量。

2.6.2.2 功能描述

RVC 采用的压缩方案较为简单，为 32 位 RISC-V 指令提供较短的 16 位版本，适用情景如下：

1. 立即数和地址偏移量较小
2. 其中一个寄存器是零寄存器 (x0)、ABI 链接寄存器 (x1) 或 ABI 堆栈指针 (x2)
3. 目标寄存器和第一个源寄存器相同
4. 使用的寄存器是 8 个最常用的寄存器，即 x8-x15

2.6.3 F 扩展

2.6.3.1 概述

每个 HP 核心实现了 RISC-V 标准单精度浮点指令集扩展，简称 RV32F 扩展。该扩展提供了寄存器文件和各种 CSR 用于配置和标志检查。完整信息请参阅《RISC-V 指令集手册 v2.2 第一卷 “RISC-V 用户级 ISA”》> 章节 F Standard Extension Version 2.0。

2.6.3.2 功能描述

- 各种指令用于执行浮点算术操作、整数格式的转换/移动操作、比较操作、分类操作和存储器加载/存储操作
- 符合 IEEE 754-2008 标准的单精度 32 位浮点操作
- 独立的 32 位宽浮点寄存器文件，包含 32 个寄存器
- 可配置的舍入模式
- 标志位支持检查不精确、上溢、下溢、无效和除以零等结果
- 次正规数 (subnormal) 和非数字 (NaN) 算术
- 状态位可配置开启/禁用扩展，程序执行过程中可追踪扩展使用情况

2.6.3.3 初始化

RV32F 扩展默认禁用，必须通过 mstatus.FS 位手动启用，否则在执行任何 RV32F 指令或访问任何相关 CSR 时将生成非法指令异常。

完整信息请参阅《RISC-V 指令集手册 v1.10 第二卷 “特权架构”》> 章节 Extension Context Status in mstatus Register。

2.6.4 Zc (Z) 扩展

2.6.4.1 概述

Zc 扩展有助于减少代码量，包含多个子集扩展。ESP32-P4 HP 核心实现的 Zc 扩展与 《Zc 规范 v1.0.4-3》 兼容，实现的子集扩展包括 Zcb、Zcmp 和 Zcmt。

2.6.4.2 功能描述

Zcb 扩展

Zcb 扩展依赖于 C 扩展，并提供更多压缩指令，适用情景如下：

- 立即数和地址偏移量较小
- 目标寄存器和第一个源寄存器相同
- 使用的寄存器是 8 个最常用的寄存器 (x8-x15)

Zcmp 扩展

Zcmp 扩展支持的指令在执行时可以等效为执行标准的 32 位 RISC-V 指令。

- PUSH、POP、POPRET 操作用于减少函数序言 (prologue) 和尾声 (epilogue) 的大小
  - PUSH 操作
    - 调整堆栈指针以创建堆栈帧
    - 将寄存器列表中指定的寄存器推送（存储）到堆栈帧
  - POP 操作
    - 从堆栈帧中弹出（加载）寄存器列表中的寄存器
    - 调整堆栈指针以销毁堆栈帧
  - POPRET 操作
    - 从堆栈帧中弹出（加载）寄存器列表中的寄存器
    - cm.popretz 还将零移动到 a0 作为返回值
    - 调整堆栈指针以销毁堆栈帧
    - 执行 ret 指令以从函数返回
- 将两个 a0-a1 移动到 s0-s7 的两个寄存器中
- 将两个 s0-s7 移动到 a0-a1

Zcmt 扩展

Zcmt 扩展也称为表跳转，它从跳转表中读取跳转目标地址，然后跳转到该地址。

表跳转使用指令存储器中的 256 项 32 位宽的表来存储函数地址。表必须至少 64 字节对齐。作为字典压缩的一种形式，表跳可减少 jal、auipc+jalr、jr、auipc+jr 指令的代码大小。

表跳转允许链接器用 cm.jalt 编码和表中的一个条目替换以下指令序列：

- 32 位 j 调用
- 32 位 jal ra 调用

2.6.4.3 Zc 指令

指令	助记符	描述
Zcb 扩展		
c.lbu	c.lbu rd', uimm(rs1')	对立即数 uimm 零扩展后与 rs1' 相加，得到一个存储器地址，从该地址加载一个字节，将该字节零扩展至 32 位，并写入 rd'
c.lhu	c.lhu rd', uimm(rs1')	对立即数 uimm 零扩展后与 rs1' 相加，得到一个存储器地址，从该地址加载一个半字，将该字节零扩展至 32 位，并写入 rd'
c.lh	c.lh rd', uimm(rs1')	对立即数 uimm 零扩展后与 rs1' 相加，得到一个存储器地址，从该地址加载一个半字，将该字节符号扩展至 32 位，并写入 rd'
c.sb	c.sb rs2', uimm(rs1')	对立即数 uimm 零扩展后与 rs1' 相加，得到一个存储器地址，将 rs2' 的最低有效字节存储到该地址
c.sh	c.sh rs2', uimm(rs1')	对立即数 uimm 零扩展后与 rs1' 相加，得到一个存储器地址，将 rs2' 的最低有效半字存储到该地址
c.zext.b	c.zext.b rsd'	将操作数的最低有效字节零扩展为 32 位，即：将高于 7 位的所有位填充为 0
c.sext.b	c.sext.b rsd'	将操作数的最低有效字节符号扩展为 32 位，即：将该字节中最高位的数值填充到比该位更高的位上
c.zext.h	c.zext.h rsd'	将操作数的最低有效半字零扩展为 32 位，即：将高于 15 位的所有位填充为 0
c.sext.h	c.sext.h rsd'	将操作数的最低有效半字符号扩展为 32 位，即：将该半字中最高位的数值填充到比该位更高的位上
c.not	c.not rsd'	对 rd'/rs1' 取反并将结果写入同一寄存器
c.mul	c.mul rsd'	将 rsd' 和 rs2' 源操作数的 32 位相乘，并将最低 32 位写入 rsd' 的结果
Zcmp 扩展		
cm.push	cm.push {reg_list}, -stack_adj	将 reg_list 中的寄存器存储到堆栈指针以下的存储器中，然后通过将堆栈指针减少一个 stack_adj 的值来创建堆栈帧，同时包含 spimm 值请求的额外堆栈空间
cm.pop	cm.pop {reg_list}, stack_adj	从堆栈存储器中加载 reg_list 中的寄存器，然后按照 stack_adj 调整堆栈指针
cm.popret	cm.popret {reg_list}, stack_adj	从堆栈存储器中加载 reg_list 中的寄存器，按照 stack_adj 调整堆栈指针，然后返回到 ra
cm.popretz	cm.popretz {reg_list}, stack_adj	从堆栈存储器中加载 reg_list 中的寄存器，通过 stack_adj 调整堆栈指针，将零移动到 a0，然后返回到 ra
cm.mva01s	cm.mva01s rs1', rs2'	将 r1s' 移动到 a0，将 r2s' 移动到 a1
cm.mva01	cm.mva01 r1s', r2s'	将 a0 移动到 r1s'，将 a1 移动到 r2s'
Zcmt 扩展		
cm.jt	cm.jt index	从存储器中的跳转向量表中读取一个条目并跳转到读取的地址
cm.jalt	cm.jalt index	从存储器中的跳转向量表中读取一个条目并跳转到读取的地址，链接到 ra

有关指令的更多详细信息，请参阅 《Zc 规范 v1.0.4-3》 。

2.6.4.4 限制

jvt 将架构状态添加到上下文中。因此，必须在上下文切换时保存和恢复架构状态。

2.6.5 位操作 (Zb) 扩展

2.6.5.1 概述

位操作（bit manipulation，简称 bitmanip）扩展，也称为 Zb 扩展集，包含多个子扩展，是对 RISC-V 基础架构的补充。这些扩展旨在提升代码密度、性能及效率。HP 核心完全支持 Zb 扩展。有关 Zb 扩展的详细规范，请参阅 Zb-specification v1.0.0。

2.6.5.2 功能描述

虽然位操作指令是为通用场景设计的，但某些指令在特定场景中更为适用。因此，bitmanip 扩展集被划分为多个较小的子扩展，而非单一的大扩展。每个子扩展都有其对应的 Zb 扩展名称，其包含的指令有在功能上相关，在使用场景上相似，甚至有时具有相同的实现逻辑。一些指令仅存在于某个子扩展中，而另一些指令可能存在于多个子扩展中。这些指令的助记符和编码独立于它们所属的子扩展。因此，在实现具有重复指令的多个子扩展时，逻辑或编码不会冗余。

2.6.6 A 扩展

2.6.6.1 概述

对原子 (A) 扩展的支持符合《RISC-V 指令集手册 v2.2 第一卷 “RISC-V 用户级 ISA”》，重点是保证向前推进，即功能上避免出现任何数据存储器被无限期锁定的情况。

2.6.6.2 功能描述

原子 (A) 扩展指令集包含可对存储器执行原子读-改-写操作的指令，用于同步同一存储器空间内运行的多个 RISC-V 硬件线程。原子指令包括两种形式：保留加载/条件存储（load-reserved/store-conditional）指令，以及原子获取并操作 (fetch-and-op) 存储器指令。目前，这些原子指令会忽略 aq（获取）和 rl（释放）位，因为在当前架构中，存储器顺序总是可以保证，因此这些位无实际作用。

2.6.6.3 加载保留字 (LR.W) 指令

LR.W 指令只是锁定正在读取的 32 位对齐存储器地址。一旦某个 4 字节的存储器空间被锁定，它将保持锁定状态，其他 hart 无法访问该区域，直到在执行过程中遇到以下任何一种情况：

- 任何读操作
- 任何写操作
- 任何中断/异常
- 向后跳转/向后分支跳转
- JALR 指令
- ECALL/EBREAK/MRET/URET 指令
- FENCE/FENCE.I 指令
- 调试模式
- 临界区超过 64 字节
- SC.W 指令中的数据地址与 LR.W 指令中的不匹配

如果发生上述任何一种情况，存储器锁将立刻被释放。如果遇到 SC.W 指令，则存储器锁不会立即被释放，但最终会被释放，有关具体如何释放的信息，请参考章节 2.6.6.4。

如果地址未对齐，则会导致异常 mcause = 6。

2.6.6.4 条件存入字 (SC.W) 指令

SC.W 指令首先检查存储器锁是否仍然有效，以及地址是否与上一条 LR.W 指令指定的地址相同，只有以上条件同时满足时，SC.W 指令才会对存储器执行写操作，并且在收到存储器发出的写操作完成确认信息后立即释放锁。

如果锁已失效（失效原因见章节 2.6.6.3），SC.W 指令将在目标寄存器 rd 中设置失败代码，目前只能设置为 1。

如果地址未对齐，则会导致异常 mcause = 6。

2.6.6.5 AMO 指令

AMO (原子性存储器操作) 指令分三步执行：

1. 从 rs1 给定的存储器地址读取数据，并将数据保存到目标寄存器 rd。
2. 将 rd 和 rs2 中的数据进行运算。
3. 将上面第 2 步得到的结果写入 rs1 给定的存储器地址。

AMO 运算共有 9 种类型：SWAP、ADD、AND、OR、XOR、MAX、MIN、MAXU 和 MINU。

在整个过程中，存储器地址保持锁定状态，其他 hart 无法访问。如果遇到未对齐的地址，则会导致异常 mcause = 6。

AMO 操作的读/写访问 (PMP/PMA) 都在第一步中进行检查。若发生错误，则 mcause = 7。

2.7 自定义指令扩展

2.7.1 硬件循环

2.7.1.1 概述

HWLP（硬件循环）是在硬件中实现的 for 循环。在执行循环前会配置循环的起始地址、结束地址以及循环次数。当指令执行遇到起始地址时，循环执行开始。当执行到结束地址时，视为一个循环计数，并再次获取起始地址。此过程重复进行，直到循环计数等于配置的循环次数。

2.7.1.2 特性

HWLP 具有以下特性：

- 与条件分支指令相比，执行时无额外惩罚
- 支持两个嵌套的 HWLP
- 如果结束地址在配置指令 1 KB 以内，则可通过单个指令 (lp.setupi) 进行配置

2.7.1.3 功能描述

HWLP 的配置可以通过 CSR 或 HWLP 的自定义指令进行，后者将在后续章节中详细描述。HWLP 需要配置起始、结束地址和循环次数。

当设置单个循环时，可以配置 Loop0 或 Loop1 CSR。在设置 HWLP 之前，请将 mhwloop_state_reg 中的 STATE 配置为非零值（零表示 OFF，默认状态）。否则，访问 HWLP 配置 CSR 将导致非法指令异常。如果要在用户模式下设置和执行 HWLP，请将 mhwloop_state_reg 中的URW 设置为 1。

核心最多支持两个 HWLP，可以嵌套执行。在这种情况下，每个循环需单独配置。请注意，如果两个循环的结束指令相同，则内循环优先于外循环。在配置嵌套循环时，Loop0 必须作为内循环，Loop1 必须作为外循环，因为在硬件实现中 Loop0 的优先级高于 Loop1。

与条件分支指令相比，HWLP 的优势在于执行时没有惩罚。分支/跳转指令使用预测逻辑来评估分支条件和分支/跳转地址。如果预测错误，则需刷新核心流水线，并获取正确地址。因此，刷新前去获取不正确的指令已经浪费了几个周期。由于 HWLP 不需要预测条件分支的结果或目标地址，因此没有惩罚，具有更高的吞吐量。

2.7.1.4 HWLP 中支持的指令/操作/模式

- 加载类型指令
- 存储类型指令
- 算术、逻辑、AIA 指令
- Zc 扩展指令
- HWLP 中的任何中断/异常
- ECALL/EBREAK
- FENCE/FENCE.I/CSRx 指令
- 原子指令

- 调试模式
- SC.W 指令中的数据地址与 LR.W 指令中的不匹配

2.7.1.5 HWLP 限制

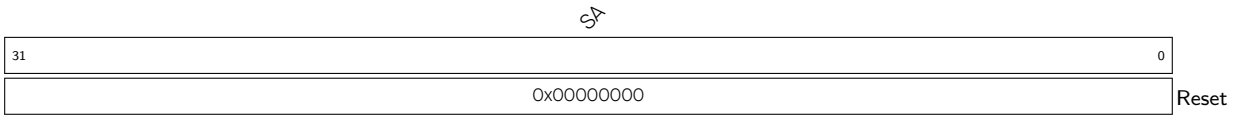
请注意，将分支或跳转指令置于硬件循环末尾，将可能引发异常行为。

2.7.1.6 寄存器列表

名称	描述	地址	访问
mhwloop0_start_addr	机器 Loop0 起始地址	0x7C6	R/W
mhwloop0_end_addr	机器 Loop0 结束地址	0x7C7	R/W
mhwloop0_count	机器 Loop0 计数	0x7C8	R/W
mhwloop1_start_addr	机器 Loop1 起始地址	0x7C9	R/W
mhwloop1_end_addr	机器 Loop1 结束地址	0x7CA	R/W
mhwloop1_count	机器 Loop1 计数	0x7CB	R/W
mext_ill_reg	机器扩展非法寄存器	0x7F0	R/W
mhwloop_state_reg	机器 HWLP 状态寄存器	0x7F1	R/W
uhwloop0_start_addr	用户 Loop0 起始地址	0x8C0	R/W
uhwloop0_end_addr	用户 Loop0 结束地址	0x8C1	R/W
uhwloop0_count	用户 Loop0 计数	0x8C2	R/W
uhwloop1_start_addr	用户 Loop1 起始地址	0x8C3	R/W
uhwloop1_end_addr	用户 Loop1 结束地址	0x8C4	R/W
uhwloop1_count	用户 Loop1 计数	0x8C5	R/W
uhwloop_state_reg	用户 HWLP 状态寄存器	0x8C6	R/W

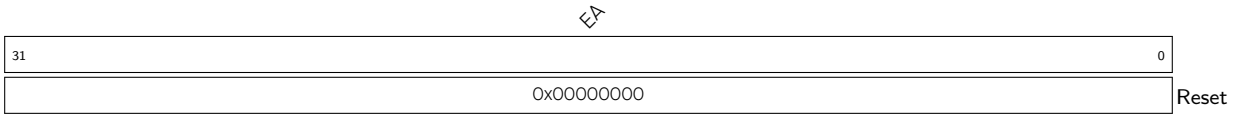
2.7.1.7 寄存器描述

Register 2.79. mhwloop0_start_addr (0x7C6)



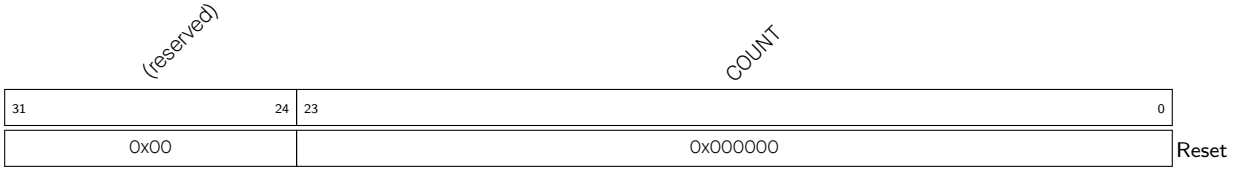
SA 配置 HW Loop0 第一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。(R/W)

Register 2.80. mhwloop0_end_addr (0x7C7)



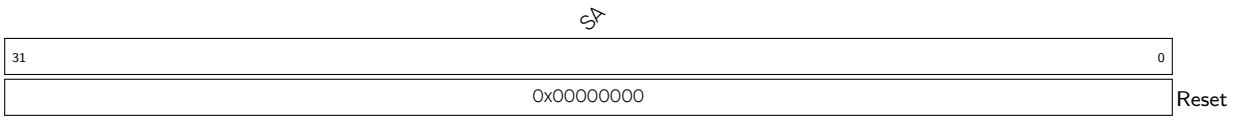
EA 配置 HW Loop0 最后一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。(R/W)

Register 2.81. mhwloop0_count (0x7C8)



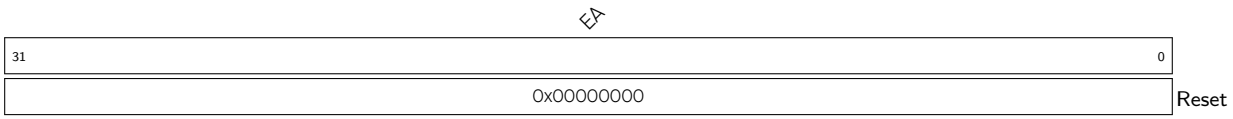
COUNT 配置 HW Loop0 的循环次数。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。(R/W)

Register 2.82. mhwloop1_start_addr (0x7C9)



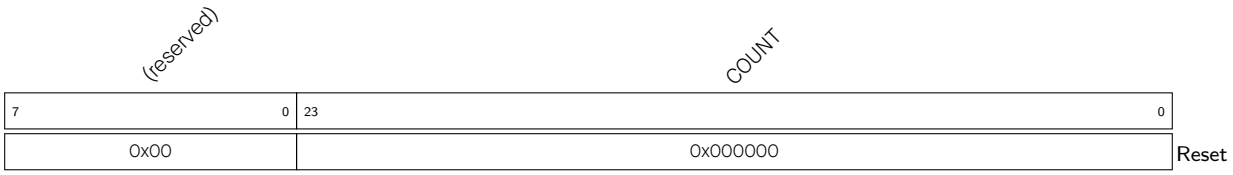
SA 配置 HW Loop1 第一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。(R/W)

Register 2.83. mhwloop1_end_addr (0x7CA)



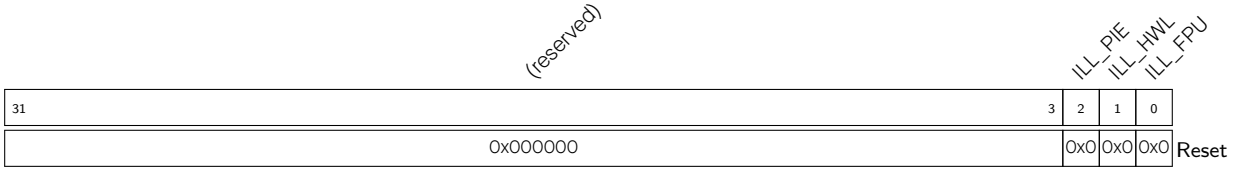
EA 配置 HW Loop1 最后一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。(R/W)

Register 2.84. mhwloop1_count (0x7CB)



COUNT 配置 HW Loop1 的循环次数。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。(R/W)

Register 2.85. mext_ill_reg (0x7F0)



ILL_FPU 表示是否由于禁用 F 扩展而触发非法指令。

0：未触发

1：已触发

(RO)

ILL_HWL 表示是否由于禁用 HWLP 扩展而触发非法指令。

0：未触发

1：已触发

(RO)

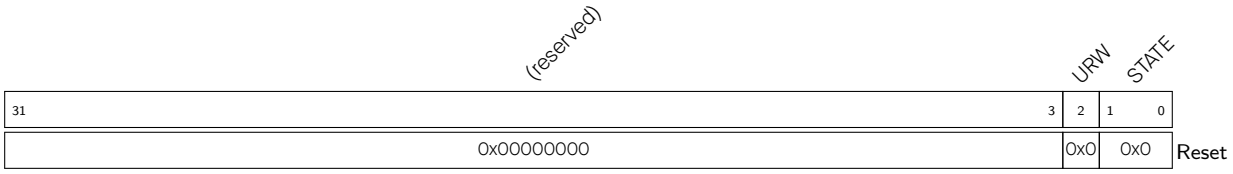
ILL_PIE 表示是否由于禁用 PIE 扩展而触发非法指令。

0：未触发

1：已触发

(RO)

Register 2.86. mhwloop_state_reg (0x7F1)



STATE 配置 HWLP 扩展的状态。

0x0：关闭（访问 HWLP 指令和 CSR 将触发非法指令异常）

0x1：初始

0x2：干净

0x3：脏

(R/W)

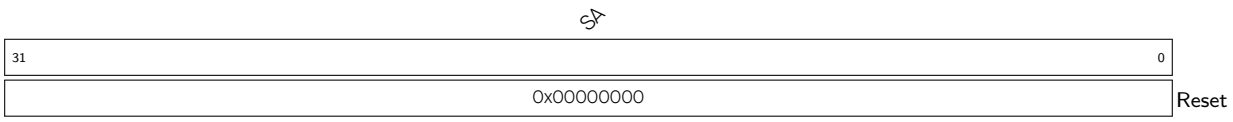
URW 配置用户模式下对用户模式 HWLOOP CSR 的访问。

0：禁用用户模式访问

1：启用用户模式访问

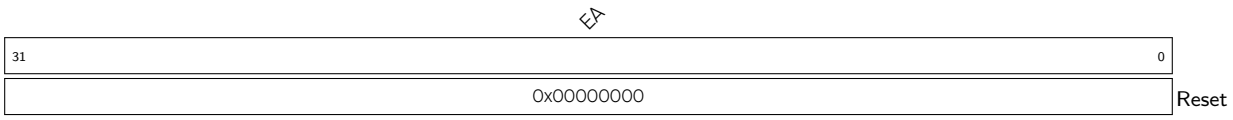
(R/W)

Register 2.87. uhwloop0_start_addr (0x8C0)



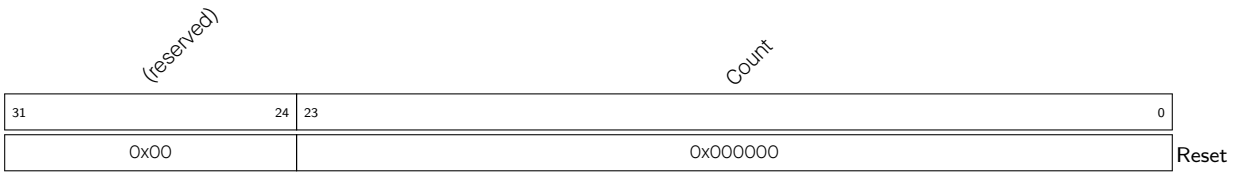
SA 配置用户模式下 HW Loop0 第一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。仅当 mhwloop_state_reg.URW 为 1 时可读写。(R/W)

Register 2.88. uhwloop0_end_addr (0x8C1)



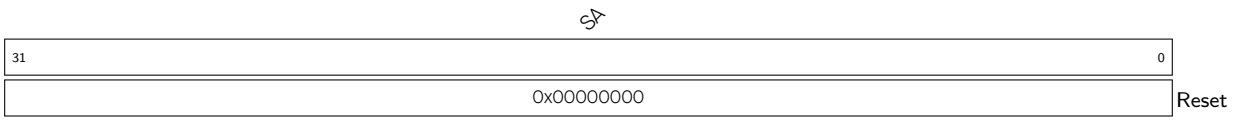
EA 配置用户模式下 HW Loop0 最后一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。仅当 mhwloop_state_reg.URW 为 1 时可读写。(R/W)

Register 2.89. uhwloop0_count (0x8C2)



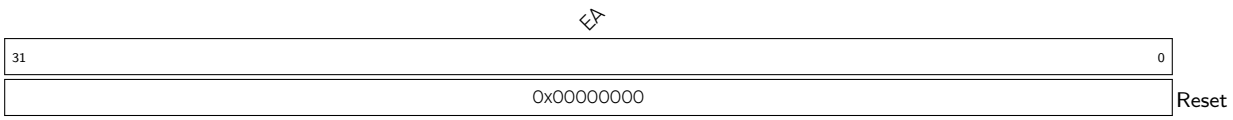
Count 配置用户模式下 HW Loop0 的循环次数。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。仅当 mhwloop_state_reg.URW 为 1 时可读写。(R/W)

Register 2.90. uhwloop1_start_addr (0x8C3)



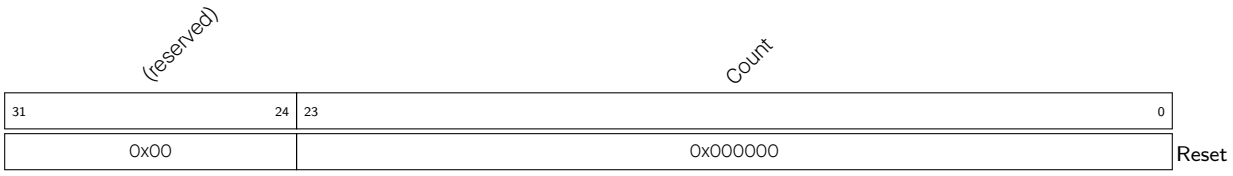
SA 配置用户模式下 HW Loop1 的第一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。仅当 mhwloop_state_reg.URW 为 1 时可读写。(R/W)

Register 2.91. uhwloop1_end_addr (0x8C4)



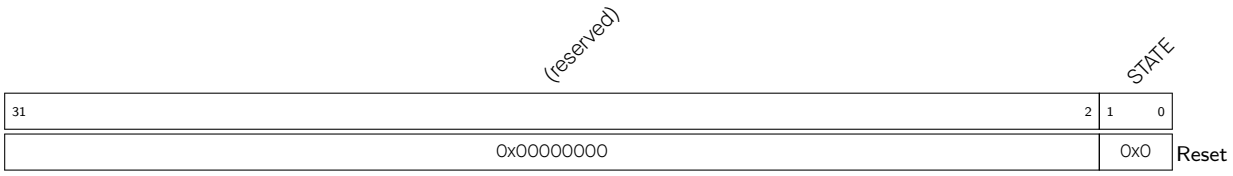
EA 配置用户模式下 HW Loop1 的最后一条指令的 32 位地址。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。仅当 mhwloop_state_reg.URW 为 1 时可读写。(R/W)

Register 2.92. uhwloop1_count (0x8C5)



Count 配置用户模式下 HW Loop1 的循环次数。当 mhwloop_state_reg.STATE 不为 0x0 时有效，即不为 OFF。仅当 mhwloop_state_reg.URW 为 1 时可读写。(R/W)

Register 2.93. uhwloop_state_reg (0x8C6)



STATE 配置用户模式下 HWLP 扩展状态。

0x0：关闭（访问 HWLP 指令和 CSR 将触发非法指令异常）

0x1：初始

0x2：干净

0x3：脏

该位的物理实现与 mhwloop_state_reg 相同。请确保在 mhwloop_state_reg.URW 域中启用了访问权限。

假定状态不为 OFF 时，每当 HWLP CSR 被修改或执行 HWLP 指令时，状态将被更新为脏。(R/W)

2.7.1.8 HWLP 指令

下表为 HWLP 指令集。

表 2.7-2. HWLP 指令

SN	指令助记符	描述	31:20	19:15	14:12	11:8	7	6:0
1	lp.setupi L, uimm_Count, uimm_EndOffset	StartAddr = PC+4 EndAddr = PC+uimm_EndOffset Count = uimm_Count	uimm_Count [11:0]	uimm_EndOffset [5:1]	101	uimm_EndOffset [9:6]	L (index)	0101011
2	lp.setup L, rs1, uimm_EndOffset	StartAddr = PC+4 EndAddr = PC+uimm_EndOffset Count = GPR[rs1]	uimm_EndOffset [12:1]	rs1 [4:0] (count)	100	X	L (index)	0101011
3	lp.setup.beqz L, rs1, EndInstrSize, uimm_EndOffset	StartAddr = PC+4 EndAddr = PC+uimm_EndOffset Count = GPR[rs1] 硬件循环结束指令大小由 EndInstrSize 控制：设为 1 时对应 32 位；设为 0 时对应 16 位	uimm_EndOffset [12:1]	rs1 [4:0] (count)	100	结束指令大小（16 位指令为 0000，32 位为 0010）	L (index)	0101011
4	lp.starti L, uimm_StartOffset	StartAddr = PC+uimm_StartOffset	uimm_StartOffset [12:1]	X	000	X	L (index)	0101011
5	lp.endi L, uimm_EndOffset	EndAddr = PC+uimm_EndOffset	uimm_EndOffset [12:1]	X	001	X	L (index)	0101011
6	lp.count L, rs1	Count = GPR[rs1]	X	rs1 [4:0] (count)	010	X	L (index)	0101011
7	lp.counti L, uimm_Count	Count = uimm_Count	uimm_Count [11:0]	X	011	X	L (index)	0101011

GoBack

2.7.2 处理器指令扩展

2.7.2.1 概述

处理器指令扩展 (PIE) ISA 自定义多种 128 位宽的 SIMD（单指令多数据）和复杂指令，用于加速边缘推理和数字信号处理应用。这些指令完全集成到 CPU 流水线中，可在 HP 核心中与其兼容的标准 RV32IMAFC ISA 无缝协同使用。PIE 扩展支持遵循乐鑫 PIE V2.2.0 版本规范。

2.7.2.2 功能描述

- 8 x 128 位宽整数向量寄存器文件
- 每个 128 位宽向量寄存器可用作 16 x 8 位元素、8 x 16 位元素或 4 x 32 位元素进行 SIMD 操作
- 向量寄存器可直接加载/存储高达 128 位宽数据
- 支持并行执行算术和存储器操作融合指令
- 向量和标量 MAC 单元能够每周期对 16 x 8 位元素或 8 x 16 位元素进行操作
- 512 位宽向量寄存器用于累加 8 位或 16 位向量 MAC 操作的结果
- 40 位宽标量寄存器用于累加 8 位或 16 位标量 MAC 操作的结果
- 支持并行 16 位 FFT 和复数数学运算的指令
- 最终结果在舍入和饱和处理后生成
- 可配置的舍入模式和缩放（算术右移）参数

有关所有可用指令和配置的完整描述，请参阅章节 *4 处理器指令扩展 (PIE)*。

2.7.2.3 初始化、上下文切换和异常

PIE 扩展默认禁用，必须通过 mext_pie_status.STATE 域手动启用。否则，在执行任何 PIE 扩展指令时将生成非法指令异常。

当 PIE 扩展禁用时发生了非法指令异常，即 mcause = 0x2 的异常，HP 核心将设置 mext_ill.PIE_ILL 位来标志该异常。

当 PIE 扩展启用时，即 mext_pie_status.STATE 位为非零值，执行任何 PIE 指令将始终导致 mext_pie_status.STATE 位更改为 DIRTY (11)。

若需启用 PIE 扩展，请执行以下步骤：

1. 将 mext_pie_status.STATE 位写入 INITIAL (01) 状态。
2. 使用 PIE 指令将 PIE 寄存器文件和累加器寄存器初始化为零，这将导致 mext_pie_status.STATE 位自动更改为 DIRTY (11)。
3. 由于此时 PIE 寄存器的值已知，将 mext_pie_status.STATE 位写入 CLEAN (10)。
4. 继续执行应用程序。
5. 在执行应用程序代码期间，如果遇到任何 PIE 指令，mext_pie_status.STATE 位将自动更改为 DIRTY (11)。

在上下文切换期间，是否保存或恢复 PIE 扩展寄存器取决于 mext_pie_status.STATE 位的值。如果值为 DIRTY，则必须将 PIE 寄存器的状态保存到堆栈中。之后，状态将更新为 CLEAN 或 OFF，具体取决于下一个上下文（线程）是否启用了 PIE 扩展。

某些 PIE 指令对寄存器域或立即数域有限制。例如，当执行加载和算术的融合指令时，如果两个并行操作的目标寄存器相同，将导致歧义，进而触发 PIE 非法异常，即 mcause = 0x1f。

2.8 存储器映射寄存器

2.8.1 概述

核心系统中存在以下紧密耦合的存储器映射寄存器：

- 每个 HP 核心的 CLIC 寄存器
- 共享的 CLINT 寄存器

2.8.2 特性

每个 HP CPU 核心支持：

- 访问另一个 HP 核心的 CLIC 寄存器
- 访问无效地址时生成总线错误异常
- 访问 CLINT 寄存器。但是，只有核心 0 可以修改 CLINT 定时器值

2.8.3 功能描述

下表为每个 HP 核心用于访问 CLIC 和 CLINT 地址范围的地址解码。

表 2.8-1. 存储器映射寄存器访问的地址解码

位	描述
31:28	固定为 0x2，即基地址
27:20	标识子模块： 0x0：CLINT 0x08：CLIC 0xB：CLIC 用户 其他：保留
19:16	控制访问类型 0x0：访问本核心的寄存器 0x1：访问另一核心的寄存器
15:0	寄存器偏移

表 2.8-2. 内部存储器地址映射

块	起始地址	结束地址
CLINT (本核心)	0x20000000	0x2000FFFF
CLIC (本核心)	0x20800000	0x2080FFFF
CLINT (另一核心)	0x20010000	0x2001FFFF
CLIC (另一核心)	0x20810000	0x2081FFFF

2.9 中断控制器

2.9.1 概述

每个 HP 核心使用 CLIC + CLINT 方案来实现中断支持。

CLINT 表示核心本地中断 (core-local interrupt)，是 HP 核心子系统中的中断源，为每个 HP 核心生成定时器和软件中断。

CLIC 表示核心本地中断控制器 (core-local interrupt controller)，为每个 HP 核心提供低延迟、向量化、抢占式中断。每个 HP 核心 CLIC 单元支持 32 个外部中断和 2 个 CLINT 中断。

2.9.2 CLIC

2.9.2.1 概述

HP 核心中实现的 CLIC 遵循 核心本地中断控制器 (CLIC) RISC-V 特权架构扩展 (10/10/2023)。

HP 核心仅支持 CLIC 操作模式，即 mtvec.MODE 硬连线为 0x3。因此，RISC-V 基本的中断处理方案和相关的 CSR（如 mie、mip、mideleg、uie 和 uip）不可用。

HP 核心支持机器模式中断和用户模式中断委派。

2.9.2.2 CLIC CSRs

以下机器模式 CSR 与 CLIC 方案中的中断处理相关：

- mstatus
- mtvec
- mtvt
- mscratch
- mepc
- mcause
- mnxti
- mintthresh
- mintstatus
- mclicbase
- mscratchcsw
- mscratchcswl

如前所述，CLIC 模式由 mtvec.MODE 域决定。在每个 HP 核心中，这些位硬连线为 0x3，表示仅支持 CLIC 模式。

在 CLIC 模式下，机器模式的全局中断启用仍由 mstatus.MIE 位决定。

每个中断可以是向量化的或非向量化的，具体取决于 clicintattr[i].SHV 域的值。

对于非向量化中断，HP 核心跳转到 mtvec.BASE 域中配置的陷阱处理程序基地址。

对于向量化中断，HP 核心跳转到向量表中相应索引对应的 32 位字地址，向量表的基地址在 mtvt 中配置。例如，对于第 i 个机器模式中断，首先加载存储器中 mtvt + 4*i 位置的 32 位地址，然后 HP 核心将加载的数据视为地址并跳转到该地址。

在接受中断后，HP 核心会立刻将中断的指令地址存储在 mepc 中。此外，HP 核心将根据中断的 ID 更新 mcause 的值。

2.9.2.3 中断映射

每个 HP 核心中的 CLIC 支持 32 个外部中断和 2 个核心本地中断。

ID 16-47 映射到外部中断，而 ID 15 及以下用于核心本地中断。

表 2.9-1. CLIC 中断映射

ID	描述
0-2	NA (保留)
3	CLINT 机器模式软件中断
4-6	NA (保留)
7	CLINT 机器模式定时器中断
8-15	NA (保留)
16	外部中断 0
17	外部中断 1
18	外部中断 2
...	...
45	外部中断 29
46	外部中断 30
47	外部中断 31

有关核心本地中断源的详细描述，请参阅章节 2.9.3。

32 个外部中断到 SoC 中断源的映射由中断矩阵管理。有关如何配置和映射到每个 HP 核心的 CLIC 中断的详细信息，请参阅章节 13 中断矩阵。

2.9.2.4 CLIC 参数

CLIC 实现的特性可以通过读取 mclicbase 和 clicinfo 寄存器获取：

- mclicbase 为只读 CSR，用于保存机器模式下 CLIC 存储器映射寄存器的基地址。
- clicinfo 为存储器映射只读寄存器，包含以下域：
  - NUM_INTERRUPTS：表示当前 CLIC 实现支持的中断数量
  - CLICINTCTLBITS：表示 clicintctl[i] 和 mintthresh.TH 中用于编码每个中断的级别和优先级的可编程高位数量。
- 假设 mclicbase 已知，与 clicinfo 对应的地址偏移量 0x0004 相加（参考章节 2.9.2.6），即可计算出 clicinfo 的完整地址。

2.9.2.5 中断级别和优先级编码

表 2.9-2 显示了配置 CLIC 中断级别和优先级的各种方式，具体取决于 mcliccfg.MNLBITS 和 clicintctl[i] 寄存器的配置的值，以及 CLICINTCTLBITS 参数的固定值 (0x3)。

表 2.9-2. CLIC 级别和优先级编码 (CLICINTCTLBITS=3)

mnlbits[3:0]	clicintctl[i][7:0]	中断级别	中断优先级
0	`ppp.....`	255	31, 63, 95, 127, 159, 191, 223, 255
1	`lpp.....`	127, 255	31, 63, 95, 127
2	`llp.....`	63, 127, 191, 255	31, 63
3-8	`lll.....`	31, 63, 95, 127, 159, 191, 223, 255	31

mcliccfg.MNLBITS 决定了 clicintctl[i] 寄存器中用于编码级别的高位数量（在上表中表示为 'l'），而剩余的低位用于编码优先级（在上表中表示为 'p'）。由于 clicintctl[i] 中 CLICINTCTLBITS 个高位是可写的，因此剩余的低位固定为 1（在上表中表示为 "."）。

级别和优先级均用于给某个中断设置更高的优先级。较高级别的中断可以抢占较低级别的中断（假设在中断处理程序中手动启用了 mstatus.MIE 以允许嵌套），但如果两个中断级别相同但优先级不同，且较低的中断正在被 CPU 处理，那么较高优先级的中断无法抢占该中断。因此，优先级仅能决定级别相同且都处于挂起状态的中断的处理顺序。如果如果两个中断的级别和优先级相同，则优先处理 ID 值较高的中断。

CPU 内部会根据有效中断级别阈值，来决定在机器模式下，是否允许新挂起的中断抢占正在执行的中断处理程序。该阈值为 mintstatus.MIL 和 mintthresh.TH 的较大值。

2.9.2.6 CLIC 存储器映射寄存器列表

下表列出了所有 CLIC 机器模式存储器映射寄存器，以及每个寄存器相对于 mclicbase 的地址偏移量和绝对地址。

请注意，clicintip[i]、clicintie[i]、clicintattr[i] 和 clicintctl[i] 为单字节寄存器，对这些寄存器进行字和半字对齐访问是原子的。

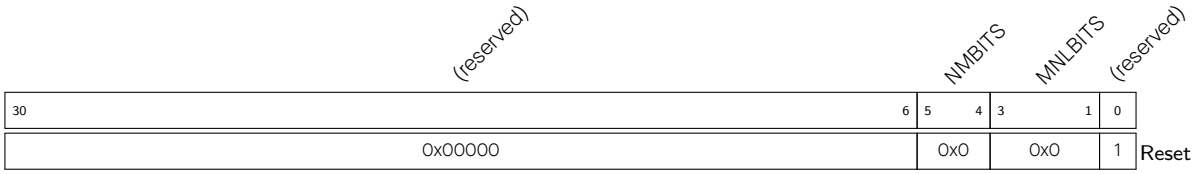
机器模式 CLIC 寄存器				
名称	描述	MCLICBASE 偏移	绝对地址	访问
mcliccfg	CLIC 机器模式全局配置寄存器	0x0000	0x20800000	R/W
clicinfo	CLIC 信息寄存器	0x0004	0x20800004	RO
mintthresh	CLIC 机器模式中断阈值寄存器	0x0008	0x20800008	R/W
clicintip[3]	CLINT 软件中断挂起寄存器	0x100C	0x2080100C	R/W
clicintie[3]	CLINT 软件中断使能寄存器	0x100D	0x2080100D	R/W
clicintattr[3]	CLINT 软件中断属性寄存器	0x100E	0x2080100E	R/W
clicintctl[3]	CLINT 软件中断级别控制寄存器	0x100F	0x2080100F	R/W
clicintip[7]	CLINT 定时器中断挂起寄存器	0x101C	0x2080101C	R/W
clicintie[7]	CLINT 定时器中断使能寄存器	0x101D	0x2080101D	R/W
clicintattr[7]	CLINT 定时器中断属性寄存器	0x101E	0x2080101E	R/W
clicintctl[7]	CLINT 定时器中断级别控制寄存器	0x101F	0x2080101F	R/W
clicintip[16]	外部中断 0 挂起寄存器	0x1040	0x20801040	R/W
clicintie[16]	外部中断 0 使能寄存器	0x1041	0x20801041	R/W
clicintattr[16]	外部中断 0 属性寄存器	0x1042	0x20801042	R/W
clicintctl[16]	外部中断 0 级别控制寄存器	0x1043	0x20801043	R/W
clicintip[17]	外部中断 1 挂起寄存器	0x1044	0x20801044	R/W
clicintie[17]	外部中断 1 使能寄存器	0x1045	0x20801045	R/W
clicintattr[17]	外部中断 1 属性寄存器	0x1046	0x20801046	R/W
clicintctl[17]	外部中断 1 级别控制寄存器	0x1047	0x20801047	R/W
……	……	……	……	……
clicintip[n]	第 n 个外部中断挂起寄存器 [6]	0x1040 + 4*n	0x20801040 + 4*n	R/W
clicintie[n]	第 n 个外部中断使能寄存器 [6]	0x1041 + 4*n	0x20801041 + 4*n	R/W

[6]需要注意的是，外部中断对应 CLIC 中断 ID 16 及以上。因此，MCLICBASE 偏移也可以写为 0x1000 + 4*i，其中 i 代表 CLIC ID，而 n = i + 16。另外，这些寄存器的描述使用 CLIC 索引 i 来指定偏移，而不是这里为了方便使用的外部中断索引 n。

名称	描述	MCLICBASE 偏移	绝对地址	访问
clicintattr[n]	第 n 个外部中断属性寄存器 [6]	0x1042 + 4*n	0x20801042 + 4*n	R/W
clicintctl[n]	第 n 个外部中断级别控制寄存器 [6]	0x1043 + 4*n	0x20801043 + 4*n	R/W
....	....	....	....	....
clicintip[46]	外部中断 30 挂起寄存器	0x10B8	0x208010B8	R/W
clicintie[46]	外部中断 30 使能寄存器	0x10B9	0x208010B9	R/W
clicintattr[46]	外部中断 30 属性寄存器	0x10BA	0x208010BA	R/W
clicintctl[46]	外部中断 30 级别控制寄存器	0x10BB	0x208010BB	R/W
clicintip[47]	外部中断 31 挂起寄存器	0x10BC	0x208010BC	R/W
clicintie[47]	外部中断 31 使能寄存器	0x10BD	0x208010BD	R/W
clicintattr[47]	外部中断 31 属性寄存器	0x10BE	0x208010BE	R/W
clicintctl[47]	外部中断 31 级别控制寄存器	0x10BF	0x208010BF	R/W

2.9.2.7 CLIC 存储器映射寄存器描述

Register 2.94. mcliccfg (0x20800000)



MNLBITS 配置 8 位的 clicintctl[i] 寄存器中有几个高位用来表示对应机器模式中断的级别。该配置为全局配置，对每个 HP 核心均有效。
有效取值范围为 0-8。如果写入超过 8 的值，会被截断 (clamp) 为 8。
(R/W)

NMBITS 配置 clicintattr[i].MODE 中有几位用来表示中断模式。该配置为全局配置，对每个 HP 核心均有效。对于仅有机器模式的系统，此域硬连线为 0。这表示 clicintattr[i].MODE 域硬连线为 0x3，因此所有中断仅在机器模式下。(RO)

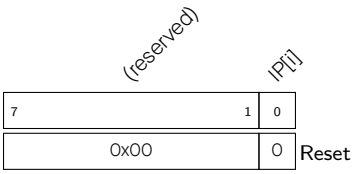
Register 2.95. clicinfo (0x20800004)



NUM_INTERRUPTS 表示此 CLIC 实现支持的中断数量。硬连线为 48。(RO)

CLICINTCTLBITS 表示在 8 位 clicintctl[i] 寄存器中实际可编程的高位数。硬连线为 0x3。(RO)

Register 2.96. clicintip[i] (0x20801000 + 4*i)



IP[i] 表示第 i 个 CLIC 机器模式中断的挂起状态。

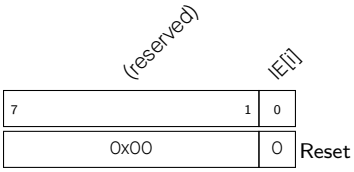
0：第 i 个中断未挂起

1：第 i 个中断挂起

- 当使用 clicintattr[i].TRIG 将第 i 个中断配置为电平类型时，此位为只读，若要清除此位，必须清除中断源。
- 当使用 clicintattr[i].TRIG 将第 i 个中断配置为边缘触发时，此位为可读可写，若需清除此位，向此寄存器写入 0 即可。

(R/W)

Register 2.97. clicintie[i] (0x20801001 + 4*i)



IE[i] 配置是否使能第 i 个 CLIC 机器模式中断。

0：禁用

1：启用

(R/W)

Register 2.98. clicintattr[i] (0x20801002 + 4*i)



SHV 配置第 i 个 CLIC 机器模式中断的硬件向量化。

0：第 i 个中断未硬件向量化。即，CPU 在接受此中断时将跳转到 mtvec 中配置的地址。

1：第 i 个中断硬件向量化。即，CPU 在接受此中断时，跳转的地址为相对于 mtvt 基地址偏移 (mtvt + 4*i) 处存储的字地址。

(R/W)

TRIG 配置第 i 个 CLIC 机器模式中断的触发类型和极性。

0x0：中断为正电平触发

0x1：中断为正边缘触发

0x2：中断为负电平触发

0x3：中断为负边缘触发

(R/W)

MODE 配置第 i 个 CLIC 中断的特权模式。

0x0：用户模式

0x3：机器模式

其他：保留，不可编程

(R/W)

Register 2.99. clicintctl[i] (0x20801003 + 4*i)



CTL[i] 配置第 i 个 CLIC 机器模式中断的级别和优先级。

- 由于在此 CLIC 实现中 CLICINTCTLBITS 参数为 3，因此在 clicintctl[i] 中实现的位数相同，因此此寄存器的低 5 位硬连线为 1。
- mcliccfg.MNLBITS 的配置值决定了 clicintctl[i] 中表示中断级别的高位数，而剩余的低位表示优先级。

(R/W)

2.9.3 核心本地中断 (CLINT)

2.9.3.1 概述

CLINT 是核心本地中断模块，包含软件中断、定时器中断、以及相应的配置寄存器。CLINT 中断源的 ID 如下表所示：

表 2.9-4. 核心本地中断源

ID	描述
3	机器模式软件中断
7	机器模式定时器中断

2.9.3.2 特性

- 2 个本地电平类型机器模式中断源，优先级可编程
- 存储器映射配置和状态寄存器
- 64 位定时器，支持中断、溢出标志和三种采样模式
- 软件中断

2.9.3.3 软件中断

机器模式软件中断通过设置或清除存储器映射寄存器 MSIP 来控制。

设置寄存器位 clicintie[3] 来在核心级别启用软件中断。

读取 CLIC 中断 3 的 clicintip 寄存器的相应位（即 clicintip[3] 位）可获取此中断的挂起状态。

2.9.3.4 定时器计数器和中断

设置 mtimectl 寄存器的 mtime_EN 位启用定时器计数器。

将 mtimecmplo 和 mtimecmphi 寄存器分别设置为与系统定时器进行比较的值的低 32 位和高 32 位。

CPU 提供两个 32 位本地存储器映射的寄存器 mtimeloadlo 和 mtimeloadhi，具有读/写访问权限，用于加载系统定时器的特定值。

当系统定时器值的 64 位值超过 mtimecmplo 和 mtimecmphi 的 64 位值时，机器模式中断触发。

读取 CLIC 中断 7 的 clicintip 寄存器的 IP 位可获取定时器中断的挂起状态。

若需取消挂起机器模式下的定时器中断，可以清除 CLIC 中断 7 的 clicintie 寄存器的 IE 位，或者更新 mtimecmplo 和 mtimecmphi 寄存器的值。

2.9.3.5 寄存器列表

本小节的所有地址均为相对于 CPU 子系统基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的图 9.3-1。

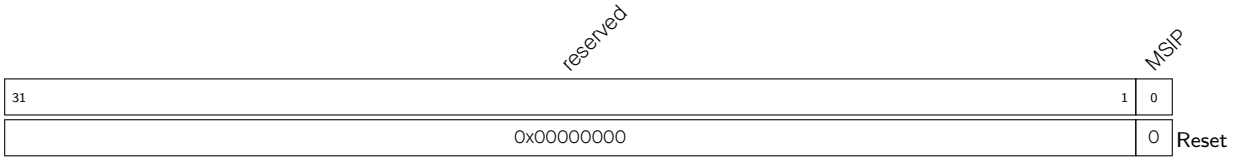
名称	描述	地址	访问
msip	核心本地机器软件中断挂起寄存器	0x0000	R/W
mtimecmplo	核心本地机器定时器比较值的低 32 位	0x4000	R/W

名称	描述	地址	访问
mtimecmphi	核心本地机器定时器比较值的高 32 位	0x4004	R/W
mtimeloadlo	核心本地机器定时器加载值的低 32 位	0x4008	R/W
mtimeloadhi	核心本地机器定时器加载值的高 32 位	0x400C	R/W
mtimectl	核心本地机器定时器中断控制/状态寄存器	0x4010	R/W
mtimelo	核心本地定时器计数值的低 32 位	0xBFF8	RO
mtimehi	核心本地定时器计数值的高 32 位	0xBFFC	RO

2.9.3.6 寄存器描述

本小节的所有地址均为相对于 CPU 子系统基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的图 9.3-1。

Register 2.100. msip (0x0000)



MSIP 配置机器软件中断的挂起状态。

0：未挂起

1：挂起

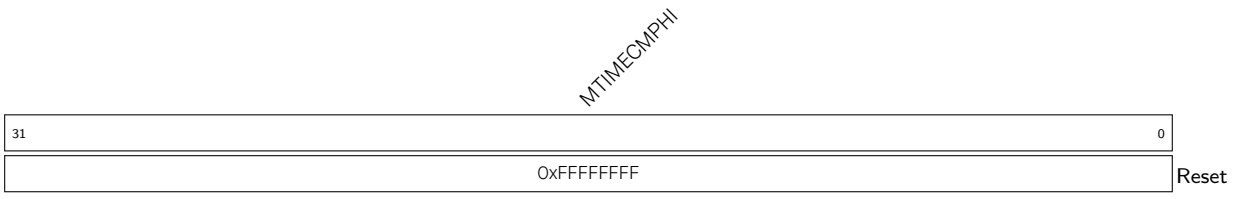
(R/W)

Register 2.101. mtimecmplo (0x4000)



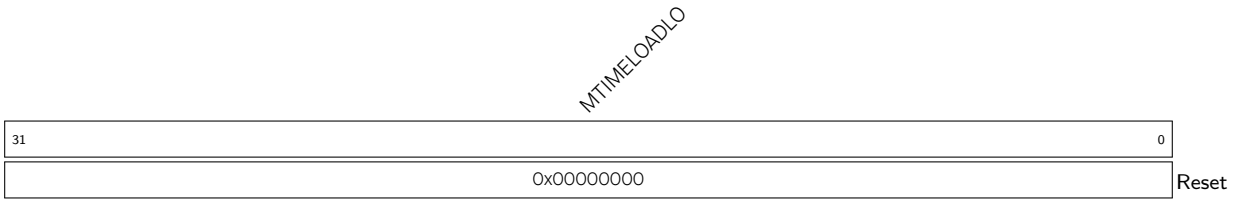
MTIMECMPLO 表示与系统计数器低 32 位进行比较的值。(R/W)

Register 2.102. mtimecmphi (0x4004)



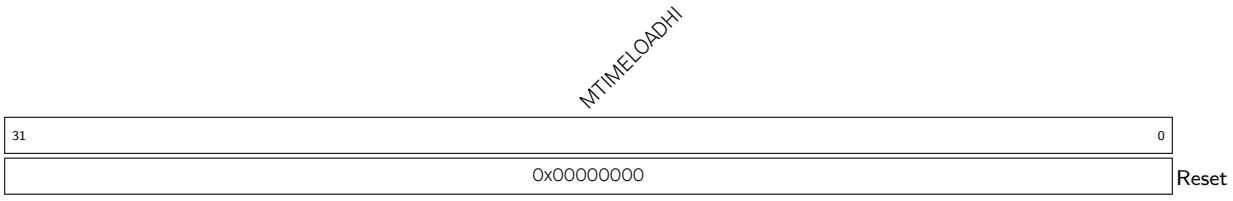
MTIMECMPHI 表示与系统计数器高 32 位进行比较的值。(R/W)

Register 2.103. mtimeloadlo (0x4008)



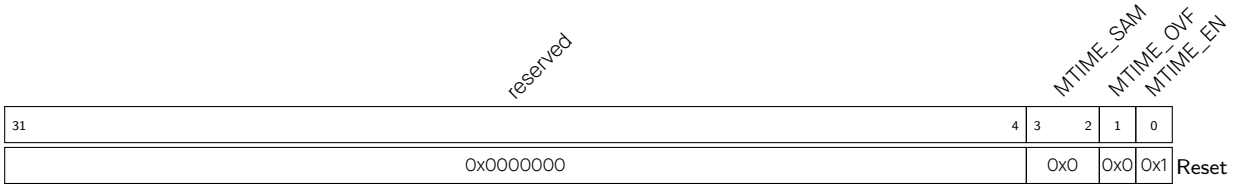
MTIME_LOADLO 表示要加载到系统计数器的低 32 位。(R/W)

Register 2.104. mtimeloadhi (0x400C)



MTIME_LOADHI 表示要加载到系统计数器的高 32 位。(R/W)

Register 2.105. mtimectl (0x4010)



MTIME_EN 配置是否启用系统计数器。

1：启用

0：禁用

仅核心 0 的 MTIMERCTL 寄存器实现了该位。写入核心 1 的 MTIMERCTL 的相应位对系统计时器无效。由于核心 1 可以访问核心 0 的 CLINT 寄存器，所以核心 1 可以通过核心 0 的 MTIMERCTL 寄存器运行或暂停计时器。

(R/W)

MTIME_OVF 配置系统计数器的溢出位。

1：表示系统计数器值为最大值，即 0xFFFFFFFFFFFFFFFF

0：其他情况

当计数器值为 0xFFFFFFFFFFFFFFFF，即最大值时，此位由硬件设置为 1。软件可以通过写入 0 来清除此位。写入 1 没有影响。

(R/W)

MTIME_SAM 配置 MTIMELO 和 MTIMEHI 寄存器的采样模式，以准确读取 64 位系统计数。在 64 位系统计数中，只有一半的数据可原子读取；因此，采样模式可以采样另一半的值，并将其存储在缓冲区中，并在下一次读取时，直接从缓冲区读取，而不是读取计数器中实际存储的数据。

0x0：无采样（默认）

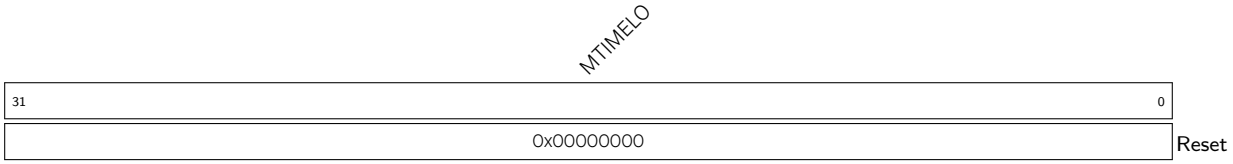
0x1：读取 MTIMELO 时采样系统计数的高 32 位

0x2：读取 MTIMEHI 时采样系统计数的低 32 位

0x3：读取 MTIMELO 或 MTIMEHI 时，采样系统计数的另一半

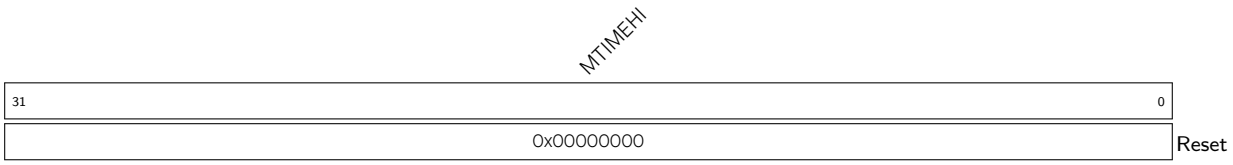
(R/W)

Register 2.106. mtimelo (0xBFF8)



MTIMELO 表示系统计数器当前低 32 位值。(RO)

Register 2.107. mtimehi (0xBFFC)



MTIMEHI 表示系统计数器当前高 32 位值。(RO)

2.10 存储器保护单元

每个 HP 核心都实现了 RISC-V 标准的物理存储器保护 (PMP) 方案和自定义物理存储器属性检查器 (PMAC)，用于防止未经授权的存储器访问。

2.10.1 标准物理存储器保护

2.10.1.1 概述

CPU 核心包含物理存储器保护 (PMP) 单元，完全符合《RISC-V 指令集手册 v1.10 第二卷 “特权架构”》。软件可以使用它为所需的存储器区域设置存储器访问特权（读、写和执行权限）。除了标准 PMP 检查外，CPU 核心还实现了自定义的物理存储器属性 (PMA) 检查器，可根据预定义的属性提供额外的权限检查。

2.10.1.2 特性

PMP 单元用于控制对物理存储器的访问，主要特性包括：

- 最多支持 32 个 PMP 表项
- 最小地址颗粒度为 128 字节
- 三种地址匹配模式：OFF、TOR（范围顶部）和 NAPOT（自然对齐的 2 的幂）。不支持 NA4（自然对齐的 4 字节区域）地址匹配模式
- 读、写和执行的权限控制
- 每个 PMP 表项均支持锁定功能
- 最大物理空间地址为 4 GB

2.10.1.3 功能描述

软件可以设置 PMP 单元的配置和地址寄存器，以保存错误并确保安全执行。配置 PMP CSR 可使能 PMP 单元，但只能在机器模式下进行配置，一旦 PMP 单元被使能，则将根据 16 个 pmpcfgX 和 pmpaddrX 寄存器（见 2.10.1.4）中的配置值对用户模式下的所有存储器访问进行写入、读取、访问权限检查。

默认情况下，PMP 允许机器模式下的所有存储器访问，而撤销用户模式下的所有访问。这意味着必须通过 pmpcfg 和 pmpaddr 寄存器（见 2.10.1.4）设置用户模式可以访问的地址范围和有效权限，以确保访问成功。但是在机器模式下没有此要求，因为机器模式下默认 PMP 允许所有访问。如果在机器模式下也需要 PMP 检查，则软件可以将所需 PMP 表项的锁定位置位来使能权限检查。锁定位一旦置位，就只能通过 CPU 复位被清零。

如果从没有执行权限的存储器区域获取指令，则处理器会生成异常，并且在 mcause CSR 中将异常原因设置为指令访问错误。同样，任何没有有效读/写权限的读写访问都将生成异常，mcause 会更新为读取存储器访问错误或写入存储器访问错误，并将存储器访问地址更新到 mtval CSR 中。

2.10.1.4 寄存器列表

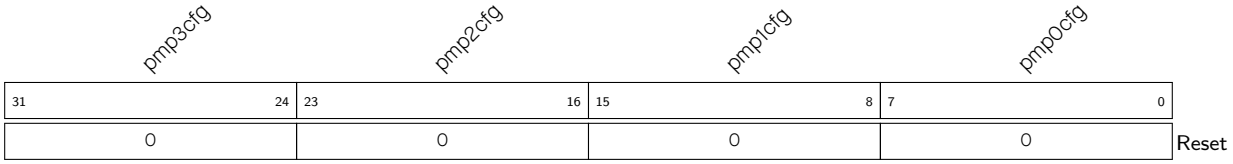
下表列出了 CPU 可访问的的 PMP CSR，只有在机器模式下才可以对它们进行读写。

名称	描述	地址	访问
pmpcfg0	物理存储器保护配置寄存器	0x3A0	R/W
pmpcfg1	物理存储器保护配置寄存器	0x3A1	R/W
pmpcfg2	物理存储器保护配置寄存器	0x3A2	R/W
pmpcfg3	物理存储器保护配置寄存器	0x3A3	R/W
pmpcfg4	物理存储器保护配置寄存器	0x3A4	R/W
pmpcfg5	物理存储器保护配置寄存器	0x3A5	R/W
pmpcfg6	物理存储器保护配置寄存器	0x3A6	R/W
pmpcfg7	物理存储器保护配置寄存器	0x3A7	R/W
pmpaddr*n* (*n*: 0-31)	物理存储器保护地址寄存器	0x3B0+0x1**n*	R/W

2.10.1.5 寄存器描述

PMP 单元实现了《RISC-V 指令集手册 v1.10 第二卷 “特权架构”》中定义的所有 pmpcfg0-3 和 pmpaddr0-15 CSR。

Register 2.108. pmpcfg0 (0x3A0)



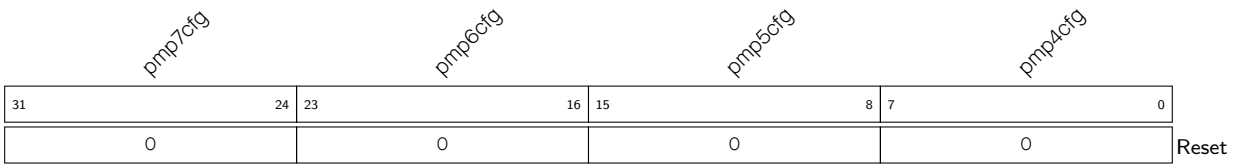
pmp3cfg PMP 表项 3 的配置寄存器。(R/W)

pmp2cfg PMP 表项 2 的配置寄存器。(R/W)

pmp1cfg PMP 表项 1 的配置寄存器。(R/W)

pmp0cfg PMP 表项 0 的配置寄存器。(R/W)

Register 2.109. pmpcfg1 (0x3A1)



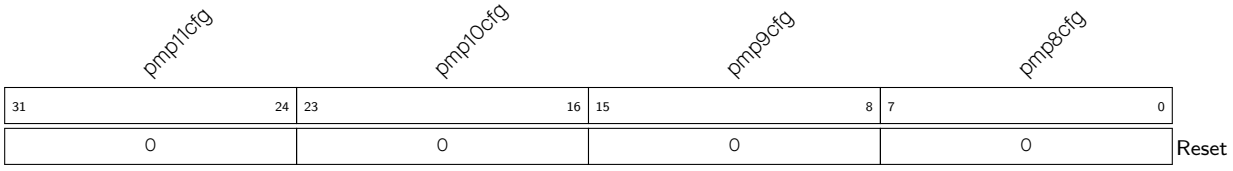
pmp7cfg PMP 表项 7 的配置寄存器。(R/W)

pmp6cfg PMP 表项 6 的配置寄存器。(R/W)

pmp5cfg PMP 表项 5 的配置寄存器。(R/W)

pmp4cfg PMP 表项 4 的配置寄存器。(R/W)

Register 2.110. pmpcfg2 (0x3A2)



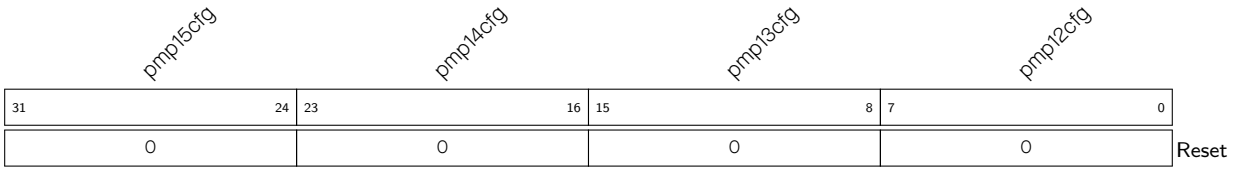
pmp11cfg PMP 表项 11 的配置寄存器。(R/W)

pmp10cfg PMP 表项 10 的配置寄存器。(R/W)

pmp9cfg PMP 表项 9 的配置寄存器。(R/W)

pmp8cfg PMP 表项 8 的配置寄存器。(R/W)

Register 2.111. pmpcfg3 (0x3A3)



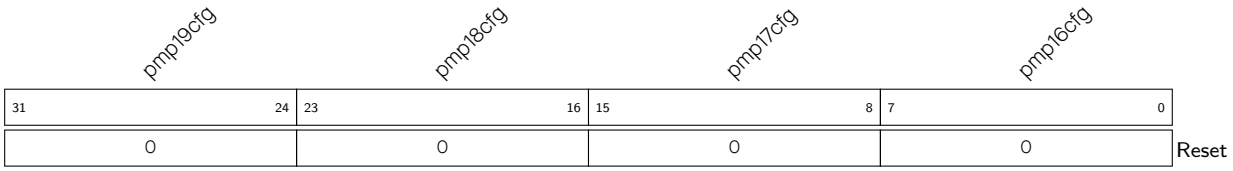
pmp15cfg PMP 表项 15 的配置寄存器。(R/W)

pmp14cfg PMP 表项 14 的配置寄存器。(R/W)

pmp13cfg PMP 表项 13 的配置寄存器。(R/W)

pmp12cfg PMP 表项 12 的配置寄存器。(R/W)

Register 2.112. pmpcfg4 (0x3A4)



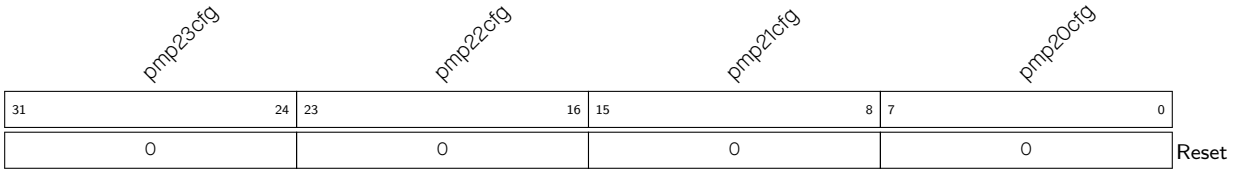
pmp19cfg PMP 表项 19 的配置寄存器。(R/W)

pmp18cfg PMP 表项 18 的配置寄存器。(R/W)

pmp17cfg PMP 表项 17 的配置寄存器。(R/W)

pmp16cfg PMP 表项 16 的配置寄存器。(R/W)

Register 2.113. pmpcfg5 (0x3A5)



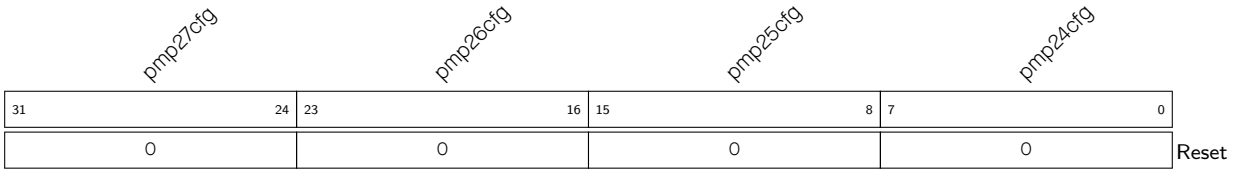
pmp23cfg PMP 表项 23 的配置寄存器。(R/W)

pmp22cfg PMP 表项 22 的配置寄存器。(R/W)

pmp21cfg PMP 表项 21 的配置寄存器。(R/W)

pmp20cfg PMP 表项 20 的配置寄存器。(R/W)

Register 2.114. pmpcfg6 (0x3A6)



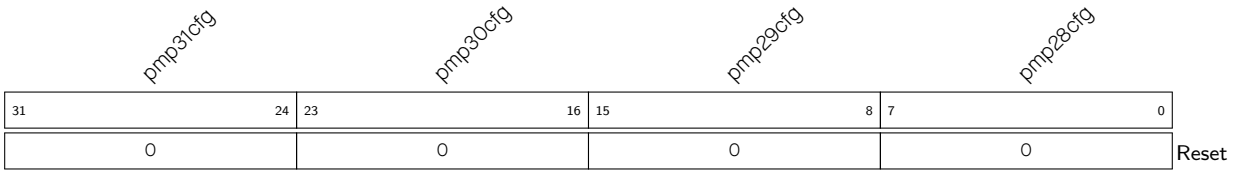
pmp27cfg PMP 表项 27 的配置寄存器。(R/W)

pmp26cfg PMP 表项 26 的配置寄存器。(R/W)

pmp25cfg PMP 表项 25 的配置寄存器。(R/W)

pmp24cfg PMP 表项 24 的配置寄存器。(R/W)

Register 2.115. pmpcfg7 (0x3A7)



pmp31cfg PMP 表项 31 的配置寄存器。(R/W)

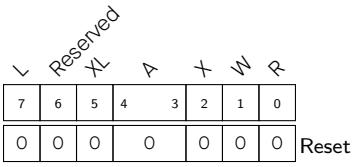
pmp30cfg PMP 表项 30 的配置寄存器。(R/W)

pmp29cfg PMP 表项 29 的配置寄存器。(R/W)

pmp28cfg PMP 表项 28 的配置寄存器。(R/W)

以下为 PMP 表项配置寄存器的格式。

Register 2.116. pmpXcfg



L 配置锁定位。一旦设置，只能通过核心复位来清除。(R/W)

XL 配置自定义锁定位。一旦设置，只能通过核心复位来清除。
0：PMP CSR 访问正常工作
1：对应的 pmpaddr 和 pmpcfg 项被锁定且无法修改
(R/W)

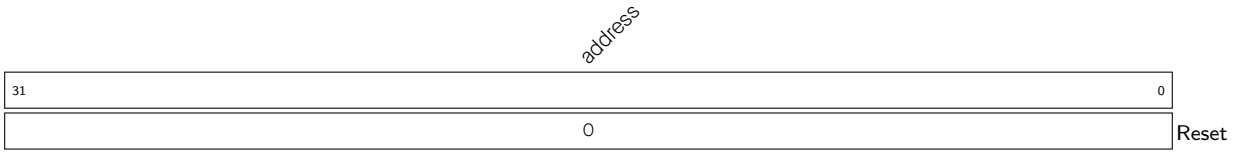
A 配置地址匹配模式。
0：关闭
1：TOR（范围顶端）
2：不支持
3：NAPOT（自然对齐的 2 的幂次方区域，大小 >= 128 字节）
(R/W)

X 配置执行权限。设置后，允许从地址范围执行。(R/W)

W 配置写入权限。设置后，允许对地址范围进行存储器写入。(R/W)

R 配置读权限。设置后，允许从地址范围进行存储器读取。(R/W)

Register 2.117. pmpaddr*n* (*n*: 0-31) (0x3B0+0x1**n*)



address 配置 pmpaddr*n* 的地址。(R/W)

2.10.2 自定义物理存储器属性 (PMA) 检查器

2.10.2.1 概述

CPU 核心还实现了自定义物理存储器属性检查器 (PMAC)，可根据 CSR 预先定义的存储器类型提供额外的权限检查。请注意，PMA 检查不区分核心的特权模式，即不区分机器和用户模式。

2.10.2.2 特性

PMAC 支持以下特性：

- 最多支持 16 个 PMA 表项
- 指定的存储器区域可配置存储器类型
- 最小地址颗粒度为 128 字节
- 三种地址匹配模式：OFF、TOR（范围顶部）和 NAPOT（自然对齐的 2 的幂）。不支持 NA4（自然对齐的 4 字节区域）地址匹配模式
- 读、写和执行的权限控制
- 每个 PMA 表项均支持锁定功能
- 最大物理空间地址为 4 GB
- 指定的存储器区域可配置属性

2.10.2.3 功能描述

软件可以设置 PMAC 单元的配置和地址寄存器，避免因访问无效存储器区域造成的错误。PMAC CSR 只能在机器模式下进行配置。一旦 PMAC 被使能，则将根据 16 个 pma_cfgX 和 pma_addrX 寄存器（见 2.10.2.4）中的配置值对各个模式下的所有存储器访问进行写入、读取、访问权限检查。访问被设置为无效的存储器的表项将导致获取错误或读写错误异常。

PMA 检查相关的的异常生成及错误处理与 PMP 检查类似。如果从配置为 null 或无效的存储器区域提取指令，则会在处理器级别生成异常，并且在 mcause CSR 中将异常原因设置为指令访问错误。同样，读写任何 null 或无效的存储器区域都将生成异常，并且 mcause 会更新为读取存储器访问错误或写入存储器访问错误。如果发生存储器读写异常，则存储器访问地址会更新到 mtval CSR 中。对于配置为有效存储器区域的 PMA 检查表项，处理方式与 PMP 检查相同。

如果软件想要关闭对 PMAC 寄存器的写入，可使用表项的锁定位。一旦任何 pma_cfgX 寄存器中的锁定位被设置，相应的 pma_cfgX 和 pma_addrX 寄存器就不能被写入，除非 CPU 复位。

PMAC CSR 中还提供了一个 4 位的域来定义存储器区域的属性。这些位不为 CPU 核心使用，而是基于地址匹配在读写接口上作为边带信号，供 cache 控制器用于其内部操作。

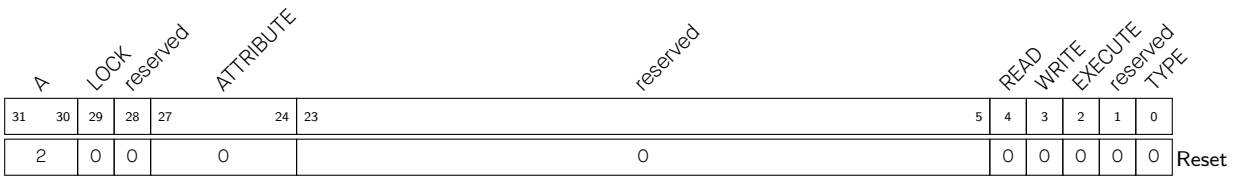
2.10.2.4 寄存器列表

下表列出了 CPU 可访问的的 PMA CSR，只有在机器模式下才可以对它们进行读写。

名称	描述	地址	访问
pma_cfg*n* (*n*: 0-15)	物理存储器属性配置寄存器	0xBC0+0x1**n*	R/W
pma_addr*n* (*n*: 0-15)	物理存储器属性地址寄存器	0xBD0+0x1**n*	R/W

2.10.2.5 寄存器

Register 2.118. pma_cfg*n* (*n*: 0-15) (0xBC0+0x1**n*)



A 配置地址类型。功能与 pmpcfg 寄存器的 A 字段相同。

0x0：OFF

0x1：TOR

0x2：NA4

0x3：NAPOT

(R/W)

LOCK 配置是否锁定对应的 pma_cfgX 和 pma_addrX。

0：不锁定

1：锁定，即撤销写入 pma_cfgX 和 pma_addrX 的权限

只有 CPU 复位才能解锁。

(R/W)

ATTRIBUTE 配置要在 DRAM 属性端口上驱动的值。

bit 27：

0：可缓存 (cacheable)

1：不可缓存 (non-cacheable)

bit 26：

0：写回（write-back）

1：写直达（write-through）

bit 25：

0：写失效分配（write miss allocate）

1：写失效不分配（write miss no allocate）

bit 24：0：读失效分配（read miss allocate）

1：读失效不分配（read miss no allocate）

(R/W)

READ 配置相应区域的读取权限。

0：不允许读取

1：允许读取

(R/W)

WRITE 配置相应区域的写权限。

0：不允许写入

1：允许写入

(R/W)

见下页……

Register 2.118. pma_cfg*n* (*n*: 0-15) (0xBC0+0x1**n*)

接上页……

EXECUTE 配置相应区域的执行权限。

0：不允许执行

1：允许执行

(R/W)

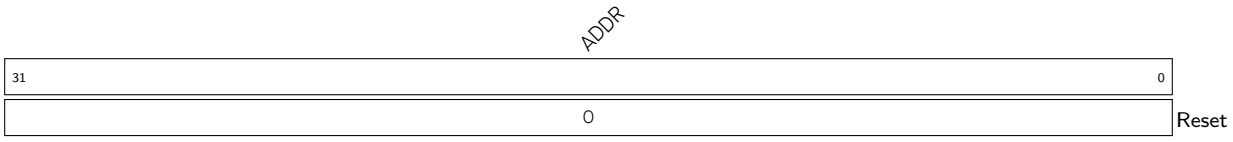
TYPE 配置区域类型。

0x0：存储器区域无效（RWX 访问将被视为 0，即使配置为 1）

0x1：存储器区域有效（RWX 访问可用）

(R/W)

Register 2.119. pma_addr*n* (*n*: 0-15) (0xBD0+0x1**n*)



ADDR 配置地址。功能与 pmpaddr 寄存器相同。(R/W)

2.11 调试和追踪

2.11.1 调试

2.11.1.1 概述

本节介绍如何调试在 HP 和 LP CPU 核心上运行的软件。调试功能由标准 JTAG 管脚提供，并符合 《RISC-V 外部调试支持规范 v0.13.2》 。

HP 核心和 LP 核心各自配备了独立的 JTAG DTM（调试传输模块），用于连接其对应的调试模块（Debug Module）。这两个 JTAG DTM 通过菊花链（daisy chain）串联至一个 JTAG 接口，调试器可通过该接口调试所需核心，并且在调试过程中，可以根据需求将未使用的 DTM 模块保持在旁路状态。两个 HP 核心共享一个调试模块 (DM)，该调试模块连接至 JTAG DTM (HP Cores) 模块，而 LP 核心使用单独的调试模块，该模块连接至 JTAG DTM (LP Core)。

请注意，由于 HP 核心和 LP 核心通过不同的调试模块访问，因此二者无法同时进行调试。

图 2.11-1 显示了外部调试支持的主要组件。

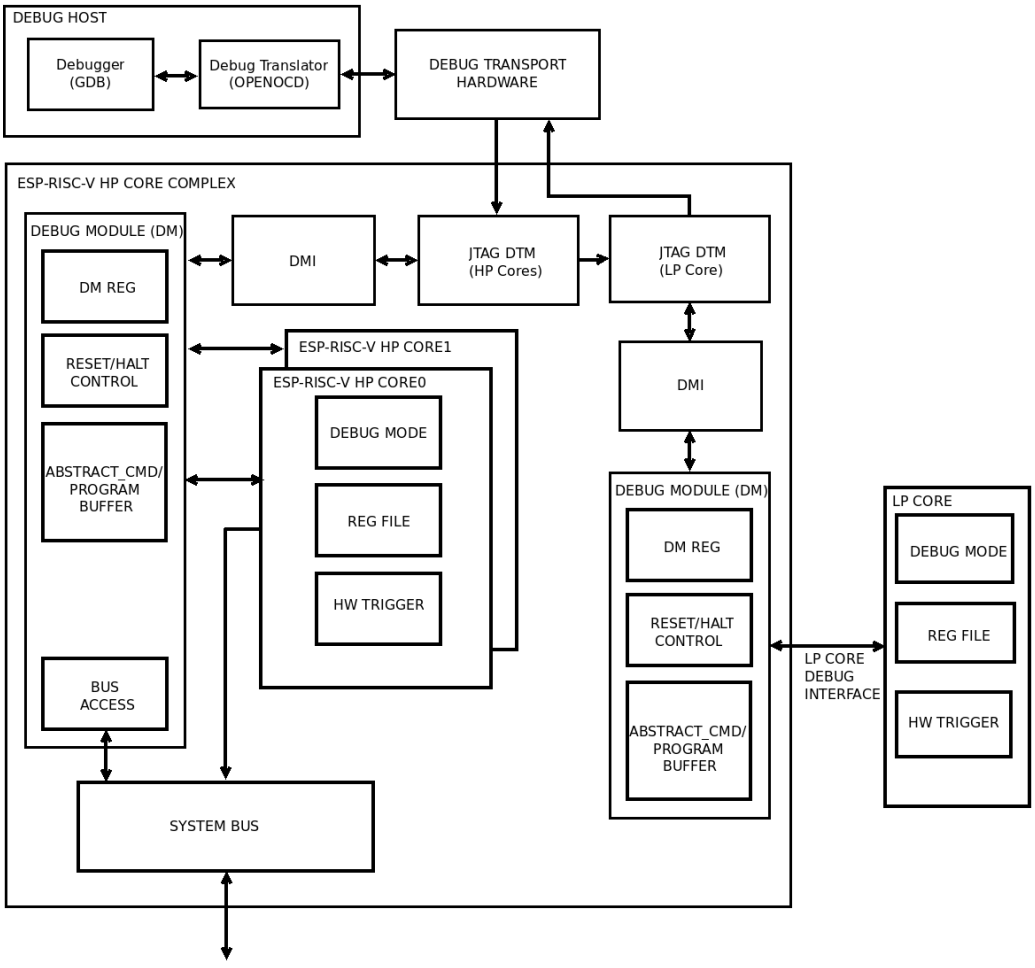


图 2.11-1. 调试系统概述

本节重点介绍 HP 核心的调试功能，有关 LP 核心的调试功能，请参阅章节 5 低功耗处理器。

用户与运行调试器（Debugger，例如 GDB）的调试主机（DEBUG HOST，例如笔记本电脑）进行交互。调试器通过调试转换器（Debug Translator，可能包含硬件驱动，例如 OPENOCD）与调试传输硬件（DEBUG

TRANSPORT，例如 ESP-Prog 适配器）进行通信。调试传输硬件通过标准 JTAG 接口将调试主机连接到 HP 核心的调试传输模块 (JTAG DTM)，绕过 LP 核心 DTM。JTAG DTM 使用调试模块接口 (DMI) 提供对调试模块 (DM) 的访问。

DM 允许调试器暂停选中的 HP 核心。抽象命令提供对 GPR（通用寄存器）的访问。程序缓冲区允许调试器在核心上执行任意代码，从而读取 CPU 核心的其他运行状态。CPU 核心的其他运行状态也可以由其他抽象命令读取。

每个 HP 核心的触发器模块支持 3 个触发器。当满足触发条件时，核心将自发暂停并通知调试模块。

系统总线访问模块可以在不影响任一 HP 核心的情况下访问存储器和外设寄存器。

2.11.1.2 功能

HP CPU 核心支持以下基本调试功能：

- 向调试器提供有关实现的必要信息
- 支持暂停和恢复核心
- 核心寄存器（包括 CSR）可以由调试器读取/写入
- 可以从复位后执行的第一条指令开始调试核心
- 可以通过调试器复位核心
- 可以通过软件断点指令暂停核心
- 硬件单步调试
- 通过程序缓冲区在暂停的核心中执行任意指令。支持 2 个字的程序缓冲区
- 支持系统总线的字对齐地址访问
- 每个 HP 核心支持 3 个硬件触发器（可用作断点/观察点），详情见章节 2.11.3
- LP 核心可调试，通过菊花链串联 LP 核心的 JTAG DTM 来实现

2.11.1.3 功能描述

如前所述，调试机制遵守 《RISC-V 外部调试支持规范 v0.13.2》 。有关功能操作的详细信息，请参阅该规范。

2.11.1.4 JTAG 控制

标准的 JTAG 接口是 DTM 访问 DM 的唯一途径。硬件提供两种 JTAG 方式：PAD_to_JTAG 和 USB_to_JTAG。

- PAD_to_JTAG：指 JTAG 信号源来自芯片 IO 管脚
- USB_to_JTAG：指 JTAG 信号源来自 USB Serial/JTAG 控制器

使用哪种 JTAG 方法取决于许多因素，具体配置方法如下表所示。

表 2.11-1. JTAG 控制

暂时关闭 JTAG [3,4]	EFUSE_DIS_USB_SERIAL_JTAG [4]	EFUSE_DIS_USB_JTAG [4]	EFUSE_DIS_PAD_JTAG [4]	EFUSE_JTAG_SEL_ENABLE [4]	Strapping 管脚 GPIO34 [5]	USB JTAG 状态	PAD JTAG 状态
0	0	0	0	0	x [2]	可用 [1]	不可用 [1]
0	0	0	0	1	1	可用	不可用
0	0	0	0	1	0	不可用	可用
0	0	0	1	x	x	可用	不可用
0	0	1	0	x	x	不可用	可用
0	0	1	1	x	x	不可用	不可用
0	1	0	0	0	x	不可用	不可用
0	1	0	0	1	0	不可用	可用
0	1	0	0	1	1	不可用	不可用
0	1	0	1	x	x	不可用	不可用
0	1	1	0	x	x	不可用	可用
0	1	1	1	x	x	不可用	不可用
1	x	x	x	x	x	不可用	不可用

说明：

1. 可用：相应的 JTAG 功能可用
   不可用：相应的 JTAG 功能不可用
2. x：无关项
3. “暂时关闭 JTAG” 表示如果 EFUSE_SOFT_DIS_JTAG[2:0] 中有偶数位 “1”，则开启 JTAG 功能（表中对应值为 1），否则，JTAG 功能被关闭（表中对应的值为 0）。但是，在 ESP32-P4 HMAC 加速器的某些特殊条件下，即使 EFUSE_SOFT_DIS_JTAG[2:0] 中有奇数位 “1”，JTAG 功能也可能被开启。有关 HMAC 如何影响 JTAG 功能的信息，请参阅 HMAC Accelerator 章节。
4. 有关 eFuse 的更多信息，请参考 eFuse Controller 章节。
5. 有关 strapping 管脚 GPIO34 的更多信息，请参考 Chip Boot Control 章节。

2.11.1.5 寄存器

以下是 HP 核心支持的调试 CSR 列表：

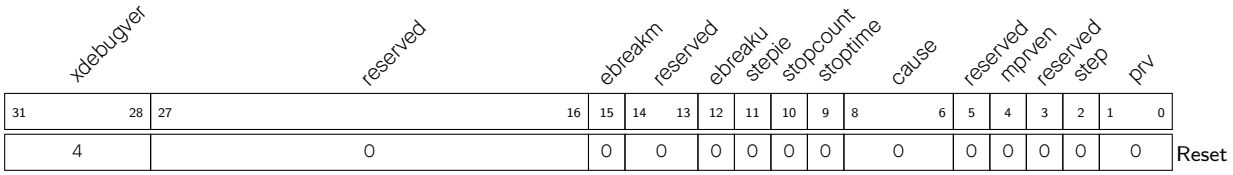
名称	描述	地址	访问
dcsr	调试控制和状态寄存器	0x7B0	R/W
dpc	调试 PC 寄存器	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试暂存寄存器 1	0x7B3	R/W

所有的实现的调试模块寄存器均符合《RISC-V 外部调试支持规范 v0.13.2》。更多信息，请参阅该规范。

2.11.1.6 寄存器描述

以下是 HP 核心支持的调试 CSR 的详细信息：

Register 2.120. dcsr (0x7B0)



xdebugver 表示调试版本。

4：存在外部调试支持

(RO)

ebreakm 当为 1 时，机器模式下的 ebreak 指令进入调试模式。

(R/W)

ebreaku 当为 1 时，用户/程序模式下的 ebreak 指令进入调试模式。

(R/W)

stepie 配置调试模式下的中断控制。

0：单步执行时禁用中断

1：单步执行时启用中断

调试器在 hart 运行时不得更改此位的值。

(R/W)

stopcount 配置调试模式下的 mcycle 计数器。

0：累加计数器

1：在调试模式或 ebreak 指令执行期间（并导致进入调试模式），不累加任何计数器，包括 cycle 和 instret CSR

(R/W)

stoptime 配置调试模式下的计时器。

0：累加计时器

1：在调试模式下不累加任何 hart 本地计时器

(R/W)

cause 配置进入调试模式的原因。当单个周期中有多个原因导致进入调试模式，则写入优先级最高的原因。

1：执行了 ebreak 指令（优先级 3）

2：触发模块导致暂停（优先级 4，最高）

3：haltreq 被设置（优先级 1）

4：由于 step 被设置，CPU 核心单步执行（优先级 0，最低）

5：由于 resethaltreq，hart 在复位后暂停（优先级 2）

其他值：保留，供未来使用

(RO)

mprven 配置在调试模式下是否使能 mstatus CSR 中的 MPRV 功能。

0：禁用

1：使能

(R/W)

见下页……

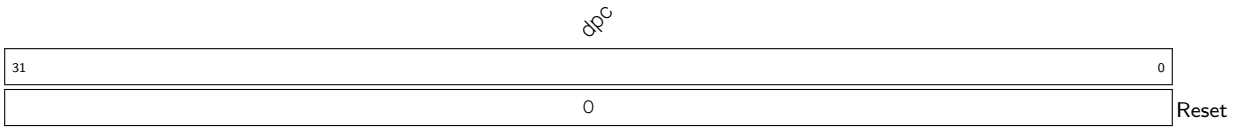
Register 2.120. dcsr (0x7B0)

接上页……

step 当被置位且不处于调试模式时，核心将仅执行单个指令，然后进入调试模式。
如果指令由于异常而未能完成，则核心将在执行异常处理程序之前立即进入调试模式，并置位相应的异常寄存器。
设置该位不会屏蔽中断，这与 RISC-V 外部调试支持规范 v0.13 中的规定不同。
(R/W)

prv 保存核心进入调试模式时的特权级别。退出调试模式时，调试器可以更改此值以改变核心的特权级别。仅支持 0x3（机器模式）和 0x0（用户模式）。(R/W)

Register 2.121. dpc (0x7B1)

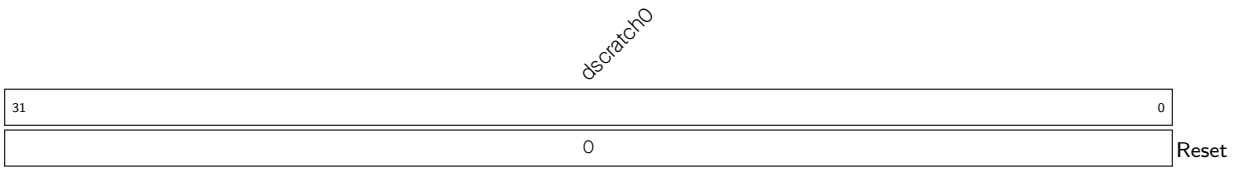


dpc 进入调试模式后，dpc 将写入下一条指令的虚拟地址。恢复执行时，CPU 核心的 PC 将更新为 dpc 保存的虚拟地址。调试器可以写入 dpc 来配置 CPU 恢复执行的位置。(R/W)

表 2.11-3. 进入调试模式时 DPC 中的虚拟地址

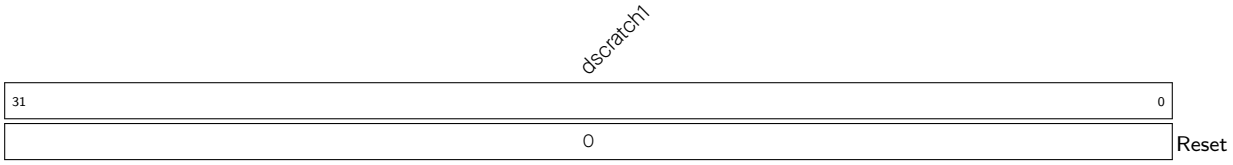
原因	DPC 中的虚拟地址
ebreak	ebreak 指令的地址
单步执行	如果没有调试进行，将要执行的下一条指令的地址，即对于不改变程序流的 32 位指令为 pc+4，跳转/分支为目标 PC 等。
触发模块	如果 timing 为 0，则为导致触发的指令的地址。如果 timing 为 1，则为进入调试模式时要执行的下一条指令的地址。
暂停请求	进入调试模式时要执行的下一条指令的地址。

Register 2.122. dscratch0 (0x7B2)



dscratch0 由调试模块内部使用。(R/W)

Register 2.123. dscratch1 (0x7B3)



dscratch1 由调试模块内部使用。(R/W)

2.11.2 调试暂停组

2.11.2.1 概述

在多核心系统中，当调试软件在某个核心上运行时，其他核心最好不要改变系统状态。这可以通过同步暂停和恢复其他核心来实现，并且需要快速实现。因此，最好不要通过主机上运行的调试器软件的命令来实现，而是通过芯片内部的架构来实现。

2.11.2.2 特性

- HP 核心之间可交叉触发
- 覆盖核心的 RunStall 功能

2.11.2.3 功能描述

HP 核心调试模块实现了《RISC-V 外部调试规范 v1.0-STABLE》（即将发布）中的 dmcs2 寄存器，以支持基于 hart 组的同时暂停/恢复。

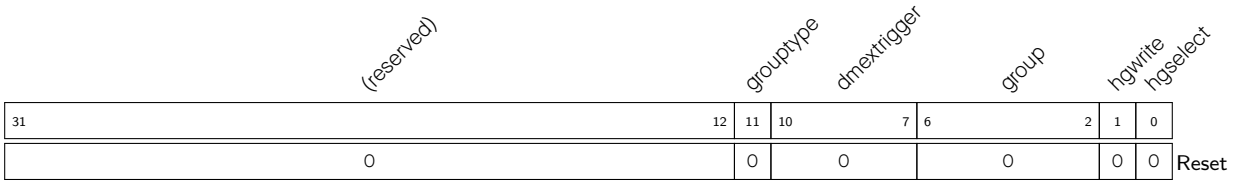
2.11.2.4 寄存器列表

下表包含调试模块内必须实现的控制寄存器。

名称	描述	地址	访问
dmcs2	调试模块控制和状态寄存器 2	0x32	R/W

2.11.2.5 寄存器描述

Register 2.124. dmcs2 (0x32)



grouptype 配置组的类别。0：将组配置为暂停 (halt) 组，且此寄存器中的其余域用于配置暂停组
1：将组配置为恢复 (resume) 组，且此寄存器中的其余域用于配置恢复组
(R/W)

dmexttrigger 配置当前选择的 DM 外部触发器。如果写入的触发器不存在，硬件将该字段更改为有效值。如果不存在 DM 外部触发器，则将其设置为 0。
(R/W)

group 当 hgselect 为 0 时，此域包含 hartsel 指定的 hart 组。
当 hgselect 为 1 时，此域包含 dmexttrigger 选择的 DM 外部触发器组。
仅当 hgwrite 为 1 时，写入此域的值才有效。
组号为一串从 0 开始的连续数字，最大编号具体取决于实现，不同类型的组的最大编号也可能不同。调试器写入编号后，应读取此域，以确认使用的是否为支持的 hart 组。如果组未被实现，则整个域为 0。
(R/W)

hgwrite 当写入 1 且 hgselect 为 0 时，对于每个选定的 hart，如果硬件支持该 hart 的组，DM 将其组更改为写入的组值。实现也可以以相同方式更改一组未选定 hart 的最小集合的组，如果由于硬件限制这是必要的。当写入 1 且 hgselect 为 1 时，如果触发器硬件支持 dmexttrigger 中设置的组，则 DM 将该触发器的组更改为 dmexttrigger 设置的组。写入 0 无效。
(R/W)

hgselect 固定为零。
0：操作 HART
(RO)

2.11.3 硬件触发器

2.11.3.1 概述

2.11.3.2 特性

每个 HP 核心实现一个硬件触发器模块，提供调试的断点和监视点功能。每个硬件触发器均支持以下特性：

- 3 个独立的可配置触发单元，可匹配程序计数器地址或加载/存储访问地址
- 可以通过引起断点异常来抢占执行
- 可以暂停执行并将控制权转移给调试器
- 支持 NAPOT（2 的幂次方对齐区域）地址编码

2.11.3.3 功能描述

硬件触发器模块提供了 7 个 CSR，见 寄存器列表。其中，tdata1 和 tdata2 是抽象 CSR，也就是说它们是用于访问某个触发单元中的内部寄存器的影子寄存器，一次访问一个触发单元。

要选择特定的触发单元，需要将相应的编号 (0-2) 写入 tselect CSR。当写入有效数值时，抽象 CSR tdata1 和 tdata2 将自动匹配该触发单元的内部寄存器。每个触发单元都有两个内部寄存器，即 mcontrol 和 maddress，它们分别与 tdata1 和 tdata2 匹配。

向 tselect 写入超过最大编号的数值时会导致该数值被裁剪为最大的编号，此编号可以被读回。这个特性可用于枚举初始化期间或使用调试器时可用的触发器。

由于软件或调试器可能需要知道所选触发器的类型以便正确解读 tdata1 和 tdata2，因此 tdata1 的 4 个位 (31-28) 对所选触发器的类型进行了编码。此域为只读访问属性，并且值始终为 0x2，代表匹配类型触发器，因此，可以推断 tdata1 和 tdata2 会通过 mcontrol 和 maddress 被解读。《RISC-V 外部调试支持规范 v0.13.2》提供了其他可能值的信息，但是该触发模块仅支持 0x2 类型。

一旦选定了触发单元，就可以通过置位 mcontrol CSR (tdata1) 中相应的域并将目标地址写入 maddress CSR (tdata2) 来对该触发单元进行配置。

通过写入 mcontrol 的 action 域，可以将每个触发单元配置为引起断点异常或进入调试模式。该域只能从调试器写入，因此默认情况下，触发器（如果启用）将引起断点异常。

每个触发单元的 mcontrol 都有一个 hit 域。在 CPU 暂停或进入异常后，通过读取该域可以查明是否是触发单元触发了。触发器触发后该域会立即被置位，但在恢复操作之前必须被手动清零，不过不清零不会影响正常执行。

每个触发单元仅支持地址匹配，该地址可以是存储器访问地址，也可以是指令的虚拟地址。通过写入所选触发单元的 maddress (tdata2) CSR，可以指定区域的地址和大小。大于 1 个字节的区域大小通过 NAPOT 编码（见表 2.11-5）指定，并通过置位 mcontrol 中 match 域来使能。注意，根据定义，NAPOT 编码地址的起始地址需与区域大小对齐（即，是区域大小的整数倍），最大支持 128 字节。

表 2.11-5. NAPOT 编码的 maddress

maddress(31-0)	起始地址	大小（字节）
aaa...aaaaaaaaa0	aaa...aaaaaaaaa0	2
aaa...aaaaaaaa01	aaa...aaaaaaaa00	4

aaa...aaaaaaa011	aaa...aaaaaaa000	8
aaa...aaaaaa0111	aaa...aaaaaa0000	16
....		
aaa...0111111111	aaa...aaa0000000	2^8

tcontrol CSR 对所有触发单元都是通用的。在机器模式下，当陷阱处理程序运行时，该寄存器可用于阻止触发器重复引起异常，而且默认会禁用 ISR 内部的断点异常，但是，出于调试目的，可以在进入 ISR 之前手动使能断点异常。如果将触发器配置为进入调试模式，则此 CSR 不相关。

2.11.3.4 触发执行流程

当触发器触发引起 hart 暂停并进入调试模式时 (action = 1)：

- dpc 在解码阶段被设置为当前程序计数器 (PC)
- dcsr 的 cause 域被设置为 2，表示暂停是由于触发器触发引起
- 与触发的触发器对应的 hit 域被置位

当触发器触发引起 hart 进入陷阱时 (action = 0)：

- mepc 在解码阶段被设置为当前 PC
- mcause 被设置为 3，即断点异常
- mpte 被设置为陷阱发生之前的 mte 的值
- mte 被设置为 0
- 与触发的触发器对应的 hit 域被置位

说明：如果两个触发器同时触发，一个 *action = 0*，另一个 *action = 1*，则 *hart* 会暂停并进入调试模式。

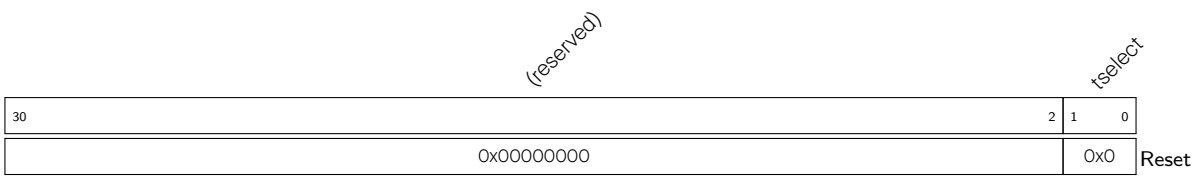
2.11.3.5 寄存器列表

下表列出了 CPU 可访问的的触发模块 CSR，只有在机器模式下才可以对它们进行读写。

名称	描述	地址	访问
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
tdata3	触发抽象数据寄存器 3	0x7A3	R/W
tinfo	触发信息寄存器	0x7A4	RO
tcontrol	全局触发控制寄存器	0x7A5	R/W
mcontext	触发上下文寄存器	0x7A8	R/W

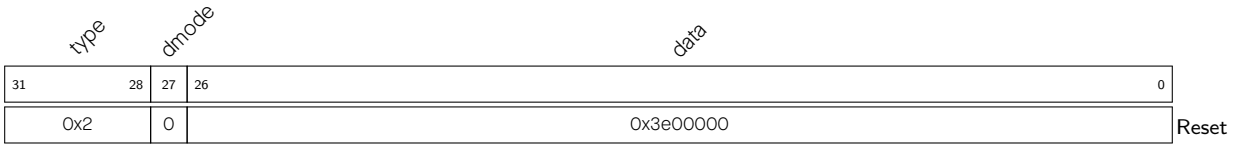
2.11.3.6 寄存器描述

Register 2.125. tselect (0x7A0)



tselect 配置触发器单元编号 (0-3)。(R/W)

Register 2.126. tdata1 (0x7A1)



type 表示触发器类型。仅支持匹配类型 (0x2)，此域保留。(RO)

dmode 如果某触发器正在被调试器使用，则此域置为 1。

0：在调试模式和机器模式下都能写入 tdata1 和 tdata2

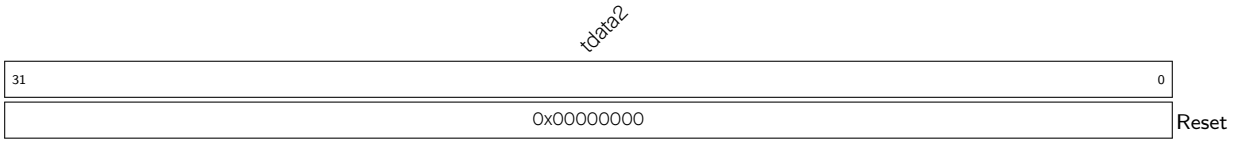
1：只有在调试模式下才能写入 tdata1 和 tdata2，其他模式下的写操作将被忽略

注意：仅支持调试模式下的写操作

(R/W)

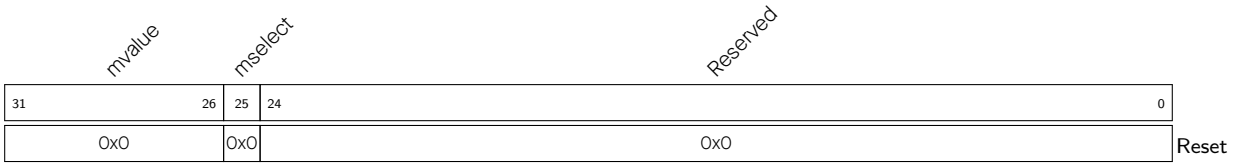
data 配置抽象 tdata1 的内容。由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 mcontrol 的域。(R/W)

Register 2.127. tdata2 (0x7A2)



tdata2 配置抽象 tdata2 的内容。由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 maddress。(R/W)

Register 2.128. tdata3 (0x7A3)

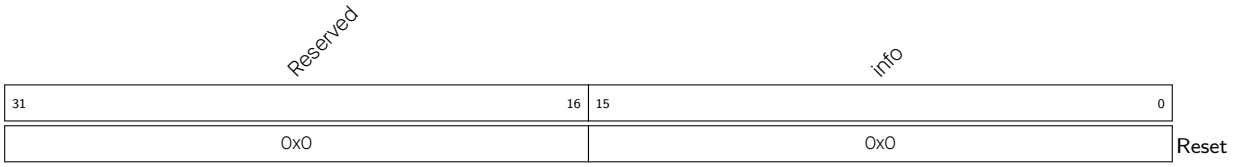


mvalue 与 mselect 一起使用的数据。(R/W)

mselect 0：忽略 mvalue
1：仅当 mcontext 的低位等于 mvalue 时，此触发器才会匹配。
(R/W)

Reserved 读取为 0。(RO)

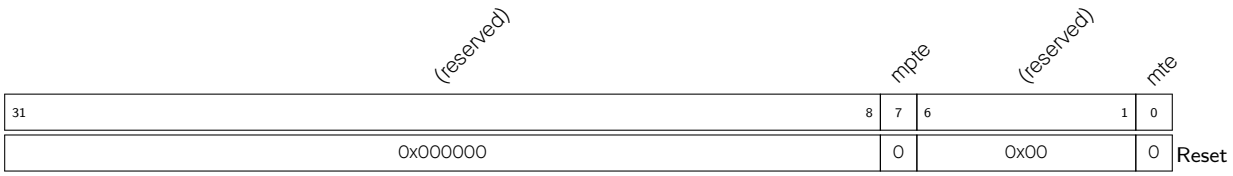
Register 2.129. tinfo (0x7A4)



Reserved 读取为 0。(RO)

info 该域的每一位与 tdata1 中列举的每种可能类型一一对应，位 N 对应于类型 N。如果该位被设置，则当前选择的触发器支持该类型。如果当前选择的触发器不存在，则此域为 1。如果类型不可写，则此寄存器可能未实现，在这种情况下读取它会导致非法指令异常。在这种情况下，调试器可以从 tdata1 读取唯一支持的类型。(RO)

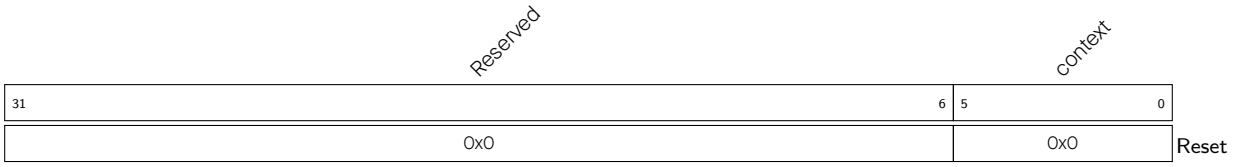
Register 2.130. tcontrol (0x7A5)



mpte 配置是否使能机器模式下前一个触发器。
当 CPU 在机器模式下进入异常，mte 的值会自动写入此域。
当 CPU 执行 MRET，此域的值会返回 mte，此域变为 0。
(R/W)

mte 配置是否使能机器模式下触发器。当 CPU 在机器模式下进入异常，此域的值会自动写入 mpte，然后此域变为 0，并且 action=0 的触发器被全局禁用。
当 CPU 执行 MRET，mpte 的值会自动返回此域。
(R/W)

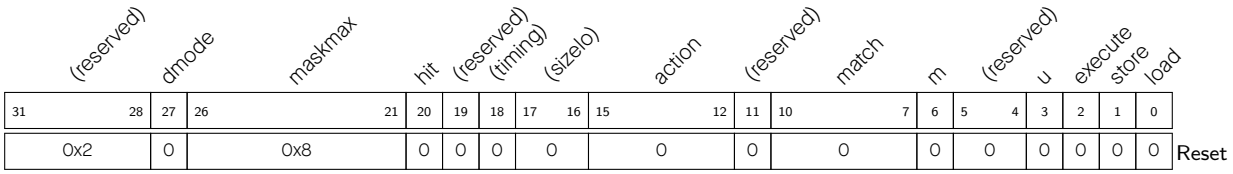
Register 2.131. MCONTEXT (0x7A8)



Reserved 读取为 0。(RO)

context 可在 M 模式和调试模式下可写入。机器模式软件可以将上下文编号写入此寄存器，用于设置仅在该上下文中激活触发器。(R/W)

Register 2.132. mcontrol (0x7A1)



(reserved)	dmode	maskmax	hit	(reserved)	timing	sizelo	action	(reserved)	match	m	(reserved)	u	execute	store	load
31 28	27	26 21	20	19	18	17 16	15 12	11	10 7	6	5 4	3	2	1	0
0x2	0	0x8	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

dmode 与 tdata1 中的 dmode 相同。(RW*)

maskmax 表示当 match 为 1 时，支持的最大自然二次幂 (NAPOT) 范围。始终读取为 0x8，这意味着只能屏蔽最低的 8 位。(RO)

hit 如果所选触发器先前已触发，则此值为 1。此位需要手动清除。(R/W)

timing 设置此域以启用加载/存储触发器动作的时序。

0：此触发器的动作将在触发它的指令执行之前执行，但在所有前面的指令提交之后执行。

1：此触发器的动作将在触发它的指令执行之后执行，且应在下一条指令执行之前执行。

(R/W)

sizelo 此域包含访问地址的最低两位。

0x0：触发器将尝试匹配任何大小地址的访问

0x1：触发器将尝试匹配 8 位存储器访问

0x2：触发器将尝试匹配 16 位存储器访问

0x3：触发器将尝试匹配 32 位存储器访问

(R/W)

action 配置所选触发器在触发时执行动作。有效选项为：

0x0：引发断点异常

0x1：进入调试模式（仅在 dmode=1 时有效）

0x2：在触发器 0 命中时启用追踪

0x3：在触发器 1 命中时启用追踪

0x4：在触发器 2 命中时启用追踪

注意：写入无效值将设置为默认值 0x0。

(R/W)

match 配置触发器进行数据/指令地址的匹配操作。有效选项为：

0x0：严格字节匹配，即与访问中某个字节对应的地址必须严格匹配 maddress 的值

0x1：NAPOT 匹配，即访问中至少有一个字节处于 maddress 中规定的 NAPOT 区域

注意：写入超过最大值的数值会被裁剪为最大值 0x1。

(R/W)

见下页……

Register 2.132. mcontrol (0x7A1)

接上页……

m 置位使选定的触发器在机器模式下操作。(R/W)

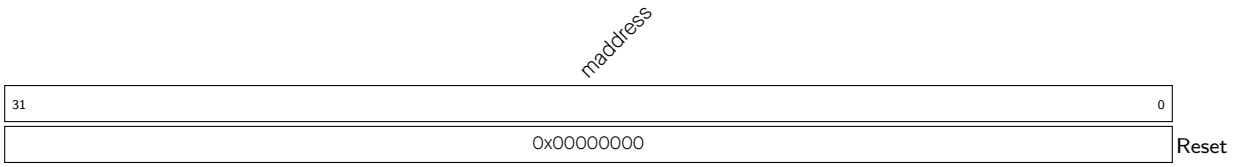
u 置位使选定的触发器在用户模式下操作。(R/W)

execute 置位使选定的触发器在 CPU 执行具有匹配的虚拟地址的指令之前触发。(R/W)

store 置位使选定的触发器在 CPU 执行具有匹配的数据地址的存储器写操作之前触发。(R/W)

load 置位使选定的触发器在 CPU 执行具有匹配的数据地址的存储器读操作之前触发。(R/W)

Register 2.133. maddress (0x7A2)



maddress 配置选定的触发器执行匹配操作时使用的地址。当 mcontrol 中的 match=1 时由 NAPOT 解码。(R/W)

2.11.4 追踪

2.11.4.1 概述

为了支持非侵入式软件调试，CPU 核心提供了指令追踪接口，为离线调试提供相关信息。追踪编码器 (Trace Encoder) 模块可通过指令追踪接口获取这些信息，并将其压缩后存储在分配的存储器中。软件解码器可直接从存储器中读取这些信息，无需中断 CPU 核心，便可重新生成 CPU 核心实际执行的程序。

2.11.4.2 特性

CPU 核心支持指令追踪功能，并按照 RISC-V 处理器追踪规范 v1.0 (RISC-V Processor Trace v1.0) 中的规定向追踪编码器提供以下信息：

- 退役指令数量
- 异常和中断是否发生及其原因、陷阱值
- hart 当前的特权级别
- 跳转、分支和返回退役指令的指令类型
- 程序计数器更改之前和之后退役指令的地址
- 根据 mcontrol 中的 action 位在触发命中时启用追踪

2.11.4.3 功能描述

每个 HP 核心实现了强制指令 delta 追踪，也称为分支追踪。它的工作原理是通过发送程序获取的 delta 信息来追踪从已知起始地址开始的执行情况。delta 通常由跳转、调用、返回、分支类型的指令、中断以及异常引入。所有此类 delta 以及其他详细信息，如原因和实际指令/地址，都通过高带宽指令追踪接口从 CPU 输出给追踪编码器，追踪编码器对这些信息进行操作，将信息压缩后存入存储器，供解码器离线调试。有关编码的更多信息，请参阅章节 3 *RISC-V 追踪编码器 (TRACE)*。

CPU 没有任何内部寄存器来控制指令追踪接口，所有相关控制寄存器都在 3 *RISC-V 追踪编码器 (TRACE)* 模块中。

2.12 性能

2.12.1 分支预测

2.12.1.1 概述

在分支指令执行前，核心会预测分支是否跳转以及跳转的目标地址。然后，基于预测的结果，从相应的 PC 获取指令。在执行分支时，验证预测是否正确，如果预测错误，则刷新流水线，并从正确的 PC 开始获取指令。预测能提高吞吐量，因为核心会根据预测结果继续获取指令，而不是等待分支指令执行完成。下面的章节会详细描述核心使用的预测技术。

2.12.1.2 特性

- 预测条件/无条件分支类型指令的结果和目标地址
- 支持分支目标预测，指令包括 BEQ、BNE、BLT、BLTU、BGE、BGEU、C.BEQZ、C.BNEZ、JAL、J、CJ
- 支持条件分支预测，指令包括：BEQ、BNE、BLT、BLTU、BGE、BGEU、C.BEQZ、C.BNEZ
- 使用 2 位饱和计数器，提高准确性
- 使用条件分支的全局历史进行预测，提高性能
- 对于无条件分支，目标预测在一个时钟周期内完成，即当预测或推测正确时，0 周期惩罚

2.12.1.3 功能描述

核心使用分支历史表 (BHT) 和分支目标缓冲区 (BTB) 来预测分支指令和跳转指令的行为。

BHT 是一个 512x16 位 SRAM 存储器，分别存在于两个核心中。在执行预测时，使用虚拟分支历史寄存器 (VBHR) 作为索引，而在更新或纠正分支类型指令执行结果（跳转或不跳转）时，由全局分支历史寄存器 (GBHR) 作为索引。GBHR 包含前 13 条分支类型指令执行结果，VBHR 包含前 12 条结果和当前分支指令的预测结果。因此，这两个寄存器都保存了先前执行的分支轨迹。基于 VBHR/GBHR 的 13 位，BHT 中存储的 2 位饱和计数器值分别被读取或写入。

如图 2.12-1 所示，2 位饱和计数器表示分支是否为：

- 弱不跳转（weakly not taken，00）
- 强不跳转（strongly not taken，01）
- 弱跳转（weakly taken，10）
- 强不跳转（strongly taken，11）

如果预测结果为弱不跳转或强不跳转，则预测分支为不跳转。如果为弱跳转或强跳转，则为跳转。

在将分支执行的实际结果更新至 BHT 时，会使用以下逻辑更新相应的饱和计数器。

BTB 存储 32 位分支目标地址，最多可存储 16 条地址，以条件或无条件分支类型指令 PC 的最后 16 位作为索引。因此，核心根据 BHT 来预测分支类型指令的结果，如果预测为跳转，跳转到 BTB 中的目标地址。对于无条件分支，核心直接跳转到 BTB 中的目标地址，因为这种情况不需要进行预测。如果预测错误，核心在执行过程中，会获得分支类型指令的实际结果和目标地址，同时也会刷新流水线，跳转到正确的地址，并更新 BTB 和 BHT 表项。

图 2.12-1 详细介绍了每个 HP 核心中使用的 2 位饱和方案。

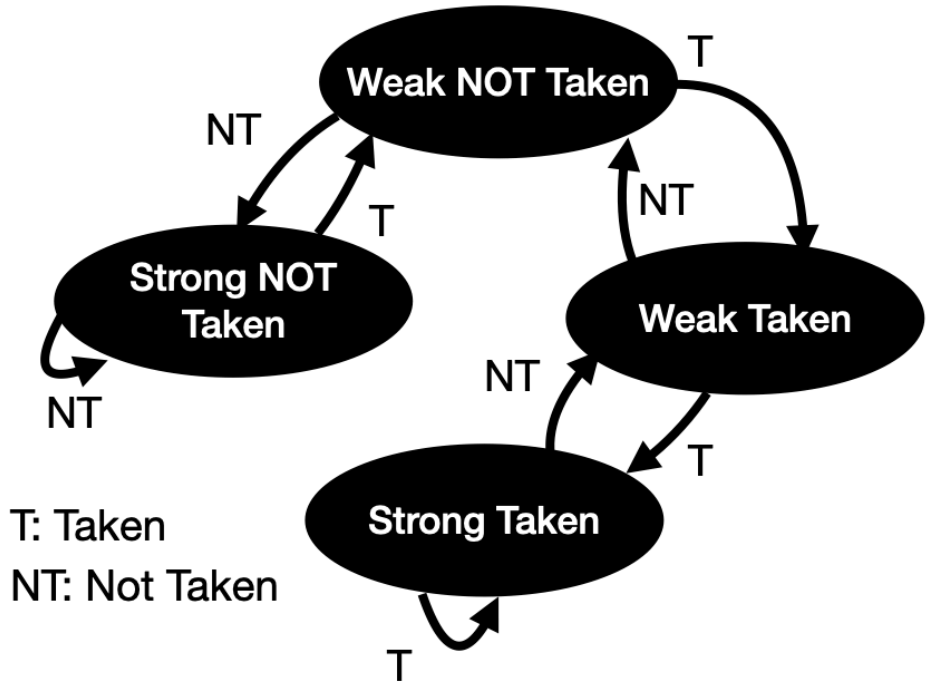


图 2.12-1. 2 位饱和计数器

2.12.2 RAS

2.12.2.1 概述

返回地址栈 (RAS) 是核心维护的一个栈，用于推送跳转和链接（JAL/JALR）类型指令的返回地址。因此，核心可以在 RET 指令被预解码时立即跳转到返回地址，从而提高函数调用返回的性能。

2.12.2.2 特性

- 避免函数调用返回时流水线停顿，提高核心性能
- 最多可存储四级函数调用嵌套

2.12.2.3 功能描述

当核心预解码 JAL/JALR/C.JALR 指令时，下一条指令的 32 位 PC 会被推送到 RAS。

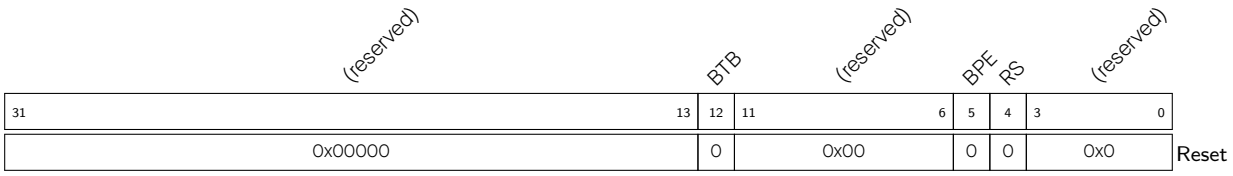
当核心预解码返回类型指令 (JALR, C.JR, C.JALR) 时，地址从 RAS 中弹出。然后，核心开始从弹出的地址获取指令。当核心执行返回指令时，如果弹出的地址与链接寄存器中存储的地址不匹配，则刷新流水线并从链接寄存器中的地址开始获取指令。RAS 最多可以存储四个返回地址，即支持四级函数调用嵌套。

RAS 预测取决于以下条件：

目标寄存器 (Rd)	源寄存器 (Rs)	RAS 操作
rd != x1	rs1 != x1	无操作
rd != x1	rs1 == x1	弹出
rd == x1	rs1 != x1	推入
rd == x1	rs1 == x1	推入并弹出

2.12.3 性能配置控制状态寄存器

Register 2.134. mhcr (0x7C1)



RS 配置是否启用返回地址栈逻辑进行预测。

0：禁用

1：启用

(R/W)

BPE 配置是否启用条件分支预测。

0：禁用

1：启用

(R/W)

BTB 配置是否启用分支目标缓冲区。

0：禁用

1：启用

(R/W)

2.13 自定义特性

2.13.1 总线错误响应

2.13.1.1 概述

在 HP 核心中，加载/存储总线错误被处理为异步异常，这意味着在检测到此类异常时，CPU 可能已经执行了引起异常的指令之后的部分指令。

在处理同步异常时，标准 CSR mcause、mepc 和 mtval 提供的上下文足够。但是，对于异步异常来说，这些信息并不够。因此，需要额外的 CSR 来给软件提供程序所需的必要上下文，来处理和恢复遇到异步总线错误异常的程序。

下一小节将介绍这些总线错误 CSR 以及获取总线错误异常上下文的推荐方法。

2.13.1.2 功能描述

有两种方法来处理加载/存储总线错误：

- 同步总线错误模式：
  当启用此模式时（通过设置 mhint.SBE 位），总线错误异常可以像其他同步异常一样进行处理。因此，执行会在发生总线错误的 PC 处触发异常，而且 mepc 持有相同的 PC。
  启用此模式的缺点是增加了所有加载/存储指令的延迟。
- 异步总线错误模式（默认）：
  - 为了在异步模式下处理总线错误，建议（但不是必须）在启动时清除 mexstatus.NMFT 位。
  - 在异步模式下，当（一个或多个）加载/存储指令导致总线错误时，CPU 会进入陷阱处理程序，MCAUSE 值会更新为 5 或 7（取决于导致错误的操作类型），mepc 保存本应在该加载/存储指令之后执行的指令的 PC。
  - 进入原因 5 或 7 的异常处理程序时，必须检查 mexstatus.BUS_ERR 位，确认是否是由于加载/存储总线错误造成的异常。如果是，则必须清除此位进行确认。
  - 然后，必须检查位 ldpc0.vld、ldpc1.vld、stpc0.vld、stpc1.vld 和 stpc2.vld 是否有效。如果有效，则必须清除这些位，并从这些 CSR 中读取相应的地址域（ldpc0.addr、ldpc1.addr、stpc0.addr、stpc1.addr 和 stpc2.addr），以获取这些故障加载/存储指令执行时的程序计数器。ldtval0、ldtval1、sttval0、sttval1 和 sttval2 保存了加载/存储指令相对应的存储器访问地址。注意，序号较低的 CSR 记录了最新的总线错误，序号较高的 CSR 保存了更早的总线错误。
  - 注意，即使原因是 5 或 7，加载和存储总线错误 CSR 都必须进行检查，因为有时有多个加载/存储指令（或未对齐访问被拆分为多个加载/存储操作）已执行并正在进行中。在首次遇到总线错误之前进入陷阱处理程序时，CPU 会完成所有挂起的存储器操作，并记录它们引起的任何总线错误。因此，检查所有 CSR 非常重要，因为这可以捕获和处理所有此类异常。
  - 注意，跟同步总线错误异常相比，异步总线错误异常发生后，恢复正常的程序执行更加复杂，因为程序可能执行了问题指令之后的很多指令。

2.13.1.3 控制状态寄存器

名称	描述	地址	访问
mexstatus	机器扩展控制和状态寄存器	0x7E1	R/W
mhint	机器隐式操作寄存器	0x7C5	R/W
ldpc0	加载总线错误 PC 寄存器 0	0xBE0	R/W
ldpc1	加载总线错误 PC 寄存器 1	0xBE1	R/W
ldtval0	加载总线错误访问地址寄存器 0	0xBE8	R/W
ldtval1	加载总线错误访问地址寄存器 1	0xBE9	R/W
stpc0	存储总线错误 PC 寄存器 0	0xBF0	R/W
stpc1	存储总线错误 PC 寄存器 1	0xBF1	R/W
stpc2	存储总线错误 PC 寄存器 2	0xBF2	R/W
sttval0	存储总线错误访问地址寄存器 0	0xBF8	R/W
sttval1	存储总线错误访问地址寄存器 1	0xBF9	R/W
sttval2	存储总线错误访问地址寄存器 2	0xBFA	R/W

2.13.2 专用 IO

2.13.2.1 概述

GPIO 通常为 APB 外设，这意味着对输出的更改和对输入的读取，可能会卡在写缓冲区或其他传输之后，并且速度通常较慢，因为 APB 总线的运行速度通常低于 CPU。作为替代方案，CPU 核心实现了 I/O 处理器特定的 CPU 寄存器 (CSR)，直接连接到 GPIO 矩阵或 IO 管脚。这些寄存器只需一条指令即可访问，因此十分迅速。

2.13.2.2 特性

- 8 个直接映射到 GPIO 的专用 IO
- 驱动输出端口无延迟
- 读取输入值有两个 CPU 周期的延迟

2.13.2.3 功能描述

CPU 核心有一组 8 个输入和输出（管脚值 + 管脚输出使能），这些输入和输出端口直接连接到 GPIO 矩阵，可通过 GPIO 矩阵映射到最上层的管脚。更多相关信息请参考 10 *GPIO 交换矩阵和 IO MUX*。

CPU 自定义了 3 个 CSR：

- GPIO_IN 只读，表示输入值。
- GPIO_OUT 可读可写，表示 GPIO 输出值。
- GPIO_OEN 可读可写，表示 GPIO 输出使能状态，控制管脚方向，拉高表示管脚配置为输出模式，拉低表示配置为输入模式。

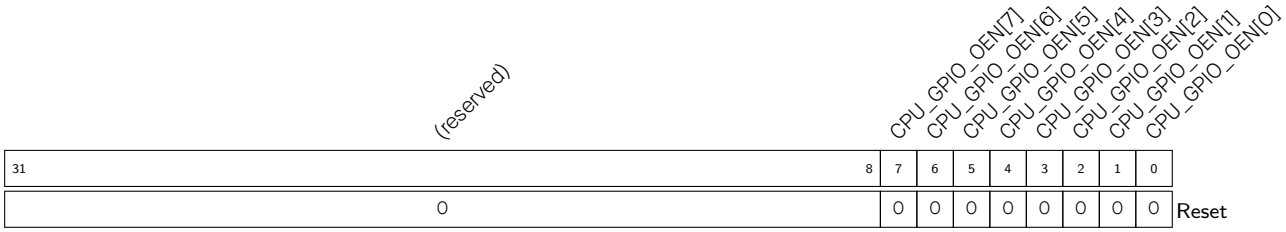
2.13.2.4 寄存器列表

下面是在核心中自定义的专用 IO CSR 的列表。

名称	描述	地址	访问
cpu_gpio_oen	GPIO 输出使能寄存器	0x803	R/W
cpu_gpio_in	GPIO 读输入值寄存器	0x804	RO
cpu_gpio_out	GPIO 写输出值寄存器	0x805	R/W

2.13.2.5 寄存器

Register 2.135. cpu_gpio_oen (0x803)



CPU_GPIO_OEN 配置是否使能 GPIOn (n=0 ~ 21) 输出。CPU_GPIO_OEN[7:0] 分别对应章节 *GPIO 交换矩阵和 IO MUX* 中表 10.12-1 里的 cpu_gpio_out_oen[7:0] 输出使能信号。CPU_GPIO_OEN 的值与 cpu_gpio_out_oen 的值对应。

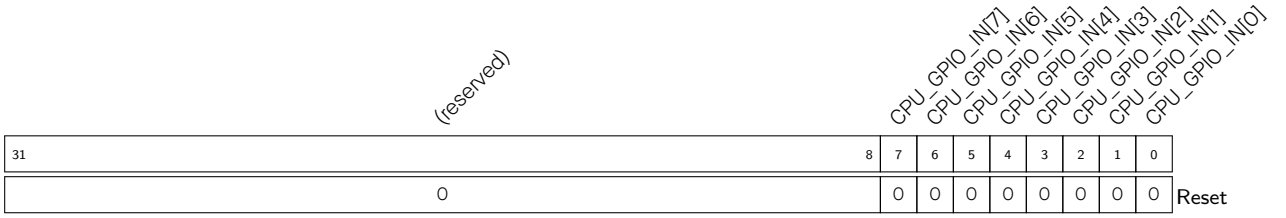
此寄存器是 CPU_GPIO_OUT 的使能寄存器。

0：关闭 GPIO 输出

1：使能 GPIO 输出

(R/W)

Register 2.136. cpu_gpio_in (0x804)



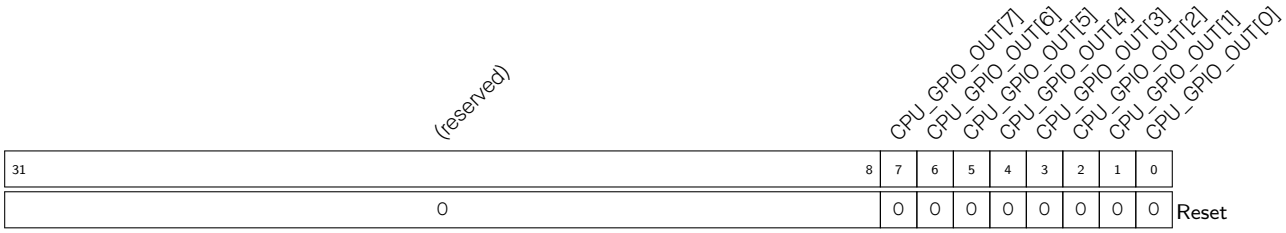
CPU_GPIO_IN 表示 SoC GPIOn (n=0 ~ 21) 的输入值（1 为高电平，0 为低电平）。

CPU_GPIO_IN[7:0] 分别对应章节 *GPIO 交换矩阵和 IO MUX* 中表 10.12-1 里的 cpu_gpio_in[7:0] 输入信号。

CPU_GPIO_IN[7:0] 只能通过 GPIO 交换矩阵映射到 GPIO。详细描述请参考章节 10.4。

(RO)

Register 2.137. cpu_gpio_out (0x805)



CPU_GPIO_OUT 向 SoC GPIOn (n=0 ~ 21) 写输出值（1 为高电平，0 为低电平）。CPU_GPIO_OEN 置位时，写输出值才有效。

CPU_GPIO_OUT[7:0] 分别对应章节 *GPIO 交换矩阵和 IO MUX* 中表 10.12-1 里的 cpu_gpio_out[7:0] 输出信号。

CPU_GPIO_OUT[7:0] 只能通过 GPIO 交换矩阵映射到 GPIO。详细描述请参考章节 10.5。

(R/W)

2.13.3 运行暂停 (RunStall)

2.13.3.1 概述

运行暂停 (RunStall) 信号允许外部代理暂停处理器，同时关闭大部分时钟树，以节省运行功耗。

RunStall 输入信号允许外部逻辑暂停处理器的内部流水线。RunStall 输入信号激活后，流水线中正在进行的所有指令将完成运行。流水线成功暂停后，核心将输出握手信号 corestalled，表示成功暂停 CPU 流水线。

2.13.3.2 功能描述

RunStall 输入信号可用于以下两种场景：

- 在复位后初始化指令和数据 RAM。如果在复位期间激活 RunStall，并在复位移除后仍然保持激活状态，那么处理器将处于暂停状态。可以初始化指令和数据 RAM，在初始化完成后，可以释放 RunStall，允许处理器开始执行代码。这种机制可以让处理器的复位向量指向本地指令 RAM 中的地址，并且复位代码可以在处理器开始执行代码之前，由外部代理写入指令 RAM。
- 冻结处理器的内部流水线。激活 RunStall 减少了处理器的活动功耗，也无需使用中断和 WAITI 选项。然而，使用 RunStall 信号所节省的功耗没有 WAITI 节省得那么多，因为 RunStall 逻辑需要处理器的某些部分在流水线暂停时，仍然保持活跃状态。

图 2.13-1 显示了 RunStall 和 CoreStalled 信号的时序图以及 CPU 核心状态的切换。

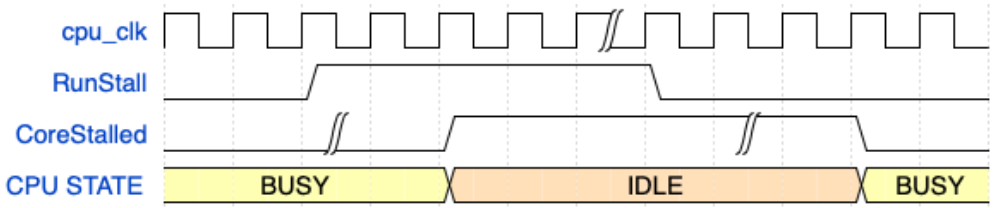


图 2.13-1. 使用 RunStall 时 CPU 核心状态的切换

2.13.3.3 寄存器列表

章节 21.2.1.10 中的 SoC 寄存器可用于激活 HP 核心的 RunStall。

2.13.4 调试辅助信息

HP CPU 核心系统向 ESP32-P4 中的调试辅助模块提供了额外的调试信息。详细信息请参阅章节 22 辅助调试。

2.13.5 核心锁定

2.13.5.1 概述

目前，RISC-V 编程模型中没有定义异常使能寄存器。而且，从异常服务程序返回到正常程序流时，MCAUSE 寄存器中的异常向量 ID 未被清除。因此，软件开发者无法仅通过编程模型中定义的寄存器，轻松判断 CPU 当前是在处理中断异常，还是在执行正常的程序流程。为了解决这一问题，CPU 实现了一个自定义的 LOCKUP 域，可以通过自定义 CSR 读取。

2.13.5.2 功能描述

当 CPU 响应异常时，会通过 MEXSTATUS 域中的 EXPT_VLD 确定当前是否正在运行异常服务程序。如果 CPU 正在处理异常，并且从异常服务程序返回之前又触发了新的异常，CPU 将被锁定。当 CPU 被锁定时：

- 锁定信号会被设置为高，来通知 SoC CPU 处于锁定状态
- CPU 停止获取和执行指令
- PC 停留在触发锁定的指令上
- MEXSTATUS 寄存器中的 LOCKUP 域置为 1

在锁定状态下，调试器仍然可以介入并执行调试命令。建议在检测到锁定状态时重置核心。

锁定规则有一些例外，当 ECALL 或 EBREAK 引起的异常嵌套在任何异常类型中时，不会触发 CPU 锁定。

2.13.5.3 寄存器列表

mexstatus.LOCKUP 位表示单个核心的锁定状态。

2.13.5.4 寄存器描述

请参阅 mexstatus CSR。

2.13.6 外部唤醒

2.13.6.1 概述

每个核心均设有一个专用唤醒输入端口，其功能由 SoC 中的寄存器组进行控制与配置。

2.13.6.2 功能描述

为降低功耗，每个 CPU 核心可执行 WFI 指令进入睡眠。睡眠唤醒源有三种：中断、暂停请求，以及专用唤醒输入端口信号。该专用端口仅需维持一个时钟周期的高电平脉冲，便可唤醒相应的核心。

端口的具体配置方法，请参见章节 21 系统寄存器 (SYSREG)。

第 3 章

RISC-V 追踪编码器 (TRACE)

ESP32-P4 的高性能 CPU (HP CPU) 通过追踪编码器 (Trace Encoder) 实现了对指令追踪接口的支持。追踪编码器连接 CPU 的指令追踪接口，将信息压缩成更小的数据包，存入内部 SRAM（详见章节 9 系统和存储器）中。

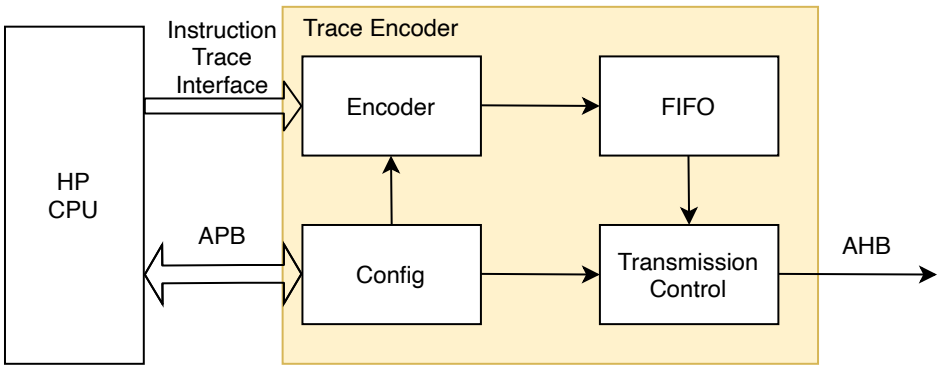


图 3.0-1. 追踪编码器概述图

3.1 术语

为了更好地说明 RISC-V 追踪编码器的功能，本章使用了以下术语。

hart	RISC-V 硬件线程
分支 (branch)	在一定条件下改变执行流程的指令
无法推测的不连续地址 (uninferable discontinuity, or updiscon)	无法仅通过程序二进制文件推测出的程序计数器地址偏移
增量 (delta)	指令在存储器中的存储地址不连续时，程序计数器的地址偏移
陷阱 (trap)	异常或中断，会触发陷阱处理程序
筛选 (qualification)	指令符合筛选条件，即称为通过筛选，才会被追踪
te_inst	编码器发送的数据包的名称
退役 (retire)	机器状态更新时，执行指令的最后一个阶段
EPC	异常程序计数器

3.2 介绍

对于复杂的系统来说，了解程序执行流程并非易事。造成软件不按预期工作的可能有很多种因素，比如与其他核、外设、实时事件的交互，执行不畅，或以上所有因素的结合。

调试器很难实时监测系统运行中的程序执行流程，因为调试器是侵入性的，会破坏系统的运行状态。但是，监测程序执行又很重要。

此时，可使用指令追踪来追踪程序执行。

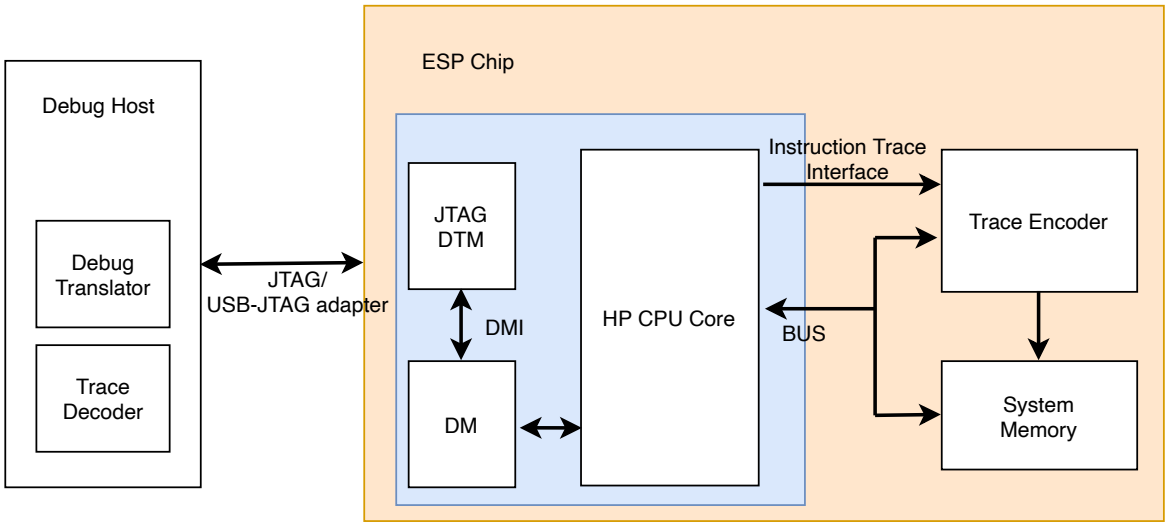


图 3.2-1. 指令追踪流程概览

图 3.2-1 为指令增量追踪的流程。每个 HP 核都有一个单独的编码器，可以独立工作，追踪流程相同：

- HP CPU 核有一个指令追踪接口，可以输出 HP CPU 执行指令的信息，包括指令地址、指令类型等。更多关于 ESP32-P4 HP CPU 指令追踪接口的信息，可参考章节 2 高性能处理器。
- 追踪编码器通过 HP CPU 的指令追踪接口获取相关指令追踪信息，将信息压缩成带宽更小的数据包，存入系统存储器中。
- 调试器（即图中的 Debug Host）可以将系统存储器中的追踪数据包通过 JTAG 或 USB Serial/JTAG 导出，然后用解码器解压缩、重建程序的执行流。追踪解码器通常是外部 PC 上的软件，在获取追踪数据包后，根据在 hart 上运行的程序二进制数据，重建程序的指令流。解码可以离线进行，也可以在 hart 执行时实时进行。

3.3 特性

- 兼容 Efficient Trace for RISC-V Version 2.0（RISC-V 高效追踪规范 v2.0），实现的参数详见表 3.3-1
- 支持增量地址模式和完整地址模式
- 支持过滤器
- 支持通过调试触发器或过滤器报告指令地址
- 支持下列边带信号 (sideband signals) 控制追踪数据流
  - 调试触发器启动或关闭编码器
  - hart 暂停时，编码器在报告最后一个数据包后停止工作
  - hart 复位后，编码器在报告最后一个数据包后停止工作

  - FIFO 即将变满时暂停 hart
- 支持任意地址范围用作追踪存储器
- 可配置的同步模式：
  - 同步计数器按包计数
  - 同步计数器按周期计数
  - 关闭同步计数器
- 支持丢包状态标识
- 支持丢包后自动重启
- 写追踪存储器时支持循环和非循环模式
- 具有两个中断：
  - 包的大小超过配置的存储器空间时触发中断
  - 丢包时触发中断
- 具有 128 × 8 位 FIFO，用于缓存数据包
- 支持 AHB 突发传输，突发长度可配置

表 3.3-1. 追踪编码器参数

参数名称	值	描述
arch_p	0	追踪器符合 RISC-V 追踪规范初始版本
bpred_size_p	0	不支持分支预测模式
cache_size_p	0	不支持跳转目标缓存模式
call_counter_size_p	0	不支持隐式返回模式
ctype_width_p	0	ctype 总线宽度
context_width_p	0	context 总线宽度
ecause_width_p	6	异常原因宽度
ecause_choice_p	0	不支持多选
f0s_width_p	0	不支持格式 0 数据包
filter_context_p	0	不支持过滤上下文信息
filter_excint_p	1	支持过滤异常原因或中断
filter_privilege_p	1	支持过滤特权
filter_tval_p	1	支持过滤陷阱值
iaddress_lsb_p	1	支持压缩指令
iaddress_width_p	32	指令总线为 32 位
iretire_width_p	1	iretire 总线宽度
ilastsize_width_p	0	ilastsize 总线宽度
itype_width_p	3	itype 总线宽度
nocontext_p	1	在 te_inst 数据包中去掉上下文信息
notime_p	1	在 te_inst 数据包中去掉时间信息
privilege_width_p	1	仅支持机器和用户模式
retires_p	1	每个块退役指令的最大数量
return_stack_size_p	0	不支持隐式返回模式

参数名称	值	描述
sijump_p	0	不支持根据顺序推测跳转
taken_branches_p	1	每个时钟周期仅一条指令退役
impdef_width_p	0	未实现

上述参数的具体描述，详见 Efficient Trace for RISC-V Version 2.0 > 章节 Parameters and Discovery。

3.4 架构概览

本章节主要介绍 ESP32-P4 追踪编码器的具体实现。

如图 3.0-1 所示，追踪编码器包含一个编码器、一个 FIFO、一个寄存器配置模块和一个发送控制模块。

编码器通过指令追踪接口接收 HP CPU 的指令信息，将指令信息压缩成不同的数据包，写入内部 FIFO。

发送控制模块通过 AHB 总线将 FIFO 中的数据写入内部 SRAM。

FIFO 的深度为 128，宽度为 8 位。存储器写入带宽不足时，FIFO 可能会溢出，发生丢包。如果发生丢包，编码器会发送一个包通知丢包情况，在 FIFO 变空前一直停止工作，如果中断已使能还会触发中断。

3.5 功能描述

3.5.1 同步

为保障追踪的有效性，追踪过程中必须定期同步。同步可以通过发送完整的指令地址实现。同步计数器的值达到 TRACE_RESYNC_PROLONGED_REG 寄存器的 TRACE_RESYNC_PROLONGED 字段的值时，编码器会发送一个同步数据包（格式 3 子格式 0，见章节 3.6.3.1）。

追踪编码器支持两种同步模式，可通过 TRACE_RESYNC_MODE 配置：

- 0：关闭同步计数器
- 1：无效值，没有作用
- 2：同步计数器按数据包计数
- 3：同步计数器按周期计数

可通过增加 TRACE_RESYNC_PROLONGED 的值，降低发送同步数据包的频率，从而降低同步数据包对带宽的占用。

3.5.2 地址模式

ESP32-P4 支持两种地址模式：增量地址模式和完整地址模式。

- 增量地址模式（默认）：在增量地址模式下，地址会编码为当前指令的实际地址与上一个包含地址的数据包报告的指令实际地址的差值。这种差值编码所需的位比完整地址少，可以令追踪压缩更高效。
- 完整地址模式：在完整地址模式下，追踪的所有地址会编码为绝对地址，而不是差值。这种编码方式相对低效，但是可以辅助软件解码器开发者进行调试。该模式可以通过置位 TRACE_FULL_ADDRESS 使能。

3.5.3 可选边带信号

Efficient Trace for RISC-V Version 2.0 > 章节 Optional Sideband Signals 介绍了用于控制编码器的可选边带信号。ESP32-P4 实现了下列信号：

表 3.5-1. 可选边带信号

信号	功能
trigger[2:0]	• 位 0 的脉冲可以令编码器开始追踪，并持续追踪直到另有通知。追踪时还需要满足其他过滤条件。 • 位 1 的脉冲可以令编码器停止追踪，直到另有通知。 • 位 2 的脉冲可以令编码器报告特定地址。 该功能可通过置位 TRACE_DM_TRIGGER_ENA 使能。触发器地址匹配时，建立 trigger 信号。例如，若 mcontrol 寄存器的 action 字段为 2，则触发器地址匹配时 trigger 信号的位 0 变为高电平；若 action 字段为 3，则触发器地址匹配时 trigger 信号的位 1 变为高电平。更多信息，详见表 3.8-1。
halted	暂停 hart。halted 信号建立时，编码器先输出一个数据包报告 hart 暂停前退役的最后一条指令的地址，然后再输出一个支持数据包指示追踪已暂停。halted 信号失效时，编码器重新开始追踪，从同步数据包开始。该功能可通过置位 TRACE_HALT_ENA 使能。
reset	复位 hart。具体行为同上文 halted 的描述。
stall	请求 hart 暂停。如果某些应用需要无损追踪，便可以在追踪编码器不能输出追踪数据包（内部 FIFO 将满）时使用该信号暂停 hart。该功能通过置位 TRACE_STALL_ENA 使能。

3.5.4 过滤

过滤机制控制编码器追踪的标准。比如，过滤机制可以让编码器在出现下列情况时追踪：

- 指令地址在特定范围内时追踪
- 从一个指令地址开始追踪，到出现第二个指令地址时结束追踪
- 针对一个或多个特定特权等级追踪
- 出现异常及/或特定异常原因触发中断处理程序时追踪

每个过滤器单元包含一个过滤器和一个比较器单元。

每个比较器单元实际包含一对比较器（主比较器和次比较器，简称 P、S），可根据要求设置与比较器单元匹配的范围，并支持：

- 从 iaddress 和 tval 中选择输入
- 每个比较器可以独立选择运算方式（<、>、=、!= 等）
- 次匹配值可以是主比较器的屏蔽值
- 两个比较器有多种一起使用的方式：P、$P\&\&S$、$!(P\&\&S)$、锁存（P 匹配时置 1，S 匹配时清零）
- 每个比较器可以报告特定指令地址

每个过滤器可以指定要过滤的 hart 指令，并具有以下功能：

- 选择比较器单元进行匹配
- 可以选择多个特权等级和异常原因作为输入进行匹配
- 选择匹配中断

3.5.5 锚定

由于数据包的长度不定，为在写入存储器时分清数据包之间的边界，ESP32-P4 编码器在数据包之间插入 0：

- 数据包的长度最大为 13 字节，因此正常数据包中不能出现连续 14 个值为 0 的字节。连续 14 个及以上值为 0 的字节后，第一个不为 0 的字节肯定是数据包的第一个字节。
- 每发送 128 个数据包，编码器会向存储器写入 14 个值为 0 的字节作为锚定符。

3.5.6 写存储器模式

写追踪存储器时，追踪数据包的大小可能会超过追踪存储器的容量。此时可配置写存储器的模式，选择是否循环写入数据包：

- 循环模式：追踪数据包的大小超过追踪存储器的容量（即 TRACE_MEM_CURRENT_ADDR_REG 的值达到 TRACE_MEM_END_ADDR_REG 的值）时，存储器循环写入数据，回到存储器的起始地址 TRACE_MEM_START_ADDR_REG，此前写入的数据会被新数据覆盖。
- 非循环模式：追踪数据包的大小超过追踪存储器的容量时，存储器不循环，停在 TRACE_MEM_END_ADDR_REG，此前写入的数据保留。

3.5.7 自动重启

因 FIFO 溢出而丢包时，编码器将停止工作，需要软件手动恢复或硬件自动重启。如果 TRACE_TRIGGER_REG 寄存器的 TRACE_RESTART_ENA 位为 1，FIFO 清空后编码器会自动重启。

如果使能自动重启功能，编码器无论出现什么情况都会自动重启。因此，要关闭编码器，必须先清除 TRACE_TRIGGER_REG 寄存器的 TRACE_RESTART_ENA 位从而关闭自动重启。

3.6 编码器输出数据包

本节主要介绍 ESP32-P4 追踪编码器输出的数据包格式。

追踪数据包包括头部、索引和有效载荷，依次按照头部、索引和有效载荷的顺序以比特流形式传输。头部、索引和有效载荷中的字段按照下文表格的罗列顺序依次传输，多位字段先传输最低有效位。

3.6.1 头部

头部为 1 个字节，格式如表 3.6-1 所示。

表 3.6-1. 头部格式

字段	位	描述	值
length	5	数据包的总长度	4~13

字段	位	描述	值
flow	2	保留	0
timestamp	1	保留	0

3.6.2 索引

索引为 2 个字节，格式如表 3.6-2 所示。

表 3.6-2. 索引格式

字段	位	描述	值
index	16	每个数据包的索引	0~65535

3.6.3 有效载荷

有效载荷的宽度为 1 至 10 字节。

3.6.3.1 格式 3

格式 3 的数据包用于同步和发送支持信息。规范中定义了四种子格式，ESP32-P4 仅支持 3 种。

格式 3 子格式 0 - 同步

该格式的数据包包含解码器完全识别指令所需的全部信息。开始追踪第一条指令时（除非指令恰好是异常处理程序中的第一条指令），或同步定时器超时触发同步时，编码器会发送该格式的数据包。该格式数据包有效载荷的长度为 5 个字节。

表 3.6-3. 格式 3 子格式 0

字段名	位	描述
format	2	11 (sync)：同步
subformat	2	00 (start)：开始追踪，或开始同步
branch	1	0：地址指向分支指令，且分支跳转 1：指令非分支指令，或分支没有跳转
privilege	1	所追踪指令的特权等级
address	31	指令的完整地址。字段值需左移 1 位，以复原字节地址
sign_extend	3	保留

格式 3 子格式 1 - 异常

该格式的数据包同样包含解码器完全识别指令所需的全部信息。发生异常或触发中断时，编码器会发送该格式的数据包。该格式的数据包中包含了异常原因、异常指令的“陷阱值”，陷阱处理程序的地址和异常指令的地址。该格式数据包有效载荷的长度不定：如果 interrupt 字段为 1，则 tvalepc 字段省略，长度为 6 个字节；否则长度为 10 个字节。

表 3.6-4. 格式 3 子格式 1

字段名	位	描述
format	2	11 (sync)：同步
subformat	2	01 (exception)：异常原因和陷阱处理程序的地址
branch	1	0：地址指向分支指令，且分支跳转 1：指令非分支指令，或分支没有跳转
privilege	1	所追踪指令的特权等级
ecause	6	异常原因
interrupt	1	中断
theaddr	1	该字段为 1 时，address 字段指向陷阱处理程序的地址。该字段为 0 时，如果异常情况是无法推测出程序计数器的目标地址偏移量，则 address 字段指向异常程序计数器的值，如果是其他异常中断，则 address 字段不定义。
address	31	指令的完整地址。地址必须向左偏移 1 位，以复原字节地址
tvalepc	32	utval、stval 或 mtval 控制/状态寄存器的值。如果 interrupt 字段为 1，该字段省略。
sign_extend	3	保留

格式 3 子格式 3 - 支持

该格式的数据包包含解码器需要的辅助信息，在追踪结束时、过滤器匹配失败或丢包时发出。该格式数据包有效载荷的长度为 2 个字节。

表 3.6-5. 格式 3 子格式 3

字段名	位	描述
format	2	11 (sync)：同步
subformat	2	11 (support)：解码器需要的辅助信息
enable	1	表示编码器已开启
encoder_mode	1	永远为 0。仅分支追踪支持
qual_status	2	表示指令的筛选状态： • 00 (no_change)：指令筛选状态没有变化 • 01 (ended_rep)：指令筛选已结束，上一个指令为最后一个通过筛选的指令 • 10 (trace lost)：丢失了一个或多个包 • 11 (ended_upd)：指令筛选已结束，即便上一个 te_inst 不是最后一个通过筛选的指令，也会因指令地址不连续、无法推测而发送

字段名	位	描述
ioptions	6	表示以下可选模式： • 根据顺序推测跳转（位 5）：该位为 1 时，不会报告根据顺序推测出的跳转目标地址。该功能未实现，因此此位始终为 0 • 分支预测（位 4）：该位为 1 时，使能分支预测。该功能未实现，因此此位始终为 0 • 缓存跳转目标（位 3）：该位为 1 时，使能跳转目标缓存。该功能未实现，因此此位始终为 0 • 完整地址（位 2）： – 0：增量地址模式 – 1：完整地址模式 • 隐式异常（位 1）：如果可以通过 ecause 字段（异常原因）确定陷阱向量，则格式 3 子格式 1 数据包不包含地址。该功能未实现，因此此位始终为 0 • 隐式返回（位 0）：该位为 1 时，不报告函数的返回地址。该功能未实现，因此此位始终为 0
sign_extend	2	保留

3.6.3.2 格式 2

该格式的数据包仅包含指令地址，在必须报告指令地址时使用（比如指令目标地址为无法推测的不连续地址，或指令为发生异常之前的最后一个指令），包中没有分支信息。

增量地址模式使能时，该格式数据包有效载荷的长度不定，在 5 到 8 个字节之间。地址字段宽度可为 8、16、24 或 32 位，取决于数据包长度。例如，如果数据包头部为 n 字节，则地址字段宽度为 $n-4$ 字节。地址字段需要通过有符号扩展到 32 位。

表 3.6-6. 格式 2

字段名	位	描述
format	2	10 (addr-only)：无分支信息
notify	1	若该位的值与地址字段的最高有效位不同，则表示 trigger[2] 或过滤器匹配触发了通知请求，数据包报告指令的目标地址不是无法推测的不连续地址
updiscon	1	若该位的值与 notify 的值不同，则表示指令的目标地址为无法推测的不连续地址，且该地址为异常、特权变更或同步之前的最后一条指令
zero_extend	4	保留
address	不定	指令的增量地址，位宽为 8、16、24 或 32 位

3.6.3.3 格式 1

该格式的数据包包含分支信息，在必须报告分支信息（比如分支映射已满时）或在上一个包后出现至少一个分支、必须报告指令地址时使用。

增量地址模式使能时，该格式数据包有效载荷的长度不定，在 5 到 12 个字节之间。地址字段宽度可为 8、16、24 或 32 位，取决于数据包长度。例如：

- 数据包头部为 n 字节
- format 字段的值为 1
- branch 字段的值为：
  - 0：数据包为不包含地址的格式 1 数据包
  - 1：branch_map 宽度为 1 位，zero_ext 宽度为 6 位，address 宽度为 $n-5$ 位
  - 2 ~ 3：branch_map 宽度为 3 位，zero_ext 宽度为 4 位，address 宽度为 $n-5$ 位
  - 4 ~ 7：branch_map 宽度为 7 位，zero_ext 宽度为 0 位，address 宽度为 $n-5$ 位
  - 8 ~ 15：branch_map 宽度为 15 位，zero_ext 宽度为 0 位，address 宽度为 $n-6$ 位
  - 16 ~ 31：branch_map 宽度为 31 位，zero_ext 宽度为 0 位，address 宽度为 $n-8$ 位

格式 1，有地址时 branch_map 字段

有效载荷长度不定。

表 3.6-7. 格式 1，有地址

字段名	位	描述
format	2	01：包含分支信息
branches	5	该字段的值决定了 branch_map 的位宽（branch_map 的有效位数量），具体如下： • 0：（该格式中此值无效） • 1：1 位 • 2 ~ 3：3 位 • 4 ~ 7：7 位 • 8 ~ 15：15 位 • 16 ~ 31：31 位 比如，若 branches = 12，则 branch_map 为 15 位，低 12 位有效。
branch_map	不定	表示分支是否跳转的位数组。位 0 对应最早执行的分支指令。每个位的含义如下： • 0：分支已跳转 • 1：分支未跳转 该字段的位宽不定，取决于 branches 字段的值
notify	1	若该位的值与地址字段的最高有效位不同，则表示 trigger[2] 或过滤器匹配触发了通知请求，数据包报告指令的目标地址不是无法推测的不连续地址
updiscon	1	若该位的值与 notify 的值不同，则表示指令的目标地址为无法推测的不连续地址，且该地址为异常、特权变更或同步之前的最后一条指令
zero_ext	不定	该字段的宽度取决于 branches 字段，具体如下： • 1：6 位 • 2 ~ 3：4 位 • 4 ~ 31：0 位
address	不定	指令的增量地址，位宽为 8、16、24 或 32 位

格式 1，无地址时 branch_map 字段

该格式数据包有效载荷的长度为 5 个字节。

表 3.6-8. 格式 1，无地址

字段名	位	描述
format	2	01：包含分支信息
branches	5	该字段的值决定了 branch_map 的位宽，仅 0 有效，对应 branch_map 的位宽为 31 位
branch_map	31	表示分支是否跳转的位数组。位 0 对应最早执行的分支指令。每个位的含义如下： • 0：分支已跳转 • 1：分支未跳转
sign_extend	2	保留

3.7 中断

ESP32-P4 的 CORE*n*_TRACE（即 HP CPU 核 0 或 HP CPU 核 1 的追踪编码器）可以生成 CORE*n*_TRACE_INTR 中断信号并发送给中断矩阵。

CORE*n*_TRACE_INTR 中断信号由 CORE*n*_TRACE 的内部中断源生成。表 3.7-1 列出了 CORE*n*_TRACE 的内部中断源、中断触发条件、以及生成的中断信号。

表 3.7-1. CORE*n*_TRACE's 的内部中断源

内部中断源	触发条件	中断信号
TRACE_MEM_FULL_INTR	数据包大小超过追踪存储器容量（即 TRACE_MEM_CURRENT_ADDR_REG 的值达到 TRACE_MEM_END_ADDR_REG 的值）时触发。如有必要，可使能中断通知 HP CPU 处理，比如可再申请新的存储器空间	CORE*n*_TRACE_INTR
TRACE_FIFO_OVERFLOW_INTR	内部 FIFO 溢出时触发，表示一个或多个数据包丢失	CORE*n*_TRACE_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 3.9 寄存器列表。

3.8 编程流程

3.8.1 配置编码器选项

编码器具有一些可选的配置选项。

- 完整地址模式
  - 编码器默认使用增量地址模式。可以置位 TRACE_FULL_ADDRESS 以启用完整地址模式用来调试。
- 丢包后重启
  - 因 FIFO 溢出而丢失追踪数据包时，编码器将自动停止。如果 TRACE_RESTART_ENA 为 1，编码器会在 FIFO 为空时重启。
- FIFO 将满时暂停 CPU
  - 如果 TRACE_STALL_ENA 为 1，FIFO 将满时会向 CPU 发出暂停请求，CPU 会一直暂停，直到请求失效。
- CPU 暂停时停止追踪

  如果 TRACE_HALT_ENA 为 1，编码器会在 CPU 暂停时先输出一个数据包，报告暂停前最后一条指令的地址，然后再输出一个支持数据包指示追踪已停止。halted 信号失效时，编码器重新开始追踪，从同步数据包开始。
- CPU 复位时停止追踪

  如果 TRACE_RESET_ENA 为 1，编码器会单独复位。编码器会在 CPU 复位时先输出一个数据包，报告复位前最后一条指令的地址，然后再输出一个支持数据包指示追踪已停止。CPU 退出复位状态时，编码器重新开始追踪，从同步数据包开始。
- 使用调试模块的触发输出

  RISC-V hart 的调试模块有一个触发单元，其匹配控制寄存器（mcontrol）有一个 4 位的 action 字段，该字段值为 2 ~ 4 时用于追踪。ESP32-P4 已实现此功能。action 字段的值详见表 3.8-1。

表 3.8-1. 调试模块的触发支持（mcontrol 寄存器的 action 字段）

值	描述
2	开始追踪
3	结束追踪
4	通知编码器报告地址

有关调试模块触发单元的详细信息，请参考章节 2 高性能处理器。

3.8.2 配置过滤器

章节 3.5.4 中描述的过滤单元可以按以下方式配置：

1. 选择以下任一匹配模式：
   - 置位 TRACE_MATCH_PRIVILEGE 启用特权匹配模式：
     - 通过 TRACE_MATCH_CHOICE_PRIVILEGE 配置特权级别
   - 置位 TRACE_MATCH_ECAUSE 启用 ecause 匹配模式：
     - 通过 TRACE_MATCH_CHOICE_ECAUSE 配置 ecause 值
   - 置位 TRACE_MATCH_INTERRUPT 启用中断匹配模式：
     - 通过 TRACE_MATCH_VALUE_INTERRUPT 配置中断级别

- 置位 TRACE_MATCH_COMP 启用比较器匹配模式：
  - 配置主比较器：
    * 通过 TRACE_P_INPUT 选择输入
    * 通过 TRACE_P_FUNCTION 选择比较器运算符
    * 通过 TRACE_FILTER_P_COMPARATOR_MATCH_REG 配置匹配值
  - 必要时配置次比较器：
    * 通过 TRACE_S_INPUT 选择输入
    * 通过 TRACE_S_FUNCTION 选择比较器运算符
    * 通过 TRACE_FILTER_S_COMPARATOR_MATCH_REG 配置匹配值
  - 通过 TRACE_MATCH_MODE 选择用于匹配的比较器：
    * 0：仅主比较器用于匹配，次比较器无效
    * 1：主比较器和次比较器都用于匹配 ($P\&\&S$)
    * 2：主比较器或次比较器都不用于匹配 $!(P\&\&S)$
    * 3：主比较器匹配时开始过滤，次比较器匹配时结束过滤

  如果置位 TRACE_P_NOTIFY 或 TRACE_S_NOTIFY，比较器会以通知的形式报告匹配地址，指令不会过滤。例如：
  - 如果 TRACE_MATCH_MODE 为 0，并且 TRACE_P_NOTIFY 为 1，则所有指令都会追踪，指令与主比较器匹配时会发出通知。
  - 如果 TRACE_MATCH_MODE 为 0，并且 TRACE_S_NOTIFY 为 1，则指令将通过主比较器过滤，指令与次比较器匹配时会发出通知。

2. 置位 TRACE_FILTER_EN 以启用过滤单元。

3.8.3 使能编码器

- 通过 TRACE_MEM_START_ADDR_REG 和 TRACE_MEM_END_ADDR_REG 配置追踪存储器的地址空间。编码器可以访问 L2 MEM 和 PSRAM：
  - L2MEM: 0x4FF00000 ~ 0x4FFBFFFF
  - PSRAM: 0x48000000 ~ 0x4BFFFFFF
- 置位 TRACE_MEM_CURRENT_ADDR_UPDATE，将 TRACE_MEM_CURRENT_ADDR_REG 的值更新为 TRACE_MEM_START_ADDR_REG 的值
- （可选）通过 TRACE_TRIGGER_REG 寄存器的 TRACE_MEM_LOOP 位配置写存储器的模式
  - 0：非循环模式
  - 1：循环模式（默认）
- 通过 TRACE_RESYNC_PROLONGED_REG 寄存器的 TRACE_RESYNC_MODE 位配置同步模式
  - 0：关闭同步计数器（默认）
  - 1：无效值，没有作用

  - 2：同步计数器按数据包计数
  - 3：同步计数器按周期计数
- （可选）通过 TRACE_RESYNC_PROLONGED_REG 配置同步计数器的阈值，默认为 128
- （可选）使能中断
  - 向 TRACE_INTR_ENA_REG 寄存器的相应位写 1 开启对应中断
  - 向TRACE_INTR_CLR_REG 寄存器的相应位写 1 清除对应中断
  - 读 TRACE_INTR_RAW_REG 寄存器的值，获取中断状态
- （可选）向 TRACE_TRIGGER_REG 寄存器的 TRACE_RESTART_ENA 位写 1，使能自动重启。该功能默认开启
- 向 TRACE_TRIGGER_REG 寄存器的 TRACE_TRIGGER_ON 字段写 1，使能编码器
- 使能追踪编码器，有以下两种方式：
  - 向 TRACE_TRIGGER_REG 寄存器的 TRACE_TRIGGER_ON 位写 1
  - 通过调试模块的 action 字段使能编码器。触发器地址匹配且 action 字段为 2 时，编码器使能。触发器的配置，请参考 RISC-V Debug Specification。

编码器使能后，会一直追踪 HP CPU 的指令接口，向追踪存储器写入数据包。

3.8.4 关闭编码器

停止输出追踪数据包，有以下两种方式：

1. 向 TRACE_TRIGGER_REG 寄存器的 TRACE_TRIGGER_OFF 位写 1 关闭编码器
2. 通过调试模块的 action 字段关闭编码器。触发器地址匹配且 action 字段为 3 时，编码器关闭。触发器的配置，请参考 RISC-V Debug Specification。

由于内部 FIFO 的存在，数据包在追踪结束后不会立即写入存储器。建议查询TRACE_FIFO_STATUS_REG 确认数据是否全部写入存储器。

表 3.8-2. 追踪状态

字段	描述
TRACE_FIFO_EMPTY	该字段如果为 1，则表示 FIFO 为空，所有数据已写入内存
TRACE_WORK_STATUS	编码器的工作状态： • 0：空闲状态，编码器未启动 • 1：编码器正常工作，输出数据包 • 2：编码器未输出数据包，CPU 已暂停或复位 • 3：编码器未输出数据包，发生数据包丢失，等待 FIFO 为空时重新启动

3.8.5 通知

如果一个目标地址很特殊，即使不是无法推测的不连续地址，用户也可能想要报告。

通知和报告特殊地址有以下两种方式：

1. 调试模块的触发器地址匹配，并且 action 为 4
2. 过滤器比较器匹配

收到通知请求时，编码器会发出格式 2 或格式 1 的数据包报告地址，具体取决于 branch_map 字段。如果 branch_map 为 0，则发送格式 2 数据包，否则发送格式 1 数据包。notify 字段表示数据包是否报告的是通知地址。如果地址是无法推测的不连续地址，即使是通知用的数据包，notify 字段也不会变为高电平，否则解码器无法正确重建指令流。

3.8.6 解码数据包

- 找到解码的第一个地址
  - 读 TRACE_INTR_RAW_REG 寄存器的 TRACE_MEM_FULL_INTR_RAW 位，查看追踪存储器是否已满
    * 如果值为 0，从 TRACE_MEM_START_ADDR_REG 寄存器读取追踪数据包
    * 如果值为 1，且开启了存储器循环模式，则之前的数据包已被覆盖。此时，读取 TRACE_MEM_CURRENT_ADDR_REG 的值，获取最后一个写入的地址，该地址即为解码器处理的第一个地址
- 用解码器解码数据包
  - 解码器从第一个地址开始读取所有数据包，结合二进制文件重建数据流
  - 如章节 3.6 所述，编码器向存储器分区边界写 14 个值为 0 的字节。因此出现 14 个值为 0 的字节时，则表示下一个非 0 字节为新数据包的头部

3.8.7 AHB 配置

ESP32-P4 编码器支持 AHB 总线，可选配置项可以控制编码器的传输带宽。

如果有多个 DMA 主机，可以通过 TRACE_AHB_CONFIG_REG 配置 AHB 突发传输，增加编码器的带宽。

- TRACE_HBURST 支持以下配置：
  - 0x0：SINGLE
  - 0x1：INCR（长度未定义）
  - 0x2：INCR4
  - 0x4：INCR8
  - 其他值：保留

  突发传输意味着仲裁成功后，可以连续传输多个字。SINGLE 传输为 1 拍，INCR 长度未定义，INCR4 为 4 拍，INCR8 为 8 拍。突发长度越长，带宽越大。
- 传输配置为 INCR 时，由于传输长度不确定，为了避免编码器长时间占用总线，传输长度达到 TRACE_MAX_INCR 后 INCR 传输结束，重新开始总线仲裁。

该配置用于调整带宽。带宽足够时，不需要配置突发传输，可使用默认配置。

3.8.8 软件保留

TRACE_WORK_STATUS 表示编码器的状态。编码器不处于空闲状态时，必须备份配置，并在芯片进入睡眠状态时还原配置。

3.9 寄存器列表

本小节的所有地址均为相对于 RISC-V 追踪编码器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

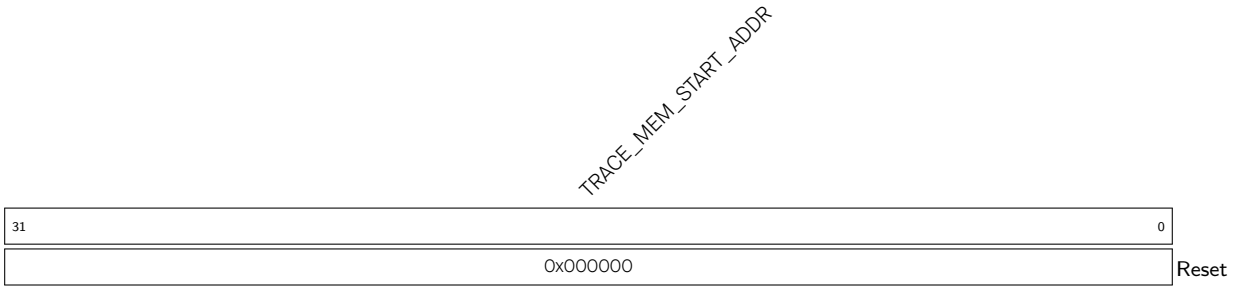
名称	描述	地址	访问
存储器配置寄存器			
TRACE_MEM_START_ADDR_REG	存储器起始地址	0x0000	R/W
TRACE_MEM_END_ADDR_REG	存储器结束地址	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	存储器当前地址	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	存储器地址更新	0x000C	WT
FIFO 状态寄存器			
TRACE_FIFO_STATUS_REG	FIFO 状态寄存器	0x0010	RO
中断寄存器			
TRACE_INTR_ENA_REG	中断使能寄存器	0x0014	R/W
TRACE_INTR_RAW_REG	中断原始状态寄存器	0x0018	RO
TRACE_INTR_CLR_REG	中断清除寄存器	0x001C	WT
追踪配置寄存器			
TRACE_TRIGGER_REG	追踪触发寄存器	0x0020	varies
TRACE_CONFIG_REG	追踪配置寄存器	0x0024	R/W
TRACE_FILTER_CONTROL_REG	过滤器控制寄存器	0x0028	R/W
TRACE_FILTER_MATCH_CONTROL_REG	过滤器匹配控制寄存器	0x002C	R/W
TRACE_FILTER_COMPARATOR_CONTROL_REG	过滤器比较器匹配控制寄存器	0x0030	R/W
TRACE_FILTER_P_COMPARATOR_MATCH_REG	主比较器匹配值寄存器	0x0034	R/W
TRACE_FILTER_S_COMPARATOR_MATCH_REG	次比较器匹配值寄存器	0x0038	R/W
TRACE_RESYNC_PROLONGED_REG	再同步配置寄存器	0x003C	R/W
TRACE_AHB_CONFIG_REG	AHB 配置寄存器	0x0040	R/W
时钟门控控制和配置寄存器			
TRACE_CLOCK_GATE_REG	时钟门控控制寄存器	0x0044	R/W
版本寄存器			
TRACE_DATE_REG	版本控制寄存器	0x03FC	varies

3.10 寄存器

本小节的所有地址均为相对于 RISC-V 追踪编码器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

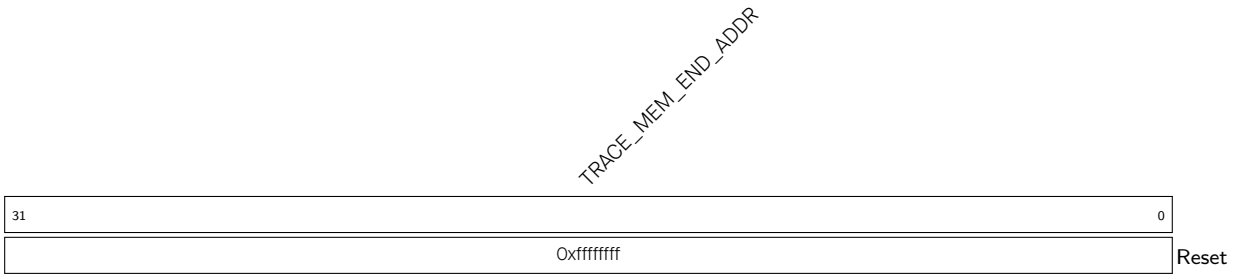
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 3.1. TRACE_MEM_START_ADDR_REG (0x0000)



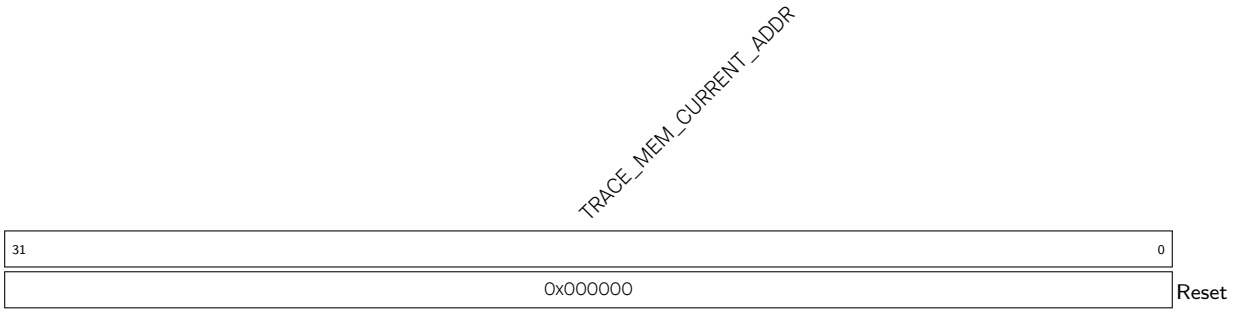
TRACE_MEM_START_ADDR 配置追踪存储器的起始地址。(R/W)

Register 3.2. TRACE_MEM_END_ADDR_REG (0x0004)



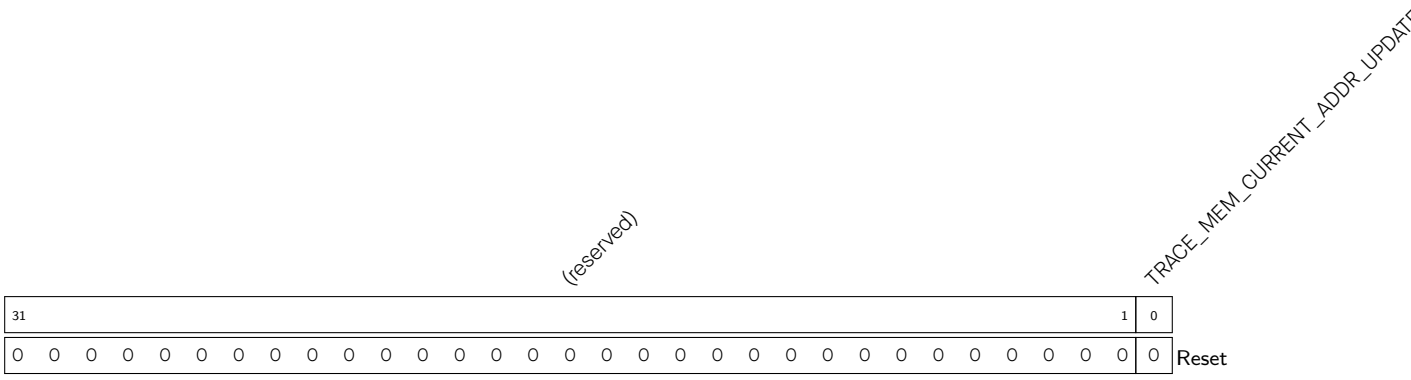
TRACE_MEM_END_ADDR 配置追踪存储器的结束地址。(R/W)

Register 3.3. TRACE_MEM_CURRENT_ADDR_REG (0x0008)



TRACE_MEM_CURRENT_ADDR 表示当前要写入的存储器地址。(RO)

Register 3.4. TRACE_MEM_ADDR_UPDATE_REG (0x000C)



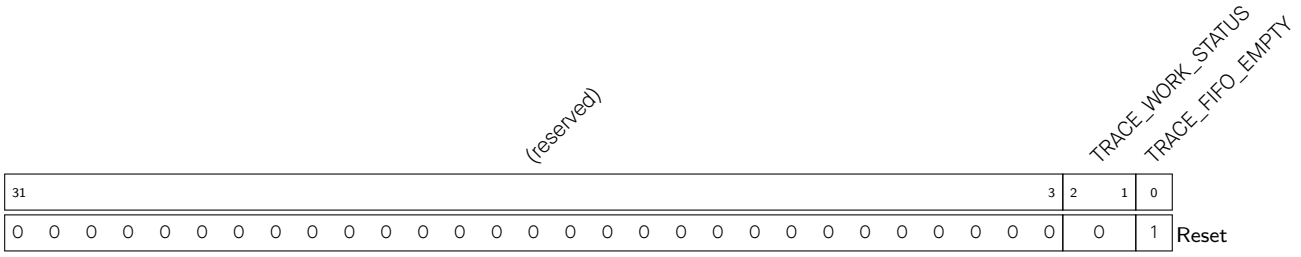
TRACE_MEM_CURRENT_ADDR_UPDATE 配置是否将 TRACE_MEM_CURRENT_ADDR 的值更新为 TRACE_MEM_START_ADDR 的值。

0：不更新

1：更新

(WT)

Register 3.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY 表示 FIFO 的状态。

0：不为空

1：为空

(RO)

TRACE_WORK_STATUS 表示编码器的状态。

0：空闲状态

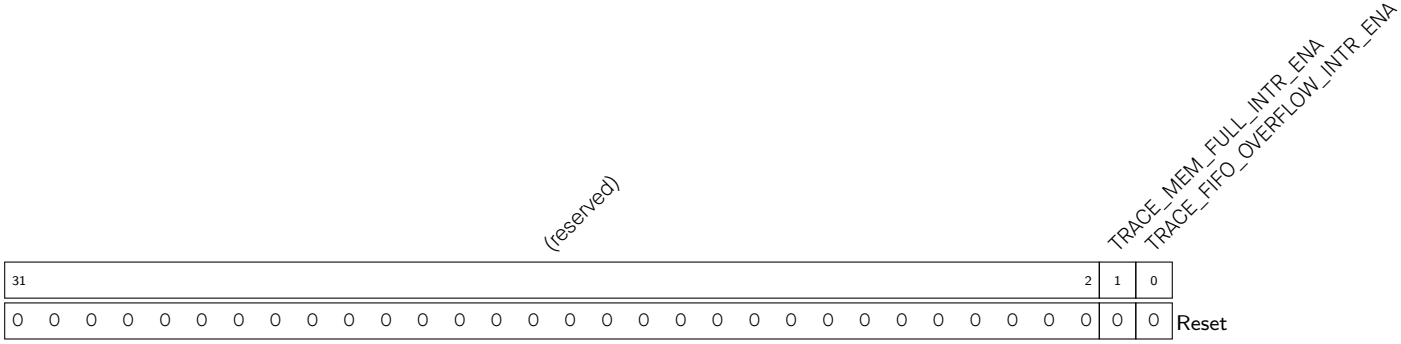
1：工作状态

2：因 hart 暂停或复位处于待机状态

3：丢失状态

(RO)

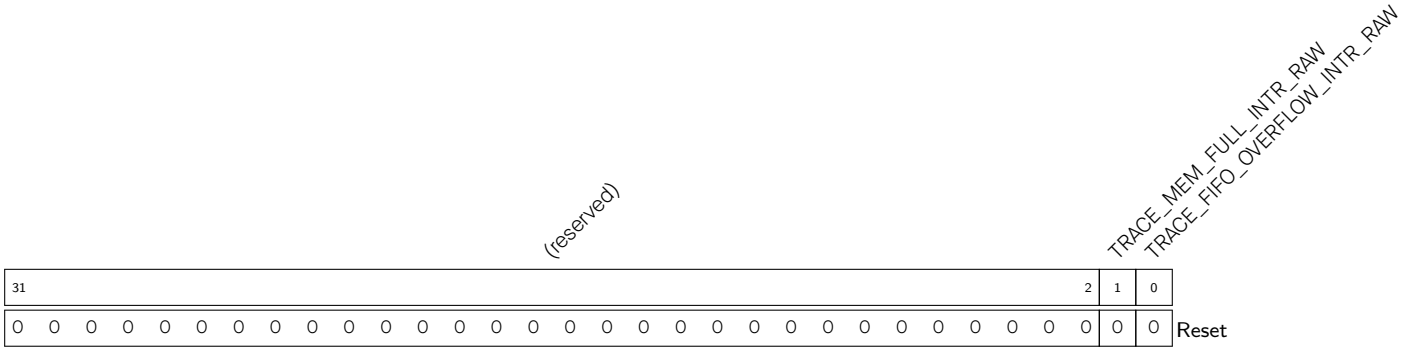
Register 3.6. TRACE_INTR_ENA_REG (0x0014)



TRACE_FIFO_OVERFLOW_INTR_ENA 写 1 使能 TRACE_FIFO_OVERFLOW_INTR。(R/W)

TRACE_MEM_FULL_INTR_ENA 写 1 使能 TRACE_MEM_FULL_INTR。(R/W)

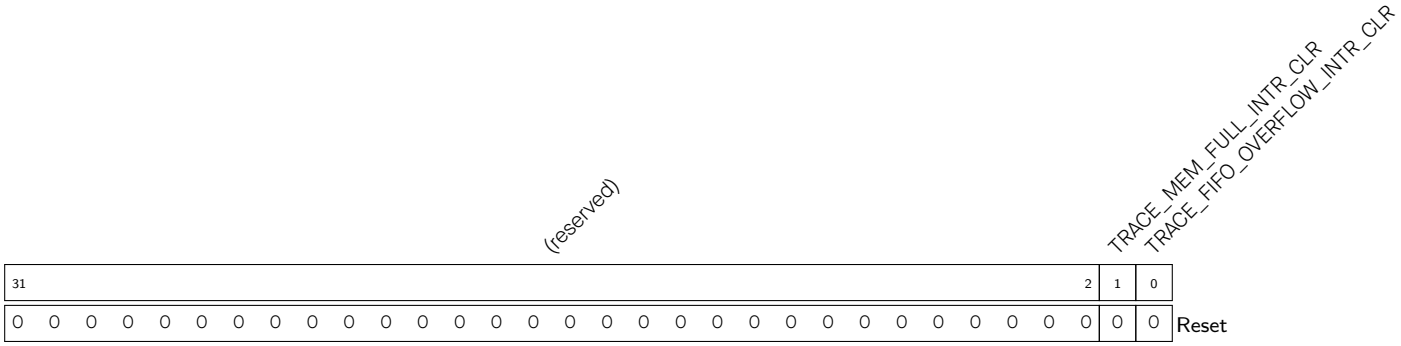
Register 3.7. TRACE_INTR_RAW_REG (0x0018)



TRACE_FIFO_OVERFLOW_INTR_RAW TRACE_FIFO_OVERFLOW_INTR 的原始中断状态。(RO)

TRACE_MEM_FULL_INTR_RAW TRACE_MEM_FULL_INTR 的原始中断状态。(RO)

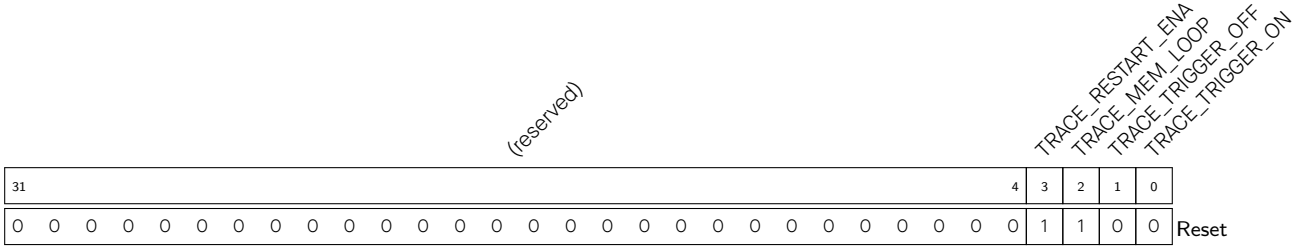
Register 3.8. TRACE_INTR_CLR_REG (0x001C)



TRACE_FIFO_OVERFLOW_INTR_CLR 写 1 清除 TRACE_FIFO_OVERFLOW_INTR。(WT)

TRACE_MEM_FULL_INTR_CLR 写 1 清除 TRACE_MEM_FULL_INTR。(WT)

Register 3.9. TRACE_TRIGGER_REG (0x0020)



TRACE_TRIGGER_ON 配置开启追踪编码器。

0：无效值，没有作用

1：开启

(WT)

TRACE_TRIGGER_OFF 配置关闭追踪编码器。

0：无效值，没有作用

1：关闭

(WT)

TRACE_MEM_LOOP 配置存储器写入模式。

0：非循环模式

1：循环模式

(R/W)

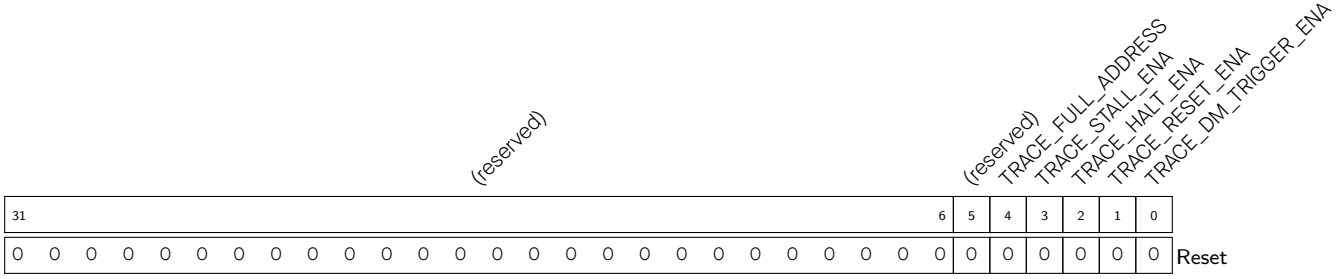
TRACE_RESTART_ENA 配置是否开启编码器自动重启功能。

0：关闭

1：开启

(R/W)

Register 3.10. TRACE_CONFIG_REG (0x0024)



TRACE_DM_TRIGGER_ENA 配置是否使能 trigger 信号。

0：禁用

1：使能

(R/W)

TRACE_RESET_ENA 配置是否使能 reset 信号。

0：禁用

1：使能

(R/W)

TRACE_HALT_ENA 配置是否使能 halted 信号。

0：禁用

1：使能

(R/W)

TRACE_STALL_ENA 配置是否使能 stall 信号。

0：禁用

1：使能

(R/W)

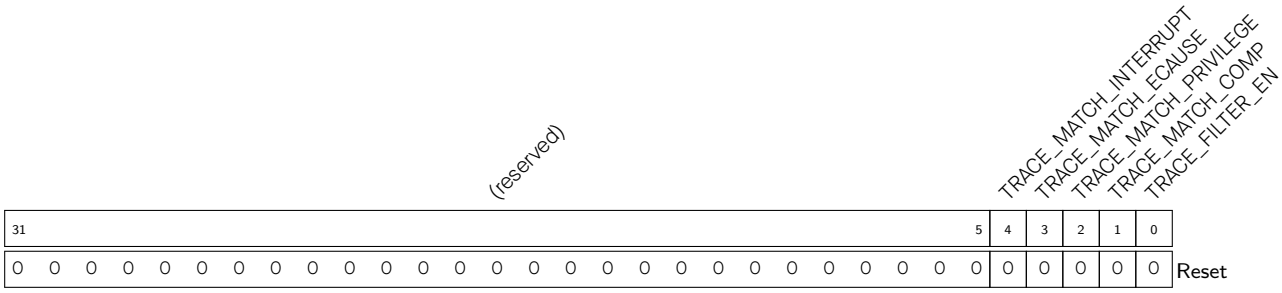
TRACE_FULL_ADDRESS 配置地址模式。

0：增量地址模式

1：完整地址模式

(R/W)

Register 3.11. TRACE_FILTER_CONTROL_REG (0x0028)



TRACE_FILTER_EN 配置是否使能过滤功能。

0：禁用

1：使能

(R/W)

TRACE_MATCH_COMP 配置是否使能比较器匹配模式。

0：禁用

1：使能

(R/W)

TRACE_MATCH_PRIVILEGE 配置是否使能特权匹配模式。

0：禁用

1：使能

(R/W)

TRACE_MATCH_ECAUSE 配置是否使能 ecause 匹配模式。

0：禁用

1：使能 (R/W)

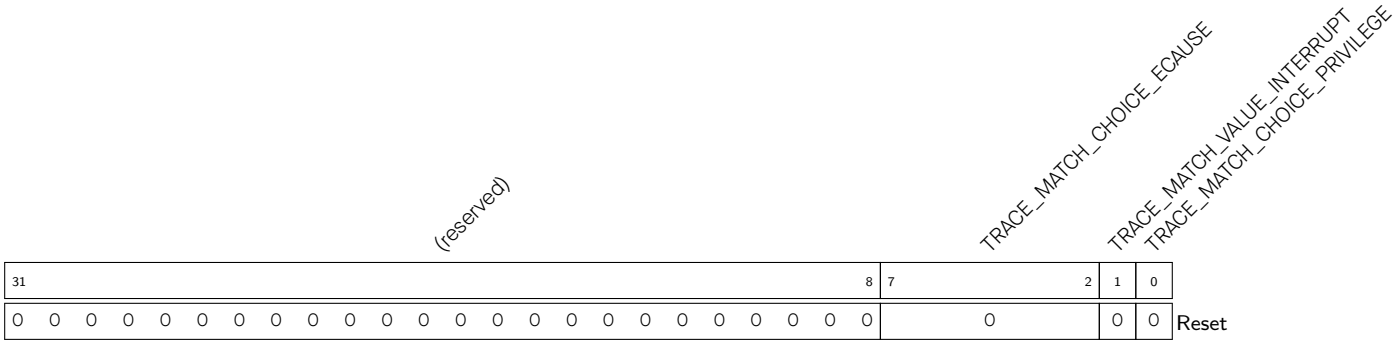
TRACE_MATCH_INTERRUPT 配置是否使能中断匹配模式。

0：禁用

1：使能

(R/W)

Register 3.12. TRACE_FILTER_MATCH_CONTROL_REG (0x002C)



TRACE_MATCH_CHOICE_PRIVILEGE 配置匹配的特权等级。

0：用户模式

1：机器模式

(R/W)

TRACE_MATCH_VALUE_INTERRUPT 配置匹配的中断。仅 TRACE_MATCH_INTERRUPT 为 1 时有效。

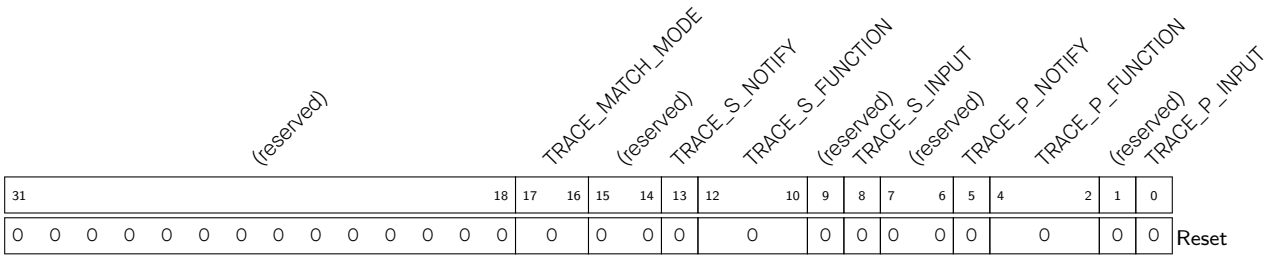
0：itype=1

1：itype=2

(R/W)

TRACE_MATCH_CHOICE_ECAUSE 配置匹配的 ecause 代码。(R/W)

Register 3.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)



TRACE_P_INPUT 配置主比较器匹配的输入。

0：iaddr

1：tval

(R/W)

TRACE_P_FUNCTION 配置主比较器的运算符。

0：等于

1：不等于

2：小于

3：小于或等于

4: 大于

5：大于或等于

其他值：始终匹配

(R/W)

TRACE_P_NOTIFY 配置是否明确报告主比较器匹配的指令地址。

0：不报告

1：报告

(R/W)

TRACE_S_INPUT 配置次比较器匹配的输入。

0：iaddr

1：tval

(R/W)

TRACE_S_FUNCTION 配置次比较器的运算符。

0：等于

1：不等于

2：小于

3：小于或等于

4: 大于

5：大于或等于

6：次匹配值是主比较器的屏蔽值

其他值：始终匹配

(R/W)

TRACE_S_NOTIFY 配置是否明确报告次比较器匹配的指令地址。

0：不报告

1：报告

(R/W)

见下页...

Register 3.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)

接上页...

TRACE_MATCH_MODE 配置比较器的匹配条件。

0：仅主比较器匹配

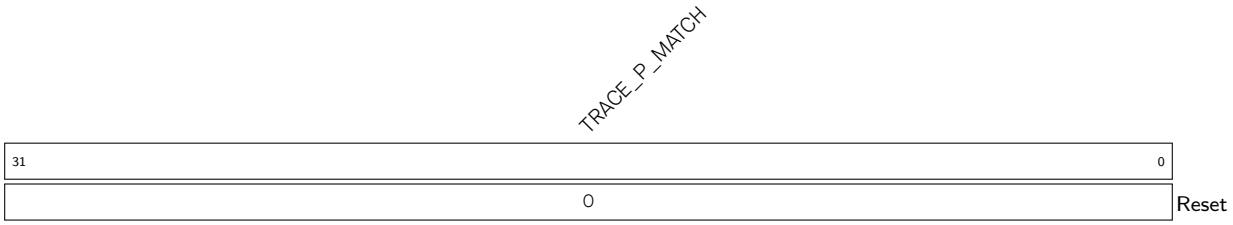
1：主比较器和次比较器都用于匹配 (P&&S)

2：主比较器和次比较器都不用于匹配!(P&&S)

3：主比较器匹配时开始过滤，次比较器匹配时结束过滤

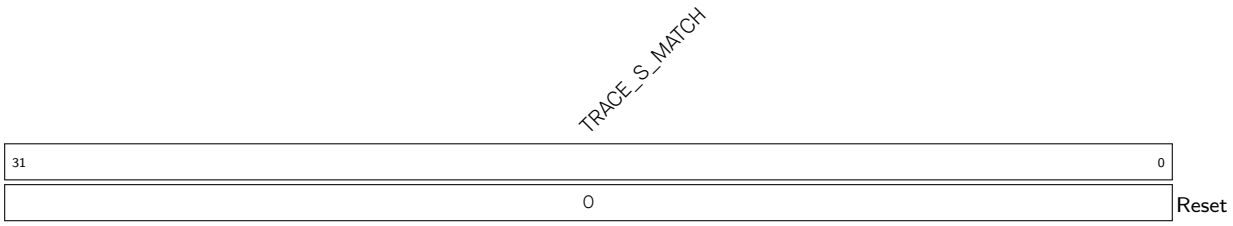
(R/W)

Register 3.14. TRACE_FILTER_P_COMPARATOR_MATCH_REG (0x0034)



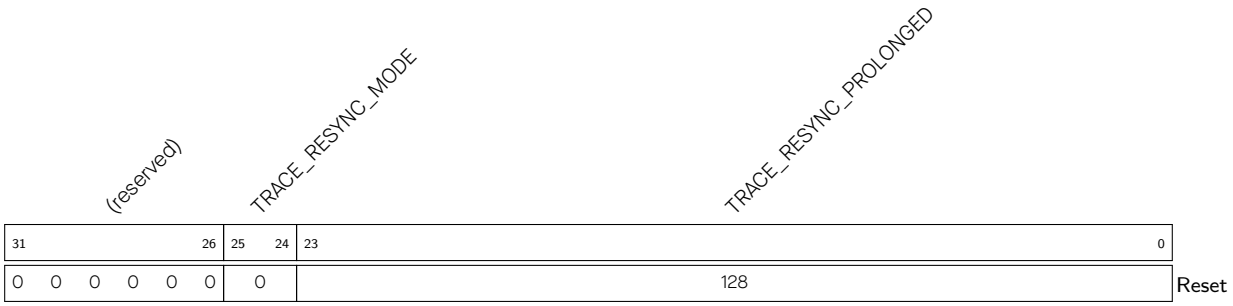
TRACE_P_MATCH 配置主比较器的匹配值。(R/W)

Register 3.15. TRACE_FILTER_S_COMPARATOR_MATCH_REG (0x0038)



TRACE_S_MATCH 配置次比较器的匹配值。(R/W)

Register 3.16. TRACE_RESYNC_PROLONGED_REG (0x003C)

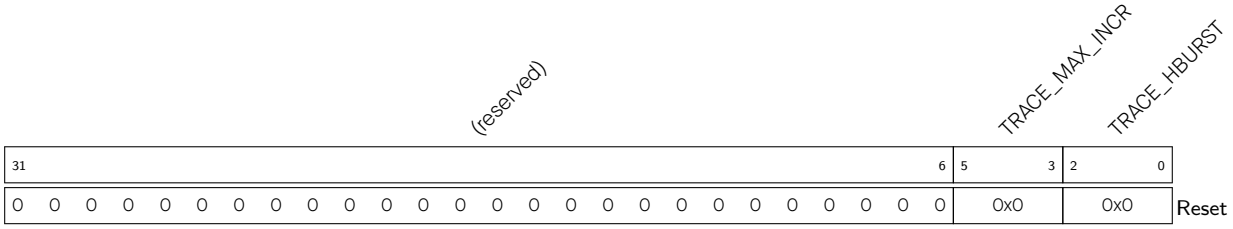


TRACE_RESYNC_PROLONGED 配置同步计数器的阈值。(R/W)

TRACE_RESYNC_MODE 配置同步模式。

0：关闭计数器
1：无效值
2：按数据包计数同步
3：按周期计数同步
(R/W)

Register 3.17. TRACE_AHB_CONFIG_REG (0x0040)

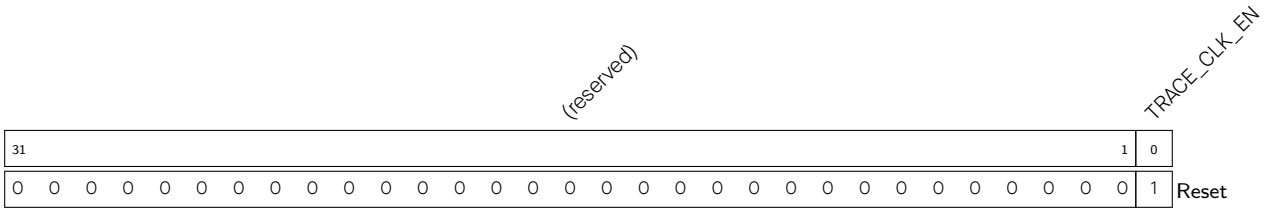


TRACE_HBURST 配置 AHB 突发传输模式。

0：SINGLE
1：INCR（不定义长度）
2：INCR4
4：INCR8
其他值：无效
(R/W)

TRACE_MAX_INCR 配置 INCR 模式下突发传输最大的长度。(R/W)

Register 3.18. TRACE_CLOCK_GATE_REG (0x0044)



TRACE_CLK_EN 配置寄存器时钟门控。

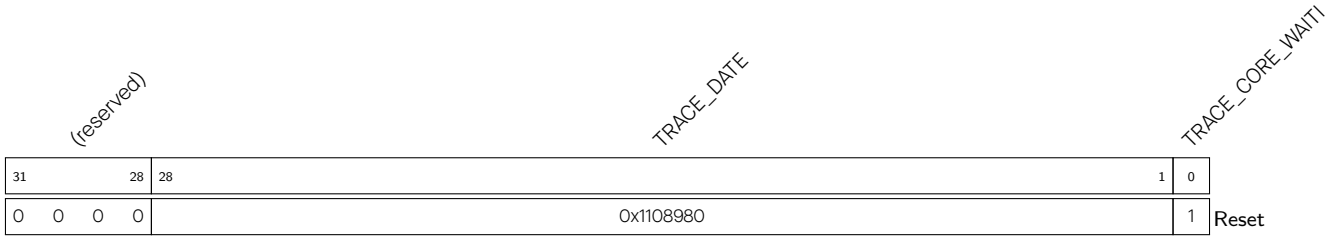
0：仅在应用写寄存器时开启时钟，从而降低功耗

1：一直强制为寄存器开启时钟

该位不影响寄存器访问。

(R/W)

Register 3.19. TRACE_DATE_REG (0x03FC)



TRACE_DATE 版本控制寄存器。(R/W)

TRACE_CORE_WAITI 表示 HP CPU 核是否处于等待中断（WFI）状态。

0: 非 WFI 状态

1: WFI 状态

(RO)

第 4 章

处理器指令扩展 (PIE)

4.1 概述

ESP32-P4 高性能 (HP)32 位 RISC-V 双核处理器支持标准 RV32IMAFCZcZb 扩展。同时提供自定义扩展 Xesploop（硬件循环），可在硬件中执行 `for` 循环以提升性能。详见 2高性能处理器 章节。此外还提供面向 AI 与 DSP 的自定义扩展 Xespv2p2，用于加速相关算法。

本章介绍基于 SIMD（单指令多数据）架构的 Xespv2p2 扩展。该扩展支持 8 位、16 位和 32 位向量运算，可提高数据吞吐量。乘法、移位、累加等算术指令还能在运算的同时完成数据传输，从而进一步提高执行效率。

4.2 主要特性

Xespv2p2 扩展具有以下特性：

- 128 位通用寄存器
- 128 位向量运算，例如乘法、加法、减法、累加、移位和比较
- 将数据传输合并到算术指令中
- 支持非对齐的 128 位向量数据
- 可配置的舍入与饱和模式

4.3 结构概述

4.3.1 内部结构示意图

本节有助于理解可用指令、其能力与限制，并不描述硬件实现细节。

下图给出 Xespv2p2 扩展面向乘累加 (multiply-accumulate, MAC) 指令的内部结构：

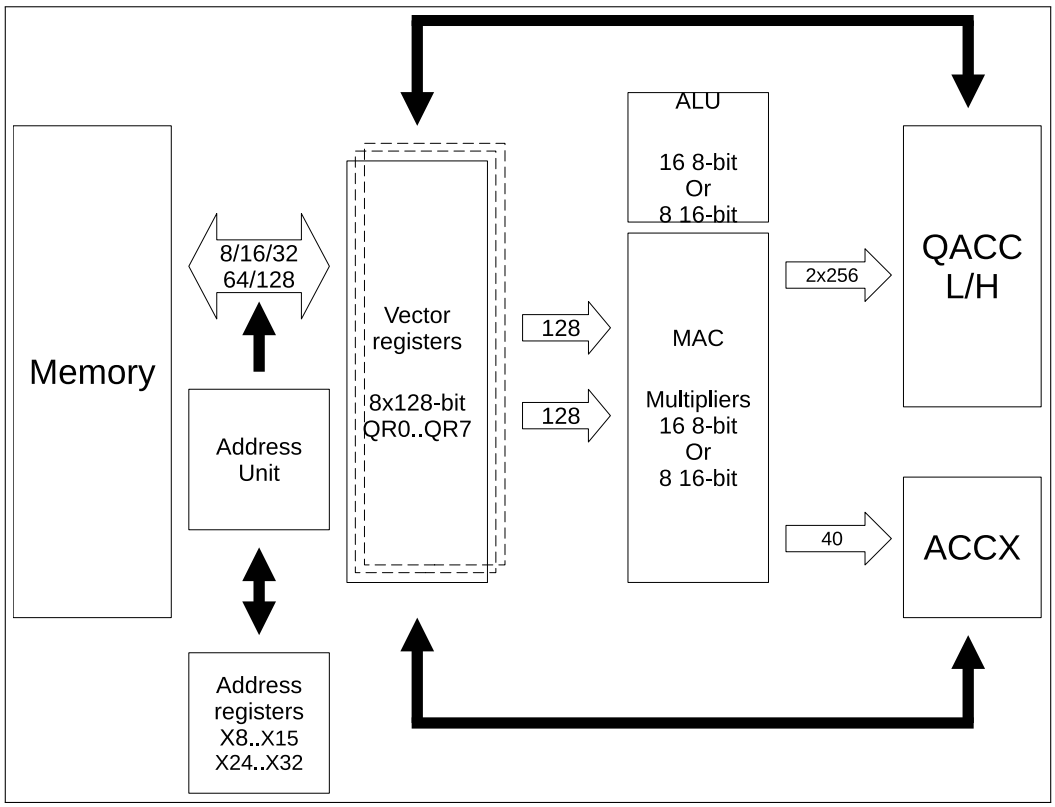


图 4.3-1. Xespv2p2 内部结构 (MAC)

上图展示了数据通路及 Xespv2p2 组件。

Xespv2p2 单元包含：

- 读取 8/16/32/64/128 位对齐数据的地址单元
- 8 个 128 位向量寄存器 QR 组成的寄存器组
- 运算逻辑单元 (ALU)，其中包括
  - 16 个 8 位乘法器
  - 8 个 16 位乘法器
- QACC_H/QACC_L：两个 256 位累加器
- XACC：40 位累加器

4.3.2 向量寄存器组

Xespv2p2 提供 8 个 128 位向量寄存器 (QR)。每个寄存器可视为 16 个 8 位元素、8 个 16 位元素或 4 个 32 位元素的数组。元素位宽由所执行指令决定。

4.3.3 运算逻辑单元 (ALU)

ALU 可按输入数据宽度工作于 8 位、16 位或 32 位：8 位输入对应 8 位 ALU，16 位输入对应 16 位 ALU，32 位输入对应 32 位 ALU。ALU 支持加法、减法等算术运算，以及与、或、移位等位运算。
ALU 操作数来自 QR 寄存器，结果可写回 QR 寄存器。

4.3.4 乘法累加 (MAC) 单元

乘累加 (MAC) 单元执行乘累加运算，包含 8 个 16 位乘法器和 16 个 8 位乘法器。MAC 结果保存在专用累加寄存器 XACC 与 QACC 中。

4.3.4.1 QACC 累加寄存器

QACC 用于保存 16 个 8 位 MAC 结果或 8 个 16 位 MAC 结果，由 256 位 QACC_H 和 256 位 QACC_L 两部分组成。在 8 位 MAC 中，每个 256 位 QACC 寄存器视为 8 个 32 位元素；在 16 位 MAC 中，视为 4 个 64 位元素。

将累加结果写回 QR 寄存器时，可通过右移把 32 位结果压缩为 8 位；对于 16 位 MAC，可通过右移把 64 位结果压缩为 16 位。

数据可从内存加载到 QACC，也可将 QACC 清零。

4.3.4.2 XACC 累加寄存器

XACC 是 40 位累加器，用于保存各乘法器输出之和。
数据可从内存加载到 XACC，也可将 XACC 清零。

4.3.5 地址单元

多数 Xespv2p2 指令可在运算的同时并行访问 128 位 QR 寄存器。地址单元可并行更新地址寄存器，减少显式更新地址的开销。
支持的地址更新包括 AR + 有符号立即数、ARx + ARy 以及 AR + 16。

4.4 扩展指令集组件

4.4.1 寄存器

本节介绍 ESP32-P4 扩展指令集所使用的寄存器。寄存器一览见表 4.4-1。

表 4.4-1. ESP32-P4 扩展指令集寄存器集合

寄存器	数量	比特位宽	访问权限	读写指令	类型
AR	16[1]	32	R/W	-	通用寄存器
QR	8	128	R/W	ESP.VLD.128.IP	自定义矢量寄存器
				ESP.VST.128.IP	
SAR	1	6	R/W	ESP.MOVX.R.SAR	专用寄存器
				ESP.MOVX.W.SAR	
SAR_BYTE	1	4	R/W	ESP.MOVX.R.SAR.BYTES	自定义专用寄存器
				ESP.MOVX.W.SAR.BYTES	
XACC	1	40	R/W	ESP.LD.XACC.IP	自定义专用寄存器
				ESP.ST.S.XACC.IP	
				ESP.ST.U.XACC.IP	
QACC_H	1	256	R/W	ESP.LD.QACC.H.L.128.IP	自定义专用寄存器
				ESP.LD.QACC.H.H.128.IP	

见下页...

表 4.4-1 ... 接上页

寄存器	数量	比特位宽	访问权限	读写指令	类型
				ESP.ST.QACC.H.L.128.IP	
				ESP.ST.QACC.H.H.128.IP	
QACC_L	1	256	R/W	ESP.LD.QACC.L.L.128.IP	自定义专用寄存器
				ESP.LD.QACC.L.H.128.IP	
				ESP.ST.QACC.L.L.128.IP	
				ESP.ST.QACC.L.H.128.IP	
FFT_BIT_WIDTH	1	4	R/W	ESP.MOVX.R.FFT.BIT.WIDTH	自定义专用寄存器
				ESP.MOVX.W.FFT.BIT.WIDTH	
UA_STATE	1	128	R/W	ESP.LD.UA.STATE.IP	自定义专用寄存器
				ESP.ST.UA.STATE.IP	
CFG	1	32	R/W	ESP.MOVX.R.CFG	自定义专用寄存器
				ESP.MOVX.W.CFG	

4.4.1.1 通用寄存器

4.4.1.1.1 AR

尽管 HP 核有 32 个通用寄存器，但仅 16 个可用于 Xespv2p2 扩展指令：x8–x15 和 x24–x31（对应 s0–s1、a0–a5、s8–s11、t3–t6）。

4.4.1.2 向量寄存器

4.4.1.2.1 QR

通用寄存器仅为 32 位宽，而 ESP32-P4 一次可访问 128 位数据，因此通用寄存器只能利用数据总线带宽的约四分之一。为此，ESP32-P4 提供了 8 个 128 位向量寄存器 (QR)。QR 主要用于承载经 128 位总线的访存数据，以及暂存 128 位向量运算结果。

执行指令时，单个 QR 寄存器按指令定义的向量位宽被视为 16 个 8 位、8 个 16 位或 4 个 32 位操作数，从而在一条指令中处理多个元素。

将 QR 用作指令操作数时，必须显式指定寄存器编号。例如：

$$ESP.VADD.S8 \quad q2, q0, q1$$

该指令以 q0、q1 为输入，并将向量加法结果写入 q2。

4.4.1.3 专用寄存器

与通用寄存器不同，专用寄存器由特定指令隐式使用，既不需要也无法在指令中显式指定。例如：

$$ESP.VMUL.S16 \quad q2, q0, q1$$

该向量乘法以 q0、q1 为输入。内部对各 32 位中间乘积右移后，保留低 16 位并组成写入 q2 的 128 位结果。移位量由移位量寄存器 (SAR) 决定，且 SAR 不会出现在操作数列表中。

4.4.1.3.1 SAR

移位量寄存器 (Shift Amount Register, SAR) 以比特为单位保存移位量。ESP32-P4 扩展指令中使用 SAR 的指令分两类：向量移位指令，包括 ESP.VSR.S32、ESP.VSL.S32 和 ESP.VSL.U32；以及乘法相关指令，包括 ESP.VMUL.*、ESP.CMUL.*、ESP.FFT.AMS.* 和 ESP.FFT.CMUL.*。

向量移位指令使用 SAR 低 5 位作为右移或左移量。乘法相关指令使用 SAR 对中间乘积右移，从而影响最终结果精度。

4.4.1.3.2 SAR_BYTE

SAR_BYTE 以字节为单位保存移位量，用于处理非对齐的 128 位数据（参见章节 4.4.2）。向量算术指令的读写地址会被强制为 16 字节对齐，但应用中的数据地址未必始终满足该对齐要求。

ESP.LD.128.USAR.IP 与 ESP.LD.128.USAR.XP 在从内存读取 128 位数据时，会将访存地址的低 4 位（非对齐偏移）写入 SAR_BYTE。

使用 SAR_BYTE 的指令同样分为两类。第一类处理 QR 中的非对齐数据，包括 ESP.SRCQ.* 与 ESP.SRC.Q*：它们从非对齐地址前后的两个对齐地址各读 16 字节，拼接后按 SAR_BYTE 中的字节数右移，得到该非对齐地址处的 128 位数据。第二类在运算过程中处理非对齐数据，指令名通常带有 “.QUP” 后缀。

4.4.1.3.3 FFT_BIT_WIDTH

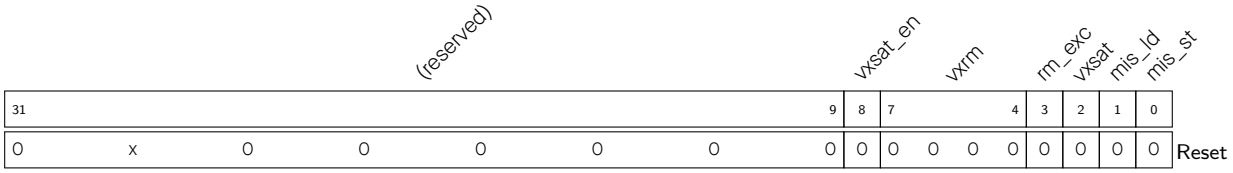
该专用寄存器仅供 ESP.FFT.BITREV 使用。其取值选择 ESP.FFT.BITREV 的工作模式：0 至 7 分别对应 3 位至 10 位模式。详见 ESP.FFT.BITREV。

4.4.1.3.4 UA_STATE

该专用寄存器仅供 ESP.FFT.AMS.S16.LD.INCP.UAUP 使用，用于保存此前从内存读入的非对齐 128 位数据。每次执行该指令时，寄存器内容与新读入的非对齐数据拼接，再经移位得到 128 位对齐结果。

4.4.1.3.5 内核配置与状态寄存器

Register 4.1. CFG



vxsat_en 使能饱和状态标志。

(R/W)

vxrm 配置舍入模式。参见表 4.4-3。

(R/W)

rm_exc UNNECESSARY 舍入异常状态（当指令 `rm` 字段与 vxrm 均为 DYNAMIC 时）。

(RO)

vxsat 饱和状态标志。

(RO)

mis_ld 使能硬件处理非对齐加载。

(R/W)

mis_st 使能硬件处理非对齐存储。

(R/W)

4.4.2 数据格式及对齐

扩展指令集支持 1 字节、2 字节、4 字节、8 字节和 16 字节数据格式。QACC_H 与 QACC_L 还使用 32 字节格式，但这两个专用寄存器与内存之间没有直接通路。可通过 5 个 4 字节 AR 寄存器或 2 个 16 字节 QR 寄存器读写 QACC_H/QACC_L。

表 4.4-2 列出常见格式的位宽与对齐要求（'x' 表示该位可为 0 或 1）。在所有格式中，HP 核以字节作为内存的最小可寻址单位。

表 4.4-2. 数据格式及对齐

数据格式	长度	内存中对齐地址低四位
1 字节	8 位	xxxx
2 字节	16 位	xxx0
4 字节	32 位	xx00
8 字节	64 位	x000
16 字节	128 位	0000

4.4.3 非对齐访存处理

ESP32-P4 HP 核 Xespv2p2 扩展的非对齐加载/存储可通过 mis_ld 与 mis_st 配置为以下两种模式：

- 0: 强制对齐模式
- 1: 硬件非对齐模式

4.4.3.1 强制对齐模式

在此模式下，扩展指令集的所有访存地址都会被强制对齐，即最低若干位被清零。例如，从 0x3fc8_0024 发起 128 位读取时，地址最低 4 位被清零，实际读出的是 0x3fc8_0020 处的数据。同理，64 位访问清除最低 3 位，32 位访问清除最低 2 位，16 位访问清除最低 1 位。

因此访存地址必须正确对齐，否则读回的数据与预期不符。应用代码中应显式声明变量或数组的对齐要求；16 字节对齐可覆盖大多数场景。

当地址是否对齐无法事先保证时，扩展提供 SAR_BYTE 以及 ESP.LD.128.USAR.*、ESP.SRC.* 等相关指令，用于重组非对齐的 128 位数据。

假设寄存器 a8 保存非对齐 128 位数据的地址，可用下列序列将其加载到 QR 寄存器 q2：

$ESP.LD.128.USAR.IP \quad q0, \quad a8, \quad 16$
$ESP.VLD.128.IP \quad q1, \quad a8, \quad 16$
$ESP.SRC.Q \quad q2, \quad q0, \quad q1$

4.4.3.2 硬件非对齐模式

在此模式下，硬件自动将非对齐访问拆分为多次对齐访问，结果与预期数据一致。

4.4.4 数据溢出与饱和

数据溢出是指结果无法放入目标寄存器。以 ESP.VMUL.S8 为例，两个 8 位数相乘得到 16 位结果，右移后仍为 16 位，但最终写入宽度为 8 位，因而可能溢出。

ESP32-P4 扩展以两种方式处理溢出：饱和与环绕（仅保留低位）。饱和将结果钳制到目标寄存器可表示的最小或最大值；使用饱和的指令会在描述中明确说明，例如 ESP.VADDS.* 会对加法结果取饱和。其余内部溢出一般采用环绕：只保留与目标位宽匹配的低位并写入结果寄存器。

1 位状态标志 vxsat 表示上一次运算是否发生饱和（仅在 vxsat_en 使能时有效）。读取 CFG 寄存器会自动清除该标志。

4.4.5 数据舍入

部分指令包含 `rm` 字段，用于选择右移时的舍入模式。表 4.4-3 列出 8 种模式及其编码。

当 `rm` 为 DYNAMIC (7) 时，指令使用 CFG 中的 4 位 vxrm。若 `rm` 与 vxrm 均为 DYNAMIC，则按 UNNECESSARY 处理：行为同 FLOOR，并在右移结果非精确时置位 rm_exc。

表 4.4-3. 舍入模式配置

模式	rm/vxrm 的值	描述
FLOOR	0	向负无穷舍入
CEILING	1	向正无穷舍入
UP	2	远离零舍入
DOWN	3	向零舍入
HALF_UP	4	舍入到最近值；若等距则向上舍入

模式	rm/vxrm 的值	描述
HALF_DOWN	5	舍入到最近值；若等距则向下舍入
HALF_EVEN	6	舍入到最近值；若等距则舍入到偶数邻居
DYNAMIC	7	使用 vxrm 中的模式。若 vxrm 也为 DYNAMIC，则按 UNNECESSARY（行为同 FLOOR，并由 rm_exc 报告）。

下表给出各舍入模式的示例结果。

表 4.4-4. 舍入操作结果

输入值	FLOOR	CEILING	UP	DOWN	HALF_UP	HALF_DOWN	HALF_EVEN	UNNECESSARY
5.5	5	6	6	5	6	5	6	Exception
2.5	2	3	3	2	3	2	2	Exception
1.6	1	2	2	1	2	2	2	Exception
1.1	1	2	2	1	1	1	1	Exception
1.0	1	1	1	1	1	1	1	1
-1.0	-1	-1	-1	-1	-1	-1	-1	-1
-1.1	-2	-1	-2	-1	-1	-1	-1	Exception
-1.6	-2	-1	-2	-1	-2	-2	-2	Exception
-2.5	-3	-2	-3	-2	-3	-2	-2	Exception
-5.5	-6	-5	-6	-5	-6	-5	-6	Exception

4.5 扩展指令列表

表 4.5-1 列出了扩展指令集中的指令类别。本节对各类别作简要说明。

表 4.5-1. 指令简介

扩展指令类别	指令[1]	参考章节
读内存指令	ESP.VLD.128.[XP/IP]	4.5.1
	ESP.VLD.[H/L].64.[XP/IP]	
	ESP.VLDBC.[8/16/32].[XP/IP]	
	ESP.VLDHBC.16.INCP	
	ESP.LD.128.USAR.[XP/IP]	
	ESP.LDQA.[U/S][8/16].128.[XP/IP]	
	ESP.LD.QACC.[H/L].[H/L].128.IP	
	ESP.LD.XACC.IP	
	ESP.LD.UA.STATE.IP	
	ESP.LDXQ.32	
	ESP.VLDEXT.[U/S].[8/16].[IP/XP]	
写内存指令	ESP.VST.128.[XP/IP]	4.5.2
	ESP.VST.[H/L].64.[XP/IP]	
	ESP.ST.QACC.[H/L].[H/L].128.IP	
	ESP.ST.[U/S].XACC.IP	
	ESP.ST.UA.STATE.IP	
	ESP.STXQ.32	
数据交换指令	ESP.MOVI.[8/16/32].A	4.5.3
	ESP.MOVI.[8/16/32].Q	
	ESP.MOVX.[R/W].CFG	
	ESP.MOVX.[R/W].SAR.BYTES	
	ESP.MOVX.[R/W].SAR	
	ESP.MOVX.[R/W].FFT.BIT.WIDTH	
	ESP.MOVX.[R/W].PERF	
	ESP.MOVX.[R/W].XACC.[L/H]	
	ESP.VZIP.[8/16/32]	
	ESP.VUNZIP.[8/16/32]	
	ESP.VEXT.[U/S].[8/16]	
	ESP.ZERO.Q	
	ESP.ZERO.QACC	
	ESP.ZERO.XACC	
	ESP.MOV.S8.QACC	
	ESP.MOV.S16.QACC	
	ESP.MOV.U8.QACC	
	ESP.MOV.U16.QACC	

见下页...

表 4.5-1 ... 接上页

扩展指令类别	指令[1]	参考章节
运算指令	ESP.VADD.S[8/16/32].[-/LD.INCP/ST.INCP]	4.5.4
	ESP.VSADDS.S[8/16]	
	ESP.VSADDS.U[8/16]	
	ESP.VSUB.S[8/16/32].[-/LD.INCP/ST.INCP]	
	ESP.VSSUBS.S[8/16]	
	ESP.VSSUBS.U[8/16]	
	ESP.VMUL.[U/S][8/16].[-/LD.INCP/ST.INCP]	
	ESP.VMULS32.S16xS16	
	ESP.CMUL.S8.[-/LD.INCP/ST.INCP]	
	ESP.CMUL.S16.[-/LD.INCP/ST.INCP]	
	ESP.VMULAS.[U/S][8/16].XACC.[-/LD.IP/LD.XP]	
	ESP.VMULAS.S[8/16].QACC_[H/L]	
	ESP.VMULAS.[U/S][8/16].QACC.[-/LD.IP/LD.XP/LDBC.INCP]	
	ESP.VSMULAS.S[8/16].QACC.[-/LD.INCP]	
	ESP.SRCMB.[U/S][8/16].QACC	
	ESP.SRCMB.[U/S][8/16].Q.QACC	
	ESP.SRS.[S/U].XACC	
	ESP.VRELU.S[8/16]	
	ESP.VPRELU.S[8/16]	
	ESP.SAT	
	ESP.ADDX2	
	ESP.ADDX4	
	ESP.SUBX2	
	ESP.SUBX4	
	ESP.VABS.[8/16/32]	
比较指令	ESP.VMAX.S[8/16/32].[-/LD.INCP/ST.INCP]	4.5.5
	ESP.MAX.S[16/32].A	
	ESP.VMIN.S[8/16/32].[-/LD.INCP/ST.INCP]	
	ESP.MIN.S[16/32].A	
	ESP.VCMP.[EQ/LT/GT].S[8/16/32]	
	ESP.VCLAMP.S16	
	ESP.VSAT.[U/S][8/16/32]	
按位逻辑操作指令	ESP.ORQ	4.5.6
	ESP.XORQ	
	ESP.ANDQ	
	ESP.NOTQ	

见下页...

表 4.5-1 ... 接上页

扩展指令类别	指令[1]	参考章节
移位操作	ESP.SRC.Q	4.5.7
	ESP.SRC.Q.LD.[XP/IP]	
	ESP.SLCI.2Q	
	ESP.SLCXXP.2Q	
	ESP.SRCI.2Q	
	ESP.SRCXXP.2Q	
	ESP.SRCQ.128.ST.INCP	
	ESP.VSRD.[8/16/32]	
	ESP.VSLD.[8/16/32]	
	ESP.VSR.U32	
	ESP.VSR.S32	
	ESP.VSL.S32	
	ESP.VSL.U32	
FFT 专用指令	ESP.FFT.R2BF.S16.[-/ST.INCP]	4.5.8
	ESP.FFT.CMUL.S16.[LD.XP/ST.XP]	
	ESP.FFT.BITREV	
	ESP.FFT.AMS.S16.[LD.INCP.UAUP/LD.INCP/LD.R32.DECP/ST.INCP]	
	ESP.FFT.VST.R32.DECP	

[1] 这些指令的详细介绍请参考 4.7 章节。

4.5.1 读内存指令

读内存指令依据保存访存地址的 AR 寄存器发出虚地址访问。多数指令先读内存，再更新地址。ESP.LDXQ.32 是例外：它先用立即数从 QR 中选取 16 位元素加到地址上，再执行加载。

下表概述读内存指令的访存行为。详情见章节 4.7。

表 4.5-2. 读内存指令

指令	定义
ESP.VLD.128.XP	读取 16-byte 数据，然后将 AR 寄存器中存储的数值加到访存地址上。
ESP.VLD.128.IP	读取 16-byte 数据，然后将立即数加到访存地址上。
ESP.VLD.[H/L].64.XP	读取 8-byte 数据，然后将 AR 寄存器中存储的数值加到访存地址上。
ESP.VLD.[H/L].64.IP	读取 8-byte 数据，然后将立即数加到访存地址上。
ESP.VLDBC.[8/16/32].XP	读取 1-byte/2-byte/4-byte 数据，然后将 AR 寄存器中存储的数值加到访存地址上。
ESP.VLDBC.[8/16/32].IP	读取 1-byte/2-byte/4-byte 数据，然后将立即数加到访存地址上。
ESP.VLDHBC.16.INCP	读取 16-byte 数据，然后将访存地址加 16。
ESP.LD.128.USAR.XP	读取 16-byte 数据，然后将 AR 寄存器中存储的数值加到访存地址上。
ESP.LD.128.USAR.IP	读取 16-byte 数据，然后将立即数加到访存地址上。
ESP.LDQA.U8.128.[XP/IP]	读取 16-byte 数据，按 1-byte 切片并零扩展为 32-bit，写入 QACC_H 与 QACC_L，再将 AR 值或立即数加到访存地址。
ESP.LDQA.U16.128.[XP/IP]	读取 16-byte 数据，按 2-byte 切片并零扩展为 64-bit，写入 QACC_H 与 QACC_L，再将 AR 值或立即数加到访存地址。
ESP.LDQA.S8.128.[XP/IP]	读取 16-byte 数据，按 1-byte 切片并符号扩展为 32-bit，写入 QACC_H 与 QACC_L，再将 AR 值或立即数加到访存地址。
ESP.LDQA.S16.128.[XP/IP]	读取 16-byte 数据，按 2-byte 切片并符号扩展为 64-bit，写入 QACC_H 与 QACC_L，再将 AR 值或立即数加到访存地址。
ESP.LD.QACC.[H/L].H.128.IP	读取 16-byte 数据，然后将立即数加到访存地址上。
ESP.LD.QACC.[H/L].L.128.IP	读取 16-byte 数据，然后将立即数加到访存地址上。
ESP.LD.XACC.IP	读取 8-byte 数据，然后将立即数加到访存地址上。
ESP.LD.UA.STATE.IP	读取 16-byte 数据，然后将立即数加到访存地址上。
ESP.LDXQ.32	先更新访存地址，再读取 4-byte 数据。
ESP.VLDEXT.[U/S][8/16].[IP/XP]	将加载数据按 8-bit 段向量扩展为 16-bit 段。

4.5.2 写内存指令

写内存指令依据保存访存地址的 AR 寄存器发出虚地址访问。多数指令先写内存，再更新地址。ESP.STXQ.32 是例外：它先用立即数从 QR 中选取 16 位元素加到地址上，再执行存储。

下表概述写内存指令的访存行为。详情见章节 4.7。

表 4.5-3. 写内存指令

指令	定义
ESP.VST.128.XP	将 16 字节数据写入内存，然后将 AR 寄存器中的值加到访问地址上。
ESP.VST.128.IP	将 16 字节数据写入内存，然后将立即数加到访问地址上。
ESP.VST.[H/L].64.XP	将 8 字节数据写入内存，然后将 AR 寄存器中的值加到访问地址上。
ESP.VST.[H/L].64.IP	将 8 字节数据写入内存，然后将立即数加到访问地址上。
ESP.ST.QACC.[H/L].H.128.IP	将 16 字节数据写入内存，然后将立即数加到访问地址上。
ESP.ST.QACC.[H/L].L.128.IP	将 16 字节数据写入内存，然后将立即数加到访问地址上。
ESP.ST.[U/S].XACC.IP	将 XACC 寄存器中的值进行零扩展或符号扩展为 8 字节数据写入内存，然后将立即数加到访问地址上。

指令	定义
ESP.ST.UA.STATE.IP	将 16 字节数据写入内存，然后将立即数加到访问地址上。
ESP.STXQ.32	先更新访问地址，然后将 4 字节数据写入内存。

4.5.3 数据交换指令

数据交换指令用于在寄存器之间传递数据。因寄存器位宽不同，立即数用于选择元素或字段，并提供零扩展与符号扩展指令。

详情见章节 4.7。

表 4.5-4. 数据交换指令

指令	定义
ESP.MOVI.8.A	将 QR 寄存器中的一段 8-bit 数据赋值给 AR 寄存器。
ESP.MOVI.16.A	将 QR 寄存器中的一段 16-bit 数据赋值给 AR 寄存器。
ESP.MOVI.32.A	将 QR 寄存器中的一段 32-bit 数据赋值给 AR 寄存器。
ESP.MOVI.8.Q	将 AR 寄存器中存储的数据赋值给 QR 寄存器中的一段 8-bit 数据空间。
ESP.MOVI.16.Q	将 AR 寄存器中存储的数据赋值给 QR 寄存器中的一段 16-bit 数据空间。
ESP.MOVI.32.Q	将 AR 寄存器中存储的数据赋值给 QR 寄存器中的一段 32-bit 数据空间。
ESP.MOVX.R.CFG	将 CFG 值读入 AR 寄存器。
ESP.MOVX.W.CFG	写入 CFG 寄存器。
ESP.MOVX.R.SAR.BYTES	将 SAR_BYTE 值读入 AR 寄存器。
ESP.MOVX.W.SAR.BYTES	写入 SAR_BYTE 寄存器。
ESP.MOVX.R.SAR	将 SAR 值读入 AR 寄存器。
ESP.MOVX.W.SAR	写入 SAR 寄存器。
ESP.MOVX.R.FFT.BIT.WIDTH	将 FFT_BIT_WIDTH 值读入 AR 寄存器。
ESP.MOVX.W.FFT.BIT.WIDTH	写入 FFT_BIT_WIDTH 寄存器。
ESP.MOVX.R.PERF	将性能计数器值读入 AR 寄存器。
ESP.MOVX.R.XACC.H	将 XACC 的高 8-bit 值读入 AR 寄存器。
ESP.MOVX.R.XACC.L	将 XACC 的低 32-bit 值读入 AR 寄存器。
ESP.MOVX.W.XACC.H	写入 XACC 的高 8-bit 值。
ESP.MOVX.W.XACC.L	写入 XACC 的低 32-bit 值。
ESP.VZIP.[8/16/32]	以 1-byte/2-byte/4-byte 为单位对两个 QR 寄存器进行编码。
ESP.VUNZIP.[8/16/32]	以 1-byte/2-byte/4-byte 为单位对两个 QR 寄存器进行解码。
ESP.VZIPT.[8/16]	以 1-byte/2-byte 为单位对 3 个 QR 寄存器进行编码。
ESP.VUNZIPT.[8/16]	以 1-byte/2-byte 为单位对 3 个 QR 寄存器进行解码。
ESP.VEXT.[U/S]8	将加载数据按 8-bit 段向量扩展为 16-bit 段。
ESP.VEXT.[U/S]16	将加载数据按 16-bit 段向量扩展为 32-bit 段。
ESP.ZERO.Q	清除指定的 QR 寄存器。
ESP.ZERO.QACC	清除 QACC_H 和 QACC_L 寄存器。
ESP.ZERO.XACC	清除指定的 XACC 寄存器。
ESP.MOV.S8.QACC	按 1-byte 对 QR 寄存器切片，并符号扩展为 32-bit 数据，然后将其赋值给 QACC_H 和 QACC_L 寄存器。
ESP.MOV.S16.QACC	按 2-byte 对 QR 寄存器切片，并符号扩展为 64-bit 数据，然后将其赋值给 QACC_H 和 QACC_L 寄存器。

指令	定义
ESP.MOV.U8.QACC	按 1-byte 对 QR 寄存器切片，并零扩展为 32-bit 数据，然后将其赋值给 QACC_H 和 QACC_L 寄存器。
ESP.MOV.U16.QACC	按 2-byte 对 QR 寄存器切片，并零扩展为 64-bit 数据，然后将其赋值给 QACC_H 和 QACC_L 寄存器。

4.5.4 运算指令

运算指令执行 SIMD 向量运算，包括向量加法、乘法、复数乘法、乘累加以及向量–标量乘累加等。

4.5.4.1 向量加法指令

ESP32-P4 提供以 1 字节、2 字节和 4 字节为元素宽度的向量加减法指令。

由于向量操作数常存放在内存中，部分向量加法指令将运算与 16 字节访存合并，并在访存后将地址寄存器加 16，指向下一块连续的 16 字节数据。请按算法需求选择指令。

这些指令还对加减法结果取饱和，以保持数值范围。

表 4.5-5. 向量加法运算指令

指令	定义
ESP.VADD.S[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量加法操作。
ESP.VADD.S[8/16/32].LD.INCP	对 1 字节/2 字节/4 字节数据执行向量加法操作，并同时从内存读取 16 字节数据。
ESP.VADD.S[8/16/32].ST.INCP	对 1 字节/2 字节/4 字节数据执行向量加法操作，并同时将 16 字节数据写入内存。
ESP.VSUB.S[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量减法操作。
ESP.VSUB.S[8/16/32].LD.INCP	对 1 字节/2 字节/4 字节数据执行向量减法操作，并同时从内存读取 16 字节数据。
ESP.VSUB.S[8/16/32].ST.INCP	对 1 字节/2 字节/4 字节数据执行向量减法操作，并同时将 16 字节数据写入内存。
ESP.VSSUBS.S[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量减法操作。
ESP.VSSUBS.U[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量减法操作。
ESP.VSADDS.S[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量加法操作。
ESP.VSADDS.U[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量加法操作。

4.5.4.2 向量乘法指令

ESP32-P4 提供有符号与无符号的 1 字节、2 字节向量乘法指令。

部分向量乘法指令将乘法与 16 字节访存合并，并在访存后将地址加 16。请按算法需求选择指令。

表 4.5-6. 运算指令

指令	定义
ESP.VMUL.U[8/16]	对无符号 1 字节/2 字节数据执行向量乘法操作。
ESP.VMUL.S[8/16]	对有符号 1 字节/2 字节数据执行向量乘法操作。

指令	定义
ESP.VMUL.U[8/16].LD.INCP	对无符号 1 字节/2 字节数据执行向量乘法操作，同时从内存读取 16 字节数据。
ESP.VMUL.S[8/16].LD.INCP	对有符号 1 字节/2 字节数据执行向量乘法操作，同时从内存读取 16 字节数据。
ESP.VMUL.U[8/16].ST.INCP	对无符号 1 字节/2 字节数据执行向量乘法操作，同时将 16 字节数据写入内存。
ESP.VMUL.S[8/16].ST.INCP	对有符号 1 字节/2 字节数据执行向量乘法操作，同时将 16 字节数据写入内存。
ESP.VMUL.S32.S16xS16	执行 16 位向量乘法操作并右移结果。

4.5.4.3 向量复数乘法指令

ESP32-P4 提供 1 字节与 2 字节有符号数据的向量复数乘法指令。

部分复数乘法指令将运算与 16 字节访存合并，并在访存后将地址加 16。请按算法需求选择指令。

表 4.5-7. 运算指令

指令	定义
ESP.CMUL.S16	对 2 字节数据执行向量复数乘法操作。
ESP.CMUL.S16.LD.INCP	对 2 字节数据执行向量复数乘法操作，并同时从内存读取 16 字节数据。
ESP.CMUL.S16.ST.INCP	对 2 字节数据执行向量复数乘法操作，并同时将 16 字节数据写入内存。
ESP.CMUL.S8	对 1 字节数据执行向量复数乘法操作。
ESP.CMUL.S8.LD.INCP	对 1 字节数据执行向量复数乘法操作，并同时从内存读取 16 字节数据。
ESP.CMUL.S8.ST.INCP	对 1 字节数据执行向量复数乘法操作，并同时将 16 字节数据写入内存。

4.5.4.4 向量乘累加指令

ESP32-P4 提供两类向量乘累加指令：一类累加到 40 位 XACC；另一类累加到 QACC_H 与 QACC_L 的对应分段。两类均支持 1 字节与 2 字节分段。

部分指令还并行执行 16 字节访存，并在其后按 AR 值或立即数更新地址。

表 4.5-8. 运算指令

指令	定义
ESP.VMULAS.[U/S][8/16].XACC	对 1 字节/2 字节段的有符号/无符号数据执行向量乘法累加操作，并将结果暂存到 XACC 寄存器。
ESP.VMULAS.[U/S][8/16].XACC.LD.IP	对 1 字节/2 字节段的有符号/无符号数据执行向量乘法累加操作，并将结果暂存到 XACC 寄存器，然后从内存读取 16 字节数据。将立即数加到地址寄存器。

指令	定义
ESP.VMULAS.[U/S][8/16].XACC.LD.XP	对 1 字节/2 字节段的有符号/无符号数据执行向量乘法累加操作，并将结果暂存到 XACC 寄存器，然后从内存读取 16 字节数据。将 AR 寄存器的值加到地址寄存器。
ESP.VCMULAS.S[8/16].QACC.[H/L]	对 1 字节/2 字节段的有符号数据执行向量乘法操作，并将结果累加到 QACC_H 或 QACC_L 寄存器的相应位段。
ESP.VCMULAS.S[8/16].QACC.[H/L].LD.IP	对 1 字节/2 字节段的有符号数据执行向量乘法操作，并将结果累加到 QACC_H 或 QACC_L 寄存器的相应位段。
ESP.VCMULAS.S[8/16].QACC.[H/L].LD.XP	对 1 字节/2 字节段的有符号数据执行向量乘法操作，并将结果累加到 QACC_H 或 QACC_L 寄存器的相应位段。
ESP.VMULAS.[U/S][8/16].QACC	对 1 字节/2 字节段的有符号/无符号数据执行向量乘法操作，并将结果累加到 QACC_H 和 QACC_L 寄存器的相应位段。
ESP.VMULAS.[U/S][8/16].QACC.LD.IP	对 1 字节/2 字节段的有符号/无符号数据执行向量乘法操作，并将结果累加到 QACC_H 和 QACC_L 寄存器的相应位段，然后从内存读取 16 字节数据。将立即数加到地址寄存器。
ESP.VMULAS.[U/S][8/16].QACC.LD.XP	对 1 字节/2 字节段的有符号/无符号数据执行向量乘法操作，并将结果累加到 QACC_H 和 QACC_L 寄存器的相应位段，然后从内存读取 16 字节数据。将 AR 寄存器的值加到地址寄存器。
ESP.VMULAS.[U/S][8/16].QACC.LDBC.INCP	对 1 字节/2 字节段的有符号/无符号数据执行向量乘法操作，并将结果累加到 QACC_H 和 QACC_L 寄存器的相应位段。然后从内存读取 1 字节/2 字节数据，并广播到 128 位 QR 寄存器。将地址寄存器加 16。

4.5.4.5 向量和标量乘累加指令

此类指令与基于 QACC 的向量乘累加类似，但一个操作数为向量、另一个为标量。同样提供可在运算同时访存的变体。

表 4.5-9. 向量与标量乘累加指令

指令	定义
ESP.VSMULAS.S[8/16].QACC	有符号 1 字节或有符号 2 字节分片的向量与标量乘累加。
ESP.VSMULAS.S[8/16].QACC.LD.INCP	有符号 1 字节或有符号 2 字节分片的向量与标量乘累加。同时从内存读取 16 字节数据，并对地址寄存器增加 16。

4.5.4.6 其他指令

本组指令包括对 QACC_H、QACC_L 与 XACC 中乘累加结果的算术右移；可通过设置移位量得到所需精度。还包括带使能条件的向量乘法。

表 4.5-10. 其他指令

指令	定义
ESP.SRCMB.S[8/16].QACC	对 QACC_H 和 QACC_L 寄存器内数值按段进行符号右移。
ESP.SRCMB.U[8/16].QACC	对 QACC_H 和 QACC_L 寄存器内数值按段进行无符号右移。
ESP.SRCMB.S[8/16].Q.QACC	对 QACC_H 和 QACC_L 寄存器内数值按段进行符号右移。
ESP.SRCMB.U[8/16].Q.QACC	对 QACC_H 和 QACC_L 寄存器内数值按段进行无符号右移。
ESP.SRS.S.XACC	对 XACC 寄存器内数值进行符号右移。
ESP.SRS.U.XACC	对 XACC 寄存器内数值进行无符号右移。
ESP.VRELU.S[8/16]	根据使能条件执行向量与标量乘法。
ESP.VPRELU.S[8/16]	根据使能条件执行向量与向量乘法。
ESP.SAT	对 32 位通用寄存器进行饱和操作，限制在 rs1 和 rs2 之间。
ESP.ADDX2	对 rs1 和 rs2 执行加法运算。
ESP.ADDX4	对 rs1 和 rs2 执行加法运算。
ESP.SUBX2	对 rs1 和 rs2 执行减法运算。
ESP.SUBX4	对 rs1 和 rs2 执行减法运算。
ESP.VABS.[8/16/32]	对有符号数取绝对值。

4.5.5 比较指令

向量比较指令以 1 字节、2 字节或 4 字节为元素宽度。包括取较大值/较小值，以及等于、大于、小于比较时生成全 1 或全 0 掩码。

部分比较指令还会执行 16 字节访存，并在其后将地址加 16。请按算法需求选择指令。

表 4.5-11. 比较指令

指令	定义
ESP.VMAX.S[8/16/32]	取两个 1-字节/2-字节/4-字节数据间较大值。
ESP.VMAX.S[8/16/32].LD.INCP	取两个 1-字节/2-字节/4-字节数据间较大值。同时从内存读取 16-字节数据。
ESP.VMAX.S[8/16/32].ST.INCP	取两个 1-字节/2-字节/4-字节间数据较大值。同时将 16-字节数据写入内存。
ESP.VMAX.S[16/32].A	在 QR 寄存器的所有元素中查找最大有符号值。
ESP.VMIN.S[8/16/32]	取两个 1-字节/2-字节/4-字节数据间较小值。
ESP.VMIN.S[8/16/32].LD.INCP	取两个 1-字节/2-字节/4-字节数据间较小值。同时从内存读取 16-字节数据。
ESP.VMIN.S[8/16/32].ST.INCP	取两个 1-字节/2-字节/4-字节数据间较小值。同时将 16-字节数据写入内存。
ESP.VMIN.S[16/32].A	在 QR 寄存器的所有元素中查找最小有符号值。
ESP.VCMP.EQ.S[8/16/32]	两个 1-字节/2-字节/4-字节数据，前者等于后者将 1-字节/2-字节/4-字节所有比特位置 1 反之置 0。
ESP.VCMP.LT.S[8/16/32]	两个 1-字节/2-字节/4-字节数据，前者小于后者将 1-字节/2-字节/4-字节所有比特位置 1 反之置 0。
ESP.VCMP.GT.S[8/16/32]	两个 1-字节/2-字节/4-字节数据，前者大于后者将 1-字节/2-字节/4-字节所有比特位置 1 反之置 0。

指令	定义
ESP.VCLAMP.S16	将 QR 寄存器中的所有元素裁剪至指定范围。
ESP.VSAT.[U/S][8/16/32]	对 QR 寄存器中的所有元素执行饱和运算。

4.5.6 按位逻辑操作指令

按位逻辑指令包括按位逻辑或 (OR)、按位逻辑与 (AND)、按位逻辑异或 (XOR) 和按位取反 (NOT) 指令。

表 4.5-12. 按位逻辑操作指令

指令	定义
ESP.ORQ	按位逻辑或 $qa = qx \| qy$
ESP.XORQ	按位逻辑异或 $qa = qx \hat{} qy$
ESP.ANDQ	按位逻辑与 $qa = qx \& qy$
ESP.NOTQ	按位取反 $qa = \sim qx$

4.5.7 移位指令

移位指令包括以 4 字节为单位的向量左移/右移，以及对拼接后 16 字节数据的移位。向量移位的移位量由 SAR 提供；拼接数据移位的移位量可由 SAR_BYTE、立即数或 AR 低位提供。请按应用选择指令。在适用情形下，这些移位遵循有符号移位语义。

表 4.5-13. 移位指令

指令	定义
ESP.SRC.Q	16-字节拼接后逻辑右移，移位数值由 SAR_BYTE 寄存器决定。
ESP.SRC.Q.LD.XP	对拼接的 16 字节数据执行逻辑右移，移位值由 SAR_BYTE 寄存器决定。与此同时，从内存读取 16 字节数据，并将寄存器值加到读取地址上。
ESP.SRC.Q.LD.IP	对拼接的 16 字节数据执行逻辑右移，移位值由 SAR_BYTE 寄存器决定。与此同时，从内存读取 16 字节数据，并将立即数加到读取地址上。
ESP.SLCI.2Q	对拼接的 16 字节数据执行逻辑左移，移位值由立即数决定。
ESP.SLCXXP.2Q	对拼接的 16 字节数据执行逻辑左移，移位值由 AR 寄存器中的值决定。
ESP.SRCI.2Q	对拼接的 16 字节数据执行逻辑右移，移位值由立即数决定。
ESP.SRCXXP.2Q	对拼接的 16 字节数据执行逻辑右移，移位值由 AR 寄存器中的值决定。
ESP.SRCQ.128.ST.INCP	对拼接的 16 字节数据执行逻辑右移，移位后的数据将被写入内存。
ESP.VSR.U32	对 4 字节数据执行向量算术右移。
ESP.VSR.S32	对 4 字节数据执行向量算术右移。
ESP.VSL.S32	对 4 字节有符号数据执行向量算术左移。
ESP.VSL.U32	对 4 字节无符号数据执行向量算术左移。
ESP.VSRD.[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量算术右移或左移。
ESP.VSLD.[8/16/32]	对 1 字节/2 字节/4 字节数据执行向量算术右移或左移。

4.5.8 快速傅里叶变换 (FFT) 指令

FFT 指令包括蝶形运算、比特反转以及实数 FFT 辅助指令。

4.5.8.1 蝶形运算指令

这些指令支持基数为 2 的蝶形运算。

表 4.5-14. 蝶形运算指令

指令	定义
ESP.FFT.R2BF.S16.	基-2 蝶形运算指令。
ESP.FFT.R2BF.S16.ST.INCP	基-2 蝶形运算同时将其中 16-字节计算结果写入内存。
ESP.FFT.CMUL.S16.LD.XP	基-2 蝶形复数运算指令。同时从内存读取 16-字节数据。
ESP.FFT.CMUL.S16.ST.XP	基-2 蝶形复数运算指令。同时将 16-字节数据（由运算结果、QR 寄存器部分片区数值组成）写入内存。

4.5.8.2 比特反转指令

比特反转位宽由 FFT_BIT_WIDTH 决定。

表 4.5-15. 比特反转指令

指令	定义
ESP.FFT.BITREV	按 FFT_BIT_WIDTH 进行比特反转

4.5.8.3 实数 FFT 指令

单条实数 FFT 指令可组合加法、乘法、移位等相关运算。

表 4.5-16. 实数 FFT 指令

指令	定义
ESP.FFT.AMS.S16.LD.INCP.UAUP	完成复杂运算的同时从内存读取 16-字节数据并输出 16-字节对齐数据。
ESP.FFT.AMS.S16.LD.INCP	完成复杂运算的同时从内存读取 16-字节数据。并对地址寄存器增加 16。
ESP.FFT.AMS.S16.LD.R32.DECP	完成复杂运算的同时从内存读取 16-字节数据，并翻转字顺序。然后对地址寄存器增加 16。
ESP.FFT.AMS.S16.ST.INCP	完成复杂运算的同时将 16-字节数据（由 AR 寄存器、QR 寄存器部分数据段组成）写入内存。
ESP.FFT.VST.R32.DECP	将 QR 寄存器按 2-字节为单位分片，移位结果并将 16 字节数据写入内存。

4.6 指令性能

理想情况下，流水线处理器每个周期发出一条指令。ESP32-P4 处理器采用五级流水线：F（取指）、D（译码）、E（执行）、M（访存）和 W（写回）。表 4.6-1 概述各级操作。

表 4.6-1. ESP32-P4 处理器五级流水线

流水级	编号	操作
F	-	取指令
D	0	读取通用寄存器 AR、QR 译码、检测内部锁并转发操作数
E	1	对算术指令，ALU 执行加、减、乘等运算 对读内存指令，生成虚地址 对分支指令，选择跳转目标
M	2	发出内存读写
W	3	将计算结果与读入数据写回寄存器

只有当指令操作数与所需硬件资源就绪时，指令才能进入流水线。实际运行中，下列冒险可能导致流水线停顿并推迟发射。

4.6.1 数据冒险

若指令 A 向寄存器 X（显式通用寄存器或隐式专用寄存器）写结果，而指令 B 以同一寄存器为输入，则称 B 依赖于 A。若 A 在流水级 SA 结束时产生结果，B 需在流水级 SB 开始时读取，则 A 必须比 B 提前 $D = max(SA - SB + 1, 0)$ 个周期发射。

若在 A 之后不足 D 个周期就取到 B，处理器会推迟发射 B，直到满足 D 个周期。这种因流水线交互造成的延迟称为内部锁 (interlock)。

例如，若 A 的 SA 为 W、B 的 SB 为 E，则 $D = max(2 - 1 + 1, 0) = 2$，如图 4.6-1 所示。

当结果在某级末可用时，相关指令通常需等待其写回寄存器后再读取。处理器也支持旁路 (bypass)：可将产生级的数据直接转发到消费级，而无需等待写回。

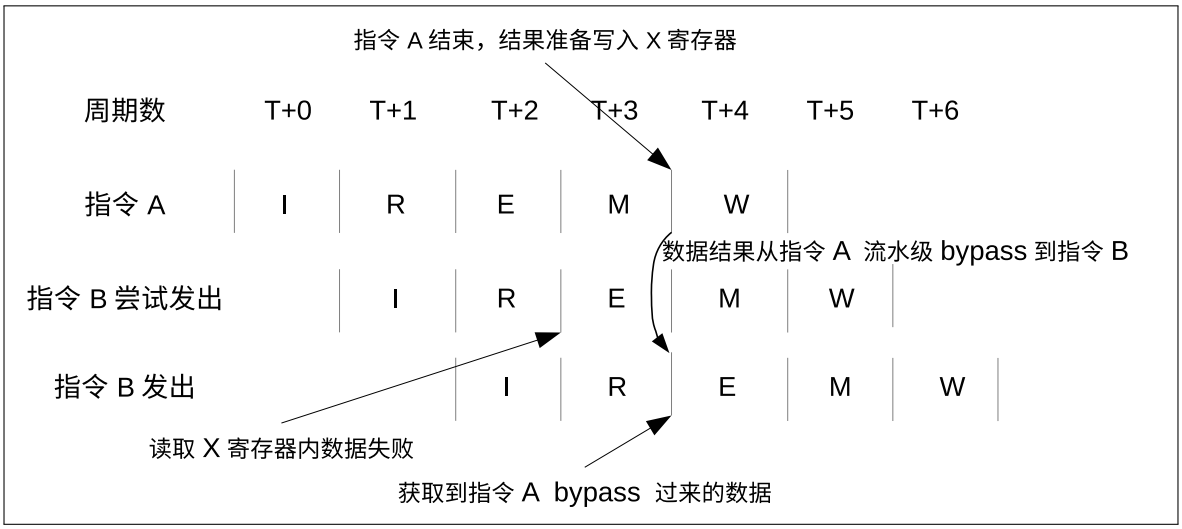


图 4.6-1. 操作数依赖导致的内部锁

指令间的数据依赖取决于所用操作数及其读/写所处流水级。章节 4.7 列出各指令操作数的 use/def 流水级。

4.6.2 硬件资源冒险

若多条指令在同一周期争用同一硬件资源，则仅允许一条占用，其余被延迟。例如处理器仅有 8 个 16 位乘法器。若指令 C 在 M 级需要全部 8 个、指令 D 在 E 级需要其中 4 个，且如图 4.6-2 所示分别在 T+0、T+1 发射，则在 T+3 会争用 4 个乘法器；处理器将把 D 推迟一个周期以避免冲突。

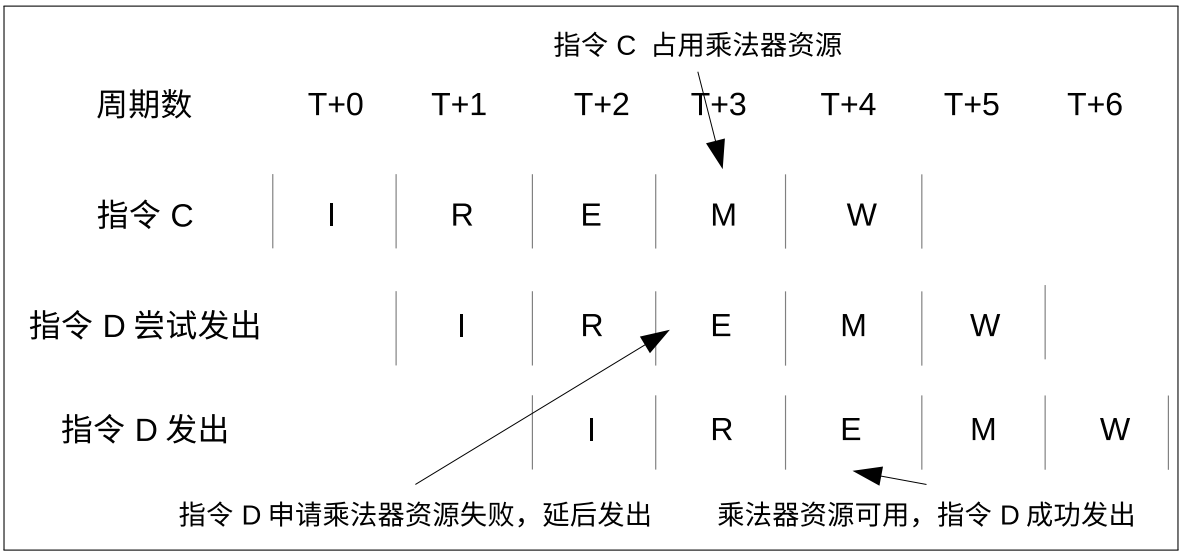


图 4.6-2. 硬件资源冒险

4.6.3 控制冒险

数据冒险与资源冒险常可通过重排代码缓解，控制冒险则更难隐藏。条件语句会编译为分支/跳转：条件成立则从目标地址取指，否则顺序取指。当分支跳转发生时，如图 4.6-3 所示，已位于 F、D 级的指令被丢弃，相当于流水线损失 2 个周期。

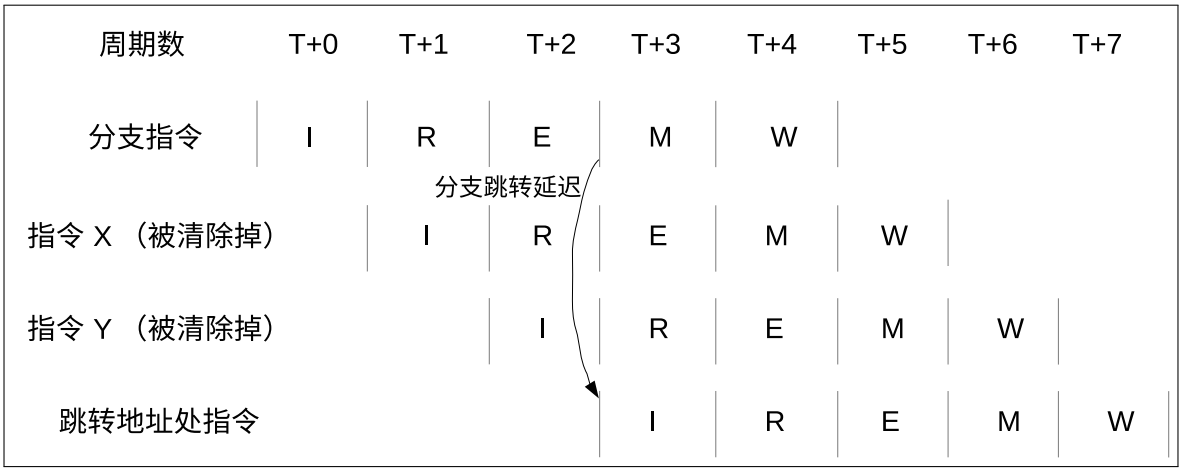


图 4.6-3. 控制冒险

4.7 扩展指令功能描述

4.7.1 指令符号与域定义

表 4.7-1 介绍了指令描述中涉及到的符号所代表的含义。阅读下文各条指令的功能描述时，请对照该表理解操作数与编码域。

`[N:M]` 表示一个字段中的位范围。比特上下限 N 和 M 都包含在内，表示该字段的位宽为 `(N-M+1)`。例如，`Qx[2:0]` 共有 3 个比特，分别是 bit0、bit1 和 bit2；`Qx[1]` 表示 bit1 的值。

表 4.7-1. 指令域名及含义

域名			含义
r*	32-bit RISC-V 标准通用寄存器	rs1	in 类型或 in-out 类型。32-bit 通用寄存器，用于存放地址和数据值。
		rs2	in 类型。32-bit 通用寄存器，用于存放移位值、自增值或计算结果。
		rd	out 类型。32-bit 通用寄存器，用于存放输出值。
q*	128-bit 通用寄存器	qu	in/out/inout 类型。128-bit 通用寄存器，用于存放读写数据或交换数据。
		qv	in/out/inout 类型。128-bit 通用寄存器。
		qx, qy, qw	in 类型。用于存放参与向量运算的数据。
		qz	out 类型。用于存放向量运算结果。
sat			1-bit 立即数。用于使能或禁用该指令结果的饱和处理。
rm			3-bit 立即数，范围为 0 到 7。用于选择右移操作的舍入模式。值为 7 (DYNAMIC) 时使用 CFG 寄存器中的 `vxrm`。
sel2			1-bit 立即数，范围为 0 到 1。用于选择信号。
sel4、upd4			2-bit 立即数，范围为 0 到 3。用于选择信号。
sel8			3-bit 立即数，范围为 0 到 7。用于选择信号。
sel16			4-bit 立即数，范围为 0 到 15。用于选择信号。
sar2			1-bit 立即数，范围为 0 到 1。表示移位的位数。
sar4			2-bit 立即数，范围为 0 到 3。表示移位的位数。
sar16			4-bit 立即数，范围为 0 到 15。表示移位的位数。
imm1			7-bit 无符号立即数，取值范围为 0 到 127，步长为 1。用于表示更新后的读写操作地址的大小。
imm2			7-bit 无符号立即数，取值范围为 0 到 254，步长为 2。用于表示更新后的读写操作地址的大小。
imm4			8-bit 有符号立即数，取值范围为 -256 到 252，步长为 8。用于表示更新后的读写操作地址的大小。
imm16			8-bit 有符号立即数，取值范围为 -2048 到 2032，步长为 16。用于表示更新后的读写操作地址的大小。

4.7.2 指令列表

4.7.2.1 ESP.VCMULAS.S16.QACC.H

语法

ESP.VCMULAS.S16.QACC.H qx,qy,sat

描述

此指令执行 16 位向量的乘法运算，并将结果存储在 QACC_H 中。

操作

```
add_0[64:0] = QACC_H[ 63:  0] + (qx[ 79: 64]*qy[ 79: 64] - qx[95:80]*qy[95:80])
add_1[64:0] = QACC_H[127: 64] + (qx[ 95: 80]*qy[ 79: 64] + qx[79:64]*qy[95:80])
add_2[64:0] = QACC_H[191:128] + (qx[111:96]*qy[111:96] - qx[127:112]*qy[127:112])
add_3[64:0] = QACC_H[255:192] + (qx[127:112]*qy[111:96] - qx[111:96]*qy[127:112])
QACC_H[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2

格式

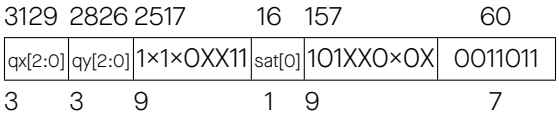


图 4.7-1. ESP.VCMULAS.S16.QACC.H

4.7.2.2 ESP.VCMULAS.S16.QACC.H.LD.IP

语法

ESP.VCMULAS.S16.QACC.H.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令执行 16 位向量的乘法，并将结果存储在 QACC_H 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 4 位符号扩展常数，然后左移 4 位。

imm16 的起始值为-128，结束值为 112，步长为 16。

操作

```
add_0[64:0] = QACC_H[63:0] + (qx[79:64]*qy[79:64] - qx[95:80]*qy[95:80])
add_1[64:0] = QACC_H[127:64] + (qx[95:80]*qy[79:64] + qx[79:64]*qy[95:80])
add_2[64:0] = QACC_H[191:128] + (qx[111:96]*qy[111:96] - qx[127:112]*qy[127:112])
add_3[64:0] = QACC_H[255:192] + (qx[127:112]*qy[111:96] - qx[111:96]*qy[127:112])

QACC_H[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2

格式

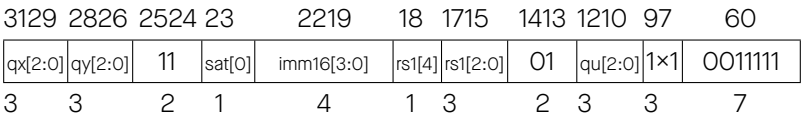


图 4.7-2. ESP.VCMULAS.S16.QACC.H.LD.IP

4.7.2.3 ESP.VCMULAS.S16.QACC.H.LD.XP

语法

ESP.VCMULAS.S16.QACC.H.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令执行 16 位向量的乘法，并将结果存储在 QACC_H 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值会增加寄存器 rs2 中的值。

操作

```
add_0[64:0] = QACC_H[63:0] + (qx[79:64]*qy[79:64] - qx[95:80]*qy[95:80])
add_1[64:0] = QACC_H[127:64] + (qx[95:80]*qy[79:64] + qx[79:64]*qy[95:80])
add_2[64:0] = QACC_H[191:128] + (qx[111:96]*qy[111:96] - qx[127:112]*qy[127:112])
add_3[64:0] = QACC_H[255:192] + (qx[127:112]*qy[111:96] - qx[111:96]*qy[127:112])

QACC_H[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2

格式

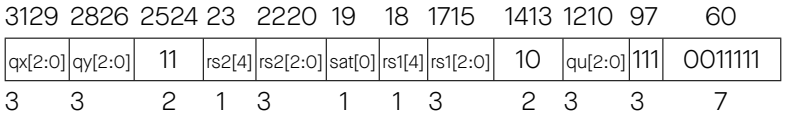


图 4.7-3. ESP.VCMULAS.S16.QACC.H.LD.XP

4.7.2.4 ESP.VCMULAS.S16.QACC.L

语法

ESP.VCMULAS.S16.QACC.L qx,qy,sat

描述

此指令执行 16 位向量的乘法运算，并将结果存储在QACC_L中。

操作

```
add_0[64:0] = QACC_L[63:0] + (qx[15:0]*qy[15:0] - qx[31:16]*qy[31:16])
add_1[64:0] = QACC_L[127:64] + (qx[31:16]*qy[15:0] + qx[15:0]*qy[31:16])
add_2[64:0] = QACC_L[191:128] + (qx[47:32]*qy[47:32] - qx[63:48]*qy[63:48])
add_3[64:0] = QACC_L[255:192] + (qx[63:48]*qy[47:32] - qx[47:32]*qy[63:48])

QACC_L[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_L[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_L[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_L[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_L	2	2

格式

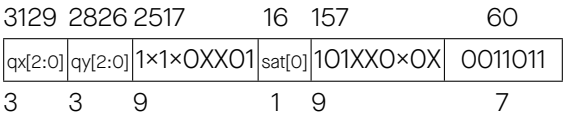


图 4.7-4. ESP.VCMULAS.S16.QACC.L

4.7.2.5 ESP.VCMULAS.S16.QACC.L.LD.IP

语法

ESP.VCMULAS.S16.QACC.L.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令执行 16 位向量的乘法，并将结果存储在 QACC_L 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，指令代码段中的 4 位符号扩展常数将寄存器 rs1 中的值增加，然后左移 4 位。

imm16 从-128 开始，到 112 结束，步长为 16。

操作

```
add_0[64:0] = QACC_L[63:0] + (qx[15:0]*qy[15:0] - qx[31:16]*qy[31:16])
add_1[64:0] = QACC_L[127:64] + (qx[31:16]*qy[15:0] + qx[15:0]*qy[31:16])
add_2[64:0] = QACC_L[191:128] + (qx[47:32]*qy[47:32] - qx[63:48]*qy[63:48])
add_3[64:0] = QACC_L[255:192] + (qx[63:48]*qy[47:32] - qx[47:32]*qy[63:48])

QACC_L[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_L[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_L[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_L[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_L	2	2

格式

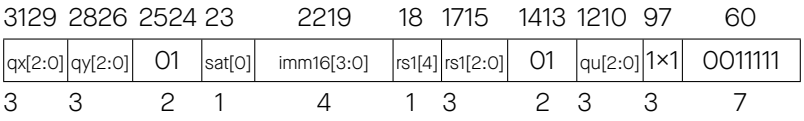


图 4.7-5. ESP.VCMULAS.S16.QACC.L.LD.IP

4.7.2.6 ESP.VCMULAS.S16.QACC.L.LD.XP

语法

ESP.VCMULAS.S16.QACC.L.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令执行 16 位向量的乘法，并将结果存储在 QACC_L 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值会增加寄存器 rs2 中的值。

操作

```
add_0[64:0] = QACC_L[63:0] + (qx[15:0]*qy[15:0] - qx[31:16]*qy[31:16])
add_1[64:0] = QACC_L[127:64] + (qx[31:16]*qy[15:0] + qx[15:0]*qy[31:16])
add_2[64:0] = QACC_L[191:128] + (qx[47:32]*qy[47:32] - qx[63:48]*qy[63:48])
add_3[64:0] = QACC_L[255:192] + (qx[63:48]*qy[47:32] - qx[47:32]*qy[63:48])

QACC_L[ 63:  0] = min(max(add_0[64:0], -2^{63}), 2^{63}-1)
QACC_L[127: 64] = min(max(add_1[64:0], -2^{63}), 2^{63}-1)
QACC_L[191:128] = min(max(add_2[64:0], -2^{63}), 2^{63}-1)
QACC_L[255:192] = min(max(add_3[64:0], -2^{63}), 2^{63}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs1	1	-
qx	1	-
qy	1	-
QACC_L	2	2

格式

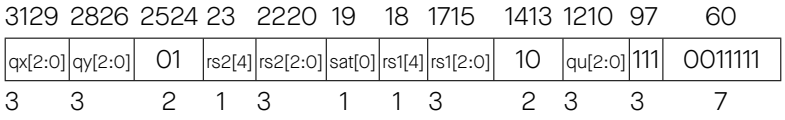


图 4.7-6. ESP.VCMULAS.S16.QACC.L.LD.XP

4.7.2.7 ESP.VCMULAS.S8.QACC.H

语法

ESP.VCMULAS.S8.QACC.H qx,qy,sat

描述

此指令执行 8 位向量的乘法运算，并将结果存储在 QACC_H 中。

操作

```
add_0[32:0] = QACC_H[ 31:  0] + (qx[ 71: 64]*qy[ 71: 64] - qx[ 79: 72]*qy[ 79: 72])
add_1[32:0] = QACC_H[ 63: 32] + (qx[ 79: 72]*qy[ 71: 64] - qx[ 71: 64]*qy[ 79: 72])

add_2[32:0] = QACC_H[ 95: 64] + (qx[ 87: 80]*qy[ 87: 80] - qx[ 95: 88]*qy[ 95: 88])
add_3[32:0] = QACC_H[127: 96] + (qx[ 95: 88]*qy[ 87: 80] - qx[ 87: 80]*qy[ 95: 88])

add_4[32:0] = QACC_H[159:128] + (qx[103: 96]*qy[103: 96] - qx[111:104]*qy[111:104])
add_5[32:0] = QACC_H[191:160] + (qx[111:104]*qy[103: 96] - qx[103: 96]*qy[111:104])

add_6[32:0] = QACC_H[223:192] + (qx[119:112]*qy[119:112] - qx[127:120]*qy[127:120])
add_7[32:0] = QACC_H[255:224] + (qx[127:120]*qy[119:112] - qx[119:112]*qy[127:120])

QACC_H[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2

格式

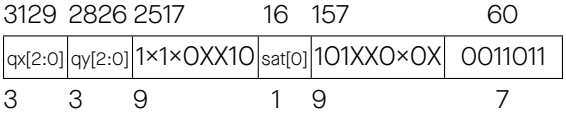


图 4.7-7. ESP.VCMULAS.S8.QACC.H

4.7.2.8 ESP.VCMULAS.S8.QACC.H.LD.IP

语法

ESP.VCMULAS.S8.QACC.H.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令执行 8 位向量的乘法，并将结果存储在 QACC_H 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，指令代码段中的 4 位符号扩展常数将寄存器 rs1 中的值增加，然后左移 4 位。

imm16 从-128 开始，到 112 结束，步长为 16。

操作

```
add_0[32:0] = QACC_H[ 31:  0] + (qx[ 71: 64]*qy[ 71: 64] - qx[ 79: 72]*qy[ 79: 72])
add_1[32:0] = QACC_H[ 63: 32] + (qx[ 79: 72]*qy[ 71: 64] - qx[ 71: 64]*qy[ 79: 72])

add_2[32:0] = QACC_H[ 95: 64] + (qx[ 87: 80]*qy[ 87: 80] - qx[ 95: 88]*qy[ 95: 88])
add_3[32:0] = QACC_H[127: 96] + (qx[ 95: 88]*qy[ 87: 80] - qx[ 87: 80]*qy[ 95: 88])

add_4[32:0] = QACC_H[159:128] + (qx[103: 96]*qy[103: 96] - qx[111:104]*qy[111:104])
add_5[32:0] = QACC_H[191:160] + (qx[111:104]*qy[103: 96] - qx[103: 96]*qy[111:104])

add_6[32:0] = QACC_H[223:192] + (qx[119:112]*qy[119:112] - qx[127:120]*qy[127:120])
add_7[32:0] = QACC_H[255:224] + (qx[127:120]*qy[119:112] - qx[119:112]*qy[127:120])

QACC_H[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + signed32(imm16)
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-

操作数	读流水级数	写流水级数
qy	1	-
QACC_H	2	2

格式

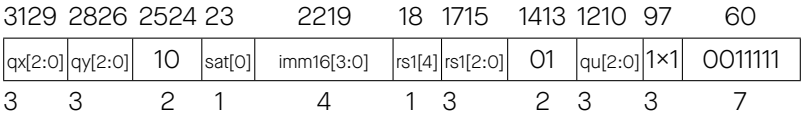


图 4.7-8. ESP.VCMULAS.S8.QACC.H.LD.IP

4.7.2.9 ESP.VCMULAS.S8.QACC.H.LD.XP

语法

ESP.VCMULAS.S8.QACC.H.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令执行 8 位向量的乘法，并将结果存储在 QACC_H 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值会增加寄存器 rs2 中的值。

操作

```
add_0[32:0] = QACC_H[ 31:  0] + (qx[ 71: 64]*qy[ 71: 64] - qx[ 79: 72]*qy[ 79: 72])
add_1[32:0] = QACC_H[ 63: 32] + (qx[ 79: 72]*qy[ 71: 64] - qx[ 71: 64]*qy[ 79: 72])

add_2[32:0] = QACC_H[ 95: 64] + (qx[ 87: 80]*qy[ 87: 80] - qx[ 95: 88]*qy[ 95: 88])
add_3[32:0] = QACC_H[127: 96] + (qx[ 95: 88]*qy[ 87: 80] - qx[ 87: 80]*qy[ 95: 88])

add_4[32:0] = QACC_H[159:128] + (qx[103: 96]*qy[103: 96] - qx[111:104]*qy[111:104])
add_5[32:0] = QACC_H[191:160] + (qx[111:104]*qy[103: 96] - qx[103: 96]*qy[111:104])

add_6[32:0] = QACC_H[223:192] + (qx[119:112]*qy[119:112] - qx[127:120]*qy[127:120])
add_7[32:0] = QACC_H[255:224] + (qx[127:120]*qy[119:112] - qx[119:112]*qy[127:120])

QACC_H[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-

操作数	读流水级数	写流水级数
qy	1	-
QACC_H	2	2

格式

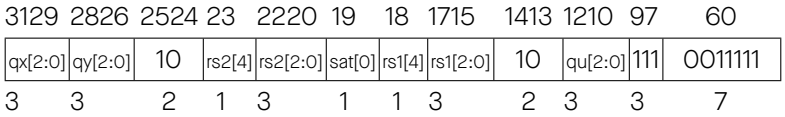


图 4.7-9. ESP.VCMULAS.S8.QACC.H.LD.XP

4.7.2.10 ESP.VCMULAS.S8.QACC.L

语法

ESP.VCMULAS.S8.QACC.L qx,qy,sat

描述

此指令执行 8 位向量的乘法运算，并将结果存储在 QACC_L 中。

操作

```
add_0[32:0] = QACC_L[ 31:  0] + (qx[  7:  0]*qy[  7:  0] - qx[ 15:  8]*qy[ 15:  8])
add_1[32:0] = QACC_L[ 63: 32] + (qx[ 15:  8]*qy[  7:  0] - qx[  7:  0]*qy[ 15:  8])

add_2[32:0] = QACC_L[ 95: 64] + (qx[ 23: 16]*qy[ 23: 16] - qx[ 31: 24]*qy[ 31: 24])
add_3[32:0] = QACC_L[127: 96] + (qx[ 31: 24]*qy[ 23: 16] - qx[ 23: 16]*qy[ 31: 24])

add_4[32:0] = QACC_L[159:128] + (qx[ 39: 32]*qy[ 39: 32] - qx[ 47: 40]*qy[ 47: 40])
add_5[32:0] = QACC_L[191:160] + (qx[ 47: 40]*qy[ 39: 32] - qx[ 39: 32]*qy[ 47: 40])

add_6[32:0] = QACC_L[223:192] + (qx[ 55: 48]*qy[ 55: 48] - qx[ 63: 56]*qy[ 63: 56])
add_7[32:0] = QACC_L[255:224] + (qx[ 63: 56]*qy[ 55: 48] - qx[ 55: 48]*qy[ 63: 56])

QACC_L[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_L	2	2

格式

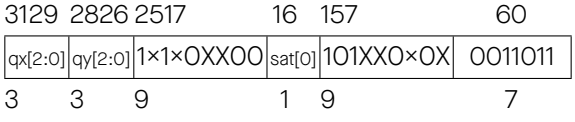


图 4.7-10. ESP.VCMULAS.S8.QACC.L

4.7.2.11 ESP.VCMULAS.S8.QACC.L.LD.IP

语法

ESP.VCMULAS.S8.QACC.L.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令执行 8 位向量的乘法，并将结果存储在 QACC_L 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 4 位符号扩展常数，然后左移 4 位。

imm16 的起始值为-128，结束值为 112，步长为 16。

操作

```
add_0[32:0] = QACC_L[ 31:  0] + (qx[  7:  0]*qy[  7:  0] - qx[ 15:  8]*qy[ 15:  8])
add_1[32:0] = QACC_L[ 63: 32] + (qx[ 15:  8]*qy[  7:  0] - qx[  7:  0]*qy[ 15:  8])

add_2[32:0] = QACC_L[ 95: 64] + (qx[ 23: 16]*qy[ 23: 16] - qx[ 31: 24]*qy[ 31: 24])
add_3[32:0] = QACC_L[127: 96] + (qx[ 31: 24]*qy[ 23: 16] - qx[ 23: 16]*qy[ 31: 24])

add_4[32:0] = QACC_L[159:128] + (qx[ 39: 32]*qy[ 39: 32] - qx[ 47: 40]*qy[ 47: 40])
add_5[32:0] = QACC_L[191:160] + (qx[ 47: 40]*qy[ 39: 32] - qx[ 39: 32]*qy[ 47: 40])

add_6[32:0] = QACC_L[223:192] + (qx[ 55: 48]*qy[ 55: 48] - qx[ 63: 56]*qy[ 63: 56])
add_7[32:0] = QACC_L[255:224] + (qx[ 63: 56]*qy[ 55: 48] - qx[ 55: 48]*qy[ 63: 56])

QACC_L[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-

操作数	读流水级数	写流水级数
qy	1	-
QACC_L	2	2

格式

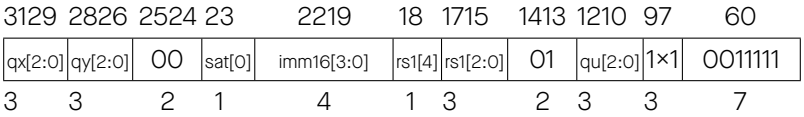


图 4.7-11. ESP.VCMULAS.S8.QACC.L.LD.IP

4.7.2.12 ESP.VCMULAS.S8.QACC.L.LD.XP

语法

ESP.VCMULAS.S8.QACC.L.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令执行 8 位向量的乘法，并将结果存储在 QACC_L 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值会增加寄存器 rs2 中的值。

操作

```
add_0[32:0] = QACC_L[ 31:  0] + (qx[  7:  0]*qy[  7:  0] - qx[ 15:  8]*qy[ 15:  8])
add_1[32:0] = QACC_L[ 63: 32] + (qx[ 15:  8]*qy[  7:  0] - qx[  7:  0]*qy[ 15:  8])

add_2[32:0] = QACC_L[ 95: 64] + (qx[ 23: 16]*qy[ 23: 16] - qx[ 31: 24]*qy[ 31: 24])
add_3[32:0] = QACC_L[127: 96] + (qx[ 31: 24]*qy[ 23: 16] - qx[ 23: 16]*qy[ 31: 24])

add_4[32:0] = QACC_L[159:128] + (qx[ 39: 32]*qy[ 39: 32] - qx[ 47: 40]*qy[ 47: 40])
add_5[32:0] = QACC_L[191:160] + (qx[ 47: 40]*qy[ 39: 32] - qx[ 39: 32]*qy[ 47: 40])

add_6[32:0] = QACC_L[223:192] + (qx[ 55: 48]*qy[ 55: 48] - qx[ 63: 56]*qy[ 63: 56])
add_7[32:0] = QACC_L[255:224] + (qx[ 63: 56]*qy[ 55: 48] - qx[ 55: 48]*qy[ 63: 56])

QACC_L[ 31:  0] = min(max(add_0[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(add_1[32:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(add_2[32:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(add_3[32:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(add_4[32:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(add_5[32:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(add_6[32:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(add_7[32:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-

操作数	读流水级数	写流水级数
qy	1	-
QACC_L	2	2

格式

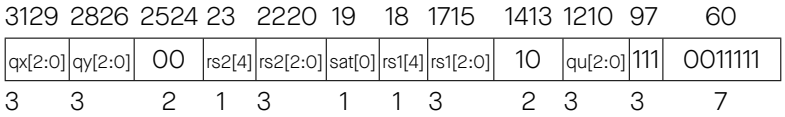


图 4.7-12. ESP.VCMULAS.S8.QACC.L.LD.XP

4.7.2.13 ESP.VMULAS.S16.QACC

语法

ESP.VMULAS.S16.QACC qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，将 8 组数据段的有符号乘法结果加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段。计算结果被饱和到一个 64 位的有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段。

操作

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

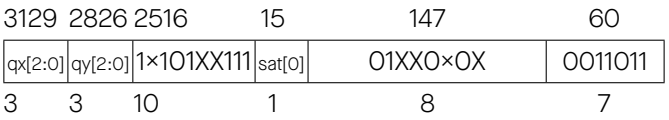


图 4.7-13. ESP.VMULAS.S16.QACC

4.7.2.14 ESP.VMULAS.S16.QACC.LD.IP

语法

ESP.VMULAS.S16.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，将 8 组段的有符号乘法结果加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段。计算结果被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 的值从-128 开始，到 112 结束，步长为 16。

操作

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

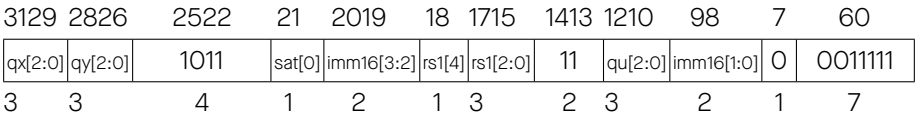


图 4.7-14. ESP.VMULAS.S16.QACC.LD.IP

4.7.2.15 ESP.VMULAS.S16.QACC.LD.XP

语法

ESP.VMULAS.S16.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，将 8 组段的有符号乘法结果加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段。计算结果被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加寄存器 rs2 中的值。

操作

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

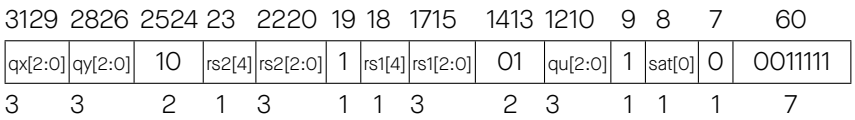


图 4.7-15. ESP.VMULAS.S16.QACC.LD.XP

4.7.2.16 ESP.VMULAS.S16.QACC.ST.IP

语法

ESP.VMULAS.S16.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，这 8 组段的有符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段中。计算结果被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

在操作过程中，寄存器 qu 的值被存储在内存中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的一个 4 位符号扩展常数，该常数被左移 4 位。

imm 的起始值为-128，结束值为 112，步长为 16。

操作

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

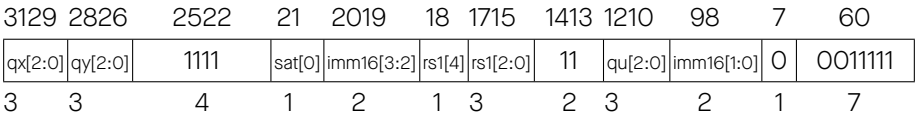


图 4.7-16. ESP.VMULAS.S16.QACC.ST.IP

4.7.2.17 ESP.VMULAS.S16.QACC.ST.XP

语法

ESP.VMULAS.S16.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，这 8 组数据段的有符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段上。计算结果被饱和到一个 64 位的有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

在操作过程中，寄存器 qu 的值被存储在内存中。访问完成后，寄存器 rs1 的值增加了寄存器 rs2 的值。

操作

```
QACC_L[63 :  0] = min(max(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], -2^63),
    2^63-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], -2^63),
    2^63-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], -2^63),
    2^63-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], -2^63),
    2^63-1)
QACC_H[63 :  0] = min(max(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], -2^63),
    2^63-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], -2^63),
    2^63-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^63),
    2^63-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^63),
    2^63-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

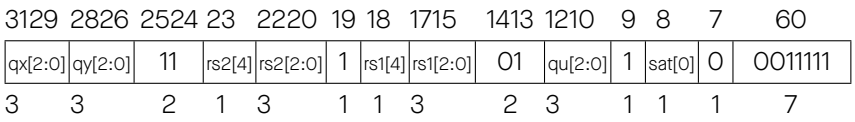


图 4.7-17. ESP.VMULAS.S16.QACC.ST.XP

4.7.2.18 ESP.VMULAS.S16.XACC

语法

ESP.VMULAS.S16.XACC qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，它对这 8 组段进行有符号的乘累加操作。累加的结果加到特殊寄存器 XACC 中的值。然后，将结果和进行饱和处理并存储在 XACC 中。

操作

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + ··· + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
XACC	3	3

格式

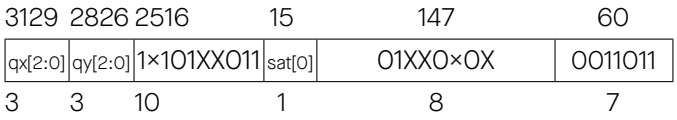


图 4.7-18. ESP.VMULAS.S16.XACC

4.7.2.19 ESP.VMULAS.S16.XACC.LD.IP

语法

ESP.VMULAS.S16.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，它对这 8 组段分别执行有符号的乘累加操作。累加的结果被添加到特殊寄存器 XACC 中的值。然后，得到的和被饱和并存储在 XACC 中。

在操作过程中，16 字节的数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的一个 4 位符号扩展常数，该常数被左移 4 位。

imm 的值从-128 开始，到 112 结束，步长为 16。

操作

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + ··· + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

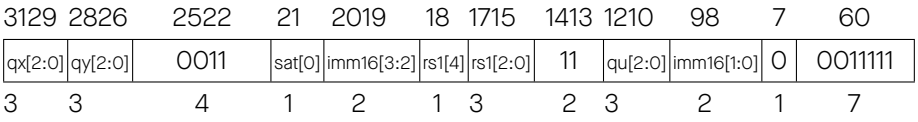


图 4.7-19. ESP.VMULAS.S16.XACC.LD.IP

4.7.2.20 ESP.VMULAS.S16.XACC.LD.XP

语法

ESP.VMULAS.S16.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割为 8 个 16 位的数据段。然后，它对每个 8 段集合执行有符号的乘累加操作。累积的结果加上特殊寄存器 XACC 中的值。然后将得到的总和饱和并存储在 XACC 中。

在操作过程中，16 字节的数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + ··· + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

格式

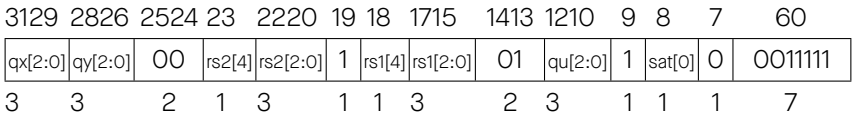


图 4.7-20. ESP.VMULAS.S16.XACC.LD.XP

4.7.2.21 ESP.VMULAS.S16.XACC.ST.IP

语法

ESP.VMULAS.S16.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，它对这 8 组段分别执行有符号的乘累加操作。累加的结果加上特殊寄存器 XACC 中的值。然后，得到的和进行饱和处理，然后存储在 XACC 中。

在操作过程中，寄存器 qu 的值被存储在内存中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的一个 4 位符号扩展常数，该常数左移 4 位。

imm 的值从-128 开始，到 112 结束，步长为 16。

操作

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + ··· + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

3129	2826	2522	21	2019	18	1715	1413	1210	98	7	60
qx[2:0]	qy[2:0]	0111	sat[0]	imm16[3:2]	rs1[4]	rs1[2:0]	11	qu[2:0]	imm16[1:0]	0	0011111
3	3	4	1	2	1	3	2	3	2	1	7

图 4.7-21. ESP.VMULAS.S16.XACC.ST.IP

4.7.2.22 ESP.VMULAS.S16.XACC.ST.XP

语法

ESP.VMULAS.S16.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割为 8 个 16 位的数据段。然后，它对每个 8 段集合执行有符号的乘累加操作。累积的结果被添加到特殊寄存器 XACC 中的值。然后，将结果总和饱和并存储在 XACC 中。

在操作过程中，寄存器 qu 的值被存储在内存中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
res[39 :  0] = qx[15 :  0] * qy[15 :  0] + qx[31 : 16] * qy[31 : 16] + ··· + qx
    [127:112] * qy[127:112]
XACC = min(max(XACC + res[39 :  0], -2^39), 2^39-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

格式

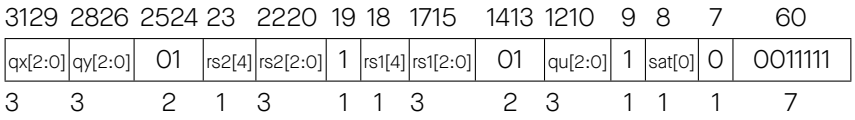


图 4.7-22. ESP.VMULAS.S16.XACC.ST.XP

4.7.2.23 ESP.VMULAS.S8.QACC

语法

ESP.VMULAS.S8.QACC qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割为 16 个每个 8 位的数据段。然后，将 16 组段的有符号乘法结果分别添加到特殊寄存器 QACC_H 和 QACC_L 中的相应 32 位数据段。计算结果被饱和到 32 位有符号数，然后存储在 QACC_H 和 QACC_L 中的相应 32 位数据段。

操作

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

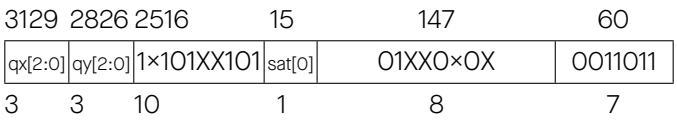


图 4.7-23. ESP.VMULAS.S8.QACC

4.7.2.24 ESP.VMULAS.S8.QACC.LD.IP

语法

ESP.VMULAS.S8.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个每个 8 位的数据段。然后，16 组段的有符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中的相应 32 位数据段。计算结果被饱和到一个 32 位有符号数，然后存储在 QACC_H 和 QACC_L 中的相应 32 位数据段。

在操作过程中，16 字节数据从内存加载到寄存器 qu。访问完成后，寄存器 rs1 中的值增加了指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 从-128 开始，到 112 结束，步长为 16。

操作

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

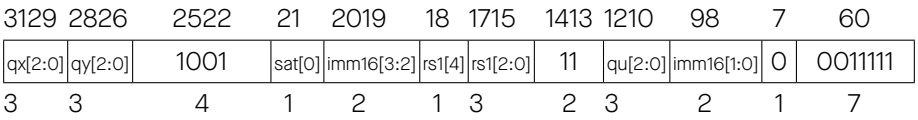


图 4.7-24. ESP.VMULAS.S8.QACC.LD.IP

4.7.2.25 ESP.VMULAS.S8.QACC.LD.XP

语法

ESP.VMULAS.S8.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割为 16 个 8 位的数据段。然后，将 16 组数据段的有符号乘法结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段中。计算结果被饱和到 32 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段中。

在操作过程中，16 字节的数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加寄存器 rs2 中的值。

操作

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

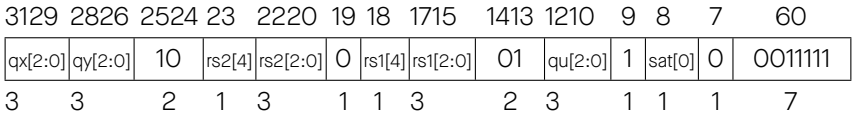


图 4.7-25. ESP.VMULAS.S8.QACC.LD.XP

4.7.2.26 ESP.VMULAS.S8.QACC.ST.IP

语法

ESP.VMULAS.S8.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，16 组段的有符号乘法结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段。计算结果被饱和到一个 32 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段。

在操作过程中，寄存器 qu 的值被存储在内存中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的一个 4 位符号扩展常数，该常数被左移 4 位。

imm 的值从-128 开始，到 112 结束，步长为 16。

操作

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

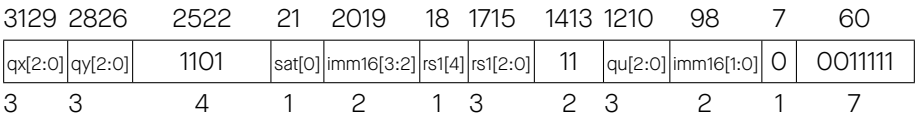


图 4.7-26. ESP.VMULAS.S8.QACC.ST.IP

4.7.2.27 ESP.VMULAS.S8.QACC.ST.XP

语法

ESP.VMULAS.S8.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，这 16 组数据段的有符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段上。计算结果被饱和到 32 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段中。

在操作过程中，寄存器 qu 的值被存储在内存中。访问完成后，寄存器 rs1 的值由寄存器 rs2 的值递增。

操作

```
QACC_L[31 :  0] = min(max(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], -2^31),
    2^31-1)
QACC_L[63 : 32] = min(max(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], -2^31),
    2^31-1)
QACC_L[95 : 64] = min(max(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], -2^31),
    2^31-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], -2^31),
    2^31-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], -2^31),
    2^31-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], -2^31),
    2^31-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], -2^31),
    2^31-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], -2^31),
    2^31-1)
QACC_H[31 :  0] = min(max(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], -2^31),
    2^31-1)
QACC_H[63 : 32] = min(max(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], -2^31),
    2^31-1)
QACC_H[95 : 64] = min(max(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], -2^31),
    2^31-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], -2^31),
    2^31-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^31),
    2^31-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^31),
    2^31-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^31),
    2^31-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^31),
    2^31-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

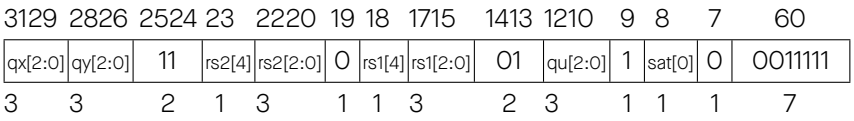


图 4.7-27. ESP.VMULAS.S8.QACC.ST.XP

4.7.2.28 ESP.VMULAS.S8.XACC

语法

ESP.VMULAS.S8.XACC qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个每个 8 位的数据段。然后，它对每个 16 个段集执行有符号的乘累加操作。累加的结果加到特殊寄存器 XACC 中的值上。然后将得到的和进行饱和处理，并存储在 XACC 中。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
XACC	3	3

格式

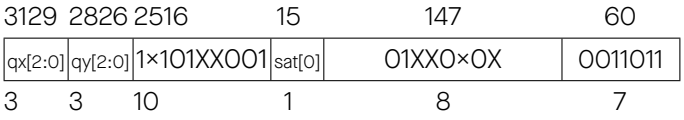


图 4.7-28. ESP.VMULAS.S8.XACC

4.7.2.29 ESP.VMULAS.S8.XACC.LD.IP

语法

ESP.VMULAS.S8.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个每个 8 位的数据段。然后，它对每个 16 段集执行有符号的乘累加操作。累积的结果被添加到特殊寄存器 XACC 中的值。然后，将结果总和饱和并存储在 XACC 中。

在操作过程中，16 字节的数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了一个从指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 从-128 开始，到 112 结束，步长为 16。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

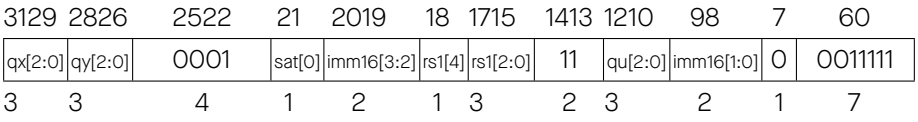


图 4.7-29. ESP.VMULAS.S8.XACC.LD.IP

4.7.2.30 ESP.VMULAS.S8.XACC.LD.XP

语法

ESP.VMULAS.S8.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，它对这 16 个段进行有符号的乘累加操作。累加的结果加上特殊寄存器 XACC 中的值。然后，将得到的和进行饱和处理，并存储在 XACC 中。

在操作过程中，16 字节的数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加寄存器 rs2 中的值。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

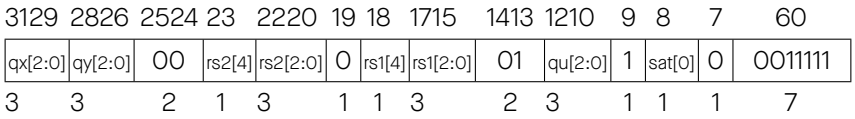


图 4.7-30. ESP.VMULAS.S8.XACC.LD.XP

4.7.2.31 ESP.VMULAS.S8.XACC.ST.IP

语法

ESP.VMULAS.S8.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，它对每个 16 个段执行有符号的乘累加操作。累加的结果被添加到特殊寄存器 XACC 中的值。然后，将结果总和饱和并存储在 XACC 中。

在操作过程中，寄存器 qu 被存储到内存中。访问完成后，寄存器 rs1 中的值增加了一个从指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 从 -128 开始，到 112 结束，步长为 16。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	1
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

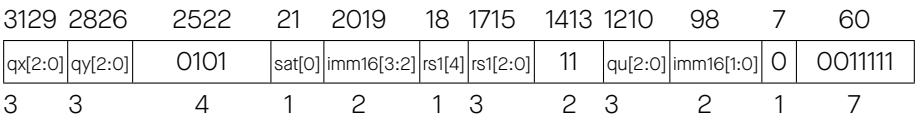


图 4.7-31. ESP.VMULAS.S8.XACC.ST.IP

4.7.2.32 ESP.VMULAS.S8.XACC.ST.XP

语法

ESP.VMULAS.S8.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

本指令将寄存器qx和qy划分为 16 个 8 位的数据段。然后，它对每个 16 个段集执行有符号的乘累加操作。累积结果加上特殊寄存器XACC中的值。然后，将得到的和进行饱和处理并存储在XACC中。

在操作过程中，寄存器qu中的值被存储到内存中。访问完成后，寄存器rs1中的值增加寄存器rs2中的值。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[39:0] = min(max(sum[40:0], -2^{39}), 2^{39}-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

格式

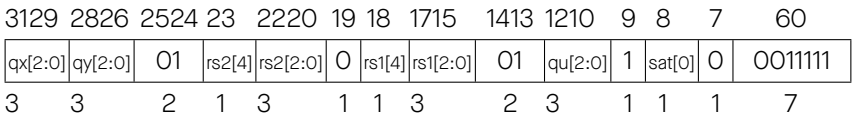


图 4.7-32. ESP.VMULAS.S8.XACC.ST.XP

4.7.2.33 ESP.VMULAS.U16.QACC

语法

ESP.VMULAS.U16.QACC qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，这 8 组数据段的无符号乘法结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段。计算结果被饱和到 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段。

操作

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

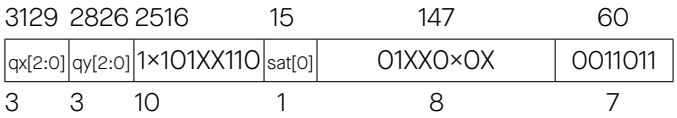


图 4.7-33. ESP.VMULAS.U16.QACC

4.7.2.34 ESP.VMULAS.U16.QACC.LD.IP

语法

ESP.VMULAS.U16.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，将这 8 组段的无符号乘法结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段中。计算结果被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

在操作过程中，16 字节的数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 的范围从-128 开始，到 112 结束，步长为 16。

操作

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

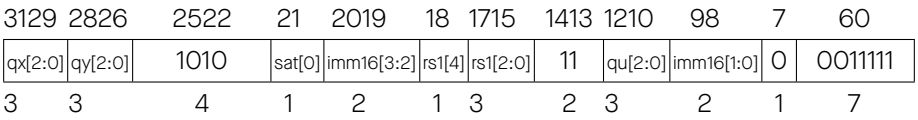


图 4.7-34. ESP.VMULAS.U16.QACC.LD.IP

4.7.2.35 ESP.VMULAS.U16.QACC.LD.XP

语法

ESP.VMULAS.U16.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分为 8 个 16 位的数据段。然后，它执行 8 组段的无符号乘法，并将结果添加到特殊寄存器 QACC_H 和 QACC_L 中的相应 64 位数据段。计算结果被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中的相应 64 位数据段。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

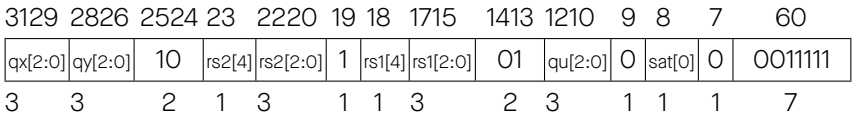


图 4.7-35. ESP.VMULAS.U16.QACC.LD.XP

4.7.2.36 ESP.VMULAS.U16.QACC.ST.IP

语法

ESP.VMULAS.U16.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，8 组段的无符号乘法结果分别添加到特殊寄存器QACC_H和QACC_L中对应的 64 位数据段中。计算结果被饱和到一个 64 位有符号数，然后存储在QACC_H和QACC_L中的相应 64 位数据段中。

在操作过程中，寄存器qu被存储到内存中。访问完成后，寄存器rs1中的值增加了指令代码段中的一个 4 位符号扩展常数，左移 4 位。

imm 从-128 开始，以 16 为步长，结束于 112。

操作

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

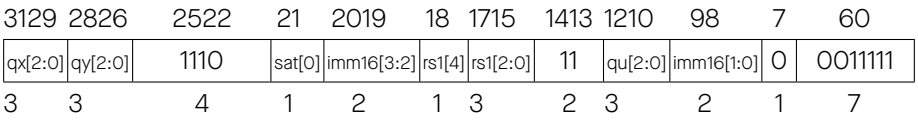


图 4.7-36. ESP.VMULAS.U16.QACC.ST.IP

4.7.2.37 ESP.VMULAS.U16.QACC.ST.XP

语法

ESP.VMULAS.U16.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，这 8 组段的无符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段。计算结果被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段。

在操作过程中，寄存器 qu 被存储到内存中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
QACC_L[63 :  0] = min(QACC_L[63 :  0] + qx[15 :  0] * qy[15 :  0], 2^63-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[31 : 16] * qy[31 : 16], 2^63-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[47 : 32] * qy[47 : 32], 2^63-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[63 : 48] * qy[63 : 48], 2^63-1)
QACC_H[63 :  0] = min(QACC_H[63 :  0] + qx[79 : 64] * qy[79 : 64], 2^63-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[95 : 80] * qy[95 : 80], 2^63-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^63-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^63-1)
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

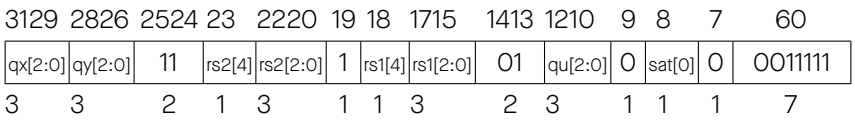


图 4.7-37. ESP.VMULAS.U16.QACC.ST.XP

4.7.2.38 ESP.VMULAS.U16.XACC

语法

ESP.VMULAS.U16.XACC qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，它对每一组的 8 个段执行无符号的乘累加操作。累加的结果被添加到特殊寄存器 XACC 中的值。然后，将结果总和进行饱和处理并存储在 XACC 中。

操作

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]

XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
XACC	3	3

格式

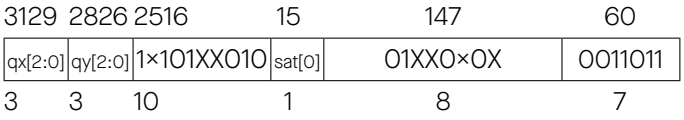


图 4.7-38. ESP.VMULAS.U16.XACC

4.7.2.39 ESP.VMULAS.U16.XACC.LD.IP

语法

ESP.VMULAS.U16.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器qx和qy分为 8 个 16 位的数据段。然后，它对每个 8 段集合执行无符号乘累加操作。累积结果加到特殊寄存器XACC中的值上。然后，将结果总和饱和并存储在XACC中。

在操作过程中，将 16 字节数据从内存加载到寄存器qu中。访问完成后，寄存器rs1中的值增加了一个从指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 从-128 开始，到 112 结束，步长为 16。

操作

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

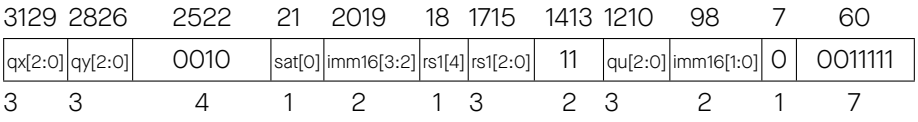


图 4.7-39. ESP.VMULAS.U16.XACC.LD.IP

4.7.2.40 ESP.VMULAS.U16.XACC.LD.XP

语法

ESP.VMULAS.U16.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割为 8 个 16 位的数据段。然后，它对每个 8 段集合执行无符号的乘累加操作。累积的结果被添加到特殊寄存器 XACC 中的值。然后，将结果总和饱和并存储在 XACC 中。

在操作过程中，它将内存中的 16 字节数据加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

格式

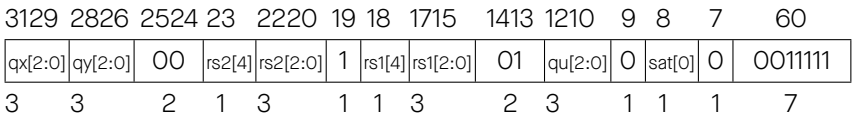


图 4.7-40. ESP.VMULAS.U16.XACC.LD.XP

4.7.2.41 ESP.VMULAS.U16.XACC.ST.IP

语法

ESP.VMULAS.U16.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，它对这 8 组数据段分别执行无符号乘累加操作。累加的结果加到特殊寄存器 XACC 中的值上。然后，得到的和被饱和并存储在 XACC 中。

在操作过程中，寄存器 qu 被存储到内存中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的一个 4 位符号扩展常数，该常数左移 4 位。

imm 的起始值为 -128，结束值为 112，步长为 16。

操作

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

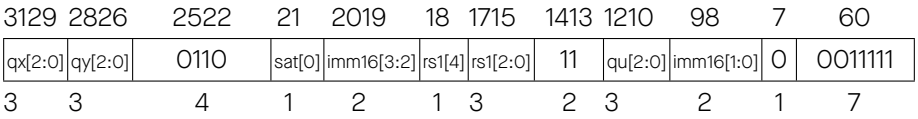


图 4.7-41. ESP.VMULAS.U16.XACC.ST.IP

4.7.2.42 ESP.VMULAS.U16.XACC.ST.XP

语法

ESP.VMULAS.U16.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，它对每个 8 段集合执行无符号乘累加操作。累积结果加上特殊寄存器 XACC 中的值。然后，将结果总和饱和并存储在 XACC 中。

在操作过程中，寄存器 qu 被存储到内存中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
add0[31:0] = qx[ 15:  0] * qy[ 15:  0]
add1[31:0] = qx[ 31: 16] * qy[ 31: 16]
...
add7[31:0] = qx[127:112] * qy[127:112]
sum[40:0] = XACC[39:0] + add0[31:0] + add1[31:0] + ... + add7[31:0]
XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

格式

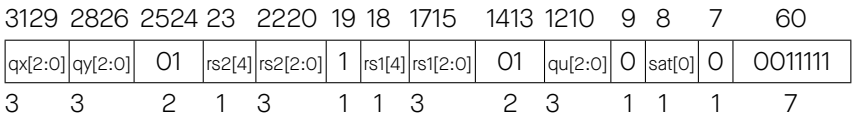


图 4.7-42. ESP.VMULAS.U16.XACC.ST.XP

4.7.2.43 ESP.VMULAS.U8.QACC

语法

ESP.VMULAS.U8.QACC qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，它对 16 组数据段进行无符号乘法，并将结果加到特殊寄存器 QACC_H 和 QACC_L 中的相应 32 位数据段。计算结果被饱和到一个 32 位无符号数，然后存储在 QACC_H 和 QACC_L 中的相应 32 位数据段。

操作

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

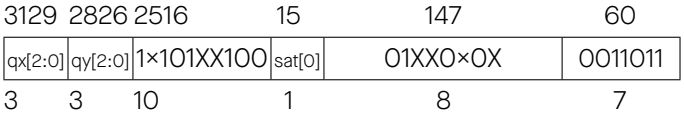


图 4.7-43. ESP.VMULAS.U8.QACC

4.7.2.44 ESP.VMULAS.U8.QACC.LD.IP

语法

ESP.VMULAS.U8.QACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个每个 8 位的数据段。然后，16 组段的无符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中的相应 32 位数据段。计算结果被饱和到一个 32 位无符号数，然后存储在 QACC_H 和 QACC_L 中的相应 32 位数据段。

在操作过程中，16 字节数据从内存加载到寄存器 qu。访问完成后，寄存器 rs1 中的值增加了指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 的值从-128 开始，到 112 结束，步长为 16。

操作

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

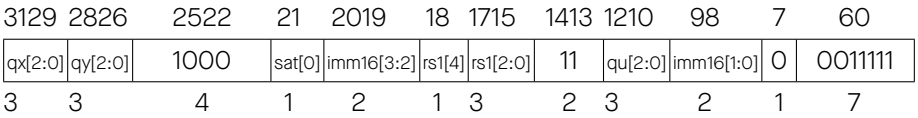


图 4.7-44. ESP.VMULAS.U8.QACC.LD.IP

4.7.2.45 ESP.VMULAS.U8.QACC.LD.XP

语法

ESP.VMULAS.U8.QACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，16 组段的无符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段。计算结果被饱和到一个 32 位的无符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段。

在操作过程中，16 字节的数据从内存加载到寄存器 qu。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

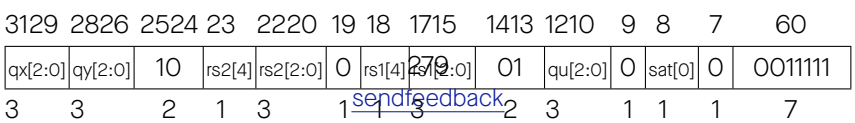


图 4.7-45. ESP.VMULAS.U8.QACC.LD.XP

4.7.2.46 ESP.VMULAS.U8.QACC.ST.IP

语法

ESP.VMULAS.U8.QACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，16 组段的无符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段。计算结果被饱和到一个 32 位无符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段。

在操作过程中，寄存器 qu 被存储到内存中。访问完成后，寄存器 rs1 中的值增加了指令代码段中的 4 位符号扩展常数，该常数被左移 4 位。

imm 的起始值为-128，结束值为 112，步长为 16。

操作

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

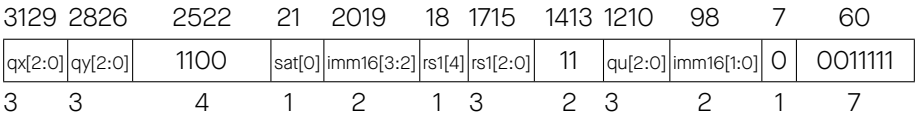


图 4.7-46. ESP.VMULAS.U8.QACC.ST.IP

4.7.2.47 ESP.VMULAS.U8.QACC.ST.XP

语法

ESP.VMULAS.U8.QACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，这 16 组数据段的无符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段。计算结果被饱和到一个 32 位的无符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段。

在操作过程中，寄存器 qu 被存储到内存中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
QACC_L[31 :  0] = min(QACC_L[31 :  0] + qx[7  :  0] * qy[7  :  0], 2^31-1)
QACC_L[63 : 32] = min(QACC_L[63 : 32] + qx[15 :  8] * qy[15 :  8], 2^31-1)
QACC_L[95 : 64] = min(QACC_L[95 : 64] + qx[23 : 16] * qy[23 : 16], 2^31-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[31 : 24] * qy[31 : 24], 2^31-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[39 : 32] * qy[39 : 32], 2^31-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[47 : 40] * qy[47 : 40], 2^31-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[55 : 48] * qy[55 : 48], 2^31-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[63 : 56] * qy[63 : 56], 2^31-1)
QACC_H[31 :  0] = min(QACC_H[31 :  0] + qx[71 : 64] * qy[71 : 64], 2^31-1)
QACC_H[63 : 32] = min(QACC_H[63 : 32] + qx[79 : 72] * qy[79 : 72], 2^31-1)
QACC_H[95 : 64] = min(QACC_H[95 : 64] + qx[87 : 80] * qy[87 : 80], 2^31-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[95 : 88] * qy[95 : 88], 2^31-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^31-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^31-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^31-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^31-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

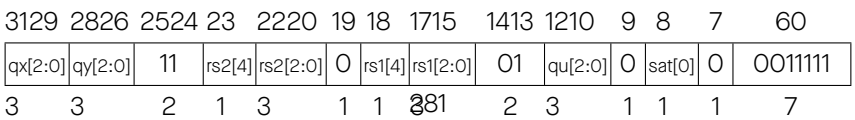


图 4.7-47. ESP.VMULAS.U8.QACC.ST.XP

4.7.2.48 ESP.VMULAS.U8.XACC

语法

ESP.VMULAS.U8.XACC qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个每个 8 位的数据段。然后，它对每个 16 组段执行无符号乘累加操作。累加的结果加到特殊寄存器 XACC 中的值。然后，将结果总和饱和并存储在 XACC 中。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]

XACC[40:0] = min(max(sum[40:0], 0), 2^{40}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
XACC	3	3

格式

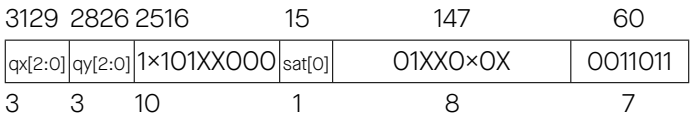


图 4.7-48. ESP.VMULAS.U8.XACC

4.7.2.49 ESP.VMULAS.U8.XACC.LD.IP

语法

ESP.VMULAS.U8.XACC.LD.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个每个 8 位的数据段。然后，它对每个 16 段集执行无符号乘累加操作。累积结果加到特殊寄存器 XACC 中的值上。然后，将结果总和饱和并存储在 XACC 中。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加了指令代码段中左移 4 位的 4 位符号扩展常数。

imm 从 -128 开始，到 112 结束，步长为 16。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

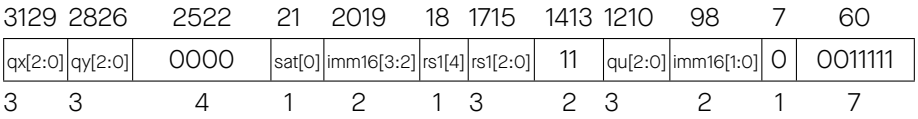


图 4.7-49. ESP.VMULAS.U8.XACC.LD.IP

4.7.2.50 ESP.VMULAS.U8.XACC.LD.XP

语法

ESP.VMULAS.U8.XACC.LD.XP qu,rs1,rs2,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分为 16 个 8 位的数据段。然后，它对每个 16 个段集执行无符号乘累加操作。累积结果加上特殊寄存器 XACC 中的值。然后将得到的和进行饱和处理，并存储在 XACC 中。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加寄存器 rs2 中的值。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

格式

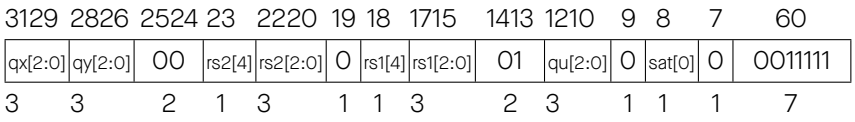


图 4.7-50. ESP.VMULAS.U8.XACC.LD.XP

4.7.2.51 ESP.VMULAS.U8.XACC.ST.IP

语法

ESP.VMULAS.U8.XACC.ST.IP qu,rs1,imm16,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，它对每一组 16 个段进行无符号乘累加操作。累加的结果加上特殊寄存器 XACC 中的值。然后，将结果和进行饱和处理并存储在 XACC 中。

在操作过程中，寄存器 qu 的值被存储到内存中。访问完成后，寄存器 rs1 中的值增加了一个从指令代码段中的 4 位符号扩展常数，该常数左移 4 位。

imm 从 -128 开始，到 112 结束，步长为 16。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
qx	1	-
qy	1	-
XACC	3	3

格式

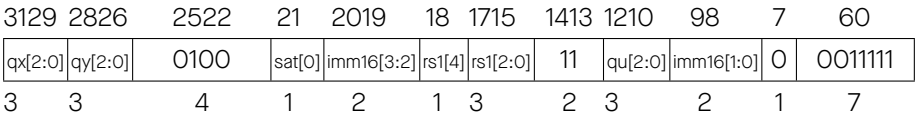


图 4.7-51. ESP.VMULAS.U8.XACC.ST.IP

4.7.2.52 ESP.VMULAS.U8.XACC.ST.XP

语法

ESP.VMULAS.U8.XACC.ST.XP qu,rs1,rs2,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割为 16 个每个 8 位的数据段。然后，它对每个 16 段执行无符号乘累加操作。累加的结果加到特殊寄存器 XACC 中的值上。然后，将结果总和饱和并存储在 XACC 中。

在操作过程中，寄存器 qu 的值被存储到内存中。访问完成后，寄存器 rs1 中的值增加了寄存器 rs2 的值。

操作

```
add0[15:0] = qx[  7:  0] * qy[  7:  0]
add1[15:0] = qx[ 15:  8] * qy[ 15:  8]
...
add15[15:0] = qx[127:120] * qy[127:120]
sum[40:0] = XACC[39:0] + add0[15:0] + add1[15:0] + ... + add15[15:0]
XACC[39:0] = min(max(sum[40:0], 0), 2^{40}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
XACC	3	3

格式

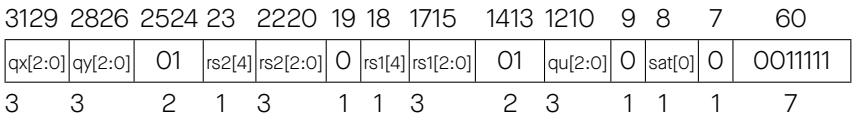


图 4.7-52. ESP.VMULAS.U8.XACC.ST.XP

4.7.2.53 ESP.VMULAS.S16.QACC.LDBC.INCP

语法

ESP.VMULAS.S16.QACC.LDBC.INCP qu,rs1,qx,qy,sat

描述

本指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，这 8 组数据段的有符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段上。计算结果被饱和到一个 64 位的有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

在操作过程中，2 字节的数据从内存中加载并广播到寄存器 qu。访问完成后，寄存器 rs1 中的值增加 2。

操作

```
QACC_L[ 63:  0] = min(max(QACC_L[ 63:  0] + qx[ 15:  0] * qy[ 15:  0], -2^{63}),
    2^{63}-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[ 31: 16] * qy[ 31: 16], -2^{63}),
    2^{63}-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[ 47: 32] * qy[ 47: 32], -2^{63}),
    2^{63}-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[ 63: 48] * qy[ 63: 48], -2^{63}),
    2^{63}-1)
QACC_H[ 63:  0] = min(max(QACC_H[ 63:  0] + qx[ 79: 64] * qy[ 79: 64], -2^{63}),
    2^{63}-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[ 95: 80] * qy[ 95: 80], -2^{63}),
    2^{63}-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * qy[111: 96], -2^{63}),
    2^{63}-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * qy[127:112], -2^{63}),
    2^{63}-1)

qu[127:0] = {8{load16(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 2
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

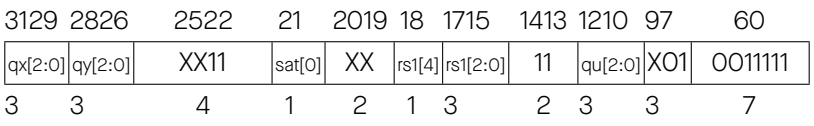


图 4.7-53. ESP.VMULAS.S16.QACC.LDBC.INCP

4.7.2.54 ESP.VMULAS.S8.QACC.LDBC.INCP

语法

ESP.VMULAS.S8.QACC.LDBC.INCP qu,rs1,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个 8 位的数据段。然后，这 16 组数据段的有符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段上。计算结果被饱和到一个 32 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段中。

在操作过程中，从内存中加载 1 字节数据并广播到寄存器 qu。访问完成后，寄存器 rs1 中的值增加 1。

操作

```
QACC_L[ 31:  0] = min(max(QACC_L[ 31:  0] + qx[  7:  0] * qy[  7:  0], -2^{31}),
    2^{31}-1)
QACC_L[ 63: 32] = min(max(QACC_L[ 63: 32] + qx[ 15:  8] * qy[ 15:  8], -2^{31}),
    2^{31}-1)
QACC_L[ 95: 64] = min(max(QACC_L[ 95: 64] + qx[ 23: 16] * qy[ 23: 16], -2^{31}),
    2^{31}-1)
QACC_L[127: 96] = min(max(QACC_L[127: 96] + qx[ 31: 24] * qy[ 31: 24], -2^{31}),
    2^{31}-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[ 39: 32] * qy[ 39: 32], -2^{31}),
    2^{31}-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[ 47: 40] * qy[ 47: 40], -2^{31}),
    2^{31}-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[ 55: 48] * qy[ 55: 48], -2^{31}),
    2^{31}-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[ 63: 56] * qy[ 63: 56], -2^{31}),
    2^{31}-1)
QACC_H[ 31:  0] = min(max(QACC_H[ 31:  0] + qx[ 71: 64] * qy[ 71: 64], -2^{31}),
    2^{31}-1)
QACC_H[ 63: 32] = min(max(QACC_H[ 63: 32] + qx[ 79: 72] * qy[ 79: 72], -2^{31}),
    2^{31}-1)
QACC_H[ 95: 64] = min(max(QACC_H[ 95: 64] + qx[ 87: 80] * qy[ 87: 80], -2^{31}),
    2^{31}-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[ 95: 88] * qy[ 95: 88], -2^{31}),
    2^{31}-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * qy[103: 96], -2^{31}),
    2^{31}-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * qy[111:104], -2^{31}),
    2^{31}-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * qy[119:112], -2^{31}),
    2^{31}-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * qy[127:120], -2^{31}),
    2^{31}-1)


qu[127:0] = {16{load8(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 1
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

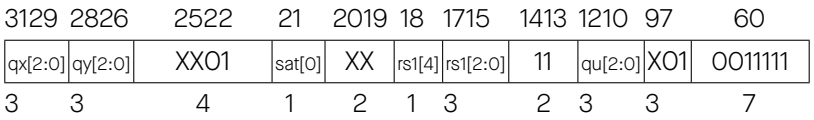


图 4.7-54. ESP.VMULAS.S8.QACC.LDBC.INCP

4.7.2.55 ESP.VMULAS.U16.QACC.LDBC.INCP

语法

ESP.VMULAS.U16.QACC.LDBC.INCP qu,rs1,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 8 个 16 位的数据段。然后，它执行 8 组段的无符号乘法，并将结果加到特殊寄存器 QACC_H 和 QACC_L 中相应的 64 位数据段。计算结果被饱和到一个 64 位无符号数，然后存储在 QACC_H 和 QACC_L 中相应的 64 位数据段。

在操作过程中，它从内存中加载 2 字节的数据，并将其广播到寄存器 qu。访问完成后，寄存器 rs1 中的值增加 2。

操作

```
QACC_L[ 63:  0] = min(QACC_L[ 63:  0] + qx[ 15:  0] * qy[ 15:  0], 2^{64}-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[ 31: 16] * qy[ 31: 16], 2^{64}-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[ 47: 32] * qy[ 47: 32], 2^{64}-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[ 63: 48] * qy[ 63: 48], 2^{64}-1)
QACC_H[ 63:  0] = min(QACC_H[ 63:  0] + qx[ 79: 64] * qy[ 79: 64], 2^{64}-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[ 95: 80] * qy[ 95: 80], 2^{64}-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * qy[111: 96], 2^{64}-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * qy[127:112], 2^{64}-1)

qu[127:0] = {8{load16(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 2
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

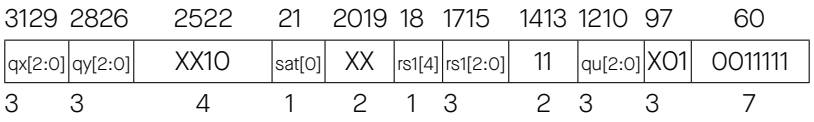


图 4.7-55. ESP.VMULAS.U16.QACC.LDBC.INCP

4.7.2.56 ESP.VMULAS.U8.QACC.LDBC.INCP

语法

ESP.VMULAS.U8.QACC.LDBC.INCP qu,rs1,qx,qy,sat

描述

此指令将寄存器 qx 和 qy 分割成 16 个每个 8 位的数据段。然后，16 组段的无符号乘法结果被加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段。计算结果被饱和到一个 32 位无符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段。

在操作过程中，从内存中加载 1 字节数据并广播到寄存器 qu。访问完成后，寄存器 rs1 中的值增加 1。

操作

```
QACC_L[ 31:  0] = min(QACC_L[ 31:  0] + qx[  7:  0] * qy[  7:  0], 2^{32}-1)
QACC_L[ 63: 32] = min(QACC_L[ 63: 32] + qx[ 15:  8] * qy[ 15:  8], 2^{32}-1)
QACC_L[ 95: 64] = min(QACC_L[ 95: 64] + qx[ 23: 16] * qy[ 23: 16], 2^{32}-1)
QACC_L[127: 96] = min(QACC_L[127: 96] + qx[ 31: 24] * qy[ 31: 24], 2^{32}-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[ 39: 32] * qy[ 39: 32], 2^{32}-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[ 47: 40] * qy[ 47: 40], 2^{32}-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[ 55: 48] * qy[ 55: 48], 2^{32}-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[ 63: 56] * qy[ 63: 56], 2^{32}-1)
QACC_H[ 31:  0] = min(QACC_H[ 31:  0] + qx[ 71: 64] * qy[ 71: 64], 2^{32}-1)
QACC_H[ 63: 32] = min(QACC_H[ 63: 32] + qx[ 79: 72] * qy[ 79: 72], 2^{32}-1)
QACC_H[ 95: 64] = min(QACC_H[ 95: 64] + qx[ 87: 80] * qy[ 87: 80], 2^{32}-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[ 95: 88] * qy[ 95: 88], 2^{32}-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * qy[103: 96], 2^{32}-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * qy[111:104], 2^{32}-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * qy[119:112], 2^{32}-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * qy[127:120], 2^{32}-1)


qu[127:0] = {16{load8(rs1[31:0])}}
rs1[31:0] = rs1[31:0] + 1
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

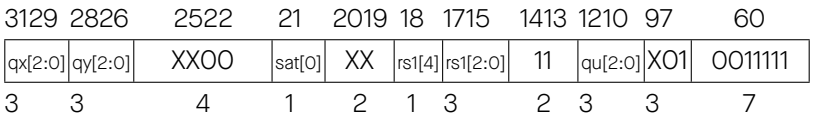


图 4.7-56. ESP.VMULAS.U8.QACC.LDBC.INCP

4.7.2.57 ESP.VSMULAS.S16.QACC

语法

ESP.VSMULAS.S16.QACC qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择八个 16 位数据段中的一个，并分别对其和寄存器 qx 中的八个 16 位数据段进行有符号乘法。得到的八个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段上。然后，每个结果都被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

操作

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(max(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[ 63:  0] = min(max(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * temp[15:0], -2^{63}), 2^{63}-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * temp[15:0], -2^{63}), 2^{63}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

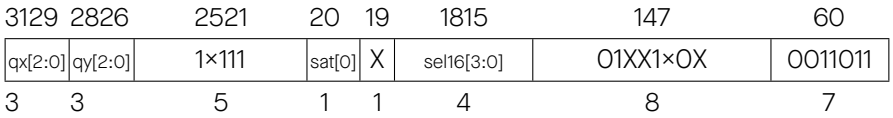


图 4.7-57. ESP.VSMULAS.S16.QACC

4.7.2.58 ESP.VSMULAS.S16.QACC.LD.INCP

语法

ESP.VSMULAS.S16.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择八个 16 位数据段中的一个，并分别对其和寄存器 qx 中的八个 16 位数据段进行有符号乘法运算。得到的八个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段上。然后，每个结果都被饱和到一个 64 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加 16。

操作

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(max(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_L[127: 64] = min(max(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_L[191:128] = min(max(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_L[255:192] = min(max(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[ 63:  0] = min(max(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[127: 64] = min(max(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[191:128] = min(max(QACC_H[191:128] + qx[111: 96] * temp[15:0], -2^{63}),
    2^{63}-1)
QACC_H[255:192] = min(max(QACC_H[255:192] + qx[127:112] * temp[15:0], -2^{63}),
    2^{63}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

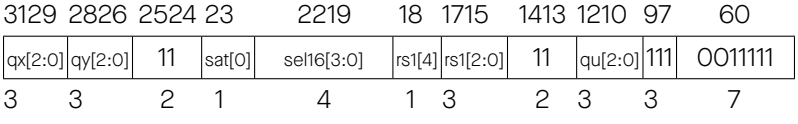


图 4.7-58. ESP.VSMULAS.S16.QACC.LD.INCP

4.7.2.59 ESP.VSMULAS.S8.QACC

语法

ESP.VSMULAS.S8.QACC qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择 16 个 8 位数据段中的一个，并分别对其和寄存器 qx 中的 16 个 8 位数据段进行有符号乘法运算。得到的 16 个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段上。然后，每个结果都被饱和到一个 32 位有符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段中。

操作

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(max(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[ 39: 32] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[ 47: 40] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[ 55: 48] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[ 63: 56] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 31:  0] = min(max(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * temp[7:0], -2^{31}), 2^{31}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-

操作数	读流水级数	写流水级数
QACC_H	2	2
QACC_L	2	2

格式

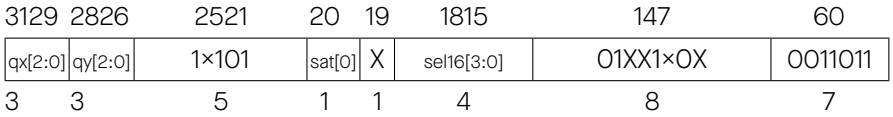


图 4.7-59. ESP.VSMULAS.S8.QACC

4.7.2.60 ESP.VSMULAS.S8.QACC.LD.INCP

语法

ESP.VSMULAS.S8.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择 16 个 8 位数据段中的一个，并分别对其和寄存器 qx 中的 16 个 8 位数据段进行有符号乘法运算。得到的 16 个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段上。然后，结果被饱和到一个 32 位有符号数，并存储在 QACC_H 和 QACC_L 中对应的 32 位数据段中。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加 16。

操作

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(max(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 63: 32] = min(max(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[ 95: 64] = min(max(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[127: 96] = min(max(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[159:128] = min(max(QACC_L[159:128] + qx[ 39: 32] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[191:160] = min(max(QACC_L[191:160] + qx[ 47: 40] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[223:192] = min(max(QACC_L[223:192] + qx[ 55: 48] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_L[255:224] = min(max(QACC_L[255:224] + qx[ 63: 56] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 31:  0] = min(max(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 63: 32] = min(max(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[ 95: 64] = min(max(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[127: 96] = min(max(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[159:128] = min(max(QACC_H[159:128] + qx[103: 96] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[191:160] = min(max(QACC_H[191:160] + qx[111:104] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[223:192] = min(max(QACC_H[223:192] + qx[119:112] * temp[7:0], -2^{31}), 2^{31}-1)
QACC_H[255:224] = min(max(QACC_H[255:224] + qx[127:120] * temp[7:0], -2^{31}), 2^{31}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

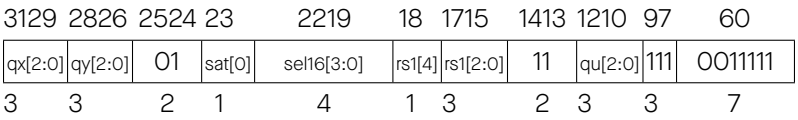


图 4.7-60. ESP.VSMULAS.S8.QACC.LD.INCP

4.7.2.61 ESP.VSMULAS.U16.QACC

语法

ESP.VSMULAS.U16.QACC qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择八个 16 位数据段中的一个，并分别对其和寄存器 qx 中的八个 16 位数据段进行无符号乘法运算。得到的八个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段上。然后，每个结果都被饱和到一个 64 位无符号数，然后存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

操作

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], 2^{64}-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], 2^{64}-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], 2^{64}-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], 2^{64}-1)
QACC_H[ 63:  0] = min(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], 2^{64}-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], 2^{64}-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * temp[15:0], 2^{64}-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * temp[15:0], 2^{64}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

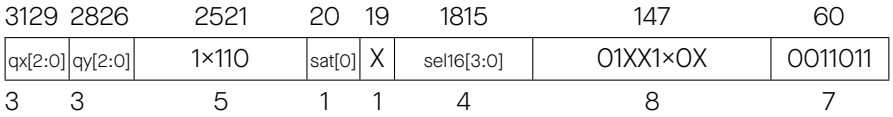


图 4.7-61. ESP.VSMULAS.U16.QACC

4.7.2.62 ESP.VSMULAS.U16.QACC.LD.INCP

语法

ESP.VSMULAS.U16.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择八个 16 位数据段中的一个，并分别对其和寄存器 qx 中的八个 16 位数据段进行无符号乘法运算。得到的 8 个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 64 位数据段上。然后，结果被饱和到一个 64 位无符号数，并存储在 QACC_H 和 QACC_L 中对应的 64 位数据段中。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加 16。

操作

```
temp[15:0] = qy[sel16*16+15:sel16*16]
QACC_L[ 63:  0] = min(QACC_L[ 63:  0] + qx[ 15:  0] * temp[15:0], 2^{64}-1)
QACC_L[127: 64] = min(QACC_L[127: 64] + qx[ 31: 16] * temp[15:0], 2^{64}-1)
QACC_L[191:128] = min(QACC_L[191:128] + qx[ 47: 32] * temp[15:0], 2^{64}-1)
QACC_L[255:192] = min(QACC_L[255:192] + qx[ 63: 48] * temp[15:0], 2^{64}-1)
QACC_H[ 63:  0] = min(QACC_H[ 63:  0] + qx[ 79: 64] * temp[15:0], 2^{64}-1)
QACC_H[127: 64] = min(QACC_H[127: 64] + qx[ 95: 80] * temp[15:0], 2^{64}-1)
QACC_H[191:128] = min(QACC_H[191:128] + qx[111: 96] * temp[15:0], 2^{64}-1)
QACC_H[255:192] = min(QACC_H[255:192] + qx[127:112] * temp[15:0], 2^{64}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

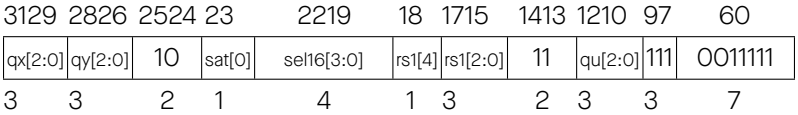


图 4.7-62. ESP.VSMULAS.U16.QACC.LD.INCP

4.7.2.63 ESP.VSMULAS.U8.QACC

语法

ESP.VSMULAS.U8.QACC qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择 16 个 8 位数据段中的一个，并分别对其和寄存器 qx 中的 16 个 8 位数据段进行无符号乘法运算。得到的 16 个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段上。然后，每个结果都被饱和到一个 32 位无符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段中。

操作

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0] , 2^{32}-1)
QACC_L[ 63: 32] = min(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0] , 2^{32}-1)
QACC_L[ 95: 64] = min(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0] , 2^{32}-1)
QACC_L[127: 96] = min(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0] , 2^{32}-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[ 39: 32] * temp[7:0] , 2^{32}-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[ 47: 40] * temp[7:0] , 2^{32}-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[ 55: 48] * temp[7:0] , 2^{32}-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[ 63: 56] * temp[7:0] , 2^{32}-1)
QACC_H[ 31:  0] = min(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0] , 2^{32}-1)
QACC_H[ 63: 32] = min(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0] , 2^{32}-1)
QACC_H[ 95: 64] = min(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0] , 2^{32}-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0] , 2^{32}-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * temp[7:0] , 2^{32}-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * temp[7:0] , 2^{32}-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * temp[7:0] , 2^{32}-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * temp[7:0] , 2^{32}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

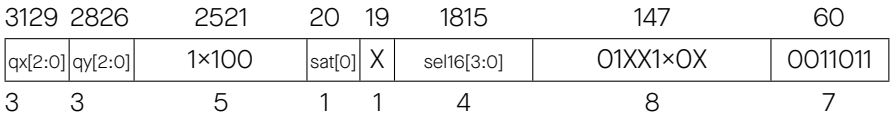


图 4.7-63. ESP.VSMULAS.U8.QACC

4.7.2.64 ESP.VSMULAS.U8.QACC.LD.INCP

语法

ESP.VSMULAS.U8.QACC.LD.INCP qu,rs1,qx,qy,sel16,sat

描述

此指令根据立即数 sel16 从寄存器 qy 中选择 16 个 8 位数据段中的一个，并分别对其和寄存器 qx 中的 16 个 8 位数据段进行无符号乘法运算。得到的 16 个结果分别加到特殊寄存器 QACC_H 和 QACC_L 中对应的 32 位数据段上。然后，每个结果都被饱和到一个 32 位无符号数，然后存储在 QACC_H 和 QACC_L 中对应的 32 位数据段中。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问完成后，寄存器 rs1 中的值增加 16。

操作

```
temp[7:0] = qy[sel16*8+7:sel16*8]
QACC_L[ 31:  0] = min(QACC_L[ 31:  0] + qx[  7:  0] * temp[7:0] , 2^{32}-1)
QACC_L[ 63: 32] = min(QACC_L[ 63: 32] + qx[ 15:  8] * temp[7:0] , 2^{32}-1)
QACC_L[ 95: 64] = min(QACC_L[ 95: 64] + qx[ 23: 16] * temp[7:0] , 2^{32}-1)
QACC_L[127: 96] = min(QACC_L[128: 96] + qx[ 31: 24] * temp[7:0] , 2^{32}-1)
QACC_L[159:128] = min(QACC_L[159:128] + qx[ 39: 32] * temp[7:0] , 2^{32}-1)
QACC_L[191:160] = min(QACC_L[191:160] + qx[ 47: 40] * temp[7:0] , 2^{32}-1)
QACC_L[223:192] = min(QACC_L[223:192] + qx[ 55: 48] * temp[7:0] , 2^{32}-1)
QACC_L[255:224] = min(QACC_L[255:224] + qx[ 63: 56] * temp[7:0] , 2^{32}-1)
QACC_H[ 31:  0] = min(QACC_H[ 31:  0] + qx[ 71: 64] * temp[7:0] , 2^{32}-1)
QACC_H[ 63: 32] = min(QACC_H[ 63: 32] + qx[ 79: 72] * temp[7:0] , 2^{32}-1)
QACC_H[ 95: 64] = min(QACC_H[ 95: 64] + qx[ 87: 80] * temp[7:0] , 2^{32}-1)
QACC_H[127: 96] = min(QACC_H[127: 96] + qx[ 95: 88] * temp[7:0] , 2^{32}-1)
QACC_H[159:128] = min(QACC_H[159:128] + qx[103: 96] * temp[7:0] , 2^{32}-1)
QACC_H[191:160] = min(QACC_H[191:160] + qx[111:104] * temp[7:0] , 2^{32}-1)
QACC_H[223:192] = min(QACC_H[223:192] + qx[119:112] * temp[7:0] , 2^{32}-1)
QACC_H[255:224] = min(QACC_H[255:224] + qx[127:120] * temp[7:0] , 2^{32}-1)

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qx	1	-
qy	1	-
QACC_H	2	2
QACC_L	2	2

格式

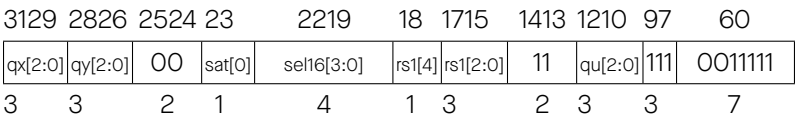


图 4.7-64. ESP.VSMULAS.U8.QACC.LD.INCP

4.7.2.65 ESP.CMUL.S16

语法

ESP.CMUL.S16 qz,qx,qy,sel4,sat,rm

描述

此指令执行 16 位有符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 32 位。复数的实部和虚部分别存储在 32 位的上 16 位和下 16 位中。计算出的实部和虚部结果存储在寄存器 qz 的相应 32 位中。

操作

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-

格式

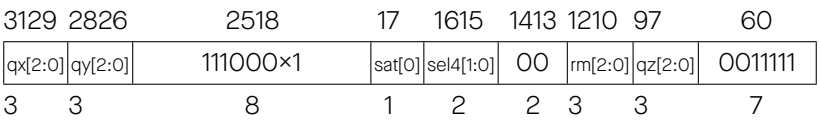


图 4.7-65. ESP.CMUL.S16

4.7.2.66 ESP.CMUL.S16.LD.INCP

语法

ESP.CMUL.S16.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 16 位有符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 32 位。复数的实部和虚部分别存储在 32 位的上 16 位和下 16 位。计算出的实部和虚部结果存储在寄存器 qz 的相应 32 位中。

同时，它还将内存中的 16 字节数据加载到寄存器 qu 中。访问后，寄存器 rs1 中的值增加 16。

操作

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

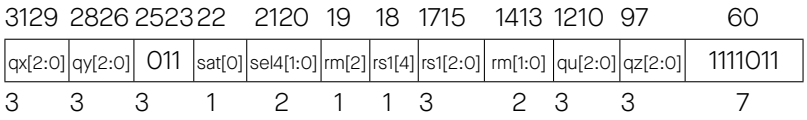


图 4.7-66. ESP.CMUL.S16.LD.INCP

4.7.2.67 ESP.CMUL.S16.ST.INCP

语法

ESP.CMUL.S16.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 16 位有符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 32 位。复数的实部和虚部分别存储在 32 位的上 16 位和下 16 位。计算出的实部和虚部结果存储在 qz 寄存器的相应 32 位中。

同时，它还将 qu 的 16 字节数据存储在内存中。访问后，rs1 寄存器中的值增加 16。

操作

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

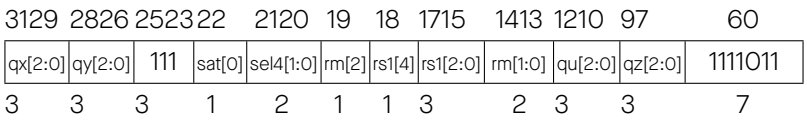


图 4.7-67. ESP.CMUL.S16.ST.INCP

4.7.2.68 ESP.CMUL.S8

语法

ESP.CMUL.S8 qz,qx,qy,sel4,sat,rm

描述

此指令执行 8 位有符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 16 位。复数的实部和虚部分别存储在 16 位中的上 8 位和下 8 位。计算出的实部和虚部结果进行右移并存储在寄存器 qz 的相应 16 位中。

操作

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5:0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5:0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5:0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5:0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5:0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5:0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5:0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5:0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5:0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5:0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5:0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5:0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5:0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5:0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5:0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5:0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5:0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5:0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5:0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5:0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5:0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5:0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5:0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5:0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5:0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5:0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5:0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5:0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5:0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-

格式

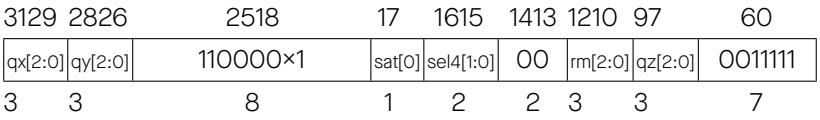


图 4.7-68. ESP.CMUL.S8

4.7.2.69 ESP.CMUL.S8.LD.INCP

语法

ESP.CMUL.S8.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 8 位有符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 16 位。复数的实部和虚部分别存储在 16 位的上 8 位和下 8 位。计算出的实部和虚部结果进行右移并存储在寄存器 qz 的相应 16 位中。

同时，它还将内存中的 16 字节数据加载到寄存器 qu 中。访问后，寄存器 rs1 中的值增加 16。

操作

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
      0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
      0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
      0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
      0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
      0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
      0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
      0]
```

```
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5 : 0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5 : 0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5 : 0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5 : 0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5 : 0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5 : 0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5 : 0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5 : 0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5 : 0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5 : 0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5 : 0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5 : 0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5 : 0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5 : 0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

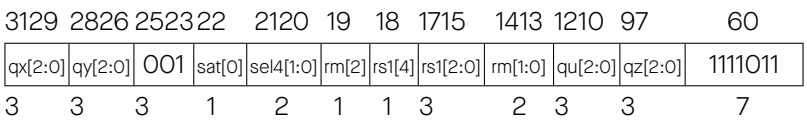


图 4.7-69. ESP.CMUL.S8.LD.INCP

4.7.2.70 ESP.CMUL.S8.ST.INCP

语法

ESP.CMUL.S8.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 8 位有符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 16 位。复数的实部和虚部分别存储在 16 位中的上 8 位和下 8 位。计算出的实部和虚部结果进行右移并存储在寄存器 qz 的相应 16 位中。

同时，它还将 qu 的 16 字节数据存储在内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
      0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
      0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
      0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
      0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
      0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
      0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
```

```
    0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
    0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
    0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
    0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
    0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
    0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
    0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
    0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
    0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
    0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
    0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
    0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
    0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
    0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
    0]
  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

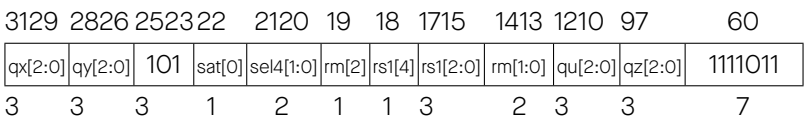


图 4.7-70. ESP.CMUL.S8.ST.INCP

4.7.2.71 ESP.CMUL.U16

语法

ESP.CMUL.U16 qz,qx,qy,sel4,sat,rm

描述

此指令执行 16 位无符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 32 位。复数的实部和虚部分别存储在 32 位中的上 16 位和下 16 位。计算出的实部和虚部结果进行右移并存储在寄存器 qz 的相应 32 位中。

操作

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-

格式

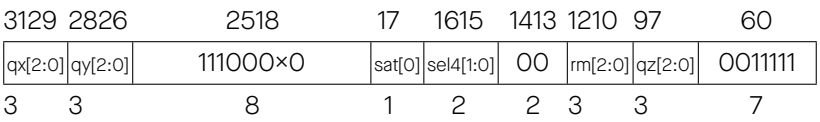


图 4.7-71. ESP.CMUL.U16

4.7.2.72 ESP.CMUL.U16.LD.INCP

语法

ESP.CMUL.U16.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 16 位无符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 32 位。复数的实部和虚部分别存储在 32 位的上 16 位和下 16 位中。计算出的实部和虚部结果存储进行右移并在寄存器 qz 的相应 32 位中。

同时，它还将内存中的 16 字节数据加载到寄存器 qu 中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

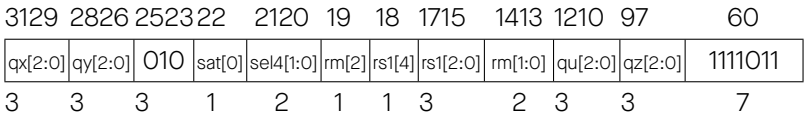


图 4.7-72. ESP.CMUL.U16.LD.INCP

4.7.2.73 ESP.CMUL.U16.ST.INCP

语法

ESP.CMUL.U16.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 16 位无符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 32 位。复数的实部和虚部分别存储在 32 位的上 16 位和下 16 位。计算出的实部和虚部结果存储进行右移并在寄存器 qz 的相应 32 位中。

同时，它还将 qu 的 16 字节数据存储在内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
if sel4 == 0:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] - qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] + qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] - qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] + qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 1:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] - qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] + qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] - qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] + qx[127:112] * qy[111: 96]) >> SAR[5:0]
if sel4 == 2:
  qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0] + qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qx[ 15:  0] * qy[ 31: 16] - qx[ 31: 16] * qy[ 15:  0]) >> SAR[5:0]
  qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32] + qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qx[ 47: 32] * qy[ 63: 48] - qx[ 63: 48] * qy[ 47: 32]) >> SAR[5:0]
if sel4 == 3:
  qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64] + qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qx[ 79: 64] * qy[ 95: 80] - qx[ 95: 80] * qy[ 79: 64]) >> SAR[5:0]
  qz[111: 96] = (qx[111: 96] * qy[111: 96] + qx[127:112] * qy[127:112]) >> SAR[5:0]
  qz[127:112] = (qx[111: 96] * qy[127:112] - qx[127:112] * qy[111: 96]) >> SAR[5:0]

  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

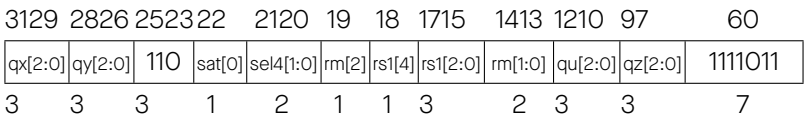


图 4.7-73. ESP.CMUL.U16.ST.INCP

4.7.2.74 ESP.CMUL.U8

语法

ESP.CMUL.U8 qz,qx,qy,sel4,sat,rm

描述

此指令执行 8 位无符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 16 位。复数的实部和虚部分别存储在 16 位中的上 8 位和下 8 位。计算出的实部和虚部结果存储进行右移并在寄存器 qz 的相应 16 位中。

操作

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
     0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
     0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
     0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
     0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
     0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
     0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
     0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
     0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
     0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
     0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
     0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
     0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
     0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
     0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
     0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
     0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
     0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
     0]
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
```

```
    0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
    0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
    0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
    0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
    0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
    0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
    0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
    0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
    0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
    0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
    0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
    0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
    0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
    0]
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-

格式

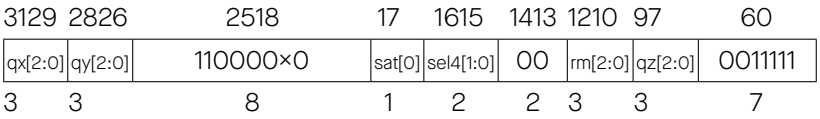


图 4.7-74. ESP.CMUL.U8

4.7.2.75 ESP.CMUL.U8.LD.INCP

语法

ESP.CMUL.U8.LD.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 8 位无符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 16 位。复数的实部和虚部分别存储在 16 位的上 8 位和下 8 位。计算出的实部和虚部结果进行右移并存储在寄存器 qz 的相应 16 位中。

同时，它还将内存中的 16 字节数据加载到寄存器 qu 中。访问后，寄存器 rs1 中的值增加 16。

操作

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
      0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
      0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
      0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
      0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
      0]
else if sel4 == 2:
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
      0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
      0]
```

```
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
      0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
      0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
      0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

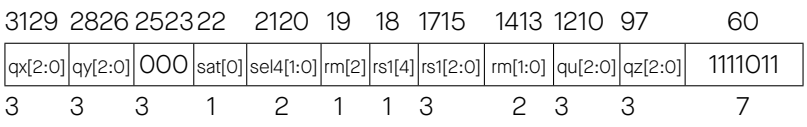


图 4.7-75. ESP.CMUL.U8.LD.INCP

4.7.2.76 ESP.CMUL.U8.ST.INCP

语法

ESP.CMUL.U8.ST.INCP qu,rs1,qz,qx,qy,sel4,sat,rm

描述

此指令执行 8 位无符号复数乘法。立即数 sel4 的范围是 0 ~ 3，它指定了两个 QR 寄存器 qx 和 qy 中用于复数乘法的 16 位。复数的实部和虚部分别存储在 16 位中的上 8 位和下 8 位。计算出的实部和虚部结果进行右移并存储在寄存器 qz 的相应 16 位中。

同时，它还将 qu 的 16 字节数据存储在内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
if sel4 == 0:
  qz[  7:  0] = (qx[  7:  0] * qy[  7:  0] - qx[ 15:  8] * qy[ 15:  8]) >> SAR[5  :
      0]
  qz[ 15:  8] = (qx[  7:  0] * qy[ 15:  8] + qx[ 15:  8] * qy[  7:  0]) >> SAR[5  :
      0]
  qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16] - qx[ 31: 16] * qy[ 31: 24]) >> SAR[5  :
      0]
  qz[ 31: 24] = (qx[ 23: 16] * qy[ 31: 24] + qx[ 31: 16] * qy[ 23: 16]) >> SAR[5  :
      0]
  qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32] - qx[ 47: 40] * qy[ 47: 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] + qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] - qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] + qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 1:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] - qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] + qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] - qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] + qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] - qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] + qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] - qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] + qx[127:120] * qy[119:112]) >> SAR[5  :
      0]
else if sel4 == 2
  qz[7  :  0] = (qx[7  :  0] * qy[7  :  0] + qx[15 :  8] * qy[15 :  8]) >> SAR[5  :
      0]
  qz[15 :  8] = (qx[7  :  0] * qy[15 :  8] - qx[15 :  8] * qy[7  :  0]) >> SAR[5  :
      0]
```

```
  qz[23 : 16] = (qx[23 : 16] * qy[23 : 16] + qx[31 : 24] * qy[31 : 24]) >> SAR[5  :
      0]
  qz[31 : 24] = (qx[23 : 16] * qy[31 : 24] - qx[31 : 24] * qy[23 : 16]) >> SAR[5  :
      0]
  qz[39 : 32] = (qx[39 : 32] * qy[39 : 32] + qx[47 : 40] * qy[47 : 40]) >> SAR[5  :
      0]
  qz[47 : 40] = (qx[39 : 32] * qy[47 : 40] - qx[47 : 40] * qy[39 : 32]) >> SAR[5  :
      0]
  qz[55 : 48] = (qx[55 : 48] * qy[55 : 48] + qx[63 : 56] * qy[63 : 56]) >> SAR[5  :
      0]
  qz[63 : 56] = (qx[55 : 48] * qy[63 : 56] - qx[63 : 56] * qy[55 : 48]) >> SAR[5  :
      0]
else if sel4 == 3:
  qz[71 : 64] = (qx[71 : 64] * qy[71 : 64] + qx[79 : 72] * qy[79 : 72]) >> SAR[5  :
      0]
  qz[79 : 72] = (qx[71 : 64] * qy[79 : 72] - qx[79 : 72] * qy[71 : 64]) >> SAR[5  :
      0]
  qz[87 : 80] = (qx[87 : 80] * qy[87 : 80] + qx[95 : 88] * qy[95 : 88]) >> SAR[5  :
      0]
  qz[95 : 88] = (qx[87 : 80] * qy[95 : 88] - qx[95 : 88] * qy[87 : 80]) >> SAR[5  :
      0]
  qz[103: 96] = (qx[103: 96] * qy[103: 96] + qx[111:104] * qy[111:104]) >> SAR[5  :
      0]
  qz[111:104] = (qx[103: 96] * qy[111:104] - qx[111:104] * qy[103: 96]) >> SAR[5  :
      0]
  qz[119:112] = (qx[119:112] * qy[119:112] + qx[127:120] * qy[127:120]) >> SAR[5  :
      0]
  qz[127:120] = (qx[119:112] * qy[127:120] - qx[127:120] * qy[119:112]) >> SAR[5  :
      0]

  qu[127:0] => store128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	2
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

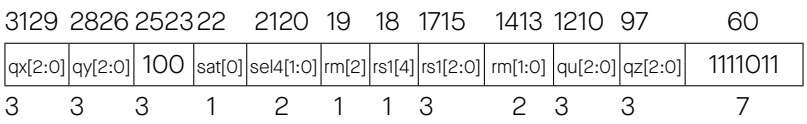


图 4.7-76. ESP.CMUL.U8.ST.INCP

4.7.2.77 ESP.MAX.S16.A

语法

ESP.MAX.S16.A qw,rd

描述

此指令将qw的 16 位元素的最大有符号值保存到rd中。

操作

```
    max_value[15 :  0] = max(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
    rd[31 :  0] = {16{max_value[15]}, max_value[15 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

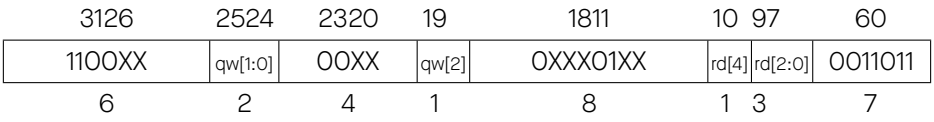


图 4.7-77. ESP.MAX.S16.A

4.7.2.78 ESP.MAX.S32.A

语法

ESP.MAX.S32.A qw,rd

描述

此指令将qw的 4 个 32 位元素的最大有符号值保存到rd。

操作

```
max_value[31 :  0] = max(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
rd[31 :  0] = max_value[31 :  0]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

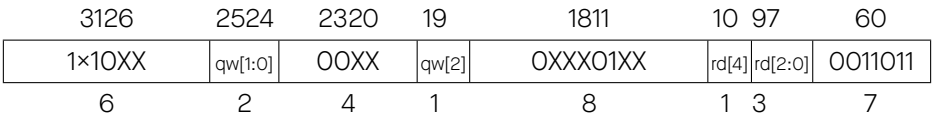


图 4.7-78. ESP.MAX.S32.A

4.7.2.79 ESP.MAX.S8.A

语法

ESP.MAX.S8.A qw,rd

描述

此指令将qw中 16 个 8 位元素的最大有符号值保存到rd。

操作

```
    max_value[7 :  0] = max(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
    rd[31 :  0] = {24{max_value[7]}, max_value[7 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

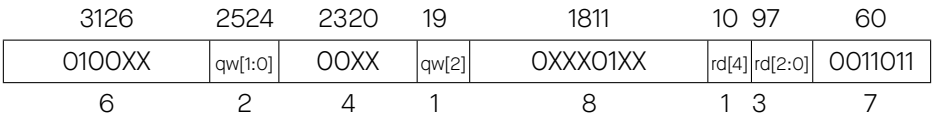


图 4.7-79. ESP.MAX.S8.A

4.7.2.80 ESP.MAX.U16.A

语法

ESP.MAX.U16.A qw,rd

描述

此指令将qw中的 16 位元素的最大无符号值保存到rd中。

操作

```
    max_value[15 :  0] = max(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
    rd[31 :  0] = {16'b0, max_value[15 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

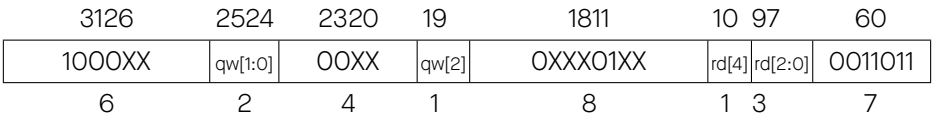


图 4.7-80. ESP.MAX.U16.A

4.7.2.81 ESP.MAX.U32.A

语法

ESP.MAX.U32.A qw,rd

描述

此指令将qw的 4 个 32 位元素的最大无符号值保存到rd。

操作

```
max_value[31 :  0] = max(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
rd[31 :  0] = max_value[31 :  0]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

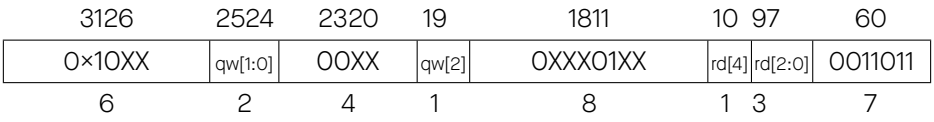


图 4.7-81. ESP.MAX.U32.A

4.7.2.82 ESP.MAX.U8.A

语法

ESP.MAX.U8.A qw,rd

描述

此指令将qw的 16 个 8 位元素的最大无符号值保存到rd。

操作

```
    max_value[7 :  0] = max(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
    rd[31 :  0] = {24{0}, max_value[7 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

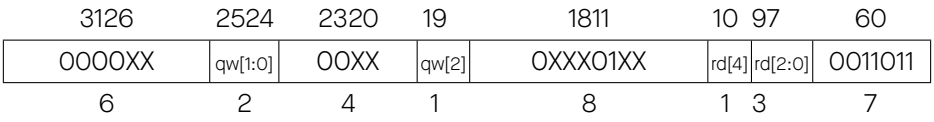


图 4.7-82. ESP.MAX.U8.A

4.7.2.83 ESP.MIN.S16.A

语法

ESP.MIN.S16.A qw,rd

描述

此指令将qw中的 16 位元素的最小有符号值保存到rd中。

操作

```
    min_value[15 :  0] = min(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
    rd[31 :  0] = {16{min_value[15]}, min_value[15 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

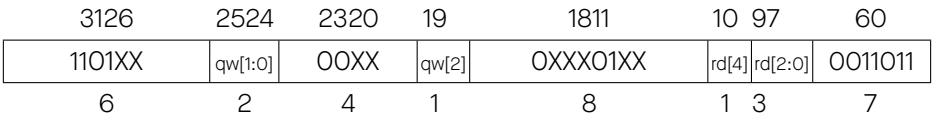


图 4.7-83. ESP.MIN.S16.A

4.7.2.84 ESP.MIN.S32.A

语法

ESP.MIN.S32.A qw,rd

描述

此指令将qw的 32 位中 4 个元素的最小有符号值保存到rd。

操作

```
min_value[31 :  0] = min(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
rd[31 :  0] = min_value[31 :  0]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

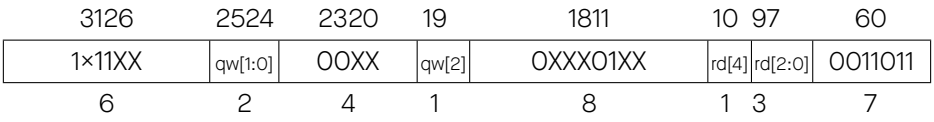


图 4.7-84. ESP.MIN.S32.A

4.7.2.85 ESP.MIN.S8.A

语法

ESP.MIN.S8.A qw,rd

描述

此指令将qw的 16 个 8 位元素的最小有符号值保存到rd。

操作

```
    min_value[7 :  0] = min(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
    rd[31 :  0] = {24{min_value[7]}, min_value[7 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

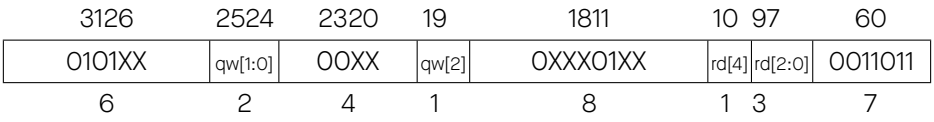


图 4.7-85. ESP.MIN.S8.A

4.7.2.86 ESP.MIN.U16.A

语法

ESP.MIN.U16.A qw,rd

描述

此指令将qw的 16 位元素的最小无符号值保存到rd中。

操作

```
    min_value[15 :  0] = min(qw[15 :  0], qw[31 : 16], ..., qw[127:112])
    rd[31 :  0] = {16'b0, min_value[15 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

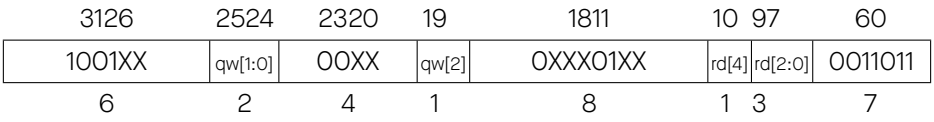


图 4.7-86. ESP.MIN.U16.A

4.7.2.87 ESP.MIN.U32.A

语法

ESP.MIN.U32.A qw,rd

描述

此指令将qw的 4 个 32 位元素的最小无符号值保存到rd。

操作

```
    min_value[31 :  0] = min(qw[31 :  0], qw[63 : 32], ..., qw[127:96])
    rd[31 :  0] = min_value[31 :  0]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

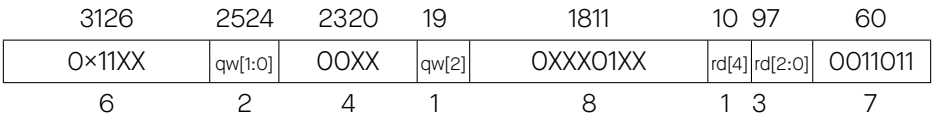


图 4.7-87. ESP.MIN.U32.A

4.7.2.88 ESP.MIN.U8.A

语法

ESP.MIN.U8.A qw,rd

描述

此指令将qw的 16 个 8 位元素的最小无符号值保存到rd。

操作

```
    min_value[7 :  0] = min(qw[7 :  0], qw[16 : 8], ..., qw[127:120])
    rd[31 :  0] = {24{0}, min_value[7 :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qw	1	-

格式

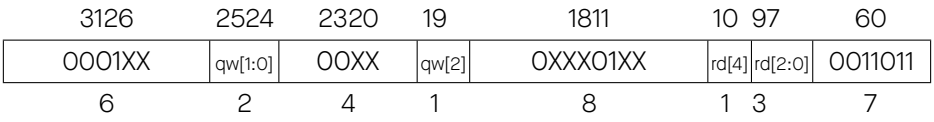


图 4.7-88. ESP.MIN.U8.A

4.7.2.89 ESP.VABS.16

语法

ESP.VABS.16 qv,qy

描述

此指令取 8 个 16 位数据段的绝对值。

操作

```
qv[ 15:  0] = (qy[ 7: 0] < 0) ? -qy[ 7: 0] : qy[ 7: 0]
qv[ 31: 16] = (qy[ 15: 16] < 0) ? -qy[ 15: 8] : qy[ 15: 8]
qv[ 47: 32] = (qy[ 47: 32] < 0) ? -qy[ 47: 16] : qy[ 47: 16]
qv[ 63: 48] = (qy[ 63: 48] < 0) ? -qy[ 63: 48] : qy[ 63: 48]
qv[ 79: 64] = (qy[ 79: 64] < 0) ? -qy[ 79: 64] : qy[ 79: 64]
qv[ 95: 80] = (qy[ 95: 80] < 0) ? -qy[ 95: 80] : qy[ 95: 80]
qv[111: 96] = (qy[111: 96] < 0) ? -qy[111: 96] : qy[111: 96]
qv[127:112] = (qy[127:112] < 0) ? -qy[127:112] : qy[127:112]
```

调度

操作数	读流水级数	写流水级数
qy	1	-
qv	-	1

格式

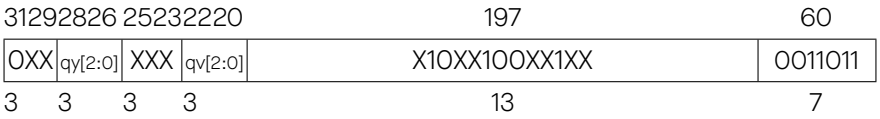


图 4.7-89. ESP.VABS.16

4.7.2.90 ESP.VABS.32

语法

ESP.VABS.32 qv,qy

描述

此指令取 4 个 32 位数据段的绝对值。

操作

```
qv[ 31:  0] = (qy[ 31: 0] < 0) ? -qy[ 31: 0] : qy[ 31: 0]
qv[ 63: 32] = (qy[ 63: 32] < 0) ? -qy[ 63: 32] : qy[ 63: 32]
qv[ 95: 64] = (qy[ 95: 64] < 0) ? -qy[ 95: 64] : qy[ 95: 64]
qv[127: 96] = (qy[127: 96] < 0) ? -qy[127: 96] : qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
qy	1	-
qv	-	1

格式

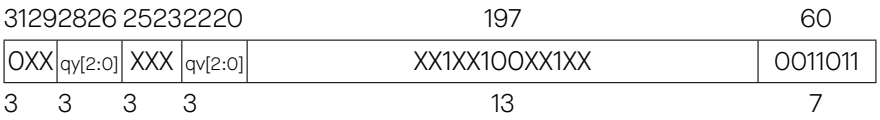


图 4.7-90. ESP.VABS.32

4.7.2.91 ESP.VABS.8

语法

ESP.VABS.8 qv,qy

描述

此指令取 16 个 8 位数据段的绝对值。

操作

```
qv[  7:  0] = (qy[ 7: 0] < 0) ? -qy[ 7: 0] : qy[ 7: 0]
qv[ 15:  8] = (qy[ 15: 8] < 0) ? -qy[ 15: 8] : qy[ 15: 8]
qv[ 23: 16] = (qy[ 23: 16] < 0) ? -qy[ 23: 16] : qy[ 23: 16]
qv[ 31: 24] = (qy[ 31: 24] < 0) ? -qy[ 31: 24] : qy[ 31: 24]
qv[ 39: 32] = (qy[ 39: 32] < 0) ? -qy[ 39: 32] : qy[ 39: 32]
qv[ 47: 40] = (qy[ 47: 40] < 0) ? -qy[ 47: 40] : qy[ 47: 40]
qv[ 55: 48] = (qy[ 55: 48] < 0) ? -qy[ 55: 48] : qy[ 55: 48]
qv[ 63: 56] = (qy[ 63: 56] < 0) ? -qy[ 63: 56] : qy[ 63: 56]
qv[ 71: 64] = (qy[ 71: 64] < 0) ? -qy[ 71: 64] : qy[ 71: 64]
qv[ 79: 72] = (qy[ 79: 72] < 0) ? -qy[ 79: 72] : qy[ 79: 72]
qv[ 87: 80] = (qy[ 87: 80] < 0) ? -qy[ 87: 80] : qy[ 87: 80]
qv[ 95: 88] = (qy[ 95: 88] < 0) ? -qy[ 95: 88] : qy[ 95: 88]
qv[103: 96] = (qy[103: 96] < 0) ? -qy[103: 96] : qy[103: 96]
qv[111:104] = (qy[111:104] < 0) ? -qy[111:104] : qy[111:104]
qv[119:112] = (qy[119:112] < 0) ? -qy[119:112] : qy[119:112]
qv[127:120] = (qy[127:120] < 0) ? -qy[127:120] : qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qy	1	-
qv	-	1

格式

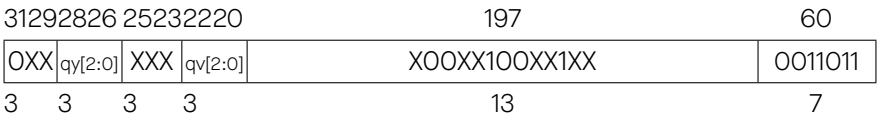


图 4.7-91. ESP.VABS.8

4.7.2.92 ESP.VADD.S16

语法

ESP.VADD.S16 qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 16 位数据上执行矢量有符号加法。然后，对计算得到的 8 个结果进行饱和，将饱和结果写入寄存器 qv。

操作

```
qv[ 15:  0] = min(max(qx[ 15:  0] + qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] + qy[ 31: 16], -2^{15}), 2^{15}-1)
...
qv[127:112] = min(max(qx[127:112] + qy[127:112], -2^{15}), 2^{15}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1

格式

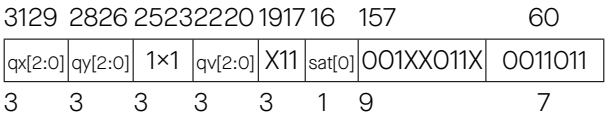


图 4.7-92. ESP.VADD.S16

4.7.2.93 ESP.VADD.S16.LD.INCP

语法

ESP.VADD.S16.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 16 位数据上执行矢量有符号加法。然后，从计算中得到的 8 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 15:  0] = min(max(qx[ 15:  0] + qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] + qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

格式

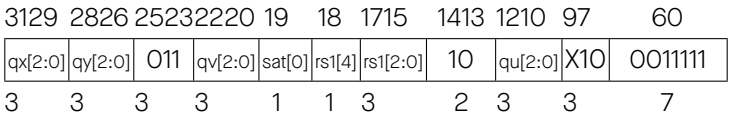


图 4.7-93. ESP.VADD.S16.LD.INCP

4.7.2.94 ESP.VADD.S16.ST.INCP

语法

ESP.VADD.S16.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 16 位数据上执行矢量有符号加法。然后，从计算中得到的 8 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，寄存器 qu 中的值存储在内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 15:  0] = min(max(qx[ 15:  0] + qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] + qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

格式

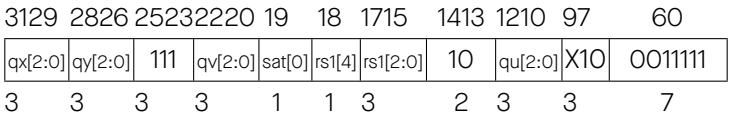


图 4.7-94. ESP.VADD.S16.ST.INCP

4.7.2.95 ESP.VADD.S32

语法

ESP.VADD.S32 qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 32 位数据上执行矢量有符号加法。然后，对从计算中得到的 4 个结果进行饱和，将饱和结果写入寄存器 qv。

操作

```
qv[ 31:  0] = min(max(qx[ 31:  0] + qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] + qy[ 63: 32], -2^{31}), 2^{31}-1)
...
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1

格式

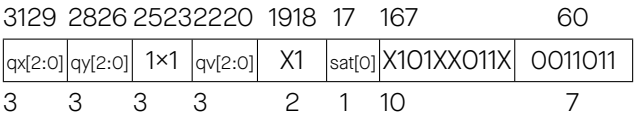


图 4.7-95. ESP.VADD.S32

4.7.2.96 ESP.VADD.S32.LD.INCP

语法

ESP.VADD.S32.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 32 位数据上执行矢量有符号加法。然后，从计算中得到的 4 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 31:  0] = min(max(qx[ 31:  0] + qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] + qy[ 63: 32], -2^{31}), 2^{31}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

格式

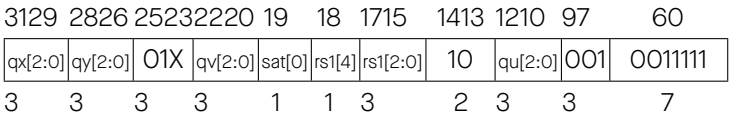


图 4.7-96. ESP.VADD.S32.LD.INCP

4.7.2.97 ESP.VADD.S32.ST.INCP

语法

ESP.VADD.S32.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 32 位数据上执行矢量有符号加法。然后，从计算中得到的 4 个结果进行饱和，饱和结果写入寄存器 qu。

在操作过程中，寄存器 qu 中的值存储在内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 31:  0] = min(max(qx[ 31:  0] + qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] + qy[ 63: 32], -2^{31}), 2^{31}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

格式

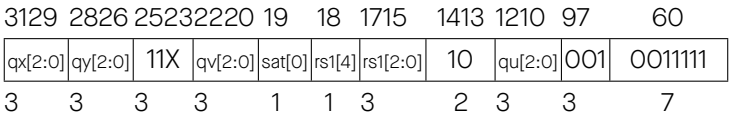


图 4.7-97. ESP.VADD.S32.ST.INCP

4.7.2.98 ESP.VADD.S8

语法

ESP.VADD.S8 qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 8 位数据上执行矢量有符号加法。然后，对计算得到的 16 个结果进行饱和，将饱和结果写入寄存器 qv。

操作

```
qv[  7:  0] = min(max(qx[  7:  0] + qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] + qy[ 15:  8], -2^{7}), 2^{7}-1)
...
qv[127:120] = min(max(qx[127:120] + qy[127:120], -2^{7}), 2^{7}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1

格式

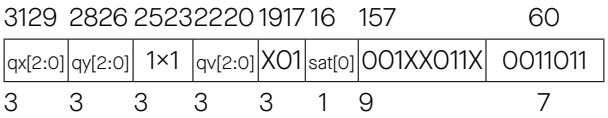


图 4.7-98. ESP.VADD.S8

4.7.2.99 ESP.VADD.S8.LD.INCP

语法

ESP.VADD.S8.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中对 8 位数据执行向量有符号加法。然后，从计算中得到的 16 个结果进行饱和，饱和结果写入寄存器 qa。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[  7:  0] = min(max(qx[  7:  0] + qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] + qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

格式

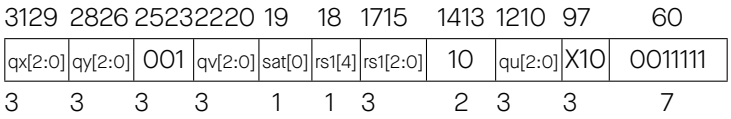


图 4.7-99. ESP.VADD.S8.LD.INCP

4.7.2.100 ESP.VADD.S8.ST.INCP

语法

ESP.VADD.S8.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 8 位数据上执行矢量有符号加法。然后，从计算中得到的 16 个结果进行饱和，饱和结果写入寄存器 qa。

在操作过程中，寄存器 qu 中的值被存储在内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[  7:  0] = min(max(qx[  7:  0] + qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] + qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

格式

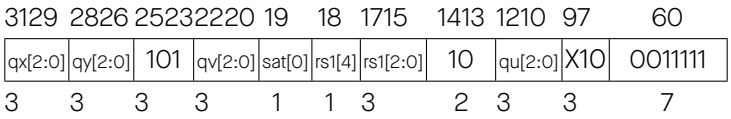


图 4.7-100. ESP.VADD.S8.ST.INCP

4.7.2.101 ESP.VADD.U16

语法

ESP.VADD.U16 qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 16 位数据上执行矢量无符号加法。然后，对从计算中得到的 8 个结果进行饱和，将饱和结果写入寄存器 qv。

操作

```
qv[ 15:  0] = min(qx[ 15:  0] + qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] + qy[ 31: 16], 2^{16}-1)
...
qv[127:112] = min(qx[127:112] + qy[127:112], 2^{16}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1

格式

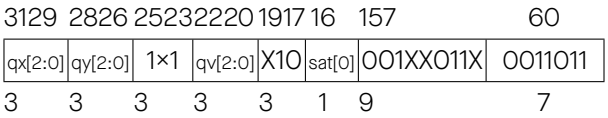


图 4.7-101. ESP.VADD.U16

4.7.2.102 ESP.VADD.U16.LD.INCP

语法

ESP.VADD.U16.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 16 位数据上执行矢量无符号加法。然后，从计算中得到的 8 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，寄存器 qu 中的值存储在内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 15:  0] = min(qx[ 15:  0] + qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] + qy[ 31: 16], 2^{16}-1)
...
qv[127:112] = min(qx[127:112] + qy[127:112], 2^{16}-1)

 qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

格式

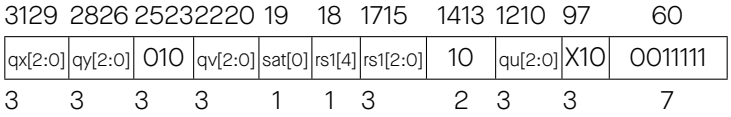


图 4.7-102. ESP.VADD.U16.LD.INCP

4.7.2.103 ESP.VADD.U16.ST.INCP

语法

ESP.VADD.U16.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 16 位数据上执行矢量无符号加法。然后，从计算中得到的 8 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 15:  0] = min(qx[ 15:  0] + qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] + qy[ 31: 16], 2^{16}-1)
...
qv[127:112] = min(qx[127:112] + qy[127:112], 2^{16}-1)

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

格式

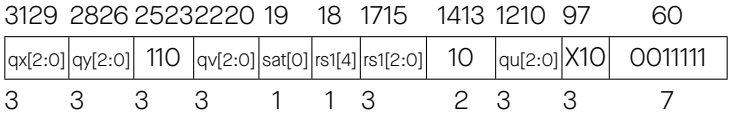


图 4.7-103. ESP.VADD.U16.ST.INCP

4.7.2.104 ESP.VADD.U32

语法

ESP.VADD.U32 qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中对 32 位数据执行矢量无符号加法。然后，对从计算中得到的 4 个结果进行饱和，将饱和结果写入寄存器 qv。

操作

```
qv[ 31:  0] = min(qx[ 31:  0] + qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] + qy[ 63: 32], 2^{32}-1)
...
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1

格式

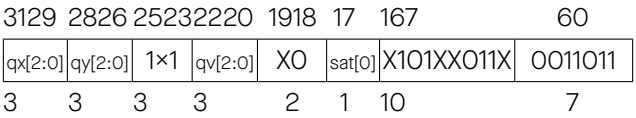


图 4.7-104. ESP.VADD.U32

4.7.2.105 ESP.VADD.U32.LD.INCP

语法

ESP.VADD.U32.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 32 位数据上执行矢量无符号加法。然后，从计算中得到的 4 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 31:  0] = min(qx[ 31:  0] + qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] + qy[ 63: 32], 2^{32}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

格式

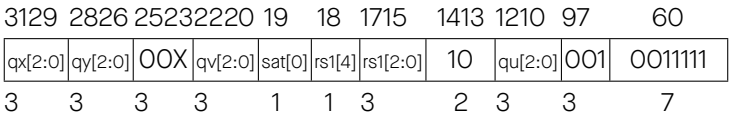


图 4.7-105. ESP.VADD.U32.LD.INCP

4.7.2.106 ESP.VADD.U32.ST.INCP

语法

ESP.VADD.U32.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中对 32 位数据执行矢量无符号加法。然后，从计算中得到的 4 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，寄存器 qu 中的值存储在内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[ 31:  0] = min(qx[ 31:  0] + qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] + qy[ 63: 32], 2^{32}-1)
...
qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

格式

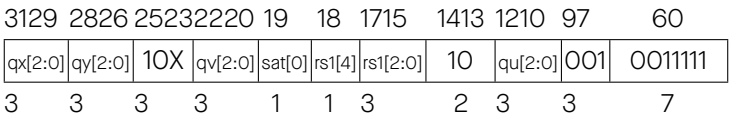


图 4.7-106. ESP.VADD.U32.ST.INCP

4.7.2.107 ESP.VADD.U8

语法

ESP.VADD.U8 qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中的 8 位数据上执行矢量无符号加法。然后，对从计算中得到的 16 个结果进行饱和，将饱和结果写入寄存器 qv。

操作

```
qv[  7:  0] = min(qx[  7:  0] + qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] + qy[ 15:  8], 2^{8}-1)
...
qv[127:120] = min(qx[127:120] + qy[127:120], 2^{8}-1)
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1

格式

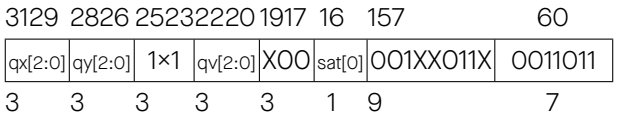


图 4.7-107. ESP.VADD.U8

4.7.2.108 ESP.VADD.U8.LD.INCP

语法

ESP.VADD.U8.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令在两个寄存器 qx 和 qy 中对 8 位数据执行矢量无符号加法。然后，从计算中得到的 16 个结果进行饱和，饱和结果写入寄存器 qv。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 中的值增加 16。

操作

```
qv[  7:  0] = min(qx[  7:  0] + qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] + qy[ 15:  8], 2^{8}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	-	2
rs1	1	1

格式

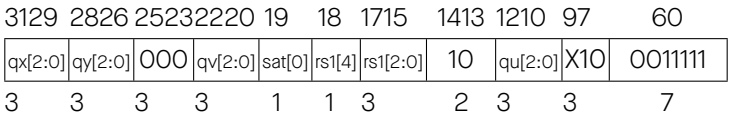


图 4.7-108. ESP.VADD.U8.LD.INCP

4.7.2.109 ESP.VADD.U8.ST.INCP

语法

ESP.VADD.U8.ST.INCP qu,rs1,qv,qx,qy,sat **描述**

此指令在两个寄存器 qx 和 qy 中对 8 位数据执行矢量无符号加法。然后，从计算中得到的 16 个结果进行饱和，将饱和结果写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储在内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[  7:  0] = min(qx[  7:  0] + qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] + qy[ 15:  8], 2^{8}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qv	-	1
qu	2	-
rs1	1	1

格式

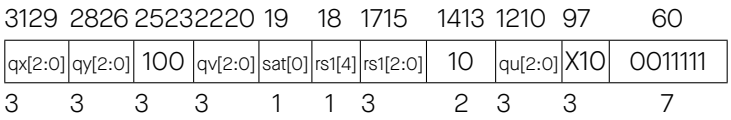


图 4.7-109. ESP.VADD.U8.ST.INCP

4.7.2.110 ESP.VCLAMP.S16

语法

ESP.VCLAMP.S16 qz,qx,sel16

描述

此指令对 16 位有符号数据执行夹紧操作。

操作

```
    qz[ 15:  0] = min( max(qx[ 15:  0], -2^{sel16}), 2^{sel16}-1 )
    qz[ 31: 16] = min( max(qx[ 31: 16], -2^{sel16}), 2^{sel16}-1 )
    qz[ 47: 32] = min( max(qx[ 47: 32], -2^{sel16}), 2^{sel16}-1 )
    qz[ 63: 48] = min( max(qx[ 63: 48], -2^{sel16}), 2^{sel16}-1 )
    qz[ 79: 64] = min( max(qx[ 79: 64], -2^{sel16}), 2^{sel16}-1 )
    qz[ 95: 80] = min( max(qx[ 95: 80], -2^{sel16}), 2^{sel16}-1 )
    qz[111: 96] = min( max(qx[111: 96], -2^{sel16}), 2^{sel16}-1 )
    qz[127:112] = min( max(qx[127:112], -2^{sel16}), 2^{sel16}-1 )
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

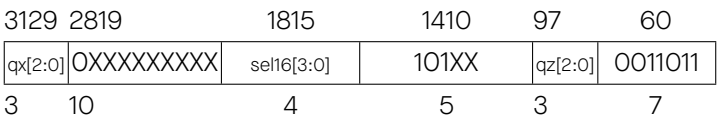


图 4.7-110. ESP.VCLAMP.S16

4.7.2.111 ESP.VMAX.S16

语法

ESP.VMAX.S16 qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位有符号向量数据段的数值大小。较大的数据段将被写入到寄存器qz中对应的 16 位数据段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

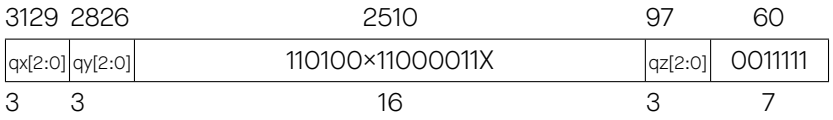


图 4.7-111. ESP.VMAX.S16

4.7.2.112 ESP.VMAX.S16.LD.INCP

语法

ESP.VMAX.S16.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位有符号向量数据段的数值。较大值的数据段被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

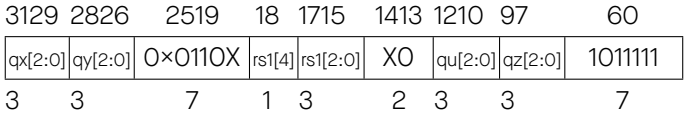


图 4.7-112. ESP.VMAX.S16.LD.INCP

4.7.2.113 ESP.VMAX.S16.ST.INCP

语法

ESP.VMAX.S16.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位有符号向量数据段的数值。较大的数据段值将被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

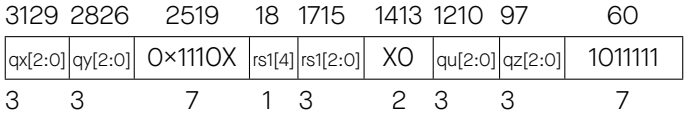


图 4.7-113. ESP.VMAX.S16.ST.INCP

4.7.2.114 ESP.VMAX.S32

语法

ESP.VMAX.S32 qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位有符号向量数据段的数值大小。较大的数据段将被写入到寄存器qz中对应的 32 位数据段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

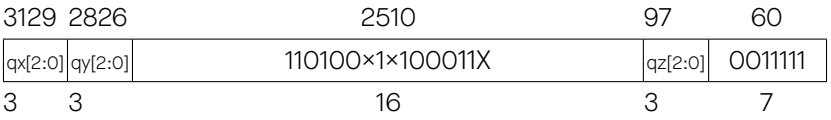


图 4.7-114. ESP.VMAX.S32

4.7.2.115 ESP.VMAX.S32.LD.INCP

语法

ESP.VMAX.S32.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位有符号向量数据段的数值大小。较大的数据段值将被写入到寄存器qz中对应的 32 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1的值增加 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

 qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

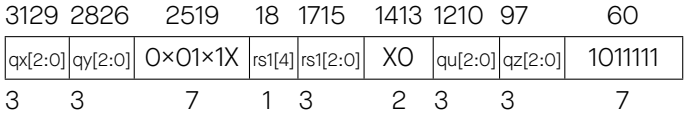


图 4.7-115. ESP.VMAX.S32.LD.INCP

4.7.2.116 ESP.VMAX.S32.ST.INCP

语法

ESP.VMAX.S32.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位有符号向量数据段的数值大小。较大的数据段值将被写入到寄存器qz中对应的 32 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

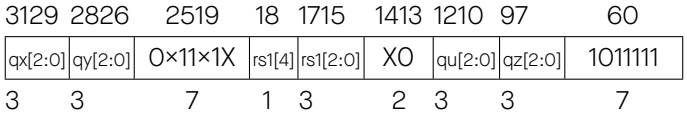


图 4.7-116. ESP.VMAX.S32.ST.INCP

4.7.2.117 ESP.VMAX.S8

语法

ESP.VMAX.S8 qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位有符号向量数据段的数值。较大值的数据段将被写入到寄存器qz中对应的 8 位数据段。

操作

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

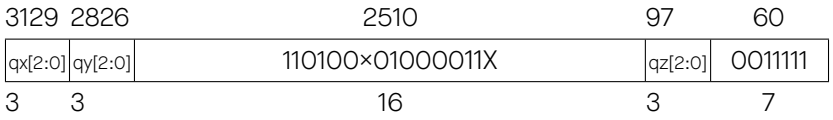


图 4.7-117. ESP.VMAX.S8

4.7.2.118 ESP.VMAX.S8.LD.INCP

语法

ESP.VMAX.S8.LD.INCP qu,rs1,qz,qx,qy

描述

本指令比较寄存器qx和qy中的 16 个 8 位有符号向量数据段的数值大小。较大的数据段将被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

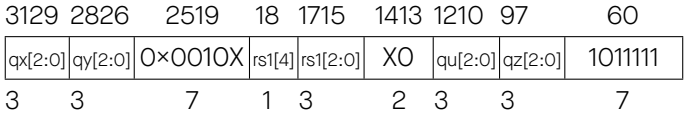


图 4.7-118. ESP.VMAX.S8.LD.INCP

4.7.2.119 ESP.VMAX.S8.ST.INCP

语法

ESP.VMAX.S8.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位向量数据段的数值。较大值的数据段将被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，寄存器qu中的值被存储到内存中。访问后，寄存器rs1中的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

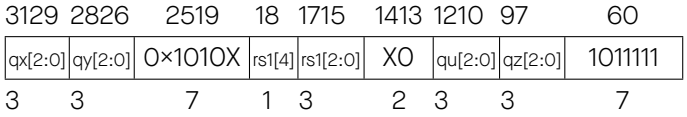


图 4.7-119. ESP.VMAX.S8.ST.INCP

4.7.2.120 ESP.VMAX.U16

语法

ESP.VMAX.U16 qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位无符号向量数据段的数值。较大值的数据段将被写入到寄存器qz中对应的 16 位数据段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

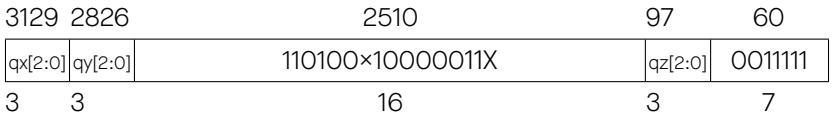


图 4.7-120. ESP.VMAX.U16

4.7.2.121 ESP.VMAX.U16.LD.INCP

语法

ESP.VMAX.U16.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位无符号向量数据段的数值。较大值的数据段被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

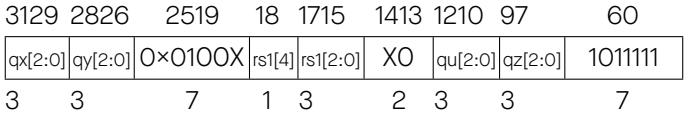


图 4.7-121. ESP.VMAX.U16.LD.INCP

4.7.2.122 ESP.VMAX.U16.ST.INCP

语法

ESP.VMAX.U16.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位无符号向量数据段的数值。较大值的数据段被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]>=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]>=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

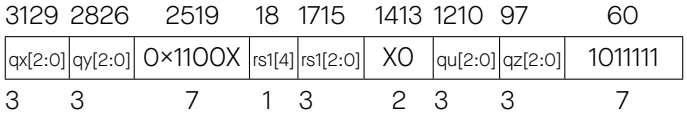


图 4.7-122. ESP.VMAX.U16.ST.INCP

4.7.2.123 ESP.VMAX.U32

语法

ESP.VMAX.U32 qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位无符号向量数据段的数值大小。较大的数据段将被写入到寄存器qz中对应的 32 位数据段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

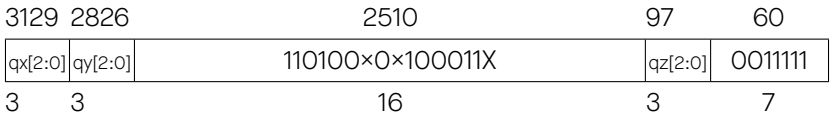


图 4.7-123. ESP.VMAX.U32

4.7.2.124 ESP.VMAX.U32.LD.INCP

语法

ESP.VMAX.U32.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位无符号向量数据段的数值。较大的数据段值将被写入到寄存器qz中对应的 32 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1的值增加 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

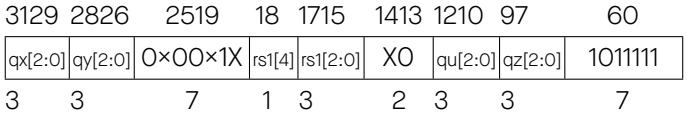


图 4.7-124. ESP.VMAX.U32.LD.INCP

4.7.2.125 ESP.VMAX.U32.ST.INCP

语法

ESP.VMAX.U32.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位无符号向量数据段的数值大小。较大的数据段值将被写入到寄存器qz中对应的 32 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]>=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]>=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

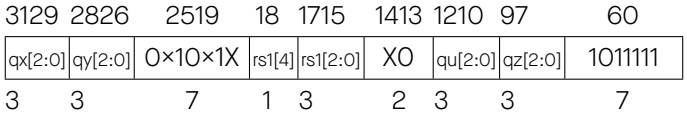


图 4.7-125. ESP.VMAX.U32.ST.INCP

4.7.2.126 ESP.VMAX.U8

语法

ESP.VMAX.U8 qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位无符号向量数据段的数值大小。较大的数据段将被写入到寄存器qz中对应的 8 位数据段。

操作

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

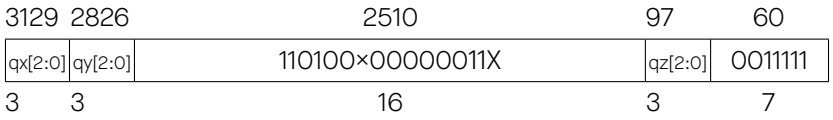


图 4.7-126. ESP.VMAX.U8

4.7.2.127 ESP.VMAX.U8.LD.INCP

语法

ESP.VMAX.U8.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位无符号向量数据段的数值。较大值的数据段被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

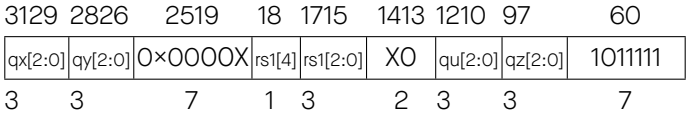


图 4.7-127. ESP.VMAX.U8.LD.INCP

4.7.2.128 ESP.VMAX.U8.ST.INCP

语法

ESP.VMAX.U8.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位无符号向量数据段的数值。较大的数据段值将被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加了 16。

操作

```
qz[  7:  0] = (qx[  7:  0]>=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]>=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]>=qy[127:120]) ? qx[127:120] : qy[127:120]

  qu[127:0] => store128(rs1[31:0])
rs1 [31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

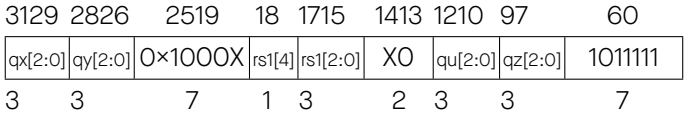


图 4.7-128. ESP.VMAX.U8.ST.INCP

4.7.2.129 ESP.VMIN.S16

语法

ESP.VMIN.S16 qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位有符号向量数据段的数值。较小值的数据段将被写入到寄存器qz中对应的 16 位数据段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

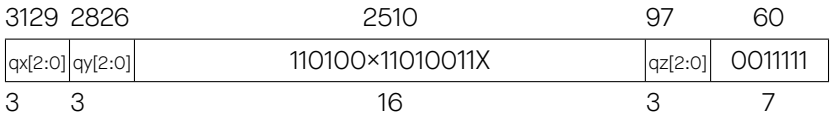


图 4.7-129. ESP.VMIN.S16

4.7.2.130 ESP.VMIN.S16.LD.INCP

语法

ESP.VMIN.S16.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位有符号向量数据段的数值。较小值的数据段被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

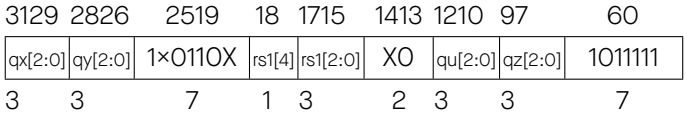


图 4.7-130. ESP.VMIN.S16.LD.INCP

4.7.2.131 ESP.VMIN.S16.ST.INCP

语法

ESP.VMIN.S16.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位有符号向量数据段的数值。较小的数据段值将被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加了 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

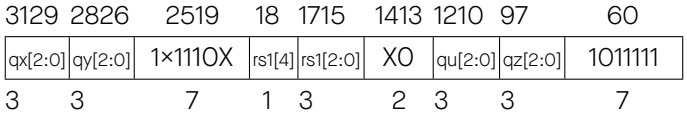


图 4.7-131. ESP.VMIN.S16.ST.INCP

4.7.2.132 ESP.VMIN.S32

语法

ESP.VMIN.S32 qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位有符号向量数据段的数值。较小值的数据段将被写入到寄存器qz中对应的 32 位数据段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

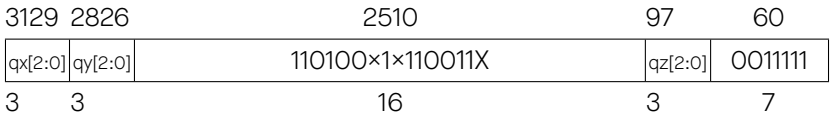


图 4.7-132. ESP.VMIN.S32

4.7.2.133 ESP.VMIN.S32.LD.INCP

语法

ESP.VMIN.S32.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位有符号向量数据段的数值。较小的数据段值将被写入到寄存器qz中对应的 32 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1中的值增加 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

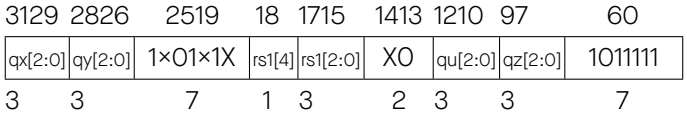


图 4.7-133. ESP.VMIN.S32.LD.INCP

4.7.2.134 ESP.VMIN.S32.ST.INCP

语法

ESP.VMIN.S32.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位有符号向量数据段的数值。较小的数据段值被写入到寄存器qz中对应的 32 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加了 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

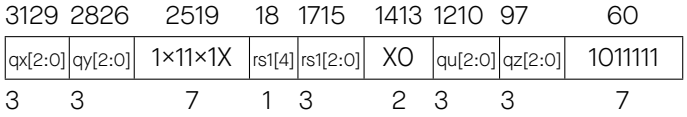


图 4.7-134. ESP.VMIN.S32.ST.INCP

4.7.2.135 ESP.VMIN.S8

语法

ESP.VMIN.S8 qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位有符号向量数据段的数值大小。较小的数据段值将被写入到寄存器qz中对应的 8 位数据段。

操作

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

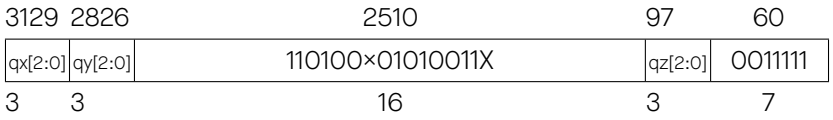


图 4.7-135. ESP.VMIN.S8

4.7.2.136 ESP.VMIN.S8.LD.INCP

语法

ESP.VMIN.S8.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位有符号向量数据段的数值。较小值的数据段被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

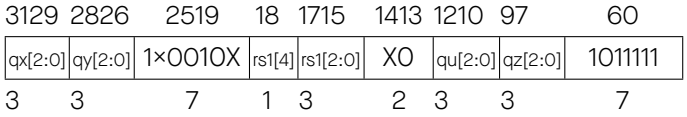


图 4.7-136. ESP.VMIN.S8.LD.INCP

4.7.2.137 ESP.VMIN.S8.ST.INCP

语法

ESP.VMIN.S8.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位有符号向量数据段的数值。较小的数据段值被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加了 16。

操作

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

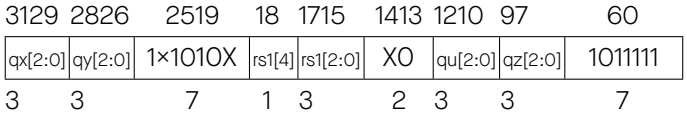


图 4.7-137. ESP.VMIN.S8.ST.INCP

4.7.2.138 ESP.VMIN.U16

语法

ESP.VMIN.U16 qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位无符号向量数据段的数值。较小的数据段值将被写入到寄存器qz中对应的 16 位数据段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

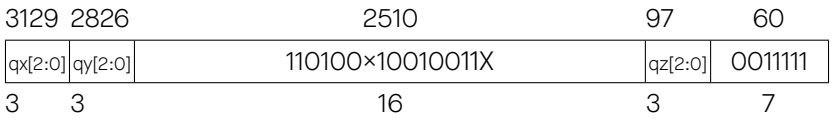


图 4.7-138. ESP.VMIN.U16

4.7.2.139 ESP.VMIN.U16.LD.INCP

语法

ESP.VMIN.U16.LD.INCP qu,rs1,qz,qx,qy

描述

本指令比较寄存器qx和qy中的八个 16 位无符号向量数据段的数值大小。较小的数据段值将被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

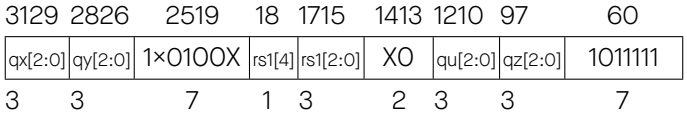


图 4.7-139. ESP.VMIN.U16.LD.INCP

4.7.2.140 ESP.VMIN.U16.ST.INCP

语法

ESP.VMIN.U16.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的八个 16 位无符号向量数据段的数值。较小的数据段值被写入到寄存器qz中对应的 16 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加了 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<=qy[ 15:  0]) ? qx[ 15:  0] : qy[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=qy[ 31: 16]) ? qx[ 31: 16] : qy[ 31: 16]
...
qz[127:112] = (qx[127:112]<=qy[127:112]) ? qx[127:112] : qy[127:112]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

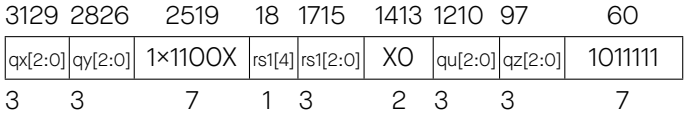


图 4.7-140. ESP.VMIN.U16.ST.INCP

4.7.2.141 ESP.VMIN.U32

语法

ESP.VMIN.U32 qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位无符号向量数据段的数值。较小值的数据段将被写入到寄存器qz中对应的 32 位数据段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

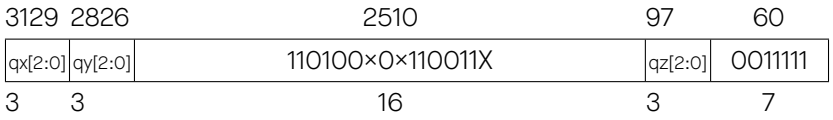


图 4.7-141. ESP.VMIN.U32

4.7.2.142 ESP.VMIN.U32.LD.INCP

语法

ESP.VMIN.U32.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器 qx 和 qy 中的四个 32 位无符号向量数据段的数值大小。较小的数据段值将被写入到寄存器 qz 中对应的 32 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 的值增加 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

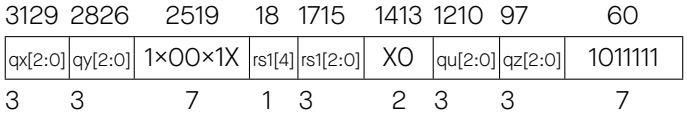


图 4.7-142. ESP.VMIN.U32.LD.INCP

4.7.2.143 ESP.VMIN.U32.ST.INCP

语法

ESP.VMIN.U32.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的四个 32 位无符号向量数据段的数值。较小的数据段值被写入到寄存器qz中对应的 32 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加了 16。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<=qy[ 31:  0]) ? qx[ 31:  0] : qy[ 31:  0]
qz[ 63: 32] = (qx[ 63: 32]<=qy[ 63: 32]) ? qx[ 63: 32] : qy[ 63: 32]
...
qz[127: 96] = (qx[127: 96]<=qy[127: 96]) ? qx[127: 96] : qy[127: 96]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

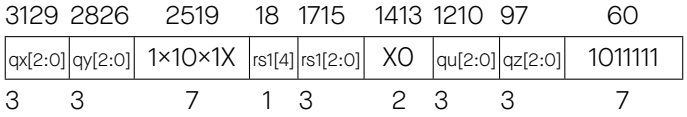


图 4.7-143. ESP.VMIN.U32.ST.INCP

4.7.2.144 ESP.VMIN.U8

语法

ESP.VMIN.U8 qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位无符号向量数据段的数值。较小的数据段值将被写入到寄存器qz中对应的 8 位数据段。

操作

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1

格式

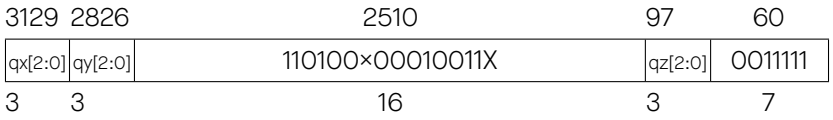


图 4.7-144. ESP.VMIN.U8

4.7.2.145 ESP.VMIN.U8.LD.INCP

语法

ESP.VMIN.U8.LD.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位无符号向量数据段的数值。较小值的数据段被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，从内存中加载 16 字节到寄存器qu。访问后，寄存器rs1的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	-	2
rs1	1	1

格式

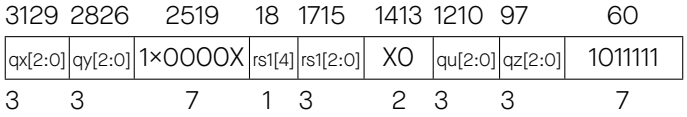


图 4.7-145. ESP.VMIN.U8.LD.INCP

4.7.2.146 ESP.VMIN.U8.ST.INCP

语法

ESP.VMIN.U8.ST.INCP qu,rs1,qz,qx,qy

描述

此指令比较寄存器qx和qy中的 16 个 8 位无符号向量数据段的数值。较小的数据段值被写入到寄存器qz中对应的 8 位数据段。

在操作过程中，寄存器qu中的值被存储在内存中。访问后，寄存器rs1中的值增加了 16。

操作

```
qz[  7:  0] = (qx[  7:  0]<=qy[  7:  0]) ? qx[  7:  0] : qy[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=qy[ 15:  8]) ? qx[ 15:  8] : qy[ 15:  8]
...
qz[127:120] = (qx[127:120]<=qy[127:120]) ? qx[127:120] : qy[127:120]

qu[127:0] = store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	1
qu	2	-
rs1	1	1

格式

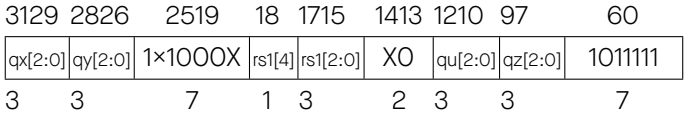


图 4.7-146. ESP.VMIN.U8.ST.INCP

4.7.2.147 ESP.VMUL.S16

语法

ESP.VMUL.S16 qz,qx,qy,sat,rm

描述

此指令对 16 位数据执行有符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 32 位数据结果通过特殊寄存器 SAR 中的值进行算术右移。然后，将移位结果的低 16 位数据写入寄存器 qz 的相应段。

操作

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-

格式

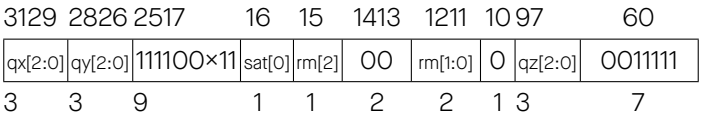


图 4.7-147. ESP.VMUL.S16

4.7.2.148 ESP.VMUL.S16.LD.INCP

语法

ESP.VMUL.S16.LD.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 16 位数据执行有符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 32 位数据结果通过特殊寄存器 SAR 的值进行算术右移。然后，将移位结果的低 16 位数据写入寄存器 qz 的相应段。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

格式

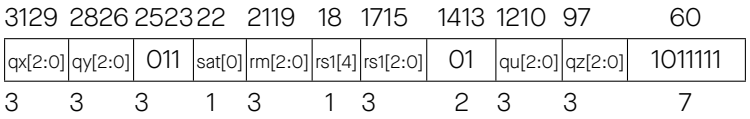


图 4.7-148. ESP.VMUL.S16.LD.INCP

4.7.2.149 ESP.VMUL.S16.S8XS8

语法

ESP.VMUL.S16.S8XS8 qz,qv,qx,qy,rm

描述

此指令对 8 位数据执行有符号向量乘法。寄存器qx和qy分别是乘数和被乘数。从计算中得到的 16 个 16 位数据结果通过特殊寄存器SAR中的值进行算术右移。然后，将移位结果写入寄存器qz和qv的相应段中。

操作

```
qz[ 15:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[111: 96] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[127:112] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]

qv[ 15:  0] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qv[ 31: 16] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qv[ 47: 32] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qv[ 63: 48] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qv[ 79: 64] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qv[ 95: 80] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qv[111: 96] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qv[127:112] = (qx[127:120] * qy[127:120]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
qv	-	2
SAR	1	-

格式

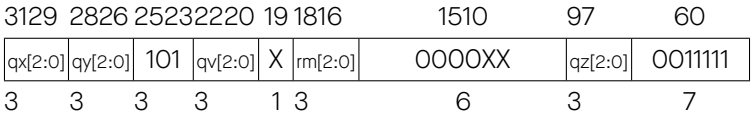


图 4.7-149. ESP.VMUL.S16.S8XS8

4.7.2.150 ESP.VMUL.S16.ST.INCP

语法

ESP.VMUL.S16.ST.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 16 位数据执行有符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 32 位数据结果通过特殊寄存器 SAR 中的值进行算术右移。然后，将移位结果的低 16 位数据写入寄存器 qz 的相应段中。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

格式

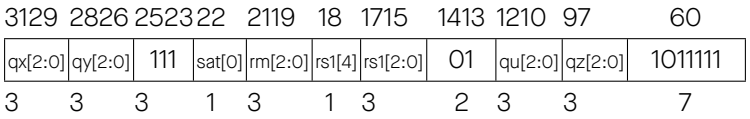


图 4.7-150. ESP.VMUL.S16.ST.INCP

4.7.2.151 ESP.VMUL.S32.S16XS16

语法

ESP.VMUL.S32.S16XS16 qz,qv,qx,qy,rm

描述

此指令对 16 位数据执行有符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 32 位数据结果将通过特殊寄存器 SAR 中的值进行算术右移。然后，将移位结果写入寄存器 qz 和 qv 的相应段中。

操作

```
qz[ 31:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 63:  0] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 95: 64] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[127: 96] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]

qv[ 31:  0] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qv[ 63: 32] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qv[ 95: 64] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qv[127: 96] = (qx[127:112] * qy[127:112]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
qv	-	2
SAR	1	-

格式

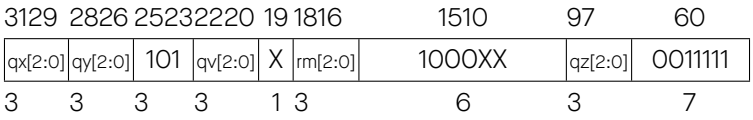


图 4.7-151. ESP.VMUL.S32.S16XS16

4.7.2.152 ESP.VMUL.S8

语法

ESP.VMUL.S8 qz,qx,qy,sat,rm

描述

此指令对 8 位数据执行有符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 16 位数据结果通过特殊寄存器 SAR 中的值进行算术右移。然后，将移位结果的低 8 位数据写入寄存器 qz 的相应段。

操作

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[4:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[4:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[4:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[4:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[4:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[4:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[4:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[4:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[4:0]
qz[ 79: 71] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[4:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[4:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[4:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[4:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[4:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[4:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[4:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-

格式

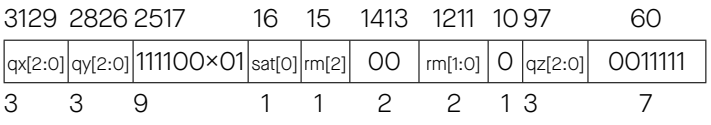


图 4.7-152. ESP.VMUL.S8

4.7.2.153 ESP.VMUL.S8.LD.INCP

语法

ESP.VMUL.S8.LD.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 8 位数据执行有符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 16 位数据结果通过特殊寄存器 SAR 的值进行算术右移。然后，将移位结果的低 8 位数据写入寄存器 qz 的相应段。

在操作过程中，从内存中加载 16 字节到寄存器 qu。访问后，寄存器 rs1 的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[4:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[4:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[4:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[4:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[4:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[4:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[4:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[4:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[4:0]
qz[ 79: 71] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[4:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[4:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[4:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[4:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[4:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[4:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[4:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

格式

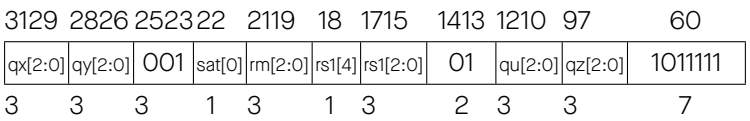


图 4.7-153. ESP.VMUL.S8.LD.INCP

4.7.2.154 ESP.VMUL.S8.ST.INCP

语法

ESP.VMUL.S8.ST.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 8 位数据执行有符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 16 位数据结果通过特殊寄存器 SAR 中的值进行算术右移。然后，将移位结果的低 8 位数据写入寄存器 qz 的相应段。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[4:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[4:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[4:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[4:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[4:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[4:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[4:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[4:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[4:0]
qz[ 79: 71] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[4:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[4:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[4:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[4:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[4:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[4:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[4:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

格式

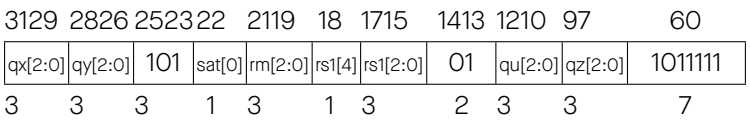


图 4.7-154. ESP.VMUL.S8.ST.INCP

4.7.2.155 ESP.VMUL.U16

语法

ESP.VMUL.U16 qz,qx,qy,sat,rm

描述

此指令对 16 位数据执行无符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 32 位数据结果被逻辑右移，移位的值由特殊寄存器 SAR 中的值决定。然后，将移位结果的低 16 位数据写入寄存器 qz 的相应段中。

操作

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-

格式

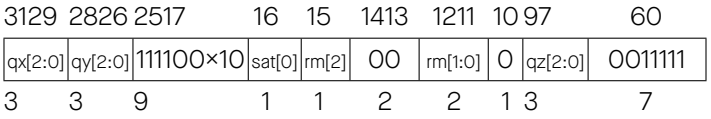


图 4.7-155. ESP.VMUL.U16

4.7.2.156 ESP.VMUL.U16.LD.INCP

语法

ESP.VMUL.U16.LD.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 16 位数据执行无符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 32 位数据结果通过特殊寄存器 SAR 中的值逻辑右移。然后，将移位结果的低 16 位数据写入寄存器 qz 的相应段。
在操作过程中，将 16 字节数据从内存加载到寄存器 qu。访问后，寄存器 rs1 中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

格式

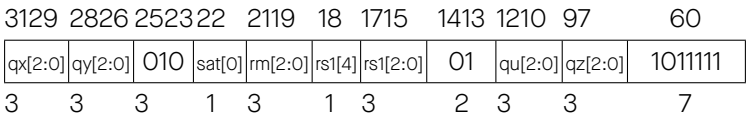


图 4.7-156. ESP.VMUL.U16.LD.INCP

4.7.2.157 ESP.VMUL.U16.ST.INCP

语法

ESP.VMUL.U16.ST.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 16 位数据执行无符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的八个 32 位数据结果被逻辑右移，移位值由特殊寄存器 SAR 中的值决定。然后，移位结果的低 16 位数据被写入寄存器 qz 的相应段。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qz[ 15:  0] = (qx[ 15:  0] * qy[ 15:  0]) >> SAR[5:0]
qz[ 31: 16] = (qx[ 31: 16] * qy[ 31: 16]) >> SAR[5:0]
qz[ 47: 32] = (qx[ 47: 32] * qy[ 47: 32]) >> SAR[5:0]
qz[ 63: 48] = (qx[ 63: 48] * qy[ 63: 48]) >> SAR[5:0]
qz[ 79: 64] = (qx[ 79: 64] * qy[ 79: 64]) >> SAR[5:0]
qz[ 95: 80] = (qx[ 95: 80] * qy[ 95: 80]) >> SAR[5:0]
qz[111: 96] = (qx[111: 96] * qy[111: 96]) >> SAR[5:0]
qz[127:112] = (qx[127:112] * qy[127:112]) >> SAR[5:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

格式

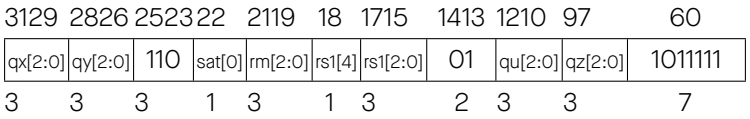


图 4.7-157. ESP.VMUL.U16.ST.INCP

4.7.2.158 ESP.VMUL.U8

语法

ESP.VMUL.U8 qz,qx,qy,sat,rm

描述

此指令对 8 位数据执行无符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的 16 个 16 位数据结果通过特殊寄存器 SAR 中的值逻辑右移。然后，将移位结果的低 8 位数据写入寄存器 qz 的相应段。

操作

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qz[ 79: 72] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[5:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

格式

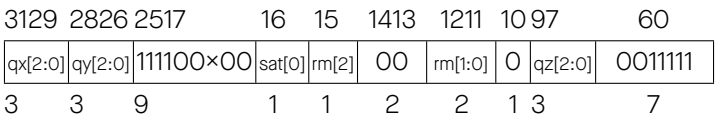


图 4.7-158. ESP.VMUL.U8

4.7.2.159 ESP.VMUL.U8.LD.INCP

语法

ESP.VMUL.U8.LD.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 8 位数据执行无符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的 16 个 32 位数据结果通过特殊寄存器 SAR 中的值逻辑右移。然后，将移位结果的低 8 位数据写入寄存器 qz 的相应段。

在操作过程中，将 16 字节数据从内存加载到寄存器 qu。访问后，寄存器 rs1 中的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qz[ 79: 72] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[5:0]

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	-	2
rs1	1	1

格式

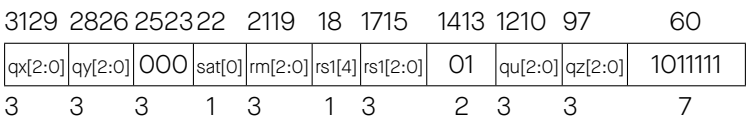


图 4.7-159. ESP.VMUL.U8.LD.INCP

4.7.2.160 ESP.VMUL.U8.ST.INCP

语法

ESP.VMUL.U8.ST.INCP qu,rs1,qz,qx,qy,sat,rm

描述

此指令对 8 位数据执行无符号向量乘法。寄存器 qx 和 qy 分别是乘数和被乘数。从计算中得到的 16 个 16 位数据结果通过特殊寄存器 SAR 的值逻辑右移。然后，将移位结果的低 8 位数据写入寄存器 qz 的相应段。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加 16。

操作

```
qz[  7:  0] = (qx[  7:  0] * qy[  7:  0]) >> SAR[5:0]
qz[ 15:  8] = (qx[ 15:  8] * qy[ 15:  8]) >> SAR[5:0]
qz[ 23: 16] = (qx[ 23: 16] * qy[ 23: 16]) >> SAR[5:0]
qz[ 31: 24] = (qx[ 31: 24] * qy[ 31: 24]) >> SAR[5:0]
qz[ 39: 32] = (qx[ 39: 32] * qy[ 39: 32]) >> SAR[5:0]
qz[ 47: 40] = (qx[ 47: 40] * qy[ 47: 40]) >> SAR[5:0]
qz[ 55: 48] = (qx[ 55: 48] * qy[ 55: 48]) >> SAR[5:0]
qz[ 63: 56] = (qx[ 63: 56] * qy[ 63: 56]) >> SAR[5:0]
qz[ 71: 64] = (qx[ 71: 64] * qy[ 71: 64]) >> SAR[5:0]
qz[ 79: 72] = (qx[ 79: 72] * qy[ 79: 72]) >> SAR[5:0]
qz[ 87: 80] = (qx[ 87: 80] * qy[ 87: 80]) >> SAR[5:0]
qz[ 95: 88] = (qx[ 95: 88] * qy[ 95: 88]) >> SAR[5:0]
qz[103: 96] = (qx[103: 96] * qy[103: 96]) >> SAR[5:0]
qz[111:104] = (qx[111:104] * qy[111:104]) >> SAR[5:0]
qz[119:112] = (qx[119:112] * qy[119:112]) >> SAR[5:0]
qz[127:120] = (qx[127:120] * qy[127:120]) >> SAR[5:0]

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
SAR	1	-
qu	2	-
rs1	1	1

格式

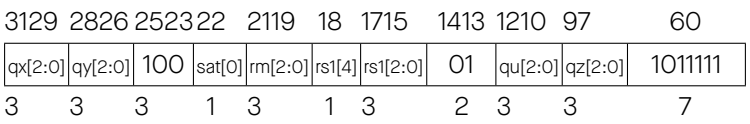


图 4.7-160. ESP.VMUL.U8.ST.INCP

4.7.2.161 ESP.VPRELU.S16

语法

ESP.VPRELU.S16 qz,qy,qx,rs1,sat,rm

描述

此指令将寄存器qx划分为 8 个 16 位的数据段。如果某个段的值不大于 0，它将与寄存器qy中对应的 16 位段的值进行乘法运算，该值由寄存器rs1中的低 6 位值右移得到。然后，将得到的结果赋值给寄存器qz中对应的 16 位数据段。否则，将qx中的段值赋给qz。数据是有符号的。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<=0) ? (qx[ 15:  0] * qy[ 15:  0]) >> rs1[5:0] : qx[ 15:  0]
qz[ 31: 16] = (qx[ 31: 16]<=0) ? (qx[ 31: 16] * qy[ 31: 16]) >> rs1[5:0] : qx[ 31: 16]
...
qz[127:112] = (qx[127:112]<=0) ? (qx[127:112] * qy[127:112]) >> rs1[5:0] : qx[127:112]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
rs1	1	-

格式

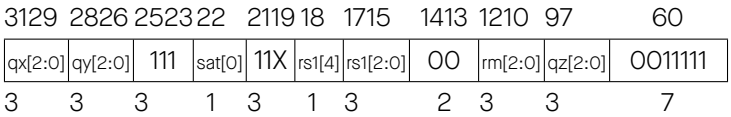


图 4.7-161. ESP.VPRELU.S16

4.7.2.162 ESP.VPRELU.S8

语法

ESP.VPRELU.S8 qz,qy,qx,rs1,sat,rm

描述

此指令将寄存器qx划分为 16 个 8 位的数据段。如果某个段的值不大于 0，它将与寄存器qy中对应的 8 位段的值进行乘法运算，然后通过寄存器rs1中的低 5 位值进行右移，然后将计算结果赋值给寄存器qz中对应的 8 位数据段。否则，将qx中的段值赋给qz。

操作

```
qz[  7:  0] = (qx[  7:  0]<=0) ? (qx[  7:  0] * qy[  7:  0]) >> rs1[4:0] : qx[  7:  0]
qz[ 15:  8] = (qx[ 15:  8]<=0) ? (qx[ 15:  8] * qy[ 15:  8]) >> rs1[4:0] : qx[ 15:  8]
...
qz[127:120] = (qx[127:120]<=0) ? (qx[127:120] * qy[127:120]) >> rs1[4:0] : qx[127:120]
```

调度

操作数	读流水级数	写流水级数
qx	1	-
qy	1	-
qz	-	2
rs1	1	-

格式

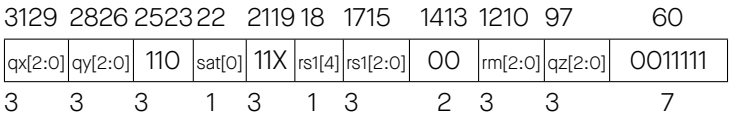


图 4.7-162. ESP.VPRELU.S8

4.7.2.163 ESP.VRELU.S16

语法

ESP.VRELU.S16 qy,rs2,rs1,sat,rm

描述

此指令将寄存器 qy 分为 8 个 16 位的数据段。如果段的值不大于 0，它将乘以寄存器 rs2 中下 16 位的值，并右移寄存器 rs1 中下 6 位的值，然后得到的结果将覆盖段的值。否则，段的值将保持不变。

操作

```
qy[ 15:  0] = (qy[ 15:  0]<=0) ? (qy[ 15:  0] * rs2[15:0]) >> rs1[5:0] : qy[ 15:
    0]
qy[ 31: 16] = (qy[ 31: 16]<=0) ? (qy[ 31: 16] * rs2[15:0]) >> rs1[5:0] : qy[ 31:
    16]
...
qy[127:112] = (qy[127:112]<=0) ? (qy[127:112] * rs2[15:0]) >> rs1[5:0] : qy
    [127:112]
```

调度

操作数	读流水级数	写流水级数
qy	1	2
rs2	1	-
rs1	1	-

格式

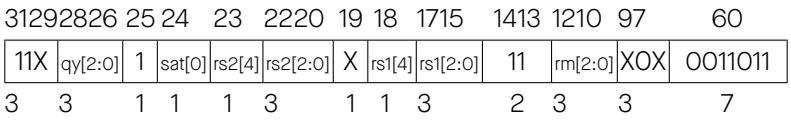


图 4.7-163. ESP.VRELU.S16

4.7.2.164 ESP.VRELU.S8

语法

ESP.VRELU.S8 qy,rs2,rs1,sat,rm

描述

此指令将寄存器qy划分为 16 个 8 位的数据段。如果段的值不大于 0，它将被寄存器rs2的低 8 位的值乘以，并右移寄存器rs1的低 5 位的值，然后得到的结果将覆盖段的值。否则，段的值将保持不变。

操作

```
qy[  7:  0] = (qy[  7:  0]<=0) ? (qy[  7:  0] * rs2[7:0]) >> rs1[4:0] : qy[  7:  0]
qy[ 15:  8] = (qy[ 15:  8]<=0) ? (qy[ 15:  8] * rs2[7:0]) >> rs1[4:0] : qy[ 15:  8]
...
qy[127:120] = (qy[127:120]<=0) ? (qy[127:120] * rs2[7:0]) >> rs1[4:0] : qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qy	1	2
rs2	1	-
rs1	1	-

格式

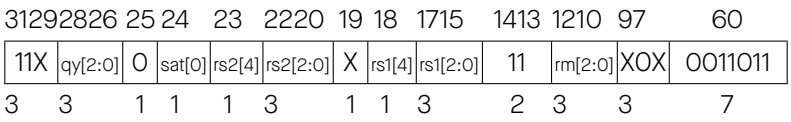


图 4.7-164. ESP.VRELU.S8

4.7.2.165 ESP.VSADDS.S16

语法

ESP.VSADDS.S16 qv,qx,rs1,sat

描述

此指令对 16 位数据执行有符号向量加法。

操作

```
qv[ 15:  0] = min( max( qx[ 15:  0] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 31: 16] = min( max( qx[ 31: 16] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 47: 32] = min( max( qx[ 47: 32] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 63: 48] = min( max( qx[ 63: 48] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 79: 64] = min( max( qx[ 79: 64] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 95: 80] = min( max( qx[ 95: 80] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[111: 96] = min( max( qx[111: 96] + rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[127:112] = min( max( qx[127:112] + rs1[15:0], -2^{15} ), 2^{15}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	2
qx	1	-
rs1	1	-

格式

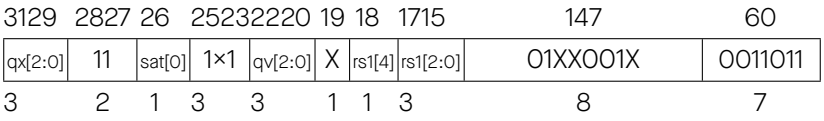


图 4.7-165. ESP.VSADDS.S16

4.7.2.166 ESP.VSADDS.S8

语法

ESP.VSADDS.S8 qv,qx,rs1,sat

描述

此指令对 8 位数据执行有符号向量加法。

操作

```
qv[  7:  0] = min( max( qx[  7:  0] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 15:  8] = min( max( qx[ 15:  8] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 23: 16] = min( max( qx[ 23: 16] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 31: 24] = min( max( qx[ 31: 24] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 39: 32] = min( max( qx[ 39: 32] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 47: 40] = min( max( qx[ 47: 40] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 55: 48] = min( max( qx[ 55: 48] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 63: 56] = min( max( qx[ 63: 56] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 71: 64] = min( max( qx[ 71: 64] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 79: 72] = min( max( qx[ 79: 72] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 87: 80] = min( max( qx[ 87: 80] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 95: 88] = min( max( qx[ 95: 88] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[103: 96] = min( max( qx[103: 96] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[111:104] = min( max( qx[111:104] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[119:112] = min( max( qx[119:112] + rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[127:120] = min( max( qx[127:120] + rs1[7:0], -2^{7} ), 2^{7}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	2
qx	1	-
rs1	1	-

格式

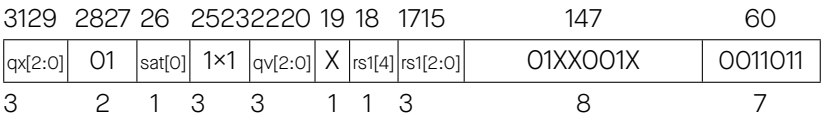


图 4.7-166. ESP.VSADDS.S8

4.7.2.167 ESP.VSADDS.U16

语法

ESP.VSADDS.U16 qv,qx,rs1,sat

描述

此指令对 16 位数据执行无符号向量加法。

操作

```
    qv[ 15:  0] = min( qx[ 15:  0] + rs1[15:0], 2^{16}-1 )
    qv[ 31: 16] = min( qx[ 31: 16] + rs1[15:0], 2^{16}-1 )
    qv[ 47: 32] = min( qx[ 47: 32] + rs1[15:0], 2^{16}-1 )
    qv[ 63: 48] = min( qx[ 63: 48] + rs1[15:0], 2^{16}-1 )
    qv[ 79: 64] = min( qx[ 79: 64] + rs1[15:0], 2^{16}-1 )
    qv[ 95: 80] = min( qx[ 95: 80] + rs1[15:0], 2^{16}-1 )
    qv[111: 96] = min( qx[111: 96] + rs1[15:0], 2^{16}-1 )
    qv[127:112] = min( qx[127:112] + rs1[15:0], 2^{16}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	2
qx	1	-
rs1	1	-

格式

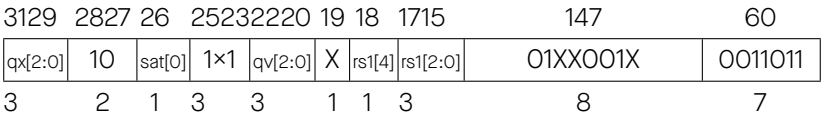


图 4.7-167. ESP.VSADDS.U16

4.7.2.168 ESP.VSADDS.U8

语法

ESP.VSADDS.U8 qv,qx,rs1,sat

描述

此指令对 8 位数据执行无符号向量加法。

操作

```
    qv[  7:  0] = min( qx[  7:  0] + rs1[7:0], 2^{8}-1 )
    qv[ 15:  8] = min( qx[ 15:  8] + rs1[7:0], 2^{8}-1 )
    qv[ 23: 16] = min( qx[ 23: 16] + rs1[7:0], 2^{8}-1 )
    qv[ 31: 24] = min( qx[ 31: 24] + rs1[7:0], 2^{8}-1 )
    qv[ 39: 32] = min( qx[ 39: 32] + rs1[7:0], 2^{8}-1 )
    qv[ 47: 40] = min( qx[ 47: 40] + rs1[7:0], 2^{8}-1 )
    qv[ 55: 48] = min( qx[ 55: 48] + rs1[7:0], 2^{8}-1 )
    qv[ 63: 56] = min( qx[ 63: 56] + rs1[7:0], 2^{8}-1 )
    qv[ 71: 64] = min( qx[ 71: 64] + rs1[7:0], 2^{8}-1 )
    qv[ 79: 72] = min( qx[ 79: 72] + rs1[7:0], 2^{8}-1 )
    qv[ 87: 80] = min( qx[ 87: 80] + rs1[7:0], 2^{8}-1 )
    qv[ 95: 88] = min( qx[ 95: 88] + rs1[7:0], 2^{8}-1 )
    qv[103: 96] = min( qx[103: 96] + rs1[7:0], 2^{8}-1 )
    qv[111:104] = min( qx[111:104] + rs1[7:0], 2^{8}-1 )
    qv[119:112] = min( qx[119:112] + rs1[7:0], 2^{8}-1 )
    qv[127:120] = min( qx[127:120] + rs1[7:0], 2^{8}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	2
qx	1	-
rs1	1	-

格式

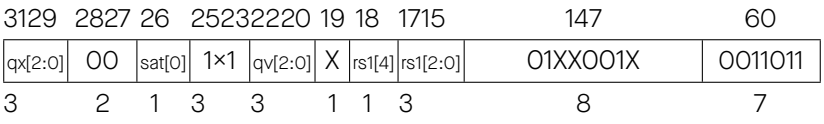


图 4.7-168. ESP.VSADDS.U8

4.7.2.169 ESP.VSAT.S16

语法

ESP.VSAT.S16 qz,qx,rs1,rs2

描述

此指令对 16 位有符号段执行矢量饱和。

操作

```
    min_t[15:0] = max( rs1[15:0], rs2[15:0] )
    max_t[15:0] = min( rs1[15:0], rs2[15:0] )

    qz[ 15:  0] = max( min( qx[ 15:  0], max_t[15:0]), min_t[15:0] )
    qz[ 31: 16] = max( min( qx[ 31: 16], maxt_[15:0]), min_t[15:0] )
    qz[ 47: 32] = max( min( qx[ 47: 32], max_t[15:0]), min_t[15:0] )
    qz[ 63: 48] = max( min( qx[ 63: 48], max_t[15:0]), min_t[15:0] )
    qz[ 79: 64] = max( min( qx[ 79: 64], max_t[15:0]), min_t[15:0] )
    qz[ 95: 80] = max( min( qx[ 95: 80], max_t[15:0]), min_t[15:0] )
    qz[111: 96] = max( min( qx[111: 96], max_t[15:0]), min_t[15:0] )
    qz[127:112] = max( min( qx[127:112], max_t[15:0]), min_t[15:0] )
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

格式

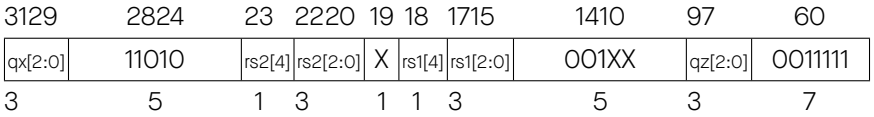


图 4.7-169. ESP.VSAT.S16

4.7.2.170 ESP.VSAT.S32

语法

ESP.VSAT.S32 qz,qx,rs1,rs2

描述

此指令对 32 位有符号段执行矢量饱和。

操作

```
    min_t[31:0] = max( rs1[31;0], rs2[31:0] )
    max_t[31:0] = min( rs1[31:0], rs2[31:0] )

    qz[ 31:  0] = max( min( qx[ 31:  0], max_t[31:0]), min_t[31:0] )
    qz[ 63: 32] = max( min( qx[ 63: 32], max_t[31:0]), min_t[31:0] )
    qz[ 95: 64] = max( min( qx[ 95: 64], max_t[31:0]), min_t[31:0] )
    qz[127: 96] = max( min( qx[127: 96], maxt_[31:0]), min_t[31:0] )
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

格式

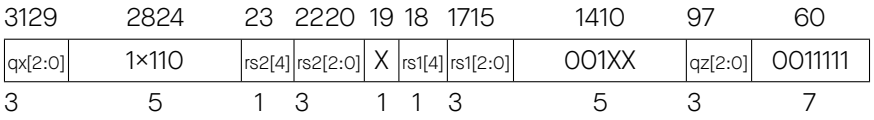


图 4.7-170. ESP.VSAT.S32

4.7.2.171 ESP.VSAT.S8

语法

ESP.VSAT.S8 qz,qx,rs1,rs2

描述

此指令对 8 位有符号段执行矢量饱和。

操作

```
min_t[7:0] = max( rs1[7:0], rs2[7:0] )
max_t[7:0] = min( rs1[7:0], rs2[7:0] )

qz[  7:  0] = max( min( qx[  7:  0], max_t[7:0]), min_t[7:0] )
qz[ 15:  8] = max( min( qx[ 15:  8], max_t[7:0]), min_t[7:0] )
qz[ 23: 16] = max( min( qx[ 23: 16], max_t[7:0]), min_t[7:0] )
qz[ 31: 24] = max( min( qx[ 31: 24], maxt_[7:0]), min_t[7:0] )
qz[ 39: 32] = max( min( qx[ 39: 32], max_t[7:0]), min_t[7:0] )
qz[ 47: 40] = max( min( qx[ 47: 40], max_t[7:0]), min_t[7:0] )
qz[ 55: 48] = max( min( qx[ 55: 48], max_t[7:0]), min_t[7:0] )
qz[ 63: 56] = max( min( qx[ 63: 56], max_t[7:0]), min_t[7:0] )
qz[ 71: 64] = max( min( qx[ 71: 64], max_t[7:0]), min_t[7:0] )
qz[ 79: 72] = max( min( qx[ 79: 72], max_t[7:0]), min_t[7:0] )
qz[ 87: 80] = max( min( qx[ 87: 80], max_t[7:0]), min_t[7:0] )
qz[ 95: 88] = max( min( qx[ 95: 88], max_t[7:0]), min_t[7:0] )
qz[103: 96] = max( min( qx[103: 96], max_t[7:0]), min_t[7:0] )
qz[111:104] = max( min( qx[111:104], max_t[7:0]), min_t[7:0] )
qz[119:112] = max( min( qx[119:112], max_t[7:0]), min_t[7:0] )
qz[127:120] = max( min( qx[127:120], max_t[7:0]), min_t[7:0] )
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

格式

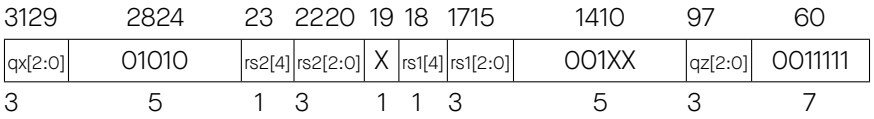


图 4.7-171. ESP.VSAT.S8

4.7.2.172 ESP.VSAT.U16

语法

ESP.VSAT.U16 qz,qx,rs1,rs2

描述

此指令对 16 位无符号段执行矢量饱和。

操作

```
    min_t[15:0] = max( rs1[15:0], rs2[15:0] )
    max_t[15:0] = min( rs1[15:0], rs2[15:0] )

    qz[ 15:  0] = max( min( qx[ 15:  0], max_t[15:0]), min_t[15:0] )
    qz[ 31: 16] = max( min( qx[ 31: 16], maxt_[15:0]), min_t[15:0] )
    qz[ 47: 32] = max( min( qx[ 47: 32], max_t[15:0]), min_t[15:0] )
    qz[ 63: 48] = max( min( qx[ 63: 48], max_t[15:0]), min_t[15:0] )
    qz[ 79: 64] = max( min( qx[ 79: 64], max_t[15:0]), min_t[15:0] )
    qz[ 95: 80] = max( min( qx[ 95: 80], max_t[15:0]), min_t[15:0] )
    qz[111: 96] = max( min( qx[111: 96], max_t[15:0]), min_t[15:0] )
    qz[127:112] = max( min( qx[127:112], max_t[15:0]), min_t[15:0] )
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

格式

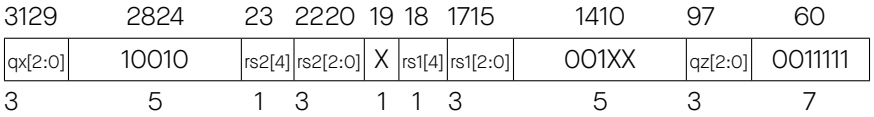


图 4.7-172. ESP.VSAT.U16

4.7.2.173 ESP.VSAT.U32

语法

ESP.VSAT.U32 qz,qx,rs1,rs2

描述

此指令对 32 位无符号段执行矢量饱和。

操作

```
    min_t[31:0] = max( rs1[31;0], rs2[31:0] )
    max_t[31:0] = min( rs1[31:0], rs2[31:0] )

    qz[ 31:  0] = max( min( qx[ 31:  0], max_t[31:0]), min_t[31:0] )
    qz[ 63: 32] = max( min( qx[ 63: 32], max_t[31:0]), min_t[31:0] )
    qz[ 95: 64] = max( min( qx[ 95: 64], max_t[31:0]), min_t[31:0] )
    qz[127: 96] = max( min( qx[127: 96], maxt_[31:0]), min_t[31:0] )
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

格式

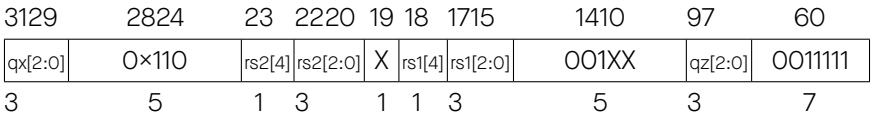


图 4.7-173. ESP.VSAT.U32

4.7.2.174 ESP.VSAT.U8

语法

ESP.VSAT.U8 qz,qx,rs1,rs2

描述

此指令对 8 位无符号段执行矢量饱和。

操作

```
    min_t[7:0] = max( rs1[7:0], rs2[7:0] )
    max_t[7:0] = min( rs1[7:0], rs2[7:0] )

    qz[  7:  0] = max( min( qx[  7:  0], max_t[7:0]), min_t[7:0] )
    qz[ 15:  8] = max( min( qx[ 15:  8], max_t[7:0]), min_t[7:0] )
    qz[ 23: 16] = max( min( qx[ 23: 16], max_t[7:0]), min_t[7:0] )
    qz[ 31: 24] = max( min( qx[ 31: 24], maxt_[7:0]), min_t[7:0] )
    qz[ 39: 32] = max( min( qx[ 39: 32], max_t[7:0]), min_t[7:0] )
    qz[ 47: 40] = max( min( qx[ 47: 40], max_t[7:0]), min_t[7:0] )
    qz[ 55: 48] = max( min( qx[ 55: 48], max_t[7:0]), min_t[7:0] )
    qz[ 63: 56] = max( min( qx[ 63: 56], max_t[7:0]), min_t[7:0] )
    qz[ 71: 64] = max( min( qx[ 71: 64], max_t[7:0]), min_t[7:0] )
    qz[ 79: 72] = max( min( qx[ 79: 72], max_t[7:0]), min_t[7:0] )
    qz[ 87: 80] = max( min( qx[ 87: 80], max_t[7:0]), min_t[7:0] )
    qz[ 95: 88] = max( min( qx[ 95: 88], max_t[7:0]), min_t[7:0] )
    qz[103: 96] = max( min( qx[103: 96], max_t[7:0]), min_t[7:0] )
    qz[111:104] = max( min( qx[111:104], max_t[7:0]), min_t[7:0] )
    qz[119:112] = max( min( qx[119:112], max_t[7:0]), min_t[7:0] )
    qz[127:120] = max( min( qx[127:120], max_t[7:0]), min_t[7:0] )
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
rs1	1	-
rs2	1	-

格式

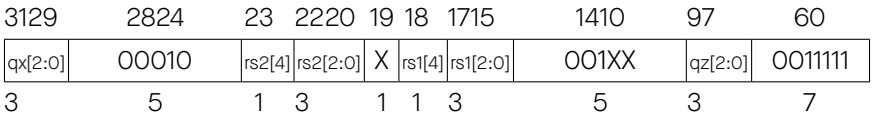


图 4.7-174. ESP.VSAT.U8

4.7.2.175 ESP.VSSUBS.S16

语法

ESP.VSSUBS.S16 qv,qx,rs1,sat

描述

此指令对 16 位数据执行有符号向量减法。

操作

```
qv[ 15:  0] = min( max( qx[ 15:  0] - rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 31: 16] = min( max( qx[ 31: 16] - rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 47: 32] = min( max( qx[ 47: 32] - rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 63: 48] = min( max( qx[ 63: 48] - rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 79: 64] = min( max( qx[ 79: 64] - rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[ 95: 80] = min( max( qx[ 95: 80] - rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[111: 96] = min( max( qx[111: 96] - rs1[15:0], -2^{15} ), 2^{15}-1 )
qv[127:112] = min( max( qx[127:112] - rs1[15:0], -2^{15} ), 2^{15}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
rs1	1	-

格式

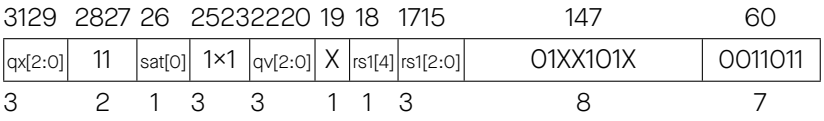


图 4.7-175. ESP.VSSUBS.S16

4.7.2.176 ESP.VSSUBS.S8

语法

ESP.VSSUBS.S8 qv,qx,rs1,sat

描述

此指令对 8 位数据执行有符号向量减法。

操作

```
qv[  7:  0] = min( max( qx[  7:  0] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 15:  8] = min( max( qx[ 15:  8] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 23: 16] = min( max( qx[ 23: 16] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 31: 24] = min( max( qx[ 31: 24] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 39: 32] = min( max( qx[ 39: 32] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 47: 40] = min( max( qx[ 47: 40] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 55: 48] = min( max( qx[ 55: 48] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 63: 56] = min( max( qx[ 63: 56] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 71: 64] = min( max( qx[ 71: 64] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 79: 72] = min( max( qx[ 79: 72] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 87: 80] = min( max( qx[ 87: 80] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[ 95: 88] = min( max( qx[ 95: 88] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[103: 96] = min( max( qx[103: 96] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[111:104] = min( max( qx[111:104] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[119:112] = min( max( qx[119:112] - rs1[7:0], -2^{7} ), 2^{7}-1 )
qv[127:120] = min( max( qx[127:120] - rs1[7:0], -2^{7} ), 2^{7}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
rs1	1	-

格式

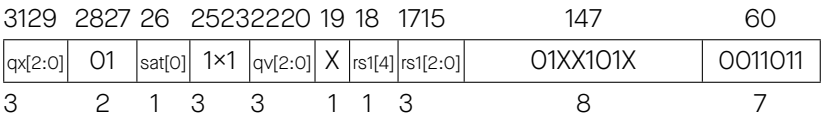


图 4.7-176. ESP.VSSUBS.S8

4.7.2.177 ESP.VSSUBS.U16

语法

ESP.VSSUBS.U16 qv,qx,rs1,sat

描述

此指令对 16 位数据执行无符号向量减法。

操作

```
qv[ 15:  0] = min( qx[ 15:  0] - rs1[15:0], 2^{16}-1 )
qv[ 31: 16] = min( qx[ 31: 16] - rs1[15:0], 2^{16}-1 )
qv[ 47: 32] = min( qx[ 47: 32] - rs1[15:0], 2^{16}-1 )
qv[ 63: 48] = min( qx[ 63: 48] - rs1[15:0], 2^{16}-1 )
qv[ 79: 64] = min( qx[ 79: 64] - rs1[15:0], 2^{16}-1 )
qv[ 95: 80] = min( qx[ 95: 80] - rs1[15:0], 2^{16}-1 )
qv[111: 96] = min( qx[111: 96] - rs1[15:0], 2^{16}-1 )
qv[127:112] = min( qx[127:112] - rs1[15:0], 2^{16}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
rs1	1	-

格式

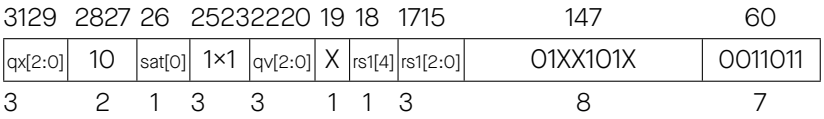


图 4.7-177. ESP.VSSUBS.U16

4.7.2.178 ESP.VSSUBS.U8

语法

ESP.VSSUBS.U8 qv,qx,rs1,sat

描述

此指令对 8 位数据执行有符号向量减法。

操作

```
    qv[  7:  0] = min( qx[  7:  0] - rs1[7:0], 2^{8}-1 )
    qv[ 15:  8] = min( qx[ 15:  8] - rs1[7:0], 2^{8}-1 )
    qv[ 23: 16] = min( qx[ 23: 16] - rs1[7:0], 2^{8}-1 )
    qv[ 31: 24] = min( qx[ 31: 24] - rs1[7:0], 2^{8}-1 )
    qv[ 39: 32] = min( qx[ 39: 32] - rs1[7:0], 2^{8}-1 )
    qv[ 47: 40] = min( qx[ 47: 40] - rs1[7:0], 2^{8}-1 )
    qv[ 55: 48] = min( qx[ 55: 48] - rs1[7:0], 2^{8}-1 )
    qv[ 63: 56] = min( qx[ 63: 56] - rs1[7:0], 2^{8}-1 )
    qv[ 71: 64] = min( qx[ 71: 64] - rs1[7:0], 2^{8}-1 )
    qv[ 79: 72] = min( qx[ 79: 72] - rs1[7:0], 2^{8}-1 )
    qv[ 87: 80] = min( qx[ 87: 80] - rs1[7:0], 2^{8}-1 )
    qv[ 95: 88] = min( qx[ 95: 88] - rs1[7:0], 2^{8}-1 )
    qv[103: 96] = min( qx[103: 96] - rs1[7:0], 2^{8}-1 )
    qv[111:104] = min( qx[111:104] - rs1[7:0], 2^{8}-1 )
    qv[119:112] = min( qx[119:112] - rs1[7:0], 2^{8}-1 )
    qv[127:120] = min( qx[127:120] - rs1[7:0], 2^{8}-1 )
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
rs1	1	-

格式

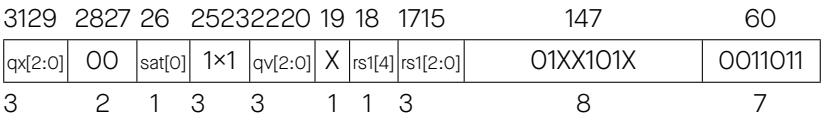


图 4.7-178. ESP.VSSUBS.U8

4.7.2.179 ESP.VSUB.S16

语法

ESP.VSUB.S16 qv,qx,qy,sat

描述

此指令对 16 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 8 个结果进行饱和处理，然后写入 qv 寄存器。

操作

```
qv[ 15:  0] = min(max(qx[ 15:  0] - qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] - qy[ 31: 16], -2^{15}), 2^{15}-1)
...
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-

格式

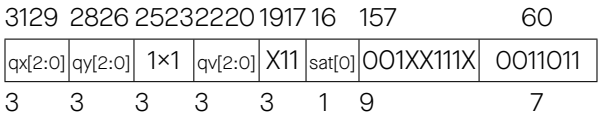


图 4.7-179. ESP.VSUB.S16

4.7.2.180 ESP.VSUB.S16.LD.INCP

语法

ESP.VSUB.S16.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 16 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 8 个结果被饱和，然后写入寄存器 qv。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[ 15:  0] = min(max(qx[ 15:  0] - qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] - qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

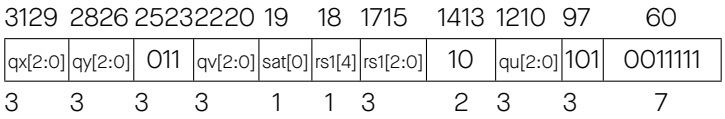


图 4.7-180. ESP.VSUB.S16.LD.INCP

4.7.2.181 ESP.VSUB.S16.ST.INCP

语法

ESP.VSUB.S16.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 16 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 8 个结果被饱和，然后写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[ 15:  0] = min(max(qx[ 15:  0] - qy[ 15:  0], -2^{15}), 2^{15}-1)
qv[ 31: 16] = min(max(qx[ 31: 16] - qy[ 31: 16], -2^{15}), 2^{15}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

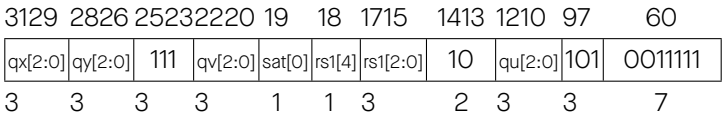


图 4.7-181. ESP.VSUB.S16.ST.INCP

4.7.2.182 ESP.VSUB.S32

语法

ESP.VSUB.S32 qv,qx,qy,sat

描述

此指令对 32 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 4 个结果进行饱和处理，然后写入寄存器 qv。

操作

```
qv[ 31:  0] = min(max(qx[ 31:  0] - qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] - qy[ 63: 32], -2^{31}), 2^{31}-1)
...
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-

格式

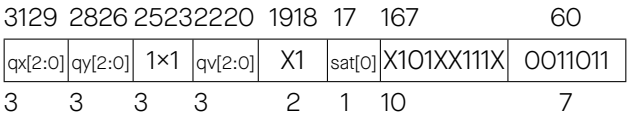


图 4.7-182. ESP.VSUB.S32

4.7.2.183 ESP.VSUB.S32.LD.INCP

语法

ESP.VSUB.S32.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 32 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 4 个结果被饱和，然后写入寄存器 qv。

在操作过程中，16 字节数据从内存加载到寄存器 qu。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[31: 0] = min(max(qx[31: 0] - qy[31: 0], -2^{31}), 2^{31}-1)
qv[63:32] = min(max(qx[63:32] - qy[63:32], -2^{31}), 2^{31}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

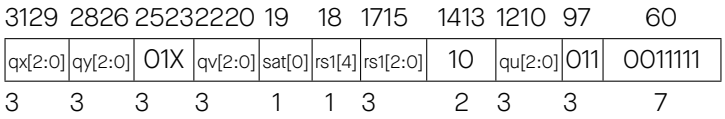


图 4.7-183. ESP.VSUB.S32.LD.INCP

4.7.2.184 ESP.VSUB.S32.ST.INCP

语法

ESP.VSUB.S32.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 32 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 4 个结果被饱和，然后写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储在内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[ 31:  0] = min(max(qx[ 31:  0] - qy[ 31:  0], -2^{31}), 2^{31}-1)
qv[ 63: 32] = min(max(qx[ 63: 32] - qy[ 63: 32], -2^{31}), 2^{31}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

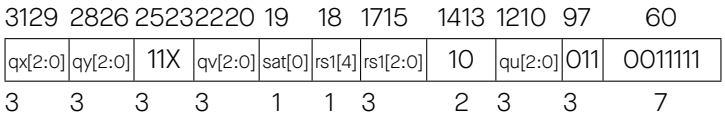


图 4.7-184. ESP.VSUB.S32.ST.INCP

4.7.2.185 ESP.VSUB.S8

语法

ESP.VSUB.S8 qv,qx,qy,sat

描述

此指令对 8 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 16 个结果进行饱和处理，然后写入寄存器 qv。

操作

```
qv[  7:  0] = min(max(qx[  7:  0] - qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] - qy[ 15:  8], -2^{7}), 2^{7}-1)
...
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-

格式

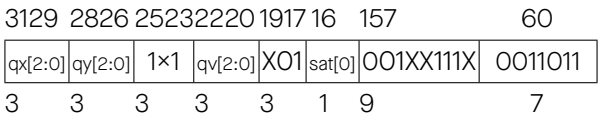


图 4.7-185. ESP.VSUB.S8

4.7.2.186 ESP.VSUB.S8.LD.INCP

语法

ESP.VSUB.S8.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 8 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 16 个结果被饱和，然后写入寄存器 qv。

在操作过程中，16 字节数据从内存加载到寄存器 qu 中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[  7:  0] = min(max(qx[  7:  0] - qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] - qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

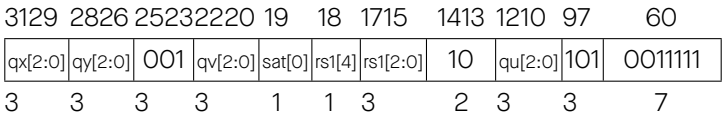


图 4.7-186. ESP.VSUB.S8.LD.INCP

4.7.2.187 ESP.VSUB.S8.ST.INCP

语法

ESP.VSUB.S8.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 8 位数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 16 个结果被饱和，然后写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[  7:  0] = min(max(qx[  7:  0] - qy[  7:  0], -2^{7}), 2^{7}-1)
qv[ 15:  8] = min(max(qx[ 15:  8] - qy[ 15:  8], -2^{7}), 2^{7}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

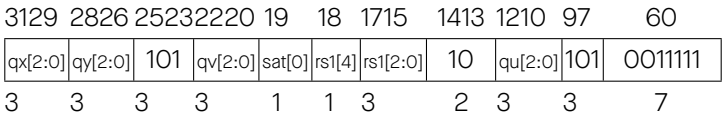


图 4.7-187. ESP.VSUB.S8.ST.INCP

4.7.2.188 ESP.VSUB.U16

语法

ESP.VSUB.U16 qv,qx,qy,sat

描述

此指令对 16 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 8 个结果进行饱和处理，然后写入 qv 寄存器。

操作

```
qv[ 15:  0] = min( qx[ 15:  0] - qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min( qx[ 31: 16] - qy[ 31: 16], 2^{16}-1)
...
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-

格式

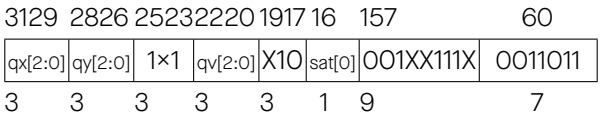


图 4.7-188. ESP.VSUB.U16

4.7.2.189 ESP.VSUB.U16.LD.INCP

语法

ESP.VSUB.U16.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 16 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 8 个结果被饱和处理，然后写入寄存器 qv。

在操作过程中，16 字节数据从内存加载到寄存器 qu。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[ 15:  0] = min(qx[ 15:  0] - qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] - qy[ 31: 16], 2^{16}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

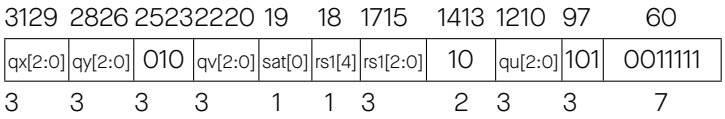


图 4.7-189. ESP.VSUB.U16.LD.INCP

4.7.2.190 ESP.VSUB.U16.ST.INCP

语法

ESP.VSUB.U16.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 16 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 8 个结果被饱和，然后写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[ 15:  0] = min(qx[ 15:  0] - qy[ 15:  0], 2^{16}-1)
qv[ 31: 16] = min(qx[ 31: 16] - qy[ 31: 16], 2^{16}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

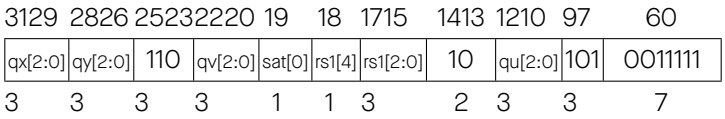


图 4.7-190. ESP.VSUB.U16.ST.INCP

4.7.2.191 ESP.VSUB.U32

语法

ESP.VSUB.U32 qv,qx,qy,sat

描述

此指令对 32 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 4 个结果进行饱和处理，然后写入寄存器 qv。

操作

```
qv[ 31:  0] = min(qx[ 31:  0] - qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] - qy[ 63: 32], 2^{32}-1)
...
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-

格式

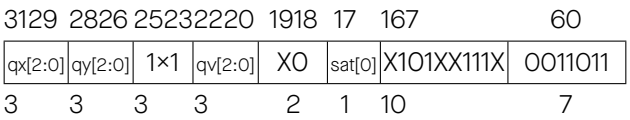


图 4.7-191. ESP.VSUB.U32

4.7.2.192 ESP.VSUB.U32.LD.INCP

语法

ESP.VSUB.U32.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 32 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 4 个结果被饱和，然后写入寄存器 qv。

在操作过程中，16 字节的数据从内存加载到寄存器 qu。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[31: 0] = min(qx[31: 0] - qy[31: 0], 2^{32}-1)
qv[63:32] = min(qx[63:32] - qy[63:32], 2^{32}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

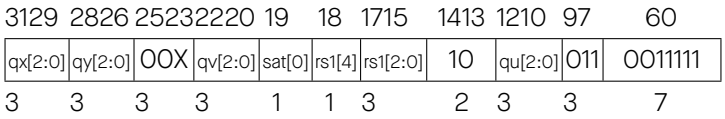


图 4.7-192. ESP.VSUB.U32.LD.INCP

4.7.2.193 ESP.VSUB.U32.ST.INCP

语法

ESP.VSUB.U32.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 32 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 4 个结果被饱和，然后写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[ 31:  0] = min(qx[ 31:  0] - qy[ 31:  0], 2^{32}-1)
qv[ 63: 32] = min(qx[ 63: 32] - qy[ 63: 32], 2^{32}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

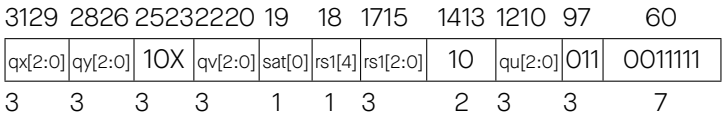


图 4.7-193. ESP.VSUB.U32.ST.INCP

4.7.2.194 ESP.VSUB.U8

语法

ESP.VSUB.U8 qv,qx,qy,sat

描述

此指令对 8 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 16 个结果进行饱和处理，然后写入寄存器 qv。

操作

```
qv[  7:  0] = min(qx[  7:  0] - qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] - qy[ 15:  8], 2^{8}-1)
...
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-

格式

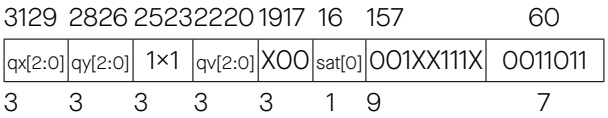


图 4.7-194. ESP.VSUB.U8

4.7.2.195 ESP.VSUB.U8.LD.INCP

语法

ESP.VSUB.U8.LD.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 8 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 16 个结果被饱和处理，然后写入寄存器 qv。

在操作过程中，16 字节数据从内存加载到寄存器 qu。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[  7:  0] = min(qx[  7:  0] - qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] - qy[ 15:  8], 2^{8}-1)
...

qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	-	2
rs1	1	1

格式

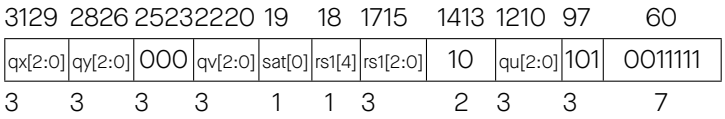


图 4.7-195. ESP.VSUB.U8.LD.INCP

4.7.2.196 ESP.VSUB.U8.ST.INCP

语法

ESP.VSUB.U8.ST.INCP qu,rs1,qv,qx,qy,sat

描述

此指令对 8 位无符号数据执行向量减法。寄存器 qx 和 qy 分别是被减数和减数。然后，从计算中得到的 16 个结果被饱和，然后写入寄存器 qv。

在操作过程中，寄存器 qu 中的值被存储到内存中。访问后，寄存器 rs1 中的值增加了 16。

操作

```
qv[  7:  0] = min(qx[  7:  0] - qy[  7:  0], 2^{8}-1)
qv[ 15:  8] = min(qx[ 15:  8] - qy[ 15:  8], 2^{8}-1)
...

qu[127:0] => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qv	-	1
qx	1	-
qy	1	-
qu	2	-
rs1	1	1

格式

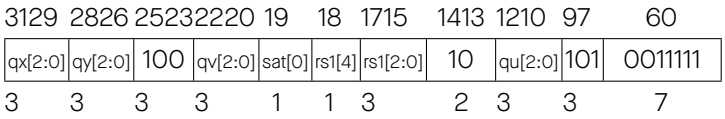


图 4.7-196. ESP.VSUB.U8.ST.INCP

4.7.2.197 ESP.ADDX2

语法

ESP.ADDX2 rd,rs1,rs2

描述

此指令将 `rs2` 向左移动 1 位并将其加到 `rs1`。

操作

```
rd = rs1 + rs2<<1
```

调度

操作数	读流水级数	写流水级数
rd	-	1
rs1	1	-
rs2	1	-

格式

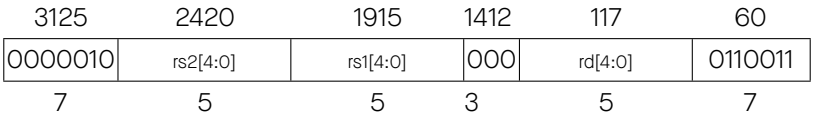


图 4.7-197. ESP.ADDX2

4.7.2.198 ESP.ADDX4

语法

ESP.ADDX4 rd,rs1,rs2

描述

此指令将 `rs2` 向左移动 2 位并将其加到 `rs1`。

操作

```
rd = rs1 + rs2<<2
```

调度

操作数	读流水级数	写流水级数
rd	-	1
rs1	1	-
rs2	1	-

格式

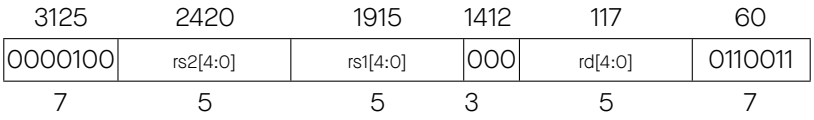


图 4.7-198. ESP.ADDX4

4.7.2.199 ESP.SAT

语法

ESP.SAT rsd,rs0,rs1

描述

此指令为 rsd 执行饱和处理。

操作

```
    min_t[31:0] = min( rs1[31:0], rs0[31:0] )
    max_t[31:0] = max( rs1[31:0], rs0[31:0] )

    rsd[31:0] = max( min( rsd[31:0], max_t[31:0]), min_t[31:0] )
```

调度

操作数	读流水级数	写流水级数
rsd	1	1
rs0	1	-
rs1	1	-

格式

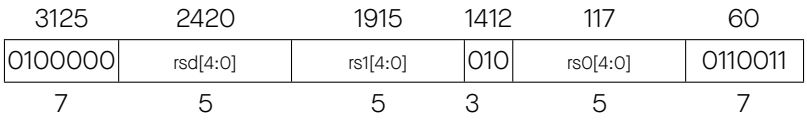


图 4.7-199. ESP.SAT

4.7.2.200 ESP.SUBX2

语法

ESP.SUBX2 rd,rs1,rs2

描述

此指令将 `rs2` 向左移动 1 位并从 `rs1` 中减去它。

操作

```
rd = rs1 - rs2<<1
```

调度

操作数	读流水级数	写流水级数
rd	-	1
rs1	1	-
rs2	1	-

格式

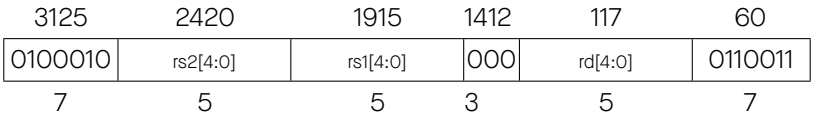


图 4.7-200. ESP.SUBX2

4.7.2.201 ESP.SUBX4

语法

ESP.SUBX4 rd,rs1,rs2

描述

此指令将 `rs2` 向左移动 2 位并从 `rs1` 中减去它。

操作

```
rd = rs1 - rs2<<2
```

调度

操作数	读流水级数	写流水级数
rd	-	1
rs1	1	-
rs2	1	-

格式

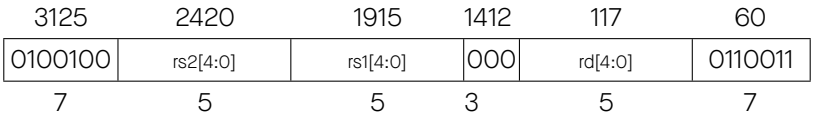


图 4.7-201. ESP.SUBX4

4.7.2.202 ESP.ANDQ

语法

ESP.ANDQ qz,qx,qy

描述

此指令对寄存器 qx 和 qy 执行位与操作，并将逻辑操作的结果写入寄存器 qz。

操作

```
qz = qx & qy
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

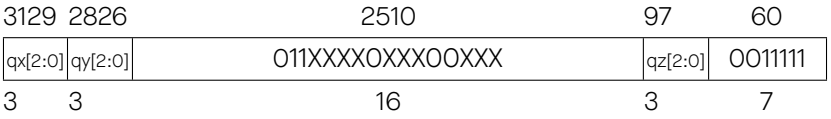


图 4.7-202. ESP.ANDQ

4.7.2.203 ESP.NOTQ

语法

ESP.NOTQ qz,qx

描述

此指令对寄存器 qx 执行位非操作，并将结果写入寄存器 qz。

操作

```
qz = ~qx
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-

格式

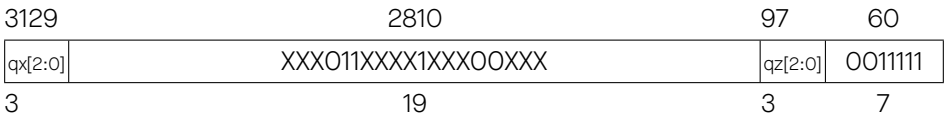


图 4.7-203. ESP.NOTQ

4.7.2.204 ESP.ORQ

语法

ESP.ORQ qz,qx,qy

描述

此指令对寄存器 qx 和 qy 执行位或操作，并将逻辑操作的结果写入寄存器 qz。

操作

```
qz = qx | qy
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

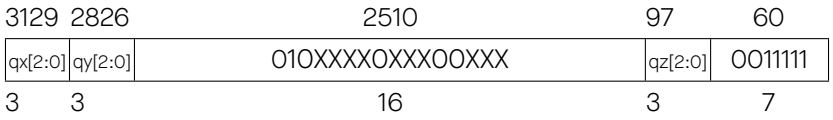


图 4.7-204. ESP.ORQ

4.7.2.205 ESP.XORQ

语法

ESP.XORQ qz,qx,qy

描述

此指令对寄存器 qx 和 qy 执行位异或操作，并将逻辑操作的结果写入寄存器 qz。

操作

```
qz = qx ^ qy
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

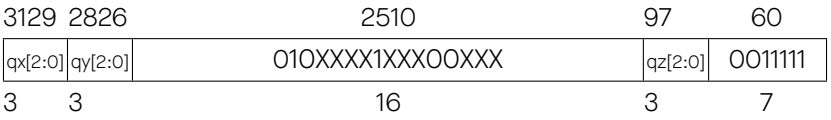


图 4.7-205. ESP.XORQ

4.7.2.206 ESP.VCMP.EQ.S16

语法

ESP.VCMP.EQ.S16 qz,qx,qy

描述

此指令比较 16 位向量数据。它比较寄存器qx和qy中的八个 16 位数据段的数值。如果值相等，它将 0xFFFF 写入寄存器qz中的相应 16 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]==qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]==qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]==qy[127:112]) ? 0xFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

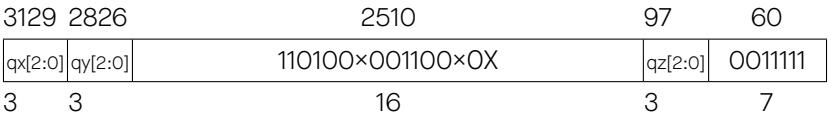


图 4.7-206. ESP.VCMP.EQ.S16

4.7.2.207 ESP.VCMP.EQ.S32

语法

ESP.VCMP.EQ.S32 qz,qx,qy

描述

此指令比较 32 位向量数据。它比较寄存器qx和qy中的四个 32 位数据段的数值。如果值相等，它将 0xFFFFFFFF 写入qz寄存器中相应的 32 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]==qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]==qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]==qy[127: 96]) ? 0xFFFFFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

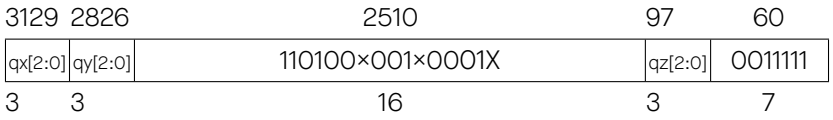


图 4.7-207. ESP.VCMP.EQ.S32

4.7.2.208 ESP.VCMP.EQ.S8

语法

ESP.VCMP.EQ.S8 qz,qx,qy

描述

此指令比较 8 位向量数据。它比较寄存器qx和qy中的 16 个 8 位数据段的数值。如果值相等，它将 0xFF 写入寄存器qz中相应的 8 位数据段。否则，它将 0 写入该段。

操作

```
qz[  7:  0] = (qx[  7:  0]==qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]==qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]==qy[127:120]) ? 0xFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

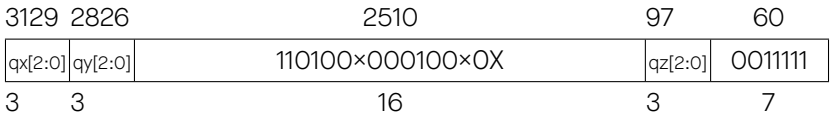


图 4.7-208. ESP.VCMP.EQ.S8

4.7.2.209 ESP.VCMP.EQ.U16

语法

ESP.VCMP.EQ.U16 qz,qx,qy

描述

这个指令比较 16 位向量数据。它比较寄存器qx和qy中的八个 16 位数据段的数值。如果这些值相等，它将 0xFFFF 写入寄存器qz中相应的 16 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]==qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]==qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]==qy[127:112]) ? 0xFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

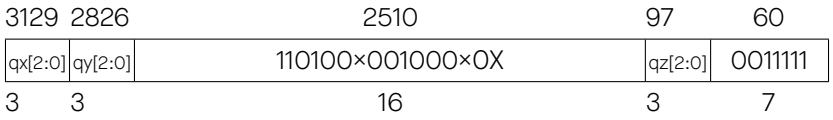


图 4.7-209. ESP.VCMP.EQ.U16

4.7.2.210 ESP.VCMP.EQ.U32

语法

ESP.VCMP.EQ.U32 qz,qx,qy

描述

此指令比较 32 位向量数据。它比较寄存器qx和qy中的四个 32 位数据段的数值。如果值相等，它将 0xFFFFFFFF 写入寄存器qz中的相应 32 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]==qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]==qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]==qy[127: 96]) ? 0xFFFFFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

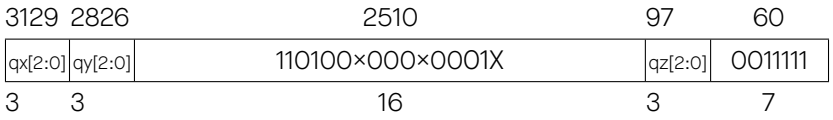


图 4.7-210. ESP.VCMP.EQ.U32

4.7.2.211 ESP.VCMP.EQ.U8

语法

ESP.VCMP.EQ.U8 qz,qx,qy

描述

此指令比较 8 位向量数据。它比较寄存器qx和qy中的 16 个 8 位数据段的数值。如果这些值相等，它将 0xFF 写入qz寄存器中相应的 8 位数据段。否则，它将 0 写入该段。

操作

```
qz[  7:  0] = (qx[  7:  0]==qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]==qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]==qy[127:120]) ? 0xFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

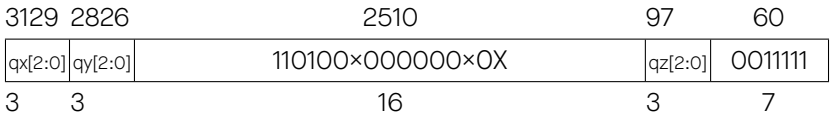


图 4.7-211. ESP.VCMP.EQ.U8

4.7.2.212 ESP.VCMP.GT.S16

语法

ESP.VCMP.GT.S16 qz,qx,qy

描述

此指令比较 16 位向量数据。它比较寄存器qx和qy中的八个 16 位数据段的数值。如果前者大于后者，它将在寄存器qz的相应 16 位数据段中写入 0xFFFF。否则，它将在该段中写入 0。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]>qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]>qy[127:112]) ? 0xFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

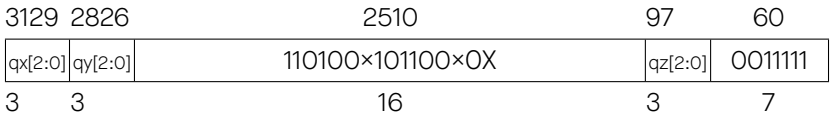


图 4.7-212. ESP.VCMP.GT.S16

4.7.2.213 ESP.VCMP.GT.S32

语法

ESP.VCMP.GT.S32 qz,qx,qy

描述

此指令比较 32 位有符号向量数据。它比较寄存器qx和qy中的四个 32 位数据段的数值。如果前者大于后者，它将 0xFFFFFFFF 写入寄存器qz中相应的 32 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]>qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]>qy[127: 96]) ? 0xFFFFFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

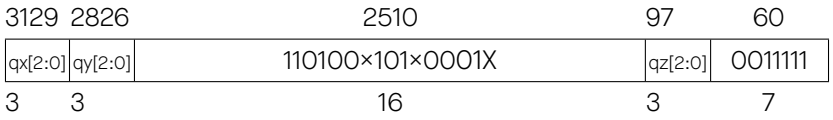


图 4.7-213. ESP.VCMP.GT.S32

4.7.2.214 ESP.VCMP.GT.S8

语法

ESP.VCMP.GT.S8 qz,qx,qy

描述

此指令比较 8 位有符号向量数据。它比较寄存器qx和qy中的 16 个 8 位数据段的数值。如果前者大于后者，它将在寄存器qz的相应 8 位数据段中写入 0xFF。否则，它将在该段中写入 0。

操作

```
qz[  7:  0] = (qx[  7:  0]>qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]>qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]>qy[127:120]) ? 0xFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

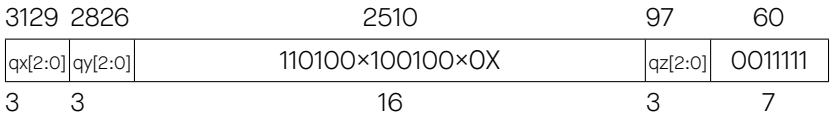


图 4.7-214. ESP.VCMP.GT.S8

4.7.2.215 ESP.VCMP.GT.U16

语法

ESP.VCMP.GT.U16 qz,qx,qy

描述

此指令比较 16 位无符号向量数据。它比较寄存器qx和qy中的八个 16 位数据段的数值。如果前者大于后者，它将 0xFFFF 写入寄存器qz中相应的 16 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]>qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]>qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]>qy[127:112]) ? 0xFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

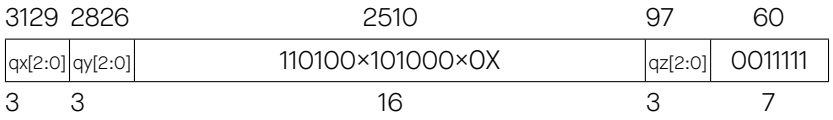


图 4.7-215. ESP.VCMP.GT.U16

4.7.2.216 ESP.VCMP.GT.U32

语法

ESP.VCMP.GT.U32 qz,qx,qy

描述

此指令比较 32 位无符号向量数据。它比较寄存器qx和qy中的四个 32 位数据段的数值。如果前者大于后者，它将 0xFFFFFFFF 写入寄存器qz中相应的 32 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]>qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]>qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]>qy[127: 96]) ? 0xFFFFFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

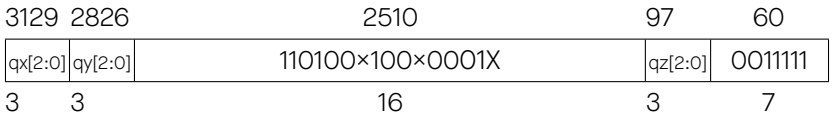


图 4.7-216. ESP.VCMP.GT.U32

4.7.2.217 ESP.VCMP.GT.U8

语法

ESP.VCMP.GT.U8 qz,qx,qy

描述

本指令比较 8 位无符号向量数据。它比较寄存器qx和qy中的 16 个 8 位数据段的数值。如果前者大于后者，它将在寄存器qz中的相应 8 位数据段写入 0xFF。否则，它在该段中写入 0。

操作

```
qz[  7:  0] = (qx[  7:  0]>qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]>qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]>qy[127:120]) ? 0xFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

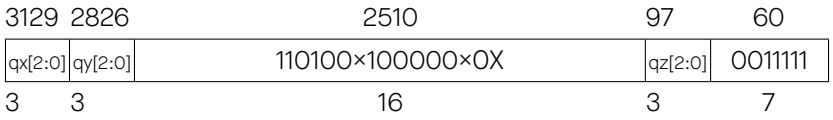


图 4.7-217. ESP.VCMP.GT.U8

4.7.2.218 ESP.VCMP.LT.S16

语法

ESP.VCMP.LT.S16 qz,qx,qy

描述

此指令比较 16 位有符号向量数据。它比较寄存器qx和qy中的八个 16 位数据段的数值。如果前者小于后者，它将 0xFFFF 写入寄存器qz中相应的 16 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]<qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]<qy[127:112]) ? 0xFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

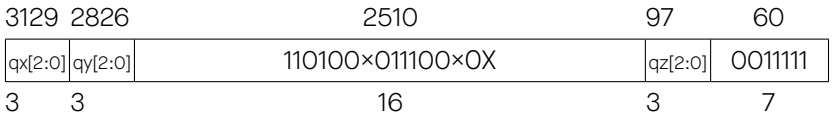


图 4.7-218. ESP.VCMP.LT.S16

4.7.2.219 ESP.VCMP.LT.S32

语法

ESP.VCMP.LT.S32 qz,qx,qy

描述

此指令比较 32 位有符号向量数据。它对寄存器qx和qy中的四个 32 位数据段的数值进行比较。如果前者小于后者，它将 0xFFFFFFFF 写入寄存器qz中相应的 32 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]<qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]<qy[127: 96]) ? 0xFFFFFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

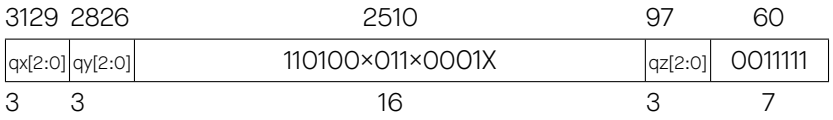


图 4.7-219. ESP.VCMP.LT.S32

4.7.2.220 ESP.VCMP.LT.S8

语法

ESP.VCMP.LT.S8 qz,qx,qy

描述

此指令比较 8 位有符号向量数据。它比较寄存器qx和qy中的 16 个 8 位数据段的数值。如果前者小于后者，它将在寄存器qz中的相应 8 位数据段写入 0xFF。否则，它在该段中写入 0。

操作

```
qz[  7:  0] = (qx[  7:  0]<qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]<qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]<qy[127:120]) ? 0xFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

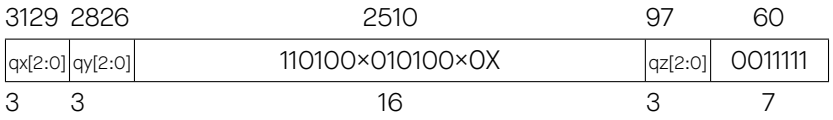


图 4.7-220. ESP.VCMP.LT.S8

4.7.2.221 ESP.VCMP.LT.U16

语法

ESP.VCMP.LT.U16 qz,qx,qy

描述

本指令比较 16 位无符号向量数据。它比较寄存器qx和qy中的八个 16 位数据段的数值。如果前者小于后者，它将在寄存器qz中的相应 16 位数据段写入 0xFFFF。否则，它在该段中写入 0。

操作

```
qz[ 15:  0] = (qx[ 15:  0]<qy[ 15:  0]) ? 0xFFFF : 0
qz[ 31: 16] = (qx[ 31: 16]<qy[ 31: 16]) ? 0xFFFF : 0
...
qz[127:112] = (qx[127:112]<qy[127:112]) ? 0xFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

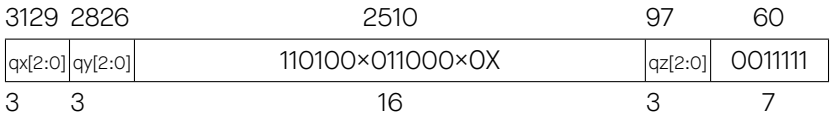


图 4.7-221. ESP.VCMP.LT.U16

4.7.2.222 ESP.VCMP.LT.U32

语法

ESP.VCMP.LT.U32 qz,qx,qy

描述

此指令比较 32 位无符号向量数据。它比较寄存器qx和qy中的四个 32 位数据段的数值。如果前者小于后者，它将 0xFFFFFFFF 写入寄存器qz中的相应 32 位数据段。否则，它将 0 写入该段。

操作

```
qz[ 31:  0] = (qx[ 31:  0]<qy[ 31:  0]) ? 0xFFFFFFFF : 0
qz[ 63: 32] = (qx[ 63: 32]<qy[ 63: 32]) ? 0xFFFFFFFF : 0
...
qz[127: 96] = (qx[127: 96]<qy[127: 96]) ? 0xFFFFFFFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

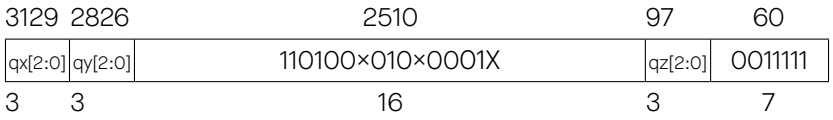


图 4.7-222. ESP.VCMP.LT.U32

4.7.2.223 ESP.VCMP.LT.U8

语法

ESP.VCMP.LT.U8 qz,qx,qy

描述

此指令比较 8 位无符号向量数据。它比较寄存器qx和qy中的 16 个 8 位数据段的数值。如果前者小于后者，它将在寄存器qz中的相应 8 位数据段写入 0xFF。否则，它在该段中写入 0。

操作

```
qz[  7:  0] = (qx[  7:  0]<qy[  7:  0]) ? 0xFF : 0
qz[ 15:  8] = (qx[ 15:  8]<qy[ 15:  8]) ? 0xFF : 0
...
qz[127:120] = (qx[127:120]<qy[127:120]) ? 0xFF : 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qx	1	-
qy	1	-

格式

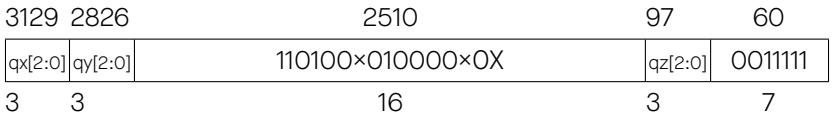


图 4.7-223. ESP.VCMP.LT.U8

4.7.2.224 ESP.MOV.S16.QACC

语法

ESP.MOV.S16.QACC qu

描述

此指令将寄存器 qs 中的 16 位数据的 8 个段有符号扩展到 64 位，并将结果写入特殊寄存器 QACC_H 和 QACC_L。

操作

```
QACC_L[ 63:  0] = {48{qu[15]}, qu[ 15:  0]}
QACC_L[127: 64] = {48{qu[31]}, qu[ 31: 16]}
QACC_L[191:128] = {48{qu[47]}, qu[ 47: 32]}
QACC_L[255:192] = {48{qu[63]}, qu[ 63: 48]}
QACC_H[ 39:  0] = {48{qu[79]}, qu[ 79: 64]}
QACC_H[ 79: 40] = {48{qu[95]}, qu[ 95: 80]}
QACC_H[119: 80] = {48{qu[111]}, qu[111: 96]}
QACC_H[159:120] = {48{qu[127]}, qu[127:112]}
```

调度

操作数	读流水级数	写流水级数
qu	2	-
QACC_H	-	2
QACC_L	-	2

格式



图 4.7-224. ESP.MOV.S16.QACC

4.7.2.225 ESP.MOV.S8.QACC

语法

ESP.MOV.S8.QACC qu

描述

此指令将寄存器 qs 中的 16 段 8 位数据符号扩展到 32 位，并将结果写入特殊寄存器 QACC_H 和 QACC_L。

操作

```
QACC_L[ 31:  0] = {24{qu[7]}, qu[  7:  0]}
QACC_L[ 63: 32] = {24{qu[15]}, qu[ 15:  8]}
QACC_L[ 95: 64] = {24{qu[23]}, qu[ 23: 16]}
...
QACC_L[255:224] = {24{qu[63]}, qu[ 63: 56]}
QACC_H[ 31:  0] = {24{qu[71]}, qu[ 71: 64]}
QACC_H[ 63: 32] = {24{qu[79]}, qu[ 79: 72]}
QACC_H[ 95: 64] = {24{qu[87]}, qu[ 87: 80]}
...
QACC_H[255:224] = {24{qu[127]}, qu[127:120]}
```

调度

操作数	读流水级数	写流水级数
qu	2	-
QACC_H	-	2
QACC_L	-	2

格式

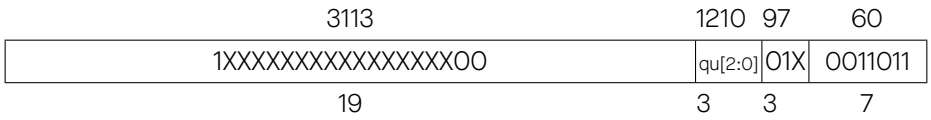


图 4.7-225. ESP.MOV.S8.QACC

4.7.2.226 ESP.MOV.U16.QACC

语法

ESP.MOV.U16.QACC qu

描述

此指令将寄存器 qs 中的 16 位数据的 8 个段进行零扩展至 64 位，并将结果写入特殊寄存器 QACC_H 和 QACC_L。

操作

```
QACC_L[ 63:  0] = {48{0}, qu[ 15:  0]}
QACC_L[127: 64] = {48{0}, qu[ 31: 16]}
QACC_L[191:128] = {48{0}, qu[ 47: 32]}
QACC_L[255:192] = {48{0}, qu[ 63: 48]}
QACC_H[ 39:  0] = {48{0}, qu[ 79: 64]}
QACC_H[ 79: 40] = {48{0}, qu[ 95: 80]}
QACC_H[119: 80] = {48{0}, qu[111: 96]}
QACC_H[159:120] = {48{0}, qu[127:112]}
```

调度

操作数	读流水级数	写流水级数
qu	2	-
QACC_H	-	2
QACC_L	-	2

格式



图 4.7-226. ESP.MOV.U16.QACC

4.7.2.227 ESP.MOV.U8.QACC

语法

ESP.MOV.U8.QACC qu

描述

此指令将寄存器 qs 中的 16 段 8 位数据零扩展到 32 位，并将结果写入特殊寄存器 QACC_H 和 QACC_L。

操作

```
QACC_L[ 31:  0] = {24{0}, qu[  7:  0]}
QACC_L[ 63: 32] = {24{0}, qu[ 15:  8]}
QACC_L[ 95: 64] = {24{0}, qu[ 23: 16]}
...
QACC_L[255:224] = {24{0}, qu[ 63: 56]}
QACC_H[ 31:  0] = {24{0}, qu[ 71: 64]}
QACC_H[ 63: 32] = {24{0}, qu[ 79: 72]}
QACC_H[ 95: 64] = {24{0}, qu[ 87: 80]}
...
QACC_H[255:224] = {24{0}, qu[127:120]}
```

调度

操作数	读流水级数	写流水级数
qu	2	-
QACC_H	-	2
QACC_L	-	2

格式



图 4.7-227. ESP.MOV.U8.QACC

4.7.2.228 ESP.MOVI.16.A

语法

ESP.MOVI.16.A qy,rd,sel16

描述

此指令从 qy 的 sel 索引中提取 16 位到rd寄存器。

操作

```
if sel == 0:    rd[31 :  0] = { 16'b0, qy[15 :  0] }
else if sel==1: rd[31 :  0] = { 16'b0, qy[31 : 16] }
else if sel==2: rd[31 :  0] = { 16'b0, qy[47 : 32] }
else if sel==3: rd[31 :  0] = { 16'b0, qy[63 : 48] }
else if sel==4: rd[31 :  0] = { 16'b0, qy[79 : 64] }
else if sel==5: rd[31 :  0] = { 16'b0, qy[95 : 80] }
else if sel==6: rd[31 :  0] = { 16'b0, qy[111: 96] }
else if sel==7: rd[31 :  0] = { 16'b0, qy[127:112] }
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qy	1	-

格式

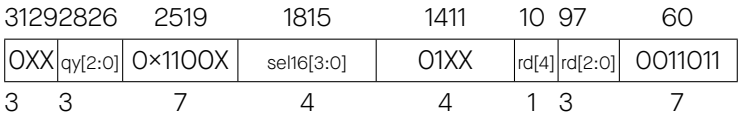


图 4.7-228. ESP.MOVI.16.A

4.7.2.229 ESP.MOVI.16.Q

语法

ESP.MOVI.16.Q qy,rs1,sel16

描述

此指令将 16 位插入到qy的 sel 索引中，来源于rs1寄存器。

操作

```
if sel == 0:        qy[31 :  0] = { qy[31 : 16], rs1[15:0] }
else if sel == 1:   qy[31 :  0] = { rs1[15:0], qy[15 :  0] }
else if sel == 2:   qy[63 : 32] = { qy[63 : 48], rs1[15:0] }
else if sel == 3:   qy[63 : 48] = { rs1[15:0], qy[47 : 32] }
else if sel == 4:   qy[95 : 64] = { qy[95 : 80], rs1[15:0] }
else if sel == 5:   qy[95 : 64] = { rs1[15:0], qy[79 : 64] }
else if sel == 6:   qy[127: 96] = { qy[127:112], rs1[15:0] }
else if sel == 7:   qy[127: 96] = { rs1[15:0], qy[111: 96] }
```

调度

操作数	读流水级数	写流水级数
qy	1	1
rs1	1	-

格式

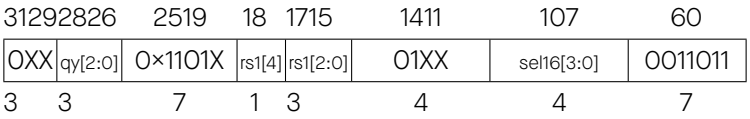


图 4.7-229. ESP.MOVI.16.Q

4.7.2.230 ESP.MOVI.32.A

语法

ESP.MOVI.32.A qy,rd,sel4

描述

此指令从 qv 的 sel 索引中提取 32 位到rd寄存器。

操作

```
if sel == 0:    rd[31 :  0] = qy[31 :  0]
else if sel==1: rd[31 :  0] = qy[63 : 32]
else if sel==2: rd[31 :  0] = qy[95 : 64]
else if sel==3: rd[31 :  0] = qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qy	1	-

格式

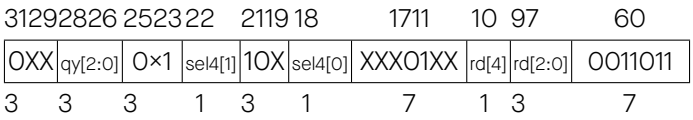


图 4.7-230. ESP.MOVI.32.A

4.7.2.231 ESP.MOVI.32.Q

语法

ESP.MOVI.32.Q qy,rs1,sel4

描述

此指令将 32 位插入到qy的 sel 索引中，来源于rs1寄存器。

操作

```
if sel == 0:        qy[31 :  0] = rs1[31:0]
else if sel == 1:   qy[63 : 32] = rs1[31:0]
else if sel == 2:   qy[95 : 64] = rs1[31:0]
else if sel == 3:   qy[127: 96] = rs1[31:0]
```

调度

操作数	读流水级数	写流水级数
qy	1	1
rs1	1	-

格式

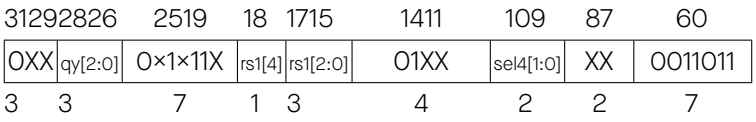


图 4.7-231. ESP.MOVI.32.Q

4.7.2.232 ESP.MOVI.8.A

语法

ESP.MOVI.8.A qy,rd,sel16

描述

此指令从 qv 的 sel 索引中提取 8 位到rd寄存器。

操作

```
if sel == 0:    rd[31 :  0] = { 24' b0, qy[7  :  0] }
else if sel==1: rd[31 :  0] = { 24' b0, qy[15 :  8] }
else if sel==2: rd[31 :  0] = { 24' b0, qy[23 : 16] }
else if sel==3: rd[31 :  0] = { 24' b0, qy[31 : 24] }
else if sel==4: rd[31 :  0] = { 24' b0, qy[39 : 32] }
else if sel==5: rd[31 :  0] = { 24' b0, qy[47 : 40] }
else if sel==6: rd[31 :  0] = { 24' b0, qy[55 : 48] }
else if sel==7: rd[31 :  0] = { 24' b0, qy[63 : 56] }
else if sel==8: rd[31 :  0] = { 24' b0, qy[71 : 64] }
else if sel==9: rd[31 :  0] = { 24' b0, qy[79 : 72] }
else if sel==10:rd[31 :  0] = { 24' b0, qy[87 : 80] }
else if sel==11:rd[31 :  0] = { 24' b0, qy[95 : 88] }
else if sel==12:rd[31 :  0] = { 24' b0, qy[103: 96] }
else if sel==13:rd[31 :  0] = { 24' b0, qy[111:104] }
else if sel==14:rd[31 :  0] = { 24' b0, qy[119:112] }
else if sel==15:rd[31 :  0] = { 24' b0, qy[127:120] }
```

调度

操作数	读流水级数	写流水级数
rd	-	2
qy	1	-

格式

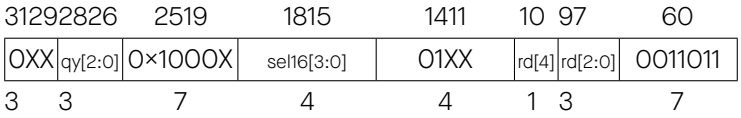


图 4.7-232. ESP.MOVI.8.A

4.7.2.233 ESP.MOVI.8.Q

语法

ESP.MOVI.8.Q qy,rs1,sel16

描述

此指令将 8 位插入到qy的 sel 索引中，来源于rs1寄存器。

操作

```
if sel == 0:        qy[31 :  0] = { qy[31 :  8], rs1[7:0] }
else if sel == 1:   qy[31 :  0] = { qy[31 : 16], rs1[7:0], qy[7  :  0] }
else if sel == 2:   qy[31 :  0] = { qy[31 :  0], rs1[7:0], qy[15 :  0] }
else if sel == 3:   qy[31 : 24] = { rs1[7:0], qy[23 :  0] }
else if sel == 4:   qy[63 : 32] = { qy[63 : 40], rs1[7:0] }
else if sel == 5:   qy[63 : 32] = { qy[63 : 48], rs1[7:0], qy[39 : 32] }
else if sel == 6:   qy[63 : 32] = { qy[63 : 56], rs1[7:0], qy[47 : 32] }
else if sel == 7:   qy[63 : 32] = { rs1[7:0], qy[55 : 32] }
else if sel == 8:   qy[95 : 64] = { qy[95 : 72], rs1[7:0] }
else if sel == 9:   qy[95 : 64] = { qy[95 : 80], rs1[7:0], qy[71 : 64] }
else if sel == 10:  qy[95 : 64] = { qy[95 : 88], rs1[7:0], qy[79 : 64] }
else if sel == 11:  qy[95 : 64] = { rs1[7:0], qy[87 : 64] }
else if sel == 12:  qy[127: 96] = { qy[127:104], rs1[7:0] }
else if sel == 13:  qy[127: 96] = { qy[127:112], rs1[7:0], qy[103: 96] }
else if sel == 14:  qy[127: 96] = { qy[127:120], rs1[7:0], qy[111: 96] }
else if sel == 15:  qy[127: 96] = { rs1[7:0], qy[119: 96] }
```

调度

操作数	读流水级数	写流水级数
qy	1	1
rs1	1	-

格式

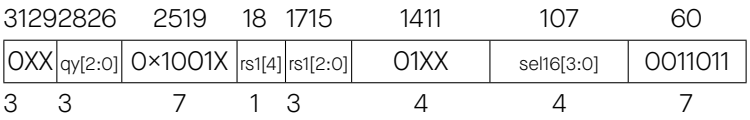


图 4.7-233. ESP.MOVI.8.Q

4.7.2.234 ESP.MOVX.R.CFG

语法

ESP.MOVX.R.CFG rd

描述

此指令将数据从特殊寄存器 CFG 移动到 rd 寄存器。

操作

```
rd[31 :  0] = CFG[31:0]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
CFG	1	-

格式

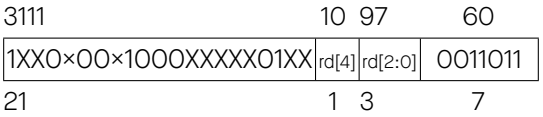


图 4.7-234. ESP.MOVX.R.CFG

4.7.2.235 ESP.MOVX.R.FFT.BIT.WIDTH

语法

ESP.MOVX.R.FFT.BIT.WIDTH rd

描述

此指令将数据从特殊寄存器 FFT_BIT_WIDTH 移动到 rd 寄存器。

操作

```
rd[31 :  0] = {28{0}, FFT_BIT_WIDTH[3  :  0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
FFT_BIT_WIDTH	1	-

格式

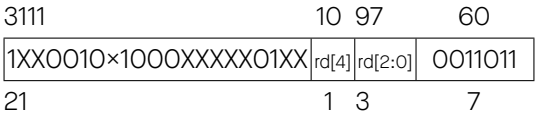


图 4.7-235. ESP.MOVX.R.FFT.BIT.WIDTH

4.7.2.236 ESP.MOVX.R.PERF

语法

ESP.MOVX.R.PERF rd,rs1 ESP.MOVX.R.PERF rd,rs1

描述

此指令将数据从特殊寄存器 PERF 移动到 rd 寄存器。

操作

```
rd[31 :  0] = PERF[31 :  0]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
PERF	1	-

格式

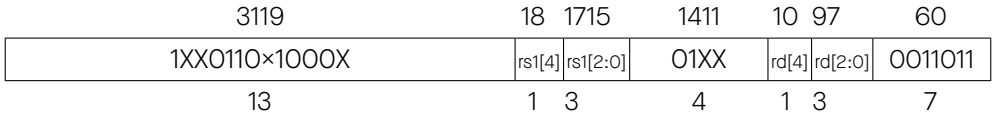


图 4.7-236. ESP.MOVX.R.PERF

4.7.2.237 ESP.MOVX.R.SAR

语法

ESP.MOVX.R.SAR rd

描述

此指令将数据从特殊寄存器 SAR 移动到 rd 寄存器。

操作

```
rd[31 :  0] = {26' b0, SAR[5:0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
SAR	1	-

格式

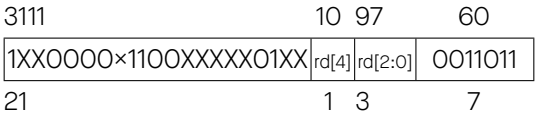


图 4.7-237. ESP.MOVX.R.SAR

4.7.2.238 ESP.MOVX.R.SAR.BYTES

语法

ESP.MOVX.R.SAR.BYTES rd

描述

此指令将数据从特殊寄存器 SAR_BYTES 移动到 rd 寄存器。

操作

```
rd[31 :  0] = {28' b0, SAR_BYTES[3:0]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
SAR_BYTES	1	-

格式

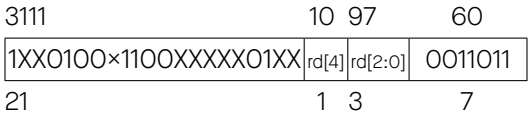


图 4.7-238. ESP.MOVX.R.SAR.BYTES

4.7.2.239 ESP.MOVX.R.XACC.H

语法

ESP.MOVX.R.XACC.H rd

描述

此指令将 8 位数据从特殊寄存器 XACC 移动到 rd 寄存器。

操作

```
rd[31 :  0] = {24'b0, XACC[39 : 32]}
```

调度

操作数	读流水级数	写流水级数
rd	-	2
XACC	2	-

格式

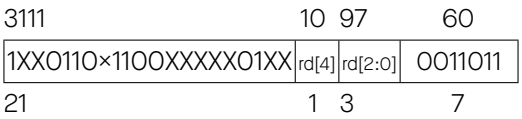


图 4.7-239. ESP.MOVX.R.XACC.H

4.7.2.240 ESP.MOVX.R.XACC.L

语法

ESP.MOVX.R.XACC.L rd

描述

此指令将特殊寄存器 XACC 的 32 位移动到 rd 寄存器。

操作

```
rd[31 :  0] = XACC[31 :  0]
```

调度

操作数	读流水级数	写流水级数
rd	-	2
XACC	2	-

格式

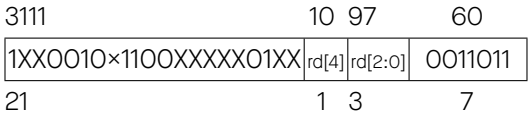


图 4.7-240. ESP.MOVX.R.XACC.L

4.7.2.241 ESP.MOVX.W.CFG

语法

ESP.MOVX.W.CFG rs1

描述

此指令将寄存器 `rs1` 的值写入特殊寄存器 `CFG`。

操作

```
CFG[31:0] = rs1[31:0]
```

调度

操作数	读流水级数	写流水级数
rs1	1	-
CFG	-	1

格式

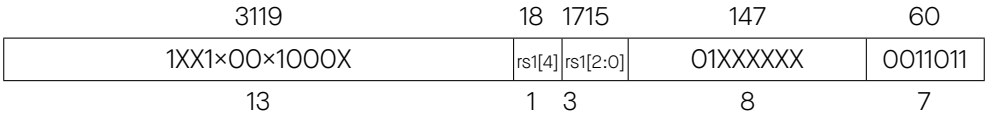


图 4.7-241. ESP.MOVX.W.CFG

4.7.2.242 ESP.MOVX.W.FFT.BIT.WIDTH

语法

ESP.MOVX.W.FFT.BIT.WIDTH rs1

描述

此指令将 `rs1` 写入 `FFT_BIT_WIDTH`。

操作

```
FFT_BIT_WIDTH[3  :  0] = rs1[3:0]
```

调度

操作数	读流水级数	写流水级数
rs1	1	-
FFT_BIT_WIDTH	-	1

格式

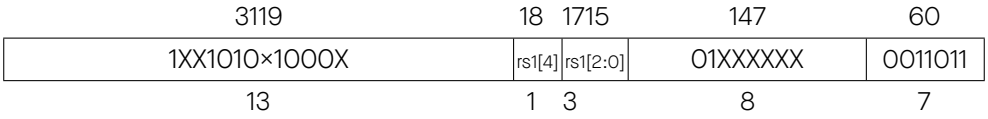


图 4.7-242. ESP.MOVX.W.FFT.BIT.WIDTH

4.7.2.243 ESP.MOVX.W.PERF

语法

ESP.MOVX.W.PERF rs1

描述

此指令将寄存器 rs1 的值写入特殊寄存器 PERF。

操作

```
PERF[31:0] = rs1[31:0]
```

调度

操作数	读流水级数	写流水级数
rs1	1	-
PERF	-	1

格式

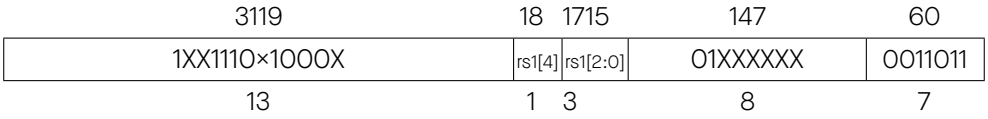


图 4.7-243. ESP.MOVX.W.PERF

4.7.2.244 ESP.MOVX.W.SAR

语法

ESP.MOVX.W.SAR rs1

描述

此指令将数据从 `rs1` 移动到特殊寄存器 SAR。

操作

```
SAR[5  :  0] = rs1[5:0]
```

调度

操作数	读流水级数	写流水级数
rs1	1	-
SAR	-	1

格式

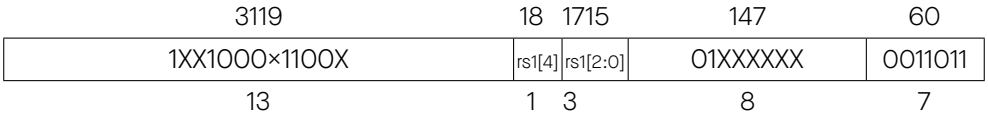


图 4.7-244. ESP.MOVX.W.SAR

4.7.2.245 ESP.MOVX.W.SAR.BYTES

语法

ESP.MOVX.W.SAR.BYTES rs1

描述

此指令将 `rs1` 写入 SAR_BYTES。

操作

```
SAR_BYTES[3  :  0] = rs1[3:0]
```

调度

操作数	读流水级数	写流水级数
rs1	1	-
SAR_BYTES	-	1

格式

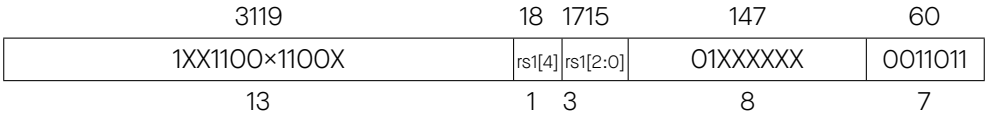


图 4.7-245. ESP.MOVX.W.SAR.BYTES

4.7.2.246 ESP.MOVX.W.XACC.H

语法

ESP.MOVX.W.XACC.H rs1

描述

此指令将 `rs1` 的值写入特殊寄存器 `XACC[39:32]`。

操作

```
XACC[39:32] = rs1[7:0]
```

调度

操作数	读流水级数	写流水级数
rs1	1	-
XACC	-	2

格式

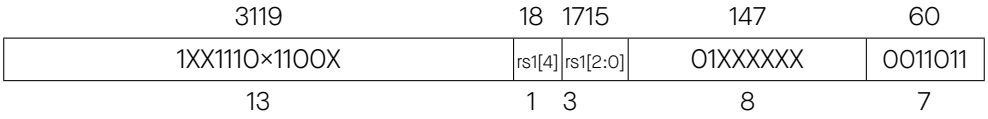


图 4.7-246. ESP.MOVX.W.XACC.H

4.7.2.247 ESP.MOVX.W.XACC.L

语法

ESP.MOVX.W.XACC.L rs1

描述

此指令将rs1的值写入特殊寄存器XACC[31:0]。

操作

```
XACC[31:0] = rs1[31:0]
```

调度

操作数	读流水级数	写流水级数
rs1	1	-
XACC	-	2

格式

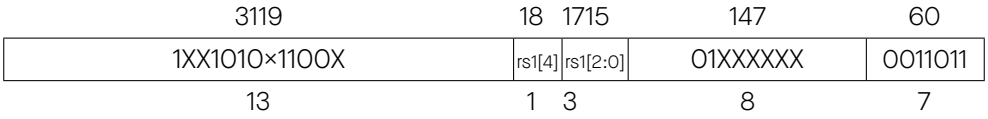


图 4.7-247. ESP.MOVX.W.XACC.L

4.7.2.248 ESP.VEXT.S16

语法

ESP.VEXT.S16 qz,qv,qw

描述

此指令将从 16 位段加载的数据符号扩展到 32 位段。

操作

```
qz[31 :  0] = { {16{qw[ 15]}}, qw[15 :  0] }
qz[63 : 32] = { {16{qw[ 31]}}, qw[31 : 16] }
qz[95 : 64] = { {16{qw[ 47]}}, qw[47 : 32] }
qz[127: 96] = { {16{qw[ 63]}}, qw[63 : 48] }
qv[31 :  0] = { {16{qw[ 79]}}, qw[79 : 64] }
qv[63 : 32] = { {16{qw[ 95]}}, qw[95 : 80] }
qv[95 : 64] = { {16{qw[111]}}, qw[111: 96] }
qv[127: 96] = { {16{qw[127]}}, qw[127:112] }
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qv	-	1
qw	1	-

格式

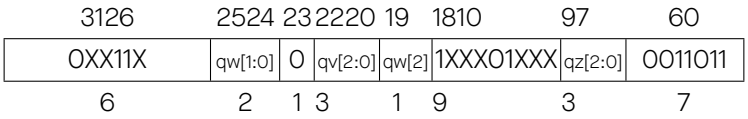


图 4.7-248. ESP.VEXT.S16

4.7.2.249 ESP.VEXT.S8

语法

ESP.VEXT.S8 qz,qv,qw

描述

此指令将从 8 位段加载的数据符号扩展到 16 位段。

操作

```
qz[15 :  0] = { {8{qw[  7]}}, qw[7  :  0] }
qz[31 : 16] = { {8{qw[ 15]}}, qw[15 :  8] }
qz[47 : 32] = { {8{qw[ 23]}}, qw[23 : 16] }
qz[63 : 48] = { {8{qw[ 31]}}, qw[31 : 24] }
qz[79 : 64] = { {8{qw[ 39]}}, qw[39 : 32] }
qz[95 : 80] = { {8{qw[ 47]}}, qw[47 : 40] }
qz[111: 96] = { {8{qw[ 55]}}, qw[55 : 48] }
qz[127:112] = { {8{qw[ 63]}}, qw[63 : 56] }
qv[15 :  0] = { {8{qw[ 71]}}, qw[71 : 64] }
qv[31 : 16] = { {8{qw[ 79]}}, qw[79 : 72] }
qv[47 : 32] = { {8{qw[ 87]}}, qw[87 : 80] }
qv[63 : 48] = { {8{qw[ 95]}}, qw[95 : 88] }
qv[79 : 64] = { {8{qw[103]}}, qw[103: 96] }
qv[95 : 80] = { {8{qw[111]}}, qw[111:104] }
qv[111: 96] = { {8{qw[119]}}, qw[119:112] }
qv[127:112] = { {8{qw[127]}}, qw[127:120] }
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qv	-	1
qw	1	-

格式

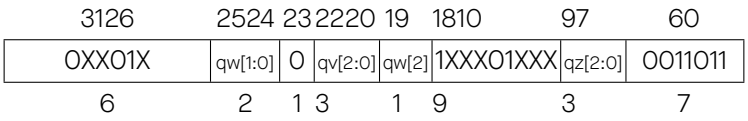


图 4.7-249. ESP.VEXT.S8

4.7.2.250 ESP.VEXT.U16

语法

ESP.VEXT.U16 qz,qv,qw

描述

此指令将从 16 位段加载的数据零扩展到 32 位段。

操作

```
qz[31 :  0] = { {16{0}}, qw[15 :  0] }
qz[63 : 32] = { {16{0}}, qw[31 : 16] }
qz[95 : 64] = { {16{0}}, qw[47 : 32] }
qz[127: 96] = { {16{0}}, qw[63 : 48] }
qv[31 :  0] = { {16{0}}, qw[79 : 64] }
qv[63 : 32] = { {16{0}}, qw[95 : 80] }
qv[95 : 64] = { {16{0}}, qw[111: 96] }
qv[127: 96] = { {16{0}}, qw[127:112] }
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qv	-	1
qw	1	-

格式

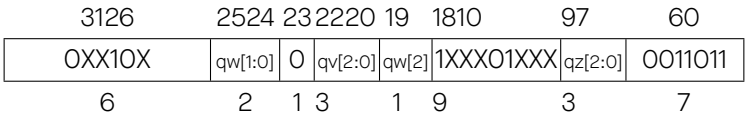


图 4.7-250. ESP.VEXT.U16

4.7.2.251 ESP.VEXT.U8

语法

ESP.VEXT.U8 qz,qv,qw

描述

此指令将从 8 位段加载的数据零扩展到 16 位段。

操作

```
qz[15 :  0] = { {8{0}}, qw[7  :  0] }
qz[31 : 16] = { {8{0}}, qw[15 :  8] }
qz[47 : 32] = { {8{0}}, qw[23 : 16] }
qz[63 : 48] = { {8{0}}, qw[31 : 24] }
qz[79 : 64] = { {8{0}}, qw[39 : 32] }
qz[95 : 80] = { {8{0}}, qw[47 : 40] }
qz[111: 96] = { {8{0}}, qw[55 : 48] }
qz[127:112] = { {8{0}}, qw[63 : 56] }
qv[15 :  0] = { {8{0}}, qw[71 : 64] }
qv[31 : 16] = { {8{0}}, qw[79 : 72] }
qv[47 : 32] = { {8{0}}, qw[87 : 80] }
qv[63 : 48] = { {8{0}}, qw[95 : 88] }
qv[79 : 64] = { {8{0}}, qw[103: 96] }
qv[95 : 80] = { {8{0}}, qw[111:104] }
qv[111: 96] = { {8{0}}, qw[119:112] }
qv[127:112] = { {8{0}}, qw[127:120] }
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qv	-	1
qw	1	-

格式

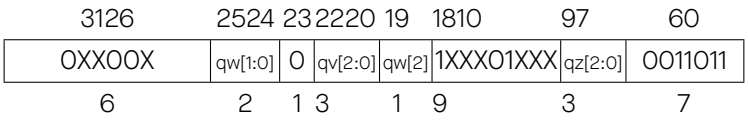


图 4.7-251. ESP.VEXT.U8

4.7.2.252 ESP.VUNZIP.16

语法

ESP.VUNZIP.16 qx,qy

描述

此指令在 16 位向量数据上实现了解压缩算法。

操作

```
qx[ 15:  0] = qx[ 15:  0]
qx[ 31: 16] = qx[ 47: 32]
qx[ 47: 32] = qx[ 79: 64]
qx[ 63: 48] = qx[111: 96]
qx[ 79: 64] = qy[ 15:  0]
qx[ 95: 80] = qy[ 47: 32]
qx[111: 96] = qy[ 79: 64]
qx[127:112] = qy[111: 96]
qy[ 15:  0] = qx[ 31: 16]
qy[ 31: 16] = qx[ 63: 48]
qy[ 47: 32] = qx[ 95: 80]
qy[ 63: 48] = qx[127:112]
qy[ 79: 64] = qy[ 31: 16]
qy[ 95: 80] = qy[ 63: 48]
qy[111: 96] = qy[ 95: 80]
qy[127:112] = qy[127:112]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

格式

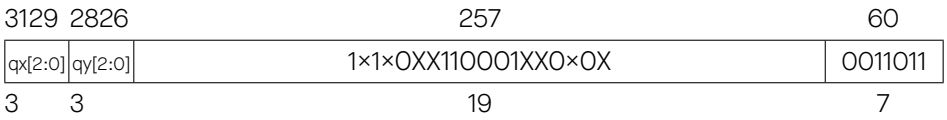


图 4.7-252. ESP.VUNZIP.16

4.7.2.253 ESP.VUNZIP.32

语法

ESP.VUNZIP.32 qx,qy

描述

此指令在 32 位向量数据上实现了解压缩算法。

操作

```
qx[ 31:  0] = qx[ 31:  0]
qx[ 63: 32] = qx[ 95: 64]
qx[ 95: 64] = qy[ 31:  0]
qx[127: 96] = qy[ 95: 64]
qy[ 31:  0] = qx[ 63: 32]
qy[ 63: 32] = qx[127: 96]
qy[ 95: 64] = qy[ 63: 32]
qy[127: 96] = qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

格式

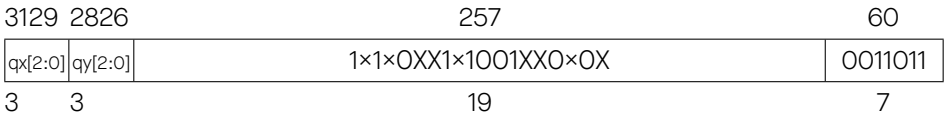


图 4.7-253. ESP.VUNZIP.32

4.7.2.254 ESP.VUNZIP.8

语法

ESP.VUNZIP.8 qx,qy

描述

此指令在 8 位向量数据上实现了解压缩算法。

操作

```
qx[  7:  0] = qx[  7:  0]
qx[ 15:  8] = qx[ 23: 16]
qx[ 23: 16] = qx[ 39: 32]
qx[ 31: 24] = qx[ 55: 48]
qx[ 39: 32] = qx[ 71: 64]
qx[ 47: 40] = qx[ 87: 80]
qx[ 55: 48] = qx[103: 96]
qx[ 63: 56] = qx[119:112]
qx[ 71: 64] = qy[  7:  0]
qx[ 79: 72] = qy[ 23: 16]
qx[ 87: 80] = qy[ 39: 32]
qx[ 95: 88] = qy[ 55: 48]
qx[103: 96] = qy[ 71: 64]
qx[111:104] = qy[ 87: 80]
qx[119:112] = qy[103: 96]
qx[127:120] = qy[119:112]
qy[  7:  0] = qx[ 15:  8]
qy[ 15:  8] = qx[ 31: 24]
qy[ 23: 16] = qx[ 47: 40]
qy[ 31: 24] = qx[ 63: 56]
qy[ 39: 32] = qx[ 79: 72]
qy[ 47: 40] = qx[ 95: 88]
qy[ 55: 48] = qx[111:104]
qy[ 63: 56] = qx[127:120]
qy[ 71: 64] = qy[ 15:  8]
qy[ 79: 72] = qy[ 31: 24]
qy[ 87: 80] = qy[ 47: 40]
qy[ 95: 88] = qy[ 63: 56]
qy[103: 96] = qy[ 79: 72]
qy[111:104] = qy[ 95: 88]
qy[119:112] = qy[111:104]
qy[127:120] = qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

格式

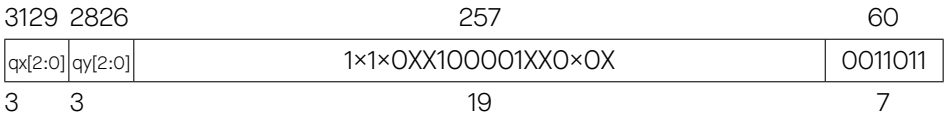


图 4.7-254. ESP.VUNZIP.8

4.7.2.255 ESP.VUNZIPT.16

语法

ESP.VUNZIPT.16 qx,qy,qw

描述

此指令交换了 qx，qy 和 qw 的值；它可以用于将 RGB 转换为 YUV。

操作

```
temp0[ 15:  0] = qx[ 15:  0]
temp1[ 15:  0] = qx[ 31: 16]
temp2[ 15:  0] = qx[ 47: 32]
temp0[ 31: 16] = qx[ 63: 48]
temp1[ 31: 16] = qx[ 79: 64]
temp2[ 31: 16] = qx[ 95: 80]
temp0[ 47: 32] = qx[111: 96]
temp1[ 47: 32] = qx[127:112]

temp2[ 47: 32] = qy[ 15:  0]
temp0[ 63: 48] = qy[ 31: 16]
temp1[ 63: 48] = qy[ 47: 32]
temp2[ 63: 48] = qy[ 63: 48]
temp0[ 79: 64] = qy[ 79: 64]
temp1[ 79: 64] = qy[ 95: 80]
temp2[ 79: 64] = qy[111: 96]
temp0[ 95: 80] = qy[127:112]

temp1[ 95: 80] = qw[ 15:  0]
temp2[ 95: 80] = qw[ 31: 16]
temp0[111: 96] = qw[ 47: 32]
temp1[111: 96] = qw[ 63: 48]
temp2[111: 96] = qw[ 79: 64]
temp0[127:112] = qw[ 95: 80]
temp1[127:112] = qw[111: 96]
temp2[127:112] = qw[127:112]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

操作数	读流水级数	写流水级数
qw	1	1

格式

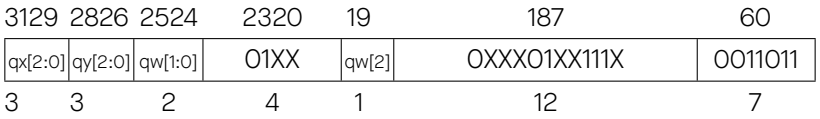


图 4.7-255. ESP.VUNZIPT.16

4.7.2.256 ESP.VUNZIPT.8

语法

ESP.VUNZIPT.8 qx,qy,qw

描述

此指令交换了 qx，qy 和 qw 的值；它可以用于将 RGB 转换为 YUV。

操作

```
temp0[  7:  0] = qx[  7:  0]
temp1[  7:  0] = qx[ 15:  8]
temp2[  7:  0] = qx[ 23: 16]
temp0[ 15:  8] = qx[ 31: 24]
temp1[ 15:  8] = qx[ 39: 32]
temp2[ 15:  8] = qx[ 47: 40]
temp0[ 23: 16] = qx[ 55: 48]
temp1[ 23: 16] = qx[ 63: 56]
temp2[ 23: 16] = qx[ 71: 64]
temp0[ 31: 24] = qx[ 79: 72]
temp1[ 31: 24] = qx[ 87: 80]
temp2[ 31: 24] = qx[ 95: 88]
temp0[ 39: 32] = qx[103: 96]
temp1[ 39: 32] = qx[111:104]
temp2[ 39: 32] = qx[119:112]
temp0[ 47: 40] = qx[127:120]
temp1[ 47: 40] = qy[  7:  0]
temp2[ 47: 40] = qy[ 15:  8]
temp0[ 55: 48] = qy[ 23: 16]
temp1[ 55: 48] = qy[ 31: 24]
temp2[ 55: 48] = qy[ 39: 32]
temp0[ 63: 56] = qy[ 47: 40]
temp1[ 63: 56] = qy[ 55: 48]
temp2[ 63: 56] = qy[ 63: 56]
temp0[ 71: 64] = qy[ 71: 64]
temp1[ 71: 64] = qy[ 79: 72]
temp2[ 71: 64] = qy[ 87: 80]
temp0[ 79: 72] = qy[ 95: 88]
temp1[ 79: 72] = qy[103: 96]
temp2[ 79: 72] = qy[111:104]
temp0[ 87: 80] = qy[119:112]
temp1[ 87: 80] = qy[127:120]

temp2[ 87: 80] = qw[  7:  0]
temp0[ 95: 88] = qw[ 15:  8]
temp1[ 95: 88] = qw[ 23: 16]
temp2[ 95: 88] = qw[ 31: 24]
temp0[103: 96] = qw[ 39: 32]
temp1[103: 96] = qw[ 47: 40]
temp2[103: 96] = qw[ 55: 48]
temp0[111:104] = qw[ 63: 56]
temp1[111:104] = qw[ 71: 64]
```

```
temp2[111:104] = qw[ 79: 72]
temp0[119:112] = qw[ 87: 80]
temp1[119:112] = qw[ 95: 88]
temp2[119:112] = qw[103: 96]
temp0[127:120] = qw[111:104]
temp1[127:120] = qw[119:112]
temp2[127:120] = qw[127:120]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1
qw	1	1

格式

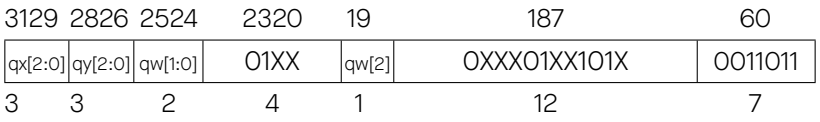


图 4.7-256. ESP.VUNZIPT.8

4.7.2.257 ESP.VZIP.16

语法

ESP.VZIP.16 qx,qy

描述

此指令交错了 qx 和 qy 的值。

操作

```
qx[ 15:  0] = qx[ 15:  0]
qx[ 31: 16] = qy[ 15:  0]
qx[ 47: 32] = qx[ 31: 16]
qx[ 63: 48] = qy[ 31: 16]
qx[ 79: 64] = qx[ 47: 32]
qx[ 95: 80] = qy[ 47: 32]
qx[111: 96] = qx[ 63: 48]
qx[127:112] = qy[ 63: 48]
qy[ 15:  0] = qx[ 79: 64]
qy[ 31: 16] = qy[ 79: 64]
qy[ 47: 32] = qx[ 95: 80]
qy[ 63: 48] = qy[ 95: 80]
qy[ 79: 64] = qx[111: 96]
qy[ 95: 80] = qy[111: 96]
qy[111: 96] = qx[127:112]
qy[127:112] = qy[127:112]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

格式

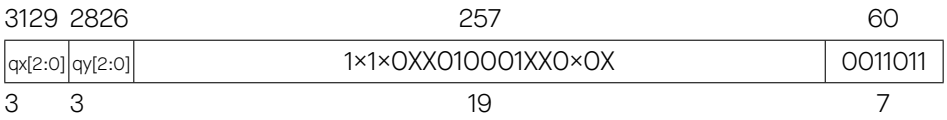


图 4.7-257. ESP.VZIP.16

4.7.2.258 ESP.VZIP.32

语法

ESP.VZIP.32 qx,qy

描述

此指令交错了 qx 和 qy 的值。

操作

```
qx[ 31:  0] = qx[ 31:  0]
qx[ 63: 32] = qy[ 31:  0]
qx[ 95: 64] = qx[ 63: 32]
qx[127: 96] = qy[ 63: 32]
qy[ 31:  0] = qx[ 95: 64]
qy[ 63: 32] = qy[ 95: 64]
qy[ 95: 64] = qx[127: 96]
qy[127: 96] = qy[127: 96]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

格式

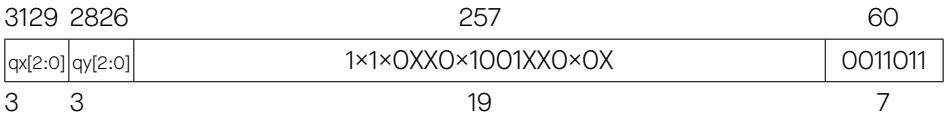


图 4.7-258. ESP.VZIP.32

4.7.2.259 ESP.VZIP.8

语法

ESP.VZIP.8 qx,qy

描述

此指令交错了 qx 和 qy 的值。

操作

```
qx[  7:  0] = qx[  7:  0]
qx[ 15:  8] = qy[  7:  0]
qx[ 23: 16] = qx[ 15:  8]
qx[ 31: 24] = qy[ 15:  8]
qx[ 39: 32] = qx[ 23: 16]
qx[ 47: 40] = qy[ 23: 16]
qx[ 55: 48] = qx[ 31: 24]
qx[ 63: 56] = qy[ 31: 24]
qx[ 71: 64] = qx[ 39: 32]
qx[ 79: 72] = qy[ 39: 32]
qx[ 87: 80] = qx[ 47: 40]
qx[ 95: 88] = qy[ 47: 40]
qx[103: 96] = qx[ 55: 48]
qx[111:104] = qy[ 55: 48]
qx[119:112] = qx[ 63: 56]
qx[127:120] = qy[ 63: 56]
qy[  7:  0] = qx[ 71: 64]
qy[ 15:  8] = qy[ 71: 64]
qy[ 23: 16] = qx[ 79: 72]
qy[ 31: 24] = qy[ 79: 72]
qy[ 39: 32] = qx[ 87: 80]
qy[ 47: 40] = qy[ 87: 80]
qy[ 55: 48] = qx[ 95: 88]
qy[ 63: 56] = qy[ 95: 88]
qy[ 71: 64] = qx[103: 96]
qy[ 79: 72] = qy[103: 96]
qy[ 87: 80] = qx[111:104]
qy[ 95: 88] = qy[111:104]
qy[103: 96] = qx[119:112]
qy[111:104] = qy[119:112]
qy[119:112] = qx[127:120]
qy[127:120] = qy[127:120]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

格式

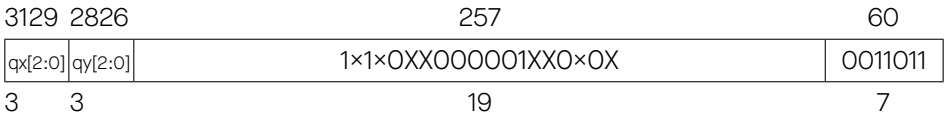


图 4.7-259. ESP.VZIP.8

4.7.2.260 ESP.VZIPT.16

语法

ESP.VZIPT.16 qx,qy,qw

描述

此指令交错 qx，qy 和 qw 的值。它可以用于图像颜色传输，例如，将 YUV 转换为 RGB。

操作

```
temp0[ 15:  0] = qx[ 15:  0]
temp0[ 31: 16] = qy[ 15:  0]
temp0[ 47: 32] = qw[ 15:  0]
temp0[ 63: 48] = qx[ 31: 16]
temp0[ 79: 64] = qy[ 31: 16]
temp0[ 95: 80] = qw[ 31: 16]
temp0[111: 96] = qx[ 47: 32]
temp0[127:112] = qy[ 47: 32]

temp1[ 15:  0] = qw[ 47: 32]
temp1[ 31: 16] = qx[ 63: 48]
temp1[ 47: 32] = qy[ 63: 48]
temp1[ 63: 48] = qw[ 63: 48]
temp1[ 79: 64] = qx[ 79: 64]
temp1[ 95: 80] = qy[ 79: 64]
temp1[111: 96] = qw[ 79: 64]
temp1[127:112] = qx[ 95: 80]

temp2[ 15:  0] = qy[ 95: 80]
temp2[ 31: 16] = qw[ 95: 80]
temp2[ 47: 32] = qx[111: 96]
temp2[ 63: 48] = qy[111: 96]
temp2[ 79: 64] = qw[111: 96]
temp2[ 95: 80] = qx[127:112]
temp2[111: 96] = qy[127:112]
temp2[127:112] = qw[127:112]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1

操作数	读流水级数	写流水级数
qw	1	1

格式

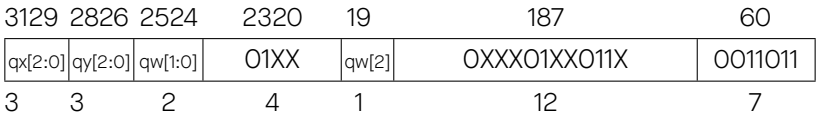


图 4.7-260. ESP.VZIPT.16

4.7.2.261 ESP.VZIPT.8

语法

ESP.VZIPT.8 qx,qy,qw

描述

此指令交错 qx，qy 和 qw 的值。它可以用于图像颜色传输，例如，将 YUV 转换为 RGB。

操作

```
temp0[  7:  0] = qx[  7:  0]
temp0[ 15:  8] = qy[  7:  0]
temp0[ 23: 16] = qw[  7:  0]
temp0[ 31: 24] = qx[ 15:  8]
temp0[ 39: 32] = qy[ 15:  8]
temp0[ 47: 40] = qw[ 15:  8]
temp0[ 55: 48] = qx[ 23: 16]
temp0[ 63: 56] = qy[ 23: 16]
temp0[ 71: 64] = qw[ 23: 16]
temp0[ 79: 72] = qx[ 31: 24]
temp0[ 87: 80] = qy[ 31: 24]
temp0[ 95: 88] = qw[ 31: 24]
temp0[103: 96] = qx[ 39: 32]
temp0[111:104] = qy[ 39: 32]
temp0[119:112] = qw[ 39: 32]
temp0[127:120] = qx[ 47: 40]

temp1[ 15:  0] = qy[ 47: 40]
temp1[ 31: 16] = qw[ 47: 40]
temp1[ 47: 32] = qx[ 55: 48]
temp1[ 63: 48] = qy[ 55: 48]
temp1[ 79: 64] = qw[ 55: 48]
temp1[ 95: 80] = qx[ 63: 56]
temp1[111: 96] = qy[ 63: 56]
temp1[127:112] = qw[ 63: 56]
temp1[ 15:  0] = qx[ 71: 64]
temp1[ 31: 16] = qy[ 71: 64]
temp1[ 47: 32] = qw[ 71: 64]
temp1[ 63: 48] = qx[ 79: 72]
temp1[ 79: 64] = qy[ 79: 72]
temp1[ 95: 80] = qw[ 79: 72]
temp1[111: 96] = qx[ 87: 80]
temp1[127:112] = qy[ 87: 80]

temp2[ 15:  0] = qw[ 87: 80]
temp2[ 31: 16] = qx[ 95: 88]
temp2[ 47: 32] = qy[ 95: 88]
temp2[ 63: 48] = qw[ 95: 88]
temp2[ 79: 64] = qx[103: 96]
temp2[ 95: 80] = qy[103: 96]
temp2[111: 96] = qw[103: 96]
temp2[127:112] = qx[111:104]
```

```
temp2[ 15:  0] = qy[111:104]
temp2[ 31: 16] = qw[111:104]
temp2[ 47: 32] = qx[119:112]
temp2[ 63: 48] = qy[119:112]
temp2[ 79: 64] = qw[119:112]
temp2[ 95: 80] = qx[127:120]
temp2[111: 96] = qy[127:120]
temp2[127:112] = qw[127:120]

qx[127:0] = temp0[127:0]
qy[127:0] = temp1[127:0]
qw[127:0] = temp2[127:0]
```

调度

操作数	读流水级数	写流水级数
qx	1	1
qy	1	1
qw	1	1

格式

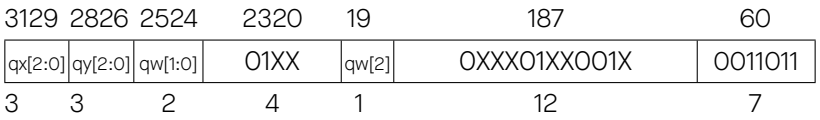


图 4.7-261. ESP.VZIPT.8

4.7.2.262 ESP.ZERO.Q

语法

ESP.ZERO.Q qz

描述

此指令将寄存器 qz 设置为零。

操作

```
qz[127:0] = 0
```

调度

操作数	读流水级数	写流水级数
qz	-	1

格式



图 4.7-262. ESP.ZERO.Q

4.7.2.263 ESP.ZERO.QACC

语法

ESP.ZERO.QACC

描述

此指令将 QACC_L 和 QACC_H 设置为零。

操作

```
QACC_L = 0
QACC_H = 0
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2

格式

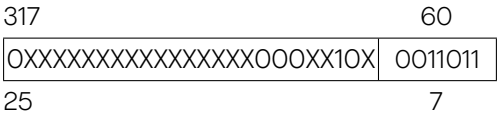


图 4.7-263. ESP.ZERO.QACC

4.7.2.264 ESP.ZERO.XACC

语法

ESP.ZERO.XACC

描述

此指令将 XACC 设置为零。

操作

```
XACC = 0
```

调度

操作数	读流水级数	写流水级数
XACC	-	2

格式



图 4.7-264. ESP.ZERO.XACC

4.7.2.265 ESP.FFT.AMS.S16.LD.INCP

语法

ESP.FFT.AMS.S16.LD.INCP qu,rs1,qz,qv,qx,qw,qy,sel2,sat

描述

这是一个专用的 FFT 指令，用于对 16 位数据段进行加法、减法、乘法、加法和减法以及移位操作。

在操作过程中，将 16 字节数据从内存加载到寄存器qu。访问后，寄存器rs1中的值增加 16。

操作

```
  temp0[16:0] = qx[ 47: 32] + qw[ 47: 32]
  temp1[16:0] = qx[ 63: 48] - qw[ 63: 48]

if sel2==0:
  temp2[33:0] = ((qx[ 47: 32] - qw[ 47: 32]) * qy[ 47: 32] - (qx[ 63: 48] + qw[ 63:
      48]) * qy[ 63: 48]) >> SAR
  temp3[33:0] = ((qx[ 47: 32] - qw[ 47: 32]) * qy[ 63: 48] + (qx[ 63: 48] + qw[ 63:
      48]) * qy[ 47: 32]) >> SAR
if sel2==1:
  temp2[33:0] = ((qx[ 63: 48] + qw[ 63: 48]) * qy[ 63: 48] + (qx[ 47: 32] - qw[ 47:
      32]) * qy[ 47: 32]) >> SAR
  temp3[33:0] = ((qx[ 63: 48] + qw[ 63: 48]) * qy[ 47: 32] - (qx[ 47: 32] - qw[ 47:
      32]) * qy[ 63: 48]) >> SAR

  add_res0[34: 0] = temp0[16:0] + temp2[33:0]
  add_res1[34: 0] = temp1[16:0] + temp3[33:0]
  sub_res0[34: 0] = temp0[16:0] - temp2[33:0]
  sub_res1[34: 0] = temp3[33:0] - temp1[16:0]

  qz[15:0] = sat_s16(add_res0[34:0])
  qz[31:16] = sat_s16(add_res1[34:0])
  qv[15:0] = sat_s16(sub_res0[34:0])
  qv[31:16] = sat_s16(sub_res1[34:0])

  qu = load128(as[31:0])
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qz	-	2
qv	-	2
qx	1	-
qw	1	-
qy	1	-
SAR	1	-

格式

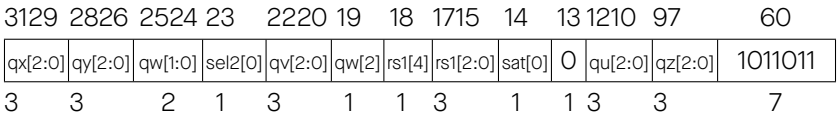


图 4.7-265. ESP.FFT.AMS.S16.LD.INCP

4.7.2.266 ESP.FFT.AMS.S16.LD.INCP.UAUP

语法

ESP.FFT.AMS.S16.LD.INCP.UAUP qu,rs1,qz,qv,qx,qw,qy,sel2,sat

描述

这是一种专用的 FFT 指令，设计用于对 16 位数据段进行加法、减法、乘法、加法和减法以及移位操作。

在操作过程中，从内存中加载 16 字节的数据。该指令将加载的数据和特殊寄存器UA_STATE中的数据组合成 32 字节的数据，然后将 32 字节的数据按照SAR_BYTE值乘以 8 的结果进行右移，并将移位结果的低 128 位赋值给寄存器qu。同时，寄存器UA_STATE将用加载的 16 字节数据进行更新。在访问之后，寄存器rs1中的值将增加 16。

操作

```
  temp0[16:0] = qx[ 15:  0] + qw[ 15:  0]
  temp1[16:0] = qx[ 31: 16] - qw[ 31: 16]

if sel2==0:
  temp2[33:0] = ((qx[ 15:  0] - qw[ 15:  0]) * qy[ 15:  0] - (qx[ 31: 16] + qw[ 31: 16]) * qy[ 31: 16]) >> SAR
  temp3[33:0] = ((qx[ 15:  0] - qw[ 15:  0]) * qy[ 31: 16] + (qx[ 31: 16] + qw[ 31: 16]) * qy[ 15:  0]) >> SAR
if sel2==1:
  temp2[33:0] = ((qx[ 31: 16] + qw[ 31: 16]) * qy[ 31: 16] + (qx[ 15:  0] - qw[ 15:  0]) * qy[ 15:  0]) >> SAR
  temp3[33:0] = ((qx[ 31: 16] + qw[ 31: 16]) * qy[ 15:  0] - (qx[ 15:  0] - qw[ 15:  0]) * qy[ 31: 16]) >> SAR

add_res0[34: 0] = temp0[16:0] + temp2[33:0]
add_res1[34: 0] = temp1[16:0] + temp3[33:0]
sub_res0[34: 0] = temp0[16:0] - temp2[33:0]
sub_res1[34: 0] = temp3[33:0] - temp1[16:0]

qz[15:0] = sat_s16(add_res0[34:0])
qz[31:16] = sat_s16(add_res1[34:0])
qv[15:0] = sat_s16(sub_res0[34:0])
qv[31:16] = sat_s16(sub_res1[34:0])

  dataIn[127:0] = load128(as[31:0])

  qu[127: 0] = {dataIn[127:0], UA_STATE[127:0]} >> {SAR_BYTE[3:0] << 3}
  UA_STATE[127:0] = dataIn[127:0]
  rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qz	-	2
qv	-	2
qx	1	-
qw	1	-
qy	1	-
SAR	1	-
SAR_BYTES	1	-

格式

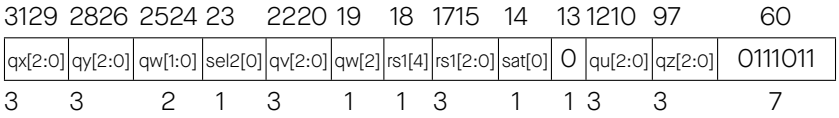


图 4.7-266. ESP.FFT.AMS.S16.LD.INCP.UAUP

4.7.2.267 ESP.FFT.AMS.S16.LD.R32.DECP

语法

ESP.FFT.AMS.S16.LD.R32.DECP qu,rs1,qz,qv,qx,qw,qy,sel2,sat

描述

这是一个专用的 FFT 指令，用于对 16 位数据段进行加法、减法、乘法、加法和减法以及移位操作。

在操作过程中，该指令强制将寄存器rs1中的访问地址的低 4 位设为 0，并以大端字序将内存中的 16 字节数据加载到寄存器qu中，即将数据的段 [127: 96] 加载到qu的 [31:0]。访问完成后，寄存器rs1中的值减少 16。

操作

```
  temp0[15:0] = qx[ 79: 64] + qy[ 79: 64]
  temp1[15:0] = qx[ 95: 80] - qy[ 95: 80]

if sel2==0:
  temp2[15:0] = ((qx[ 79: 64] - qw[ 79: 64]) * qy[ 79: 64] - (qx[ 95: 80] + qw[ 95:
      80]) * qy[ 95: 80]) >> SAR
  temp3[15:0] = ((qx[ 79: 64] - qw[ 79: 64]) * qy[ 95: 80] + (qx[ 95: 80] + qw[ 95:
      80]) * qy[ 79: 64]) >> SAR
if sel2==1:
  temp2[15:0] = ((qx[ 95: 80] + qw[ 95: 80]) * qy[ 95: 80] + (qx[ 79: 64] - qw[ 79:
      64]) * qy[ 79: 64]) >> SAR
  temp3[15:0] = ((qx[ 95: 80] + qw[ 95: 80]) * qy[ 79: 64] - (qx[ 79: 64] - qw[ 79:
      64]) * qy[ 95: 80]) >> SAR

  qz[79: 64] = temp0[15:0] + temp2[15:0]
  qz[95: 80] = temp1[15:0] + temp3[15:0]
  qv[79:64] = temp0[15:0] - temp2[15:0]
  qv[95:80] = temp3[15:0] - temp1[15:0]
  {qu[31: 0], qu[63: 32], qu[95: 64], qu[127: 96]} = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] - 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
qz	-	2
qv	-	2
qx	1	-
qw	1	-
qy	1	-
SAR	1	-

格式

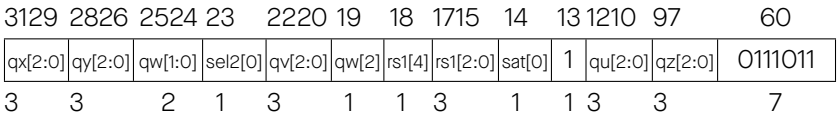


图 4.7-267. ESP.FFT.AMS.S16.LD.R32.DECP

4.7.2.268 ESP.FFT.AMS.S16.ST.INCP

语法

ESP.FFT.AMS.S16.ST.INCP qu,qz,rs2,rs1,qx,qw,qy,sel2,sat

描述

这是一个专用的 FFT 指令，用于对 16 位数据段执行加法、减法、乘法、加法和减法以及移位操作。

在操作过程中，将寄存器qu和rs2中的值拼接成 16 字节的数据，并将拼接结果存储到内存中。访问完成后，寄存器rs1中的值增加 16。此外，操作结果更新到寄存器rs2中。

操作

```
  temp0[15:0] = qx[111: 96] + qw[111: 96]
  temp1[15:0] = qx[127:112] - qw[127:112]

if sel2==0:
  temp2[15:0] = ((qx[111: 96] - qw[111: 96]) * qy[111: 96] - (qx[127:112] + qw
      [127:112]) * qy[127:112]) >> SAR[5:0]
  temp3[15:0] = ((qx[111: 96] - qw[111: 96]) * qy[127:112] + (qx[127:112] + qw
      [127:112]) * qy[111: 96]) >> SAR[5:0]
  {qv[ 95: 80] >> 1, qv[ 79: 64] >> 1, qv[ 63: 48] >> 1, qv[ 47: 32] >> 1, qv[ 31:
      16] >> 1, qv[ 15:  0] >> 1, rs2[31: 16] >> 1, rs2[15:0] >> 1} => store128(rs
      2[31:0])
if sel2==1:
  temp2[15:0] = ((qx[127:112] + qw[127:112]) * qy[127:112] + (qx[111: 96] - qw[111:
      96]) * qy[111: 96]) >> SAR[5:0]
  temp3[15:0] = ((qx[127:112] + qw[127:112]) * qy[111: 96] - (qx[111: 96] - qw[111:
      96]) * qy[127:112]) >> SAR[5:0]
  {qu[ 95: 64], qu[ 63: 32], qu[ 31:  0], rs2[31:0]} => store128(rs1[31:0])

  temp4[16:0] = temp1[15:0] + temp3[15:0]
  temp5[16:0] = temp0[15:0] + temp2[15:0]
  qz[111: 96] = temp0[15:0] - temp2[15:0]
  qz[127:112] = temp3[15:0] - temp1[15:0]
  rs2 = {temp4[15:0], temp5[15:0]}
  rs1[31:0] = rs1[31:0] +16
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
qz	-	2
qx	1	-
qw	1	-
qy	1	-
SAR	1	-

格式

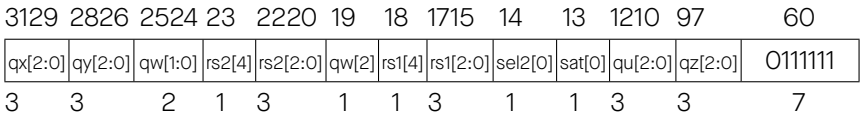


图 4.7-268. ESP.FFT.AMS.S16.ST.INCP

4.7.2.269 ESP.FFT.CMUL.S16.LD.XP

语法

ESP.FFT.CMUL.S16.LD.XP qu,rs1,rs2,qz,qy,qx,sel8,sat

描述

此指令执行 16 位有符号复数乘法。它类似于 ESP.CMUL.S16，只是寄存器 qx 和 qy 的顺序被颠倒。

在操作过程中，16 字节数据从内存加载到寄存器 qu。访问完成后，寄存器 rs1 中的值会增加寄存器 rs2 中的值。

操作

```
if sel8 == 0:
  qz[ 15:  0] = (qy[ 15:  0] * qx[ 15:  0] + qy[ 31: 16] * qx[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qy[ 31: 16] * qx[ 15:  0] - qy[ 15:  0] * qx[ 31: 16]) >> SAR[5:0]
if sel8 == 1:
  qz[ 15:  0] = (qy[ 15:  0] * qx[ 15:  0] - qy[ 31: 16] * qx[ 31: 16]) >> SAR[5:0]
  qz[ 31: 16] = (qy[ 31: 16] * qx[ 15:  0] + qy[ 15:  0] * qx[ 31: 16]) >> SAR[5:0]
if sel8 == 2:
  qz[ 47: 32] = (qy[ 47: 32] * qx[ 47: 32] + qy[ 63: 48] * qx[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qy[ 63: 48] * qx[ 47: 32] - qy[ 47: 32] * qx[ 63: 48]) >> SAR[5:0]
if sel8 == 3:
  qz[ 47: 32] = (qy[ 47: 32] * qx[ 47: 32] - qy[ 63: 48] * qx[ 63: 48]) >> SAR[5:0]
  qz[ 63: 48] = (qy[ 63: 48] * qx[ 47: 32] + qy[ 47: 32] * qx[ 63: 48]) >> SAR[5:0]
if sel8 == 4:
  qz[ 79: 64] = (qy[ 79: 64] * qx[ 79: 64] + qy[ 95: 80] * qx[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qy[ 95: 80] * qx[ 79: 64] - qy[ 79: 64] * qx[ 95: 80]) >> SAR[5:0]
if sel8 == 5:
  qz[ 79: 64] = (qy[ 79: 64] * qx[ 79: 64] - qy[ 95: 80] * qx[ 95: 80]) >> SAR[5:0]
  qz[ 95: 80] = (qy[ 95: 80] * qx[ 79: 64] + qy[ 79: 64] * qx[ 95: 80]) >> SAR[5:0]

  qu[127:0] = load128(rs1[31:0])
  rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
qz	-	2
qx	1	-
qy	1	-
SAR	1	-

格式

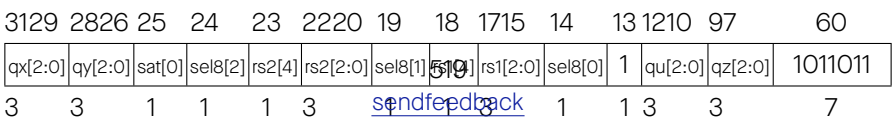


图 4.7-269. ESP.FFT.CMUL.S16.LD.XP

4.7.2.270 ESP.FFT.CMUL.S16.ST.XP

语法

ESP.FFT.CMUL.S16.ST.XP qy,qx,qu,rs1,rs2,sel8,upd4,sel4,sat

描述

此指令执行 16 位有符号复数乘法。它类似于 ESP.CMUL.S16，只是寄存器 qx 和 qy 的顺序被颠倒。

操作的结果和寄存器 qx 和 qu 中的数据段，由立即数据 upd4 指定，被连接成 128 位。然后将这些写入内存。

在访问完成后，寄存器 rs1 中的值将增加寄存器 rs2 中的值。

操作

```
if sel8 == 6:
  temp[15: 0] = (qy[111: 96] * qx[111: 96] + qy[127:112] * qx[127:112]) >> SAR[5:0]
  temp[31:16] = (qy[127:112] * qx[111: 96] - qy[111: 96] * qx[127:112]) >> SAR[5:0]
if sel8 == 7:
  temp[15: 0] = (qy[111: 96] * qx[111: 96] - qy[127:112] * qx[127:112]) >> SAR[5:0]
  temp[31:16] = (qy[127:112] * qx[111: 96] + qy[111: 96] * qx[127:112]) >> SAR[5:0]

if upd4 == 0:  // normal radix 2
  {temp[31:0], qu[ 95:  0]} => store128(rs1[31:0])
if upd4 == 1:  // radix2 last second stage
  {
       temp[31:0],
       qu[ 95: 64],
       qy[ 63: 48] >> sel4,
       qy[ 47: 32] >> sel4,
       qy[ 31: 16] >> sel4,
       qy[ 15:  0] >> sel4
  } => store128(rs1[31:0])
if upd4 == 2:  // radix2 last stage
  {
       temp[31:0],
       qy[ 63: 48] >> sel4,
       qy[ 47: 32] >> sel4,
       qu[ 95: 64],
       qy[ 31: 16] >> sel4,
       qy[ 15:  0] >> sel4
  } => store128(rs1[31:0])

  rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-
qx	1	-
qy	1	-
SAR	1	-

格式

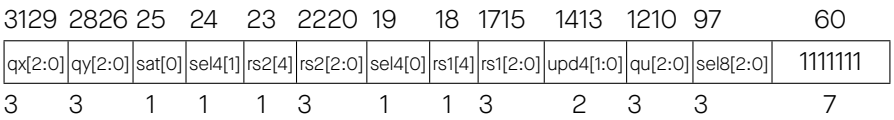


图 4.7-270. ESP.FFT.CMUL.S16.ST.XP

4.7.2.271 ESP.FFT.R2BF.S16

语法

ESP.FFT.R2BF.S16 qz,qv,qx,qy,sel2,sat

描述

此指令在寄存器 qx 和 qy 中的 16 字节值上执行基数-2 蝴蝶操作，数据位宽为 16 位。一些计算结果被写入寄存器 qz，而其他的被写入寄存器 qv。

操作

```
if sel2==0:
  op_a[127:  0] = {qy[ 63:  0], qx[ 63:  0]}
  op_b[127:  0] = {qy[127: 64], qx[127: 64]}
if sel2==1:
  op_a[127:  0] = {qy[ 95: 64], qy[ 31:  0], qx[ 95: 64], qx[ 31:  0]}
  op_b[127:  0] = {qy[127: 96], qy[ 63: 32], qx[127: 96], qx[ 63: 32]}

  qz[ 15:  0] = op_a[ 15:  0] + op_b[ 15:  0]
  qz[ 31: 16] = op_a[ 31: 16] + op_b[ 31: 16]
  qz[ 47: 32] = op_a[ 47: 32] + op_b[ 47: 32]
  qz[ 63: 48] = op_a[ 63: 48] + op_b[ 63: 48]
  qz[ 79: 64] = op_a[ 15:  0] - op_b[ 15:  0]
  qz[ 95: 80] = op_a[ 31: 16] - op_b[ 31: 16]
  qz[111: 96] = op_a[ 47: 32] - op_b[ 47: 32]
  qz[127:112] = op_a[ 63: 48] - op_b[ 63: 48]

  qv[ 15:  0] = op_a[ 79: 64] + op_b[ 79: 64]
  qv[ 31: 16] = op_a[ 95: 80] + op_b[ 95: 80]
  qv[ 47: 32] = op_a[111: 96] + op_b[111: 96]
  qv[ 63: 48] = op_a[127:112] + op_b[127:112]
  qv[ 79: 64] = op_a[ 79: 64] - op_b[ 79: 64]
  qv[ 95: 80] = op_a[ 95: 80] - op_b[ 95: 80]
  qv[111: 96] = op_a[111: 96] - op_b[111: 96]
  qv[127:112] = op_a[127:112] - op_b[127:112]
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qv	-	1
qx	1	-
qy	1	-

格式

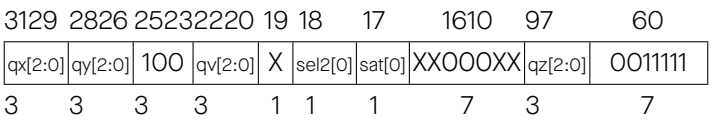


图 4.7-271. ESP.FFT.R2BF.S16

4.7.2.272 ESP.FFT.R2BF.S16.ST.INCP

语法

ESP.FFT.R2BF.S16.ST.INCP qz,qx,qy,rs1,sel4,sat

描述

此指令在寄存器 qx 和 qy 中的 16 字节值上执行基数-2 蝴蝶操作，数据位宽为 16 位。一些计算结果被写入寄存器 qz，而其他的则进行算术移位并写入由 rs1 指示的内存地址。访问完成后，寄存器 rs1 中的值增加 16。

操作

```
qz[ 15:  0] = qx[ 15:  0] - qy[ 15:  0]
qz[ 31: 16] = qx[ 31: 16] - qy[ 31: 16]
qz[ 47: 32] = qx[ 47: 32] - qy[ 47: 32]
qz[ 63: 48] = qx[ 63: 48] - qy[ 63: 48]
qz[ 79: 64] = qx[ 79: 64] - qy[ 79: 64]
qz[ 95: 80] = qx[ 95: 80] - qy[ 95: 80]
qz[111: 96] = qx[111: 96] - qy[111: 96]
qz[127:112] = qx[127:112] - qy[127:112]

{
  (qx[127:112] + qy[127:112]) >> sel4,
  (qx[111: 96] + qy[111: 96]) >> sel4,
  (qx[ 95: 80] + qy[ 95: 80]) >> sel4,
  (qx[ 79: 64] + qy[ 79: 64]) >> sel4,
  (qx[ 63: 48] + qy[ 63: 48]) >> sel4,
  (qx[ 47: 32] + qy[ 47: 32]) >> sel4,
  (qx[ 31: 16] + qy[ 31: 16]) >> sel4,
  (qx[ 15:  0] + qy[ 15:  0]) >> sel4
} => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	1
rs1	1	1
qx	1	-
qy	1	-

格式

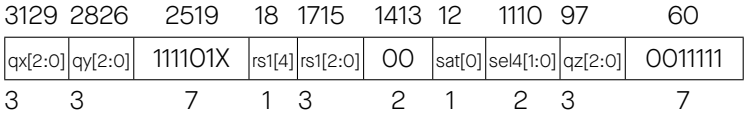


图 4.7-272. ESP.FFT.R2BF.S16.ST.INCP

4.7.2.273 ESP.FFT.VST.R32.DECP

语法

ESP.FFT.VST.R32.DECP qu,rs1,sel2

描述

这是一个专用的 FFT 指令。此指令将寄存器qu中的数据分为 8 个 16 位数据段，并根据立即数sel2对它们进行 0 或 1 的算术右移，最后将结果写入由寄存器rs1指示的内存地址，以字大端顺序。访问完成后，寄存器rs1中的值将减少 16。

操作

```
{
  qu[ 31: 16] >> sel2,
  qu[ 15:  0] >> sel2,
  qu[ 63: 48] >> sel2,
  qu[ 47: 32] >> sel2,
  qu[ 95: 80] >> sel2,
  qu[ 79: 64] >> sel2,
  qu[127:112] >> sel2,
  qu[111: 96] >> sel2
} => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] - 16
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1

格式

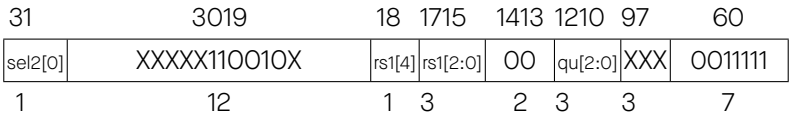


图 4.7-273. ESP.FFT.VST.R32.DECP

4.7.2.274 ESP.LD.128.USAR.IP

语法

ESP.LD.128.USAR.IP qu,rs1,imm16

描述

此指令将 16 字节的数据从内存加载到寄存器qu。同时，它将寄存器rs1中的低 4 位的值保存到特殊寄存器SAR_BYTE。在访问完成后，寄存器rs1中的值会增加指令代码段中的一个 8 位符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
qu[127:0] = load128(rs1[31:0])
SAR_BYTE = rs1[3:0]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
SAR_BYTE	-	1

格式

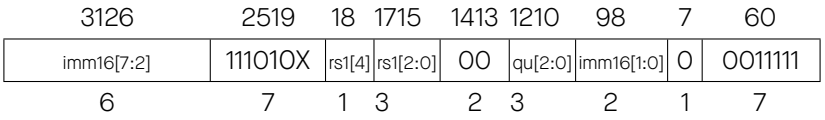


图 4.7-274. ESP.LD.128.USAR.IP

4.7.2.275 ESP.LD.128.USAR.XP

语法

ESP.LD.128.USAR.XP qu,rs1,rs2

描述

此指令将 16 字节的数据从内存加载到寄存器qu中。同时，它将rs1中的低 4 位的值保存到特殊寄存器SAR_BYTE中。访问完成后，寄存器rs1中的值会增加寄存器rs2中的值。

操作

```
qu[127:0] = load128(rs1[31:0])
SAR_BYTE = rs1[3:0]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-
SAR_BYTE	-	1

格式

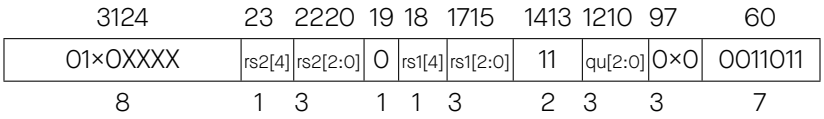


图 4.7-275. ESP.LD.128.USAR.XP

4.7.2.276 ESP.LD.XACC.IP

语法

ESP.LD.XACC.IP rs1,imm8

描述

从内存中加载 64 位数据，并将其低 40 位保存到特殊寄存器XACC。访问完成后，寄存器rs1中的值会增加一个在指令代码段中左移 3 位的 8 位符号扩展常数。

imm 的起始值为-1024，结束值为 1016，步长为 8。

操作

```
DataIn[63:0] = load64(rs1[31:0])
XACC[39:0] = DataIn[39:0]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
XACC	-	2
rs1	1	1

格式

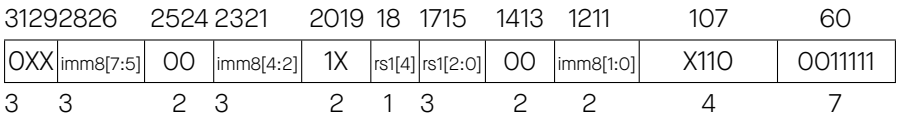


图 4.7-276. ESP.LD.XACC.IP

4.7.2.277 ESP.LDQA.S16.128.IP

语法

ESP.LDQA.S16.128.IP rs1,imm16

描述

此指令从内存中加载 16 字节数据，将其划分为 8 个 16 位段，对每个段进行符号扩展至 64 位，然后将结果分别存储到 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值将增加指令代码段中的一个 8 位符号扩展常数，左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{dataIn[15]}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{dataIn[31]}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{dataIn[47]}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{dataIn[63]}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{dataIn[79]}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{dataIn[95]}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{dataIn[111]}, dataIn[111: 96]}
QACC_H[159:192] = {48{dataIn[127]}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1

格式

3127	26	2524	2321	2019	18	1715	1413	129	87	60
0XX11	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	01	0011111
5	1	2	3	2	1	3	2	4	2	7

图 4.7-277. ESP.LDQA.S16.128.IP

4.7.2.278 ESP.LDQA.S16.128.XP

语法

ESP.LDQA.S16.128.XP rs1,rs2

描述

此指令从内存中加载 16 字节数据，将其划分为 8 个 16 位的段，对每个段进行符号扩展至 64 位，然后将结果分别存储在 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值将增加寄存器rs2中的值。

操作

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{dataIn[15]}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{dataIn[31]}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{dataIn[47]}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{dataIn[63]}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{dataIn[79]}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{dataIn[95]}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{dataIn[111]}, dataIn[111: 96]}
QACC_H[159:192] = {48{dataIn[127]}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

格式

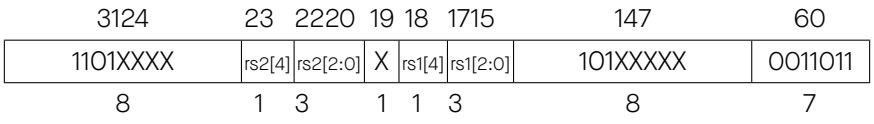


图 4.7-278. ESP.LDQA.S16.128.XP

4.7.2.279 ESP.LDQA.S8.128.IP

语法

ESP.LDQA.S8.128.IP rs1,imm16

描述

此指令从内存中加载 16 字节数据，将其划分为 16 个 8 位段，对每个段进行符号扩展至 32 位，然后将结果分别存储在 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值将增加指令代码段中的 8 位符号扩展常数，左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{DataIn[ 7]}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{DataIn[15]}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{DataIn[23]}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{DataIn[63]}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{DataIn[71]}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{DataIn[79]}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{DataIn[87]}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{DataIn[127]}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1

格式

3127	26	2524	2321	2019	18	1715	1413	129	87	60
0XX01	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	01	0011111
5	1	2	3	2	1	3	2	4	2	7

图 4.7-279. ESP.LDQA.S8.128.IP

4.7.2.280 ESP.LDQA.S8.128.XP

语法

ESP.LDQA.S8.128.XP rs1,rs2

描述

此指令从内存中加载 16 字节数据，将其划分为 16 个 8 位段，对每个段进行符号扩展至 32 位，然后将结果分别存储在 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值将增加寄存器rs2中的值。

操作

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{DataIn[ 7]}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{DataIn[15]}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{DataIn[23]}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{DataIn[63]}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{DataIn[71]}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{DataIn[79]}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{DataIn[87]}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{DataIn[127]}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

格式

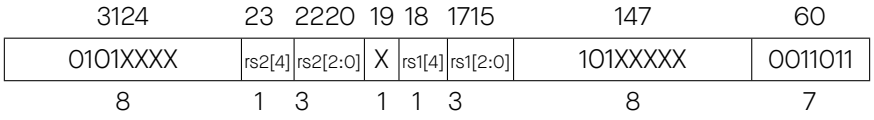


图 4.7-280. ESP.LDQA.S8.128.XP

4.7.2.281 ESP.LDQA.U16.128.IP

语法

ESP.LDQA.U16.128.IP rs1,imm16

描述

此指令从内存中加载 16 字节数据，将其分为 8 个 16 位段，将每个段零扩展到 64 位，然后将结果分别存储在 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值会增加指令代码段中的一个 8 位符号扩展常数，左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{0}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{0}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{0}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{0}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{0}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{0}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{0}, dataIn[111: 96]}
QACC_H[159:192] = {48{0}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1

格式

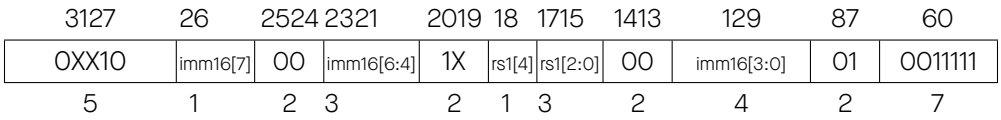


图 4.7-281. ESP.LDQA.U16.128.IP

4.7.2.282 ESP.LDQA.U16.128.XP

语法

ESP.LDQA.U16.128.XP rs1,rs2

描述

此指令从内存中加载 16 字节数据，将其划分为 8 个 16 位的段，将每个段零扩展到 64 位，然后将结果分别存储在 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值将增加寄存器rs2中的值。

操作

```
dataIn[127:0] = load128(rs1[31:0])
QACC_L[ 63:  0] = {48{0}, dataIn[ 15:  0]}
QACC_L[127: 64] = {48{0}, dataIn[ 31: 16]}
QACC_L[191:128] = {48{0}, dataIn[ 47: 32]}
QACC_L[255:192] = {48{0}, dataIn[ 63: 48]}

QACC_H[ 63:  0] = {48{0}, dataIn[ 79: 64]}
QACC_H[127: 64] = {48{0}, dataIn[ 95: 80]}
QACC_H[191:128] = {48{0}, dataIn[111: 96]}
QACC_H[159:192] = {48{0}, dataIn[127:112]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

格式

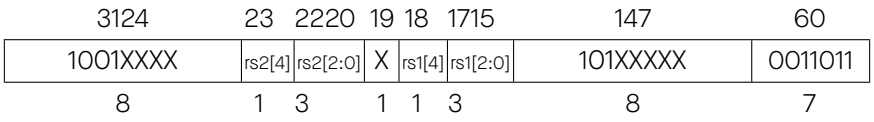


图 4.7-282. ESP.LDQA.U16.128.XP

4.7.2.283 ESP.LDQA.U8.128.IP

语法

ESP.LDQA.U8.128.IP rs1,imm16

描述

此指令从内存中加载 16 字节数据，将其划分为 16 个 8 位段，将每个段零扩展到 32 位，然后将结果分别存储在 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值将增加指令代码段中的 8 位符号扩展常数，该常数左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{0}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{0}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{0}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{0}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{0}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{0}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{0}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{0}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1

格式

3127	26	2524	2321	2019	18	1715	1413	129	87	60
0XX00	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	01	0011111
5	1	2	3	2	1	3	2	4	2	7

图 4.7-283. ESP.LDQA.U8.128.IP

4.7.2.284 ESP.LDQA.U8.128.XP

语法

ESP.LDQA.U8.128.XP rs1,rs2

描述

此指令从内存中加载 16 字节数据，将其划分为 16 个 8 位段，将每个段零扩展到 32 位，然后将结果分别存储在 256 位特殊寄存器QACC_L和QACC_H中。访问完成后，寄存器rs1中的值将增加寄存器rs2中的值。

操作

```
DataIn[127:0] = load128(rs1[31:0])
QACC_L[31 :  0] = {24{0}, DataIn[7  :  0]}
QACC_L[63 : 32] = {24{0}, DataIn[15 :  8]}
QACC_L[95 : 64] = {24{0}, DataIn[23 : 16]}
...
QACC_L[255:224] = {24{0}, DataIn[63 : 56]}
QACC_H[31 :  0] = {24{0}, DataIn[71 : 64]}
QACC_H[63 : 32] = {24{0}, DataIn[79 : 72]}
QACC_H[95 : 64] = {24{0}, DataIn[87 : 80]}
...
QACC_H[255:224] = {24{0}, DataIn[127:120]}

rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
QACC_L	-	2
rs1	1	1
rs2	1	-

格式

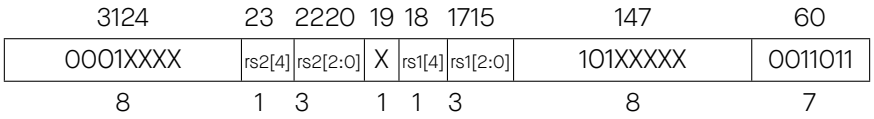


图 4.7-284. ESP.LDQA.U8.128.XP

4.7.2.285 ESP.VLDBC.16.IP

语法

ESP.VLDBC.16.IP qu,rs1,imm2

描述

这些指令将 16 位数据从内存加载并广播到寄存器 qu，并在加载后通过立即值更新指针。

imm 的起始值为-256，结束值为 254，步长为 2。

操作

```
qu[127:0] = {8{load16({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1

格式

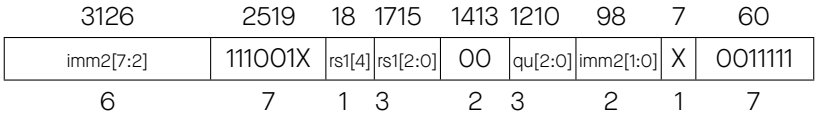


图 4.7-285. ESP.VLDBC.16.IP

4.7.2.286 ESP.VLDBC.16.XP

语法

ESP.VLDBC.16.XP qu,rs1,rs2

描述

这些指令将 16 位数据从内存加载并广播到寄存器 qu，并在加载后通过寄存器 rs2 中的值更新指针。

操作

```
qu[127:0] = {8{load16({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-

格式

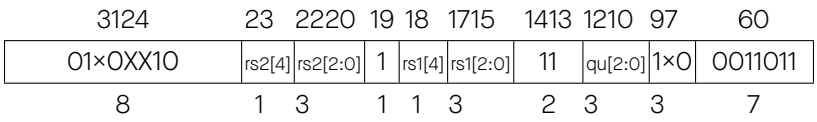


图 4.7-286. ESP.VLDBC.16.XP

4.7.2.287 ESP.VLDBC.32.IP

语法

ESP.VLDBC.32.IP qu,rs1,imm4

描述

这些指令将 32 位数据从内存加载并广播到寄存器qu，并通过立即值更新指针后的位置。

imm 的起始值为-512，结束值为 508，步长为 4。

操作

```
qu[127:0] = {4{load32({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1

格式

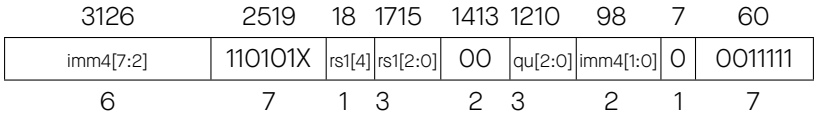


图 4.7-287. ESP.VLDBC.32.IP

4.7.2.288 ESP.VLDBC.32.XP

语法

ESP.VLDBC.32.XP qu,rs1,rs2

描述

这些指令从内存中加载 32 位数据并广播到寄存器qu，并在加载后通过寄存器rs2中的值更新指针。

操作

```
qu[127:0] = {4{load32({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-

格式

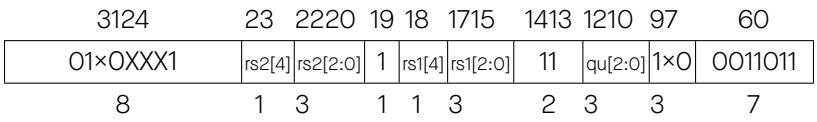


图 4.7-288. ESP.VLDBC.32.XP

4.7.2.289 ESP.VLDBC.8.IP

语法

ESP.VLDBC.8.IP qu,rs1,imm1

描述

这些指令将 8 位数据从内存加载并广播到寄存器 qu，并通过立即值更新指针后的位置。

imm 的起始值为-128，结束值为 127，步长为 1。

操作

```
qu[127:0] = {16{load8({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1

格式

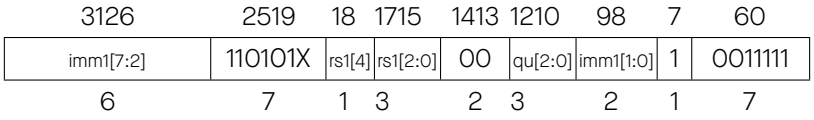


图 4.7-289. ESP.VLDBC.8.IP

4.7.2.290 ESP.VLDBC.8.XP

语法

ESP.VLDBC.8.XP qu,rs1,rs2

描述

这些指令从内存中加载 8 位数据并广播到寄存器qu，并在加载后通过寄存器rs2中的值更新指针。

操作

```
qu[127:0] = {16{load8({rs1[31:0]})}}
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-

格式

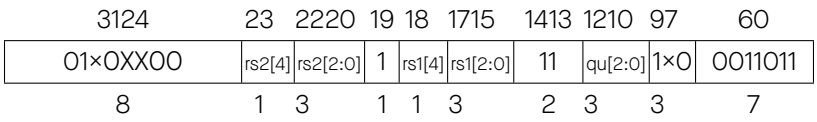


图 4.7-290. ESP.VLDBC.8.XP

4.7.2.291 ESP.VLDEXT.S16.IP

语法

ESP.VLDEXT.S16.IP qu,qz,rs1,imm16

描述

此指令从内存中加载 128 位数据，并将数据作为 16 位段符号扩展到 32 位，指针按立即数进行自增。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 31:  0] = { {16{DataIn[ 15]}}, DataIn[ 15:  0] }
qu[ 63: 32] = { {16{DataIn[ 31]}}, DataIn[ 31: 16] }
qu[ 95: 64] = { {16{DataIn[ 47]}}, DataIn[ 47: 32] }
qu[127: 96] = { {16{DataIn[ 63]}}, DataIn[ 63: 48] }
qz[ 31:  0] = { {16{DataIn[ 79]}}, DataIn[ 79: 64] }
qz[ 63: 32] = { {16{DataIn[ 95]}}, DataIn[ 95: 80] }
qz[ 95: 64] = { {16{DataIn[111]}}, DataIn[111: 96] }
qz[127: 96] = { {16{DataIn[127]}}, DataIn[127:112] }
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1

格式

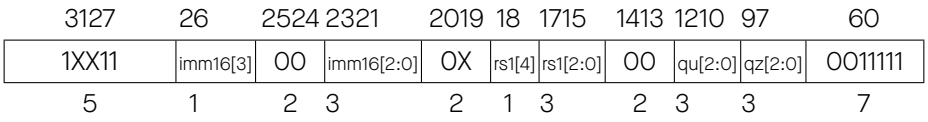


图 4.7-291. ESP.VLDEXT.S16.IP

4.7.2.292 ESP.VLDEXT.S16.XP

语法

ESP.VLDEXT.S16.XP qu,qz,rs1,rs2

描述

此指令从内存中加载 128 位数据，并将数据从 16 位段符号扩展到 32 位段。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 31:  0] = { {16{DataIn[ 15]}}, DataIn[ 15:  0] }
qu[ 63: 32] = { {16{DataIn[ 31]}}, DataIn[ 31: 16] }
qu[ 95: 64] = { {16{DataIn[ 47]}}, DataIn[ 47: 32] }
qu[127: 96] = { {16{DataIn[ 63]}}, DataIn[ 63: 48] }
qz[ 31:  0] = { {16{DataIn[ 79]}}, DataIn[ 79: 64] }
qz[ 63: 32] = { {16{DataIn[ 95]}}, DataIn[ 95: 80] }
qz[ 95: 64] = { {16{DataIn[111]}}, DataIn[111: 96] }
qz[127: 96] = { {16{DataIn[127]}}, DataIn[127:112] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

格式

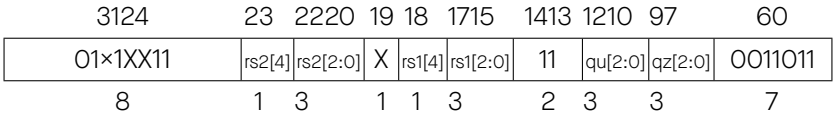


图 4.7-292. ESP.VLDEXT.S16.XP

4.7.2.293 ESP.VLDEXT.S8.IP

语法

ESP.VLDEXT.S8.IP qu,qz,rs1,imm16

描述

此指令从内存中加载 128 位数据，并将数据作为 8 位段符号扩展到 16 位，指针通过立即值进行后更新。

imm 从-2048 开始，到 2032 结束，步长为 16。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{DataIn[  7]}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{DataIn[ 15]}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{DataIn[ 23]}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{DataIn[ 31]}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{DataIn[ 39]}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{DataIn[ 47]}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{DataIn[ 55]}}, DataIn[ 55: 48] }
qu[127:112] = { {8{DataIn[ 63]}}, DataIn[ 63: 56] }
qz[ 15:  0] = { {8{DataIn[ 71]}}, DataIn[ 71: 64] }
qz[ 31: 16] = { {8{DataIn[ 79]}}, DataIn[ 79: 72] }
qz[ 47: 32] = { {8{DataIn[ 87]}}, DataIn[ 87: 80] }
qz[ 63: 48] = { {8{DataIn[ 95]}}, DataIn[ 95: 88] }
qz[ 79: 64] = { {8{DataIn[103]}}, DataIn[103: 96] }
qz[ 95: 80] = { {8{DataIn[111]}}, DataIn[111:104] }
qz[111: 96] = { {8{DataIn[119]}}, DataIn[119:112] }
qz[127:112] = { {8{DataIn[127]}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1

格式

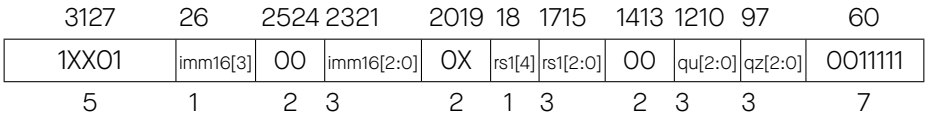


图 4.7-293. ESP.VLDEXT.S8.IP

4.7.2.294 ESP.VLDEXT.S8.XP

语法

ESP.VLDEXT.S8.XP qu,qz,rs1,rs2

描述

此指令从内存中加载 128 位数据，并将数据从 8 位段扩展到 16 位段。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{DataIn[  7]}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{DataIn[ 15]}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{DataIn[ 23]}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{DataIn[ 31]}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{DataIn[ 39]}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{DataIn[ 47]}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{DataIn[ 55]}}, DataIn[ 55: 48] }
qu[127:112] = { {8{DataIn[ 63]}}, DataIn[ 63: 56] }
qz[ 15:  0] = { {8{DataIn[ 71]}}, DataIn[ 71: 64] }
qz[ 31: 16] = { {8{DataIn[ 79]}}, DataIn[ 79: 72] }
qz[ 47: 32] = { {8{DataIn[ 87]}}, DataIn[ 87: 80] }
qz[ 63: 48] = { {8{DataIn[ 95]}}, DataIn[ 95: 88] }
qz[ 79: 64] = { {8{DataIn[103]}}, DataIn[103: 96] }
qz[ 95: 80] = { {8{DataIn[111]}}, DataIn[111:104] }
qz[111: 96] = { {8{DataIn[119]}}, DataIn[119:112] }
qz[127:112] = { {8{DataIn[127]}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

格式

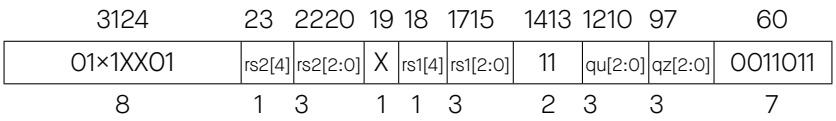


图 4.7-294. ESP.VLDEXT.S8.XP

4.7.2.295 ESP.VLDEXT.U16.IP

语法

ESP.VLDEXT.U16.IP qu,qz,rs1,imm16

描述

此指令从内存中加载 128 位数据，并将数据从 16 位段扩展到 32 位段。imm从-2048 开始，结束于 2032，步长为 16。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 31:  0] = { {16{0}}, DataIn[ 15:  0] }
qu[ 63: 32] = { {16{0}}, DataIn[ 31: 16] }
qu[ 95: 64] = { {16{0}}, DataIn[ 47: 32] }
qu[127: 96] = { {16{0}}, DataIn[ 63: 48] }
qz[ 31:  0] = { {16{0}}, DataIn[ 79: 64] }
qz[ 63: 32] = { {16{0}}, DataIn[ 95: 80] }
qz[ 95: 64] = { {16{0}}, DataIn[111: 96] }
qz[127: 96] = { {16{0}}, DataIn[127:112] }
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1

格式

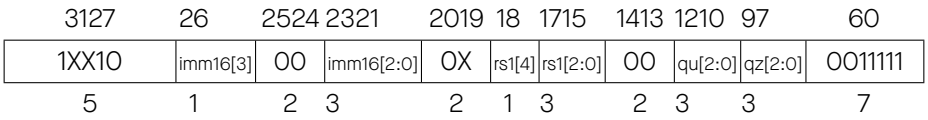


图 4.7-295. ESP.VLDEXT.U16.IP

4.7.2.296 ESP.VLDEXT.U16.XP

语法

ESP.VLDEXT.U16.XP qu,qz,rs1,rs2

描述

此指令从内存中加载 128 位数据，并将数据从 16 位段扩展到 32 位段。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 31:  0] = { {16{0}}, DataIn[ 15:  0] }
qu[ 63: 32] = { {16{0}}, DataIn[ 31: 16] }
qu[ 95: 64] = { {16{0}}, DataIn[ 47: 32] }
qu[127: 96] = { {16{0}}, DataIn[ 63: 48] }
qz[ 31:  0] = { {16{0}}, DataIn[ 79: 64] }
qz[ 63: 32] = { {16{0}}, DataIn[ 95: 80] }
qz[ 95: 64] = { {16{0}}, DataIn[111: 96] }
qz[127: 96] = { {16{0}}, DataIn[127:112] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

格式

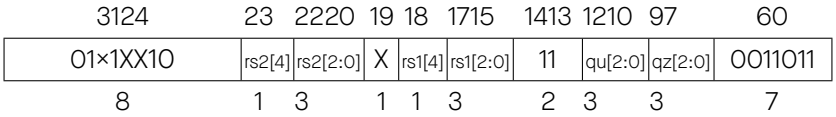


图 4.7-296. ESP.VLDEXT.U16.XP

4.7.2.297 ESP.VLDEXT.U8.IP

语法

ESP.VLDEXT.U8.IP qu,qz,rs1,imm16

描述

此指令从内存中加载 128 位数据，并将数据从 8 位段扩展到 16 位段。

`imm` 从-2048 开始，结束于 2032，步长为 16。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{0}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{0}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{0}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{0}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{0}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{0}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{0}}, DataIn[ 55: 48] }
qu[127:112] = { {8{0}}, DataIn[ 63: 56] }
qz[ 15:  0] = { {8{0}}, DataIn[ 71: 64] }
qz[ 31: 16] = { {8{0}}, DataIn[ 79: 72] }
qz[ 47: 32] = { {8{0}}, DataIn[ 87: 80] }
qz[ 63: 48] = { {8{0}}, DataIn[ 95: 88] }
qz[ 79: 64] = { {8{0}}, DataIn[103: 96] }
qz[ 95: 80] = { {8{0}}, DataIn[111:104] }
qz[111: 96] = { {8{0}}, DataIn[119:112] }
qz[127:112] = { {8{0}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1

格式

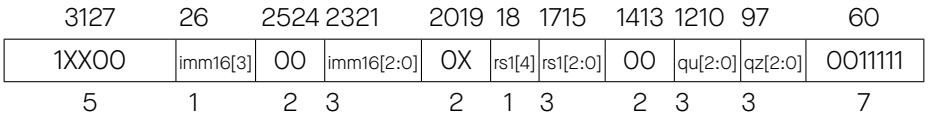


图 4.7-297. ESP.VLDEXT.U8.IP

4.7.2.298 ESP.VLDEXT.U8.XP

语法

ESP.VLDEXT.U8.XP qu,qz,rs1,rs2

描述

此指令从内存中加载 128 位数据，并将数据从 8 位段扩展到 16 位段。

操作

```
DataIn[127:0] = load128(rs1[31:0])
qu[ 15:  0] = { {8{0}}, DataIn[  7:  0] }
qu[ 31: 16] = { {8{0}}, DataIn[ 15:  8] }
qu[ 47: 32] = { {8{0}}, DataIn[ 23: 16] }
qu[ 63: 48] = { {8{0}}, DataIn[ 31: 24] }
qu[ 79: 64] = { {8{0}}, DataIn[ 39: 32] }
qu[ 95: 80] = { {8{0}}, DataIn[ 47: 40] }
qu[111: 96] = { {8{0}}, DataIn[ 55: 48] }
qu[127:112] = { {8{0}}, DataIn[ 63: 56] }
qz[ 15:  0] = { {8{0}}, DataIn[ 71: 64] }
qz[ 31: 16] = { {8{0}}, DataIn[ 79: 72] }
qz[ 47: 32] = { {8{0}}, DataIn[ 87: 80] }
qz[ 63: 48] = { {8{0}}, DataIn[ 95: 88] }
qz[ 79: 64] = { {8{0}}, DataIn[103: 96] }
qz[ 95: 80] = { {8{0}}, DataIn[111:104] }
qz[111: 96] = { {8{0}}, DataIn[119:112] }
qz[127:112] = { {8{0}}, DataIn[127:120] }
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qv	-	2
rs1	1	1
rs2	1	-

格式

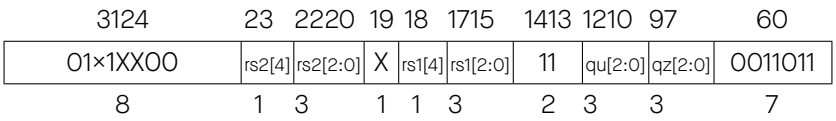


图 4.7-298. ESP.VLDEXT.U8.XP

4.7.2.299 ESP.VLDHBC.16.INCP

语法

ESP.VLDHBC.16.INCP qu,qz,rs1

描述

此指令从内存中加载 16 字节的数据，按照以下方式将其扩展到 256 位，并将结果赋值给寄存器 qu 和 qz。访问后，寄存器 rs1 中的值将增加 16。

操作

```
dataIn[127:0] = load128(rs1[31:0])
qu  = {2{dataIn[ 63: 48]}, 2{dataIn[ 47: 32]}, 2{dataIn[ 31: 16]}, 2{dataIn[ 15: 0]} }
qz = {2{dataIn[127:112]}, 2{dataIn[111: 96]}, 2{dataIn[ 95: 80]}, 2{dataIn[ 79: 64]} }
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qz	-	2
rs1	1	1

格式

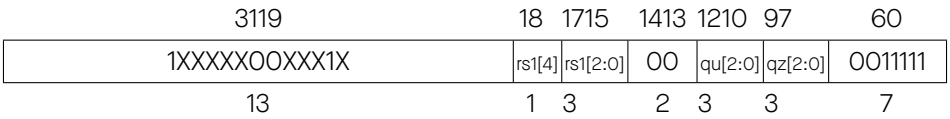


图 4.7-299. ESP.VLDHBC.16.INCP

4.7.2.300 ESP.LD.QACC.H.H.128.IP

语法

ESP.LD.QACC.H.H.128.IP rs1,imm16

描述

此指令将 16 字节的数据从内存加载到特殊寄存器 `QACC_H[255:128]` 中。访问完成后，寄存器 `rs1` 中的值会增加指令代码段中的一个 8 位带符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
QACC_H[255:  128] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
rs1	1	1

格式

3127	26	2524	2321	2019	18	1715	1413	129	87	60
0XX01	imm16[7]	00	imm16[6:4]	0X	rs1[4]	rs1[2:0]	00	imm16[3:0]	XX	0011111
5	1	2	3	2	1	3	2	4	2	7

图 4.7-300. ESP.LD.QACC.H.H.128.IP

4.7.2.301 ESP.LD.QACC.H.L.128.IP

语法

ESP.LD.QACC.H.L.128.IP rs1,imm16

描述

此指令将 16 字节数据从内存加载到特殊寄存器 `QACC_H[127:0]`。访问完成后，寄存器 `rs1` 中的值将增加指令代码段中的一个 8 位符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
QACC_H[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	-	2
rs1	1	1

格式

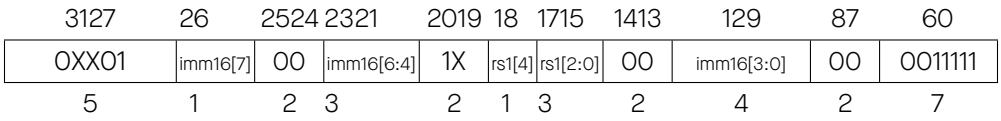


图 4.7-301. ESP.LD.QACC.H.L.128.IP

4.7.2.302 ESP.LD.QACC.L.H.128.IP

语法

ESP.LD.QACC.L.H.128.IP rs1,imm16

描述

此指令将 16 字节的数据从内存加载到特殊寄存器 QACC_L[255:128] 中。访问完成后，寄存器 rs1 中的值将增加指令代码段中的一个 8 位符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
QACC_L[255:  128] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_L	-	2
rs1	1	1

格式

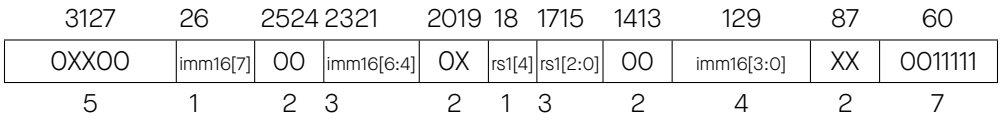


图 4.7-302. ESP.LD.QACC.L.H.128.IP

4.7.2.303 ESP.LD.QACC.L.L.128.IP

语法

ESP.LD.QACC.L.L.128.IP rs1,imm16

描述

此指令将 16 字节的数据从内存加载到特殊寄存器 `QACC_L[127:0]` 中。访问完成后，寄存器 `rs1` 中的值会增加指令代码段中的一个 8 位符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
QACC_L[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_L	-	2
rs1	1	1

格式

31–27	26	25–24	23–21	20–19	18	17–15	14–13	12–9	8–7	6–0
0XX00	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	00	0011111
5	1	2	3	2	1	3	2	4	2	7

图 4.7-303. ESP.LD.QACC.L.L.128.IP

4.7.2.304 ESP.LD.UA.STATE.IP

语法

ESP.LD.UA.STATE.IP rs1,imm16

描述

此指令将 16 字节的数据从内存加载到特殊寄存器 UA_STATE[127:0] 中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位带符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
UA_STATE[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
UA_STATE	-	2
rs1	1	1

格式

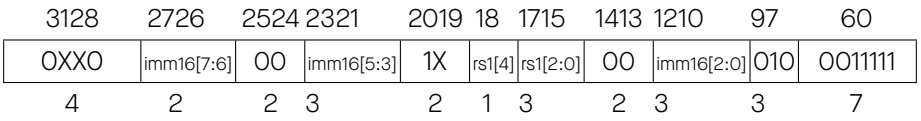


图 4.7-304. ESP.LD.UA.STATE.IP

4.7.2.305 ESP.LDXQ.32

语法

ESP.LDXQ.32 qu,qw,rs1,sel4,sel8

描述

此指令根据立即数 sel8 从 qw 中选择 8 个 16 位数据段中的一个作为加数。它将加数左移 2 位后与地址寄存器 rs1 的值进行相加，使用结果作为访问地址，并根据立即数 sel4 的值将加载的数据存储在寄存器 qu 中的一个 32 位数据段中。

操作

```
  vaddr0[31:0] = rs1[31:0] + qw[ 15:  0] * 4
  vaddr1[31:0] = rs1[31:0] + qw[ 31: 16] * 4
  vaddr2[31:0] = rs1[31:0] + qw[ 47: 32] * 4
  vaddr3[31:0] = rs1[31:0] + qw[ 63: 47] * 4
  vaddr4[31:0] = rs1[31:0] + qw[ 79: 64] * 4
  vaddr5[31:0] = rs1[31:0] + qw[ 95: 80] * 4
  vaddr6[31:0] = rs1[31:0] + qw[111: 96] * 4
  vaddr7[31:0] = rs1[31:0] + qw[127:112] * 4

if sel8 == 0:
  dataIn[31:0] = load32(vaddr0[31:0])
if sel8 == 1:
  dataIn[31:0] = load32(vaddr1[31:0])
if sel8 == 2:
  dataIn[31:0] = load32(vaddr2[31:0])
if sel8 == 3:
  dataIn[31:0] = load32(vaddr3[31:0])
if sel8 == 4:
  dataIn[31:0] = load32(vaddr4[31:0])
if sel8 == 5:
  dataIn[31:0] = load32(vaddr5[31:0])
if sel8 == 6:
  dataIn[31:0] = load32(vaddr6[31:0])
if sel8 == 7:
  dataIn[31:0] = load32(vaddr7[31:0])

  qu[32*sel4+31:32*sel4] = dataIn[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
qw	0	-
rs1	1	-

格式

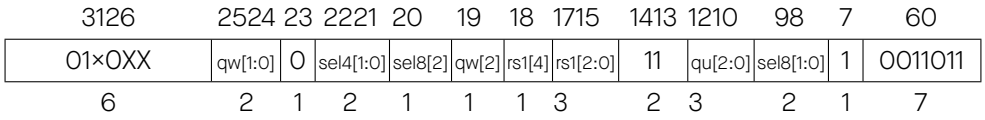


图 4.7-305. ESP.LDXQ.32

4.7.2.306 ESP.ST.QACC.H.H.128.IP

语法

ESP.ST.QACC.H.H.128.IP rs1,imm16

描述

此指令将特殊寄存器 QACC_H 中的上 128 位存储到内存中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位带符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
MEM(rs1[31:0]) = QACC_H[255:128]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	2	-
rs1	1	1

格式

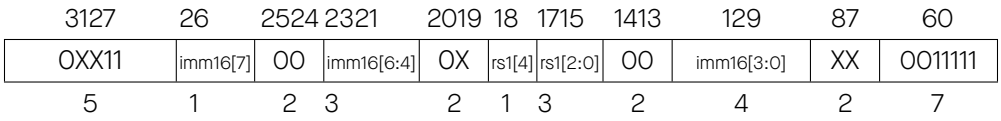


图 4.7-306. ESP.ST.QACC.H.H.128.IP

4.7.2.307 ESP.ST.QACC.H.L.128.IP

语法

ESP.ST.QACC.H.L.128.IP rs1,imm16

描述

此指令将特殊寄存器 QACC_H 中的低 128 位存储到内存中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位带符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
MEM(rs1[31:0]) = QACC_H[127:0]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_H	2	-
rs1	1	1

格式

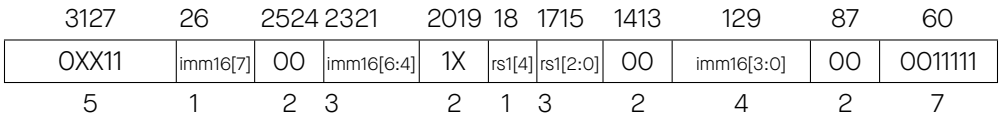


图 4.7-307. ESP.ST.QACC.H.L.128.IP

4.7.2.308 ESP.ST.QACC.L.H.128.IP

语法

ESP.ST.QACC.L.H.128.IP rs1,imm16

描述

此指令将特殊寄存器 QACC_L 中的上 128 位存储到内存中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位带符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
MEM(rs1[31:0]) = QACC_L[255:128]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_L	2	-
rs1	1	1

格式

3127	26	2524	2321	2019	18	1715	1413	129	87	60
0XX10	imm16[7]	00	imm16[6:4]	0X	rs1[4]	rs1[2:0]	00	imm16[3:0]	XX	0011111
5	1	2	3	2	1	3	2	4	2	7

图 4.7-308. ESP.ST.QACC.L.H.128.IP

4.7.2.309 ESP.ST.QACC.L.L.128.IP

语法

ESP.ST.QACC.L.L.128.IP rs1,imm16

描述

此指令将特殊寄存器 QACC_L 中的低 128 位存储到内存中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位带符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
MEM(rs1[31:0]) = QACC_L[127:0]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
QACC_L	2	-
rs1	1	1

格式

3127	26	2524	2321	2019	18	1715	1413	129	87	60
0XX10	imm16[7]	00	imm16[6:4]	1X	rs1[4]	rs1[2:0]	00	imm16[3:0]	00	0011111
5	1	2	3	2	1	3	2	4	2	7

图 4.7-309. ESP.ST.QACC.L.L.128.IP

4.7.2.310 ESP.ST.UA.STATE.IP

语法

ESP.ST.UA.STATE.IP rs1,imm16

描述

此指令将特殊寄存器 UA_STATE 中的 128 位数据存储到内存中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位带符号扩展常数，然后左移 4 位。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
MEM(rs1[31:0]) = UA_STATE[127:0]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
UA_STATE	2	-
rs1	1	1

格式

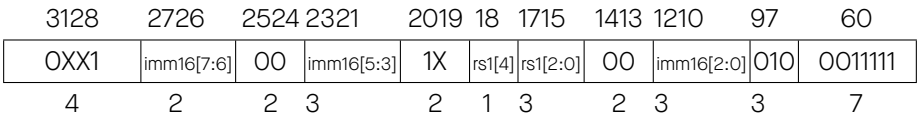


图 4.7-310. ESP.ST.UA.STATE.IP

4.7.2.311 ESP.STXQ.32

语法

ESP.STXQ.32 qu,qw,rs1,sel4,sel8

描述

此指令根据立即数 `sel4` 从寄存器 `qw` 中选择四个 32 位数据段中的一个。然后，它根据立即数 `sel8` 从寄存器 `qw` 中选择八个 16 位数据段中的一个加数，将加数左移 2 位后加到地址寄存器 `rs1` 的值上，并使用结果作为写地址。

操作

```
  vaddr0[31:0] = rs1[31:0] + qw[ 15:  0] * 4
  vaddr1[31:0] = rs1[31:0] + qw[ 31: 16] * 4
  vaddr2[31:0] = rs1[31:0] + qw[ 47: 32] * 4
  vaddr3[31:0] = rs1[31:0] + qw[ 63: 47] * 4
  vaddr4[31:0] = rs1[31:0] + qw[ 79: 64] * 4
  vaddr5[31:0] = rs1[31:0] + qw[ 95: 80] * 4
  vaddr6[31:0] = rs1[31:0] + qw[111: 96] * 4
  vaddr7[31:0] = rs1[31:0] + qw[127:112] * 4

if sel8 == 0:
  qu[32*sel4+31:32*sel4] =>store32(vaddr0[31:0])
if sel8 == 1:
  qu[32*sel4+31:32*sel4] =>store32(vaddr1[31:0])
if sel8 == 2:
  qu[32*sel4+31:32*sel4] =>store32(vaddr2[31:0])
if sel8 == 3:
  qu[32*sel4+31:32*sel4] =>store32(vaddr3[31:0])
if sel8 == 4:
  qu[32*sel4+31:32*sel4] =>store32(vaddr4[31:0])
if sel8 == 5:
  qu[32*sel4+31:32*sel4] =>store32(vaddr5[31:0])
if sel8 == 6:
  qu[32*sel4+31:32*sel4] =>store32(vaddr6[31:0])
if sel8 == 7:
  qu[32*sel4+31:32*sel4] =>store32(vaddr7[31:0])
```

调度

操作数	读流水级数	写流水级数
qu	2	-
qw	0	-
rs1	1	1

格式

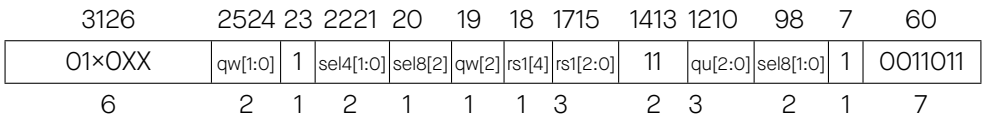


图 4.7-311. ESP.STXQ.32

4.7.2.312 ESP.VLD.128.IP

语法

ESP.VLD.128.IP qu,rs1,imm16

描述

此指令将 128 位数据从内存加载到寄存器qu，并通过立即数增加指针。

imm 的起始值为-2048，结束值为 2032，步长为 16。

操作

```
qu[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1

格式

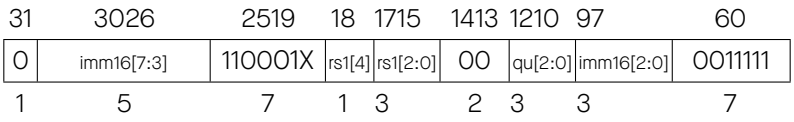


图 4.7-312. ESP.VLD.128.IP

4.7.2.313 ESP.VLD.128.XP

语法

ESP.VLD.128.XP qu,rs1,rs2

描述

此指令将 128 位数据从内存加载到寄存器qu中，指针按寄存器rs2的值增加。

操作

```
qu[127:  0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-

格式

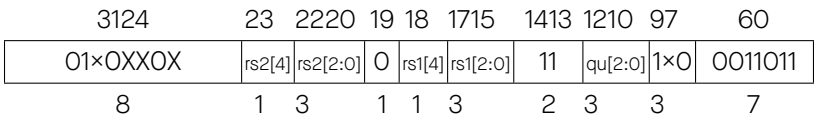


图 4.7-313. ESP.VLD.128.XP

4.7.2.314 ESP.VLD.H.64.IP

语法

ESP.VLD.H.64.IP qu,rs1,imm8

描述

此指令将 64 位数据从内存加载到寄存器qu的上 64 位段中，并通过立即数增加指针。
imm 从-1024 开始，结束于 1016，并以 8 为步长。

操作

```
qu[ 127:  64] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1

格式

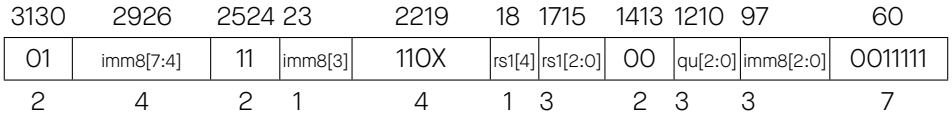


图 4.7-314. ESP.VLD.H.64.IP

4.7.2.315 ESP.VLD.H.64.XP

语法

ESP.VLD.H.64.XP qu,rs1,rs2

描述

此指令从内存中加载 64 位数据到寄存器qu的上 64 位段，指针由寄存器rs2中的值增加。

操作

```
qu[ 127:  64] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-

格式

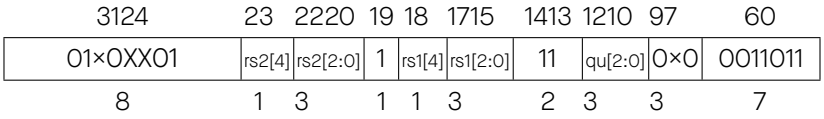


图 4.7-315. ESP.VLD.H.64.XP

4.7.2.316 ESP.VLD.L.64.IP

语法

ESP.VLD.L.64.IP qu,rs1,imm8

描述

此指令将 64 位数据从内存加载到寄存器qu的下 64 位段中，指针由立即数增加。
imm 从-1024 开始，到 1016 结束，步长为 8。

操作

```
qu[ 63:  0] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1

格式

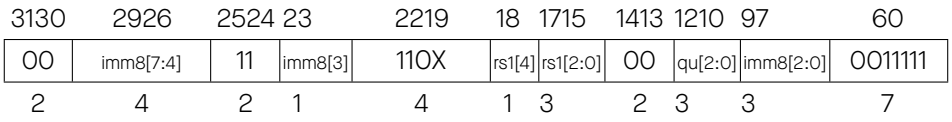


图 4.7-316. ESP.VLD.L.64.IP

4.7.2.317 ESP.VLD.L.64.XP

语法

ESP.VLD.L.64.XP qu,rs1,rs2

描述

此指令从内存中加载 64 位数据到qu寄存器的低 64 位段，指针由rs2寄存器中的值增加。

操作

```
qu[ 63:  0] = load64(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	1
rs2	1	-

格式

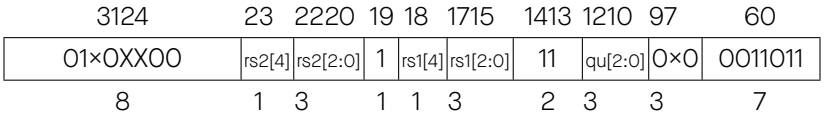


图 4.7-317. ESP.VLD.L.64.XP

4.7.2.318 ESP.VST.128.IP

语法

ESP.VST.128.IP qu,rs1,imm16

描述

此指令将寄存器 qu 中的 128 位存储到内存中，并通过立即数增加指针。

imm 的起始值为-2048，结束值为 2032，步进值为 16。

操作

```
MEM(rs1[31:0]) = qu[127:  0]
rs1[31:0] = rs1[31:0] +imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1

格式

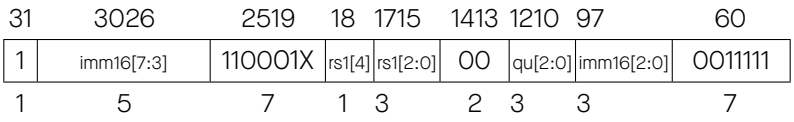


图 4.7-318. ESP.VST.128.IP

4.7.2.319 ESP.VST.128.XP

语法

ESP.VST.128.XP qu,rs1,rs2

描述

此指令将寄存器 qu 中的 128 位存储到内存中，并通过寄存器 rs2 中的值增加指针。

操作

```
MEM(rs1[31:0]) = qu[127:  0]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1

格式

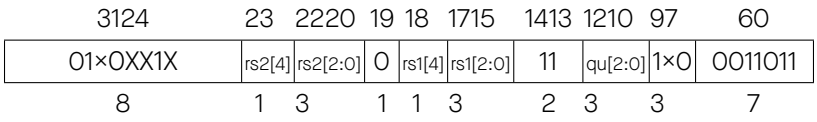


图 4.7-319. ESP.VST.128.XP

4.7.2.320 ESP.VST.H.64.IP

语法

ESP.VST.H.64.IP qu,rs1,imm8

描述

此指令将寄存器 qu 的上 64 位存储到内存中，指针通过立即数增加。

imm 从 -1024 开始，到 1016 结束，步长为 8。

操作

```
MEM(rs1[31:0]) = qu[127:  64]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1

格式

3130	2926	2524	23	2219	18	1715	1413	1210	97	60
11	imm8[7:4]	11	imm8[3]	110X	rs1[4]	rs1[2:0]	00	qu[2:0]	imm8[2:0]	0011111
2	4	2	1	4	1	3	2	3	3	7

图 4.7-320. ESP.VST.H.64.IP

4.7.2.321 ESP.VST.H.64.XP

语法

ESP.VST.H.64.XP qu,rs1,rs2

描述

此指令将寄存器 qu 中的上 64 位存储到内存中，指针由寄存器 rs2 中的值增加。

操作

```
MEM(rs1[31:0]) = qu[127:  64]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-

格式

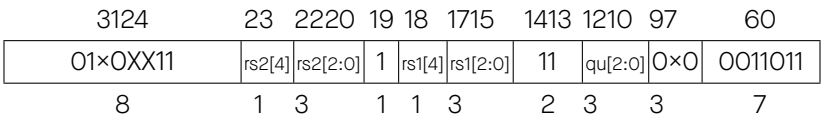


图 4.7-321. ESP.VST.H.64.XP

4.7.2.322 ESP.VST.L.64.IP

语法

ESP.VST.L.64.IP qu,rs1,imm8

描述

此指令将寄存器 qu 的低 64 位中的 64 位存储到内存中，并通过立即数增加指针。

imm 从 -1024 开始，到 1016 结束，步长为 8。

操作

```
MEM(rs1[31:0]) = qu[63:  0]
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1

格式

31..30	29..26	25..24	23	22..19	18	17..15	14..13	12..10	9..7	6..0
10	imm8[7:4]	11	imm8[3]	110X	rs1[4]	rs1[2:0]	00	qu[2:0]	imm8[2:0]	0011111
2	4	2	1	4	1	3	2	3	3	7

图 4.7-322. ESP.VST.L.64.IP

4.7.2.323 ESP.VST.L.64.XP

语法

ESP.VST.L.64.XP qu,rs1,rs2

描述

此指令将寄存器 qu 的低 64 位中的 64 位存储到内存中，指针由寄存器 rs2 中的值增加。

操作

```
MEM(rs1[31:0]) = qu[63:  0]
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qu	2	-
rs1	1	1
rs2	1	-

格式

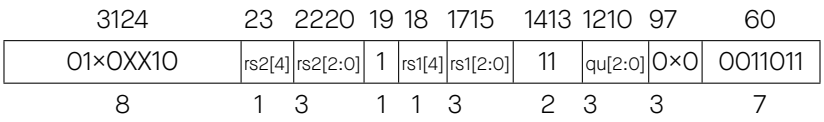


图 4.7-323. ESP.VST.L.64.XP

4.7.2.324 ESP.SLCI.2Q

语法

ESP.SLCI.2Q qy,qw,sel16

描述

此指令对寄存器qw和qy的 32 字节串联进行左移，并用 0 填充低位。移位结果的上 128 位写入寄存器qy，下 128 位写入qw。左移的数量是sel16和 1 的总和的 8 倍。

操作

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} << ((sel16[3:0]+1)*8)
```

调度

操作数	读流水级数	写流水级数
qy	1	1
qw	1	1

格式

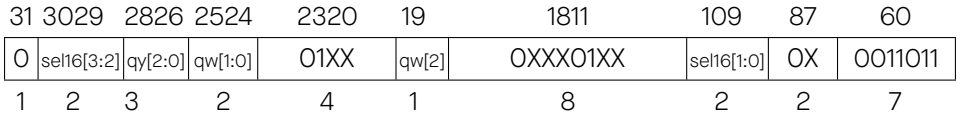


图 4.7-324. ESP.SLCI.2Q

4.7.2.325 ESP.SLCXXP.2Q

语法

ESP.SLCXXP.2Q qy,qw,rs1,rs2

描述

此指令对寄存器qy和qw的 32 字节串联进行左移，并用 0 填充低位。移位结果的上 128 位写入寄存器qy，下 128 位写入寄存器qw。左移的数量为 8，由寄存器rs1的低 4 位值加 1 得出。在上述操作之后，rs1中的值会增加rs2中的值。

操作

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} << ((rs1[3:0]+1)*8)
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qy	1	1
qw	1	1
rs1	1	1
rs2	1	-

格式

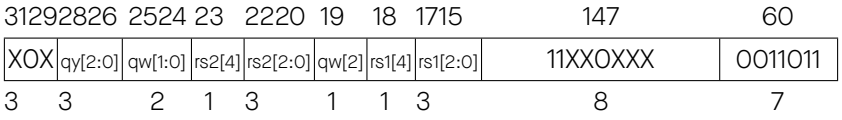


图 4.7-325. ESP.SLCXXP.2Q

4.7.2.326 ESP.SRC.Q

语法

ESP.SRC.Q qz,qw,qy

描述

此指令对寄存器 qy 和 qw 的 32 字节串联进行算术右移，这两个寄存器持有从两个连续对齐地址加载的数据。通过这种方式，您可以获得未对齐的 16 字节数据，然后将其写入寄存器 qw。右移的数量是由 8 执行的 SAR_BYTE。

操作

```
qz[127:0] = {qy[127: 0], qw[127: 0]} >> {SAR_BYTE[3:0] << 3}
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-

格式

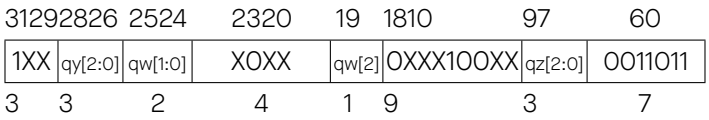


图 4.7-326. ESP.SRC.Q

4.7.2.327 ESP.SRC.Q.LD.IP

语法

ESP.SRC.Q.LD.IP qu,rs1,imm16,qw,qy

描述

此指令对寄存器qy和qw的 32 字节串联进行算术右移，这些寄存器保存了从两个连续对齐地址加载的数据。通过这种方式，您可以获得未对齐的 16 字节数据，该数据将被写入寄存器qw。右移的数量是由 8 执行的SAR_BYTE。

同时，16 字节数据从内存加载到寄存器qu。访问完成后，寄存器rs1中的值将增加imm。

imm从-2048 开始，到 2032 结束，步长为 16。

操作

```
qw[127:0] = {qy[127: 0], qw[127: 0]} >> {SAR_BYTE[3:0] << 3}
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-
qu	-	2
rs1	1	1

格式

3129	2826	2524	2320	19	18	1715	1413	1210	9	87	60
imm16[7:5]	qy[2:0]	qw[1:0]	imm16[4:1]	qw[2]	rs1[4]	rs1[2:0]	10	qu[2:0]	imm16[0]	00	0011111
3	3	2	4	1	1	3	2	3	1	2	7

图 4.7-327. ESP.SRC.Q.LD.IP

4.7.2.328 ESP.SRC.Q.LD.XP

语法

ESP.SRC.Q.LD.XP qu,rs1,rs2,qw,qy

描述

此指令对寄存器qy和qw的 32 字节串联进行算术右移，这些寄存器保存了从两个连续对齐地址加载的数据。通过这种方式，您可以获得未对齐的 16 字节数据，这将被写入寄存器qw。右移的数量是SAR_BYTE和 8 的乘积。

同时，16 字节数据从内存加载到寄存器qu。访问完成后，寄存器rs1中的值将增加寄存器rs2中的值。

操作

```
qw[127:0] = {qy[127: 0], qw[127: 0]} >> {SAR_BYTE[3:0] << 3}
qu[127:0] = load128(rs1[31:0])
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-
qu	-	2
rs1	1	1
rs2	1	-

格式

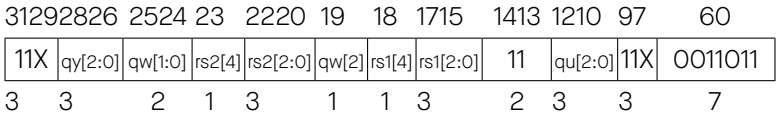


图 4.7-328. ESP.SRC.Q.LD.XP

4.7.2.329 ESP.SRCI.2Q

语法

ESP.SRCI.2Q qy,qw,sel16

描述

此指令对寄存器 qw 和 qy 的 32 字节串联进行逻辑右移，并用 0 填充高位。移位结果的上 128 位写入寄存器 qy，下 128 位写入 qw。右移的数量是 sel16 和 1 的和的 8 倍。

操作

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} >> ((sel16+1)*8)
qy[127:127-8*sel16] = 0
```

调度

操作数	读流水级数	写流水级数
qz	1	1
qw	1	1

格式

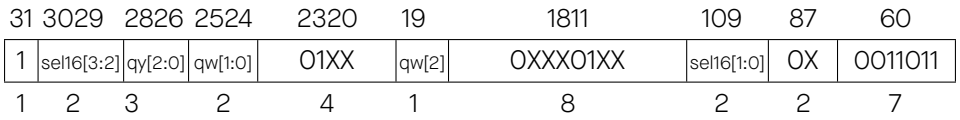


图 4.7-329. ESP.SRCI.2Q

4.7.2.330 ESP.SRCMB.S16.Q.QACC

语法

ESP.SRCMB.S16.Q.QACC qu,qw,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 8 个 64 位的数据段，并分别进行算术右移操作，移位量来自 qw。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 16 位有符号数，并将饱和后得到的 8 个 16 位数据写入寄存器 qu。

操作

```
shf_value0[5:0] = (qw[ 15: 0] >= 0x3f) ? 0x3f : qw[ 15: 0]
shf_value1[5:0] = (qw[ 31: 16] >= 0x3f) ? 0x3f : qw[ 31: 16]
shf_value2[5:0] = (qw[ 47: 32] >= 0x3f) ? 0x3f : qw[ 47: 32]
shf_value3[5:0] = (qw[ 63: 48] >= 0x3f) ? 0x3f : qw[ 63: 48]
shf_value4[5:0] = (qw[ 79: 64] >= 0x3f) ? 0x3f : qw[ 79: 64]
shf_value5[5:0] = (qw[ 95: 80] >= 0x3f) ? 0x3f : qw[ 95: 80]
shf_value6[5:0] = (qw[111: 96] >= 0x3f) ? 0x3f : qw[111: 96]
shf_value7[5:0] = (qw[127:112] >= 0x3f) ? 0x3f : qw[127:112]

DATA_L[ 63:  0] = QACC_L[ 63:  0] >> shf_value0[5:0]
DATA_L[127: 64] = QACC_L[127: 64] >> shf_value1[5:0]
DATA_L[191:128] = QACC_L[191:128] >> shf_value2[5:0]
DATA_L[255:192] = QACC_L[255:192] >> shf_value3[5:0]
DATA_H[ 63:  0] = QACC_H[ 63:  0] >> shf_value4[5:0]
DATA_H[127: 64] = QACC_H[127: 64] >> shf_value5[5:0]
DATA_H[191:128] = QACC_H[191:128] >> shf_value6[5:0]
DATA_H[255:192] = QACC_H[255:192] >> shf_value7[5:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[15: 0] = DATA_L[15:0]
  qu[31:16] = DATA_L[79:64]
  qu[47:32] = DATA_L[143:128]
  qu[63:48] = DATA_L[207:192]
  qu[79:64] = DATA_H[15:0]
  qu[95:80] = DATA_H[79:64]
  qu[111:96] = DATA_H[143:128]
  qu[127:112] = DATA_H[207:192]
else if sel2 == 1:
  qu[ 15:  0] = min(max(DATA_L[63:0], -2^{15}), 2^{15}-1)
  qu[ 31: 16] = min(max(DATA_L[127:64], -2^{15}), 2^{15}-1)
  qu[ 47: 32] = min(max(DATA_L[191:128], -2^{15}), 2^{15}-1)
  qu[ 63: 48] = min(max(DATA_L[255:192], -2^{15}), 2^{15}-1)
  qu[ 79: 64] = min(max(DATA_H[63:0], -2^{15}), 2^{15}-1)
  qu[ 95: 80] = min(max(DATA_H[127:64], -2^{15}), 2^{15}-1)
  qu[111: 96] = min(max(DATA_H[191:128], -2^{15}), 2^{15}-1)
  qu[127:112] = min(max(DATA_H[255:192], -2^{15}), 2^{15}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

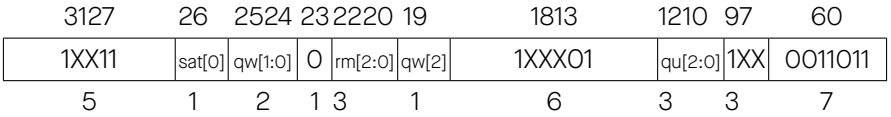


图 4.7-330. ESP.SRCMB.S16.Q.QACC

4.7.2.331 ESP.SRCMB.S16.QACC

语法

ESP.SRCMB.S16.QACC qu,rs1,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 8 个 64 位的数据段，并分别执行算术右移操作。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 16 位有符号数，并将饱和后得到的 8 个 16 位数据段写入 qu 寄存器。

操作

```
temp0[63:0] = QACC_L[ 63:  0]
...
temp3[63:0] = QACC_L[255:192]
temp4[63:0] = QACC_H[ 63:  0]
...
temp7[63:0] = QACC_H[255:192]

temp_shf0[63:0] = temp0[63:0] >> rs1[5:0]
temp_shf1[63:0] = temp1[63:0] >> rs1[5:0]
...
temp_shf7[63:0] = temp7[63:0] >> rs1[5:0]

QACC_L[ 63:  0] = temp_shf0[63:0]
...
QACC_L[255:192] = temp_shf3[63:0]
QACC_H[ 63:  0] = temp_shf4[63:0]
...
QACC_H[255:192] = temp_shf7[63:0]

if sel2 == 0:
  qu[ 15:  0] = temp0[15:0]
  qu[ 31: 16] = temp1[15:0]
  qu[ 47: 32] = temp2[15:0]
  qu[ 63: 48] = temp3[15:0]
  qu[ 79: 64] = temp4[15:0]
  qu[ 95: 80] = temp5[15:0]
  qu[111: 96] = temp6[15:0]
  qu[127:112] = temp7[15:0]
else if sel2==1:
  qu[ 15:  0] = min(max(temp_shf0[63:0], -2^{15}), 2^{15}-1)
  ...
  qu[ 63: 48] = min(max(temp_shf3[63:0], -2^{15}), 2^{15}-1)
  qu[ 79: 64] = min(max(temp_shf4[63:0], -2^{15}), 2^{15}-1)
  ...
  qu[127:112] = min(max(temp_shf7[63:0], -2^{15}), 2^{15}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2

操作数	读流水级数	写流水级数
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

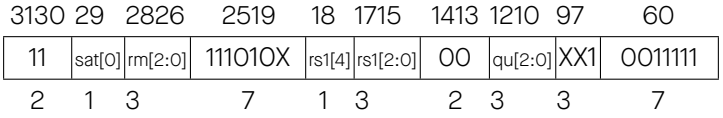


图 4.7-331. ESP.SRCMB.S16.QACC

4.7.2.332 ESP.SRCMB.S8.Q.QACC

语法

ESP.SRCMB.S8.Q.QACC qu,qw,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 16 个 32 位的数据段，并分别进行算术右移操作，移位量来自 qw。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 8 位有符号数，并将饱和后得到的 16 个 8 位数据写入寄存器 qu。

操作

```
shf_value0[5:0] = (qw[ 7: 0] >= 0x1f) ? 0x1f : qw[ 7: 0]
shf_value1[5:0] = (qw[ 15: 8] >= 0x1f) ? 0x1f : qw[ 15: 8]
shf_value2[5:0] = (qw[ 23: 16] >= 0x1f) ? 0x1f : qw[ 23: 16]
shf_value3[5:0] = (qw[ 31: 24] >= 0x1f) ? 0x1f : qw[ 31: 24]
shf_value4[5:0] = (qw[ 39: 32] >= 0x1f) ? 0x1f : qw[ 39: 32]
shf_value5[5:0] = (qw[ 47: 40] >= 0x1f) ? 0x1f : qw[ 47: 40]
shf_value6[5:0] = (qw[ 55: 48] >= 0x1f) ? 0x1f : qw[ 55: 48]
shf_value7[5:0] = (qw[ 63: 56] >= 0x1f) ? 0x1f : qw[ 63: 56]
shf_value8[5:0] = (qw[ 71: 64] >= 0x1f) ? 0x1f : qw[ 71: 64]
shf_value9[5:0] = (qw[ 79: 72] >= 0x1f) ? 0x1f : qw[ 79: 72]
shf_value10[5:0] = (qw[ 87: 80] >= 0x1f) ? 0x1f : qw[ 87: 80]
shf_value11[5:0] = (qw[ 95: 88] >= 0x1f) ? 0x1f : qw[ 95: 88]
shf_value12[5:0] = (qw[103: 96] >= 0x1f) ? 0x1f : qw[103: 96]
shf_value13[5:0] = (qw[111:104] >= 0x1f) ? 0x1f : qw[111:104]
shf_value14[5:0] = (qw[119:112] >= 0x1f) ? 0x1f : qw[119:112]
shf_value15[5:0] = (qw[127:120] >= 0x1f) ? 0x1f : qw[127:120]

DATA_L[ 31:  0] = QACC_L[ 31:  0] >> shf_value0[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> shf_value1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> shf_value2[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> shf_value3[4:0]
DATA_L[159:128] = QACC_L[159:128] >> shf_value4[4:0]
DATA_L[191:160] = QACC_L[191:160] >> shf_value5[4:0]
DATA_L[223:192] = QACC_L[223:192] >> shf_value6[4:0]
DATA_L[255:224] = QACC_L[255:224] >> shf_value7[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> shf_value8[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> shf_value9[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> shf_value10[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> shf_value11[4:0]
DATA_H[159:128] = QACC_H[159:128] >> shf_value12[4:0]
DATA_H[191:160] = QACC_H[191:160] >> shf_value13[4:0]
DATA_H[223:192] = QACC_H[223:192] >> shf_value14[4:0]
DATA_H[255:224] = QACC_H[255:224] >> shf_value15[4:0]


QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[  7:  0] = DATA_L[  7:  0]
```

```
    qu[ 15:  8] = DATA_L[ 39: 32]
    qu[ 23: 16] = DATA_L[ 71: 64]
    qu[ 31: 24] = DATA_L[103: 96]
    qu[ 39: 32] = DATA_L[135:128]
    qu[ 47: 40] = DATA_L[167:160]
    qu[ 55: 48] = DATA_L[199:192]
    qu[ 63: 56] = DATA_L[231:224]
    qu[ 71: 64] = DATA_H[  7:  0]
    qu[ 79: 72] = DATA_H[ 39: 32]
    qu[ 87: 80] = DATA_H[ 71: 64]
    qu[ 95: 88] = DATA_H[103: 96]
    qu[103: 96] = DATA_H[135:128]
    qu[111:104] = DATA_H[167:160]
    qu[119:112] = DATA_H[199:192]
    qu[127:120] = DATA_H[231:224]
  else if sel2 == 1:
    qu[  7:  0] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
    qu[ 15:  8] = min(max(DATA_L[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 23: 16] = min(max(DATA_L[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 31: 24] = min(max(DATA_L[127: 96], -2^{7}), 2^{7}-1)
    qu[ 39: 32] = min(max(DATA_L[159:128], -2^{7}), 2^{7}-1)
    qu[ 47: 40] = min(max(DATA_L[191:160], -2^{7}), 2^{7}-1)
    qu[ 55: 48] = min(max(DATA_L[223:192], -2^{7}), 2^{7}-1)
    qu[ 63: 56] = min(max(DATA_L[255:224], -2^{7}), 2^{7}-1)
    qu[ 71: 64] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
    qu[ 79: 72] = min(max(DATA_H[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 87: 80] = min(max(DATA_H[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 95: 88] = min(max(DATA_H[127: 96], -2^{7}), 2^{7}-1)
    qu[103: 96] = min(max(DATA_H[159:128], -2^{7}), 2^{7}-1)
    qu[111:104] = min(max(DATA_H[191:160], -2^{7}), 2^{7}-1)
    qu[119:112] = min(max(DATA_H[223:192], -2^{7}), 2^{7}-1)
    qu[127:120] = min(max(DATA_H[255:224], -2^{7}), 2^{7}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

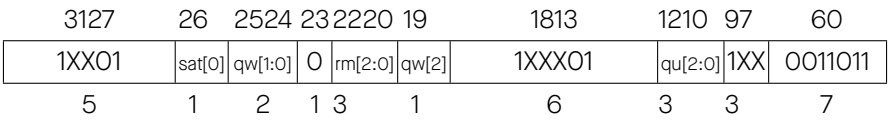


图 4.7-332. ESP.SRCMB.S8.Q.QACC

4.7.2.333 ESP.SRCMB.S8.QACC

语法

ESP.SRCMB.S8.QACC qu,rs1,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 16 个 32 位数据段，并分别执行算术右移操作。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 8 位有符号数，并将饱和后得到的 16 个 8 位数据写入寄存器 qu。

操作

```
DATA_L[ 31:  0] = QACC_L[ 31:  0] >> rs1[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> rs1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> rs1[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> rs1[4:0]
DATA_L[159:128] = QACC_L[159:128] >> rs1[4:0]
DATA_L[191:160] = QACC_L[191:160] >> rs1[4:0]
DATA_L[223:192] = QACC_L[223:192] >> rs1[4:0]
DATA_L[255:224] = QACC_L[255:224] >> rs1[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> rs1[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> rs1[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> rs1[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> rs1[4:0]
DATA_H[159:128] = QACC_H[159:128] >> rs1[4:0]
DATA_H[191:160] = QACC_H[191:160] >> rs1[4:0]
DATA_H[223:192] = QACC_H[223:192] >> rs1[4:0]
DATA_H[255:224] = QACC_H[255:224] >> rs1[4:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[  7:  0] = DATA_L[  7:  0]
  qu[ 15:  8] = DATA_L[ 39: 32]
  qu[ 23: 16] = DATA_L[ 71: 64]
  qu[ 31: 24] = DATA_L[103: 96]
  qu[ 39: 32] = DATA_L[135:128]
  qu[ 47: 40] = DATA_L[167:160]
  qu[ 55: 48] = DATA_L[199:192]
  qu[ 63: 56] = DATA_L[231:224]
  qu[ 71: 64] = DATA_H[  7:  0]
  qu[ 79: 72] = DATA_H[ 39: 32]
  qu[ 87: 80] = DATA_H[ 71: 64]
  qu[ 95: 88] = DATA_H[103: 96]
  qu[103: 96] = DATA_H[135:128]
  qu[111:104] = DATA_H[167:160]
  qu[119:112] = DATA_H[199:192]
  qu[127:120] = DATA_H[231:224]
else if sel2 == 1:
  qu[  7:  0] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
```

```
    qu[ 15:  8] = min(max(DATA_L[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 23: 16] = min(max(DATA_L[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 31: 24] = min(max(DATA_L[127: 96], -2^{7}), 2^{7}-1)
    qu[ 39: 32] = min(max(DATA_L[159:128], -2^{7}), 2^{7}-1)
    qu[ 47: 40] = min(max(DATA_L[191:160], -2^{7}), 2^{7}-1)
    qu[ 55: 48] = min(max(DATA_L[223:192], -2^{7}), 2^{7}-1)
    qu[ 63: 56] = min(max(DATA_L[255:224], -2^{7}), 2^{7}-1)
    qu[ 71: 64] = min(max(DATA_L[ 31:  0], -2^{7}), 2^{7}-1)
    qu[ 79: 72] = min(max(DATA_H[ 63: 32], -2^{7}), 2^{7}-1)
    qu[ 87: 80] = min(max(DATA_H[ 95: 64], -2^{7}), 2^{7}-1)
    qu[ 95: 88] = min(max(DATA_H[127: 96], -2^{7}), 2^{7}-1)
    qu[103: 96] = min(max(DATA_H[159:128], -2^{7}), 2^{7}-1)
    qu[111:104] = min(max(DATA_H[191:160], -2^{7}), 2^{7}-1)
    qu[119:112] = min(max(DATA_H[223:192], -2^{7}), 2^{7}-1)
    qu[127:120] = min(max(DATA_H[255:224], -2^{7}), 2^{7}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

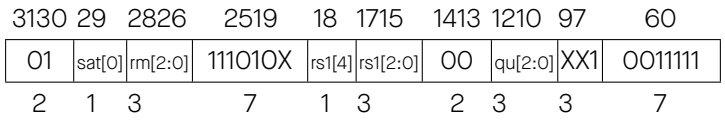


图 4.7-333. ESP.SRCMB.S8.QACC

4.7.2.334 ESP.SRCMB.U16.Q.QACC

语法

ESP.SRCMB.U16.Q.QACC qu,qw,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 8 个 64 位的数据段，并分别进行算术右移操作，移位量来自 qw。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 16 位无符号数，并将饱和后得到的 8 个 16 位数据写入寄存器 qu。

操作

```
shf_value0[5:0] = (qw[ 15: 0] >= 0x3f) ? 0x3f : qw[ 15: 0]
shf_value1[5:0] = (qw[ 31: 16] >= 0x3f) ? 0x3f : qw[ 31: 16]
shf_value2[5:0] = (qw[ 47: 32] >= 0x3f) ? 0x3f : qw[ 47: 32]
shf_value3[5:0] = (qw[ 63: 48] >= 0x3f) ? 0x3f : qw[ 63: 48]
shf_value4[5:0] = (qw[ 79: 64] >= 0x3f) ? 0x3f : qw[ 79: 64]
shf_value5[5:0] = (qw[ 95: 80] >= 0x3f) ? 0x3f : qw[ 95: 80]
shf_value6[5:0] = (qw[111: 96] >= 0x3f) ? 0x3f : qw[111: 96]
shf_value7[5:0] = (qw[127:112] >= 0x3f) ? 0x3f : qw[127:112]

DATA_L[ 63:  0] = QACC_L[ 63:  0] >> shf_value0[5:0]
DATA_L[127: 64] = QACC_L[127: 64] >> shf_value1[5:0]
DATA_L[191:128] = QACC_L[191:128] >> shf_value2[5:0]
DATA_L[255:192] = QACC_L[255:192] >> shf_value3[5:0]
DATA_H[ 63:  0] = QACC_H[ 63:  0] >> shf_value4[5:0]
DATA_H[127: 64] = QACC_H[127: 64] >> shf_value5[5:0]
DATA_H[191:128] = QACC_H[191:128] >> shf_value6[5:0]
DATA_H[255:192] = QACC_H[255:192] >> shf_value7[5:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[15: 0] = DATA_L[15:0]
  qu[31:16] = DATA_L[79:64]
  qu[47:32] = DATA_L[143:128]
  qu[63:48] = DATA_L[207:192]
  qu[79:64] = DATA_H[15:0]
  qu[95:80] = DATA_H[79:64]
  qu[111:96] = DATA_H[143:128]
  qu[127:112] = DATA_H[207:192]
else if sel2 == 1:
  qu[ 15:  0] = min(DATA_L[ 63:  0], 2^{16}-1)
  qu[ 31: 16] = min(DATA_L[127: 64], 2^{16}-1)
  qu[ 47: 32] = min(DATA_L[191:128], 2^{16}-1)
  qu[ 63: 48] = min(DATA_L[255:192], 2^{16}-1)
  qu[ 79: 64] = min(DATA_H[ 63:  0], 2^{16}-1)
  qu[ 95: 80] = min(DATA_H[127: 64], 2^{16}-1)
  qu[111: 96] = min(DATA_H[191:128], 2^{16}-1)
  qu[127:112] = min(DATA_H[255:192], 2^{16}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

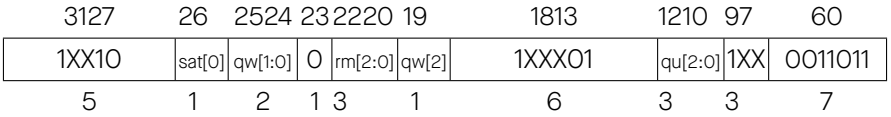


图 4.7-334. ESP.SRCMB.U16.Q.QACC

4.7.2.335 ESP.SRCMB.U16.QACC

语法

ESP.SRCMB.U16.QACC qu,rs1,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 8 个 64 位的数据段，并分别进行算术右移操作。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 16 位无符号数，并将饱和后得到的 8 个 16 位数据段写入 qu 寄存器。

操作

```
temp0[63:0] = QACC_L[ 63:  0]
...
temp3[63:0] = QACC_L[255:192]
temp4[63:0] = QACC_H[ 63:  0]
...
temp7[63:0] = QACC_H[255:192]

temp_shf0[63:0] = temp0[63:0] >> rs1[5:0]
temp_shf1[63:0] = temp1[63:0] >> rs1[5:0]
...
temp_shf7[63:0] = temp7[63:0] >> rs1[5:0]

QACC_L[ 63:  0] = temp_shf0[63:0]
...
QACC_L[255:192] = temp_shf3[63:0]
QACC_H[ 63:  0] = temp_shf4[63:0]
...
QACC_H[255:192] = temp_shf7[63:0]

if sel2 == 0:
  qu[ 15:  0] = temp0[15:0]
  qu[ 31: 16] = temp1[15:0]
  qu[ 47: 32] = temp2[15:0]
  qu[ 63: 48] = temp3[15:0]
  qu[ 79: 64] = temp4[15:0]
  qu[ 95: 80] = temp5[15:0]
  qu[111: 96] = temp6[15:0]
  qu[127:112] = temp7[15:0]
else if sel2==1:
  qu[ 15:  0] = min(temp_shf0[63:0], 2^{16}-1)
  ...
  qu[ 63: 48] = min(temp_shf3[63:0], 2^{16}-1)
  qu[ 79: 64] = min(temp_shf4[63:0], 2^{16}-1)
  ...
  qu[127:112] = min(temp_shf7[63:0], 2^{16}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2

操作数	读流水级数	写流水级数
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

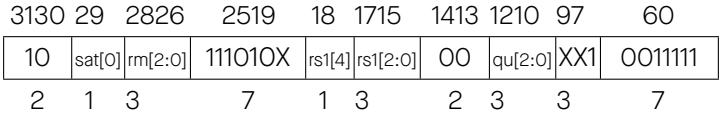


图 4.7-335. ESP.SRCMB.U16.QACC

4.7.2.336 ESP.SRCMB.U8.Q.QACC

语法

ESP.SRCMB.U8.Q.QACC qu,qw,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 16 个 32 位的数据段，并分别进行算术右移操作，移位量来自 qw。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 8 位无符号数，并将饱和后得到的 16 个 8 位数据写入寄存器 qu。

操作

```
shf_value0[5:0] = (qw[ 7: 0] >= 0x1f) ? 0x1f : qw[ 7: 0]
shf_value1[5:0] = (qw[ 15: 8] >= 0x1f) ? 0x1f : qw[ 15: 8]
shf_value2[5:0] = (qw[ 23: 16] >= 0x1f) ? 0x1f : qw[ 23: 16]
shf_value3[5:0] = (qw[ 31: 24] >= 0x1f) ? 0x1f : qw[ 31: 24]
shf_value4[5:0] = (qw[ 39: 32] >= 0x1f) ? 0x1f : qw[ 39: 32]
shf_value5[5:0] = (qw[ 47: 40] >= 0x1f) ? 0x1f : qw[ 47: 40]
shf_value6[5:0] = (qw[ 55: 48] >= 0x1f) ? 0x1f : qw[ 55: 48]
shf_value7[5:0] = (qw[ 63: 56] >= 0x1f) ? 0x1f : qw[ 63: 56]
shf_value8[5:0] = (qw[ 71: 64] >= 0x1f) ? 0x1f : qw[ 71: 64]
shf_value9[5:0] = (qw[ 79: 72] >= 0x1f) ? 0x1f : qw[ 79: 72]
shf_value10[5:0] = (qw[ 87: 80] >= 0x1f) ? 0x1f : qw[ 87: 80]
shf_value11[5:0] = (qw[ 95: 88] >= 0x1f) ? 0x1f : qw[ 95: 88]
shf_value12[5:0] = (qw[103: 96] >= 0x1f) ? 0x1f : qw[103: 96]
shf_value13[5:0] = (qw[111:104] >= 0x1f) ? 0x1f : qw[111:104]
shf_value14[5:0] = (qw[119:112] >= 0x1f) ? 0x1f : qw[119:112]
shf_value15[5:0] = (qw[127:120] >= 0x1f) ? 0x1f : qw[127:120]

DATA_L[ 31:  0] = QACC_L[ 31:  0] >> shf_value0[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> shf_value1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> shf_value2[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> shf_value3[4:0]
DATA_L[159:128] = QACC_L[159:128] >> shf_value4[4:0]
DATA_L[191:160] = QACC_L[191:160] >> shf_value5[4:0]
DATA_L[223:192] = QACC_L[223:192] >> shf_value6[4:0]
DATA_L[255:224] = QACC_L[255:224] >> shf_value7[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> shf_value8[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> shf_value9[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> shf_value10[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> shf_value11[4:0]
DATA_H[159:128] = QACC_H[159:128] >> shf_value12[4:0]
DATA_H[191:160] = QACC_H[191:160] >> shf_value13[4:0]
DATA_H[223:192] = QACC_H[223:192] >> shf_value14[4:0]
DATA_H[255:224] = QACC_H[255:224] >> shf_value15[4:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[  7:  0] = DATA_L[  7:  0]
  qu[ 15:  8] = DATA_L[ 39: 32]
```

```
    qu[ 23: 16] = DATA_L[ 71: 64]
    qu[ 31: 24] = DATA_L[103: 96]
    qu[ 39: 32] = DATA_L[135:128]
    qu[ 47: 40] = DATA_L[167:160]
    qu[ 55: 48] = DATA_L[199:192]
    qu[ 63: 56] = DATA_L[231:224]
    qu[ 71: 64] = DATA_H[  7:  0]
    qu[ 79: 72] = DATA_H[ 39: 32]
    qu[ 87: 80] = DATA_H[ 71: 64]
    qu[ 95: 88] = DATA_H[103: 96]
    qu[103: 96] = DATA_H[135:128]
    qu[111:104] = DATA_H[167:160]
    qu[119:112] = DATA_H[199:192]
    qu[127:120] = DATA_H[231:224]
  else if sel2 == 1:
    qu[  7:  0] = min(DATA_L[ 31:  0], 2^{8}-1)
    qu[ 15:  8] = min(DATA_L[ 63: 32], 2^{8}-1)
    qu[ 23: 16] = min(DATA_L[ 95: 64], 2^{8}-1)
    qu[ 31: 24] = min(DATA_L[127: 96], 2^{8}-1)
    qu[ 39: 32] = min(DATA_L[159:128], 2^{8}-1)
    qu[ 47: 40] = min(DATA_L[191:160], 2^{8}-1)
    qu[ 55: 48] = min(DATA_L[223:192], 2^{8}-1)
    qu[ 63: 56] = min(DATA_L[255:224], 2^{8}-1)
    qu[ 71: 64] = min(DATA_L[ 31:  0], 2^{8}-1)
    qu[ 79: 72] = min(DATA_H[ 63: 32], 2^{8}-1)
    qu[ 87: 80] = min(DATA_H[ 95: 64], 2^{8}-1)
    qu[ 95: 88] = min(DATA_H[127: 96], 2^{8}-1)
    qu[103: 96] = min(DATA_H[159:128], 2^{8}-1)
    qu[111:104] = min(DATA_H[191:160], 2^{8}-1)
    qu[119:112] = min(DATA_H[223:192], 2^{8}-1)
    qu[127:120] = min(DATA_H[255:224], 2^{8}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

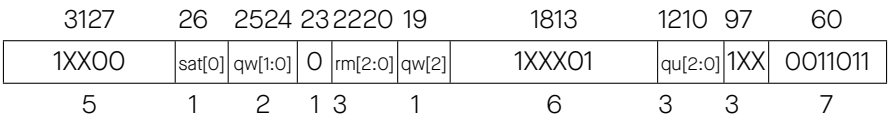


图 4.7-336. ESP.SRCMB.U8.Q.QACC

4.7.2.337 ESP.SRCMB.U8.QACC

语法

ESP.SRCMB.U8.QACC qu,rs1,sat,rm

描述

此指令从特殊寄存器 QACC_H 和 QACC_L 中提取 16 个 32 位数据段，并分别执行算术右移操作。在将移位结果写回 QACC_H 和 QACC_L 时，该指令将结果饱和到 8 位无符号数，并将饱和后得到的 16 个 8 位数据写入寄存器 qu。

操作

```
DATA_L[ 31:  0] = QACC_L[ 31:  0] >> rs1[4:0]
DATA_L[ 63: 32] = QACC_L[ 63: 32] >> rs1[4:0]
DATA_L[ 95: 64] = QACC_L[ 95: 64] >> rs1[4:0]
DATA_L[127: 96] = QACC_L[127: 96] >> rs1[4:0]
DATA_L[159:128] = QACC_L[159:128] >> rs1[4:0]
DATA_L[191:160] = QACC_L[191:160] >> rs1[4:0]
DATA_L[223:192] = QACC_L[223:192] >> rs1[4:0]
DATA_L[255:224] = QACC_L[255:224] >> rs1[4:0]
DATA_H[ 31:  0] = QACC_H[ 31:  0] >> rs1[4:0]
DATA_H[ 63: 32] = QACC_H[ 63: 32] >> rs1[4:0]
DATA_H[ 95: 64] = QACC_H[ 95: 64] >> rs1[4:0]
DATA_H[127: 96] = QACC_H[127: 96] >> rs1[4:0]
DATA_H[159:128] = QACC_H[159:128] >> rs1[4:0]
DATA_H[191:160] = QACC_H[191:160] >> rs1[4:0]
DATA_H[223:192] = QACC_H[223:192] >> rs1[4:0]
DATA_H[255:224] = QACC_H[255:224] >> rs1[4:0]

QACC_L[255:0] = DATA_L[255:0]
QACC_H[255:0] = DATA_H[255:0]

if sel2 == 0:
  qu[  7:  0] = DATA_L[  7:  0]
  qu[ 15:  8] = DATA_L[ 39: 32]
  qu[ 23: 16] = DATA_L[ 71: 64]
  qu[ 31: 24] = DATA_L[103: 96]
  qu[ 39: 32] = DATA_L[135:128]
  qu[ 47: 40] = DATA_L[167:160]
  qu[ 55: 48] = DATA_L[199:192]
  qu[ 63: 56] = DATA_L[231:224]
  qu[ 71: 64] = DATA_H[  7:  0]
  qu[ 79: 72] = DATA_H[ 39: 32]
  qu[ 87: 80] = DATA_H[ 71: 64]
  qu[ 95: 88] = DATA_H[103: 96]
  qu[103: 96] = DATA_H[135:128]
  qu[111:104] = DATA_H[167:160]
  qu[119:112] = DATA_H[199:192]
  qu[127:120] = DATA_H[231:224]
else if sel2 == 1:
  qu[  7:  0] = min(DATA_L[ 31:  0], 2^{8}-1)
```

```
    qu[ 15:  8] = min(DATA_L[ 63: 32], 2^{8}-1)
    qu[ 23: 16] = min(DATA_L[ 95: 64], 2^{8}-1)
    qu[ 31: 24] = min(DATA_L[127: 96], 2^{8}-1)
    qu[ 39: 32] = min(DATA_L[159:128], 2^{8}-1)
    qu[ 47: 40] = min(DATA_L[191:160], 2^{8}-1)
    qu[ 55: 48] = min(DATA_L[223:192], 2^{8}-1)
    qu[ 63: 56] = min(DATA_L[255:224], 2^{8}-1)
    qu[ 71: 64] = min(DATA_L[ 31:  0], 2^{8}-1)
    qu[ 79: 72] = min(DATA_H[ 63: 32], 2^{8}-1)
    qu[ 87: 80] = min(DATA_H[ 95: 64], 2^{8}-1)
    qu[ 95: 88] = min(DATA_H[127: 96], 2^{8}-1)
    qu[103: 96] = min(DATA_H[159:128], 2^{8}-1)
    qu[111:104] = min(DATA_H[191:160], 2^{8}-1)
    qu[119:112] = min(DATA_H[223:192], 2^{8}-1)
    qu[127:120] = min(DATA_H[255:224], 2^{8}-1)
```

调度

操作数	读流水级数	写流水级数
qu	-	2
rs1	1	-
QACC_H	1	2
QACC_L	1	2

格式

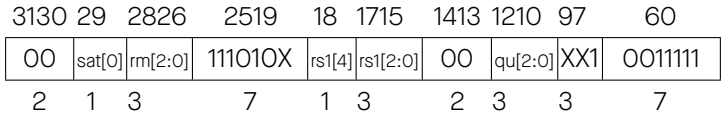


图 4.7-337. ESP.SRCMB.U8.QACC

4.7.2.338 ESP.SRCQ.128.ST.INCP

语法

ESP.SRCQ.128.ST.INCP qw,qy,rs1

描述

此指令对寄存器 qw 和 qy 的 32 字节串联进行算术右移。然后，它将移位结果的低 128 位写入内存。访问后，寄存器 rs1 中的值增加 16。

操作

```
{qy[127:  0], qw[127:  0]} >> {SAR_BYTE[3:0] << 3}  => store128(rs1[31:0])
rs1[31:0] = rs1[31:0] + 16
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qu	1	1
qv	1	2
rs1	1	1
SAR_BYTES	1	-

格式

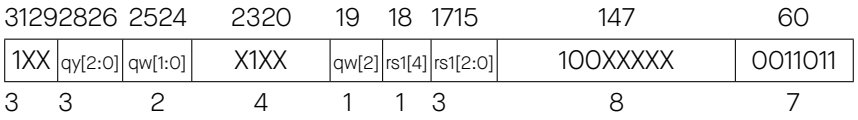


图 4.7-338. ESP.SRCQ.128.ST.INCP

4.7.2.339 ESP.SRCXXP.2Q

语法

ESP.SRCXXP.2Q qy,qw,rs1,rs2

描述

此指令对寄存器qw和qy的 32 字节串联进行逻辑右移，并用 0 填充高位。移位结果的上 128 位写入寄存器qy，下 128 位写入qw。右移的数量是寄存器rs1的低 4 位值加 1 后的总和的 8 倍。在上述操作之后，rs1中的值增加了rs2中的值。

操作

```
{qy[127:  0], qw[127:  0]} = {qy[127:  0], qw[127:  0]} >> ((rs1[3:0]+1)*8)
qy[127:127-8*rs1[3:  0]] = 0
rs1[31:0] = rs1[31:0] + rs2[31:0]
```

调度

操作数	读流水级数	写流水级数
qy	1	1
qw	1	1
rs1	1	1
rs2	1	-

格式

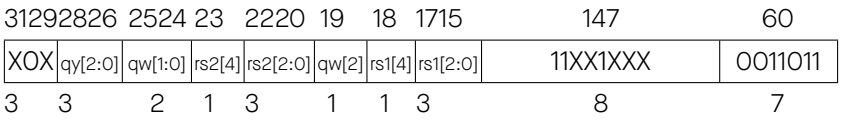


图 4.7-339. ESP.SRCXXP.2Q

4.7.2.340 ESP.SRS.S.XACC

语法

ESP.SRS.S.XACC rd,rs1,sat,rm

描述

此指令对特殊寄存器 XACC 执行算术右移。在将移位结果写回 XACC 时，该指令将移位结果饱和为 32 位有符号数，并将饱和结果写入寄存器 rd。

操作

```
temp_shf[39:0] = XACC[39:0] >> rs1[5:0]
XACC = temp_shf[39:0]
rd = min(max(temp_shf[39:0], -2^{31}), 2^{31}-1)
```

调度

操作数	读流水级数	写流水级数
rd	-	2
rs1	1	-
XACC	1	2

格式

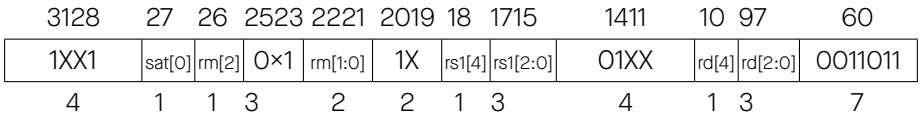


图 4.7-340. ESP.SRS.S.XACC

4.7.2.341 ESP.SRS.U.XACC

语法

ESP.SRS.U.XACC rd,rs1,sat,rm

描述

此指令对特殊寄存器 XACC 执行算术右移。在将移位结果写回 XACC 时，该指令将移位结果饱和为 32 位无符号数，并将饱和结果写入寄存器 rd。

操作

```
temp_shf[39:0] = XACC[39:0] >> rs1[5:0]
XACC = temp_shf[39:0]
rd = min(temp_shf[39:0], 2^{32}-1)
```

调度

操作数	读流水级数	写流水级数
rd	-	2
rs1	1	-
XACC	1	2

格式

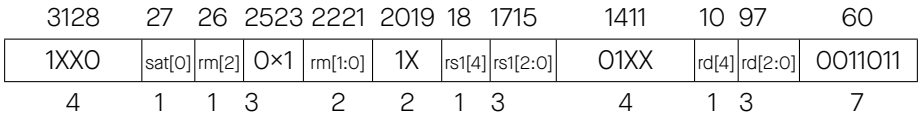


图 4.7-341. ESP.SRS.U.XACC

4.7.2.342 ESP.VSLD.16

语法

ESP.VSLD.16 qu,qy,qw,sat,rm

描述

此指令执行 16 位向量架构移位，移位量来自qw的段。如果移位量是负数，则执行右移；否则，执行左移。

操作

```
qu[ 15:  0] = (qw[ 4: 0] < 0) ? (qy[ 15: 0] » (-qw[ 4: 0]) ) : (qy[ 15: 0] « qw[ 4: 0])
qu[ 31: 16] = (qw[ 20: 16] < 0) ? (qy[ 31: 16] » (-qw[ 20: 16]) ) : (qy[ 31: 16] « qw[ 20: 16])
qu[ 47: 32] = (qw[ 36: 32] < 0) ? (qy[ 47: 32] » (-qw[ 36: 32]) ) : (qy[ 47: 32] « qw[ 36: 32])
qu[ 63: 48] = (qw[ 52: 48] < 0) ? (qy[ 63: 48] » (-qw[ 52: 48]) ) : (qy[ 63: 48] « qw[ 52: 48])
qu[ 79: 64] = (qw[ 68: 64] < 0) ? (qy[ 79: 64] » (-qw[ 68: 64]) ) : (qy[ 79: 64] « qw[ 68: 64])
qu[ 95: 80] = (qw[ 84: 80] < 0) ? (qy[ 95: 80] » (-qw[ 84: 80]) ) : (qy[ 95: 80] « qw[ 84: 80])
qu[111: 96] = (qw[100: 96] < 0) ? (qy[111: 96] » (-qw[100: 96]) ) : (qy[111: 96] « qw[100: 96])
qu[127:112] = (qw[116:112] < 0) ? (qy[127:112] » (-qw[116:112]) ) : (qy[127:112] « qw[116:112])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
qw	1	-

格式

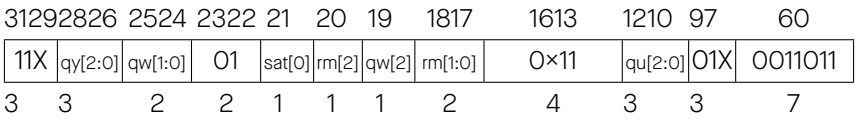


图 4.7-342. ESP.VSLD.16

4.7.2.343 ESP.VSLD.32

语法

ESP.VSLD.32 qu,qy,qw,sat,rm

描述

此指令执行 32 位向量架构移位，移位量来自qw的段。如果移位量是负数，则执行右移；否则，执行左移。

操作

```
qu[ 31:  0] = (qw[ 5: 0] < 0) ? (qy[ 31: 0] » (-qw[ 5: 0]) ) : (qy[ 31: 0] « qw[ 5: 0])
qu[ 63: 32] = (qw[ 37: 32] < 0) ? (qy[ 63: 32] » (-qw[ 37: 32]) ) : (qy[ 63: 32] « qw[ 37: 32])
qu[ 95: 64] = (qw[ 69: 64] < 0) ? (qy[ 95: 64] » (-qw[ 69: 64]) ) : (qy[ 95: 64] « qw[ 69: 64])
qu[127: 96] = (qw[101: 96] < 0) ? (qy[127: 96] » (-qw[101: 96]) ) : (qy[127: 96] « qw[101: 96])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
qw	1	-

格式

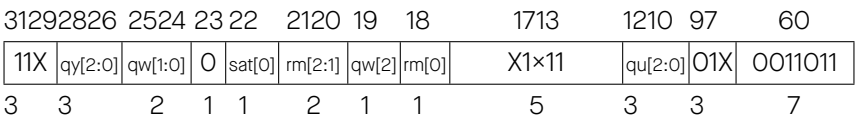


图 4.7-343. ESP.VSLD.32

4.7.2.344 ESP.VSLD.8

语法

ESP.VSLD.8 qu,qy,qw,sat,rm

描述

此指令执行 8 位向量架构移位，移位量来自qw的段。如果移位量是负数，则执行右移；否则，执行左移。

操作

```
qu[  7:  0] = (qw[ 3: 0] < 0) ? (qy[ 7: 0] » (-qw[ 3: 0]) ) : (qy[ 7: 0] « qw[ 3: 0])
qu[ 15:  8] = (qw[ 11: 8] < 0) ? (qy[ 15: 8] » (-qw[ 11: 8]) ) : (qy[ 15: 8] « qw[ 11: 8])
qu[ 23: 16] = (qw[ 19: 16] < 0) ? (qy[ 23: 16] » (-qw[ 19: 16]) ) : (qy[ 23: 16] « qw[ 19: 16])
qu[ 31: 24] = (qw[ 27: 24] < 0) ? (qy[ 31: 24] » (-qw[ 27: 24]) ) : (qy[ 31: 24] « qw[ 27: 24])
qu[ 39: 32] = (qw[ 35: 32] < 0) ? (qy[ 39: 32] » (-qw[ 35: 32]) ) : (qy[ 39: 32] « qw[ 35: 32])
qu[ 47: 40] = (qw[ 43: 40] < 0) ? (qy[ 47: 40] » (-qw[ 43: 40]) ) : (qy[ 47: 40] « qw[ 43: 40])
qu[ 55: 48] = (qw[ 51: 48] < 0) ? (qy[ 55: 48] » (-qw[ 51: 48]) ) : (qy[ 55: 48] « qw[ 51: 48])
qu[ 63: 56] = (qw[ 59: 56] < 0) ? (qy[ 63: 56] » (-qw[ 59: 56]) ) : (qy[ 63: 56] « qw[ 59: 56])
qu[ 71: 64] = (qw[ 67: 64] < 0) ? (qy[ 71: 64] » (-qw[ 67: 64]) ) : (qy[ 71: 64] « qw[ 67: 64])
qu[ 79: 72] = (qw[ 75: 72] < 0) ? (qy[ 79: 72] » (-qw[ 75: 72]) ) : (qy[ 79: 72] « qw[ 75: 72])
qu[ 87: 80] = (qw[ 83: 80] < 0) ? (qy[ 87: 80] » (-qw[ 83: 80]) ) : (qy[ 87: 80] « qw[ 83: 80])
qu[ 95: 88] = (qw[ 91: 88] < 0) ? (qy[ 95: 88] » (-qw[ 91: 88]) ) : (qy[ 95: 88] « qw[ 91: 88])
qu[103: 96] = (qw[ 99: 96] < 0) ? (qy[103: 96] » (-qw[ 99: 96]) ) : (qy[103: 96] « qw[ 99: 96])
qu[111:104] = (qw[107:104] < 0) ? (qy[111:104] » (-qw[107:104]) ) : (qy[111:104] « qw[107:104])
qu[119:112] = (qw[115:112] < 0) ? (qy[119:112] » (-qw[115:112]) ) : (qy[119:112] « qw[115:112])
qu[127:120] = (qw[123:120] < 0) ? (qy[127:120] » (-qw[123:120]) ) : (qy[127:120] « qw[123:120])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
qw	1	-

格式

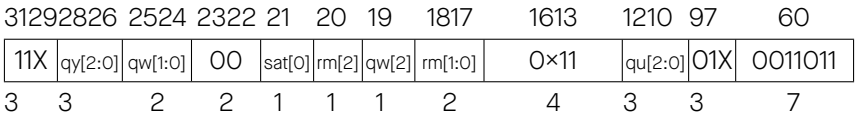


图 4.7-344. ESP.VSLD.8

4.7.2.345 ESP.VSR.S32

语法

ESP.VSR.S32 qu,qy,rm

描述

此指令对寄存器 qy 中的四个 32 位数据段分别执行算术右移。移位量是 6 位特殊寄存器 SAR 中的值。在移位过程中，高位用符号位填充。移位结果的低 32 位存储在寄存器 qu 中对应的数据段中。

操作

```
qu[ 31:  0] = (qy[ 31:  0] >> SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] >> SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] >> SAR[5:0])
qu[127: 96] = (qy[127: 96] >> SAR[5:0])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
SAR	1	-

格式

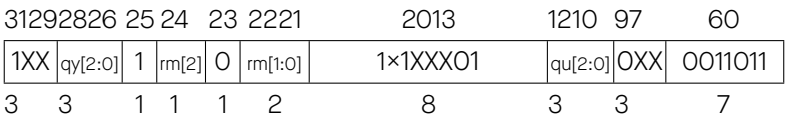


图 4.7-345. ESP.VSR.S32

4.7.2.346 ESP.VSR.U32

语法

ESP.VSR.U32 qu,qy,rm

描述

此指令对寄存器qy中的四个 32 位无符号数据段分别执行算术右移。移位量是 6 位特殊寄存器SAR中的值。在移位过程中，高位用符号位填充。移位结果的低 32 位存储在寄存器qu中相应的数据段中。

操作

```
qu[ 31:  0] = (qy[ 31:  0] >> SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] >> SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] >> SAR[5:0])
qu[127: 96] = (qy[127: 96] >> SAR[5:0])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
SAR	1	-

格式

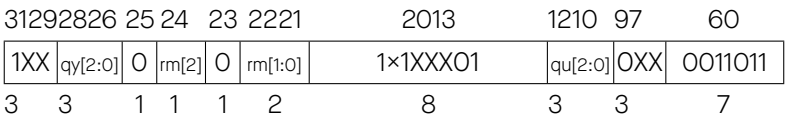


图 4.7-346. ESP.VSR.U32

4.7.2.347 ESP.VSRD.16

语法

ESP.VSRD.16 qu,qy,qw,sat,rm

描述

此指令执行 16 位向量架构的移位，移位量来自qw的段。如果移位量是负数，则执行左移；否则，执行右移。

操作

```
qu[ 15:  0] = (qw[ 4: 0] < 0) ? (qy[ 15: 0] « (-qw[ 4: 0]) ) : (qy[ 15: 0] » qw[ 4: 0])
qu[ 31: 16] = (qw[ 20: 16] < 0) ? (qy[ 31: 16] « (-qw[ 20: 16]) ) : (qy[ 31: 16] » qw[ 20: 16])
qu[ 47: 32] = (qw[ 36: 32] < 0) ? (qy[ 47: 32] « (-qw[ 36: 32]) ) : (qy[ 47: 32] » qw[ 36: 32])
qu[ 63: 48] = (qw[ 52: 48] < 0) ? (qy[ 63: 48] « (-qw[ 52: 48]) ) : (qy[ 63: 48] » qw[ 52: 48])
qu[ 79: 64] = (qw[ 68: 64] < 0) ? (qy[ 79: 64] « (-qw[ 68: 64]) ) : (qy[ 79: 64] » qw[ 68: 64])
qu[ 95: 80] = (qw[ 84: 80] < 0) ? (qy[ 95: 80] « (-qw[ 84: 80]) ) : (qy[ 95: 80] » qw[ 84: 80])
qu[111: 96] = (qw[100: 96] < 0) ? (qy[111: 96] « (-qw[100: 96]) ) : (qy[111: 96] » qw[100: 96])
qu[127:112] = (qw[116:112] < 0) ? (qy[127:112] « (-qw[116:112]) ) : (qy[127:112] » qw[116:112])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
qw	1	-

格式

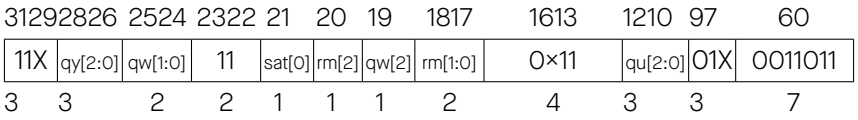


图 4.7-347. ESP.VSRD.16

4.7.2.348 ESP.VSRD.32

语法

ESP.VSRD.32 qu,qy,qw,sat,rm

描述

此指令执行 32 位向量架构移位，移位量来自qw的段。如果移位量是负数，则执行左移；否则，执行右移。

操作

```
qu[ 31:  0] = (qw[ 5: 0] < 0) ? (qy[ 31: 0] « (-qw[ 5: 0]) ) : (qy[ 31: 0] » qw[ 5: 0])
qu[ 63: 32] = (qw[ 37: 32] < 0) ? (qy[ 63: 32] « (-qw[ 37: 32]) ) : (qy[ 63: 32] » qw[ 37: 32])
qu[ 95: 64] = (qw[ 69: 64] < 0) ? (qy[ 95: 64] « (-qw[ 69: 64]) ) : (qy[ 95: 64] » qw[ 69: 64])
qu[127: 96] = (qw[101: 96] < 0) ? (qy[127: 96] « (-qw[101: 96]) ) : (qy[127: 96] » qw[101: 96])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
qw	1	-

格式

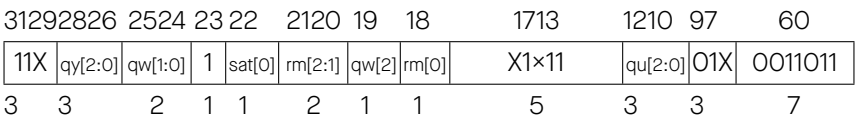


图 4.7-348. ESP.VSRD.32

4.7.2.349 ESP.VSRD.8

语法

ESP.VSRD.8 qu,qy,qw,sat,rm

描述

此指令执行一个 8 位向量架构的移位，移位量来自qw的段。如果移位量是负数，则执行左移；否则，执行右移。

操作

```
qu[  7:  0] = (qw[ 3: 0] < 0) ? (qy[ 7: 0] « (-qw[ 3: 0]) ) : (qy[ 7: 0] » qw[ 3: 0])
qu[ 15:  8] = (qw[ 11: 8] < 0) ? (qy[ 15: 8] « (-qw[ 11: 8]) ) : (qy[ 15: 8] » qw[ 11: 8])
qu[ 23: 16] = (qw[ 19: 16] < 0) ? (qy[ 23: 16] « (-qw[ 19: 16]) ) : (qy[ 23: 16] » qw[ 19: 16])
qu[ 31: 24] = (qw[ 27: 24] < 0) ? (qy[ 31: 24] « (-qw[ 27: 24]) ) : (qy[ 31: 24] » qw[ 27: 24])
qu[ 39: 32] = (qw[ 35: 32] < 0) ? (qy[ 39: 32] « (-qw[ 35: 32]) ) : (qy[ 39: 32] » qw[ 35: 32])
qu[ 47: 40] = (qw[ 43: 40] < 0) ? (qy[ 47: 40] « (-qw[ 43: 40]) ) : (qy[ 47: 40] » qw[ 43: 40])
qu[ 55: 48] = (qw[ 51: 48] < 0) ? (qy[ 55: 48] « (-qw[ 51: 48]) ) : (qy[ 55: 48] » qw[ 51: 48])
qu[ 63: 56] = (qw[ 59: 56] < 0) ? (qy[ 63: 56] « (-qw[ 59: 56]) ) : (qy[ 63: 56] » qw[ 59: 56])
qu[ 71: 64] = (qw[ 67: 64] < 0) ? (qy[ 71: 64] « (-qw[ 67: 64]) ) : (qy[ 71: 64] » qw[ 67: 64])
qu[ 79: 72] = (qw[ 75: 72] < 0) ? (qy[ 79: 72] « (-qw[ 75: 72]) ) : (qy[ 79: 72] » qw[ 75: 72])
qu[ 87: 80] = (qw[ 83: 80] < 0) ? (qy[ 87: 80] « (-qw[ 83: 80]) ) : (qy[ 87: 80] » qw[ 83: 80])
qu[ 95: 88] = (qw[ 91: 88] < 0) ? (qy[ 95: 88] « (-qw[ 91: 88]) ) : (qy[ 95: 88] » qw[ 91: 88])
qu[103: 96] = (qw[ 99: 96] < 0) ? (qy[103: 96] « (-qw[ 99: 96]) ) : (qy[103: 96] » qw[ 99: 96])
qu[111:104] = (qw[107:104] < 0) ? (qy[111:104] « (-qw[107:104]) ) : (qy[111:104] » qw[107:104])
qu[119:112] = (qw[115:112] < 0) ? (qy[119:112] « (-qw[115:112]) ) : (qy[119:112] » qw[115:112])
qu[127:120] = (qw[123:120] < 0) ? (qy[127:120] « (-qw[123:120]) ) : (qy[127:120] » qw[123:120])
```

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
qw	1	-

格式

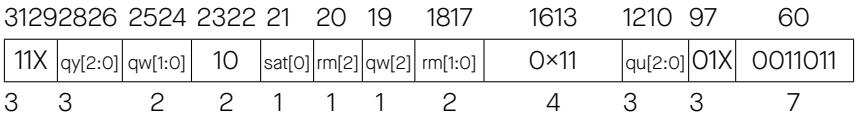


图 4.7-349. ESP.VSRD.8

4.7.2.350 ESP.ST.S.XACC.IP

语法

ESP.ST.S.XACC.IP rs1,imm8

描述

此指令将特殊寄存器 XACC 符号扩展到 64 位，并将结果存储在内存中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位符号扩展常数，该常数左移 3 位。

imm 从 -1024 开始，到 1016 结束，步长为 8。

操作

```
MEM(rs1[31:0]) = {24{XACC[39]}, XACC[39:0]}
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
XACC	2	-
rs1	1	1

格式

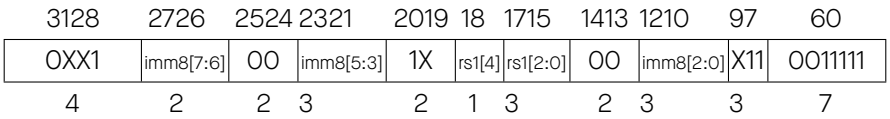


图 4.7-350. ESP.ST.S.XACC.IP

4.7.2.351 ESP.ST.U.XACC.IP

语法

ESP.ST.U.XACC.IP rs1,imm8

描述

此指令将特殊寄存器 XACC 零扩展到 64 位，并将结果存储在内存中。访问完成后，寄存器 rs1 中的值会增加指令代码段中的一个 8 位有符号扩展常数，该常数左移 3 位。

imm 从 -1024 开始，到 1016 结束，步长为 8。

操作

```
MEM(rs1[31:0]) = {24{0}, XACC[39:0]}
rs1[31:0] = rs1[31:0] + imm
```

调度

操作数	读流水级数	写流水级数
XACC	2	-
rs1	1	1

格式

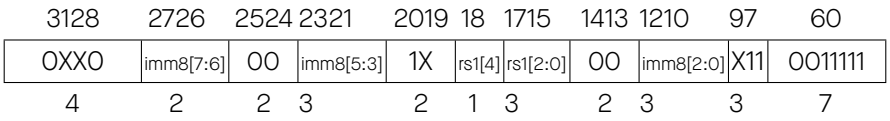


图 4.7-351. ESP.ST.U.XACC.IP

4.7.2.352 ESP.FFT.BITREV

语法

ESP.FFT.BITREV qv,rs1

描述

此指令根据特殊寄存器 FFT_BIT_WIDTH 的值交换不同位宽的数据的位顺序。然后，它比较交换前后的数据，取较大的值，用 0 填充高位，直到达到 16 位，并将结果写入寄存器 qa 的相应数据段。在下文中，Switchx 是一个表示 x 位的位反转的函数。Switch5(0b10100) = 0b00101。这里，0b10100 表示 5 位二进制数据。

操作

```
  temp0[15:0] = rs1[15:0]
  temp1[15:0] = rs1[15:0] + 1
  temp2[15:0] = rs1[15:0] + 2
  temp3[15:0] = rs1[15:0] + 3
  temp4[15:0] = rs1[15:0] + 4
  temp5[15:0] = rs1[15:0] + 5
  temp6[15:0] = rs1[15:0] + 6
  temp7[15:0] = rs1[15:0] + 7
if FFT_BIT_WIDTH==3:
  Switch3(X[2:0]) = X[0:2]
  qv[ 15:  0] = {13'h0, max(tmp0[2:0], Switch3(tmp0[2:0]))}
  qv[ 31: 16] = {13'h0, max(tmp1[2:0], Switch3(tmp1[2:0]))}
  qv[ 47: 32] = {13'h0, max(tmp2[2:0], Switch3(tmp2[2:0]))}
  qv[ 63: 48] = {13'h0, max(tmp3[2:0], Switch3(tmp3[2:0]))}
  qv[ 79: 64] = {13'h0, max(tmp4[2:0], Switch3(tmp4[2:0]))}
  qv[ 95: 80] = {13'h0, max(tmp5[2:0], Switch3(tmp5[2:0]))}
  qv[127: 96] = 0
if FFT_BIT_WIDTH==4:
  Switch4(X[3:0]) = X[0:3]
  qv[ 15:  0] = {12'h0, max(tmp0[3:0], Switch4(tmp0[3:0]))}
  qv[ 31: 16] = {12'h0, max(tmp1[3:0], Switch4(tmp1[3:0]))}
  qv[ 47: 32] = {12'h0, max(tmp2[3:0], Switch4(tmp2[3:0]))}
  qv[ 63: 48] = {12'h0, max(tmp3[3:0], Switch4(tmp3[3:0]))}
  qv[ 79: 64] = {12'h0, max(tmp4[3:0], Switch4(tmp4[3:0]))}
  qv[ 95: 80] = {12'h0, max(tmp5[3:0], Switch4(tmp5[3:0]))}
  qv[111: 96] = {12'h0, max(tmp6[3:0], Switch4(tmp6[3:0]))}
  qv[127:112] = {12'h0, max(tmp7[3:0], Switch4(tmp7[3:0]))}
  ...
if FFT_BIT_WIDTH==10:
  Switch10(X[9:0]) = X[0:9]
  qv[ 15:  0] = {6'h0, max(tmp0[9:0], Switch10(tmp0[9:0]))}
  qv[ 31: 16] = {6'h0, max(tmp1[9:0], Switch10(tmp1[9:0]))}
  qv[ 47: 32] = {6'h0, max(tmp2[9:0], Switch10(tmp2[9:0]))}
  qv[ 63: 48] = {6'h0, max(tmp3[9:0], Switch10(tmp3[9:0]))}
  qv[ 79: 64] = {6'h0, max(tmp4[9:0], Switch10(tmp4[9:0]))}
  qv[ 95: 80] = {6'h0, max(tmp5[9:0], Switch10(tmp5[9:0]))}
  qv[111: 96] = {6'h0, max(tmp6[9:0], Switch10(tmp6[9:0]))}
  qv[127:112] = {6'h0, max(tmp7[9:0], Switch10(tmp7[9:0]))}

  rs1[31:0] = rs1[31:0] + 8
```

调度

操作数	读流水级数	写流水级数
qv	-	1
rs1	1	1

格式

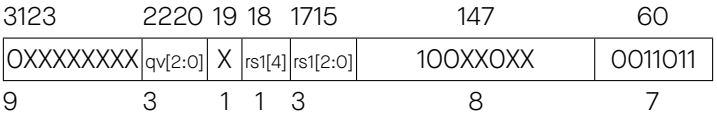


图 4.7-352. ESP.FFT.BITREV

4.7.2.353 ESP.SRC.Q.QUP

语法

ESP.SRC.Q.QUP qz,qw,qy

描述

此指令对寄存器 qw 和 qy 的 32 字节拼接结果执行算术右移，这两个寄存器保存了从两个连续对齐地址加载的数据。通过这种方式，可以获得未对齐的 16 字节数据，并将其写入寄存器 qz。右移的位数是 SAR_BYTE 的值乘以 8 bit。同时，寄存器 qw 的值更新为 qy。

操作

```
qz[127:  0] = {qy[127:  0], qw[127:  0]} >> {SAR_BYTE[3:0] << 3}
qw = qy
```

调度

操作数	读流水级数	写流水级数
qz	-	1
qw	1	-
qy	1	-
SAR_BYTES	1	-

格式

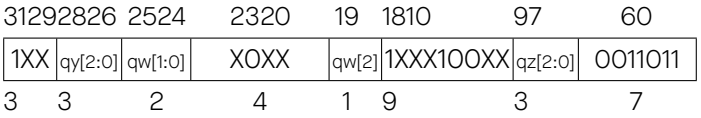


图 4.7-353. ESP.SRC.Q.QUP

4.7.2.354 ESP.VSL.S32

语法

ESP.VSL.S32 qu,qy,sat

描述

此指令对寄存器qy中的四个 32 位有符号数据段分别进行左移。左移的位数是 6 位特殊寄存器SAR中的值。在移位过程中，低位用 0 填充。移位结果的低 32 位存储在寄存器qu中对应的数据段中。

操作

```
qu[ 31:  0] = (qy[ 31:  0] << SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] << SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] << SAR[5:0])
qu[127: 96] = (qy[127: 96] << SAR[5:0])
```

调度

操作数	读流水级数	写流水级数
qu	-	1
qy	1	-
SAR	1	-

格式

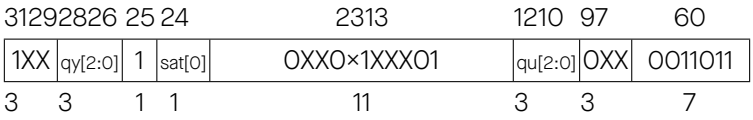


图 4.7-354. ESP.VSL.S32

4.7.2.355 ESP.VSL.U32

语法

ESP.VSL.U32 qu,qy,sat

描述

此指令对寄存器 qy 中的四个 32 位无符号数据段分别执行左移。左移量由 6 位专用寄存器 SAR 的值决定。移位过程中低位补 0。移位结果的低 32 位存入寄存器 qu 的对应数据段。

操作

```
qu[ 31:  0] = (qy[ 31:  0] << SAR[5:0])
qu[ 63: 32] = (qy[ 63: 32] << SAR[5:0])
qu[ 95: 64] = (qy[ 95: 64] << SAR[5:0])
qu[127: 96] = (qy[127: 96] << SAR[5:0])
```

流水线调度

操作数	Use	Def
qu	-	1
qy	1	-
SAR	1	-

格式

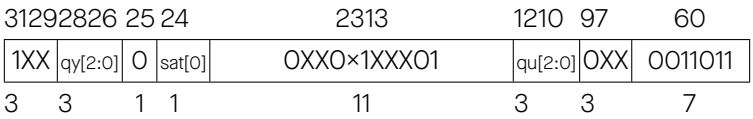


图 4.7-355. ESP.VSL.U32

第 5 章

低功耗处理器

5.1 概述

ESP32-P4 低功耗处理器 (Low-Power CPU, LP CPU) 是基于 RISC-V ISA 的 32 位处理器，包括整数 (I)、乘法/除法 (M)、原子 (A) 和压缩 (C) 标准扩展。LP CPU 内核具有 2 级有序标量流水线，具有极低的功耗。CPU 内核架构包含中断控制器 (INTC)、调试模块 (DM) 和用于访问存储器和外设的系统总线 (SYS BUS) 接口。

LP CPU 默认处于睡眠状态（详见章节 5.10），在芯片进入 Deep-sleep 时可保持上电状态（详见章节 15 低功耗管理），能够访问绝大部分外设和存储器，详情请参考 9 系统和存储器 章节。LP CPU 有两种应用场景：

- 功耗不敏感场景：高性能处理器 (HP CPU) 处于工作状态，LP CPU 可协助 HP CPU 完成一些对速度和效率不敏感的控制或计算
- 功耗敏感场景：HP CPU 处于掉电状态来节省功耗时，可以唤醒 LP CPU 处理一些外部唤醒事件

HP CPU 和 LP CPU 可访问的资源如图 5.1-1 所示：

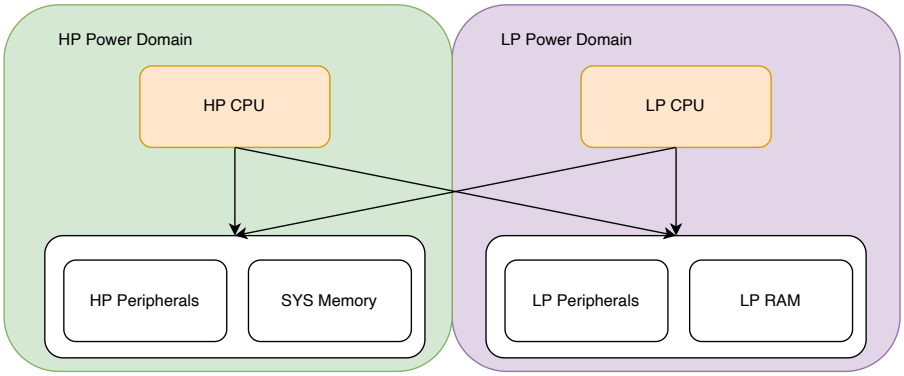


图 5.1-1. LP CPU 概览

在上图中，HP 电源域 (HP Power Domain) 和 LP 电源域 (LP Power Domain) 可分别上下电，当其中一个电源域处于掉电状态时，另一个电源域的 CPU 无法访问该域的资源，只有当该域处于上电状态时才可访问其资源，更多关于电源的信息，请参考章节 15 低功耗管理。

5.2 特性

LP CPU 具有以下特性：

- 时钟工作频率高达 40 MHz
- 支持 18 个向量中断
- 调试模块 (DM) 符合 RISC-V 调试规范 v0.13，支持通过行业标准的 JTAG/USB 端口连接外部调试器

- 硬件触发器符合 RISC-V 调试规范 v0.13，具有 2 个断点/观察点
- 支持核心性能指标事件
- 可唤醒 HP CPU 或向 HP CPU 发送中断
- 可访问 HP 存储器和 LP 存储器
- 可访问所有外设空间

5.3 配置与状态寄存器 (CSR)

5.3.1 寄存器列表

下表为 CPU 可访问的 CSR 列表。除了自定义的性能计数器 CSR 外，所有已实现的 CSR 都遵循 RISC-V 指令集手册 V1.10 第二卷“特权架构”(RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10) 中所述的位域标准映射。必须注意的是，受 CPU 中实现的功能子集的限制，即使在标准 CSR 中也并非实现了所有位域。有关详细的 CSR 寄存器描述，请参阅章节 5.3.2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
机器模式信息 CSR			
mhartid	机器模式 hart ID 寄存器	0xF14	RO
机器模式陷阱设置 CSR			
mstatus	机器模式状态寄存器	0x300	R/W
misa [1]	机器模式 ISA 寄存器	0x301	R/W
mie	机器模中断使能寄存器	0x304	R/W
mtvec [2]	机器模式陷阱向量寄存器	0x305	R/W
机器模式陷阱处理 CSR			
mscratch	机器模式暂存寄存器	0x340	R/W
mepc	机器模式陷阱程序计数器	0x341	R/W
mcause [3]	机器模式陷阱原因寄存器	0x342	R/W
mtval	机器模式陷阱值寄存器	0x343	R/W
mip	机器模中断等待寄存器	0x344	R/W
触发器模块 CSR（与调试模块共用）			
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
调试模式 CSR			
dcsr	调试模式控制与状态寄存器	0x7B0	R/W
dpc	调试模式 PC 寄存器	0x7B1	R/W
dscratch0	调试模式暂存寄存器 0	0x7B2	R/W
dscratch1	调试模式暂存寄存器 1	0x7B3	R/W
机器模式计数器/定时器 CSR			

[1]尽管 misa 具有 R/W 属性，但由于它的域是硬连线的，所以写操作无效。在 RISC-V 术语中称为 WARL（写入任意数值读取合法数值）。
[2]mtvec 仅支持在向量模式下对陷阱处理进行配置，基地址为 256 字节对齐。
[3]mcause 中反映的外部中断 ID 也包括 RISC-V 标准为处理器内部中断源预留的 ID。

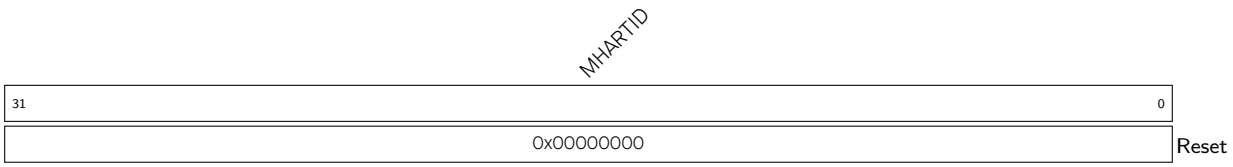
名称	描述	地址	访问
mcycle	机器模式时钟周期计数器寄存器	0xB00	R/W
minstret	机器模式退役指令计数器寄存器	0xB02	R/W
mhpmcounter*n*(*n*:3-12)	机器模式性能监测计数器寄存器	0xB00+*n*	R/W
mcycleh	mcycle 的高 32 位	0xB80	R/W
minstreth	minstret 的高 32 位	0xB82	R/W
mhpmcounter*n*h(*n*:3-12)	mhpmcounter*n*(*n*:3-12) 的高 32 位	0xB80+*n*	R/W
机器模式计数器设置 CSR			
mcountinhibit	机器模式计数器控制寄存器	0x320	R/W

请注意，如果对上表中只读属性的任何 CSR 尝试执行写入/置位/清除操作，CPU 将生成非法指令异常。

5.3.2 寄存器

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 5.1. mhartid (0xF14)



MHARTID 表示硬件线程编号。LP CPU 硬件线程编号为 0。(RO)

Register 5.2. mstatus (0x300)



MIE 写 1 使能全局机器模式中断。(R/W)

MPIE 写 1 使能（陷阱）之前的机器模式中断。(R/W)

MPP 配置（陷阱）之前的机器特权模式。

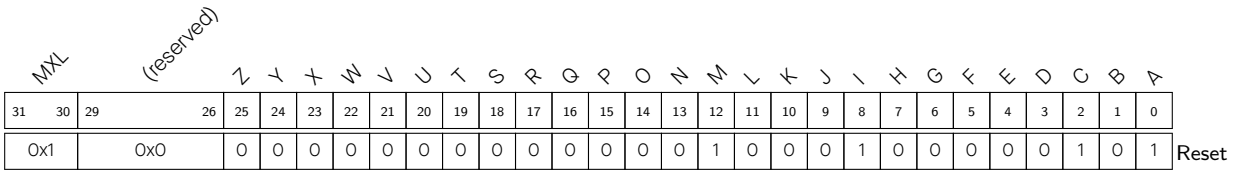
0x3：机器模式

其他值：无效

注意：仅低位可写。由于高位直接绑定低位，写入高位将被忽略。

(R/W)

Register 5.3. misa (0x301)



MXL	(reserved)	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
31 30	29 26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1

Reset

MXL 机器 XLEN = 1（32 位）。(RO)

Z 保留 = 0。(RO)

Y 保留 = 0。(RO)

X 非标准扩展 = 0。(RO)

W 保留 = 0。(RO)

V 保留 = 0。(RO)

U 实现用户模式 = 0。(RO)

T 保留 = 0。(RO)

S 实现监督模式 = 0。(RO)

R 保留 = 0。(RO)

Q 四精度浮点扩展 = 0。(RO)

P 保留 = 0。(RO)

O 保留 = 0。(RO)

N 支持用户级别中断 = 0。(RO)

M 整数乘除法标准扩展 = 1。(RO)

L 保留 = 0。(RO)

K 保留 = 0。(RO)

J 保留 = 0。(RO)

I RV32I 基本 ISA = 1。(RO)

H 虚拟机管理程序扩展 = 0。(RO)

G 其他标准扩展 = 0。(RO)

F 单精度浮点扩展 = 0。(RO)

E RV32E 基本 ISA = 0。(RO)

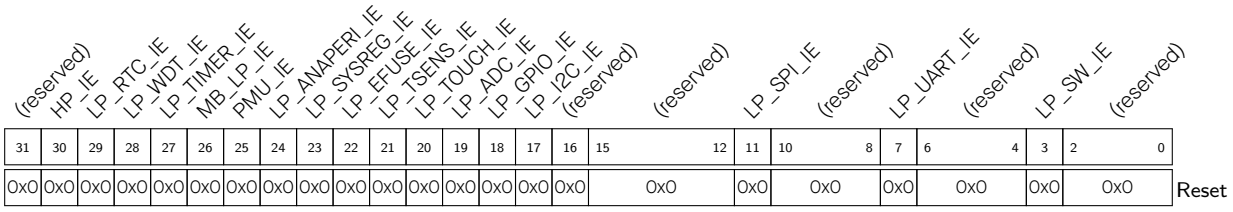
D 双精度浮点扩展 = 0。(RO)

C 压缩标准扩展 = 1。(RO)

B 保留 = 0。(RO)

A 原子标准扩展 = 1。(RO)

Register 5.4. mie (0x304)



LP_SW_IE 写 1 使能 LP 软件中断。(R/W)

LP_UART_IE 写 1 使能 LP UART 中断。(R/W)

LP_SPI_IE 写 1 使能 LP SPI 中断。(R/W)

LP_I2C_IE 写 1 使能 I2C 中断。(R/W)

LP_GPIO_IE 写 1 使能 GPIO 中断。(R/W)

LP_ADC_IE 写 1 使能 ADC 中断。(R/W)

LP_TOUCH_IE 写 1 使能 TOUCH 中断。(R/W)

LP_TSENS_IE 写 1 使能 TSENS 中断。(R/W)

LP_EFUSE_IE 写 1 使能 EFUSE 中断。(R/W)

LP_SYSREG_IE 写 1 使能 LP SYSREG 中断。(R/W)

LP_ANAPERI_IE 写 1 使能 ANAPERI 中断。(R/W)

LP_PMU_IE 写 1 使能 PMU 中断。(R/W)

LP_MB_IE 写 1 使能 Mailbox 中断。(R/W)

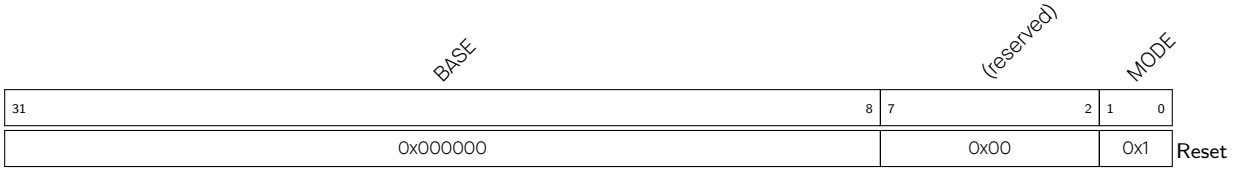
LP_TIMER_IE 写 1 使能 LP TIMER 中断。(R/W)

LP_WDT_IE 写 1 使能 WDT 中断。(R/W)

LP_RTC_IE 写 1 使能 RTC 中断。(R/W)

HP_IE 写 1 使能 HP 中断。(R/W)

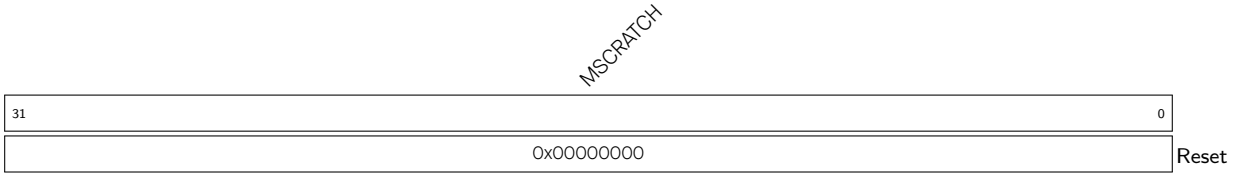
Register 5.5. mtvec (0x305)



MODE 表示机器模式中断是否为向量模式。仅支持向量模式 **0x1**。(RO)

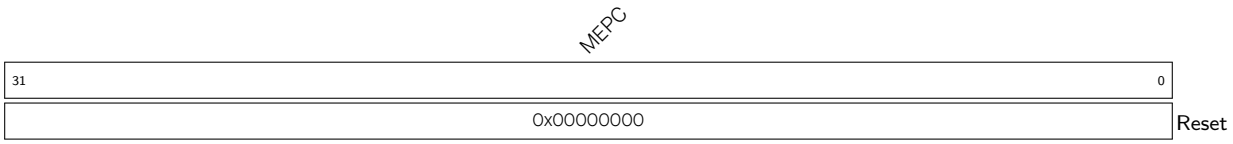
BASE 配置陷阱向量基址的高 24 位，地址为 256 字节对齐。(R/W)

Register 5.6. mscratch (0x340)



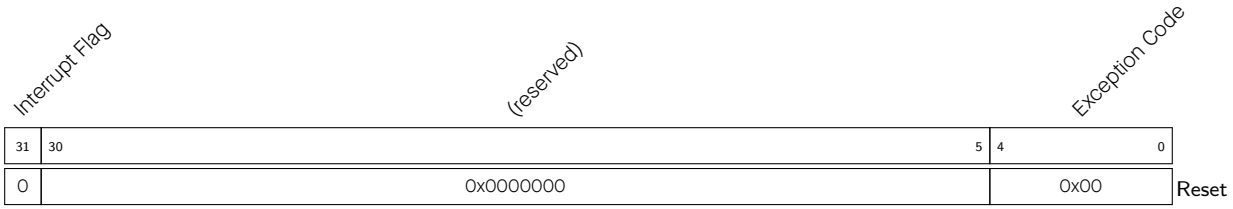
MSCRATCH 保存用户自定义的机器暂存信息。(R/W)

Register 5.7. mepc (0x341)



MEPC 配置机器陷阱/异常程序计数器。当 CPU 遇到陷阱时，此域将自动更新为 CPU 将要执行的指令的地址。(R/W)

Register 5.8. mcause (0x342)



Exception Code CPU 进入异常时，此域将自动更新为最近的陷阱或中断的唯一 ID。可能的异常 ID：

0x2：非法指令

0x3：硬件断点/观察点或 EBREAK

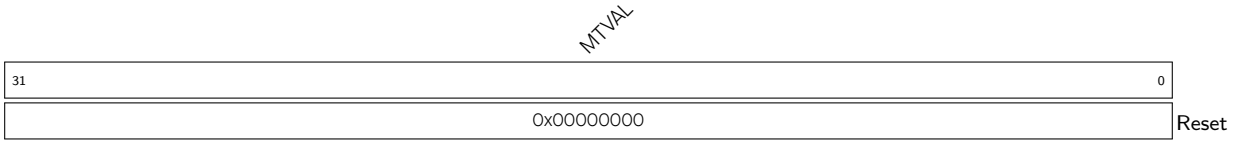
0x6：原子指令非对齐

注意：异常 ID 0x0（指令地址非对齐）不存在，因为 CPU 在取指时始终屏蔽地址的最低位。

(R/W)

Interrupt Flag CPU 进入陷阱时，此标志位将自动更新。如果被置位，则表示最近的陷阱是由中断引起。在异常情况下保持为 0。(R/W)

Register 5.9. mtval (0x343)



MTVAL 配置机器模式陷阱值。将自动更新为与异常有关的数据，该数据可能有助于处理该异常。根据异常编号有以下解读：

0x1：指令地址错误

0x2：指令 opcode 错误

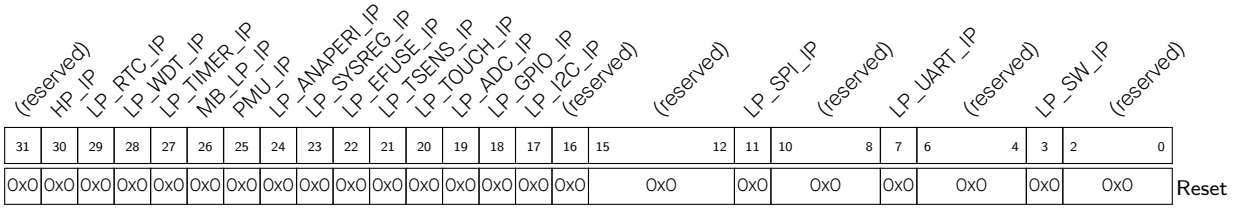
0x5：存储器读操作的数据地址错误

0x7：存储器写操作的数据地址错误

注意：该寄存器不支持其他异常 ID 和中断

(R/W)

Register 5.10. mip (0x344)



LP_SW_IP 配置 LP 软件中断的等待状态。

0：不等待

1：等待

(R/W)

LP_UART_IP 配置 LP UART 中断的等待状态。

0：不等待

1：等待

(R/W)

LP_SPI_IP 配置 LP SPI 中断的等待状态。

0：不等待

1：等待

(R/W)

LP_I2C_IP 配置 LP I2C 中断的等待状态。

0：不等待

1：等待

(R/W)

LP_GPIO_IP 配置 LP GPIO 中断的等待状态。

0：不等待

1：等待

(R/W)

LP_ADC_IP 配置 LP ADC 中断的等待状态。

0：不等待

1：等待

(R/W)

LP_TOUCH_IP 配置 LP TOUCH 中断的等待状态。

0：不等待

1：等待

(R/W)

LP_TSENS_IP 配置 LP TSENS 中断的等待状态。

0：不等待

1：等待

(R/W)

LP_EFUSE_IP 配置 LP EFUSE 中断的等待状态。

0：不等待

1：等待

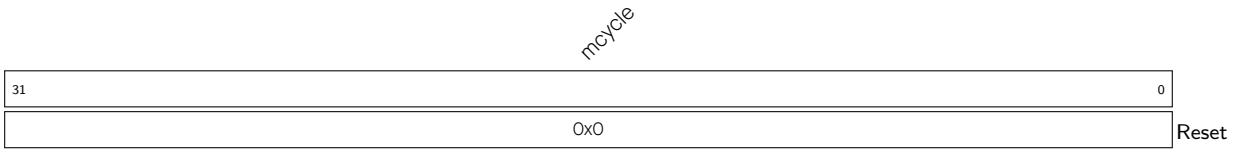
(R/W)

LP_SYSREG_IP 配置 LP SYSREG 中断的等待状态。

0：不等待

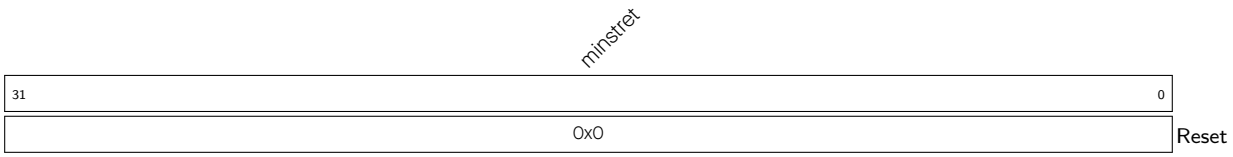
1：等待

Register 5.11. mcycle (0xB00)



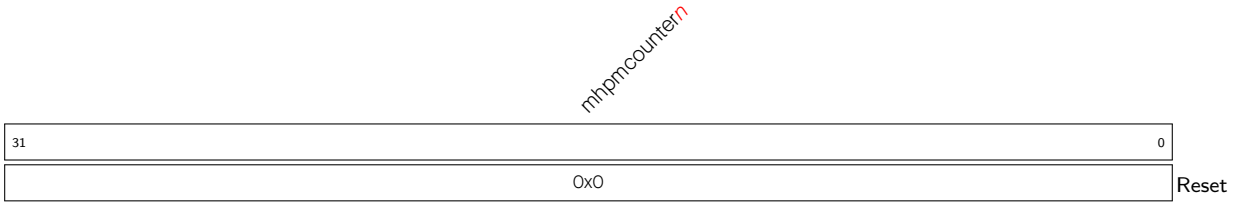
MCYCLE 配置时钟周期计数器的低 32 位。(R/W)

Register 5.12. minstret (0xB02)



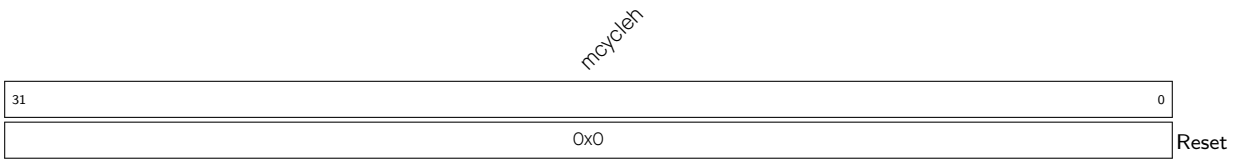
MINSTRET 配置指令计数器的低 32 位。(R/W)

Register 5.13. mhpmcounter*n*(*n*: 3-12) (0xB00+*n*)



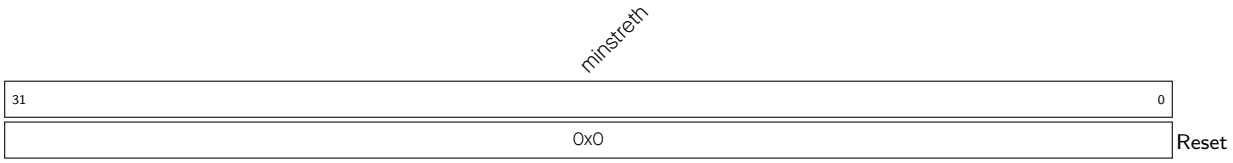
MHPMCOUNTER*n* 配置性能计数器 *n* 的低 32 位。(R/W)

Register 5.14. mcycleh (0xB80)



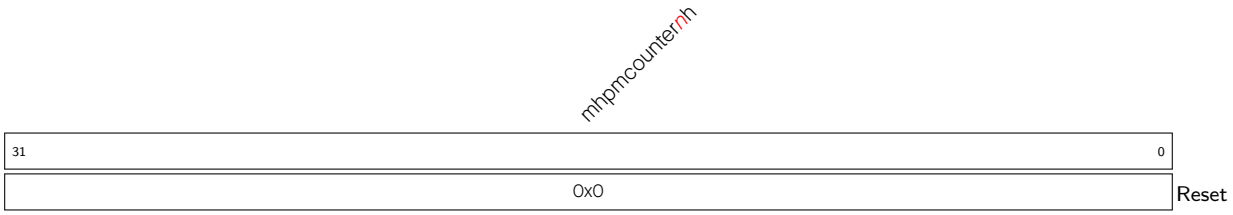
MCYCLEH 配置时钟周期计数器的高 32 位。(R/W)

Register 5.15. minstreth (0xB82)



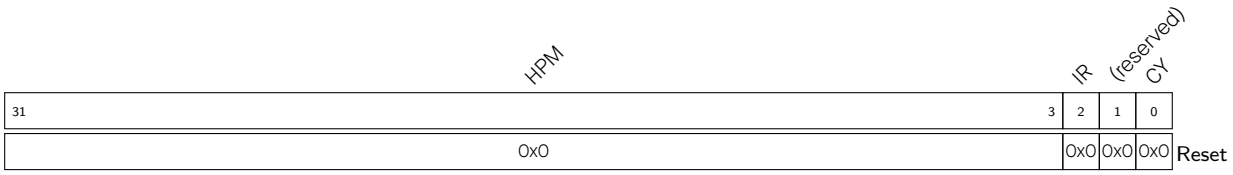
MINSTRETH 配置指令计数器的高 32 位。(R/W)

Register 5.16. mhpmcounter*n*(*n*: 3-12)h (0xB80+*n*)



MHPMCOUNTER*n*h 配置性能计数器 *n* 的高 32 位。(R/W)

Register 5.17. mcountinhibit (0x320)



HPM 配置性能计数器 *n*(*n*:3-12) 是否递增计数。

0：不计数

1：递增计数

(R/W)

IR 配置指令计数器是否递增计数。

0：不计数

1：递增计数

(R/W)

CY 配置时钟周期计数器是否递增计数。

0：不计数

1：递增计数

(R/W)

5.4 中断和异常

低功耗 CPU 根据 RISC-V 指令集手册 V1.10 第二卷 “特权架构” 实现了中断和异常的处理。当 CPU 进入中断/异常处理程序时，内核将执行以下操作：

- 将当前 PC 保存到 mepc CSR 中
- 将 mstatus 的 MIE 保存到 mstatus 的 MPIE
- 将当前特权模式保存到 mstatus 的 MPP
- 将 mstatus 的 MIE 清 0
- 特权模式切换到机器模式
- 跳转到处理程序地址
  - 对于所有异常原因，跳转到 mtvec CSR 中向量表的基地址
  - 对于中断，跳转到 $mtvec + 4 * ID$ (ID 为中断的编号，参见章节 5.4.1)
- 执行 mret 指令后，内核跳转到之前保存在 mepc CSR 中的程序计数器，并将 mstatus 的 MIE 恢复为 mstatus 的 MPIE，将当前特权模式恢复为 mstatus 的 MPP

当内核启动时，向量表的基地址被初始化为启动地址 0x50000000。启动后可通过写入 mtvec CSR 更改基地址。有关配置和状态寄存器的详细信息，请参阅章节 5.3.1。

内核在复位后从 LP_SYSTEM_LP_CORE_BOOT_ADDR_REG 寄存器指定的地址偏移 0x80 开始取指，寄存器默认值为 LP ROM 起始地址 0x50100000。

5.4.1 中断

ESP32-P4 低功耗 CPU 支持 18 个中断入口，每个中断对应一个 ID，假定为 n (n = 3, 7, 11, 16 ~ 30)，每个中断都有特定的入口地址 $mtvec + 4 * n$，每个中断都连接特定的外设源，图 5.4-1 列出了外设源与中断号的映射关系：

读取寄存器 LPINTR_STATUS_REG（只读）中特定位的值可以获取 LP CPU 外部中断源当前的状态。寄存器 LPINTR_STATUS_REG 与中断源的关系请参考寄存器说明。

表 5.4-1. LP CPU 中断映射

中断 ID	外设源
3	LP_SW_INTR
7	LP_UART_INTR
11	LP_SPI_INTR
17	LP_I2C_INTR
18	LP_GPIO_INTR
19	LP_ADC_INTR
20	LP_TOUCH_INTR
21	LP_TSENS_INTR
22	LP_EFUSE_INTR
23	LP_SYSREG_INTR
24	LP_ANAPERI_INTR

中断 ID	外设源
25	PMU_REG_0_INTR、PMU_REG_1_INTR
26	MB_HP_INTR、MB_LP_INTR
27	LP_TIMER_REG_0_INTR、LP_TIMER_REG_1_INTR
28	LP_WDT_INTR
29	LP_RTC_INTR
30	HP_INTR

5.4.2 中断处理

默认情况下，全局禁用中断，因为 mstatus 的 MIE 的复位值为 0。软件必须设置该位以启用全局中断。

假定使能的中断 ID 为 n：

1. 中断使能
   - 置位 mstatus 的 MIE 域，启用全局中断使能
   - 置位 mie CSR 的第 n 比特，使能中断 n
2. 中断使能后，LP CPU 即可响应中断，同时去配置相应外设的中断，使其能够触发中断信号给 LP CPU。
3. 触发中断后 LP CPU 会跳转到地址 $mtvec + 4 * n$。
4. 中断清除，只需要清除各外设的中断信号即可。

5.4.3 异常

低功耗 CPU 支持 RISC-V 标准异常，能够触发以下异常：

表 5.4-2. LP CPU 异常原因

异常编号	描述
1	指令访问错误
2	非法指令
3	断点 (EBREAK)
5	加载访问错误
6	原子指令非对齐
7	存储访问错误

5.5 调试

本节介绍如何调试和测试低功耗 CPU。调试功能由标准 JTAG 管脚提供，并符合 RISC-V 外部调试支持规范 v0.13 (RISC-V External Debug Support, Version 0.13)。

ESP32-P4 系统调试结构请参考章节 2 高性能处理器 中的图 2.11-1 调试系统概述。

用户与运行调试器（Debugger，例如 GDB）的调试主机（DEBUG HOST，例如笔记本电脑）进行交互。调试器通过调试转换器（Debug Translator，可能包含硬件驱动，例如 OpenOCD）与调试传输硬件（DEBUG TRANSPORT，例如 ESP-Prog 适配器）进行通信。调试传输硬件通过标准 JTAG 接口将调试主机连接到处理器内核的调试传输模块 (JTAG DTM)。JTAG DTM 使用调试模块接口 (DMI) 提供对调试模块 (DM) 的访问。

ESP32-P4 有两个 DTM 通过菊花链结构互联，其中 TAP0 用于连接 HP CPU，TAP1 用于连接 LP CPU。

HP 调试模块 (DM) 支持 RISC-V 外部调试支持规范 v0.13 的多核调试，能够同时控制 HP 双核 CPU。LP 调试模块 (DM) 仅调试单核 LP CPU。

低功耗 CPU 实现了四个内核调试寄存器：dcsr、dpc、dscratch0 和 dscratch1。所有这些寄存器只能从调试模式访问，如果软件试图在内核未处于调试模式的情况下进行访问，则会触发非法指令异常。

5.5.1 特性

低功耗 CPU 调试模块支持以下特性：

- 访问有关实现的必要信息
- 支持暂停和恢复 CPU 内核
- 读写 CPU 内核寄存器（包括 CSR）
- 复位 CPU 内核
- 在软件断点（植入的断点指令）上暂停 CPU
- 硬件单步调试
- 支持两个硬件触发器（可用作断点/观察点），具体见章节 5.6

5.5.2 功能描述

调试机制遵守 RISC-V 外部调试支持规范 v0.13。有关调试功能的详细介绍，请参考 RISC-V 外部调试支持规范 v0.13 (RISC-V External Debug Support, Version 0.13)。

根据 RISC-V 外部调试支持规范，hart 存在 nonexistent、unavail、running、halted 等状态，低功耗 CPU 默认处于 unavail 状态，需要用户配置 LPPERI_CPU_REG 寄存器清除 unavail 状态，从而能够连接低功耗 CPU 进行调试。

5.5.3 寄存器列表

下表列出了低功耗 CPU 支持的调试 CSR。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
dcsr	调试控制和状态寄存器	0x7B0	R/W
dpc	调试 PC 寄存器	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试暂存寄存器 1	0x7B3	R/W

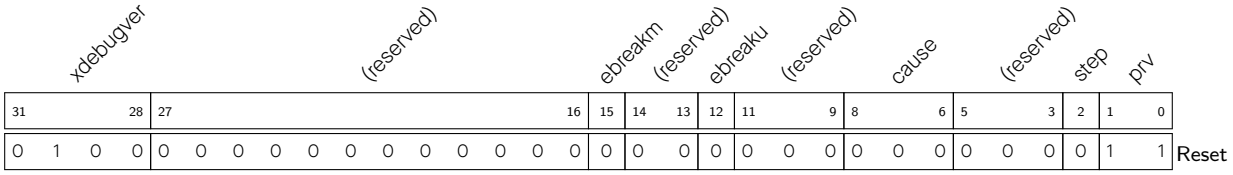
所有调试模块寄存器的实现均符合 RISC-V 外部调试支持规范 v0.13。请参考 RISC-V 外部调试支持规范 v0.13 获取详细信息。

5.5.4 寄存器

以下是低功耗 CPU 支持的调试 CSR 的详细描述。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 5.18. dcsr (0x7B0)



xdebugver 表示调试版本。

4：存在外部调试支持

(RO)

ebreakm 配置机器模式下执行 EBREAK 指令的行为。

0：触发异常，mcause 是 3。

1：进入调试模式

(R/W)

ebreaku 配置用户模式下执行 EBREAK 指令的行为。

0：如特权模式所述，触发异常，mcause 是 3。

1: 进入调试模式

(R/W)

cause 表示进入调试模式的原因。当单个周期中有多个原因导致进入调试模式，会反映出具有最高优先级数值的那个原因。

1：执行了一条 EBREAK 指令（优先级 3）

2：触发模块引起暂停（优先级 4）

3：haltreq 被置位（优先级 2）

4：step 被置位导致 CPU 单步执行（优先级 1）

其他值：保留，供日后使用

(RO)

step 当被置位且不处于调试模式时，内核将仅执行单个指令，然后进入调试模式。

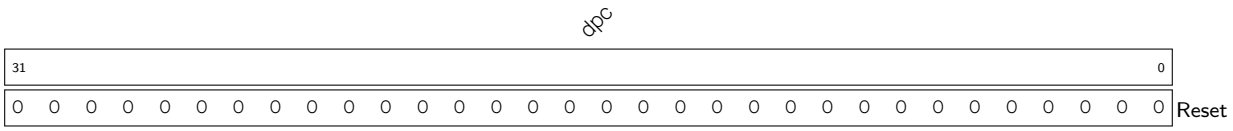
如果指令由于异常而未能完成，则内核将在执行异常处理程序之前立即进入调试模式，并置位相应的异常寄存器。

设置该位不会屏蔽中断，这与 RISC-V 外部调试支持规范 v0.13 中的规定不同。

(R/W)

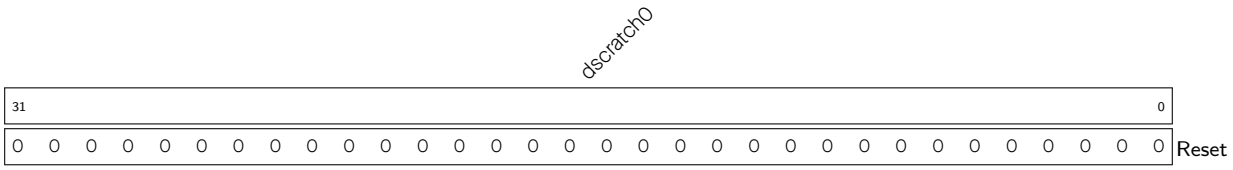
prv 保存 CPU 进入调试模式时候的特权级别。退出调试模式时，调试器可以更改此值以改变内核的特权级别。仅支持 **0x3**（机器模式）和 **0x0**（用户模式）。(RO)

Register 5.19. dpc (0x7B1)



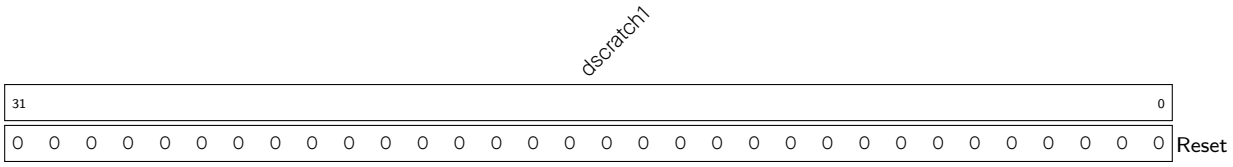
dpc 进入调试模式时，dpc 会更新为将要执行的下一条指令的地址。恢复时，PC 设置为存储在 dpc 中的地址。调试模式可能会修改 dpc。可在调试模式下访问。(R/W)

Register 5.20. dscratch0 (0x7B2)



dscratch0 供调试模块内部使用。(R/W)

Register 5.21. dscratch1 (0x7B3)



dscratch1 供调试模块内部使用。(R/W)

5.6 硬件触发器

5.6.1 特性

硬件触发器模块提供了断点和观察点功能，供调试使用。硬件触发器具有以下特性：

- 2 个独立触发单元
- 每个单元都可以配置为匹配程序计数器的地址
- 可以暂停执行并将控制权转移给调试器

5.6.2 功能描述

硬件触发器模块提供了 3 个 CSR，见章节 5.6。其中，tdata1 和 tdata2 是抽象 CSR，也就是说它们是用于访问某个触发单元中的内部寄存器的影子寄存器，一次访问一个触发单元。

要选择特定的触发单元，需要将相应的编号 (0-1) 写入 tselect CSR。当写入有效数值时，抽象 CSR tdata1 和 tdata2 将自动匹配该触发单元的内部寄存器。每个触发单元都有两个内部寄存器，即 mcontrol 和 maddress，

它们分别与 tdata1 和 tdata2 匹配。

向 tselect 写入超过 1 的数值时会导致 tselect 写入 1。

由于软件或调试器可能需要知道所选触发器的类型以便正确解读 tdata1 和 tdata2，因此 tdata1 的 4 个位 (31-28) 对所选触发器的类型进行了编码。此域为只读访问属性，并且值始终为 0x2，代表触发器仅支持地址/数据匹配。因此，可以推断 tdata1 和 tdata2 会通过 mcontrol 和 maddress 被解读。RISC-V 外部调试支持规范 v0.13 提供了其他可能值的信息，但是该触发模块仅支持 0x2 类型。

一旦选定了触发单元，就可以通过置位 mcontrol CSR (tdata1) 中相应的域并将目标地址写入 maddress CSR (tdata2) 来对该触发单元进行配置。

5.6.3 触发执行流程

当触发器触发引起硬件线程暂停并进入调试模式时 (action = 1)：

- dpc 被设置为当前 PC（在解码阶段）
- dcsr 的 cause 域被设置为 2，表示暂停是由于触发器触发引起

5.6.4 寄存器列表

下表列出了 CPU 可访问的的触发模块 CSR，只有在机器模式下才可以对它们进行读写。

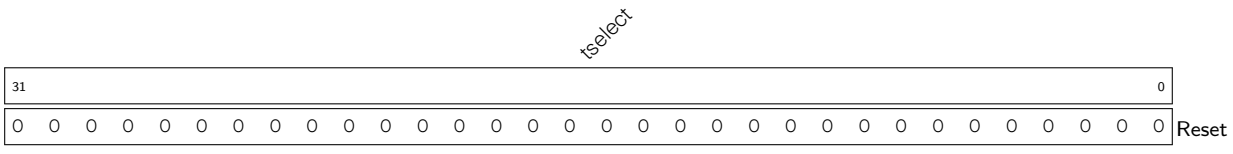
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
mcontrol	tdata1 影子寄存器	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
maddress	tdata2 影子寄存器	0x7A2	R/W

5.6.5 寄存器

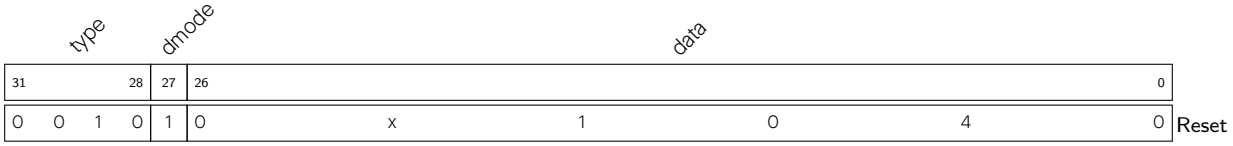
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 5.22. tselect (0x7A0)



tselect 配置选择触发器单元。(R/W)

Register 5.23. tdata1 (0x7A1)



type 表示触发器类型。仅支持匹配类型 (0x2)，此域保留。(RO)

dmode 如果某触发器正在被调试器使用，则此域置为 1。仅支持调试模式使用，此域保留。(RO)

data 配置抽象 tdata1 的内容。由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 mcontrol 的域。(R/W)

Register 5.24. tdata2 (0x7A2)



tdata2 配置抽象 tdata2 的内容。由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 maddress。(R/W)

Register 5.25. mcontrol (0x7A1)

(reserved)	dmode	maskmax	hit	select	timing	sizelo	action	chain	match	m	(reserved)	s	u	execute	store	load
31 28	27	26 21	20	19	18	17 16	15 12	11	10 7	6	5	4	3	2	1	0
0x2	0	0x0	0	0	0	0	0001	0	0	0	0	0	0	0	0	0

Reset

dmode 与 tdata1 的 dmode 一致。(RO)

maskmax 表示硬件支持的最大自然对齐的二次幂范围。

0：一个字节的范围，仅支持精确匹配

其他值：不支持

(RO)

hit 硬件未实现，始终为 0。(RO)

select 选择地址匹配或数据匹配。

0：匹配虚拟地址

1：匹配读取或写入的数据值或执行的指令

注意：只实现了地址匹配，此域总为 0。

(RO)

timing 表示触发器何时被触发。

0：当匹配后，触发器会在指令执行前触发

1：当匹配后，触发器会在指令执行后触发

注意：此域总为 0。

(RO)

sizelo 仅支持任意大小匹配，此域总为 0。(RO)

action 配置选定的触发器在触发时进行以下操作。

0x0：引起断点异常

0x1：进入调试模式（仅当 dmode = 1 时有效）

注意：仅支持进入调试模式，此域总为 1。

(RO)

CHAIN 硬件未实现，始终为 0。(RO)

match 配置触发器进行数据/指令地址的匹配操作。

0x0：精确字节匹配，即与访问中某个字节对应的地址必须精确匹配 maddress 的值。

0x1：NAPOT 匹配，即访问中至少有一个字节处于 maddress 中规定的 NAPOT 区域。

注意：仅支持精确字节匹配，此域总为 0。

(R/W)

m 置位使选定的触发器在机器模式下操作。(RO)

s 置位使选定的触发器在监督模式下操作。不支持在监督模式下操作，此域总为 0。(RO)

u 置位使选定的触发器在用户模式下操作。不支持在用户模式下操作，此域总为 0。(RO)

见下页……

Register 5.25. mcontrol (0x7A1)

接上页……

execute 置位使选定的触发器匹配指令的虚拟地址。

0：不使能

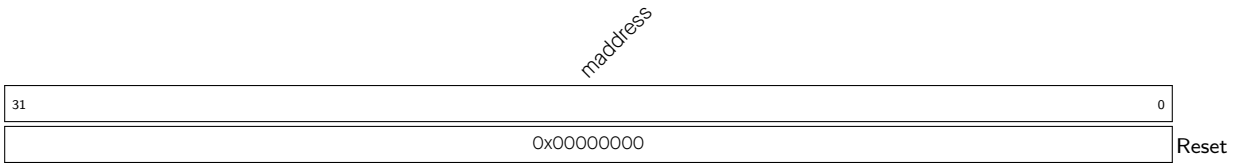
1：使能

(R/W)

store 置位使选定的触发器匹配存储器写操作的虚拟地址。硬件不支持，此域总为 0。(RO)

load 置位使选定的触发器匹配存储器读操作的虚拟地址。硬件不支持，此域总为 0。(RO)

Register 5.26. maddress (0x7A2)



maddress 配置选定的触发器执行匹配操作时使用的地址。(R/W)

5.7 性能计数器

低功耗 CPU 实现了时钟周期计数器 mcycle(h)、指令计数器 minstret(h) 以及 10 个事件计数器 mhpmcounter*n*(*n*:3-12) 。时钟周期计数器和指令计数器始终可用且为 64 位，其余性能计数器为 40 位。

默认情况下，所有计数器在复位后启用。通过 mcountinhibit CSR 中的相应位来单独启用/禁用对应的计数器。

每个计数器的计数事件都是固定的，参考表 5.7-1。

表 5.7-1. 性能计数器

计数器	计数事件
mcycle	统计时钟周期数
minstret	统计指令数量
mhpmcounter3	统计内存访问的等待周期
mhpmcounter4	统计取指等待周期
mhpmcounter5	统计内存读操作的数量，非对齐读视作两次。
mhpmcounter6	统计内存写操作的数量，非对齐写视作两次。
mhpmcounter7	统计无条件跳转指令的数量 (jal, jr, jalr)
mhpmcounter8	统计分支指令的数量
mhpmcounter9	统计分支指令跳转的数量
mhpmcounter10	统计压缩指令的数量
mhpmcounter11	统计乘法指令的等待周期

计数器	计数事件
mhpmcounter12	统计除法指令的等待周期

5.8 系统访问

LP CPU 可访问存储器和外设，当访问非法地址（不属于存储器或者外设地址空间），将根据访问类型触发对应的访问错误异常。

5.8.1 存储器访问

ESP32-P4 低功耗 CPU 可以访问的存储器资源有 LP ROM、LP SRAM 和 L2 SRAM，详情参考章节 9 系统和存储器。其中：

- LP ROM：16 KB, 地址范围为 0x5010_0000 ~ 0x5010_3FFF，可进行取指、读数据等操作
- LP SRAM：32 KB, 地址范围为 0x5010_8000 ~ 0x5010_FFFF，可进行取指、读数据、写数据等操作
- L2 SRAM：768 KB, 地址范围为 0x4FF0_0000 ~ 0x4FFB_FFFF，可进行取指、读数据、写数据等操作

说明：
LP CPU 访问 L2 SRAM 的延迟很大，一次访问大约需要 20 个 LP CPU 时钟周期，但可以无延迟访问 LP ROM 和 LP SRAM。

LP CPU 支持原子指令集，LP CPU 和 HP CPU 可通过原子指令访问存储器，从而实现内存访问的原子性。关于原子指令集的细节请参考 RISC-V 指令集手册 v2.2 第一卷“非特权架构”(RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Version 2.2)。

注意：只有 L2 SRAM 可以支持 HP CPU 和 LP CPU 的原子访问。

5.8.2 外设访问

章节 9 系统和存储器 表 9.3-2 详细列出了低功耗 CPU 可访问的外设及基地址。

5.9 事件任务矩阵功能

在 ESP32-P4 中，LP CPU 支持事件任务矩阵 (ETM) 功能，即可以通过任意外设的 ETM 事件触发 LP CPU 的 ETM 任务，或者通过 LP CPU 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 LP CPU 相关的 ETM 任务和 ETM 事件。

LP CPU 可接收的 ETM 任务有：

- ULP_TASK_WAKEUP_CPU：唤醒 LP CPU
- ULP_TASK_INT_CPU：触发 LP CPU 中断

LP CPU 可产生的 ETM 事件有：

- ULP_EVT_ERR_INTR：表示 LP CPU 发生异常
- ULP_EVT_START_INTR：表示 LP CPU 时钟打开

- ULP_EVT_HALT：表示 LP CPU 进入休眠状态

5.10 睡眠和唤醒流程

5.10.1 特性

以下是低功耗 CPU 支持的睡眠和唤醒特性：

- 低功耗 CPU 工作在低功耗系统中，可以在高性能系统进入睡眠的时候，独立进入睡眠、从睡眠中醒来以及进行工作
- 低功耗 CPU 可以根据软件运行状态，主动配置寄存器进入睡眠状态
- 低功耗 CPU 支持多个唤醒源
- 外设可拒绝 LP CPU 进入睡眠

5.10.2 流程

低功耗 CPU 默认处于睡眠状态，低功耗 CPU 的睡眠唤醒模块将按照如下流程唤醒开始工作以及进入睡眠。

LP CPU 的唤醒源配置如表 5.10-1 所示。

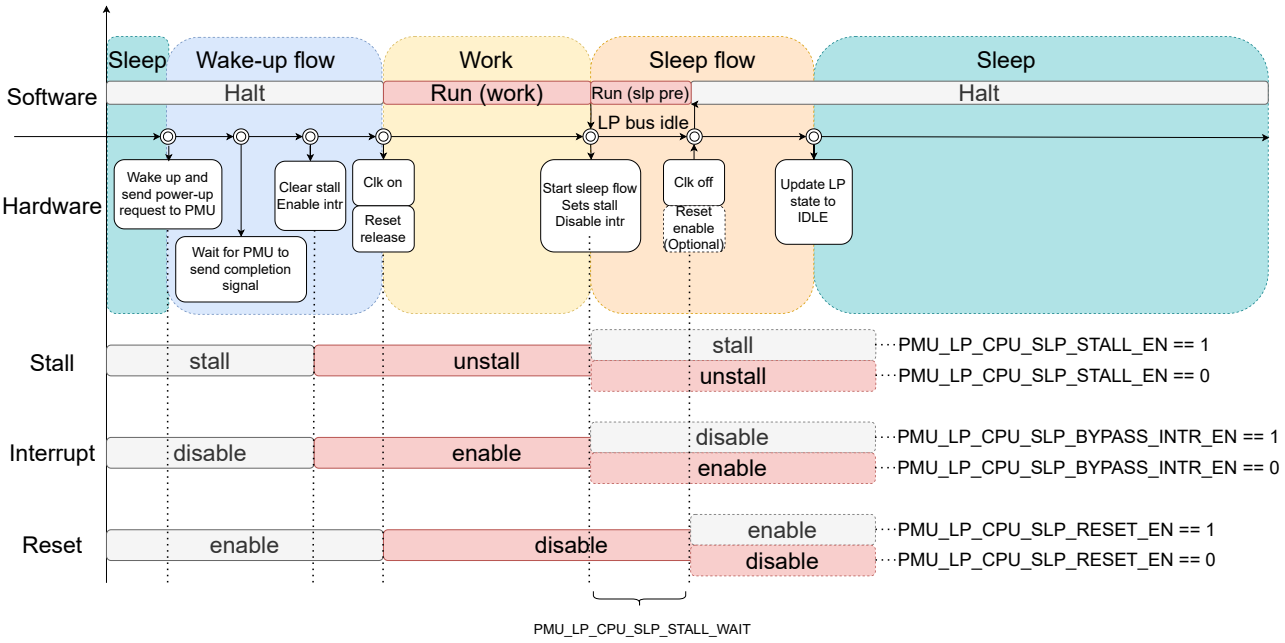


图 5.10-1. LP CPU 唤醒和睡眠流程

LP CPU 上电后的首次启动取决于 HP CPU 对能唤醒使能信号和唤醒源的配置。

- LP CPU 的初始化
  - 初始化 LP 存储器。
  - 启动 LP CPU。由于 LP CPU 的启动依赖于唤醒过程，所以推荐使用 PMU_HP_TRIGGER_LP 寄存器来开始 LP CPU 的初始化，方法如下：
    * 将 PMU_LP_CPU_WAKEUP_EN 设置为 0x400000

    * 将 PMU_HP_TRIGGER_LP 设置为 0x1
    * LP CPU 被唤醒后开始运行

- 唤醒启动流程 (Wake-up flow)：
  - 睡眠唤醒模块接收到唤醒信号向电池管理单元 (PMU) 发送上电请求。
  - 如果当前功耗状态（时钟、电源等）已经满足低功耗 CPU 需求，则 PMU 立刻回复完成信号，否则将调整功耗状态再回复完成信号。
  - 睡眠唤醒模块关闭低功耗 CPU 的 STALL 状态，并打开中断接收使能。
  - 睡眠唤醒模块开启时钟并释放复位（如果睡眠没有使能复位则忽略该步骤），然后开始工作。
- 睡眠流程 (Sleep flow)：
  - 低功耗 CPU 配置 PMU_LP_CPU_SLEEP_REQ 寄存器，睡眠唤醒模块启动睡眠流程。
  - 如果 PMU_LP_CPU_SLP_STALL_EN 为 1，睡眠唤醒模块则使能低功耗 CPU 的 STALL 状态，如果为 0，则不使能低功耗 CPU 的 STALL 状态。如果 PMU_LP_CPU_SLP_BYPASS_INTR_EN 为 1，睡眠唤醒模块则屏蔽低功耗 CPU 的所有中断，如果为 0，则不屏蔽 CPU 的中断。
  - 睡眠唤醒模块等待 PMU_LP_CPU_SLP_STALL_WAIT 个低功耗 CPU 时钟周期后，关闭低功耗 CPU 的时钟，如果 PMU_LP_CPU_SLP_RESET_EN 为 1，则使能低功耗 CPU 的复位。
  - 睡眠唤醒模块更新低功耗 CPU 的状态为 IDLE 状态，PMU 可以进入更深层次的睡眠。

5.10.3 唤醒源

表 5.10-1. 唤醒源

寄存器值[1]	唤醒源	描述
BIT(9)	LP IO	该唤醒源使用 LP IO 的中断状态寄存器信号，详情请见章节 10 *GPIO 交换矩阵和 IO MUX*
BIT(10)	LP UART	LP UART 接收一定数量的 RX 脉冲，需要使能 REG_UART_WAKEUP_EN，详情请见章节 45 *UART 控制器 (UART)*
BIT(13)	RTC 定时器	使用 RTC 定时器的 target 0 超时中断控制，详情请见章节 15 低功耗管理
BIT(14)	欠压检测	发生欠压时，可唤醒 LP CPU，详情请见章节 25 欠压检测器
BIT(17)	ETM	可接收来自 ETM 的唤醒源，详情请见章节 14 事件任务矩阵 *(ETM)*
BIT(18)	RTC 定时器	使用 RTC 定时器的 target 1 超时中断控制，详情请见章节 15 低功耗管理
BIT(19)	LP I2S 音频	I2S 接受到固定音频，可唤醒 LP CPU，详情请见章节 49 *I2S 控制器 (I2S)*
BIT(22)	寄存器 PMU_HP_TRIGGER_LP	通过主核置位寄存器 PMU_HP_TRIGGER_LP 可唤醒低功耗 CPU，置位 REG_HP_SW_TRIGGER_INT_CLR 可清除唤醒源

[1] 寄存器 PMU_LP_CPU_WAKEUP_EN 的值

5.10.4 拒绝睡眠

当低功耗 CPU 准备进入睡眠状态时，可以被外设事件拒绝，即外设事件可以阻止低功耗 CPU 进入睡眠。以下是外设事件阻止低功耗 CPU 进入睡眠状态的配置流程：

1. 低功耗 CPU 配置 PMU_SLP_REJECT_EN 使能外设阻止低功耗 CPU 进入睡眠状态的功能
2. 低功耗 CPU 配置 PMU_SLEEP_REJECT_ENA 使能对应外设阻止低功耗 CPU 进入睡眠状态

阻止低功耗 CPU 进入睡眠的外设源头与唤醒源一致。

第 II 卷

系统 DMA

该部分侧重描述系统内管理和传输数据，重点介绍直接内存访问 (DMA) 控制器，并为芯片与各种外设或设备之间的高效数据传输提供了指南。

第 6 章

通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)

6.1 概述

通用直接存储访问 (General Direct Memory Access, GDMA) 用于在外设与存储器之间以及存储器与存储器之间提供高速数据传输。软件可以在无需 CPU 干预的情况下通过 GDMA 快速搬移数据，从而降低了 CPU 的工作负载，提高了效率。

ESP32-P4 的 GDMA 控制器有两种，分别可以直接访问 AHB 或 AXI 总线，以下简称为 GDMA-AHB 和 GDMA-AXI，两种 GDMA 各有 6 个独立的通道，其中包括 3 个发送通道和 3 个接收通道。这 6 个通道被支持 GDMA 功能的外设所共享，用户可以将通道分配给任何支持 DMA 功能的外设。GDMA-AHB 支持的外设包括 I3C、UHCI、I2S、ADC、RMT，GDMA-AXI 支持的外设包括 LCD、CAM、两个通用 SPI、PARLIO、AES、SHA。在 ESP32-P4，GDMA-AHB 和 GDMA-AXI 内外存储皆可访问。

GDMA 控制器支持通道间可配置优先级及权重仲裁以管理外设不同的带宽需求。

下文中除非另有说明，否则“GDMA 控制器”指代 GDMA-AHB 和 GDMA-AXI。

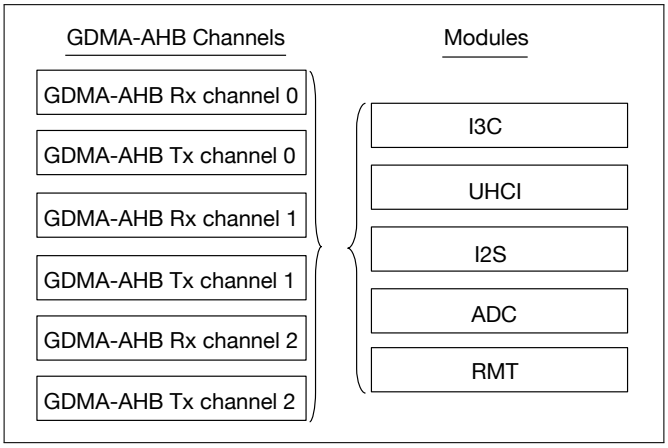


图 6.1-1. 共用 GDMA-AHB 通道的模块

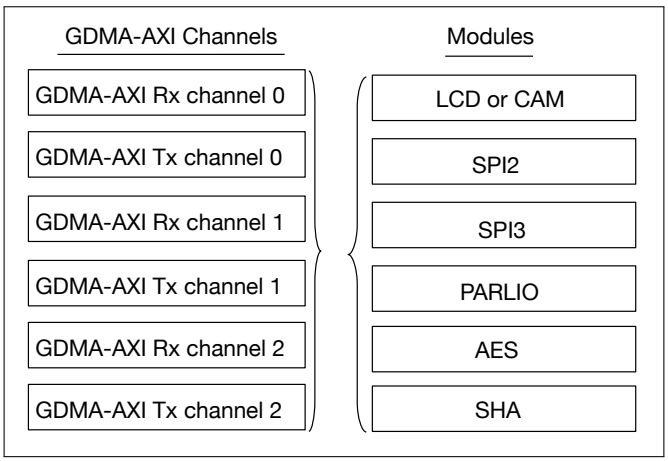


图 6.1-2. 共用 GDMA-AXI 通道的模块

6.2 特性

GDMA-AHB 和 GDMA-AXI 具有以下特性：

- 架构：
  - GDMA-AHB：AHB 总线架构
  - GDMA-AXI：AXI 总线架构，支持深度为 8 的乱序传输 (out of order) 和深度为 8 的（挂起传输）（outstanding 传输）
- 数据传输以字节为单位，传输数据量可软件编程
- 支持任意地址和大小 (size) 访问
- 对齐要求：
  - GDMA-AHB：
    - 描述符存储地址：字对齐
    - 数据地址和长度：
      - 内部存储器和外部存储器的非加密空间：无要求
      - 外部存储器的加密空间：16 字节对齐
  - GDMA-AXI：
    - 描述符存储地址：双字对齐
    - 数据地址和长度：
      - 内部存储器和外部存储器的非加密空间：无要求
      - 外部存储器的加密空间：16 字节对齐
- 支持链表
- 访问存储器时，GDMA-AHB 支持 INCR4、INCR8 或 INCR16 突发传输，GDMA-AXI 支持 INCR 突发传输
- 各有三个传输通道和三个接收通道
- 任一通道支持可配置的外设选择

- 支持通道间优先级和权重仲裁配置
- 支持存储搬运功能
- 支持数据传输 CRC 计算功能
- GDMA-AXI 支持软件切换链表中断响应机制

6.3 架构

ESP32-P4 中所有需要进行高速数据传输的模块都具有 GDMA 功能。GDMA 控制器与 CPU 的数据总线使用相同的地址空间访问存储器。图 6.3-1 为 GDMA 控制器基本架构图。

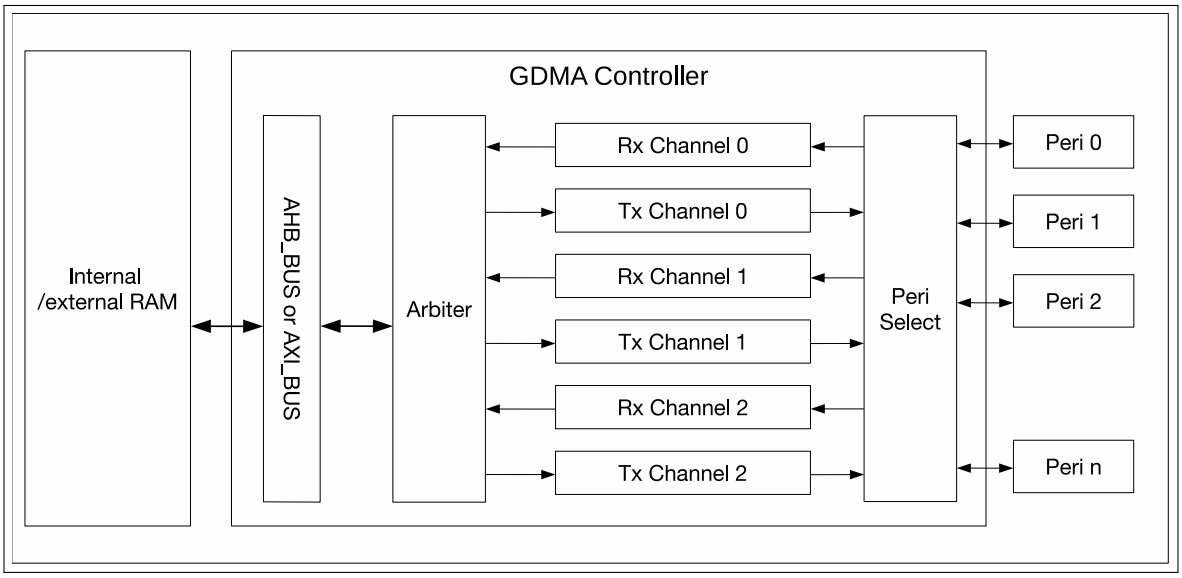


图 6.3-1. GDMA 控制器的架构

两种 GDMA 控制器各有 6 个独立的通道，其中包括 3 个发送通道和 3 个接收通道。每个通道可选择与不同的外设相连，从而实现通道资源由外设共享。

GDMA-AHB 和 GDMA-AXI 通过 AHB_BUS 或 AXI_BUS 将数据存入片内外存储器，或者将数据从片内外存储器取出。在通过总线传输数据之前，GDMA 采用可配置的仲裁机制对每个通道的读写请求进行仲裁。存储器的具体使用范围详见章节 9 *系统和存储器*。

软件可以通过挂载链表的方式来使用 GDMA 控制器。GDMA-AHB 链表存储在片内存储器中，GDMA-AXI 链表存储在片内或片外存储器中。这些链表包括 outlink*n* 与 inlink*n*，本文以 *n* 来表示通道号，*n* 为 0 ~ 2。GDMA 从存储器中取得链表，然后根据 outlink*n* 中的内容将相应存储器中的数据发送出去，也可根据 inlink*n* 中的内容将接收的数据存入指定存储器地址空间。

6.4 功能描述

6.4.1 链表

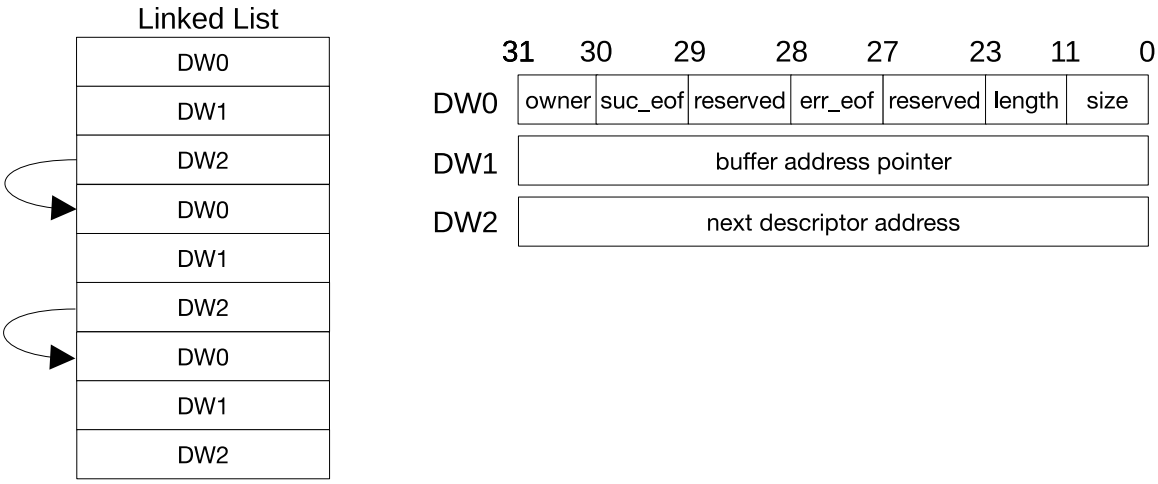


图 6.4-1. 链表结构图

图 6.4-1 所示为链表的结构图。发送链表与接收链表结构相同。每个链表由一个或者若干个描述符构成，一个描述符由 3 个字组成。链表应存放在存储器中，供 GDMA 控制器使用。描述符每一字段的意义如下：

- owner (DW0) [31]：表示当前描述符对应的 buffer 允许的操作者。
  0：允许的操作者为 CPU。
  1：允许的操作者为 GDMA 控制器。
  在 GDMA 使用完该描述符对应的 buffer 后，对于接收描述符，硬件默认会自动将该位清零；对于发送描述符，需要将 AHB/AXI_DMA_OUT_AUTO_WRBACK_CH*n* 置 1，硬件才会自动将该位清零。软件也可通过置位 AHB/AXI_DMA_OUT_LOOP_TEST_CH*n* 或 AHB/AXI_DMA_IN_LOOP_TEST_CH*n* 来关闭硬件自动清零的功能。软件在挂载链表时需要将该位置 1。

  注意：本文以 AHB/AXI_DMA_OUT 开头的寄存器对应 TX 通道寄存器，以 AHB/AXI_DMA_IN 开头的寄存器对应 RX 通道寄存器。

- suc_eof (DW0) [30]：表示一个描述符对应的数据成功传输后是否触发 AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT 或 AHB/AXI_DMA_OUT_EOF_CH*n*_INT 中断。
  0：当前描述符成功传输后不会触发中断；
  1：当前描述符成功传输后触发中断。
  对于接收描述符，需要软件将该位写 0，硬件会在接收到包含 EOF 标志的数据后将该位置 1。
  对于发送描述符，需要软件按需要将描述符中的该位置 1。如果软件将在某个描述符中将该位配置为 1，则 GDMA 在处理完该描述符时，会在发送给外设的数据中加入 EOF 标志，告知外设该段数据是一个阶段性结束。

- Reserved (DW0) [29]：保留。此位为无关项。

- err_eof (DW0) [28]：表示接收结束错误标志。
  0：接收数据没有错误。
  1：接收数据有错误。
  该位只用于 UHCI 或 PARLIO 利用 GDMA 接收数据。对于接收描述符，硬件在收完描述符对应的数据段并检测到接收数据错误会将该位置 1。

- Reserved (DW0) [27:24]：保留。

- length (DW0) [23:12]：表示当前描述符对应的 buffer 中的有效字节数。对于发送描述符，该段由软件填写，表示从 buffer 中读取数据时需要读取的字节数；对于接收描述符，该段由硬件使用完该 buffer 后或者接收到最后一个数据时自动填写，表示 buffer 中存储的有效字节数。
- size (DW0) [11:0]：表示当前描述符对应的 buffer 容量的字节数。
- buffer address pointer (DW1)：buffer 的地址。
- next descriptor address (DW2)：下一个描述符的地址。如果当前描述符为链表中最后一个描述符时，该值可以为 0。对于 GDMA-AHB，该地址必须指向片内存储器的地址空间；对于 GDMA-AXI，该地址必须指向片内外存储器的地址空间。

> **说明：**
> 需要注意，GDMA-AHB 的链表描述符本身存储地址要求 4 字节对齐，GDMA-AXI 要求 8 字节对齐。

用 GDMA 接收数据时，如果接收数据的长度小于当前描述符指定的 buffer size，那么下一个描述符对应的接收数据不会占用该 buffer 的剩余空间。

6.4.2 外设到存储及存储到外设的数据传输

GDMA 支持存储到外设及外设到存储的数据传输，分别对应 TX 及 RX 功能。TX 通道通过 outlink*n* 实现将指定存储区域中的数据搬运到外设的发送端；RX 通道通过 inlink*n* 实现将外设接收到的数据搬运到指定的存储区域。

每个 RX/TX 通道均可以被配置连接到任意一个支持 GDMA 功能的外设，表 6.4-1 和表 6.4-2 所示为通过寄存器配置通道与其对应外设的关系。其中 “Dummy-*n*” 是 GDMA 在实现存储到存储数据搬运功能时可选配的寄存器值。当其中一个通道已经与某一个外设连接时，其他通道将不能配置为与该外设连接。

表 6.4-1. GDMA-AHB 配置寄存器与外设选择关系表

AHB_DMA_PERI_IN_SEL_CH*n* AHB_DMA_PERI_OUT_SEL_CH*n*	外设
0	I3C
1	Dummy-1
2	UHCI
3	I2S0
4	I2S1
5	I2S2
6 ~ 7	Dummy-6 ~ 7
8	ADC
9	Dummy-9
10	RMT
11 ~ 15	Dummy-11 ~ 15
16 ~ 63	无效值

表 6.4-2. GDMA-AXI 配置寄存器与外设选择关系表

AXI_DMA_PERI_IN_SEL_CH*n*	

AXI_DMA_PERI_OUT_SEL_CH*n*	外设
0	LCD 或 CAM
1	SPI2
2	SPI3
3	PARLIO
4	AES
5	SHA
6 ~ 15	Dummy-6 ~ 15
16 ~ 63	无效值

6.4.3 存储到存储数据传输

GDMA 支持存储到存储的数据传输。置位 AHB/AXI_DMA_MEM_TRANS_EN_CH*n*，TX 通道*n* 的输出将与 RX 通道*n* 的输入相连，从而使能存储到存储的数据传输功能。需要注意的是，一个 TX 通道只与其编号对应的 RX 通道相连而实现存储到存储的数据传输，并且需要将 AHB/AXI_DMA_PERI_IN_SEL_CH*n* 和 AHB/AXI_DMA_PERI_OUT_SEL_CH*n* 配置成相同且对应为 Dummy 的值。

GDMA-AHB 仅通道 1、通道 2 支持存储到存储的数据传输。AHB_DMA_PERI_IN_SEL_CH*n* 和 AHB_DMA_PERI_OUT_SEL_CH*n* (*n* = 1 ~2) 需配置成相同且对应为 Dummy 的值，但 AHB_DMA_AHBINF_RESP_ERR_ID 抓取到的 ID 为固定值。ID 为 11 时，指 GDMA-AHB 通道 1 的存储到存储的传输；ID 为 12 时，指 GDMA-AHB 通道 2 的存储到存储传输。

6.4.4 启动 GDMA

软件通过挂载链表的方式来使用 GDMA。对于接收数据，软件挂载好接收链表并准备好接收数据，配置 AHB/AXI_DMA_INLINK_ADDR_CH*n* 字段指向第一个接收链表描述符，置位 AHB/AXI_DMA_INLINK_START_CH*n* 位启动 GDMA。对于发送数据，软件挂载好发送链表并准备好发送数据，配置 AHB/AXI_DMA_OUTLINK_ADDR_CH*n* 字段指向第一个发送链表描述符，置位 AHB/AXI_DMA_OUTLINK_START_CH*n* 位启动 GDMA。AHB/AXI_DMA_INLINK_START_CH*n* 与 AHB/AXI_DMA_OUTLINK_START_CH*n* 位由硬件自动清零。

有时您可能想要在 DMA 数据传输已经开始后追加更多描述符。要挂载更多描述符，原本看似只需清空已挂载链表最后一个描述符的 EOF 位，并将该描述符的 next descriptor address (DW2) 字段配置为新链表第一个描述符的地址。但如果 DMA 数据传输已经或即将结束，这个方法便无法奏效。GDMA 控制器有专门的逻辑来确保数据传输继续或重启：如果数据传输仍在进行，GDMA 控制器会确保顾及到新追加的描述符；如果数据传输已经结束，GDMA 控制器会重启数据传输，传输新追加的描述符。这个逻辑由 Restart 功能实现。

软件使用 Restart 功能时，需要重写已挂载链表的最后一个描述符，使其第三个字中的内容（即 DW2）指向新链表的首地址；然后置位 AHB/AXI_DMA_INLINK_RESTART_CH*n* 或者 AHB/AXI_DMA_OUTLINK_RESTART_CH*n*（这两个位由硬件自动清零），如图 6.4-2 所示，硬件会在读取已挂载链表的最后一个描述符时，获取新挂载链表的地址，从而继续处理新挂载的链表。

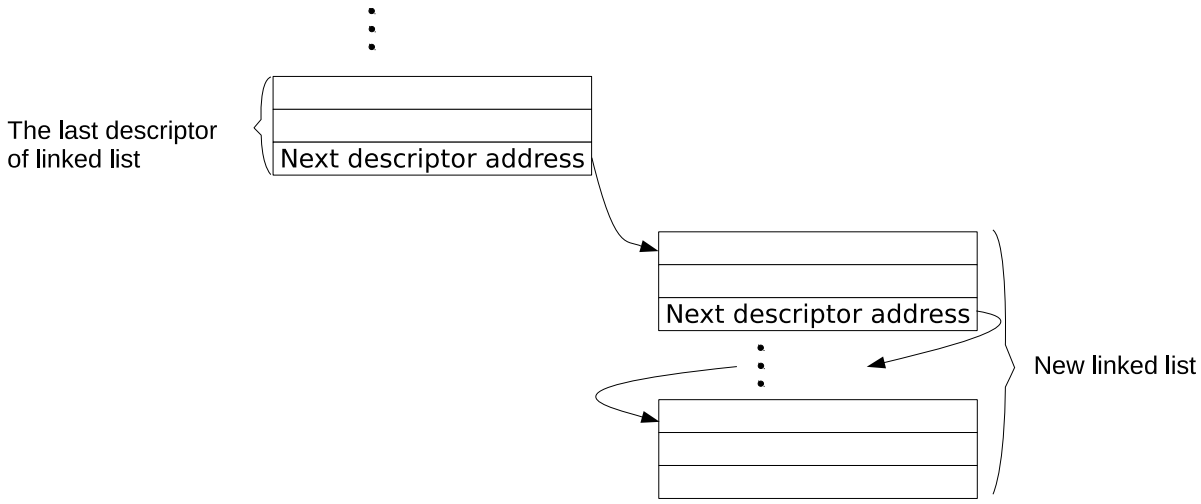


图 6.4-2. 链表关系图

6.4.5 读链表

软件在配置并启动 GDMA 后，GDMA 会从存储器读取链表。GDMA 会检查读入的链表描述符是否正确。只有当链表描述符通过检查时，GDMA 对应的通道才会开始搬运数据。当链表描述符没有通过检查，硬件将触发描述符错误中断（AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT 或者 AHB/AXI_OUT_DSCR_ERR_CH*n*_INT），同时该通道将会处于阻塞状态，停止工作。

描述符检查项包括：

- AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* 或者 AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* 置 1 时，检查描述符的 owner 位。如果该位为 0，表示当前操作者为 CPU，则不通过检查。AHB/AXI_DMA_IN_CHECK_OWNER_CH*n* 或者 AHB/AXI_DMA_OUT_CHECK_OWNER_CH*n* 置 0 时，不检查描述符的 owner 位。
- 检查描述符本身的存储地址是否在配置的存储器地址空间内。如果 GDMA-AHB 的描述符地址在 AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR 值的范围内，则通过检查。如果 GDMA-AXI 的描述符在 AXI_DMA_ACCESS_INTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_INTR_MEM_END_ADDR 或 AXI_DMA_ACCESS_EXTR_MEM_START_ADDR ~ AXI_DMA_ACCESS_EXTR_MEM_END_ADDR 值的范围内，则通过检查。请参见章节 6.4.8）。

软件在检查到通道描述符错误中断后，需要复位对应的通道，并置位 AHB/AXI_DMA_OUTLINK_START_CH*n* 或者 AHB/AXI_DMA_INLINK_START_CH*n* 位启动 GDMA。

6.4.6 GDMA-AXI 切换链表的中断响应机制

GDMA-AXI 支持在链表切换完成时触发中断。软件可将 AXI_DMA_LINK_SWITCH_STATE_CH*n* 配置为 1，使能 GDMA-AXI 切换链表。链表切换成功后，如果 AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT 已使能，GDMA-AXI 会产生该中断，并将 AXI_DMA_LINK_SWITCH_STATE_CH*n* 清零，表示链表已完成切换。

该功能仅用于状态指示，不影响 GDMA-AXI 的正常传输。

6.4.7 数据传输结束标志

注意： 本章中提到发送链表描述符的 EOF 为描述符中的第一个字中的 bit 30 指示的 suc_eof，接收链表描述符的 EOF 可以为 bit 30 指示的 suc_eof 和 bit 28 指示的 err_eof。

GDMA 通过 EOF 来指示对应描述符所需传输的数据段传输结束。

发送数据时，产生 EOF 和 TOTAL EOF 两种中断：

- AHB/AXI_DMA_OUT_EOF_CH*n*_INT：链表中的任意一个描述符有配置 suc_eof 标志位，则当该描述符对应的数据传输完成后，GDMA 都会会产生 EOF 中断。置位 AHB/AXI_DMA_OUT_EOF_CH*n*_INT_ENA 位使能该中断。
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT：如果该链表中的最后一个描述符配置了 suc_eof 标志位，则当该链表的最后一个描述符对应的数据传输完成后，GDMA 会产生 TOTAL EOF 中断。置位 AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA 位使能该中断。

接收数据时，可以产生以下中断：

- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT：表示带有 EOF 标志的数据段接收完成。置位 AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA 位使能该中断。
- （仅 UHCI 和 PARLIO）AHB/AXI_DMA_IN_ERR_CH*n*_EOF_INT：表示对应描述符所需传输的数据段接收完成但接收数据有错误。置位 AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT_ENA 使能该中断。需要注意的是，只有当通道连接的外设为 UHCI 或 PARLIO 时，才支持该中断。

软件在检测到 AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT 或 AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断时，可以通过获取 AHB/AXI_DMA_OUT_EOF_DES_ADDR_CH*n* 或 AHB/AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* 字段中的的值，获取当前已经完成数据传输的描述符的存储地址。这样，软件可以知道哪些描述符已经被使用并根据需要回收描述符。RX 方向还支持记录 AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT 中断触发时描述符的存储地址，可通过 AHB/AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n* 字段获取。

6.4.8 访问存储

GDMA-AHB 的 RX/TX 通道可访问片内外存储器，可访问的地址空间为寄存器 AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR 的配置值区间。

GDMA-AXI 的 RX/TX 通道可访问：

- AXI_DMA_ACCESS_INTR_MEM_START_ADDR 和 AXI_DMA_ACCESS_INTR_MEM_END_ADDR 配置的内部存储器区间；
- AXI_DMA_ACCESS_EXTR_MEM_START_ADDR 和AXI_DMA_ACCESS_EXTR_MEM_END_ADDR 配置的外部存储器区间。

为加速数据传输速率，支持突发传输模式。GDMA-AHB 默认不使能突发传输模式，可以通过置位 AHB_DMA_IN_DATA_BURST_EN_CH*n* 使能 RX 通道突发传输模式，置位 AHB_DMA_OUT_DATA_BURST_EN_CH*n* 使能 TX 通道突发传输模式。GDMA-AXI 突发传输模式默认打开且无法关闭，但是用户可以通过AXI_DMA_IN_BURST_SIZE_SEL_CH*n* 和 AXI_DMA_OUT_BURST_SIZE_SEL_CH*n* 来控制 AXI 总线突发传输传输的数据量，将突发传输长度配置为 8/16/32/64/128 字节。

GDMA-AHB 和 GDMA-AXI 访问配置的内部存储器空间和外部存储器的非加密空间时，链表描述符参数无对齐要求；访问外部存储器的加密空间（详见章节 34 *片外存储器加密与解密 (XTS_AES)*）时，buffer address pointer 和对应数据需按 16 字节对齐，其他参数无要求。

6.4.9 仲裁

为了确保及时响应 SPI 等高速低延迟外设的请求，GDMA 引入通道优先级仲裁和通道权重仲裁两种仲裁机制。

- 优先级仲裁方式：GDMA 支持 6（0 ~ 5）个等级的优先级，数字越大优先级越高。若多个通道在同一时刻存在数据传输需求，GDMA 会按照通道优先级从高到低顺序响应每个通道的数据传输请求；其数值越大，对应的优先级越高，请求响应越及时
- 权重仲裁方式：
  - 每个通道的的权重（即令牌数）可以配置为 0 ~ 15 之间的任意值。GDMA 将时间按照一定的总线 (AHB/AXI) 时钟周期长度划分成片，在每个时间片内，各个通道的数据传输请求次数应该将按照通道所配置的令牌数进行分配，在任意一个时间片内，某通道每请求一次数据传输消耗一次令牌，若某个通道在某个时间片内消耗尽令牌数，则该通道请求将不再被响应，直到仲裁时间达到时间片规定长度或者所有通道的令牌数均已经消耗尽，此时将刷新令牌数，进入下一个时间片周期内。
  - 若通道令牌数配置为非零值，即使无数据传输请求，也会在每个时间片内等待所有通道的令牌数耗尽，不会提前退出当前仲裁时间片。通道无数据传输请求且令牌值非 0 时，会导致一些带宽浪费，基于此，GDMA 提供额外的权重仲裁优化开关。权重仲裁优化打开时，如果通道无数据传输需求，则该通道令牌数会被认为是 0，不再参与仲裁，当所有通道均无数据传输请求时，仲裁器提前退出当前时间片，以此提高仲裁响应时间，提高传输速率。

说明：

- 若同时配置优先级仲裁和通道权重仲裁，且优先级不同，权重相同，则在各个通道传输相同的数据量，且数据量较少的前提下，优先级高的通道优先传输完成。
- 若优先级不同且权重也不同，各个通道传输数据量相同且较大，则权重较大的通道是能优先分配到大带宽的，会优先完成传输。

6.4.10 总线错误响应信息记录

在 GDMA-AHB 搬运数据期间，如果总线上从机侧回复了错误响应 (Error Response)，数据传输不会中断，GDMA-AHB 依旧会按照已有的链表搬运数据搬运。这样可能会导致完成数据搬运后，实际得到的数据与预期不符。为了便于定位问题，GDMA-AHB 传输数据过程中，如果发生了错误相应，可以通过以下内部寄存器存储相关传输信息：

- AHB_DMA_AHBINF_RESP_ERR_ADDR：记录本次总线传输对应的 32 位地址。
- AHB_DMA_AHBINF_RESP_ERR_CH_ID：记录本次总线传输使用的 GDMA-AHB 通道。
- AHB_DMA_AHBINF_RESP_ERR_ID：记录本次总线上外设与存储器传输或存储器与存储器传输的传输 ID。
- AHB_DMA_AHBINF_RESP_ERR_WR：记录本次总线传输对应的传输类型。

GDMA-AHB 作为主机发起 AHB 总线传输且收到错误响应时，还会产生错误相应中断来提醒 CPU。中断产生后，GDMA-AHB 仍不会中断数据传输。此外，GDMA-AHB 不会锁存第一笔发生错误响应的 AHB 总线传输相关寄存器信息。如果连续发生多笔错误响应传输，上述错误响应记录寄存器的内容会实时更新，最终寄存器中保留的信息是查询寄存器之前最后一笔发生错误响应的 AHB 总线传输。

6.4.11 自动时钟门控

GDMA-AHB 在工作时，某些场景只需要用到部分通道，此时可关闭未使用通道的时钟来降低功耗。同理，内部仲裁和总线接口模块仅在有数据传输时才需要工作，在空闲时也可以通过关闭时钟来降低功耗。目前 GDMA-AHB 内部有以下模块：

- 总线接口处理模块
- 仲裁模块
- RX 数据处理模块
- RX 描述符处理模块
- TX 数据处理模块
- TX 描述符处理模块
- 寄存器配置同步模块

这些模块可以在空闲时自动关闭模块时钟，来更大限度的减小功耗，也支持通过寄存器配置强制保持时钟开启。例如，置位 AHB_DMA_AHBINF_CLK_EN 可使总线接口处理模块时钟保持常开，其余模块时钟常开配置与其类似。

6.4.12 CRC 校验计算功能

目前 GDMA 在收发方向都添加了 CRC 计算模块，可以支持任意多项式。模块采用并行计算方法，每次计算 8 位。GDMA-AHB CRC 计算结果最大支持 32 位宽，GDMA-AXI 仅支持最大 16 位 CRC 计算结果。每次 CRC 计算以链表为单位，每次从链表的第一个描述符计算到最后一个描述符，如果传输过程中触发 Restart 功能，则本次 CRC 计算功能不再有效。

为解释寄存器配置原则，假设以下前提：

- 每拍要计算的传输数据记为 data[7:0]；
- 计算要求 5 位的 CRC 结果，使用的多项式为 $crc[4:0] = x \oplus 5 + x \oplus 2 + 1$；
- CRC 初始计算值为 0x1F，每个时钟周期计算的中间结果为 crc_tmp[4:0]；

以 GDMA 从存储器读取数据送入外设方向为例，硬件将提供以下几种字段，具体的配置方法如下：

- AHB/AXI_DMA_OUT_CRC_INIT_DATA_CH*n*：配置初始值，即上述例子中的 0x1F。AHB_DMA_OUT_CRC_INIT_DATA_CH*n* 最大支持配置 32 位初始值配置，也就是支持 32 位的 CRC 计算结果，AXI_DMA_OUT_CRC_INIT_DATA_CH*n* 只支持最高 16 位；
- AHB/AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n*：用于存储一次 DMA 传输（直到本次链表最后一个描述符完成）的所有数据的 CRC 计算结果，最大支持 32 位；
- AHB/AXI_DMA_TX_CRC_WIDTH_CH*n*：表示本次 CRC 计算结果的长度范围；
- AHB/AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n*：用于锁存 CRC 计算结果；
- AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*：表示 crc_tmp[4:0] 中的哪些位用于计算 crc[x]（x 通过AHB/AXI_DMA_TX_CRC_EN_ADDR_CH*n* 配置）。如果AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n* 配置为 0xF，则表示只有 crc_tmp[3]、crc_tmp[2]、crc_tmp[1] 和 crc_tmp[0] 参与计算，其他位（即 crc_tmp[4]）不参与计算。

- AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*：表示 data[7:0] 中的哪些位用于计算 crc[x]（x 通过 AHB/AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n* 配置）。GDMA 内部计算 CRC 结果以字节为单位，因此该字段只需要配置低 8 位。如果 AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* 配置为 0xexprefixF，则表示只有 data[3]、data[2]、data[1] 和 data[0] 参与计算。

在 GDMA 中，CRC 计算就是配置一个由计算多项式衍生的并行计算开关矩阵，假设需要计算 CRC 位宽为 n 位，则矩阵的规格为 n 行，$n+8$ 列。

由以上的前提假设，该矩阵规格为 5 行 × 13 列），则实际硬件 CRC 计算公式为（$\oplus$ 表示异或操作）：

$$crc[0] = crc_tmp[0] \oplus crc_tmp[2] \oplus crc_tmp[3] \oplus data[0] \oplus data[3] \oplus data[5] \oplus data[6]$$
$$crc[1] = crc_tmp[1] \oplus crc_tmp[3] \oplus crc_tmp[4] \oplus data[1] \oplus data[4] \oplus data[6] \oplus data[7]$$
$$crc[2] = crc_tmp[0] \oplus crc_tmp[3] \oplus crc_tmp[4] \oplus data[0] \oplus data[2] \oplus data[3] \oplus data[6] \oplus data[7]$$
$$crc[3] = crc_tmp[0] \oplus crc_tmp[1] \oplus crc_tmp[4] \oplus data[1] \oplus data[4] \oplus data[3] \oplus data[7]$$
$$crc[4] = crc_tmp[1] \oplus crc_tmp[2] \oplus data[2] \oplus data[4] \oplus data[5]$$

上述的计算公式可以抽象为图 6.4-3 所示的开关矩阵：

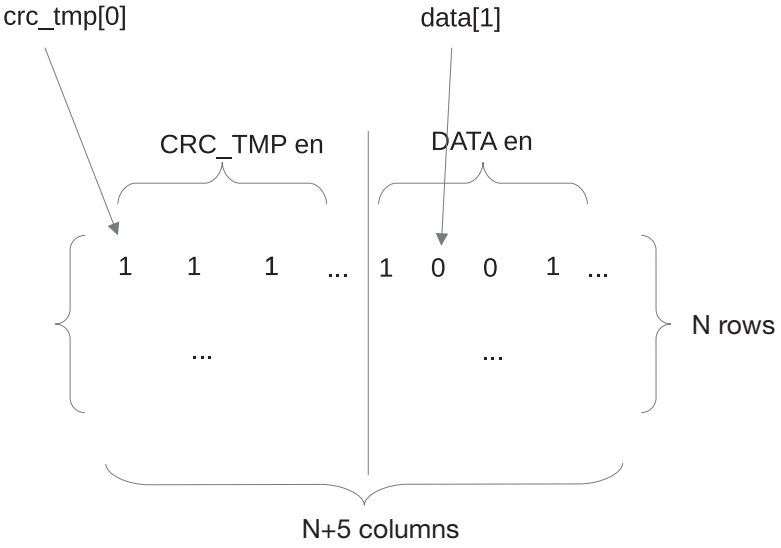


图 6.4-3. CRC 计算矩阵示意图

说明：
上述计算方法为 LSFR 的字节型方法，可以在相关 CRC 功能网站输入目标多项式和并行计算的数据位宽（在本芯片中并行计算位宽恒为 8）即可得到相关结果。

6.5 事件任务矩阵功能

在 ESP32-P4 中，GDMA 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GDMA 的 ETM 任务，或者通过 GDMA 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 GDMA 相关的 ETM 任务和 ETM 事件。

GDMA 可接收的 ETM 任务有：

- GDMA_AHB/AXI_TASK_IN_START_CH*n*：触发时开启 GDMA 的接收通道 *n* 进行数据传输。
- GDMA_AHB/AXI_TASK_OUT_START_CH*n*：触发时开启 GDMA 的发送通道 *n* 进行数据传输。

说明：
以上两个 ETM 任务与 CPU 配置 AHB/AXI_DMA_INLINK_START_CH*n*、AHB/AXI_DMA_OUTLINK_START_CH*n* 实现的功能相同。AHB/AXI_DMA_IN_ETM_EN_CH*n* 或 AHB/AXI_DMA_OUT_ETM_EN_CH*n* 为 1 时，只有 ETM 可以配置对应方向及对应通道 GDMA 的启动，AHB/AXI_DMA_IN_ETM_EN_CH*n* 或 AHB/AXI_DMA_OUT_ETM_EN_CH*n* 为 0 时，仅可通过 CPU 使能对应的 GDMA。

GDMA 可产生的 ETM 事件有：

- GDMA_AHB/AXI_EVT_IN_DONE_CH*n*：表示通道 *n* 接收方向链表描述符指向的数据接收完成。
- GDMA_AHB/AXI_EVT_IN_SUC_EOF_CH*n*：通道 *n* 接收方向 EOF 位为 1 的链表描述符指向的数据接收完成指示。
- GDMA_AHB/AXI_EVT_IN_FIFO_EMPTY_CH*n*：通道 *n* 接收方向 FIFO 非空到空状态改变指示。
- GDMA_AHB/AXI_EVT_IN_FIFO_FULL_CH*n*：通道 *n* 接收方向 FIFO 非满到满状态改变指示。
- GDMA_AHB/AXI_EVT_OUT_DONE_CH*n*：通道 *n* 表示通道 *n* 发送方向链表描述符指向的数据发送完成。
- GDMA_AHB/AXI_EVT_OUT_SUC_EOF_CH*n*：通道 *n* 发送方向 EOF 位为 1 的链表描述符指向的数据发送完成指示。
- GDMA_AHB/AXI_EVT_OUT_TOTAL_EOF_CH*n*：通道 *n* 发送方向最后一个并且 EOF 位为 1 的链表描述指向的数据发送符完成指示。
- GDMA_AHB/AXI_EVT_OUT_FIFO_EMPTY_CH*n*：通道 *n* 发送方向 FIFO 非空到空状态改变指示。
- GDMA_AHB/AXI_EVT_OUT_FIFO_FULL_CH*n*：通道 *n* 发送方向 FIFO 非满到满状态改变指示。

在具体应用中，GDMA 的 ETM 事件可以用来触发 GDMA 的 ETM 任务，例如，GDMA_AHB/AXI_EVT_OUT_TOTAL_EOF_CH0 事件可以触发 GDMA_AHB/AXI_TASK_IN_START_CH1 任务，从而触发新一轮的 GDMA 操作。

6.6 中断

ESP32-P4 的 GDMA 可以生成以下中断信号并发送给中断矩阵。

- AHB/AXI_PDMA_IN_CH0_INTR
- AHB/AXI_PDMA_IN_CH1_INTR
- AHB/AXI_PDMA_IN_CH2_INTR
- AHB/AXI_PDMA_OUT_CH0_INTR
- AHB/AXI_PDMA_OUT_CH1_INTR
- AHB/AXI_PDMA_OUT_CH2_INTR

这些中断信号由 GDMA 的内部中断源生成。

以下中断源生成 AHB/AXI_PDMA_IN_CH*n*_INTR (*n* 为 0 ~ 2) 中断信号：

- AHB/AXI_DMA_IN_DONE_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成时触发此中断。

- AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成，并且描述符的 suc_eof 为 1 时触发此中断。
- AHB/AXI_DMA_IN_ERR_EOF_CH*n*_INT：对于接收通道*n*，当接收的描述符对应数据段中有错误发生时触发此中断。（该中断只用于外设选择 UHCI (UART0/UART1) 或 PARLIO 时）。
- AHB/AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT：对于接收通道*n*，当接收链表描述符指向的 buffer size 小于待接收数据长度时触发此中断。
- AHB/AXI_DMA_IN_DSCR_ERR_CH*n*_INT：对于接收通道*n*，当接收链表描述符未通过任一描述符检查时触发此中断。
- AHB_DMA_INFIFO_OVF_CH*n*_INT 或 AXI_DMA_INFIFO_L1/L2/L3_OVF_CH*n*_INT：GDMA-AHB 接收数据方向缓存或 GDMA-AXI 接受数据方向 L1、L2、L3 缓存溢出时触发此中断。
- AHB/AXI_DMA_INFIFO_UDF_CH*n*_INT 或 AXI_DMA_INFIFO_L1/L2/L3_UDF_CH*n*_INT：GDMA-AHB 接收数据方向缓存或 GDMA-AXI 接受数据方向 L1、L2、L3 缓存无数据时触发此中断。
- AHB_DMA_IN_AHBINF_RESP_ERR_CH*n*_INT：GDMA-AHB 接收通道*n* 进行数据传输时，AHB 总线上如果有错误相应，触发此中断。

以下中断源生成 AHB/AXI_PDMA_OUT_CH*n*_INTR (*n* 为 0 ~ 2) 中断信号：

- AHB/AXI_DMA_OUT_DONE_CH*n*_INT：对于发送通道*n*，当一个发送链表描述符对应的数据发送完成时触发此中断。
- AHB/AXI_DMA_OUT_EOF_CH*n*_INT：对于发送通道*n*，当发送描述符的 EOF 位为 1，且 AHB/AXI_DMA_OUT_EOF_MODE_CH*n* 为 0 时，该描述符对应的最后一个数据由总线传入 GDMA TX 通道时，该中断触发；当发送描述符的 EOF 位为 1，且 AHB/AXI_DMA_OUT_EOF_MODE_CH*n* 为 1 时，该描述符对应的最后一个数据从 GDMA TX 通道发出时，该中断触发。
- AHB/AXI_DMA_OUT_DSCR_ERR_CH*n*_INT：对于发送通道*n*，当发送链表描述符未通过任一描述符检查时会触发此中断。
- AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT：对于发送通道*n*，当一个链表（可包含多个链表描述符）对应的所有数据都已发送完成时触发此中断。
- AHB_DMA_OUTFIFO_OVF_CH*n*_INT 或 AXI_DMA_OUTFIFO_L1/L2/L3_OVF_CH*n*_INT：GDMA-AHB 发送数据方向缓存或 GDMA-AXI 发送数据方向 L1、L2、L3 缓存溢出时触发此中断。
- AHB_DMA_OUTFIFO_UDF_CH*n*_INT 或 AXI_DMA_OUTFIFO_L1/L2/L3_UDF_CH*n*_INT：GDMA-AHB 发送数据方向缓存或 GDMA-AXI 发送数据方向 L1、L2、L3 缓存无数据时触发此中断。
- AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT：GDMA-AXI 发送通道 *n* 从一个链表切换到下一个链表时触发此中断。
- AHB_DMA_OUT_AHBINF_RESP_ERR_CH*n*_INT：GDMA-AHB 发送通道*n* 进行数据传输时，AHB 总线上如果有错误相应，触发此中断。

6.7 编程流程

可通过配置 HP_SYS_CLKRST_AHB/AXI_PDMA_SYS_CLK_EN 寄存器管理 GDMA 时钟门控，默认时钟门控打开。可通过配置 HP_SYS_CLKRST_RST_EN_AHB/AXI_PDMA 对 GDMA 模块全局复位。

6.7.1 GDMA TX 通道配置流程

利用 GDMA 发送数据时，GDMA TX 通道的软件配置流程如下：

1. 对寄存器 AHB/AXI_DMA_OUT_RST_CH*n* 置 1 然后置 0，复位 GDMA TX 通道状态机和 FIFO 指针；
2. 挂载好发送链表，配置寄存器 AHB/AXI_DMA_OUTLINK_ADDR_CH*n* 指向第一个发送链表描述符；
3. 配置 AHB/AXI_DMA_PERI_OUT_SEL_CH*n* 为对应的外设号，见表 6.4-1 和 6.4-2；
4. 置位 AHB/AXI_DMA_OUTLINK_START_CH*n* 启动 GDMA TX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；
6. 等待 AHB/AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT 中断，即数据传输完成。

6.7.2 GDMA RX 通道配置流程

利用 GDMA 接收数据时，GDMA RX 通道的软件配置流程如下：

1. 对寄存器 AHB/AXI_DMA_IN_RST_CH*n* 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针；
2. 挂载好接收链表，配置寄存器 AHB/AXI_DMA_INLINK_ADDR_CH*n* 指向第一个接收链表描述符；
3. 配置 AHB/AXI_DMA_PERI_IN_SEL_CH*n* 为对应的外设号，见表 6.4-1 和 6.4-2；
4. 置位 AHB/AXI_DMA_INLINK_START_CH*n* 启动 GDMA RX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；

6.7.3 GDMA 存储器到存储器配置流程

利用 GDMA 从存储到存储搬运数据时配置流程如下：

1. 对寄存器 AHB/AXI_DMA_OUT_RST_CH*n* 置 1 然后置 0，复位 GDMA TX 通道状态机和 FIFO 指针；
2. 对寄存器 AHB/AXI_DMA_IN_RST_CH*n* 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针；
3. 挂载好发送链表，配置寄存器 AHB/AXI_DMA_OUTLINK_ADDR_CH*n* 指向第一个发送链表描述符；
4. 挂载好接收链表，配置寄存器 AHB/AXI_DMA_INLINK_ADDR_CH*n* 指向第一个接收链表描述符；
5. 置位 AHB/AXI_DMA_MEM_TRANS_EN_CH*n* 使能存储器到存储器传输功能；
6. 置位 AHB/AXI_DMA_OUTLINK_START_CH*n* 启动 GDMA TX 通道发送数据；
7. 置位 AHB/AXI_DMA_INLINK_START_CH*n* 启动 GDMA RX 通道发送数据；
8. 如果某个发送链表描述符的 suc_eof 位配置为 1，则该描述符对应的数据段传输结束后会触发 AHB/AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。

6.7.4 通道优先级和权重配置流程

优先级仲裁配置方法如下：

1. 配置 AHB/AXI_DMA_TX_PRI_CH*n* 和 AHB/AXI_DMA_RX_PRI_CH*n* 为 TX 和 RX 通道的优先级。

权重仲裁配置方法如下：

1. 配置权重仲裁的时间片长度：

- GDMA-AHB：配置 AHB_DMA_ARB_TIMEOUT，同时适用于 TX 和 RX 方向；
- GDMA-AXI：TX 和 RX 方向通过 AXI_DMA_ARB_TIMEOUT_TX 和 AXI_DMA_ARB_TIMEOUT_RX 独立配置；

2. 配置 AHB/AXI_DMA_TX_CH_ARB_WEIGH_CH*n* 和 AHB/AXI_DMA_RX_CH_ARB_WEIGH_CH*n* 为各个通道的令牌值；
3. 清除 AHB/AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* 和AHB/AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*，使能 TX 和 RX 方向的权重仲裁优化；
4. 使能权重仲裁：
   - GDMA-AHB：置位同一字段 AHB_DMA_WEIGHT_EN，同时使能 TX 和 RX 方向；
   - GDMA-AXI：分别置位 AXI_DMA_WEIGHT_EN_TX 和 AXI_DMA_WEIGHT_EN_RX，使能 TX 和 RX 方向；

6.7.5 CRC 计算配置流程

1. 配置 AHB/AXI_DMA_TX_CRC_EN_ADDR_CH*n*、AHB/AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*、AHB/AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*、AHB/AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*，确定异或矩阵中，crc_tmp 和 data 中哪些位参与计算；
2. 配置 AHB/AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n* 为高，然后配置为低，锁存上述四个字段的值。至此矩阵中某一行的配置已完成。
3. 重复步骤 1 和步骤 2，直到矩阵的所有行的值都配置完成，即可启动 GDMA 进行数据传输。

6.8 寄存器列表

6.8.1 GDMA-AHB 寄存器列表

本小节的所有地址均为相对于 GDMA-AHB 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
AHB_DMA_IN_INT_RAW_CH0_REG	RX 通道 0 原始中断状态寄存器	0x0000	R/WTC/SS
AHB_DMA_IN_INT_ST_CH0_REG	RX 通道 0 屏蔽中断状态寄存器	0x0004	RO
AHB_DMA_IN_INT_ENA_CH0_REG	RX 通道 0 中断使能寄存器	0x0008	R/W
AHB_DMA_IN_INT_CLR_CH0_REG	RX 通道 0 中断清除寄存器	0x000C	WT
AHB_DMA_IN_INT_RAW_CH1_REG	RX 通道 1 原始中断状态寄存器	0x0010	R/WTC/SS
AHB_DMA_IN_INT_ST_CH1_REG	RX 通道 1 屏蔽中断状态寄存器	0x0014	RO
AHB_DMA_IN_INT_ENA_CH1_REG	RX 通道 1 中断使能寄存器	0x0018	R/W
AHB_DMA_IN_INT_CLR_CH1_REG	RX 通道 1 中断清除寄存器	0x001C	WT
AHB_DMA_IN_INT_RAW_CH2_REG	RX 通道 2 原始中断状态寄存器	0x0020	R/WTC/SS
AHB_DMA_IN_INT_ST_CH2_REG	RX 通道 2 屏蔽中断状态寄存器	0x0024	RO
AHB_DMA_IN_INT_ENA_CH2_REG	RX 通道 2 中断使能寄存器	0x0028	R/W
AHB_DMA_IN_INT_CLR_CH2_REG	RX 通道 2 中断清除寄存器	0x002C	WT
AHB_DMA_OUT_INT_RAW_CH0_REG	TX 通道 0 原始中断状态寄存器	0x0030	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH0_REG	TX 通道 0 屏蔽中断状态寄存器	0x0034	RO
AHB_DMA_OUT_INT_ENA_CH0_REG	TX 通道 0 中断使能寄存器	0x0038	R/W
AHB_DMA_OUT_INT_CLR_CH0_REG	TX 通道 0 中断清除寄存器	0x003C	WT
AHB_DMA_OUT_INT_RAW_CH1_REG	TX 通道 1 原始中断状态寄存器	0x0040	R/WTC/SS
AHB_DMA_OUT_INT_ST_CH1_REG	TX 通道 1 屏蔽中断状态寄存器	0x0044	RO
AHB_DMA_OUT_INT_ENA_CH1_REG	TX 通道 1 中断使能寄存器	0x0048	R/W
AHB_DMA_OUT_INT_CLR_CH1_REG	TX 通道 1 中断清除寄存器	0x004C	WT
AHB_DMA_OUT_INT_RAW_CH2_REG	TX 通道 2 原始中断状态寄存器	0x0050	R/WTC/SS

GoBack

名称	描述	地址	访问
AHB_DMA_OUT_INT_ST_CH2_REG	TX 通道 2 屏蔽中断状态寄存器	0x0054	RO
AHB_DMA_OUT_INT_ENA_CH2_REG	TX 通道 2 中断使能寄存器	0x0058	R/W
AHB_DMA_OUT_INT_CLR_CH2_REG	TX 通道 2 中断清除寄存器	0x005C	WT
调试寄存器			
AHB_DMA_AHB_TEST_REG	保留	0x0060	R/W
配置寄存器			
AHB_DMA_MISC_CONF_REG	杂项寄存器	0x0064	R/W
AHB_DMA_IN_CONF0_CH0_REG	RX 通道 0 的配置寄存器 0	0x0070	R/W
AHB_DMA_IN_CONF1_CH0_REG	RX 通道 0 的配置寄存器 1	0x0074	R/W
AHB_DMA_IN_POP_CH0_REG	RX 通道 0 的弹出控制寄存器	0x007C	varies
AHB_DMA_IN_LINK_CH0_REG	RX 通道 0 的链表描述符配置和控制寄存器	0x0080	varies
AHB_DMA_OUT_CONF0_CH0_REG	TX 通道 0 的配置寄存器 0	0x00D0	R/W
AHB_DMA_OUT_CONF1_CH0_REG	TX 通道 0 的配置寄存器 1	0x00D4	R/W
AHB_DMA_OUT_PUSH_CH0_REG	TX 通道 0 的推入控制寄存器	0x00DC	varies
AHB_DMA_OUT_LINK_CH0_REG	TX 通道 0 的链表描述符配置和控制寄存器	0x00E0	varies
AHB_DMA_IN_CONF0_CH1_REG	RX 通道 1 的配置寄存器 0	0x0130	R/W
AHB_DMA_IN_CONF1_CH1_REG	RX 通道 1 的配置寄存器 1	0x0134	R/W
AHB_DMA_IN_POP_CH1_REG	RX 通道 1 的弹出控制寄存器	0x013C	varies
AHB_DMA_IN_LINK_CH1_REG	RX 通道 1 的链表描述符配置和控制寄存器	0x0140	varies
AHB_DMA_OUT_CONF0_CH1_REG	TX 通道 1 的配置寄存器 0	0x0190	R/W
AHB_DMA_OUT_CONF1_CH1_REG	TX 通道 1 的配置寄存器 1	0x0194	R/W
AHB_DMA_OUT_PUSH_CH1_REG	TX 通道 1 的推入控制寄存器	0x019C	varies
AHB_DMA_OUT_LINK_CH1_REG	TX 通道 1 的链表描述符配置和控制寄存器	0x01A0	varies
AHB_DMA_IN_CONF0_CH2_REG	RX 通道 2 的配置寄存器 0	0x01F0	R/W
AHB_DMA_IN_CONF1_CH2_REG	RX 通道 2 的配置寄存器 1	0x01F4	R/W
AHB_DMA_IN_POP_CH2_REG	RX 通道 2 的弹出控制寄存器	0x01FC	varies
AHB_DMA_IN_LINK_CH2_REG	RX 通道 2 的链表描述符配置和控制寄存器	0x0200	varies
AHB_DMA_OUT_CONF0_CH2_REG	TX 通道 2 的配置寄存器 0	0x0250	R/W
AHB_DMA_OUT_CONF1_CH2_REG	TX 通道 2 的配置寄存器 1	0x0254	R/W

名称	描述	地址	访问
AHB_DMA_OUT_PUSH_CH2_REG	TX 通道 2 的推入控制寄存器	0x025C	varies
AHB_DMA_OUT_LINK_CH2_REG	TX 通道 2 的链表描述符配置和控制寄存器	0x0260	varies
AHB_DMA_OUT_CRC_INIT_DATA_CH0_REG	TX 通道 0 CRC 初始值配置寄存器	0x02BC	R/W
AHB_DMA_TX_CRC_WIDTH_CH0_REG	TX 通道 0 CRC 计算结果位宽配置寄存器	0x02C0	R/W
AHB_DMA_OUT_CRC_CLEAR_CH0_REG	TX 通道 0 CRC 计算结果清除寄存器	0x02C4	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH0_REG	TX 通道 0 CRC 计算结果寄存器	0x02C8	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH0_REG	CRC TX 通道 0 CRC 中间结果掩码寄存器	0x02CC	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH0_REG	TX 通道 0 CRC 中间结果掩码目标寄存器	0x02D0	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH0_REG	TX 通道 0 CRC 输入数据掩码寄存器	0x02D4	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH0_REG	TX 通道 0 CRC 输入数据掩码目标寄存器	0x02D8	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH0_REG	TX 通道 0 仲裁权重配置寄存器	0x02DC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH0_REG	TX 通道 0 权重仲裁优化使能寄存器	0x02E0	R/W
AHB_DMA_OUT_CRC_INIT_DATA_CH1_REG	TX 通道 1 CRC 初始值配置寄存器	0x02E4	R/W
AHB_DMA_TX_CRC_WIDTH_CH1_REG	TX 通道 1 CRC 计算结果位宽配置寄存器	0x02E8	R/W
AHB_DMA_OUT_CRC_CLEAR_CH1_REG	TX 通道 1 CRC 计算结果清除寄存器	0x02EC	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH1_REG	TX 通道 1 CRC 计算结果寄存器	0x02F0	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH1_REG	TX 通道 1 CRC 中间结果掩码寄存器	0x02F4	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH1_REG	TX 通道 1 CRC 中间结果掩码目标寄存器	0x02F8	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH1_REG	TX 通道 1 CRC 输入数据掩码寄存器	0x02FC	R/W
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH1_REG	TX 通道 1 CRC 输入数据掩码目标寄存器	0x0300	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH1_REG	TX 通道 1 仲裁权重配置寄存器	0x0304	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH1_REG	TX 通道 1 权重仲裁优化使能寄存器	0x0308	R/W
AHB_DMA_OUT_CRC_INIT_DATA_CH2_REG	TX 通道 2 CRC 初始值配置寄存器	0x030C	R/W
AHB_DMA_TX_CRC_WIDTH_CH2_REG	TX 通道 2 CRC 计算结果位宽配置寄存器	0x0310	R/W
AHB_DMA_OUT_CRC_CLEAR_CH2_REG	TX 通道 2 CRC 计算结果清除寄存器	0x0314	R/W
AHB_DMA_OUT_CRC_FINAL_RESULT_CH2_REG	TX 通道 2 CRC 计算结果寄存器	0x0318	RO
AHB_DMA_TX_CRC_EN_WR_DATA_CH2_REG	TX 通道 2 CRC 中间结果掩码寄存器	0x031C	R/W
AHB_DMA_TX_CRC_EN_ADDR_CH2_REG	TX 通道 2 CRC 中间结果掩码目标寄存器	0x0320	R/W
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH2_REG	TX 通道 2 CRC 输入数据掩码寄存器	0x0324	R/W

名称	描述	地址	访问
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH2_REG	TX 通道 2 CRC 输入数据掩码目标寄存器	0x0328	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH2_REG	TX 通道 2 仲裁权重配置寄存器	0x032C	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH2_REG	TX 通道 2 权重仲裁优化使能寄存器	0x0330	R/W
AHB_DMA_IN_CRC_INIT_DATA_CH0_REG	RX 通道 0 CRC 初始值配置寄存器	0x0334	R/W
AHB_DMA_RX_CRC_WIDTH_CH0_REG	RX 通道 0 CRC 计算结果位宽配置寄存器	0x0338	R/W
AHB_DMA_IN_CRC_CLEAR_CH0_REG	RX 通道 0 CRC 计算结果清除寄存器	0x033C	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH0_REG	RX 通道 0 CRC 计算结果寄存器	0x0340	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH0_REG	RX 通道 0 CRC 中间结果掩码寄存器	0x0344	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH0_REG	RX 通道 0 CRC 输入数据掩码目标寄存器	0x0348	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH0_REG	RX 通道 0 CRC 输入数据掩码寄存器	0x034C	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH0_REG	RX 通道 0 CRC 输入数据掩码目标寄存器	0x0350	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH0_REG	RX 通道 0 仲裁权重配置寄存器	0x0354	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH0_REG	RX 通道 0 权重仲裁优化使能寄存器	0x0358	R/W
AHB_DMA_IN_CRC_INIT_DATA_CH1_REG	RX 通道 1 CRC 初始值配置寄存器	0x035C	R/W
AHB_DMA_RX_CRC_WIDTH_CH1_REG	RX 通道 1 CRC 计算结果位宽配置寄存器	0x0360	R/W
AHB_DMA_IN_CRC_CLEAR_CH1_REG	RX 通道 1 CRC 计算结果清除寄存器	0x0364	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH1_REG	RX 通道 1 CRC 计算结果寄存器	0x0368	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH1_REG	RX 通道 1 CRC 中间结果掩码寄存器	0x036C	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH1_REG	RX 通道 1 CRC 输入数据掩码目标寄存器	0x0370	R/W
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH1_REG	RX 通道 1 CRC 输入数据掩码寄存器	0x0374	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH1_REG	RX 通道 1 CRC 输入数据掩码目标寄存器	0x0378	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH1_REG	RX 通道 1 仲裁权重配置寄存器	0x037C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH1_REG	RX 通道 1 权重仲裁优化使能寄存器	0x0380	R/W
AHB_DMA_IN_CRC_INIT_DATA_CH2_REG	RX 通道 2 CRC 初始值配置寄存器	0x0384	R/W
AHB_DMA_RX_CRC_WIDTH_CH2_REG	RX 通道 2 CRC 计算结果位宽配置寄存器	0x0388	R/W
AHB_DMA_IN_CRC_CLEAR_CH2_REG	RX 通道 2 CRC 计算结果清除寄存器	0x038C	R/W
AHB_DMA_IN_CRC_FINAL_RESULT_CH2_REG	RX 通道 2 CRC 计算结果寄存器	0x0390	RO
AHB_DMA_RX_CRC_EN_WR_DATA_CH2_REG	RX 通道 2 CRC 中间结果掩码寄存器	0x0394	R/W
AHB_DMA_RX_CRC_EN_ADDR_CH2_REG	RX 通道 2 CRC 输入数据掩码目标寄存器	0x0398	R/W

名称	描述	地址	访问
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH2_REG	RX 通道 2 CRC 输入数据掩码寄存器	0x039C	R/W
AHB_DMA_RX_CRC_DATA_EN_ADDR_CH2_REG	RX 通道 2 CRC 输入数据掩码目标寄存器	0x03A0	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH2_REG	RX 通道 2 仲裁权重配置寄存器	0x03A4	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH2_REG	RX 通道 2 权重仲裁优化使能寄存器	0x03A8	R/W
AHB_DMA_IN_LINK_ADDR_CH0_REG	RX 通道 0 链表描述符配置寄存器	0x03AC	R/W
AHB_DMA_IN_LINK_ADDR_CH1_REG	RX 通道 1 链表描述符配置寄存器	0x03B0	R/W
AHB_DMA_IN_LINK_ADDR_CH2_REG	RX 通道 2 链表描述符配置寄存器	0x03B4	R/W
AHB_DMA_OUT_LINK_ADDR_CH0_REG	TX 通道 0 链表描述符配置寄存器	0x03B8	R/W
AHB_DMA_OUT_LINK_ADDR_CH1_REG	TX 通道 1 链表描述符配置寄存器	0x03BC	R/W
AHB_DMA_OUT_LINK_ADDR_CH2_REG	TX 通道 2 链表描述符配置寄存器	0x03C0	R/W
AHB_DMA_INTR_MEM_START_ADDR_REG	可访问地址空间起始地址配置寄存器	0x03C4	R/W
AHB_DMA_INTR_MEM_END_ADDR_REG	可访问地址空间结束地址配置寄存器	0x03C8	R/W
AHB_DMA_ARB_TIMEOUT_REG	权重仲裁超时配置寄存器	0x03DC	R/W
AHB_DMA_WEIGHT_EN_REG	权重仲裁使能寄存器	0x0400	R/W
AHB_DMA_MODULE_CLK_EN_REG	AHB DMA 内部各模块强制时钟使能寄存器	0x0404	R/W
版本寄存器			
AHB_DMA_DATE_REG	版本控制寄存器	0x0068	R/W
状态寄存器			
AHB_DMA_INFIFO_STATUS_CH0_REG	RX 通道 0 FIFO 状态	0x0078	RO
AHB_DMA_IN_STATE_CH0_REG	RX 通道 0 状态	0x0084	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX 通道 0 传输结束 (EOF) 时的接收链表描述符地址	0x0088	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX 通道 0 上发生错误时的接收链表描述符地址	0x008C	RO
AHB_DMA_IN_DSCR_CH0_REG	RX 通道 0 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0090	RO
AHB_DMA_IN_DSCR_BF0_CH0_REG	RX 通道 0 当前已预读取的接收链表描述符地址	0x0094	RO
AHB_DMA_IN_DSCR_BF1_CH0_REG	RX 通道 0 上一个已预读取的接收链表描述符地址	0x0098	RO
AHB_DMA_OUTFIFO_STATUS_CH0_REG	TX 通道 0 FIFO 状态	0x00D8	RO
AHB_DMA_OUT_STATE_CH0_REG	TX 通道 0 状态	0x00E4	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX 通道 0 传输结束 (EOF) 时的发送链表描述符地址	0x00E8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	TX 通道 0 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x00EC	RO

名称	描述	地址	访问
AHB_DMA_OUT_DSCR_CH0_REG	TX 通道 0 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x00F0	RO
AHB_DMA_OUT_DSCR_BF0_CH0_REG	TX 通道 0 当前已预读取的发送链表描述符地址	0x00F4	RO
AHB_DMA_OUT_DSCR_BF1_CH0_REG	TX 通道 0 上一个已预读取的发送链表描述符地址	0x00F8	RO
AHB_DMA_INFIFO_STATUS_CH1_REG	RX 通道 1 FIFO 状态	0x0138	RO
AHB_DMA_IN_STATE_CH1_REG	RX 通道 1 状态	0x0144	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX 通道 1 传输结束 (EOF) 时的接收链表描述符地址	0x0148	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX 通道 1 上发生错误时的接收链表描述符地址	0x014C	RO
AHB_DMA_IN_DSCR_CH1_REG	RX 通道 1 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0150	RO
AHB_DMA_IN_DSCR_BF0_CH1_REG	RX 通道 1 当前已预读取的接收链表描述符地址	0x0154	RO
AHB_DMA_IN_DSCR_BF1_CH1_REG	RX 通道 1 上一个已预读取的接收链表描述符地址	0x0158	RO
AHB_DMA_OUTFIFO_STATUS_CH1_REG	TX 通道 1 FIFO 状态	0x0198	RO
AHB_DMA_OUT_STATE_CH1_REG	TX 通道 1 状态	0x01A4	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的发送链表描述符地址	0x01A8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x01AC	RO
AHB_DMA_OUT_DSCR_CH1_REG	TX 通道 1 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x01B0	RO
AHB_DMA_OUT_DSCR_BF0_CH1_REG	TX 通道 1 当前已预读取的发送链表描述符地址	0x01B4	RO
AHB_DMA_OUT_DSCR_BF1_CH1_REG	TX 通道 1 上一个已预读取的发送链表描述符地址	0x01B8	RO
AHB_DMA_INFIFO_STATUS_CH2_REG	RX 通道 2 FIFO 状态	0x01F8	RO
AHB_DMA_IN_STATE_CH2_REG	RX 通道 2 状态	0x0204	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX 通道 2 传输结束 (EOF) 时的接收链表描述符地址	0x0208	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	RX 通道 2 上发生错误时的接收链表描述符地址	0x020C	RO
AHB_DMA_IN_DSCR_CH2_REG	RX 通道 2 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0210	RO
AHB_DMA_IN_DSCR_BF0_CH2_REG	RX 通道 2 当前已预读取的接收链表描述符地址	0x0214	RO
AHB_DMA_IN_DSCR_BF1_CH2_REG	RX 通道 2 上一个已预读取的接收链表描述符地址	0x0218	RO
AHB_DMA_OUTFIFO_STATUS_CH2_REG	TX 通道 2 FIFO 状态	0x0258	RO
AHB_DMA_OUT_STATE_CH2_REG	TX 通道 2 状态	0x0264	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH2_REG	TX 通道 2 传输结束 (EOF) 时的发送链表描述符地址	0x0268	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX 通道 2 传输结束 (EOF) 时接收链表描述符地址	0x0208	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	当 RX 通道 2 上发生错误时接收链表描述符地址	0x020C	RO

名称	描述	地址	访问
AHB_DMA_IN_DSCR_CH2_REG	当前预读取接收链表描述符指向的下一个接收链表描述符的地址在 RX 通道 2 上	0x0210	RO
AHB_DMA_IN_DSCR_BF0_CH2_REG	当前预读取接收链表描述符的地址在 RX 通道 2 上	0x0214	RO
AHB_DMA_IN_DSCR_BF1_CH2_REG	上一个已预读取的取接收链表描述符的地址在 RX 通道 2 上	0x0218	RO
AHB_DMA_OUTFIFO_STATUS_CH2_REG	TX 通道 2 的 FIFO 状态	0x0258	RO
AHB_DMA_OUT_STATE_CH2_REG	TX 通道 2 的状态	0x0264	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH2_REG	当 TX 通道 2 传输结束 (EOF) 时发送链表描述符地址	0x0268	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	TX 通道 2 传输结束 (EOF) 时最后一个发送链表描述符地址	0x026C	RO
AHB_DMA_OUT_DSCR_CH2_REG	TX 通道 2 当前预读取发送链表描述符指向的下一个发送链表描述符的地址	0x0270	RO
AHB_DMA_OUT_DSCR_BF0_CH2_REG	TX 通道 2 当前预读取发送链表描述符的地址	0x0274	RO
AHB_DMA_OUT_DSCR_BF1_CH2_REG	TX 通道 2 上一个已预读取的取发送链表描述符的地址在	0x0278	RO
AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG	AHB DMA 收到 AHB 总线错误响应对应传输的地址	0x0408	RO
AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG	AHB DMA 收到 AHB 总线错误响应对应传输的传输类型、ID 和通道 ID	0x040C	RO
AHB_DMA_IN_DONE_DES_ADDR_CH0_REG	记录 RX 通道 0 当前已完成描述符的地址	0x0410	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH0_REG	记录 TX 通道 0 当前已完成描述符的地址	0x0414	RO
AHB_DMA_IN_DONE_DES_ADDR_CH1_REG	记录 RX 通道 1 当前已完成描述符的地址	0x0418	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH1_REG	记录 TX 通道 1 当前已完成描述符的地址	0x041C	RO
AHB_DMA_IN_DONE_DES_ADDR_CH2_REG	记录 RX 通道 2 当前已完成描述符的地址	0x0420	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH2_REG	记录 TX 通道 2 当前已完成描述符的地址	0x0424	RO
优先级寄存器			
AHB_DMA_IN_PRI_CH0_REG	RX 通道 0 的优先级寄存器	0x009C	R/W
AHB_DMA_OUT_PRI_CH0_REG	TX 通道 0 的优先级寄存器	0x00FC	R/W
AHB_DMA_IN_PRI_CH1_REG	RX 通道 1 的优先级寄存器	0x015C	R/W
AHB_DMA_OUT_PRI_CH1_REG	TX 通道 1 的优先级寄存器	0x01BC	R/W
AHB_DMA_IN_PRI_CH2_REG	RX 通道 2 的优先级寄存器	0x021C	R/W
AHB_DMA_OUT_PRI_CH2_REG	TX 通道 2 的优先级寄存器	0x027C	R/W
外设选择寄存器			
AHB_DMA_IN_PERI_SEL_CH0_REG	RX 通道 0 的外设选择寄存器	0x00A0	R/W
AHB_DMA_OUT_PERI_SEL_CH0_REG	TX 通道 0 的外设选择寄存器	0x0100	R/W
AHB_DMA_IN_PERI_SEL_CH1_REG	RX 通道 1 的外设选择寄存器	0x0160	R/W

名称	描述	地址	访问
AHB_DMA_OUT_PERI_SEL_CH1_REG	TX 通道 1 的外设选择寄存器	0x01C0	R/W
AHB_DMA_IN_PERI_SEL_CH2_REG	RX 通道 2 的外设选择寄存器	0x0220	R/W
AHB_DMA_OUT_PERI_SEL_CH2_REG	TX 通道 2 的外设选择寄存器	0x0280	R/W

6.8.2 GDMA-AXI 寄存器列表

本小节的所有地址均为相对于 GDMA-AXI 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
AXI_DMA_IN_INT_RAW_CH0_REG	RX 通道 0 原始中断状态寄存器	0x0000	R/WTC/SS
AXI_DMA_IN_INT_ST_CH0_REG	RX 通道 0 屏蔽中断状态寄存器	0x0004	RO
AXI_DMA_IN_INT_ENA_CH0_REG	RX 通道 0 中断使能寄存器	0x0008	R/W
AXI_DMA_IN_INT_CLR_CH0_REG	RX 通道 0 中断清除寄存器	0x000C	WT
AXI_DMA_IN_INT_RAW_CH1_REG	RX 通道 1 原始中断状态寄存器	0x0068	R/WTC/SS
AXI_DMA_IN_INT_ST_CH1_REG	RX 通道 1 屏蔽中断状态寄存器	0x006C	RO
AXI_DMA_IN_INT_ENA_CH1_REG	RX 通道 1 中断使能寄存器	0x0070	R/W
AXI_DMA_IN_INT_CLR_CH1_REG	RX 通道 1 中断清除寄存器	0x0074	WT
AXI_DMA_IN_INT_RAW_CH2_REG	RX 通道 2 原始中断状态寄存器	0x00D0	R/WTC/SS
AXI_DMA_IN_INT_ST_CH2_REG	RX 通道 2 屏蔽中断状态寄存器	0x00D4	RO
AXI_DMA_IN_INT_ENA_CH2_REG	RX 通道 2 中断使能寄存器	0x00D8	R/W
AXI_DMA_IN_INT_CLR_CH2_REG	RX 通道 2 中断清除寄存器	0x00DC	WT
AXI_DMA_OUT_INT_RAW_CH0_REG	TX 通道 0 原始中断状态寄存器	0x0138	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH0_REG	TX 通道 0 屏蔽中断状态寄存器	0x013C	RO
AXI_DMA_OUT_INT_ENA_CH0_REG	TX 通道 0 中断使能寄存器	0x0140	R/W
AXI_DMA_OUT_INT_CLR_CH0_REG	TX 通道 0 中断清除寄存器	0x0144	WT
AXI_DMA_OUT_INT_RAW_CH1_REG	TX 通道 1 原始中断状态寄存器	0x01A0	R/WTC/SS

GoBack

名称	描述	地址	访问
AXI_DMA_OUT_INT_ST_CH1_REG	TX 通道 1 屏蔽中断状态寄存器	0x01A4	RO
AXI_DMA_OUT_INT_ENA_CH1_REG	TX 通道 1 中断使能寄存器	0x01A8	R/W
AXI_DMA_OUT_INT_CLR_CH1_REG	TX 通道 1 中断清除寄存器	0x01AC	WT
AXI_DMA_OUT_INT_RAW_CH2_REG	TX 通道 2 原始中断状态寄存器	0x0208	R/WTC/SS
AXI_DMA_OUT_INT_ST_CH2_REG	TX 通道 2 屏蔽中断状态寄存器	0x020C	RO
AXI_DMA_OUT_INT_ENA_CH2_REG	TX 通道 2 中断使能寄存器	0x0210	R/W
AXI_DMA_OUT_INT_CLR_CH2_REG	TX 通道 2 中断清除寄存器	0x0214	WT
配置寄存器			
AXI_DMA_IN_CONF0_CH0_REG	RX 通道 0 的配置寄存器 0	0x0010	R/W
AXI_DMA_IN_CONF1_CH0_REG	RX 通道 0 的配置寄存器 1	0x0014	R/W
AXI_DMA_IN_POP_CH0_REG	RX 通道 0 的弹出控制寄存器	0x001C	varies
AXI_DMA_IN_LINK1_CH0_REG	RX 通道 0 的链表描述符配置和控制寄存器 1	0x0020	varies
AXI_DMA_IN_LINK2_CH0_REG	RX 通道 0 的链表描述符配置和控制寄存器 2	0x0024	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH0_REG	RX 通道 0 CRC 初始值配置寄存器	0x0048	R/W
AXI_DMA_RX_CRC_WIDTH_CH0_REG	RX 通道 0 CRC 计算结果位宽配置寄存器	0x004C	R/W
AXI_DMA_IN_CRC_CLEAR_CH0_REG	RX 通道 0 CRC 计算结果清除寄存器	0x0050	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH0_REG	RX 通道 0 CRC 计算结果寄存器	0x0054	RO
AXI_DMA_RX_CRC_EN_ADDR_CH0_REG	TX 通道 0 CRC 中间结果掩码目标寄存器	0x005C	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH0_REG	CRC RX 通道 0 CRC 中间结果掩码寄存器	0x0060	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH0_REG	RX 通道 0 CRC 数据输入掩码目标寄存器	0x0064	R/W
AXI_DMA_IN_CONF0_CH1_REG	RX 通道 1 的配置寄存器 0	0x0078	R/W
AXI_DMA_IN_CONF1_CH1_REG	RX 通道 1 的配置寄存器 1	0x007C	R/W
AXI_DMA_IN_POP_CH1_REG	RX 通道 1 的弹出控制寄存器	0x0084	varies
AXI_DMA_IN_LINK1_CH1_REG	RX 通道 1 的链表描述符配置和控制寄存器 1	0x0088	varies
AXI_DMA_IN_LINK2_CH1_REG	RX 通道 1 的链表描述符配置和控制寄存器 2	0x008C	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH1_REG	RX 通道 1 CRC 初始值配置寄存器	0x00B0	R/W
AXI_DMA_RX_CRC_WIDTH_CH1_REG	RX 通道 1 CRC 计算结果位宽配置寄存器	0x00B4	R/W
AXI_DMA_IN_CRC_CLEAR_CH1_REG	RX 通道 1 CRC 计算结果清除寄存器	0x00B8	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH1_REG	RX 通道 1 CRC 计算结果寄存器	0x00BC	RO

名称	描述	地址	访问
AXI_DMA_RX_CRC_EN_ADDR_CH1_REG	RX 通道 1 CRC 中间结果掩码目标寄存器	0x00C4	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH1_REG	CRC RX 通道 1 CRC 中间结果掩码寄存器	0x00C8	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH1_REG	RX 通道 1 CRC 数据输入掩码目标寄存器	0x00CC	R/W
AXI_DMA_IN_CONF0_CH2_REG	RX 通道 2 的配置寄存器 0	0x00E0	R/W
AXI_DMA_IN_CONF1_CH2_REG	RX 通道 2 的配置寄存器 1	0x00E4	R/W
AXI_DMA_IN_POP_CH2_REG	RX 通道 2 的弹出控制寄存器	0x00EC	varies
AXI_DMA_IN_LINK1_CH2_REG	RX 通道 2 的链表描述符配置和控制寄存器 1	0x00F0	varies
AXI_DMA_IN_LINK2_CH2_REG	RX 通道 2 的链表描述符配置和控制寄存器 2	0x00F4	R/W
AXI_DMA_IN_CRC_INIT_DATA_CH2_REG	RX 通道 2 CRC 初始值配置寄存器	0x0118	R/W
AXI_DMA_RX_CRC_WIDTH_CH2_REG	RX 通道 2 CRC 计算结果位宽配置寄存器	0x011C	R/W
AXI_DMA_IN_CRC_CLEAR_CH2_REG	RX 通道 2 CRC 计算结果清除寄存器	0x0120	R/W
AXI_DMA_IN_CRC_FINAL_RESULT_CH2_REG	RX 通道 2 CRC 计算结果寄存器	0x0124	RO
AXI_DMA_RX_CRC_EN_ADDR_CH2_REG	RX 通道 2 CRC 中间结果掩码目标寄存器	0x012C	R/W
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH2_REG	CRC RX 通道 2 CRC 中间结果掩码寄存器	0x0130	R/W
AXI_DMA_RX_CRC_DATA_EN_ADDR_CH2_REG	RX 通道 2 CRC 数据输入掩码目标寄存器	0x0134	R/W
AXI_DMA_OUT_CONF0_CH0_REG	TX 通道 0 的配置寄存器 0	0x0148	R/W
AXI_DMA_OUT_CONF1_CH0_REG	TX 通道 0 的配置寄存器 1	0x014C	R/W
AXI_DMA_OUT_PUSH_CH0_REG	TX 通道 0 的推入控制寄存器	0x0154	varies
AXI_DMA_OUT_LINK1_CH0_REG	TX 通道 0 的链表描述符配置和控制寄存器 1	0x0158	varies
AXI_DMA_OUT_LINK2_CH0_REG	TX 通道 0 的链表描述符配置和控制寄存器 2	0x015C	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH0_REG	TX 通道 0 CRC 初始值配置寄存器	0x0180	R/W
AXI_DMA_TX_CRC_WIDTH_CH0_REG	TX 通道 0 CRC 计算结果位宽配置寄存器	0x0184	R/W
AXI_DMA_OUT_CRC_CLEAR_CH0_REG	TX 通道 0 CRC 计算结果清除寄存器	0x0188	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH0_REG	TX 通道 0 CRC 计算结果寄存器	0x018C	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH0_REG	CRC TX 通道 0 CRC 中间结果掩码寄存器	0x0190	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH0_REG	TX 通道 0 CRC 中间结果掩码目标寄存器	0x0194	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH0_REG	TX 通道 0 CRC 数据输入掩码寄存器	0x0198	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH0_REG	TX 通道 0 CRC 数据输入掩码目标寄存器	0x019C	R/W
AXI_DMA_OUT_CONF0_CH1_REG	TX 通道 1 的配置寄存器 0	0x01B0	R/W

名称	描述	地址	访问
AXI_DMA_OUT_CONF1_CH1_REG	TX 通道 1 的配置寄存器 1	0x01B4	R/W
AXI_DMA_OUT_PUSH_CH1_REG	TX 通道 1 的推入控制寄存器	0x01BC	varies
AXI_DMA_OUT_LINK1_CH1_REG	TX 通道 1 的链表描述符配置和控制寄存器 1	0x01C0	varies
AXI_DMA_OUT_LINK2_CH1_REG	TX 通道 1 的链表描述符配置和控制寄存器 2	0x01C4	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH1_REG	TX 通道 1 CRC 初始值配置寄存器	0x01E8	R/W
AXI_DMA_TX_CRC_WIDTH_CH1_REG	TX 通道 1 CRC 计算结果位宽配置寄存器	0x01EC	R/W
AXI_DMA_OUT_CRC_CLEAR_CH1_REG	TX 通道 1 CRC 计算结果清除寄存器	0x01F0	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH1_REG	TX 通道 1 CRC 计算结果寄存器	0x01F4	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH1_REG	CRC TX 通道 1 CRC 中间结果掩码寄存器	0x01F8	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH1_REG	TX 通道 1 CRC 中间结果掩码目标寄存器	0x01FC	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH1_REG	TX 通道 1 CRC 数据输入掩码目标寄存器	0x0200	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH1_REG	TX 通道 1 CRC 数据输入掩码目标寄存器	0x0204	R/W
AXI_DMA_OUT_CONF0_CH2_REG	TX 通道 2 的配置寄存器 0	0x0218	R/W
AXI_DMA_OUT_CONF1_CH2_REG	TX 通道 2 的配置寄存器 1	0x021C	R/W
AXI_DMA_OUT_PUSH_CH2_REG	TX 通道 2 的推入控制寄存器	0x0224	varies
AXI_DMA_OUT_LINK1_CH2_REG	TX 通道 2 的链表描述符配置和控制寄存器	0x0228	varies
AXI_DMA_OUT_LINK2_CH2_REG	TX 通道 2 的链表描述符配置和控制寄存器 2 1	0x022C	R/W
AXI_DMA_OUT_CRC_INIT_DATA_CH2_REG	TX 通道 2 CRC 初始值配置寄存器	0x0250	R/W
AXI_DMA_TX_CRC_WIDTH_CH2_REG	TX 通道 2 CRC 计算结果位宽配置寄存器	0x0254	R/W
AXI_DMA_OUT_CRC_CLEAR_CH2_REG	TX 通道 2 CRC 计算结果清除寄存器	0x0258	R/W
AXI_DMA_OUT_CRC_FINAL_RESULT_CH2_REG	TX 通道 2 CRC 计算结果寄存器	0x025C	RO
AXI_DMA_TX_CRC_EN_WR_DATA_CH2_REG	CRC TX 通道 2 CRC 中间结果掩码寄存器	0x0260	R/W
AXI_DMA_TX_CRC_EN_ADDR_CH2_REG	TX 通道 2 CRC 中间结果掩码目标寄存器	0x0264	R/W
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH2_REG	TX 通道 2 CRC 数据输入掩码目标寄存器	0x0268	R/W
AXI_DMA_TX_CRC_DATA_EN_ADDR_CH2_REG	TX 通道 2 CRC 数据输入掩码目标寄存器	0x026C	R/W
AXI_DMA_ARB_TIMEOUT_REG	仲裁超时配置寄存器	0x0270	R/W
AXI_DMA_WEIGHT_EN_REG	权重仲裁使能寄存器	0x0274	R/W
AXI_DMA_IN_MEM_CONF_REG	RX 通道内部存储器电源配置寄存器	0x0278	R/W
AXI_DMA_INTR_MEM_START_ADDR_REG	可访问的内部存储器起始地址配置寄存器	0x027C	R/W

名称	描述	地址	访问
AXI_DMA_INTR_MEM_END_ADDR_REG	可访问的内部存储器结束地址配置寄存器	0x0280	R/W
AXI_DMA_EXTR_MEM_START_ADDR_REG	可访问的外部存储器起始地址配置寄存器	0x0284	R/W
AXI_DMA_EXTR_MEM_END_ADDR_REG	可访问的外部存储器结束地址配置寄存器	0x0288	R/W
AXI_DMA_MISC_CONF_REG	杂项寄存器	0x02A8	R/W
AXI_DMA_LINK_SWITCH_STATE_REG	链表切换配置寄存器	0x02DC	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH0_REG	CRC RX 通道 0 CRC 中间结果掩码寄存器	0x0058	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH1_REG	CRC RX 通道 1 CRC 中间结果掩码寄存器	0x00C0	R/W
AXI_DMA_RX_CRC_EN_WR_DATA_CH2_REG	CRC RX 通道 2 CRC 中间结果掩码寄存器	0x0128	R/W
状态寄存器			
AXI_DMA_INFIFO_STATUS_CH0_REG	RX 通道 0 FIFO 状态寄存器	0x0018	RO
AXI_DMA_IN_STATE_CH0_REG	RX 通道 0 状态寄存器	0x0028	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX 通道 0 传输结束 (EOF) 时的接收链表描述符地址	0x002C	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX 通道 0 上发生错误时的接收链表描述符地址	0x0030	RO
AXI_DMA_IN_DSCR_CH0_REG	RX 通道 0 预读取的接收链表描述符指向的下一个接收链表描述符的地址	0x0034	RO
AXI_DMA_IN_DSCR_BF0_CH0_REG	RX 通道 0 当前预读取的接收链表描述符的地址	0x0038	RO
AXI_DMA_IN_DSCR_BF1_CH0_REG	RX 通道 0 上一个预读取的接收链表描述符的地址	0x003C	RO
AXI_DMA_INFIFO_STATUS_CH1_REG	RX 通道 1 FIFO 状态寄存器	0x0080	RO
AXI_DMA_IN_STATE_CH1_REG	RX 通道 1 状态寄存器	0x0090	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX 通道 1 传输结束 (EOF) 时的接收链表描述符地址	0x0094	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX 通道 1 上发生错误时的接收链表描述符地址	0x0098	RO
AXI_DMA_IN_DSCR_CH1_REG	RX 通道 1 预读取的接收链表描述符指向的下一个接收链表描述符的地址	0x009C	RO
AXI_DMA_IN_DSCR_BF0_CH1_REG	RX 通道 1 当前预读取的接收链表描述符的地址	0x00A0	RO
AXI_DMA_IN_DSCR_BF1_CH1_REG	RX 通道 1 上一个预读取的接收链表描述符的地址	0x00A4	RO
AXI_DMA_INFIFO_STATUS_CH2_REG	RX 通道 2 FIFO 状态寄存器	0x00E8	RO
AXI_DMA_IN_STATE_CH2_REG	RX 通道 2 状态寄存器	0x00F8	RO
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX 通道 2 传输结束 (EOF) 时的接收链表描述符地址	0x00FC	RO
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	RX 通道 2 上发生错误时的接收链表描述符地址	0x0100	RO
AXI_DMA_IN_DSCR_CH2_REG	RX 通道 2 预读取的接收链表描述符指向的下一个接收链表描述符的地址	0x0104	RO
AXI_DMA_IN_DSCR_BF0_CH2_REG	RX 通道 2 当前预读取的接收链表描述符的地址	0x0108	RO

名称	描述	地址	访问
AXI_DMA_IN_DSCR_BF1_CH2_REG	RX 通道 2 上一个预读取的接收链表描述符的地址	0x010C	RO
AXI_DMA_OUTFIFO_STATUS_CH0_REG	TX 通道 0 FIFO 状态	0x0150	RO
AXI_DMA_OUT_STATE_CH0_REG	TX 通道 0 状态	0x0160	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX 通道 0 传输结束 (EOF) 时的发送链表描述符地址	0x0164	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	TX 通道 0 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x0168	RO
AXI_DMA_OUT_DSCR_CH0_REG	TX 通道 0 预读取的发送链表描述符指向的下一个发送链表描述符的地址	0x016C	RO
AXI_DMA_OUT_DSCR_BF0_CH0_REG	TX 通道 0 当前预读取的发送链表描述符的地址	0x0170	RO
AXI_DMA_OUT_DSCR_BF1_CH0_REG	TX 通道 0 上一个预读取的发送链表描述符的地址	0x0174	RO
AXI_DMA_OUTFIFO_STATUS_CH1_REG	TX 通道 0 FIFO 状态	0x01B8	RO
AXI_DMA_OUT_STATE_CH1_REG	TX 通道 1 状态	0x01C8	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的发送链表描述符地址	0x01CC	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x01D0	RO
AXI_DMA_OUT_DSCR_CH1_REG	TX 通道 1 预读取的发送链表描述符指向的下一个发送链表描述符的地址	0x01D4	RO
AXI_DMA_OUT_DSCR_BF0_CH1_REG	TX 通道 1 当前预读取的发送链表描述符的地址	0x01D8	RO
AXI_DMA_OUT_DSCR_BF1_CH1_REG	TX 通道 1 上一个预读取的发送链表描述符的地址	0x01DC	RO
AXI_DMA_OUTFIFO_STATUS_CH2_REG	TX 通道 2 FIFO 状态	0x0220	RO
AXI_DMA_OUT_STATE_CH2_REG	TX 通道 2 状态	0x0230	RO
AXI_DMA_OUT_EOF_DES_ADDR_CH2_REG	TX 通道 2 传输结束 (EOF) 时的发送链表描述符地址	0x0234	RO
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	TX 通道 2 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x0238	RO
AXI_DMA_OUT_DSCR_CH2_REG	TX 通道 2 预读取的发送链表描述符指向的下一个发送链表描述符的地址	0x023C	RO
AXI_DMA_OUT_DSCR_BF0_CH2_REG	TX 通道 2 当前预读取的发送链表描述符的地址	0x0240	RO
AXI_DMA_OUT_DSCR_BF1_CH2_REG	TX 通道 2 上一个预读取的发送链表描述符的地址	0x0244	RO
AXI_DMA_IN_RESET_AVAIL_CH0_REG	RX 通道 0 复位状态寄存器	0x028C	RO
AXI_DMA_IN_RESET_AVAIL_CH1_REG	RX 通道 1 复位状态寄存器	0x0290	RO
AXI_DMA_IN_RESET_AVAIL_CH2_REG	RX 通道 2 复位状态寄存器	0x0294	RO
AXI_DMA_OUT_RESET_AVAIL_CH0_REG	TX 通道 0 复位状态寄存器	0x0298	RO
AXI_DMA_OUT_RESET_AVAIL_CH1_REG	TX 通道 1 复位状态寄存器	0x029C	RO
AXI_DMA_OUT_RESET_AVAIL_CH2_REG	TX 通道 2 复位状态寄存器	0x02A0	RO
优先级寄存器			

名称	描述	地址	访问
AXI_DMA_IN_PRI_CH0_REG	RX 通道 0 的优先级寄存器	0x0040	R/W
AXI_DMA_IN_PRI_CH1_REG	RX 通道 1 的优先级寄存器	0x00A8	R/W
AXI_DMA_IN_PRI_CH2_REG	RX 通道 2 的优先级寄存器	0x0110	R/W
AXI_DMA_OUT_PRI_CH0_REG	TX 通道 0 的优先级寄存器	0x0178	R/W
AXI_DMA_OUT_PRI_CH1_REG	TX 通道 1 的优先级寄存器	0x01E0	R/W
AXI_DMA_OUT_PRI_CH2_REG	TX 通道 2 的优先级寄存器	0x0248	R/W
外设选择寄存器			
AXI_DMA_IN_PERI_SEL_CH0_REG	RX 通道 0 的外设选择寄存器	0x0044	R/W
AXI_DMA_IN_PERI_SEL_CH1_REG	RX 通道 1 的外设选择寄存器	0x00AC	R/W
AXI_DMA_IN_PERI_SEL_CH2_REG	RX 通道 2 的外设选择寄存器	0x0114	R/W
AXI_DMA_OUT_PERI_SEL_CH0_REG	TX 通道 0 的外设选择寄存器	0x017C	R/W
AXI_DMA_OUT_PERI_SEL_CH1_REG	TX 通道 1 的外设选择寄存器	0x01E4	R/W
AXI_DMA_OUT_PERI_SEL_CH2_REG	TX 通道 2 的外设选择寄存器	0x024C	R/W
版本寄存器			
AXI_DMA_DATE_REG	版本控制寄存器	0x02D8	R/W

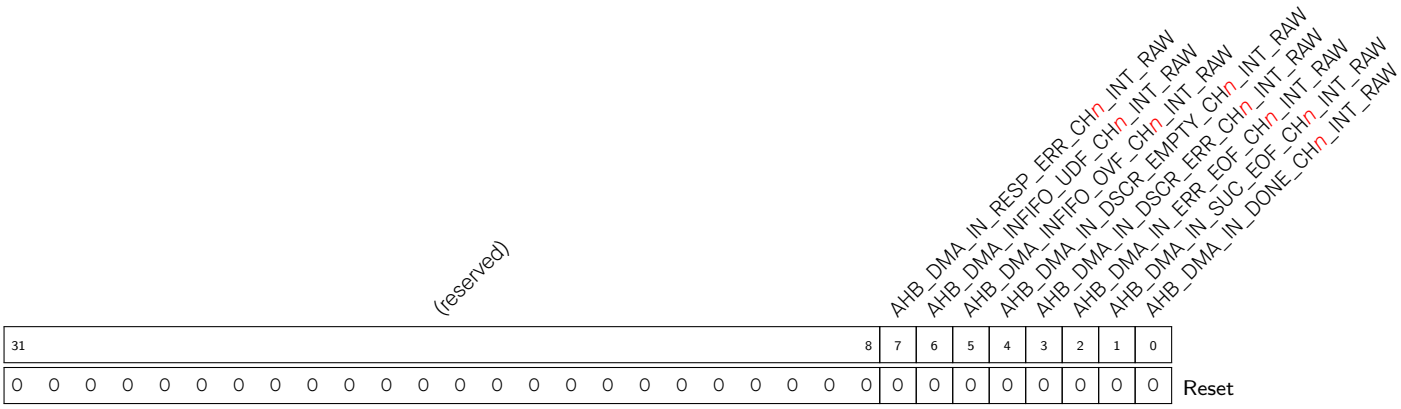
6.9 寄存器

6.9.1 GDMA-AHB 寄存器

本小节的所有地址均为相对于 GDMA-AHB 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 6.1. AHB_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x10**n*)



AHB_DMA_IN_DONE_CH*n*_INT_RAW AHB_DMA_IN_DONE_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_RAW AHB_DMA_IN_SUC_EOF_CH*n*_INT 的原始中断状态。对于 UHCI，接收链表描述符指向的最后一个字节数据接收成功、且接收通道 *n* 没有检测到数据错误时，该位变为 1。(R/WTC/SS)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_RAW AHB_DMA_IN_ERR_EOF_CH*n*_INT 的原始中断状态。仅对 UHCI 有效。(R/WTC/SS)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_RAW AHB_DMA_IN_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

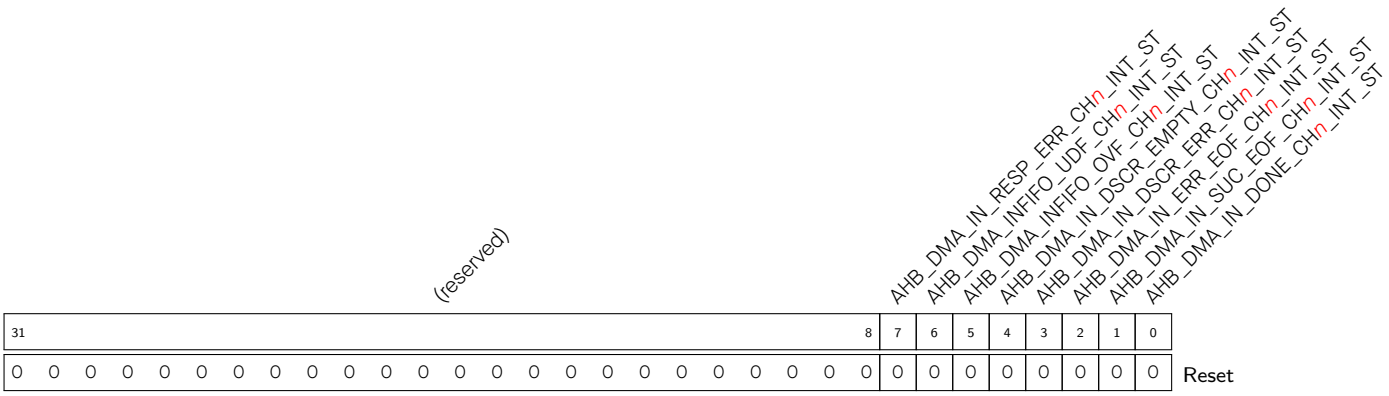
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_RAW AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_INFIFO_OVF_CH*n*_INT_RAW AHB_DMA_INFIFO_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_INFIFO_UDF_CH*n*_INT_RAW AHB_DMA_INFIFO_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_RAW AHB_DMA_IN_RESP_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 6.2. AHB_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x10***n*)



AHB_DMA_IN_DONE_CH*n*_INT_ST AHB_DMA_IN_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_ST AHB_DMA_IN_SUC_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_ST AHB_DMA_IN_ERR_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ST AHB_DMA_IN_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

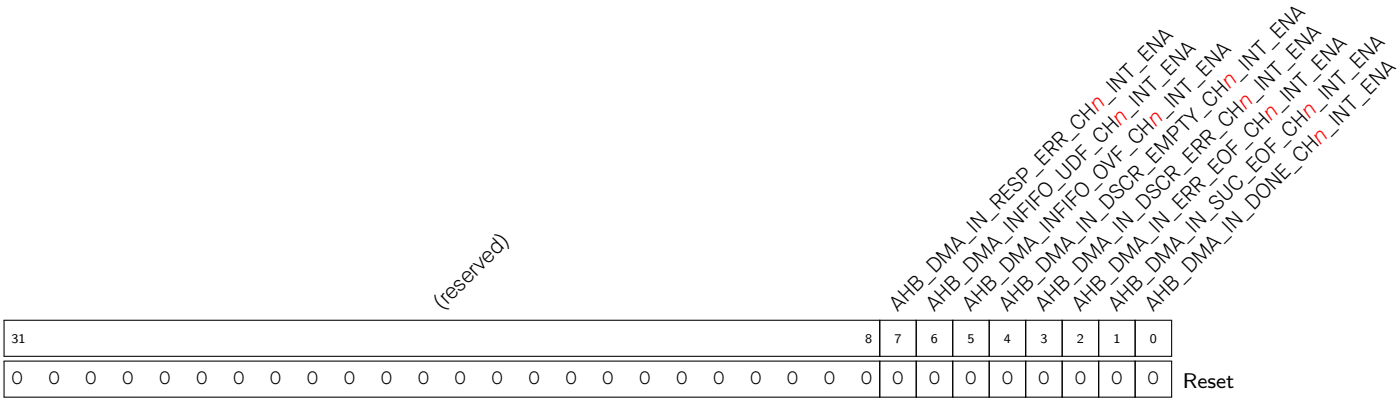
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ST AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_INFIFO_OVF_CH*n*_INT_ST AHB_DMA_INFIFO_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_INFIFO_UDF_CH*n*_INT_ST AHB_DMA_INFIFO_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_ST AHB_DMA_IN_RESP_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

Register 6.3. AHB_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x10*n*)



AHB_DMA_IN_DONE_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_DONE_CH*n*_INT。(R/W)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_SUC_EOF_CH*n*_INT。(R/W)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_ERR_EOF_CH*n*_INT。(R/W)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_DSCR_ERR_CH*n*_INT。(R/W)

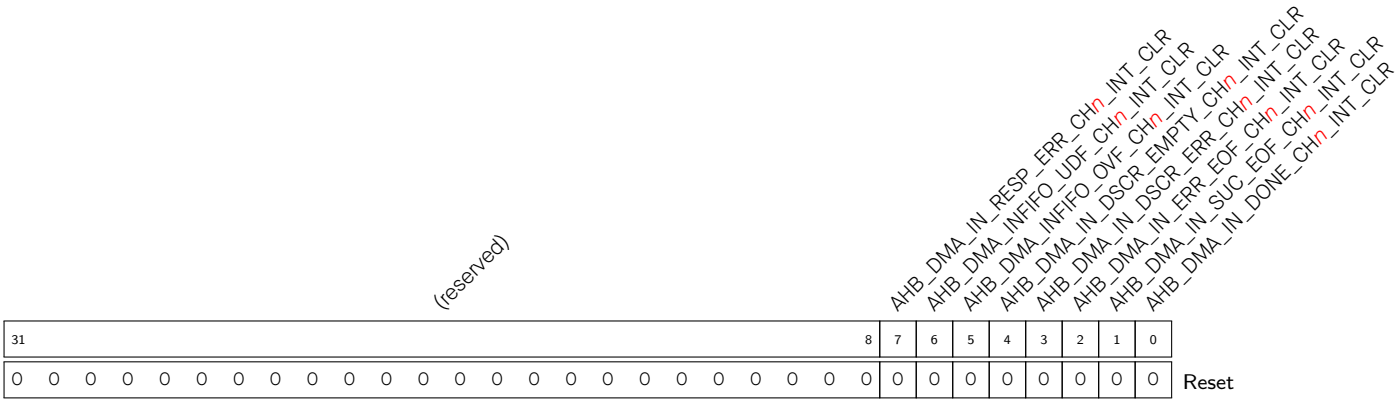
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT。(R/W)

AHB_DMA_INFIFO_OVF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_INFIFO_OVF_CH*n*_INT。(R/W)

AHB_DMA_INFIFO_UDF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_INFIFO_UDF_CH*n*_INT。(R/W)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_RESP_ERR_CH*n*_INT。(R/W)

Register 6.4. AHB_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x10**n*)



AHB_DMA_IN_DONE_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_DONE_CH*n*_INT。(WT)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_SUC_EOF_CH*n*_INT。(WT)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_ERR_EOF_CH*n*_INT。(WT)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_DSCR_ERR_CH*n*_INT。(WT)

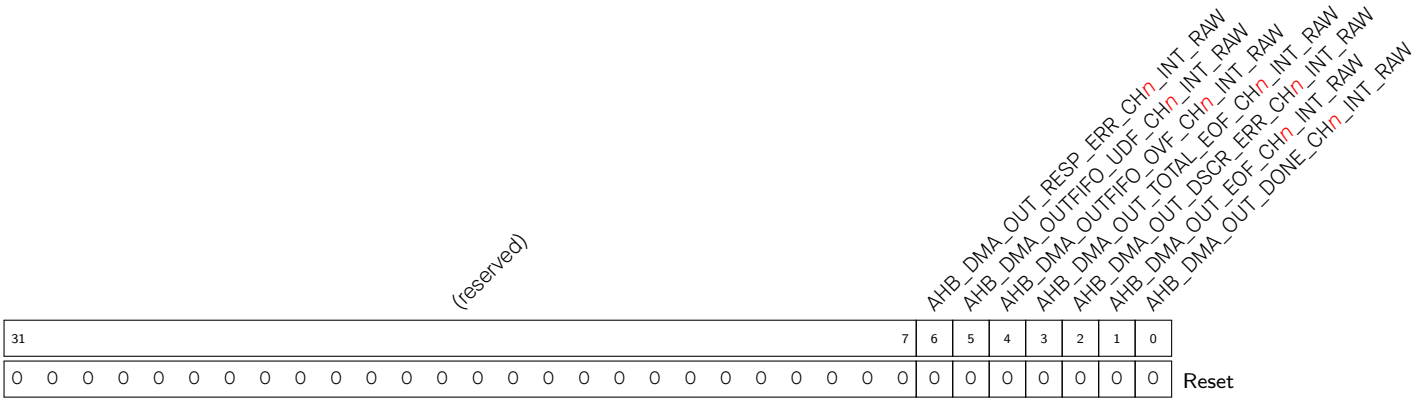
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT。(WT)

AHB_DMA_INFIFO_OVF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_INFIFO_OVF_CH*n*_INT。(WT)

AHB_DMA_INFIFO_UDF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_INFIFO_UDF_CH*n*_INT。(WT)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_RESP_ERR_CH*n*_INT。(WT)

Register 6.5. AHB_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0030+0x10*n*)



AHB_DMA_OUT_DONE_CH*n*_INT_RAW AHB_DMA_OUT_DONE_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_EOF_CH*n*_INT_RAW AHB_DMA_OUT_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_RAW AHB_DMA_OUT_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

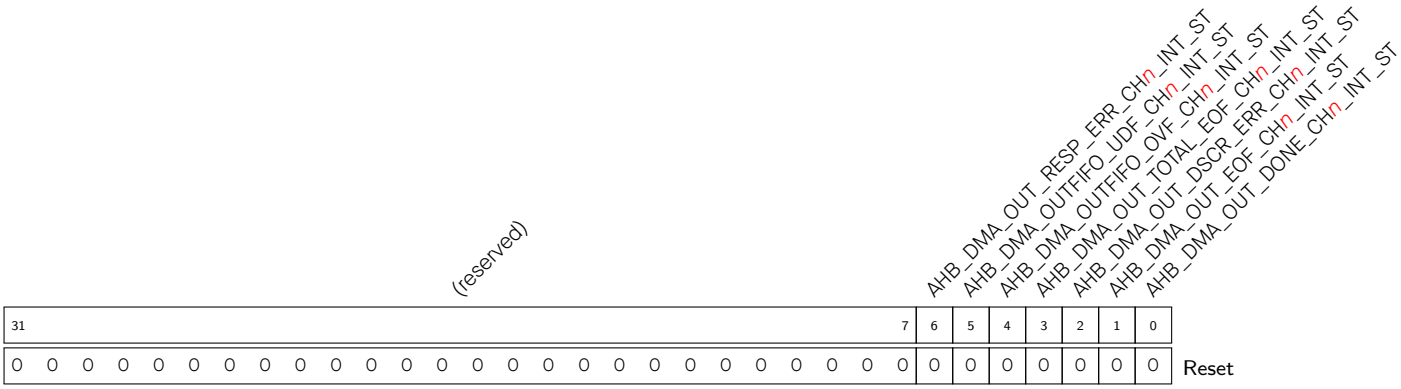
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_RAW AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_RAW AHB_DMA_OUTFIFO_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_RAW AHB_DMA_OUTFIFO_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_RAW AHB_DMA_OUT_RESP_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 6.6. AHB_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-2) (0x0034+0x10*n*)



AHB_DMA_OUT_DONE_CH*n*_INT_ST AHB_DMA_OUT_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_EOF_CH*n*_INT_ST AHB_DMA_OUT_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ST AHB_DMA_OUT_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

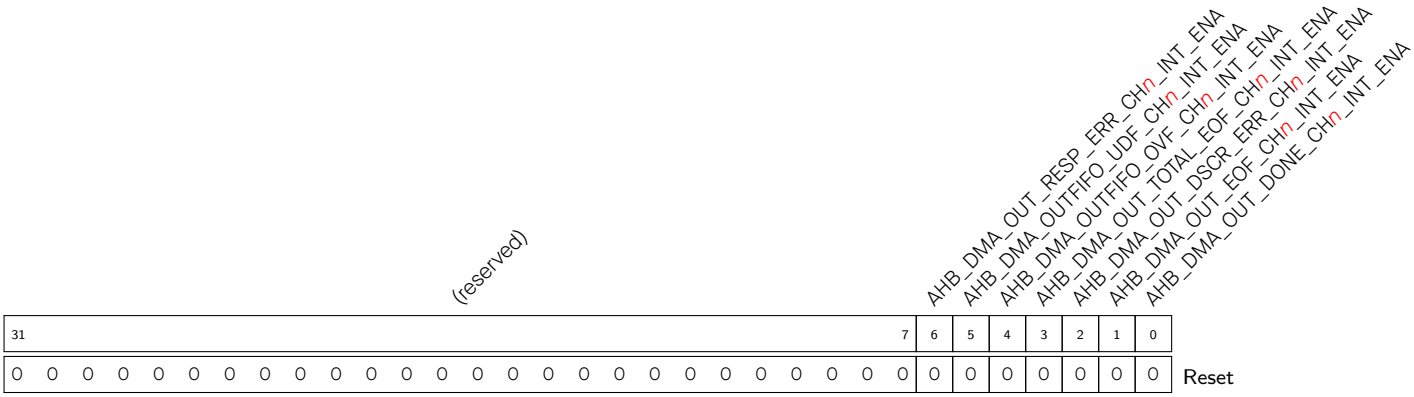
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ST AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_ST AHB_DMA_OUTFIFO_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_ST AHB_DMA_OUTFIFO_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_ST AHB_DMA_OUT_RESP_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

Register 6.7. AHB_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0038+0x10**n*)



AHB_DMA_OUT_DONE_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_DONE_CH*n*_INT。(R/W)

AHB_DMA_OUT_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_EOF_CH*n*_INT。(R/W)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_DSCR_ERR_CH*n*_INT。(R/W)

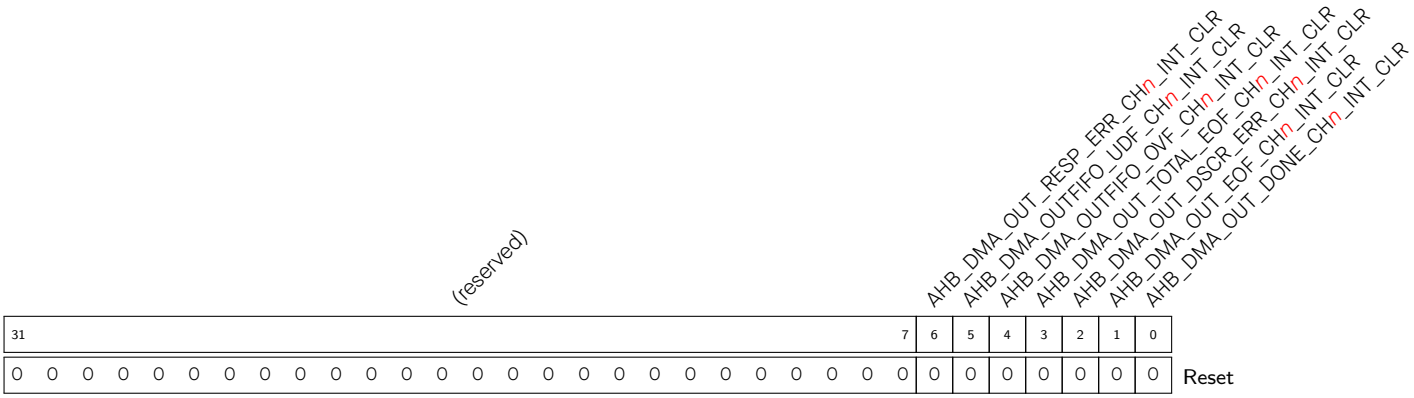
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT。(R/W)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUTFIFO_OVF_CH*n*_INT。(R/W)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUTFIFO_UDF_CH*n*_INT。(R/W)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_RESP_ERR_CH*n*_INT。(R/W)

Register 6.8. AHB_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x003C+0x10**n*)



AHB_DMA_OUT_DONE_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_DONE_CH*n*_INT。(WT)

AHB_DMA_OUT_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_EOF_CH*n*_INT。(WT)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_DSCR_ERR_CH*n*_INT。(WT)

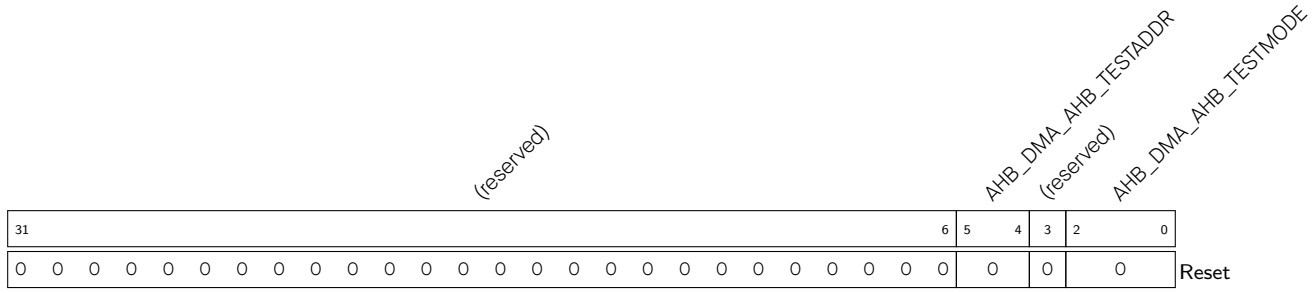
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT。(WT)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUTFIFO_OVF_CH*n*_INT。(WT)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUTFIFO_UDF_CH*n*_INT。(WT)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_RESP_ERR_CH*n*_INT。(WT)

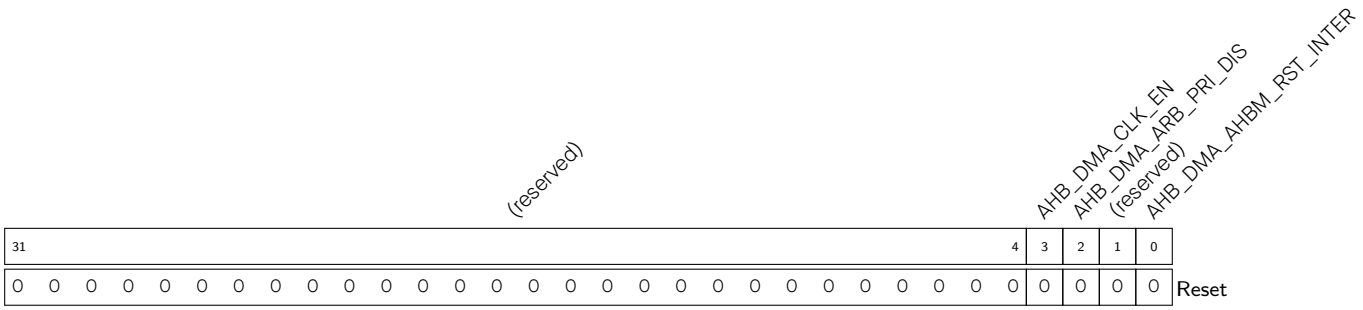
Register 6.9. AHB_DMA_AHB_TEST_REG (0x0060)



AHB_DMA_AHB_TESTMODE 保留。(R/W)

AHB_DMA_AHB_TESTADDR 保留。(R/W)

Register 6.10. AHB_DMA_MISC_CONF_REG (0x0064)



AHB_DMA_AHBM_RST_INTER 先写 1 再写 0 复位内部 AHB FSM。(R/W)

AHB_DMA_ARB_PRI_DIS 配置关闭优先级仲裁。

0：开启

1：关闭

(R/W)

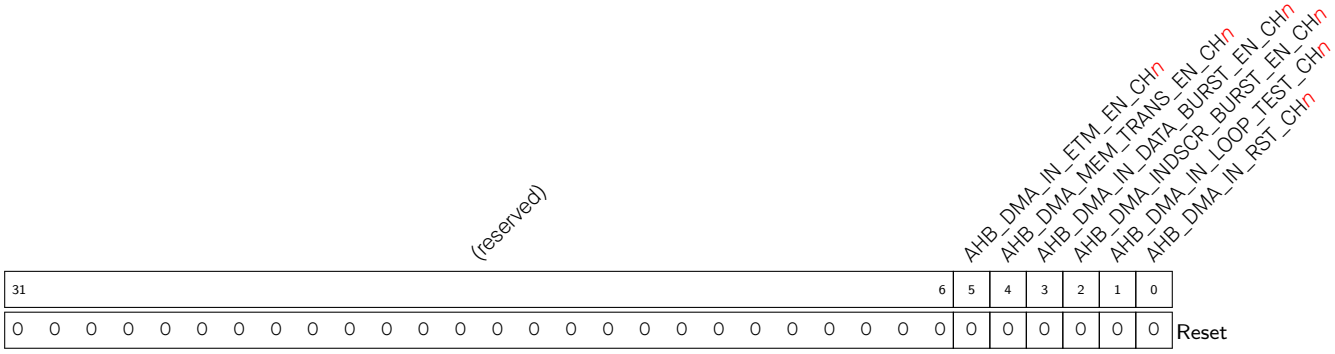
AHB_DMA_CLK_EN 配置 AHB DMA 时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 6.11. AHB_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0070+0xC0**n*)



AHB_DMA_IN_RST_CH*n* 先写 1 再写 0 复位接收通道 *n* 的 FSM 和 RX FIFO 指针。(R/W)

AHB_DMA_IN_LOOP_TEST_CH*n* 保留。(R/W)

AHB_DMA_INDSCR_BURST_EN_CH*n* 配置开启 INCR 突发传输让接收通道 *n* 读取描述符。

0：关闭

1：开启

(R/W)

AHB_DMA_IN_DATA_BURST_EN_CH*n* 配置开启接收通道 *n* 突发传输。

0：关闭

1：开启

(R/W)

AHB_DMA_MEM_TRANS_EN_CH*n* (*n* = 1 ~2) 配置开启存储器到存储器数据传输。通道 0 不支持此功能。

0：关闭

1：开启

(R/W)

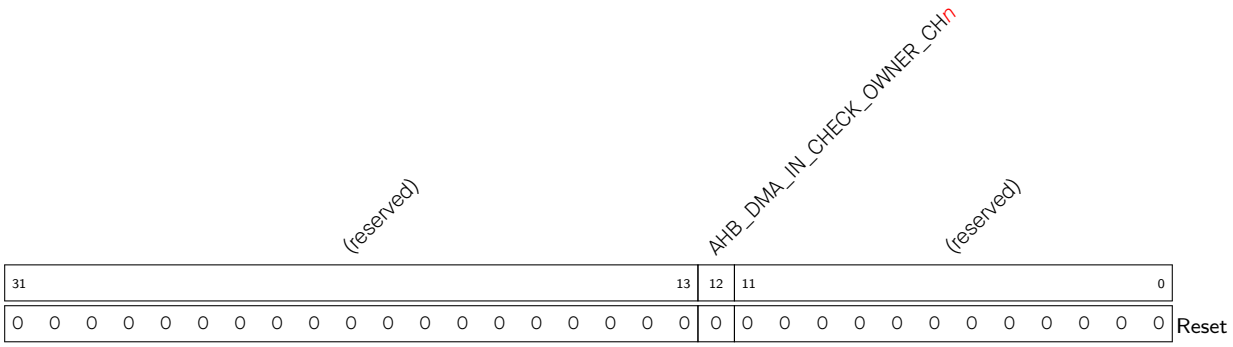
AHB_DMA_IN_ETM_EN_CH*n* 配置开启接收通道 *n* 的 ETM 控制。

0：关闭

1：开启

(R/W)

Register 6.12. AHB_DMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0074+0xC0*n*)



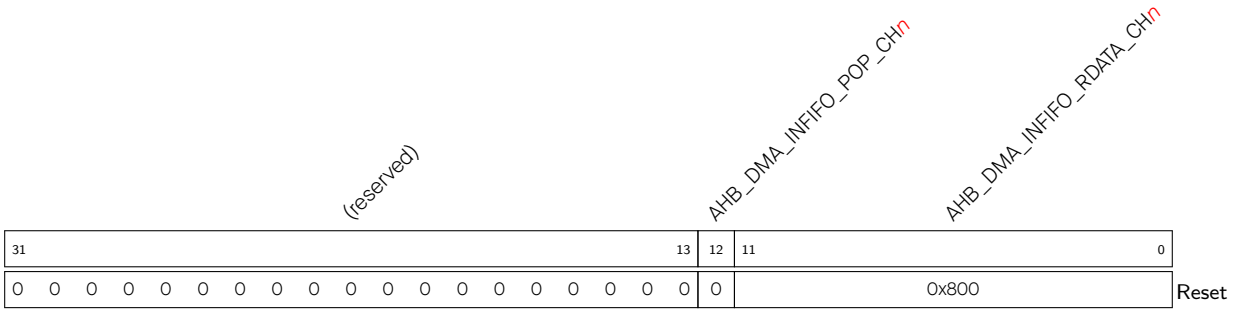
AHB_DMA_IN_CHECK_OWNER_CH*n* 配置开启接收通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

Register 6.13. AHB_DMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x007C+0xC0*n*)



AHB_DMA_INFIFO_RDATA_CH*n* 表示从 AHB DMA RX FIFO 弹出的数据。(RO)

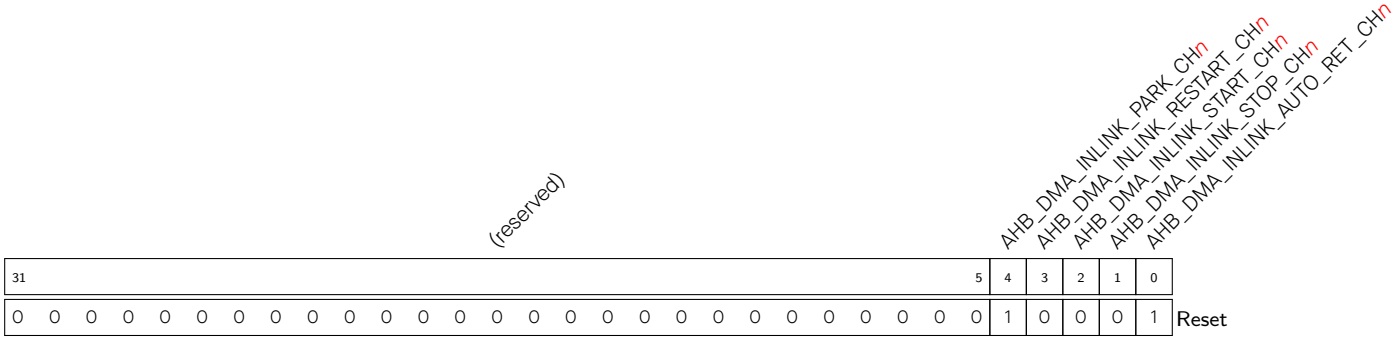
AHB_DMA_INFIFO_POP_CH*n* 配置从 AHB DMA RX FIFO 中弹出数据。

0：无效值。没有作用

1：弹出

(WT)

Register 6.14. AHB_DMA_IN_LINK_CH*n*_REG (*n*: 0-2) (0x0080+0xC0*n*)



AHB_DMA_INLINK_AUTO_RET_CH*n* 配置在接收数据有错时返回到当前接收链表描述符的地址。

0：不返回

1：返回

(R/W)

AHB_DMA_INLINK_STOP_CH*n* 配置接收通道 *n* 停止接收数据。

0：无效值。没有作用

1：停止

(WT)

AHB_DMA_INLINK_START_CH*n* 配置开启接收通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

AHB_DMA_INLINK_RESTART_CH*n* 配置重启接收通道 *n* 进行 AHB DMA 传输。

0：无效值。没有作用

1：重启

(WT)

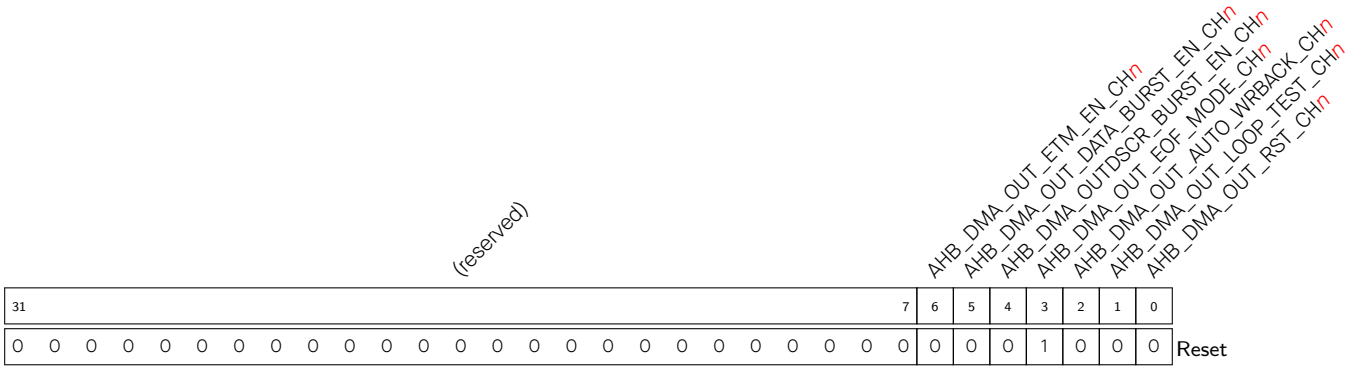
AHB_DMA_INLINK_PARK_CH*n* 表示接收链表描述符 FSM 的状态。

0：运行

1：空闲

(RO)

Register 6.15. AHB_DMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x00D0+0xC0**n*)



AHB_DMA_OUT_RST_CH*n* 配置发送通道 *n* FSM 和 TX FIFO 指针的复位状态。

0：复位释放

1：复位

(R/W)

AHB_DMA_OUT_LOOP_TEST_CH*n* 保留。(R/W)

AHB_DMA_OUT_AUTO_WRBACK_CH*n* 配置在发送完 TX FIFO 中的所有数据时是否开启发送链表自动回写。

0：关闭

1：开启

(R/W)

AHB_DMA_OUT_EOF_MODE_CH*n* 配置生成 EOF 标志的时间。

0：待发送数据推入 AHB DMA FIFO 时生成发送通道 *n* 的 EOF 标志。

1：待发送数据从 AHB DMA FIFO 中弹出时生成发送通道 *n* 的 EOF 标志。

(R/W)

AHB_DMA_OUTDSCR_BURST_EN_CH*n* 配置开启 INCR 突发传输，让发送通道 *n* 读取描述符。

0：关闭

1：开启

(R/W)

AHB_DMA_OUT_DATA_BURST_EN_CH*n* 配置开启发送通道 *n* INCR 突发传输。

0：关闭

1：开启

(R/W)

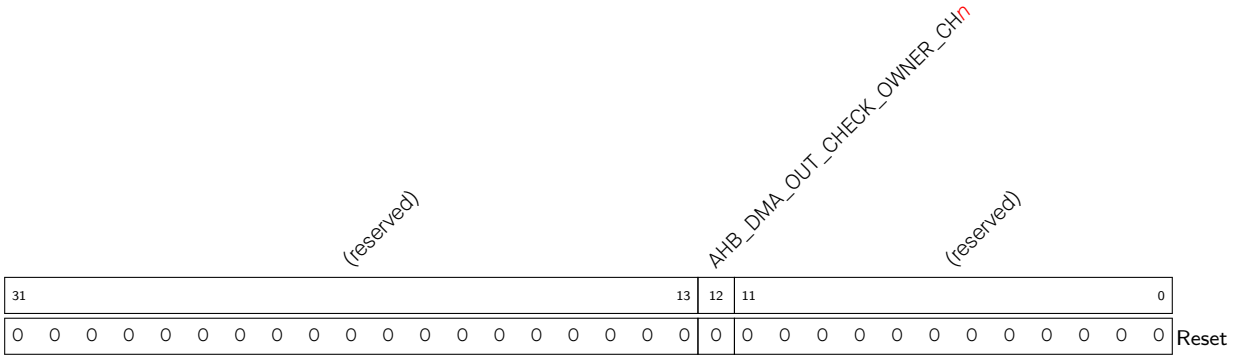
AHB_DMA_OUT_ETM_EN_CH*n* 配置开启发送通道 *n* 的 ETM 控制。

0：关闭

1：开启

(R/W)

Register 6.16. AHB_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x00D4+0xC0*n*)



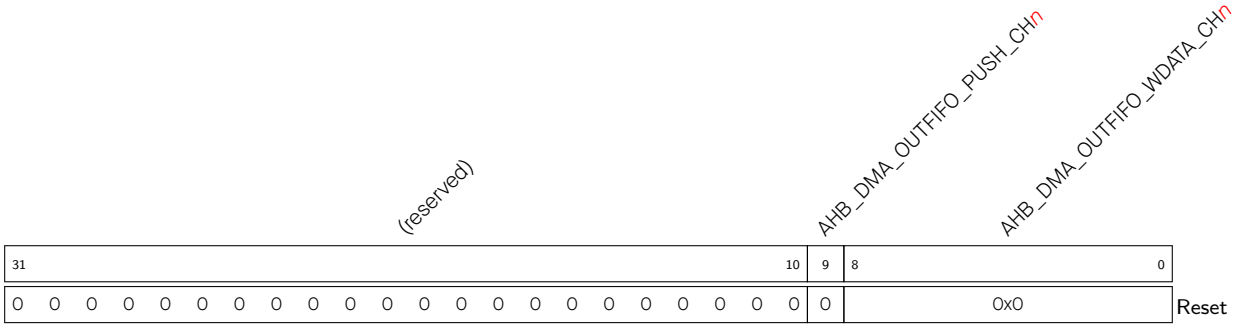
AHB_DMA_OUT_CHECK_OWNER_CH*n* 配置开启发送通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

Register 6.17. AHB_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x00DC+0xC0*n*)



AHB_DMA_OUTFIFO_WDATA_CH*n* 表示需要推入 AHB DMA TX FIFO 中的数据。(R/W)

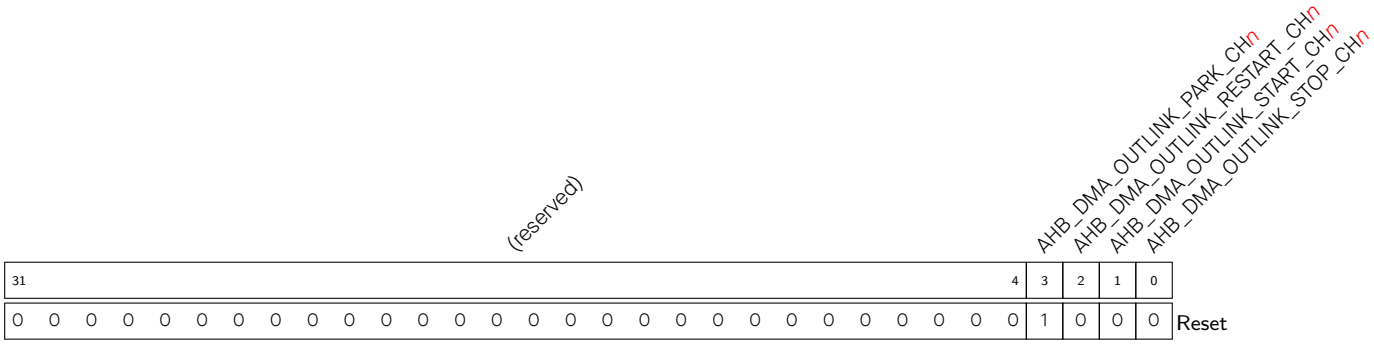
AHB_DMA_OUTFIFO_PUSH_CH*n* 配置将数据推入 AHB DMA TX FIFO。

0：无效值。没有作用

1：推入

(WT)

Register 6.18. AHB_DMA_OUT_LINK_CH*n*_REG (*n*: 0-2) (0x00E0+0xC0**n*)



AHB_DMA_OUTLINK_STOP_CH*n* 配置发送通道 *n* 停止发送数据。

0：无效值。没有作用

1：停止

(WT)

AHB_DMA_OUTLINK_START_CH*n* 配置开启发送通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

AHB_DMA_OUTLINK_RESTART_CH*n* 配置重启发送通道 *n* 进行 AHB DMA 传输。

0：无效值。没有作用

1：重启

(WT)

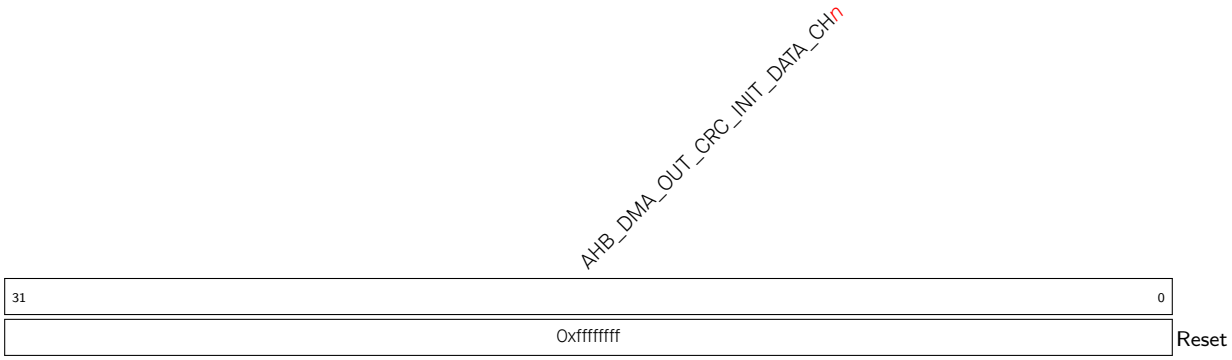
AHB_DMA_OUTLINK_PARK_CH*n* 表示发送链表描述符 FSM 的状态。

0：运行

1：空闲

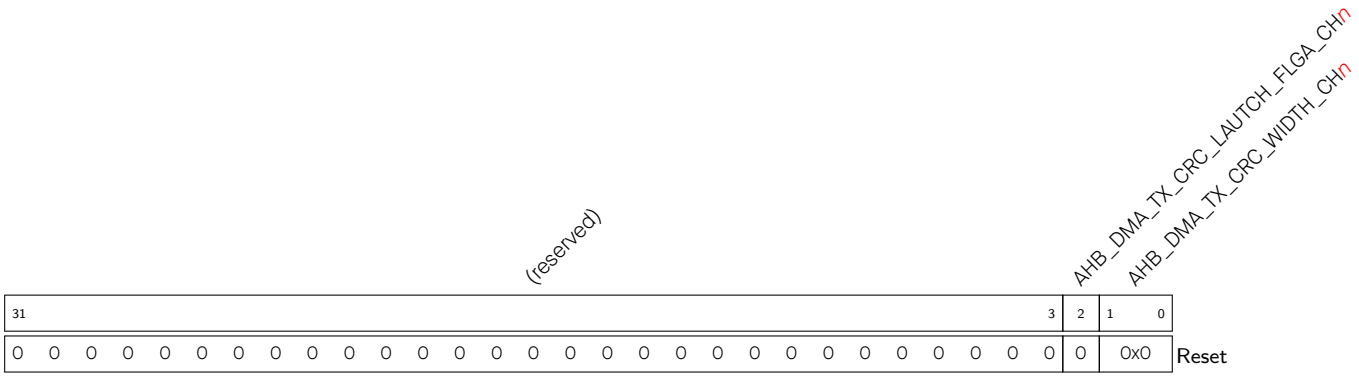
(RO)

Register 6.19. AHB_DMA_OUT_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x02BC+0x28**n*)



AHB_DMA_OUT_CRC_INIT_DATA_CH*n* 配置发送通道 *n* 的 CRC 初始值。(R/W)

Register 6.20. AHB_DMA_TX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x02C0+0x28**n*)



AHB_DMA_TX_CRC_WIDTH_CH*n* 配置发送通道 *n* 的 CRC 计算结果位宽。

0：≤ 8 位

1：≤ 16 位

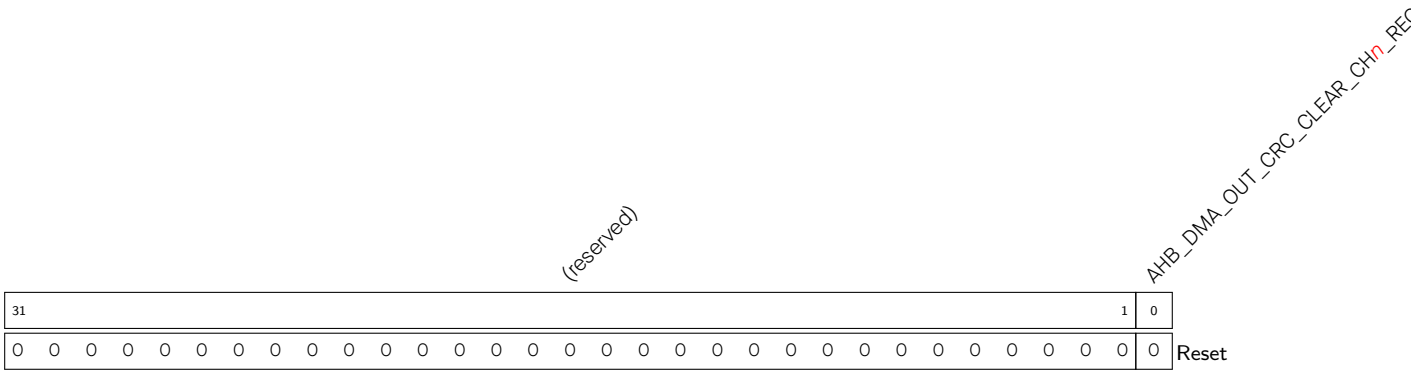
2：≤ 24 位

3：≤ 32 位

(R/W)

AHB_DMA_TX_CRC_LAUTCH_FLGA_CH*n* 先写 1 再写 0 锁存 AHB_DMA_TX_CRC_EN_ADDR_CH*n*、AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n*、AHB_DMA_TX_CRC_EN_WR_DATA_CH*n* 和 AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* 的值。(R/W)

Register 6.21. AHB_DMA_OUT_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x02C4+0x28**n*)



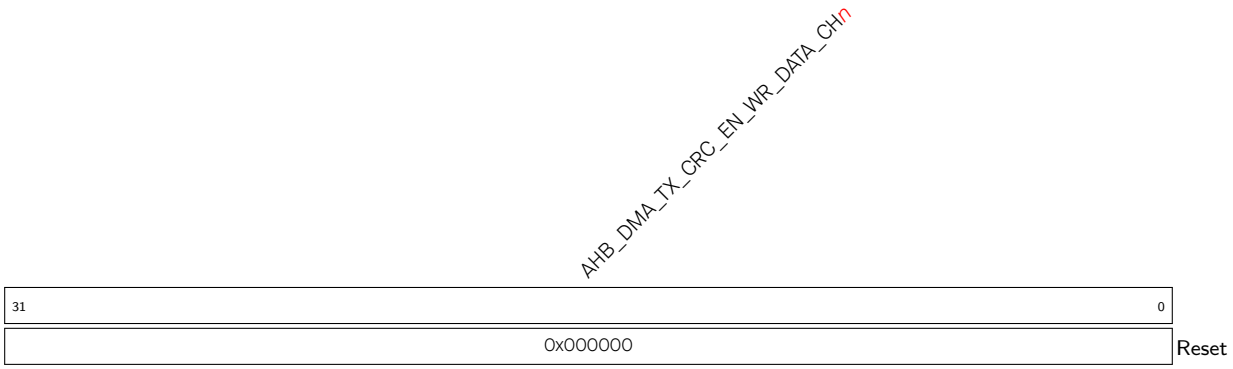
AHB_DMA_OUT_CRC_CLEAR_CH*n*_REG 保留。(R/W)

Register 6.22. AHB_DMA_OUT_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x02C8+0x28*n)



AHB_DMA_OUT_CRC_FINAL_RESULT_CH*n* 表示发送通道 *n* 的 CRC 计算结果。(RO)

Register 6.23. AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x02CC+0x28*n)



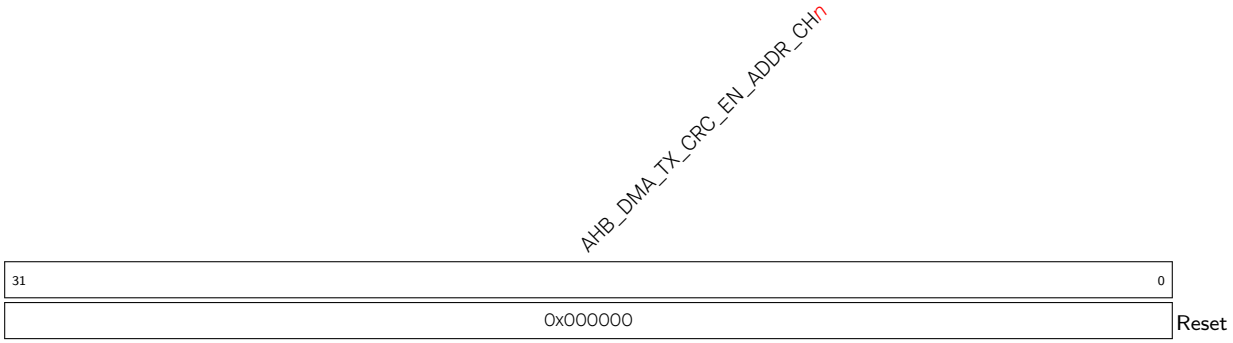
AHB_DMA_TX_CRC_EN_WR_DATA_CH*n* 配置发送通道 *n* 中间结果的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

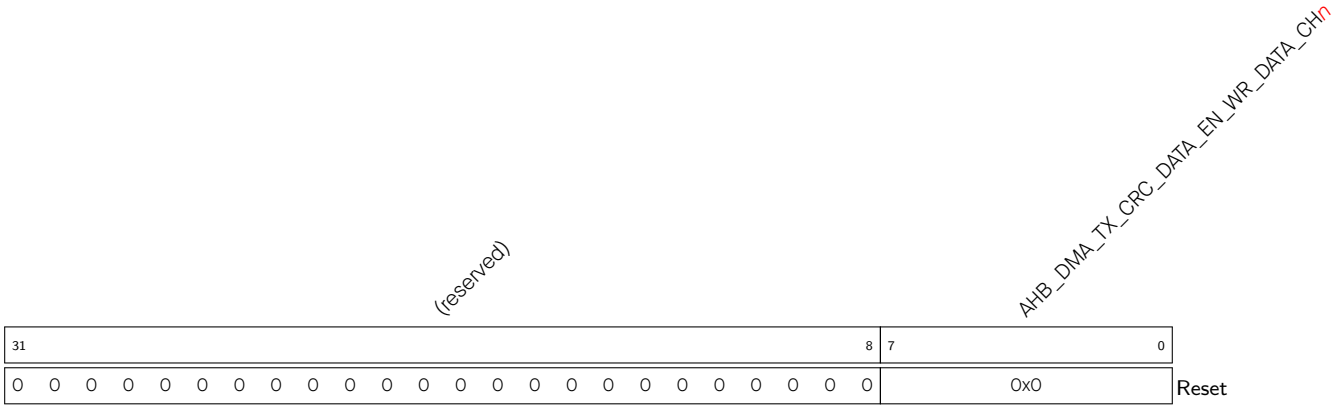
(R/W)

Register 6.24. AHB_DMA_TX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x02D0+0x28**n*)



AHB_DMA_TX_CRC_EN_ADDR_CH*n* 配置发送通道 *n* AHB_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

Register 6.25. AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x02D4+0x28**n*)



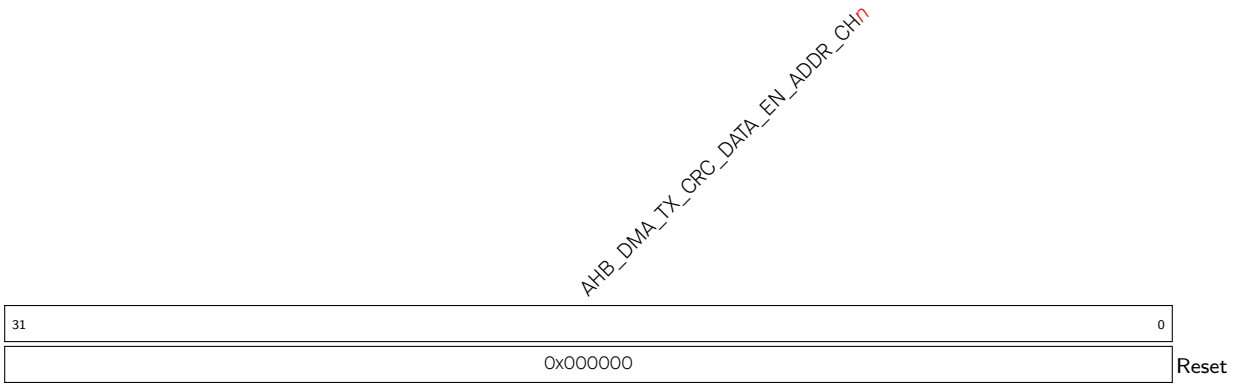
AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* 配置发送通道 *n* 数据的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

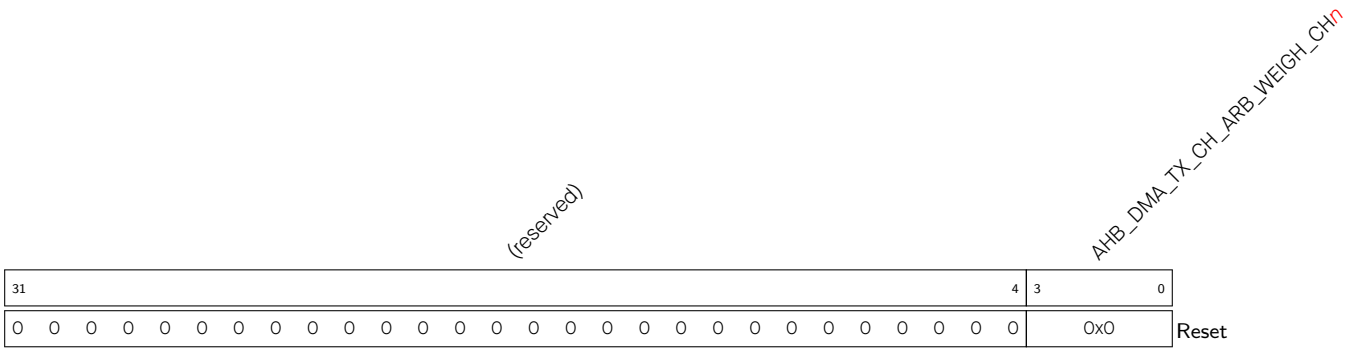
(R/W)

Register 6.26. AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x02D8+0x28**n*)



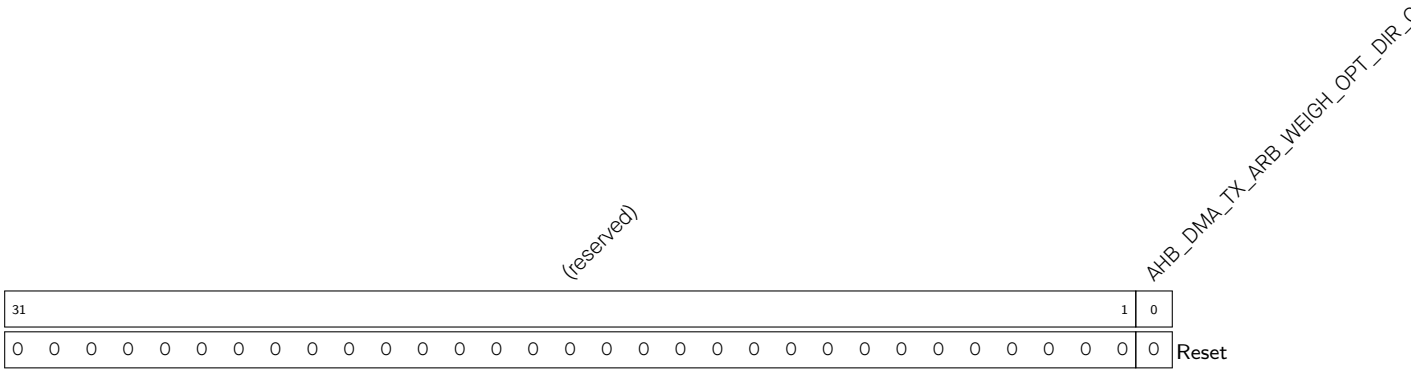
AHB_DMA_TX_CRC_DATA_EN_ADDR_CH*n* 配置发送通道 *n* AHB_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

Register 6.27. AHB_DMA_TX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x02DC+0x28**n*)



AHB_DMA_TX_CH_ARB_WEIGH_CH*n* 配置发送通道 *n* 的权重（即令牌数）。取值范围：0 ~ 15。(R/W)

Register 6.28. AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x02E0+0x28*n)



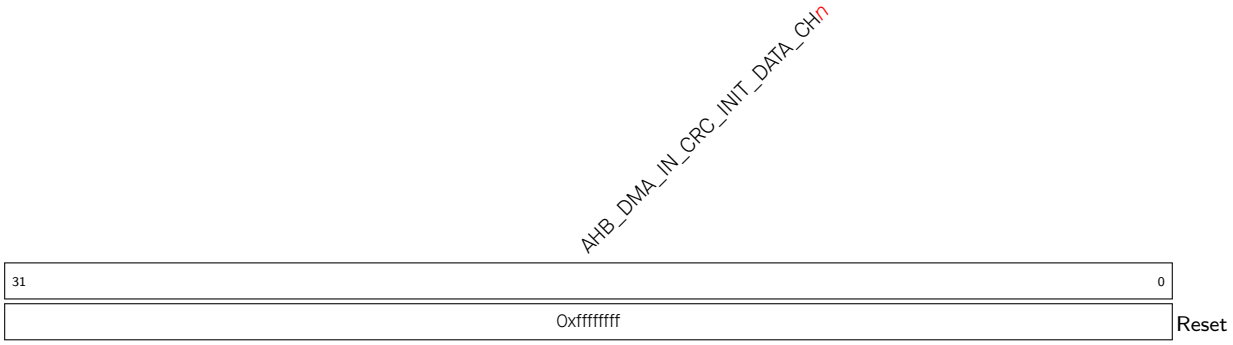
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* 配置发送通道 *n* 是否开启权重仲裁优化。

0：开启

1：关闭

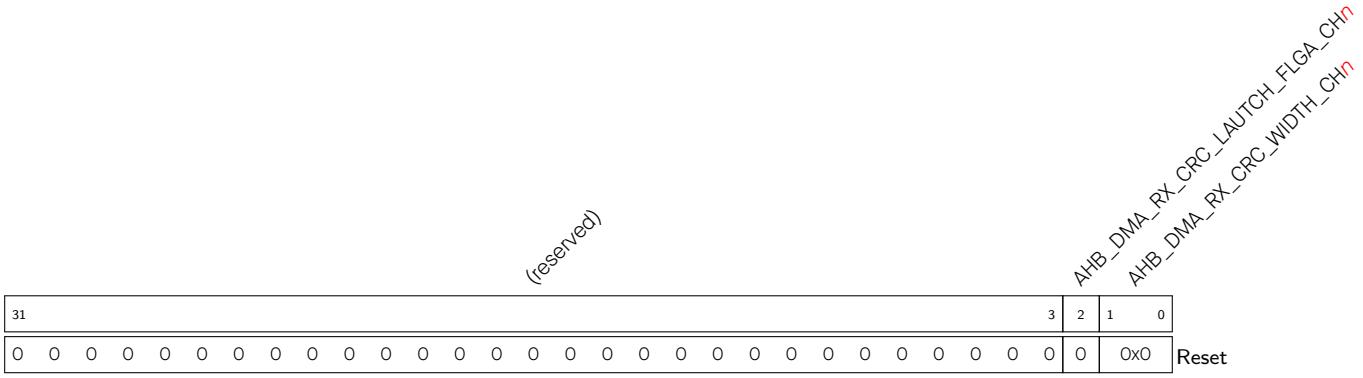
(R/W)

Register 6.29. AHB_DMA_IN_CRC_INIT_DATA_CH*n*_REG (*n*: 0-2) (0x0334+0x28*n)



AHB_DMA_IN_CRC_INIT_DATA_CH*n* 配置接收通道 *n* 的 CRC 初始值。(R/W)

Register 6.30. AHB_DMA_RX_CRC_WIDTH_CH*n*_REG (*n*: 0-2) (0x0338+0x28**n*)



AHB_DMA_RX_CRC_WIDTH_CH*n* 配置接收通道 *n* 的 CRC 计算结果位宽。

0：≤ 8 位

1：≤ 16 位

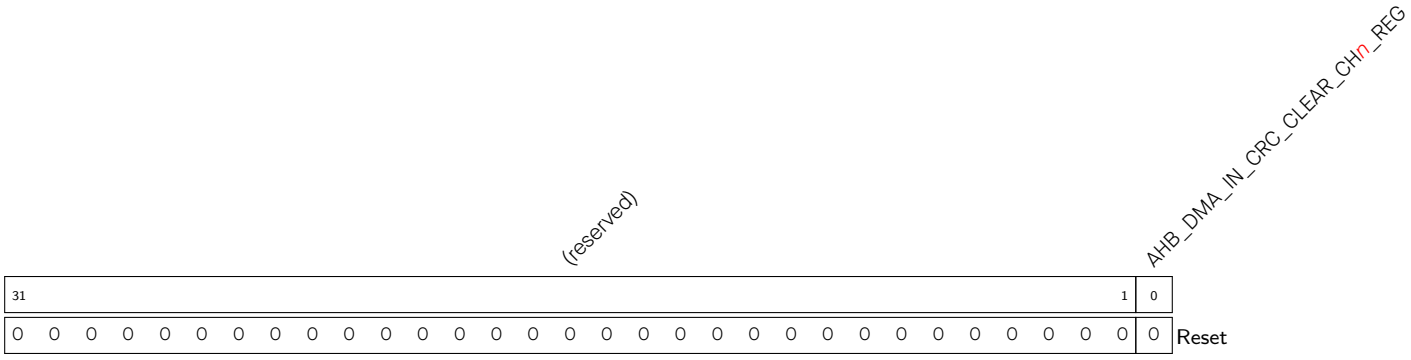
2：≤ 24 位

3：≤ 32 位

(R/W)

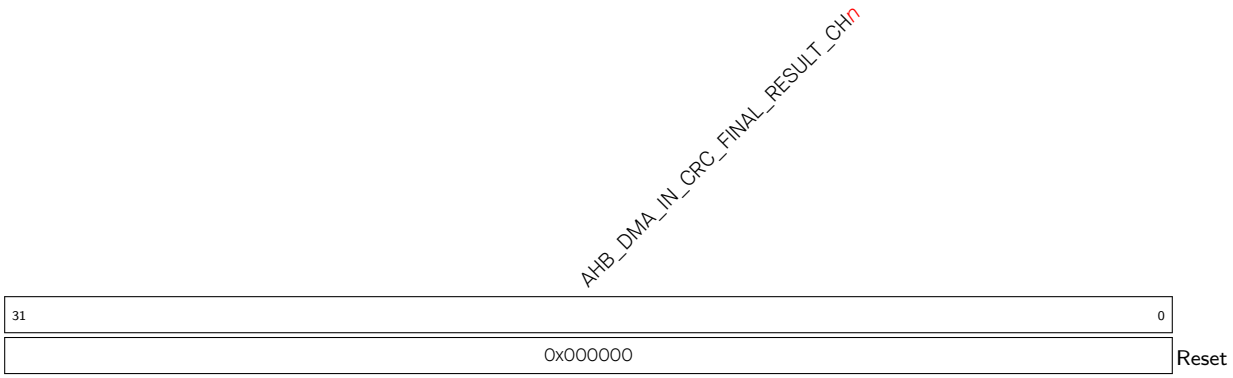
AHB_DMA_RX_CRC_LAUTCH_FLGA_CH*n* 先写 1 再写 0 锁存 AHB_DMA_RX_CRC_EN_ADDR_CH*n*、AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n*、AHB_DMA_RX_CRC_EN_WR_DATA_CH*n* 和 AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* 的值。(R/W)

Register 6.31. AHB_DMA_IN_CRC_CLEAR_CH*n*_REG (*n*: 0-2) (0x033C+0x28**n*)



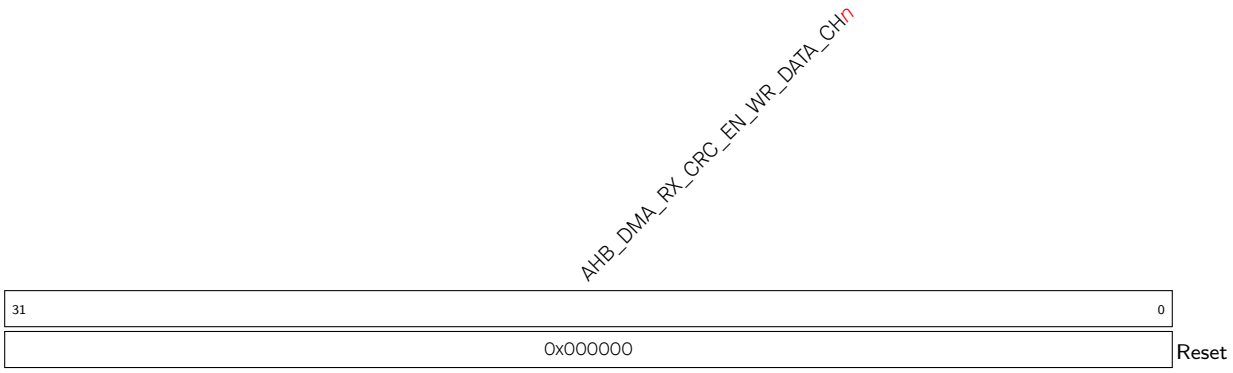
AHB_DMA_IN_CRC_CLEAR_CH*n*_REG 保留。(R/W)

Register 6.32. AHB_DMA_IN_CRC_FINAL_RESULT_CH*n*_REG (*n*: 0-2) (0x0340+0x28**n*)



AHB_DMA_IN_CRC_FINAL_RESULT_CH*n* 表示接收通道 *n* 的 CRC 计算结果。(RO)

Register 6.33. AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x0344+0x28**n*)



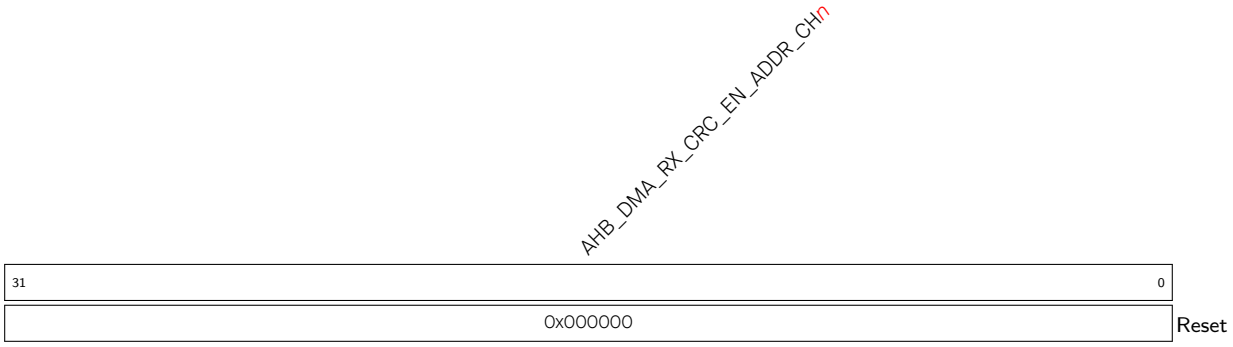
AHB_DMA_RX_CRC_EN_WR_DATA_CH*n* 配置接收通道 *n* 中间结果的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

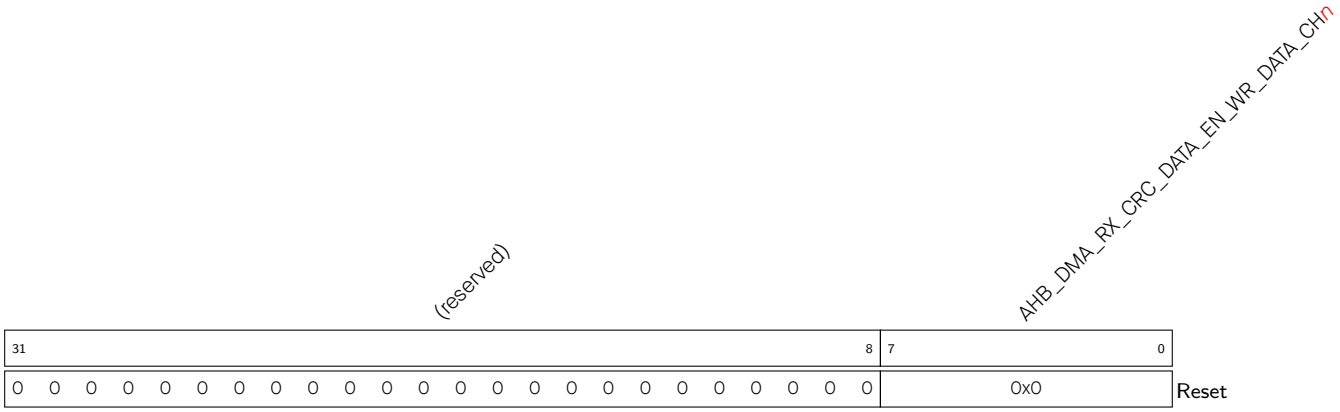
(R/W)

Register 6.34. AHB_DMA_RX_CRC_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0348+0x28**n*)



AHB_DMA_RX_CRC_EN_ADDR_CH*n* 配置接收通道 *n* AHB_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

Register 6.35. AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*: 0-2) (0x034C+0x28**n*)



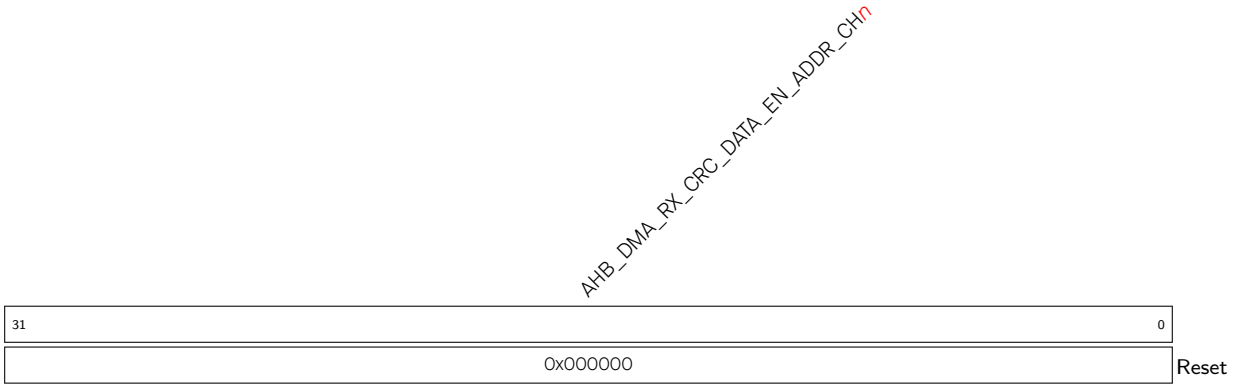
AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* 接收通道 *n* 数据的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

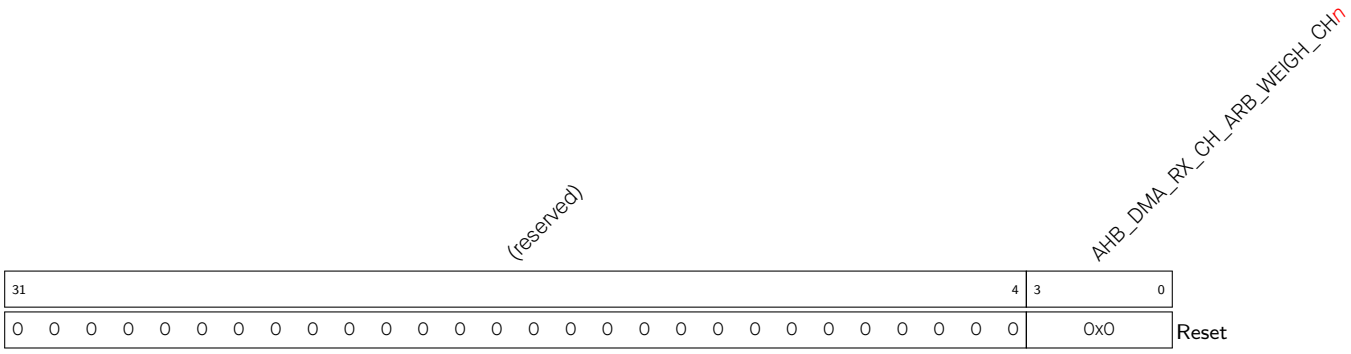
(R/W)

Register 6.36. AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*: 0-2) (0x0350+0x28*n)



AHB_DMA_RX_CRC_DATA_EN_ADDR_CH*n* 配置接收通道 *n* AHB_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

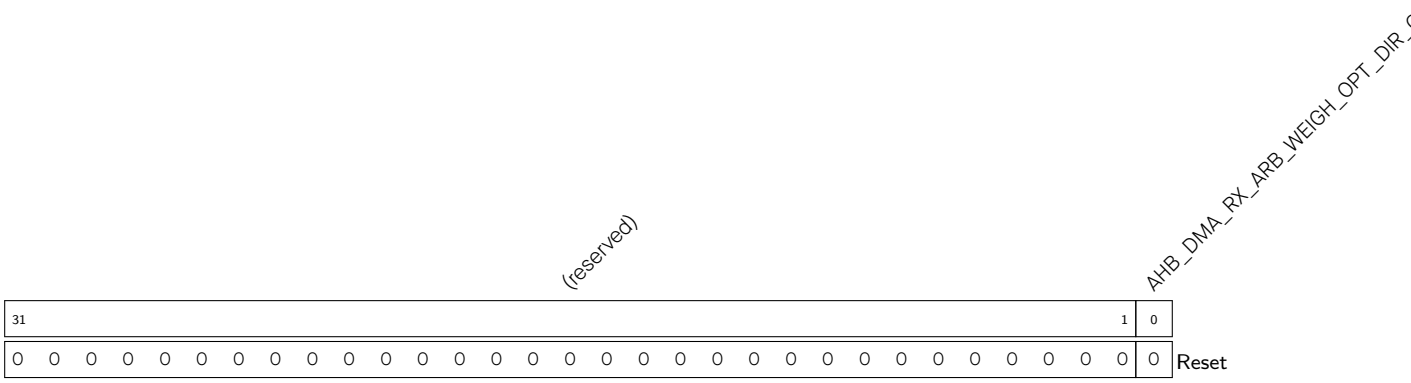
Register 6.37. AHB_DMA_RX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x0354+0x28*n)



AHB_DMA_RX_CH_ARB_WEIGH_CH*n* 配置接收通道 *n* 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

Register 6.38. AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x0358+0x28**n*)



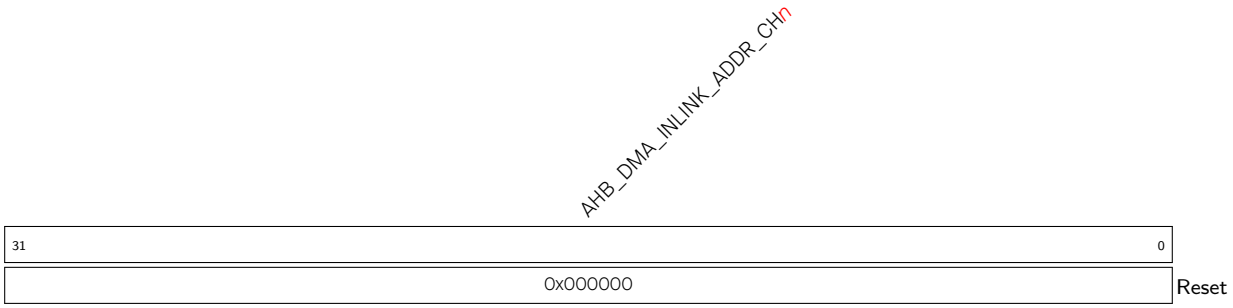
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* 配置接收通道 *n* 是否开启权重仲裁优化。

0：开启

1：关闭

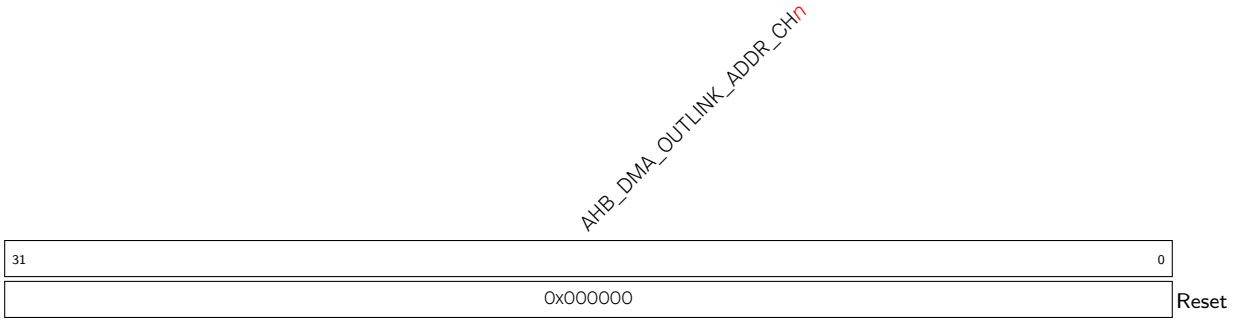
(R/W)

Register 6.39. AHB_DMA_IN_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03AC+0x4**n*)



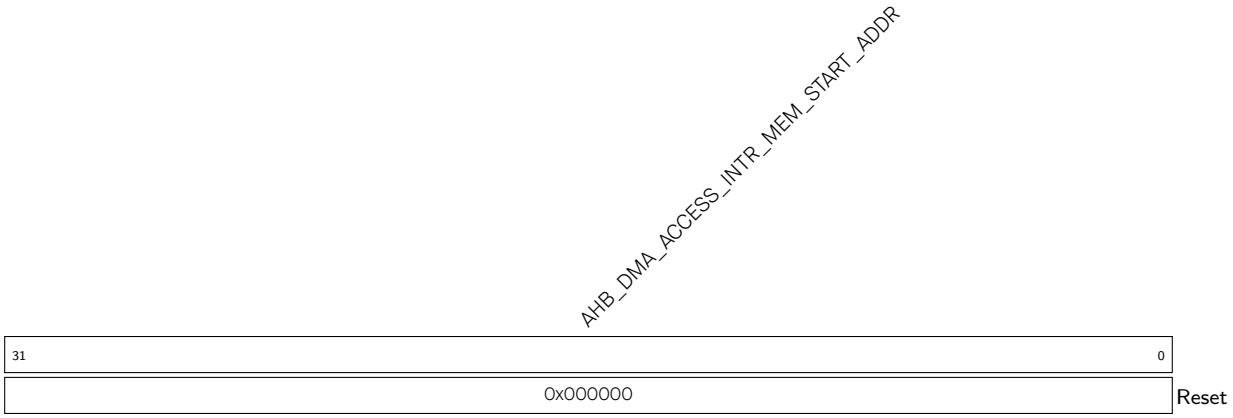
AHB_DMA_INLINK_ADDR_CH*n* 表示第一个接收链表描述符的地址。(R/W)

Register 6.40. AHB_DMA_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03B8+0x4**n*)



AHB_DMA_OUTLINK_ADDR_CH*n* 表示第一个发送链表描述符的地址。(R/W)

Register 6.41. AHB_DMA_INTR_MEM_START_ADDR_REG (0x03C4)



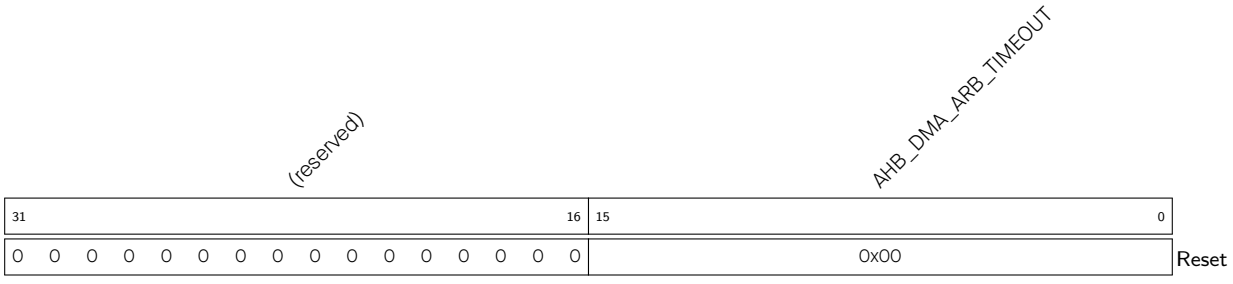
AHB_DMA_ACCESS_INTR_MEM_START_ADDR 配置可访问地址空间的起始地址。(R/W)

Register 6.42. AHB_DMA_INTR_MEM_END_ADDR_REG (0x03C8)



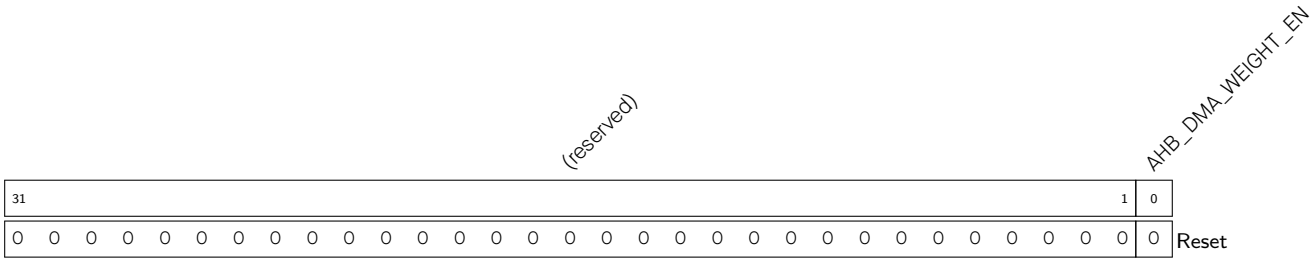
AHB_DMA_ACCESS_INTR_MEM_END_ADDR 配置可访问地址空间的结束地址。(R/W)

Register 6.43. AHB_DMA_ARB_TIMEOUT_REG (0x03DC)



AHB_DMA_ARB_TIMEOUT 配置权重仲裁时间片。单位：AHB 总线时钟周期。(R/W)

Register 6.44. AHB_DMA_WEIGHT_EN_REG (0x0400)



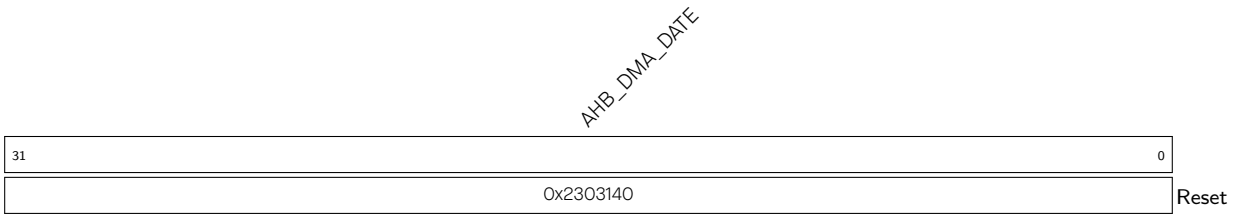
AHB_DMA_WEIGHT_EN 配置开启权重仲裁。

0：关闭

1：开启

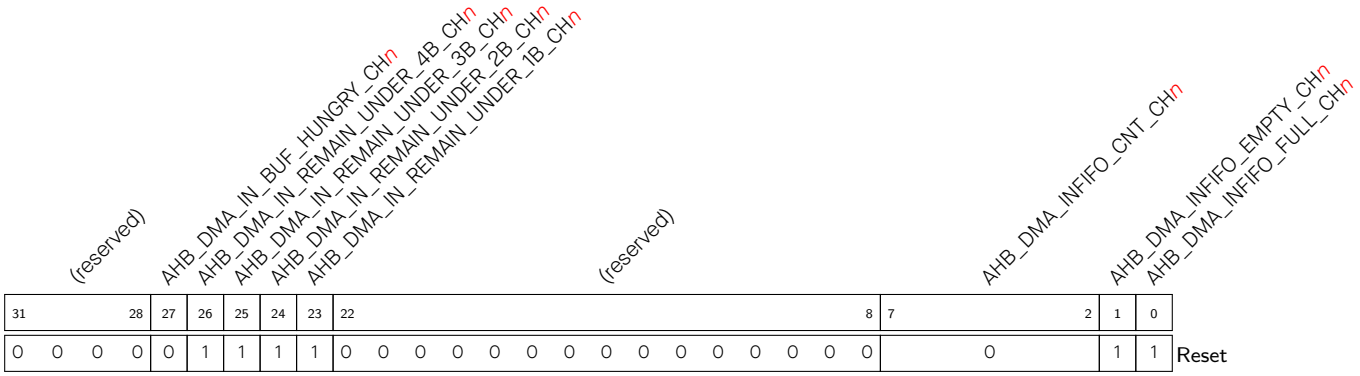
(R/W)

Register 6.45. AHB_DMA_DATE_REG (0x0068)



AHB_DMA_DATE 版本控制寄存器。(R/W)

Register 6.46. AHB_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0078+0xC0*n*)



AHB_DMA_INFIFO_FULL_CH*n* 表示 L1 RX FIFO 是否为满。

0：未满

1：已满

(RO)

AHB_DMA_INFIFO_EMPTY_CH*n* 表示 L1 RX FIFO 是否为空。

0：未空

1：已空

(RO)

AHB_DMA_INFIFO_CNT_CH*n* 表示接收通道 *n* L1 RX FIFO 中的字节数。(RO)

AHB_DMA_IN_REMAIN_UNDER_1B_CH*n* 保留。(RO)

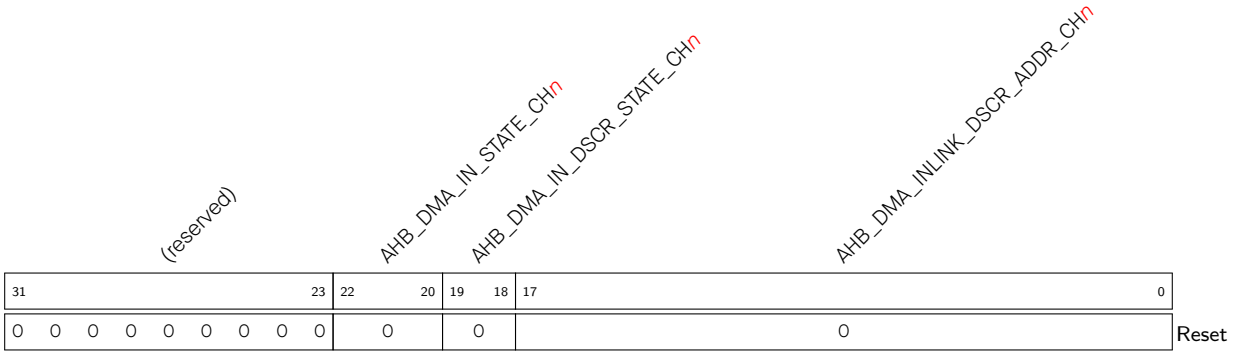
AHB_DMA_IN_REMAIN_UNDER_2B_CH*n* 保留。(RO)

AHB_DMA_IN_REMAIN_UNDER_3B_CH*n* 保留。(RO)

AHB_DMA_IN_REMAIN_UNDER_4B_CH*n* 保留。(RO)

AHB_DMA_IN_BUF_HUNGRY_CH*n* 保留。(RO)

Register 6.47. AHB_DMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0084+0xC0*n*)

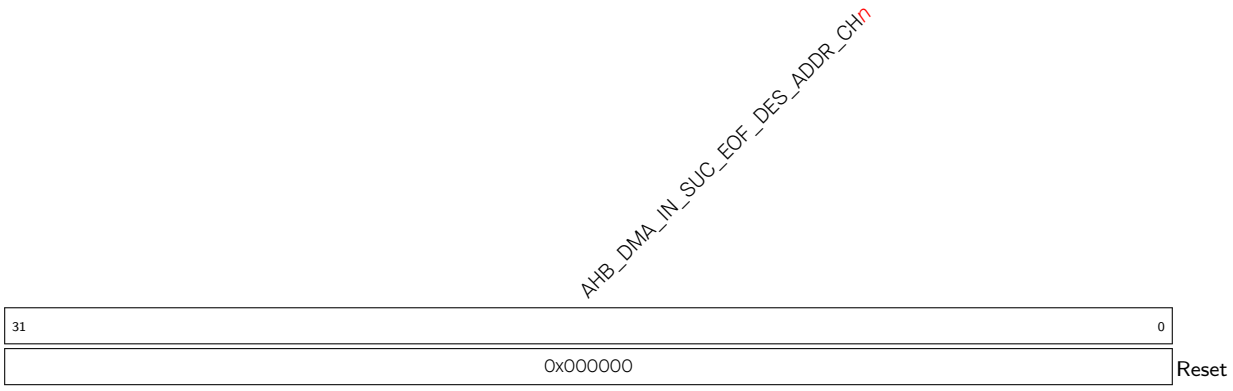


AHB_DMA_INLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的接收链表描述符所在地址的低 18 位。如果当前处理的接收链表描述符是最后一个描述符，则该字段表示当前处理的接收链表描述符地址。(RO)

AHB_DMA_IN_DSCR_STATE_CH*n* 保留。(RO)

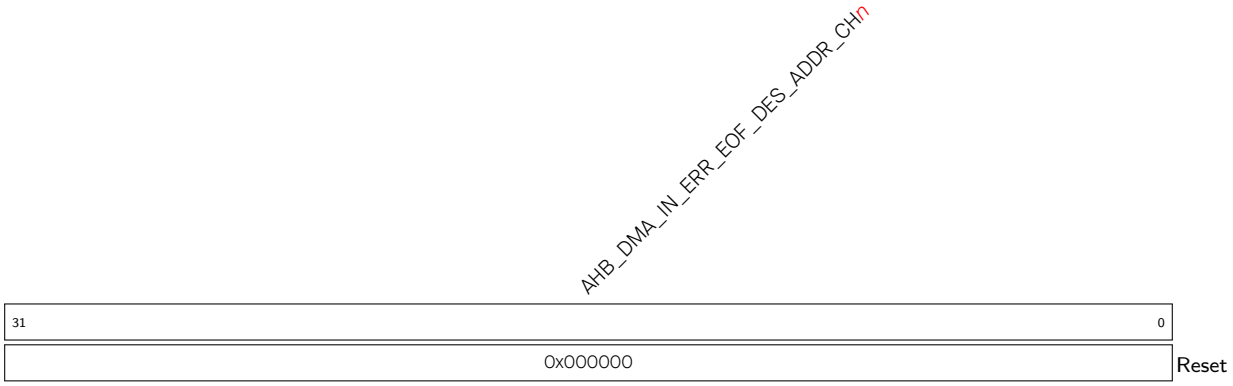
AHB_DMA_IN_STATE_CH*n* 保留。(RO)

Register 6.48. AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0088+0xC0*n*)



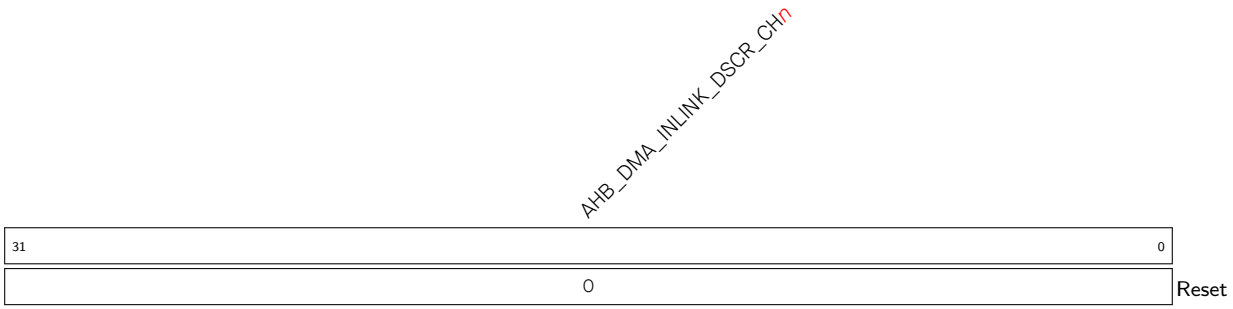
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的接收链表描述符的地址。(RO)

Register 6.49. AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x008C+0xC0*n*)



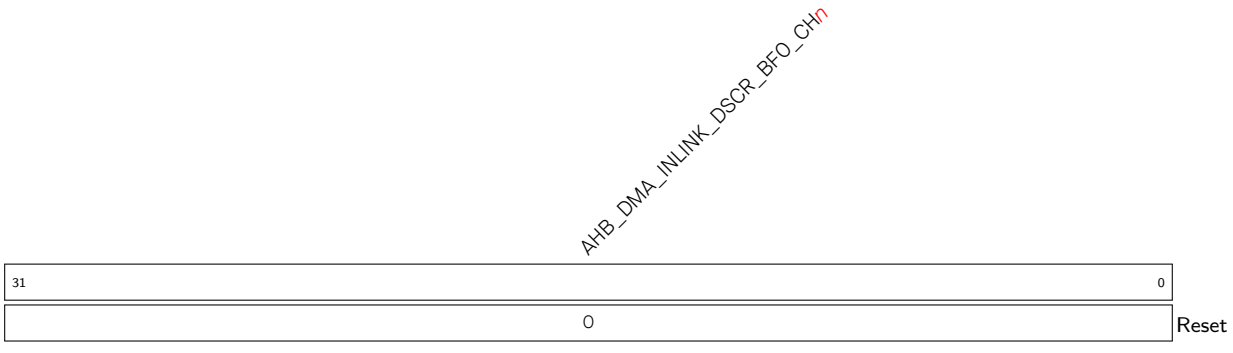
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n* 表示当前接收数据有错时，相应接收链表描述符的位置。仅对 UHCI 有效。(RO)

Register 6.50. AHB_DMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0090+0xC0*n*)



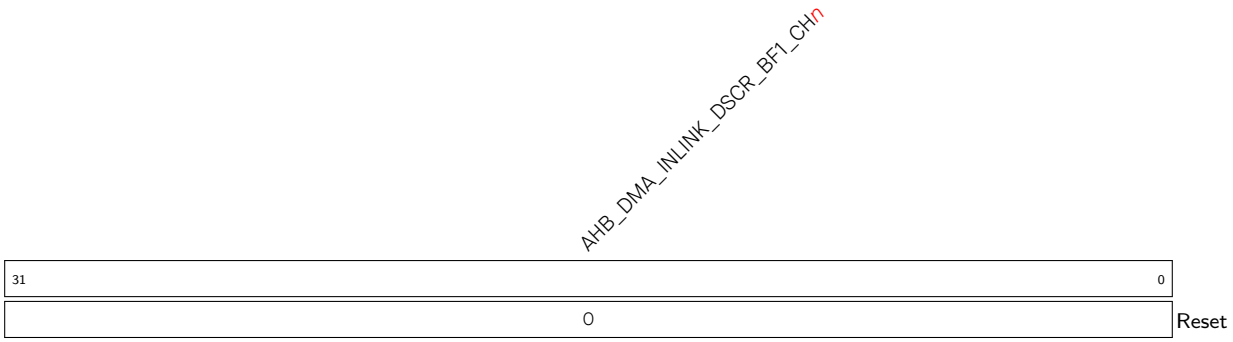
AHB_DMA_INLINK_DSCR_CH*n* 表示当前已预读取的接收链表描述符指向的下一个接收链表描述符地址 x+1。(RO)

Register 6.51. AHB_DMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0094+0xC0*n*)



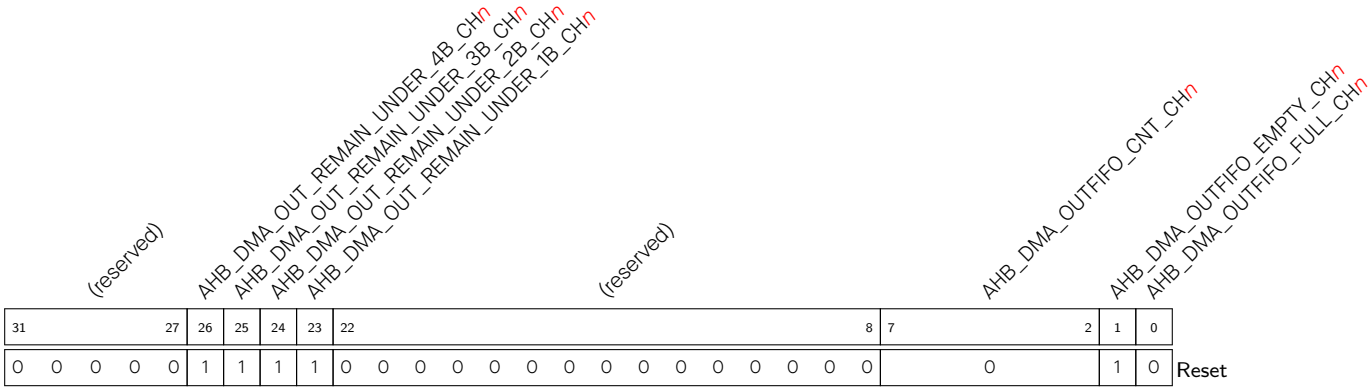
AHB_DMA_INLINK_DSCR_BF0_CH*n* 表示当前已预读取的接收链表描述符所在地址 x。(RO)

Register 6.52. AHB_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0098+0xC0*n*)



AHB_DMA_INLINK_DSCR_BF1_CH*n* 表示前一个已预读取的接收链表描述符所在地址 x-1。(RO)

Register 6.53. AHB_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x00D8+0xC0*n*)



AHB_DMA_OUTFIFO_FULL_CH*n* 表示 L1 TX FIFO 是否为满。

0：未满

1：已满

(RO)

AHB_DMA_OUTFIFO_EMPTY_CH*n* 表示 L1 TX FIFO 是否为空。

0：未空

1：已空

(RO)

AHB_DMA_OUTFIFO_CNT_CH*n* 表示发送通道 *n* L1 TX FIFO 中的字节数。(RO)

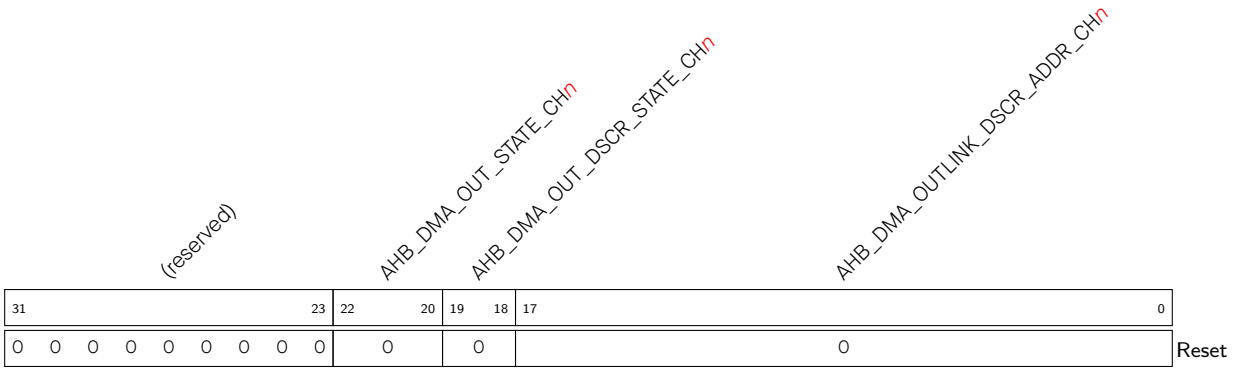
AHB_DMA_OUT_REMAIN_UNDER_1B_CH*n* 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_2B_CH*n* 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_3B_CH*n* 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_4B_CH*n* 保留。(RO)

Register 6.54. AHB_DMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x00E4+0xC0*n*)

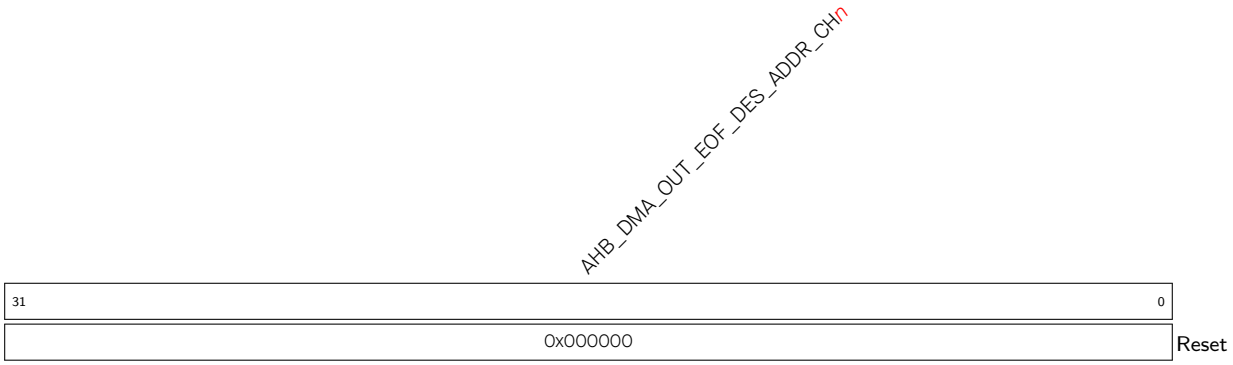


AHB_DMA_OUTLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的发送接收链表描述符所在地址的低 18 位。如果当前处理的发送链表描述符是最后一个描述符，则该字段表示当前处理的发送链表描述符地址。(RO)

AHB_DMA_OUT_DSCR_STATE_CH*n* 保留。(RO)

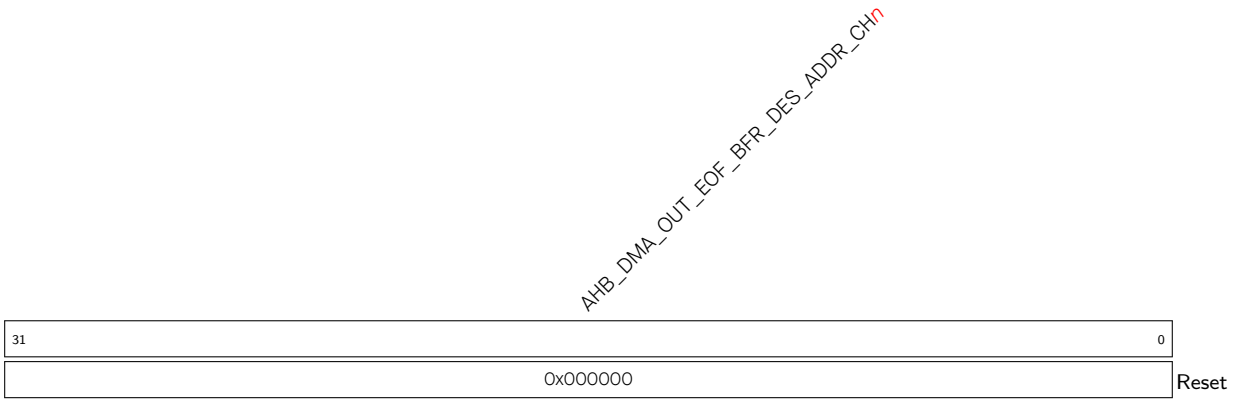
AHB_DMA_OUT_STATE_CH*n* 保留。(RO)

Register 6.55. AHB_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00E8+0xC0*n*)



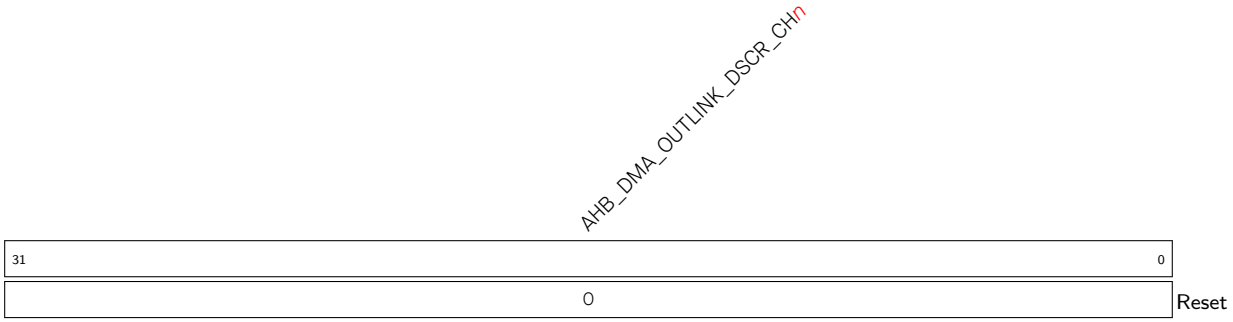
AHB_DMA_OUT_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的发送链表描述符的地址。(RO)

Register 6.56. AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00EC+0xC0**n*)



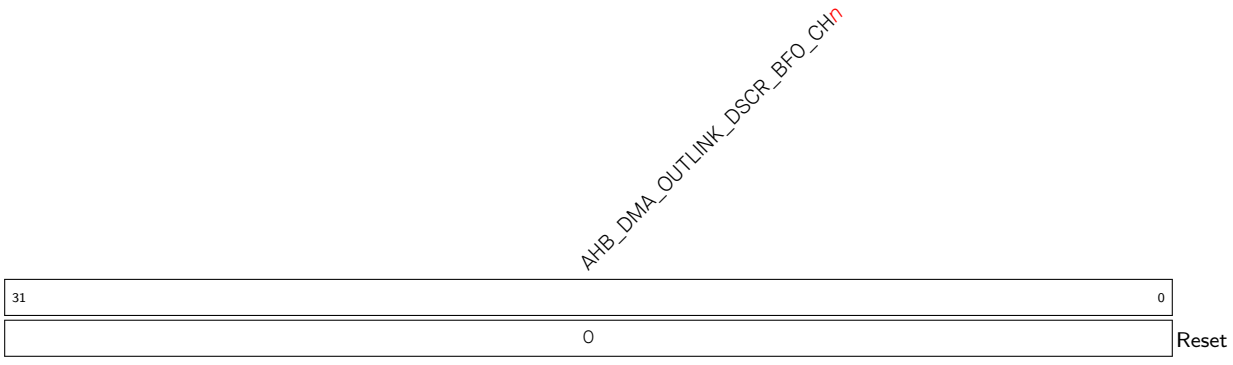
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* 表示倒数第二个发送链表描述符的地址。(RO)

Register 6.57. AHB_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x00F0+0xC0**n*)



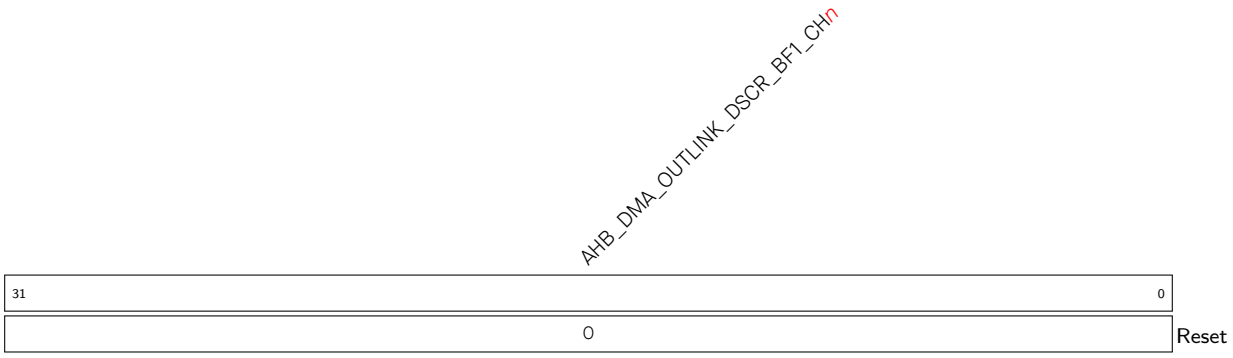
AHB_DMA_OUTLINK_DSCR_CH*n* 表示当前已预读取的发送链表描述符指向的下一个发送链表描述符地址 y+1。(RO)

Register 6.58. AHB_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x00F4+0xC0**n*)



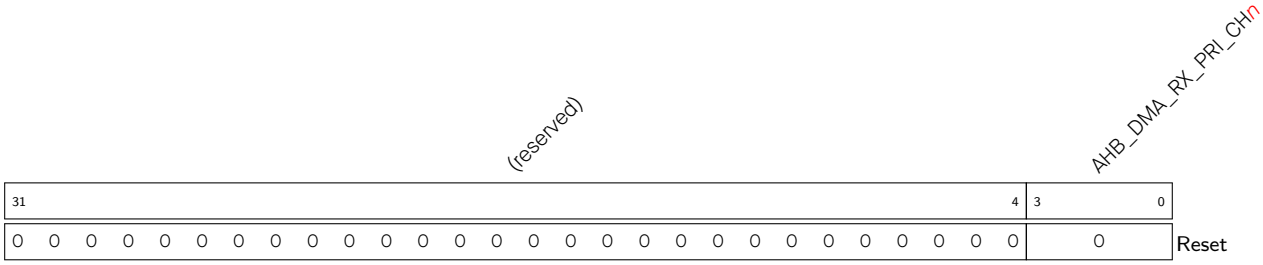
AHB_DMA_OUTLINK_DSCR_BF0_CH*n* 表示当前已预读取的发送链表描述符所在地址 y。(RO)

Register 6.59. AHB_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x00F8+0xC0*n*)



AHB_DMA_OUTLINK_DSCR_BF1_CH*n* 表示前一个已预读取的发送链表描述符所在地址 y-1。(RO)

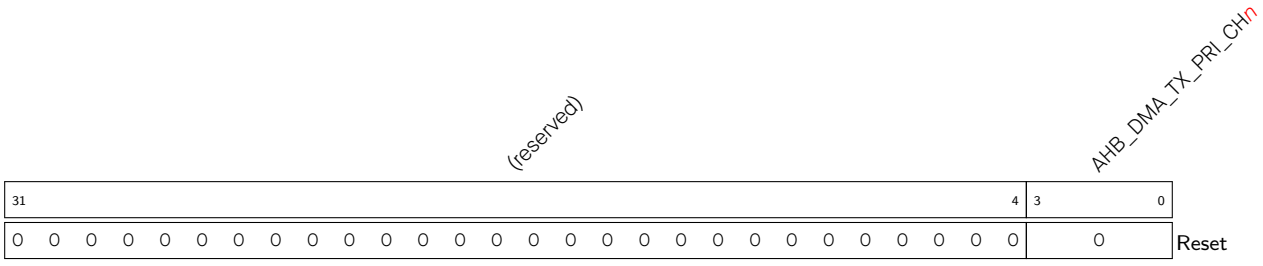
Register 6.60. AHB_DMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x009C+0xC0*n*)



AHB_DMA_RX_PRI_CH*n* 配置接收通道 *n* 的优先级。值越大，优先级越高。

取值范围：0 ~ 5。(R/W)

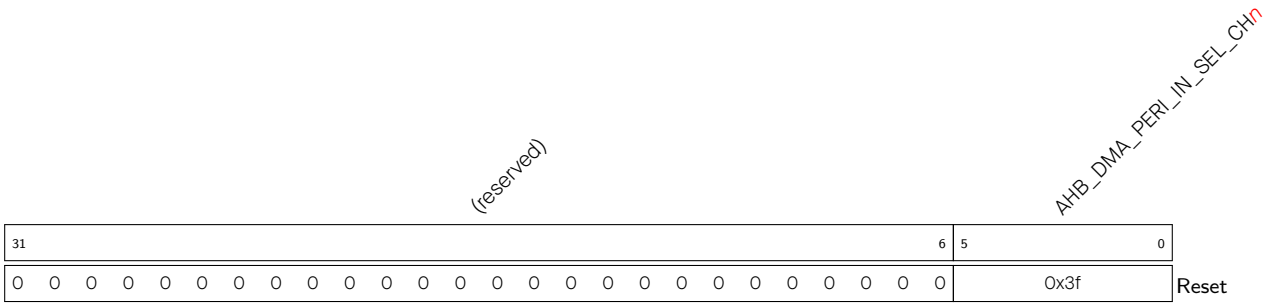
Register 6.61. AHB_DMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x00FC+0xC0*n*)



AHB_DMA_TX_PRI_CH*n* 配置发送通道 *n* 的优先级。值越大，优先级越高。

取值范围：0 ~ 5。(R/W)

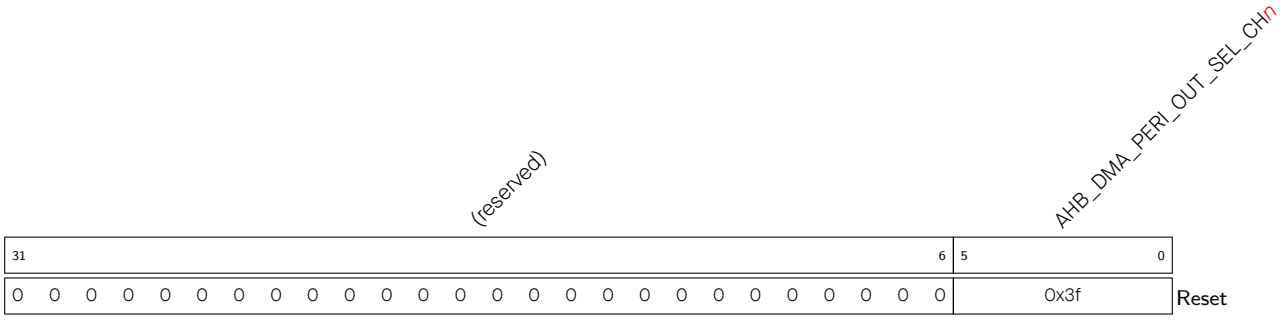
Register 6.62. AHB_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x00A0+0xC0**n*)



AHB_DMA_PERI_IN_SEL_CH*n* 配置连接接收通道 *n* 的外设。

0：I3C
1：Dummy-1
2：UHCI
3：I2S0
4：I2S1
5：I2S2
6：Dummy-6
7：Dummy-7
8：ADC
9：Dummy-9
10：RMT
11 ~ 15：Dummy-11 ~ Dummy-15
16 ~ 63：无效值
(R/W)

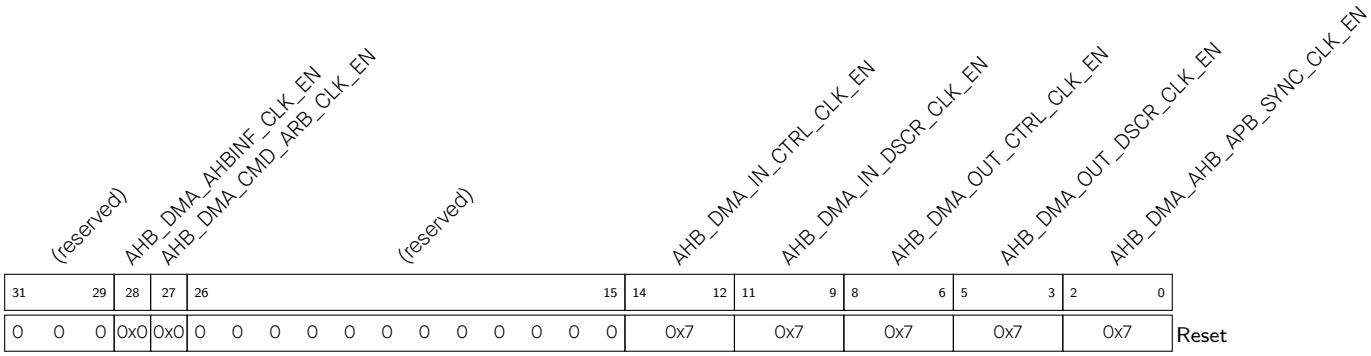
Register 6.63. AHB_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0100+0xC0**n*)



AHB_DMA_PERI_OUT_SEL_CH*n* 配置连接发送通道 *n* 的外设。

0：I3C
1：Dummy-1
2：UHCI
3：I2S0
4：I2S1
5：I2S2
6：Dummy-6
7：Dummy-7
8：ADC
9：Dummy-9
10：RMT
11 ~ 15：Dummy-11 ~ Dummy-15
16 ~ 63：无效值
(R/W)

Register 6.64. AHB_DMA_MODULE_CLK_EN_REG (0x0404)



AHB_DMA_AHB_APB_SYNC_CLK_EN 配置强制开启 AHB DMA 寄存器配置同步模块时钟。

位 0 对应发送、接收通道 0，位 1 对应发送、接收通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_OUT_DSCR_CLK_EN 配置强制开启 AHB DMA TX 链表描述符处理模块时钟。

位 0 对应发送通道 0，位 1 对应发送通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_OUT_CTRL_CLK_EN 配置强制开启 AHB DMA TX 数据处理模块时钟。

位 0 对应发送通道 0，位 1 对应发送通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_IN_DSCR_CLK_EN 配置强制开启 AHB DMA RX 链表描述符处理模块时钟。

位 0 对应接收通道 0，位 1 对应接收通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_IN_CTRL_CLK_EN 配置强制开启 AHB DMA RX 数据处理模块时钟。

位 0 对应接收通道 0，位 1 对应接收通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_CMD_ARB_CLK_EN 配置强制开启 AHB DMA 仲裁模块时钟。

0：不强制开启

1：强制开启

(R/W)

见下页...

Register 6.64. AHB_DMA_MODULE_CLK_EN_REG (0x0404)

接上页...

AHB_DMA_AHBINF_CLK_EN 配置强制开启 AHB DMA 总线接口处理模块时钟。

0：不强制开启

1：强制开启

(R/W)

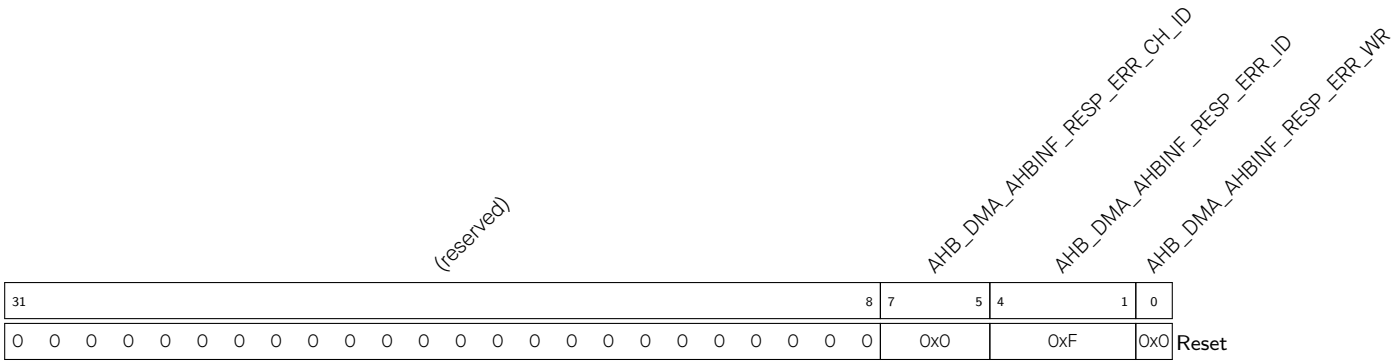
Register 6.65. AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG (0x0408)



AHB_DMA_AHBINF_RESP_ERR_ADDR 表示 AHB DMA 收到 AHB 总线错误响应对应传输的地址。

(RO)

Register 6.66. AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG (0x040C)



AHB_DMA_AHBINF_RESP_ERR_CH_ID 表示 AHB DMA 收到 AHB 总线错误响应对应传输的通道 ID。

位 [0:1]：通道号。范围：0 ~2

位 [2]：通道方向。0：RX 方向

1：TX 方向

(RO)

AHB_DMA_AHBINF_RESP_ERR_ID 表示 AHB DMA 收到 AHB 总线错误响应对应传输的传输 ID。

0：I3C 和存储器之间的传输

2：UHCI 和存储器之间的传输

3：I2S0 和存储器之间的传输

4：I2S1 和存储器之间的传输

5：I2S2 和存储器之间的传输

8：ADC 和存储器之间的传输

10：RMT 和存储器之间的传输

11：通道 1 存储器到存储器的传输

12：通道 2 存储器到存储器的传输

其余值为无效值

(RO)

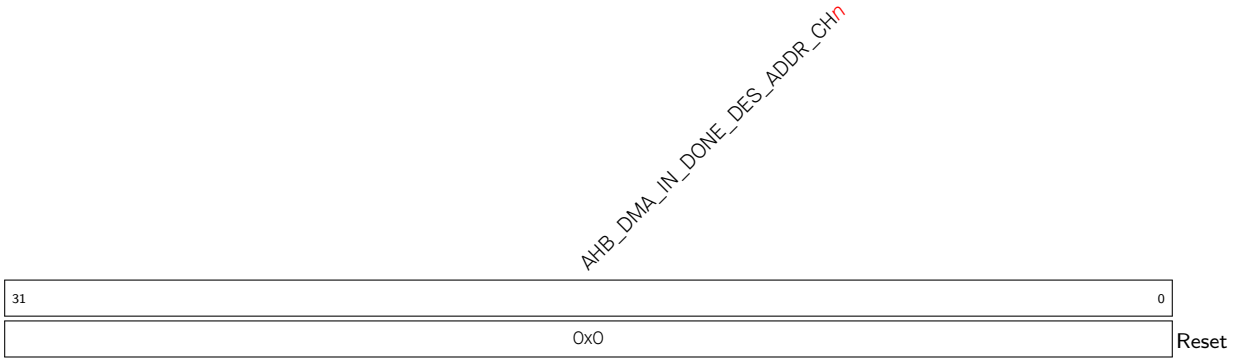
AHB_DMA_AHBINF_RESP_ERR_WR 表示 AHB DMA 收到 AHB 总线错误响应对应传输的类型。

0：读传输

1：写传输

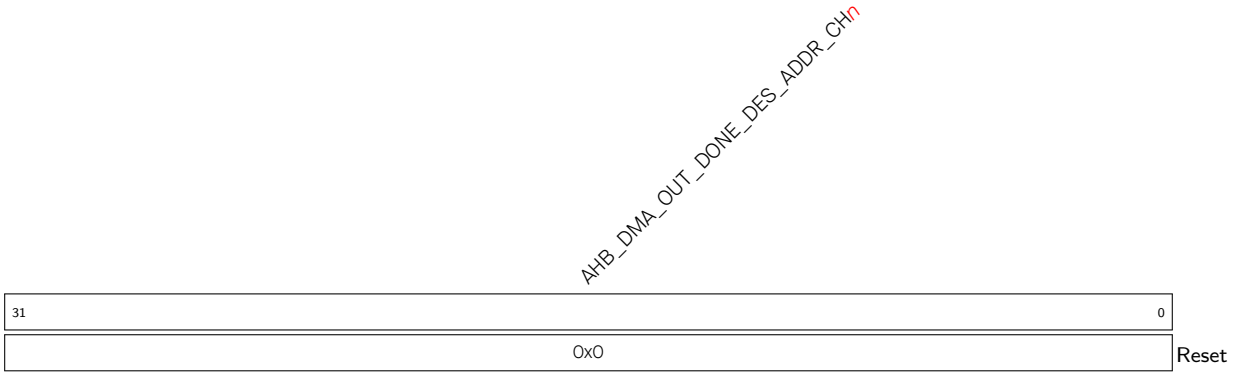
(RO)

Register 6.67. AHB_DMA_IN_DONE_DES_ADDR_CH*n*_REG(*n*: 0-2) (0x0410+0x08**n*)



AHB_DMA_IN_DONE_DES_ADDR_CH*n* 表示 RX 方向传输完成时对应描述符的地址。(RO)

Register 6.68. AHB_DMA_OUT_DONE_DES_ADDR_CH*n*_REG(*n*: 0-2) (0x0414+0x08**n*)



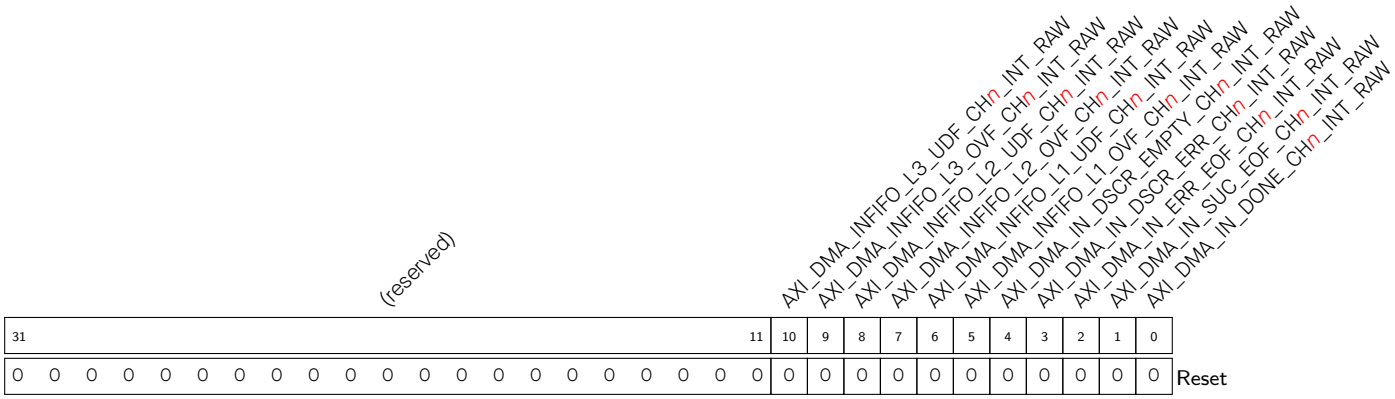
AHB_DMA_OUT_DONE_DES_ADDR_CH*n* 表示 TX 方向传输完成时对应描述符的地址。(RO)

6.9.2 GDMA-AXI 寄存器

本小节的所有地址均为相对于 GDMA-AXI 基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 6.69. AXI_DMA_IN_INT_RAW_CH*n*_REG (*n*：0-2) (0x0000+0x68**n*)



AXI_DMA_IN_DONE_CH*n*_INT_RAW AXI_DMA_IN_DONE_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_RAW AXI_DMA_IN_SUC_EOF_CH*n*_INT 的原始中断状态。对于 UHCI，当接收链表描述符指向的最后一个数据字节已被接收，并且未检测到接收通道 0 的数据错误时，此位变为 1。(R/WTC/SS)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_RAW AXI_DMA_IN_ERR_EOF_CH*n*_INT 的原始中断状态。仅对 UHCI 有效。(R/WTC/SS)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_RAW AXI_DMA_IN_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_RAW AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_RAW AXI_DMA_INFIFO_L1_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_RAW AXI_DMA_INFIFO_L1_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

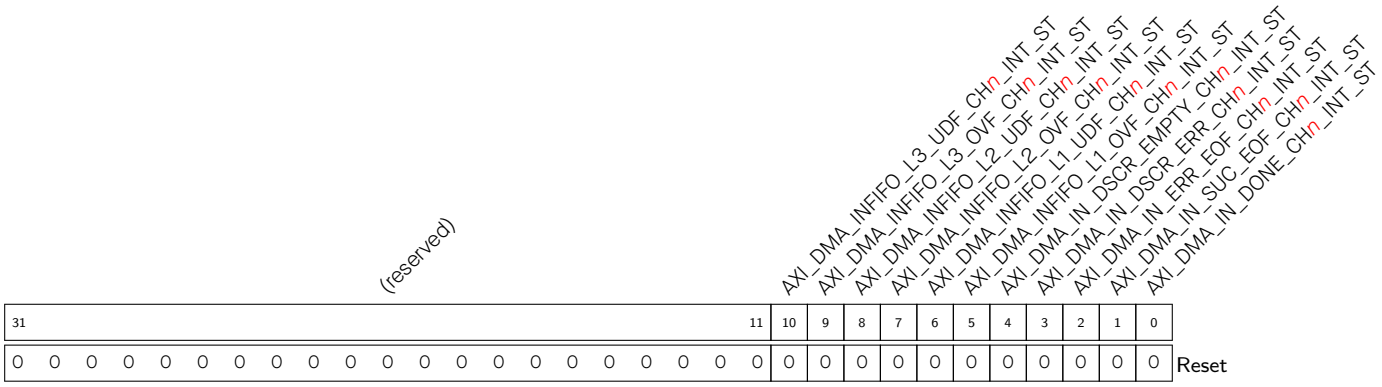
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_RAW AXI_DMA_INFIFO_L2_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_RAW AXI_DMA_INFIFO_L2_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_RAW AXI_DMA_INFIFO_L3_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_RAW AXI_DMA_INFIFO_L3_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 6.70. AXI_DMA_IN_INT_ST_CH*n*_REG (*n*：0-2) (0x0004+0x68***n*)



AXI_DMA_IN_DONE_CH*n*_INT_ST AXI_DMA_IN_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_ST AXI_DMA_IN_SUC_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_ST AXI_DMA_IN_ERR_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_ST AXI_DMA_IN_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_ST AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_ST AXI_DMA_INFIFO_L2_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_ST AXI_DMA_INFIFO_L2_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

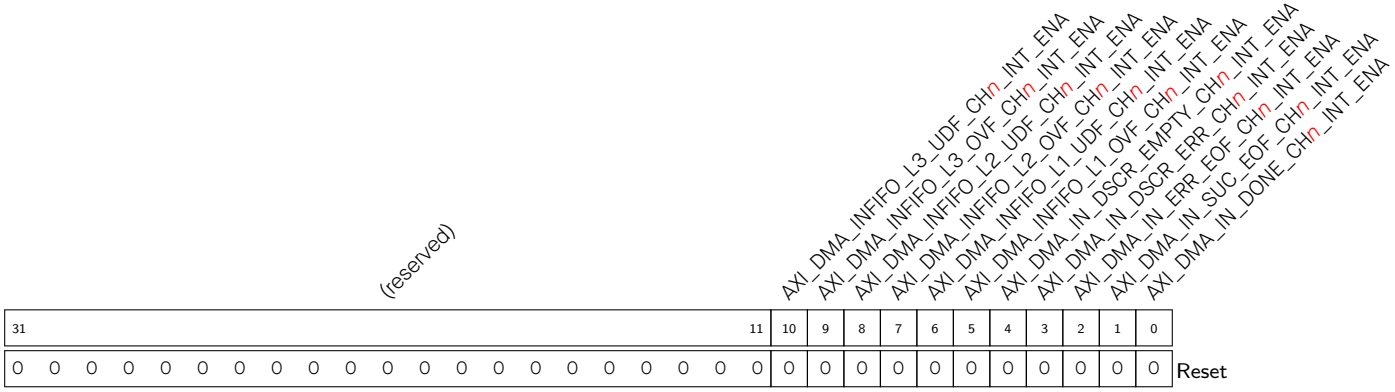
AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_ST AXI_DMA_INFIFO_L1_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_ST AXI_DMA_INFIFO_L1_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_ST AXI_DMA_INFIFO_L3_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_ST AXI_DMA_INFIFO_L3_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

Register 6.71. AXI_DMA_IN_INT_ENA_CH*n*_REG (*n*：0-2) (0x0008+0x68***n*)



AXI_DMA_IN_DONE_CH*n*_INT_ENA 写 1 使能 AXI_DMA_IN_DONE_CH*n*_INT。(R/W)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_IN_SUC_EOF_CH*n*_INT。(R/W)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_IN_ERR_EOF_CH*n*_INT。(R/W)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 AXI_DMA_IN_DSCR_ERR_CH*n*_INT。(R/W)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_ENA 写 1 使能 AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT。(R/W)

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_INFIFO_L1_OVF_CH*n*_INT。(R/W)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_INFIFO_L1_UDF_CH*n*_INT。(R/W)

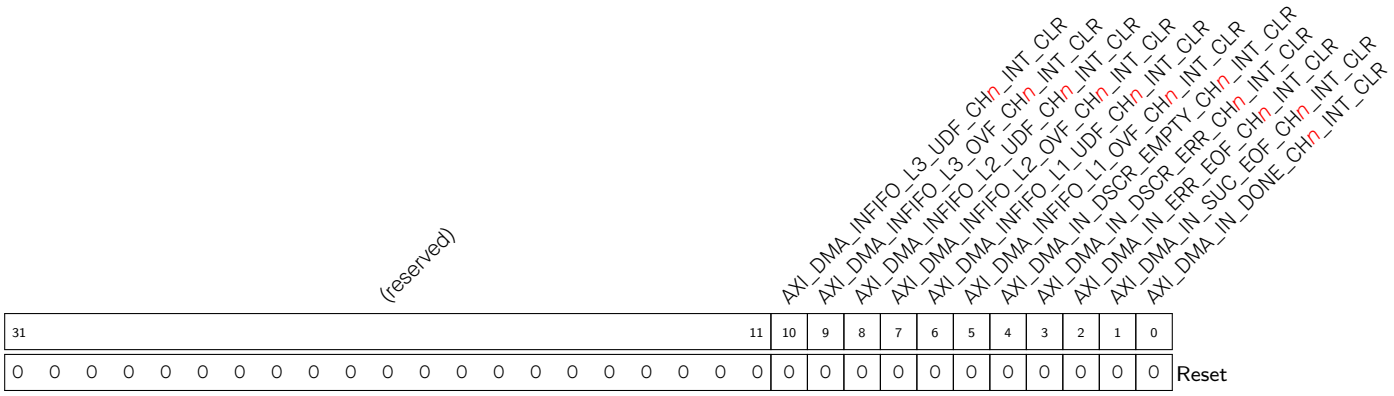
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_INFIFO_L2_OVF_CH*n*_INT。(R/W)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_INFIFO_L2_UDF_CH*n*_INT。(R/W)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_INFIFO_L3_OVF_CH*n*_INT。(R/W)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_INFIFO_L3_UDF_CH*n*_INT。(R/W)

Register 6.72. AXI_DMA_IN_INT_CLR_CH*n*_REG (*n*：0-2) (0x000C+0x68***n*)



AXI_DMA_IN_DONE_CH*n*_INT_CLR 写 1 清除 AXI_DMA_IN_DONE_CH*n*_INT。(WT)

AXI_DMA_IN_SUC_EOF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_IN_SUC_EOF_CH*n*_INT。(WT)

AXI_DMA_IN_ERR_EOF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_IN_ERR_EOF_CH*n*_INT。(WT)

AXI_DMA_IN_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 AXI_DMA_IN_DSCR_ERR_CH*n*_INT。(WT)

AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT_CLR 写 1 清除 AXI_DMA_IN_DSCR_EMPTY_CH*n*_INT。(WT)

AXI_DMA_INFIFO_L1_OVF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_INFIFO_L1_OVF_CH*n*_INT。(WT)

AXI_DMA_INFIFO_L1_UDF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_INFIFO_L1_UDF_CH*n*_INT。(WT)

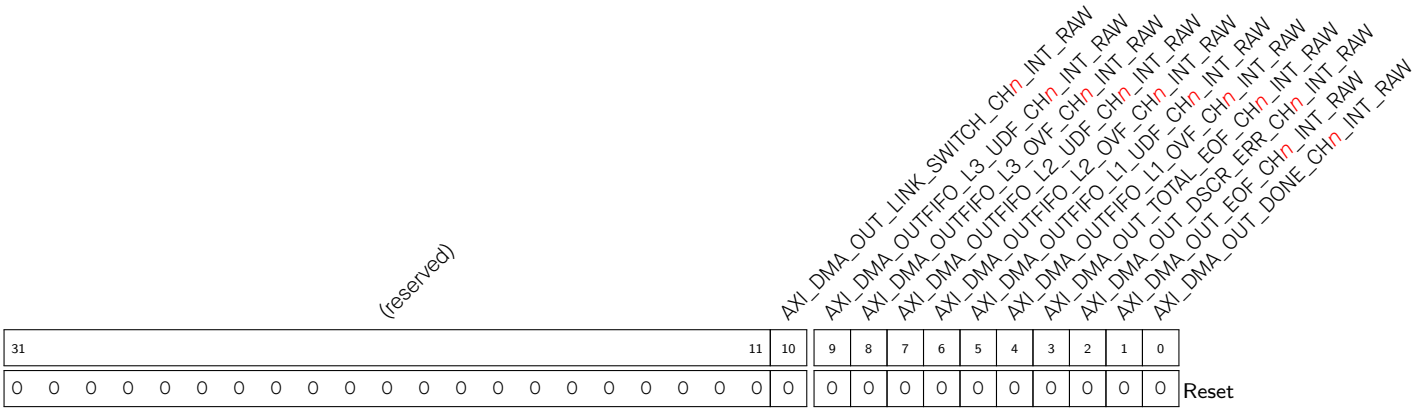
AXI_DMA_INFIFO_L2_OVF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_INFIFO_L2_OVF_CH*n*_INT。(WT)

AXI_DMA_INFIFO_L2_UDF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_INFIFO_L2_UDF_CH*n*_INT。(WT)

AXI_DMA_INFIFO_L3_OVF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_INFIFO_L3_OVF_CH*n*_INT。(WT)

AXI_DMA_INFIFO_L3_UDF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_INFIFO_L3_UDF_CH*n*_INT。(WT)

Register 6.73. AXI_DMA_OUT_INT_RAW_CH*n*_REG (*n*：0-2) (0x0138+0x68*n)



AXI_DMA_OUT_DONE_CH*n*_INT_RAW AXI_DMA_OUT_DONE_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUT_EOF_CH*n*_INT_RAW AXI_DMA_OUT_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_RAW AXI_DMA_OUT_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_RAW AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_RAW AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_RAW AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_RAW AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

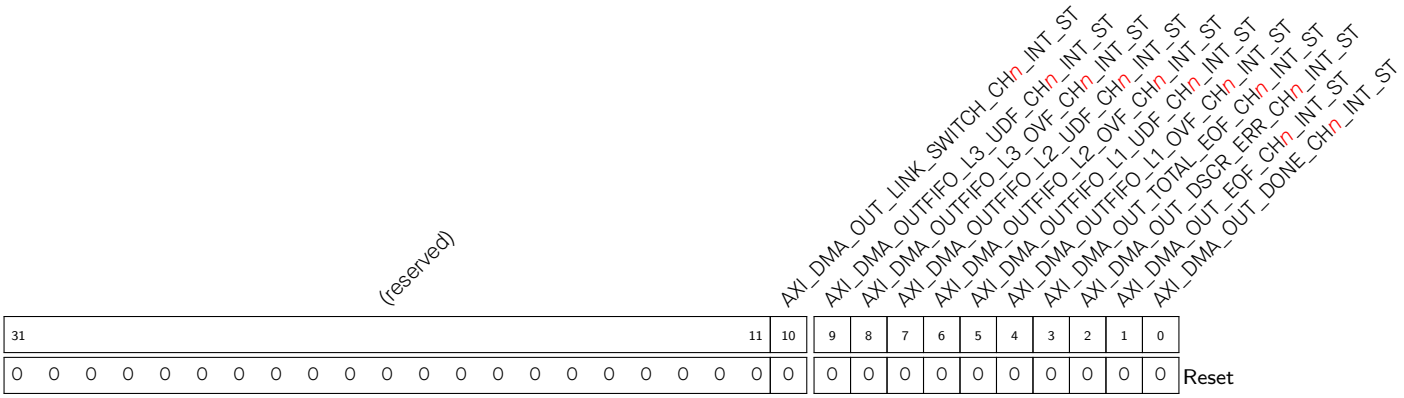
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_RAW AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_RAW AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_RAW AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_RAW AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 6.74. AXI_DMA_OUT_INT_ST_CH*n*_REG (*n*：0-2) (0x013C+0x68*n)



AXI_DMA_OUT_DONE_CH*n*_INT_ST AXI_DMA_OUT_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUT_EOF_CH*n*_INT_ST AXI_DMA_OUT_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_ST AXI_DMA_OUT_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ST AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_ST AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_ST AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_ST AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

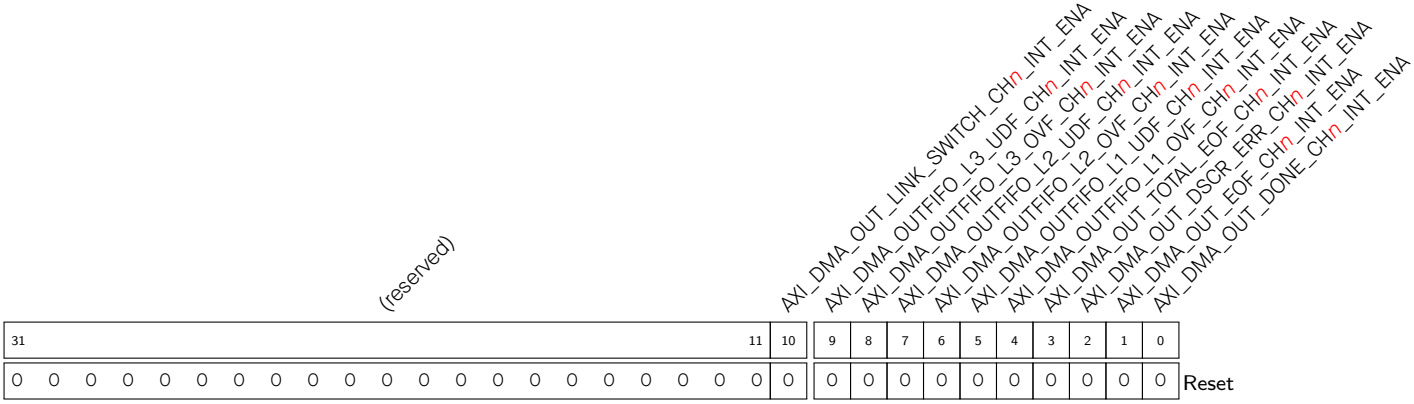
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_ST AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_ST AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_ST AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_ST AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT 的屏蔽中断状态。(RO)

Register 6.75. AXI_DMA_OUT_INT_ENA_CH*n*_REG (*n*：0-2) (0x0140+0x68*n)



AXI_DMA_OUT_DONE_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUT_DONE_CH*n*_INT。(R/W)

AXI_DMA_OUT_EOF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUT_EOF_CH*n*_INT。(R/W)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUT_DSCR_ERR_CH*n*_INT。(R/W)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT。(R/W)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT。(R/W)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT。(R/W)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT。(R/W)

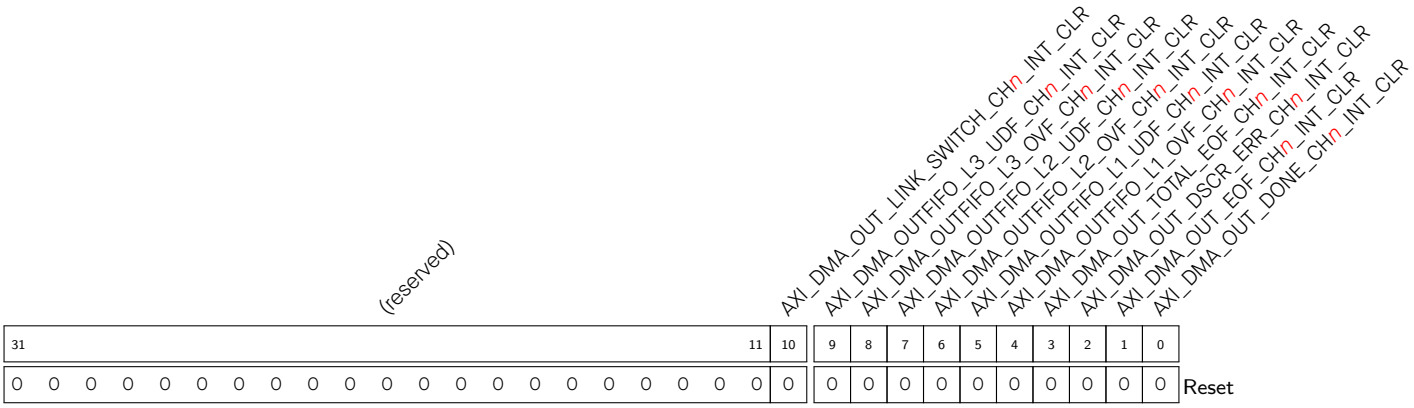
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT。(R/W)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT。(R/W)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT。(R/W)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_ENA 写 1 使能 AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT。(RO)

Register 6.76. AXI_DMA_OUT_INT_CLR_CH*n*_REG (*n*：0-2) (0x0144+0x68*n)



AXI_DMA_OUT_DONE_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUT_DONE_CH*n*_INT。(WT)

AXI_DMA_OUT_EOF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUT_EOF_CH*n*_INT。(WT)

AXI_DMA_OUT_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUT_DSCR_ERR_CH*n*_INT。(WT)

AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUT_TOTAL_EOF_CH*n*_INT。(WT)

AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUTFIFO_L1_OVF_CH*n*_INT。(WT)

AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUTFIFO_L1_UDF_CH*n*_INT。(WT)

AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUTFIFO_L2_OVF_CH*n*_INT。(WT)

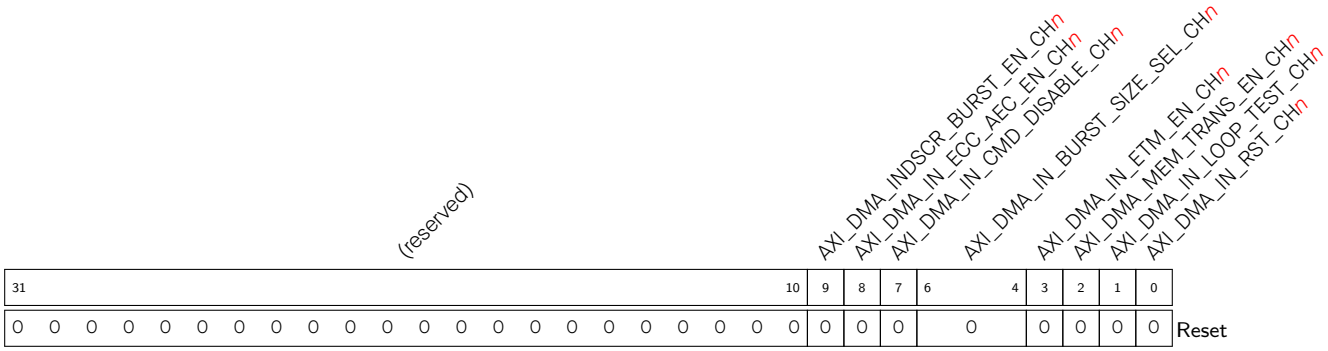
AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUTFIFO_L2_UDF_CH*n*_INT。(WT)

AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUTFIFO_L3_OVF_CH*n*_INT。(WT)

AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUTFIFO_L3_UDF_CH*n*_INT。(WT)

AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT_CLR 写 1 清除 AXI_DMA_OUT_LINK_SWITCH_CH*n*_INT。(RO)

Register 6.77. AXI_DMA_IN_CONFO_CHn_REG (n：0-2) (0x0010+0x68*n)



AXI_DMA_IN_RST_CHn 先写 1 再写 0 复位接收通道 *n* 的 FSM 和 RX FIFO 指针。(R/W)

AXI_DMA_IN_LOOP_TEST_CHn 保留。(R/W)

AXI_DMA_MEM_TRANS_EN_CHn 配置是否开启存储器到存储器的数据传输。

0：关闭

1：开启

(R/W)

AXI_DMA_IN_ETM_EN_CHn 配置是否开启接收通道 *n* 的 ETM 控制。

0：关闭

1：开启

(R/W)

AXI_DMA_IN_BURST_SIZE_SEL_CHn 配置接收通道 *n* 的突发长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

AXI_DMA_IN_CMD_DISABLE_CHn 配置是否禁用接收通道 *n* 上的命令。

0：开启

1：禁用

(R/W)

AXI_DMA_IN_ECC_AEC_EN_CHn 配置 AXI DMA 是否可以通过接收通道 *n* 访问外部存储器的 ECC 和 AES 区间。

0：不访问

1：访问

(R/W)

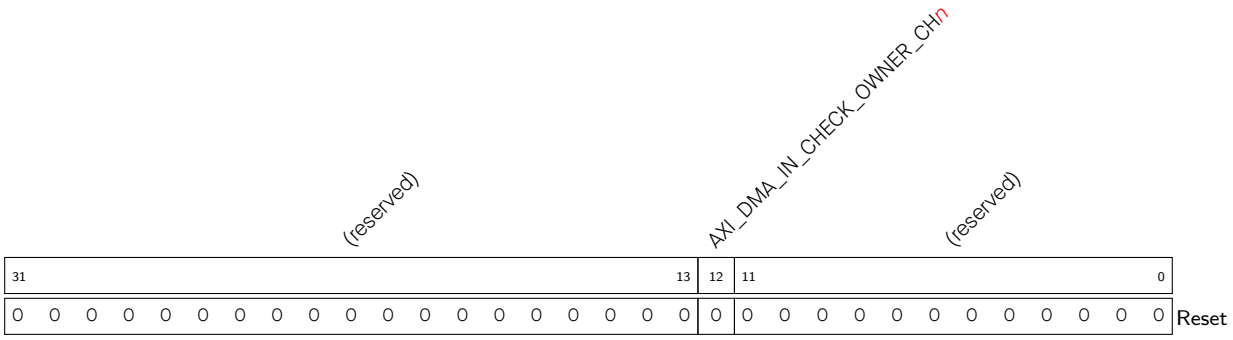
AXI_DMA_INDSCR_BURST_EN_CHn 配置是否开启 INCR 突发传输让接收通道 *n* 在访问内部存储器时读取描述符。

0：关闭

1：开启

(R/W)

Register 6.78. AXI_DMA_IN_CONF1_CH*n*_REG (*n*：0-2) (0x0014+0x68**n*)



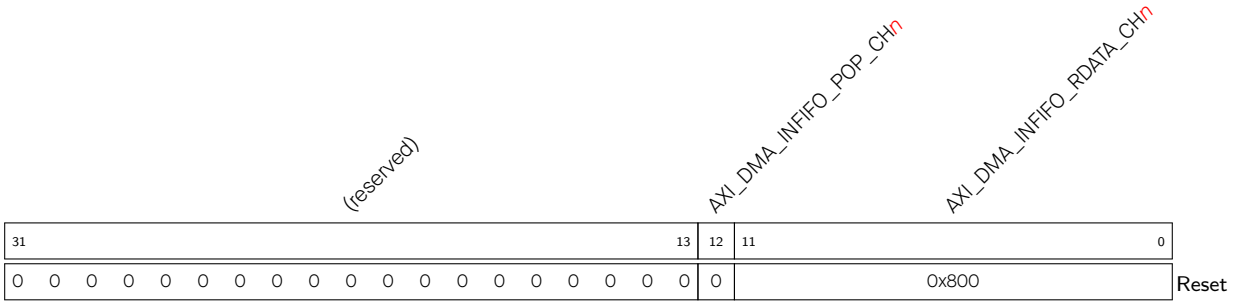
AXI_DMA_IN_CHECK_OWNER_CH*n* 配置是否开启接收通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

Register 6.79. AXI_DMA_IN_POP_CH*n*_REG (*n*：0-2) (0x001C+0x68**n*)



AXI_DMA_INFIFO_RDATA_CH*n* 表示从 AXI DMA RX FIFO 弹出的数据。(RO)

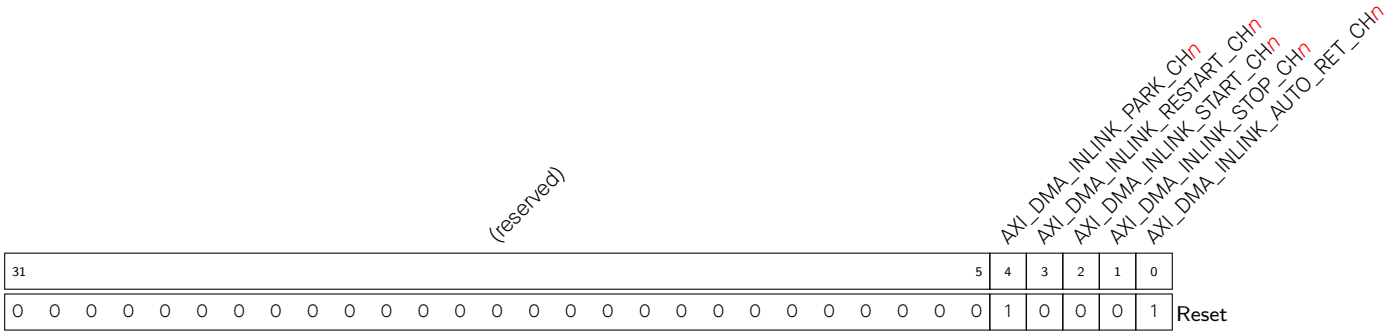
AXI_DMA_INFIFO_POP_CH*n* 配置是否从 AXI DMA RX FIFO 弹出数据。

0：无效值。没有作用

1：弹出

(WT)

Register 6.80. AXI_DMA_IN_LINK1_CH*n*_REG (*n*：0-2) (0x0020+0x68**n*)



AXI_DMA_INLINK_AUTO_RET_CH*n* 配置是否在接收数据有错时返回到当前接收链表描述符的地址。

0：不返回

1：返回

(R/W)

AXI_DMA_INLINK_STOP_CH*n* 配置接收通道 *n* 是否停止接收数据。

0：无效值。没有作用

1：停止

(WT)

AXI_DMA_INLINK_START_CH*n* 配置是否开启接收通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

AXI_DMA_INLINK_RESTART_CH*n* 配置是否重启接收通道 *n* 进行 AXI DMA 传输。

0：无效值。没有作用

1：重启

(WT)

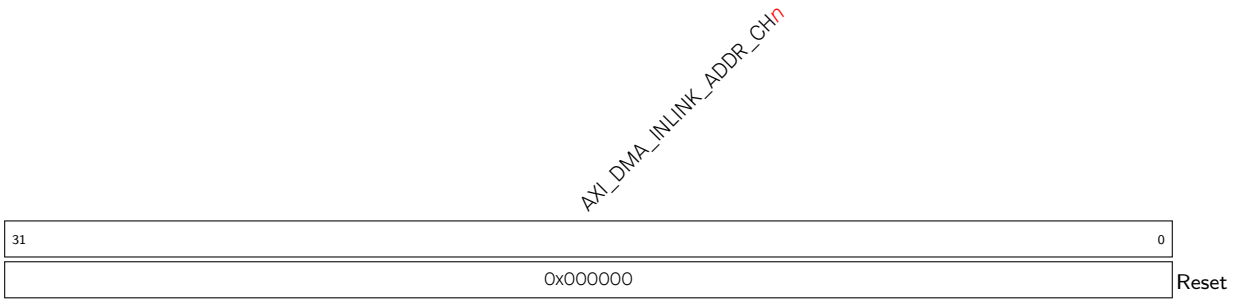
AXI_DMA_INLINK_PARK_CH*n* 表示接收链表描述符 FSM 的状态。

0：运行

1：空闲

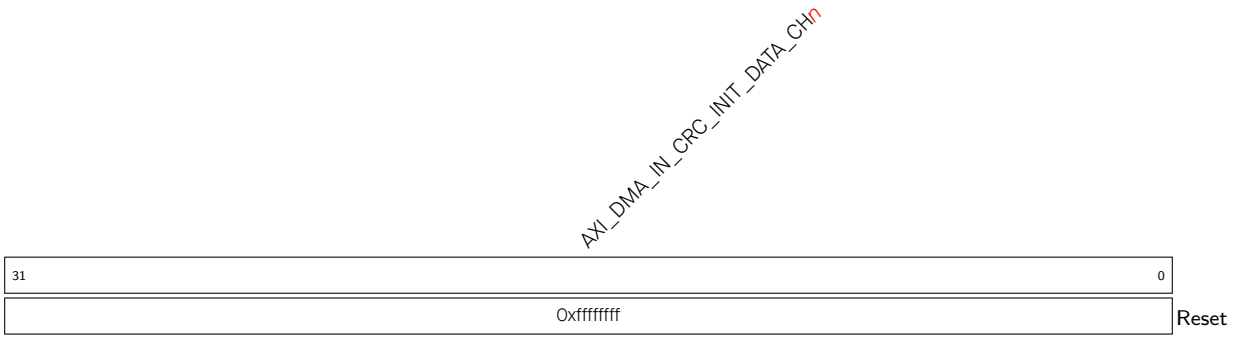
(RO)

Register 6.81. AXI_DMA_IN_LINK2_CH*n*_REG (*n*：0-2) (0x0024+0x68**n*)



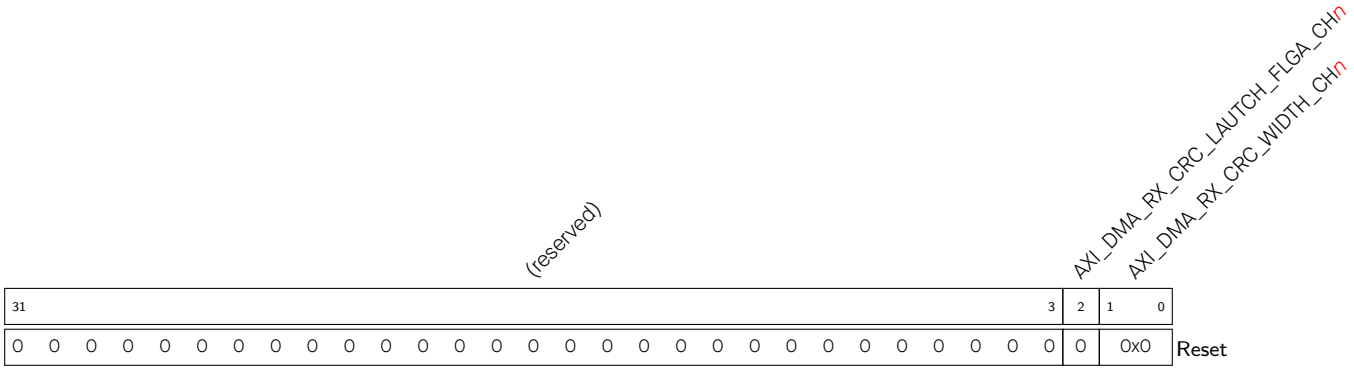
AXI_DMA_INLINK_ADDR_CH*n* 表示第一个接收链表描述符的地址。(R/W)

Register 6.82. AXI_DMA_IN_CRC_INIT_DATA_CH*n*_REG (*n*：0-2) (0x0048+0x68**n*)



AXI_DMA_IN_CRC_INIT_DATA_CH*n* 配置接收通道 *n* 的 CRC 初始值。(R/W)

Register 6.83. AXI_DMA_RX_CRC_WIDTH_CH*n*_REG (*n*：0-2) (0x004C+0x68**n*)



AXI_DMA_RX_CRC_WIDTH_CH*n* 配置接收通道 *n* 的 CRC 计算结果位宽。

0：≤ 8 位

1：≤ 16 位

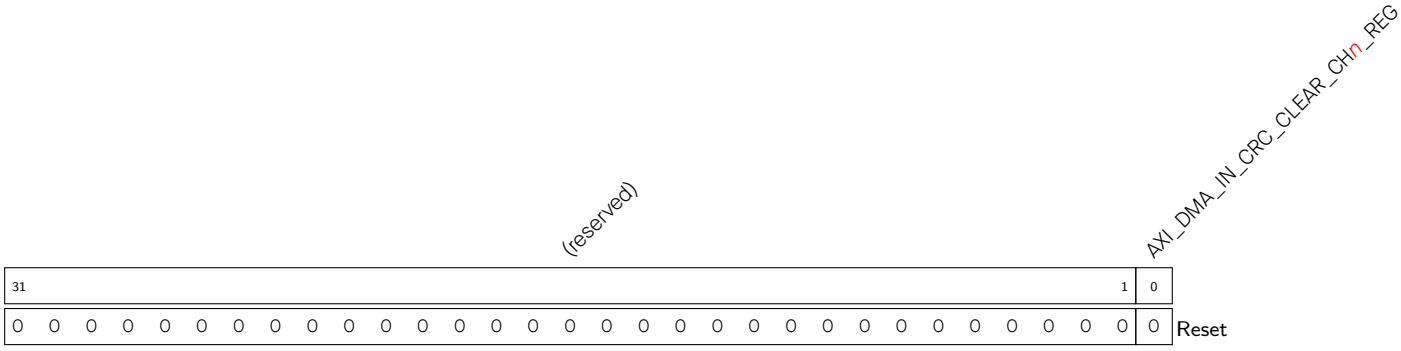
2：≤ 24 位

3：≤ 32 位

(R/W)

AXI_DMA_RX_CRC_LAUTCH_FLGA_CH*n* 先写 1 再写 0 锁存 AXI_DMA_RX_CRC_EN_ADDR_CH*n*、AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n*、AXI_DMA_RX_CRC_EN_WR_DATA_CH*n* 和 AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* 的值。(R/W)

Register 6.84. AXI_DMA_IN_CRC_CLEAR_CH*n*_REG (*n*：0-2) (0x0050+0x68**n*)



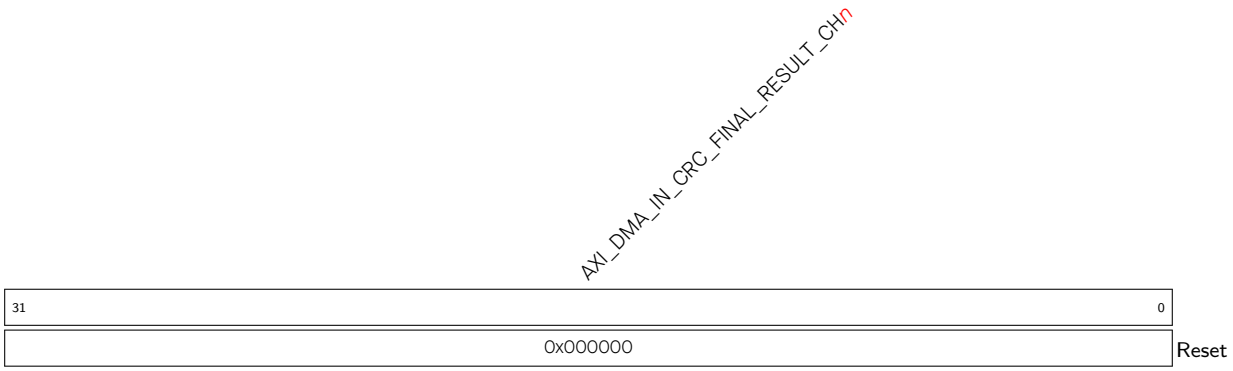
AXI_DMA_IN_CRC_CLEAR_CH*n*_REG 配置是否清除接收通道 *n* 的 CRC 计算结果。

0：不清除

1：清除

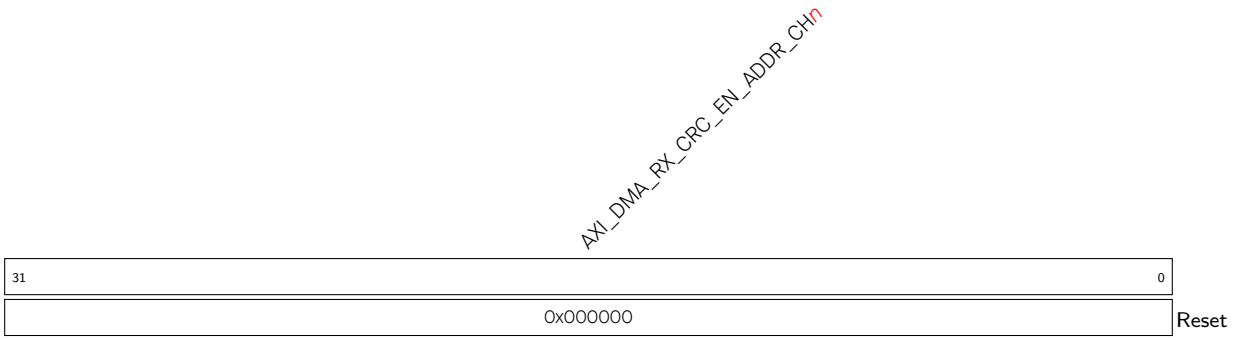
(R/W)

Register 6.85. AXI_DMA_IN_CRC_FINAL_RESULT_CH*n*_REG (*n*：0-2) (0x0054+0x68**n*)



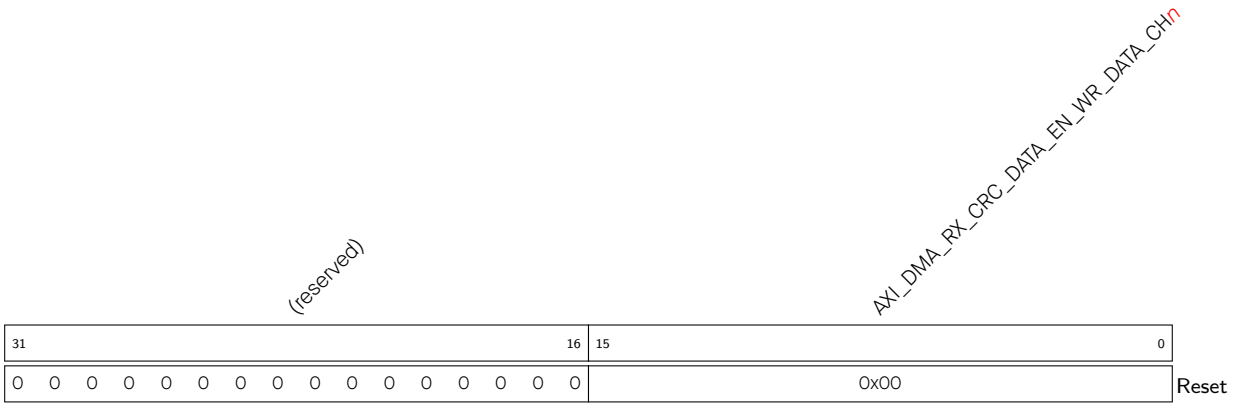
AXI_DMA_IN_CRC_FINAL_RESULT_CH*n* 表示接收通道 *n* 的 CRC 计算结果。(RO)

Register 6.86. AXI_DMA_RX_CRC_EN_ADDR_CH*n*_REG (*n*：0-2) (0x005C+0x68**n*)



AXI_DMA_RX_CRC_EN_ADDR_CH*n* 配置发送通道 *n* AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

Register 6.87. AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*：0-2) (0x0060+0x68**n*)



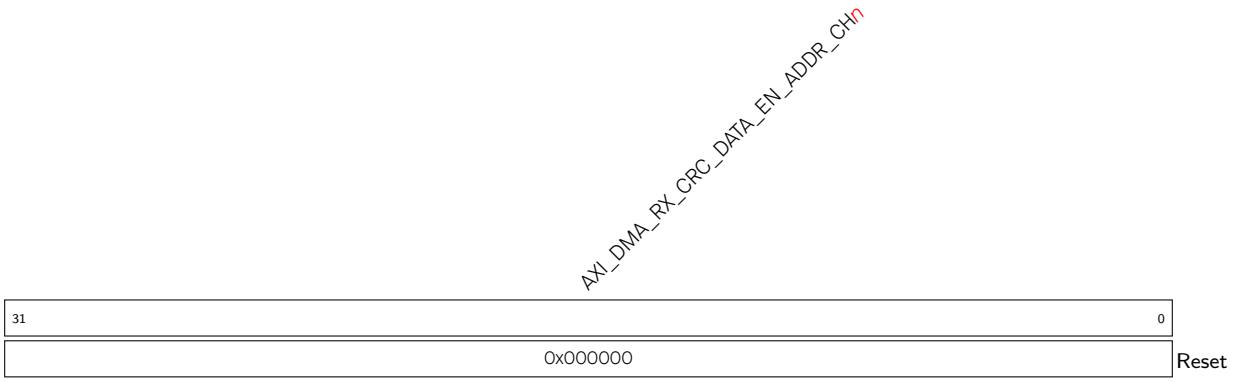
AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n* 配置发送通道 *n* 中间结果的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

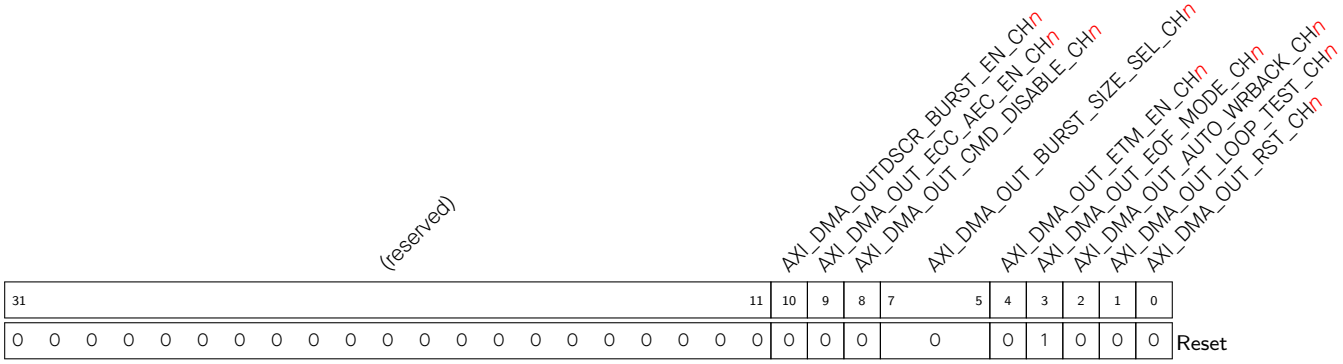
(R/W)

Register 6.88. AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*：0-2) (0x0064+0x68**n*)



AXI_DMA_RX_CRC_DATA_EN_ADDR_CH*n* 配置发送通道 *n* AXI_DMA_RX_CRC_DATA_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

Register 6.89. AXI_DMA_OUT_CONF0_CH*n*_REG(*n*：0-2) (0x0148+0x68*n)



AXI_DMA_OUT_RST_CH*n* 配置发送通道 *n* FSM 和 TX FIFO 指针的复位状态。

0：复位释放

1：复位

(R/W)

AXI_DMA_OUT_LOOP_TEST_CH*n* 保留。(R/W)

AXI_DMA_OUT_AUTO_WRBACK_CH*n* 配置在发送完 TX FIFO 中的所有数据时是否开启发送链表自动回写。

0：关闭

1：开启

(R/W)

AXI_DMA_OUT_EOF_MODE_CH*n* 配置生成 EOF 标志的时间。

0：待发送数据推入 AXI DMA FIFO 时生成发送通道 *n* 的 EOF 标志。

1：待发送数据从 AXI DMA FIFO 中弹出时生成发送通道 *n* 的 EOF 标志。

(R/W)

AXI_DMA_OUT_ETM_EN_CH*n* 配置发送通道 *n* 是否开启 ETM 控制。

0：关闭

1：开启

((R/W)

AXI_DMA_OUT_BURST_SIZE_SEL_CH*n* 配置发送通道 *n* 的突发传输长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

AXI_DMA_OUT_CMD_DISABLE_CH*n* 配置是否禁用发送通道 *n* 上的命令。

0：开启

1：禁用

(R/W)

见下页...

Register 6.89. AXI_DMA_OUT_CONF0_CH*n*_REG(*n*：0-2) (0x0148+0x68***n*)

接上页...

AXI_DMA_OUT_ECC_AEC_EN_CH*n* 配置 AXI DMA 是否可以通过发送通道 *n* 访问外部存储器的 ECC 和 AES 区间。

0：不访问

1：访问

(R/W)

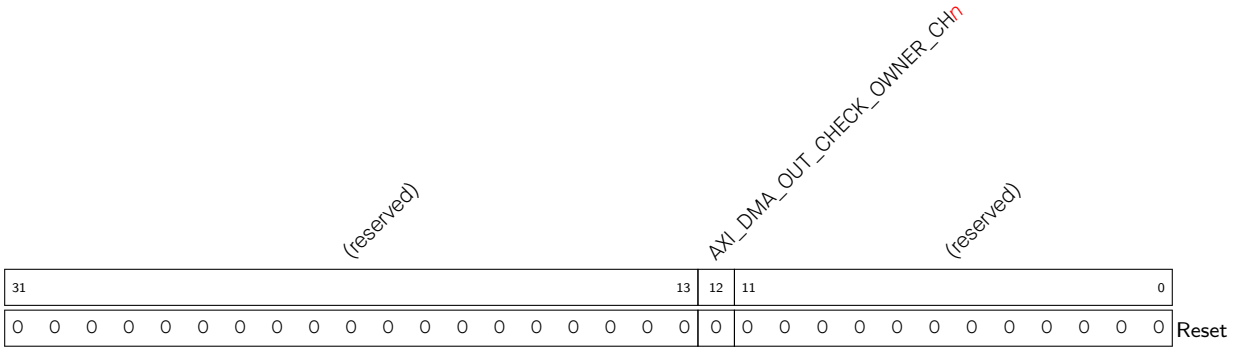
AXI_DMA_OUTDSCR_BURST_EN_CH*n* 配置是否开启 INCR 突发传输，让发送通道 *n* 读取描述符。

0：关闭

1：开启

(R/W)

Register 6.90. AXI_DMA_OUT_CONF1_CH*n*_REG (*n*：0-2) (0x014C+0x68***n*)



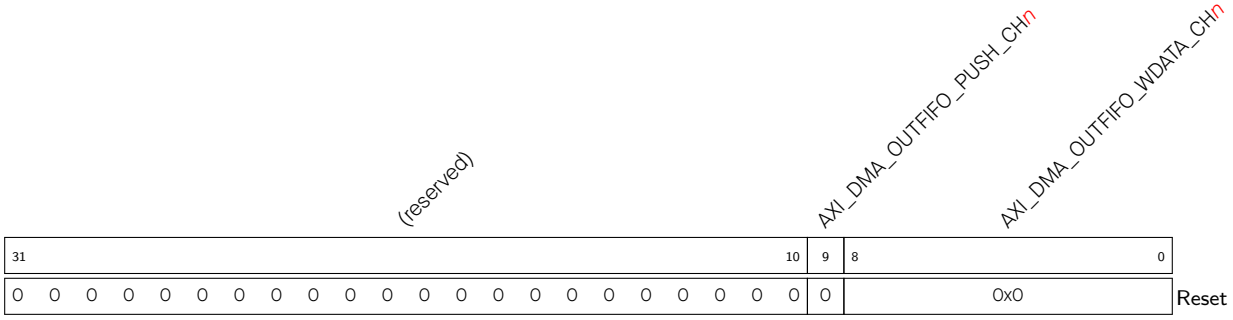
AXI_DMA_OUT_CHECK_OWNER_CH*n* 配置发送通道 *n* 是否开启 owner 位检查。

0：关闭

1：开启

(R/W)

Register 6.91. AXI_DMA_OUT_PUSH_CH*n*_REG (*n*：0-2) (0x0154+0x68**n*)



AXI_DMA_OUTFIFO_WDATA_CH*n* 表示需要推入 AXI DMA TX FIFO 的数据。(R/W)

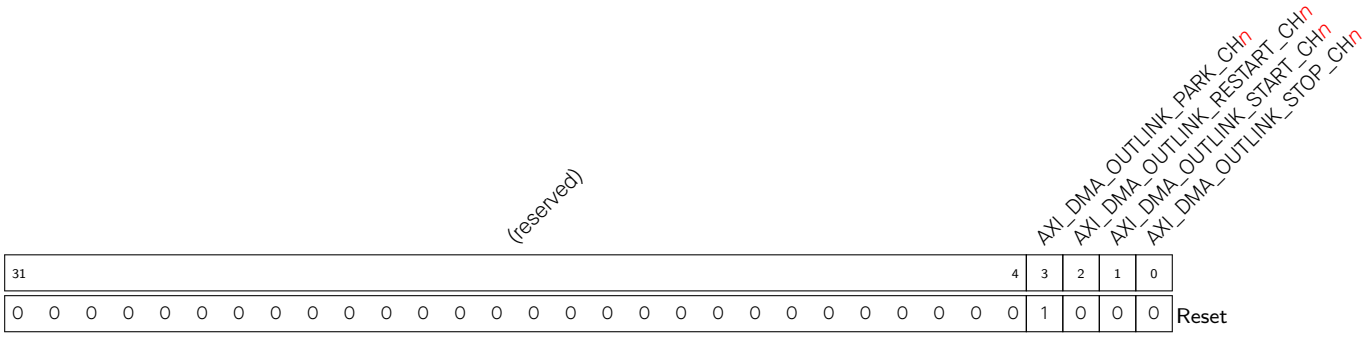
AXI_DMA_OUTFIFO_PUSH_CH*n* 配置是否将数据推入 AXI DMA TX FIFO。

0：无效值。没有作用

1：推送

(WT)

Register 6.92. AXI_DMA_OUT_LINK1_CH*n*_REG (*n*：0-2) (0x0158+0x68**n*)



AXI_DMA_OUTLINK_STOP_CH*n* 配置是否停止发送通道 *n* 的数据传输。

0：无效值。没有作用

1：停止

(WT)

AXI_DMA_OUTLINK_START_CH*n* 配置是否开启发送通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

AXI_DMA_OUTLINK_RESTART_CH*n* 配置是否重启发送通道 *n* 进行 AXI DMA 传输。

0：无效值。没有作用

1：重启

(WT)

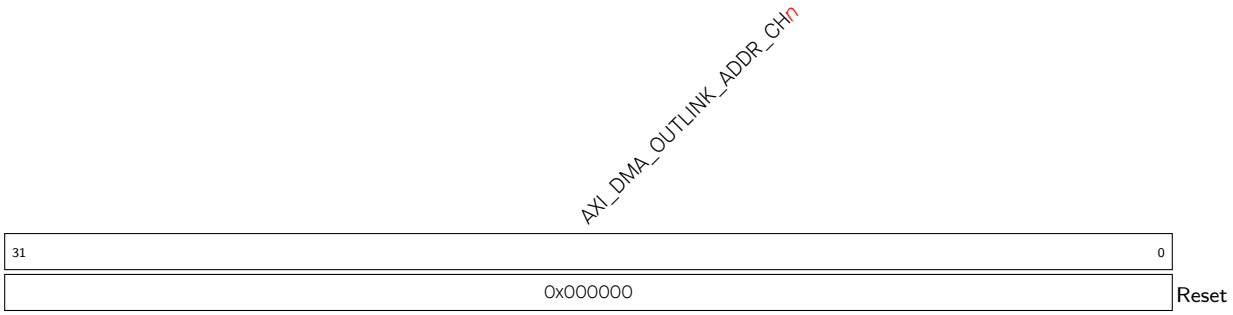
AXI_DMA_OUTLINK_PARK_CH*n* 表示发送链表描述符的 FSM 的状态。

0：运行

1：空闲

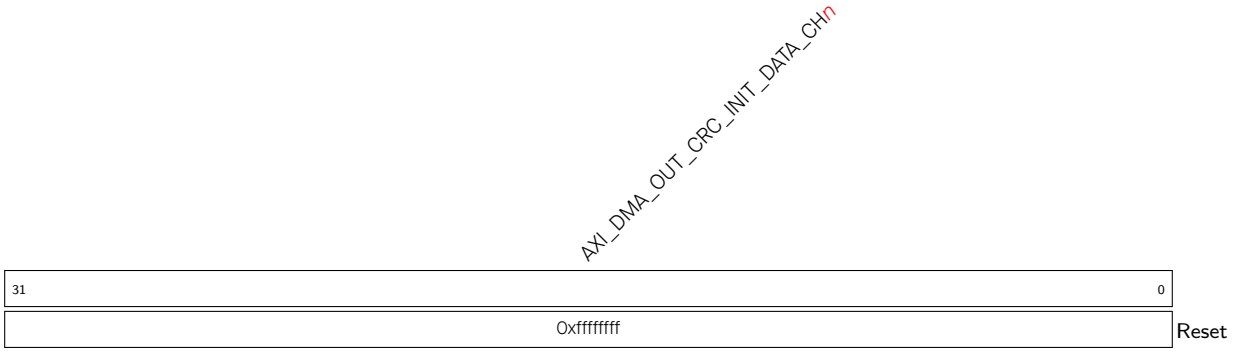
(RO)

Register 6.93. AXI_DMA_OUT_LINK2_CH*n*_REG (*n*：0-2) (0x015C+0x68**n*)



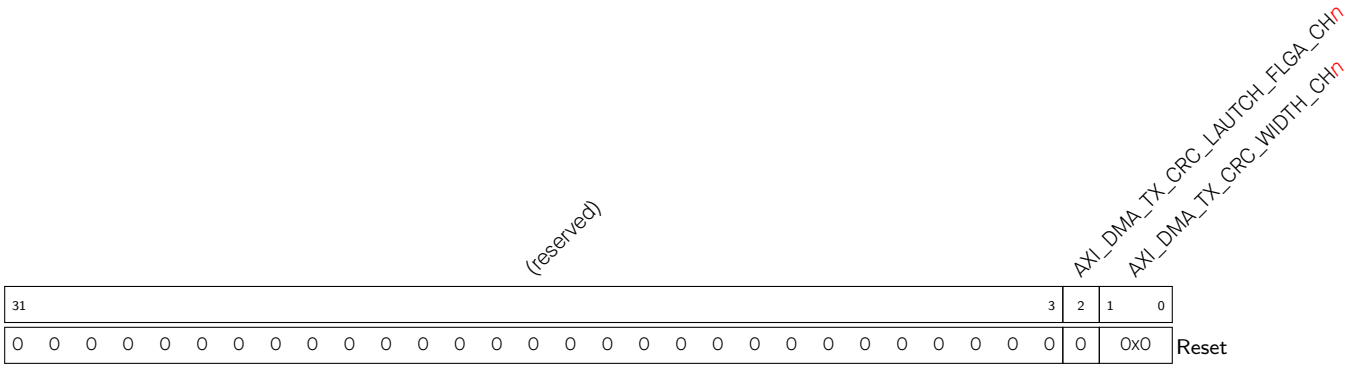
AXI_DMA_OUTLINK_ADDR_CH*n* 表示第一个发送链表描述符的地址。(R/W)

Register 6.94. AXI_DMA_OUT_CRC_INIT_DATA_CH*n*_REG (*n*：0-2) (0x0180+0x68**n*)



AXI_DMA_OUT_CRC_INIT_DATA_CH*n* 配置发送通道 *n* 的 CRC 初始值。(R/W)

Register 6.95. AXI_DMA_TX_CRC_WIDTH_CH*n*_REG (*n*：0-2) (0x0184+0x68**n*)



AXI_DMA_TX_CRC_WIDTH_CH*n* 配置发送通道 *n* 的 CRC 计算结果位宽。

0：≤ 8 位

1：≤ 16 位

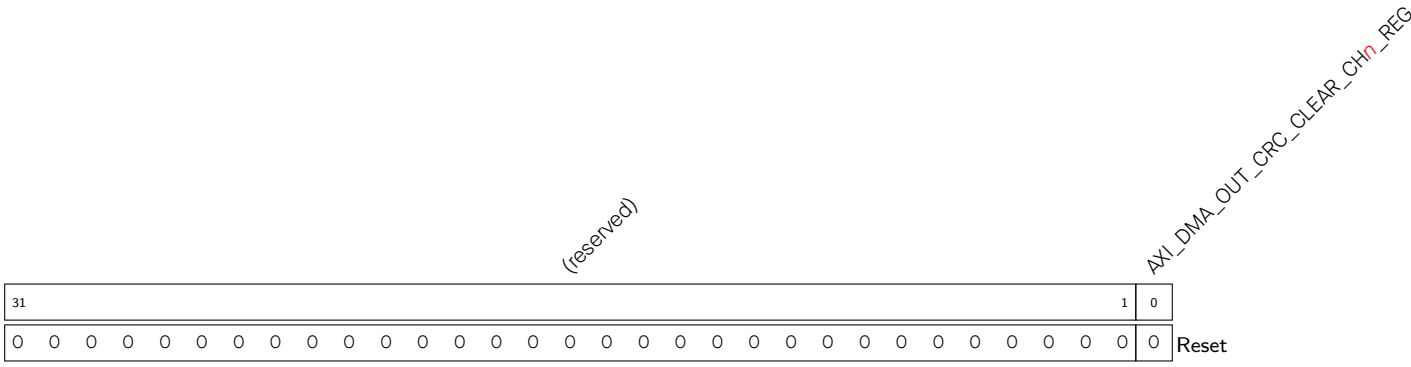
2：≤ 24 位

3：≤ 32 位

(R/W)

AXI_DMA_TX_CRC_LAUTCH_FLGA_CH*n* 先写 1 再写 0 锁存 AXI_DMA_TX_CRC_EN_ADDR_CH*n*、AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*、AXI_DMA_TX_CRC_EN_WR_DATA_CH*n* 和 AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* 的值。(R/W)

Register 6.96. AXI_DMA_OUT_CRC_CLEAR_CH*n*_REG (*n*：0-2) (0x0188+0x68**n*)



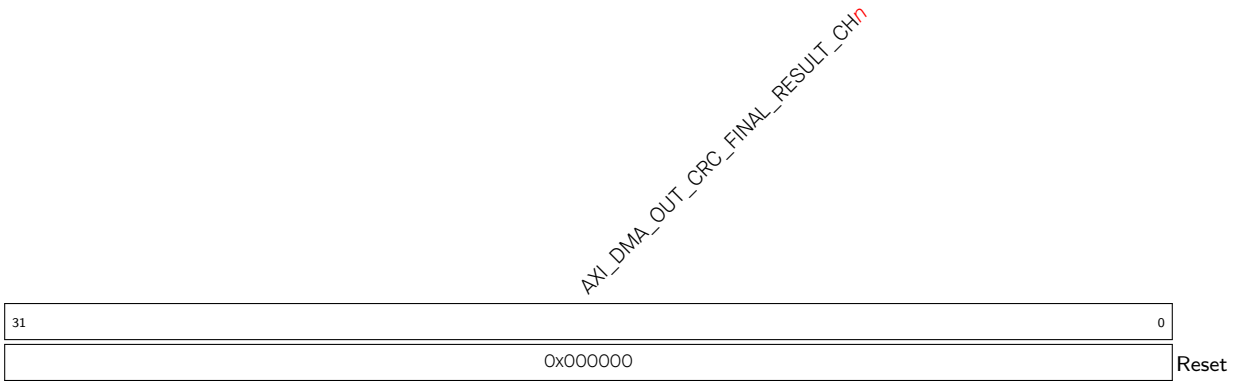
AXI_DMA_OUT_CRC_CLEAR_CH*n*_REG 配置是否清除发送通道 *n* 的 CRC 计算结果。

0：不清除

1：清除

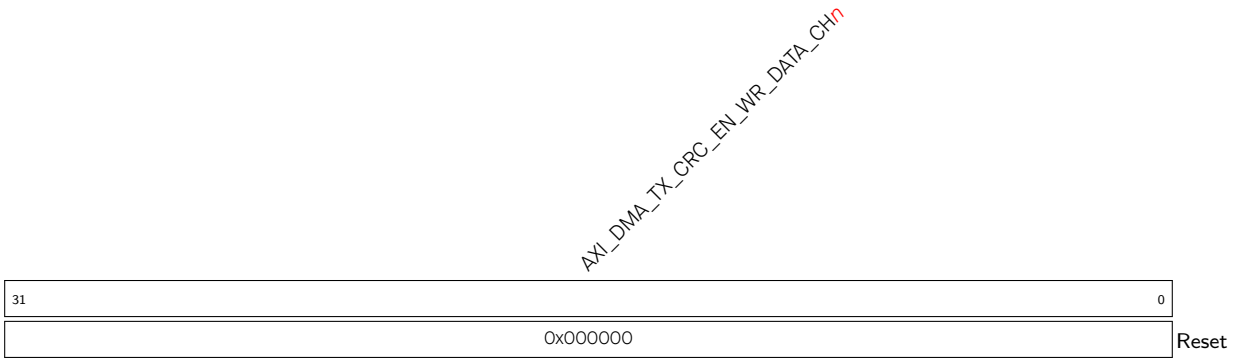
(R/W)

Register 6.97. AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n*_REG (*n*：0-2) (0x018C+0x68**n*)



AXI_DMA_OUT_CRC_FINAL_RESULT_CH*n* 表示发送通道 *n* 的 CRC 计算结果。(RO)

Register 6.98. AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG (*n*：0-2) (0x0190+0x68**n*)



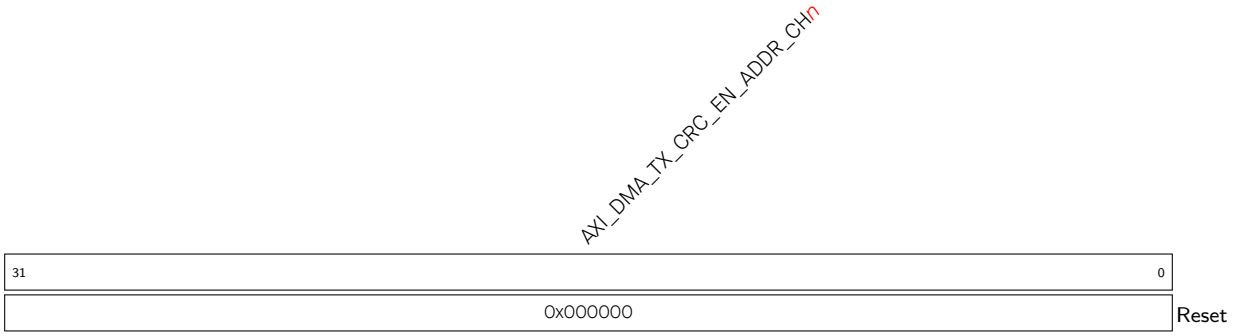
AXI_DMA_TX_CRC_EN_WR_DATA_CH*n* 配置发送通道 *n* 中间结果的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

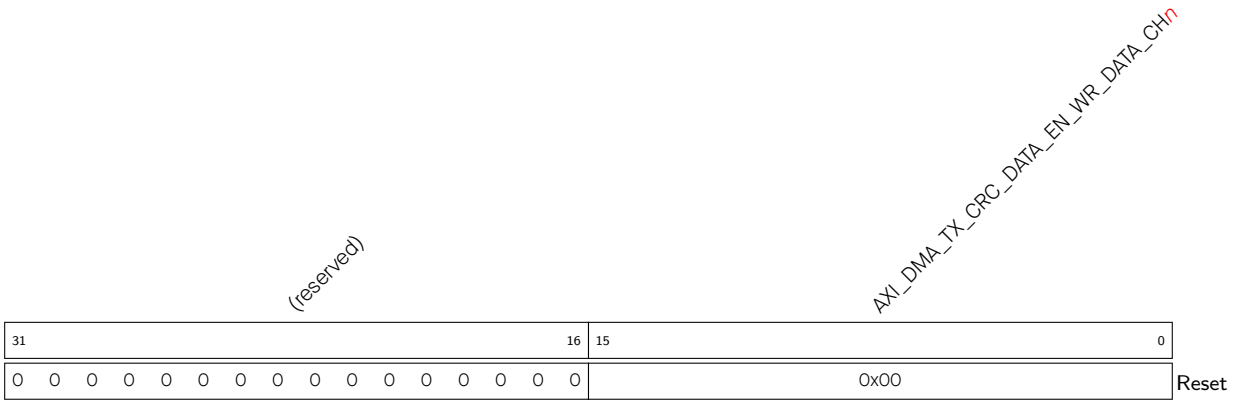
(R/W)

Register 6.99. AXI_DMA_TX_CRC_EN_ADDR_CH*n*_REG (*n*：0-2) (0x0194+0x68**n*)



AXI_DMA_TX_CRC_EN_ADDR_CH*n* 配置发送通道 *n* AXI_DMA_TX_CRC_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

Register 6.100. AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG (*n*：0-2) (0x0198+0x68**n*)



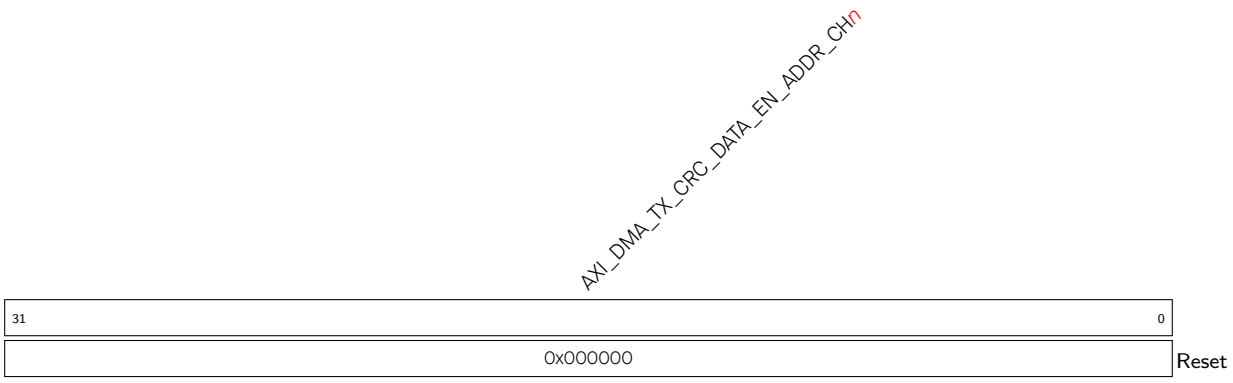
AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n* 配置发送通道 *n* 数据的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

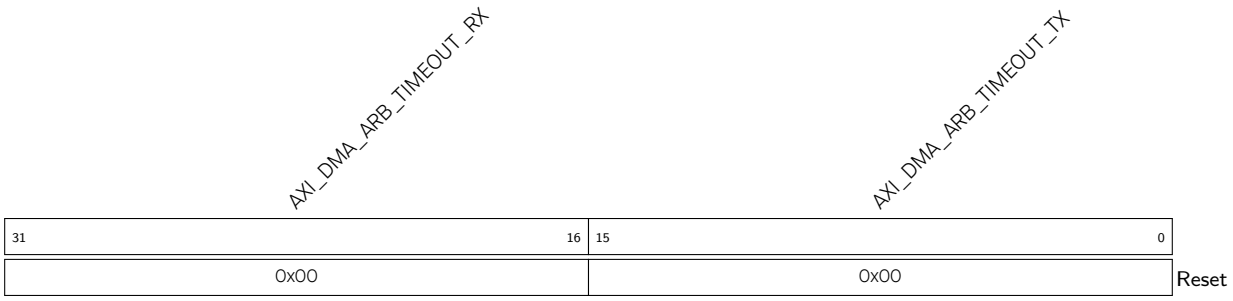
(R/W)

Register 6.101. AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n*_REG (*n*：0-2) (0x019C+0x68**n*)



AXI_DMA_TX_CRC_DATA_EN_ADDR_CH*n* 配置发送通道 *n* AXI_DMA_TX_CRC_DATA_EN_WR_DATA_CH*n*_REG 寄存器的值适用于 CRC 计算结果的哪一位。(R/W)

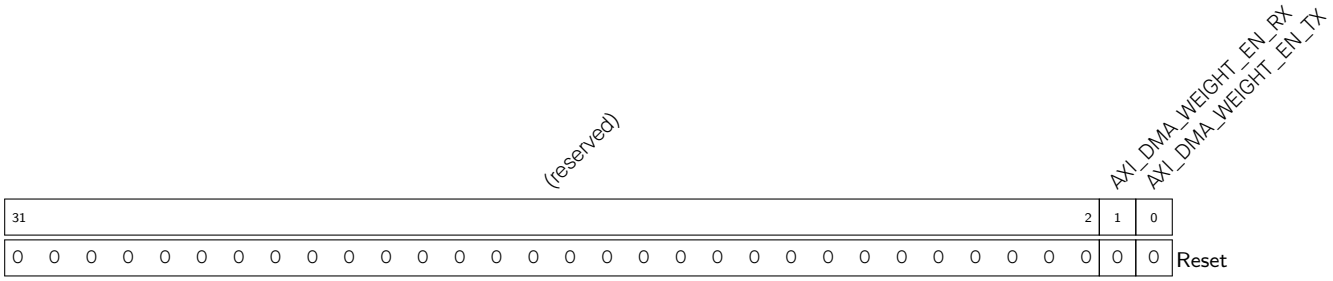
Register 6.102. AXI_DMA_ARB_TIMEOUT_REG (0x0270)



AXI_DMA_ARB_TIMEOUT_TX 配置 TX 方向的时间片。单位：AXI 总线时钟周期。(R/W)

AXI_DMA_ARB_TIMEOUT_RX 配置 RX 方向的时间片。单位：AXI 总线时钟周期。(R/W)

Register 6.103. AXI_DMA_WEIGHT_EN_REG (0x0274)



AXI_DMA_WEIGHT_EN_TX 配置是否开启 TX 方向的权重仲裁。

0：关闭

1：开启

(R/W)

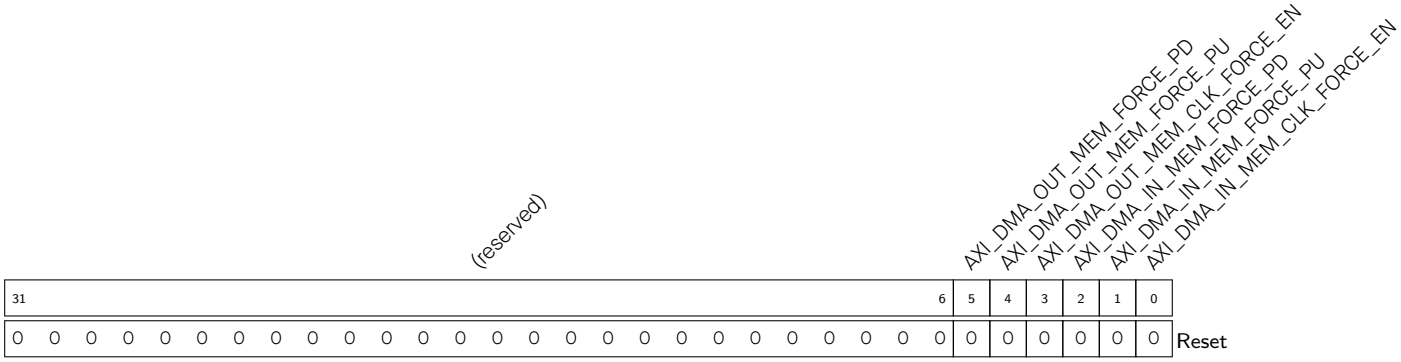
AXI_DMA_WEIGHT_EN_RX 配置是否开启 RX 方向的权重仲裁。

0：关闭

1：开启

(R/W)

Register 6.104. AXI_DMA_IN_MEM_CONF_REG (0x0278)



AXI_DMA_IN_MEM_CLK_FORCE_EN 配置 RX 方向的存储器时钟门控。

0：仅在 AXI DMA 访问存储器时支持时钟

1：始终强制开启时钟

(R/W)

AXI_DMA_IN_MEM_FORCE_PU 配置是否强制为 RX 方向的存储器上电。

0：不强制上电

1：强制上电

(R/W)

AXI_DMA_IN_MEM_FORCE_PD 配置是否强制为 RX 方向的存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

AXI_DMA_OUT_MEM_CLK_FORCE_EN 配置 TX 方向的存储器时钟门控。

0：仅在 AXI DMA 访问存储器时支持时钟

1：始终强制开启时钟

(R/W)

AXI_DMA_OUT_MEM_FORCE_PU 配置是否强制为 TX 方向的存储器上电。

0：不强制上电

1：强制上电

(R/W)

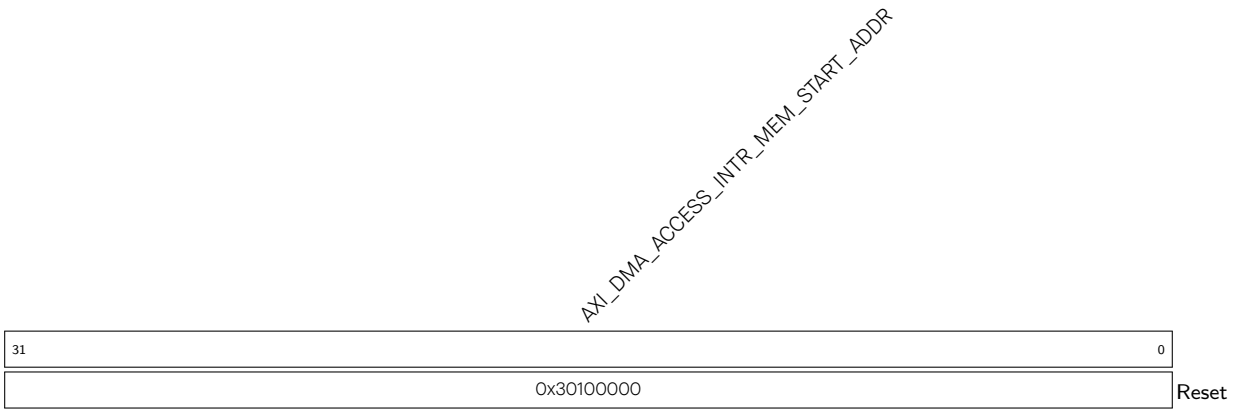
AXI_DMA_OUT_MEM_FORCE_PD 配置是否强制为 TX 方向的存储器掉电。

0：不强制掉电

1：强制掉电

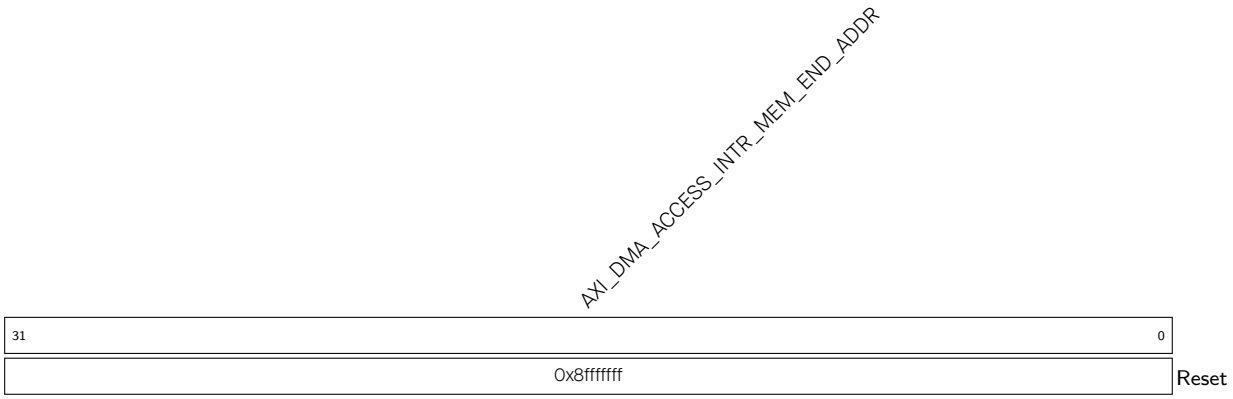
(R/W)

Register 6.105. AXI_DMA_INTR_MEM_START_ADDR_REG (0x027C)



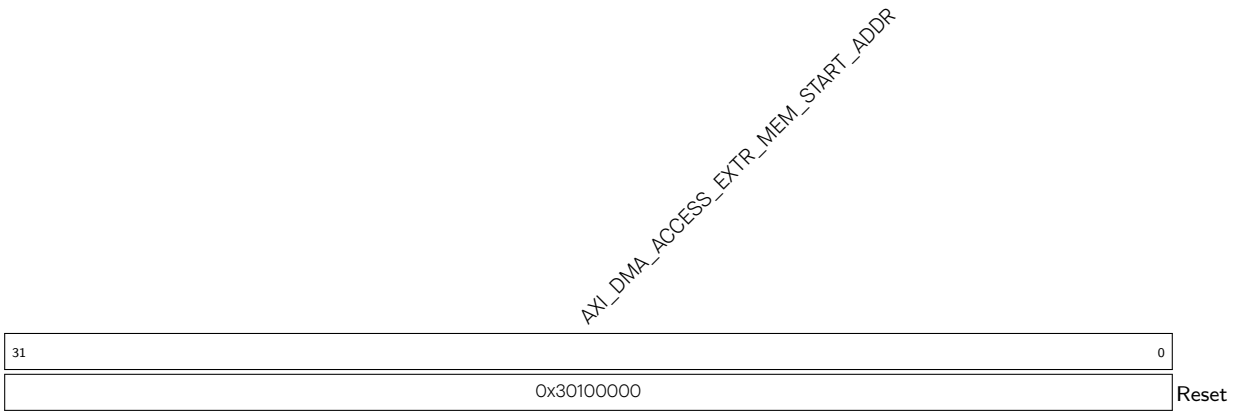
AXI_DMA_ACCESS_INTR_MEM_START_ADDR 配置可访问内部存储器地址空间的起始地址。(R/W)

Register 6.106. AXI_DMA_INTR_MEM_END_ADDR_REG (0x0280)



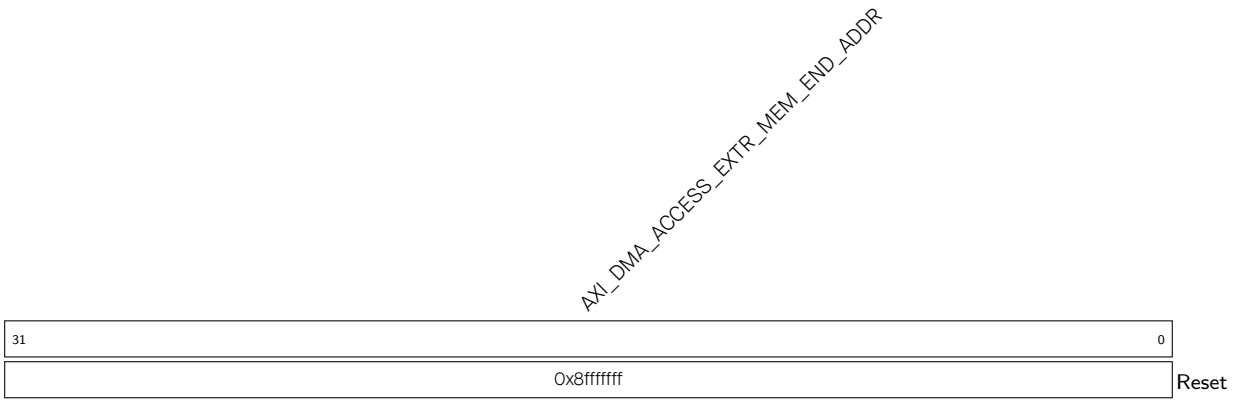
AXI_DMA_ACCESS_INTR_MEM_END_ADDR 配置可访问内部存储器地址空间的结束地址。(R/W)

Register 6.107. AXI_DMA_EXTR_MEM_START_ADDR_REG (0x0284)



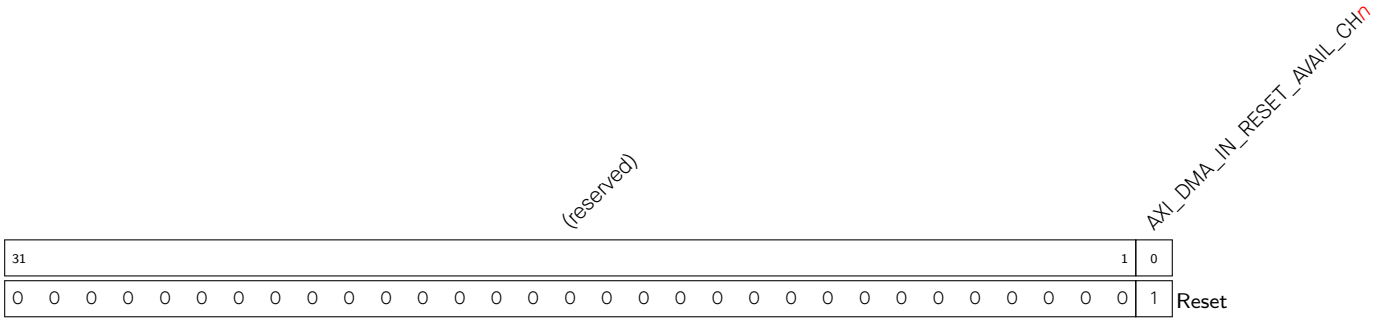
AXI_DMA_ACCESS_EXTR_MEM_START_ADDR 配置可访问外部存储器地址空间的起始地址。(R/W)

Register 6.108. AXI_DMA_EXTR_MEM_END_ADDR_REG (0x0288)



AXI_DMA_ACCESS_EXTR_MEM_END_ADDR 配置可访问外部存储器地址空间的结束地址。(R/W)

Register 6.109. AXI_DMA_IN_RESET_AVAIL_CH*n*_REG (*n*: 0-2) (0x028C+0x4**n*)



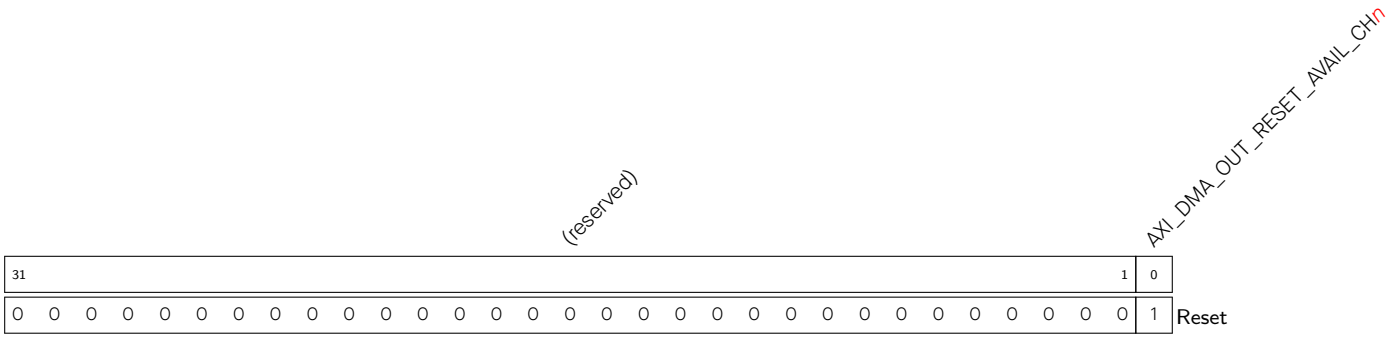
AXI_DMA_IN_RESET_AVAIL_CH*n* 表示接收通道 *n* 的是否允许复位。

0: 不可以进行软件复位

1: 允许复位

(RO)

Register 6.110. AXI_DMA_OUT_RESET_AVAIL_CH*n*_REG (*n*：0-2) (0x0298+0x4***n*)



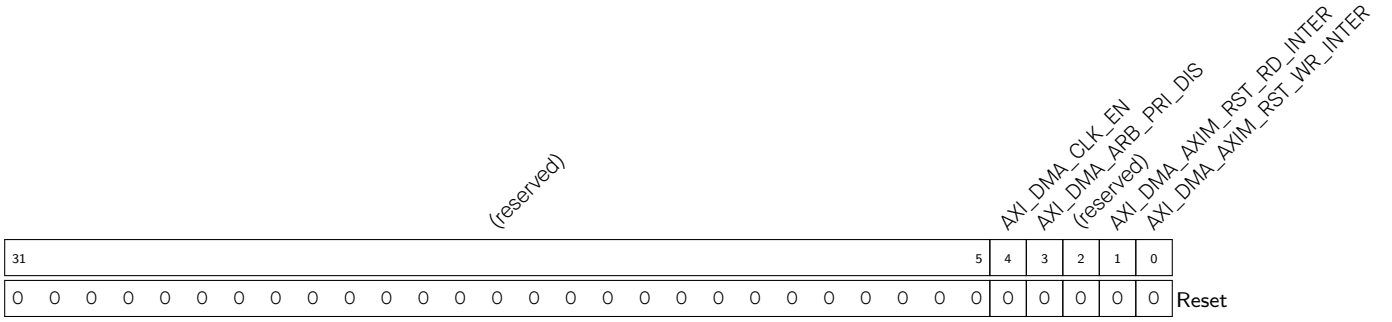
AXI_DMA_OUT_RESET_AVAIL_CH*n* 表示发送通道 *n* 是否允许复位。

0：不可以进行软件复位

1：允许复位

(RO)

Register 6.111. AXI_DMA_MISC_CONF_REG (0x02A8)



AXI_DMA_AXIM_RST_WR_INTER 先写 1 再写 0 复位内部 AXI 写 FSM。(R/W)

AXI_DMA_AXIM_RST_RD_INTER 先写 1 再写 0 复位内部 AXI 读 FSM。(R/W)

AXI_DMA_ARB_PRI_DIS 配置是否禁用优先级仲裁。

0：开启

1：禁用

(R/W)

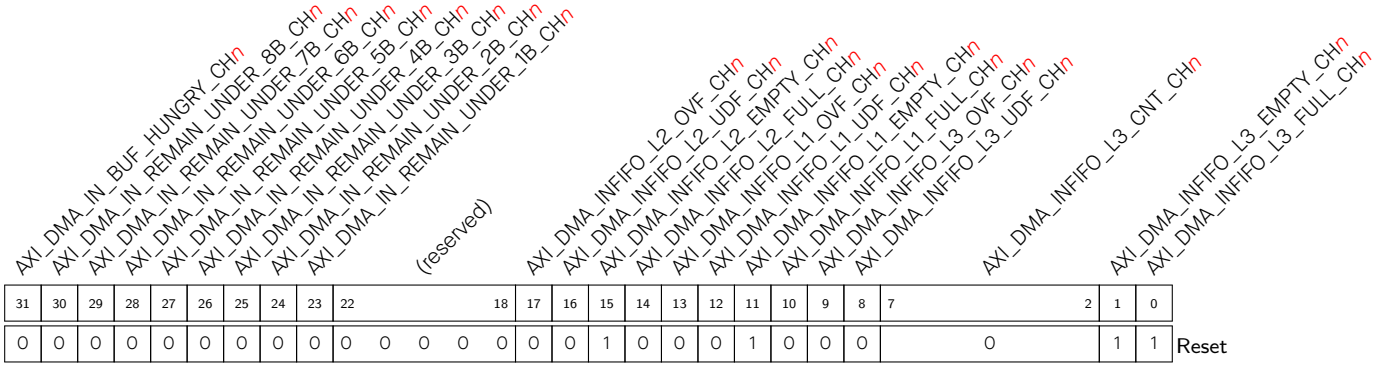
AXI_DMA_CLK_EN 配置 AXI DMA 时钟门控。

0：仅在应用程序写入寄存器时支持时钟。

1：始终强制打开寄存器的时钟。

(R/W)

Register 6.112. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*：0-2) (0x0018+0x68**n*)



AXI_DMA_INFIFO_L3_FULL_CH*n* 表示 L3 RX FIFO 是否已满。

0：未满

1：已满

(RO)

AXI_DMA_INFIFO_L3_EMPTY_CH*n* 表示 L3 RX FIFO 是否为空。

0：不为空

1：为空

(RO)

AXI_DMA_INFIFO_L3_CNT_CH*n* 表示 L3 RX FIFO 中的数据字节数。(RO)

AXI_DMA_INFIFO_L3_UDF_CH*n* 表示 L3 RX FIFO 是否下溢。

0：无下溢

1：下溢

(RO)

AXI_DMA_INFIFO_L3_OVF_CH*n* 表示 L3 RX FIFO 是否溢出。

0：无溢出

1：溢出

(RO)

AXI_DMA_INFIFO_L1_FULL_CH*n* 表示 L1 RX FIFO 是否已满。

0：未满

1：已满

(RO)

AXI_DMA_INFIFO_L1_EMPTY_CH*n* 表示 L1 RX FIFO 是否为空。

0：不为空

1：为空

(RO)

AXI_DMA_INFIFO_L1_UDF_CH*n* 表示 L1 RX FIFO 是否下溢。

0：无下溢

1：下溢

(RO)

见下页...

Register 6.112. AXI_DMA_INFIFO_STATUS_CH*n*_REG (*n*：0-2) (0x0018+0x68**n*)

接上页...

AXI_DMA_INFIFO_L1_OVF_CH*n* 表示 L1 RX FIFO 是否溢出。

0：无溢出

1：溢出

(RO)

AXI_DMA_INFIFO_L2_FULL_CH*n* 表示 L2 RX FIFO 是否已满。

0：未满

1：已满

(RO)

AXI_DMA_INFIFO_L2_EMPTY_CH*n* 表示 L1 RX FIFO 是否为空。

0：不为空

1：为空

(RO)

AXI_DMA_INFIFO_L2_UDF_CH*n* 表示 L2 RX FIFO 是否下溢。

0：无下溢

1：下溢

(RO)

AXI_DMA_INFIFO_L2_OVF_CH*n* 表示 L2 RX FIFO 是否溢出。

0：无溢出

1：溢出

(RO)

AXI_DMA_IN_REMAIN_UNDER_1B_CH*n* 保留。(RO)

AXI_DMA_IN_REMAIN_UNDER_2B_CH*n* 保留。(RO)

AXI_DMA_IN_REMAIN_UNDER_3B_CH*n* 保留。(RO)

AXI_DMA_IN_REMAIN_UNDER_4B_CH*n* 保留。(RO)

AXI_DMA_IN_REMAIN_UNDER_5B_CH*n* 保留。(RO)

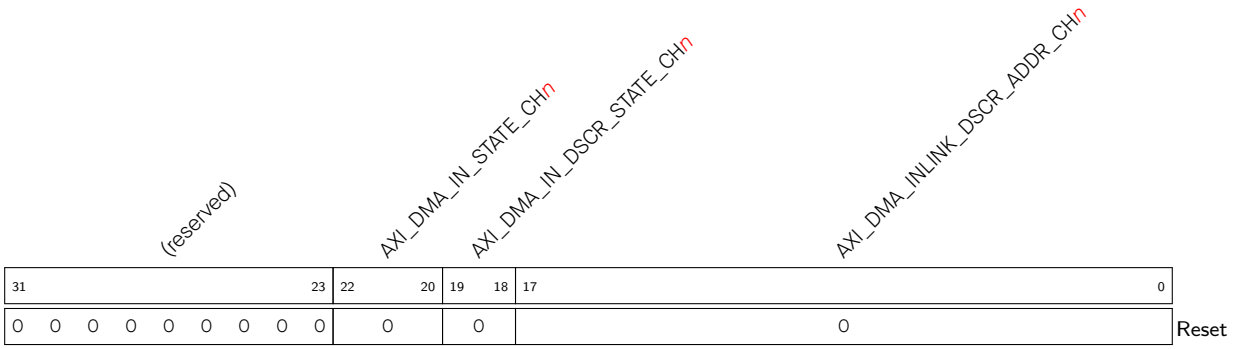
AXI_DMA_IN_REMAIN_UNDER_6B_CH*n* 保留。(RO)

AXI_DMA_IN_REMAIN_UNDER_7B_CH*n* 保留。(RO)

AXI_DMA_IN_REMAIN_UNDER_8B_CH*n* 保留。(RO)

AXI_DMA_IN_BUF_HUNGRY_CH*n* 保留。(RO)

Register 6.113. AXI_DMA_IN_STATE_CH*n*_REG (*n*：0-2) (0x0028+0x68**n*)

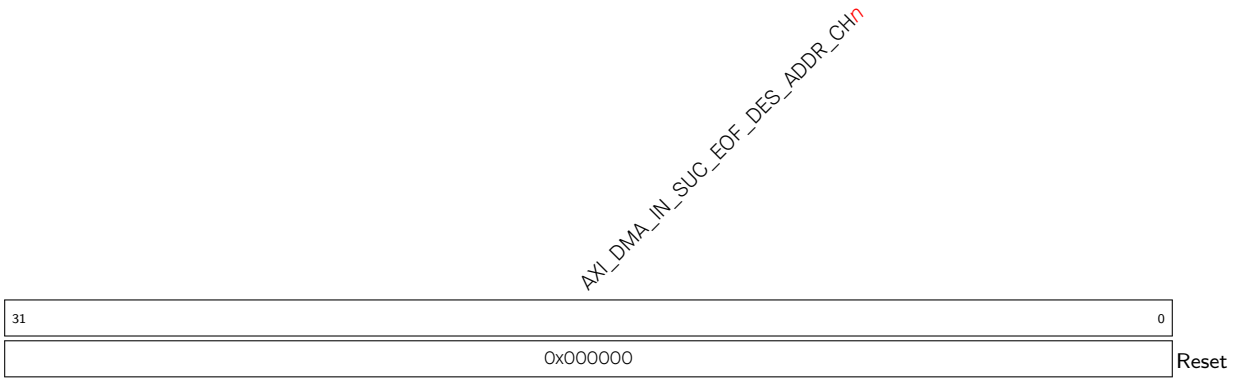


AXI_DMA_INLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的接收链表描述符所在地址的低 18 位。如果当前处理的接收链表描述符是最后一个描述符，则该字段表示当前处理的接收链表描述符地址。(RO)

AXI_DMA_IN_DSCR_STATE_CH*n* 保留。(RO)

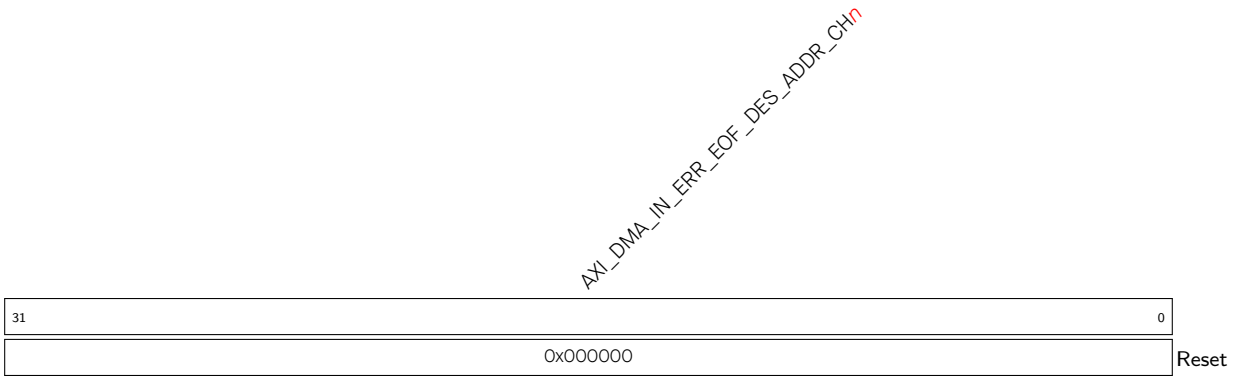
AXI_DMA_IN_STATE_CH*n* 保留。(RO)

Register 6.114. AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*：0-2) (0x002C+0x68**n*)



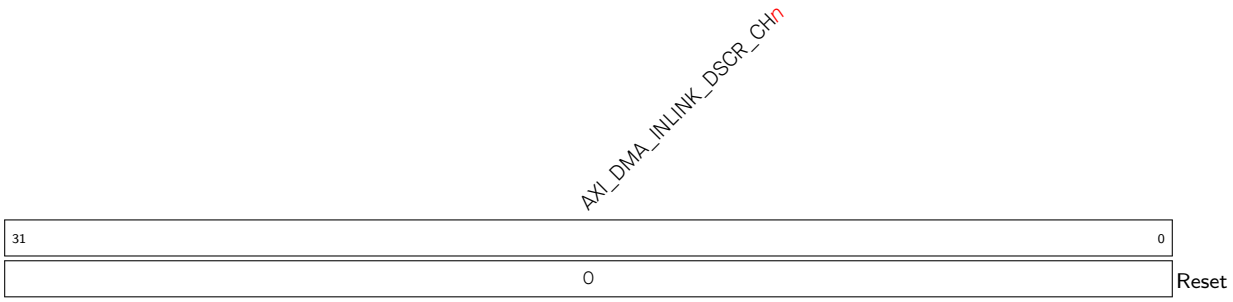
AXI_DMA_IN_SUC_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的接收链表描述符的地址。(RO)

Register 6.115. AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*：0-2) (0x0030+0x68**n*)



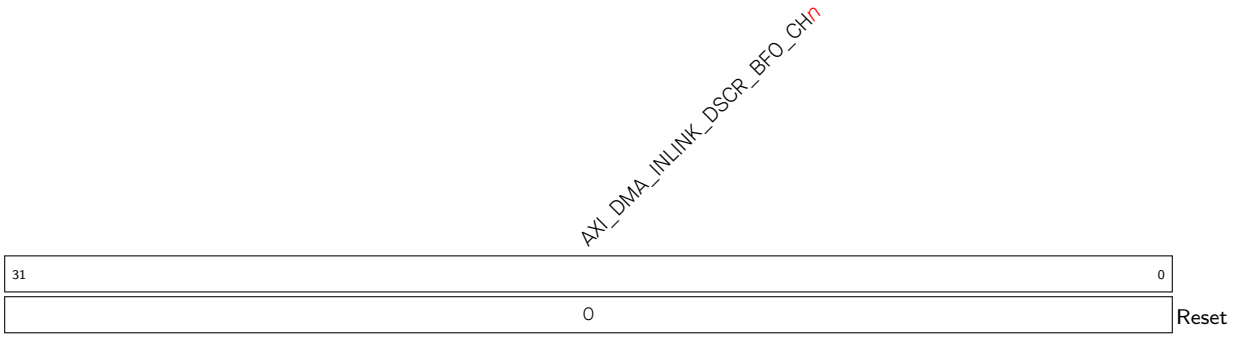
AXI_DMA_IN_ERR_EOF_DES_ADDR_CH*n* 表示当前接收数据有错时，相应接收链表描述符的位置。仅对 UHCI 有效。(RO)

Register 6.116. AXI_DMA_IN_DSCR_CH*n*_REG (*n*：0-2) (0x0034+0x68**n*)



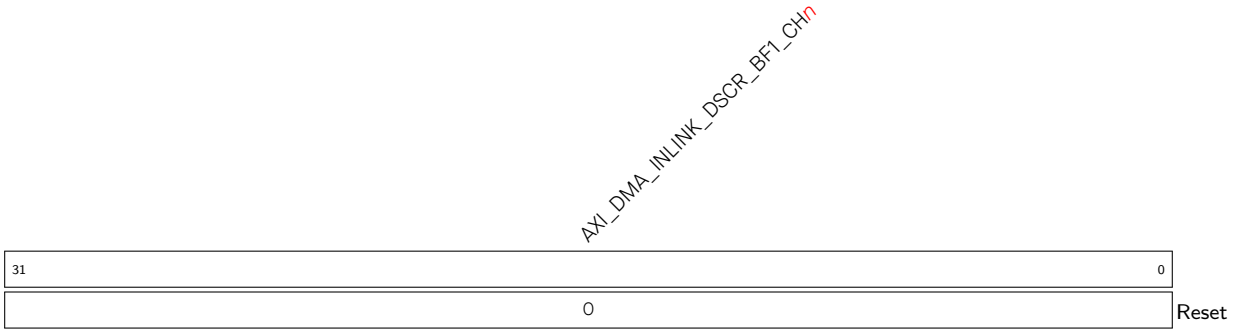
AXI_DMA_INLINK_DSCR_CH*n* 表示当前已预读取的接收链表描述符指向的下一个接收链表描述符地址 x+1。(RO)

Register 6.117. AXI_DMA_IN_DSCR_BF0_CH*n*_REG (*n*：0-2) (0x0038+0x68**n*)



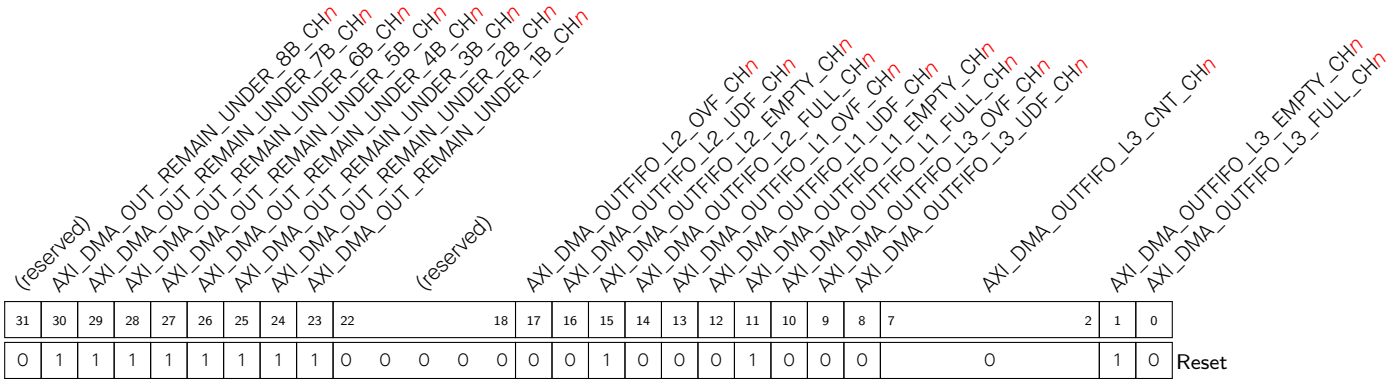
AXI_DMA_INLINK_DSCR_BF0_CH*n* 表示当前已预读取的接收链表描述符所在地址 x。(RO)

Register 6.118. AXI_DMA_IN_DSCR_BF1_CH*n*_REG (*n*：0-2) (0x003C+0x68**n*)



AXI_DMA_INLINK_DSCR_BF1_CH*n* 表示前一个已预读取的接收链表描述符所在地址 x-1。(RO)

Register 6.119. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*：0-2) (0x0150+0x68*n)



AXI_DMA_OUTFIFO_L3_FULL_CH*n* 表示 L3 TX FIFO 是否已满。

0：未满

1：已满

(RO)

AXI_DMA_OUTFIFO_L3_EMPTY_CH*n* 表示 L3 TX FIFO 是否为空。

0：不为空

1：空

(RO)

AXI_DMA_OUTFIFO_L3_CNT_CH*n* 表示 L3 TX FIFO 中的数据字节数。(RO)

AXI_DMA_OUTFIFO_L3_UDF_CH*n* 表示 L3 TX FIFO 是否下溢。

0：无下溢

1：下溢

(RO)

AXI_DMA_OUTFIFO_L3_OVF_CH*n* 表示 L3 TX FIFO 是否溢出。

0：无溢出

1：溢出

(RO)

AXI_DMA_OUTFIFO_L1_FULL_CH*n* 表示 L1 TX FIFO 是否已满。

0：未满

1：已满

(RO)

AXI_DMA_OUTFIFO_L1_EMPTY_CH*n* 表示 L1 TX FIFO 是否为空。

0：不为空

1：空

(RO)

AXI_DMA_OUTFIFO_L1_UDF_CH*n* 表示 L1 TX FIFO 是否下溢。

0：无下溢

1：下溢

(RO)

见下页...

Register 6.119. AXI_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*：0-2) (0x0150+0x68**n*)

接上页...

AXI_DMA_OUTFIFO_L1_OVF_CH*n* 表示 L1 TX FIFO 是否溢出。

0：无溢出

1：溢出

(RO)

AXI_DMA_OUTFIFO_L2_FULL_CH*n* 表示 L2 TX FIFO 是否已满。

0：未满

1：已满

(RO)

AXI_DMA_OUTFIFO_L2_EMPTY_CH*n* 表示 L2 TX FIFO 是否为空。

0：不为空

1：空

(RO)

AXI_DMA_OUTFIFO_L2_UDF_CH*n* 表示 L2 TX FIFO 是否下溢。

0：无下溢

1：下溢

(RO)

AXI_DMA_OUTFIFO_L2_OVF_CH*n* 表示 L2 TX FIFO 是否溢出。

0：无溢出

1：溢出

(RO)

AXI_DMA_OUT_REMAIN_UNDER_1B_CH*n* 保留。(RO)

AXI_DMA_OUT_REMAIN_UNDER_2B_CH*n* 保留。(RO)

AXI_DMA_OUT_REMAIN_UNDER_3B_CH*n* 保留。(RO)

AXI_DMA_OUT_REMAIN_UNDER_4B_CH*n* 保留。(RO)

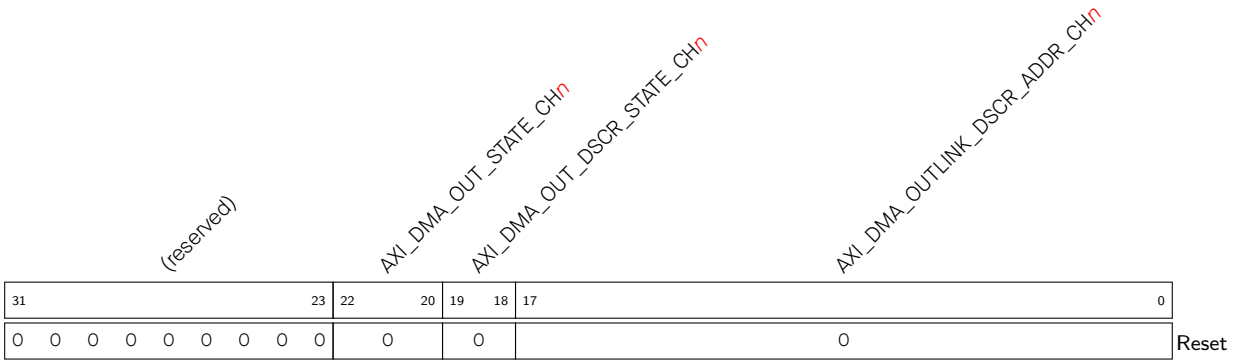
AXI_DMA_OUT_REMAIN_UNDER_5B_CH*n* 保留。(RO)

AXI_DMA_OUT_REMAIN_UNDER_6B_CH*n* 保留。(RO)

AXI_DMA_OUT_REMAIN_UNDER_7B_CH*n* 保留。(RO)

AXI_DMA_OUT_REMAIN_UNDER_8B_CH*n* 保留。(RO)

Register 6.120. AXI_DMA_OUT_STATE_CH*n*_REG (*n*：0-2) (0x0160+0x68**n*)

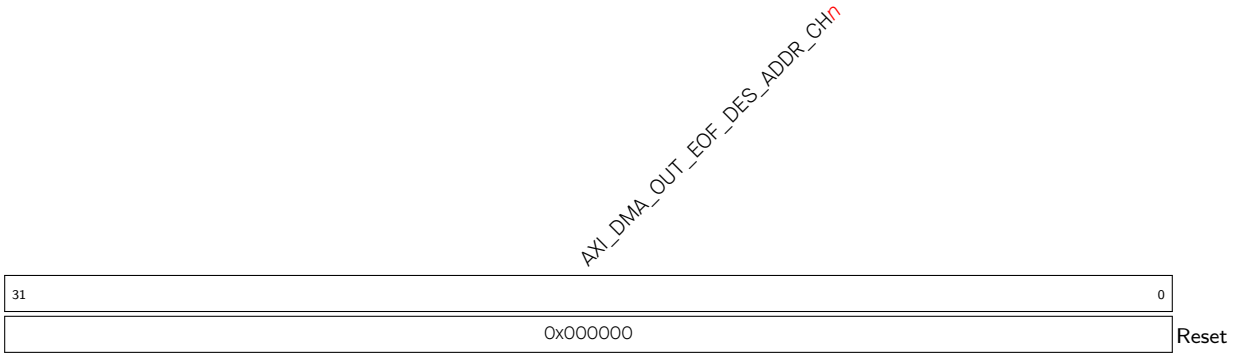


AXI_DMA_OUTLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的发送接收链表描述符所在地址的低 18 位。如果当前处理的发送链表描述符是最后一个描述符，则该字段表示当前处理的发送链表描述符地址。(RO)

AXI_DMA_OUT_DSCR_STATE_CH*n* 保留。(RO)

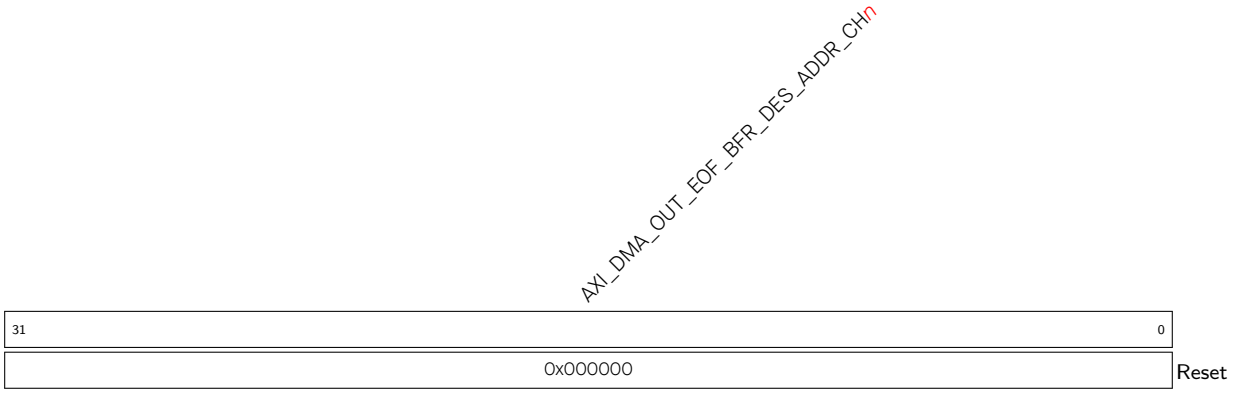
AXI_DMA_OUT_STATE_CH*n* 保留。(RO)

Register 6.121. AXI_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*：0-2) (0x0164+0x68**n*)



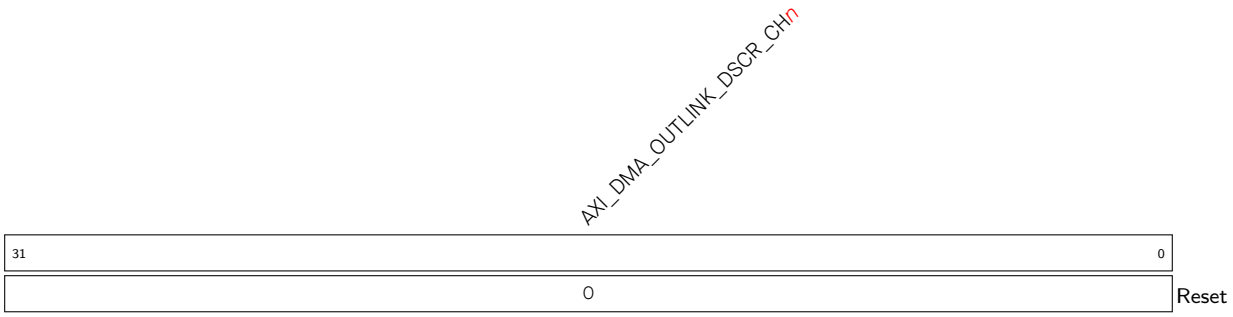
AXI_DMA_OUT_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的发送链表描述符的地址。(RO)

Register 6.122. AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*：0-2) (0x0168+0x68**n*)



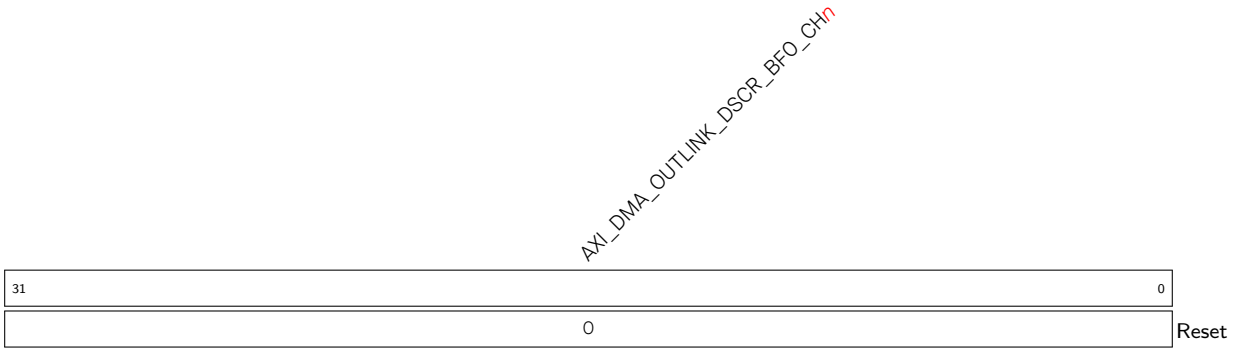
AXI_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* 表示倒数第二个发送链表描述符的地址。(RO)

Register 6.123. AXI_DMA_OUT_DSCR_CH*n*_REG (*n*：0-2) (0x016C+0x68**n*)



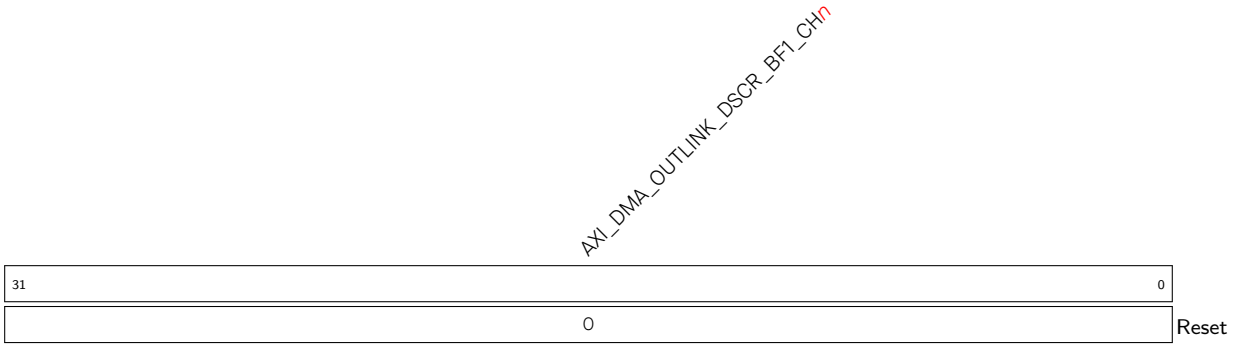
AXI_DMA_OUTLINK_DSCR_CH*n* 表示当前已预读取的发送链表描述符指向的下一个发送链表描述符地址 y+1。(RO)

Register 6.124. AXI_DMA_OUT_DSCR_BFO_CH*n*_REG (*n*：0-2) (0x0170+0x68**n*)



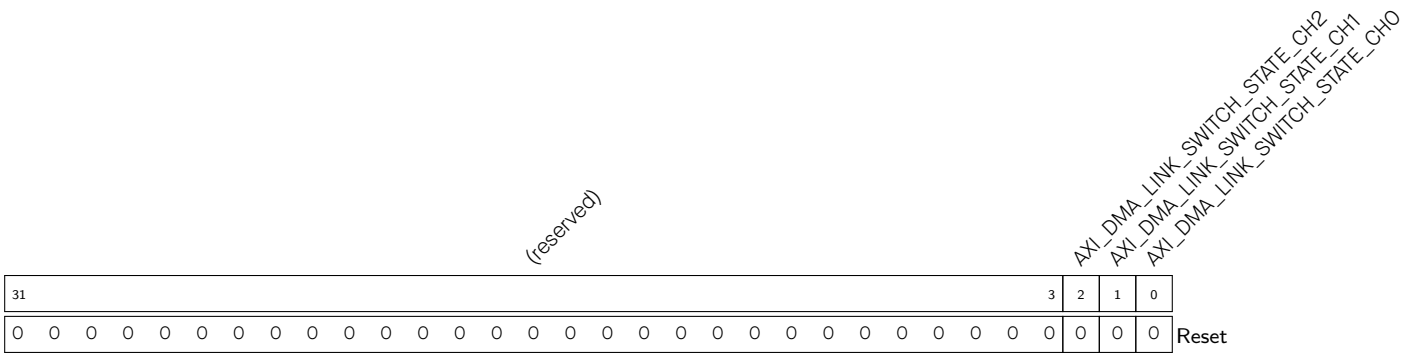
AXI_DMA_OUTLINK_DSCR_BFO_CH*n* 表示当前已预读取的发送链表描述符所在地址 y。(RO)

Register 6.125. AXI_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*：0-2) (0x0174+0x68**n*)



AXI_DMA_OUTLINK_DSCR_BF1_CH*n* 表示前一个已预读取的发送链表描述符所在地址 y-1。(RO)

Register 6.126. AXI_DMA_LINK_SWITCH_STATE_REG (0x02DC)



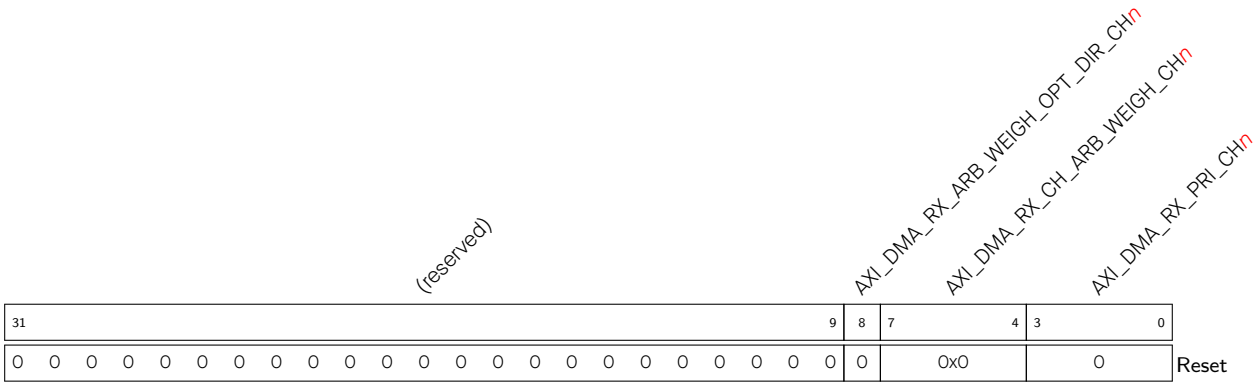
AXI_DMA_LINK_SWITCH_STATE_CH*n* 配置是否切换 AXI DMA 读取的链表。

0：不切换

1：切换

(R/W)

Register 6.127. AXI_DMA_IN_PRI_CH*n*_REG (*n*：0-2) (0x0040+0x68*n)



AXI_DMA_RX_PRI_CH*n* 配置接收通道 *n* 的优先级。值越大，优先级越高。

取值范围：0 ~ 5。(R/W)

AXI_DMA_RX_CH_ARB_WEIGH_CH*n* 配置接收通道 *n* 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

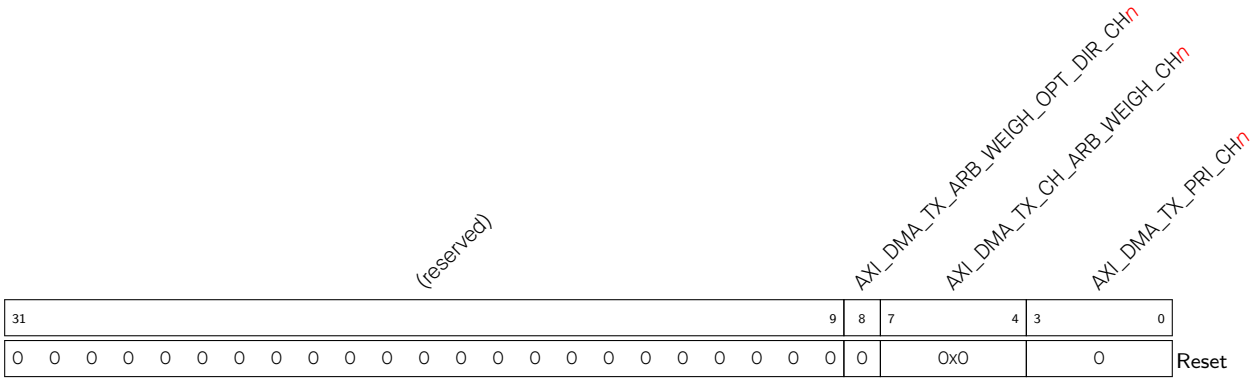
AXI_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* 配置接收通道 *n* 是否开启权重仲裁优化。

0：开启

1：关闭

(R/W)

Register 6.128. AXI_DMA_OUT_PRI_CH*n*_REG (*n*：0-2) (0x0178+0x68*n)



AXI_DMA_TX_PRI_CH*n* 配置发送通道 *n* 的优先级。值越大，优先级越高。

取值范围：0 ~ 5。(R/W)

AXI_DMA_TX_CH_ARB_WEIGH_CH*n* 配置发送通道 *n* 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

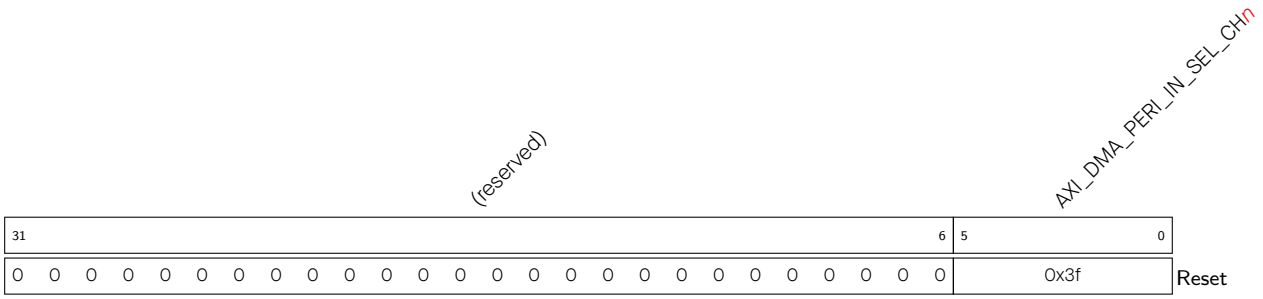
AXI_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* 配置发送通道 *n* 是否开启权重仲裁优化。

0：开启

1：关闭

(R/W)

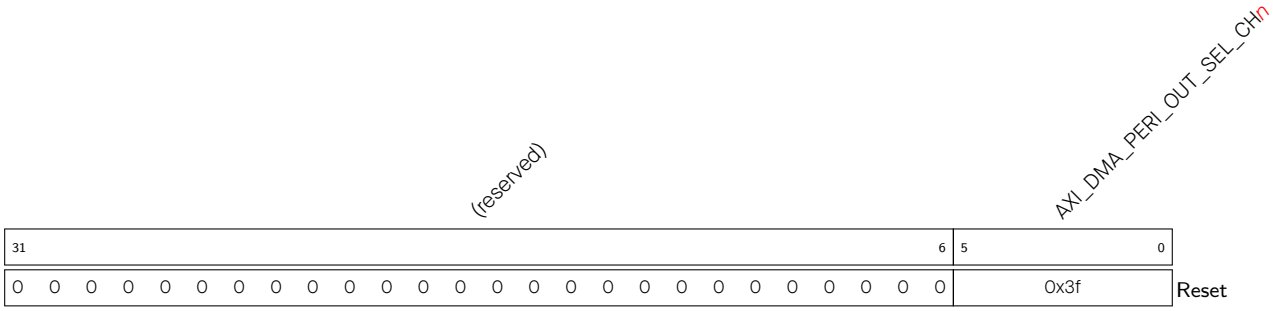
Register 6.129. AXI_DMA_IN_PERI_SEL_CH*n*_REG (*n*：0-2) (0x0044+0x68**n*)



AXI_DMA_PERI_IN_SEL_CH*n* 配置连接到接收通道 *n* 的外设。

0：LCD_CAM
1：SPI2
2：SPI3
3：PARLIO
4：AES
5：SHA
6 ~ 15：Dummy-6 ~ Dummy-15
16 ~ 63：无效
(R/W)

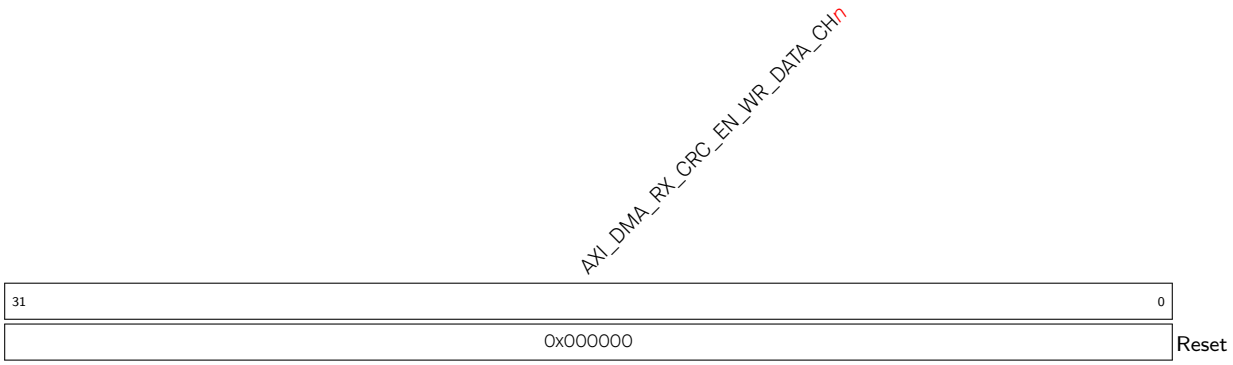
Register 6.130. AXI_DMA_OUT_PERI_SEL_CH*n*_REG (*n*：0-2) (0x017C+0x68**n*)



AXI_DMA_PERI_OUT_SEL_CH*n* 配置连接到发送通道 *n* 的外设。

0：LCD_CAM
1：SPI2
2：SPI3
3：PARLIO
4：AES
5：SHA
6 ~ 15：Dummy-6 ~ Dummy-15
16 ~ 63：无效
(R/W)

Register 6.131. AXI_DMA_RX_CRC_EN_WR_DATA_CH*n*_REG (*n*：0-2) (0x0058+0x68**n*)



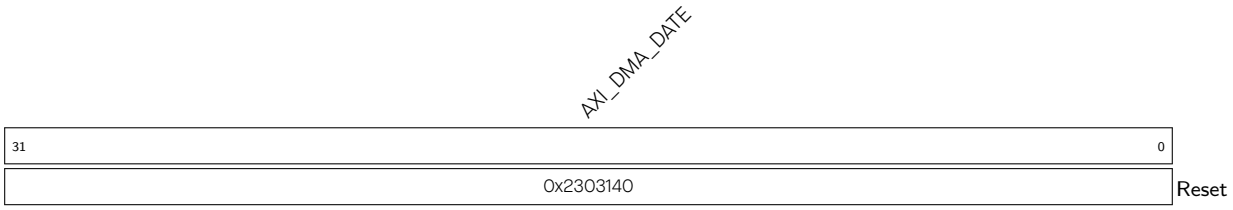
AXI_DMA_RX_CRC_EN_WR_DATA_CH*n* 配置接收通道 *n* 中间结果的每一位是否参与 CRC 计算矩阵。

0：不参与

1：参与

(R/W)

Register 6.132. AXI_DMA_DATE_REG (0x02D8)



AXI_DMA_DATE 版本控制寄存器。(R/W)

第 7 章

VDMA 控制器 (VDMA)

7.1 概述

DMA（直接内存访问）是指不依赖 CPU，在存储器和外设之间完成数据搬运。ESP32-P4 上的 VDMA 控制器是一种通用 DMA，支持存储器到存储器、存储器到外设、外设到存储器的高速数据传输。VDMA 控制器以 AXI 作为总线接口，采用 AXI3 标准协议，带有两个 AXI 主机接口，支持用户动态地选择主机接口进行数据传输。

7.2 术语

下列术语是在本章节的语境下定义的。

- **AXI**：全称为 Advanced eXtensible Interface（高级可扩展接口），是 AMBA 3（Advanced Microcontroller Bus Architecture，高级微控制器总线架构）协议的一部分，为存储器和外设之间、存储器和存储器之间提供了高宽带、低延迟的数据传输。
- **外设**：指的是与 VDMA 之间有握手接口的系统模块，即表 7.4-1 列出的外设。
- **存储器**：存储器总是为 DMA 传输做好准备，与 VDMA 之间没有握手接口。VDMA 支持的存储器包括 HP L2MEM、外部 flash 和外部 RAM。
- **源**：指的是源外设或源存储器，VDMA 通过 AXI 接口从源中读取数据，然后将数据存储在通道 FIFO 中。
- **源传输**：指的是源到 VDMA 的传输。
- **目标**：指的是目标外设或目标存储器，VDMA 将存储在通道 FIFO 中的数据写入目标。
- **目标传输**：指的是 VDMA 到目标的传输。
- **通道**：指的是源和目标之间的数据通路。
- **主机接口**：指的是 VDMA 的两个 AXI 主机接口。
- **从机接口**：指的是用于配置寄存器的 APB3 接口。
- **握手接口**：握手接口用于在 VDMA 和源或目标外设之间执行握手。握手接口用于请求、确认和控制 VDMA 传输。
- **流控制器**：决定 DMA 块传输的长度并终止传输的设备，可以是 VDMA、源或目标外设。存储器不能用作流控制器。

7.3 特性

VDMA 具有以下特性：

- 4 个通道，每个通道都是单向的，支持源到目标的数据传输

- 2 个 AXI 主机接口
- 支持与 MIPI DSI（显示串行接口）和 ISP（图像信号处理器）的握手
- 支持存储器与存储器之间、ISP 与存储器之间、MIPI DSI 与存储器之间的 DMA 传输
- DMA 传输层次有多个级别
- 每个通道的传输类型、传输长度、传输大小可配置
- 支持单块传输
- 支持基于连续地址、自动重新加载、影子寄存器和链表的多块传输
- 支持源传输和目标传输独立配置多块传输类型
- 通道禁用而不丢失数据
- 通道暂停、恢复和中止
- 可配置的通道优先级仲裁
- 支持使用 VDMA 或外设作为流控制器
- 外设的握手接口和通道之间的映射可配置

7.4 结构概览

图 7.4-1 所示为 VDMA 的架构，以及 VDMA 与其他系统模块的连接。

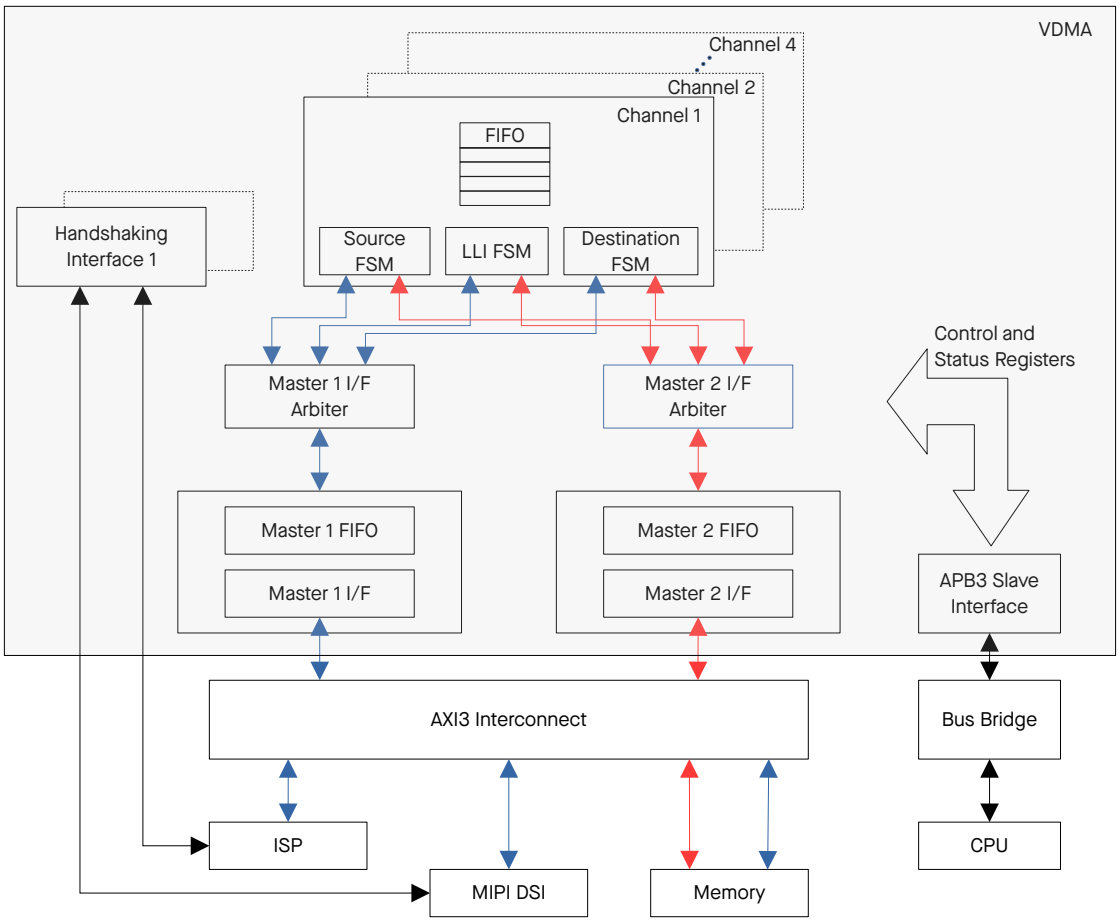


图 7.4-1. VDMA 架构

从图中可以看出：

- CPU 通过 APB3 从机接口 (APB3 Slave Interface) 访问 VDMA 的寄存器。
- VDMA 有 4 个通道（Channel 1 到 Channel 4）。每个通道对应一个 FIFO，深度为 64，宽度为 64 位。
- VDMA 有 2 个 AXI 主机接口（Master 1 和 Master 2）。
  - Master 1 可以访问 ISP、MIPI DSI 和存储器，如图中蓝色箭头所示。
  - Master 2 只能访问存储器，如图中红色箭头所示。
  - Master 1 和 Master 2 各有一个仲裁器，对通道上的读请求和写请求进行仲裁。
- VDMA 有 3 个握手接口。表 7.4-1 为握手接口和外设之间的映射。

表 7.4-1. 握手接口与外设映射表

握手接口编号	外设
0	MIPI DSI
1	ISP（ISP 到存储器）
2	ISP（存储器到 ISP）

更多关于 ISP 和 MIPI DSI 的信息请参考章节 38 图像信号处理器 *(ISP)*、43 *MIPI DSI*。

7.5 功能描述

7.5.1 传输层次

传输最多分为 4 个级别：DMA 传输级别、块传输级别、transaction（事务传输）级别和 AXI 传输级别。

图 7.5-1 所示为 VDMA 与外设之间传输的 4 个传输级别的层次结构。

图 7.5-2 所示为 VDMA 与存储器之间传输的 3 个传输级别结构：DMA 传输、块传输和 AXI 传输。VDMA 与存储器之间的传输没有 transaction 级别，因为存储器始终假定为准备好进行数据传输。

以下关于 DMA 传输、块传输和 AXI 传输的描述均适用于存储器和外设。**transaction 级别的描述只适用于 VDMA 与外设之间的传输**。

- **DMA 传输**：软件决定 DMA 传输中的块数。DMA 传输完成后，VDMA 会禁用通道并产生中断。然后可以对通道重新配置用以进行新的 DMA 传输。DMA 传输可能是单块传输，也可能是多块传输。
- **块传输**：块传输传输的是一个 VDMA 数据块，其数量由块长度表示，由流控制器决定。块传输有两种类型：单块传输和多块传输。由图 7.5-1 可知，一个块传输可能由多个 transaction（事务传输）组成。
- **transaction（事务传输）**：一次外设请求是单次 transaction 或突发 transaction，表示需要一个 AXI 单次传输或多个 AXI 突发传输。
  - 单次 transaction：长度始终为 1，单位为 DMAC_CH*n*_SRC_TR_WIDTH。
  - 突发 transaction：长度通过源的 DMAC_CH*n*_SRC_MSIZE 或目标的 DMAC_CH*n*_DST_MSIZE 被写到 VDMA 中，通常与源或目标外设中的 FIFO 大小有某种关系。具体请参考各外设章节。
- **AXI 传输**：指的是 AXI 协议的传输，分为 AXI 单次传输和 AXI 突发传输。

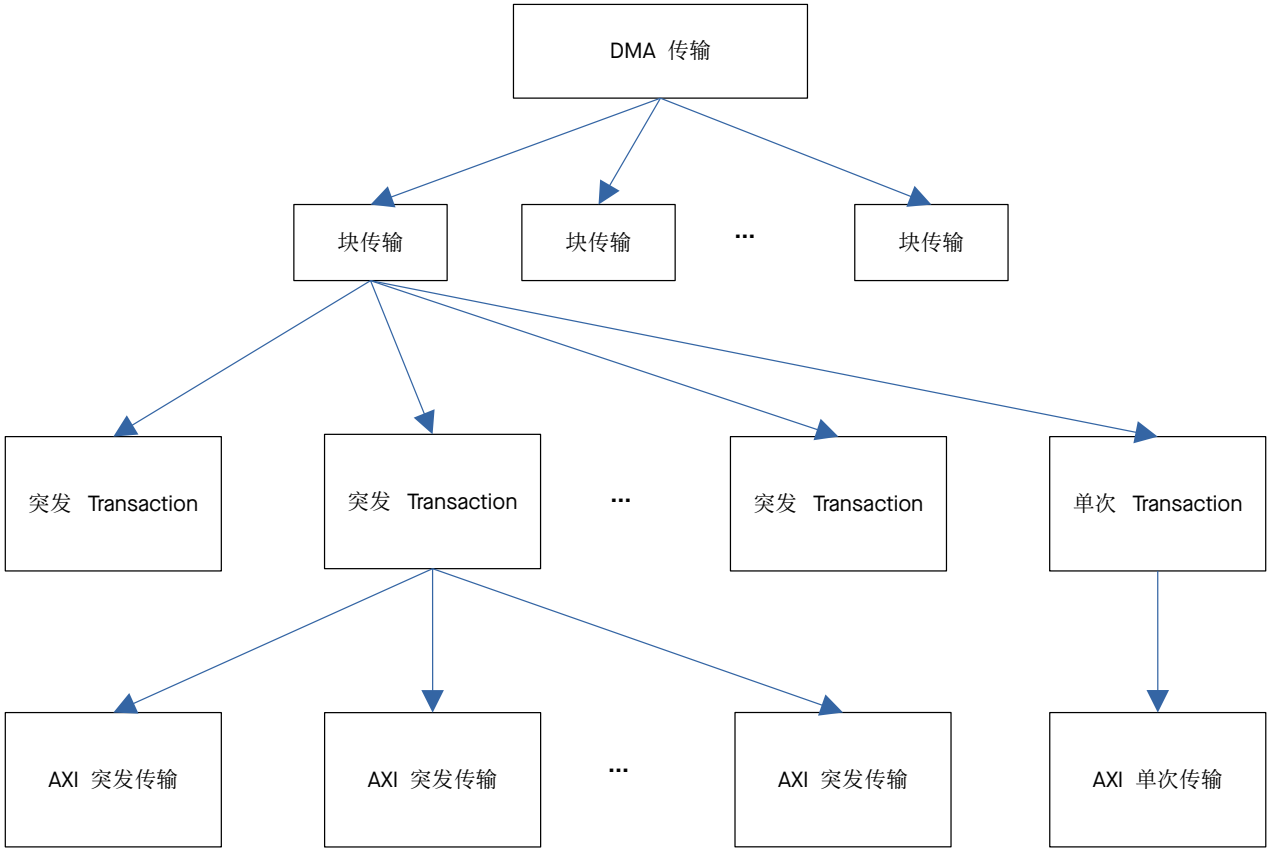


图 7.5-1. 外设的传输层次结构

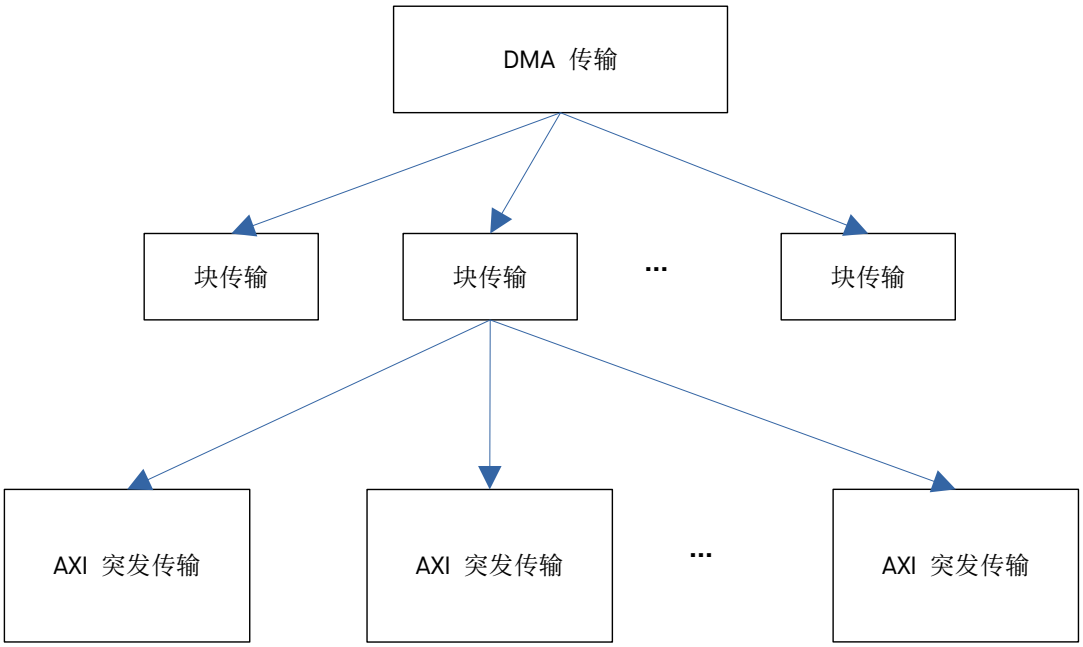


图 7.5-2. 存储器的传输层次结构

7.5.2 仲裁机制

仲裁机制对所有通道的读请求和写请求进行仲裁。

VDMA 采用优先级和公平平等 (fair-among-equals) 的仲裁机制。每个通道的优先级可在通道配置寄存器 DMAC_CH*n*_CFG1_REG 中定义。

优先级的范围是 0（最低）到 3（最高），多个通道可以被赋予相同的优先级。

优先级和公平平等仲裁的仲裁机制工作原理如下：

- 根据请求的优先级决定向哪个请求客户端发出授权信号。具有最高优先级的请求将被授予访问通道的权限。
- 两个或多个请求具有相同的优先级时，具有相同优先级的请求客户端将随机地获得权限。

7.5.2.1 读仲裁器

读仲裁器仲裁的是读请求，包括：

- VDMA 获取源的数据或状态的读请求 (Source Data/Status Read Request)
- VDMA 获取目标的状态的读请求 (Destination Status Read Request)
- VDMA 获取链表描述符的读请求 (LLI Descriptor Read Request)

读仲裁器的示意图如图 7.5-3 所示。

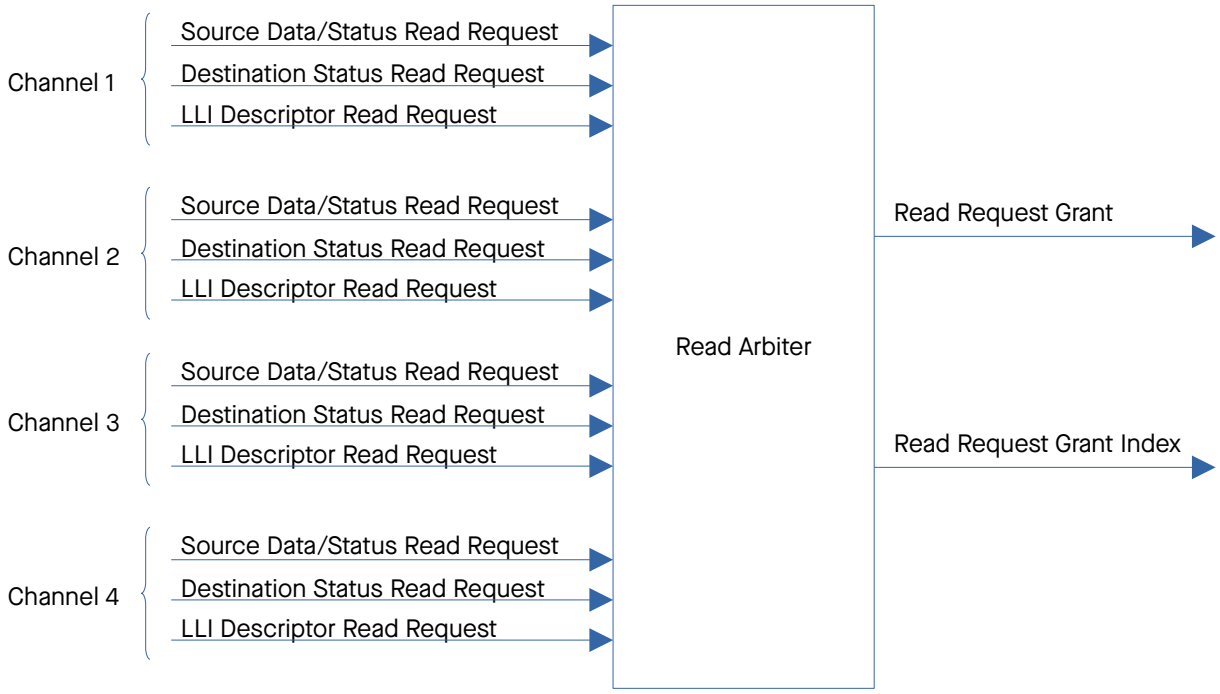


图 7.5-3. VDMA 读仲裁器

无论通道是否配置为最大优先级，获取链表描述符的读请求都将优先获得仲裁权限。

7.5.2.2 写仲裁器

写仲裁器仲裁的是写请求，包括：

- 目标数据写请求 (Destination Data Write Request)
- 链表描述符回写请求 (LLI Descriptor Write-Back Request)

写仲裁器的示意图如图 7.5-4 所示。

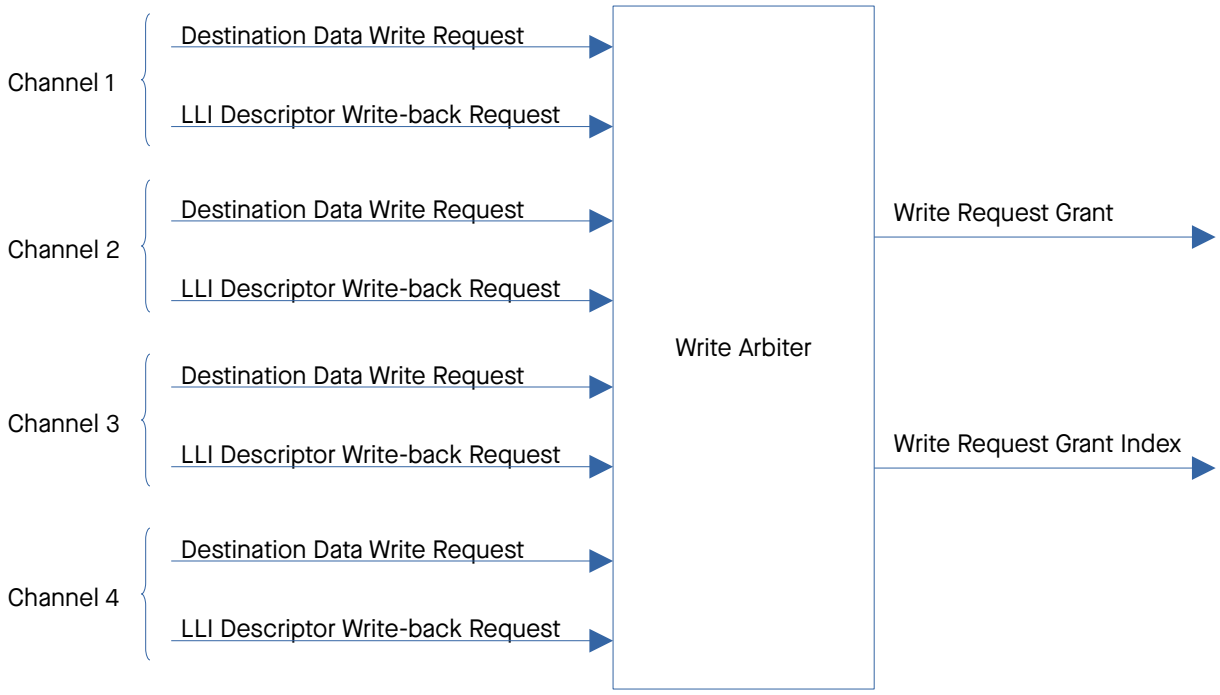


图 7.5-4. VDMA 写仲裁器

在相同的通道中，目标数据写请求优先级高于链表描述符回写的优先级；如果不同通道具有相同的优先级，则目标数据写请求具有最高优先级。

7.5.3 握手接口

外设使用握手接口向 VDMA 表明它已准备好通过 AXI 总线来传输或接收数据。握手接口的操作取决于流控制器是外设还是 VDMA。

VDMA 在每次握手时执行一次单次 transaction 或突发 transaction。在某些情况下，仅使用突发 transaction 无法完成块传输。这种情况通常发生在块大小不是突发 transaction 长度的整数倍时。在这种情况下，块传输使用突发 transaction，直到块的剩余大小小于突发 transaction 中的数据量为止。此时，使用单次 transaction 完成块传输。

源的突发 transaction 大小由 DMAC_CH*n*_SRC_MSIZE 决定，目标的突发 transaction 大小由 DMAC_CH*n*_DST_MSIZE 决定，这也是用于每次握手的突发 transaction 的大小。DMAC_CH*n*_SRC_MSIZE 或 DMAC_CH*n*_DST_MSIZE 的值应与外设的配置一致。更多信息，请参考相关外设章节。

VDMA 与 MIPI DSI、ISP （ISP 到存储器）和 ISP（存储器到 ISP）之间有 3 个握手接口。表 7.4-1 说明了握手接口和外设之间的映射关系。

7.5.4 传输控制

传输控制方便将数据从源搬运到目标。来自源的数据在被发送到目标之前暂时存储在通道的 FIFO 中。如果源和目标使用不同的传输大小进行数据传输，VDMA 将数据进行打包和解包，以适应 FIFO 的配置。

对于特定的 DMA 传输，传输类型和流控制配置由 DMAC_CH*n*_TT_FC 决定。

7.5.4.1 单块传输

如果一个 DMA 传输仅由单个块组成，软件可以将 DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE 设置为 0，将 VDMA 配置为连续地址的单块传输。此时，VDMA 在完成 DMAC_CH*n*_BLOCK_TS 定义的块长度传输时，将禁用通道。

> **说明：**
> 单块传输其实只是多块传输的一种特例，两者并无严格区分。在本文档中，单块传输指的是只有一个块，多块传输至少包含两个块。

7.5.4.2 多块传输

如果一个 DMA 传输包含多个块，软件可以配置 DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE 来选择多块传输类型。根据控制传输的寄存器（DMAC_CH*n*_SARO_REG、DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO/1_REG）的更新方式不同，共有以下四种多块传输类型：

1. 连续地址
2. 自动重新加载，寄存器的值从初始值重新加载
3. 影子寄存器，寄存器的值从影子寄存器加载
4. 链表，寄存器的值从下一个链表加载

表 7.5-1 列出了寄存器更新方法的所有可能组合。

表 7.5-1. 多块传输寄存器更新方法

DMAC_CH*n*_CFGO_REG		寄存器更新方法				
DMAC_CH*n*_SRC_MULTBLK_TYPE	DMAC_CH*n*_DST_MULTBLK_TYPE	DMAC_CH*n*_SARO_REG	DMAC_CH*n*_DARO_REG	DMAC_CH*n*_CTLO_REG /CTL1_REG	DMAC_CH*n*_BLOCK_TS	说明
00	00	连续地址	连续地址	不更新	不更新	单块或多块的最后一块
00	01	连续地址	从初始值重新加载	从初始值重新加载	从初始值重新加载	-
00	10	连续地址	从影子寄存器加载	从影子寄存器加载	从影子寄存器加载	-
01	00	从初始值重新加载	连续地址	从初始值重新加载	从初始值重新加载	-
01	01	从初始值重新加载	从初始值重新加载	从初始值重新加载	从初始值重新加载	-

01	10	从初始值重新加载	从影子寄存器加载	从影子寄存器加载	从影子寄存器加载	-
10	00	从影子寄存器加载	连续地址	从影子寄存器加载	从影子寄存器加载	-
10	01	从影子寄存器加载	从初始值重新加载	从影子寄存器加载	从影子寄存器加载	-
10	10	从影子寄存器加载	从影子寄存器加载	从影子寄存器加载	从影子寄存器加载	-
00	11	连续地址	从下一个链表加载	从下一个链表加载	从下一个链表加载	-
01	11	从初始值重新加载	从下一个链表加载	从下一个链表加载	从下一个链表加载	-
11	00	从下一个链表加载	连续地址	从下一个链表加载	从下一个链表加载	-
11	01	从下一个链表加载	从初始值重新加载	从下一个链表加载	从下一个链表加载	-
11	11	从下一个链表加载	从下一个链表加载	从下一个链表加载	从下一个链表加载	-
10	11	-	-	-	-	无效。生成中断 *
11	10	-	-	-	-	无效。生成中断 *

* 说明：生成的中断为 DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INT。

连续地址

在这种情况下，每个连续块的地址从前一个块的结束地址开始。配置 DMAC_CH*n*_SRC_MULTBLK_TYPE 或 DMAC_CH*n*_DST_MULTBLK_TYPE 为 0 使源地址或目标地址在块之间连续。

在多块传输的情况下，DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE 的值不能同时被选择为连续地址，即不能同时为 0。若某一块传输过程中 DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE 都被配置成了 0，那么当前传输就会被认为是最后一个块。

在多块传输中，如果需要 DMAC_CH*n*_SARO_REG 和 DMAC_CH*n*_DARO_REG 在块之间地址连续，可以通过使用链表或影子寄存器间接实现此功能。

> **说明：**
> 如果启用了基于连续地址的多块传输（即 DMAC_CH*n*_SRC_MULTBLK_TYPE 或 DMAC_CH*n*_DST_MULTBLK_TYPE 为 0），则在完整的 DMA 传输中必须至少有两个块，否则会导致不可预测的行为。

从初始值重新加载

在这种情况下，DMAC_CH*n*_SARO_REG 和 DMAC_CH*n*_DARO_REG 通道寄存器在新的块传输开始时从它们的初始值重新加载。VDMA 不会进行下一个块传输，直到软件清除相应通道的块传输完成中断，即将 DMAC_CH*n*_CLEAR_SRC_TRANSCOMP_INTSTAT 和 DMAC_CH*n*_CLEAR_DST_TRANSCOMP_INTSTAT 置为 1。

说明：
如果启用了基于自动重新加载的多块传输，则在完整的 DMA 传输中必须至少有两个块，否则会导致不可预测的行为。

从影子寄存器加载

在这种情况下，DMAC_CH*n*_SARO_REG、DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG 在每个块完成时与它们对应的影子寄存器进行同步。影子寄存器的值将用于后续的块。

软件将对应块的配置写到相应的影子寄存器。在完成上一个块传输时，VDMA 将影子寄存器的内容复制到 DMAC_CH*n*_SARO_REG、DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG 然后再开始新的块传输。

对 DMAC_CH*n*_SARO_REG、DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG 的读操作总是返回与当前块传输对应的数据，而不是对应下一个块的影子寄存器的内容。

DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 指示影子寄存器的内容是否有效（1 为有效，0 为无效）。如果在影子寄存器获取阶段，此位的值为 0，VDMA 将忽略影子寄存器的内容并生成 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT 中断。VDMA 等待软件将 1 写入 DMAC_CH*n*_BLK_TFR_RESUMEREQ 以指示影子寄存器可用，然后再尝试读取影子寄存器以继续下一个块传输。

说明：
在从影子寄存器加载的多块传输下，DMAC_CH*n*_SARO_REG、DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG、DMAC_CH*n*_CTL1_REG 寄存器里的实际值是当前传输对应的配置。软件写这些寄存器其实是写到对应的影子寄存器里面。

链表

在这种情况下，VDMA 会从存储器中获取某个块的块描述符，用来更新 DMAC_CH*n*_SARO_REG、DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG，然后再发起块传输。这个过程被称为 LLI（链表项）更新。VDMA 的块链 (block chaining) 特性使用链表指针寄存器 DMAC_CH*n*_LOCO 来存储下一个链表项在存储器中的地址，以此实现数据传输的无缝连续。图 7.5-5 显示了 VDMA 链表项。

31	0
Reserved	
Reserved	
CHn_LLP_STATUS[63:32]	
CHn_LLP_STATUS[31:0]	
Write Back for CHn_DSTAT	
Write Back for CHn_SSTAT	
CHn_CTL1[31:0]	
CHn_CTL0[31:0]	
Reserved	
CHn_LLP0[31:6]	Reserved[5:0]
Reserved	
CHn_BLOCK_TS[31:0]	
Reserved	
CHn_DAR[31:0]	
Reserved	
CHn_SAR[31:0]	

图 7.5-5. VDMA 链表项

VDMA 具有动态扩展链表的特性，不需要提前在系统内存中创建整个链表。要实现此功能，可以将 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 配置为 1，如果链表项是最后一个则 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 也应配置为 1。

对于基于链表的多块传输，DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 用来指示从存储器中获取的链表项是否有效（1 为有效，0 为无效）。如果 LLI 无效，VDMA 将丢弃 LLI 并生成 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT 中断。此中断会导致 VDMA 停止相应的通道，等待软件将 DMAC_CH*n*_BLK_TFR_RESUMEREQ 写为 1 以指示 LLI 可用，然后再尝试读取 LLI。

说明：

对于预取的 LLI，如果 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 为 0，则不会生成 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT 中断。VDMA 在完成当前块传输后，再次尝试获取 LLI，只有当 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 仍然为 0 时，才会生成 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT 中断。

下列 LLI 字段由软件准备：

- CH*n*_SAR：字段描述同 DMAC_CH*n*_SARO_REG
- CH*n*_DAR：字段描述同 DMAC_CH*n*_DARO_REG
- CH*n*_BLOCK_TS：字段描述同 DMAC_CH*n*_BLOCK_TS
- CH*n*_LLP0[31:6]：字段描述同 DMAC_CH*n*_LLP0_REG[31:6]

- CH*n*_CTL0：字段描述同 DMAC_CH*n*_CTL0_REG
- CH*n*_CTL1：字段描述同 DMAC_CH*n*_CTL1_REG

如果启用了状态回写选项，VDMA 将 CH*n*_CTL0、CH*n*_CTL1、CH*n*_LLP_STATUS、CH*n*_SSTAT 和 CH*n*_DSTAT 信息回写到为该字段定义的位置，即从地址 [CH*n*_LLP] + 0x20 到 [CH*n*_LLP] + 0x34。CH*n*_SSTAT 和 CH*n*_DSTAT 的回写可以通过配置 DMAC_CH*n*_SRC_STAT_EN 和 DMAC_CH*n*_DST_STAT_EN 单独启用或禁用。

说明：

- 如果 CH*n*_SSTAT 和 CH*n*_DSTAT 回写没有启用，不要使用 PSRAM 来存储 LLI。
- LLI 回写操作后，DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 将为 0。

图 7.5-6 显示了 CH*n*_LLP_STATUS 回写字段：

63 0

*	Reserved[61:47]	Data Left in Channel FIFO (CHn_Status[46:32])	Reserved[31:22]	Completed Block Transfer Size (CHn_Status[21:0])

说明：* 代表 Status Indication (CHn_IntStatus[1:0])

图 7.5-6. CH*n*_LLP_STATUS 回写字段

CH*n*_LLP_STATUS[63] 和 CH*n*_LLP_STATUS[62] 分别表示 DMA_TFR_DONE 和 BLOCK_TFR_DONE 状态。这两个字段是 LLI 回写过程中最后更新的字段。软件应该确保 BLOCK_TFR_DONE 为 1 再使用 CH*n*_SSTAT 和 CH*n*_DSTAT 信息。DMA_TFR_DONE 和 BLOCK_TFR_DONE 在传输最后一个块后会被置为 1。

CH*n*_STATUS[21:0] 对应 DMAC_CH*n*_CMPLTD_BLK_TFR_SIZE。CH*n*_STATUS[46:32] 对应 DMAC_CH*n*_DATA_LEFT_IN_FIFO[14:0]。

块间传输暂停

在每个块传输结束时，如果满足以下条件，则产生块传输完成中断 DMAC_CH*n*_BLOCK_TFR_DONE_INT：

- 使能全局中断（DMAC_INT_EN 置 1）
- 使能通道块传输完成中断（DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSTAT 置 1，DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSIGNAL 置 1，DMAC_CH*n*_IOC_BLKTFR 置 1）

对于基于连续地址和自动重新加载的多块传输（源和目标外设都不使用基于影子寄存器或链表的多块传输），DMA 传输会在中断 DMAC_CH*n*_BLOCK_TFR_DONE_INT 置 1 后自动停止。VDMA 不会继续进行下一个块传输，直到软件将 DMAC_CH*n*_CLEAR_BLOCK_TFR_DONE_INTSTAT 置 1 清除中断。

块之间的通道暂停用于确保倒数第二个块的块传输完成 ISR（中断服务例程）在最后一个块传输开始之前得到响应。这确保 ISR 在最后一个块传输完成之前清零 DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE。

多块传输结束

如果源外设或目标外设使用基于影子寄存器或链表的多块传输，则指示位 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 表明当前块是否是最后一个。如果

DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 为 1，VDMA 认为当前块是最后一块，并在当前块传输结束时完成 DMA 传输操作。

对于基于连续地址和自动重新加载的多块传输（源和目标外设都不使用基于影子寄存器或链表的多块传输），如果对应的多块类型寄存器 DMAC_CH*n*_SRC_MULTBLK_TYPE 和/或 DMAC_CH*n*_DST_MULTBLK_TYPE 在块传输结束时为 0，VDMA 认为前一个块为最后一个块，并完成 DMA 传输操作。

7.5.5 流控制器

DMA 块传输的长度由流控制器决定。流控制器可以是 VDMA、源或目标外设，存储器不能用作流控制器。

如果在启用通道之前知道块的长度，那么应该将 VDMA 配置为流控制器。如果在启用通道之前不知道块的长度，源或目标外设应该被配置为流控制器，并由其决定何时终止 DMA 块传输。

7.5.6 通道挂起和恢复

软件可以在 DMA 传输时挂起和恢复通道。挂起通道的操作如下：

1. 软件将 DMAC_CH*n*_SUSP 置为 1。
2. VDMA 在完成源已发起的所有 AXI 传输后，暂停来自源的所有传输。
3. VDMA 将 DMAC_CH*n*_CH_SRC_SUSPENDED_INT 置为 1 表示源传输被挂起并产生一个挂起中断。
4. VDMA 将通道 FIFO 里的数据全部传输给目标。当 DMAC_CH*n*_SRC_TR_WIDTH < DMAC_CH*n*_DST_TR_WIDTH 并且 DMAC_CH*n*_SUSP 为 1 时，通道 FIFO 中可能还有数据，但不足以形成一次 DMAC_CH*n*_DST_TR_WIDTH 宽度的单次传输。当通道恢复后，FIFO 中剩余的数据会被传输到目标。
5. VDMA 将 DMAC_CH*n*_CH_SUSPENDED_INT 置为 1 表示通道被挂起，并产生中断。

通道被挂起后，软件可以将 DMAC_CH*n*_SUSP 置 0，退出通道挂起模式。然后 VDMA 将恢复 DMA 传输。

7.5.7 通道禁用

在正常情况下，软件通过将 DMAC_CH*n*_EN 置为 1 来启用一个通道，VDMA 在通道传输结束时清零 DMAC_CH*n*_EN。软件可以在传输完成之前禁用通道。

7.5.7.1 传输完成前禁用被挂起的通道

要在传输完成前禁用被挂起的通道：

- 在 VDMA 将 DMAC_CH*n*_CH_SUSPENDED_INT 置 1 后将 DMAC_CH*n*_EN 置为 0。当 DMAC_CH*n*_SRC_TR_WIDTH < DMAC_CH*n*_DST_TR_WIDTH 并且 DMAC_CH*n*_SUSP 为 1 时，通道 FIFO 中可能还有数据，但不足以形成一次 DMAC_CH*n*_DST_TR_WIDTH 宽度的单次传输。在这种情况下，FIFO 中剩余的数据不会被传输到目标，会丢失。
- VDMA 将 DMAC_CH*n*_CH_DISABLED_INT 置为 1 表示通道被禁用，并产生中断。

7.5.7.2 传输完成前禁用未被挂起的通道

要在传输完成前禁用未被挂起的通道：

- 软件在 DMA 传输时将 DMAC_CH*n*_EN 置 0。
- VDMA 在完成源已发起的所有 AXI 传输后，停止来自源的所有传输。

- VDMA 将通道 FIFO 里的数据全部传输给目标。当 DMAC_CH*n*_SRC_TR_WIDTH < DMAC_CH*n*_DST_TR_WIDTH 并且 DMAC_CH*n*_EN 为 0 时，通道 FIFO 中可能还有数据，但不足以形成一次 DMAC_CH*n*_DST_TR_WIDTH 宽度的单次传输。在这种情况下，FIFO 中剩余的数据不会被传输到目标，会丢失。
- VDMA 将 DMAC_CH*n*_CH_DISABLED_INT 置为 1 表示通道被禁用，并产生中断。

说明：
一旦软件将 DMAC_CH*n*_EN 置 0，发起了通道禁用程序，用户无法在 DMAC_CH*n*_CH_DISABLED_INT 置 1 之前将 DMAC_CH*n*_EN 置 1 来重新使能通道。VDMA 会忽略这个写操作。

7.5.8 低功耗技术

VDMA 支持低功耗技术，这是一种智能低功耗逻辑，可以在各种场景下自动感知多个 DMA 通道、从机总线接口和 AXI 主机接口通道上的空闲时间，然后切断这些模块的时钟，从而降低总体功耗。低功耗逻辑反应迅速，可以在监测到任何活动时向这些模块提供时钟，而不会影响性能。

DMAC_LOWPOWER_CFGO_REG 可以用来启用低功耗并控制低功耗计数器。

7.5.8.1 DMA 通道低功耗技术

DMA 通道占用了 VDMA 上的大部分面积，比其他模块消耗更多的功耗，因此更加需要优化其功耗。

DMA 通道低功耗技术使用 DMA 通道低功耗状态机监测通道上的活动。每当监测到 DMA 通道的低功耗条件时，DMA 通道低功耗延迟计数器就开始启动。状态机等待直到计数器到期，然后 DMA 通道通过关闭 VDMA 的时钟 AXI_DMAC_CORE_CLK 进入低功耗状态。DMA 通道一直保持这个状态，直到在通道上检测到任何活动。在某些情况下，如果在延迟计数器运行时监测到活动，则计数器将重置并停止计数，直到再次检测到低功耗条件。

7.5.8.2 从机总线接口上的低功耗技术

从机总线接口单元 (SBIU) 通过 APB3 从机接口访问 VDMA 的内部寄存器。低功耗技术可以优化 SBIU 的功耗。

从机总线接口上的低功耗技术使用 SBIU 状态机监测从机总线接口上的活动。一旦检测到低功耗条件（即没有 APB 读写传输正在进行并且 SBIU 状态机处于空闲状态），SBIU 低功耗延迟计数器将开始计数。当延迟计数器到期时，SBIU 通过关闭 PCLK 或 AXI_DMAC_CORE_CLK（根据时钟模式配置）进入低功耗状态。SBIU 一直保持这种状态，直到在从机总线接口上检测到读或写传输。在某些情况下，如果在延迟计数器运行时监测到传输，则计数器将重置并停止计数，直到再次检测到低功耗条件。

如果在 SBIU 处于低功耗状态时检测到读或写传输，则低功耗逻辑会立即响应，打开 PCLK 或 AXI_DMAC_CORE_CLK 从而退出低功耗状态。

7.5.8.3 AXI 主机接口通道上的低功耗技术

AXI 主机接口通过 AXI 总线在存储器和外设之间传输数据。

每当检测到 AXI 主机接口通道上的低功耗条件时，AXI 主机接口通道低功耗延迟计数器开始计时。AXI 低功耗逻辑等待直到延迟计数器到期，然后 AXI 主机接口通道关闭 AXI 主机接口的时钟和 AXI_DMAC_CORE_CLK 进入低功耗状态。AXI 主机接口通道一直保持这个状态，直到在通道上检测到任何活动。在某些情况下，如果在延迟计数器运行时监测到活动，则计数器将重置并停止计数，直到再次检测到低功耗条件。

7.5.8.4 全局低功耗技术

全局低功耗技术根据 DMA 通道、从机总线接口和 AXI 主机接口通道的空闲状态实现低功耗选项。如果上述模块都指示 VDMA 处于空闲状态，则全局低功耗延迟计数器将启动。

全局低功耗逻辑等待直到低功耗延迟计数器到期，然后 VDMA 关闭所有模块的时钟来进入低功耗状态。VDMA 一致保持这个状态，直到在任意模块上检测到任何活动。在某些情况下，如果在延迟计数器运行时监测到活动，则计数器将重置并停止计数，直到再次检测到低功耗条件。

7.6 中断

ESP32-P4 的 VDMA 可以生成 AXI_DMAC_INTR 中断信号并发送给 中断矩阵。

中断信号由 VDMA 的内部中断源生成。

- 一般中断：
  - DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INT：从机接口未定义寄存器解码错误中断
  - DMAC_SLVIF_COMMONREG_RD2WO_ERR_INT：从机接口一般寄存器读入只写错误中断
  - DMAC_SLVIF_COMMONREG_WR2RO_ERR_INT：从机接口一般寄存器写到只读错误中断
  - DMAC_SLVIF_COMMONREG_DEC_ERR_INT：从机接口一般寄存器解码错误中断
- 通道中断 (*n* = 1 ~ 4)：
  - DMAC_CH*n*_CH_ABORTED_INT：通道中止中断
  - DMAC_CH*n*_CH_DISABLED_INT：通道禁用中断
  - DMAC_CH*n*_CH_SUSPENDED_INT：通道挂起中断
  - DMAC_CH*n*_CH_SRC_SUSPENDED_INT：通道源挂起中断
  - DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INT：影子寄存器写有效错误中断
  - DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INT：从机接口写启用通道错误中断
  - DMAC_CH*n*_SLVIF_RD2RWO_ERR_INT：从机接口读到只写错误中断
  - DMAC_CH*n*_SLVIF_WR2RO_ERR_INT：从机接口写入只读错误中断
  - DMAC_CH*n*_SLVIF_DEC_ERR_INT：从机接口解码错误中断
  - DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INT：从机接口多块类型错误中断
  - DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT：影子寄存器或 LLI 无效错误中断
  - DMAC_CH*n*_LLI_WR_SLV_ERR_INT：LLI 写从机错误中断
  - DMAC_CH*n*_LLI_RD_SLV_ERR_INT：LLI 读从机错误中断
  - DMAC_CH*n*_LLI_WR_DEC_ERR_INT：LLI 写解码错误中断
  - DMAC_CH*n*_LLI_RD_DEC_ERR_INT：LLI 读解码错误中断
  - DMAC_CH*n*_DST_SLV_ERR_INT：目标从机错误中断
  - DMAC_CH*n*_SRC_SLV_ERR_INT：源从机错误中断
  - DMAC_CH*n*_DST_DEC_ERR_INT：目标解码错误中断

- DMAC_CH*n*_SRC_DEC_ERR_INT：源解码错误中断
- DMAC_CH*n*_DST_TRANSCOMP_INT：目标传输完成中断
- DMAC_CH*n*_SRC_TRANSCOMP_INT：源传输完成中断
- DMAC_CH*n*_DMA_TFR_DONE_INT：DMA 传输完成中断
- DMAC_CH*n*_BLOCK_TFR_DONE_INT：块传输完成中断

7.7 配置流程

7.7.1 通用配置流程

以下两个步骤是通用的配置流程，应在进行其他配置之前执行。

1. 配置 HP_SYS_CLKRST_GDMA_SYS_CLK_EN 为 1，使能 VDMA 的总线时钟。
2. 通过 TEE_DMA_GDMA_CH*n*_W_PMS 和 TEE_DMA_GDMA_CH*n*_R_PMS 配置 VDMA 的访问权限。更多信息请查看章节 *20 权限控制 (PMS)*。

7.7.2 配置基于影子寄存器的多块传输

1. 读取 DMAC_CH*n*_EN 选择一个可用的通道。
2. 配置 DMAC_CH*n*_SRC_MULTBLK_TYPE 和/或 DMAC_CH*n*_DST_MULTBLK_TYPE 为 2，选择源传输和/或目标传输为基于影子寄存器的多块传输。
3. 通过 DMAC_CH*n*_SARO_REG 和/或 DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG、DMAC_CH*n*_CTL1_REG 配置第一个块。VDMA 将这些值加载到相应的影子寄存器。

   注意：必须最后再将 DMAC_CH*n*_CTL1_REG 的 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 位配置为 1。
4. 配置 DMAC_CH*n*_EN 为 1，使能相应通道。
5. VDMA 根据配置发起 DMA 块传输操作。
   - 块传输可能立即启动，也可能在握手请求之后启动，这取决于 DMAC_CH*n*_TT_FC。具体为，
     - 当源是存储器的时候，源传输就立即启动；当目标是存储器的时候，目标传输就立即启动。
     - 当源是外设时，从源获取数据需要等待与源的握手之后；当目标是外设时，往目标写入数据需要等待与目标的握手之后。
   - VDMA 检查 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 的值，如果为 0，VDMA 等待 DMAC_CH*n*_BLK_TFR_RESUMEREQ 置 1 来指示影子寄存器可用，然后再次尝试影子寄存器获取操作。在这种情况下，VDMA 可能生成 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT 中断。
   - VDMA 检查 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 的值，如果为 1，VDMA 将影子寄存器内容复制到用于执行 DMA 传输的寄存器中（DMAC_CH*n*_SARO_REG 和/或 DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG），并且清零 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID。
     - 如果 VDMA 读到 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 为 1，则认为当前块是最后一块，并在当前块传输结束时完成 DMA 传输操作。

- 如果 VDMA 读到 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 为 0，则认为还有一个或多个块需要传输，并在当前块传输结束时再次检查 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID。

6. 软件等待 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 自动清零。
7. 通过 DMAC_CH*n*_SARO_REG 和/或 DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG、DMAC_CH*n*_CTL1_REG 配置下一个块的信息。若下一个块是最后一个块，需要将 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 配置为 1。

   注意：必须最后再将 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 配置为 1。
8. 软件等待 DMAC_CH*n*_BLOCK_TFR_DONE_INT 中断，或者轮询块传输完成状态指示位 DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT 直到读到 1。之后回到步骤 6 循环。

说明：如果产生了 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT 中断，推荐按照以下步骤恢复传输：

- 通过 DMAC_CH*n*_SARO_REG 和/或 DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG 配置下一个块。
- 配置 DMAC_CH*n*_CLEAR_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT 清除中断。
- 配置 DMAC_CH*n*_BLK_TFR_RESUMEREQ 为 1 请求恢复块传输。

7.7.3 配置基于链表的多块传输

1. 读取 DMAC_CH*n*_EN 选择一个可用的通道。
2. 配置 DMAC_CH*n*_SRC_MULTBLK_TYPE 和/或 DMAC_CH*n*_DST_MULTBLK_TYPE 为 3，选择源传输和/或目标传输为基于链表的多块传输。
3. 通过 DMAC_CH*n*_LLPO_REG 配置第一个链表项的基地址，以及链表所在的主机接口。
4. 在系统存储器中创建一个或多个链表项。可以提前创建这个链表项或者使用 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 和 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 动态地扩展链表。动态扩展链表的具体操作为：
   - 在 DMA 传输过程中，当链表没有准备好时，软件可以把链表中 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 置 0，然后等待准备好链表后，再置 1，用于拓展链表。
   - 当要结束 DMA 传输时，把最后一个链表的 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 置 1，来表示传输结束。
5. 配置 DMAC_CH*n*_EN 为 1，使能相应通道。

   说明：
   步骤 4 和 5 的顺序可以互换。但是，如果先执行步骤 5，或者在多块传输期间，下一个块传输的链表项在系统存储器中不可用（即获取的链表项的 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 为 0），则 VDMA 可能会产生 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT 中断。
6. VDMA 根据配置发起 DMA 块传输。块传输可能立即启动，也可能在硬件握手请求之后启动，这取决于 DMAC_CH*n*_TT_FC。VDMA 将链表内容复制到用于执行 DMA 传输的寄存器中（DMAC_CH*n*_SARO_REG 和/或 DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG），然后发起 DMA 块传输。

7. 在链表获取阶段：
    - 如果 VDMA 读到 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 为 1，则认为当前块是最后一个块，并在当前块传输结束时完成 DMA 传输。
    - 如果 VDMA 读到 DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 为 0，则认为还有一个或多个块需要传输，并且回到步骤 6。
    - 如果 VDMA 读到 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 为 0，则可能产生 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT 中断。VDMA 等待直到软件将 DMAC_CH*n*_BLK_TFR_RESUMEREQ 置 1 来指示链表的可用性，然后再尝试读取链表。

7.7.4 配置单块传输

1. 读取 DMAC_CH*n*_EN 选择一个可用的通道。
2. 配置 DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE 为 0，选择连续地址的单块传输。
3. 配置 DMAC_CH*n*_SARO_REG 和/或 DMAC_CH*n*_DARO_REG、DMAC_CH*n*_BLOCK_TS、DMAC_CH*n*_CTLO_REG 和 DMAC_CH*n*_CTL1_REG。
4. 配置 DMAC_CH*n*_EN 为 1 使能相应通道。
5. 软件等待 DMAC_CH*n*_BLOCK_TFR_DONE_INT 中断，或轮询块传输完成状态指示位 DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT 直到读到 1。

7.8 寄存器列表

本小节的所有地址均为相对于 VDMA 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
版本寄存器			
DMAC_IDO_REG	VDMA ID 编号	0x0000	RO
DMAC_COMPVERO_REG	VDMA 版本编号	0x0008	RO
配置寄存器			
DMAC_CFGO_REG	VDMA 配置寄存器	0x0010	R/W
DMAC_CHENO_REG	VDMA 通道使能寄存器	0x0018	varies
DMAC_RESETO_REG	VDMA 复位控制寄存器	0x0058	R/W
DMAC_LOWPOWER_CFGO_REG	VDMA 低功耗配置寄存器 0	0x0060	R/W
DMAC_LOWPOWER_CFG1_REG	VDMA 低功耗配置寄存器 1	0x0064	R/W
DMAC_CH1_SARO_REG	VDMA 通道 1 源地址寄存器	0x0100	R/W
DMAC_CH1_DARO_REG	VDMA 通道 1 目标地址寄存器	0x0108	R/W
DMAC_CH1_BLOCK_TSO_REG	VDMA 通道 1 块传输大小寄存器	0x0110	R/W
DMAC_CH1_CTLO_REG	VDMA 通道 1 控制寄存器 0	0x0118	R/W
DMAC_CH1_CTL1_REG	VDMA 通道 1 控制寄存器 1	0x011C	R/W
DMAC_CH1_CFGO_REG	VDMA 通道 1 配置寄存器 0	0x0120	varies
DMAC_CH1_CFG1_REG	VDMA 通道 1 配置寄存器 1	0x0124	varies
DMAC_CH1_LLPO_REG	VDMA 通道 1 链表指针寄存器	0x0128	R/W
DMAC_CH1_BLK_TFR_RESUMEREQO_REG	VDMA 通道 1 块传输恢复请求寄存器	0x0148	WO
DMAC_CH1_AXI_IDO_REG	VDMA 通道 1 AXI ID 寄存器	0x0150	R/W
DMAC_CH2_SARO_REG	VDMA 通道 2 源地址寄存器	0x0200	R/W
DMAC_CH2_DARO_REG	VDMA 通道 2 目标地址寄存器	0x0208	R/W
DMAC_CH2_BLOCK_TSO_REG	VDMA 通道 2 块传输大小寄存器	0x0210	R/W
DMAC_CH2_CTLO_REG	VDMA 通道 2 控制寄存器 0	0x0218	R/W
DMAC_CH2_CTL1_REG	VDMA 通道 2 控制寄存器 1	0x021C	R/W
DMAC_CH2_CFGO_REG	VDMA 通道 2 配置寄存器 0	0x0220	varies
DMAC_CH2_CFG1_REG	VDMA 通道 2 配置寄存器 1	0x0224	varies
DMAC_CH2_LLPO_REG	VDMA 通道 2 链表指针寄存器	0x0228	R/W
DMAC_CH2_BLK_TFR_RESUMEREQO_REG	VDMA 通道 2 块传输恢复请求寄存器	0x0248	WO
DMAC_CH2_AXI_IDO_REG	VDMA 通道 2 AXI ID 寄存器	0x0250	R/W
DMAC_CH3_SARO_REG	VDMA 通道 3 源地址寄存器	0x0300	R/W
DMAC_CH3_DARO_REG	VDMA 通道 3 目标地址寄存器	0x0308	R/W
DMAC_CH3_BLOCK_TSO_REG	VDMA 通道 3 块传输大小寄存器	0x0310	R/W
DMAC_CH3_CTLO_REG	VDMA 通道 3 控制寄存器 0	0x0318	R/W
DMAC_CH3_CTL1_REG	VDMA 通道 3 控制寄存器 1	0x031C	R/W
DMAC_CH3_CFGO_REG	VDMA 通道 3 配置寄存器 0	0x0320	varies
DMAC_CH3_CFG1_REG	VDMA 通道 3 配置寄存器 1	0x0324	varies
DMAC_CH3_LLPO_REG	VDMA 通道 3 链表指针寄存器	0x0328	R/W

名称	描述	地址	访问
DMAC_CH3_BLK_TFR_RESUMEREQO_REG	VDMA 通道 3 块传输恢复请求寄存器	0x0348	WO
DMAC_CH3_AXI_IDO_REG	VDMA 通道 3 AXI ID 寄存器	0x0350	R/W
DMAC_CH4_SARO_REG	VDMA 通道 4 源地址寄存器	0x0400	R/W
DMAC_CH4_DARO_REG	VDMA 通道 4 目标地址寄存器	0x0408	R/W
DMAC_CH4_BLOCK_TSO_REG	VDMA 通道 4 块传输大小寄存器	0x0410	R/W
DMAC_CH4_CTLO_REG	VDMA 通道 4 控制寄存器 0	0x0418	R/W
DMAC_CH4_CTL1_REG	VDMA 通道 4 控制寄存器 1	0x041C	R/W
DMAC_CH4_CFGO_REG	VDMA 通道 4 配置寄存器 0	0x0420	varies
DMAC_CH4_CFG1_REG	VDMA 通道 4 配置寄存器 1	0x0424	varies
DMAC_CH4_LLPO_REG	VDMA 通道 4 链表指针寄存器	0x0428	R/W
DMAC_CH4_BLK_TFR_RESUMEREQO_REG	VDMA 通道 4 块传输恢复请求寄存器	0x0448	WO
DMAC_CH4_AXI_IDO_REG	VDMA 通道 4 AXI ID 寄存器	0x0450	R/W
中断寄存器			
DMAC_INTSTATUSO_REG	VDMA 中断状态寄存器	0x0030	RO
DMAC_COMMONREG_INTCLEARO_REG	VDMA 一般中断清除寄存器	0x0038	WO
DMAC_COMMONREG_INTSTATUS_ENABLEO_REG	VDMA 一般中断状态使能寄存器	0x0040	varies
DMAC_COMMONREG_INTSIGNAL_ENABLEO_REG	VDMA 一般中断信号使能寄存器	0x0048	varies
DMAC_COMMONREG_INTSTATUSO_REG	VDMA 一般中断状态寄存器	0x0050	RO
DMAC_CH1_INTSTATUS_ENABLEO_REG	VDMA 通道 1 中断状态使能寄存器	0x0180	varies
DMAC_CH1_INTSTATUSO_REG	VDMA 通道 1 中断状态寄存器	0x0188	RO
DMAC_CH1_INTSIGNAL_ENABLEO_REG	VDMA 通道 1 中断信号使能寄存器	0x0190	varies
DMAC_CH1_INTCLEARO_REG	VDMA 通道 1 中断清除寄存器	0x0198	WO
DMAC_CH2_INTSTATUS_ENABLEO_REG	VDMA 通道 2 中断状态使能寄存器	0x0280	varies
DMAC_CH2_INTSTATUSO_REG	VDMA 通道 2 中断状态寄存器	0x0288	RO
DMAC_CH2_INTSIGNAL_ENABLEO_REG	VDMA 通道 2 中断信号使能寄存器	0x0290	varies
DMAC_CH2_INTCLEARO_REG	VDMA 通道 2 中断清除寄存器	0x0298	WO
DMAC_CH3_INTSTATUS_ENABLEO_REG	VDMA 通道 3 中断状态使能寄存器	0x0380	varies
DMAC_CH3_INTSTATUSO_REG	VDMA 通道 3 中断状态寄存器	0x0388	RO
DMAC_CH3_INTSIGNAL_ENABLEO_REG	VDMA 通道 3 中断信号使能寄存器	0x0390	varies
DMAC_CH3_INTCLEARO_REG	VDMA 通道 3 中断清除寄存器	0x0398	WO
DMAC_CH4_INTSTATUS_ENABLEO_REG	VDMA 通道 4 中断状态使能寄存器	0x0480	varies
DMAC_CH4_INTSTATUSO_REG	VDMA 通道 4 中断状态寄存器	0x0488	RO
DMAC_CH4_INTSIGNAL_ENABLEO_REG	VDMA 通道 4 中断信号使能寄存器	0x0490	varies
DMAC_CH4_INTCLEARO_REG	VDMA 通道 4 中断清除寄存器	0x0498	WO
状态寄存器			
DMAC_CH1_STATUSO_REG	VDMA 通道 1 状态寄存器 0	0x0130	RO
DMAC_CH1_STATUS1_REG	VDMA 通道 1 状态寄存器 1	0x0134	RO
DMAC_CH1_SSTATO_REG	VDMA 通道 1 源状态寄存器	0x0160	RO
DMAC_CH1_DSTATO_REG	VDMA 通道 1 目标状态寄存器	0x0168	RO
DMAC_CH1_SSTATARO_REG	VDMA 通道 1 源状态地址寄存器	0x0170	R/W
DMAC_CH1_DSTATARO_REG	VDMA 通道 1 目标状态地址寄存器	0x0178	R/W
DMAC_CH2_STATUSO_REG	VDMA 通道 2 状态寄存器 0	0x0230	RO
DMAC_CH2_STATUS1_REG	VDMA 通道 2 状态寄存器 1	0x0234	RO

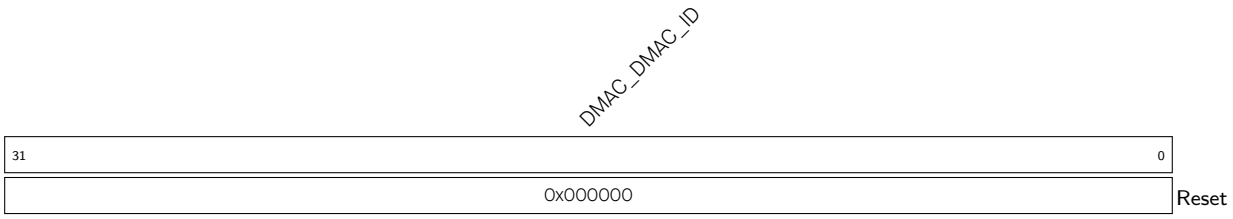
名称	描述	地址	访问
DMAC_CH2_SSTAT0_REG	VDMA 通道 2 源状态寄存器	0x0260	RO
DMAC_CH2_DSTAT0_REG	VDMA 通道 2 目标状态寄存器	0x0268	RO
DMAC_CH2_SSTATAR0_REG	VDMA 通道 2 源状态地址寄存器	0x0270	R/W
DMAC_CH2_DSTATAR0_REG	VDMA 通道 2 目标状态地址寄存器	0x0278	R/W
DMAC_CH3_STATUS0_REG	VDMA 通道 3 状态寄存器 0	0x0330	RO
DMAC_CH3_STATUS1_REG	VDMA 通道 3 状态寄存器 1	0x0334	RO
DMAC_CH3_SSTAT0_REG	VDMA 通道 3 源状态寄存器	0x0360	RO
DMAC_CH3_DSTAT0_REG	VDMA 通道 3 目标状态寄存器	0x0368	RO
DMAC_CH3_SSTATAR0_REG	VDMA 通道 3 源状态地址寄存器	0x0370	R/W
DMAC_CH3_DSTATAR0_REG	VDMA 通道 3 目标状态地址寄存器	0x0378	R/W
DMAC_CH4_STATUS0_REG	VDMA 通道 4 状态寄存器 0	0x0430	RO
DMAC_CH4_STATUS1_REG	VDMA 通道 4 状态寄存器 1	0x0434	RO
DMAC_CH4_SSTAT0_REG	VDMA 通道 4 源状态寄存器	0x0460	RO
DMAC_CH4_DSTAT0_REG	VDMA 通道 4 目标状态寄存器	0x0468	RO
DMAC_CH4_SSTATAR0_REG	VDMA 通道 4 源状态地址寄存器	0x0470	R/W
DMAC_CH4_DSTATAR0_REG	VDMA 通道 4 目标状态地址寄存器	0x0478	R/W

7.9 寄存器

本小节的所有地址均为相对于 VDMA 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

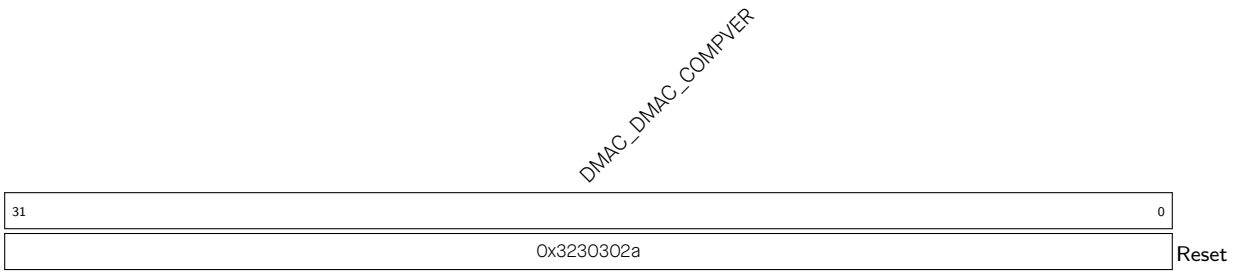
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 7.1. DMAC_IDO_REG (0x0000)



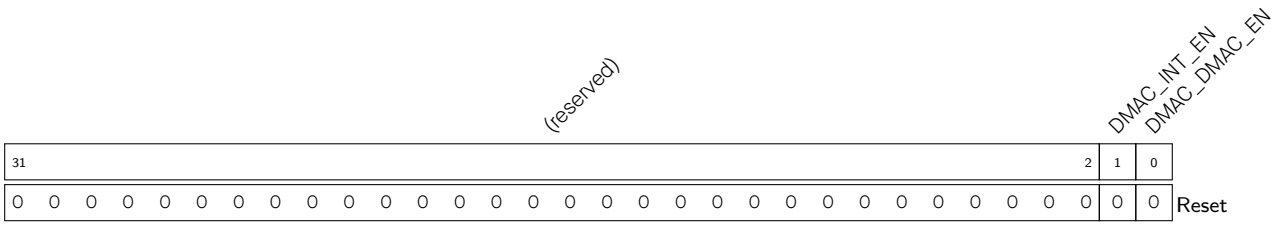
DMAC_DMAC_ID 表示 VDMA 的 ID 编号。(RO)

Register 7.2. DMAC_COMPVERO_REG (0x0008)



DMAC_DMAC_COMPVER 表示 VDMA 的版本。(RO)

Register 7.3. DMAC_CFG0_REG (0x0010)



DMAC_DMAC_EN 配置是否启用 VDMA。

0：禁用

1：启用

说明：如果在通道仍然活跃的情况下此位被清零，那么此位仍然返回 1，表示有通道仍然活跃，直到硬件终止所有通道上的活动。此时，此位为 0。

(R/W)

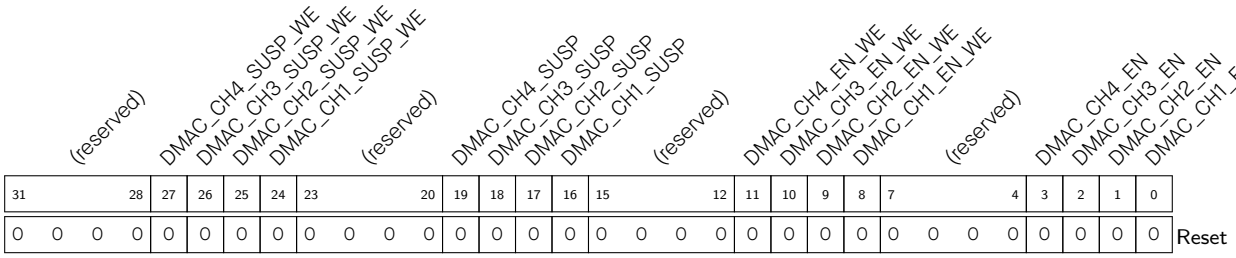
DMAC_INT_EN 配置是否启用全局中断。

0：禁用

1：启用

(R/W)

Register 7.4. DMAC_CHENO_REG (0x0018)



DMAC_CH*n*_EN (*n*: 1-4) 配置是否启用通道 *n*。当 DMA 传输的最后一个 AXI 传输结束时，硬件自动将此位清零。因此软件可以轮询此位来确定何时这个通道可以用于新的 DMA 传输。

0：禁用通道 *n*

1：启用通道 *n*

(R/W)

DMAC_CH*n*_EN_WE (*n*: 1-4) 配置是否使能写入 DMAC_CH*n*_EN。此位回读的值始终为 0。

0：禁止写入 DMAC_CH*n*_EN

1：使能写入 DMAC_CH*n*_EN

(WO)

DMAC_CH*n*_SUSP (*n*: 1-4) 配置是否挂起通道 *n*。

0：不挂起通道 *n*

1：挂起通道 *n*

VDMA 将 DMAC_CH*n*_CH_SUSPENDED_INTSTAT 置为 1 后，软件可以将此位置为 0，从而退出通道挂起模式。

说明：当通道 *n* 被禁用时此位被清零。

(R/W)

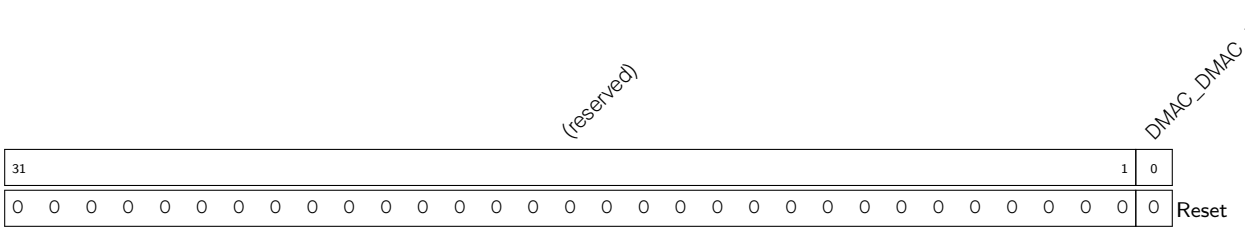
DMAC_CH*n*_SUSP_WE (*n*: 1-4) 配置是否使能写入 DMAC_CH*n*_SUSP。此位回读的值始终为 0。

0：禁止写入 DMAC_CH*n*_SUSP

1：使能写入 DMAC_CH*n*_SUSP

(WO)

Register 7.5. DMAC_RESETO_REG (0x0058)

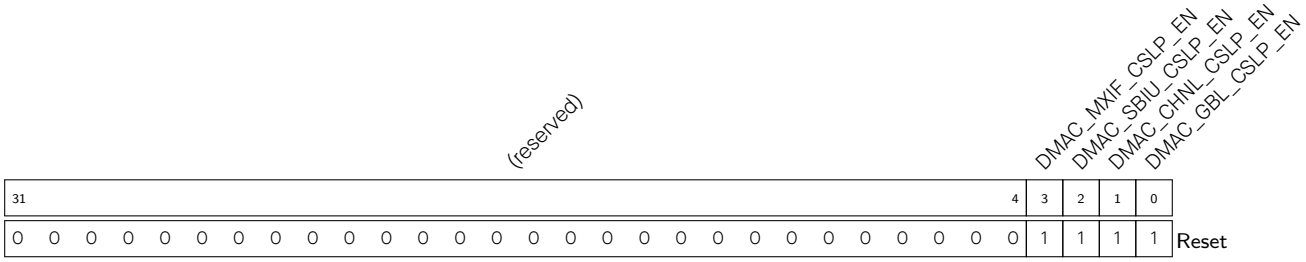


DMAC_DMAC_RST 配置是否复位 VDMA。软件将此位置为 1 来复位 VDMA，然后轮询此位，检查是否为 0。VDMA 将复位除了 APB 从机总线接口模块之外的其他所有模块并将此位清零。

说明：软件不允许向此位写 0。

(R/W)

Register 7.6. DMAC_LOWPOWER_CFG0_REG (0x0060)



DMAC_GBL_CSLP_EN 配置是否启用全局低功耗特性。

0：禁用

1：启用

(R/W)

DMAC_CHNL_CSLP_EN 配置是否启用通道低功耗特性。

0：禁用

1：启用

(R/W)

DMAC_SBIU_CSLP_EN 配置是否启用从机总线接口上的低功耗特性。

0：禁用

1：启用

(R/W)

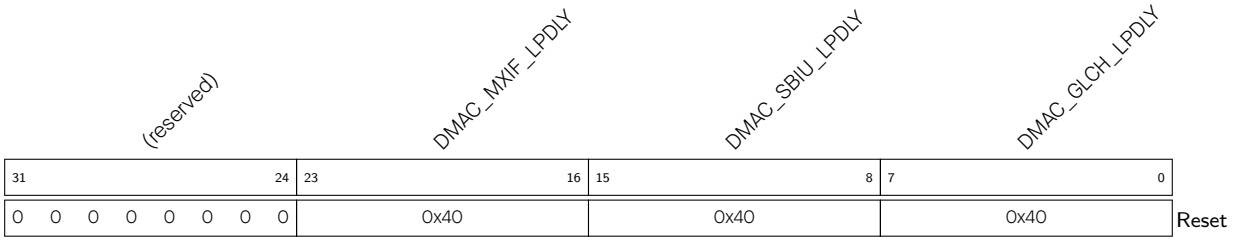
DMAC_MXIF_CSLP_EN 配置是否启用 AXI 主机接口上的低功耗特性。

0：禁用

1：启用

(R/W)

Register 7.7. DMAC_LOWPOWER_CFG1_REG (0x0064)

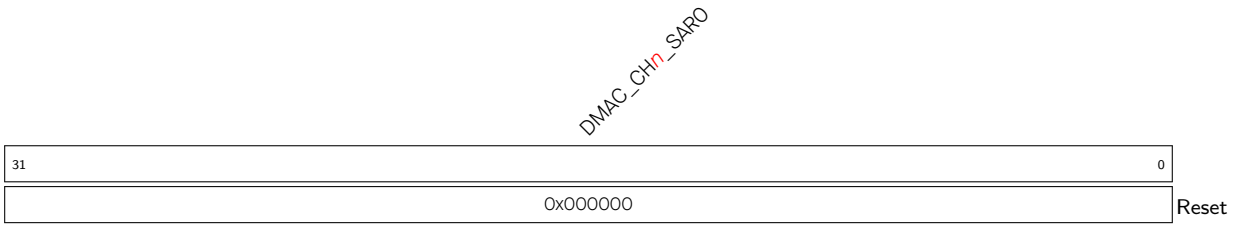


DMAC_GLCH_LPDLY 配置 AXI 主机接口低功耗延迟计数器的值。写入值必须大于或等于 0x4，如果小于 0x4，则此位重置为 0x40。(R/W)

DMAC_SBIU_LPDLY 配置 SBIU 低功耗延迟计数器的值。写入值必须大于或等于 0x4，如果小于 0x4，则此位重置为 0x40。(R/W)

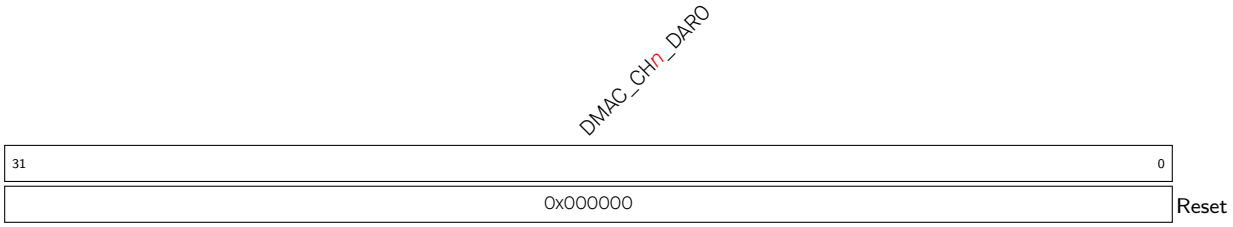
DMAC_MXIF_LPDLY 配置全局和 VDMA 通道低功耗延迟计数器的值。写入值必须大于或等于 0x4，如果小于 0x4，则此位重置为 0x40。(R/W)

Register 7.8. DMAC_CH*n*_SAR0_REG (*n*: 1-4) (0x0100**n*)



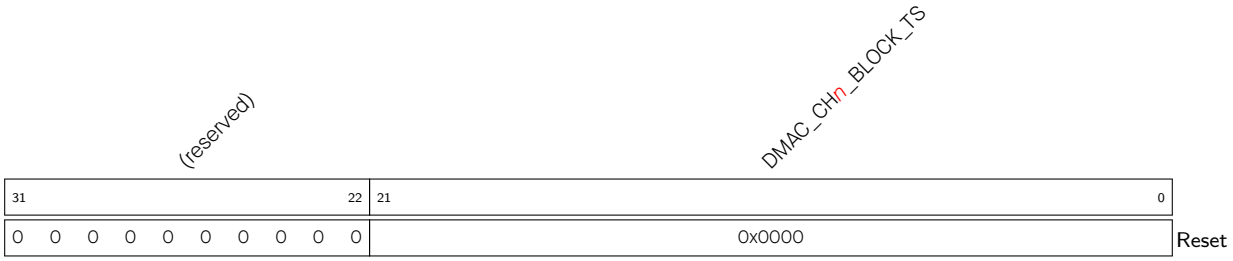
DMAC_CH*n*_SAR0 配置 DMA 传输的源地址。每次源传输后此位被更新。DMAC_CH*n*_SINC 决定块传输期间地址是否递增或保持不变。(R/W)

Register 7.9. DMAC_CH*n*_DAR0_REG (*n*: 1-4) (0x0100**n* + 0x0008)



DMAC_CH*n*_DAR0 配置 DMA 传输的目标地址。每次目标传输后此位被更新。DMAC_CH*n*_DINC 决定块传输期间地址是否递增或保持不变。(R/W)

Register 7.10. DMAC_CH*n*_BLOCK_TS0_REG (*n*: 1-4) (0x0100**n* + 0x0010)

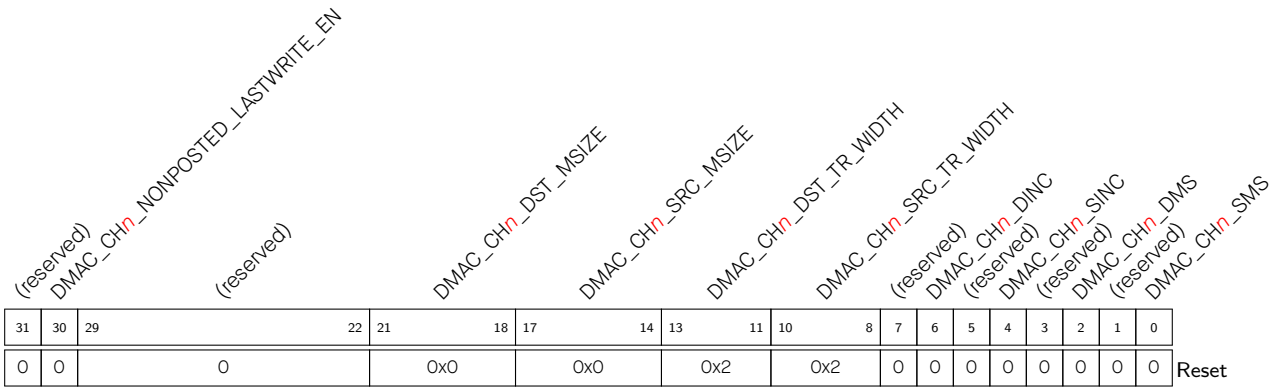


DMAC_CH*n*_BLOCK_TS 配置 DMA 块传输中 DMAC_CH*n*_SRC_TR_WIDTH 的总个数，取值范围为 0 ~ 4,194,303。

块传输大小 = 配置值 + 1

(R/W)

Register 7.11. DMAC_CH*n*_CTL0_REG (*n*: 1-4) (0x0100*n* + 0x0018)



DMAC_CH*n*_SMS 配置访问源的 AXI 主机。

0：AXI 主机 1

1：AXI 主机 2

(R/W)

DMAC_CH*n*_DMS 配置访问目标的 AXI 主机。

0：AXI 主机 1

1：AXI 主机 2

(R/W)

DMAC_CH*n*_SINC 配置源地址是否递增。如果 VDMA 通过一个固定地址从源外设 FIFO 中读取数据，则配置此位为 1。

0：递增

1：不变

(R/W)

DMAC_CH*n*_DINC 配置目标地址是否递增。如果 VDMA 通过一个固定地址向源外设 FIFO 中写数据，则配置此位为 1。

0：递增

1：不变

(R/W)

DMAC_CH*n*_SRC_TR_WIDTH 配置源传输宽度。

0x0：8 位

0x1：16 位

0x2：32 位

0x3：64 位

(R/W)

见下页...

Register 7.11. DMAC_CH*n*_CTL0_REG (*n*: 1-4) (0x0100**n* + 0x0018)

接上页...

DMAC_CH*n*_DST_TR_WIDTH 配置目标传输宽度。

0x0：8 位
0x1：16 位
0x2：32 位
0x3：64 位
(R/W)

DMAC_CH*n*_SRC_MSIZE 配置源突发 transaction 长度。每次握手接口发起源突发 transaction 请求的时候，从源那里读到的宽度为 DMAC_CH*n*_SRC_TR_WIDTH 的数据项的个数。

0x0：1 个数据项
0x1：4 个数据项
0x2：8 个数据项
0x3：16 个数据项
0x4：32 个数据项
0x5：64 个数据项
0x6：128 个数据项
0x7：256 个数据项
0x8：512 个数据项
0x9：1024 个数据项
(R/W)

DMAC_CH*n*_DST_MSIZE 配置目标突发 transaction 长度。每次握手接口发起目标突发 transaction 请求的时候，从目标那里读到的宽度为 DMAC_CH*n*_DST_TR_WIDTH 的数据项的个数。

0x0：1 个数据项
0x1：4 个数据项
0x2：8 个数据项
0x3：16 个数据项
0x4：32 个数据项
0x5：64 个数据项
0x6：128 个数据项
0x7：256 个数据项
0x8：512 个数据项
0x9：1024 个数据项
(R/W)

见下页...

Register 7.11. DMAC_CH*n*_CTL0_REG (*n*: 1-4) (0x0100**n* + 0x0018)

接上页...

DMAC_CH*n*_NONPOSTED_LASTWRITE_EN 配置能否在块传输期间使用 non-posted 类型的写操作。

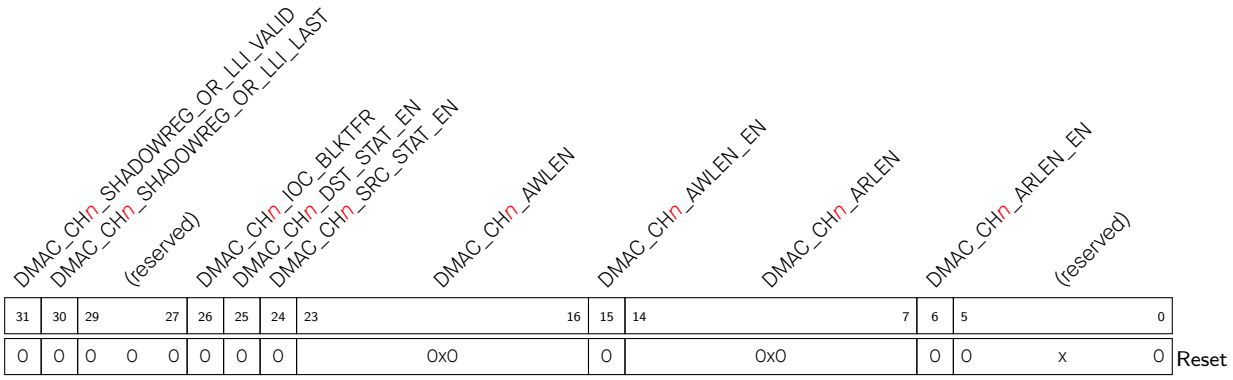
posted 类型指的是发送数据，不需要将数据写入到目标中（数据可能还处于链路中），VDMA 就可以认为该笔写请求完成。non-posted 类型指的是发送数据，需要将数据写入到目标中，并得到响应，VDMA 才可以认为该笔写请求完成。

0：可以在整个块传输期间使用 posted 类型的写操作

1：可以在块传输最后（在块内）使用 posted 类型的写操作。最后一次写必须是 non-posted 类型的，这是为了实现块完成中断的生成与最后写入数据到达目标之间的同步。

(R/W)

Register 7.12. DMAC_CH*n*_CTL1_REG (*n*: 1-4) (0x0100*n* + 0x001C)



DMAC_CH*n*_ARLEN_EN 配置是否使能源突发传输长度。

0：VDMA 使用任意小于或等于 16 的值作为源突发传输的长度。

1：VDMA 使用 DMAC_CH*n*_ARLEN 的值作为源突发传输长度，直到可能的最大程度；其余的传输使用最大可能突发传输长度。例如：数据量为 192 字节，设置源突发传输长度 DMAC_CH*n*_ARLEN 为 128 字节，那么传完 128 字节之后，剩余的 64 字节就是最大可能的突发传输长度。

(R/W)

DMAC_CH*n*_ARLEN 配置源突发传输长度。指定的突发传输长度用于源数据传输，直到可能的最大程度；其余的传输使用小于或等于 16 的最大可能值。

(R/W)

DMAC_CH*n*_AWLEN_EN 配置是否使能目标突发传输长度。

0：VDMA 使用任意小于或等于 16 的值作为目标突发传输长度。

1：VDMA 使用 DMAC_CH*n*_AWLEN 的值作为目标突发传输长度，直到可能的最大程度；其余的传输使用最大可能突发传输长度。

(R/W)

DMAC_CH*n*_AWLEN 配置目标突发传输长度。指定的突发传输长度用于目标数据传输，直到可能的最大程度；其余的传输使用小于或等于 16 的最大可能值。

(R/W)

DMAC_CH*n*_SRC_STAT_EN 配置是否使能获取源状态。

使能从由 DMAC_CH*n*_SSTATARO 指向的通道 *n* 的源外设中获取状态，并将其存储到 DMAC_CH*n*_SSTAT。如果源或目标外设使用基于链表的多块传输，则在每个块传输结束时将该值写回到链表的 DMAC_CH*n*_SSTAT。

0：不获取源设备状态

1：获取源设备状态并将值存储到 DMAC_CH*n*_SSTAT。

(R/W)

见下页...

Register 7.12. DMAC_CH*n*_CTL1_REG (*n*: 1-4) (0x0100*n* + 0x001C)

接上页...

DMAC_CH*n*_DST_STAT_EN 配置是否使能获取目标状态。

使能从由 DMAC_CH*n*_DSTATARO 指向的通道 *n* 的目标外设中获取状态，并将其存储到 DMAC_CH*n*_DSTAT。如果源或目标外设使用基于链表的多块传输，则在每个块传输结束时将该值写回到链表的 DMAC_CH*n*_DSTAT。

0：不获取目标设备状态

1：获取目标设备状态并将值存储到 DMAC_CH*n*_DSTAT。

(R/W)

DMAC_CH*n*_IOC_BLKTFR 配置是否启用基于影子寄存器或链表的多块传输的块传输完成中断。

说明： 如果源和目标使用基于连续地址或自动重新加载的多块传输，那么此字段的值无法按块修改。此外，在通道启用之前编程的值将用于 DMA 传输中的所有块。

0：禁用 DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT

1:启用 DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT,需要 DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSTAT 为 1；若需要输出 DMAC_CH*n*_BLOCK_TFR_DONE_INT 中断，需 DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSIGNAL 为 1。

(R/W)

DMAC_CH*n*_SHADOWREG_OR_LLI_LAST 配置影子寄存器或链表项是否为最后一个。

0：不是最后一个影子寄存器或链表项

1：是最后一个影子寄存器或链表项

(R/W)

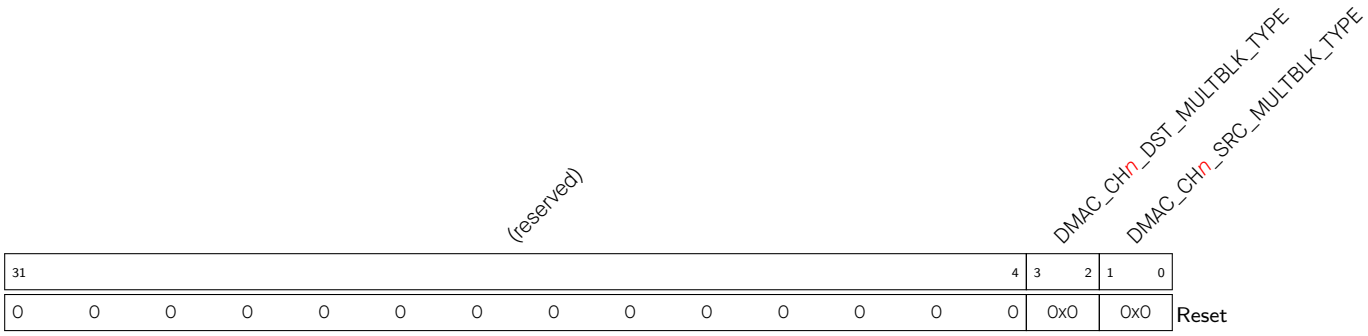
DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 配置影子寄存器内容或链表项是否有效。

0：无效

1：有效

(R/W)

Register 7.13. DMAC_CH*n*_CFG0_REG (*n*: 1-4) (0x0100*n* + 0x0120)



DMAC_CH*n*_SRC_MULTBLK_TYPE 配置源传输的多块传输类型。

0：连续地址

1：自动重新加载

2：影子寄存器

3：链表

(R/W)

DMAC_CH*n*_DST_MULTBLK_TYPE 配置目标传输的多块传输类型。

0：连续地址

1：自动重新加载

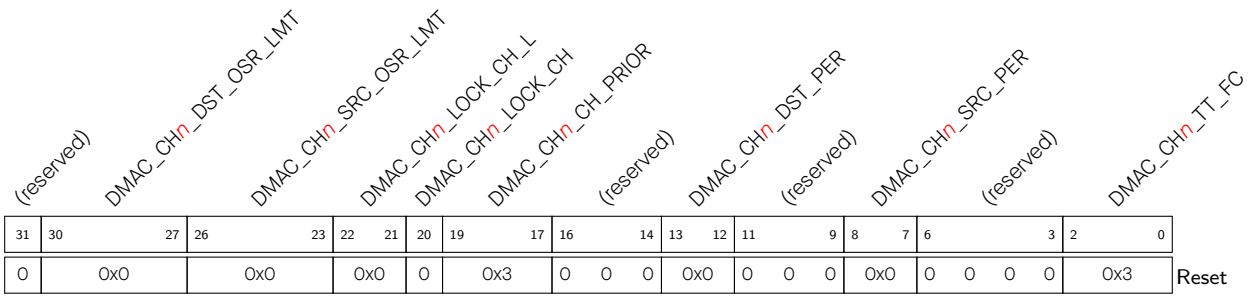
2：影子寄存器

3：链表

如果 DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE 都为 0，则为连续地址的单块传输或多块传输的最后一个块。

(R/W)

Register 7.14. DMAC_CH*n*_CFG1_REG (*n*: 1-4) (0x0100*n* + 0x0124)



DMAC_CH*n*_TT_FC 配置传输类型和流控制器。

0x0：传输类型为存储器到存储器，流控制器为 VDMA

0x1：传输类型为存储器到目标外设，流控制器为 VDMA

0x2：传输类型为源外设到存储器，流控制器为 VDMA

0x3：无效值

0x4：传输类型为源外设到存储器，流控制器为源外设

0x5：无效值

0x6：传输类型为存储器到目标外设，流控制器为目标外设

0x7：无效值

(R/W)

DMAC_CH*n*_SRC_PER 配置通道 *n* 源的握手接口 (0 ~ 2)。(R/W)

DMAC_CH*n*_DST_PER 配置通道 *n* 目标的握手接口 (0 ~ 2)。(R/W)

DMAC_CH*n*_CH_PRIOR 配置通道优先级 (0 ~ 3)，3 为最高优先级，0 为最低优先级。范围以外的取值会导致错误行为。(R/W)

DMAC_CH*n*_LOCK_CH VDMA 不支持锁定。读此位将始终返回 0。(RO)

DMAC_CH*n*_LOCK_CH_L VDMA 不支持锁定。读此位将始终返回 0。(RO)

DMAC_CH*n*_SRC_OSR_LMT 配置源 AXI Outstanding 请求的上限。最多支持 16 个 AXI outstanding 请求。

源 AXI Outstanding 请求的上限 = 配置值 + 1。

AXI Outstanding 是 AXI 协议中的一个特性，可以在接收到上一个请求响应之前发送请求，提高总线接口的带宽。(R/W)

DMAC_CH*n*_DST_OSR_LMT 配置目标 AXI Outstanding 请求的上限。最多支持 16 个 AXI outstanding 请求。

目标 AXI Outstanding 请求的上限 = 配置值 + 1。

(R/W)

Register 7.15. DMAC_CH*n*_LLP0_REG (*n*: 1-4) (0x0100*n + 0x0128)



DMAC_CH*n*_LMS 配置读写链表项所用的 AXI 主机。

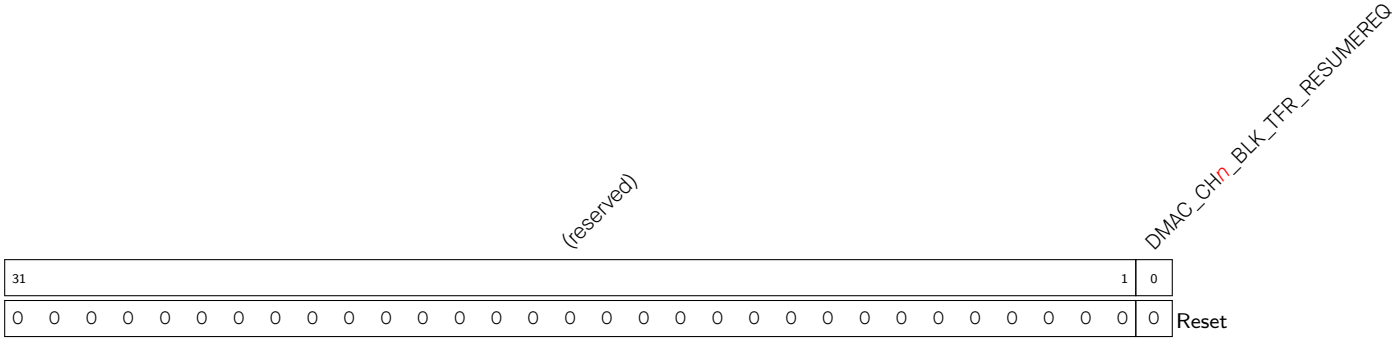
0：AXI 主机 1

1：AXI 主机 2

(R/W)

DMAC_CH*n*_LOC0 配置第一个链表项在存储器中的起始地址。起始地址的六个最低有效位 (LSB) 没有存储，因为假定地址是 64 字节边界对齐的。(R/W)

Register 7.16. DMAC_CH*n*_BLK_TFR_RESUMEREQ0_REG (*n*: 1-4) (0x0100*n + 0x0148)



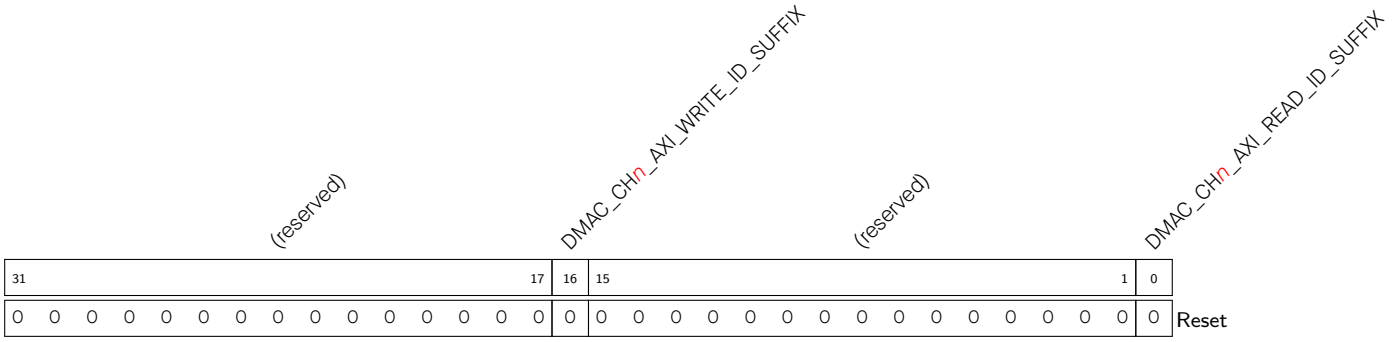
DMAC_CH*n*_BLK_TFR_RESUMEREQ 配置是否在基于链表或影子寄存器的多块传输时请求恢复块传输。

0：不请求恢复块传输

1：请求恢复块传输

(WO)

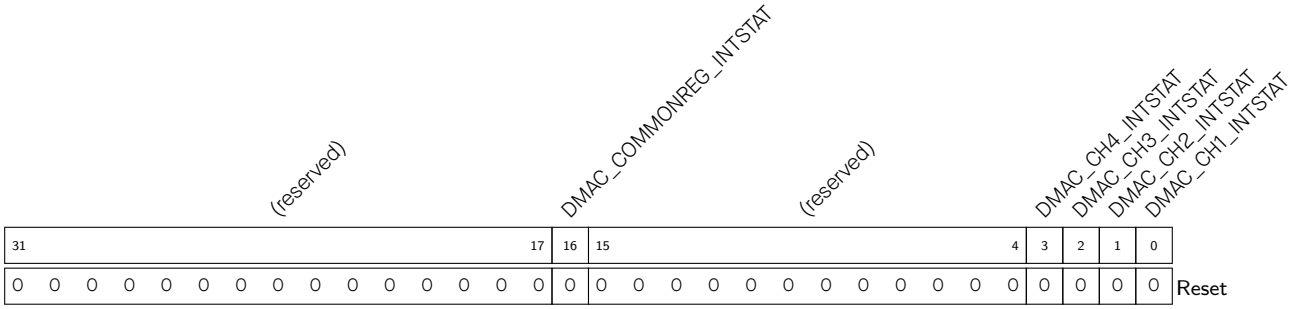
Register 7.17. DMAC_CH*n*_AXI_IDO_REG (0x0150)



DMAC_CH*n*_AXI_READ_ID_SUFFIX 配置 AXI 读请求的 ID 的最低位。(R/W)

DMAC_CH*n*_AXI_WRITE_ID_SUFFIX 配置 AXI 写请求的 ID 的最低位。(R/W)

Register 7.18. DMAC_INTSTATUSO_REG (0x0030)



DMAC_CH*n*_INTSTAT (*n*: 1-4) 表示通道 *n* 的中断激活状态。

1：通道 *n* 中断已激活

0：通道 *n* 中断未激活

(RO)

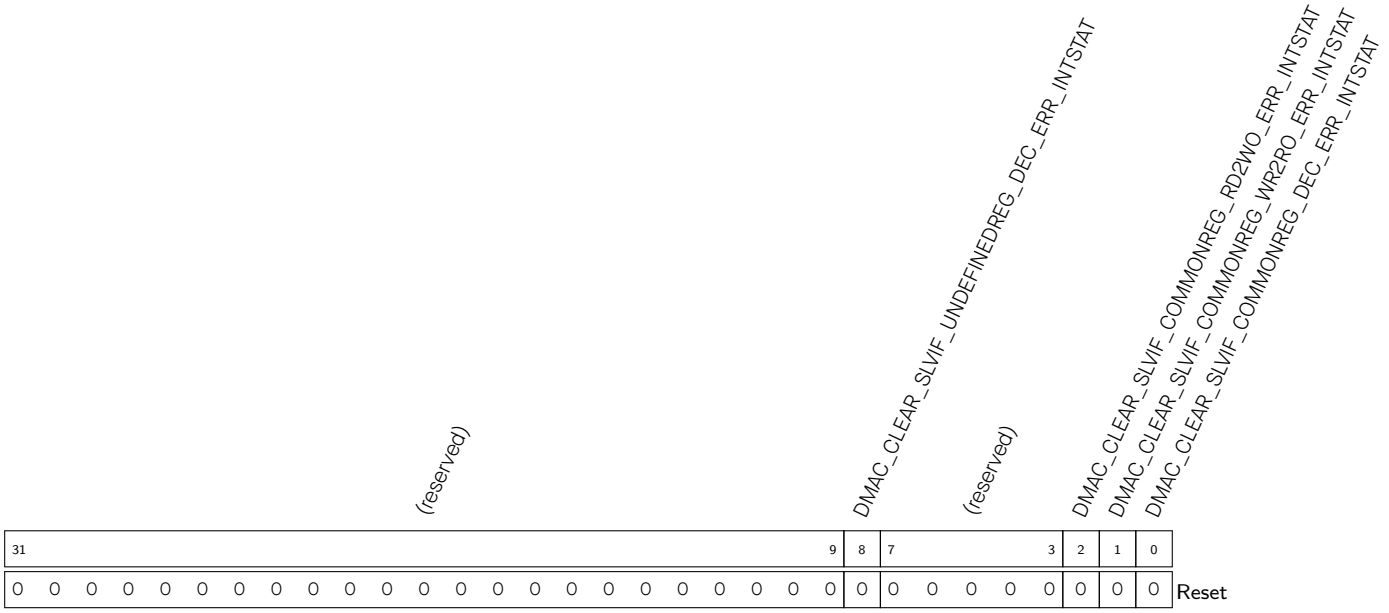
DMAC_COMMONREG_INTSTAT 表示一般寄存器中断的激活状态。

1：一般寄存器中断已激活

0：一般寄存器中断未激活

(RO)

Register 7.19. DMAC_COMMONREG_INTCLEAR0_REG (0x0038)



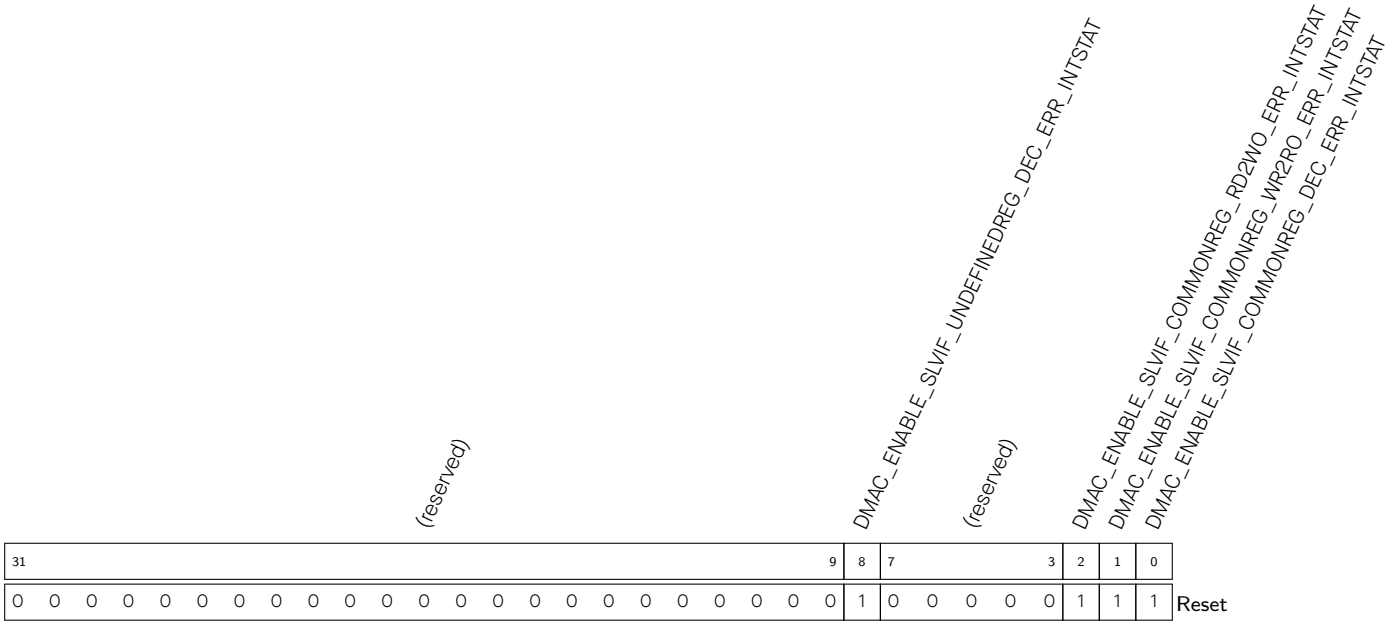
DMAC_CLEAR_SLVIF_COMMONREG_DEC_ERR_INTSTAT 写 1 清除
DMAC_SLVIF_COMMONREG_DEC_ERR_INTSTAT。(WO)

DMAC_CLEAR_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT 写 1 清除
DMAC_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT。(WO)

DMAC_CLEAR_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT 写 1 清除
DMAC_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT。(WO)

DMAC_CLEAR_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT 写 1 清除
DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT。(WO)

Register 7.20. DMAC_COMMONREG_INTSTATUS_ENABLE0_REG (0x0040)



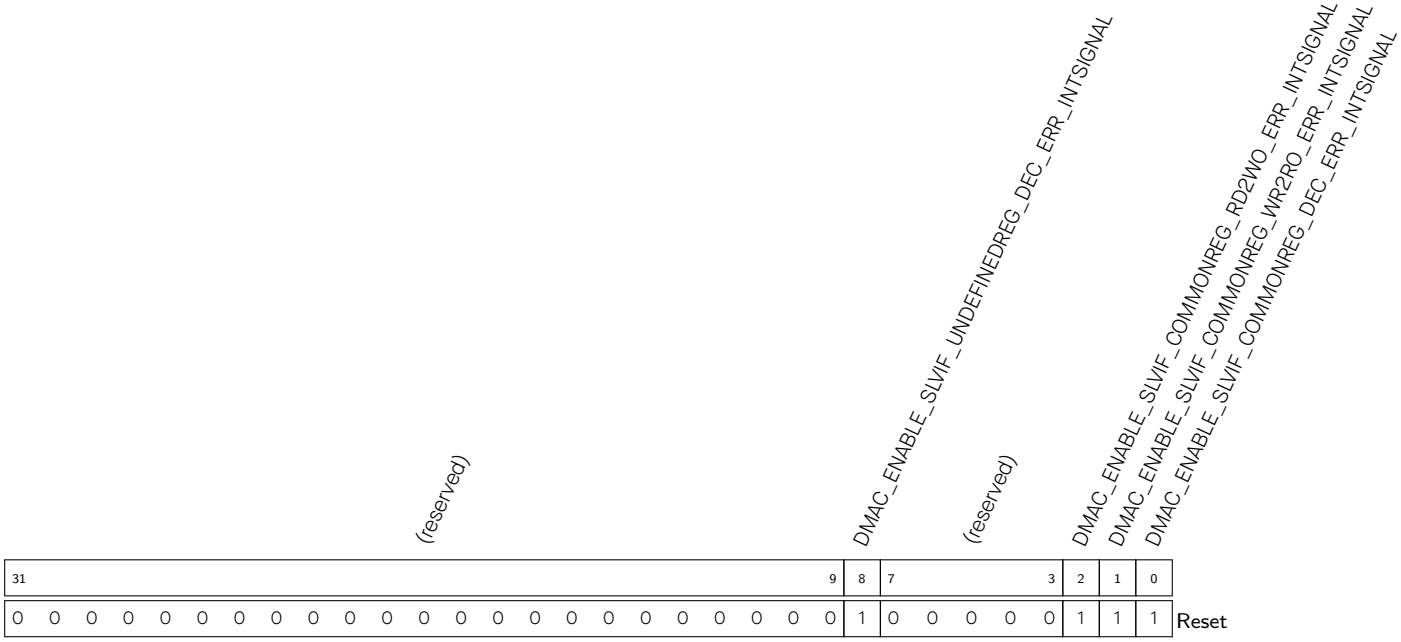
DMAC_ENABLE_SLVIF_COMMONREG_DEC_ERR_INTSTAT 写 1 使 能
DMAC_SLVIF_COMMONREG_DEC_ERR_INTSTAT。(R/W)

DMAC_ENABLE_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT 写 1 使 能
DMAC_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT。(R/W)

DMAC_ENABLE_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT 写 1 使 能
DMAC_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT。(R/W)

DMAC_ENABLE_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT 写 1 使 能
DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT。(R/W)

Register 7.21. DMAC_COMMONREG_INTSIGNAL_ENABLEO_REG (0x0048)



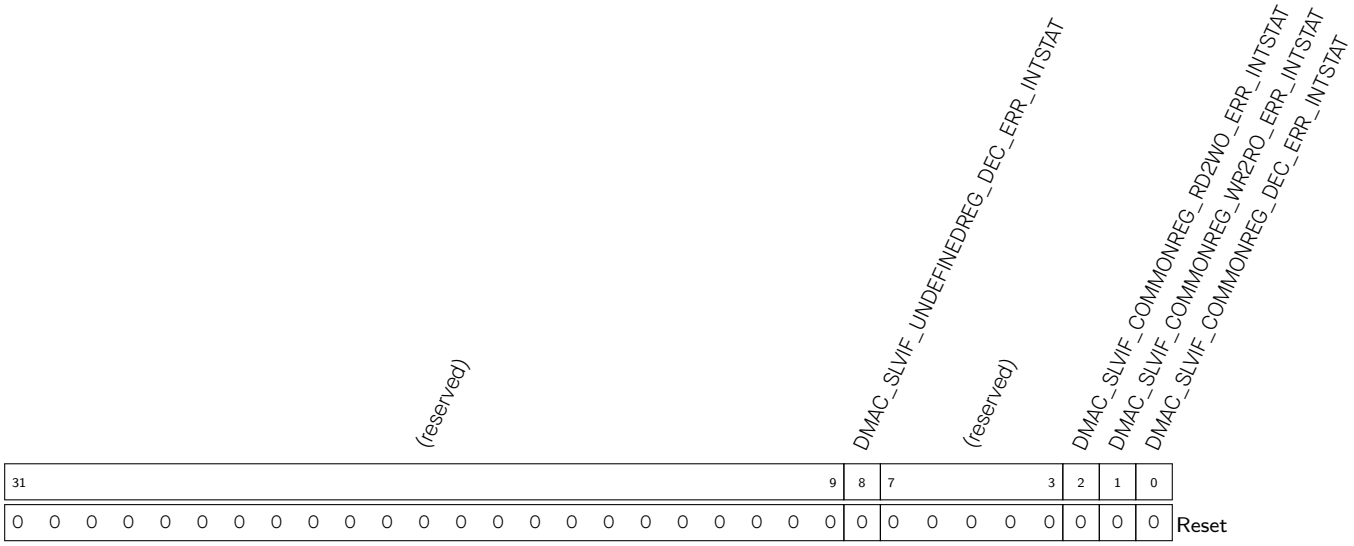
DMAC_ENABLE_SLVIF_COMMONREG_DEC_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_SLVIF_COMMONREG_DEC_ERR_INT。(R/W)

DMAC_ENABLE_SLVIF_COMMONREG_WR2RO_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_SLVIF_COMMONREG_WR2RO_ERR_INT。(R/W)

DMAC_ENABLE_SLVIF_COMMONREG_RD2WO_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_SLVIF_COMMONREG_RD2WO_ERR_INT。(R/W)

DMAC_ENABLE_SLVIF_UNDEFINEDREG_DEC_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INT。(R/W)

Register 7.22. DMAC_COMMONREG_INTSTATUS0_REG (0x0050)



DMAC_SLVIF_COMMONREG_DEC_ERR_INTSTAT 表示从机接口一般寄存器解码错误中断状态指示位。这个错误发生在 VDMA 访问一般寄存器空间 (0x000 到 0xFFF) 的无效地址时。

0：没有错误发生

1：一般寄存器解码错误发生

(RO)

DMAC_SLVIF_COMMONREG_WR2RO_ERR_INTSTAT 表示从机接口一般寄存器写入只读错误状态指示位。这个错误发生在一般寄存器空间 (0x000 到 0xFFF) 内写入一个只读寄存器的时候。

0：没有错误发生

1：一般寄存器写入只读错误发生

(RO)

DMAC_SLVIF_COMMONREG_RD2WO_ERR_INTSTAT 表示从机接口一般寄存器读到只写错误状态指示位。这个错误发生在一般寄存器空间 (0x000 到 0xFFF) 内读到一个只写寄存器的时候。

0：没有错误发生

1：一般寄存器读到只写错误发生

(RO)

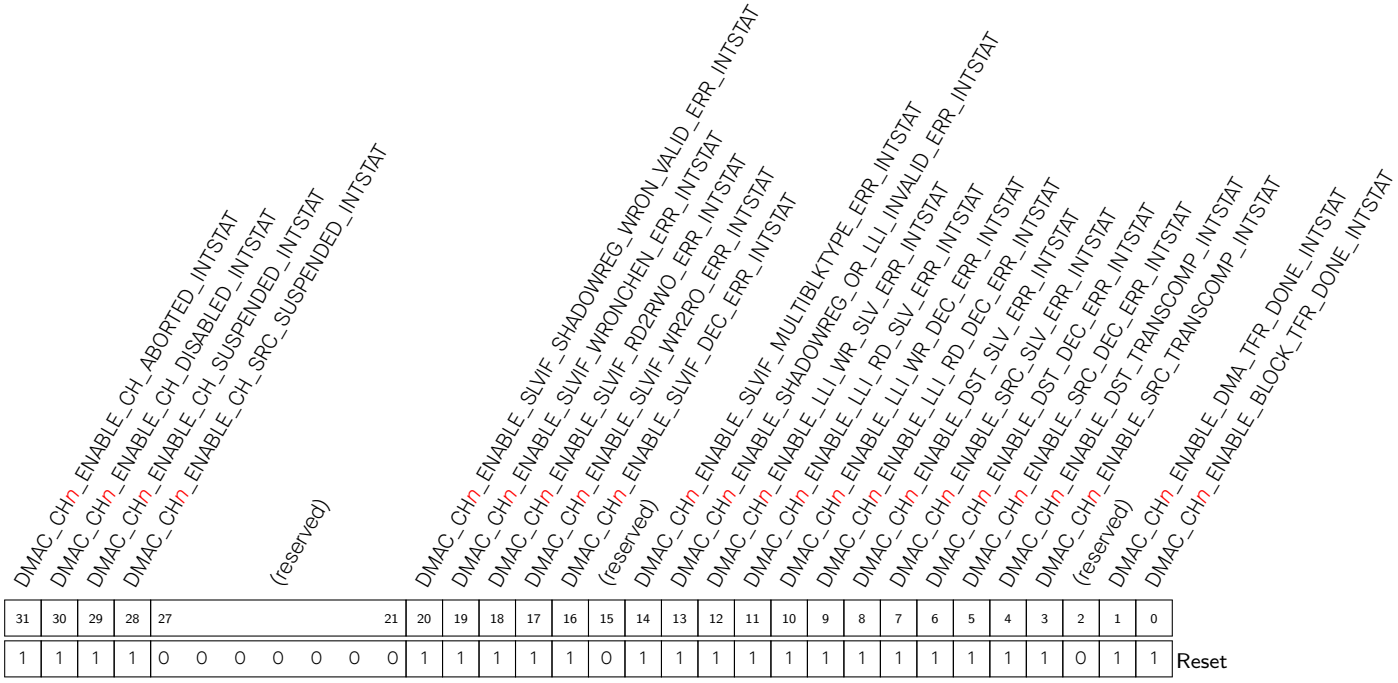
DMAC_SLVIF_UNDEFINEDREG_DEC_ERR_INTSTAT 表示从机接口未定义寄存器解码错误状态指示位。这个错误发生在访问未定义地址空间 (>0x4FF) 的时候。

0：没有错误发生

1：从机接口未定义寄存器解码错误发生

(RO)

Register 7.23. DMAC_CH*n*_INTSTATUS_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0180)



DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSTAT 写 1 使能 DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_DMA_TFR_DONE_INTSTAT 写 1 使能 DMAC_CH*n*_DMA_TFR_DONE_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SRC_TRANSCOMP_INTSTAT 写 1 使能 DMAC_CH*n*_SRC_TRANSCOMP_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_DST_TRANSCOMP_INTSTAT 写 1 使能 DMAC_CH*n*_DST_TRANSCOMP_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SRC_DEC_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SRC_DEC_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_DST_DEC_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_DST_DEC_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SRC_SLV_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SRC_SLV_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_DST_SLV_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_DST_SLV_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_LLI_RD_DEC_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_LLI_RD_DEC_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_LLI_WR_DEC_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_LLI_WR_DEC_ERR_INTSTAT。(R/W)

见下页...

Register 7.23. DMAC_CH*n*_INTSTATUS_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0180)

接上页...

DMAC_CH*n*_ENABLE_LLI_RD_SLV_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_LLI_RD_SLV_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_LLI_WR_SLV_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_LLI_WR_SLV_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_MULTIBLKTYPE_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_DEC_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SLVIF_DEC_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_WR2RO_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SLVIF_WR2RO_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_RD2RWO_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SLVIF_RD2RWO_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_WRONCHEN_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT 写 1 使能 DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT。(R/W)

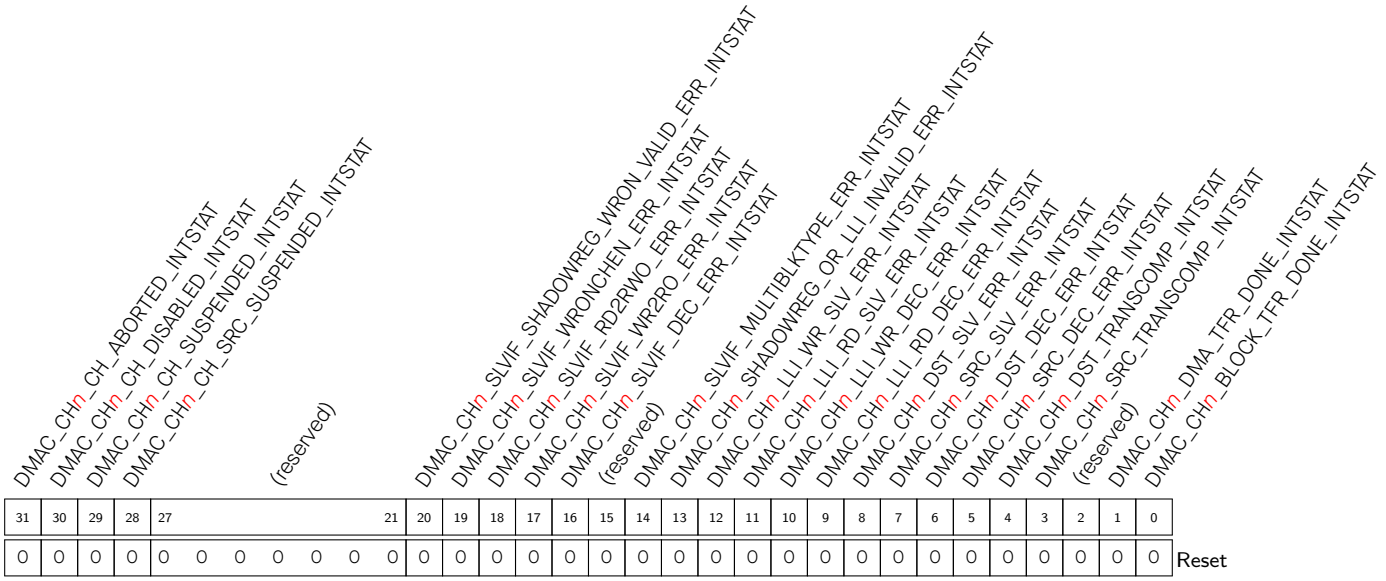
DMAC_CH*n*_ENABLE_CH_SRC_SUSPENDED_INTSTAT 写 1 使能 DMAC_CH*n*_CH_SRC_SUSPENDED_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_CH_SUSPENDED_INTSTAT 写 1 使能 DMAC_CH*n*_CH_SUSPENDED_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_CH_DISABLED_INTSTAT 写 1 使能 DMAC_CH*n*_CH_DISABLED_INTSTAT。(R/W)

DMAC_CH*n*_ENABLE_CH_ABORTED_INTSTAT 写 1 使能 DMAC_CH*n*_CH_ABORTED_INTSTAT。(R/W)

Register 7.24. DMAC_CH*n*_INTSTATUSO_REG (*n*: 1-4) (0x0100*n + 0x0188)



DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT 表示块传输完成状态指示位。(RO)

DMAC_CH*n*_DMA_TFR_DONE_INTSTAT 表示 DMA 传输完成状态指示位。(RO)

DMAC_CH*n*_SRC_TRANSCOMP_INTSTAT 表示源 DMA 传输完成状态指示位。(RO)

DMAC_CH*n*_DST_TRANSCOMP_INTSTAT 表示目标 DMA 传输完成状态指示位。(RO)

DMAC_CH*n*_SRC_DEC_ERR_INTSTAT 表示源解码错误状态指示位。在源数据传输过程中，由主机接口检测到解码错误。(RO)

DMAC_CH*n*_DST_DEC_ERR_INTSTAT 表示目标解码错误状态指示位。在目标数据传输过程中，由主机接口检测到解码错误。(RO)

DMAC_CH*n*_SRC_SLV_ERR_INTSTAT 表示源从机错误状态指示位。在源数据传输过程中，由主机接口检测到从机错误。如果读取数据的从机接口发出从机错误，则会发生此错误。(RO)

DMAC_CH*n*_DST_SLV_ERR_INTSTAT 表示目标从机错误状态指示位。在目标数据传输过程中，由主机接口检测到从机错误。如果写入数据的从机接口发出从机错误，则会发生此错误。(RO)

DMAC_CH*n*_LLI_RD_DEC_ERR_INTSTAT 表示 LLI 读解码错误状态指示位。在 LLI 读操作过程中，由主机接口检测到解码错误。(RO)

DMAC_CH*n*_LLI_WR_DEC_ERR_INTSTAT 表示 LLI 写解码错误状态指示位。在 LLI 回写操作过程中，由主机接口检测到解码错误。(RO)

DMAC_CH*n*_LLI_RD_SLV_ERR_INTSTAT 表示 LLI 读从机错误状态指示位。在 LLI 读操作过程中，由主机接口检测到从机错误。(RO)

DMAC_CH*n*_LLI_WR_SLV_ERR_INTSTAT 表示 LLI 写从机错误状态指示位。在 LLI 回写操作过程中，由主机接口检测到从机错误。(RO)

DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT 表示影子寄存器或 LLI 无效错误状态指示位。在 VDMA 影子寄存器或 LLI 获取阶段，如果 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 为 0，则发生此错误。(RO)

见下页...

Register 7.24. DMAC_CH*n*_INTSTATUSO_REG (*n*: 1-4) (0x0100*n* + 0x0188)

接上页...

DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INTSTAT 表示从机接口多块类型错误状态指示位。如果 DMAC_CH*n*_SRC_MULTBLK_TYPE 和 DMAC_CH*n*_DST_MULTBLK_TYPE 里面配置的多块传输类型无效，则会产生此错误。(RO)

DMAC_CH*n*_SLVIF_DEC_ERR_INTSTAT 表示从机接口解码错误状态指示位。在访问寄存器过程中，由 VDMA 生成解码错误。如果访问通道 *n* 的寄存器空间中的无效地址，则会产生此错误。(RO)

DMAC_CH*n*_SLVIF_WR2RO_ERR_INTSTAT 表示从机接口写入只读错误状态指示位。这个错误发生在写入只读寄存器的时候。(RO)

DMAC_CH*n*_SLVIF_RD2RWO_ERR_INTSTAT 表示从机接口读到只写错误状态指示位。这个错误发生在读到只写寄存器的时候。(RO)

DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INTSTAT 表示从机接口写入启用通道错误状态指示位。如果在通道启用时对寄存器执行非法写操作，并且此写操作违反 VDMA 规范的时候，则会发生此错误。(RO)

DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT 表示影子寄存器写有效错误状态指示位。如果启用了基于影子寄存器的多块传输，并且当 DMAC_CH*n*_SHADOWREG_OR_LLI_VALID 为 1 时，软件尝试写入影子寄存器，则会发生此错误。(RO)

DMAC_CH*n*_CH_SRC_SUSPENDED_INTSTAT 表示通道源挂起状态指示位。

0：通道源未被挂起

1：通道源被挂起

(RO)

DMAC_CH*n*_CH_SUSPENDED_INTSTAT 表示通道挂起状态指示位。

0：通道未被挂起

1：通道被挂起

(RO)

DMAC_CH*n*_CH_DISABLED_INTSTAT 表示通道禁用状态指示位。

0：通道被禁用

1：通道未被禁用

(RO)

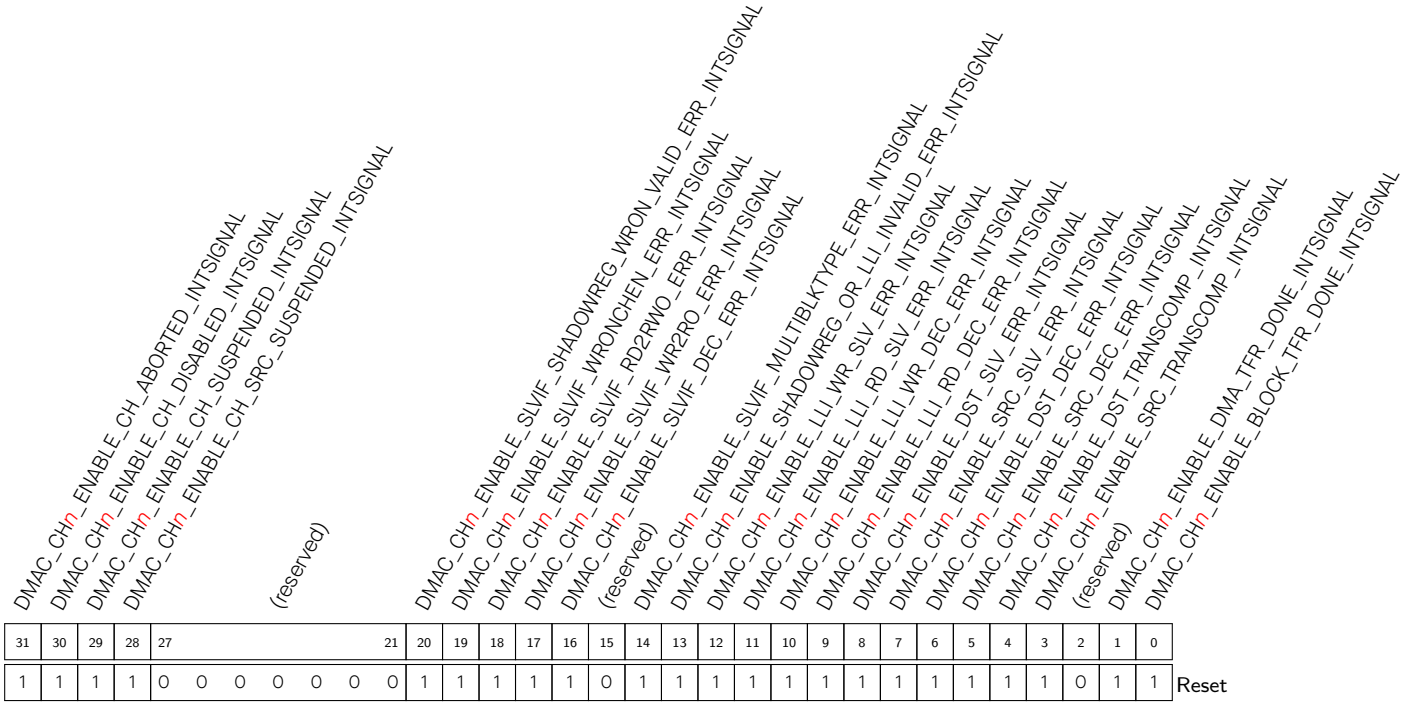
DMAC_CH*n*_CH_ABORTED_INTSTAT 表示通道中止状态指示位。

0：通道被中止

1：通道未被中止

(RO)

Register 7.25. DMAC_CH*n*_INTSIGNAL_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0190)



DMAC_CH*n*_ENABLE_BLOCK_TFR_DONE_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_BLOCK_TFR_DONE_INT。(R/W)

DMAC_CH*n*_ENABLE_DMA_TFR_DONE_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_DMA_TFR_DONE_INT。(R/W)

DMAC_CH*n*_ENABLE_SRC_TRANSCOMP_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SRC_TRANSCOMP_INT。(R/W)

DMAC_CH*n*_ENABLE_DST_TRANSCOMP_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_DST_TRANSCOMP_INT。(R/W)

DMAC_CH*n*_ENABLE_SRC_DEC_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SRC_DEC_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_DST_DEC_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_DST_DEC_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SRC_SLV_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SRC_SLV_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_DST_SLV_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_DST_SLV_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_LLI_RD_DEC_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_LLI_RD_DEC_ERR_INT。(R/W)

见下页...

Register 7.25. DMAC_CH*n*_INTSIGNAL_ENABLEO_REG (*n*: 1-4) (0x0100*n* + 0x0190)

接上页...

DMAC_CH*n*_ENABLE_LLI_WR_DEC_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_LLI_WR_DEC_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_LLI_RD_SLV_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_LLI_RD_SLV_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_LLI_WR_SLV_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_LLI_WR_SLV_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SHADOWREG_OR_LLI_INVALID_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_MULTIBLKTYPE_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_DEC_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SLVIF_DEC_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_WR2RO_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SLVIF_WR2RO_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_RD2RWO_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SLVIF_RD2RWO_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_WRONCHEN_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INT。(R/W)

DMAC_CH*n*_ENABLE_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INT。(R/W)

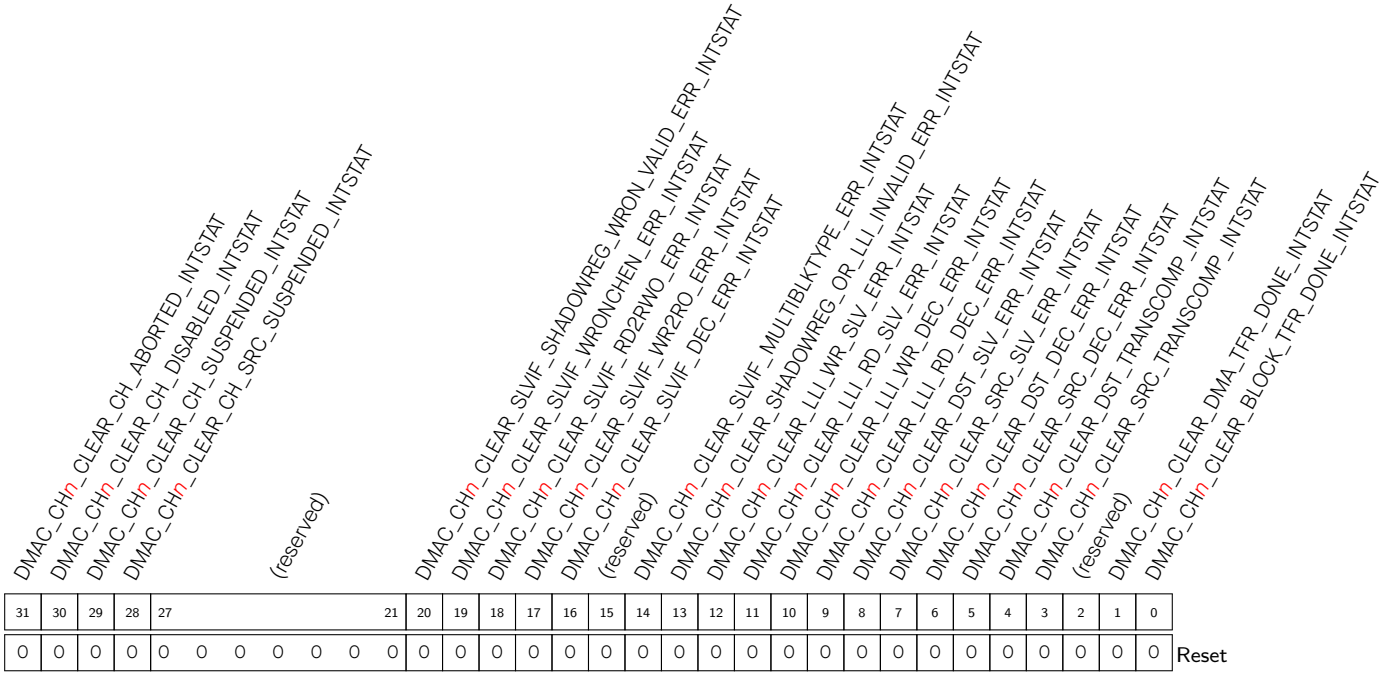
DMAC_CH*n*_ENABLE_CH_SRC_SUSPENDED_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_CH_SRC_SUSPENDED_INT。(R/W)

DMAC_CH*n*_ENABLE_CH_SUSPENDED_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_CH_SUSPENDED_INT。(R/W)

DMAC_CH*n*_ENABLE_CH_DISABLED_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_CH_DISABLED_INT。(R/W)

DMAC_CH*n*_ENABLE_CH_ABORTED_INTSIGNAL 写 1 在端口层使能 DMAC_CH*n*_CH_ABORTED_INT。(R/W)

Register 7.26. DMAC_CH*n*_INTCLEAR0_REG (*n*: 1-4) (0x0100**n* + 0x0198)



DMAC_CH*n*_CLEAR_BLOCK_TFR_DONE_INTSTAT 写 1 清除 DMAC_CH*n*_BLOCK_TFR_DONE_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_DMA_TFR_DONE_INTSTAT 写 1 清除 DMAC_CH*n*_DMA_TFR_DONE_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SRC_TRANSCOMP_INTSTAT 写 1 清除 DMAC_CH*n*_SRC_TRANSCOMP_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_DST_TRANSCOMP_INTSTAT 写 1 清除 DMAC_CH*n*_DST_TRANSCOMP_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SRC_DEC_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SRC_DEC_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_DST_DEC_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_DST_DEC_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SRC_SLV_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SRC_SLV_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_DST_SLV_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_DST_SLV_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_LLI_RD_DEC_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_LLI_RD_DEC_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_LLI_WR_DEC_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_LLI_WR_DEC_ERR_INTSTAT。(WO)

见下页...

Register 7.26. DMAC_CH*n*_INTCLEAR0_REG (*n*: 1-4) (0x0100*n* + 0x0198)

接上页...

DMAC_CH*n*_CLEAR_LLI_RD_SLV_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_LLI_RD_SLV_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_LLI_WR_SLV_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_LLI_WR_SLV_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SHADOWREG_OR_LLI_INVALID_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SLVIF_MULTIBLKTYPE_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SLVIF_MULTIBLKTYPE_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SLVIF_DEC_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SLVIF_DEC_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SLVIF_WR2RO_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SLVIF_WR2RO_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SLVIF_RD2RWO_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SLVIF_RD2RWO_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SLVIF_WRONCHEN_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SLVIF_WRONCHEN_ERR_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT 写 1 清除 DMAC_CH*n*_SLVIF_SHADOWREG_WRON_VALID_ERR_INTSTAT。(WO)

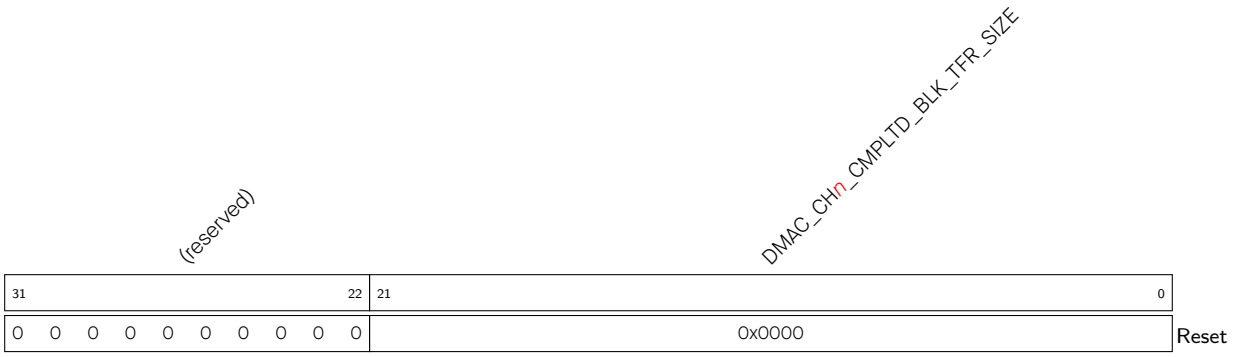
DMAC_CH*n*_CLEAR_CH_SRC_SUSPENDED_INTSTAT 写 1 清除 DMAC_CH*n*_CH_SRC_SUSPENDED_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_CH_SUSPENDED_INTSTAT 写 1 清除 DMAC_CH*n*_CH_SUSPENDED_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_CH_DISABLED_INTSTAT 写 1 清除 DMAC_CH*n*_CH_DISABLED_INTSTAT。(WO)

DMAC_CH*n*_CLEAR_CH_ABORTED_INTSTAT 写 1 清除 DMAC_CH*n*_CH_ABORTED_INTSTAT。(WO)

Register 7.27. DMAC_CH*n*_STATUS0_REG (*n*: 1-4) (0x0100**n* + 0x0130)



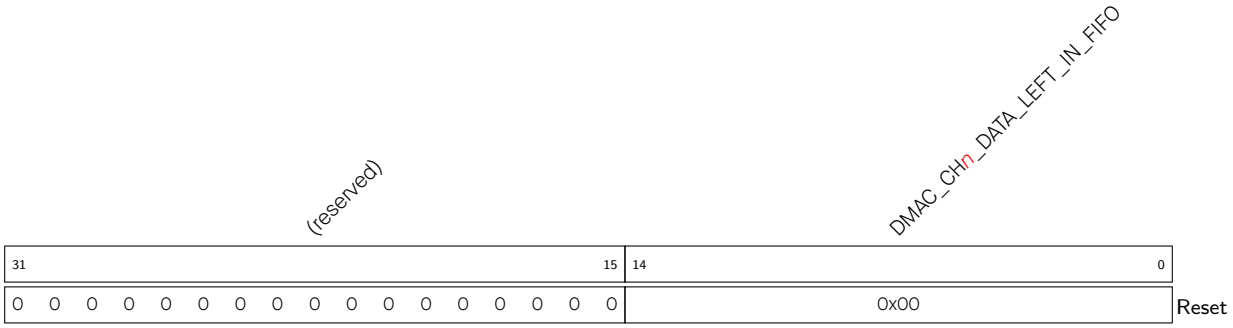
DMAC_CH*n*_CMPLTD_BLK_TFR_SIZE 表示在上一个块传输中以 DMAC_CH*n*_SRC_TR_WIDTH 宽度传输的数据总数。

如果在 DMA 传输过程中发生任何错误，块传输可能会被提前终止，在这种情况下，这个值表示当前块中无错误传输的实际数据。

在启用通道时，该字段将被清零。

(RO)

Register 7.28. DMAC_CH*n*_STATUS1_REG (*n*: 1-4) (0x0100**n* + 0x0134)



DMAC_CH*n*_DATA_LEFT_IN_FIFO 表示在完成当前块传输后 VDMA 通道 FIFO 中剩余的数据总数。通道 FIFO 中的数据宽度等于 DMAC_CH*n*_SRC_TR_WIDTH。

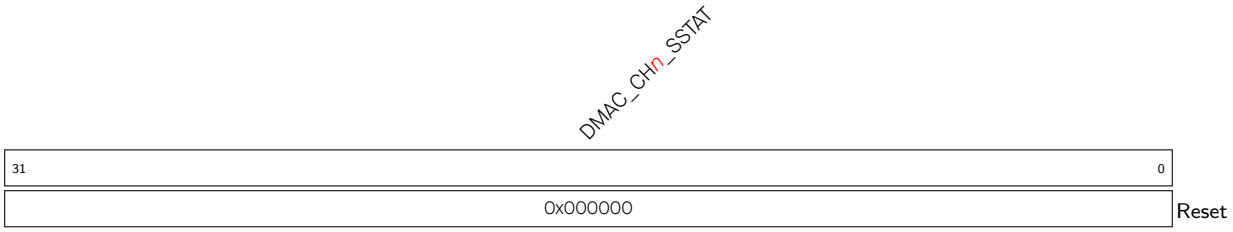
对于正常的块传输完成且没有错误，该字段的值为 0。如果在 DMA 传输过程中发生任何错误，块传输可能会被提前终止，在这种情况下，该字段表示通道 FIFO 中剩余的数据，这些数据无法传输到目标外设。

在启用通道时，该字段将被清零。

如果 DMAC_CH*n*_DST_TR_WIDTH > DMAC_CH*n*_SRC_TR_WIDTH，则 FIFO 中可能会有剩余数据，这些数据不足以形成一次 DMAC_CH*n*_SRC_TR_WIDTH 宽度的单次传输，这种情况下，该字段将返回 0。

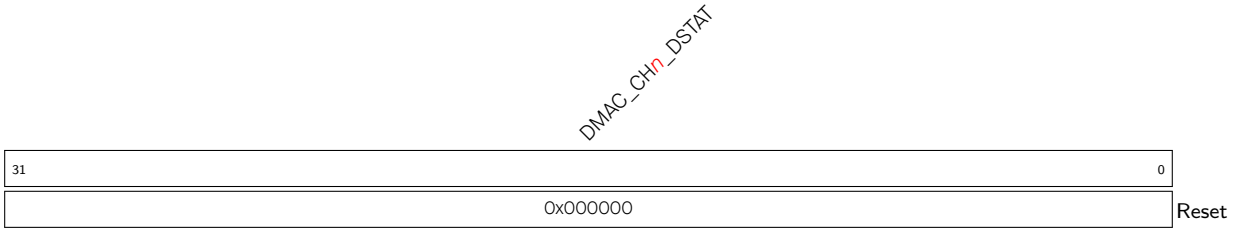
(RO)

Register 7.29. DMAC_CH*n*_SSTAT0_REG (*n*: 1-4) (0x0100*n* + 0x0160)



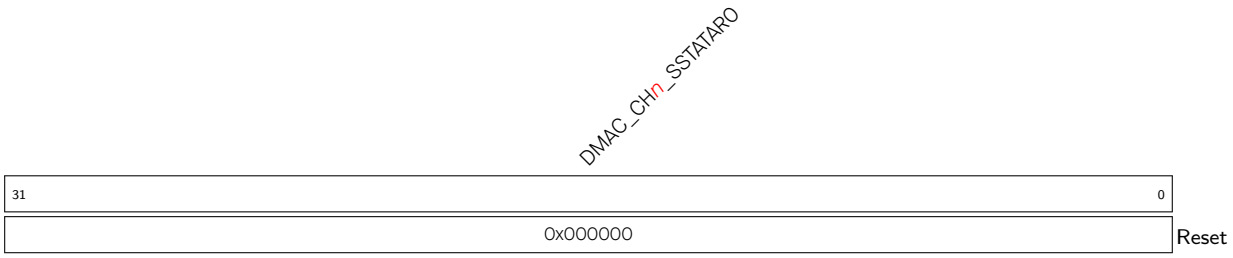
DMAC_CH*n*_SSTAT 表示硬件从 DMAC_CH*n*_SSTATARO_REG 寄存器内容指向的地址处获取的源状态信息。(RO)

Register 7.30. DMAC_CH*n*_DSTAT0_REG (*n*: 1-4) (0x0100*n* + 0x0168)



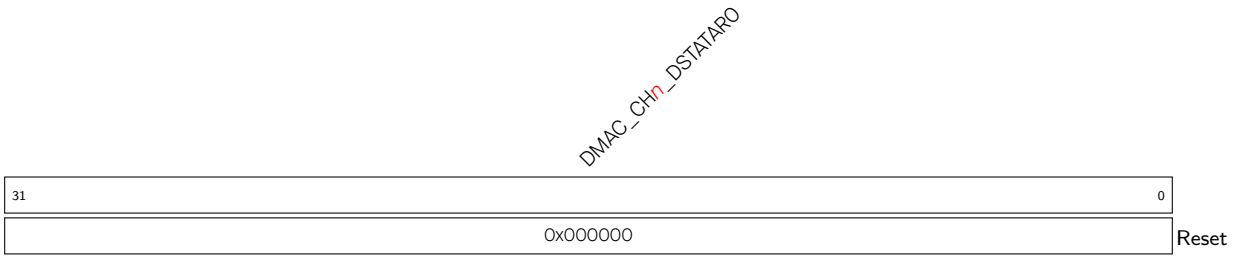
DMAC_CH*n*_DSTAT 表示硬件从 DMAC_CH*n*_DSTATARO_REG 寄存器内容指向的地址处获取的目标状态信息。(RO)

Register 7.31. DMAC_CH*n*_SSTATARO_REG (*n*: 1-4) (0x0100*n* + 0x0170)



DMAC_CH*n*_SSTATARO 配置硬件获取源状态信息的地址。(R/W)

Register 7.32. DMAC_CH*n*_DSTATARO_REG (*n*: 1-4) (0x0100**n* + 0x0178)



DMAC_CH*n*_DSTATARO 配置硬件获取目标状态信息的地址。 (R/W)

第 8 章

2D-DMA 控制器 (2D-DMA)

8.1 概述

2D-DMA 控制器是专用于二维图像处理的 DMA，在支持 GDMA-AXI 的全部功能基础上，增加了宏块重排 (Reorder) 和颜色空间转换（Color Space Convert: CSC）功能，能够更好地支持 JPEG 和 PPA 外设的数据传输需求。2D-DMA 支持存储器到存储器的传输，可以将宏块从存储器的一段地址空间搬运到另一段地址空间，并完成颜色空间转换。

8.2 特性

- 1 个 AXI 主机接口
- 支持首地址非对齐的数据传输
- 支持存储到存储、外设到存储 (RX)、以及存储到外设 (TX) 的数据传输
- 包含 4 个存储到外设通道，3 个外设到存储通道
- 支持 PPA 和 JPEG 图像编解码器外设
- 支持宏块重排序功能
- 支持颜色空间转换功能
- 支持通道优先级、权重配置

8.3 架构

ESP32-P4 中只有 JPEG 和 PPA 两个外设支持 2D-DMA 功能。2D-DMA 控制器与 CPU 的数据总线使用相同的地址空间访问存储器。图 8.3-1 为 2D-DMA 控制器基本架构图。

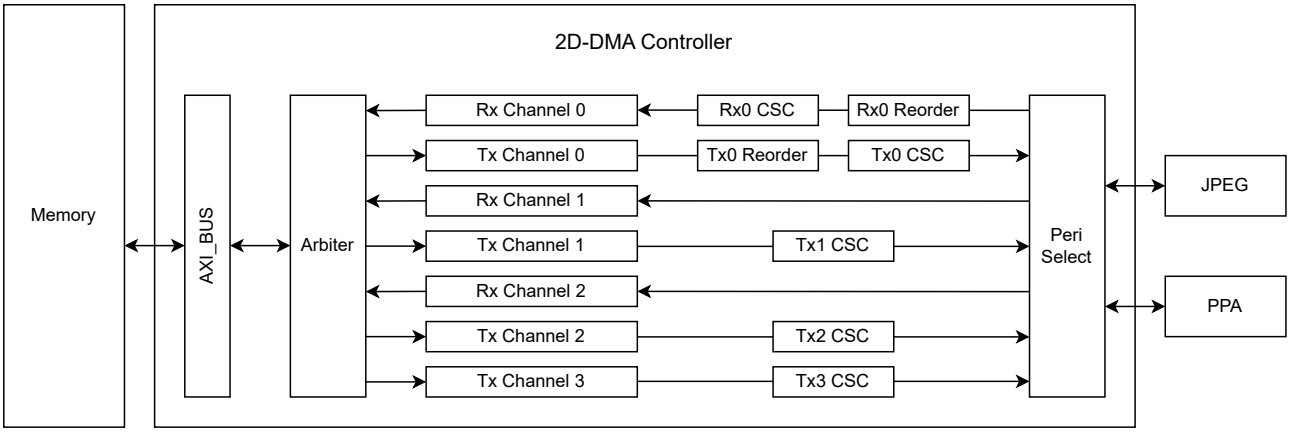


图 8.3-1. 2D-DMA 基本架构图

2D-DMA 控制器有 7 个独立的通道，其中包括 4 个发送通道和 3 个接收通道。每个通道可选择与不同的外设相连，从而实现通道资源由外设共享。

2D-DMA 控制器支持宏块重排 (Reorder) 和颜色空间转换 (CSC) 功能，仅发送通道 0 和接收通道 0 支持宏块重排，仅发送通道 0、1、2、3 和接收通道 0 支持颜色空间转换功能。

2D-DMA 通过 AXI_BUS 将数据写入存储器，或者从存储器读取数据，其中存储器包含片内存储器和片外存储器。在通过总线传输数据之前，2D-DMA 采用可配置的仲裁机制对每个通道的读写请求进行仲裁。内外部存储器的具体使用范围详见章节 9 *系统和存储器*。

软件可以通过挂载链表的方式来使用 2D-DMA 控制器。链表可以存储在片内或片外存储器中。这些链表包括 outlink*n* 与 inlink*n*，本文以 *n* 来表示通道号，outlink*n* 中，n 的范围是 0 ~ 3，但是 inlink*n* 中，*n* 的范围是 0 ~ 2。2D-DMA 从存储器中取得链表，然后根据 outlink*n* 中的内容将相应存储器中的数据发送出去，也可根据 inlink*n* 中的内容将接收的数据存入指定存储器地址空间。

8.4 功能描述

8.4.1 传输模式

2D-DMA 共支持 4 种传输模式：

- 1D 模式：将链表描述符 DW0 的 2DEN 置 0，DW2 的 mod 置 0 进入该模式，该模式下 2D-DMA 将根据链表配置的数据长度、存储器空间大小进行存储器数据的读写操作，操作地址是连续的，与 GDMA-AXI 功能相同。
- 2D-MOD0 模式：将链表描述符 DW0 的 2DEN 置 1，DW2 的 mod 置 0 进入该模式，该模式下，2D-DMA 从 HA*VA 图像的 (X, Y) 坐标开始，读/写一个 hb*vb 宏块，地址在每行最后一个读/写数据处理完成后都会跳变到下一行需要被读/写的第一个数据地址。
- 2D-MOD1 模式：将链表描述符 DW0 的 2DEN 置 1，DW2 的 mod 置 1 进入该模式，该模式下，2D-DMA 从 HA*VA 图像的首地址开始，连续读/写多个单位大小的宏块，直到 HA*VA 图像读/写完成。对于 JPEG 外设，在该模式下需要配置寄存器 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* 或 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n*，选择读取宏块的单位大小，有三种尺寸可选（水平 × 垂直）：8 像素 × 8 像素、8 像素 × 16 像素、16 像素 × 16 像素，如何选择可参照章节 37 *JPEG 图像编解码器*。宏块单位大小与链表描述符 hb、vb 的选择可参照表 8.4-4 和表 8.4-5。

- DSCR-PORT 模式：将链表描述符 DW0 的 2DEN 置 1，DW2 的 mod 置 0，并配置寄存器 DMA2D_OUT_DSCR_PORT_EN_CH*n* 或 DMA2D_IN_DSCR_PORT_EN_CH*n* 为 1 进入该模式，该模式仅为 PPA 应用，具体操作可参照章节 39 像素处理加速器 *(PPA)*。

1D 模式、2D-MODO 模式、2D-MOD1 模式和 DSCR-PORT 模式均支持外设到存储及存储到外设的数据传输。仅 1D 模式和 2D-MODO 模式支持存储到存储数据传输。

8.4.2 链表

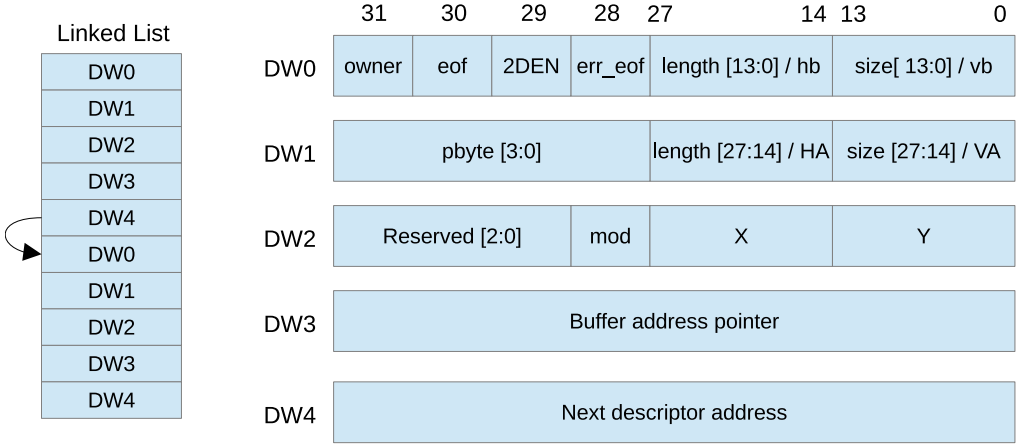


图 8.4-1. 链表结构图

图 8.4-1 所示为链表的结构图。发送链表与接收链表结构相同。每个链表由一个或者若干个描述符构成，一个描述符由 5 个字组成。链表应存放在存储器中，供 2D-DMA 控制器使用。描述符每一字段的意义如下：

注意：本文以 DMA2D_OUT 开头的寄存器对应 TX 通道寄存器，以 DMA2D_IN 开头的寄存器对应 RX 通道寄存器。

表 8.4-1. 描述符字段说明

位	名称	说明
DW0[31]	owner	表示当前描述符对应的 buffer 允许的操作者。 • 0：允许的操作者为 CPU • 1: 1：允许的操作者为 2D-DMA 控制器 在 2D-DMA 使用完该描述符对应的 buffer 后，对于接收描述符，硬件默认会自动将该位清零；对于发送描述符，需要将 DMA2D_OUT_AUTO_WRBACK_CH*n* 置 1，硬件才会自动将该位清零。软件在挂载链表时需要将该位置 1。
DW0[30]	eof	表示结束标志。 • 0：当前描述符不是一张图片的最后一个描述符 • 1：当前描述符为一张图片的最后一个描述符 对于接收描述符，需要软件将该位写 0，硬件会在收完一张图片后将该位置 1。对于发送描述符，需要软件在一张图片最后一个描述符中的该位置 1。

位	名称	说明
DW0[29]	2DEN	2D 功能使能标志。 • 0：2D-DMA 启用 1D 工作模式 • 1：2D-DMA 启用 2D 工作模式
DW0[28]	err_eof	表示接收结束错误标志（仅 JPEG 支持）。 • 0：接收数据没有错误 • 1：接收数据有错误 对于接收描述符，硬件在收完一张图片并检测到接收数据错误会将该位置 1。
DW0[27:14]	length[13:0]/hb	启用 2D 工作模式时为 **hb** – 表示想要搬运的宏块的水平宽度，单位为像素。 启用 1D 工作模式时为 **length[13:0]** – 即 length 的低 14 位。length 表示当前描述符对应的 buffer 中的有效字节数。对于发送描述符，该段由软件填写，表示从 buffer 中读取数据时需要读取的字节数；对于接收描述符，该段由硬件使用完该 buffer 后或者接收到最后一个数据时自动填写，表示 buffer 中存储的有效字节数。
DW0[13:0]	size[13:0]/vb	启用 2D 工作模式时为 **vb** – 表示想要搬运的宏块的垂直高度，单位为像素。 启用 1D 工作模式时为 **size[13:0]** – 即 size 的低 14 位。size 表示当前描述符对应的 buffer 容量的字节数。
DW1[31:28]	pbyte	启用 2D 工作模式时，表示从存储器搬运 / 搬运到存储器的图像单个像素占用的字节数。（如应用了颜色空间转换功能，该字段表示的是存储器中存放的图像每个像素所占用的字节数。） • 0: 0.5 byte/pixel • 1: 1 byte/pixel • 2: 1.5 bytes/pixel • 3: 2 bytes/pixel • 4: 3 bytes/pixel • 5: 4 bytes/pixel 启用 1D 工作模式时，该字段没有使用，可为任意值。
DW1[27:14]	length[27:14]/HA	启用 2D 工作模式时为 **HA** – 表示想要搬运的宏块对应的原始图像的水平宽度，单位为像素。 启用 1D 工作模式时为 **length[27:14]** – 即 length 的高 14 位。
DW1[13:0]	size[27:14]/VA	启用 2D 工作模式时为 **VA** – 表示想要搬运的宏块对应的原始图像的垂直高度，单位为像素。 启用 1D 工作模式时为 **size[27:14]** – 即 size 的高 14 位。
DW2[31:29]	Reserved	保留。此位为无关项。

位	名称	说明
DW2[28]	mod	宏块读取模式。 • 0：2D-DMA 只读/写一次水平宽度为 hb、垂直高度为 vb 的宏块 • 1：2D-DMA 读/写多次水平宽度为 hb、垂直高度为 vb 的宏块，直到将水平宽度为 HA，垂直高度为 VA 的图像数据读/写完成 1D 模式下，该位必须为 0。 **说明：** hb/vb/HA/VA 的单位为像素。
DW2[27:14]	X	启用 2D 工作模式时，表示 hb*vb 宏块在 HA*VA 图像中的 X 坐标，单位为像素。 启用 1D 工作模式时，该域没有使用，可为任意值。
DW2[13:0]	Y	启用 2D 工作模式时，表示 hb*vb 宏块在 HA*VA 图像中的 Y 坐标，单位为像素。 启用 1D 工作模式时，该域没有使用，可为任意值。
DW3	Buffer address pointer	启用 2D 工作模式时，表示 HA*VA 宏块原点坐标在 buffer 中的地址（原始图像在存储器中第一个字节的地址）。 启用 1D 工作模式时，表示 buffer 的起始地址。
DW4	Next descriptor address	下一个描述符的地址。如果当前描述符为链表中最后一个描述符时，该值可以为 0。该地址可以指向片内或片外存储器的地址空间。 **说明：** 需要注意，2D-DMA 链表描述符的存放首地址要求 8 字节对齐。

8.4.3 2D-MOD1 模式和 DSCR-PORT 模式的补数方式

本小节仅应用于 2D-DMA 的发送方向且选择 2D-MOD1 或 DSCR-PORT 模式。

JPEG 解码方向输入的图像数据的基本单元为 8 × 8 或 8 × 16 或 16 × 16（像素）的块，在 2D-MOD1 模式下当图像的长宽不满足基本单元的整数倍时，需要根据上一个宏块或者当前宏块作补齐操作，完成整数个基本单元的传输。

PPA 在发送通道 DSCR-PORT 模式的补数基本单元为 36 × 36（YUV420）、36 × 34（YUV422）和 34 × 34 三种（水平尺寸 × 垂直尺寸），可通过配置寄存器 DMA2D_OUT_DSCR_PORT_BLK_H_CH*n* 和 DMA2D_OUT_DSCR_PORT_BLK_V_CH*n* 来配置不同发送通道基本单元的宽/高。当图像的长宽不满足基本单元的整数倍时，需要做数据补齐操作。

补数规则如下：

水平方向不满一行时，补齐的数据与描述符中 pbyte 字段相关：

- 当 pbyte 为 0 (0.5 byte/piexl)，要求剩余像素为偶数个，补齐数据为最后 0.5 byte

- 当 pbyte 为 1 (1 byte/piexl)，补齐数据为最后 1 byte
- 当 pbyte 为 2 (1.5 byte/piexl)，要求剩余像素为偶数个，补齐数据为最后 3 byte
- 当 pbyte 为 3 (2 byte/piexl)，补齐数据为最后 2 byte
- 当 pbyte 为 4 (3 byte/piexl)，补齐数据为最后 3 byte
- 当 pbyte 为 5 (4 byte/piexl)，补齐数据为最后 4 byte

垂直方向不满一个 block 时，以最后一行补齐剩下的行。

补齐方式如图 8.4-2 和图 8.4-3 所示。

图 8.4-2 中，一个蓝色方框表示一个基本单元 (8 × 8 或 8 × 16 或 16 × 16（像素）的块)，右侧和下侧一个黄色方框加上一个绿色方框表示一个基本单元，右下角一个黄色方框和两个绿色方框表示一个基本单元。

图 8.4-3 中一个方框表示一个像素，方框内的不同数字表示不同的像素值。

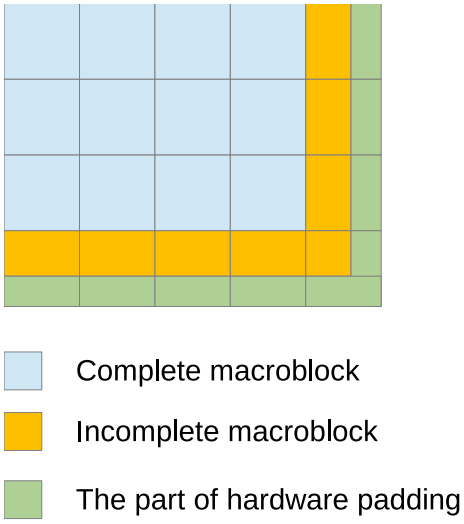


图 8.4-2. 补齐示意图（整体）

1	2	3	3	3	3	3	3
4	5	6	6	6	6	6	6
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9
7	8	9	9	9	9	9	9

图 8.4-3. 补齐示意图（细节）

8.4.4 外设到存储及存储到外设的数据传输

当 DMA2D_MEM_TRANS_EN_CH*n* 配置为 0 时，2D-DMA 进入外设模式。发送通道从存储器中读取数据后传输给外设，接收通道接收来自外设的数据后写入存储器。

2D-DMA 支持存储到外设及外设到存储的数据传输，分别对应 TX 及 RX 功能。TX 通道通过 outlink*n* 实现将指定存储区域中的数据搬运到外设的发送端；RX 通道通过 inlink*n* 实现将外设接收到的数据搬运到指定的存储区域。

每个 RX/TX 通道均可以被配置连接到任意一个支持 2D-DMA 功能的外设，表 8.4-2 和表 8.4-3 所示为通过寄存器配置通道与其对应外设的关系。其中"Dummy-*n*" 是 2D-DMA 在实现存储到存储数据搬运功能时可选配的寄存器值。当其中一个通道已经与某一个外设连接时，其他通道将不能配置为与该外设连接。

表 8.4-2. 2D-DMA TX 方向配置寄存器与外设选择关系表

DMA2D_PERI_OUT_SEL_CH*n*	外设
0	JPEG
1	PPA 的 SRM 模块
2	PPA 的 BLENDO 模块
3	PPA 的 BLEND1 模块
4 ~ 7	Dummy-4 ~ 7

表 8.4-3. 2D-DMA RX 方向配置寄存器与外设选择关系表

DMA2D_PERI_IN_SEL_CH*n*	外设
0	JPEG
1	PPA 的 SRM 模块
2	PPA 的 BLEND 通道
3 ~ 7	Dummy-3 ~ 7

8.4.5 存储到存储数据传输

2D-DMA 只有 1D 模式和 2D-MODO 模式支持存储到存储的数据传输。置位 DMA2D_IN_MEM_TRANS_EN_CH*n*，TX 通道*n* 的输出将与 RX 通道*n* 的输入相连，从而使能存储到存储的数据传输功能。需要注意的是，一个 TX 通道只与其编号对应的 RX 通道相连而实现存储到存储的数据传输，并且需要将 DMA2D_IN_PERI_SEL_CH*n* 和 DMA2D_OUT_PERI_SEL_CH*n* 配置成 Dummy 的值。

可以搭配颜色空间转换功能，完成将一块宏块从一段存储器地址空间转移到另一段地址空间，并转换颜色空间。

8.4.6 宏块重排功能

8.4.6.1 发送方向宏块重排功能

当 2D-DMA 通过 SPI 访问外部存储器时，为了提高 SPI 的带宽利用率，需要以较大的 burst length 发起 AXI 传输。对于 2D-MOD1，2D-DMA 可以通过一次取若干个宏块来提高 AXI 的 burst length。例如，一次读取 5 个 8x8

的宏块，此时，描述符中的 hb 设为 40，vb 设为 8。而对于 JPEG，2D-DMA 需要按照宏块顺序给出数据。也就是需要重新排列取出的若干个宏块以满足 JPEG 的需求。

推荐宏块重排的配置见表 8.4-4。

表 8.4-4. 2D-DMA 发送方向宏块重排推荐配置表

byte/pixel	宏块大小（像素）	最佳宏块数	hb	vb
3	8 × 8	5	40	8
3	16 × 8	2	32	8
3	16 × 16	2	32	16
2	8 × 8	8	64	8
2	16 × 8	4	64	8
2	16 × 16	3	48	16
1	8 × 8	16	128	8

发送方向宏块重排需要配置宏块大小寄存器 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n*。

8.4.6.2 接收方向宏块重排功能

目前接收宏块重排仅支持 JPEG 解码（码流为 1 个 scan，详见章节 37 *JPEG 图像编解码器*）数据重排，将来自 JPEG 解码的颜色分量数据包拆解为像素格式并组装成较大的宏块通过 2D-DMA 写入存储器。

JPEG 解码的数据格式包括：GRAY、YUV444、YUV422、YUV420。推荐宏块重排的配置见表 8.4-5。

表 8.4-5. 2D-DMA 接收方向宏块重排推荐配置表（不做颜色格式转换）

颜色格式	宏块大小（像素）	最佳宏块数	hb	vb
GRAY	8 × 8	12	96	8
YUV444	8 × 8	5	40	8
YUV422	16 × 8	4	64	8
YUV420	16 × 16	3	48	16

接收方向宏块重排需要配置宏块大小寄存器 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n*。

JPEG 解码输出的宏块排布如图 8.4-4 所示，其中每一个蓝色正方形表示连续的 8x8 byte 数据。图中每一个颜色格式均展示了 4 个 8 × 8 像素的宏块。

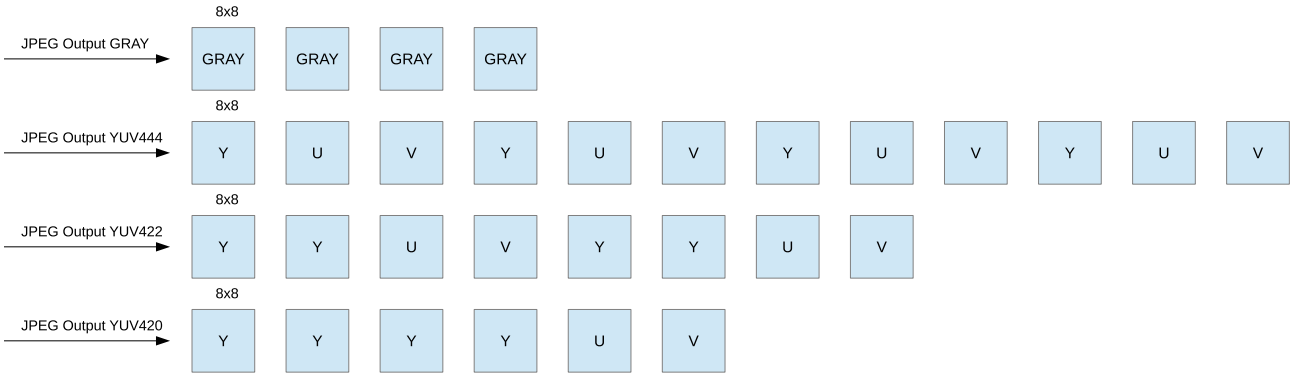


图 8.4-4. JPEG 解码输出数据宏块排布

8.4.7 颜色空间转换功能

发送侧将从存储器读取到的数据按像素组装后，像素数据可进行颜色空间转换。暂不支持 YUV420 格式转换。可支持：

- RGB888 转 RGB565
- RGB565 转 RGB888
- RGB888 转 YUV444
- RGB888 转 YUV422（RGB 一行像素个数需要为偶数个）
- YUV444 转 RGB888
- YUV422 转 RGB888

接收侧只有 JPEG 需要颜色空间转换，支持的格式如下：

- YUV444 转 RGB888/RGB565/YUV422/YUV420
- YUV422 转 YUV444 转 RGB888/RGB565/YUV420
- YUV420 转 YUV444 转 RGB888/RGB565

说明：

- 对于 JPEG 解码，即 2D-DMA 的接收侧，硬件会强制将 YUV422 和 YUV420 转换成 YUV444 格式，软件可选是否利用颜色空间转换功能，将其转换为 RGB888/RGB565。

颜色空间转换模块的结构示意图如 8.4-5 所示。

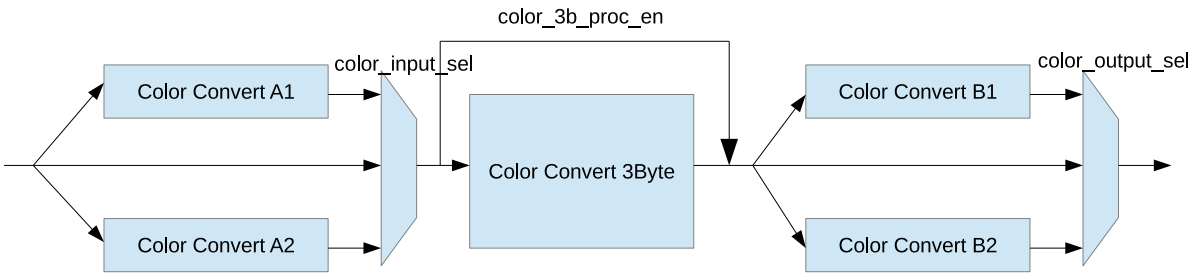


图 8.4-5. 颜色空间转换示意图

颜色空间转换分为 3 级进行，第一级进行输入颜色格式转换，可将特定的 2 Byte/pixel 的像素格式转换为 3 Byte/pixel，通过 DMA2D_OUT_COLOR_INPUT_SEL_CH*n* 或 DMA2D_IN_COLOR_INPUT_SEL_CH0 进行源数据选择。（可点击寄存器链接查看第一级支持的颜色格式转换）

第二级通过公式对选择后的源数据计算处理。计算公式如下：

$$256 * Q = A[9:0] * x + B[10:0] * y + C[9:0] * z + D[17:0]$$

其中 Q 为输出数据，x、y、z 为输入数据的高、中、低字节，A、B、C、D 为对应乘法/加法系数，为有符号数。同一时刻可处理最多 3 Byte 数据，因此需要通过以下字段配置 3*4=12 个参数：

- DMA2D_OUT_COLOR_PARAM_H0_CH*n*/DMA2D_IN_COLOR_PARAM_H0_CH0

- DMA2D_OUT_COLOR_PARAM_H1_CH*n*/DMA2D_IN_COLOR_PARAM_H1_CH0
- DMA2D_OUT_COLOR_PARAM_M0_CH*n*/DMA2D_IN_COLOR_PARAM_M0_CH0
- DMA2D_OUT_COLOR_PARAM_M1_CH*n*/DMA2D_IN_COLOR_PARAM_M1_CH0
- DMA2D_OUT_COLOR_PARAM_L0_CH*n*/DMA2D_IN_COLOR_PARAM_L0_CH0
- DMA2D_OUT_COLOR_PARAM_L1_CH*n*/DMA2D_IN_COLOR_PARAM_L1_CH0

字段名称中的 PARAM_H*x*、PARAM_M*x* 和 PARAM_L*x*（*x* 为 0 或 1）分别表示高、中、低三个字节各自的参数。以 PARAM_H*x* 为例，PARAM_H0 的低 10 位为系数 A，PARAM_H0 的高 11 位为系数 B，PARAM_H1 的低 10 位为系数 C，PARAM_H1 的高 18 位为系数 D。

第三级与第一级类似，通过 DMA2D_OUT_COLOR_OUTPUT_SEL_CH*n* 或 DMA2D_IN_COLOR_OUTPUT_SEL_CH0 选择直接输出或者进行第三级颜色空间转换模块后输出。

下表列出常用的颜色格式转换需要的参数配置：

表 8.4-6. TX 方向颜色格式转换参数配置

转换类型	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
不转换	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
仅乱序	2/3	0	2	N/A	N/A	N/A	N/A	N/A	N/A
RGB888 → RGB565	3	0	0	N/A	N/A	N/A	N/A	N/A	N/A
RGB565 → RGB888	0	0	2	N/A	N/A	N/A	N/A	N/A	N/A
RGB888 → (601) YUV444	3	1	2	20442	400019	1EDBDA	2000070	1E8870	20003EE
RGB888 → (709) YUV444	3	1	2	2742F	400010	1EABE6	2000070	1E6870	20003F6
RGB888 → (601) YUV422-MIPI	3	1	1	20442	400019	1EDBDA	2000070	1E8870	20003EE
RGB888 → (709) YUV422-MIPI	3	1	1	2742F	400010	1EABE6	2000070	1E6870	20003F6
YUV444 → (601) RGB888	3	1	2	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV444 → (709) RGB888	3	1	2	12A	C21E5CB	1F252A	1338778	8752A	B7D0800
YUV422-MIPI → (601) RGB888	1	1	2	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV422-MIPI → (709) RGB888	1	1	2	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

表 8.4-7. RX 方向颜色格式转换参数配置

转换类型	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
不转换	7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
仅乱序	1/2	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV422 → YUV444	0	0	1	N/A	N/A	N/A	N/A	N/A	N/A

见下页

表 8.4-7 – 接上页

转换类型	input_sel	proc_en	output_sel	param_h0	param_h1	param_m0	param_m1	param_l0	param_l1
YUV420 → YUV444	0	0	1	N/A	N/A	N/A	N/A	N/A	N/A
YUV444 → YUV422	0	0	2	N/A	N/A	N/A	N/A	N/A	N/A
YUV444 → YUV420	0	0	3	N/A	N/A	N/A	N/A	N/A	N/A
YUV422 → YUV420	0	0	3	N/A	N/A	N/A	N/A	N/A	N/A
YUV420 → (601) RGB888/RGB565	0	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV422 → (601) RGB888/RGB565	0	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV444 → (601) RGB888/RGB565	2	1	1/0	12A	C86D999	1E712A	21E4F30	8112A	BAD3000
YUV420 → (709) RGB888/RGB565	0	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800
YUV422 → (709) RGB888/RGB565	0	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800
YUV444 → (709) RGB888/RGB565	2	1	1/0	12A	C21E5CB	1F252A	1338778	8752A	B7D0800

另外，颜色格式转换模块支持单个像素（最大 3 byte）的字节乱序处理，发送侧在格式转换之前支持，接收侧在格式转换之前及之后均支持。颜色格式转换模块需要的像素字节顺序如表 8.4-8、表 8.4-9、表 8.4-10 所示。对于 JPEG 解码方向，接收侧宏块重排后在不配置乱序处理的情况下，像素字节顺序符合颜色格式转换所需字节顺序。假设输入数据的字节序 MSB 到 LSB 用 2、1、0 表示，字节乱序支持输出 210、201、102、120、021、012 六种顺序，默认输出为 2、1、0，可根据需要进行配置。可配置 DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 / DMA2D_IN_SCRAMBLE_SEL_POST_CH0 在颜色格式转换前 / 后对接收侧数据进行乱序处理。可配置 DMA2D_OUT_SCRAMBLE_SEL_PRE_CH*n* 在颜色格式转换前对发送侧数据进行乱序处理。

表 8.4-8. RGBB888 的存储格式

	LSB								MSB
LINE	B1	G1	R1	B2	G2	R2	B3	G3	R3

表 8.4-9. YUV444 的存储格式

	LSB								MSB
LINE	V1	U1	Y1	V2	U2	Y2	V3	U3	Y3

表 8.4-10. YUV422 的存储格式

	LSB							MSB
LINE	U1	Y1	V1	Y2	U3	Y3	V3	Y4

8.4.8 启动 2D-DMA

软件通过挂载链表的方式来使用 2D-DMA。对于接收通道，软件挂载好接收链表并准备好接收数据，配置 DMA2D_INLINK_ADDR_CH*n* 字段指向第一个接收链表描述符，置位 DMA2D_INLINK_START_CH*n* 位启动 2D-DMA。对于发送通道，软件挂载好发送链表并准备好发送数据，配置 DMA2D_OUTLINK_ADDR_CH*n* 字段指向第一个发送链表描述符，置位 DMA2D_OUTLINK_START_CH*n* 位启动 2D-DMA。
DMA2D_INLINK_START_CH*n* 与 DMA2D_OUTLINK_START_CH*n* 位由硬件自动清零。

有时您可能想要在 2D-DMA 数据传输已经开始后追加更多描述符。要挂载更多描述符，原本看似只需清空已挂载链表最后一个描述符的 eof 位，并将该描述符的 next descriptor address (DW4) 字段配置为新链表第一个描述符的地址。但如果 DMA 数据传输已经或即将结束，这个方法便无法奏效。2D-DMA 控制器有专门的逻辑来确保数据传输继续或重启：如果数据传输仍在进行，2D-DMA 控制器会确保顾及到新追加的描述符；如果数据传输已经结束，2D-DMA 控制器会重启数据传输，传输新追加的描述符。这个逻辑由 Restart 功能实现。

软件使用 Restart 功能时，需要先置位 DMA2D_INLINK_STOP_CH*n* 或者 DMA2D_OUTLINK_STOP_CH*n* 寄存器，让 2D-DMA 处理完当前描述符后暂停处理下一个描述符，然后重写已挂载链表的最后一个描述符，使其第五个字中的内容（即 DW4）指向新链表的首地址，最后置位 DMA2D_INLINK_RESTART_CH*n* 或者 DMA2D_OUTLINK_RESTART_CH*n*（这两个位由硬件自动清零），硬件会在读取已挂载链表的最后一个描述符时，获取新挂载链表的地址，从而继续处理新挂载的链表。

8.4.9 读链表

软件在配置并启动 2D-DMA 后，2D-DMA 会从存储器读取链表。2D-DMA 会检查读入的链表描述符是否正确。只有当链表描述符通过检查时，2D-DMA 对应的通道才会开始搬运数据。当链表描述符没有通过检查，硬件将触发描述符错误中断（DMA2D_IN_DSCR_ERR_CH*n*_INT 或者 DMA2D_OUT_DSCR_ERR_CH*n*_INT），同时该通道将会处于阻塞状态，停止工作，可重新置位 DMA2D_OUTLINK_START_CH*n* 或者 DMA2D_INLINK_START_CH*n* 再次开启该通道。

描述符检查项包括：

- DMA2D_IN_CHECK_OWNER_CH*n* 或者 DMA2D_OUT_CHECK_OWNER_CH*n* 置 1 时，检查描述符的 owner 位。如果该位为 0，表示当前操作者为 CPU，则不通过检查。DMA2D_IN_CHECK_OWNER_CH*n* 或者 DMA2D_OUT_CHECK_OWNER_CH*n* 置 0 时，不检查描述符的 owner 位。
- 检查描述符本身的存储地址是否在配置的存储器地址空间内。如果 2D-DMA 的描述符地址在 DMA2D_ACCESS_INTR_MEM_START_ADDR ~ DMA2D_ACCESS_INTR_MEM_END_ADDR（配置值应为片内存储器地址）或 DMA2D_ACCESS_EXTR_MEM_START_ADDR ~ DMA2D_ACCESS_EXTR_MEM_END_ADDR（配置值应为片外存储器地址）值的范围内，则通过检查。请参见章节 8.4.11）。
- 检查描述符的 HA/VA/hb/vb/SIZE 其中任意一个字段是否为 0，如果为 0，则不通过检查。其中 HA/VA/hb/vb 在 1D 模式下不检查，SIZE 仅在 1D 模式下检查。

软件在检查到通道描述符错误中断后，需要复位对应的通道，并置位 DMA2D_OUTLINK_START_CH*n* 或者 DMA2D_INLINK_START_CH*n* 位重新启动 2D-DMA。
可以将 DMA2D_INDSCR_BURST_EN_CH*n* 或 DMA2D_OUTDSCR_BURST_EN_CH*n* 置 1，使能 2D-DMA 以 AXI burst 传输快速从存储器获取接收/发送侧链表描述符。

8.4.10 数据传输结束标志

注意： 本章中提到发送链表描述符的 EOF 为描述符中的第一个字中的 bit 30 指示的 eof，接收链表描述符的

EOF 可以为 bit 30 指示的 eof 和 bit 28 指示的 err_eof。

2D-DMA 通过 EOF 来指示一张图片传输结束。

发送数据时，产生 EOF 和 TOTAL EOF 两种中断：

- DMA2D_OUT_EOF_CH*n*_INT：链表中的任意一个描述符有配置 eof 标志位，则当该描述符对应的数据传输完成后，2D-DMA 都会产生 EOF 中断。通常在一张图片的最后一个描述符配置 eof。置位 DMA2D_OUT_EOF_CH*n*_INT_ENA 位使能该中断。
- DMA2D_OUT_TOTAL_EOF_CH*n*_INT：如果该链表中的最后一个描述符配置了 eof 标志位，则当该链表的最后一个描述符对应的数据传输完成后，2D-DMA 会产生 TOTAL EOF 中断。置位 DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ENA 位使能该中断。

接收数据时，可以产生以下中断：

- DMA2D_IN_SUC_EOF_CH*n*_INT：表示一张图片数据接收完成。置位 DMA2D_IN_SUC_EOF_CH*n*_INT_ENA 位使能该中断。
- （仅 JPEG）DMA2D_IN_ERR_CH*n*_EOF_INT：表示一张图片接收完成但接收数据有错误。置位 DMA2D_IN_ERR_EOF_CH*n*_INT_ENA 使能该中断。需要注意的是，只有当通道连接的外设为 JPEG 时，才支持该中断。

软件在检测到 DMA2D_OUT_TOTAL_EOF_CH*n*_INT 或 DMA2D_IN_SUC_EOF_CH*n*_INT 中断时，可以通过获取 DMA2D_OUT_EOF_DES_ADDR_CH*n* 或 DMA2D_IN_SUC_EOF_DES_ADDR_CH*n* 字段中的的值，获取当前已经完成数据传输的描述符的存储地址。这样，软件可以知道哪些描述符已经被使用并根据需要回收描述符。RX 方向还支持记录 DMA2D_IN_ERR_EOF_CH*n*_INT 中断触发时描述符的存储地址，可通过 DMA2D_IN_ERR_EOF_DES_ADDR_CH*n* 字段获取。

8.4.11 访问存储

2D-DMA 任意 RX/TX 通道均可以访问片内和片外存储器，其可访问的片内地址空间为寄存器 DMA2D_ACCESS_INTR_MEM_START_ADDR ~ DMA2D_ACCESS_INTR_MEM_END_ADDR 的配置值区间，可访问的片外地址空间为 DMA2D_ACCESS_EXTR_MEM_START_ADDR ~ DMA2D_ACCESS_EXTR_MEM_END_ADDR 的配置值区间。

2D-DMA 利用 AXI 总线在片内进行数据传输，AXI 总线的突发传输模式默认打开且无法关闭，但是用户可以通过DMA2D_IN_MEM_BURST_LENGTH_CH*n* 和 DMA2D_OUT_MEM_BURST_LENGTH_CH*n* 来控制 AXI 总线突发传输传输的数据量，将突发传输长度配置为 8/16/32/64/128 字节。

2D-DMA 访问外部存储器的加密空间（详见章节 *34 片外存储器加密与解密 (XTS_AES)*）时，buffer address pointer 和对应数据量需按 16 字节对齐，其他参数无要求；访问配置的片内存储器空间和外部存储器的非加密空间时，链表描述符参数无对齐要求。

8.4.12 仲裁

为了确保及时响应高速低延迟外设的请求，2D-DMA 引入通道优先级仲裁和通道权重仲裁两种仲裁机制。

- 优先级仲裁方式：2D-DMA 支持 16（0 ~15）个等级的优先级，数字越大优先级越高。若多个通道在同一时刻存在数据传输需求，2D-DMA 会按照通道优先级从高到低顺序响应每个通道的数据传输请求；优先级越高，请求响应越及时。
- 权重仲裁方式：

- 每个通道的的权重（即令牌数）可以配置为 0 ~ 15 之间的任意值。2D-DMA 将时间按照一定的总线 (AXI) 时钟周期长度划分成片，在每个时间片内，各个通道的数据传输请求次数应该将按照通道所配置的令牌数进行分配，在任意一个时间片内，某通道每请求一次数据传输消耗一次令牌，若某个通道在某个时间片内消耗尽令牌数，则该通道请求将不再被响应，直到仲裁时间达到时间片规定长度或者所有通道的令牌数均已经消耗尽，此时将刷新令牌数，进入下一个时间片周期内。
- 若通道令牌数配置为非零值，即使无数据传输请求，也会在每个时间片内等待所有通道的令牌数耗尽，不会提前退出当前仲裁时间片。通道无数据传输请求且令牌值非 0 时，会导致一些带宽浪费，基于此，2D-DMA 提供额外的权重仲裁优化开关。权重仲裁优化打开时，如果通道无数据传输需求，则该通道令牌数会被认为是 0，不再参与仲裁，当所有通道均无数据传输请求时，仲裁器提前退出当前时间片，以此提高仲裁响应时间，提高传输速率。

说明：

- 若同时配置优先级仲裁和通道权重仲裁，且优先级不同，权重相同，则在各个通道传输相同的数据量，且数据量较少的前提下，优先级高的通道优先传输完成。
- 若优先级不同且权重也不同，各个通道传输数据量相同且较大，则权重较大的通道是能优先分配到大带宽的，会优先完成传输。

8.5 事件任务矩阵功能

在 ESP32-P4 中，2D-DMA 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 2D-DMA 的 ETM 任务，或者通过 2D-DMA 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 2D-DMA 相关的 ETM 任务和 ETM 事件。

2D-DMA 可接收的 ETM 任务有：

- DMA2D_TASK_IN_START_CH*n*：触发时开启 2D-DMA 的接收通道 *n* 进行数据传输。
- DMA2D_TASK_OUT_START_CH*n*：触发时开启 2D-DMA 的发送通道 *n* 进行数据传输。

说明：
以上两个 ETM 任务与 CPU 配置 DMA2D_INLINK_START_CH*n*、DMA2D_OUTLINK_START_CH*n* 实现的功能相同。当 DMA2D_IN_ETM_EN_CH*n* 或 DMA2D_OUT_ETM_EN_CH*n* 为 1 时，只有 ETM 可以配置对应方向及对应通道 2D-DMA 的启动，当 DMA2D_IN_ETM_EN_CH*n* 或 DMA2D_OUT_ETM_EN_CH*n* 为 0 时，仅可通过 CPU 使能对应的 2D-DMA。

- DMA2D_TASK_IN_DSCR_READY_CH*n*：触发时允许 2D-DMA 的接收通道 *n* 处理下一个链表描述符。
- DMA2D_TASK_OUT_DSCR_READY_CH*n*：触发时允许 2D-DMA 的发送通道 *n* 处理下一个链表描述符。

说明：
正常模式下 2D-DMA 开始工作后，每处理完一个链表描述符，如果还存在下一个链表描述符（即 next descriptor address 字段非 0），则会继续处理下一个链表描述符并按照描述符的描述信息读/写存储器，直到所有描述符都被处理完成。当配置 DMA2D_IN_ETM_LOOP_EN_CH*n* 或 DMA2D_OUT_ETM_LOOP_EN_CH*n* 为 1 时，2D-DMA 处理完当前描述符后会等待 DMA2D_TASK_IN_DSCR_READY_CH*n* 或 DMA2D_TASK_OUT_DSCR_READY_CH*n* 任务，才可以开始处理下一个链表描述符。
DMA2D_TASK_IN_DSCR_READY_CH*n* 或 DMA2D_TASK_OUT_DSCR_READY_CH*n* 任务可以存储到任务缓存中，

最多可以缓存 4 个任务。当 2D-DMA 接收到此任务，但是当前链表描述符还未处理完成时，可以缓存任务。任务缓存的深度由 DMA2D_IN_DSCR_TASK_MAX_CH*n* 或 DMA2D_OUT_DSCR_TASK_MAX_CH*n* 决定。
当任务缓存超过最大值时，会上报 DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT 或 DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT 中断。

2D-DMA 可产生的 ETM 事件有：

- DMA2D_EVT_IN_DONE_CH*n*：表示通道 *n* 接收方向链表描述符指向的数据接收完成。
- DMA2D_EVT_IN_SUC_EOF_CH*n*：通道 *n* 接收方向 eof 位为 1 的链表描述符指向的数据接收完成指示。
- DMA2D_EVT_OUT_DONE_CH*n*：通道 *n* 表示通道 *n* 发送方向链表描述符指向的数据发送完成。
- DMA2D_EVT_OUT_SUC_EOF_CH*n*：通道 *n* 发送方向 eof 位为 1 的链表描述符指向的数据发送完成指示。
- DMA2D_EVT_OUT_TOTAL_EOF_CH*n*：通道 *n* 发送方向最后一个并且 eof 位为 1 的链表描述指向的数据发送符完成指示。

在具体应用中，2D-DMA 的 ETM 事件可以用来触发 2D-DMA 的 ETM 任务，例如，DMA2D_EVT_OUT_TOTAL_EOF_CH0 事件可以触发 DMA2D_TASK_IN_START_CH1 任务，从而触发新一轮的 2D-DMA 操作。

8.6 中断

ESP32-P4 的 2D-DMA 可以生成以下中断信号并发送给中断矩阵。

- DMA2D_IN_CH0_INTR
- DMA2D_IN_CH1_INTR
- DMA2D_IN_CH2_INTR
- DMA2D_OUT_CH0_INTR
- DMA2D_OUT_CH1_INTR
- DMA2D_OUT_CH2_INTR
- DMA2D_OUT_CH3_INTR

这些中断信号由 2D-DMA 的内部中断源生成。

以下中断源生成 DMA2D_IN_CH*n*_INTR (*n* 为 0 ~ 2) 中断信号：

- DMA2D_IN_DONE_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成时触发此中断。
- DMA2D_IN_SUC_EOF_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成，并且描述符的 eof 为 1 时（即当一个接收链表对应的一张图片接收完成时）触发此中断。
- DMA2D_IN_ERR_EOF_CH*n*_INT：对于接收通道*n*，当接收的一张图片中有错误发生时触发此中断。(该中断只用于外设选择 JPEG 时)。
- DMA2D_IN_DSCR_EMPTY_CH*n*_INT：对于接收通道*n*，当接收链表描述符指向的 buffer size 小于待接收数据长度时触发此中断。
- DMA2D_IN_DSCR_ERR_CH*n*_INT：对于接收通道*n*，当接收链表描述符未通过任一描述符检查时触发此中断。

- DMA2D_INFIFO_L1/L2/L3_OVF_CH*n*_INT：当 2D-DMA 接收数据方向缓存写溢出时触发此中断。
- DMA2D_INFIFO_L1/L2/L3_UDF_CH*n*_INT：当 2D-DMA 接收数据方向缓存读溢出时触发此中断。
- DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT：当 2D-DMA 的 DMA2D_TASK_IN_DSCR_READY_CH*n* 任务缓存超过最大值时触发此中断。
- DMA2D_INFIFO_RO_OVF_CH0_INT（仅通道 0 支持）：当 2D-DMA 接收数据方向宏块重排功能的缓存写溢出时触发此中断。
- DMA2D_INFIFO_RO_UDF_CH0_INT（仅通道 0 支持）：当 2D-DMA 接收数据方向宏块重排功能的缓存读溢出时触发此中断。

以下中断源生成 DMA2D_OUT_CH*n*_INTR (*n* 为 0 ~ 3) 中断信号：

- DMA2D_OUT_DONE_CH*n*_INT：对于发送通道*n*，当一个发送链表描述符对应的数据发送完成时触发此中断。
- DMA2D_OUT_EOF_CH*n*_INT：对于发送通道*n*，当发送描述符的 eof 位为 1，且 DMA2D_OUT_EOF_MODE_CH*n* 为 0 时，该描述符对应的最后一个数据由总线传入 2D-DMA TX 通道时，该中断触发；当发送描述符的 eof 位为 1，且 DMA2D_OUT_EOF_MODE_CH*n* 为 1 时，该描述符对应的最后一个数据从 2D-DMA TX 通道发出时，该中断触发。
- DMA2D_OUT_DSCR_ERR_CH*n*_INT：对于发送通道*n*，当发送链表描述符未通过任一描述符检查时会触发此中断。
- DMA2D_OUT_TOTAL_EOF_CH*n*_INT：对于发送通道*n*，当一个链表（可包含多个链表描述符）对应的所有数据都已发送完成时触发此中断。
- DMA2D_OUTFIFO_L1/L2/L3_OVF_CH*n*_INT：当 2D-DMA 发送数据方向缓存写溢出时触发此中断。
- DMA2D_OUTFIFO_L1/L2/L3_UDF_CH*n*_INT：当 2D-DMA 发送数据方向缓存读溢出时触发此中断。
- DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT：当 2D-DMA 的 DMA2D_TASK_OUT_DSCR_READY_CH*n* 任务缓存超过最大值时时触发此中断。
- DMA2D_OUTFIFO_RO_OVF_CH0_INT（仅通道 0 支持）：当 2D-DMA 发送数据方向宏块重排/颜色空间转换功能的缓存写溢出时触发此中断。
- DMA2D_OUTFIFO_RO_UDF_CH0_INT（仅通道 0 支持）：当 2D-DMA 发送数据方向宏块重排/颜色空间转换功能的缓存读溢出时触发此中断。

8.7 编程流程

可通过配置 HP_SYS_CLKRST_DMA2D_SYS_CLK_EN 寄存器管理 2D-DMA 时钟门控，默认时钟门控打开。可通过配置 HP_SYS_CLKRST_RST_EN_DMA2D 对 2D-DMA 模块全局复位。

8.7.1 2D-DMA 通用配置

配置片内、片外存储器访问空间及突发传输的数据量，详见章节 8.4.11。

8.7.2 2D-DMA 模式专属配置

- 1D 模式：无专属配置

- 2D-MOD0 模式：无专属配置
- 2D-MOD1 模式：对于 JPEG 外设，需要配置寄存器 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* 或 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n*，选择读取宏块的基本单元，有三种尺寸可选（水平 × 垂直）：8 像素 × 8 像素、8 像素 × 16 像素、16 像素 × 16 像素，如何选择可参照 JPEG 相关章节。宏块单位大小与链表描述符 hb、vb 的选择可参照表 8.4-4、8.4-5。
- DSCR-PORT 模式：配置寄存器 DMA2D_OUT_DSCR_PORT_EN_CH*n* 为 1 进入该模式，该模式仅为 PPA 应用，具体操作可参照 PPA 相关章节。

8.7.3 2D-DMA TX 通道配置流程

利用 2D-DMA 发送数据时，2D-DMA TX 通道的软件配置流程如下：

1. 对寄存器 DMA2D_OUT_RST_CH*n* 置 1 然后置 0，复位 2D-DMA TX 通道状态机和 FIFO 指针；
2. 挂载好发送链表，配置寄存器 DMA2D_OUTLINK_ADDR_CH*n* 指向第一个发送链表描述符；
3. 配置 DMA2D_PERI_OUT_SEL_CH*n* 为对应的外设号，见表 8.4-2；
4. 置位 DMA2D_OUTLINK_START_CH*n* 启动 2D-DMA TX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；
6. 等待 DMA2D_OUT_EOF_CH*n*_INT 中断，即数据传输完成。

8.7.4 2D-DMA RX 通道配置流程

利用 2D-DMA 接收数据时，2D-DMA RX 通道的软件配置流程如下：

1. 对寄存器 DMA2D_IN_RST_CH*n* 置 1 然后置 0，复位 2D-DMA RX 通道状态机和 FIFO 指针；
2. 挂载好接收链表，配置寄存器 DMA2D_INLINK_ADDR_CH*n* 指向第一个接收链表描述符；
3. 配置 DMA2D_PERI_IN_SEL_CH*n* 为对应的外设号，见表 8.4-3；
4. 置位 DMA2D_INLINK_START_CH*n* 启动 2D-DMA RX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；
6. 等待 DMA2D_IN_SUC_EOF_CH*n*_INT 中断，即一张图片接收完成。

8.7.5 2D-DMA 存储器到存储器配置流程

利用 2D-DMA 从存储到存储搬运数据时配置流程如下：

1. 对寄存器 DMA2D_OUT_RST_CH*n* 置 1 然后置 0，复位 2D-DMA TX 通道状态机和 FIFO 指针；
2. 对寄存器 DMA2D_IN_RST_CH*n* 置 1 然后置 0，复位 2D-DMA RX 通道状态机和 FIFO 指针；
3. 挂载好发送链表，配置寄存器 DMA2D_OUTLINK_ADDR_CH*n* 指向第一个发送链表描述符；
4. 挂载好接收链表，配置寄存器 DMA2D_INLINK_ADDR_CH*n* 指向第一个接收链表描述符；
5. 配置 DMA2D_PERI_OUT_SEL_CH*n* 为某一个 Dummy 号，见表 8.4-2；
6. 配置 DMA2D_PERI_IN_SEL_CH*n* 为某一个 Dummy 号，见表 8.4-3；
7. 置位 DMA2D_MEM_TRANS_EN_CH*n* 使能存储器到存储器传输功能；

8. 置位 DMA2D_OUTLINK_START_CH*n* 启动 2D-DMA TX 通道发送数据；
9. 置位 DMA2D_INLINK_START_CH*n* 启动 2D-DMA RX 通道发送数据；
10. 等待 DMA2D_IN_SUC_EOF_CH*n*_INT 中断，即一次数据搬运完成。

8.7.6 通道优先级和权重配置流程

优先级仲裁配置方法如下：

1. 配置 DMA2D_OUT_ARB_PRIORITY_CH*n* 和 DMA2D_IN_ARB_PRIORITY_CH*n* 为 TX 和 RX 通道的优先级。

权重仲裁配置方法如下：

1. 配置 DMA2D_OUT_ARB_TIMEOUT_NUM 和 DMA2D_IN_ARB_TIMEOUT_NUM 为 TX 和 RX 权重仲裁的时间片长度；
2. 配置 DMA2D_OUT_ARB_TOKEN_NUM_CH*n* 和 DMA2D_IN_ARB_TOKEN_NUM_CH*n* 为各个通道的令牌值；
3. 清零 DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH_CH*n* 和DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n*，使能 TX 和 RX 方向的权重仲裁优化；
4. 置位 DMA2D_OUT_WEIGHT_EN 和 DMA2D_IN_WEIGHT_EN，使能权重仲裁。

8.7.7 工作状态下复位流程

当 2D-DMA 处于工作状态，即链表描述符未全部处理完成时，如果想要复位 2D-DMA 的 AXI 功能（包括 2D-DMA 系统复位 HP_SYS_CLKRST_RST_EN_DMA2D 及 AXI 功能复位 DMA2D_AXIM_RD_RST 和DMA2D_AXIM_WR_RST），必须先暂停当前 2D-DMA 工作，流程如下：

1. 置位 DMA2D_IN_CMD_DISABLE_CH*n* 和 DMA2D_OUT_CMD_DISABLE_CH*n* 寄存器，暂停 2D-DMA；
2. 等待 DMA2D_OUT_RESET_AVAIL_CH*n* 和 DMA2D_IN_RESET_AVAIL_CH*n* 寄存器置 1 后可进行复位操作；
3. 复位完成后将 DMA2D_IN_CMD_DISABLE_CH*n* 和 DMA2D_OUT_CMD_DISABLE_CH*n* 寄存器置 0。

8.8 寄存器列表

本小节的所有地址均为相对于 2D-DMA 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
DMA2D_OUT_CONF0_CH0_REG	配置 TX 通道 0	0x0000	R/W
DMA2D_OUT_PUSH_CH0_REG	配置 TX 通道 0 的 FIFO	0x0018	varies
DMA2D_OUT_LINK_CONF_CH0_REG	配置 TX 通道 0 发送链表描述符的操作	0x001C	varies
DMA2D_OUT_LINK_ADDR_CH0_REG	配置 TX 通道 0 发送链表描述符的地址	0x0020	R/W
DMA2D_OUT_ARB_CH0_REG	配置 TX 通道 0 的仲裁	0x003C	R/W
DMA2D_OUT_RO_PD_CONF_CH0_REG	配置 TX 通道 0 宏块重排存储器	0x0044	R/W
DMA2D_OUT_COLOR_CONVERT_CH0_REG	配置 TX 通道 0 的颜色空间转换功能	0x0048	R/W
DMA2D_OUT_SCRAMBLE_CH0_REG	配置 TX 通道 0 的乱序处理	0x004C	R/W
DMA2D_OUT_COLOR_PARAM0_CH0_REG	配置 TX 通道 0 的颜色空间转换参数	0x0050	R/W
DMA2D_OUT_COLOR_PARAM1_CH0_REG	配置 TX 通道 0 的颜色空间转换参数	0x0054	R/W
DMA2D_OUT_COLOR_PARAM2_CH0_REG	配置 TX 通道 0 的颜色空间转换参数	0x0058	R/W
DMA2D_OUT_COLOR_PARAM3_CH0_REG	配置 TX 通道 0 的颜色空间转换参数	0x005C	R/W
DMA2D_OUT_COLOR_PARAM4_CH0_REG	配置 TX 通道 0 的颜色空间转换参数	0x0060	R/W
DMA2D_OUT_COLOR_PARAM5_CH0_REG	配置 TX 通道 0 的颜色空间转换参数	0x0064	R/W
DMA2D_OUT_ETM_CONF_CH0_REG	配置 TX 通道 0 的 ETM	0x0068	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH0_REG	配置 DSCR-PORT 模式的 TX 通道 0 的块大小	0x006C	R/W
DMA2D_OUT_CONF0_CH1_REG	配置 TX 通道 1	0x0100	R/W
DMA2D_OUT_PUSH_CH1_REG	配置 TX 通道 1 的 FIFO	0x0118	varies
DMA2D_OUT_LINK_CONF_CH1_REG	配置 TX 通道 1 发送链表描述符的操作	0x011C	varies
DMA2D_OUT_LINK_ADDR_CH1_REG	配置 TX 通道 1 发送链表描述符的地址	0x0120	R/W
DMA2D_OUT_ARB_CH1_REG	配置 TX 通道 1 的仲裁	0x013C	R/W
DMA2D_OUT_COLOR_CONVERT_CH1_REG	配置 TX 通道 1 的颜色空间转换功能	0x0148	R/W
DMA2D_OUT_SCRAMBLE_CH1_REG	配置 TX 通道 1 的乱序处理	0x014C	R/W

GoBack

名称	描述	地址	访问
DMA2D_OUT_COLOR_PARAM0_CH1_REG	配置 TX 通道 1 的颜色空间转换参数	0x0150	R/W
DMA2D_OUT_COLOR_PARAM1_CH1_REG	配置 TX 通道 1 的颜色空间转换参数	0x0154	R/W
DMA2D_OUT_COLOR_PARAM2_CH1_REG	配置 TX 通道 1 的颜色空间转换参数	0x0158	R/W
DMA2D_OUT_COLOR_PARAM3_CH1_REG	配置 TX 通道 1 的颜色空间转换参数	0x015C	R/W
DMA2D_OUT_COLOR_PARAM4_CH1_REG	配置 TX 通道 1 的颜色空间转换参数	0x0160	R/W
DMA2D_OUT_COLOR_PARAM5_CH1_REG	配置 TX 通道 1 的颜色空间转换参数	0x0164	R/W
DMA2D_OUT_ETM_CONF_CH1_REG	配置 TX 通道 1 的 ETM	0x0168	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH1_REG	配置 DSCR-PORT 模式的 TX 通道 1 的块大小	0x016C	R/W
DMA2D_OUT_CONF0_CH2_REG	配置 TX 通道 2	0x0200	R/W
DMA2D_OUT_PUSH_CH2_REG	配置 TX 通道 2 的 FIFO	0x0218	varies
DMA2D_OUT_LINK_CONF_CH2_REG	配置 TX 通道 2 发送链表描述符的操作	0x021C	varies
DMA2D_OUT_LINK_ADDR_CH2_REG	配置 TX 通道 2 发送链表描述符的地址	0x0220	R/W
DMA2D_OUT_ARB_CH2_REG	配置 TX 通道 2 的仲裁	0x023C	R/W
DMA2D_OUT_COLOR_CONVERT_CH2_REG	配置 TX 通道 2 的颜色空间转换功能	0x0248	R/W
DMA2D_OUT_SCRAMBLE_CH2_REG	配置 TX 通道 2 的乱序处理	0x024C	R/W
DMA2D_OUT_COLOR_PARAM0_CH2_REG	配置 TX 通道 2 的颜色空间转换参数	0x0250	R/W
DMA2D_OUT_COLOR_PARAM1_CH2_REG	配置 TX 通道 2 的颜色空间转换参数	0x0254	R/W
DMA2D_OUT_COLOR_PARAM2_CH2_REG	配置 TX 通道 2 的颜色空间转换参数	0x0258	R/W
DMA2D_OUT_COLOR_PARAM3_CH2_REG	配置 TX 通道 2 的颜色空间转换参数	0x025C	R/W
DMA2D_OUT_COLOR_PARAM4_CH2_REG	配置 TX 通道 2 的颜色空间转换参数	0x0260	R/W
DMA2D_OUT_COLOR_PARAM5_CH2_REG	配置 TX 通道 2 的颜色空间转换参数	0x0264	R/W
DMA2D_OUT_ETM_CONF_CH2_REG	配置 TX 通道 2 的 ETM	0x0268	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH2_REG	配置 DSCR-PORT 模式的 TX 通道 2 的块大小	0x026C	R/W
DMA2D_OUT_CONF0_CH3_REG	配置 TX 通道 3	0x0300	R/W
DMA2D_OUT_PUSH_CH3_REG	配置 TX 通道 3 的 FIFO	0x0318	varies
DMA2D_OUT_LINK_CONF_CH3_REG	配置 TX 通道 3 发送链表描述符的操作	0x031C	varies
DMA2D_OUT_LINK_ADDR_CH3_REG	配置 TX 通道 3 发送链表描述符的地址	0x0320	R/W
DMA2D_OUT_ARB_CH3_REG	配置 TX 通道 3 的仲裁	0x033C	R/W
DMA2D_OUT_COLOR_CONVERT_CH3_REG	配置 TX 通道 3 的颜色空间转换功能	0x0348	R/W

GoBack

名称	描述	地址	访问
DMA2D_OUT_SCRAMBLE_CH3_REG	配置 TX 通道 3 的乱序处理	0x034C	R/W
DMA2D_OUT_COLOR_PARAM0_CH3_REG	配置 TX 通道 3 的颜色空间转换参数	0x0350	R/W
DMA2D_OUT_COLOR_PARAM1_CH3_REG	配置 TX 通道 3 的颜色空间转换参数	0x0354	R/W
DMA2D_OUT_COLOR_PARAM2_CH3_REG	配置 TX 通道 3 的颜色空间转换参数	0x0358	R/W
DMA2D_OUT_COLOR_PARAM3_CH3_REG	配置 TX 通道 3 的颜色空间转换参数	0x035C	R/W
DMA2D_OUT_COLOR_PARAM4_CH3_REG	配置 TX 通道 3 的颜色空间转换参数	0x0360	R/W
DMA2D_OUT_COLOR_PARAM5_CH3_REG	配置 TX 通道 3 的颜色空间转换参数	0x0364	R/W
DMA2D_OUT_ETM_CONF_CH3_REG	配置 TX 通道 3 的 ETM	0x0368	R/W
DMA2D_OUT_DSCR_PORT_BLK_CH3_REG	配置 DSCR-PORT 模式的 TX 通道 3 的块大小	0x036C	R/W
DMA2D_IN_CONF0_CH0_REG	配置 RX 通道 0	0x0500	R/W
DMA2D_IN_POP_CH0_REG	配置 RX 通道 0 的 FIFO	0x0518	varies
DMA2D_IN_LINK_CONF_CH0_REG	配置 RX 通道 0 接收链表描述符的操作	0x051C	varies
DMA2D_IN_LINK_ADDR_CH0_REG	配置 RX 通道 0 接收链表描述符的地址	0x0520	R/W
DMA2D_IN_ARB_CH0_REG	配置 RX 通道 0 的仲裁	0x0540	R/W
DMA2D_IN_RO_PD_CONF_CH0_REG	配置 RX 通道 0 的宏块重排存储器	0x0548	R/W
DMA2D_IN_COLOR_CONVERT_CH0_REG	配置 RX 通道 0 的颜色空间转换功能	0x054C	R/W
DMA2D_IN_SCRAMBLE_CH0_REG	配置 RX 通道 0 的乱序处理	0x0550	R/W
DMA2D_IN_COLOR_PARAM0_CH0_REG	配置 RX 通道 0 的颜色空间转换参数	0x0554	R/W
DMA2D_IN_COLOR_PARAM1_CH0_REG	配置 RX 通道 0 的颜色空间转换参数	0x0558	R/W
DMA2D_IN_COLOR_PARAM2_CH0_REG	配置 RX 通道 0 的颜色空间转换参数	0x055C	R/W
DMA2D_IN_COLOR_PARAM3_CH0_REG	配置 RX 通道 0 的颜色空间转换参数	0x0560	R/W
DMA2D_IN_COLOR_PARAM4_CH0_REG	配置 RX 通道 0 的颜色空间转换参数	0x0564	R/W
DMA2D_IN_COLOR_PARAM5_CH0_REG	配置 RX 通道 0 的颜色空间转换参数	0x0568	R/W
DMA2D_IN_ETM_CONF_CH0_REG	配置 RX 通道 0 的 ETM	0x056C	R/W
DMA2D_IN_CONF0_CH1_REG	配置 RX 通道 1	0x0600	R/W
DMA2D_IN_POP_CH1_REG	配置 RX 通道 1 的 RX FIFO	0x0618	varies
DMA2D_IN_LINK_CONF_CH1_REG	配置 RX 通道 1 接收链表描述符的操作	0x061C	varies
DMA2D_IN_LINK_ADDR_CH1_REG	配置 RX 通道 1 接收链表描述符的地址	0x0620	R/W
DMA2D_IN_ARB_CH1_REG	配置 RX 通道 1 的仲裁	0x0640	R/W

名称	描述	地址	访问
DMA2D_IN_ETM_CONF_CH1_REG	配置 RX 通道 1 的 ETM	0x0648	R/W
DMA2D_IN_CONF0_CH2_REG	配置 RX 通道 2	0x0700	R/W
DMA2D_IN_POP_CH2_REG	配置 RX 通道 2 的 RX FIFO	0x0718	varies
DMA2D_IN_LINK_CONF_CH2_REG	配置 RX 通道 2 接收链表描述符的操作	0x071C	varies
DMA2D_IN_LINK_ADDR_CH2_REG	配置 RX 通道 2 接收链表描述符的地址	0x0720	R/W
DMA2D_IN_ARB_CH2_REG	配置 RX 通道 2 的仲裁	0x0740	R/W
DMA2D_IN_ETM_CONF_CH2_REG	配置 RX 通道 2 的 ETM	0x0748	R/W
DMA2D_RST_CONF_REG	配置 AXI 复位	0x0A04	R/W
DMA2D_INTR_MEM_START_ADDR_REG	内部存储器可访问地址空间的起始地址	0x0A08	R/W
DMA2D_INTR_MEM_END_ADDR_REG	内部存储器可访问地址空间的结束地址	0x0A0C	R/W
DMA2D_EXTR_MEM_START_ADDR_REG	外部存储器可访问地址空间的起始地址	0x0A10	R/W
DMA2D_EXTR_MEM_END_ADDR_REG	外部存储器可访问地址空间的结束地址	0x0A14	R/W
DMA2D_OUT_ARB_CONFIG_REG	配置 TX 方向的仲裁	0x0A18	R/W
DMA2D_IN_ARB_CONFIG_REG	配置 RX 方向的仲裁	0x0A1C	R/W
中断寄存器			
DMA2D_OUT_INT_RAW_CH0_REG	TX 通道 0 的原始中断状态	0x0004	R/WTC/SS
DMA2D_OUT_INT_ENA_CH0_REG	TX 通道 0 的中断使能位	0x0008	R/W
DMA2D_OUT_INT_ST_CH0_REG	TX 通道 0 的屏蔽中断状态	0x000C	RO
DMA2D_OUT_INT_CLR_CH0_REG	TX 通道 0 的中断清除位	0x0010	WT
DMA2D_OUT_INT_RAW_CH1_REG	TX 通道 1 的原始中断状态	0x0104	R/WTC/SS
DMA2D_OUT_INT_ENA_CH1_REG	TX 通道 1 的中断使能位	0x0108	R/W
DMA2D_OUT_INT_ST_CH1_REG	TX 通道 1 的屏蔽中断状态	0x010C	RO
DMA2D_OUT_INT_CLR_CH1_REG	TX 通道 1 的中断清除位	0x0110	WT
DMA2D_OUT_INT_RAW_CH2_REG	TX 通道 2 的原始中断状态	0x0204	R/WTC/SS
DMA2D_OUT_INT_ENA_CH2_REG	TX 通道 2 的中断使能位	0x0208	R/W
DMA2D_OUT_INT_ST_CH2_REG	TX 通道 2 的屏蔽中断状态	0x020C	RO
DMA2D_OUT_INT_CLR_CH2_REG	TX 通道 2 的中断清除位	0x0210	WT
DMA2D_OUT_INT_RAW_CH3_REG	TX 通道 3 的原始中断状态	0x0304	R/WTC/SS
DMA2D_OUT_INT_ENA_CH3_REG	TX 通道 3 的中断使能位	0x0308	R/W

名称	描述	地址	访问
DMA2D_OUT_INT_ST_CH3_REG	TX 通道 3 的屏蔽中断状态	0x030C	RO
DMA2D_OUT_INT_CLR_CH3_REG	TX 通道 3 的中断清除位	0x0310	WT
DMA2D_IN_INT_RAW_CH0_REG	RX 通道 0 的原始中断状态	0x0504	R/WTC/SS
DMA2D_IN_INT_ENA_CH0_REG	RX 通道 0 的中断使能位	0x0508	R/W
DMA2D_IN_INT_ST_CH0_REG	RX 通道 0 的屏蔽中断状态	0x050C	RO
DMA2D_IN_INT_CLR_CH0_REG	RX 通道 0 的中断清除位	0x0510	WT
DMA2D_IN_INT_RAW_CH1_REG	RX 通道 1 的原始中断状态	0x0604	R/WTC/SS
DMA2D_IN_INT_ENA_CH1_REG	RX 通道 1 的中断使能位	0x0608	R/W
DMA2D_IN_INT_ST_CH1_REG	RX 通道 1 的屏蔽中断状态	0x060C	RO
DMA2D_IN_INT_CLR_CH1_REG	RX 通道 1 的中断清除位	0x0610	WT
DMA2D_IN_INT_RAW_CH2_REG	RX 通道 2 的原始中断状态	0x0704	R/WTC/SS
DMA2D_IN_INT_ENA_CH2_REG	RX 通道 2 的中断使能位	0x0708	R/W
DMA2D_IN_INT_ST_CH2_REG	RX 通道 2 的屏蔽中断状态	0x070C	RO
DMA2D_IN_INT_CLR_CH2_REG	RX 通道 2 的中断清除位	0x0710	WT
状态寄存器			
DMA2D_OUTFIFO_STATUS_CH0_REG	表示 TX 通道 0 的 FIFO 状态	0x0014	RO
DMA2D_OUT_STATE_CH0_REG	表示 TX 通道 0 发送链表描述符的工作状态	0x0024	RO
DMA2D_OUT_EOF_DES_ADDR_CH0_REG	表示 TX 通道 0 传输结束 (EOF) 时的发送链表描述符地址	0x0028	RO
DMA2D_OUT_DSCR_CH0_REG	表示 TX 通道 0 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x002C	RO
DMA2D_OUT_DSCR_BF0_CH0_REG	表示 TX 通道 0 当前已预读取的发送链表描述符地址	0x0030	RO
DMA2D_OUT_DSCR_BF1_CH0_REG	表示 TX 通道 0 上一个已预读取的发送链表描述符地址	0x0034	RO
DMA2D_OUTFIFO_STATUS_CH1_REG	表示 TX 通道 1 的 FIFO 状态	0x0114	RO
DMA2D_OUT_STATE_CH1_REG	表示 TX 通道 0 发送链表描述符的工作状态	0x0124	RO
DMA2D_OUT_EOF_DES_ADDR_CH1_REG	表示 TX 通道 1 传输结束 (EOF) 时的发送链表描述符地址	0x0128	RO
DMA2D_OUT_DSCR_CH1_REG	表示 TX 通道 1 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x012C	RO
DMA2D_OUT_DSCR_BF0_CH1_REG	表示 TX 通道 1 当前已预读取的发送链表描述符地址	0x0130	RO
DMA2D_OUT_DSCR_BF1_CH1_REG	表示 TX 通道 1 上一个已预读取的发送链表描述符地址	0x0134	RO

名称	描述	地址	访问
DMA2D_OUTFIFO_STATUS_CH2_REG	表示 TX 通道 2 的 FIFO 状态	0x0214	RO
DMA2D_OUT_STATE_CH2_REG	表示 TX 通道 0 发送链表描述符的工作状态	0x0224	RO
DMA2D_OUT_EOF_DES_ADDR_CH2_REG	表示 TX 通道 2 传输结束 (EOF) 时的发送链表描述符地址	0x0228	RO
DMA2D_OUT_DSCR_CH2_REG	表示 TX 通道 2 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x022C	RO
DMA2D_OUT_DSCR_BF0_CH2_REG	表示 TX 通道 2 当前已预读取的发送链表描述符地址	0x0230	RO
DMA2D_OUT_DSCR_BF1_CH2_REG	表示 TX 通道 2 上一个已预读取的发送链表描述符地址	0x0234	RO
DMA2D_INFIFO_STATUS_CH0_REG	表示 RX 通道 0 的 FIFO 状态	0x0514	RO
DMA2D_IN_STATE_CH0_REG	表示 RX 通道 0 接收链表描述符的工作状态	0x0524	RO
DMA2D_IN_SUC_EOF_DES_ADDR_CH0_REG	表示 RX 通道 0 传输结束 (EOF) 时的接收链表描述符地址	0x0528	RO
DMA2D_IN_ERR_EOF_DES_ADDR_CH0_REG	表示 RX 通道 0 上发生错误时的接收链表描述符地址	0x052C	RO
DMA2D_IN_DSCR_CH0_REG	表示 RX 通道 0 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0530	RO
DMA2D_IN_DSCR_BF0_CH0_REG	表示 RX 通道 0 当前已预读取的接收链表描述符地址	0x0534	RO
DMA2D_IN_DSCR_BF1_CH0_REG	表示 RX 通道 0 上一个已预读取的接收链表描述符地址	0x0538	RO
DMA2D_INFIFO_STATUS_CH1_REG	表示 RX 通道 1 的 FIFO 状态	0x0614	RO
DMA2D_IN_STATE_CH1_REG	表示 RX 通道 1 接收链表描述符的工作状态	0x0624	RO
DMA2D_IN_SUC_EOF_DES_ADDR_CH1_REG	表示 RX 通道 1 传输结束 (EOF) 时的接收链表描述符地址	0x0628	RO
DMA2D_IN_ERR_EOF_DES_ADDR_CH1_REG	表示 RX 通道 1 上发生错误时的接收链表描述符地址	0x062C	RO
DMA2D_IN_DSCR_CH1_REG	表示 RX 通道 1 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0630	RO
DMA2D_IN_DSCR_BF0_CH1_REG	表示 RX 通道 1 当前已预读取的接收链表描述符地址	0x0634	RO
DMA2D_IN_DSCR_BF1_CH1_REG	表示 RX 通道 1 上一个已预读取的接收链表描述符地址	0x0638	RO
DMA2D_AXI_ERR_REG	表示 AXI 总线的状态	0x0A00	RO
DMA2D_DATE_REG	版本寄存器	0x0A2C	R/W
外设选择寄存器			
DMA2D_OUT_PERI_SEL_CH0_REG	配置 TX 通道 0 连接的外设	0x0038	R/W
DMA2D_OUT_PERI_SEL_CH1_REG	配置 TX 通道 1 连接的外设	0x0138	R/W
DMA2D_OUT_PERI_SEL_CH2_REG	配置 TX 通道 2 连接的外设	0x0238	R/W

名称	描述	地址	访问
DMA2D_OUT_PERI_SEL_CH3_REG	配置 TX 通道 3 连接的外设	0x0338	R/W
DMA2D_IN_PERI_SEL_CH0_REG	配置 RX 通道 0 连接的外设	0x053C	R/W
DMA2D_IN_PERI_SEL_CH1_REG	配置 RX 通道 1 连接的外设	0x063C	R/W
DMA2D_IN_PERI_SEL_CH2_REG	配置 RX 通道 2 连接的外设	0x073C	R/W

8.9 寄存器

本小节的所有地址均为相对于 2D-DMA 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看看章节 如何配置寄存器的保留域。

Register 8.1. DMA2D_OUT_CONF0_CHn_REG (n: 0-3) (0x0000+0x100*n)

Bits	Field	Reset
31:27	(reserved)	0 0 0 0 0
26	DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CHn	0
25	DMA2D_OUT_CMD_DISABLE_CHn	0
24	DMA2D_OUT_RST_CHn	0
23:17	(reserved)	0 0 0 0 0 0 0
16	DMA2D_OUT_REORDER_EN_CH0	0
15:12	(reserved)	0 0 0 0
11	DMA2D_OUT_DSCR_PORT_EN_CHn	0
10:9	DMA2D_OUT_MACRO_BLOCK_SIZE_CHn	0x0
8:6	DMA2D_OUT_MEM_BURST_LENGTH_CHn	0x0
5	(reserved)	0
4	DMA2D_OUT_CHECK_OWNER_CHn	0
3	DMA2D_OUT_ECC_AES_EN_CHn	0
2	DMA2D_OUTDSCR_BURST_EN_CHn	0
1	DMA2D_OUT_EOF_MODE_CHn	1
0	DMA2D_OUT_AUTO_WRBACK_CHn	0

DMA2D_OUT_AUTO_WRBACK_CHn 配置在发送完 TX FIFO 中的所有数据时是否开启发送链表自动回写。

0：关闭

1：开启

(R/W)

DMA2D_OUT_EOF_MODE_CHn 配置生成 EOF 标志的时间。

0：待发送数据推入 2DDMA FIFO 时生成发送通道 n 的 EOF 标志。

1：待发送数据从 2DDMA FIFO 中弹出时生成发送通道 n 的 EOF 标志。

(R/W)

DMA2D_OUTDSCR_BURST_EN_CHn 配置是否开启 INCR 突发传输，让发送通道 n 读取描述符。

0：关闭

1：开启

(R/W)

DMA2D_OUT_ECC_AES_EN_CHn 配置是否开启发送通道 n 访问外部存储器 ECC 和 AES 加密区间的权限。

0：关闭

1：开启

此时加密区间的起始地址和响应数据需 16 字节对齐。

(R/W)

见下页...

Register 8.1. DMA2D_OUT_CONF0_CH*n*_REG (*n*: 0-3) (0x0000+0x100**n*)

接上页...

DMA2D_OUT_CHECK_OWNER_CH*n* 配置发送通道 *n* 是否开启 owner 位检查。

0：关闭

1：开启

(R/W)

DMA2D_OUT_MEM_BURST_LENGTH_CH*n* 配置发送通道 *n* 的突发传输长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

其他值：无效

(R/W)

DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* 配置发送通道 *n* 的宏块大小。仅用于 2D-MOD1 模式。

0：8 * 8 像素

1：8 * 16 像素

2：16 *16 像素

3：无宏块

(R/W)

DMA2D_OUT_DSCR_PORT_EN_CH*n* 配置发送通道 *n* 是否开启 DSCR-PORT 模式。

0：关闭

1：开启

(R/W)

DMA2D_OUT_REORDER_EN_CH0 配置发送通道 0 是否开启宏块重排。

0：关闭

1：开启

(R/W)

DMA2D_OUT_RST_CH*n* 配置是否复位发送通道 *n*。

0：复位释放

1：复位

(R/W)

见下页...

Register 8.1. DMA2D_OUT_CONF0_CH*n*_REG (*n*: 0-3) (0x0000+0x100**n*)

接上页...

DMA2D_OUT_CMD_DISABLE_CH*n* 配置发送通道 *n* 的复位指令。

0：复位释放时清零

1：复位前暂停传输

(R/W)

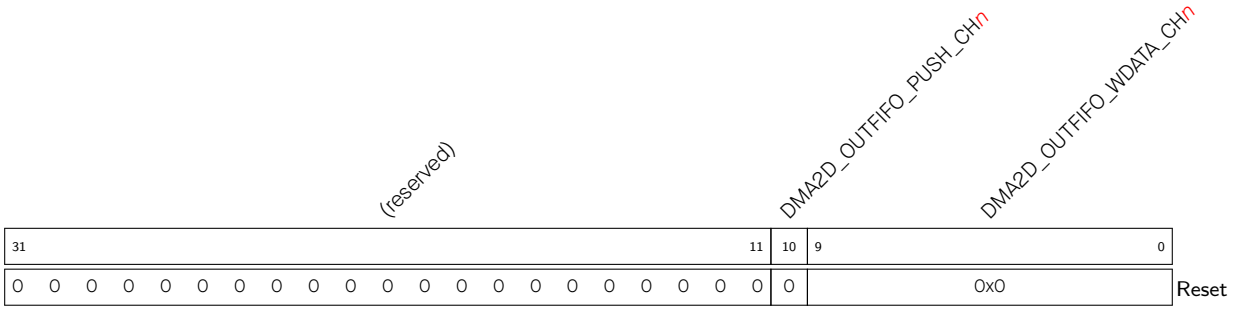
DMA2D_OUT_ARB_WEIGHT_OPT_DIS_CH*n* 配置发送通道 *n* 是否开启权重仲裁优化。

0：开启

1：关闭

(R/W)

Register 8.2. DMA2D_OUT_PUSH_CH*n*_REG (*n*: 0-3) (0x0018+0x100**n*)



DMA2D_OUTFIFO_WDATA_CH*n* 配置需要推入 2D-DMA TX FIFO 的数据。(R/W)

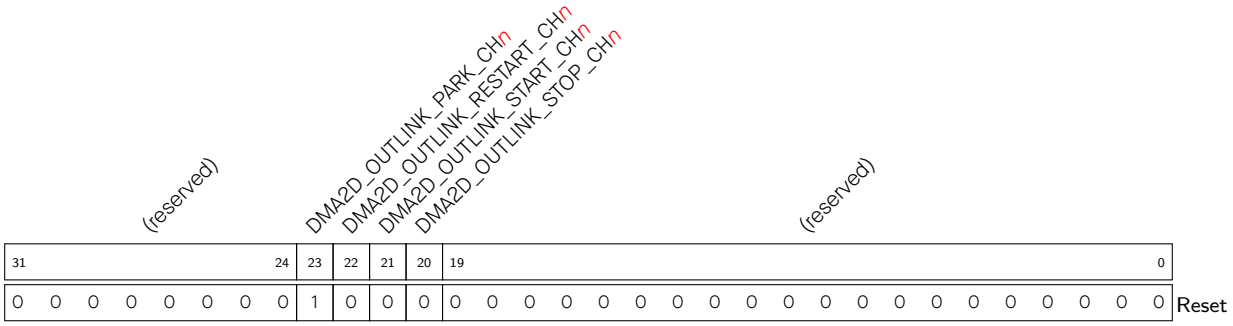
DMA2D_OUTFIFO_PUSH_CH*n* 配置是否将数据推入 2D-DMA TX FIFO。

0：不推送

1：推送

(R/W/SC)

Register 8.3. DMA2D_OUT_LINK_CONF_CH*n*_REG (*n*: 0-3) (0x001C+0x100**n*)



DMA2D_OUTLINK_STOP_CH*n* 配置是否停止发送通道 *n* 的数据传输。

0：无效值。没有作用

1：停止

(R/W/SC)

DMA2D_OUTLINK_START_CH*n* 配置是否开启发送通道 *n* 进行数据传输。

0：关闭

1：开启

(R/W/SC)

DMA2D_OUTLINK_RESTART_CH*n* 配置是否重启发送通道 *n* 进行 2D-DMA 传输。

0：无效值。没有作用

1：重启

(R/W/SC)

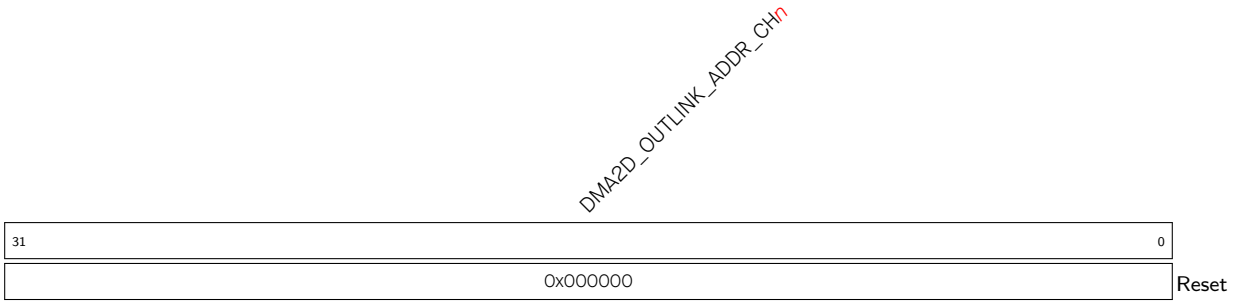
DMA2D_OUTLINK_PARK_CH*n* 表示发送链表描述符的 FSM 的状态。

0：运行

1：空闲

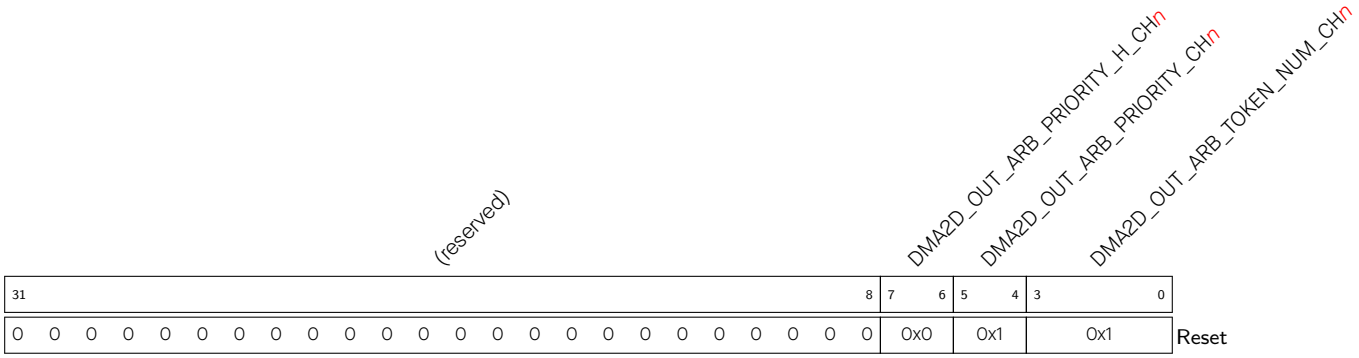
(RO)

Register 8.4. DMA2D_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-3) (0x0020+0x100**n*)



DMA2D_OUTLINK_ADDR_CH*n* 表示第一个发送链表描述符的地址。(R/W)

Register 8.5. DMA2D_OUT_ARB_CH*n*_REG (*n*: 0-3) (0x003C+0x100**n*)



DMA2D_OUT_ARB_TOKEN_NUM_CH*n* 配置发送通道 *n* 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

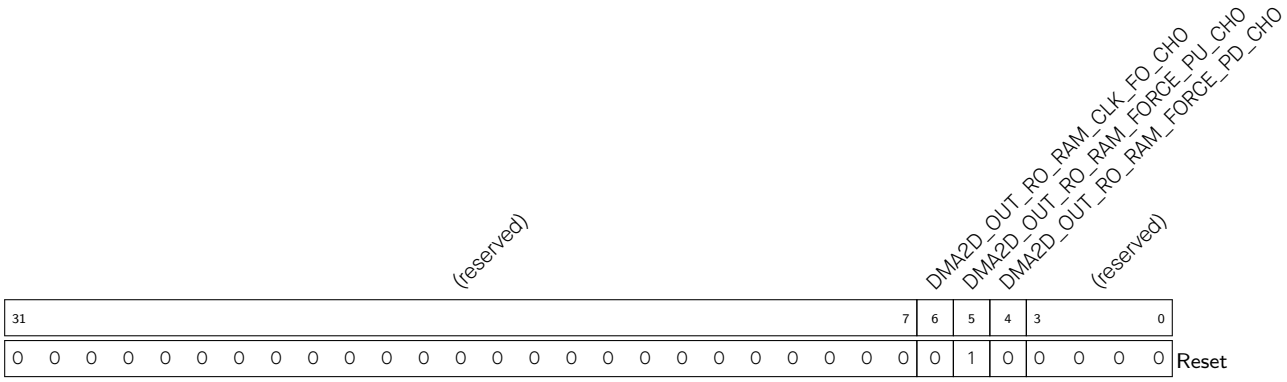
DMA2D_OUT_ARB_PRIORITY_CH*n* 配置发送通道 *n* 优先级的低两位。

该字段和 DMA2D_OUT_ARB_PRIORITY_H_CH*n* 共同配置发送通道 *n* 的四位优先级数值，最低值为 0，最高值为 15。(R/W)

DMA2D_OUT_ARB_PRIORITY_H_CH*n* 配置发送通道 *n* 优先级的高两位。

该字段和 DMA2D_OUT_ARB_PRIORITY_CH*n* 共同配置发送通道 *n* 的四位优先级数值，最低值为 0，最高值为 15。(R/W)

Register 8.6. DMA2D_OUT_RO_PD_CONF_CH0_REG (0x0044)



DMA2D_OUT_RO_RAM_FORCE_PD_CH0 配置是否强制关闭发送通道 0 的宏块重排 RAM 的电源。

0：不强制关闭电源

1：强制关闭电源

(R/W)

DMA2D_OUT_RO_RAM_FORCE_PU_CH0 配置是否强制开启发送通道 0 的宏块重排 RAM 的电源。

0：不强制开启电源

1：强制开启电源

(R/W)

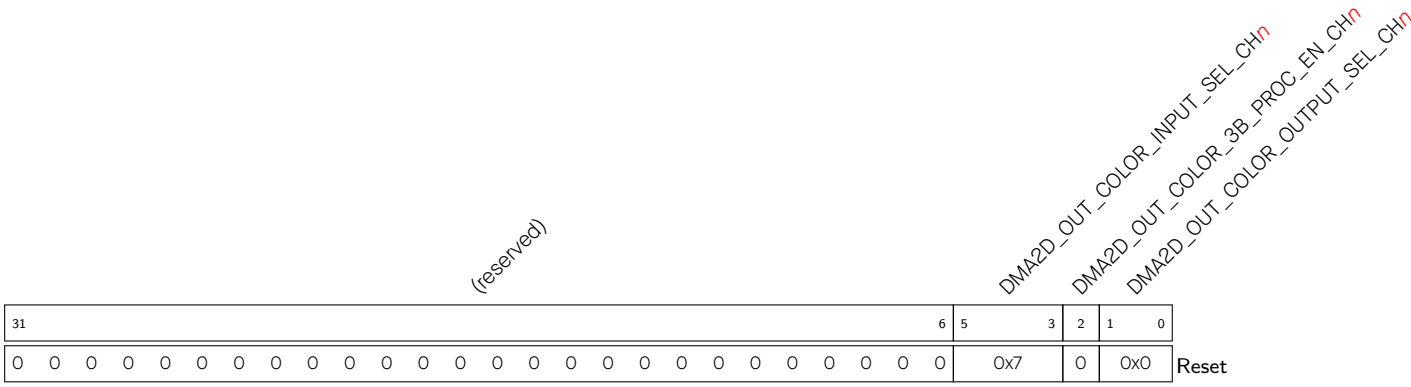
DMA2D_OUT_RO_RAM_CLK_FO_CH0 配置发送通道 0 访问 2D-DMA RAM 的时钟门控。

0：使用时钟门控

1：强制开启时钟，绕过时钟门控

(R/W)

Register 8.7. DMA2D_OUT_COLOR_CONVERT_CH*n*_REG (*n*: 0-3) (0x0048+0x100**n*)



DMA2D_OUT_COLOR_OUTPUT_SEL_CH*n* 配置发送通道 *n* 颜色空间转换第三级的输出。

0：RGB888 转 RGB565

1：YUV444 转 YUV422

2：直接输出

其他值：无效

(R/W)

DMA2D_OUT_COLOR_3B_PROC_EN_CH*n* 配置是否开启发送通道 *n* 颜色空间转换的第二级。

0：关闭

1：开启

(R/W)

DMA2D_OUT_COLOR_INPUT_SEL_CH*n* 配置发送通道 *n* 颜色空间转换第一级的输出。

0：RGB565 转 RGB888

1：YUV422 转 YUV444

2：其他 2 字节/像素的颜色格式

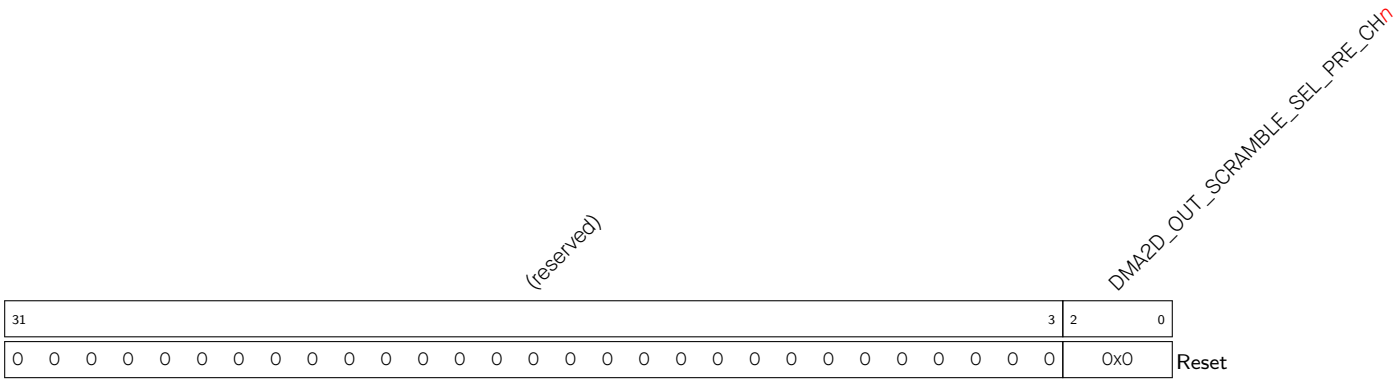
3：其他 3 字节/像素的颜色格式

7：关闭颜色空间转换

其他值：无效

(R/W)

Register 8.8. DMA2D_OUT_SCRAMBLE_CH*n*_REG (*n*: 0-3) (0x004C+0x100*n*)



DMA2D_OUT_SCRAMBLE_SEL_PRE_CH*n* 配置发送通道 *n* 颜色空间转换前的字节乱序处理。

0：BYTE2-1-0

1：BYTE2-0-1

2：BYTE1-0-2

3：BYTE1-2-0

4：BYTE0-2-1

5：BYTE0-1-2

其他值：无效

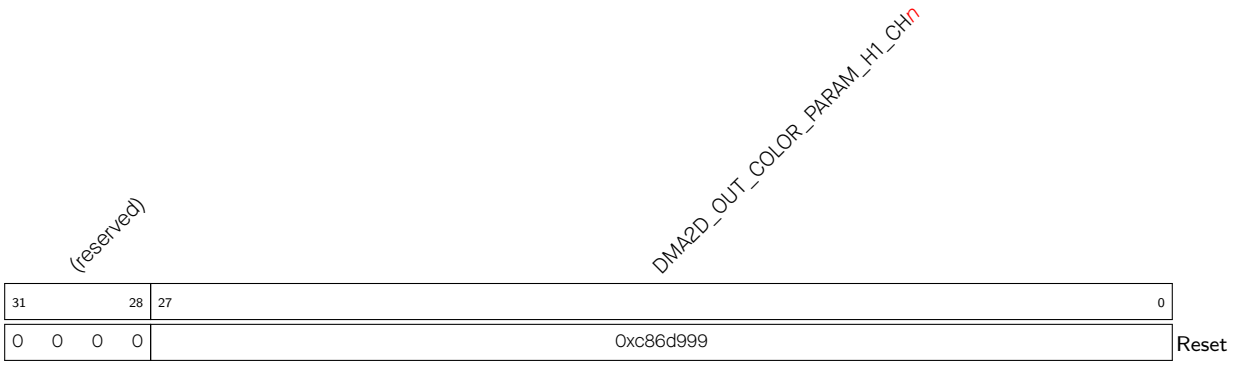
(R/W)

Register 8.9. DMA2D_OUT_COLOR_PARAM0_CH*n*_REG (*n*: 0-3) (0x0050+0x100*n*)



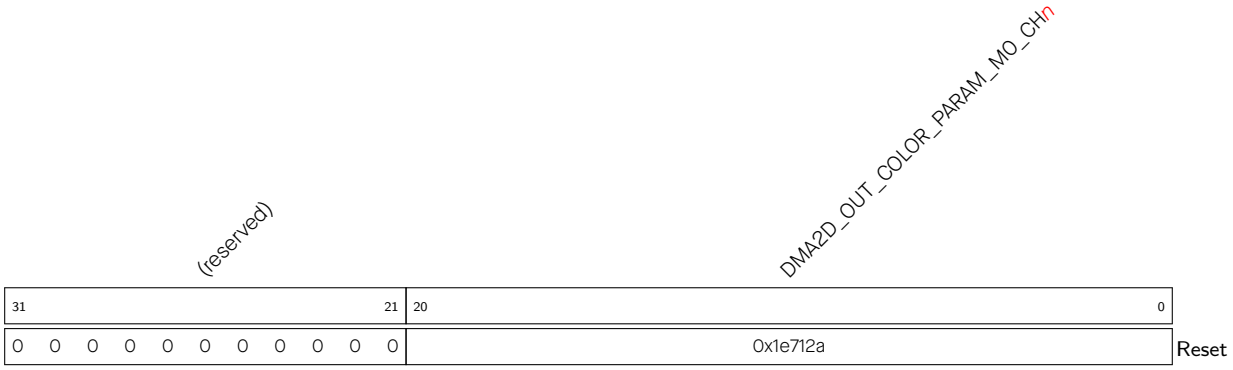
DMA2D_OUT_COLOR_PARAM_H0_CH*n* 配置发送通道 *n* 颜色空间转换第二级输入的最高有效字节的系数 A 和 B。(R/W)

Register 8.10. DMA2D_OUT_COLOR_PARAM1_CH*n*_REG (*n*: 0-3) (0x0054+0x100*n*)



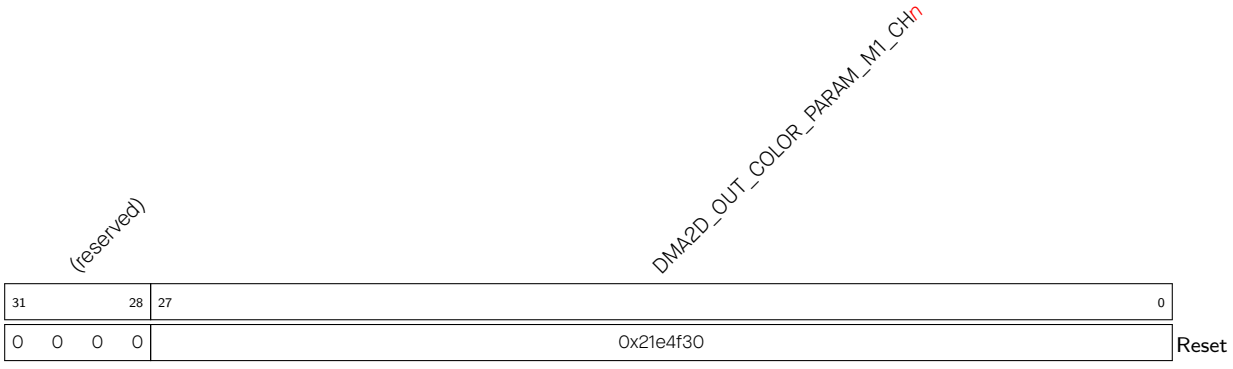
DMA2D_OUT_COLOR_PARAM_H1_CH*n* 配置发送通道 *n* 颜色空间转换第二级输入的最高有效字节的系数 C 和 D。(R/W)

Register 8.11. DMA2D_OUT_COLOR_PARAM2_CH*n*_REG (*n*: 0-3) (0x0058+0x100*n*)



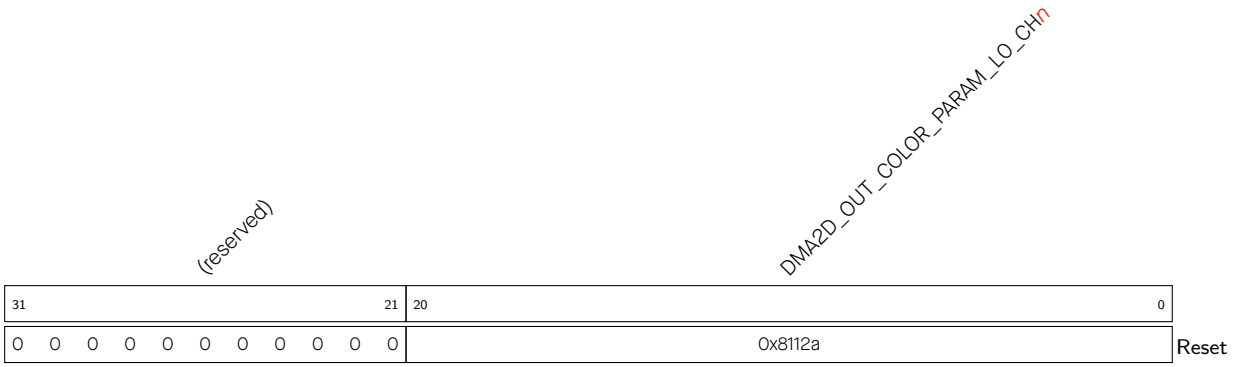
DMA2D_OUT_COLOR_PARAM_M0_CH*n* 配置发送通道 *n* 颜色空间转换第二级输入的中间字节的系数 A 和 B。(R/W)

Register 8.12. DMA2D_OUT_COLOR_PARAM3_CH*n*_REG (*n*: 0-3) (0x005C+0x100*n*)



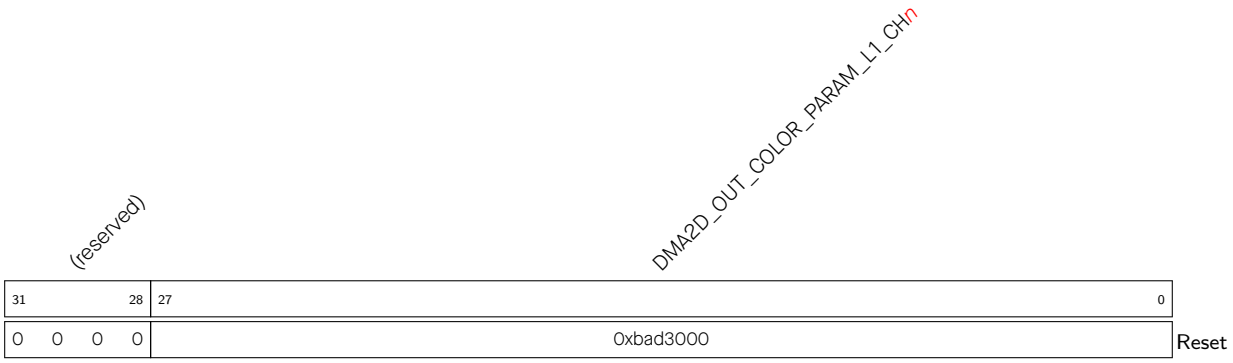
DMA2D_OUT_COLOR_PARAM_M1_CH*n* 配置发送通道 *n* 颜色空间转换第二级输入的中间字节的系数 C 和 D。(R/W)

Register 8.13. DMA2D_OUT_COLOR_PARAM4_CH*n*_REG (*n*: 0-3) (0x0060+0x100**n*)



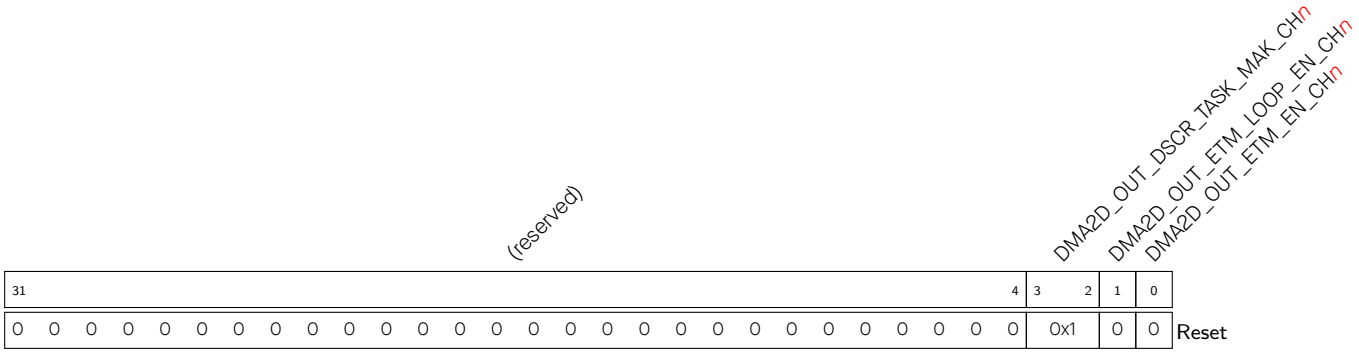
DMA2D_OUT_COLOR_PARAM_L0_CH*n* 配置发送通道 *n* 颜色空间转换第二级输入的最低有效字节的系数 A 和 B。(R/W)

Register 8.14. DMA2D_OUT_COLOR_PARAM5_CH*n*_REG (*n*: 0-3) (0x0064+0x100**n*)



DMA2D_OUT_COLOR_PARAM_L1_CH*n* 配置发送通道 *n* 颜色空间转换第二级输入的最低有效字节的系数 C 和 D。(R/W)

Register 8.15. DMA2D_OUT_ETM_CONF_CH*n*_REG (*n*: 0-3) (0x0068+0x100**n*)



DMA2D_OUT_ETM_EN_CH*n* 配置发送通道 *n* 是否开启 ETM 功能。

0：关闭

1：开启

((R/W)

DMA2D_OUT_ETM_LOOP_EN_CH*n* 配置发送通道 *n* 是否使用 ETM 任务指示下一个描述符的处理。

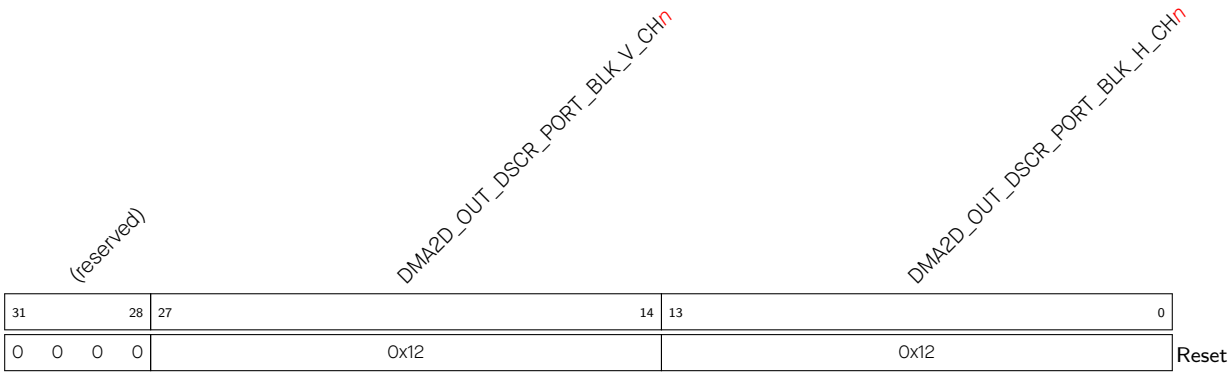
0：不使用 ETM 任务

1：使用 ETM 任务

(R/W)

DMA2D_OUT_DSCR_TASK_MAK_CH*n* 配置发送通道 *n* 可缓存任务的最大值。(R/W)

Register 8.16. DMA2D_OUT_DSCR_PORT_BLK_CH*n*_REG (*n*: 0-3) (0x006C+0x100**n*)



DMA2D_OUT_DSCR_PORT_BLK_H_CH*n* 配置 DSCR-PORT 模式下发送通道 *n* 基本单元的水平宽度。

单位：像素。(R/W)

DMA2D_OUT_DSCR_PORT_BLK_V_CH*n* 配置 DSCR-PORT 模式下发送通道 *n* 基本单元的水平高度。

单位：像素。(R/W)

Register 8.17. DMA2D_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0500+0x100**n*)

Bits	Field	Reset
31:27	(reserved)	0 0 0 0 0
26	DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n*	0
25	DMA2D_IN_CMD_DISABLE_CH*n*	0
24	DMA2D_IN_RST_CH*n*	0
23:17	(reserved)	0 0 0 0 0 0 0
16	DMA2D_IN_REORDER_EN_CH0	0
15:12	(reserved)	0 0 0 0
11	DMA2D_IN_DSCR_PORT_EN_CH*n*	0
10:9	DMA2D_IN_MACRO_BLOCK_SIZE_CH*n*	0x0
8:6	DMA2D_IN_MEM_BURST_LENGTH_CH*n*	0x0
5	reserved	0
4	DMA2D_IN_CHECK_OWNER_CH*n*	0
3	DMA2D_IN_ECC_AES_EN_CH*n*	0
2	DMA2D_INDSCR_BURST_EN_CH*n*	0
1	(reserved)	0
0	DMA2D_IN_MEM_TRANS_EN_CH*n*	0

DMA2D_IN_MEM_TRANS_EN_CH*n* 配置是否开启发送和接收通道 *n* 进行存储器到存储器的数据传输。

0：关闭

1：开启

(R/W)

DMA2D_INDSCR_BURST_EN_CH*n* 配置是否在接收通道 *n* 在访问内部存储器读取描述符时开启 INCR 突发传输。

0：关闭

1：开启

(R/W)

DMA2D_IN_ECC_AES_EN_CH*n* 配置是否开启接受通道 *n* 访问外部存储器 ECC 和 AES 加密区间的权限。

0：关闭

1：开启。此时加密区间的起始地址和响应数据需 16 字节对齐。

(R/W)

DMA2D_IN_CHECK_OWNER_CH*n* 配置是否开启接收通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

DMA2D_IN_MEM_BURST_LENGTH_CH*n* 配置接受通道 *n* 的突发传输长度。

0: 8 字节

1: 16 字节

2: 32 字节

3: 64 字节

4: 128 字节

其他值：无效

(R/W)

见下页...

Register 8.17. DMA2D_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0500+0x100**n*)

接上页...

DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* 配置接收通道 *n* 的宏块大小。仅用于 2D-MOD1 模式。

0：8 * 8 像素
1：8 * 16 像素
2：16 *16 像素
3：无宏块
(R/W)

DMA2D_IN_DSCR_PORT_EN_CH*n* 配置接收通道 *n* 是否开启 DSCR-PORT 模式。

0：关闭
1：开启
(R/W)

DMA2D_IN_REORDER_EN_CHO 配置接收通道 0 是否开启宏块重排。

0：关闭
1：开启
(R/W)

DMA2D_IN_RST_CH*n* 配置是否复位接收通道 *n*。

0：复位释放
1：复位
(R/W)

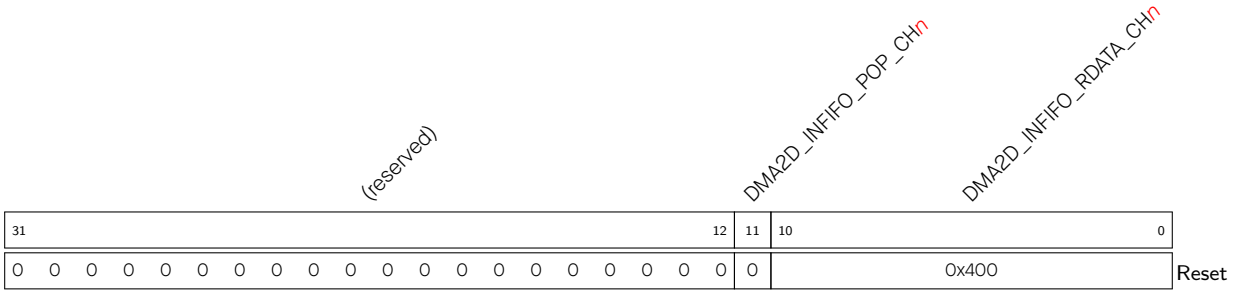
DMA2D_IN_CMD_DISABLE_CH*n* 配置接收通道 *n* 的复位指令。

0：复位释放时清零
1：复位前暂停传输
(R/W)

DMA2D_IN_ARB_WEIGHT_OPT_DIS_CH*n* 配置接收通道 *n* 是否开启权重仲裁优化。

0：开启
1：关闭
(R/W)

Register 8.18. DMA2D_IN_POP_CH*n*_REG (*n*: 0-2) (0x0518+0x100**n*)



DMA2D_INFIFO_RDATA_CH*n* 表示从 2D-DMA RX FIFO 弹出的数据。(RO)

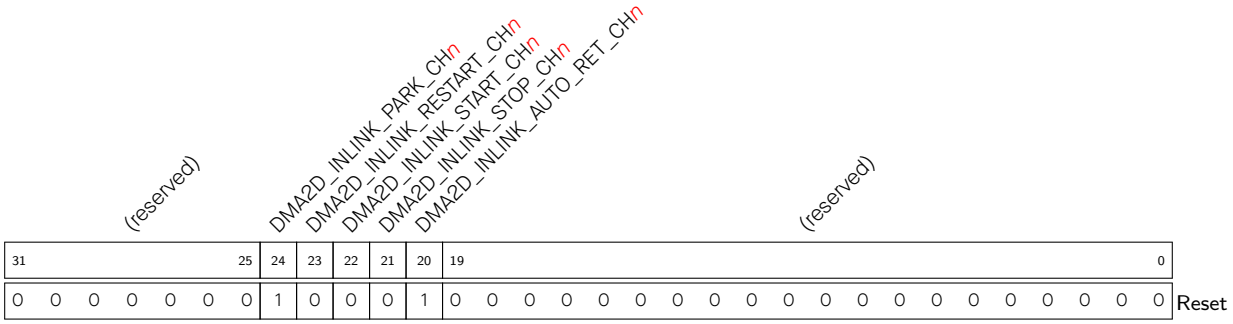
DMA2D_INFIFO_POP_CH*n* 配置是否从 2D-DMA RX FIFO 弹出数据。

0：不弹出

1：弹出

(R/W/SC)

Register 8.19. DMA2D_IN_LINK_CONF_CH*n*_REG (*n*: 0-2) (0x051C+0x100*n)



DMA2D_INLINK_AUTO_RET_CH*n* 配置回写到接收链表描述符 owner 位的值。

0：回写 0

1：回写 1

(R/W)

DMA2D_INLINK_STOP_CH*n* 配置接收通道 *n* 是否停止接收数据。

0：无效值。没有作用

1：停止

(R/W/SC)

DMA2D_INLINK_START_CH*n* 配置是否开启接收通道 *n* 进行数据传输。

0：关闭

1：开启

(R/W/SC)

DMA2D_INLINK_RESTART_CH*n* 配置是否重启接收通道 *n* 进行 2D-DMA 传输。

0：无效值。没有作用

1：重启

(R/W/SC)

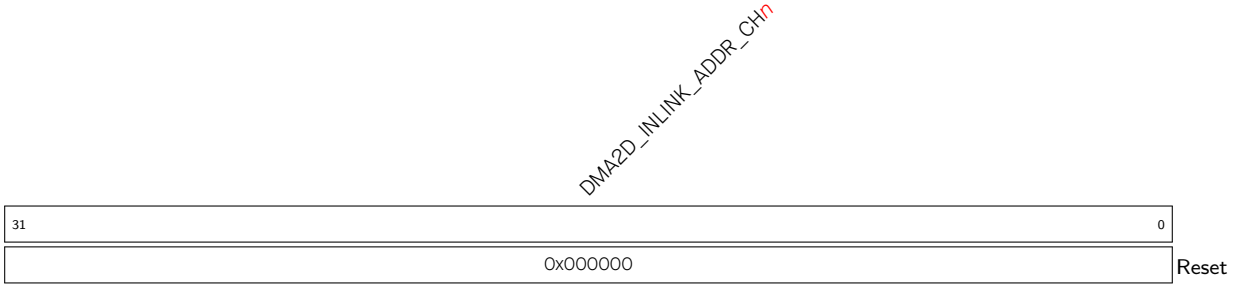
DMA2D_INLINK_PARK_CH*n* 表示接收链表描述符 FSM 的状态。

0：运行

1：空闲

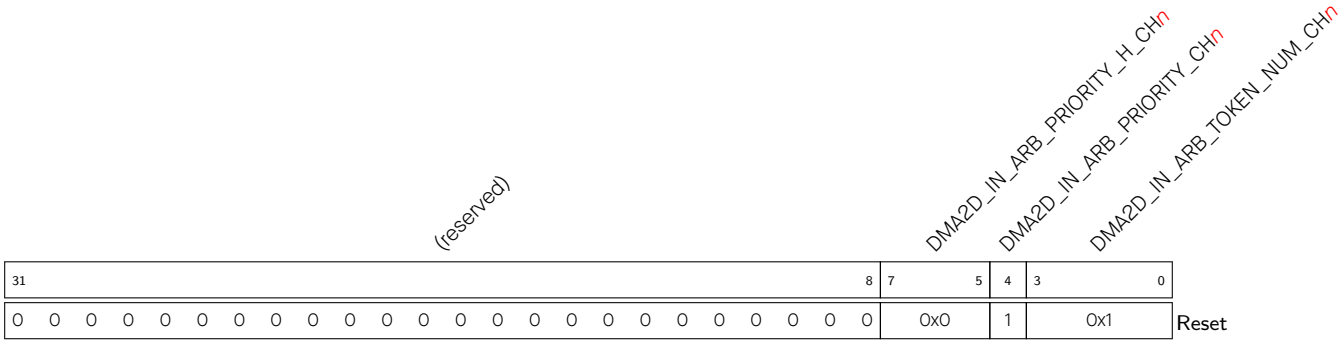
(RO)

Register 8.20. DMA2D_IN_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x0520+0x100*n)



DMA2D_INLINK_ADDR_CH*n* 表示第一个接收链表描述符的地址。(R/W)

Register 8.21. DMA2D_IN_ARB_CH*n*_REG (*n*: 0-2) (0x0540+0x100**n*)



DMA2D_IN_ARB_TOKEN_NUM_CH*n* 配置接收通道 *n* 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

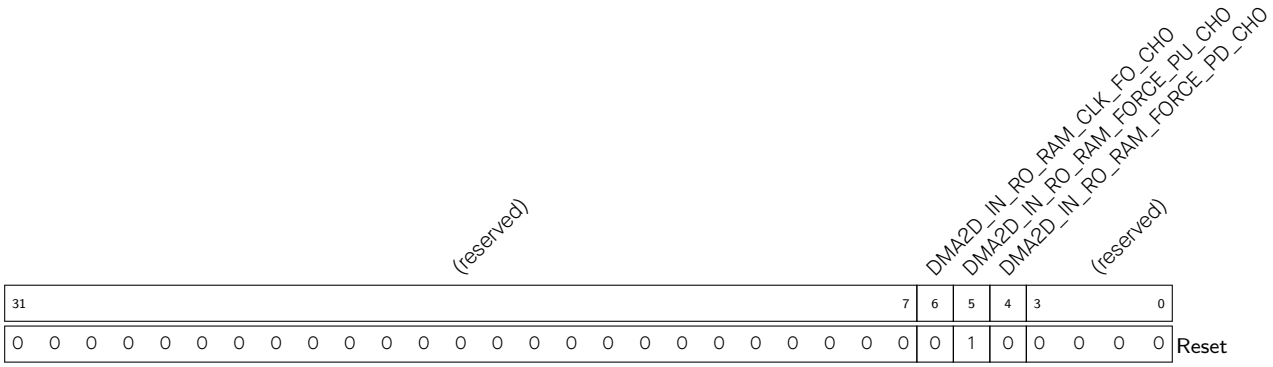
DMA2D_IN_ARB_PRIORITY_CH*n* 配置接收通道 *n* 优先级的低两位。

该字段和 [DMA2D_IN_ARB_PRIORITY_H_CH*n* 共同配置接收通道 *n* 的四位优先级数值，最低值为 0，最高值为 15。(R/W)

DMA2D_IN_ARB_PRIORITY_H_CH*n* 配置接收通道 *n* 优先级的高两位。

该字段和 DMA2D_IN_ARB_PRIORITY_CH*n* 共同配置接收通道 *n* 的四位优先级数值，最低值为 0，最高值为 15。(R/W)

Register 8.22. DMA2D_IN_RO_PD_CONF_CH0_REG (0x0548)



DMA2D_IN_RO_RAM_FORCE_PD_CH0 配置是否强制关闭接收通道 0 的宏块重排 RAM 的电源。

0：不强制关闭电源

1：强制关闭电源

(R/W)

DMA2D_IN_RO_RAM_FORCE_PU_CH0 配置是否强制开启接收通道 0 的宏块重排 RAM 的电源。

0：不强制开启电源

1：强制开启电源

(R/W)

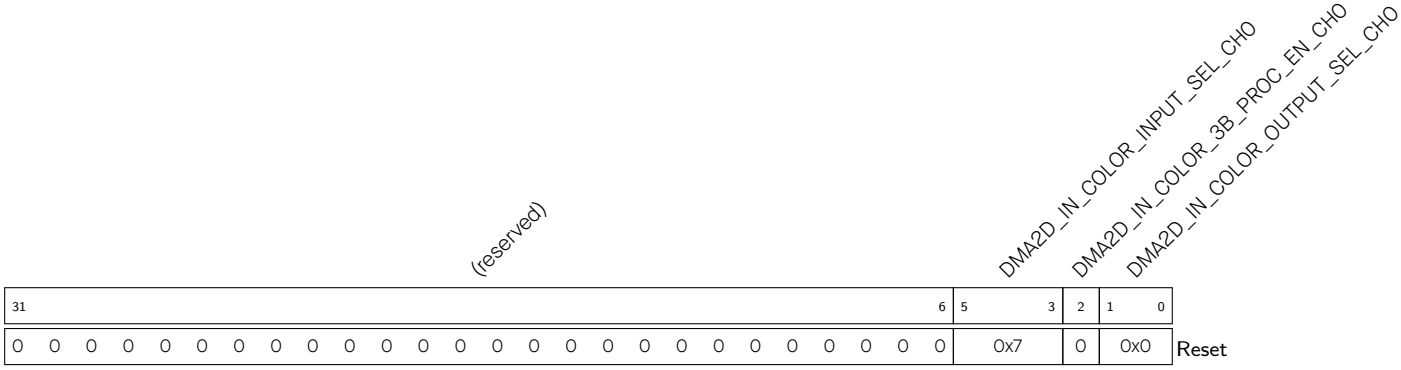
DMA2D_IN_RO_RAM_CLK_FO_CH0 配置接收通道 0 访问 2D-DMA RAM 的时钟门控。

0：使用时钟门控

1：强制开启时钟，绕过时钟门控

(R/W)

Register 8.23. DMA2D_IN_COLOR_CONVERT_CH0_REG (0x054C)



DMA2D_IN_COLOR_OUTPUT_SEL_CH0 配置接收通道 0 颜色空间转换第三级的输出。

0：RGB888 转 RGB565

1：直接输出

(R/W)

DMA2D_IN_COLOR_3B_PROC_EN_CH0 配置是否开启接收通道 0 颜色空间转换的第二级。

0：关闭

1：开启

(R/W)

DMA2D_IN_COLOR_INPUT_SEL_CH0 配置接收通道 0 颜色空间转换第一级的输出。

0：YUV422/420 转 YUV444

1：YUV422

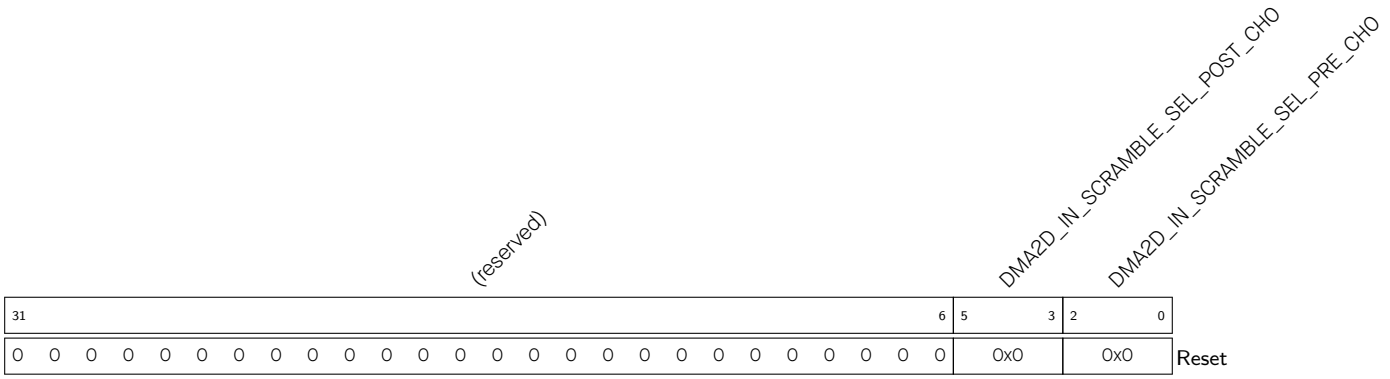
2：YUV444/420

7：关闭颜色空间转换

其他值：无效

(R/W)

Register 8.24. DMA2D_IN_SCRAMBLE_CH0_REG (0x0550)



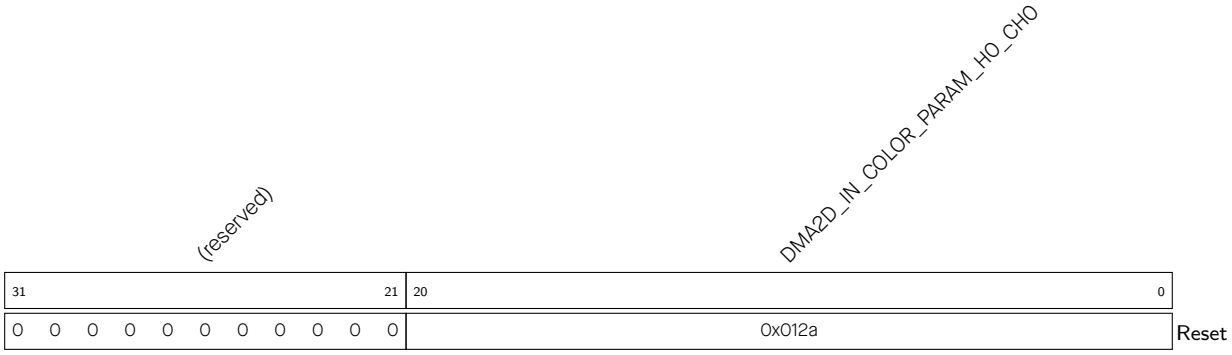
DMA2D_IN_SCRAMBLE_SEL_PRE_CH0 配置接收通道 0 颜色空间转换前的字节乱序处理。

0：BYTE2-1-0
1：BYTE2-0-1
2：BYTE1-0-2
3：BYTE1-2-0
4：BYTE0-2-1
5：BYTE0-1-2
其他值：无效
(R/W)

DMA2D_IN_SCRAMBLE_SEL_POST_CH0 配置接收通道 0 颜色空间转换后的字节乱序处理。

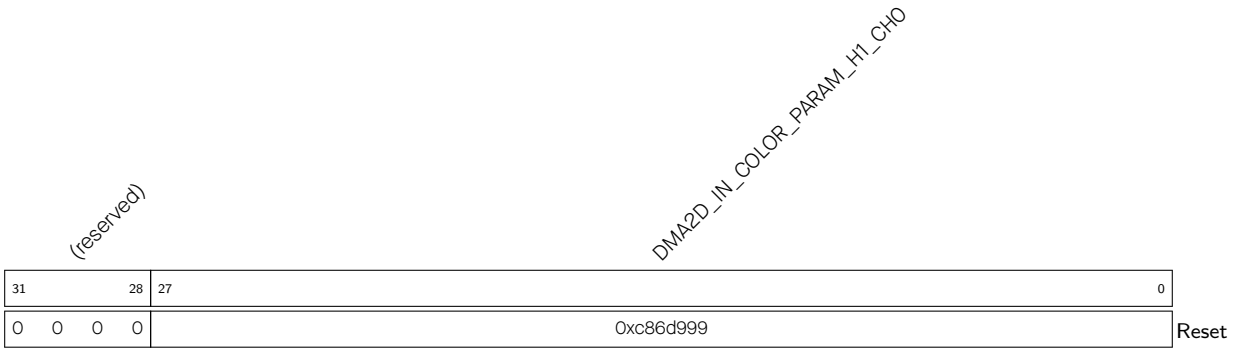
0：BYTE2-1-0
1：BYTE2-0-1
2：BYTE1-0-2
3：BYTE1-2-0
4：BYTE0-2-1
5：BYTE0-1-2
其他值：无效
(R/W)

Register 8.25. DMA2D_IN_COLOR_PARAM0_CH0_REG (0x0554)



DMA2D_IN_COLOR_PARAM_H0_CH0 配置接收通道 0 颜色空间转换第二级输入的最高有效字节的系数 A 和 B。(R/W)

Register 8.26. DMA2D_IN_COLOR_PARAM1_CH0_REG (0x0558)



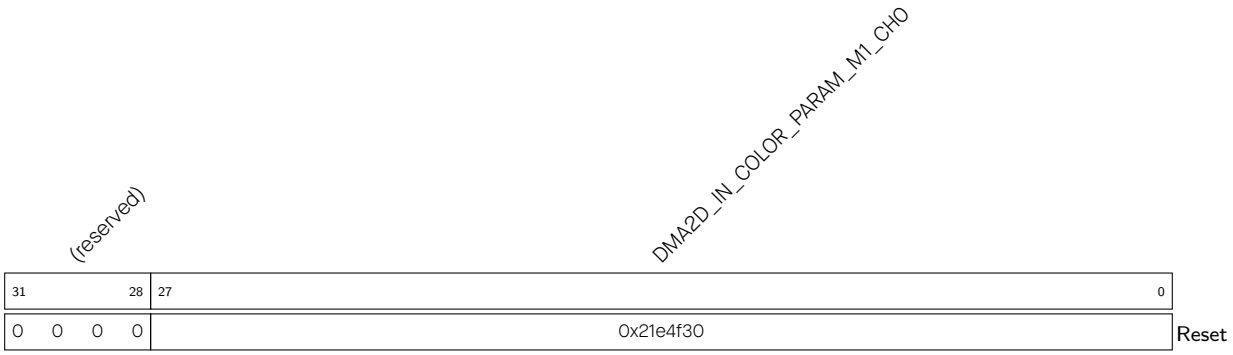
DMA2D_IN_COLOR_PARAM_H1_CH0 配置接收通道 0 颜色空间转换第二级输入的最高有效字节的系数 C 和 D。(R/W)

Register 8.27. DMA2D_IN_COLOR_PARAM2_CH0_REG (0x055C)



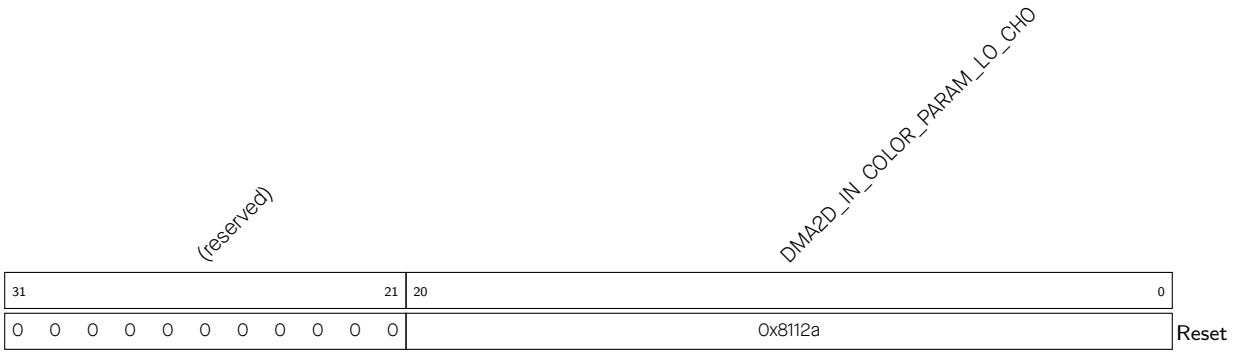
DMA2D_IN_COLOR_PARAM_M0_CH0 配置接收通道 0 颜色空间转换第二级输入的中间字节的系数 A 和 B。(R/W)

Register 8.28. DMA2D_IN_COLOR_PARAM3_CH0_REG (0x0560)



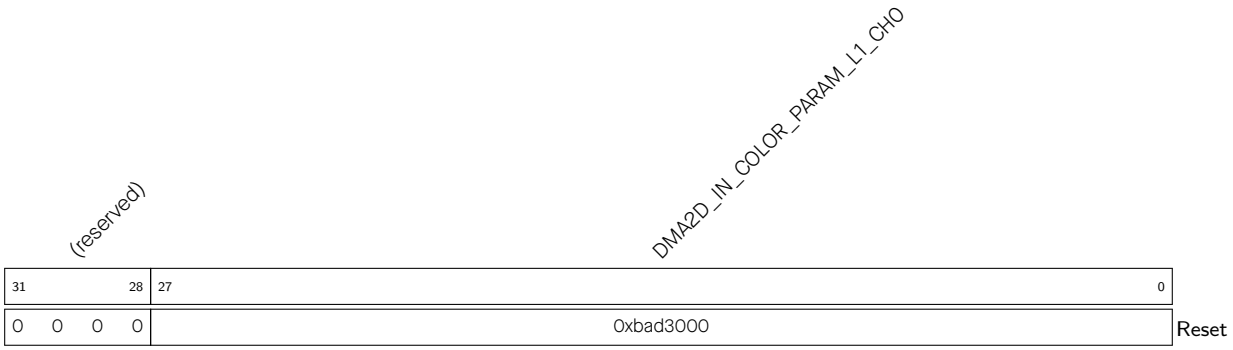
DMA2D_IN_COLOR_PARAM_M1_CH0 配置接收通道 0 颜色空间转换第二级输入的中间字节的系数 C 和 D。(R/W)

Register 8.29. DMA2D_IN_COLOR_PARAM4_CH0_REG (0x0564)



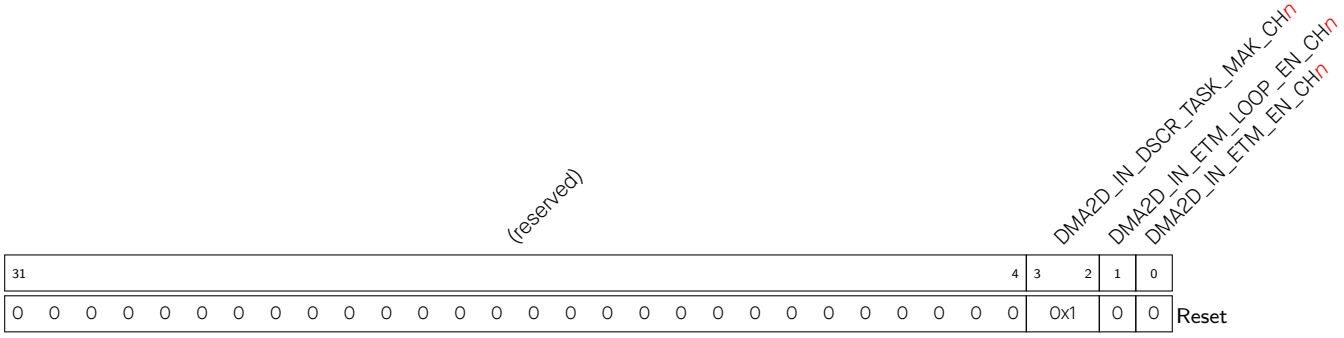
DMA2D_IN_COLOR_PARAM_L0_CH0 配置接收通道 0 颜色空间转换第二级输入的最低有效字节的系数 A 和 B。(R/W)

Register 8.30. DMA2D_IN_COLOR_PARAM5_CH0_REG (0x0568)



DMA2D_IN_COLOR_PARAM_L1_CH0 配置接收通道 0 颜色空间转换第二级输入的最低有效字节的系数 C 和 D。(R/W)

Register 8.31. DMA2D_IN_ETM_CONF_CH*n*_REG (*n*: 0-2) (0x056C+0x100**n*)



DMA2D_IN_ETM_EN_CH*n* 配置接收通道 *n* 是否开启 ETM 功能。

0：关闭

1：开启

(R/W)

DMA2D_IN_ETM_LOOP_EN_CH*n* 配置接收通道 *n* 是否使用 ETM 任务指示下一个描述符的处理。

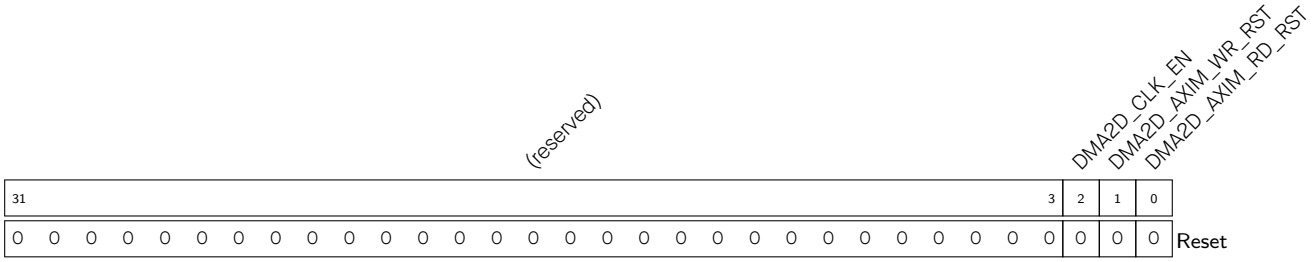
0：不使用 ETM 任务

1：使用 ETM 任务

(R/W)

DMA2D_IN_DSCR_TASK_MAK_CH*n* 配置接收通道 *n* 可缓存任务的最大值。(R/W)

Register 8.32. DMA2D_RST_CONF_REG (0x0A04)



DMA2D_AXIM_RD_RST 配置是否复位 AXI 主机读数据 FIFO。

0：无效值

1：复位

(R/W)

DMA2D_AXIM_WR_RST 配置是否复位 AXI 主机写数据 FIFO。

0：无效值

1：复位

(R/W)

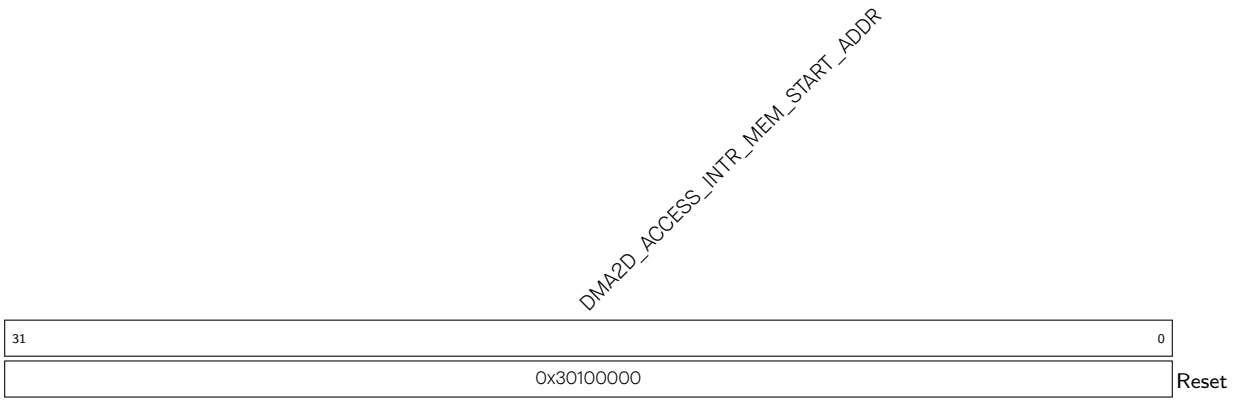
DMA2D_CLK_EN 配置寄存器时钟门控。

0：仅在应用程序写入寄存器时支持时钟。

1：始终强制打开寄存器的时钟。

(R/W)

Register 8.33. DMA2D_INTR_MEM_START_ADDR_REG (0x0A08)



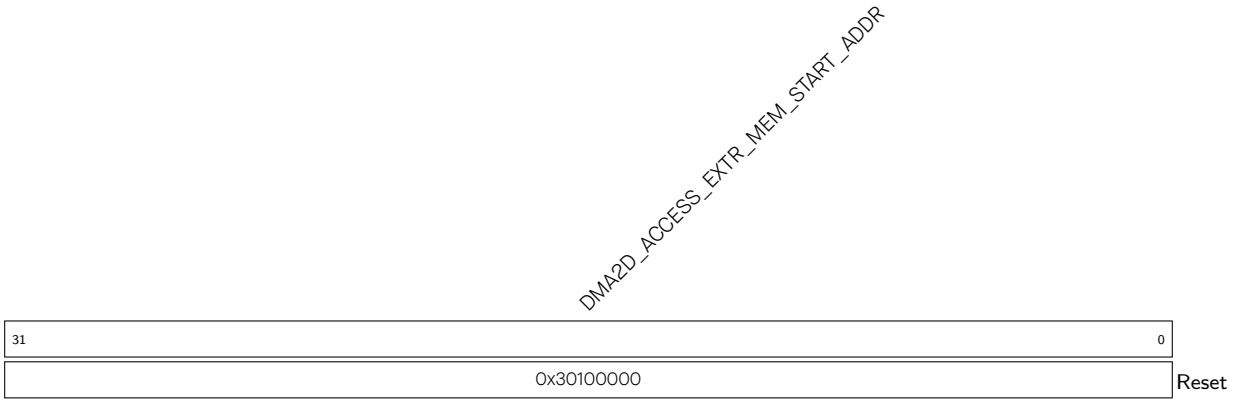
DMA2D_ACCESS_INTR_MEM_START_ADDR 配置可访问内部存储器地址空间的起始地址。(R/W)

Register 8.34. DMA2D_INTR_MEM_END_ADDR_REG (0x0A0C)



DMA2D_ACCESS_INTR_MEM_END_ADDR 配置可访问内部存储器地址空间的结束地址。访问超过区间的地址会造成描述符错误。(R/W)

Register 8.35. DMA2D_EXTR_MEM_START_ADDR_REG (0x0A10)



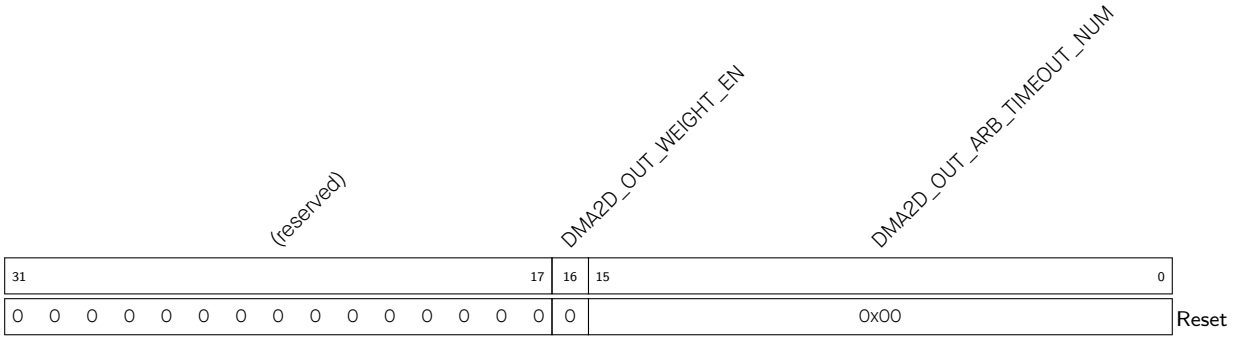
DMA2D_ACCESS_EXTR_MEM_START_ADDR 配置可访问外部存储器地址空间的起始地址。(R/W)

Register 8.36. DMA2D_EXTR_MEM_END_ADDR_REG (0x0A14)



DMA2D_ACCESS_EXTR_MEM_END_ADDR 配置可访问外部存储器地址空间的结束地址。访问超过区间的地址会造成描述符错误。(R/W)

Register 8.37. DMA2D_OUT_ARB_CONFIG_REG (0x0A18)



DMA2D_OUT_ARB_TIMEOUT_NUM 配置 TX 方向时间片。单位：AXI 总线时钟周期。(R/W)

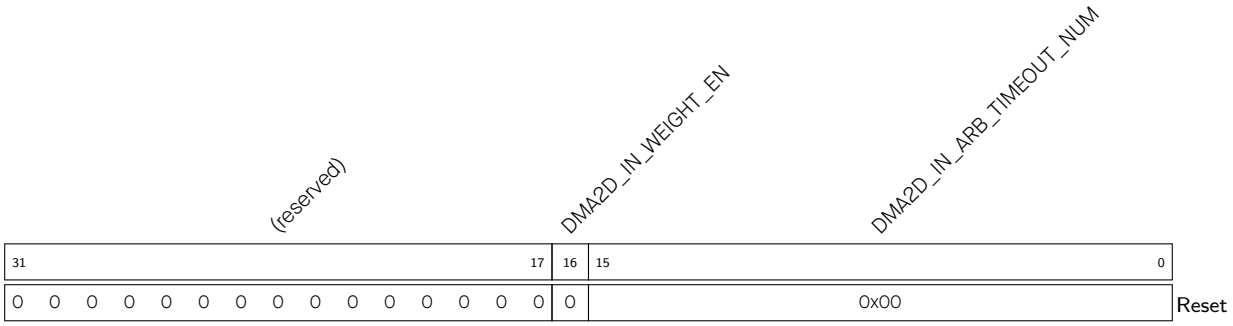
DMA2D_OUT_WEIGHT_EN 配置开启 TX 方向权重仲裁。

0：关闭

1：开启

(R/W)

Register 8.38. DMA2D_IN_ARB_CONFIG_REG (0x0A1C)



DMA2D_IN_ARB_TIMEOUT_NUM 配置 RX 方向时间片。单位：AXI 总线时钟周期。(R/W)

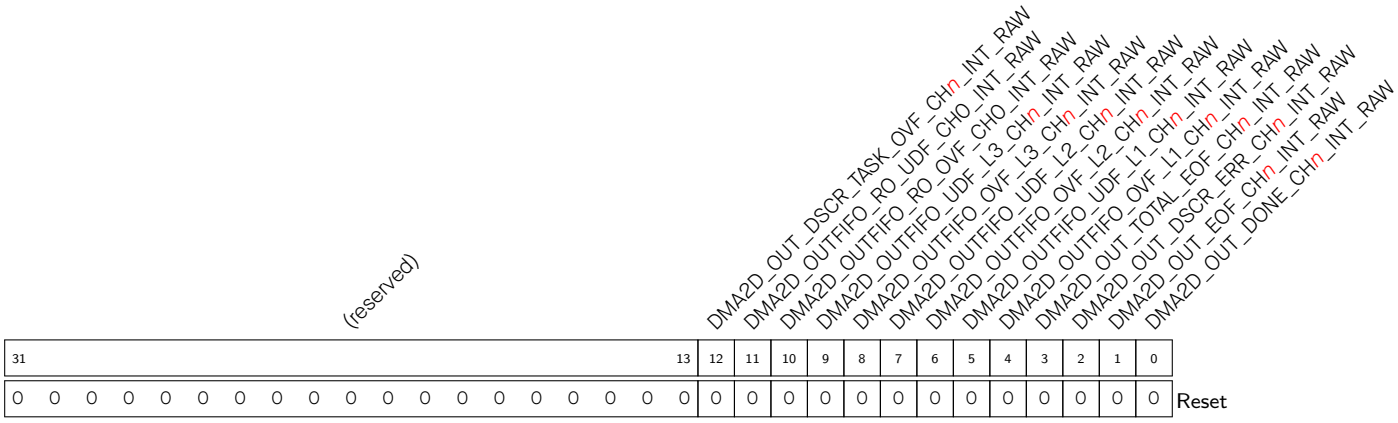
DMA2D_IN_WEIGHT_EN 配置开启 RX 方向权重仲裁。

0：关闭

1：开启

(R/W)

Register 8.39. DMA2D_OUT_INT_RAW_CH*n*_REG (*n*: 0-3) (0x0004+0x100*n*)



DMA2D_OUT_DONE_CH*n*_INT_RAW DMA2D_OUT_DONE_CH*n*_INT 的原始中断状态。(R/ WTC/SS)

DMA2D_OUT_EOF_CH*n*_INT_RAW DMA2D_OUT_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_RAW DMA2D_OUT_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_RAW DMA2D_OUT_TOTAL_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_RAW DMA2D_OUTFIFO_OVF_L1_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_RAW DMA2D_OUTFIFO_UDF_L1_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_RAW DMA2D_OUTFIFO_OVF_L2_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_RAW DMA2D_OUTFIFO_UDF_L2_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_RAW DMA2D_OUTFIFO_OVF_L3_CH*n*_INT 的原始中断状态。(R/WTC/SS)

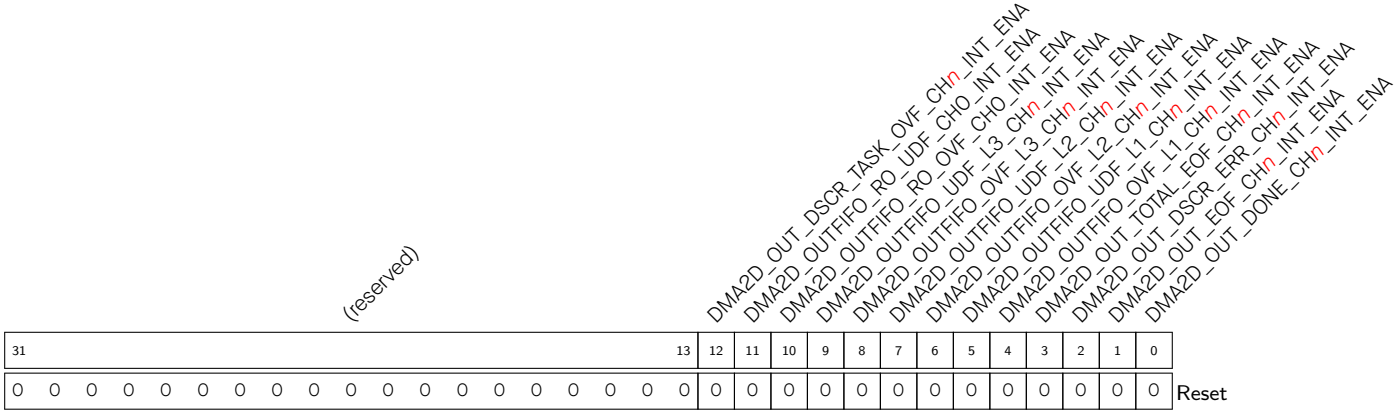
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_RAW DMA2D_OUTFIFO_UDF_L3_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_RAW DMA2D_OUTFIFO_RO_OVF_CHO_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_RAW DMA2D_OUTFIFO_RO_UDF_CHO_INT 的原始中断状态。(R/WTC/SS)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_RAW DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 8.40. DMA2D_OUT_INT_ENA_CH*n*_REG (*n*: 0-3) (0x0008+0x100*n*)



DMA2D_OUT_DONE_CH*n*_INT_ENA 写 1 使能 DMA2D_OUT_DONE_CH*n*_INT。(R/W)

DMA2D_OUT_EOF_CH*n*_INT_ENA 写 1 使能 DMA2D_OUT_EOF_CH*n*_INT。(R/W)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 DMA2D_OUT_DSCR_ERR_CH*n*_INT。(R/W)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ENA 写 1 使能 DMA2D_OUT_TOTAL_EOF_CH*n*_INT。(R/W)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_ENA 写 1 使能 DMA2D_OUTFIFO_OVF_L1_CH*n*_INT。(R/W)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_ENA 写 1 使能 DMA2D_OUTFIFO_UDF_L1_CH*n*_INT。(R/W)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_ENA 写 1 使能 DMA2D_OUTFIFO_OVF_L2_CH*n*_INT。(R/W)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_ENA 写 1 使能 DMA2D_OUTFIFO_UDF_L2_CH*n*_INT。(R/W)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_ENA 写 1 使能 DMA2D_OUTFIFO_OVF_L3_CH*n*_INT。(R/W)

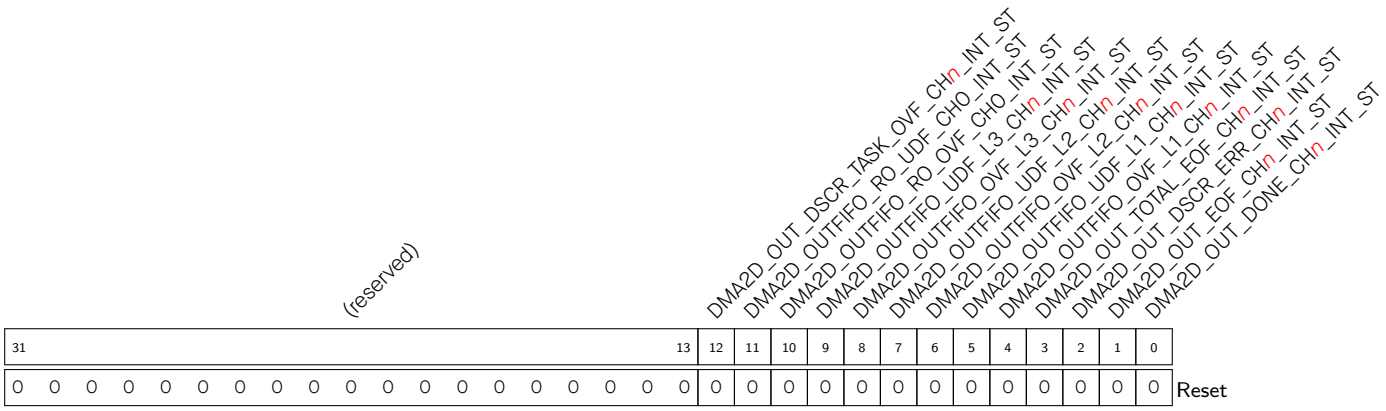
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ENA 写 1 使能 DMA2D_OUTFIFO_UDF_L3_CH*n*_INT。(R/W)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_ENA 写 1 使能 DMA2D_OUTFIFO_RO_OVF_CHO_INT。(R/W)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_ENA 写 1 使能 DMA2D_OUTFIFO_RO_UDF_CHO_INT。(R/W)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_ENA 写 1 使能 DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT。(R/W)

Register 8.41. DMA2D_OUT_INT_ST_CH*n*_REG (*n*: 0-3) (0x000C+0x100*n*)



DMA2D_OUT_DONE_CH*n*_INT_ST DMA2D_OUT_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUT_EOF_CH*n*_INT_ST DMA2D_OUT_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_ST DMA2D_OUT_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_ST DMA2D_OUT_TOTAL_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_ST DMA2D_OUTFIFO_OVF_L1_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_ST DMA2D_OUTFIFO_UDF_L1_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_ST DMA2D_OUTFIFO_OVF_L2_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_ST DMA2D_OUTFIFO_UDF_L2_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_ST DMA2D_OUTFIFO_OVF_L3_CH*n*_INT 的屏蔽中断状态。(RO)

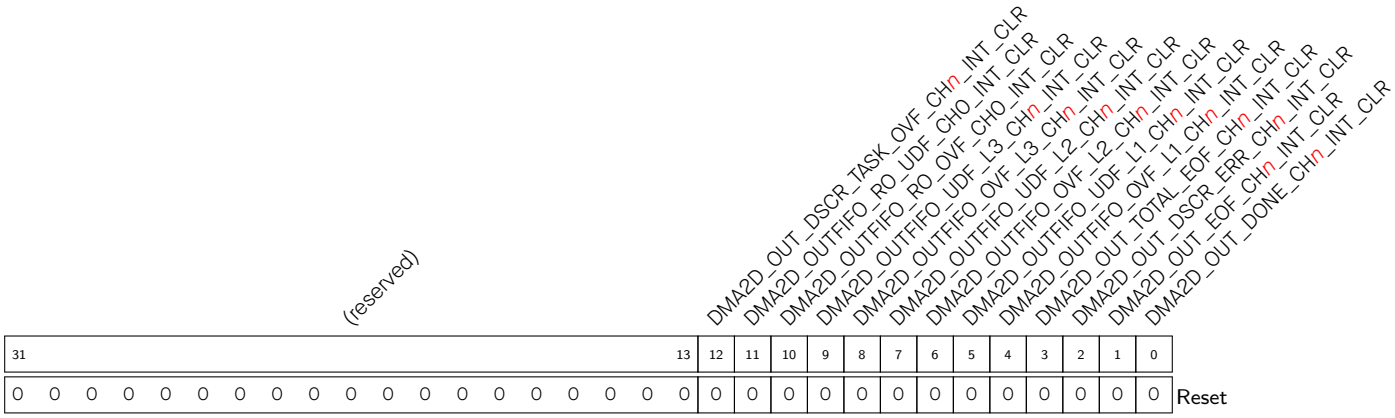
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_ST DMA2D_OUTFIFO_UDF_L3_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_OUTFIFO_RO_OVF_CH0_INT_ST DMA2D_OUTFIFO_RO_OVF_CH0_INT 的屏蔽中断状态。(RO)

DMA2D_OUTFIFO_RO_UDF_CH0_INT_ST DMA2D_OUTFIFO_RO_UDF_CH0_INT 的屏蔽中断状态。(RO)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_ST DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

Register 8.42. DMA2D_OUT_INT_CLR_CH*n*_REG (*n*: 0-3) (0x0010+0x100*n)



DMA2D_OUT_DONE_CH*n*_INT_CLR 写 1 清除 DMA2D_OUT_DONE_CH*n*_INT。(WT)

DMA2D_OUT_EOF_CH*n*_INT_CLR 写 1 清除 DMA2D_OUT_EOF_CH*n*_INT。(WT)

DMA2D_OUT_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 DMA2D_OUT_DSCR_ERR_CH*n*_INT。(WT)

DMA2D_OUT_TOTAL_EOF_CH*n*_INT_CLR 写 1 清除 DMA2D_OUT_TOTAL_EOF_CH*n*_INT。(WT)

DMA2D_OUTFIFO_OVF_L1_CH*n*_INT_CLR 写 1 清除 DMA2D_OUTFIFO_OVF_L1_CH*n*_INT。(WT)

DMA2D_OUTFIFO_UDF_L1_CH*n*_INT_CLR 写 1 清除 DMA2D_OUTFIFO_UDF_L1_CH*n*_INT。(WT)

DMA2D_OUTFIFO_OVF_L2_CH*n*_INT_CLR 写 1 清除 DMA2D_OUTFIFO_OVF_L2_CH*n*_INT。(WT)

DMA2D_OUTFIFO_UDF_L2_CH*n*_INT_CLR 写 1 清除 DMA2D_OUTFIFO_UDF_L2_CH*n*_INT。(WT)

DMA2D_OUTFIFO_OVF_L3_CH*n*_INT_CLR 写 1 清除 DMA2D_OUTFIFO_OVF_L3_CH*n*_INT。(WT)

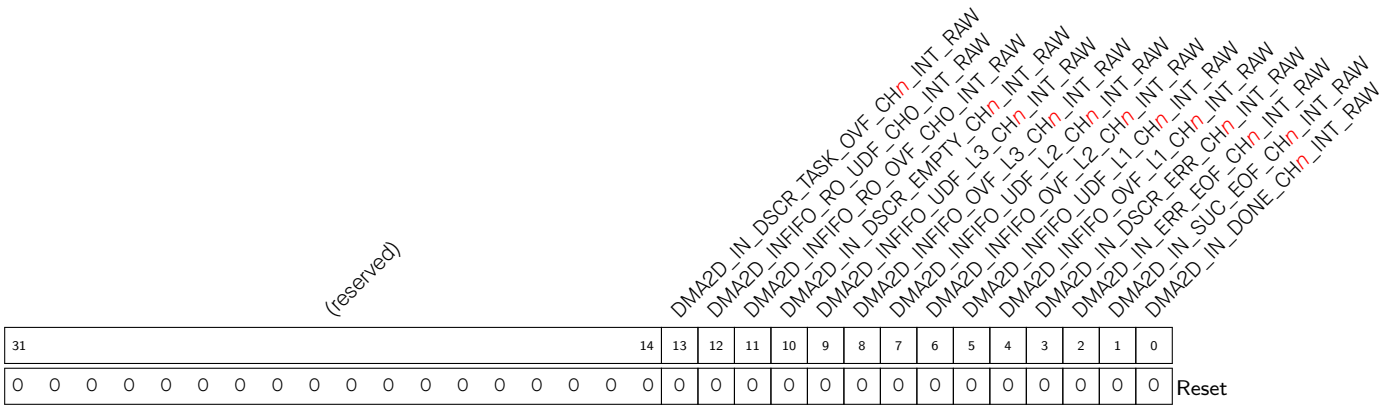
DMA2D_OUTFIFO_UDF_L3_CH*n*_INT_CLR 写 1 清除 DMA2D_OUTFIFO_UDF_L3_CH*n*_INT。(WT)

DMA2D_OUTFIFO_RO_OVF_CHO_INT_CLR 写 1 清除 DMA2D_OUTFIFO_RO_OVF_CHO_INT。(WT)

DMA2D_OUTFIFO_RO_UDF_CHO_INT_CLR 写 1 清除 DMA2D_OUTFIFO_RO_UDF_CHO_INT。(WT)

DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT_CLR 写 1 清除 DMA2D_OUT_DSCR_TASK_OVF_CH*n*_INT。(WT)

Register 8.43. DMA2D_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0504+0x100**n*)



DMA2D_IN_DONE_CH*n*_INT_RAW DMA2D_IN_DONE_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_IN_SUC_EOF_CH*n*_INT_RAW DMA2D_IN_SUC_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_IN_ERR_EOF_CH*n*_INT_RAW DMA2D_IN_ERR_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_IN_DSCR_ERR_CH*n*_INT_RAW DMA2D_IN_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_RAW DMA2D_INFIFO_OVF_L1_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_RAW DMA2D_INFIFO_UDF_L1_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_RAW DMA2D_INFIFO_OVF_L2_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_RAW DMA2D_INFIFO_UDF_L2_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_RAW DMA2D_INFIFO_OVF_L3_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_RAW DMA2D_INFIFO_UDF_L3_CH*n*_INT 的原始中断状态。(R/WTC/SS)

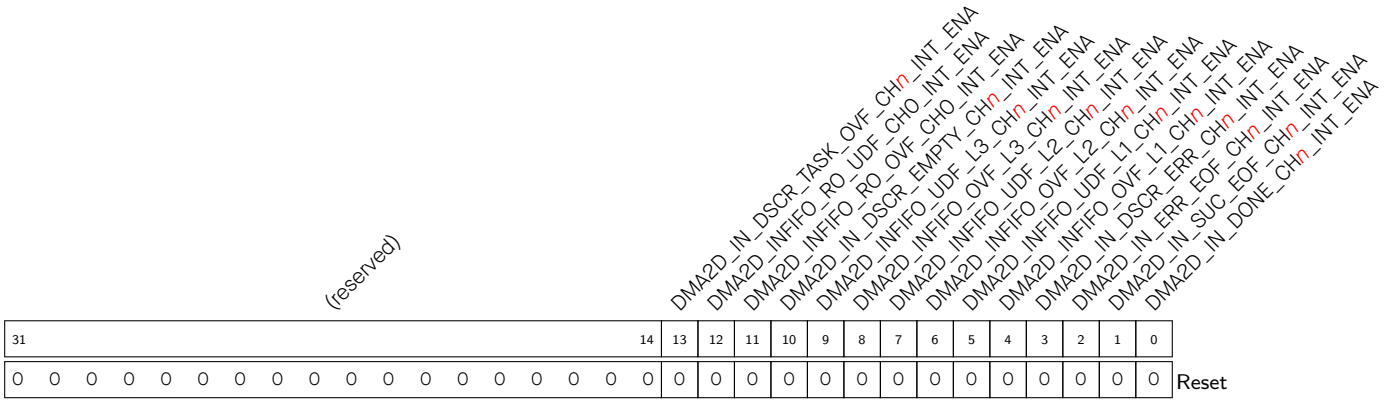
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_RAW DMA2D_IN_DSCR_EMPTY_CH*n*_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_RO_OVF_CHO_INT_RAW DMA2D_INFIFO_RO_OVF_CHO_INT 的原始中断状态。(R/WTC/SS)

DMA2D_INFIFO_RO_UDF_CHO_INT_RAW DMA2D_INFIFO_RO_UDF_CHO_INT 的原始中断状态。(R/WTC/SS)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_RAW DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 8.44. DMA2D_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0508+0x100**n*)



DMA2D_IN_DONE_CH*n*_INT_ENA 写 1 使能 DMA2D_IN_DONE_CH*n*_INT。(R/W)

DMA2D_IN_SUC_EOF_CH*n*_INT_ENA 写 1 使能 DMA2D_IN_SUC_EOF_CH*n*_INT。(R/W)

DMA2D_IN_ERR_EOF_CH*n*_INT_ENA 写 1 使能 DMA2D_IN_ERR_EOF_CH*n*_INT。(R/W)

DMA2D_IN_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 DMA2D_IN_DSCR_ERR_CH*n*_INT。(R/W)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_ENA 写 1 使能 DMA2D_INFIFO_OVF_L1_CH*n*_INT。(R/W)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_ENA 写 1 使能 DMA2D_INFIFO_UDF_L1_CH*n*_INT。(R/W)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_ENA 写 1 使能 DMA2D_INFIFO_OVF_L2_CH*n*_INT。(R/W)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_ENA 写 1 使能 DMA2D_INFIFO_UDF_L2_CH*n*_INT。(R/W)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_ENA 写 1 使能 DMA2D_INFIFO_OVF_L3_CH*n*_INT。(R/W)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_ENA 写 1 使能 DMA2D_INFIFO_UDF_L3_CH*n*_INT。(R/W)

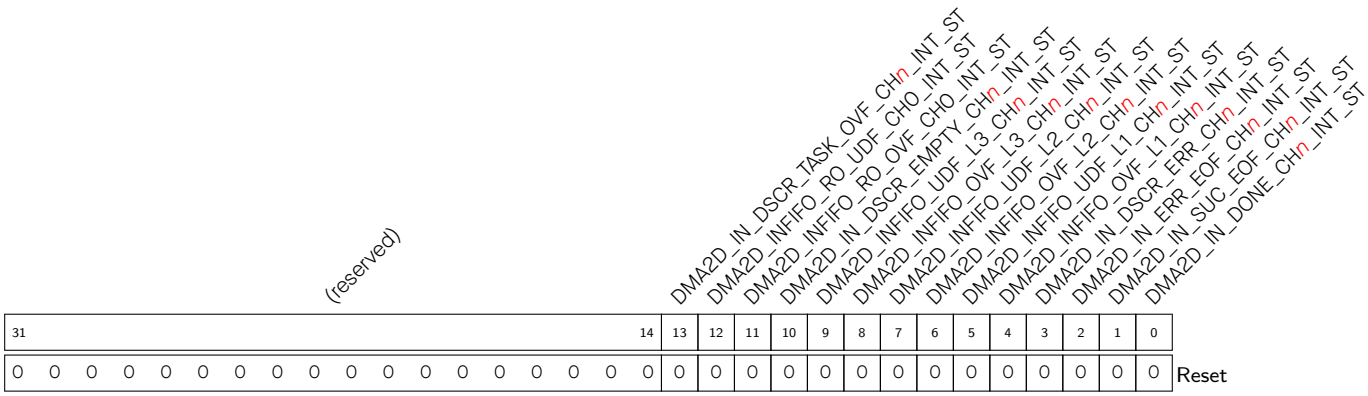
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ENA 写 1 使能 DMA2D_IN_DSCR_EMPTY_CH*n*_INT。(R/W)

DMA2D_INFIFO_RO_OVF_CHO_INT_ENA 写 1 使能 DMA2D_INFIFO_RO_OVF_CHO_INT。(R/W)

DMA2D_INFIFO_RO_UDF_CHO_INT_ENA 写 1 使能 DMA2D_INFIFO_RO_UDF_CHO_INT。(R/W)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_ENA 写 1 使能 DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT。(R/W)

Register 8.45. DMA2D_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x050C+0x100**n*)



DMA2D_IN_DONE_CH*n*_INT_ST DMA2D_IN_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_IN_SUC_EOF_CH*n*_INT_ST DMA2D_IN_SUC_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_IN_ERR_EOF_CH*n*_INT_ST DMA2D_IN_ERR_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_IN_DSCR_ERR_CH*n*_INT_ST DMA2D_IN_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_ST the DMA2D_INFIFO_OVF_L1_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_ST the DMA2D_INFIFO_UDF_L1_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_ST the DMA2D_INFIFO_OVF_L2_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_ST the DMA2D_INFIFO_UDF_L2_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_ST the DMA2D_INFIFO_OVF_L3_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_ST the DMA2D_INFIFO_UDF_L3_CH*n*_INT 的屏蔽中断状态。(RO)

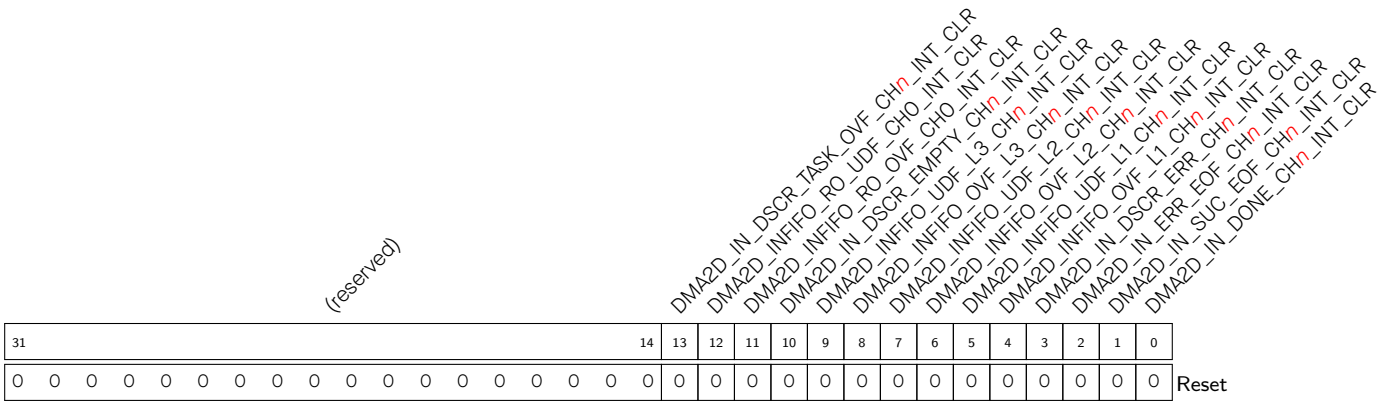
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_ST DMA2D_IN_DSCR_EMPTY_CH*n*_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_RO_OVF_CH0_INT_ST the DMA2D_INFIFO_RO_OVF_CH0_INT 的屏蔽中断状态。(RO)

DMA2D_INFIFO_RO_UDF_CH0_INT_ST the DMA2D_INFIFO_RO_UDF_CH0_INT 的屏蔽中断状态。(RO)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_ST the DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

Register 8.46. DMA2D_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x0510+0x100**n*)



DMA2D_IN_DONE_CH*n*_INT_CLR 写 1 清除 DMA2D_IN_DONE_CH*n*_INT。(WT)

DMA2D_IN_SUC_EOF_CH*n*_INT_CLR 写 1 清除 DMA2D_IN_SUC_EOF_CH*n*_INT。(WT)

DMA2D_IN_ERR_EOF_CH*n*_INT_CLR 写 1 清除 DMA2D_IN_ERR_EOF_CH*n*_INT。(WT)

DMA2D_IN_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 DMA2D_IN_DSCR_ERR_CH*n*_INT。(WT)

DMA2D_INFIFO_OVF_L1_CH*n*_INT_CLR 写 1 清除 DMA2D_INFIFO_OVF_L1_CH*n*_INT。(WT)

DMA2D_INFIFO_UDF_L1_CH*n*_INT_CLR 写 1 清除 DMA2D_INFIFO_UDF_L1_CH*n*_INT。(WT)

DMA2D_INFIFO_OVF_L2_CH*n*_INT_CLR 写 1 清除 DMA2D_INFIFO_OVF_L2_CH*n*_INT。(WT)

DMA2D_INFIFO_UDF_L2_CH*n*_INT_CLR 写 1 清除 DMA2D_INFIFO_UDF_L2_CH*n*_INT。(WT)

DMA2D_INFIFO_OVF_L3_CH*n*_INT_CLR 写 1 清除 DMA2D_INFIFO_OVF_L3_CH*n*_INT。(WT)

DMA2D_INFIFO_UDF_L3_CH*n*_INT_CLR 写 1 清除 DMA2D_INFIFO_UDF_L3_CH*n*_INT。(WT)

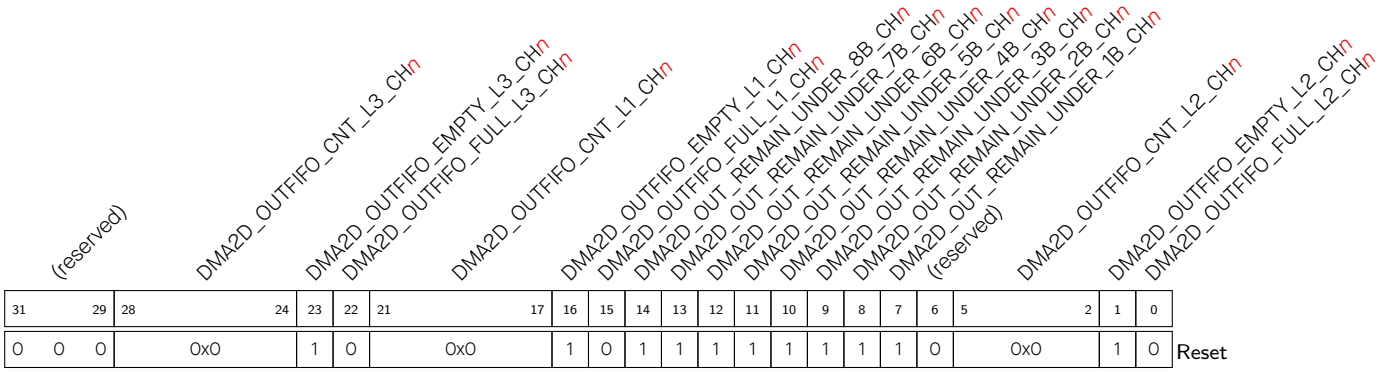
DMA2D_IN_DSCR_EMPTY_CH*n*_INT_CLR 写 1 清除 DMA2D_IN_DSCR_EMPTY_CH*n*_INT。(WT)

DMA2D_INFIFO_RO_OVF_CHO_INT_CLR 写 1 清除 DMA2D_INFIFO_RO_OVF_CHO_INT。(WT)

DMA2D_INFIFO_RO_UDF_CHO_INT_CLR 写 1 清除 DMA2D_INFIFO_RO_UDF_CHO_INT。(WT)

DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT_CLR 写 1 清除 DMA2D_IN_DSCR_TASK_OVF_CH*n*_INT。(WT)

Register 8.47. DMA2D_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-3) (0x0014+0x100**n*)



DMA2D_OUTFIFO_FULL_L2_CH*n* 表示发送通道 *n* 的 L2 TX FIFO 是否为满。Represents whether the L2 TX FIFO for TX channel *n* 是否为满。

0：未满

1：已满

(RO)

DMA2D_OUTFIFO_EMPTY_L2_CH*n* 表示发送通道 *n* 的 L2 TX FIFO 是否为空。

0：未空

1：已空

(RO)

DMA2D_OUTFIFO_CNT_L2_CH*n* 表示发送通道 *n* L2 TX FIFO 中的字节数。(RO)

DMA2D_OUT_REMAIN_UNDER_1B_CH*n* 保留。(RO)

DMA2D_OUT_REMAIN_UNDER_2B_CH*n* 保留。(RO)

DMA2D_OUT_REMAIN_UNDER_3B_CH*n* 保留。(RO)

DMA2D_OUT_REMAIN_UNDER_4B_CH*n* 保留。(RO)

DMA2D_OUT_REMAIN_UNDER_5B_CH*n* 保留。(RO)

DMA2D_OUT_REMAIN_UNDER_6B_CH*n* 保留。(RO)

DMA2D_OUT_REMAIN_UNDER_7B_CH*n* 保留。(RO)

DMA2D_OUT_REMAIN_UNDER_8B_CH*n* 保留。(RO)

DMA2D_OUTFIFO_FULL_L1_CH*n* 表示发送通道 *n* 的 L1 TX FIFO 是否为满。

0：未满

1：已满

(RO)

DMA2D_OUTFIFO_EMPTY_L1_CH*n* 表示发送通道 *n* 的 L1 TX FIFO 是否为空。

0：未空

1：已空

(RO)

DMA2D_OUTFIFO_CNT_L1_CH*n* 表示发送通道 *n* L1 TX FIFO 中的字节数。(RO)

见下页...

Register 8.47. DMA2D_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-3) (0x0014+0x100**n*)

接上页...

DMA2D_OUTFIFO_FULL_L3_CH*n* 表示发送通道 *n* 的 L3 TX FIFO 是否为满。

0：未满

1：已满

(RO)

DMA2D_OUTFIFO_EMPTY_L3_CH*n* 表示发送通道 *n* 的 L3 TX FIFO 是否为空。

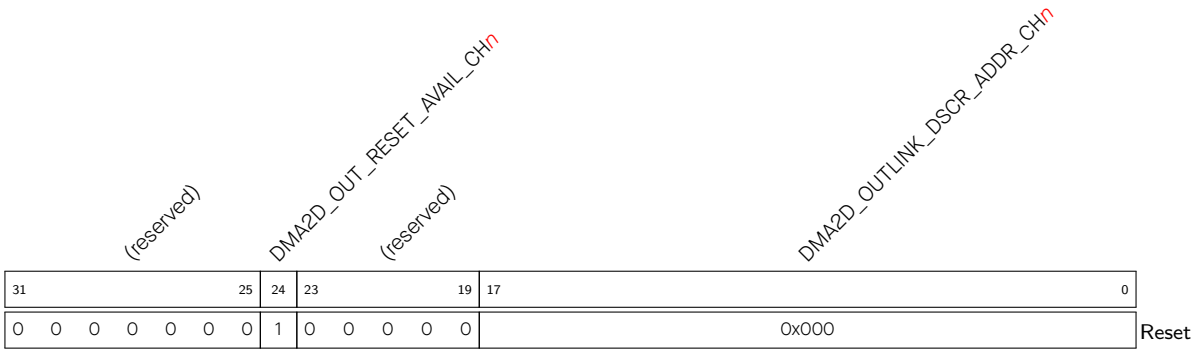
0：未空

1：已空

(RO)

DMA2D_OUTFIFO_CNT_L3_CH*n* 表示发送通道 *n* L3 TX FIFO 中的字节数。(RO)

Register 8.48. DMA2D_OUT_STATE_CH*n*_REG (*n*: 0-3) (0x0024+0x100**n*)



DMA2D_OUTLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的发送接收链表描述符所在地址的低 18 位。如果当前处理的发送链表描述符是最后一个描述符，则该字段表示当前处理的发送链表描述符地址。(RO)

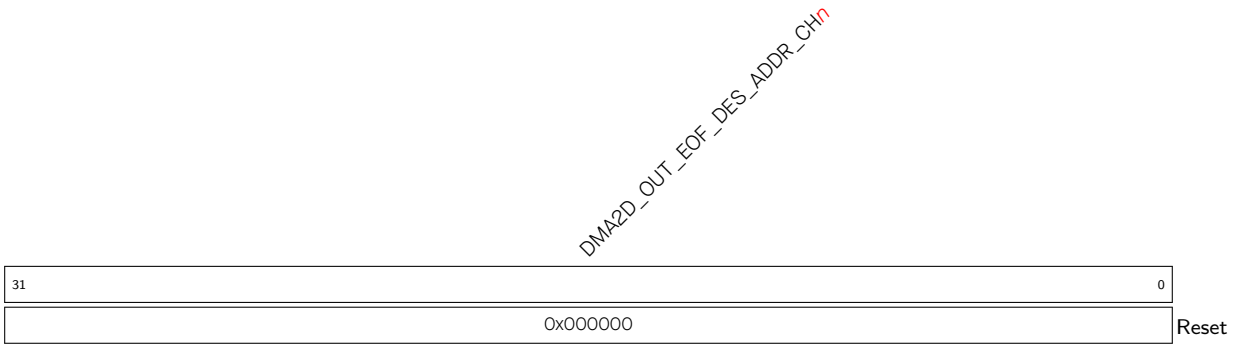
DMA2D_OUT_RESET_AVAIL_CH*n* 表示复位发送通道 *n* 是否安全。

0：不安全

1：安全

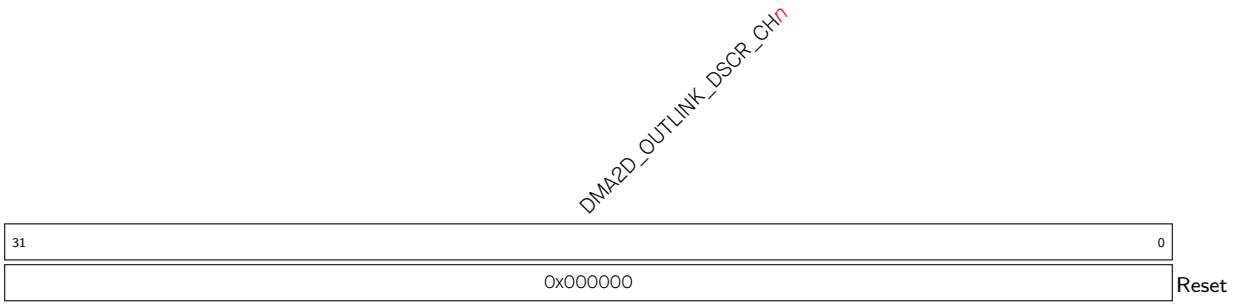
(RO)

Register 8.49. DMA2D_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-3) (0x0028+0x100**n*)



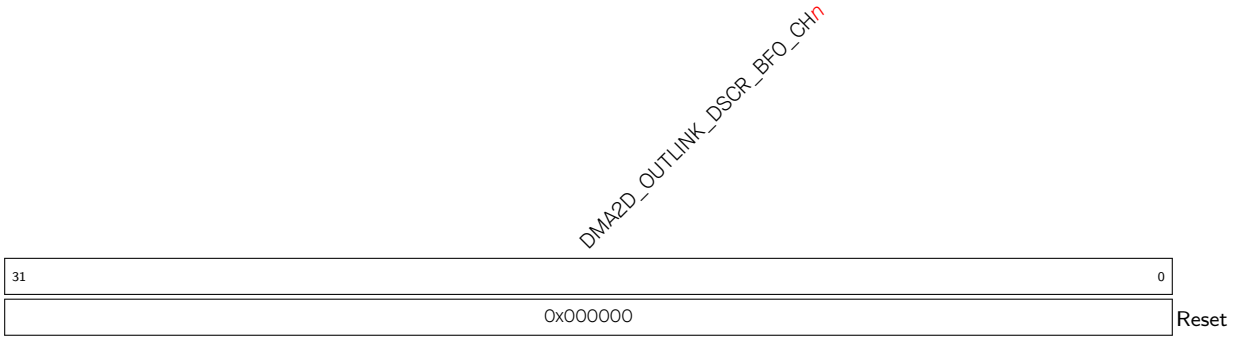
DMA2D_OUT_EOF_DES_ADDR_CH*n* 表示 eof 位为 1 的发送链表描述符的地址。(RO)

Register 8.50. DMA2D_OUT_DSCR_CH*n*_REG (*n*: 0-3) (0x002C+0x100**n*)



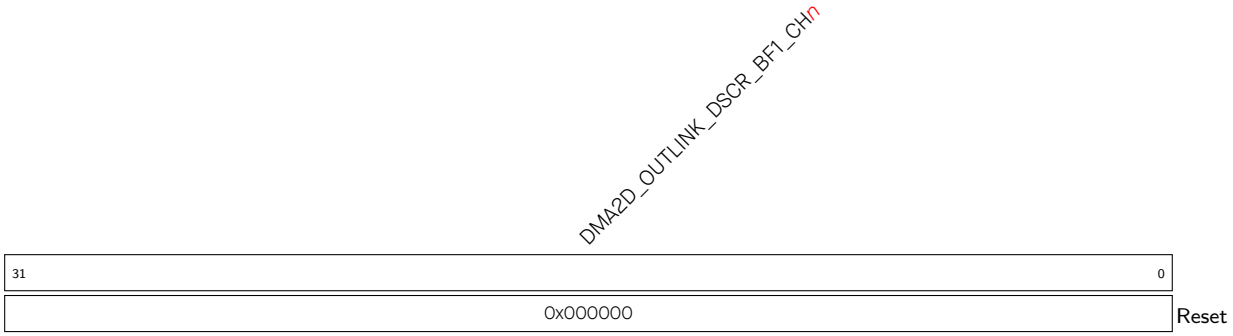
DMA2D_OUTLINK_DSCR_CH*n* 表示当前已预读取的发送链表描述符指向的下一个发送链表描述符地址。(RO)

Register 8.51. DMA2D_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-3) (0x0030+0x100**n*)



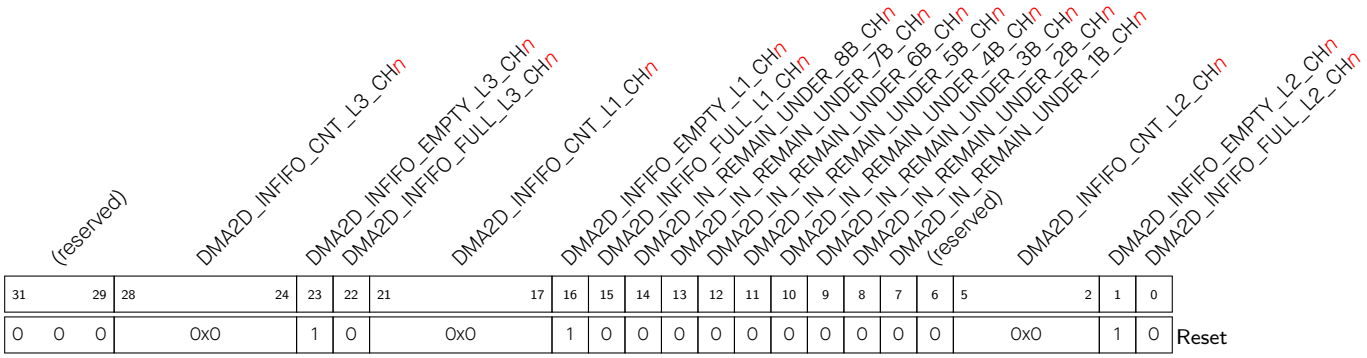
DMA2D_OUTLINK_DSCR_BF0_CH*n* 表示当前已预读取的发送链表描述符所在地址。(RO)

Register 8.52. DMA2D_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-3) (0x0034+0x100**n*)



DMA2D_OUTLINK_DSCR_BF1_CH*n* 表示前一个已预读取的发送链表描述符所在地址。(RO)

Register 8.53. DMA2D_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0514+0x100**n*)



DMA2D_INFIFO_FULL_L2_CH*n* 表示接收通道 *n* 的 L2 RX FIFO 是否为满。

0：未满

1：已满

(RO)

DMA2D_INFIFO_EMPTY_L2_CH*n* 表示接收通道 *n* 的 L2 RX FIFO 是否为空。

0：未空

1：已空

(RO)

DMA2D_INFIFO_CNT_L2_CH*n* 表示接收通道 *n* L2 RX FIFO 中的字节数。(RO)

DMA2D_IN_REMAIN_UNDER_1B_CH*n* 保留。(RO)

DMA2D_IN_REMAIN_UNDER_2B_CH*n* 保留。(RO)

DMA2D_IN_REMAIN_UNDER_3B_CH*n* 保留。(RO)

DMA2D_IN_REMAIN_UNDER_4B_CH*n* 保留。(RO)

DMA2D_IN_REMAIN_UNDER_5B_CH*n* 保留。(RO)

DMA2D_IN_REMAIN_UNDER_6B_CH*n* 保留。(RO)

DMA2D_IN_REMAIN_UNDER_7B_CH*n* 保留。(RO)

DMA2D_IN_REMAIN_UNDER_8B_CH*n* 保留。(RO)

DMA2D_INFIFO_FULL_L1_CH*n* 表示接收通道 *n* 的 L1 RX FIFO 是否为满。

0：未满

1：已满

(RO)

DMA2D_INFIFO_EMPTY_L1_CH*n* 表示接收通道 *n* 的 L1 RX FIFO 是否为空。

0：未空

1：已空

(RO)

DMA2D_INFIFO_CNT_L1_CH*n* 表示接收通道 *n* L1 RX FIFO 中的字节数。(RO)

见下页...

Register 8.53. DMA2D_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0514+0x100**n*)

接上页...

DMA2D_INFIFO_FULL_L3_CH*n* 表示接收通道 *n* 的 L3 RX FIFO 是否为满。

0：未满

1：已满

(RO)

DMA2D_INFIFO_EMPTY_L3_CH*n* 表示接收通道 *n* 的 L3 RX FIFO 是否为空。

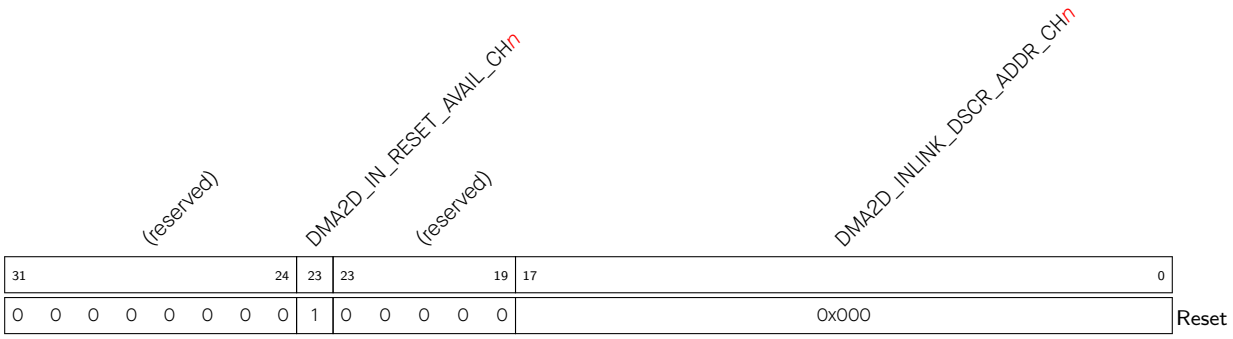
0：未空

1：已空

(RO)

DMA2D_INFIFO_CNT_L3_CH*n* 表示接收通道 *n* L3 RX FIFO 中的字节数。(RO)

Register 8.54. DMA2D_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0524+0x100**n*)



DMA2D_INLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的接收链表描述符所在地址的低 18 位。如果当前处理的接收链表描述符是最后一个描述符，则该字段表示当前处理的接收链表描述符地址。(RO)

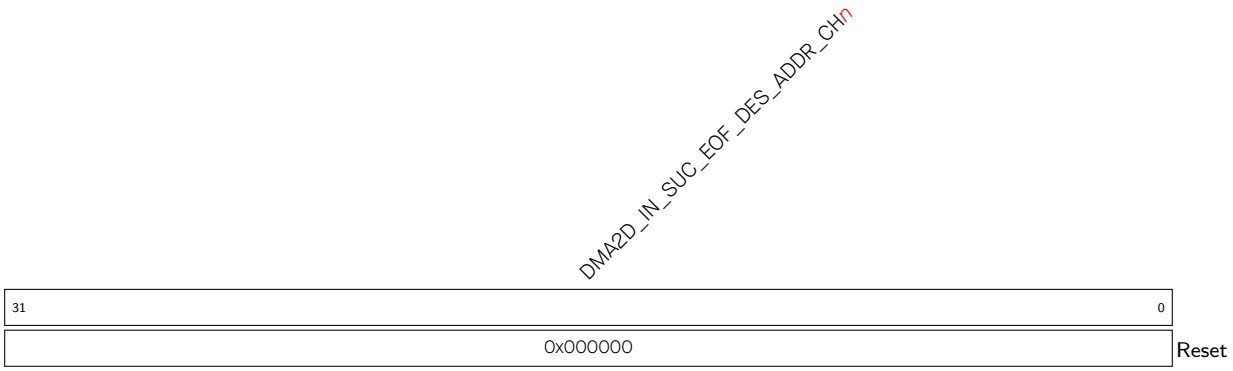
DMA2D_IN_RESET_AVAIL_CH*n* 表示复位接收通道 *n* 是否安全。

0：不安全

1：安全

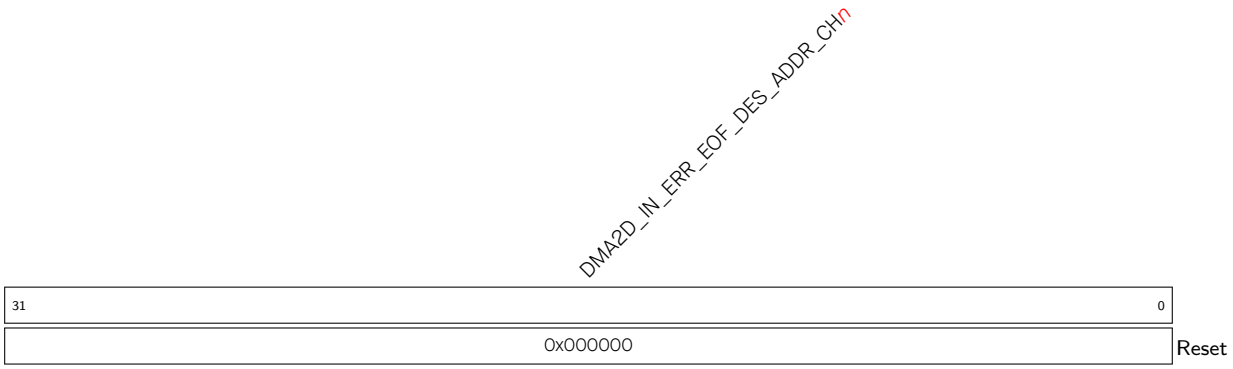
(RO)

Register 8.55. DMA2D_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0528+0x100**n*)



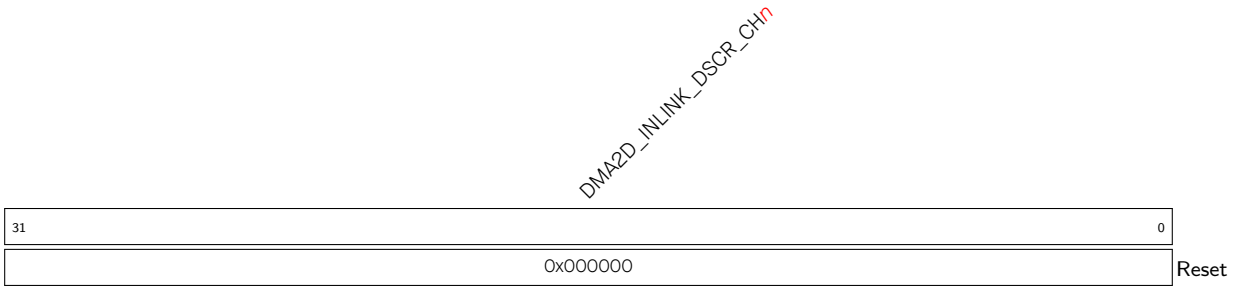
DMA2D_IN_SUC_EOF_DES_ADDR_CH*n* 表示 eof 位为 1 的接收链表描述符的地址。(RO)

Register 8.56. DMA2D_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x052C+0x100**n*)



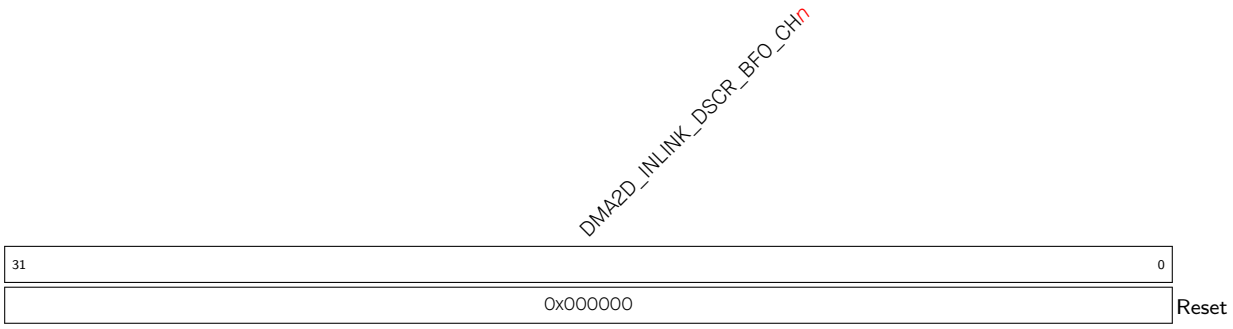
DMA2D_IN_ERR_EOF_DES_ADDR_CH*n* 表示当前接收数据有错时，相应接收链表描述符的位置。仅对 JPEG 有效。(RO)

Register 8.57. DMA2D_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0530+0x100**n*)



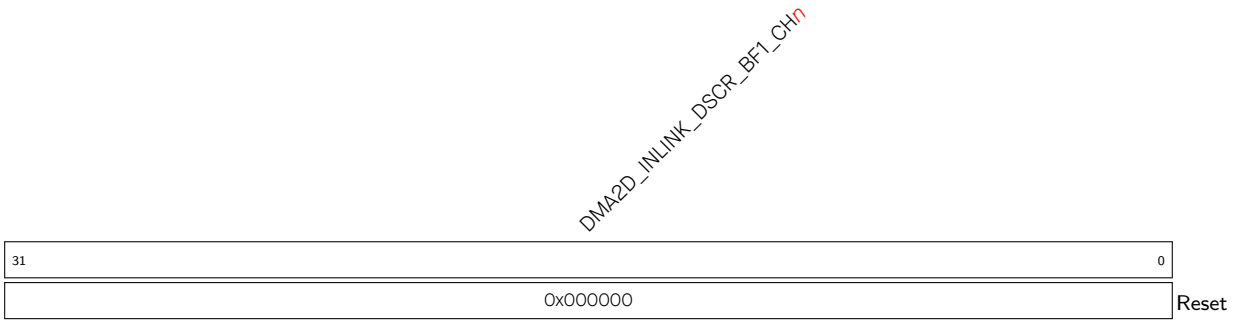
DMA2D_INLINK_DSCR_CH*n* 表示当前已预读取的接收链表描述符指向的下一个接收链表描述符地址。(RO)

Register 8.58. DMA2D_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0534+0x100**n*)



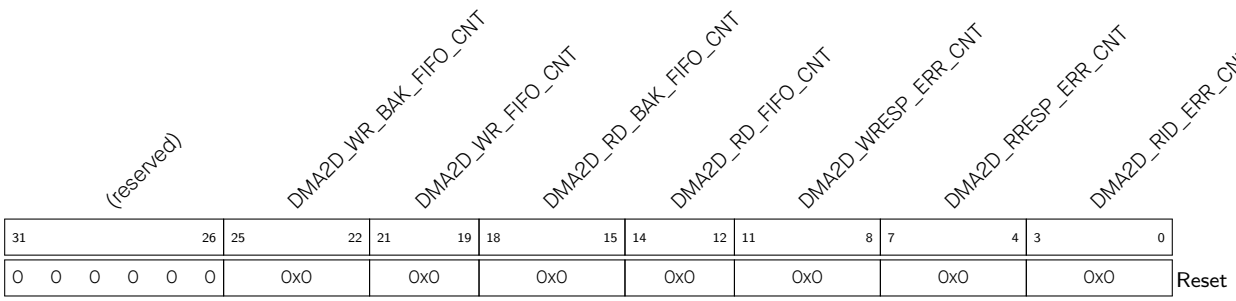
DMA2D_INLINK_DSCR_BF0_CH*n* 表示当前已预读取的接收链表描述符所在地址。(RO)

Register 8.59. DMA2D_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0538+0x100**n*)



DMA2D_INLINK_DSCR_BF1_CH*n* 表示前一个已预读取的接收链表描述符所在地址。(RO)

Register 8.60. DMA2D_AXI_ERR_REG (0x0A00)



DMA2D_RID_ERR_CNT 表示 AXI 读取 ID 错误的数量。(RO)

DMA2D_RRESP_ERR_CNT 表示 AXI 读取响应错误的数量。(RO)

DMA2D_WRESP_ERR_CNT 表示 AXI 写入响应错误的数量。(RO)

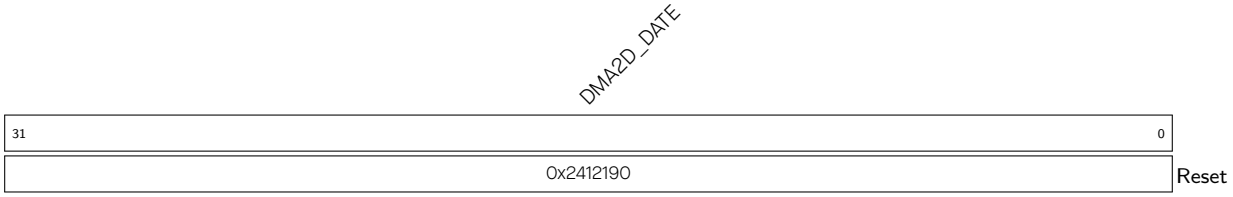
DMA2D_RD_FIFO_CNT 表示 AXI 读取命令 FIFO 中剩余的命令数量。(RO)

DMA2D_RD_BAK_FIFO_CNT 表示 AXI 读取备用命令 FIFO 中剩余的命令数量。(RO)

DMA2D_WR_FIFO_CNT 表示 AXI 写入命令 FIFO 中剩余的命令数量。(RO)

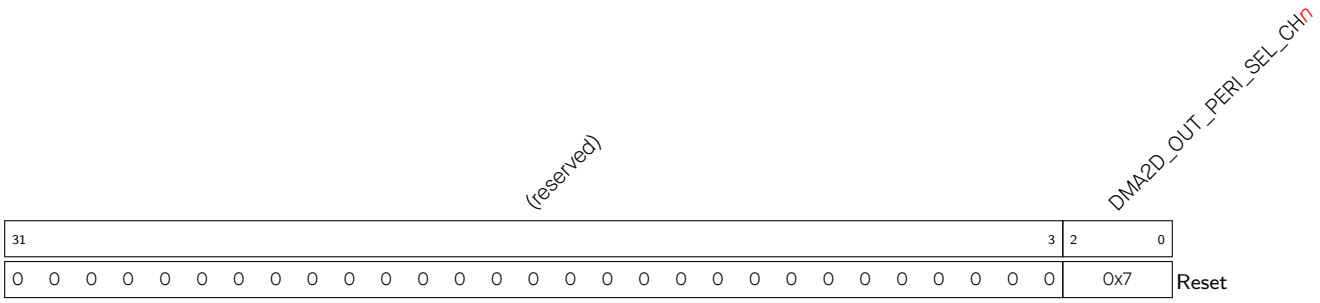
DMA2D_WR_BAK_FIFO_CNT 表示 AXI 写入备用命令 FIFO 中剩余的命令数量。(RO)

Register 8.61. DMA2D_DATE_REG (0x0A2C)



DMA2D_DATE 版本控制寄存器。(R/W)

Register 8.62. DMA2D_OUT_PERI_SEL_CH*n*_REG (*n*: 0-3) (0x0038+0x100**n*)



DMA2D_OUT_PERI_SEL_CH*n* 配置发送通道 *n* 连接的外设。

0：JPEG

1：PPA 的 SRM 模块

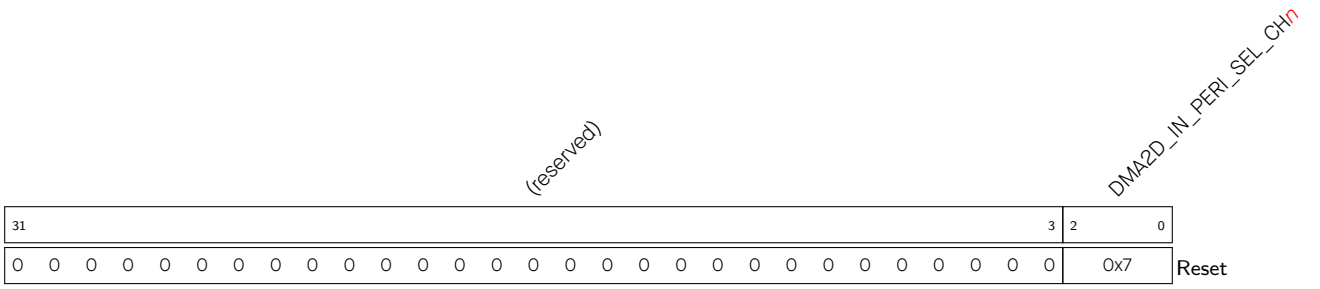
2：PPA 的 BLEND0 通道

3：PPA 的 BLEND1 通道

4 ~ 7：Dummy 4 ~ 7

(R/W)

Register 8.63. DMA2D_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x053C+0x100**n*)



DMA2D_IN_PERI_SEL_CH*n* 配置接收通道 *n* 连接的外设。

0：JPEG

1：PPA 的 SRM 模块

2：PPA 的 BLEND 模块

3 ~ 7：Dummy 3 ~ 7

(R/W)

第 III 卷

存储器组织结构

该部分详细描述系统的存储器组织结构，例如 RAM、ROM、eFuse 和外部存储器的组织和映射，为理解与存储器相关的子系统提供了框架。

第 9 章

系统和存储器

9.1 概述

ESP32-P4 集成了两个处理器，包括：

- 一个高性能 32 位 RISC-V 双核处理器 (HP CPU)，五级流水线，主频达 400 MHz。
- 一个低功耗 32 位 RISC-V 单核处理器 (LP CPU)，两级流水线，主频达 40 MHz。

所有的内部存储器、外部存储器以及外设均分布在 HP CPU 和 LP CPU 的总线上。

9.2 主要特性

- **地址空间**
  - 896 KB HP 内部存储器空间，可被 HP CPU 的指令总线或数据总线通过 cache 进行访问
  - 8 KB HP 内部存储器空间，可被 HP CPU 的指令总线或数据总线访问
  - 48 KB LP 内部存储器空间，可被 LP CPU 的指令总线或数据总线进行零等待访问
  - 1256 KB 外设地址空间
  - 64 MB 外部 flash 虚拟地址空间，可被指令总线或数据总线访问
  - 64 MB 外部 RAM 虚拟地址空间，可被指令总线或数据总线访问
  - 768 KB 内部 DMA 地址空间
  - 128 MB 外部 DMA 地址空间
- **内部存储器**
  - 128 KB HP ROM
  - 768 KB HP L2MEM
  - 8 KB HP SPM（暂存内存）
  - 16 KB LP ROM
  - 32 KB LP SRAM
- **外部存储器**
  - 最大支持 64 MB 外部 flash 和 RAM
- **外设空间**
  - 总计 96 个模块/外设

- GDMA
  - 7 个模块/外设支持 GDMA-AHB
  - 6 个模块/外设支持 GDMA-AXI

9.3 功能描述

9.3.1 地址映射

图 9.3-1 展示了系统结构与地址映射结构。所有非保留地址均可通过指令总线和数据总线进行访问，即指令总线和数据总线访问的地址空间一样。HP CPU 和 LP CPU 的数据总线与指令总线均为小端序。除 HP CPU DBUS 为 128 位数据宽度外，其他总线均为 32 位数据宽度。HP CPU 可以通过数据总线进行单字节、双字节、四字节的数据访问，也可以在 AI 指令 (AI Instruction) 中使用最多 16 字节对齐。LP CPU 可以通过数据总线进行单字节、双字节、四字节的数据访问。

HP CPU 支持：

- 直接通过数据总线和指令总线访问 HP SPM。
- 通过 cache 访问映射到地址空间的内部存储器和外部存储器。
  - 128 KB HP ROM (0x4FC0_0000 ~ 0x4FC1_FFFF)
  - 768 KB HP L2MEM (0x4FF0_0000 ~ 0x4FFB_FFFF)
  - 64 MB 外部 flash (0x4000_0000 ~ 0x43FF_FFFF)
  - 64 MB 外部 RAM (0x4800_0000 ~ 0x4BFF_FFFF)

> **说明：**
> 软件可以以 cache-able 或 noncache-able 的方式使用以上 “0x4” 开头的地址。其中 cache-able 访问方式是指通过 cache 进行访问；noncache-able 访问方式是不通过 cache 进行直接访问，与下文直接访问以 “0x8” 开头的地址类似，但直接访问 “0x8” 开头的地址多用于调试。cache-able 访问还是 noncache-able 访问则是由 CPU PMA (Physical Memory Attribution) 控制。

- 直接访问映射到地址空间的内部存储器和外部存储器，带宽相对通过 cache 访问较慢。
  - 128 KB HP ROM (0x8FC0_0000 ~ 0x8FC1_FFFF)
  - 768 KB HP L2MEM (0x8FF0_0000 ~ 0x8FFB_FFFF)
  - 32 KB LP SRAM (0x5010_8000 ~ 0x5010_FFFF)
  - 64 MB 外部 flash (0x8000_0000 ~ 0x83FF_FFFF)
  - 64 MB 外部 RAM (0x8800_0000 ~ 0x8BFF_FFFF)
- 直接通过数据总线访问模块/外设。
  - HP CPU 外设
  - HP 外设
  - LP 外设

LP CPU 支持：

- 直接通过数据总线和指令总线访问 LP SRAM 和 LP ROM。
- 支持访问内部存储器和外部存储器。
  - 128 KB HP ROM (0x4FC0_0000 ~ 0x4FC1_FFFF)
  - 768 KB HP L2MEM (0x4FF0_0000 ~ 0x4FFB_FFFF)
  - 64 MB 外部 flash (0x4000_0000 ~ 0x43FF_FFFF)
  - 64 MB 外部 RAM (0x4800_0000 ~ 0x4BFF_FFFF)
- 直接通过数据总线访问模块/外设。
  - HP CPU 外设
  - HP 外设
  - LP 外设

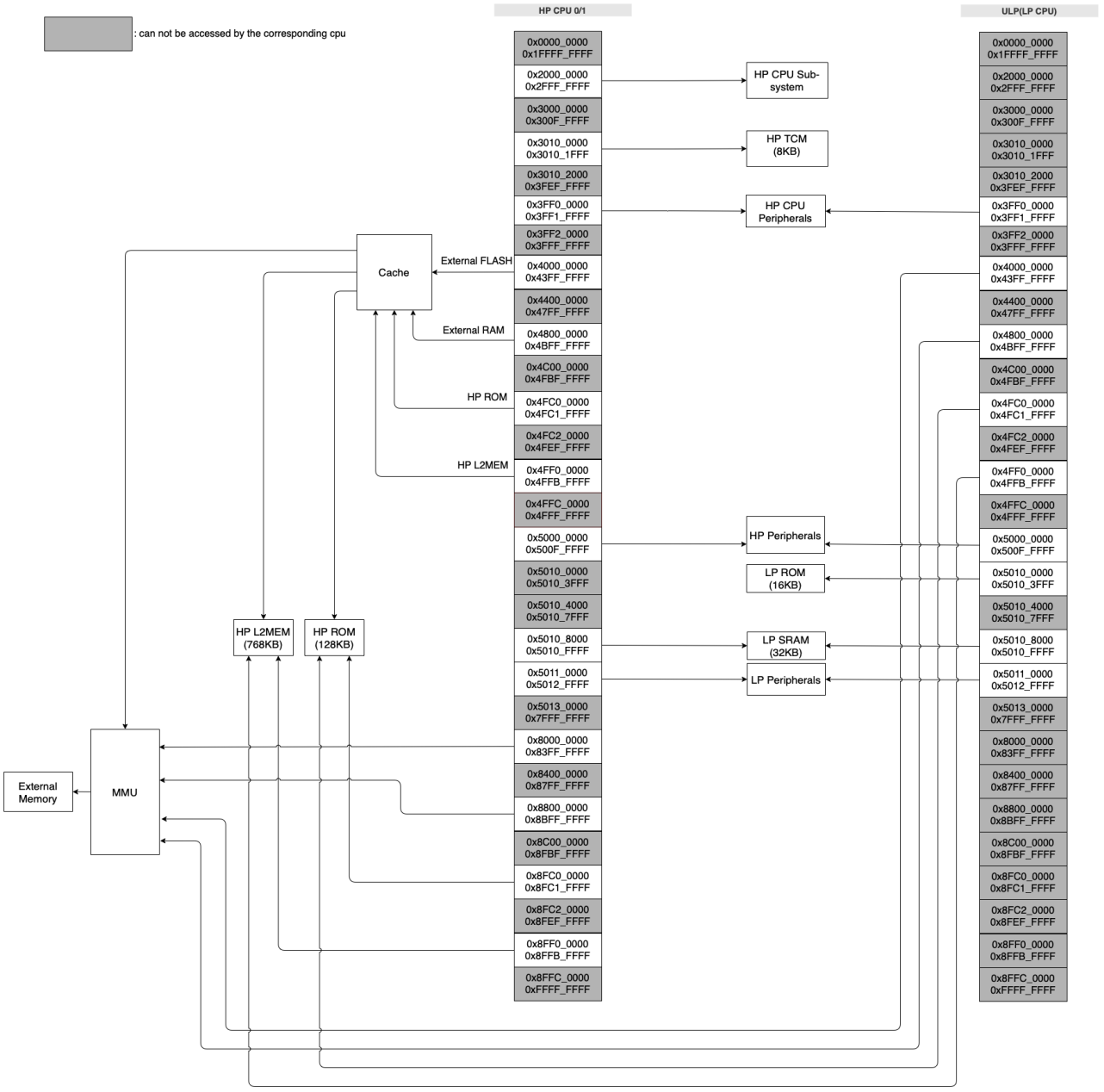


图 9.3-1. 系统结构和地址映射

- 地址空间中可用的地址范围可能大于实际可用的内存。
- 关于 HP CPU，请参考章节 2 高性能处理器。

表 9.3-1 列出了数据总线与指令总线中的各段地址所能访问的目标。

表 9.3-1. 地址映射

总线类型	边界地址		容量	目标*
	低位地址	高位地址		
	0x0000_0000	0x300F_FFFF		保留
数据总线/指令总线	0x3010_0000	0x3010_1FFF	8 KB	HP SPM
	0x3010_2000	0x3FEF_FFFF		保留

见下页

表 9.3-1 – 接上页

总线类型	边界地址		容量	目标*
	低位地址	高位地址		
数据总线/指令总线	0x3FF0_0000	0x3FF1_FFFF	128 KB	HP CPU 外设
	0x3FF2_0000	0x3FFF_FFFF		保留
数据总线/指令总线	0x4000_0000	0x43FF_FFFF	64 MB	外部 flash
	0x4400_0000	0x47FF_FFFF		保留
数据总线/指令总线	0x4800_0000	0x4BFF_FFFF	64 MB	外部 RAM
	0x4C00_000	0x4FBF_FFFF		保留
数据总线/指令总线	0x4FC0_0000	0x4FC1_FFFF	128 KB	HP ROM
	0x4FC2_0000	0x4FEF_FFFF		保留
数据总线/指令总线	0x4FF0_0000	0x4FFB_FFFF	768 KB	HP L2MEM
	0x4FFC_0000	0x4FFF_FFFF		保留
数据总线/指令总线	0x5000_0000	0x500F_FFFF	1 MB	HP 外设
数据总线/指令总线	0x5010_0000	0x5010_3FFF	16 KB	LP ROM
	0x5010_4000	0x5010_7FFF		保留
数据总线/指令总线	0x5010_8000	0x5010_FFFF	32 KB	LP SRAM
数据总线/指令总线	0x5011_0000	0x5012_FFFF	128 KB	LP 外设
	0x5013_0000	0x7FFF_FFFF		保留
数据总线/指令总线	0x8000_0000	0x83FF_FFFF	64 MB	外部 flash（CPU 直接访问）
	0x8400_0000	0x87FF_FFFF		保留
数据总线/指令总线	0x8800_0000	0x8BFF_FFFF	64 MB	外部 RAM（CPU 直接访问）
	0x8C00_000	0x8FBF_FFFF		保留
数据总线/指令总线	0x8FC0_0000	0x8FC1_FFFF	128 KB	HP ROM（CPU 直接访问）
	0x8FC2_0000	0x8FEF_FFFF		保留
数据总线/指令总线	0x8FF0_0000	0x8FFB_FFFF	768 KB	HP L2MEM（CPU 直接访问）
	0x8FFC_0000	0xFFFF_FFFF		保留

* 0x4xxx_xxxx 开头的地址可配置为 cache-able 或 noncache-able，具体由 CPU PMA 控制。0x8xxx_xxxx 开头的地址为 CPU 直接访问。

* 关于 HP CPU 和 LP CPU 可以访问的目标，请参考图 9.3-1。

9.3.2 内部存储器

ESP32-P4 的内部存储器包括如下类型：

- **HP ROM (128 KB)**：HP 系统的只读存储器，不可编程。用于存放系统底层软件的 ROM 代码和只读数据。HP CPU 通过 l2 cache 访问 HP ROM，访问频率为 HP CPU 频率的一半。
- **HP L2MEM (768 KB)**：易失性存储器，可被 HP CPU、LP CPU 或 DMA 外设访问。访问频率为 HP CPU 频率的一半。HP L2MEM 可配置为在 Light-sleep 模式下保持供电，因此可用于数据保留或寄存器备份。
- **LP ROM (16 KB)**：LP 系统的只读存储器。用于存放 LP CPU 启动代码以及一些系统基础函数。LP ROM 可被 LP CPU 零等待访问。
- **LP SRAM (32 KB)**：易失性存储器，可被 HP CPU 和 LP CPU 访问。访问频率与 LP CPU 频率相同。
- **HP SPM (8 KB)**：易失性存储器，可被 HP CPU 访问，每两个周期完成一次访问。

1. HP ROM

HP ROM 的容量为 128 KB，属于只读存储器，由 HP CPU 使用指令总线或数据总线通过以下地址进行访问，见表 9.3-1：

- 0x4FC0_0000（可配置为 cache-able 或 noncache-able）~ 0x4FC1_FFFF（可配置为 cache-able 或 noncache-able）。
- 0x8FC0_0000（CPU 直接访问）~ 0x8FC1_FFFF（CPU 直接访问）。

2. HP L2MEM

HP L2MEM 的容量为 768 KB，属于可读可写存储器，由 HP CPU，LP CPU 使用指令总线或数据总线或 DMA 主机通过 AHB 矩阵或 AXI 矩阵通过以下地址进行访问，见表 9.3-1：

- 0x4FF0_0000（可配置为 cache-able 或 noncache-able）~ 0x4FFB_FFFF（可配置为 cache-able 或 noncache-able）。
- 0x8FF0_0000（CPU 直接访问）~ 0x8FFB_FFFF（CPU 直接访问）。

L2MEM 支持多种容量划分配置：L2 RAM 与 L2 cache 共享 768 KB 总容量，可在以下四种配置中选择其一：

- Config 0：L2 RAM 256 KB，L2 cache 512 KB
- Config 1：L2 RAM 512 KB，L2 cache 256 KB
- Config 2：L2 RAM 640 KB，L2 cache 128 KB
- Config 3：L2 RAM 768 KB，L2 cache 0 KB

上述四种配置下 HP L2MEM 的地址布局如图 9.3-2 所示。

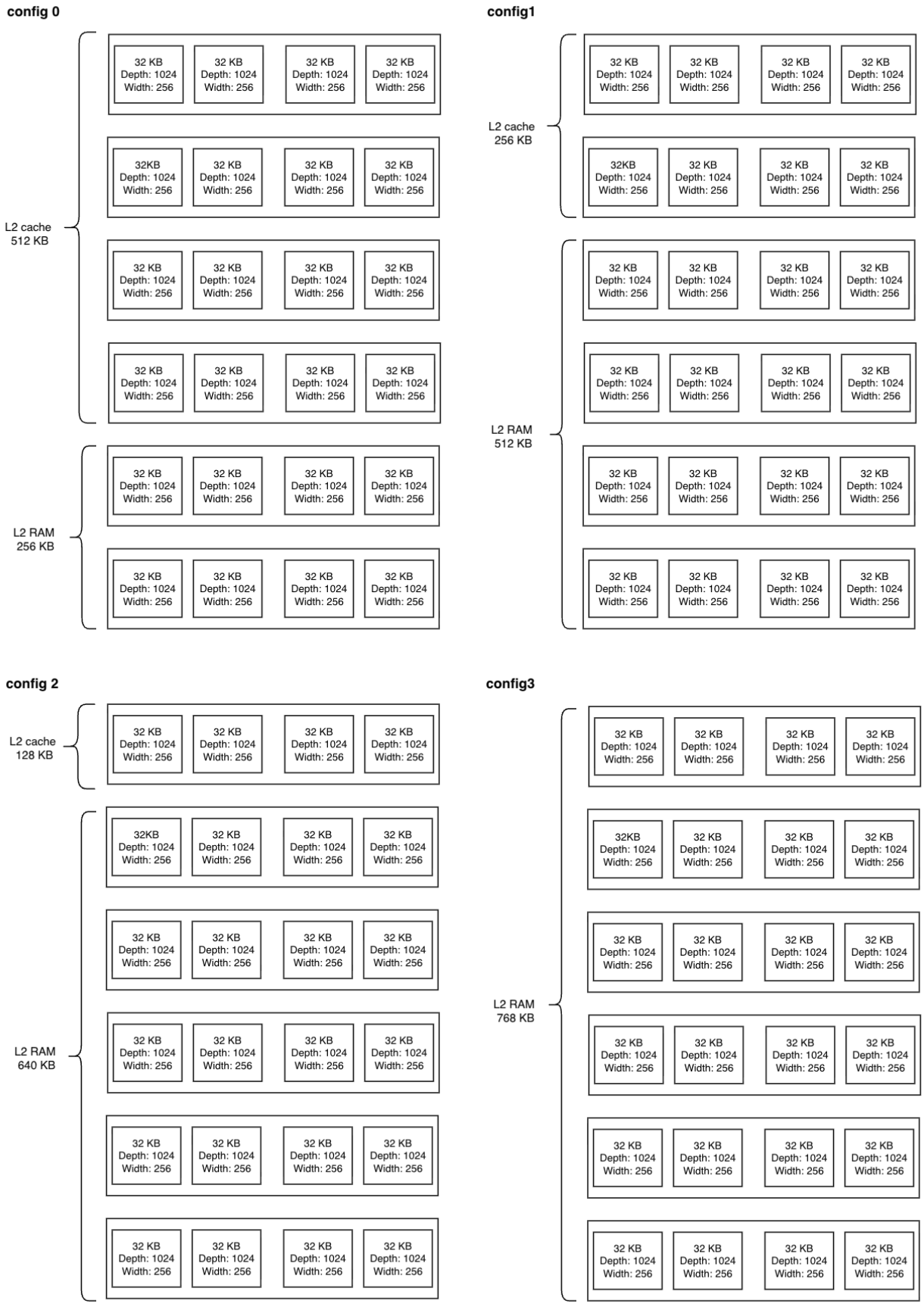


图 9.3-2. HP L2MEM 地址布局

HP L2MEM 可被以下外设访问：

- USB 2.0 全速 OTG、USB 2.0 高速 OTG、GMAC、SDMMC、TRACE、GDMA-AHB 和 SYSMON 通过 AHB 矩阵访问。
- GDMA、GDMA-AXI、2D-DMA 和 H264 编码器通过 AXI 矩阵访问。

HP L2MEM 支持纠错码校验 (Error Correction Code check)，即 ECC 校验，可以纠正 1 位内存翻转错误。如果发生 1 位翻转错误，HP L2MEM 中的数据仍然有效。在 ECC 检查之前，需要初始化 HP L2MEM。

HP L2MEM 由 6 个存储单元组成，每个为 128 KB。初始化过程可以为每个单元单独配置，以优化功耗。建议按顺序逐一初始化存储单元，不建议同时初始化所有单元。初始化整个 768 KB HP L2MEM 会导致过多的功耗，触发欠压。

配置步骤

以单元 0 为例进行说明，其他单元的操作与单元 0 相同。

- 置位 HP_SYSTEM_L2_MEM_UNIT0_REFRESH_EN，并清零 HP_SYSTEM_L2_MEM_REFRESH_CNT_RESET，开始初始化 HP L2MEM。
- 读取 HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE 的值。
- HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE 的值变为 1 后，置位 HP_SYSTEM_L2_RAM_UNIT0_ECC_EN 开始 ECC 校验。
- HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE 的值变为 1 之后，建议清零 HP_SYSTEM_L2_MEM_UNIT0_REFRESH_EN 并置位 HP_SYSTEM_L2_MEM_REFRESH_CNT_RESET，结束初始化。

在初始化过程中，避免访问初始化中的 HP L2MEM 单元。

3. LP ROM

LP ROM 的容量为 16 KB，属于只读存储器，由 LP CPU 使用指令总线或数据总线通过共用地址 (0x5010_0000 ~ 0x5010_3FFF) 进行访问，如表 9.3-1 所示。

4. LP SRAM

LP SRAM 的容量为 32 KB，属于可读可写存储器，由 HP CPU 使用 AHB 矩阵或 LP CPU 使用指令总线或数据总线通过共用地址 (0x5010_8000 ~ 0x5010_FFFF) 进行访问，如表 9.3-1 所示。LP SRAM 支持原子操作。

5. HP SPM

HP SPM 的容量为 8 KB，属于可读可写存储器，由 HP CPU 使用指令总线或数据总线通过共用地址 (0x3010_0000 ~ 0x3010_1FFF) 进行访问，如表 9.3-1 所示。HP SPM 支持 HP CPU 原子操作。

HP SPM 支持奇偶校验，检测 1 位内存翻转错误。在奇偶校验之前，需要先初始化 HP SPM。

配置步骤

- 置位 HP_SYSTEM_SPM_INIT_EN，并清零 HP_SYSTEM_SPM_INIT_CNT_RESET，开始初始化 HP SPM。
- 读取 HP_SYSTEM_SPM_INIT_DONE 的值。
- HP_SYSTEM_SPM_INIT_DONE 的值变为 1 后，置位 HP_SYSTEM_HP_SPM_PARITY_CHECK_EN 开始奇偶校验。
- HP_SYSTEM_SPM_INIT_CNT_RESET 的值变为 1 之后，建议清零 HP_SYSTEM_SPM_INIT_EN 并置位 HP_SYSTEM_SPM_INIT_CNT_RESET，结束初始化。

在初始化过程中，避免访问 HP SPM。

9.3.3 外部存储器

ESP32-P4 支持以 SPI、Dual SPI、Quad SPI、QPI 等接口形式连接外部 flash，支持以 OPI、HPI 等接口形式连接外部 RAM。ESP32-P4 还支持以下安全机制，用以保护用户存储在外部 flash 和 RAM 中的数据：

- 支持基于 XTS-AES 的硬件手动和自动加解密功能，从而保护开发者外部 flash 中的程序和数据。
- 支持基于 XTS-AES 的自动加解密功能，从而保护开发者外部 RAM 中的数据。

9.3.3.1 外部存储器地址映射

HP CPU 可使用 cache-able 或 noncache-able 的方式通过地址段 0x4000_0000 ~ 0x43FF_FFFF 或 0x4800_0000 ~ 0x4BFF_FFFF 访问外部 flash 或外部 RAM。HP CPU 也可以通过地址段 0x8000_0000 ~ 0x83FF_FFFF 或 0x8800_0000 ~ 0x8BFF_FFFF 直接访问外部 flash 或外部 RAM，该方式相对通过 cache 的访问较慢，因此通常用于调试。

HP CPU 访问外部存储器时，将根据内存管理单元 (Memory Management Unit, MMU) 中的信息，把 HP CPU 的地址映射为具体的实地址：

- (0x4000_0000 ~ 0x43FF_FFFF) 映射为 HP CPU 通过 cache 访问外部 flash 的实地址。
- (0x4800_0000 ~ 0x4BFF_FFFF) 映射为 HP CPU 通过 cache 访问外部 RAM 的实地址。
- (0x8000_0000 ~ 0x83FF_FFFF) 映射为 HP CPU 直接访问外部 flash 的实地址。
- (0x8800_0000 ~ 0x8BFF_FFFF) 映射为 HP CPU 直接访问外部 RAM 的实地址。

经过地址映射，ESP32-P4 最大支持 64 MB 的外部 flash 和 64 MB 的外部 RAM。请注意，指令总线地址空间 (64 MB) 和数据总线地址空间 (64 MB) 是共用的。

9.3.3.2 cache

如图 9.3-3 所示，ESP32-P4 采用两级 cache 系统，具有如下特性：

- 一级 cache (l1 cache) 包括：
  - 指令 cache (icache)：大小为 16 KB，块大小为 64 B，四路组相联
  - 数据 cache (dcache)：大小为 64 KB，块大小为 64 B，两路组相联，支持透写 (write-through) 和回写 (write-back) 两种写策略
- 二级 cache (l2 cache)：大小为 128 KB/256 KB/512 KB，块大小为 64 B/128 B，八路组相联
- 支持 cache-able 访问和 noncache-able 访问，可通过 CPU PMA 配置访问属性
- 支持预取 (preload) 功能
- 支持锁定 (lock) 功能
- 支持关键字优先 (critical word first) 和提前重启 (early restart)

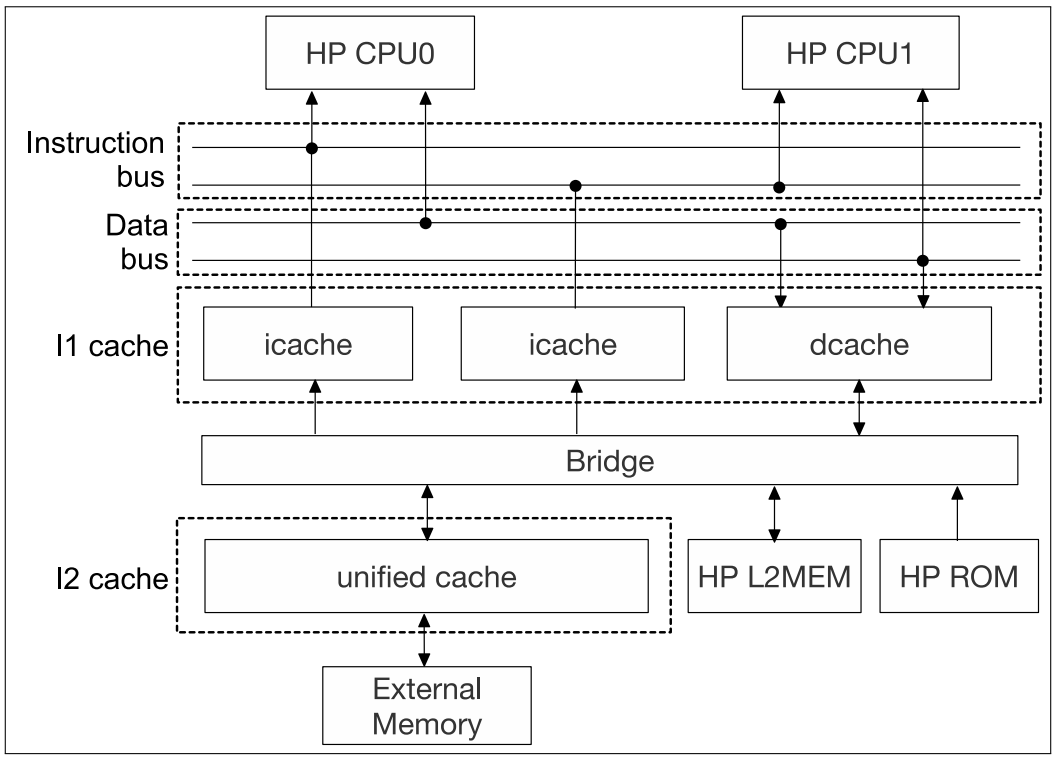


图 9.3-3. cache 结构

9.3.3.3 cache 操作

ESP32-P4 I1 cache 和 I2 cache 支持以下操作：

1. **回写 (Write-back)**（仅 dcache 和 I2 cache 支持该功能）：
   - 清除标记存储器中的脏位，并将新数据更新到外部存储器。
   - 回写操作完成后，新数据将被更新到外部存储器中。
   - 可以选择是否对 cache 中的数据作失效处理。
   - 如果没有选择失效处理，那么 CPU 可以直接从 cache 读取/写入数据，而不会发生缓存缺失。
2. **清除 (Clean)**（仅 dcache 和 I2 cache 支持该功能）：
   - 清除标记存储器中的脏位，但不把新数据更新到外部存储器。
   - 清理操作完成后，外部存储器中存储的仍然是旧数据，而 cache 会保留新的数据，但 cache 并不知道其存储的为新数据。
   - 然后 CPU 可以直接从 cache 读取/写入数据，而不会发生缓存缺失。
3. **失效 (Invalidate)**：
   - 删除 cache 中的有效数据。
   - 该操作完成后，删除的数据将仅存于外部存储器中。
   - 如果 HP CPU 接着去访问该数据，那么需要访问外部存储器。
   - 该操作包括两种类型：
     - 手动失效 (Manual-Invalidate)：仅对 cache 中落入指定区域的地址对应的数据做失效处理。

  - 整体失效 (Invalidate-All)：会对 cache 中的所有数据做失效处理。

4. **预取 (Preload)**：
   - 将指令和数据提前加载到 cache 中。
   - 预取操作的最小单位为 1 个缓存块。
   - 该操作包括两种类型：
     - 手动预取 (Manual-Preload)：硬件按软件指定的虚地址预取一段连续的数据。
     - 自动预取 (Auto-Preload)：硬件根据当前命中/缺失（取决于配置）的地址，自动预取一段连续的数据。
5. **锁定/解锁 (Lock/Unlock)**：
   - 保护 cache 中的数据不被替换掉。
   - 该锁定操作包括两种类型：
     - 预锁定：在填充缺失数据到 cache 时，如果该数据落在指定区域，cache 则将该数据锁定，未落入指定区域的数据不会被锁定。
     - 手动锁定：检查 cache 中的数据，并将落在指定区域的数据锁定，未落入指定区域的数据不会被锁定。
   - 当缺失发生时，cache 会优先替换掉未被锁定的那一路的数据，因此锁定区域的数据会一直保存在 cache 中。
   - 但当所有路都被锁定时，cache 将进行正常替换，就像所有路都没有被锁定一样。
   - 解锁是锁定的逆操作，但解锁只有手动解锁。
   - 请注意，手动失效操作只对未被锁定的数据起作用。如果想对已锁定的数据执行手动失效操作，请先解锁这些数据。

9.3.4 DMA 地址空间

ESP32-P4 支持 AHB 通用直接内存访问 (GDMA-AHB)、AXI 通用直接内存访问 (GDMA-AXI) 和 VDMA。均可提供直接内存访问服务。在 ESP32-P4 中，USB 2.0 高速 OTG、USB 2.0 全速 OTG、GDMAC、Trace0/1、VDMA、2D-DMA、H264 编码器等主机模块也都内置了 DMA 功能模块，也可以提供直接内存访问服务。

上述所有支持 DMA 的主机模块都可以访问：

- HP L2MEM (0x4FF0_0000~0x4FFB_FFFF)。
- 外部 flash (0x4000_0000 ~ 0x43FF_FFFF)。
- 外部 RAM (0x4800_0000 ~ 0x4BFF_FFFF)。

此外，VDMA 主机 1 还可以访问：

- MIPI CSI 内存 (0x5010_4000 ~ 0x5010_4FFF)。
- MIPI DSI 内存 (0x5010_5000 ~ 0x5010_5FFF)。

说明：
当使用 DMA 访问任何存储器时，都需要获取对应的访问权限，否则访问将会失败。关于权限管理的更多信息，请参考

章节 20 权限控制 (PMS)。

9.3.5 模块/外设地址映射

表 9.3-2 详细列出了模块/外设地址空间的各段地址与其能访问到的模块/外设的映射关系。其中，“边界地址”（包括低位地址和高位地址）栏中的两列数值共同决定了对应模块/外设的地址空间。

表 9.3-2. 模块/外设地址空间映射表

目标	边界地址		容量 (KB)
	低位地址	高位地址	
HP CPU 外设 (HP CPU PERI)			
保留	0x3FF0_0000	0x3FF0_3FFF	
RISC-V 追踪编码器 0	0x3FF0_4000	0x3FF0_4FFF	4
RISC-V 追踪编码器 1	0x3FF0_5000	0x3FF0_5FFF	4
总线监测	0x3FF0_6000	0x3FF0_6FFF	4
保留	0x3FF0_7000	0x3FF0_DFFF	
L2MEM 监测	0x3FF0_E000	0x3FF0_EFFF	4
SPM 监测	0x3FF0_F000	0x3FF0_FFFF	4
HP 外设 0 (HP PERIO)			
USB 2.0 高速 OTG	0x5000_0000	0x5003_FFFF	256
USB 2.0 全速 OTG	0x5004_0000	0x5007_FFFF	256
USB 2.0 全速 OTG PHY	0x5008_0000	0x5008_0FFF	4
VDMA 控制器	0x5008_1000	0x5008_1FFF	4
保留	0x5008_2000	0x5008_2FFF	
SD/MMC 主机控制器	0x5008_3000	0x5008_3FFF	4
H264 编码器	0x5008_4000	0x5008_4FFF	4
GDMA-AHB	0x5008_5000	0x5008_5FFF	4
JPEG 图像编解码器	0x5008_6000	0x5008_6FFF	4
像素处理加速器 (PPA)	0x5008_7000	0x5008_7FFF	4
2D-DMA 控制器	0x5008_8000	0x5008_8FFF	4
密钥管理器	0x5008_9000	0x5008_9FFF	4
GDMA-AXI	0x5008_A000	0x5008_BFFF	8
AES 加速器 (AES)	0x5009_0000	0x5009_0FFF	4
SHA 加速器 (SHA)	0x5009_1000	0x5009_1FFF	4
RSA 加速器 (RSA)	0x5009_2000	0x5009_2FFF	4
ECC 加速器 (ECC)	0x5009_3000	0x5009_3FFF	4
RSA 数字签名外设 (RSA_DS)	0x5009_4000	0x5009_4FFF	4
HMAC 加速器 (HMAC)	0x5009_5000	0x5009_5FFF	4
ECDSA 数字签名外设 (ECDSA_DS)	0x5009_6000	0x5009_6FFF	4
保留	0x5009_7000	0x5009_7FFF	
GMAC	0x5009_8000	0x5009_BFFF	16
USB 2.0 高速 OTG PHY	0x5009_C000	0x5009_CFFF	4
保留	0x5009_D000	0x5009_DFFF	

见下页

表 9.3-2 – 接上页

目标	边界地址		容量 (KB)
	低位地址	高位地址	
CSI 主机	0x5009_F000	0x500F_FFFF	4
DSI 主机	0x500A_0000	0x500A_0FFF	4
图像信号处理器 (ISP)	0x500A_1000	0x500A_1FFF	4
红外遥控 (RMT)	0x500A_2000	0x500A_2FFF	4
比特传输控制器	0x500A_3000	0x500A_3FFF	4
AXI ICM	0x500A_4000	0x500A_4FFF	4
HP 外设权限控制 (HP_PERI_PMS)	0x500A_5000	0x5009_57FF	2
LP2HP 外设权限控制 (LP2HP_PERI_PMS)	0x500A_5800	0x500A_5FFF	2
HP DMA 权限控制 (HP_DMA_PMS)	0x500A_6000	0x500A_6FFF	4
H264 DMA	0x500A_7000	0x500A_7FFF	4
保留	0x500A_8000	0x500B_FFFF	
HP 外设 1 (HP PERI1)			
电机控制脉宽调制器 0 (MCPWM0)	0x500C_0000	0x500C_0FFF	4
电机控制脉宽调制器 1 (MCPWM1)	0x500C_1000	0x500C_1FFF	4
定时器组 0 (TIMG0)	0x500C_2000	0x500C_2FFF	4
定时器组 1 (TIMG1)	0x500C_3000	0x500C_3FFF	4
I2C 控制器 0 (I2C0)	0x500C_4000	0x500C_4FFF	4
I2C 控制器 1 (I2C1)	0x500C_5000	0x500C_5FFF	4
I2S 控制器 0 (I2S0)	0x500C_6000	0x500C_6FFF	4
I2S 控制器 1 (I2S1)	0x500C_7000	0x500C_7FFF	4
I2S 控制器 2 (I2S2)	0x500C_8000	0x500C_8FFF	4
脉冲计数控制器 (PCNT)	0x500C_9000	0x500C_9FFF	4
UART 控制器 0 (UART0)	0x500C_A000	0x500C_AFFF	4
UART 控制器 1 (UART1)	0x500C_B000	0x500C_BFFF	4
UART 控制器 2 (UART2)	0x500C_C000	0x500C_CFFF	4
UART 控制器 3 (UART3)	0x500C_D000	0x500C_DFFF	4
UART 控制器 4 (UART4)	0x500C_E000	0x500C_EFFF	4
并行 IO 控制器 (PARL_IO)	0x500C_F000	0x500C_FFFF	4
通用 SPI 2 (GP-SPI2)	0x500D_0000	0x500D_0FFF	4
通用 SPI 3 (GP-SPI3)	0x500D_1000	0x500D_1FFF	4
USB 串口/JTAG 控制器	0x500D_2000	0x500D_2FFF	4
LED PWM 控制器 (LEDC)	0x500D_3000	0x500D_3FFF	4
保留	0x500D_4000	0x500D_4FFF	
事件任务矩阵 (SOC_ETM)	0x500D_5000	0x500D_5FFF	4
中断矩阵 (INTMTX)	0x500D_6000	0x500D_6FFF	4
双线汽车接口 0 (TWAI0)	0x500D_7000	0x500D_7FFF	4
双线汽车接口 1 (TWAI1)	0x500D_8000	0x500D_8FFF	4
双线汽车接口 2 (TWAI2)	0x500D_9000	0x500D_9FFF	4
I3C 主机控制器	0x500D_A000	0x500D_AFFF	4
I3C 从机控制器	0x500D_B000	0x500D_BFFF	4
LCD 与 Camera 控制器 (LCD_CAM)	0x500D_C000	0x500D_CFFF	4

见下页

表 9.3-2 – 接上页

目标	边界地址		容量 (KB)
	低位地址	高位地址	
保留	0x500D_D000	0x500D_DFFF	
ADC 控制器	0x500D_E000	0x500D_EFFF	4
UHCI 控制器 (UHCI)	0x500D_F000	0x500D_FFFF	4
GPIO 交换矩阵	0x500E_0000	0x500E_0FFF	4
IO MUX	0x500E_1000	0x500E_1FFF	4
系统定时器 (SYSTIMER)	0x500E_2000	0x500E_2FFF	4
保留	0x500E_3000	0x500E_4FFF	
HP 系统寄存器 (SYSREG)	0x500E_5000	0x500E_5FFF	4
复位与时钟	0x500E_6000	0x500E_6FFF	4
保留	0x500E_7000	0x500F_FFFF	
MIPI Camera 和 LCD 存储器			
MIPI Camera 存储器	0x5010_4000	0x5010_4FFF	4
MIPI LCD 存储器	0x5010_5000	0x5010_5FFF	4
LP 常开外设 (LP AON PERI)			
LP 系统寄存器	0x5011_0000	0x5011_0FFF	4
LP 常开时钟与复位	0x5011_1000	0x5011_1FFF	4
LP 定时器	0x5011_2000	0x5011_2FFF	4
LP 模拟外设 (LP ANAPERI)	0x5011_3000	0x5011_3FFF	4
LP HUK	0x5011_4000	0x5011_4FFF	4
电源管理单元 (PMU)	0x5011_5000	0x5011_5FFF	4
LP 看门狗定时器 (LP WDT)	0x5011_6000	0x5011_6FFF	4
保留	0x5011_7000	0x5011_7FFF	
LP 信箱控制器 (LP MB)	0x5011_8000	0x5011_8FFF	4
保留	0x5011_A000	0x5011_FFFF	
LP 外设 (LP PERI)			
LP 外设时钟与复位 (LP PERICLKRST)	0x5012_0000	0x5012_0FFF	4
LP UART	0x5012_1000	0x5012_1FFF	4
LP I2C	0x5012_2000	0x5012_2FFF	4
LP SPI	0x5012_3000	0x5012_3FFF	4
LP 模拟 I2C	0x5012_4000	0x5012_4FFF	4
LP I2S	0x5012_5000	0x5012_5FFF	4
保留	0x5012_6000	0x5012_6FFF	4
LP ADC	0x5012_7000	0x5012_7FFF	4
保留	0x5012_8000	0x5012_9FFF	
LP GPIO 交换矩阵	0x5012_A000	0x5012_AFFF	4
LP IO MUX	0x5012_B000	0x5012_BFFF	4
LP 中断 (LP INTR)	0x5012_C000	0x5012_CFFF	4
LP eFuse	0x5012_D000	0x5012_DFFF	4
LP 外设权限控制 (LP_PERI_PMS)	0x5012_E000	0x5012_E7FF	2
HP2LP 外设权限控制 (HP2LP_PERI_PMS)	0x5012_E800	0x5012_EFFF	2
LP 温度传感器	0x5012_F000	0x5012_FFFF	4

说明：

如图 9.3-1 所示，HP CPU 和 LP CPU 可以访问表 9.3-2 所列的 HP CPU PERI、HP PERIO、HP PERI1、LP AON PERI 和 LP PERI 外设。

9.4 eFuse 控制器 (EFUSE)

9.4.1 概述

ESP32-P4 系统中有一块 4096 位的 eFuse 存储器用于存储参数内容和用户数据，参数内容包括一些硬件模块的控制参数、系统数据参数以及加解密模块使用的密钥等。eFuse 存储器的各个位一旦被烧写为 1，则不能再恢复为 0。eFuse 控制器按照用户配置完成对 eFuse 存储器中各参数中的各个位的烧写。从芯片外部，eFuse 数据只能通过 eFuse 控制器读取。对于某些数据，例如存储在 eFuse 中供硬件加密模块（例如数字签名、HMAC 等）在内部使用的某些密钥，如果未启用读保护，则可以从芯片外部读取该数据；如果启用了读保护，则无法从芯片外部读取该数据。

9.4.2 主要特性

- 支持 4096 位一次性可编程存储（最多有 1792 个保留位供用户使用，具体取决于 EFUSE_KEY_PURPOSE_*n* 是否都配置成 0，见表 9.4-2）
- 烧写保护可配置
- 读取保护可配置
- 使用多种硬件编码方式保护参数内容

9.4.3 功能描述

9.4.3.1 结构

eFuse 控制器和 eFuse 存储器共同组成了 eFuse 系统。其内部的数据通路如图 9.4-1 所示。

用户可以通过将待烧写的数据填入烧写寄存器中并执行烧写指令，通过 eFuse 控制器完成对 eFuse 存储器内部各个位的烧写。详细烧写步骤可以参考章节 9.4.3.2。

用户不能直接读取 eFuse 存储器中烧写的信息内容，因此需要通过 eFuse 控制器将烧写的数据信息读取到对应地址段的读寄存器内。如果数据在读取过程中出现了和 eFuse 存储器不一致的错误，eFuse 控制器还可以通过硬件编码机制对数据进行自动校正（详见章节 9.4.3.1.3 ），并将错误信息自动更新至错误报告寄存器中。详细的读取参数步骤可以参考章节 9.4.3.3。

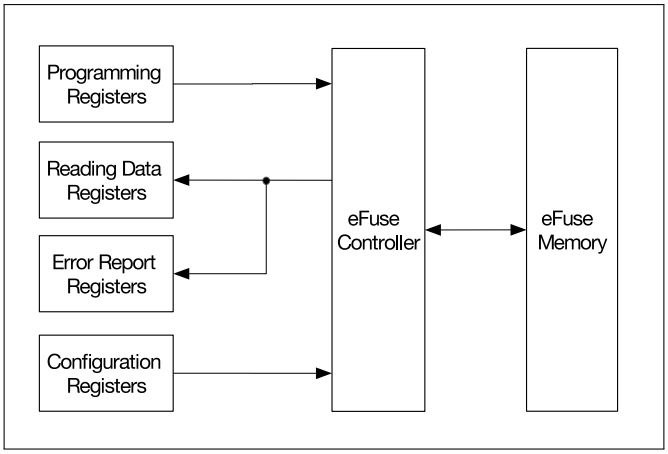


图 9.4-1. eFuse 系统数据通路

eFuse 存储器从结构上分成 11 个块 (BLOCK0 ~ BLOCK10)。

BLOCK0 存储了大部分供软硬件使用的控制参数。

表 9.4-1 列出了用户可访问（可读并可用）的所有 BLOCK0 中的参数名称、位宽、是否可供硬件使用、烧写保护位、以及简要的功能描述，若需了解更多有关该参数的描述，请点击表格中对应参数的链接。

在这些参数中，**EFUSE_WR_DIS** 用于控制其他参数的烧写保护状态，**EFUSE_RD_DIS** 用于控制 BLOCK4 ~ BLOCK10 的读取保护状态。更多关于这两个参数的信息请见章节 9.4.3.1.1、9.4.3.1.2。

表 9.4-1. eFuse BLOCK0 中的参数

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	功能描述
EFUSE_WR_DIS	32	Y	N/A	表示是否禁用 eFuse 控制器对 eFuse 位的烧写。
EFUSE_RD_DIS	7	Y	0	表示是否禁用从 eFuse 存储器的 BLOCK4 ~ BLOCK10 读取数据。
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1	2	N	N/A	表示当主引导程序失败时，ROM 引导程序使用的恢复引导程序的起始 Flash 扇区（Flash 扇区大小为 0x1000）。
EFUSE_DIS_USB_JTAG	1	Y	2	设置此位以禁用 USB 设备模块中 USB 切换到 JTAG 的功能。
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2	1	N	N/A	表示当主引导程序失败时，ROM 引导程序使用的恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。
EFUSE_DIS_FORCE_DOWNLOAD	1	Y	2	设置此位以禁用强制芯片进入下载模式的功能。
EFUSE_SPI_DOWNLOAD_MSPI_DIS	1	Y	2	设置此位以禁用 SYS AXI 矩阵在 boot_mode_download 期间访问 MSPI flash/MSPI RAM。
EFUSE_DIS_TWAI	1	Y	2	设置此位以禁用 TWAI 功能。
EFUSE_JTAG_SEL_ENABLE	1	Y	2	当 EFUSE_DIS_PAD_JTAG 和 EFUSE_DIS_USB_JTAG 都等于 0 时，设置此位启用通过 strapping gpio25 来选择 usb_to_jtag 和 pad_to_jtag。
EFUSE_SOFT_DIS_JTAG	3	Y	31	设置奇数个 1 以软件方式禁用 JTAG。JTAG 可以在 HMAC 模块中启用。
EFUSE_DIS_PAD_JTAG	1	Y	2	设置此位彻底禁用 JTAG。JTAG 将被永久禁用。
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	1	Y	2	设置此位以禁用 flash 手动加密功能（SPI 引导模式除外）。
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6	4	N	N/A	表示当主引导程序失败时，ROM 引导程序使用的恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。
EFUSE_USB_PHY_SEL	1	Y	30	配置两个 PHY 与 USB 接口之间的对应关系。
EFUSE_HUK_GEN_STATE	5	Y	19	设置这些位以控制 HUK 生成模式的验证。当 1 的位数为偶数时该值有效，为奇数时无效。
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7	1	N	N/A	表示当主引导程序失败时，ROM 引导程序使用的恢复引导程序的起始 Flash 扇区（Flash 扇区大小为 0x1000）。

下页继续

GoBack

表 9.4-1 – 续上页

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	功能描述
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10	3	N	N/A	表示当主引导程序失败时，ROM 引导程序使用的恢复引导程序的起始 Flash 扇区（Flash 扇区大小为 0x1000）。
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11	1	N	N/A	表示当主引导程序失败时，ROM 引导程序使用的恢复引导程序的起始 Flash 扇区（Flash 扇区大小为 0x1000）。
EFUSE_KM_RND_SWITCH_CYCLE	1	Y	1	设置这些位以控制密钥管理器随机数切换周期。
EFUSE_KM_DEPLOY_ONLY_ONCE	4	Y	1	此字段与 EFUSE_KM_DEPLOY_ONLY_ONCE_H 一起，组成 EFUSE_KM_DEPLOY_ONLY_ONCE_H，EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]。每个位控制对应的密钥是否只能部署一次。
EFUSE_FORCE_USE_KEY_MANAGER_KEY	4	Y	1	此字段与 EFUSE_FORCE_USE_KEY_MANAGER_KEY_H 一起，组成 EFUSE_FORCE_USE_KEY_MANAGER_KEY_H，EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]。每个位决定对应的密钥是否必须来自密钥管理器。
EFUSE_FORCE_DISABLE_SW_INIT_KEY	1	Y	1	设置此位以禁用软件写入的初始化密钥，并强制使用 efuse_init_key。
EFUSE_KM_XTS_KEY_LENGTH_256	1	N	1	设置此位以选用 XTS-512 做 Flash 加密；否则密钥管理器使用 XTS-256。
EFUSE_ECC_FORCE_CONST_TIME	1	Y	14	设置此位以永久开启 ECC 恒定时间模式。
EFUSE_WDT_DELAY_SEL	1	Y	2	选择 LP WDT 超时阈值。
EFUSE_SPI_BOOT_CRYPT_CNT	3	Y	4	设置此位以启用 SPI 引导加密/解密。
EFUSE_SECURE_BOOT_KEY_REVOKE0	1	N	5	设置此位以启用撤销第一个安全引导密钥。
EFUSE_SECURE_BOOT_KEY_REVOKE1	1	N	6	设置此位以启用撤销第二个安全引导密钥。
EFUSE_SECURE_BOOT_KEY_REVOKE2	1	N	7	设置此位以启用撤销第三个安全引导密钥。
EFUSE_KEY_PURPOSE_0	4	Y	8	Key0 的用途。
EFUSE_KEY_PURPOSE_1	4	Y	9	Key1 的用途。
EFUSE_KEY_PURPOSE_2	4	Y	10	Key2 的用途。

下页继续

GoBack

表 9.4-1 – 续上页

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	功能描述
EFUSE_KEY_PURPOSE_3	4	Y	11	Key3 的用途。
EFUSE_KEY_PURPOSE_4	4	Y	12	Key4 的用途。
EFUSE_KEY_PURPOSE_5	4	Y	13	Key5 的用途。
EFUSE_SEC_DPA_LEVEL	2	Y	14	配置时钟随机分频模式以确定 DPA 安全级别。
EFUSE_XTS_DPA_CLK_ENABLE	1	Y	14	设置此位以启用 xts 时钟抗 DPA 攻击功能。
EFUSE_SECURE_BOOT_EN	1	N	15	设置此位以启用安全引导。
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	1	N	16	设置此位以启用撤销激进安全引导。
EFUSE_KM_DEPLOY_ONLY_ONCE_H	1	Y	1	此字段与 EFUSE_KM_DEPLOY_ONLY_ONCE 一起，组成 EFUSE_KM_DEPLOY_ONLY_ONCE_H, EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]。每个位控制对应的密钥是否只能部署一次。
EFUSE_FORCE_USE_KEY_MANAGER_KEY_H	1	Y	1	EFUSE_FORCE_USE_KEY_MANAGER_KEY 和 EFUSE_FORCE_USE_KEY_MANAGER_KEY 共同组成一个字段：EFUSE_FORCE_USE_KEY_MANAGER_KEY_H, EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]。设置每个位以控制对应的密钥是否必须来自密钥管理器。
EFUSE_FLASH_ECC_EN	1	N	18	设置此位以启用 Flash 引导的 ECC。
EFUSE_DIS_USB_OTG_DOWNLOAD_MODE	1	N	18	设置此位以禁用通过 USB-OTG 下载。
EFUSE_FLASH_TPUW	4	N	18	配置 Flash 上电后的等待时间，单位为 ms。
EFUSE_DIS_DOWNLOAD_MODE	1	N	18	设置此位以禁用下载模式（boot_mode[3:0] = 0, 1, 2, 4, 5, 6, 7）。
EFUSE_DIS_DIRECT_BOOT	1	N	18	设置此位以禁用直接引导模式。
EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT	1	N	18	设置此位以禁用 ROM 引导期间的 USB-Serial-JTAG 打印。
EFUSE_LOCK_KM_KEY	1	N	1	设置此位以在部署后锁定密钥管理器密钥。
EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE	1	N	18	设置此位以禁用 USB-Serial-JTAG 下载功能。
EFUSE_ENABLE_SECURITY_DOWNLOAD	1	N	18	设置此位以启用安全下载模式。

下页继续

GoBack

表 9.4-1 – 续上页

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	功能描述
EFUSE_UART_PRINT_CONTROL	2	N	18	设置 UART 打印的类型。
EFUSE_FORCE_SEND_RESUME	1	N	18	设置此位以强制 ROM 代码在 SPI 引导期间发送 resume 命令。
EFUSE_SECURE_VERSION	16	N	18	ESP-IDF 防回滚功能使用的安全版本。
EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE	1	N	18	表示在使能安全启动时，是否禁用唤醒时的快速验证 (FAST VERIFY ON WAKE) 功能。
EFUSE_HYS_EN_PAD	1	Y	2	设置这些位以启用 PADO 27 的滞回功能。
EFUSE_KEY_PURPOSE_0_H	1	Y	8	Key0 的用途。第 5 位。
EFUSE_KEY_PURPOSE_1_H	1	Y	9	Key1 的用途。第 5 位。
EFUSE_KEY_PURPOSE_2_H	1	Y	10	Key2 的用途。第 5 位。
EFUSE_KEY_PURPOSE_3_H	1	Y	11	Key3 的用途。第 5 位。
EFUSE_KEY_PURPOSE_4_H	1	Y	12	Key4 的用途。第 5 位。
EFUSE_OPXA_TIEH_SEL_0	2	Y	2	输出 LDO VO0 tieh 源选择。
EFUSE_PVT_GLITCH_EN	1	Y	3	表示是否启用 PVT 电源毛刺监控功能。
EFUSE_KEY_PURPOSE_5_H	1	Y	13	Key5 的用途。第 5 位。
EFUSE_KM_DISABLE_DEPLOY_MODE_H	1	Y	1	和 EFUSE_KM_DISABLE_DEPLOY_MODE 共同组成一个字段：EFUSE_KM_DISABLE_DEPLOY_MODE_H, EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]。设置每个位以控制对应的密钥的新值部署的部署模式是否被禁用。
EFUSE_KM_DISABLE_DEPLOY_MODE	4	Y	1	此字段与 EFUSE_KM_DISABLE_DEPLOY_MODE_H 一起，组成一个完整字段 EFUSE_KM_DISABLE_DEPLOY_MODE_H, EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]。每个位控制对应的密钥的新值部署的部署模式是否被禁用。
EFUSE_XTS_DPA_PSEUDO_LEVEL	2	Y	14	设置此位以控制 xts 伪轮抗 dpa 攻击功能。
EFUSE_HP_PWR_SRC_SEL	1	Y	17	HP 系统电源源选择。
EFUSE_SECURE_BOOT_SHA384_EN	1	N	N/A	表示是否启用使用 SHA-384 的安全引导。

下页继续

GoBack

表 9.4-1 – 续上页

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	功能描述
EFUSE_DIS_WDT	1	Y	2	设置此位以禁用看门狗。
EFUSE_DIS_SWD	1	Y	2	设置此位以禁用超级看门狗。
EFUSE_PVT_GLITCH_MODE	2	Y	3	用于配置毛刺模式。

表 9.4-2 为密钥用途各个数值对应的含义。通过配置参数 EFUSE_KEY_PURPOSE_*n* 来声明 KEY*n* 用途 (*n*: 0 ~ 5)。

表 9.4-2. 密钥用途数值对应的含义

密钥用途数值	含义
0	指定为用户使用
1	指定为 ECDSA_KEY 使用
2	指定为 XTS_AES_256_KEY_1 使用
3	指定为 XTS_AES_256_KEY_2 使用
4	指定为 XTS_AES_128_KEY 使用（用于 flash/SRAM 加解密）
5	指定为 HMAC Downstream（下行）模式（JTAG 和数字签名）使用
6	指定为 HMAC Downstream 模式下的 JTAG 使用
7	指定为 HMAC Downstream 模式下的数字签名使用
8	指定为 HMAC Upstream（上行）模式使用
9	指定为 SECURE_BOOT_DIGEST0 使用（secure boot 密钥摘要）
10	指定为 SECURE_BOOT_DIGEST1 使用（secure boot 密钥摘要）
11	指定为 SECURE_BOOT_DIGEST2 使用（secure boot 密钥摘要）
12	指定为 KM_INIT_KEY 使用

表 9.4-3 列出了 BLOCK1 ~ BLOCK10 中存储的参数的信息。

表 9.4-3. BLOCK1~BLOCK10 中的参数

BLOCK	参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	EFUSE_RD_DIS 读取保护位	功能描述
BLOCK1	EFUSE_MAC	48	Y	20	N/A	表示 MAC 地址。低 48 位。
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	表示系统数据第零部分的前 78 位。
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	系统数据
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	用户数据
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	KEY0 或用户数据
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	KEY1 或用户数据
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	KEY2 或用户数据
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	KEY3 或用户数据
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	KEY4 或用户数据
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	KEY5 或用户数据
BLOCK10	EFUSE_SYS_DATA_PART2	192	N	29	6	表示系统数据第二部分的前 32 位。
	EFUSE_USB_DEVICE_EXCHG_PINS	1	Y	29	6	启用 USB-Serial-JTAG 的 D+ 和 D- 引脚交换功能。
	EFUSE_USB_OTG11_EXCHG_PINS	1	Y	29	6	启用 USB OTG11 的 D+ 和 D- 引脚交换功能。

其中，BLOCK4 ~ 9 可用于存储 KEY0 ~ 5，表示 eFuse 中至多可以烧写 6 个 256 位的密钥。每烧写一个密钥，还需要烧写该密钥用途的数值（见表 9.4-2）。例如，用户将用于 HMAC Downstream 模式下的 JTAG 功能的密钥烧写到 KEY3（即 BLOCK7），还需要将密钥用途的数值 6 烧写到 EFUSE_KEY_PURPOSE_3。

说明：

请不要将 XTS-AES 密钥烧写进 KEY5 BLOCK，即 BLOCK9，否则可能发生密钥无法读取的情况。建议首先将 XTS-AES 密钥烧写到前几个 BLOCK 中，即 BLOCK4 ~ BLOCK8，最后一个 BLOCK，即 BLOCK9，用来烧写其他密钥。

BLOCK1 ~ BLOCK10 均采用 RS 编码方式，因此参数烧写受到一定的限制，具体请参考章节 9.4.3.1.3 和章节 9.4.3.2。

GoBack

9.4.3.1.1 EFUSE_WR_DIS

参数 EFUSE_WR_DIS 决定了 eFuse 存储器中所有的参数是否处于烧写保护状态。烧写完 EFUSE_WR_DIS 参数后，须执行一次 eFuse 读操作以使新值生效。

表 9.4-1 以及表 9.4-3 中的“EFUSE_WR_DIS 烧写保护位”列描述了各参数的烧写保护状态具体由 EFUSE_WR_DIS 的哪个位决定。

当某个参数对应的烧写保护位为 0 时，表示此参数未处于烧写保护状态，可以烧写该参数，但已经被烧写的参数不能被重复烧写。

当某个参数对应的烧写保护位为 1 时，表示此参数处于烧写保护状态，此参数的每一个位都无法被更改，未被烧写的位永远为 0，已经被烧写的位永远为 1。因此如果某个参数已经处于烧写保护状态了，则会一直处在该状态，无法再更改。

9.4.3.1.2 EFUSE_RD_DIS

所有参数中，只有 BLOCK4 ~ BLOCK10 的参数可以被配置为用户读取保护状态，即表 9.4-3 中“EFUSE_RD_DIS 读取保护”列非“N/A”的参数。烧写完 EFUSE_RD_DIS 参数后，须执行一次 eFuse 读操作以使新值生效。

参数 EFUSE_RD_DIS 中的某个位为 0，表示此位管理的参数未处于用户读取保护状态；某个位为 1，表示此位管理的参数处于用户读取保护状态。

除 BLOCK4 ~ BLOCK10 之外，其他参数不受读取保护状态的约束，均可被用户读取。

BLOCK4 ~ BLOCK10 即使被配置处于读取保护状态，仍然可以通过设置 EFUSE_KEY_PURPOSE_*n* 供硬件加密模块在内部使用。

9.4.3.1.3 数据存储方式

eFuse 使用硬件编码机制保护数据，编码方案与过程对用户不可见。

BLOCK0 使用 4 备份方式存储参数，即 BLOCK0 中的所有参数（除了 EFUSE_WR_DIS）均在 eFuse 存储器中存储了 4 份。4 备份机制对用户不可见。

BLOCK0 中 EFUSE_WR_DIS 为 32 比特，其余参数为 152 比特，因此 BLOCK0 在 eFuse 存储器中共占据了 $32 + 152 \times 4 = 640$ 比特的存储空间。

BLOCK1 ~ BLOCK10 使用 RS (44, 32) 编码方式，最多支持自动校正 6 个字节。本文 RS (44, 32) 使用的本源多项式为 $p(x) = x^8 + x^4 + x^3 + x^2 + 1$。

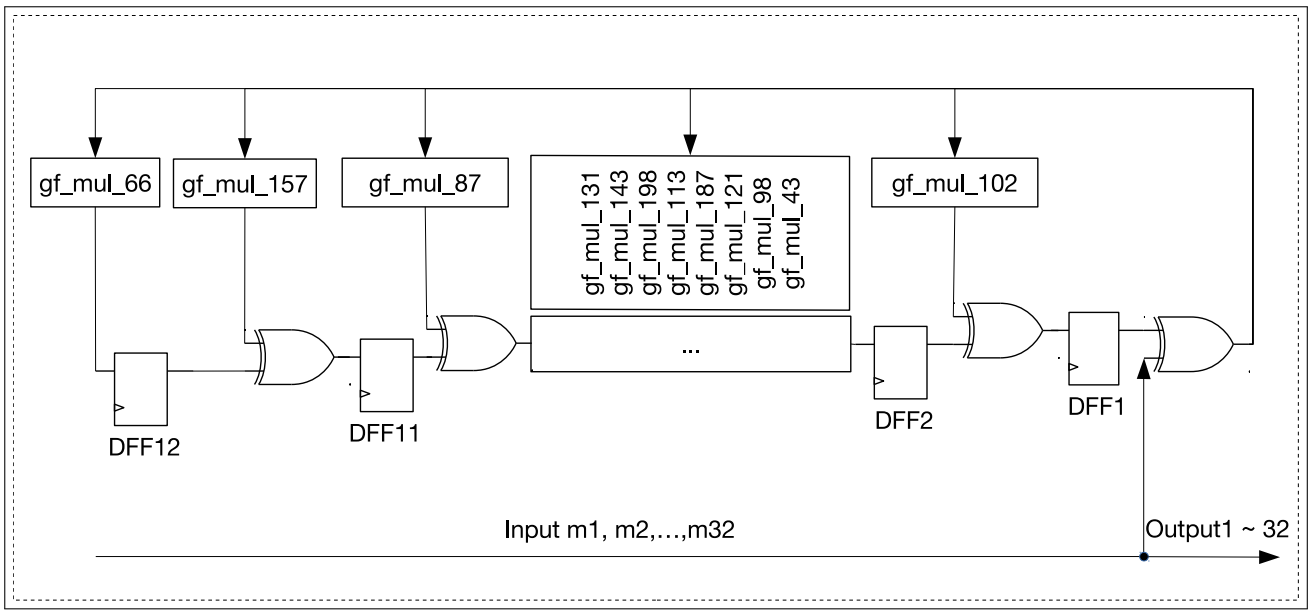


图 9.4-2. 移位寄存器电路图（前 32 字节）

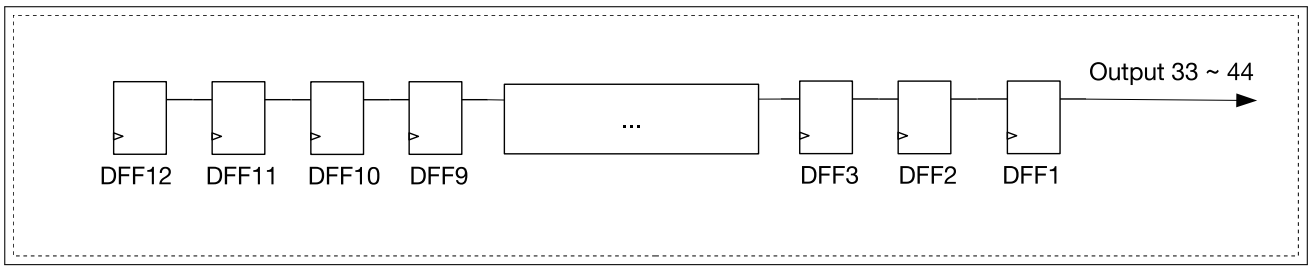


图 9.4-3. 移位寄存器电路图（后 12 字节）

如图 9.4-2 和 9.4-3 所示，移位寄存器电路对 32 字节参数进行 RS (44, 32) 编码处理，将 32 字节数据处理为 44 字节，其中：

- 字节 [0:31] 为数据本身
- 字节 [32:43] 为储存在 8 位触发器 DFF1, DFF2, ..., DFF12 中的奇偶校验字节（gf_mul_n 为 $GF(2^8)$ 域中某一字节数据与元素 α^n 相乘的结果，n 为整数）

然后，硬件将这 44 字节数据一起烧入 eFuse。eFuse 控制器会在读 eFuse 的过程中自动完成解码和自动校正。

由于 RS 校验码是在整个 32 字节的 eFuse 块上生成的，因此每个块只能写入一次。

BLOCK1 由于数据不足 32 字节，因此不足 32 字节的部分在 RS (44, 32) 编码时硬件会将其视为 0，不会影响最终的编码结果。使用 RS (44，32) 编码方式的块中，BLOCK1 数据参数为 24 字节，RS 校验码为 12 字节，因此 BLOCK1 在 eFuse 存储器中共占据了 $24 + 12 = 36$ 字节的存储空间。

其余块 (BLOCK2 ~ BLOCK10) 的数据参数均为 32 字节，RS 校验码为 12 字节，因此在 eFuse 存储器中共占据了 $(32 + 12) \times 9 = 396$ 字节的存储空间。

9.4.3.2 烧写参数

烧写 eFuse 参数时，需要按块烧写。BLOCK0 ~ BLOCK10 共用同一段地址来存储即将烧写的参数。通过配置 EFUSE_BLK_NUM 参数表明当前需要烧写的是哪一个块。

由于读取数据的寄存器和烧写数据的寄存器在位置上存在一一对应的关系（详见表 9.4-4），用户可以通过查找读取寄存器的参数描述和位置确定待烧写数据在烧写寄存器中的具体位置。

例如，用户想将 BLOCK0 中的 EFUSE_USB_DEVICE_EXCHG_PINS 烧写成 1，可以先查找 BLOCK0 读取数据的寄存器 EFUSE_RD_REPEAT_DATA0 ~ 4_REG，定位到 EFUSE_USB_DEVICE_EXCHG_PINS 参数位于 EFUSE_RD_REPEAT_DATA0_REG 寄存器的第 8 个比特，因此用户可以将 EFUSE_PGM_DATA1_REG 待烧写数据的第 8 个比特写成 1，并执行烧写步骤，待烧写步骤完成后，eFuse 存储器中的对应位将被成功烧写为 1。

烧写准备

- 烧写 BLOCK0
  1. 将寄存器域 EFUSE_BLK_NUM 配置为 0。
  2. EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG 配置 BLOCK0 即将烧写的参数。EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG 以及 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中的数据不影响 BLOCK0 的烧写。
- BLOCK1 中数据信息在出厂时已经烧写完毕，不允许用户再次烧写。
- 烧写 BLOCK2 ~ 10
  1. 将寄存器域 EFUSE_BLK_NUM 配置为烧写的块数值。
  2. EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 配置即将烧写的参数，EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中配置对应的 RS 校验码。

烧写流程

烧写参数的流程如下：

1. 配置 EFUSE_BLK_NUM 参数，决定烧写哪一个块。
2. 将需要烧写的参数填写到寄存器 EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 和 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中。
3. 确保 eFuse 烧写电压 VDDQ 的配置正确，具体请参考章节 9.4.3.4。
4. 配置寄存器 EFUSE_CONF_REG 的 EFUSE_OP_CODE 位域为 0x5A5A。
5. 配置寄存器 EFUSE_CMD_REG 的 EFUSE_PGM_CMD 位域为 1。
6. 轮询寄存器 EFUSE_CMD_REG 直到其为 0x0，或者等待 PGM_DONE（烧写完成）中断产生。识别 PGM_DONE 或 READ_DONE （读取完成）中断产生的方法详见章节 9.4.3.3 最后的说明。
7. 将 EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 和 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中写入的参数清零。
8. 执行更新 eFuse 控制器读寄存器操作使写入的新值生效，具体请参考章节 9.4.3.3。
9. 检查错误寄存器内容。若读取错误寄存器内数值不为 0，需要再次执行上述步骤 1 ~ 7 重新烧写一次。

   通过该方式可以解决由于烧写不充分导致错误寄存器内数值不为 0 的问题。

   对于不同的 eFuse 块，需要检查的错误寄存器如下。

   - BLOCK0：EFUSE_RD_REPEAT_DATA_ERR0_REG ~ EFUSE_RD_REPEAT_DATA_ERR4_REG
   - BLOCK1：EFUSE_RD_MAC_SYS_ERR_NUM、EFUSE_RD_MAC_SYS_FAIL

- BLOCK2：EFUSE_RD_SYS_PART1_DATA_ERR_NUM、EFUSE_RD_SYS_PART1_DATA_FAIL
- BLOCK3：EFUSE_RD_USR_DATA_ERR_NUM、EFUSE_RD_USR_DATA_FAIL
- BLOCK4：EFUSE_RD_KEY0_DATA_ERR_NUM、EFUSE_RD_KEY0_DATA_FAIL
- BLOCK5：EFUSE_RD_KEY1_DATA_ERR_NUM、EFUSE_RD_KEY1_DATA_FAIL
- BLOCK6：EFUSE_RD_KEY2_DATA_ERR_NUM、EFUSE_RD_KEY2_DATA_FAIL
- BLOCK7：EFUSE_RD_KEY3_DATA_ERR_NUM、EFUSE_RD_KEY3_DATA_FAIL
- BLOCK8：EFUSE_RD_KEY4_DATA_ERR_NUM、EFUSE_RD_KEY4_DATA_FAIL
- BLOCK9：EFUSE_RD_KEY5_DATA_ERR_NUM、EFUSE_RD_KEY5_DATA_FAIL
- BLOCK10：EFUSE_RD_SYS_PART2_DATA_ERR_NUM、EFUSE_RD_SYS_PART2_DATA_FAIL

限制

BLOCK0 中不同的参数，甚至对于同一个参数中的不同位可以在多次烧写中分别完成。但是并不推荐这样做，而是建议尽量减少烧写次数。我们建议对于某个参数中的所有需要烧写的位都在一次烧写中完成。并且当 EFUSE_WR_DIS 的某个位管理的所有参数都烧写之后，就立即烧写 EFUSE_WR_DIS 的这个位。甚至可以在同一次烧写中既烧写 EFUSE_WR_DIS 的某个位管理的所有参数，同时也烧写 EFUSE_WR_DIS 的这个位。

BLOCK1 中数据信息在出厂时已经烧写完毕，不允许再次烧写。

BLOCK2 ~ 10 中每一个 BLOCK 都只能烧写一次，不允许重复烧写。

9.4.3.3 用户读取参数

用户不能直接读取 eFuse 存储器中烧写的信息内容。eFuse 控制器能够将烧写的数据信息读取到对应的地址段的寄存器内，用户再通过读取以 EFUSE_RD_ 开始的寄存器来获取 eFuse 存储器信息。下表 9.4-4 列出了读取数据的寄存器名称以及对应烧写时的烧写寄存器名称。

表 9.4-4. 用户读取与烧写寄存器

BLOCK	读寄存器	烧写寄存器
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_PART1_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY*n*_DATA0 ~ 7_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_PART2_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

更新 eFuse 控制器的读寄存器

eFuse 控制器通过读取 eFuse 存储器来更新相应寄存器的数据。读取操作在系统复位时进行，也可以根据需要由用户手动触发（例如在需要读取新烧写 eFuse 存储器中的数据内容时）。用户触发 eFuse 存储器读取操作的流程如下：

1. 配置寄存器 EFUSE_CONF_REG 的 EFUSE_OP_CODE 位域为 0x5AA5。
2. 配置寄存器 EFUSE_CMD_REG 的 EFUSE_READ_CMD 位域为 1。

3. 轮询寄存器 EFUSE_CMD_REG 直到其为 0x0，或者等待 READ_DONE 中断产生，识别 PGM_DONE 或 READ_DONE 中断产生的方法详见下方说明。
4. 用户从 eFuse 存储器中读取参数的值。

eFuse 控制器的读寄存器中的数值将一直保持到下一次执行更新 eFuse 存储器读取操作。

烧写错误检测

烧写错误记录寄存器用于检查已存入 eFuse 存储器的参数是否完整。例如：四备份参数是否一致，以及受 RS 编码保护的参数是否成功解码。

EFUSE_RD_REPEAT_ERR0 ~ 3_REG 寄存器用于指示 BLOCK0 中除了 EFUSE_WR_DIS 外的其他参数的烧写是否出错。对应位为 1 代表烧写出错，为 0 代表烧写正确。

EFUSE_RD_RS_ERR0 ~ 1_REG 寄存器记录 eFuse 控制器读 BLOCK1 ~ BLOCK10 过程中，纠错的字节数目以及 RS 解码是否失败的信息。

每次更新 eFuse 控制器读寄存器操作完成之后，上述寄存器内的数值都会被更新。

识别烧写/读取操作完成

识别烧写/读取操作完成的方法如下。位 1 对应烧写操作，位 0 对应读取操作。

- 方法 1：轮询寄存器 EFUSE_INT_RAW_REG 的位 1/0，当值为 1 时，则表示烧写/读取操作完成。
- 方法 2:
  1. 将寄存器 EFUSE_INT_ENA_REG 的位 1/0 置 1，使 eFuse 控制器能够产生 PGM_DONE 或 READ_DONE 中断。
  2. 配置中断矩阵使 CPU 能够响应 eFuse 的中断信号，可参见 13 中断矩阵。
  3. 等待 PGM_DONE 或 READ_DONE 中断产生。
  4. 对寄存器 EFUSE_INT_CLR_REG 的位 1/0 置 1 以清除 PGM_DONE 或 READ_DONE 中断。

注意事项

在 eFuse 控制器执行寄存器更新操作过程中，会复用 EFUSE_PGM_DATA*n*_REG (n=0, 1, ..., 7) 寄存器的存储空间，所以在启动 eFuse 控制器更新寄存器之前，不要将有意义的数据写入上述寄存器中。
芯片启动过程中，eFuse 控制器会自动更新 eFuse 存储器数据到用户可访问的寄存器。用户可以通过读取相应的寄存器获取 eFuse 存储器内烧写的数据。因此，用户无需再驱动 eFuse 控制器执行读更新操作。

9.4.3.4 eFuse VDDQ 时序

eFuse 控制器工作在 20 MHz 时钟频率下，其烧写电压 VDDQ 的配置参数需要满足以下条件：

- EFUSE_DAC_NUM（烧写电压上升周期数），默认烧写电压为 2.5 V，每个上升周期增加 0.01 V，该参数对应的默认值为 255；
- EFUSE_DAC_CLK_DIV（烧写电压时钟分频系数），要求烧写电压时钟周期大于 1 μs；
- EFUSE_PWR_ON_NUM （eFuse 烧写电压上电等待时间），要求该等待时间结束后烧写电压已稳定，即要求配置数值大于 EFUSE_DAC_CLK_DIV ×EFUSE_DAC_NUM；
- EFUSE_PWR_OFF_NUM（烧写电压掉电等待时间），要求该时间大于 10 μs；

表 9.4-5. VDDQ 默认时序参数配置

EFUSE_DAC_NUM	EFUSE_DAC_CLK_DIV	EFUSE_PWR_ON_NUM	EFUSE_PWR_OFF_NUM
0xFF	0x28	0x3000	0x190

9.4.3.5 硬件模块使用参数

硬件模块使用参数是通过电路连接实现的，用户无法干预这个过程。硬件使用的参数为表 9.4-1 和 9.4-3 “硬件使用”一栏中标记为 “Y” 的参数。

9.4.4 中断

ESP32-P4 的 eFuse 控制器可以生成以下中断信号并发送给 中断矩阵。

- EFUSE_INT

以上中断信号由 eFuse 控制器的内部中断源生成。表 9.4-6 列出了 eFuse 控制器的内部中断源、中断触发条件、以及生成的中断信号。

表 9.4-6. eFuse 控制器的内部中断源

内部中断源	触发条件	中断信号
EFUSE_PGM_DONE_INT	eFuse 烧写完成	EFUSE_INT
EFUSE_READ_DONE_INT	eFuse 读取完成	

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 9.4.5 寄存器列表。

9.4.5 寄存器列表

本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问权限
编程数据寄存器			
EFUSE_PGM_DATA0_REG	表示 pgm_data0	0x0000	R/W
EFUSE_PGM_DATA1_REG	表示 pgm_data1	0x0004	R/W
EFUSE_PGM_DATA2_REG	表示 pgm_data2	0x0008	R/W
EFUSE_PGM_DATA3_REG	表示 pgm_data3	0x000C	R/W
EFUSE_PGM_DATA4_REG	表示 pgm_data4	0x0010	R/W
EFUSE_PGM_DATA5_REG	表示 pgm_data5	0x0014	R/W
EFUSE_PGM_DATA6_REG	表示 pgm_data6	0x0018	R/W
EFUSE_PGM_DATA7_REG	表示 pgm_data7	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	表示 pgm_check_value0	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	表示 pgm_check_value1	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	表示 pgm_check_value2	0x0028	R/W
block0 寄存器			
EFUSE_RD_WR_DIS_REG	表示 rd_wr_dis	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	表示 rd_repeat_data	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	表示 rd_repeat_data	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	表示 rd_repeat_data	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	表示 rd_repeat_data	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	表示 rd_repeat_data	0x0040	RO
block1 寄存器			
EFUSE_RD_MAC_SYS0_REG	表示 rd_mac_sys	0x0044	RO
EFUSE_RD_MAC_SYS1_REG	表示 rd_mac_sys	0x0048	RO
EFUSE_RD_MAC_SYS3_REG	表示 rd_mac_sys	0x0050	RO
EFUSE_RD_MAC_SYS4_REG	表示 rd_mac_sys	0x0054	RO
EFUSE_RD_MAC_SYS5_REG	表示 rd_mac_sys	0x0058	RO
block2 寄存器			
EFUSE_RD_SYS_PART1_DATA0_REG	表示 rd_sys_part1_data0	0x005C	RO
EFUSE_RD_SYS_PART1_DATA1_REG	表示 rd_sys_part1_data1	0x0060	RO
EFUSE_RD_SYS_PART1_DATA2_REG	表示 rd_sys_part1_data2	0x0064	RO
EFUSE_RD_SYS_PART1_DATA3_REG	表示 rd_sys_part1_data3	0x0068	RO
EFUSE_RD_SYS_PART1_DATA4_REG	表示 rd_sys_part1_data4	0x006C	RO
EFUSE_RD_SYS_PART1_DATA5_REG	表示 rd_sys_part1_data5	0x0070	RO
EFUSE_RD_SYS_PART1_DATA6_REG	表示 rd_sys_part1_data6	0x0074	RO
EFUSE_RD_SYS_PART1_DATA7_REG	表示 rd_sys_part1_data7	0x0078	RO
block3 寄存器			
EFUSE_RD_USR_DATA0_REG	表示 rd_usr_data0	0x007C	RO
EFUSE_RD_USR_DATA1_REG	表示 rd_usr_data1	0x0080	RO
EFUSE_RD_USR_DATA2_REG	表示 rd_usr_data2	0x0084	RO

名称	描述	地址	访问权限
EFUSE_RD_USR_DATA3_REG	表示 rd_usr_data3	0x0088	RO
EFUSE_RD_USR_DATA4_REG	表示 rd_usr_data4	0x008C	RO
EFUSE_RD_USR_DATA5_REG	表示 rd_usr_data5	0x0090	RO
EFUSE_RD_USR_DATA6_REG	表示 rd_usr_data6	0x0094	RO
EFUSE_RD_USR_DATA7_REG	表示 rd_usr_data7	0x0098	RO
block4 寄存器			
EFUSE_RD_KEY0_DATA0_REG	表示 rd_key0_data0	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	表示 rd_key0_data1	0x00A0	RO
EFUSE_RD_KEY0_DATA2_REG	表示 rd_key0_data2	0x00A4	RO
EFUSE_RD_KEY0_DATA3_REG	表示 rd_key0_data3	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	表示 rd_key0_data4	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	表示 rd_key0_data5	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	表示 rd_key0_data6	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	表示 rd_key0_data7	0x00B8	RO
block5 寄存器			
EFUSE_RD_KEY1_DATA0_REG	表示 rd_key1_data0	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	表示 rd_key1_data1	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	表示 rd_key1_data2	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	表示 rd_key1_data3	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	表示 rd_key1_data4	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	表示 rd_key1_data5	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	表示 rd_key1_data6	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	表示 rd_key1_data7	0x00D8	RO
block6 寄存器			
EFUSE_RD_KEY2_DATA0_REG	表示 rd_key2_data0	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	表示 rd_key2_data1	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	表示 rd_key2_data2	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	表示 rd_key2_data3	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	表示 rd_key2_data4	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	表示 rd_key2_data5	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	表示 rd_key2_data6	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	表示 rd_key2_data7	0x00F8	RO
block7 寄存器			
EFUSE_RD_KEY3_DATA0_REG	表示 rd_key3_data0	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	表示 rd_key3_data1	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	表示 rd_key3_data2	0x0104	RO
EFUSE_RD_KEY3_DATA3_REG	表示 rd_key3_data3	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	表示 rd_key3_data4	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	表示 rd_key3_data5	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	表示 rd_key3_data6	0x0114	RO
EFUSE_RD_KEY3_DATA7_REG	表示 rd_key3_data7	0x0118	RO
block8 寄存器			
EFUSE_RD_KEY4_DATA0_REG	表示 rd_key4_data0	0x011C	RO

名称	描述	地址	访问权限
EFUSE_RD_KEY4_DATA1_REG	表示 rd_key4_data1	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	表示 rd_key4_data2	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	表示 rd_key4_data3	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	表示 rd_key4_data4	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	表示 rd_key4_data5	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	表示 rd_key4_data6	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	表示 rd_key4_data7	0x0138	RO
block9 寄存器			
EFUSE_RD_KEY5_DATA0_REG	表示 rd_key5_data0	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	表示 rd_key5_data1	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	表示 rd_key5_data2	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	表示 rd_key5_data3	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	表示 rd_key5_data4	0x014C	RO
EFUSE_RD_KEY5_DATA5_REG	表示 rd_key5_data5	0x0150	RO
EFUSE_RD_KEY5_DATA6_REG	表示 rd_key5_data6	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	表示 rd_key5_data7	0x0158	RO
block10 寄存器			
EFUSE_RD_SYS_PART2_DATA0_REG	表示 rd_sys_part2_data	0x015C	RO
EFUSE_RD_SYS_PART2_DATA1_REG	表示 rd_sys_part2_data	0x0160	RO
EFUSE_RD_SYS_PART2_DATA2_REG	表示 rd_sys_part2_data	0x0164	RO
EFUSE_RD_SYS_PART2_DATA3_REG	表示 rd_sys_part2_data	0x0168	RO
EFUSE_RD_SYS_PART2_DATA4_REG	表示 rd_sys_part2_data	0x016C	RO
EFUSE_RD_SYS_PART2_DATA5_REG	表示 rd_sys_part2_data	0x0170	RO
EFUSE_RD_SYS_PART2_DATA7_REG	表示 rd_sys_part2_data	0x0178	RO
block0 错误报告寄存器			
EFUSE_RD_REPEAT_DATA_ERR0_REG	表示 rd_repeat_data_err	0x017C	RO
EFUSE_RD_REPEAT_DATA_ERR1_REG	表示 rd_repeat_data_err	0x0180	RO
EFUSE_RD_REPEAT_DATA_ERR2_REG	表示 rd_repeat_data_err	0x0184	RO
EFUSE_RD_REPEAT_DATA_ERR3_REG	表示 rd_repeat_data_err	0x0188	RO
EFUSE_RD_REPEAT_DATA_ERR4_REG	表示 rd_repeat_data_err	0x018C	RO
EFUSE ECDSA 配置寄存器			
EFUSE_ECDSA_REG	eFuse ECDSA 配置寄存器	0x01B0	varies
RS 块错误报告寄存器			
EFUSE_RD_RS_DATA_ERR0_REG	表示 rd_rs_data_err	0x01C0	RO
EFUSE_RD_RS_DATA_ERR1_REG	表示 rd_rs_data_err	0x01C4	RO
EFUSE_CLK_REG	eFuse 时钟配置寄存器	0x01C8	R/W
EFUSE_CONF_REG	eFuse 操作模式配置寄存器	0x01CC	R/W
EFUSE_STATUS_REG	eFuse 状态寄存器	0x01D0	RO
EFUSE_CMD_REG	eFuse 命令寄存器	0x01D4	varies
EFUSE_INT_RAW_REG	eFuse 原始中断寄存器	0x01D8	R/ SS/ WTC
EFUSE_INT_ST_REG	eFuse 中断状态寄存器	0x01DC	RO
EFUSE_INT_ENA_REG	eFuse 中断使能寄存器	0x01E0	R/W

名称	描述	地址	访问权限
EFUSE_INT_CLR_REG	eFuse 中断清除寄存器	0x01E4	WT
EFUSE_DAC_CONF_REG	控制 eFuse 编程电压	0x01E8	R/W
EFUSE_RD_TIM_CONF_REG	配置读时序参数	0x01EC	R/W
EFUSE_WR_TIM_CONF1_REG	eFuse 编程时序参数配置寄存器 1	0x01F0	R/W
EFUSE_WR_TIM_CONF2_REG	eFuse 编程时序参数配置寄存器 2	0x01F4	R/W
EFUSE_WR_TIM_CONF0_RS_BYPASS_REG	eFuse 编程时序参数和 RS 旁路操作配置寄存器 0	0x01F8	varies
EFUSE 版本寄存器			
EFUSE_DATE_REG	eFuse 版本寄存器	0x01FC	R/W
EFUSE_APB2OTP BLOCK0 寄存器			
EFUSE_APB2OTP_WR_DIS_REG	eFuse apb2otp block0 数据寄存器 1	0x0800	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W1_REG	eFuse apb2otp block0 数据寄存器 2	0x0804	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W2_REG	eFuse apb2otp block0 数据寄存器 3	0x0808	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W3_REG	eFuse apb2otp block0 数据寄存器 4	0x080C	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W4_REG	eFuse apb2otp block0 数据寄存器 5	0x0810	RO
EFUSE_APB2OTP_BLK0_BACKUP1_W5_REG	eFuse apb2otp block0 数据寄存器 6	0x0814	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W1_REG	eFuse apb2otp block0 数据寄存器 7	0x0818	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W2_REG	eFuse apb2otp block0 数据寄存器 8	0x081C	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W3_REG	eFuse apb2otp block0 数据寄存器 9	0x0820	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W4_REG	eFuse apb2otp block0 数据寄存器 10	0x0824	RO
EFUSE_APB2OTP_BLK0_BACKUP2_W5_REG	eFuse apb2otp block0 数据寄存器 11	0x0828	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W1_REG	eFuse apb2otp block0 数据寄存器 12	0x082C	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W2_REG	eFuse apb2otp block0 数据寄存器 13	0x0830	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W3_REG	eFuse apb2otp block0 数据寄存器 14	0x0834	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W4_REG	eFuse apb2otp block0 数据寄存器 15	0x0838	RO
EFUSE_APB2OTP_BLK0_BACKUP3_W5_REG	eFuse apb2otp block0 数据寄存器 16	0x083C	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W1_REG	eFuse apb2otp block0 数据寄存器 17	0x0840	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W2_REG	eFuse apb2otp block0 数据寄存器 18	0x0844	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W3_REG	eFuse apb2otp block0 数据寄存器 19	0x0848	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W4_REG	eFuse apb2otp block0 数据寄存器 20	0x084C	RO
EFUSE_APB2OTP_BLK0_BACKUP4_W5_REG	eFuse apb2otp block0 数据寄存器 21	0x0850	RO
EFUSE_APB2OTP BLOCK1 寄存器			
EFUSE_APB2OTP_BLK1_W1_REG	eFuse apb2otp block1 数据寄存器 1	0x0854	RO
EFUSE_APB2OTP_BLK1_W2_REG	eFuse apb2otp block1 数据寄存器 2	0x0858	RO
EFUSE_APB2OTP_BLK1_W3_REG	eFuse apb2otp block1 数据寄存器 3	0x085C	RO
EFUSE_APB2OTP_BLK1_W4_REG	eFuse apb2otp block1 数据寄存器 4	0x0860	RO
EFUSE_APB2OTP_BLK1_W5_REG	eFuse apb2otp block1 数据寄存器 5	0x0864	RO
EFUSE_APB2OTP_BLK1_W6_REG	eFuse apb2otp block1 数据寄存器 6	0x0868	RO
EFUSE_APB2OTP_BLK1_W7_REG	eFuse apb2otp block1 数据寄存器 7	0x086C	RO
EFUSE_APB2OTP_BLK1_W8_REG	eFuse apb2otp block1 数据寄存器 8	0x0870	RO
EFUSE_APB2OTP_BLK1_W9_REG	eFuse apb2otp block1 数据寄存器 9	0x0874	RO
EFUSE_APB2OTP BLOCK2 寄存器			
EFUSE_APB2OTP_BLK2_W1_REG	eFuse apb2otp block2 数据寄存器 1	0x0878	RO

名称	描述	地址	访问权限
EFUSE_APB2OTP_BLK2_W2_REG	eFuse apb2otp block2 数据寄存器 2	0x087C	RO
EFUSE_APB2OTP_BLK2_W3_REG	eFuse apb2otp block2 数据寄存器 3	0x0880	RO
EFUSE_APB2OTP_BLK2_W4_REG	eFuse apb2otp block2 数据寄存器 4	0x0884	RO
EFUSE_APB2OTP_BLK2_W5_REG	eFuse apb2otp block2 数据寄存器 5	0x0888	RO
EFUSE_APB2OTP_BLK2_W6_REG	eFuse apb2otp block2 数据寄存器 6	0x088C	RO
EFUSE_APB2OTP_BLK2_W7_REG	eFuse apb2otp block2 数据寄存器 7	0x0890	RO
EFUSE_APB2OTP_BLK2_W8_REG	eFuse apb2otp block2 数据寄存器 8	0x0894	RO
EFUSE_APB2OTP_BLK2_W9_REG	eFuse apb2otp block2 数据寄存器 9	0x0898	RO
EFUSE_APB2OTP_BLK2_W10_REG	eFuse apb2otp block2 数据寄存器 10	0x089C	RO
EFUSE_APB2OTP_BLK2_W11_REG	eFuse apb2otp block2 数据寄存器 11	0x08A0	RO
EFUSE_APB2OTP BLOCK3 寄存器			
EFUSE_APB2OTP_BLK3_W1_REG	eFuse apb2otp block3 数据寄存器 1	0x08A4	RO
EFUSE_APB2OTP_BLK3_W2_REG	eFuse apb2otp block3 数据寄存器 2	0x08A8	RO
EFUSE_APB2OTP_BLK3_W3_REG	eFuse apb2otp block3 数据寄存器 3	0x08AC	RO
EFUSE_APB2OTP_BLK3_W4_REG	eFuse apb2otp block3 数据寄存器 4	0x08B0	RO
EFUSE_APB2OTP_BLK3_W5_REG	eFuse apb2otp block3 数据寄存器 5	0x08B4	RO
EFUSE_APB2OTP_BLK3_W6_REG	eFuse apb2otp block3 数据寄存器 6	0x08B8	RO
EFUSE_APB2OTP_BLK3_W7_REG	eFuse apb2otp block3 数据寄存器 7	0x08BC	RO
EFUSE_APB2OTP_BLK3_W8_REG	eFuse apb2otp block3 数据寄存器 8	0x08C0	RO
EFUSE_APB2OTP_BLK3_W9_REG	eFuse apb2otp block3 数据寄存器 9	0x08C4	RO
EFUSE_APB2OTP_BLK3_W10_REG	eFuse apb2otp block3 数据寄存器 10	0x08C8	RO
EFUSE_APB2OTP_BLK3_W11_REG	eFuse apb2otp block3 数据寄存器 11	0x08CC	RO
EFUSE_APB2OTP BLOCK4 寄存器			
EFUSE_APB2OTP_BLK4_W1_REG	eFuse apb2otp block4 数据寄存器 1	0x08D0	RO
EFUSE_APB2OTP_BLK4_W2_REG	eFuse apb2otp block4 数据寄存器 2	0x08D4	RO
EFUSE_APB2OTP_BLK4_W3_REG	eFuse apb2otp block4 数据寄存器 3	0x08D8	RO
EFUSE_APB2OTP_BLK4_W4_REG	eFuse apb2otp block4 数据寄存器 4	0x08DC	RO
EFUSE_APB2OTP_BLK4_W5_REG	eFuse apb2otp block4 数据寄存器 5	0x08E0	RO
EFUSE_APB2OTP_BLK4_W6_REG	eFuse apb2otp block4 数据寄存器 6	0x08E4	RO
EFUSE_APB2OTP_BLK4_W7_REG	eFuse apb2otp block4 数据寄存器 7	0x08E8	RO
EFUSE_APB2OTP_BLK4_W8_REG	eFuse apb2otp block4 数据寄存器 8	0x08EC	RO
EFUSE_APB2OTP_BLK4_W9_REG	eFuse apb2otp block4 数据寄存器 9	0x08F0	RO
EFUSE_APB2OTP_BLK4_W10_REG	eFuse apb2otp block4 数据寄存器 10	0x08F4	RO
EFUSE_APB2OTP_BLK4_W11_REG	eFuse apb2otp block4 数据寄存器 11	0x08F8	RO
EFUSE_APB2OTP BLOCK5 寄存器			
EFUSE_APB2OTP_BLK5_W1_REG	eFuse apb2otp block5 数据寄存器 1	0x08FC	RO
EFUSE_APB2OTP_BLK5_W2_REG	eFuse apb2otp block5 数据寄存器 2	0x0900	RO
EFUSE_APB2OTP_BLK5_W3_REG	eFuse apb2otp block5 数据寄存器 3	0x0904	RO
EFUSE_APB2OTP_BLK5_W4_REG	eFuse apb2otp block5 数据寄存器 4	0x0908	RO
EFUSE_APB2OTP_BLK5_W5_REG	eFuse apb2otp block5 数据寄存器 5	0x090C	RO
EFUSE_APB2OTP_BLK5_W6_REG	eFuse apb2otp block5 数据寄存器 6	0x0910	RO
EFUSE_APB2OTP_BLK5_W7_REG	eFuse apb2otp block5 数据寄存器 7	0x0914	RO
EFUSE_APB2OTP_BLK5_W8_REG	eFuse apb2otp block5 数据寄存器 8	0x0918	RO

名称	描述	地址	访问权限
EFUSE_APB2OTP_BLK5_W9_REG	eFuse apb2otp block5 数据寄存器 9	0x091C	RO
EFUSE_APB2OTP_BLK5_W10_REG	eFuse apb2otp block5 数据寄存器 10	0x0920	RO
EFUSE_APB2OTP_BLK5_W11_REG	eFuse apb2otp block5 数据寄存器 11	0x0924	RO
EFUSE_APB2OTP BLOCK6 寄存器			
EFUSE_APB2OTP_BLK6_W1_REG	eFuse apb2otp block6 数据寄存器 1	0x0928	RO
EFUSE_APB2OTP_BLK6_W2_REG	eFuse apb2otp block6 数据寄存器 2	0x092C	RO
EFUSE_APB2OTP_BLK6_W3_REG	eFuse apb2otp block6 数据寄存器 3	0x0930	RO
EFUSE_APB2OTP_BLK6_W4_REG	eFuse apb2otp block6 数据寄存器 4	0x0934	RO
EFUSE_APB2OTP_BLK6_W5_REG	eFuse apb2otp block6 数据寄存器 5	0x0938	RO
EFUSE_APB2OTP_BLK6_W6_REG	eFuse apb2otp block6 数据寄存器 6	0x093C	RO
EFUSE_APB2OTP_BLK6_W7_REG	eFuse apb2otp block6 数据寄存器 7	0x0940	RO
EFUSE_APB2OTP_BLK6_W8_REG	eFuse apb2otp block6 数据寄存器 8	0x0944	RO
EFUSE_APB2OTP_BLK6_W9_REG	eFuse apb2otp block6 数据寄存器 9	0x0948	RO
EFUSE_APB2OTP_BLK6_W10_REG	eFuse apb2otp block6 数据寄存器 10	0x094C	RO
EFUSE_APB2OTP_BLK6_W11_REG	eFuse apb2otp block6 数据寄存器 11	0x0950	RO
EFUSE_APB2OTP BLOCK7 寄存器			
EFUSE_APB2OTP_BLK7_W1_REG	eFuse apb2otp block7 数据寄存器 1	0x0954	RO
EFUSE_APB2OTP_BLK7_W2_REG	eFuse apb2otp block7 数据寄存器 2	0x0958	RO
EFUSE_APB2OTP_BLK7_W3_REG	eFuse apb2otp block7 数据寄存器 3	0x095C	RO
EFUSE_APB2OTP_BLK7_W4_REG	eFuse apb2otp block7 数据寄存器 4	0x0960	RO
EFUSE_APB2OTP_BLK7_W5_REG	eFuse apb2otp block7 数据寄存器 5	0x0964	RO
EFUSE_APB2OTP_BLK7_W6_REG	eFuse apb2otp block7 数据寄存器 6	0x0968	RO
EFUSE_APB2OTP_BLK7_W7_REG	eFuse apb2otp block7 数据寄存器 7	0x096C	RO
EFUSE_APB2OTP_BLK7_W8_REG	eFuse apb2otp block7 数据寄存器 8	0x0970	RO
EFUSE_APB2OTP_BLK7_W9_REG	eFuse apb2otp block7 数据寄存器 9	0x0974	RO
EFUSE_APB2OTP_BLK7_W10_REG	eFuse apb2otp block7 数据寄存器 10	0x0978	RO
EFUSE_APB2OTP_BLK7_W11_REG	eFuse apb2otp block7 数据寄存器 11	0x097C	RO
EFUSE_APB2OTP BLOCK8 寄存器			
EFUSE_APB2OTP_BLK8_W1_REG	eFuse apb2otp block8 数据寄存器 1	0x0980	RO
EFUSE_APB2OTP_BLK8_W2_REG	eFuse apb2otp block8 数据寄存器 2	0x0984	RO
EFUSE_APB2OTP_BLK8_W3_REG	eFuse apb2otp block8 数据寄存器 3	0x0988	RO
EFUSE_APB2OTP_BLK8_W4_REG	eFuse apb2otp block8 数据寄存器 4	0x098C	RO
EFUSE_APB2OTP_BLK8_W5_REG	eFuse apb2otp block8 数据寄存器 5	0x0990	RO
EFUSE_APB2OTP_BLK8_W6_REG	eFuse apb2otp block8 数据寄存器 6	0x0994	RO
EFUSE_APB2OTP_BLK8_W7_REG	eFuse apb2otp block8 数据寄存器 7	0x0998	RO
EFUSE_APB2OTP_BLK8_W8_REG	eFuse apb2otp block8 数据寄存器 8	0x099C	RO
EFUSE_APB2OTP_BLK8_W9_REG	eFuse apb2otp block8 数据寄存器 9	0x09A0	RO
EFUSE_APB2OTP_BLK8_W10_REG	eFuse apb2otp block8 数据寄存器 10	0x09A4	RO
EFUSE_APB2OTP_BLK8_W11_REG	eFuse apb2otp block8 数据寄存器 11	0x09A8	RO
EFUSE_APB2OTP BLOCK9 寄存器			
EFUSE_APB2OTP_BLK9_W1_REG	eFuse apb2otp block9 数据寄存器 1	0x09AC	RO
EFUSE_APB2OTP_BLK9_W2_REG	eFuse apb2otp block9 数据寄存器 2	0x09B0	RO
EFUSE_APB2OTP_BLK9_W3_REG	eFuse apb2otp block9 数据寄存器 3	0x09B4	RO

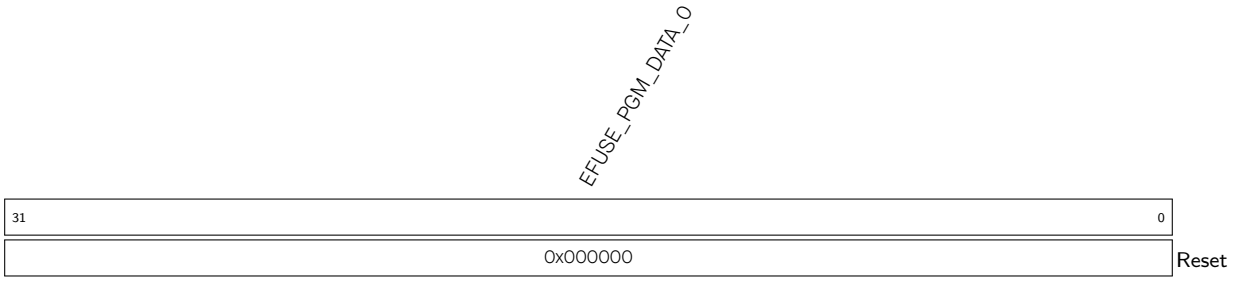
名称	描述	地址	访问权限
EFUSE_APB2OTP_BLK9_W4_REG	eFuse apb2otp block9 数据寄存器 4	0x09B8	RO
EFUSE_APB2OTP_BLK9_W5_REG	eFuse apb2otp block9 数据寄存器 5	0x09BC	RO
EFUSE_APB2OTP_BLK9_W6_REG	eFuse apb2otp block9 数据寄存器 6	0x09C0	RO
EFUSE_APB2OTP_BLK9_W7_REG	eFuse apb2otp block9 数据寄存器 7	0x09C4	RO
EFUSE_APB2OTP_BLK9_W8_REG	eFuse apb2otp block9 数据寄存器 8	0x09C8	RO
EFUSE_APB2OTP_BLK9_W9_REG	eFuse apb2otp block9 数据寄存器 9	0x09CC	RO
EFUSE_APB2OTP_BLK9_W10_REG	eFuse apb2otp block9 数据寄存器 10	0x09D0	RO
EFUSE_APB2OTP_BLK9_W11_REG	eFuse apb2otp block9 数据寄存器 11	0x09D4	RO
EFUSE_APB2OTP BLOCK10 寄存器			
EFUSE_APB2OTP_BLK10_W1_REG	eFuse apb2otp block10 数据寄存器 1	0x09D8	RO
EFUSE_APB2OTP_BLK10_W2_REG	eFuse apb2otp block10 数据寄存器 2	0x09DC	RO
EFUSE_APB2OTP_BLK10_W3_REG	eFuse apb2otp block10 数据寄存器 3	0x09E0	RO
EFUSE_APB2OTP_BLK10_W4_REG	eFuse apb2otp block10 数据寄存器 4	0x09E4	RO
EFUSE_APB2OTP_BLK10_W5_REG	eFuse apb2otp block10 数据寄存器 5	0x09E8	RO
EFUSE_APB2OTP_BLK10_W6_REG	eFuse apb2otp block10 数据寄存器 6	0x09EC	RO
EFUSE_APB2OTP_BLK10_W7_REG	eFuse apb2otp block10 数据寄存器 7	0x09F0	RO
EFUSE_APB2OTP_BLK10_W8_REG	eFuse apb2otp block10 数据寄存器 8	0x09F4	RO
EFUSE_APB2OTP_BLK10_W9_REG	eFuse apb2otp block10 数据寄存器 9	0x09F8	RO
EFUSE_APB2OTP_BLK10_W10_REG	eFuse apb2otp block10 数据寄存器 10	0x09FC	RO
EFUSE_APB2OTP_BLK10_W11_REG	eFuse apb2otp block10 数据寄存器 11	0x0A00	RO
EFUSE_APB2OTP 使能寄存器			
EFUSE_APB2OTP_EN_REG	eFuse apb2otp 使能配置寄存器	0x0A08	R/W

9.4.6 寄存器

本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

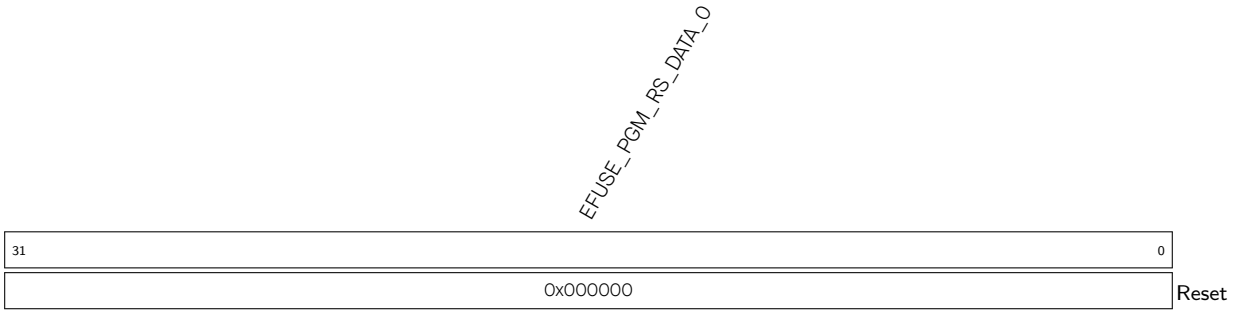
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 9.1. EFUSE_PGM_DATA*n*_REG (*n*: 0-7) (0x0000+0x4**n*)



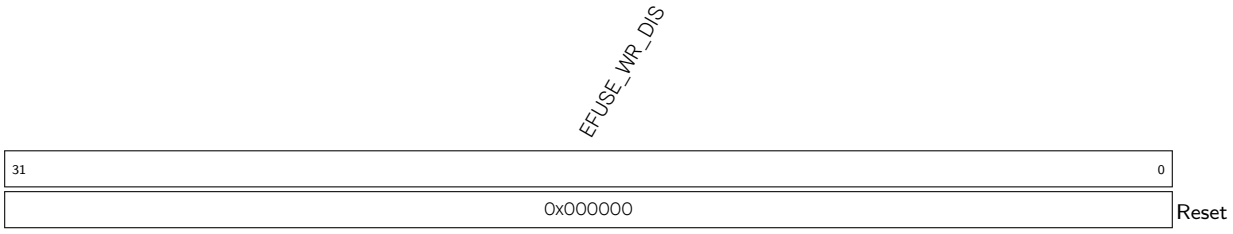
EFUSE_PGM_DATA_*n* 配置待烧写数据的第 *n* 个 32 位数据。(R/W)

Register 9.2. EFUSE_PGM_CHECK_VALUE*n*_REG (*n*: 0-2) (0x0020+0x4**n*)



EFUSE_PGM_RS_DATA_*n* 配置待烧写的第 *n* 个 RS 码。(R/W)

Register 9.3. EFUSE_RD_WR_DIS_REG (0x002C)

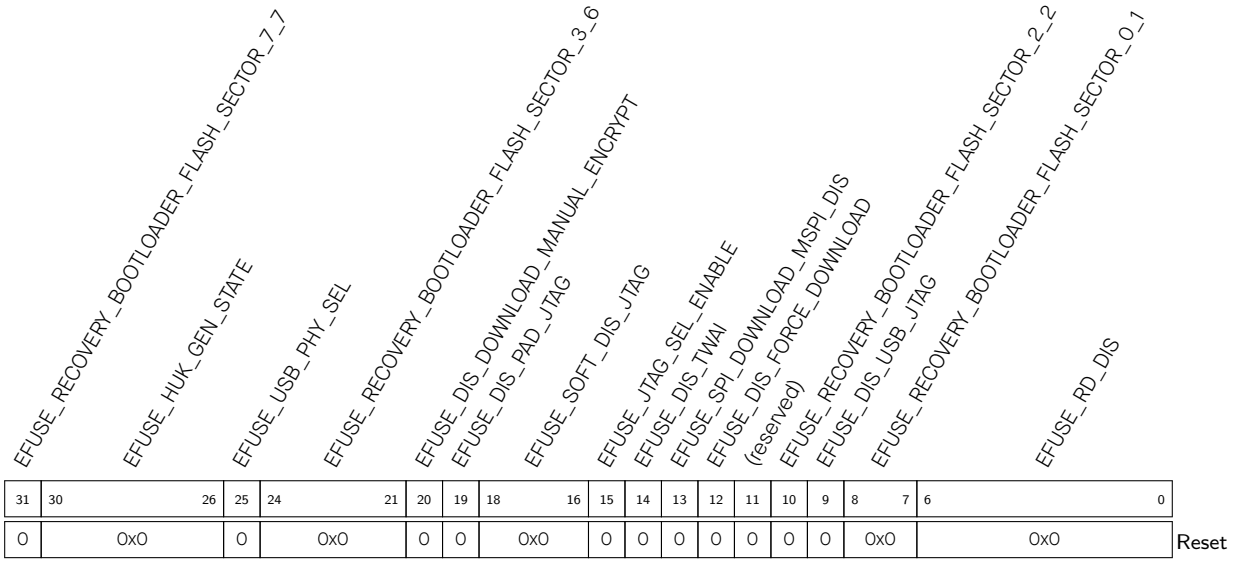


EFUSE_WR_DIS 表示是否禁用 eFuse 存储器各个位的烧写。关于该字段位与 eFuse 存储器位的映射关系，请参考表 9.4-1 和表 9.4-3。(RO)

1: 禁用

0: 使能

Register 9.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS 表示是否禁用 eFuse 各个 block (block4 block10) 的读取。(RO)

1: 禁用

0: 使能

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1 表示恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。当主引导程序失败时，ROM 引导程序从此扇区加载并执行恢复引导程序。(RO)

0 和 0xFFF：此功能被禁用

EFUSE_DIS_USB_JTAG 表示是否禁用 USB-Serial-JTAG 模块中 USB 转 JTAG 功能。(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2 表示恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。当主引导程序失败时，ROM 引导程序从此扇区加载并执行恢复引导程序。(RO)

0 和 0xFFF：此功能被禁用

EFUSE_DIS_FORCE_DOWNLOAD 表示是否禁用强制芯片进入 Download 模式的功能。(RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS 表示在 boot_mode_download 过程中是否关闭 SYS AXI 对 MSPI flash/MSPI RAM 的访问。(RO)

EFUSE_DIS_TWAI 表示是否禁用 TWAI 功能。(RO)

EFUSE_JTAG_SEL_ENABLE 表示当 EFUSE_DIS_PAD_JTAG 和 EFUSE_DIS_USB_JTAG 均配置为 0 时，是否允许通过 strapping 管脚 GPIO25 选择 JTAG 信号源（usb_to_jtag 或 pad_to_jtag）。(RO)

EFUSE_SOFT_DIS_JTAG 表示是否软禁用 PAD JTAG。可通过 HMAC 重新启动。(RO)

奇数个比特为 1：禁用

偶数个比特为 1：使能

EFUSE_DIS_PAD_JTAG 表示是否硬禁用 PAD JTAG（永久）。(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 表示是否禁用 flash 手动加密功能（SPI 启动模式除外）。(RO)

见下页...

Register 9.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

接上页...

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6 表示恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。当主引导程序失败时，ROM 引导程序从此扇区加载并执行恢复引导程序。(RO)

0 和 0xFFF：此功能被禁用

EFUSE_USB_PHY_SEL 配置两个 PHY 与 USB 接口的对应关系。(RO)

0: GPIO24/GPIO25 上的 PHY 连接到 usb_device，而 GPIO26/GPIO27 上的 PHY 连接到 usb_otg11

1: GPIO26/GPIO27 上的 PHY 连接到 usb_device，而 GPIO24/GPIO25 上的 PHY 连接到 usb_otg11

EFUSE_HUK_GEN_STATE 表示 HUK 生成模式的验证状态。(RO)

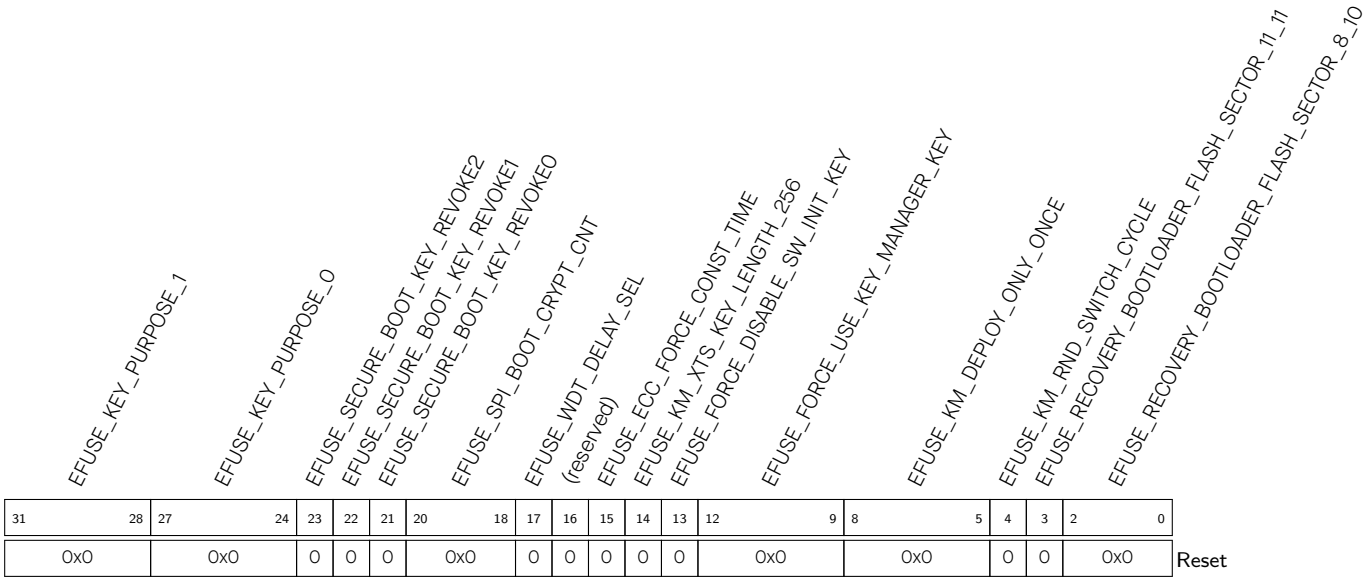
值为 1 的位数为奇数：无效

值为 1 的位数为偶数：有效

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7 表示恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。当主引导程序失败时，ROM 引导程序从此扇区加载并执行恢复引导程序。(RO)

0 和 0xFFF：恢复引导程序功能被禁用

Register 9.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)



EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10 表示恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。当主引导程序失败时，ROM 引导程序从此扇区加载并执行恢复引导程序。(RO)

0 和 0xFFF：恢复引导程序功能被禁用

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11 表示恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。当主引导程序失败时，ROM 引导程序从此扇区加载并执行恢复引导程序。(RO)

0 和 0xFFF：恢复引导程序功能被禁用

EFUSE_KM_RND_SWITCH_CYCLE 表示密钥管理器随机数切换周期。(RO)

0: 由寄存器控制

1: 8 个 km clk 周期

2: 16 个 km 周期

3: 32 个 km 周期

EFUSE_KM_DEPLOY_ONLY_ONCE EFUSE_KM_DEPLOY_ONLY_ONCE 和 EFUSE_KM_DEPLOY_ONLY_ONCE_H 共同组成一个字段：EFUSE_KM_DEPLOY_ONLY_ONCE_H, EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]。表示每个对应密钥是否只能部署一次。(RO)

1: 是

0: 否

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

bit 4: psram

见下页...

Register 9.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)

接上页...

EFUSE_FORCE_USE_KEY_MANAGER_KEY EFUSE_FORCE_USE_KEY_MANAGER_KEY 和 EFUSE_FORCE_USE_KEY_MANAGER_KEY_H 共同组成一个字段: EFUSE_FORCE_USE_KEY_MANAGER_KEY_H, EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]。表示每个对应密钥是否必须来自密钥管理器。(RO)
1: 是
0: 否
bit 0: ecsda
bit 1: xts
bit 2: hmac
bit 3: ds
bit 4: psram

EFUSE_FORCE_DISABLE_SW_INIT_KEY 表示是否禁用软件写入的初始化密钥，并强制使用 efuse_init_key。(RO)

EFUSE_KM_XTS_KEY_LENGTH_256 表示 flash 加密是否使用 XTS-512 密钥。否则，密钥管理器将使用 XTS-256 密钥。(RO)

EFUSE_ECC_FORCE_CONST_TIME 表示 ECC 常量时间模式是否永久开启。(RO)

EFUSE_WDT_DELAY_SEL 表示 LP WDT 超时阈值选择，计算公式为:

$$\text{startup_timeout} = \text{initial_timeout_value} \times 2^{(\text{EFUSE_WDT_DELAY_SEL}+1)}$$

(RO)

EFUSE_SPI_BOOT_CRYPT_CNT 表示是否使能 SPI boot 加解密。(RO)
奇数个比特为 1：使能
偶数个比特为 1：不使能

EFUSE_SECURE_BOOT_KEY_REVOKE0 表示是否撤销安全启动秘钥 0。(RO)

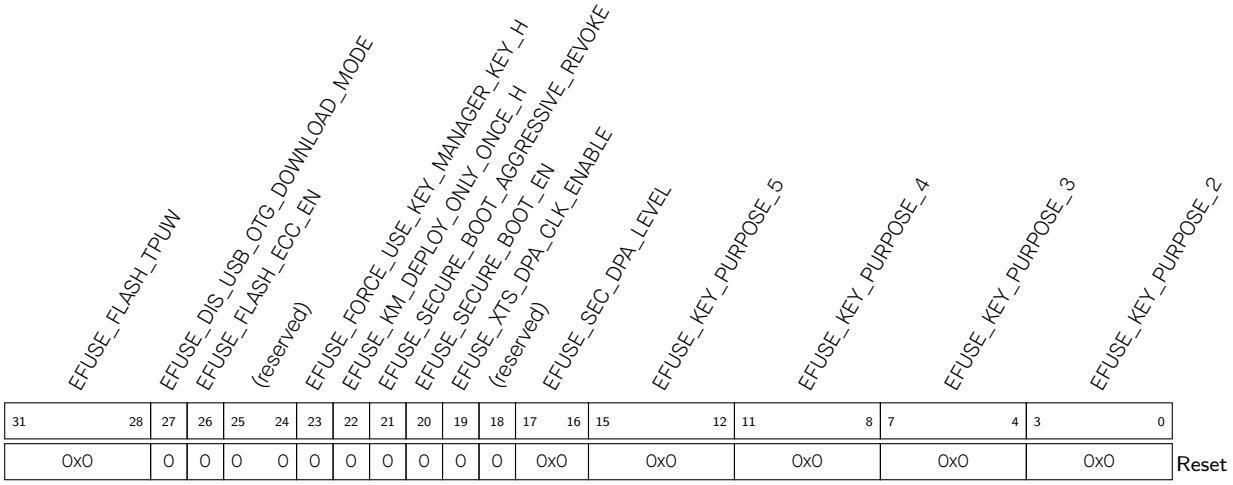
EFUSE_SECURE_BOOT_KEY_REVOKE1 表示是否撤销安全启动秘钥 1。(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 表示是否撤销安全启动秘钥 2。(RO)

EFUSE_KEY_PURPOSE_0 表示 Key0 用途，更多信息请见表 9.4-2。(RO)

EFUSE_KEY_PURPOSE_1 表示 Key1 用途，更多信息请见表 9.4-2。(RO)

Register 9.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 表示 Key2 用途，更多信息请见表 9.4-2。(RO)

EFUSE_KEY_PURPOSE_3 表示 Key3 用途，更多信息请见表 9.4-2。(RO)

EFUSE_KEY_PURPOSE_4 表示 Key4 用途，更多信息请见表 9.4-2。(RO)

EFUSE_KEY_PURPOSE_5 表示 Key5 用途，更多信息请见表 9.4-2。(RO)

EFUSE_SEC_DPA_LEVEL 表示防 DPA 攻击的安全级别。该寄存器域实际上通过配置时钟随机分频模式来调整防 DPA 攻击的安全等级。(RO)

EFUSE_XTS_DPA_CLK_ENABLE 表示 XTS 时钟防 DPA 攻击功能是否使能。(RO)

EFUSE_SECURE_BOOT_EN 表示是否使能安全启动。(RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE 表示是否使能安全启动的激进撤销。(RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_H EFUSE_KM_DEPLOY_ONLY_ONCE 和 EFUSE_KM_DEPLOY_ONLY_ONCE_H 共同组成一个字段：EFUSE_KM_DEPLOY_ONLY_ONCE_H & EFUSE_KM_DEPLOY_ONLY_ONCE[3:0]。表示每个对应密钥是否只能部署一次。(RO)

1: 是

0: 否

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

bit 4: psram

见下页...

Register 9.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)

接上页...

EFUSE_FORCE_USE_KEY_MANAGER_KEY_H EFUSE_FORCE_USE_KEY_MANAGER_KEY 和 EFUSE_FORCE_USE_KEY_MANAGER_KEY_H 共同组成一个字段：EFUSE_FORCE_USE_KEY_MANAGER_KEY_H & EFUSE_FORCE_USE_KEY_MANAGER_KEY[3:0]。表示每个对应密钥是否必须来自密钥管理器。(RO)

1: 是

0: 否

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

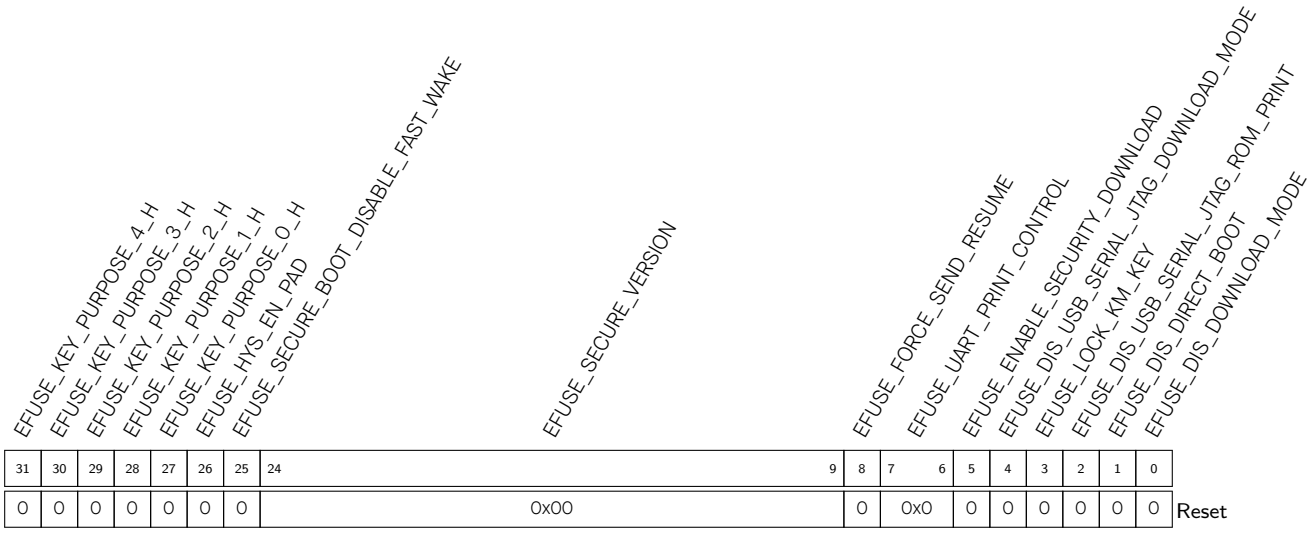
bit 4: psram

EFUSE_FLASH_ECC_EN 表示 flash boot 期间是否使能 ECC。(RO)

EFUSE_DIS_USB_OTG_DOWNLOAD_MODE 表示是否禁用通过 USB-OTG 下载。(RO)

EFUSE_FLASH_TPUW 表示 flash 上电等待时间。单位：ms。当值小于 15 时，等待时间为配置值。当值大于等于 15 时，等待时间固定为 30 ms。(RO)

Register 9.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



EFUSE_DIS_DOWNLOAD_MODE 表示是否禁用所有 Download 模式。(RO)

EFUSE_DIS_DIRECT_BOOT 表示是否禁用 direct boot 模式。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 表示是否禁用 ROM boot 过程中的 USB-Serial-JTAG 打印。(RO)

EFUSE_LOCK_KM_KEY 表示密钥管理器密钥在部署后是否被锁定。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE 表示是否禁用 USB-Serial-JTAG 下载功能。(RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD 表示是否使能安全下载。(RO)

EFUSE_UART_PRINT_CONTROL 表示 UART 打印类型。(RO)

00: 强制打印
01: GPIO8 低电平复位时使能打印
10: GPIO8 高电平复位时使能打印
11: 强制关闭打印

EFUSE_FORCE_SEND_RESUME 表示是否强制 ROM 代码在 SPI 启动过程中发送恢复指令。(RO)

EFUSE_SECURE_VERSION 表示安全版本，用于 ESP-IDF 防回滚功能。(RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE 表示安全启动使能时是否禁用 FAST VERIFY ON WAKE。(RO)

1: 禁用
0: 使能

见下页...

Register 9.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)

接上页...

EFUSE_HYS_EN_PAD 表示是否使能 PAD0 27 的迟滞功能。(RO)

EFUSE_KEY_PURPOSE_0_H 表示 Key0 用途。第 5 位。(RO)

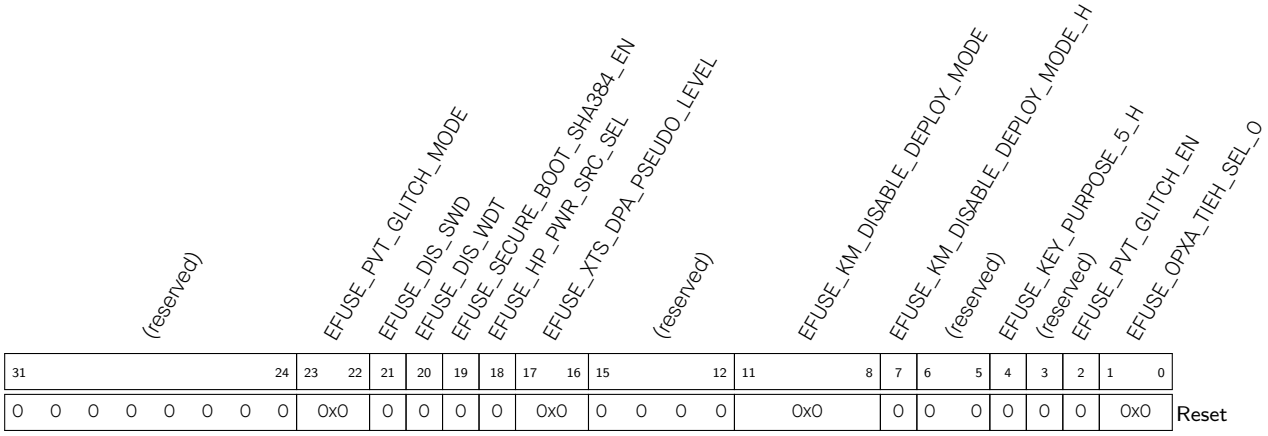
EFUSE_KEY_PURPOSE_1_H 表示 Key1 用途。第 5 位。(RO)

EFUSE_KEY_PURPOSE_2_H 表示 Key2 用途。第 5 位。(RO)

EFUSE_KEY_PURPOSE_3_H 表示 Key3 用途。第 5 位。(RO)

EFUSE_KEY_PURPOSE_4_H 表示 Key4 用途。第 5 位。(RO)

Register 9.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



EFUSE_OPXA_TIEH_SEL_0 表示 LDO VOO 供电源头的控制选择。(RO)

0: 供电一直拉高

1: 由 SDMMC1 外设控制

2: 由寄存器域控制

3: 由 SDMMC0 外设控制

EFUSE_PVT_GLITCH_EN 表示是否使能 PVT 电源毛刺监控功能。(RO)

1: 使能

0: 禁用

EFUSE_KEY_PURPOSE_5_H 表示 Key5 用途。第 5 位。(RO)

EFUSE_KM_DISABLE_DEPLOY_MODE_H EFUSE_KM_DISABLE_DEPLOY_MODE 和 EFUSE_KM_-DISABLE_DEPLOY_MODE_H 共同组成一个字段：EFUSE_KM_DISABLE_DEPLOY_MODE_H, EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]。表示每个对应密钥的新值部署模式是否被禁用。(RO)

1: 是

0: 否

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

bit 4: psram

EFUSE_KM_DISABLE_DEPLOY_MODE EFUSE_KM_DISABLE_DEPLOY_MODE 和 EFUSE_KM_-DISABLE_DEPLOY_MODE_H 共同组成一个字段：EFUSE_KM_DISABLE_DEPLOY_MODE_H, EFUSE_KM_DISABLE_DEPLOY_MODE[3:0]。表示每个对应密钥的新值部署模式是否被禁用。(RO)

1: 是

0: 否

bit 0: ecsda

bit 1: xts

bit 2: hmac

bit 3: ds

bit 4: psram

见下页...

Register 9.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)

接上页...

EFUSE_XTS_DPA_PSEUDO_LEVEL 表示 XTS 伪轮防 DPA 攻击功能控制。(RO)

0: 由寄存器控制

1-3: 值越大，在 XTS-AES 计算中插入的伪轮数越多

EFUSE_HP_PWR_SRC_SEL 表示 HP 系统电源来源。(RO)

0: LDO

1: DCDC

EFUSE_SECURE_BOOT_SHA384_EN 表示是否使能使用 SHA-384 的安全启动。(RO)

0: 禁用

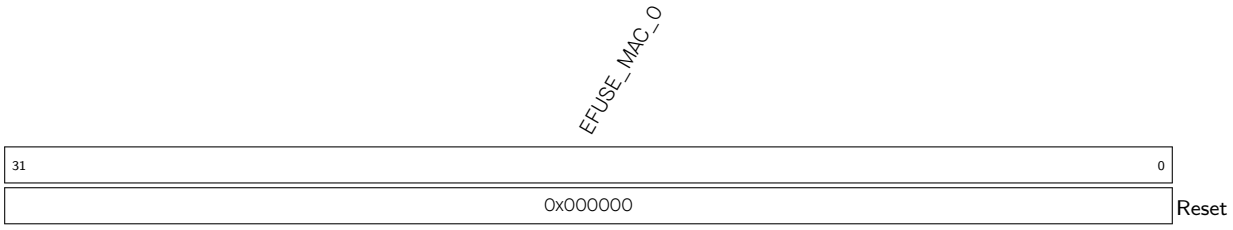
1: 使能

EFUSE_DIS_WDT 表示是否禁用看门狗。(RO)

EFUSE_DIS_SWD 表示是否禁用超级看门狗。(RO)

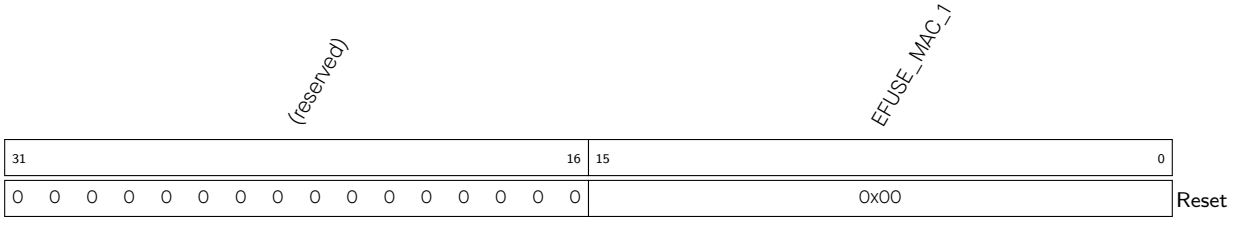
EFUSE_PVT_GLITCH_MODE 配置 PVT 毛刺处理模式。(RO)

Register 9.9. EFUSE_RD_MAC_SYS0_REG (0x0044)



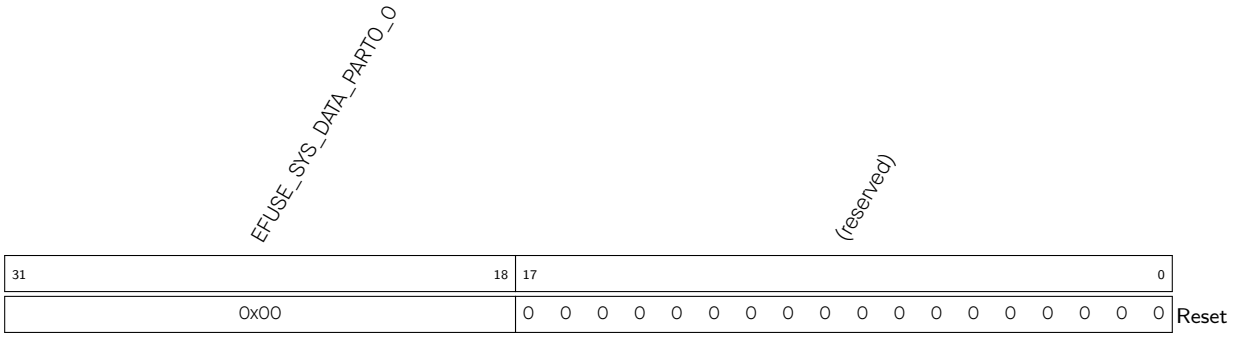
EFUSE_MAC_0 表示 MAC 地址低 32 位。(RO)

Register 9.10. EFUSE_RD_MAC_SYS1_REG (0x0048)



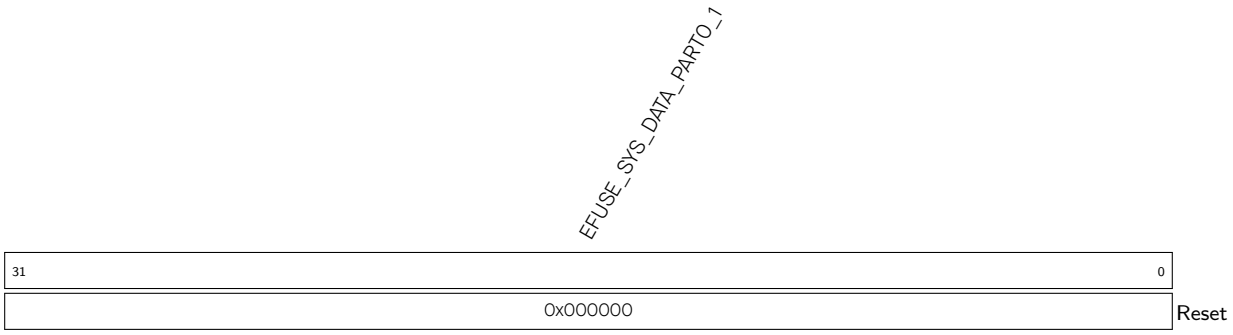
EFUSE_MAC_1 表示 MAC 地址高 16 位。(RO)

Register 9.11. EFUSE_RD_MAC_SYS3_REG (0x0050)



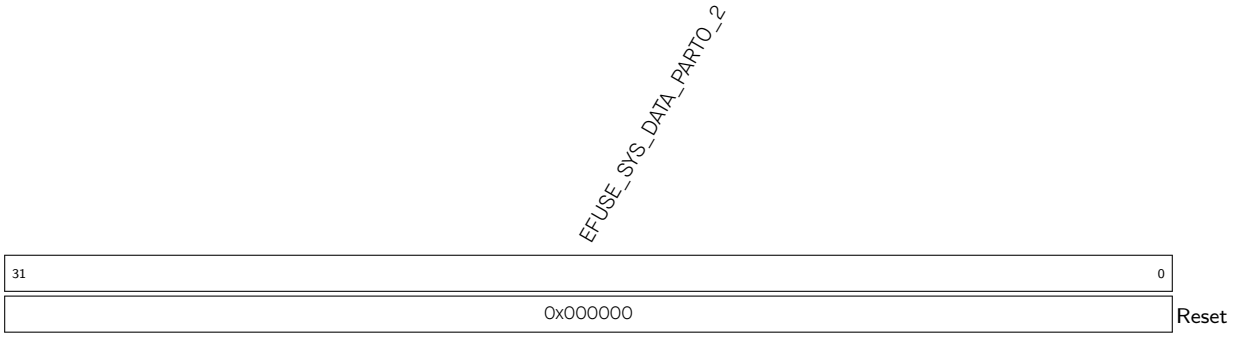
EFUSE_SYS_DATA_PART0_0 表示系统数据第 0 部分的第 1 个 14 位内容。(RO)

Register 9.12. EFUSE_RD_MAC_SYS4_REG (0x0054)



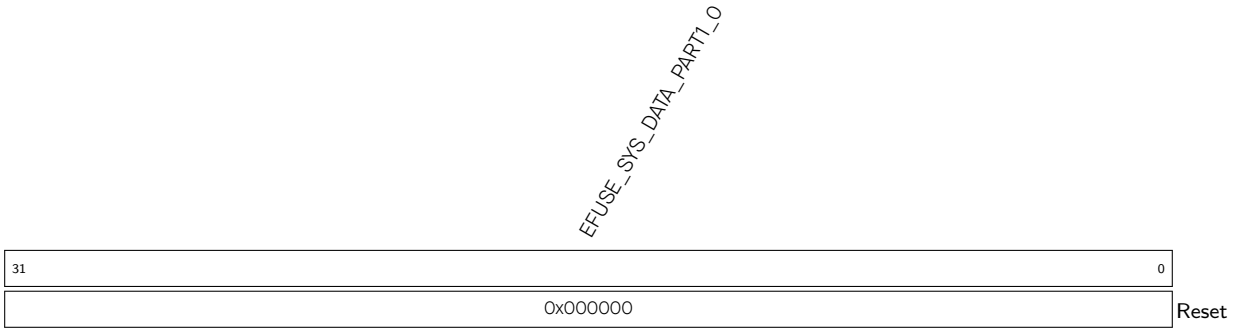
EFUSE_SYS_DATA_PART0_1 表示系统数据第 0 部分的第 1 个 32 位内容。(RO)

Register 9.13. EFUSE_RD_MAC_SYS5_REG (0x0058)



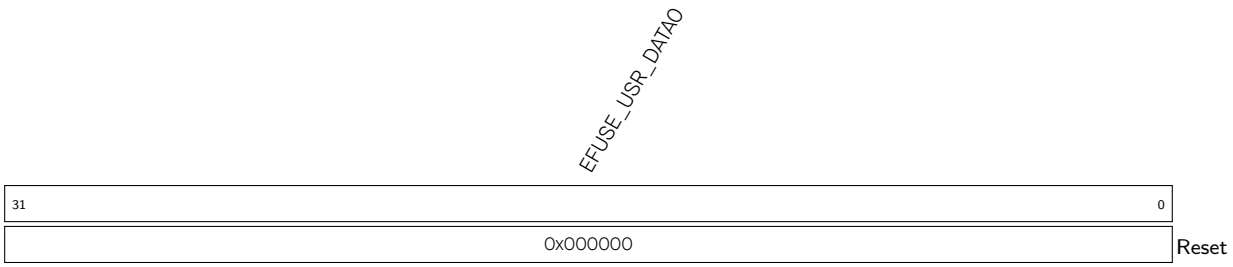
EFUSE_SYS_DATA_PART0_2 表示系统数据第 0 部分的第 2 个 32 位内容。(RO)

Register 9.14. EFUSE_RD_SYS_PART1_DATA*n*_REG (*n*: 0-7) (0x005C+0x4**n*)



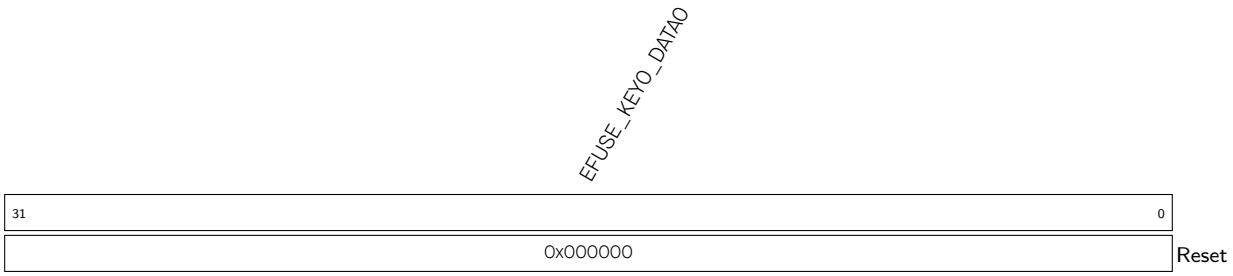
EFUSE_SYS_DATA_PART1_*n* 表示系统数据第 1 部分的第 *n* 个 32 位内容。(RO)

Register 9.15. EFUSE_RD_USR_DATA*n*_REG (*n*: 0-7) (0x007C+0x4**n*)



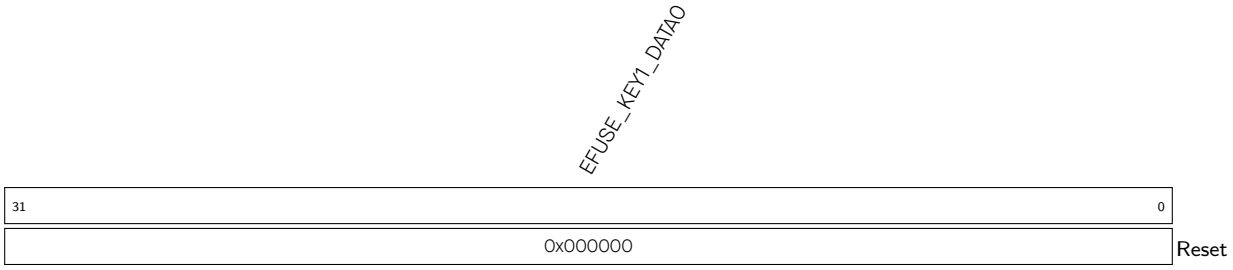
EFUSE_USR_DATA*n* 表示 BLOCK3 (user) 第 *n* 个 32 位内容。(RO)

Register 9.16. EFUSE_RD_KEY0_DATA*n*_REG (*n*: 0-7) (0x009C+0x4**n*)



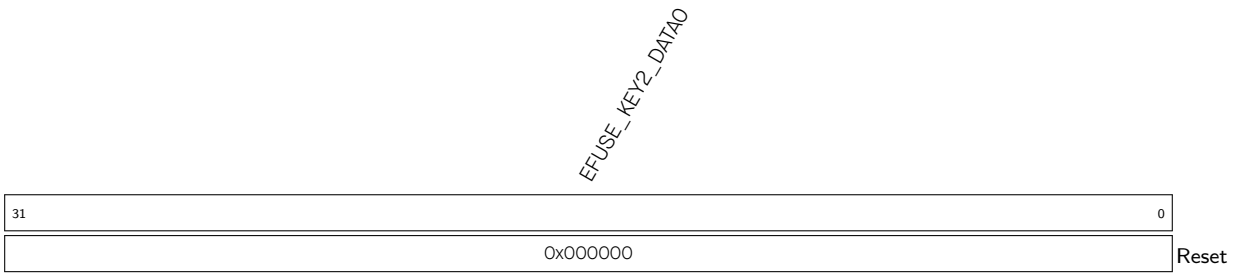
EFUSE_KEY0_DATA*n* 表示 KEY0 第 *n* 个 32 位内容。(RO)

Register 9.17. EFUSE_RD_KEY1_DATA*n*_REG (*n*: 0-7) (0x00BC+0x4*n*)



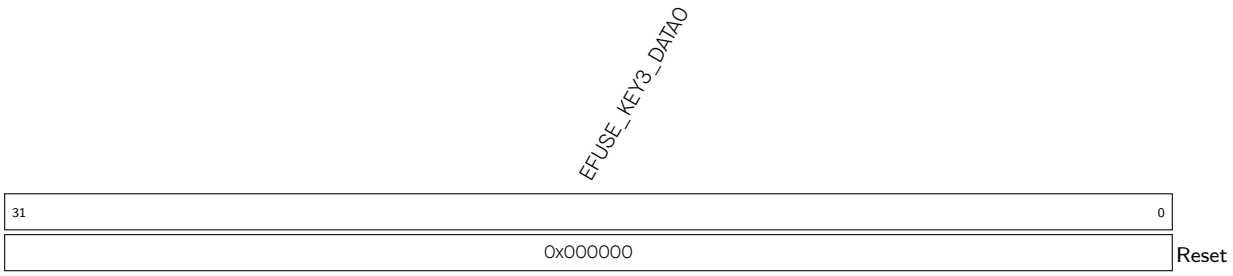
EFUSE_KEY1_DATA*n* 表示 KEY1 第 *n* 个 32 位内容。(RO)

Register 9.18. EFUSE_RD_KEY2_DATA*n*_REG (*n*: 0-7) (0x00DC+0x4*n*)



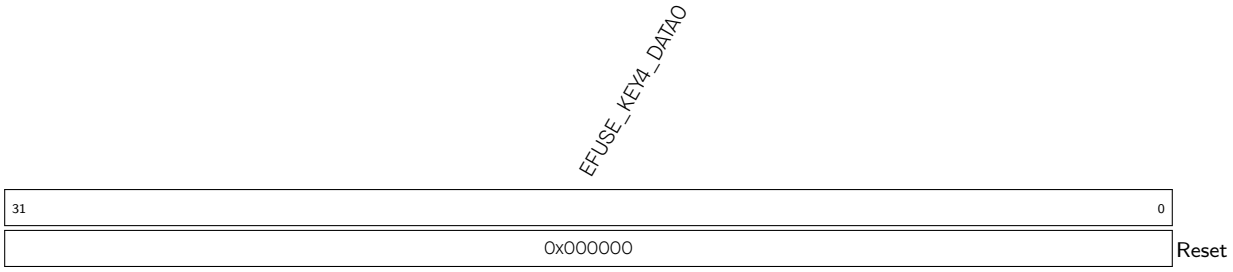
EFUSE_KEY2_DATA*n* 表示 KEY2 第 *n* 个 32 位内容。(RO)

Register 9.19. EFUSE_RD_KEY3_DATA*n*_REG (*n*: 0-7) (0x00FC+0x4*n*)



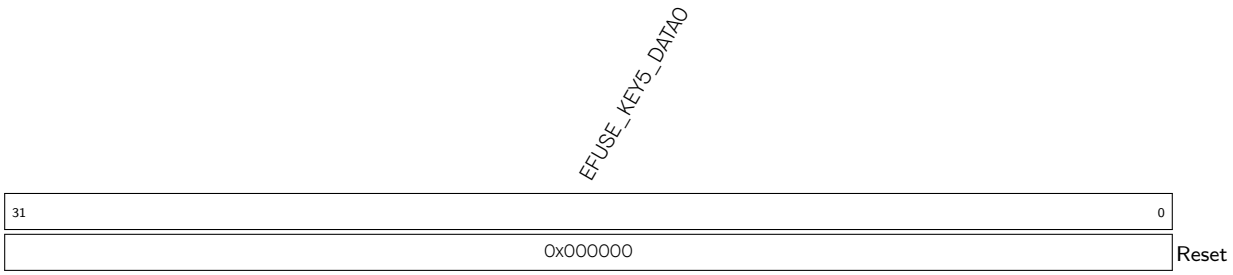
EFUSE_KEY3_DATA*n* 表示 KEY3 第 *n* 个 32 位内容。(RO)

Register 9.20. EFUSE_RD_KEY4_DATA*n*_REG (*n*: 0-7) (0x011C+0x4*n*)



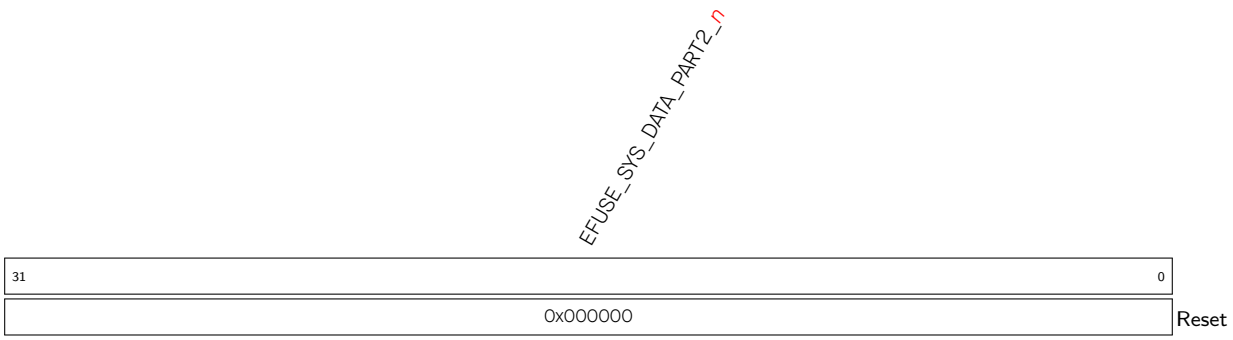
EFUSE_KEY4_DATA*n* 表示 KEY4 第 *n* 个 32 位内容。(RO)

Register 9.21. EFUSE_RD_KEY5_DATA*n*_REG (*n*: 0-7) (0x013C+0x4*n*)



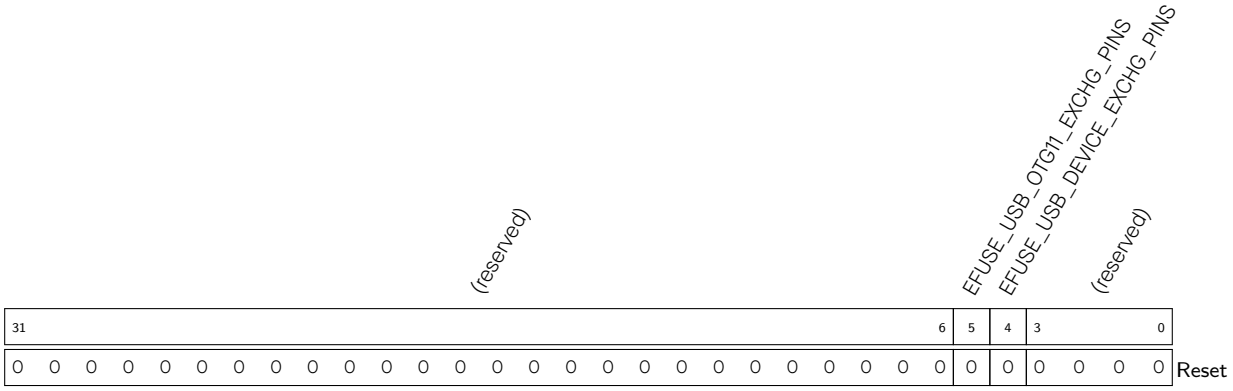
EFUSE_KEY5_DATA*n* 表示 KEY5 第 *n* 个 32 位内容。(RO)

Register 9.22. EFUSE_RD_SYS_PART2_DATA*n*_REG (*n*: 0-5) (0x015C+0x4*n*)



EFUSE_SYS_DATA_PART2_*n* 表示系统数据第 2 部分的第 *n* 个 32 位内容。(RO)

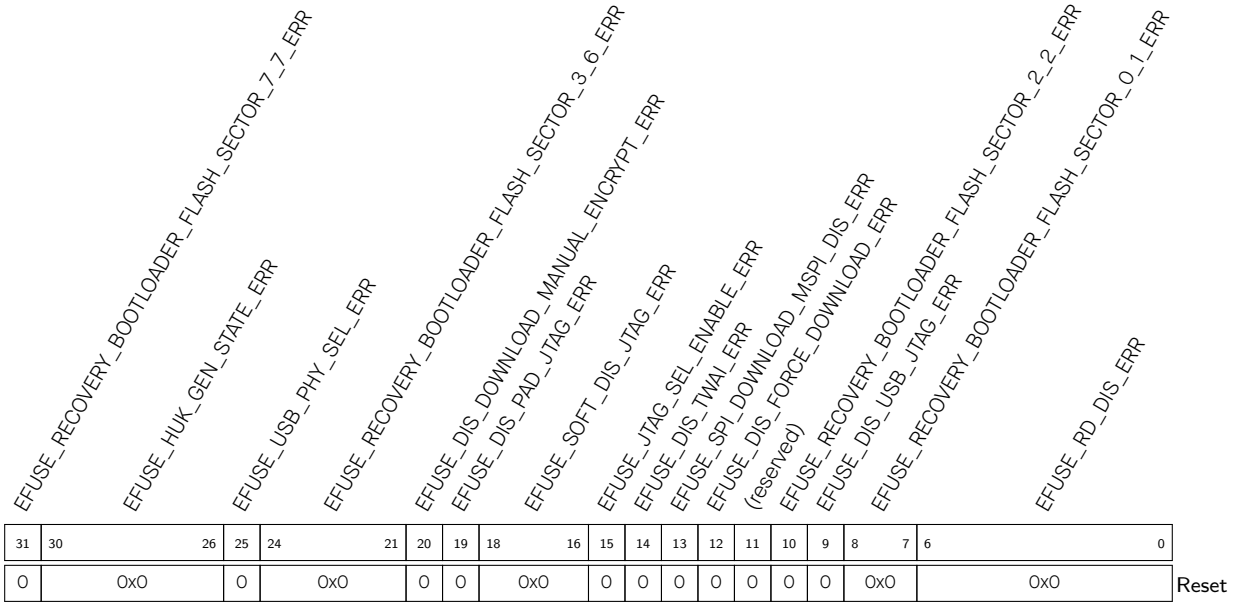
Register 9.23. EFUSE_RD_SYS_PART2_DATA7_REG (0x0178)



EFUSE_USB_DEVICE_EXCHG_PINS 表示是否使能 USB-Serial-JTAG 的 D+ 和 D- 交换引脚。(RO)

EFUSE_USB_OTG11_EXCHG_PINS 表示是否使能 USB OTG11 的 D + 和 D - 交换引脚。(RO)

Register 9.24. EFUSE_RD_REPEAT_DATA_ERR0_REG (0x017C)



EFUSE_RD_DIS_ERR 表示 EFUSE_RD_DIS 的烧写错误。(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_0_1 的烧写错误。(RO)

EFUSE_DIS_USB_JTAG_ERR 表示 EFUSE_DIS_USB_JTAG 的烧写错误。(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_2_2 的烧写错误。(RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR 表示 EFUSE_DIS_FORCE_DOWNLOAD 的烧写错误。(RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS_ERR 表示 EFUSE_SPI_DOWNLOAD_MSPI_DIS 的烧写错误。(RO)

EFUSE_DIS_TWAI_ERR 表示 EFUSE_DIS_TWAI 的烧写错误。(RO)

EFUSE_JTAG_SEL_ENABLE_ERR 表示 EFUSE_JTAG_SEL_ENABLE 的烧写错误。(RO)

EFUSE_SOFT_DIS_JTAG_ERR 表示 EFUSE_SOFT_DIS_JTAG 的烧写错误。(RO)

EFUSE_DIS_PAD_JTAG_ERR 表示 EFUSE_DIS_PAD_JTAG 的烧写错误。(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR 表示 EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 的烧写错误。(RO)

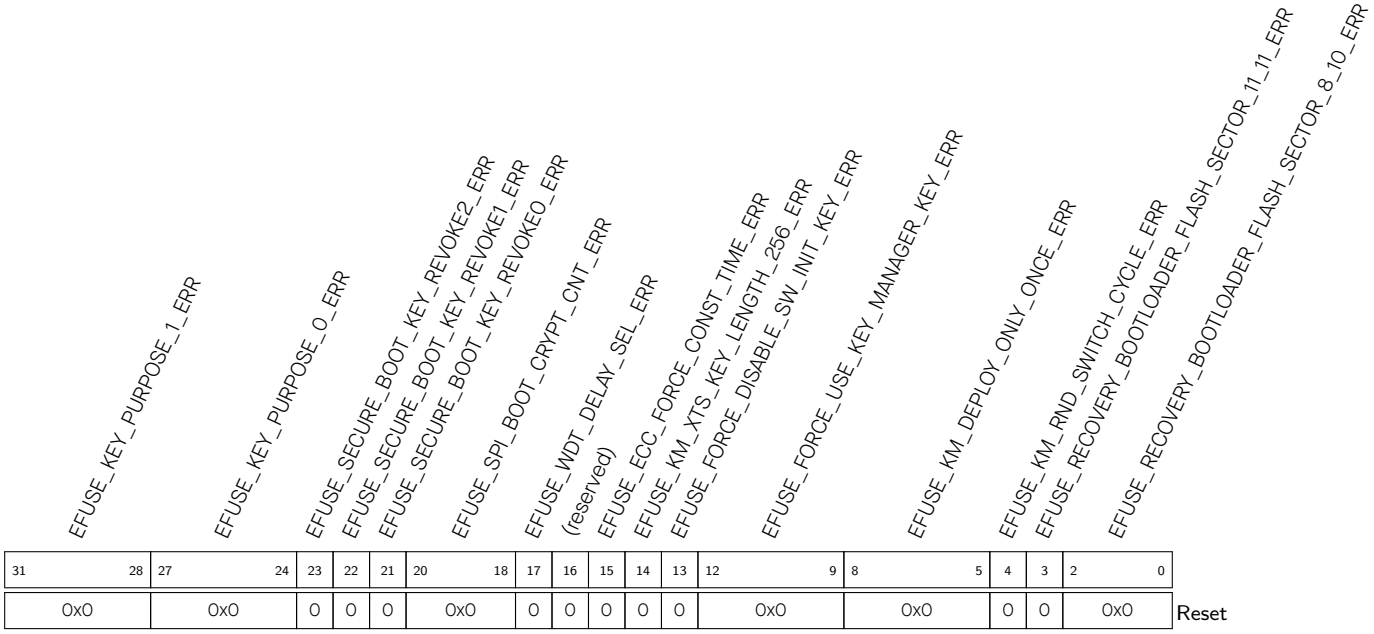
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_3_6 的烧写错误。(RO)

EFUSE_USB_PHY_SEL_ERR 表示 EFUSE_USB_PHY_SEL 的烧写错误。(RO)

EFUSE_HUK_GEN_STATE_ERR 表示 EFUSE_HUK_GEN_STATE 的烧写错误。(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_7_7 的烧写错误。(RO)

Register 9.25. EFUSE_RD_REPEAT_DATA_ERR1_REG (0x0180)



EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_8_10 的烧写错误。(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_11_11 的烧写错误。(RO)

EFUSE_KM_RND_SWITCH_CYCLE_ERR 表示 EFUSE_KM_RND_SWITCH_CYCLE 的烧写错误。(RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_ERR 表示 EFUSE_KM_DEPLOY_ONLY_ONCE 的烧写错误。(RO)

EFUSE_FORCE_USE_KEY_MANAGER_KEY_ERR 表示 EFUSE_FORCE_USE_KEY_MANAGER_KEY 的烧写错误。(RO)

EFUSE_FORCE_DISABLE_SW_INIT_KEY_ERR 表示 EFUSE_FORCE_DISABLE_SW_INIT_KEY 的烧写错误。(RO)

EFUSE_KM_XTS_KEY_LENGTH_256_ERR 表示 EFUSE_KM_XTS_KEY_LENGTH_256 的烧写错误。(RO)

EFUSE_ECC_FORCE_CONST_TIME_ERR 表示 EFUSE_ECC_FORCE_CONST_TIME 的烧写错误。(RO)

EFUSE_WDT_DELAY_SEL_ERR 表示 EFUSE_WDT_DELAY_SEL 的烧写错误。(RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR 表示 EFUSE_SPI_BOOT_CRYPT_CNT 的烧写错误。(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR 表示 EFUSE_SECURE_BOOT_KEY_REVOKE0 的烧写错误。(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR 表示 EFUSE_SECURE_BOOT_KEY_REVOKE1 的烧写错误。(RO)

见下页...

Register 9.25. EFUSE_RD_REPEAT_DATA_ERR1_REG (0x0180)

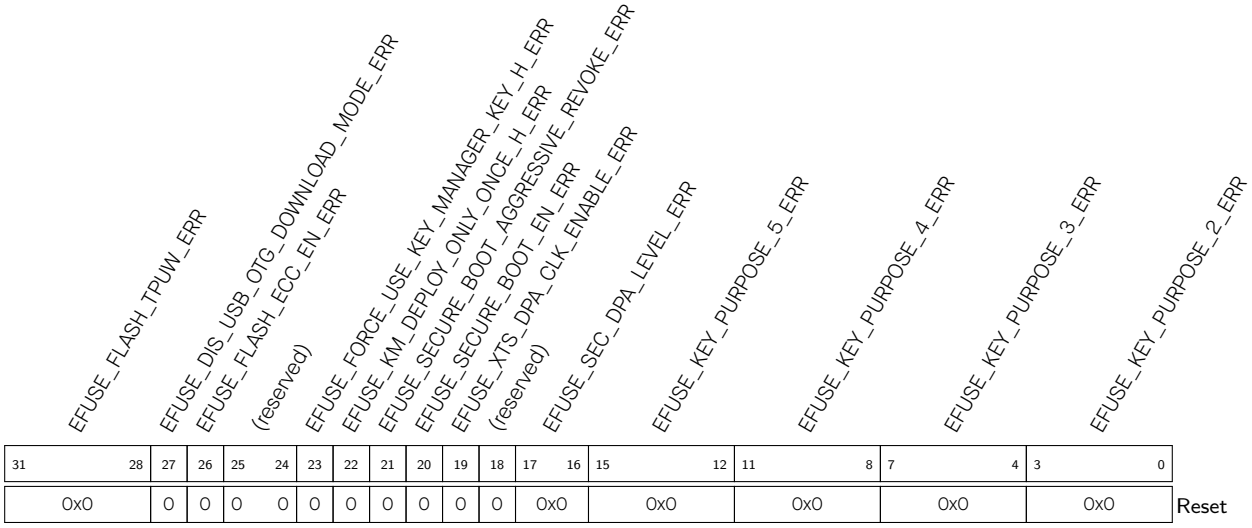
接上页...

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR 表示 EFUSE_SECURE_BOOT_KEY_REVOKE2 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_0_ERR 表示 EFUSE_KEY_PURPOSE_0 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_1_ERR 表示 EFUSE_KEY_PURPOSE_1 的烧写错误。(RO)

Register 9.26. EFUSE_RD_REPEAT_DATA_ERR2_REG (0x0184)



EFUSE_KEY_PURPOSE_2_ERR 表示 EFUSE_KEY_PURPOSE_2 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_3_ERR 表示 EFUSE_KEY_PURPOSE_3 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_4_ERR 表示 EFUSE_KEY_PURPOSE_4 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_5_ERR 表示 EFUSE_KEY_PURPOSE_5 的烧写错误。(RO)

EFUSE_SEC_DPA_LEVEL_ERR 表示 EFUSE_SEC_DPA_LEVEL 的烧写错误。(RO)

EFUSE_XTS_DPA_CLK_ENABLE_ERR 表示 EFUSE_XTS_DPA_CLK_ENABLE 的烧写错误。(RO)

EFUSE_SECURE_BOOT_EN_ERR 表示 EFUSE_SECURE_BOOT_EN 的烧写错误。(RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR 表示 EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE 的烧写错误。(RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_H_ERR 表示 EFUSE_KM_DEPLOY_ONLY_ONCE_H 的烧写错误。(RO)

EFUSE_FORCE_USE_KEY_MANAGER_KEY_H_ERR 表示 EFUSE_FORCE_USE_KEY_MANAGER_KEY_H 的烧写错误。(RO)

EFUSE_FLASH_ECC_EN_ERR 表示 EFUSE_FLASH_ECC_EN 的烧写错误。(RO)

EFUSE_DIS_USB_OTG_DOWNLOAD_MODE_ERR 表示 EFUSE_DIS_USB_OTG_DOWNLOAD_MODE 的烧写错误。(RO)

EFUSE_FLASH_TPUW_ERR 表示 EFUSE_FLASH_TPUW 的烧写错误。(RO)

Register 9.27. EFUSE_RD_REPEAT_DATA_ERR3_REG (0x0188)



EFUSE_DIS_DOWNLOAD_MODE_ERR 表示 EFUSE_DIS_DOWNLOAD_MODE 的烧写错误。(RO)

EFUSE_DIS_DIRECT_BOOT_ERR 表示 EFUSE_DIS_DIRECT_BOOT 的烧写错误。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT_ERR 表示 EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 的烧写错误。(RO)

EFUSE_LOCK_KM_KEY_ERR 表示 EFUSE_LOCK_KM_KEY 的烧写错误。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE_ERR 表示 EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE 的烧写错误。(RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR 表示 EFUSE_ENABLE_SECURITY_DOWNLOAD 的烧写错误。(RO)

EFUSE_UART_PRINT_CONTROL_ERR 表示 EFUSE_UART_PRINT_CONTROL 的烧写错误。(RO)

EFUSE_FORCE_SEND_RESUME_ERR 表示 EFUSE_FORCE_SEND_RESUME 的烧写错误。(RO)

EFUSE_SECURE_VERSION_ERR 表示 EFUSE_SECURE_VERSION 的烧写错误。(RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE_ERR 表示 EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE 的烧写错误。(RO)

EFUSE_HYS_EN_PAD_ERR 表示 EFUSE_HYS_EN_PAD 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_0_H_ERR 表示 EFUSE_KEY_PURPOSE_0_H 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_1_H_ERR 表示 EFUSE_KEY_PURPOSE_1_H 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_2_H_ERR 表示 EFUSE_KEY_PURPOSE_2_H 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_3_H_ERR 表示 EFUSE_KEY_PURPOSE_3_H 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_4_H_ERR 表示 EFUSE_KEY_PURPOSE_4_H 的烧写错误。(RO)

Register 9.28. EFUSE_RD_REPEAT_DATA_ERR4_REG (0x018C)



EFUSE_OPXA_TIEH_SEL_0_ERR 表示 EFUSE_OPXA_TIEH_SEL_0 的烧写错误。(RO)

EFUSE_PVT_GLITCH_EN_ERR 表示 EFUSE_PVT_GLITCH_EN 的烧写错误。(RO)

EFUSE_KEY_PURPOSE_5_H_ERR 表示 EFUSE_KEY_PURPOSE_5_H 的烧写错误。(RO)

EFUSE_KM_DISABLE_DEPLOY_MODE_H_ERR 表示 EFUSE_KM_DISABLE_DEPLOY_MODE_H 的烧写错误。(RO)

EFUSE_KM_DISABLE_DEPLOY_MODE_ERR 表示 EFUSE_KM_DISABLE_DEPLOY_MODE 的烧写错误。(RO)

EFUSE_XTS_DPA_PSEUDO_LEVEL_ERR 表示 EFUSE_XTS_DPA_PSEUDO_LEVEL 的烧写错误。(RO)

EFUSE_HP_PWR_SRC_SEL_ERR 表示 EFUSE_HP_PWR_SRC_SEL 的烧写错误。(RO)

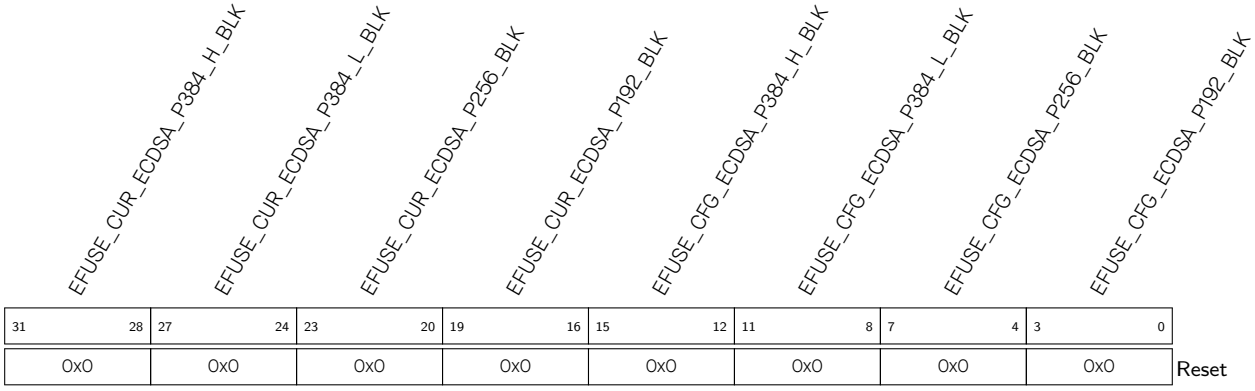
EFUSE_SECURE_BOOT_SHA384_EN_ERR 表示 EFUSE_SECURE_BOOT_SHA384_EN 的烧写错误。(RO)

EFUSE_DIS_WDT_ERR 表示 EFUSE_DIS_WDT 的烧写错误。(RO)

EFUSE_DIS_SWD_ERR 表示 EFUSE_DIS_SWD 的烧写错误。(RO)

EFUSE_PVT_GLITCH_MODE_ERR 表示 EFUSE_PVT_GLITCH_MODE 的烧写错误。(RO)

Register 9.29. EFUSE_ECDSA_REG (0x01B0)



EFUSE_CFG_ECDSA_P192_BLK 配置用于 ECDSA P192 密钥输出的 block 的编号。(R/W)

EFUSE_CFG_ECDSA_P256_BLK 配置用于 ECDSA P256 密钥输出的 block 的编号。(R/W)

EFUSE_CFG_ECDSA_P384_L_BLK 配置用于 ECDSA P384 密钥低部分输出的 block 的编号。(R/W)

EFUSE_CFG_ECDSA_P384_H_BLK 配置用于 ECDSA P384 密钥高部分输出的 block 的编号。(R/W)

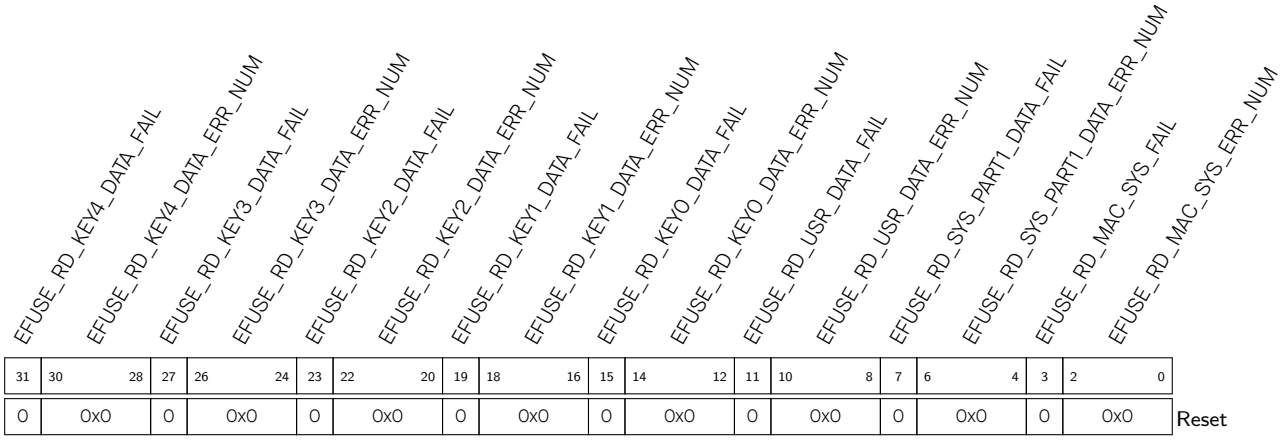
EFUSE_CUR_ECDSA_P192_BLK 表示用于 ECDSA P192 密钥输出的 block。(RO)

EFUSE_CUR_ECDSA_P256_BLK 表示用于 ECDSA P256 密钥输出的 block。(RO)

EFUSE_CUR_ECDSA_P384_L_BLK 表示用于 ECDSA P384 密钥低部分输出的 block。(RO)

EFUSE_CUR_ECDSA_P384_H_BLK 表示用于 ECDSA P384 密钥高部分输出的 block。(RO)

Register 9.30. EFUSE_RD_RS_DATA_ERR0_REG (0x01C0)



EFUSE_RD_MAC_SYS_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_MAC_SYS_FAIL 表示 MAC_SYS 的数据是否烧写失败。(RO)

0: 无烧写错误，MAC_SYS 的数据是可靠的

1: 用户数据烧写失败，错误字节数超过 6

EFUSE_RD_SYS_PART1_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_SYS_PART1_DATA_FAIL 表示第 1 部分的系统数据是否烧写失败。(RO)

0: 无烧写错误，第 1 部分的系统数据是可靠的

1: 用户数据烧写失败，错误字节数超过 6

EFUSE_RD_USR_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_USR_DATA_FAIL 表示用户数据烧写是否失败。(RO)

0: 无烧写错误，用户数据是可靠的

1: 用户数据烧写失败，错误字节数超过 6

EFUSE_RD_KEY0_DATA_ERR_NUM 表示错误字节数。(RO)

该信号的值表示 rd_key0_data 中的错误字节数。(RO)

EFUSE_RD_KEY0_DATA_FAIL 表示寄存器错误状态。(RO)

0: 表示无错误，rd_key0_data 的数据是可靠的

1: 表示 rd_key0_data 烧写失败，错误字节数超过 6。

EFUSE_RD_KEY1_DATA_ERR_NUM 表示错误字节数。(RO)

该信号的值表示 rd_key1_data 中的错误字节数。(RO)

EFUSE_RD_KEY1_DATA_FAIL 表示寄存器错误状态。(RO)

0: 表示无错误，rd_key1_data 的数据是可靠的

1: 表示 rd_key1_data 烧写失败，错误字节数超过 6。

EFUSE_RD_KEY2_DATA_ERR_NUM 表示错误字节数。(RO)

该信号的值表示 rd_key2_data 中的错误字节数。(RO)

EFUSE_RD_KEY2_DATA_FAIL 表示寄存器错误状态。(RO)

0: 表示无错误，rd_key2_data 的数据是可靠的

1: 表示 rd_key2_data 烧写失败，错误字节数超过 6。

见下页...

Register 9.30. EFUSE_RD_RS_DATA_ERR0_REG (0x01C0)

接上页...

EFUSE_RD_KEY3_DATA_ERR_NUM 表示错误字节数。(RO)

该信号的值表示 rd_key3_data 中的错误字节数。(RO)

EFUSE_RD_KEY3_DATA_FAIL 表示寄存器错误状态。(RO)

0: 表示无错误，rd_key3_data 的数据是可靠的

1: 表示 rd_key3_data 烧写失败，错误字节数超过 6。

EFUSE_RD_KEY4_DATA_ERR_NUM 表示错误字节数。(RO)

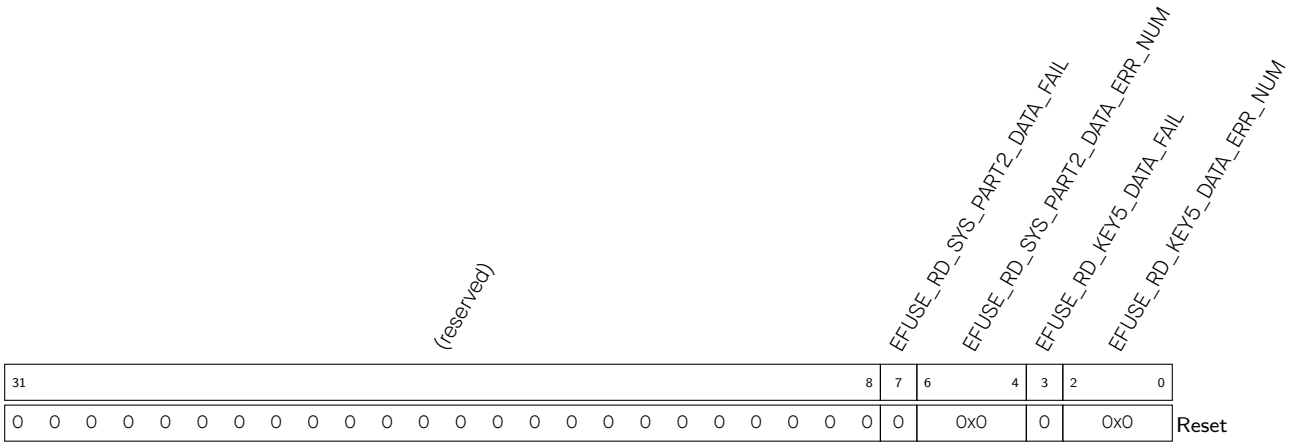
该信号的值表示 rd_key4_data 中的错误字节数。

EFUSE_RD_KEY4_DATA_FAIL 表示寄存器错误状态。(RO)

0: 表示无错误，rd_key4_data 的数据是可靠的

1: 表示 rd_key4_data 烧写失败，错误字节数超过 6。

Register 9.31. EFUSE_RD_RS_DATA_ERR1_REG (0x01C4)



EFUSE_RD_KEY5_DATA_ERR_NUM 表示错误字节数。(RO)

该信号的值表示 rd_key5_data 中的错误字节数。(RO)

EFUSE_RD_KEY5_DATA_FAIL 表示寄存器错误状态。(RO)

0: 表示无错误，rd_key5_data 的数据是可靠的

1: 表示 rd_key5_data 烧写失败，错误字节数超过 6。

EFUSE_RD_SYS_PART2_DATA_ERR_NUM 表示错误字节数。(RO)

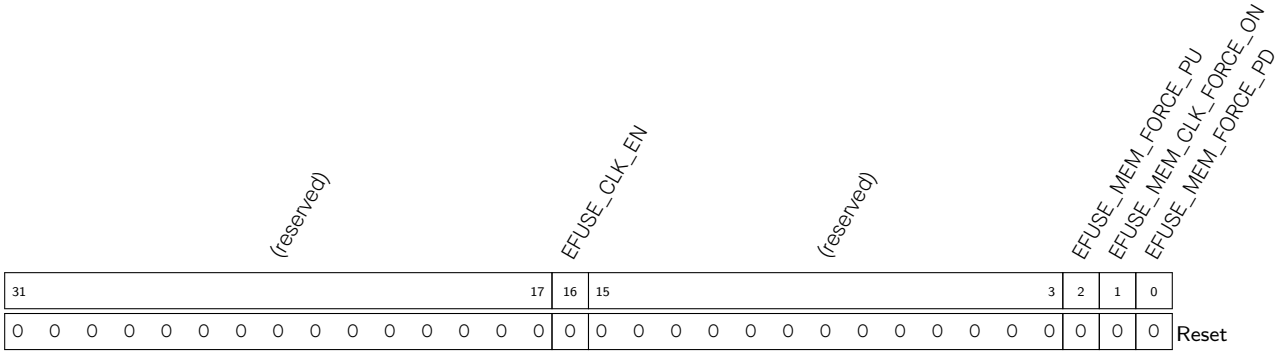
该信号的值表示 rd_sys_part2_data 中的错误字节数。(RO)

EFUSE_RD_SYS_PART2_DATA_FAIL 表示寄存器错误状态。(RO)

0: 表示无错误，rd_sys_part2_data 的数据是可靠的

1: 表示 rd_sys_part2_data 烧写失败，错误字节数超过 6。

Register 9.32. EFUSE_CLK_REG (0x01C8)



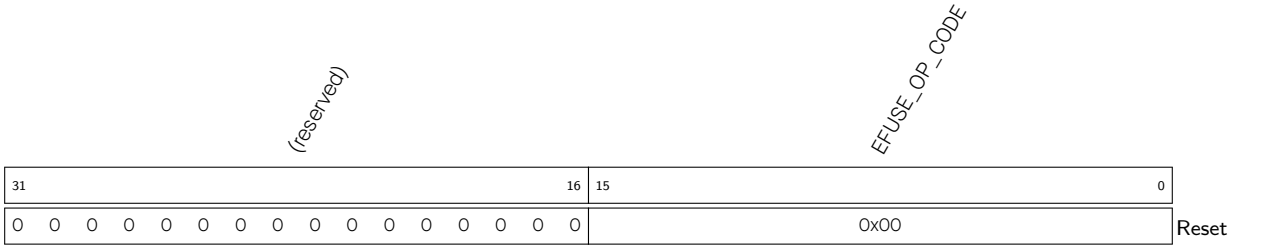
EFUSE_MEM_FORCE_PD 配置是否强制 eFuse SRAM 下电。(R/W)

EFUSE_MEM_CLK_FORCE_ON 配置是否强制激活 eFuse SRAM 时钟信号。(R/W)

EFUSE_MEM_FORCE_PU 配置是否强制 eFuse SRAM 上电。(R/W)

EFUSE_CLK_EN 配置是否强制使能 eFuse 寄存器配置时钟信号。(R/W)

Register 9.33. EFUSE_CONF_REG (0x01CC)

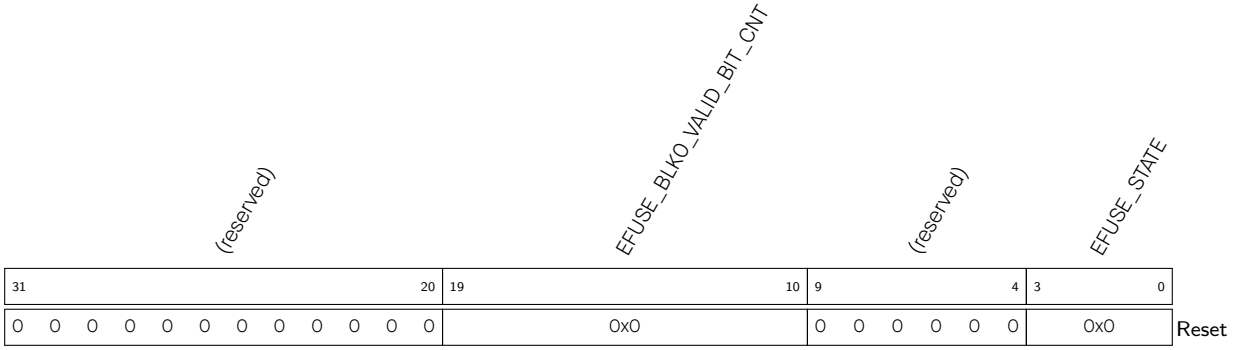


EFUSE_OP_CODE 配置操作指令类型。(R/W)

0x5A5A：烧写操作指令

0x5AA5：读操作指令

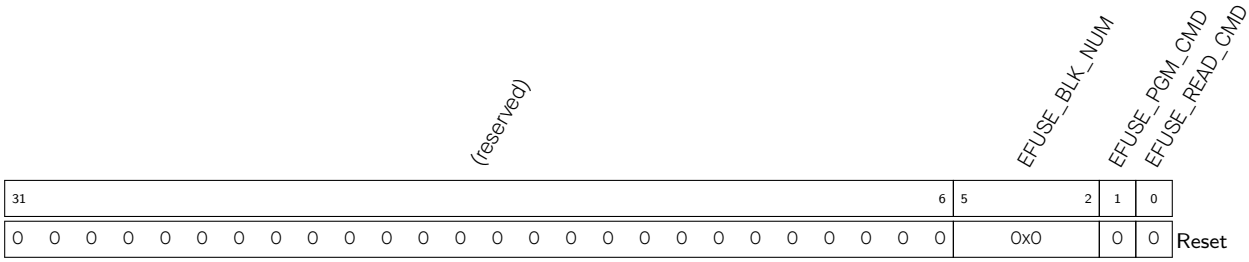
Register 9.34. EFUSE_STATUS_REG (0x01D0)



EFUSE_STATE 表示 eFuse 状态机的状态。(RO)

EFUSE_BLK0_VALID_BIT_CNT 表示 block 中有效位的数量。(RO)

Register 9.35. EFUSE_CMD_REG (0x01D4)

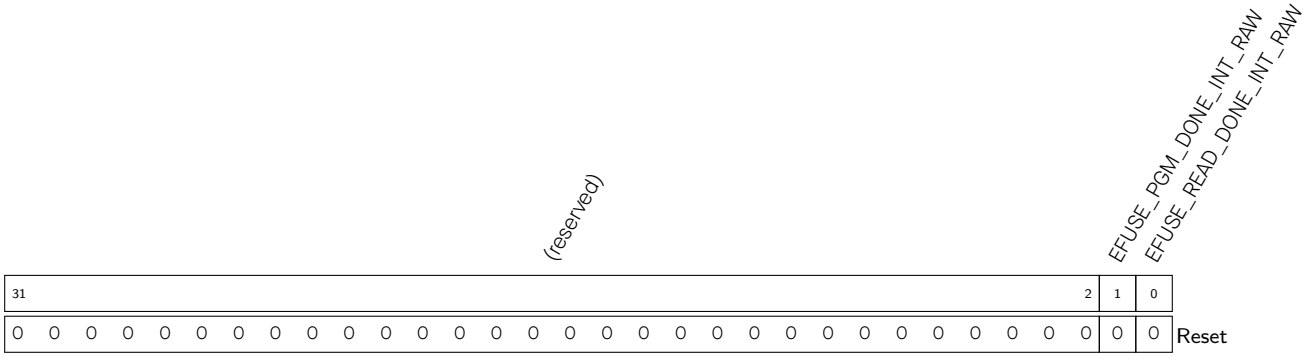


EFUSE_READ_CMD 配置是否发送读指令。(R/W/SC)

EFUSE_PGM_CMD 配置是否发送烧写指令。(R/W/SC)

EFUSE_BLK_NUM 配置待烧写的 block 编号。值 0 10 分别对应 BLOCK0 BLOCK10。(R/W)

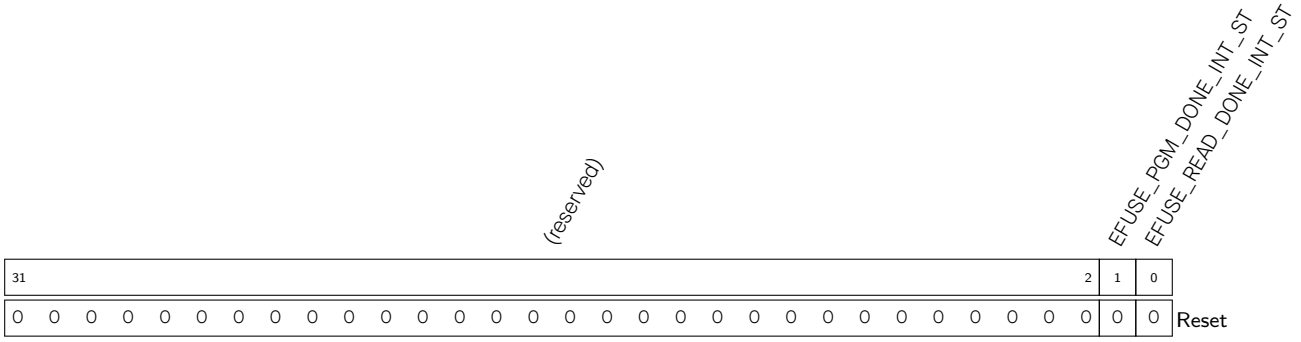
Register 9.36. EFUSE_INT_RAW_REG (0x01D8)



EFUSE_READ_DONE_INT_RAW read_done 中断的原始中断状态。(R/SS/WTC)

EFUSE_PGM_DONE_INT_RAW pgm_done 中断的原始中断状态。(R/SS/WTC)

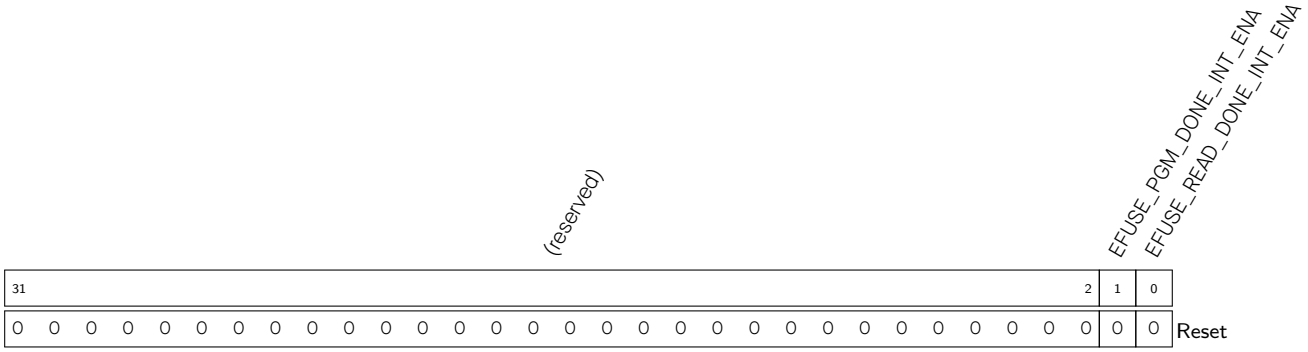
Register 9.37. EFUSE_INT_ST_REG (0x01DC)



EFUSE_READ_DONE_INT_ST read_done 中断的屏蔽中断状态。(RO)

EFUSE_PGM_DONE_INT_ST pgm_done 中断的屏蔽中断状态。(RO)

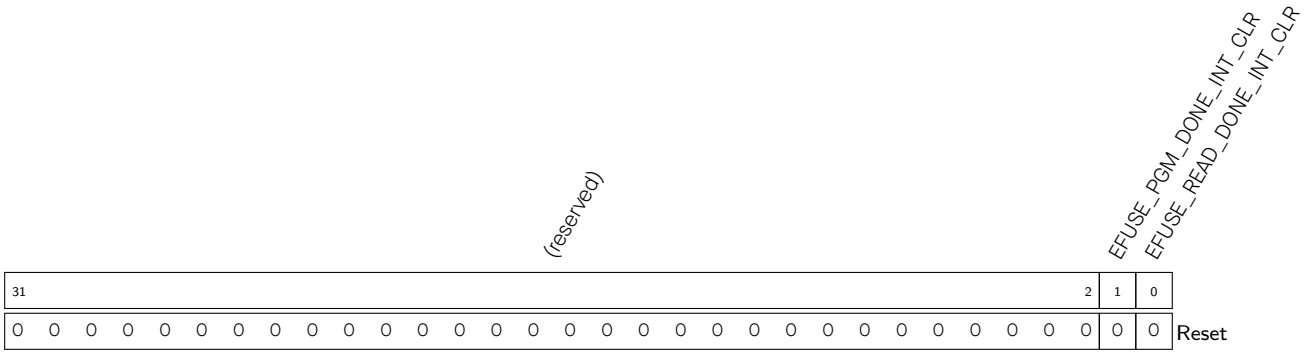
Register 9.38. EFUSE_INT_ENA_REG (0x01E0)



EFUSE_READ_DONE_INT_ENA 写 1 使能 read_done 中断。(R/W)

EFUSE_PGM_DONE_INT_ENA 写 1 使能 pgm_done 中断。(R/W)

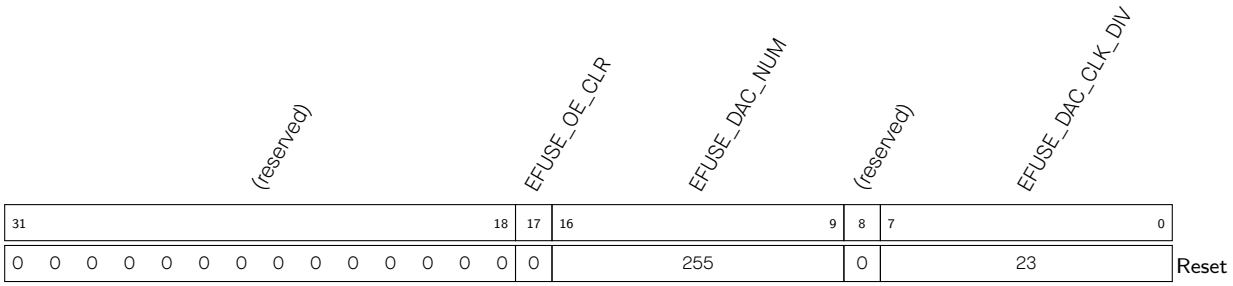
Register 9.39. EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR 写 1 清除 read_done 中断。(WT)

EFUSE_PGM_DONE_INT_CLR 写 1 清除 pgm_done 中断。(WT)

Register 9.40. EFUSE_DAC_CONF_REG (0x01E8)

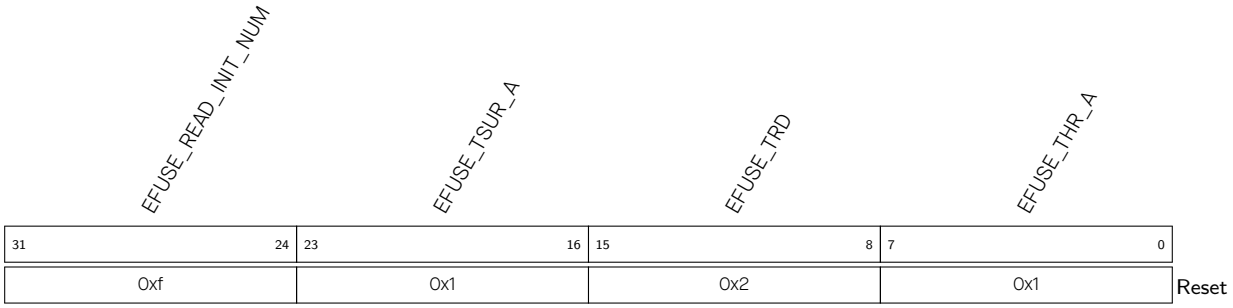


EFUSE_DAC_CLK_DIV 配置烧写电压上升时钟的分频系数。(R/W)

EFUSE_DAC_NUM 配置烧写电压的上升周期。(R/W)

EFUSE_OE_CLR 配置是否降低烧写电压的电源供应。(R/W)

Register 9.41. EFUSE_RD_TIM_CONF_REG (0x01EC)



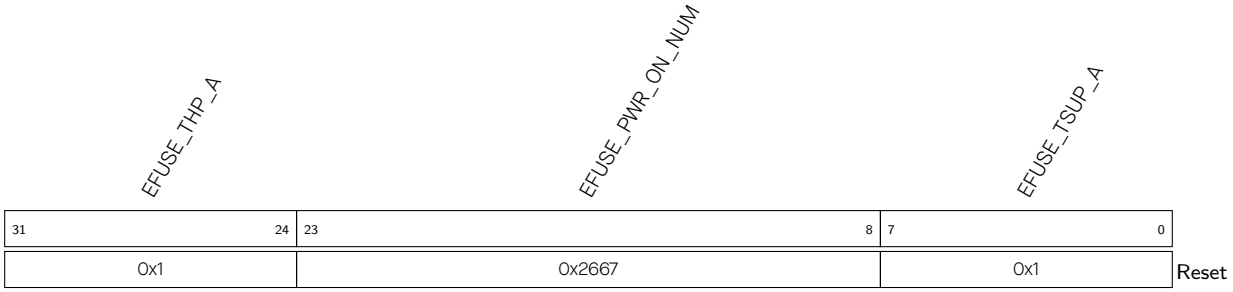
EFUSE_THR_A 配置读保持时间。(R/W)

EFUSE_TRD 配置读时间。(R/W)

EFUSE_TSUR_A 配置读建立时间。(R/W)

EFUSE_READ_INIT_NUM 配置读取 eFuse 存储器的等待时间。(R/W)

Register 9.42. EFUSE_WR_TIM_CONF1_REG (0x01F0)

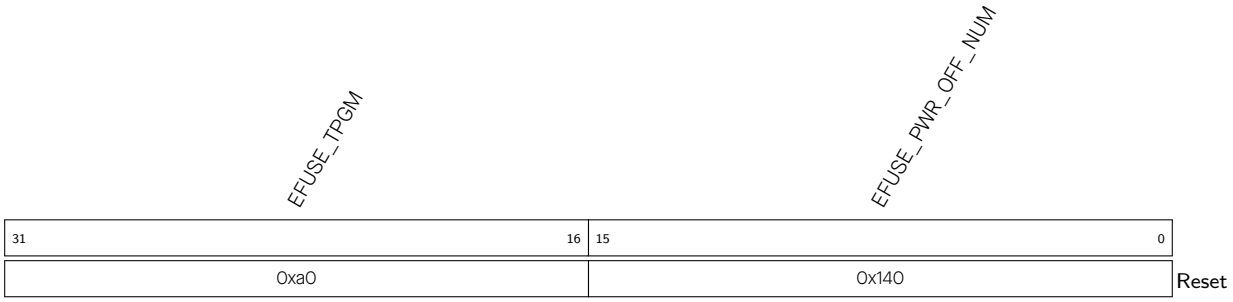


EFUSE_TSUP_A 配置烧写建立时间。(R/W)

EFUSE_PWR_ON_NUM 配置 VDDQ 的上电时间。(R/W)

EFUSE_THP_A 配置烧写保持时间。(R/W)

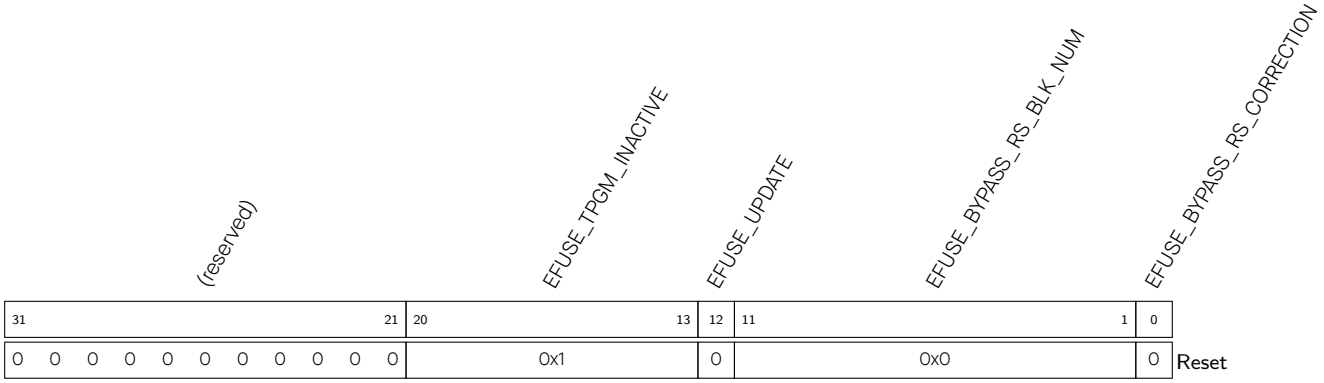
Register 9.43. EFUSE_WR_TIM_CONF2_REG (0x01F4)



EFUSE_PWR_OFF_NUM 配置 VDDQ 的断电时间。(R/W)

EFUSE_TPGM 配置有效烧写时间。(R/W)

Register 9.44. EFUSE_WR_TIM_CONF0_RS_BYPASS_REG (0x01F8)



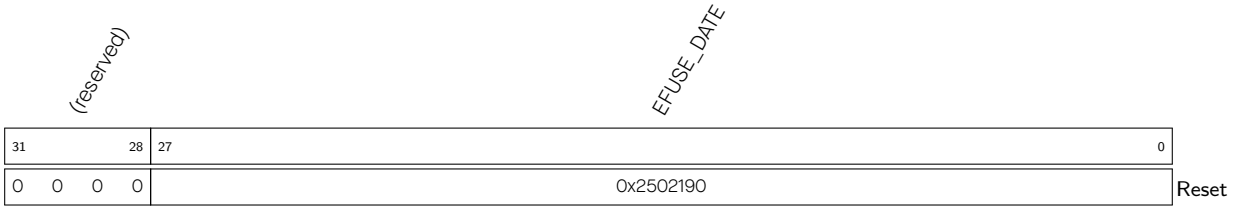
EFUSE_BYPASS_RS_CORRECTION 配置是否绕过 Reed-Solomon 纠错步骤。(R/W)

EFUSE_BYPASS_RS_BLK_NUM 配置二次烧写操作的块号。(R/W)

EFUSE_UPDATE 写 1 更新多位寄存器信号。(WT)

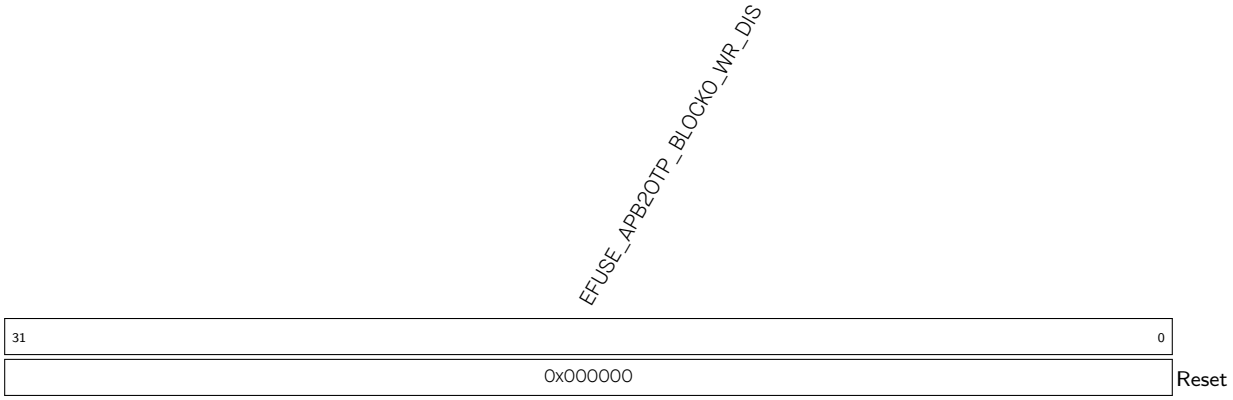
EFUSE_TPGM_INACTIVE 配置无效烧写时间。(R/W)

Register 9.45. EFUSE_DATE_REG (0x01FC)



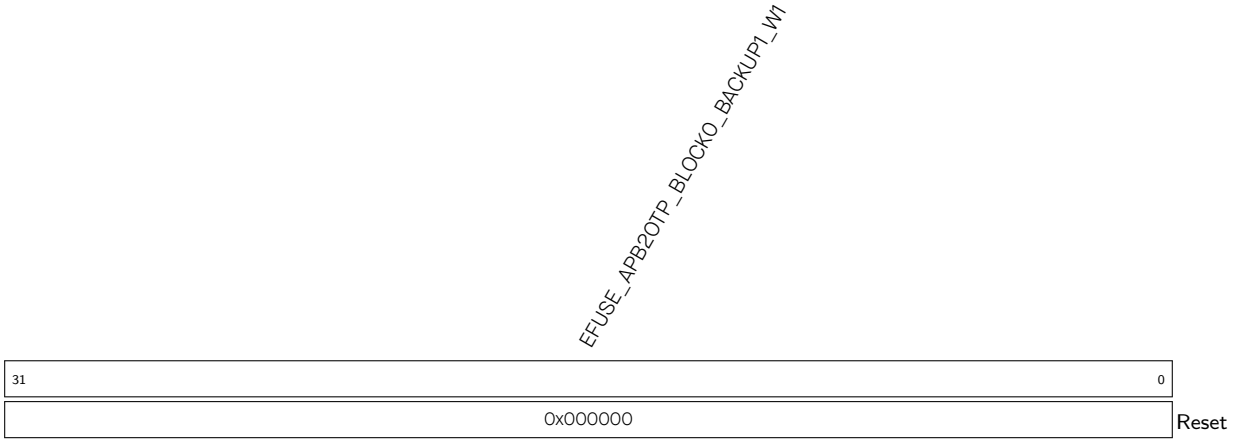
EFUSE_DATE 配置 eFuse 版本。(R/W)

Register 9.46. EFUSE_APB2OTP_WR_DIS_REG (0x0800)



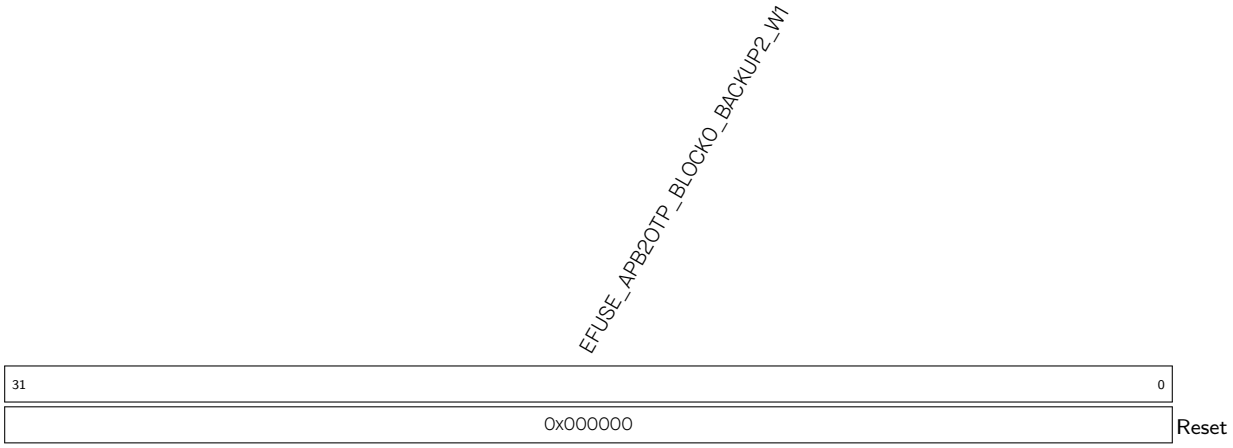
EFUSE_APB2OTP_BLOCK0_WR_DIS 表示 OTP block0 写禁用数据。(RO)

Register 9.47. EFUSE_APB2OTP_BLK0_BACKUP1_W*n*_REG (*n*: 1-5) (0x0800+0x4**n*)



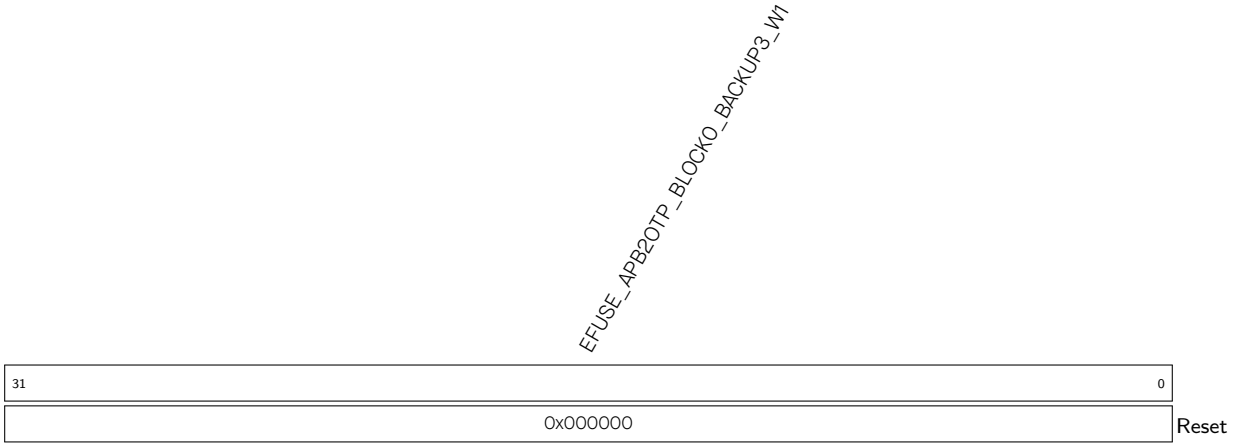
EFUSE_APB2OTP_BLOCK0_BACKUP1_W*n* 表示 OTP block0 backup1 word*n* 数据。(RO)

Register 9.48. EFUSE_APB2OTP_BLK0_BACKUP2_W*n*_REG (*n*: 1-5) (0x0814+0x4**n*)



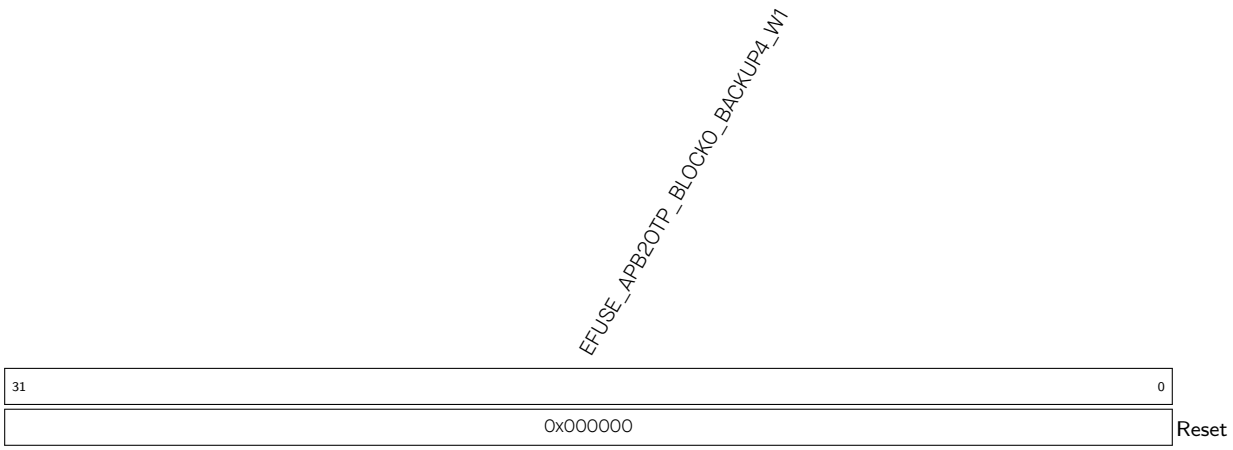
EFUSE_APB2OTP_BLOCK0_BACKUP2_W*n* 表示 OTP block0 backup2 word*n* 数据。(RO)

Register 9.49. EFUSE_APB2OTP_BLK0_BACKUP3_W*n*_REG (*n*: 1-5) (0x0828+0x4**n*)



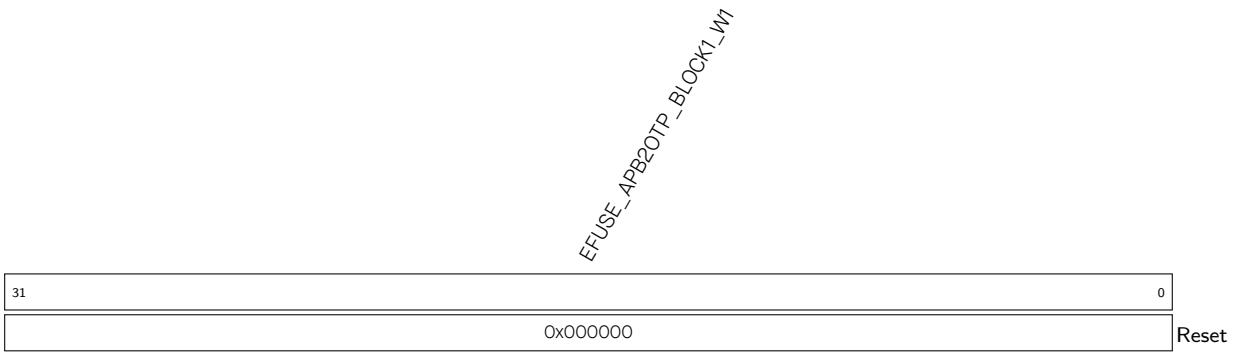
EFUSE_APB2OTP_BLOCK0_BACKUP3_W*n* 表示 OTP block0 backup3 word*n* 数据。(RO)

Register 9.50. EFUSE_APB2OTP_BLK0_BACKUP4_W*n*_REG (*n*: 1-5) (0x083C+0x4**n*)



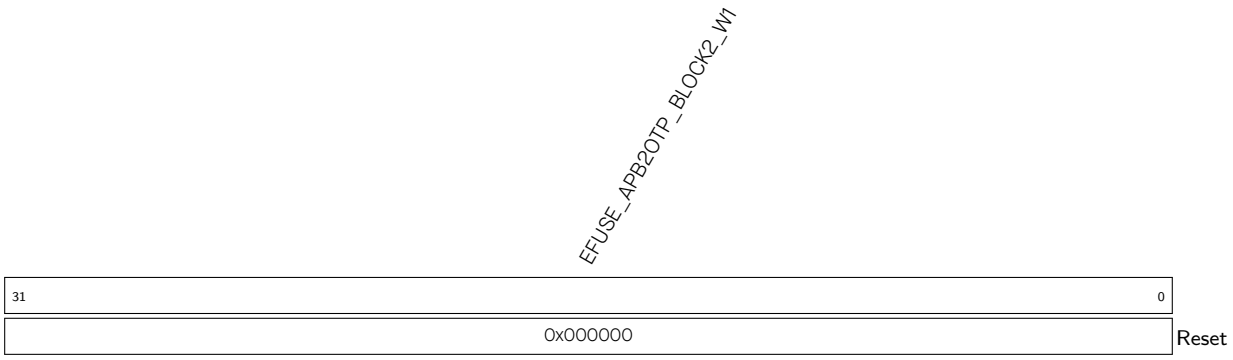
EFUSE_APB2OTP_BLOCK0_BACKUP4_W*n* 表示 OTP block0 backup4 word*n* 数据。(RO)

Register 9.51. EFUSE_APB2OTP_BLK1_W*n*_REG (*n*: 1-9) (0x0850+0x4**n*)



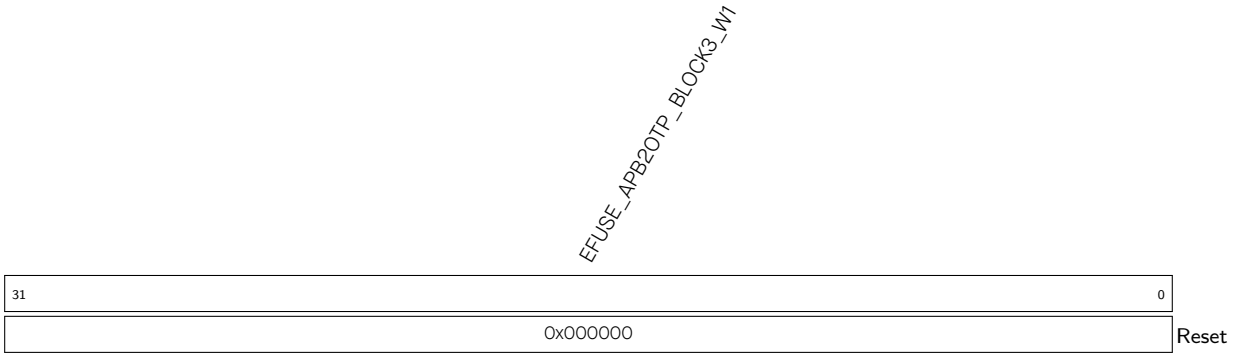
EFUSE_APB2OTP_BLOCK1_W*n* 表示 OTP block1 的第 *n* 个 32 位字数据。(RO)

Register 9.52. EFUSE_APB2OTP_BLK2_W*n*_REG (*n*: 1-11) (0x0874+0x4**n*)



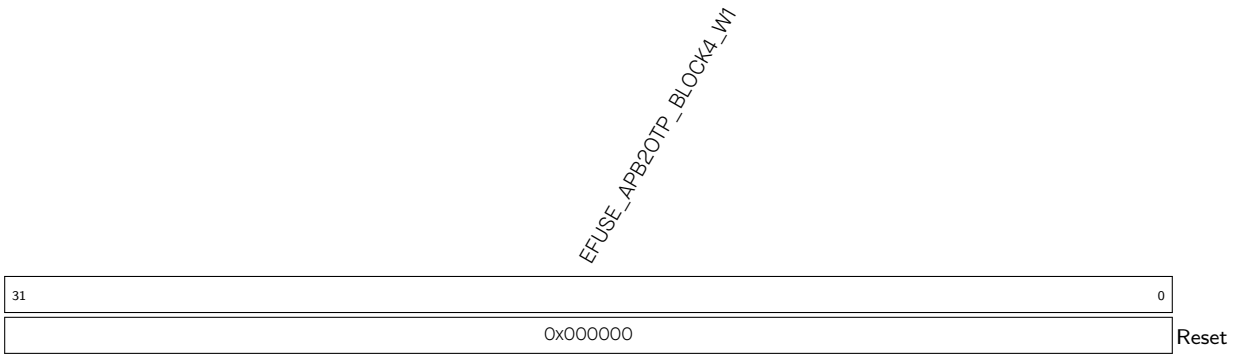
EFUSE_APB2OTP_BLOCK2_W*n* 表示 OTP block2 的第 *n* 个 32 位字数据。(RO)

Register 9.53. EFUSE_APB2OTP_BLK3_W*n*_REG (*n*: 1-11) (0x08A0+0x4**n*)



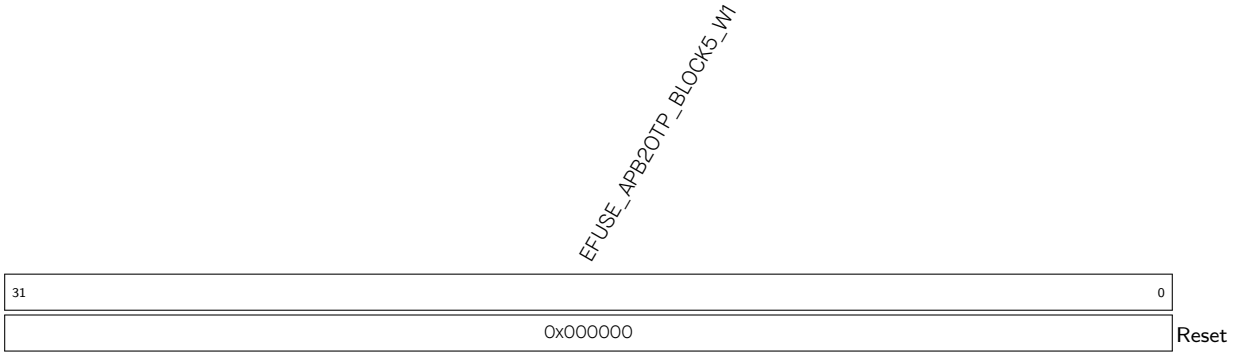
EFUSE_APB2OTP_BLOCK3_W*n* 表示 OTP block3 的第 *n* 个 32 位字数据。(RO)

Register 9.54. EFUSE_APB2OTP_BLK4_W*n*_REG (*n*: 1-11) (0x08CC+0x4**n*)



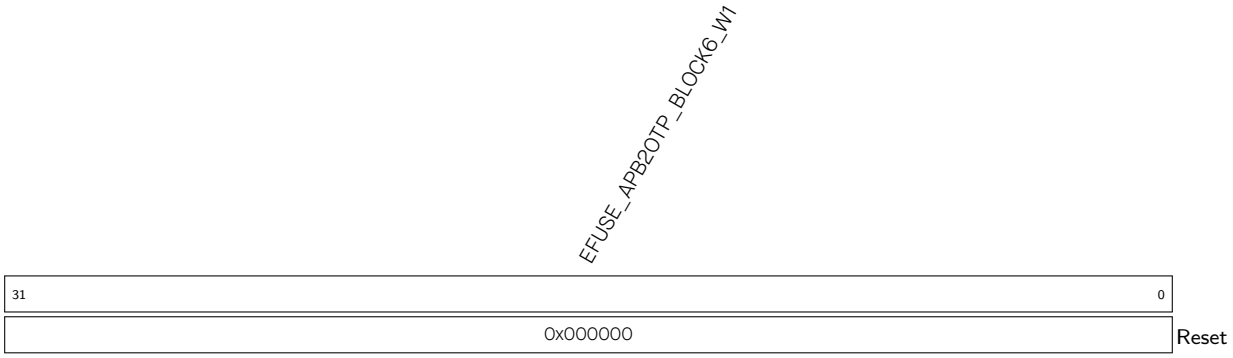
EFUSE_APB2OTP_BLOCK4_W*n* 表示 OTP block4 的第 *n* 个 32 位字数据。(RO)

Register 9.55. EFUSE_APB2OTP_BLK5_W*n*_REG (*n*: 1-11) (0x08F8+0x4**n*)



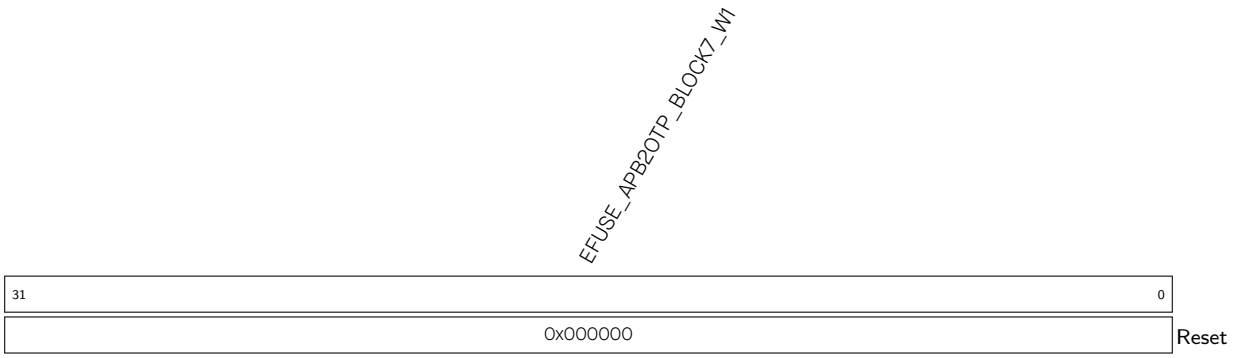
EFUSE_APB2OTP_BLOCK5_W*n* 表示 OTP block5 的第 *n* 个 32 位字数据。(RO)

Register 9.56. EFUSE_APB2OTP_BLK6_W*n*_REG (*n*: 1-11) (0x0924+0x4**n*)



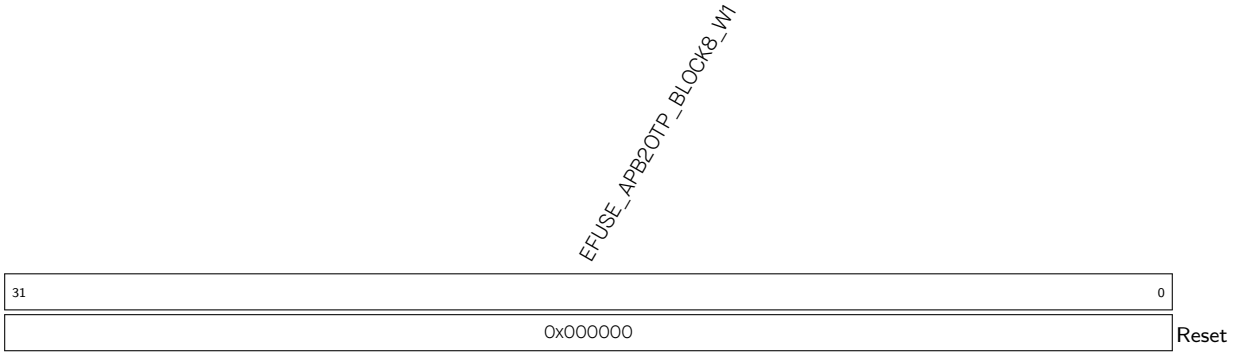
EFUSE_APB2OTP_BLOCK6_W*n* 表示 OTP block6 的第 *n* 个 32 位字数据。(RO)

Register 9.57. EFUSE_APB2OTP_BLK7_W*n*_REG (*n*: 1-11) (0x0950+0x4**n*)



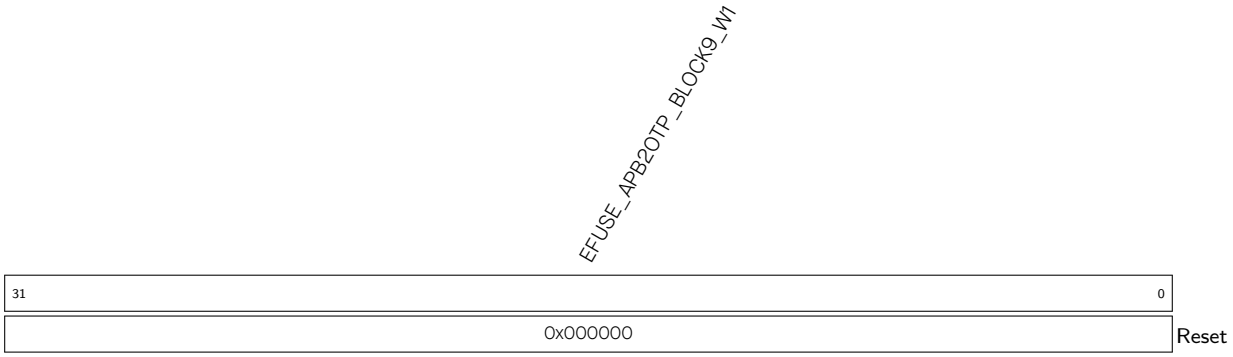
EFUSE_APB2OTP_BLOCK7_W*n* 表示 OTP block7 的第 *n* 个 32 位字数据。(RO)

Register 9.58. EFUSE_APB2OTP_BLK8_W*n*_REG (*n*: 1-11) (0x097C+0x4**n*)



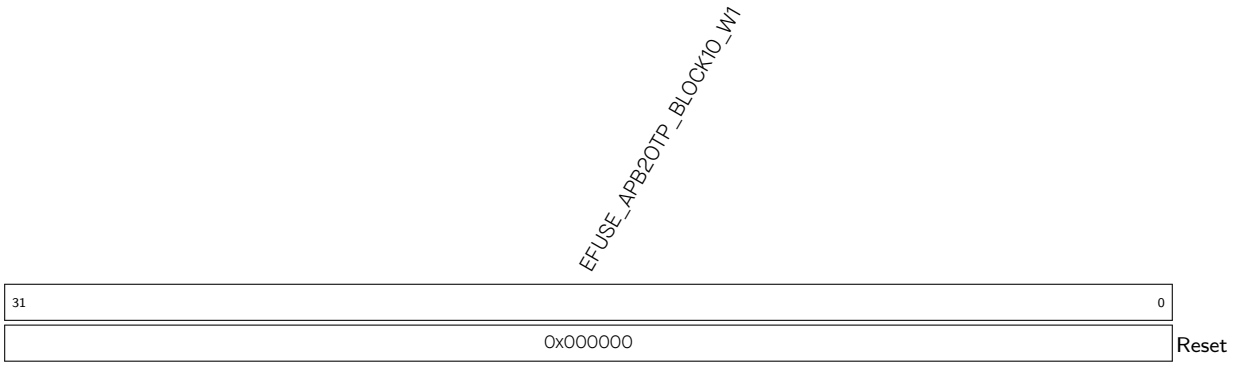
EFUSE_APB2OTP_BLOCK8_W*n* 表示 OTP block8 的第 *n* 个 32 位字数据。(RO)

Register 9.59. EFUSE_APB2OTP_BLK9_W*n*_REG (*n*: 1-11) (0x09A8+0x4**n*)



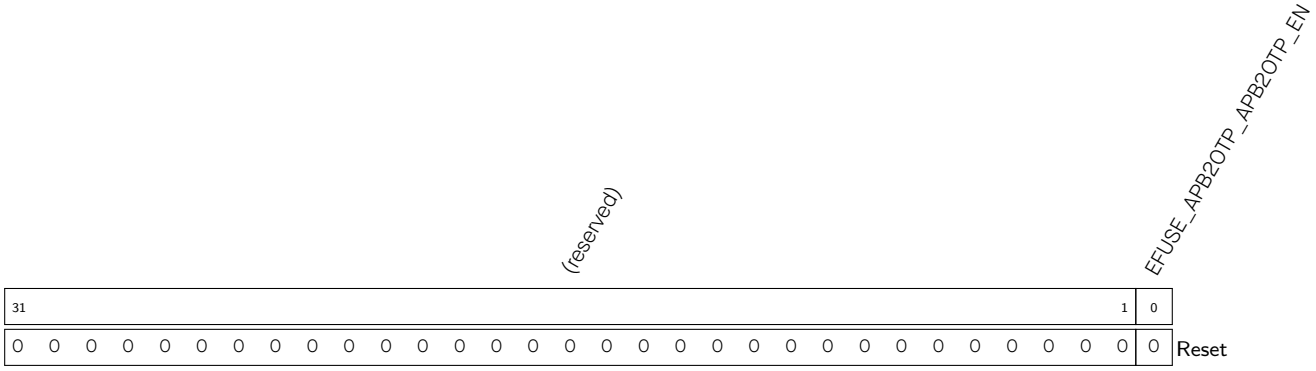
EFUSE_APB2OTP_BLOCK9_W*n* 表示 OTP block9 的第 *n* 个 32 位字数据。(RO)

Register 9.60. EFUSE_APB2OTP_BLK10_W*n*_REG (*n*: 1-11) (0x09D4+0x4**n*)



EFUSE_APB2OTP_BLOCK10_W*n* 表示 OTP block10 的第 *n* 个 32 位字数据。(RO)

Register 9.61. EFUSE_APB2OTP_EN_REG (0x0A08)



EFUSE_APB2OTP_APB2OTP_EN 配置是否启用 APB2OTP 模式。(R/W)

第 IV 卷

系统组件

该部分介绍芯片的系统级功能，包括系统启动、时钟、GPIO、定时器、看门狗、调试辅助、事件和中断处理、低功耗管理以及各种系统寄存器。

第 10 章

GPIO 交换矩阵和 IO MUX

10.1 概述

GPIO 交换矩阵是一种硬件结构，允许芯片上的 GPIO 管脚与外设输入/输出信号之间进行动态的重新映射和交换，以提供更大的灵活性。IO MUX（输入/输出多路复用）是一种技术，允许同一管脚在不同时间承担不同的功能，通过配置实现对管脚功能的动态切换。ESP32-P4 提供两组 GPIO 交换矩阵和 IO MUX：

- HP GPIO 交换矩阵和 HP IO MUX
- LP GPIO 交换矩阵和 LP IO MUX

ESP32-P4 共有 55 个 GPIO 管脚，其中包括 16 个低功耗 (LP) GPIO 管脚和 39 个高性能 (HP) GPIO 管脚。每个管脚都可用作一个通用 IO，或连接一个内部的外设信号。

- 利用 HP GPIO 交换矩阵和 HP IO MUX，可配置 HP 外设模块的输入信号来源于任何的 GPIO 管脚，并且 HP 外设模块的输出信号也可连接到任意 GPIO 管脚。
- 利用 LP GPIO 交换矩阵和 LP IO MUX，可配置 LP 外设模块的输入信号来源于任何的 LP GPIO 管脚，并且 LP 外设模块的输出信号也可连接到任意 LP GPIO 管脚。

这些模块共同组成了芯片的 GPIO 控制。上述 55 个 GPIO 管脚的编号为：GPIO0 ~ GPIO54。

- GPIO 管脚 0 ~ 15 为 LP GPIO 管脚，可由 HP 或 LP 外设使用。
- GPIO 管脚 16 ~ 54 为 HP GPIO 管脚，只能由 HP 外设使用。

该 ESP32-P4 芯片有专用于连接外部 flash 和封装内 PSRAM 的管脚，此类管脚不可用于其他用途。更多信息见《*ESP32-P4 技术规格书*》> 章节*芯片与 flash 的管脚对应关系*。

除非另有说明，本章节所述的 GPIO 交换矩阵同时包括 HP GPIO 交换矩阵和 LP GPIO 交换矩阵；IO MUX 同时包括 HP IO MUX 和 LP IO MUX。

10.2 主要特性

10.2.1 HP GPIO 交换矩阵和 HP IO MUX

HP GPIO 交换矩阵具有以下特性：

- HP 外设输入输出信号和 GPIO 管脚之间的全交换矩阵
- 222 个 HP 外设输入信号可以选择任意一个 GPIO 管脚的输入信号
- 每个 GPIO 管脚的输出信号可以来自 232 个 HP 外设输出信号的任意一个
- HP 外设输入信号经 GPIO SYNC 模块同步至 HP IO MUX 运行时钟
- 支持 GPIO 滤波器对输入信号进行滤波

- 支持毛刺滤波器对输入信号进行二次滤波
- 支持 Sigma Delta 调制输出 (SDM)
- 支持 GPIO 简单输入输出
- 支持 HP GPIO 唤醒

HP IO MUX 具有以下特性：

- 控制 55 个 GPIO (GPIO0 ~ GPIO54) 供 HP 外设使用
- 为每个 GPIO 管脚提供一个寄存器 IO_MUX_GPIO*n*_REG，用于控制管脚的输入/输出、上拉/下拉、驱动强度、功能选择等配置
- 支持高频信号如 SPI、EMAC 等直接通过 HP IO MUX 输入和输出外设，实现更好的高频数字特性

10.2.2 LP GPIO 交换矩阵和 LP IO MUX

LP GPIO 交换矩阵具有如下特性：

- LP 外设输入输出信号和 LP GPIO 管脚之间的全交换矩阵
- 14 个 LP 外设输入信号可以选择任意一个 LP GPIO 管脚的输入信号
- 每个 LP GPIO 管脚的输出信号可以来自 14 个 LP 外设输出信号的任意一个
- 支持 GPIO 滤波器对输入信号进行滤波
- 支持 GPIO 简单输入输出
- 支持 LP GPIO 唤醒

LP IO MUX 具有如下特性：

- 16 个 LP GPIO 管脚 (GPIO0 ~ GPIO15) 供 LP 外设使用
- 为每个 LP GPIO 管脚提供一个 LP_IOMUX_PAD*x*_REG 寄存器，用于控制管脚的输入/输出、上拉/下拉、驱动强度、功能选择、IO MUX 选择等配置

10.3 结构概览

图 10.3-1 所示为 HP GPIO 交换矩阵、HP IO MUX、LP GPIO 交换矩阵和 LP IO MUX 将信号引入外设和引出至管脚的具体过程。

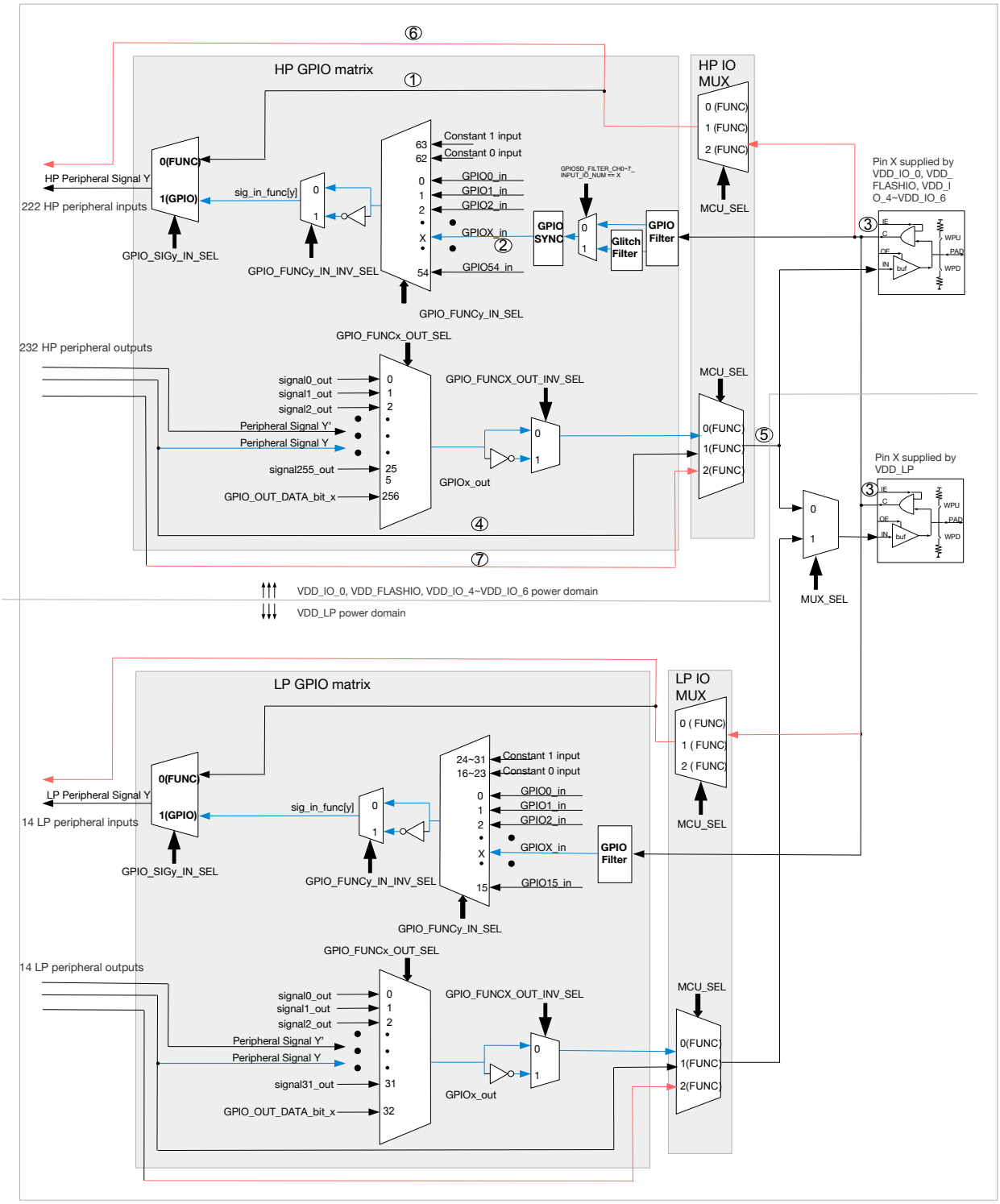


图 10.3-1. HP GPIO 交换矩阵、HP IO MUX 、LP GPIO 交换矩阵和 LP IO MUX 框图

以下各点内容是对上图数字标注区域的解释说明，以 HP 系统为例：

- ① 部分输入信号既可以通过 HP GPIO 交换矩阵连接外设又可以直接通过 HP IO MUX 直连外设，这些输入信号在表 10.12-1 “信号可经由 IO MUX 直接输入”一栏中被标为“yes”。剩余其他信号只能通过 HP GPIO

交换矩阵连接至外设。

- ② ESP32-P4 共有 55 个 GPIO 管脚，因此从 GPIO SYNC 进入到 HP GPIO 交换矩阵的输入共有 55 个。
- ③ 位于 VDD_IO_0、VDD_FLASHIO、VDD_IO_4 ~ VDD_IO_6 和 VDD_LP 电源域的管脚由 IE、OE、WPU 和 WPD 信号控制。
- ④ 部分输出信号既可以通过 HP GPIO 交换矩阵连接外设又可通过 HP IO MUX 直连管脚，这些输出信号在表 10.12-1 “信号可经由 IO MUX 直接输出”一栏中被标为“yes”。剩余其他信号只能通过 HP GPIO 交换矩阵连接至管脚。
- ⑤ 从 HP GPIO 交换矩阵到 HP IO MUX 的输出共有 55 个，对应 GPIO *X*：0 ~ 54。
- ⑥ 仅通过 HP IO MUX 的外设输入。
- ⑦ 仅通过 HP IO MUX 的外设输出。

图 10.3-2 展示了芯片焊盘 (PAD) 的内部结构，即芯片逻辑与 GPIO 管脚之间的电气接口。55 个 GPIO 管脚均采用这一结构，且由 IE、OE、WPU 和 WPD 信号控制。信号配置请参考 IO_MUX_GPIO*x*_REG 寄存器或 LP_IOMUX_PAD*x*_REG 寄存器。

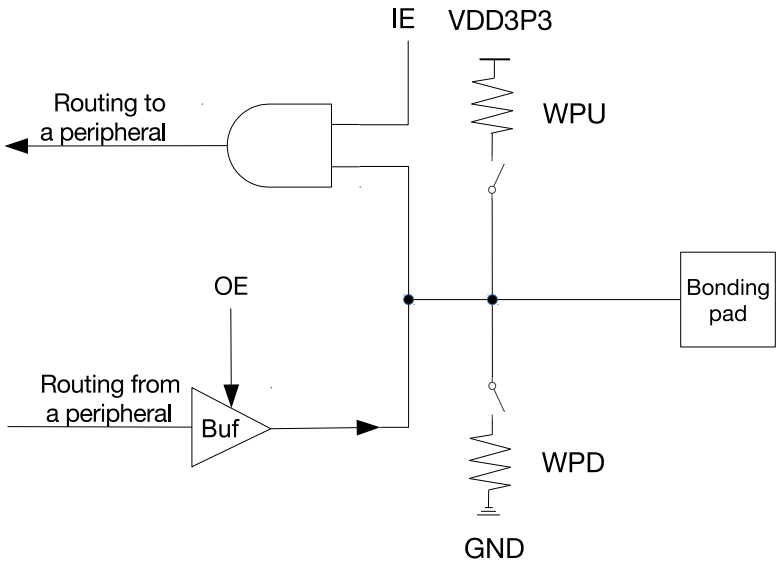


图 10.3-2. 焊盘内部结构

- IE：输入使能
- OE：输出使能
- WPU：内部弱上拉电阻
- WPD：内部弱下拉电阻
- Bonding pad：接合焊盘，芯片逻辑的结点，实现芯片封装内晶片与 GPIO 管脚之间的物理连接。

10.4 通过 GPIO 交换矩阵的外设输入

10.4.1 概述

通过 HP GPIO 交换矩阵接收外部输入信号，一般步骤如下：

- 配置 HP GPIO 交换矩阵从 55 个 GPIO (0 ~ 54) 中获取外部输入信号，见表 10.12-1。
- 配置外设输入选择通过 HP GPIO 交换矩阵接收输入信号。
- 配置 GPIO 管脚由 HP IO MUX 控制。

具体配置流程，可参考图 10.3-1 和章节 10.4.7。

通过 LP GPIO 交换矩阵接收外部输入信号，一般步骤如下：

- 配置 LP GPIO 交换矩阵从 16 个 GPIO (0 ~ 15) 中获取外部输入信号，见表 10.13-1。
- 配置外设输入选择通过 LP GPIO 交换矩阵接收输入信号。
- 配置 LP GPIO 管脚由 LP IO MUX 控制。

具体配置流程，可参考图 10.3-1 和章节 10.4.7。

如图 10.3-1 所示，使用 GPIO 交换矩阵将信号从管脚输入时，所有外部输入信号均来源于 GPIO 管脚，然后经 GPIO 滤波器进行滤波，见章节 10.4.7 中的步骤 2。

毛刺滤波器仅在 HP GPIO 交换矩阵中可用。毛刺滤波器硬件可从 GPIO 滤波器输出的信号中选择八个信号进行滤波，其他没被选中的信号则直接进入 GPIO SYNC 硬件，见章节 10.4.7 中的步骤 3。

所有信号经 GPIO 滤波器硬件或毛刺滤波器硬件滤波后，均将被 GPIO SYNC 硬件同步至 IO MUX 运行时钟，然后进入 GPIO 交换矩阵，见章节 10.4.2。外部输入信号也可经 IO MUX 直接输入至外设，但信号无法进行滤波及时钟同步操作。

10.4.2 信号同步

HP GPIO 交换矩阵支持信号同步功能。GPIO SYNC 模块的功能如图 10.4-1 所示。其中，negative sync 为 GPIO 输入经过 HP IO MUX 运行时钟的下降沿同步，positive sync 为 GPIO 输入经过 HP IO MUX 运行时钟的上升沿同步。

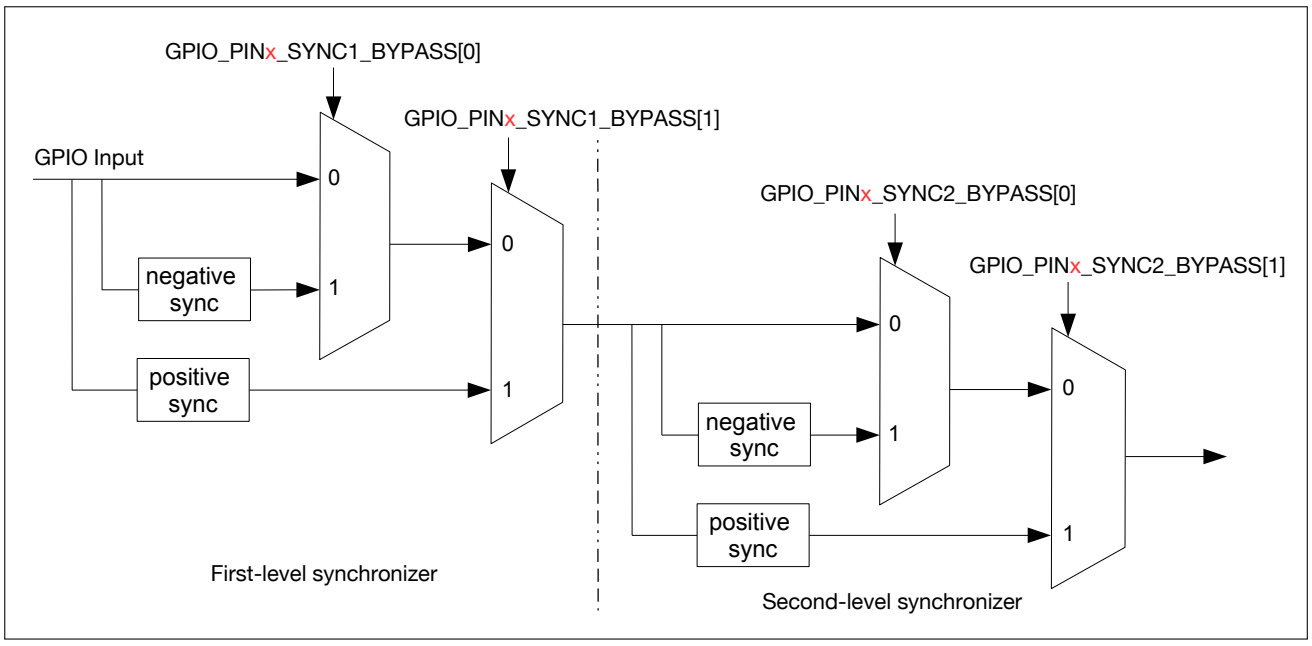


图 10.4-1. GPIO 输入经 HP IO MUX 运行时钟上升沿或下降沿同步

同步器 (synchronizer) 默认关闭同步功能。但如果一个异步外设信号连接到管脚时，该信号应通过两级同步器（即图 10.4-1 中的 first-level synchronizer 和 second-level synchronizer）进行同步，以减小亚稳态产生的概率。

10.4.3 GPIO 滤波器

HP GPIO 交换矩阵和 LP GPIO 交换矩阵均支持 GPIO 滤波功能。使能 GPIO 滤波器功能后，只有当输入信号的有效宽度大于两个时钟周期时，输入信号才会被采样。否则，输入信号将会被滤掉。见图 10.4-2。

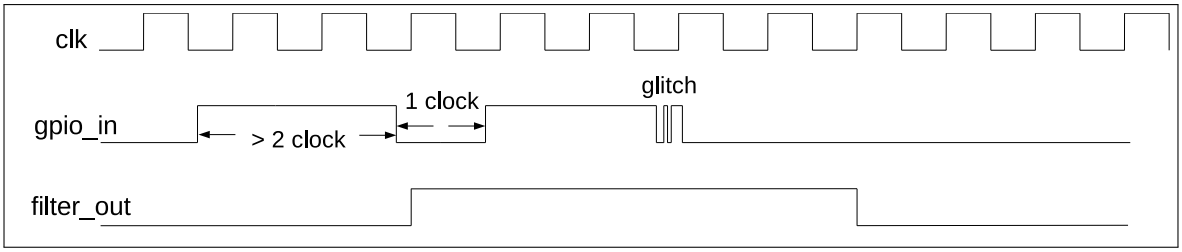


图 10.4-2. GPIO 输入信号滤波时序图

10.4.4 毛刺滤波器

毛刺滤波功能仅在 HP GPIO 交换矩阵中可用。毛刺滤波器硬件支持八个通道，每个通道可从 GPIO 滤波器硬件输出的 55 个信号 (0 ~ 54) 中选择一个信号，进行二次滤波。该毛刺滤波器硬件可用于对慢速信号进行滤波。见章节 10.4.7.1 中的步骤 3。

10.4.5 简单 GPIO 输入

HP GPIO 交换矩阵和 LP GPIO 交换矩阵均支持简单 GPIO 输入。使用简单 GPIO 输入模式时，任意 GPIO 管脚的值均可随时读取，而无需将 GPIO 管脚输入绑定到某个外设信号。

配置 HP GPIO 交换矩阵实现简单 GPIO 输入，具体过程如下：

- 置位 IO_MUX_GPIO*x*_REG 的 IO_MUX_GPIO*x*_MCU_IE 位，使能管脚输入。
- 读取 GPIO_IN_REG[*x*]，即可实现简单 GPIO 输入。

配置 LP GPIO 交换矩阵实现简单 GPIO 输入，具体过程如下：

- 置位 LP_IO_MUX_PAD*x*_REG 的 LP_IOMUX_PAD*x*_FUN_IE 位，使能管脚输入。
- 读取 LP_GPIO_IN_REG[*x*]，即可实现简单 GPIO 输入。

10.4.6 GPIO 唤醒

10.4.6.1 HP GPIO 唤醒

当 HP 系统处于时钟门控模式时，HP GPIO 唤醒功能可用于将系统从 Light-sleep 模式唤醒。当 HP 系统掉电时，HP GPIO 唤醒功能不可用。GPIO0 ~ GPIO54 可通过配置 GPIO_PIN*x*_WAKEUP_ENABLE (*x*: 0 ~ 54) 使能 HP GPIO 唤醒功能。

HP GPIO 唤醒功能支持高电压敏感和低电压敏感唤醒，具体由寄存器配置决定，见下表。

表 10.4-1. HP GPIO 唤醒信号触发条件和清除条件

GPIO_PIN*x*_INT_TYPE[1,2]	触发条件	清除条件
5	GPIO 输入为高电平	GPIO 输入为低电平
4	GPIO 输入为低电平	GPIO 输入为高电平

[1] *x* = 0 ~ 54。

[2] 配置为其他值时，HP GPIO 唤醒功能不可用。

10.4.6.2 LP GPIO 唤醒

当 LP 外设处于非电源门控模式时，LP GPIO 唤醒功能可用于将系统从 Deep-sleep 模式唤醒。当 LP 外设掉电时，LP GPIO 唤醒功能不可用。GPIO0 ~ GPIO15 可通过配置 LP_GPIO_PIN*x*_WAKEUP_ENABLE (*x* = 0 ~ 15) 使能 HP GPIO 唤醒功能。

LP GPIO 唤醒支持上升沿敏感、下降沿敏感、上升沿和下降沿敏感、高电压敏感以及低电压敏感唤醒。具体情况由寄存器配置决定，见下表。

表 10.4-2. LP GPIO 唤醒信号触发条件和清除条件

LP_GPIO_PIN*x*_INT_TYPE[1,2]	触发条件	清除条件
1	GPIO 输入从低电平跳变为高电平	置位 LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR
2	GPIO 输入从高电平跳变为低电平	置位 LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR
3	GPIO 输入从高电平跳变为低电平，或从低电平跳变为高电平	置位 LP_GPIO_PIN*x*_EDGE_WAKEUP_CLR
4	GPIO 输入为低电平	GPIO 输入为高电平
5	GPIO 输入为高电平	GPIO 输入为低电平

[1] *x* = 0 ~ 15。

[2] 配置为其他值时，LP GPIO 唤醒功能不可用。

10.4.7 编程流程

10.4.7.1 HP GPIO 交换矩阵

把某个 HP 外设输入信号 *Y* 绑定到某个 GPIO 管脚 *X* 的配置过程如下：

1. 在 HP GPIO 交换矩阵中配置外设信号 *Y* 的 GPIO_FUNC*y*_IN_SEL_CFG_REG 寄存器：
   - 置位 GPIO_SIG*y*_IN_SEL 选择通过 HP GPIO 交换矩阵接收外部输入信号。
   - 设置 GPIO_FUNC*y*_IN_SEL 为需要的 GPIO 管脚编号，此处应为 *X*。

   注意：并不是所有的 HP 外设信号都有有效的 GPIO_SIG*y*_IN_SEL 位，即有些外设信号只能通过 HP GPIO 交换矩阵接收外部输入信号。
2. 可选：置位 IO_MUX_GPIO*n*_FILTER_EN 使能 GPIO 管脚的输入信号滤波功能。
3. 启用毛刺滤波器功能，步骤如下：
   - 配置 GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM 为 *m*。此处 *n* 为通道编号，取值范围为：0 ~ 7。*m* 为 GPIO 管脚编号，取值范围为：0 ~ 54。
   - 配置 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 为 **VALUE1**，配置 GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 为 **VALUE2**。在 **VALUE1** + 1 个周期内，如果存在 **VALUE2** + 1 个输入信号与当前输出信号值不匹配，毛刺滤波器硬件将反转输出信号。GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 和 GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 可以配置为相同的值 **VALUE3**，那么只有宽度大于 **VALUE3** + 1 个时钟周期的信号才会被采样。
   - 置位 GPIOSD_FILTER_CH*n*_EN 使能通道 *n*。

   相关示例见图 10.4-3，其中 GPIO_EXT_FILTER_CH*x*_WINDOW_WIDTH 配置为 3 GPIO_EXT_FILTER_CH*x*_WINDOW_THRES 配置为 2。在 T1 之前的四个时钟周期内，输出信号值 (signal_out) 一直为 “0”。输入信号值 (signal_in) 有三个时钟周期为 “1”，则输出信号在 T1 之后反转为 “1”。

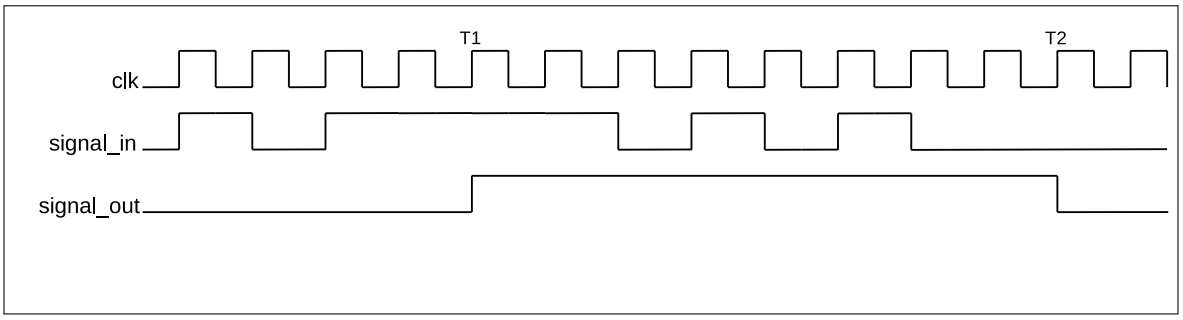


图 10.4-3. 毛刺滤波器时序示例

4. 同步 GPIO 输入信号。配置 GPIO 管脚 *X* 的 GPIO_PIN*x*_REG 来同步 GPIO 输入信号，过程如下：
   - 如图 10.4-1 所示，配置 GPIO_PIN*x*_SYNC1_BYPASS 使能输入信号在第一级同步中为上升沿或下降沿同步。
   - 如图 10.4-1 所示，配置 GPIO_PIN*x*_SYNC2_BYPASS 使能输入信号在第二级同步中为上升沿或下降沿同步。

5. 配置 HP IO MUX 寄存器使能 GPIO 管脚的输入功能。配置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_REG，过程如下：
   - 置位 IO_MUX_GPIO*x*_FUN_IE 使能输入[2]。
   - 置位或清零 IO_MUX_GPIO*x*_FUN_WPU 和 IO_MUX_GPIO*x*_FUN_WPD，使能或关闭内部上拉/下拉电阻。

说明：

1. 同一个输入管脚可以同时绑定多个输入信号。
2. 置位 GPIO_FUNC*y*_IN_INV_SEL 可以把输入信号取反。
3. 无需将输入信号绑定到一个 GPIO 管脚也可以使 HP 外设读取恒低或恒高电平的输入值。实现方式为选择特定的 GPIO_FUNC*y*_IN_SEL 输入值而不是一个 GPIO 序号：
   - 设置 GPIO_FUNC*y*_IN_SEL 为 62，则输入信号恒为 0。
   - 设置 GPIO_FUNC*y*_IN_SEL 为 63，则输入信号恒为 1。

编程示例

要把 UART0 TXD 输入信号[3]（uart0_rxd_pad_in，信号索引号 10）绑定到 GPIO7，请按照以下步骤操作：

1. 置位 GPIO_FUNC10_IN_SEL_CFG_REG 中的 GPIO_SIG10_IN_SEL 位，使能通过 HP GPIO 交换矩阵接收外部输入信号。
2. 配置 GPIO_FUNC10_IN_SEL_CFG_REG 中的 GPIO_FUNC10_IN_SEL 为 7，即选择管脚 GPIO7。
3. 置位 IO_MUX_GPIO7_REG 中 IO_MUX_GPIO7_FUN_IE 位使能管脚输入。

10.4.7.2 LP GPIO 交换矩阵

通过 LP GPIO 交换矩阵进行外设输入的编程过程与章节 10.4.7.1 中的描述相似，区别之处在于：

- LP GPIO 交换矩阵没有毛刺滤波功能。
- 相关的控制寄存器和状态寄存器需要使用 LP GPIO 寄存器，见章节 10.19.4 和章节 10.19.5。

说明：

无需将输入信号绑定到一个 LP GPIO 管脚也可以使 LP 外设读取恒低或恒高电平的输入值。实现方式为选择特定的 LP_GPIO_SIG*y*_IN_SEL 输入值而不是一个 GPIO 序号：

- 设置 LP_GPIO_SIG*y*_IN_SEL 为 16，则输入信号恒为 0。
- 设置 LP_GPIO_SIG*y*_IN_SEL 为 24，则输入信号恒为 1。

10.5 通过 GPIO 交换矩阵的外设输出

10.5.1 概述

通过 GPIO 交换矩阵输出外设信号，一般步骤如下：

- 配置 HP GPIO 交换矩阵将 HP 外设信号（即在表 10.12-1 中“输出信号”一栏所列出的信号）输出到 55 个 GPIO (0 ~ 54) 管脚

- 或配置 LP GPIO 交换矩阵将 LP 外设信号（即在表 10.13-1 中“输出信号”一栏所列出的信号）输出到 16 个 LP GPIO (0 ~ 15) 管脚
- 输出信号从外设输出到 GPIO 交换矩阵，然后到达 IO MUX。IO MUX 必须设置相应管脚为 GPIO 功能，这样输出 GPIO 信号就能连接到相应管脚。

说明：
表 10.12-1 中输出索引号为 250 ~ 255 的外设信号，没有连接至外设，可配置为从一个 GPIO 管脚输出后，直接由另一个 GPIO 管脚输入（索引号：250 ~ 255）。

10.5.2 简单 GPIO 输出

GPIO 交换矩阵也可用于简单 GPIO 输出，即 GPIO 管脚可直接输出所期望的输出值，而无需将某个外设信号绑定到该 GPIO 管脚。

对于 HP GPIO 交换矩阵，具体配置如下：

- 设置 GPIO_FUNC*x*_OUT_SEL 为特定的外设索引值 256 (0x100)。
- 设置 GPIO_OUT_REG/GPIO_OUT1_REG 中相应位的值为期望 GPIO 输出的值。

说明：

- GPIO_OUT_REG[0] ~ GPIO_OUT_REG[31] 对应 GPIO0 ~ GPIO31，GPIO_OUT1_REG[0] ~ GPIO_OUT1_REG[22] 对应 GPIO32 ~ GPIO54。
- 推荐使用相应的 W1TS 和 W1TC 寄存器，例如 GPIO_OUT/OUT1_W1TS 或 GPIO_OUT/OUT1_W1TC 来置位/清除 GPIO_OUT/OUT1_REG。

对于 LP GPIO 交换矩阵，具体配置如下：

- 设置 LP_GPIO_FUNC*x*_OUT_SEL 为特定的外设索引值 32 (0x20)。
- 设置 LP_GPIO_OUT_REG 中相应位的值为期望 GPIO 输出的值。

说明：
推荐使用相应的 W1TS 和 W1TC 寄存器，例如 LP_GPIO_OUT_DATA_W1TS 或 LP_GPIO_OUT_DATA_W1TC 来置位/清除 LP_GPIO_OUT_REG。

10.5.3 Sigma Delta 调制输出 (SDM)

10.5.3.1 功能描述

HP 外设输出信号中有八个信号（在表 10.12-1 中索引为：72 ~ 79）支持 1-bit 二阶 SDM 调制输出。上述八个信号通道默认输出使能。Sigma Delta 调制器可实现输出可配占空比的 PDM（脉冲密度调制）信号。二阶 Sigma Delta 调制器函数为：

$$H(z) = \mathrm{X(z)z^{-1}} + \mathrm{E(z)(1\text{-}z^{-1})^2}$$

E(z) 为量化误差，X(z) 为输入。
Sigma Delta 调制器内部支持对 IO MUX 运行时钟的 1 ~ 256 倍分频：

- 置位 GPIO_EXT_SD_FUNCTION_CLK_EN 使能调制器时钟。
- 配置 GPIO_EXT_SD*n*_PRESCALE 实现分频。*n* 取值范围为 0 ~ 7，对应八个信号通道。

分频后的时钟周期为调制器输出单位脉冲的周期。

GPIO_EXT_SD*n*_IN 为有符号数，范围为 [-128, 127]，配置此寄存器控制输出 PDM 信号的占空比 [1]。

- GPIO_EXT_SD*n*_IN = -128，调制器输出信号占空比为 0%。
- GPIO_EXT_SD*n*_IN = 0，调制器输出信号占空比接近 50%。
- GPIO_EXT_SD*n*_IN = 127，调制器输出信号占空比接近 100%。

PDM 信号占空比计算公式为：

$$Duty_Cycle = \frac{GPIO_EXT_SDn_IN + 128}{256}$$

说明：
对 PDM 信号来说，占空比是指在若干脉冲周期内（比如 256 个脉冲周期），高电平占整个统计周期的比值。

10.5.3.2 配置方法

SDM 的配置方法如下：

- 将 SDM 输出经 HP GPIO 交换矩阵连接至相应管脚，见章节 10.5.4。
- 置位 GPIO_EXT_SD_FUNCTION_CLK_EN，使能 SDM 时钟
- 配置 GPIO_EXT_SD*n*_PRESCALE 设置时钟分频系数。
- 配置 GPIO_EXT_SD*n*_IN 设置 SDM 输出信号的占空比。

10.5.4 编程流程

10.5.4.1 HP GPIO 交换矩阵

如图 10.3-1 所示，表 10.12-1 中“输出信号”列的某一个信号通过 HP GPIO 交换矩阵到达 HP IO MUX，然后连接到某个 GPIO 管脚。

输出外设信号 *Y* 到某一 GPIO 管脚 *X*[1] 的步骤如下：

1. 在 HP GPIO 交换矩阵中配置 GPIO 管脚 *X* 的 GPIO_FUNC*x*_OUT_SEL_CFG_REG 和 GPIO_ENABLE_REG[*x*]。推荐使用相应 W1TS（写 1 置位）和 W1TC（写 1 清零）来更新 GPIO_ENABLE_REG 中的值。
    - 设置 GPIO_FUNC*x*_OUT_SEL_CFG_REG 中的 GPIO_FUNC*x*_OUT_SEL 字段为外设输出信号 Y 的索引号 (Y)。
    - 要将信号强制使能为输出模式，需要将 GPIO 管脚 *X* 对应的 GPIO_FUNC*x*_OUT_SEL_CFG_REG 寄存器中 GPIO_FUNC*x*_OE_SEL 字段置位；同时需要将 GPIO_ENABLE_W1TS_REG 中的相应位置位。或者，将 GPIO_FUNC*x*_OE_SEL 清零，即选择采用外设的输出使能信号，此时输出使能信号由内部逻辑功能决定。比如，表 10.12-1 中“GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号”一栏的 spi2_dqs_pad_oe 信号。

- 置位 GPIO_ENABLE_W1TC_REG 中相应位可以关闭 GPIO 管脚的输出。

2. 要选择以开漏方式输出，可以设置 GPIO 管脚 *X* 的 GPIO_PIN*x*_REG 寄存器中 GPIO_PIN*x*_PAD_DRIVER 位。
3. 配置 HP IO MUX 寄存器来选择经由 HP GPIO 交换矩阵输出信号。配置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_REG 的过程如下：
   - 设置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_MCU_SEL 为所需的管脚功能。此处选择数值 1，即功能 1（GPIO 功能），适用于所有管脚。
   - 设置 IO_MUX_GPIO*x*_FUN_DRV 字段为特定的输出强度值 (0 ~ 3)，值越大，输出驱动能力越强。
   - 在开漏模式下，通过置位/清零 IO_MUX_GPIO*x*_FUN_WPU 和 IO_MUX_GPIO*x*_FUN_WPD 使能或关闭上拉/下拉电阻。
4. 使能迟滞功能：
   - GPIO0 ~ GPIO15：置位 LP_IOMUX_LP_GPIO_HYS[*x*]（x: 0 ~ 15，对应 GPIO0 ~ GPIO15）。
   - GPIO16 ~ GPIO47：置位 HP_SYSTEM_GPIO_0_HYS_LOW[*x*]（x: 0 ~ 31，对应 GPIO16 ~ GPIO47）。
   - GPIO48 ~ GPIO54：置位 HP_SYSTEM_GPIO_0_HYS_HIGH[*x*]（x: 0 ~ 6，对应 GPIO48 ~ GPIO54）。

说明：

- 某一个外设的输出信号可以同时从多个管脚输出。
- 置位 GPIO_FUNC*x*_OUT_INV_SEL 可以把输出的信号取反。

10.5.4.2 LP GPIO 交换矩阵

通过 LP GPIO 交换矩阵进行外设输出的编程过程与章节 10.5.4.1 中的描述相似，区别之处在于：相关的控制寄存器和状态寄存器需要使用 LP GPIO 寄存器，见章节 10.19.4 和章节 10.19.5。

10.6 IO MUX 的直接输入输出功能

10.6.1 概述

SDMMC、EMAC 等数字信号可以旁路 GPIO 交换矩阵以实现更好的高频数字特性。所以此类信号可以直接通过 IO MUX 输入和输出。这样比使用 GPIO 交换矩阵的灵活度要低，即每个 GPIO 管脚的 IO MUX 寄存器只有较少的功能选择，但可以实现更好的高频数字特性。

ESP32-P4 中有 16 个 GPIO 管脚具有低功耗 (LP) 性能。这些管脚可由 HP IO MUX 或 LP IO MUX 控制。如果由 LP IO MUX 控制，这些管脚将旁路 HP IO MUX 和 HP GPIO 交换矩阵，直接供 LP 系统中的外设使用。

当这些管脚被配置为 LP GPIO 管脚，仍然能够在芯片处于 Deep-sleep 模式下由 LP 系统的外设控制，也可以作为唤醒源将芯片从 Deep-sleep 中唤醒。

10.6.2 功能描述

LP 管脚 (GPIO0 ~ GPIO15) 由 LP_IOMUX_PAD*n*_REG 中的 LP_IOMUX_PAD*n*_MUX_SEL (*n* = 0 ~ 15) 位控制。此位默认置为 0，通过 HP IO MUX 输入输出信号。

如果置位 LP_IOMUX_PAD*n*_MUX_SEL，则控制权交由 LP IO MUX 接管。在这种模式下，LP_IOMUX_PAD*n*_REG 寄存器用于控制 LP GPIO 管脚。表 10.15-1 列出了 GPIO 管脚的 LP 功能。请注意 LP_IOMUX_PAD*n*_REG 寄存器使用的是 LP GPIO 管脚的序号，不是 HP GPIO 管脚的序号。

10.6.2.1 HP IO MUX

对于 HP 外设输入信号，旁路 HP GPIO 交换矩阵必须配置以下两个字段：

1. GPIO 管脚的 IO_MUX_GPIO*n*_MCU_SEL 必须配置为相应的管脚功能。表 10.14-1 列出了管脚功能。
2. 清除 GPIO_SIG*n*_IN_SEL，直接将输入信号连接到外设。

对于 HP 外设输出信号，旁路 HP GPIO 交换矩阵只需将 GPIO 管脚的 IO_MUX_GPIO*n*_MCU_SEL 配置为相应的管脚功能即可。

10.6.2.2 LP IO MUX

对于 LP 外设输入信号，旁路 LP GPIO 交换矩阵必须配置以下两个字段：

1. GPIO 管脚的 LP_IOMUX_PAD*n*_MUX_SEL 必须配置为相应的管脚功能。表 10.14-1 列出了管脚功能。
2. 清零 LP_GPIO_SIG*n*_IN_SEL，直接将输入信号连接到外设。

对于 LP 外设输出信号，旁路 LP GPIO 交换矩阵必须将 LP GPIO 管脚的 IP_IOMUX_PAD*n*_MCU_SEL 配置为相应的管脚功能即可。

> **说明：**
> 并非所有外设输入/输出信号均可直接通过 IO MUX 连接到外设，某些输入/输出信号只能通过 GPIO 交换矩阵连接到外设。

10.7 模拟功能描述

10.7.1 概述

ESP32-P4 共有 34 个 GPIO 管脚具有模拟功能，其中包括 16 个 LP GPIO 管脚和 18 个 HP GPIO 管脚。见表 10.16-1。

10.7.2 模拟功能描述

当使用管脚的模拟功能时，需要确保该管脚处于浮空状态，此时外部模拟信号通过 GPIO 管脚直接与内部的模拟信号相连。相关配置如下：

- 清除 IO_MUX_GPIO*n*_FUN_IE、IO_MUX_GPIO*n*_FUN_WPU 和 IO_MUX_GPIO*n*_FUN_WPD 将管脚设置为浮空状态。
- 向 GPIO_ENABLE_W1TC 或 GPIO_ENABLE1_W1TC 中对应位写 1，清除输出使能。

对于表 10.16-1 中列出的不同模拟功能的使用方法，请参考相关章节：

- XTAL_32K 相关功能见章节 15 *低功耗管理*。
- TOUCH 相关功能见章节 63 *触摸传感器 (TOUCH)*。

- ADC 相关功能见章节 65 *ADC 控制器 (ADC)*。
- COMP 相关功能见章节 66 模拟电压比较器。

说明：
表 10.16-1 中 GPIO51 ~ GPIO54 有两种模拟功能，可以同时使用。

10.8 Light-sleep 模式管脚功能

当 ESP32-P4 处于 Light-sleep 模式时管脚可以有不同的功能。如果某一 GPIO 管脚的 IO_MUX_GPIO*n*_REG 寄存器中 IO_MUX_GPIO*n*_SLP_SEL 置位时，芯片处于 Light-sleep 模式下将由另一组不同的寄存器控制管脚。

表 10.8-1. IO MUX Light-sleep 管脚功能控制寄存器

IO MUX 功能	正常工作模式 OR IO_MUX_GPIO*n*_SLP_SEL = 0	Light-sleep 模式 AND IO_MUX_GPIO*n*_SLP_SEL = 1
输出驱动强度	IO_MUX_GPIO*n*_FUN_DRV	IO_MUX_GPIO*n*_MCU_DRV
上拉电阻	IO_MUX_GPIO*n*_FUN_WPU	IO_MUX_GPIO*n*_MCU_WPU
下拉电阻	IO_MUX_GPIO*n*_FUN_WPD	IO_MUX_GPIO*n*_MCU_WPD
输入使能	IO_MUX_GPIO*n*_FUN_IE	IO_MUX_GPIO*n*_MCU_IE
输出使能	由 GPIO 交换矩阵的 OE_SEL*	IO_MUX_GPIO*n*_MCU_OE

说明：
如果 IO_MUX_GPIO*n*_SLP_SEL 置为 0，则芯片在正常工作和 Light-sleep 模式下，管脚的功能一样。此时，具体的输出使能配置请参考 10.5.4 章节。

10.9 GPIO 管脚的 Hold 特性

每个 GPIO 管脚都有单独的 Hold 功能，由电源管理单元 (PMU) 或寄存器控制。管脚的 Hold 功能被置上后，管脚在置上 Hold 那一刻的状态会被强制保持，无论内部信号如何变化，修改 IO MUX 配置或者 GPIO 配置，都不会改变管脚的状态。应用如果希望在看门狗超时触发内核复位时或者 Deep-sleep 触发内核复位时管脚的状态不被改变，就需要提前把 Hold 置上。

HP 管脚 (GPIO16 ~ GPIO54)

每个 HP 管脚的 Hold 状态由各管脚的 Hold 使能信号和全局的 Hold 使能信号经过或运算后共同控制：

- 各管脚的 Hold 使能信号由以下寄存器控制：
  - Deep-sleep 模式下：
    * LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0[*n*] (*n* = 16~31)，控制 GPIO16~GPIO31 各个管脚的 Hold 使能信号。
    * LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL1[*n*] (*n* = 0~22)，控制 GPIO32~GPIO54 各个管脚的 Hold 使能信号。

- 非 Deep-sleep 模式下：
  * HP_SYSTEM_GPIO_O_HOLD_LOW[*n*] (*n* = 0～31)，控制 GPIO16～GPIO47 各个管脚的 Hold 使能信号。
  * HP_SYSTEM_GPIO_O_HOLD_HIGH[*n*] (*n* = 0～6)，控制 GPIO48～GPIO54 各个管脚的 Hold 使能信号。

  说明：
  在非 Deep-sleep 模式下，除了上述 HP_SYSTEM 寄存器外，LP_SYSTEM 寄存器（LP_SYSTEM_REG_PAD_RTC_HOLD_CTRLO/1）同样可以控制对应管脚的 Hold，使能效果一致。

- PMU_TIE_HIGH_HP_PAD_HOLD_ALL，控制 HP 管脚全局的 Hold 使能信号。

要使用此功能，请按照以下步骤操作：

- 若要在 Deep-sleep 中保持管脚输入输出的状态值，需要在掉电之前将寄存器 LP_SYSTEM_REG_PAD_RTC_HOLD_CTRLO[*n*] (*n* = 16～31) 或 LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL1[*n*] (*n* = 0～22) 置位。在芯片被唤醒后，若要关闭 Hold 功能，可将上述两个寄存器清除。
- 若要在非 Deep-sleep 模式下保持管脚输入输出的状态值，需要在掉电之前将寄存器 HP_SYSTEM_GPIO_O_HOLD_LOW[*n*] (*n* = 0～31) 或 HP_SYSTEM_GPIO_O_HOLD_HIGH[*n*] (*n* = 0～6) 置位。在芯片被唤醒后，若要关闭 Hold 功能，可将上述两个寄存器清除。如上文所述，你也可以使用 LP_SYSTEM 寄存器（LP_SYSTEM_REG_PAD_RTC_HOLD_CTRLO/1）来控制对应管脚的 Hold 功能。
- 或者，将 PMU_TIE_HIGH_HP_PAD_HOLD_ALL 置位，用以保持所有 HP 管脚的状态值。在芯片被唤醒后，也可将 PMU_TIE_LOW_HP_PAD_HOLD_ALL 置位，用以关闭所有 HP 管脚的 Hold 功能。
- 或者，也可以在 Deep-sleep 之前，配置 PMU 任务。进入 Deep-sleep 之后，HP 管脚将由 PMU 自动设置 Hold 功能。

LP 管脚 (GPIO0 ～ GPIO15)

每个 LP 管脚的 Hold 状态由各管脚的 Hold 使能信号和全局的 Hold 使能信号经过或运算后共同控制：

- 各管脚的 Hold 使能信号由下面的寄存器控制：
  - Deep-sleep 模式下：LP_SYSTEM_REG_PAD_RTC_HOLD_CTRLO[*n*] (*n* = 0～15)，控制 GPIO0～GPIO15 各个管脚的 Hold 使能信号。
  - 非 Deep-sleep 模式下：LP_IOMUX_LP_PAD_HOLD_REG[*n*] (*n* = 0～15)，控制 GPIO0～GPIO15 各个管脚的 Hold 使能信号。

  说明：
  在非 Deep-sleep 模式下，除了上述 LP_IOMUX 寄存器外，LP_SYSTEM 寄存器（LP_SYSTEM_REG_PAD_RTC_HOLD_CTRLO）同样可以控制对应 LP 管脚的 Hold，使能效果一致。

- PMU_TIE_HIGH_LP_PAD_HOLD_ALL 控制 LP 管脚全局的 Hold 使能信号。

要使用此功能，请按照以下步骤操作：

- 若要在 Deep-sleep 中保持 LP 管脚输入输出的状态值，用户可置位 LP_SYSTEM_REG_PAD_RTC_HOLD_CTRLO[*n*] (*n* = 0～15)（*n* = 0～15，对应管脚 GPIO0～GPIO15），将此

寄存器清零可关闭对应管脚的 Hold 功能。

- 若要在非 Deep-sleep 模式中保持 LP 管脚输入输出的状态值，用户可置位 LP_IOMUX_LP_PAD_HOLD_REG[*n*]，也可将 LP_IOMUX_LP_PAD_HOLD_REG[*n*] 清零以关闭 LP GPIO*n* 的 Hold 功能。如上文所述，你也可以使用 LP_SYSTEM 寄存器（LP_SYSTEM_REG_PAD_RTC_HOLD_CTRL0）来控制对应 LP 管脚的 Hold 功能。
- 或者，也可置位 PMU_TIE_HIGH_LP_PAD_HOLD_ALL 来保持所有 LP 管脚的状态值，也可置位 PMU_TIE_LOW_LP_PAD_HOLD_ALL 关闭所有 LP 管脚的 Hold 功能。
- 或者，也可在 Deep-sleep 之前，配置 PMU 任务。进入 Deep-sleep 之后，LP 管脚将由 PMU 自动设置 Hold 功能。

10.10 GPIO 管脚的迟滞特性

每个 GPIO 管脚都具有迟滞功能。当未使能迟滞功能时，如图 10.10-1 所示，芯片焊盘 (PAD) 上输入给芯片内部的信号 (C) 的电平翻转只有一个阈值（Vt，约 1.7 V）。当 PAD 的电压高于 Vt 时，C 的电平为高，否则为低。但如果 PAD 上的信号有噪声，可能会影响 C 上的信号。

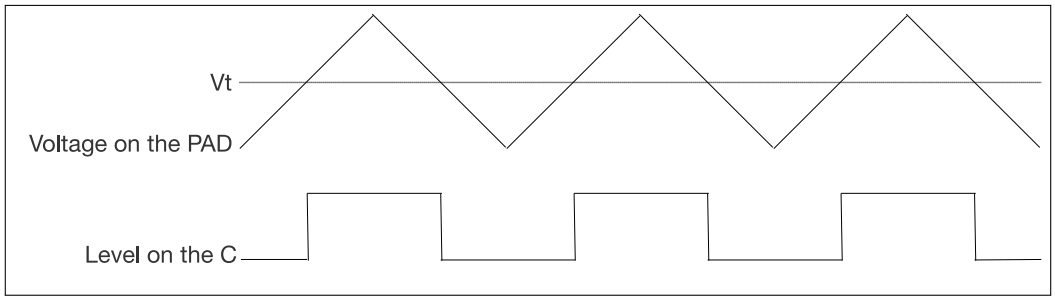


图 10.10-1. 未使能迟滞功能时 PAD 上的电平翻转示例

当使能迟滞功能时，如图 10.10-2 所示，C 的电平翻转有两个阈值，分别为高电平阈值（Vth，约 1.7 V）和低电平阈值（Vtl，约 1.4 V）。当 PAD 的电压由低变高时，如果电压高于 Vth，则 C 的电平为高。当 PAD 的电压由高变低时，若电压低于 Vtl，则 C 的电平为低。当 PAD 的电压在 Vth 和 Vtl 之间时，C 的电平不变。迟滞功能可以减少噪声的影响，起到抗干扰作用，减少 C 的电平翻转次数。

要使能迟滞功能，请按照以下步骤操作：

- 置位 LP_IOMUX_LP_GPIO_HYS[*n*]（*n* = 0 ~ 15，对应 GPIO0 ~ GPIO15），并置位 HP_SYS_GPIO0_HYS_LOW[*n*]（*n* = 0 ~ 31 对应 GPIO16 ~ GPIO47）或置位 HP_SYS_GPIO0_HYS_HIGH[*n*]（*n* = 0 ~ 6，对应 GPIO48 ~ GPIO54），使能 GPIO*n* 的迟滞功能。
- 清零 LP_IOMUX_LP_GPIO_HYS[*n*]（*n* = 0 ~ 15，对应 GPIO0 ~ GPIO15），并清零 HP_SYS_GPIO0_HYS_LOW[*n*]（*n* = 0 ~ 31，对应 GPIO16 ~ GPIO47），或清零 HP_SYS_GPIO0_HYS_HIGH[*n*]（*n* = 0 ~ 6，对应 GPIO48 ~ GPIO54），关闭 GPIO*n* 的迟滞功能。

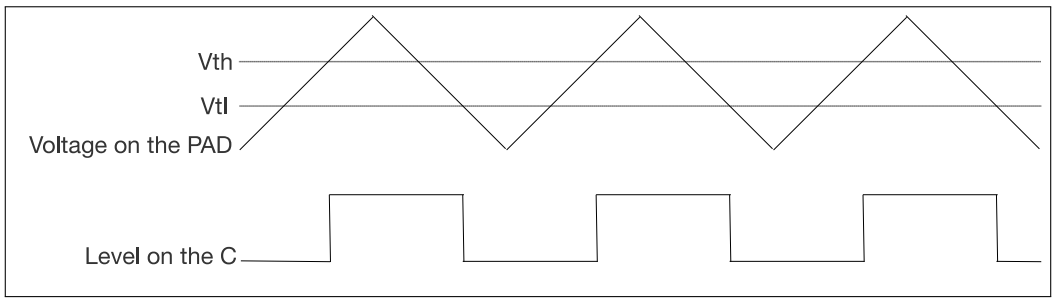


图 10.10-2. 使能迟滞功能时 PAD 上的电平翻转示例

10.11 GPIO 管脚供电和电源管理

10.11.1 GPIO 管脚供电

GPIO 管脚供电请参考《*ESP32-P4 技术规格书*》中管脚定义章节。所有管脚均可用于将芯片从 Light-sleep 中唤醒，但仅有 VDD_LP 域中的管脚 (GPIO0 ~ GPIO15) 可用于将芯片从 Deep-sleep 唤醒。

10.11.2 电源管理

ESP32-P4 的管脚可由如下不同的电源域供电。

- VDD_LP：LP GPIO 管脚的输入电源
- VDD_IO_0、VDD_FLASHIO、VDD_IO_4 ~ VDD_IO_6：HP GPIO 管脚的输入电源

10.12 HP 外设信号列表

表 10.12-1 列出了所有经由 HP GPIO 交换矩阵的外设输入输出信号。

请注意 GPIO_FUNC*n*_OE_SEL 位的配置：

- GPIO_FUNC*n*_OE_SEL = 1，则寄存器 GPIO_ENABLE/ENABLE1_REG 中的相应位 *n* 将用于控制信号输出使能。
  - GPIO_ENABLE/ENABLE1_REG[*n*] = 0：输出关闭。
  - GPIO_ENABLE/ENABLE1_REG[*n*] = 1：输出使能。
- GPIO_FUNC*n*_OE_SEL = 0，则输出信号的使能由外设控制，例如表 10.12-1 中“GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号”一栏的 spi2_dqs_pad_oe。注意，使能信号 spi2_dqs_pad_oe 可设置为 1 (1'd1) 或 0 (1'd0)，具体由外设的配置决定。如果“GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号”一栏中为 1'd1，则表示输出信号默认始终使能。

说明：

信号连续编号，但并非所有信号均有效。

- 表 10.12-1 “输入信号”一栏中有名字的信号均为有效输入信号。
- 表 10.12-1 “输出信号”一栏中有名字的信号均为有效输出信号。

表 10.12-1. HP GPIO 交换矩阵外设信号

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
0	-	-	-	sd_card_cclk_2_pad_out	1'd1	no
1	sd_card_ccmd_2_pad_in	1	no	sd_card_ccmd_2_pad_out	sd_card_ccmd_2_pad_oe	no
2	sd_card_cdata0_2_pad_in	1	no	sd_card_cdata0_2_pad_out	sd_card_cdata0_2_pad_oe	no
3	sd_card_cdata1_2_pad_in	1	no	sd_card_cdata1_2_pad_out	sd_card_cdata1_2_pad_oe	no
4	sd_card_cdata2_2_pad_in	1	no	sd_card_cdata2_2_pad_out	sd_card_cdata2_2_pad_oe	no
5	sd_card_cdata3_2_pad_in	1	no	sd_card_cdata3_2_pad_out	sd_card_cdata3_2_pad_oe	no
6	sd_card_cdata4_2_pad_in	1	no	sd_card_cdata4_2_pad_out	sd_card_cdata4_2_pad_oe	no
7	sd_card_cdata5_2_pad_in	1	no	sd_card_cdata5_2_pad_out	sd_card_cdata5_2_pad_oe	no
8	sd_card_cdata6_2_pad_in	1	no	sd_card_cdata6_2_pad_out	sd_card_cdata6_2_pad_oe	no
9	sd_card_cdata7_2_pad_in	1	no	sd_card_cdata7_2_pad_out	sd_card_cdata7_2_pad_oe	no
10	uart0_rxd_pad_in	0	yes	uart0_txd_pad_out	1'd1	yes
11	uart0_cts_pad_in	0	yes	uart0_rts_pad_out	1'd1	yes
12	uart0_dsr_pad_in	0	no	uart0_dtr_pad_out	1'd1	no
13	uart1_rxd_pad_in	0	yes	uart1_txd_pad_out	1'd1	yes
14	uart1_cts_pad_in	0	yes	uart1_rts_pad_out	1'd1	yes
15	uart1_dsr_pad_in	0	no	uart1_dtr_pad_out	1'd1	no
16	uart2_rxd_pad_in	0	no	uart2_txd_pad_out	1'd1	no
17	uart2_cts_pad_in	0	no	uart2_rts_pad_out	1'd1	no
18	uart2_dsr_pad_in	0	no	uart2_dtr_pad_out	1'd1	no
19	uart3_rxd_pad_in	0	no	uart3_txd_pad_out	1'd1	no
20	uart3_cts_pad_in	0	no	uart3_rts_pad_out	1'd1	no
21	uart3_dsr_pad_in	0	no	uart3_dtr_pad_out	1'd1	no
22	uart4_rxd_pad_in	0	no	uart4_txd_pad_out	1'd1	no
23	uart4_cts_pad_in	0	no	uart4_rts_pad_out	1'd1	no
24	uart4_dsr_pad_in	0	no	uart4_dtr_pad_out	1'd1	no
25	i2s0_o_bck_pad_in	0	no	i2s0_o_bck_pad_out	1'd1	no

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
26	i2s0_mclk_pad_in	0	no	i2s0_mclk_pad_out	1'd1	no
27	i2s0_o_ws_pad_in	0	no	i2s0_o_ws_pad_out	1'd1	no
28	i2s0_i_sd_pad_in	0	no	i2s0_o_sd_pad_out	1'd1	no
29	i2s0_i_bck_pad_in	0	no	i2s0_i_bck_pad_out	1'd1	no
30	i2s0_i_ws_pad_in	0	no	i2s0_i_ws_pad_out	1'd1	no
31	i2s1_o_bck_pad_in	0	no	i2s1_o_bck_pad_out	1'd1	no
32	i2s1_mclk_pad_in	0	no	i2s1_mclk_pad_out	1'd1	no
33	i2s1_o_ws_pad_in	0	no	i2s1_o_ws_pad_out	1'd1	no
34	i2s1_i_sd_pad_in	0	no	i2s1_o_sd_pad_out	1'd1	no
35	i2s1_i_bck_pad_in	0	no	i2s1_i_bck_pad_out	1'd1	no
36	i2s1_i_ws_pad_in	0	no	i2s1_i_ws_pad_out	1'd1	no
37	i2s2_o_bck_pad_in	0	no	i2s2_o_bck_pad_out	1'd1	no
38	i2s2_mclk_pad_in	0	no	i2s2_mclk_pad_out	1'd1	no
39	i2s2_o_ws_pad_in	0	no	i2s2_o_ws_pad_out	1'd1	no
40	i2s2_i_sd_pad_in	0	no	i2s2_o_sd_pad_out	1'd1	no
41	i2s2_i_bck_pad_in	0	no	i2s2_i_bck_pad_out	1'd1	no
42	i2s2_i_ws_pad_in	0	no	i2s2_i_ws_pad_out	1'd1	no
43	i2s0_i_sd1_pad_in	0	no	i2s0_o_sd1_pad_out	1'd1	no
44	i2s0_i_sd2_pad_in	0	no	spi2_dqs_pad_out	spi2_dqs_pad_oe	yes
45	i2s0_i_sd3_pad_in	0	no	spi3_cs2_pad_out	spi3_cs2_pad_oe	no
46	-	-	-	spi3_cs1_pad_out	spi3_cs1_pad_oe	no
47	spi3_ck_pad_in	0	no	spi3_ck_pad_out	spi3_ck_pad_oe	no
48	spi3_q_pad_in	0	no	spi3_qo_pad_out	spi3_q_pad_oe	no
49	spi3_d_pad_in	0	no	spi3_d_pad_out	spi3_d_pad_oe	no
50	spi3_hold_pad_in	0	no	spi3_hold_pad_out	spi3_hold_pad_oe	no
51	spi3_wp_pad_in	0	no	spi3_wp_pad_out	spi3_wp_pad_oe	no
52	spi3_cs_pad_in	0	no	spi3_cs_pad_out	spi3_cs_pad_oe	no

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
53	spi2_ck_pad_in	0	yes	spi2_ck_pad_out	spi2_ck_pad_oe	yes
54	spi2_q_pad_in	0	yes	spi2_q_pad_out	spi2_q_pad_oe	yes
55	spi2_d_pad_in	0	yes	spi2_d_pad_out	spi2_d_pad_oe	yes
56	spi2_hold_pad_in	0	yes	spi2_hold_pad_out	spi2_hold_pad_oe	yes
57	spi2_wp_pad_in	0	yes	spi2_wp_pad_out	spi2_wp_pad_oe	yes
58	spi2_io4_pad_in	0	yes	spi2_io4_pad_out	spi2_io4_pad_oe	yes
59	spi2_io5_pad_in	0	yes	spi2_io5_pad_out	spi2_io5_pad_oe	yes
60	spi2_io6_pad_in	0	yes	spi2_io6_pad_out	spi2_io6_pad_oe	yes
61	spi2_io7_pad_in	0	yes	spi2_io7_pad_out	spi2_io7_pad_oe	yes
62	spi2_cs_pad_in	0	yes	spi2_cs_pad_out	spi2_cs_pad_oe	yes
63	pcnt_rst_pad_in0	0	no	spi2_cs1_pad_out	spi2_cs1_pad_oe	no
64	pcnt_rst_pad_in1	0	no	spi2_cs2_pad_out	spi2_cs2_pad_oe	no
65	pcnt_rst_pad_in2	0	no	spi2_cs3_pad_out	spi2_cs3_pad_oe	no
66	pcnt_rst_pad_in3	0	no	spi2_cs4_pad_out	spi2_cs4_pad_oe	no
67	-	-	-	spi2_cs5_pad_out	spi2_cs5_pad_oe	no
68	i2c0_scl_pad_in	1	no	i2c0_scl_pad_out	i2c0_scl_oe_pad_out	no
69	i2c0_sda_pad_in	1	no	i2c0_sda_pad_out	i2c0_sda_oe_pad_out	no
70	i2c1_scl_pad_in	1	no	i2c1_scl_pad_out	i2c1_scl_oe_pad_out	no
71	i2c1_sda_pad_in	1	no	i2c1_sda_pad_out	i2c1_sda_oe_pad_out	no
72	-	-	-	gpio_sd0_out	1'd1	no
73	-	-	-	gpio_sd1_out	1'd1	no
74	uart0_slp_clk_pad_in	0	no	gpio_sd2_out	1'd1	no
75	uart1_slp_clk_pad_in	0	no	gpio_sd3_out	1'd1	no
76	uart2_slp_clk_pad_in	0	no	gpio_sd4_out	1'd1	no
77	uart3_slp_clk_pad_in	0	no	gpio_sd5_out	1'd1	no
78	uart4_slp_clk_pad_in	0	no	gpio_sd6_out	1'd1	no
79	-	-	-	gpio_sd7_out	1'd1	no

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
80	twai0_rx_pad_in	1	no	twai0_tx_pad_out	1'd1	no
81	-	-	-	twai0_bus_off_on_pad_out	1'd1	no
82	-	-	-	twai0_clkout_pad_out	1'd1	no
83	twai1_rx_pad_in	1	no	twai1_tx_pad_out	1'd1	no
84	-	-	-	twai1_bus_off_on_pad_out	1'd1	no
85	-	-	-	twai1_clkout_pad_out	1'd1	no
86	twai2_rx_pad_in	1	no	twai2_tx_pad_out	1'd1	no
87	-	-	-	twai2_bus_off_on_pad_out	1'd1	no
88	-	-	-	twai2_clkout_pad_out	1'd1	no
89	pwm0_sync0_pad_in	0	no	pwm0_ch0_a_pad_out	1'd1	no
90	pwm0_sync1_pad_in	0	no	pwm0_ch0_b_pad_out	1'd1	no
91	pwm0_sync2_pad_in	0	no	pwm0_ch1_a_pad_out	1'd1	no
92	pwm0_f0_pad_in	0	no	pwm0_ch1_b_pad_out	1'd1	no
93	pwm0_f1_pad_in	0	no	pwm0_ch2_a_pad_out	1'd1	no
94	pwm0_f2_pad_in	0	no	pwm0_ch2_b_pad_out	1'd1	no
95	pwm0_cap0_pad_in	0	no	pwm1_ch0_a_pad_out	1'd1	no
96	pwm0_cap1_pad_in	0	no	pwm1_ch0_b_pad_out	1'd1	no
97	pwm0_cap2_pad_in	0	no	pwm1_ch1_a_pad_out	1'd1	no
98	pwm1_sync0_pad_in	0	no	pwm1_ch1_b_pad_out	1'd1	no
99	pwm1_sync1_pad_in	0	no	pwm1_ch2_a_pad_out	1'd1	no
100	pwm1_sync2_pad_in	0	no	pwm1_ch2_b_pad_out	1'd1	no
101	pwm1_f0_pad_in	0	no	-	-	-
102	pwm1_f1_pad_in	0	no	-	-	-
103	pwm1_f2_pad_in	0	no	-	-	-
104	pwm1_cap0_pad_in	0	no	-	-	-
105	pwm1_cap1_pad_in	0	no	twai0_standby_pad_out	1'd1	no
106	pwm1_cap2_pad_in	0	no	twai1_standby_pad_out	1'd1	no

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
107	gmii_mdi_pad_in	0	no	twai2_standby_pad_out	1'd1	no
108	gmac_phy_col_pad_in	0	no	gmii_mdc_pad_out	1'd1	no
109	gmac_phy_crs_pad_in	0	no	gmii_mdo_pad_out	gmii_mdo_oe_pad_out	no
110	usb_otg11_iddig_pad_in	0	no	usb_srp_dischrgvbus_pad_out	1'd1	no
111	usb_otg11_avalid_pad_in	0	no	usb_otg11_idpullup_pad_out	1'd1	no
112	usb_srp_bvalid_pad_in	0	no	usb_otg11_dppulldown_pad_out	1'd1	no
113	usb_otg11_vbusvalid_pad_in	0	no	usb_otg11_dmpulldown_pad_out	1'd1	no
114	usb_srp_sessend_pad_in	0	no	usb_otg11_drvvbus_pad_out	1'd1	no
115	-	-	-	usb_srp_chrgvbus_pad_out	1'd1	no
116	-	-	-	-	-	-
117	ulpi_clk_pad_in	0	no	-	-	-
118	usb_hsphy_refclk_in	0	no	-	-	-
119	-	-	-	-	-	-
120	-	-	-	-	-	-
121	-	-	-	-	-	-
122	-	-	-	-	-	-
123	-	-	-	-	-	-
124	-	-	-	-	-	-
125	-	-	-	-	-	-
126	sd_card_detect_n_1_pad_in	0	no	ledc_ls_sig_out_pad_out0	1'd1	no
127	sd_card_detect_n_2_pad_in	0	no	ledc_ls_sig_out_pad_out1	1'd1	no
128	sd_card_int_n_1_pad_in	1	no	ledc_ls_sig_out_pad_out2	1'd1	no
129	sd_card_int_n_2_pad_in	1	no	ledc_ls_sig_out_pad_out3	1'd1	no
130	sd_card_write_prt_1_pad_in	0	no	ledc_ls_sig_out_pad_out4	1'd1	no

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
131	sd_card_write_prt_2_pad_in	0	no	ledc_ls_sig_out_pad_out5	1'd1	no
132	sd_data_strobe_1_pad_in	0	no	ledc_ls_sig_out_pad_out6	1'd1	no
133	sd_data_strobe_2_pad_in	0	no	ledc_ls_sig_out_pad_out7	1'd1	no
134	i3c_mst_scl_pad_in	1	no	i3c_mst_scl_pad_out	i3c_mst_scl_oe_pad_out	no
135	i3c_mst_sda_pad_in	1	no	i3c_mst_sda_pad_out	i3c_mst_sda_oe_pad_out	no
136	i3c_slv_scl_pad_in	1	no	i3c_slv_scl_pad_out	i3c_slv_scl_oe_pad_out	no
137	i3c_slv_sda_pad_in	1	no	i3c_slv_sda_pad_out	i3c_slv_sda_oe_pad_out	no
138	-	-	-	i3c_mst_scl_pullup_en_pad_out	1'd1	no
139	-	-	-	i3c_mst_sda_pullup_en_pad_out	1'd1	no
140	usb_jtag_tdo_bridge_pad_in	0	no	usb_jtag_tdi_bridge_pad_out	1'd1	no
141	pcnt_sig_ch0_pad_in0	0	no	usb_jtag_tms_bridge_pad_out	1'd1	no
142	pcnt_sig_ch0_pad_in1	0	no	usb_jtag_tck_bridge_pad_out	1'd1	no
143	pcnt_sig_ch0_pad_in2	0	no	usb_jtag_trst_bridge_pad_out	1'd1	no
144	pcnt_sig_ch0_pad_in3	0	no	lcd_cs_pad_out	1'd1	no
145	pcnt_sig_ch1_pad_in0	0	no	lcd_dc_pad_out	1'd1	no
146	pcnt_sig_ch1_pad_in1	0	no	sd_rst_n_1_pad_out	1'd1	no
147	pcnt_sig_ch1_pad_in2	0	no	sd_rst_n_2_pad_out	1'd1	no
148	pcnt_sig_ch1_pad_in3	0	no	sd_ccmd_od_pullup_en_n_pad_out	1'd1	no
149	pcnt_ctrl_ch0_pad_in0	0	no	lcd_pclk_pad_out	1'd1	no
150	pcnt_ctrl_ch0_pad_in1	0	no	cam_clk_pad_out	1'd1	no
151	pcnt_ctrl_ch0_pad_in2	0	no	lcd_h_enable_pad_out	1'd1	no
152	pcnt_ctrl_ch0_pad_in3	0	no	lcd_h_sync_pad_out	1'd1	no
153	pcnt_ctrl_ch1_pad_in0	0	no	lcd_v_sync_pad_out	1'd1	no

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
154	pcnt_ctrl_ch1_pad_in1	0	no	lcd_data_out_pad_out0	1'd1	no
155	pcnt_ctrl_ch1_pad_in2	0	no	lcd_data_out_pad_out1	1'd1	no
156	pcnt_ctrl_ch1_pad_in3	0	no	lcd_data_out_pad_out2	1'd1	no
157	-	-	-	lcd_data_out_pad_out3	1'd1	no
158	cam_pclk_pad_in	0	no	lcd_data_out_pad_out4	1'd1	no
159	cam_h_enable_pad_in	0	no	lcd_data_out_pad_out5	1'd1	no
160	cam_h_sync_pad_in	0	no	lcd_data_out_pad_out6	1'd1	no
161	cam_v_sync_pad_in	0	no	lcd_data_out_pad_out7	1'd1	no
162	cam_data_in_pad_in0	0	no	lcd_data_out_pad_out8	1'd1	no
163	cam_data_in_pad_in1	0	no	lcd_data_out_pad_out9	1'd1	no
164	cam_data_in_pad_in2	0	no	lcd_data_out_pad_out10	1'd1	no
165	cam_data_in_pad_in3	0	no	lcd_data_out_pad_out11	1'd1	no
166	cam_data_in_pad_in4	0	no	lcd_data_out_pad_out12	1'd1	no
167	cam_data_in_pad_in5	0	no	lcd_data_out_pad_out13	1'd1	no
168	cam_data_in_pad_in6	0	no	lcd_data_out_pad_out14	1'd1	no
169	cam_data_in_pad_in7	0	no	lcd_data_out_pad_out15	1'd1	no
170	cam_data_in_pad_in8	0	no	lcd_data_out_pad_out16	1'd1	no
171	cam_data_in_pad_in9	0	no	lcd_data_out_pad_out17	1'd1	no
172	cam_data_in_pad_in10	0	no	lcd_data_out_pad_out18	1'd1	no
173	cam_data_in_pad_in11	0	no	lcd_data_out_pad_out19	1'd1	no
174	cam_data_in_pad_in12	0	no	lcd_data_out_pad_out20	1'd1	no
175	cam_data_in_pad_in13	0	no	lcd_data_out_pad_out21	1'd1	no
176	cam_data_in_pad_in14	0	no	lcd_data_out_pad_out22	1'd1	no
177	cam_data_in_pad_in15	0	no	lcd_data_out_pad_out23	1'd1	no
178	gmac_phy_rxdv_pad_in	0	yes	gmac_phy_txen_pad_out	1'd1	yes
179	gmac_phy_rxd0_pad_in	0	yes	gmac_phy_txd0_pad_out	1'd1	yes
180	gmac_phy_rxd1_pad_in	0	yes	gmac_phy_txd1_pad_out	1'd1	yes

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
181	gmac_phy_rxd2_pad_in	0	no	gmac_phy_txd2_pad_out	1'd1	no
182	gmac_phy_rxd3_pad_in	0	no	gmac_phy_txd3_pad_out	1'd1	no
183	gmac_phy_rxer_pad_in	0	yes	gmac_phy_txer_pad_out	1'd1	yes
184	gmac_rx_clk_pad_in	0	no	-	-	-
185	gmac_tx_clk_pad_in	0	no	-	-	-
186	parlio_rx_clk_pad_in	0	no	parlio_rx_clk_pad_out	1'd1	no
187	parlio_tx_clk_pad_in	0	no	parlio_tx_clk_pad_out	1'd1	no
188	parlio_rx_data0_pad_in	0	no	parlio_tx_data0_pad_out	1'd1	no
189	parlio_rx_data1_pad_in	0	no	parlio_tx_data1_pad_out	1'd1	no
190	parlio_rx_data2_pad_in	0	no	parlio_tx_data2_pad_out	1'd1	no
191	parlio_rx_data3_pad_in	0	no	parlio_tx_data3_pad_out	1'd1	no
192	parlio_rx_data4_pad_in	0	no	parlio_tx_data4_pad_out	1'd1	no
193	parlio_rx_data5_pad_in	0	no	parlio_tx_data5_pad_out	1'd1	no
194	parlio_rx_data6_pad_in	0	no	parlio_tx_data6_pad_out	1'd1	no
195	parlio_rx_data7_pad_in	0	no	parlio_tx_data7_pad_out	1'd1	no
196	parlio_rx_data8_pad_in	0	no	parlio_tx_data8_pad_out	1'd1	no
197	parlio_rx_data9_pad_in	0	no	parlio_tx_data9_pad_out	1'd1	no
198	parlio_rx_data10_pad_in	0	no	parlio_tx_data10_pad_out	1'd1	no
199	parlio_rx_data11_pad_in	0	no	parlio_tx_data11_pad_out	1'd1	no
200	parlio_rx_data12_pad_in	0	no	parlio_tx_data12_pad_out	1'd1	no
201	parlio_rx_data13_pad_in	0	no	parlio_tx_data13_pad_out	1'd1	no
202	parlio_rx_data14_pad_in	0	no	parlio_tx_data14_pad_out	1'd1	no
203	parlio_rx_data15_pad_in	0	no	parlio_tx_data15_pad_out	1'd1	no
204	-	-	-	-	-	-
205	-	-	-	-	-	-
206	-	-	-	-	-	-
207	-	-	-	-	-	-

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
208	-	-	-	-	-	-
209	-	-	-	-	-	-
210	-	-	-	-	-	-
211	-	-	-	-	-	-
212	-	-	-	constant0_pad_out	1'd1	no
213	-	-	-	constant1_pad_out	1'd1	no
214	core_gpio_in_pad_in0	0	no	core_gpio_out_pad_out0	core_gpio_oe_pad_out0	no
215	core_gpio_in_pad_in1	0	no	core_gpio_out_pad_out1	core_gpio_oe_pad_out1	no
216	core_gpio_in_pad_in2	0	no	core_gpio_out_pad_out2	core_gpio_oe_pad_out2	no
217	core_gpio_in_pad_in3	0	no	core_gpio_out_pad_out3	core_gpio_oe_pad_out3	no
218	core_gpio_in_pad_in4	0	no	core_gpio_out_pad_out4	core_gpio_oe_pad_out4	no
219	core_gpio_in_pad_in5	0	no	core_gpio_out_pad_out5	core_gpio_oe_pad_out5	no
220	core_gpio_in_pad_in6	0	no	core_gpio_out_pad_out6	core_gpio_oe_pad_out6	no
221	core_gpio_in_pad_in7	0	no	core_gpio_out_pad_out7	core_gpio_oe_pad_out7	no
222	core_gpio_in_pad_in8	0	no	core_gpio_out_pad_out8	core_gpio_oe_pad_out8	no
223	core_gpio_in_pad_in9	0	no	core_gpio_out_pad_out9	core_gpio_oe_pad_out9	no
224	core_gpio_in_pad_in10	0	no	core_gpio_out_pad_out10	core_gpio_oe_pad_out10	no
225	core_gpio_in_pad_in11	0	no	core_gpio_out_pad_out11	core_gpio_oe_pad_out11	no
226	core_gpio_in_pad_in12	0	no	core_gpio_out_pad_out12	core_gpio_oe_pad_out12	no
227	core_gpio_in_pad_in13	0	no	core_gpio_out_pad_out13	core_gpio_oe_pad_out13	no
228	core_gpio_in_pad_in14	0	no	core_gpio_out_pad_out14	core_gpio_oe_pad_out14	no
229	core_gpio_in_pad_in15	0	no	core_gpio_out_pad_out15	core_gpio_oe_pad_out15	no
230	core_gpio_in_pad_in16	0	no	core_gpio_out_pad_out16	core_gpio_oe_pad_out16	no
231	core_gpio_in_pad_in17	0	no	core_gpio_out_pad_out17	core_gpio_oe_pad_out17	no
232	core_gpio_in_pad_in18	0	no	core_gpio_out_pad_out18	core_gpio_oe_pad_out18	no
233	core_gpio_in_pad_in19	0	no	core_gpio_out_pad_out19	core_gpio_oe_pad_out19	no
234	core_gpio_in_pad_in20	0	no	core_gpio_out_pad_out20	core_gpio_oe_pad_out20	no

信号索引	输入信号	默认值	信号可经由 HP IO MUX 直接输入输出	输出信号	GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 HP IO MUX 直接输出
235	core_gpio_in_pad_in21	0	no	core_gpio_out_pad_out21	core_gpio_oe_pad_out21	no
236	core_gpio_in_pad_in22	0	no	core_gpio_out_pad_out22	core_gpio_oe_pad_out22	no
237	core_gpio_in_pad_in23	0	no	core_gpio_out_pad_out23	core_gpio_oe_pad_out23	no
238	core_gpio_in_pad_in24	0	no	core_gpio_out_pad_out24	core_gpio_oe_pad_out24	no
239	core_gpio_in_pad_in25	0	no	core_gpio_out_pad_out25	core_gpio_oe_pad_out25	no
240	core_gpio_in_pad_in26	0	no	core_gpio_out_pad_out26	core_gpio_oe_pad_out26	no
241	core_gpio_in_pad_in27	0	no	core_gpio_out_pad_out27	core_gpio_oe_pad_out27	no
242	core_gpio_in_pad_in28	0	no	parlio_tx_cs_pad_out	1'd1	no
243	core_gpio_in_pad_in29	0	no	emac_ptp_pps_pad_out	1'd1	no
244	core_gpio_in_pad_in30	0	no	ana_comp0_out	1'd1	no
245	core_gpio_in_pad_in31	0	no	ana_comp1_out	1'd1	no
246	rmt_sig_pad_in0	0	no	rmt_sig_pad_out0	1'd1	no
247	rmt_sig_pad_in1	0	no	rmt_sig_pad_out1	1'd1	no
248	rmt_sig_pad_in2	0	no	rmt_sig_pad_out2	1'd1	no
249	rmt_sig_pad_in3	0	no	rmt_sig_pad_out3	1'd1	no
250	sig_in_func250	0	no	sig_in_func250	1'd1	no
251	sig_in_func251	0	no	sig_in_func251	1'd1	no
252	sig_in_func252	0	no	sig_in_func252	1'd1	no
253	sig_in_func253	0	no	sig_in_func253	1'd1	no
254	sig_in_func254	0	no	sig_in_func254	1'd1	no
255	sig_in_func255	0	no	sig_in_func255	1'd1	no

10.13 LP 外设信号列表

表 10.13-1 列出了所有经由 LP GPIO 交换矩阵的外设输入输出信号。

请注意 LP_GPIO_FUNC*n*_OE_SEL 位的配置：

- LP_GPIO_FUNC*n*_OE_SEL = 1，则寄存器 LP_GPIO_ENABLE_REG 中的相应位 *n* 将用于控制信号输出使能。
  - LP_GPIO_ENABLE_REG[*n*] = 0：输出关闭。
  - LP_GPIO_ENABLE_REG[*n*] = 1：输出使能。
- LP_GPIO_FUNC*n*_OE_SEL = 0，则输出信号的使能由外设控制，例如表 10.13-1 中 “LP_GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号” 一栏的 lp_spi_ck_pad_oe。注意，使能信号 lp_spi_ck_pad_oe 可设置为 1 (1'd1) 或 0 (1'd0)，具体由外设的配置决定。如果 “LP_GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号” 一栏中为 1'd1，则表示输出信号默认始终使能。

说明：

信号连续编号，但并非所有信号均有效。

- 表 10.13-1 “输入信号” 一栏中有名字的信号均为有效输入信号。
- 表 10.13-1 “输出信号” 一栏中有名字的信号均为有效输出信号。

表 10.13-1. LP GPIO 交换矩阵外设信号

信号索引	输入信号	默认值	信号可经由 LP IO MUX 直接输入输出	输出信号	LP_GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 LP IO MUX 直接输出
0	lp_i2c_scl_pad_in	1	no	lp_i2c_scl_pad_out	lp_i2c_scl_pad_oe	no
1	lp_i2c_sda_pad_in	1	no	lp_i2c_sda_pad_out	lp_i2c_sda_pad_oe	no
2	lp_uart_rxd_pad_in	0	yes	lp_uart_txd_pad_out	1'd1	yes
3	lp_uart_ctsn_pad_in	1	no	lp_uart_rtsn_pad_out	1'd1	no
4	lp_uart_dsrn_pad_in	1	no	lp_uart_dtrn_pad_out	1'd1	no
5	lp_spi_ck_pad_in	0	no	lp_spi_ck_pad_out	lp_spi_ck_pad_oe	no
6	lp_spi_cs_pad_in	0	no	lp_spi_cs_pad_out	lp_spi_cs_pad_oe	no
7	lp_spi_d_pad_in	0	no	lp_spi_d_pad_out	lp_spi_d_pad_oe	no
8	lp_spi_q_pad_in	0	no	lp_spi_q_pad_out	lp_spi_q_pad_oe	no
9	lp_i2s_i_bck_pad_in	0	no	lp_i2s_i_bck_pad_out	1'd1	no
10	lp_i2s_i_sd_pad_in	0	no	lp_i2s_o_sd_pad_out	1'd1	no
11	lp_i2s_i_ws_pad_in	0	no	lp_i2s_i_ws_pad_out	1'd1	no
12	lp_i2s_o_bck_pad_in	0	no	lp_i2s_o_bck_pad_out	1'd1	no
13	lp_i2s_o_ws_pad_in	0	no	lp_i2s_o_ws_pad_out	1'd1	no
14	-	-	-	-	-	-
15	-	-	-	-	-	-
16	-	-	-	-	-	-
17	-	-	-	-	-	-
18	-	-	-	-	-	-
19	-	-	-	-	-	-
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入输出	输出信号	LP_GPIO_FUNCn_OE_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
23	-	-	-	-	-	-
24	-	-	-	-	-	-
25	-	-	-	-	-	-
26	-	-	-	-	-	-
27	-	-	-	-	-	-
28	-	-	-	-	-	-
29	-	-	-	-	-	-
30	-	-	-	-	-	-
31	-	-	-	-	-	-

10.14 HP IO MUX 管脚功能列表

表 10.14-1 列出了每个 GPIO 管脚的 HP IO MUX 功能及其默认状态。

表 10.14-1. HP IO MUX 管脚功能

名称	功能 0	功能 1	功能 2	功能 3	驱动强度	复位	说明
GPIO0	GPIO0	GPIO0	-	-	2	0	R
GPIO1	GPIO1	GPIO1	-	-	2	0	R
GPIO2	MTCK	GPIO2	-	-	2	1*	R
GPIO3	MTDI	GPIO3	-	-	2	1	R
GPIO4	MTMS	GPIO4	-	-	2	1	R
GPIO5	MTDO	GPIO5	-	-	2	0	R

见下页

表 10.14-1 – 接上页

名称	功能 0	功能 1	功能 2	功能 3	驱动强度	复位	说明
GPIO6	GPIO6	GPIO6	-	SPI2_HOLD_PAD	2	0	R
GPIO7	GPIO7	GPIO7	-	SPI2_CS_PAD	2	0	R
GPIO8	GPIO8	GPIO8	UART0_RTS_PAD	SPI2_D_PAD	2	0	R
GPIO9	GPIO9	GPIO9	UART0_CTS_PAD	SPI2_CK_PAD	2	0	R
GPIO10	GPIO10	GPIO10	UART1_TXD_PAD	SPI2_Q_PAD	2	0	R
GPIO11	GPIO11	GPIO11	UART1_RXD_PAD	SPI2_WP_PAD	2	0	R
GPIO12	GPIO12	GPIO12	UART1_RTS_PAD	-	2	0	R
GPIO13	GPIO13	GPIO13	UART1_CTS_PAD	-	2	0	R
GPIO14	GPIO14	GPIO14	—	-	2	0	R
GPIO15	GPIO15	GPIO15	—	-	2	0	R
GPIO16	GPIO16	GPIO16	—	-	2	0	R
GPIO17	GPIO17	GPIO17	—	-	2	0	R
GPIO18	GPIO18	GPIO18	—	-	2	0	R
GPIO19	GPIO19	GPIO19	—	-	2	0	R
GPIO20	GPIO20	GPIO20	—	-	2	0	R
GPIO21	GPIO21	GPIO21	—	-	2	0	R
GPIO22	GPIO22	GPIO22	—	-	2	0	R
GPIO23	GPIO23	GPIO23	—	REF_50M_CLK_PAD	2	0	R
GPIO24	GPIO24	GPIO24	—	-	3	0	R, USB
GPIO25	GPIO25	GPIO25	—	-	3	3*	R, USB
GPIO26	GPIO26	GPIO26	—	-	2	0	R, USB
GPIO27	GPIO27	GPIO27	—	-	2	0	R, USB
GPIO28	GPIO28	GPIO28	SPI2_CS_PAD	GMAC_PHY_RXDV_PAD	2	0	—
GPIO29	GPIO29	GPIO29	SPI2_D_PAD	GMAC_PHY_RXD0_PAD	2	0	—
GPIO30	GPIO30	GPIO30	SPI2_CK_PAD	GMAC_PHY_RXD1_PAD	2	0	—
GPIO31	GPIO31	GPIO31	SPI2_Q_PAD	GMAC_PHY_RXER_PAD	2	0	—

见下页

GoBack

表 10.14-1 – 接上页

名称	功能 0	功能 1	功能 2	功能 3	驱动强度	复位	说明
GPIO32	GPIO32	GPIO32	SPI2_HOLD_PAD	GMAC_RMII_CLK_PAD	2	0	—
GPIO33	GPIO33	GPIO33	SPI2_WP_PAD	GMAC_PHY_TXEN_PAD	2	0	—
GPIO34	GPIO34	GPIO34	SPI2_IO4_PAD	GMAC_PHY_TXD0_PAD	2	0	—
GPIO35	GPIO35	GPIO35	SPI2_IO5_PAD	GMAC_PHY_TXD1_PAD	2	0	—
GPIO36	GPIO36	GPIO36	SPI2_IO6_PAD	GMAC_PHY_TXER_PAD	2	0	—
GPIO37	UART0_TXD_PAD	GPIO37	SPI2_IO7_PAD	-	2	5	—
GPIO38	UART0_RXD_PAD	GPIO38	SPI2_DQS_PAD	-	2	0	—
GPIO39	SD1_CDATA0_PAD	GPIO39	—	REF_50M_CLK_PAD	2	0	—
GPIO40	SD1_CDATA1_PAD	GPIO40	—	GMAC_PHY_TXEN_PAD	2	0	—
GPIO41	SD1_CDATA2_PAD	GPIO41	—	GMAC_PHY_TXD0_PAD	2	0	—
GPIO42	SD1_CDATA3_PAD	GPIO42	—	GMAC_PHY_TXD1_PAD	2	0	—
GPIO43	SD1_CCLK_PAD	GPIO43	—	GMAC_PHY_TXER_PAD	2	0	—
GPIO44	SD1_CCMD_PAD	GPIO44	—	GMAC_RMII_CLK_PAD	2	0	—
GPIO45	SD1_CDATA4_PAD	GPIO45	—	GMAC_PHY_RXDV_PAD	2	0	—
GPIO46	SD1_CDATA5_PAD	GPIO46	—	GMAC_PHY_RXD0_PAD	2	0	—
GPIO47	SD1_CDATA6_PAD	GPIO47	—	GMAC_PHY_RXD1_PAD	2	0	—
GPIO48	SD1_CDATA7_PAD	GPIO48	—	GMAC_PHY_RXER_PAD	2	0	—
GPIO49	GPIO49	GPIO49	—	GMAC_PHY_TXEN_PAD	2	0	R
GPIO50	GPIO50	GPIO50	—	GMAC_RMII_CLK_PAD	2	0	R
GPIO51	GPIO51	GPIO51	—	GMAC_PHY_RXDV_PAD	2	0	R
GPIO52	GPIO52	GPIO52	—	GMAC_PHY_RXD0_PAD	2	0	R
GPIO53	GPIO53	GPIO53	—	GMAC_PHY_RXD1_PAD	2	0	R
GPIO54	GPIO54	GPIO54	—	GMAC_PHY_RXER_PAD	2	0	R

驱动强度

“驱动强度”一栏所示为每个管脚复位后的默认驱动强度。

- **0** - 驱动电流 = ~5 mA
- **1** - 驱动电流 = ~10 mA
- **2** - 驱动电流 = ~20 mA
- **3** - 驱动电流 = ~40 mA

复位

“复位”一栏所示为每个管脚复位后的默认配置。

- **0** - 输入关闭，高阻 (IE = 0)
- **1** - 输入使能，高阻 (IE = 1)
- **1*** - eFuse 的 EFUSE_DIS_PAD_JTAG 位为
  0 时（初始默认值），管脚复位后输入使能，上拉电阻使能 (IE = 1, WPU = 1)
  1 时，管脚复位后输入关闭，高阻 (IE = 0)
- **3*** - 输入关闭，上拉电阻使能 (IE = 0, USB_PU = 1)，具体见说明。
- **5** - 输入关闭 (IE = 0)，输出由功能 0 的外设控制，默认输出 1。

建议对处于高阻态的管脚配置上拉或下拉，以避免不必要的耗电。用户可在 PCB 设计中实现上下拉，或在软件初始化时开启管脚自带的上下拉。

说明

- **R** - 管脚具有模拟功能。
- **USB** - USB 管脚的上拉电阻由管脚上拉和 USB 上拉共同控制，当其中任意一个为 1 时，对应管脚上拉电阻使能。

注意：
HP IO MUX 中，没有用到的管脚需要配置成 GPIO 功能。

10.15 LP IO MUX 管脚功能列表

表 10.15-1 列出了每个 GPIO 管脚的 LP IO MUX 功能。GPIO 管脚默认由 HP IO MUX 控制，因此表 10.15-1 仅列出了管脚的 LP IO MUX 功能。

表 10.15-1. LP IO MUX 管脚功能

名称	功能 0	功能 1
GPIO0	LP_GPIO0	LP_GPIO0
GPIO1	LP_GPIO1	LP_GPIO1
GPIO2	LP_GPIO2	LP_GPIO2
GPIO3	LP_GPIO3	LP_GPIO3
GPIO4	LP_GPIO4	LP_GPIO4

名称	功能 0	功能 1
GPIO5	LP_GPIO5	LP_GPIO5
GPIO6	LP_GPIO6	LP_GPIO6
GPIO7	LP_GPIO7	LP_GPIO7
GPIO8	LP_GPIO8	LP_GPIO8
GPIO9	LP_GPIO9	LP_GPIO9
GPIO10	LP_GPIO10	LP_GPIO10
GPIO11	LP_GPIO11	LP_GPIO11
GPIO12	LP_GPIO12	LP_GPIO12
GPIO13	LP_GPIO13	LP_GPIO13
GPIO14	LP_UART_TXD_PAD	LP_GPIO14
GPIO15	LP_UART_RXD_PAD	LP_GPIO15

注意：
LP IO MUX 中，没有用到的管脚需要配置成 LP_GPIO 功能。

10.16 GPIO 管脚模拟功能列表

表 10.16-1 列出了 GPIO 管脚及其相应的模拟功能。

表 10.16-1. GPIO 管脚模拟功能

名称	模拟功能 0	模拟功能 1
GPIO0	XTAL_32K_N	—
GPIO1	XTAL_32K_P	—
GPIO2	TOUCH_CHANNEL0	—
GPIO3	TOUCH_CHANNEL1	—
GPIO4	TOUCH_CHANNEL2	—
GPIO5	TOUCH_CHANNEL3	—
GPIO6	TOUCH_CHANNEL4	—
GPIO7	TOUCH_CHANNEL5	—
GPIO8	TOUCH_CHANNEL6	—
GPIO9	TOUCH_CHANNEL7	—
GPIO10	TOUCH_CHANNEL8	—
GPIO11	TOUCH_CHANNEL9	—
GPIO12	TOUCH_CHANNEL10	—
GPIO13	TOUCH_CHANNEL11	—
GPIO14	TOUCH_CHANNEL12	—
GPIO15	TOUCH_CHANNEL13	—
GPIO16	ADC1_CHANNEL0	—
GPIO17	ADC1_CHANNEL1	—
GPIO18	ADC1_CHANNEL2	—
GPIO19	ADC1_CHANNEL3	—

名称	模拟功能 0	模拟功能 1
GPIO20	ADC1_CHANNEL4	—
GPIO21	ADC1_CHANNEL5	—
GPIO22	ADC1_CHANNEL6	—
GPIO23	ADC1_CHANNEL7	—
GPIO24	USB1P1_N0	—
GPIO25	USB1P1_P0	—
GPIO26	USB1P1_N1	—
GPIO27	USB1P1_P1	—
GPIO49	ADC2_CHANNEL0	—
GPIO50	ADC2_CHANNEL1	—
GPIO51	ADC2_CHANNEL2	ANA_COMP0
GPIO52	ADC2_CHANNEL3	ANA_COMP0
GPIO53	ADC2_CHANNEL4	ANA_COMP1
GPIO54	ADC2_CHANNEL5	ANA_COMP1

10.17 事件任务矩阵 (ETM) 功能

在 ESP32-P4 中，GPIO 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GPIO 的 ETM 任务，或者通过 GPIO 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 GPIO 相关的 ETM 任务和 ETM 事件。

GPIO ETM 有八个任务通道 *x* (0 ~ 7)。每个任务通道可接收的 ETM 任务有：

- GPIO_TASK_CH*x*_SET：触发时 GPIO 变为高电平。
- GPIO_TASK_CH*x*_CLEAR：触发时 GPIO 变为低电平。
- GPIO_TASK_CH*x*_TOGGLE：触发时 GPIO 电平翻转。

将任务通道 *x* 配置到 GPIO*y* 的示例如下：

- 配置 IO_MUX_GPIO*y*_MCU_SEL 的值为 1，使得 GPIO*y* 选择表 10.14-1 中的功能 1。
- 配置 GPIO_ENABLE_REG[*y*] 的值为 1。
- 配置 GPIO_EXT_ETM_TASK_GPIO*y*_SEL 为 *x*。
- 置位 GPIO_EXT_ETM_TASK_GPIO*y*_EN，使能 GPIO*y* 受到任务通道 *x* 的控制。

说明：

- 一个任务通道可被一个或多个 GPIO 选择。
- 如果 GPIO*y* 选择的任务通道 *x* 其信号（GPIO_TASK_CH*x*_SET、GPIO_TASK_CH*x*_CLEAR 和 GPIO_TASK_CH*x*_TOGGLE）中有两个或三个同时有效，则 GPIO_TASK_CH*x*_SET 优先级最高，GPIO_TASK_CH*x*_CLEAR 优先级次之，GPIO_TASK_CH*x*_TOGGLE 优先级最低。
- GPIO*y* 受到 ETM 任务通道控制时，GPIO_OUT_REG、GPIO_FUNC*n*_OUT_INV_SEL 和 GPIO_FUNC*n*_OUT_SEL 的值可能会被硬件修改。在 GPIO 退出 ETM 任务通道控制后，建议对上述寄存器重新进行配置。

GPIO 有八个事件通道，每个事件通道可以生成的 ETM 事件有：

- GPIO_EVT_CH*x*_RISE_EDGE：表示对应 GPIO 滤波器（见图 10.3-1）输出信号出现上升沿。
- GPIO_EVT_CH*x*_FALL_EDGE：表示对应 GPIO 滤波器（见图 10.3-1）输出信号出现下降沿。
- GPIO_EVT_CH*x*_ANY_EDGE：表示对应 GPIO 滤波器（见图 10.3-1）输出信号发生翻转。

事件通道具体配置如下：

- 置位 GPIO_EXT_ETM_CH*x*_EVENT_EN 使能事件通道 *x* (0 ~ 7)。
- 配置 GPIO_EXT_ETM_CH*x*_EVENT_SEL 为 *y* (0 ~ 54)，即选择 55 个 GPIO 中的一个 GPIO。

说明：
一个 GPIO 可被一个或多个事件通道选择。

GPIO 的 ETM 事件可以用来触发 GPIO 的 ETM 任务。例如，事件通道 0 选择 GPIO0，GPIO1 选择任务通道 0，并且 GPIO_EVT_CH*0*_RISE_EDGE 事件用于触发 GPIO_TASK_CH*0*_TOGGLE 任务。当通过 GPIO0 向芯片输入一个方波信号时，芯片通过 GPIO1 输出一个二分频的方波信号。

10.18 中断

ESP32-P4 的 HP IO MUX 和 HP GPIO 交换矩阵可以生成以下中断信号并发送给中断矩阵。

- GPIO_INTR0
- GPIO_INTR1
- GPIO_INTR2
- GPIO_INTR3
- GPIO_PAD_COMP_INTR

HP IO MUX 和 HP GPIO 交换矩阵有多个中断源可生成以上中断信号。其中，GPIO_PAD_COMP_INTR 中断的详细信息请参考章节 66 模拟电压比较器。表 10.18-1 列出了 HP IO MUX 和 HP GPIO 交换矩阵内部的其他中断源及其触发条件。

表 10.18-1. HP IO MUX 和 HP GPIO 交换矩阵内部中断源

内部中断源	触发条件	中断信号
GPIO_INTR0	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[0] (n: 0 ~ 31)	GPIO_INTR0
GPIO_INTR0	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[0] (n: 0 ~ 22)	GPIO_INTR0
GPIO_INTR1	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[1] (n: 0 ~ 31)	GPIO_INTR1
GPIO_INTR1	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[1] (n: 0 ~ 22)	GPIO_INTR1
GPIO_INTR2	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[3] (n: 0 ~ 31)	GPIO_INTR2
GPIO_INTR2	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[3] (n: 0 ~ 22)	GPIO_INTR2

GPIO_INTR3	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[4] (n: 0 ~ 31)	GPIO_INTR3
GPIO_INTR3	GPIO_STATUS1_INTERRUPT[*n*] & GPIO_PIN*n+32*_INT_ENA[4] (n: 0 ~ 22)	GPIO_INTR3

ESP32-P4 的 LP IO MUX 和 LP GPIO 交换矩阵可以生成以下中断信号并发送给中断矩阵。

- LP_GPIO_INTR

表 10.18-2 列出了 LP IO MUX 和 LP GPIO 交换矩阵内部的中断源及其触发条件。

表 10.18-2. LP IO MUX 和 LP GPIO 交换矩阵内部中断源

内部中断源	触发条件	中断信号
LP_GPIO_INTR	LP_GPIO_STATUS_DATA[*n*]（n: 0-15）	LP_GPIO_INTR

10.19 寄存器列表

10.19.1 HP GPIO 交换矩阵寄存器列表

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。请查看章节*寄存器的访问类型*，了解**访问**列缩写的含义。

名称	描述	访问	地址
配置寄存器			
GPIO_OUT_REG	GPIO0 ~ GPIO31 输出寄存器	0x0004	R/W/SC/WTC
GPIO_OUT_W1TS_REG	GPIO0 ~ GPIO31 输出置位寄存器	0x0008	WT
GPIO_OUT_W1TC_REG	GPIO0 ~ GPIO31 输出清除寄存器	0x000C	WT
GPIO_OUT1_REG	GPIO32 ~ GPIO56 输出寄存器	0x0010	R/W/SC/WTC
GPIO_OUT1_W1TS_REG	GPIO32 ~ GPIO56 输出置位寄存器	0x0014	WT
GPIO_OUT1_W1TC_REG	GPIO32 ~ GPIO56 输出清除寄存器	0x0018	WT
GPIO_ENABLE_REG	GPIO0 ~ GPIO31 输出使能寄存器	0x0020	R/W/WTC
GPIO_ENABLE_W1TS_REG	GPIO0 ~ GPIO31 输出使能置位寄存器	0x0024	WT
GPIO_ENABLE_W1TC_REG	GPIO0 ~ GPIO31 输出使能清除寄存器	0x0028	WT
GPIO_ENABLE1_REG	GPIO32 ~ GPIO56 输出使能寄存器	0x002C	R/W/WTC
GPIO_ENABLE1_W1TS_REG	GPIO32 ~ GPIO56 输出使能置位寄存器	0x0030	WT
GPIO_ENABLE1_W1TC_REG	GPIO32 ~ GPIO56 输出使能清除寄存器	0x0034	WT
GPIO_STRAP_REG	Strapping 管脚寄存器	0x0038	RO
GPIO_IN_REG	GPIO0 ~ GPIO31 输入寄存器	0x003C	RO
GPIO_IN1_REG	GPIO32 ~ GPIO56 输入寄存器	0x0040	RO
GPIO_STATUS_REG	GPIO0 ~ GPIO31 中断状态寄存器	0x0044	R/W/WTC
GPIO_STATUS_W1TS_REG	GPIO0 ~ GPIO31 中断状态置位寄存器	0x0048	WT
GPIO_STATUS_W1TC_REG	GPIO0 ~ GPIO31 中断状态清除寄存器	0x004C	WT
GPIO_STATUS1_REG	GPIO32 ~ GPIO56 中断状态寄存器	0x0050	R/W/WTC
GPIO_STATUS1_W1TS_REG	GPIO32 ~ GPIO56 中断状态置位寄存器	0x0054	WT

名称	描述	访问	地址
GPIO_STATUS1_W1TC_REG	GPIO32 ~ GPIO56 中断状态清除寄存器	0x0058	WT
GPIO_INTR_0_REG	GPIO0 ~ GPIO31 中断 0 的状态寄存器	0x005C	RO
GPIO_INTR1_0_REG	GPIO32 ~ GPIO56 中断 0 的状态寄存器	0x0060	RO
GPIO_INTR_1_REG	GPIO0 ~ GPIO31 中断 1 的状态寄存器	0x0064	RO
GPIO_INTR1_1_REG	GPIO32 ~ GPIO56 中断 1 的状态寄存器	0x0068	RO
GPIO_STATUS_NEXT_REG	GPIO0 ~ GPIO31 中断源寄存器	0x006C	RO
GPIO_STATUS_NEXT1_REG	GPIO32 ~ GPIO56 中断源寄存器	0x0070	RO
GPIO_PIN0_REG	GPIO0 管脚配置寄存器	0x0074	R/W
GPIO_PIN1_REG	GPIO1 管脚配置寄存器	0x0078	R/W
GPIO_PIN2_REG	GPIO2 管脚配置寄存器	0x007C	R/W
GPIO_PIN3_REG	GPIO3 管脚配置寄存器	0x0080	R/W
GPIO_PIN4_REG	GPIO4 管脚配置寄存器	0x0084	R/W
...	...	...	...
GPIO_PIN54_REG	GPIO54 管脚配置寄存器	0x014C	R/W
GPIO_PIN55_REG	GPIO55 管脚配置寄存器	0x0150	R/W
GPIO_PIN56_REG	GPIO56 管脚配置寄存器	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	输入信号 1 配置寄存器	0x015C	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	输入信号 2 配置寄存器	0x0160	R/W
GPIO_FUNC3_IN_SEL_CFG_REG	输入信号 3 配置寄存器	0x0164	R/W
...	...	...	...
GPIO_FUNC253_IN_SEL_CFG_REG	输入信号 253 配置寄存器	0x054C	R/W
GPIO_FUNC254_IN_SEL_CFG_REG	输入信号 254 配置寄存器	0x0550	R/W
GPIO_FUNC255_IN_SEL_CFG_REG	输入信号 255 配置寄存器	0x0554	R/W
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 输出配置寄存器	0x0558	varies
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 输出配置寄存器	0x055C	varies
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 输出配置寄存器	0x0560	varies
GPIO_FUNC3_OUT_SEL_CFG_REG	GPIO3 输出配置寄存器	0x0564	varies
...	...	...	...
GPIO_FUNC53_OUT_SEL_CFG_REG	GPIO53 输出配置寄存器	0x062C	varies
GPIO_FUNC54_OUT_SEL_CFG_REG	GPIO54 输出配置寄存器	0x0630	varies
GPIO_FUNC55_OUT_SEL_CFG_REG	GPIO55 输出配置寄存器	0x0634	varies
GPIO_FUNC56_OUT_SEL_CFG_REG	GPIO56 输出配置寄存器	0x0638	varies
GPIO_INTR_2_REG	GPIO0 ~ GPIO31 中断 2 的状态寄存器	0x063C	RO
GPIO_INTR1_2_REG	GPIO32 ~ GPIO56 中断 2 的状态寄存器	0x0640	RO
GPIO_INTR_3_REG	GPIO0 ~ GPIO31 中断 3 的状态寄存器	0x0644	RO
GPIO_INTR1_3_REG	GPIO32 ~ GPIO56 中断 3 的状态寄存器	0x0648	RO
GPIO_CLOCK_GATE_REG	GPIO 时钟门控寄存器	0x064C	R/W
GPIO_ZERO_DET0_FILTER_CNT_REG	模拟电压比较器的过零检测阈值配置寄存器	0x0710	R/W
GPIO_ZERO_DET1_FILTER_CNT_REG	模拟电压比较器的过零检测阈值配置寄存器	0x0714	R/W
GPIO 中断寄存器			
GPIO_INT_RAW_REG	模拟电压比较器的原始中断寄存器	0x0700	R/WTC/SS
GPIO_INT_ST_REG	模拟电压比较器的中断状态寄存器	0x0704	RO
GPIO_INT_ENA_REG	模拟电压比较器的中断使能寄存器	0x0708	R/W

名称	描述	访问	地址
GPIO_INT_CLR_REG	模拟电压比较器的中断清除寄存器	0x070C	WT
版本寄存器			
GPIO_DATE_REG	版本控制寄存器	0x07FC	R/W

10.19.2 HP IO MUX 寄存器列表

本小节的所有地址均为相对于 HP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
IO MUX 配置寄存器			
IO_MUX_GPIO0_REG	GPIO0 的 IO MUX 配置寄存器	0x0004	R/W
IO_MUX_GPIO1_REG	GPIO1 的 IO MUX 配置寄存器	0x0008	R/W
IO_MUX_GPIO2_REG	GPIO2 的 IO MUX 配置寄存器	0x000C	R/W
IO_MUX_GPIO3_REG	GPIO3 的 IO MUX 配置寄存器	0x0010	R/W
IO_MUX_GPIO4_REG	GPIO4 的 IO MUX 配置寄存器	0x0014	R/W
IO_MUX_GPIO5_REG	GPIO5 的 IO MUX 配置寄存器	0x0018	R/W
IO_MUX_GPIO6_REG	GPIO6 的 IO MUX 配置寄存器	0x001C	R/W
IO_MUX_GPIO7_REG	GPIO7 的 IO MUX 配置寄存器	0x0020	R/W
IO_MUX_GPIO8_REG	GPIO8 的 IO MUX 配置寄存器	0x0024	R/W
IO_MUX_GPIO9_REG	GPIO9 的 IO MUX 配置寄存器	0x0028	R/W
IO_MUX_GPIO10_REG	GPIO10 的 IO MUX 配置寄存器	0x002C	R/W
IO_MUX_GPIO11_REG	GPIO11 的 IO MUX 配置寄存器	0x0030	R/W
IO_MUX_GPIO12_REG	GPIO12 的 IO MUX 配置寄存器	0x0034	R/W
IO_MUX_GPIO13_REG	GPIO13 的 IO MUX 配置寄存器	0x0038	R/W
IO_MUX_GPIO14_REG	GPIO14 的 IO MUX 配置寄存器	0x003C	R/W
IO_MUX_GPIO15_REG	GPIO15 的 IO MUX 配置寄存器	0x0040	R/W
IO_MUX_GPIO16_REG	GPIO16 的 IO MUX 配置寄存器	0x0044	R/W
IO_MUX_GPIO17_REG	GPIO17 的 IO MUX 配置寄存器	0x0048	R/W
IO_MUX_GPIO18_REG	GPIO18 的 IO MUX 配置寄存器	0x004C	R/W
IO_MUX_GPIO19_REG	GPIO19 的 IO MUX 配置寄存器	0x0050	R/W
IO_MUX_GPIO20_REG	GPIO20 的 IO MUX 配置寄存器	0x0054	R/W
IO_MUX_GPIO21_REG	GPIO21 的 IO MUX 配置寄存器	0x0058	R/W
IO_MUX_GPIO22_REG	GPIO22 的 IO MUX 配置寄存器	0x005C	R/W
IO_MUX_GPIO23_REG	GPIO23 的 IO MUX 配置寄存器	0x0060	R/W
IO_MUX_GPIO24_REG	GPIO24 的 IO MUX 配置寄存器	0x0064	R/W
IO_MUX_GPIO25_REG	GPIO25 的 IO MUX 配置寄存器	0x0068	R/W
IO_MUX_GPIO26_REG	GPIO26 的 IO MUX 配置寄存器	0x006C	R/W
IO_MUX_GPIO27_REG	GPIO27 的 IO MUX 配置寄存器	0x0070	R/W
IO_MUX_GPIO28_REG	GPIO28 的 IO MUX 配置寄存器	0x0074	R/W
IO_MUX_GPIO29_REG	GPIO29 的 IO MUX 配置寄存器	0x0078	R/W
IO_MUX_GPIO30_REG	GPIO30 的 IO MUX 配置寄存器	0x007C	R/W

名称	描述	地址	访问
IO_MUX_GPIO31_REG	GPIO31 的 IO MUX 配置寄存器	0x0080	R/W
IO_MUX_GPIO32_REG	GPIO32 的 IO MUX 配置寄存器	0x0084	R/W
IO_MUX_GPIO33_REG	GPIO33 的 IO MUX 配置寄存器	0x0088	R/W
IO_MUX_GPIO34_REG	GPIO34 的 IO MUX 配置寄存器	0x008C	R/W
IO_MUX_GPIO35_REG	GPIO35 的 IO MUX 配置寄存器	0x0090	R/W
IO_MUX_GPIO36_REG	GPIO36 的 IO MUX 配置寄存器	0x0094	R/W
IO_MUX_GPIO37_REG	GPIO37 的 IO MUX 配置寄存器	0x0098	R/W
IO_MUX_GPIO38_REG	GPIO38 的 IO MUX 配置寄存器	0x009C	R/W
IO_MUX_GPIO39_REG	GPIO39 的 IO MUX 配置寄存器	0x00A0	R/W
IO_MUX_GPIO40_REG	GPIO40 的 IO MUX 配置寄存器	0x00A4	R/W
IO_MUX_GPIO41_REG	GPIO41 的 IO MUX 配置寄存器	0x00A8	R/W
IO_MUX_GPIO42_REG	GPIO42 的 IO MUX 配置寄存器	0x00AC	R/W
IO_MUX_GPIO43_REG	GPIO43 的 IO MUX 配置寄存器	0x00B0	R/W
IO_MUX_GPIO44_REG	GPIO44 的 IO MUX 配置寄存器	0x00B4	R/W
IO_MUX_GPIO45_REG	GPIO45 的 IO MUX 配置寄存器	0x00B8	R/W
IO_MUX_GPIO46_REG	GPIO46 的 IO MUX 配置寄存器	0x00BC	R/W
IO_MUX_GPIO47_REG	GPIO47 的 IO MUX 配置寄存器	0x00C0	R/W
IO_MUX_GPIO48_REG	GPIO48 的 IO MUX 配置寄存器	0x00C4	R/W
IO_MUX_GPIO49_REG	GPIO49 的 IO MUX 配置寄存器	0x00C8	R/W
IO_MUX_GPIO50_REG	GPIO50 的 IO MUX 配置寄存器	0x00CC	R/W
IO_MUX_GPIO51_REG	GPIO51 的 IO MUX 配置寄存器	0x00D0	R/W
IO_MUX_GPIO52_REG	GPIO52 的 IO MUX 配置寄存器	0x00D4	R/W
IO_MUX_GPIO53_REG	GPIO53 的 IO MUX 配置寄存器	0x00D8	R/W
IO_MUX_GPIO54_REG	GPIO54 的 IO MUX 配置寄存器	0x00DC	R/W
版本寄存器			
IO_MUX_DATE_REG	版本控制寄存器	0x0104	R/W

10.19.3 GPIO EXT 寄存器列表

GPIO EXT 寄存器包括 SDM 寄存器、毛刺滤波器寄存器和 ETM 寄存器。

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址 + 0x0F00 的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

请查看章节*寄存器的访问类型*，了解**访问**列缩写的含义。

名称	描述	地址	访问
SDM 配置寄存器			
GPIO_EXT_SIGMADELTA0_REG	SDM 通道 0 的占空比配置寄存器	0x0000	R/W
GPIO_EXT_SIGMADELTA1_REG	SDM 通道 1 的占空比配置寄存器	0x0004	R/W
GPIO_EXT_SIGMADELTA2_REG	SDM 通道 2 的占空比配置寄存器	0x0008	R/W
GPIO_EXT_SIGMADELTA3_REG	SDM 通道 3 的占空比配置寄存器	0x000C	R/W
GPIO_EXT_SIGMADELTA4_REG	SDM 通道 4 的占空比配置寄存器	0x0010	R/W
GPIO_EXT_SIGMADELTA5_REG	SDM 通道 5 的占空比配置寄存器	0x0014	R/W
GPIO_EXT_SIGMADELTA6_REG	SDM 通道 6 的占空比配置寄存器	0x0018	R/W

名称	描述	地址	访问
GPIO_EXT_SIGMADELTA7_REG	SDM 通道 7 的占空比配置寄存器	0x001C	R/W
GPIO_EXT_SIGMADELTA_MISC_REG	MISC 寄存器	0x0024	R/W
毛刺滤波器配置寄存器			
GPIO_EXT_GLITCH_FILTER_CH0_REG	毛刺滤波器通道 0 的配置寄存器	0x0030	R/W
GPIO_EXT_GLITCH_FILTER_CH1_REG	毛刺滤波器通道 1 的配置寄存器	0x0034	R/W
GPIO_EXT_GLITCH_FILTER_CH2_REG	毛刺滤波器通道 2 的配置寄存器	0x0038	R/W
GPIO_EXT_GLITCH_FILTER_CH3_REG	毛刺滤波器通道 3 的配置寄存器	0x003C	R/W
GPIO_EXT_GLITCH_FILTER_CH4_REG	毛刺滤波器通道 4 的配置寄存器	0x0040	R/W
GPIO_EXT_GLITCH_FILTER_CH5_REG	毛刺滤波器通道 5 的配置寄存器	0x0044	R/W
GPIO_EXT_GLITCH_FILTER_CH6_REG	毛刺滤波器通道 6 的配置寄存器	0x0048	R/W
GPIO_EXT_GLITCH_FILTER_CH7_REG	毛刺滤波器通道 7 的配置寄存器	0x004C	R/W
ETM 配置寄存器			
GPIO_EXT_ETM_EVENT_CH0_CFG_REG	ETM 通道 0 的配置寄存器	0x0060	R/W
GPIO_EXT_ETM_EVENT_CH1_CFG_REG	ETM 通道 1 的配置寄存器	0x0064	R/W
GPIO_EXT_ETM_EVENT_CH2_CFG_REG	ETM 通道 2 的配置寄存器	0x0068	R/W
GPIO_EXT_ETM_EVENT_CH3_CFG_REG	ETM 通道 3 的配置寄存器	0x006C	R/W
GPIO_EXT_ETM_EVENT_CH4_CFG_REG	ETM 通道 4 的配置寄存器	0x0070	R/W
GPIO_EXT_ETM_EVENT_CH5_CFG_REG	ETM 通道 5 的配置寄存器	0x0074	R/W
GPIO_EXT_ETM_EVENT_CH6_CFG_REG	ETM 通道 6 的配置寄存器	0x0078	R/W
GPIO_EXT_ETM_EVENT_CH7_CFG_REG	ETM 通道 7 的配置寄存器	0x007C	R/W
GPIO_EXT_ETM_TASK_P0_CFG_REG	ETM 的 GPIO 选择配置寄存器 0	0x00A0	R/W
GPIO_EXT_ETM_TASK_P1_CFG_REG	ETM 的 GPIO 选择配置寄存器 1	0x00A4	R/W
GPIO_EXT_ETM_TASK_P2_CFG_REG	ETM 的 GPIO 选择配置寄存器 2	0x00A8	R/W
GPIO_EXT_ETM_TASK_P3_CFG_REG	ETM 的 GPIO 选择配置寄存器 3	0x00AC	R/W
GPIO_EXT_ETM_TASK_P4_CFG_REG	ETM 的 GPIO 选择配置寄存器 4	0x00B0	R/W
GPIO_EXT_ETM_TASK_P5_CFG_REG	ETM 的 GPIO 选择配置寄存器 5	0x00B4	R/W
GPIO_EXT_ETM_TASK_P6_CFG_REG	ETM 的 GPIO 选择配置寄存器 6	0x00B8	R/W
GPIO_EXT_ETM_TASK_P7_CFG_REG	ETM 的 GPIO 选择配置寄存器 7	0x00BC	R/W
GPIO_EXT_ETM_TASK_P8_CFG_REG	ETM 的 GPIO 选择配置寄存器 8	0x00C0	R/W
GPIO_EXT_ETM_TASK_P9_CFG_REG	ETM 的 GPIO 选择配置寄存器 9	0x00C4	R/W
GPIO_EXT_ETM_TASK_P10_CFG_REG	ETM 的 GPIO 选择配置寄存器 10	0x00C8	R/W
GPIO_EXT_ETM_TASK_P11_CFG_REG	ETM 的 GPIO 选择配置寄存器 11	0x00CC	R/W
GPIO_EXT_ETM_TASK_P12_CFG_REG	ETM 的 GPIO 选择配置寄存器 12	0x00D0	R/W
GPIO_EXT_ETM_TASK_P13_CFG_REG	ETM 的 GPIO 选择配置寄存器 13	0x00D4	R/W
版本寄存器			
GPIO_EXT_VERSION_REG	版本控制寄存器	0x00FC	R/W

10.19.4 LP GPIO 交换矩阵寄存器列表

本小节的所有地址均为相对于 LP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。请查看章节*寄存器的访问类型*，了解**访问**列缩写的含义。

名称	描述	地址	访问
时钟使能寄存器			
LP_GPIO_CLK_EN_REG	LP GPIO 时钟门控寄存器	0x0000	R/W
配置寄存器			
LP_GPIO_OUT_REG	LP GPIO 输出寄存器	0x0008	R/W/WTC
LP_GPIO_OUT_W1TS_REG	LP GPIO 输出置位寄存器	0x000C	WT
LP_GPIO_OUT_W1TC_REG	LP GPIO 输出清除寄存器	0x0010	WT
LP_GPIO_ENABLE_REG	LP GPIO 输出使能寄存器	0x0014	R/W/WTC
LP_GPIO_ENABLE_W1TS_REG	LP GPIO 输出使能置位寄存器	0x0018	WT
LP_GPIO_ENABLE_W1TC_REG	LP GPIO 输出使能清除寄存器	0x001C	WT
LP_GPIO_STATUS_REG	LP GPIO 中断状态寄存器	0x0020	R/W/WTC
LP_GPIO_STATUS_W1TS_REG	LP GPIO 中断状态置位寄存器	0x0024	WT
LP_GPIO_STATUS_W1TC_REG	LP GPIO 中断状态清除寄存器	0x0028	WT
LP_GPIO_STATUS_NEXT_REG	LP GPIO 中断源寄存器	0x002C	RO
LP_GPIO_IN_REG	LP GPIO 输入寄存器	0x0030	RO
LP_GPIO_PIN0_REG	LP GPIO0 配置寄存器	0x0034	varies
LP_GPIO_PIN1_REG	LP GPIO1 配置寄存器	0x0038	varies
LP_GPIO_PIN2_REG	LP GPIO2 配置寄存器	0x003C	varies
LP_GPIO_PIN3_REG	LP GPIO3 配置寄存器	0x0040	varies
LP_GPIO_PIN4_REG	LP GPIO4 配置寄存器	0x0044	varies
LP_GPIO_PIN5_REG	LP GPIO5 配置寄存器	0x0048	varies
LP_GPIO_PIN6_REG	LP GPIO6 配置寄存器	0x004C	varies
LP_GPIO_PIN7_REG	LP GPIO7 配置寄存器	0x0050	varies
LP_GPIO_PIN8_REG	LP GPIO8 配置寄存器	0x0054	varies
LP_GPIO_PIN9_REG	LP GPIO9 配置寄存器	0x0058	varies
LP_GPIO_PIN10_REG	LP GPIO10 配置寄存器	0x005C	varies
LP_GPIO_PIN11_REG	LP GPIO11 配置寄存器	0x0060	varies
LP_GPIO_PIN12_REG	LP GPIO12 配置寄存器	0x0064	varies
LP_GPIO_PIN13_REG	LP GPIO13 配置寄存器	0x0068	varies
LP_GPIO_PIN14_REG	LP GPIO14 配置寄存器	0x006C	varies
LP_GPIO_PIN15_REG	LP GPIO15 配置寄存器	0x0070	varies
LP_GPIO_FUNC0_IN_SEL_CFG_REG	输入信号 0 的配置寄存器	0x0074	R/W
LP_GPIO_FUNC1_IN_SEL_CFG_REG	输入信号 1 的配置寄存器	0x0078	R/W
LP_GPIO_FUNC2_IN_SEL_CFG_REG	输入信号 2 的配置寄存器	0x007C	R/W
LP_GPIO_FUNC3_IN_SEL_CFG_REG	输入信号 3 的配置寄存器	0x0080	R/W
LP_GPIO_FUNC4_IN_SEL_CFG_REG	输入信号 4 的配置寄存器	0x0084	R/W
LP_GPIO_FUNC5_IN_SEL_CFG_REG	输入信号 5 的配置寄存器	0x0088	R/W
LP_GPIO_FUNC6_IN_SEL_CFG_REG	输入信号 6 的配置寄存器	0x008C	R/W
LP_GPIO_FUNC7_IN_SEL_CFG_REG	输入信号 7 的配置寄存器	0x0090	R/W
LP_GPIO_FUNC8_IN_SEL_CFG_REG	输入信号 8 的配置寄存器	0x0094	R/W
LP_GPIO_FUNC9_IN_SEL_CFG_REG	输入信号 9 的配置寄存器	0x0098	R/W
LP_GPIO_FUNC10_IN_SEL_CFG_REG	输入信号 10 的配置寄存器	0x009C	R/W
LP_GPIO_FUNC11_IN_SEL_CFG_REG	输入信号 11 的配置寄存器	0x00A0	R/W
LP_GPIO_FUNC12_IN_SEL_CFG_REG	输入信号 12 的配置寄存器	0x00A4	R/W

名称	描述	地址	访问
LP_GPIO_FUNC13_IN_SEL_CFG_REG	输入信号 13 的配置寄存器	0x00A8	R/W
LP_GPIO_FUNC0_OUT_SEL_CFG_REG	LP GPIO0 的输出配置寄存器	0x00F4	R/W
LP_GPIO_FUNC1_OUT_SEL_CFG_REG	LP GPIO1 的输出配置寄存器	0x00F8	R/W
LP_GPIO_FUNC2_OUT_SEL_CFG_REG	LP GPIO2 的输出配置寄存器	0x00FC	R/W
LP_GPIO_FUNC3_OUT_SEL_CFG_REG	LP GPIO3 的输出配置寄存器	0x0100	R/W
LP_GPIO_FUNC4_OUT_SEL_CFG_REG	LP GPIO4 的输出配置寄存器	0x0104	R/W
LP_GPIO_FUNC5_OUT_SEL_CFG_REG	LP GPIO5 的输出配置寄存器	0x0108	R/W
LP_GPIO_FUNC6_OUT_SEL_CFG_REG	LP GPIO6 的输出配置寄存器	0x010C	R/W
LP_GPIO_FUNC7_OUT_SEL_CFG_REG	LP GPIO7 的输出配置寄存器	0x0110	R/W
LP_GPIO_FUNC8_OUT_SEL_CFG_REG	LP GPIO8 的输出配置寄存器	0x0114	R/W
LP_GPIO_FUNC9_OUT_SEL_CFG_REG	LP GPIO9 的输出配置寄存器	0x0118	R/W
LP_GPIO_FUNC10_OUT_SEL_CFG_REG	LP GPIO10 的输出配置寄存器	0x011C	R/W
LP_GPIO_FUNC11_OUT_SEL_CFG_REG	LP GPIO11 的输出配置寄存器	0x0120	R/W
LP_GPIO_FUNC12_OUT_SEL_CFG_REG	LP GPIO12 的输出配置寄存器	0x0124	R/W
LP_GPIO_FUNC13_OUT_SEL_CFG_REG	LP GPIO13 的输出配置寄存器	0x0128	R/W
LP_GPIO_FUNC14_OUT_SEL_CFG_REG	LP GPIO14 的输出配置寄存器	0x012C	R/W
LP_GPIO_FUNC15_OUT_SEL_CFG_REG	LP GPIO15 的输出配置寄存器	0x0130	R/W
版本寄存器			
LP_GPIO_VER_DATE_REG	版本控制寄存器	0x0004	R/W

10.19.5 LP IO MUX 寄存器

本小节的所有地址均为相对于 LP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
时钟使能寄存器			
LP_IOMUX_CLK_EN_REG	LP IO MUX 时钟门控寄存器	0x0000	R/W
配置寄存器			
LP_IOMUX_PAD0_REG	GPIO0 的 LP IO MUX 配置寄存器	0x0008	R/W
LP_IOMUX_PAD1_REG	GPIO1 的 LP IO MUX 配置寄存器	0x000C	R/W
LP_IOMUX_PAD2_REG	GPIO2 的 LP IO MUX 配置寄存器	0x0010	R/W
LP_IOMUX_PAD3_REG	GPIO3 的 LP IO MUX 配置寄存器	0x0014	R/W
LP_IOMUX_PAD4_REG	GPIO4 的 LP IO MUX 配置寄存器	0x0018	R/W
LP_IOMUX_PAD5_REG	GPIO5 的 LP IO MUX 配置寄存器	0x001C	R/W
LP_IOMUX_PAD6_REG	GPIO6 的 LP IO MUX 配置寄存器	0x0020	R/W
LP_IOMUX_PAD7_REG	GPIO7 的 LP IO MUX 配置寄存器	0x0024	R/W
LP_IOMUX_PAD8_REG	GPIO8 的 LP IO MUX 配置寄存器	0x0028	R/W
LP_IOMUX_PAD9_REG	GPIO9 的 LP IO MUX 配置寄存器	0x002C	R/W
LP_IOMUX_PAD10_REG	GPIO10 的 LP IO MUX 配置寄存器	0x0030	R/W
LP_IOMUX_PAD11_REG	GPIO11 的 LP IO MUX 配置寄存器	0x0034	R/W
LP_IOMUX_PAD12_REG	GPIO12 的 LP IO MUX 配置寄存器	0x0038	R/W

名称	描述	地址	访问
LP_IOMUX_PAD13_REG	GPIO13 的 LP IO MUX 配置寄存器	0x003C	R/W
LP_IOMUX_PAD14_REG	GPIO14 的 LP IO MUX 配置寄存器	0x0040	R/W
LP_IOMUX_PAD15_REG	GPIO15 的 LP IO MUX 配置寄存器	0x0044	R/W
LP_IOMUX_LP_PAD_HOLD_REG	LP GPIO Hold 控制寄存器	0x004C	R/W
LP_IOMUX_LP_PAD_HYS_REG	LP GPIO 迟滞控制寄存器	0x0050	R/W
版本寄存器			
LP_IOMUX_VER_DATE_REG	版本控制寄存器	0x0004	R/W

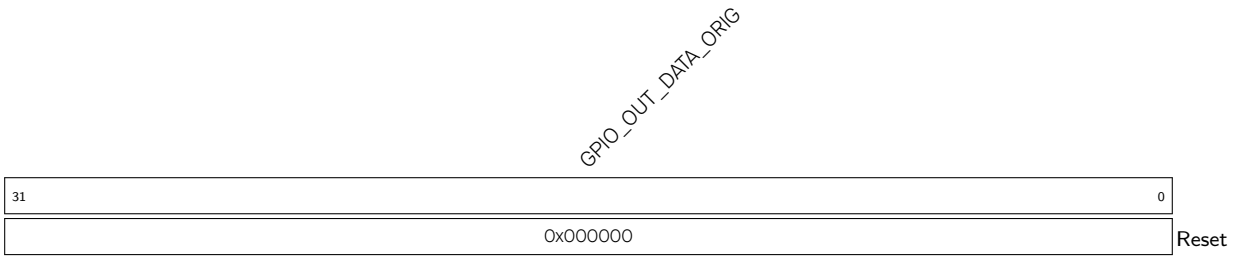
10.20 寄存器

10.20.1 HP GPIO 交换矩阵寄存器

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 10.1. GPIO_OUT_REG (0x0004)



GPIO_OUT_DATA_ORIG 配置简单 GPIO 输出模式下 GPIO0 ~ GPIO31 的输出值。

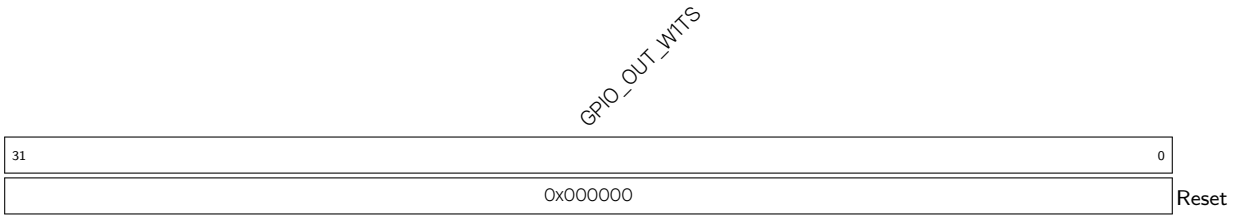
0：低电平

1：高电平

bit0 ~ bit31 的值分别对应 GPIO0 ~ 31 的输出值。

(R/W/SC/WTC)

Register 10.2. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS 配置是否置位 GPIO0 ~ GPIO31 的输出寄存器 GPIO_OUT_REG。

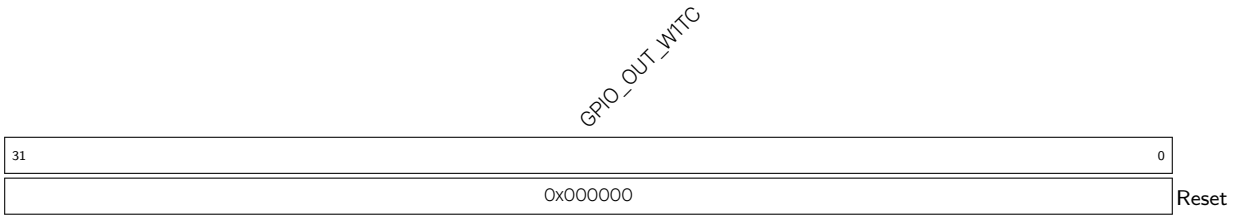
0: 不置位

1: GPIO_OUT_REG 中相应位也将置 1

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。推荐使用此寄存器来置位 GPIO_OUT_REG。

(WT)

Register 10.3. GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC 配置是否清零 GPIO0 ~ GPIO31 输出寄存器 GPIO_OUT_REG。

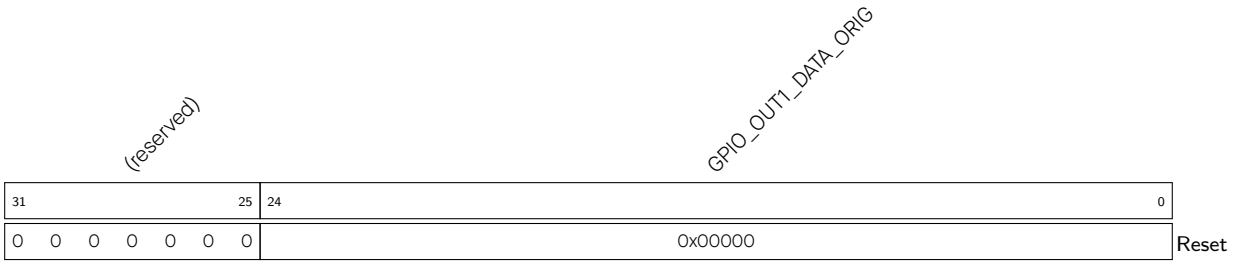
0：不清零

1: GPIO_OUT_REG 中相应位将被清零

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。推荐使用此寄存器来清零 GPIO_OUT_REG。

(WT)

Register 10.4. GPIO_OUT1_REG (0x0010)



GPIO_OUT1_DATA_ORIG 配置简单输出模式下 GPIO32 ~ GPIO54 的输出值。

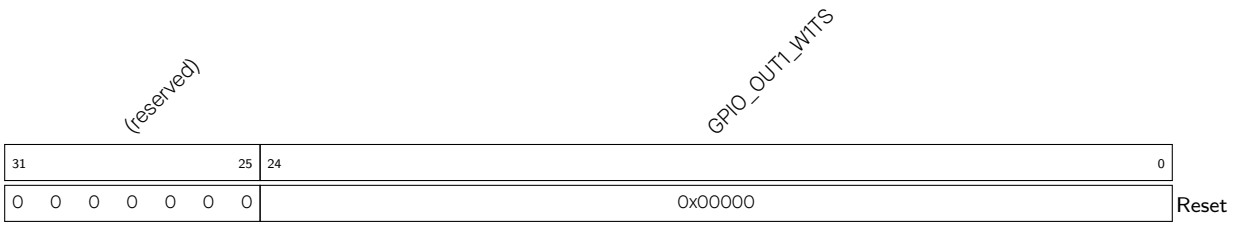
0：低电平

1：高电平

bit0 ~ bit22 的值分别对应 GPIO32 ~ GPIO54 的输出值。

(R/W/SC/WTC)

Register 10.5. GPIO_OUT1_W1TS_REG (0x0014)



GPIO_OUT1_W1TS 配置是否置位 GPIO32 ~ GPIO54 的输出寄存器 GPIO_OUT1_REG。

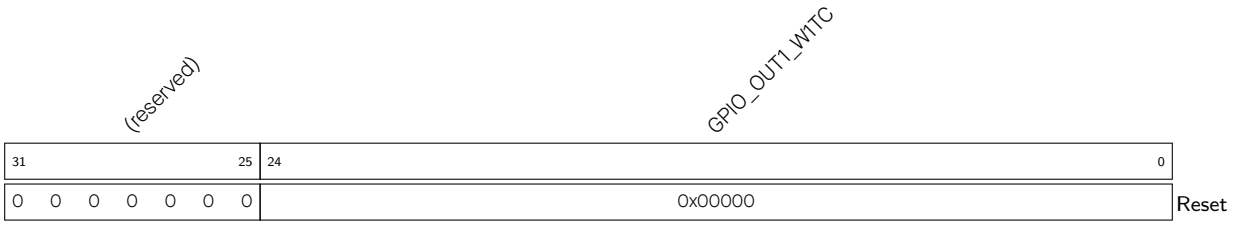
0：不置位

1：GPIO_OUT1_REG 中相应位也将置 1

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。推荐使用此寄存器来置位 GPIO_OUT1_REG。

(WT)

Register 10.6. GPIO_OUT1_W1TC_REG (0x0018)



GPIO_OUT1_W1TC 配置是否清零 GPIO32 ~ GPIO54 输出寄存器 GPIO_OUT1_REG。

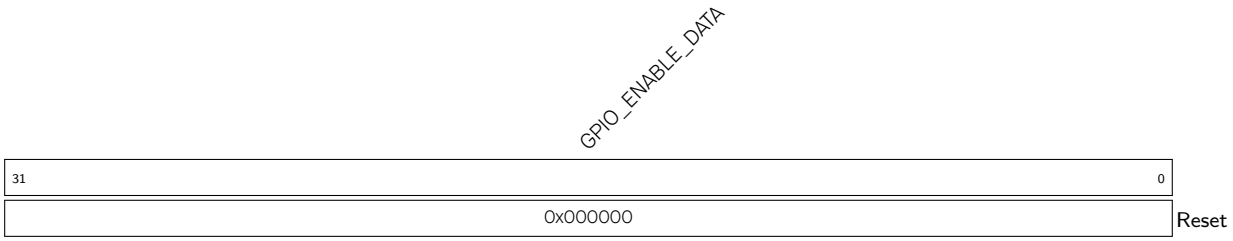
0：不清零

1：GPIO_OUT1_REG 中相应位将被清零

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。推荐使用此寄存器来清零 GPIO_OUT_REG。

(WT)

Register 10.7. GPIO_ENABLE_REG (0x0020)



GPIO_ENABLE_DATA 配置是否使能 GPIO0 ~ GPIO31 输出。

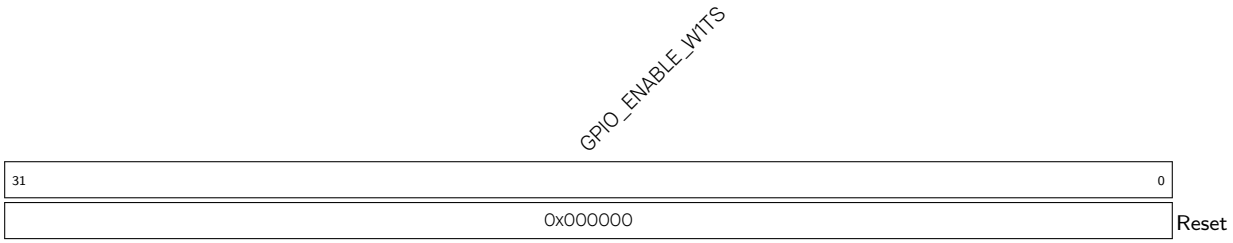
0：不使能

1：使能

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。

(R/W/WTC)

Register 10.8. GPIO_ENABLE_W1TS_REG (0x0024)



GPIO_ENABLE_W1TS 配置是否置位 GPIO0 ~ GPIO31 的输出使能寄存器 GPIO_ENABLE_REG。

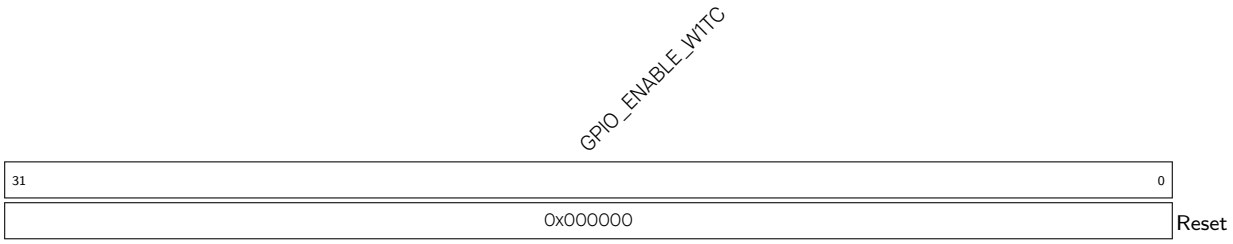
0：不置位

1：GPIO_ENABLE_REG 中相应位也将置 1

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。推荐使用该寄存器来置位 GPIO_ENABLE_REG。

(WT)

Register 10.9. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC 配置是否清零 GPIO0 ~ GPIO31 的输出使能寄存器 GPIO_ENABLE_REG。

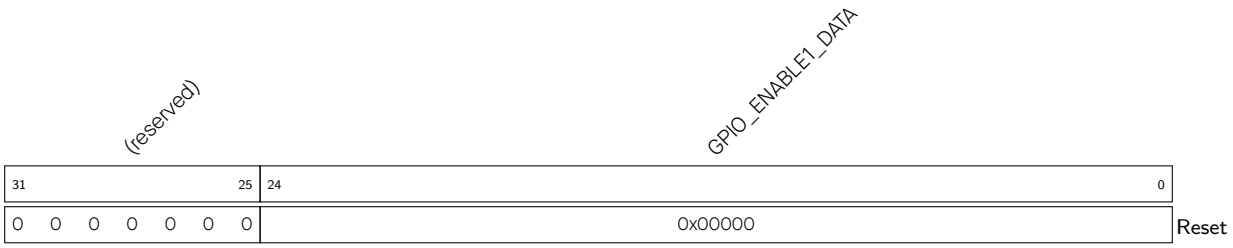
0：不清零

1：GPIO_ENABLE_REG 中相应位将被清零

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。推荐使用该寄存器清零 GPIO_ENABLE_REG。

(WT)

Register 10.10. GPIO_ENABLE1_REG (0x002C)



GPIO_ENABLE1_DATA 配置是否使能 GPIO32 ~ GPIO54 输出。

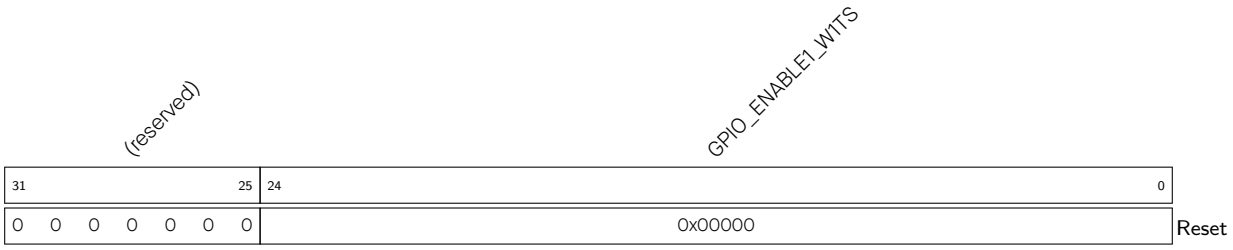
0：不使能

1：使能

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。

(R/W/WTC)

Register 10.11. GPIO_ENABLE1_W1TS_REG (0x0030)



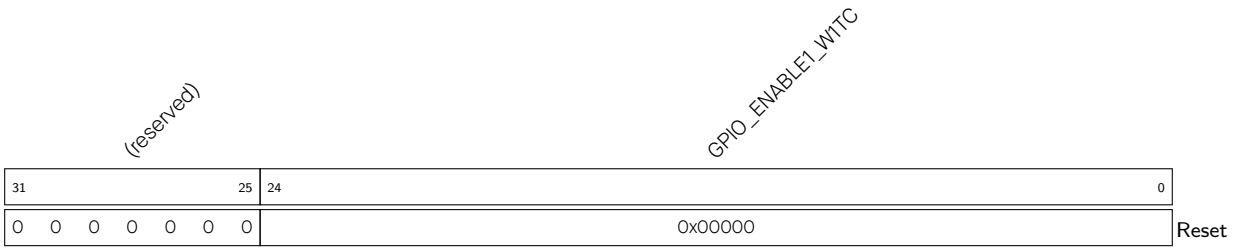
GPIO_ENABLE1_W1TS 配置是否置位 GPIO32 ~ GPIO54 的输出使能寄存器 GPIO_ENABLE1_REG。

0：不置位

1：GPIO_ENABLE1_REG 中相应位也将置 1

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。推荐使用该寄存器来置位 GPIO_ENABLE1_REG。(WT)

Register 10.12. GPIO_ENABLE1_W1TC_REG (0x0034)



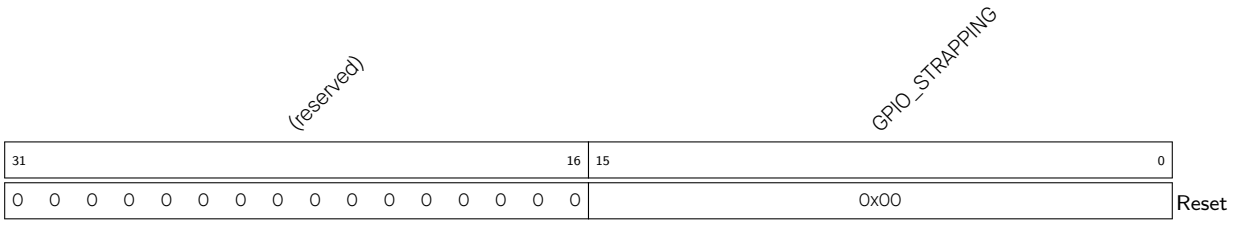
GPIO_ENABLE1_W1TC 配置是否清零 GPIO32 ~ GPIO54 的输出使能寄存器 GPIO_ENABLE1_REG。

0：不清零

1：GPIO_ENABLE1_REG 中相应位将被清零

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。推荐使用该寄存器清零 GPIO_ENABLE1_REG。(WT)

Register 10.13. GPIO_STRAP_REG (0x0038)



GPIO_STRAPPING 表示 GPIO Strapping 管脚的值。

bit0：GPIO38

bit1：GPIO37

bit2：GPIO36

bit3：GPIO35

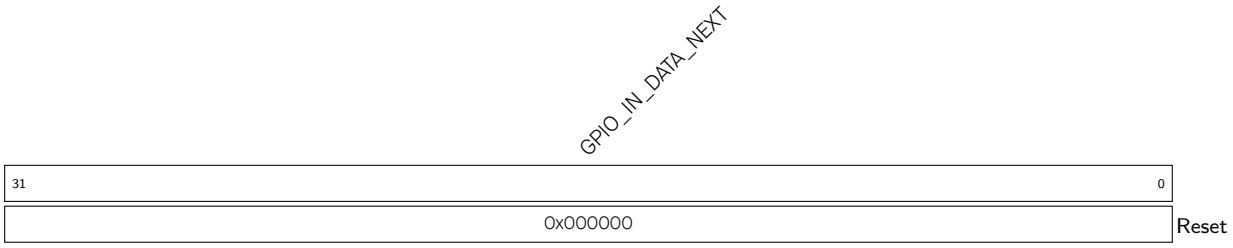
bit4：GPIO34

bit5 ~ bit15：无效

更多有关 Strapping 管脚控制的功能，见《*ESP32-P4 技术规格书*》 > 章节 *Strapping 管脚*。

(RO)

Register 10.14. GPIO_IN_REG (0x003C)



GPIO_IN_DATA_NEXT 表示 GPIO0 ~ GPIO31 的输入值。

每一位均代表一个管脚的输入值：

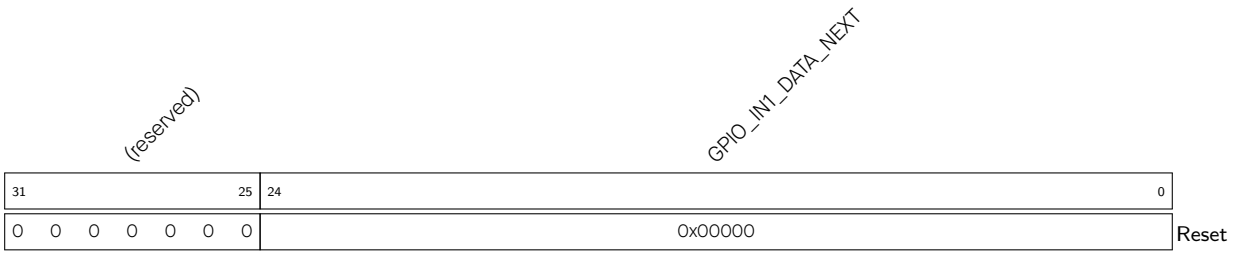
0：低电平

1：高电平

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。

(RO)

Register 10.15. GPIO_IN1_REG (0x0040)



GPIO_IN1_DATA_NEXT 表示 GPIO32 ~ GPIO54 的输入值。每一位均代表一个管脚的输入值：

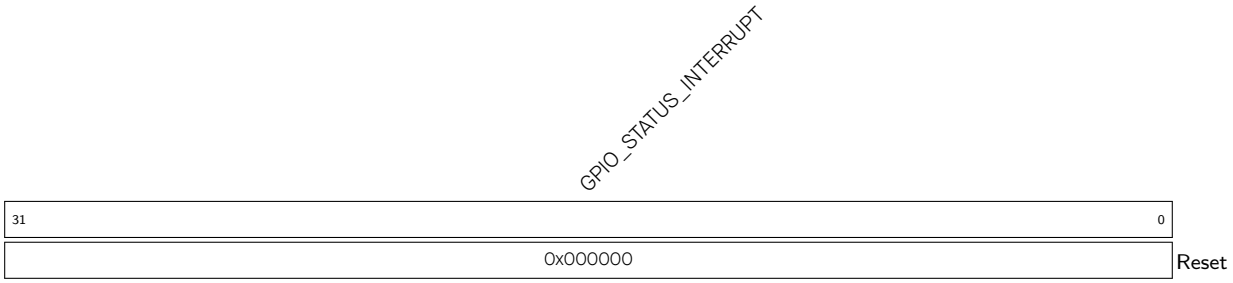
0：低电平

1：高电平

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。

(RO)

Register 10.16. GPIO_STATUS_REG (0x0044)



GPIO_STATUS_INTERRUPT GPIO0 ~ GPIO31 的中断状态寄存器，并且软件可配置该寄存器的值。

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。

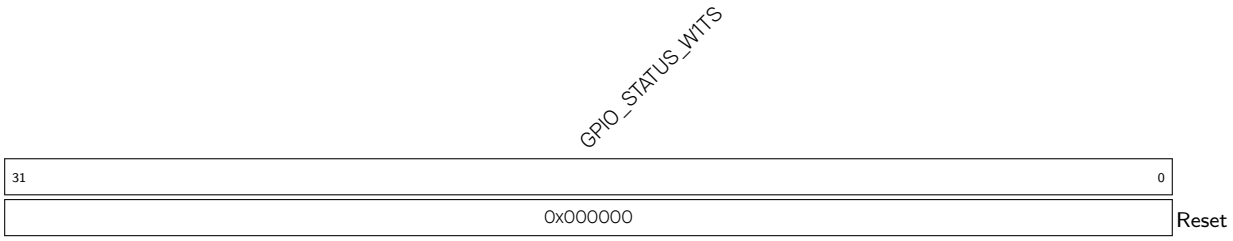
每一位均代表对应 GPIO 的状态：

0：表示对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所选择的中断，或软件配置该位寄存器的值为 0。

1：表示对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所选择的中断，或软件配置该位寄存器的值为 1。

(R/W/WTC)

Register 10.17. GPIO_STATUS_W1TS_REG (0x0048)



GPIO_STATUS_W1TS 配置是否置位 GPIO0 ~ GPIO31 的中断状态寄存器 GPIO_STATUS_INTERRUPT。

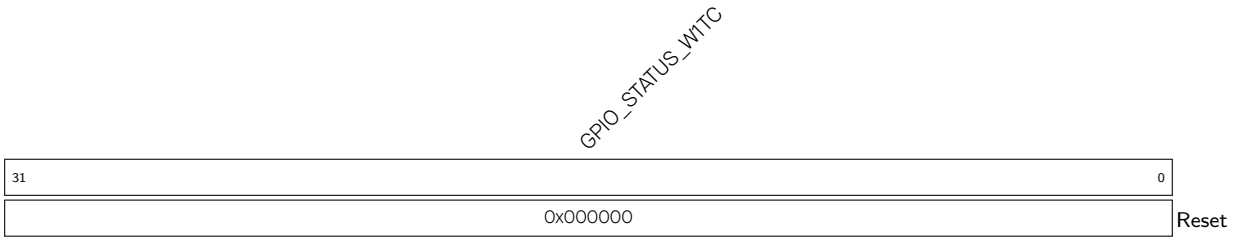
bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。

每一位置 1，则 GPIO_STATUS_INTERRUPT 中的相应位也置 1。

推荐使用此寄存器来置位 GPIO_STATUS_INTERRUPT。

(WT)

Register 10.18. GPIO_STATUS_W1TC_REG (0x004C)



GPIO_STATUS_W1TC 配置是否清零 GPIO0 ~ GPIO31 的中断状态寄存器 GPIO_STATUS_INTERRUPT。

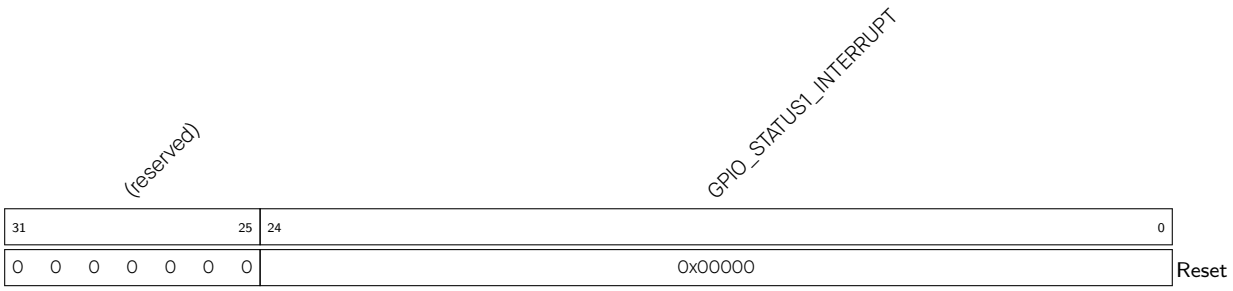
bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。

每一位置 1，则 GPIO_STATUS_INTERRUPT 中的相应位也清零。

推荐使用此寄存器来清零 GPIO_STATUS_INTERRUPT。

(WT)

Register 10.19. GPIO_STATUS1_REG (0x0050)



GPIO_STATUS1_INTERRUPT 表示 GPIO32 ~ GPIO54 的中断状态寄存器，并且软件可配置该寄存器的值。

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。

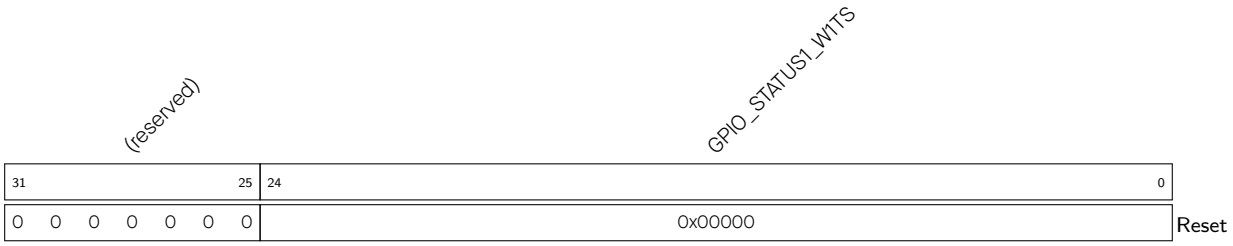
每一位均代表对应 GPIO 的状态：

0：表示对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所选择的中断，或软件配置该位寄存器的值为 0。

1：表示对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所选择的中断，或软件配置该位寄存器的值为 1。

(R/W/WTC)

Register 10.20. GPIO_STATUS1_W1TS_REG (0x0054)



GPIO_STATUS1_W1TS 配置是否置位 GPIO32 ~ GPIO54 的中断状态寄存器 GPIO_STATUS1_INTERRUPT。

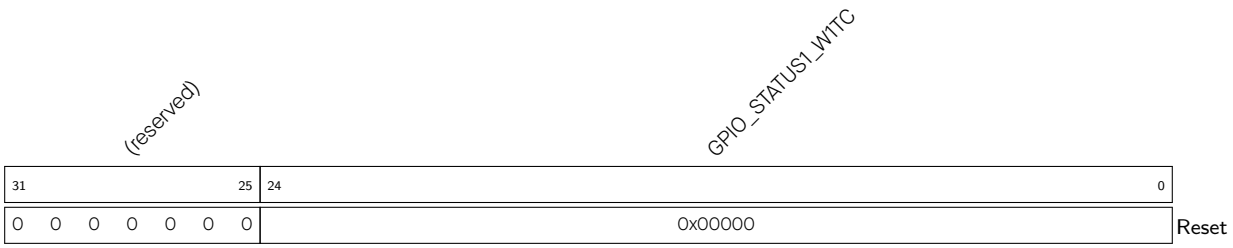
bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。

每一位置 1，则 GPIO_STATUS1_INTERRUPT 中的相应位也置 1。

推荐使用此寄存器来置位 GPIO_STATUS1_INTERRUPT。

(WT)

Register 10.21. GPIO_STATUS1_W1TC_REG (0x0058)



GPIO_STATUS1_W1TC 配置是否清零 GPIO32 ~ GPIO54 的中断状态寄存器 GPIO_STATUS1_INTERRUPT。

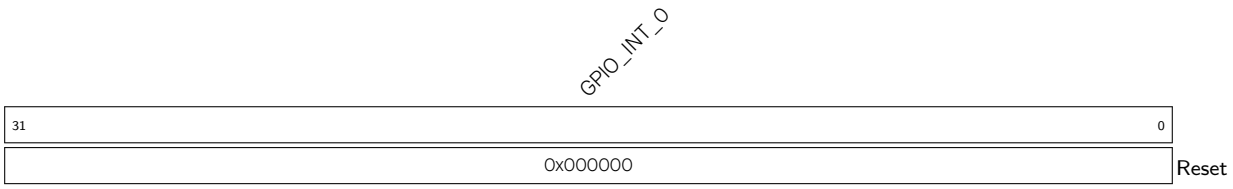
bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。

每一位置 1，则 GPIO_STATUS1_INTERRUPT 中的相应位将被清零。

推荐使用此寄存器来清零 GPIO_STATUS1_INTERRUPT。

(WT)

Register 10.22. GPIO_INTR_0_REG (0x005C)



GPIO_INT_0 表示 GPIO0 ~ GPIO31 的 CPU 中断 0 的屏蔽中断状态位。每位代表：

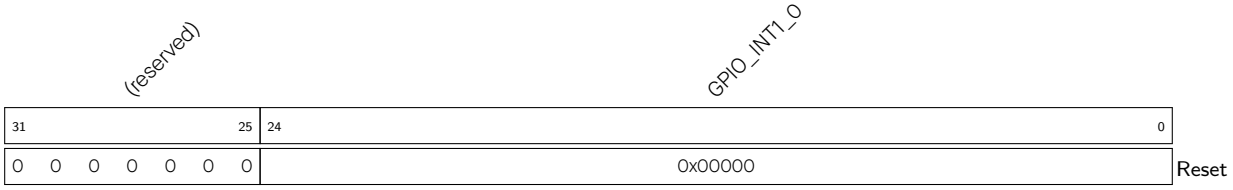
0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。如果 GPIO_PIN*n*_REG 中 bit13 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS_REG 中相应 bit 的中断状态一致。

(RO)

Register 10.23. GPIO_INTR1_0_REG (0x0060)



GPIO_INT1_0 表示 GPIO32 ~ GPIO54 的 CPU 中断 0 的屏蔽中断状态位。每位代表：

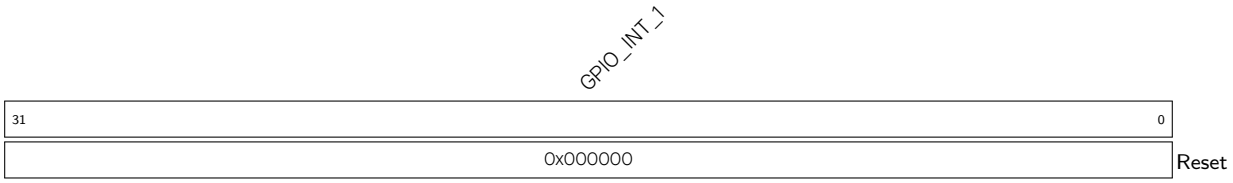
0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。如果 GPIO_PIN*n*_REG 中 bit13 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS1_REG 中相应 bit 的中断状态一致。

(RO)

Register 10.24. GPIO_INTR_1_REG (0x0064)



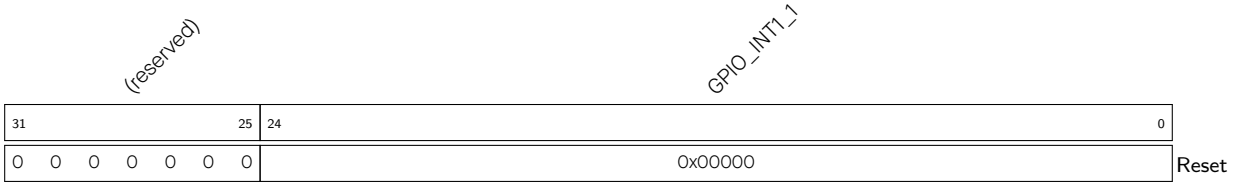
GPIO_INT_1 表示 GPIO0 ~ GPIO31 的 CPU 中断 1 的屏蔽中断状态位。每位代表：

0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。如果 GPIO_PIN*n*_REG 中 bit14 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS_REG 中相应 bit 的中断状态一致。(RO)

Register 10.25. GPIO_INTR1_1_REG (0x0068)



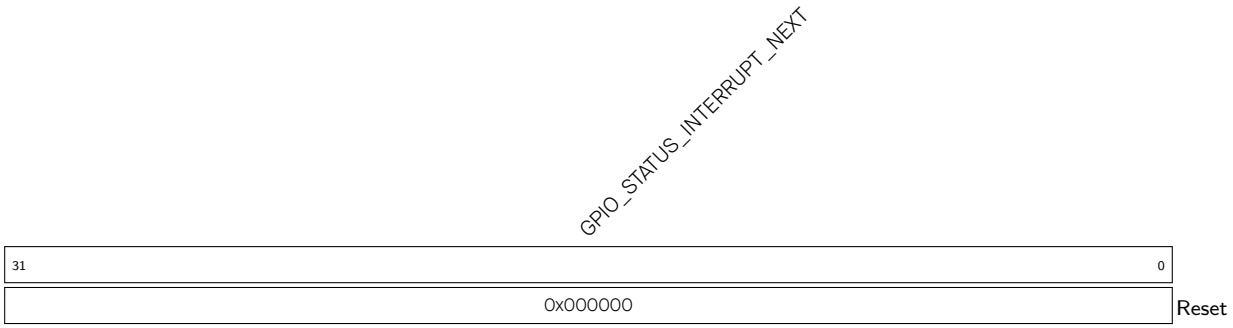
GPIO_INT1_1 表示 GPIO32 ~ GPIO54 的 CPU 中断 1 的屏蔽中断状态位。每位代表：

0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。如果 GPIO_PIN*n*_REG 中 bit14 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS1_REG 中相应 bit 的中断状态一致。(RO)

Register 10.26. GPIO_STATUS_NEXT_REG (0x006C)



GPIO_STATUS_INTERRUPT_NEXT 表示 GPIO0 ~ GPIO31 的中断源信号。

bit0 ~ bit31 分别对应 GPIO0 ~ 31。每位代表：

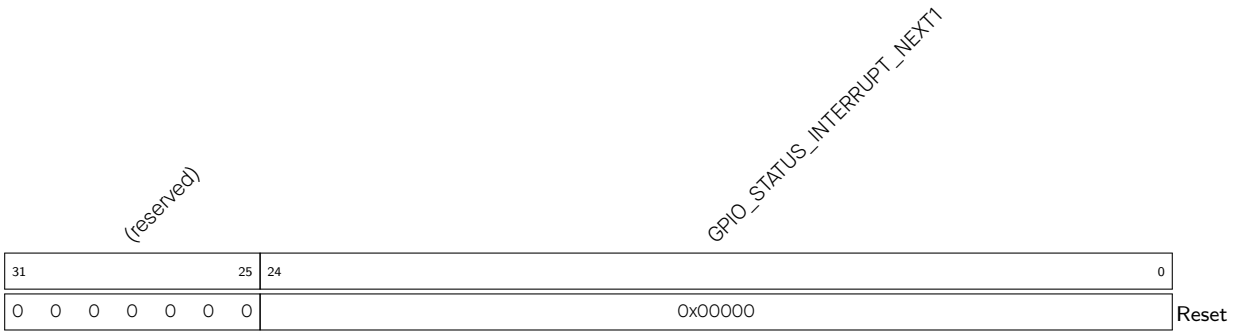
0：表示 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。

(RO)

Register 10.27. GPIO_STATUS_NEXT1_REG (0x0070)



GPIO_STATUS_INTERRUPT_NEXT1 表示 GPIO32 ~ GPIO54 的中断源信号。

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。每位代表：

0：表示 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。

(RO)

Register 10.28. GPIO_PIN*n*_REG (*n*: 0 - 54) (0x0074+0x4**n*)



GPIO_PIN*n*_SYNC2_BYPASS 配置是否使能 GPIO 输入数据在第二级同步中 HP IO MUX 运行时钟上升沿同步或下降沿同步。

0：不同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

GPIO_PIN*n*_PAD_DRIVER 配置选择管脚的输出驱动模式。

0：正常输出

1：开漏输出

(R/W)

GPIO_PIN*n*_SYNC1_BYPASS 配置是否使能 GPIO 输入数据在第一级同步中 HP IO MUX 运行时钟上升沿同步或下降沿同步。

0：不同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

GPIO_PIN*n*_INT_TYPE 配置 GPIO 中断类型。

0：关闭 GPIO 中断

1：上升沿触发

2：下降沿触发

3：任一沿触发

4：低电平触发

5：高电平触发

(R/W)

GPIO_PIN*n*_WAKEUP_ENABLE 配置是否使能 GPIO 唤醒功能。

0：不使能

1：使能

该功能只能将 CPU 从 Light-sleep 中唤醒。

(R/W)

见下页

Register 10.28. GPIO_PIN*n*_REG (*n*: 0 - 54) (0x0074+0x4**n*)

接上页

GPIO_PIN*n*_INT_ENA 配置是否使能 CPU 中断或 CPU 非屏蔽中断。

bit13：配置是否使能 CPU 中断 0：

0：不使能

1：使能

bit14：配置是否使能 CPU 中断 1：

0：不使能

1：使能

bit15：无效

bit16：配置是否使能 CPU 中断 2：

0：不使能

1：使能

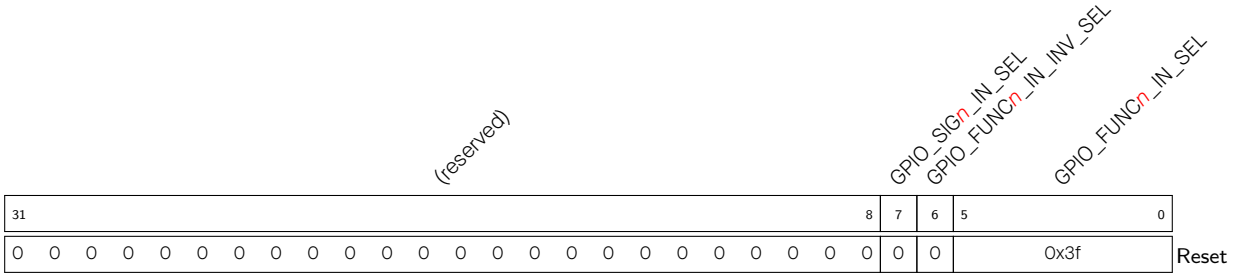
bit17：配置是否使能 CPU 中断 3：

0：不使能

1：使能

(R/W)

Register 10.29. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0 - 255) (0x015C+4**n*)



GPIO_FUNC*n*_IN_SEL 配置从 55 个 GPIO 中选择一个管脚连接输入信号 *n*。

0：选择 GPIO0

1：选择 GPIO1

.....

53：选择 GPIO53

54：选择 GPIO54

55 ~ 61：无效

62：选择恒低电平输入

63：选择恒高电平输入

(R/W)

GPIO_FUNC*n*_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

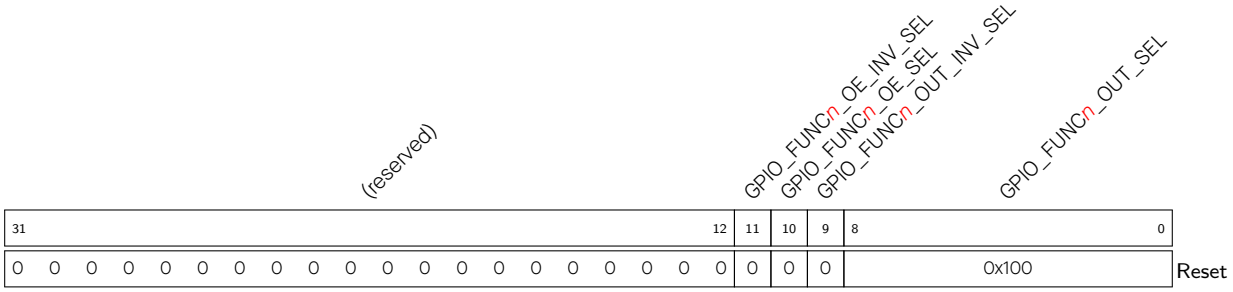
GPIO_SIG*n*_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0：旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设

1：通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 10.30. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0 - 54) (0x0558+0x4**n*)



GPIO_FUNC*n*_OUT_SEL 配置从 256 个外设信号中选择一个信号 *Y* (0 <= *Y* < 256) 输出至 GPIO*n*。

0：选择信号 0

1：选择信号 1

.....

254：选择信号 254

255：选择信号 255

256：GPIO_OUT_REG/ GPIO_OUT1_REG 和 GPIO_ENABLE_REG/ GPIO_ENABLE1_REG 中的 bit*n* 将用作输出值和输出使能信号。

257 ~ 511：无效

信号列表见表 10.12-1。

(R/W/SC)

GPIO_FUNC*n*_OUT_INV_SEL 配置是否反转输出值。

0：不反转

1：反转

(R/W/SC)

GPIO_FUNC*n*_OE_SEL 配置选择输出使能信号。

0：使用外设的输出使能信号

1：强制使用 GPIO_ENABLE_REG 的 bit*n* (*n*: 0 ~ 31) 和 GPIO_ENABLE1_REG 的 bit*n* - 32 (*n*: 32 ~ 54) 用作输出使能信号。

(R/W)

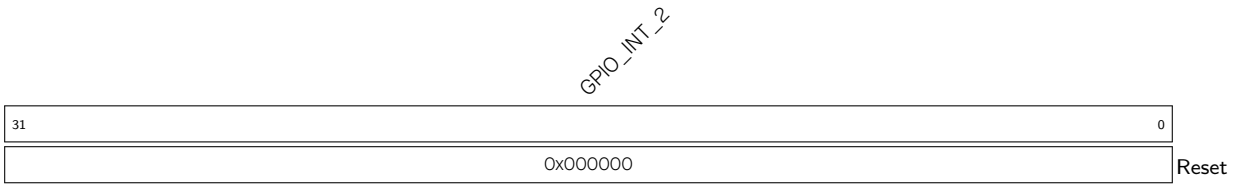
GPIO_FUNC*n*_OE_INV_SEL 配置是否反转输出使能信号。

0：不反转

1：反转

(R/W)

Register 10.31. GPIO_INTR_2_REG (0x063C)



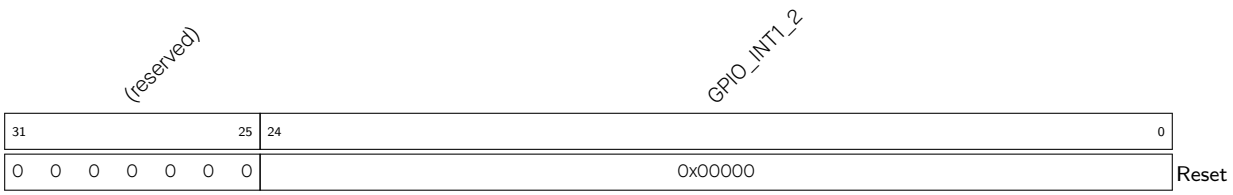
GPIO_INT_2 表示 GPIO0 ~ GPIO31 的 CPU 中断 2 的屏蔽中断状态位。每位代表：

0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。如果 GPIO_PIN*n*_REG 中 bit16 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS_REG 中相应 bit 的中断状态一致。(RO)

Register 10.32. GPIO_INTR1_2_REG (0x0640)



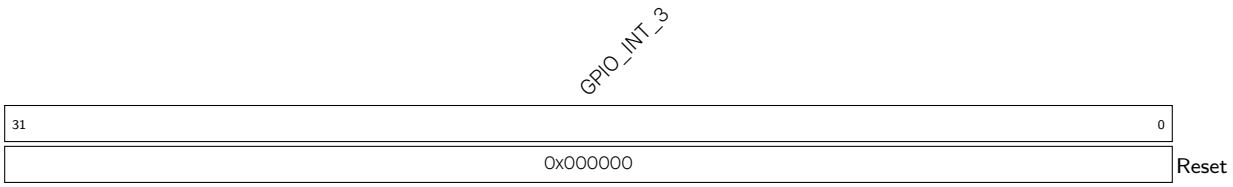
GPIO_INT1_2 表示 GPIO32 ~ GPIO54 的 CPU 中断 2 的屏蔽中断状态位。每位代表：

0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。如果 GPIO_PIN*n*_REG 中 bit16 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS1_REG 中相应 bit 的中断状态一致。(RO)

Register 10.33. GPIO_INTR_3_REG (0x0644)



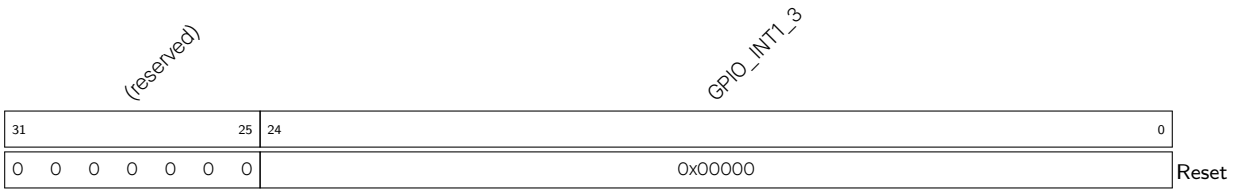
GPIO_INT_3 表示 GPIO0 ~ GPIO31 的 CPU 中断 3 的屏蔽中断状态位。每位代表：

0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit0 ~ bit31 分别对应 GPIO0 ~ GPIO31。如果 GPIO_PIN*n*_REG 中 bit17 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS_REG 中相应 bit 的中断状态一致。(RO)

Register 10.34. GPIO_INTR1_3_REG (0x0648)



GPIO_INT1_3 表示 GPIO32 ~ GPIO54 的 CPU 中断 3 的屏蔽中断状态位。每位代表：

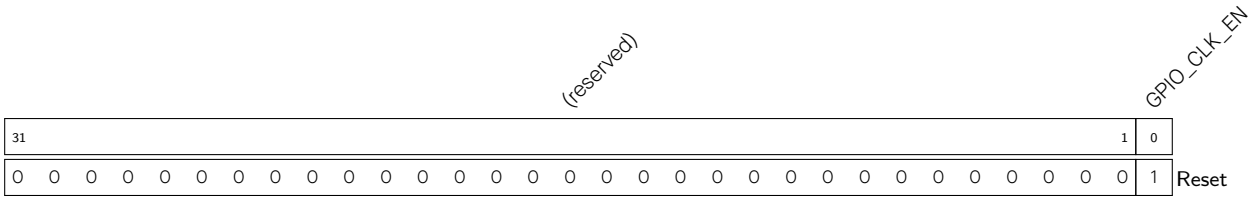
0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

Bit0 ~ bit22 分别对应 GPIO32 ~ GPIO54。如果 GPIO_PIN*n*_REG 中 bit17 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS1_REG 中相应 bit 的中断状态一致。

(RO)

Register 10.35. GPIO_CLOCK_GATE_REG (0x064C)



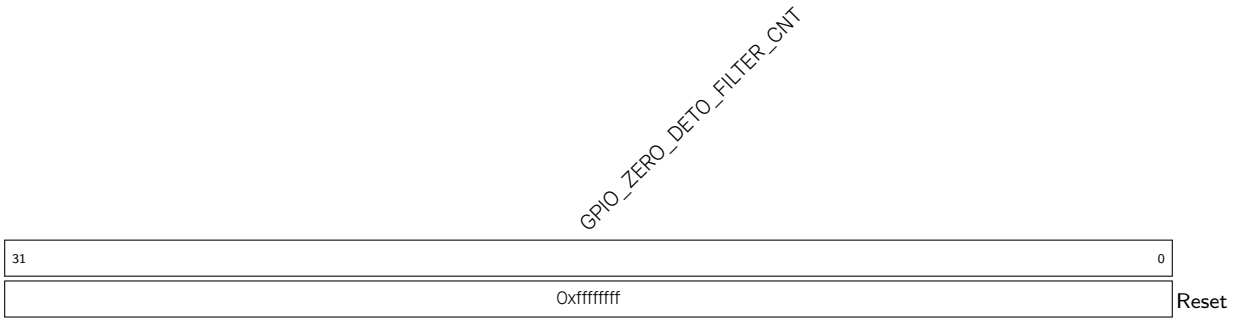
GPIO_CLK_EN 配置是否使能时钟门控。

0：不使能

1：此位置 1，则时钟自由运转。

(R/W)

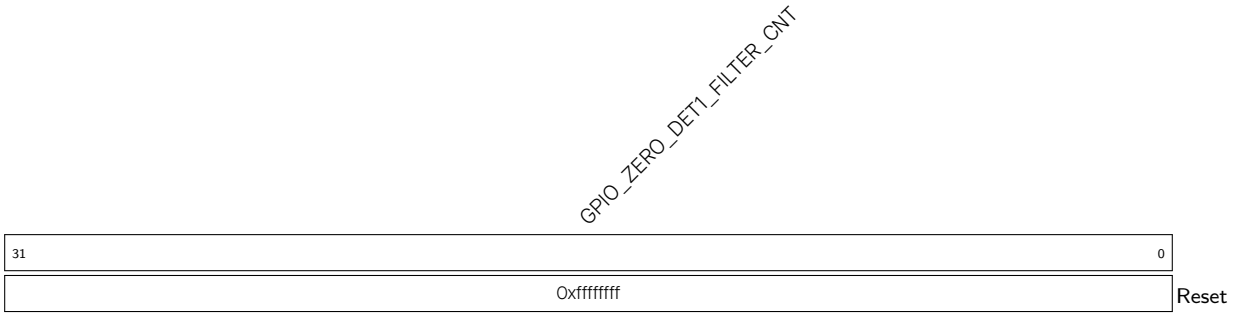
Register 10.36. GPIO_ZERO_DET0_FILTER_CNT_REG (0x0710)



GPIO_ZERO_DET0_FILTER_CNT 配置模拟电压比较器 0 的过零检测的阈值。GPIO_ZERO_DET0_FILTER_CNT 个 HP IO MUX 运行时钟周期内的中断将被 CPU 屏蔽。

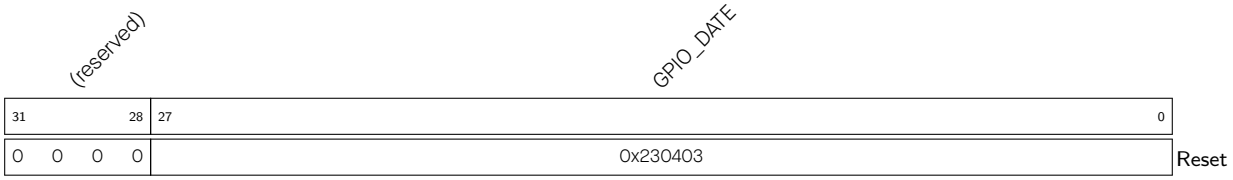
(R/W)

Register 10.37. GPIO_ZERO_DET1_FILTER_CNT_REG (0x0714)



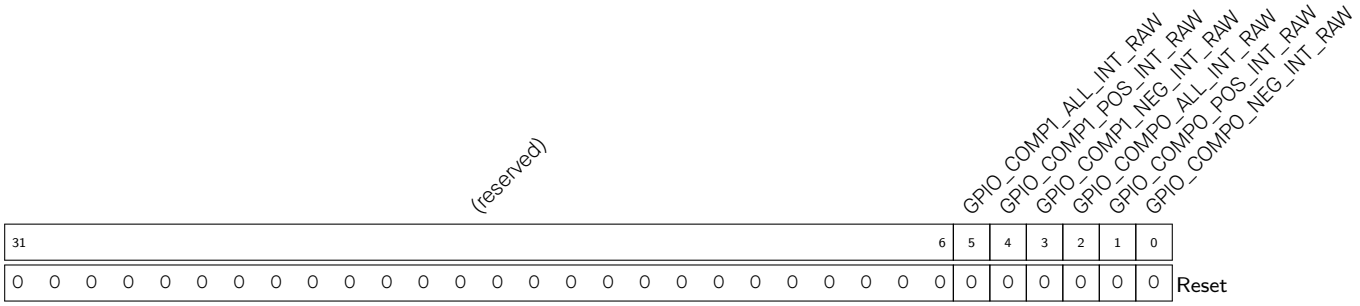
GPIO_ZERO_DET1_FILTER_CNT 配置模拟电压比较器 1 的过零检测的阈值。GPIO_ZERO_DET1_FILTER_CNT 个 HP IO MUX 运行时钟周期内的中断将被 CPU 屏蔽。(R/W)

Register 10.38. GPIO_DATE_REG (0x07FC)



GPIO_DATE 版本控制寄存器。(R/W)

Register 10.39. GPIO_INT_RAW_REG (0x0700)



GPIO_COMP0_NEG_INT_RAW GPIO_COMP0_NEG_INT 的原始中断状态。(R/WTC/SS)

GPIO_COMP0_POS_INT_RAW GPIO_COMP0_POS_INT 的原始中断状态。(R/WTC/SS)

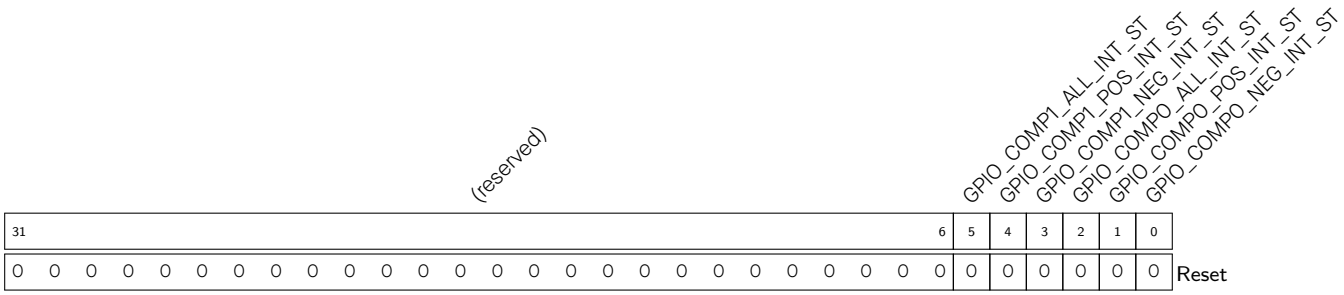
GPIO_COMP0_ALL_INT_RAW GPIO_COMP0_ALL_INT 的原始中断状态。(R/WTC/SS)

GPIO_COMP1_NEG_INT_RAW GPIO_COMP1_NEG_INT 的原始中断状态。(R/WTC/SS)

GPIO_COMP1_POS_INT_RAW GPIO_COMP1_POS_INT 的原始中断状态。(R/WTC/SS)

GPIO_COMP1_ALL_INT_RAW GPIO_COMP1_ALL_INT 的原始中断状态。(R/WTC/SS)

Register 10.40. GPIO_INT_ST_REG (0x0704)



GPIO_COMP0_NEG_INT_ST GPIO_COMP0_NEG_INT 的屏蔽中断状态。(RO)

GPIO_COMP0_POS_INT_ST GPIO_COMP0_POS_INT 的屏蔽中断状态。(RO)

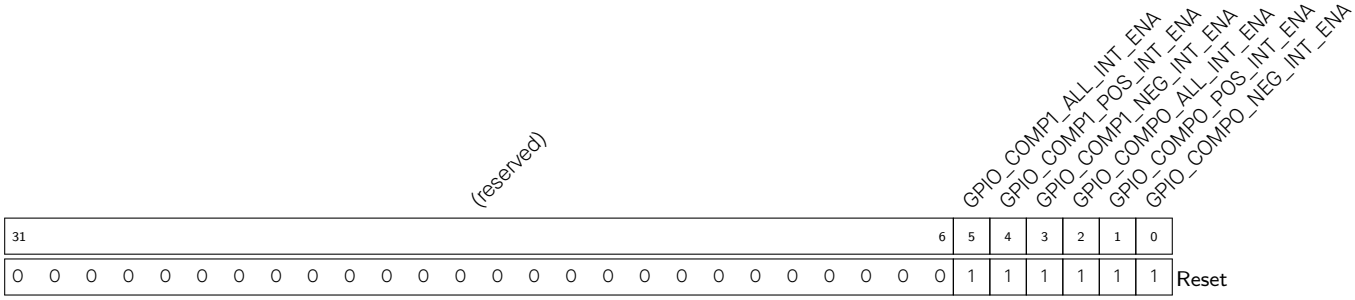
GPIO_COMP0_ALL_INT_ST GPIO_COMP0_ALL_INT 的屏蔽中断状态。(RO)

GPIO_COMP1_NEG_INT_ST GPIO_COMP1_NEG_INT 的屏蔽中断状态。(RO)

GPIO_COMP1_POS_INT_ST GPIO_COMP1_POS_INT 的屏蔽中断状态。(RO)

GPIO_COMP1_ALL_INT_ST GPIO_COMP1_ALL_INT 的屏蔽中断状态。(RO)

Register 10.41. GPIO_INT_ENA_REG (0x0708)



GPIO_COMP0_NEG_INT_ENA 写 1 使能 GPIO_COMP0_NEG_INT 中断。(R/W)

GPIO_COMP0_POS_INT_ENA 写 1 使能 GPIO_COMP0_POS_INT 中断。(R/W)

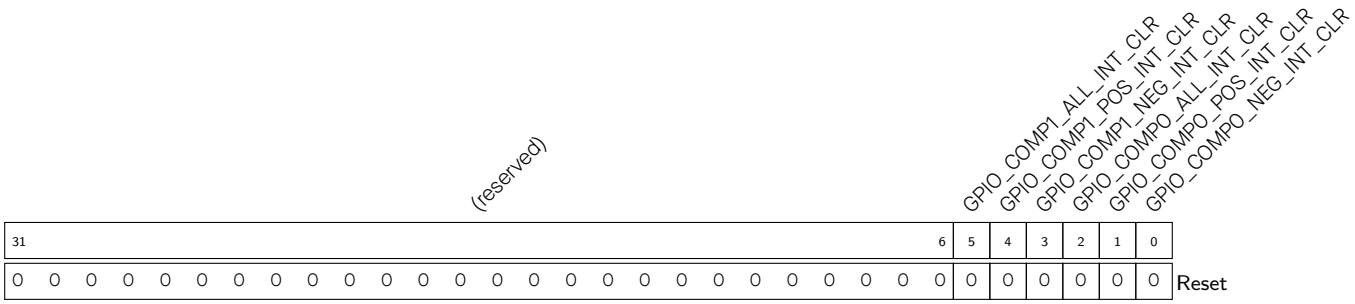
GPIO_COMP0_ALL_INT_ENA 写 1 使能 GPIO_COMP0_ALL_INT 中断。(R/W)

GPIO_COMP1_NEG_INT_ENA 写 1 使能 GPIO_COMP1_NEG_INT 中断。(R/W)

GPIO_COMP1_POS_INT_ENA 写 1 使能 GPIO_COMP1_POS_INT 中断。(R/W)

GPIO_COMP1_ALL_INT_ENA 写 1 使能 GPIO_COMP1_ALL_INT 中断。(R/W)

Register 10.42. GPIO_INT_CLR_REG (0x070C)



GPIO_COMP0_NEG_INT_CLR 写 1 清除 GPIO_COMP0_NEG_INT。(WT)

GPIO_COMP0_POS_INT_CLR 写 1 清除 GPIO_COMP0_POS_INT。(WT)

GPIO_COMP0_ALL_INT_CLR 写 1 清除 GPIO_COMP0_ALL_INT。(WT)

GPIO_COMP1_NEG_INT_CLR 写 1 清除 GPIO_COMP1_NEG_INT。(WT)

GPIO_COMP1_POS_INT_CLR 写 1 清除 GPIO_COMP1_POS_INT。(WT)

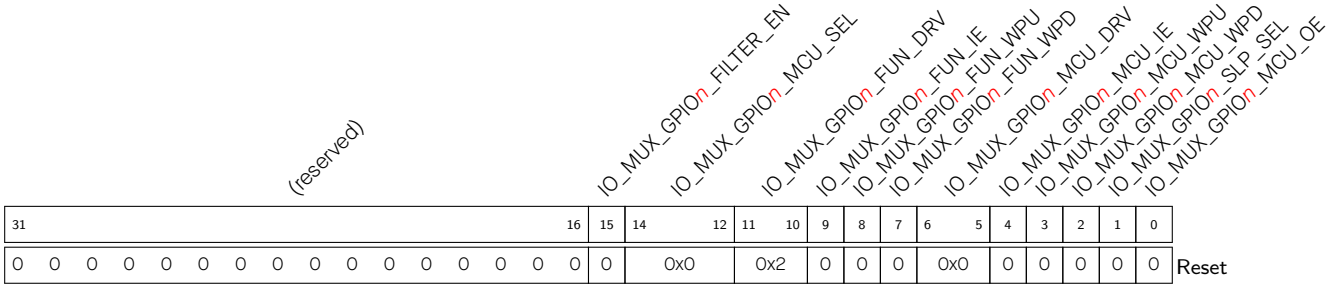
GPIO_COMP1_ALL_INT_CLR 写 1 清除 GPIO_COMP1_ALL_INT。(WT)

10.20.2 HP IO MUX 寄存器

本小节的所有地址均为相对于 HP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 10.43. IO_MUX_GPIO*n*_REG (*n*: 0 - 54) (0x0004+4**n*)



IO_MUX_GPIO*n*_MCU_OE 配置是否使能睡眠模式下 GPIO*n* 的输出。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_SLP_SEL 配置是否使能 GPIO*n* 进入睡眠模式。

0：不进入

1：进入

(R/W)

IO_MUX_GPIO*n*_MCU_WPD 配置是否使能睡眠模式下 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_WPU 配置是否使能睡眠模式下 GPIO*n* 的上拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_IE 配置是否使能睡眠模式下 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_DRV 配置睡眠模式下 GPIO*n* 的驱动强度。

0：~5 mA

1：~10 mA

2：~20 mA

3：~40 mA

(R/W)

见下页

Register 10.43. IO_MUX_GPIO*n*_REG (*n*: 0 - 54) (0x0004+4**n*)

接上页

IO_MUX_GPIO*n*_FUN_WPD 配置是否使能 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_FUN_WPU 配置是否使能 GPIO*n* 的上拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_FUN_IE 配置是否使能 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_FUN_DRV 配置 GPIO*n* 的驱动强度。

0：~5 mA

1：~10 mA

2：~20 mA

3：~40 mA

(R/W)

IO_MUX_GPIO*n*_MCU_SEL 配置选择管脚的 HP IO MUX 功能。

0：选择功能 0

1：选择功能 1

.....

(R/W)

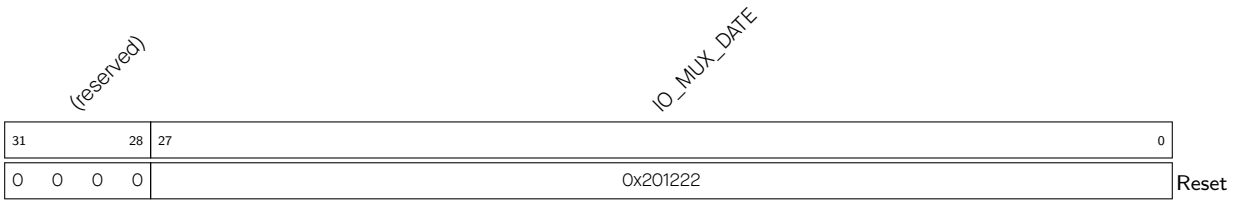
IO_MUX_GPIO*n*_FILTER_EN 配置是否使能管脚输入信号的滤波功能。

0：不使能

1：使能

(R/W)

Register 10.44. IO_MUX_DATE_REG (0x0104)



IO_MUX_DATE 版本控制寄存器。

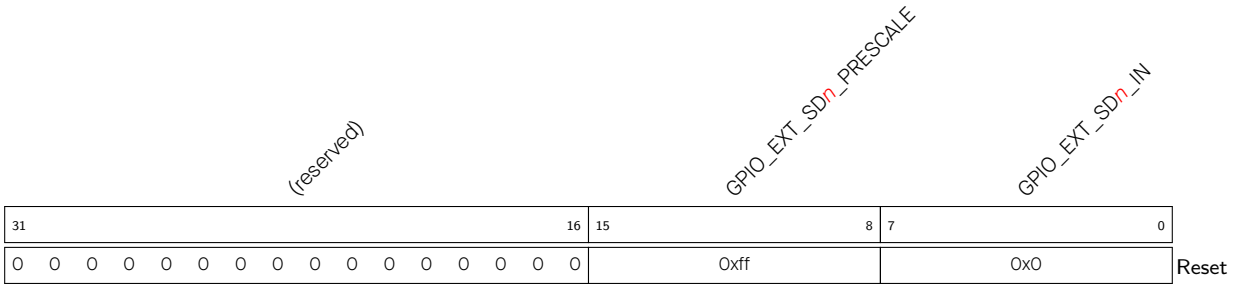
(R/W)

10.20.3 GPIO EXT 寄存器

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址 + 0x0F00 的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 10.45. GPIO_EXT_SIGMADELTA*n*_REG (*n*: 0 - 7) (0x0000+0x4**n*)



GPIO_EXT_SD*n*_IN (*n*: 0 - 7) 配置 SDM 输出信号的占空比。

(R/W)

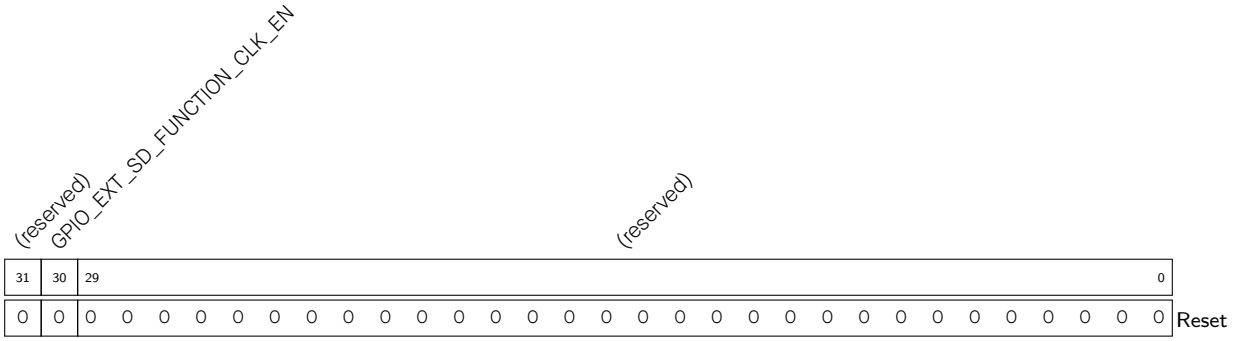
GPIO_EXT_SD*n*_PRESCALE (*n*: 0 - 7) 配置 HP IO MUX 运行时钟的分频系数，分频系数有效值为 0 ~ 255。

0：不分频

1 ~ 255：2 ~ 256 分频

(R/W)

Register 10.46. GPIO_EXT_SIGMADELTA_MISC_REG (0x0024)



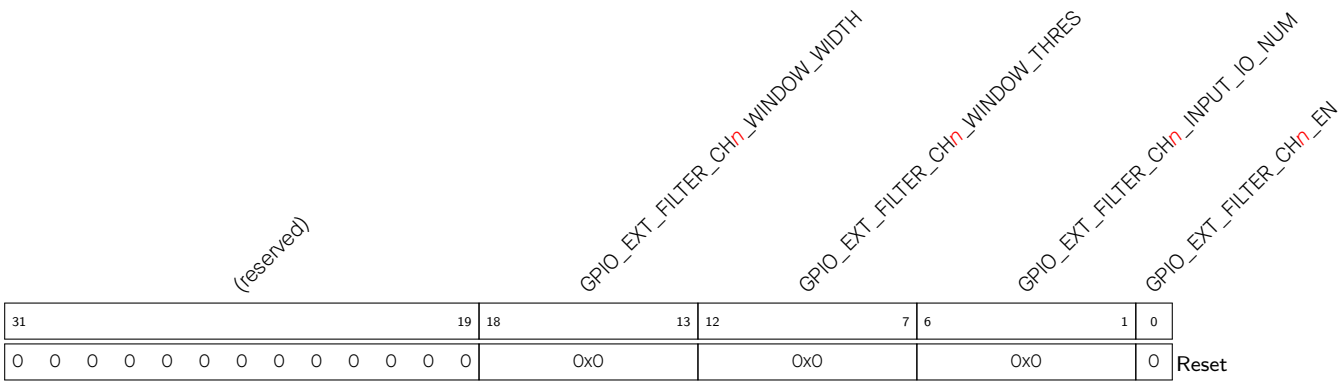
GPIO_EXT_SD_FUNCTION_CLK_EN 配置是否使能 SDM 时钟。

0：不使能

1：使能

(R/W)

Register 10.47. GPIO_EXT_GLITCH_FILTER_CH*n*_REG (*n*: 0 - 7) (0x0030+0x4*n*)



GPIO_EXT_FILTER_CH*n*_EN 配置是否使能毛刺滤波器的通道 *n*。

0：不使能

1：使能

(R/W)

GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM 配置选择毛刺滤波器的 GPIO 输入。

0：选择 GPIO0

1：选择 GPIO1

.....

53：选择 GPIO53

54：选择 GPIO54

(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 配置毛刺滤波器的窗口阈值。窗口阈值应小于或等于 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH。

单位：HP IO MUX 运行时钟的一个周期

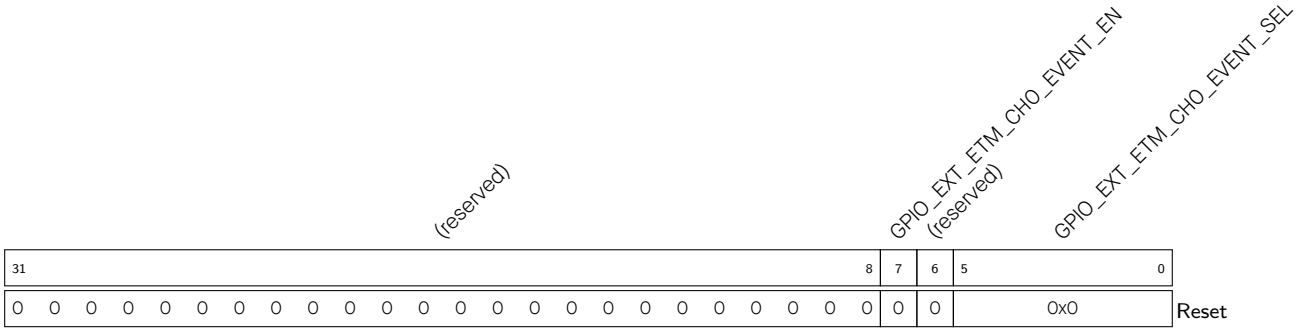
(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 配置毛刺滤波器的窗口宽度。窗口宽度的有效值为 0 ~ 62。63 为保留值，不能使用。

单位：HP IO MUX 运行时钟的一个周期

(R/W)

Register 10.48. GPIO_EXT_ETM_EVENT_CH*n*_CFG_REG (*n*: 0 - 7) (0x0060+0x4*n)



GPIO_EXT_ETM_CH*n*_EVENT_SEL 配置选择 ETM 事件通道使用的 GPIO。

0：选择 GPIO0

1：选择 GPIO1

.....

53：选择 GPIO53

54：选择 GPIO54

(R/W)

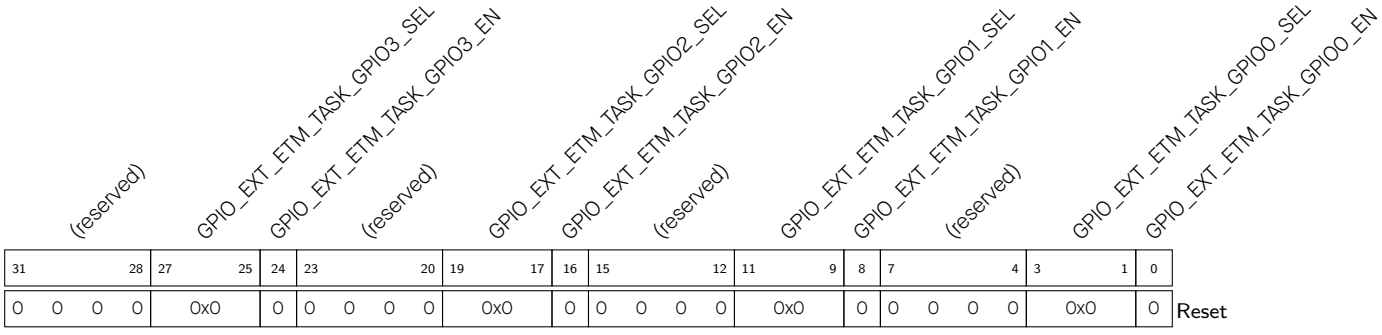
GPIO_EXT_ETM_CH*n*_EVENT_EN 配置是否使能 ETM 事件发送。

0：不使能

1：使能

(R/W)

Register 10.49. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x00A0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 0 - 3) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 0 - 3) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

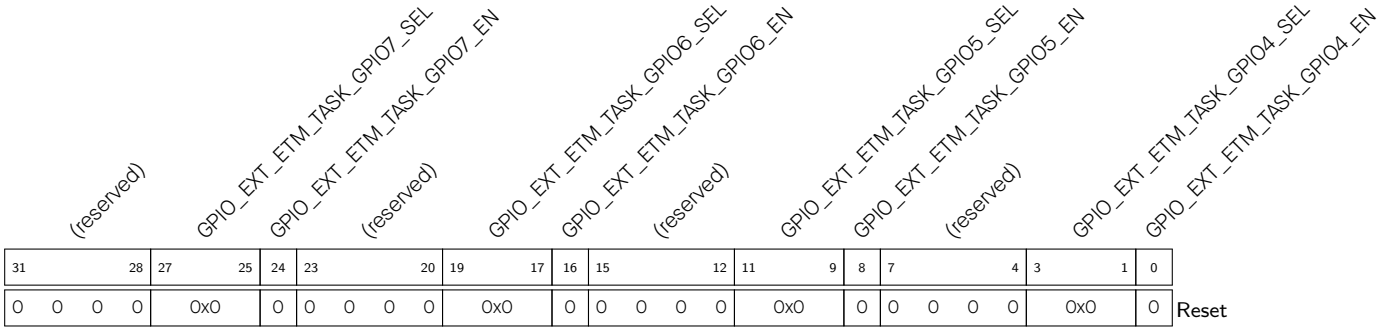
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.50. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x00A4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 4 - 7) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 4 - 7) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.51. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x00A8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 8 - 11) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 8 - 11) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

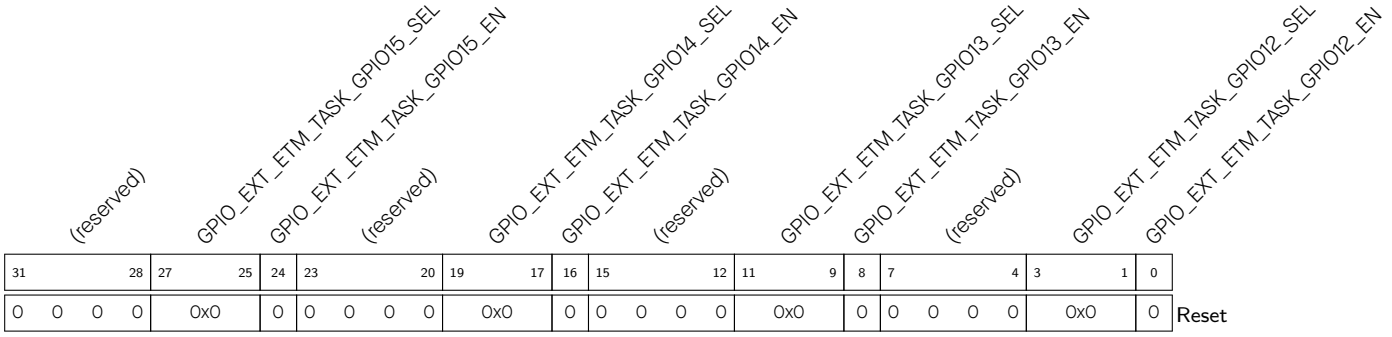
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.52. GPIO_EXT_ETM_TASK_P3_CFG_REG (0x00AC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 12 - 15) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 12 - 15) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.53. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x00B0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 16 - 19) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 16 - 19) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.54. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x00B4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 20 - 23) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 20 - 23) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

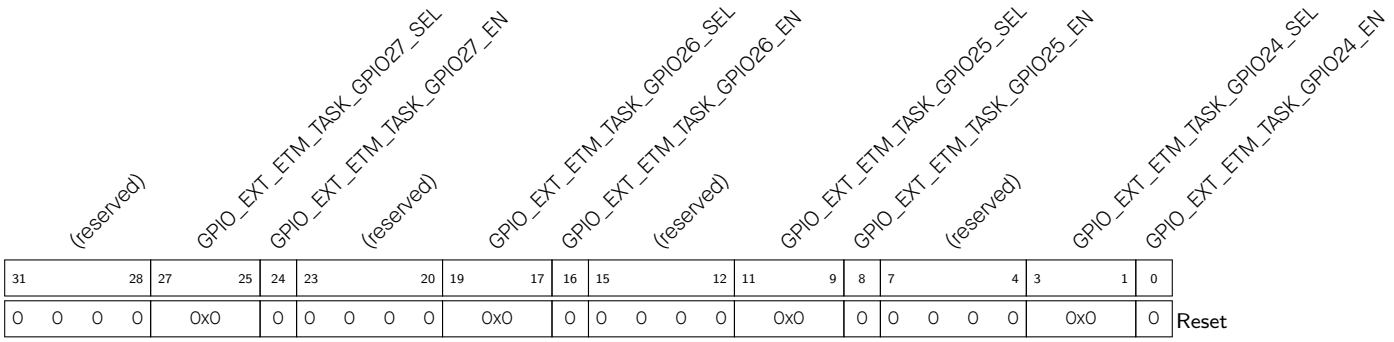
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.55. GPIO_EXT_ETM_TASK_P6_CFG_REG (0x00B8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 24 - 27) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 24 - 27) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.56. GPIO_EXT_ETM_TASK_P7_CFG_REG (0x00BC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 28 - 31) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 28 - 31) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

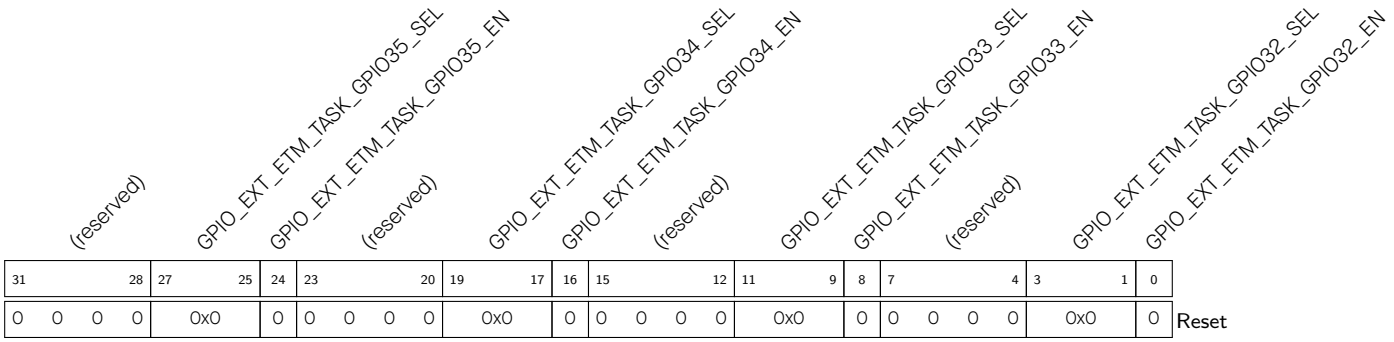
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.57. GPIO_EXT_ETM_TASK_P8_CFG_REG (0x00C0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 32 - 35) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 32 - 35) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

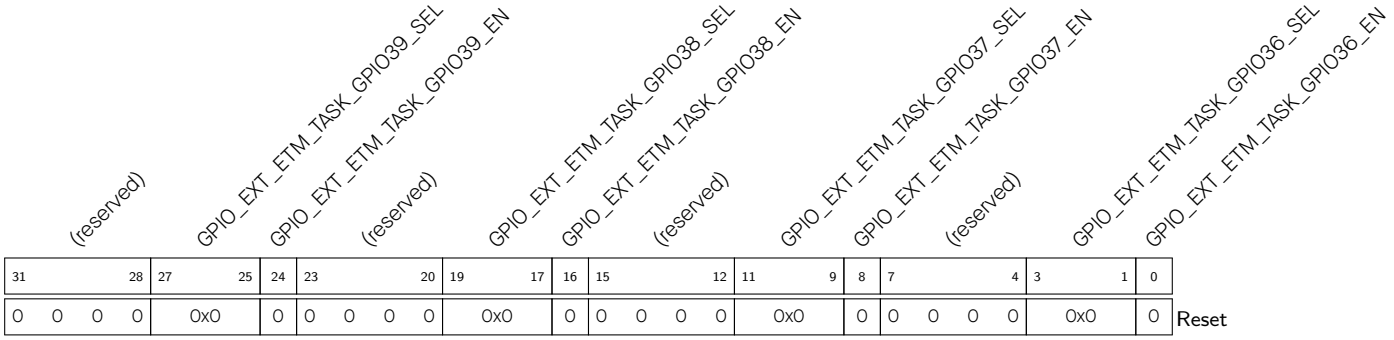
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.58. GPIO_EXT_ETM_TASK_P9_CFG_REG (0x00C4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 36 - 39) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 36 - 39) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

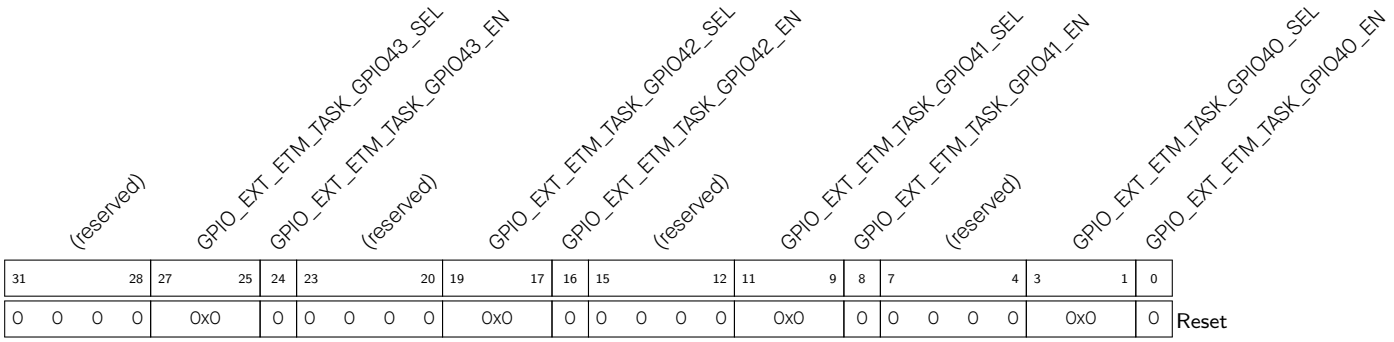
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.59. GPIO_EXT_ETM_TASK_P10_CFG_REG (0x00C8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 40 - 43) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 40 - 43) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.60. GPIO_EXT_ETM_TASK_P11_CFG_REG (0x00CC)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 44 - 47) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 44 - 47) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

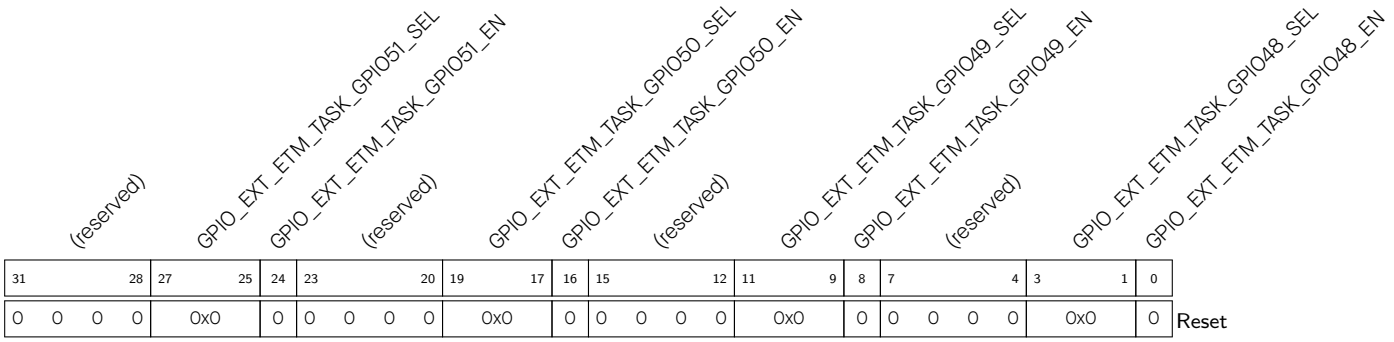
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.61. GPIO_EXT_ETM_TASK_P12_CFG_REG (0x00D0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 48 - 51) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 48 - 51) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.62. GPIO_EXT_ETM_TASK_P13_CFG_REG (0x00D4)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n*: 52 - 54) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n*: 52 - 54) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

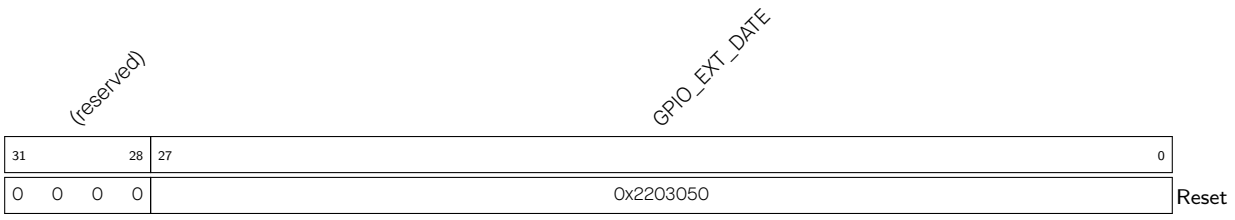
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 10.63. GPIO_EXT_VERSION_REG (0x00FC)



GPIO_EXT_DATE 版本控制寄存器。

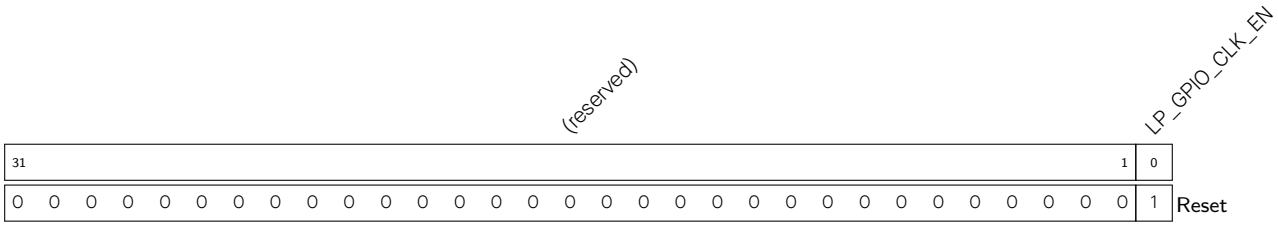
(R/W)

10.20.4 LP GPIO 矩阵寄存器

本小节的所有地址均为相对于 LP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 10.64. LP_GPIO_CLK_EN_REG (0x0000)



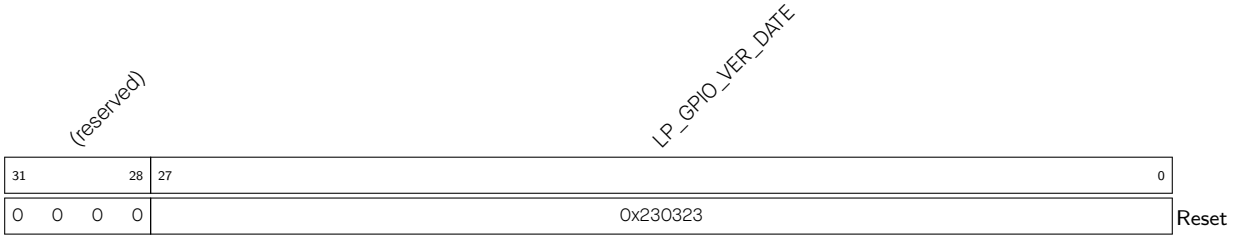
LP_GPIO_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能，则时钟自由运转。

(R/W)

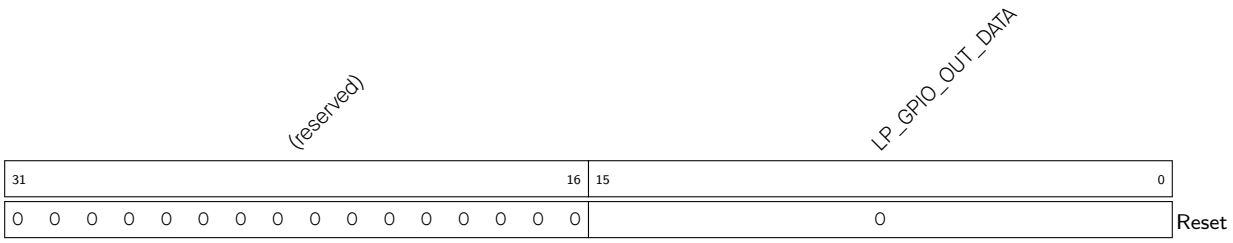
Register 10.65. LP_GPIO_VER_DATE_REG (0x0004)



LP_GPIO_VER_DATE 版本控制寄存器。

(R/W)

Register 10.66. LP_GPIO_OUT_REG (0x0008)



LP_GPIO_OUT_DATA 配置简单 GPIO 输出模式下 GPIO0 ~ GPIO15 的输出值。

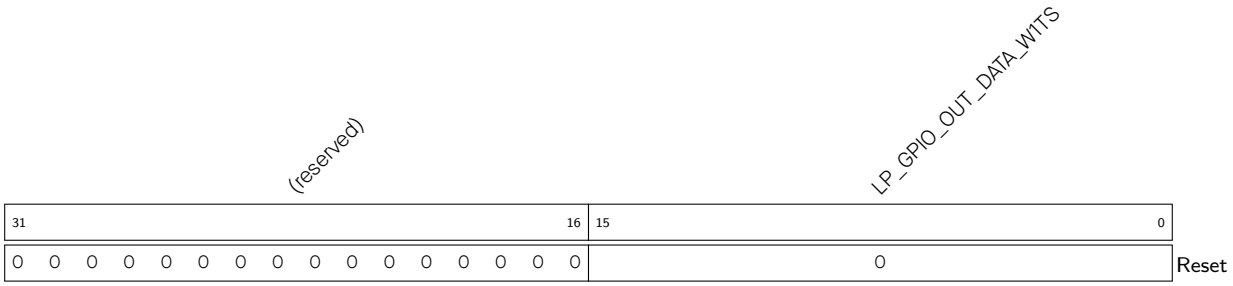
0：低电平

1：高电平

bit0 ~ bit15 的值分别对应 GPIO0 ~ GPIO15 的输出值。bit16 ~ bit31 无效。

(R/W/WTC)

Register 10.67. LP_GPIO_OUT_W1TS_REG (0x000C)



LP_GPIO_OUT_DATA_W1TS 配置是否置位 GPIO0 ~ GPIO15 的输出寄存器 LP_GPIO_OUT_REG。

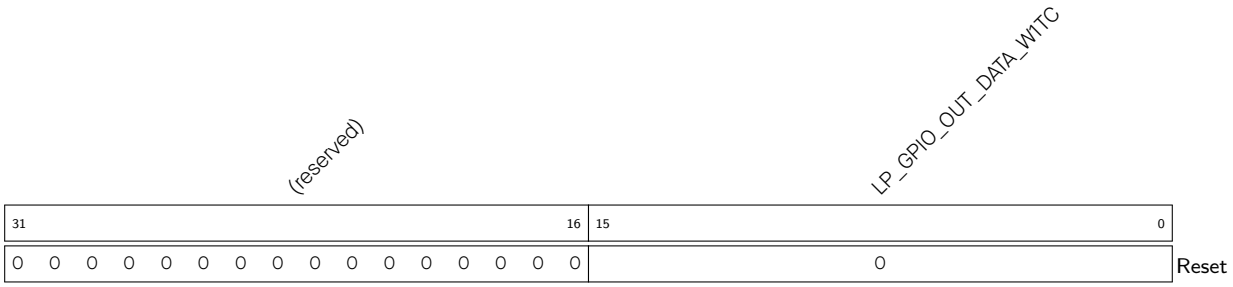
0：不置位

1：LP_GPIO_OUT_REG 中相应位也将置 1

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。推荐使用此寄存器来置位 LP_GPIO_OUT_REG。

(WT)

Register 10.68. LP_GPIO_OUT_W1TC_REG (0x0010)



LP_GPIO_OUT_DATA_W1TC 配置是否清零 GPIO0 ~ GPIO15 的输出寄存器 LP_GPIO_OUT_REG。

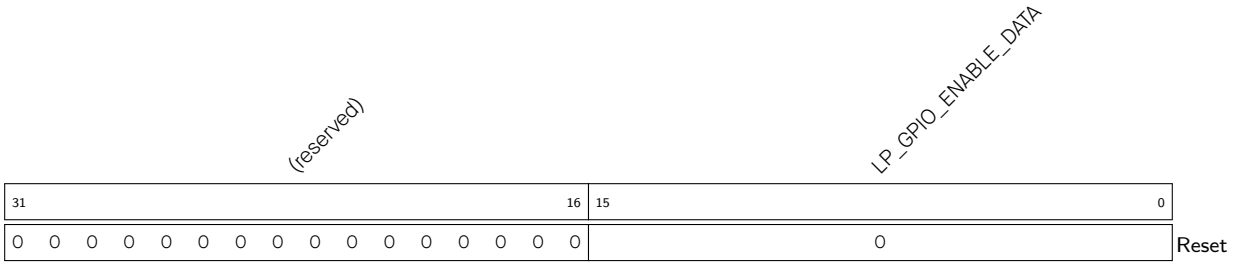
0：不清零

1：LP_GPIO_OUT_REG 中相应位将被清零。

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。推荐使用此寄存器来清零 LP_GPIO_OUT_REG。

(WT)

Register 10.69. LP_GPIO_ENABLE_REG (0x0014)



LP_GPIO_ENABLE_DATA 配置是否使能 GPIO0 ~ GPIO15 的输出。

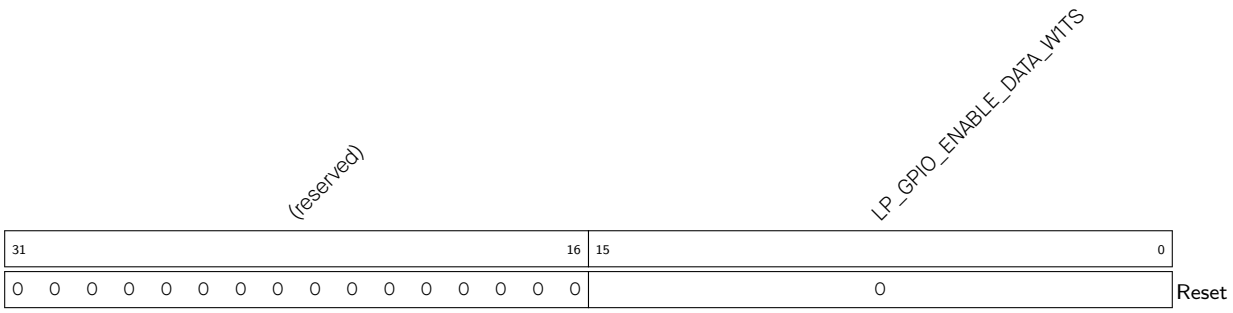
0：不使能

1：使能

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。

(R/W/WTC)

Register 10.70. LP_GPIO_ENABLE_W1TS_REG (0x0018)



LP_GPIO_ENABLE_DATA_W1TS 配置是否置位 GPIO0 ~ GPIO15 的输出使能寄存器 LP_GPIO_ENABLE_REG。

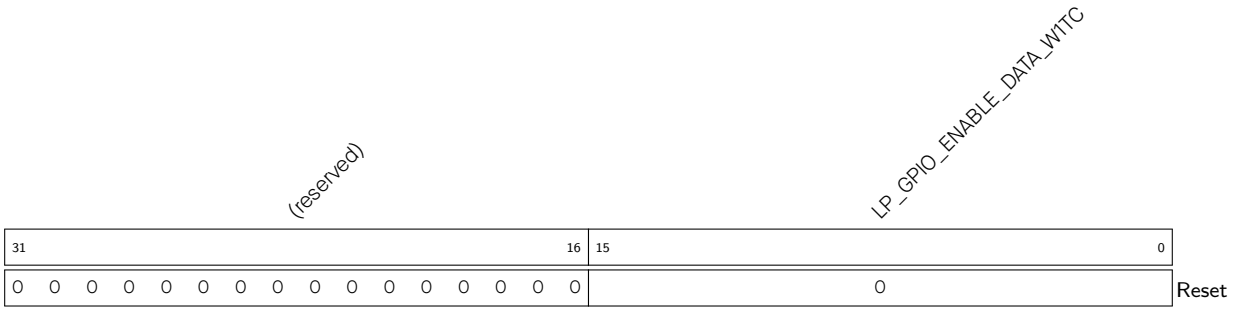
0：不置位

1：LP_GPIO_ENABLE_REG 中相应位也将置 1

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。推荐使用该寄存器置位 LP_GPIO_ENABLE_REG。

(WT)

Register 10.71. LP_GPIO_ENABLE_W1TC_REG (0x001C)



LP_GPIO_ENABLE_DATA_W1TC 配置是否清零 GPIO0 ~ GPIO15 的输出使能寄存器 LP_GPIO_ENABLE_REG。

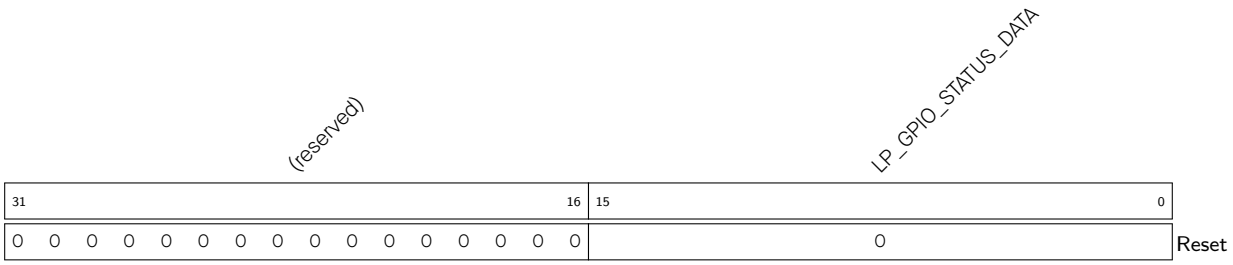
0：不清零

1：LP_GPIO_ENABLE_REG 中相应位将被清零。

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。推荐使用该寄存器清零 LP_GPIO_ENABLE_REG。

(WT)

Register 10.72. LP_GPIO_STATUS_REG (0x0020)



LP_GPIO_STATUS_DATA 表示 GPIO0 ~ GPIO15 的中断状态，软件可配置。

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。

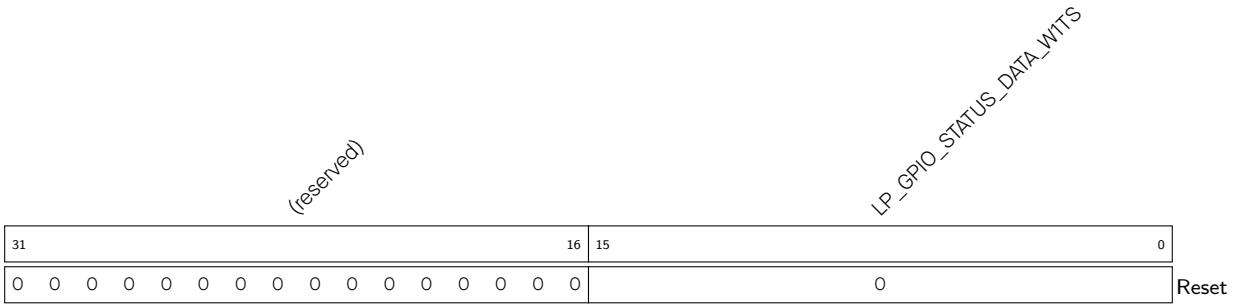
每一位均代表对应 GPIO 的状态：

0：表示对应 GPIO 未产生 LP_GPIO_PIN*n*_INT_TYPE 所配置的中断，或软件配置该位寄存器的值为 0。

1：表示对应 GPIO 产生了 LP_GPIO_PIN*n*_INT_TYPE 所选择的中断，或软件配置该位寄存器的值为 1。

(R/W/WTC)

Register 10.73. LP_GPIO_STATUS_W1TS_REG (0x0024)



LP_GPIO_STATUS_DATA_W1TS 配置是否置位 GPIO0 ~ GPIO15 的中断状态寄存器 LP_GPIO_STATUS_DATA。

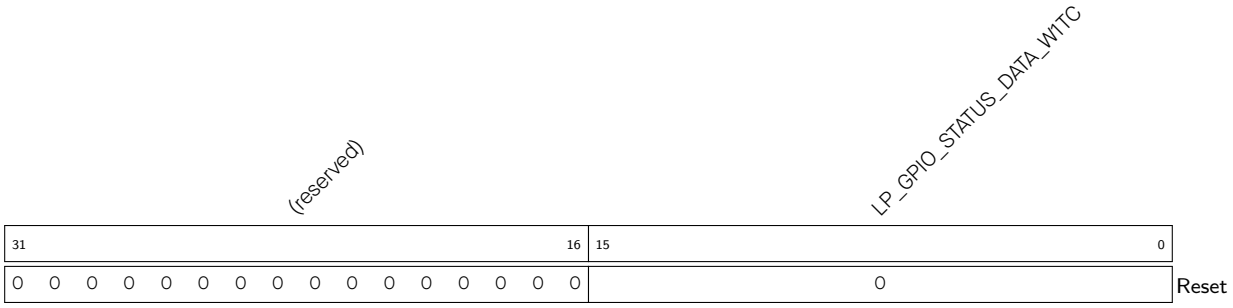
bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。

每一位置 1，则 LP_GPIO_STATUS_DATA 中的相应位也置 1。

推荐使用此寄存器来置位 LP_GPIO_STATUS_DATA。

(WT)

Register 10.74. LP_GPIO_STATUS_W1TC_REG (0x0028)



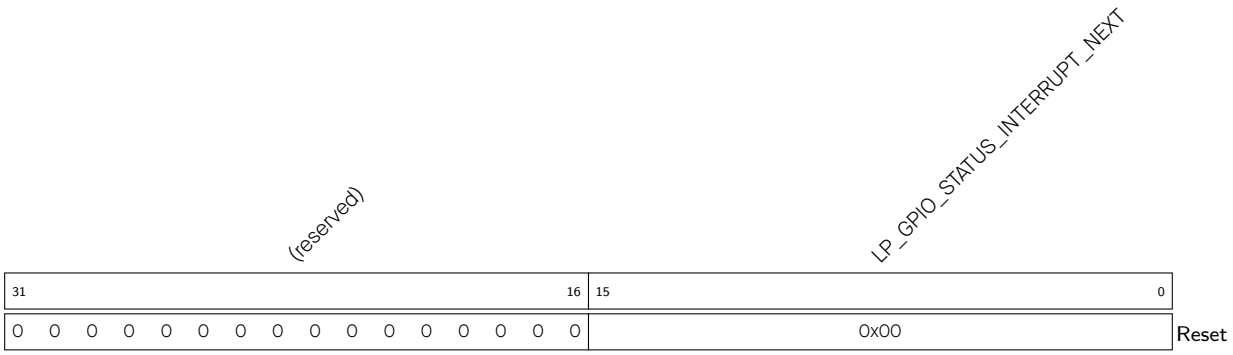
LP_GPIO_STATUS_DATA_W1TC 配置是否清零 GPIO0 ~ GPIO15 的中断状态寄存器 LP_GPIO_STATUS_DATA。

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。

每一位置 1，则 LP_GPIO_STATUS_DATA 中的相应位将被清零。推荐使用此寄存器来清零 LP_GPIO_STATUS_DATA。

(WT)

Register 10.75. LP_GPIO_STATUS_NEXT_REG (0x002C)



LP_GPIO_STATUS_INTERRUPT_NEXT 表示 GPIO0 ~ GPIO15 的中断源信号。

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。每位代表：

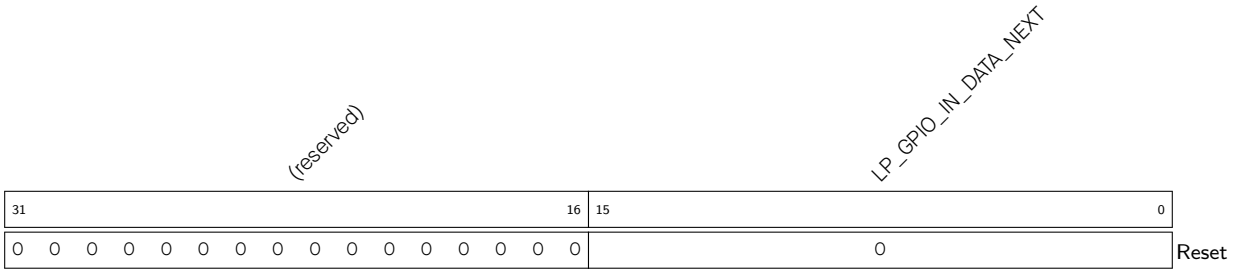
0：表示 GPIO 未产生 LP_GPIO_PIN*n*_INT_TYPE 所配置的中断。

1：GPIO 产生了 LP_GPIO_PIN*n*_INT_TYPE 所配置的中断。

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。

(RO)

Register 10.76. LP_GPIO_IN_REG (0x0030)



LP_GPIO_IN_DATA_NEXT 表示 GPIO0 ~ GPIO15 的输入值。每一位均代表一个管脚的输入值：

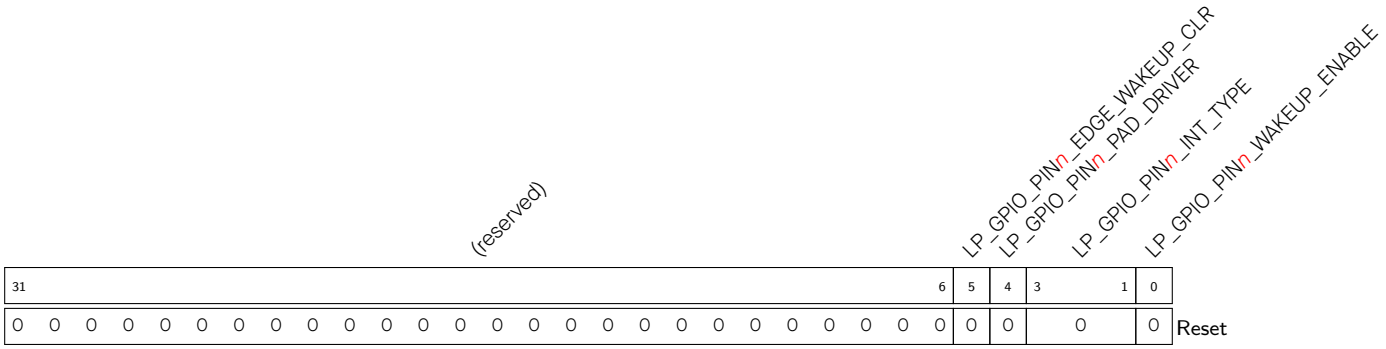
0：低电平

1：高电平

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。bit16 ~ bit31 无效。

(RO)

Register 10.77. LP_GPIO_PIN*n*_REG (*n*: 0 - 15) (0x0034+4**n*)



LP_GPIO_PIN*n*_WAKEUP_ENABLE 配置是否使能 GPIO 唤醒功能。

0：不使能

1：使能

该功能只能将 CPU 从 Deep-sleep 中唤醒。

(R/W)

LP_GPIO_PIN*n*_INT_TYPE 配置 GPIO 中断类型。

0：关闭 GPIO 中断

1：上升沿触发

2：下降沿触发

3：任一沿触发

4：低电平触发

5：高电平触发

(R/W)

LP_GPIO_PIN*n*_PAD_DRIVER 配置选择管脚的输出驱动模式。

0：正常输出

1：开漏输出

(R/W)

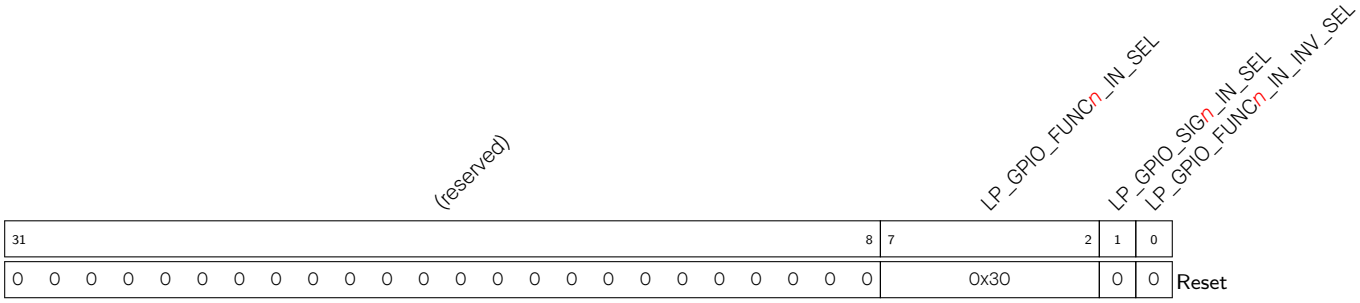
LP_GPIO_PIN*n*_EDGE_WAKEUP_CLR 配置是否在 PMU 中清除跳变沿触发的 GPIO 唤醒源信号。

0：无效

1：LP_PMU_xxx 对应位将被清除。

(WT)

Register 10.78. LP_GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0 - 15) (0x0074+4**n*)



LP_GPIO_FUNC*n*_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

LP_GPIO_SIG*n*_IN_SEL 配置是否通过 LP GPIO 交换矩阵连接信号。

0：旁路 LP GPIO 交换矩阵，即直接通过 LP IO MUX 连接信号与外设

1：通过 LP GPIO 交换矩阵连接信号与外设

(R/W)

LP_GPIO_FUNC*n*_IN_SEL 配置从 16 个 LP GPIO 中选择一个管脚连接输入信号 *n*。

0：选择 GPIO0

1：选择 GPIO1

.....

14：选择 GPIO14

15：选择 GPIO15

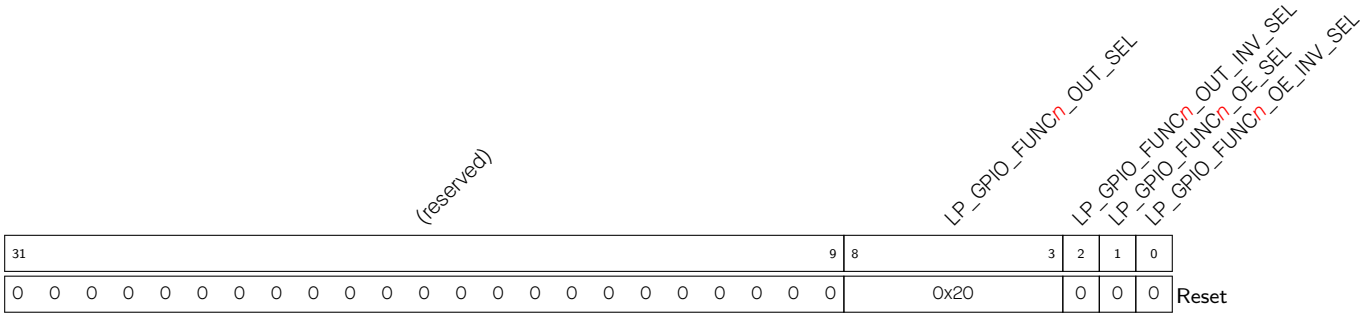
16 ~ 31：无效

32 ~ 47：选择恒低电平输入

48 ~ 63：选择恒高电平输入

(R/W)

Register 10.79. LP_GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0 - 15) (0x00F4+4**n*)



LP_GPIO_FUNC*n*_OE_INV_SEL 配置是否反转输出使能信号。

0：不反转

1：反转

(R/W)

LP_GPIO_FUNC*n*_OE_SEL 配置选择输出使能信号。

0：使用外设的输出使能信号

1：强制使用 LP_GPIO_ENABLE_REG 的 bit*n* 用作输出使能信号。

(R/W)

LP_GPIO_FUNC*n*_OUT_INV_SEL 配置是否反转输出值。

0：不反转

1：反转

(R/W)

LP_GPIO_FUNC*n*_OUT_SEL 配置从 32 个外设信号中选择一个信号 *Y* (0 <= *Y* < 32) 输出至 GPIO*n*。

0：选择信号 0

1：选择信号 1

.....

30：选择信号 30

31：选择信号 31

或者

32：LP_GPIO_OUT_REG 和 LP_GPIO_ENABLE_REG 中的 bit*n* 将用作输出值和输出使能信号。

信号列表见表 10.13-1。

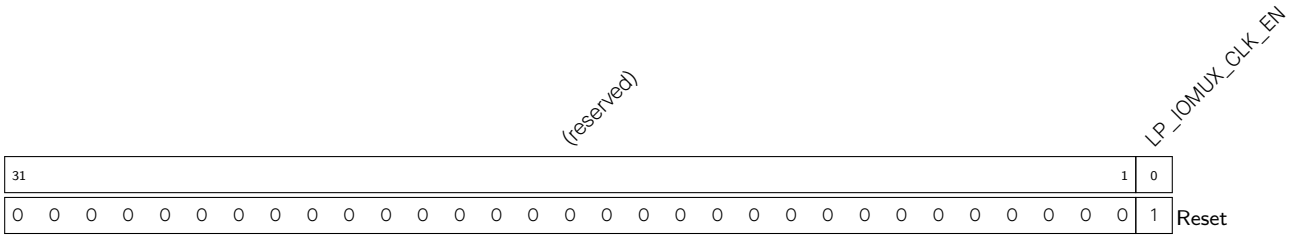
(R/W)

10.20.5 LP IO MUX 寄存器

本小节的所有地址均为相对于 LP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 10.80. LP_IOMUX_CLK_EN_REG (0x0000)



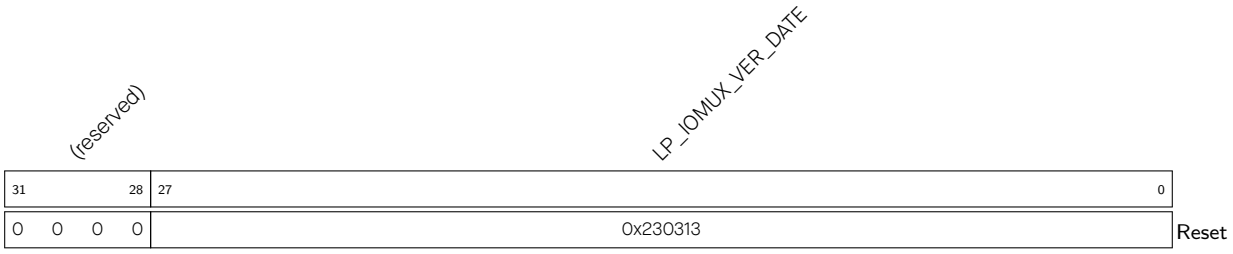
LP_IOMUX_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能，则时钟自由运转。

(R/W)

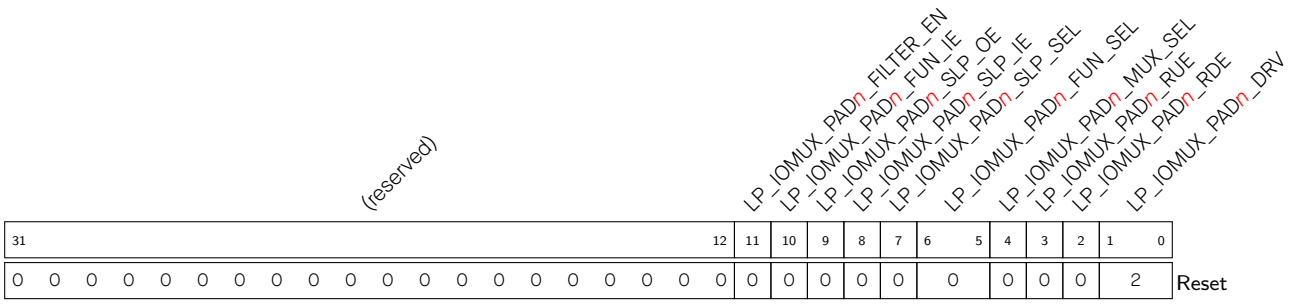
Register 10.81. LP_IOMUX_VER_DATE_REG (0x0004)



LP_IOMUX_VER_DATE 版本控制寄存器。

(R/W)

Register 10.82. LP_IOMUX_PAD*n*_REG (*n*: 0 - 15) (0x0008+4**n*)



LP_IOMUX_PAD*n*_DRV 配置 GPIO*n* 的驱动强度。

0：~5 mA

1：~10 mA

2：~20 mA

3：~40 mA

(R/W)

LP_IOMUX_PAD*n*_RDE 配置是否使能 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

LP_IOMUX_PAD*n*_RUE 配置是否使能 GPIO*n* 的上拉电阻。

0：不使能

1：使能

(R/W)

LP_IOMUX_PAD*n*_MUX_SEL 配置 GPIO*n* 是由 HP IO MUX 控制还是由 LP IO MUX 控制。

0：GPIO*n* 由 HP IO MUX 控制

1：GPIO*n* 由 LP IO MUX 控制

(R/W)

LP_IOMUX_PAD*n*_FUN_SEL 配置选择该信号的 LP IO MUX 功能。

0：选择功能 0

1：选择功能 1

.....

(R/W)

LP_IOMUX_PAD*n*_SLP_SEL 配置是否使能 GPIO*n* 进入睡眠模式。

0：不进入

1：进入

(R/W)

LP_IOMUX_PAD*n*_SLP_IE 配置是否使能睡眠模式下 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

见下页

Register 10.83. LP_IOMUX_PADn_REG (n: 0 - 15) (0x0008+4*n)

接上页

LP_IOMUX_PADn_SLP_OE 配置是否使能睡眠模式下 GPIOn 的输出。

0：不使能

1：使能

(R/W)

LP_IOMUX_PADn_FUN_IE 配置是否使能 GPIOn 的输入。

0：不使能

1：使能

(R/W)

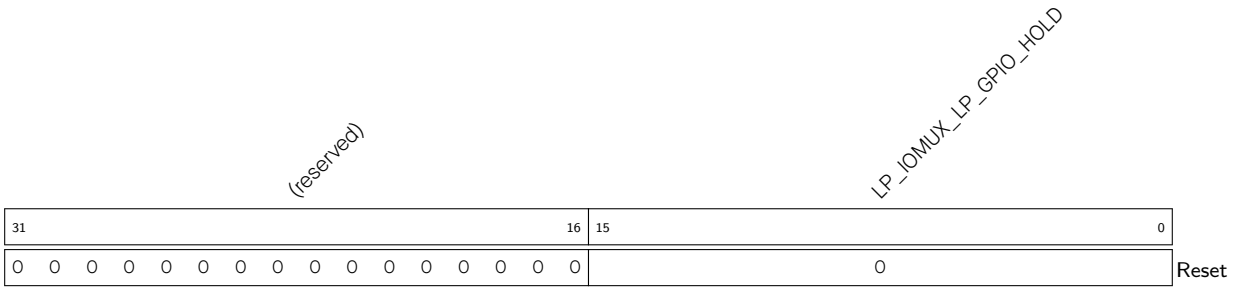
LP_IOMUX_PADn_FILTER_EN 配置是否使能管脚输入信号的滤波功能。

0：不使能

1：使能

(R/W)

Register 10.84. LP_IOMUX_LP_PAD_HOLD_REG (0x004C)



LP_IOMUX_LP_GPIO_HOLD 配置是否使能 GPIOn (n = 0 ~ 15) 的 Hold 功能。

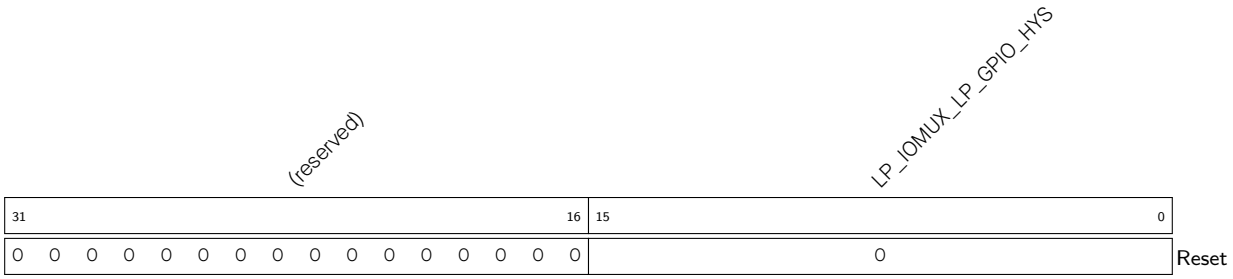
0：不使能

1：使能

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。

(R/W)

Register 10.85. LP_IOMUX_LP_PAD_HYS_REG (0x0050)



LP_IOMUX_LP_GPIO_HYS 配置是否使能 GPIO*n* (*n* = 0 ~ 15) 的迟滞功能。

0：不使能

1：使能

bit0 ~ bit15 分别对应 GPIO0 ~ GPIO15。

(R/W)

第 11 章

复位和时钟

11.1 复位

11.1.1 概述

ESP32-P4 提供四种级别的复位方式，分别是 CPU 复位 (CPU reset)、内核复位 (Core reset)、系统复位 (System reset) 和芯片复位 (Chip reset)。除芯片复位外，其他复位方式不影响片上内存存储的数据。图 11.1-1 展示了整个芯片系统的结构以及四种复位等级。

11.1.2 结构图

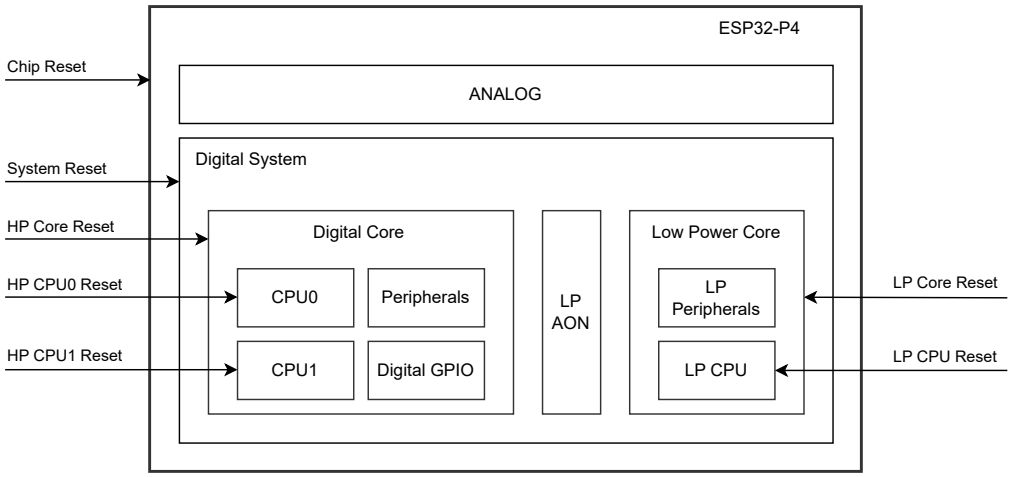


图 11.1-1. 四种复位等级

ESP32-P4 的数字系统分为两部分：高性能系统 (High Performance System, HP system) 和 低功耗系统 (Low Power System, LP system)。其中高性能系统包含数字内核 (Digital Core) 和 HP MEM，低功耗系统中包含低功耗常开系统（Low Power Always On，LP AON）、低功耗内核 (LP Core)、LP MEM。详细信息可参考图 11.1-1（HP MEM 和 LP MEM 不会复位，因此未在图中标注）。

11.1.3 特性

- 支持四种复位等级：
  - CPU 复位：复位 CPU 核，HP CPU0、HP CPU1、LP CPU 各有一套独立复位。其中：
    * HP CPU0 上电后会自动释放复位，从 CPU Reset Vector (0x30100000) 开始执行；
    * HP CPU1 上电后默认处于复位状态，需要向 HP_SYS_CLKRST_RST_EN_CORE1_GLOBAL 写 0 手动释放复位。复位释放后，程序将从 CPU Reset Vector (0x30100000) 开始执行；

* LP CPU 上电后处于复位状态，需要配置 PMU 释放。复位释放后，LP CPU 将从 LP_SYSTEM_LP_CPU_BOOT_ADDR（默认为 0x50100000）开始执行，详见章节 15 低功耗管理；

- 内核复位：包括 HP 以及 LP 的内核复位，复位除 LP AON 以外的其他数字系统。HP 内核复位包括 HP CPU0、HP CPU1、HP 外设、HP GPIO 等，LP 内核复位包括 LP CPU、LP 外设等；
- 系统复位：复位包括低功耗系统在内的整个数字系统；
- 芯片复位：复位整个芯片。

- 支持软件复位和硬件复位：
  - 软件复位：CPU 配置相关寄存器可触发软件复位，见章节 15 低功耗管理；
  - 硬件复位：硬件复位直接由硬件电路触发。

11.1.4 功能描述

上述任一复位发生时，CPU 将立刻被复位。复位释放后，可通过读取以下字段获取复位源：

- HP CPU0：LP_CLKRST_HPCORE0_RESET_CAUSE
- HP CPU1：LP_CLKRST_HPCORE1_RESET_CAUSE
- LP CPU：LP_CLKRST_LPCORE_RESET_CAUSE

表 11.1-1 和 11.1-2 列出了从上述寄存器中可能读出的复位源及其触发的复位等级。

表 11.1-1. HP CPU 复位源

编码	复位源	复位等级	说明
0x01	芯片复位[1]	芯片复位	—
0x03	系统软件复位	HP 内核复位	通过写 LP_SYSTEM_SYS_SW_RST 寄存器触发的系统复位
0x05	PMU 内核复位	HP 内核复位	详见章节 15 低功耗管理
0x07	MWDT 内核复位[2]	HP 内核复位	详见章节 18 看门狗定时器 (WDT)
0x09	RWDT 内核复位	HP 内核复位	详见章节 18 看门狗定时器 (WDT)
0x0B	MWDT CPU 复位	HP CPU0/1 复位	详见章节 18 看门狗定时器 (WDT)
0x0C	软件 CPU 复位	HP CPU0/1 复位	配置对应寄存器 LP_CLKRST_HPCORE0_SW_RESET 或 LP_CLKRST_HPCORE1_SW_RESET 触发
0x0D	RWDT CPU 复位	HP CPU0/1 复位	详见章节 18 看门狗定时器 (WDT)
0x0F	欠压系统复位	芯片复位或系统复位	欠压检测器触发的复位。欠压检测器在检测到欠压状态时，将根据寄存器配置，选择触发系统复位或者芯片复位，详见章节 15 低功耗管理
0x10	RWDT 系统复位	系统复位	详见章节 18 看门狗定时器 (WDT)
0x12	超级看门狗复位	系统复位	详见章节 18 看门狗定时器 (WDT)
0x13	电源毛刺复位	系统复位	详见章节 25 欠压检测器
0x1B	PSDET 复位	系统复位	详见章节 26 电源检测器 (PSDET)
0x14	eFuse 复位	HP 内核复位	eFuse CRC 校验错误触发复位

见下页

表 11.1-1 – 接上页

编码	复位源	复位等级	说明
0x16	USB (JTAG) 复位	HP 内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 JTAG 接口将触发此复位，详见章节 54 *USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)*
0x17	USB (UART) 复位	HP 内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 Serial 接口将触发此复位，详见章节 54 *USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)*
0x18	JTAG CPU 复位	HP CPU0/1 复位	接收“GDB 复位 CPU”指令时触发
0x1A	lockup 复位	HP CPU0/1 复位	CPU 发生 lockup 时复位。可通过配置寄存器 LP_CLKRST_HPCOREx_LOCKUP_RESET_EN 控制 HP CPU0、HP CPU1 能否被 lockup 复位

- 芯片复位的触发源包括以下两项：
  - 芯片上电
  - 欠压检测器
- HP 包含两个看门狗定时器，称作主系统看门狗定时器 (MWDT)，可通过 HP_SYS_CLKRST_HPCOREO_WDT_RESET_SOURCE_SEL 和 HP_SYS_CLKRST_HPCOREO_WDT_RESET_SOURCE_SEL 分别配置 HP CPU0 和 HP CPU1 可被哪个 MWDT 复位。

表 11.1-2. LP CPU 复位源

编码	复位源	复位等级	说明
0x01	芯片复位	芯片复位	—
0x09	PMU LP PERI 复位	LP 内核复位	详见章节 15 低功耗管理
0x0A	PMU LP CPU 复位[1]	LP CPU 复位	详见章节 15 低功耗管理
0x0F	欠压系统复位	芯片复位或系统复位	欠压检测器触发的复位
0x10	RWDT 系统复位	系统复位	详见章节 18 *看门狗定时器 (WDT)*
0x12	超级看门狗复位	系统复位	详见章节 18 *看门狗定时器 (WDT)*
0x13	电源毛刺复位	系统复位	详见章节 25 欠压检测器
0x14	软件 LP CPU 复位	LP CPU 复位	配置寄存器 LPPERI_RST_EN_LP_CORE 触发

[1] 芯片上电后，LP CPU 处于被 PMU 复位状态。为了在 LP CPU 第一次运行时能得到正确的复位源（即 0x01），PMU LP CPU 复位 (0x0A) 的复位源处于被屏蔽状态。在 LP CPU 第一次正常运行后，需要清零 LP_CLKRST_LPCORE_RESET_CAUSE_PMU_LP_CPU_MASK 寄存器以取消对 0x0A 复位源的屏蔽。

11.1.5 外设复位

外设可单独复位，也可通过配置内核复位、系统复位、芯片复位连带复位。

ESP32-P4 外设复位寄存器包括三个模块：HP_SYS_CLKRST、LP_CLKRST、LPPERI，可通过章节 11.4寄存器列表 查看各外设对应的复位寄存器。

11.2 时钟

11.2.1 概述

ESP32-P4 的时钟主要来源于振荡器（oscillator, OSC，包括 RC 振荡电路）、晶振 (XTAL) 和 PLL 时钟生成电路。上述时钟源产生的时钟经时钟分频器或时钟选择器等时钟模块的处理，使得大部分功能模块可以根据不同功耗和性能需求来获取及选择对应频率的工作时钟。图 11.2-1 为 HP 和 LP 系统时钟结构，图中包含主时钟通路和典型的外设时钟产生电路，由于 ESP32-P4 外设数量庞大，受限于篇幅，没有将所有外设的时钟呈现在图中。

11.2.2 结构图

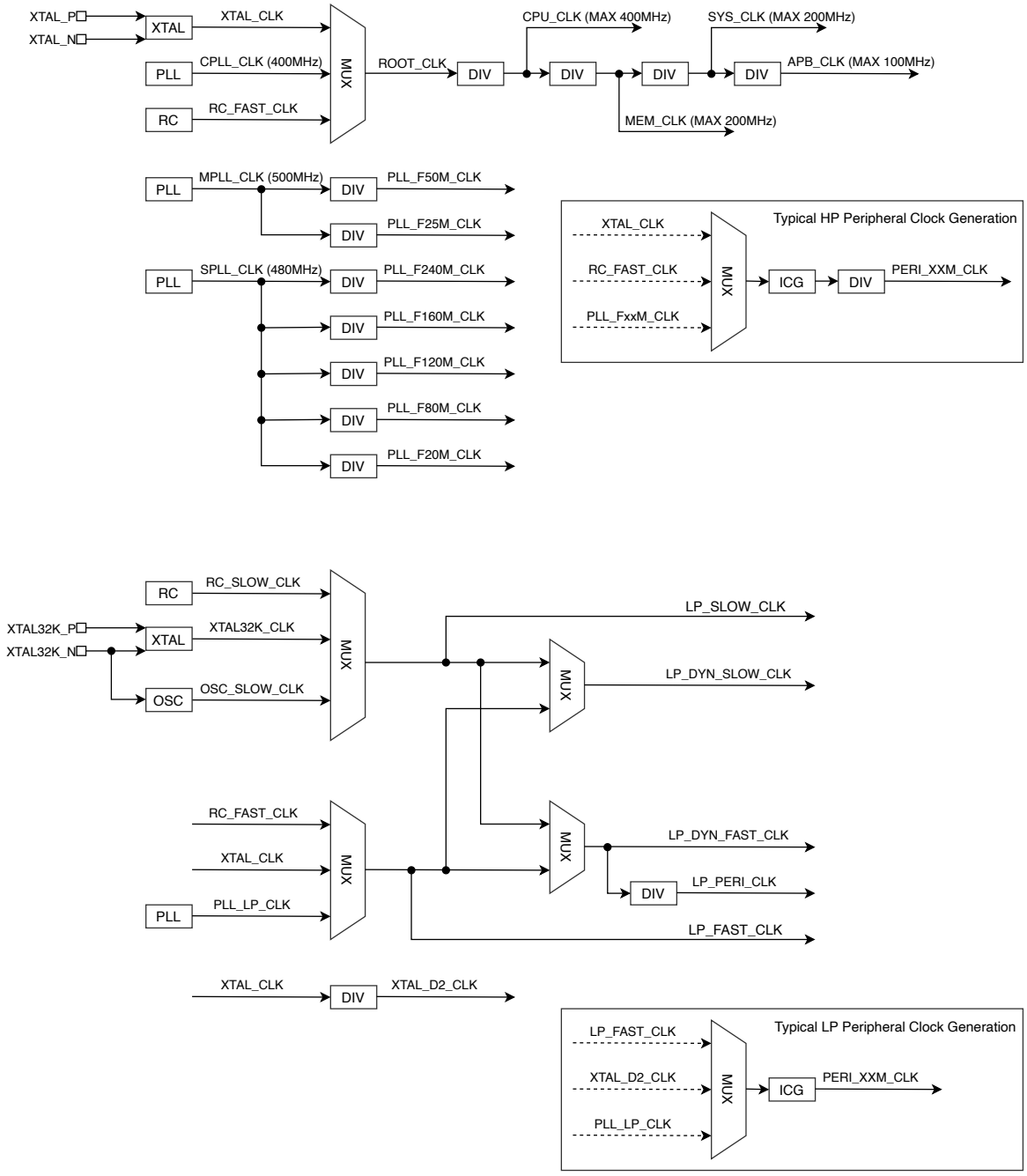


图 11.2-1. HP/LP 系统时钟

11.2.3 特性

ESP32-P4 的时钟根据频率不同，可分为：

- 高性能时钟，主要为 HP CPU0/1 和 HP 数字外设提供工作时钟
  - CPLL_CLK：360 MHz 内部 PLL 时钟（参考时钟是 XTAL_CLK）
  - MPLL_CLK：500 MHz 内部 PLL 时钟（参考时钟是 XTAL_CLK）
  - SPLL_CLK：480 MHz 内部 PLL 时钟（参考时钟是 XTAL_CLK）

- 低功耗时钟，主要为低功耗系统以及部分处于低功耗模式的外设提供工作时钟
  - XTAL32K_CLK：32 kHz 外部晶振时钟
  - RC_SLOW_CLK：150 kHz 内置慢速 RC 振荡器
  - OSC_SLOW_CLK：外置低速时钟，通常频率为 32 kHz，通过 XTAL_32K_N 输入，在配置好这个 GPIO 的状态后需要配置 Hold 功能，具体请参考章节 10 *GPIO 交换矩阵和 IO MUX* > 10.9 *GPIO 管脚的 Hold 特性*
  - XTAL_CLK：40 MHz 外部晶振时钟
  - RC_FAST_CLK：内置快速 RC 振荡器时钟，频率可调节（通常为 20 MHz）
  - PLL_LP_CLK：内部 PLL 时钟，通常为 8 MHz，参考时钟可选为 XTAL32K_CLK

11.2.4 功能描述

11.2.4.1 高性能系统时钟

如图 11.2-1 所示，XTAL_CLK、CPLL_CLK 和 RC_FAST_CLK 经过三选一逻辑产生 ROOT_CLK，ROOT_CLK 的时钟源选择请参考表 11.2-1。

表 11.2-1. ROOT_CLK 时钟源选择

LP_CLKRST_HP_ROOT_CLK_SRC_SEL	时钟源
0	XTAL_CLK
1	CPLL_CLK
2	RC_FAST_CLK

ROOT_CLK 通过一系列串行结构的分频器依次产生：

- CPU_CLK：驱动 HP CPU 及其相联逻辑
- MEM_CLK：驱动内部 memory（L2 Cache、L2MEM、ROM）及其相关逻辑
- SYS_CLK：HP 高速总线时钟，主要驱动 AXI/AHB 总线逻辑
- APB_CLK：HP 低速总线时钟，主要驱动 APB 总线逻辑

以上四类衍生时钟是 SoC 时钟，与章节 11.2.4.3 外设时钟 提到的相区别和对应（SoC 时钟均为同步时钟，SoC 时钟与任何外设时钟均为异步关系，外设时钟之间除非特别标注均互为异步关系）。

以上 SoC 时钟允许的最高频率如下，软件需要保证 SoC 时钟频率不超过最高频率：

- CPU_CLK：360 MHz
- MEM_CLK：200 MHz
- SYS_CLK：200 MHz
- APB_CLK：100 MHz

每个模块使用到的 SoC 时钟会受到一个独立的时钟门控单元控制，例如用户期望打开 HP CPU0 的时钟而关闭 HP CPU1 的时钟，可以置位寄存器 HP_SYS_CLKRST_COREO_CPU_CLK_EN，清零 HP_SYS_CLKRST_CORE1_CPU_CLK_EN 实现，其中寄存器域名字中的"CPU_CLK" 代表其属于图 11.2-1 中 CPU_CLK 这一时钟组。

ESP32-P4 的内部 PLL 时钟 MPLL_CLK 和 SPLL_CLK 通过一组并行分频器产生供外设使用的参考时钟。

除了上述两个 PLL 外，还有两个特定模块专用的 PLL：

- SDIO_PLL，输出三个时钟：
  - SDIO_PLL0_CLK，用于 SDIO 的高速 SDIO_SLF_CLK 时钟（内部信号时钟），也能作为 SDIO 低速时钟的源
  - SDIO_PLL1_CLK 用于 SDIO 的高速 SDIO_DRV_CLK 时钟（输出信号驱动时钟）
  - SDIO_PLL2_CLK 用于 SDIO 的高速 SDIO_SAM_CLK 时钟（输入信号采样时钟）
- APLL_CLK，I2S 的专用时钟

一个典型的外设时钟产生电路由时钟选择器、时钟门控单元和分频器组成。在高性能模式下，通过选择高速时钟源以及不对时钟分频来获取更高处理速度；在低功耗模式下，可通过选择低速时钟源或使用高倍率分频系数降低功耗；在外设关断情况下，可以直接关断时钟。

说明：
任何对 HP_SYS_CLKRST_ROOT_CLK_CTRL0/1/2/3_REG 寄存器的修改，都必须置位 HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE 字段后才能真正生效。

11.2.4.2 低功耗系统时钟

低功耗系统能够在大多数时钟源关闭的状态下工作。低功耗系统时钟包括 LP_SLOW_CLK 时钟、LP_FAST_CLK、LP_DYN_SLOW_CLK、LP_DYN_FAST_CLK、XTAL_D2_CLK 时钟。

LP_SLOW_CLK 和 LP_FAST_CLK 的时钟源为低频时钟，其中：

- LP_SLOW_CLK 时钟有四种可能的时钟源，见表 11.2-2：

表 11.2-2. LP_SLOW_CLK 时钟源选择

LP_CLKRST_SLOW_CLK_SEL	时钟源
0	RC_SLOW_CLK
1	XTAL32K_CLK
2	无效值
3	OSC_SLOW_CLK

- LP_FAST_CLK 时钟有三种可能的时钟源，见表 11.2-3：

表 11.2-3. LP_FAST_CLK 时钟源选择

LP_CLKRST_FAST_CLK_SEL	时钟源
0	RC_FAST_CLK
1	XTAL_CLK
2	PLL_LP_CLK

LP_DYN_SLOW_CLK 和 LP_DYN_FAST_CLK 时钟源均为 LP_SLOW_CLK 或 LP_FAST_CLK，根据芯片电源的模式（详见章节 15 低功耗管理）选择时钟源，保证在各种电源模式下这两个时钟均能正常工作，用于驱动 LP AON 的模块。LP_DYN_SLOW_CLK 和 LP_DYN_FAST_CLK 始终保持同步关系：

- 芯片电源为 Active 模式时始终选择 LP_FAST_CLK 作为时钟源，但此时 LP_DYN_SLOW_CLK 频率和 LP_SLOW_CLK 相同，相位和 LP_FAST_CLK 一致；
- 芯片电源为 Deep-sleep 模式时根据 PMU 配置选择时钟。若 LP CPU 关闭，则选择 LP_SLOW_CLK 作为时钟源，此时 LP_DYN_FAST_CLK 频率、相位和 LP_SLOW_CLK 相同；若 LP CPU 使能，则选择 LP_FAST_CLK 为时钟源。

LP_PERI_CLK 经过 LP_DYN_FAST_CLK 分频得到，驱动所有 LP 外设总线。

XTAL_D2_CLK 为 XTAL_CLK 的二分频。

11.2.4.3 外设时钟

表 11.2-4、表 11.2-5、表 11.2-6、和表 11.2-7 分别列出了接入各个外设时钟的衍生高性能/低功耗时钟源、接入高性能系统各个外设的时钟以及接入低功耗系统各个外设的时钟。

表 11.2-4. 衍生高性能时钟源

衍生时钟	源时钟							衍生时钟						
	RC_FAST_CLK	XTAL_CLK	CPLL_CLK	MPLL_CLK	SPLL_CLK	APLL_CLK	SDIO_PLL0_CLK	ROOT_CLK	CPU_CLK	MEM_CLK	SYS_CLK	PLL_F80M_CLK	PLL_F160M_CLK	PLL_F240M_CLK
ROOT_CLK	Y	Y	Y											
CPU_CLK								Y						
MEM_CLK									Y					
SYS_CLK										Y				
APB_CLK											Y			
PLL_F50M_CLK				Y										
PLL_F25M_CLK				Y										
PLL_F240M_CLK					Y									
PLL_F160M_CLK					Y									
PLL_F120M_CLK					Y									
PLL_F80M_CLK					Y									
PLL_F20M_CLK					Y									
PLL_F12M_CLK					Y									
PLL_F48M_CLK					Y									
FLASH_CLK		Y	Y		Y									
PSRAM_CLK		Y	Y	Y	Y									
GPSPI_CLK	Y	Y			Y	Y	Y							
ADC_CLK	Y	Y										Y		
UART_CLK	Y	Y										Y		
CRYPTO_CLK	Y	Y											Y	Y

表 11.2-5. 高性能外设时钟

外设	时钟	时钟源														衍生时钟						Clock from IO
		RC_FAST_CLK	XTAL_CLK	APLL_CLK	CPLL_CLK	SPLL_CLK	MPLL_CLK	SDIO_PLL0_CLK	PLL_F12M_CLK	PLL_F20M_CLK	PLL_F25M_CLK	PLL_F48M_CLK	PLL_F80M_CLK	PLL_F120M_CLK	PLL_F160M_CLK	PLL_F240M_CLK	FLASH_CLK	PSRAM_CLK	GPSPI_CLK	ADC_CLK	UART_CLK	CRYPTO_CLK	
SPI0/1 (flash)	FLASH_PLL_CLK																Y						
	FLASH_CORE_CLK																Y						
SPI0/1 (PSRAM)	PSRAM_PLL_CLK																	Y					
	PSRAM_CORE_CLK																	Y					
I2S	I2S_RX_CLK		Y	Y											Y								PAD_I2S_MCLK
	I2S_TX_CLK		Y	Y											Y								PAD_I2S_MCLK
I2C	I2C_CLK	Y	Y																				
MCPWM	MCPWM_CLK	Y	Y												Y								
TIMG	TIMERGRP_T0_CLK	Y	Y										Y										
	TIMERGRP_T1_CLK	Y	Y										Y										
MWDT	TIMERGRP_WDT_CLK	Y	Y										Y										
TWAI	TWAI_CLK	Y	Y																				
GP-SPI2/GP-SPI3	GPSPI_HS_CLK																		Y				
	GPSPI_MST_CLK																		Y				
EMAC	EMAC_RMII_CLK																						PAD_EMAC_TXRX_CLK PAD_EMAC_RX_CLK PAD_EMAC_TX_CLK
	EMAC_RX_CLK																						PAD_EMAC_TXRX_CLK PAD_EMAC_RX_CLK
	EMAC_TX_CLK																						PAD_EMAC_TXRX_CLK PAD_EMAC_TX_CLK
	EMAC_PTP_REF_CLK		Y										Y										
CAM OUT	CAM_CLK		Y	Y											Y								
IO MUX	IOMUX_CLK		Y										Y										
LCD	LCD_CLK		Y	Y											Y								

见下页

表 11.2-5 – 接上页

外设	时钟	时钟源															衍生时钟						Clock from IO
		RC_FAST_CLK	XTAL_CLK	APLL_CLK	CPLL_CLK	SPLL_CLK	MPLL_CLK	SDIO_PLL0_CLK	PLL_F12M_CLK	PLL_F20M_CLK	PLL_F25M_CLK	PLL_F48M_CLK	PLL_F80M_CLK	PLL_F120M_CLK	PLL_F160M_CLK	PLL_F240M_CLK	FLASH_CLK	PSRAM_CLK	GPSPI_CLK	ADC_CLK	UART_CLK	CRYPTO_CLK	
LEDC	LEDC_CLK	Y	Y										Y										
ADC	HPADC_CLK																			Y			
	HPADC1_SARCLK																			Y			
	HPADC2_SARCLK																			Y			
AES SHA RSA ECC RSA_DS HMAC Key Manager ECDSA_DS	AES_CLK SHA_CLK RSA_CLK ECC_CLK DSA_CLK HMAC_CLK KM_CLK ECDSA_DS_CLK SEC_CLK																					Y	
RMT	RMT_CLK	Y	Y										Y										
System Timer	SYSTIMER_CLK	Y	Y																				
UART	UART_FCLK																				Y		
	UART_SCLK																				Y		
	UART_SLP_CLK																						PAD_UART_SLP_CLK
PARLIO	PARLIO_RX_CLK	Y	Y												Y								PAD_PARLIO_RX_CLK
	PARLIO_TX_CLK	Y	Y												Y								PAD_PARLIO_TX_CLK
I3S	I3C_MST_CLK		Y											Y	Y								
ISP	ISP_CLK		Y												Y	Y							
MIPI DSI	DSI_DPHY_CFG_CLK	Y								Y	Y												
	DSI_DPHY_PLL_REFCLK		Y	Y	Y	Y	Y																
	DSI_DPICLK		Y	Y											Y	Y							
MIPI CSI	CSI_DPHY_CFG_CLK	Y								Y	Y												
H264	H264_CLK		Y													Y							
USB_SERIAL_JTAG	USB_SERIAL_JTAG_CLK											Y											
USB 2.0 全速 OTG	USB_OTG11_CLK											Y											
USB 2.0 高速 OTG	USB_HS_PHY_REFCLK								Y	Y													PAD_USB_HSPHY_REFCLK
SD/ MMC 主机控制器（SD-HOST）	SDIO_DRV_CLK (low-speed) SDIO_SLF_CLK (low-speed) SDIO_SAM_CLK (low-speed)							Y							Y								

表 11.2-6. 衍生低功耗时钟源

衍生时钟	源时钟						衍生时钟					Clock from IO
	RC_SLOW_CLK 150 kHz	XTAL32K_CLK 32 kHz	OSC_SLOW_CLK 32 kHz	RC_FAST_CLK 20 MHz	XTAL_CLK 40 MHz	PLL_LP_CLK 8 MHz	LP_SLOW_CLK	LP_FAST_CLK	LP_DYN_SLOW_CLK	LP_DYN_FAST_CLK	XTAL_D2_CLK	
LP_FAST_CLK				Y	Y	Y						
LP_SLOW_CLK	Y	Y	Y									
LP_DYN_FAST_CLK							Y	Y				
LP_DYN_SLOW_CLK							Y	Y				
LP_PERI_CLK										Y		
XTAL_D2_CLK					Y							

表 11.2-7. 低功耗系统外设时钟

衍生时钟	源时钟						衍生时钟						Clock from IO
	RC_SLOW_CLK 150 kHz	XTAL32K_CLK 32 kHz	OSC_SLOW_CLK 32 kHz	RC_FAST_CLK 20 MHz	XTAL_CLK 40 MHz	PLL_LP_CLK 8 MHz	LP_SLOW_CLK	LP_FAST_CLK	LP_DYN_SLOW_CLK	LP_DYN_FAST_CLK	LP_PERI_CLK	XTAL_D2_CLK	
eFuse 控制器 (EFUSE)												Y	
RTC 看门狗定时器 (RWDT)	Y								Y	Y			
实时时钟定时器									Y	Y			
欠压检测器										Y			
电源管理单元 (PMU)	Y							Y	Y	Y			
LP UART						Y				Y		Y	PAD_LP_UART_SLP_CK
LP SPI													PAD_LP_SPI_CK
LP I2C						Y				Y		Y	
模拟 I2C 控制器								Y					
LP I2S 控制器						Y				Y		Y	
LP ADC 控制器											Y		

说明：
为了防止 DPA (Differential Power Analysis) 攻击加解密外设，加解密功能时钟采用随机分频策略。根据随机分频范围划分为三个安全等级。可通过配置 HP_SYS_CLKRST_SEC_DPA_CFG_SEL 选择安全等级。
当 HP_SYS_CLKRST_SEC_DPA_CFG_SEL 为 1 时，安全等级由 HP_SYS_CLKRST_SEC_DPA_LEVEL 决定，否则由 EFUSE_SEC_DPA_LEVEL 决定。

11.3 配置流程

11.3.1 高性能系统时钟配置

- 可配置 HP_SYS_CLKRST_CPU_CLK_DIV_NUM（分频系数整数部分）、HP_SYS_CLKRST_CPU_CLK_DIV_NUMERATOR（分频系数小数部分分子）、HP_SYS_CLKRST_CPU_CLK_DIV_DENOMINATOR（分频系数小数部分分母）控制 CPU_CLK 的分频；向 HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE 写 1 使 CPU_CLK 配置的分频系数生效。
- 可配置 HP_SYS_CLKRST_MEM_CLK_DIV_NUM（分频系数整数部分）、HP_SYS_CLKRST_MEM_CLK_DIV_NUMERATOR（分频系数小数部分分子）、HP_SYS_CLKRST_MEM_CLK_DIV_DENOMINATOR（分频系数小数部分分母）控制 MEM_CLK 的分频；向 HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE 写 1 使 MEM_CLK 配置的分频系数生效。
- 可配置 HP_SYS_CLKRST_SYS_CLK_DIV_NUM（分频系数整数部分）、HP_SYS_CLKRST_SYS_CLK_DIV_NUMERATOR（分频系数小数部分分子）、HP_SYS_CLKRST_SYS_CLK_DIV_DENOMINATOR（分频系数小数部分分母）控制 SYS_CLK 的分频；向 HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE 写 1 使 SYS_CLK 配置的分频系数生效。
- 可配置 HP_SYS_CLKRST_APB_CLK_DIV_NUM（分频系数整数部分）、HP_SYS_CLKRST_APB_CLK_DIV_NUMERATOR（分频系数小数部分分子）、HP_SYS_CLKRST_APB_CLK_DIV_DENOMINATOR（分频系数小数部分分母）控制 APB_CLK 的分频；向 HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE 写 1 使 APB_CLK 配置的分频系数生效。
- 可配置 LP_CLKRST_HP_ROOT_CLK_SRC_SEL 选择 HP 的 ROOT_CLK 时钟源。

注意：
1. 推荐在上述 SoC 时钟分频配置完成并更新配置之后再进行 ROOT_CLK 时钟源选择配置。
2. 切换 ROOT_CLK 时钟源前软件需要保证使用了正确的 SoC 时钟分频，防止各时钟超过允许的最大频率。
3. 向 HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE 写 1 会同时使 CPU_CLK、MEM_CLK、SYS_CLK、APB_CLK 配置的分频系数生效，但不推荐一次性生效所有分配系数。建议每配置一个模块生效一次。
4. 在高分频系数向低分频系数切换过程中，推荐按照 APB_CLK → SYS_CLK → MEM_CLK → CPU_CLK 的顺序配置；在低分频系数向高分频系数切换过程中，推荐按照 CPU_CLK → MEM_CLK → SYS_CLK → APB_CLK 的顺序配置；以防止出现高频时钟毛刺。

11.3.2 低功耗系统时钟配置

- 可配置 LP_CLKRST_SLOW_CLK_SEL 选择 LP_SLOW_CLK 的时钟源。
- 可配置 LP_CLKRST_FAST_CLK_SEL 选择 LP_FAST_CLK 的时钟源。

- 可配置 LP_CLKRST_LP_PERI_DIV_NUM 设置 LP_PERI_CLK 的时钟分频系数。

11.3.3 外设时钟复位配置

注意：

ESP32-P4 具有低功耗特性，因此有些外设时钟默认为关闭状态。在启用这些外设之前，必须将外设的时钟打开，即把相应的 CLK_EN 位置为 1，并且解除外设的复位状态，即把相应的 RST_EN 位置为 0。

大部分外设模块的时钟都可以分为总线时钟和功能时钟。外设模块的总线时钟主要是用于配置外设模块的寄存器。功能时钟主要是模块工作所使用的时钟，例如 UART 的参考时钟。大部分模块的功能时钟可以选择多个时钟源。在门控寄存器的描述中会说明该寄存器属于总线（SYS_CLK、APB_CLK）时钟门控寄存器还是功能时钟门控寄存器。

ESP32-P4 的总线时钟开关、功能时钟的开关以及时钟源选择和时钟分频的配置寄存器均放在了 HP_SYS_CLKRST、LP_CLKRST、LPPERI 中。其中 HP 的外设，除了 EMAC、SDMMC、USB Serial/JTAG、USB 2.0 全速 OTG、USB 2.0 高速 OTG 的复位配置寄存器在 LP_CLKRST 外，其余所有均在 HP_SYS_CLKRST。更多信息，详见章节 11.4 寄存器列表。

修改时钟配置时，推荐先把对应门控关闭，修改好分频、时钟源选择等配置后，再重新开。

功能模块不工作时，可以配置对应寄存器关闭功能时钟，关闭模块功能时钟不会影响系统其他部分。下文以 I2C 时钟配置为例：

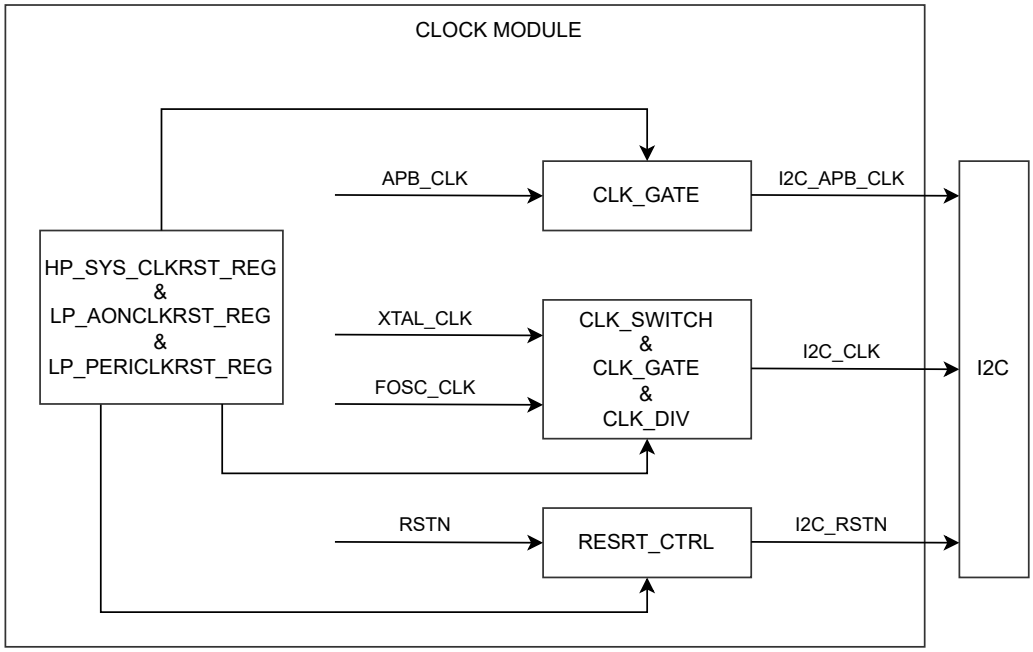


图 11.3-1. 时钟配置示例

图 11.3-1 是 I2C 的时钟结构图，其他模块的时钟结构类似。其中 CLK_SWITCH 的功能是选择一个时钟输出，CLK_GATE 的功能是打开/关闭时钟。

对于要求低功耗的场景，当模块不工作时，除了关闭功能时钟，还可以关闭模块的总线时钟来进一步降低功耗。注意，如果先关闭模块总线时钟，那么模块的功能时钟还是有可能继续工作，所以推荐先关闭功能时钟再关闭总线时钟，打开时钟时推荐先打开总线时钟再打开功能时钟。

说明：

本章节中所有分频寄存器配置的值为实际分频值减去 1。

11.4 寄存器列表

11.4.1 复位与时钟 (HP_SYS_CLKRST) 寄存器列表

本小节所有地址均为相对于 HP_SYS_CLKRST 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时钟门使能寄存器			
HP_SYS_CLKRST_CLK_EN0_REG	寄存器时钟门使能寄存器	0x0000	R/W
HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG	系统和外设时钟门配置寄存器	0x00B4	R/W
根时钟配置寄存器			
HP_SYS_CLKRST_ROOT_CLK_CTRL0_REG	根时钟配置寄存器 0	0x0004	变化
HP_SYS_CLKRST_ROOT_CLK_CTRL1_REG	根时钟配置寄存器 1	0x0008	R/W
HP_SYS_CLKRST_ROOT_CLK_CTRL2_REG	根时钟配置寄存器 2	0x000C	R/W
HP_SYS_CLKRST_ROOT_CLK_CTRL3_REG	根时钟配置寄存器 3	0x0010	R/W
总线时钟配置寄存器			
HP_SYS_CLKRST_SOC_CLK_CTRL0_REG	总线时钟配置寄存器 0	0x0014	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL1_REG	总线时钟配置寄存器 1	0x0018	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL2_REG	总线时钟配置寄存器 2	0x001C	R/W
HP_SYS_CLKRST_SOC_CLK_CTRL3_REG	总线时钟配置寄存器 3	0x0020	R/W
参考时钟配置寄存器			
HP_SYS_CLKRST_REF_CLK_CTRL0_REG	参考时钟配置寄存器 0	0x0024	R/W
HP_SYS_CLKRST_REF_CLK_CTRL1_REG	参考时钟配置寄存器 1	0x0028	R/W
HP_SYS_CLKRST_REF_CLK_CTRL2_REG	参考时钟配置寄存器 2	0x002C	R/W
外设时钟配置寄存器			
HP_SYS_CLKRST_PERI_CLK_CTRL00_REG	外设时钟配置寄存器	0x0030	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL01_REG	外设时钟配置寄存器	0x0034	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL02_REG	外设时钟配置寄存器	0x0038	变化
HP_SYS_CLKRST_PERI_CLK_CTRL03_REG	外设时钟配置寄存器	0x003C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL10_REG	外设时钟配置寄存器	0x0040	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL11_REG	外设时钟配置寄存器	0x0044	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL12_REG	外设时钟配置寄存器	0x0048	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL13_REG	外设时钟配置寄存器	0x004C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL14_REG	外设时钟配置寄存器	0x0050	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL15_REG	外设时钟配置寄存器	0x0054	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL16_REG	外设时钟配置寄存器	0x0058	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL17_REG	外设时钟配置寄存器	0x005C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL18_REG	外设时钟配置寄存器	0x0060	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL19_REG	外设时钟配置寄存器	0x0064	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL110_REG	外设时钟配置寄存器	0x0068	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL111_REG	外设时钟配置寄存器	0x006C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL112_REG	外设时钟配置寄存器	0x0070	R/W

名称	描述	地址	访问
HP_SYS_CLKRST_PERI_CLK_CTRL113_REG	外设时钟配置寄存器	0x0074	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL114_REG	外设时钟配置寄存器	0x0078	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL115_REG	外设时钟配置寄存器	0x007C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL116_REG	外设时钟配置寄存器	0x0080	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL117_REG	外设时钟配置寄存器	0x0084	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL118_REG	外设时钟配置寄存器	0x0088	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL119_REG	外设时钟配置寄存器	0x008C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL120_REG	外设时钟配置寄存器	0x0090	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL20_REG	外设时钟配置寄存器	0x0094	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL21_REG	外设时钟配置寄存器	0x0098	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL22_REG	外设时钟配置寄存器	0x009C	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL23_REG	外设时钟配置寄存器	0x00A0	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL24_REG	外设时钟配置寄存器	0x00A4	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL25_REG	外设时钟配置寄存器	0x00A8	R/W
HP_SYS_CLKRST_PERI_CLK_CTRL26_REG	外设时钟配置寄存器	0x00AC	R/W
DPA 时钟配置寄存器			
HP_SYS_CLKRST_DPA_CTRL0_REG	DPA 时钟配置寄存器	0x00B8	R/W
模拟 PLL 校准配置寄存器			
HP_SYS_CLKRST_ANA_PLL_CTRL0_REG	模拟 PLL 校准配置寄存器	0x00BC	变化
外设复位寄存器			
HP_SYS_CLKRST_HP_RST_EN0_REG	外设复位寄存器 0	0x00C0	R/W
HP_SYS_CLKRST_HP_RST_EN1_REG	外设复位寄存器 1	0x00C4	R/W
HP_SYS_CLKRST_HP_RST_EN2_REG	外设复位寄存器 2	0x00C8	R/W
HP_SYS_CLKRST_HP_FORCE_NORST0_REG	外设复位禁用寄存器 0	0x00CC	R/W
HP_SYS_CLKRST_HP_FORCE_NORST1_REG	外设复位禁用寄存器 1	0x00D0	R/W
HP CPU WDT 复位配置寄存器			
HP_SYS_CLKRST_HPWDT_CORE0_RST_CTRL0_REG	HP CPU0 WDT 复位配置寄存器	0x00D4	R/W
HP_SYS_CLKRST_HPWDT_CORE1_RST_CTRL0_REG	HP CPU1 WDT 复位配置寄存器	0x00D8	R/W
HP_SYS_CLKRST_HPCORE_WDT_RESET_SOURCE0_REG	HP CPU WDT 复位源配置寄存器	0x00EC	R/W
CPU 时钟状态寄存器			
HP_SYS_CLKRST_CPU_SRC_FREQ0_REG	CPU 源时钟频率寄存器	0x00DC	RO
HP_SYS_CLKRST_CPU_CLK_STATUS0_REG	CPU 时钟状态寄存器	0x00E0	RO
WAITI 信号控制寄存器			
HP_SYS_CLKRST_CPU_WAITI_CTRL0_REG	WAITI 信号控制寄存器	0x00F4	R/W

11.4.2 LP 常开时钟与复位 (LP_CLKRST) 寄存器列表

本小节地址均为相对于 LP_CLKRST 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2

名称	描述	地址	访问
配置寄存器			
LP_CLKRST_LP_CLK_CONF_REG	LP 根时钟配置寄存器	0x0000	R/W
LP_CLKRST_LP_CLK_EN_REG	LP 常开外设时钟配置寄存器	0x0008	R/W
LP_CLKRST_LP_RST_EN_REG	LP 常开外设复位配置寄存器	0x000C	R/W
LP_CLKRST_RESET_CAUSE_REG	HP CPU 和 LP CPU 复位原因寄存器	0x0010	可变
LP_CLKRST_HPCPU_RESET_CTRL0_REG	HP CPU 复位配置寄存器 0	0x0014	可变
LP_CLKRST_HPCPU_RESET_CTRL1_REG	HP CPU 复位配置寄存器 1	0x0018	R/W
LP_CLKRST_FOSC_CNTL_REG	RC_FAST_CLK 频率配置寄存器	0x001C	R/W
LP_CLKRST_XTAL32K_REG	XTAL32K_CLK 配置寄存器	0x0030	R/W
LP_CLKRST_HP_CLK_CTRL_REG	HP 时钟控制寄存器	0x0040	R/W
LP_CLKRST_HP_USB_CLKRST_CTRL0_REG	USB 时钟和复位控制寄存器 0	0x0044	R/W
LP_CLKRST_HP_USB_CLKRST_CTRL1_REG	USB 时钟和复位控制寄存器 1	0x0048	R/W
LP_CLKRST_HP_SDMMC_EMAC_RST_CTRL_REG	SDMMC 和 EMAC 复位控制寄存器	0x004C	R/W
版本寄存器			
LP_CLKRST_DATE_REG	版本控制寄存器	0x03FC	R/W

11.4.3 LP 外设时钟与复位 (LPPERI) 寄存器列表

本小节地址均为相对于 LPPERI 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2

名称	描述	地址	访问权限
配置寄存器			
LPPERI_CLK_EN_REG	低功耗外设总线时钟配置寄存器	0x0000	R/W
LPPERI_CORE_CLK_SEL_REG	低功耗外设功能时钟配置寄存器	0x0004	R/W
LPPERI_RESET_EN_REG	低功耗外设复位配置寄存器	0x0008	varies
LPPERI_CPU_REG	低功耗 CPU 访问配置寄存器	0x000C	R/W
LPPERI_MEM_CTRL_REG	低功耗 UART 杂项控制寄存器	0x0028	varies
LPPERI_ADC_CTRL_REG	低功耗 ADC 时钟配置寄存器	0x002C	R/W
LPPERI_LP_I2S_RXCLK_DIV_NUM_REG	低功耗 I2S 接收时钟配置寄存器	0x0030	R/W
LPPERI_LP_I2S_RXCLK_DIV_XYZ_REG	低功耗 I2S 接收时钟配置寄存器	0x0034	R/W
版本寄存器			
LPPERI_DATE_REG	版本控制寄存器	0x03FC	R/W

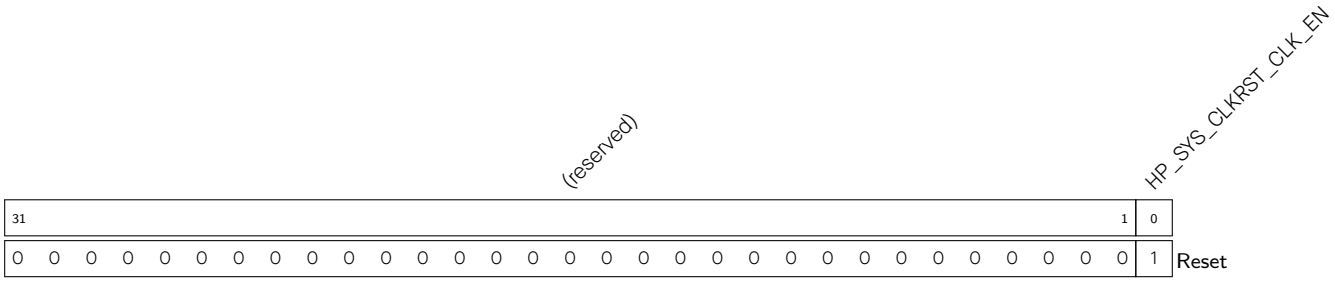
11.5 寄存器

11.5.1 复位与时钟 (HP_SYS_CLKRST) 寄存器

本小节的所有地址均为相对于 HP_SYS_CLKRST 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 11.1. HP_SYS_CLKRST_CLK_EN0_REG (0x0000)



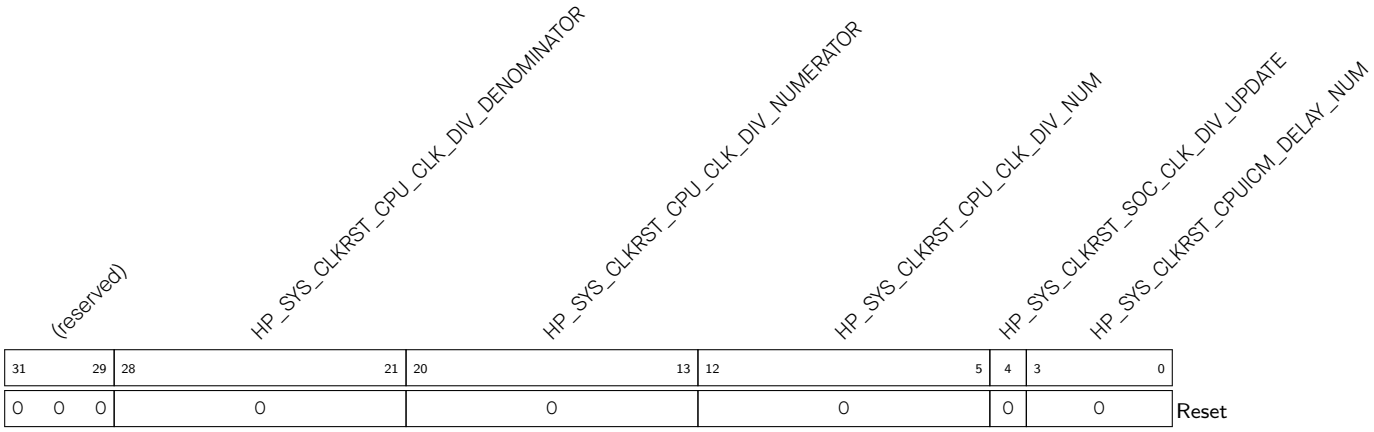
HP_SYS_CLKRST_CLK_EN 配置寄存器时钟门控。

0：仅在应用写寄存器时开启时钟

1：强制开启配置寄存器时钟

(R/W)

Register 11.2. HP_SYS_CLKRST_ROOT_CLK_CTRL0_REG (0x0004)



HP_SYS_CLKRST_CPUICM_DELAY_NUM 配置进入 WFI 模式至 CPU 时钟真正关断所需的时间。单位：时钟周期。(R/W)

HP_SYS_CLKRST_SOC_CLK_DIV_UPDATE 配置更新 CPU_CLK、MEM_CLK、SYS_CLK、APB_CLK 的分频系数。

0：不更新

1：更新

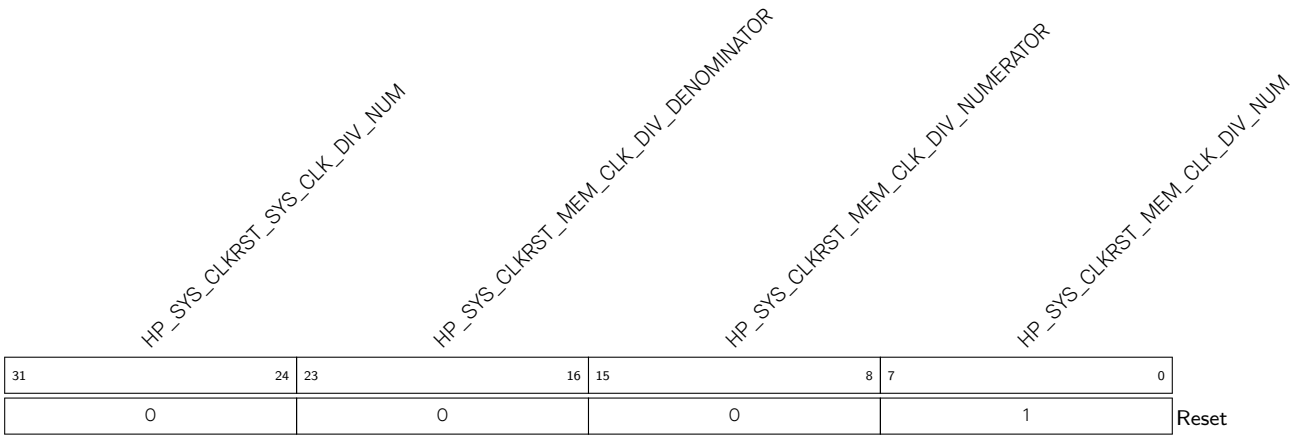
(WT)

HP_SYS_CLKRST_CPU_CLK_DIV_NUM 配置 CPU_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_CPU_CLK_DIV_NUMERATOR 配置 CPU_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_CPU_CLK_DIV_DENOMINATOR 配置 CPU_CLK 时钟分频系数小数部分的分母。(R/W)

Register 11.3. HP_SYS_CLKRST_ROOT_CLK_CTRL1_REG (0x0008)



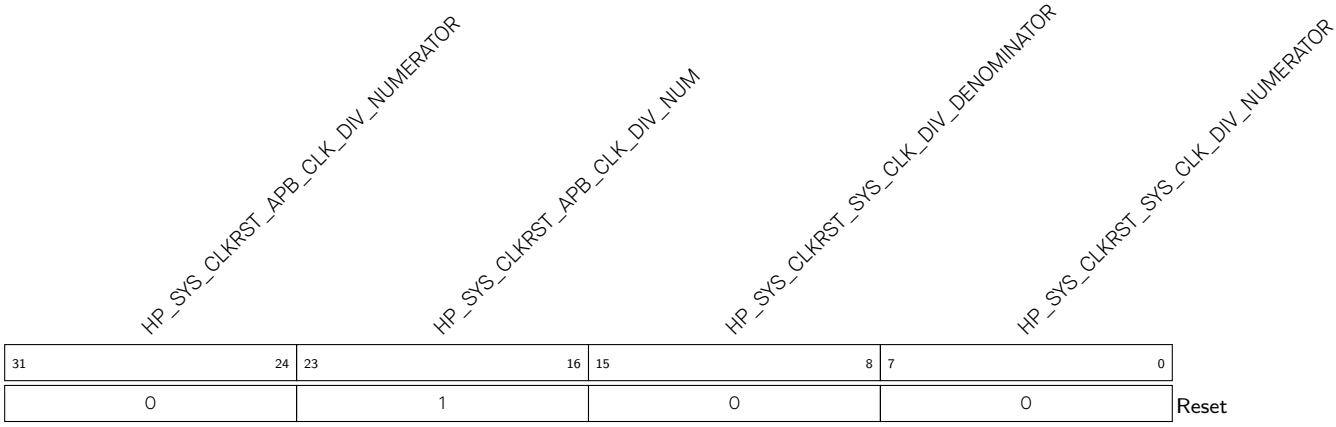
HP_SYS_CLKRST_MEM_CLK_DIV_NUM 配置 MEM_CLK 分频系数的整数部分。(R/W)

HP_SYS_CLKRST_MEM_CLK_DIV_NUMERATOR 配置 MEM_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_MEM_CLK_DIV_DENOMINATOR 配置 MEM_CLK 分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_SYS_CLK_DIV_NUM 配置 SYS_CLK 分频系数的整数部分。(R/W)

Register 11.4. HP_SYS_CLKRST_ROOT_CLK_CTRL2_REG (0x000C)



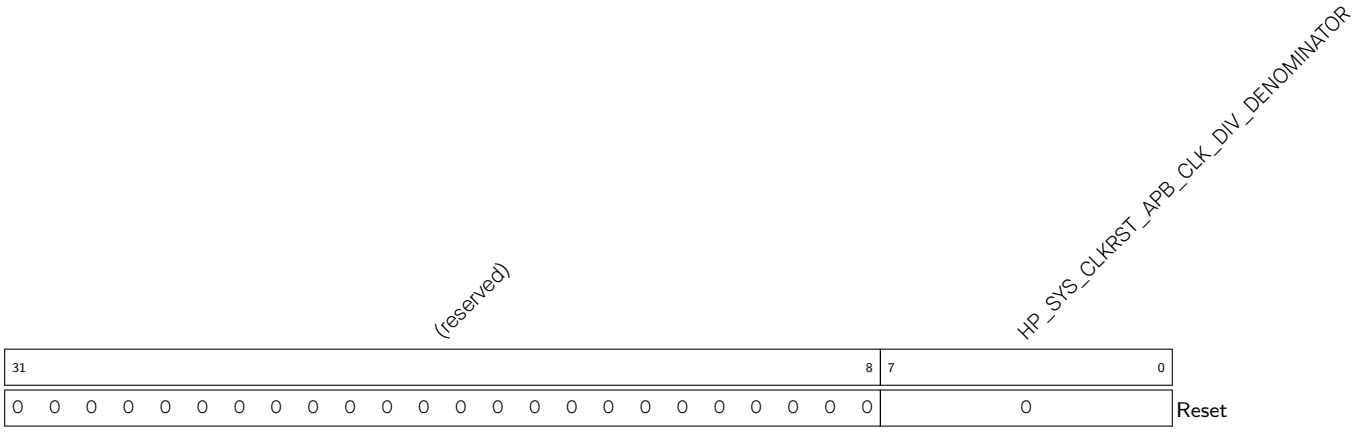
HP_SYS_CLKRST_SYS_CLK_DIV_NUMERATOR 配置 SYS_CLK 分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_SYS_CLK_DIV_DENOMINATOR 配置 SYS_CLK 分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_APB_CLK_DIV_NUM 配置 APB_CLK 分频系数的整数部分。(R/W)

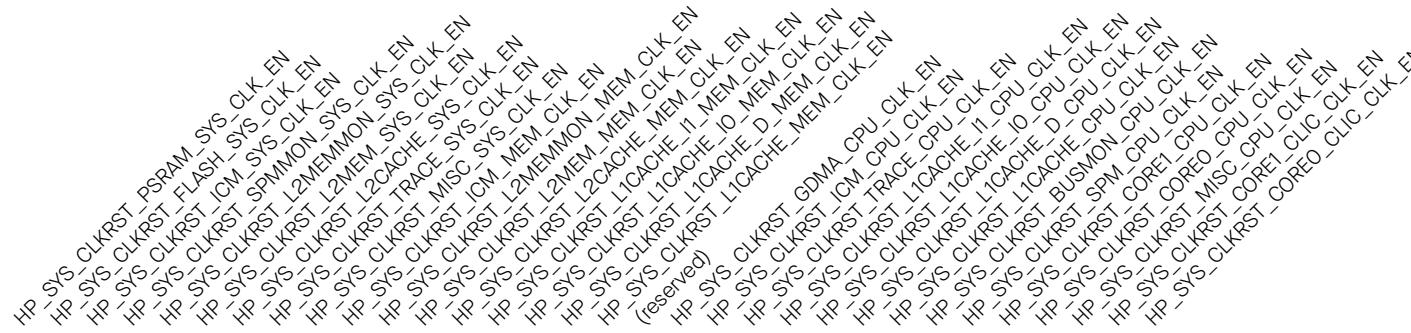
HP_SYS_CLKRST_APB_CLK_DIV_NUMERATOR 配置 APB_CLK 分频系数小数部分的分子。(R/W)

Register 11.5. HP_SYS_CLKRST_ROOT_CLK_CTRL3_REG (0x0010)



HP_SYS_CLKRST_APB_CLK_DIV_DENOMINATOR 配置 APB_CLK 分频系数小数部分的分母。(R/W)

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)



HP_SYS_CLKRST_CORE0_CLIC_CLK_EN 配置是否使能 HP CPU0 CLIC 的总线时钟。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CORE1_CLIC_CLK_EN 配置是否使能 HP CPU1 CLIC 的总线时钟。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_MISC_CPU_CLK_EN 配置是否使能 MISC（杂项）的总线时钟（CPU_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CORE0_CPU_CLK_EN 配置是否使能 HP CPU0 的总线时钟（CPU_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CORE1_CPU_CLK_EN 配置是否使能 HP CPU1 的总线时钟（CPU_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_SPM_CPU_CLK_EN 配置是否使能 SPM 监控的总线时钟（CPU_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

接上页...

HP_SYS_CLKRST_BUSMON_CPU_CLK_EN 配置是否使能总线监控的总线时钟(CPU_CLK 时钟域)。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_L1CACHE_CPU_CLK_EN 配置是否使能 L1 Cache 的总线时钟（CPU_CLK 时钟域)。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_L1CACHE_D_CPU_CLK_EN 配置是否使能 L1 D-Cache 的总线时钟 (CPU_CLK 时钟域)。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_L1CACHE_I0_CPU_CLK_EN 配置是否使能 L1 I-Cache 0 的总线时钟（CPU_CLK 时钟域)。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_L1CACHE_I1_CPU_CLK_EN 配置是否使能 L1 I-Cache 1 的总线时钟（CPU_CLK 时钟域)。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_TRACE_CPU_CLK_EN 配置是否使能 RISC-V 追踪编码器的总线时钟（CPU_CLK 时钟域)。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_ICM_CPU_CLK_EN 配置总线模块的总线时钟（CPU_CLK 时钟域)。

0：禁用

1：使能

(R/W)

见下页...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

接上页...

HP_SYS_CLKRST_GDMA_CPU_CLK_EN 配置是否使能 VDMA 的总线时钟（CPU_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L1CACHE_MEM_CLK_EN 配置是否使能 L1 Cache 的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L1CACHE_D_MEM_CLK_EN 配置是否使能 L1 D-Cache 的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L1CACHE_I0_MEM_CLK_EN 配置是否使能 L1 I-Cache0 的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L1CACHE_I1_MEM_CLK_EN 配置是否使能 L1 I-Cache1 的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L2CACHE_MEM_CLK_EN 配置是否使能 L2 Cache 的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L2MEM_MEM_CLK_EN 配置是否使能 L2 存储器的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

见下页...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

接上页...

HP_SYS_CLKRST_L2MEMMON_MEM_CLK_EN 配置是否使能 L1 存储器监控的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_ICM_MEM_CLK_EN 配置总线模块的总线时钟（MEM_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_MISC_SYS_CLK_EN 配置是否使能 misc（杂项）的总线时钟（SYS_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_TRACE_SYS_CLK_EN 配置是否使能 RISC-V 追踪编码器的总线时钟（SYS_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L2CACHE_SYS_CLK_EN 配置是否使能 L2 Cache 的总线时钟（SYS_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L2MEM_SYS_CLK_EN 配置是否使能 L2 存储器的总线时钟（SYS_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_L2MEMMON_SYS_CLK_EN 配置是否使能 L2 存储器监控的总线时钟（SYS_CLK 时钟域）。
0：禁用
1：使能
(R/W)

见下页...

Register 11.6. HP_SYS_CLKRST_SOC_CLK_CTRL0_REG (0x0014)

接上页...

HP_SYS_CLKRST_SPMMON_SYS_CLK_EN 配置是否使能 L2 SPM 监控的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_ICM_SYS_CLK_EN 配置总线模块的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_FLASH_SYS_CLK_EN 配置是否使能 flash 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

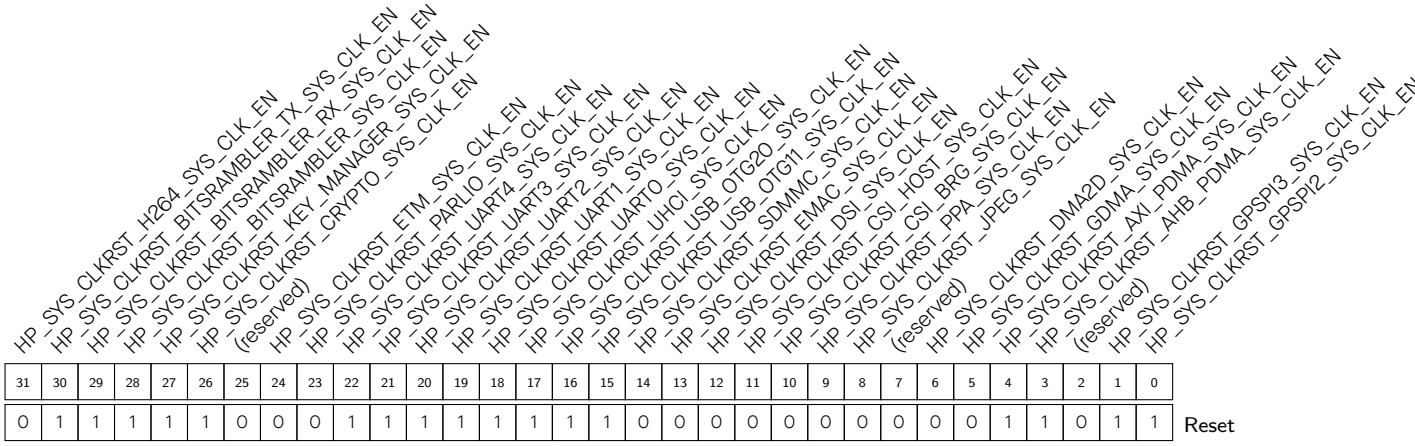
HP_SYS_CLKRST_PSRAM_SYS_CLK_EN 配置是否使能 PSRAM 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)



HP_SYS_CLKRST_GPSPI2_SYS_CLK_EN 配置是否使能 GP-SPI2 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_GPSPI3_SYS_CLK_EN 配置是否使能 GP-SPI3 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_AHB_PDMA_SYS_CLK_EN 配置是否使能 GDMA-AHB 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_AXI_PDMA_SYS_CLK_EN 配置是否使能 GDMA-AXI 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_GDMA_SYS_CLK_EN 配置是否使能 VDMA 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

接上页...

HP_SYS_CLKRST_DMA2D_SYS_CLK_EN 配置是否使能 2DDMA 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_JPEG_SYS_CLK_EN 配置是否使能 JPEG 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_PPA_SYS_CLK_EN 配置是否使能 PPA 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CSI_BRG_SYS_CLK_EN 配置是否使能 CSI Bridge 和 ISP 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CSI_HOST_SYS_CLK_EN 配置是否使能 CSI Host 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_DSI_SYS_CLK_EN 配置是否使能 DSI 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

接上页...

HP_SYS_CLKRST_EMAC_SYS_CLK_EN 配置是否使能 EMAC 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_SDMMC_SYS_CLK_EN 配置是否使能 SDMMC 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_USB_OTG11_SYS_CLK_EN 配置是否使能 USB 2.0 全速 OTG 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_USB_OTG20_SYS_CLK_EN 配置是否使能 USB 2.0 高速 OTG 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UHCI_SYS_CLK_EN 配置是否使能 UHCI 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART0_SYS_CLK_EN 配置是否使能 UART0 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART1_SYS_CLK_EN 配置是否使能 UART1 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

接上页...

HP_SYS_CLKRST_UART2_SYS_CLK_EN 配置是否使能 UART2 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART3_SYS_CLK_EN 配置是否使能 UART3 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART4_SYS_CLK_EN 配置是否使能 UART4 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_PARLIO_SYS_CLK_EN 配置是否使能 PARLIO 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_ETM_SYS_CLK_EN 配置是否使能 ETM 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_SYS_CLK_EN 配置是否使能加密模块的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_KEY_MANAGER_SYS_CLK_EN 配置是否使能密钥管理器的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.7. HP_SYS_CLKRST_SOC_CLK_CTRL1_REG (0x0018)

接上页...

HP_SYS_CLKRST_BITSRAMBLER_SYS_CLK_EN 配置是否使能比特传输控制器的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_BITSRAMBLER_RX_SYS_CLK_EN 配置是否使能比特传输控制器 RX 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_BITSRAMBLER_TX_SYS_CLK_EN 配置是否使能比特传输控制器 TX 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

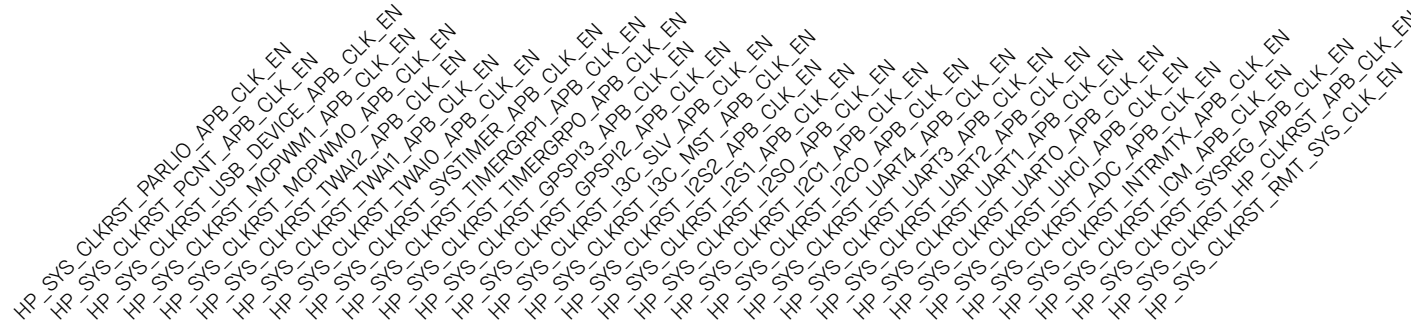
HP_SYS_CLKRST_H264_SYS_CLK_EN 配置是否使能 H264 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	1	0	Reset

HP_SYS_CLKRST_RMT_SYS_CLK_EN 配置是否使能 RMT 的总线时钟（SYS_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_HP_CLKRST_APB_CLK_EN 配置是否使能复位和时钟模块的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_SYSREG_APB_CLK_EN 配置是否使能 HP 系统寄存器的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_ICM_APB_CLK_EN 配置总线模块的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_INTRMTX_APB_CLK_EN 配置是否使能中断矩阵的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_ADC_APB_CLK_EN 配置是否使能 ADC 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

接上页...

HP_SYS_CLKRST_UHCI_APB_CLK_EN 配置是否使能 UHCI 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART0_APB_CLK_EN 配置是否使能 UART0 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART1_APB_CLK_EN 配置是否使能 UART1 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART2_APB_CLK_EN 配置是否使能 UART2 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART3_APB_CLK_EN 配置是否使能 UART3 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_UART4_APB_CLK_EN 配置是否使能 UART4 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2C0_APB_CLK_EN 配置是否使能 I2C0 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2C1_APB_CLK_EN 配置是否使能 I2C1 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

接上页...

HP_SYS_CLKRST_I2S0_APB_CLK_EN 配置是否使能 I2S0 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S1_APB_CLK_EN 配置是否使能 I2S1 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S2_APB_CLK_EN 配置是否使能 I2S2 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I3C_MST_APB_CLK_EN 配置是否使能 I3C 主机的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I3C_SLV_APB_CLK_EN 配置是否使能 I3C 从机的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_GPSPI2_APB_CLK_EN 配置是否使能 GP-SPI2 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_GPSPI3_APB_CLK_EN 配置是否使能 GP-SPI3 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

见下页...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

接上页...

HP_SYS_CLKRST_TIMERGRP0_APB_CLK_EN 配置是否使能定时器组 0 的总线时钟（APB_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_TIMERGRP1_APB_CLK_EN 配置是否使能定时器组 1 的总线时钟（APB_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_SYSTIMER_APB_CLK_EN 配置是否使能系统定时器的总线时钟（APB_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_TWAI0_APB_CLK_EN 配置是否使能 TWAI0 的总线时钟（APB_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_TWAI1_APB_CLK_EN 配置是否使能 TWAI1 的总线时钟（APB_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_TWAI2_APB_CLK_EN 配置是否使能 TWAI2 的总线时钟（APB_CLK 时钟域）。
0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_MCPWM0_APB_CLK_EN 配置是否使能 MCPWM0 的总线时钟（APB_CLK 时钟域）。
0：禁用
1：使能
(R/W)

见下页...

Register 11.8. HP_SYS_CLKRST_SOC_CLK_CTRL2_REG (0x001C)

接上页...

HP_SYS_CLKRST_MCPWM1_APB_CLK_EN 配置是否使能 MCPWM1 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_USB_DEVICE_APB_CLK_EN 配置是否使能 USB 串口/ JTAG 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_PCNT_APB_CLK_EN 配置是否使能 PCNT 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

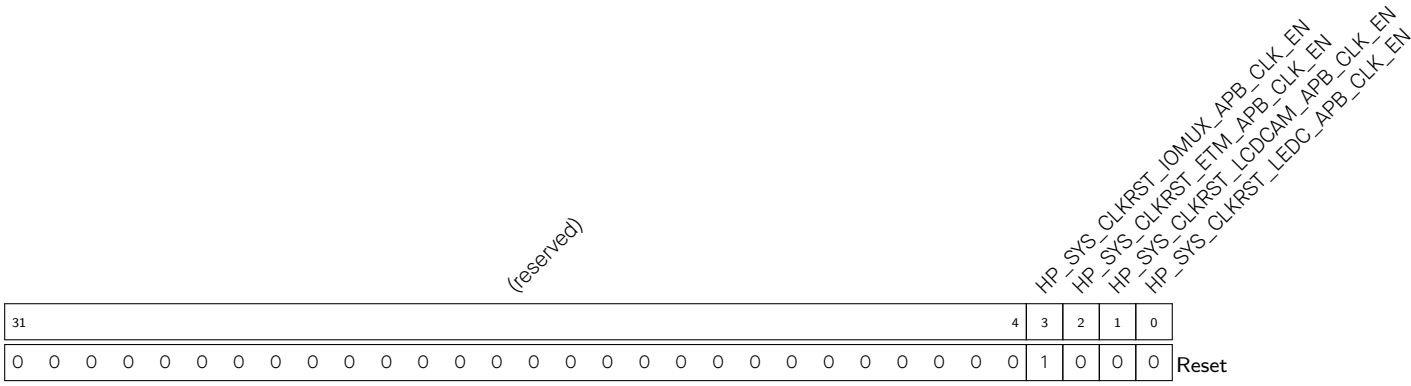
HP_SYS_CLKRST_PARLIO_APB_CLK_EN 配置是否使能 PARLIO 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

Register 11.9. HP_SYS_CLKRST_SOC_CLK_CTRL3_REG (0x0020)



HP_SYS_CLKRST_LEDC_APB_CLK_EN 配置是否使能 LEDC 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_LCDCAM_APB_CLK_EN 配置是否使能 LCD_CAM 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_ETM_APB_CLK_EN 配置是否使能 ETM 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

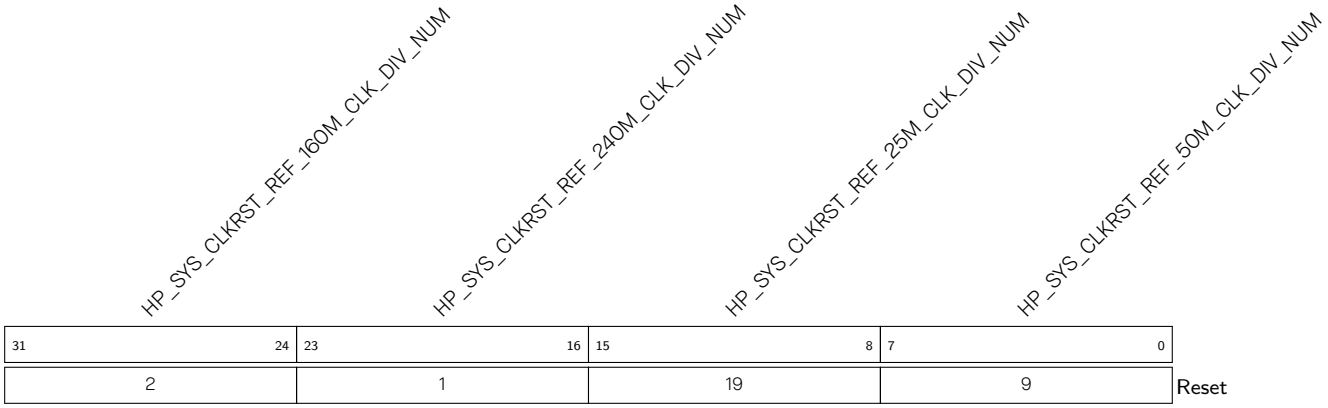
HP_SYS_CLKRST_IOMUX_APB_CLK_EN 配置是否使能 IO MUX 的总线时钟（APB_CLK 时钟域）。

0：禁用

1：使能

(R/W)

Register 11.10. HP_SYS_CLKRST_REF_CLK_CTRL0_REG (0x0024)



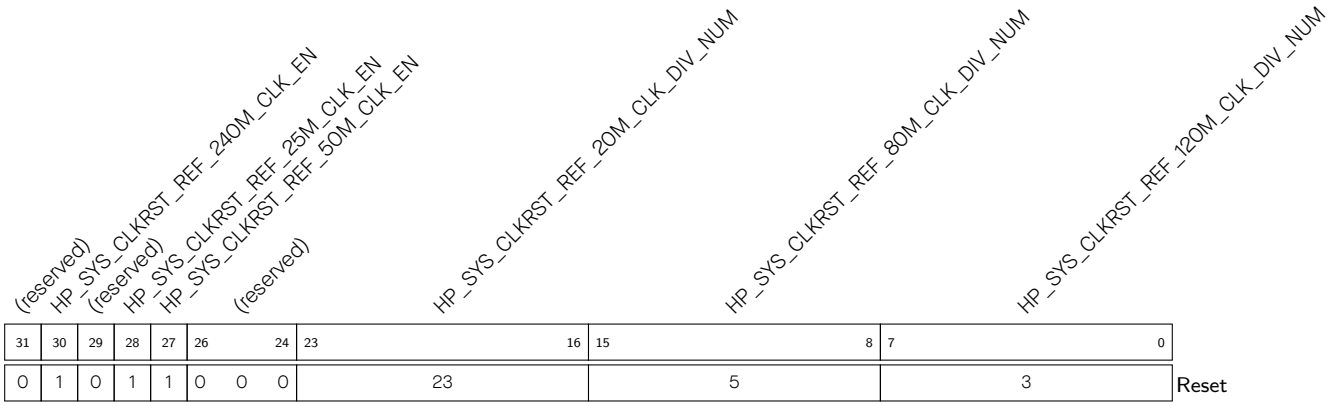
HP_SYS_CLKRST_REF_50M_CLK_DIV_NUM 配置 PLL_F50M_CLK 的分频系数。不建议用户更改。(R/W)

HP_SYS_CLKRST_REF_25M_CLK_DIV_NUM 配置 PLL_F25M_CLK 的分频系数。不建议用户更改。(R/W)

HP_SYS_CLKRST_REF_240M_CLK_DIV_NUM 配置 PLL_F240M_CLK 分的频系数。不建议用户更改。(R/W)

HP_SYS_CLKRST_REF_160M_CLK_DIV_NUM 配置 PLL_F160M_CLK 的分频系数。不建议用户更改。(R/W)

Register 11.11. HP_SYS_CLKRST_REF_CLK_CTRL1_REG (0x0028)



HP_SYS_CLKRST_REF_120M_CLK_DIV_NUM 配置 PLL_F120M_CLK 的分频系数。(R/W)

HP_SYS_CLKRST_REF_80M_CLK_DIV_NUM 配置 PLL_F80M_CLK 的分频系数。(R/W)

HP_SYS_CLKRST_REF_20M_CLK_DIV_NUM 配置 PLL_F20M_CLK 的分频系数。(R/W)

HP_SYS_CLKRST_REF_50M_CLK_EN 配置是否使能 PLL_F50M_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_REF_25M_CLK_EN 配置是否使能 PLL_F25M_CLK。

0：禁用

1：使能

(R/W)

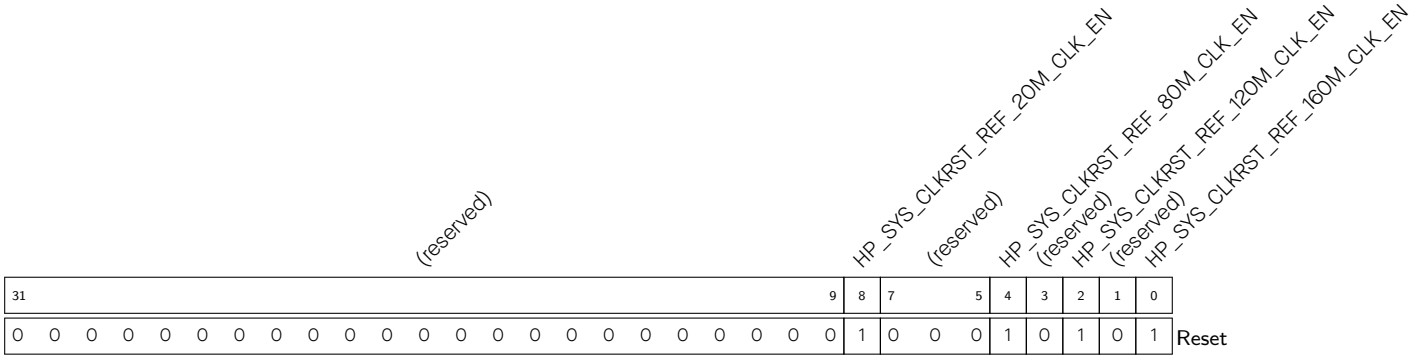
HP_SYS_CLKRST_REF_240M_CLK_EN 配置是否使能 PLL_F240M_CLK。

0：禁用

1：使能

(R/W)

Register 11.12. HP_SYS_CLKRST_REF_CLK_CTRL2_REG (0x002C)



HP_SYS_CLKRST_REF_160M_CLK_EN 配置是否使能 PLL_F160M_CLK。

0：禁用

1：使能

不建议用户更改。(R/W)

HP_SYS_CLKRST_REF_120M_CLK_EN 配置是否使能 PLL_F120M_CLK。

0：禁用

1：使能

不建议用户更改。(R/W)

HP_SYS_CLKRST_REF_80M_CLK_EN 配置是否使能 PLL_F80M_CLK。

0：禁用

1：使能

不建议用户更改。(R/W)

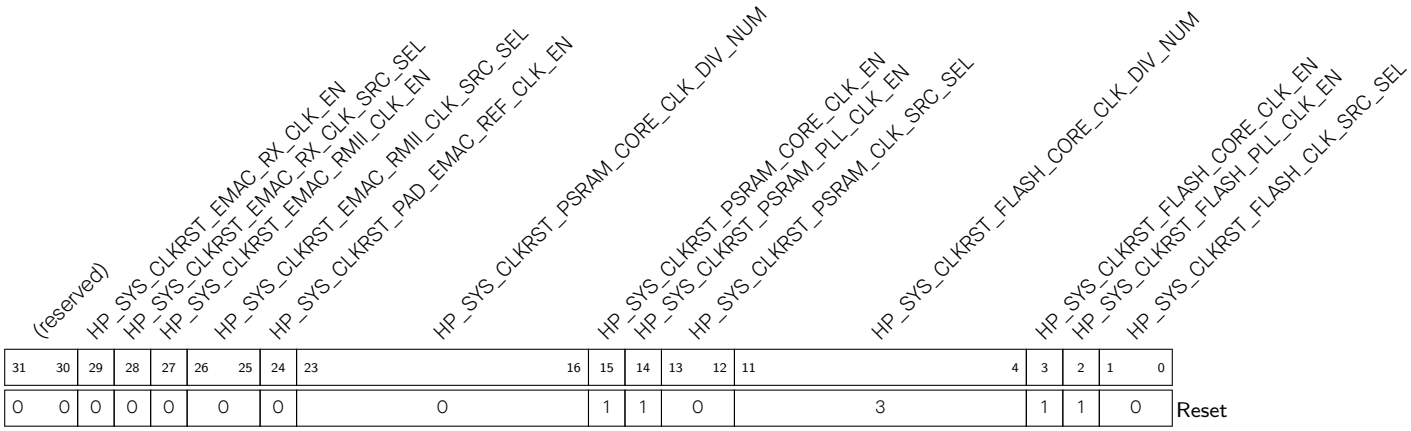
HP_SYS_CLKRST_REF_20M_CLK_EN 配置是否使能 PLL_F20M_CLK。

0：禁用

1：使能

不建议用户更改。(R/W)

Register 11.13. HP_SYS_CLKRST_PERI_CLK_CTRL00_REG (0x0030)



HP_SYS_CLKRST_FLASH_CLK_SRC_SEL 配置 FLASH_CLK 时钟源。

0：XTAL_CLK

1：SPLL_CLK (480 MHz)

2：CPLL_CLK (360 MHz)

3：非法值

(R/W)

HP_SYS_CLKRST_FLASH_PLL_CLK_EN 配置是否使能 FLASH_PLL_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_FLASH_CORE_CLK_EN 配置是否使能 FLASH_CORE_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_FLASH_CORE_CLK_DIV_NUM 配置 FLASH_CORE_CLK 的时钟分频系数。(R/W)

HP_SYS_CLKRST_PSRAM_CLK_SRC_SEL 配置 PSRAM_CLK 时钟源。

0：XTAL_CLK

1：MPLL_CLK

2：SPLL_CLK (480 MHz)

3：CPLL_CLK (360 MHz)

(R/W)

HP_SYS_CLKRST_PSRAM_PLL_CLK_EN 配置是否使能 PSRAM_PLL_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_PSRAM_CORE_CLK_EN 配置是否使能 PSRAM_CORE_CLK。

0：禁用

1：使能

(R/W)

见下页...

Register 11.13. HP_SYS_CLKRST_PERI_CLK_CTRL00_REG (0x0030)

接上页...

HP_SYS_CLKRST_PSRAM_CORE_CLK_DIV_NUM 配置 PSRAM_CORE_CLK 时钟分频系数。(R/W)

HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN 配置是否使能 PAD_EMAC_REF_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_EMAC_RMII_CLK_SRC_SEL 配置 EMAC_RMII_CLK 时钟源。

0：PAD_EMAC_TXRX_CLK

1：PAD_EMAC_RX_CLK

2：PAD_EMAC_TX_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_EMAC_RMII_CLK_EN 配置是否使能 EMAC_RMII_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL 配置 EMAC_RX_CLK 时钟源。

0：PAD_EMAC_TXRX_CLK

1：PAD_EMAC_RX_CLK

(R/W)

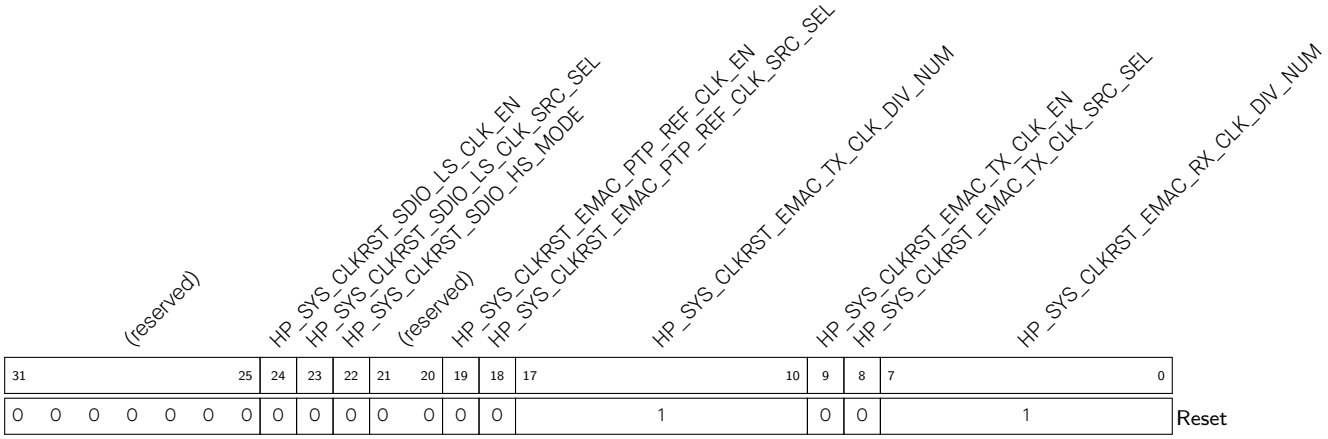
HP_SYS_CLKRST_EMAC_RX_CLK_EN 配置是否使能 EMAC_RX_CLK。

0：禁用

1：使能

(R/W)

Register 11.14. HP_SYS_CLKRST_PERI_CLK_CTRL01_REG (0x0034)



HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM 配置 EMAC_RX_CLK 时钟分频系数。(R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL 配置 EMAC_TX_CLK 时钟源。

0：PAD_EMAC_TXRX_CLK

1：PAD_EMAC_TX_CLK

(R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_EN 配置是否使能 EMAC_TX_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM 配置 EMAC_TX_CLK 时钟分频系数。(R/W)

HP_SYS_CLKRST_EMAC_PTP_REF_CLK_SRC_SEL 配置 EMAC_PTP_REF_CLK 时钟源。

0：XTAL_CLK

1：PLL_F80M_CLK

(R/W)

HP_SYS_CLKRST_EMAC_PTP_REF_CLK_EN 配置是否使能 EMAC_PTP_REF_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_SDIO_HS_MODE 配置 SDIO 时钟的模式。

0：将 SDIO 时钟切换到低速模式

1：将 SDIO 时钟切换到高速模式

(R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_SRC_SEL 配置 SDIO 低速时钟源。

0：PLL_F160M_CLK

1：APLL_CLK

(R/W)

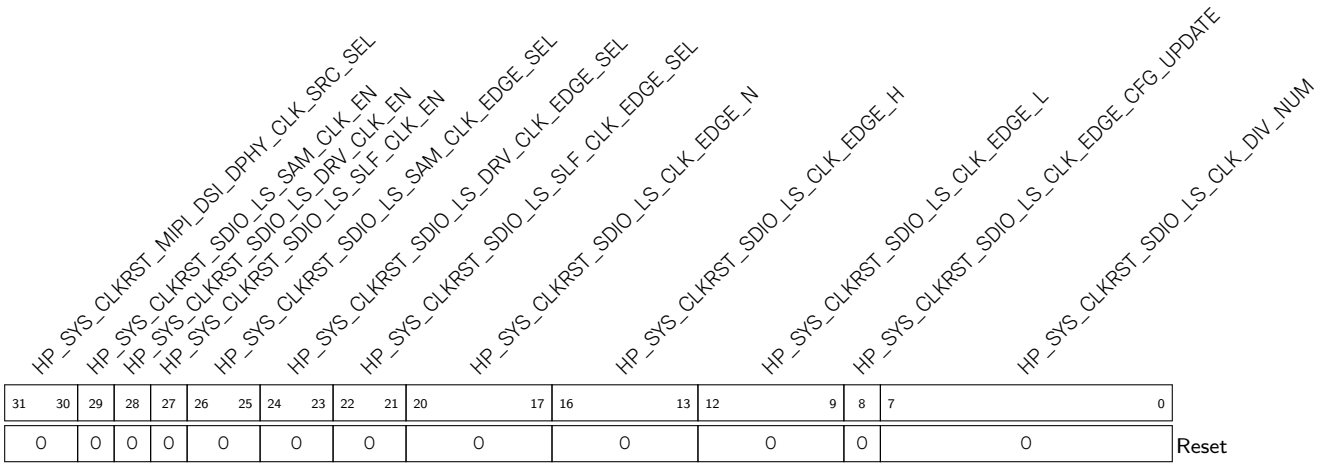
HP_SYS_CLKRST_SDIO_LS_CLK_EN 配置是否使能 SDIO 低速时钟。

0：禁用

1：使能

(R/W)

Register 11.15. HP_SYS_CLKRST_PERI_CLK_CTRL02_REG (0x0038)



HP_SYS_CLKRST_SDIO_LS_CLK_DIV_NUM 配置 SDIO 低速时钟的分频系数。(R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_CFG_UPDATE 配置是否更新 SDIO 低速时钟相位配置。

0：不更新

1：更新 (WT)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_L 配置 SDIO 低速时钟信号跳转到低电平的阈值。(R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_H 配置 SDIO 低速时钟信号跳转到高电平的阈值。(R/W)

HP_SYS_CLKRST_SDIO_LS_CLK_EDGE_N 配置 SDIO 低速时钟计数值清零的阈值。(R/W)

HP_SYS_CLKRST_SDIO_LS_SLF_CLK_EDGE_SEL 配置 SDIO_SLF_CLK 选择 SDIO 低速时钟相位。(R/W)

HP_SYS_CLKRST_SDIO_LS_DRV_CLK_EDGE_SEL 配置 SDIO_DRV_CLK 选择 SDIO 低速时钟相位。(R/W)

HP_SYS_CLKRST_SDIO_LS_SAM_CLK_EDGE_SEL 配置 SDIO_SAM_CLK 选择 SDIO 低速时钟相位。(R/W)

HP_SYS_CLKRST_SDIO_LS_SLF_CLK_EN 配置是否使能 SDIO_SLF_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_SDIO_LS_DRV_CLK_EN 配置是否使能 SDIO_DRV_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_SDIO_LS_SAM_CLK_EN 配置是否使能 SDIO_SAM_CLK。

0：禁用

1：使能

(R/W)

见下页...

Register 11.15. HP_SYS_CLKRST_PERI_CLK_CTRL02_REG (0x0038)

接上页...

HP_SYS_CLKRST_MIPI_DSI_DPHY_CLK_SRC_SEL 配置 DSI_DPHY_CFG_CLK 和 DSI_DPHY_PLL_REFCLK 的时钟源。

0：PLL_F20M_CLK

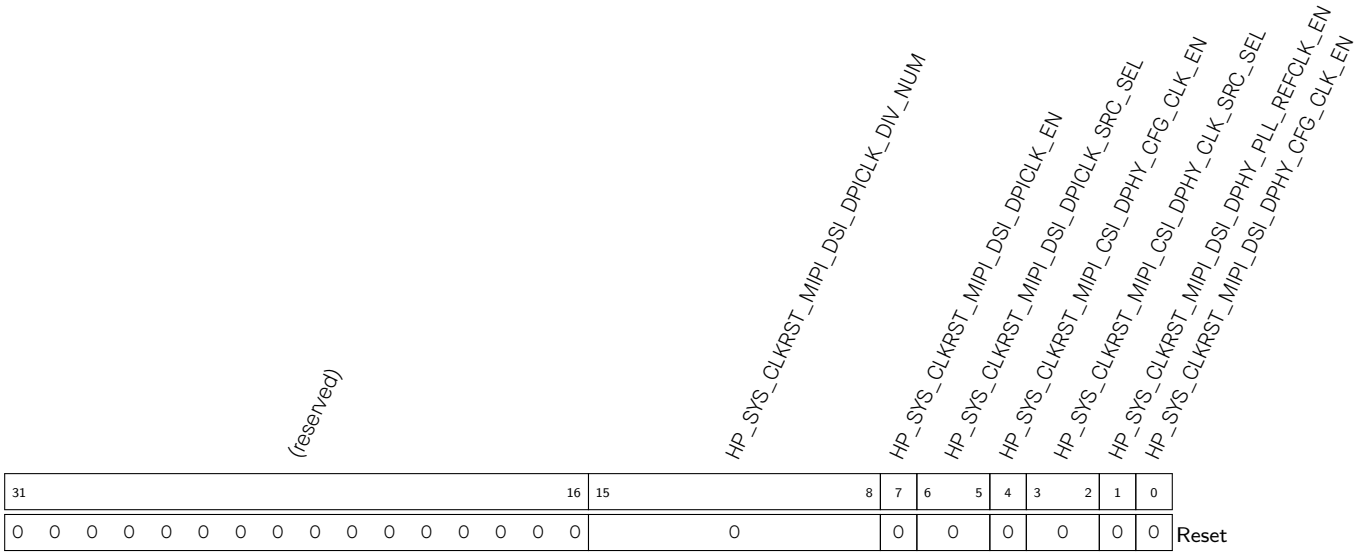
1：RC_FAST_CLK

2：PLL_F25M_CLK

3：非法值

(R/W)

Register 11.16. HP_SYS_CLKRST_PERI_CLK_CTRL03_REG (0x003C)



HP_SYS_CLKRST_MIPI_DSI_DPHY_CFG_CLK_EN 配置是否使能 DSI_DPHY_CFG_CLK 时钟。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPHY_PLL_REFCLK_EN 配置是否使能 DSI_DPHY_PLL_REFCLK 时钟。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_MIPI_CSI_DPHY_CLK_SRC_SEL 配置 CSI_DPHY_CFG_CLK 的时钟源。

0：PLL_F20M_CLK

1：RC_FAST_CLK

2：PLL_F25M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_MIPI_CSI_DPHY_CFG_CLK_EN 配置是否使能 CSI_DPHY_CFG_CLK 时钟。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPICLK_SRC_SEL 配置 DSI_DPI_CLK 的时钟源。

0：XTAL_CLK

1：PLL_F240M_CLK

2：PLL_F160M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_MIPI_DSI_DPICLK_EN 配置是否使能 DSI_DPI_CLK 时钟。

0：禁用

1：使能

(R/W)

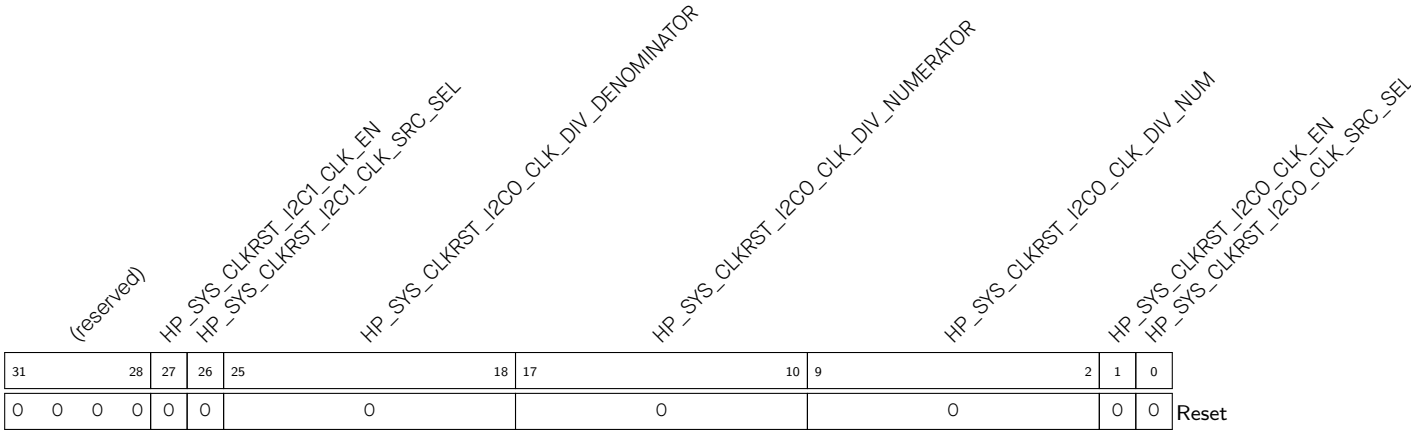
见下页...

Register 11.16. HP_SYS_CLKRST_PERI_CLK_CTRL03_REG (0x003C)

接上页...

HP_SYS_CLKRST_MIPI_DSI_DPICLK_DIV_NUM 配置 DSI_DPI_CLK 时钟分频系数。(R/W)

Register 11.17. HP_SYS_CLKRST_PERI_CLK_CTRL10_REG (0x0040)



HP_SYS_CLKRST_I2C0_CLK_SRC_SEL 配置 I2C0_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_I2C0_CLK_EN 配置是否使能 I2C0_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_NUM 配置 I2C0_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_NUMERATOR 配置 I2C0_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_I2C0_CLK_DIV_DENOMINATOR 配置 I2C0_CLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_I2C1_CLK_SRC_SEL 配置 I2C1_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

(R/W)

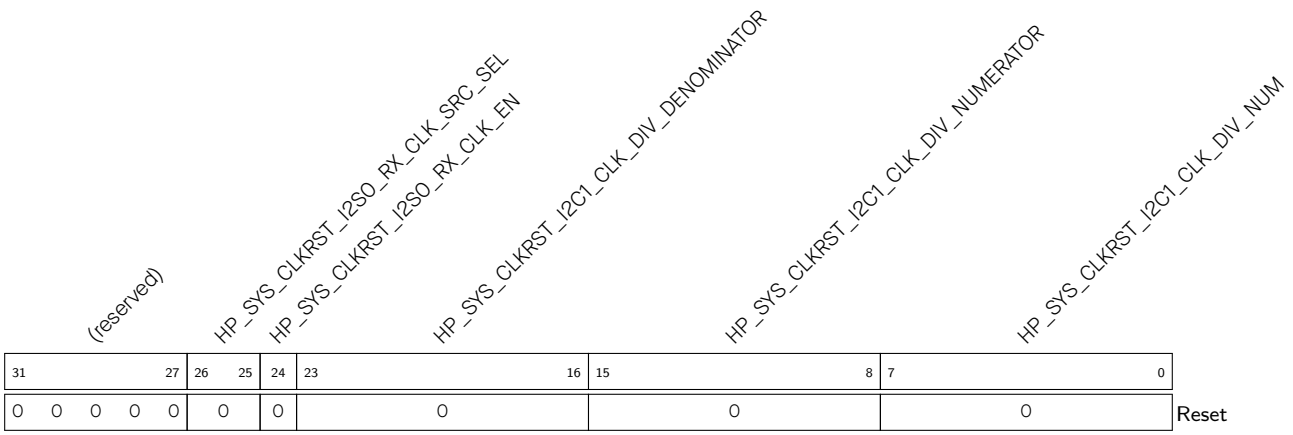
HP_SYS_CLKRST_I2C1_CLK_EN 配置是否使能 I2C1_CLK。

0：禁用

1：使能

(R/W)

Register 11.18. HP_SYS_CLKRST_PERI_CLK_CTRL11_REG (0x0044)



HP_SYS_CLKRST_I2C1_CLK_DIV_NUM 配置 I2C1_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_I2C1_CLK_DIV_NUMERATOR 配置 I2C1_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_I2C1_CLK_DIV_DENOMINATOR 配置 I2C1_CLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_I2S0_RX_CLK_EN 配置是否使能 I2S0_RX_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S0_RX_CLK_SRC_SEL 配置 I2S0_RX_CLK 的时钟源。

0：XTAL_CLK

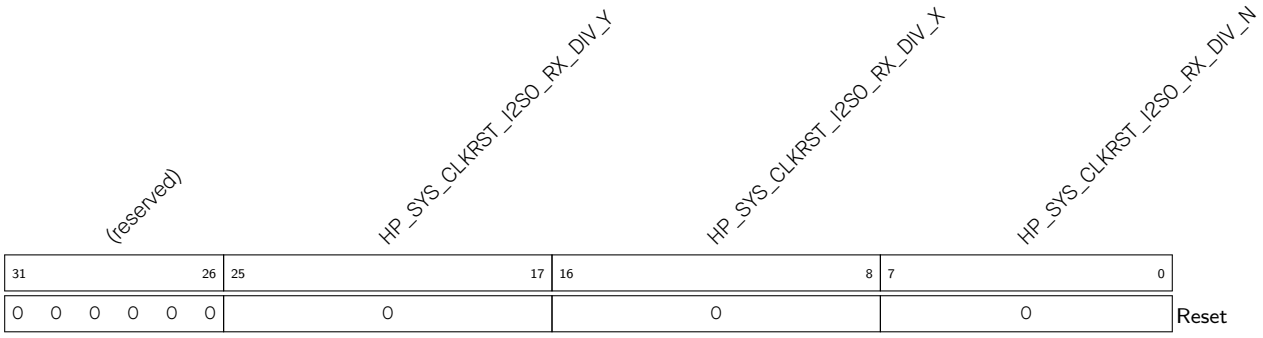
1：APLL_CLK

2：PAD_I2S0_MCLK

3：非法值

(R/W)

Register 11.19. HP_SYS_CLKRST_PERI_CLK_CTRL12_REG (0x0048)

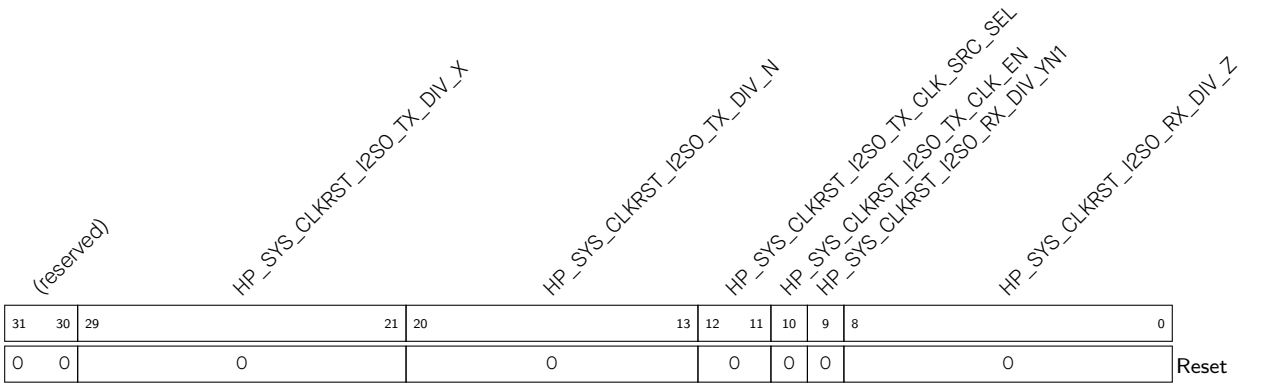


HP_SYS_CLKRST_I2S0_RX_DIV_N 配置 I2S0_RX_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_X 配置 I2S0_RX_CLK 时钟分频的系数 x (R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_Y 配置 I2S0_RX_CLK 时钟分频的系数 y。(R/W)

Register 11.20. HP_SYS_CLKRST_PERI_CLK_CTRL13_REG (0x004C)



HP_SYS_CLKRST_I2S0_RX_DIV_Z 配置 I2S0_RX_CLK 时钟分频的系数 z。(R/W)

HP_SYS_CLKRST_I2S0_RX_DIV_YN1 配置 I2S0_RX_CLK 时钟分频的系数 yn1。(R/W)

HP_SYS_CLKRST_I2S0_TX_CLK_EN 配置是否使能 I2S0_TX_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S0_TX_CLK_SRC_SEL 配置 I2S0_TX_CLK 的时钟源。

0：XTAL_CLK

1：APLL_CLK

2：PAD_I2S0_MCLK

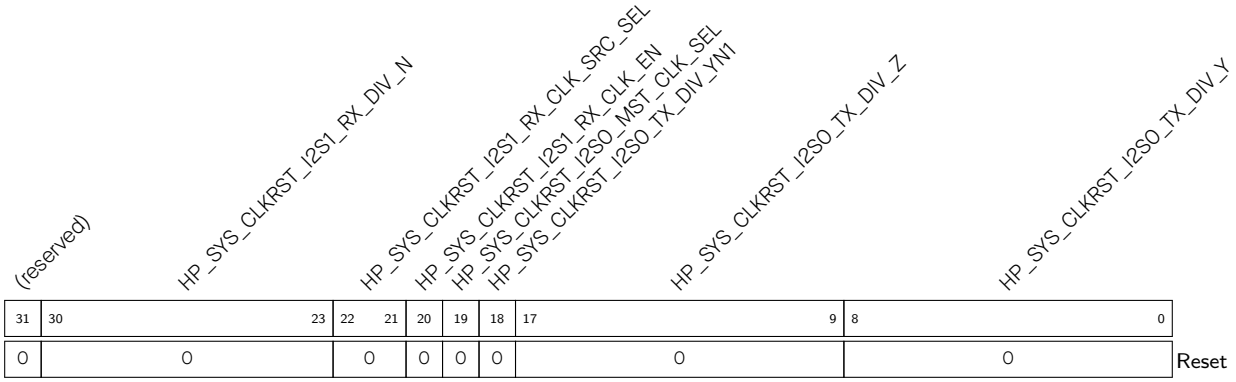
3：非法值

(R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_N 配置 I2S0_TX_CLK 的时钟分频系数整数部分。(R/W)

HP_SYS_CLKRST_I2S0_TX_DIV_X 配置 I2S0_TX_CLK 时钟分频的系数 x。(R/W)

Register 11.21. HP_SYS_CLKRST_PERI_CLK_CTRL14_REG (0x0050)



HP_SYS_CLKRST_I2SO_TX_DIV_Y 配置 I2SO_TX_CLK 时钟分频的系数 y。(R/W)

HP_SYS_CLKRST_I2SO_TX_DIV_Z 配置 I2SO_TX_CLK 时钟分频的系数 z。(R/W)

HP_SYS_CLKRST_I2SO_TX_DIV_YN1 配置 I2SO_TX_CLK 时钟分频的系数 yn1。(R/W)

HP_SYS_CLKRST_I2SO_MST_CLK_SEL 配置输出时钟 PAD_I2SO_MST_CLK 的时钟源。

0：I2SO_RX_CLK

1：I2SO_TX_CLK

(R/W)

HP_SYS_CLKRST_I2S1_RX_CLK_EN 配置是否使能 I2S1_RX_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S1_RX_CLK_SRC_SEL 配置 I2S1_RX_CLK 的时钟源。

0：XTAL_CLK

1：APLL_CLK

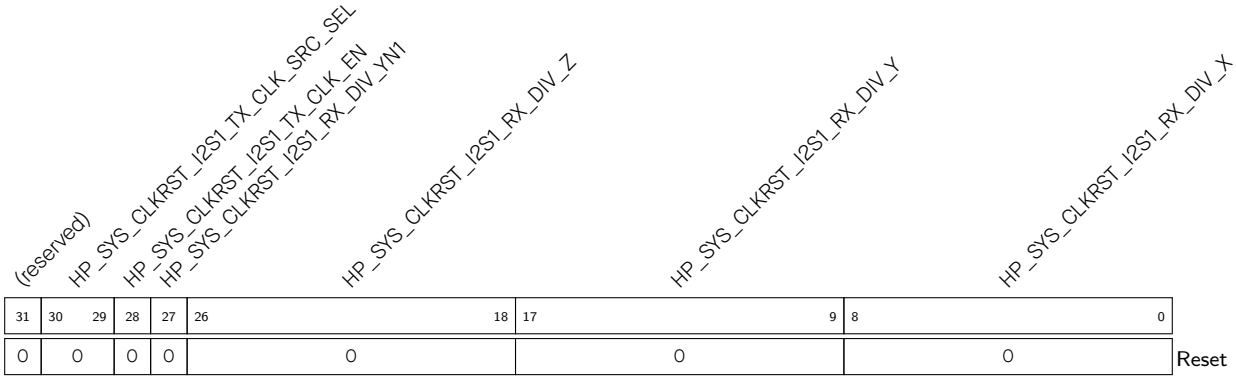
2：PAD_I2S1_MCLK

3：非法值

(R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_N 配置 I2S1_RX_CLK 的时钟分频系数整数部分。(R/W)

Register 11.22. HP_SYS_CLKRST_PERI_CLK_CTRL15_REG (0x0054)



HP_SYS_CLKRST_I2S1_RX_DIV_X 配置 I2S1_RX_CLK 时钟分频的系数 x。(R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_Y 配置 I2S1_RX_CLK 时钟分频的系数 y。(R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_Z 配置 I2S1_RX_CLK 时钟分频的系数 z。(R/W)

HP_SYS_CLKRST_I2S1_RX_DIV_YN1 配置 I2S1_RX_CLK 时钟分频的系数 yn1。(R/W)

HP_SYS_CLKRST_I2S1_TX_CLK_EN 配置是否使能 PI2S1_TX_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S1_TX_CLK_SRC_SEL 配置 I2S1_TX_CLK 的时钟源。

0：XTAL_CLK

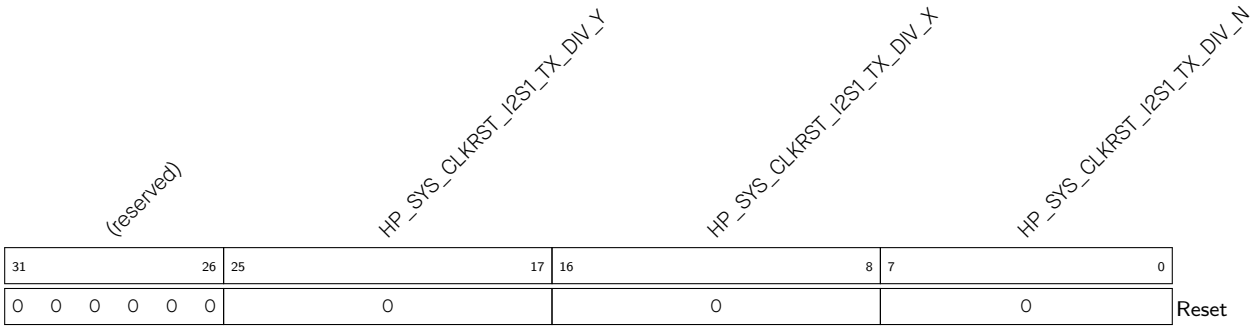
1：APLL_CLK

2：PAD_I2S1_MCLK

3：非法值

(R/W)

Register 11.23. HP_SYS_CLKRST_PERI_CLK_CTRL16_REG (0x0058)

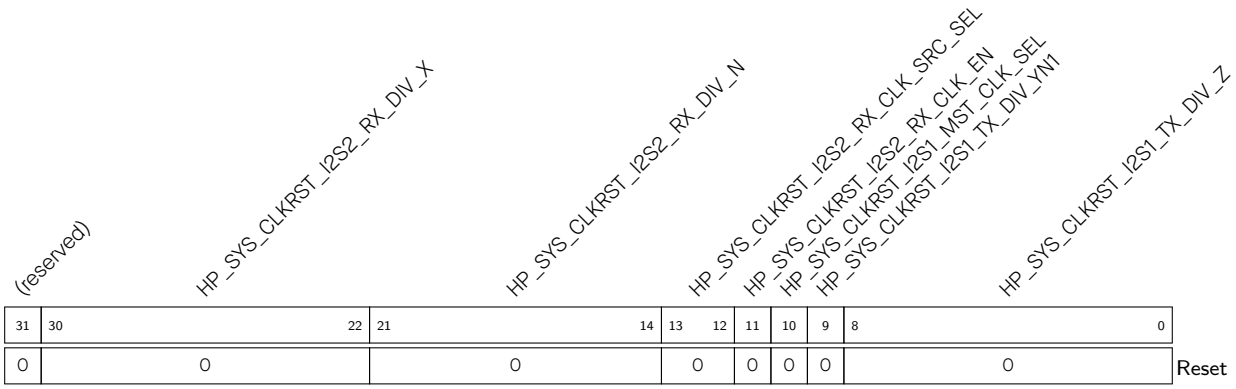


HP_SYS_CLKRST_I2S1_TX_DIV_N 配置 I2S1_TX_CLK 的时钟分频系数整数部分 (R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_X 配置 I2S1_TX_CLK 时钟分频的系数 x。(R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_Y 配置 I2S1_TX_CLK 时钟分频的系数 y。(R/W)

Register 11.24. HP_SYS_CLKRST_PERI_CLK_CTRL17_REG (0x005C)



HP_SYS_CLKRST_I2S1_TX_DIV_Z 配置 I2S1_TX_CLK 时钟分频的系数 z。(R/W)

HP_SYS_CLKRST_I2S1_TX_DIV_YN1 配置 I2S1_TX_CLK 时钟分频的系数 yn1。(R/W)

HP_SYS_CLKRST_I2S1_MST_CLK_SEL 配置输出时钟 PAD_I2S1_MST_CLK 的时钟源。

0：I2S1_RX_CLK

1：I2S1_TX_CLK

(R/W)

HP_SYS_CLKRST_I2S2_RX_CLK_EN 配置是否使能 I2S2_RX_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S2_RX_CLK_SRC_SEL 配置 I2S2_RX_CLK 的时钟源。

0：XTAL_CLK

1：APLL_CLK

2：PAD_I2S2_MCLK

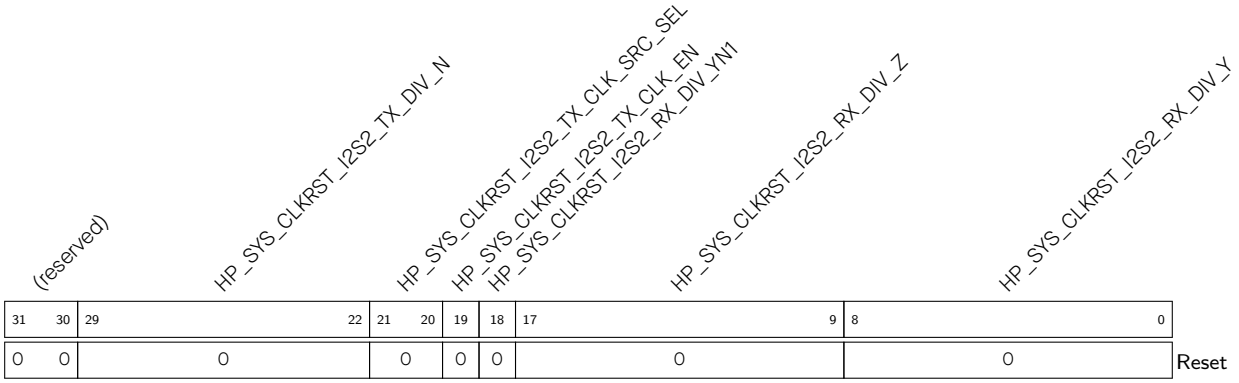
3：非法值

(R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_N 配置 I2S2_RX_CLK 的时钟分频系数整数部分。(R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_X 配置 I2S2_RX_CLK 时钟分频的系数 x。(R/W)

Register 11.25. HP_SYS_CLKRST_PERI_CLK_CTRL18_REG (0x0060)



HP_SYS_CLKRST_I2S2_RX_DIV_Y 配置 I2S2_RX_CLK 时钟分频的系数 y。(R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_Z 配置 I2S2_RX_CLK 时钟分频的系数 z。(R/W)

HP_SYS_CLKRST_I2S2_RX_DIV_YN1 配置 I2S2_RX_CLK 时钟分频的系数 yn1。(R/W)

HP_SYS_CLKRST_I2S2_TX_CLK_EN 配置是否使能 I2S2_TX_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_I2S2_TX_CLK_SRC_SEL 配置 I2S2_TX_CLK 的时钟源。

0：XTAL_CLK

1：APLL_CLK

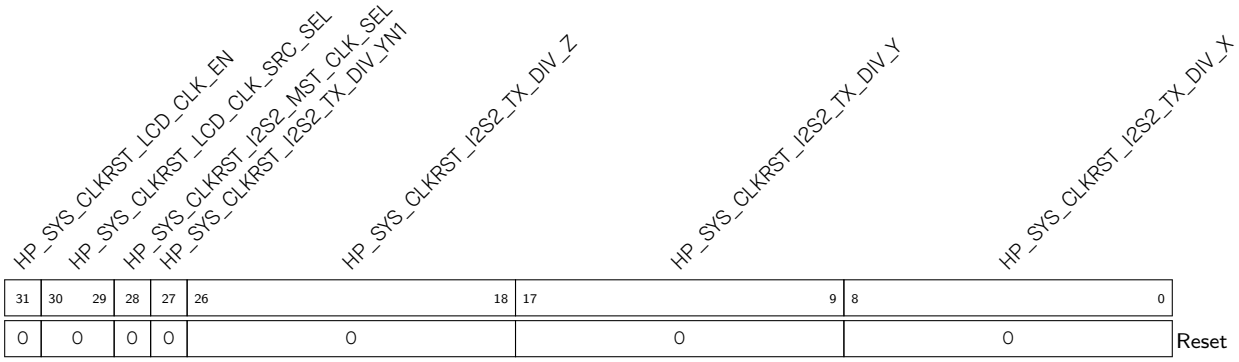
2：PAD_I2S2_MCLK

3：非法值

(R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_N 配置 I2S2_TX_CLK 的时钟分频系数整数部分。(R/W)

Register 11.26. HP_SYS_CLKRST_PERI_CLK_CTRL19_REG (0x0064)



HP_SYS_CLKRST_I2S2_TX_DIV_X 配置 I2S2_TX_CLK 时钟分频的系数 x。(R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_Y 配置 I2S2_TX_CLK 时钟分频的系数 y。(R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_Z 配置 I2S2_TX_CLK 时钟分频的系数 z。(R/W)

HP_SYS_CLKRST_I2S2_TX_DIV_YN1 配置 I2S2_TX_CLK 时钟分频的系数 yn1。(R/W)

HP_SYS_CLKRST_I2S2_MST_CLK_SEL 配置输出时钟 PAD_I2S2_MST_CLK 的时钟源。

0：I2S2_RX_CLK

1：I2S2_TX_CLK

(R/W)

HP_SYS_CLKRST_LCD_CLK_SRC_SEL 配置 LCD_CLK 的时钟源。

0：XTAL_CLK

1：PLL_F160M_CLK

2：APLL_CLK

3：非法值

(R/W)

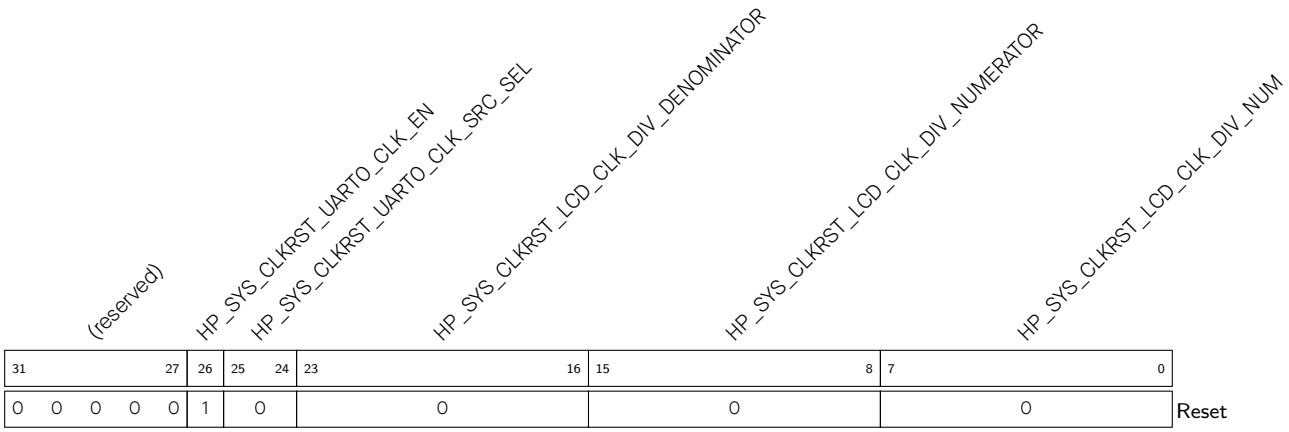
HP_SYS_CLKRST_LCD_CLK_EN 配置是否使能 LCD_CLK。

0：禁用

1：使能

(R/W)

Register 11.27. HP_SYS_CLKRST_PERI_CLK_CTRL110_REG (0x0068)



HP_SYS_CLKRST_LCD_CLK_DIV_NUM 配置 LCD_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR 配置 LCD_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR 配置 LCD_CLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_UART0_CLK_SRC_SEL 配置 UART0_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

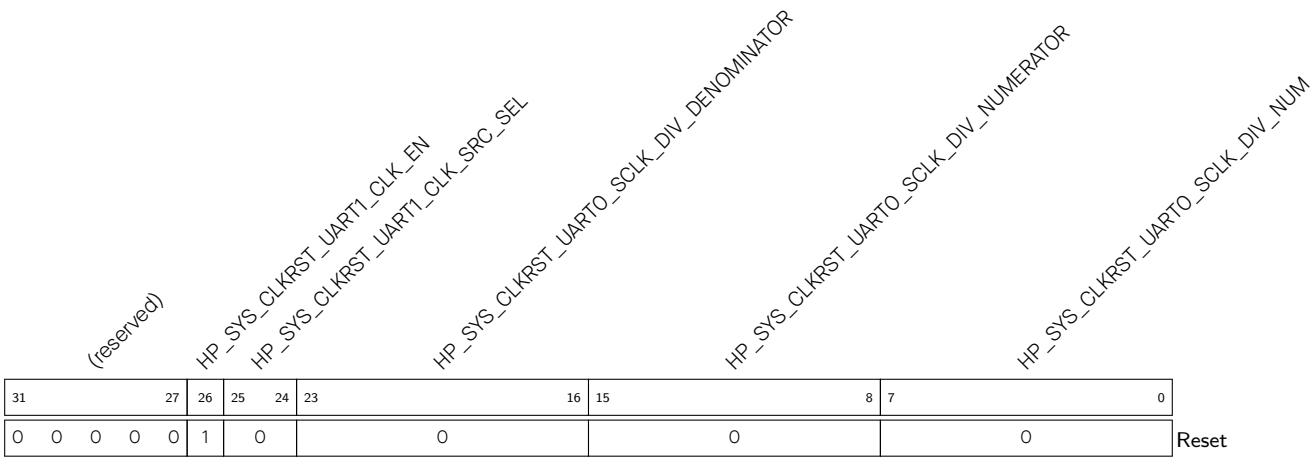
HP_SYS_CLKRST_UART0_CLK_EN 配置是否使能 UART0_CLK。

0：禁用

1：使能

(R/W)

Register 11.28. HP_SYS_CLKRST_PERI_CLK_CTRL111_REG (0x006C)



HP_SYS_CLKRST_UART0_SCLK_DIV_NUM 配置 UART0_SCLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_UART0_SCLK_DIV_NUMERATOR 配置 UART0_SCLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_UART0_SCLK_DIV_DENOMINATOR 配置 UART0_SCLK 时钟分频系数小数部分的分母。(R/W)

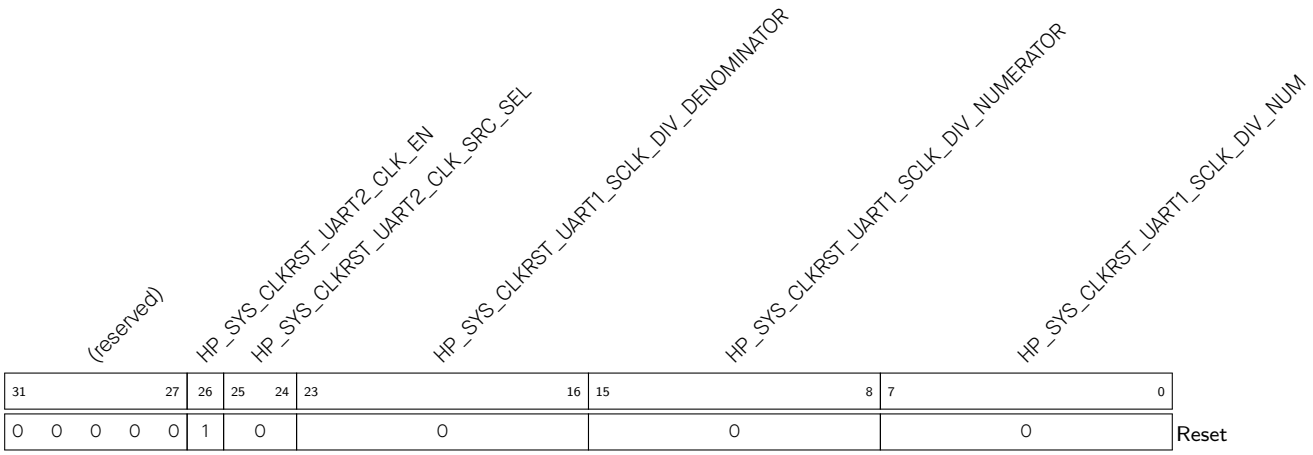
HP_SYS_CLKRST_UART1_CLK_SRC_SEL 配置 UART1_CLK 的时钟源。

0：XTAL_CLK
1：RC_FAST_CLK
2：PLL_F80M_CLK
3：非法值
(R/W)

HP_SYS_CLKRST_UART1_CLK_EN 配置是否使能 UART1_CLK。

0：禁用
1：使能
(R/W)

Register 11.29. HP_SYS_CLKRST_PERI_CLK_CTRL112_REG (0x0070)



HP_SYS_CLKRST_UART1_SCLK_DIV_NUM 配置 UART1_SCLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_UART1_SCLK_DIV_NUMERATOR 配置 UART1_SCLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_UART1_SCLK_DIV_DENOMINATOR 配置 UART1_SCLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_UART2_CLK_SRC_SEL 配置 UART2_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

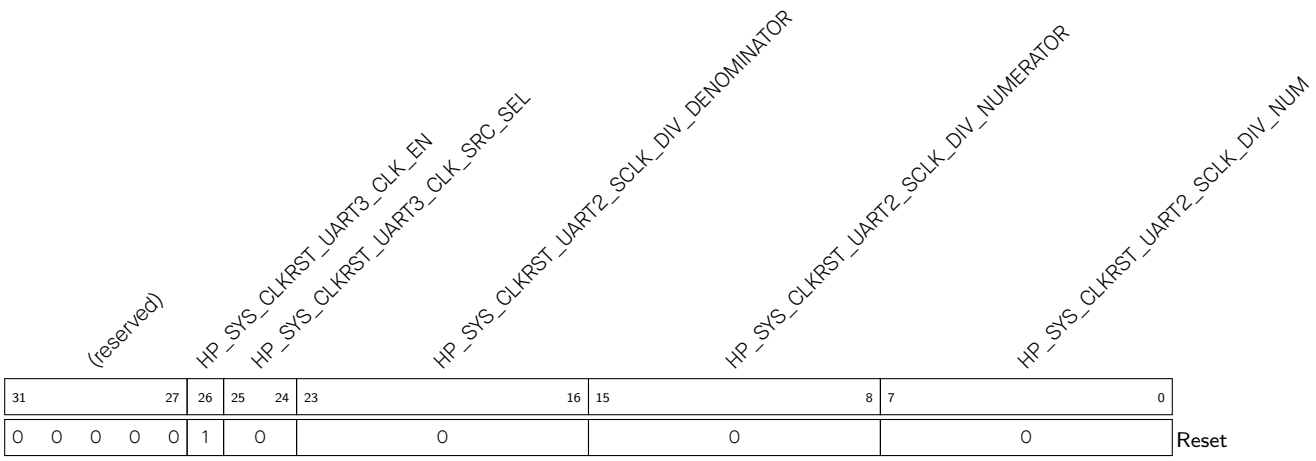
HP_SYS_CLKRST_UART2_CLK_EN 配置是否使能 UART2_CLK 时钟。

0：禁用

1：使能

(R/W)

Register 11.30. HP_SYS_CLKRST_PERI_CLK_CTRL113_REG (0x0074)



HP_SYS_CLKRST_UART2_SCLK_DIV_NUM 配置 UART2_SCLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_UART2_SCLK_DIV_NUMERATOR 配置 UART2_SCLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_UART2_SCLK_DIV_DENOMINATOR 配置 UART2_SCLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_UART3_CLK_SRC_SEL 配置 UART3_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

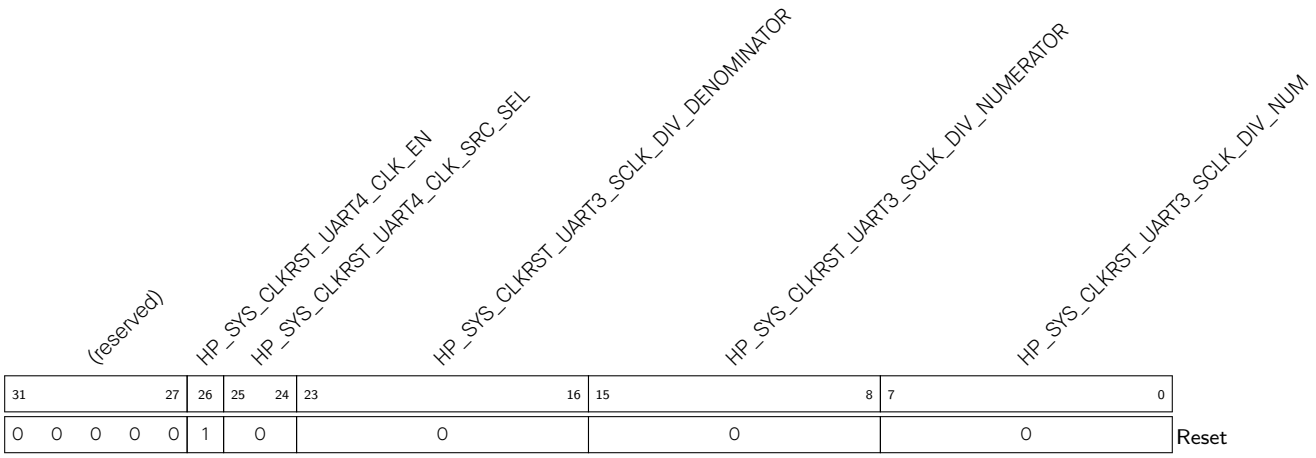
HP_SYS_CLKRST_UART3_CLK_EN 配置是否使能 UART3_CLK 时钟。

0：禁用

1：使能

(R/W)

Register 11.31. HP_SYS_CLKRST_PERI_CLK_CTRL114_REG (0x0078)



HP_SYS_CLKRST_UART3_SCLK_DIV_NUM 配置 UART3_SCLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_UART3_SCLK_DIV_NUMERATOR 配置 UART3_SCLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_UART3_SCLK_DIV_DENOMINATOR 配置 UART3_SCLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_UART4_CLK_SRC_SEL 配置 UART4_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

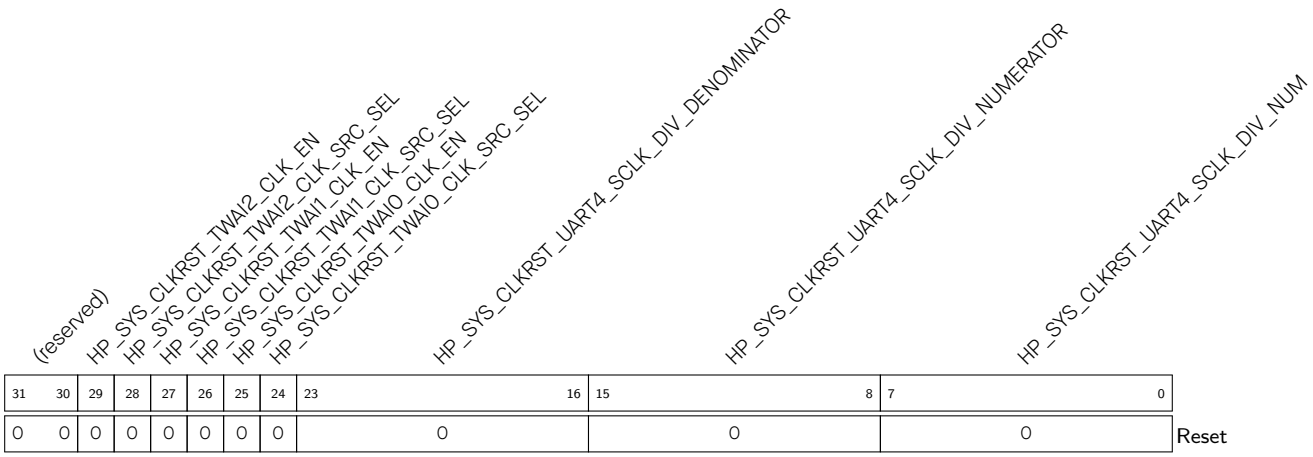
HP_SYS_CLKRST_UART4_CLK_EN 配置是否使能 UART4_CLK 时钟。

0：禁用

1：使能

(R/W)

Register 11.32. HP_SYS_CLKRST_PERI_CLK_CTRL115_REG (0x007C)



HP_SYS_CLKRST_UART4_SCLK_DIV_NUM 配置 UART4_SCLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_UART4_SCLK_DIV_NUMERATOR 配置 UART4_SCLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_UART4_SCLK_DIV_DENOMINATOR 配置 UART4_SCLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_TWAI0_CLK_SRC_SEL 配置 TWAI0_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_TWAI0_CLK_EN 配置是否使能 TWAI0_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_TWAI1_CLK_SRC_SEL 配置 TWAI1_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

(R/W)

HP_SYS_CLKRST_TWAI1_CLK_EN 配置是否使能 TWAI1_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_TWAI2_CLK_SRC_SEL 配置 TWAI2_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

(R/W)

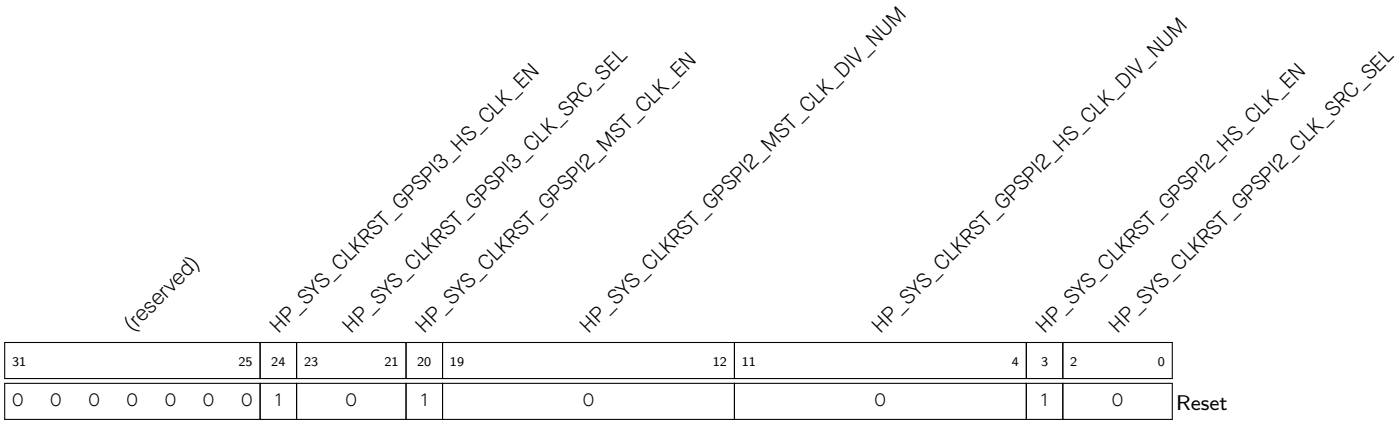
HP_SYS_CLKRST_TWAI2_CLK_EN 配置是否使能 TWAI2_CLK。

0：禁用

1：使能

(R/W)

Register 11.33. HP_SYS_CLKRST_PERI_CLK_CTRL116_REG (0x0080)



HP_SYS_CLKRST_GPSPI2_CLK_SRC_SEL 配置 GPSPI2_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：SDIO_PLL0_CLK

3：APLL_CLK

4：SPLL_CLK (480 MHz)

5-7：非法值

(R/W)

HP_SYS_CLKRST_GPSPI2_HS_CLK_EN 配置是否使能 GPSPI2_HS_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_GPSPI2_HS_CLK_DIV_NUM 配置 GPSPI2_HS_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_GPSPI2_MST_CLK_DIV_NUM 配置 GPSPI2_MST_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_GPSPI2_MST_CLK_EN 配置是否使能 GPSPI2_MST_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_GPSPI3_CLK_SRC_SEL 配置 GPSPI3_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：SDIO_PLL0_CLK

3：APLL_CLK

4：SPLL_CLK (480 MHz)

5-7：非法值

(R/W)

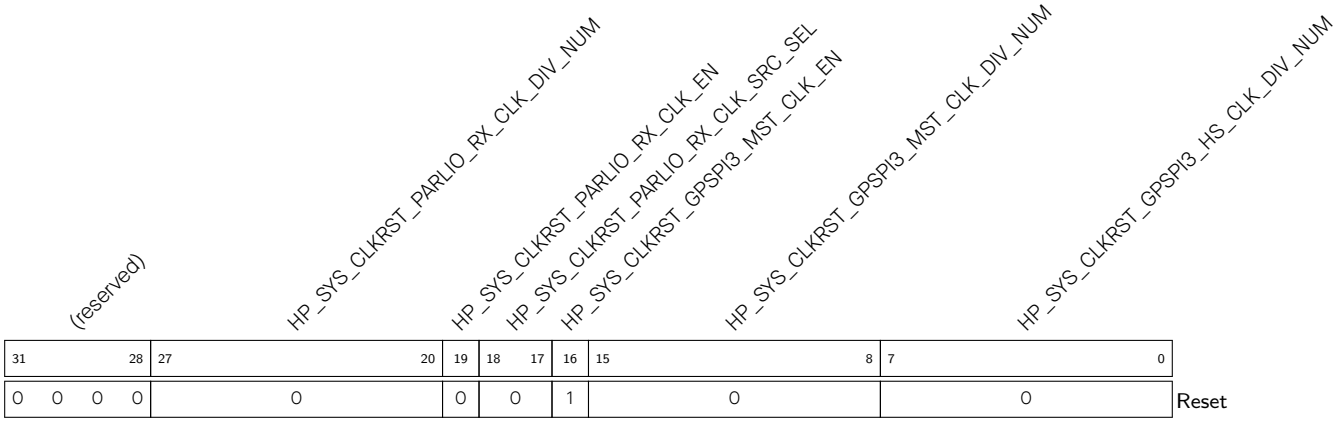
HP_SYS_CLKRST_GPSPI3_HS_CLK_EN 配置是否使能 GPSPI3_HS_CLK。

0：禁用

1：使能

(R/W)

Register 11.34. HP_SYS_CLKRST_PERI_CLK_CTRL117_REG (0x0084)



HP_SYS_CLKRST_GPSPI3_HS_CLK_DIV_NUM 配置 GPSPI3_HS_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_GPSPI3_MST_CLK_DIV_NUM 配置 GPSPI3_MST_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_GPSPI3_MST_CLK_EN 配置是否使能 GPSPI3_MST_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_SRC_SEL 配置 PARLIO_RX_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F160M_CLK

3：PAD_PARLIO_RX_CLK

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_EN 配置是否使能 PARLIO_RX_CLK。

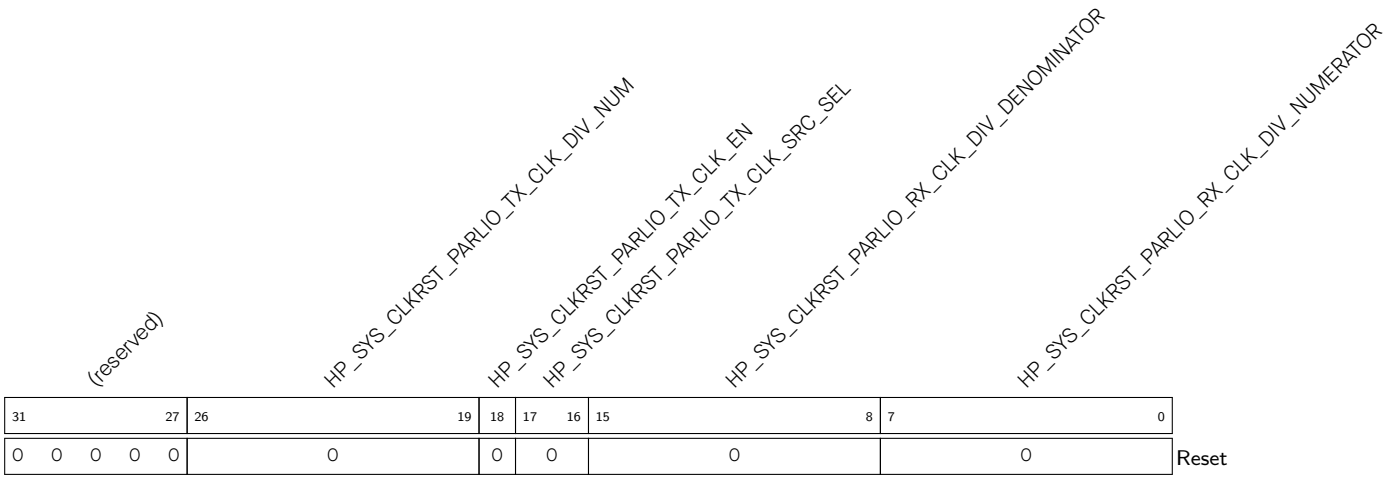
0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUM 配置 PARLIO_RX_CLK 时钟分频系数的整数部分。(R/W)

Register 11.35. HP_SYS_CLKRST_PERI_CLK_CTRL118_REG (0x0088)



HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUMERATOR 配置 PARLIO_RX_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_DENOMINATOR 配置 PARLIO_RX_CLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_SRC_SEL 配置 PARLIO_TX_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F160M_CLK

3：PAD_PARLIO_TX_CLK

(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_EN 配置是否使能 PARLIO_TX_CLK。

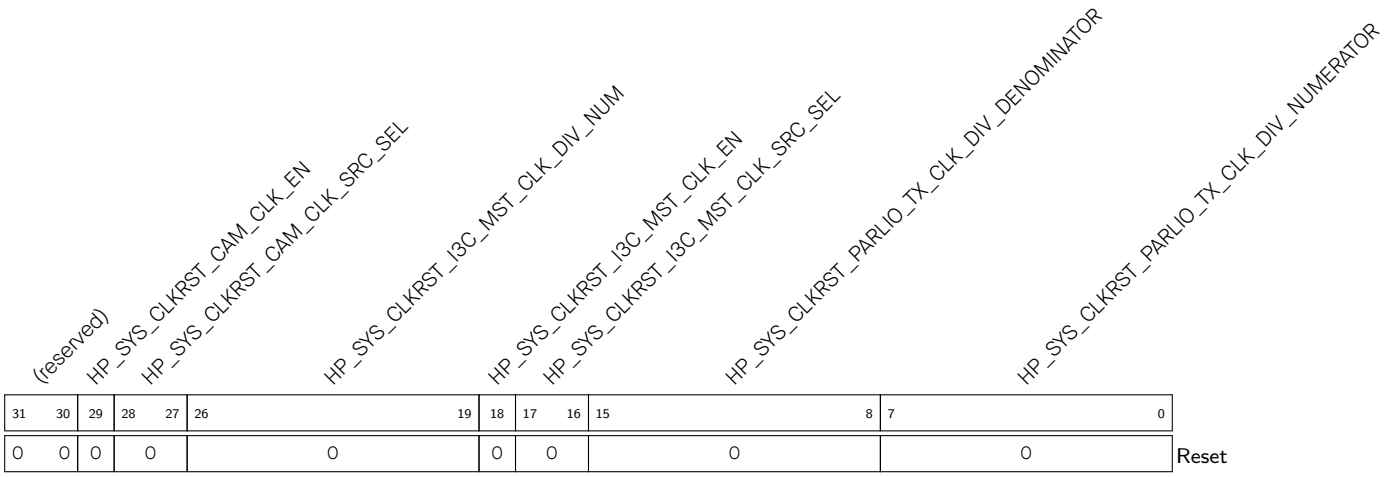
0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUM 配置 PARLIO_TX_CLK 时钟分频系数的整数部分。(R/W)

Register 11.36. HP_SYS_CLKRST_PERI_CLK_CTRL119_REG (0x008C)



HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUMERATOR 配置 PARLIO_TX_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_DENOMINATOR 配置 PARLIO_TX_CLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_I3C_MST_CLK_SRC_SEL 配置 I3C_MST_CLK 的时钟源。

0：XTAL_CLK
1：PLL_F160M_CLK
2：PLL_F120M_CLK
3：非法值
(R/W)

HP_SYS_CLKRST_I3C_MST_CLK_EN 配置是否使能 I3C_MST_CLK。

0：禁用
1：使能
(R/W)

HP_SYS_CLKRST_I3C_MST_CLK_DIV_NUM 配置 I3C_MST_CLK 时钟分频系数的整数部分。(R/W)

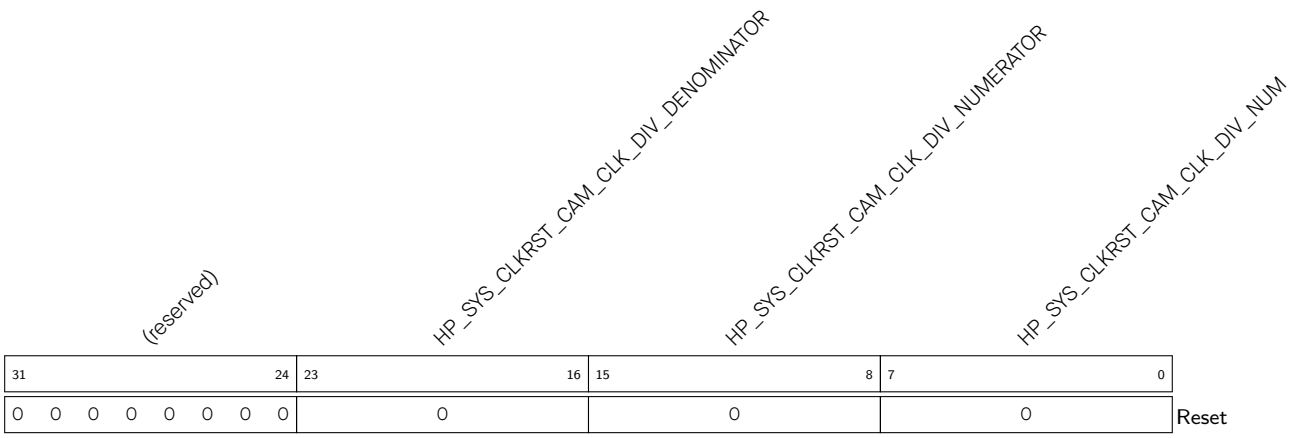
HP_SYS_CLKRST_CAM_CLK_SRC_SEL 配置输出时钟 CAM_CLK 的时钟源。

0：XTAL_CLK
1：PLL_F160M_CLK
2：APLL_CLK
3：非法值
(R/W)

HP_SYS_CLKRST_CAM_CLK_EN 配置是否使能输出时钟 CAM_CLK。

0：禁用
1：使能
(R/W)

Register 11.37. HP_SYS_CLKRST_PERI_CLK_CTRL120_REG (0x0090)

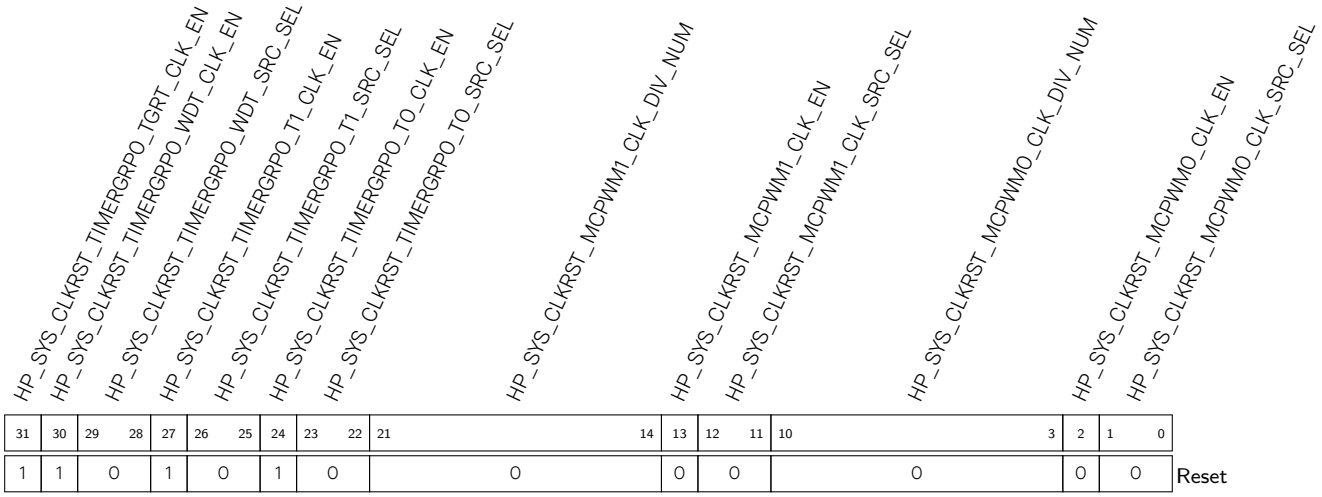


HP_SYS_CLKRST_CAM_CLK_DIV_NUM 配置输出时钟 CAM_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR 配置输出时钟 CAM_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR 配置输出时钟 CAM_CLK 时钟分频系数小数部分的分母。(R/W)

Register 11.38. HP_SYS_CLKRST_PERI_CLK_CTRL20_REG (0x0094)



HP_SYS_CLKRST_MCPWM0_CLK_SRC_SEL 配置 MCPWM0_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F160_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_MCPWM0_CLK_EN 配置是否使能 MCPWM0_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_MCPWM0_CLK_DIV_NUM 配置 MCPWM0_CLK 的时钟分频系数。(R/W)

HP_SYS_CLKRST_MCPWM1_CLK_SRC_SEL 配置 MCPWM1_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F160_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_MCPWM1_CLK_EN 配置是否使能 MCPWM1_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_MCPWM1_CLK_DIV_NUM 配置 MCPWM1_CLK 的时钟分频系数。(R/W)

HP_SYS_CLKRST_TIMERGRP0_T0_SRC_SEL 配置 TIMERGRP0_T0_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

见下页...

Register 11.38. HP_SYS_CLKRST_PERI_CLK_CTRL20_REG (0x0094)

接上页...

HP_SYS_CLKRST_TIMERGRP0_T0_CLK_EN 配置是否使能 TIMERGRP0_T0_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_TIMERGRP0_T1_SRC_SEL 配置 TIMERGRP0_T1_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_TIMERGRP0_T1_CLK_EN 配置是否使能 TIMERGRP0_T1_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_TIMERGRP0_WDT_SRC_SEL 配置 TIMERGRP0_WDT_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_TIMERGRP0_WDT_CLK_EN 配置是否使能 TIMERGRP0_WDT_CLK。

0：禁用

1：使能

(R/W)

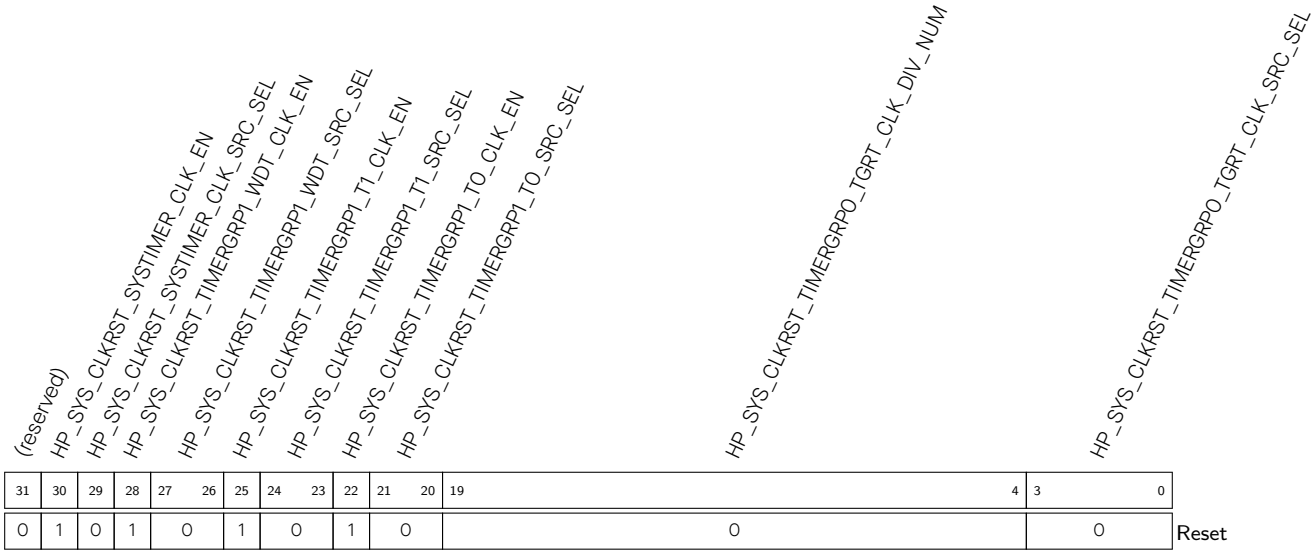
HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_EN 配置是否使能 TIMERGRP0_TGRT_CLK。

0：禁用

1：使能

(R/W)

Register 11.39. HP_SYS_CLKRST_PERI_CLK_CTRL21_REG (0x0098)



HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL 配置 TIMERGRP0_TGRT_CLK 的时钟源。

0：MPLL_CLK (500 MHz)
1：SPLL_CLK (480 MHz)
2：CPLL_CLK (360 MHz)
3：APLL_CLK
4：SDIO_PLL0_CLK
5：SDIO_PLL1_CLK
6：SDIO_PLL2_CLK
7：RC_FAST_CLK
8：RC_SLOW_CLK
9：无效值
10：XTAL32K_CLK
11：PLL_LP_CLK
12-15：时钟为 0
(R/W)

HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_DIV_NUM 配置 TIMERGRP0_TGRT_CLK 的时钟分频系数。(R/W)

HP_SYS_CLKRST_TIMERGRP1_T0_SRC_SEL 配置 TIMERGRP1_T0_CLK 的时钟源。

0：XTAL_CLK
1：RC_FAST_CLK
2：PLL_F80M_CLK
3：非法值
(R/W)

HP_SYS_CLKRST_TIMERGRP1_T0_CLK_EN 配置是否使能 TIMERGRP1_T0_CLK。

0：禁用
1：使能
(R/W)

见下页...

Register 11.39. HP_SYS_CLKRST_PERI_CLK_CTRL21_REG (0x0098)

接上页...

HP_SYS_CLKRST_TIMERGRP1_T1_SRC_SEL 配置 TIMERGRP1_T1_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_TIMERGRP1_T1_CLK_EN 配置是否使能 TIMERGRP1_T1_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_TIMERGRP1_WDT_SRC_SEL 配置 TIMERGRP1_WDT_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_TIMERGRP1_WDT_CLK_EN 配置是否使能 TIMERGRP1_WDT_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_SYSTIMER_CLK_SRC_SEL 配置 SYSTIMER_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

(R/W)

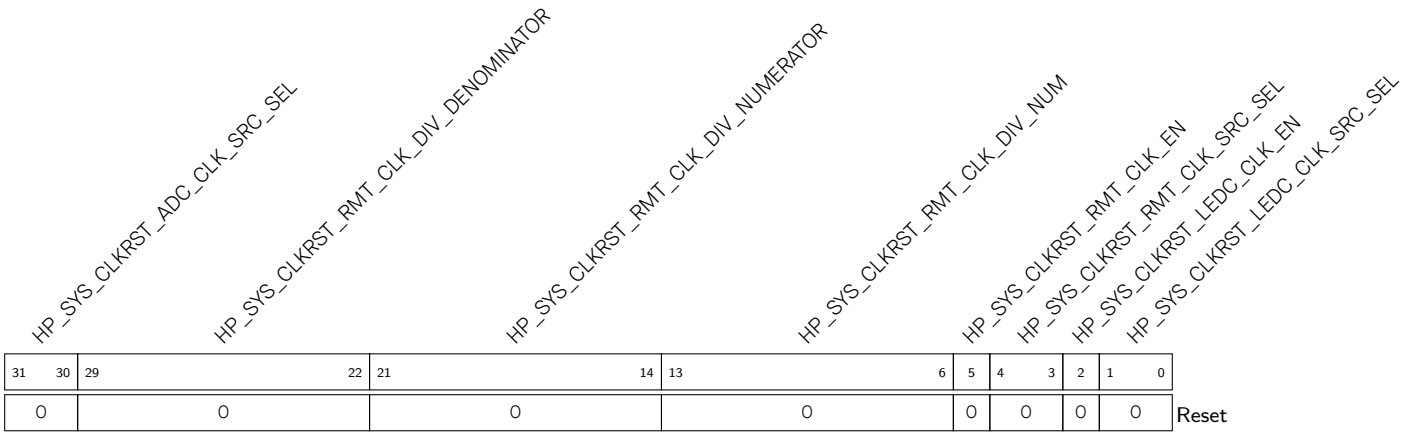
HP_SYS_CLKRST_SYSTIMER_CLK_EN 配置是否使能 SYSTIMER_CLK。

0：禁用

1：使能

(R/W)

Register 11.40. HP_SYS_CLKRST_PERI_CLK_CTRL22_REG (0x009C)



HP_SYS_CLKRST_LEDC_CLK_SRC_SEL 配置 LEDC_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_LEDC_CLK_EN 配置是否使能 LEDC_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_RMT_CLK_SRC_SEL 配置 RMT_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

HP_SYS_CLKRST_RMT_CLK_EN 配置是否使能 RMT_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_NUM 配置 RMT_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_NUMERATOR 配置 RMT_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_RMT_CLK_DIV_DENOMINATOR 配置 RMT_CLK 时钟分频系数小数部分的分母。(R/W)

HP_SYS_CLKRST_ADC_CLK_SRC_SEL 配置 HPADC_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F80M_CLK

3：非法值

(R/W)

Register 11.41. HP_SYS_CLKRST_PERI_CLK_CTRL23_REG (0x00A0)



HP_SYS_CLKRST_ADC_CLK_EN 配置是否使能 HPADC_CLK。

0：禁用

1：使能

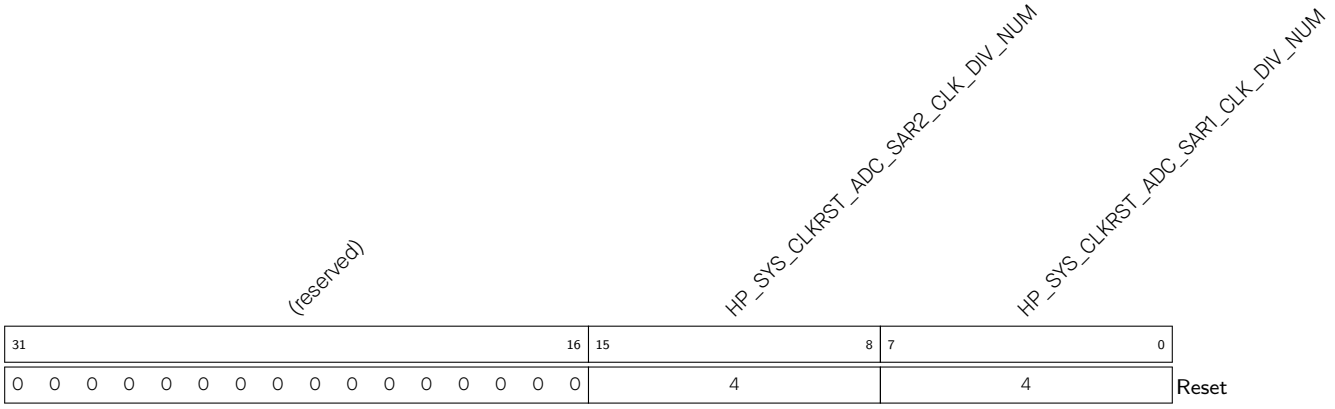
(R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_NUM 配置 HPADC_CLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_NUMERATOR 配置 HPADC_CLK 时钟分频系数小数部分的分子。(R/W)

HP_SYS_CLKRST_ADC_CLK_DIV_DENOMINATOR 配置 HPADC_CLK 时钟分频系数小数部分的分母。(R/W)

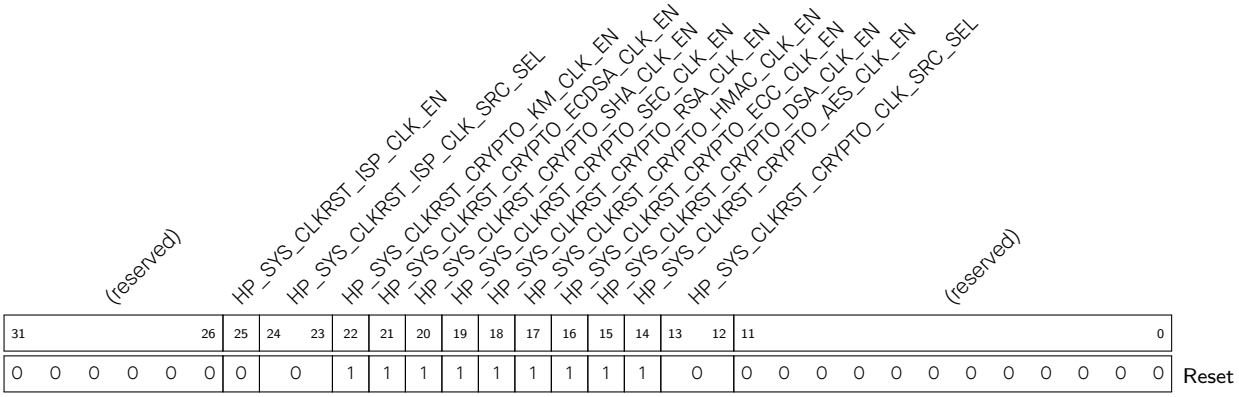
Register 11.42. HP_SYS_CLKRST_PERI_CLK_CTRL24_REG (0x00A4)



HP_SYS_CLKRST_ADC_SAR1_CLK_DIV_NUM 配置 HPADC1_SARCLK 时钟分频系数的整数部分。(R/W)

HP_SYS_CLKRST_ADC_SAR2_CLK_DIV_NUM 配置 HPADC2_SARCLK 时钟分频系数的整数部分。(R/W)

Register 11.43. HP_SYS_CLKRST_PERI_CLK_CTRL25_REG (0x00A8)



HP_SYS_CLKRST_CRYPTO_CLK_SRC_SEL 配置 CRYPTO_CLK 的时钟源。

0：XTAL_CLK

1：RC_FAST_CLK

2：PLL_F240M_CLK

3：PLL_F160M_CLK

(R/W)

HP_SYS_CLKRST_CRYPTO_AES_CLK_EN 配置是否使能 AES_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_DSA_CLK_EN 配置是否使能 RSA_DS 时钟。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_ECC_CLK_EN 配置是否使能 ECC_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_HMAC_CLK_EN 配置是否使能 HMAC_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_RSA_CLK_EN 配置是否使能 RSA_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_SEC_CLK_EN 配置是否使能 SEC_CLK。

0：禁用

1：使能

(R/W)

见下页...

Register 11.43. HP_SYS_CLKRST_PERI_CLK_CTRL25_REG (0x00A8)

接上页...

HP_SYS_CLKRST_CRYPTO_SHA_CLK_EN 配置是否使能 SHA_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_ECDSA_CLK_EN 配置是否使能 ECDSA_DS 时钟 (ECDSA_DS_CLK)。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_CRYPTO_KM_CLK_EN 配置是否使能 KM_CLK。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_ISP_CLK_SRC_SEL 配置 ISP_CLK 的时钟源。

0：XTAL_CLK

1：PLL_F240M_CLK

2：PLL_F160M_CLK

3：非法值

(R/W)

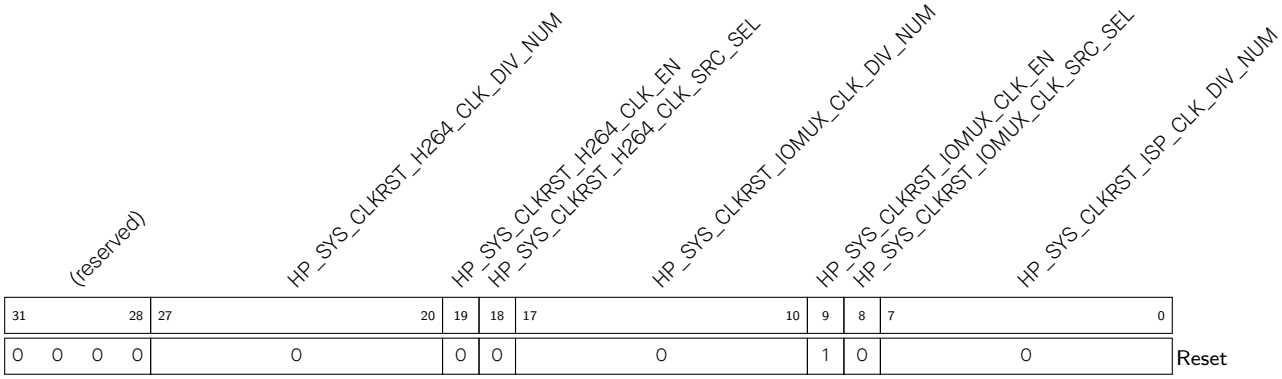
HP_SYS_CLKRST_ISP_CLK_EN 配置是否使能 ISP_CLK。

0：禁用

1：使能

(R/W)

Register 11.44. HP_SYS_CLKRST_PERI_CLK_CTRL26_REG (0x00AC)



HP_SYS_CLKRST_ISP_CLK_DIV_NUM 配置 ISP_CLK 的时钟分频系数。(R/W)

HP_SYS_CLKRST_IOMUX_CLK_SRC_SEL 配置 IOMUX_CLK 的时钟源。

0：XTAL_CLK

1：PLL_F80M_CLK

(R/W)

HP_SYS_CLKRST_IOMUX_CLK_EN 配置是否使能 IOMUX_CLK 时钟。

0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_IOMUX_CLK_DIV_NUM 配置 IOMUX_CLK 的时钟分频系数。(R/W)

HP_SYS_CLKRST_H264_CLK_SRC_SEL 配置 H264_CLK 的时钟源。

0：XTAL_CLK

1：PLL_F240M_CLK

(R/W)

HP_SYS_CLKRST_H264_CLK_EN 配置是否使能 H264_CLK 时钟。

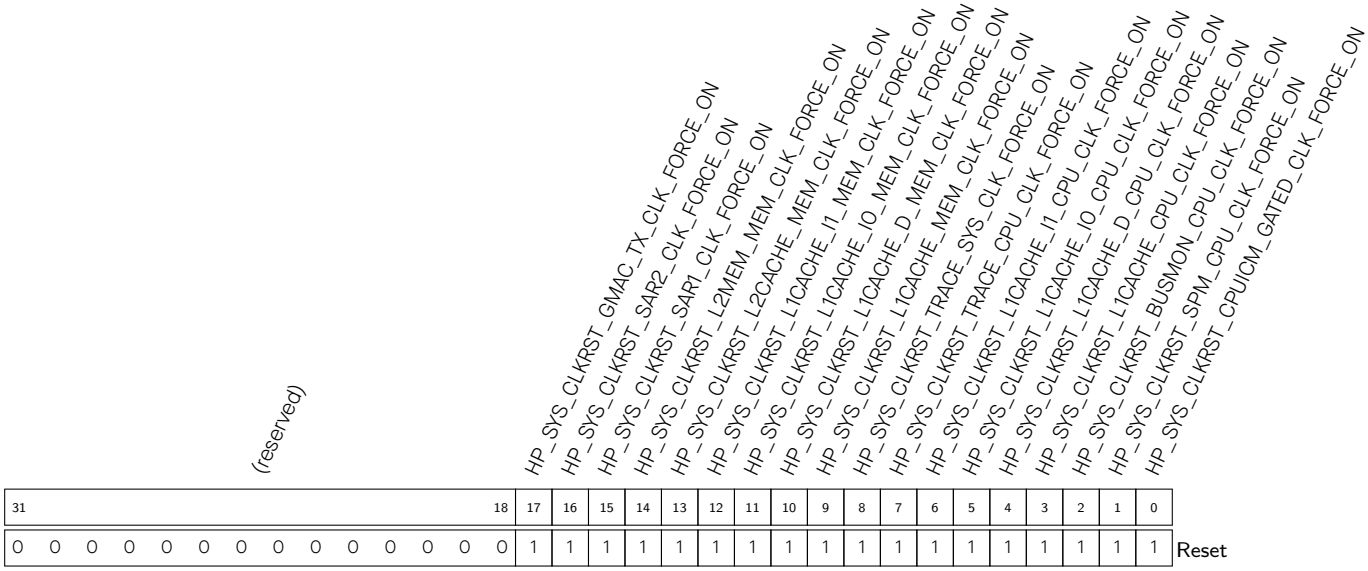
0：禁用

1：使能

(R/W)

HP_SYS_CLKRST_H264_CLK_DIV_NUM 配置 H264_CLK 的时钟分频系数。(R/W)

Register 11.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)



HP_SYS_CLKRST_CPUICM_GATED_CLK_FORCE_ON 配置 CPUICM_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_SPM_CPU_CLK_FORCE_ON 配置 SPM_CPU_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_BUSMON_CPU_CLK_FORCE_ON 配置 BUSMON_CPU_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L1CACHE_CPU_CLK_FORCE_ON 配置 L1CACHE_CPU_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L1CACHE_D_CPU_CLK_FORCE_ON 配置 L1DCACHE_CPU_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L1CACHE_IO_CPU_CLK_FORCE_ON 配置 L1IOCACHE_CPU_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

见下页...

Register 11.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)

接上页...

HP_SYS_CLKRST_L1CACHE_I1_CPU_CLK_FORCE_ON 配置 L1I1CACHE_CPU_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_TRACE_CPU_CLK_FORCE_ON 配置 TRACE_CPU_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_TRACE_SYS_CLK_FORCE_ON 配置 TRACE_SYS_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L1CACHE_MEM_CLK_FORCE_ON 配置 L1CACHE_MEM_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L1CACHE_D_MEM_CLK_FORCE_ON 配置 L1DCACHE_MEM_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L1CACHE_I0_MEM_CLK_FORCE_ON 配置 L1I0CACHE_MEM_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L1CACHE_I1_MEM_CLK_FORCE_ON 配置 L1I1CACHE_MEM_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_L2CACHE_MEM_CLK_FORCE_ON 配置 L2CACHE_MEM_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

见下页...

Register 11.45. HP_SYS_CLKRST_CLK_FORCE_ON_CTRL0_REG (0x00B4)

接上页...

HP_SYS_CLKRST_L2MEM_MEM_CLK_FORCE_ON 配置 L2MEM_MEM_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_SAR1_CLK_FORCE_ON 配置 HPADC1_SARCLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

HP_SYS_CLKRST_SAR2_CLK_FORCE_ON 配置 HPADC2_SARCLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

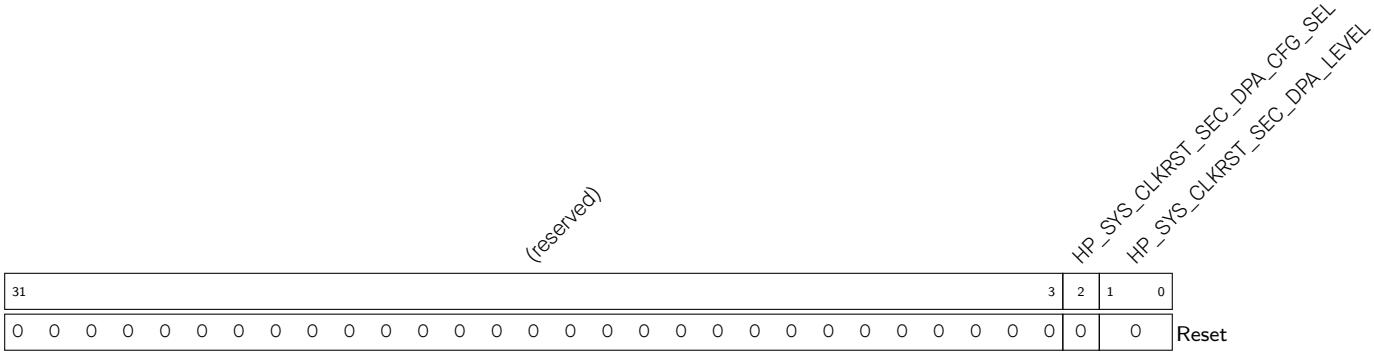
HP_SYS_CLKRST_GMAC_TX_CLK_FORCE_ON 配置 EMAC_TX_CLK 时钟门控路径。

0：时钟门控开启

1：强制打开时钟，旁路时钟门控

(R/W)

Register 11.46. HP_SYS_CLKRST_DPA_CTRL0_REG (0x00B8)



HP_SYS_CLKRST_SEC_DPA_LEVEL 配置 CRYPTO_CLK 的防 DPA 攻击的安全等级。

0：低安全等级

1：高安全等级

(R/W)

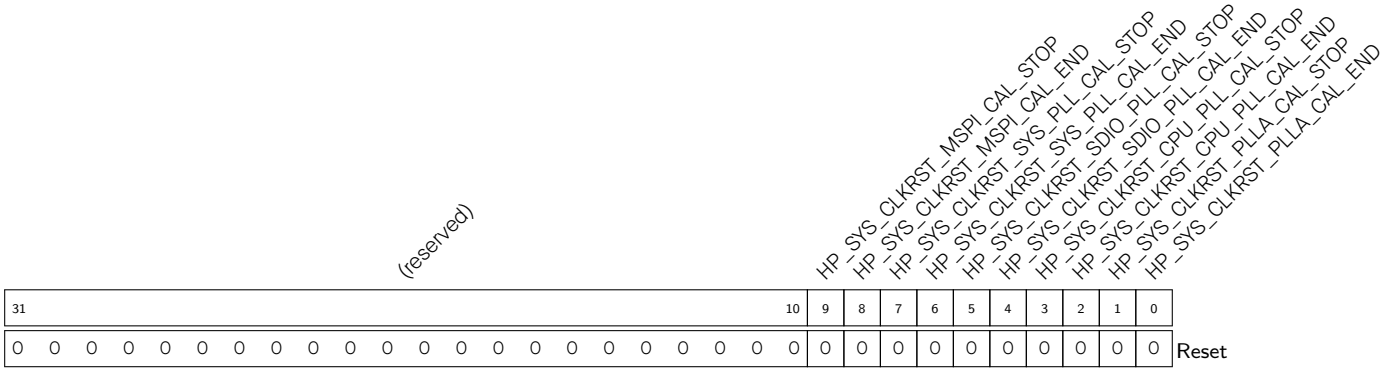
HP_SYS_CLKRST_SEC_DPA_CFG_SEL 配置 CRYPTO_CLK 的防 DPA 攻击的安全等级来源。

0：安全等级由 EFUSE_SEC_DPA_LEVEL 决定

1：安全等级由 HP_SYS_CLKRST_SEC_DPA_LEVEL 决定

(R/W)

Register 11.47. HP_SYS_CLKRST_ANA_PLL_CTRL0_REG (0x00BC)



HP_SYS_CLKRST_PLLA_CAL_END 表示 APLL_CLK 校准结束信号。

0：正在校准

1：校准完成

(RO)

HP_SYS_CLKRST_PLLA_CAL_STOP 配置 APLL_CLK 校准。

0：开始校准

1：停止校准

(R/W)

HP_SYS_CLKRST_CPU_PLL_CAL_END 表示 CPLL_CLK 校准结束信号。

0：正在校准

1：校准完成

(RO)

HP_SYS_CLKRST_CPU_PLL_CAL_STOP 配置 CPLL_CLK 校准。

0：开始校准

1：停止校准

(R/W)

HP_SYS_CLKRST_SDIO_PLL_CAL_END 表示 SDIO_PLL 校准结束信号。

0：正在校准

1：校准完成

(RO)

HP_SYS_CLKRST_SDIO_PLL_CAL_STOP 配置 SDIO_PLL 校准。

0：开始校准

1：停止校准

(R/W)

见下页...

Register 11.47. HP_SYS_CLKRST_ANA_PLL_CTRL0_REG (0x00BC)

接上页...

HP_SYS_CLKRST_SYS_PLL_CAL_END 表示 SPLL_CLK 校准结束信号。

0：正在校准

1：校准完成

(RO)

HP_SYS_CLKRST_SYS_PLL_CAL_STOP 配置 SPLL_CLK 校准。

0：开始校准

1：停止校准

(R/W)

HP_SYS_CLKRST_MSPI_CAL_END 表示 MPLL_CLK 校准结束信号。

0：正在校准

1：校准完成

(RO)

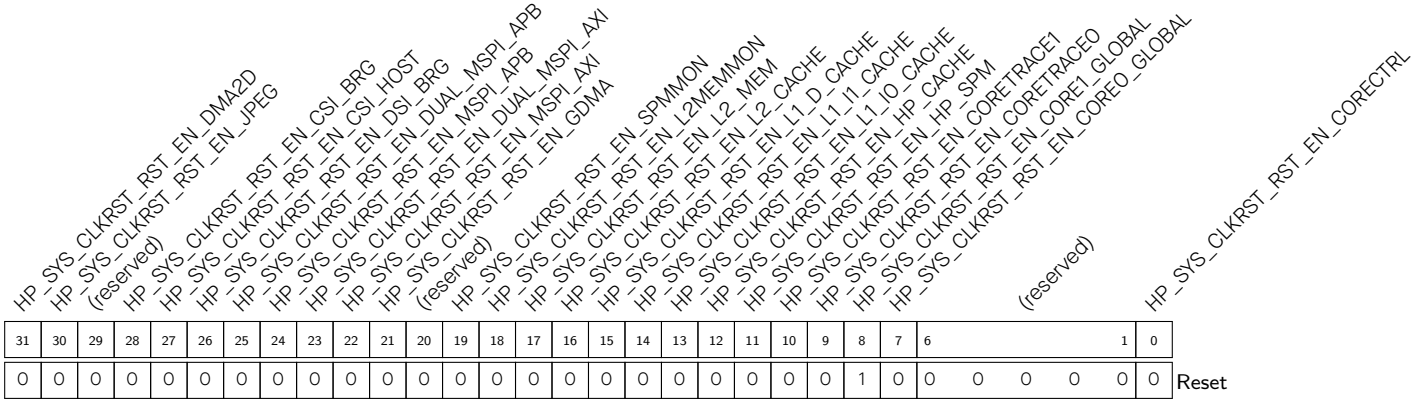
HP_SYS_CLKRST_MSPI_CAL_STOP 配置 MPLL_CLK 校准。

0：开始校准

1：停止校准

(R/W)

Register 11.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)



HP_SYS_CLKRST_RST_EN_CORECTRL 配置是否复位 CORECTRL。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CORE0_GLOBAL 配置是否复位 HP CPU0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CORE1_GLOBAL 配置是否复位 HP CPU1。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CORETRACE0 配置是否复位 RISC-V 追踪编码器 0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CORETRACE1 配置是否复位 RISC-V 追踪编码器 1。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_HP_SPM 配置是否复位 SPM。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)

接上页...

HP_SYS_CLKRST_RST_EN_HP_CACHE 配置是否复位 L1 I-Cache0、L1 I-Cache1、L1 D-Cache、L2 Cache。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_L1_I0_CACHE 配置是否复位 L1 I-Cache0。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_L1_I1_CACHE 配置是否复位 L1 I-Cache1。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_L1_D_CACHE 配置是否复位 L1 D-Cache。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_L2_CACHE 配置是否复位 L2 Cache。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_L2_MEM 配置是否复位 L2 MEM。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_L2MEMMON 配置是否复位 L2 MEM 监控。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_SPMMON 配置是否复位 SPM 监控。
0：复位释放
1：复位
(R/W)

HP_SYS_CLKRST_RST_EN_GDMA 配置是否复位 VDMA。
0：复位释放
1：复位
(R/W)

见下页...

Register 11.48. HP_SYS_CLKRST_HP_RST_EN0_REG (0x00C0)

接上页...

HP_SYS_CLKRST_RST_EN_MSPI_AXI 配置是否复位 FLASH MSPI AXI。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_DUAL_MSPI_AXI 配置是否复位 PSRAM MSPI AXI。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_MSPI_APB 配置是否复位 FLASH MSPI APB。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_DUAL_MSPI_APB 配置是否复位 PSRAM MSPI APB。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_DSI_BRG 配置是否复位 DSI Bridge。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CSI_HOST 配置是否复位 CSI Host。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CSI_BRG 配置是否复位 CSI Bridge 和 ISP。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_JPEG 配置是否复位 JPEG。

0：复位释放

1：复位

(R/W)

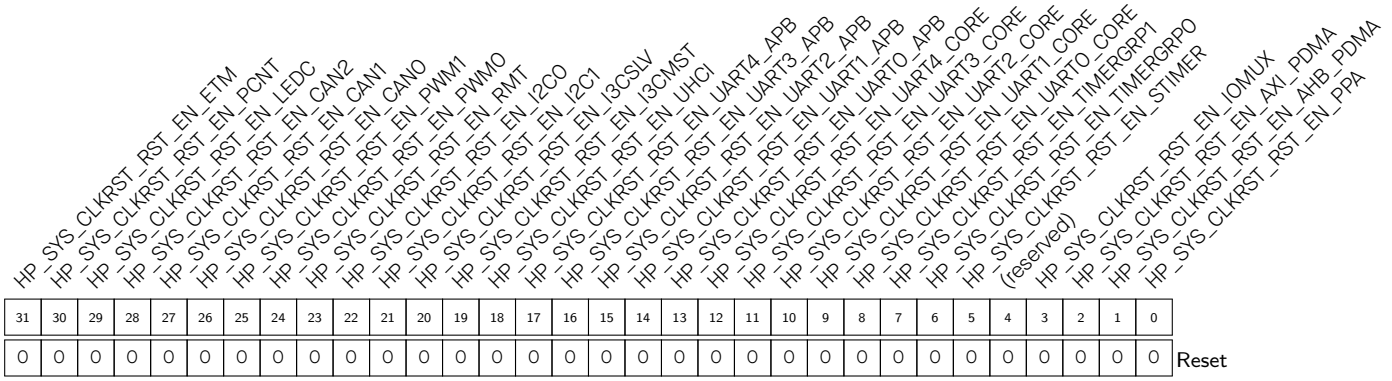
HP_SYS_CLKRST_RST_EN_DMA2D 配置是否复位 2DDMA。

0：复位释放

1：复位

(R/W)

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)



HP_SYS_CLKRST_RST_EN_PPA 配置是否复位 PPA。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_AHB_PDMA 配置是否复位 GDMA-AHB。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_AXI_PDMA 配置是否复位 GDMA-AXI。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_IOMUX 配置是否复位 IO MUX。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_STIMER 配置是否复位系统定时器。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_TIMERGRP0 配置是否复位定时器组 0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_TIMERGRP1 配置是否复位定时器组 1。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

接上页...

HP_SYS_CLKRST_RST_EN_UART0_CORE 配置是否复位 UART0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART1_CORE 配置是否复位 UART1。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART2_CORE 配置是否复位 UART2。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART3_CORE 配置是否复位 UART3。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART4_CORE 配置是否复位 UART4。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART0_APB 配置是否复位 UART0 APB。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART1_APB 配置是否复位 UART1 APB。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART2_APB 配置是否复位 UART2 APB。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UART3_APB 配置是否复位 UART3 APB。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

接上页...

HP_SYS_CLKRST_RST_EN_UART4_APB 配置是否复位 UART4 APB。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_UHCI 配置是否复位 UHCI。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_I3CMST 配置是否复位 I3C 主机。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_I3CSLV 配置是否复位 I3C 从机。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_I2C1 配置是否复位 I2C1。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_I2C0 配置是否复位 I2C0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_RMT 配置是否复位 RMT。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_PWM0 配置是否复位 MCPMW0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_PWM1 配置是否复位 MCPWM1。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.49. HP_SYS_CLKRST_HP_RST_EN1_REG (0x00C4)

接上页...

HP_SYS_CLKRST_RST_EN_CAN0 配置是否复位 TWAI0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CAN1 配置是否复位 TWAI。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CAN2 配置是否复位 TWAI。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_LEDC 配置是否复位 LEDC。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_PCNT 配置是否复位 PCNT。

0：复位释放

1：复位

(R/W)

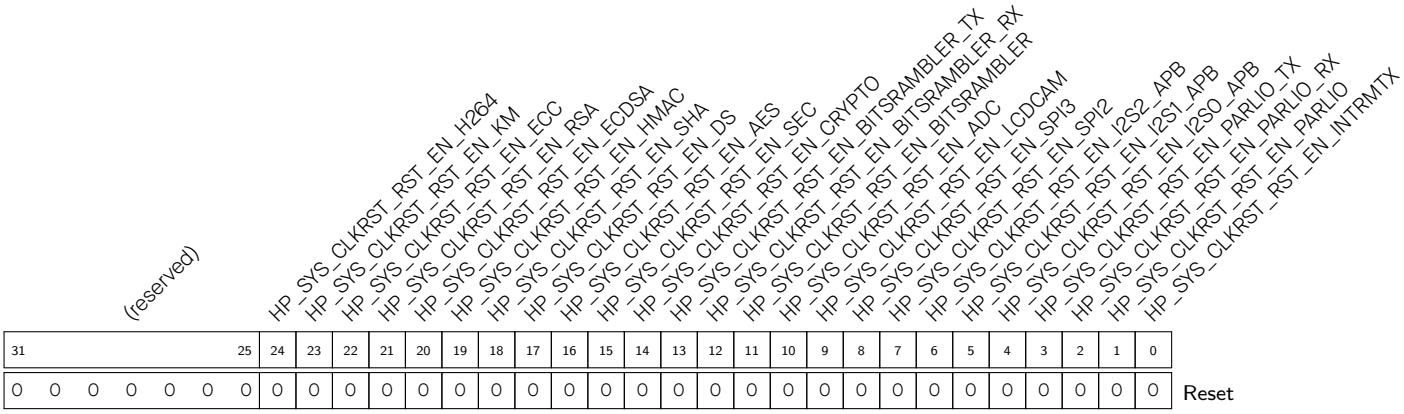
HP_SYS_CLKRST_RST_EN_ETM 配置是否复位 ETM。

0：复位释放

1：复位

(R/W)

Register 11.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)



HP_SYS_CLKRST_RST_EN_INTRMTX 配置是否复位中断矩阵。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO 配置是否复位 PARLIO。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO_RX 配置是否复位 PARLIO RX。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_PARLIO_TX 配置是否复位 PARLIO TX。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_I2S0_APB 配置是否复位 I2S0。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_I2S1_APB 配置是否复位 I2S1。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_I2S2_APB 配置是否复位 I2S2。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)

接上页...

HP_SYS_CLKRST_RST_EN_SPI2 配置是否复位 SPI2。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_SPI3 配置是否复位 SPI3。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_LCDCAM 配置是否复位 LCD_CAM。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_ADC 配置是否复位 ADC。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER 配置是否复位比特传输控制器。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER_RX 配置是否复位比特传输控制器 RX。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_BITSRAMBLER_TX 配置是否复位比特传输控制器 TX。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_CRYPTO 配置是否复位加密模块。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_SEC 配置是否复位 SEC。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.50. HP_SYS_CLKRST_HP_RST_EN2_REG (0x00C8)

接上页...

HP_SYS_CLKRST_RST_EN_AES 配置是否复位 AES。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_DS 配置是否复位 RSA_DS 外设。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_SHA 配置是否复位 SHA。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_HMAC 配置是否复位 HMAC。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_ECDSA 配置是否复位 ECDSA_DS。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_RSA 配置是否复位 RSA。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_ECC 配置是否复位 ECC。

0：复位释放

1：复位

(R/W)

HP_SYS_CLKRST_RST_EN_KM 配置是否复位密钥管理器。

0：复位释放

1：复位

(R/W)

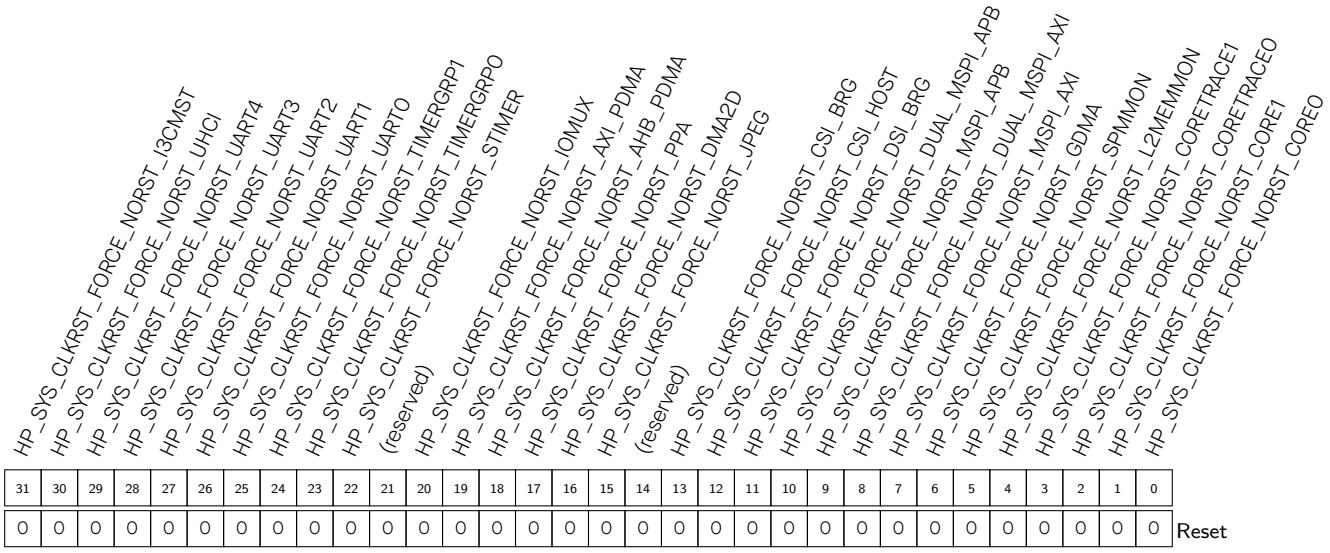
HP_SYS_CLKRST_RST_EN_H264 配置是否复位 H264。

0：复位释放

1：复位

(R/W)

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)



HP_SYS_CLKRST_FORCE_NORST_CORE0 配置 HP CPU0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORE1 配置 HP CPU1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORETRACE0 配置 RISC-V 追踪编码器 0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CORETRACE1 配置 RISC-V 追踪编码器 1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_L2MEMMON 配置 L2 MEM 监控是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPMMON 配置 SPM 监控是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_GDMA 配置 VDMA 是否可复位。

0：可复位

1：强制不可复位

(R/W)

见下页...

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

接上页...

HP_SYS_CLKRST_FORCE_NORST_MSPI_AXI 配置 FLASH MSPI 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DUAL_MSPI_AXI 配置 PSRAM MSPI 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_MSPI_APB 配置 FLASH MSPI APB 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DUAL_MSPI_APB 配置 PSRAM MSPI APB 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DSI_BRG 配置 DSI Bridge 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CSI_HOST 配置 CSI Host 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CSI_BRG 配置 CSI Bridge、ISP 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_JPEG 配置 JPEG 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_DMA2D 配置 2DDMA 是否可复位。

0：可复位

1：强制不可复位

(R/W)

见下页...

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORST0_REG (0x00CC)

接上页...

HP_SYS_CLKRST_FORCE_NORST_PPA 配置 PPA 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_AHB_PDMA 配置 GDMA-AHB 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_AXI_PDMA 配置 GDMA-AXI 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_IOMUX 配置 IO MUX 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_STIMER 配置系统定时器是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_TIMERGRP0 配置定时器组 0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_TIMERGRP1 配置定时器组 1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART0 配置 UART0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

见下页...

Register 11.51. HP_SYS_CLKRST_HP_FORCE_NORSTO_REG (0x00CC)

接上页...

HP_SYS_CLKRST_FORCE_NORST_UART1 配置 UART1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART2 配置 UART2 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART3 配置 UART3 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UART4 配置 UART4 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_UHCI 配置 UHCI 是否可复位。

0：可复位

1：强制不可复位

(R/W)

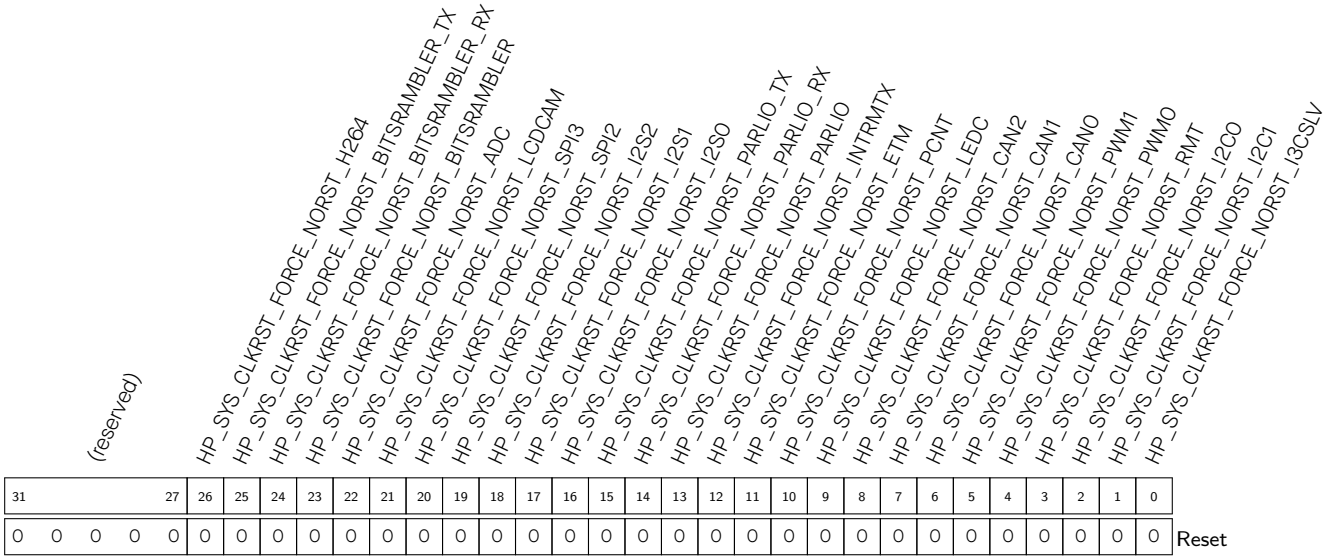
HP_SYS_CLKRST_FORCE_NORST_I3CMST 配置 I3C 主机是否可复位。

0：可复位

1：强制不可复位

(R/W)

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)



HP_SYS_CLKRST_FORCE_NORST_I3CSLV 配置 I3C 从机是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2C1 配置 I2C1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2C0 配置 I2C0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_RMT 配置 RMT 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PWM0 配置 MCPWM0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PWM1 配置 MCPWM1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CAN0 配置 TWAI0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

见下页...

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

接上页...

HP_SYS_CLKRST_FORCE_NORST_CAN1 配置 TWAI1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_CAN2 配置 TWAI2 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_LEDC 配置 LEDC 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PCNT 配置 PCNT 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_ETM 配置 ETM 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_INTRMTX 配置中断矩阵是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO 配置 PARLIO 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO_RX 配置 PARLIO RX 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_PARLIO_TX 配置 PARLIO TX 是否可复位。

0：可复位

1：强制不可复位

(R/W)

见下页...

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

接上页...

HP_SYS_CLKRST_FORCE_NORST_I2S0 配置 I2S0 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2S1 配置 I2S1 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_I2S2 配置 I2S2 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPI2 配置 SPI2 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_SPI3 配置 SPI3 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_LCDCAM 配置 LCD_CAM 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_ADC 配置 ADC 是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER 配置比特传输控制器是否可复位。

0：可复位

1：强制不可复位

(R/W)

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER_RX 配置比特传输控制器 RX 是否可复位。

0：可复位

1：强制不可复位

(R/W)

见下页...

Register 11.52. HP_SYS_CLKRST_HP_FORCE_NORST1_REG (0x00D0)

接上页...

HP_SYS_CLKRST_FORCE_NORST_BITSRAMBLER_TX 配置比特传输控制器 TX 是否可复位。

0：可复位

1：强制不可复位

(R/W)

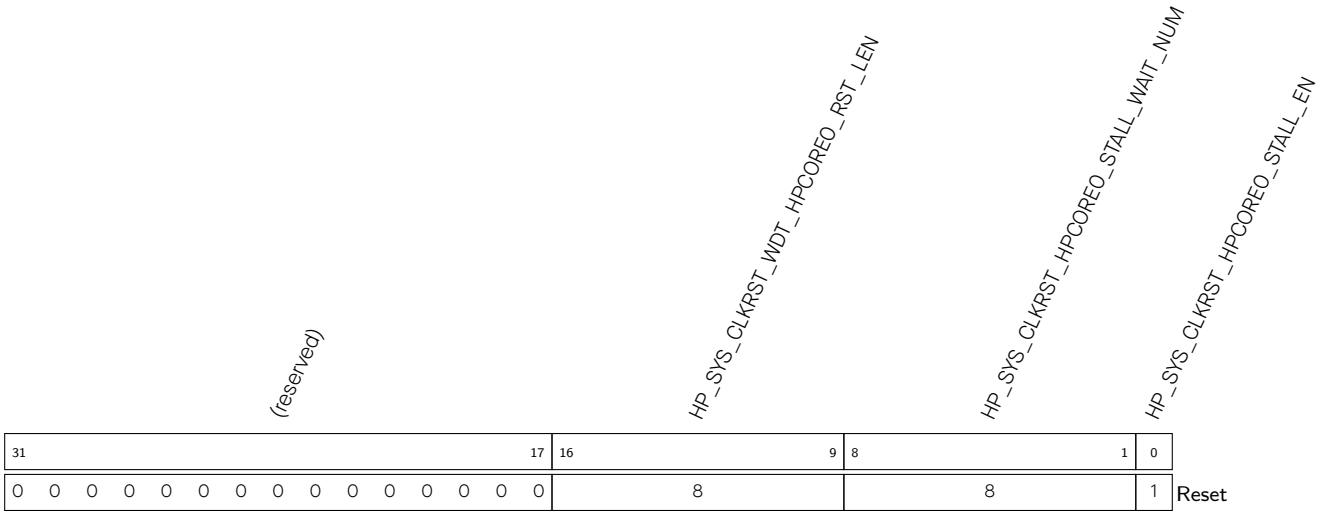
HP_SYS_CLKRST_FORCE_NORST_H264 配置 H264 是否可复位。

0：可复位

1：强制不可复位

(R/W)

Register 11.53. HP_SYS_CLKRST_HPWDT_CORE0_RST_CTRL0_REG (0x00D4)



HP_SYS_CLKRST_HPCORE0_STALL_EN 配置 MWDT 触发 HP CPU0 复位的行为。

0：立刻复位 HP CPU0

1：先暂停 (Stall) HP CPU0

(R/W)

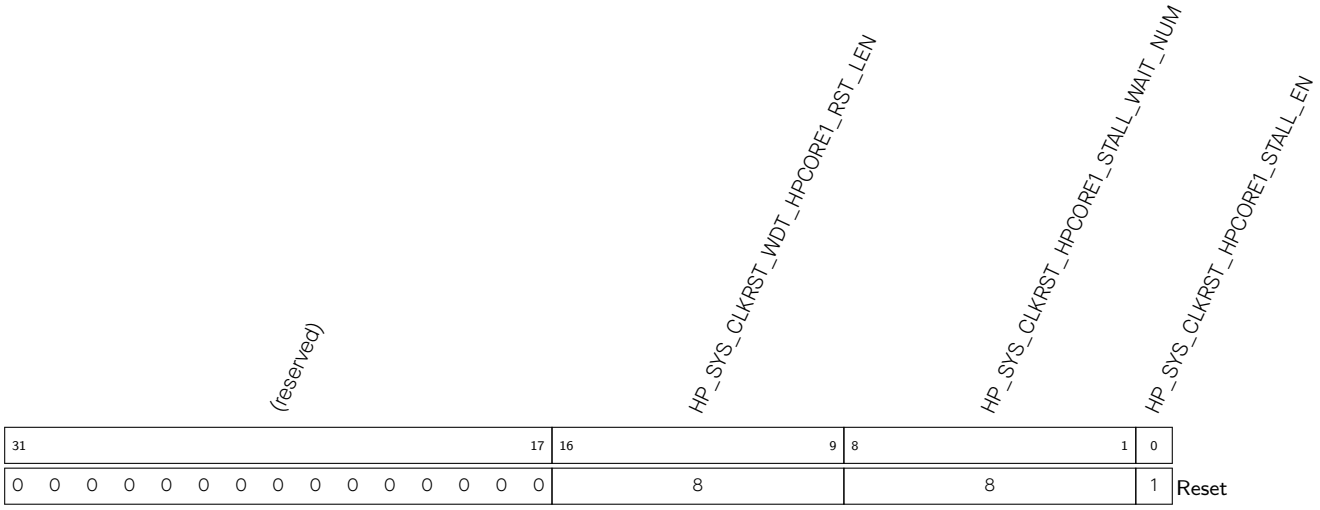
HP_SYS_CLKRST_HPCORE0_STALL_WAIT_NUM 配置 MWDT 触发 HP CPU0 复位且 HP_SYS_CLKRST_HPCORE0_STALL_EN 置 1 时，HP CPU0 暂停 (Stall) 的时长。

单位：时钟周期。(R/W)

HP_SYS_CLKRST_WDT_HPCORE0_RST_LEN 配置 MWDT 触发 HP CPU0 复位时，复位持续的时长。

单位：时钟周期。(R/W)

Register 11.54. HP_SYS_CLKRST_HPWDT_CORE1_RST_CTRL0_REG (0x00D8)



HP_SYS_CLKRST_HPCORE1_STALL_EN 配置 MWDT 触发 HP CPU1 复位的行为。

0：立刻复位 HP CPU1

1：先暂停 (Stall) HP CPU1

(R/W)

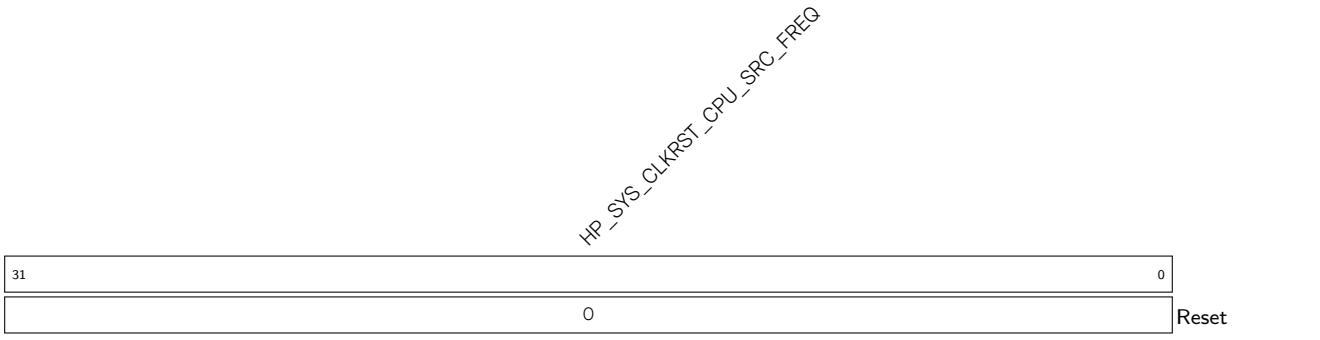
HP_SYS_CLKRST_HPCORE1_STALL_WAIT_NUM 配置 MWDT 触发 HP CPU1 复位且 HP_SYS_CLKRST_HPCORE1_STALL_EN 置 1 时，HP CPU1 暂停 (Stall) 的时长。

单位：时钟周期。(R/W)

HP_SYS_CLKRST_WDT_HPCORE1_RST_LEN 配置 MWDT 触发 HP CPU1 复位时，复位持续的时长。

单位：时钟周期。(R/W)

Register 11.55. HP_SYS_CLKRST_CPU_SRC_FREQ0_REG (0x00DC)



HP_SYS_CLKRST_CPU_SRC_FREQ 表示 ROOT_CLK 的源时钟。

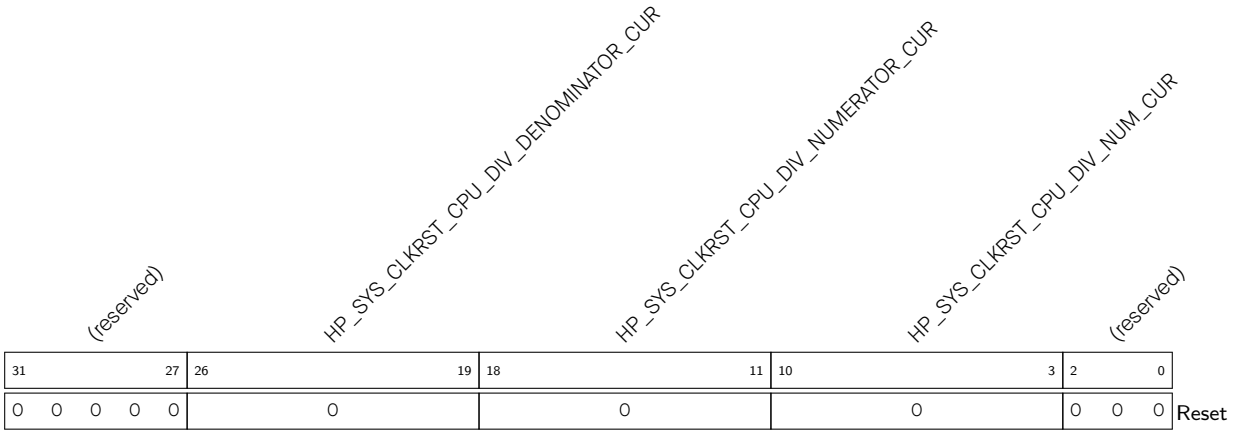
80：RC_FAST_CLK

160：XTAL_CLK

1600：CPLL_CLK

(RO)

Register 11.56. HP_SYS_CLKRST_CPU_CLK_STATUS0_REG (0x00E0)

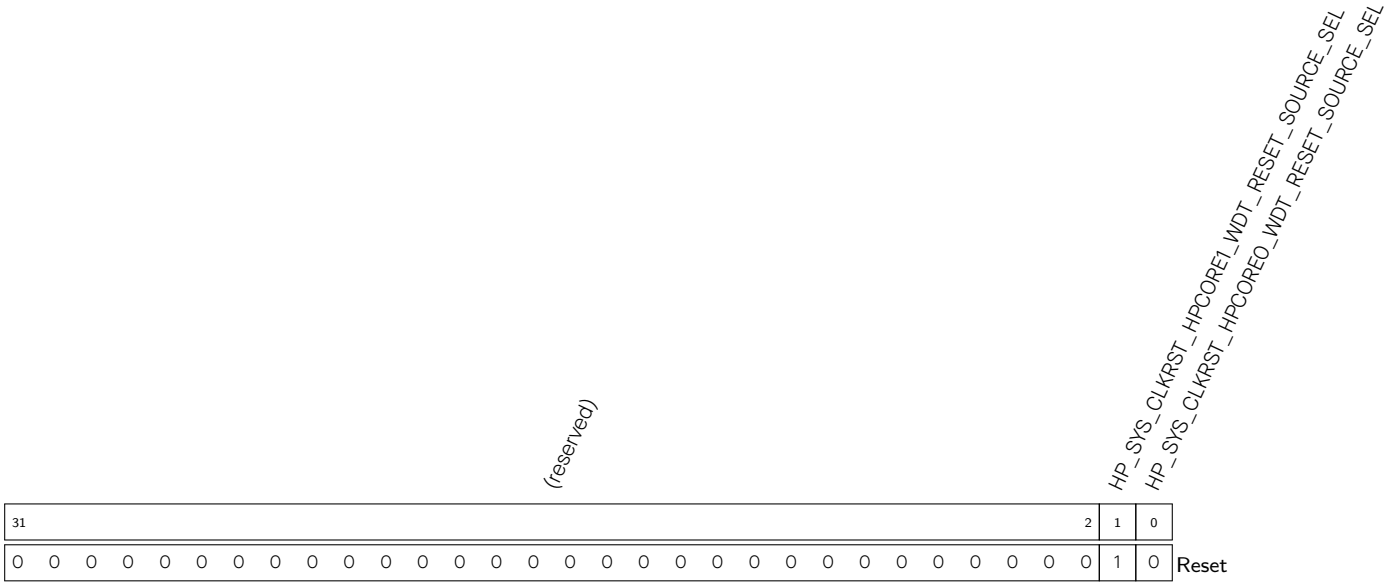


HP_SYS_CLKRST_CPU_DIV_NUM_CUR 表示当前 CPU_CLK 分频系数的整数部分。(RO)

HP_SYS_CLKRST_CPU_DIV_NUMERATOR_CUR 表示当前 CPU_CLK 分频系数小数部分的分子。(RO)

HP_SYS_CLKRST_CPU_DIV_DENOMINATOR_CUR 表示当前 CPU_CLK 分频系数小数部分的分母。(RO)

Register 11.57. HP_SYS_CLKRST_HPCORE_WDT_RESET_SOURCE0_REG (0x00EC)



HP_SYS_CLKRST_HPCORE0_WDT_RESET_SOURCE_SEL 配置用于触发 HP CPU0 复位的 MWDT。

0：MWDT0

1：MWDT1

(R/W)

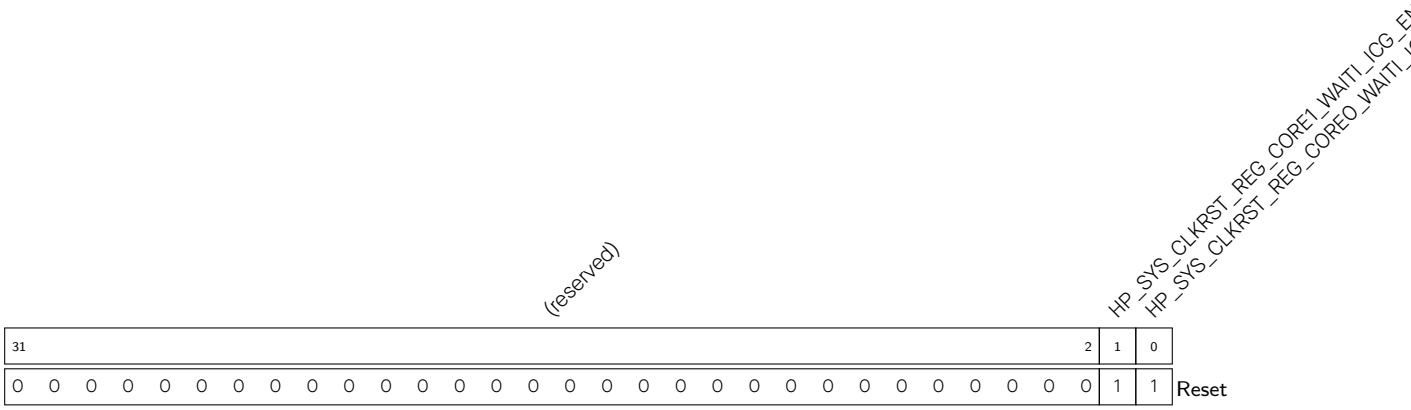
HP_SYS_CLKRST_HPCORE1_WDT_RESET_SOURCE_SEL 配置用于触发 HP CPU1 复位的 MWDT。

0：MWDT0

1：MWDT1

(R/W)

Register 11.58. HP_SYS_CLKRST_CPU_WAITI_CTRL0_REG (0x00F4)



HP_SYS_CLKRST_REG_CORE0_WAITI_ICG_EN 配置 CPU Core0 的 WAITI 信号是否控制时钟门控。

0：CPU Core0 的 WAITI 信号不用于控制时钟

1：CPU Core0 的 WAITI 信号控制时钟

如果 Core0 和 Core1 的该位均为 1，则仅当 Core0 和 Core1 的 WAITI 信号均有效时才会关闭相关时钟。

(R/W)

HP_SYS_CLKRST_REG_CORE1_WAITI_ICG_EN 配置 CPU Core1 的 WAITI 信号是否控制时钟门控。

0：CPU Core1 的 WAITI 信号不用于控制时钟

1：CPU Core1 的 WAITI 信号控制时钟

如果 Core0 和 Core1 的该位均为 1，则仅当 Core0 和 Core1 的 WAITI 信号均有效时才会关闭相关时钟。

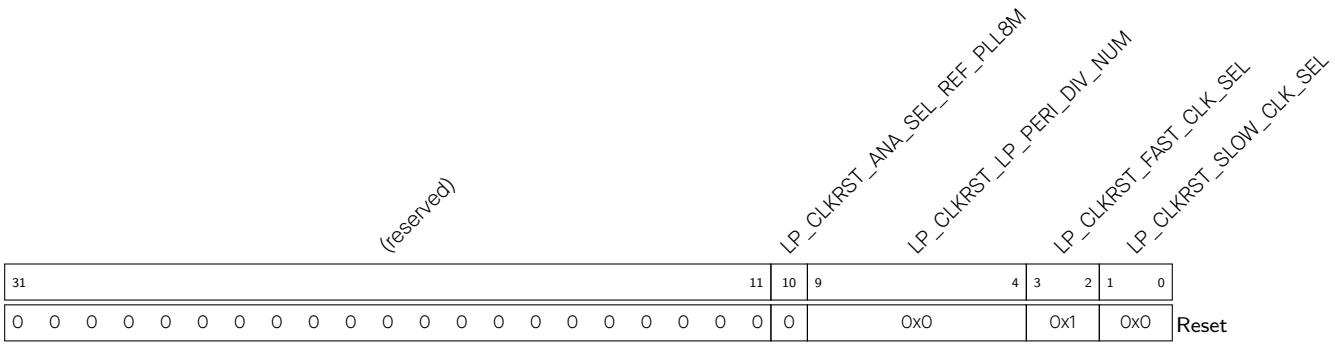
(R/W)

11.5.2 LP 常开时钟与复位 (LP_CLKRST) 寄存器

本小节的所有地址均为相对于 LP_CLKRST 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 11.59. LP_CLKRST_LP_CLK_CONF_REG (0x0000)



LP_CLKRST_SLOW_CLK_SEL 配置 LP_SLOW_CLK 的时钟源。

0：RC_SLOW_CLK

1：XTAL32K_CLK

2：无效值

3：OSC_SLOW_CLK

(R/W)

LP_CLKRST_FAST_CLK_SEL 配置 LP_FAST_CLK 的时钟源。

0：RC_FAST_CLK

1：XTAL_CLK

2：PLL_LP_CLK

3：非法值

(R/W)

LP_CLKRST_LP_PERI_DIV_NUM 配置 LP_PERI_CLK 的分频系数。(R/W)

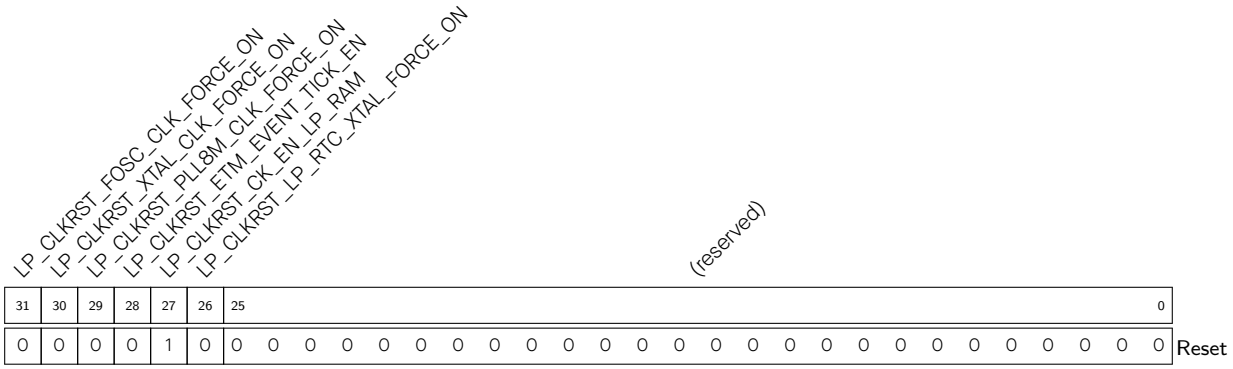
LP_CLKRST_ANA_SEL_REF_PLL8M 配置 PLL_LP_CLK 的参考时钟。

0：无效值

1：XTAL32K_CLK

(R/W)

Register 11.60. LP_CLKRST_LP_CLK_EN_REG (0x0008)



LP_CLKRST_LP_RTC_XTAL_FORCE_ON 配置是否强制打开 LP CALI 的 XTAL_CLK 控制。

0：交由硬件控制

1：强制打开，旁路硬件控制

(R/W)

LP_CLKRST_CK_EN_LP_RAM 配置是否使能 LP RAM 的时钟。

0：禁用

1：使能

(R/W)

LP_CLKRST_ETM_EVENT_TICK_EN 配置是否使能 ETM 事件计时器。

0：禁用

1：使能

(R/W)

LP_CLKRST_PLL8M_CLK_FORCE_ON 配置是否强制打开 LP_FAST_CLK 源头的 PLL_LP_CLK。

0：交由硬件控制

1：强制打开，旁路硬件控制

(R/W)

LP_CLKRST_XTAL_CLK_FORCE_ON 配置是否强制打开 LP_FAST_CLK 源头的 XTAL_CLK。

0：交由硬件控制

1：强制打开，旁路硬件控制

(R/W)

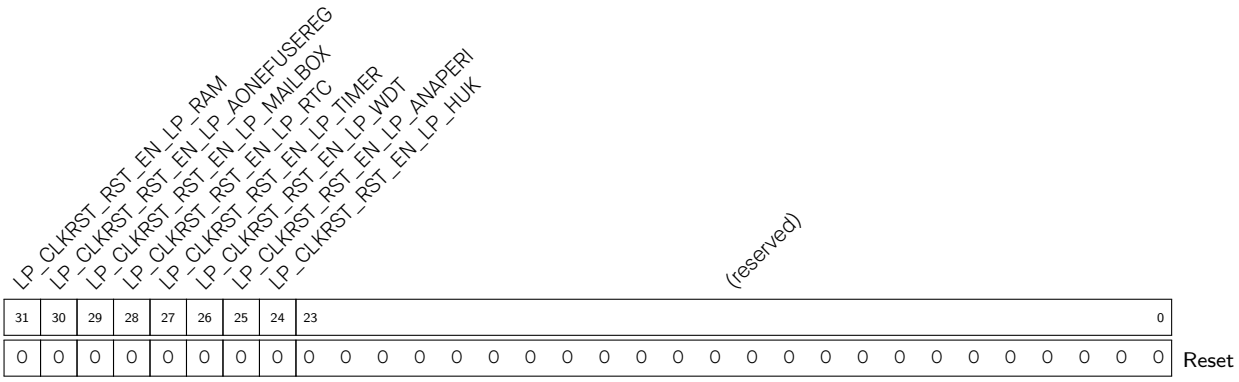
LP_CLKRST_FOSC_CLK_FORCE_ON 配置是否强制打开 LP_FAST_CLK 源头的 RC_FAST_CLK。

0：交由硬件控制

1：强制打开，旁路硬件控制

(R/W)

Register 11.61. LP_CLKRST_LP_RST_EN_REG (0x000C)



LP_CLKRST_RST_EN_LP_HUK 配置是否复位 LP HUK。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_LP_ANAPERI 配置是否复位 LP 模拟外设。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_LP_WDT 配置是否复位 RWDT。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_LP_TIMER 配置是否复位 LP 定时器。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_LP_RTC 配置是否复位 LP 时钟校准模块。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_LP_MAILBOX 配置是否复位 LP 信箱控制器。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.61. LP_CLKRST_LP_RST_EN_REG (0x000C)

接上页...

LP_CLKRST_RST_EN_LP_AONEFUSEREG 配置是否复位 LP eFuse。

0：复位释放

1：复位

(R/W)

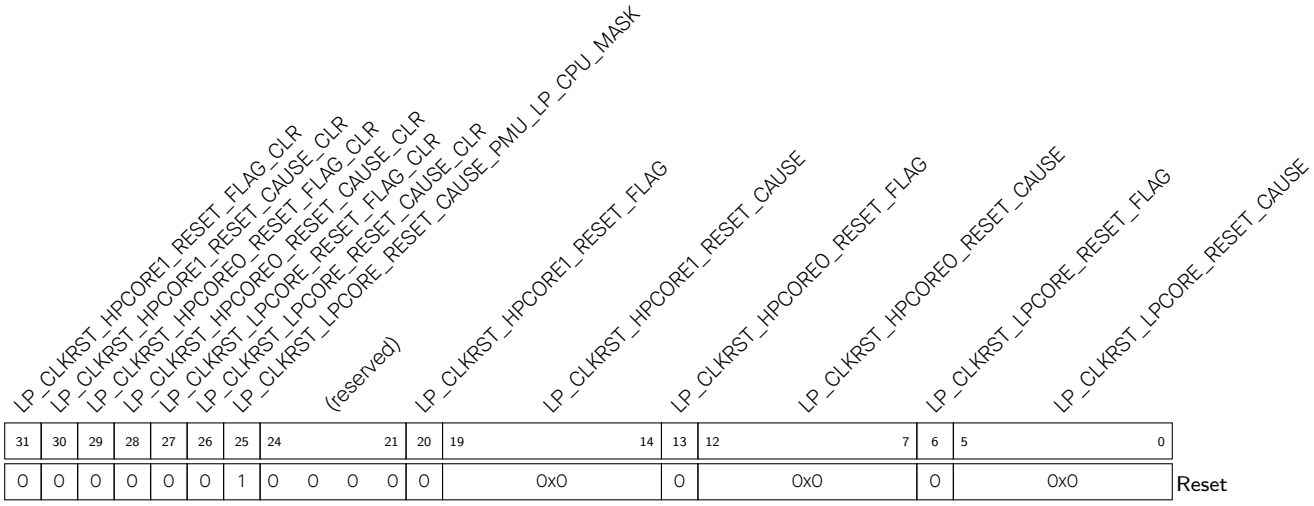
LP_CLKRST_RST_EN_LP_RAM 配置是否复位 LP RAM。

0：复位释放

1：复位

(R/W)

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)



LP_CLKRST_LPCORE_RESET_CAUSE 表示 LP CPU 复位源。

0x01：芯片复位

0x09：PMU LP PERI 复位

0x0A：PMU LP CPU 复位

0x0F：欠压系统复位

0x10：RWDT 系统复位

0x12：超级看门狗复位

0x13：电源毛刺复位

0x14：软件 LP CPU 复位

(RO)

LP_CLKRST_LPCORE_RESET_FLAG 表示 LP CPU 复位标志。

0：未复位

1：已复位

(RO)

见下页...

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)

接上页...

LP_CLKRST_HPCORE0_RESET_CAUSE 表示 HP CPU0 复位源。

0x01：芯片复位
0x03：系统软件复位
0x05：PMU 内核复位
0x07：MWDT 内核复位
0x09：RWDT 内核复位
0x0B：MWDT CPU 复位
0x0C：软件 CPU 复位
0x0D：RWDT CPU 复位
0x0F：欠压系统复位
0x10：RWDT 系统复位
0x12：超级看门狗复位
0x13：电源毛刺复位
0x14：eFuse 复位
0x16：USB (JTAG) 复位
0x17：USB (UART) 复位
0x18：JTAG CPU 复位
0x1A：Lockup 复位
0x1B：PSDET 复位
(RO)

LP_CLKRST_HPCORE0_RESET_FLAG 表示 HP CPU0 复位标志。

0：未复位
1：已复位
(RO)

见下页...

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)

接上页...

LP_CLKRST_HPCORE1_RESET_CAUSE 表示 HP CPU1 复位源。

0x01：芯片复位
0x03：系统软件复位
0x05：PMU 内核复位
0x07：MWDT 内核复位
0x09：RWDT 内核复位
0x0B：MWDT CPU 复位
0x0C：软件 CPU 复位
0x0D：RWDT CPU 复位
0x0F：欠压系统复位
0x10：RWDT 系统复位
0x12：超级看门狗复位
0x13：电源毛刺复位
0x14：eFuse 复位
0x16：USB (JTAG) 复位
0x17：USB (UART) 复位
0x18：JTAG CPU 复位
0x1A：Lockup 复位
(RO)

LP_CLKRST_HPCORE1_RESET_FLAG 表示 HP CPU1 复位标志。

0：未复位
1：已复位
(RO)

LP_CLKRST_LPCORE_RESET_CAUSE_PMU_LP_CPU_MASK 配置是否屏蔽 PMU LP CPU 复位的复位源代码 0x0A。

0：不屏蔽
1：屏蔽
(R/W)

LP_CLKRST_LPCORE_RESET_CAUSE_CLR 写 1 清除 LP_CLKRST_LPCORE_RESET_CAUSE。(WT)

LP_CLKRST_LPCORE_RESET_FLAG_CLR 写 1 清除 LP_CLKRST_LPCORE_RESET_FLAG。(WT)

见下页...

Register 11.62. LP_CLKRST_RESET_CAUSE_REG (0x0010)

接上页...

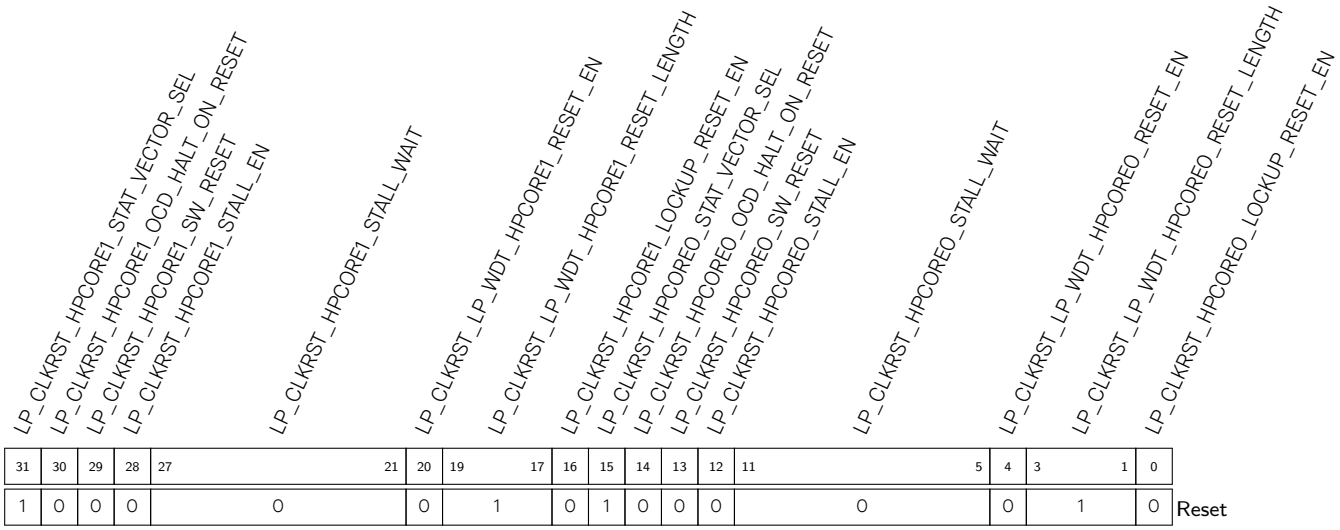
LP_CLKRST_HPCORE0_RESET_CAUSE_CLR 写 1 清除 LP_CLKRST_HPCORE0_RESET_CAUSE。(WT)

LP_CLKRST_HPCORE0_RESET_FLAG_CLR 写 1 清除 LP_CLKRST_HPCORE0_RESET_FLAG。(WT)

LP_CLKRST_HPCORE1_RESET_CAUSE_CLR 写 1 清除 LP_CLKRST_HPCORE1_RESET_CAUSE。(WT)

LP_CLKRST_HPCORE1_RESET_FLAG_CLR 写 1 清除 LP_CLKRST_HPCORE1_RESET_FLAG。(WT)

Register 11.63. LP_CLKRST_HPCPU_RESET_CTRL0_REG (0x0014)



LP_CLKRST_HPCORE0_LOCKUP_RESET_EN 配置是否使能 Lockup 触发 HP CPU0 复位。

0：禁用

1：使能

(R/W)

LP_CLKRST_LP_WDT_HPCORE0_RESET_LENGTH 配置 RWDT 和软件触发 HP CPU0 复位时，复位的持续时间。

单位：时钟周期。(R/W)

LP_CLKRST_LP_WDT_HPCORE0_RESET_EN 配置是否使能 RWDT 触发 HP CPU0 复位。

0：禁用

1：使能

(R/W)

LP_CLKRST_HPCORE0_STALL_WAIT 配置 RWDT 和软件触发 HP CPU0 复位时，HP CPU0 暂停 (Stall) 的时间。

单位：时钟周期。(R/W)

LP_CLKRST_HPCORE0_STALL_EN 配置 RWDT 和软件触发 HP CPU0 复位的行为。

0：立刻复位 HP CPU0

1：先暂停 (Stall) HP CPU0

(R/W)

LP_CLKRST_HPCORE0_SW_RESET 写 1 触发 HP CPU0 软件复位。(WT)

LP_CLKRST_HPCORE0_OCD_HALT_ON_RESET 配置 HP CPU0 是否在复位释放后第一条指令停住。

0：不停

1：停住

(R/W)

LP_CLKRST_HPCORE0_STAT_VECTOR_SEL 配置 HP CPU0 启动的地址。

0：从 LP SPM RAM (0x50108000) 启动

1：从 HP SPM ROM (0x4FC00000) 启动

(R/W)

见下页...

Register 11.63. LP_CLKRST_HPCPU_RESET_CTRL0_REG (0x0014)

接上页...

LP_CLKRST_HPCORE1_LOCKUP_RESET_EN 配置是否使能 Lockup 触发 HP CPU1 复位。

0：禁用

1：使能

(R/W)

LP_CLKRST_LP_WDT_HPCORE1_RESET_LENGTH 配置 RWDT 和软件触发 HP CPU1 复位时，复位的持续时间。

单位：时钟周期。(R/W)

LP_CLKRST_LP_WDT_HPCORE1_RESET_EN 配置是否使能 RWDT 触发 HP CPU1 复位。

0：禁用

1：使能

(R/W)

LP_CLKRST_HPCORE1_STALL_WAIT 配置 RWDT 和软件触发 HP CPU1 复位时，HP CPU0 暂停 (Stall) 的时间。

单位：时钟周期。(R/W)

LP_CLKRST_HPCORE1_STALL_EN 配置 RWDT 和软件触发 HP CPU1 复位的行为。

0：立刻复位 HP CPU1

1：先暂停 (Stall) HP CPU1

(R/W)

LP_CLKRST_HPCORE1_SW_RESET 写 1 触发 HP CPU1 软件复位。(WT)

LP_CLKRST_HPCORE1_OCD_HALT_ON_RESET 配置 HP CPU1 是否在复位释放后第一条指令停住。

0：不停

1：停住

(R/W)

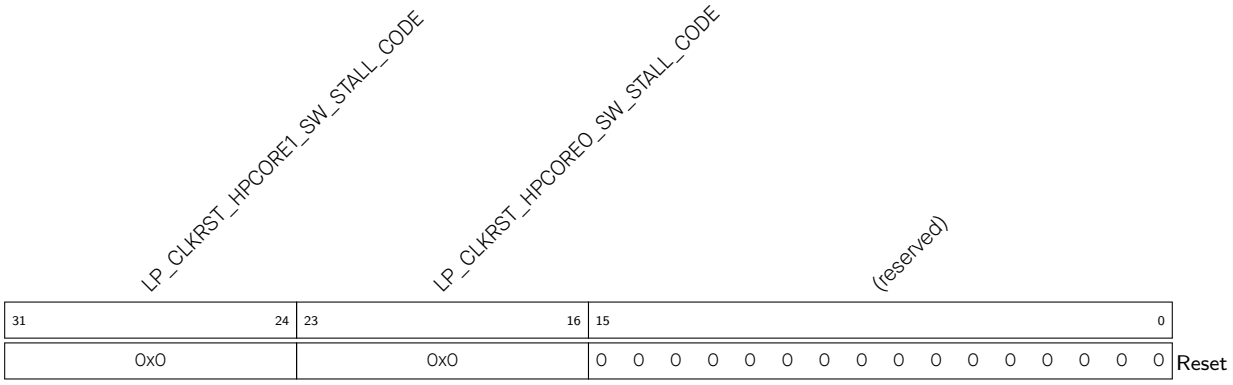
LP_CLKRST_HPCORE1_STAT_VECTOR_SEL 配置 HP CPU1 启动的地址。

0：从 LP SPM RAM (0x50108000) 启动

1：从 HP SPM ROM (0x4FC00000) 启动

(R/W)

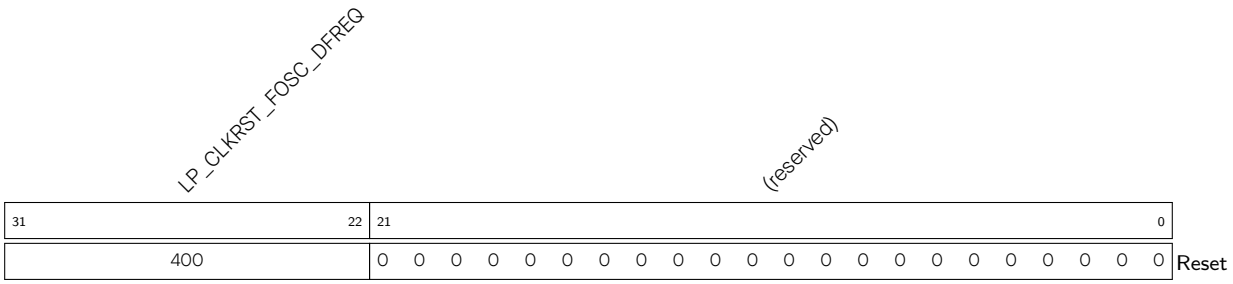
Register 11.64. LP_CLKRST_HPCPU_RESET_CTRL1_REG (0x0018)



LP_CLKRST_HPCORE0_SW_STALL_CODE 配置软件触发 HP CPU0 暂停的阈值。(R/W)

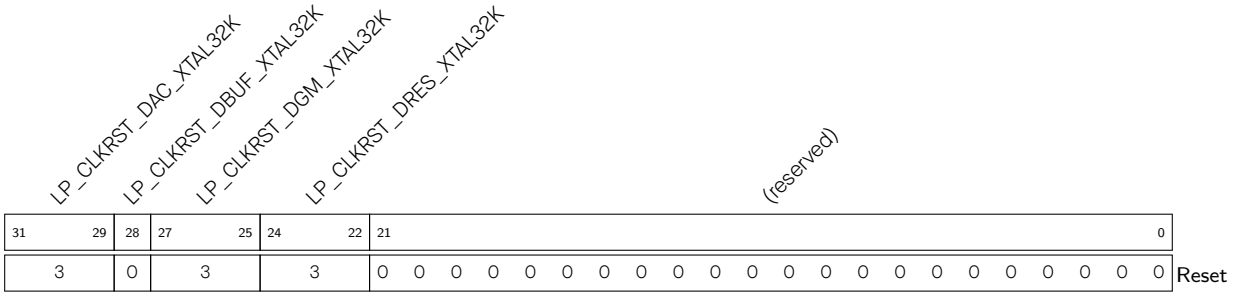
LP_CLKRST_HPCORE1_SW_STALL_CODE 配置软件触发 HP CPU1 暂停的阈值。(R/W)

Register 11.65. LP_CLKRST_FOSC_CNTL_REG (0x001C)



LP_CLKRST_FOSC_DFREQ 配置 RC_FAST_CLK 的频率。(R/W)

Register 11.66. LP_CLKRST_XTAL32K_REG (0x0030)



LP_CLKRST_DRES_XTAL32K 配置 XTAL32K_CLK 的偏置电阻。

0：关闭内部电阻，使用外部电阻

1-3：内部电阻偏置控制值

(R/W)

LP_CLKRST_DGM_XTAL32K 配置 XTAL32K_CLK 的 gm。(R/W)

LP_CLKRST_DBUF_XTAL32K 配置 XTAL32K_CLK 的 buffer。

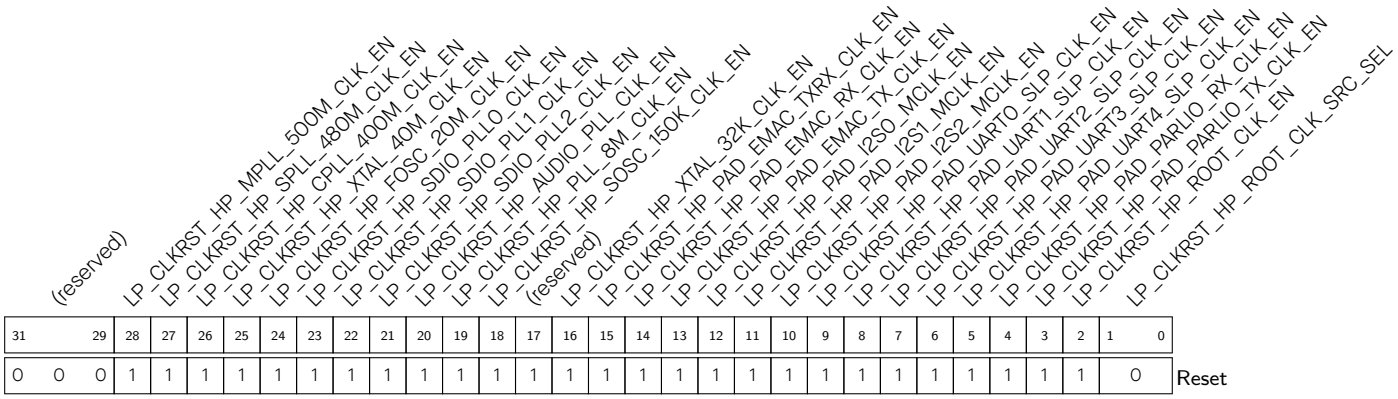
0：单端 buffer

1：差分 buffer

(R/W)

LP_CLKRST_DAC_XTAL32K 配置 XTAL32K_CLK 的偏置电流 DAC。(R/W)

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)



LP_CLKRST_HP_ROOT_CLK_SRC_SEL 配置 ROOT_CLK 的时钟源。

0：XTAL_CLK

1：CPLL_CLK (360 MHz)

2：RC_FAST_CLK

3：非法值

(R/W)

LP_CLKRST_HP_ROOT_CLK_EN 配置是否使能 ROOT_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_PARLIO_TX_CLK_EN 配置是否使能来自 IO 管脚的 PAD_PARLIO_TX_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_PARLIO_RX_CLK_EN 配置是否使能来自 IO 管脚的 PAD_PARLIO_RX_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_UART4_SLP_CLK_EN 配置是否使能来自 IO 管脚的 PAD_UART4_SLP_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_UART3_SLP_CLK_EN 配置是否使能来自 IO 管脚的 PAD_UART3_SLP_CLK。

0：禁用

1：使能

(R/W)

见下页...

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

接上页...

LP_CLKRST_HP_PAD_UART2_SLP_CLK_EN 配置是否使能来自 IO 管脚的 PAD_UART2_SLP_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_UART1_SLP_CLK_EN 配置是否使能来自 IO 管脚的 PAD_UART1_SLP_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_UART0_SLP_CLK_EN 配置是否使能来自 IO 管脚的 PAD_UART0_SLP_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_I2S2_MCLK_EN 配置是否使能来自 IO 管脚的 PAD_I2S2_MCLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_I2S1_MCLK_EN 配置是否使能来自 IO 管脚的 PAD_I2S1_MCLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_I2S0_MCLK_EN 配置是否使能来自 IO 管脚的 PAD_I2S0_MCLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_EMAC_TX_CLK_EN 配置是否使能来自 IO 管脚的 PAD_EMAC_RX_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PAD_EMAC_RX_CLK_EN 配置是否使能来自 IO 管脚的 PAD_EMAC_TX_CLK。

0：禁用

1：使能

(R/W)

见下页...

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

接上页...

LP_CLKRST_HP_PAD_EMAC_TXRX_CLK_EN 配置是否使能来自 IO 管脚的 PAD_EMAC_TXRX_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_XTAL_32K_CLK_EN 配置是否使能 XTAL32K_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_SOSC_150K_CLK_EN 配置是否使能 RC_SLOW_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_PLL_8M_CLK_EN 配置是否使能 PLL_LP_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_AUDIO_PLL_CLK_EN 配置是否使能 APLL_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_SDIO_PLL2_CLK_EN 配置是否使能 SDIO_PLL2_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_SDIO_PLL1_CLK_EN 配置是否使能 SDIO_PLL1_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_SDIO_PLL0_CLK_EN 配置是否使能 SDIO_PLL0_CLK。

0：禁用

1：使能

(R/W)

见下页...

Register 11.67. LP_CLKRST_HP_CLK_CTRL_REG (0x0040)

接上页...

LP_CLKRST_HP_FOSC_20M_CLK_EN 配置是否使能 RC_FAST_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_XTAL_40M_CLK_EN 配置是否使能 XTAL_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_CPLL_400M_CLK_EN 配置是否使能 CPLL_CLK。

0：禁用

1：使能

(R/W)

LP_CLKRST_HP_SPLL_480M_CLK_EN 配置是否使能 SPLL_CLK。

0：禁用

1：使能

(R/W)

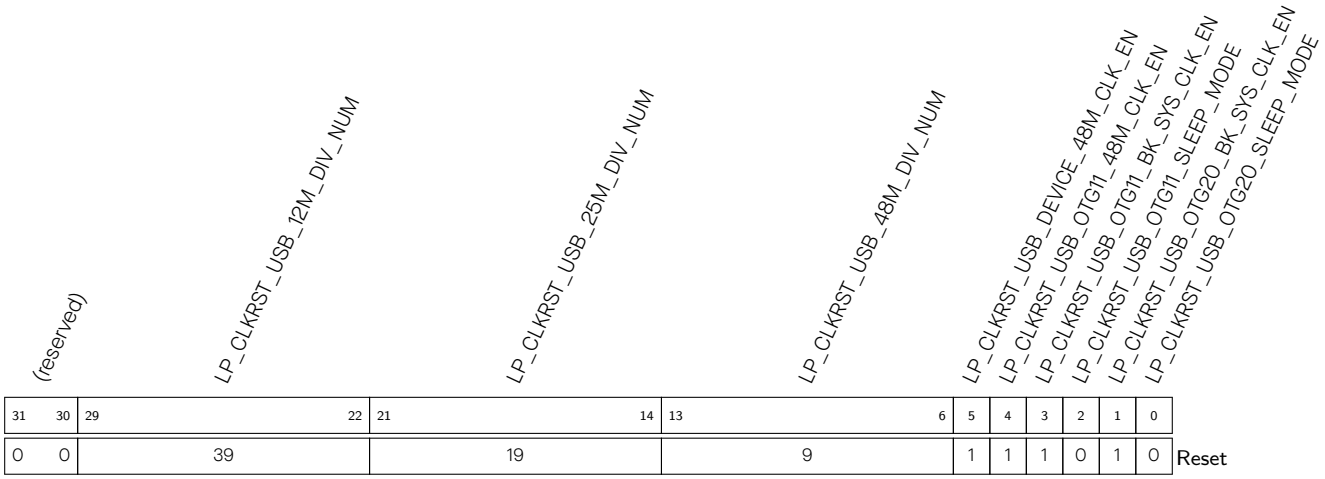
LP_CLKRST_HP_MPLL_500M_CLK_EN 配置是否使能 MPLL_CLK。

0：禁用

1：使能

(R/W)

Register 11.68. LP_CLKRST_HP_USB_CLKRST_CTRL0_REG (0x0044)



LP_CLKRST_USB_OTG20_SLEEP_MODE 未使用。(R/W)

LP_CLKRST_USB_OTG20_BK_SYS_CLK_EN 未使用。(R/W)

LP_CLKRST_USB_OTG11_SLEEP_MODE 未使用。(R/W)

LP_CLKRST_USB_OTG11_BK_SYS_CLK_EN 未使用。(R/W)

LP_CLKRST_USB_OTG11_48M_CLK_EN 配置是否使能 USB 2.0 全速 OTG PHY 时钟。

0：禁用

1：使能

(R/W)

LP_CLKRST_USB_DEVICE_48M_CLK_EN 配置是否使能 USB 串口/JTAG PHY 时钟。

0：禁用

1：使能

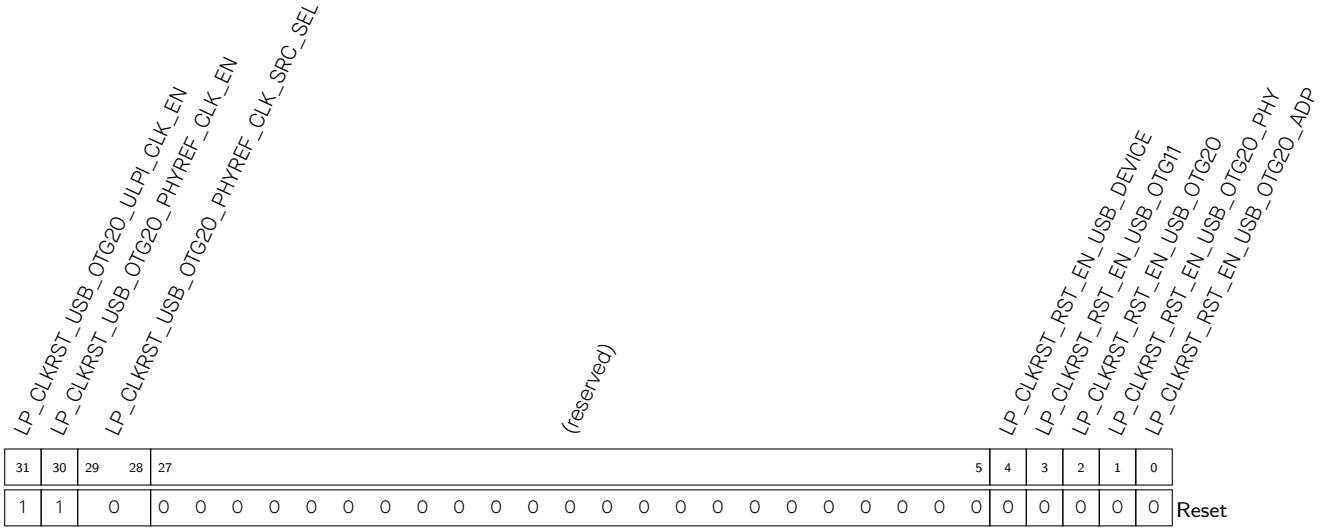
(R/W)

LP_CLKRST_USB_48M_DIV_NUM 配置 USB 480M 到 25M 的分频数。(R/W)

LP_CLKRST_USB_25M_DIV_NUM 配置 USB 500M 到 25M 的分频数。(R/W)

LP_CLKRST_USB_12M_DIV_NUM 配置 USB 480M 到 12M 的分频数。(R/W)

Register 11.69. LP_CLKRST_HP_USB_CLKRST_CTRL1_REG (0x0048)



LP_CLKRST_RST_EN_USB_OTG20_ADP 配置是否复位 USB 2.0 高速 OTG ADP。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_USB_OTG20_PHY 配置是否复位 USB 2.0 高速 OTG PHY。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_USB_OTG20 配置是否复位 USB 2.0 高速 OTG。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_USB_OTG11 配置是否复位 USB 2.0 全速 OTG。

0：复位释放

1：复位

(R/W)

LP_CLKRST_RST_EN_USB_DEVICE 配置是否复位 USB 串口/JTAG。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.69. LP_CLKRST_HP_USB_CLKRST_CTRL1_REG (0x0048)

接上页...

LP_CLKRST_USB_OTG20_PHYREF_CLK_SRC_SEL 配置 USB 2.0 高速 OTG PHY 的时钟源。

0：PLL_F12M_CLK

1：PLL_F25M_CLK

2：PAD_USB_HSPHY_REFCLK

3：非法值

(R/W)

LP_CLKRST_USB_OTG20_PHYREF_CLK_EN 配置是否使能 USB 2.0 高速 OTG PHY REFCLK。

0：禁用

1：使能

(R/W)

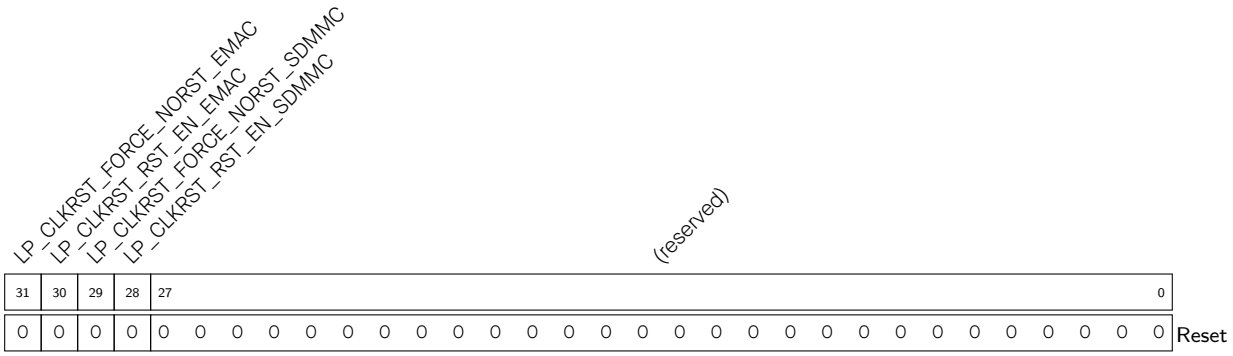
LP_CLKRST_USB_OTG20_ULPI_CLK_EN 配置是否使能 USB 2.0 高速 OTG ULPI 时钟。

0：禁用

1：使能

(R/W)

Register 11.70. LP_CLKRST_HP_SDMMC_EMAC_RST_CTRL_REG (0x004C)



LP_CLKRST_RST_EN_SDMMC 配置是否复位 SDMMC。

0：复位释放

1：复位

(R/W)

LP_CLKRST_FORCE_NORST_SDMMC 配置 SDMMC 是否可复位。

0：可复位

1：强制不可复位

(R/W)

LP_CLKRST_RST_EN_EMAC 配置是否复位 EMAC。

0：复位释放

1：复位

(R/W)

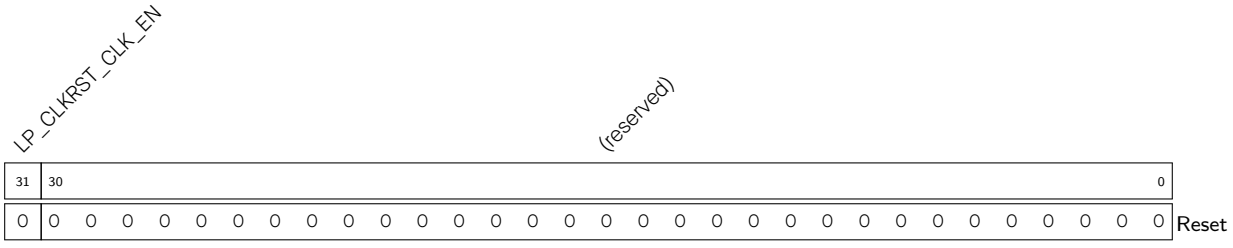
LP_CLKRST_FORCE_NORST_EMAC 配置 EMAC 是否可复位。

0：可复位

1：强制不可复位

(R/W)

Register 11.71. LP_CLKRST_DATE_REG (0x03FC)



LP_CLKRST_CLK_EN 配置寄存器时钟门控。

0：仅在应用写寄存器时开启时钟

1：强制开启配置寄存器时钟

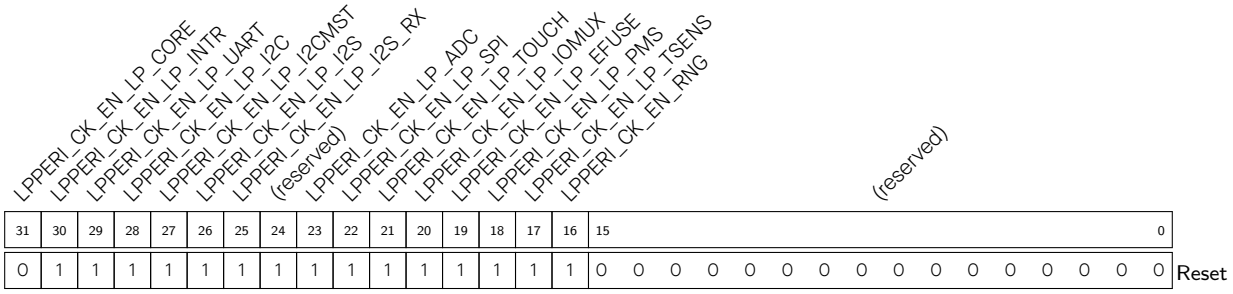
(R/W)

11.5.3 LP 外设时钟与复位 (LPPERI) 寄存器

本小节的所有地址均为相对于 LPPERI 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 11.72. LPPERI_CLK_EN_REG (0x0000)



LPPERI_CK_EN_RNG 配置是否使能 LP RNG 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_TSENS 配置是否使能 LP 温度传感器时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_PMS 配置是否使能 LP PMS 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_EFUSE 配置是否使能 LP eFuse 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_IOMUX 配置是否使能 LP IO MUX 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_TOUCH 配置是否使能 LP 触摸传感器时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_SPI 配置是否使能 LP SPI 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_ADC 配置是否使能 LP ADC 时钟。

0：禁用

1：使能

(R/W)

见下页...

Register 11.72. LPPERI_CLK_EN_REG (0x0000)

接上页...

LPPERI_CK_EN_LP_I2S_RX 配置是否使能 LP I2S RX 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_I2S 配置是否使能 LP I2S 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_I2CMST 配置是否使能 LP I2C 主机时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_I2C 配置是否使能 LP I2C 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_UART 配置是否使能 LP UART 时钟。

0：禁用

1：使能

(R/W)

LPPERI_CK_EN_LP_INTR 配置是否使能 LP 中断矩阵时钟。

0：禁用

1：使能

(R/W)

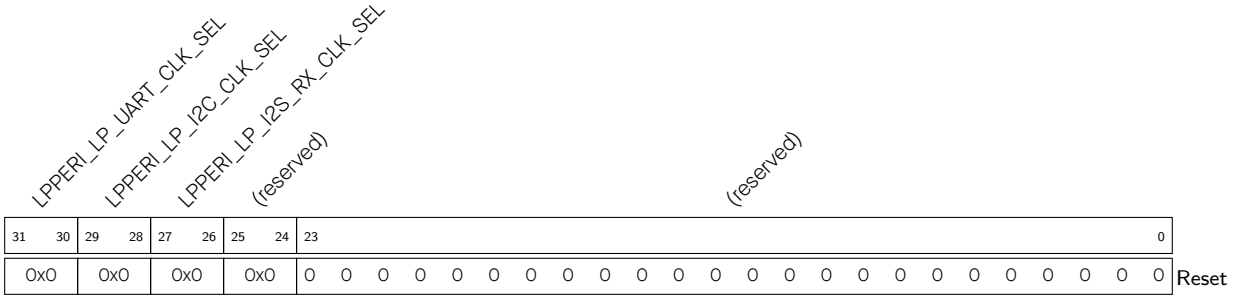
LPPERI_CK_EN_LP_CORE 配置是否强制打开 LP CPU 时钟。

0：交由硬件控制

1：强制打开，旁路硬件控制

(R/W)

Register 11.73. LPPERI_CORE_CLK_SEL_REG (0x0004)



LPPERI_LP_I2S_RX_CLK_SEL 配置 LP I2S RX 时钟源。

0：LP_FAST_CLK

1：XTAL_D2_CLK

2：PLL_LP_CLK

3：非法值

(R/W)

LPPERI_LP_I2C_CLK_SEL 配置 LP I2C 时钟源。

0：LP_FAST_CLK

1：XTAL_D2_CLK

2：PLL_LP_CLK

3：非法值

(R/W)

LPPERI_LP_UART_CLK_SEL 配置 LP UART 时钟源。

0：RC_FAST_CLK

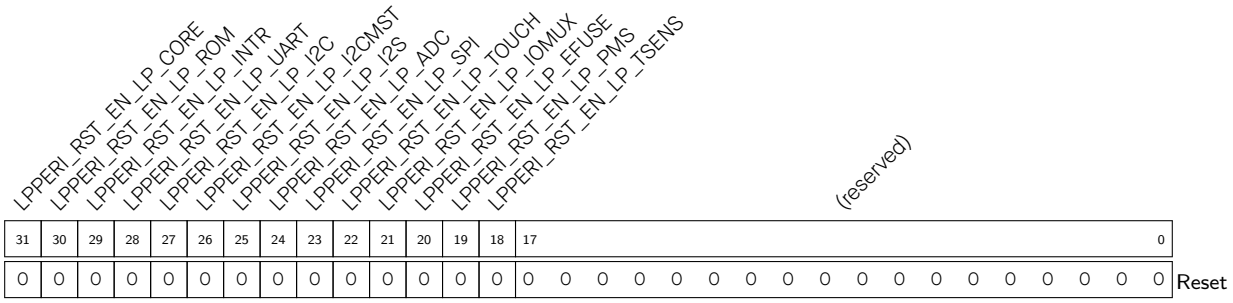
1：XTAL_D2_CLK

2：PLL_LP_CLK

3：非法值

(R/W)

Register 11.74. LPPERI_RESET_EN_REG (0x0008)



LPPERI_RST_EN_LP_TSENS 配置是否复位 LP 温度传感器。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_PMS 配置是否复位 LP PMS。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_EFUSE 配置是否复位 LP eFuse。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_IOMUX 配置是否复位 LP IO MUX 。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_TOUCH 配置是否复位 LP 触摸传感器。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_SPI 配置是否复位 LP SPI。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_ADC 配置是否复位 LP ADC。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_I2S 配置是否复位 LP I2S。

0：复位释放

1：复位

(R/W)

见下页...

Register 11.74. LPPERI_RESET_EN_REG (0x0008)

接上页...

LPPERI_RST_EN_LP_I2CMST 配置是否复位 LP I2C 主机。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_I2C 配置是否复位 LP I2C。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_UART 配置是否复位 LP UART。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_INTR 配置是否复位 LP 中断矩阵。

0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_ROM 配置是否复位 LP ROM。

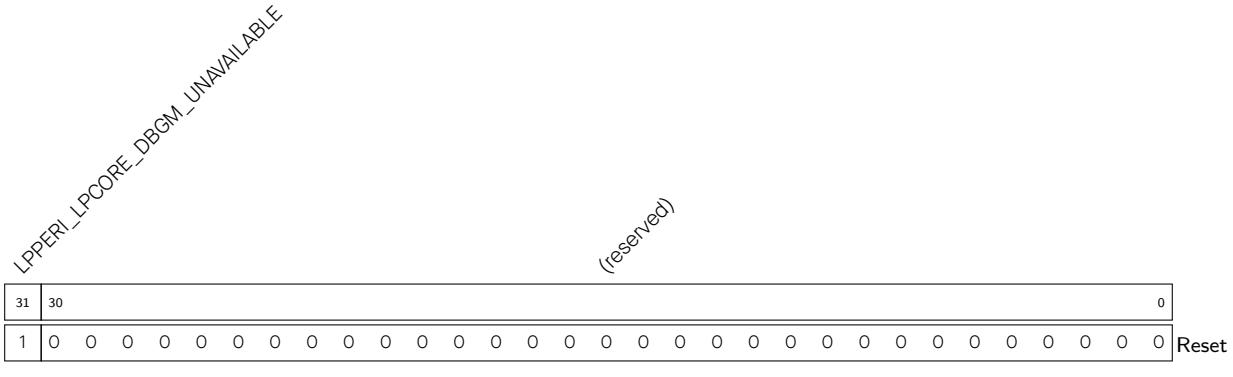
0：复位释放

1：复位

(R/W)

LPPERI_RST_EN_LP_CORE 写 1 触发 LP CPU 软件复位。(WT)

Register 11.75. LPPERI_CPU_REG (0x000C)



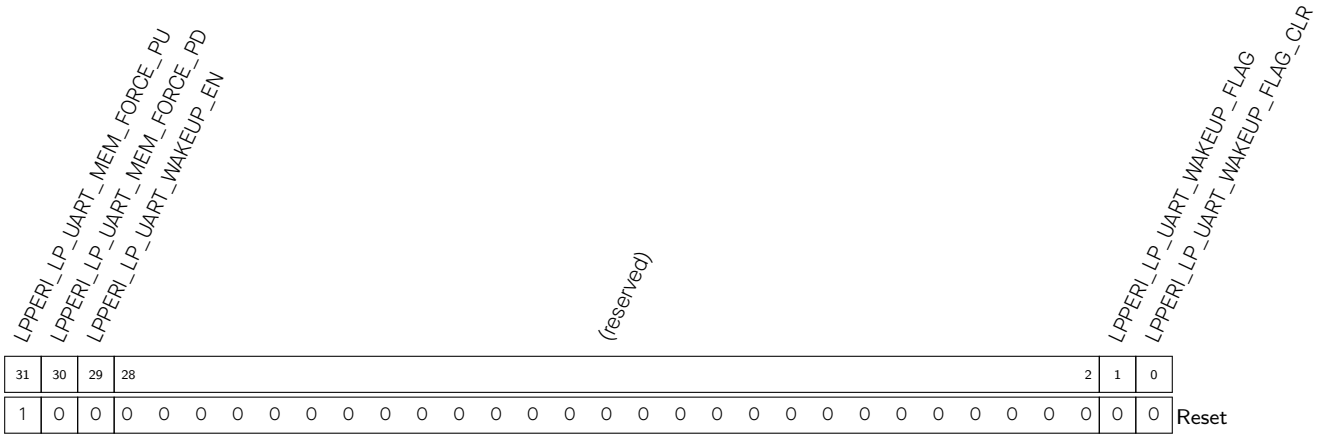
LPPERI_LPCORE_DBGM_UNAVAILABLE 配置 LP CPU 是否可通过 HP CPU 访问。

0：不可访问

1：可访问

(R/W)

Register 11.76. LPPERI_MEM_CTRL_REG (0x0028)



LPPERI_LP_UART_WAKEUP_FLAG_CLR 写 1 清除 LPPERI_LP_UART_WAKEUP_FLAG。(WT)

LPPERI_LP_UART_WAKEUP_FLAG 表示是否发生 LP UART 唤醒。

0：未发生 LP UART 唤醒

1：发生 LP UART 唤醒

(R/WTC/SS)

LPPERI_LP_UART_WAKEUP_EN 配置是否使能 LP UART 唤醒功能。

0：使能

1：禁用

(R/W)

LPPERI_LP_UART_MEM_FORCE_PD 配置是否强制 LP UART 存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

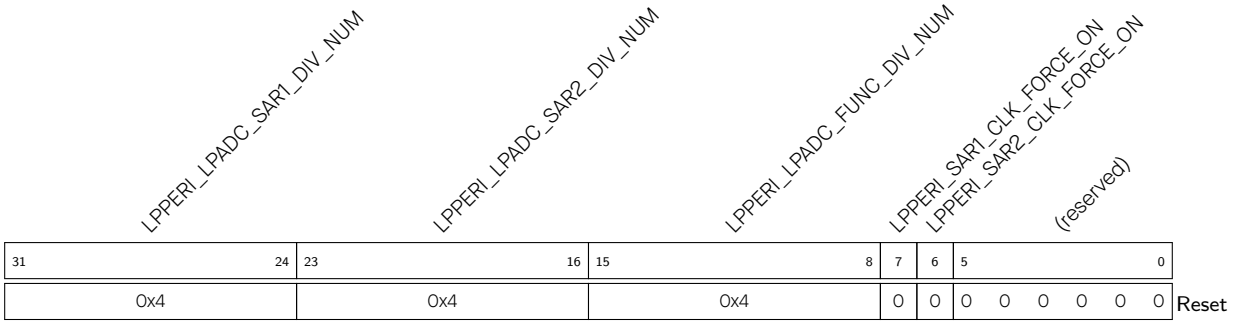
LPPERI_LP_UART_MEM_FORCE_PU 配置是否强制 LP UART 存储器上电。

0：不强制上电

1：强制上电

(R/W)

Register 11.77. LPPERI_ADC_CTRL_REG (0x002C)



LPPERI_SAR2_CLK_FORCE_ON 配置是否强制打开 LP ADC SAR1 时钟。

0：交由硬件控制

1：强制打开，旁路硬件控制

(R/W)

LPPERI_SAR1_CLK_FORCE_ON 配置是否强制打开 LP ADC SAR2 时钟。

0：交由硬件控制

1：强制打开，旁路硬件控制

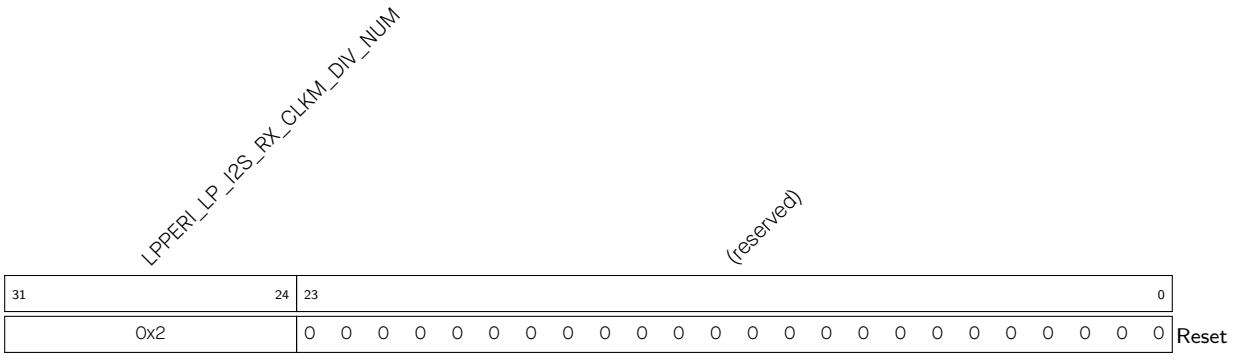
(R/W)

LPPERI_LPADC_FUNC_DIV_NUM 配置 LP ADC 功能时钟分频系数。(R/W)

LPPERI_LPADC_SAR2_DIV_NUM 配置 LP ADC SAR1 时钟分频系数。(R/W)

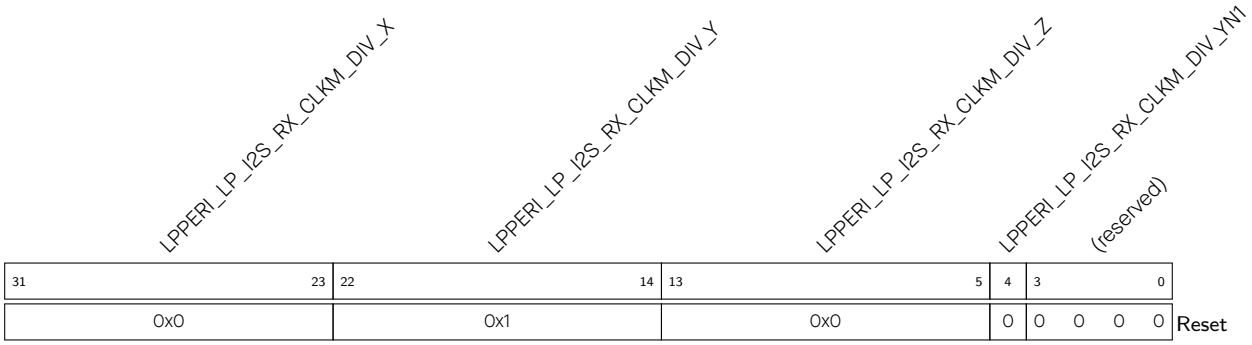
LPPERI_LPADC_SAR1_DIV_NUM 配置 LP ADC SAR2 时钟分频系数。(R/W)

Register 11.78. LPPERI_LP_I2S_RXCLK_DIV_NUM_REG (0x0030)



LPPERI_LP_I2S_RX_CLKM_DIV_NUM 配置 LP I2S RX 时钟分频系数的整数部分。(R/W)

Register 11.79. LPPERI_LP_I2S_RXCLK_DIV_XYZ_REG (0x0034)



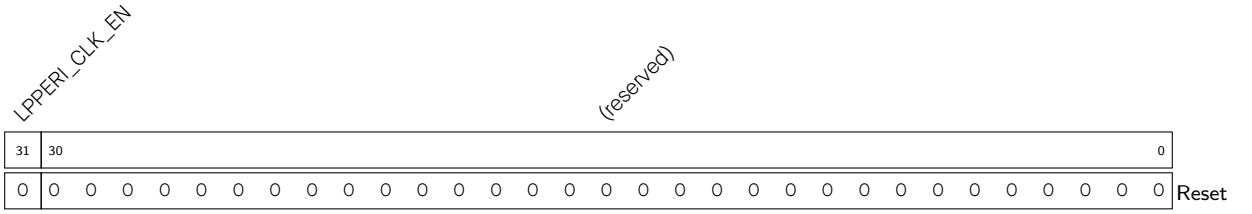
LPPERI_LP_I2S_RX_CLKM_DIV_YN1 配置 LP I2S RX 时钟分频的系数 yn1。(R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_Z 配置 LP I2S RX 时钟分频的系数 z。(R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_Y 配置 LP I2S RX 时钟分频的系数 y。(R/W)

LPPERI_LP_I2S_RX_CLKM_DIV_X 配置 LP I2S RX 时钟分频的系数 x。(R/W)

Register 11.80. LPPERI_DATE_REG (0x03FC)



LPPERI_CLK_EN 配置寄存器时钟门控。

0：仅在应用写寄存器时开启时钟

1：强制开启配置寄存器时钟

(R/W)

第 12 章

芯片 Boot 控制

12.1 概述

芯片 Boot 过程以及芯片的某些功能是在上电或硬件复位时由 Strapping 管脚和一些 eFuse 比特位来确定的，具体包括以下功能：

- 控制 Boot 模式
- 控制 ROM 代码日志打印
- 控制 JTAG 信号源

ESP32-P4 共有五个 Strapping 管脚：

- GPIO34
- GPIO35
- GPIO36
- GPIO37
- GPIO38

在芯片复位过程中（请参考章节 11 复位和时钟），硬件将采样 Strapping 管脚电平存储到锁存器中，并一直保持到芯片掉电。Strapping 管脚的锁存值可以通过软件从寄存器 GPIO_STRAPPING 中读取。

12.2 功能描述

本小节主要介绍芯片复位时的功能以及控制该功能使用到的 Strapping 管脚和 eFuse 组合模式。

注意：
请使用本章节所介绍的组合，其它组合可能会导致不可控结果。

12.2.1 默认配置

GPIO35 默认连接内部上拉电阻。如果这一管脚没有外部连接或者连接的外部线路处于高阻抗状态，内部弱上拉将决定这一管脚输入电平的默认值，如表 12.2-1 所示。

表 12.2-1. 管脚默认上拉/下拉

管脚	默认值
GPIO34	浮空
GPIO35	上拉
GPIO36	浮空
GPIO37	浮空
GPIO38	浮空

如需改变 Strapping 管脚的默认值，用户可以应用外部下拉/上拉电阻，或者应用主机 MCU 的 GPIO 来控制 ESP32-P4 上电复位时的 Strapping 管脚电平。复位释放后，Strapping 管脚和普通管脚功能相同。

12.2.2 Boot 模式控制

复位释放后，GPIO35，GPIO36，GPIO37 和 GPIO38 共同控制 Boot 模式。表 12.2-2 列出了 GPIO35，GPIO36，GPIO37 和 GPIO38 的 Strapping 值及其对应的系统 Boot 模式。

表 12.2-2. 系统 Boot 模式

Boot 模式	GPIO35	GPIO36	GPIO37	GPIO38
SPI Boot 模式（默认）	1	x[1]	x	x
Joint Download 模式[2]	0	1	x	x
SPI Download 模式[3]	0	0	0	1
无效组合[4]	0	0	1	x
	0	0	0	0

[1] x：任何取值均不会对结果有影响，因此可忽略。

[2] Joint Download 模式：Joint Download 模式下支持以下下载方式：

- USB-Serial-JTAG Download
- UART Download
- SPI Slave Download
- USB 2.0 OTG Download

[3] SPI Download 模式：只有使用 SPI Download 模式时才需要预留 GPIO37 和 GPIO38。GPIO37 和 GPIO38 默认浮空，在复位时处于高阻抗状态。

[4] 无效组合：该组合会触发意外行为，应当避免。

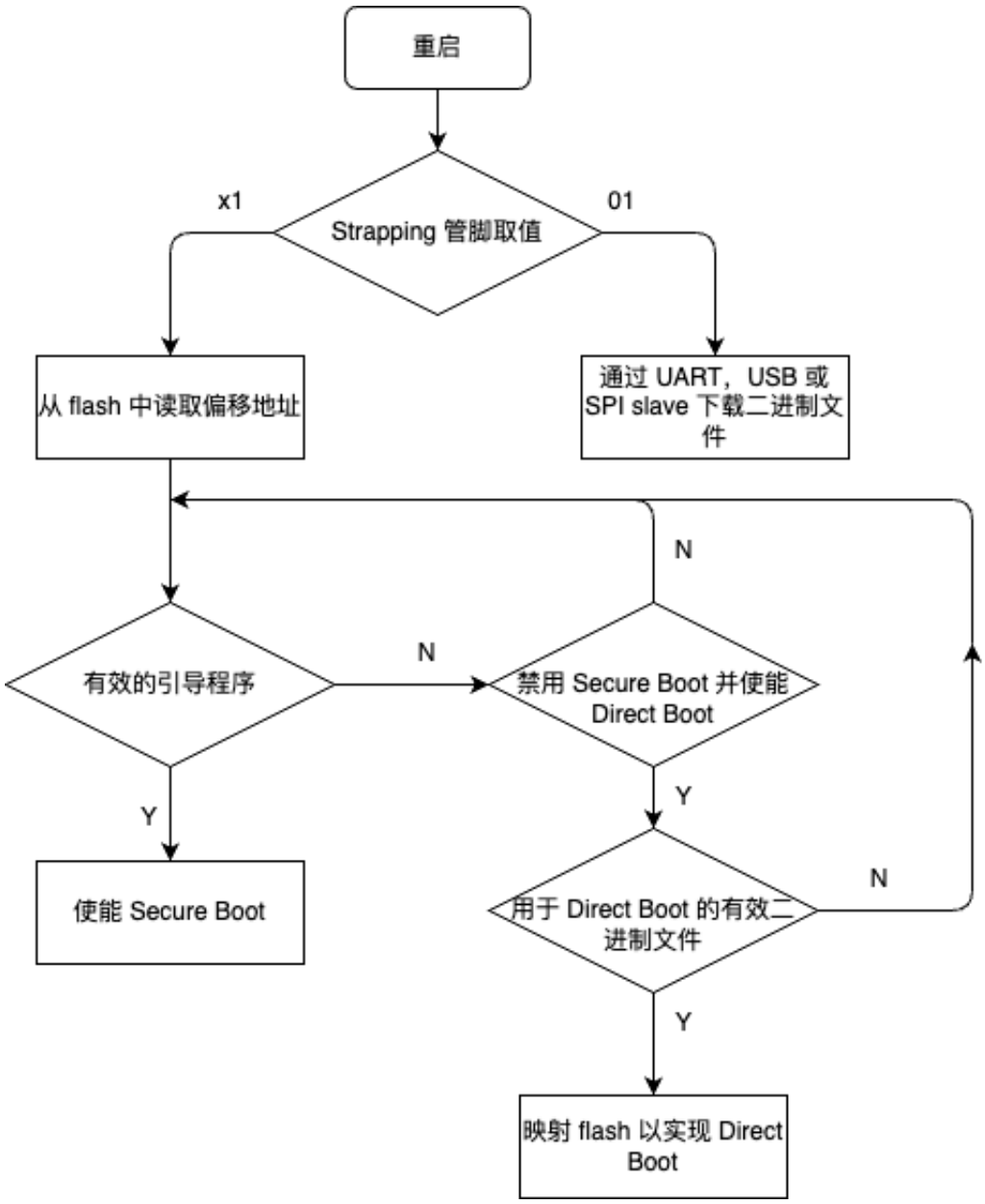
在 SPI Boot 模式下，ROM 引导加载程序通过从 SPI flash 中读取程序来启动系统。SPI Boot 模式可进一步细分为以下两种 Boot 方式：

- 常规 flash Boot 方式：支持安全启动。ROM 引导加载程序将程序从 flash 加载到 L2MEM，并执行。在大多数实际应用场景中，上述执行的程序多为二级引导程序，该二级引导程序将启动最终的应用程序。
- 直接 Boot 方式：不支持安全启动，程序直接从 flash 中运行。如需使能这一启动方式，请确保下载至 flash 的二进制文件其前两个字为 0xaedb041d。详细的 Boot 流程，见图 12.2-1。

在 Joint Download 模式下，用户可通过 UART0，SPI slave，USB 2.0 OTG 或 USB 串口/JTAG 接口将二进制文件下载至 flash，或将二进制文件下载至 L2MEM 并运行 L2MEM 中的程序。

在 SPI Download 模式下，用户可通过 SPI 接口将二进制文件下载至 flash，或将二进制文件下载至 L2MEM 并运行 L2MEM 中的程序。

芯片 Boot 的具体流程见图 12.2-1。



注：图中的 “x1” 和 “01” 为 Strapping 管脚 GPIO35 和 GPIO36 的组合值，见表 12.2-2。

图 12.2-1. 芯片 Boot 流程

下面几个 eFuse 比特位可用于控制 Boot 模式的具体行为：

- EFUSE_DIS_FORCE_DOWNLOAD

  如果此 eFuse 设置为 0（默认），软件可通过设置 LPSYSREG_FORCE_DOWNLOAD_BOOT，触发 CPU 复位，将芯片 Boot 模式强制从 SPI Boot 模式切换至 Joint Download 模式；如果此 eFuse 设置为 1，则禁用 LPSYSREG_FORCE_DOWNLOAD_BOOT。

- EFUSE_DIS_DOWNLOAD_MODE

如果此 eFuse 设置为 1，则禁用 Joint Download 模式。

- EFUSE_ENABLE_SECURITY_DOWNLOAD

  如果此 eFuse 设置为 1，则在 Joint Download 模式下，只允许读取、写入和擦除明文 flash，不支持 L2MEM 或寄存器操作。如已禁用 Joint Download 模式，请忽略此 eFuse。

- EFUSE_DIS_DIRECT_BOOT

  如果此 eFuse 设置为 1，SPI Boot 模式下的 Direct Boot 会被禁用。

USB Serial/JTAG 控制器可将芯片从 SPI Boot 模式强制切换到 Joint Download 模式，或从 Joint Download 模式强制切换到 SPI Boot 模式。更多信息，请参考章节 *54 USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)*。

可以设置 eFuse 位 EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE 来禁止 USB 串口/JTAG 控制器强制切换到 Joint Download 模式。

12.2.3 ROM 代码日志打印控制

系统在 SPI Boot 模式下的早期阶段，ROM 代码日志可打印至：

- **（默认）UART0 和 USB 串口/JTAG 控制器**
- **UART0**
- **USB 串口/JTAG 控制器**

EFUSE_UART_PRINT_CONTROL 和 GPIO36 控制 ROM 日志打印至 **UART0**，如表 12.2-3 ROM 日志打印 所示。

表 12.2-3. ROM 日志打印

eFuse[1]	GPIO36	ROM 代码日志打印
0	x[2]	始终使能
1	0	使能
	1	关闭
2	0	关闭
	1	使能
3	x	始终关闭

[1] eFuse: EFUSE_UART_PRINT_CONTROL

[2] x：任何取值均不会对结果有影响，因此可忽略。

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 控制打印至 **USB 串口/JTAG 控制器**。该位为 1 时禁用打印至 USB 串口/JTAG 控制器。该位为 0，且 USB 串口/JTAG 控制器已通过 EFUSE_DIS_USB_SERIAL_JTAG 开启，ROM 日志可打印到 USB 串口/JTAG 控制器。

注意，在 EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 设置为 0，即可打印至 USB 的情况下，如果 USB 串口/JTAG 控制器已被禁用，则 ROM 代码将无法打印到 USB 串口/JTAG 控制器。

12.2.4 JTAG 信号源控制

在系统 Boot 早期阶段，GPIO34 与 EFUSE_DIS_PAD_JTAG、EFUSE_DIS_USB_JTAG 和 EFUSE_JTAG_SEL_ENABLE 一起控制 JTAG 信号源，见表 12.2-4。

表 12.2-4. JTAG 信号源控制

eFuse 1[a]	eFuse 2[b]	eFuse 3[c]	GPIO34	JTAG 信号源
0	0	0	x[d]	JTAG 信号来自 USB Serial/JTAG 控制器
		1	0	JTAG 信号来自相应管脚 MTDI、MTCK、MTMS 和 MTDO
			1	JTAG 信号来自 USB Serial/JTAG 控制器
0	1	x	x	JTAG 信号来自相应管脚 MTDI、MTCK、MTMS 和 MTDO
1	0	x	x	JTAG 信号来自 USB Serial/JTAG 控制器
1	1	x	x	JTAG 被禁用

[a] eFuse 1: EFUSE_DIS_PAD_JTAG

[b] eFuse 2: EFUSE_DIS_USB_JTAG

[c] eFuse 3: EFUSE_JTAG_SEL_ENABLE

[d] x：任何取值均不会对结果有影响，因此可忽略。

第 13 章

中断矩阵

13.1 概述

ESP32-P4 中断矩阵将任一外部中断源单独映射到 ESP-RISC-V CPU 的任一外部中断上，以便在外设中断信号产生后，及时通知 HP CPU0 或 HP CPU1 进行处理。

外部中断源必须经中断矩阵分配至 HP CPU0/HP CPU1 外部中断，主要是因为：

- ESP32-P4 有 128 个外部中断源，但 HP CPU0/HP CPU1 只支持 32 个外部中断。因此，将这些外部中断源映射至 HP CPU0 中断或 HP CPU1 必须使用中断矩阵。
- 通过中断矩阵，可以根据应用需要，将一个外部中断源映射至 HP CPU0 的任一外部中断或 HP CPU1 的任一外部中断上。

说明：
本章节只涉及将外部中断源映射到 CPU0 或 CPU1 中断，关于中断配置、中断向量表、中断服务程序推荐处理机制请参考章节 2 高性能处理器。

13.2 ESP32-P4 中断术语

下列有关中断的术语是在《ESP32-P4 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

13.2.1 中断

中断指的是在特定事件或条件下，CPU 中止当前执行去处理中断服务程序（ISR）。中断机制使 CPU 能够迅速响应特定的事件。

《ESP32-P4 技术参考手册》里涉及的“中断”是个广义的概念，可以泛指中断信号或中断源。

13.2.2 中断信号/中断源

中断信号和中断源只是定义的角度不同，实际意思相同，都是描述同一个信号。

从外设的角度来看，中断信号是由外设的内部中断源产生的，并发送到中断矩阵。

从中断矩阵的角度来看，它接收来自外设发送的中断信号，并将其视为中断源。然后，中断矩阵将 CPU 外部中断信号发送到 CPU。

从 CPU 的角度来看，来自中断矩阵的中断信号又变成中断源，与核心本地中断源一起发送给 CPU 中断控制器。

13.2.3 ESP32-P4 中断流

图 13.2-1 所示为 ESP32-P4 的中断流。

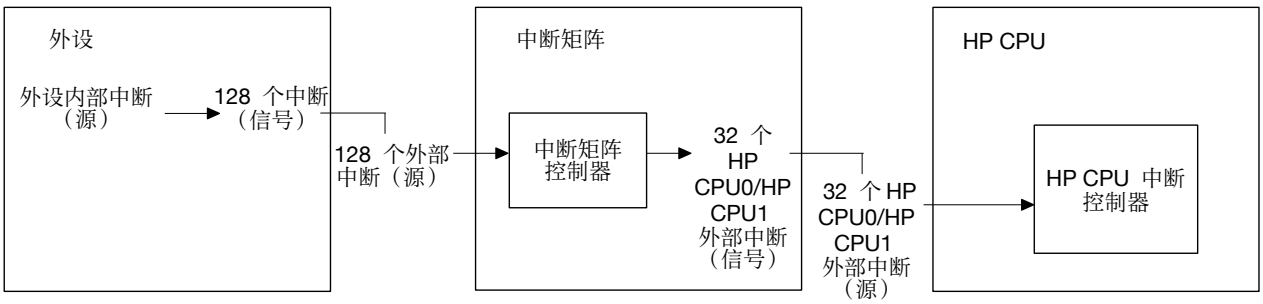


图 13.2-1. ESP32-P4 的中断流

13.3 特性

ESP32-P4 的中断矩阵模块具有如下特性：

- 接收 128 个外部中断源作为输入
- 生成 32 个 HP CPU0 的外部中断和 32 个 HP CPU1 的外部中断作为输出
- 支持查询外部中断源当前的中断状态
- 支持将多个中断源映射到单个 HP CPU0 中断或 HP CPU1 中断（即共享中断）
- 支持 HP CPU0 和 HP CPU1 用户模式中断重映射到机器模式中断

中断矩阵的结构如图 13.3-1 所示。

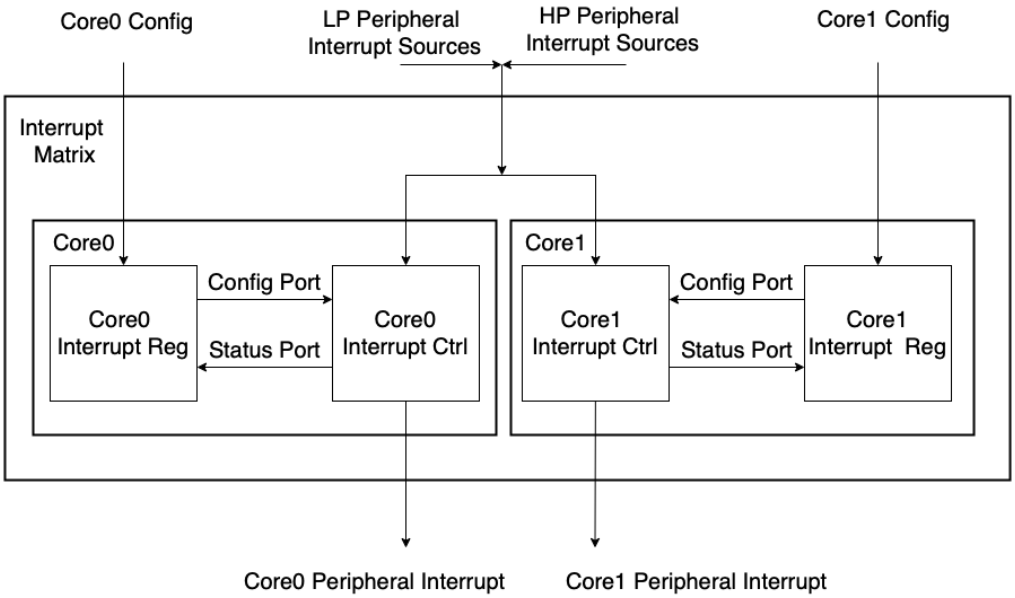


图 13.3-1. 中断矩阵结构图

所有外部中断源产生的中断信号均可由 HP CPU0 或 HP CPU1 进行处理。配置 HP CPU0 中断寄存器（图 13.3-1

Core0 Interrupt Reg 模块）可将外部中断源分配给 HP CPU0 的外部中断，此时外部中断源产生的中断信号会由 HP CPU0 响应处理。同理，配置 HP CPU1 中断寄存器（图 13.3-1 Core1 Interrupt Reg 模块）可将中断信号交由 HP CPU1 响应处理。也可将外部中断源同时分配给 HP CPU0 和 HP CPU1，此时 HP CPU0 和 HP CPU1 都会接收到中断信号。

13.4 功能描述

13.4.1 外部中断源

ESP32-P4 共有 128 个有效外部中断源。表 13.4-1 列出了所有外部中断源及其对应的中断源映射寄存器与中断状态寄存器。

- “No.”：表示外部中断源序号，范围：0 ~ 82，84 ~ 98，100 ~ 111，113 ~ 130（83，99 和 112 保留）
- “章节”：详细描述外部中断源的章节
- “中断源”：外部中断源名称
- “中断源映射寄存器”：用于将外部中断源分配至 HP CPU 外部中断
- “中断状态寄存器”：用于读取中断源的中断状态
  - “中断状态寄存器 - 位”：表示在中断状态寄存器中的比特位置，用于记录相应中断源的状态
  - “中断状态寄存器 - 名称”：表示中断状态寄存器的名称

中断源与同一行中断源映射寄存器和中断状态寄存器一一对应，例如：中断源 RMT_INTR 对应的中断源映射寄存器为 CORE*x*_RMT_INT_MAP_REG，对应的中断状态寄存器为 CORE*x*_INTR_STATUS_REG_1_REG。

注意，表格中的 CORE*x* 可以是 CORE0 (HP CPU0) 或 CORE1 (HP CPU1)。

表 13.4-1. CPU 外部中断源映射寄存器、外部中断状态寄存器、外部中断源

No.	章节	中断源	中断源映射寄存器	中断状态寄存器 位	中断状态寄存器 名称
0	实时时钟定时器	LP_RTC_INTR	COREx LP_RTC_INT_MAP_REG	0	COREx_INTR_STATUS_REG_0_REG
1	看门狗定时器 (WDT)	LP_WDT_INTR	COREx_LP_WDT_INT_MAP_REG	1	
2	实时时钟定时器	LP_TIMER_REG_0_INTR	COREx_LP_TIMER_REG_0_INT_MAP_REG	2	
3	实时时钟定时器	LP_TIMER_REG_1_INTR	COREx_LP_TIMER_REG_1_INT_MAP_REG	3	
4	LP 信箱控制器	MB_LP_INTR	COREx_MB_LP_INT_MAP_REG	4	
5	LP 信箱控制器	MB_HP_INTR	COREx_MB_LP_INT_MAP_REG	5	
6	低功耗管理	PMU_REG_0_INTR	COREx_PMU_REG_0_INT_MAP_REG	6	
7	低功耗管理	PMU_REG_1_INTR	COREx_PMU_REG_1_INT_MAP_REG	7	
8	欠压检测器	LP_ANAPERI_INTR	COREx_PMU_REG_0_INT_MAP_REG	8	
9	ADC 控制器 (ADC)	LP_ADC_INTR	COREx_PMU_REG_0_INT_MAP_REG	9	
10	GPIO 交换矩阵和 IO MUX	LP_GPIO_INTR	COREx_LP_GPIO_INT_MAP_REG	10	
11	I2C 控制器 (I2C)	LP_I2C_INTR	COREx_LP_I2C_INT_MAP_REG	11	
12	I2S 控制器 (I2S)	LP_I2S_INTR	COREx_LP_I2S_INT_MAP_REG	12	
13	SPI 控制器 (SPI)	LP_SPI_INTR	COREx_LP_SPI_INT_MAP_REG	13	
14	触摸传感器 (TOUCH)	LP_TOUCH_INTR	COREx_LP_TOUCH_INT_MAP_REG	14	
15	温度传感器 (TSENS)	LP_TSENS_INTR	COREx_LP_TSENS_INT_MAP_REG	15	
16	UART 控制器 (UART)	LP_UART_INTR	COREx_LP_UART_INT_MAP_REG	16	
17	eFuse 控制器 (EFUSE)	LP_EFUSE_INTR	COREx_LP_EFUSE_INT_MAP_REG	17	
18	低功耗处理器	LP_SW_INTR	COREx_LP_SW_INT_MAP_REG	18	
19	系统寄存器 (SYSREG)	LP_SYSREG_INTR	COREx_LP_SYSREG_INT_MAP_REG	19	
20	密钥管理器	LP_HUK_INTR	COREx_LP_HUK_INT_MAP_REG	20	
21	系统寄存器 (SYSREG)	SYS_ICM_INTR	COREx_SYS_ICM_INT_MAP_REG	21	
22	USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)	USB_DEVICE_INTR	COREx_USB_DEVICE_INT_MAP_REG	22	
23	SD/MMC 主机控制器 (SDHOST)	SDIO_HOST_INTR	COREx_SDIO_HOST_INT_MAP_REG	23	
24	VDMA 控制器 (VDMA)	GDMA_INTR	COREx_GDMA_INT_MAP_REG	24	
25	SPI 控制器 (SPI)	SPI2_INTR	COREx_SPI2_INT_MAP_REG	25	
26	SPI 控制器 (SPI)	SPI3_INTR	COREx_SPI3_INT_MAP_REG	26	
27	I2S 控制器 (I2S)	I2S0_INTR	COREx_I2S0_INT_MAP_REG	27	
28	I2S 控制器 (I2S)	I2S1_INTR	COREx_I2S1_INT_MAP_REG	28	
29	I2S 控制器 (I2S)	I2S2_INTR_0	COREx_I2S2_INT_MAP_REG	29	
30	UART 控制器 (UART)	UHCI0_INTR	COREx_UHCI0_INT_MAP_REG	30	
31	UART 控制器 (UART)	UART0_INTR	COREx_UART0_INT_MAP_REG	31	

No.	章节	中断源	中断源映射寄存器	中断状态寄存器 位	中断状态寄存器 名称
32	UART 控制器 (UART)	UART1_INTR	COREx_UART1_INT_MAP_REG	0	COREx_INTR_STATUS_REG_1_REG
33	UART 控制器 (UART)	UART2_INTR	COREx_UART2_INT_MAP_REG	1	
34	UART 控制器 (UART)	UART3_INTR	COREx_UART3_INT_MAP_REG	2	
35	UART 控制器 (UART)	UART4_INTR	COREx_UART4_INT_MAP_REG	3	
36	LCD 与 Camera 控制器 (LCD_CAM)	LCD_CAM_INTR	COREx_LCD_CAM_INT_MAP_REG	4	
37	ADC 控制器 (ADC)	ADC_INTR	COREx_ADC_INT_MAP_REG	5	
38	电机控制脉宽调制器 (MCPWM)	PWM0_INTR	COREx_PWM0_INT_MAP_REG	6	
39	电机控制脉宽调制器 (MCPWM)	PWM1_INTR	COREx_PWM1_INT_MAP_REG	7	
40	双线汽车接口 (TWAI)	TWAI0_INTR	COREx_TWAI0_INT_MAP_REG	8	
41	双线汽车接口 (TWAI)	TWAI1_INTR	COREx_TWAI1_INT_MAP_REG	9	
42	双线汽车接口 (TWAI)	TWAI2_INTR	COREx_TWAI2_INT_MAP_REG	10	
43	红外遥控 (RMT)	RMT_INTR	COREx_RMT_INT_MAP_REG	11	
44	I2C 控制器 (I2C)	I2C0_INTR	COREx_I2C0_INT_MAP_REG	12	
45	I2C 控制器 (I2C)	I2C1_INTR	COREx_I2C1_INT_MAP_REG	13	
46	定时器组 (TIMG)	TIMERGRP0_T0_INTR	COREx_TIMERGRP0_T0_INT_MAP_REG	14	
47	定时器组 (TIMG)	TIMERGRP0_T1_INTR	COREx_TIMERGRP0_T1_INT_MAP_REG	15	
48	定时器组 (TIMG)	TIMERGRP0_WDT_INTR	COREx_TIMERGRP0_WDT_INT_MAP_REG	16	
49	定时器组 (TIMG)	TIMERGRP1_T0_INTR	COREx_TIMRTGRP1_T0_INT_MAP_REG	17	
50	定时器组 (TIMG)	TIMERGRP1_T1_INTR	COREx_TIMERGRP1_T1_INT_MAP_REG	18	
51	定时器组 (TIMG)	TIMERGRP1_WDT_INTR	COREx_TIMERGRP1_WDT_INT_MAP_REG	19	
52	LED PWM 控制器 (LEDC)	LEDC_INTR	COREx_LEDC_INT_MAP_REG	20	
53	系统定时器	SYSTIMER_TARGET0_INTR	COREx_SYSTIMER_TARGET0_INT_MAP_REG	21	
54	系统定时器	SYSTIMER_TARGET1_INTR	COREx_SYSTIMER_TARGET1_INT_MAP_REG	22	
55	系统定时器	SYSTIMER_TARGET2_INTR	COREx_SYSTIMER_TARGET2_INT_MAP_REG	23	
56	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AHB_PDMA_IN_CH0_INTR	COREx_AHB_PDMA_IN_CH0_INT_MAP_REG	24	
57	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AHB_PDMA_IN_CH1_INTR	COREx_AHB_PDMA_IN_CH1_INT_MAP_REG	25	
58	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AHB_PDMA_IN_CH2_INTR	COREx_AHB_PDMA_IN_CH2_INT_MAP_REG	26	
59	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AHB_PDMA_OUT_CH0_INTR	COREx_AHB_PDMA_OUT_CH0_INT_MAP_REG	27	
60	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AHB_PDMA_OUT_CH1_INTR	COREx_AHB_PDMA_OUT_CH1_INT_MAP_REG	28	
61	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AHB_PDMA_OUT_CH2_INTR	COREx_AHB_PDMA_OUT_CH2_INT_MAP_REG	29	
62	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AXI_PDMA_IN_CH0_INTR	COREx_AXI_PDMA_IN_CH0_INT_MAP_REG	30	
63	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AXI_PDMA_IN_CH1_INTR	COREx_AXI_PDMA_IN_CH1_INT_MAP_REG	31	

No.	章节	中断源	中断源映射寄存器	中断状态寄存器 位	中断状态寄存器 名称
64	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AXI_PDMA_IN_CH2_INTR	COREx_AXI_PDMA_IN_CH2_INT_MAP_REG	0	COREx_INTR_STATUS_REG_2_REG
65	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AXI_PDMA_OUT_CH0_INTR	COREx_AXI_PDMA_OUT_CH0_INT_MAP_REG	1	
66	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AXI_PDMA_OUT_CH1_INTR	COREx_AXI_PDMA_OUT_CH1_INT_MAP_REG	2	
67	通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)	AXI_PDMA_OUT_CH2_INTR	COREx_AXI_PDMA_OUT_CH2_INT_MAP_REG	3	
68	RSA 加速器 (RSA)	RSA_INTR	COREx_RSA_INT_MAP_REG	4	
69	AES 加速器 (AES)	AES_INTR	COREx_AES_INT_MAP_REG	5	
70	SHA 加速器 (SHA)	SHA_INTR	COREx_SHA_INT_MAP_REG	6	
71	SHA 加速器 (SHA)	ECC_INTR	COREx_ECC_INT_MAP_REG	7	
72	ECDSA 数字签名外设 (ECDSA_DS)	ECDSA_INTR	COREx_ECDSA_INT_MAP_REG	8	
73	密钥管理器	KM_INTR	COREx_KM_INT_MAP_REG	9	
74	GPIO 交换矩阵和 IO MUX	GPIO_INTR0	COREx_GPIO_INTR0_MAP_REG	10	
75	GPIO 交换矩阵和 IO MUX	GPIO_INTR1	COREx_GPIO_INTR1_MAP_REG	11	
76	GPIO 交换矩阵和 IO MUX	GPIO_INTR2	COREx_GPIO_INTR2_MAP_REG	12	
77	GPIO 交换矩阵和 IO MUX	GPIO_INTR3	COREx_GPIO_INTR3_MAP_REG	13	
78	GPIO 交换矩阵和 IO MUX	GPIO_PAD_COMP_INTR	COREx_GPIO_PAD_COMP_INT_MAP_REG	14	
79	系统寄存器 (SYSREG)	CPU_INTR_FROM_CPU_0	COREx_CPU_INTR_FROM_CPU_0_MAP_REG	15	
80	系统寄存器 (SYSREG)	CPU_INTR_FROM_CPU_1	COREx_CPU_INTR_FROM_CPU_1_MAP_REG	16	
81	系统寄存器 (SYSREG)	CPU_INTR_FROM_CPU_2	COREx_CPU_INTR_FROM_CPU_2_MAP_REG	17	
82	系统寄存器 (SYSREG)	CPU_INTR_FROM_CPU_3	COREx_CPU_INTR_FROM_CPU_3_MAP_REG	18	
83	n/a	保留	保留	19	
84	SPI 控制器 (SPI)	FLASH_MSPI_INTR	COREx_FLASH_MSPI_INT_MAP_REG	20	
85	MIPI CSI	CSI_BRIDGE_INTR	COREx_CSI_BRIDGE_INT_MAP_REG	21	
86	MIPI DSI	DSI_BRIDGE_INTR	COREx_DSI_BRIDGE_INT_MAP_REG	22	
87	MIPI CSI	CSI_INTR	COREx_CSI_INT_MAP_REG	23	
88	MIPI DSI	DSI_INTR	COREx_DSI_INT_MAP_REG	24	
89	以太网介质访问控制器 (EMAC)	GMII_PHY_INTR	COREx_GMII_PHY_INT_MAP_REG	25	
90	以太网介质访问控制器 (EMAC)	LPI_INTR	COREx_LPI_INT_MAP_REG	26	
91	以太网介质访问控制器 (EMAC)	PMT_INTR	COREx_PMT_INT_MAP_REG	27	
92	以太网介质访问控制器 (EMAC)	ETH_MAC_INTR	COREx_ETH_MAC_INT_MAP_REG	28	
93	USB 2.0 高速 OTG	USB_OTG_INTR	COREx_USB_OTG_INT_MAP_REG	29	
94	USB 2.0 高速 OTG	USB_OTG_ENDP_MULTI_PROC_INTR	COREx_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	30	
95	JPEG 图像编解码器	JPEG_INTR	COREx_JPEG_INT_MAP_REG	31	

No.	章节	中断源	中断源映射寄存器	中断状态寄存器 位	中断状态寄存器 名称
96	像素处理加速器 (PPA)	PPA_INTR	COREx_PPA_INT_MAP_REG	0	COREx_INTR_STATUS_REG_3_REG
97	RISC-V 追踪编码器 (TRACE)	COREO_TRACE_INTR	COREx_COREO_TRACE_INT_MAP_REG	1	
98	RISC-V 追踪编码器 (TRACE)	CORE1_TRACE_INTR	COREx_CORE1_TRACE_INT_MAP_REG	2	
99	n/a	保留	保留	3	
100	图像信号处理器 (ISP)	ISP_INTR	COREx_ISP_INT_MAP_REG	4	
101	I3C 控制器 [to be added later]	I3C_MST_INTR	COREx_I3C_MST_INT_MAP_REG	5	
102	I3C 控制器 [to be added later]	I3C_SLV_INTR	COREx_I3C_SLV_INT_MAP_REG	6	
103	USB 2.0 全速 OTG	USB_OTG11_INTR	COREx_USB_OTG11_INT_MAP_REG	7	
104	2D-DMA 控制器 (2D-DMA)	DMA2D_IN_CH0_INTR	COREx_DMA2D_IN_CH0_INT_MAP_REG	8	
105	2D-DMA 控制器 (2D-DMA)	DMA2D_IN_CH1_INTR	COREx_DMA2D_IN_CH1_INT_MAP_REG	9	
106	2D-DMA 控制器 (2D-DMA)	DMA2D_OUT_CH0_INTR	COREx_DMA2D_OUT_CH0_INT_MAP_REG	10	
107	2D-DMA 控制器 (2D-DMA)	DMA2D_OUT_CH1_INTR	COREx_DMA2D_OUT_CH1_INT_MAP_REG	11	
108	2D-DMA 控制器 (2D-DMA)	DMA2D_OUT_CH2_INTR	COREx_DMA2D_OUT_CH2_INT_MAP_REG	12	
109	SPI 控制器 (SPI)	PSRAM_MSPI_INTR	COREx_PSRAM_MSPI_INT_MAP_REG	13	
110	系统寄存器 (SYSREG)	HP_SYSREG_INTR	COREx_HP_SYSREG_INT_MAP_REG	14	
111	脉冲计数控制器 (PCNT)	PCNT_INTR	COREx_PCNT_INT_MAP_REG	15	
112	n/a	保留	保留	16	
113	并行 IO 控制器 (PARLIO)	HP_PARLIO_RX_INTR	COREx_HP_PARLIO_RX_INT_MAP_REG	17	
114	并行 IO 控制器 (PARLIO)	HP_PARLIO_TX_INTR	COREx_HP_PARLIO_TX_INT_MAP_REG	18	
115	H264 编码器	H264_DMA_OUT_CH0_INTR	COREx_H264_DMA_OUT_CH0_INT_MAP_REG	19	
116	H264 编码器	H264_DMA_OUT_CH1_INTR	COREx_H264_DMA_OUT_CH1_INT_MAP_REG	20	
117	H264 编码器	H264_DMA_OUT_CH2_INTR	COREx_H264_DMA_OUT_CH2_INT_MAP_REG	21	
118	H264 编码器	H264_DMA_OUT_CH3_INTR	COREx_H264_DMA_OUT_CH3_INT_MAP_REG	22	
119	H264 编码器	H264_DMA_OUT_CH4_INTR	COREx_H264_DMA_OUT_CH4_INT_MAP_REG	23	
120	H264 编码器	H264_DMA_IN_CH0_INTR	COREx_H264_DMA_IN_CH0_INT_MAP_REG	24	
121	H264 编码器	H264_DMA_IN_CH1_INTR	COREx_H264_DMA_IN_CH1_INT_MAP_REG	25	
122	H264 编码器	H264_DMA_IN_CH2_INTR	COREx_H264_DMA_IN_CH2_INT_MAP_REG	26	
123	H264 编码器	H264_DMA_IN_CH3_INTR	COREx_H264_DMA_IN_CH3_INT_MAP_REG	27	
124	H264 编码器	H264_DMA_IN_CH4_INTR	COREx_H264_DMA_IN_CH4_INT_MAP_REG	28	
125	H264 编码器	H264_DMA_IN_CH5_INTR	COREx_H264_DMA_IN_CH5_INT_MAP_REG	29	
126	H264 编码器	H264_REG_INTR	COREx_H264_REG_INT_MAP_REG	30	
127	辅助调试	ASSIST_DEBUG_INTR	COREx_ASSIST_DEBUG_INT_MAP_REG	31	

No.	章节	中断源	中断源映射寄存器	中断状态寄存器 位	中断状态寄存器 名称
128	*2D-DMA 控制器 (2D-DMA)*	DMA2D_IN_CH2_INTR	COREx_DMA2D_IN_CH2_INT_MAP_REG	0	COREx_INTR_STATUS_REG_4_REG
129	*2D-DMA 控制器 (2D-DMA)*	DMA2D_OUT_CH3_INTR	COREx_DMA2D_OUT_CH3_INT_MAP_REG	1	
130	*AXI 性能监测器*	AXI_PERF_MON_INTR	COREx_AXI_PERF_MON_INT_MAP_REG	3	

13.4.2 分配外部中断源至 HP CPU 外部中断

在本小节中，我们将使用以下术语描述中断矩阵相关操作：

- *SOURCE*：代表某个外部中断源，详见表 13.4-1。
- COREx_*SOURCE*_MAP：HP CPU*x* 外部中断源 (*SOURCE*) 的中断源映射寄存器。
- Num_P：HP CPU 中断编号，范围：16 ~ 47。
- Interrupt_P：表示中断号为 Num_P 的 HP CPU 中断。

13.4.2.1 分配一个外部中断源 *SOURCE* 至 HP CPU*x* 外部中断

将外部中断源 *SOURCE* 对应的寄存器 COREx_*SOURCE*_MAP 配成 Num_P，即可将该中断源分配至序号为 Num_P 的 HP CPU*x* 中断 (Interrupt_P)。

13.4.2.2 分配多个外部中断源 *SOURCE* 至 HP CPU*x* 外部中断

将各个中断源对应的寄存器 COREx_*SOURCE*_MAP 均配置成相同的 Num_P，即可将多个中断源 *SOURCE* 分配至同一 HP CPU*x* 外部中断 Interrupt_P。上述任一外设中断均会触发 HP CPU*x* 外部中断 Interrupt_P。待中断触发后，HP CPU*x* 需查询中断状态寄存器，判断产生中断的外设。更多信息，见章节 2 高性能处理器。

13.4.2.3 关闭 HP CPU*x* 外部中断源 *SOURCE*

将中断源对应的寄存器 COREx_*SOURCE*_MAP 配置成 0，即可关闭外部中断源。

13.4.3 查询 HP CPU*x* 外部中断源当前的中断状态

配置开启外设中断源后，读取寄存器 COREx_INTR_STATUS_*n*（只读）中特定位的值可以获取 HP CPU*x* 外部中断源当前的中断状态。寄存器 COREx_INTR_STATUS_*n* 与外部中断源的对应关系如表 13.4-1 所示。

13.4.4 中断重映射

ESP32-P4 的 RISC-V HP CPU*x*（x 可为 0 或 1）支持机器模式（Machine Mode）和用户模式（User Mode）。

HP CPU*x* 一共有 32 个外部中断，每个外部中断均可被 HP CPU*x* 委托为机器模式或用户模式中断。

在正常情况下，当 HP CPU*x* 运行在机器模式时，仅能响应机器模式中断，无法响应用户模式中断。当 HP CPU*x* 运行在用户模式时，可以响应机器模式和用户模式的所有中断。

为支持 HP CPU*x* 在机器模式下间接响应用户模式中断，系统引入了中断重映射（Interrupt Remap）机制。

中断重映射是指将 32 个 HP CPU0/HP CPU1 外部中断 (信号) 中，原本委托给用户模式的中断，根据软件配置，统一重新映射到一个指定的机器模式中断上。通过该方式，间接实现 HP CPU0/HP CPU1 在机器模式下响应用户模式中断。

中断重映射功能特性：

- 使能中断重映射功能：每个外部中断源都可独立配置是否参与中断重映射。如将 COREx_*SOURCE*_SRC_PASS_IN_SEC 置 1 使能中断矩阵中断源 *SOURCE* 的中断重映射功能。
- 指定目标机器模式中断：所有被重映射的用户模式中断信号，均通过 COREx_INTR_SIG_IDX_ASSERT_IN_SEC_REG 统一映射至同一个指定的机器模式中断上。如

COREx_INTR_SIG_IDX_ASSERT_IN_SEC_REG 写入值 n，选择将 HP CPUx 用户模式中断信号重映射到中断号为 n 的机器模式中断。

- 重映射相关寄存器的访问限制: 以下与重映射相关的寄存器及字段，仅允许 HP CPUx 在机器模式下写入：
  - COREx_INTR_SIG_IDX_ASSERT_IN_SEC_REG
  - COREx_*SOURCE*_SRC_PASS_IN_SEC
  - COREx_*SOURCE*_SRC_IN_SEC_FLAG
  - 当 COREx_*SOURCE*_SRC_IN_SEC_FLAG 为 1 时，COREx_*SOURCE*_MAP 也仅可在机器模式下写入。

当中断源 *SOURCE* 已映射至 COREx 外部中断信号 *n* 时，仅当同时满足以下条件，中断源 *SOURCE* 才会发生中断重映射：

- COREx_*SOURCE*_SRC_PASS_IN_SEC 被置为 1
- COREx 当前运行在机器模式
- COREx 的外部中断信号 *n* 在 CPU 内部被委托给用户模式
- COREx_INTR_SIG_IDX_ASSERT_IN_SEC_REG 所指定的目标中断信号，在 CPU 内部被委托给机器模式

13.5 寄存器列表

本小节的所有地址均为相对于中断矩阵 HP CPU0/HP CPU1 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

13.5.1 HP CPU0 中断矩阵寄存器列表

名称	描述	地址	访问
COREO_LP_RTC_INT_MAP_REG	LP_RTC_INTR 映射寄存器	0x0000	R/W
COREO_LP_WDT_INT_MAP_REG	LP_WDT_INTR 映射寄存器	0x0004	R/W
COREO_LP_TIMER_REG_0_INT_MAP_REG	LP_TIMER_REG_0_INTR 映射寄存器	0x0008	R/W
COREO_LP_TIMER_REG_1_INT_MAP_REG	LP_TIMER_REG_1_INTR 映射寄存器	0x000C	R/W
COREO_MB_HP_INT_MAP_REG	MB_HP_INTR 映射寄存器	0x0010	R/W
COREO_MB_LP_INT_MAP_REG	MB_LP_INTR 映射寄存器	0x0014	R/W
COREO_PMU_REG_0_INT_MAP_REG	PMU_REG_0_INTR 映射寄存器	0x0018	R/W
COREO_PMU_REG_1_INT_MAP_REG	PMU_REG_1_INTR 映射寄存器	0x001C	R/W
COREO_LP_ANAPERI_INT_MAP_REG	LP_ANAPERI_INTR 映射寄存器	0x0020	R/W
COREO_LP_ADC_INT_MAP_REG	LP_ADC_INTR 映射寄存器	0x0024	R/W
COREO_LP_GPIO_INT_MAP_REG	LP_GPIO_INTR 映射寄存器	0x0028	R/W
COREO_LP_I2C_INT_MAP_REG	LP_I2C_INTR 映射寄存器	0x002C	R/W
COREO_LP_I2S_INT_MAP_REG	LP_I2S_INTR 映射寄存器	0x0030	R/W
COREO_LP_SPI_INT_MAP_REG	LP_SPI_INTR 映射寄存器	0x0034	R/W
COREO_LP_TOUCH_INT_MAP_REG	LP_TOUCH_INTR 映射寄存器	0x0038	R/W
COREO_LP_TSENS_INT_MAP_REG	LP_TSENS_INTR 映射寄存器	0x003C	R/W
COREO_LP_UART_INT_MAP_REG	LP_UART_INTR 映射寄存器	0x0040	R/W
COREO_LP_EFUSE_INT_MAP_REG	LP_EFUSE_INTR 映射寄存器	0x0044	R/W
COREO_LP_SW_INT_MAP_REG	LP_SW_INTR 映射寄存器	0x0048	R/W
COREO_LP_SYSREG_INT_MAP_REG	LP_SYSREG_INTR 映射寄存器	0x004C	R/W
COREO_LP_HUK_INT_MAP_REG	LP_HUK_INTR 映射寄存器	0x0050	R/W

GoBack

GoBack

名称	描述	地址	访问
COREO_SYS_ICM_INT_MAP_REG	SYS_ICM_INTR 映射寄存器	0x0054	R/W
COREO_USB_DEVICE_INT_MAP_REG	USB_DEVICE_INTR 映射寄存器	0x0058	R/W
COREO_SDIO_HOST_INT_MAP_REG	SDIO_HOST_INTR 映射寄存器	0x005C	R/W
COREO_GDMA_INT_MAP_REG	GDMA_INTR 映射寄存器	0x0060	R/W
COREO_SPI2_INT_MAP_REG	SPI2_INTR 映射寄存器	0x0064	R/W
COREO_SPI3_INT_MAP_REG	SPI3_INTR 映射寄存器	0x0068	R/W
COREO_I2S0_INT_MAP_REG	I2S0_INTR 映射寄存器	0x006C	R/W
COREO_I2S1_INT_MAP_REG	I2S1_INTR 映射寄存器	0x0070	R/W
COREO_I2S2_INT_MAP_REG	I2S2_INTR 映射寄存器	0x0074	R/W
COREO_UHCI0_INT_MAP_REG	UHCI0_INTR 映射寄存器	0x0078	R/W
COREO_UART0_INT_MAP_REG	UART0_INTR 映射寄存器	0x007C	R/W
COREO_UART1_INT_MAP_REG	UART1_INTR 映射寄存器	0x0080	R/W
COREO_UART2_INT_MAP_REG	UART2_INTR 映射寄存器	0x0084	R/W
COREO_UART3_INT_MAP_REG	UART3_INTR 映射寄存器	0x0088	R/W
COREO_UART4_INT_MAP_REG	UART4_INTR 映射寄存器	0x008C	R/W
COREO_LCD_CAM_INT_MAP_REG	LCD_CAM_INTR 映射寄存器	0x0090	R/W
COREO_ADC_INT_MAP_REG	ADC_INTR 映射寄存器	0x0094	R/W
COREO_PWM0_INT_MAP_REG	PWM0_INTR 映射寄存器	0x0098	R/W
COREO_PWM1_INT_MAP_REG	PWM1_INTR 映射寄存器	0x009C	R/W
COREO_TWAI0_INT_MAP_REG	TWAI0_INTR 映射寄存器	0x00A0	R/W
COREO_TWAI1_INT_MAP_REG	TWAI1_INTR 映射寄存器	0x00A4	R/W
COREO_TWAI2_INT_MAP_REG	TWAI2_INTR 映射寄存器	0x00A8	R/W
COREO_RMT_INT_MAP_REG	RMT_INTR 映射寄存器	0x00AC	R/W
COREO_I2C0_INT_MAP_REG	I2C0_INTR 映射寄存器	0x00B0	R/W
COREO_I2C1_INT_MAP_REG	I2C1_INTR 映射寄存器	0x00B4	R/W
COREO_TIMERGRP0_T0_INT_MAP_REG	TIMERGRP0_T0_INTR 映射寄存器	0x00B8	R/W
COREO_TIMERGRP0_T1_INT_MAP_REG	TIMERGRP0_T1_INTR 映射寄存器	0x00BC	R/W
COREO_TIMERGRP0_WDT_INT_MAP_REG	TIMERGRP0_WDT_INTR 映射寄存器	0x00C0	R/W
COREO_TIMERGRP1_T0_INT_MAP_REG	TIMERGRP1_T0_INTR 映射寄存器	0x00C4	R/W

名称	描述	地址	访问
COREO_TIMERGRP1_T1_INT_MAP_REG	TIMERGRP1_T1_INTR 映射寄存器	0x00C8	R/W
COREO_TIMERGRP1_WDT_INT_MAP_REG	TIMERGRP1_WDT_INTR 映射寄存器	0x00CC	R/W
COREO_LEDC_INT_MAP_REG	LEDC_INTR 映射寄存器	0x00D0	R/W
COREO_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INTR 映射寄存器	0x00D4	R/W
COREO_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INTR 映射寄存器	0x00D8	R/W
COREO_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INTR 映射寄存器	0x00DC	R/W
COREO_AHB_PDMA_IN_CH0_INT_MAP_REG	AHB_PDMA_IN_CH0_INTR 映射寄存器	0x00E0	R/W
COREO_AHB_PDMA_IN_CH1_INT_MAP_REG	AHB_PDMA_IN_CH1_INTR 映射寄存器	0x00E4	R/W
COREO_AHB_PDMA_IN_CH2_INT_MAP_REG	AHB_PDMA_IN_CH2_INTR 映射寄存器	0x00E8	R/W
COREO_AHB_PDMA_OUT_CH0_INT_MAP_REG	AHB_PDMA_OUT_CH0_INTR 映射寄存器	0x00EC	R/W
COREO_AHB_PDMA_OUT_CH1_INT_MAP_REG	AHB_PDMA_OUT_CH1_INTR 映射寄存器	0x00F0	R/W
COREO_AHB_PDMA_OUT_CH2_INT_MAP_REG	AHB_PDMA_OUT_CH2_INTR 映射寄存器	0x00F4	R/W
COREO_AXI_PDMA_IN_CH0_INT_MAP_REG	AXI_PDMA_IN_CH0_INTR 映射寄存器	0x00F8	R/W
COREO_AXI_PDMA_IN_CH1_INT_MAP_REG	AXI_PDMA_IN_CH1_INTR 映射寄存器	0x00FC	R/W
COREO_AXI_PDMA_IN_CH2_INT_MAP_REG	AXI_PDMA_IN_CH2_INTR 映射寄存器	0x0100	R/W
COREO_AXI_PDMA_OUT_CH0_INT_MAP_REG	AXI_PDMA_OUT_CH0_INTR 映射寄存器	0x0104	R/W
COREO_AXI_PDMA_OUT_CH1_INT_MAP_REG	AXI_PDMA_OUT_CH1_INTR 映射寄存器	0x0108	R/W
COREO_AXI_PDMA_OUT_CH2_INT_MAP_REG	AXI_PDMA_OUT_CH2_INTR 映射寄存器	0x010C	R/W
COREO_RSA_INT_MAP_REG	RSA_INTR 映射寄存器	0x0110	R/W
COREO_AES_INT_MAP_REG	AES_INTR 映射寄存器	0x0114	R/W
COREO_SHA_INT_MAP_REG	SHA_INTR 映射寄存器	0x0118	R/W
COREO_ECC_INT_MAP_REG	ECC_INTR 映射寄存器	0x011C	R/W
COREO_ECDSA_INT_MAP_REG	ECDSA_INTR 映射寄存器	0x0120	R/W
COREO_KM_INT_MAP_REG	KM_INTR 映射寄存器	0x0124	R/W
COREO_GPIO_INT0_MAP_REG	GPIO_INTR0 映射寄存器	0x0128	R/W
COREO_GPIO_INT1_MAP_REG	GPIO_INTR1 映射寄存器	0x012C	R/W

GoBack

GoBack

名称	描述	地址	访问
COREO_GPIO_INT2_MAP_REG	GPIO_INTR2 映射寄存器	0x0130	R/W
COREO_GPIO_INT3_MAP_REG	GPIO_INTR3 映射寄存器	0x0134	R/W
COREO_GPIO_PAD_COMP_INT_MAP_REG	GPIO_PAD_COMP_INTR 映射寄存器	0x0138	R/W
COREO_CPU_INT_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 映射寄存器	0x013C	R/W
COREO_CPU_INT_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 映射寄存器	0x0140	R/W
COREO_CPU_INT_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 映射寄存器	0x0144	R/W
COREO_CPU_INT_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 映射寄存器	0x0148	R/W
COREO_FLASH_MSPI_INT_MAP_REG	FLASH_MSPI_INTR 映射寄存器	0x0150	R/W
COREO_CSI_BRIDGE_INT_MAP_REG	CSI_BRIDGE_INTR 映射寄存器	0x0154	R/W
COREO_DSI_BRIDGE_INT_MAP_REG	DSI_BRIDGE_INTR 映射寄存器	0x0158	R/W
COREO_CSI_INT_MAP_REG	CSI_INTR 映射寄存器	0x015C	R/W
COREO_DSI_INT_MAP_REG	DSI_INTR 映射寄存器	0x0160	R/W
COREO_GMII_PHY_INT_MAP_REG	GMII_PHY_INTR 映射寄存器	0x0164	R/W
COREO_LPI_INT_MAP_REG	LPI_INTR 映射寄存器	0x0168	R/W
COREO_PMT_INT_MAP_REG	PMT_INTR 映射寄存器	0x016C	R/W
COREO_ETH_MAC_INT_MAP_REG	ETH_MAC_INTR 映射寄存器	0x0170	R/W
COREO_USB_OTG_INT_MAP_REG	USB_OTG_INTR 映射寄存器	0x0174	R/W
COREO_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	USB_OTG_ENDP_MULTI_PROC_INTR 映射寄存器	0x0178	R/W
COREO_JPEG_INT_MAP_REG	JPEG_INTR 映射寄存器	0x017C	R/W
COREO_PPA_INT_MAP_REG	PPA_INTR 映射寄存器	0x0180	R/W
COREO_COREO_TRACE_INT_MAP_REG	COREO_TRACE_INTR 映射寄存器	0x0184	R/W
COREO_CORE1_TRACE_INT_MAP_REG	CORE1_TRACE_INTR 映射寄存器	0x0188	R/W
COREO_ISP_INT_MAP_REG	ISP_INTR 映射寄存器	0x0190	R/W
COREO_I3C_MST_INT_MAP_REG	I3C_MST_INTR 映射寄存器	0x0194	R/W
COREO_I3C_SLV_INT_MAP_REG	I3C_SLV_INTR 映射寄存器	0x0198	R/W
COREO_USB_OTG11_INT_MAP_REG	USB_OTG11_INTR 映射寄存器	0x019C	R/W
COREO_DMA2D_IN_CH0_INT_MAP_REG	DMA2D_IN_CH0_INTR 映射寄存器	0x01A0	R/W
COREO_DMA2D_IN_CH1_INT_MAP_REG	DMA2D_IN_CH1_INTR 映射寄存器	0x01A4	R/W

GoBack

名称	描述	地址	访问
COREO_DMA2D_OUT_CHO_INT_MAP_REG	DMA2D_OUT_CHO_INTR 映射寄存器	0x01A8	R/W
COREO_DMA2D_OUT_CH1_INT_MAP_REG	DMA2D_OUT_CH1_INTR 映射寄存器	0x01AC	R/W
COREO_DMA2D_OUT_CH2_INT_MAP_REG	DMA2D_OUT_CH2_INTR 映射寄存器	0x01B0	R/W
COREO_PSRAM_MSPI_INT_MAP_REG	PSRAM_MSPI_INTR 映射寄存器	0x01B4	R/W
COREO_HP_SYSREG_INT_MAP_REG	HP_SYSREG_INTR 映射寄存器	0x01B8	R/W
COREO_PCNT_INT_MAP_REG	PCNT_INTR 映射寄存器	0x01BC	R/W
COREO_HP_PARLIO_RX_INT_MAP_REG	HP_PARLIO_RX_INTR 映射寄存器	0x01C4	R/W
COREO_HP_PARLIO_TX_INT_MAP_REG	HP_PARLIO_TX_INTR 映射寄存器	0x01C8	R/W
COREO_H264_DMA2D_OUT_CHO_INT_MAP_REG	H264_DMA2D_OUT_CHO_INTR 映射寄存器	0x01CC	R/W
COREO_H264_DMA2D_OUT_CH1_INT_MAP_REG	H264_DMA2D_OUT_CH1_INTR 映射寄存器	0x01D0	R/W
COREO_H264_DMA2D_OUT_CH2_INT_MAP_REG	H264_DMA2D_OUT_CH2_INTR 映射寄存器	0x01D4	R/W
COREO_H264_DMA2D_OUT_CH3_INT_MAP_REG	H264_DMA2D_OUT_CH3_INTR 映射寄存器	0x01D8	R/W
COREO_H264_DMA2D_OUT_CH4_INT_MAP_REG	H264_DMA2D_OUT_CH4_INTR 映射寄存器	0x01DC	R/W
COREO_H264_DMA2D_IN_CHO_INT_MAP_REG	H264_DMA2D_IN_CHO_INTR 映射寄存器	0x01E0	R/W
COREO_H264_DMA2D_IN_CH1_INT_MAP_REG	H264_DMA2D_IN_CH1_INTR 映射寄存器	0x01E4	R/W
COREO_H264_DMA2D_IN_CH2_INT_MAP_REG	H264_DMA2D_IN_CH2_INTR 映射寄存器	0x01E8	R/W
COREO_H264_DMA2D_IN_CH3_INT_MAP_REG	H264_DMA2D_IN_CH3_INTR 映射寄存器	0x01EC	R/W
COREO_H264_DMA2D_IN_CH4_INT_MAP_REG	H264_DMA2D_IN_CH4_INTR 映射寄存器	0x01F0	R/W

名称	描述	地址	访问
COREO_H264_DMA2D_IN_CH5_INT_MAP_REG	H264_DMA2D_IN_CH5_INTR 映射寄存器	0x01F4	R/W
COREO_H264_REG_INT_MAP_REG	H264_REG_INTR 映射寄存器	0x01F8	R/W
COREO_ASSIST_DEBUG_INT_MAP_REG	ASSIST_DEBUG_INTR 映射寄存器	0x01FC	R/W
COREO_INTR_STATUS_REG_0_REG	中断源 0 ~ 31 的状态寄存器	0x0200	RO
COREO_INTR_STATUS_REG_1_REG	中断源 32 ~ 63 的状态寄存器	0x0204	RO
COREO_INTR_STATUS_REG_2_REG	中断源 64 ~ 95 的状态寄存器	0x0208	RO
COREO_INTR_STATUS_REG_3_REG	中断源 96 ~ 127 的状态寄存器	0x020C	RO
COREO_CLOCK_GATE_REG	时钟门控寄存器	0x0210	R/W
COREO_DMA2D_IN_CH2_INT_MAP_REG	DMA2D_IN_CH2_INTR 映射寄存器	0x214	R/W
COREO_DMA2D_OUT_CH3_INT_MAP_REG	DMA2D_OUT_CH3_INTR 映射寄存器	0x218	R/W
COREO_AXI_PERF_MON_INT_MAP_REG	AXI_PERF_MON_INTR 映射寄存器	0x21C	R/W
COREO_INTR_STATUS_REG_4_REG	中断源 128 ~ 130 的状态寄存器	0x0220	RO
COREO_INTR_SIG_IDX_ASSERT_IN_SEC_REG	CORE0 中断委托配置寄存器	0x0228	R/W
COREO_INTR_SEC_STATUS_REG	CORE0 中断委托状态寄存器	0x022C	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_0_REG	中断源 0 ~ 31 的中断委托状态寄存器	0x0230	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_1_REG	中断源 32 ~ 63 的中断委托状态寄存器	0x0234	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_2_REG	中断源 64 ~ 95 的中断委托状态寄存器	0x0238	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_3_REG	中断源 96 ~ 127 的中断委托状态寄存器	0x023C	RO
COREO_INTR_SRC_PAS_IN_SEC_STATUS_4_REG	中断源 128 ~ 130 的中断委托状态寄存器	0x0240	RO
COREO_INTERRUPT_REG_DATE_REG	版本控制寄存器	0x03FC	R/W

13.5.2 HP CPU1 中断矩阵寄存器列表

名称	描述	地址	访问
CORE1_LP_RTC_INT_MAP_REG	LP_RTC_INTR 映射寄存器	0x0000	R/W
CORE1_LP_WDT_INT_MAP_REG	LP_WDT_INTR 映射寄存器	0x0004	R/W
CORE1_LP_TIMER_REG_0_INT_MAP_REG	LP_TIMER_REG_0_INTR 映射寄存器	0x0008	R/W

GoBack

名称	描述	地址	访问
CORE1_LP_TIMER_REG_1_INT_MAP_REG	LP_TIMER_REG_1_INTR 映射寄存器	0x000C	R/W
CORE1_MB_HP_INT_MAP_REG	MB_HP_INTR 映射寄存器	0x0010	R/W
CORE1_MB_LP_INT_MAP_REG	MB_LP_INTR 映射寄存器	0x0014	R/W
CORE1_PMU_REG_0_INT_MAP_REG	PMU_REG_0_INTR 映射寄存器	0x0018	R/W
CORE1_PMU_REG_1_INT_MAP_REG	PMU_REG_1_INTR 映射寄存器	0x001C	R/W
CORE1_LP_ANAPERI_INT_MAP_REG	LP_ANAPERI_INTR 映射寄存器	0x0020	R/W
CORE1_LP_ADC_INT_MAP_REG	LP_ADC_INTR 映射寄存器	0x0024	R/W
CORE1_LP_GPIO_INT_MAP_REG	LP_GPIO_INTR 映射寄存器	0x0028	R/W
CORE1_LP_I2C_INT_MAP_REG	LP_I2C_INTR 映射寄存器	0x002C	R/W
CORE1_LP_I2S_INT_MAP_REG	LP_I2S_INTR 映射寄存器	0x0030	R/W
CORE1_LP_SPI_INT_MAP_REG	LP_SPI_INTR 映射寄存器	0x0034	R/W
CORE1_LP_TOUCH_INT_MAP_REG	LP_TOUCH_INTR 映射寄存器	0x0038	R/W
CORE1_LP_TSENS_INT_MAP_REG	LP_TSENS_INTR 映射寄存器	0x003C	R/W
CORE1_LP_UART_INT_MAP_REG	LP_UART_INTR 映射寄存器	0x0040	R/W
CORE1_LP_EFUSE_INT_MAP_REG	LP_EFUSE_INTR 映射寄存器	0x0044	R/W
CORE1_LP_SW_INT_MAP_REG	LP_SW_INTR 映射寄存器	0x0048	R/W
CORE1_LP_SYSREG_INT_MAP_REG	LP_SYSREG_INTR 映射寄存器	0x004C	R/W
CORE1_LP_HUK_INT_MAP_REG	LP_HUK_INTR 映射寄存器	0x0050	R/W
CORE1_SYS_ICM_INT_MAP_REG	SYS_ICM_INTR 映射寄存器	0x0054	R/W
CORE1_USB_DEVICE_INT_MAP_REG	USB_DEVICE_INTR 映射寄存器	0x0058	R/W
CORE1_SDIO_HOST_INT_MAP_REG	SDIO_HOST_INTR 映射寄存器	0x005C	R/W
CORE1_GDMA_INT_MAP_REG	GDMA_INTR 映射寄存器	0x0060	R/W
CORE1_SPI2_INT_MAP_REG	SPI2_INTR 映射寄存器	0x0064	R/W
CORE1_SPI3_INT_MAP_REG	SPI3_INTR 映射寄存器	0x0068	R/W
CORE1_I2S0_INT_MAP_REG	I2S0_INTR 映射寄存器	0x006C	R/W
CORE1_I2S1_INT_MAP_REG	I2S1_INTR 映射寄存器	0x0070	R/W
CORE1_I2S2_INT_MAP_REG	I2S2_INTR 映射寄存器	0x0074	R/W
CORE1_UHCI0_INT_MAP_REG	UHCI0_INTR 映射寄存器	0x0078	R/W
CORE1_UART0_INT_MAP_REG	UART0_INTR 映射寄存器	0x007C	R/W

GoBack

名称	描述	地址	访问
CORE1_UART1_INT_MAP_REG	UART1_INTR 映射寄存器	0x0080	R/W
CORE1_UART2_INT_MAP_REG	UART2_INTR 映射寄存器	0x0084	R/W
CORE1_UART3_INT_MAP_REG	UART3_INTR 映射寄存器	0x0088	R/W
CORE1_UART4_INT_MAP_REG	UART4_INTR 映射寄存器	0x008C	R/W
CORE1_LCD_CAM_INT_MAP_REG	LCD_CAM_INTR 映射寄存器	0x0090	R/W
CORE1_ADC_INT_MAP_REG	ADC_INTR 映射寄存器	0x0094	R/W
CORE1_PWM0_INT_MAP_REG	PWM0_INTR 映射寄存器	0x0098	R/W
CORE1_PWM1_INT_MAP_REG	PWM1_INTR 映射寄存器	0x009C	R/W
CORE1_TWAI0_INT_MAP_REG	TWAI0_INTR 映射寄存器	0x00A0	R/W
CORE1_TWAI1_INT_MAP_REG	TWAI1_INTR 映射寄存器	0x00A4	R/W
CORE1_TWAI2_INT_MAP_REG	TWAI2_INTR 映射寄存器	0x00A8	R/W
CORE1_RMT_INT_MAP_REG	RMT_INTR 映射寄存器	0x00AC	R/W
CORE1_I2C0_INT_MAP_REG	I2C0_INTR 映射寄存器	0x00B0	R/W
CORE1_I2C1_INT_MAP_REG	I2C1_INTR 映射寄存器	0x00B4	R/W
CORE1_TIMERGRP0_T0_INT_MAP_REG	TIMERGRP0_T0_INTR 映射寄存器	0x00B8	R/W
CORE1_TIMERGRP0_T1_INT_MAP_REG	TIMERGRP0_T1_INTR 映射寄存器	0x00BC	R/W
CORE1_TIMERGRP0_WDT_INT_MAP_REG	TIMERGRP0_WDT_INTR 映射寄存器	0x00C0	R/W
CORE1_TIMERGRP1_T0_INT_MAP_REG	TIMERGRP1_T0_INTR 映射寄存器	0x00C4	R/W
CORE1_TIMERGRP1_T1_INT_MAP_REG	TIMERGRP1_T1_INTR 映射寄存器	0x00C8	R/W
CORE1_TIMERGRP1_WDT_INT_MAP_REG	TIMERGRP1_WDT_INTR 映射寄存器	0x00CC	R/W
CORE1_LEDC_INT_MAP_REG	LEDC_INTR 映射寄存器	0x00D0	R/W
CORE1_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INTR 映射寄存器	0x00D4	R/W
CORE1_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INTR 映射寄存器	0x00D8	R/W
CORE1_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INTR 映射寄存器	0x00DC	R/W
CORE1_AHB_PDMA_IN_CH0_INT_MAP_REG	AHB_PDMA_IN_CH0_INTR 映射寄存器	0x00E0	R/W
CORE1_AHB_PDMA_IN_CH1_INT_MAP_REG	AHB_PDMA_IN_CH1_INTR 映射寄存器	0x00E4	R/W
CORE1_AHB_PDMA_IN_CH2_INT_MAP_REG	AHB_PDMA_IN_CH2_INTR 映射寄存器	0x00E8	R/W
CORE1_AHB_PDMA_OUT_CH0_INT_MAP_REG	AHB_PDMA_OUT_CH0_INTR 映射寄存器	0x00EC	R/W

名称	描述	地址	访问
CORE1_AHB_PDMA_OUT_CH1_INT_MAP_REG	AHB_PDMA_OUT_CH1_INTR 映射寄存器	0x00F0	R/W
CORE1_AHB_PDMA_OUT_CH2_INT_MAP_REG	AHB_PDMA_OUT_CH2_INTR 映射寄存器	0x00F4	R/W
CORE1_AXI_PDMA_IN_CH0_INT_MAP_REG	AXI_PDMA_IN_CH0_INTR 映射寄存器	0x00F8	R/W
CORE1_AXI_PDMA_IN_CH1_INT_MAP_REG	AXI_PDMA_IN_CH1_INTR 映射寄存器	0x00FC	R/W
CORE1_AXI_PDMA_IN_CH2_INT_MAP_REG	AXI_PDMA_IN_CH2_INTR 映射寄存器	0x0100	R/W
CORE1_AXI_PDMA_OUT_CH0_INT_MAP_REG	AXI_PDMA_OUT_CH0_INTR 映射寄存器	0x0104	R/W
CORE1_AXI_PDMA_OUT_CH1_INT_MAP_REG	AXI_PDMA_OUT_CH1_INTR 映射寄存器	0x0108	R/W
CORE1_AXI_PDMA_OUT_CH2_INT_MAP_REG	AXI_PDMA_OUT_CH2_INTR 映射寄存器	0x010C	R/W
CORE1_RSA_INT_MAP_REG	RSA_INTR 映射寄存器	0x0110	R/W
CORE1_AES_INT_MAP_REG	AES_INTR 映射寄存器	0x0114	R/W
CORE1_SHA_INT_MAP_REG	SHA_INTR 映射寄存器	0x0118	R/W
CORE1_ECC_INT_MAP_REG	ECC_INTR 映射寄存器	0x011C	R/W
CORE1_ECDSA_INT_MAP_REG	ECDSA_INTR 映射寄存器	0x0120	R/W
CORE1_KM_INT_MAP_REG	KM_INTR 映射寄存器	0x0124	R/W
CORE1_GPIO_INT0_MAP_REG	GPIO_INTR0 映射寄存器	0x0128	R/W
CORE1_GPIO_INT1_MAP_REG	GPIO_INTR1 映射寄存器	0x012C	R/W
CORE1_GPIO_INT2_MAP_REG	GPIO_INTR2 映射寄存器	0x0130	R/W
CORE1_GPIO_INT3_MAP_REG	GPIO_INTR3 映射寄存器	0x0134	R/W
CORE1_GPIO_PAD_COMP_INT_MAP_REG	GPIO_PAD_COMP_INTR 映射寄存器	0x0138	R/W
CORE1_CPU_INT_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 映射寄存器	0x013C	R/W
CORE1_CPU_INT_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 映射寄存器	0x0140	R/W
CORE1_CPU_INT_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 映射寄存器	0x0144	R/W
CORE1_CPU_INT_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 映射寄存器	0x0148	R/W
CORE1_FLASH_MSPI_INT_MAP_REG	FLASH_MSPI_INTR 映射寄存器	0x0150	R/W
CORE1_CSI_BRIDGE_INT_MAP_REG	CSI_BRIDGE_INTR 映射寄存器	0x0154	R/W
CORE1_DSI_BRIDGE_INT_MAP_REG	DSI_BRIDGE_INTR 映射寄存器	0x0158	R/W
CORE1_CSI_INT_MAP_REG	CSI_INTR 映射寄存器	0x015C	R/W

GoBack

名称	描述	地址	访问
CORE1_DSI_INT_MAP_REG	DSI_INTR 映射寄存器	0x0160	R/W
CORE1_GMII_PHY_INT_MAP_REG	GMII_PHY_INTR 映射寄存器	0x0164	R/W
CORE1_LPI_INT_MAP_REG	LPI_INTR 映射寄存器	0x0168	R/W
CORE1_PMT_INT_MAP_REG	PMT_INTR 映射寄存器	0x016C	R/W
CORE1_ETH_MAC_INT_MAP_REG	ETH_MAC_INTR 映射寄存器	0x0170	R/W
CORE1_USB_OTG_INT_MAP_REG	USB_OTG_INTR 映射寄存器	0x0174	R/W
CORE1_USB_OTG_ENDP_MULTI_PROC_INT_MAP_REG	USB_OTG_ENDP_MULTI_PROC_INTR 映射寄存器	0x0178	R/W
CORE1_JPEG_INT_MAP_REG	JPEG_INTR 映射寄存器	0x017C	R/W
CORE1_PPA_INT_MAP_REG	PPA_INTR 映射寄存器	0x0180	R/W
CORE1_CORE0_TRACE_INT_MAP_REG	CORE0_TRACE_INTR 映射寄存器	0x0184	R/W
CORE1_CORE1_TRACE_INT_MAP_REG	CORE1_TRACE_INTR 映射寄存器	0x0188	R/W
CORE1_ISP_INT_MAP_REG	ISP_INTR 映射寄存器	0x0190	R/W
CORE1_I3C_MST_INT_MAP_REG	I3C_MST_INTR 映射寄存器	0x0194	R/W
CORE1_I3C_SLV_INT_MAP_REG	I3C_SLV_INTR 映射寄存器	0x0198	R/W
CORE1_USB_OTG11_INT_MAP_REG	USB_OTG11_INTR 映射寄存器	0x019C	R/W
CORE1_DMA2D_IN_CH0_INT_MAP_REG	DMA2D_IN_CH0_INTR 映射寄存器	0x01A0	R/W
CORE1_DMA2D_IN_CH1_INT_MAP_REG	DMA2D_IN_CH1_INTR 映射寄存器	0x01A4	R/W
CORE1_DMA2D_OUT_CH0_INT_MAP_REG	DMA2D_OUT_CH0_INTR 映射寄存器	0x01A8	R/W
CORE1_DMA2D_OUT_CH1_INT_MAP_REG	DMA2D_OUT_CH1_INTR 映射寄存器	0x01AC	R/W
CORE1_DMA2D_OUT_CH2_INT_MAP_REG	DMA2D_OUT_CH2_INTR 映射寄存器	0x01B0	R/W
CORE1_PSRAM_MSPI_INT_MAP_REG	PSRAM_MSPI_INTR 映射寄存器	0x01B4	R/W
CORE1_HP_SYSREG_INT_MAP_REG	HP_SYSREG_INTR 映射寄存器	0x01B8	R/W
CORE1_PCNT_INT_MAP_REG	PCNT_INTR 映射寄存器	0x01BC	R/W
CORE1_HP_PARLIO_RX_INT_MAP_REG	HP_PARLIO_RX_INTR 映射寄存器	0x01C4	R/W
CORE1_HP_PARLIO_TX_INT_MAP_REG	HP_PARLIO_TX_INTR 映射寄存器	0x01C8	R/W
CORE1_H264_DMA2D_OUT_CH0_INT_MAP_REG	H264_DMA2D_OUT_CH0_INTR 映射寄存器	0x01CC	R/W

GoBack

名称	描述	地址	访问
CORE1_H264_DMA2D_OUT_CH1_INT_MAP_REG	H264_DMA2D_OUT_CH1_INTR 映射寄存器	0x01D0	R/W
CORE1_H264_DMA2D_OUT_CH2_INT_MAP_REG	H264_DMA2D_OUT_CH2_INTR 映射寄存器	0x01D4	R/W
CORE1_H264_DMA2D_OUT_CH3_INT_MAP_REG	H264_DMA2D_OUT_CH3_INTR 映射寄存器	0x01D8	R/W
CORE1_H264_DMA2D_OUT_CH4_INT_MAP_REG	H264_DMA2D_OUT_CH4_INTR 映射寄存器	0x01DC	R/W
CORE1_H264_DMA2D_IN_CH0_INT_MAP_REG	H264_DMA2D_IN_CH0_INTR 映射寄存器	0x01E0	R/W
CORE1_H264_DMA2D_IN_CH1_INT_MAP_REG	H264_DMA2D_IN_CH1_INTR 映射寄存器	0x01E4	R/W
CORE1_H264_DMA2D_IN_CH2_INT_MAP_REG	H264_DMA2D_IN_CH2_INTR 映射寄存器	0x01E8	R/W
CORE1_H264_DMA2D_IN_CH3_INT_MAP_REG	H264_DMA2D_IN_CH3_INTR 映射寄存器	0x01EC	R/W
CORE1_H264_DMA2D_IN_CH4_INT_MAP_REG	H264_DMA2D_IN_CH4_INTR 映射寄存器	0x01F0	R/W
CORE1_H264_DMA2D_IN_CH5_INT_MAP_REG	H264_DMA2D_IN_CH5_INTR 映射寄存器	0x01F4	R/W
CORE1_H264_REG_INT_MAP_REG	H264_REG_INTR 映射寄存器	0x01F8	R/W
CORE1_ASSIST_DEBUG_INT_MAP_REG	ASSIST_DEBUG_INTR 映射寄存器	0x01FC	R/W
CORE1_INTR_STATUS_REG_0_REG	中断源 0 ~ 31 的状态寄存器	0x0200	RO
CORE1_INTR_STATUS_REG_1_REG	中断源 32 ~ 63 的状态寄存器	0x0204	RO
CORE1_INTR_STATUS_REG_2_REG	中断源 64 ~ 95 的状态寄存器	0x0208	RO
CORE1_INTR_STATUS_REG_3_REG	中断源 96 ~ 127 的状态寄存器	0x020C	RO
CORE1_CLOCK_GATE_REG	时钟门控寄存器	0x0210	R/W
CORE1_DMA2D_IN_CH2_INT_MAP_REG	DMA2D_IN_CH2_INTR 映射寄存器	0x214	R/W
CORE1_DMA2D_OUT_CH3_INT_MAP_REG	DMA2D_OUT_CH3_INTR 映射寄存器	0x218	R/W

GoBack

名称	描述	地址	访问
CORE1_AXI_PERF_MON_INT_MAP_REG	AXI_PERF_MON_INTR 映射寄存器	0x21C	R/W
CORE1_INTR_STATUS_REG_4_REG	中断源 128 ~ 130 的状态寄存器	0x0220	RO
CORE1_INTR_SIG_IDX_ASSERT_IN_SEC_REG	CORE1 中断委托配置寄存器	0x0228	R/W
CORE1_INTR_SEC_STATUS_REG	CORE1 中断委托状态寄存器	0x022C	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_0_REG	中断源 0 ~ 31 的中断委托状态寄存器	0x0230	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_1_REG	中断源 32 ~ 63 的中断委托状态寄存器	0x0234	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_2_REG	中断源 64 ~ 95 的中断委托状态寄存器	0x0238	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_3_REG	中断源 96 ~ 127 的中断委托状态寄存器	0x023C	RO
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_4_REG	中断源 128 ~ 130 的中断委托状态寄存器	0x0240	RO
CORE1_INTERRUPT_REG_DATE_REG	版本控制寄存器	0x03FC	R/W

13.6 寄存器

本小节的所有地址均为相对于中断矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

13.6.1 HP CPU0 中断矩阵寄存器

Register 13.1. COREO_*LP_RTC_INT*_MAP_REG (0x0000)

Register 13.2. COREO_*LP_WDT_INT*_MAP_REG (0x0004)

Register 13.3. COREO_*LP_TIMER_REG_O_INT*_MAP_REG (0x0008)

Register 13.4. COREO_*LP_TIMER_REG_1_INT*_MAP_REG (0x000C)

Register 13.5. COREO_*MB_HP_INT*_MAP_REG (0x0010)

Register 13.6. COREO_*MB_LP_INT*_MAP_REG (0x0014)

Register 13.7. COREO_*PMU_REG_O_INT*_MAP_REG (0x0018)

Register 13.8. COREO_*PMU_REG_1_INT*_MAP_REG (0x001C)

Register 13.9. COREO_*LP_ANAPERI_INT*_MAP_REG (0x0020)

Register 13.10. COREO_*LP_ADC_INT*_MAP_REG (0x0024)

Register 13.11. COREO_*LP_GPIO_INT*_MAP_REG (0x0028)

Register 13.12. COREO_*LP_I2C_INT*_MAP_REG (0x002C)

Register 13.13. COREO_*LP_I2S_INT*_MAP_REG (0x0030)

Register 13.14. COREO_*LP_SPI_INT*_MAP_REG (0x0034)

Register 13.15. COREO_*LP_TOUCH_INT*_MAP_REG (0x0038)

Register 13.16. COREO_*LP_TSENS_INT*_MAP_REG (0x003C)

Register 13.17. COREO_*LP_UART_INT*_MAP_REG (0x0040)

Register 13.18. COREO_*LP_EFUSE_INT*_MAP_REG (0x0044)

Register 13.19. COREO_*LP_SW_INT*_MAP_REG (0x0048)

Register 13.20. COREO_*LP_SYSREG_INT*_MAP_REG (0x004C)

Register 13.21. COREO_*LP_HUK_INT*_MAP_REG (0x0050)

Register 13.22. COREO_*SYS_ICM_INT*_MAP_REG (0x0054)

Register 13.23. COREO_*USB_DEVICE_INT*_MAP_REG (0x0058)

Register 13.24. COREO_*SDIO_HOST_INT*_MAP_REG (0x005C)

Register 13.25. COREO_*GDMA_INT*_MAP_REG (0x0060)

Register 13.26. COREO_*SPI2_INT*_MAP_REG (0x0064)

Register 13.27. COREO_*SPI3_INT*_MAP_REG (0x0068)

Register 13.28. COREO_*I2SO_INT*_MAP_REG (0x006C)

Register 13.29. COREO_*I2S1_INT*_MAP_REG (0x0070)

Register 13.30. COREO_*I2S2_INT*_MAP_REG (0x0074)

Register 13.31. COREO_*UHCI0_INT*_MAP_REG (0x0078)

Register 13.32. COREO_*UART0_INT*_MAP_REG (0x007C)

Register 13.33. COREO_*UART1_INT*_MAP_REG (0x0080)

Register 13.34. COREO_*UART2_INT*_MAP_REG (0x0084)

Register 13.35. COREO_*UART3_INT*_MAP_REG (0x0088)

Register 13.36. COREO_*UART4_INT*_MAP_REG (0x008C)

Register 13.37. COREO_*LCD_CAM_INT*_MAP_REG (0x0090)

Register 13.38. COREO_*ADC_INT*_MAP_REG (0x0094)

Register 13.39. COREO_*PWM0_INT*_MAP_REG (0x0098)

Register 13.40. COREO_*PWM1_INT*_MAP_REG (0x009C)

Register 13.41. COREO_*TWAI0_INT*_MAP_REG (0x00A0)

Register 13.42. COREO_*TWAI1_INT*_MAP_REG (0x00A4)

Register 13.43. COREO_*TWAI2_INT*_MAP_REG (0x00A8)

Register 13.44. COREO_*RMT_INT*_MAP_REG (0x00AC)

Register 13.45. COREO_*I2C0_INT*_MAP_REG (0x00B0)

Register 13.46. COREO_*I2C1_INT*_MAP_REG (0x00B4)

Register 13.47. COREO_*TIMERGRP0_T0_INT*_MAP_REG (0x00B8)

Register 13.48. COREO_*TIMERGRP0_T1_INT*_MAP_REG (0x00BC)

Register 13.49. COREO_*TIMERGRP0_WDT_INT*_MAP_REG (0x00C0)

Register 13.50. COREO_*TIMERGRP1_T0_INT*_MAP_REG (0x00C4)

Register 13.51. COREO_*TIMERGRP1_T1_INT*_MAP_REG (0x00C8)

Register 13.52. COREO_*TIMERGRP1_WDT_INT*_MAP_REG (0x00CC)

Register 13.53. COREO_*LEDC_INT*_MAP_REG (0x00D0)

Register 13.54. COREO_*SYSTIMER_TARGET0_INT*_MAP_REG (0x00D4)

Register 13.55. COREO_*SYSTIMER_TARGET1_INT*_MAP_REG (0x00D8)

Register 13.56. COREO_*SYSTIMER_TARGET2_INT*_MAP_REG (0x00DC)

Register 13.57. COREO_*AHB_PDMA_IN_CH0_INT*_MAP_REG (0x00E0)

Register 13.58. COREO_*AHB_PDMA_IN_CH1_INT*_MAP_REG (0x00E4)

Register 13.59. COREO_*AHB_PDMA_IN_CH2_INT*_MAP_REG (0x00E8)

Register 13.60. COREO_*AHB_PDMA_OUT_CH0_INT*_MAP_REG (0x00EC)

Register 13.61. COREO_*AHB_PDMA_OUT_CH1_INT*_MAP_REG (0x00F0)

Register 13.62. COREO_*AHB_PDMA_OUT_CH2_INT*_MAP_REG (0x00F4)

Register 13.63. COREO_*AXI_PDMA_IN_CH0_INT*_MAP_REG (0x00F8)

Register 13.64. COREO_*AXI_PDMA_IN_CH1_INT*_MAP_REG (0x00FC)

Register 13.65. COREO_*AXI_PDMA_IN_CH2_INT*_MAP_REG (0x0100)

Register 13.66. COREO_*AXI_PDMA_OUT_CHO_INT*_MAP_REG (0x0104)

Register 13.67. COREO_*AXI_PDMA_OUT_CH1_INT*_MAP_REG (0x0108)

Register 13.68. COREO_*AXI_PDMA_OUT_CH2_INT*_MAP_REG (0x010C)

Register 13.69. COREO_*RSA_INT*_MAP_REG (0x0110)

Register 13.70. COREO_*AES_INT*_MAP_REG (0x0114)

Register 13.71. COREO_*SHA_INT*_MAP_REG (0x0118)

Register 13.72. COREO_*ECC_INT*_MAP_REG (0x011C)

Register 13.73. COREO_*ECDSA_INT*_MAP_REG (0x0120)

Register 13.74. COREO_*KM_INT*_MAP_REG (0x0124)

Register 13.75. COREO_*GPIO_INTO*_MAP_REG (0x0128)

Register 13.76. COREO_*GPIO_INT1*_MAP_REG (0x012C)

Register 13.77. COREO_*GPIO_INT2*_MAP_REG (0x0130)

Register 13.78. COREO_*GPIO_INT3*_MAP_REG (0x0134)

Register 13.79. COREO_*GPIO_PAD_COMP_INT*_MAP_REG (0x0138)

Register 13.80. COREO_*CPU_INT_FROM_CPU_0*_MAP_REG (0x013C)

Register 13.81. COREO_*CPU_INT_FROM_CPU_1*_MAP_REG (0x0140)

Register 13.82. COREO_*CPU_INT_FROM_CPU_2*_MAP_REG (0x0144)

Register 13.83. COREO_*CPU_INT_FROM_CPU_3*_MAP_REG (0x0148)

Register 13.84. COREO_*FLASH_MSPI_INT*_MAP_REG (0x0150)

Register 13.85. COREO_*CSI_BRIDGE_INT*_MAP_REG (0x0154)

Register 13.86. COREO_*DSI_BRIDGE_INT*_MAP_REG (0x0158)

Register 13.87. COREO_*CSI_INT*_MAP_REG (0x015C)

Register 13.88. COREO_*DSI_INT*_MAP_REG (0x0160)

Register 13.89. COREO_*GMII_PHY_INT*_MAP_REG (0x0164)

Register 13.90. COREO_*LPI_INT*_MAP_REG (0x0168)

Register 13.91. COREO_*PMT_INT*_MAP_REG (0x016C)

Register 13.92. COREO_*ETH_MAC_INT*_MAP_REG (0x0170)

Register 13.93. COREO_*USB_OTG_INT*_MAP_REG (0x0174)

Register 13.94. COREO_*USB_OTG_ENDP_MULTI_PROC_INT*_MAP_REG (0x0178)

Register 13.95. COREO_*JPEG_INT*_MAP_REG (0x017C)

Register 13.96. COREO_*PPA_INT*_MAP_REG (0x0180)

Register 13.97. COREO_*COREO_TRACE_INT*_MAP_REG (0x0184)

Register 13.98. COREO_*CORE1_TRACE_INT*_MAP_REG (0x0188)

Register 13.99. COREO_*ISP_INT*_MAP_REG (0x0190)

Register 13.100. COREO_*I3C_MST_INT*_MAP_REG (0x0194)

Register 13.101. COREO_*I3C_SLV_INT*_MAP_REG (0x0198)

Register 13.102. COREO_*USB_OTG11_INT*_MAP_REG (0x019C)

Register 13.103. COREO_*DMA2D_IN_CH0_INT*_MAP_REG (0x01A0)

Register 13.104. COREO_*DMA2D_IN_CH1_INT*_MAP_REG (0x01A4)

Register 13.105. COREO_*DMA2D_OUT_CH0_INT*_MAP_REG (0x01A8)

Register 13.106. COREO_*DMA2D_OUT_CH1_INT*_MAP_REG (0x01AC)

Register 13.107. COREO_*DMA2D_OUT_CH2_INT*_MAP_REG (0x01B0)

Register 13.108. COREO_*PSRAM_MSPI_INT*_MAP_REG (0x01B4)

Register 13.109. COREO_*HP_SYSREG_INT*_MAP_REG (0x01B8)

Register 13.110. COREO_*PCNT_INT*_MAP_REG (0x01BC)

Register 13.111. COREO_*HP_PARLIO_RX_INT*_MAP_REG (0x01C4)

Register 13.112. COREO_*HP_PARLIO_TX_INT*_MAP_REG (0x01C8)

Register 13.113. COREO_*H264_DMA2D_OUT_CH0_INT*_MAP_REG (0x01CC)

Register 13.114. COREO_*H264_DMA2D_OUT_CH1_INT*_MAP_REG (0x01D0)

Register 13.115. COREO_*H264_DMA2D_OUT_CH2_INT*_MAP_REG (0x01D4)

Register 13.116. COREO_*H264_DMA2D_OUT_CH3_INT*_MAP_REG (0x01D8)

Register 13.117. COREO_*H264_DMA2D_OUT_CH4_INT*_MAP_REG (0x01DC)

Register 13.118. COREO_*H264_DMA2D_IN_CH0_INT*_MAP_REG (0x01E0)

Register 13.119. COREO_*H264_DMA2D_IN_CH1_INT*_MAP_REG (0x01E4)

Register 13.120. COREO_*H264_DMA2D_IN_CH2_INT*_MAP_REG (0x01E8)

Register 13.121. COREO_*H264_DMA2D_IN_CH3_INT*_MAP_REG (0x01EC)

Register 13.122. COREO_*H264_DMA2D_IN_CH4_INT*_MAP_REG (0x01F0)

Register 13.123. COREO_*H264_DMA2D_IN_CH5_INT*_MAP_REG (0x01F4)

Register 13.124. COREO_*H264_REG_INT*_MAP_REG (0x01F8)

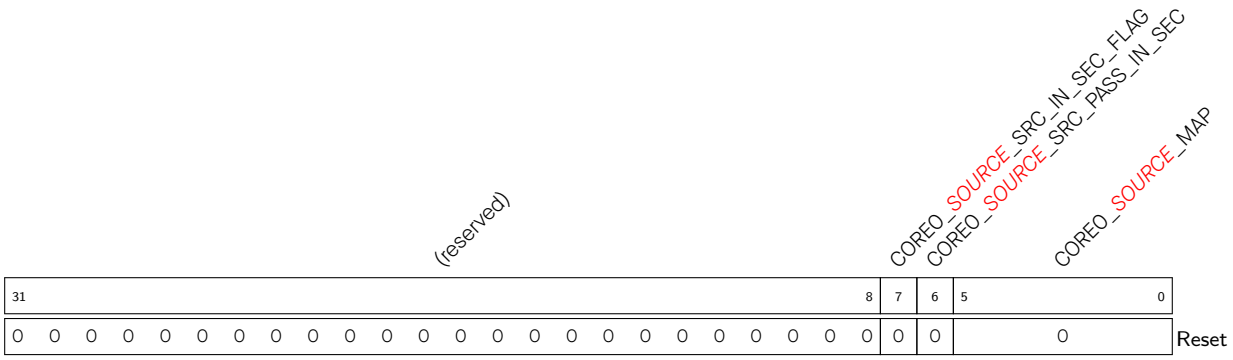
Register 13.125. COREO_*ASSIST_DEBUG_INT*_MAP_REG (0x01FC)

Register 13.126. COREO_*DMA2D_IN_CH2_INT*_MAP_REG (0x0214)

Register 13.127. COREO_*DMA2D_OUT_CH3_INT*_MAP_REG (0x0218)

Register 13.128. COREO_*AXI_PERF_MON_INT*_MAP_REG (0x021C)

Register 13.129. COREO_*SOURCE*_MAP_REG (0x0000 - 0x01FC, 0x0214 - 0x021C)



COREO_*SOURCE*_MAP 若 COREO_*SOURCE*_SRC_IN_SEC_FLAG 为 1，则仅处于机器模式下的 CPU 可写。

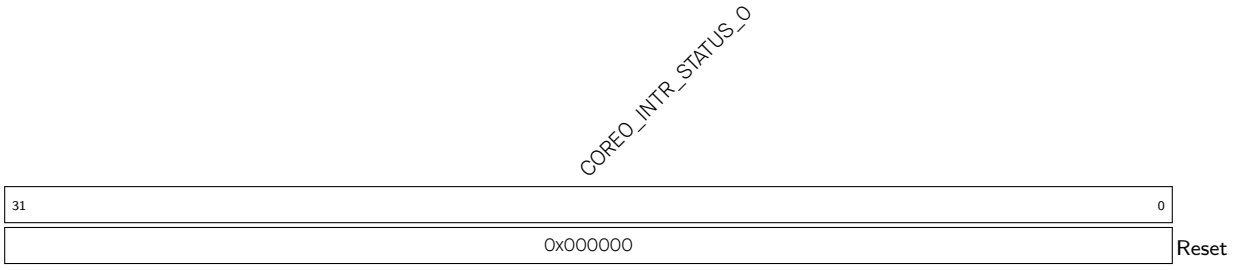
配置该寄存器将中断源 *Source* 的中断信号映射至 CPU0 外部中断，可配置范围为 16 ~ 47。当配置为 0 ~ 15 时，对应的外部中断源将被禁用（推荐设置为 0）。其它值无效。

中断源 *Source* 见表 13.4-1。(R/W)

COREO_*SOURCE*_SRC_PASS_IN_SEC 写 1 允许将中断源 *Source* 的中断信号委托至 CPU0 外部机器模式中断。仅处于机器模式下的 CPU 可写。(R/W)

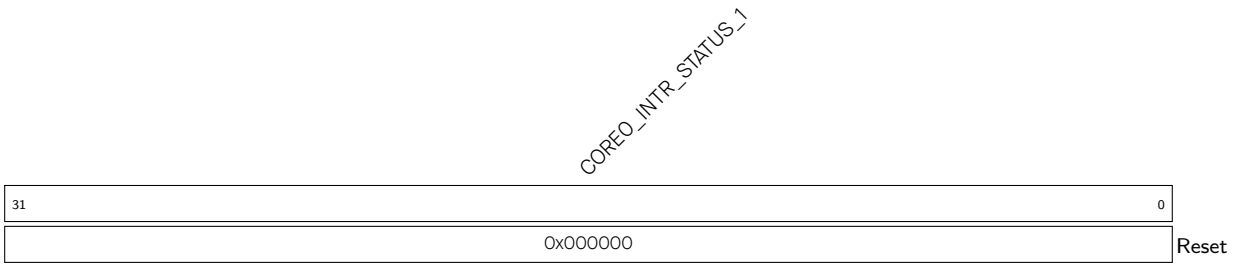
COREO_*SOURCE*_SRC_IN_SEC_FLAG 写 1 将中断源 *Source* 的中断信号配置为安全中断源 (即其中断映射关系仅机器模式下的 CPU 可配置)。仅处于机器模式下的 CPU 可写。(R/W)

Register 13.130. COREO_INTR_STATUS_REG_0_REG (0x0200)



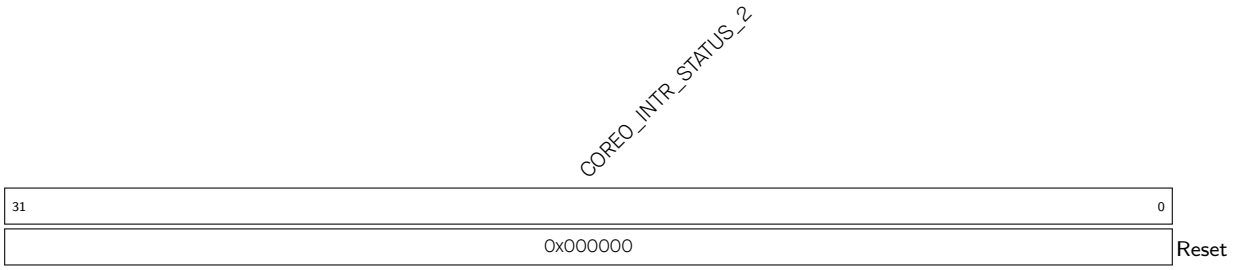
COREO_INTR_STATUS_0 中断源 0 ~ 31 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.131. COREO_INTR_STATUS_REG_1_REG (0x0204)



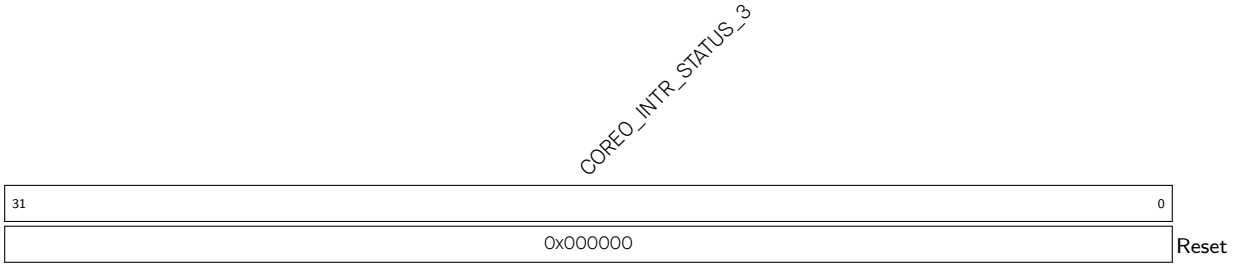
COREO_INTR_STATUS_1 中断源 32 ~ 63 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.132. COREO_INTR_STATUS_REG_2_REG (0x0208)



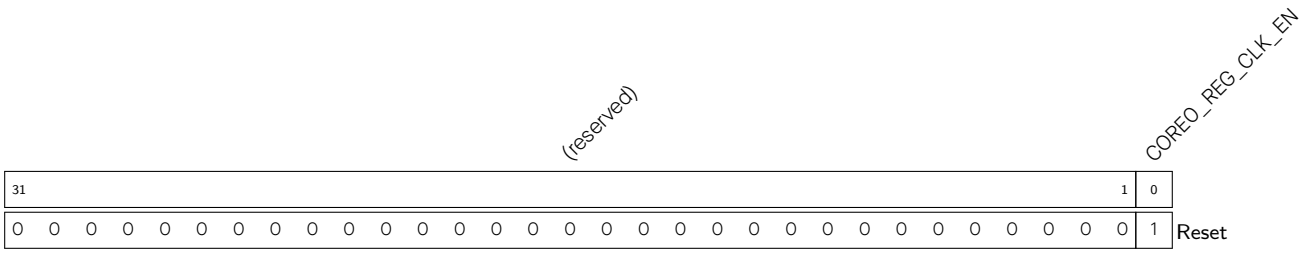
COREO_INTR_STATUS_2 中断源 64 ~ 95 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.133. COREO_INTR_STATUS_REG_3_REG (0x020C)



COREO_INTR_STATUS_3 中断源 96 ~ 127 的状态寄存器（99 和 112 保留），每一位均代表一个中断源的状态。(RO)

Register 13.134. COREO_CLOCK_GATE_REG (0x0210)



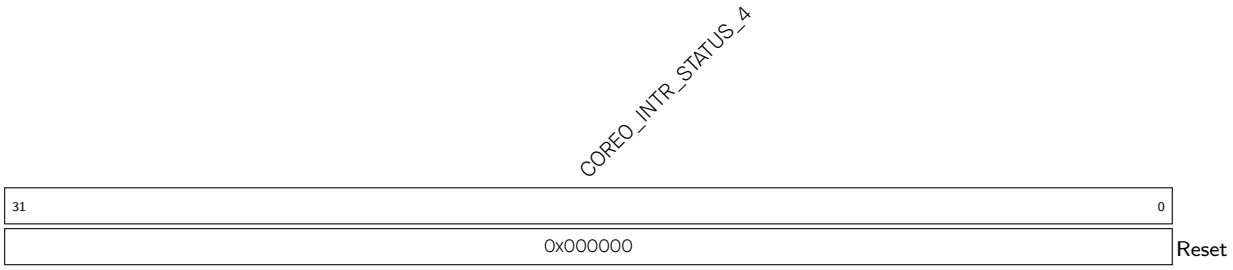
COREO_REG_CLK_EN 配置是否强制开启中断寄存器时钟门控。

0：无效

1：强制开启时钟门控

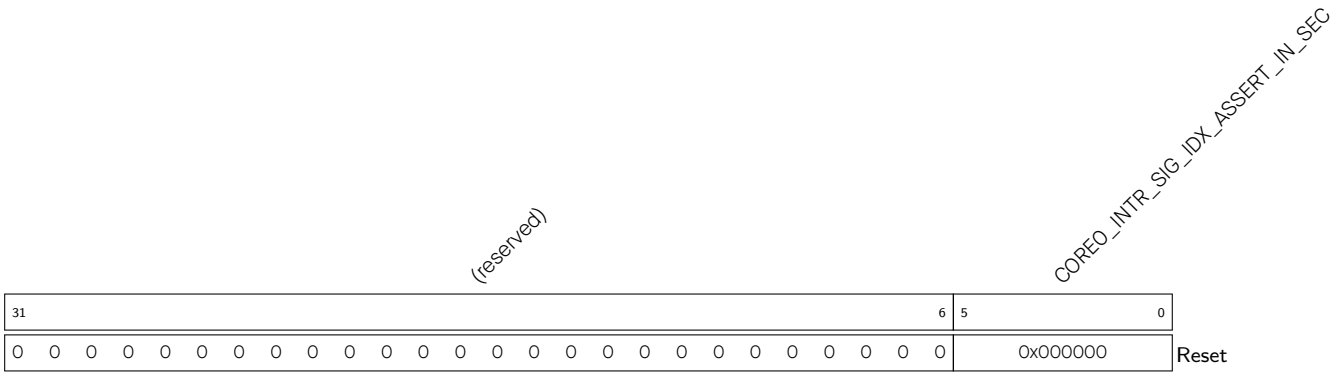
(R/W)

Register 13.135. COREO_INTR_STATUS_REG_4_REG (0x0220)



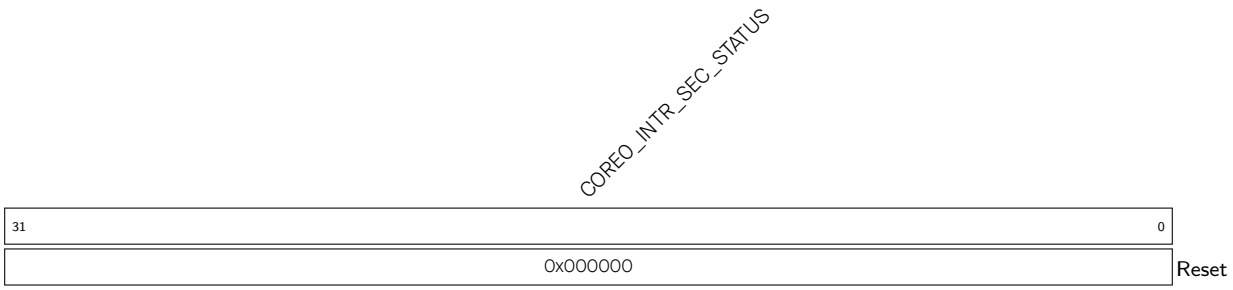
COREO_INTR_STATUS_4 中断源 128 ~ 130 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.136. COREO_INTR_SIG_IDX_ASSERT_IN_SEC_REG (0x0228)



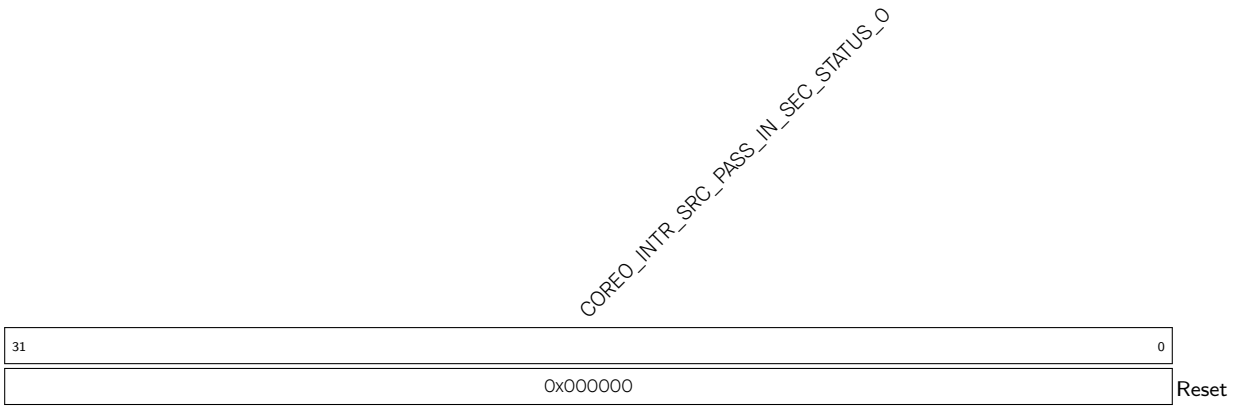
COREO_INTR_SIG_IDX_ASSERT_IN_SEC 配置 CPU0 的哪一个外部机器模式中断来作为中断委托功能的目标。有效取值范围为 16 ~ 47。(R/W)

Register 13.137. COREO_INTR_SEC_STATUS_REG (0x022C)



COREO_INTR_SEC_STATUS 表示当 COREO 处于机器模式下时，所有已被委托至机器模式中断的用户模式中断信号。(RO)

Register 13.138. COREO_INTR_SRC_PASS_IN_SEC_STATUS_0_REG (0x0230)



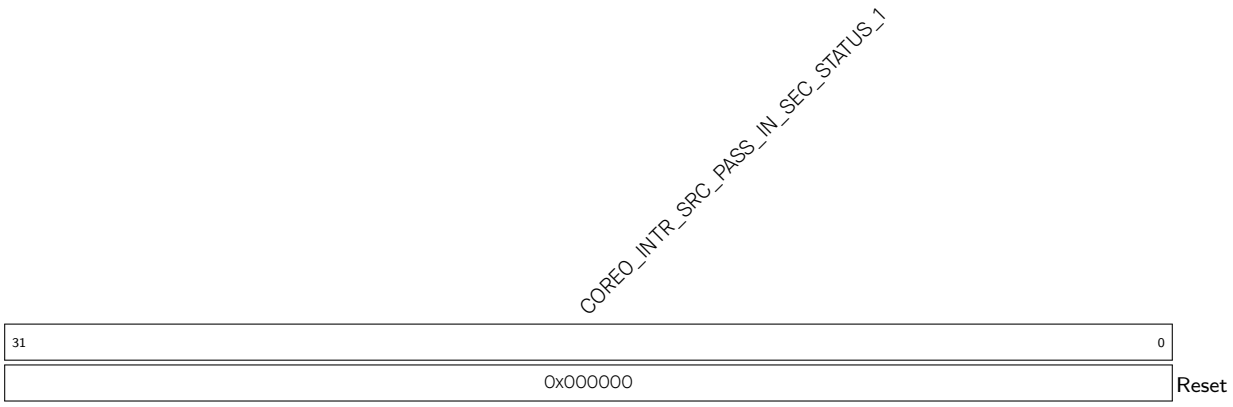
COREO_INTR_SRC_PASS_IN_SEC_STATUS_0 表示中断源 0 ~ 31 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.139. COREO_INTR_SRC_PASS_IN_SEC_STATUS_1_REG (0x0234)



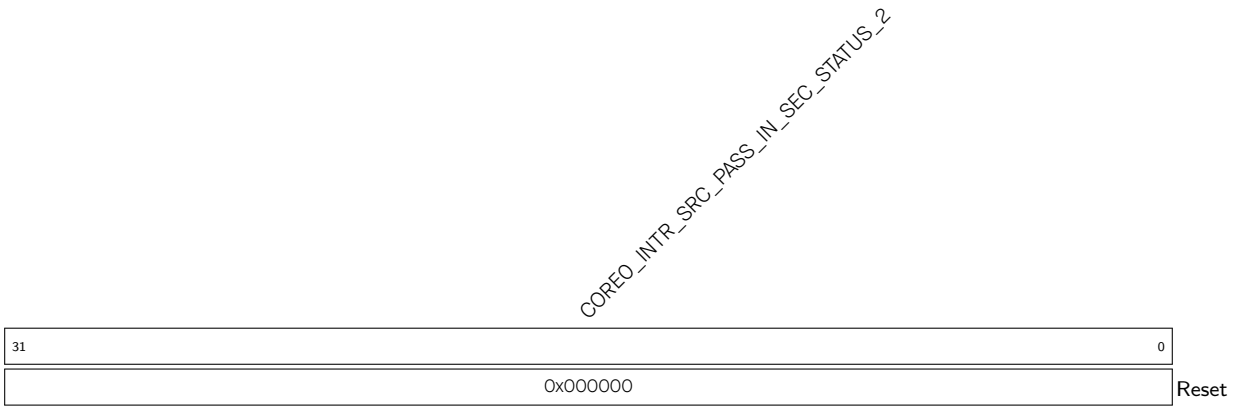
COREO_INTR_SRC_PASS_IN_SEC_STATUS_1 表示中断源 32 ~ 63 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.140. COREO_INTR_SRC_PASS_IN_SEC_STATUS_2_REG (0x0238)



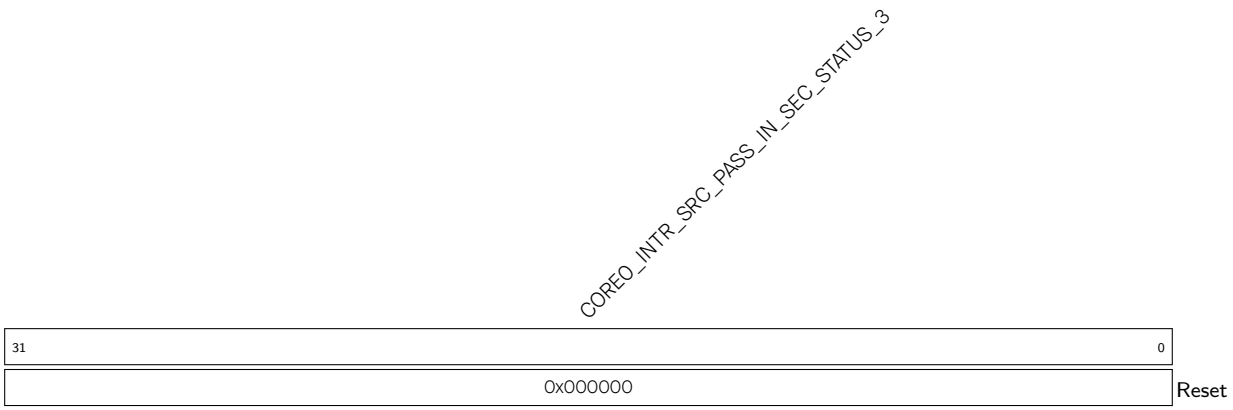
COREO_INTR_SRC_PASS_IN_SEC_STATUS_2 表示中断源 64 ~ 95 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.141. COREO_INTR_SRC_PASS_IN_SEC_STATUS_3_REG (0x023C)



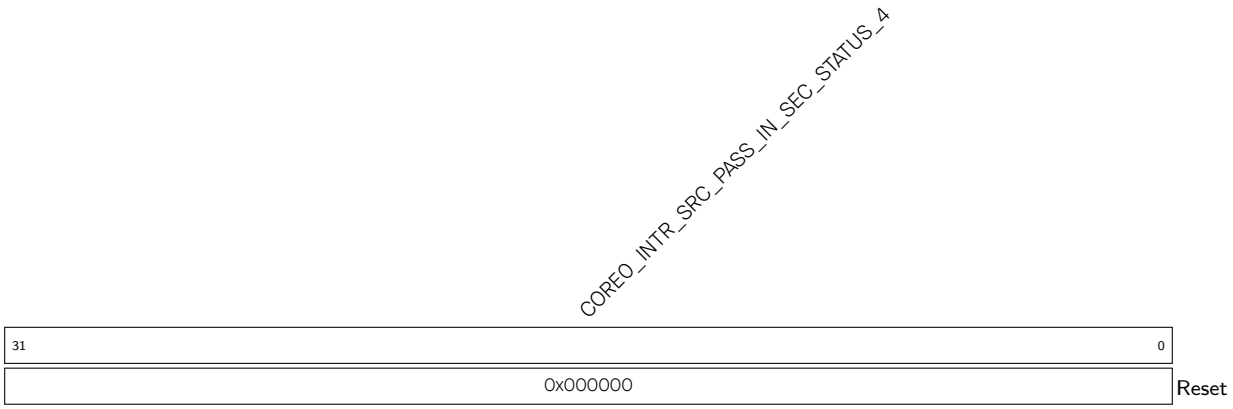
COREO_INTR_SRC_PASS_IN_SEC_STATUS_3 表示中断源 96 ~ 127 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.142. COREO_INTR_SRC_PASS_IN_SEC_STATUS_4_REG (0x0240)



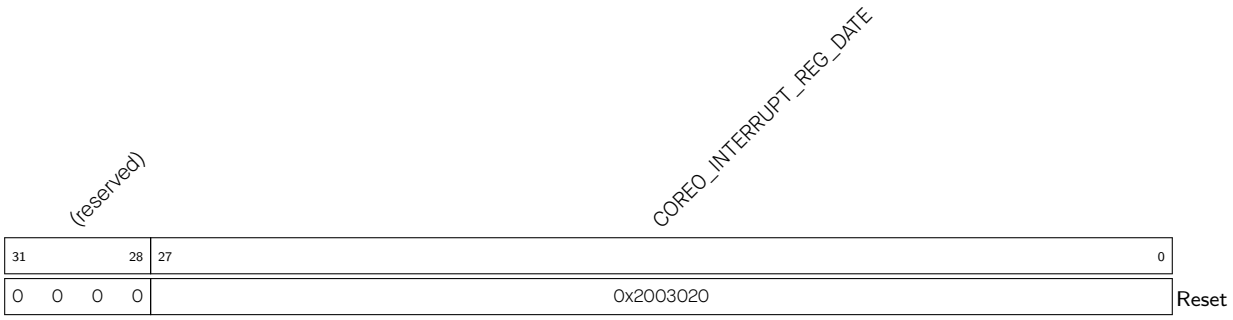
COREO_INTR_SRC_PASS_IN_SEC_STATUS_4 表示中断源 128 ~ 130 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.143. COREO_INTERRUPT_REG_DATE_REG (0x03FC)



COREO_INTERRUPT_REG_DATE 版本控制寄存器。(R/W)

13.6.2 HP CPU1 中断矩阵寄存器

Register 13.144. CORE1_*LP_RTC_INT*_MAP_REG (0x0000)

Register 13.145. CORE1_*LP_WDT_INT*_MAP_REG (0x0004)

Register 13.146. CORE1_*LP_TIMER_REG_O_INT*_MAP_REG (0x0008)

Register 13.147. CORE1_*LP_TIMER_REG_1_INT*_MAP_REG (0x000C)

Register 13.148. CORE1_*MB_HP_INT*_MAP_REG (0x0010)

Register 13.149. CORE1_*MB_LP_INT*_MAP_REG (0x0014)

Register 13.150. CORE1_*PMU_REG_O_INT*_MAP_REG (0x0018)

Register 13.151. CORE1_*PMU_REG_1_INT*_MAP_REG (0x001C)

Register 13.152. CORE1_*LP_ANAPERI_INT*_MAP_REG (0x0020)

Register 13.153. CORE1_*LP_ADC_INT*_MAP_REG (0x0024)

Register 13.154. CORE1_*LP_GPIO_INT*_MAP_REG (0x0028)

Register 13.155. CORE1_*LP_I2C_INT*_MAP_REG (0x002C)

Register 13.156. CORE1_*LP_I2S_INT*_MAP_REG (0x0030)

Register 13.157. CORE1_*LP_SPI_INT*_MAP_REG (0x0034)

Register 13.158. CORE1_*LP_TOUCH_INT*_MAP_REG (0x0038)

Register 13.159. CORE1_*LP_TSENS_INT*_MAP_REG (0x003C)

Register 13.160. CORE1_*LP_UART_INT*_MAP_REG (0x0040)

Register 13.161. CORE1_*LP_EFUSE_INT*_MAP_REG (0x0044)

Register 13.162. CORE1_*LP_SW_INT*_MAP_REG (0x0048)

Register 13.163. CORE1_*LP_SYSREG_INT*_MAP_REG (0x004C)

Register 13.164. CORE1_*LP_HUK_INT*_MAP_REG (0x0050)

Register 13.165. CORE1_*SYS_ICM_INT*_MAP_REG (0x0054)

Register 13.166. CORE1_*USB_DEVICE_INT*_MAP_REG (0x0058)

Register 13.167. CORE1_SDIO_HOST_INT_MAP_REG (0x005C)
Register 13.168. CORE1_GDMA_INT_MAP_REG (0x0060)
Register 13.169. CORE1_SPI2_INT_MAP_REG (0x0064)
Register 13.170. CORE1_SPI3_INT_MAP_REG (0x0068)
Register 13.171. CORE1_I2S0_INT_MAP_REG (0x006C)
Register 13.172. CORE1_I2S1_INT_MAP_REG (0x0070)
Register 13.173. CORE1_I2S2_INT_MAP_REG (0x0074)
Register 13.174. CORE1_UHCI0_INT_MAP_REG (0x0078)
Register 13.175. CORE1_UART0_INT_MAP_REG (0x007C)
Register 13.176. CORE1_UART1_INT_MAP_REG (0x0080)
Register 13.177. CORE1_UART2_INT_MAP_REG (0x0084)
Register 13.178. CORE1_UART3_INT_MAP_REG (0x0088)
Register 13.179. CORE1_UART4_INT_MAP_REG (0x008C)
Register 13.180. CORE1_LCD_CAM_INT_MAP_REG (0x0090)
Register 13.181. CORE1_ADC_INT_MAP_REG (0x0094)
Register 13.182. CORE1_PWM0_INT_MAP_REG (0x0098)
Register 13.183. CORE1_PWM1_INT_MAP_REG (0x009C)
Register 13.184. CORE1_TWAI0_INT_MAP_REG (0x00A0)
Register 13.185. CORE1_TWAI1_INT_MAP_REG (0x00A4)
Register 13.186. CORE1_TWAI2_INT_MAP_REG (0x00A8)
Register 13.187. CORE1_RMT_INT_MAP_REG (0x00AC)
Register 13.188. CORE1_I2C0_INT_MAP_REG (0x00B0)
Register 13.189. CORE1_I2C1_INT_MAP_REG (0x00B4)
Register 13.190. CORE1_TIMERGRP0_T0_INT_MAP_REG (0x00B8)
Register 13.191. CORE1_TIMERGRP0_T1_INT_MAP_REG (0x00BC)
Register 13.192. CORE1_TIMERGRP0_WDT_INT_MAP_REG (0x00C0)
Register 13.193. CORE1_TIMERGRP1_T0_INT_MAP_REG (0x00C4)
Register 13.194. CORE1_TIMERGRP1_T1_INT_MAP_REG (0x00C8)
Register 13.195. CORE1_TIMERGRP1_WDT_INT_MAP_REG (0x00CC)
Register 13.196. CORE1_LEDC_INT_MAP_REG (0x00D0)
Register 13.197. CORE1_SYSTIMER_TARGET0_INT_MAP_REG (0x00D4)
Register 13.198. CORE1_SYSTIMER_TARGET1_INT_MAP_REG (0x00D8)
Register 13.199. CORE1_SYSTIMER_TARGET2_INT_MAP_REG (0x00DC)
Register 13.200. CORE1_AHB_PDMA_IN_CH0_INT_MAP_REG (0x00E0)
Register 13.201. CORE1_AHB_PDMA_IN_CH1_INT_MAP_REG (0x00E4)

Register 13.202. CORE1_*AHB_PDMA_IN_CH2_INT*_MAP_REG (0x00E8)

Register 13.203. CORE1_*AHB_PDMA_OUT_CH0_INT*_MAP_REG (0x00EC)

Register 13.204. CORE1_*AHB_PDMA_OUT_CH1_INT*_MAP_REG (0x00F0)

Register 13.205. CORE1_*AHB_PDMA_OUT_CH2_INT*_MAP_REG (0x00F4)

Register 13.206. CORE1_*AXI_PDMA_IN_CH0_INT*_MAP_REG (0x00F8)

Register 13.207. CORE1_*AXI_PDMA_IN_CH1_INT*_MAP_REG (0x00FC)

Register 13.208. CORE1_*AXI_PDMA_IN_CH2_INT*_MAP_REG (0x0100)

Register 13.209. CORE1_*AXI_PDMA_OUT_CH0_INT*_MAP_REG (0x0104)

Register 13.210. CORE1_*AXI_PDMA_OUT_CH1_INT*_MAP_REG (0x0108)

Register 13.211. CORE1_*AXI_PDMA_OUT_CH2_INT*_MAP_REG (0x010C)

Register 13.212. CORE1_*RSA_INT*_MAP_REG (0x0110)

Register 13.213. CORE1_*AES_INT*_MAP_REG (0x0114)

Register 13.214. CORE1_*SHA_INT*_MAP_REG (0x0118)

Register 13.215. CORE1_*ECC_INT*_MAP_REG (0x011C)

Register 13.216. CORE1_*ECDSA_INT*_MAP_REG (0x0120)

Register 13.217. CORE1_*KM_INT*_MAP_REG (0x0124)

Register 13.218. CORE1_*GPIO_INT0*_MAP_REG (0x0128)

Register 13.219. CORE1_*GPIO_INT1*_MAP_REG (0x012C)

Register 13.220. CORE1_*GPIO_INT2*_MAP_REG (0x0130)

Register 13.221. CORE1_*GPIO_INT3*_MAP_REG (0x0134)

Register 13.222. CORE1_*GPIO_PAD_COMP_INT*_MAP_REG (0x0138)

Register 13.223. CORE1_*CPU_INT_FROM_CPU_0*_MAP_REG (0x013C)

Register 13.224. CORE1_*CPU_INT_FROM_CPU_1*_MAP_REG (0x0140)

Register 13.225. CORE1_*CPU_INT_FROM_CPU_2*_MAP_REG (0x0144)

Register 13.226. CORE1_*CPU_INT_FROM_CPU_3*_MAP_REG (0x0148)

Register 13.227. CORE1_*FLASH_MSPI_INT*_MAP_REG (0x0150)

Register 13.228. CORE1_*CSI_BRIDGE_INT*_MAP_REG (0x0154)

Register 13.229. CORE1_*DSI_BRIDGE_INT*_MAP_REG (0x0158)

Register 13.230. CORE1_*CSI_INT*_MAP_REG (0x015C)

Register 13.231. CORE1_*DSI_INT*_MAP_REG (0x0160)

Register 13.232. CORE1_*GMII_PHY_INT*_MAP_REG (0x0164)

Register 13.233. CORE1_*LPI_INT*_MAP_REG (0x0168)

Register 13.234. CORE1_*PMT_INT*_MAP_REG (0x016C)

Register 13.235. CORE1_*ETH_MAC_INT*_MAP_REG (0x0170)

Register 13.236. CORE1_*USB_OTG_INT*_MAP_REG (0x0174)

Register 13.237. CORE1_*USB_OTG_ENDP_MULTI_PROC_INT*_MAP_REG (0x0178)

Register 13.238. CORE1_*JPEG_INT*_MAP_REG (0x017C)

Register 13.239. CORE1_*PPA_INT*_MAP_REG (0x0180)

Register 13.240. CORE1_*CORE0_TRACE_INT*_MAP_REG (0x0184)

Register 13.241. CORE1_*CORE1_TRACE_INT*_MAP_REG (0x0188)

Register 13.242. CORE1_*ISP_INT*_MAP_REG (0x0190)

Register 13.243. CORE1_*I3C_MST_INT*_MAP_REG (0x0194)

Register 13.244. CORE1_*I3C_SLV_INT*_MAP_REG (0x0198)

Register 13.245. CORE1_*USB_OTG11_INT*_MAP_REG (0x019C)

Register 13.246. CORE1_*DMA2D_IN_CH0_INT*_MAP_REG (0x01A0)

Register 13.247. CORE1_*DMA2D_IN_CH1_INT*_MAP_REG (0x01A4)

Register 13.248. CORE1_*DMA2D_OUT_CH0_INT*_MAP_REG (0x01A8)

Register 13.249. CORE1_*DMA2D_OUT_CH1_INT*_MAP_REG (0x01AC)

Register 13.250. CORE1_*DMA2D_OUT_CH2_INT*_MAP_REG (0x01B0)

Register 13.251. CORE1_*PSRAM_MSPI_INT*_MAP_REG (0x01B4)

Register 13.252. CORE1_*HP_SYSREG_INT*_MAP_REG (0x01B8)

Register 13.253. CORE1_*PCNT_INT*_MAP_REG (0x01BC)

Register 13.254. CORE1_*HP_PAU_INT*_MAP_REG (0x01C0)

Register 13.255. CORE1_*HP_PARLIO_RX_INT*_MAP_REG (0x01C4)

Register 13.256. CORE1_*HP_PARLIO_TX_INT*_MAP_REG (0x01C8)

Register 13.257. CORE1_*H264_DMA2D_OUT_CH0_INT*_MAP_REG (0x01CC)

Register 13.258. CORE1_*H264_DMA2D_OUT_CH1_INT*_MAP_REG (0x01D0)

Register 13.259. CORE1_*H264_DMA2D_OUT_CH2_INT*_MAP_REG (0x01D4)

Register 13.260. CORE1_*H264_DMA2D_OUT_CH3_INT*_MAP_REG (0x01D8)

Register 13.261. CORE1_*H264_DMA2D_OUT_CH4_INT*_MAP_REG (0x01DC)

Register 13.262. CORE1_*H264_DMA2D_IN_CH0_INT*_MAP_REG (0x01E0)

Register 13.263. CORE1_*H264_DMA2D_IN_CH1_INT*_MAP_REG (0x01E4)

Register 13.264. CORE1_*H264_DMA2D_IN_CH2_INT*_MAP_REG (0x01E8)

Register 13.265. CORE1_*H264_DMA2D_IN_CH3_INT*_MAP_REG (0x01EC)

Register 13.266. CORE1_*H264_DMA2D_IN_CH4_INT*_MAP_REG (0x01F0)

Register 13.267. CORE1_*H264_DMA2D_IN_CH5_INT*_MAP_REG (0x01F4)

Register 13.268. CORE1_*H264_REG_INT*_MAP_REG (0x01F8)

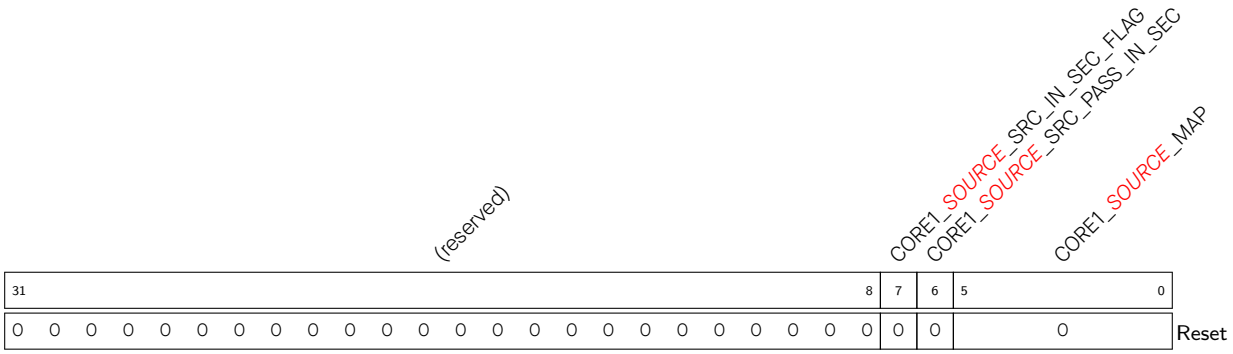
Register 13.269. CORE1_*ASSIST_DEBUG_INT*_MAP_REG (0x01FC)

Register 13.270. CORE1_*DMA2D_IN_CH2_INT*_MAP_REG (0x0214)

Register 13.271. CORE1_*DMA2D_OUT_CH3_INT*_MAP_REG (0x0218)

Register 13.272. CORE1_*AXI_PERF_MON_INT*_MAP_REG (0x021C)

Register 13.273. CORE1_*SOURCE*_MAP_REG (0x0000 - 0x01FC, 0x0214 - 0x021C)



CORE1_*SOURCE*_MAP 若 CORE1_*SOURCE*_SRC_IN_SEC_FLAG 为 1，则仅处于机器模式下的 CPU 可写。

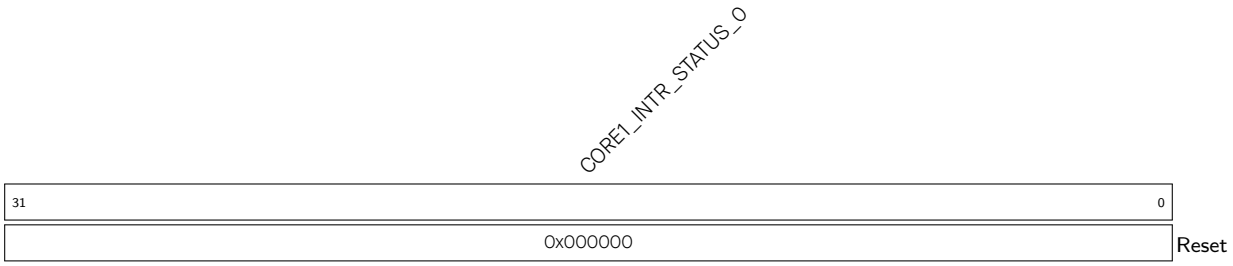
配置该寄存器将中断源 *Source* 的中断信号映射至 CPU1 外部中断，可配置范围为 16 ~ 47。当配置为 0 ~ 15 时，对应的外部中断源将被禁用（推荐设置为 0）。其它值无效。

中断源 *Source* 见表 13.4-1。(R/W)

CORE1_*SOURCE*_SRC_PASS_IN_SEC 写 1 允许将中断源 *Source* 的中断信号委托至 CPU1 外部机器模式中断。仅处于机器模式下的 CPU 可写。(R/W)

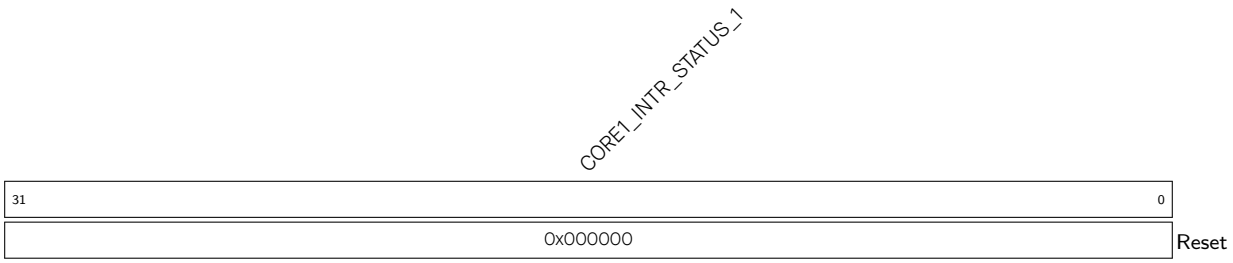
CORE1_*SOURCE*_SRC_IN_SEC_FLAG 写 1 将中断源 *Source* 的中断信号配置为安全中断源 (即其中断映射关系仅机器模式下的 CPU 可配置)。仅处于机器模式下的 CPU 可写。(R/W)

Register 13.274. CORE1_INTR_STATUS_REG_0_REG (0x0200)



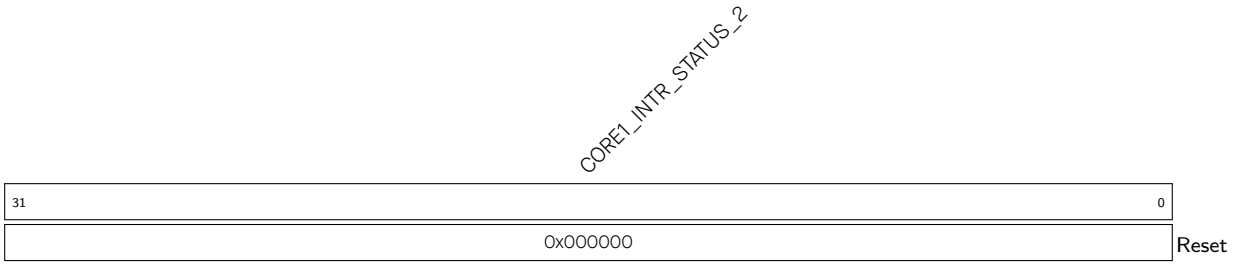
CORE1_INTR_STATUS_0 中断源 0 ~ 31 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.275. CORE1_INTR_STATUS_REG_1_REG (0x0204)



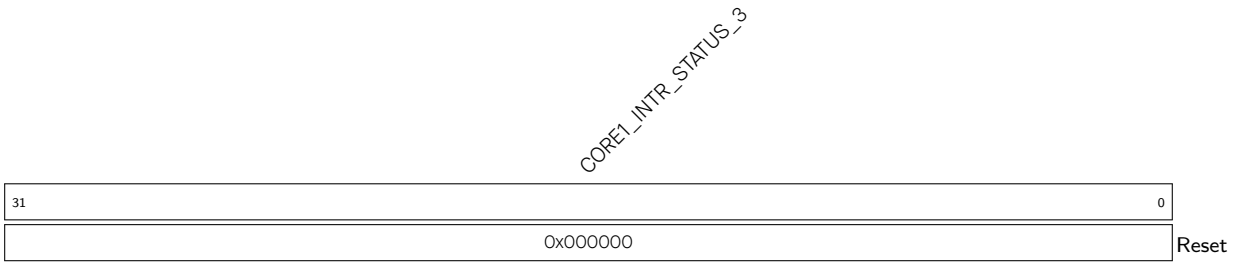
CORE1_INTR_STATUS_1 中断源 32 ~ 63 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.276. CORE1_INTR_STATUS_REG_2_REG (0x0208)



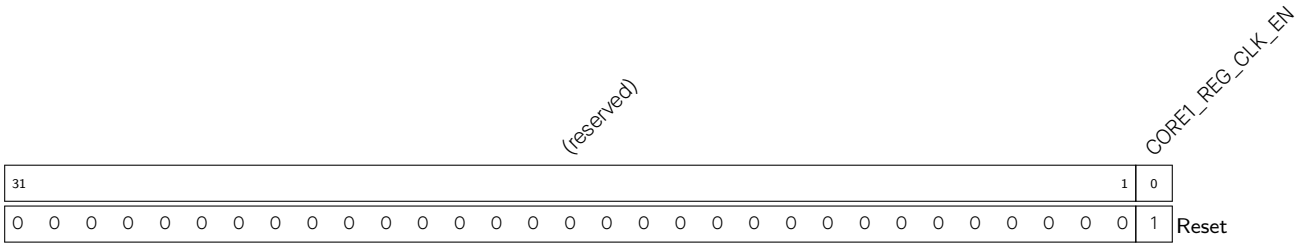
CORE1_INTR_STATUS_2 中断源 64 ~ 95 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.277. CORE1_INTR_STATUS_REG_3_REG (0x020C)



CORE1_INTR_STATUS_3 中断源 96 ~ 127 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.278. CORE1_CLOCK_GATE_REG (0x0210)



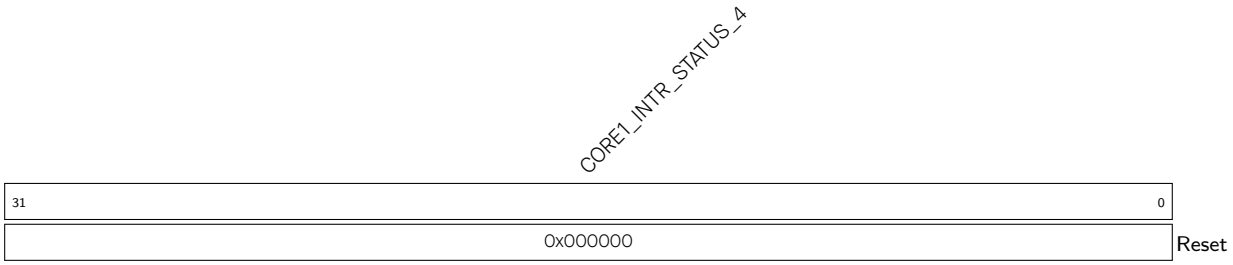
CORE1_REG_CLK_EN 配置是否强制开启中断寄存器时钟门控。

0：无效

1：强制开启时钟门控

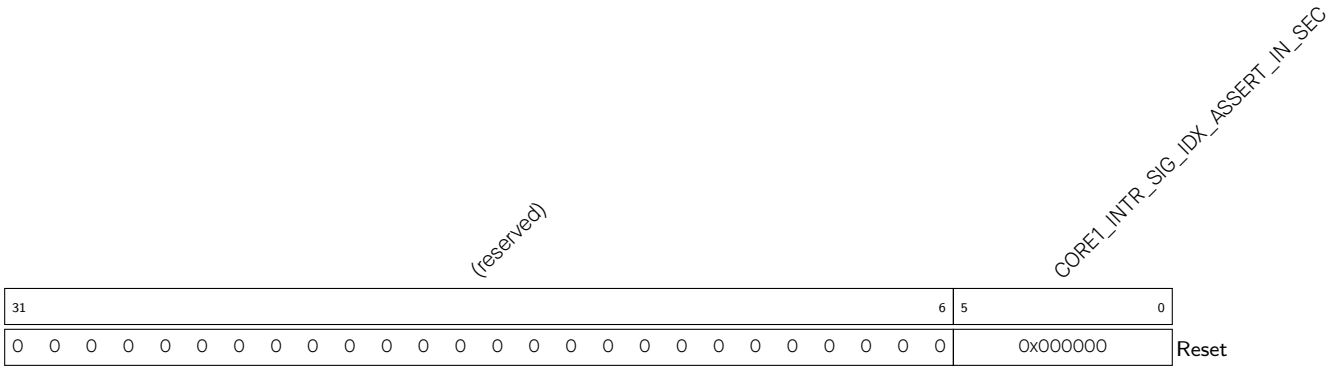
(R/W)

Register 13.279. CORE1_INTR_STATUS_REG_4_REG (0x0220)



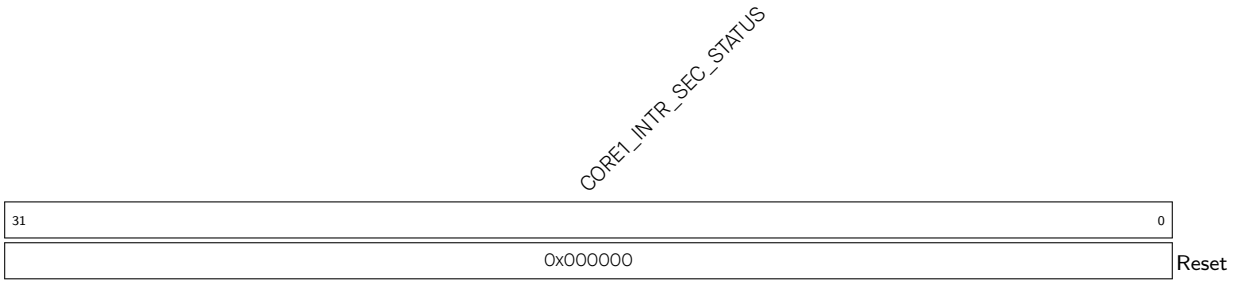
CORE1_INTR_STATUS_4 中断源 128 ~ 130 的状态寄存器，每一位均代表一个中断源的状态。(RO)

Register 13.280. CORE1_INTR_SIG_IDX_ASSERT_IN_SEC_REG (0x0228)



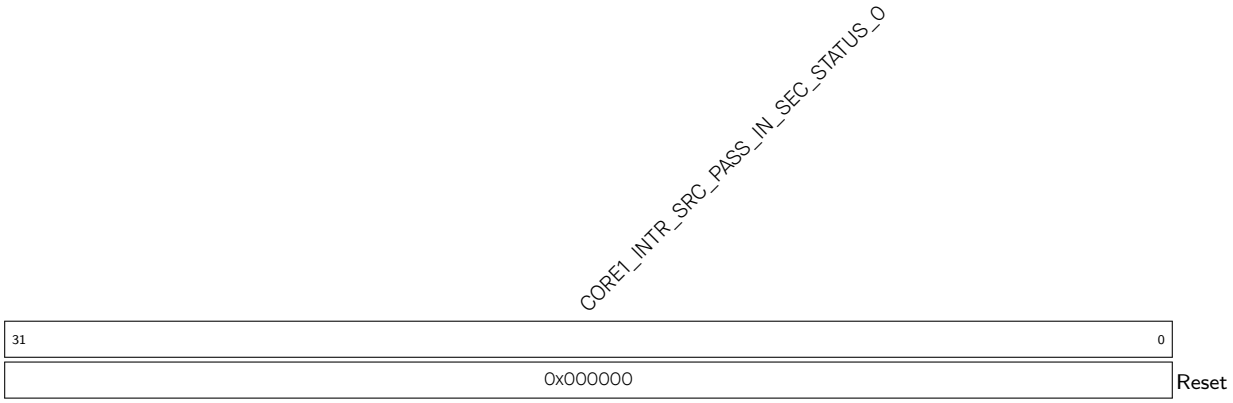
CORE1_INTR_SIG_IDX_ASSERT_IN_SEC 配置 CPU1 的哪一个外部机器模式中断来作为中断委托功能的目标。有效取值范围为 16 ~ 47。(R/W)

Register 13.281. CORE1_INTR_SEC_STATUS_REG (0x022C)



CORE1_INTR_SEC_STATUS 表示当 CORE1 处于机器模式下时，所有已被委托至机器模式中断的用户模式中断信号。(RO)

Register 13.282. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_0_REG (0x0230)



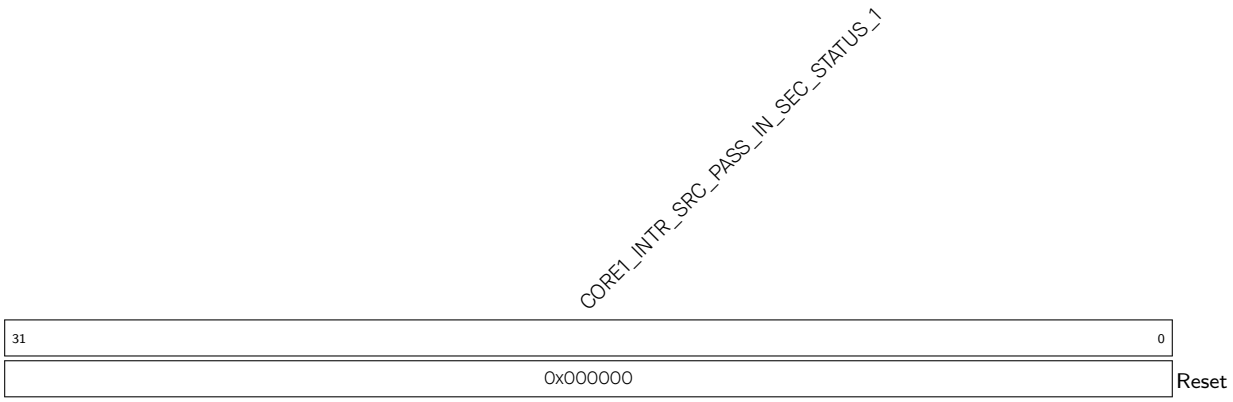
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_0 表示中断源 0 ~ 31 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.283. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_1_REG (0x0234)



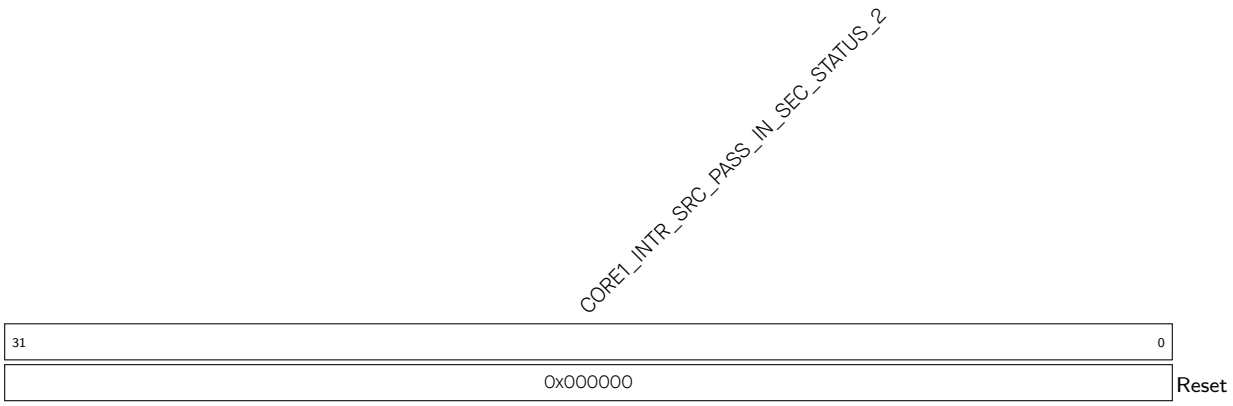
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_1 表示中断源 32 ~ 63 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.284. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_2_REG (0x0238)



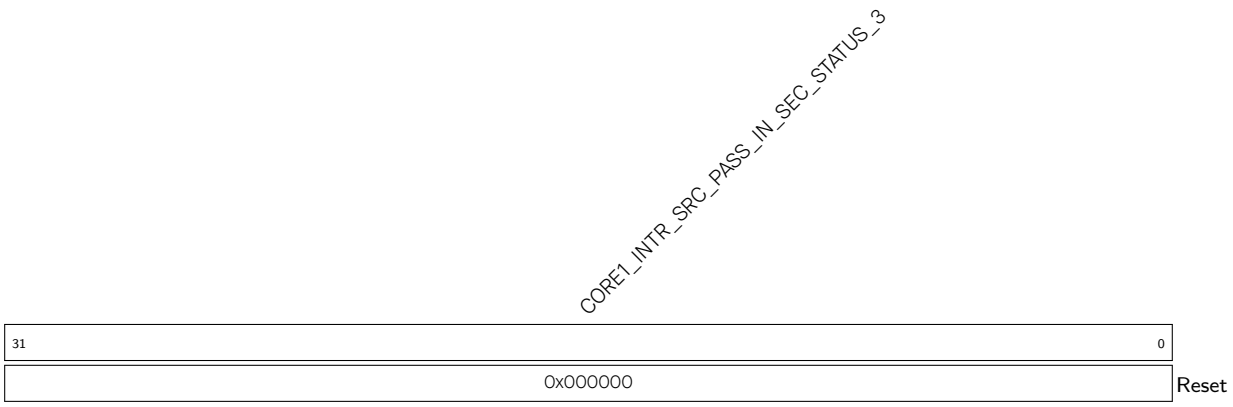
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_2 表示中断源 64 ~ 95 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.285. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_3_REG (0x023C)



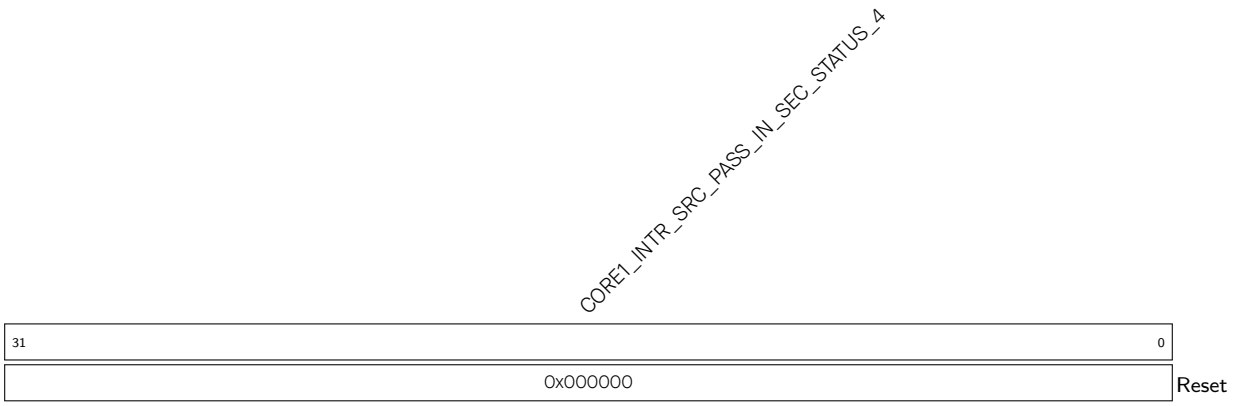
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_3 表示中断源 96 ~ 127 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.286. CORE1_INTR_SRC_PASS_IN_SEC_STATUS_4_REG (0x0240)



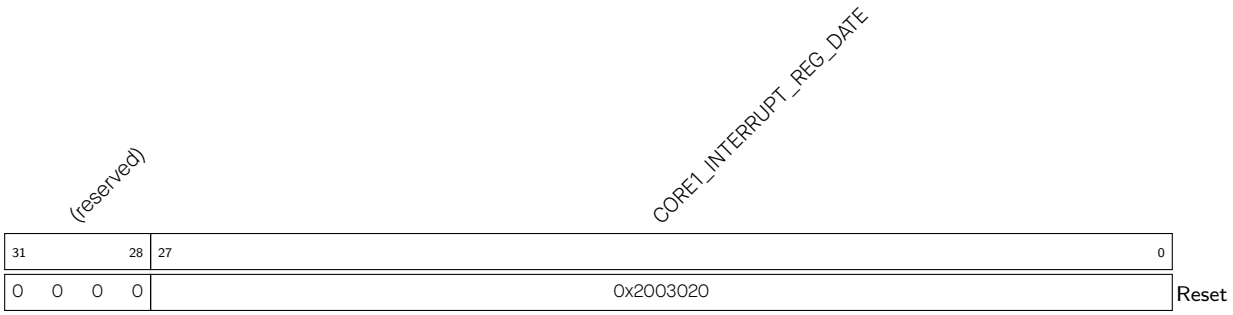
CORE1_INTR_SRC_PASS_IN_SEC_STATUS_4 表示中断源 128 ~ 130 是否使能了中断委托功能，每一位均代表一个中断源的使能状态。

0: 对应中断源未使能中断委托功能

1: 对应中断源使能了中断委托功能

(RO)

Register 13.287. CORE1_INTERRUPT_REG_DATE_REG (0x03FC)



CORE1_INTERRUPT_REG_DATE 版本控制寄存器。(R/W)

第 14 章

事件任务矩阵 (ETM)

14.1 概述

事件任务矩阵 (ETM) 外设包含 50 个可配置通道。每个通道可以将任意指定外设的事件映射到任意指定外设的任务，从而触发外设执行指定任务，无需 CPU 干预。

14.2 特性

事件任务矩阵具有以下特性：

- 支持从多个外设接收多种事件
- 支持为多个外设生成多种任务
- 拥有 50 个可独立配置的 ETM 通道
- 每个通道接收到的事件可以是所有事件中的任意一个，每个通道接收到的事件可以映射到任意的任务上输出
- 每个 ETM 通道都可以独立使能。当通道未使能时，它不会响应所配置的事件，也不会生成要映射到的任务
- 支持查看每个事件和任务的触发状态
- 能够产生事件、接收任务的外设有：GPIO、LED PWM、通用定时器、RTC 定时器、系统定时器、MCPWM、温度传感器、ADC、I2S、LP CPU、GDMA-AHB、GDMA-AXI、2D DMA 和 PMU

由于 50 个 ETM 通道的功能和操作都是相同的。以下部分将 ETM 通道统称为通道*n*（其中 *n* 范围从 0 到 49）。

14.3 功能描述

14.3.1 架构

事件任务矩阵有 50 个独立通道，每个通道可以选择所有事件中的任意一个作为该通道的输入（参考 14.3.2 章节），并且可以选择映射事件到所有任务中的任何一个（参考 14.3.3 章节）。每个通道都有独立的使能配置位（参考 14.3.6 章节）。

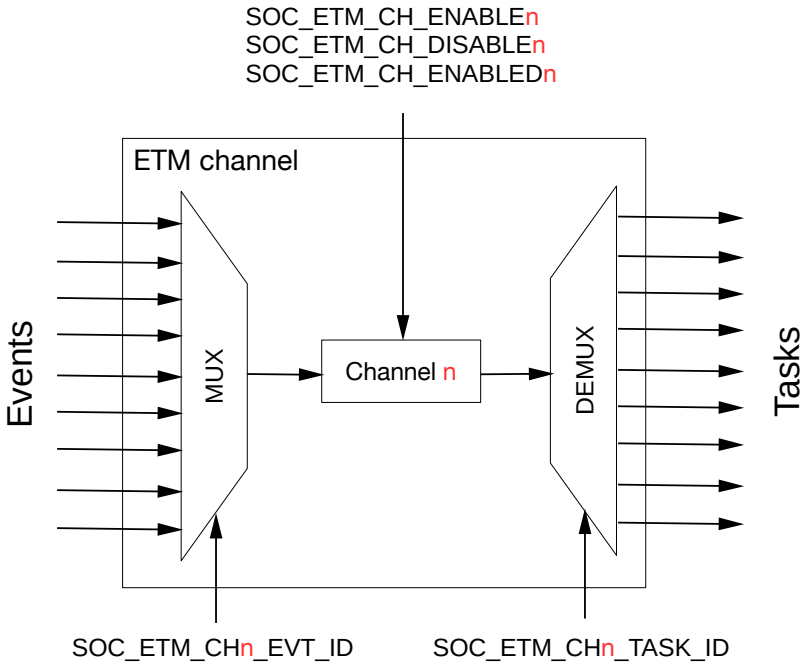


图 14.3-1. 事件任务矩阵通道*n* 架构

图 14.3-1 展示了单个事件任务矩阵通道的结构，其中 SOC_ETM_CH*n*_EVT_ID 字段配置 MUX 选择任意一个事件作为通道*n* 的输入，SOC_ETM_CH*n*_TASK_ID 字段配置 DEMUX 以映射通道*n* 所选择的事件到任意一个任务。字段 SOC_ETM_CH_ENABLE*n* 和 SOC_ETM_CH_DISABLE*n* 用于使能或关闭通道*n*，字段 SOC_ETM_CH_ENABLED*n* 用于指示通道*n* 的开启或关闭状态。

14.3.2 事件

每个事件任务矩阵通道可以选择所有事件之一作为其接收的事件。每个通道都有字段 SOC_ETM_CH*n*_EVT_ID，用于指定选择哪个事件。字段 SOC_ETM_CH*n*_EVT_ID 的配置值与所选择的事件的关系如表 14.3-1 所示。

表 14.3-1. 事件任务矩阵通道*n* 可选事件

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
1	GPIO_EVT_CH0_RISE_EDGE	GPIO
2	GPIO_EVT_CH1_RISE_EDGE	
3	GPIO_EVT_CH2_RISE_EDGE	
4	GPIO_EVT_CH3_RISE_EDGE	
5	GPIO_EVT_CH4_RISE_EDGE	
6	GPIO_EVT_CH5_RISE_EDGE	
7	GPIO_EVT_CH6_RISE_EDGE	
8	GPIO_EVT_CH7_RISE_EDGE	
9	GPIO_EVT_CH0_FALL_EDGE	
10	GPIO_EVT_CH1_FALL_EDGE	
11	GPIO_EVT_CH2_FALL_EDGE	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
12	GPIO_EVT_CH3_FALL_EDGE	
13	GPIO_EVT_CH4_FALL_EDGE	
14	GPIO_EVT_CH5_FALL_EDGE	
15	GPIO_EVT_CH6_FALL_EDGE	
16	GPIO_EVT_CH7_FALL_EDGE	
17	GPIO_EVT_CH0_ANY_EDGE	
18	GPIO_EVT_CH1_ANY_EDGE	
19	GPIO_EVT_CH2_ANY_EDGE	
20	GPIO_EVT_CH3_ANY_EDGE	
21	GPIO_EVT_CH4_ANY_EDGE	
22	GPIO_EVT_CH5_ANY_EDGE	
23	GPIO_EVT_CH6_ANY_EDGE	
24	GPIO_EVT_CH7_ANY_EDGE	
25	GPIO_EVT_ZERO_DET_POS0	
26	GPIO_EVT_ZERO_DET_NEG0	
27	GPIO_EVT_ZERO_DET_POS1	
28	GPIO_EVT_ZERO_DET_NEG1	
29	LEDC_EVT_DUTY_CHNG_END_CH0	LED PWM 控制器 (LEDC)
30	LEDC_EVT_DUTY_CHNG_END_CH1	
31	LEDC_EVT_DUTY_CHNG_END_CH2	
32	LEDC_EVT_DUTY_CHNG_END_CH3	
33	LEDC_EVT_DUTY_CHNG_END_CH4	
34	LEDC_EVT_DUTY_CHNG_END_CH5	
35	LEDC_EVT_DUTY_CHNG_END_CH6	
36	LEDC_EVT_DUTY_CHNG_END_CH7	
37	LEDC_EVT_OVF_CNT_PLS_CH0	
38	LEDC_EVT_OVF_CNT_PLS_CH1	
39	LEDC_EVT_OVF_CNT_PLS_CH2	
40	LEDC_EVT_OVF_CNT_PLS_CH3	
41	LEDC_EVT_OVF_CNT_PLS_CH4	
42	LEDC_EVT_OVF_CNT_PLS_CH5	
43	LEDC_EVT_OVF_CNT_PLS_CH6	
44	LEDC_EVT_OVF_CNT_PLS_CH7	
45	LEDC_EVT_TIME_OVF_TIMER0	
46	LEDC_EVT_TIME_OVF_TIMER1	
47	LEDC_EVT_TIME_OVF_TIMER2	
48	LEDC_EVT_TIME_OVF_TIMER3	
49	LEDC_EVT_TIMER0_CMP	
50	LEDC_EVT_TIMER1_CMP	
51	LEDC_EVT_TIMER2_CMP	
52	LEDC_EVT_TIMER3_CMP	
53	TG0_EVT_CNT_CMP_TIMER0	通用定时器 0
55	TG1_EVT_CNT_CMP_TIMER0	通用定时器 1
57	SYSTIMER_EVT_CNT_CMP0	系统定时器
58	SYSTIMER_EVT_CNT_CMP1	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
59	SYSTIMER_EVT_CNT_CMP2	
60	MCPWM0_EVT_TIMER0_STOP	MCPWM0
61	MCPWM0_EVT_TIMER1_STOP	
62	MCPWM0_EVT_TIMER2_STOP	
63	MCPWM0_EVT_TIMER0_TEZ	
64	MCPWM0_EVT_TIMER1_TEZ	
65	MCPWM0_EVT_TIMER2_TEZ	
66	MCPWM0_EVT_TIMER0_TEP	
67	MCPWM0_EVT_TIMER1_TEP	
68	MCPWM0_EVT_TIMER2_TEP	
69	MCPWM0_EVT_OP0_TEA	
70	MCPWM0_EVT_OP1_TEA	
71	MCPWM0_EVT_OP2_TEA	
72	MCPWM0_EVT_OP0_TEB	
73	MCPWM0_EVT_OP1_TEB	
74	MCPWM0_EVT_OP2_TEB	
75	MCPWM0_EVT_F0	
76	MCPWM0_EVT_F1	
77	MCPWM0_EVT_F2	
78	MCPWM0_EVT_F0_CLR	
79	MCPWM0_EVT_F1_CLR	
80	MCPWM0_EVT_F2_CLR	
81	MCPWM0_EVT_TZ0_CBC	
82	MCPWM0_EVT_TZ1_CBC	
83	MCPWM0_EVT_TZ2_CBC	
84	MCPWM0_EVT_TZ0_OST	
85	MCPWM0_EVT_TZ1_OST	
86	MCPWM0_EVT_TZ2_OST	
87	MCPWM0_EVT_CAP0	
88	MCPWM0_EVT_CAP1	
89	MCPWM0_EVT_CAP2	
90	MCPWM0_EVT_OP0_TEE1	
91	MCPWM0_EVT_OP1_TEE1	
92	MCPWM0_EVT_OP2_TEE1	
93	MCPWM0_EVT_OP0_TEE2	
94	MCPWM0_EVT_OP1_TEE2	
95	MCPWM0_EVT_OP2_TEE2	
96	MCPWM1_EVT_TIMER0_STOP	MCPWM1
97	MCPWM1_EVT_TIMER1_STOP	
98	MCPWM1_EVT_TIMER2_STOP	
99	MCPWM1_EVT_TIMER0_TEZ	
100	MCPWM1_EVT_TIMER1_TEZ	
101	MCPWM1_EVT_TIMER2_TEZ	
102	MCPWM1_EVT_TIMER0_TEP	
103	MCPWM1_EVT_TIMER1_TEP	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
104	MCPWM1_EVT_TIMER2_TEP	
105	MCPWM1_EVT_OP0_TEA	
106	MCPWM1_EVT_OP1_TEA	
107	MCPWM1_EVT_OP2_TEA	
108	MCPWM1_EVT_OP0_TEB	
109	MCPWM1_EVT_OP1_TEB	
110	MCPWM1_EVT_OP2_TEB	
111	MCPWM1_EVT_F0	
112	MCPWM1_EVT_F1	
113	MCPWM1_EVT_F2	
114	MCPWM1_EVT_F0_CLR	
115	MCPWM1_EVT_F1_CLR	
116	MCPWM1_EVT_F2_CLR	
117	MCPWM1_EVT_TZ0_CBC	
118	MCPWM1_EVT_TZ1_CBC	
119	MCPWM1_EVT_TZ2_CBC	
120	MCPWM1_EVT_TZ0_OST	
121	MCPWM1_EVT_TZ1_OST	
122	MCPWM1_EVT_TZ2_OST	
123	MCPWM1_EVT_CAP0	
124	MCPWM1_EVT_CAP1	
125	MCPWM1_EVT_CAP2	
126	MCPWM1_EVT_OP0_TEE1	
127	MCPWM1_EVT_OP1_TEE1	
128	MCPWM1_EVT_OP2_TEE1	
129	MCPWM1_EVT_OP0_TEE2	
130	MCPWM1_EVT_OP1_TEE2	
131	MCPWM1_EVT_OP2_TEE2	
132	ADC_EVT_CONV_CMPLT0	ADC 控制器 (ADC)
133	ADC_EVT_EQ_ABOVE_THRESH0	
134	ADC_EVT_EQ_ABOVE_THRESH1	
135	ADC_EVT_EQ_BELOW_THRESH0	
136	ADC_EVT_EQ_BELOW_THRESH1	
138	ADC_EVT_STOPPED0	
139	ADC_EVT_STARTED0	
148	TMPSNSR_EVT_OVER_LIMIT	温度传感器 (TSENS)
149	I2S0_EVT_RX_DONE	I2S0
150	I2S0_EVT_TX_DONE	
151	I2S0_EVT_X_WORDS_RECEIVED	
152	I2S0_EVT_X_WORDS_SENT	
153	I2S1_EVT_RX_DONE	I2S1
154	I2S1_EVT_TX_DONE	
155	I2S1_EVT_X_WORDS_RECEIVED	
156	I2S1_EVT_X_WORDS_SENT	
157	I2S2_EVT_RX_DONE	I2S2

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
158	I2S2_EVT_TX_DONE	
159	I2S2_EVT_X_WORDS_RECEIVED	
160	I2S2_EVT_X_WORDS_SENT	
161	ULP_EVT_ERR_INTR	低功耗处理器
162	ULP_EVT_HALT	
163	ULP_EVT_START_INTR	
164	RTC_EVT_TICK	实时时钟定时器
165	RTC_EVT_OVF	
166	RTC_EVT_CMP	
167	GDMA_AHB_EVT_IN_DONE_CH0	GDMA-AHB
168	GDMA_AHB_EVT_IN_DONE_CH1	
169	GDMA_AHB_EVT_IN_DONE_CH2	
170	GDMA_AHB_EVT_IN_SUC_EOF_CH0	
171	GDMA_AHB_EVT_IN_SUC_EOF_CH1	
172	GDMA_AHB_EVT_IN_SUC_EOF_CH2	
173	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0	
174	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1	
175	GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2	
176	GDMA_AHB_EVT_IN_FIFO_FULL_CH0	
177	GDMA_AHB_EVT_IN_FIFO_FULL_CH1	
178	GDMA_AHB_EVT_IN_FIFO_FULL_CH2	
179	GDMA_AHB_EVT_OUT_DONE_CH0	
180	GDMA_AHB_EVT_OUT_DONE_CH1	
181	GDMA_AHB_EVT_OUT_DONE_CH2	
182	GDMA_AHB_EVT_OUT_EOF_CH0	
183	GDMA_AHB_EVT_OUT_EOF_CH1	
184	GDMA_AHB_EVT_OUT_EOF_CH2	
185	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0	
186	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1	
187	GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2	
188	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0	
189	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1	
190	GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2	
191	GDMA_AHB_EVT_OUT_FIFO_FULL_CH0	
192	GDMA_AHB_EVT_OUT_FIFO_FULL_CH1	
193	GDMA_AHB_EVT_OUT_FIFO_FULL_CH2	
194	GDMA_AXI_EVT_IN_DONE_CH0	GDMA-AXI
195	GDMA_AXI_EVT_IN_DONE_CH1	
196	GDMA_AXI_EVT_IN_DONE_CH2	
197	GDMA_AXI_EVT_IN_SUC_EOF_CH0	
198	GDMA_AXI_EVT_IN_SUC_EOF_CH1	
199	GDMA_AXI_EVT_IN_SUC_EOF_CH2	
200	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0	
201	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1	
202	GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
203	GDMA_AXI_EVT_IN_FIFO_FULL_CH0	
204	GDMA_AXI_EVT_IN_FIFO_FULL_CH1	
205	GDMA_AXI_EVT_IN_FIFO_FULL_CH2	
206	GDMA_AXI_EVT_OUT_DONE_CH0	
207	GDMA_AXI_EVT_OUT_DONE_CH1	
208	GDMA_AXI_EVT_OUT_DONE_CH2	
209	GDMA_AXI_EVT_OUT_EOF_CH0	
210	GDMA_AXI_EVT_OUT_EOF_CH1	
211	GDMA_AXI_EVT_OUT_EOF_CH2	
212	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0	
213	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1	
214	GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2	
215	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0	
216	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1	
217	GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2	
218	GDMA_AXI_EVT_OUT_FIFO_FULL_CH0	
219	GDMA_AXI_EVT_OUT_FIFO_FULL_CH1	
220	GDMA_AXI_EVT_OUT_FIFO_FULL_CH2	
221	PMU_EVT_SLEEP_WEEKUP	PMU
222	DMA2D_EVT_IN_DONE_CH0	2D-DMA 控制器 (2D-DMA)
223	DMA2D_EVT_IN_DONE_CH1	
224	DMA2D_EVT_IN_SUC_EOF_CH0	
225	DMA2D_EVT_IN_SUC_EOF_CH1	
226	DMA2D_EVT_OUT_DONE_CH0	
227	DMA2D_EVT_OUT_DONE_CH1	
228	DMA2D_EVT_OUT_DONE_CH2	
229	DMA2D_EVT_OUT_EOF_CH0	
230	DMA2D_EVT_OUT_EOF_CH1	
231	DMA2D_EVT_OUT_EOF_CH2	
232	DMA2D_EVT_OUT_TOTAL_EOF_CH0	
233	DMA2D_EVT_OUT_TOTAL_EOF_CH1	
234	DMA2D_EVT_OUT_TOTAL_EOF_CH2	

以上任意事件产生时，对应外设会生成一个脉冲信号。脉冲信号为高电平时，视为接收到相应的事件。

对于事件更详细的描述，请参考产生该事件外设的章节。

14.3.3 任务

每个事件任务矩阵通道可以选择将本通道的事件映射到所有任务中的任何一个。每个通道都有字段 SOC_ETM_CH*n*_TASK_ID，用于选择要映射到的任务。字段 SOC_ETM_CH*n*_TASK_ID 的配置值与要映射到的任务的关系如表 14.3-2 所示。

表 14.3-2. 事件任务矩阵通道*n* 可映射任务

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
1	GPIO_TASK_CH0_SET	GPIO
2	GPIO_TASK_CH1_SET	
3	GPIO_TASK_CH2_SET	
4	GPIO_TASK_CH3_SET	
5	GPIO_TASK_CH4_SET	
6	GPIO_TASK_CH5_SET	
7	GPIO_TASK_CH6_SET	
8	GPIO_TASK_CH7_SET	
9	GPIO_TASK_CH0_CLEAR	
10	GPIO_TASK_CH1_CLEAR	
11	GPIO_TASK_CH2_CLEAR	
12	GPIO_TASK_CH3_CLEAR	
13	GPIO_TASK_CH4_CLEAR	
14	GPIO_TASK_CH5_CLEAR	
15	GPIO_TASK_CH6_CLEAR	
16	GPIO_TASK_CH7_CLEAR	
17	GPIO_TASK_CH0_TOGGLE	
18	GPIO_TASK_CH1_TOGGLE	
19	GPIO_TASK_CH2_TOGGLE	
20	GPIO_TASK_CH3_TOGGLE	
21	GPIO_TASK_CH4_TOGGLE	
22	GPIO_TASK_CH5_TOGGLE	
23	GPIO_TASK_CH6_TOGGLE	
24	GPIO_TASK_CH7_TOGGLE	
25	LEDC_TASK_TIMER0_RES_UPDATE	LED PWM 控制器 (LEDC)
26	LEDC_TASK_TIMER1_RES_UPDATE	
27	LEDC_TASK_TIMER2_RES_UPDATE	
28	LEDC_TASK_TIMER3_RES_UPDATE	
29	LEDC_TASK_DUTY_SCALE_UPDATE_CH0	
30	LEDC_TASK_DUTY_SCALE_UPDATE_CH1	
31	LEDC_TASK_DUTY_SCALE_UPDATE_CH2	
32	LEDC_TASK_DUTY_SCALE_UPDATE_CH3	
33	LEDC_TASK_DUTY_SCALE_UPDATE_CH4	
34	LEDC_TASK_DUTY_SCALE_UPDATE_CH5	
35	LEDC_TASK_DUTY_SCALE_UPDATE_CH6	
36	LEDC_TASK_DUTY_SCALE_UPDATE_CH7	
37	LEDC_TASK_TIMER0_CAP	
38	LEDC_TASK_TIMER1_CAP	
39	LEDC_TASK_TIMER2_CAP	
40	LEDC_TASK_TIMER3_CAP	
41	LEDC_TASK_SIG_OUT_DIS_CH0	
42	LEDC_TASK_SIG_OUT_DIS_CH1	
43	LEDC_TASK_SIG_OUT_DIS_CH2	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
44	LEDC_TASK_SIG_OUT_DIS_CH3	
45	LEDC_TASK_SIG_OUT_DIS_CH4	
46	LEDC_TASK_SIG_OUT_DIS_CH5	
47	LEDC_TASK_SIG_OUT_DIS_CH6	
48	LEDC_TASK_SIG_OUT_DIS_CH7	
49	LEDC_TASK_OVF_CNT_RST_CH0	
50	LEDC_TASK_OVF_CNT_RST_CH1	
51	LEDC_TASK_OVF_CNT_RST_CH2	
52	LEDC_TASK_OVF_CNT_RST_CH3	
53	LEDC_TASK_OVF_CNT_RST_CH4	
54	LEDC_TASK_OVF_CNT_RST_CH5	
55	LEDC_TASK_OVF_CNT_RST_CH6	
56	LEDC_TASK_OVF_CNT_RST_CH7	
57	LEDC_TASK_TIMER0_RST	
58	LEDC_TASK_TIMER1_RST	
59	LEDC_TASK_TIMER2_RST	
60	LEDC_TASK_TIMER3_RST	
61	LEDC_TASK_TIMER0_RESUME	
62	LEDC_TASK_TIMER1_RESUME	
63	LEDC_TASK_TIMER2_RESUME	
64	LEDC_TASK_TIMER3_RESUME	
65	LEDC_TASK_TIMER0_PAUSE	
66	LEDC_TASK_TIMER1_PAUSE	
67	LEDC_TASK_TIMER2_PAUSE	
68	LEDC_TASK_TIMER3_PAUSE	
69	LEDC_TASK_GAMMA_RESTART_CH0	
70	LEDC_TASK_GAMMA_RESTART_CH1	
71	LEDC_TASK_GAMMA_RESTART_CH2	
72	LEDC_TASK_GAMMA_RESTART_CH3	
73	LEDC_TASK_GAMMA_RESTART_CH4	
74	LEDC_TASK_GAMMA_RESTART_CH5	
75	LEDC_TASK_GAMMA_RESTART_CH6	
76	LEDC_TASK_GAMMA_RESTART_CH7	
77	LEDC_TASK_GAMMA_PAUSE_CH0	
78	LEDC_TASK_GAMMA_PAUSE_CH1	
79	LEDC_TASK_GAMMA_PAUSE_CH2	
80	LEDC_TASK_GAMMA_PAUSE_CH3	
81	LEDC_TASK_GAMMA_PAUSE_CH4	
82	LEDC_TASK_GAMMA_PAUSE_CH5	
83	LEDC_TASK_GAMMA_PAUSE_CH6	
84	LEDC_TASK_GAMMA_PAUSE_CH7	
85	LEDC_TASK_GAMMA_RESUME_CH0	
86	LEDC_TASK_GAMMA_RESUME_CH1	
87	LEDC_TASK_GAMMA_RESUME_CH2	
88	LEDC_TASK_GAMMA_RESUME_CH3	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
89	LEDC_TASK_GAMMA_RESUME_CH4	
90	LEDC_TASK_GAMMA_RESUME_CH5	
91	LEDC_TASK_GAMMA_RESUME_CH6	
92	LEDC_TASK_GAMMA_RESUME_CH7	
93	TG0_TASK_CNT_START_TIMER0	通用定时器 0
94	TG0_TASK_ALARM_START_TIMER0	
95	TG0_TASK_CNT_STOP_TIMER0	
96	TG0_TASK_CNT_RELOAD_TIMER0	
97	TG0_TASK_CNT_CAP_TIMER0	
103	TG1_TASK_CNT_START_TIMER0	通用定时器 1
104	TG1_TASK_ALARM_START_TIMER0	
105	TG1_TASK_CNT_STOP_TIMER0	
106	TG1_TASK_CNT_RELOAD_TIMER0	
107	TG1_TASK_CNT_CAP_TIMER0	
113	MCPWM0_TASK_CMPR0_A_UP	MCPWM0
114	MCPWM0_TASK_CMPR1_A_UP	
115	MCPWM0_TASK_CMPR2_A_UP	
116	MCPWM0_TASK_CMPR0_B_UP	
117	MCPWM0_TASK_CMPR1_B_UP	
118	MCPWM0_TASK_CMPR2_B_UP	
119	MCPWM0_TASK_GEN_STOP	
120	MCPWM0_TASK_TIMER0_SYN	
121	MCPWM0_TASK_TIMER1_SYN	
122	MCPWM0_TASK_TIMER2_SYN	
123	MCPWM0_TASK_TIMER0_PERIOD_UP	
124	MCPWM0_TASK_TIMER1_PERIOD_UP	
125	MCPWM0_TASK_TIMER2_PERIOD_UP	
126	MCPWM0_TASK_TZ0_OST	
127	MCPWM0_TASK_TZ1_OST	
128	MCPWM0_TASK_TZ2_OST	
129	MCPWM0_TASK_CLR0_OST	
130	MCPWM0_TASK_CLR1_OST	
131	MCPWM0_TASK_CLR2_OST	
132	MCPWM0_TASK_CAP0	
133	MCPWM0_TASK_CAP1	
134	MCPWM0_TASK_CAP2	
135	MCPWM1_TASK_CMPR0_A_UP	MCPWM1
136	MCPWM1_TASK_CMPR1_A_UP	
137	MCPWM1_TASK_CMPR2_A_UP	
138	MCPWM1_TASK_CMPR0_B_UP	
139	MCPWM1_TASK_CMPR1_B_UP	
140	MCPWM1_TASK_CMPR2_B_UP	
141	MCPWM1_TASK_GEN_STOP	
142	MCPWM1_TASK_TIMER0_SYN	
143	MCPWM1_TASK_TIMER1_SYN	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
144	MCPWM1_TASK_TIMER2_SYN	
145	MCPWM1_TASK_TIMER0_PERIOD_UP	
146	MCPWM1_TASK_TIMER1_PERIOD_UP	
147	MCPWM1_TASK_TIMER2_PERIOD_UP	
148	MCPWM1_TASK_TZ0_OST	
149	MCPWM1_TASK_TZ1_OST	
150	MCPWM1_TASK_TZ2_OST	
151	MCPWM1_TASK_CLR0_OST	
152	MCPWM1_TASK_CLR1_OST	
153	MCPWM1_TASK_CLR2_OST	
154	MCPWM1_TASK_CAP0	
155	MCPWM1_TASK_CAP1	
156	MCPWM1_TASK_CAP2	
157	ADC_TASK_SAMPLE0	ADC 控制器 (ADC)
159	ADC_TASK_START0	
160	ADC_TASK_STOP0	
165	TMPSNSR_TASK_START_SAMPLE	温度传感器 (TSENS)
166	TMPSNSR_TASK_STOP_SAMPLE	
167	I2S0_TASK_START_RX	I2S0
168	I2S0_TASK_START_TX	
169	I2S0_TASK_STOP_RX	
170	I2S0_TASK_STOP_TX	
171	I2S1_TASK_START_RX	I2S1
172	I2S1_TASK_START_TX	
173	I2S1_TASK_STOP_RX	
174	I2S1_TASK_STOP_TX	
175	I2S2_TASK_START_RX	I2S2
176	I2S2_TASK_START_TX	
177	I2S2_TASK_STOP_RX	
178	I2S2_TASK_STOP_TX	
179	ULP_TASK_WEAKUP_CPU	低功耗处理器
180	ULP_TASK_INT_CPU	
185	GDMA_AHB_TASK_IN_START_CH0	GDMA-AHB
186	GDMA_AHB_TASK_IN_START_CH1	
187	GDMA_AHB_TASK_IN_START_CH2	
188	GDMA_AHB_TASK_OUT_START_CH0	
189	GDMA_AHB_TASK_OUT_START_CH1	
190	GDMA_AHB_TASK_OUT_START_CH2	
191	GDMA_AXI_TASK_IN_START_CH0	GDMA-AXI
192	GDMA_AXI_TASK_IN_START_CH1	
193	GDMA_AXI_TASK_IN_START_CH2	
194	GDMA_AXI_TASK_OUT_START_CH0	
195	GDMA_AXI_TASK_OUT_START_CH1	
196	GDMA_AXI_TASK_OUT_START_CH2	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
197	PMU_TASK_SLEEP_REQ	PMU
198	DMA2D_TASK_IN_START_CH0	2D-DMA 控制器 (2D-DMA)
199	DMA2D_TASK_IN_START_CH1	
200	DMA2D_TASK_IN_DSCR_READY_CH0	
201	DMA2D_TASK_IN_DSCR_READY_CH1	
202	DMA2D_TASK_OUT_START_CH0	
203	DMA2D_TASK_OUT_START_CH1	
204	DMA2D_TASK_OUT_START_CH2	
205	DMA2D_TASK_OUT_DSCR_READY_CH0	
206	DMA2D_TASK_OUT_DSCR_READY_CH1	
207	DMA2D_TASK_OUT_DSCR_READY_CH2	

一个通道接收到一个有效事件脉冲信号时，会产生映射任务脉冲信号。

对于任务更详细的描述，请参考接收该任务外设的章节。

来自多个不同通道的事件可以选择映射到同一个任务。例如，可以将多个通道的字段 SOC_ETM_CH*n*_TASK_ID 都配置为相同的值，而字段 SOC_ETM_CH*n*_EVT_ID 可以配置为相同或不同的值。在这种情况下，当任意一个通道接收到的事件有效时，指定的任务就会被生成。当多个通道接收到的事件同时有效时，指定的任务只会产生一次。

14.3.4 事件任务触发状态

ETM 外设支持通过读取寄存器来查看事件和任务的触发状态。

- 事件触发状态：读取寄存器 SOC_ETM_EVT_ST*n*_REG 中所要查看事件对应的域，如果该域的值为 1，表示对应的事件被 ETM 接收过，如果该域的值为 0，表示对应的事件未被 ETM 接收到。向寄存器 SOC_ETM_EVT_ST*n*_CLR_REG 中所要查看事件对应的域写 1，可以将寄存器 SOC_ETM_EVT_ST*n*_REG 中所要查看事件对应的域清 0。
- 任务触发状态：读取寄存器 SOC_ETM_TASK_ST*n*_REG 中所要查看任务对应的域，如果该域的值为 1，表示 ETM 产生过对应的任务，如果该域的值为 0，表示 ETM 未产生过对应的任务。向寄存器 SOC_ETM_TASK_ST*n*_CLR_REG 中所要查看任务对应的域写 1，可以将寄存器 SOC_ETM_TASK_ST*n*_REG 中所要查看任务对应的域清 0。

14.3.5 时序考虑因素

ETM 外设接收的事件、发送的任务和 ETM 通道之间驱动时钟的结构如图 14.3-2 所示。

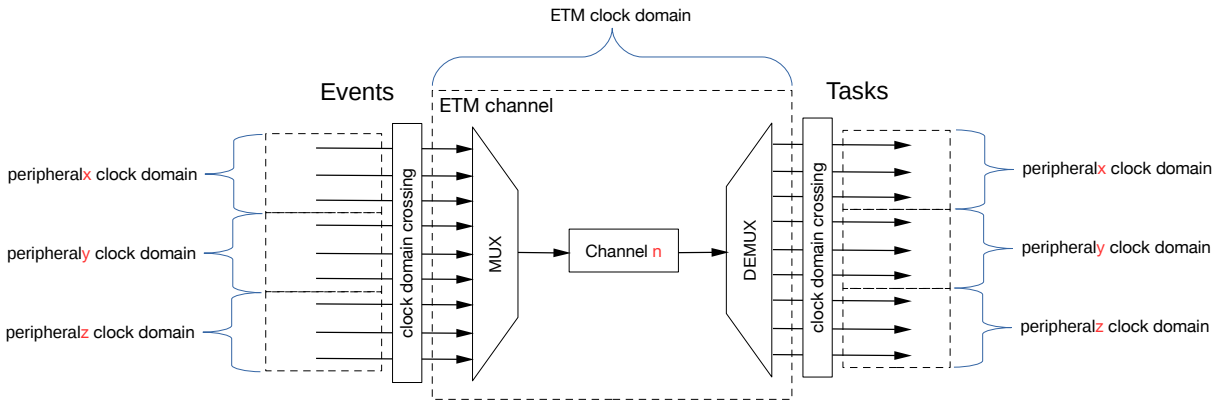


图 14.3-2. 事件任务矩阵时钟结构

ETM 运行在 SYS_CLK 时钟域下（详见章节 11 复位和时钟），每个事件对应一个由相应外设在其时钟域中产生的脉冲信号，每个任务被 ETM 映射为对应外设时钟域下的脉冲信号。由于产生事件的外设、事件任务矩阵和接收任务的外设不一定由同个时钟驱动，需要同步处理，两个连续时间之间需要根据生成事件外设的时钟、ETM 的时钟、执行任务外设的时钟设置延迟，否则会丢失事件。

为了让事件任务矩阵成功接收到每个事件，对于每个可以产生事件的外设，其产生的两个连续事件脉冲之间的时间间隔必须大于一个事件任务矩阵的时钟周期，即 $ceil(\frac{peripheral_clock_frequenty}{ETM_clock_frequenty})$ 个外设时钟周期。

例如，假设外设 A 所在时钟域（REF_F80M_CLK）为 80 MHz，ETM 所在时钟域（SYS_CLK）为 40 MHz，外设 A 生成事件 1，ETM 接收事件 1。要成功接收每个事件 1，两个连续事件 1 之间的时间间隔必须大于两个外设 A 的时钟周期（即一个 ETM 时钟周期）。

同样的，为了使事件任务矩阵成功地将接收到的事件（即同步到事件任务矩阵时钟域的事件）映射为外设的任务，事件任务矩阵时钟域中的两个连续事件脉冲之间的时间间隔必须大于一个外设的时钟周期，即 $ceil(\frac{ETM_clock_frequenty}{peripheral_clock_frequenty})$ 个事件任务矩阵时钟周期。

例如，假设外设 B 所在时钟域（FOSC_20M_CLK）为 20 MHz，ETM 所在时钟域（SYS_CLK）为 40 MHz，ETM 将事件映射到任务 1，外设 B 接收任务 1。要成功将每个接收到的事件映射到任务 1，两个连续事件之间的时间间隔必须大于两个 ETM 时钟周期（即一个外设 B 时钟周期）。

因此，要将外设 A 产生的两个连续事件完整映射到外设 B 的两个任务中，这两个事件之间的时间间隔必须大于 $ceil(\frac{peripheral_A_clock_frequenty}{ETM_clock_frequenty}) * ceil(\frac{ETM_clock_frequenty}{peripheral_B_clock_frequenty})$ 个外设 A 时钟周期。

例如，假设外设 A 所在时钟域（REF_F80M_CLK）为 80 MHz，外设 B 所在时钟域（FOSC_20M_CLK）为 20 MHz，ETM 所在时钟域（SYS_CLK）为 40 MHz，外设 A 生成事件 1，ETM 将事件映射到任务 1，外设 B 接收任务 1。要成功将每个事件 1（外设 A 生成）映射到任务 1（外设 B 接收），两个连续事件 1 之间的时间间隔必须大于 $2 * 2 = 4$ 个外设 A 时钟周期。

14.3.6 通道控制

事件任务矩阵的每个通道都可以独立配置为使能或关闭。当通道*n* 配置为使能时，如果该通道接收到所配置的事件（字段 SOC_ETM_CH*n*_EVT_ID），该事件将映射到所配置的任务上（字段 SOC_ETM_CH*n*_TASK_ID）。当通道*n* 配置为不使能时，即使该通道接收到所配置的事件（字段 SOC_ETM_CH*n*_EVT_ID），也不会产生任何任务。

使能事件任务矩阵通道*n*，需向字段 SOC_ETM_CH_ENABLE*n* 写 1。关闭事件任务矩阵通道*n*，需向字段 SOC_ETM_CH_DISABLE*n* 写 1。

通道*n* 的状态可通过读取字段 SOC_ETM_CH_ENABLED*n* 的值获取。1 表示通道*n* 已使能，0 表示通道*n* 未使能。

如果将字段 SOC_ETM_CH*n*_EVT_ID 的值配置为 0 或将字段 SOC_ETM_CH*n*_TASK_ID 的值配置为 0，事件任务矩阵通道*n* 也将被关闭。

事件任务矩阵通道*n* 的完整配置过程如下：

1. 向字段 HP_SYS_CLKRST_ETM_SYS_CLK_EN 写 1，开启 ETM 的时钟
2. 配置字段 SOC_ETM_CH*n*_EVT_ID 的值，选择通道*n* 所要接收的事件
3. 配置字段 SOC_ETM_CH*n*_TASK_ID 的值，选择通道*n* 所接收到的事件要映射到的任务
4. 向字段 SOC_ETM_CH_ENABLE*n* 写 1 使能通道
5. 当通道*n* 不再需要将配置的事件映射到配置的任务时，向字段 SOC_ETM_CH_DISABLE*n* 写 1 来关闭通道*n*。之后，如果要为通道*n* 配置新的事件和任务的映射关系，则重复步骤 2-4，如果不做任何配置，则通道*n* 保持停止运行状态
6. 若需复位整个 ETM，即复位所有通道的配置，可通过向 HP_SYS_CLKRST_RST_EN_ETM 字段先写 1 再写 0 来实现

14.4 寄存器列表

本小节的所有地址均为相对于事件任务矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
状态寄存器			
SOC_ETM_CH_ENA_AD0_REG	通道状态寄存器	0x0000	R/ WTC/ WTS
SOC_ETM_CH_ENA_AD1_REG	通道状态寄存器	0x000C	R/ WTC/ WTS
SOC_ETM_EVT_ST0_REG	事件状态寄存器	0x01A8	R/WTC/SS
SOC_ETM_EVT_ST1_REG	事件状态寄存器	0x01B0	R/WTC/SS
SOC_ETM_EVT_ST2_REG	事件状态寄存器	0x01B8	R/WTC/SS
SOC_ETM_EVT_ST3_REG	事件状态寄存器	0x01C0	R/WTC/SS
SOC_ETM_EVT_ST4_REG	事件状态寄存器	0x01C8	R/WTC/SS
SOC_ETM_EVT_ST5_REG	事件状态寄存器	0x01D0	R/WTC/SS
SOC_ETM_EVT_ST6_REG	事件状态寄存器	0x01D8	R/WTC/SS
SOC_ETM_EVT_ST7_REG	事件状态寄存器	0x01E0	R/WTC/SS
SOC_ETM_TASK_ST0_REG	任务状态寄存器	0x01E8	R/WTC/SS
SOC_ETM_TASK_ST1_REG	任务状态寄存器	0x01F0	R/WTC/SS
SOC_ETM_TASK_ST2_REG	任务状态寄存器	0x01F8	R/WTC/SS
SOC_ETM_TASK_ST3_REG	任务状态寄存器	0x0200	R/WTC/SS
SOC_ETM_TASK_ST4_REG	任务状态寄存器	0x0208	R/WTC/SS
SOC_ETM_TASK_ST5_REG	任务状态寄存器	0x0210	R/WTC/SS
SOC_ETM_TASK_ST6_REG	任务状态寄存器	0x0218	R/WTC/SS
配置寄存器			
SOC_ETM_CH_ENA_AD0_SET_REG	通道使能寄存器	0x0004	WT
SOC_ETM_CH_ENA_AD0_CLR_REG	通道关闭寄存器	0x0008	WT
SOC_ETM_CH_ENA_AD1_SET_REG	通道使能寄存器	0x0010	WT
SOC_ETM_CH_ENA_AD1_CLR_REG	通道关闭寄存器	0x0014	WT
SOC_ETM_CH0_EVT_ID_REG	通道 0 事件 ID 寄存器	0x0018	R/W
SOC_ETM_CH0_TASK_ID_REG	通道 0 任务 ID 寄存器	0x001C	R/W
SOC_ETM_CH1_EVT_ID_REG	通道 1 事件 ID 寄存器	0x0020	R/W
SOC_ETM_CH1_TASK_ID_REG	通道 1 任务 ID 寄存器	0x0024	R/W
SOC_ETM_CH2_EVT_ID_REG	通道 2 事件 ID 寄存器	0x0028	R/W
SOC_ETM_CH2_TASK_ID_REG	通道 2 任务 ID 寄存器	0x002C	R/W
SOC_ETM_CH3_EVT_ID_REG	通道 3 事件 ID 寄存器	0x0030	R/W
SOC_ETM_CH3_TASK_ID_REG	通道 3 任务 ID 寄存器	0x0034	R/W
SOC_ETM_CH4_EVT_ID_REG	通道 4 事件 ID 寄存器	0x0038	R/W
SOC_ETM_CH4_TASK_ID_REG	通道 4 任务 ID 寄存器	0x003C	R/W
SOC_ETM_CH5_EVT_ID_REG	通道 5 事件 ID 寄存器	0x0040	R/W
SOC_ETM_CH5_TASK_ID_REG	通道 5 任务 ID 寄存器	0x0044	R/W

名称	描述	地址	访问
SOC_ETM_CH6_EVT_ID_REG	通道 6 事件 ID 寄存器	0x0048	R/W
SOC_ETM_CH6_TASK_ID_REG	通道 6 任务 ID 寄存器	0x004C	R/W
SOC_ETM_CH7_EVT_ID_REG	通道 7 事件 ID 寄存器	0x0050	R/W
SOC_ETM_CH7_TASK_ID_REG	通道 7 任务 ID 寄存器	0x0054	R/W
SOC_ETM_CH8_EVT_ID_REG	通道 8 事件 ID 寄存器	0x0058	R/W
SOC_ETM_CH8_TASK_ID_REG	通道 8 任务 ID 寄存器	0x005C	R/W
SOC_ETM_CH9_EVT_ID_REG	通道 9 事件 ID 寄存器	0x0060	R/W
SOC_ETM_CH9_TASK_ID_REG	通道 9 任务 ID 寄存器	0x0064	R/W
SOC_ETM_CH10_EVT_ID_REG	通道 10 事件 ID 寄存器	0x0068	R/W
SOC_ETM_CH10_TASK_ID_REG	通道 10 任务 ID 寄存器	0x006C	R/W
SOC_ETM_CH11_EVT_ID_REG	通道 11 事件 ID 寄存器	0x0070	R/W
SOC_ETM_CH11_TASK_ID_REG	通道 11 任务 ID 寄存器	0x0074	R/W
SOC_ETM_CH12_EVT_ID_REG	通道 12 事件 ID 寄存器	0x0078	R/W
SOC_ETM_CH12_TASK_ID_REG	通道 12 任务 ID 寄存器	0x007C	R/W
SOC_ETM_CH13_EVT_ID_REG	通道 13 事件 ID 寄存器	0x0080	R/W
SOC_ETM_CH13_TASK_ID_REG	通道 13 任务 ID 寄存器	0x0084	R/W
SOC_ETM_CH14_EVT_ID_REG	通道 14 事件 ID 寄存器	0x0088	R/W
SOC_ETM_CH14_TASK_ID_REG	通道 14 任务 ID 寄存器	0x008C	R/W
SOC_ETM_CH15_EVT_ID_REG	通道 15 事件 ID 寄存器	0x0090	R/W
SOC_ETM_CH15_TASK_ID_REG	通道 15 任务 ID 寄存器	0x0094	R/W
SOC_ETM_CH16_EVT_ID_REG	通道 16 事件 ID 寄存器	0x0098	R/W
SOC_ETM_CH16_TASK_ID_REG	通道 16 任务 ID 寄存器	0x009C	R/W
SOC_ETM_CH17_EVT_ID_REG	通道 17 事件 ID 寄存器	0x00A0	R/W
SOC_ETM_CH17_TASK_ID_REG	通道 17 任务 ID 寄存器	0x00A4	R/W
SOC_ETM_CH18_EVT_ID_REG	通道 18 事件 ID 寄存器	0x00A8	R/W
SOC_ETM_CH18_TASK_ID_REG	通道 18 任务 ID 寄存器	0x00AC	R/W
SOC_ETM_CH19_EVT_ID_REG	通道 19 事件 ID 寄存器	0x00B0	R/W
SOC_ETM_CH19_TASK_ID_REG	通道 19 任务 ID 寄存器	0x00B4	R/W
SOC_ETM_CH20_EVT_ID_REG	通道 20 事件 ID 寄存器	0x00B8	R/W
SOC_ETM_CH20_TASK_ID_REG	通道 20 任务 ID 寄存器	0x00BC	R/W
SOC_ETM_CH21_EVT_ID_REG	通道 21 事件 ID 寄存器	0x00C0	R/W
SOC_ETM_CH21_TASK_ID_REG	通道 21 任务 ID 寄存器	0x00C4	R/W
SOC_ETM_CH22_EVT_ID_REG	通道 22 事件 ID 寄存器	0x00C8	R/W
SOC_ETM_CH22_TASK_ID_REG	通道 22 任务 ID 寄存器	0x00CC	R/W
SOC_ETM_CH23_EVT_ID_REG	通道 23 事件 ID 寄存器	0x00D0	R/W
SOC_ETM_CH23_TASK_ID_REG	通道 23 任务 ID 寄存器	0x00D4	R/W
SOC_ETM_CH24_EVT_ID_REG	通道 24 事件 ID 寄存器	0x00D8	R/W
SOC_ETM_CH24_TASK_ID_REG	通道 24 任务 ID 寄存器	0x00DC	R/W
SOC_ETM_CH25_EVT_ID_REG	通道 25 事件 ID 寄存器	0x00E0	R/W
SOC_ETM_CH25_TASK_ID_REG	通道 25 任务 ID 寄存器	0x00E4	R/W
SOC_ETM_CH26_EVT_ID_REG	通道 26 事件 ID 寄存器	0x00E8	R/W
SOC_ETM_CH26_TASK_ID_REG	通道 26 任务 ID 寄存器	0x00EC	R/W
SOC_ETM_CH27_EVT_ID_REG	通道 27 事件 ID 寄存器	0x00F0	R/W

名称	描述	地址	访问
SOC_ETM_CH27_TASK_ID_REG	通道 27 任务 ID 寄存器	0x00F4	R/W
SOC_ETM_CH28_EVT_ID_REG	通道 28 事件 ID 寄存器	0x00F8	R/W
SOC_ETM_CH28_TASK_ID_REG	通道 28 任务 ID 寄存器	0x00FC	R/W
SOC_ETM_CH29_EVT_ID_REG	通道 29 事件 ID 寄存器	0x0100	R/W
SOC_ETM_CH29_TASK_ID_REG	通道 29 任务 ID 寄存器	0x0104	R/W
SOC_ETM_CH30_EVT_ID_REG	通道 30 事件 ID 寄存器	0x0108	R/W
SOC_ETM_CH30_TASK_ID_REG	通道 30 任务 ID 寄存器	0x010C	R/W
SOC_ETM_CH31_EVT_ID_REG	通道 31 事件 ID 寄存器	0x0110	R/W
SOC_ETM_CH31_TASK_ID_REG	通道 31 任务 ID 寄存器	0x0114	R/W
SOC_ETM_CH32_EVT_ID_REG	通道 32 事件 ID 寄存器	0x0118	R/W
SOC_ETM_CH32_TASK_ID_REG	通道 32 任务 ID 寄存器	0x011C	R/W
SOC_ETM_CH33_EVT_ID_REG	通道 33 事件 ID 寄存器	0x0120	R/W
SOC_ETM_CH33_TASK_ID_REG	通道 33 任务 ID 寄存器	0x0124	R/W
SOC_ETM_CH34_EVT_ID_REG	通道 34 事件 ID 寄存器	0x0128	R/W
SOC_ETM_CH34_TASK_ID_REG	通道 34 任务 ID 寄存器	0x012C	R/W
SOC_ETM_CH35_EVT_ID_REG	通道 35 事件 ID 寄存器	0x0130	R/W
SOC_ETM_CH35_TASK_ID_REG	通道 35 任务 ID 寄存器	0x0134	R/W
SOC_ETM_CH36_EVT_ID_REG	通道 36 事件 ID 寄存器	0x0138	R/W
SOC_ETM_CH36_TASK_ID_REG	通道 36 任务 ID 寄存器	0x013C	R/W
SOC_ETM_CH37_EVT_ID_REG	通道 37 事件 ID 寄存器	0x0140	R/W
SOC_ETM_CH37_TASK_ID_REG	通道 37 任务 ID 寄存器	0x0144	R/W
SOC_ETM_CH38_EVT_ID_REG	通道 38 事件 ID 寄存器	0x0148	R/W
SOC_ETM_CH38_TASK_ID_REG	通道 38 任务 ID 寄存器	0x014C	R/W
SOC_ETM_CH39_EVT_ID_REG	通道 39 事件 ID 寄存器	0x0150	R/W
SOC_ETM_CH39_TASK_ID_REG	通道 39 任务 ID 寄存器	0x0154	R/W
SOC_ETM_CH40_EVT_ID_REG	通道 40 事件 ID 寄存器	0x0158	R/W
SOC_ETM_CH40_TASK_ID_REG	通道 40 任务 ID 寄存器	0x015C	R/W
SOC_ETM_CH41_EVT_ID_REG	通道 41 事件 ID 寄存器	0x0160	R/W
SOC_ETM_CH41_TASK_ID_REG	通道 41 任务 ID 寄存器	0x0164	R/W
SOC_ETM_CH42_EVT_ID_REG	通道 42 事件 ID 寄存器	0x0168	R/W
SOC_ETM_CH42_TASK_ID_REG	通道 42 任务 ID 寄存器	0x016C	R/W
SOC_ETM_CH43_EVT_ID_REG	通道 43 事件 ID 寄存器	0x0170	R/W
SOC_ETM_CH43_TASK_ID_REG	通道 43 任务 ID 寄存器	0x0174	R/W
SOC_ETM_CH44_EVT_ID_REG	通道 44 事件 ID 寄存器	0x0178	R/W
SOC_ETM_CH44_TASK_ID_REG	通道 44 任务 ID 寄存器	0x017C	R/W
SOC_ETM_CH45_EVT_ID_REG	通道 45 事件 ID 寄存器	0x0180	R/W
SOC_ETM_CH45_TASK_ID_REG	通道 45 任务 ID 寄存器	0x0184	R/W
SOC_ETM_CH46_EVT_ID_REG	通道 46 事件 ID 寄存器	0x0188	R/W
SOC_ETM_CH46_TASK_ID_REG	通道 46 任务 ID 寄存器	0x018C	R/W
SOC_ETM_CH47_EVT_ID_REG	通道 47 事件 ID 寄存器	0x0190	R/W
SOC_ETM_CH47_TASK_ID_REG	通道 47 任务 ID 寄存器	0x0194	R/W
SOC_ETM_CH48_EVT_ID_REG	通道 48 事件 ID 寄存器	0x0198	R/W
SOC_ETM_CH48_TASK_ID_REG	通道 48 任务 ID 寄存器	0x019C	R/W

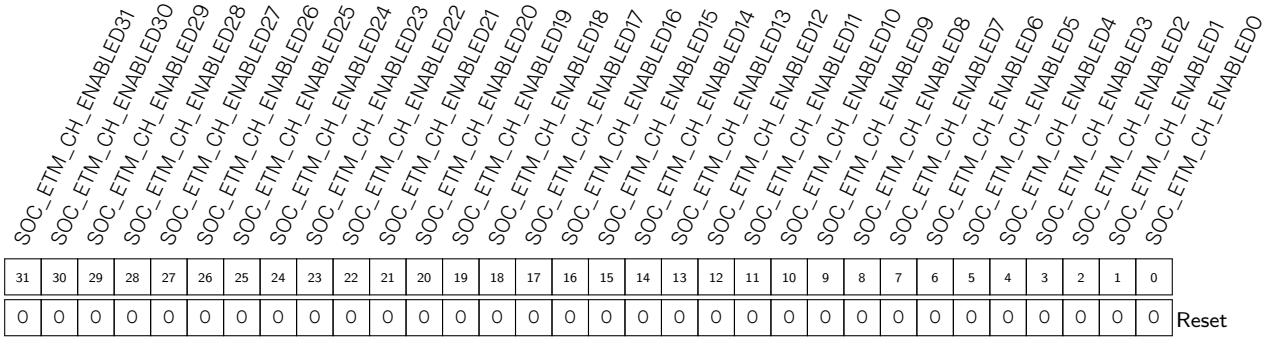
名称	描述	地址	访问
SOC_ETM_CH49_EVT_ID_REG	通道 49 事件 ID 寄存器	0x01A0	R/W
SOC_ETM_CH49_TASK_ID_REG	通道 49 任务 ID 寄存器	0x01A4	R/W
SOC_ETM_EVT_ST0_CLR_REG	事件状态清除寄存器	0x01AC	WT
SOC_ETM_EVT_ST1_CLR_REG	事件状态清除寄存器	0x01B4	WT
SOC_ETM_EVT_ST2_CLR_REG	事件状态清除寄存器	0x01BC	WT
SOC_ETM_EVT_ST3_CLR_REG	事件状态清除寄存器	0x01C4	WT
SOC_ETM_EVT_ST4_CLR_REG	事件状态清除寄存器	0x01CC	WT
SOC_ETM_EVT_ST5_CLR_REG	事件状态清除寄存器	0x01D4	WT
SOC_ETM_EVT_ST6_CLR_REG	事件状态清除寄存器	0x01DC	WT
SOC_ETM_EVT_ST7_CLR_REG	事件状态清除寄存器	0x01E4	WT
SOC_ETM_TASK_ST0_CLR_REG	任务状态清除寄存器	0x01EC	WT
SOC_ETM_TASK_ST1_CLR_REG	任务状态清除寄存器	0x01F4	WT
SOC_ETM_TASK_ST2_CLR_REG	任务状态清除寄存器	0x01FC	WT
SOC_ETM_TASK_ST3_CLR_REG	任务状态清除寄存器	0x0204	WT
SOC_ETM_TASK_ST4_CLR_REG	任务状态清除寄存器	0x020C	WT
SOC_ETM_TASK_ST5_CLR_REG	任务状态清除寄存器	0x0214	WT
SOC_ETM_TASK_ST6_CLR_REG	任务状态清除寄存器	0x021C	WT
SOC_ETM_CLK_EN_REG	ETM 时钟使能寄存器	0x0220	R/W
版本寄存器			
SOC_ETM_DATE_REG	版本控制寄存器	0x0224	R/W

14.5 寄存器

本小节的所有地址均为相对于事件任务矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 14.1. SOC_ETM_CH_ENA_AD0_REG (0x0000)



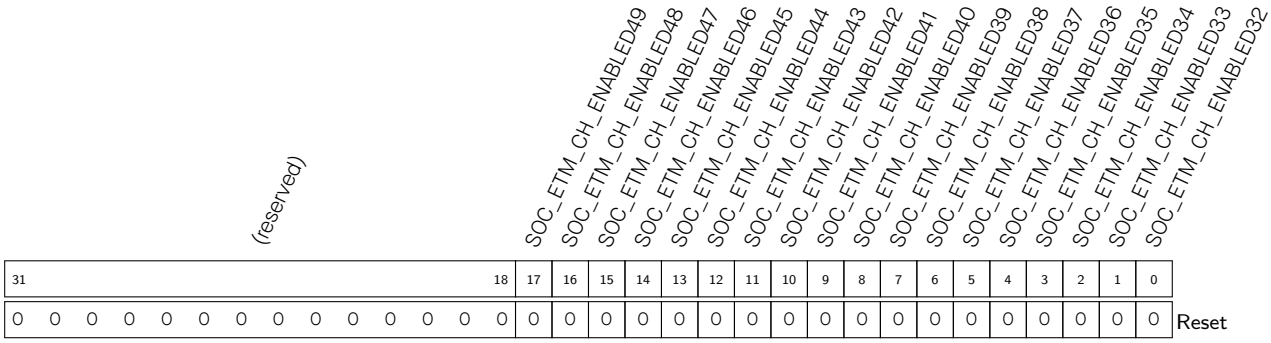
SOC_ETM_CH_ENABLED*n* (*n*: 0-31) 表示通道*n* 的状态。

0：已关闭

1：已使能

(R/WTC/SS)

Register 14.2. SOC_ETM_CH_ENA_AD1_REG (0x000C)



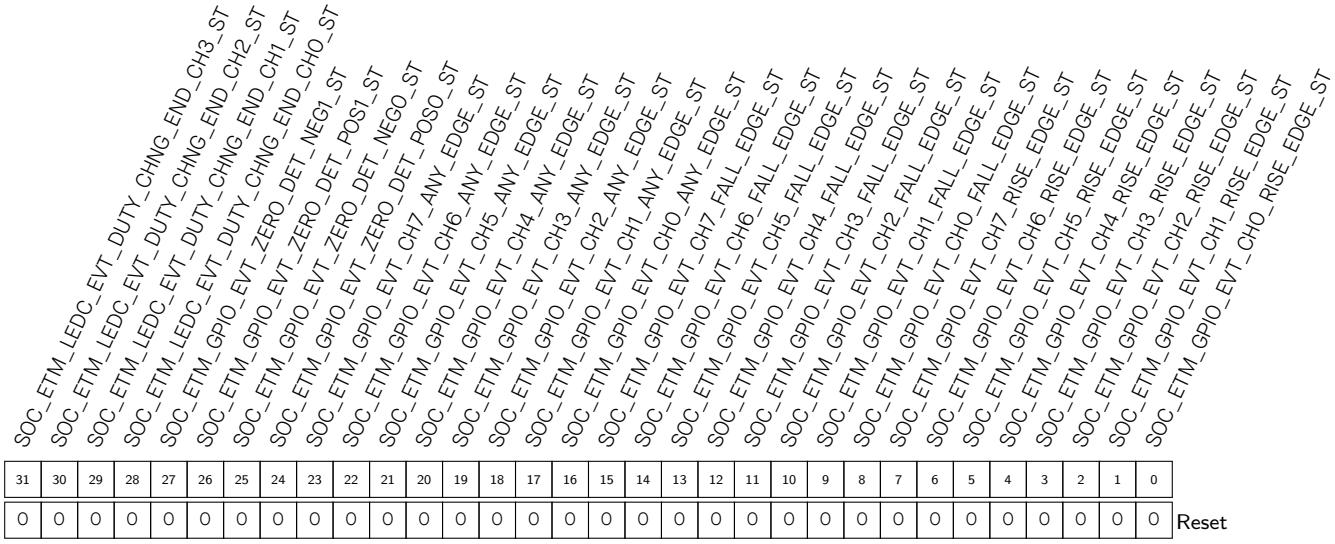
SOC_ETM_CH_ENABLED*n* (*n*: 32-49) 表示通道*n* 的状态。

0：已关闭

1：已使能

(R/WTC/SS)

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST 表示 GPIO_EVT_CH0_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST 表示 GPIO_EVT_CH1_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST 表示 GPIO_EVT_CH2_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST 表示 GPIO_EVT_CH3_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST 表示 GPIO_EVT_CH4_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST 表示 GPIO_EVT_CH5_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST 表示 GPIO_EVT_CH6_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST 表示 GPIO_EVT_CH7_RISE_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST 表示 GPIO_EVT_CH0_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST 表示 GPIO_EVT_CH1_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST 表示 GPIO_EVT_CH2_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST 表示 GPIO_EVT_CH3_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST 表示 GPIO_EVT_CH4_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST 表示 GPIO_EVT_CH5_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST 表示 GPIO_EVT_CH6_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST 表示 GPIO_EVT_CH7_FALL_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST 表示 GPIO_EVT_CH0_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST 表示 GPIO_EVT_CH1_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST 表示 GPIO_EVT_CH2_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST 表示 GPIO_EVT_CH3_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST 表示 GPIO_EVT_CH4_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST 表示 GPIO_EVT_CH5_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST 表示 GPIO_EVT_CH6_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST 表示 GPIO_EVT_CH7_ANY_EDGE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST 表示 GPIO_EVT_ZERO_DET_POS0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.3. SOC_ETM_EVT_ST0_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST 表示 GPIO_EVT_ZERO_DET_NEG0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS1_ST 表示 GPIO_EVT_ZERO_DET_POS1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG1_ST 表示 GPIO_EVT_ZERO_DET_NEG1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

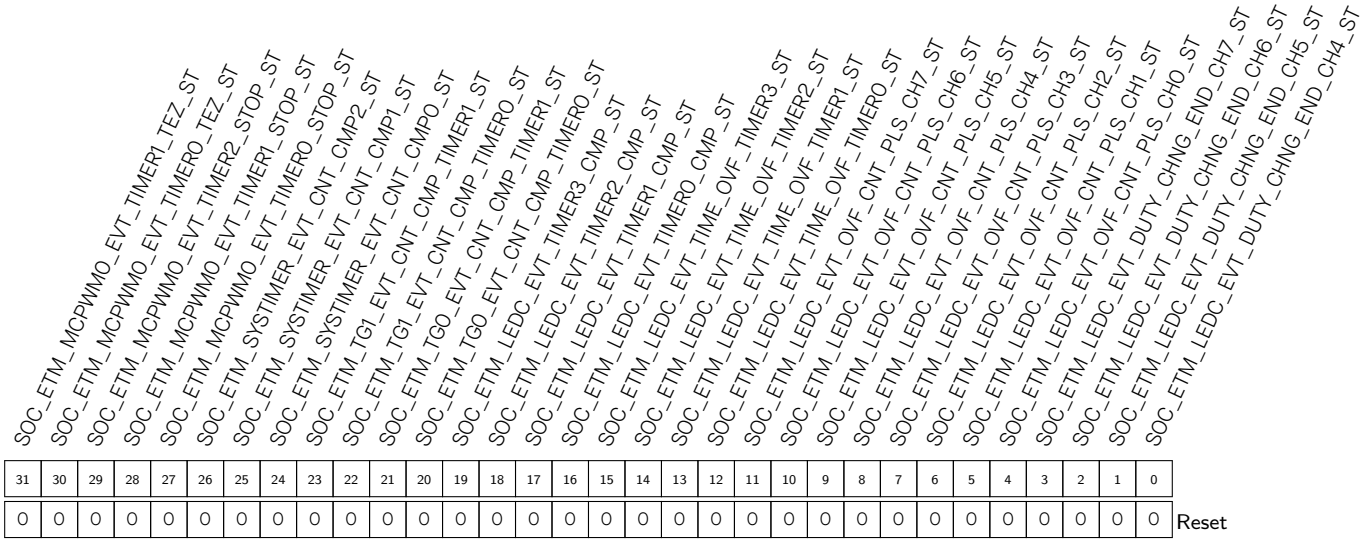
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH3 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)



SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH4 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH5 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH6_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH6 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH7_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH7 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH3 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH4 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH5 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH6_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH6 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH7_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH7 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST 表示 LEDC_EVT_TIME_OVF_TIMER0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST 表示 LEDC_EVT_TIME_OVF_TIMER1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST 表示 LEDC_EVT_TIME_OVF_TIMER2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST 表示 LEDC_EVT_TIME_OVF_TIMER3 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST 表示 LEDC_EVT_TIMER0_CMP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST 表示 LEDC_EVT_TIMER1_CMP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST 表示 LEDC_EVT_TIMER2_CMP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST 表示 LEDC_EVT_TIMER3_CMP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST 表示 TG0_EVT_CNT_CMP_TIMER0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST 表示 TG0_EVT_CNT_CMP_TIMER1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST 表示 TG1_EVT_CNT_CMP_TIMER0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST 表示 TG1_EVT_CNT_CMP_TIMER1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST 表示 SYSTIMER_EVT_CNT_CMP0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST 表示 SYSTIMER_EVT_CNT_CMP1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST 表示 SYSTIMER_EVT_CNT_CMP2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST 表示 MCPWM0_EVT_TIMER0_STOP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST 表示 MCPWM0_EVT_TIMER1_STOP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST 表示 MCPWM0_EVT_TIMER2_STOP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST 表示 MCPWM0_EVT_TIMER0_TEZ 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

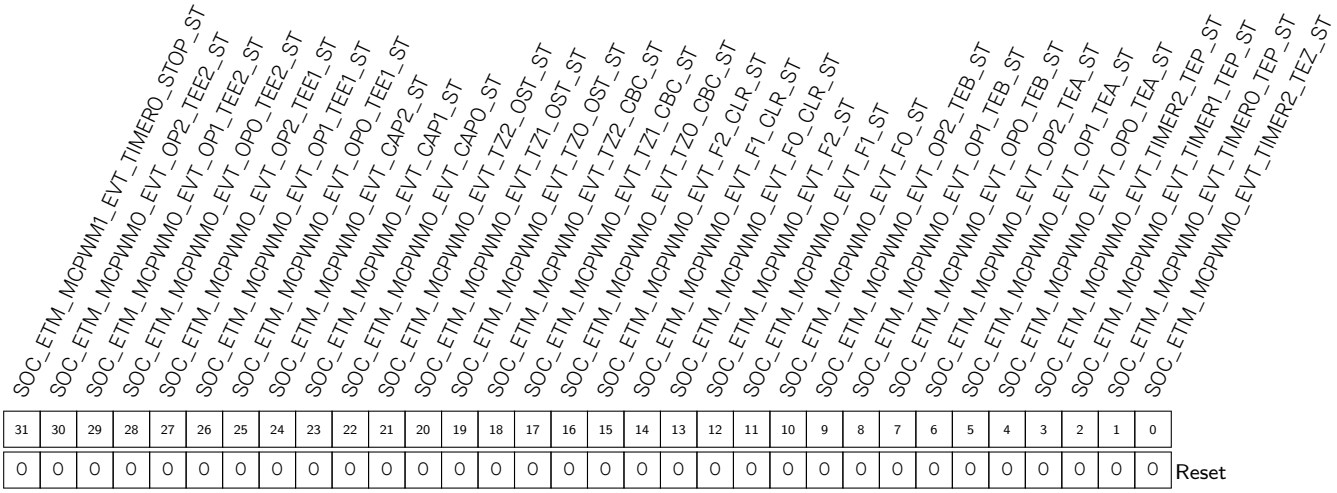
SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST 表示 MCPWM0_EVT_TIMER1_TEZ 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)



SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST 表示 MCPWM0_EVT_TIMER2_TEZ 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST 表示 MCPWM0_EVT_TIMER0_TEP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST 表示 MCPWM0_EVT_TIMER1_TEP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST 表示 MCPWM0_EVT_TIMER2_TEP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST 表示 MCPWM0_EVT_OP0_TEA 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEA_ST 表示 MCPWM0_EVT_OP1_TEA 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEA_ST 表示 MCPWM0_EVT_OP2_TEA 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST 表示 MCPWM0_EVT_OP0_TEB 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST 表示 MCPWM0_EVT_OP1_TEB 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST 表示 MCPWM0_EVT_OP2_TEB 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F0_ST 表示 MCPWM0_EVT_F0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_ST 表示 MCPWM0_EVT_F1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F2_ST 表示 MCPWM0_EVT_F2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F0_CLR_ST 表示 MCPWM0_EVT_F0_CLR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_CLR_ST 表示 MCPWM0_EVT_F1_CLR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F2_CLR_ST 表示 MCPWM0_EVT_F2_CLR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST 表示 MCPWM0_EVT_TZ0_CBC 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST 表示 MCPWM0_EVT_TZ1_CBC 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST 表示 MCPWM0_EVT_TZ2_CBC 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST 表示 MCPWM0_EVT_TZ0_OST 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST 表示 MCPWM0_EVT_TZ1_OST 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST 表示 MCPWM0_EVT_TZ2_OST 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP0_ST 表示 MCPWM0_EVT_CAP0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP1_ST 表示 MCPWM0_EVT_CAP1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP2_ST 表示 MCPWM0_EVT_CAP2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST 表示 MCPWM0_EVT_OP0_TEE1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST 表示 MCPWM0_EVT_OP1_TEE1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST 表示 MCPWM0_EVT_OP2_TEE1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST 表示 MCPWM0_EVT_OP0_TEE2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST 表示 MCPWM0_EVT_OP1_TEE2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST 表示 MCPWM0_EVT_OP2_TEE2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

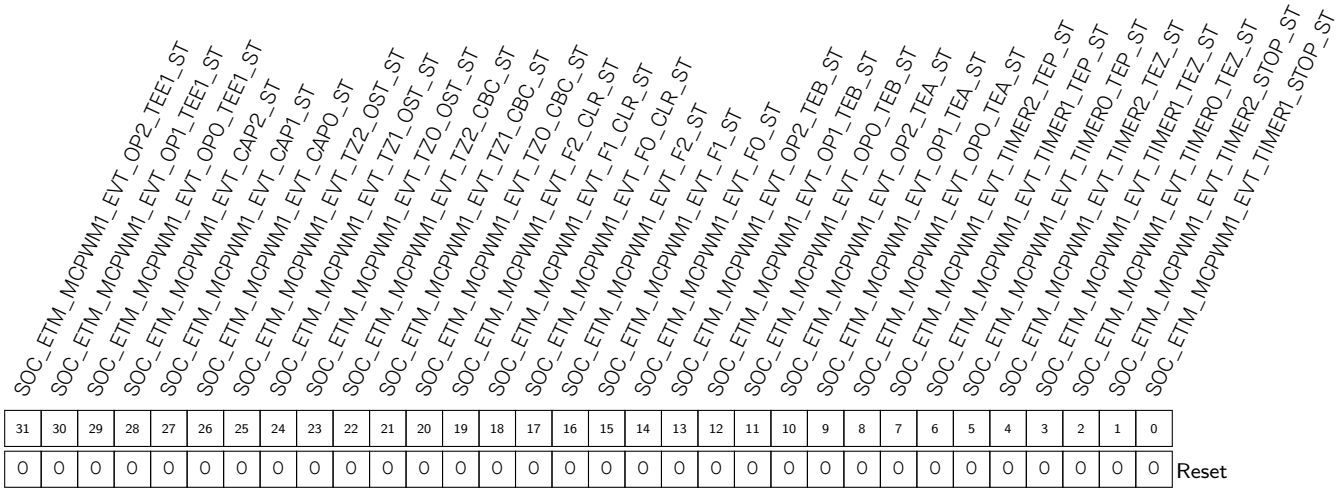
SOC_ETM_MCPWM1_EVT_TIMER0_STOP_ST 表示 MCPWM1_EVT_TIMER0_STOP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)



SOC_ETM_MCPWM1_EVT_TIMER1_STOP_ST 表示 MCPWM1_EVT_TIMER1_STOP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_STOP_ST 表示 MCPWM1_EVT_TIMER2_STOP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER0_TEZ_ST 表示 MCPWM1_EVT_TIMER0_TEZ 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER1_TEZ_ST 表示 MCPWM1_EVT_TIMER1_TEZ 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER2_TEZ_ST 表示 MCPWM1_EVT_TIMER2_TEZ 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER0_TEP_ST 表示 MCPWM1_EVT_TIMER0_TEP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TIMER1_TEP_ST 表示 MCPWM1_EVT_TIMER1_TEP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)

接上页...

SOC_ETM_MCPWM1_EVT_TIMER2_TEP_ST 表示 MCPWM1_EVT_TIMER2_TEP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEA_ST 表示 MCPWM1_EVT_OP0_TEA 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEA_ST 表示 MCPWM1_EVT_OP1_TEA 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEA_ST 表示 MCPWM1_EVT_OP2_TEA 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEB_ST 表示 MCPWM1_EVT_OP0_TEB 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEB_ST 表示 MCPWM1_EVT_OP1_TEB 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEB_ST 表示 MCPWM1_EVT_OP2_TEB 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F0_ST 表示 MCPWM1_EVT_F0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F1_ST 表示 MCPWM1_EVT_F1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)

接上页...

SOC_ETM_MCPWM1_EVT_F2_ST 表示 MCPWM1_EVT_F2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F0_CLR_ST 表示 MCPWM1_EVT_F0_CLR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F1_CLR_ST 表示 MCPWM1_EVT_F1_CLR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_F2_CLR_ST 表示 MCPWM1_EVT_F2_CLR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ0_CBC_ST 表示 MCPWM1_EVT_TZ0_CBC 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ1_CBC_ST 表示 MCPWM1_EVT_TZ1_CBC 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ2_CBC_ST 表示 MCPWM1_EVT_TZ2_CBC 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ0_OST_ST 表示 MCPWM1_EVT_TZ0_OST 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_TZ1_OST_ST 表示 MCPWM1_EVT_TZ1_OST 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.6. SOC_ETM_EVT_ST3_REG (0x01C0)

接上页...

SOC_ETM_MCPWM1_EVT_TZ2_OST_ST 表示 MCPWM1_EVT_TZ2_OST 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP0_ST 表示 MCPWM1_EVT_CAP0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP1_ST 表示 MCPWM1_EVT_CAP1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_CAP2_ST 表示 MCPWM1_EVT_CAP2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP0_TEE1_ST 表示 MCPWM1_EVT_OP0_TEE1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEE1_ST 表示 MCPWM1_EVT_OP1_TEE1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

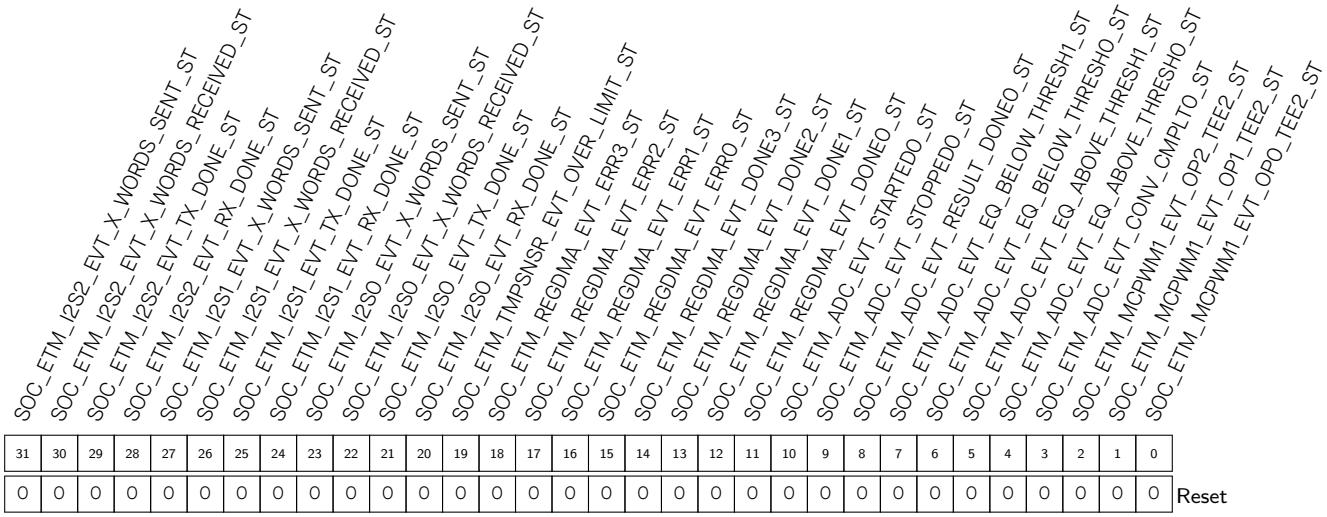
SOC_ETM_MCPWM1_EVT_OP2_TEE1_ST 表示 MCPWM1_EVT_OP2_TEE1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)



SOC_ETM_MCPWM1_EVT_OP0_TEE2_ST 表示 MCPWM1_EVT_OP0_TEE2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP1_TEE2_ST 表示 MCPWM1_EVT_OP1_TEE2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_MCPWM1_EVT_OP2_TEE2_ST 表示 MCPWM1_EVT_OP2_TEE2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ADC_EVT_CONV_CMPLT0_ST 表示 ADC_EVT_CONV_CMPLT0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST 表示 ADC_EVT_EQ_ABOVE_THRESH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST 表示 ADC_EVT_EQ_ABOVE_THRESH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST 表示 ADC_EVT_EQ_BELOW_THRESH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)

接上页...

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST 表示 ADC_EVT_EQ_BELOW_THRESH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST 表示 ADC_EVT_RESULT_DONE0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ADC_EVT_STOPPED0_ST 表示 ADC_EVT_STOPPED0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ADC_EVT_STARTED0_ST 表示 ADC_EVT_STARTED0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE0_ST 表示 REGDMA_EVT_DONE0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE1_ST 表示 REGDMA_EVT_DONE1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE2_ST 表示 REGDMA_EVT_DONE2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_REGDMA_EVT_DONE3_ST 表示 REGDMA_EVT_DONE3 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR0_ST 表示 REGDMA_EVT_ERR0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)

接上页...

SOC_ETM_REGDMA_EVT_ERR1_ST 表示 REGDMA_EVT_ERR1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR2_ST 表示 REGDMA_EVT_ERR2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_REGDMA_EVT_ERR3_ST 表示 REGDMA_EVT_ERR3 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST 表示 TMPSNSR_EVT_OVER_LIMIT 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S0_EVT_RX_DONE_ST 表示 I2S0_EVT_RX_DONE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S0_EVT_TX_DONE_ST 表示 I2S0_EVT_TX_DONE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST 表示 I2S0_EVT_X_WORDS_RECEIVED 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST 表示 I2S0_EVT_X_WORDS_SENT 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S1_EVT_RX_DONE_ST 表示 I2S1_EVT_RX_DONE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.7. SOC_ETM_EVT_ST4_REG (0x01C8)

接上页...

SOC_ETM_I2S1_EVT_TX_DONE_ST 表示 I2S1_EVT_TX_DONE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S1_EVT_X_WORDS_RECEIVED_ST 表示 I2S1_EVT_X_WORDS_RECEIVED 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S1_EVT_X_WORDS_SENT_ST 表示 I2S1_EVT_X_WORDS_SENT 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S2_EVT_RX_DONE_ST 表示 I2S2_EVT_RX_DONE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S2_EVT_TX_DONE_ST 表示 I2S2_EVT_TX_DONE 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_I2S2_EVT_X_WORDS_RECEIVED_ST 表示 I2S2_EVT_X_WORDS_RECEIVED 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

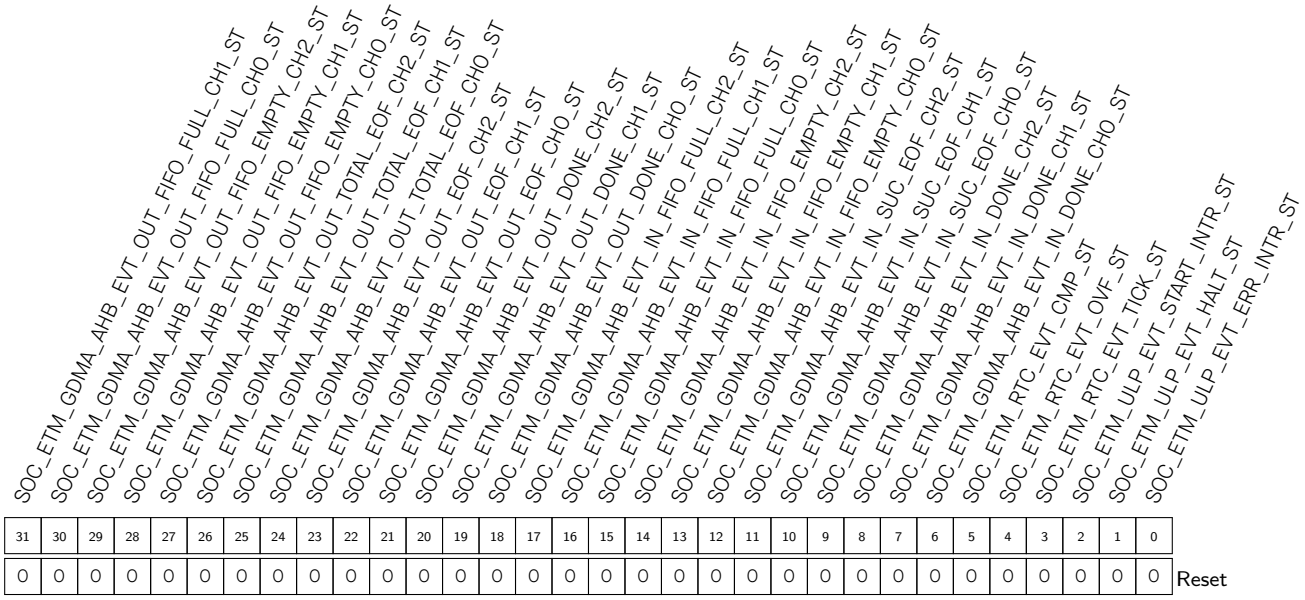
SOC_ETM_I2S2_EVT_X_WORDS_SENT_ST 表示 I2S2_EVT_X_WORDS_SENT 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)



SOC_ETM_ULP_EVT_ERR_INTR_ST 表示 ULP_EVT_ERR_INTR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ULP_EVT_HALT_ST 表示 ULP_EVT_HALT 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_ULP_EVT_START_INTR_ST 表示 ULP_EVT_START_INTR 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_RTC_EVT_TICK_ST 表示 RTC_EVT_TICK 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_RTC_EVT_OVF_ST 表示 RTC_EVT_OVF 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_RTC_EVT_CMP_ST 表示 RTC_EVT_CMP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST 表示 GDMA_AHB_EVT_IN_DONE_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

接上页...

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST 表示 GDMA_AHB_EVT_IN_DONE_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH2_ST 表示 GDMA_AHB_EVT_IN_DONE_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST 表示 GDMA_AHB_EVT_IN_SUC_EOF_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST 表示 GDMA_AHB_EVT_IN_SUC_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH2_ST 表示 GDMA_AHB_EVT_IN_SUC_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST 表示 GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST 表示 GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2_ST 表示 GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

接上页...

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST 表示 GDMA_AHB_EVT_IN_FIFO_FULL_CH0 的状态。
0：未接收到
1：已接收到
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST 表示 GDMA_AHB_EVT_IN_FIFO_FULL_CH1 的状态。
0：未接收到
1：已接收到
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH2_ST 表示 GDMA_AHB_EVT_IN_FIFO_FULL_CH2 的状态。
0：未接收到
1：已接收到
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST 表示 GDMA_AHB_EVT_OUT_DONE_CH0 的状态。
0：未接收到
1：已接收到
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST 表示 GDMA_AHB_EVT_OUT_DONE_CH1 的状态。
0：未接收到
1：已接收到
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH2_ST 表示 GDMA_AHB_EVT_OUT_DONE_CH2 的状态。
0：未接收到
1：已接收到
(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST 表示 GDMA_AHB_EVT_OUT_EOF_CH0 的状态。
0：未接收到
1：已接收到
(R/WTC/SS)

见下页...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

接上页...

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST 表示 GDMA_AHB_EVT_OUT_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH2_ST 表示 GDMA_AHB_EVT_OUT_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST 表示 GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST 表示 GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2_ST 表示 GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST 表示 GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST 表示 GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.8. SOC_ETM_EVT_ST5_REG (0x01D0)

接上页...

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2_ST 表示 GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST 表示 GDMA_AHB_EVT_OUT_FIFO_FULL_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

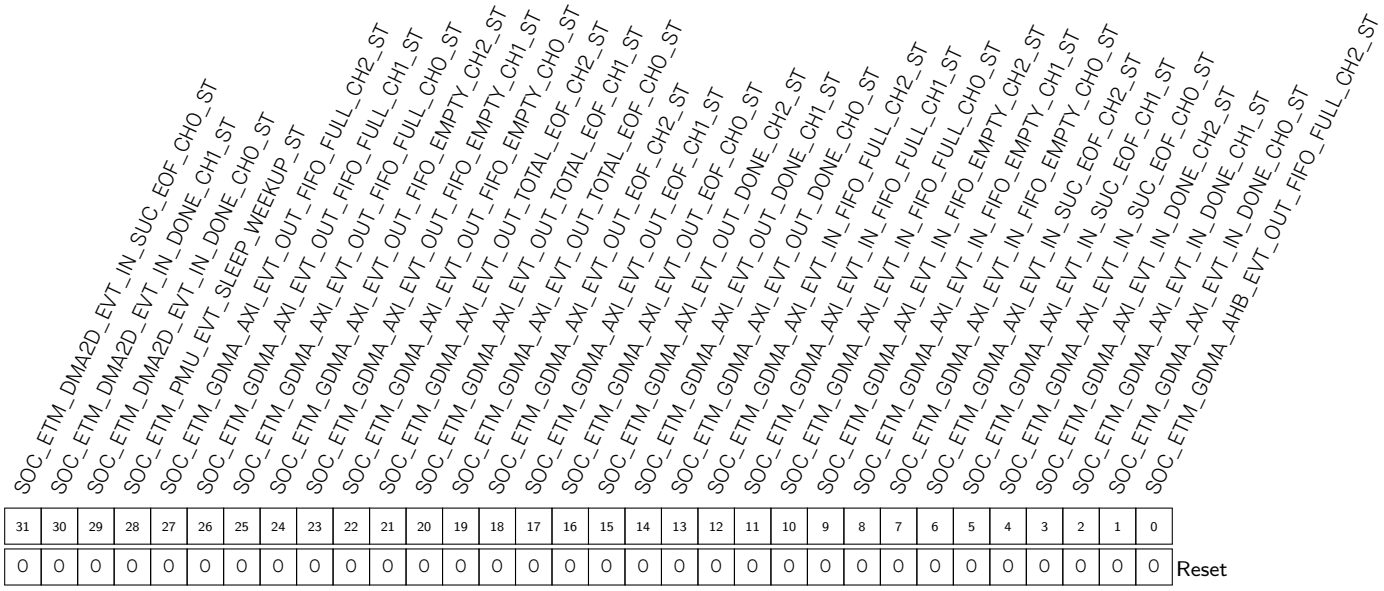
SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST 表示 GDMA_AHB_EVT_OUT_FIFO_FULL_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)



SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH2_ST 表示 GDMA_AHB_EVT_OUT_FIFO_FULL_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH0_ST 表示 GDMA_AXI_EVT_IN_DONE_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH1_ST 表示 GDMA_AXI_EVT_IN_DONE_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH2_ST 表示 GDMA_AXI_EVT_IN_DONE_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH0_ST 表示 GDMA_AXI_EVT_IN_SUC_EOF_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH1_ST 表示 GDMA_AXI_EVT_IN_SUC_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页。

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

接上页...

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH2_ST 表示 GDMA_AXI_EVT_IN_SUC_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0_ST 表示 GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1_ST 表示 GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2_ST 表示 GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH0_ST 表示 GDMA_AXI_EVT_IN_FIFO_FULL_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH1_ST 表示 GDMA_AXI_EVT_IN_FIFO_FULL_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

接上页...

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH2_ST 表示 GDMA_AXI_EVT_IN_FIFO_FULL_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH0_ST 表示 GDMA_AXI_EVT_OUT_DONE_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH1_ST 表示 GDMA_AXI_EVT_OUT_DONE_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH2_ST 表示 GDMA_AXI_EVT_OUT_DONE_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH0_ST 表示 GDMA_AXI_EVT_OUT_EOF_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH1_ST 表示 GDMA_AXI_EVT_OUT_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH2_ST 表示 GDMA_AXI_EVT_OUT_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0_ST 表示 GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

接上页...

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1_ST 表示 GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2_ST 表示 GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0_ST 表示 GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1_ST 表示 GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2_ST 表示 GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH0_ST 表示 GDMA_AXI_EVT_OUT_FIFO_FULL_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

见下页...

Register 14.9. SOC_ETM_EVT_ST6_REG (0x01D8)

接上页...

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH1_ST 表示 GDMA_AXI_EVT_OUT_FIFO_FULL_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH2_ST 表示 GDMA_AXI_EVT_OUT_FIFO_FULL_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST 表示 PMU_EVT_SLEEP_WEEKUP 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_DONE_CH0_ST 表示 DMA2D_EVT_IN_DONE_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_IN_DONE_CH1_ST 表示 DMA2D_EVT_IN_DONE_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

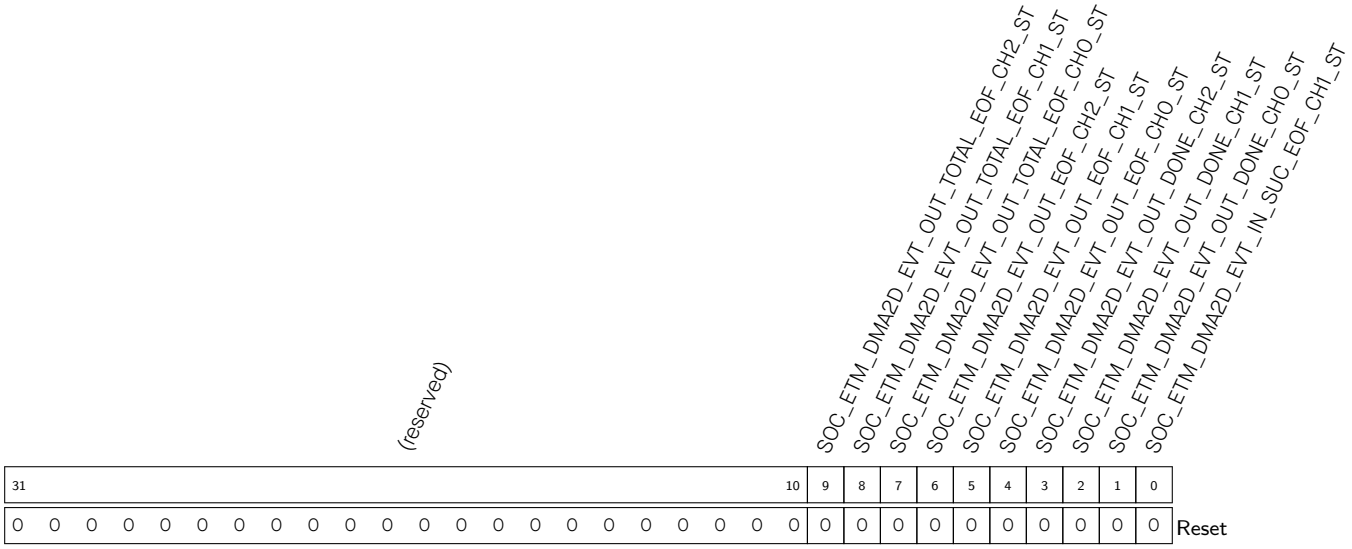
SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH0_ST 表示 DMA2D_EVT_IN_SUC_EOF_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.10. SOC_ETM_EVT_ST7_REG (0x01E0)



SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH1_ST 表示 DMA2D_EVT_IN_SUC_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH0_ST 表示 DMA2D_EVT_OUT_DONE_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH1_ST 表示 DMA2D_EVT_OUT_DONE_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH2_ST 表示 DMA2D_EVT_OUT_DONE_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH0_ST 表示 DMA2D_EVT_OUT_EOF_CH0 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH1_ST 表示 DMA2D_EVT_OUT_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH2_ST 表示 DMA2D_EVT_OUT_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH0_ST 表示 DMA2D_EVT_OUT_TOTAL_EOF_CH0 的状态。

0：未接收到

Register 14.10. SOC_ETM_EVT_ST7_REG (0x01E0)

接上页...

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH1_ST 表示 DMA2D_EVT_OUT_TOTAL_EOF_CH1 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

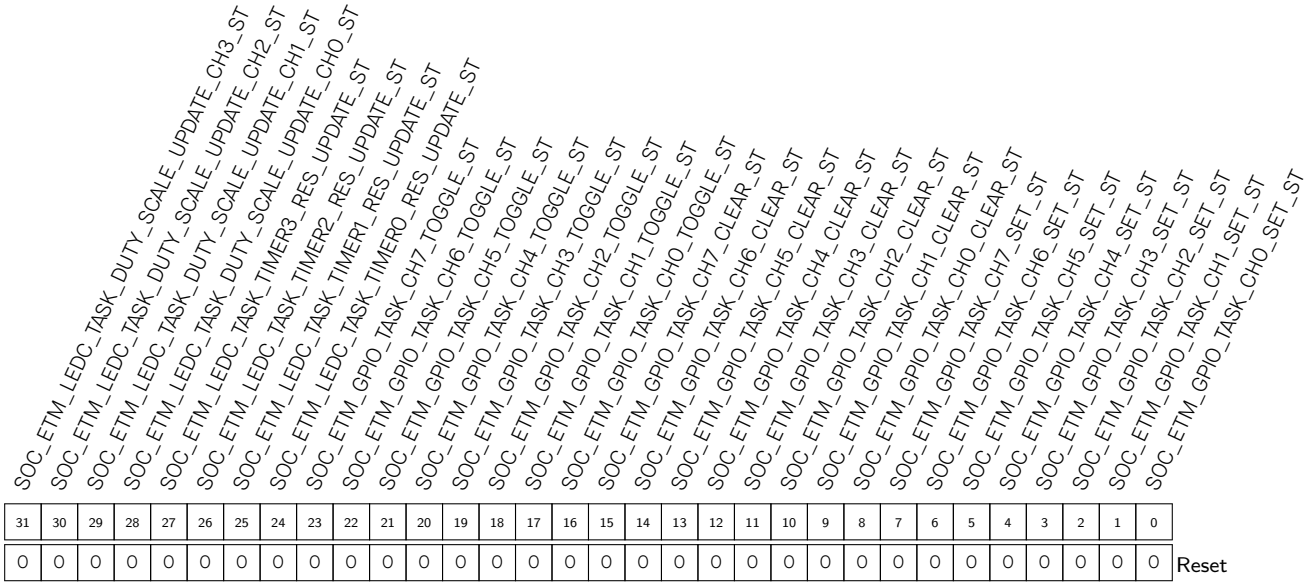
SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH2_ST 表示 DMA2D_EVT_OUT_TOTAL_EOF_CH2 的状态。

0：未接收到

1：已接收到

(R/WTC/SS)

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)



SOC_ETM_GPIO_TASK_CH0_SET_ST 表示 GPIO_TASK_CH0_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_SET_ST 表示 GPIO_TASK_CH1_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_SET_ST 表示 GPIO_TASK_CH2_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_SET_ST 表示 GPIO_TASK_CH3_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_SET_ST 表示 GPIO_TASK_CH4_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_SET_ST 表示 GPIO_TASK_CH5_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_SET_ST 表示 GPIO_TASK_CH6_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)

接上页...

SOC_ETM_GPIO_TASK_CH7_SET_ST 表示 GPIO_TASK_CH7_SET 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST 表示 GPIO_TASK_CH0_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST 表示 GPIO_TASK_CH1_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST 表示 GPIO_TASK_CH2_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST 表示 GPIO_TASK_CH3_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST 表示 GPIO_TASK_CH4_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST 表示 GPIO_TASK_CH5_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST 表示 GPIO_TASK_CH6_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST 表示 GPIO_TASK_CH7_CLEAR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)

接上页...

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST 表示 GPIO_TASK_CH0_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST 表示 GPIO_TASK_CH1_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST 表示 GPIO_TASK_CH2_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST 表示 GPIO_TASK_CH3_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST 表示 GPIO_TASK_CH4_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST 表示 GPIO_TASK_CH5_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST 表示 GPIO_TASK_CH6_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST 表示 GPIO_TASK_CH7_TOGGLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST 表示 LEDC_TASK_TIMER0_RES_UPDATE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.11. SOC_ETM_TASK_ST0_REG (0x01E8)

接上页...

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST 表示 LEDC_TASK_TIMER1_RES_UPDATE 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST 表示 LEDC_TASK_TIMER2_RES_UPDATE 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST 表示 LEDC_TASK_TIMER3_RES_UPDATE 的状态。
0：未产生
1：已产生
(R/WTC/SS)

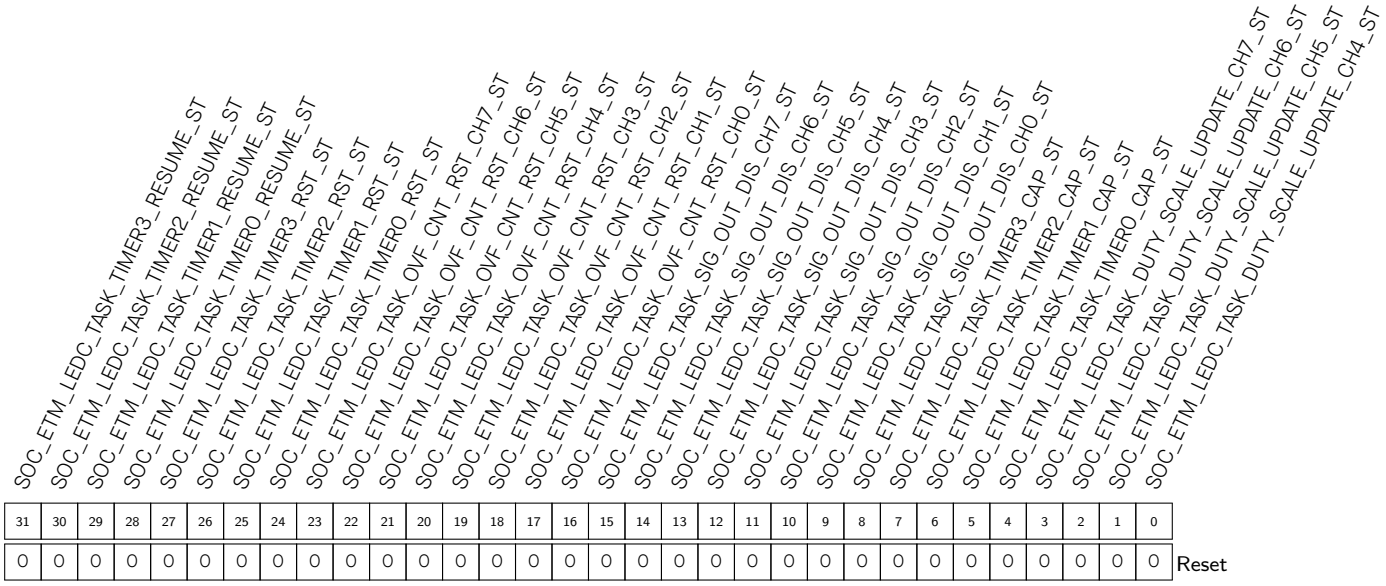
SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH0 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH1 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH2 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH3 的状态。
0：未产生
1：已产生
(R/WTC/SS)

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)



SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH4 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH5 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH6_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH6 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH7_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH7 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST 表示 LEDC_TASK_TIMER0_CAP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST 表示 LEDC_TASK_TIMER1_CAP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

接上页...

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST 表示 LEDC_TASK_TIMER2_CAP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST 表示 LEDC_TASK_TIMER3_CAP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH3 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH4 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH5 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH6_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH6 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

接上页...

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH7_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH7 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST 表示 LEDC_TASK_OVF_CNT_RST_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST 表示 LEDC_TASK_OVF_CNT_RST_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST 表示 LEDC_TASK_OVF_CNT_RST_CH2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST 表示 LEDC_TASK_OVF_CNT_RST_CH3 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST 表示 LEDC_TASK_OVF_CNT_RST_CH4 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST 表示 LEDC_TASK_OVF_CNT_RST_CH5 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

接上页...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH6_ST 表示 LEDC_TASK_OVF_CNT_RST_CH6 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH7_ST 表示 LEDC_TASK_OVF_CNT_RST_CH7 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST 表示 LEDC_TASK_TIMER0_RST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST 表示 LEDC_TASK_TIMER1_RST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RST_ST 表示 LEDC_TASK_TIMER2_RST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST 表示 LEDC_TASK_TIMER3_RST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST 表示 LEDC_TASK_TIMER0_RESUME 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST 表示 LEDC_TASK_TIMER1_RESUME 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.12. SOC_ETM_TASK_ST1_REG (0x01F0)

接上页...

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST 表示 LEDC_TASK_TIMER2_RESUME 的状态。

0：未产生

1：已产生

(R/WTC/SS)

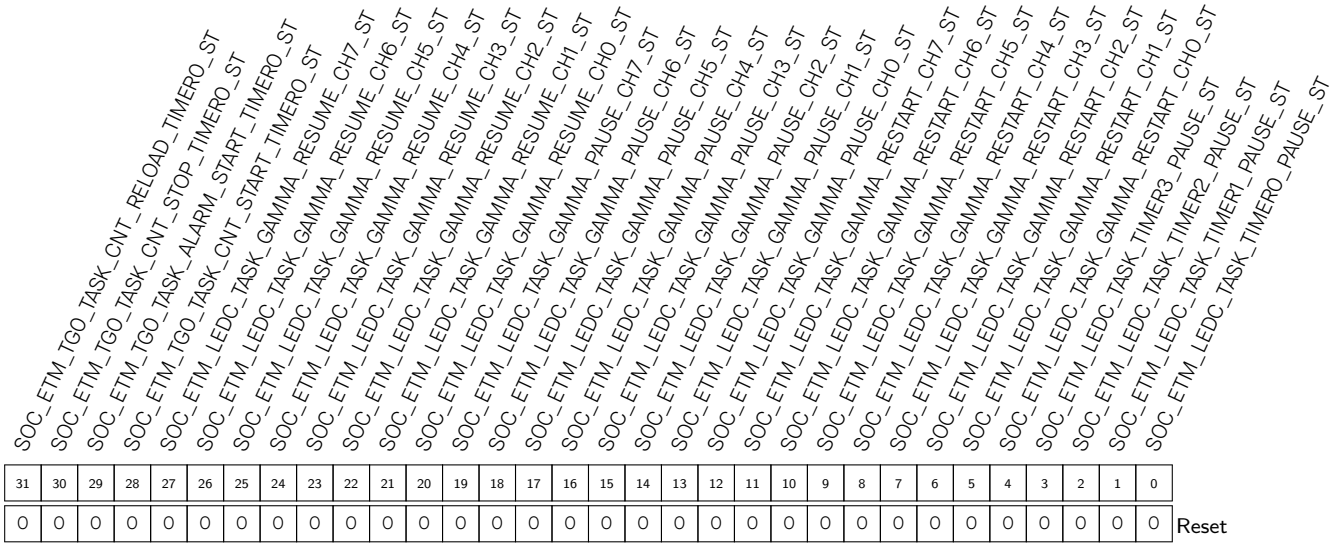
SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST 表示 LEDC_TASK_TIMER3_RESUME 的状态。

0：未产生

1：已产生

(R/WTC/SS)

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)



SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST 表示 LEDC_TASK_TIMER0_PAUSE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST 表示 LEDC_TASK_TIMER1_PAUSE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST 表示 LEDC_TASK_TIMER2_PAUSE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST 表示 LEDC_TASK_TIMER3_PAUSE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST 表示 LEDC_TASK_GAMMA_RESTART_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST 表示 LEDC_TASK_GAMMA_RESTART_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST 表示 LEDC_TASK_GAMMA_RESTART_CH2 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST 表示 LEDC_TASK_GAMMA_RESTART_CH3 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST 表示 LEDC_TASK_GAMMA_RESTART_CH4 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST 表示 LEDC_TASK_GAMMA_RESTART_CH5 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH6_ST 表示 LEDC_TASK_GAMMA_RESTART_CH6 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH7_ST 表示 LEDC_TASK_GAMMA_RESTART_CH7 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH0 的状态。
0：未产生
1：已产生
(R/WTC/SS)

见下页...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH3 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH4 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH5 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH6_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH6 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH7_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH7 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST 表示 LEDC_TASK_GAMMA_RESUME_CH0 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST 表示 LEDC_TASK_GAMMA_RESUME_CH1 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST 表示 LEDC_TASK_GAMMA_RESUME_CH2 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST 表示 LEDC_TASK_GAMMA_RESUME_CH3 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST 表示 LEDC_TASK_GAMMA_RESUME_CH4 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST 表示 LEDC_TASK_GAMMA_RESUME_CH5 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH6_ST 表示 LEDC_TASK_GAMMA_RESUME_CH6 的状态。
0：未产生
1：已产生
(R/WTC/SS)

见下页...

Register 14.13. SOC_ETM_TASK_ST2_REG (0x01F8)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH7_ST 表示 LEDC_TASK_GAMMA_RESUME_CH7 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST 表示 TGO_TASK_CNT_START_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST 表示 TGO_TASK_ALARM_START_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST 表示 TGO_TASK_CNT_STOP_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

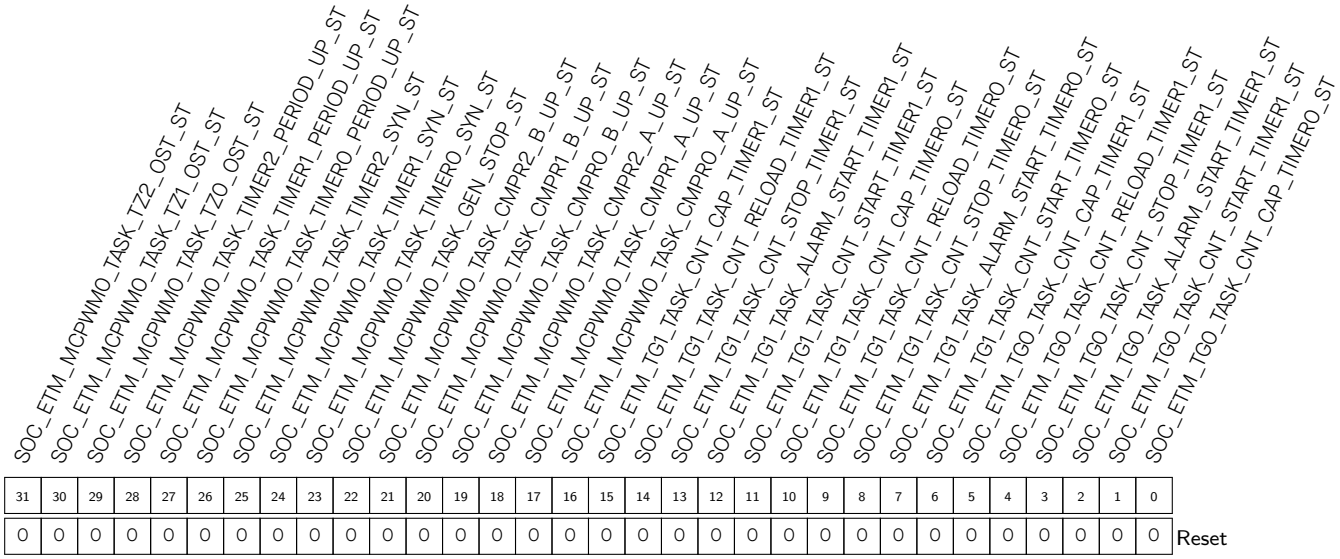
SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST 表示 TGO_TASK_CNT_RELOAD_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)



SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST 表示 TG0_TASK_CNT_CAP_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST 表示 TG0_TASK_CNT_START_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST 表示 TG0_TASK_ALARM_START_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_STOP_TIMER1_ST 表示 TG0_TASK_CNT_STOP_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST 表示 TG0_TASK_CNT_RELOAD_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST 表示 TG0_TASK_CNT_CAP_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

接上页...

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST 表示 TG1_TASK_CNT_START_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST 表示 TG1_TASK_ALARM_START_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST 表示 TG1_TASK_CNT_STOP_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST 表示 TG1_TASK_CNT_RELOAD_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST 表示 TG1_TASK_CNT_CAP_TIMER0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST 表示 TG1_TASK_CNT_START_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST 表示 TG1_TASK_ALARM_START_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST 表示 TG1_TASK_CNT_STOP_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

接上页...

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST 表示 TG1_TASK_CNT_RELOAD_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST 表示 TG1_TASK_CNT_CAP_TIMER1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST 表示 MCPWM0_TASK_CMPR0_A_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST 表示 MCPWM0_TASK_CMPR1_A_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST 表示 MCPWM0_TASK_CMPR2_A_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST 表示 MCPWM0_TASK_CMPR0_B_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST 表示 MCPWM0_TASK_CMPR1_B_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST 表示 MCPWM0_TASK_CMPR2_B_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

接上页...

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST 表示 MCPWM0_TASK_GEN_STOP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST 表示 MCPWM0_TASK_TIMER0_SYN 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST 表示 MCPWM0_TASK_TIMER1_SYN 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST 表示 MCPWM0_TASK_TIMER2_SYN 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST 表示 MCPWM0_TASK_TIMER0_PERIOD_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST 表示 MCPWM0_TASK_TIMER1_PERIOD_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST 表示 MCPWM0_TASK_TIMER2_PERIOD_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST 表示 MCPWM0_TASK_TZ0_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.14. SOC_ETM_TASK_ST3_REG (0x0200)

接上页...

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST 表示 MCPWM0_TASK_TZ1_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

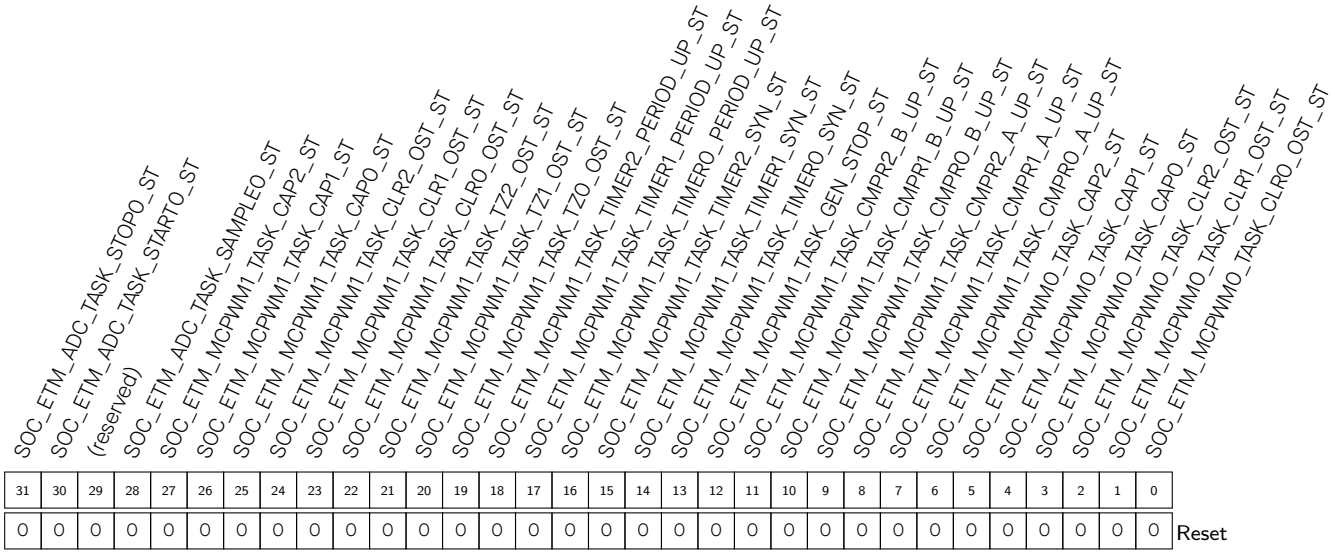
SOC_ETM_MCPWM0_TASK_TZ2_OST_ST 表示 MCPWM0_TASK_TZ2_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)



SOC_ETM_MCPWM0_TASK_CLR0_OST_ST 表示 MCPWM0_TASK_CLR0_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST 表示 MCPWM0_TASK_CLR1_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST 表示 MCPWM0_TASK_CLR2_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP0_ST 表示 MCPWM0_TASK_CAP0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP1_ST 表示 MCPWM0_TASK_CAP1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP2_ST 表示 MCPWM0_TASK_CAP2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR0_A_UP_ST 表示 MCPWM1_TASK_CMPR0_A_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)

接上页...

SOC_ETM_MCPWM1_TASK_CMPR1_A_UP_ST 表示 MCPWM1_TASK_CMPR1_A_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR2_A_UP_ST 表示 MCPWM1_TASK_CMPR2_A_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR0_B_UP_ST 表示 MCPWM1_TASK_CMPR0_B_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR1_B_UP_ST 表示 MCPWM1_TASK_CMPR1_B_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CMPR2_B_UP_ST 表示 MCPWM1_TASK_CMPR2_B_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_GEN_STOP_ST 表示 MCPWM1_TASK_GEN_STOP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER0_SYN_ST 表示 MCPWM1_TASK_TIMER0_SYN 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER1_SYN_ST 表示 MCPWM1_TASK_TIMER1_SYN 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER2_SYN_ST 表示 MCPWM1_TASK_TIMER2_SYN 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)

接上页...

SOC_ETM_MCPWM1_TASK_TIMER0_PERIOD_UP_ST 表示 MCPWM1_TASK_TIMER0_PERIOD_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER1_PERIOD_UP_ST 表示 MCPWM1_TASK_TIMER1_PERIOD_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TIMER2_PERIOD_UP_ST 表示 MCPWM1_TASK_TIMER2_PERIOD_UP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TZ0_OST_ST 表示 MCPWM1_TASK_TZ0_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TZ1_OST_ST 表示 MCPWM1_TASK_TZ1_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_TZ2_OST_ST 表示 MCPWM1_TASK_TZ2_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR0_OST_ST 表示 MCPWM1_TASK_CLR0_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CLR1_OST_ST 表示 MCPWM1_TASK_CLR1_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.15. SOC_ETM_TASK_ST4_REG (0x0208)

接上页...

SOC_ETM_MCPWM1_TASK_CLR2_OST_ST 表示 MCPWM1_TASK_CLR2_OST 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CAP0_ST 表示 MCPWM1_TASK_CAP0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CAP1_ST 表示 MCPWM1_TASK_CAP1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_MCPWM1_TASK_CAP2_ST 表示 MCPWM1_TASK_CAP2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_ADC_TASK_SAMPLE0_ST 表示 ADC_TASK_SAMPLE0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_ADC_TASK_START0_ST 表示 ADC_TASK_START0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

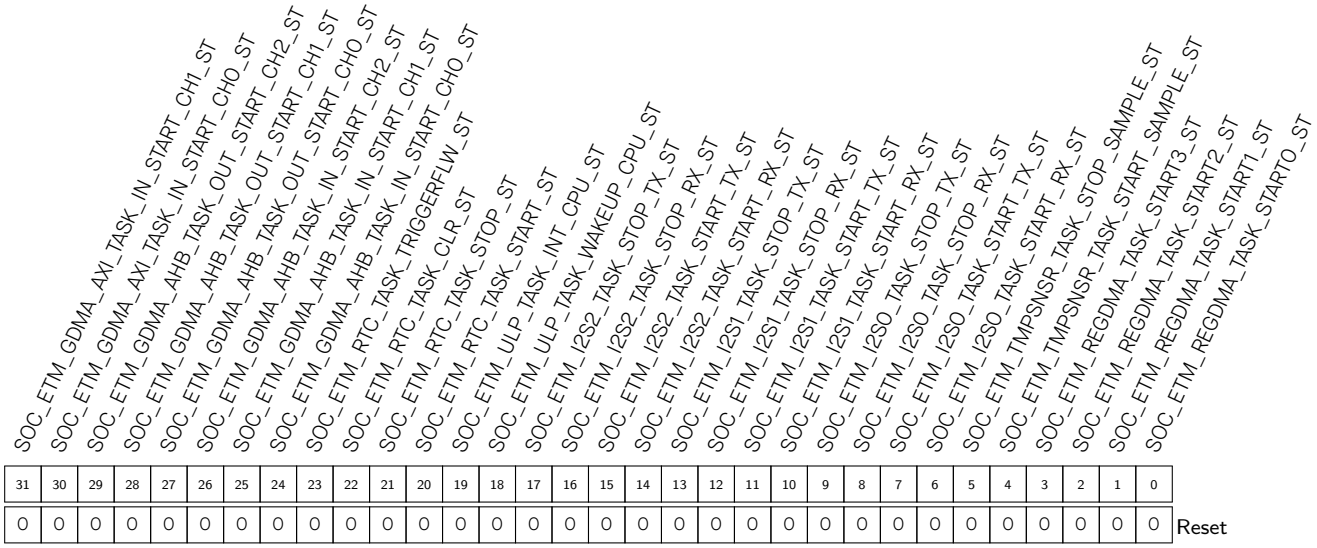
SOC_ETM_ADC_TASK_STOP0_ST 表示 ADC_TASK_STOP0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_REGDMA_TASK_START0_ST 表示 REGDMA_TASK_START0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START1_ST 表示 REGDMA_TASK_START1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START2_ST 表示 REGDMA_TASK_START2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_REGDMA_TASK_START3_ST 表示 REGDMA_TASK_START3 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST 表示 TMPSNSR_TASK_START_SAMPLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST 表示 TMPSNSR_TASK_STOP_SAMPLE 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)

接上页...

SOC_ETM_I2S0_TASK_START_RX_ST 表示 I2S0_TASK_START_RX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S0_TASK_START_TX_ST 表示 I2S0_TASK_START_TX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S0_TASK_STOP_RX_ST 表示 I2S0_TASK_STOP_RX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S0_TASK_STOP_TX_ST 表示 I2S0_TASK_STOP_TX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S1_TASK_START_RX_ST 表示 I2S1_TASK_START_RX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S1_TASK_START_TX_ST 表示 I2S1_TASK_START_TX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S1_TASK_STOP_RX_ST 表示 I2S1_TASK_STOP_RX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S1_TASK_STOP_TX_ST 表示 I2S1_TASK_STOP_TX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S2_TASK_START_RX_ST 表示 I2S2_TASK_START_RX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)

接上页...

SOC_ETM_I2S2_TASK_START_TX_ST 表示 I2S2_TASK_START_TX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S2_TASK_STOP_RX_ST 表示 I2S2_TASK_STOP_RX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_I2S2_TASK_STOP_TX_ST 表示 I2S2_TASK_STOP_TX 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST 表示 ULP_TASK_WAKEUP_CPU 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_ULP_TASK_INT_CPU_ST 表示 ULP_TASK_INT_CPU 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_RTC_TASK_START_ST 表示 RTC_TASK_START 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_RTC_TASK_STOP_ST 表示 RTC_TASK_STOP 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_RTC_TASK_CLR_ST 表示 RTC_TASK_CLR 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST 表示 RTC_TASK_TRIGGERFLW 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.16. SOC_ETM_TASK_ST5_REG (0x0210)

接上页...

SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST 表示 GDMA_AHB_TASK_IN_START_CH0 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST 表示 GDMA_AHB_TASK_IN_START_CH1 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH2_ST 表示 GDMA_AHB_TASK_IN_START_CH2 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST 表示 GDMA_AHB_TASK_OUT_START_CH0 的状态。
0：未产生
1：已产生
(R/WTC/SS)

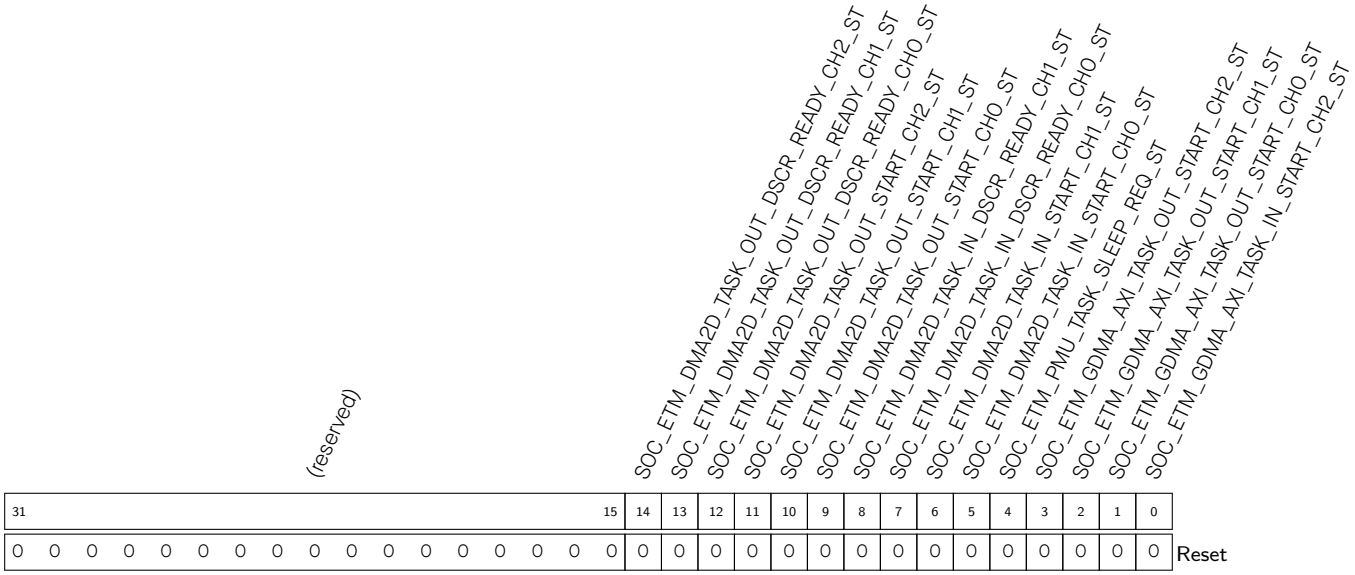
SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST 表示 GDMA_AHB_TASK_OUT_START_CH1 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH2_ST 表示 GDMA_AHB_TASK_OUT_START_CH2 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH0_ST 表示 GDMA_AXI_TASK_IN_START_CH0 的状态。
0：未产生
1：已产生
(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH1_ST 表示 GDMA_AXI_TASK_IN_START_CH1 的状态。
0：未产生
1：已产生
(R/WTC/SS)

Register 14.17. SOC_ETM_TASK_ST6_REG (0x0218)



SOC_ETM_GDMA_AXI_TASK_IN_START_CH2_ST 表示 GDMA_AXI_TASK_IN_START_CH2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH0_ST 表示 GDMA_AXI_TASK_OUT_START_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH1_ST 表示 GDMA_AXI_TASK_OUT_START_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH2_ST 表示 GDMA_AXI_TASK_OUT_START_CH2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST 表示 PMU_TASK_SLEEP_REQ 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_START_CH0_ST 表示 DMA2D_TASK_IN_START_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.17. SOC_ETM_TASK_ST6_REG (0x0218)

接上页...

SOC_ETM_DMA2D_TASK_IN_START_CH1_ST 表示 DMA2D_TASK_IN_START_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH0_ST 表示 DMA2D_TASK_IN_DSCR_READY_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH1_ST 表示 DMA2D_TASK_IN_DSCR_READY_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH0_ST 表示 DMA2D_TASK_OUT_START_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH1_ST 表示 DMA2D_TASK_OUT_START_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_START_CH2_ST 表示 DMA2D_TASK_OUT_START_CH2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH0_ST 表示 DMA2D_TASK_OUT_DSCR_READY_CH0 的状态。

0：未产生

1：已产生

(R/WTC/SS)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH1_ST 表示 DMA2D_TASK_OUT_DSCR_READY_CH1 的状态。

0：未产生

1：已产生

(R/WTC/SS)

见下页...

Register 14.17. SOC_ETM_TASK_ST6_REG (0x0218)

接上页...

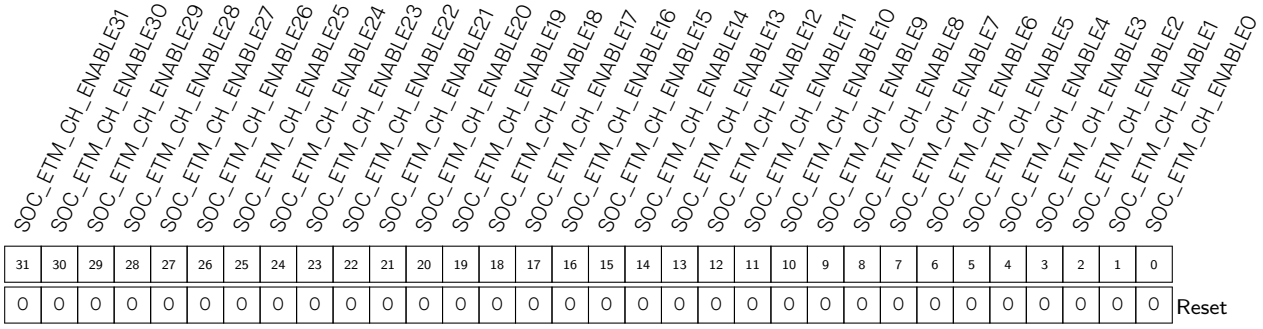
SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH2_ST 表示 DMA2D_TASK_OUT_DSCR_READY_CH2 的状态。

0：未产生

1：已产生

(R/WTC/SS)

Register 14.18. SOC_ETM_CH_ENA_AD0_SET_REG (0x0004)



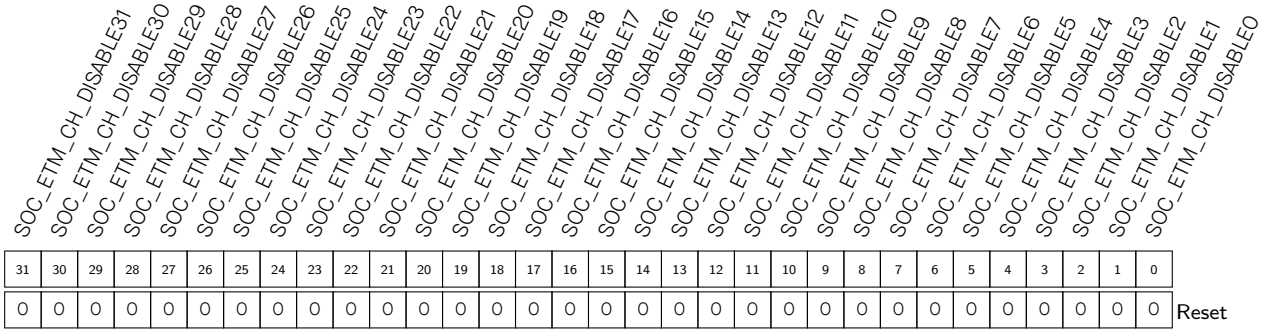
SOC_ETM_CH_ENABLE*n* (*n*: 0-31) 配置使能通道*n* 的状态。

0：无效值，没有作用

1：使能

(WT)

Register 14.19. SOC_ETM_CH_ENA_AD0_CLR_REG (0x0008)



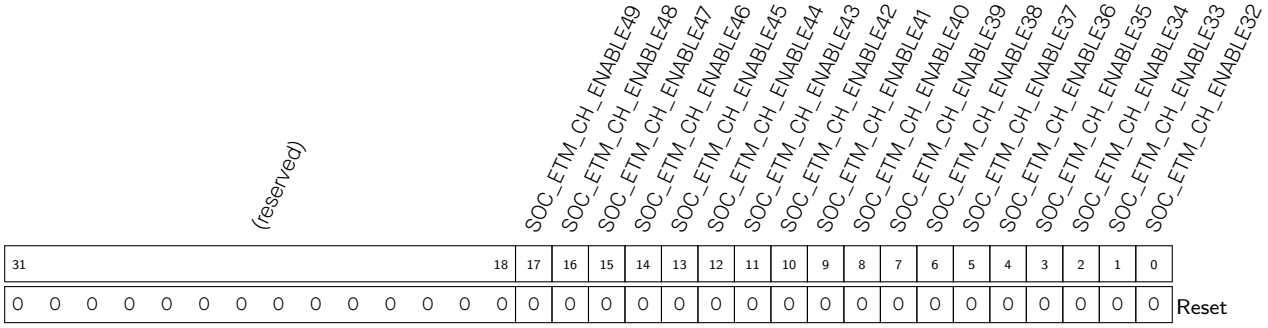
SOC_ETM_CH_DISABLE*n* (*n*: 0-31) 配置关闭通道 0 的状态。

0：无效值，没有作用

1：关闭

(WT)

Register 14.20. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)



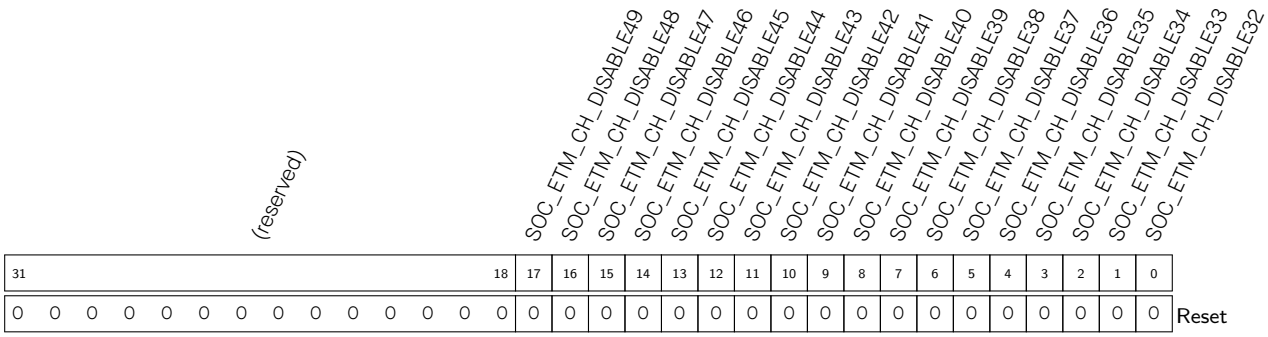
SOC_ETM_CH_ENABLE*n* (*n*: 32-49) 配置使能通道*n* 的状态。

0：无效值，没有作用

1：使能

(WT)

Register 14.21. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)



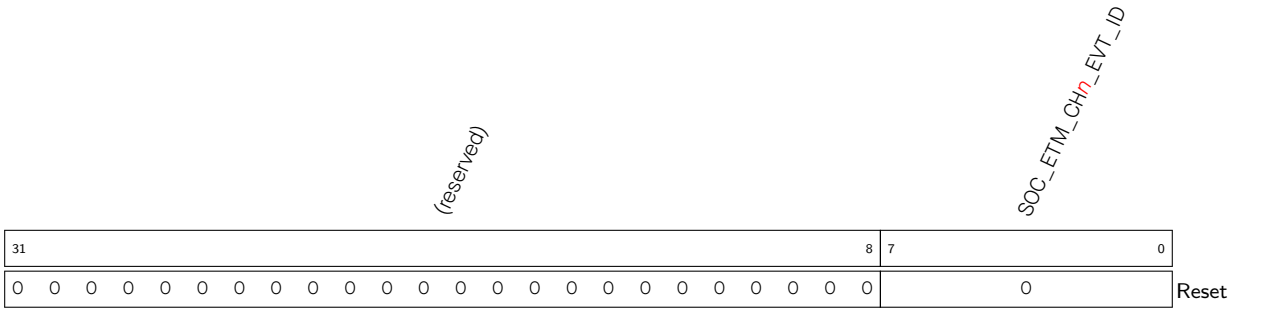
SOC_ETM_CH_DISABLE*n* (*n*: 32-49) 配置关闭通道*n* 的状态。

0：无效值，没有作用

1：关闭

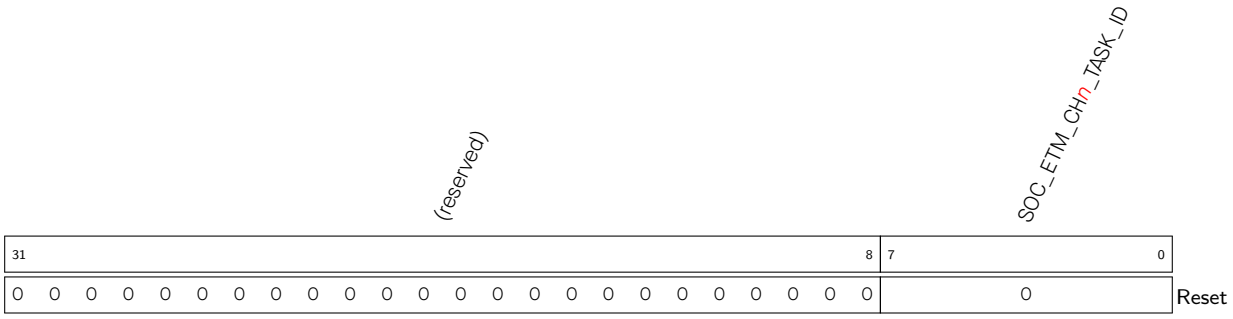
(WT)

Register 14.22. SOC_ETM_CH*n*_EVT_ID_REG (*n*: 0-49) (0x0018+0x8*n)



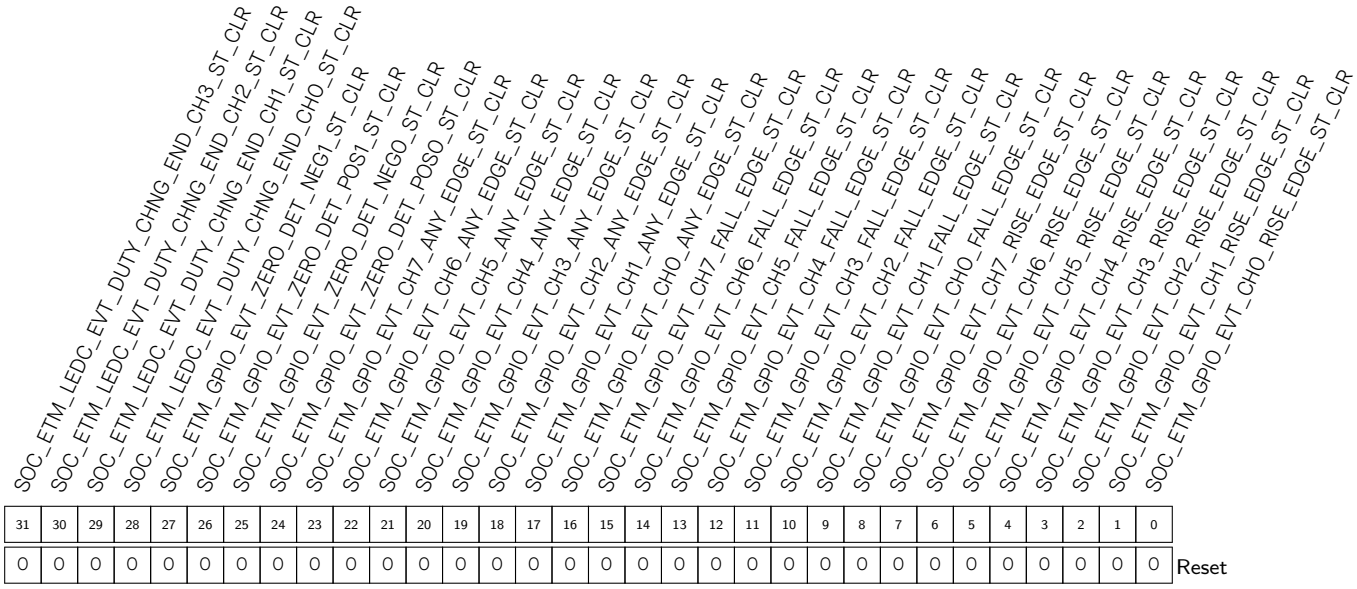
SOC_ETM_CH*n*_EVT_ID (*n*: 0-49) 配置通道*n* 事件 ID，详见表 14.3-1。(R/W)

Register 14.23. SOC_ETM_CH*n*_TASK_ID_REG (*n*: 0-49) (0x001C+0x8**n*)



SOC_ETM_CH*n*_TASK_ID (*n*: 0-49) 配置通道*n* 任务 ID，详见表 14.3-2。(R/W)

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH0_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH1_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH2_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH3_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH4_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH5_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH6_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST_CLR 配置清除 GPIO_EVT_CH7_RISE_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH0_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH1_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH2_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH3_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH4_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH5_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH6_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST_CLR 配置清除 GPIO_EVT_CH7_FALL_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH0_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH1_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH2_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH3_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH4_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH5_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH6_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST_CLR 配置清除 GPIO_EVT_CH7_ANY_EDGE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST_CLR 配置清除 GPIO_EVT_ZERO_DET_POS0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST_CLR 配置清除 GPIO_EVT_ZERO_DET_NEG0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.24. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_ZERO_DET_POS1_ST_CLR 配置清除 GPIO_EVT_ZERO_DET_POS1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG1_ST_CLR 配置清除 GPIO_EVT_ZERO_DET_NEG1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

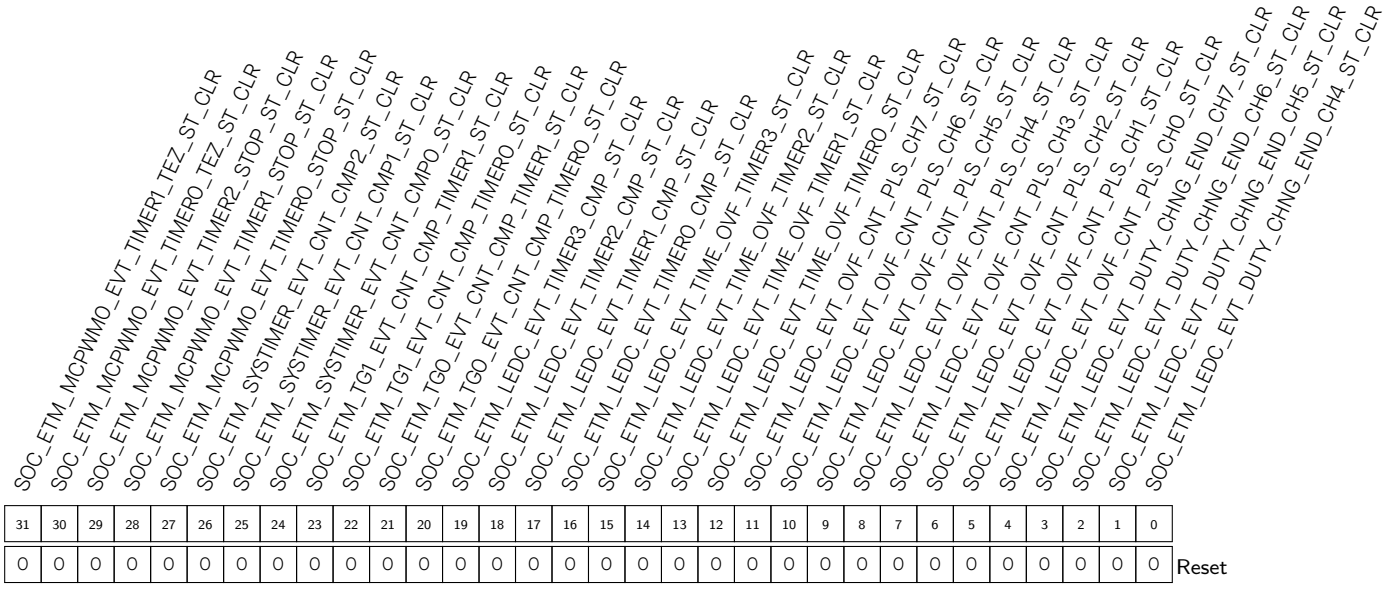
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH3 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)



SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH4 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH5 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH6_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH6 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH7_ST_CLR 配置清除 LEDC_EVT_DUTY_CHNG_END_CH7 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH3 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH4 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH5 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH6_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH6 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH7_ST_CLR 配置清除 LEDC_EVT_OVF_CNT_PLS_CH7 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST_CLR 配置清除 LEDC_EVT_TIME_OVF_TIMER0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST_CLR 配置清除 LEDC_EVT_TIME_OVF_TIMER1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST_CLR 配置清除 LEDC_EVT_TIME_OVF_TIMER2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST_CLR 配置清除 LEDC_EVT_TIME_OVF_TIMER3 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST_CLR 配置清除 LEDC_EVT_TIMER0_CMP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST_CLR 配置清除 LEDC_EVT_TIMER1_CMP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST_CLR 配置清除 LEDC_EVT_TIMER2_CMP 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST_CLR 配置清除 LEDC_EVT_TIMER3_CMP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_EVT_CNT_CMP_TIMER0_ST_CLR 配置清除 TGO_EVT_CNT_CMP_TIMER0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_EVT_CNT_CMP_TIMER1_ST_CLR 配置清除 TGO_EVT_CNT_CMP_TIMER1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST_CLR 配置清除 TG1_EVT_CNT_CMP_TIMER0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST_CLR 配置清除 TG1_EVT_CNT_CMP_TIMER1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST_CLR 配置清除 SYSTIMER_EVT_CNT_CMP0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST_CLR 配置清除 SYSTIMER_EVT_CNT_CMP1 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.25. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST_CLR 配置清除 SYSTIMER_EVT_CNT_CMP2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST_CLR 配置清除 MCPWM0_EVT_TIMER0_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST_CLR 配置清除 MCPWM0_EVT_TIMER1_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST_CLR 配置清除 MCPWM0_EVT_TIMER2_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST_CLR 配置清除 MCPWM0_EVT_TIMER0_TEZ 的状态。

0：无效值，没有作用

1：清除

(WT)

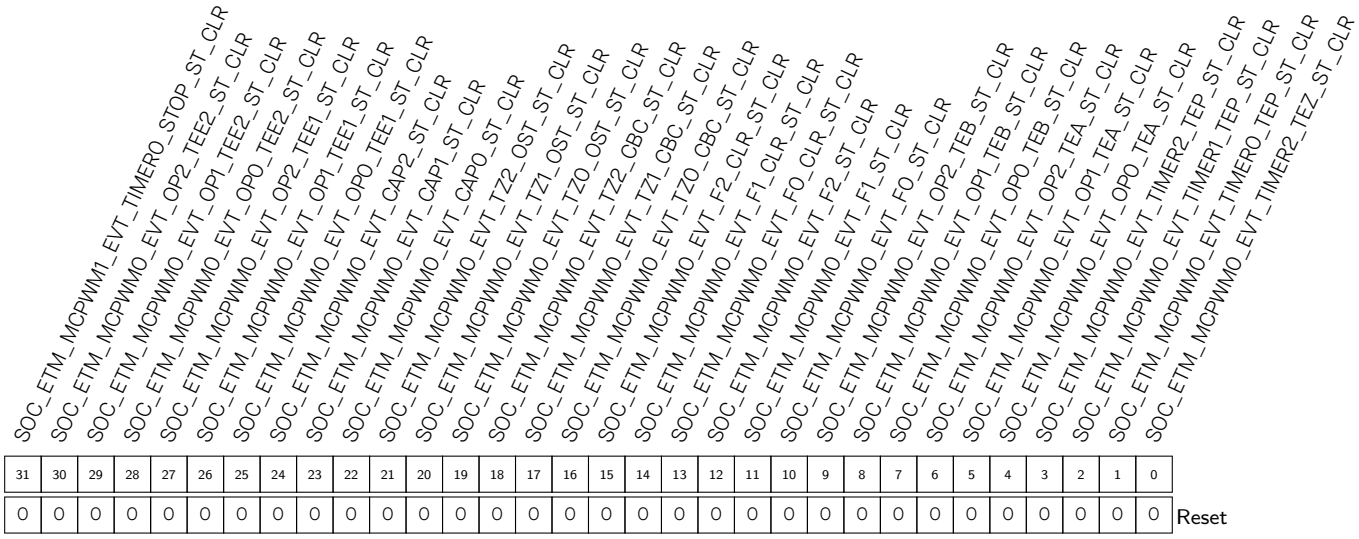
SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST_CLR 配置清除 MCPWM0_EVT_TIMER1_TEZ 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)



SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST_CLR 配置清除 MCPWM0_EVT_TIMER2_TEZ 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST_CLR 配置清除 MCPWM0_EVT_TIMER0_TEP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST_CLR 配置清除 MCPWM0_EVT_TIMER1_TEP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST_CLR 配置清除 MCPWM0_EVT_TIMER2_TEP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST_CLR 配置清除 MCPWM0_EVT_OP0_TEA 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEA_ST_CLR 配置清除 MCPWM0_EVT_OP1_TEA 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWM0_EVT_OP2_TEA_ST_CLR 配置清除 MCPWM0_EVT_OP2_TEA 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST_CLR 配置清除 MCPWM0_EVT_OP0_TEB 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST_CLR 配置清除 MCPWM0_EVT_OP1_TEB 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST_CLR 配置清除 MCPWM0_EVT_OP2_TEB 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F0_ST_CLR 配置清除 MCPWM0_EVT_F0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F1_ST_CLR 配置清除 MCPWM0_EVT_F1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F2_ST_CLR 配置清除 MCPWM0_EVT_F2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F0_CLR_ST_CLR 配置清除 MCPWM0_EVT_F0_CLR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F1_CLR_ST_CLR 配置清除 MCPWM0_EVT_F1_CLR 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWM0_EVT_F2_CLR_ST_CLR 配置清除 MCPWM0_EVT_F2_CLR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST_CLR 配置清除 MCPWM0_EVT_TZ0_CBC 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST_CLR 配置清除 MCPWM0_EVT_TZ1_CBC 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST_CLR 配置清除 MCPWM0_EVT_TZ2_CBC 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST_CLR 配置清除 MCPWM0_EVT_TZ0_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST_CLR 配置清除 MCPWM0_EVT_TZ1_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST_CLR 配置清除 MCPWM0_EVT_TZ2_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_CAP0_ST_CLR 配置清除 MCPWM0_EVT_CAP0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_CAP1_ST_CLR 配置清除 MCPWM0_EVT_CAP1 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.26. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWM0_EVT_CAP2_ST_CLR 配置清除 MCPWM0_EVT_CAP2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST_CLR 配置清除 MCPWM0_EVT_OP0_TEE1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST_CLR 配置清除 MCPWM0_EVT_OP1_TEE1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST_CLR 配置清除 MCPWM0_EVT_OP2_TEE1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST_CLR 配置清除 MCPWM0_EVT_OP0_TEE2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST_CLR 配置清除 MCPWM0_EVT_OP1_TEE2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST_CLR 配置清除 MCPWM0_EVT_OP2_TEE2 的状态。

0：无效值，没有作用

1：清除

(WT)

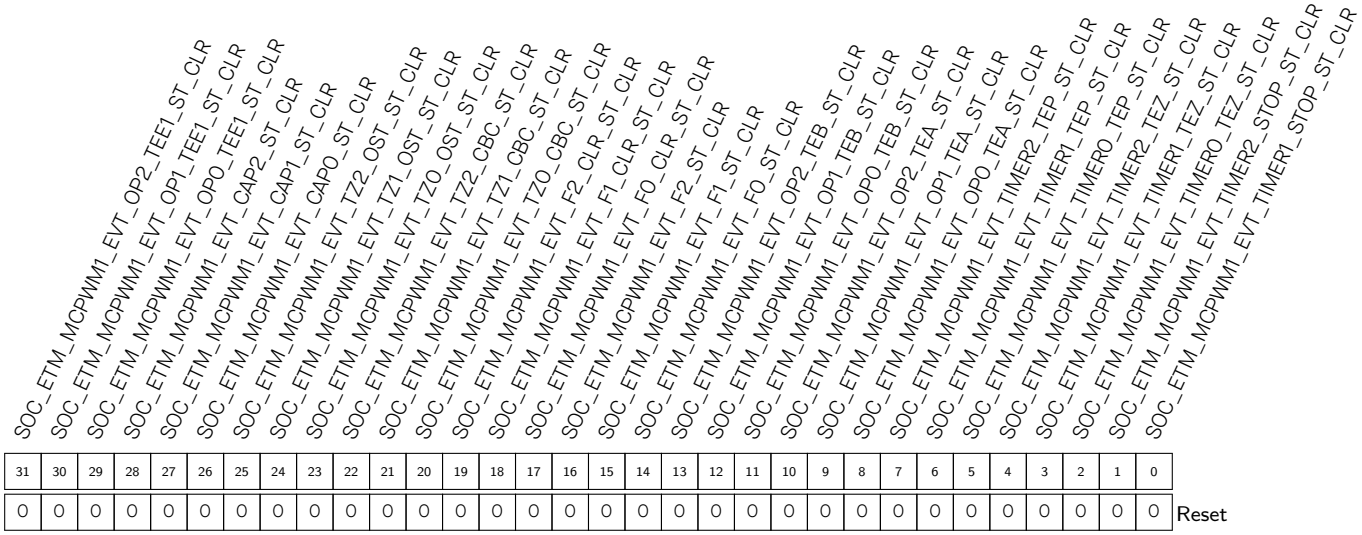
SOC_ETM_MCPWM1_EVT_TIMER0_STOP_ST_CLR 配置清除 MCPWM1_EVT_TIMER0_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)



SOC_ETM_MCPWM1_EVT_TIMER1_STOP_ST_CLR 配置清除 MCPWM1_EVT_TIMER1_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_STOP_ST_CLR 配置清除 MCPWM1_EVT_TIMER2_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TIMER0_TEZ_ST_CLR 配置清除 MCPWM1_EVT_TIMER0_TEZ 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TIMER1_TEZ_ST_CLR 配置清除 MCPWM1_EVT_TIMER1_TEZ 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_TEZ_ST_CLR 配置清除 MCPWM1_EVT_TIMER2_TEZ 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TIMER0_TEP_ST_CLR 配置清除 MCPWM1_EVT_TIMER0_TEP 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_MCPWM1_EVT_TIMER1_TEP_ST_CLR 配置清除 MCPWM1_EVT_TIMER1_TEP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TIMER2_TEP_ST_CLR 配置清除 MCPWM1_EVT_TIMER2_TEP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEA_ST_CLR 配置清除 MCPWM1_EVT_OP0_TEA 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEA_ST_CLR 配置清除 MCPWM1_EVT_OP1_TEA 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEA_ST_CLR 配置清除 MCPWM1_EVT_OP2_TEA 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEB_ST_CLR 配置清除 MCPWM1_EVT_OP0_TEB 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEB_ST_CLR 配置清除 MCPWM1_EVT_OP1_TEB 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEB_ST_CLR 配置清除 MCPWM1_EVT_OP2_TEB 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_MCPWM1_EVT_F0_ST_CLR 配置清除 MCPWM1_EVT_F0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_F1_ST_CLR 配置清除 MCPWM1_EVT_F1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_F2_ST_CLR 配置清除 MCPWM1_EVT_F2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_F0_CLR_ST_CLR 配置清除 MCPWM1_EVT_F0_CLR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_F1_CLR_ST_CLR 配置清除 MCPWM1_EVT_F1_CLR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_F2_CLR_ST_CLR 配置清除 MCPWM1_EVT_F2_CLR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TZ0_CBC_ST_CLR 配置清除 MCPWM1_EVT_TZ0_CBC 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TZ1_CBC_ST_CLR 配置清除 MCPWM1_EVT_TZ1_CBC 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TZ2_CBC_ST_CLR 配置清除 MCPWM1_EVT_TZ2_CBC 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.27. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_MCPWM1_EVT_TZ0_OST_ST_CLR 配置清除 MCPWM1_EVT_TZ0_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TZ1_OST_ST_CLR 配置清除 MCPWM1_EVT_TZ1_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_TZ2_OST_ST_CLR 配置清除 MCPWM1_EVT_TZ2_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_CAP0_ST_CLR 配置清除 MCPWM1_EVT_CAP0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_CAP1_ST_CLR 配置清除 MCPWM1_EVT_CAP1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_CAP2_ST_CLR 配置清除 MCPWM1_EVT_CAP2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP0_TEE1_ST_CLR 配置清除 MCPWM1_EVT_OP0_TEE1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEE1_ST_CLR 配置清除 MCPWM1_EVT_OP1_TEE1 的状态。

0：无效值，没有作用

1：清除

(WT)

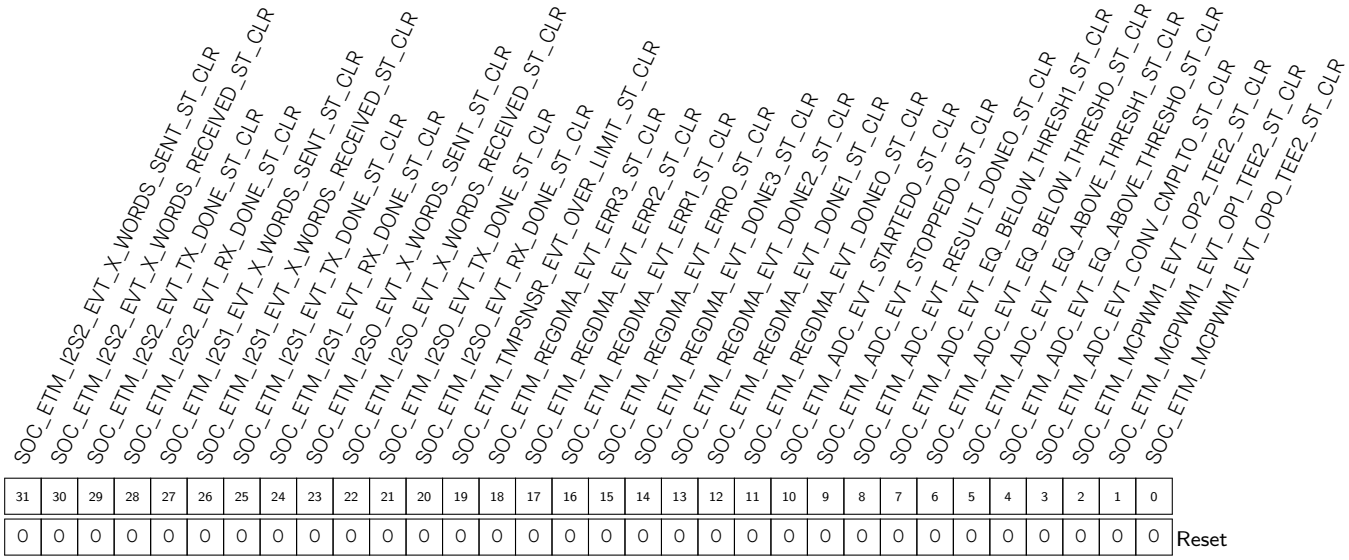
SOC_ETM_MCPWM1_EVT_OP2_TEE1_ST_CLR 配置清除 MCPWM1_EVT_OP2_TEE1 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)



SOC_ETM_MCPWM1_EVT_OP0_TEE2_ST_CLR 配置清除 MCPWM1_EVT_OP0_TEE2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP1_TEE2_ST_CLR 配置清除 MCPWM1_EVT_OP1_TEE2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_EVT_OP2_TEE2_ST_CLR 配置清除 MCPWM1_EVT_OP2_TEE2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_CONV_CMPLTO_ST_CLR 配置清除 ADC_EVT_CONV_CMPLTO 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST_CLR 配置清除 ADC_EVT_EQ_ABOVE_THRESH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST_CLR 配置清除 ADC_EVT_EQ_ABOVE_THRESH1 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

接上页...

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST_CLR 配置清除 ADC_EVT_EQ_BELOW_THRESH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST_CLR 配置清除 ADC_EVT_EQ_BELOW_THRESH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST_CLR 配置清除 ADC_EVT_RESULT_DONE0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_STOPPED0_ST_CLR 配置清除 ADC_EVT_STOPPED0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_STARTED0_ST_CLR 配置清除 ADC_EVT_STARTED0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_EVT_DONE0_ST_CLR 配置清除 REGDMA_EVT_DONE0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_EVT_DONE1_ST_CLR 配置清除 REGDMA_EVT_DONE1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_EVT_DONE2_ST_CLR 配置清除 REGDMA_EVT_DONE2 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

接上页...

SOC_ETM_REGDMA_EVT_DONE3_ST_CLR 配置清除 REGDMA_EVT_DONE3 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_EVT_ERR0_ST_CLR 配置清除 REGDMA_EVT_ERR0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_EVT_ERR1_ST_CLR 配置清除 REGDMA_EVT_ERR1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_EVT_ERR2_ST_CLR 配置清除 REGDMA_EVT_ERR2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_EVT_ERR3_ST_CLR 配置清除 REGDMA_EVT_ERR3 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST_CLR 配置清除 TMPSNSR_EVT_OVER_LIMIT 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_EVT_RX_DONE_ST_CLR 配置清除 I2S0_EVT_RX_DONE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_EVT_TX_DONE_ST_CLR 配置清除 I2S0_EVT_TX_DONE 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

接上页...

SOC_ETM_I2S0_EVT_X_WORDS_RECEIVED_ST_CLR 配置清除 I2S0_EVT_X_WORDS_RECEIVED 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_EVT_X_WORDS_SENT_ST_CLR 配置清除 I2S0_EVT_X_WORDS_SENT 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_EVT_RX_DONE_ST_CLR 配置清除 I2S1_EVT_RX_DONE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_EVT_TX_DONE_ST_CLR 配置清除 I2S1_EVT_TX_DONE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_EVT_X_WORDS_RECEIVED_ST_CLR 配置清除 I2S1_EVT_X_WORDS_RECEIVED 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_EVT_X_WORDS_SENT_ST_CLR 配置清除 I2S1_EVT_X_WORDS_SENT 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S2_EVT_RX_DONE_ST_CLR 配置清除 I2S2_EVT_RX_DONE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S2_EVT_TX_DONE_ST_CLR 配置清除 I2S2_EVT_TX_DONE 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.28. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

接上页...

SOC_ETM_I2S2_EVT_X_WORDS_RECEIVED_ST_CLR 配置清除 I2S2_EVT_X_WORDS_RECEIVED 的状态。

0：无效值，没有作用

1：清除

(WT)

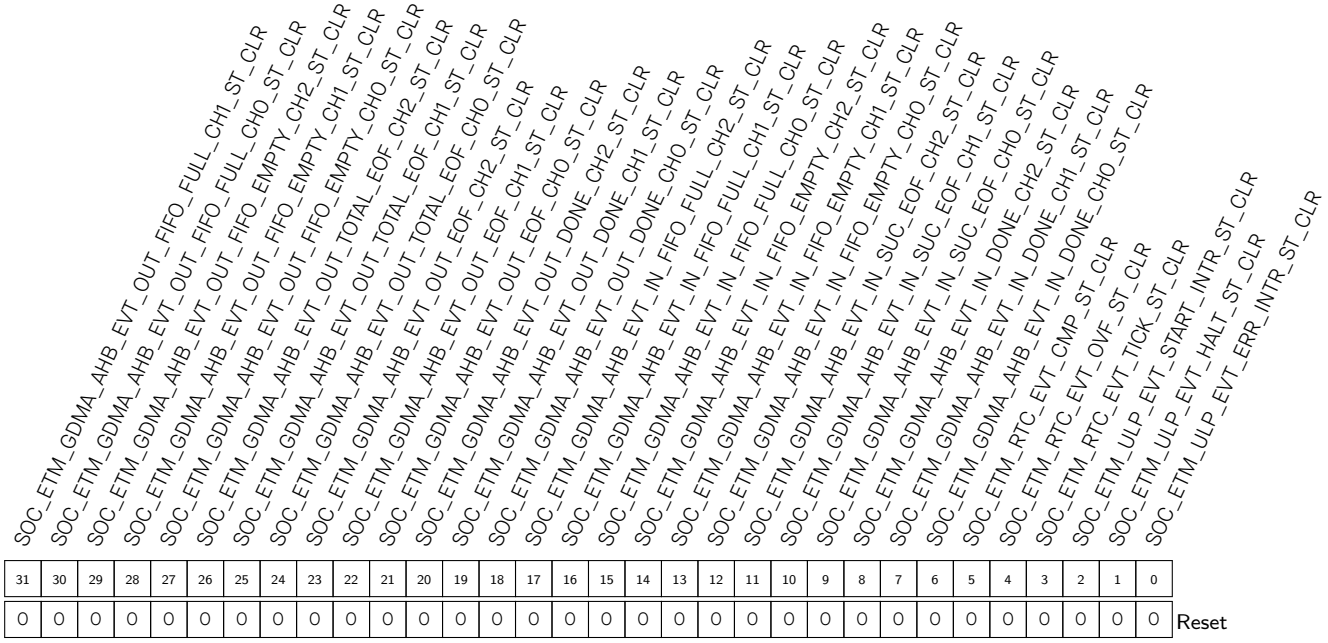
SOC_ETM_I2S2_EVT_X_WORDS_SENT_ST_CLR 配置清除 I2S2_EVT_X_WORDS_SENT 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_ULP_EVT_ERR_INTR_ST_CLR 配置清除 ULP_EVT_ERR_INTR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ULP_EVT_HALT_ST_CLR 配置清除 ULP_EVT_HALT 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ULP_EVT_START_INTR_ST_CLR 配置清除 ULP_EVT_START_INTR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_EVT_TICK_ST_CLR 配置清除 RTC_EVT_TICK 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_EVT_OVF_ST_CLR 配置清除 RTC_EVT_OVF 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_EVT_CMP_ST_CLR 配置清除 RTC_EVT_CMP 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

接上页...

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_IN_DONE_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_IN_DONE_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_DONE_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_IN_DONE_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_IN_SUC_EOF_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_IN_SUC_EOF_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_SUC_EOF_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_IN_SUC_EOF_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_IN_FIFO_EMPTY_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_IN_FIFO_EMPTY_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

接上页...

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_IN_FIFO_EMPTY_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_IN_FIFO_FULL_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_IN_FIFO_FULL_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_IN_FIFO_FULL_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_IN_FIFO_FULL_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_DONE_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_DONE_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_DONE_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_DONE_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_EOF_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

接上页...

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_EOF_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_EOF_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_EOF_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_TOTAL_EOF_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_TOTAL_EOF_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_TOTAL_EOF_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_FIFO_EMPTY_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.29. SOC_ETM_EVT_ST5_CLR_REG (0x01D4)

接上页...

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH0_ST_CLR 配 置 清 除

GDMA_AHB_EVT_OUT_FIFO_FULL_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH1_ST_CLR 配 置 清 除

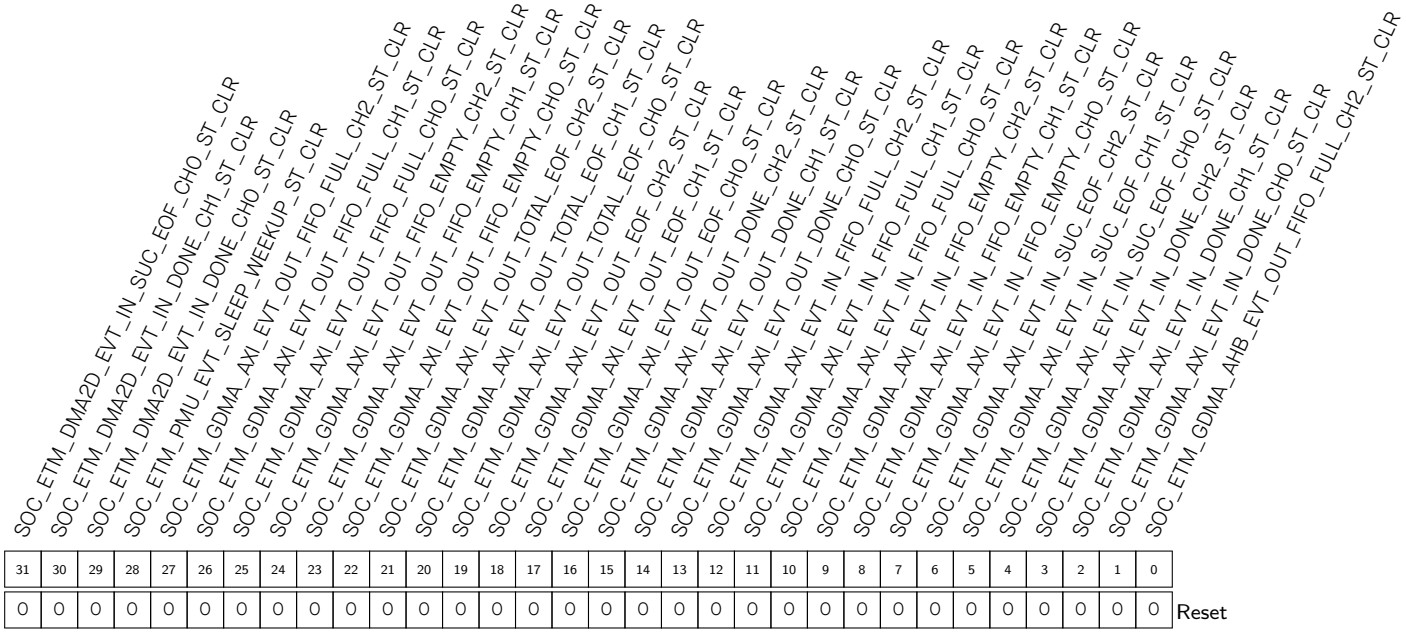
GDMA_AHB_EVT_OUT_FIFO_FULL_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_GDMA_AHB_EVT_OUT_FIFO_FULL_CH2_ST_CLR 配置清除 GDMA_AHB_EVT_OUT_FIFO_FULL_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH0_ST_CLR 配置清除 GDMA_AXI_EVT_IN_DONE_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH1_ST_CLR 配置清除 GDMA_AXI_EVT_IN_DONE_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_DONE_CH2_ST_CLR 配置清除 GDMA_AXI_EVT_IN_DONE_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH0_ST_CLR 配置清除 GDMA_AXI_EVT_IN_SUC_EOF_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

接上页...

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH1_ST_CLR 配置清除 GDMA_AXI_EVT_IN_SUC_EOF_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_SUC_EOF_CH2_ST_CLR 配置清除 GDMA_AXI_EVT_IN_SUC_EOF_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0_ST_CLR 配置清除 GDMA_AXI_EVT_IN_FIFO_EMPTY_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1_ST_CLR 配置清除 GDMA_AXI_EVT_IN_FIFO_EMPTY_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2_ST_CLR 配置清除 GDMA_AXI_EVT_IN_FIFO_EMPTY_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH0_ST_CLR 配置清除 GDMA_AXI_EVT_IN_FIFO_FULL_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH1_ST_CLR 配置清除 GDMA_AXI_EVT_IN_FIFO_FULL_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

接上页...

SOC_ETM_GDMA_AXI_EVT_IN_FIFO_FULL_CH2_ST_CLR 配置清除 GDMA_AXI_EVT_IN_FIFO_FULL_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH0_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_DONE_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH1_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_DONE_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_DONE_CH2_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_DONE_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH0_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_EOF_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH1_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_EOF_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_EOF_CH2_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_EOF_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

接上页...

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0_ST_CLR 配 置 清 除

GDMA_AXI_EVT_OUT_TOTAL_EOF_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1_ST_CLR 配 置 清 除

GDMA_AXI_EVT_OUT_TOTAL_EOF_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2_ST_CLR 配 置 清 除

GDMA_AXI_EVT_OUT_TOTAL_EOF_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR 配 置 清 除

GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR 配 置 清 除

GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR 配 置 清 除

GDMA_AXI_EVT_OUT_FIFO_EMPTY_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH0_ST_CLR 配 置 清 除

GDMA_AXI_EVT_OUT_FIFO_FULL_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.30. SOC_ETM_EVT_ST6_CLR_REG (0x01DC)

接上页...

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH1_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_FIFO_FULL_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_EVT_OUT_FIFO_FULL_CH2_ST_CLR 配置清除 GDMA_AXI_EVT_OUT_FIFO_FULL_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST_CLR 配置清除 PMU_EVT_SLEEP_WEEKUP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_IN_DONE_CH0_ST_CLR 配置清除 DMA2D_EVT_IN_DONE_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_IN_DONE_CH1_ST_CLR 配置清除 DMA2D_EVT_IN_DONE_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

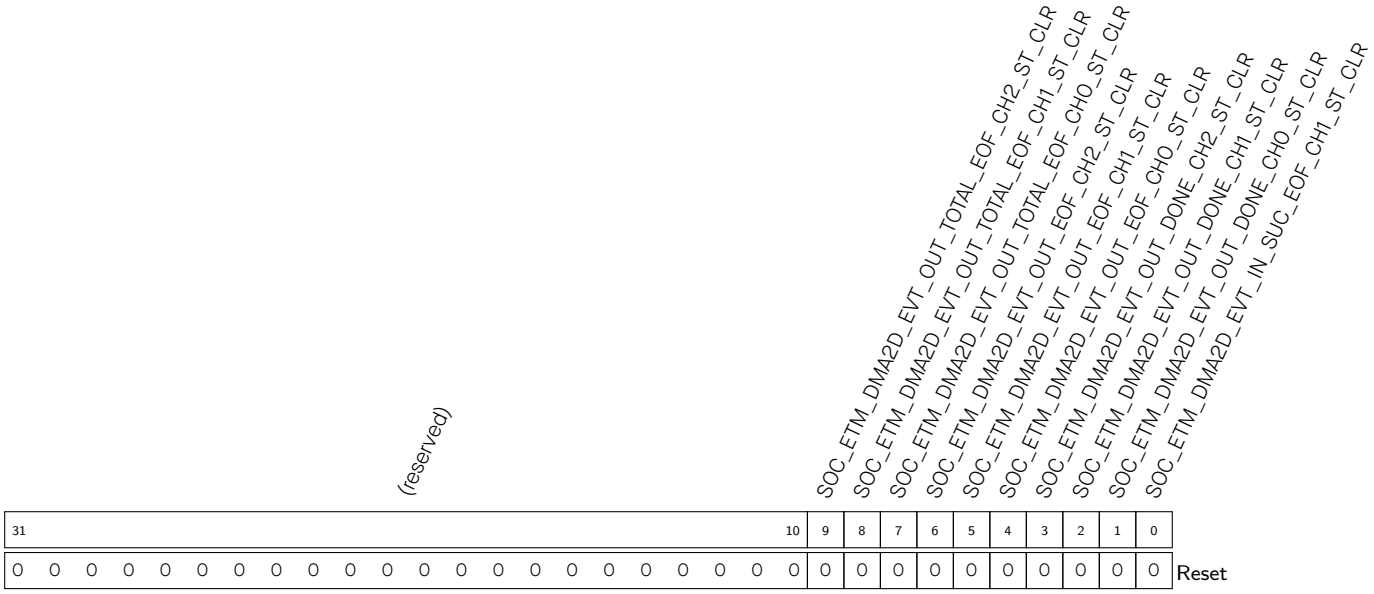
SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH0_ST_CLR 配置清除 DMA2D_EVT_IN_SUC_EOF_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.31. SOC_ETM_EVT_ST7_CLR_REG (0x01E4)



SOC_ETM_DMA2D_EVT_IN_SUC_EOF_CH1_ST_CLR 配置清除 DMA2D_EVT_IN_SUC_EOF_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH0_ST_CLR 配置清除 DMA2D_EVT_OUT_DONE_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH1_ST_CLR 配置清除 DMA2D_EVT_OUT_DONE_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_OUT_DONE_CH2_ST_CLR 配置清除 DMA2D_EVT_OUT_DONE_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH0_ST_CLR 配置清除 DMA2D_EVT_OUT_EOF_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.31. SOC_ETM_EVT_ST7_CLR_REG (0x01E4)

接上页...

SOC_ETM_DMA2D_EVT_OUT_EOF_CH1_ST_CLR 配置清除 DMA2D_EVT_OUT_EOF_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_OUT_EOF_CH2_ST_CLR 配置清除 DMA2D_EVT_OUT_EOF_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH0_ST_CLR 配置清除 DMA2D_EVT_OUT_TOTAL_EOF_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH1_ST_CLR 配置清除 DMA2D_EVT_OUT_TOTAL_EOF_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

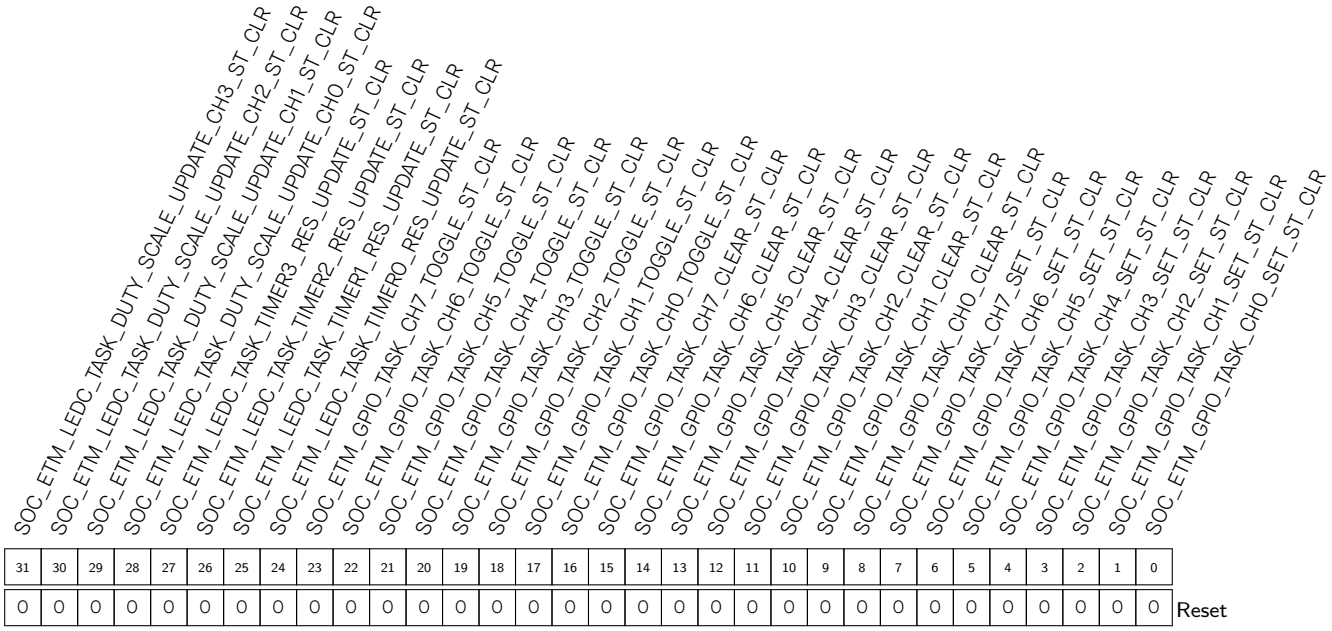
SOC_ETM_DMA2D_EVT_OUT_TOTAL_EOF_CH2_ST_CLR 配置清除 DMA2D_EVT_OUT_TOTAL_EOF_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)



SOC_ETM_GPIO_TASK_CH0_SET_ST_CLR 配置清除 GPIO_TASK_CH0_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH1_SET_ST_CLR 配置清除 GPIO_TASK_CH1_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH2_SET_ST_CLR 配置清除 GPIO_TASK_CH2_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH3_SET_ST_CLR 配置清除 GPIO_TASK_CH3_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH4_SET_ST_CLR 配置清除 GPIO_TASK_CH4_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH5_SET_ST_CLR 配置清除 GPIO_TASK_CH5_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

接上页...

SOC_ETM_GPIO_TASK_CH6_SET_ST_CLR 配置清除 GPIO_TASK_CH6_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH7_SET_ST_CLR 配置清除 GPIO_TASK_CH7_SET 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH0_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH1_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH2_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH3_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH4_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH5_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

接上页...

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH6_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST_CLR 配置清除 GPIO_TASK_CH7_CLEAR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH0_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH1_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH2_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH3_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH4_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH5_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH6_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.32. SOC_ETM_TASK_ST0_CLR_REG (0x01EC)

接上页...

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST_CLR 配置清除 GPIO_TASK_CH7_TOGGLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST_CLR 配置清除 LEDC_TASK_TIMER0_RES_UPDATE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST_CLR 配置清除 LEDC_TASK_TIMER1_RES_UPDATE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST_CLR 配置清除 LEDC_TASK_TIMER2_RES_UPDATE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST_CLR 配置清除 LEDC_TASK_TIMER3_RES_UPDATE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST_CLR 配置清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST_CLR 配置清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST_CLR 配置清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH2 的状态。

0：无效值，没有作用

1：清除

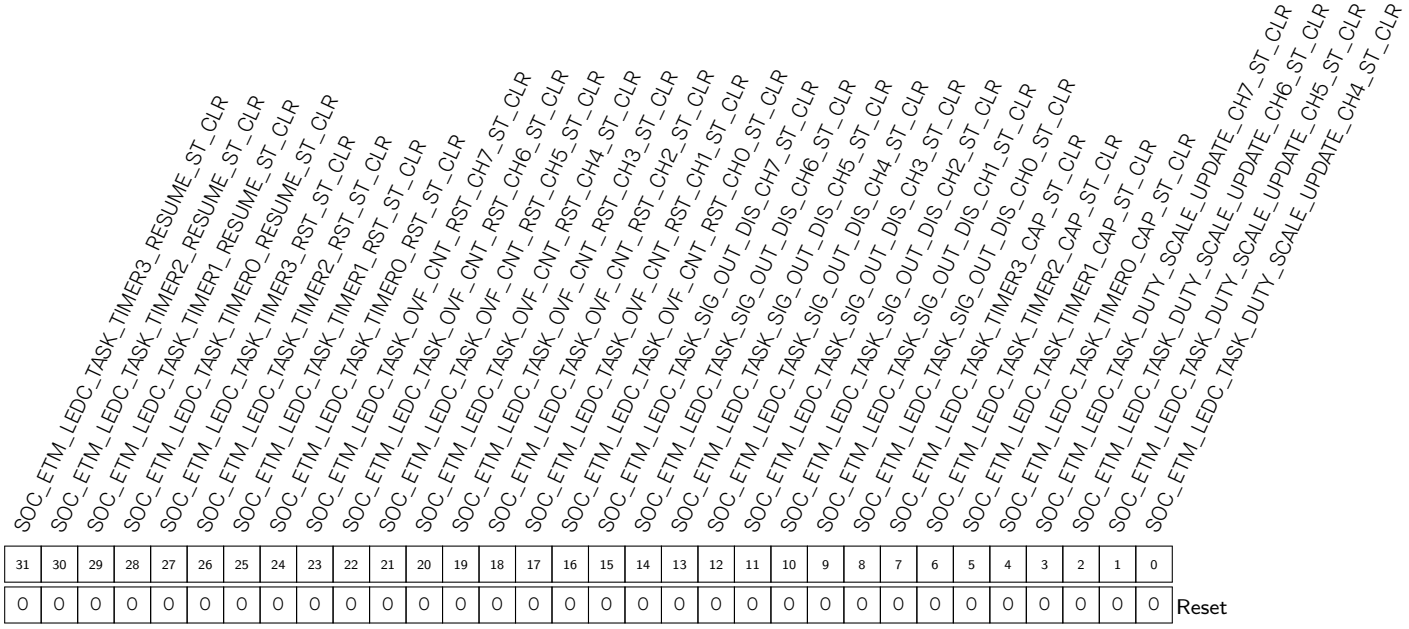
(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST_CLR 配置清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH3 的状态。

0：无效值，没有作用

1：清除

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)



SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST_CLR 配 置 清 除

LEDC_TASK_DUTY_SCALE_UPDATE_CH4 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST_CLR 配 置 清 除

LEDC_TASK_DUTY_SCALE_UPDATE_CH5 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH6_ST_CLR 配 置 清 除

LEDC_TASK_DUTY_SCALE_UPDATE_CH6 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH7_ST_CLR 配 置 清 除

LEDC_TASK_DUTY_SCALE_UPDATE_CH7 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST_CLR 配置清除 LEDC_TASK_TIMER0_CAP 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

接上页...

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST_CLR 配置清除 LEDC_TASK_TIMER1_CAP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST_CLR 配置清除 LEDC_TASK_TIMER2_CAP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST_CLR 配置清除 LEDC_TASK_TIMER3_CAP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH3 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

接上页...

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH4 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH5 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH6_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH6 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH7_ST_CLR 配置清除 LEDC_TASK_SIG_OUT_DIS_CH7 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

接上页...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH3 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH4 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH5 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH6_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH6 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH7_ST_CLR 配置清除 LEDC_TASK_OVF_CNT_RST_CH7 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST_CLR 配置清除 LEDC_TASK_TIMER0_RST 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST_CLR 配置清除 LEDC_TASK_TIMER1_RST 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER2_RST_ST_CLR 配置清除 LEDC_TASK_TIMER2_RST 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.33. SOC_ETM_TASK_ST1_CLR_REG (0x01F4)

接上页...

SOC_ETM_LEDC_TASK_TIMER3_RST_ST_CLR 配置清除 LEDC_TASK_TIMER3_RST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST_CLR 配置清除 LEDC_TASK_TIMER0_RESUME 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST_CLR 配置清除 LEDC_TASK_TIMER1_RESUME 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST_CLR 配置清除 LEDC_TASK_TIMER2_RESUME 的状态。

0：无效值，没有作用

1：清除

(WT)

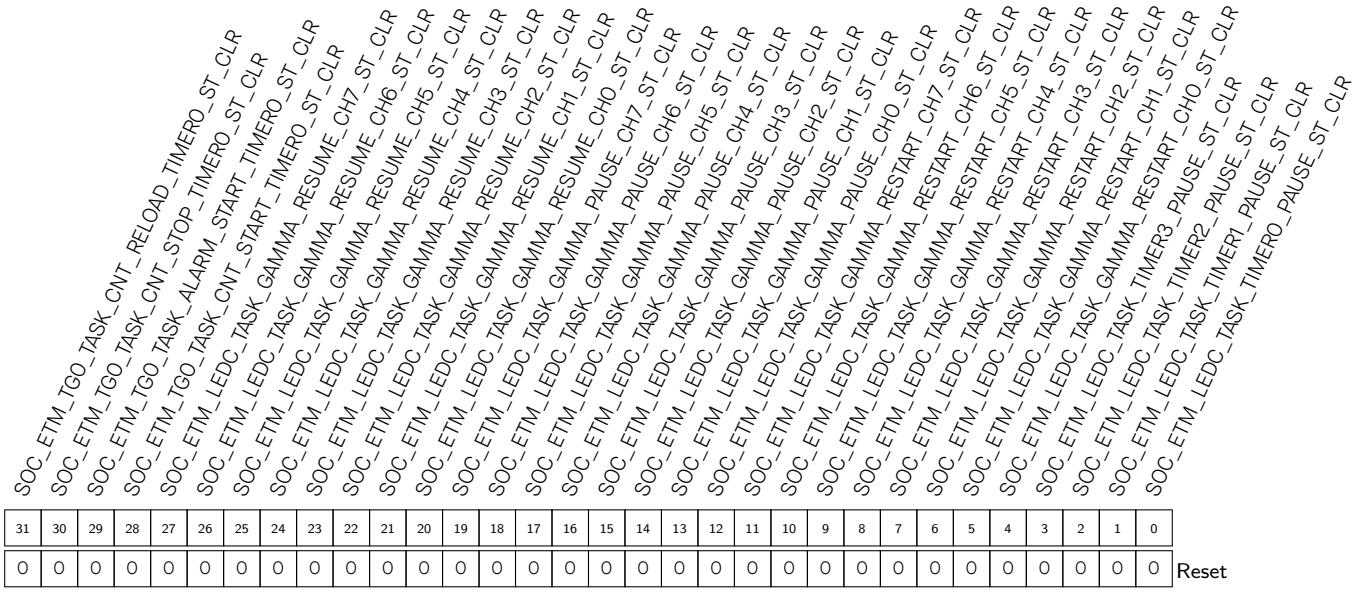
SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST_CLR 配置清除 LEDC_TASK_TIMER3_RESUME 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)



SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST_CLR 配置清除 LEDC_TASK_TIMER0_PAUSE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST_CLR 配置清除 LEDC_TASK_TIMER1_PAUSE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST_CLR 配置清除 LEDC_TASK_TIMER2_PAUSE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST_CLR 配置清除 LEDC_TASK_TIMER3_PAUSE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH3 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH4 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH5 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH6_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH6 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH7_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESTART_CH7 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH3 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH4 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH5 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH6_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH6 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH7_ST_CLR 配置清除 LEDC_TASK_GAMMA_PAUSE_CH7 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH3 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH4 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH5 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH6_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH6 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH7_ST_CLR 配置清除 LEDC_TASK_GAMMA_RESUME_CH7 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST_CLR 配置清除 TGO_TASK_CNT_START_TIMER0 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.34. SOC_ETM_TASK_ST2_CLR_REG (0x01FC)

接上页...

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST_CLR 配置清除 TGO_TASK_ALARM_START_TIMER0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST_CLR 配置清除 TGO_TASK_CNT_STOP_TIMER0 的状态。

0：无效值，没有作用

1：清除

(WT)

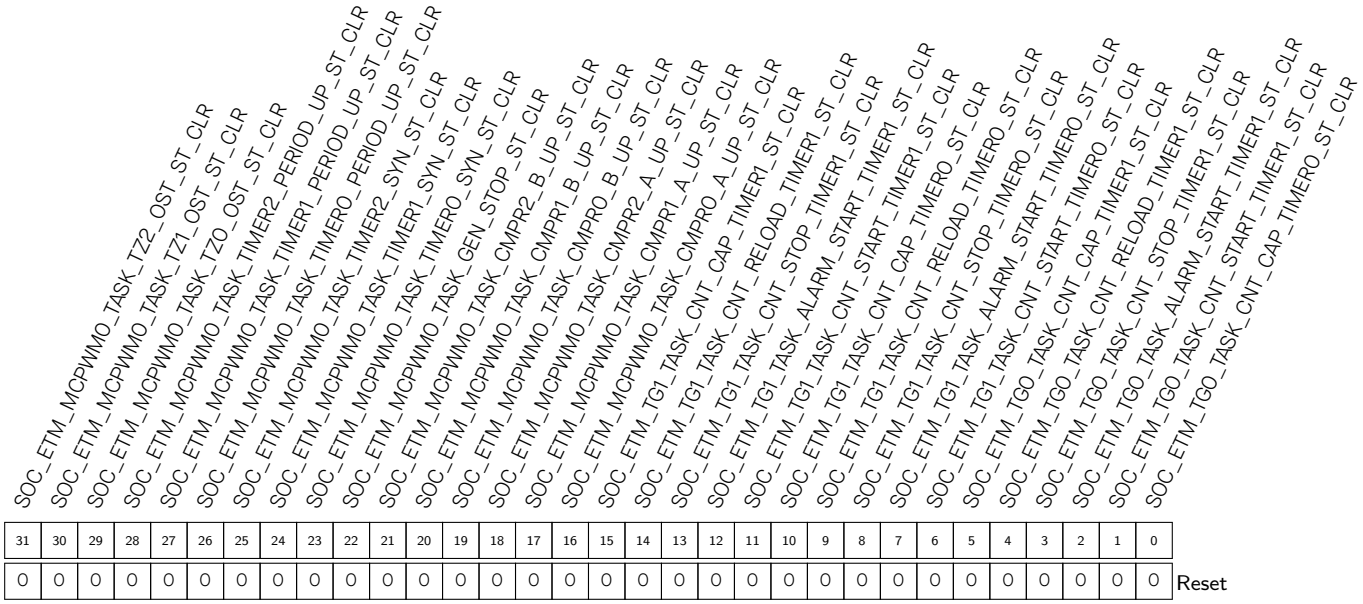
SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST_CLR 配置清除 TGO_TASK_CNT_RELOAD_TIMER0 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_TGO_TASK_CNT_CAP_TIMER0_ST_CLR 配置清除 TGO_TASK_CNT_CAP_TIMER0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_START_TIMER1_ST_CLR 配置清除 TGO_TASK_CNT_START_TIMER1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_ALARM_START_TIMER1_ST_CLR 配置清除 TGO_TASK_ALARM_START_TIMER1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER1_ST_CLR 配置清除 TGO_TASK_CNT_STOP_TIMER1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER1_ST_CLR 配置清除 TGO_TASK_CNT_RELOAD_TIMER1 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

接上页...

SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST_CLR 配置清除 TG0_TASK_CNT_CAP_TIMER1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST_CLR 配置清除 TG1_TASK_CNT_START_TIMER0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST_CLR 配置清除 TG1_TASK_ALARM_START_TIMER0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST_CLR 配置清除 TG1_TASK_CNT_STOP_TIMER0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST_CLR 配置清除 TG1_TASK_CNT_RELOAD_TIMER0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST_CLR 配置清除 TG1_TASK_CNT_CAP_TIMER0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST_CLR 配置清除 TG1_TASK_CNT_START_TIMER1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST_CLR 配置清除 TG1_TASK_ALARM_START_TIMER1 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

接上页...

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST_CLR 配置清除 TG1_TASK_CNT_STOP_TIMER1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST_CLR 配置清除 TG1_TASK_CNT_RELOAD_TIMER1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST_CLR 配置清除 TG1_TASK_CNT_CAP_TIMER1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST_CLR 配置清除 MCPWM0_TASK_CMPR0_A_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST_CLR 配置清除 MCPWM0_TASK_CMPR1_A_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST_CLR 配置清除 MCPWM0_TASK_CMPR2_A_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST_CLR 配置清除 MCPWM0_TASK_CMPR0_B_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST_CLR 配置清除 MCPWM0_TASK_CMPR1_B_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

接上页...

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST_CLR 配置清除 MCPWM0_TASK_CMPR2_B_UP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST_CLR 配置清除 MCPWM0_TASK_GEN_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST_CLR 配置清除 MCPWM0_TASK_TIMER0_SYN 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST_CLR 配置清除 MCPWM0_TASK_TIMER1_SYN 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST_CLR 配置清除 MCPWM0_TASK_TIMER2_SYN 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST_CLR 配置清除 MCPWM0_TASK_TIMER0_PERIOD_UP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST_CLR 配置清除 MCPWM0_TASK_TIMER1_PERIOD_UP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST_CLR 配置清除 MCPWM0_TASK_TIMER2_PERIOD_UP 的状态。

0：无效值，没有作用

1：清除

(WT)

返回页...

Register 14.35. SOC_ETM_TASK_ST3_CLR_REG (0x0204)

接上页...

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST_CLR 配置清除 MCPWM0_TASK_TZ0_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST_CLR 配置清除 MCPWM0_TASK_TZ1_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

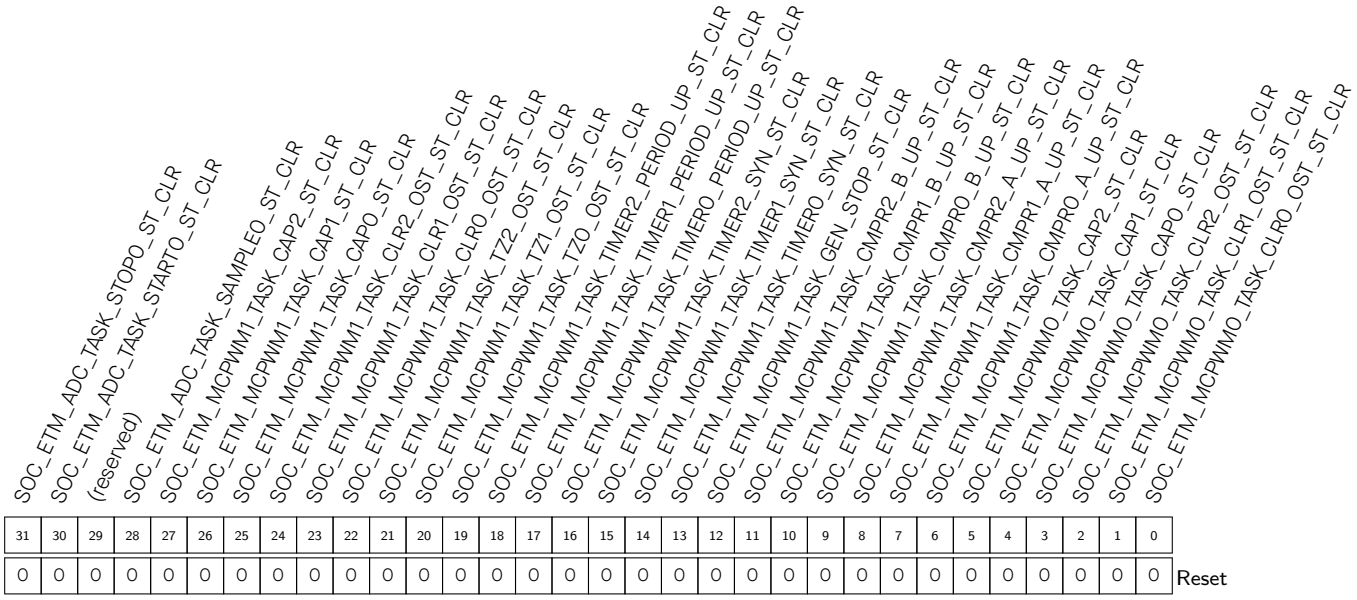
SOC_ETM_MCPWM0_TASK_TZ2_OST_ST_CLR 配置清除 MCPWM0_TASK_TZ2_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)



SOC_ETM_MCPWM0_TASK_CLR0_OST_ST_CLR 配置清除 MCPWM0_TASK_CLR0_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST_CLR 配置清除 MCPWM0_TASK_CLR1_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST_CLR 配置清除 MCPWM0_TASK_CLR2_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CAP0_ST_CLR 配置清除 MCPWM0_TASK_CAP0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CAP1_ST_CLR 配置清除 MCPWM0_TASK_CAP1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CAP2_ST_CLR 配置清除 MCPWM0_TASK_CAP2 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

接上页...

SOC_ETM_MCPWM1_TASK_CMPR0_A_UP_ST_CLR 配置清除 MCPWM1_TASK_CMPR0_A_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM1_TASK_CMPR1_A_UP_ST_CLR 配置清除 MCPWM1_TASK_CMPR1_A_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM1_TASK_CMPR2_A_UP_ST_CLR 配置清除 MCPWM1_TASK_CMPR2_A_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM1_TASK_CMPR0_B_UP_ST_CLR 配置清除 MCPWM1_TASK_CMPR0_B_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM1_TASK_CMPR1_B_UP_ST_CLR 配置清除 MCPWM1_TASK_CMPR1_B_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM1_TASK_CMPR2_B_UP_ST_CLR 配置清除 MCPWM1_TASK_CMPR2_B_UP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM1_TASK_GEN_STOP_ST_CLR 配置清除 MCPWM1_TASK_GEN_STOP 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM1_TASK_TIMER0_SYN_ST_CLR 配置清除 MCPWM1_TASK_TIMER0_SYN 的状态。
0：无效值，没有作用
1：清除
(WT)

接下页...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

接上页...

SOC_ETM_MCPWM1_TASK_TIMER1_SYN_ST_CLR 配置清除 MCPWM1_TASK_TIMER1_SYN 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_TIMER2_SYN_ST_CLR 配置清除 MCPWM1_TASK_TIMER2_SYN 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_TIMER0_PERIOD_UP_ST_CLR 配置清除 MCPWM1_TASK_TIMER0_PERIOD_UP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_TIMER1_PERIOD_UP_ST_CLR 配置清除 MCPWM1_TASK_TIMER1_PERIOD_UP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_TIMER2_PERIOD_UP_ST_CLR 配置清除 MCPWM1_TASK_TIMER2_PERIOD_UP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_TZ0_OST_ST_CLR 配置清除 MCPWM1_TASK_TZ0_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_TZ1_OST_ST_CLR 配置清除 MCPWM1_TASK_TZ1_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_TZ2_OST_ST_CLR 配置清除 MCPWM1_TASK_TZ2_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

接上页...

SOC_ETM_MCPWM1_TASK_CLR0_OST_ST_CLR 配置清除 MCPWM1_TASK_CLR0_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_CLR1_OST_ST_CLR 配置清除 MCPWM1_TASK_CLR1_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_CLR2_OST_ST_CLR 配置清除 MCPWM1_TASK_CLR2_OST 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_CAP0_ST_CLR 配置清除 MCPWM1_TASK_CAP0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_CAP1_ST_CLR 配置清除 MCPWM1_TASK_CAP1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM1_TASK_CAP2_ST_CLR 配置清除 MCPWM1_TASK_CAP2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_TASK_SAMPLE0_ST_CLR 配置清除 ADC_TASK_SAMPLE0 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.36. SOC_ETM_TASK_ST4_CLR_REG (0x020C)

接上页...

SOC_ETM_ADC_TASK_STARTO_ST_CLR 配置清除 ADC_TASK_STARTO 的状态。

0：无效值，没有作用

1：清除

(WT)

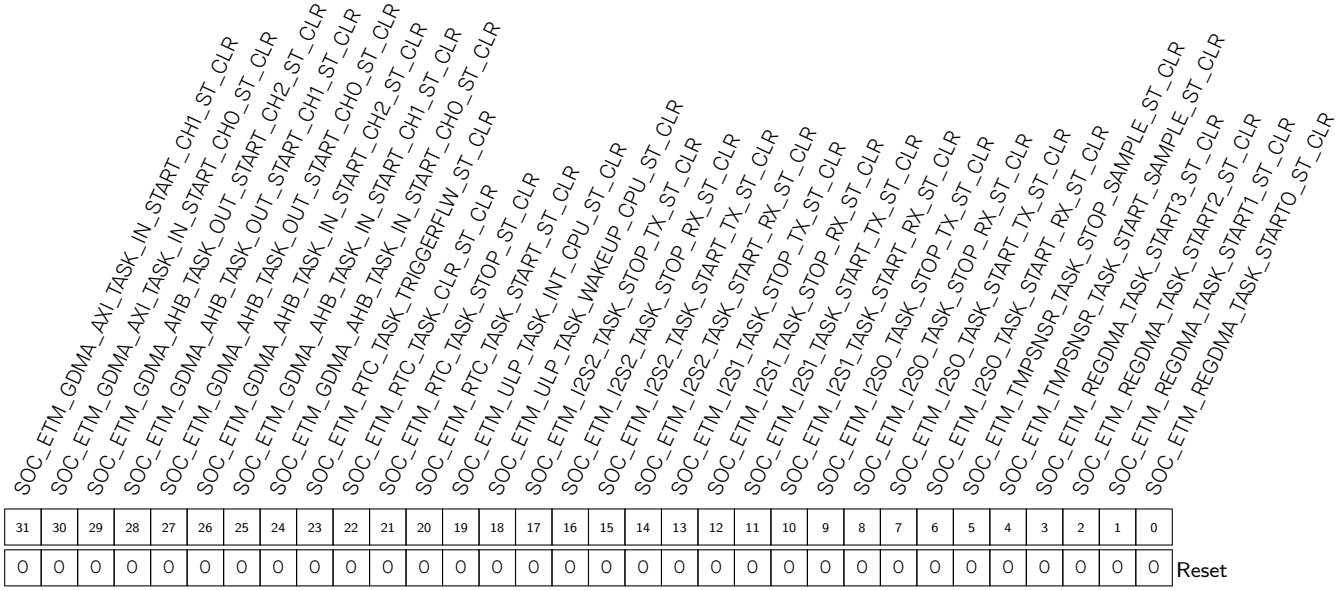
SOC_ETM_ADC_TASK_STOPO_ST_CLR 配置清除 ADC_TASK_STOPO 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)



SOC_ETM_REGDMA_TASK_START0_ST_CLR 配置清除 REGDMA_TASK_START0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_TASK_START1_ST_CLR 配置清除 REGDMA_TASK_START1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_TASK_START2_ST_CLR 配置清除 REGDMA_TASK_START2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_REGDMA_TASK_START3_ST_CLR 配置清除 REGDMA_TASK_START3 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST_CLR 配置清除 TMPSNSR_TASK_START_SAMPLE 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST_CLR 配置清除 TMPSNSR_TASK_STOP_SAMPLE 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

接上页...

SOC_ETM_I2S0_TASK_START_RX_ST_CLR 配置清除 I2S0_TASK_START_RX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_TASK_START_TX_ST_CLR 配置清除 I2S0_TASK_START_TX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_TASK_STOP_RX_ST_CLR 配置清除 I2S0_TASK_STOP_RX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_TASK_STOP_TX_ST_CLR 配置清除 I2S0_TASK_STOP_TX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_TASK_START_RX_ST_CLR 配置清除 I2S1_TASK_START_RX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_TASK_START_TX_ST_CLR 配置清除 I2S1_TASK_START_TX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_TASK_STOP_RX_ST_CLR 配置清除 I2S1_TASK_STOP_RX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S1_TASK_STOP_TX_ST_CLR 配置清除 I2S1_TASK_STOP_TX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S2_TASK_START_RX_ST_CLR 配置清除 I2S2_TASK_START_RX 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

接上页...

SOC_ETM_I2S2_TASK_START_TX_ST_CLR 配置清除 I2S2_TASK_START_TX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S2_TASK_STOP_RX_ST_CLR 配置清除 I2S2_TASK_STOP_RX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S2_TASK_STOP_TX_ST_CLR 配置清除 I2S2_TASK_STOP_TX 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST_CLR 配置清除 ULP_TASK_WAKEUP_CPU 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ULP_TASK_INT_CPU_ST_CLR 配置清除 ULP_TASK_INT_CPU 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_TASK_START_ST_CLR 配置清除 RTC_TASK_START 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_TASK_STOP_ST_CLR 配置清除 RTC_TASK_STOP 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_TASK_CLR_ST_CLR 配置清除 RTC_TASK_CLR 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST_CLR 配置清除 RTC_TASK_TRIGGERFLW 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.37. SOC_ETM_TASK_ST5_CLR_REG (0x0214)

接上页...

SOC_ETM_GDMA_AHB_TASK_IN_START_CH0_ST_CLR 配置清除 GDMA_AHB_TASK_IN_START_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH1_ST_CLR 配置清除 GDMA_AHB_TASK_IN_START_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_TASK_IN_START_CH2_ST_CLR 配置清除 GDMA_AHB_TASK_IN_START_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH0_ST_CLR 配置清除 GDMA_AHB_TASK_OUT_START_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

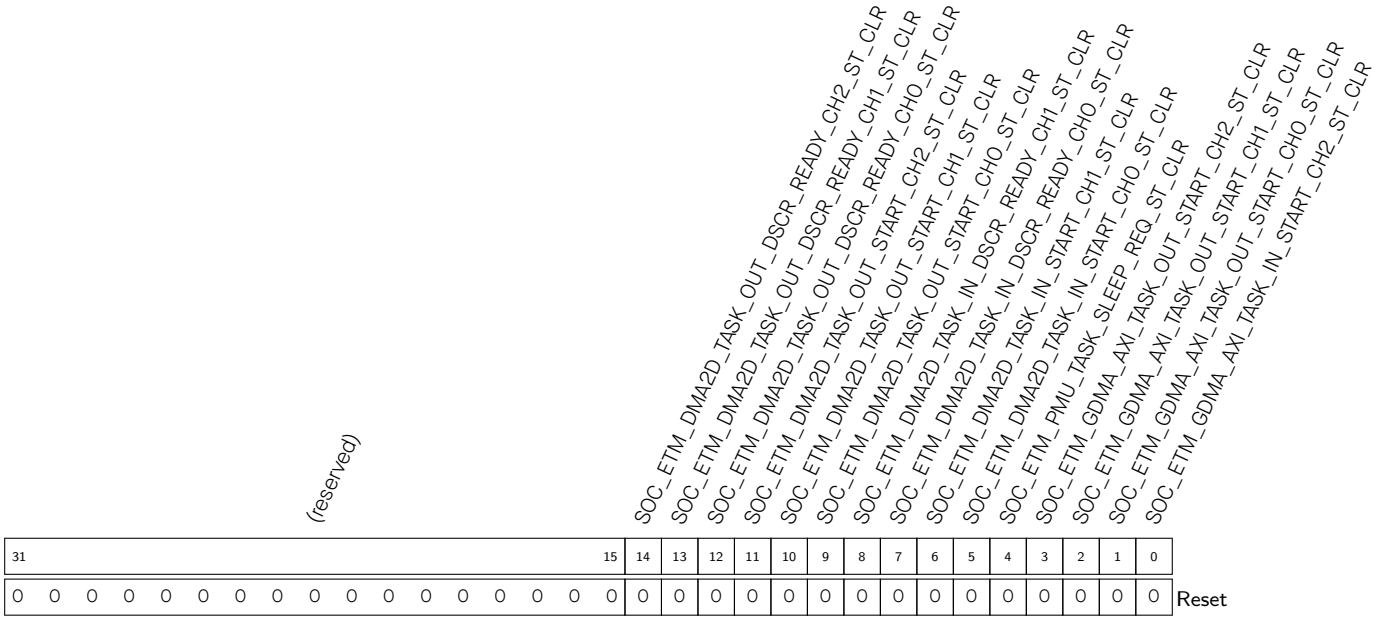
SOC_ETM_GDMA_AHB_TASK_OUT_START_CH1_ST_CLR 配置清除 GDMA_AHB_TASK_OUT_START_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AHB_TASK_OUT_START_CH2_ST_CLR 配置清除 GDMA_AHB_TASK_OUT_START_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH0_ST_CLR 配置清除 GDMA_AXI_TASK_IN_START_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_AXI_TASK_IN_START_CH1_ST_CLR 配置清除 GDMA_AXI_TASK_IN_START_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

Register 14.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)



SOC_ETM_GDMA_AXI_TASK_IN_START_CH2_ST_CLR 配置清除 GDMA_AXI_TASK_IN_START_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH0_ST_CLR 配置清除 GDMA_AXI_TASK_OUT_START_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH1_ST_CLR 配置清除 GDMA_AXI_TASK_OUT_START_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_AXI_TASK_OUT_START_CH2_ST_CLR 配置清除 GDMA_AXI_TASK_OUT_START_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST_CLR 配置清除 PMU_TASK_SLEEP_REQ 的状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 14.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)

接上页...

SOC_ETM_DMA2D_TASK_IN_START_CH0_ST_CLR 配置清除 DMA2D_TASK_IN_START_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_DMA2D_TASK_IN_START_CH1_ST_CLR 配置清除 DMA2D_TASK_IN_START_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH0_ST_CLR 配置清除 DMA2D_TASK_IN_DSCR_READY_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_DMA2D_TASK_IN_DSCR_READY_CH1_ST_CLR 配置清除 DMA2D_TASK_IN_DSCR_READY_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH0_ST_CLR 配置清除 DMA2D_TASK_OUT_START_CH0 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH1_ST_CLR 配置清除 DMA2D_TASK_OUT_START_CH1 的状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_DMA2D_TASK_OUT_START_CH2_ST_CLR 配置清除 DMA2D_TASK_OUT_START_CH2 的状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 14.38. SOC_ETM_TASK_ST6_CLR_REG (0x021C)

接上页...

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH0_ST_CLR 配 置 清 除

DMA2D_TASK_OUT_DSCR_READY_CH0 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH1_ST_CLR 配 置 清 除

DMA2D_TASK_OUT_DSCR_READY_CH1 的状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_DMA2D_TASK_OUT_DSCR_READY_CH2_ST_CLR 配 置 清 除

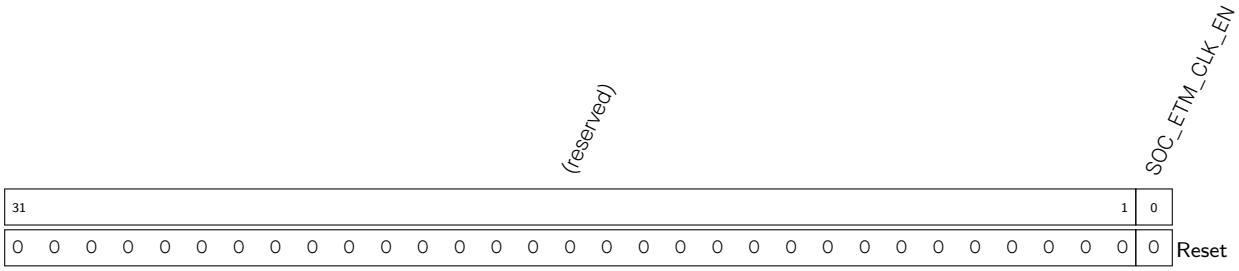
DMA2D_TASK_OUT_DSCR_READY_CH2 的状态。

0：无效值，没有作用

1：清除

(WT)

Register 14.39. SOC_ETM_CLK_EN_REG (0x0220)



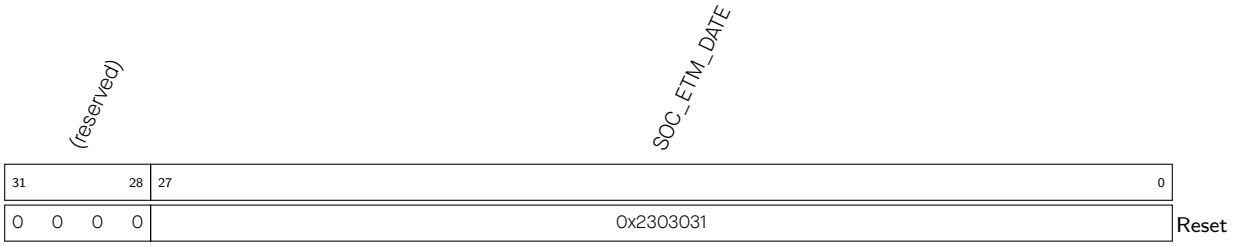
SOC_ETM_CLK_EN 配置寄存器时钟门控。

0：仅在应用写寄存器时开启时钟

1：强制为寄存器开启时钟门控

(R/W)

Register 14.40. SOC_ETM_DATE_REG (0x0224)



SOC_ETM_DATE 版本控制寄存器。(R/W)

第 15 章

低功耗管理

15.1 概述

ESP32-P4 拥有先进的低功耗管理系统，可以优化设备的功耗，并同时保持高性能。

低功耗管理系统通过采用多种节能技术，例如睡眠模式、动态电压和频率调节以及外设电源门控等，可以有效地将设备功耗降至最低。

在这一管理系统中，电源管理单元 (PMU) 扮演着核心角色，它可以灵活地控制各个电源域的上下电，以此实现芯片性能、功耗和唤醒时间之间的最佳平衡。

15.2 术语

下列有关低功耗管理的术语是在《ESP32-P4 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

低功耗管理系统	指的是管理芯片功耗的整个系统。
电源管理单元 (PMU)	指的是负责芯片功耗管理的核心控制电路，它包括对芯片电源、时钟以及功耗相关逻辑的控制。
电源域	指的是可独立控制上下电的最小单元。一个电源域可能包含一个或多个模块。
PMU 状态	指的是 PMU 状态机的状态。用户可以配置不同电源域在每一种状态下的时钟开关和电源开关。
功耗模式	指的是预设的典型功耗模式，可满足一些常见场景的功耗需求。

15.3 主要特性

PMU 具有以下特性：

- 支持 3 种可配置 PMU 状态，软件可根据需求灵活配置
  - HP_ACTIVE
  - HP_SLEEP
  - LP_SLEEP
- 支持 5 种预设功耗模式，可满足多种应用场景中的典型功耗需求
  - Active
  - Light-sleep0（外设不下电，仅 CPU 下电，不关闭时钟，功耗大，唤醒延迟小）
  - Light-sleep1（外设不下电，仅关闭时钟，功耗较大，唤醒延迟较小）

  - Light-sleep2（外设下电，需要备份恢复数据，功耗较小，唤醒延迟大）
  - Deep-sleep
- LP SRAM (32 KB)：任何模式下均保持上电，可存放 HP CPU 不掉电数据，LP CPU 的指令和数据
- 可灵活配置的备份 DMA，可在芯片切换 PMU 状态时，备份与恢复 CPU 及外设的关键数据
- 功耗控制器，可根据芯片所处的功耗模式切换状态，以控制电源和时钟等

15.4 功能描述

ESP32-P4 的低功耗管理系统包括如下组成部分：

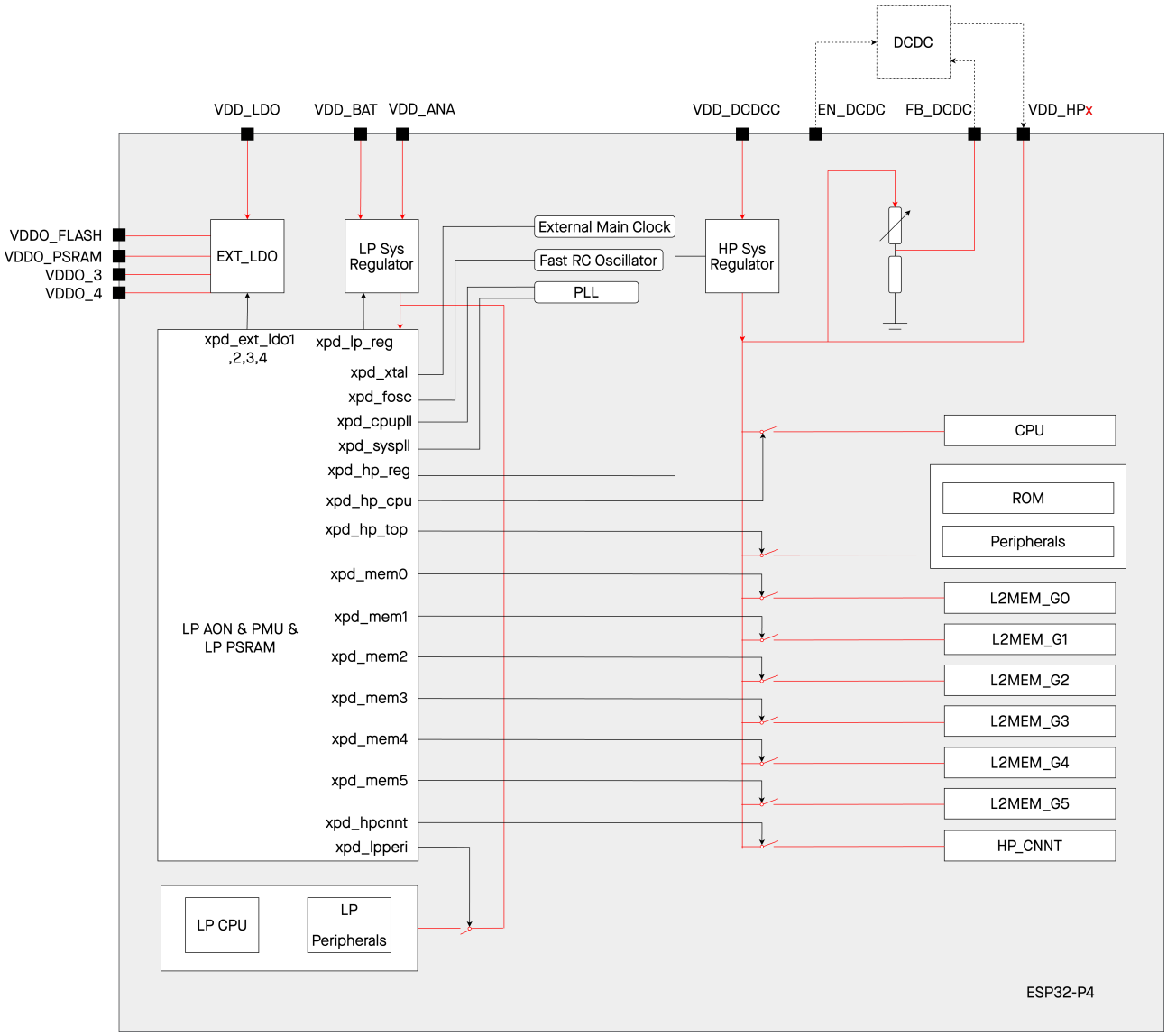
- **电源结构**：即供电管理网络，包括调压器、数字电源域、模拟电源域等部分。
- **PMU 控制器**：PMU 的核心部分，用于实现模拟电源、数字电源、时钟等系统控制。
- **1 个 RTC 定时器**：更多信息，请参考章节 19 实时时钟定时器。
- **16 个 LP GPIO 管脚 (GPIO0–GPIO15)**：这 16 个管脚永远处于工作状态，不受功耗模式的影响，可用作睡眠模式下的唤醒源，也可以作为标准的 GPIO 管脚使用。更多细节，请见章节 10 *GPIO 交换矩阵和 IO MUX*。
- **LP SRAM**：HP CPU 与 LP CPU 均可访问。HP CPU 访问时，该存储器使用 HP CPU 时钟；在 LP CPU 访问时，使用 LP CPU 时钟。
- **欠压检测器**: 用于实时监测供电管脚的电压，确保芯片不会因为电压毛刺或欠压而工作异常。有关欠压检测器的详细介绍，请参考章节 25 欠压检测器。

以下详细介绍这几个组成部分。

15.4.1 电源结构

ESP32-P4 的电源结构如图 15.4-1 所示，主要包含了：

- 一个 DCDC 调压反馈器
- 两个内部调压器
- 四个输出调压器 (EXT_LDO)
- 数字电源域
- 模拟电源域
- 电池电源域



—>: 控制信号 —: 电源线

图 15.4-1. ESP32-P4 电源结构

15.4.1.1 调压器

如图 15.4-1 所示，ESP32-P4 的模拟部分包含 1 个外置带反馈调节的 DCDC、2 个内部 HP/LP 系统调压器，负责调节不同电源域的供电：

- DCDC 调压反馈器，可以调节片外 DCDC 的电压。
- HP（高性能）系统调压器，驱动能力较强，功耗较大，输出电压可调，仅适用于上电和睡眠状态。
- LP（低功耗）系统调压器，负责调节低功耗模块的供电，输出电压同样可调。

DCDC 调压反馈器和 HP 系统调压器负责给 HP 系统供电。片外 DCDC 通过内部的电压反馈器可以调节输出电压。芯片上电默认使用 HP 系统调压器供电，上电完成后，建议切换至 DCDC 供电，以获得更好的效率和负载能力。

15.4.1.2 数字电源域

- HP 系统的数字电源域包括：

  - **CPU (PD_HP_CPU)**：主要包括 CPU 及支持其工作的外设。
  - **Peripherals + ROM (PD_TOP)**：主要包括总线、HP 外设等功能模块。
  - **L2MEM_G0–G5**：细分了 6 个子电源域，如图 15.4-1 所示。
  - **HP_CNNT (PD_HP_CNNT)**：主要包括 USB 以及 SDIO 等模块。

  调压器与每个电源域之间均有一个独立的电源开关，可实现独立的上下电控制。

- LP 系统的数字电源域包括：
  - **LP PD peripherals (PD_LP_PERI)**：主要包括 LP CPU、LP 外设。
  - **LP always-on (PD_AON)**：主要包括 LP AON 外设（如 RTC 定时器）和 PMU 控制模块，LP SRAM，所有模式下均不掉电。

15.4.1.3 模拟电源域

- HP 系统的模拟电源域包括：
  - CPLL_CLK
  - SPLL_CLK
  - Audio PLL
  - SDIO PLL
- LP 系统的模拟电源域包括：
  - External Main Clock（外部主时钟）
  - Fast RC Oscillator（快速 RC 振荡器）

15.4.1.4 电池电源域

电池电源域 (VDD_BAT) 为备用电源域，当所有外部供电 (VDD_ANA) 全部关断的时候，低功耗系统调压器、LP_SLOW_CLK、LP_FAST_CLK 可以全部切换到 VDD_BAT 电源供电。

15.4.2 PMU

ESP32-P4 的 PMU 控制着每个电源域功耗相关的组件，如电源、时钟，它主要由如下几个部分构成：

- **PMU 主状态机**：记录并切换 PMU 状态
- **睡眠与唤醒控制器**：用于向 PMU 主状态机发起睡眠与唤醒请求
- **电源控制器**：可根据芯片所处的功耗模式切换状态，以控制电源和时钟等。电源控制器包括：
  - **数字电源控制器**：用于控制数字电源域的上下电
  - **模拟电源控制器**：用于控制模拟模块的使能，比如调压器、模拟时钟等
  - **时钟控制器**：用于管理外设时钟的门控，以及选择数字时钟的模拟时钟源
  - **数据备份控制器**：用于控制 PMU 状态切换时的数据备份与恢复过程
  - **系统控制器**：用于控制部分系统级模块，如睡眠模式下（此时 CPU 不可用）看门狗功能的挂起等

PMU 的工作流程为：

- 睡眠与唤醒控制器向 PMU 主状态机发送睡眠或唤醒请求。
- PMU 主状态机产生电源门控、时钟门控和复位信号。
- 电源控制器和时钟控制器会根据 PMU 主状态机产生的信号，打开或关闭不同的电源域和时钟信号，从而让芯片进入或退出各种低功耗模式。

PMU 的工作流程如图 15.4-2 所示，其中，HP_SWITCH 和 LP_SWITCH 的定义如下：

- HP_SWITCH：HP 系统在 HP_ACTIVE 和 HP_SLEEP 之间切换的中间状态。硬件将在这个状态完成各个控制器的控制切换。
- LP_SWITCH：LP 系统在 HP_SLEEP 和 LP_SLEEP 之间切换的中间状态。硬件将在这个状态完成各个控制器的控制切换。

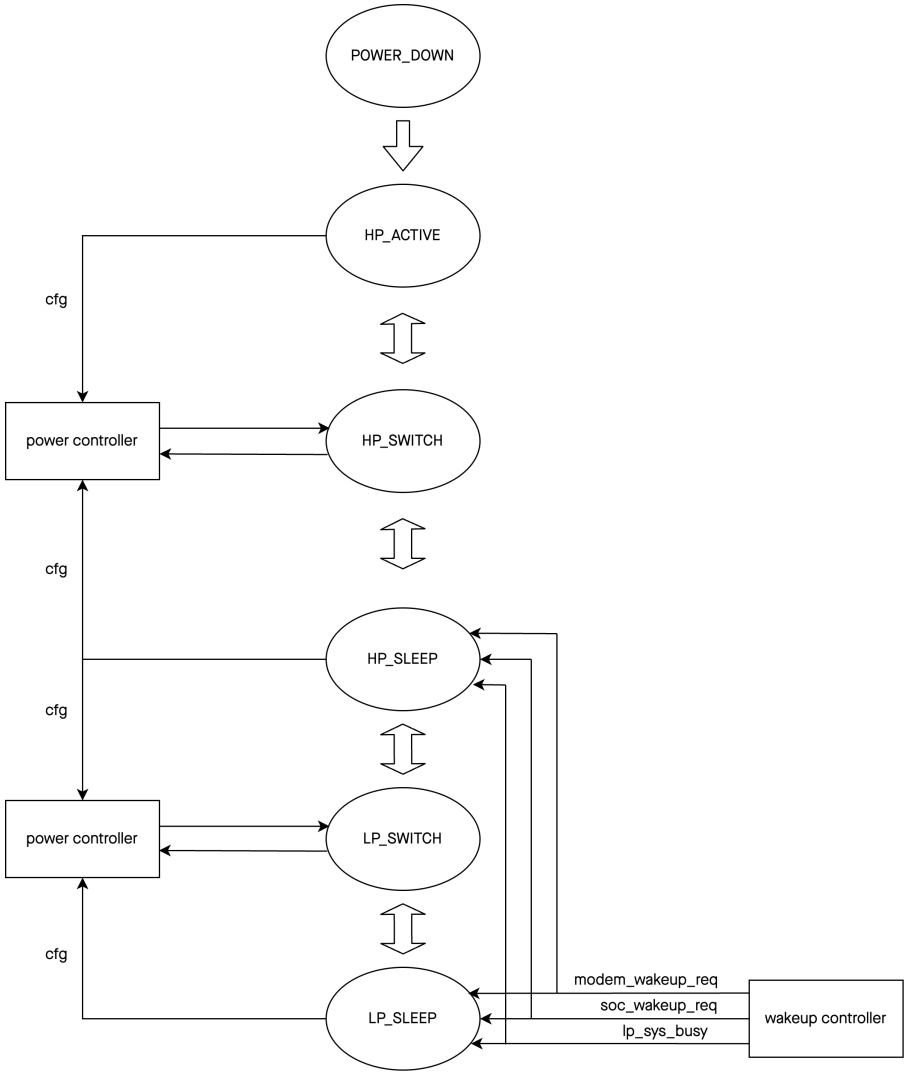


图 15.4-2. PMU 的主要工作流程

以下章节分别介绍 PMU 的各个主要组成部分。

15.4.2.1 PMU 主状态机

PMU 主状态机可以接收睡眠与唤醒信号，并通过功耗控制器完成电源上下电和时钟开关，从而实现 PMU 状态的切换，使芯片在目标 PMU 状态下达到性能与功耗的平衡。

PMU 主状态机包括 3 个 PMU 状态，每个状态由不同的睡眠与唤醒信号控制切换。软件可对每个 PMU 状态进行个性化的配置，如配置电源、时钟等。软件可基于这 3 个 PMU 状态，拓展出不同的应用层面的功耗模式。

3 个 PMU 状态为：

- **HP_ACTIVE**：所有电路最大程度开启的 PMU 状态，该状态支持 HP 系统和 LP 系统工作。
- **HP_SLEEP**：HP 系统睡眠状态，支持 LP 外设工作。
- **LP_SLEEP**：HP 系统与 LP 外设均处于睡眠状态，always-on 电路保持工作。

说明：
HP 和 LP 系统的划分为：

- LP 系统（外设）：LP RISC-V 32-bit Microprocessor、LP GPIO、LP UART、LP I2C、LP I2S、触摸传感器、温度传感器、LP ADC、LP SPI
- LP 系统（always-on 电路）：PMU，RTC 看门狗定时器，超级看门狗
- HP 系统：除 LP 系统以外的所有外设

更多内容可参考 《ESP32-P4 技术规格书》 > 功能框图。

HP_ACTIVE 为 HP 系统的工作状态，HP_SLEEP 为 HP 系统的睡眠状态，LP_SLEEP 为 LP 系统的睡眠状态。LP 系统的工作状态复用 HP_SLEEP 状态，因此也被称为 HP_SLEEP。

- 某个模块如果属于 HP 系统，则只能配置在 HP_ACTIVE/HP_SLEEP 两种状态下的上下电，在 LP_SLEEP 的状态下会复用 HP_SLEEP 的状态。
- 某个模块如果属于 LP 系统，则只能配置 HP_SLEEP/LP_SLEEP 状态下的上下电，在 HP_ACTIVE 状态下会复用 HP_SLEEP 的状态。

例如，HP CPU 属于 HP 系统，则可以通过以下寄存器配置 HP CPU 在 HP_ACTIVE/HP_SLEEP 状态下的上下电，LP_SLEEP 复用 HP_SLEEP 的配置。

- HP_ACTIVE：PMU_HP_ACTIVE_PD_HP_CPU_PD_EN
- HP_SLEEP：PMU_HP_SLEEP_PD_HP_CPU_PD_EN

类似地，用户可以用同样的方法，定义其他电源域在不同 PMU 状态下的上下电，具体寄存器请见第 15.8 小节。

说明：
为方便描述，后文中此类寄存器全部统称为 PMU_*n1*_PD_*POWERDOMAIN*_PD_EN，*n1* 代表 PMU 的 3 种状态。

配置完成后，PMU 将借助各控制器使这些配置生效，详见下方各章节的描述。

15.4.2.2 睡眠与唤醒控制器

睡眠与唤醒控制器负责向 PMU 主状态机发起睡眠与唤醒请求。ESP32-P4 支持多种唤醒源将 CPU 从不同功耗模式下唤醒，用户可按照表 15.4-1 使能对应的唤醒源。

表 15.4-1 中的唤醒源都能够唤醒 HP CPU 和 LP CPU。配置 PMU_WAKEUP_ENA 唤醒的是 HP CPU，配置 PMU_LP_CPU_WAKEUP_EN 唤醒的是 LP CPU。同时配置两个寄存器可以同时唤醒 HP CPU 和 LP CPU。

表 15.4-1. 唤醒源

PMU_WAKEUP_ENA/PMU_LP_CPU_WAKEUP_EN 的位	唤醒源	源电源域
0	HP_SDIO	PD_HP_CNNT
1	软件唤醒	PD_LP_PERI
2	HP GPIO	PD_TOP
3	HP USB	PD_HP_CNNT
4	HP UART4	PD_TOP
5	HP UART3	PD_TOP
6	HP UART2	PD_TOP
7	HP UART1	PD_TOP
8	HP UART0	PD_TOP
9	LP GPIO	PD_LP_PERI
10	LP UART	PD_LP_PERI
11	LP TOUCH	PD_LP_PERI
12	EXT IO	PD_LP_PERI
13	RTC_TIMER_TAR0	PD_AON
14	BROWN OUT	PD_AON
15	VBAT_UNDERVOLTAGE	PD_AON
16	LP CORE EXCEPTION	PD_LP_PERI
17	ETM	PD_TOP
18	RTC_TIMER_TAR1	PD_AON
19	LP I2S	PD_LP_PERI
20	GMAC PMT	PD_HP_CNNT
21	GMAC LPI	PD_HP_CNNT
22	LP CPU	PD_AON
23	HP CPU	PD_TOP

ESP32-P4 提供了硬件拒绝睡眠的机制，即当某些外设处于不可中断的工作状态而 HP CPU 试图进入睡眠状态时，外设将会发出唤醒信号，以此来拒绝 HP CPU 的睡眠命令，从而保证外设正常工作。

表 15.4-1 中的唤醒源均可配置成作为拒绝睡眠的事件，用户可配置下列寄存器实现拒绝睡眠的功能。PMU_SLEEP_REJECT_ENA 和 PMU_SLP_REJECT_CAUSE_REG 的配置值与唤醒源的对应关系均与表 15.4-1 中一致。

- 使能拒绝睡眠功能
  - 置位 PMU_SLP_REJECT_EN 使能拒绝睡眠功能
  - 配置 PMU_SLEEP_REJECT_ENA 选择拒绝睡眠信号源
- 读取寄存器 PMU_SLP_REJECT_CAUSE_REG 确认拒绝睡眠的事件源

15.4.2.3 模拟电源控制器

模拟电源控制器用于控制模拟电路部分（包括调压器、高性能时钟、低速时钟）在某种 PMU 状态下的上下电情况。

调压器的具体配置如下：

- 配置寄存器 PMU_*n1*_HP_REGULATOR_XPD 或 PMU_*n1*_LP_REGULATOR_XPD 可使能或关闭目标 PMU 状态下 HP 系统或 LP 系统调压器的输出电压。LP 系统调压器不建议关闭，否则会导致 PMU 本身掉电，从而导致芯片异常。

高性能时钟（XTAL_CLK 和 PLL 时钟）的具体配置如下：

- XTAL_CLK：配置 PMU_HP_SLEEP_XPD_XTAL 为 1，在芯片 PMU 状态切换至 HP_SLEEP 的过程中，开启 XTAL_CLK。

  注意：为避免 XTAL_CLK 起振阶段的不稳定时钟影响电路，可配置 PMU_WAIT_XTL_STABLE 实现 XTAL_CLK 全局门控的延迟打开，即该门控将在 XTAL_CLK 上电后的 PMU_WAIT_XTL_STABLE 个 CLK_DYN_FAST_CLK 周期时间后打开。

- CPLL_CLK：PMU 可在芯片处于不同 PMU 状态下打开 CPLL_CLK，仅需配置不同 PMU 状态下的 PMU_*n1*_XPD_PLL[0] 为 1。例如，配置 PMU_HP_ACTIVE_XPD_PLL[0] 为 1，则在芯片处于 HP_ACTIVE 状态下，打开 CPLL_CLK 时钟。

- SPLL_CLK：PMU 可在芯片处于不同 PMU 状态下打开 SPLL_CLK，仅需配置不同 PMU 状态下的 PMU_*n1*_XPD_PLL[1] 为 1。例如，配置 PMU_HP_ACTIVE_XPD_PLL[1] 为 1，则在芯片处于 HP_ACTIVE 状态下，打开 SPLL_CLK 时钟。

- Audio PLL：PMU 可在芯片处于不同 PMU 状态下打开 Audio_PLL_CLK，仅需配置不同 PMU 状态下的 PMU_*n1*_XPD_PLL[2] 为 1。例如，配置 PMU_HP_ACTIVE_XPD_PLL[2] 为 1，则在芯片处于 HP_ACTIVE 状态下，打开 Audio PLL 时钟。

- SDIO PLL：PMU 可在芯片处于不同 PMU 状态下打开 SDIO_PLL_CLK，仅需配置不同 PMU 状态下的 PMU_*n1*_XPD_PLL[3] 寄存器为 1。例如，配置 PMU_HP_ACTIVE_XPD_PLL[3] 为 1，则在芯片处于 HP_ACTIVE 状态下，打开 SDIO PLL 时钟。

  注意：

  - 在打开 PLL 时钟前还需确保 XTAL_CLK 已经稳定开启。
  - 为避免 PLL 时钟起振阶段的不稳定时钟影响电路，可配置 PMU_WAIT_PLL_STABLE 实现 PLL 时钟全局门控的延迟打开，即该门控将在 PLL 时钟上电后的 PMU_WAIT_PLL_STABLE 个 CLK_DYN_FAST_CLK 周期时间后打开。

低速时钟（RC_FAST_CLK 和 XTAL32K_CLK）功耗较低，在 HP_ACTIVE、HP_SLEEP 状态下的开关情况由 PMU_HP_SLEEP_XPD_FOSC_CLK 控制，在 LP_SLEEP 状态下由 PMU_LP_SLEEP_XPD_FOSC_CLK 控制。

15.4.2.4 数字电源控制器

数字电源控制器用于控制数字电源域在某种 PMU 状态下的上下电情况。与模拟电源控制器不同的是，数字电源控制器不直接控制调压器，而是通过控制与调压器相连的电源开关，实现各个数字电源域的上下电。

数字电源域中 LP PD Peripherals 只支持在 HP_SLEEP 与 LP_SLEEP 两个 PMU 状态之间切换时上下电，其余电源域只支持在 HP_SLEEP、HP_ACTIVE 两个 PMU 状态间切换时上下电。

当芯片在不同 PMU 状态之间切换时，如果某电源域在当下 PMU 状态下的上下电配置与即将切换至的上下电配置不符，则将激活上下电流程。以 CPU 电源域从上电到下电的流程为例。配置寄存器 PMU_HP_SLEEP_PD_TOP_PD_EN 为 1，表示在 HP_SLEEP 状态下关闭 CPU 电源域的供电；或配置为 0，表示在 HP_SLEEP 状态下对 CPU 电源域供电。PMU 将做出如下控制：

- 使能数字隔离单元，确保下电模块不往上电模块输出不稳定的电平值。当有电源域掉电时，该模块的输出将被钳位至固定值。
- 使能复位，CPU 电源域掉电后，该电源域的全局复位信号处于复位状态，在重新上电后的一段时间内该电源域仍然保持复位，这将确保该电源域在上电的过程中存在一个复位到释放的过程，消除上下电引入的寄存器不定态。

下面说明各个数字电源域的开关方式：

- L2MEM_G0–G5

  L2MEM_G0–G5 电源域的上下电状态统一由 PMU_*n1*_PD_HP_MEM_PD_EN 控制。

- HP_CNNT

  HP_CNNT 电源域的上下电状态由 PMU_*n1*_PD_CNNT_PD_EN 控制。

- PD_HP_CPU

  PD_HP_CPU 电源域的上下电状态由 PMU_*n1*_PD_HP_CPU_PD_EN 控制。

- Peripherals + ROM (PD_TOP)

  可通过寄存器 PMU_*n1*_PD_TOP_PD_EN 配置各电源域在对应 PMU 状态下的上下电。

  Peripherals 掉电时还存在以下几个可配功能：

  - Peripherals 掉电后会导致 GPIO 管脚状态不稳定，可通过 PMU 控制 GPIO 管脚（LP GPIO 除外）进入保持状态。例如配置 PMU_HP_SLEEP_HP_PAD_HOLD_ALL 使 GPIO 管脚在 HP_SLEEP 下保持 Peripherals 掉电前的相关配置。
  - 在 HP_ACTIVE 状态下，支持标准供电下的 Memory Deep-sleep 模式，该模式下存储器将无法读写，但可以保持数据。配置寄存器 PMU_*n1*_HP_MEM_DSLP 可实现 Memory Deep-sleep 模式。

- LP PD Peripherals

  LP PD Peripherals 电源域在芯片处于 HP_ACTIVE 状态下始终保持上电，无法配置，仅允许配置 HP_SLEEP 与 LP_SLEEP 状态下的上下电。此电源域拥有独立的数字电源开关，且它的上下电控制不依赖于其他数字电源域的上下电。

15.4.2.5 时钟控制器

时钟控制器主要用于芯片 PMU 状态切换时，对时钟源的选择及部分时钟的使能，以此实现对高性能系统时钟与低功耗系统时钟的控制。

高性能系统时钟包括 ROOT_CLK 和高性能系统外设时钟。当芯片的 PMU 状态在 HP_ACTIVE、HP_SLEEP 之间切换时，PMU 可实现对 ROOT_CLK 时钟源的切换、开关、分频，以及高性能系统外设时钟的开关。

- ROOT_CLK 控制的相关寄存器及功能介绍如下：
  - 配置不同 PMU 状态对应的 PMU_*n1*_SYS_CLK_SLP_SEL 寄存器为 1，表示芯片进入该 PMU 状态后，时钟源受 PMU 控制。
  - 配置不同 PMU 状态对应的 PMU_*n1*_ICG_SYS_CLOCK_EN 寄存器为 0，表示芯片进入该 PMU 状态后，关闭 ROOT_CLK 的时钟门控，即该时钟将不可用。
  - 配置不同 PMU 状态对应的 PMU_*n1*_DIG_SYS_CLK_SEL，选择芯片进入对应 PMU 状态后使用的时钟源，具体请见章节 11 复位和时钟 > 表 11.2-1。

- 高性能系统外设时钟控制的相关寄存器及功能介绍如下：
  - 配置 PMU_*n1*_ICG_SLP_SEL 寄存器为 1，目标状态下的各时钟门控的开关将受 PMU 的控制，该寄存器为 0 时，门控开关由 PCR 寄存器决定。
  - 配置寄存器 PMU_*n1*_DIG_ICG_FUNC_EN 决定目标 PMU 状态的功能时钟门控开关。具体配置请见表 15.4-2。

表 15.4-2. 高性能系统外设功能时钟

PMU_*n1*_DIG_ICG_FUNC_EN 位	时钟
bit 0	HP_CPU0_CLK
bit 1	HP_CPU1_CLK
bit 2	HP_CPU0_CLIC_CLK
bit 3	HP_CPU1_CLIC_CLK
bit 4	MISC_CPU_CLK
bit 5	MISC_SYS_CLK
bit 6	ICM_CPU_CLK
bit 7	ICM_SYS_CLK
bit 8	ICM_MEM_CLK
bit 9	ICM_APB_CLK
bit 10	SPM_CPU_CLK
bit 11	L2MEM_MEM_CLK
bit 12	L2MEM_SYS_CLK
bit 13	L1CACHE_CPU_CLK
bit 14	L1CACHE_D_CPU_CLK
bit 15	L1CACHE_I0_CPU_CLK
bit 16	L1CACHE_I1_CPU_CLK
bit 17	L1CACHE_MEM_CLK
bit 18	L1CACHE_D_MEM_CLK
bit 19	L1CACHE_I0_MEM_CLK
bit 20	L1CACHE_I1_MEM_CLK
bit 21	L2CACHE_MEM_CLK
bit 22	L2CACHE_SYS_CLK
bit 23	REGDMA_SYS_CLK
bit 24	HP_CLKRST_APB_CLK
bit 25	SYSREG_APB_CLK
bit 26	INTRMTX_CLK
bit 27	N/A
bit 28	N/A
bit 29	N/A
bit 30	N/A
bit 31	N/A

低功耗系统时钟主要用于低功耗系统，包括如下时钟：

- LP_SLOW_CLK

- LP_FAST_CLK
- LP_DYN_SLOW_CLK
- LP_DYN_FAST_CLK
- XTAL_D2_CLK

其中 LP_DYN_FAST_CLK 的时钟频率根据 PMU 状态由硬件完成如下固定切换（用户无法更改）：

- LP_SLEEP：频率与 LP_SLOW_CLK 一致。
- HP_ACTIVE、HP_SLEEP：频率与 LP_FAST_CLK 一致。

15.4.2.6 数据备份控制器

ESP32-P4 支持在切换 PMU 状态时，驱动一个 Retention DMA 模块，进行存储器与外设间的数据搬移，从而在电源域掉电时备份数据并在再次上电时恢复数据。

数据搬移操作由 Peripherals 电源域中的模块实现，PMU 只产生相关控制信号。需要注意的是，决定数据搬移的控制寄存器存在方向性，因为与 PMU 的其他控制寄存器不同的是，这些控制行为由原来的 PMU 状态和目标 PMU 状态共同决定。

比如，从 HP_ACTIVE 至 HP_SLEEP 与从 HP_SLEEP 至 HP_ACTIVE 所需要备份的寄存器显然是不同的，因此下文的寄存器名中的 *n2* 存在如下可能：

- HP_SLEEP2ACTIVE
- HP_ACTIVE2SLEEP

下面将介绍 PMU 支持的 Retention DMA 控制：

- 使能数据搬移：配置寄存器 PMU_*n2*_BACKUP_EN 为 1，表示在进行对应的 PMU 状态切换时使能数据搬移。
- 打开对应时钟：在数据搬移开始前，配置 PMU_*n2*_BACKUP_CLK_SEL 选择 Retention DMA 工作时钟源，配置 PMU_*n1*_BACKUP_ICG_FUNC_EN 为 1 打开时钟。

  数据搬移结束后，门控时钟寄存器 PMU_*n1*_BACKUP_ICG_FUNC_EN 的值将由目标状态下的 PMU_*n1*_DIG_ICG_FUNC_EN 决定。
- 配置数据搬移方向：配置 PMU_*n2*_BACKUP_MODE 寄存器的最高位：
  - 1：外设到存储器
  - 0：存储器到外设
- 选择链表指针：配置 PMU_*n2*_BACKUP_MODE 寄存器的低两位，选择链表指针：
  - 0：PAU_LINK_ADDR_0
  - 1：PAU_LINK_ADDR_1
  - 2：PAU_LINK_ADDR_2
  - 3：PAU_LINK_ADDR_3

15.4.2.7 系统控制器

系统控制器用于在芯片切换 PMU 状态时控制部分功能模块，以实现芯片在各 PMU 状态下稳定、节能地工作。系统控制器支持以下功能：

- **暂停看门狗功能**：配置 PMU_*n1*_DIG_PAUSE_WDT 寄存器为 1，可在 PMU 状态切换至对应 PMU 状态时禁用 RTC 看门狗定时器 (RWDT) 功能。注意，任一睡眠模式下若配置该寄存器为 0，表示不禁用该功能，则 RWDT 将因为没有 CPU 参与喂狗而产生复位。
- **切换 GPIO 至管脚睡眠模式**，该模式下将固定管脚的配置。例如将某个 GPIO 管脚固定为输入，低驱动模式以降低其功耗，并作为一个唤醒管脚使用，配置 PMU_*n1*_HP_PAD_HOLD_ALL 寄存器为 1，可在 PMU 状态切换至对应 PMU 状态后锁存当前的配置作为 GPIO 的睡眠配置。GPIO 管脚保持功能的详细介绍请参考章节 10 *GPIO 交换矩阵和 IO MUX* > 小节 10.9 *GPIO 管脚的 Hold 特性*。
- **禁用 UART 唤醒功能**：配置 PMU_*n1*_UART_WAKEUP_EN 寄存器为 0，禁用对应 PMU 状态下的 4 种 UART 唤醒模式。有关 UART 唤醒模式的具体介绍，请见章节 45 *UART 控制器 (UART)*。
- **暂停 CPU 运行**：配置 PMU_*n1*_DIG_CPU_STALL 寄存器为 1，在对应 PMU 状态下将 CPU 挂起。

15.4.2.8 电池供电控制器

如图 15.4-1 所示，低功耗系统调压器可以由 VDD_BAT（电池）或 VDD_ANA（模拟）供电。PMU_*n1*_VDDBAT_MODE 控制是否使能电池供电，0 是不使能电池供电，此时低功耗系统调压器由 VDD_ANA 供电，可用于 Active 模式或 Light-sleep0/1 模式。PMU_*n1*_VDDBAT_MODE 为 1 时，使能电池供电，可用于包括 VDD_ANA 在内的主供电切断的 Deep-sleep 模式下，此时只有 LP AON & PMU 电源域有电。

15.4.2.9 输出调压器控制

如图 15.4-1 所示，ESP32-P4 有四个可以输出的调压器：

- VDDO_FLASH (VO1)，通常用于片上 flash 调压
- VDDO_PSRAM (VO2)，通常用于片上 PSRAM 调压
- VDDO_3 (VO3) 和 VDDO_4 (VO4)，可以将输入的 VDD_LDO（通常为 3.3 V）转换成 flash、PSRAM、MIPI、SDIO 等外设所需要的电压。

15.4.2.9.1 调压器特性

调压器支持如下特性：

- **支持 LDO 模式和透传模式**。LDO 模式支持输出 0.5 V –2.7 V，透传模式输出 VDD_LDO 的电压。模式选择由硬件状态机或者寄存器控制，控制权由寄存器控制。flash 调压器 (VO1) 的控制权还支持 eFuse 控制。
  - VO1 的模式控制关系如图 15.4-3 所示。图中最右边 TIEH_SEL 为 0，选择 LDO 模式；为 1，选择透传模式。

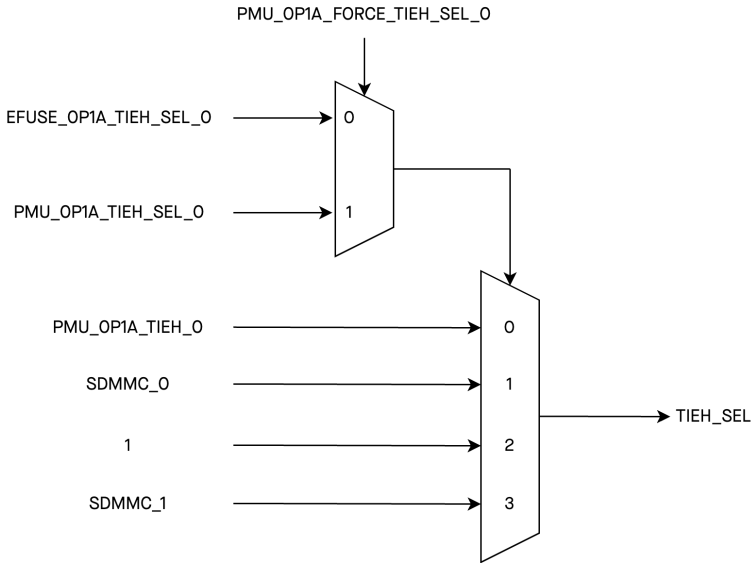


图 15.4-3. VO1 模式选择控制图

- VO2/3/4 的模式控制关系如图 15.4-4 所示。图中最右边 TIEH_SEL 为 0，选择 LDO 模式；为 1，选择透传模式。

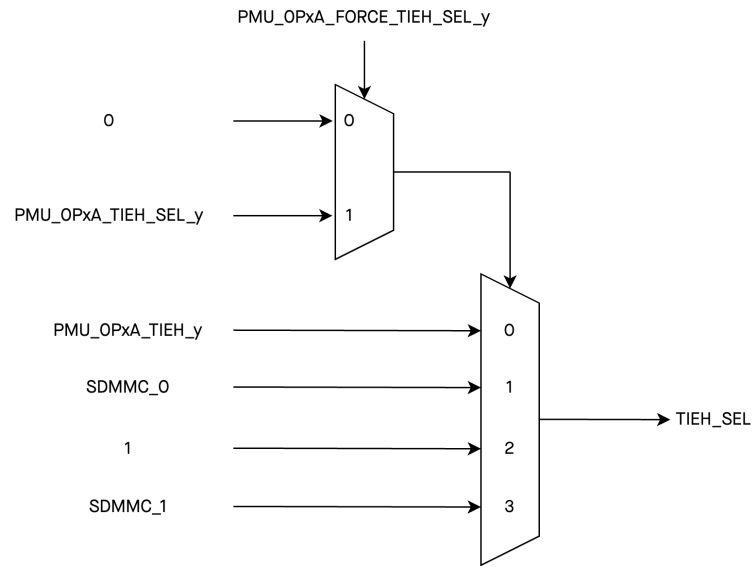


图 15.4-4. VO2/3/4 模式选择控制图

表 15.4-3. VO2/3/4 寄存器对应关系

调压器	PMU_0PxA_FORCE_TIEH_SEL_y	PMU_0PxA_TIEH_SEL_y	PMU_0PxA_TIEH_y
VO2	PMU_0P1A_FORCE_TIEH_SEL_1	PMU_0P1A_TIEH_SEL_1	PMU_0P1A_TIEH_1
VO3	PMU_0P2A_FORCE_TIEH_SEL_0	PMU_0P2A_TIEH_SEL_0	PMU_0P2A_TIEH_0
VO4	PMU_0P2A_FORCE_TIEH_SEL_1	PMU_0P2A_TIEH_SEL_1	PMU_0P2A_TIEH_1

- **支持 LDO 输出电压设置**：当选择 LDO 模式时，输出电压由模拟寄存器 PMU_ANA_0PxA_DREF_y（设置基准电压 Vref）和 PMU_ANA_0PxA_MUL_y（设置倍率 MUL）共同决定，满足

$V_{OUT} = \text{MUL} \times V_{REF}$。其中 Vref 的编码范围为 0.5 V–1.6 V，MUL 的编码范围为 1.0–2.75（步进 0.25），详见 PMU_EXT_LDO_P0_0P1A_ANA_REG–PMU_EXT_LDO_P0_0P2A_ANA_REG 的字段说明。

- **支持开关控制**。除了 flash 调压器 (VO1) 支持 PMU 状态机切换电源以外，其余调压器只支持软件开关。表 15.4-4 中寄存器为 1 时开启调压器，否则关闭调压器。

表 15.4-4. 调压器开关控制

调压器	调压器开关控制
VO1	PMU_*n1*_REGULATOR_VO1_XPD
VO2	PMU_0P1A_XPD_1
VO3	PMU_0P2A_XPD_0
VO4	PMU_0P2A_XPD_1

- **限流保护**：调压开启时，为防止瞬态电流过大，损坏电容，提供限流保护功能，需要在开启之前开启限流保护。当电容充电完成之后，关闭限流保护。电容充电时间与负载电容、电压有关。计算公式为：$T_{ms} = C_F \times V_V / 50$。表 15.4-5 为 VO 的限流保护开启使能寄存器，寄存器为 1 时开启限流保护，反之关闭。

表 15.4-5. 限流保护控制

调压器	限流保护控制
VO1	FSM 或 PMU_ANA_0P1A_EN_CUR_LIM_0 控制
VO2	PMU_ANA_0P1A_EN_CUR_LIM_1
VO3	PMU_ANA_0P2A_EN_CUR_LIM_0
VO4	PMU_ANA_0P2A_EN_CUR_LIM_1

由于开启电源的时间较长，系统为每个调压器提供了等待计数器。每个调压器开启电源的时刻会触发计数器开始计数，计数器有两个阶段（stage 1/stage 2，对应寄存器 TARGET0/TARGET1），每个阶段会触发不同的中断。通常可将 stage 1 (TARGET0) 配置为限流保护持续时间，将 stage 2 (TARGET1) 配置为输出电压稳定等待时间。计数器的配置见下表：

表 15.4-6. 调压器等待计数器

调压器	计数器目标 1	计数器目标 2
VO1	PMU_0P1A_TARGET0_0[6:0]	PMU_0P1A_TARGET1_0[7:0]
VO2	PMU_0P1A_TARGET0_1[7:0]	PMU_0P1A_TARGET1_1[7:0]
VO3	PMU_0P2A_TARGET0_0[7:0]	PMU_0P2A_TARGET1_0[7:0]
VO4	PMU_0P2A_TARGET0_1[7:0]	PMU_0P2A_TARGET1_1[7:0]

注意：

1. PMU_0P1A_TARGET0_0 的 bit7 用于清空计数器，当该 bit 为 1 时，计数器处于复位状态。

2. 计数器的时钟为 LP_DYN_SLOW_CLK 的 16 分频。

15.4.2.9.2 板级用法示例

- **示例 1：VO3 为 MIPI D-PHY 供电（2.5 V，LDO 模式）**
  在板级上将 VO3 引脚连接至 VDD_MIPI_DPHY 等需要 2.5 V 的电源网络。配置思路：选择 LDO 模式，并设置 $V_{OUT} = 2.5\,\text{V}$。一种可行组合为 $V_{REF} = 1.0\,\text{V}$（DREF=9）与 MUL = 2.5（MUL=6），即

PMU_ANA_0P2A_DREF_0=9、PMU_ANA_0P2A_MUL_0=6，并将 PMU_0P2A_TIEH_0=0（LDO）。

- **示例 2：VO4 为 VDD_IO_5 供电（SD 卡接口电压切换）**
  在参考设计中，VO4 可能连接至 VDD_IO_5 以为部分 GPIO/SDIO 供电。
  – 输出 3.3 V：可选择透传模式（例如设置 PMU_0P2A_TIEH_1=1）。
  – 输出 1.8 V：选择 LDO 模式，并设置 $V_{OUT} = 1.8\,V$。一种可行组合为 $V_{REF} = 0.9\,V$（DREF=8）与 MUL = 2.0（MUL=4），即 PMU_ANA_0P2A_DREF_1=8、PMU_ANA_0P2A_MUL_1=4，并将 PMU_0P2A_TIEH_1=0（LDO）。

15.4.2.9.3 软件配置步骤

推荐的软件配置步骤（示例）如下：

1. 配置输出模式：设置 PMU_0PxA_FORCE_TIEH_SEL_y 使模式控制由寄存器生效，然后通过 PMU_0PxA_TIEH_SEL_y/PMU_0PxA_TIEH_y 选择 LDO 或透传模式（见表 15.4-3）。
2. 若使用 LDO 模式，配置输出电压：设置对应的 PMU_ANA_0PxA_DREF_y 与 PMU_ANA_0PxA_MUL_y 以满足 $V_{OUT} = \text{MUL} \times V_{REF}$。
3. 配置等待计数器：将 stage 1 (TARGET0) 设置为限流保护持续时间，将 stage 2 (TARGET1) 设置为稳定等待时间（见表 15.4-6）。
4. 上电前使能限流保护：设置对应 PMU_ANA_0PxA_EN_CUR_LIM_y=1（见表 15.4-5）。
5. 打开调压器：设置对应 PMU_0PxA_XPD_y=1（见表 15.4-4），并等待 stage 1 超时中断（例如 VO3 对应 PMU_0P2A_CNT_TARGET0_REACH_0_INT，VO4 对应 PMU_0P2A_CNT_TARGET0_REACH_1_INT）。
6. 收到 stage 1 中断后关闭限流保护：将 PMU_ANA_0PxA_EN_CUR_LIM_y 置 0；如需保证稳定，可继续等待 stage 2 超时中断（...CNT_TARGET1_REACH..._INT）。

15.5 功耗模式

ESP32-P4 有 3 个可配置的 PMU 状态，为了方便用户的使用，我们基于这 3 个可配置的 PMU 状态，提供了 5 种预设功耗模式，可满足最常见的场景需求，具体如表 15.5-1 所示。

表 15.5-1. 预设的功耗模式

功耗模式	电源域							
	LP always-on	LP PD peripherals	HP_CPU	TOP	HPCNNT	RC_FAST_CLK	XTAL_CLK	PLL
Active	ON	ON	ON	ON	ON	ON	ON	ON
Light-sleep0	ON	ON	OFF	ON	ON	ON	ON	ON
Light-sleep1	ON	ON	OFF	ON	ON	ON/OFF	ON/OFF	ON/OFF
Light-sleep2	ON	ON/OFF	OFF	OFF	ON	ON/OFF	ON/OFF	ON/OFF
Deep-sleep	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF

说明：

1. 具体功耗数据可见《ESP32-P4 技术规格书》。
2. 具体可支持的唤醒源，请见表 15.4-1。

15.6 事件任务矩阵功能

在 ESP32-P4 中，低功耗管理模块支持 ETM 功能，即可以通过任意外设的 ETM 事件触发低功耗管理模块的 ETM 任务，或者通过低功耗管理模块的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与低功耗管理模块相关的 ETM 任务和 ETM 事件。

低功耗管理模块可接收的 ETM 任务有：

- PMU_TASK_SLEEP_REQ：该任务将触发 PMU 开始睡眠流程。

低功耗管理模块可产生的 ETM 事件有：

- PMU_EVT_SLEEP_WAKEUP：该事件表示 PMU 被唤醒至 HP_ACTIVE。

15.7 中断

ESP32-P4 的低功耗管理模块可以生成以下中断信号：

- PMU_INTR
- PMU_LP_INT

其中，PMU_INTR 发送给 中断矩阵；PMU_LP_INT 输出到 LP CPU。

这些中断信号由内部中断源生成。具体为：

PMU_INTR：

- PMU_SOC_WAKEUP_INT：当芯片被唤醒至 HP_ACTIVE 时触发。
- PMU_SOC_SLEEP_REJECT_INT：当拒绝睡眠源拒绝了一个睡眠请求时触发。
- PMU_SW_INT：当用 LP CPU 作为唤醒源将芯片唤醒至 HP_ACTIVE 时触发。
- PMU_SDIO_IDLE_INT：当 SDIO 处于 IDLE 状态时触发。
- PMU_LP_CPU_EXC_INT：当 LP CPU 异常时触发。
- PMU_OP2A_CNT_TARGET1_REACH_1_INT：VO4 调压器等待计数器目标 1 超时中断。
- PMU_OP2A_CNT_TARGET0_REACH_1_INT：VO4 调压器等待计数器目标 0 超时中断。
- PMU_OP2A_CNT_TARGET1_REACH_0_INT：VO3 调压器等待计数器目标 1 超时中断。
- PMU_OP2A_CNT_TARGET0_REACH_0_INT：VO3 调压器等待计数器目标 0 超时中断。
- PMU_OP1A_CNT_TARGET1_REACH_1_INT：VO2 调压器等待计数器目标 1 超时中断。
- PMU_OP1A_CNT_TARGET0_REACH_1_INT：VO2 调压器等待计数器目标 0 超时中断。
- PMU_OP1A_CNT_TARGET1_REACH_0_INT：VO1 调压器等待计数器目标 1 超时中断。
- PMU_OP1A_CNT_TARGET0_REACH_0_INT：VO1 调压器等待计数器目标 0 超时中断。

PMU_LP_INT：

- PMU_LP_CPU_SLEEP_REJECT_INT：当 LP CPU 睡眠被拒绝时触发。
- PMU_HP_SW_TRIGGER_INT：当用 HP CPU 作为唤醒源唤醒 LP CPU 时触发。
- PMU_ACTIVE_SWITCH_SLEEP_START_INT：当 PMU 状态开始从 HP_ACTIVE 切换至 HP_SLEEP 时触发。
- PMU_SLEEP_SWITCH_ACTIVE_START_INT：当 PMU 状态开始从 HP_SLEEP 切换至 HP_ACTIVE 时触发。
- PMU_ACTIVE_SWITCH_SLEEP_END_INT：当 PMU 状态完成从 HP_ACTIVE 切换至 HP_SLEEP 时触发。
- PMU_SLEEP_SWITCH_ACTIVE_END_INT：当 PMU 状态完成从 HP_SLEEP 切换至 HP_ACTIVE 时触发。
- PMU_LP_CPU_WAKEUP_INT：当 LP CPU 被唤醒时触发。
- PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT：VO4 调压器等待计数器目标 1 超时中断。
- PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT：VO4 调压器等待计数器目标 0 超时中断。
- PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT：VO3 调压器等待计数器目标 1 超时中断。
- PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT：VO3 调压器等待计数器目标 0 超时中断。
- PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT：VO2 调压器等待计数器目标 1 超时中断。
- PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT：VO2 调压器等待计数器目标 0 超时中断。
- PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT：VO1 调压器等待计数器目标 1 超时中断。
- PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT：VO1 调压器等待计数器目标 0 超时中断。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 15.8 寄存器列表。

15.8 寄存器列表

本小节的所有地址均为相对于 PMU 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
PMU_HP_ACTIVE_DIG_POWER_REG	HP_ACTIVE 状态下数字电源域配置寄存器	0x0000	R/W
PMU_HP_ACTIVE_ICG_HP_FUNC_REG	HP_ACTIVE 状态下高性能系统外设功能时钟控制寄存器	0x0004	R/W
PMU_HP_ACTIVE_HP_SYS_CNTL_REG	HP_ACTIVE 状态下系统控制寄存器	0x0010	R/W
PMU_HP_ACTIVE_HP_CK_POWER_REG	HP_ACTIVE 状态下时钟源电源控制寄存器	0x0014	R/W
PMU_HP_ACTIVE_BIAS_REG	HP_ACTIVE 状态下的 BIAS 控制寄存器	0x0018	R/W
PMU_HP_ACTIVE_BACKUP_REG	HP_ACTIVE 状态下数据备份控制寄存器	0x001C	R/W
PMU_HP_ACTIVE_BACKUP_CLK_REG	HP_ACTIVE 状态下备份模块工作时钟控制寄存器	0x0020	R/W
PMU_HP_ACTIVE_SYSCLK_REG	HP_ACTIVE 状态下系统时钟控制寄存器	0x0024	R/W
PMU_HP_ACTIVE_HP_REGULATORO_REG	HP_ACTIVE 状态下调压器电源控制寄存器	0x0028	varies
PMU_HP_ACTIVE_XTAL_REG	HP_ACTIVE 状态下 XTAL_CLK 电源控制寄存器	0x0030	R/W
PMU_HP_SLEEP_DIG_POWER_REG	HP_SLEEP 状态下数字电源域控制寄存器	0x0068	R/W
PMU_HP_SLEEP_ICG_HP_FUNC_REG	HP_SLEEP 状态下高性能系统外设功能时钟控制寄存器	0x006C	R/W
PMU_HP_SLEEP_HP_SYS_CNTL_REG	HP_SLEEP 状态下系统控制寄存器	0x0078	R/W
PMU_HP_SLEEP_HP_CK_POWER_REG	HP_SLEEP 状态下时钟源电源控制寄存器	0x007C	R/W
PMU_HP_SLEEP_BIAS_REG	HP_SLEEP 状态下的 BIAS 控制寄存器	0x0080	R/W
PMU_HP_SLEEP_BACKUP_REG	HP_SLEEP 状态下数据备份控制寄存器	0x0084	R/W
PMU_HP_SLEEP_BACKUP_CLK_REG	目前状态为 HP_SLEEP 状态时，数据备份时钟控制寄存器	0x0088	R/W
PMU_HP_SLEEP_SYSCLK_REG	HP_SLEEP 状态下系统时钟控制寄存器	0x008C	R/W
PMU_HP_SLEEP_HP_REGULATORO_REG	HP_SLEEP 状态下调压器控制寄存器	0x0090	R/W
PMU_HP_SLEEP_XTAL_REG	HP_SLEEP 状态下 XTAL_CLK 电源控制寄存器	0x0098	R/W
PMU_HP_SLEEP_LP_REGULATORO_REG	HP_SLEEP 状态下 LP 系统的调压器的控制寄存器	0x009C	R/W

名称	描述	地址	访问
PMU_HP_SLEEP_LP_DIG_POWER_REG	HP_SLEEP 状态下 LP 系统数字电源域控制寄存器	0x00A8	R/W
PMU_HP_SLEEP_LP_CK_POWER_REG	HP_ACTIVE/HP_SLEEP 状态下低速时钟电源控制寄存器	0x00AC	R/W
PMU_LP_SLEEP_LP_REGULATOR0_REG	HP_SLEEP 状态下 LP 系统调压器的控制寄存器	0x00B4	R/W
PMU_LP_SLEEP_XTAL_REG	LP_SLEEP 状态下 XTAL_CLK 电源控制寄存器	0x00BC	R/W
PMU_LP_SLEEP_LP_DIG_POWER_REG	LP_SLEEP 状态下数字电源域控制寄存器	0x00C0	R/W
PMU_LP_SLEEP_LP_CK_POWER_REG	LP_SLEEP 状态下低速时钟电源控制寄存器	0x00C4	R/W
PMU_IMM_HP_CK_POWER_REG	软件控制时钟电源寄存器	0x00CC	WT
PMU_IMM_SLEEP_SYSCLK_REG	软件控制系统时钟寄存器	0x00D0	WT
PMU_IMM_HP_FUNC_ICG_REG	软件控制外设门控时钟寄存器	0x00D4	WT
PMU_IMM_HP_APB_ICG_REG	软件控制 APB 时钟寄存器	0x00D8	WT
PMU_IMM_PAD_HOLD_ALL_REG	软件控制 PAD 的保持功能	0x00E4	varies
PMU_POWER_CK_WAIT_CNTL_REG	时钟门控等待时间控制寄存器	0x011C	R/W
PMU_SLP_WAKEUP_CNTL0_REG	睡眠请求寄存器	0x0120	WT
PMU_SLP_WAKEUP_CNTL1_REG	睡眠拒绝寄存器	0x0124	R/W
PMU_SLP_WAKEUP_CNTL2_REG	唤醒源使能寄存器	0x0128	R/W
PMU_SLP_WAKEUP_CNTL3_REG	最小睡眠时间控制寄存器	0x012C	R/W
PMU_SLP_WAKEUP_CNTL4_REG	睡眠拒绝事件清除寄存器	0x0130	WT
PMU_SLP_WAKEUP_STATUS0_REG	唤醒事件寄存器	0x0144	RO
PMU_SLP_WAKEUP_STATUS1_REG	复位拒绝事件寄存器	0x0148	RO
PMU_HP_CK_CNTL_REG	HP 系统的时钟门控控制寄存器	0x0154	R/W
PMU_RF_PWC_REG	SAR ADC 上电配置寄存器	0x015C	R/W
PMU_INT_RAW_REG	PMU 睡眠/唤醒原始中断	0x0164	R/WTC/SS
PMU_HP_INT_ST_REG	PMU 睡眠/唤醒状态中断	0x0168	RO
PMU_HP_INT_ENA_REG	PMU 睡眠/唤醒中断使能	0x016C	R/W
PMU_HP_INT_CLR_REG	PMU 睡眠/唤醒中断清除	0x0170	WT
PMU_LP_INT_RAW_REG	PMU 睡眠/唤醒原始中断	0x0174	R/WTC/SS
PMU_LP_INT_ST_REG	PMU 睡眠/唤醒状态中断	0x0178	RO
PMU_LP_INT_ENA_REG	PMU 睡眠/唤醒中断使能	0x017C	R/W
PMU_LP_INT_CLR_REG	PMU 睡眠/唤醒中断清除	0x0180	varies
PMU_LP_CPU_PWR0_REG	LP CPU 睡眠/唤醒控制寄存器	0x0184	varies
PMU_LP_CPU_PWR1_REG	LP CPU 睡眠/唤醒控制寄存器	0x0188	WT
PMU_LP_CPU_PWR2_REG	LP CPU 睡眠/唤醒控制寄存器	0x018C	R/W
PMU_LP_CPU_PWR3_REG	LP CPU 睡眠/唤醒控制寄存器	0x0190	RO
PMU_LP_CPU_PWR4_REG	LP CPU 睡眠/唤醒控制寄存器	0x0194	R/W
PMU_LP_CPU_PWR5_REG	LP CPU 睡眠/唤醒控制寄存器	0x0198	RO
PMU_HP_LP_CPU_COMM_REG	LP CPU 睡眠/唤醒控制寄存器	0x019C	WT
PMU_EXT_LDO_P0_0P1A_REG	VO1 调压器控制寄存器	0x01B8	R/W

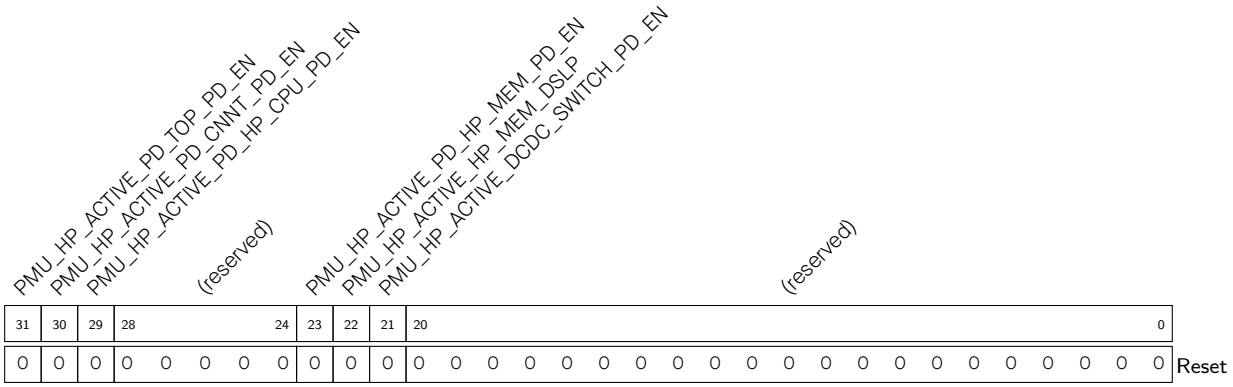
名称	描述	地址	访问
PMU_EXT_LDO_P0_OP1A_ANA_REG	VO1 调压器控制寄存器	0x01BC	R/W
PMU_EXT_LDO_P0_OP2A_REG	VO3 调压器控制寄存器	0x01C0	R/W
PMU_EXT_LDO_P0_OP2A_ANA_REG	VO3 调压器控制寄存器	0x01C4	R/W
PMU_EXT_LDO_P1_OP1A_REG	VO2 调压器控制寄存器	0x01D0	R/W
PMU_EXT_LDO_P1_OP1A_ANA_REG	VO2 调压器控制寄存器	0x01D4	R/W
PMU_EXT_LDO_P1_OP2A_REG	VO4 调压器控制寄存器	0x01D8	R/W
PMU_EXT_LDO_P1_OP2A_ANA_REG	VO4 调压器控制寄存器	0x01DC	R/W
PMU_EXT_WAKEUP_LV_REG	EXT 唤醒电平控制寄存器	0x01E8	R/W
PMU_EXT_WAKEUP_SEL_REG	EXT 唤醒 IO 控制寄存器	0x01EC	R/W
PMU_EXT_WAKEUP_ST_REG	EXT 唤醒状态寄存器	0x01F0	RO
PMU_EXT_WAKEUP_CNTL_REG	EXT 唤醒控制寄存器	0x01F4	R/W
PMU_SDIO_WAKEUP_CNTL_REG	SDIO 唤醒控制寄存器	0x01F8	R/W
PMU_TOUCH_PWR_CNTL_REG	TOUCH 睡眠/唤醒控制寄存器	0x0210	R/W
PMU_DATE_REG	版本控制寄存器	0x03FC	R/W

15.9 寄存器

本小节的所有地址均为相对于 PMU 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 15.1. PMU_HP_ACTIVE_DIG_POWER_REG (0x0000)



PMU_HP_ACTIVE_DCDC_SWITCH_PD_EN 配置是否在 HP_ACTIVE 状态下断开所有来源于外部 DCDC 的供电。

0：不断开

1：断开

(R/W)

PMU_HP_ACTIVE_HP_MEM_DSLP 配置是否在 HP_ACTIVE 状态下让 L2MEM_G0–G5 进入 Deep-sleep。

0：不进入 Deep-sleep

1：进入 Deep-sleep

(R/W)

PMU_HP_ACTIVE_PD_HP_MEM_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 L2MEM_G0–G5 电源域。

0：打开

1：关闭

(R/W)

PMU_HP_ACTIVE_PD_HP_CPU_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 HP_CPU 电源域。

0：打开

1：关闭

(R/W)

PMU_HP_ACTIVE_PD_CNNT_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 HP_CNNT 电源域。

0：打开

1：关闭

(R/W)

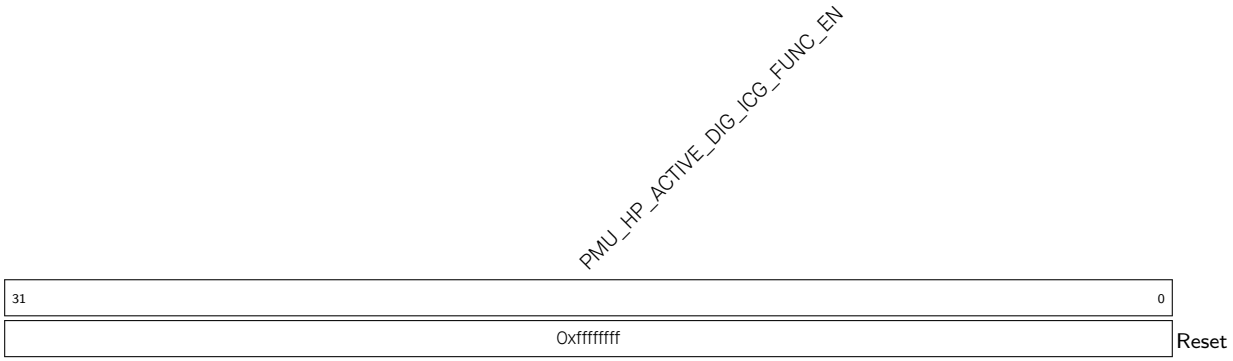
PMU_HP_ACTIVE_PD_TOP_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 PD_TOP 电源域。

0：打开

1：关闭

(R/W)

Register 15.2. PMU_HP_ACTIVE_ICG_HP_FUNC_REG (0x0004)



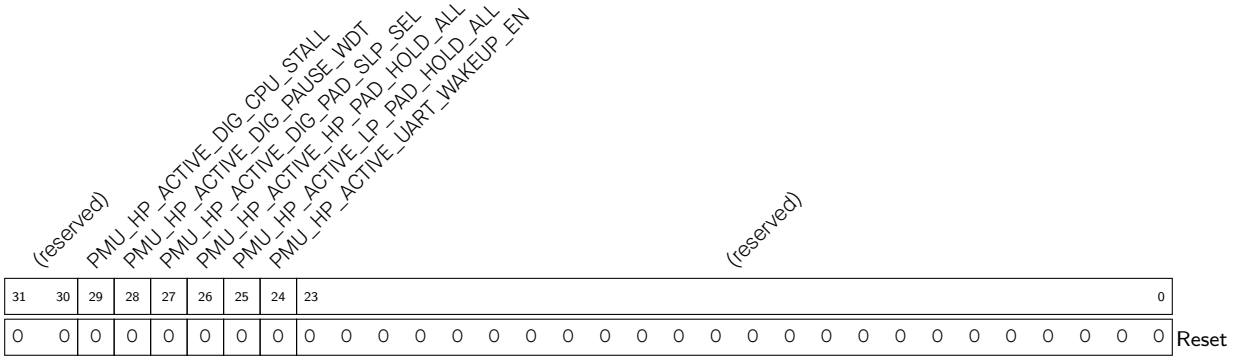
PMU_HP_ACTIVE_DIG_ICG_FUNC_EN 配置 HP_ACTIVE 状态下高性能系统外设功能时钟的开关。具体配置请见表 15.4-2。

0：关闭

1：打开

(R/W)

Register 15.3. PMU_HP_ACTIVE_HP_SYS_CNTL_REG (0x0010)



PMU_HP_ACTIVE_UART_WAKEUP_EN 配置是否在 HP_ACTIVE 状态下使能 UART 唤醒功能。

0：关闭唤醒功能

1：使能唤醒功能

(R/W)

PMU_HP_ACTIVE_LP_PAD_HOLD_ALL 配置是否在 HP_ACTIVE 状态下保持 LP GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_ACTIVE_HP_PAD_HOLD_ALL 配置是否在 HP_ACTIVE 状态下保持 GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_ACTIVE_DIG_PAD_SLP_SEL 配置是否在 HP_ACTIVE 状态下使用 GPIO 的 Light-sleep 模式配置。

0：正常工作模式的配置

1：Light-sleep 模式配置，详见章节 10 *GPIO 交换矩阵和 IO MUX* > 小节 10.8 *Light-sleep 模式管脚功能*。

(R/W)

PMU_HP_ACTIVE_DIG_PAUSE_WDT 配置是否在 HP_ACTIVE 状态下暂停看门狗。

0：不暂停

1：暂停

(R/W)

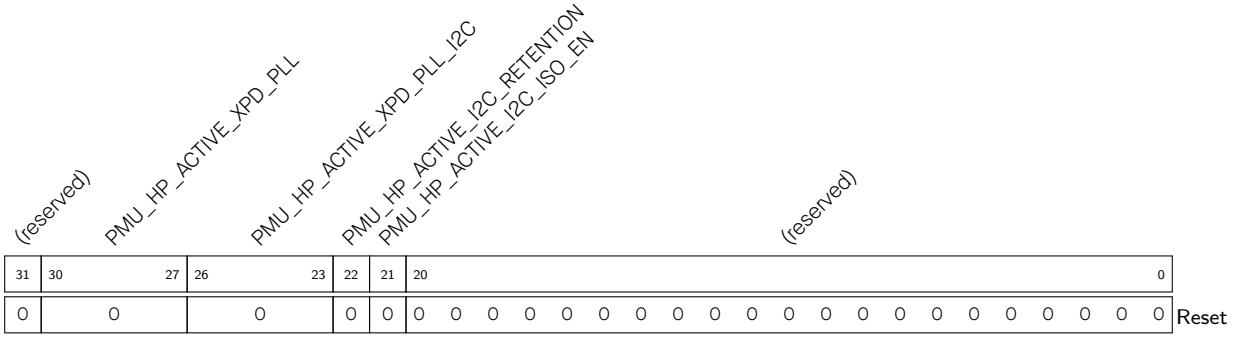
PMU_HP_ACTIVE_DIG_CPU_STALL 配置是否在 HP_ACTIVE 状态下暂停 CPU。

0：不暂停

1：暂停

(R/W)

Register 15.4. PMU_HP_ACTIVE_HP_CK_POWER_REG (0x0014)



PMU_HP_ACTIVE_I2C_ISO_EN 配置是否在 HP_ACTIVE 状态打开 ANALOG_I2C。

0：关闭

1：开启

(R/W)

PMU_HP_ACTIVE_I2C_RETENTION 配置是否在 HP_ACTIVE 状态使能 ANALOG_I2C 进入 retention 状态。

0：不使能

1：使能

(R/W)

PMU_HP_ACTIVE_XPD_PLL_I2C 配置是否在 HP_ACTIVE 状态下开启 PLL 的 I2C 控制器，该组寄存器为 bitmap，每个 bit 控制一个 PLL 的 I2C 寄存器组。

0：关闭

1：开启

(R/W)

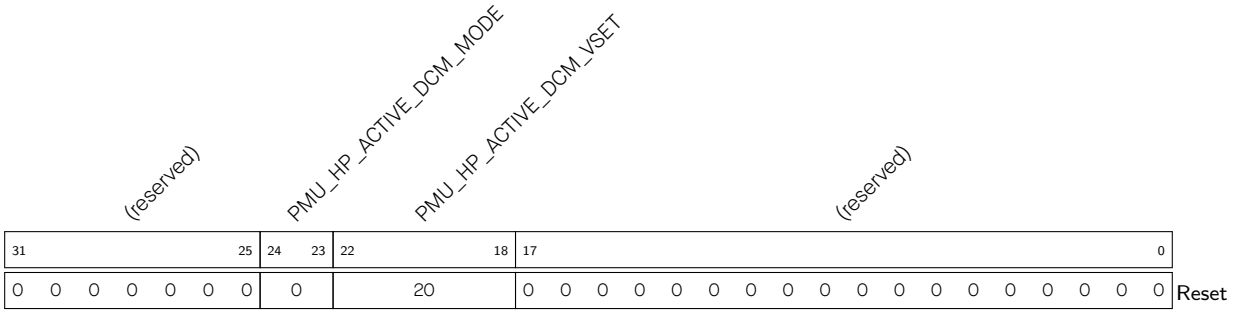
PMU_HP_ACTIVE_XPD_PLL 配置是否在 HP_ACTIVE 状态下开启 HP 系统的模拟电源域，每个 bit 控制一个电源域，bit 0 对应 CPLL_CLK，bit 1 对应 SPLL_CLK，bit 2 对应 Audio PLL，bit 3 对应 SDIO PLL。

0：关闭

1：开启

(R/W)

Register 15.5. PMU_HP_ACTIVE_BIAS_REG (0x0018)



PMU_HP_ACTIVE_DCM_VSET 在 HP_ACTIVE 状态下调节 DCDC 的电压。(R/W)

PMU_HP_ACTIVE_DCM_MODE 配置是否在 HP_ACTIVE 状态下使能 DCDC 的状态。

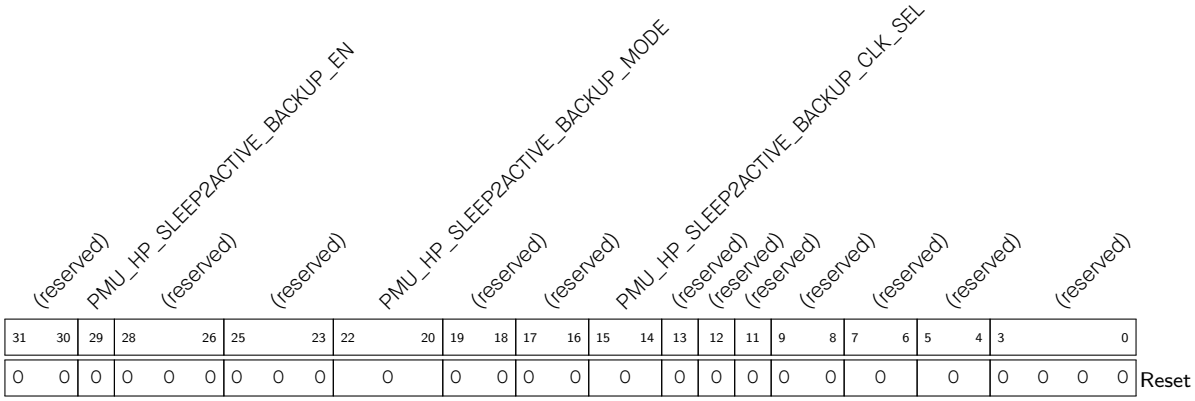
0：不使能

1：使能

2, 3：保留

(R/W)

Register 15.6. PMU_HP_ACTIVE_BACKUP_REG (0x001C)



PMU_HP_SLEEP2ACTIVE_BACKUP_CLK_SEL 配置备份模块从 HP_SLEEP 切换为 HP_ACTIVE 时的功能时钟源。

0：时钟源为 XTAL

1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

PMU_HP_SLEEP2ACTIVE_BACKUP_MODE 配置从 HP_SLEEP 切换为 HP_ACTIVE 时数据搬移的方向和链表指针。

最高位：

0：存储器到外设

1：外设到存储器

低两位：

0：PAU_LINK_ADDR_0

1：PAU_LINK_ADDR_1

2：PAU_LINK_ADDR_2

3：PAU_LINK_ADDR_3

(R/W)

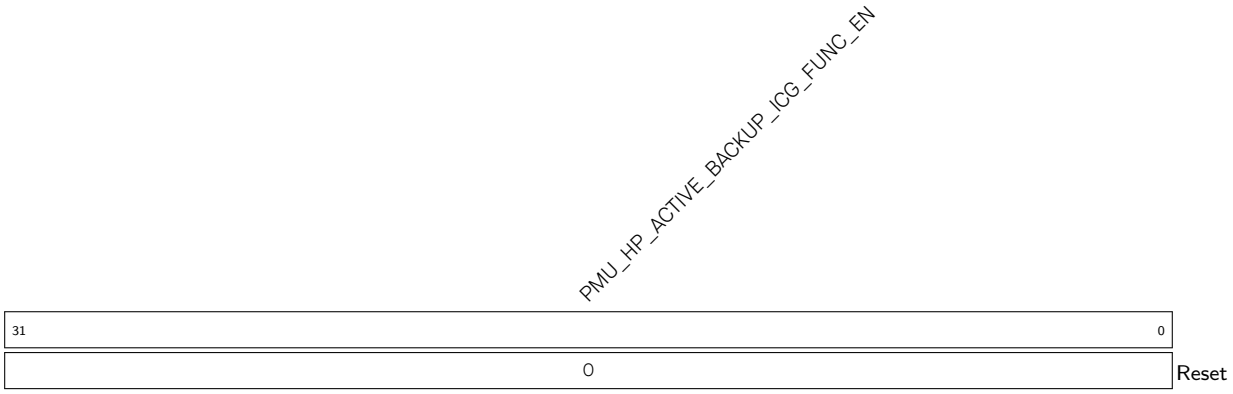
PMU_HP_SLEEP2ACTIVE_BACKUP_EN 配置是否在从 HP_SLEEP 切换为 HP_ACTIVE 时使能数据搬移。

0：禁能

1：使能

(R/W)

Register 15.7. PMU_HP_ACTIVE_BACKUP_CLK_REG (0x0020)



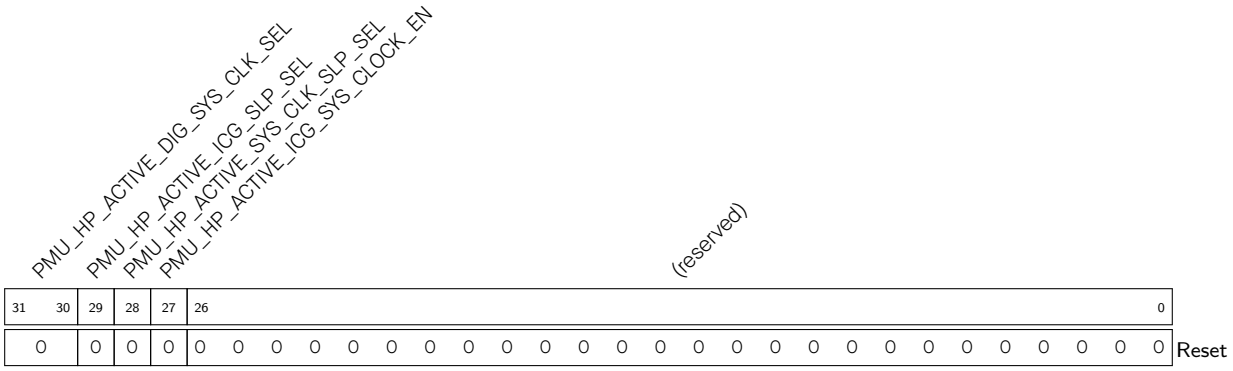
PMU_HP_ACTIVE_BACKUP_ICG_FUNC_EN 配置目标状态为 HP_ACTIVE 时不同外设功能时钟的开关情况，不同位代表不同外设，详见表 15.4-2。

0：关闭

1：打开

(R/W)

Register 15.8. PMU_HP_ACTIVE_SYSCLK_REG (0x0024)



PMU_HP_ACTIVE_ICG_SYS_CLOCK_EN 配置是否在 HP_ACTIVE 状态下打开 ROOT_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_ACTIVE_SYS_CLK_SLP_SEL 配置是否在 HP_ACTIVE 状态下由 PMU 控制时钟源。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_ACTIVE_ICG_SLP_SEL 配置是否在 HP_ACTIVE 状态下是否由 PMU 控制各时钟门控的开关。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_ACTIVE_DIG_SYS_CLK_SEL 配置 HP_ACTIVE 状态下 ROOT_CLK 的时钟源。

0：时钟源为 XTAL

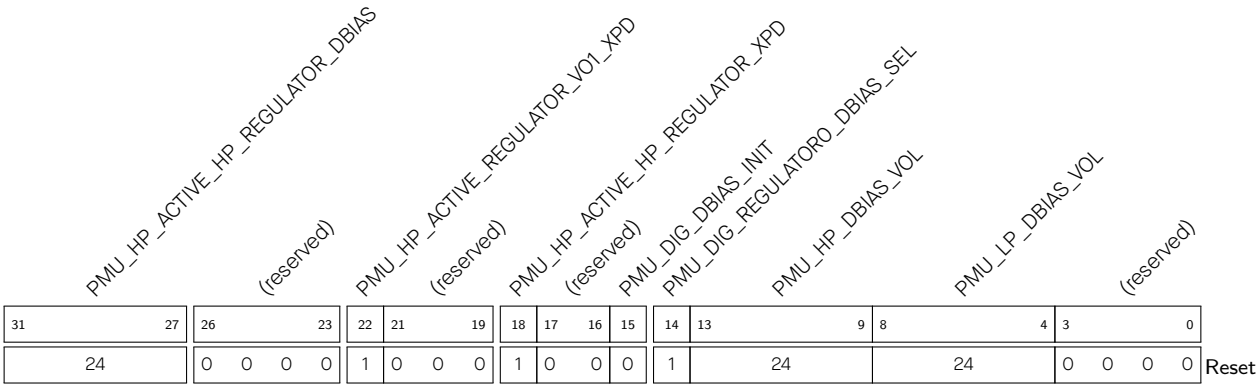
1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

Register 15.9. PMU_HP_ACTIVE_HP_REGULATOR0_REG (0x0028)



PMU_LP_DBIAS_VOL 表示当前的低功耗系统调压器的电压。(RO)

PMU_HP_DBIAS_VOL 表示当前的高性能系统调压器的电压。(RO)

PMU_DIG_REGULATOR0_DBIAS_SEL 选择高性能系统调压器的电压控制权。

0：硬件自动调压

1：软件调压

(R/W)

PMU_DIG_DBIAS_INIT 更新初始化 PVT 的电压配置。(WT)

PMU_HP_ACTIVE_HP_REGULATOR_XPD 配置是否在 HP_ACTIVE 状态下打开高性能系统调压器。

0：关闭高性能系统调压器

1：打开高性能系统调压器

(R/W)

PMU_HP_ACTIVE_REGULATOR_VO1_XPD 配置 HP_ACTIVE 状态下 VO1 的开关。

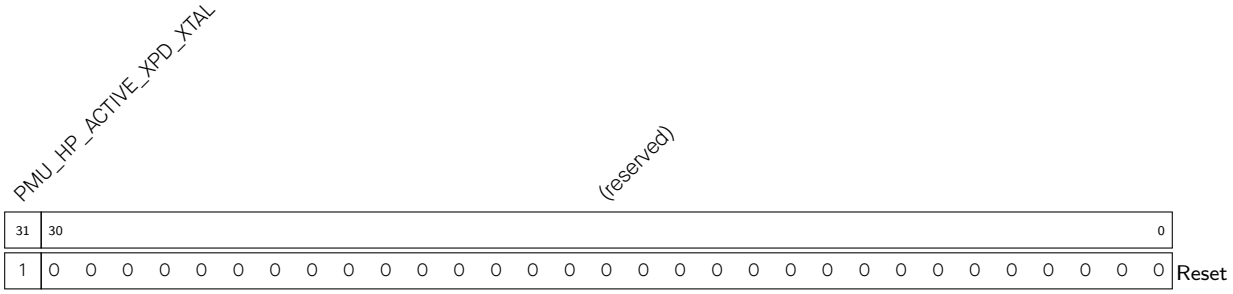
0：关闭

1：打开

(R/W)

PMU_HP_ACTIVE_HP_REGULATOR_DBIAS 在 HP_ACTIVE 状态下调节高性能系统调压器的电压，寄存器值越大，电压越高。(R/W)

Register 15.10. PMU_HP_ACTIVE_XTAL_REG (0x0030)



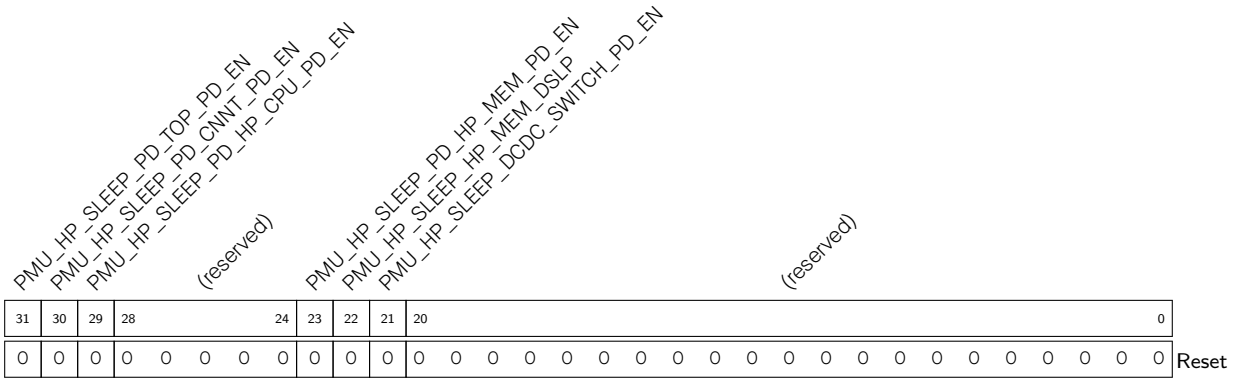
PMU_HP_ACTIVE_XPD_XTAL 配置是否在 HP_ACTIVE 状态下打开 XTAL_CLK 的模拟源。

0：关闭

1：打开

(R/W)

Register 15.11. PMU_HP_SLEEP_DIG_POWER_REG (0x0068)



PMU_HP_SLEEP_DCDC_SWITCH_PD_EN 配置是否在 HP_SLEEP 状态下断开所有来源于外部 DCDC 的供电。

0：不断开

1：断开

(R/W)

PMU_HP_SLEEP_HP_MEM_DSLP 配置是否在 HP_SLEEP 状态下让 L2MEM_G0–G5 进入 Deep-sleep。

0：不进入 Deep-sleep

1：进入 Deep-sleep

(R/W)

PMU_HP_SLEEP_PD_HP_MEM_PD_EN 配置是否在 HP_SLEEP 状态下关闭 L2MEM_G0–G5 电源域。

0：打开

1：关闭

(R/W)

PMU_HP_SLEEP_PD_HP_CPU_PD_EN 配置是否在 HP_SLEEP 状态下关闭 HP_CPU 电源域

0：打开

1：关闭

(R/W)

PMU_HP_SLEEP_PD_CNNT_PD_EN 配置是否在 HP_SLEEP 状态下关闭 HP_CNNT 电源域

0：打开

1：关闭

(R/W)

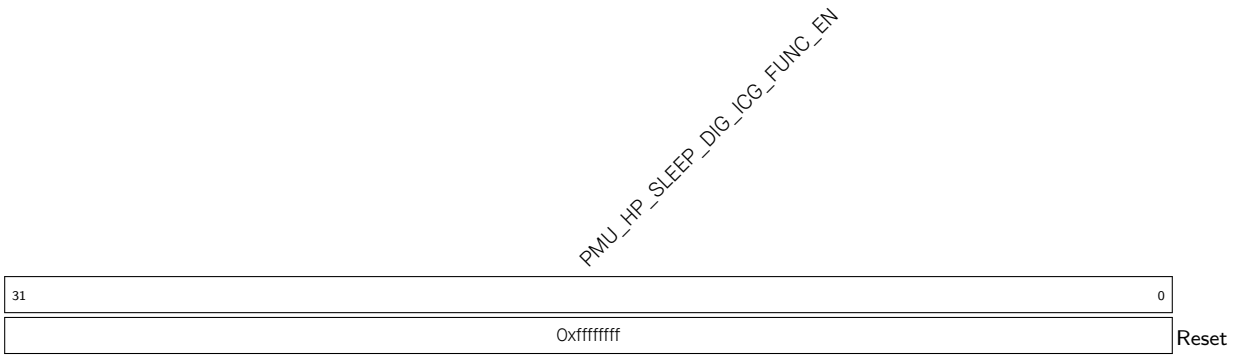
PMU_HP_SLEEP_PD_TOP_PD_EN 配置是否在 HP_SLEEP 状态下关闭 PD_TOP 电源域。

0：打开

1：关闭

(R/W)

Register 15.12. PMU_HP_SLEEP_ICG_HP_FUNC_REG (0x006C)



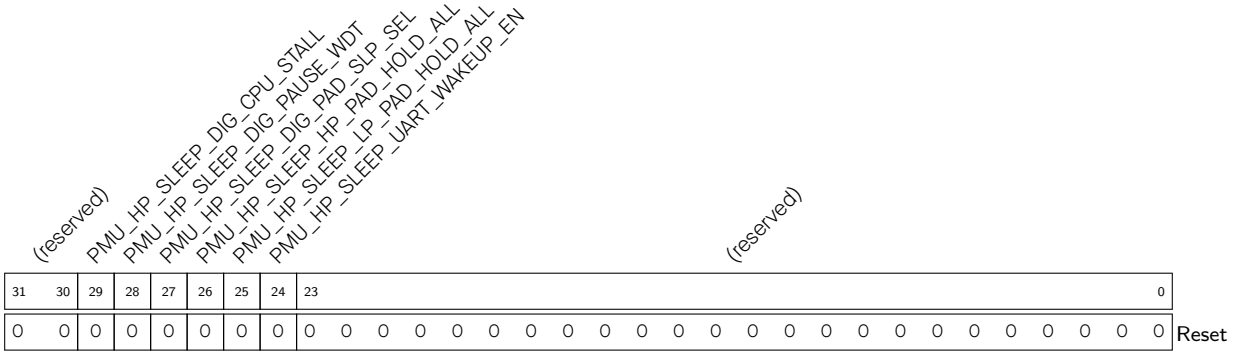
PMU_HP_SLEEP_DIG_ICG_FUNC_EN 配置 HP_SLEEP 状态下高性能系统外设功能时钟的开关。具体配置请见表 15.4-2。

0：关闭

1：打开

(R/W)

Register 15.13. PMU_HP_SLEEP_HP_SYS_CNTL_REG (0x0078)



PMU_HP_SLEEP_UART_WAKEUP_EN 配置是否在 HP_SLEEP 状态下使能 UART 唤醒功能。

0：关闭唤醒功能

1：使能唤醒功能

(R/W)

PMU_HP_SLEEP_LP_PAD_HOLD_ALL 配置是否在 HP_SLEEP 状态下保持 LP GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_SLEEP_HP_PAD_HOLD_ALL 配置是否在 HP_SLEEP 状态下保持 GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_SLEEP_DIG_PAD_SLP_SEL 配置是否在 HP_SLEEP 状态下使用 GPIO 的 Light-sleep 模式配置。

0：正常工作模式的配置

1：Light-sleep 模式配置，详见章节 10 *GPIO 交换矩阵和 IO MUX* > 小节 10.8 *Light-sleep 模式管脚功能*。

(R/W)

PMU_HP_SLEEP_DIG_PAUSE_WDT 配置是否在 HP_SLEEP 状态下暂停看门狗。

0：不暂停

1：暂停

(R/W)

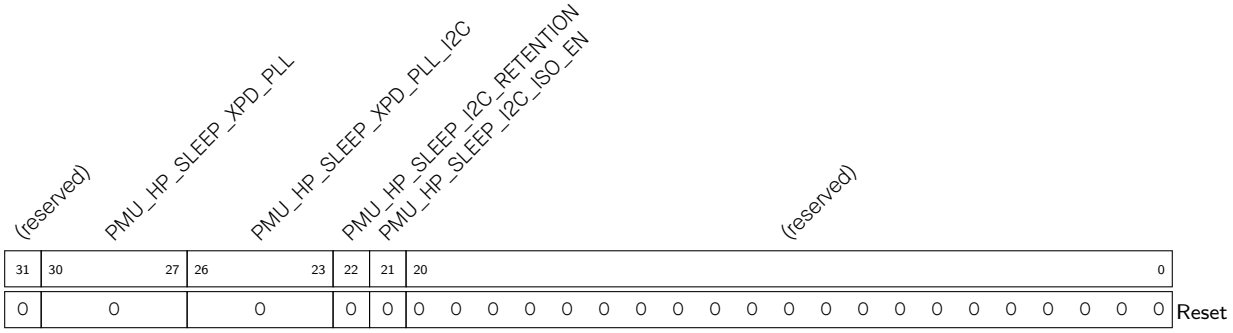
PMU_HP_SLEEP_DIG_CPU_STALL 配置是否在 HP_SLEEP 状态下暂停 CPU。

0：不暂停

1：暂停

(R/W)

Register 15.14. PMU_HP_SLEEP_HP_CK_POWER_REG (0x007C)



PMU_HP_SLEEP_I2C_ISO_EN 配置是否在 HP_SLEEP 状态打开 ANALOG_I2C。

0：关闭

1：开启

(R/W)

PMU_HP_SLEEP_I2C_RETENTION 配置是否在 HP_SLEEP 状态使能 ANALOG_I2C 进入 retention 状态。

0：不使能

1：使能

(R/W)

PMU_HP_SLEEP_XPD_PLL_I2C 配置是否在 HP_SLEEP 状态开启 PLL I2C 组，该组寄存器为 bitmap，每个位控制一个 PLL I2C。

0：关闭

1：开启

(R/W)

PMU_HP_SLEEP_XPD_PLL 配置是否在 HP_SLEEP 状态下开启 HP 系统的模拟电源域，每个 bit 控制一个电源域，bit 0 对应 CPLL_CLK，bit 1 对应 SPLL_CLK，bit 2 对应 Audio PLL，bit 3 对应 SDIO PLL。

0：关闭

1：开启

(R/W)

Register 15.15. PMU_HP_SLEEP_BIAS_REG (0x0080)



PMU_HP_SLEEP_DCM_VSET 调节 HP_SLEEP 下 DCDC 的电压。(R/W)

PMU_HP_SLEEP_DCM_MODE 配置是否在 HP_SLEEP 状态下使能 DCDC 的状态。

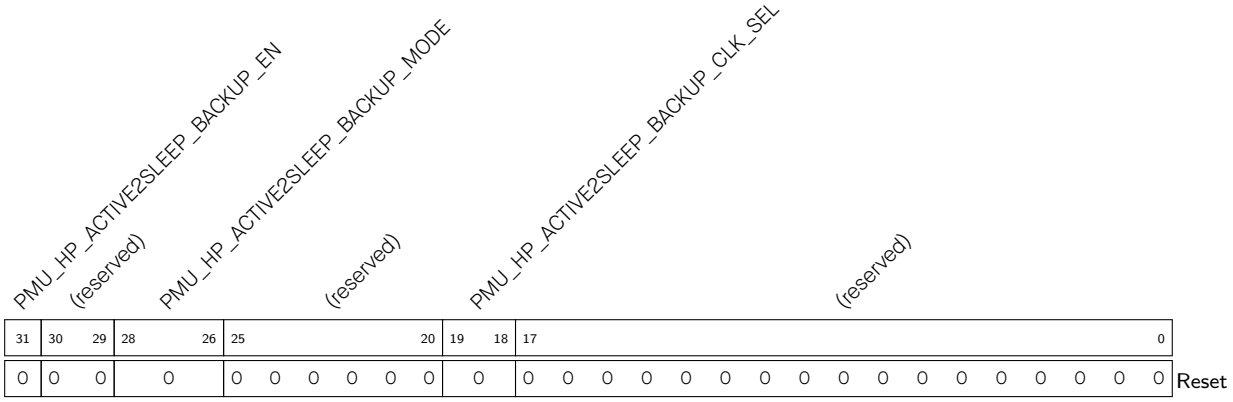
0：不使能

1：使能

2, 3：保留

(R/W)

Register 15.16. PMU_HP_SLEEP_BACKUP_REG (0x0084)



PMU_HP_ACTIVE2SLEEP_BACKUP_CLK_SEL 配置备份模块从 HP_ACTIVE 切换为 HP_SLEEP 时的功能时钟源。

0：时钟源为 XTAL

1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

PMU_HP_ACTIVE2SLEEP_BACKUP_MODE 配置从 HP_ACTIVE 切换为 HP_SLEEP 时数据搬移的方向和链表指针。

最高位：

0：存储器到外设

1：外设到存储器

低两位：

0：PAU_LINK_ADDR_0

1：PAU_LINK_ADDR_1

2：PAU_LINK_ADDR_2

3：PAU_LINK_ADDR_3

(R/W)

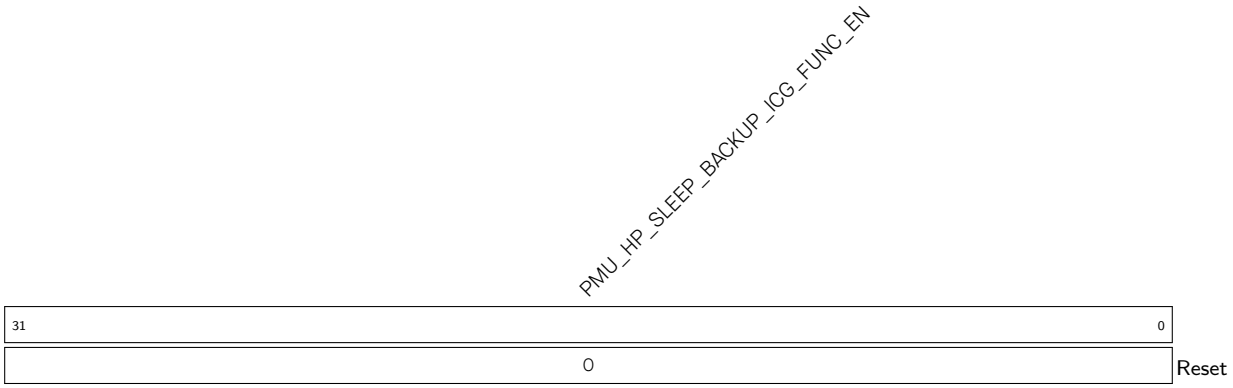
PMU_HP_ACTIVE2SLEEP_BACKUP_EN 配置是否在从 HP_ACTIVE 切换为 HP_SLEEP 时使能数据搬移。

0：禁能

1：使能

(R/W)

Register 15.17. PMU_HP_SLEEP_BACKUP_CLK_REG (0x0088)



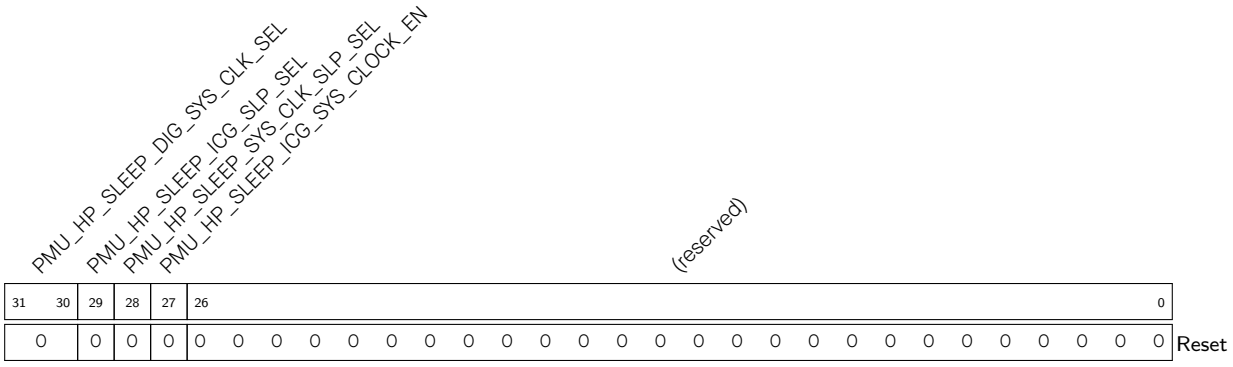
PMU_HP_SLEEP_BACKUP_ICG_FUNC_EN 配置目标状态为 HP_SLEEP 时不同外设功能时钟的开关情况，不同位代表不同外设，详见表 15.4-2。

0：关闭

1：打开

(R/W)

Register 15.18. PMU_HP_SLEEP_SYSCLK_REG (0x008C)



PMU_HP_SLEEP_ICG_SYS_CLOCK_EN 配置是否在 HP_SLEEP 状态下打开 ROOT_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_SLEEP_SYS_CLK_SLP_SEL 配置是否在 HP_SLEEP 状态下由 PMU 控制时钟源。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_SLEEP_ICG_SLP_SEL 配置是否在 HP_SLEEP 状态下是否由 PMU 控制各时钟门控的开关。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_SLEEP_DIG_SYS_CLK_SEL 配置 HP_SLEEP 状态下 ROOT_CLK 的时钟源。

0：时钟源为 XTAL

1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

Register 15.19. PMU_HP_SLEEP_HP_REGULATOR0_REG (0x0090)



PMU_HP_SLEEP_HP_REGULATOR_XPD 配置是否在 HP_SLEEP 状态下打开 HP 系统调压器。

0：关闭 HP 系统调压器

1：打开 HP 系统调压器

(R/W)

PMU_HP_SLEEP_REGULATOR_VO1_XPD 配置 HP_SLEEP 状态下 VO1 的开关。

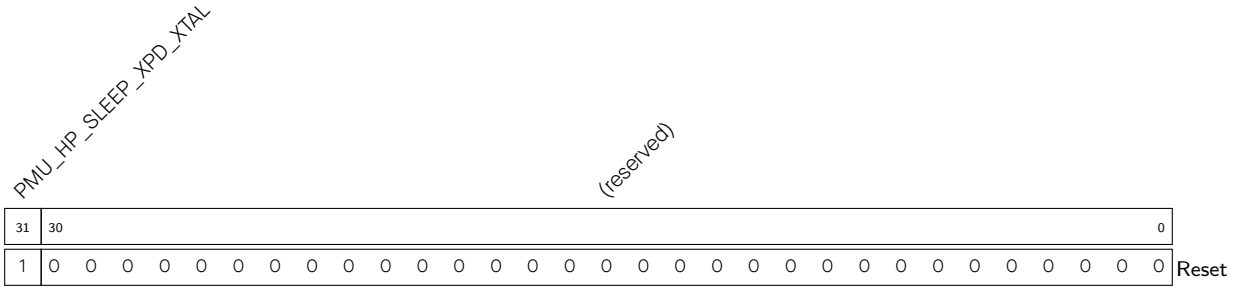
0：关闭

1：打开

(R/W)

PMU_HP_SLEEP_HP_REGULATOR_DBIAS 在 HP_SLEEP 状态下调节 HP 系统调压器的电压，寄存器值越大，电压越高。(R/W)

Register 15.20. PMU_HP_SLEEP_XTAL_REG (0x0098)



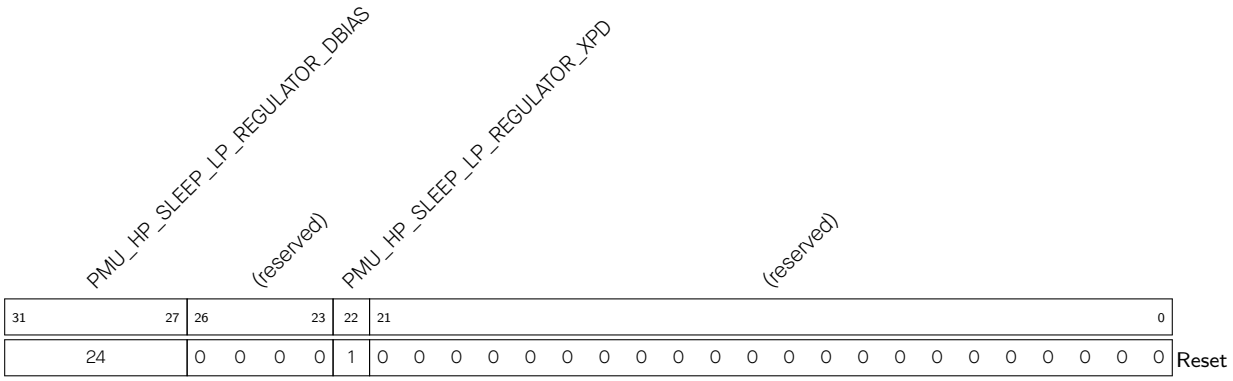
PMU_HP_SLEEP_XPD_XTAL 配置是否在 HP_SLEEP 状态下打开 XTAL_CLK 的模拟源。

0：关闭

1：打开

(R/W)

Register 15.21. PMU_HP_SLEEP_LP_REGULATOR0_REG (0x009C)



PMU_HP_SLEEP_LP_REGULATOR_XPD 配置是否在 HP_ACTIVE/HP_SLEEP 状态下打开 HP 系统调压器。

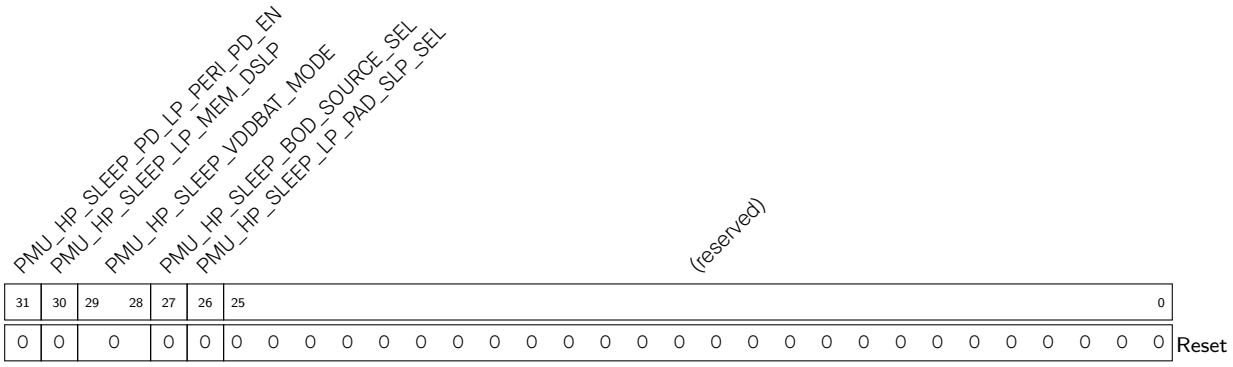
0：关闭 LP 系统调压器

1：打开 LP 系统调压器

(R/W)

PMU_HP_SLEEP_LP_REGULATOR_DBIAS 在 HP_ACTIVE/HP_SLEEP 状态下调节 LP 调压器的电压，寄存器值越大，电压越高。(R/W)

Register 15.22. PMU_HP_SLEEP_LP_DIG_POWER_REG (0x00A8)



PMU_HP_SLEEP_LP_PAD_SLP_SEL 配置是否在 HP_ACTIVE/HP_SLEEP 状态下使能 LP GPIO 功能。

0：不使能

1：使能

(R/W)

PMU_HP_SLEEP_BOD_SOURCE_SEL 配置在 HP_ACTIVE/HP_SLEEP 状态下使能欠压检测器。

(R/W)

PMU_HP_SLEEP_VDDBAT_MODE 配置是否在 HP_ACTIVE/HP_SLEEP 状态下使能 VBAT 电源。

0：不使能

1：使能

(R/W)

PMU_HP_SLEEP_LP_MEM_DSLP 配置是否在 HP_ACTIVE/HP_SLEEP 状态下使低功耗管理系统的 LP SRAM 进入 Memory Deep-sleep 模式。

0：不使能

1：使能

(R/W)

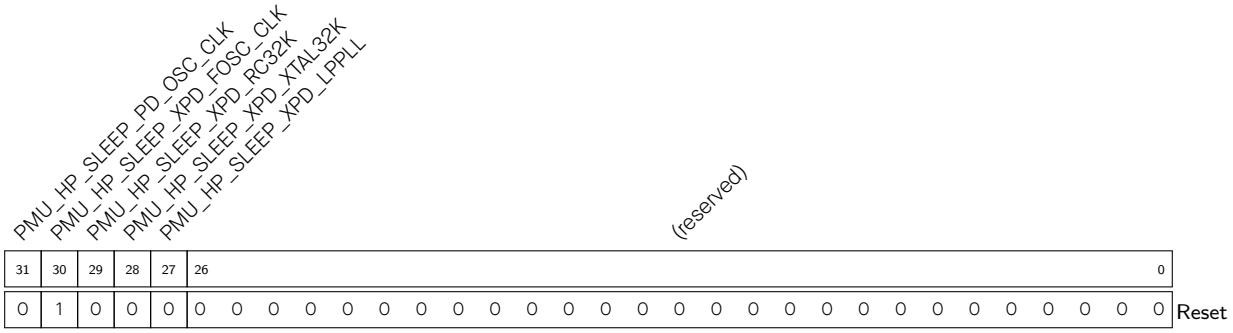
PMU_HP_SLEEP_PD_LP_PERI_PD_EN 配置是否在 HP_ACTIVE/HP_SLEEP 状态下关闭 LP PD Peripherals 电源域。

0：打开

1：关闭

(R/W)

Register 15.23. PMU_HP_SLEEP_LP_CK_POWER_REG (0x00AC)



PMU_HP_SLEEP_XPD_LPPLL 配置是否在 HP_ACTIVE/HP_SLEEP 状态下打开 PLL_LP_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_SLEEP_XPD_XTAL32K 配置是否在 HP_ACTIVE/HP_SLEEP 状态下打开 XTAL32K_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_SLEEP_XPD_RC32K 配置是否在 HP_ACTIVE/HP_SLEEP 状态下打开 RC32K_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_SLEEP_XPD_FOSC_CLK 配置是否在 HP_ACTIVE/HP_SLEEP 状态下打开 RC_FAST_CLK。

0：关闭

1：打开

(R/W)

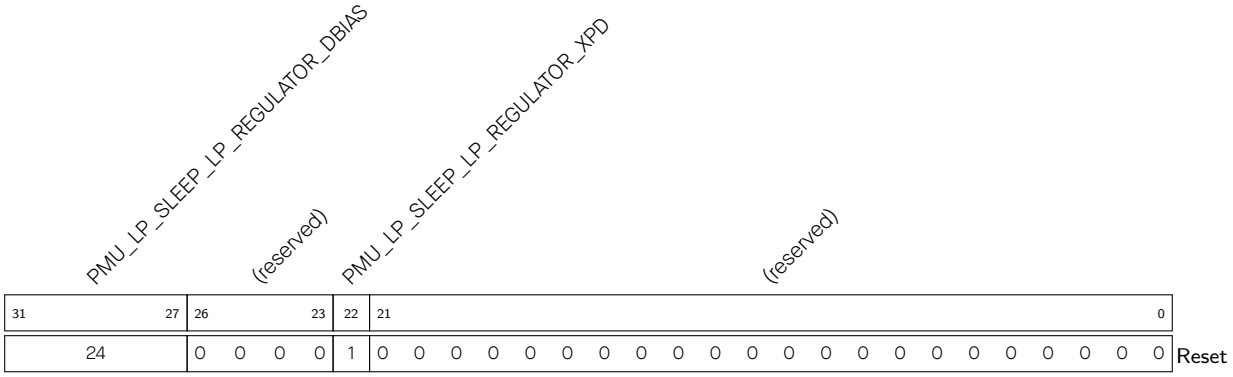
PMU_HP_SLEEP_PD_OSC_CLK 配置是否在 HP_ACTIVE/HP_SLEEP 状态下打开 RC_SLOW_CLK。

0：关闭

1：打开

(R/W)

Register 15.24. PMU_LP_SLEEP_LP_REGULATOR0_REG (0x00B4)



PMU_LP_SLEEP_LP_REGULATOR_XPD 配置是否在 LP_SLEEP 状态下使能 LP 系统调压器。

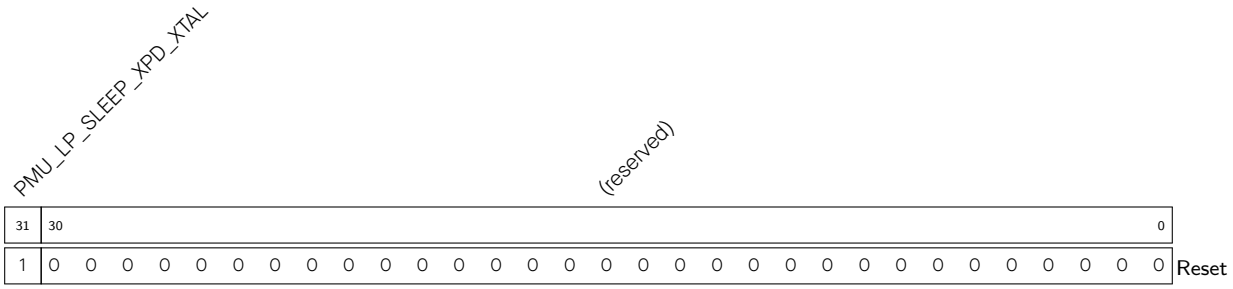
0：不使能

1：使能

(R/W)

PMU_LP_SLEEP_LP_REGULATOR_DBIAS 在 LP_SLEEP 状态下调节 LP 系统调压器的电压，寄存器值越大，电压越高。(R/W)

Register 15.25. PMU_LP_SLEEP_XTAL_REG (0x00BC)



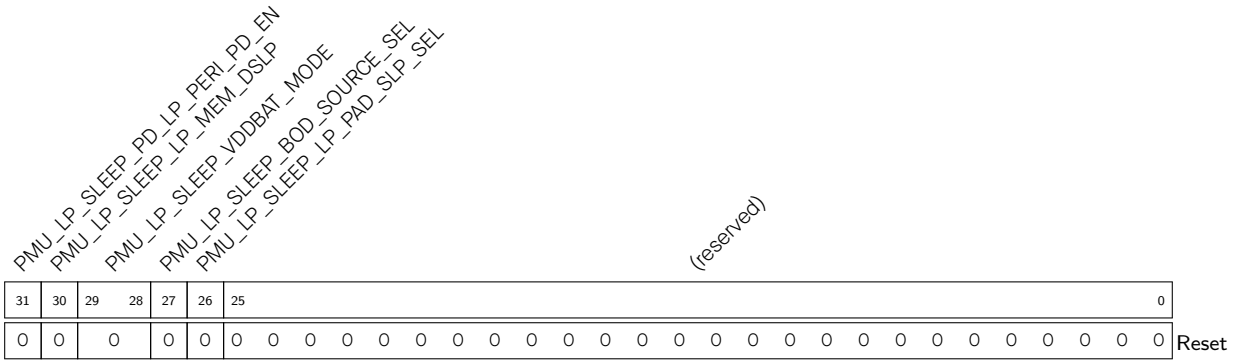
PMU_LP_SLEEP_XPD_XTAL 配置是否在 LP_SLEEP 状态下打开 XTAL_CLK 的模拟源。

0：关闭

1：打开

(R/W)

Register 15.26. PMU_LP_SLEEP_LP_DIG_POWER_REG (0x00C0)



PMU_LP_SLEEP_LP_PAD_SLP_SEL 配置是否在 LP_SLEEP 状态下使能 LP GPIO 功能。

0：不使能

1：使能

(R/W)

PMU_LP_SLEEP_BOD_SOURCE_SEL 配置在 LP_SLEEP 状态下使能欠压检测器。

(R/W)

PMU_LP_SLEEP_VDDBAT_MODE 配置是否在 LP_SLEEP 状态下使能 VBAT 电源。

0：不使能

1：使能

(R/W)

PMU_LP_SLEEP_LP_MEM_DSLP 配置是否在 LP_SLEEP 状态下使低功耗管理系统的 LP SRAM 进入 Memory Deep-sleep 模式。

0：不使能

1：使能

(R/W)

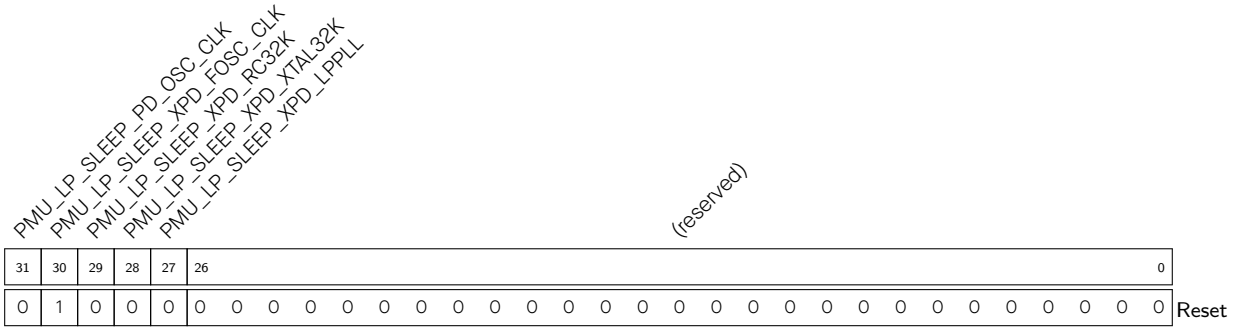
PMU_LP_SLEEP_PD_LP_PERI_PD_EN 配置是否在 LP_SLEEP 状态下关闭 LP PD Peripherals 电源域。

0：打开

1：关闭

(R/W)

Register 15.27. PMU_LP_SLEEP_LP_CK_POWER_REG (0x00C4)



PMU_LP_SLEEP_XPD_LPPLL 配置是否在 LP_SLEEP 状态下打开 PLL_LP_CLK。

0：关闭

1：打开

(R/W)

PMU_LP_SLEEP_XPD_XTAL32K 配置是否在 LP_SLEEP 状态下打开 XTAL32K_CLK。

0：关闭

1：打开

(R/W)

PMU_LP_SLEEP_XPD_RC32K 配置是否在 LP_SLEEP 状态下打开 RC32K_CLK。

0：关闭

1：打开

(R/W)

PMU_LP_SLEEP_XPD_FOSC_CLK 配置是否在 LP_SLEEP 状态下打开 RC_FAST_CLK。

0：关闭

1：打开

(R/W)

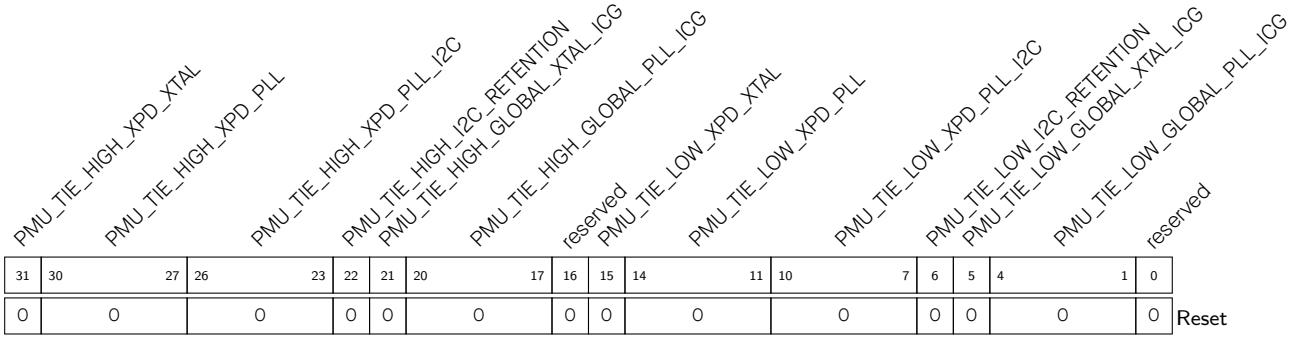
PMU_LP_SLEEP_PD_OSC_CLK 配置是否在 LP_SLEEP 状态下关闭 RC_SLOW_CLK。

0：打开

1：关闭

(R/W)

Register 15.28. PMU_IMM_HP_CK_POWER_REG (0x00CC)



PMU_TIE_LOW_GLOBAL_PLL_ICG 配置关闭全局 PLL 组的时钟门控位图。

0：无操作

1：关闭

(WT)

PMU_TIE_LOW_GLOBAL_XTAL_ICG 配置关闭全局 XTAL 的时钟门控。

0：无操作

1：关闭

(WT)

PMU_TIE_LOW_I2C_RETENTION 配置关闭 ANALOG I2C retention。

0：无操作

1：关闭

(WT)

PMU_TIE_LOW_XPD_PLL_I2C 配置关闭全局 PLL I2C 组的时钟电源位图。

0：无操作

1：关闭

(WT)

PMU_TIE_LOW_XPD_PLL 配置关闭全局 PLL 组的时钟电源位图。

0：无操作

1：关闭

(WT)

PMU_TIE_LOW_XPD_XTAL 配置关闭全局 XTAL 的时钟电源。

0：无操作

1：关闭

(WT)

PMU_TIE_HIGH_GLOBAL_PLL_ICG 配置开启全局 PLL 组的时钟门控位图。

0：无操作

1：开启

(WT)

见下页...

Register 15.28. PMU_IMM_HP_CK_POWER_REG (0x00CC)

接上页...

PMU_TIE_HIGH_GLOBAL_XTAL_ICG 配置开启全局 XTAL 的时钟门控。

0：无操作

1：开启

(WT)

PMU_TIE_HIGH_I2C_RETENTION 配置开启 ANALOG I2C retention。

0：无操作

1：开启

(WT)

PMU_TIE_HIGH_XPD_PLL_I2C 配置开启全局 PLL I2C 组的时钟电源位图。

0：无操作

1：开启

(WT)

PMU_TIE_HIGH_XPD_PLL 配置开启全局 PLL 组的时钟电源位图。

0：无操作

1：开启

(WT)

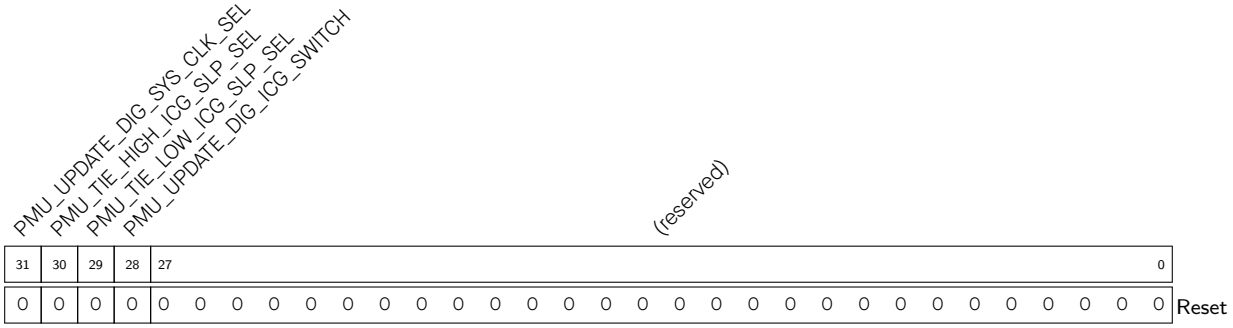
PMU_TIE_HIGH_XPD_XTAL 配置开启全局 XTAL 的时钟电源。

0：无操作

1：开启

(WT)

Register 15.29. PMU_IMM_SLEEP_SYSCLK_REG (0x00D0)



PMU_UPDATE_DIG_ICG_SWITCH 立即更新功能时钟门控状态切换。(WT)

PMU_TIE_LOW_ICG_SLP_SEL 配置关闭功能模块时钟门控的 PMU 控制。

0：无操作

1：关闭

(WT)

PMU_TIE_HIGH_ICG_SLP_SEL 配置开启功能模块时钟门控的 PMU 控制。

0：无操作

1：开启

(WT)

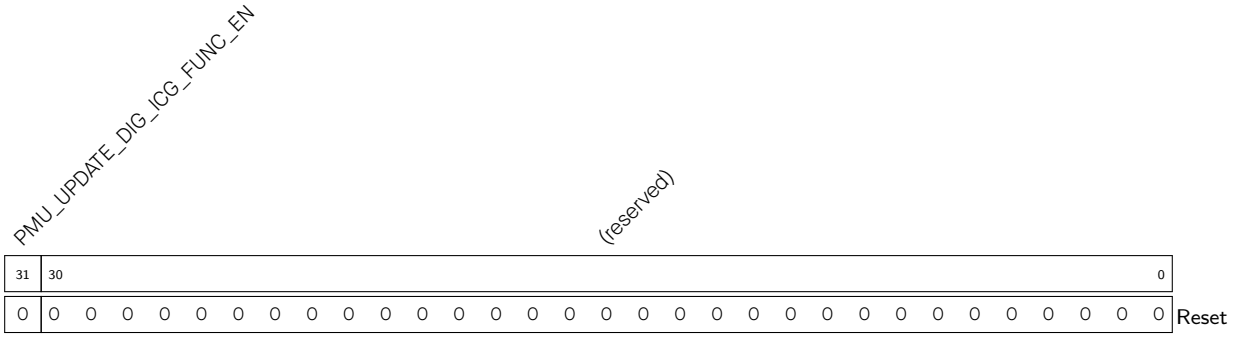
PMU_UPDATE_DIG_SYS_CLK_SEL 立即更新系统时钟选择。

0：无操作

1：更新当前 PMU 状态的 SYS_CLK_SEL 配置到使用端

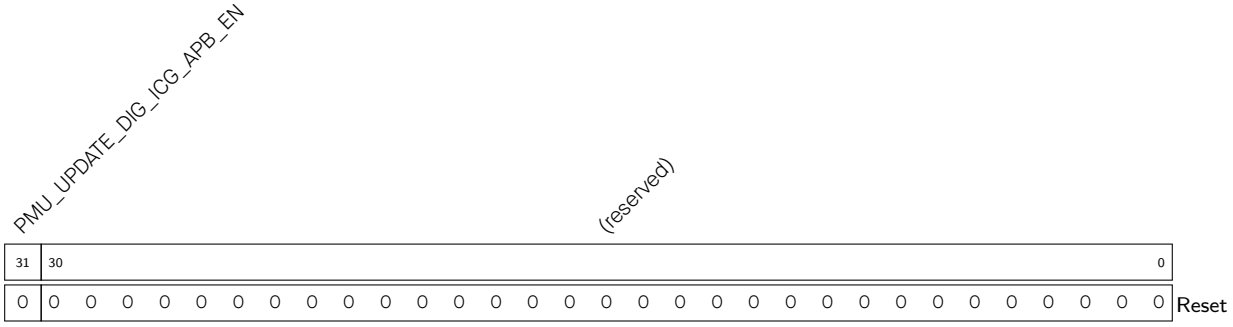
(WT)

Register 15.30. PMU_IMM_HP_FUNC_ICG_REG (0x00D4)



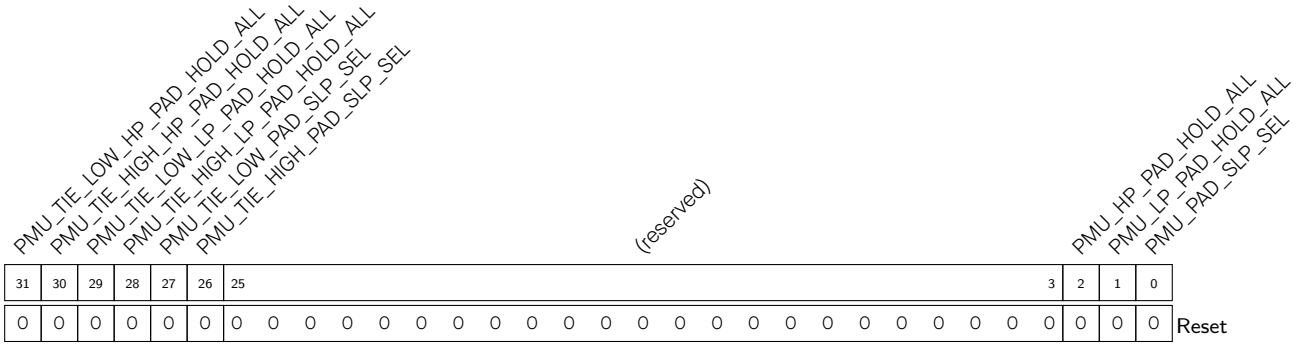
PMU_UPDATE_DIG_ICG_FUNC_EN 立即更新功能模块时钟门控的位图。(WT)

Register 15.31. PMU_IMM_HP_APB_ICG_REG (0x00D8)



PMU_UPDATE_DIG_ICG_APB_EN 立即更新 APB 时钟门控的位图。(WT)

Register 15.32. PMU_IMM_PAD_HOLD_ALL_REG (0x00E4)



PMU_PAD_SLP_SEL 表示当前 GPIO 的功能是否使用睡眠的配置。

0：不是

1：是

(RO)

PMU_LP_PAD_HOLD_ALL 表示当前 LP PAD 是否处于 HOLD 状态。

0：不是

1：是

(RO)

PMU_HP_PAD_HOLD_ALL 表示当前 HP PAD 是否处于 HOLD 状态。

0：不是

1：是

(RO)

PMU_TIE_HIGH_PAD_SLP_SEL 立即配置 GPIO 功能使用睡眠配置。(WT)

PMU_TIE_LOW_PAD_SLP_SEL 立即解除 GPIO 功能使用睡眠配置。(WT)

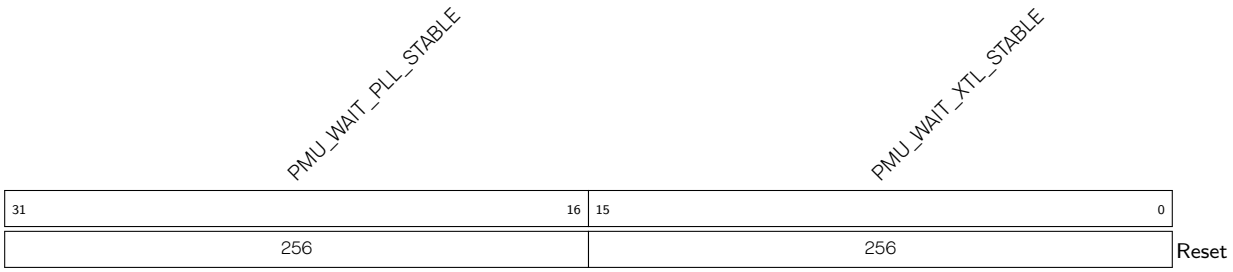
PMU_TIE_HIGH_LP_PAD_HOLD_ALL 立即配置 LP PAD 进入 HOLD 状态。(WT)

PMU_TIE_LOW_LP_PAD_HOLD_ALL 立即解除 LP PAD 进入 HOLD 状态。(WT)

PMU_TIE_HIGH_HP_PAD_HOLD_ALL 立即配置 HP PAD 进入 HOLD 状态。(WT)

PMU_TIE_LOW_HP_PAD_HOLD_ALL 立即解除 HP PAD 进入 HOLD 状态。(WT)

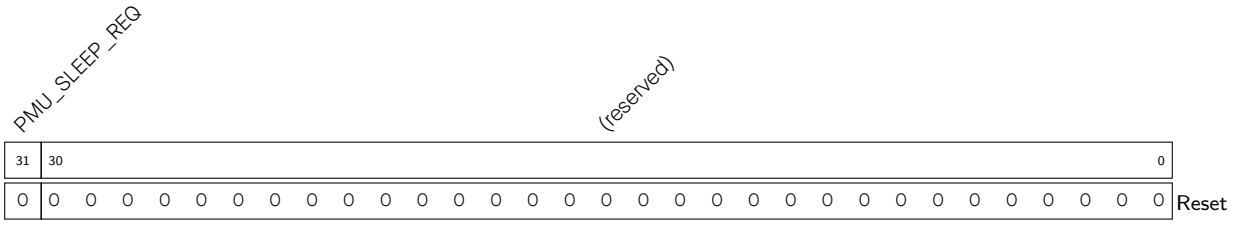
Register 15.33. PMU_POWER_CK_WAIT_CNTL_REG (0x011C)



PMU_WAIT_XTL_STABLE 配置开启 XTAL 电源后，开启 XTAL 全局时钟门控的等待时间。单位是 LP_DYN_FAST_CLK。(R/W)

PMU_WAIT_PLL_STABLE 配置开启 PLL 电源后，开启 PLL 全局时钟门控的等待时间。单位是 LP_DYN_FAST_CLK。(R/W)

Register 15.34. PMU_SLP_WAKEUP_CNTL0_REG (0x0120)



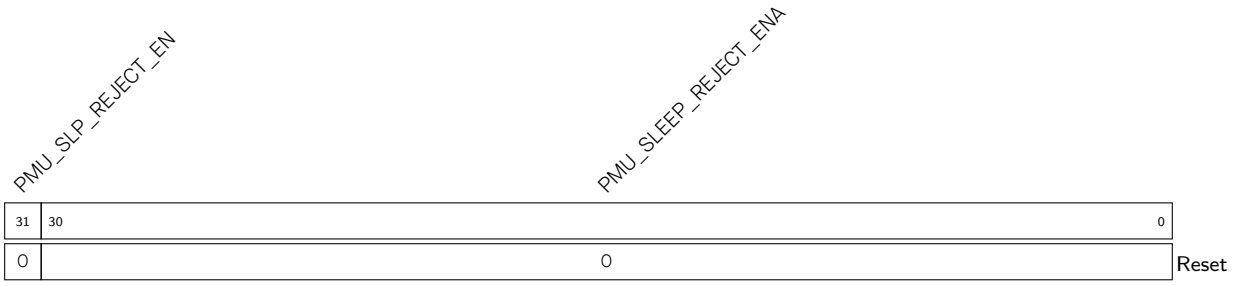
PMU_SLEEP_REQ 配置是否将芯片的 PMU 状态切换至 HP_SLEEP 或 LP_SLEEP。

0：不切换

1：切换，目标状态取决于 LP CPU 的工作状态。

(WT)

Register 15.35. PMU_SLP_WAKEUP_CNTL1_REG (0x0124)



PMU_SLEEP_REJECT_ENA 配置拒绝睡眠信号源。配置值与拒绝睡眠信号源的对应关系见表 15.4-1。(R/W)

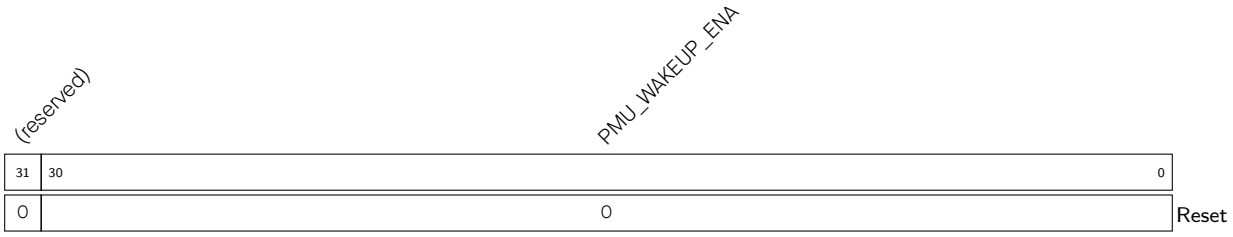
PMU_SLP_REJECT_EN 配置是否使能拒绝睡眠功能。

0：禁能

1：使能

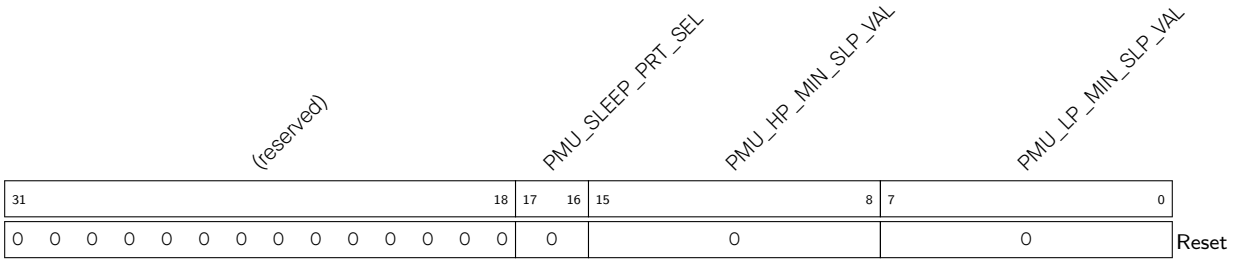
(R/W)

Register 15.36. PMU_SLP_WAKEUP_CNTL2_REG (0x0128)



PMU_WAKEUP_ENA 配置唤醒源。配置值与唤醒源的对应关系见表 15.4-1。(R/W)

Register 15.37. PMU_SLP_WAKEUP_CNTL3_REG (0x012C)



PMU_LP_MIN_SLP_VAL 配置进入 LP_SLEEP 的最小睡眠时间。单位是 LP_DYN_SLOW_CLK。(R/W)

PMU_HP_MIN_SLP_VAL 配置进入 HP_SLEEP 的最小睡眠时间。单位是 LP_DYN_SLOW_CLK。(R/W)

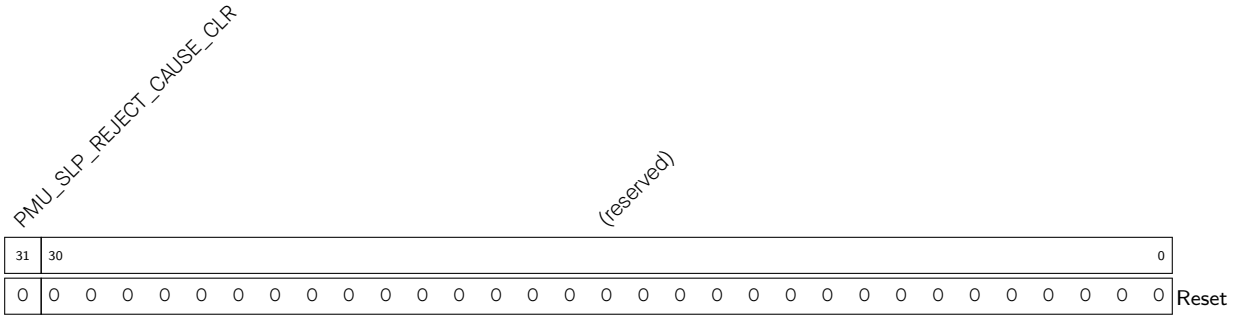
PMU_SLEEP_PRT_SEL 配置最小睡眠时间模式。

0, 1, 3：保留

2：同时保护 LP_SLEEP 和 HP_SLEEP

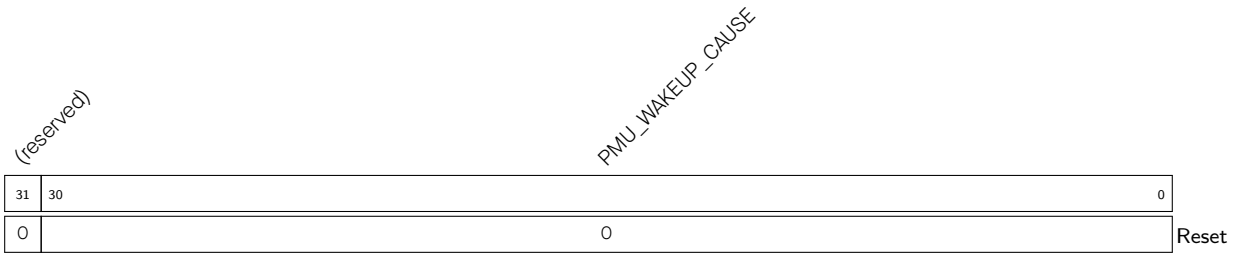
(R/W)

Register 15.38. PMU_SLP_WAKEUP_CNTL4_REG (0x0130)



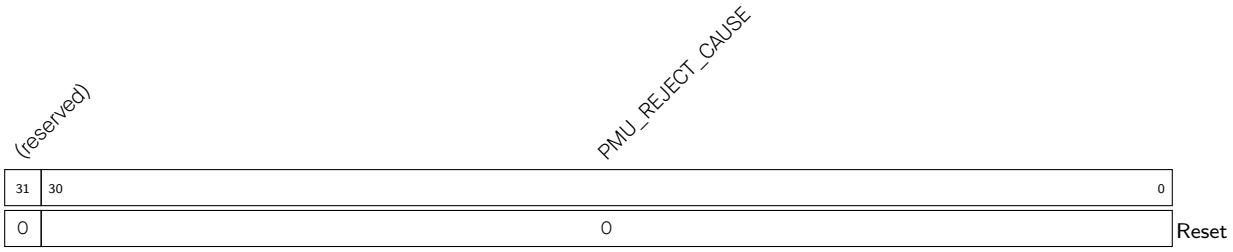
PMU_SLP_REJECT_CAUSE_CLR 写 1 清除 PMU_REJECT_CAUSE。(WT)

Register 15.39. PMU_SLP_WAKEUP_STATUS0_REG (0x0144)



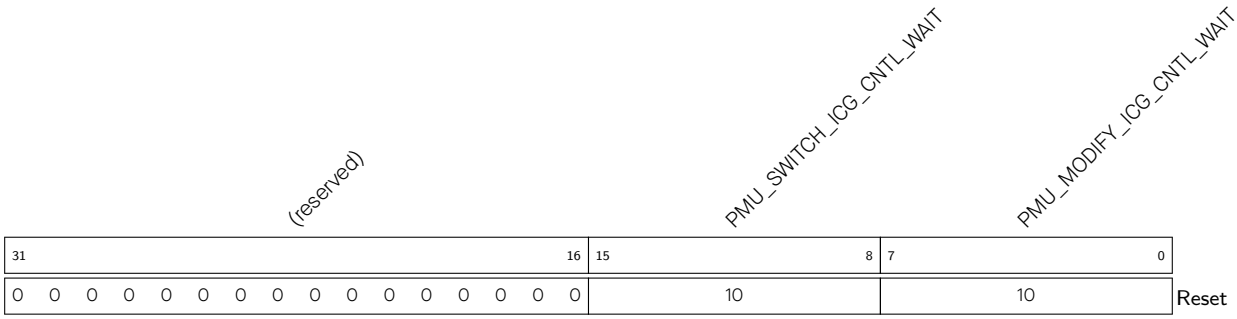
PMU_WAKEUP_CAUSE 指示唤醒源。配置值与唤醒源的对应关系请见表 15.4-1。(RO)

Register 15.40. PMU_SLP_WAKEUP_STATUS1_REG (0x0148)



PMU_REJECT_CAUSE 指示拒绝唤醒的事件源。配置值与唤醒源的对应关系请见表 15.4-1。(RO)

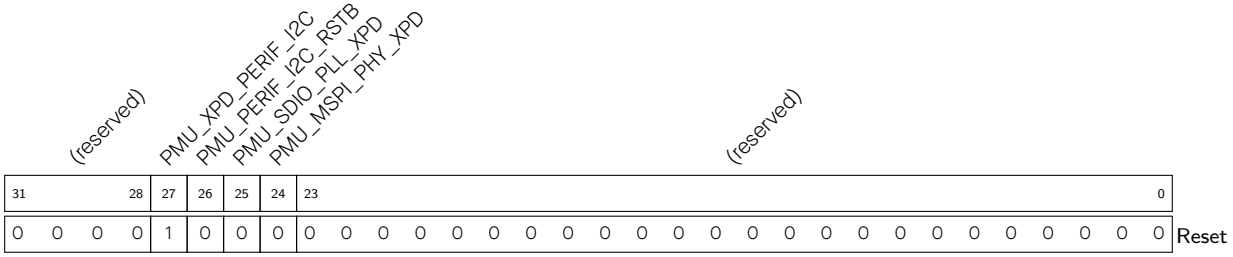
Register 15.41. PMU_HP_CK_CNTL_REG (0x0154)



PMU_MODIFY_ICG_CNTL_WAIT 配置修改功能模块时钟门控的等待时间。单位是 LP_DYN_FAST_CLK。(R/W)

PMU_SWITCH_ICG_CNTL_WAIT 配置切换功能模块时钟门控的等待时间。单位是 LP_DYN_FAST_CLK。(R/W)

Register 15.42. PMU_RF_PWC_REG (0x015C)



PMU_MSPI_PHY_XPD 配置是否开启 MSPI PHY。

0：关闭

1：开启

(R/W)

PMU_SDIO_PLL_XPD 配置是否开启 SDIO PLL。

0：关闭

1：开启

(R/W)

PMU_PERIF_I2C_RSTB 配置是否使能 PERI I2C 的复位。

0：复位

1：释放

(R/W)

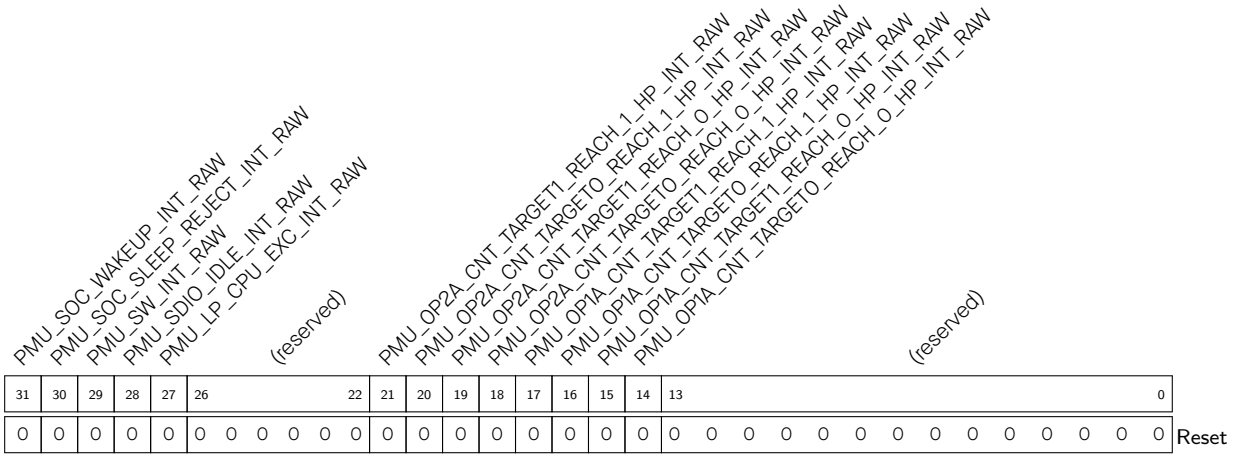
PMU_XPD_PERIF_I2C 配置是否开启 PERI I2C 电源。

0：关闭

1：开启

(R/W)

Register 15.43. PMU_INT_RAW_REG (0x0164)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_RAW PMU_OP1A_CNT_TARGET0_REACH_0_INT 的原始中断状态。(R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_RAW PMU_OP1A_CNT_TARGET1_REACH_0_INT 的原始中断状态。(R/WTC/SS)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_RAW PMU_OP1A_CNT_TARGET0_REACH_1_INT 的原始中断状态。(R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_RAW PMU_OP1A_CNT_TARGET1_REACH_1_INT 的原始中断状态。(R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_RAW PMU_OP2A_CNT_TARGET0_REACH_0_INT 的原始中断状态。(R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_RAW PMU_OP2A_CNT_TARGET1_REACH_0_INT 的原始中断状态。(R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_RAW PMU_OP2A_CNT_TARGET0_REACH_1_INT 的原始中断状态。(R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_RAW PMU_OP2A_CNT_TARGET1_REACH_1_INT 的原始中断状态。(R/WTC/SS)

PMU_LP_CPU_EXC_INT_RAW PMU_LP_CPU_EXC_INT 的原始中断状态。(R/WTC/SS)

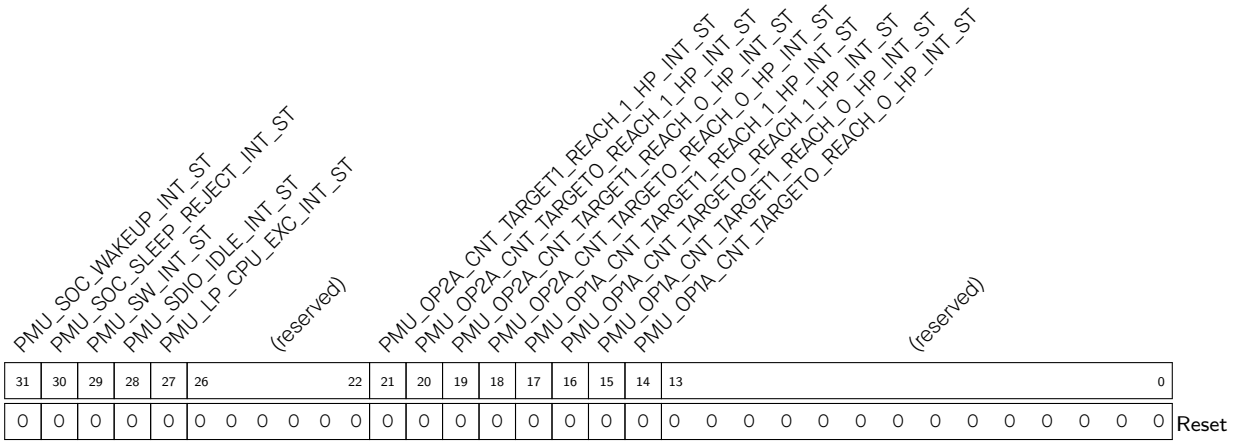
PMU_SDIO_IDLE_INT_RAW PMU_SDIO_IDLE_INT 的原始中断状态。(R/WTC/SS)

PMU_SW_INT_RAW PMU_SW_INT 的原始中断状态。(R/WTC/SS)

PMU_SOC_SLEEP_REJECT_INT_RAW PMU_SOC_SLEEP_REJECT_INT 的原始中断状态。(R/WTC/SS)

PMU_SOC_WAKEUP_INT_RAW PMU_SOC_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 15.44. PMU_HP_INT_ST_REG (0x0168)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_ST PMU_OP1A_CNT_TARGET0_REACH_0_INT 的屏蔽中断状态。(RO)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_ST PMU_OP1A_CNT_TARGET1_REACH_0_INT 的屏蔽中断状态。(RO)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_ST PMU_OP1A_CNT_TARGET0_REACH_1_INT 的屏蔽中断状态。(RO)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_ST PMU_OP1A_CNT_TARGET1_REACH_1_INT 的屏蔽中断状态。(RO)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_ST PMU_OP2A_CNT_TARGET0_REACH_0_INT 的屏蔽中断状态。(RO)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_ST PMU_OP2A_CNT_TARGET1_REACH_0_INT 的屏蔽中断状态。(RO)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_ST PMU_OP2A_CNT_TARGET0_REACH_1_INT 的屏蔽中断状态。(RO)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_ST PMU_OP2A_CNT_TARGET1_REACH_1_INT 的屏蔽中断状态。(RO)

PMU_LP_CPU_EXC_INT_ST PMU_LP_CPU_EXC_INT 的屏蔽中断状态。(RO)

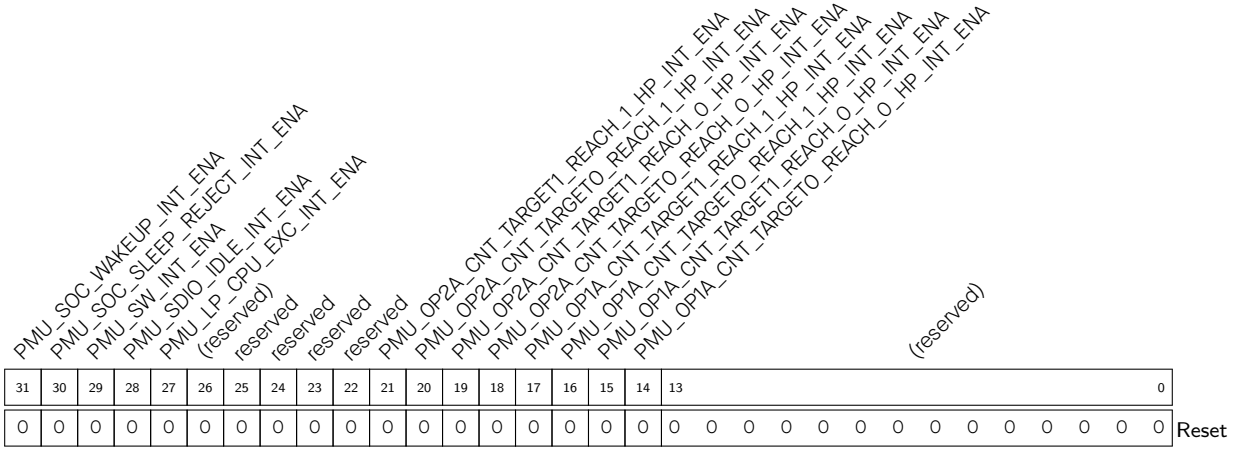
PMU_SDIO_IDLE_INT_ST PMU_SDIO_IDLE_INT 的屏蔽中断状态。(RO)

PMU_SW_INT_ST PMU_SW_INT 的屏蔽中断状态。(RO)

PMU_SOC_SLEEP_REJECT_INT_ST PMU_SOC_SLEEP_REJECT_INT 的屏蔽中断状态。(RO)

PMU_SOC_WAKEUP_INT_ST PMU_SOC_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 15.45. PMU_HP_INT_ENA_REG (0x016C)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET0_REACH_0_INT。(R/W)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET1_REACH_0_INT。(R/W)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET0_REACH_1_INT。(R/W)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET1_REACH_1_INT。(R/W)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET0_REACH_0_INT。(R/W)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET1_REACH_0_INT。(R/W)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET0_REACH_1_INT。(R/W)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET1_REACH_1_INT。(R/W)

PMU_LP_CPU_EXC_INT_ENA 写 1 使能 PMU_LP_CPU_EXC_INT。(R/W)

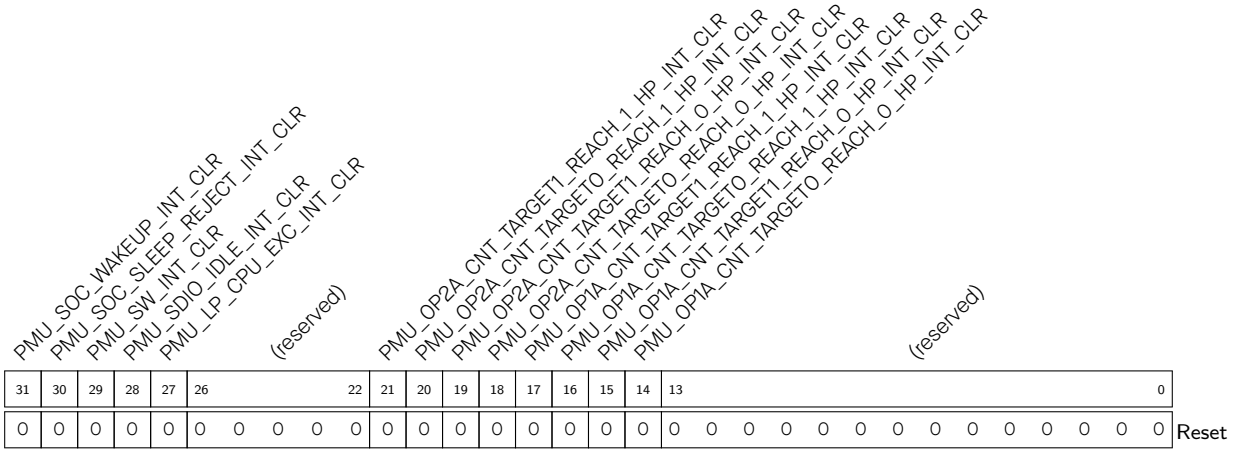
PMU_SDIO_IDLE_INT_ENA 写 1 使能 PMU_SDIO_IDLE_INT。(R/W)

PMU_SW_INT_ENA 写 1 使能 PMU_SW_INT。(R/W)

PMU_SOC_SLEEP_REJECT_INT_ENA 写 1 使能 PMU_SOC_SLEEP_REJECT_INT。(R/W)

PMU_SOC_WAKEUP_INT_ENA 写 1 使能 PMU_SOC_WAKEUP_INT。(R/W)

Register 15.46. PMU_HP_INT_CLR_REG (0x0170)



PMU_OP1A_CNT_TARGET0_REACH_0_HP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET0_REACH_0_INT。(WT)

PMU_OP1A_CNT_TARGET1_REACH_0_HP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET1_REACH_0_INT。(WT)

PMU_OP1A_CNT_TARGET0_REACH_1_HP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET0_REACH_1_INT。(WT)

PMU_OP1A_CNT_TARGET1_REACH_1_HP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET1_REACH_1_INT。(WT)

PMU_OP2A_CNT_TARGET0_REACH_0_HP_INT_CLR 写 1 清除 VPMU_OP2A_CNT_TARGET0_REACH_0_INT。(WT)

PMU_OP2A_CNT_TARGET1_REACH_0_HP_INT_CLR 写 1 清除 PMU_OP2A_CNT_TARGET1_REACH_0_INT。(WT)

PMU_OP2A_CNT_TARGET0_REACH_1_HP_INT_CLR 写 1 清除 PMU_OP2A_CNT_TARGET0_REACH_1_INT。(WT)

PMU_OP2A_CNT_TARGET1_REACH_1_HP_INT_CLR 写 1 清除 PMU_OP2A_CNT_TARGET1_REACH_1_INT。(WT)

PMU_LP_CPU_EXC_INT_CLR 写 1 清除 PMU_LP_CPU_EXC_INT。(WT)

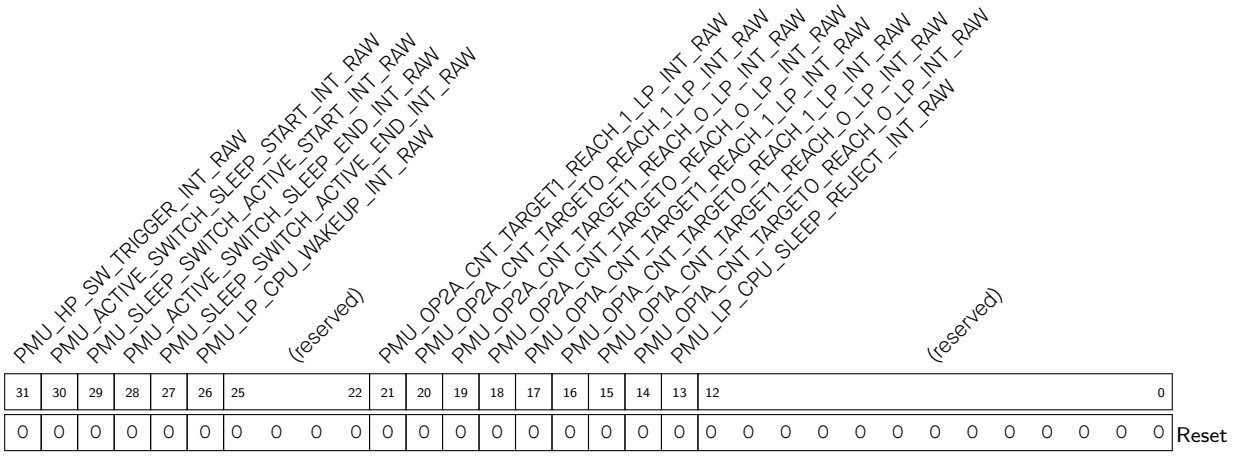
PMU_SDIO_IDLE_INT_CLR 写 1 清除 PMU_SDIO_IDLE_INT。(WT)

PMU_SW_INT_CLR 写 1 清除 PMU_SW_INT。(WT)

PMU_SOC_SLEEP_REJECT_INT_CLR 写 1 清除 PMU_SOC_SLEEP_REJECT_INT。(WT)

PMU_SOC_WAKEUP_INT_CLR 写 1 清除 PMU_SOC_WAKEUP_INT。(WT)

Register 15.47. PMU_LP_INT_RAW_REG (0x0174)



PMU_LP_CPU_SLEEP_REJECT_INT_RAW PMU_LP_CPU_SLEEP_REJECT_INT 的原始中断位。(R/WTC/SS)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_RAW PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT 的原始中断位。(R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_RAW PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT 的原始中断位。(R/WTC/SS)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_RAW PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT 的原始中断位。(R/WTC/SS)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_RAW PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT 的原始中断位。(R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_RAW PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT 的原始中断位。(R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_RAW PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT 的原始中断位。(R/WTC/SS)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_RAW PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT 的原始中断位。(R/WTC/SS)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_RAW PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT 的原始中断位。(R/WTC/SS)

PMU_LP_CPU_WAKEUP_INT_RAW PMU_LP_CPU_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_RAW PMU_SLEEP_SWITCH_ACTIVE_END_INT 的原始中断状态。(R/WTC/SS)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_RAW PMU_ACTIVE_SWITCH_SLEEP_END_INT 的原始中断状态。(R/WTC/SS)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_RAW PMU_SLEEP_SWITCH_ACTIVE_START_INT 的原始中断状态。(R/WTC/SS)

见下页...

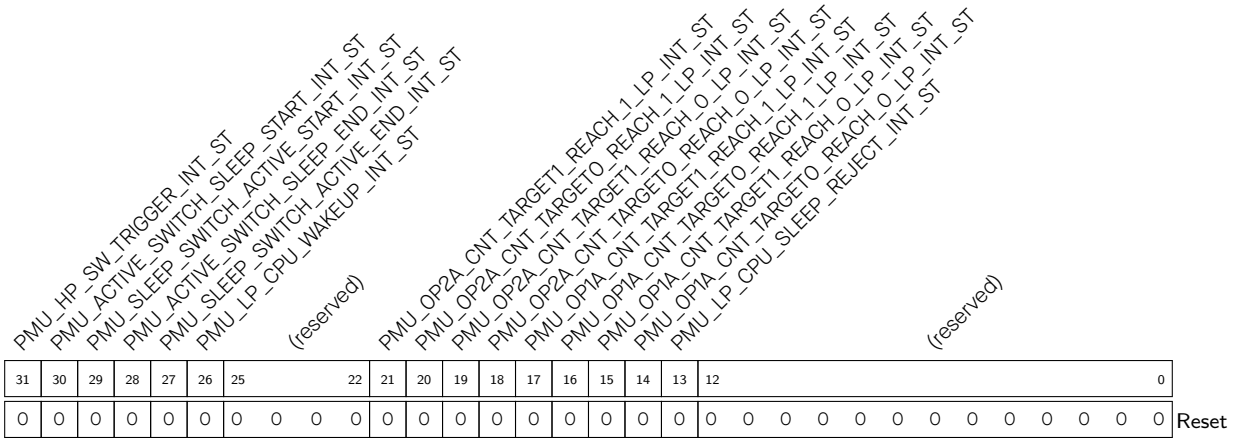
Register 15.47. PMU_LP_INT_RAW_REG (0x0174)

接上页...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_RAW PMU_ACTIVE_SWITCH_SLEEP_START_INT 的原始中断状态。(R/WTC/SS)

PMU_HP_SW_TRIGGER_INT_RAW PMU_HP_SW_TRIGGER_INT 的原始中断状态。(R/WTC/SS)

Register 15.48. PMU_LP_INT_ST_REG (0x0178)



PMU_LP_CPU_SLEEP_REJECT_INT_ST PMU_LP_CPU_SLEEP_REJECT_INT 的原始中断位。(RO)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_ST PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT 的屏蔽中断位。(RO)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_ST PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT 的屏蔽中断位。(RO)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_ST PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT 的屏蔽中断位。(RO)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_ST PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT 的屏蔽中断位。(RO)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_ST PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT 的屏蔽中断位。(RO)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_ST PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT 的屏蔽中断位。(RO)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_ST PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT 的屏蔽中断位。(RO)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_ST PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT 的屏蔽中断位。(RO)

PMU_LP_CPU_WAKEUP_INT_ST PMU_LP_CPU_WAKEUP_INT 的屏蔽中断状态。(RO)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_ST PMU_SLEEP_SWITCH_ACTIVE_END_INT 的屏蔽中断状态。(RO)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_ST PMU_ACTIVE_SWITCH_SLEEP_END_INT 的屏蔽中断状态。(RO)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_ST PMU_SLEEP_SWITCH_ACTIVE_START_INT 的屏蔽中断状态。(RO)

见下页...

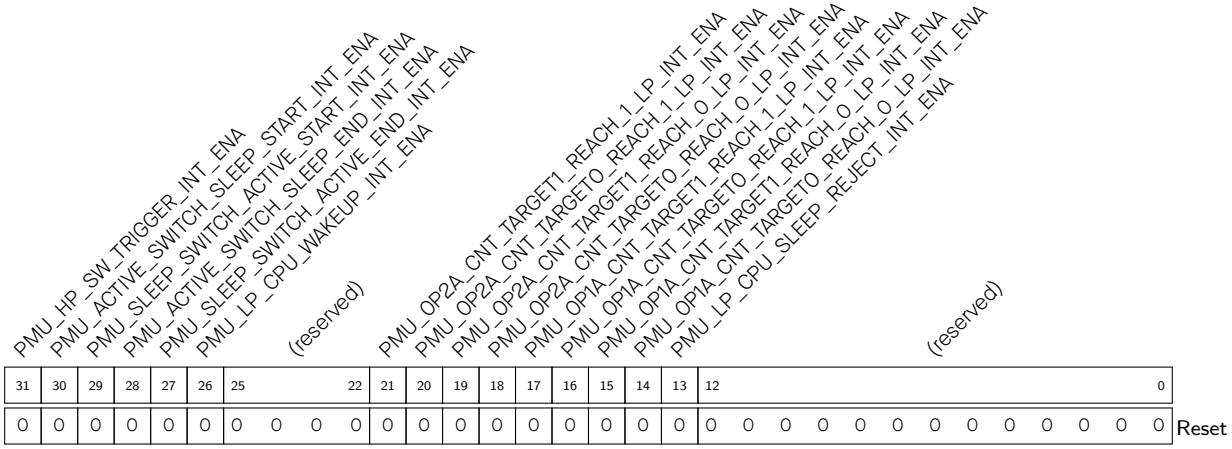
Register 15.48. PMU_LP_INT_ST_REG (0x0178)

接上页...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_ST PMU_ACTIVE_SWITCH_SLEEP_START_INT 的屏蔽中断状态。(RO)

PMU_HP_SW_TRIGGER_INT_ST PMU_HP_SW_TRIGGER_INT 的屏蔽中断状态。(RO)

Register 15.49. PMU_LP_INT_ENA_REG (0x017C)



PMU_LP_CPU_SLEEP_REJECT_INT_ENA 写 1 使能 PMU_LP_CPU_SLEEP_REJECT_INT。(R/W)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT。(R/W)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT。(R/W)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT。(R/W)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_ENA 写 1 使能 PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT。(R/W)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT。(R/W)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT。(R/W)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT。(R/W)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_ENA 写 1 使能 PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT。(R/W)

PMU_LP_CPU_WAKEUP_INT_ENA 写 1 使能 PMU_LP_CPU_WAKEUP_INT。(R/W)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_ENA 写 1 使能 PMU_SLEEP_SWITCH_ACTIVE_END_INT。(R/W)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_ENA 写 1 使能 PMU_ACTIVE_SWITCH_SLEEP_END_INT。(R/W)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_ENA 写 1 使能 PMU_SLEEP_SWITCH_ACTIVE_START_INT。(R/W)

见下页...

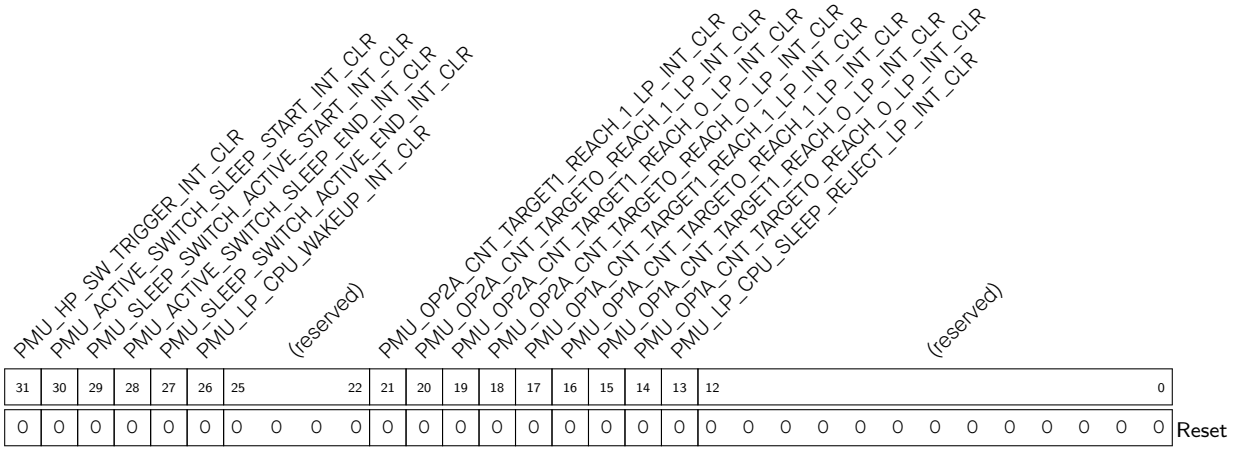
Register 15.49. PMU_LP_INT_ENA_REG (0x017C)

接上页...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_ENA 写 1 使能 PMU_ACTIVE_SWITCH_SLEEP_START_INT。(R/W)

PMU_HP_SW_TRIGGER_INT_ENA 写 1 使能 PMU_HP_SW_TRIGGER_INT。(R/W)

Register 15.50. PMU_LP_INT_CLR_REG (0x0180)



PMU_LP_CPU_SLEEP_REJECT_LP_INT_CLR 写 1 清除 PMU_LP_CPU_SLEEP_REJECT_INT。(WT)

PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET0_REACH_0_LP_INT。(WT)

PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET1_REACH_0_LP_INT。(WT)

PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET0_REACH_1_LP_INT。(WT)

PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT_CLR 写 1 清除 PMU_OP1A_CNT_TARGET1_REACH_1_LP_INT。(WT)

PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT_CLR 写 1 清除 PMU_OP2A_CNT_TARGET0_REACH_0_LP_INT。(WT)

PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT_CLR 写 1 清除 PMU_OP2A_CNT_TARGET1_REACH_0_LP_INT。(WT)

PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT_CLR 写 1 清除 PMU_OP2A_CNT_TARGET0_REACH_1_LP_INT。(WT)

PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT_CLR 写 1 清除 PMU_OP2A_CNT_TARGET1_REACH_1_LP_INT。(WT)

PMU_LP_CPU_WAKEUP_INT_CLR 写 1 清除 PMU_LP_CPU_WAKEUP_INT。(WT)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_CLR 写 1 清除 PMU_SLEEP_SWITCH_ACTIVE_END_INT。(WT)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_CLR 写 1 清除 PMU_ACTIVE_SWITCH_SLEEP_END_INT。(WT)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_CLR 写 1 清除 PMU_SLEEP_SWITCH_ACTIVE_START_INT。(WT)

见下页...

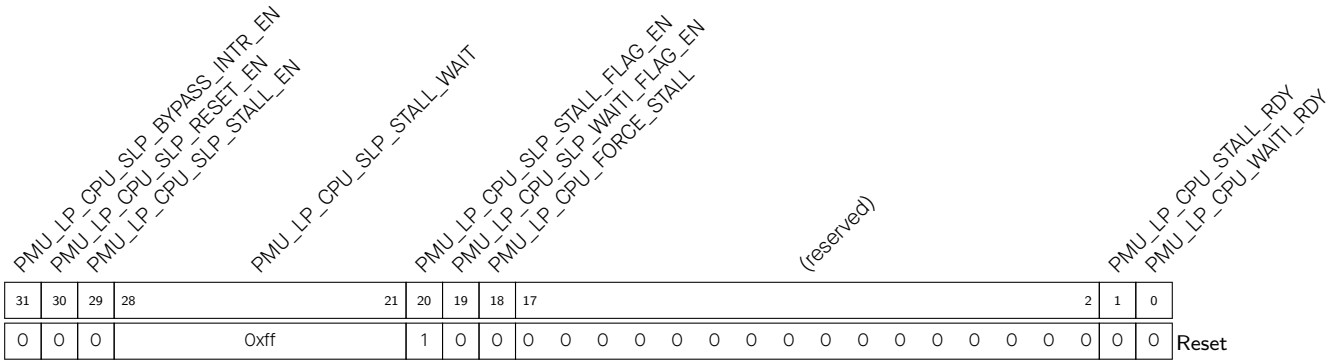
Register 15.50. PMU_LP_INT_CLR_REG (0x0180)

接上页...

PMU_ACTIVE_SWITCH_SLEEP_START_INT_CLR 写 1 清除 PMU_ACTIVE_SWITCH_SLEEP_START_INT。(WT)

PMU_HP_SW_TRIGGER_INT_CLR 写 1 清除 PMU_HP_SW_TRIGGER_INT。(WT)

Register 15.51. PMU_LP_CPU_PWR0_REG (0x0184)



PMU_LP_CPU_WAITI_RDY 表示 LP CPU 是否进入 waiti 状态。

0：不进入状态

1：进入状态

(RO)

PMU_LP_CPU_STALL_RDY 表示 LP CPU 是否进入 stall 状态。

0：不进入状态

1：进入状态

(RO)

PMU_LP_CPU_FORCE_STALL 配置 LP CPU 进入 stall 状态。

0：不进入状态

1：进入状态

(R/W)

PMU_LP_CPU_SLP_WAITI_FLAG_EN 配置 LP CPU 在进入睡眠状态时是否需要进入 waiti 状态。

0：不进入状态

1：进入状态

(R/W)

PMU_LP_CPU_SLP_STALL_FLAG_EN 配置 LP CPU 在进入睡眠状态时是否需要进入 stall 状态。

0：不进入状态

1：进入状态

(R/W)

PMU_LP_CPU_SLP_STALL_WAIT 配置 LP CPU 在进入睡眠状态使能 stall 状态之后，等待 stall 生效的时间。单位是 LP_DYN_FAST_CLK。(R/W)

PMU_LP_CPU_SLP_STALL_EN 配置是否暂停进入睡眠模式的 LP CPU。

0：不暂停

1：暂停

(R/W)

PMU_LP_CPU_SLP_RESET_EN 配置是否复位进入睡眠模式的 LP CPU。

0：不复位

1：复位

(R/W)

见下页...

Register 15.51. PMU_LP_CPU_PWR0_REG (0x0184)

接上页...

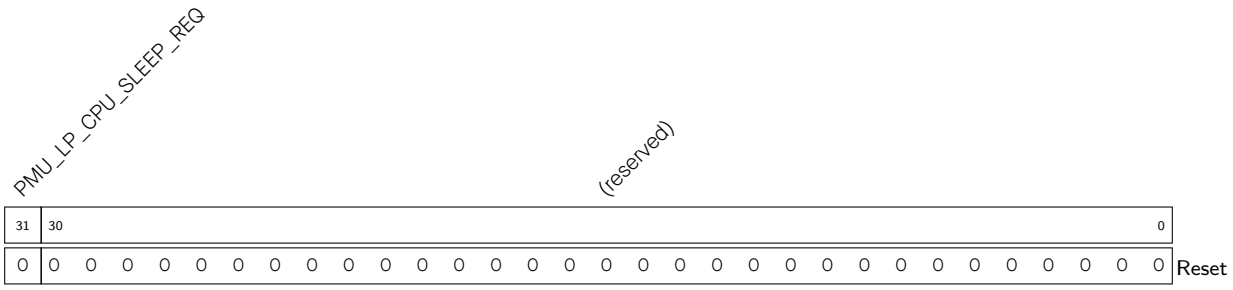
PMU_LP_CPU_SLP_BYPASS_INTR_EN 配置是否为进入睡眠模式的 LP CPU 打开中断信号。

0：关闭中断信号

1：打开中断信号

(R/W)

Register 15.52. PMU_LP_CPU_PWR1_REG (0x0188)



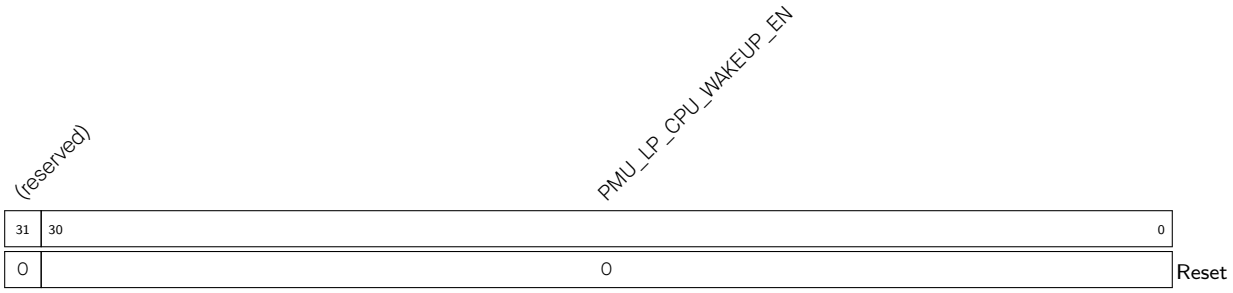
PMU_LP_CPU_SLEEP_REQ LP CPU 睡眠请求寄存器。

0：不睡眠

1：进入睡眠

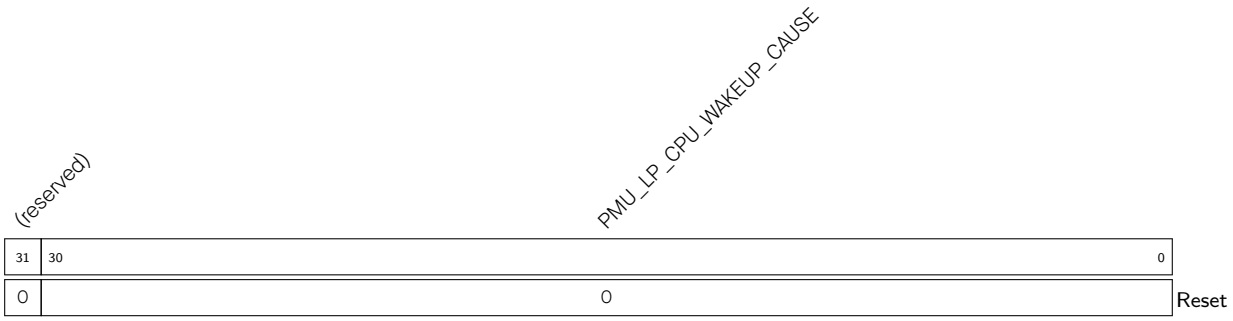
(WT)

Register 15.53. PMU_LP_CPU_PWR2_REG (0x018C)



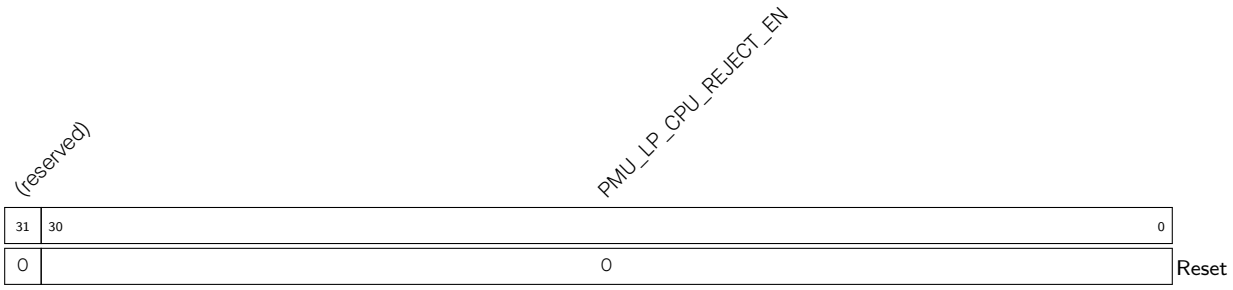
PMU_LP_CPU_WAKEUP_EN 配置 LP CPU 的唤醒源。具体请参考表 15.4-1 唤醒源。(R/W)

Register 15.54. PMU_LP_CPU_PWR3_REG (0x0190)



PMU_LP_CPU_WAKEUP_CAUSE 表示 LP CPU 唤醒的原因。(RO)

Register 15.55. PMU_LP_CPU_PWR4_REG (0x0194)



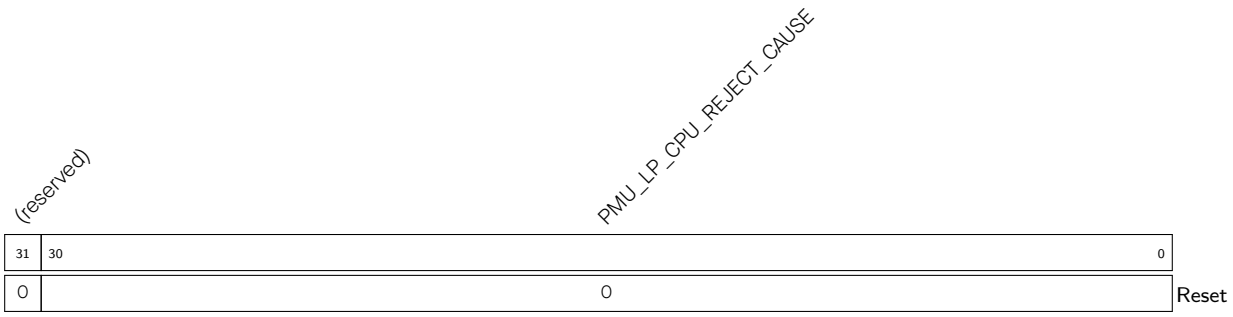
PMU_LP_CPU_REJECT_EN 配置是否使能 LP CPU 拒绝睡眠功能。

0：禁能

1：使能

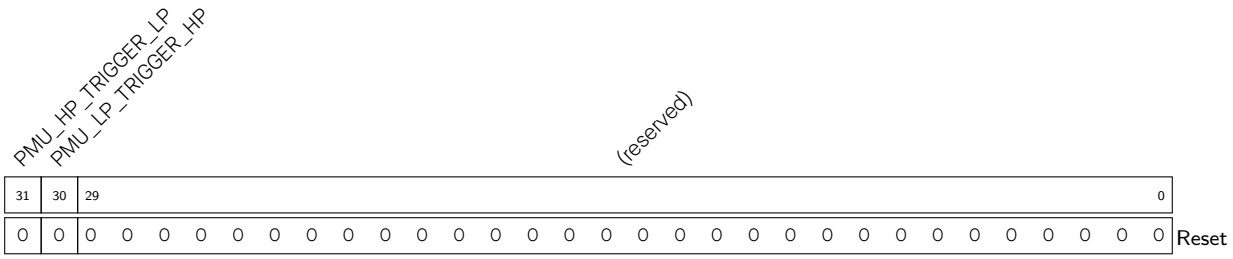
(R/W)

Register 15.56. PMU_LP_CPU_PWR5_REG (0x0198)



PMU_LP_CPU_REJECT_CAUSE 表示 LP CPU 拒绝睡眠的原因。(RO)

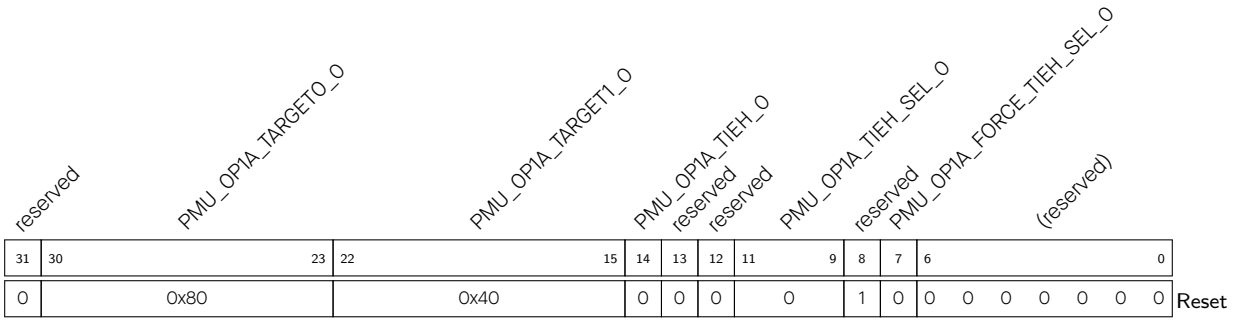
Register 15.57. PMU_HP_LP_CPU_COMM_REG (0x019C)



PMU_LP_TRIGGER_HP LP CPU 配置该寄存器为 1 时，芯片被唤醒。(WT)

PMU_HP_TRIGGER_LP HP CPU 配置该寄存器为 1 时，LP CPU 被唤醒。(WT)

Register 15.58. PMU_EXT_LDO_P0_0P1A_REG (0x01B8)



PMU_0P1A_FORCE_TIEH_SEL_0 选择 VO1 调压器的模式选择控制权。

1：模式选择由寄存器 PMU_0P1A_TIEH_SEL_0 控制

0：模式选择由 eFuse 控制

(R/W)

PMU_0P1A_TIEH_SEL_0 选择 VO1 调压器模式。

0：由寄存器 PMU_0P1A_TIEH_0 控制

1：由 SDMMC_0 控制

2：透传模式

3：由 SDMMC_1 控制

(R/W)

PMU_0P1A_TIEH_0 选择 VO1 调压器模式。

0：LDO 模式

1：透传模式

(R/W)

PMU_0P1A_TARGET1_0 配置 VO1 调压器等待计数器阶段 2 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

PMU_0P1A_TARGET0_0 配置 VO1 调压器等待计数器阶段 1 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

Register 15.59. PMU_EXT_LDO_P0_0P1A_ANA_REG (0x01BC)

31–28	27	26	25–23	22–0	
PMU_ANA_0P1A_DREF_0	PMU_ANA_0P1A_EN_CUR_LIM_0	PMU_ANA_0P1A_EN_VDET_0	PMU_ANA_0P1A_MUL_0	(reserved)	
0xb	0	0	0x2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

PMU_ANA_0P1A_MUL_0 配置 MUL，调压器的输出电压为 MUL × Vref。

0-7：MUL 从 1 到 2.75，步进为 0.25。

(R/W)

PMU_ANA_0P1A_EN_VDET_0 配置是否使能 VO1 电压检测。

0：不使能

1：使能

(R/W)

PMU_ANA_0P1A_EN_CUR_LIM_0 配置是否使能 VO1 限流保护。

0：不使能

1：使能

(R/W)

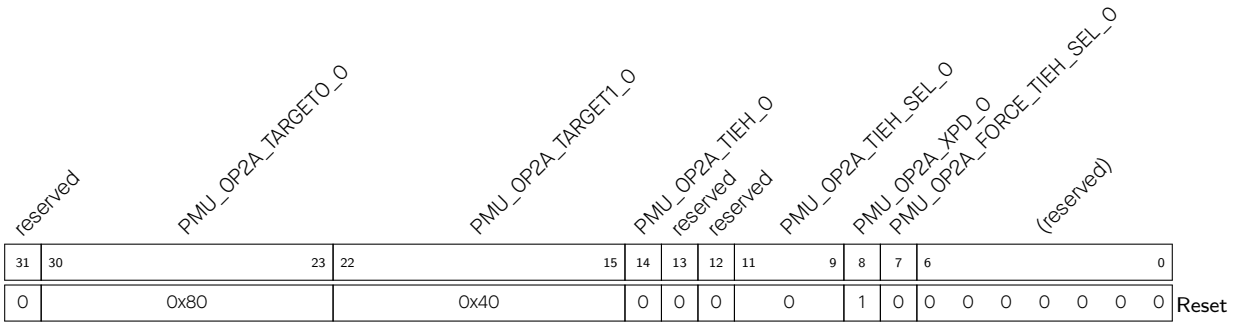
PMU_ANA_0P1A_DREF_0 配置 LDO Vref，调压器的输出电压为 MUL × Vref。

0-8：Vref = 0.5 V - 0.9 V，步进为 50 mV

9-15：Vref = 1.0 V - 1.6 V，步进为 100 mV

(R/W)

Register 15.60. PMU_EXT_LDO_P0_0P2A_REG (0x01C0)



PMU_0P2A_FORCE_TIEH_SEL_0 选择 VO3 调压器模式。

0：由寄存器 PMU_0P2A_TIEH_SEL_0 控制

1：由 SDMMC_0 控制

2：透传模式

3：由 SDMMC_1 控制

(R/W)

PMU_0P2A_XPD_0 配置软件开启 VO3 调压器。

0：关闭

1：打开

(R/W)

PMU_0P2A_TIEH_SEL_0 选择 VO3 调压器模式。

0：由寄存器 PMU_0P2A_TIEH_0 控制

1：由 SDMMC_0 控制

2：透传模式

3：由 SDMMC_1 控制

(R/W)

PMU_0P2A_TIEH_0 选择 VO3 调压器模式。

0：LDO 模式

1：透传模式

(R/W)

PMU_0P2A_TARGET1_0 配置 VO3 调压器等待计数器阶段 2 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

PMU_0P2A_TARGET0_0 配置 VO3 调压器等待计数器阶段 1 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

Register 15.61. PMU_EXT_LDO_P0_0P2A_ANA_REG (0x01C4)

PMU_ANA_0P2A_DREF_0	PMU_ANA_0P2A_EN_CUR_LIM_0	PMU_ANA_0P2A_EN_VDET_0	PMU_ANA_0P2A_MUL_0	(reserved)	
31–28	27	26	25–23	22–0	
0xb	0	0	0x2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

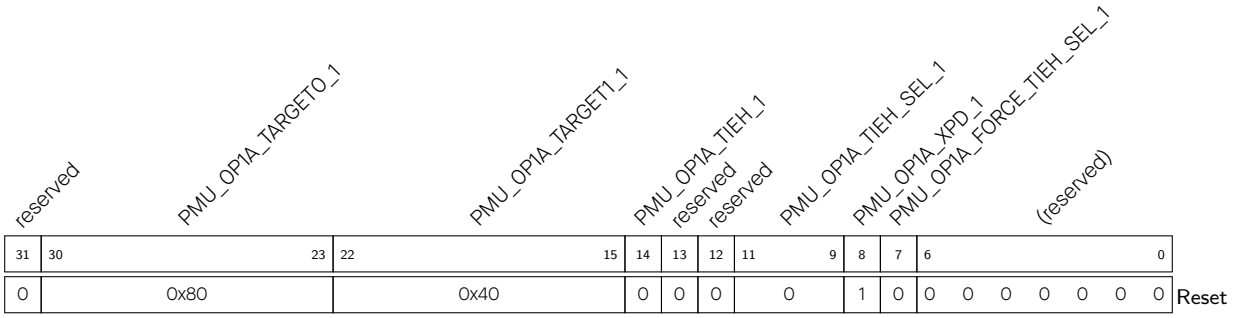
PMU_ANA_0P2A_MUL_0 配置 MUL，调压器的输出电压为 MUL × Vref。
0-7：MUL 从 1 到 2.75，步进为 0.25。
(R/W)

PMU_ANA_0P2A_EN_VDET_0 配置是否使能 VO3 电压检测。
0：不使能
1：使能
(R/W)

PMU_ANA_0P2A_EN_CUR_LIM_0 配置是否使能 VO3 限流保护。
0：不使能
1：使能
(R/W)

PMU_ANA_0P2A_DREF_0 配置 LDO Vref，调压器的输出电压为 MUL × Vref。
0-8：Vref = 0.5 V - 0.9 V，步进为 50 mV
9-15：Vref = 1.0 V - 1.6 V，步进为 100 mV
(R/W)

Register 15.62. PMU_EXT_LDO_P1_0P1A_REG (0x01D0)



PMU_0P1A_FORCE_TIEH_SEL_1 选择 VO2 调压器模式。

0：由寄存器 PMU_0P1A_TIEH_SEL_1 控制

1：由 SDMMC_0 控制

2：透传模式

3：由 SDMMC_1 控制

(R/W)

PMU_0P1A_XPD_1 配置软件开启 VO2 调压器。

0：关闭

1：打开

(R/W)

PMU_0P1A_TIEH_SEL_1 选择 VO2 调压器模式。

0：由寄存器 PMU_0P1A_TIEH_1 控制

1：由 SDMMC_0 控制

2：透传模式

3：由 SDMMC_1 控制

(R/W)

PMU_0P1A_TIEH_1 选择 VO2 调压器模式。

0：LDO 模式

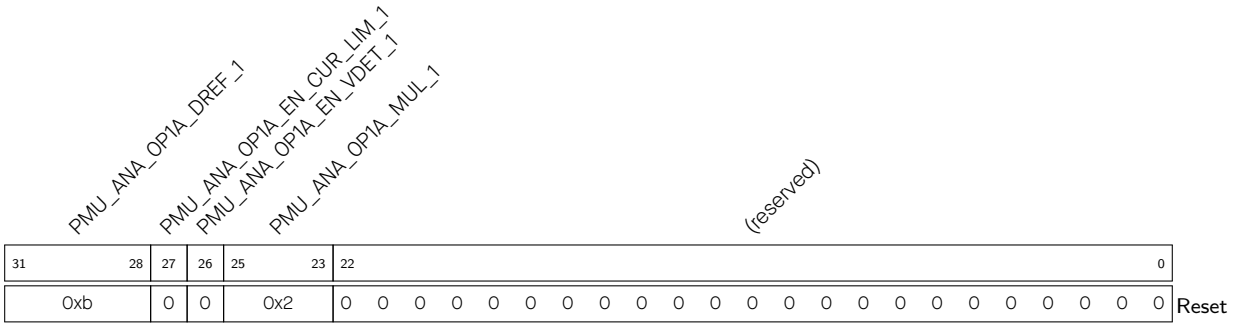
1：透传模式

(R/W)

PMU_0P1A_TARGET1_1 配置 VO2 调压器等待计数器阶段 2 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

PMU_0P1A_TARGET0_1 配置 VO2 调压器等待计数器阶段 1 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

Register 15.63. PMU_EXT_LDO_P1_0P1A_ANA_REG (0x01D4)



PMU_ANA_0P1A_MUL_1 配置 MUL，调压器的输出电压为 MUL × Vref。

0-7：MUL 从 1 到 2.75，步进为 0.25。

(R/W)

PMU_ANA_0P1A_EN_VDET_1 配置是否使能 VO2 电压检测。

0：不使能

1：使能

(R/W)

PMU_ANA_0P1A_EN_CUR_LIM_1 配置是否使能 VO2 限流保护。

0：不使能

1：使能

(R/W)

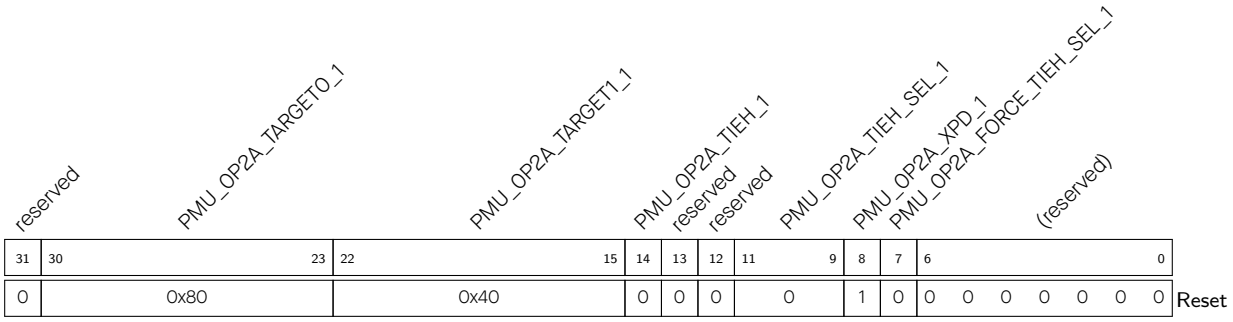
PMU_ANA_0P1A_DREF_1 配置 LDO Vref，调压器的输出电压为 MUL × Vref。

0-8：Vref = 0.5 V - 0.9 V，步进为 50 mV

9-15：Vref = 1.0 V - 1.6 V，步进为 100 mV

(R/W)

Register 15.64. PMU_EXT_LDO_P1_0P2A_REG (0x01D8)



PMU_0P2A_FORCE_TIEH_SEL_1 选择 VO4 调压器模式。

0：由寄存器 PMU_0P2A_TIEH_SEL_1 控制

1：由 SDMMC_0 控制

2：透传模式

3：由 SDMMC_1 控制

(R/W)

PMU_0P2A_XPD_1 配置软件开启 VO4 调压器。

0：关闭

1：打开

(R/W)

PMU_0P2A_TIEH_SEL_1 选择 VO4 调压器模式。

0：由寄存器 PMU_0P2A_TIEH_1 控制

1：由 SDMMC_0 控制

2：透传模式

3：由 SDMMC_1 控制

(R/W)

PMU_0P2A_TIEH_1 选择 VO4 调压器模式。

0：LDO 模式

1：透传模式

(R/W)

PMU_0P2A_TARGET1_1 配置 VO4 调压器等待计数器阶段 2 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

PMU_0P2A_TARGET0_1 配置 VO4 调压器等待计数器阶段 1 的超时时间，单位是 LP_DYN_SLOW_CLK 的 16 分频。(R/W)

Register 15.65. PMU_EXT_LDO_P1_OP2A_ANA_REG (0x01DC)

PMU_ANA_OP2A_DREF_1	PMU_ANA_OP2A_EN_CUR_LIM_1	PMU_ANA_OP2A_EN_VDET_1	PMU_ANA_OP2A_MUL_1	(reserved)	
31 28	27	26	25 23	22 0	
0xb	0	0	0x2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

PMU_ANA_OP2A_MUL_1 配置 MUL，调压器的输出电压为 MUL × Vref。

0-7：MUL 从 1 到 2.75，步进为 0.25。

(R/W)

PMU_ANA_OP2A_EN_VDET_1 配置是否使能 VO4 电压检测。

0：不使能

1：使能

(R/W)

PMU_ANA_OP2A_EN_CUR_LIM_1 配置是否使能 VO4 限流保护。

0：不使能

1：使能

(R/W)

PMU_ANA_OP2A_DREF_1 配置 LDO Vref，调压器的输出电压为 MUL × Vref。

0-8：Vref = 0.5 V - 0.9 V，步进为 50 mV

9-15：Vref = 1.0 V - 1.6 V，步进为 100 mV

(R/W)

Register 15.66. PMU_EXT_WAKEUP_LV_REG (0x01E8)

PMU_EXT_WAKEUP_LV	
31 0	
0	Reset

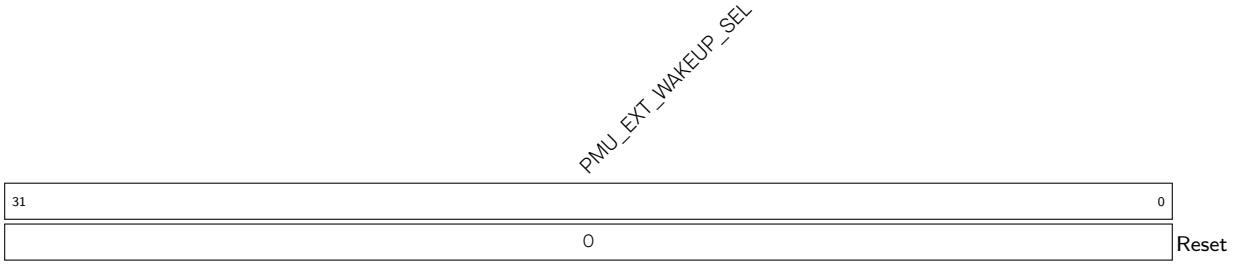
PMU_EXT_WAKEUP_LV 配置 EXT 唤醒的唤醒电平位图。

0：低电平唤醒

1：高电平唤醒

(R/W)

Register 15.67. PMU_EXT_WAKEUP_SEL_REG (0x01EC)



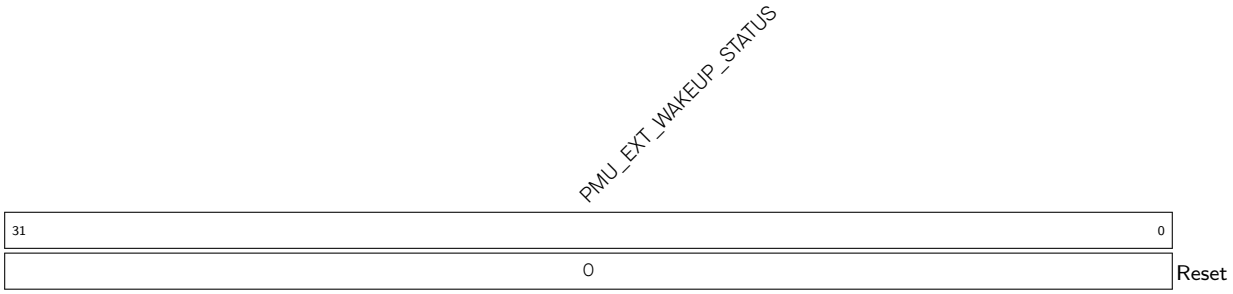
PMU_EXT_WAKEUP_SEL 配置 EXT 唤醒的唤醒使能位图。

0：禁止

1：使能

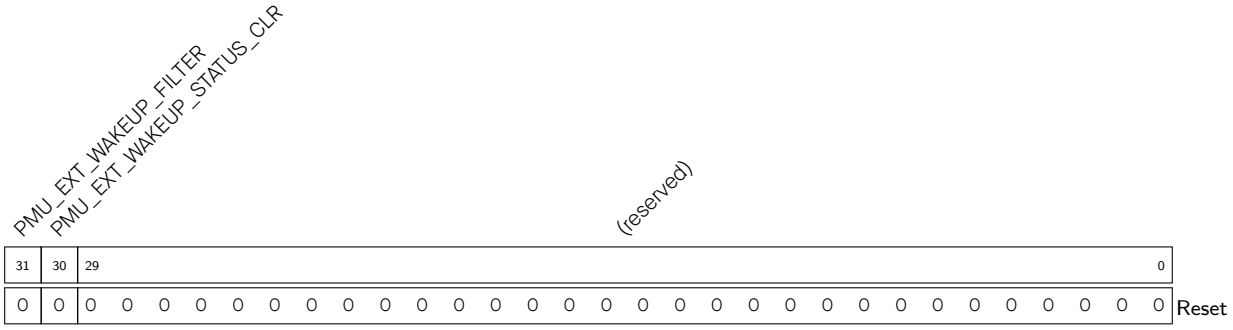
(R/W)

Register 15.68. PMU_EXT_WAKEUP_ST_REG (0x01F0)



PMU_EXT_WAKEUP_STATUS 获取唤醒的 GPIO 编号。0 对应 GPIO0，1 对应 GPIO1，以此类推。(RO)

Register 15.69. PMU_EXT_WAKEUP_CNTL_REG (0x01F4)



PMU_EXT_WAKEUP_STATUS_CLR 清空唤醒记录。(R/W)

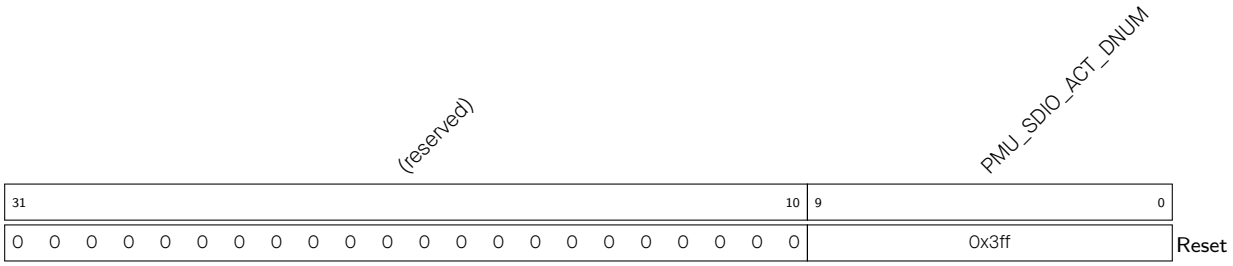
PMU_EXT_WAKEUP_FILTER 配置是否使能唤醒滤波器。

0：禁止

1：使能

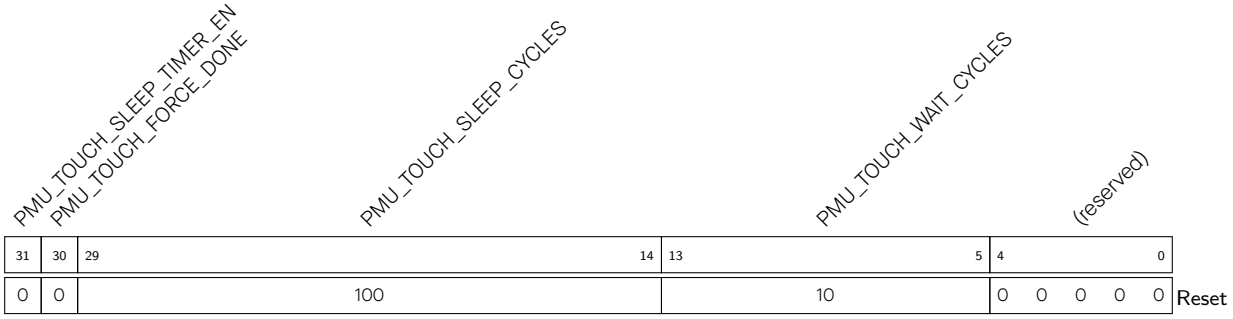
(R/W)

Register 15.70. PMU_SDIO_WAKEUP_CNTL_REG (0x01F8)



PMU_SDIO_ACT_DNUM 配置 SDIO 的工作状态的维持时间。单位是 LP_DYN_FAST_CLK。(R/W)

Register 15.71. PMU_TOUCH_PWR_CNTL_REG (0x0210)



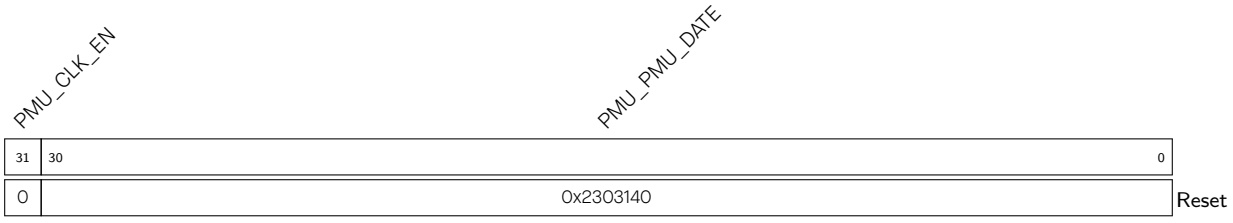
PMU_TOUCH_WAIT_CYCLES 配置 TOUCH 定时器唤醒 PMU 等待的时间，单位是 LP_DYN_SLOW_CLK。(R/W)

PMU_TOUCH_SLEEP_CYCLES 配置 TOUCH 的睡眠时间，单位是 LP_DYN_SLOW_CLK。(R/W)

PMU_TOUCH_FORCE_DONE 强制返回 TOUCH 定时器的工作状态。(R/W)

PMU_TOUCH_SLEEP_TIMER_EN 使能 TOUCH 定时器开始工作。(R/W)

Register 15.72. PMU_DATE_REG (0x03FC)



PMU_PMU_DATE 版本寄存器。(R/W)

PMU_CLK_EN 强制开启寄存器的自动时钟门控。(R/W)

第 16 章

系统定时器

16.1 概述

ESP32-P4 芯片内置 52 位系统定时器。该定时器可用于生成操作系统所需的滴答定时中断，也可以用作普通的定时器生成周期或单次延时中断。

16.2 主要特性

系统定时器具有如下特性：

- 集成两个 52 位计数器和三个 52 位比较器
- 支持软件访问由 APB_CLK 时钟驱动的寄存器
- 支持 CNT_CLK 时钟计数，两次计数周期的平均频率为 16 MHz
- 配置 XTAL_CLK (40 MHz) 作为 CNT_CLK 时钟源
- 支持 52 位报警值 (t) 和 26 位报警周期 (δt)
- 支持两种报警模式：
  - 单次报警模式：根据设定的目标报警值 (t)，生成一次性报警
  - 周期报警模式：根据设定的报警周期 (δt)，生成周期性报警
- 支持根据设置的报警值 (t) 或报警周期 (δt)，通过三个比较器生成三个独立中断
- 支持软件配置基准计数值。例如，支持从 Light-sleep 唤醒之后，系统定时器通过软件加载 RTC 定时器记录的睡眠时间，并进行补偿
- CPU 处于停止状态或处于在线调试状态时，系统定时器可选择停止运行或继续运行
- 支持事件任务矩阵 (ETM) 事件报警

16.3 系统定时器架构

系统定时器内置两个计数器（UNIT0 和 UNIT1）以及三个比较器（COMP0、COMP1 和 COMP2）。比较器用于监控计数器的计数值是否达到报警值。定时器的功能块图见图 16.3-1。

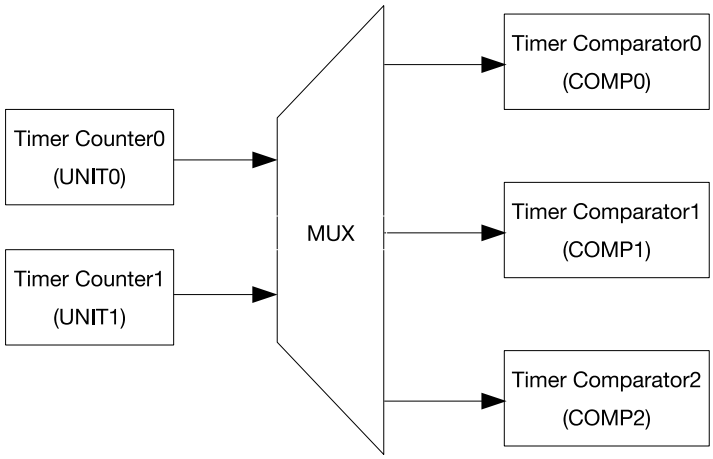


图 16.3-1. 系统定时器结构图

16.4 时钟源选择

计数器和比较器使用 XTAL_CLK 或者 RC_FAST_CLK 作为时钟源。可通过配置寄存器 HP_SYS_CLKRST_PERI_CLK_CTRL21_REG 的 HP_SYS_CLKRST_SYSTIMER_CLK_SRC_SEL 字段来选择时钟源。XTAL_CLK 经 2.5 分频后，生成频率为 $f_{XTAL_CLK}/2.5$ 的计数器时钟信号 CNT_CLK。也就是说，CNT_CLK 的实际平均频率为 16 MHz，如图 16.5-1 所示。每个 CNT_CLK 时钟周期，计数递增 1/16 μs，即 16 个周期递增 1 μs。

APB_CLK 为配置寄存器等软件操作提供时钟信号。更多有关 APB_CLK 的信息，见章节 11 复位和时钟。

用户可使用系统寄存器的两个相关位来控制系统定时器，具体如下：

- 置位寄存器 HP_SYS_CLKRST_SOC_CLK_CTRL2_REG 中 HP_SYS_CLKRST_SYSTIMER_APB_CLK_EN 位，使能系统定时器的 APB
  _CLK 信号。
- 置位寄存器 HP_SYS_CLKRST_HP_RST_EN1_REG 中 HP_SYS_CLKRST_RST_EN_SYSTIMER 位，复位系统定时器。

注意，复位后，系统定时器的寄存器将恢复到默认值。更多信息可参考章节 11 复位和时钟。

16.5 功能描述

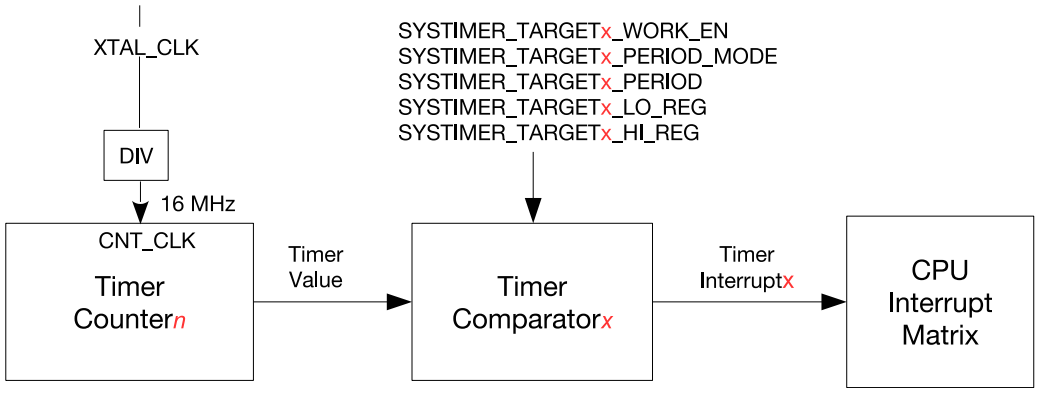


图 16.5-1. 系统定时器生成报警

图 16.5-1 展示了系统定时器生成报警的过程。过程中需要用到一个计数器和一个比较器，比较器将根据比较结果，生成报警中断。

16.5.1 计数器

系统定时器集成两个 52 位计数器，用 UNIT*n* 表示，*n* 可以取 0 或 1。计数器使用 16 MHz CNT_CLK 作为计数时钟。用户可通过配置寄存器 SYSTIMER_CONF_REG 中的下面三个位来控制计数器 UNIT*n*，具体如下：

- SYSTIMER_TIMER_UNIT*n*_WORK_EN：置位此位，使能计数器 UNIT*n*。
- SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN：置位此位，CPU0 停止运行后，计数器 UNIT*n* 也将停止工作。CPU 恢复运行后，计数器重新开始工作。
- SYSTIMER_TIMER_UNIT*n*_CORE1_STALL_EN：置位此位，CPU1 停止运行后，计数器 UNIT*n* 也将停止工作。CPU 恢复运行后，计数器重新开始工作。

假设 CPU 当前状态为停止工作，UNIT*n* 的具体配置见下表：

表 16.5-1. UNIT*n* 配置控制位

SYSTIMER_TIMER_ UNIT*n*_WORK_EN	SYSTIMER_TIMER_ UNIT*n*_CORE0_STALL_EN	SYSTIMER_TIMER_ UNIT*n*_CORE1_STALL_EN	计数器 UNIT*n*
0	x*	x*	未处于工作状态。
1	x*	1	暂停计数，但在 CPU core1 恢复运行后继续计数。
1	1	x*	暂停计数，但在 CPU core0 恢复运行后继续计数。
1	0	0	不受影响，照常计数。

* x：无关项

计数器 UNIT*n* 处于工作状态时，计数值按计数周期递增。UNIT*n* 停止工作时，计数值将保持不变，不再递增。

计数起始值的低 32 位和高 20 位分别从 SYSTIMER_TIMER_UNIT*n*_LOAD_LO 和 SYSTIMER_TIMER_

UNIT*n*_LOAD_HI 中装载。置位 SYSTIMER_TIMER_UNIT*n*_LOAD 将触发重装载事件，当前计数起始值立即更新。如果计数器 UNIT*n* 处于工作状态，则从新装载的计数值开始计数。

置位 SYSTIMER_TIMER_UNIT*n*_UPDATE 将触发更新事件。当前计数值的低 32 位和高 20 位被锁存至寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_LO 和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 后，SYSTIMER_TIMER_UNIT*n*_VALUE_VALID 会被置起。在下次更新事件发生前，SYSTIMER_TIMER_UNIT*n*_VALUE_LO 和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 寄存器中的值保持不变。

16.5.2 比较器和报警

系统定时器有三个 52 位比较器，用 COMP*x* 表示，其中 *x* 可以取 0、1、2。比较器可根据设置的不同报警值 (t) 或报警周期 (δt)，触发不同的中断。

用户可配置寄存器 SYSTIMER_TARGET*x*_PERIOD_MODE，选择比较器 COMP*x* 生成报警的模式：

- 1：周期报警模式
- 0：单次报警模式

选择周期报警模式时，寄存器 SYSTIMER_TARGET*x*_PERIOD 中的值为报警周期 (δt)。假设当前计数值为 t1，经过一段时间，当计数值达到 t1 + δt 时，将触发一次报警中断。再经过一段时间，当计数值达到 t1 + 2*δt 时，将再一次触发报警中断，以此类推。通过上述方式即可实现周期性报警。

选择单次报警模式时，SYSTIMER_TIMER_TARGET*x*_LO 和 SYSTIMER_TIMER_TARGET*x*_HI 分别提供报警值 (t) 的低 32 位和高 20 位。假设当前计数值为 t2 (t2 <= t)，经过一段时间，当计数到报警值 (t) 时，触发一次报警。与周期报警模式不同，单次报警模式仅生成一次报警中断。

用户可通过寄存器 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择用于与 COMP*x* 进行比较的计数器值，然后生成报警：

- 1：与计数器 UNIT*1* 的计数值进行比较
- 0：与计数器 UNIT*0* 的计数值进行比较

置位 SYSTIMER_TARGET*x*_WORK_EN，COMP*x* 开始进行比较：

- 在单次报警模式下，COMP*x* 将比较计数器中的实际计数值与寄存器中设置的报警值 (t)；
- 在周期报警模式下，COMP*x* 将比较计数器中的实际计数值与 t1 + n*δt (n = 1,2,3...)。

在单次报警模式下，当实际计数值等于报警值 (t)，或在周期报警模式下，实际计数值等于 t1 + n*δt (n = 1,2,3...) 时，将触发一次报警中断。但如果设定的报警值 (t) 小于当前计数值，即报警值 (t) 已成为过去，在当前计数值超过设定的报警值 (t) 一定范围 (0 ~ 2^{51} -1) 时，也将立即触发中断。无论是在单次报警模式还是在周期报警模式下，都可以从 SYSTIMER_TARGET*x*_LO_RO 和 SYSTIMER_TARGET*x*_HI_RO 中读取实际报警值的低 32 位和高 20 位。当前计数值 t_c、报警值 t_t 和触发报警的关系如下表所示：

表 16.5-2. 报警触发条件

t_c 与 t_t 的关系	触发条件
t_c- t_t <= 0	当 t_c = t_t 时，触发报警
0 <= t_c - t_t < 2^{51} - 1 (t_c < 2^{51} 且 t_t < 2^{51}； 或 t_c >= 2^{51} 且 t_t >= 2^{51})	立即触发报警

t_c - t_t >= 2^{51} - 1	t_c 达到最大值 52'hfffffffffffff 后溢出，然后从 0 开始计数，计数达到 t_t 时触发报警

16.5.3 事件任务矩阵

在 ESP32-P4 中，系统定时器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发系统定时器的 ETM 任务，或者通过系统定时器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与系统定时器相关的 ETM 任务和 ETM 事件。

系统定时器可产生的 ETM 事件有：

- SYSTIMER_EVT_CNT_CMP*x*：表示 COMP*x* 产生的报警事件。

当 SYSTIMER_ETM_EN 置 1 时，对应的报警事件会传到 ETM 模块。

16.5.4 同步操作

计数器与比较器的工作时钟和软件的配置时钟频率并不相同，因此在配置部分寄存器时，必须进行同步。完整的同步过程包括下面两个步骤：

1. 通过软件向配置寄存器字段写入合适的值，见表 16.5-3 第一列。
2. 通过软件置位相应的同步使能位，开始同步操作，见表 16.5-3 第二列。

表 16.5-3. 同步操作

需要同步的字段	同步使能位
SYSTIMER_TIMER_UNIT*n*_LOAD_LO SYSTIMER_TIMER_UNIT*n*_LOAD_HI	SYSTIMER_TIMER_UNIT*n*_LOAD
SYSTIMER_TARGET*x*_PERIOD SYSTIMER_TIMER_TARGET*x*_HI SYSTIMER_TIMER_TARGET*x*_LO	SYSTIMER_TIMER_COMP*x*_LOAD

计数器与比较器的工作时钟和软件的读取时钟频率并不相同，因此在读取部分寄存器时，也需要进行同步。完整的同步步骤如下：

1. 软件将 1 写入更新寄存器 SYSTIMER_TIMER_UNIT*n*_UPDATE 中。
2. 软件读取相应位 SYSTIMER_TIMER_UNIT*n*_VALUE_VALID 为有效，即表示同步完成。
3. 软件读取相应的状态寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 和 SYSTIMER_TIMER_UNIT*n*_VALUE_LO。

16.6 中断

ESP32-P4 的系统定时器可以生成以下中断信号并发送给 中断矩阵。

- SYSTIMER_TARGET0_INT
- SYSTIMER_TARGET1_INT

- SYSTIMER_TARGET2_INT

以上中断信号由系统定时器的内部中断源生成。表 16.6-1 列出了系统定时器的内部中断源、中断触发条件以及生成的中断信号。

表 16.6-1. 系统定时器的内部中断源

内部中断源	触发条件	中断信号
SYSTIMER_INT0	比较器 COMP0 报警时	SYSTIMER_TARGET0_INT
SYSTIMER_INT1	比较器 COMP1 报警时	SYSTIMER_TARGET1_INT
SYSTIMER_INT2	比较器 COMP2 报警时	SYSTIMER_TARGET2_INT

以上中断为电平类型中断。比较器开始报警时，中断信号拉高。中断信号将一直保持高电平，直至软件清除中断。用户可置位 SYSTIMER_TARGETx_INT_ENA 使能中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 16.8 寄存器列表。

16.7 编程示例

注意，在配置 COMP*x* 和 UNIT*n* 过程中，需保证对应的 COMP 和 UNIT 处于工作状态。

16.7.1 读取当前计数器的值

1. 置位 SYSTIMER_TIMER_UNIT*n*_UPDATE，将计数器 UNIT*n* 的值更新至寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 和 SYSTIMER_TIMER_UNIT*n*_VALUE_LO；
2. 轮询 SYSTIMER_TIMER_UNIT*n*_VALUE_VALID，直至其值为 1。之后，用户可从寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 和 SYSTIMER_TIMER_UNIT*n*_VALUE_LO 中读取计数器的值；
3. 读取寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_LO（低 32 位）和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI（高 20 位）。

16.7.2 在单次报警模式下配置一次性报警

1. 设置 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择与 COMP*x* 进行比较的计数器（UNIT*0* 或 UNIT*1*）。
2. 读取当前计数器的值，步骤见章节 16.7.1。读取的当前值可用于计算步骤 4 中的报警值 (t)。
3. 清除 SYSTIMER_TARGET*x*_PERIOD_MODE，使能单次报警模式。
4. 设置报警值 (t)，并将报警值 (t) 的低 32 位和高 20 位分别写入 SYSTIMER_TIMER_TARGET*x*_LO 和 SYSTIMER_TIMER_TARGET*x*_HI。
5. 置位 SYSTIMER_TIMER_COMP*x*_LOAD，同步报警值 (t)，即将报警值 (t) 装载至比较器 COMP*x*。
6. 置位 SYSTIMER_TARGET*x*_WORK_EN 使能选择的比较器 COMP*x*；比较器 COMP*x* 开始比较计数值与报警值 (t)。

7. 置位 SYSTIMER_TARGET*x*_INT_ENA，使能中断。UNIT*n* 达到报警值 (t) 时，触发一次报警中断 SYSTIMER_
   TARGET*x*_INT。

16.7.3 在周期报警模式下配置周期性报警

1. 设置 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择与 COMP*x* 进行比较的计数器。
2. 将报警周期 (δt) 写入 SYSTIMER_TARGET*x*_PERIOD。
3. 置位 SYSTIMER_TIMER_COMP*x*_LOAD 同步报警周期值，即将 (δt) 装载至比较器 COMP*x*。
4. 先清除再置位 SYSTIMER_TARGET*x*_PERIOD_MODE 将 COMP*x* 配置为周期报警模式。
5. 置位 SYSTIMER_TARGET*x*_WORK_EN 使能选择的比较器 COMP*x*；比较器 COMP*x* 开始将计数值与计数初始值 + n*δt (n = 1, 2, 3...) 进行比较。
6. 置位 SYSTIMER_TARGET*x*_INT_ENA，使能中断。UNIT*n* 计数达到计数初始值 + n*δt (n = 1, 2, 3...) 时，触发一次 SYSTIMER_TARGET*x*_INT 中断。

16.7.4 唤醒后时间补偿

1. 在芯片进入 Light-sleep 模式之前，用户需配置 RTC 定时器，精确记录睡眠时间，见章节 15 低功耗管理。
2. 从 Light-sleep 模式唤醒后，读取 RTC 定时器记录的睡眠时间。
3. 读取当前系统定时器的计数值，见章节 16.7.1。
4. 将 RTC 记录的以 RTC_SLOW_CLK 周期为单位的睡眠时间，转换成以 CNT_CLK (16 MHz) 周期为单位的睡眠时间。例如，如果 RTC_SLOW_CLK 频率为 32 kHz，则 RTC 定时器记录的时间乘以 500。
5. 将 RTC 定时器转换后的值加到系统定时器的当前计数值：
   - 将计算所得值的低 32 位写入 SYSTIMER_TIMER_UNIT*n*_LOAD_LO，高 20 位写入 SYSTIMER_TIMER_UNIT*n*_LOAD_HI。
   - 置位 SYSTIMER_TIMER_UNIT*n*_LOAD，将新的定时器值装载到系统定时器。至此，系统定时器更新完成。

16.8 寄存器列表

本小节的所有地址均为相对于系统定时器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时钟控制寄存器			
SYSTIMER_CONF_REG	配置系统定时器时钟	0x0000	R/W
UNIT*0* 控制和配置寄存器			
SYSTIMER_UNIT0_OP_REG	读取 UNT0 的值到相应寄存器	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	待装载至 UNIT0 的值，高 20 位	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	待装载至 UNIT0 的值，低 32 位	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 的读值，高 20 位	0x0040	RO
SYSTIMER_UNIT0_VALUE_LO_REG	UNIT0 的读值，低 32 位	0x0044	RO
SYSTIMER_UNIT0_LOAD_REG	计数器 UNIT0 的装载同步寄存器	0x005C	WT
UNIT*1* 控制和配置寄存器			
SYSTIMER_UNIT1_OP_REG	读取计数器 UNIT1 的值	0x0008	varies
SYSTIMER_UNIT1_LOAD_HI_REG	待装载至计数器 UNIT1 的值，高 20 位	0x0014	R/W
SYSTIMER_UNIT1_LOAD_LO_REG	待装载至计数器 UNIT1 的值，低 32 位	0x0018	R/W
SYSTIMER_UNIT1_VALUE_HI_REG	计数器 UNIT1 的读值，高 20 位	0x0048	RO
SYSTIMER_UNIT1_VALUE_LO_REG	计数器 UNIT1 的读值，低 32 位	0x004C	RO
SYSTIMER_UNIT1_LOAD_REG	计数器 UNIT1 的装载同步寄存器	0x0060	WT
比较器 COMP*0* 的控制和配置寄存器			
SYSTIMER_TARGET0_HI_REG	待装载至比较器 COMP0 的报警值，高 20 位	0x001C	R/W
SYSTIMER_TARGET0_LO_REG	待装载至比较器 COMP0 的报警值，低 32 位	0x0020	R/W
SYSTIMER_TARGET0_CONF_REG	配置比较器 COMP0 的报警模式	0x0034	R/W
SYSTIMER_COMP0_LOAD_REG	比较器 COMP0 的装载同步寄存器	0x0050	WT
比较器 COMP*1* 的控制和配置寄存器			
SYSTIMER_TARGET1_HI_REG	待装载至比较器 COMP1 的报警值，高 20 位	0x0024	R/W
SYSTIMER_TARGET1_LO_REG	待装载至比较器 COMP1 的报警值，低 32 位	0x0028	R/W
SYSTIMER_TARGET1_CONF_REG	配置比较器 COMP1 的报警模式	0x0038	R/W
SYSTIMER_COMP1_LOAD_REG	比较器 COMP1 的装载同步寄存器	0x0054	WT
比较器 COMP*2* 的控制和配置寄存器			
SYSTIMER_TARGET2_HI_REG	待装载至比较器 COMP2 的报警值，高 20 位	0x002C	R/W
SYSTIMER_TARGET2_LO_REG	待装载至比较器 COMP2 的报警值，低 32 位	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	配置比较器 COMP2 的报警模式	0x003C	R/W
SYSTIMER_COMP2_LOAD_REG	比较器 COMP2 的装载同步寄存器	0x0058	WT
中断寄存器			
SYSTIMER_INT_ENA_REG	系统定时器的中断使能寄存器	0x0064	R/W
SYSTIMER_INT_RAW_REG	系统定时器的原始中断寄存器	0x0068	R/WTC/SS
SYSTIMER_INT_CLR_REG	系统定时器的中断清除寄存器	0x006C	WT

名称	描述	地址	访问
SYSTIMER_INT_ST_REG	系统定时器的中断状态寄存器	0x0070	RO
COMP*0* 状态寄存器			
SYSTIMER_REAL_TARGET0_LO_REG	COMP0 的实际报警值，低 32 位	0x0074	RO
SYSTIMER_REAL_TARGET0_HI_REG	COMP0 的实际报警值，高 20 位	0x0078	RO
COMP*1* 状态寄存器			
SYSTIMER_REAL_TARGET1_LO_REG	COMP1 的实际报警值，低 32 位	0x007C	RO
SYSTIMER_REAL_TARGET1_HI_REG	COMP1 的实际报警值，高 20 位	0x0080	RO
COMP*2* 状态寄存器			
SYSTIMER_REAL_TARGET2_LO_REG	COMP2 的实际报警值，低 32 位	0x0084	RO
SYSTIMER_REAL_TARGET2_HI_REG	COMP2 的实际报警值，高 20 位	0x0088	RO
版本寄存器			
SYSTIMER_DATE_REG	版本控制寄存器	0x00FC	R/W

16.9 寄存器

本小节的所有地址均为相对于系统定时器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

Register 16.1. SYSTIMER_CONF_REG (0x0000)

Bit	Field	Reset
31	SYSTIMER_CLK_EN	0
30	SYSTIMER_TIMER_UNIT0_WORK_EN	1
29	SYSTIMER_TIMER_UNIT1_WORK_EN	0
28	SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN	0
27	SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN	0
26	SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN	1
25	SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN	1
24	SYSTIMER_TARGET0_WORK_EN	0
23	SYSTIMER_TARGET1_WORK_EN	0
22	SYSTIMER_TARGET2_WORK_EN	0
21:2	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1	SYSTIMER_ETM_EN	0
0	(reserved)	0

SYSTIMER_ETM_EN 配置是否使能生成 ETM 事件。
0：无效
1：使能生成 ETM 事件
(R/W)

SYSTIMER_TARGET2_WORK_EN 配置是否使能 COMP2。
0：不使能
1：使能 COMP2
(R/W)

SYSTIMER_TARGET1_WORK_EN 配置是否使能 COMP1。具体配置值请见 SYSTIMER_TARGET2_WORK_EN。(R/W)

SYSTIMER_TARGET0_WORK_EN 配置是否使能 COMP0。具体配置值请见 SYSTIMER_TARGET2_WORK_EN。(R/W)

SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT1 是否也将停止工作。
0：无效
1：计数器 UNIT1 停止工作
(R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT1 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

见下页...

Register 16.1. SYSTIMER_CONF_REG (0x0000)

... 接上页

SYSTIMER_TIMER_UNIT1_WORK_EN 配置是否使能计数器 UNIT1。

0：无效

1：使能计数器 UNIT1

(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN 配置是否使能计数器 UNIT0。

0：无效

1：使能计数器 UNIT0

(R/W)

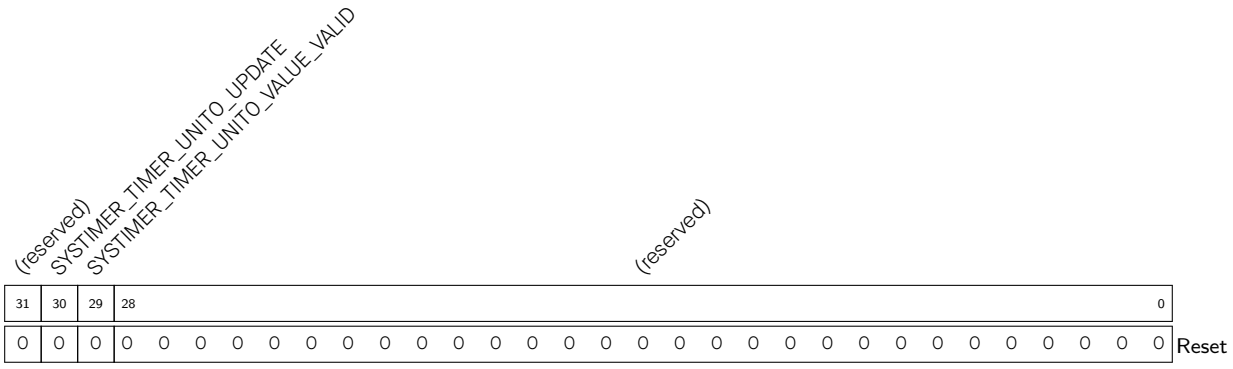
SYSTIMER_CLK_EN 配置寄存器时钟门控。

1：持续开启读写寄存器的时钟。

0：只在读写寄存器时打开所需时钟。

(R/W)

Register 16.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID 表示计数器 UNIT0 的值是否已同步读取至寄存器，且读值有效。

0：UNIT0 的值未同步读取至寄存器，读值无效

1：UNIT0 的值已同步读取至寄存器，读值有效

(R/SS/WTC)

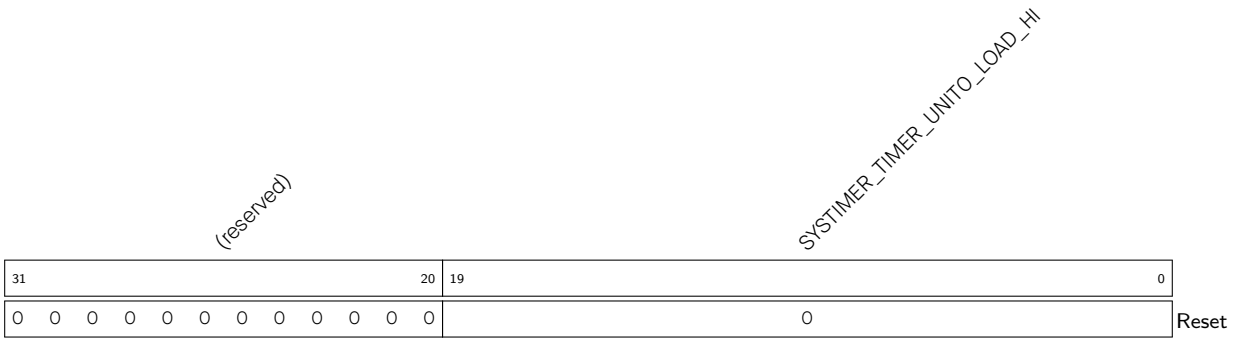
SYSTIMER_TIMER_UNIT0_UPDATE 配置是否更新计数器 UNIT0，即读取计数器 UNIT0 的值到寄存器 SYSTIMER_TIMER_UNIT0_VALUE_HI 和 SYSTIMER_TIMER_UNIT0_VALUE_LO。

0：无效

1：更新计数器 UNIT0

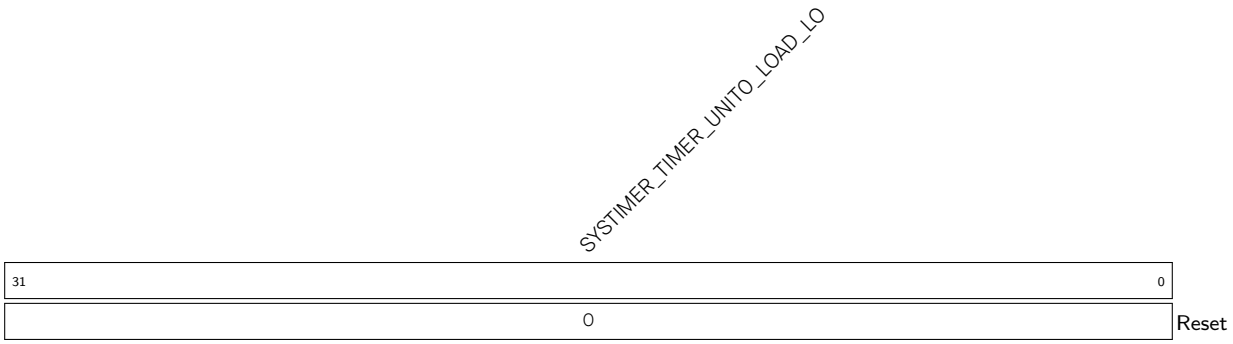
(WT)

Register 16.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



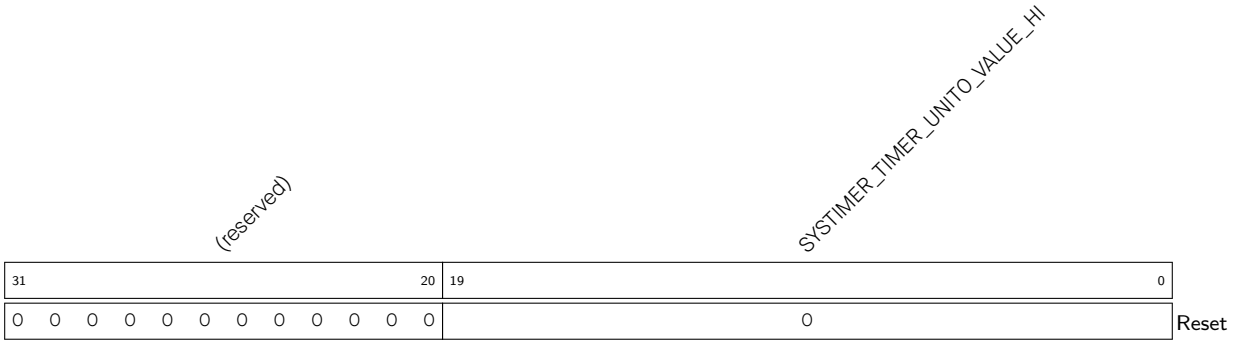
SYSTIMER_TIMER_UNIT0_LOAD_HI 配置待装载至计数器 UNIT0 的值，高 20 位。(R/W)

Register 16.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



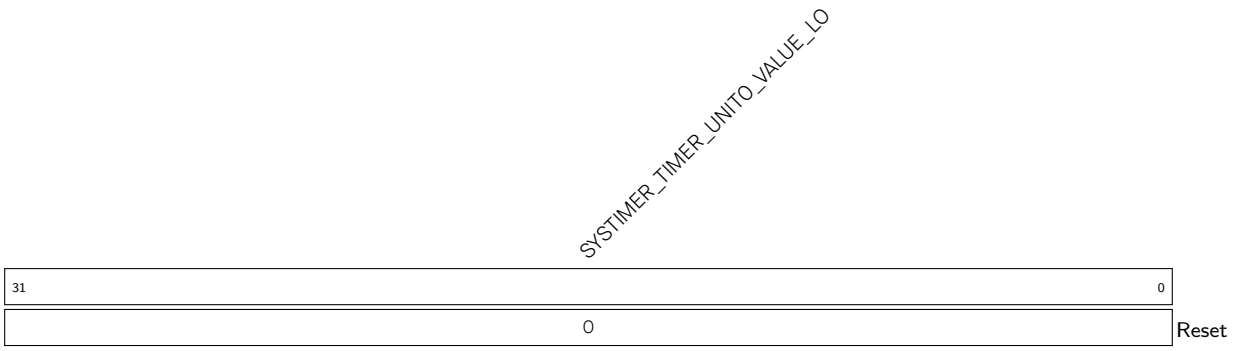
SYSTIMER_TIMER_UNIT0_LOAD_LO 配置待装载至计数器 UNIT0 的值，低 32 位。(R/W)

Register 16.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



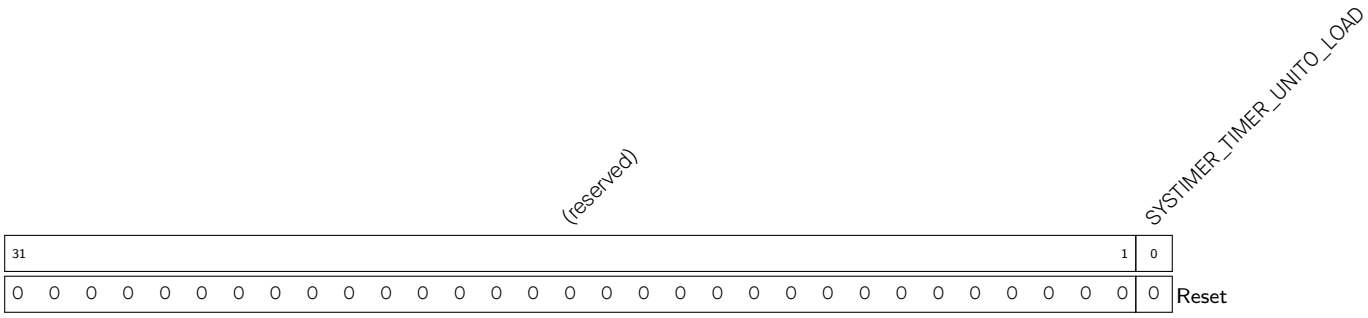
SYSTIMER_TIMER_UNIT0_VALUE_HI 表示计数器 UNIT0 的读数，高 20 位。(RO)

Register 16.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



SYSTIMER_TIMER_UNIT0_VALUE_LO 表示计数器 UNIT0 的读数，低 32 位。(RO)

Register 16.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



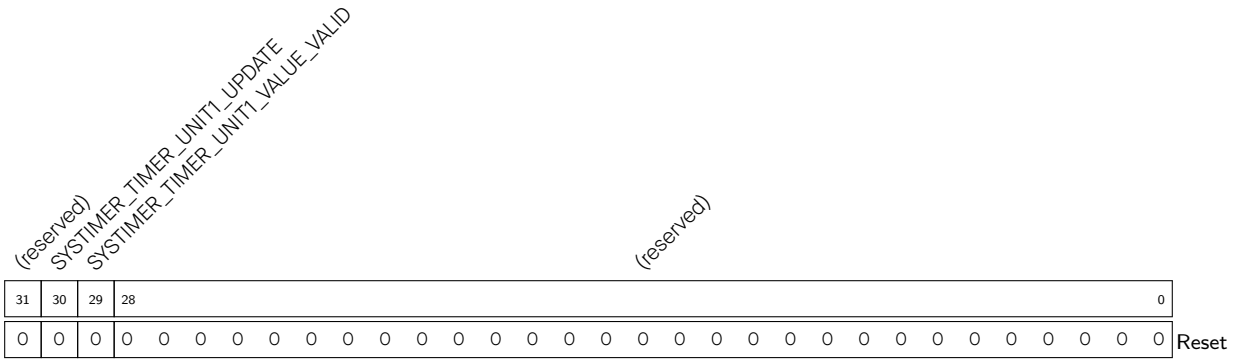
SYSTIMER_TIMER_UNIT0_LOAD 配置是否重新装载计数器 UNIT0，即是否重新装载寄存器 SYSTIMER_TIMER_UNIT0_LOAD_HI 和 SYSTIMER_TIMER_UNIT0_LOAD_LO 的值到计数器 UNIT0。

0：无效

1：重新装载计数器 UNIT0

(WT)

Register 16.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID 表示计数器 UNIT1 的值已同步读取至寄存器，且读值有效。(R/SS/WTC)

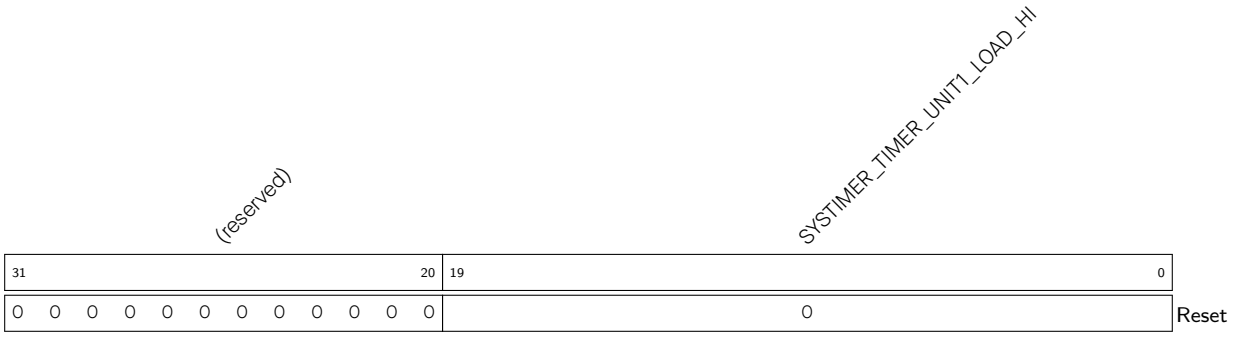
SYSTIMER_TIMER_UNIT1_UPDATE 配置是否更新计数器 UNIT1，即将计数器 UNIT1 的值读取到寄存器 SYSTIMER_TIMER_UNIT1_VALUE_HI 和 SYSTIMER_TIMER_UNIT1_VALUE_LO。

0：无效

1：更新计数器 UNIT1

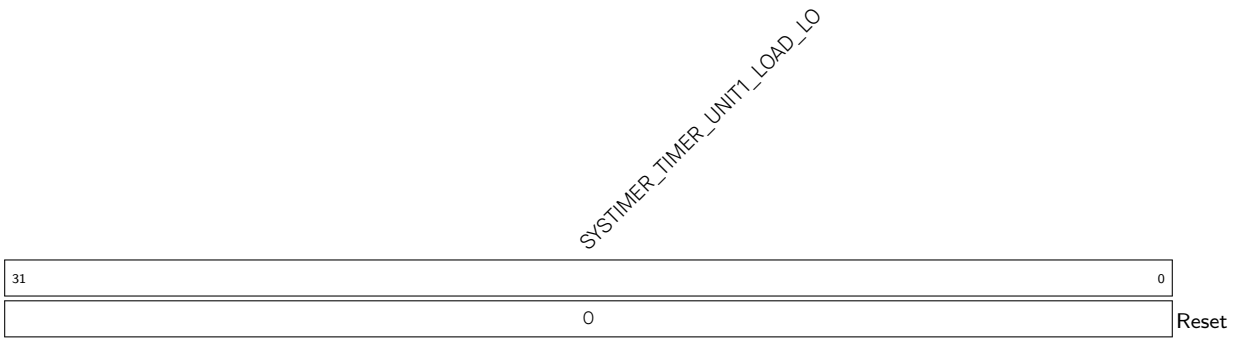
(WT)

Register 16.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



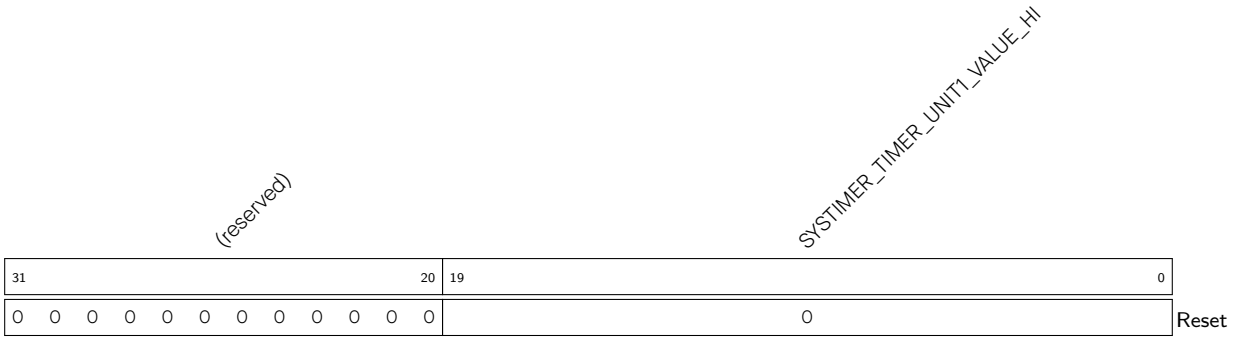
SYSTIMER_TIMER_UNIT1_LOAD_HI 配置待装载至计数器 UNIT1 的值，高 20 位。(R/W)

Register 16.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



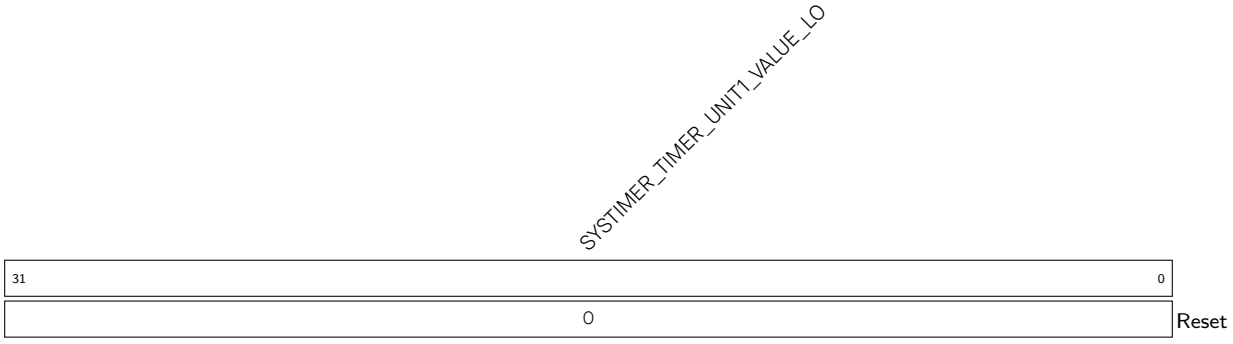
SYSTIMER_TIMER_UNIT1_LOAD_LO 配置待装载至计数器 UNIT1 的值，低 32 位。(R/W)

Register 16.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



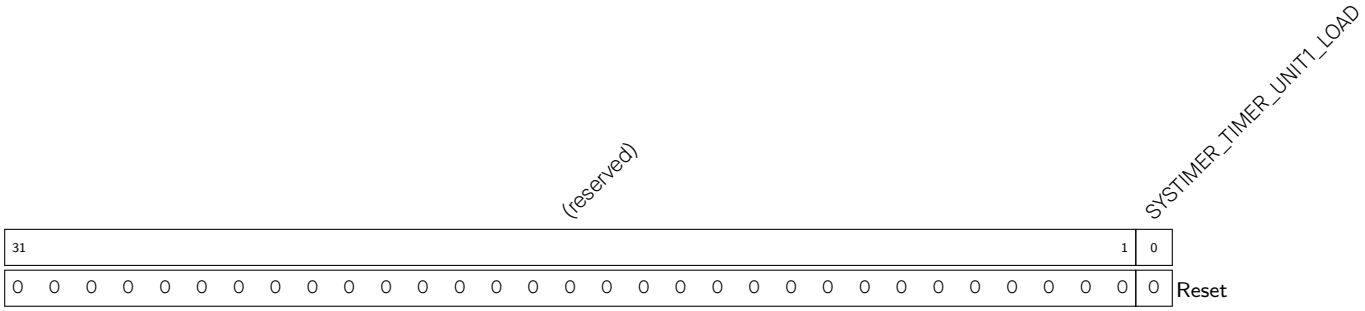
SYSTIMER_TIMER_UNIT1_VALUE_HI 表示计数器 UNIT1 的读数，高 20 位。(RO)

Register 16.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



SYSTIMER_TIMER_UNIT1_VALUE_LO 表示计数器 UNIT1 的读数，低 32 位。(RO)

Register 16.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



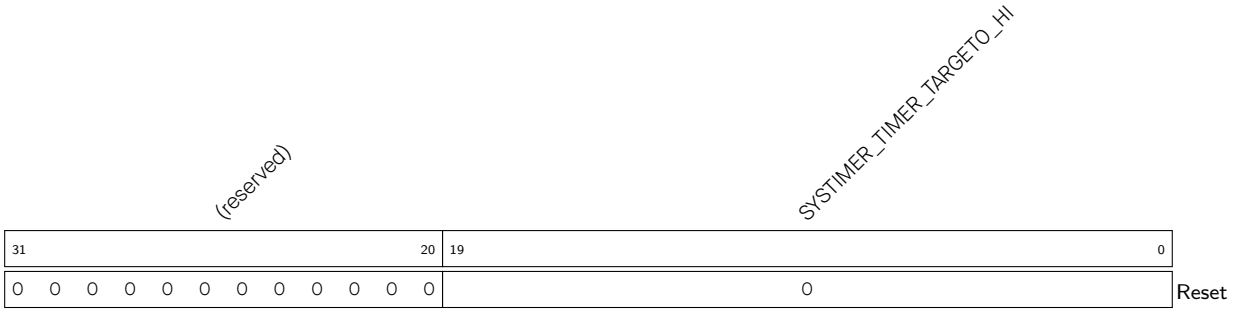
SYSTIMER_TIMER_UNIT1_LOAD 配置是否重新装载计数器 UNIT1，即是否重新装载寄存器 SYSTIMER_TIMER_UNIT1_LOAD_HI 和 SYSTIMER_TIMER_UNIT1_LOAD_LO 的值到计数器 UNIT1。

0：无效

1：重新装载计数器 UNIT1

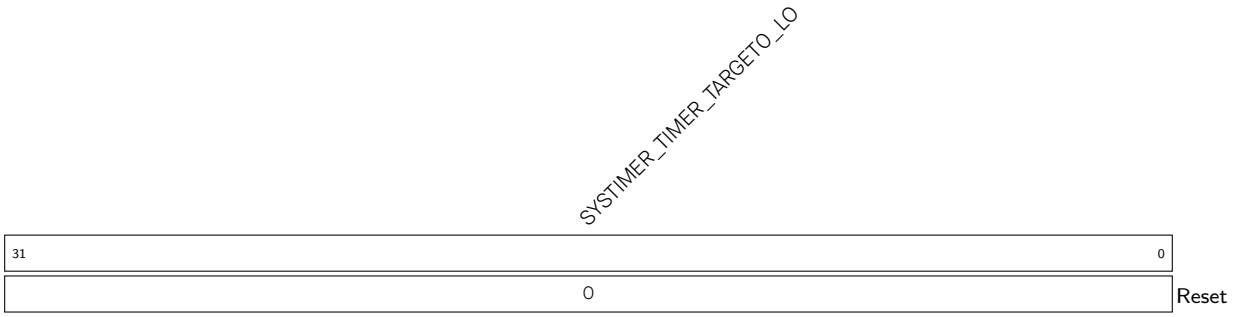
(WT)

Register 16.14. SYSTIMER_TARGET0_HI_REG (0x001C)



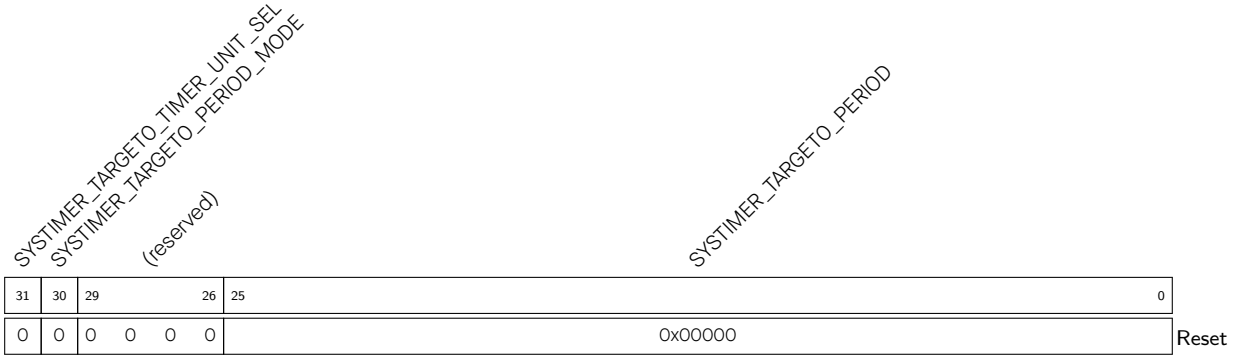
SYSTIMER_TIMER_TARGET0_HI 配置待装载至 COMP0 的报警值，高 20 位。(R/W)

Register 16.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO 配置待装载至 COMP0 的报警值，低 32 位。(R/W)

Register 16.16. SYSTIMER_TARGET0_CONF_REG (0x0034)



SYSTIMER_TARGET0_PERIOD 配置待装载至 COMP0 的报警周期。(R/W)

SYSTIMER_TARGET0_PERIOD_MODE 选择比较器 COMP0 生成报警的模式。

0：单次报警模式

1：周期报警模式

(R/W)

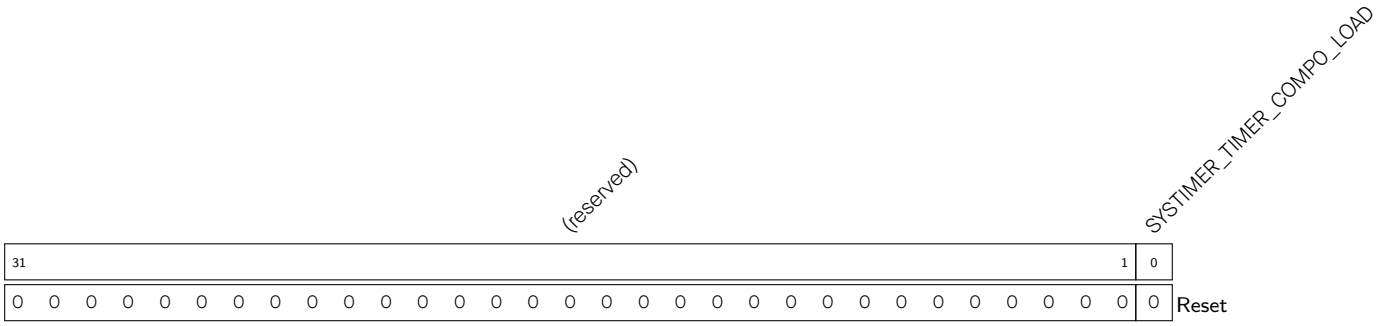
SYSTIMER_TARGET0_TIMER_UNIT_SEL 选择要与 COMP0 比较的计数器。

0：与计数器 UNIT*0* 的计数值进行比较

1：与计数器 UNIT*1* 的计数值进行比较

(R/W)

Register 16.17. SYSTIMER_COMP0_LOAD_REG (0x0050)



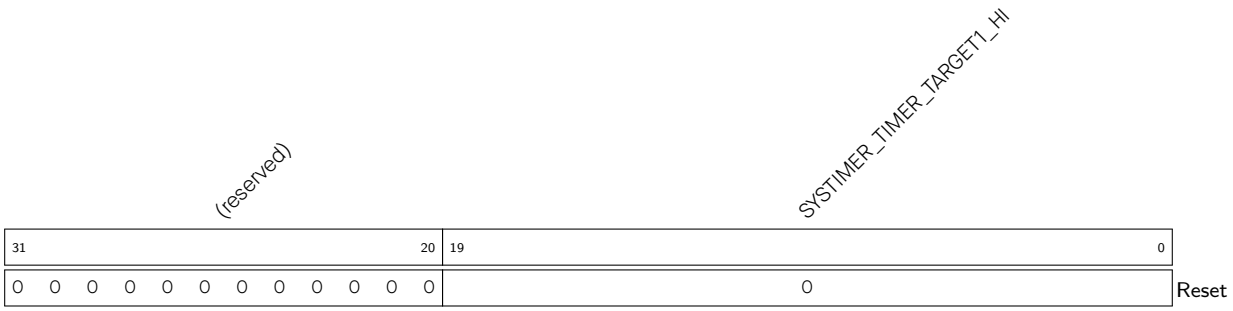
SYSTIMER_TIMER_COMP0_LOAD 配置是否使能比较器 COMP0 同步，即是否重新装载报警值或报警周期到 COMP0。

0：无效

1：使能比较器 COMP0 同步

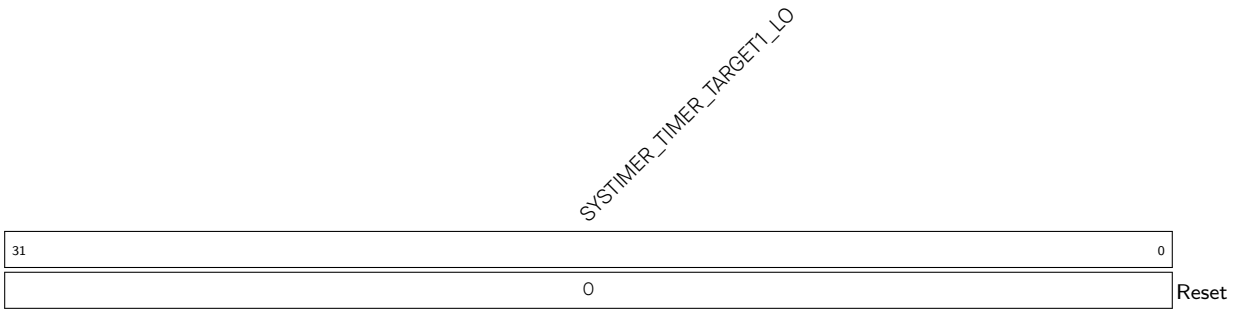
(WT)

Register 16.18. SYSTIMER_TARGET1_HI_REG (0x0024)



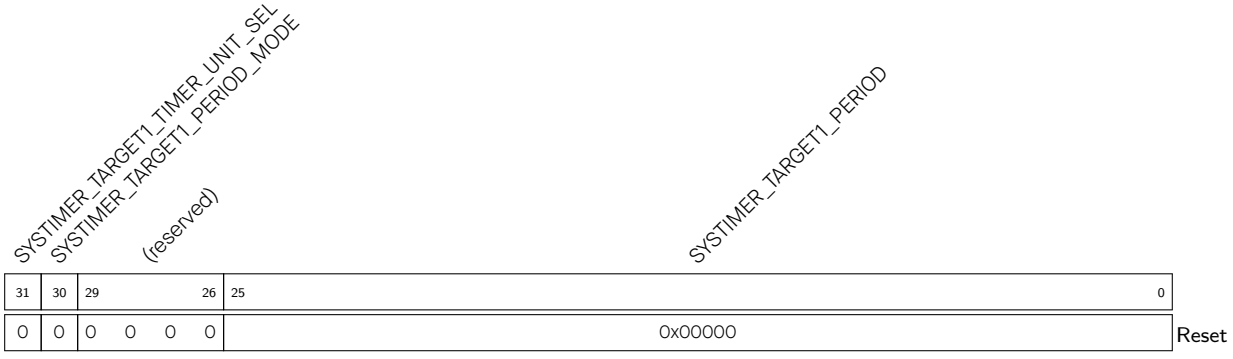
SYSTIMER_TIMER_TARGET1_HI 配置待装载至 COMP1 的报警值，高 20 位。(R/W)

Register 16.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO 配置待装载至 COMP1 的报警值，低 32 位。(R/W)

Register 16.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

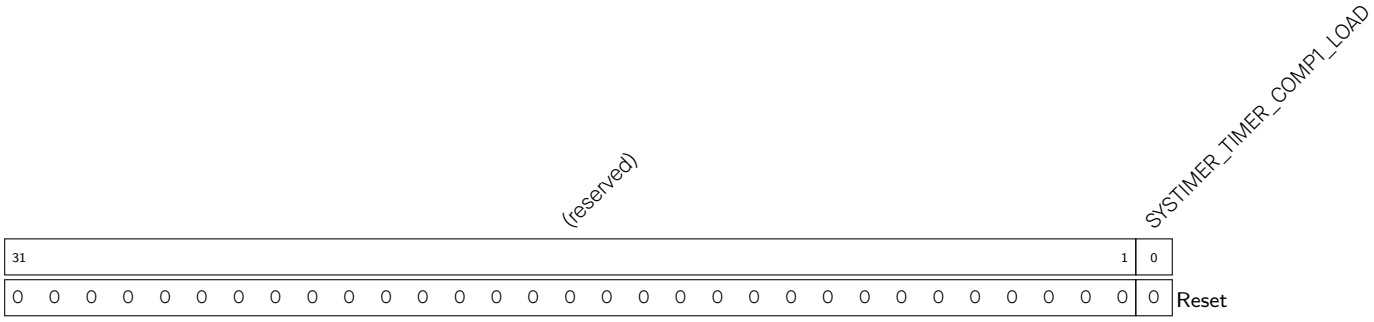


SYSTIMER_TARGET1_PERIOD 配置待装载至 COMP1 的报警周期。(R/W)

SYSTIMER_TARGET1_PERIOD_MODE 选择比较器 COMP1 生成报警的模式。具体配置值请见 SYSTIMER_TARGET0_PERIOD_MODE。(R/W)

SYSTIMER_TARGET1_TIMER_UNIT_SEL 选择要与 COMP1 比较的计数器。具体配置值请见 SYSTIMER_TARGET0_TIMER_UNIT_SEL。(R/W)

Register 16.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



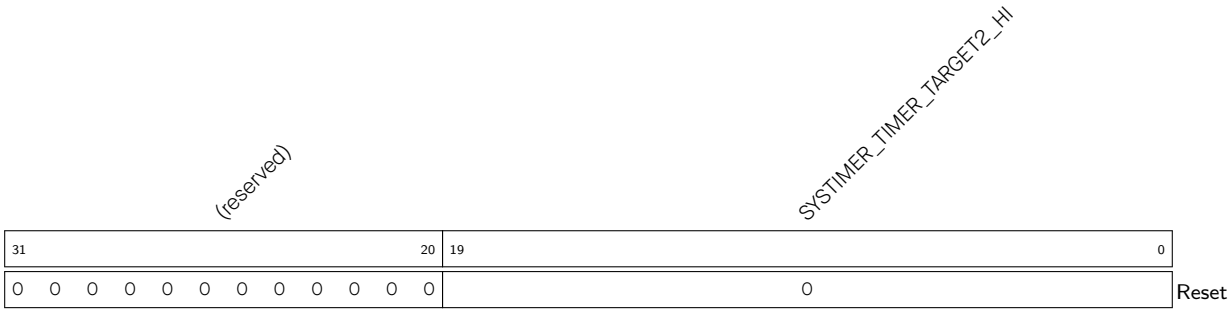
SYSTIMER_TIMER_COMP1_LOAD 配置是否使能比较器 COMP1 同步，即是否重新装载报警值或报警周期到 COMP1。

0：无效

1：使能比较器 COMP1 同步

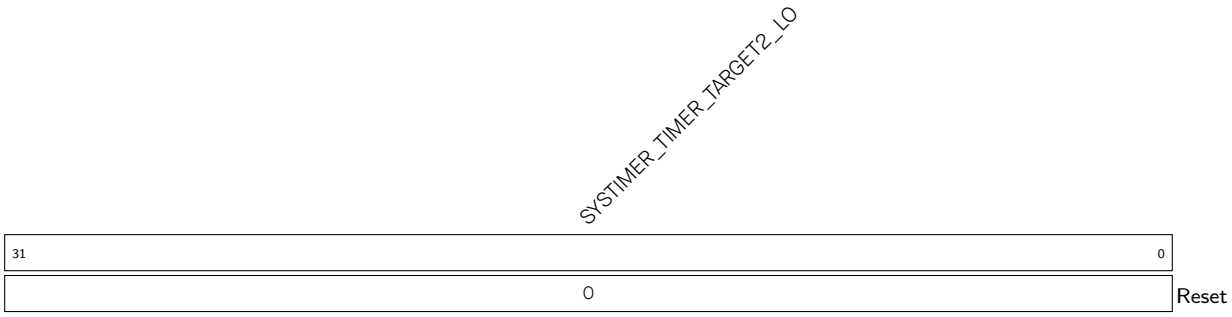
(WT)

Register 16.22. SYSTIMER_TARGET2_HI_REG (0x002C)



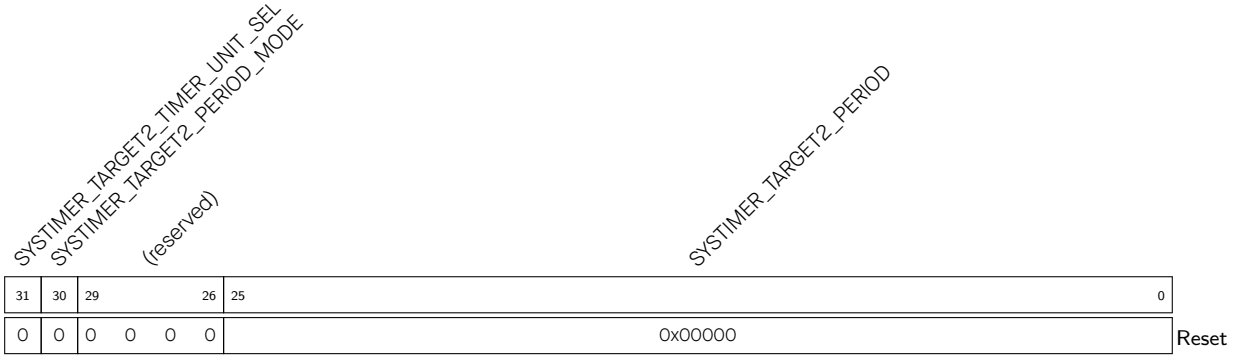
SYSTIMER_TIMER_TARGET2_HI 配置待装载至比较器 COMP2 的报警值，高 20 位。(R/W)

Register 16.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO 配置待装载至比较器 COMP2 的报警值，低 32 位。(R/W)

Register 16.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

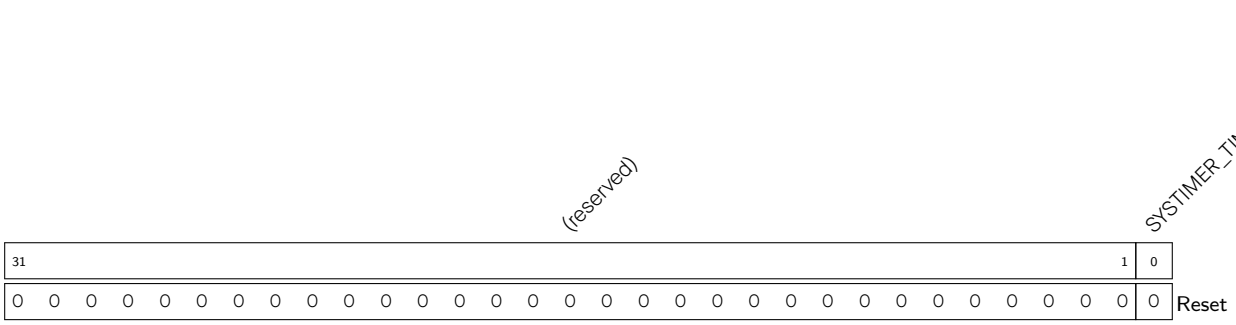


SYSTIMER_TARGET2_PERIOD 配置待装载至 COMP2 的报警周期。(R/W)

SYSTIMER_TARGET2_PERIOD_MODE 选择比较器 COMP2 生成报警的模式。具体配置值请见 SYSTIMER_TARGETO_PERIOD_MODE。(R/W)

SYSTIMER_TARGET2_TIMER_UNIT_SEL 选择要与 COMP2 比较的计数器。具体配置值请见 SYSTIMER_TARGETO_TIMER_UNIT_SEL。(R/W)

Register 16.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



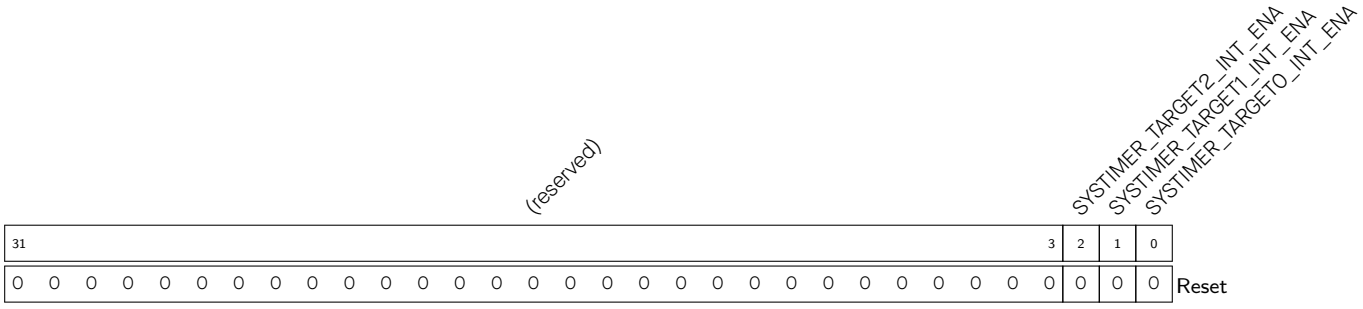
SYSTIMER_TIMER_COMP2_LOAD 配置是否使能比较器 COMP2 同步，即是否重新装载报警值或报警周期到 COMP2。

0：无效

1：使能比较器 COMP2 同步

(WT)

Register 16.26. SYSTIMER_INT_ENA_REG (0x0064)

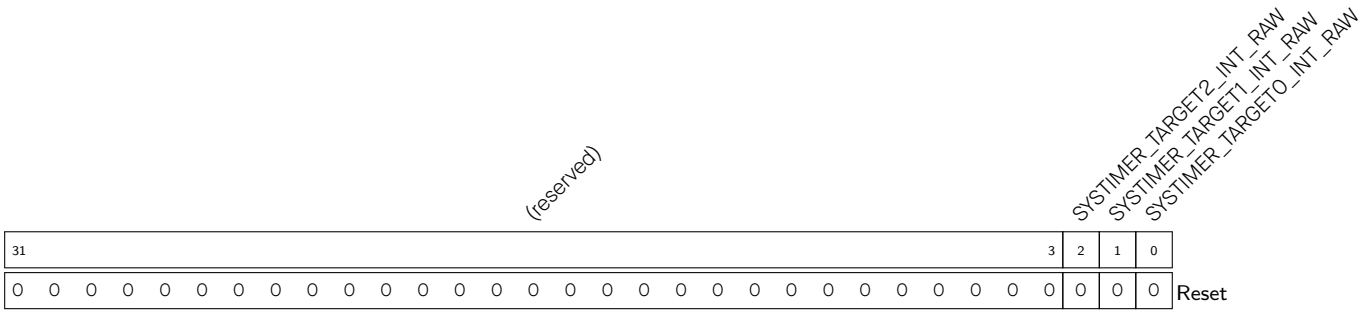


SYSTIMER_TARGET0_INT_ENA 写 1 使能 SYSTIMER_TARGET0_INT。(R/W)

SYSTIMER_TARGET1_INT_ENA 写 1 使能 SYSTIMER_TARGET1_INT。(R/W)

SYSTIMER_TARGET2_INT_ENA 写 1 使能 SYSTIMER_TARGET2_INT。(R/W)

Register 16.27. SYSTIMER_INT_RAW_REG (0x0068)

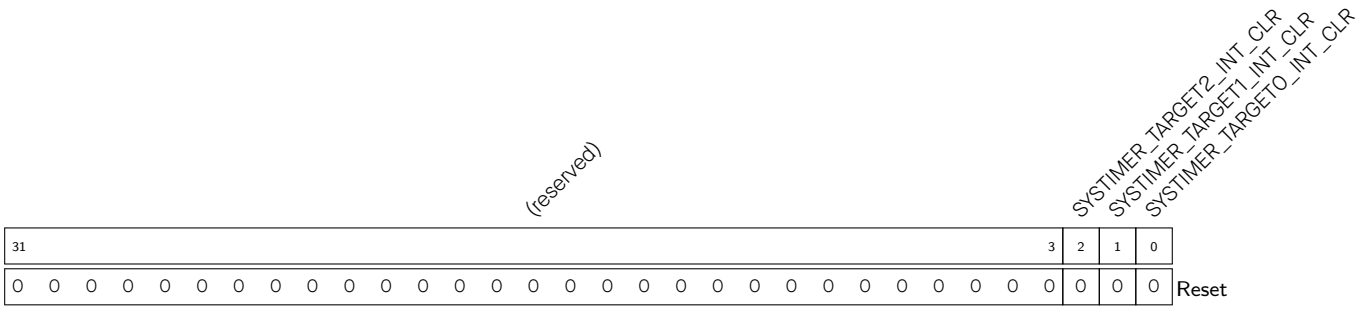


SYSTIMER_TARGET0_INT_RAW SYSTIMER_TARGET0_INT 的原始中断状态。(R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW SYSTIMER_TARGET1_INT 的原始中断状态。(R/WTC/SS)

SYSTIMER_TARGET2_INT_RAW SYSTIMER_TARGET2_INT 的原始中断状态。(R/WTC/SS)

Register 16.28. SYSTIMER_INT_CLR_REG (0x006C)

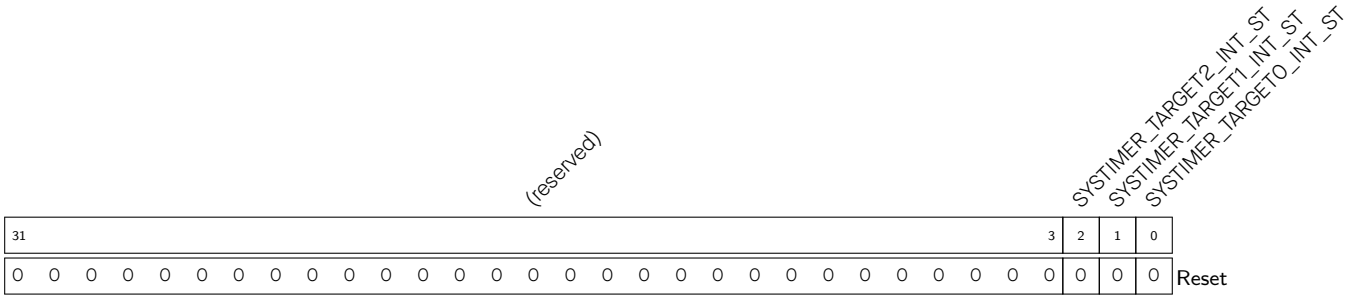


SYSTIMER_TARGET0_INT_CLR 写 1 清除 SYSTIMER_TARGET0_INT。(WT)

SYSTIMER_TARGET1_INT_CLR 写 1 清除 SYSTIMER_TARGET1_INT。(WT)

SYSTIMER_TARGET2_INT_CLR 写 1 清除 SYSTIMER_TARGET2_INT。(WT)

Register 16.29. SYSTIMER_INT_ST_REG (0x0070)

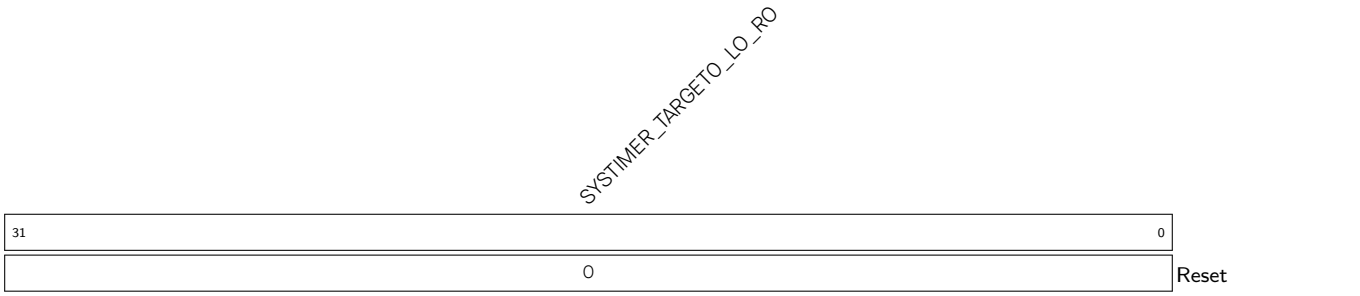


SYSTIMER_TARGET0_INT_ST SYSTIMER_TARGET0_INT 的屏蔽中断状态。(RO)

SYSTIMER_TARGET1_INT_ST SYSTIMER_TARGET1_INT 的屏蔽中断状态。(RO)

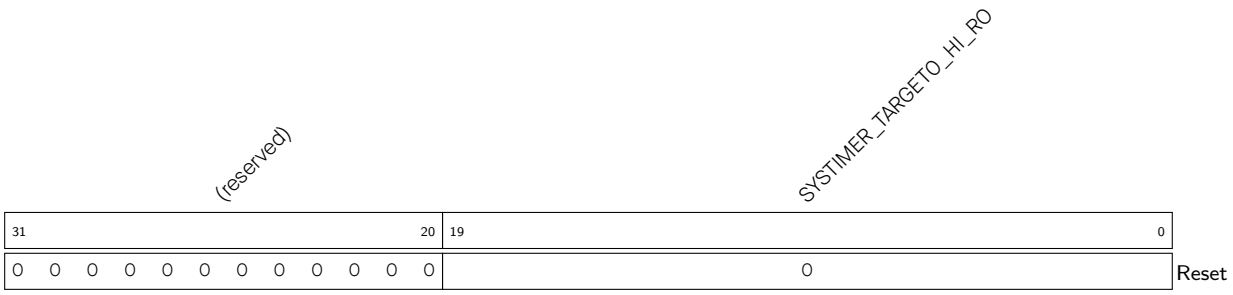
SYSTIMER_TARGET2_INT_ST SYSTIMER_TARGET2_INT 的屏蔽中断状态。(RO)

Register 16.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)



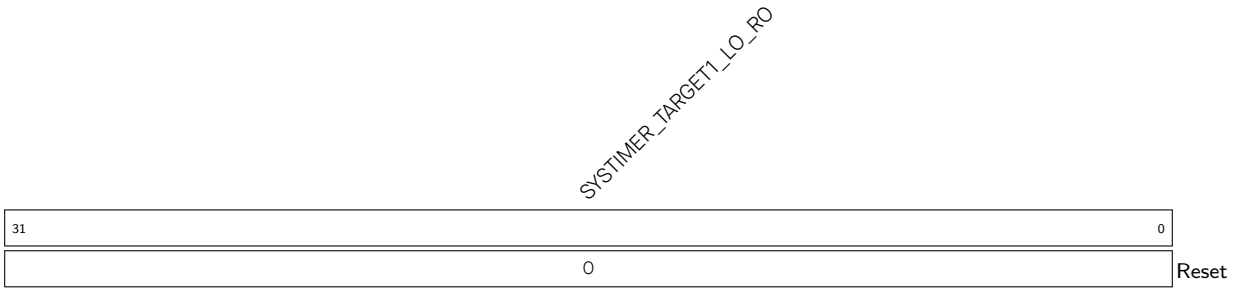
SYSTIMER_TARGET0_LO_RO 表示 COMP0 的实际报警值，低 32 位。(RO)

Register 16.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)



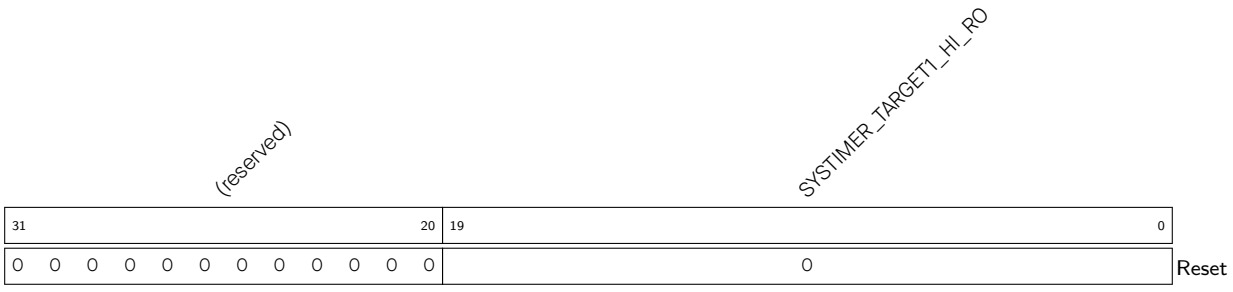
SYSTIMER_TARGET0_HI_RO 表示 COMP0 的实际报警值，高 20 位。(RO)

Register 16.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)



SYSTIMER_TARGET1_LO_RO 表示 COMP1 的实际报警值，低 32 位。(RO)

Register 16.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)



SYSTIMER_TARGET1_HI_RO 表示 COMP1 的实际报警值，高 20 位。(RO)

Register 16.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)



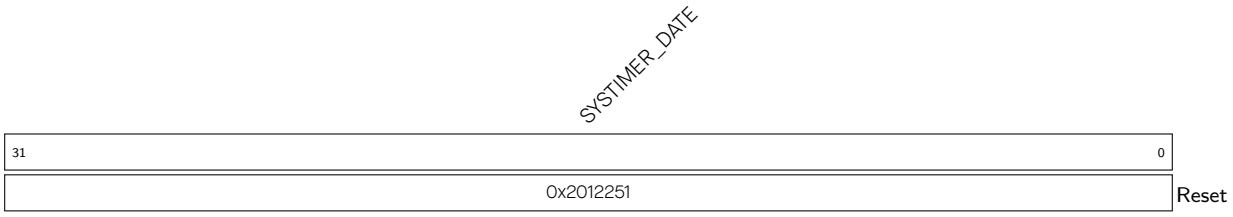
SYSTIMER_TARGET2_LO_RO 表示 COMP2 的实际报警值，低 32 位。(RO)

Register 16.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)



SYSTIMER_TARGET2_HI_RO 表示 COMP2 的实际报警值，高 20 位。(RO)

Register 16.36. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE 版本控制寄存器 (R/W)

第 17 章

定时器组 (TIMG)

17.1 概述

通用定时器可用于准确设定时间间隔、在一定间隔后触发（周期或非周期的）中断或充当硬件时钟。如图 17.1-1 所示，ESP32-P4 包含两个定时器组，即定时器组 0 和定时器组 1（下文用 TIMG*n* 表示，*n* 为 0 或 1）。每个定时器组有两个通用定时器（下文用 T*x* 表示，*x* 为 0 或 1）和一个主系统看门狗定时器 (MWDT)。通用定时器基于 16 位预分频器和 54 位可自动重新加载的可逆计数器。

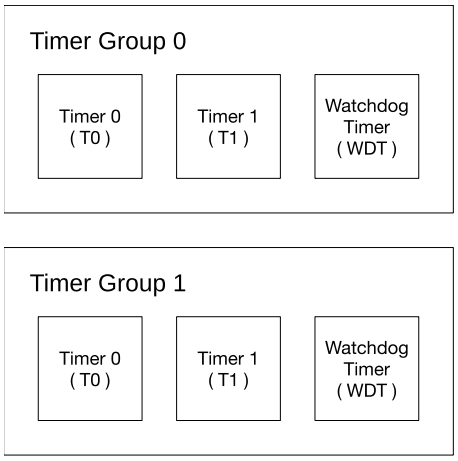


图 17.1-1. 定时器组概览

本章虽然包含主系统看门狗定时器的寄存器描述，但其功能描述请参阅章节 18 *看门狗定时器 (WDT)*。本章中“定时器”指代通用定时器。

17.2 主要特性

定时器具有如下功能：

- 54 位时基计数器，可配置成递增或递减
- 三个时钟源：PLL_F80M_CLK 或 XTAL_CLK 或 RC_FAST_CLK 时钟
- 16 位时钟预分频器，分频系数为 2 到 65536
- 可读取时基计数器的实时值
- 暂停和恢复时基计数器
- 可配置的警报生成机制
- 计数器值重新加载——报警时自动重新加载或软件控制的即时重新加载
- 时钟频率计算——基于晶振时钟计算导入 TIMG0 的被测时钟频率，该时钟可称之为 TIMG0_CALI_CLK

- 电平触发中断
- 支持多个 ETM 任务和事件

17.3 功能描述

图 17.3-1 为定时器组 TIMG*n* 的 T*x* 定时器。T*x* 包含一个 16 位整数预分频器、一个时基计数器和一个用于产生警报的比较器。

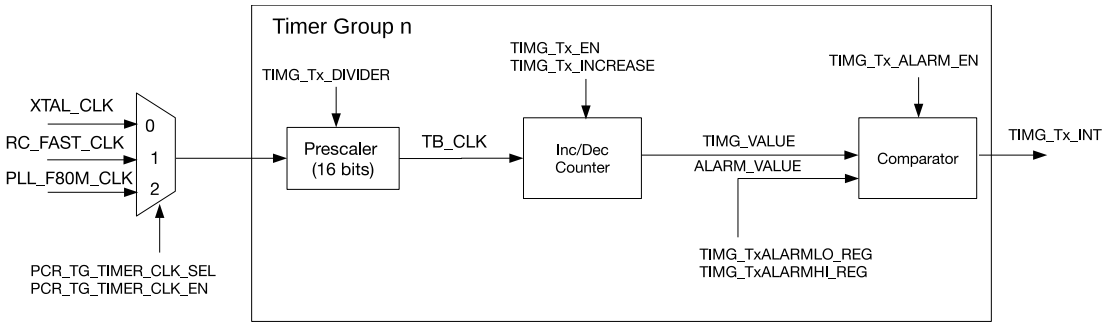


图 17.3-1. 定时器组架构

17.3.1 16 位预分频器与时钟选择器

以定时器组 TIMG*n* 中的 T*x* 定时器为例：

- 该定时器可通过配置 HP_SYS_CLKRST_PERI_CLK_CTRL20_REG 寄存器或 HP_SYS_CLKRST_PERI_CLK_CTRL21_REG 寄存器的 HP_SYS_CLKRST_TIMERGRP*n*_T*x*_SRC_SEL 字段来选择时钟源。该字段为 0 时，选择 XTAL_CLK 为时钟源；为 1 时，选择 RC_FAST_CLK 为时钟源；为 2 时，选择 PLL_F80M_CLK 为时钟源。
- 要开启所选时钟，需配置 HP_SYS_CLKRST_PERI_CLK_CTRL20_REG 寄存器或 HP_SYS_CLKRST_PERI_CLK_CTRL21_REG 寄存器的 HP_SYS_CLKRST_TIMERGRP*n*_T*x*_CLK_EN 字段为 1，要关闭时钟需清零该字段。所选时钟经 16 位预分频器分频，产生时基计数器使用的时基计数器时钟 (TB_CLK)。16 位预分频器的分频系数可通过 TIMG_T*x*_DIVIDER 字段配置。

TIMG_T*x*_DIVIDER 字段可配置为 0 ~ 65535 之间的任意值，分频系数的范围为 2 ~ 65536。具体来说：

- 当 TIMG_T*x*_DIVIDER 字段置 0 时：分频系数为 65536。
- 当 TIMG_T*x*_DIVIDER 字段置 1 时：分频系数为 2。
- 当 TIMG_T*x*_DIVIDER 字段置 2 时：分频系数仍为 2。
- 当 TIMG_T*x*_DIVIDER 字段置 3 ~ 65535 时：分频系数为 3 ~ 65535。

要更改 16 位预分频器，需先重新配置 TIMG_T*x*_DIVIDER 字段，再将 TIMG_T*x*_DIVCNT_RST 置 1，同时需要关闭定时器（即 TIMG_T*x*_EN 必须清零），否则会造成不可预知的结果。

17.3.2 54 位时基计数器

54 位时基计数器基于 TB_CLK，可通过 TIMG_T*x*_INCREASE 字段配置为递增或递减。时基计数器可通过置位或清零 TIMG_T*x*_EN 字段使能或关闭。使能时，时基计数器的值会在每个 TB_CLK 周期递增或递减。关闭时，时

基计数器暂停计数。注意，无论TIMG_Tx_EN 是否置位，TIMG_Tx_INCREASE 字段都可以更改，时基计数器可立即改变计数方向。

时基计数器 54 位定时器的当前值必须被锁入两个寄存器，才能被 CPU 读取（因为 CPU 为 32 位）。向 TIMG_TxUPDATE_REG 写入任意值时，54 位定时器的值开始被锁入寄存器 TIMG_TxLO_REG 和 TIMG_TxHI_REG，两个寄存器分别锁存低 32 位和高 22 位。当 TIMG_TxUPDATE_REG 被硬件清零，表明锁存操作已经完成，可以从这两个寄存器中读取当前计数值。在 TIMG_TxUPDATE_REG 被写入新值之前，保持寄存器 TIMG_TxLO_REG 和 TIMG_TxHI_REG 的值不变，以供 32 位的 CPU 读值。

17.3.3 警报生成

定时器可配置为在当前值与报警值相同时触发报警。报警会产生中断，同时（可选择）让定时器的当前值自动重新加载（详见第 17.3.4 节）。

54 位报警值可在 TIMG_TxALARMLO_REG 和 TIMG_TxALARMHI_REG 配置，两者分别代表报警值的低 32 位和高 22 位。但是，只有置位 TIMG_Tx_ALARM_EN 字段使能报警功能后，配置的报警值才会生效。为解决报警使能“过晚”（即报警使能时，定时器的值已过报警值），出现以下情况时硬件会立即触发报警：

- 可逆计数器向上计数时，定时器的当前值高于报警值（在一定范围内）
- 可逆计数器向下计数时，定时器的当前值低于报警值（在一定范围内）

表 17.3-1 和表 17.3-2 说明了定时器当前值、报警值与报警触发的关系。假设定时器当前值和报警值如下：

- TIMG_VALUE = {TIMG_TxHI_REG, TIMG_TxLO_REG}
- ALARM_VALUE = {TIMG_TxALARMHI_REG, TIMG_TxALARMLO_REG}

表 17.3-1. 可逆计数器向上计数时的报警场景

场景	范围	报警
1	ALARM_VALUE − TIMG_VALUE $> 2^{53}$	触发
2	$0 <$ ALARM_VALUE − TIMG_VALUE $\leq 2^{53}$	可逆计数器向上计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
3	$0 \leq$ TIMG_VALUE − ALARM_VALUE $< 2^{53}$	触发
4	TIMG_VALUE − ALARM_VALUE $\geq 2^{53}$	可逆计数器向上计数达到最大值时，重新开始从 0 向上计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

表 17.3-2. 可逆计数器向下计数时的报警场景

场景	范围	报警
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	触发
6	$0 <$ TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	可逆计数器向下计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
7	$0 \leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	触发
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	可逆计数器向下计数达到最小值时，重新开始从最大值向下计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

报警时，TIMG_Tx_ALARM_EN 字段自动清零，在下次置位 TIMG_Tx_ALARM_EN 前不会再次报警。

17.3.4 定时器重新加载

定时器重新加载指将定时器的低 32 位和高 22 位分别更新为 TIMG_T*x*_LOAD_LO 和 TIMG_T*x*_LOAD_HI 字段存储的重新加载值。但是，把重新加载值写入 TIMG_T*x*_LOAD_LO 和 TIMG_T*x*_LOAD_HI 字段不会改变定时器的当前值。写入的重新加载值会被定时器忽视，直到重新加载事件被触发。重新加载事件可由软件即时重新加载或报警时自动重新加载触发。

CPU 在寄存器 TIMG_T*x*LOAD_REG 写任意值会触发软件即时重新加载，定时器的当前值会立即改变。若此时 TIMG_T*x*_EN 是置位状态，定时器会继续从新数值开始递增或递减计数。若此时 TIMG_T*x*_ALARM_EN 置位，仍会根据表 17.3-1 或表 17.3-2 中的关系在对应时刻产生报警。若 TIMG_T*x*_EN 是清零状态，定时器将保持当前值，直至计数重新使能。

报警时，自动重新加载功能可让定时器在报警时重新加载，从重新加载值开始继续递增或递减计数。该功能通常用于周期性报警时重置定时器的值。要使能报警时自动重新加载，需将 TIMG_T*x*_AUTORELOAD 字段置 1。如未使能该功能，报警后定时器的值会在过报警值后继续递增或递减。

17.3.5 事件任务矩阵功能

在 ESP32-P4 中，TIMG*n* 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 TIMG*n* 的 ETM 任务，或者通过 TIMG*n* 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 TIMG*n* 相关的 ETM 任务和 ETM 事件。

TIMG*n* 可接收的 ETM 任务有：

- TG0_TASK_CNT_START_TIMER0：触发时启用时基计数器。
- TG1_TASK_CNT_START_TIMER0：触发时启用时基计数器。
- TG0_TASK_CNT_STOP_TIMER0：触发时关闭时基计数器。
- TG1_TASK_CNT_STOP_TIMER0：触发时关闭时基计数器。

> **说明：**
> 以上两个 ETM 任务与 APB 配置 TIMG_T0_EN 实现的功能相同。当这些操作同时发生时，各操作优先级由高到低如下：
>
> 1. TG0_TASK_CNT_START_TIMER0 与 TG1_TASK_CNT_START_TIMER0：触发时启用时基计数器
> 2. TG0_TASK_CNT_STOP_TIMER0 与 TG1_TASK_CNT_STOP_TIMER0：触发时关闭时基计数器
> 3. APB 方式配置 TIMG_T0_EN：启用或关闭时基计数器

- TG0_TASK_ALARM_START_TIMER0：触发时启用警报生成。
- TG1_TASK_ALARM_START_TIMER0：触发时启用警报生成。

> **说明：**
> 通过 APB 方式配置 TIMG_T0_ALARM_EN 以及硬件事件触发也可以启用警报生成功能。当这些操作同时发生时，各个操作的优先级从高到低依次为：
>
> 1. TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0：触发时启用警报生成
> 2. 警报事件：触发时关闭警报生成
> 3. APB 方式配置 TIMG_T0_ALARM_EN：启用/关闭警报生成

- TGO_TASK_CNT_CAP_TIMER0：触发时会将当前计数器的值更新至 TIMG_TOLO_REG 和 TIMG_TOHI_REG 寄存器。
- TG1_TASK_CNT_CAP_TIMER0：触发时会将当前计数器的值更新至 TIMG_TOLO_REG 和 TIMG_TOHI_REG 寄存器。
- TGO_TASK_CNT_RELOAD_TIMER0：触发时会用存储在 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 中的重新加载值覆盖当前计数器值。
- TG1_TASK_CNT_RELOAD_TIMER0：触发时会用存储在 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 中的重新加载值覆盖当前计数器值。

TIMG*n* 可产生的 ETM 事件有：

- TGO_EVT_CNT_CMP_TIMER0：表示 TIMG0 的 T0 定时器中断事件。
- TG1_EVT_CNT_CMP_TIMER0：表示 TIMG1 的 T0 定时器中断事件。

> **说明：**
> 只有当 TIMG*n*_ETM_EN 设置为 1 时，TIMG*n* 的 ETM 任务和事件才会生效。

在具体应用中，TIMG*n* 的 ETM 事件可以用来触发 TIMG*n* 的 ETM 任务。

例如，TGO_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件分别可以触发 TGO_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务，从而实现周期性报警。具体配置步骤请参考 17.4.4 通过 ETM 设置定时器用于周期性报警。

17.3.6 TIMG0_CALI_CLK 频率计算

定时器组可以以 XTAL_CLK 为基准时钟，计算 TIMG0_CALI_CLK 的实际频率。计算方式如下：

1. 通过周期性或单次计算的方式启动频率计算模块（详见章节 17.4.5）；
2. 在接收到计算开始的信号后，两个分别工作在 XTAL_CLK 以及 TIMG0_CALI_CLK 的计数器同时开始计数，当 TIMG0_CALI_CLK 的计数器达到设定的计算周期 C0 时，同时停止两个计数器；
3. 通过 XTAL_CLK 的计数器值 C1 即可计算 TIMG0_CALI_CLK 的时钟频率：$f_cali = \frac{C0 \times f_XTAL_CLK}{C1}$

17.3.7 中断

ESP32-P4 的 TIMG*n* 可以生成以下中断信号并发送给中断矩阵。

- TG*n*_T*x*_INT
- TG*n*_WDT_INT

这些中断信号由 TIMG*n* 的内部中断源生成。表 17.3-3 列出了 TIMG*n* 的内部中断源、中断触发条件、以及生成的中断信号。

表 17.3-3. TIMG*n* 的内部中断源

内部中断源	触发条件	中断信号
TIMG_T*x*_INT	T*x* 定时器产生告警	TG*n*_T*x*_INT
TIMG_WDT_INT	看门狗定时器产生告警	TG*n*_WDT_INT

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 17.5 寄存器列表。

17.4 配置与使用

17.4.1 定时器用作简单时钟

1. 配置时基计数器。
   - 配置 HP_SYS_CLKRST_PERI_CLK_CTRL21_REG 寄存器的 HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL 字段选择时钟源。
   - 置位 TIMG_T*x*_DIVIDER 配置 16 位预分频器。
   - 置位或清除 TIMG_T*x*_INCREASE 配置定时器方向。
   - 在 TIMG_T*x*_LOAD_LO 和 TIMG_T*x*_LOAD_HI 上写初始值设置定时器的初始值，然后在 TIMG_T*x*LOAD_REG 上写任意值将初始值重新加载进定时器。
2. 置位 TIMG_T*x*_EN 开启定时器。
3. 获得定时器的当前值。
   - 在 TIMG_T*x*UPDATE_REG 上写任意值锁存定时器的当前值。
   - 等待硬件将 TIMG_T*x*UPDATE_REG 清 0。
   - 从 TIMG_T*x*LO_REG 和 TIMG_T*x*HI_REG 读取锁存的定时器值。

17.4.2 定时器用于单次报警

1. 按照第 17.4.1 节的第 1 步配置时基计数器。
2. 配置报警。
   - 置位 TIMG_T*x*ALARMLO_REG 和 TIMG_T*x*ALARMHI_REG 配置报警值。
   - 置位 TIMG_T*x*_INT_ENA 使能中断。
3. 清零 TIMG_T*x*_AUTORELOAD 关闭自动重新加载。
4. 置位 TIMG_T*x*_ALARM_EN 开启报警。
5. 处理报警中断。
   - 置位定时器在 TIMG_T*x*_INT_CLR 的对应位清除中断。
   - 清零 TIMG_T*x*_EN 关闭定时器。

17.4.3 通过 APB 设置定时器用于周期性报警

1. 按照第 17.4.1 节的第 1 步配置时基计数器。

2. 按照第 17.4.2 节的第 2 步配置报警。
3. 置位 TIMG_T*x*_AUTORELOAD 使能自动重新加载，将重新加载值写入 TIMG_T*x*_LOAD_LO 和 TIMG_T*x*_LOAD_HI。
4. 置位 TIMG_T*x*_ALARM_EN 开启报警。
5. 处理报警中断（每次报警时重复）。
   - 置位定时器在 TIMG_T*x*_INT_CLR 的对应位清除中断。
   - 如下一次报警需要新的报警值和重新加载值（即每次都有不同的报警间隔），则应根据需要重新配置 TIMG_T*x*ALARMLO_REG、TIMG_T*x*ALARMHI_REG、TIMG_T*x*_LOAD_LO 和 TIMG_T*x*_LOAD_HI 。否则，上述寄存器应保持不变。
   - 置位 TIMG_T*x*_ALARM_EN 重新使能报警。
6. （最后一次报警时）关闭定时器。
   - 置位定时器在 TIMG_T*x*_INT_CLR 的对应位清除中断。
   - 清零 TIMG_T*x*_EN 关闭定时器。

17.4.4 通过 ETM 设置定时器用于周期性报警

1. 使能 ETM 模块的时钟
2. 将 ETM 事件映射到相应的 ETM 任务（使用 ETM 事件触发对应的 ETM 任务）
   - 如果 TIMG_T0_AUTORELOAD 置 1，通过一个 ETM 通道映射 TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件到 TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务。
   - 如果 TIMG_T0_AUTORELOAD 置 0，除了映射 TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件到 TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务外，还需要用另一个 ETM 通道映射 TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件到 TG0_TASK_CNT_RELOAD_TIMER0 与 TG1_TASK_CNT_RELOAD_TIMER0 任务。
3. 选择使能一个或两个 ETM 通道。
4. 将 TIMER_ETM_EN 置 1 从而使能定时器组的 ETM 事件和任务。
5. 按照第 17.4.1 节的第 1 步配置时基计数器。
6. 按照第 17.4.2 节的第 2 步配置报警。
7. 通过 TIMG_T0_LOAD_LO 和 TIMG_T0_LOAD_HI 配置重新加载值。
8. 处理 TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1)
   - 当警报生成时，TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件也会产生，此时报警器关闭警报生成。
   - 如果 TIMG_T0_AUTORELOAD 为 1，重新加载值会覆盖当前计数器的值。TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 会重新使能警报生成。
   - 如果 TIMG_T0_AUTORELOAD 为 0，由于 TG0_TASK_CNT_RELOAD_TIMER0 与 TG1_TASK_CNT_RELOAD_TIMER0 任务，重新加载值会覆盖当前计数器的值。

TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务会重新使能警报生成。

9. （最后一次报警时）关闭定时器。
   - 关闭用于映射定时器组事件和任务的 ETM 通道。
   - 设置 TIMER_ETM_EN 为 0。
   - 置位定时器在 TIMG_T0_INT_CLR 的对应位清除中断。
   - 清零 TIMG_T0_EN 关闭定时器。

17.4.5 RTC_SLOW_CLK 频率计算

1. 单次计算
   - 设置 HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_EN 开启 TIMG0_CALI_CLK 时钟门控。
   - 设置 HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL 择一路时钟作为 TIMG0_CALI_CLK 的时钟源。
   - 设置 HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_DIV_NUM 输出整数分频后的 TIMG0_CALI_CLK。
   - 设置 TIMG_RTC_CALI_MAX 配置频率计算时间。
   - 清空 TIMG_RTC_CALI_START_CYCLING 选择单次校准模式，然后配置 TIMG_RTC_CALI_START 开启两个计数器。
   - 等待 TIMG_RTC_CALI_RDY 的值变为 1，读取 TIMG_RTC_CALI_VALUE 获取 XTAL_CLK 计数器值，根据章节 17.3.6 的公式计算 RTC_SLOW_CLK 频率。
2. 周期性计算
   - 设置 HP_SYS_CLKRST_TIMERGRP0_TGRT_CLK_SRC_SEL 选择需要计算频率的时钟（RTC_SLOW_CLK 的时钟源），设置 TIMG_RTC_CALI_MAX 配置频率计算时间。
   - 使能 TIMG_RTC_CALI_START_CYCLING，硬件将不间断进行频率计算过程。
   - 只要 TIMG_RTC_CALI_CYCLING_DATA_VLD 为 1，即表示 TIMG_RTC_CALI_VALUE 有效。
3. 超时
   如果 RTC_SLOW_CLK 的计数器没有在 TIMG_RTC_CALI_TIMEOUT_RST_CNT 的 XTAL_CLK 计数器内完成计数，将置位 TIMG_RTC_CALI_TIMEOUT 标记计算超时。

17.5 寄存器列表

本小节的所有地址均为相对于定时器组基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
定时器 0 控制和配置寄存器			
TIMG_T0CONFIG_REG	定时器 0 配置寄存器	0x0000	varies
TIMG_T0LO_REG	定时器 0 的当前值，低 32 位	0x0004	RO
TIMG_T0HI_REG	定时器 0 的当前值，高 22 位	0x0008	RO
TIMG_T0UPDATE_REG	写值将当前定时器的值复制到 TIMG_T0LO_REG 或 TIMG_T0HI_REG	0x000C	R/ W/ SC
TIMG_T0ALARMLO_REG	定时器 0 的报警值，低 32 位	0x0010	R/W
TIMG_T0ALARMHI_REG	定时器 0 的报警值，高 22 位	0x0014	R/W
TIMG_T0LOADLO_REG	定时器 0 的重新加载值，低 32 位	0x0018	R/W
TIMG_T0LOADHI_REG	定时器 0 的重新加载值，高 22 位	0x001C	R/W
TIMG_T0LOAD_REG	写值从 TIMG_T0LOADLO_REG 或 TIMG_T0LOADHI_REG 上加载定时器	0x0020	WT
定时器 1 控制和配置寄存器			
TIMG_T1CONFIG_REG	定时器 1 配置寄存器	0x0024	varies
TIMG_T1LO_REG	定时器 1 的当前值，低 32 位	0x0028	RO
TIMG_T1HI_REG	定时器 1 的当前值，高 22 位	0x002C	RO
TIMG_T1UPDATE_REG	写值将当前定时器的值复制到 TIMG_T1LO_REG 与 TIMG_T1HI_REG	0x0030	R/ W/ SC
TIMG_T1ALARMLO_REG	定时器 1 的报警值，低 32 位	0x0034	R/W
TIMG_T1ALARMHI_REG	定时器 1 的报警值，高 22 位	0x0038	R/W
TIMG_T1LOADLO_REG	定时器 1 的重新加载值，低 32 位	0x003C	R/W
TIMG_T1LOADHI_REG	定时器 1 的重新加载值，高 22 位	0x0040	R/W
TIMG_T1LOAD_REG	写值从 TIMG_T1LOADLO_REG 或 TIMG_T1LOADHI_REG 上加载定时器	0x0044	WT
看门狗定时器控制和配置寄存器			
TIMG_WDTCONFIG0_REG	看门狗定时器配置寄存器	0x0048	varies
TIMG_WDTCONFIG1_REG	看门狗定时器预分频器寄存器	0x004C	varies
TIMG_WDTCONFIG2_REG	看门狗定时器阶段 0 超时值	0x0050	R/W
TIMG_WDTCONFIG3_REG	看门狗定时器阶段 1 超时值	0x0054	R/W
TIMG_WDTCONFIG4_REG	看门狗定时器阶段 2 超时值	0x0058	R/W
TIMG_WDTCONFIG5_REG	看门狗定时器阶段 3 超时值	0x005C	R/W
TIMG_WDTFEED_REG	写值喂看门狗定时器	0x0060	WT
TIMG_WDTWPROTECT_REG	看门狗写保护寄存器	0x0064	R/W
RTC 校准控制和配置寄存器			
TIMG_RTCCALICFG_REG	RTC 校准配置寄存器	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC 校准配置寄存器 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC 校准配置寄存器 2	0x0080	varies

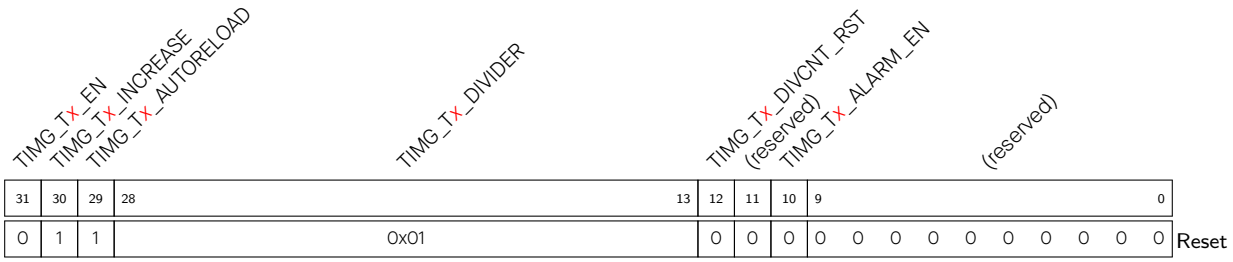
名称	描述	地址	访问
中断寄存器			
TIMG_INT_ENA_TIMERS_REG	中断使能位	0x0070	R/W
TIMG_INT_RAW_TIMERS_REG	原始中断状态	0x0074	R/ SS/ WTC
TIMG_INT_ST_TIMERS_REG	屏蔽中断状态	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	中断清除位	0x007C	WT
版本寄存器			
TIMG_NTIMERS_DATE_REG	版本控制寄存器	0x00F8	R/W
时钟配置寄存器			
TIMG_REGCLK_REG	定时器组时钟门控寄存器	0x00FC	R/W

17.6 寄存器

本小节的所有地址均为相对于定时器组基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 17.1. TIMG_TxCONFIG_REG (x: 0-1) (0x0000+0x24*x)



TIMG_Tx_ALARM_EN 配置是否使能定时器 Tx 的报警功能。报警时，此位自动清零。

0：禁用

1：使能

(R/W/SC)

TIMG_Tx_DIVCNT_RST 配置是否复位定时器 Tx 时钟分频器的计数器。

0：无效

1：复位

(WT)

TIMG_Tx_DIVIDER 表示定时器 Tx 时钟 (Tx_clk) 的预分频器值。(R/W)

TIMG_Tx_AUTORELOAD 配置是否使能定时器 Tx 报警时自动重新加载。

0：无效

1：使能

(R/W)

TIMG_Tx_INCREASE 配置定时器 Tx 的时基计数器的计数方向。

0：递减

1：递增

(R/W)

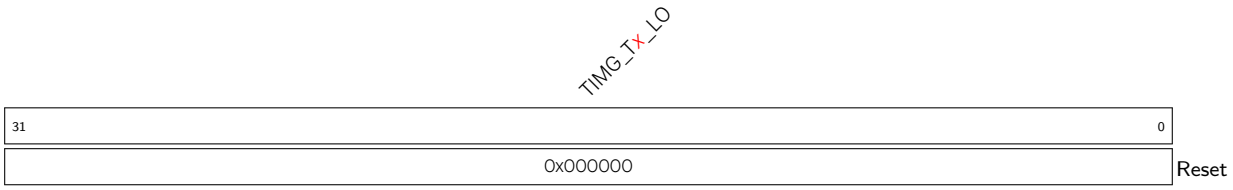
TIMG_Tx_EN 配置是否使能定时器 Tx 的时基计数器。

0：禁用

1：使能

(R/W/SS/SC)

Register 17.2. TIMG_TxLO_REG (x: 0-1) (0x0004+0x24*x)

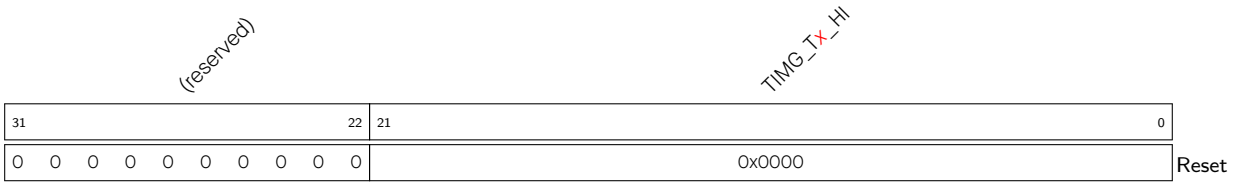


TIMG_Tx_LO 表示定时器 Tx 时基计数器的低 32 位的值。仅在 TIMG_TxUPDATE_REG 写值后有效。

单位：Tx_clk

(RO)

Register 17.3. TIMG_TxHI_REG (x: 0-1) (0x0008+0x24*x)

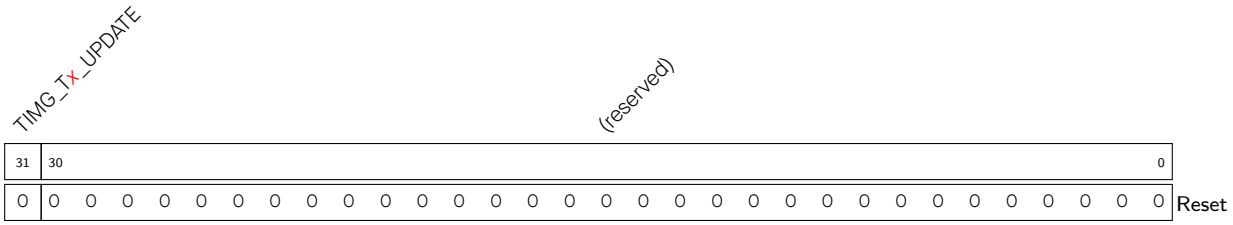


TIMG_Tx_HI 表示定时器 Tx 时基计数器的高 22 位的值。仅在 TIMG_TxUPDATE_REG 写值后有效。

单位：Tx_clk

(RO)

Register 17.4. TIMG_TxUPDATE_REG (x: 0-1) (0x000C+0x24*x)



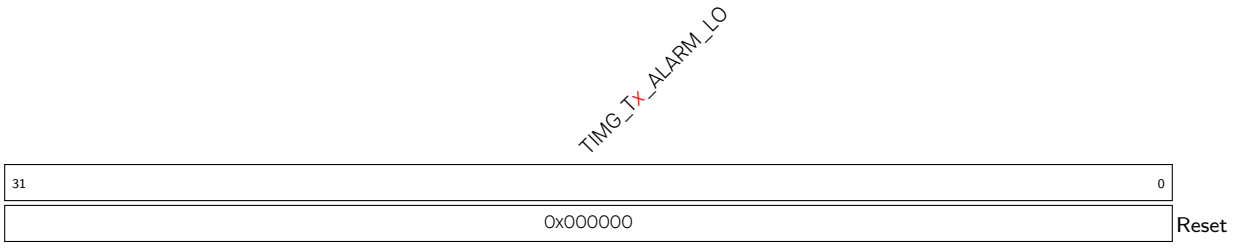
TIMG_Tx_UPDATE 配置锁存计数器的值。

0：锁存

1：锁存

(R/W/SC)

Register 17.5. TIMG_TxALARMLO_REG (x: 0-1) (0x0010+0x24*x)

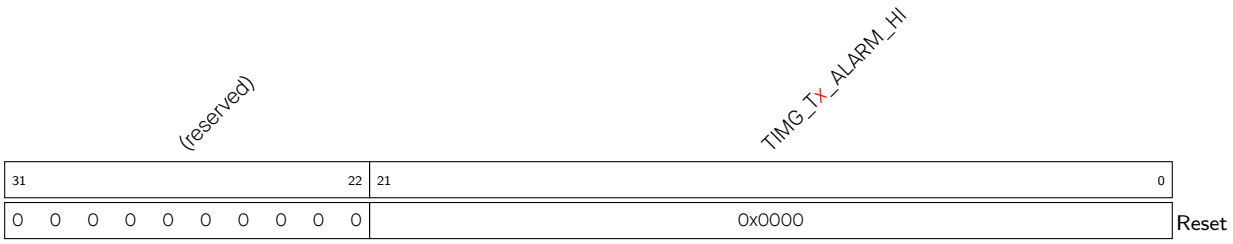


TIMG_Tx_ALARM_LO 配置定时器 Tx 时基计数器触发报警值的低 32 位。仅在 TIMG_Tx_ALARM_EN 为 1 时有效。

单位：Tx_clk

(R/W)

Register 17.6. TIMG_TxALARMHI_REG (x: 0-1) (0x0014+0x24*x)

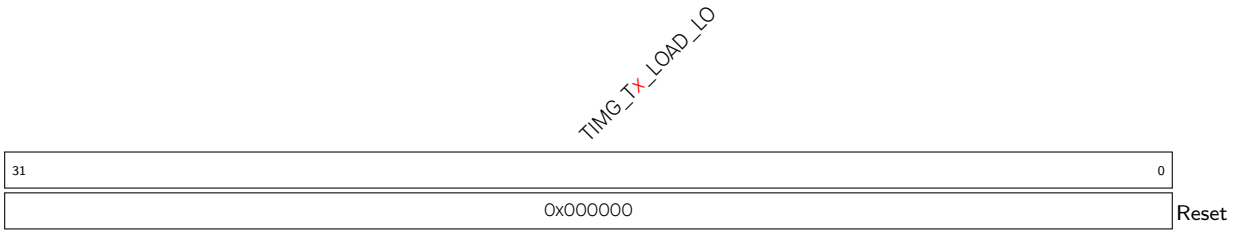


TIMG_Tx_ALARM_HI 配置定时器 Tx 时基计数器触发报警值的高 22 位。仅在 TIMG_Tx_ALARM_EN 为 1 时有效。

单位：Tx_clk

(R/W)

Register 17.7. TIMG_TxLOADLO_REG (x: 0-1) (0x0018+0x24*x)

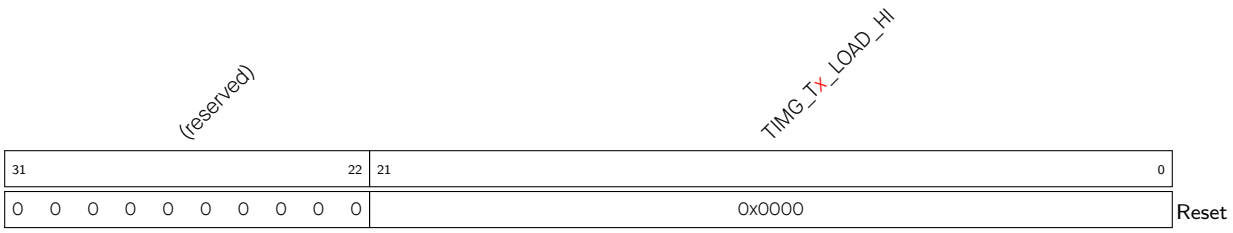


TIMG_Tx_LOAD_LO 配置定时器 Tx 时基计数器重新加载的低 32 位值。

单位：Tx_clk

(R/W)

Register 17.8. TIMG_TxLOADHI_REG (x: 0-1) (0x001C+0x24*x)

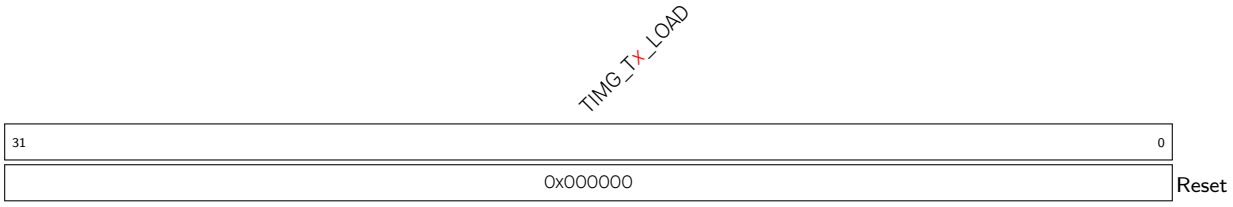


TIMG_Tx_LOAD_HI 配置定时器 Tx 时基计数器重新加载的高 22 位值。

单位：Tx_clk

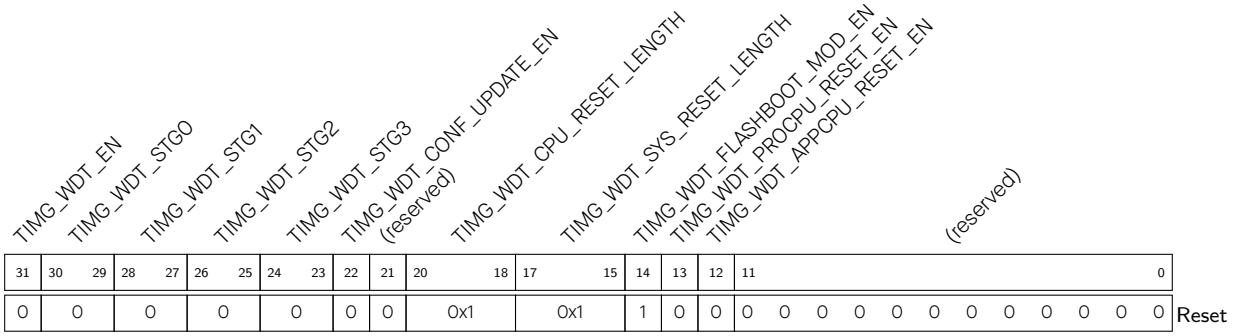
(R/W)

Register 17.9. TIMG_TxLOAD_REG (x: 0-1) (0x0020+0x24*x)



TIMG_Tx_LOAD 写任意值触发定时器 Tx 时基计数器重新加载。(WT)

Register 17.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_APPCPU_RESET_EN 配置是否屏蔽 MWDT 产生的 APP CPU 复位。仅在禁用写保护时有效。

0：屏蔽

1：不屏蔽

(R/W)

TIMG_WDT_PROCPU_RESET_EN 配置是否屏蔽 MWDT 产生的 Pro CPU 复位。仅在禁用写保护时有效。

0：屏蔽

1：不屏蔽

(R/W)

TIMG_WDT_FLASHBOOT_MOD_EN 配置是否使能 flash 启动保护。

0：禁用

1：使能

(R/W)

TIMG_WDT_SYS_RESET_LENGTH 配置系统复位信号长度。仅在禁用写保护时有效。

单位：mwdt_clk

0: 100 ns
1: 200 ns
2: 300 ns
3: 400 ns
4: 500 ns
5: 800 ns
6: 1.6 μs
7: 3.2 μs

(R/W)

TIMG_WDT_CPU_RESET_LENGTH 配置 CPU 复位信号长度。仅在禁用写保护时有效。

单位：mwdt_clk

0: 100 ns
1: 200 ns
2: 300 ns
3: 400 ns
4: 500 ns
5: 800 ns
6: 1.6 μs
7: 3.2 μs

(R/W)

见下页...

Register 17.10. TIMG_WDTCONFIG0_REG (0x0048)

接上页...

TIMG_WDT_CONF_UPDATE_EN 配置以更新看门狗定时器的配置寄存器。

0：无效
1：更新
(WT)

TIMG_WDT_STG3 配置阶段 3 的超时动作。仅在禁用写保护时有效。

0：无效
1：中断
2：复位 CPU
3：复位系统
(R/W)

TIMG_WDT_STG2 配置阶段 2 的超时动作。仅在禁用写保护时有效。

0：无效
1：中断
2：复位 CPU
3：复位系统
(R/W)

TIMG_WDT_STG1 配置阶段 1 的超时动作。仅在禁用写保护时有效。

0：无效
1：中断
2：复位 CPU
3：复位系统
(R/W)

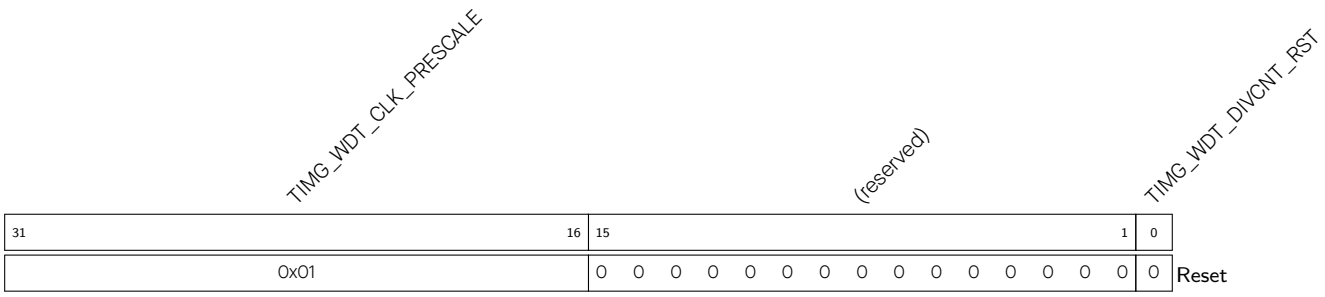
TIMG_WDT_STG0 配置阶段 0 的超时动作。仅在禁用写保护时有效。

0：无效
1：中断
2：复位 CPU
3：复位系统
(R/W)

TIMG_WDT_EN 配置是否使能 MWDT。仅在禁用写保护时有效。

0：禁用
1：使能
(R/W)

Register 17.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST 配置是否复位看门狗定时器时钟分频器的计数器。

0：无效

1：复位

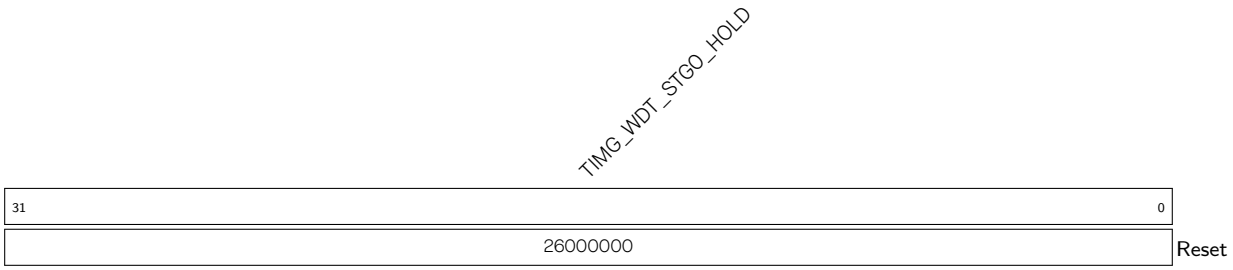
(WT)

TIMG_WDT_CLK_PRESCALE 配置 MWDT 时钟预分频器值。仅在禁用写保护时有效。

MWDT 时钟周期 = 12.5 ns * TIMG_WDT_CLK_PRESCALE

(R/W)

Register 17.12. TIMG_WDTCONFIG2_REG (0x0050)

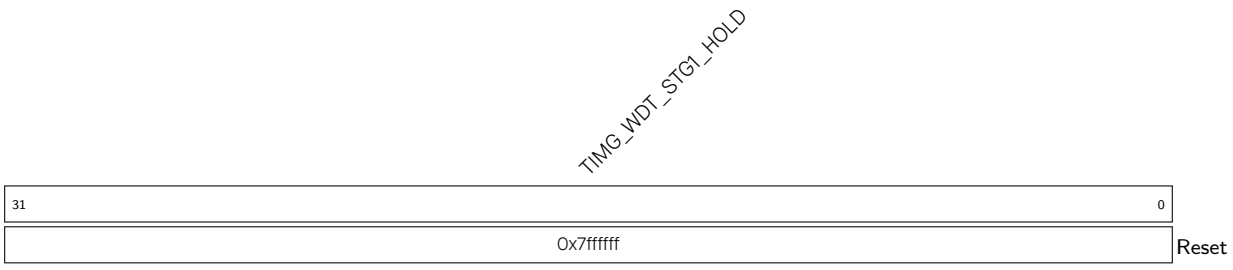


TIMG_WDT_STG0_HOLD 配置阶段 0 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk

(R/W)

Register 17.13. TIMG_WDTCONFIG3_REG (0x0054)

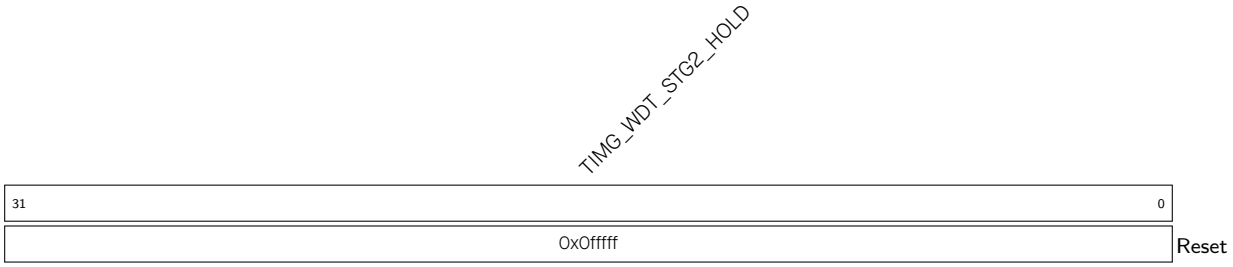


TIMG_WDT_STG1_HOLD 配置阶段 1 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk

(R/W)

Register 17.14. TIMG_WDTCONFIG4_REG (0x0058)

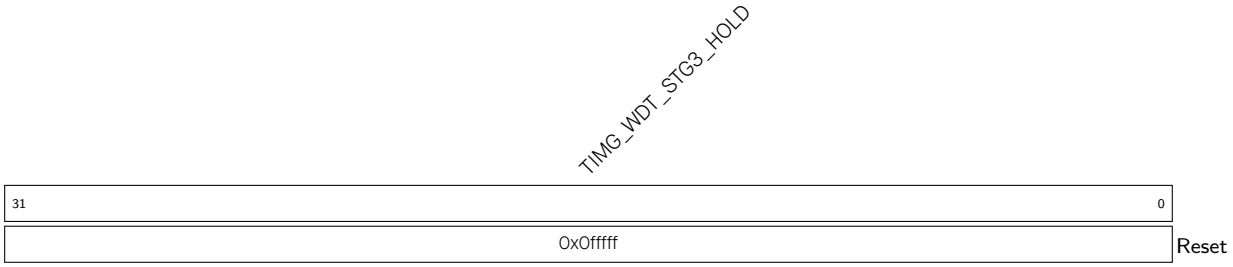


TIMG_WDT_STG2_HOLD 配置阶段 2 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk

(R/W)

Register 17.15. TIMG_WDTCONFIG5_REG (0x005C)

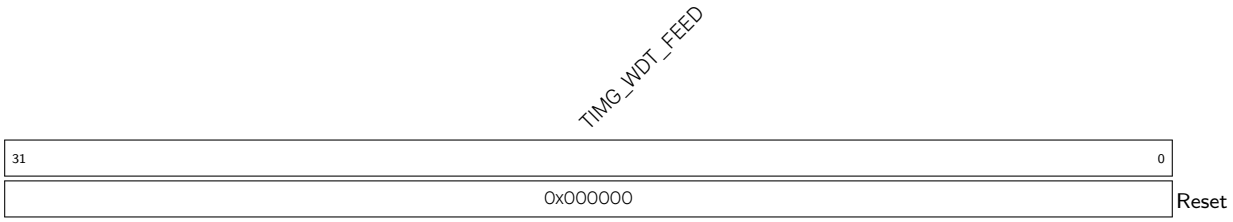


TIMG_WDT_STG3_HOLD 配置阶段 3 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk

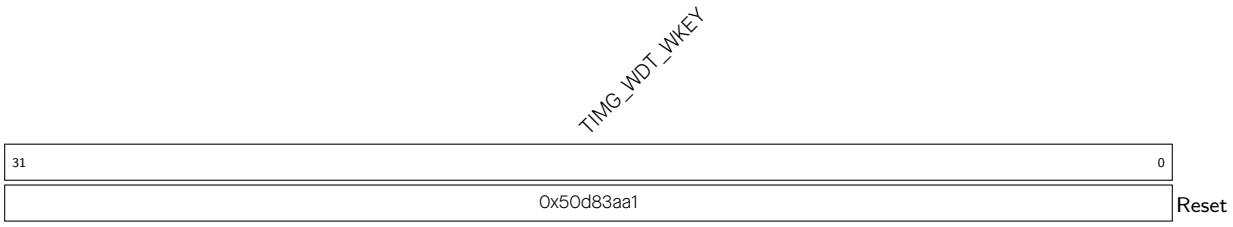
(R/W)

Register 17.16. TIMG_WDTFEED_REG (0x0060)



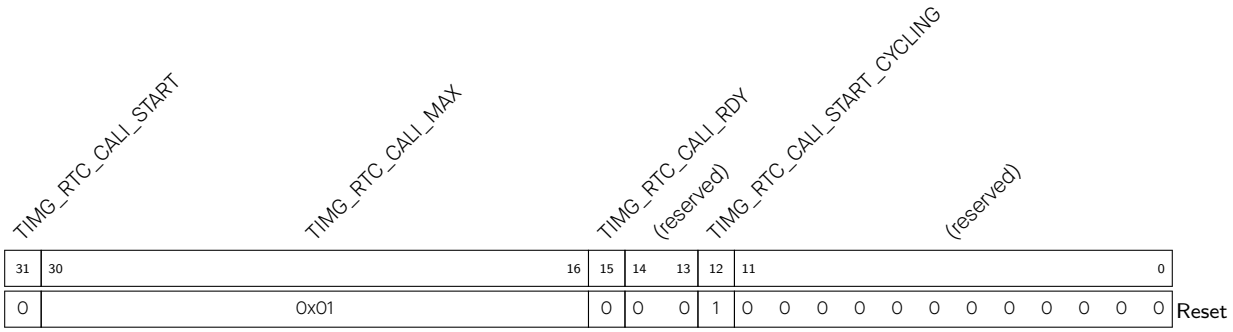
TIMG_WDT_FEED 写任意值喂 MWDT。仅在禁用写保护时有效。(WT)

Register 17.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY 配置一个与复位值不同的值以使能写保护。(R/W)

Register 17.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING 配置频率计算模式。

0：单次频率计算模式

1：周期性频率计算模式

(R/W)

TIMG_RTC_CALI_RDY 表示单次频率计算是否完成。

0：未完成

1：完成

(RO)

TIMG_RTC_CALI_MAX 配置计算 RTC 慢速时钟 SLOW_CLK 频率的时间。

单位：XTAL_CLK

(R/W)

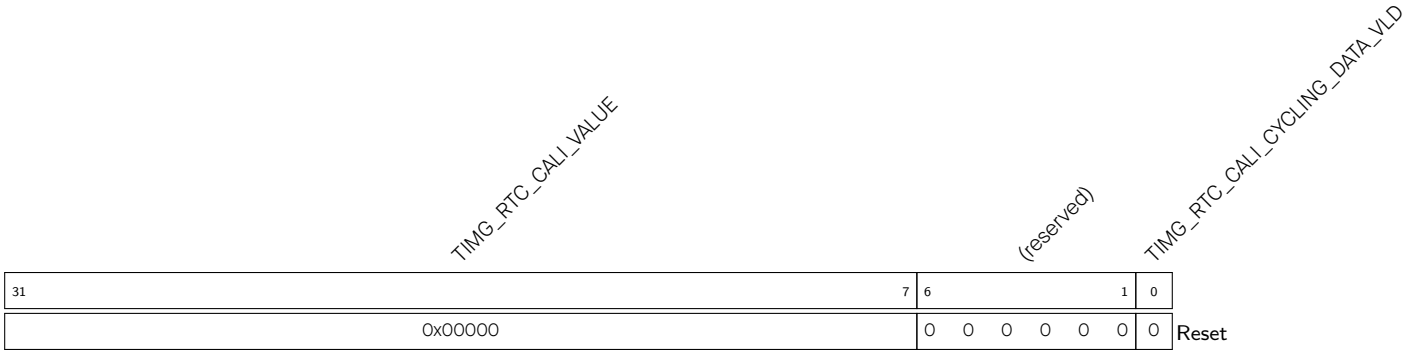
TIMG_RTC_CALI_START 配置是否使能单次频率计算。

0：禁用

1：使能

(R/W)

Register 17.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD 表示周期性频率计算是否完成。

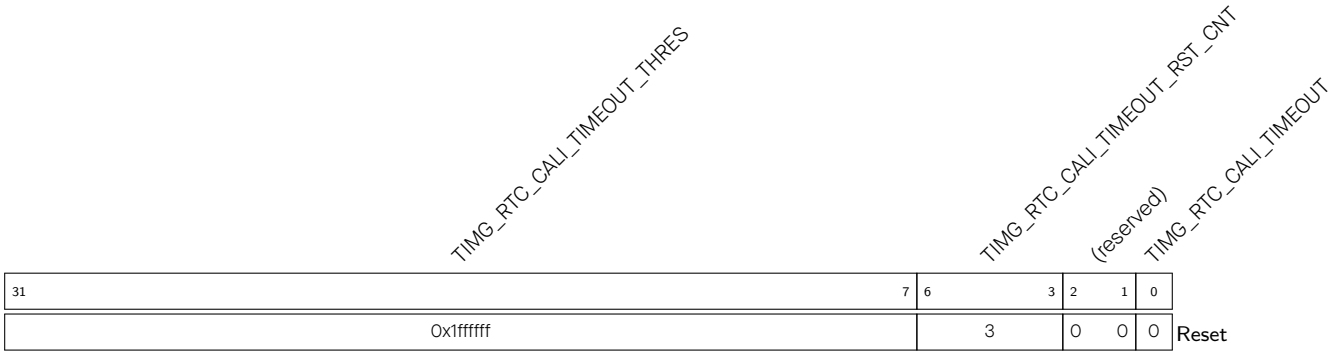
0：未完成

1：完成

(RO)

TIMG_RTC_CALI_VALUE 单次或周期性频率计算完成时，读取此位计算 RTC 慢速时钟 RTC_SLOW_CLK 的频率。(RO)

Register 17.20. TIMG_RTCCALICFG2_REG (0x0080)



TIMG_RTC_CALI_TIMEOUT 表示频率计算是否超时。

0：未超时

1：超时

(RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT 配置频率计算超时复位的周期。

单位：XTAL_CLK

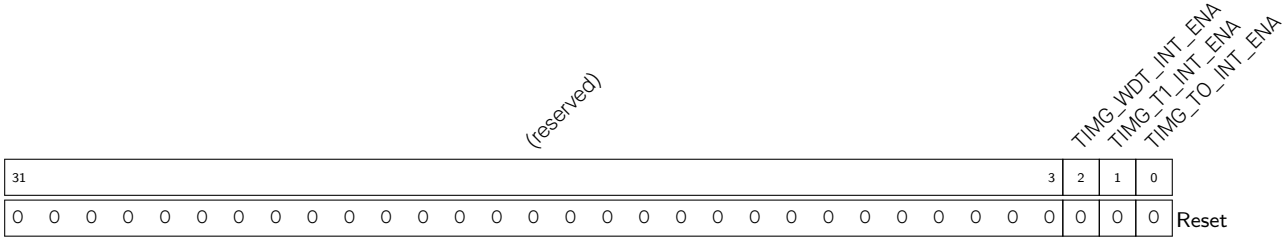
(R/W)

TIMG_RTC_CALI_TIMEOUT_THRES 配置 RTC 频率计算定时器的阈值。频率计算定时器的值超过此值时触发超时。

单位：XTAL_CLK

(R/W)

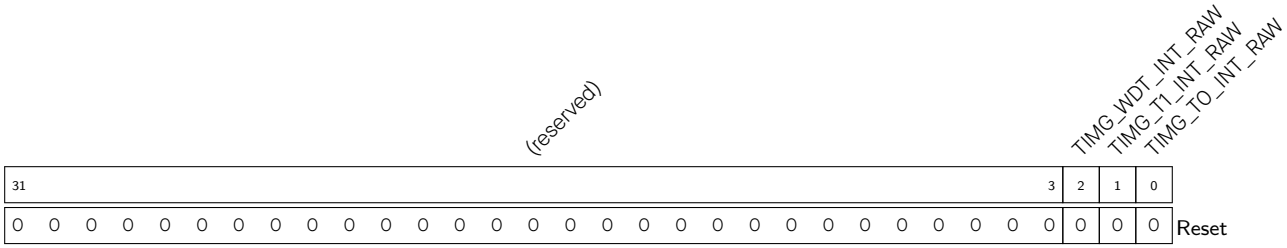
Register 17.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



TIMG_T*x*_INT_ENA (*x*: 0-1) 写 1 使能 TIMG_T*x*_INT 中断。(R/W)

TIMG_WDT_INT_ENA 写 1 使能 TIMG_WDT_INT 中断。(R/W)

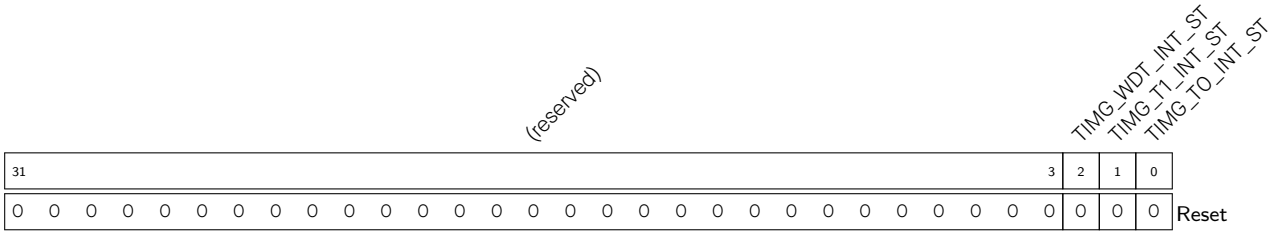
Register 17.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_T*x*_INT_RAW (*x*: 0-1) TIMG_T*x*_INT 中断的原始中断状态位。(R/SS/WTC)

TIMG_WDT_INT_RAW TIMG_WDT_INT 中断的原始中断状态位。(R/SS/WTC)

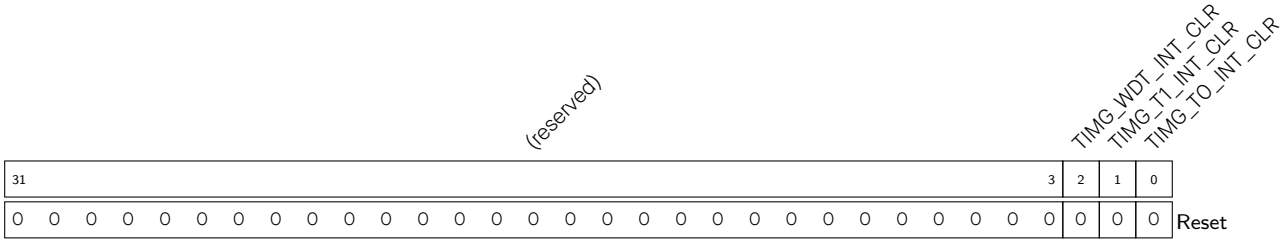
Register 17.23. TIMG_INT_ST_TIMERS_REG (0x0078)



TIMG_T*x*_INT_ST (*x*: 0-1) TIMG_T*x*_INT 中断的屏蔽中断状态位。(RO)

TIMG_WDT_INT_ST TIMG_WDT_INT 中断的屏蔽中断状态位。(RO)

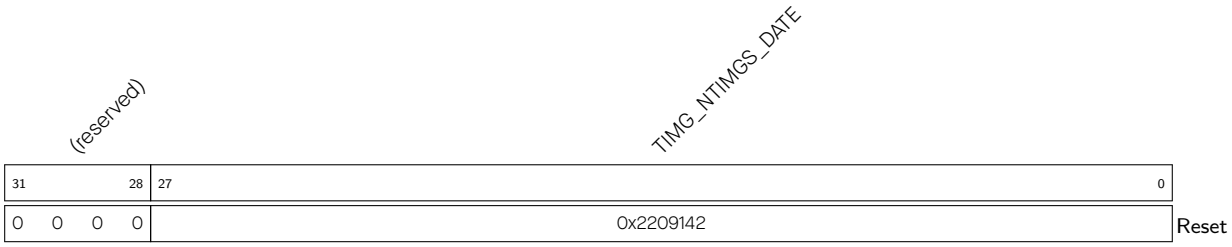
Register 17.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T*x*_INT_CLR (*x*: 0-1) 置位此位，清除 TIMG_T*x*_INT 中断。(WT)

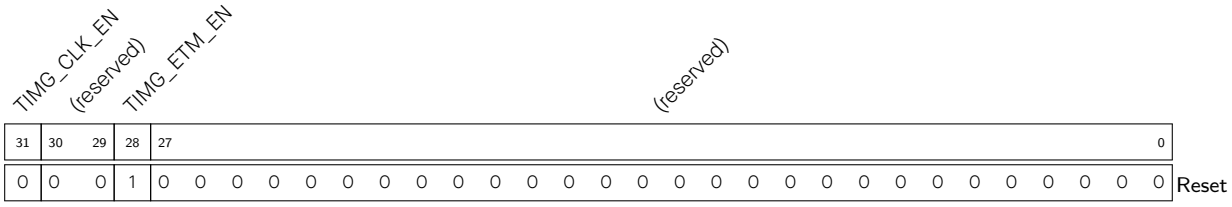
TIMG_WDT_INT_CLR 置位此位，清除 TIMG_WDT_INT 中断。(WT)

Register 17.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE 版本控制寄存器。(R/W)

Register 17.26. TIMG_REGCLK_REG (0x00FC)



TIMG_ETM_EN 配置是否使能定时器的 ETM 任务和事件。

0：禁用

1：使能

(R/W)

TIMG_CLK_EN 配置是否使能寄存器的门时钟信号。

0：寄存器无法通过软件读取或写入

1：寄存器可以通过软件读取或写入

(R/W)

第 18 章

看门狗定时器 (WDT)

18.1 概述

看门狗定时器是一种硬件定时器，用于监视系统运行状态。软件必须定期喂狗（复位看门狗定时器）。若系统出现故障、死锁、程序卡住或其他异常情况导致软件未及时喂狗，看门狗定时器将超时并触发中断或复位，将系统导向至已知状态进行异常处理或重新启动。因此，看门狗定时器有助于检测、处理系统或软件的错误行为。

如图 18.1-1 所示，ESP32-P4 中有三个数字看门狗定时器：章节 17 定时器组 (TIMG) 描述的两个定时器组中各有一个主系统看门狗定时器，缩写为 MWDT，LP 系统中有一个 RTC 看门狗定时器，缩写为 RWDT。数字看门狗在运行期间会经历四个阶段（除非看门狗按时喂狗或者处于关闭状态），每个阶段均可配置单独的超时时间和超时动作，其中 MWDT 支持中断、HP CPU 复位和 HP 内核复位三种超时动作，RWDT 支持中断、HP CPU 复位、HP 内核复位和系统复位四种超时动作（详见章节 18.2.2.2 阶段与超时动作）。每个阶段的超时时间都可单独设置。

在 SPI Boot 模式下，RWDT 和定时器组 0 的 MWDT 会默认使能，以检测引导过程中发生的错误，并恢复运行。

ESP32-P4 中还有一个模拟看门狗定时器——超级看门狗 (SWD)。超级看门狗是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。

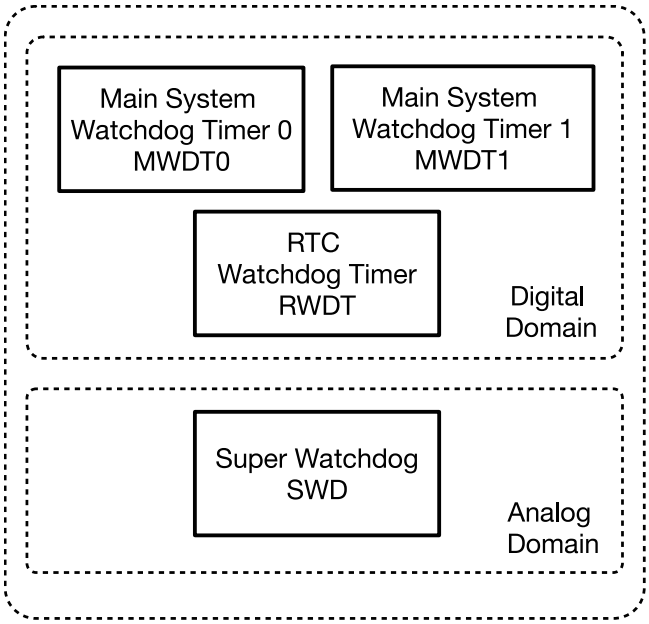


图 18.1-1. 看门狗定时器概览

请注意，MWDT 寄存器部分详见章节 17 定时器组 (TIMG)，RWDT 和 SWD 寄存器部分详见 18.5 寄存器列表。

18.2 数字看门狗定时器

18.2.1 主要特性

看门狗定时器具有如下特性：

- 四个阶段，每个阶段都可配置超时时间和超时动作
- 超时动作
  - MWDT：中断、HP CPU 复位、HP 内核复位
  - RWDT：中断、HP CPU 复位、HP 内核复位、系统复位
- 阶段 0 flash 启动保护（SPI Boot 模式）：
  - MWDT0：超时触发 HP 内核复位
  - RWDT：超时触发系统复位
- 写保护，使能时寄存器仅可读取
- 32 位超时计数器
- 时钟源：
  - MWDT：PLL_F80M_CLK、RC_FAST_CLK 或 XTAL_CLK
  - RWDT：LP_DYN_SLOW_CLK

18.2.2 功能描述

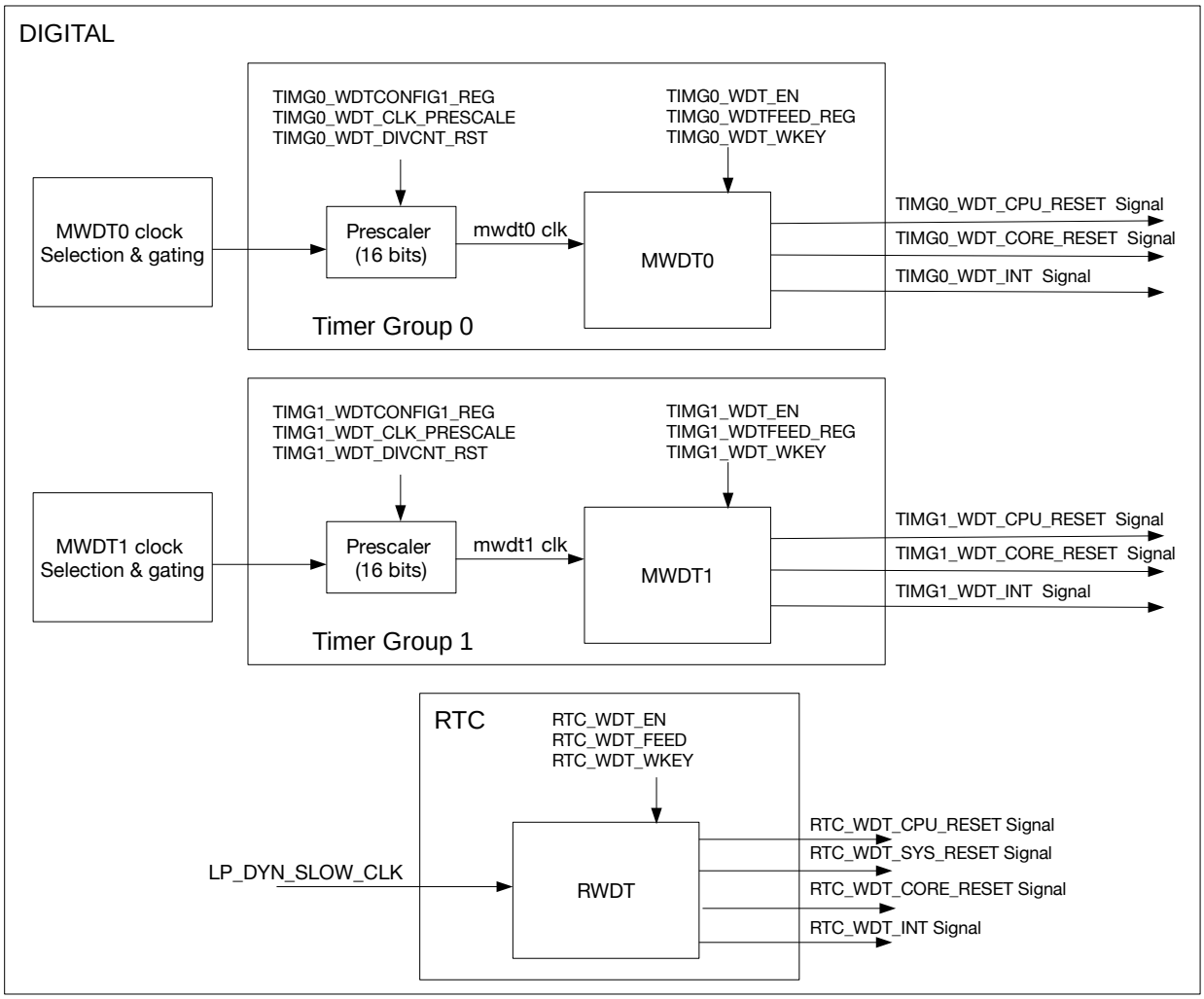


图 18.2-1. ESP32-P4 的数字看门狗定时器

图 18.2-1 为 ESP32-P4 数字系统中的三个看门狗定时器。

18.2.2.1 时钟源与 32 位计数器

看门狗定时器的核心是一个 32 位计数器。

以 MWDT0 为例：

- MWDT0 可通过设置 HP_SYS_CLKRST_PERI_CLK_CTRL20_REG 寄存器的 HP_SYS_TIMERGRP0_WDT_SRC_SEL 字段选择 PLL_F80M_CLK 时钟、RC_FAST_CLK 或 XTAL_CLK（外部时钟）作为时钟源。
- 将 HP_SYS_CLKRST_PERI_CLK_CTRL20_REG 寄存器的 HP_SYS_TIMERGRP0_WDT_CLK_EN 字段置 1 开启时钟，清零关闭时钟。时钟经由可配置的 16 位预分频器分频。详见章节 11 复位和时钟 的表 11.2-1。
- MWDT0 的 16 位预分频器可通过 TIMG_WDTCONFIG1_REG 寄存器的 TIMG_WDT_CLK_PRESCALE 字段配置。TIMG_WDT_DIVCNT_RST 字段置位时，预分频器复位，可立即重新配置。

RWDT 直接将 RTC 慢速时钟 LP_DYN_SLOW_CLK（详见章节 11 复位和时钟）用作时钟源。

MWDT 和 RWDT 看门狗可分别通过设置 TIMG_WDT_EN 和 RTC_WDT_EN 字段使能。看门狗使能后，其内部 32 位计数器的值会在每个时钟源周期内累加 1，直到达到该阶段的超时时间（即在该阶段发生超时）。如发生超时，计数器的值会重置为 0，同时看门狗进入下一阶段。如果软件成功喂狗，看门狗定时器会回到阶段 0，并将计数器的值重置为 0。软件向 TIMG_WDTFEED_REG 寄存器内写入任意值，便可为 MDWT 喂狗；向 RTC_WDT_FEED 写 1，便可为 RWDT 喂狗。

18.2.2.2 阶段与超时动作

定时器在各阶段可以配置不同的超时时间和对应的超时动作。某一阶段超时会触发对应的超时动作，同时计数器的值被重置为 0，看门狗进入下一阶段。

MWDT 和 RWDT 有四个阶段（称为阶段 0 至阶段 3）。看门狗定时器会循环工作（即从阶段 0 至阶段 3，再回到阶段 0）。

MWDT 每个阶段的超时时间可用 TIMG_WDTCONFIG*i*_REG（i 的范围是 2 到 5）寄存器配置，RWDT 的超时时间可用 RTC_WDT_STG*j*_HOLD（j 的范围是 0 到 3）字段配置。

值得注意的是，RWDT 在阶段 0 的超时时间 ($Thold_0$) 受 eFuse 寄存器 EFUSE_RD_REPEAT_DATA0_REG 的 EFUSE_WDT_DELAY_SEL 字段和 RTC_WDT_STG0_HOLD 字段共同影响，关系如下：

$$T_{hold0} = RTC_WDT_STG0_HOLD << (EFUSE_WDT_DELAY_SEL + 1)$$

其中，<< 为左移运算符。例如，RTC_WDT_STG0_HOLD 配置为 100，EFUSE_WDT_DELAY_SEL 为 1，则 $Thold_0$ 为 400 个时钟周期。

如某个阶段超时，下列超时动作之一将会执行：

表 18.2-1. 超时动作

超时动作	描述
中断	触发中断
HP CPU 复位	复位 HP CPU0 和 HP CPU1
HP 内核复位	HP 内核复位包括 HP CPU0、HP CPU1、HP 外设、HP GPIO 等
系统复位	复位包括低功耗系统在内的整个数字系统
关闭	对系统不产生影响

MWDT 所有阶段的超时动作均在 TIMG_WDTCONFIG0_REG 寄存器中配置。RWDT 的超时动作可在 RTC_WDT_CONFIG0_REG 寄存器配置。

18.2.2.3 写保护

看门狗定时器对于检测和处理系统或软件错误而言至关重要，不应轻易关闭（例如，因写寄存器位置错误而误将看门狗关闭）。因此，MWDT 和 RWDT 引入写保护机制，防止看门狗因无意的写操作而被关闭或篡改。

写保护机制通过每个看门狗定时器的写密钥字段运行（MWDT 看门狗使用 TIMG_WDT_WKEY，RWDT 看门狗使用 RTC_WDT_WKEY）。必须向看门狗定时器的写密钥字段写入 0x50D83AA1，才能修改其它看门狗寄存器。如果写密钥字段的值不是 0x50D83AA1，任何试图向看门狗定时器寄存器（除了向写密钥字段本身）写值的操作都会被忽略。推荐按以下步骤访问看门狗定时器：

1. 将 0x50D83AA1 写入看门狗定时器的写密钥字段，关闭写保护。
2. 根据需要修改看门狗，如喂狗或改变配置。
3. 向看门狗定时器的写密钥字段上写入除 0x50D83AA1 以外的任意值，重新使能写保护。

18.2.2.4 flash 引导保护

在 SPI boot 模式下，MWDT0 和 RWDT 会默认使能。MWDT0 的阶段 0 的默认超时动作为 HP 内核复位（复位主系统）。RWDT 的阶段 0 超时动作为系统复位（复位主系统和 RTC）。完成 ROM 阶段的 SPI boot 引导后，应将 TIMG_WDT_FLASHBOOT_MOD_EN 和 RTC_WDT_FLASHBOOT_MOD_EN 位清零，分别关闭 MWDT0 和 RWDT 的 flash 引导保护。然后，软件可以配置 MWDT0 和 RWDT。

18.3 超级看门狗定时器

超级看门狗 (SWD) 是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。SWD 包含一个看门狗电路，需在每个超时阶段（约不足一秒）至少喂狗一次。该电路会在看门狗超时时间约 100 ms 之前发送 WD_INTR 信号提醒系统喂狗。

如果系统不回应 SWD 的喂狗请求，看门狗超时，SWD 会产生系统电平信号 SWD_RSTB，复位芯片上的整个数字电路（即超时动作中的系统复位）。

SWD 的时钟源固定，不可选择。

18.3.1 主要特性

SWD 具有如下特性：

- 超低功耗
- 用中断提醒 SWD 即将超时
- 软件有多种专用的方法喂 SWD，让 SWD 监控整个操作系统的工作状态

18.3.2 SWD 控制器

18.3.2.1 结构

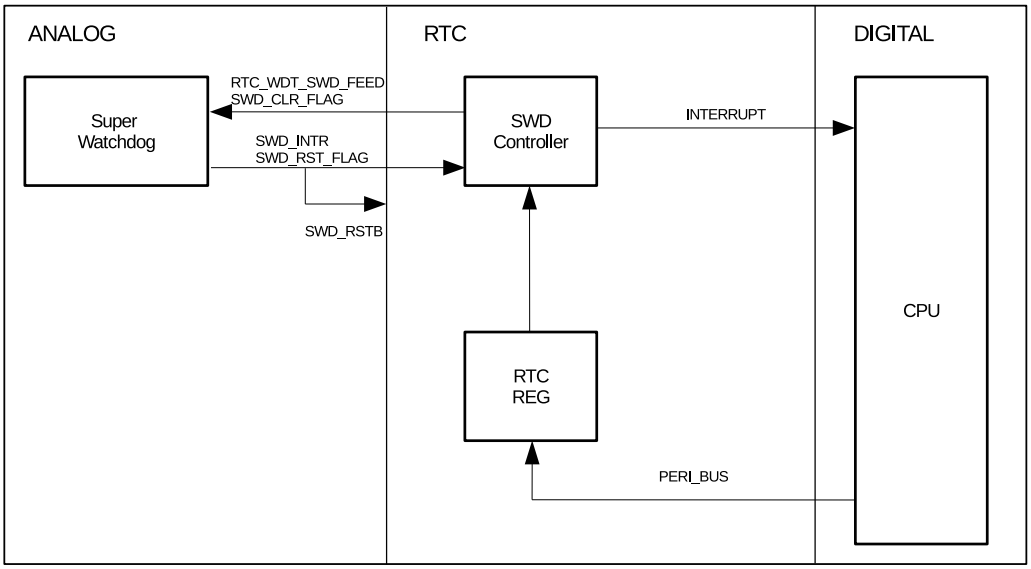


图 18.3-1. SWD 控制器结构

18.3.2.2 工作流程

正常状态下：

- SWD 控制器收到 SWD 的喂狗请求。
- SWD 控制器向主 CPU 发送中断。
- 主 CPU 通过置位 RTC_WDT_SWD_FEED 直接喂狗。
- CPU 喂狗时，需要先向 RTC_WDT_SWD_WKEY 写 0x50D83AA1 关闭 SWD 控制器的写保护。这样做可以防止系统在数字电路异常状态下运行时误喂 SWD。
- 如将 RTC_WDT_SWD_AUTO_FEED_EN 置 1，SWD 控制器也可配置为在不需要 CPU 干预的情况下喂 SWD。

复位后：

- 可查看 LP_AONCLKRST_LPCORE_RESET_CAUSE[4:0] 获知 HP CPU 复位原因。
  如 LP_AONCLKRST_LPCORE_RESET_CAUSE[4:0] == 0x12，则表示上一次复位的原因是 SWD 复位。
- 置位 RTC_WDT_SWD_RST_FLAG_CLR 清除 SWD 复位标志。

18.4 中断

看门狗定时器中断，请前往章节 17 定时器组 *(TIMG)* 的第 17.3.7 节 中断 查看。

18.5 寄存器列表

MWDT 寄存器是定时器组模块的一部分，在章节 17 定时器组 (TIMG) 的第 17.5 节 寄存器列表 中有详细描述。

本小节包含 RWDT 和 SWD 相关寄存器介绍，其地址均为相对于 RTC_WDT 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

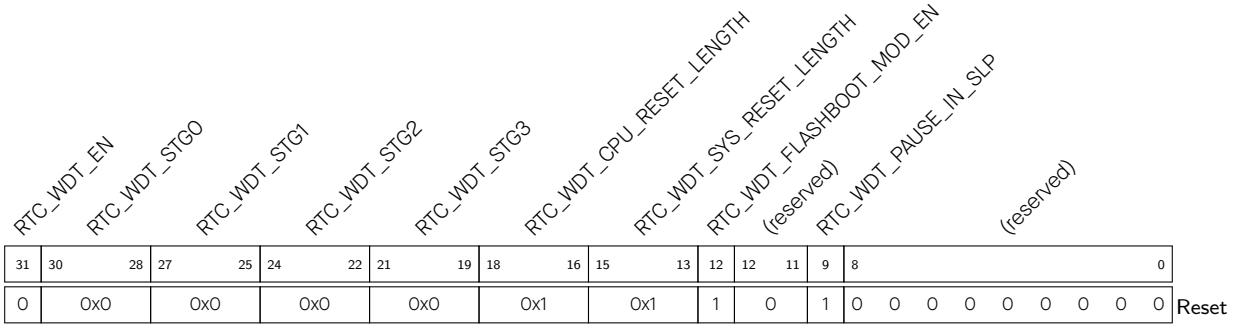
名称	描述	地址	访问
配置寄存器			
RTC_WDT_CONFIG0_REG	RWDT 配置寄存器	0x0000	R/W
RTC_WDT_CONFIG1_REG	配置 RWDT 阶段 0 的超时时间	0x0004	R/W
RTC_WDT_CONFIG2_REG	配置 RWDT 阶段 1 的超时时间	0x0008	R/W
RTC_WDT_CONFIG3_REG	配置 RWDT 阶段 2 的超时时间	0x000C	R/W
RTC_WDT_CONFIG4_REG	配置 RWDT 阶段 2 的超时时间	0x0010	R/W
RTC_WDT_FEED_REG	配置喂狗 RWDT	0x0014	WT
RTC_WDT_WPROTECT_REG	配置锁住 RWDT 的配置寄存器	0x0018	R/W
RTC_WDT_SWD_CONFIG_REG	SWD 配置寄存器	0x001C	varies
RTC_WDT_SWD_WPROTECT_REG	配置锁住 SWD 的配置寄存器	0x0020	R/W
RTC_WDT_INT_RAW_REG	配置是否产生 WDT 超时中断	0x0024	R/WTC/SS
RTC_WDT_INT_ST_REG	WDT 中断状态寄存器	0x0028	RO
RTC_WDT_INT_ENA_REG	WDT 中断使能寄存器	0x002C	R/W
RTC_WDT_INT_CLR_REG	WDT 中断清除寄存器	0x0030	WT
RTC_WDT_DATE_REG	版本控制寄存器	0x03FC	R/W

18.6 寄存器

MWDT 寄存器是定时器组模块的一部分，在章节 17 定时器组 *(TIMG)* 的第 17.5 节 寄存器列表 中有详细描述。

本小节包含 RWDT 和 SWD 相关寄存器介绍，其地址均为相对于 RTC_WDT 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

Register 18.1. RTC_WDT_CONFIG0_REG (0x0000)



RTC_WDT_PAUSE_IN_SLP 配置芯片处于睡眠模式（CPU 不可工作）时是否禁用 RWDT。

0：使能

1：禁用

(R/W)

RTC_WDT_FLASHBOOT_MOD_EN 配置在 SPI boot 模式下是否自动使能 RWDT。

0：禁用

1：使能

(R/W)

RTC_WDT_SYS_RESET_LENGTH 配置 HP 内核复位时间。

单位：LP_DYN_FAST_CLK

(R/W)

RTC_WDT_CPU_RESET_LENGTH 配置 HP CPU 复位时间。

单位：LP_DYN_FAST_CLK

(R/W)

RTC_WDT_STG3 配置阶段 3 的超时复位动作。

0：无效

1：中断

2：HP CPU 复位

3：HP 内核复位

4：系统复位

(R/W)

见下页...

Register 18.1. RTC_WDT_CONFIG0_REG (0x0000)

接上页...

RTC_WDT_STG2 配置阶段 2 的超时复位动作。

0：无效
1：中断
2：HP CPU 复位
3：HP 内核复位
4：系统复位
(R/W)

RTC_WDT_STG1 配置阶段 1 的超时复位动作。

0：无效
1：中断
2：HP CPU 复位
3：HP 内核复位
4：系统复位
(R/W)

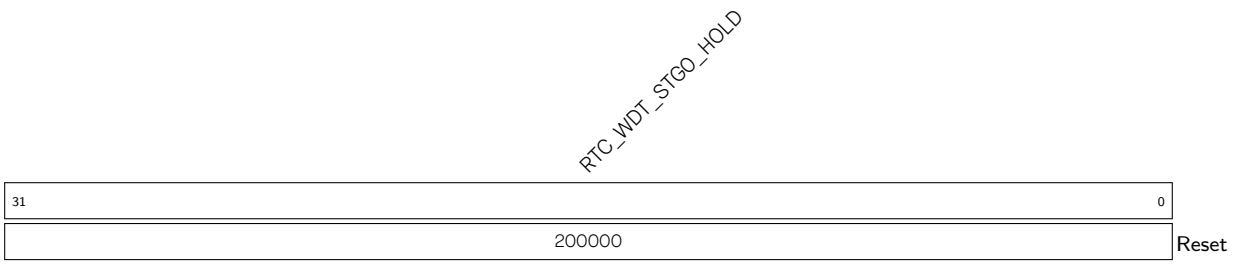
RTC_WDT_STG0 配置阶段 0 的超时复位动作。

0：无效
1：中断
2：HP CPU 复位
3：HP 内核复位
4：系统复位
(R/W)

RTC_WDT_EN 配置是否使能 RWDT。

0：禁用 RWDT
1：使能 RWDT
(R/W)

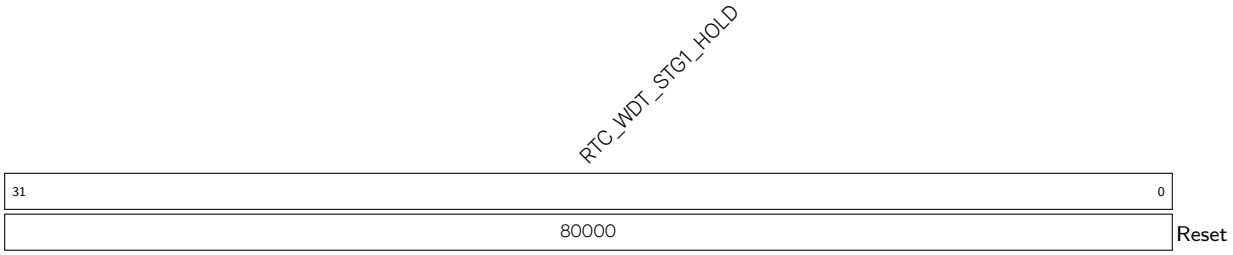
Register 18.2. RTC_WDT_CONFIG1_REG (0x0004)



RTC_WDT_STG0_HOLD 配置阶段 0 的超时时间。

单位：LP_DYN_SLOW_CLK
(R/W)

Register 18.3. RTC_WDT_CONFIG2_REG (0x0008)

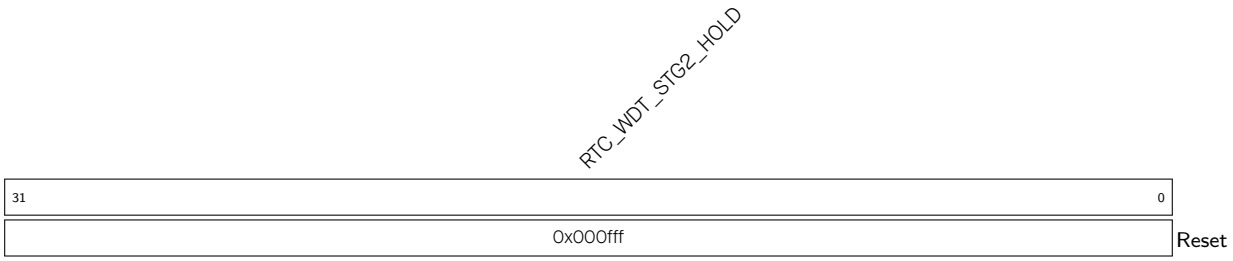


RTC_WDT_STG1_HOLD 配置阶段 1 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 18.4. RTC_WDT_CONFIG3_REG (0x000C)

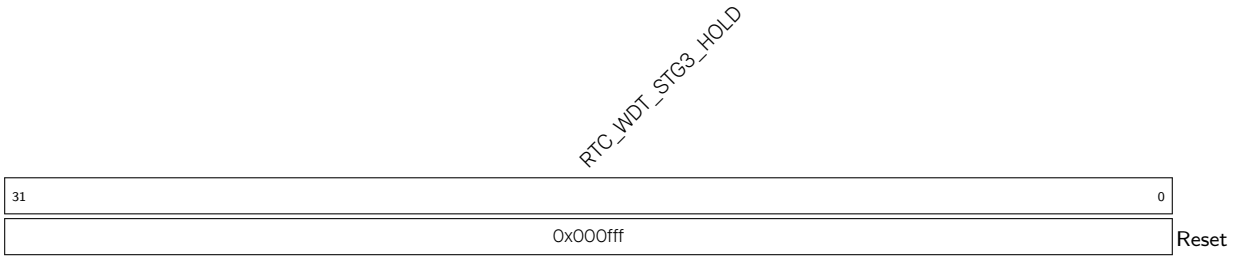


RTC_WDT_STG2_HOLD 配置阶段 2 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 18.5. RTC_WDT_CONFIG4_REG (0x0010)

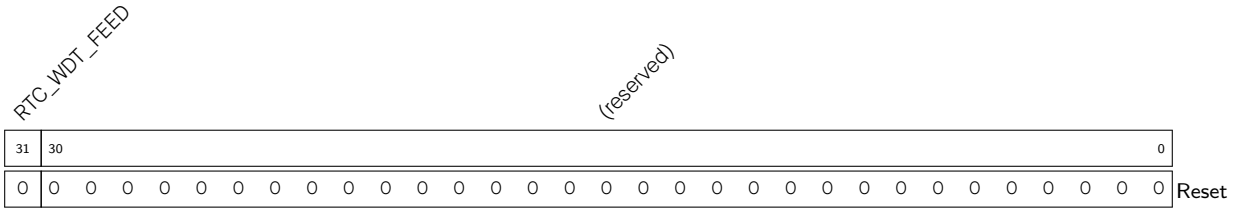


RTC_WDT_STG3_HOLD 配置阶段 3 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 18.6. RTC_WDT_FEED_REG (0x0014)



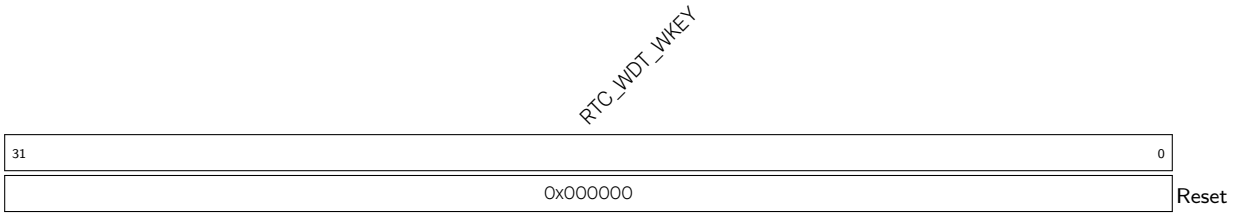
RTC_WDT_FEED 配置此位喂狗 RWDT。

0：无效

1：喂狗

(WT)

Register 18.7. RTC_WDT_WPROTECT_REG (0x0018)



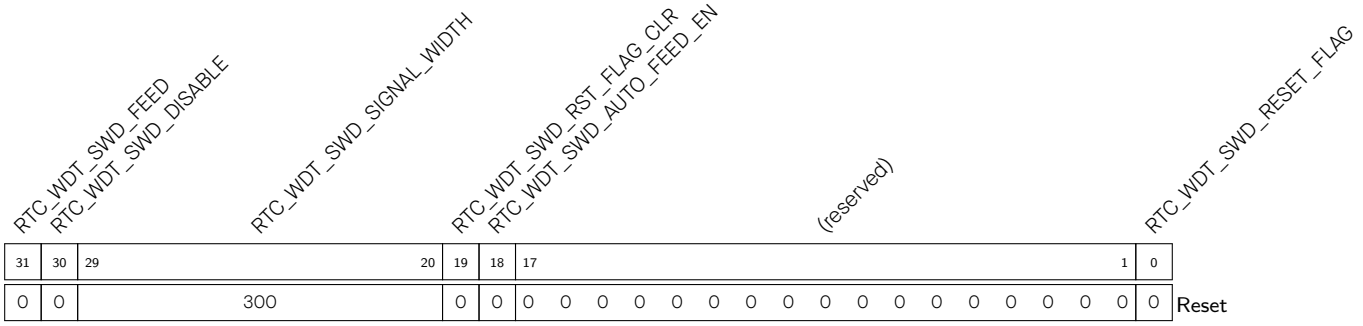
RTC_WDT_WKEY 配置该寄存器来锁住或解锁 RWDT 的配置寄存器。

0x50D83AA1: 解锁 RWDT 的配置寄存器

其他值: 锁住 RWDT 的配置寄存器，软件不能修改 RWDT 的配置

(R/W)

Register 18.8. RTC_WDT_SWD_CONFIG_REG (0x001C)



RTC_WDT_SWD_RESET_FLAG 表示 SWD 是否产生了复位信号。

0：否

1：是

(RO)

RTC_WDT_SWD_AUTO_FEED_EN 配置此位使能硬件自动喂狗。

0：禁止硬件自动喂狗

1：使能硬件自动喂狗

(R/W)

RTC_WDT_SWD_RST_FLAG_CLR 配置此位清除 SWD 复位标志。

0：无效

1：清除复位标志

(WT)

RTC_WDT_SWD_SIGNAL_WIDTH 配置输出到模拟电路的信号长度。

单位：LP_DYN_FAST_CLK

(R/W)

RTC_WDT_SWD_DISABLE 配置此位禁用 SWD。

0：使能

1：禁用

(R/W)

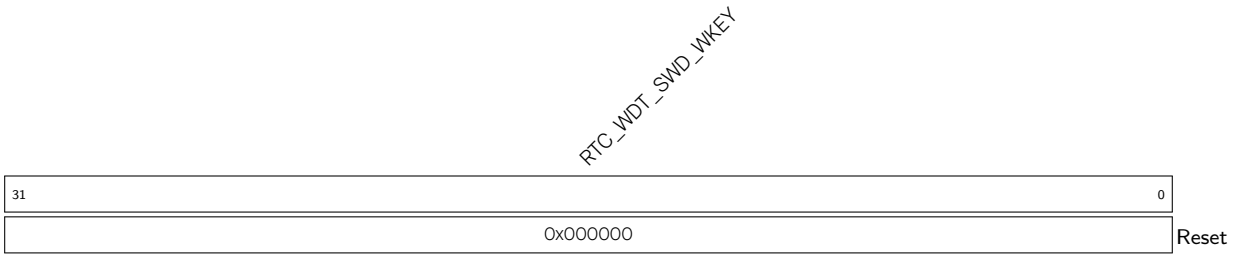
RTC_WDT_SWD_FEED 配置此位喂狗 SWD。

0：无效

1: 喂狗 SWD

(WT)

Register 18.9. RTC_WDT_SWD_WPROTECT_REG (0x0020)



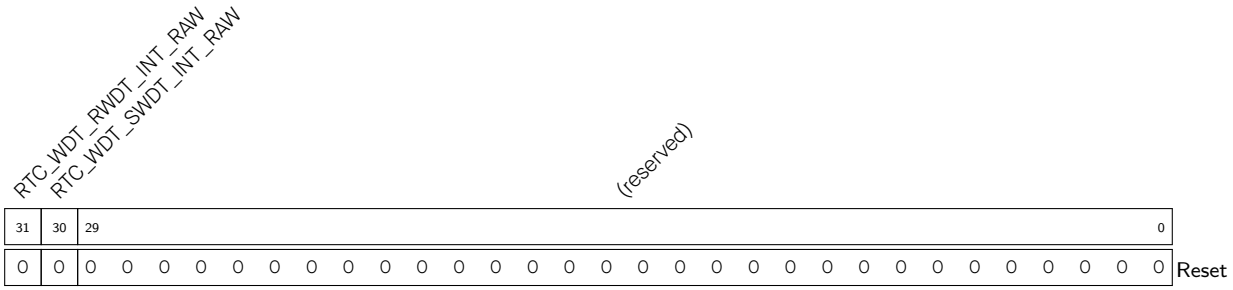
RTC_WDT_SWD_WKEY 配置该寄存器来锁住或解锁 SWDT 的配置寄存器。

0x50D83AA1: 解锁 SWD 的配置寄存器

其他值: 锁住 SWD 的配置寄存器，软件不能修改 SWD 的配置

(R/W)

Register 18.10. RTC_WDT_INT_RAW_REG (0x0024)



RTC_WDT_SWDT_INT_RAW 表示 SWD 是否产生超时中断。

0：否

1：是

(R/WTC/SS)

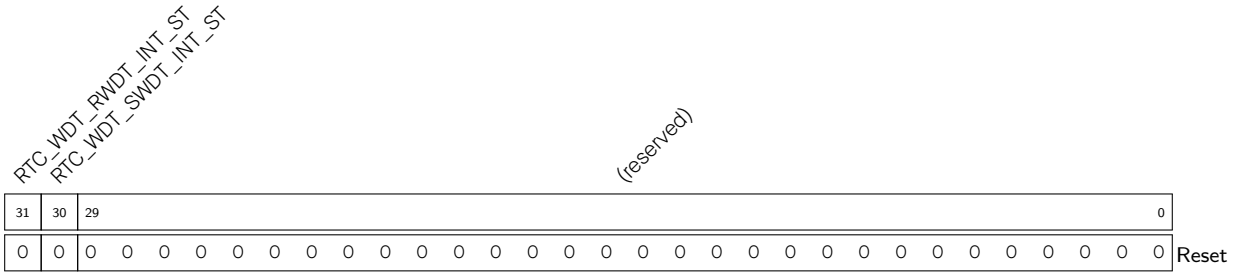
RTC_WDT_RWDT_INT_RAW 表示 RWDT 是否产生超时中断。

0：否

1：是

(R/WTC/SS)

Register 18.11. RTC_WDT_INT_ST_REG (0x0028)



RTC_WDT_SWDT_INT_ST 表示 SWD 是否产生并发送超时中断给 CPU。

0：否

1：是

(RO)

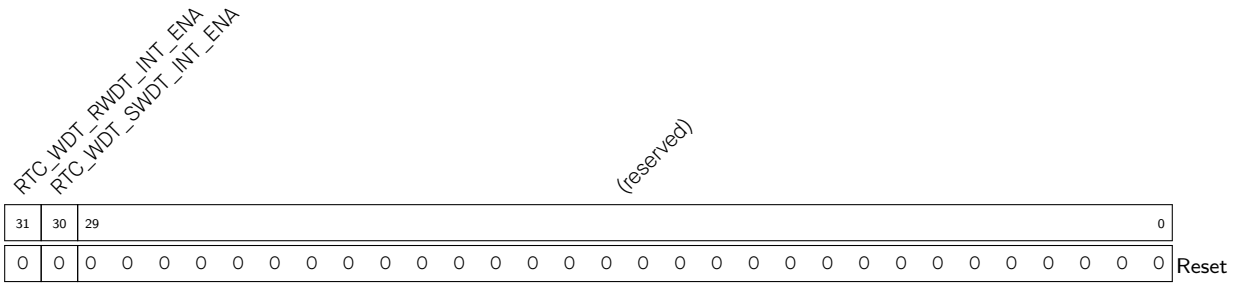
RTC_WDT_RWDT_INT_ST 表示 RWDT 是否产生并发送超时中断给 CPU。

0：否

1：是

(RO)

Register 18.12. RTC_WDT_INT_ENA_REG (0x002C)



RTC_WDT_SWDT_INT_ENA 配置是否使能 SWD 发送超时中断。

0：否

1：是

(R/W)

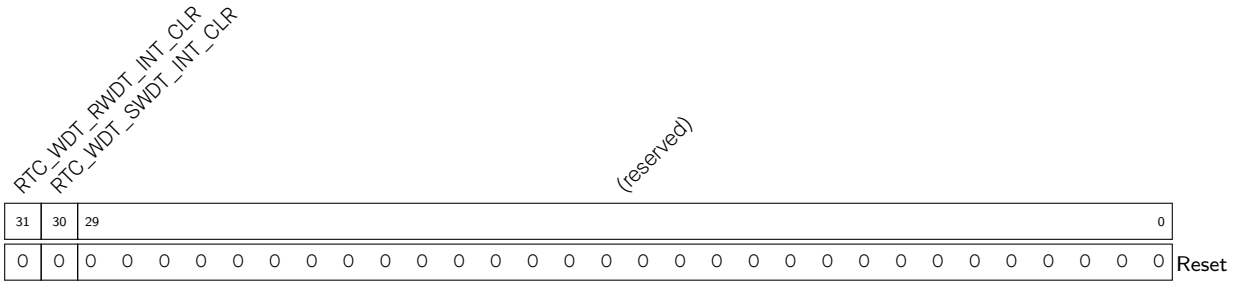
RTC_WDT_RWDT_INT_ENA 配置是否使能 RWDT 发送超时中断。

0：否

1：是

(R/W)

Register 18.13. RTC_WDT_INT_CLR_REG (0x0030)



RTC_WDT_SWDT_INT_CLR 配置是否清除 SWD 发送到 CPU 的超时中断信号。

0：否

1：是

(WT)

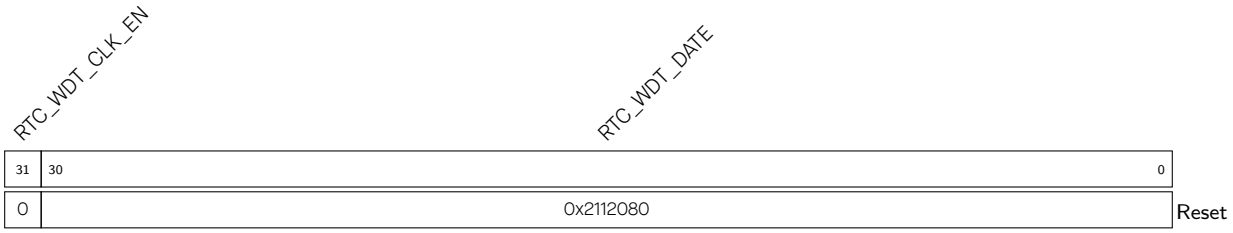
RTC_WDT_RWDT_INT_CLR 配置是否清除 RWDT 发送到 CPU 的超时中断信号。

0：否

1：是

(WT)

Register 18.14. RTC_WDT_DATE_REG (0x03FC)



RTC_WDT_DATE 版本控制寄存器。(R/W)

RTC_WDT_CLK_EN 保留。(R/W)

第 19 章

实时时钟定时器

19.1 概述

RTC Timer 是实现 ESP32-P4 低功耗管理的一个重要模块。RTC Timer 是一个 48 位的可读计时器，主要作用是在低功耗模式下，当 HP 系统中的定时器外设不可用时，继续为系统提供定时器服务。同时还支持配置定时器中断、记录系统中特定事件发生的时刻。

19.2 主要特性

- 48 位的计时器
- 触发特定事件时可记录事件发生的时刻，支持的特定事件有：
  - HP 系统复位
  - CPU 进入 stall 状态
  - CPU 退出 stall 状态
  - 晶振时钟开启
  - 晶振时钟关闭
- 通过配置寄存器触发 RTC Timer 记录当前时间
- 支持缓存最近两次特定事件的发生的时间
- 支持在目标时刻产生中断，目标时刻可配置，可同时配置两个目标时刻
- 除 LP 系统上电复位外的其余任何复位/睡眠均不会使 RTC Timer 停止或复位

19.3 功能描述

- 48 位计时器
  - RTC Timer 的实现基于一个 48 位的计时器，其驱动时钟是由位于 AlwaysOn 电源域的 RTC_SLOW_CLK 提供，采用不间断的循环计时（LP 系统复位除外），当 48 位计时器溢出时会产生溢出中断。
- 记录特定事件的发生时刻
  - RTC Timer 支持记录三类特定事件的发生时刻：
    * HP 系统复位
    * CPU 进入或退出 stall 状态

* 晶振时钟开启或关闭

- 三类事件都有各自独立的配置寄存器，当配置寄存器使能后，若发生对应的事件，产生的硬件脉冲就会触发 RTC Timer 记录当前时间。对应的配置寄存器见下表：

表 19.3-1. 配置 RTC Timer 记录特定事件的发生时刻

配置寄存器	描述
RTC_TIMER_MAIN_TIMER_SYS_RST	使能 RTC Timer 记录系统复位时刻。
RTC_TIMER_MAIN_TIMER_SYS_STALL	使能 RTC Timer 记录 CPU 进入或退出 stall 状态时刻。
RTC_TIMER_MAIN_TIMER_XTAL_OFF	使能 RTC Timer 记录晶振时钟开启或关闭时刻。

- 通过配置寄存器触发 RTC Timer 记录当前时间
  - 除了上述三类特定事件发生时可以触发 RTC Timer 记录当前时间，还可以通过配置寄存器 RTC_TIMER_MAIN_TIMER_UPDATE 触发 RTC Timer 记录当前时间。
  - 配置 RTC_TIMER_MAIN_TIMER_UPDATE 寄存器会产生一个脉冲信号，触发 RTC Timer 记录当前时间。
- 目标时刻产生计数中断
  - RTC Timer 支持同时配置两个目标时刻，分别为目标时刻 0 和目标时刻 1，当计时器到达目标时刻 0 时会触发 RTC_TIMER_CMP0_INT 中断，当计时器到达目标时刻 1 时会触发 RTC_TIMER_CMP1_INT 中断。
  - 目标时刻的配置寄存器为：

表 19.3-2. RTC Timer 目标时刻配置

配置寄存器	描述
RTC_TIMER_MAIN_TIMER_TAR_EN*n*	*n* = 0 or 1，配置目标时刻使能。
RTC_TIMER_MAIN_TIMER_TAR_LOW*n*	*n* = 0 or 1，配置目标时刻的低 32 位。
RTC_TIMER_MAIN_TIMER_TAR_HIGH*n*	*n* = 0 or 1，配置目标时刻的高 16 位。

- 读取缓存时刻
  - RTC Timer 有两组寄存器用于缓存特定事件的发生时刻，分别为寄存器组 0 和寄存器组 1；
  - 寄存器组 0 包括寄存器RTC_TIMER_MAIN_BUF0_LOW_REG 和 RTC_TIMER_MAIN_BUF0_HIGH_REG，用于记录 RTC Timer 定时器在当前触发下的计数值；
  - 寄存器组 1 包括寄存器 RTC_TIMER_MAIN_BUF1_LOW_REG 和 RTC_TIMER_MAIN_BUF1_HIGH_REG，用于记录 RTC Timer 定时器在上一次触发下的计数值；
  - 每次有新的触发，上一次触发时的记录将从寄存器组 0 移至寄存器组 1（寄存器组 1 中之前的记录将被覆盖），而本次触发的记录将存储在寄存器组 0。因此，RTC Timer 最多可同时记录两次触发的值。

19.4 事件任务矩阵功能

在 ESP32-P4 中，RTC Timer 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 RTC Timer 的 ETM 任务，或者通过 RTC Timer 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 RTC Timer 相关的 ETM 任务和 ETM 事件。

RTC Timer 不支持 ETM 任务。

RTC Timer 可产生的 ETM 事件有：

- RTC_EVT_CMP：表示 RTC Timer 计时到达寄存器 RTC_TIMER_MAIN_TIMER_TAR_LOW0 和 RTC_TIMER_MAIN_TIMER_TAR_HIGH0 所配置的目标时刻 0。
- RTC_EVT_OVF：表示 48 位计时器计数溢出。
- RTC_EVT_TICK：表示 RTC 计时器递增 1 时触发的事件。

19.5 中断

ESP32-P4 的 RTC Timer 可以生成以下中断信号并发送给中断矩阵。

- LP_TIMER_REG_0_INTR
- LP_TIMER_REG_1_INTR

这些中断信号由 RTC Timer 的内部中断源生成。表 19.5-1 列出了 RTC Timer 的内部中断源、中断触发条件、以及生成的中断信号。

表 19.5-1. RTC Timer 的内部中断源

内部中断源	触发条件	中断信号
RTC_TIMER_CMP0_INT	RTC Timer 计时到达目标时刻 0	LP_TIMER_REG_0_INTR
RTC_TIMER_CMP1_INT	RTC Timer 计时到达目标时刻 1	LP_TIMER_REG_1_INTR
RTC_TIMER_OVERFLOW_INT	48 位计时器计数溢出	LP_TIMER_REG_0_INTR LP_TIMER_REG_1_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 19.6 寄存器列表。

19.6 寄存器列表

本小节的所有地址均为相对于 RTC Timer 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

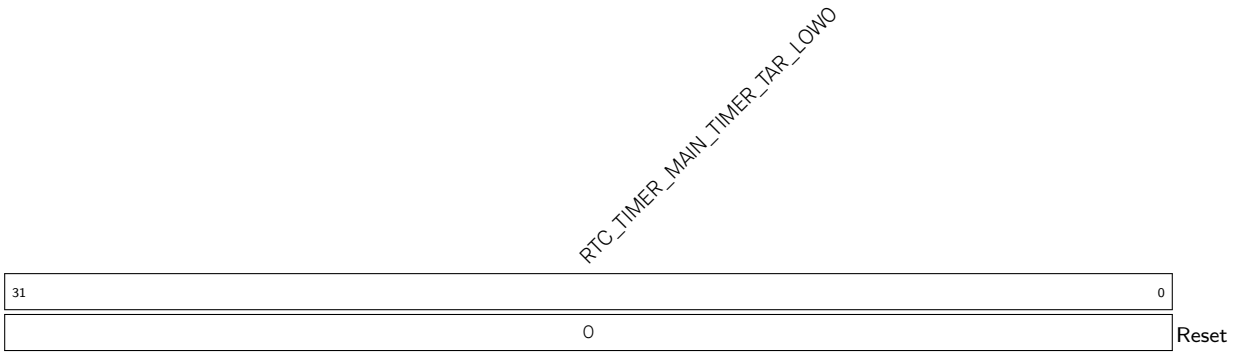
名称	描述	地址	访问
配置寄存器			
RTC_TIMER_TARO_LOW_REG	目标时刻 0，低 32 位	0x0000	R/W
RTC_TIMER_TARO_HIGH_REG	目标时刻 0，高 16 位	0x0004	varies
RTC_TIMER_TAR1_LOW_REG	目标时刻 1，低 32 位	0x0008	R/W
RTC_TIMER_TAR1_HIGH_REG	目标时刻 1，高 16 位	0x000C	varies
RTC_TIMER_UPDATE_REG	锁存计时器当前值使能	0x0010	varies
RTC_TIMER_MAIN_BUFO_LOW_REG	计时器缓存值 0，低 32 位	0x0014	RO
RTC_TIMER_MAIN_BUFO_HIGH_REG	计时器缓存值 0，高 16 位	0x0018	RO
RTC_TIMER_MAIN_BUF1_LOW_REG	计时器缓存值 1，低 32 位	0x001C	RO
RTC_TIMER_MAIN_BUF1_HIGH_REG	计时器缓存值 1，高 16 位	0x0020	RO
中断寄存器			
RTC_TIMER_INT_RAW_REG	计时器到达目标时刻 0 的原始中断状态寄存器	0x0028	RO
RTC_TIMER_INT_ST_REG	计时器到达目标时刻 0 的中断状态寄存器	0x002C	RO
RTC_TIMER_INT_ENA_REG	计时器到达目标时刻 0 的中断使能寄存器	0x0030	R/W
RTC_TIMER_INT_CLR_REG	计时器到达目标时刻 0 的中断清除寄存器	0x0034	WO
RTC_TIMER_LP_INT_RAW_REG	计时器到达目标时刻 1 的原始中断状态寄存器	0x0038	RO
RTC_TIMER_LP_INT_ST_REG	计时器到达目标时刻 1 的中断状态寄存器	0x003C	RO
RTC_TIMER_LP_INT_ENA_REG	计时器到达目标时刻 1 的中断使能寄存器	0x0040	R/W
RTC_TIMER_LP_INT_CLR_REG	计时器到达目标时刻 1 的中断清除寄存器	0x0044	WO
版本寄存器			
RTC_TIMER_DATE_REG	版本控制寄存器	0x03FC	R/W

19.7 寄存器

本小节的所有地址均为相对于 RTC Timer 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

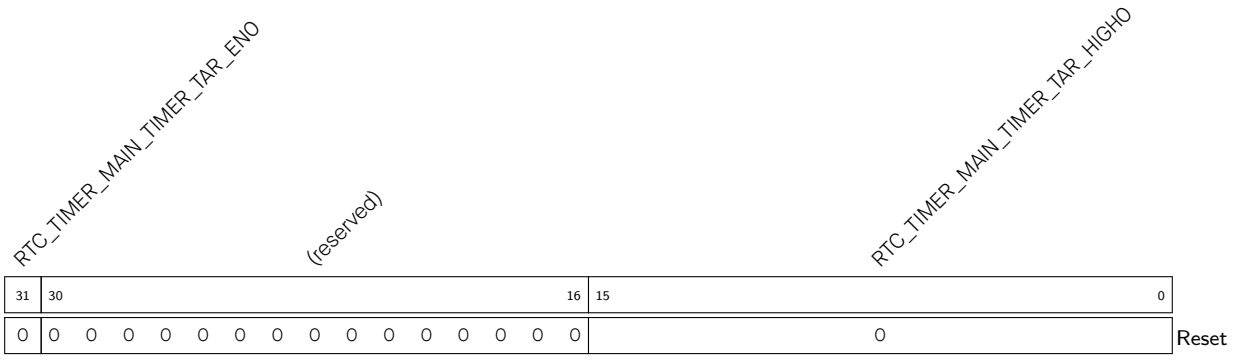
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 19.1. RTC_TIMER_TAR0_LOW_REG (0x0000)



RTC_TIMER_MAIN_TIMER_TAR_LOW0 计时器的目标时刻 0，低 32 位。(R/W)

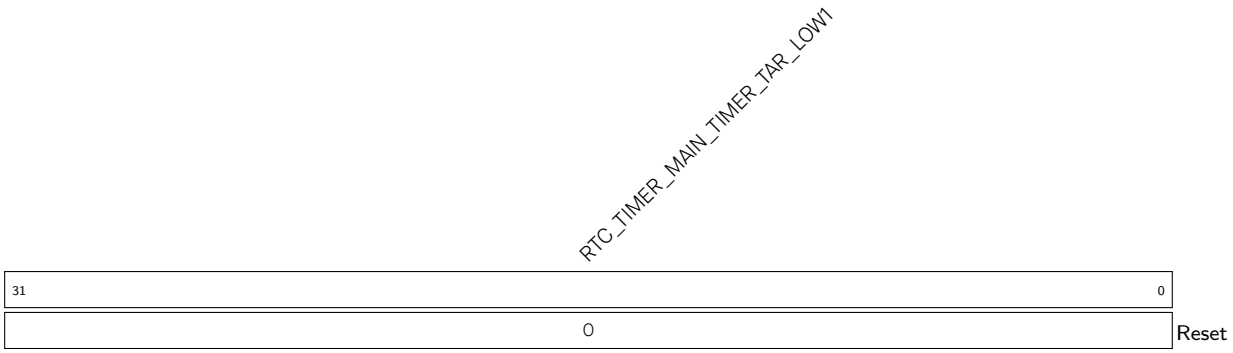
Register 19.2. RTC_TIMER_TAR0_HIGH_REG (0x0004)



RTC_TIMER_MAIN_TIMER_TAR_HIGH0 计时器的目标时刻 0，高 16 位。(R/W)

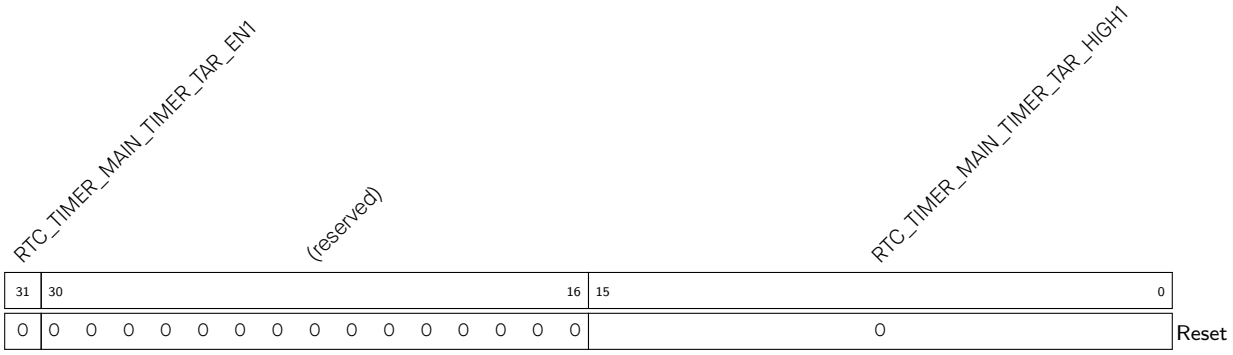
RTC_TIMER_MAIN_TIMER_TAR_EN0 计时器的目标时刻 0 使能。(WO)

Register 19.3. RTC_TIMER_TAR1_LOW_REG (0x0008)



RTC_TIMER_MAIN_TIMER_TAR_LOW1 计时器的目标时刻 1，低 32 位。(R/W)

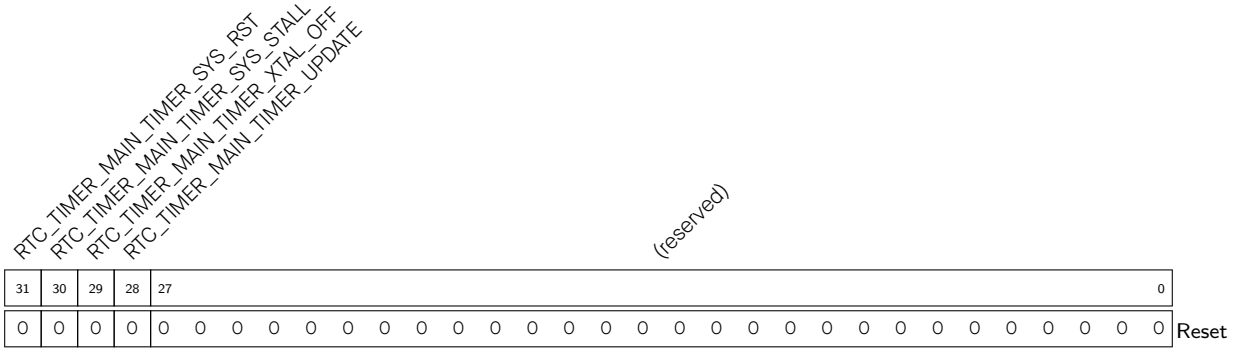
Register 19.4. RTC_TIMER_TAR1_HIGH_REG (0x000C)



RTC_TIMER_MAIN_TIMER_TAR_HIGH1 计时器的目标时刻 1，高 16 位。(R/W)

RTC_TIMER_MAIN_TIMER_TAR_EN1 计时器的目标时刻 1 使能。(WO)

Register 19.5. RTC_TIMER_UPDATE_REG (0x0010)



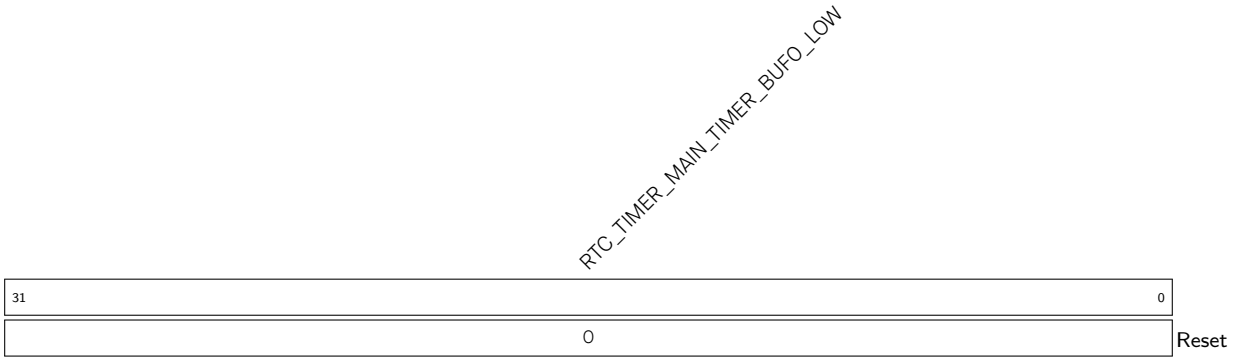
RTC_TIMER_MAIN_TIMER_UPDATE 软件配置记录当前时刻。(WO)

RTC_TIMER_MAIN_TIMER_XTAL_OFF 记录晶振时钟开启或关闭时刻使能。(R/W)

RTC_TIMER_MAIN_TIMER_SYS_STALL 记录 CPU 进入或退出 stall 状态时刻使能。(R/W)

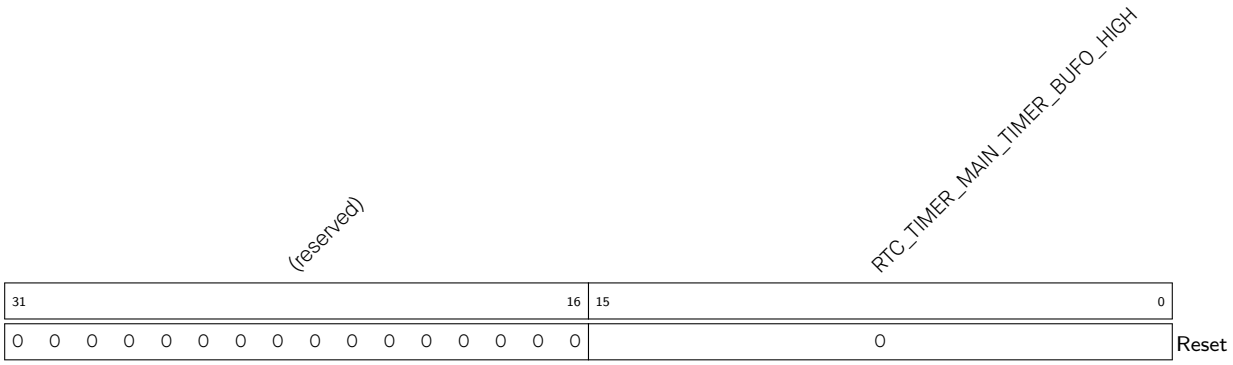
RTC_TIMER_MAIN_TIMER_SYS_RST 记录系统复位时刻使能。(R/W)

Register 19.6. RTC_TIMER_MAIN_BUF0_LOW_REG (0x0014)



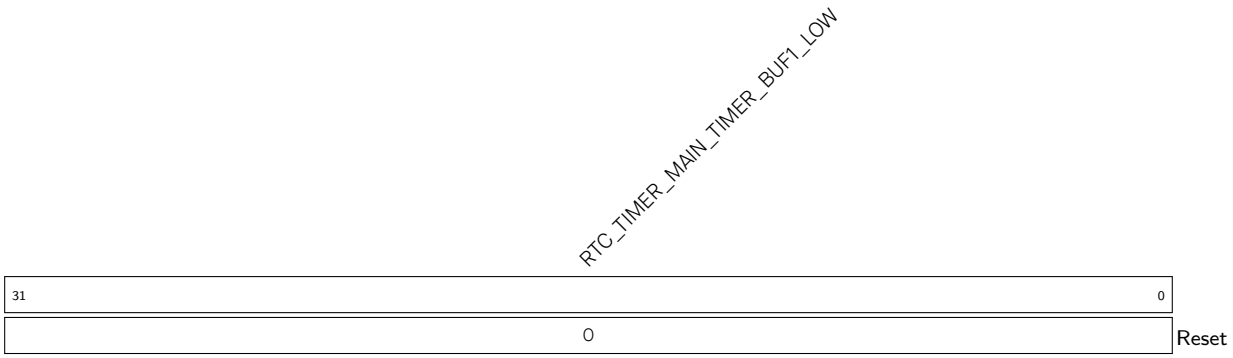
RTC_TIMER_MAIN_TIMER_BUF0_LOW 计时器缓存值 0，低 32 位。(RO)

Register 19.7. RTC_TIMER_MAIN_BUF0_HIGH_REG (0x0018)



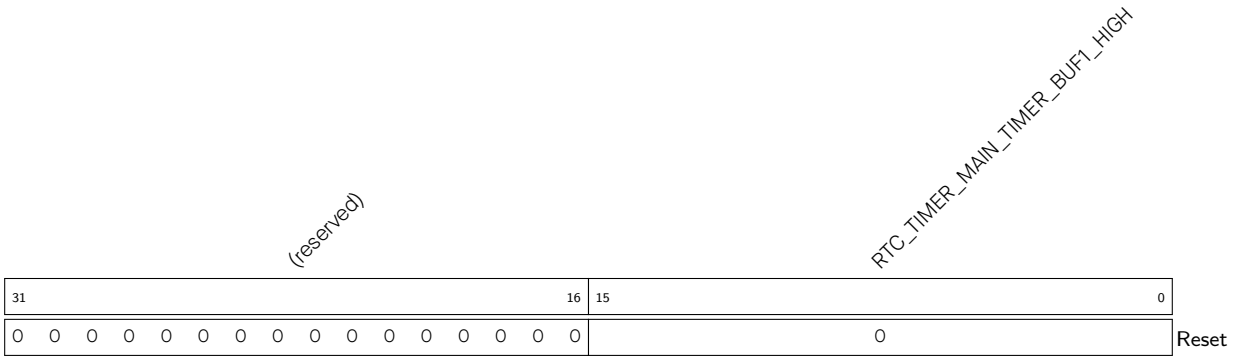
RTC_TIMER_MAIN_TIMER_BUF0_HIGH 计时器缓存值 0，高 16 位。(RO)

Register 19.8. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



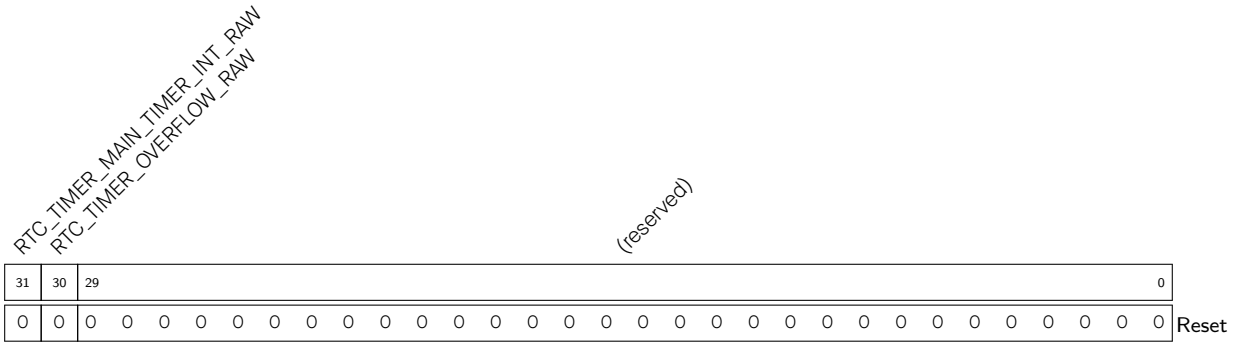
RTC_TIMER_MAIN_TIMER_BUF1_LOW 计时器缓存值 1，低 32 位。(RO)

Register 19.9. RTC_TIMER_MAIN_BUF1_HIGH_REG (0x0020)



RTC_TIMER_MAIN_TIMER_BUF1_HIGH 计时器缓存值 1，高 16 位。(RO)

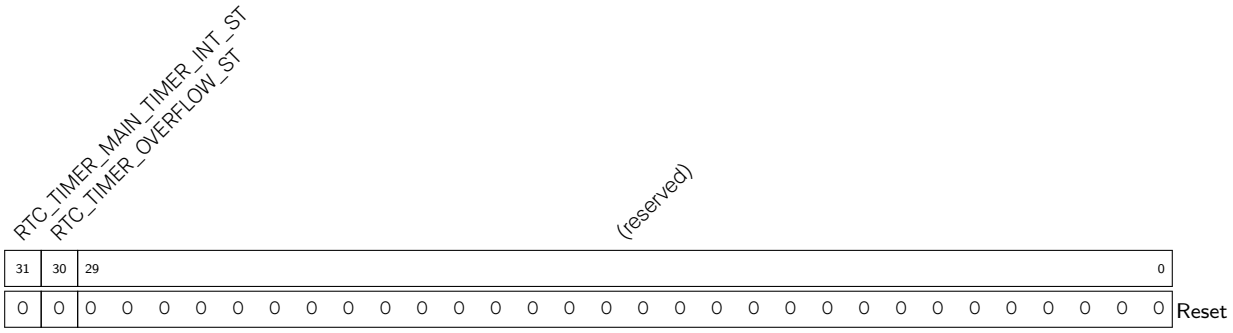
Register 19.10. RTC_TIMER_INT_RAW_REG (0x0028)



RTC_TIMER_OVERFLOW_RAW 48 位计时器溢出的原始中断。(RO)

RTC_TIMER_MAIN_TIMER_INT_RAW 计时器到达目标时刻 0 的原始中断。(RO)

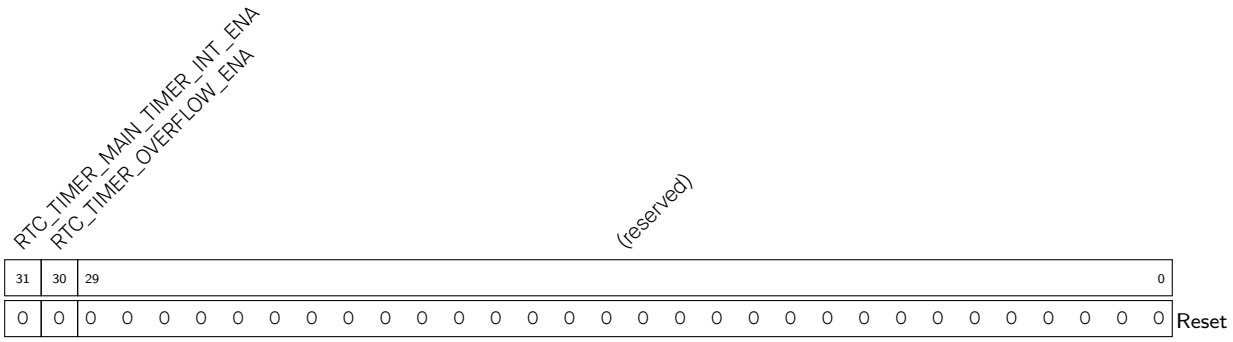
Register 19.11. RTC_TIMER_INT_ST_REG (0x002C)



RTC_TIMER_OVERFLOW_ST 48 位计时器溢出的状态中断。(RO)

RTC_TIMER_MAIN_TIMER_INT_ST 计时器到达目标时刻 0 的状态中断。(RO)

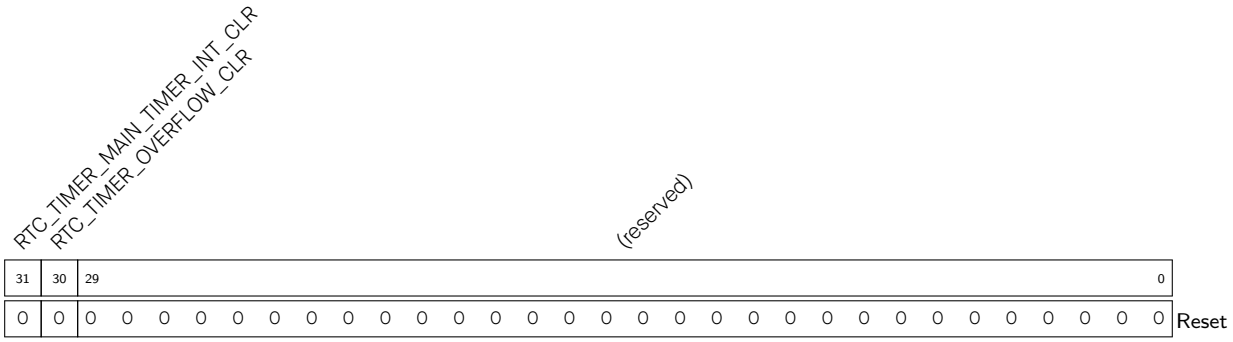
Register 19.12. RTC_TIMER_INT_ENA_REG (0x0030)



RTC_TIMER_OVERFLOW_ENA 48 位计时器溢出的中断使能。(R/W)

RTC_TIMER_MAIN_TIMER_INT_ENA 计时器到达目标时刻 0 的中断使能。(R/W)

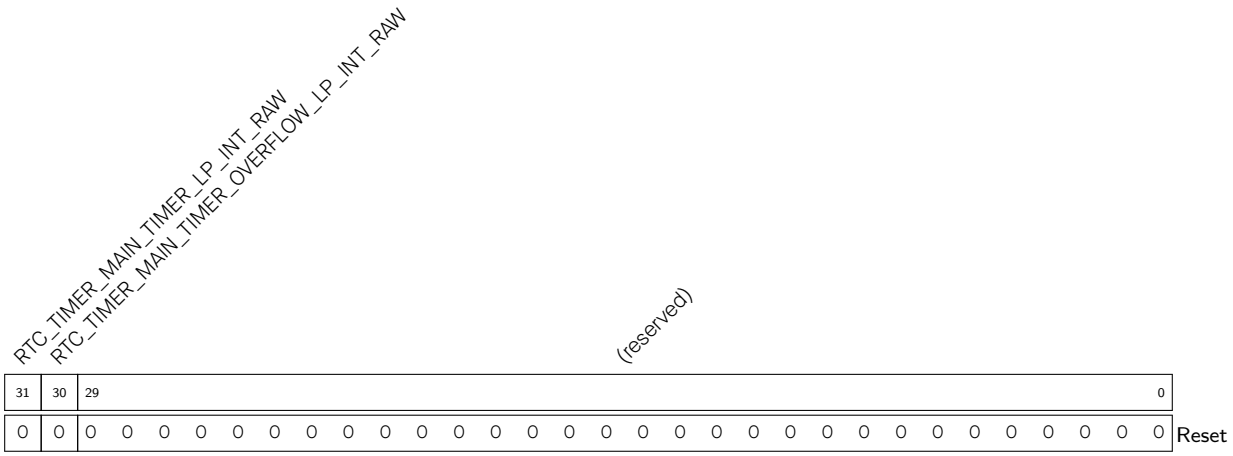
Register 19.13. RTC_TIMER_INT_CLR_REG (0x0034)



RTC_TIMER_OVERFLOW_CLR 48 位计时器溢出中断清除。(WO)

RTC_TIMER_MAIN_TIMER_INT_CLR 计时器到达目标时刻 0 的中断清除。(WO)

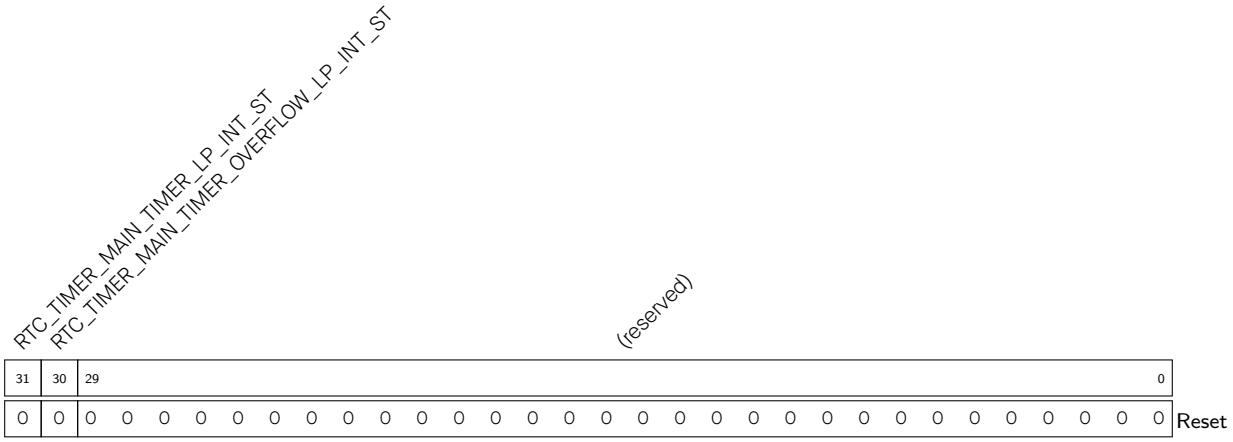
Register 19.14. RTC_TIMER_LP_INT_RAW_REG (0x0038)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_RAW 48 位计时器溢出的原始中断状态。(RO)

RTC_TIMER_MAIN_TIMER_LP_INT_RAW 计时器到达目标时刻 1 的原始中断。(RO)

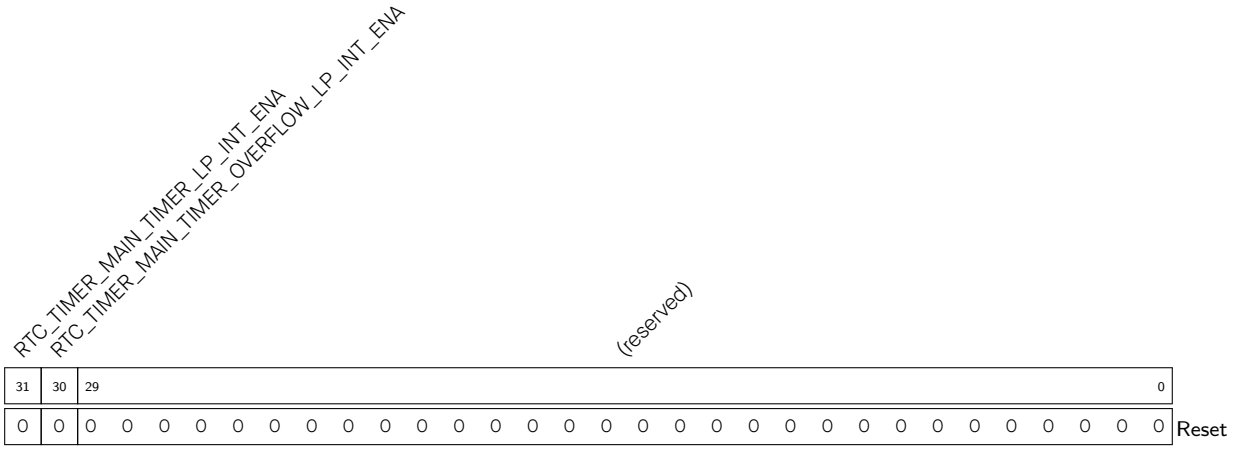
Register 19.15. RTC_TIMER_LP_INT_ST_REG (0x003C)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ST 48 位计时器溢出的状态中断。(RO)

RTC_TIMER_MAIN_TIMER_LP_INT_ST 计时器到达目标时刻 1 的状态中断。(RO)

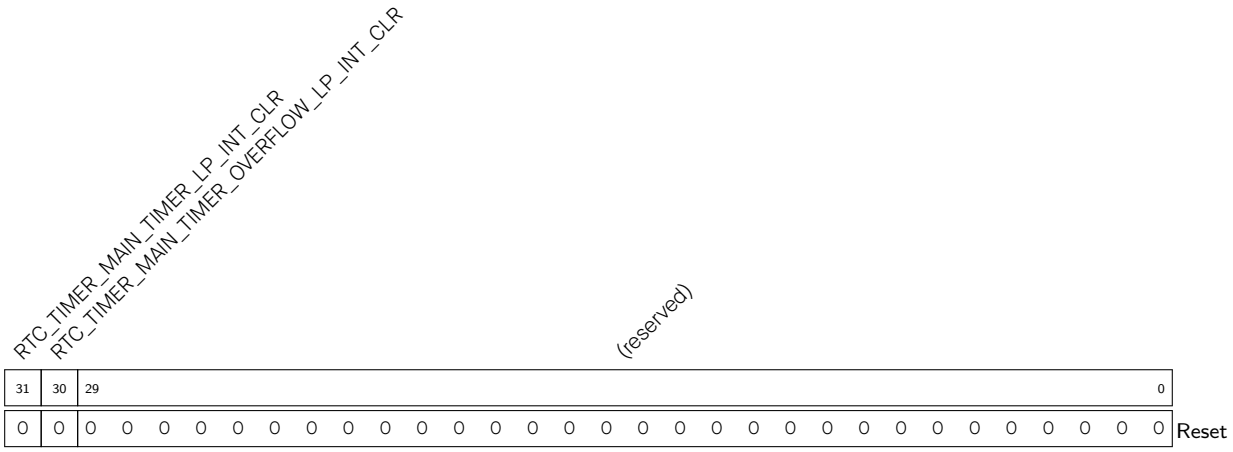
Register 19.16. RTC_TIMER_LP_INT_ENA_REG (0x0040)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ENA 48 位计时器溢出的中断使能。(R/W)

RTC_TIMER_MAIN_TIMER_LP_INT_ENA 计时器到达目标时刻 1 的中断使能。(R/W)

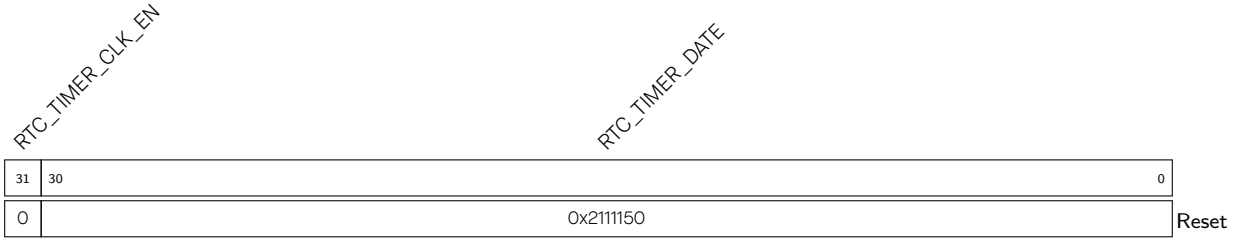
Register 19.17. RTC_TIMER_LP_INT_CLR_REG (0x0044)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_CLR 48 位计时器溢出中断清除。(WO)

RTC_TIMER_MAIN_TIMER_LP_INT_CLR 计时器到达目标时刻 1 的中断清除。(WO)

Register 19.18. RTC_TIMER_DATE_REG (0x03FC)



RTC_TIMER_DATE 版本控制寄存器。(R/W)

RTC_TIMER_CLK_EN 寄存器读写时钟使能。(R/W)

第 20 章

权限控制 (PMS)

20.1 概述

ESP32-P4 整个系统的权限控制可以分为两部分：PMP（Physical Memory Protection，物理存储器保护）和 APM（Access Permission Management，访问权限管理）。

PMP 位于 HP CPU 内部，能够管控 HP CPU 访问所有的地址空间。APM 则是一个作用于总线端口处的访问权限管理模块，不仅可以独立管理 CPU（HP CPU0/1 和 LP CPU）访问全片外设内部寄存器（见表 20.1-1 中 HP CPU PERI、HP PERI 和 LP PERI）、部分内存和外存的访问权限，还可以独立控制 VDMA，USB OTG 2.0 等支持 DMA 传输的主设备访问部分内存和外存的访问权限。

APM 通过与预先配置的地址区间（或固定的地址区间）及每个地址区间对应的访问权限，以及总线上所携带的信息，如主机 ID 号、模式、访问地址等信息作对比，从而判断是否允许访问。通过 APM，用户可实现精确控制所有主机对存储器和外设寄存器的访问权限。

PMP 和 APM 的管控区域分布如表 20.1-1 所示。对于主机 HP CPU0/1，PMP 和 APM 的管控关系见图 20.1-1。

表 20.1-1. PMP 和 APM 的管控区域

主机 / 从机	HP CPU0	HP CPU1	LP CPU	DMA 主机[1]
HP ROM[2]	PMP	PMP	HP APM	DMA APM
HP SPM	PMP	PMP	N/A	N/A
HP L2MEM[2]	PMP	PMP	HP APM	DMA APM
EXT MEM[2]	PMP	PMP	HP APM	DMA APM
Direct Access HP ROM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
Direct Access HP L2MEM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
Direct Access EXT MEM[3]	PMP + HP APM	PMP + HP APM	N/A	N/A
HP CPU PERI[4]	PMP + HP APM	PMP + HP APM	HP APM	N/A
HP PERI[5]	PMP + HP APM	PMP + HP APM	HP APM	N/A
LP PERI[6]	PMP + LP APM	PMP + LP APM	LP APM	N/A
LP ROM	N/A	N/A	N/A	N/A
LP SRAM	PMP + LP APM	PMP + LP APM	N/A	N/A

[1] 如 VDMA、USB 2.0 OTG、MEM_MONITOR 等其他支持 DMA 传输的主机。

[2]
- HP ROM (0x4FC0_0000 - 0x4FC1_FFFF)
- HP L2MEM (0x4FF0_0000 - 0x4FFB_FFFF)
- EXT MEM 包含外部 flash (0x4000_0000 - 0x43FF_FFFF) 和外部 RAM (0x4800_0000 - 0x4BFF_FFFF)

[3] 指 HP CPU0 和 HP CPU1 不经过 cache 直接访问内存和外存：
- HP ROM (0x8FC0_0000 - 0x8FC1_FFFF)
- HP L2MEM (0x8FF0_0000 - 0x8FFB_FFFF)
- EXT MEM 包含外部 flash (0x8000_0000 - 0x83FF_FFFF) 和外部 RAM (0x8800_0000 - 0x8BFF_FFFF)

[4] HP CPU 相关外设寄存器 (0x3FF0_0000 - 0x3FF1_FFFF)。

[5] HP 系统相关外设寄存器 (0x5000_0000 - 0x500F_FFFF)。

[6] LP 系统相关外设寄存器 (0x5011_0000 - 0x5012_FFFF)。

[7] 小节 20.3.1 将介绍 APM 分为 HP APM、LP APM 和 DMA APM。

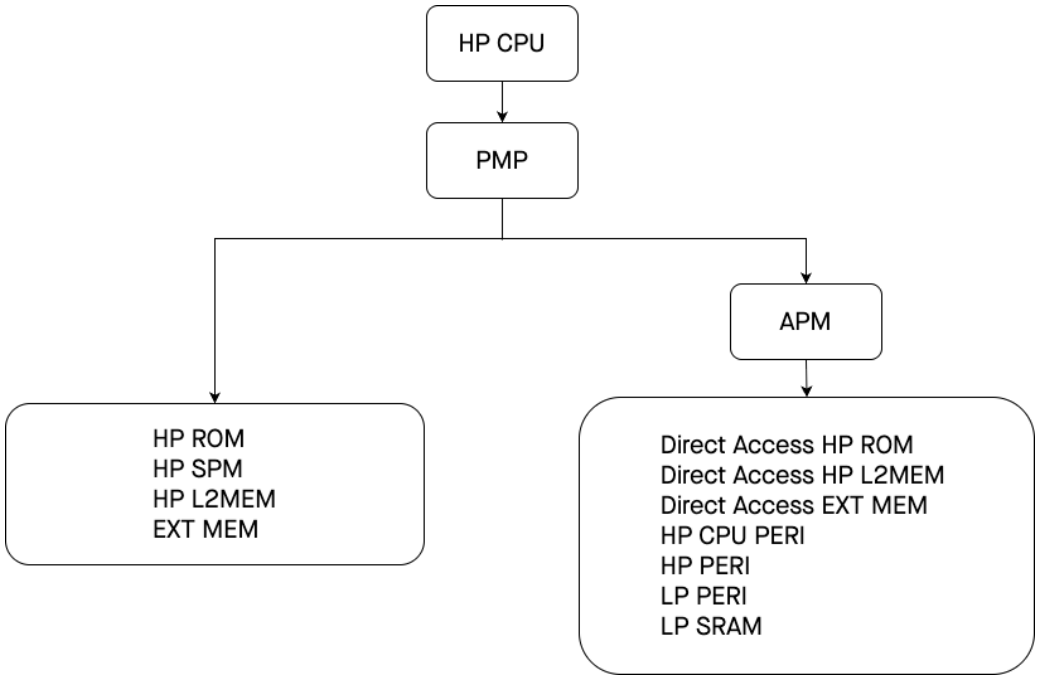


图 20.1-1. PMP 和 APM 的管控关系

如图所示，HP CPU 访问 HP ROM、HP SPM、HP L2MEM 和 EXT MEM 只受 PMP 管控，APM 无法管控。HP CPU0 和 HP CPU1 访问全片外设寄存器以及不通过 cache 直接访问 HP ROM、HP L2MEM 以及 EXT MEM，不仅需要先接受 PMP 的权限管控，还需要再接受 APM 的权限管控。如果 PMP 检查不通过，则不会再触发 APM 权限管控。

PMP 相关的寄存器位于 HP CPU 内部，需要特殊指令进行获取或配置。更多有关 PMP 配置和使用的内容，请参考章节 高性能处理器 > 2.10.1 标准物理存储器保护。

以下章节将详细介绍 APM 模块的特性、功能与配置。当提到 HP CPU0/1 访问 HP ROM/HP L2MEM/EXT MEM 时都是指 Direct Access HP ROM/HP L2MEM/EXT MEM。

20.2 主要特性

ESP32-P4 的 APM 控制器具有以下特性：

- 最多可为 DMA 主机配置 32 组有效地址区间

- 支持独立控制 CPU 每种模式下对内部存储器、外部存储器以及外设寄存器的访问权限
- 支持中断功能
- 支持异常信息记录功能

20.3 功能描述

20.3.1 结构概览

APM 权限管理模块可分为 3 个部分，共有 5 个寄存器模块：

- **DMA APM**，寄存器模块为 HP_DMA_PMS_REG
  - 负责管理所有 DMA 主机可访问的地址区间，最多可管理 32 个地址区间
  - 可独立控制每一个 DMA 主机读写每一个地址区间的访问权限
- **HP APM**，寄存器模块为 HP_PERI_PMS_REG 和 LP2HP_PERI_PMS_REG，负责管理 HP 系统中外设寄存器 (HP CPU PERI, HP PERI)、内存 (HP ROM, HP L2MEM) 和外存 (EXT MEM) 的访问权限，具体为：
  - 分别控制 HP CPUO/1 在用户模式下访问上述所有从机的权限
  - 分别控制 HP CPUO/1 在机器模式下访问上述所有从机的权限
  - 控制 LP CPU 在机器模式下访问上述所有从机的权限
- **LP APM**，寄存器模块为 LP_PERI_PMS_REG 和 HP2LP_PERI_PMS_REG，负责管理 LP 系统中外设寄存器 (LP PERI) 和内存 (LP SRAM) 的访问权限，具体为：
  - 分别控制 HP CPUO/1 在用户模式下访问 LP PERI 和 LP SRAM 的权限
  - 分别控制 HP CPUO/1 在机器模式下访问 LP PERI 和 LP SRAM 的权限
  - 控制 LP CPU 在机器模式下访问 LP PERI 的权限

图 20.3-1 显示了 APM 管控的总线访问路径以及对应的寄存器模块。

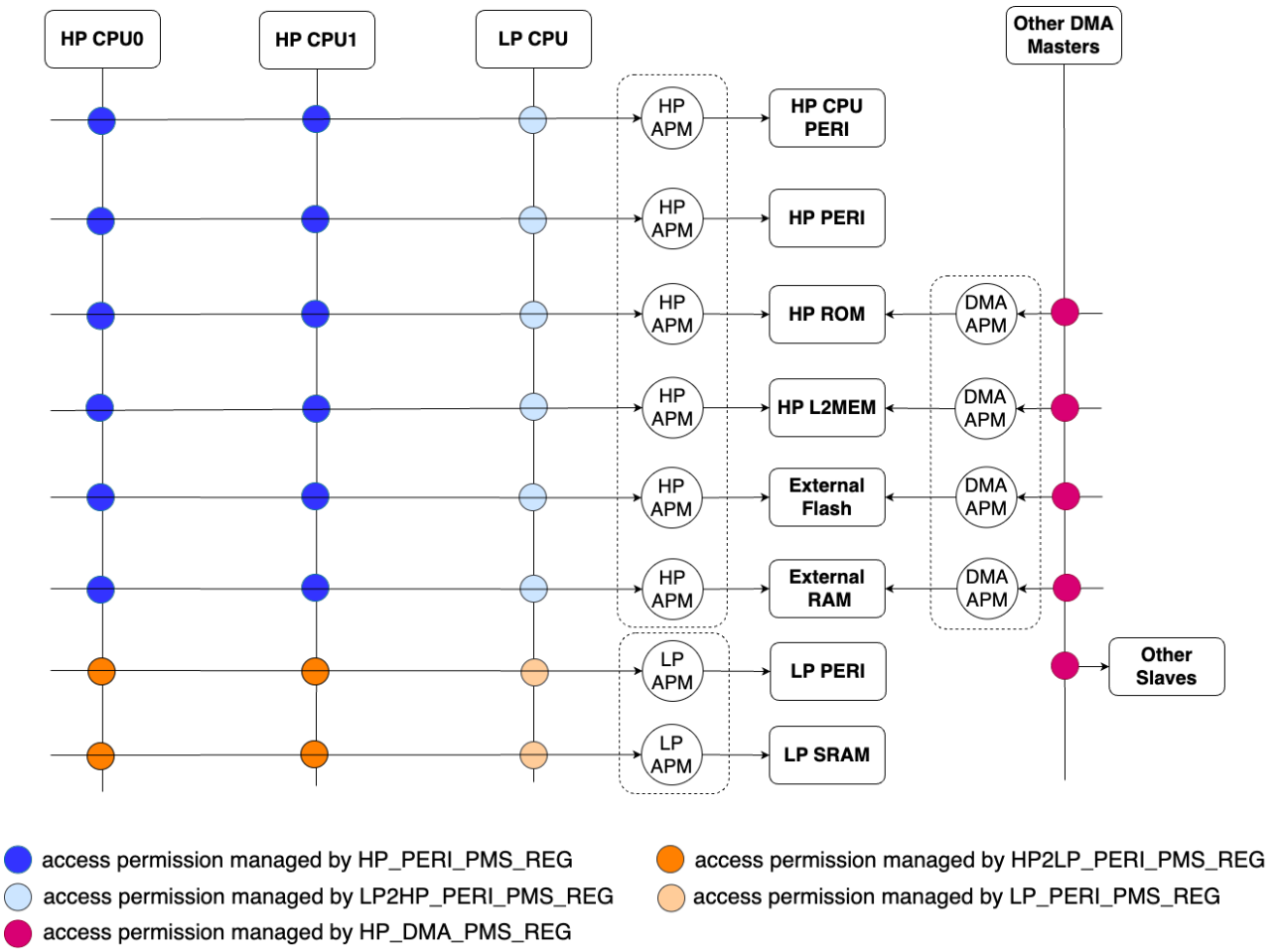


图 20.3-1. APM 控制器结构图

说明：

- 寄存器名称“HP2LP”及“LP2HP”中的“HP”主要是指 HP CPU0/1，“LP”主要是指 LP CPU。
- 由表 20.1-1 可知，HP CPU 和 LP CPU 访问 HP ROM、HP L2MEM、外部 flash 和 RAM 的地址空间不同。对于 HP CPU0/1，HP APM 管理的是 HP CPU0/1 不经过 cache 直接访问内存和外存的通路。
- LP_PERI_PMS_REG 中还额外包含了 8 个可配置地址区间及其对应的访问权限，可作用于 HP APM 和 LP APM，以管理全片外设寄存器（HP CPU PERI、HP PERI、LP PERI）的访问权限。

20.3.2 地址区间及访问权限

APM 管理的地址区间分为固定地址区间和可配置地址区间。DMA APM 最多可管理 32 组可配置地址区间，HP APM 和 LP APM 除了控制固定地址区间外，还可管理 8 个可配置地址区间。

20.3.2.1 HP APM 和 LP APM 管理的地址区间及访问权限

HP APM 和 LP APM 控制的访问通路上，内部存储器和外部存储器通路采用固定地址区间管理访问权限，而全片外设寄存器通路上不仅支持固定地址区间的权限控制，还支持 8 个可配置地址区间。

- 固定地址区间
  - 固定地址区间即各个外设寄存器、内存和外存的有效地址空间。内存和外存地址空间见表 20.1-1，各个外设寄存器的地址空间详见章节 9 *系统和存储器* > 表 9.3-2。
  - 要管理固定地址区间的访问权限，请参考图 20.3-1 确定访问通路及其对应的寄存器模块。
- 可配置地址区间
  - APM 可通过 LP_PERI_PMS_REG 中的 PMS_PERI_REGION*n*/*m*_LOW_REG 和 PMS_PERI_REGION*n*/*m*_HIGH_REG（*n*/*m* = 0 ~ 7）配置 8 个地址区间，作用于 LP CPU 和 HP CPU0/1 到全片外设寄存器的访问通路。
  - 可通过 LP_PERI_PMS_REG 中 PMS_PERI_REGION_PMS_REG 管理可配置地址区间的访问权限。

对全片外设寄存器访问进行访问权限管理时，可配置地址区间权限管理要先于固定地址区间权限管理，具体为：

- 若访问地址在可配置地址区间内，则本次访问首先会接受可配置地址区间的检查。
  - 如果有权限，则本次访问将继续接受后续固定地址区间的权限管理。
  - 如果该访问没有权限，则访问被禁止，不会接受后续固定地址区间的权限管理。
- 若访问地址在可配置地址区间之外，则本次访问直接跳过可配置地址区间检查，直接接受固定地址区间的权限管理。
- 可配置区间管理功能是对固定区间管理的补充。例如，配置了某固定地址区间的访问权限，将其中 2 个可配置地址区间限定在该固定地址区间内，并禁用其访问权限，便可实现在固定地址区间中设置 2 个禁止访问的地址区间。

20.3.2.2 DMA APM 管理的地址区间及访问权限

寄存器模块 HP_DMA_PMS_REG 中提供了 32 组可配置地址区间 (PMS_DMA_REGION*n*_LOW_REG ~ PMS_DMA_REGION*n*_HIGH_REG, *n* = 0 ~ 31)，控制所有 DMA 主机的访问地址空间。配置该地址区间时，由于地址最小以 4K 字节对齐，故低 12 位为 0。

HP_DMA_PMS_REG 为每一个 DMA 主机访问每一个地址区间分配了独立的读权限和写权限。

20.4 配置流程

由于 HP CPU 有不同的工作模式（用户模式和机器模式），HP APM 和 LP APM 为不同工作模式分配了相应的访问管理权限。因此当 HP CPU 作为主机访问内存、外存或外设寄存器时，首先需要配置 HP CPU 的工作模式。

由于 LP CPU 仅支持机器模式，因此 HP APM 和 LP APM 仅为 LP CPU 的机器模式分配了访问权限位。

支持 DMA 传输的主设备不分访问模式，但 DMA APM 为 DMA 主设备分别提供了读和写操作管理权限。

20.4.1 配置 HP CPU0/1 的访问权限

HP CPU0/1 访问内存、外存或者外设寄存器路径中 APM 管控的配置流程为：

1. 配置 HP CPU0/1 工作模式（用户模式或机器模式）。关于 CPU 用户模式及机器模式的配置方法请参考 RISC-V 指令集手册 v1.10 第二卷 “特权架构”（RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10）。
2. 如果访问外设寄存器，可额外管理 8 个可配置地址区间，请参考章节 20.3.2。
3. 配置访问权限：
   - 通过 HP_PERI_PMS_REG 配置 HP 系统一侧的内存（HP L2MEM 和 HP ROM）、外存（External RAM 和 flash）和外设寄存器（HP PERI 和 HP CPU PERI）的访问权限：
     - 通过 PMS_CORE*n*_*MODE_NAME*_*MODULE_NAME*_ALLOW 管理 HP CPU*n* 在 *MODE_NAME* 模式下访问 *MODULE_NAME* 模块的权限，比如 PMS_CORE*n*_MM_PSRAM_ALLOW。
   - 通过 HP2LP_PERI_PMS_REG 配置 LP SRAM 和 LP PERI 的访问权限：
     - 通过 PMS_HP_CORE*n*_*MODE_NAME*_*MODULE_NAME*_ALLOW 管理 HP CPU*n* 在 *MODE_NAME* 模式下访问 *MODULE_NAME* 模块的权限，比如 PMS_HP_CORE*n*_MM_LP_SYSREG_ALLOW。

20.4.2 配置 LP CPU 的访问权限

LP CPU 仅支持机器模式，LP CPU 访问内存、外存或者外设寄存器路径中 APM 管控的配置流程为：

1. 如果访问外设寄存器，可额外管理 8 个可配置地址区间，请参考章节 20.3.2。
2. 配置访问权限：
   - 通过 LP2HP_PERI_PMS_REG 配置 HP 系统一侧的内存（HP L2MEM 和 HP ROM）、外存（External RAM 和 flash）和外设寄存器（HP PERI 和 HP CPU PERI）的访问权限：
     - 通过 PMS_LP_MM_*MODULE_NAME*_ALLOW 管理 LP CPU 在机器模式下访问 *MODULE_NAME* 模块的权限，比如 PMS_LP_MM_PSRAM_ALLOW。
   - 通过 LP_PERI_PMS_REG 配置 LP 一侧的外设寄存器 (LP PERI) 的访问权限：
     - 通过 PMS_LP_MM_*MODULE_NAME*_ALLOW 管理 LP CPU 在机器模式下访问 *MODULE_NAME* 模块的权限，比如 PMS_LP_MM_LP_SYSREG_ALLOW。

20.4.3 配置其他 DMA 主机的访问权限

DMA 主机访问内存、外存以及其它地址空间路径中 APM 管控的配置流程为：

1. 配置所有 DMA 主机可访问的地址区间：
   - 通过 PMS_DMA_REGION*n*_LOW_REG 配置 region*n* 的起始地址的高 20 位
   - 通过 PMS_DMA_REGION*n*_HIGH_REG 配置 region*n* 的结束地址的高 20 位
2. 配置每个 DMA 主机 (*DMA_MASTER*) 各自访问每一个地址区间的读写权限：
   - 通过 PMS_*DMA_MASTER*_PMS_R_REG 配置读操作权限，如 PMS_DMA_GMAC_PMS_R_REG
   - 通过 PMS_*DMA_MASTER*_PMS_W_REG 配置写操作权限，如 PMS_DMA_GMAC_PMS_W_REG

20.5 越权访问与中断

APM 模块管理总线访问通路时，会解析总线携带的信息（主机 ID、访问方向、访问地址、甚至是运行模式），然后根据此信息确定该访问的管理权限配置。当 APM 确认此次访问无权限时，ESP32-P4 将视其为越权访问，将进行如下处理：

- 拒绝该访问请求，并作错误响应
- HP APM 会触发 ICM_SYS_REG_INT 中断信号
- DMA APM 会触发 ICM_SYS_REG_INT 中断信号
- LP APM 会触发 LP_SYSREG_INT 中断信号

这些中断信号由 APM 管理模块的内部中断源生成。表 20.5-1 列出了 APM 管理模块的内部中断源、中断触发条件、以及生成的中断信号。

表 20.5-1. APM 内部中断源

内部中断源	触发条件	中断信号
ICM_CPU_ADDRHOLE_INTR	HP APM 检测到越权访问；或当主机为 USB 2.0 高速 OTG、USB 2.0 全速 OTG、EMAC、SDMMC、Trace0/1、GDMA-AHB、L2MEM 监测器和 SPM 监测器时，DMA APM 检测到越权访问	ICM_SYS_REG_INT
ICM_SYS_ADDRHOLE_INTR	当主机为 VDMA、GDMA-AXI、2D-DMA 和 H264 DMA 时 DMA APM 发生了越权访问	ICM_SYS_REG_INT
LP_ADDRHOLE_INTR	LP APM 检测到越权访问	LP_SYSREG_INTR

APM 会自动记录非法访问的相关信息，包括主机 ID、访问方向（读或者写）、访问地址、非法访问标志等信息。所有这些信息可以从章节 21 *系统寄存器 (SYSREG)* 中的相关寄存器中获取。

20.6 寄存器列表

20.6.1 HP_DMA_PMS_REG

本小节的所有地址均为相对于 HP_DMA_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
版本控制寄存器			
PMS_DMA_DATE_REG	版本控制寄存器	0x0000	R/W
时钟门控寄存器			
PMS_DMA_CLK_EN_REG	时钟门控寄存器	0x0004	R/W
地址区间配置寄存器			
PMS_DMA_REGION0_LOW_REG	地址区间 0 起始地址配置寄存器	0x0008	R/W
PMS_DMA_REGION0_HIGH_REG	地址区间 0 结束地址配置寄存器	0x000C	R/W
PMS_DMA_REGION1_LOW_REG	地址区间 1 起始地址配置寄存器	0x0010	R/W
PMS_DMA_REGION1_HIGH_REG	地址区间 1 结束地址配置寄存器	0x0014	R/W
PMS_DMA_REGION2_LOW_REG	地址区间 2 起始地址配置寄存器	0x0018	R/W
PMS_DMA_REGION2_HIGH_REG	地址区间 2 结束地址配置寄存器	0x001C	R/W
PMS_DMA_REGION3_LOW_REG	地址区间 3 起始地址配置寄存器	0x0020	R/W
PMS_DMA_REGION3_HIGH_REG	地址区间 3 结束地址配置寄存器	0x0024	R/W
PMS_DMA_REGION4_LOW_REG	地址区间 4 起始地址配置寄存器	0x0028	R/W
PMS_DMA_REGION4_HIGH_REG	地址区间 4 结束地址配置寄存器	0x002C	R/W
PMS_DMA_REGION5_LOW_REG	地址区间 5 起始地址配置寄存器	0x0030	R/W
PMS_DMA_REGION5_HIGH_REG	地址区间 5 结束地址配置寄存器	0x0034	R/W
PMS_DMA_REGION6_LOW_REG	地址区间 6 起始地址配置寄存器	0x0038	R/W
PMS_DMA_REGION6_HIGH_REG	地址区间 6 结束地址配置寄存器	0x003C	R/W
PMS_DMA_REGION7_LOW_REG	地址区间 7 起始地址配置寄存器	0x0040	R/W
PMS_DMA_REGION7_HIGH_REG	地址区间 7 结束地址配置寄存器	0x0044	R/W
PMS_DMA_REGION8_LOW_REG	地址区间 8 起始地址配置寄存器	0x0048	R/W
PMS_DMA_REGION8_HIGH_REG	地址区间 8 结束地址配置寄存器	0x004C	R/W
PMS_DMA_REGION9_LOW_REG	地址区间 9 起始地址配置寄存器	0x0050	R/W
PMS_DMA_REGION9_HIGH_REG	地址区间 9 结束地址配置寄存器	0x0054	R/W
PMS_DMA_REGION10_LOW_REG	地址区间 10 起始地址配置寄存器	0x0058	R/W
PMS_DMA_REGION10_HIGH_REG	地址区间 10 结束地址配置寄存器	0x005C	R/W
PMS_DMA_REGION11_LOW_REG	地址区间 11 起始地址配置寄存器	0x0060	R/W
PMS_DMA_REGION11_HIGH_REG	地址区间 11 结束地址配置寄存器	0x0064	R/W
PMS_DMA_REGION12_LOW_REG	地址区间 12 起始地址配置寄存器	0x0068	R/W
PMS_DMA_REGION12_HIGH_REG	地址区间 12 结束地址配置寄存器	0x006C	R/W
PMS_DMA_REGION13_LOW_REG	地址区间 13 起始地址配置寄存器	0x0070	R/W
PMS_DMA_REGION13_HIGH_REG	地址区间 13 结束地址配置寄存器	0x0074	R/W
PMS_DMA_REGION14_LOW_REG	地址区间 14 起始地址配置寄存器	0x0078	R/W
PMS_DMA_REGION14_HIGH_REG	地址区间 14 结束地址配置寄存器	0x007C	R/W

名称	描述	地址	访问
PMS_DMA_REGION15_LOW_REG	地址区间 15 起始地址配置寄存器	0x0080	R/W
PMS_DMA_REGION15_HIGH_REG	地址区间 15 结束地址配置寄存器	0x0084	R/W
PMS_DMA_REGION16_LOW_REG	地址区间 16 起始地址配置寄存器	0x0088	R/W
PMS_DMA_REGION16_HIGH_REG	地址区间 16 结束地址配置寄存器	0x008C	R/W
PMS_DMA_REGION17_LOW_REG	地址区间 17 起始地址配置寄存器	0x0090	R/W
PMS_DMA_REGION17_HIGH_REG	地址区间 17 结束地址配置寄存器	0x0094	R/W
PMS_DMA_REGION18_LOW_REG	地址区间 18 起始地址配置寄存器	0x0098	R/W
PMS_DMA_REGION18_HIGH_REG	地址区间 18 结束地址配置寄存器	0x009C	R/W
PMS_DMA_REGION19_LOW_REG	地址区间 19 起始地址配置寄存器	0x00A0	R/W
PMS_DMA_REGION19_HIGH_REG	地址区间 19 结束地址配置寄存器	0x00A4	R/W
PMS_DMA_REGION20_LOW_REG	地址区间 20 起始地址配置寄存器	0x00A8	R/W
PMS_DMA_REGION20_HIGH_REG	地址区间 20 结束地址配置寄存器	0x00AC	R/W
PMS_DMA_REGION21_LOW_REG	地址区间 21 起始地址配置寄存器	0x00B0	R/W
PMS_DMA_REGION21_HIGH_REG	地址区间 21 结束地址配置寄存器	0x00B4	R/W
PMS_DMA_REGION22_LOW_REG	地址区间 22 起始地址配置寄存器	0x00B8	R/W
PMS_DMA_REGION22_HIGH_REG	地址区间 22 结束地址配置寄存器	0x00BC	R/W
PMS_DMA_REGION23_LOW_REG	地址区间 23 起始地址配置寄存器	0x00C0	R/W
PMS_DMA_REGION23_HIGH_REG	地址区间 23 结束地址配置寄存器	0x00C4	R/W
PMS_DMA_REGION24_LOW_REG	地址区间 24 起始地址配置寄存器	0x00C8	R/W
PMS_DMA_REGION24_HIGH_REG	地址区间 24 结束地址配置寄存器	0x00CC	R/W
PMS_DMA_REGION25_LOW_REG	地址区间 25 起始地址配置寄存器	0x00D0	R/W
PMS_DMA_REGION25_HIGH_REG	地址区间 25 结束地址配置寄存器	0x00D4	R/W
PMS_DMA_REGION26_LOW_REG	地址区间 26 起始地址配置寄存器	0x00D8	R/W
PMS_DMA_REGION26_HIGH_REG	地址区间 26 结束地址配置寄存器	0x00DC	R/W
PMS_DMA_REGION27_LOW_REG	地址区间 27 起始地址配置寄存器	0x00E0	R/W
PMS_DMA_REGION27_HIGH_REG	地址区间 27 结束地址配置寄存器	0x00E4	R/W
PMS_DMA_REGION28_LOW_REG	地址区间 28 起始地址配置寄存器	0x00E8	R/W
PMS_DMA_REGION28_HIGH_REG	地址区间 28 结束地址配置寄存器	0x00EC	R/W
PMS_DMA_REGION29_LOW_REG	地址区间 29 起始地址配置寄存器	0x00F0	R/W
PMS_DMA_REGION29_HIGH_REG	地址区间 29 结束地址配置寄存器	0x00F4	R/W
PMS_DMA_REGION30_LOW_REG	地址区间 30 起始地址配置寄存器	0x00F8	R/W
PMS_DMA_REGION30_HIGH_REG	地址区间 30 结束地址配置寄存器	0x00FC	R/W
PMS_DMA_REGION31_LOW_REG	地址区间 31 起始地址配置寄存器	0x0100	R/W
PMS_DMA_REGION31_HIGH_REG	地址区间 31 结束地址配置寄存器	0x0104	R/W
DMA 主机读写权限配置寄存器			
PMS_DMA_GDMA_CH0_R_PMS_REG	GDMA ch0 读权限控制寄存器	0x0108	R/W
PMS_DMA_GDMA_CH0_W_PMS_REG	GDMA ch0 写权限控制寄存器	0x010C	R/W
PMS_DMA_GDMA_CH1_R_PMS_REG	GDMA ch1 读权限控制寄存器	0x0110	R/W
PMS_DMA_GDMA_CH1_W_PMS_REG	GDMA ch1 写权限控制寄存器	0x0114	R/W
PMS_DMA_GDMA_CH2_R_PMS_REG	GDMA ch2 读权限控制寄存器	0x0118	R/W
PMS_DMA_GDMA_CH2_W_PMS_REG	GDMA ch2 写权限控制寄存器	0x011C	R/W
PMS_DMA_GDMA_CH3_R_PMS_REG	GDMA ch3 读权限控制寄存器	0x0120	R/W
PMS_DMA_GDMA_CH3_W_PMS_REG	GDMA ch3 写权限控制寄存器	0x0124	R/W

名称	描述	地址	访问
PMS_DMA_AHB_PDMA_ADC_R_PMS_REG	GDMA-AHB ADC 读权限控制寄存器	0x0128	R/W
PMS_DMA_AHB_PDMA_ADC_W_PMS_REG	GDMA-AHB ADC 写权限控制寄存器	0x012C	R/W
PMS_DMA_AHB_PDMA_I2S0_R_PMS_REG	GDMA-AHB I2S0 读权限控制寄存器	0x0130	R/W
PMS_DMA_AHB_PDMA_I2S0_W_PMS_REG	GDMA-AHB I2S0 写权限控制寄存器	0x0134	R/W
PMS_DMA_AHB_PDMA_I2S1_R_PMS_REG	GDMA-AHB I2S1 读权限控制寄存器	0x0138	R/W
PMS_DMA_AHB_PDMA_I2S1_W_PMS_REG	GDMA-AHB I2S1 写权限控制寄存器	0x013C	R/W
PMS_DMA_AHB_PDMA_I2S2_R_PMS_REG	GDMA-AHB I2S2 读权限控制寄存器	0x0140	R/W
PMS_DMA_AHB_PDMA_I2S2_W_PMS_REG	GDMA-AHB I2S2 写权限控制寄存器	0x0144	R/W
PMS_DMA_AHB_PDMA_I3C_MST_R_PMS_REG	GDMA-AHB I3C MST 读权限控制寄存器	0x0148	R/W
PMS_DMA_AHB_PDMA_I3C_MST_W_PMS_REG	GDMA-AHB I3C MST 写权限控制寄存器	0x014C	R/W
PMS_DMA_AHB_PDMA_UHCI0_R_PMS_REG	GDMA-AHB UHCI 读权限控制寄存器	0x0150	R/W
PMS_DMA_AHB_PDMA_UHCI0_W_PMS_REG	GDMA-AHB UHCI 写权限控制寄存器	0x0154	R/W
PMS_DMA_AHB_PDMA_RMT_R_PMS_REG	GDMA-AHB RMT 读权限控制寄存器	0x0158	R/W
PMS_DMA_AHB_PDMA_RMT_W_PMS_REG	GDMA-AHB RMT 写权限控制寄存器	0x0170	R/W
PMS_DMA_AXI_PDMA_LCDCAM_R_PMS_REG	GDMA-AXI LCDCAM 读权限控制寄存器	0x0174	R/W
PMS_DMA_AXI_PDMA_LCDCAM_W_PMS_REG	GDMA-AXI LCDCAM 写权限控制寄存器	0x0178	R/W
PMS_DMA_AXI_PDMA_GPSPI2_R_PMS_REG	GDMA-AXI GP-SPI2 读权限控制寄存器	0x017C	R/W
PMS_DMA_AXI_PDMA_GPSPI2_W_PMS_REG	GDMA-AXI GP-SPI2 写权限控制寄存器	0x0180	R/W
PMS_DMA_AXI_PDMA_GPSPI3_R_PMS_REG	GDMA-AXI GP-SPI3 读权限控制寄存器	0x0184	R/W
PMS_DMA_AXI_PDMA_GPSPI3_W_PMS_REG	GDMA-AXI GP-SPI3 写权限控制寄存器	0x0188	R/W
PMS_DMA_AXI_PDMA_PARLIO_R_PMS_REG	GDMA-AXI PARLIO 读权限控制寄存器	0x018C	R/W
PMS_DMA_AXI_PDMA_PARLIO_W_PMS_REG	GDMA-AXI PARLIO 写权限控制寄存器	0x0190	R/W
PMS_DMA_AXI_PDMA_AES_R_PMS_REG	GDMA-AXI AES 读权限控制寄存器	0x0194	R/W
PMS_DMA_AXI_PDMA_AES_W_PMS_REG	GDMA-AXI AES 写权限控制寄存器	0x0198	R/W
PMS_DMA_AXI_PDMA_SHA_R_PMS_REG	GDMA-AXI SHA 读权限控制寄存器	0x019C	R/W
PMS_DMA_AXI_PDMA_SHA_W_PMS_REG	GDMA-AXI SHA 写权限控制寄存器	0x01A0	R/W
PMS_DMA_DMA2D_JPEG_PMS_R_REG	2D-DMA JPEG 读权限控制寄存器	0x01A4	R/W
PMS_DMA_DMA2D_JPEG_PMS_W_REG	2D-DMA JPEG 写权限控制寄存器	0x01A8	R/W
PMS_DMA_USB_PMS_R_REG	USB 2.0 高速 OTG 读权限控制寄存器	0x01AC	R/W
PMS_DMA_USB_PMS_W_REG	USB 2.0 高速 OTG 写权限控制寄存器	0x01B0	R/W
PMS_DMA_GMAC_PMS_R_REG	EMAC 读权限控制寄存器	0x01B4	R/W
PMS_DMA_GMAC_PMS_W_REG	EMAC 写权限控制寄存器	0x01B8	R/W
PMS_DMA_SDMMC_PMS_R_REG	SDMMC 读权限控制寄存器	0x01BC	R/W
PMS_DMA_SDMMC_PMS_W_REG	SDMMC 写权限控制寄存器	0x01C0	R/W
PMS_DMA_USBOTG11_PMS_R_REG	USB 2.0 全速 OTG 读权限控制寄存器	0x01C4	R/W
PMS_DMA_USBOTG11_PMS_W_REG	USB 2.0 全速 OTG 写权限控制寄存器	0x01C8	R/W
PMS_DMA_TRACE0_PMS_R_REG	TRACE0 读权限控制寄存器	0x01CC	R/W
PMS_DMA_TRACE0_PMS_W_REG	TRACE0 写权限控制寄存器	0x01D0	R/W
PMS_DMA_TRACE1_PMS_R_REG	TRACE1 读权限控制寄存器	0x01D4	R/W
PMS_DMA_TRACE1_PMS_W_REG	TRACE1 写权限控制寄存器	0x01D8	R/W
PMS_DMA_L2MEM_MON_PMS_W_REG	L2MEM 监测器写权限控制寄存器	0x01E0	R/W

名称	描述	地址	访问
PMS_DMA_SPM_MON_PMS_W_REG	SPM 监测器写权限控制寄存器	0x01E8	R/W
PMS_DMA_H264_PMS_R_REG	H264 DMA 读权限控制寄存器	0x01FC	R/W
PMS_DMA_H264_PMS_W_REG	H264 DMA 写权限控制寄存器	0x0200	R/W
PMS_DMA_DMA2D_PPA_PMS_R_REG	2D-DMA PPA 读权限控制寄存器	0x0204	R/W
PMS_DMA_DMA2D_PPA_PMS_W_REG	2D-DMA PPA 写权限控制寄存器	0x0208	R/W
PMS_DMA_DMA2D_DUMMY_PMS_R_REG	2D-DMA Dummy 读权限控制寄存器	0x020C	R/W
PMS_DMA_DMA2D_DUMMY_PMS_W_REG	2D-DMA Dummy 写权限控制寄存器	0x0210	R/W
PMS_DMA_AHB_PDMA_DUMMY_R_PMS_REG	GDMA-AHB Dummy 读权限控制寄存器	0x0214	R/W
PMS_DMA_AHB_PDMA_DUMMY_W_PMS_REG	GDMA-AHB Dummy 写权限控制寄存器	0x0218	R/W
PMS_DMA_AXI_PDMA_DUMMY_R_PMS_REG	GDMA-AXI Dummy 读权限控制寄存器	0x021C	R/W
PMS_DMA_AXI_PDMA_DUMMY_W_PMS_REG	GDMA-AXI Dummy 写权限控制寄存器	0x0220	R/W

20.6.2 HP_PERI_PMS_REG

本小节的所有地址均为相对于 HP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
版本控制寄存器			
PMS_HP_PERI_PMS_DATE_REG	版本控制寄存器	0x0000	R/W
时钟门控寄存器			
PMS_HP_PERI_PMS_CLK_EN_REG	时钟门控寄存器	0x0004	R/W
HP CPU 权限管理寄存器			
PMS_COREO_MM_HP_PERI_PMS_REGO_REG	HP CPU0 在机器模式下的权限控制寄存器 0	0x0008	R/W
PMS_COREO_MM_HP_PERI_PMS_REG1_REG	HP CPU0 在机器模式下的权限控制寄存器 1	0x000C	R/W
PMS_COREO_MM_HP_PERI_PMS_REG2_REG	HP CPU0 在机器模式下的权限控制寄存器 2	0x0010	R/W
PMS_COREO_MM_HP_PERI_PMS_REG3_REG	HP CPU0 在机器模式下的权限控制寄存器 3	0x0014	R/W
PMS_COREO_UM_HP_PERI_PMS_REGO_REG	HP CPU0 在用户模式下的权限控制寄存器 0	0x0018	R/W
PMS_COREO_UM_HP_PERI_PMS_REG1_REG	HP CPU0 在用户模式下的权限控制寄存器 1	0x001C	R/W
PMS_COREO_UM_HP_PERI_PMS_REG2_REG	HP CPU0 在用户模式下的权限控制寄存器 2	0x0020	R/W
PMS_COREO_UM_HP_PERI_PMS_REG3_REG	HP CPU0 在用户模式下的权限控制寄存器 3	0x0024	R/W
PMS_CORE1_MM_HP_PERI_PMS_REGO_REG	HP CPU1 在机器模式下的权限控制寄存器 0	0x0028	R/W

名称	描述	地址	访问
PMS_CORE1_MM_HP_PERI_PMS_REG1_REG	HP CPU1 在机器模式下的权限控制寄存器 1	0x002C	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG2_REG	HP CPU1 在机器模式下的权限控制寄存器 2	0x0030	R/W
PMS_CORE1_MM_HP_PERI_PMS_REG3_REG	HP CPU1 在机器模式下的权限控制寄存器 3	0x0034	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG0_REG	HP CPU1 在用户模式下的权限控制寄存器 0	0x0038	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG1_REG	HP CPU1 在用户模式下的权限控制寄存器 1	0x003C	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG2_REG	HP CPU1 在用户模式下的权限控制寄存器 2	0x0040	R/W
PMS_CORE1_UM_HP_PERI_PMS_REG3_REG	HP CPU1 在用户模式下的权限控制寄存器 3	0x0044	R/W

20.6.3 LP2HP_PERI_PMS_REG

本小节的所有地址均为相对于 LP2HP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
版本控制寄存器			
PMS_LP2HP_PERI_PMS_DATE_REG	版本控制寄存器	0x0000	R/W
时钟门控寄存器			
PMS_LP2HP_PERI_PMS_CLK_EN_REG	时钟门控寄存器	0x0004	R/W
LP CPU 权限管理寄存器			
PMS_LP_MM_PMS_REG0_REG	LP CPU 在机器模式下的权限控制寄存器 0	0x0008	R/W
PMS_LP_MM_PMS_REG1_REG	LP CPU 在机器模式下的权限控制寄存器 1	0x0030	R/W
PMS_LP_MM_PMS_REG2_REG	LP CPU 在机器模式下的权限控制寄存器 2	0x00A4	R/W
PMS_LP_MM_PMS_REG3_REG	LP CPU 在机器模式下的权限控制寄存器 3	0x011C	R/W

20.6.4 LP_PERI_PMS_REG

本小节的所有地址均为相对于 LP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
版本控制寄存器			
PMS_LP_PERI_PMS_DATE_REG	版本控制寄存器	0x0000	R/W
时钟门控寄存器			
PMS_LP_PERI_PMS_CLK_EN_REG	时钟门控寄存器	0x0004	R/W

名称	描述	地址	访问
LP CPU 权限管理寄存器			
PMS_LP_MM_LP_PERI_PMS_REG0_REG	LP CPU 在机器模式下的权限控制寄存器 0	0x0008	R/W
可配置地址区间配置寄存器			
PMS_PERI_REGION0_LOW_REG	地址区间 0 起始地址配置寄存器	0x000C	R/W
PMS_PERI_REGION0_HIGH_REG	地址区间 0 结束地址配置寄存器	0x0010	R/W
PMS_PERI_REGION1_LOW_REG	地址区间 1 起始地址配置寄存器	0x0014	R/W
PMS_PERI_REGION_PMS_REG	地址区间 0-1 访问权限配置寄存器	0x001C	R/W
PMS_PERI_REGION_2_TO_7_PMS_REG	地址区间 2-7 访问权限配置寄存器	0x0020	R/W
PMS_PERI_REGION2_LOW_REG	地址区间 2 起始地址配置寄存器	0x0024	R/W
PMS_PERI_REGION2_HIGH_REG	地址区间 2 结束地址配置寄存器	0x0028	R/W
PMS_PERI_REGION3_LOW_REG	地址区间 3 起始地址配置寄存器	0x002C	R/W
PMS_PERI_REGION3_HIGH_REG	地址区间 3 结束地址配置寄存器	0x0030	R/W
PMS_PERI_REGION4_LOW_REG	地址区间 4 起始地址配置寄存器	0x0034	R/W
PMS_PERI_REGION4_HIGH_REG	地址区间 4 结束地址配置寄存器	0x0038	R/W
PMS_PERI_REGION5_LOW_REG	地址区间 5 起始地址配置寄存器	0x003C	R/W
PMS_PERI_REGION5_HIGH_REG	地址区间 5 结束地址配置寄存器	0x0040	R/W
PMS_PERI_REGION6_LOW_REG	地址区间 6 起始地址配置寄存器	0x0044	R/W
PMS_PERI_REGION6_HIGH_REG	地址区间 6 结束地址配置寄存器	0x0048	R/W
PMS_PERI_REGION7_LOW_REG	地址区间 7 起始地址配置寄存器	0x004C	R/W
PMS_PERI_REGION7_HIGH_REG	地址区间 7 结束地址配置寄存器	0x0050	R/W
PMS_PERI_REGION1_HIGH_REG	地址区间 1 结束地址配置寄存器	0x0018	R/W

20.6.5 HP2LP_PERI_PMS_REG

本小节的所有地址均为相对于 HP2LP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
版本控制寄存器			
PMS_HP2LP_PERI_PMS_DATE_REG	版本控制寄存器	0x0000	R/W
时钟门控寄存器			
PMS_HP2LP_PERI_PMS_CLK_EN_REG	时钟门控寄存器	0x0004	R/W
HP CPU 权限管理寄存器			
PMS_HP_CORE0_MM_PMS_REG0_REG	HP CPU0 在机器模式下的权限控制寄存器 0	0x0008	R/W
PMS_HP_CORE0_UM_PMS_REG0_REG	HP CPU0 在用户模式下的权限控制寄存器 0	0x000C	R/W
PMS_HP_CORE1_MM_PMS_REG0_REG	HP CPU1 在机器模式下的权限控制寄存器 0	0x0010	R/W
PMS_HP_CORE1_UM_PMS_REG0_REG	HP CPU1 在用户模式下的权限控制寄存器 0	0x0014	R/W

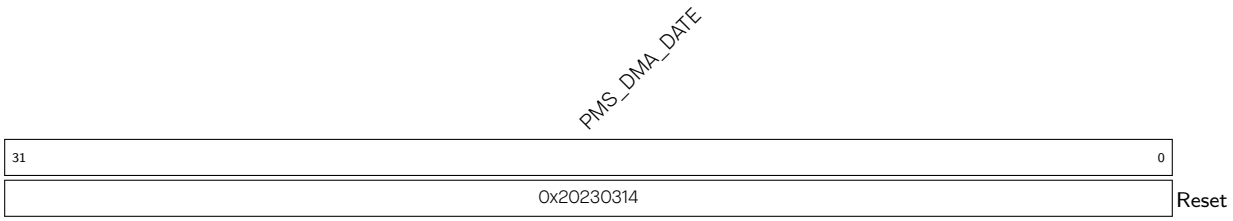
20.7 寄存器

20.7.1 HP_DMA_PMS_REG

本小节的所有地址均为相对于 HP_DMA_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

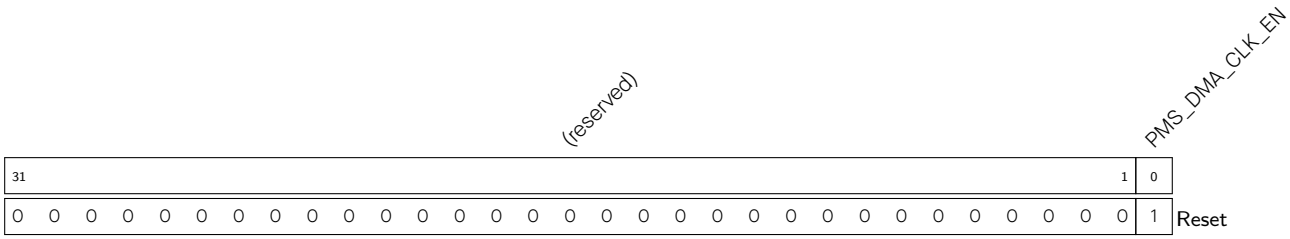
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 20.1. PMS_DMA_DATE_REG (0x0000)



PMS_DMA_DATE 版本控制寄存器。(R/W)

Register 20.2. PMS_DMA_CLK_EN_REG (0x0004)



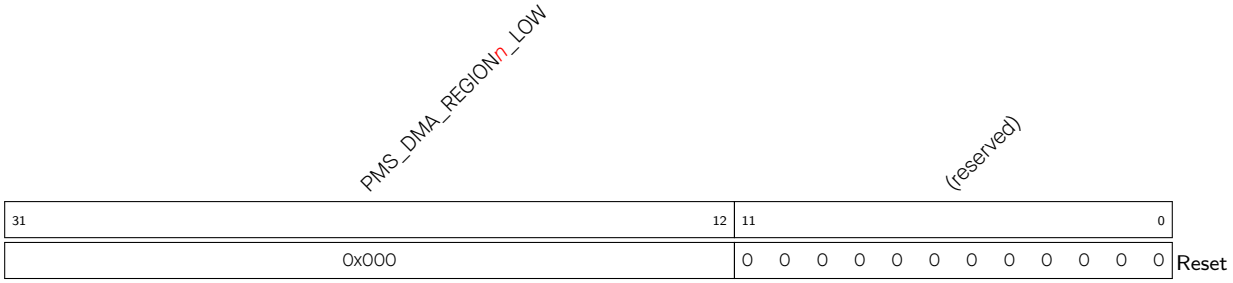
PMS_DMA_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

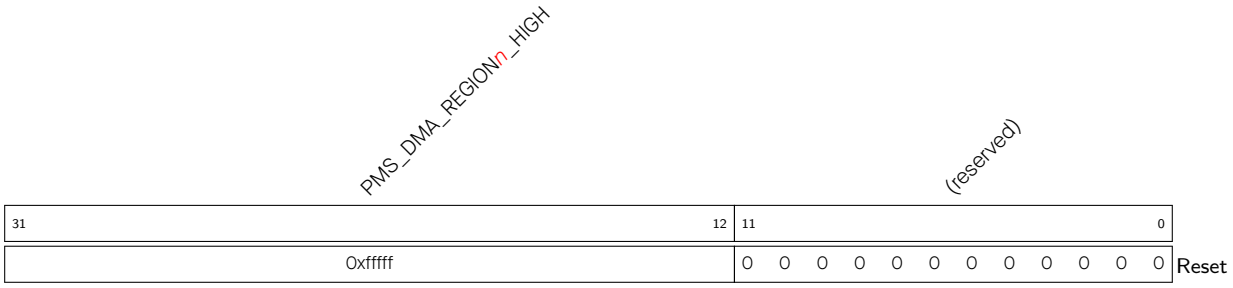
(R/W)

Register 20.3. PMS_DMA_REGION*n*_LOW_REG (*n*: 0-31) (0x0008+0x8**n*)



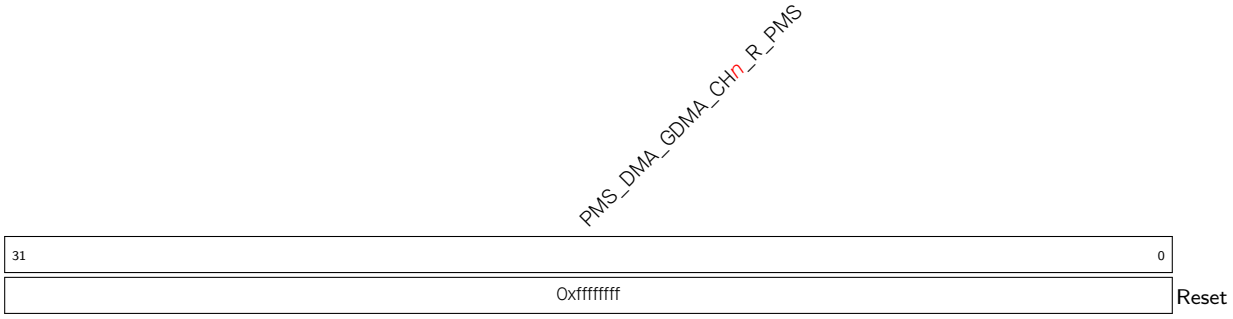
PMS_DMA_REGION*n*_LOW 配置地址区间 *n* 起始地址的高 20 位。(R/W)

Register 20.4. PMS_DMA_REGIONn_HIGH_REG (n: 0-31) (0x000C+0x8*n)



PMS_DMA_REGIONn_HIGH 配置地址区间 n 结束地址的高 20 位。(R/W)

Register 20.5. PMS_DMA_GDMA_CHn_R_PMS_REG (n: 0-3) (0x0108+0x8*n)



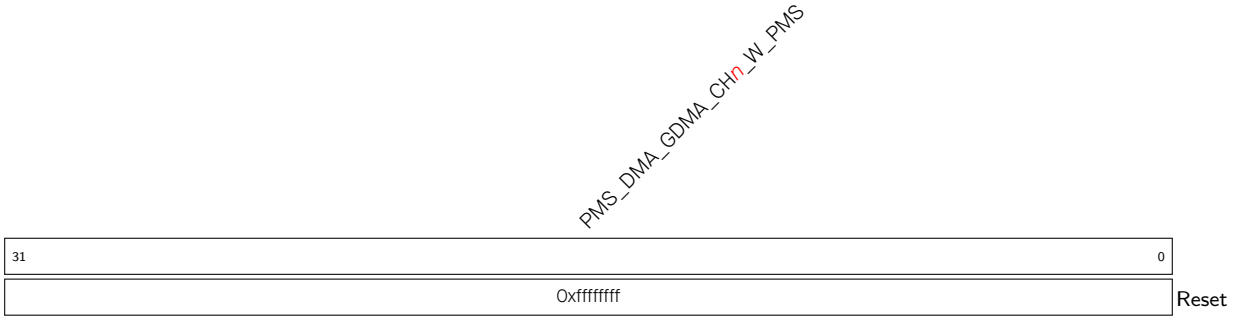
PMS_DMA_GDMA_CHn_R_PMS 配置 GDMA chn 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.6. PMS_DMA_GDMA_CHn_W_PMS_REG (n: 0-3) (0x010C+0x8*n)



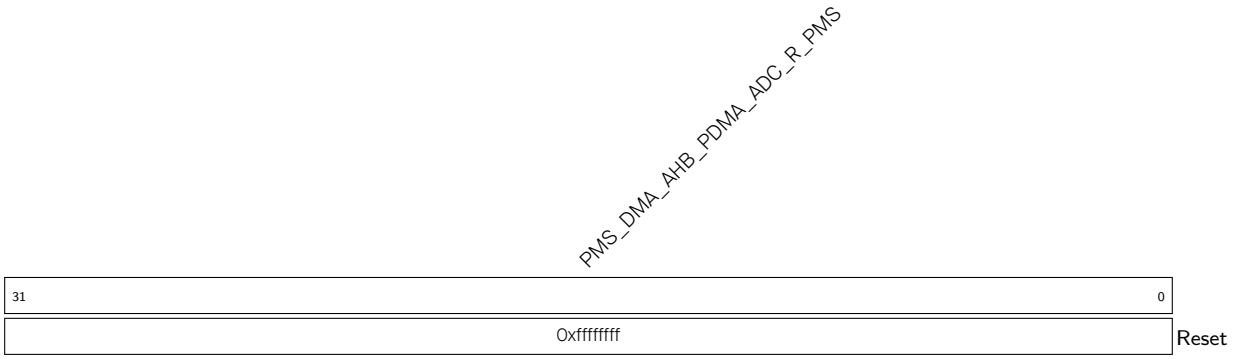
PMS_DMA_GDMA_CHn_W_PMS 配置 GDMA chn 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.7. PMS_DMA_AHB_PDMA_ADC_R_PMS_REG (0x0128)



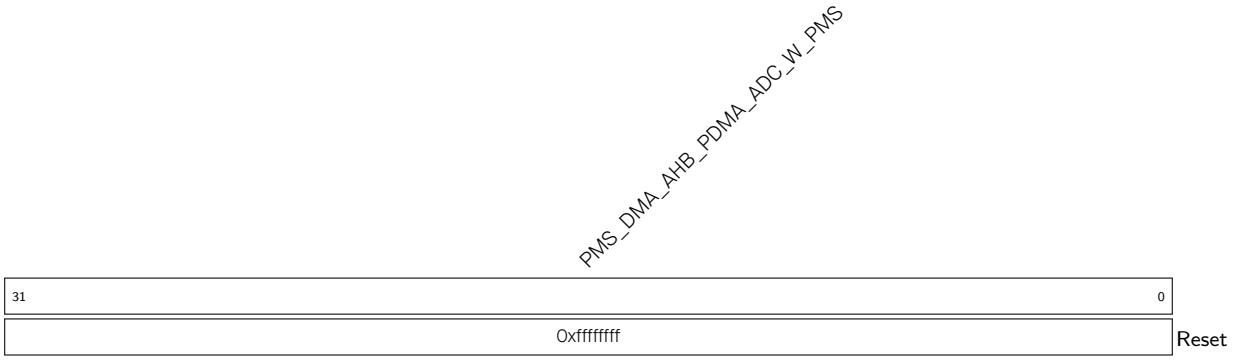
PMS_DMA_AHB_PDMA_ADC_R_PMS 配置 GDMA-AHB 应 ADC 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.8. PMS_DMA_AHB_PDMA_ADC_W_PMS_REG (0x012C)



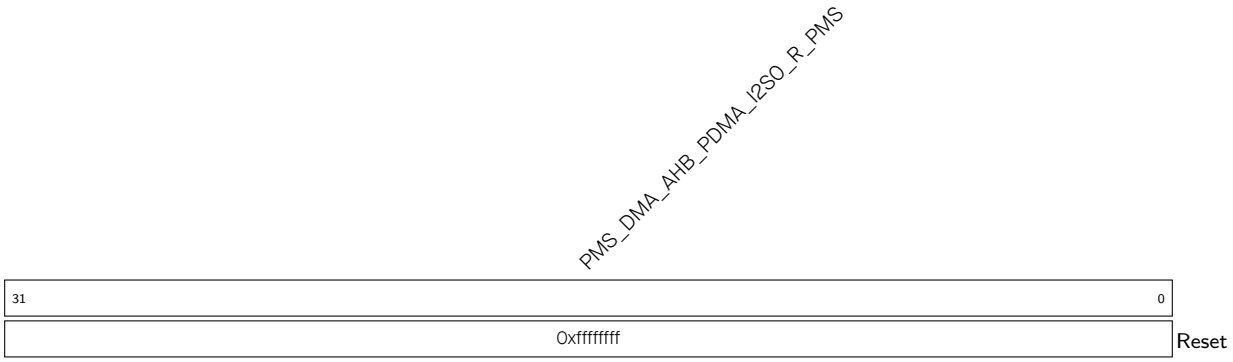
PMS_DMA_AHB_PDMA_ADC_W_PMS 配置 GDMA-AHB 应 ADC 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.9. PMS_DMA_AHB_PDMA_I2S0_R_PMS_REG (0x0130)



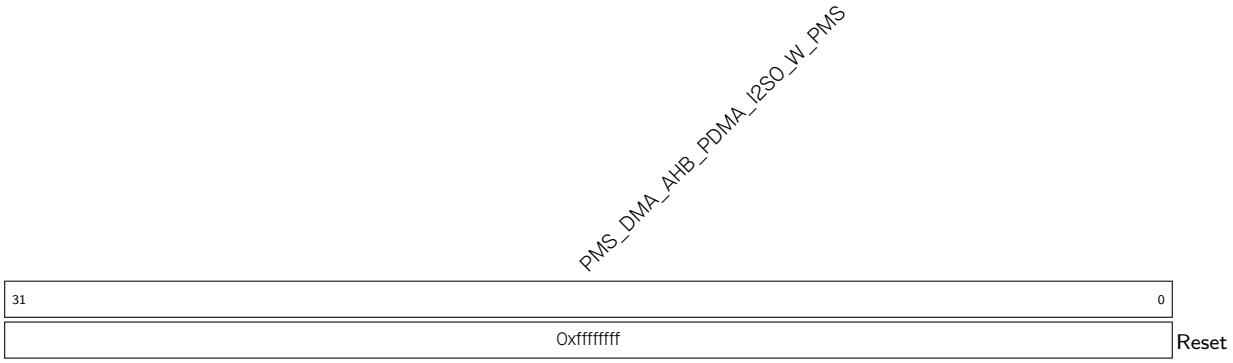
PMS_DMA_AHB_PDMA_I2S0_R_PMS 配置 GDMA-AHB 应 I2S0 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.10. PMS_DMA_AHB_PDMA_I2S0_W_PMS_REG (0x0134)



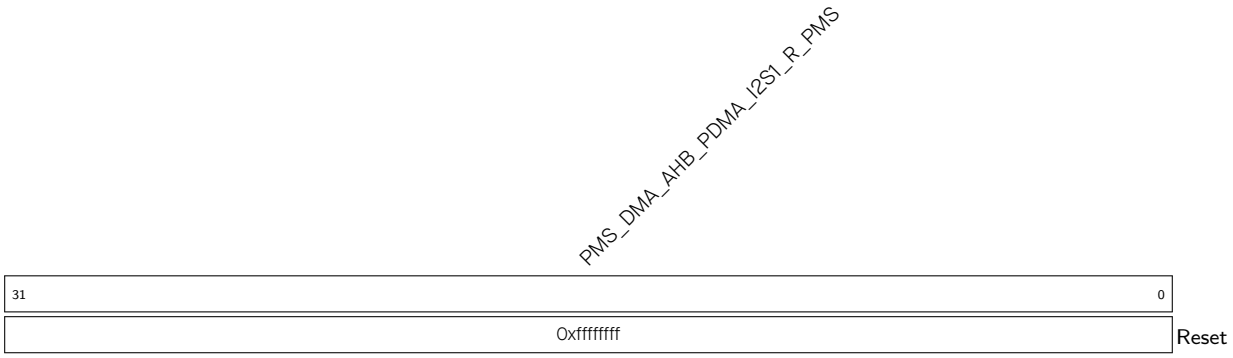
PMS_DMA_AHB_PDMA_I2S0_W_PMS 配置 GDMA-AHB 应 I2S0 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.11. PMS_DMA_AHB_PDMA_I2S1_R_PMS_REG (0x0138)



PMS_DMA_AHB_PDMA_I2S1_R_PMS 配置 GDMA-AHB 应 I2S1 的请求读 32 个地址区间的权限。

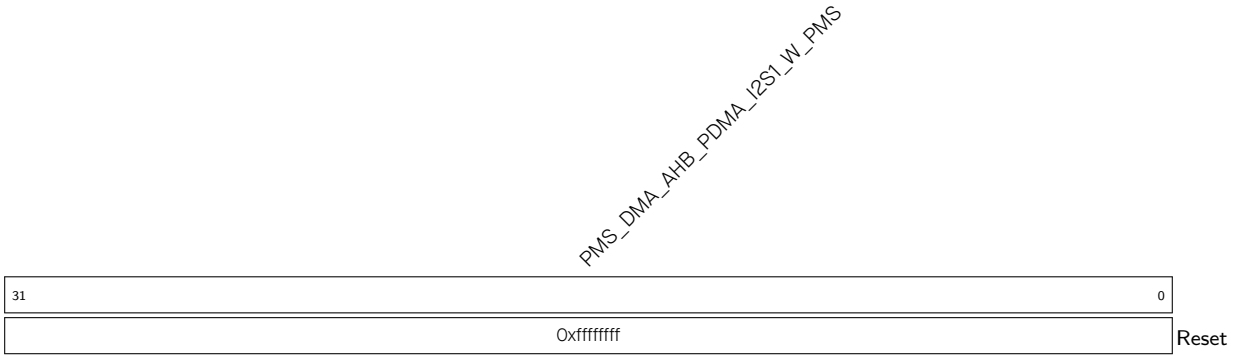
bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.12. PMS_DMA_AHB_PDMA_I2S1_W_PMS_REG (0x013C)



PMS_DMA_AHB_PDMA_I2S1_W_PMS 配置 GDMA-AHB 应 I2S1 的请求写 32 个地址区间的权限。

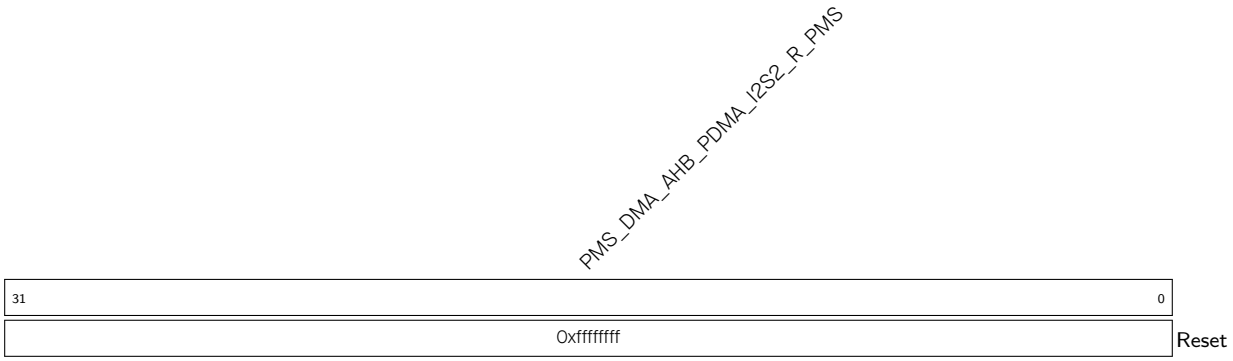
bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.13. PMS_DMA_AHB_PDMA_I2S2_R_PMS_REG (0x0140)



PMS_DMA_AHB_PDMA_I2S2_R_PMS 配置 GDMA-AHB 应 I2S2 的请求读 32 个地址区间的权限。

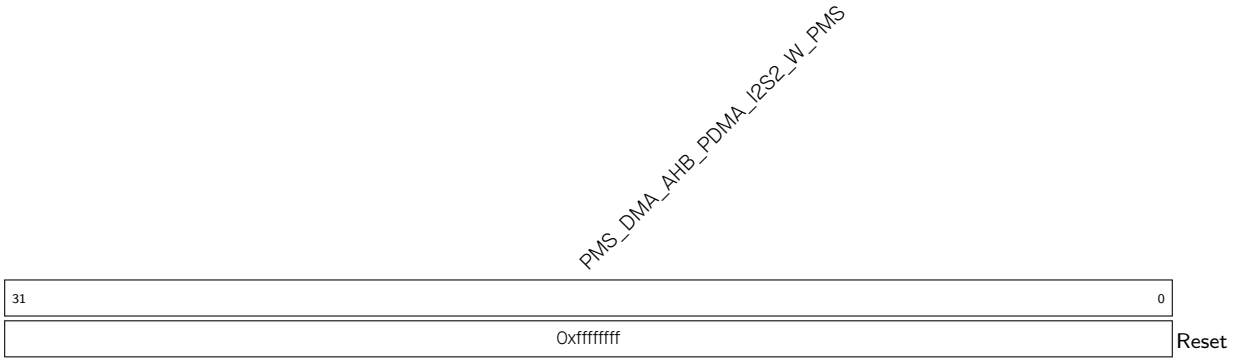
bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.14. PMS_DMA_AHB_PDMA_I2S2_W_PMS_REG (0x0144)



PMS_DMA_AHB_PDMA_I2S2_W_PMS 配置 GDMA-AHB 应 I2S2 的请求写 32 个地址区间的权限。

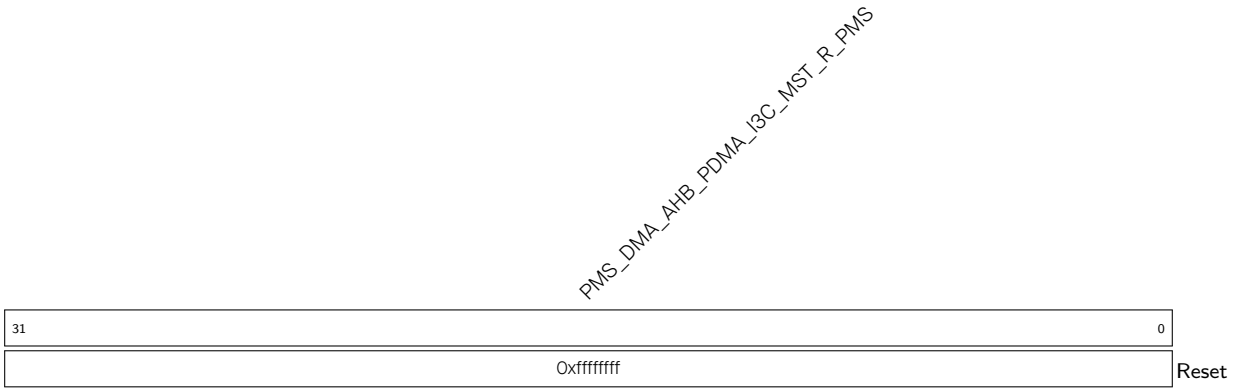
bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.15. PMS_DMA_AHB_PDMA_I3C_MST_R_PMS_REG (0x0148)



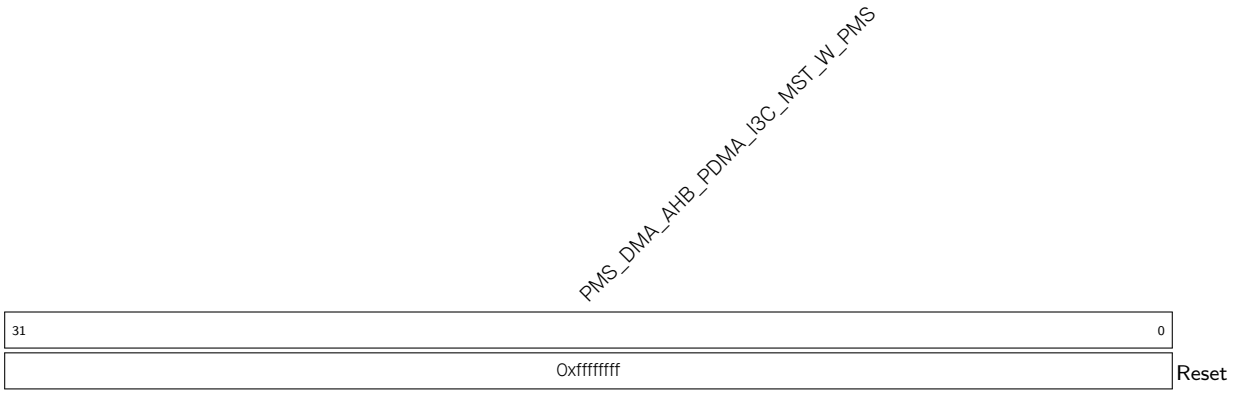
PMS_DMA_AHB_PDMA_I3C_MST_R_PMS 配置 GDMA-AHB 应 I3C 主机的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.16. PMS_DMA_AHB_PDMA_I3C_MST_W_PMS_REG (0x014C)



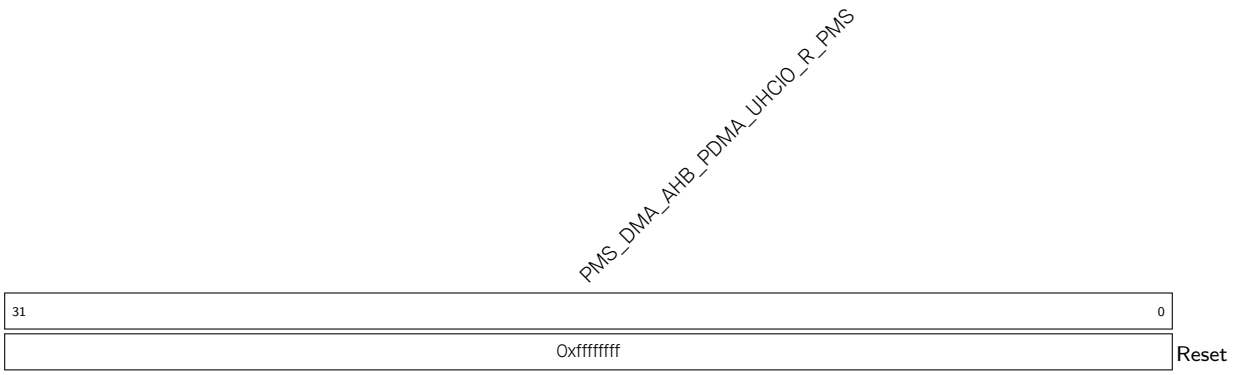
PMS_DMA_AHB_PDMA_I3C_MST_W_PMS 配置 GDMA-AHB 应 I3C 主机的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.17. PMS_DMA_AHB_PDMA_UHCI0_R_PMS_REG (0x0150)



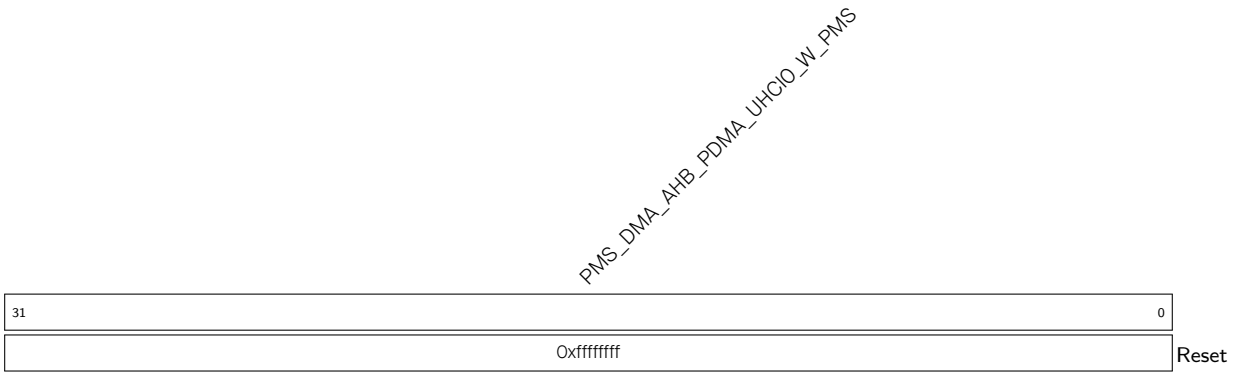
PMS_DMA_AHB_PDMA_UHCI0_R_PMS 配置 GDMA-AHB 应 UHCI 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.18. PMS_DMA_AHB_PDMA_UHCI0_W_PMS_REG (0x0154)



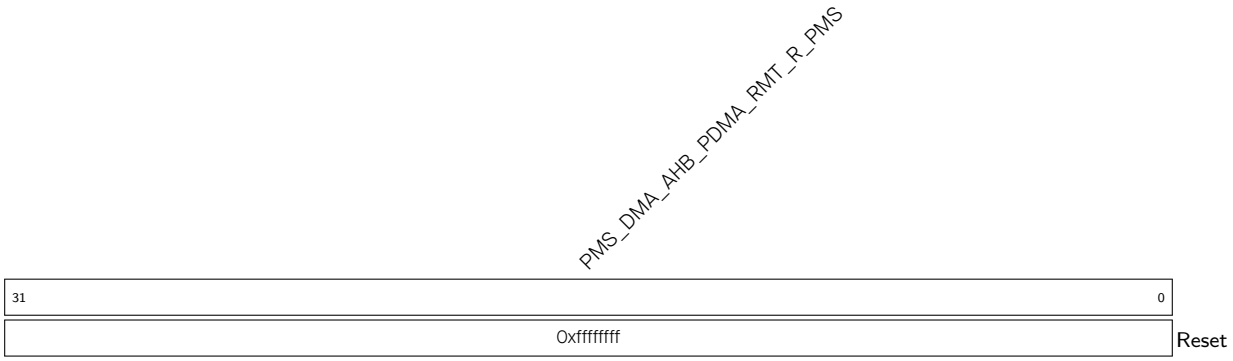
PMS_DMA_AHB_PDMA_UHCI0_W_PMS 配置 GDMA-AHB 应 UHCI 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.19. PMS_DMA_AHB_PDMA_RMT_R_PMS_REG (0x0158)



PMS_DMA_AHB_PDMA_RMT_R_PMS 配置 GDMA-AHB 应 RMT 的请求读 32 个地址区间的权限。

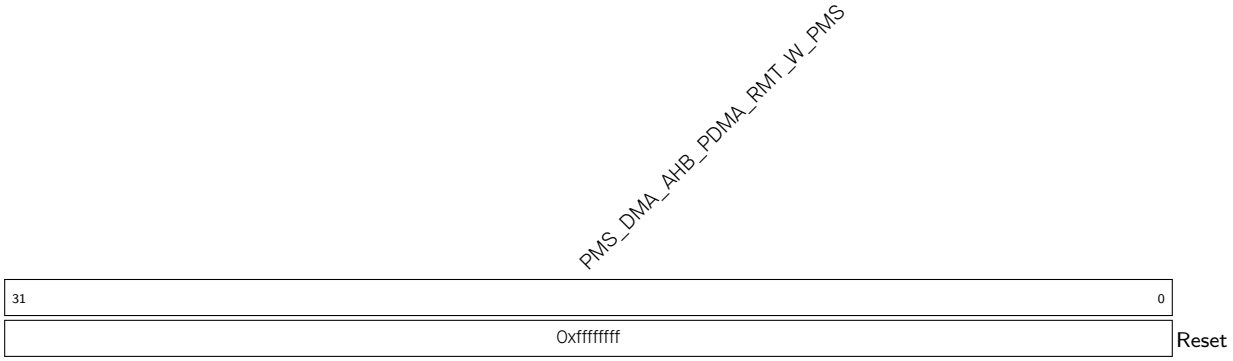
bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.20. PMS_DMA_AHB_PDMA_RMT_W_PMS_REG (0x0170)



PMS_DMA_AHB_PDMA_RMT_W_PMS 配置 GDMA-AHB 应 RMT 的请求写 32 个地址区间的权限。

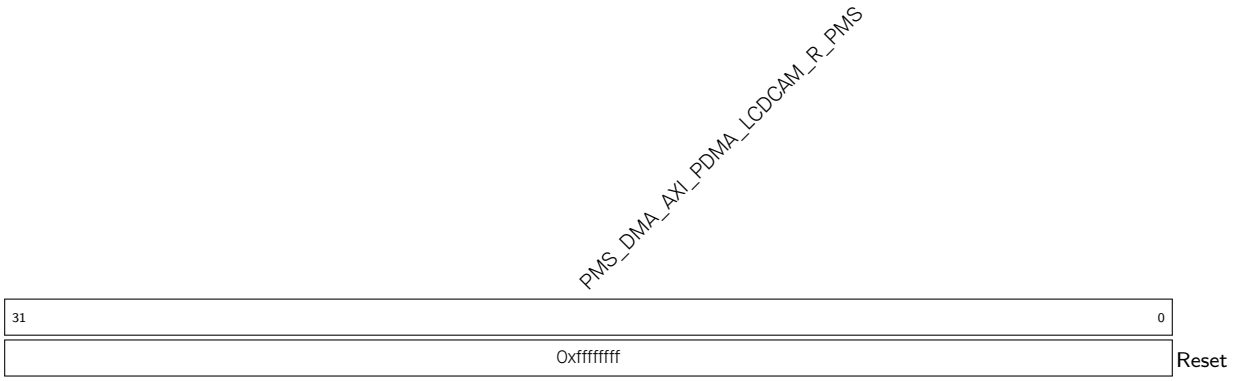
bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.21. PMS_DMA_AXI_PDMA_LCDCAM_R_PMS_REG (0x0174)



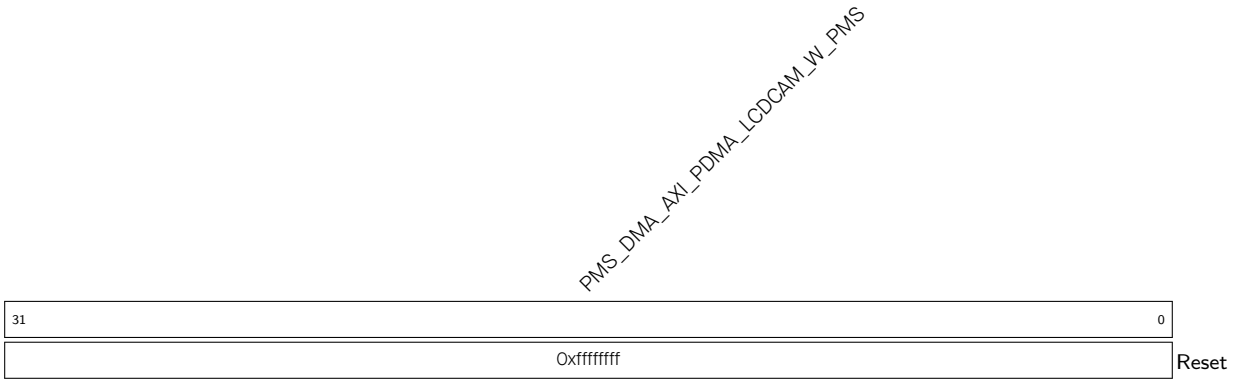
PMS_DMA_AXI_PDMA_LCDCAM_R_PMS 配置 GDMA-AXI 应 LCD_CAM 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.22. PMS_DMA_AXI_PDMA_LCDCAM_W_PMS_REG (0x0178)



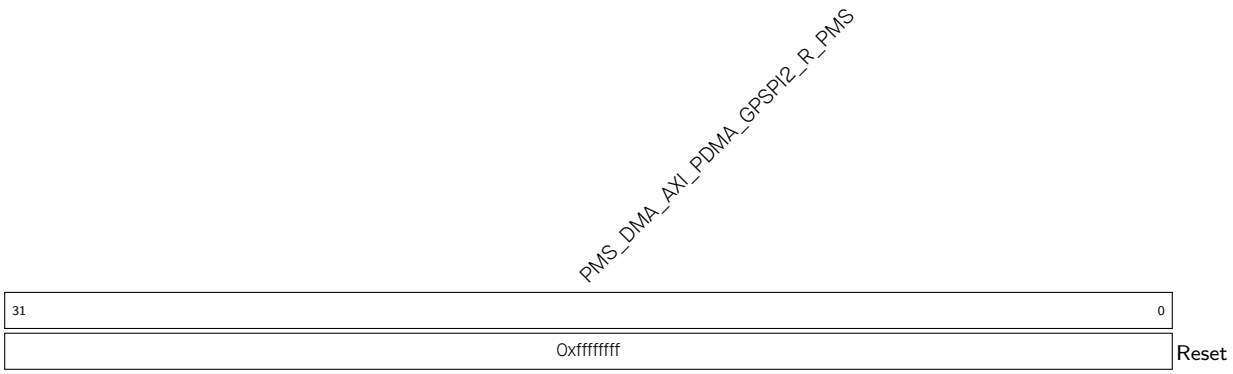
PMS_DMA_AXI_PDMA_LCDCAM_W_PMS 配置 GDMA-AXI 应 LCD_CAM 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.23. PMS_DMA_AXI_PDMA_GPSPI2_R_PMS_REG (0x017C)



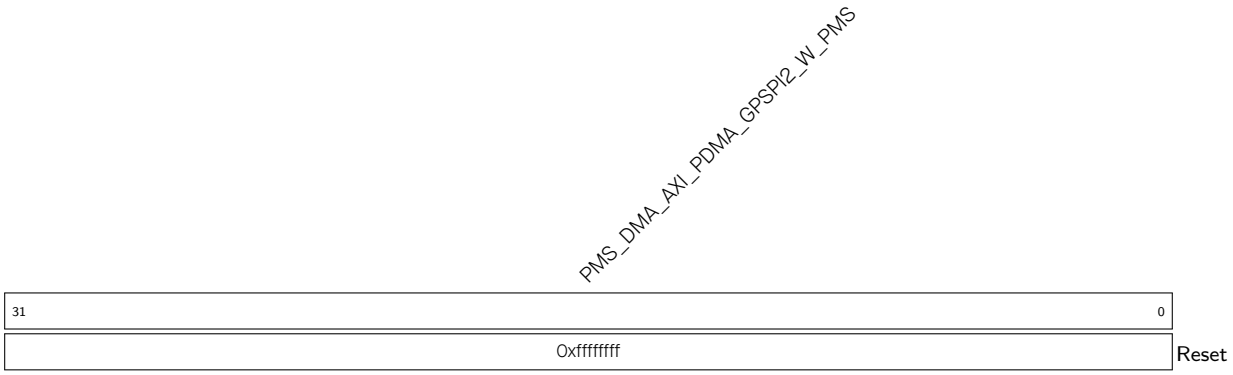
PMS_DMA_AXI_PDMA_GPSPI2_R_PMS 配置 GDMA-AXI 应 GP-SPI2 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.24. PMS_DMA_AXI_PDMA_GPSPI2_W_PMS_REG (0x0180)



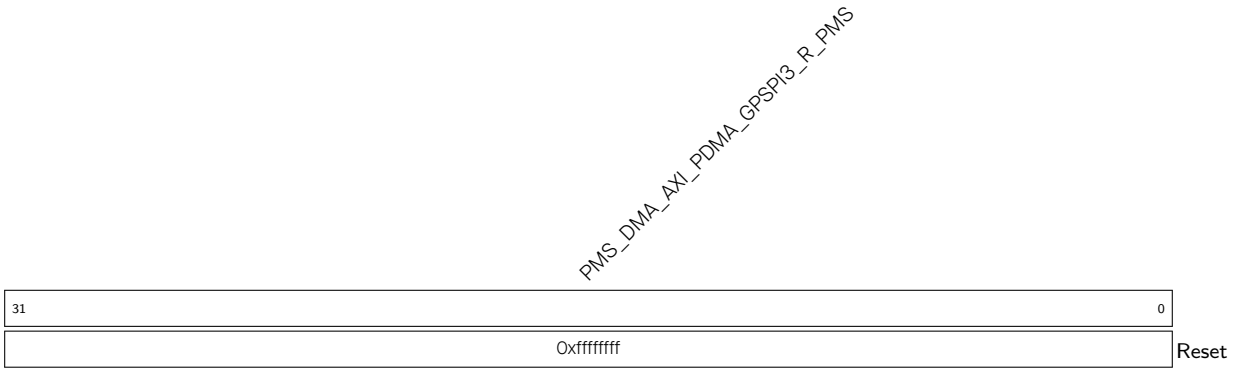
PMS_DMA_AXI_PDMA_GPSPI2_W_PMS 配置 GDMA-AXI 应 GP-SPI2 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.25. PMS_DMA_AXI_PDMA_GPSPI3_R_PMS_REG (0x0184)



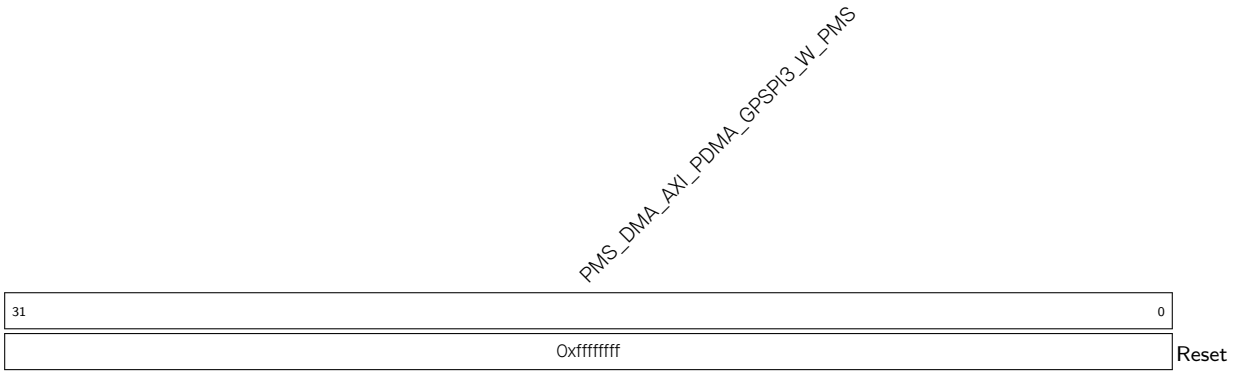
PMS_DMA_AXI_PDMA_GPSPI3_R_PMS 配置 GDMA-AXI 应 GP-SPI3 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.26. PMS_DMA_AXI_PDMA_GPSPI3_W_PMS_REG (0x0188)



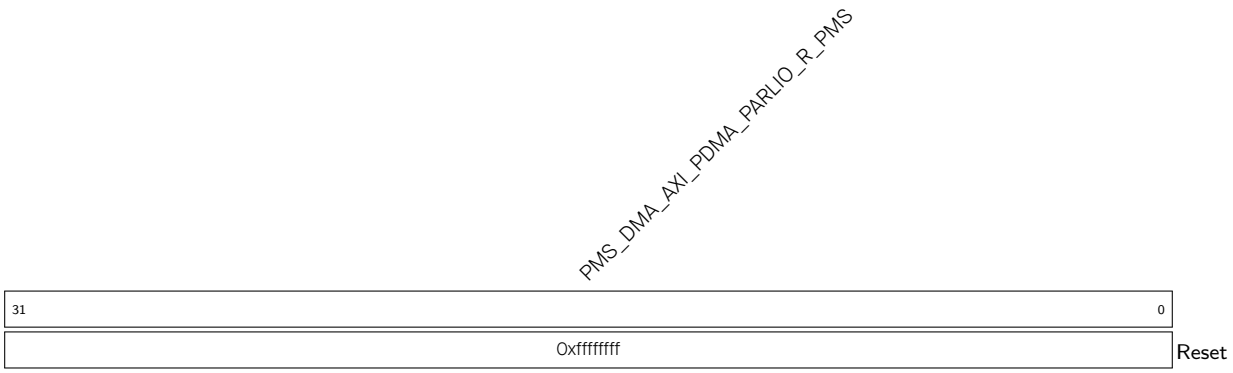
PMS_DMA_AXI_PDMA_GPSPI3_W_PMS 配置 GDMA-AXI 应 GP-SPI3 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.27. PMS_DMA_AXI_PDMA_PARLIO_R_PMS_REG (0x018C)



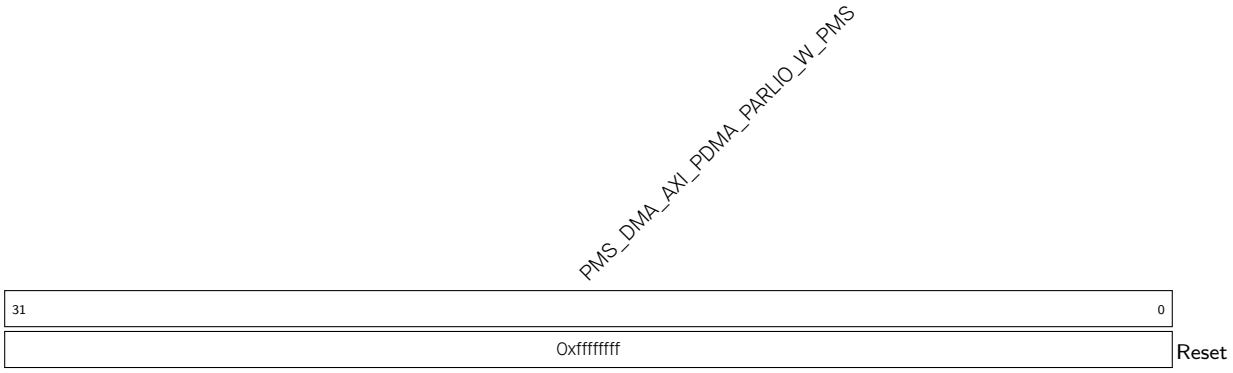
PMS_DMA_AXI_PDMA_PARLIO_R_PMS 配置 GDMA-AXI 应 PARLIO（并行 IO 控制器）的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.28. PMS_DMA_AXI_PDMA_PARLIO_W_PMS_REG (0x0190)



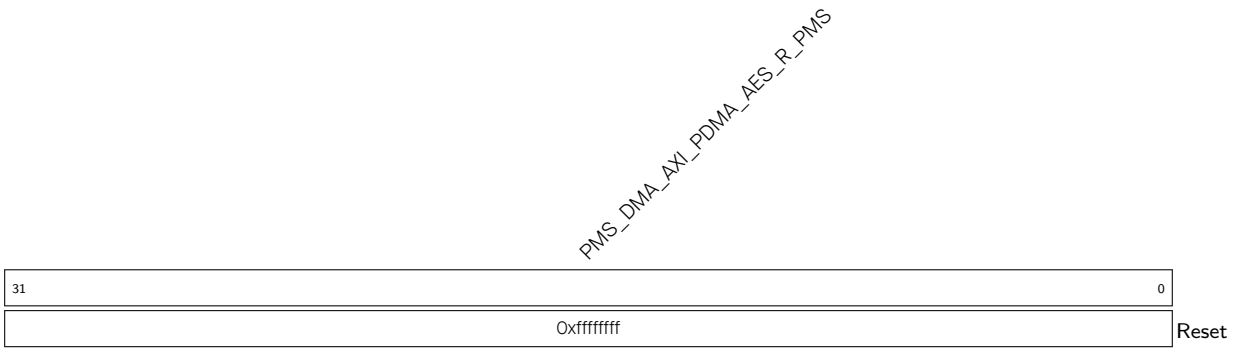
PMS_DMA_AXI_PDMA_PARLIO_W_PMS 配置 GDMA-AXI 应 PARLIO 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.29. PMS_DMA_AXI_PDMA_AES_R_PMS_REG (0x0194)



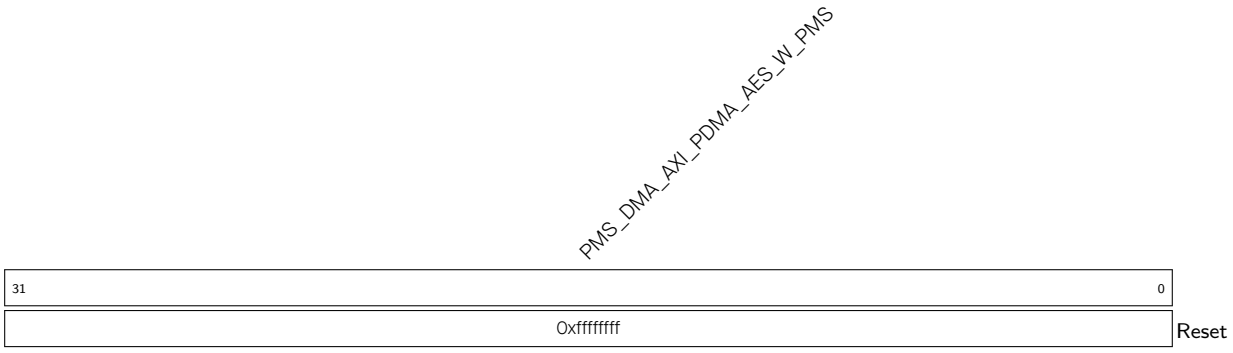
PMS_DMA_AXI_PDMA_AES_R_PMS 配置 GDMA-AXI 应 AES 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.30. PMS_DMA_AXI_PDMA_AES_W_PMS_REG (0x0198)



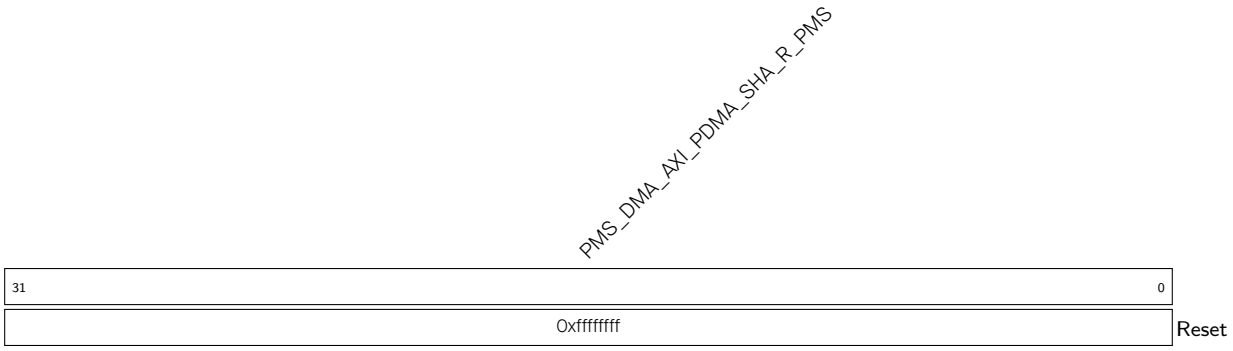
PMS_DMA_AXI_PDMA_AES_W_PMS 配置 GDMA-AXI 应 AES 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.31. PMS_DMA_AXI_PDMA_SHA_R_PMS_REG (0x019C)



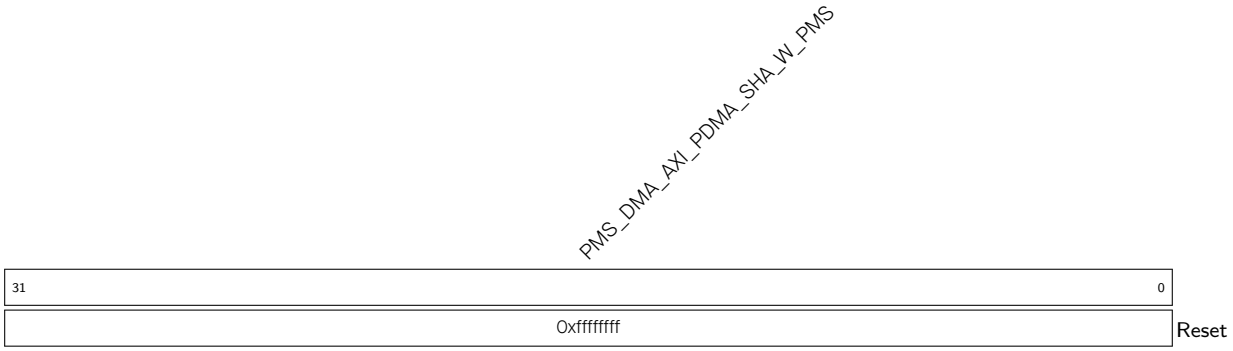
PMS_DMA_AXI_PDMA_SHA_R_PMS 配置 GDMA-AXI 应 SHA 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.32. PMS_DMA_AXI_PDMA_SHA_W_PMS_REG (0x01A0)



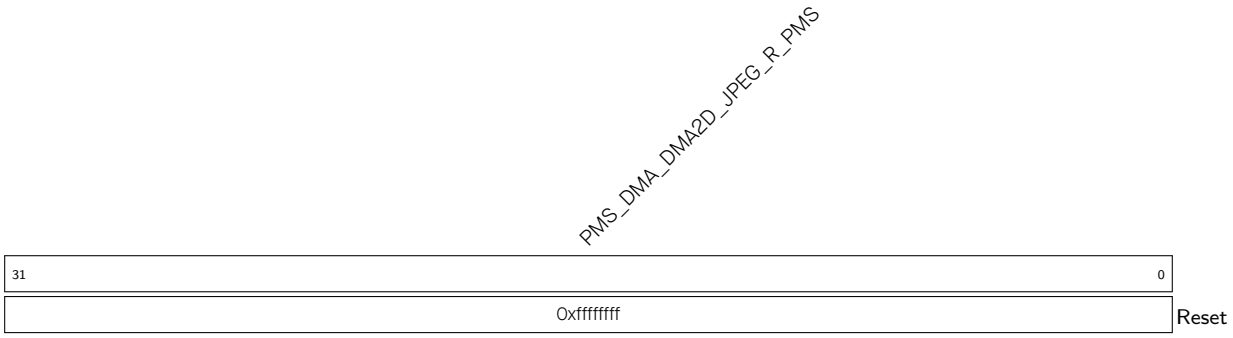
PMS_DMA_AXI_PDMA_SHA_W_PMS 配置 GDMA-AXI 应 SHA 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.33. PMS_DMA_DMA2D_JPEG_PMS_R_REG (0x01A4)



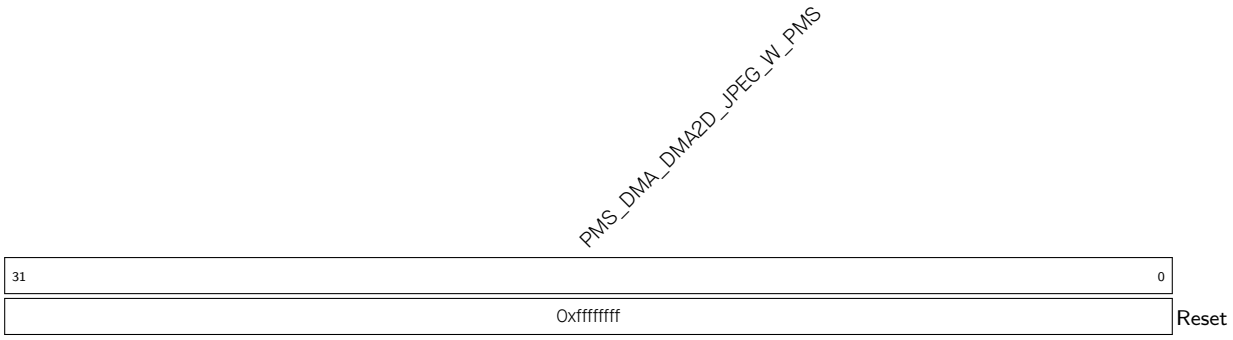
PMS_DMA_DMA2D_JPEG_R_PMS 配置 2D-DMA 应 JPEG 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.34. PMS_DMA_DMA2D_JPEG_PMS_W_REG (0x01A8)



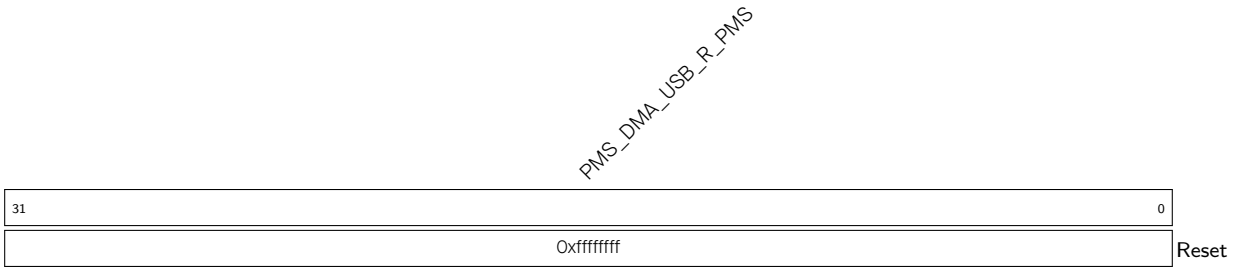
PMS_DMA_DMA2D_JPEG_W_PMS 配置 2D-DMA 应 JPEG 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.35. PMS_DMA_USB_PMS_R_REG (0x01AC)



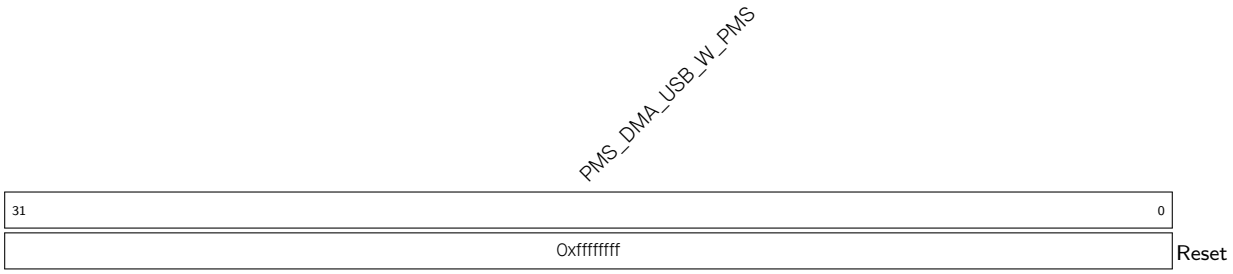
PMS_DMA_USB_R_PMS 配置 USB 2.0 高速 OTG 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.36. PMS_DMA_USB_PMS_W_REG (0x01B0)



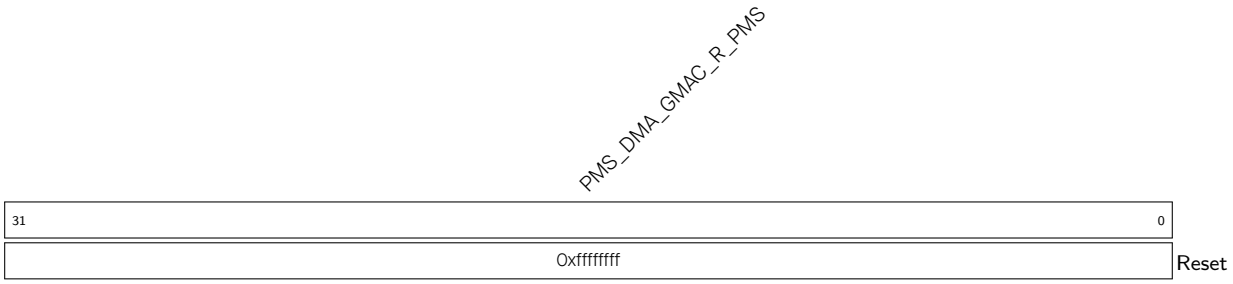
PMS_DMA_USB_W_PMS 配置 USB 2.0 高速 OTG 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.37. PMS_DMA_GMAC_PMS_R_REG (0x01B4)



PMS_DMA_GMAC_R_PMS 配置 EMAC 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.38. PMS_DMA_GMAC_PMS_W_REG (0x01B8)



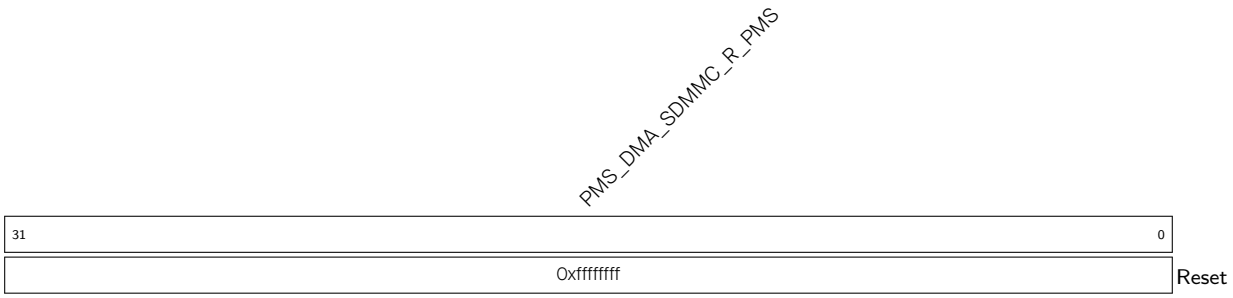
PMS_DMA_GMAC_W_PMS 配置 EMAC 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.39. PMS_DMA_SDMMC_PMS_R_REG (0x01BC)



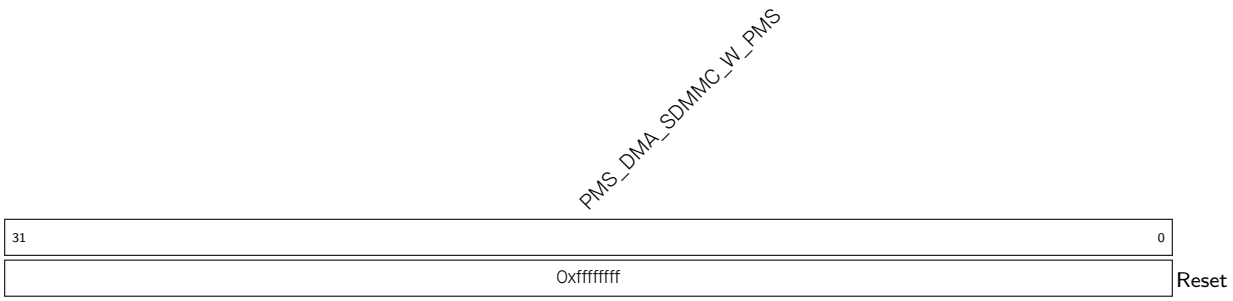
PMS_DMA_SDMMC_R_PMS 配置 SDMMC 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.40. PMS_DMA_SDMMC_PMS_W_REG (0x01C0)



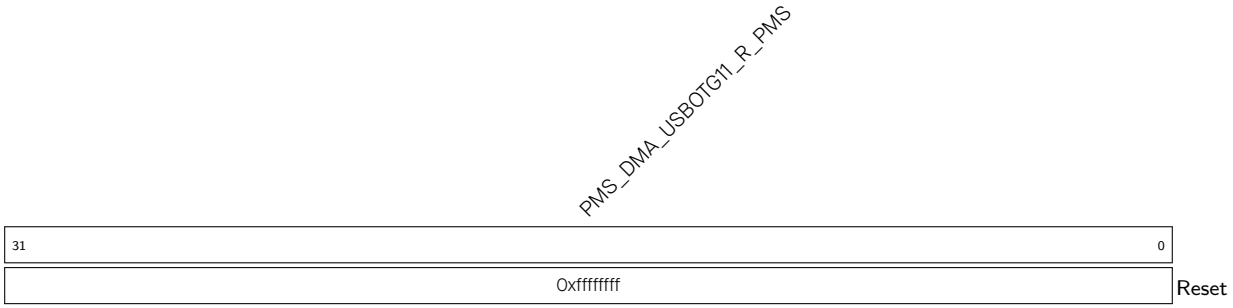
PMS_DMA_SDMMC_W_PMS 配置 SDMMC 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.41. PMS_DMA_USBOTG11_PMS_R_REG (0x01C4)



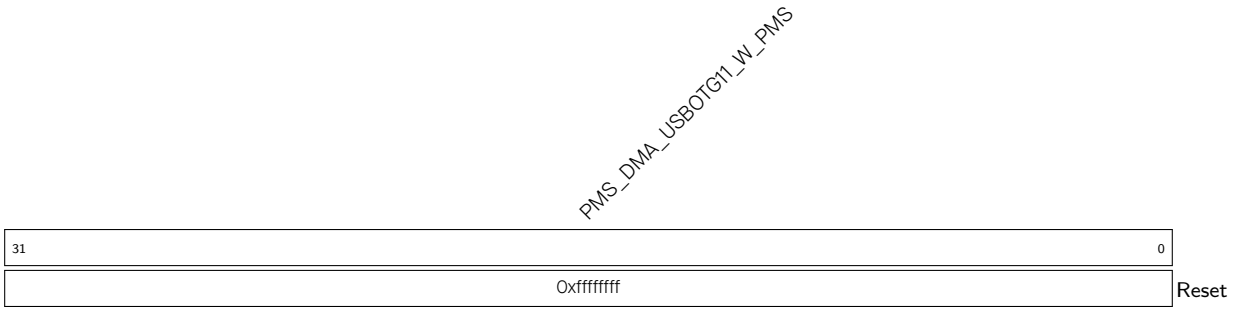
PMS_DMA_USBOTG11_R_PMS 配置 USB 2.0 全速 OTG 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.42. PMS_DMA_USBOTG11_PMS_W_REG (0x01C8)



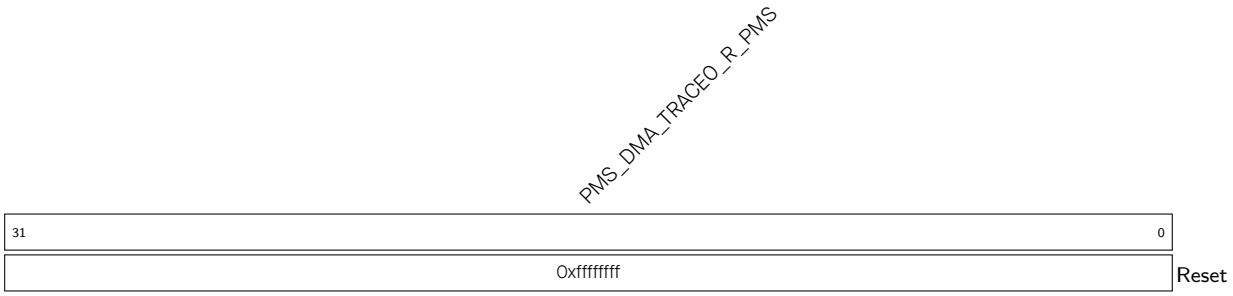
PMS_DMA_USBOTG11_W_PMS 配置 USB 2.0 全速 OTG 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.43. PMS_DMA_TRACE0_PMS_R_REG (0x01CC)



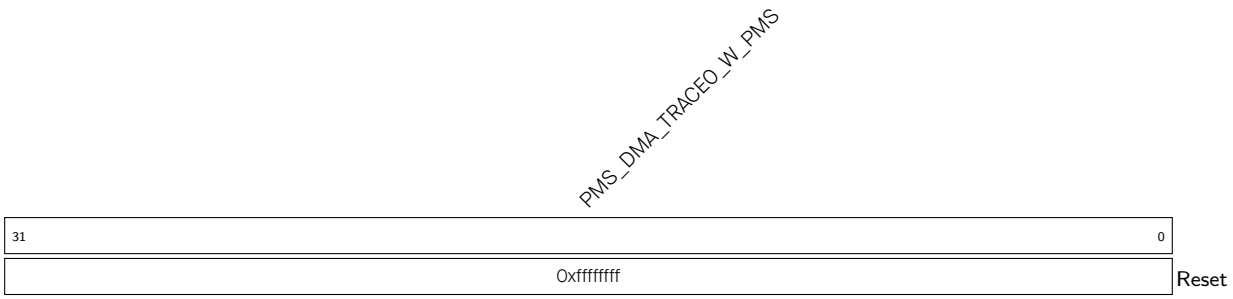
PMS_DMA_TRACE0_R_PMS 配置 TRACE0 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.44. PMS_DMA_TRACE0_PMS_W_REG (0x01D0)



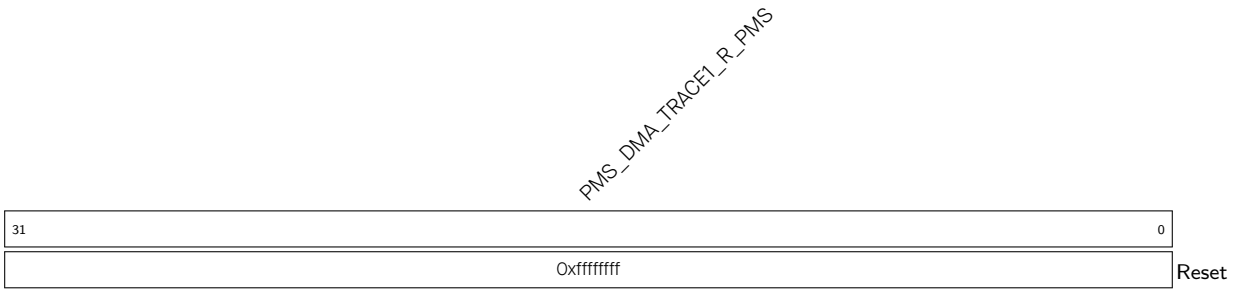
PMS_DMA_TRACE0_W_PMS 配置 TRACE0 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.45. PMS_DMA_TRACE1_PMS_R_REG (0x01D4)



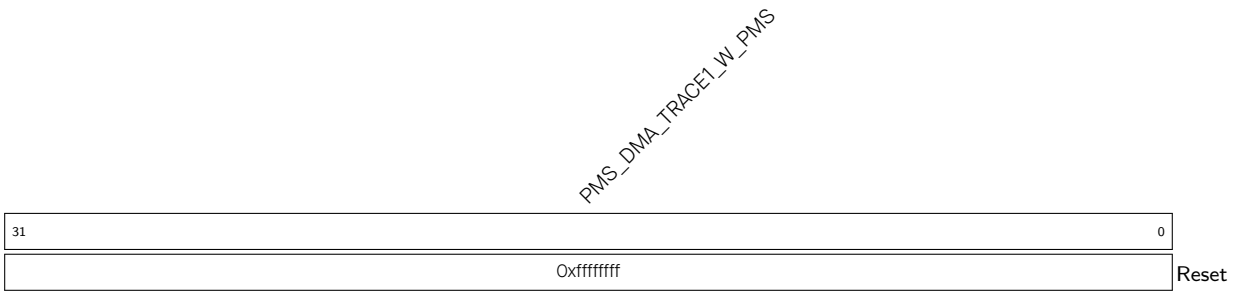
PMS_DMA_TRACE1_R_PMS 配置 TRACE1 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.46. PMS_DMA_TRACE1_PMS_W_REG (0x01D8)



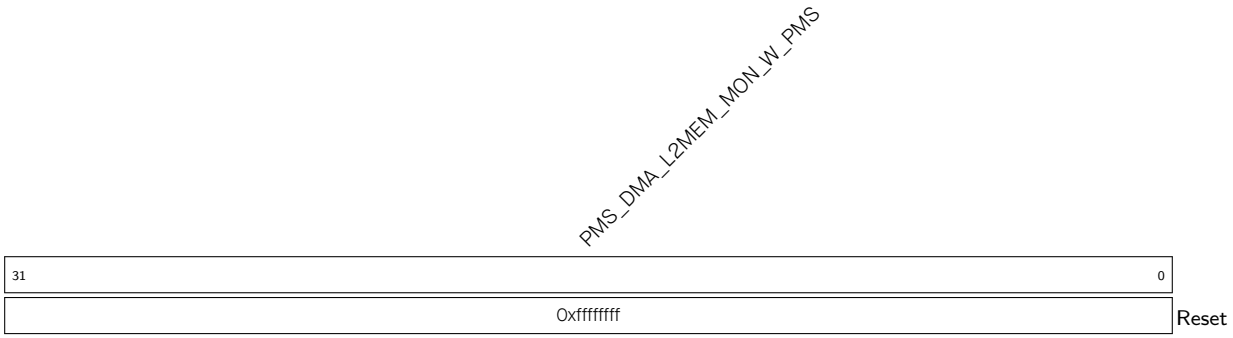
PMS_DMA_TRACE1_W_PMS 配置 TRACE1 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.47. PMS_DMA_L2MEM_MON_PMS_W_REG (0x01E0)



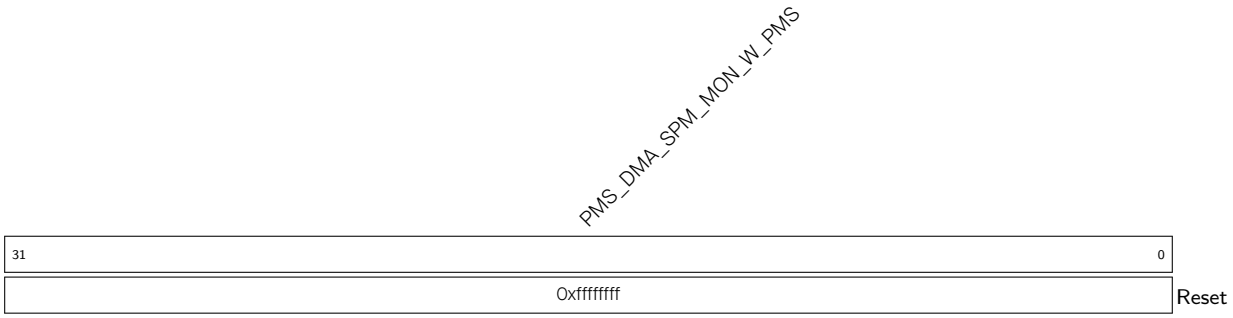
PMS_DMA_L2MEM_MON_W_PMS 配置 L2MEM 监测器写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.48. PMS_DMA_SPM_MON_PMS_W_REG (0x01E8)



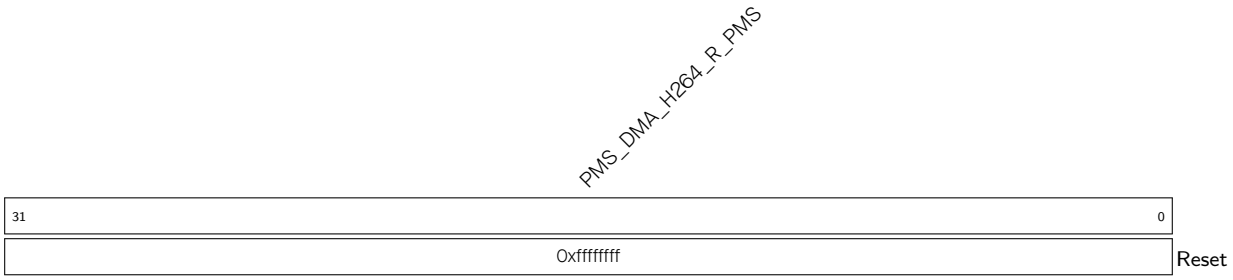
PMS_DMA_SPM_MON_W_PMS 配置 SPM 监测器写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.49. PMS_DMA_H264_PMS_R_REG (0x01FC)



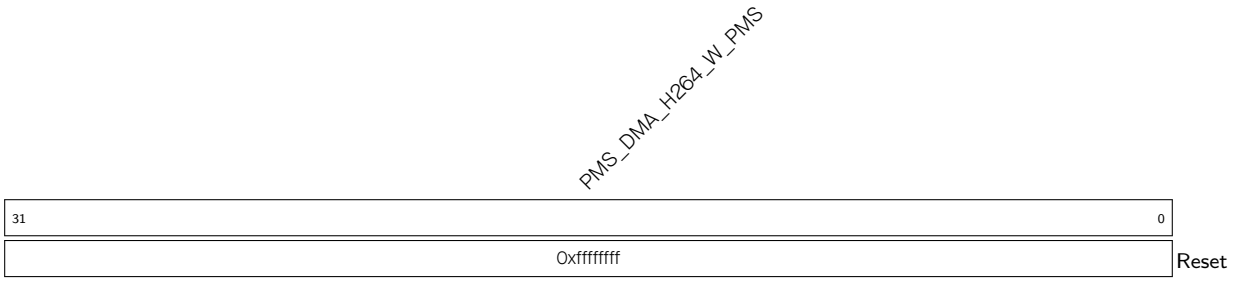
PMS_DMA_H264_R_PMS 配置 H264 DMA 读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.50. PMS_DMA_H264_PMS_W_REG (0x0200)



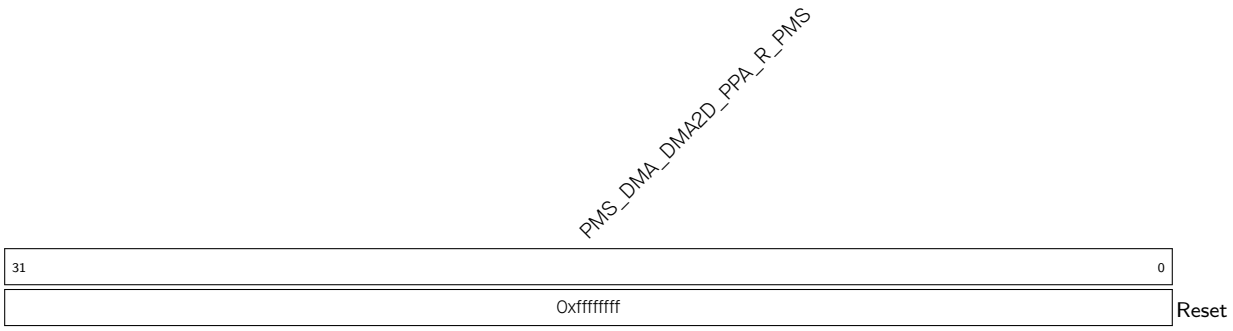
PMS_DMA_H264_W_PMS 配置 H264 DMA 写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.51. PMS_DMA_DMA2D_PPA_PMS_R_REG (0x0204)



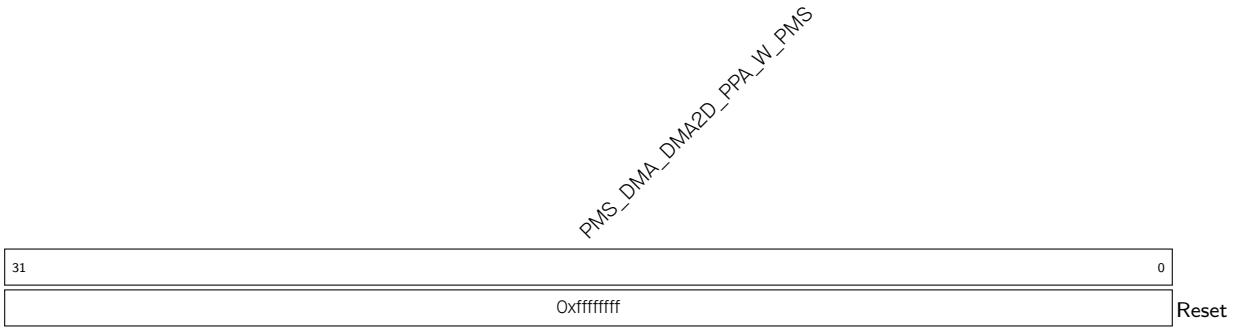
PMS_DMA_DMA2D_PPA_R_PMS 配置 2D-DMA 应 PPA（像素处理加速器）的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.52. PMS_DMA_DMA2D_PPA_PMS_W_REG (0x0208)



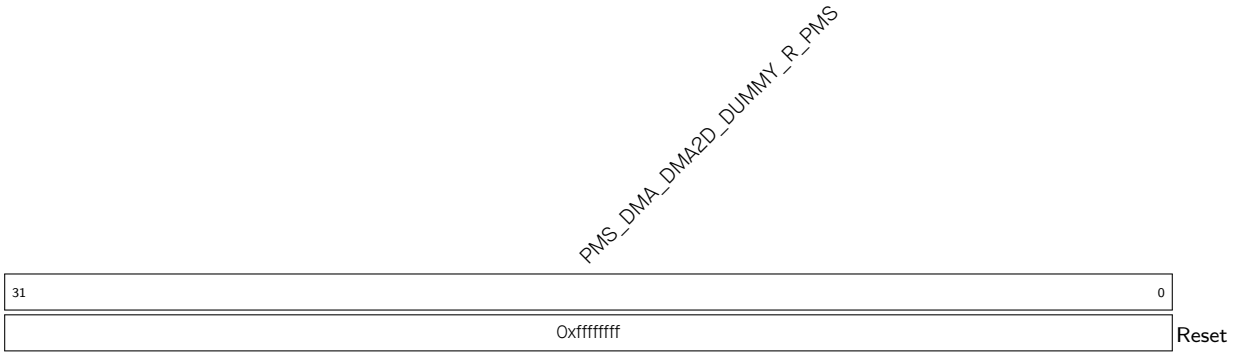
PMS_DMA_DMA2D_PPA_W_PMS 配置 2D-DMA 应 PPA（像素处理加速器）的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.53. PMS_DMA_DMA2D_DUMMY_PMS_R_REG (0x020C)



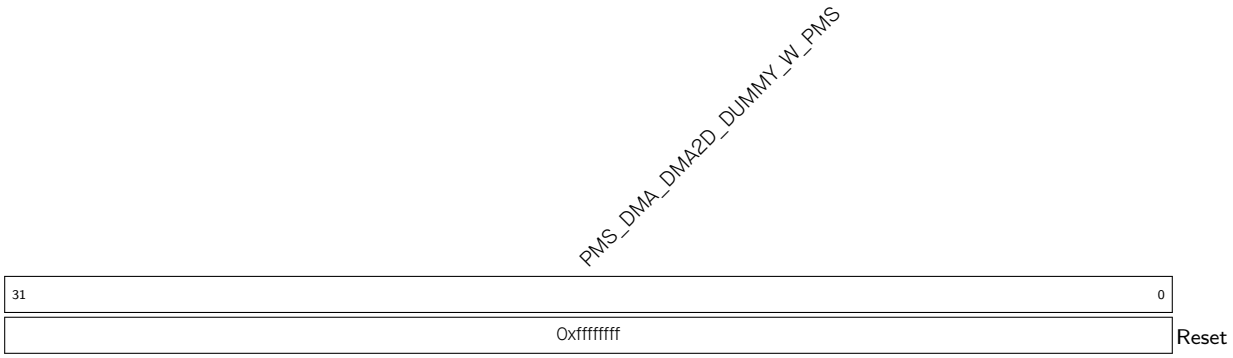
PMS_DMA_DMA2D_DUMMY_R_PMS 配置 2D-DMA 应 Dummy 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.54. PMS_DMA_DMA2D_DUMMY_PMS_W_REG (0x0210)



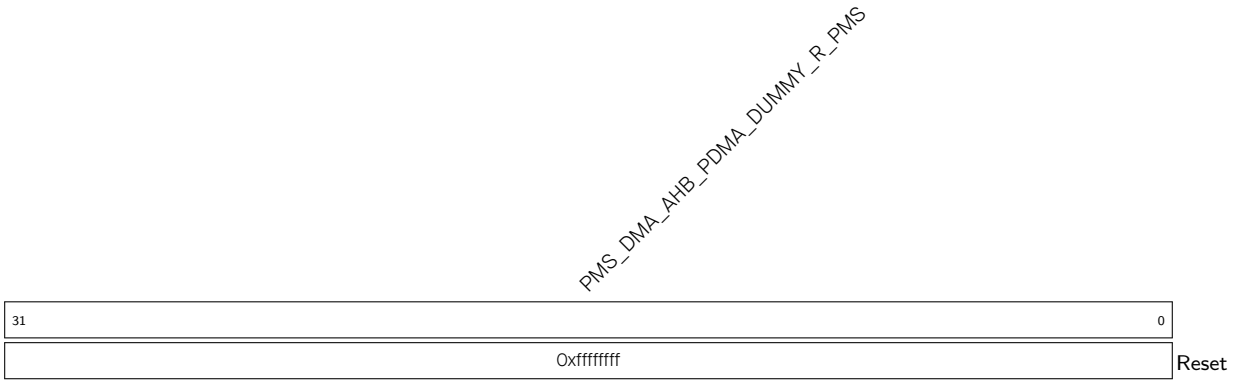
PMS_DMA_DMA2D_DUMMY_W_PMS 配置 2D-DMA 应 Dummy 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.55. PMS_DMA_AHB_PDMA_DUMMY_R_PMS_REG (0x0214)



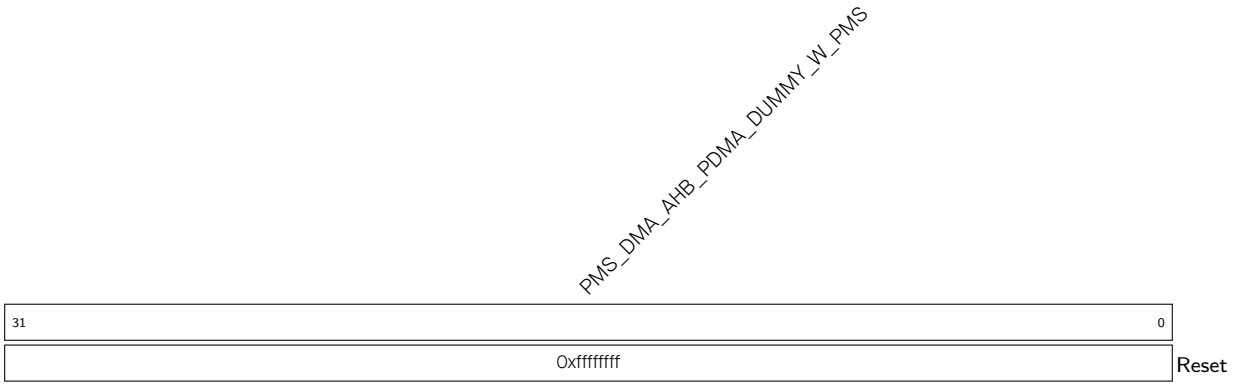
PMS_DMA_AHB_PDMA_DUMMY_R_PMS 配置 GDMA-AHB 应 Dummy 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.56. PMS_DMA_AHB_PDMA_DUMMY_W_PMS_REG (0x0218)



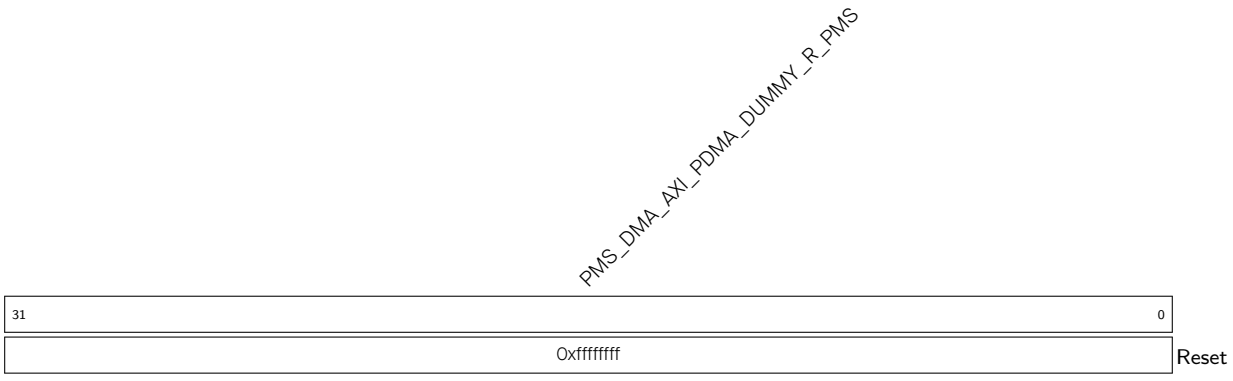
PMS_DMA_AHB_PDMA_DUMMY_W_PMS 配置 GDMA-AHB 应 Dummy 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

(R/W)

Register 20.57. PMS_DMA_AXI_PDMA_DUMMY_R_PMS_REG (0x021C)



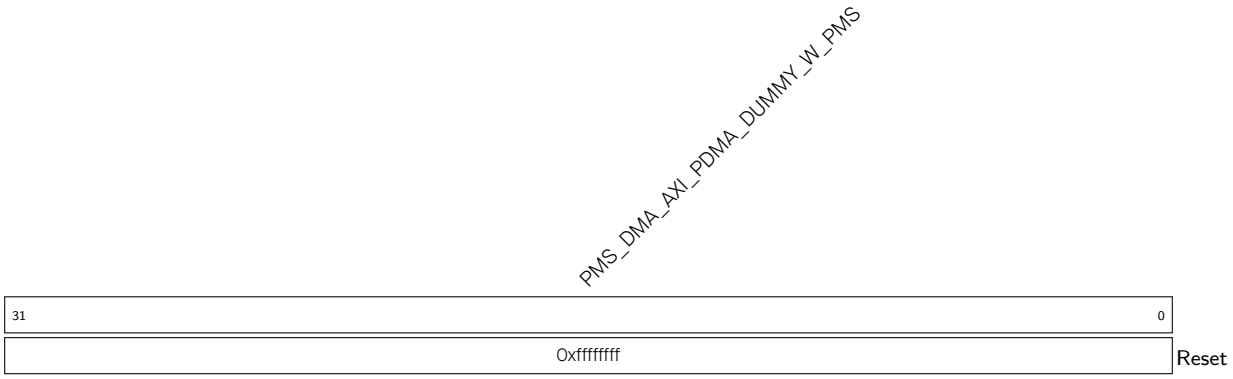
PMS_DMA_AXI_PDMA_DUMMY_R_PMS 配置 GDMA-AXI 应 Dummy 的请求读 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭读权限。

1：使能读权限。

(R/W)

Register 20.58. PMS_DMA_AXI_PDMA_DUMMY_W_PMS_REG (0x0220)



PMS_DMA_AXI_PDMA_DUMMY_W_PMS 配置 GDMA-AXI 应 Dummy 的请求写 32 个地址区间的权限。bit0 对应地址区间 0，以此类推。

0：关闭写权限。

1：使能写权限。

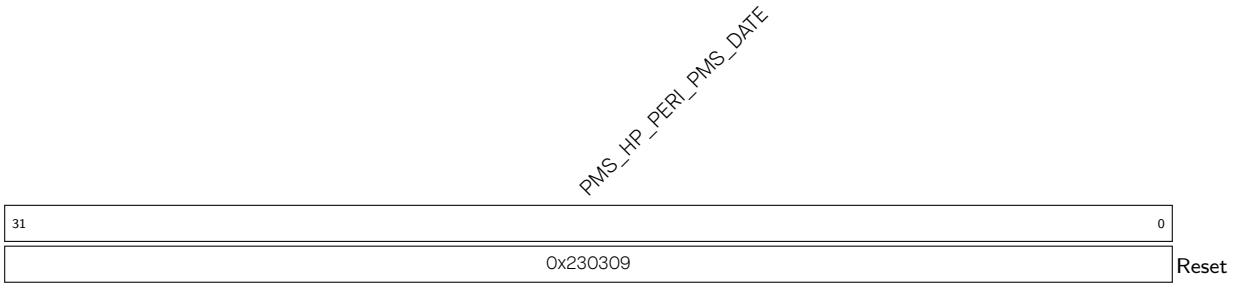
(R/W)

20.7.2 HP_PERI_PMS_REG

本小节的所有地址均为相对于 HP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

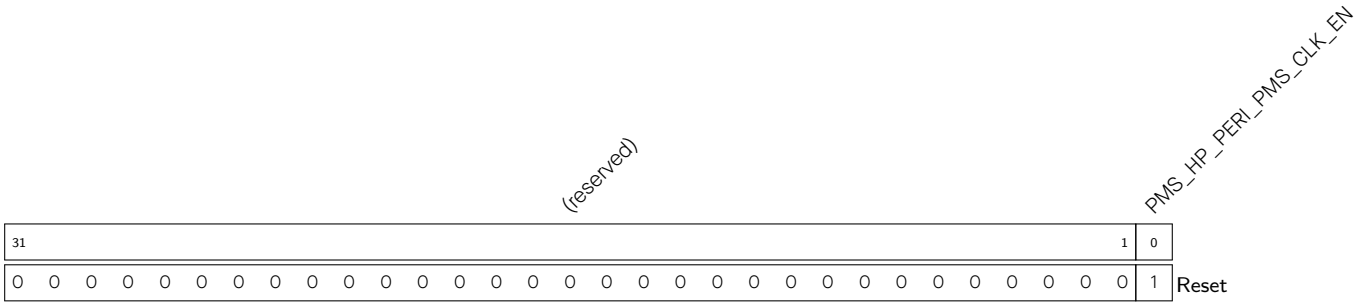
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 20.59. PMS_HP_PERI_PMS_DATE_REG (0x0000)



PMS_HP_PERI_PMS_DATE 版本控制寄存器。(R/W)

Register 20.60. PMS_HP_PERI_PMS_CLK_EN_REG (0x0004)



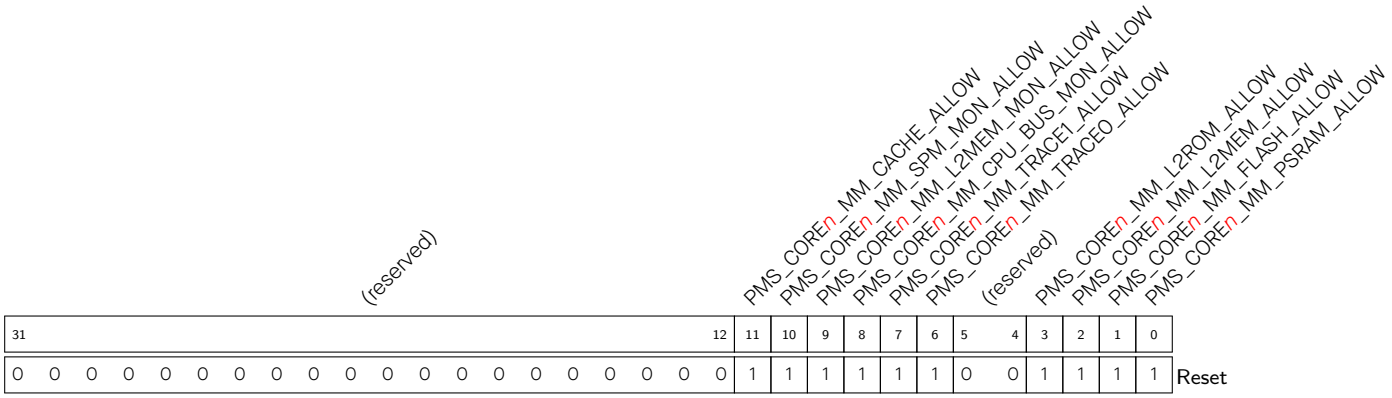
PMS_HP_PERI_PMS_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 20.61. PMS_CORE*n*_MM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0008+0x20*n)



PMS_CORE*n*_MM_PSRAM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限不通过 cache 直接访问外部 RAM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_FLASH_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限不通过 cache 直接访问外部 flash。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_L2MEM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限不通过 cache 直接访问 HP L2MEM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_L2ROM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限不通过 cache 直接访问 HP ROM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_TRACE0_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 TRACE0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_TRACE1_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 TRACE1。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.61. PMS_CORE*n*_MM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0008+0x20**n*)

接上页……

PMS_CORE*n*_MM_CPU_BUS_MON_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 CPU 总线监测器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_L2MEM_MON_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 L2MEM 监测器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_SPM_MON_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 SPM 监测器。

0：无权限

1：有权限

(R/W)

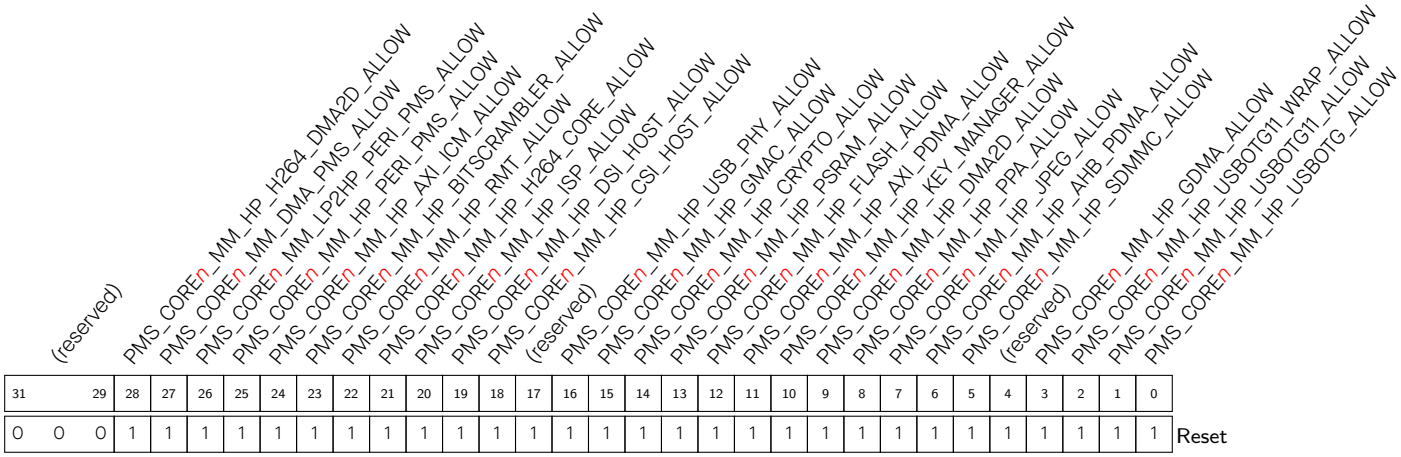
PMS_CORE*n*_MM_CACHE_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 cache。

0：无权限

1：有权限

(R/W)

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)



PMS_CORE*n*_MM_HP_USBOTG_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 USB 2.0 高速 OTG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_USBOTG11_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 USB 2.0 全速 OTG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_USBOTG11_WRAP_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 USB 2.0 全速 OTG wrap。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_GDMA_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP VDMA。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_SDMMC_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP SDMMC。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_MM_HP_AHB_PDMA_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 GDMA-AHB。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_JPEG_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP JPEG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_PPA_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP PPA。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_DMA2D_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 2D-DMA。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_KEY_MANAGER_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 密钥管理器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_AXI_PDMA_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP GDMA-AXI。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_FLASH_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP flash MSPI 控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_PSRAM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP PSRAM MSPI 控制器。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_MM_HP_CRYPTO_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP CRYPTO（加密模块，包括 AES/SHA/RSA/HMAC 加速器）。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_GMAC_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP EMAC。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_USB_PHY_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 USB 2.0 高速 OTG PHY。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_CSI_HOST_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP MIPI CSI 主机。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_DSI_HOST_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP MIPI DSI 主机。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_ISP_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP ISP（图像信号处理器）。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_H264_CORE_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP H264 编码器。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.62. PMS_CORE*n*_MM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_MM_HP_RMT_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP RMT。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_BITSCRAMBLER_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 比特传输控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_AXI_ICM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP AXI ICM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_PERI_PMS_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_LP2HP_PERI_PMS_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP2HP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_DMA_PMS_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP_DMA_PMS_REG。

0：无权限

1：有权限

(R/W)

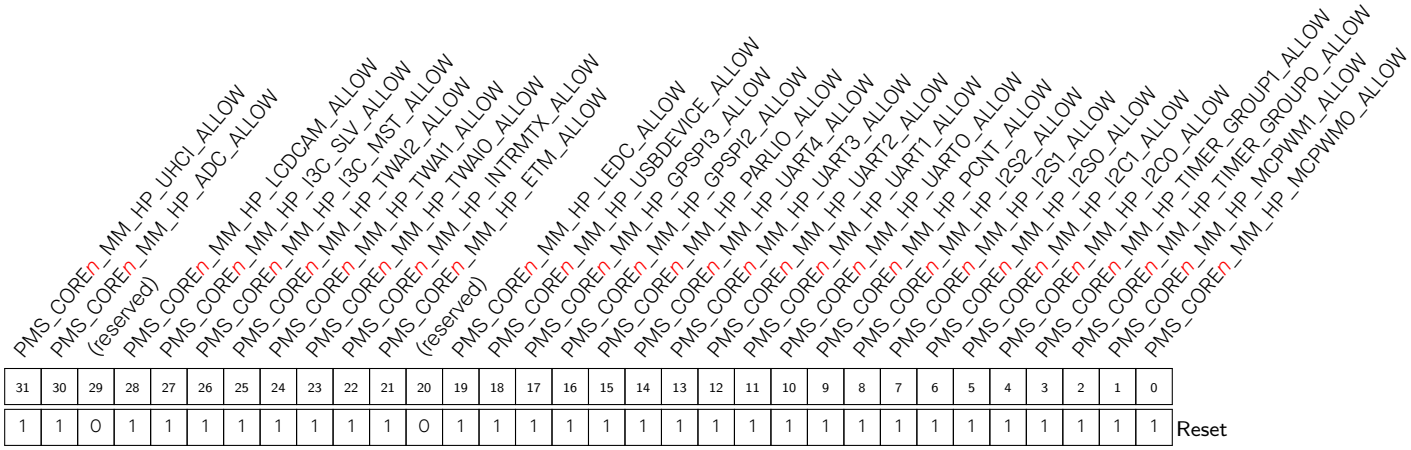
PMS_CORE*n*_MM_HP_H264_DMA2D_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 2D-DMA。

0：无权限

1：有权限

(R/W)

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0010+0x20*n)



PMS_CORE*n*_MM_HP_MCPWM0_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP MCPWM0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_MCPWM1_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP MCPWM1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_TIMER_GROUP0_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 定时器组 0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_TIMER_GROUP1_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 定时器组 1.

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_I2C0_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP I2C0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_I2C1_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP I2C1。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_MM_HP_I2S0_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP I2S0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_I2S1_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP I2S1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_I2S2_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP I2S2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_PCNT_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP PCNT（脉冲计数控制器）。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_UART0_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP UART0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_UART1_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP UART1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_UART2_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP UART2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_UART3_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP UART3。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_UART4_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP UART4。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_MM_HP_PARLIO_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 并行 IO 控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_GPSPI2_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP GPSPI2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_GPSPI3_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP GPSPI3。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_USBDEVICE_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP USB 串口/JTAG 控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_LEDC_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP LEDC。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_ETM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP ETM（事件任务矩阵）。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_INTRMTX_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 中断矩阵。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_TWAIO_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP TWAIO。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.63. PMS_CORE*n*_MM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_MM_HP_TWAI1_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP TWAI1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_TWAI2_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP TWAI2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_I3C_MST_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP I3C 主机控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_I3C_SLV_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP I3C 从机控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_LCDCAM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP LCD_CAM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_ADC_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP ADC。

0：无权限

1：有权限

(R/W)

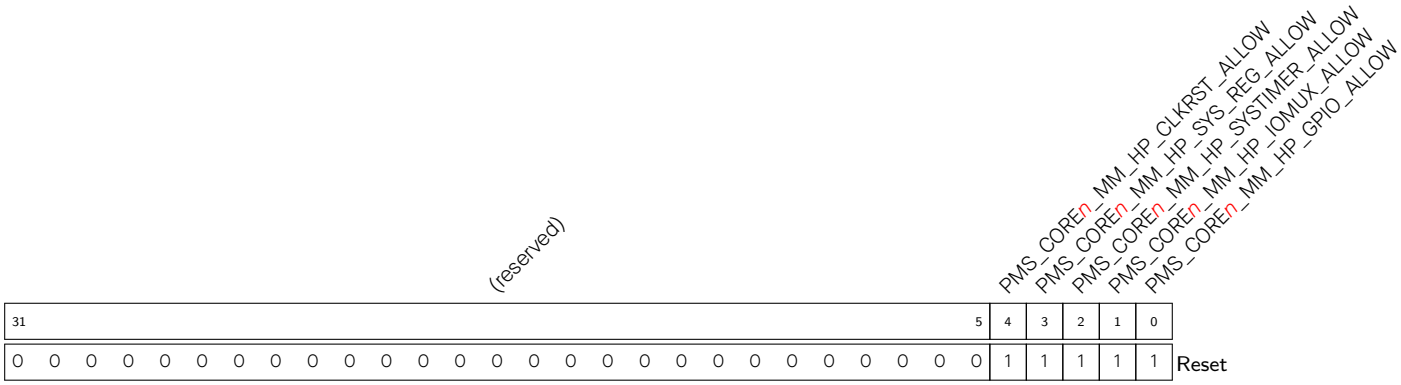
PMS_CORE*n*_MM_HP_UHCI_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP UHCI。

0：无权限

1：有权限

(R/W)

Register 20.64. PMS_CORE*n*_MM_HP_PERI_PMS_REG3_REG (*n*: 0-1) (0x0014+0x20*n)



PMS_CORE*n*_MM_HP_GPIO_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP GPIO 矩阵。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_IOMUX_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP IO MUX。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_SYSTIMER_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 系统定时器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_MM_HP_SYS_REG_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP 系统寄存器。

0：无权限

1：有权限

(R/W)

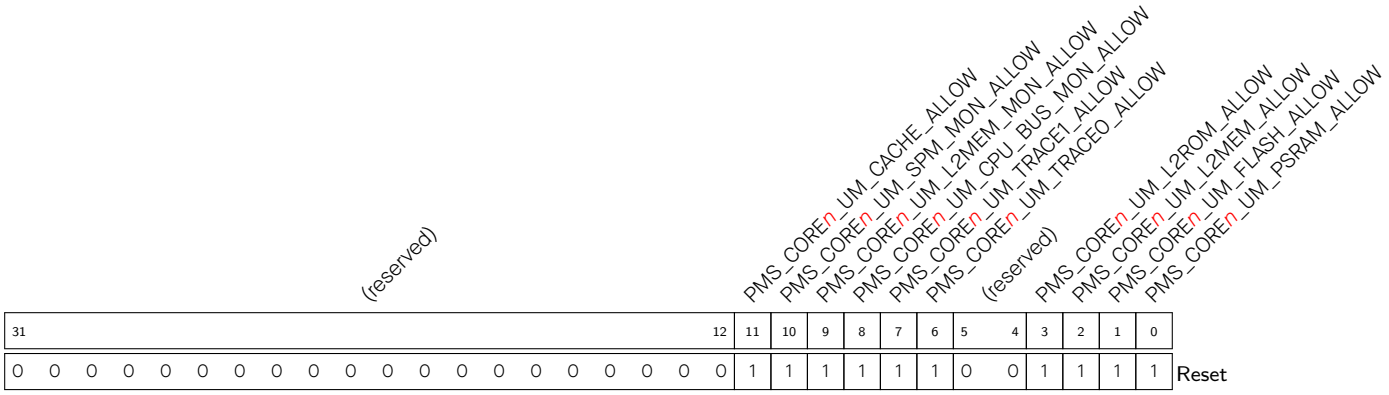
PMS_CORE*n*_MM_HP_CLKRST_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP_SYS_CLKRST。

0：无权限

1：有权限

(R/W)

Register 20.65. PMS_CORE*n*_UM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x0018+0x20*n)



PMS_CORE*n*_UM_PSRAM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问外部 RAM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_FLASH_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问外部 flash。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_L2MEM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP L2MEM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_L2ROM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP ROM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_TRACE0_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 TRACE0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_TRACE1_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 TRACE1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_CPU_BUS_MON_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 CPU 总线监测器。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.65. PMS_CORE*n*_UM_HP_PERI_PMS_REG0_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_UM_L2MEM_MON_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 L2MEM 监测器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_SPM_MON_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 SPM 监测器。

0：无权限

1：有权限

(R/W)

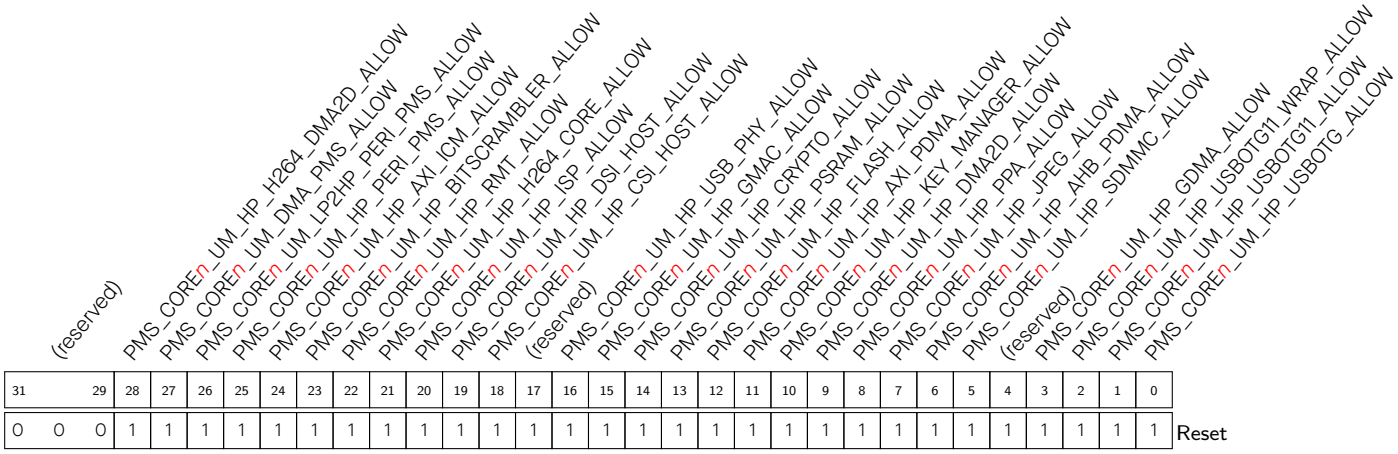
PMS_CORE*n*_UM_CACHE_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 cache。

0：无权限

1：有权限

(R/W)

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x001C+0x20*n*)



PMS_CORE*n*_UM_HP_USBOTG_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 USB 2.0 高速 OTG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_USBOTG11_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 USB 2.0 全速 OTG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_USBOTG11_WRAP_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 USB 2.0 全速 OTG wrap。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_GDMA_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP VDMA。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_SDMMC_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP SDMMC。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_AHB_PDMA_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 GDMA-AHB。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20*n)

接上页……

PMS_CORE*n*_UM_HP_JPEG_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP JPEG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_PPA_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP PPA。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_DMA2D_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 2D-DMA。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_KEY_MANAGER_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 密钥管理器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_AXI_PDMA_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP GDMA-AXI。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_FLASH_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP flash MSPI 控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_PSRAM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP PSRAM MSPI 控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_CRYPTO_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 加密模块。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20*n)

接上页……

PMS_CORE*n*_UM_HP_GMAC_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP EMAC。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_USB_PHY_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 USB 2.0 高速 OTG PHY。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_CSI_HOST_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP MIPI CSI 主机。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_DSI_HOST_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP MIPI DSI 主机。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_ISP_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP ISP。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_H264_CORE_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP H264 编码器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_RMT_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP RMT。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_BITSCRAMBLER_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 比特传输控制器。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.66. PMS_CORE*n*_UM_HP_PERI_PMS_REG1_REG (*n*: 0-1) (0x000C+0x20**n*)

接上页……

PMS_CORE*n*_UM_HP_AXI_ICM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP AXI ICM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_PERI_PMS_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_LP2HP_PERI_PMS_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP2HP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_DMA_PMS_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP_DMA_PMS_REG。

0：无权限

1：有权限

(R/W)

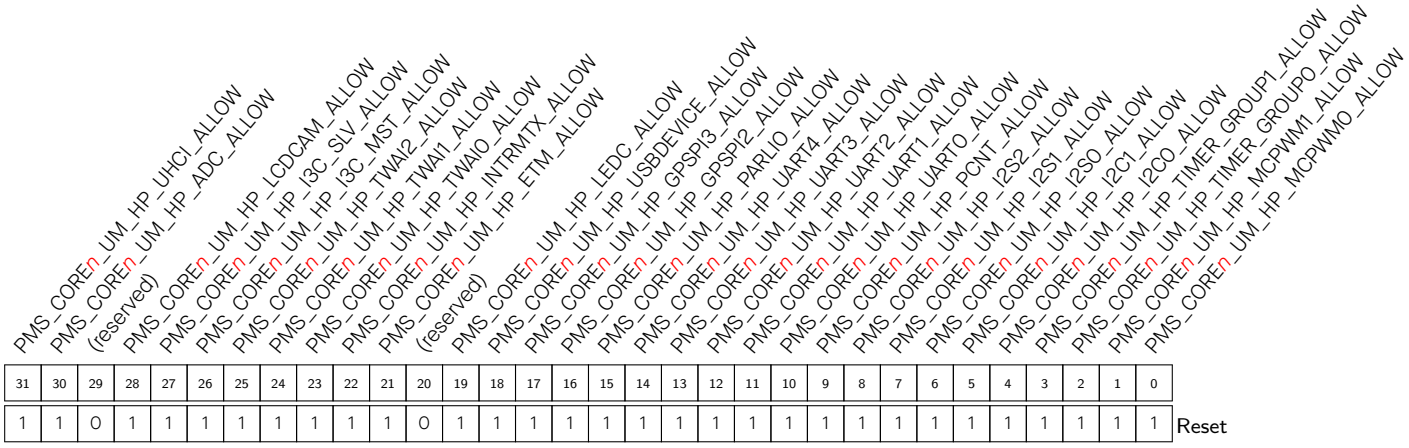
PMS_CORE*n*_UM_HP_H264_DMA2D_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 2D-DMA。

0：无权限

1：有权限

(R/W)

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Reset

PMS_CORE*n*_UM_HP_MCPWM0_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP MCPWM0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_MCPWM1_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP MCPWM1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_TIMER_GROUP0_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 定时器组 0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_TIMER_GROUP1_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 定时器组 1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_I2C0_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP I2C0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_I2C1_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP I2C1。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

接上页……

PMS_CORE*n*_UM_HP_I2S0_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP I2S0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_I2S1_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP I2S1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_I2S2_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP I2S2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_PCNT_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP PCNT。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_UART0_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP UART0。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_UART1_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP UART1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_UART2_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP UART2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_UART3_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP UART3。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_UART4_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP UART4。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

接上页……

PMS_CORE*n*_UM_HP_PARLIO_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 并行 IO 控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_GPSPI2_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP GP-SPI2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_GPSPI3_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP GP-SPI3。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_USBDEVICE_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP USB 串口/JTAG 控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_LEDC_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP LEDC。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_ETM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP ETM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_INTRMTX_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 中断矩阵。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_TWAIO_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP TWAIO。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.67. PMS_CORE*n*_UM_HP_PERI_PMS_REG2_REG (*n*: 0-1) (0x0020+0x20**n*)

接上页……

PMS_CORE*n*_UM_HP_TWAI1_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP TWAI1。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_TWAI2_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP TWAI2。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_I3C_MST_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP I3C 主机控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_I3C_SLV_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP I3C 从机控制器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_LCDCAM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP LCD_CAM。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_ADC_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP ADC。

0：无权限

1：有权限

(R/W)

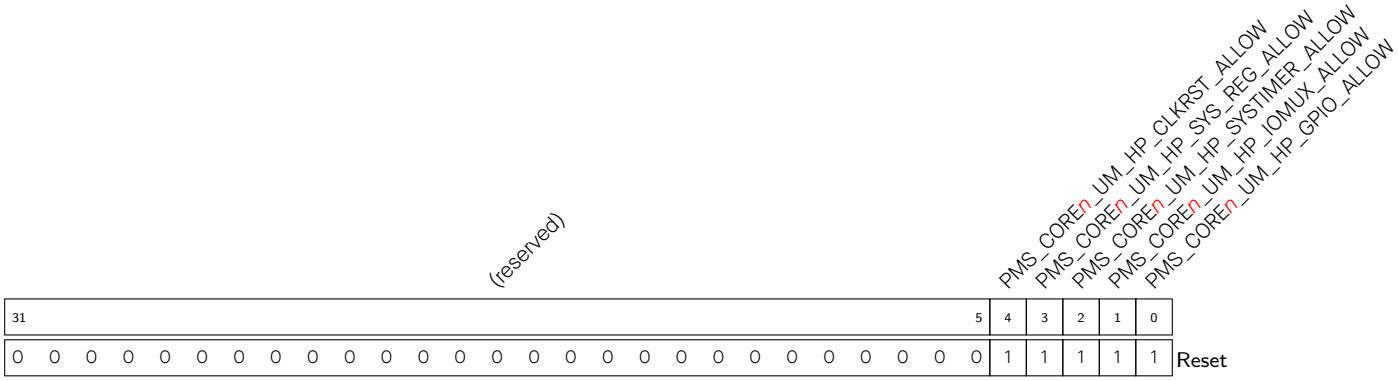
PMS_CORE*n*_UM_HP_UHCI_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP UHCI。

0：无权限

1：有权限

(R/W)

Register 20.68. PMS_CORE*n*_UM_HP_PERI_PMS_REG3_REG (*n*: 0-1) (0x0024+0x20*n)



PMS_CORE*n*_UM_HP_GPIO_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP GPIO 矩阵。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_IOMUX_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP IO MUX。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_SYSTIMER_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 系统定时器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_SYS_REG_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP 系统寄存器。

0：无权限

1：有权限

(R/W)

PMS_CORE*n*_UM_HP_CLKRST_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP_SYS_CLKRST。

0：无权限

1：有权限

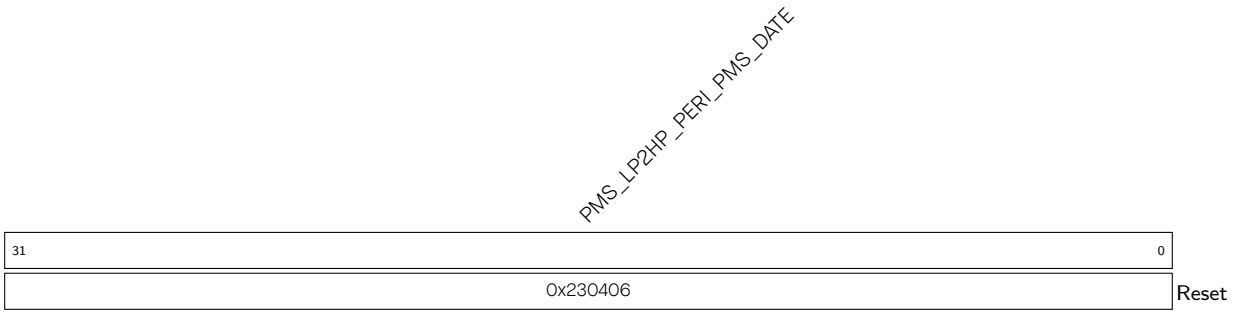
(R/W)

20.7.3 LP2HP_PERI_PMS_REG

本小节的所有地址均为相对于 LP2HP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

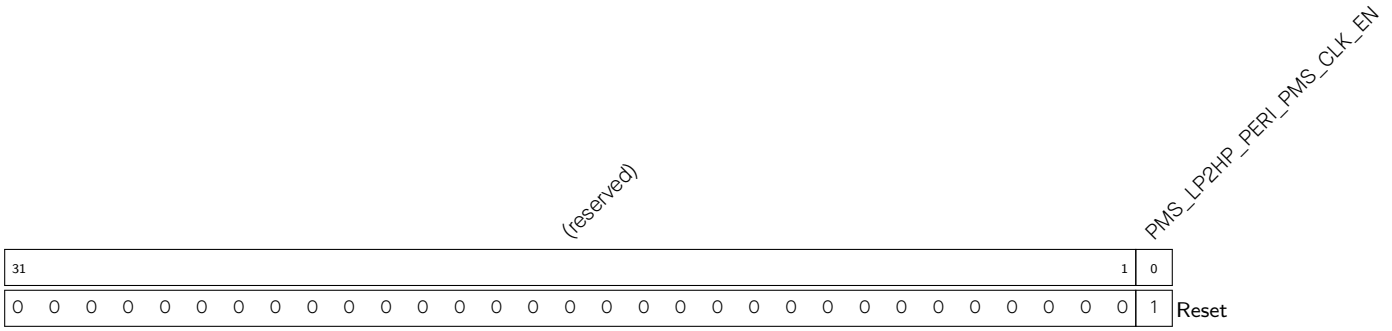
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 20.69. PMS_LP2HP_PERI_PMS_DATE_REG (0x0000)



PMS_LP2HP_PERI_PMS_DATE 版本控制寄存器。(R/W)

Register 20.70. PMS_LP2HP_PERI_PMS_CLK_EN_REG (0x0004)



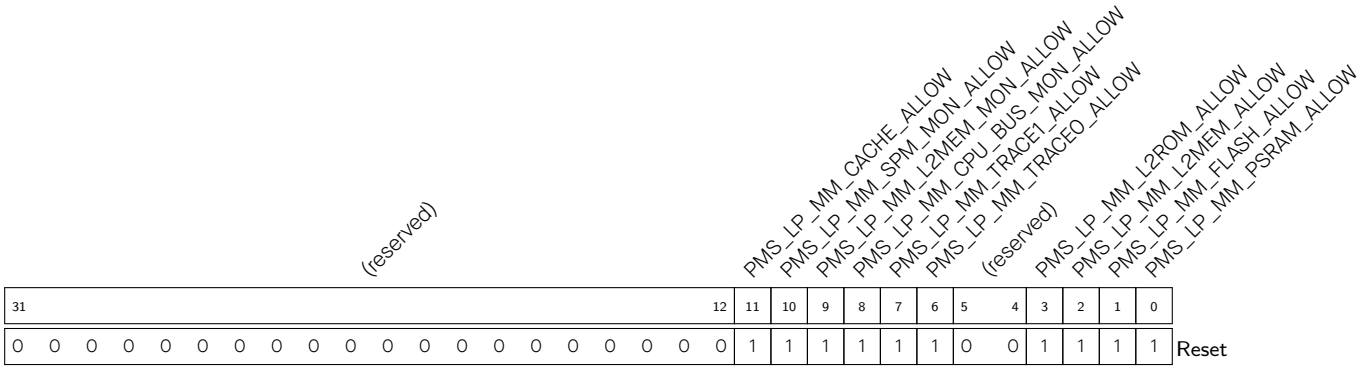
PMS_LP2HP_PERI_PMS_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 20.71. PMS_LP_MM_PMS_REG0_REG (0x0008)



PMS_LP_MM_PSRAM_ALLOW 配置 LP CPU 在机器模式下是否有权限不经过 cache 直接访问外部 RAM。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_FLASH_ALLOW 配置 LP CPU 在机器模式下是否有权限不经过 cache 直接访问外部 flash。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_L2MEM_ALLOW 配置 LP CPU 在机器模式下是否有权限不经过 cache 直接访问 HP L2M2M。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_L2ROM_ALLOW 配置 LP CPU 在机器模式下是否有权限不经过 cache 直接访问 HP ROM。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_TRACE0_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 TRACE0。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_TRACE1_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 TRACE1。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.71. PMS_LP_MM_PMS_REG0_REG (0x0008)

接上页……

PMS_LP_MM_CPU_BUS_MON_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 CPU 总线监控器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_L2MEM_MON_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 L2MEM 监控器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_SPM_MON_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 SPM 监控器。

0：无权限

1：有权限

(R/W)

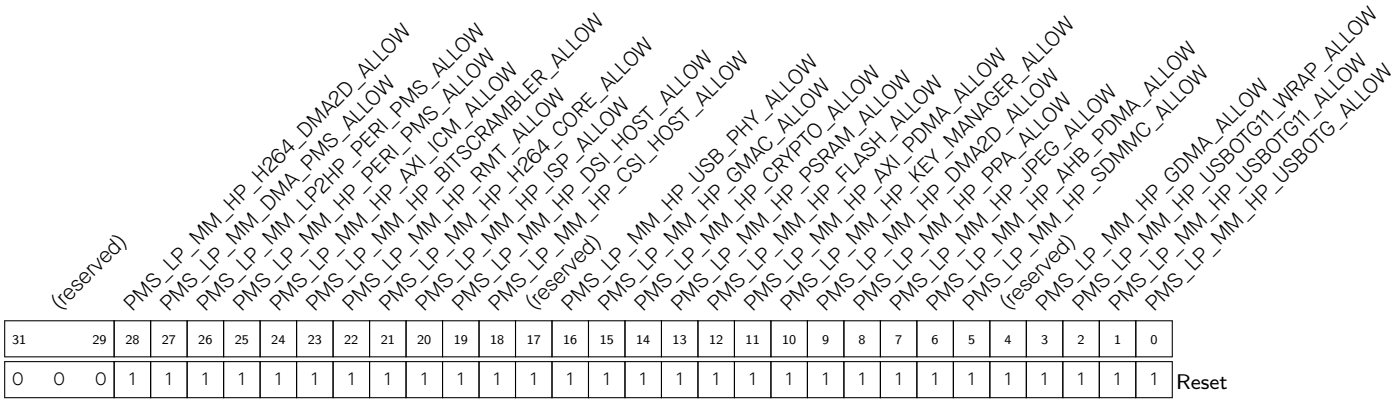
PMS_LP_MM_CACHE_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 cache。

0：无权限

1：有权限

(R/W)

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)



PMS_LP_MM_HP_USBOTG_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 USB 2.0 高速 OTG。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_USBOTG11_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 USB 2.0 全速 OTG。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_USBOTG11_WRAP_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 USB 2.0 全速 OTG wrap。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_GDMA_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP VDMA。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_SDMMC_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP SDMMC。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_AHB_PDMA_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 GDMA-AHB。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

接上页……

PMS_LP_MM_HP_JPEG_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP JPEG。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_PPA_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP PPA。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_DMA2D_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 2D-DMA。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_KEY_MANAGER_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 密钥管理器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_AXI_PDMA_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP GDMA-AXI。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_FLASH_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP flash MSPI 控制器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_PSRAM_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP PSRAM MSPI 控制器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_CRYPTO_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 加密模块。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

接上页……

PMS_LP_MM_HP_GMAC_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP EMAC。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_USB_PHY_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 USB 2.0 高速 OTG PHY。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_CSI_HOST_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP MIPI CSI 主机。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_DSI_HOST_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP MIPI DSI 主机。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_ISP_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP ISP。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_H264_CORE_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP H264 编码器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_RMT_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP RMT。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_BITSCRAMBLER_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 比特传输控制器。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.72. PMS_LP_MM_PMS_REG1_REG (0x0030)

接上页……

PMS_LP_MM_HP_AXI_ICM_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP AXI ICM。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_PERI_PMS_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP2HP_PERI_PMS_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP2HP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_DMA_PMS_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP_DMA_PMS_REG。

0：无权限

1：有权限

(R/W)

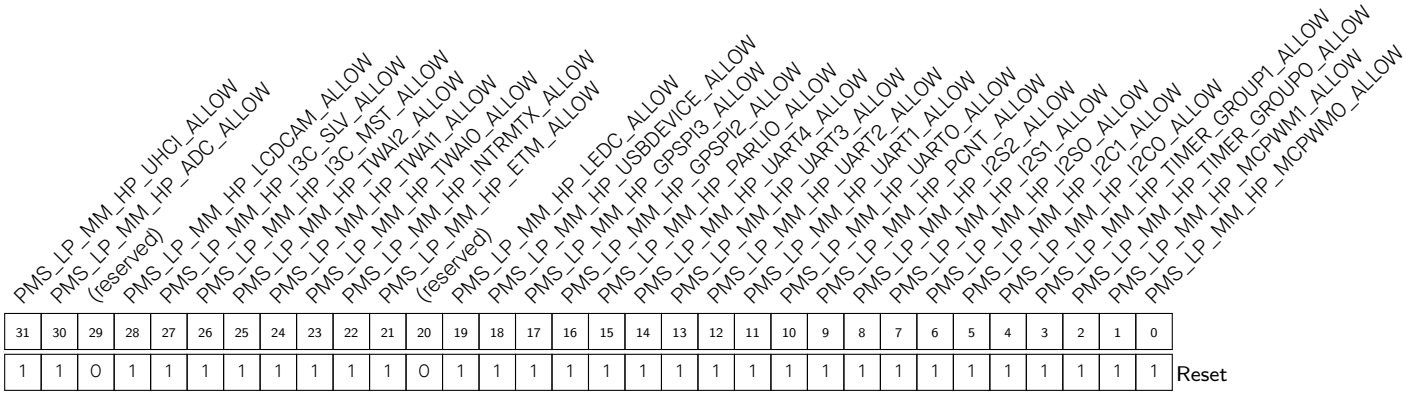
PMS_LP_MM_HP_H264_DMA2D_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 2D-DMA。

0：无权限

1：有权限

(R/W)

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)



PMS_LP_MM_HP_MCPWM0_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP MCPWM0。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_MCPWM1_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP MCPWM1。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_TIMER_GROUP0_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 定时器组 0。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_TIMER_GROUP1_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 定时器组 1。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_I2C0_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP I2C0。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_I2C1_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP I2C1。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_I2S0_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP I2S0。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

接上页……

PMS_LP_MM_HP_I2S1_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP I2S1。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_I2S2_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP I2S2。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_PCNT_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP PCNT。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_UART0_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP UART0。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_UART1_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP UART1。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_UART2_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP UART2。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_UART3_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP UART3。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_UART4_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP UART4。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_PARLIO_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 并行 IO 控制器。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

接上页……

PMS_LP_MM_HP_GPSPI2_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP GP-SPI2。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_GPSPI3_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP GP-SPI3。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_USBDEVICE_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP USB 串口/JTAG 控制器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_LEDC_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP LEDC。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_ETM_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP ETM。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_INTRMTX_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 中断矩阵。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_TWAI0_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP TWAI0。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_TWAI1_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP TWAI1。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_TWAI2_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP TWAI2。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.73. PMS_LP_MM_PMS_REG2_REG (0x00A4)

接上页……

PMS_LP_MM_HP_I3C_MST_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP I3C 主机控制器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_I3C_SLV_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP I3C 从机控制器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_LCDCAM_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP LCD_CAM。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_ADC_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP ADC。

0：无权限

1：有权限

(R/W)

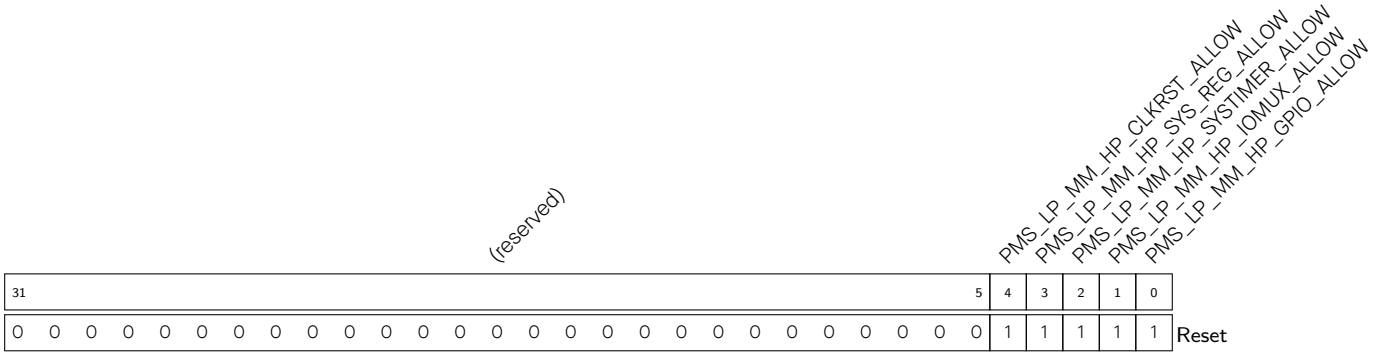
PMS_LP_MM_HP_UHCI_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP UHCI。

0：无权限

1：有权限

(R/W)

Register 20.74. PMS_LP_MM_PMS_REG3_REG (0x011C)



PMS_LP_MM_HP_GPIO_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP GPIO 矩阵。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_IOMUX_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP IO MUX。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_SYSTIMER_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 系统定时器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_SYS_REG_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP 系统寄存器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP_CLKRST_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP_SYS_CLKRST。

0：无权限

1：有权限

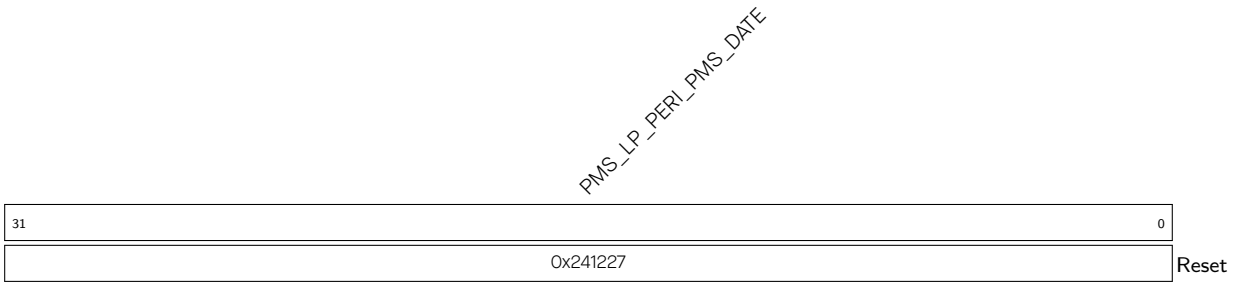
(R/W)

20.7.4 LP_PERI_PMS_REG

本小节的所有地址均为相对于 LP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

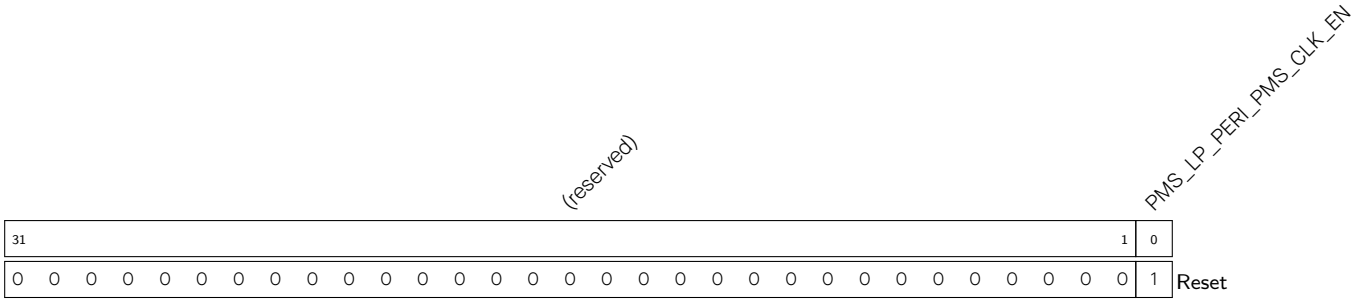
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 20.75. PMS_LP_PERI_PMS_DATE_REG (0x0000)



PMS_LP_PERI_PMS_DATE 版本控制寄存器。(R/W)

Register 20.76. PMS_LP_PERI_PMS_CLK_EN_REG (0x0004)



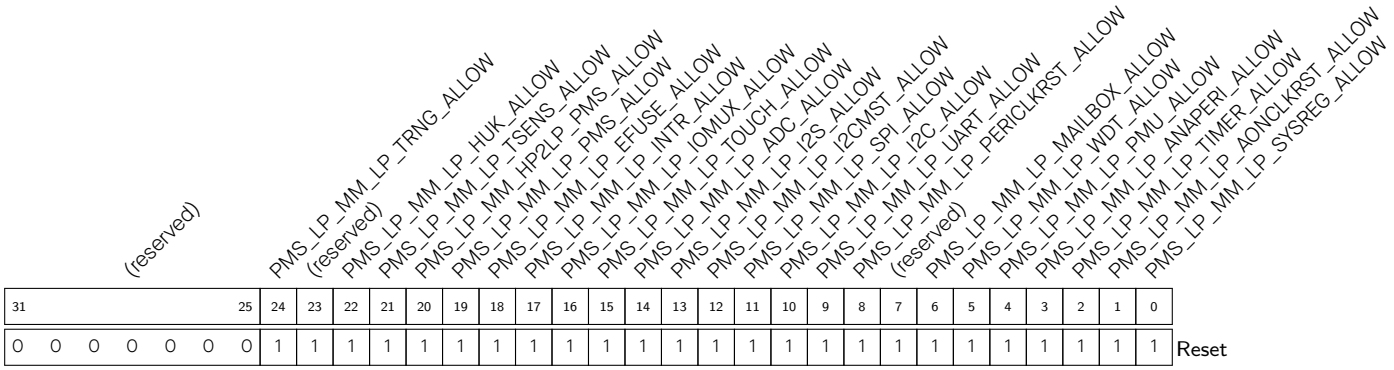
PMS_LP_PERI_PMS_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 20.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)



PMS_LP_MM_LP_SYSREG_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP 系统寄存器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_AONCLKRST_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP_AONCLKRST（LP 常开时钟与复位）。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_TIMER_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP 定时器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_ANAPERI_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP ANAPERI（模拟外设）。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_PMU_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP PMU（电源管理单元）。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_WDT_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP WDT（看门狗定时器）。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

接上页……

PMS_LP_MM_LP_MAILBOX_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP 信箱控制器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_PERICLKRST_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP PERICLKRST（外设时钟与复位）。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_UART_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP UART。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_I2C_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP I2S。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_SPI_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP SPI。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_I2CMST_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP I2C 主机。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_I2S_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP I2S。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_ADC_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP ADC。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_TOUCH_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP 触摸传感器。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.77. PMS_LP_MM_LP_PERI_PMS_REG0_REG (0x0008)

接上页……

PMS_LP_MM_LP_IOMUX_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP IO MUX。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_INTR_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP INTR（中断）。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_EFUSE_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP eFuse。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_PMS_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_HP2LP_PMS_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 HP2LP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_TSENS_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP 温度传感器。

0：无权限

1：有权限

(R/W)

PMS_LP_MM_LP_HUK_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP HUK（硬件唯一密钥）。

0：无权限

1：有权限

(R/W)

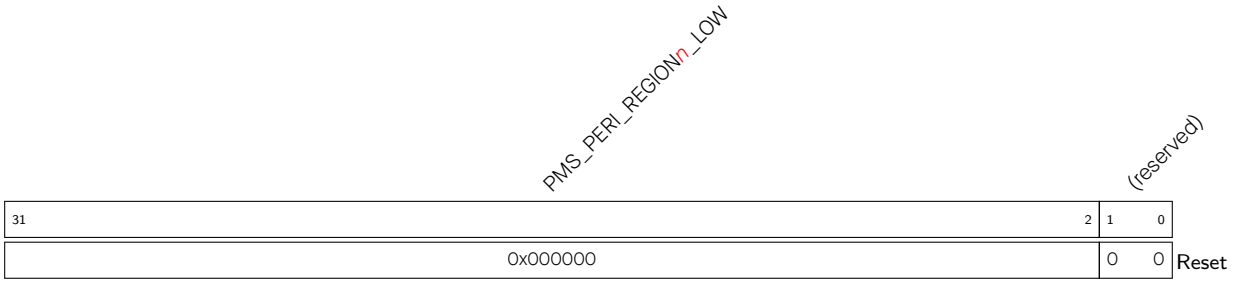
PMS_LP_MM_LP_TRNG_ALLOW 配置 LP CPU 在机器模式下是否有权限访问 LP TRNG。

0：无权限

1：有权限

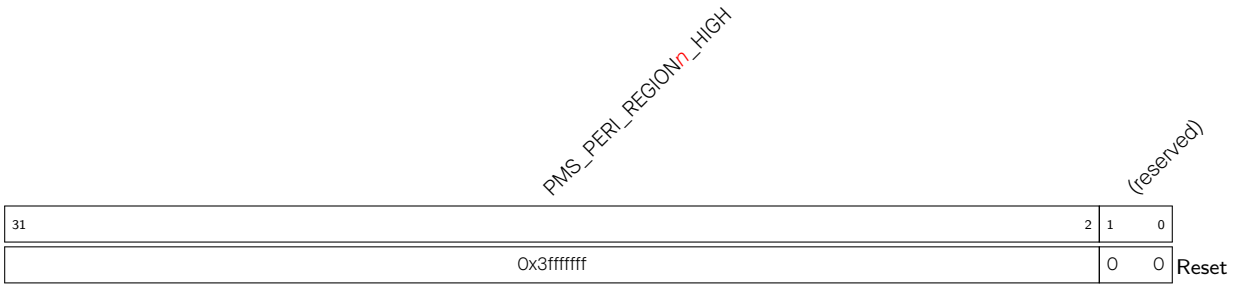
(R/W)

Register 20.78. PMS_PERI_REGION*n*_LOW_REG (*n*: 0-1) (0x000C+0x8**n*)



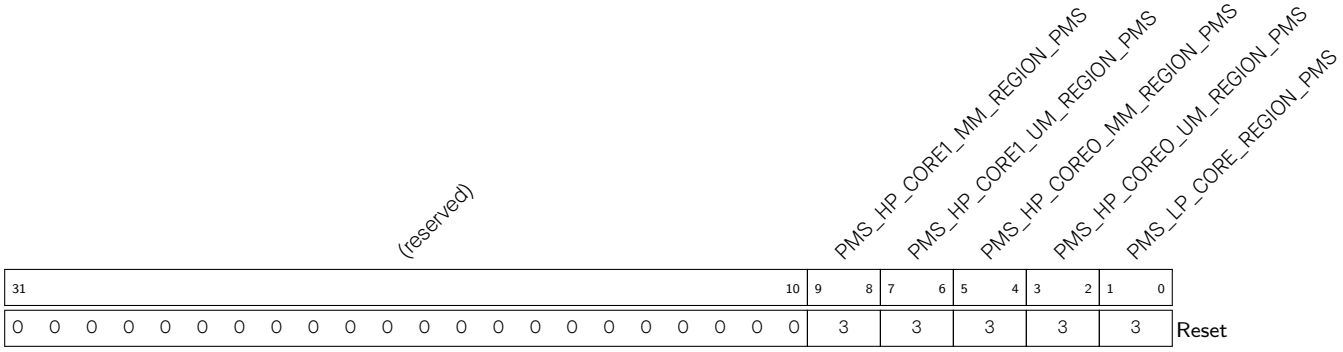
PMS_PERI_REGION*n*_LOW 配置外设寄存器地址区间 *n* 的起始地址的高 30 位。(R/W)

Register 20.79. PMS_PERI_REGION*n*_HIGH_REG (*n*: 0-1) (0x0010+0x8**n*)



PMS_PERI_REGION*n*_HIGH 配置外设寄存器地址区间 *n* 的结束地址的高 30 位。(R/W)

Register 20.80. PMS_PERI_REGION_PMS_REG (0x001C)



PMS_LP_CORE_REGION_PMS 配置 LP CPU 在机器模式下是否有权限访问地址区间 0 和 1。bit0 对应地址区间 0，bit1 对应地址区间 1。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE0_UM_REGION_PMS 配置 HP CPU0 在用户模式下是否有权限访问地址区间 0 和 1。bit2 对应地址区间 0，bit3 对应地址区间 1。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE0_MM_REGION_PMS 配置 HP CPU0 在机器模式下是否有权限访问地址区间 0 和 1。bit4 对应地址区间 0，bit5 对应地址区间 1。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE1_UM_REGION_PMS 配置 HP CPU1 在用户模式下是否有权限访问地址区间 0 和 1。bit6 对应地址区间 0，bit7 对应地址区间 1。

0：无权限

1：有权限

(R/W)

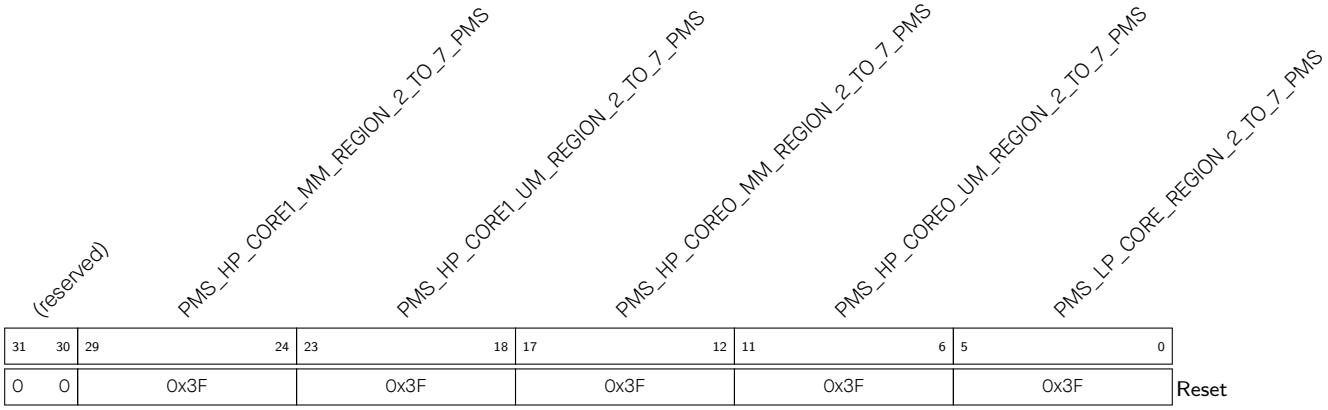
PMS_HP_CORE1_MM_REGION_PMS 配置 HP CPU1 在机器模式下是否有权限访问地址区间 0 和 1。bit8 对应地址区间 0，bit9 对应地址区间 1。

0：无权限

1：有权限

(R/W)

Register 20.81. PMS_PERI_REGION_2_TO_7_PMS_REG (0x0020)



PMS_LP_CORE_REGION_2_TO_7_PMS 配置 LP CPU 在机器模式下是否有权限访问地址区间 2 到 7。bit0 对应地址区间 2，bit5 对应地址区间 7。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE0_UM_REGION_2_TO_7_PMS 配置 HP CPU0 在用户模式下是否有权限访问地址区间 2 到 7。bit0 对应地址区间 2，bit5 对应地址区间 7。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE0_MM_REGION_2_TO_7_PMS 配置 HP CPU0 在机器模式下是否有权限访问地址区间 2 到 7。bit0 对应地址区间 2，bit5 对应地址区间 7。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE1_UM_REGION_2_TO_7_PMS 配置 HP CPU1 在用户模式下是否有权限访问地址区间 2 到 7。bit0 对应地址区间 2，bit5 对应地址区间 7。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE1_MM_REGION_2_TO_7_PMS 配置 HP CPU1 在机器模式下是否有权限访问地址区间 2 到 7。bit0 对应地址区间 2，bit5 对应地址区间 7。

0：无权限

1：有权限

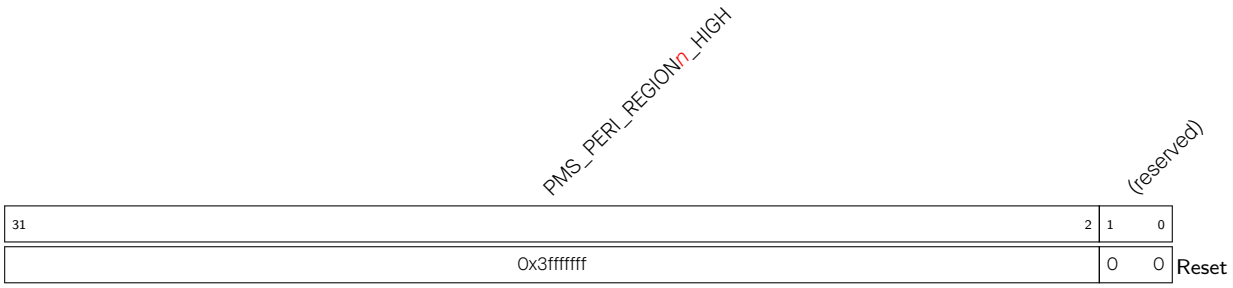
(R/W)

Register 20.82. PMS_PERI_REGION*m*_LOW_REG (*m*: 2-7) (0x0024+0x8*(*m*-2))



PMS_PERI_REGION*m*_LOW 配置外设寄存器地址区间 *m* 的起始地址的高 30 位。(R/W)

Register 20.83. PMS_PERI_REGION*m*_HIGH_REG (*m*: 2-7) (0x0028+0x8*(*m*-2))



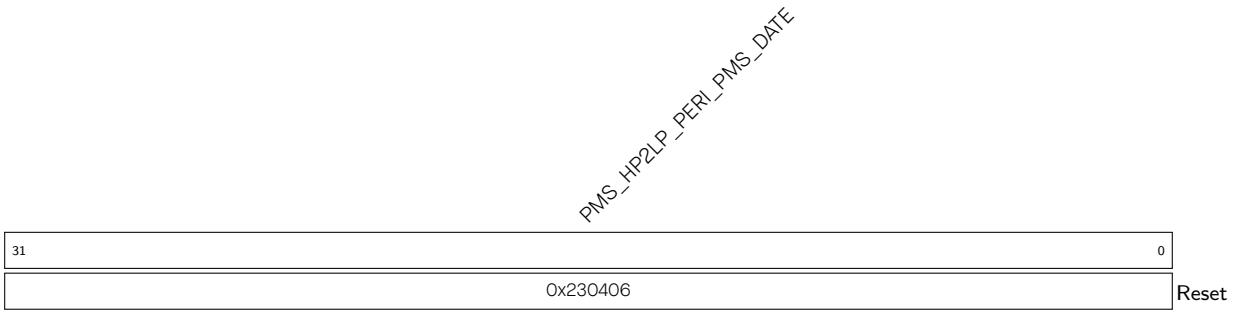
PMS_PERI_REGION*m*_HIGH 配置外设寄存器地址区间 *m* 的结束地址的高 30 位。(R/W)

20.7.5 HP2LP_PERI_PMS_REG

本小节的所有地址均为相对于 HP2LP_PERI_PMS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

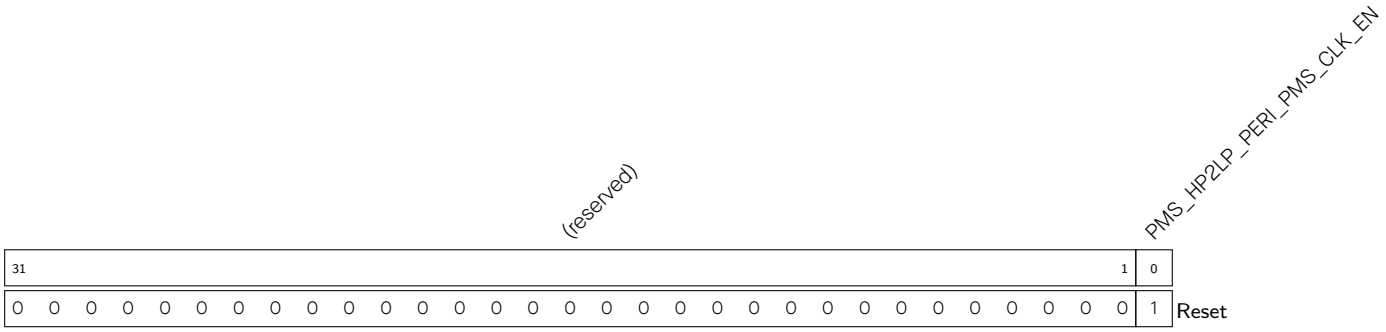
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 20.84. PMS_HP2LP_PERI_PMS_DATE_REG (0x0000)



PMS_HP2LP_PERI_PMS_DATE 版本控制寄存器。(R/W)

Register 20.85. PMS_HP2LP_PERI_PMS_CLK_EN_REG (0x0004)



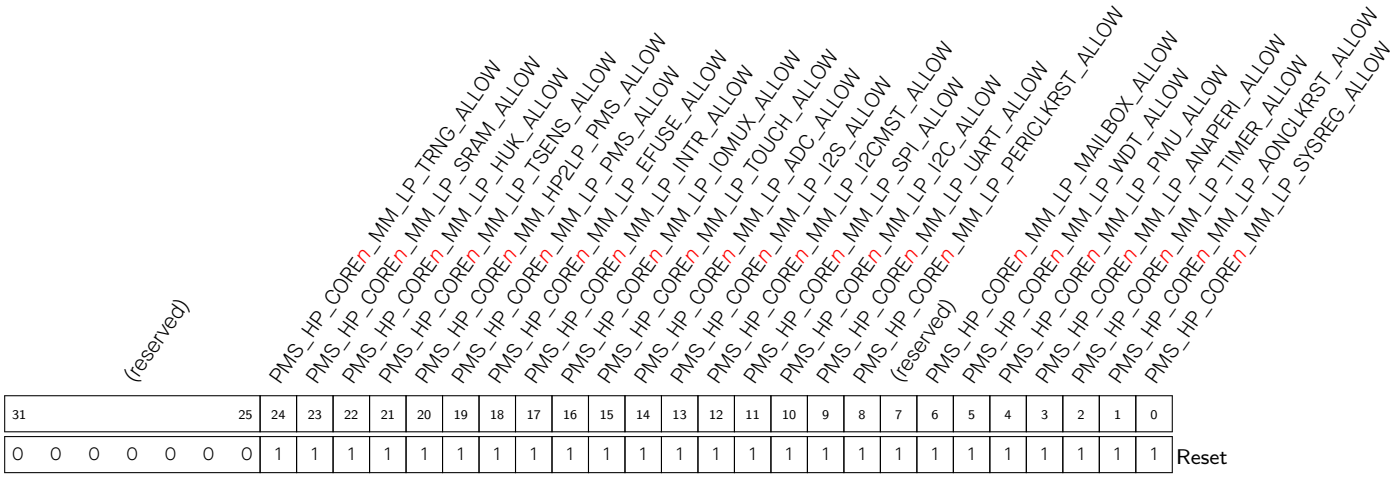
PMS_HP2LP_PERI_PMS_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8*n*)



PMS_HP_CORE*n*_MM_LP_SYSREG_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP 系统寄存器。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_MM_LP_AONCLKRST_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP_AONCLKRST。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_MM_LP_TIMER_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP 定时器。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_MM_LP_ANAPERI_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP ANAPERI。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_MM_LP_PMU_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP PMU。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_MM_LP_WDT_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP WDT。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

接上页……

PMS_HP_CORE*n*_MM_LP_MAILBOX_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP 信箱控制器。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_PERICLKRST_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP PERICLKRST（LP 外设时钟与复位）。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_UART_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP UART。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_I2C_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP I2C。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_SPI_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP SPI。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_I2CMST_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP I2C 主机。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_I2S_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP I2S。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_ADC_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP ADC。
0：无权限
1：有权限
(R/W)

见下页……

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

接上页……

PMS_HP_CORE*n*_MM_LP_TOUCH_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP 触摸传感器。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_IOMUX_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP IO MUX。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_INTR_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP INTR。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_EFUSE_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP eFuse。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_PMS_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP_PERI_PMS。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_HP2LP_PMS_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 HP2LP_PERI_PMS。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_TSENS_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP 温度传感器。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_MM_LP_HUK_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP HUK。
0：无权限
1：有权限
(R/W)

见下页……

Register 20.86. PMS_HP_CORE*n*_MM_PMS_REG0_REG (*n*: 0-1) (0x0008+0x8**n*)

接上页……

PMS_HP_CORE*n*_MM_LP_SRAM_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP SRAM。

0：无权限

1：有权限

(R/W)

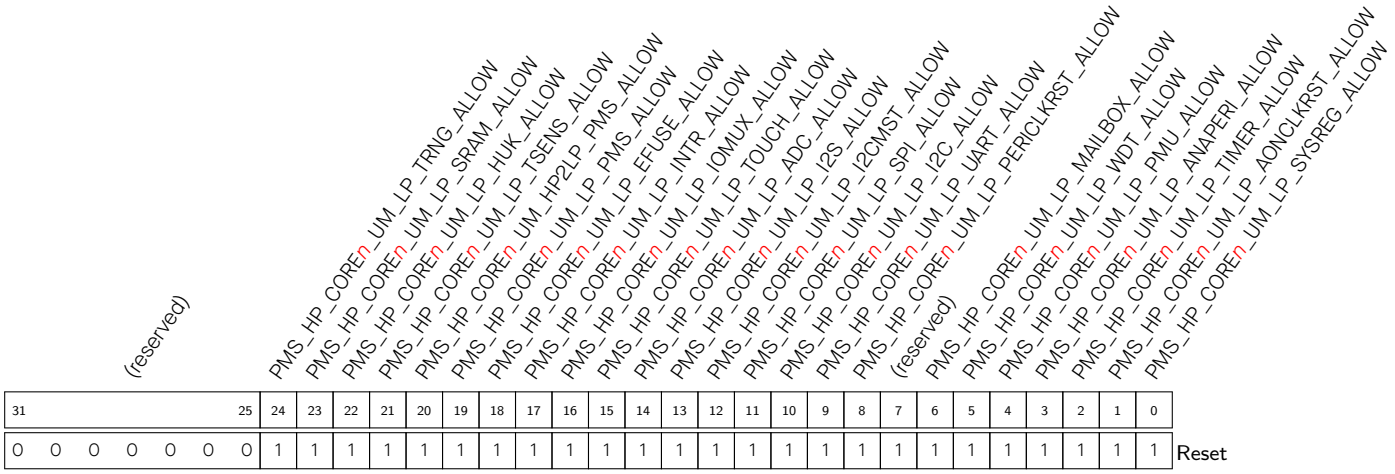
PMS_HP_CORE*n*_MM_LP_TRNG_ALLOW 配置 HP CPU*n* 在机器模式下是否有权限访问 LP TRNG。

0：无权限

1：有权限

(R/W)

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)



PMS_HP_CORE*n*_UM_LP_SYSREG_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP 系统寄存器。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_AONCLKRST_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP_AONCLKRST。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_TIMER_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP 定时器。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_ANAPERI_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP ANAPERI。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_PMU_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP PMU。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_WDT_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP WDT。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8*n)

接上页……

PMS_HP_CORE*n*_UM_LP_MAILBOX_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP 信箱控制器。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_UM_LP_PERICLKRST_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP PERICLKRST。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_UM_LP_UART_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP UART。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_UM_LP_I2C_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP I2C。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_UM_LP_SPI_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP SPI。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_UM_LP_I2CMST_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP I2C 主机。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_UM_LP_I2S_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP I2S。
0：无权限
1：有权限
(R/W)

PMS_HP_CORE*n*_UM_LP_ADC_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP ADC。
0：无权限
1：有权限
(R/W)

见下页……

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

接上页……

PMS_HP_CORE*n*_UM_LP_TOUCH_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP 触摸传感器。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_IOMUX_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP IO MUX。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_INTR_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP INTR。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_EFUSE_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP eFuse。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_PMS_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_HP2LP_PMS_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 HP2LP_PERI_PMS_REG。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_TSENS_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP 温度传感器。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_HUK_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP HUK。

0：无权限

1：有权限

(R/W)

见下页……

Register 20.87. PMS_HP_CORE*n*_UM_PMS_REG0_REG (*n*: 0-1) (0x000C+0x8**n*)

接上页……

PMS_HP_CORE*n*_UM_LP_SRAM_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP SRAM。

0：无权限

1：有权限

(R/W)

PMS_HP_CORE*n*_UM_LP_TRNG_ALLOW 配置 HP CPU*n* 在用户模式下是否有权限访问 LP TRNG。

0：无权限

1：有权限

(R/W)

第 21 章

系统寄存器 (SYSREG)

21.1 概述

ESP32-P4 支持丰富的辅助功能，可由专用寄存器进行配置。本章将主要介绍这些寄存器。有关具体功能的介绍，请前往对应章节。

21.2 功能描述

21.2.1 HP 系统寄存器

21.2.1.1 外部存储器加密/解密配置

寄存器 HP_SYSTEM_CRYPTO_CTRL_REG 用于配置外部存储器（外部 Flash 和 RAM）的加解密选项。详细信息请参见章节 *34 片外存储器加密与解密 (XTS_AES)*。

21.2.1.2 HP 外设时钟配置与电源控制

以下寄存器用于控制对应外设中存储器的电源信号：

- RSA：HP_SYSTEM_HP_RSA_PD_CTRL_REG
- ECC：HP_SYSTEM_HP_ECC_PD_CTRL_REG
- UART：HP_SYSTEM_HP_UART_PD_CTRL_REG

寄存器 HP_SYSTEM_HP_PERI_MEM_CLK_FORCE_ON_REG 中的以下字段用于控制对应外设中存储器的时钟门控信号：

- GDMA：HP_SYSTEM_GDMA_MEM_CLK_FORCE_ON
- 比特调节器 (BitScrambler)：
  - TX 存储器：HP_SYSTEM_BITSCRAMBLER_TX_MEM_CLK_FORCE_ON
  - RX 存储器：HP_SYSTEM_BITSCRAMBLER_RX_MEM_CLK_FORCE_ON
- RMT：HP_SYSTEM_RMT_MEM_CLK_FORCE_ON

21.2.1.3 HP Cache 时钟与复位配置

ESP32-P4 允许软件直接控制 HP cache 的时钟与复位信号。

寄存器 HP_SYSTEM_HP_CACHE_CLK_CONFIG_REG 中的以下字段用于控制不同 HP cache 的时钟信号，将相应字段写为 1 可使能对应 cache 的时钟。

- L1 ichache 0：HP_SYSTEM_L1_IO_CACHE_CLK_ON

- L1 ichache 1：HP_SYSTEM_L1_I1_CACHE_CLK_ON
- L1 dcache：HP_SYSTEM_L1_D_CACHE_CLK_ON
- L2 cache：HP_SYSTEM_L2_CACHE_CLK_ON

寄存器 HP_SYSTEM_CACHE_RESET_CONFIG_REG 中的以下字段用于控制不同 HP cache 的复位信号，可通过向相应字段先写 1 后写 0 实现复位。

- L1 ichache 0：HP_SYSTEM_L1_I0_CACHE_RESET
- L1 ichache 1：HP_SYSTEM_L1_I1_CACHE_RESET
- L1 dcache：HP_SYSTEM_L1_D_CACHE_RESET

21.2.1.4 HP SPM 与 HP L2MEM 时钟配置

以下寄存器字段用于控制 HP SPM（暂存内存）与 HP L2MEM 的时钟信号，将相应字段写为 1 可强制使能对应存储器的时钟。

- HP SPM：HP_SYSTEM_HP_SPM_CLK_FORCE_ON
- L2MEM：HP_SYSTEM_L2_MEM_CLK_FORCE_ON
- L2ROM：HP_SYSTEM_L2_ROM_CLK_FORCE_ON

21.2.1.5 HP SPM 奇偶校验配置

ESP32-P4 在 HP SPM 中实现了奇偶校验功能，用于检测单比特内存翻转错误。

该功能可通过以下寄存器进行配置与控制：

- HP_SYSTEM_HP_SPM_PARITY_CHECK_CTRL_REG：写 1 可使能 HP SPM 的奇偶校验。
- HP_SYSTEM_HP_SPM_INIT_REG：在使能奇偶校验前配置 HP SPM 的初始化。
- 中断（HP_SPM_PARITY_ERR_INT）相关寄存器：
  - HP_SYSTEM_HP_SPM_INT_RAW_REG：HP_SPM_PARITY_ERR_INT 的原始中断状态
  - HP_SYSTEM_HP_SPM_INT_ST_REG：HP_SPM_PARITY_ERR_INT 的屏蔽中断状态
  - HP_SYSTEM_HP_SPM_INT_ENA_REG：写 1 可使能 HP_SPM_PARITY_ERR_INT
  - HP_SYSTEM_HP_SPM_INT_CLR_REG：写 1 可清除 HP_SPM_PARITY_ERR_INT
- HP_SYSTEM_HP_SPM_PARITY_INT_RECORD_REG：记录 HP_SPM_PARITY_ERR_INT 发生时的地址。
- HP_SYSTEM_HP_SPM_ERR_RESP_CTRL_REG：写 1 后，HP SPM 会将奇偶校验错误报告给 HP CPU 并触发异常。

详细配置步骤请参考章节 9 *系统和存储器*。

21.2.1.6 HP L2MEM ECC 校验配置

ESP32-P4 在 HP L2MEM（包括 L2RAM 和缓存）中实现了 ECC（Error Correction Code，纠错码）校验功能，用于修正单比特内存翻转错误。

该功能可通过以下寄存器进行配置与控制：

- HP_SYSTEM_L2_MEM_L2_RAM_ECC_REG 和 HP_SYSTEM_L2_MEM_L2_CACHE_ECC_REG：分别写 1 以使能 L2MEM 和缓存的 ECC 校验。
- HP_SYSTEM_L2_MEM_REFRESH_REG：在使能 ECC 校验前配置 HP L2MEM 的初始化。
- 中断（L2_MEM_ECC_ERR_INT）相关寄存器：
  - HP_SYSTEM_L2_MEM_INT_RAW_REG：L2_MEM_ECC_ERR_INT 的原始中断状态
  - HP_SYSTEM_L2_MEM_INT_ST_REG：L2_MEM_ECC_ERR_INT 的屏蔽中断状态
  - HP_SYSTEM_L2_MEM_INT_ENA_REG：写 1 可使能 L2_MEM_ECC_ERR_INT
  - HP_SYSTEM_L2_MEM_INT_CLR_REG：写 1 可清除 L2_MEM_ECC_ERR_INT
- HP_SYSTEM_L2_MEM_INT_RECORD1_REG：记录 L2_MEM_ECC_ERR_INT 发生时的信息。
- HP_SYSTEM_L2_MEM_ERR_RESP_CTRL_REG：写 1 后，HP L2MEM 会将 ECC 错误报告给 HP CPU 并触发异常。

详细配置步骤请参考章节 9 *系统和存储器*。

21.2.1.7 比特调节器 (BitScrambler) 配置

ESP32-P4 有部分外设支持 DMA，但有时需要比特调节器以修改传输数据的格式，否则可能出现问题。

通过配置 HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG，可将此类外设分别连接到比特调节器的 RX 和 TX 通道。

详细配置步骤请参考章节 62 *比特调节器*。

有关支持 DMA 的外设列表，请参考章节 11 *复位和时钟*。

21.2.1.8 以太网 MAC 控制

ESP32-P4 可通过以太网 MAC（媒体访问控制器）按照 IEEE 802.3 标准进行数据收发。

相关控制与配置寄存器如下：

- HP_SYSTEM_GMAC_CTRL0_REG
- HP_SYSTEM_GMAC_CTRL1_REG
- HP_SYSTEM_GMAC_CTRL2_REG

详细信息请参考章节 55 *以太网介质访问控制器 (EMAC)*。

21.2.1.9 USB 2.0 高速 OTG 控制

ESP32-P4 搭载了 USB 2.0 高速 OTG 外设（OTG_HS），可通过 HP_SYSTEM_USBOTG20_CTRL_REG 进行控制。

寄存器 HP_SYSTEM_USB_OTGHS_PHY_ST_REG 表示 USB OTGHS PHY 状态。

详细信息请参考章节 52 *USB 2.0 高速 OTG*。

21.2.1.10 CPU 控制与记录

ESP32-P4 可通过以下系统寄存器配置 CPU 调试功能并记录其状态信息：

- HP_SYSTEM_CPU_CORESTALLED_ST_REG：记录 CPU0 和 CPU1 的 core stalled 状态。
- HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_REG：写 1 以启用 HP CPU 与 LP CPU 之间的调试 runstall 功能。
- HP_SYSTEM_HP_CORE_DMACTIVE_LPCORE_REG：代表 LP CPU 调试模块的 dmactive 值。
- HP_SYSTEM_HP_CPU_WAKEUP_EVENT_REG：写 1 以唤醒 HP CPU0 或 HP CPU1。

21.2.1.11 HP 到 LP 中断与唤醒控制

HP 系统寄存器提供分组控制，用于将外设中断和唤醒事件从 HP 域路由至 LP CPU 和 PMU。

- 中断使能：HP_SYSTEM_HP2LP_INTR_GROUP0_EN_REG ~ HP_SYSTEM_HP2LP_INTR_GROUP3_EN_REG
- 中断状态：HP_SYSTEM_HP2LP_INTR_GROUP0_ST_REG ~ HP_SYSTEM_HP2LP_INTR_GROUP3_ST_REG
- 唤醒使能：HP_SYSTEM_HP2LP_WAKEUP_GROUP0_EN_REG ~ HP_SYSTEM_HP2LP_WAKEUP_GROUP3_EN_REG

详细信息请参考章节 2 高性能处理器。

21.2.1.12 HP GPIO 保持控制

ESP32-P4 的每个 GPIO 管脚都具备独立的保持功能，可通过以下寄存器进行控制：

- HP_SYSTEM_GPIO_O_HOLD_CTRL0_REG：控制 HP GPIO16 ~ HP GPIO47 的保持信号。
- HP_SYSTEM_GPIO_O_HOLD_CTRL1_REG：控制 GPIO48 ~ HP GPIO56 的保持信号。
- HP_SYSTEM_GPIO_DED_HOLD_CTRL_REG：低 6 位控制 Flash 管脚的保持信号，高 20 位控制 16 线 PSRAM 管脚的保持信号。

详细信息请参考章节 10 *GPIO 交换矩阵和 IO MUX*。

21.2.1.13 HP GPIO 输出控制

ESP32-P4 的每个 GPIO 管脚均可通过以下寄存器配置为直接输出所需数值（即迟滞功能）：

- HP_SYSTEM_GPIO_O_HYS_CTRL0_REG：配置是否使能 HP GPIO16 ~ HP GPIO47 的迟滞功能。
- HP_SYSTEM_GPIO_O_HYS_CTRL1_REG：配置是否使能 HP GPIO48 ~ HP GPIO56 的迟滞功能。

详细信息请参考章节 10 *GPIO 交换矩阵和 IO MUX* 中的 10.10 小节。

21.2.1.14 非法访问与未授权访问

当发生非法访问或未授权访问时，ESP32-P4 的总线将返回错误响应，并记录访问地址、访问方向以及主机 ID 等信息。

- 非法访问：访问未映射的地址空间。
- 未授权访问：访问已映射但无访问权限的地址空间。

相关信息将记录在以下寄存器中：

- HP CPU 和其他 AHB 主机（ICM_CPU_ADDRHOLE_INT）：
  - HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR_REG
  - HP_SYSTEM_ICM_CPU_ADDRHOLE_INFO_REG
- AXI 主机（ICM_SYS_ADDRHOLE_INT）：
  - HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR_REG
  - HP_SYSTEM_ICM_SYS_ADDRHOLE_INFO_REG

有关地址空间映射的详细信息，请参考章节 9 *系统和存储器*。

有关访问权限管理的详细信息，请参考章节 20 *权限控制 (PMS)*。

21.2.1.15 HP CPU 总线超时保护

ESP32-P4 支持对 HP CPU 的 IBUS、DBUS 和 AHB 总线进行超时保护。

当传输开始时，超时保护模块内部的计数器会在每个时钟周期内加一。

- 如果累计值未超过配置的超时阈值，并且从机已响应此次传输，则计数器将被清零。
- 如果累计值超过了配置的超时阈值，且从机仍未响应传输请求，则超时保护模块将接管总线，向 HP CPU 返回错误响应，同时触发中断。

相关的控制与配置寄存器说明如下。

HP CPU IBUS 超时保护

配置 HP_SYSTEM_HP_CORE_IBUS_TIMEOUT_REG 可使能超时保护功能，并设置 HP CPU 通过 IBUS 访问 HP SPM、L2MEM 和外部内存（EXT MEM）的超时阈值。

当超时发生时，将触发 HP_CORE*x*_IBUS_TIMEOUT_INT 中断。

HP CPU DBUS 超时保护

配置 HP_SYSTEM_HP_CORE_DBUS_TIMEOUT_REG 可使能超时保护功能，并设置 HP CPU 通过 DBUS 访问 HP SPM、L2MEM 和外部内存（EXT MEM）的超时阈值。

当超时发生时，将触发 HP_CORE*x*_DBUS_TIMEOUT_INT 中断。

HP CPU AHB 总线超时保护

配置 HP_SYSTEM_HP_CORE_AHB_TIMEOUT_REG 可使能超时保护功能，并设置 HP CPU 通过 AHB 总线访问 LP 外设和 HP 外设寄存器的超时阈值。

当超时发生时，将触发 HP_CORE*x*_AHB_TIMEOUT_INT 中断。

21.2.1.16 AXI 总线矩阵

ESP32-P4 的 AXI 总线矩阵连接多个符合 AXI 协议的主机和从机，可实现高效通信。

- 主机包括所有 AHB 主机（通过 AHB2AXI 桥接器连接的 master port1）、HP CPU 缓存（master port2）、DW GDMA_0（master port6）、DW GDMA_1（master port7）、AXI GDMA（master port9）、2DDMA（master port11）、H264_0（master port12）以及 H264_1（master port13）；
- 从机包括 L2MEM（slave port2）、Flash（slave port3）、PSRAM（slave port4）、DSI Host（slave port6）以及 CSI Host（slave port7）。

除 master port2 外，AXI 总线矩阵中所有 AXI 主端口均支持 QoS（服务质量）带宽调节功能。此外，AXI 总线矩阵还具备死锁超时保护机制，用于 slave 端口的仲裁优先级配置，以及用于 master 端口的 QoS 仲裁优先级配置。

死锁超时保护

ESP32-P4 的死锁超时保护功能始终启用。可通过 HP_SYSTEM_ICM_DLOCK_TIMEOUT_REG 配置 AXI 总线矩阵中所有主机访问 flash、PSRAM、DSI HOST 和 CSI HOST 等 AXI 从机时的超时阈值。

当发生超时时，AXI 总线矩阵会将传输方向、AXI ID、主机 ID 和从机 ID 记录在 HP_SYSTEM_ICM_DLOCK_STATUS_REG 中，并同时触发 ICM_DLOCK_INTR 中断。

从机仲裁优先级

通过 HP_SYSTEM_ICM_SLV_ARB_PRIORITY_REG 配置 AXI Matrix 中从机的读取数据和响应仲裁优先级。

数值越高，优先级越高。

主机 QoS 仲裁优先级

通过 HP_SYSTEM_ICM_MST_ARQOS_REG0_REG 和 HP_SYSTEM_ICM_MST_AWQOS_REG0_REG 配置 AXI Matrix 中主机的读取地址通道和写入地址通道的 QoS（服务质量）值。

QoS 值越高，仲裁时的优先级越高。

21.2.1.17 加速写

以下寄存器用于配置加速写 (Post Write) 模式：

- HP_SYSTEM_PERI1_APB_POSTW_EN_REG：使能外设之间 APB2APB 桥的加速写模式。
- HP_SYSTEM_APB_SYNC_POSTW_EN_REG：使能 DSI、CSI 和 ISP 中 APB2APB 桥的加速写模式。
- HP_SYSTEM_CPU_ICM_H2X_POST_WR_EN：使能 AHB2AXI 桥从 AHB 主机到 AXI 从机的加速写模式。

21.2.2 LP 系统寄存器

21.2.2.1 LP 定时器 Stall 控制

LP_SYSTEM_SYSTIMER_STALL_SEL 字段可选择 LP 定时器的 systimer_stall 信号源是来自 HP CPU0 或 HP CPU1：

- 0：来自 CPU0
- 1：来自 CPU1

21.2.2.2 聚焦离子束 (FIB) 控制

LP_SYSTEM_LP_FIB_SEL 字段可选择欠压检测器 (brownout detector) 和超级看门狗 (super watchdog) 的控制源是来自模拟系统或软件寄存器。

- Bit-1：选择欠压检测器的控制源。
  - 0：由软件寄存器控制
  - 1：由模拟系统控制
- Bit-2：选择超级看门狗的控制源。
  - 0：由软件寄存器控制
  - 1：由模拟系统控制

通过软件寄存器配置 LP_SYSTEM_DIG_FIB 来启用欠压检测器或超级看门狗。

- Bit-1：写 1 以启用欠压检测器。
- Bit-2：写 1 以启用超级看门狗。

读取 LP_SYSTEM_ANA_FIB 来查看欠压检测器或超级看门狗的 FIB 状态。

- Bit-1：欠压检测器的 FIB 状态
- Bit-2：超级看门狗的 FIB 状态

21.2.2.3 启动模式

配置 LP_SYSTEM_FORCE_DOWNLOAD_BOOT，可以强制将芯片从 SPI Boot 模式切换到 Joint Download Boot 模式。

21.2.2.4 LP CPU 控制

通过 LP_SYSTEM_SYS_CTRL_REG 配置 LP CPU。具体如下：

- LP_SYSTEM_LP_CPU_BOOT_ADDR：配置 LP CPU 的启动地址。
- LP_SYSTEM_LP_CPU_EXC_PC_REG：表示 LP CPU 的异常 PC 地址。
- LP_SYSTEM_LP_CPU_DBG_PC_REG：表示 LP CPU 的当前 PC 地址。
- LP_SYSTEM_LP_CORE_ERR_RESP_DIS：配置是否禁用 LP CPU 各总线的错误响应。
  - Bit-0：IBUS
  - Bit-1：DBUS

  - Bit-2：AHB 总线

- LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG：指示由 LP CPU 唤醒事件触发的 ETM 任务的状态。
- LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG_CLR：写 1 以清除 LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG。

21.2.2.5 复位控制

ESP32-P4 支持多种不同级别的复位类型，具体可通过系统寄存器进行控制，具体见下方各小节。

有关所有复位类型和复位源的完整列表，请参阅第 11 章 复位和时钟。

模拟和系统复位源旁路控制

LP_SYSTEM_ANA_RST_BYPASS：配置是否绕过以下复位源：

- 欠压检测器（Brown-out detector）
- 超级看门狗（Super watchdog）
- 电源毛刺检测器（Power glitch detector）

LP_SYSTEM_SYS_RST_BYPASS：配置是否绕过以下复位源：

- 软件复位
- HP 看门狗定时器（HP Watchdog timer）
- LP 看门狗定时器（LP Watchdog timer）
- eFuse

上电复位旁路控制

LP_SYSTEM_HP_PO_RSTN_BYPASS_CTRL：配置在上电复位期间是否绕过 HP 电源域的以下复位源。

- Bit-0：eFuse
- Bit-1：LP 看门狗定时器
- Bit-2：USB (JTAG)
- Bit-3：USB (UART)
- Bit-4：未使用
- Bit-5：未使用
- Bit-6：HP 看门狗定时器
- Bit-7：软件复位

LP_SYSTEM_HP_PO_CNNT_RSTN_BYPASS_CTRL：配置在上电复位期间是否绕过 HP CNNT 电源域的以下复位源。

- Bit-0：eFuse
- Bit-1：LP 看门狗定时器
- Bit-2：USB (JTAG)

- Bit-3：USB (UART)
- Bit-4：未使用
- Bit-5：未使用
- Bit-6：HP 看门狗定时器
- Bit-7：软件复位

有关 ESP32-P4 电源域的详细信息，请参阅章节 15 低功耗管理。

软件系统复位

通过配置 LP_SYSTEM_SYS_SW_RST 触发软件系统复位。

IOMUX 复位

配置 LP_SYSTEM_IO_MUX_RESET_DISABLE 禁用 HP IOMUX 复位。

21.2.2.6 唤醒控制

以下寄存器可控制一些 PMU 唤醒源的行为。

LP_SYSTEM_USBOTG20_IN_SUSPEND：配置是否启用来自 USB OTG2.0 的唤醒信号。

LP_SYSTEM_USBOTG20_WAKEUP_CLR：写 1 以清除来自 USB OTG2.0 的唤醒信号至 PMU。

LP_SYSTEM_USBOTG20_LS_MODE：配置 USB OTG2.0 是否工作在低速模式。

寄存器 LP_SYSTEM_HP_USB_OTGHS_PHY_CTRL_REG 用于控制 USB OTGHS PHY 的上拉和下拉信号。

21.2.2.7 Pad 保持控制

以下寄存器用于在睡眠或复位期间保持 pad 状态：

- LP_SYSTEM_PAD_RTC_HOLD_CTRL0_REG：控制 pad 0 ~ pad 31
- LP_SYSTEM_PAD_RTC_HOLD_CTRL1_REG：控制 pad 32 ~ pad 56
- LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL_REG：控制 flash 和 PSRAM pad

21.2.2.8 放电控制

寄存器 LP_SYSTEM_DISCHARGE_REG 用于控制 LDO flash 和 LP PUF 存储器的放电。

21.2.2.9 LPROM 和 LP SPM 时钟控制

LP_SYSTEM_LP_SPM_ROM_CLK_FORCE_ON：如果 LP CPU 时钟已开启，则设置此位以强制开启 LP SPM_ROM 时钟。

LP_SYSTEM_LP_SPM_RAM_CLK_FORCE_ON：设置此位以强制开启 LP SPM_RAM 时钟。

21.2.2.10 模拟电压比较器控制

模拟电压比较器*x* (*x* = 0, 1) 通过以下寄存器字段进行配置：

- LP_SYSTEM_XPD_COMP*x* (*x* = 0, 1)：配置是否启用模拟电压比较器*x*。

- LP_SYSTEM_MODE_COMP*x* (*x* = 0, 1)：配置比较模式：
  - 0：将主电压与外部参考电压进行比较
  - 1：将主电压与内部参考电压进行比较
- LP_SYSTEM_DREF_COMP*x* (*x* = 0, 1)：配置内部参考电压。
  - 可调节电压范围为 0 到 (0.7 * VDDPST_6) V。
  - 可调节步长为 0.1 * VDDPST_6 V。

详细信息请参阅章节 66 模拟电压比较器。

21.2.2.11 非法访问与未授权访问

当发生非法访问或未授权访问时，ESP32-P4 的总线将返回错误响应，并记录访问地址、访问方向以及主机 ID 等信息。

- 非法访问：访问未映射的地址空间。
- 未授权访问：访问已映射但无访问权限的地址空间。

相关信息将记录在以下寄存器中：

- LP CPU 的 IBUS 和 DBUS 访问（IDBUS_ADDRHOLE_INTR）：
  - LP_SYSTEM_IDBUS_ADDRHOLE_ADDR_REG
  - LP_SYSTEM_IDBUS_ADDRHOLE_INFO_REG
- LP CPU 的 AHB 总线或 HP CPU 访问（LP_ADDRHOLE_INTR）：
  - LP_SYSTEM_LP_ADDRHOLE_ADDR_REG
  - LP_SYSTEM_LP_ADDRHOLE_INFO_REG

有关地址空间映射的详细信息，请参考章节 9 系统和存储器。

有关访问权限管理的详细信息，请参考章节 20 权限控制 (PMS)。

21.2.2.12 LP CPU 总线超时保护

ESP32-P4 支持对 LP CPU 的 IBUS、DBUS 和 AHB 总线进行超时保护。

当传输开始时，超时保护模块内部的计数器会在每个时钟周期内加一。

- 如果累计值未超过配置的超时阈值，并且从机已响应此次传输，则计数器将被清零。
- 如果累计值超过了配置的超时阈值，且从机仍未响应传输请求，则超时保护模块将接管总线，向 LP CPU 返回错误响应，同时触发中断。

相关的控制与配置寄存器说明如下。

LP CPU IBUS 超时保护

配置 LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_REG 可使能超时保护功能，并设置 LP CPU 通过 IBUS 访问 LPROM 和 LP SPM 的超时阈值。当超时发生时，将触发 LP_CORE_IBUS_TIMEOUT_INTR 中断。

LP CPU DBUS 超时保护

配置 LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_REG 可使能超时保护功能，并设置 LP CPU 通过 DBUS 访问 LPROM 和 LP SPM 的超时阈值。

当超时发生时，将触发 LP_CORE_DBUS_TIMEOUT_INTR 中断。

LP CPU AHB 总线超时保护

配置 LP_SYSTEM_LP_CORE_AHB_TIMEOUT_REG 可使能超时保护功能，并设置 LP CPU 通过 AHB 总线访问 LP 外设、HP 外设和 CPU 外设的超时阈值。

当超时发生时，将触发 LP_CORE_AHB_TIMEOUT_INTR 中断。

21.2.2.13 随机数生成器控制

寄存器 LP_SYSTEM_RNG_DATA_REG 和 LP_SYSTEM_RNG_CFG_REG 用于配置和控制随机数生成器。

21.3 中断

ESP32-P4 的系统寄存器可以生成以下中断信号并发送给中断矩阵。

- HP_SYSREG_INTR
- SYS_ICM_INTR
- LP_SYSREG_INTR

这些中断信号由系统寄存器的内部中断源生成。表 21.3-1 列出了系统寄存器的内部中断源、中断触发条件、以及生成的中断信号。

表 21.3-1. 系统寄存器的内部中断源

内部中断源	触发条件	中断信号
CPU_INT_FROM_CPU0	软件中断。通过向 HP_SYSTEM_HP_CPU_INT_FROM_CPU_0_REG 写入 1 触发	HP_SYSREG_INTR
CPU_INT_FROM_CPU1	软件中断。通过向 HP_SYSTEM_HP_CPU_INT_FROM_CPU_1_REG 写入 1 触发	
CPU_INT_FROM_CPU2	软件中断。通过向 HP_SYSTEM_HP_CPU_INT_FROM_CPU_2_REG 写入 1 触发	
CPU_INT_FROM_CPU3	软件中断。通过向 HP_SYSTEM_HP_CPU_INT_FROM_CPU_3_REG 写入 1 触发	
L2_MEM_ERR_RESP_INT	当 L2MEM 发生错误响应时触发	
L2_MEM_EXCEED_ADDR_INT	当 L2MEM 超出其地址范围时触发	
L2_MEM_ECC_ERR_INT	当 L2MEM 中检测到并纠正了单比特翻转错误时触发	
HP_SPM_PARITY_ERR_INT	当 HP SPM 发生奇偶校验错误时触发	
CPU_ICM_H2X_BRESP_ERR_INT	在 AHB2AXI 的加速写模式中发生 BRESP 错误时触发	

HP_CORE1_DBUS_TIMEOUT_INT	当 HP CPU1 的 DBUS 发生超时时触发	
HP_CORE0_DBUS_TIMEOUT_INT	当 HP CPU0 的 DBUS 发生超时时触发	
HP_CORE1_IBUS_TIMEOUT_INT	当 HP CPU1 的 IBUS 发生超时时触发	
HP_CORE0_IBUS_TIMEOUT_INT	当 HP CPU0 的 IBUS 发生超时时触发	
HP_CORE1_AHB_TIMEOUT_INT	当 HP CPU1 的 AHB 总线发生超时时触发	
HP_CORE0_AHB_TIMEOUT_INT	当 HP CPU0 的 AHB 总线发生超时时触发	
ICM_CPU_ADDRHOLE_INT	当 HP CPU 和 AHB 矩阵发生非法/未授权访问时触发	SYS_ICM_INTR
ICM_SYS_ADDRHOLE_INT	当 HP AXI 矩阵发生非法/未授权访问时触发	
ICM_DLOCK_INT	当 AXI 矩阵中的任何 AXI 传输超时时触发	LP_SYSREG_INTR
SLOW_CLK_TICK_INT	ETM 时钟事件中断。当 LP CPU 未处于睡眠状态时，以 LP_SLOW_CLK 的频率周期性触发。可以通过 LP_CLKRST_ETM_EVENT_TICK_EN 禁用此事件源。	
ETM_TASK_ULP_INT	来自 ULP_TASK_INT_CPU。详细信息请参阅章节 14 事件任务矩阵 *(ETM)*	
LP_CORE_DBUS_TIMEOUT_INT	当 LP CPU 的 DBUS 发生超时时触发	
LP_CORE_IBUS_TIMEOUT_INT	当 LP CPU 的 IBUS 发生超时时触发	
LP_CORE_AHB_TIMEOUT_INT	当 LP CPU 的 AHB 总线发生超时时触发	
IDBUS_ADDRHOLE_INT	当 LP CPU 的 IBUS 或 DBUS 发生非法/未授权访问时触发	
LP_ADDRHOLE_INT	当 LP AHB 矩阵发生非法/未授权访问时触发	

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

21.4 寄存器列表

21.4.1 HP 寄存器列表

本小节的所有地址均为相对于 HP 系统寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

名称	描述	地址	访问
版本控制寄存器			
HP_SYSTEM_VER_DATE_REG	版本控制	0x0000	R/W
HP_SYSTEM_CLK_EN_REG	时钟门控	0x0004	R/W
中断寄存器			
HP_SYSTEM_HP_CPU_INT_FROM_CPU_0_REG	软件中断 0	0x0010	R/W
HP_SYSTEM_HP_CPU_INT_FROM_CPU_1_REG	软件中断 1	0x0014	R/W
HP_SYSTEM_HP_CPU_INT_FROM_CPU_2_REG	软件中断 2	0x0018	R/W
HP_SYSTEM_HP_CPU_INT_FROM_CPU_3_REG	软件中断 3	0x001C	R/W
HP_SYSTEM_HP_SPM_INT_RAW_REG	HP SPM 原始中断状态	0x00FC	R/WTC/SS
HP_SYSTEM_HP_SPM_INT_ST_REG	HP SPM 屏蔽中断状态	0x0100	RO
HP_SYSTEM_HP_SPM_INT_ENA_REG	HP SPM 中断使能	0x0104	R/W
HP_SYSTEM_HP_SPM_INT_CLR_REG	HP SPM 中断清除	0x0108	WT
HP_SYSTEM_HP_CORE_TIMEOUT_INT_RAW_REG	HP CPU 超时原始中断状态	0x01A8	R/WTC/SS
HP_SYSTEM_HP_CORE_TIMEOUT_INT_ST_REG	HP CPU 超时屏蔽中断状态	0x01AC	RO
HP_SYSTEM_HP_CORE_TIMEOUT_INT_ENA_REG	HP CPU 超时中断使能	0x01B0	R/W
HP_SYSTEM_HP_CORE_TIMEOUT_INT_CLR_REG	HP CPU 超时中断清除	0x01B4	WT
HP_SYSTEM_HP_CACHE_CLK_CONFIG_REG	Cache 时钟配置	0x0020	R/W
HP_SYSTEM_CACHE_RESET_CONFIG_REG	Cache 重置配置	0x0024	R/W
HP_SYSTEM_HP_SPM_RAM_PWR_CTRL0_REG	HP SPM 时钟强制开启	0x003C	R/W
HP_SYSTEM_L2_ROM_PWR_CTRL0_REG	L2ROM 时钟强制开启	0x0040	R/W
HP_SYSTEM_HP_L2_MEM_RAM_PWR_CTRL0_REG	L2MEM 时钟强制开启	0x0060	R/W
HP_SYSTEM_CPU_CORESTALLED_ST_REG	HP CPU 停滞状态	0x0064	RO
HP_SYSTEM_CRYPTO_CTRL_REG	加密控制	0x0070	R/W
HP_SYSTEM_GPIO_O_HOLD_CTRL0_REG	HP GPIO 保持控制 0	0x0074	R/W
HP_SYSTEM_GPIO_O_HOLD_CTRL1_REG	HP GPIO 保持控制 1	0x0078	R/W
HP_SYSTEM_L2_MEM_INT_RAW_REG	HP L2MEM 原始中断状态	0x009C	R/WTC/SS
HP_SYSTEM_L2_MEM_INT_ST_REG	HP L2MEM 屏蔽中断状态	0x00A0	RO
HP_SYSTEM_L2_MEM_INT_ENA_REG	HP L2MEM 中断使能	0x00A4	R/W
HP_SYSTEM_L2_MEM_INT_CLR_REG	HP L2MEM 中断清除	0x00A8	WT
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_RAW_REG	AHB 到 AXI BRESP 错误原始中断状态	0x0188	R/WTC/SS
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ST_REG	AHB 到 AXI BRESP 错误中断状态	0x018C	RO

名称	描述	地址	访问
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ENA_REG	AHB 到 AXI BRESP 错误中断使能	0x0190	R/W
HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_CLR_REG	AHB 到 AXI BRESP 错误中断清除	0x0194	WT
HP_SYSTEM_L2_MEM_L2_RAM_ECC_REG	HP L2MEM ECC 使能	0x00AC	R/W
HP_SYSTEM_L2_MEM_INT_RECORD0_REG	HP L2MEM 中断记录 0	0x00B0	RO
HP_SYSTEM_L2_MEM_INT_RECORD1_REG	HP L2MEM 中断记录 1	0x00B4	RO
HP_SYSTEM_L2_MEM_L2_CACHE_ECC_REG	L2 缓存 ECC 使能	0x00C4	R/W
HP_SYSTEM_HP_L1CACHE_BUS0_ID_REG	L1 缓存总线 0 ID	0x00C8	R/W
HP_SYSTEM_HP_L1CACHE_BUS1_ID_REG	L1 缓存总线 1 ID	0x00CC	R/W
HP_SYSTEM_GPIO_DED_HOLD_CTRL_REG	HP GPIO 保持控制	0x00F0	R/W
HP_SYSTEM_HP_USB20OTG_MEM_CTRL_REG	USB OTG FS 时钟强制开启	0x00F8	R/W
HP_SYSTEM_HP_SPM_PARITY_INT_RECORD_REG	HP SPM 奇偶校验中断状态	0x010C	RO
HP_SYSTEM_HP_L1_CACHE_PWR_CTRL_REG	HP L1 缓存内存强制开启	0x0110	R/W
HP_SYSTEM_HP_L2_CACHE_PWR_CTRL_REG	HP L2 缓存内存强制开启	0x0114	R/W
HP_SYSTEM_HP_CPU_WAITI_CONF_REG	CPU_WAITI 配置	0x0118	R/W
HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_REG	CPU 调试运行暂停配置	0x011C	R/W
HP_SYSTEM_HP_CORE_AHB_TIMEOUT_REG	HP CPU AHB 总线超时	0x0120	R/W
HP_SYSTEM_HP_CORE_IBUS_TIMEOUT_REG	HP CPU IBUS 超时	0x0124	R/W
HP_SYSTEM_HP_CORE_DBUS_TIMEOUT_REG	HP CPU DBUS 超时	0x0128	R/W
HP_SYSTEM_HP_ICM_CPU_H2X_CFG_REG	AHB 到 AXI 桥配置	0x0138	可变
HP_SYSTEM_PERI1_APB_POSTW_EN_REG	HP PERI1 APB 后写使能	0x013C	R/W
HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG	比特接收器外设选择	0x0140	R/W
HP_SYSTEM_APB_SYNC_POSTW_EN_REG	HP PERI0 APB 加速写使能	0x0144	R/W
HP_SYSTEM_HP_RSA_PD_CTRL_REG	RSA 内存 PD 控制	0x01D0	R/W
HP_SYSTEM_HP_ECC_PD_CTRL_REG	ECC 内存 PD 控制	0x01D4	R/W
HP_SYSTEM_HP_RNG_CFG_REG	RNG 配置	0x01D8	R/W
HP_SYSTEM_HP_UART_PD_CTRL_REG	内存 PD 控制	0x01DC	R/W
HP_SYSTEM_HP_PERI_MEM_CLK_FORCE_ON_REG	HP PERI 内存时钟强制开启	0x01E0	R/W
HP_SYSTEM_HP_CPU_WAKEUP_EVENT_REG	CPU 唤醒事件控制	0x01EC	R/W
HP_SYSTEM_HP2LP_INTR_GROUP0_EN_REG	HP 到 LP 中断使能组 0	0x01F0	R/W
HP_SYSTEM_HP2LP_INTR_GROUP1_EN_REG	HP 到 LP 中断使能组 1	0x01F4	R/W
HP_SYSTEM_HP2LP_INTR_GROUP2_EN_REG	HP 到 LP 中断使能组 2	0x01F8	R/W
HP_SYSTEM_HP2LP_INTR_GROUP3_EN_REG	HP 到 LP 中断使能组 3	0x01FC	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP0_EN_REG	HP 到 LP 唤醒使能组 0	0x0210	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP1_EN_REG	HP 到 LP 唤醒使能组 1	0x0214	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP2_EN_REG	HP 到 LP 唤醒使能组 2	0x0218	R/W
HP_SYSTEM_HP2LP_WAKEUP_GROUP3_EN_REG	HP 到 LP 唤醒使能组 3	0x021C	R/W
HP_SYSTEM_GDMA_CTRL_REG	DW-GDMA 控制	0x0148	R/W
HP_SYSTEM_GMAC_CTRL0_REG	GMAC 控制 0	0x014C	可变
HP_SYSTEM_GMAC_CTRL1_REG	GMAC 控制 1	0x0150	RO
HP_SYSTEM_GMAC_CTRL2_REG	GMAC 控制 2	0x0154	RO
HP_SYSTEM_USBOTG20_CTRL_REG	USB OTGHS 控制	0x015C	可变
HP_SYSTEM_HP_SPM_ERR_RESP_CTRL_REG	HP SPM 错误响应使能	0x0160	R/W
HP_SYSTEM_L2_MEM_REFRESH_REG	HP SPM 刷新	0x0164	可变

名称	描述	地址	访问
HP_SYSTEM_HP_SPM_INIT_REG	HP SPM 初始化	0x0168	可变
HP_SYSTEM_HP_SPM_PARITY_CHECK_CTRL_REG	HP SPM 奇偶校验使能	0x016C	R/W
HP_SYSTEM_L2_MEM_ERR_RESP_CTRL_REG	HP L2MEM 错误响应	0x0198	R/W
HP_SYSTEM_L2_MEM_AHB_BUFFER_CTRL_REG	HP L2MEM AHB 缓冲使能	0x019C	R/W
HP_SYSTEM_HP_CORE_DMACTIVE_LPCORE_REG	HP CPU 调试模块激活	0x01A0	RO
控制寄存器			
HP_SYSTEM_HP_CORE_ERR_RESP_DIS_REG	HP CPU 错误响应屏蔽	0x01A4	R/W
HP_SYSTEM_GPIO_O_HYS_CTRL0_REG	HP GPIO 迟滞控制 0	0x01C0	R/W
HP_SYSTEM_GPIO_O_HYS_CTRL1_REG	HP GPIO 迟滞控制 1	0x01C4	R/W
状态寄存器			
HP_SYSTEM_USB_OTGHS_PHY_ST_REG	USB OTGHS PHY 状态	0x01E8	RO
HP_SYSTEM_HP2LP_INTR_GROUP0_ST_REG	HP 到 LP 中断状态组 0	0x0200	RO
HP_SYSTEM_HP2LP_INTR_GROUP1_ST_REG	HP 到 LP 中断状态组 1	0x0204	RO
HP_SYSTEM_HP2LP_INTR_GROUP2_ST_REG	HP 到 LP 中断状态组 2	0x0208	RO
HP_SYSTEM_HP2LP_INTR_GROUP3_ST_REG	HP 到 LP 中断状态组 3	0x020C	RO

21.4.2 ICM 寄存器列表

本小节的所有地址均为相对于 AXI ICM 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

名称	描述	地址	访问
版本寄存器			
HP_SYSTEM_ICM_VER_DATE_REG	版本控制	0x0000	R/W
时钟门控寄存器			
HP_SYSTEM_ICM_CLK_EN_REG	时钟门控	0x0004	R/W
状态寄存器			
HP_SYSTEM_ICM_DLOCK_STATUS_REG	ICM deadlock 状态	0x0008	RO
HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR_REG	系统 ADDRHOLE 地址	0x0038	RO
HP_SYSTEM_ICM_SYS_ADDRHOLE_INFO_REG	系统 ADDRHOLE 信息	0x003C	RO
HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR_REG	CPU ADDRHOLE 地址	0x0040	RO
HP_SYSTEM_ICM_CPU_ADDRHOLE_INFO_REG	CPU ADDRHOLE 信息	0x0044	RO
中断寄存器			
HP_SYSTEM_ICM_INT_RAW_REG	ICM 中断原始状态	0x000C	R/WTC /SS
HP_SYSTEM_ICM_INT_ST_REG	ICM 屏蔽中断状态	0x0010	RO
HP_SYSTEM_ICM_INT_ENA_REG	ICM 中断使能	0x0014	R/W
HP_SYSTEM_ICM_INT_CLR_REG	ICM 中断清除	0x0018	WT
配置寄存器			
HP_SYSTEM_ICM_SLV_ARB_PRIORITY_REG	从机仲裁优先级	0x0024	R/W
HP_SYSTEM_ICM_MST_ARQOS_REG0_REG	ARQoS 配置	0x0028	R/W
HP_SYSTEM_ICM_MST_AWQOS_REG0_REG	AWQoS 配置	0x0030	R/W

名称	描述	地址	访问
HP_SYSTEM_ICM_DLOCK_TIMEOUT_REG	ICM deadlock 超时	0x0048	R/W

21.4.3 LP 寄存器列表

本小节的所有地址均为相对于 LP 系统寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

名称	描述	地址	访问
配置 & 控制寄存器			
LP_SYSTEM_LP_SYS_VER_DATE_REG	版本控制	0x0000	R/W
LP_SYSTEM_SYS_CTRL_REG	系统控制	0x0008	varies
LP_SYSTEM_LP_CLK_CTRL_REG	时钟控制	0x000C	R/W
LP_SYSTEM_LP_RST_CTRL_REG	复位控制	0x0010	R/W
LP_SYSTEM_LP_CORE_BOOT_ADDR_REG	LP CPU 启动地址	0x0018	R/W
LP_SYSTEM_LP_SPM_PWR_CTRL_REG	LPROM 和 LP SPM 时钟控制	0x0024	R/W
LP_SYSTEM_USB_CTRL_REG	USB 控制	0x0100	varies
LP_SYSTEM_PAD_COMP0_REG	Pad 补偿 0	0x0148	R/W
LP_SYSTEM_PAD_COMP1_REG	Pad 补偿 1	0x014C	R/W
LP_SYSTEM_HP_MEM_AUX_CTRL_REG	高性能内存辅助控制	0x0180	R/W
LP_SYSTEM_LP_MEM_AUX_CTRL_REG	低功耗内存辅助控制	0x0184	R/W
LP_SYSTEM_HP_ROM_AUX_CTRL_REG	高性能 ROM 辅助控制	0x0188	R/W
LP_SYSTEM_LP_ROM_AUX_CTRL_REG	低功耗 ROM 辅助控制	0x018C	R/W
LP_SYSTEM_HP_POR_RST_BYPASS_CTRL_REG	上电复位控制	0x01A0	R/W
LP_SYSTEM_LP_CORE_AHB_TIMEOUT_REG	CPU AHB 总线超时	0x01B0	R/W
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_REG	CPU IBUS 超时	0x01B4	R/W
LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_REG	CPU DBUS 超时	0x01B8	R/W
LP_SYSTEM_LP_CORE_ERR_RESP_DIS_REG	CPU 错误响应控制	0x01BC	R/W
LP_SYSTEM_RNG_CFG_REG	RNG 配置	0x01C0	varies
LP_SYSTEM_PAD_RTC_HOLD_CTRL0_REG	Pad 保持控制 0	0x01C4	R/W
LP_SYSTEM_PAD_RTC_HOLD_CTRL1_REG	Pad 保持控制 1	0x01C8	R/W
LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL_REG	专用 Pad 保持控制	0x01CC	R/W
LP_SYSTEM_DISCHARGE_REG	PUF 存储器与 LDO flash 放电控制	0x0200	R/W
LP_SYSTEM_HP_USB_OTGHS_PHY_CTRL_REG	USB OTGHS PHY 控制	0x0204	R/W
状态寄存器			
LP_SYSTEM_LP_ADDRHOLE_ADDR_REG	AHB 总线 ADDRHOLE	0x0168	RO
LP_SYSTEM_LP_ADDRHOLE_INFO_REG	AHB 总线 ADDRHOLE 信息	0x016C	RO
LP_SYSTEM_LP_CPU_DBG_PC_REG	LP CPU 调试 PC	0x0190	RO
LP_SYSTEM_LP_CPU_EXC_PC_REG	LP CPU 异常 PC	0x0194	RO
LP_SYSTEM_IDBUS_ADDRHOLE_ADDR_REG	CPU IBUS 和 DBUS 地址 ADDRHOLE	0x0198	RO
LP_SYSTEM_IDBUS_ADDRHOLE_INFO_REG	CPU IBUS 和 DBUS ADDRHOLE 信息	0x019C	RO
LP_SYSTEM_RNG_DATA_REG	RNG 数据	0x01A4	RO
中断寄存器			

名称	描述	地址	访问
LP_SYSTEM_INT_RAW_REG	原始中断状态	0x0170	R/SS / WTC
LP_SYSTEM_INT_ST_REG	屏蔽中断状态	0x0174	RO
LP_SYSTEM_INT_ENA_REG	中断使能	0x0178	R/W
LP_SYSTEM_INT_CLR_REG	中断清除	0x017C	WT

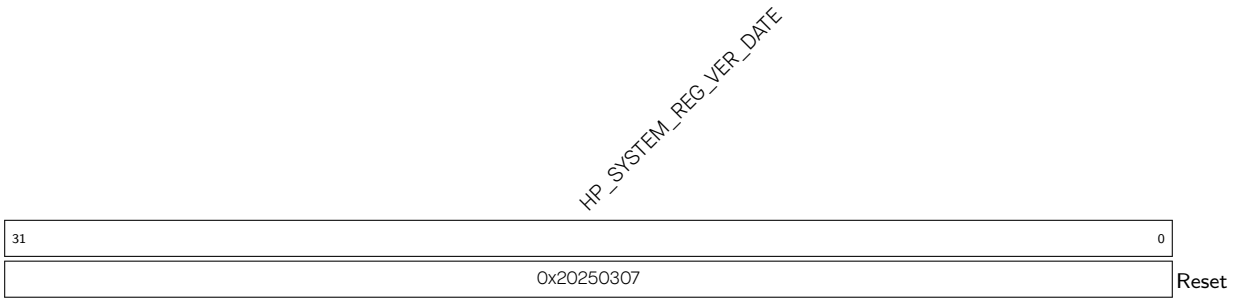
21.5 寄存器

21.5.1 HP 寄存器

本小节的所有地址均为相对于 HP 系统寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

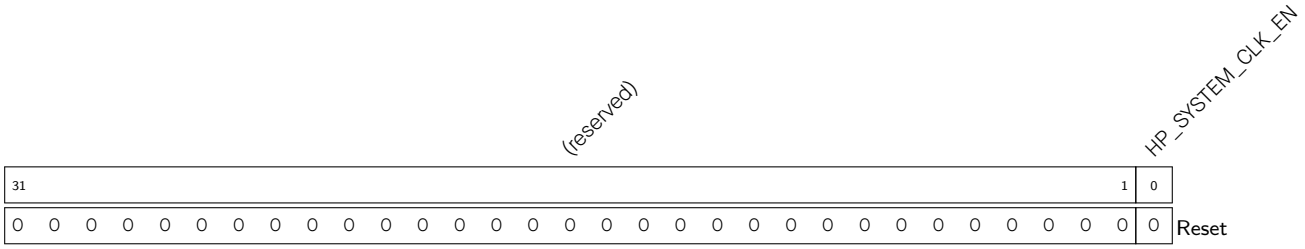
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 21.1. HP_SYSTEM_VER_DATE_REG (0x0000)



HP_SYSTEM_REG_VER_DATE 版本控制寄存器。(R/W)

Register 21.2. HP_SYSTEM_CLK_EN_REG (0x0004)



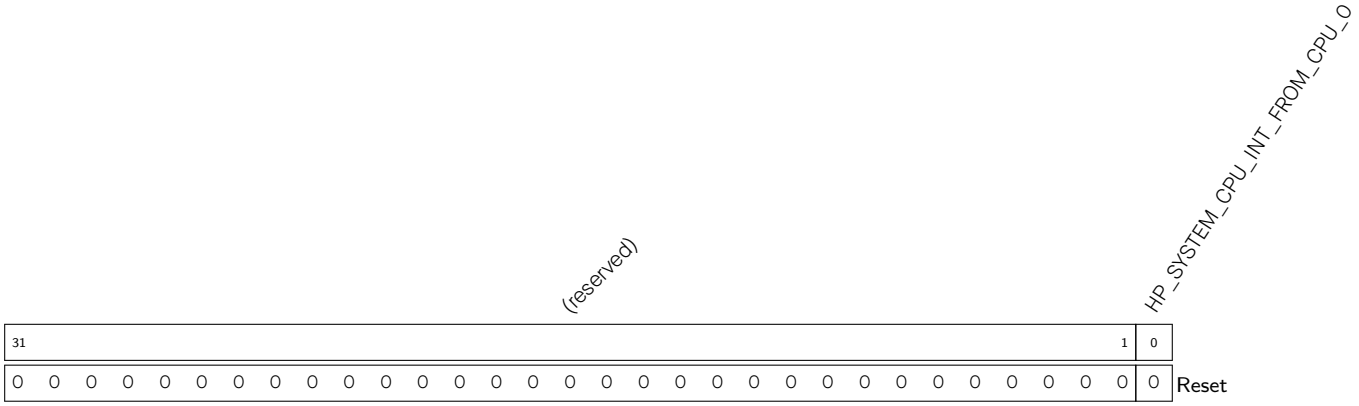
HP_SYSTEM_CLK_EN 配置是否启用系统时钟。

0：禁用

1：启用

(R/W)

Register 21.3. HP_SYSTEM_HP_CPU_INT_FROM_CPU_0_REG (0x0010)



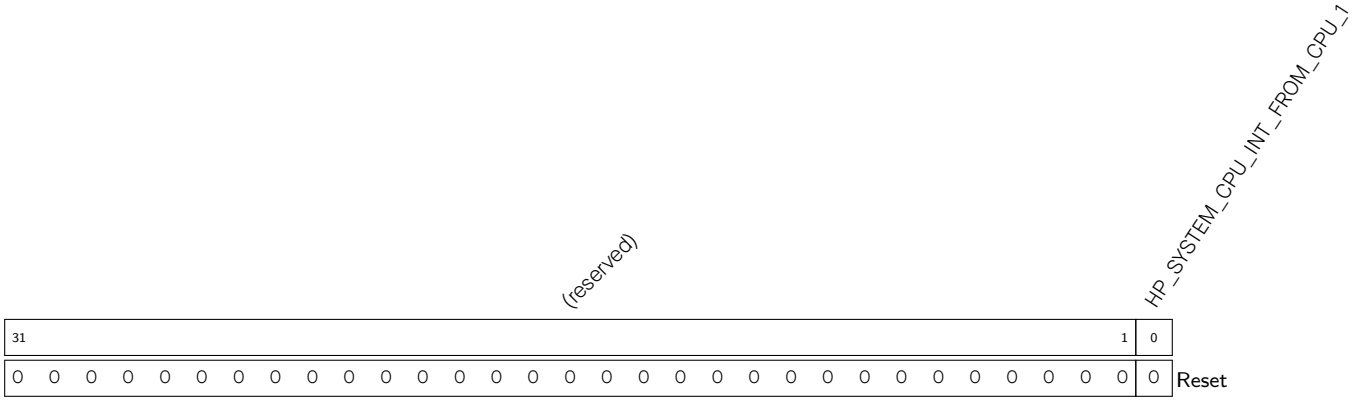
HP_SYSTEM_CPU_INT_FROM_CPU_0 配置是否生成 CPU_INT_FROM_CPU0。

0：无作用

1：生成

(R/W)

Register 21.4. HP_SYSTEM_HP_CPU_INT_FROM_CPU_1_REG (0x0014)



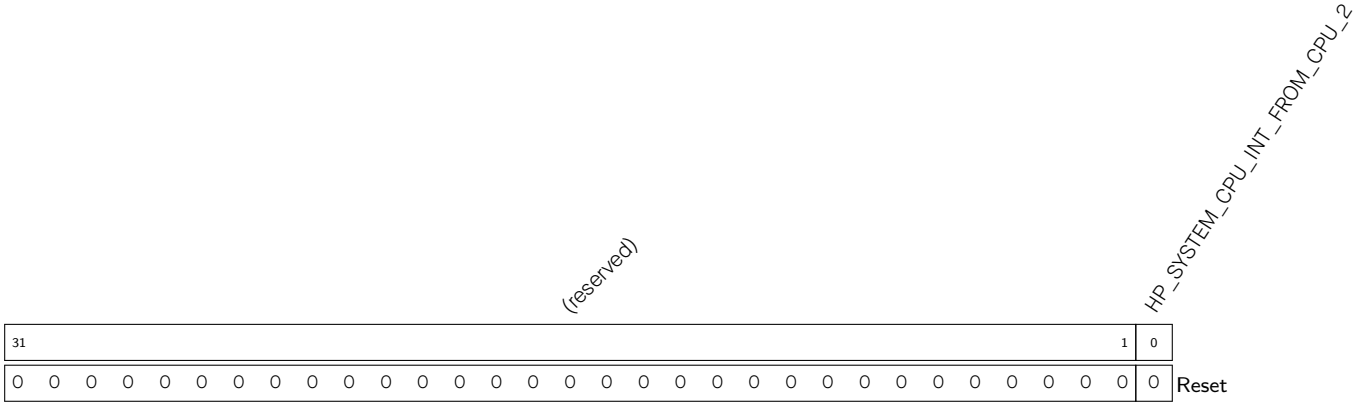
HP_SYSTEM_CPU_INT_FROM_CPU_1 配置是否生成 CPU_INT_FROM_CPU1。

0：无作用

1：生成

(R/W)

Register 21.5. HP_SYSTEM_HP_CPU_INT_FROM_CPU_2_REG (0x0018)



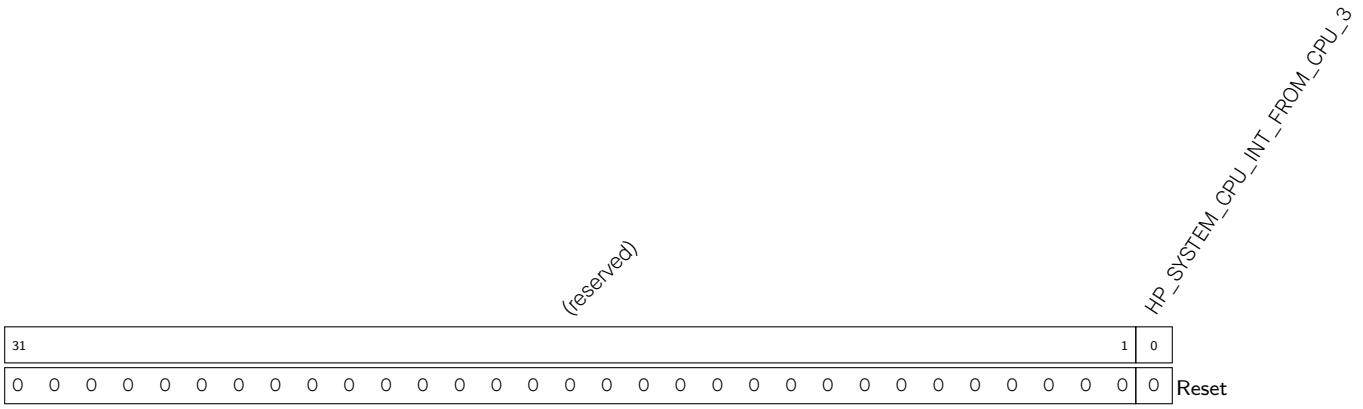
HP_SYSTEM_CPU_INT_FROM_CPU_2 配置是否生成 CPU_INT_FROM_CPU2。

0：无作用

1：生成

(R/W)

Register 21.6. HP_SYSTEM_HP_CPU_INT_FROM_CPU_3_REG (0x001C)



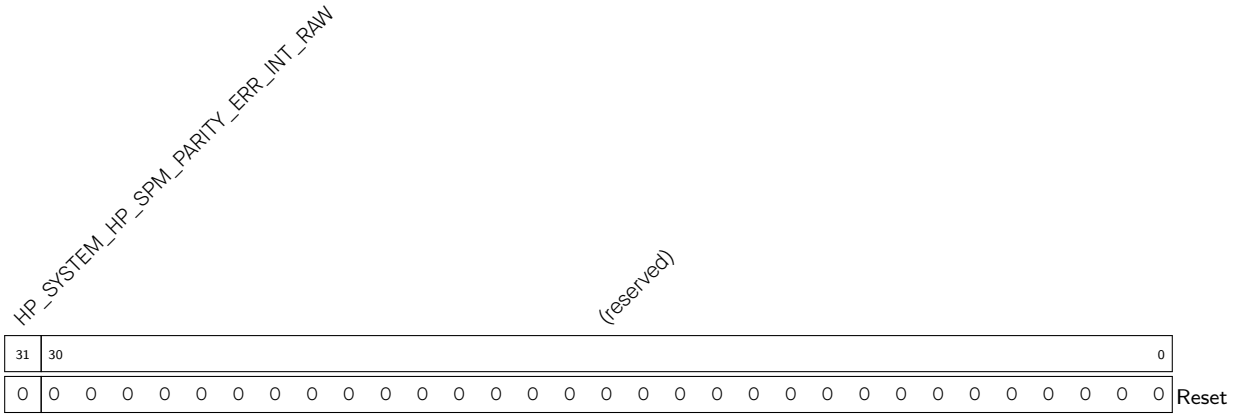
HP_SYSTEM_CPU_INT_FROM_CPU_3 配置是否生成 CPU_INT_FROM_CPU3。

0：无作用

1：生成

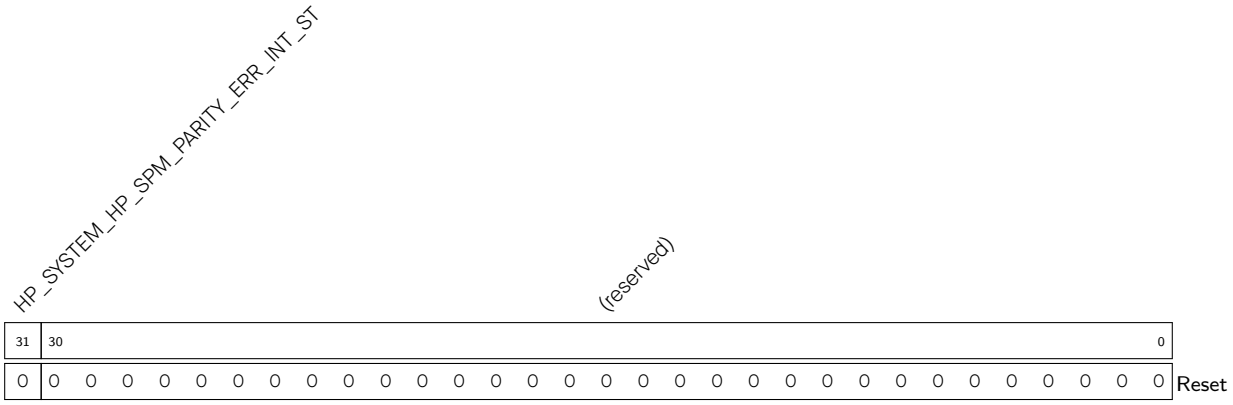
(R/W)

Register 21.7. HP_SYSTEM_HP_SPM_INT_RAW_REG (0x00FC)



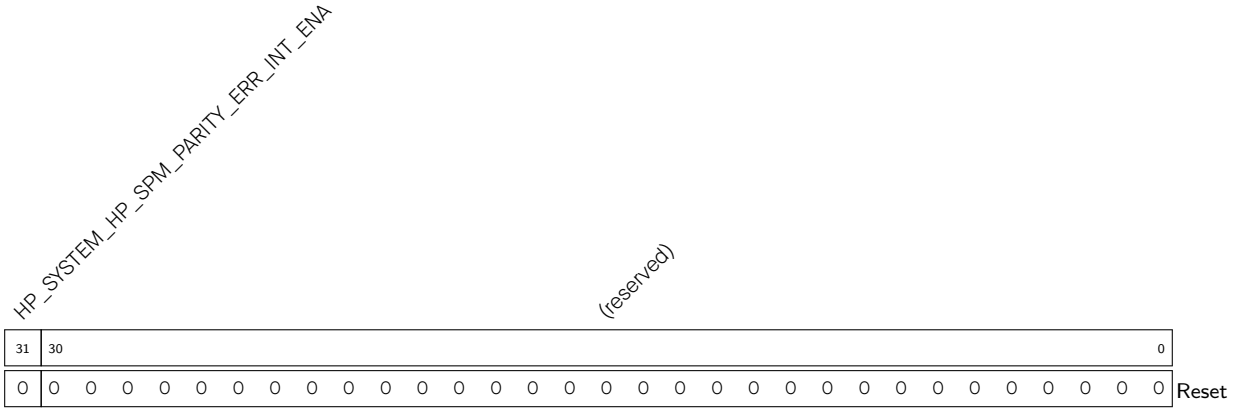
HP_SYSTEM_HP_SPM_PARITY_ERR_INT_RAW HP_SPM_PARITY_ERR_INT 的原始中断状态，当发生奇偶校验错误时触发。(R/WTC/SS)

Register 21.8. HP_SYSTEM_HP_SPM_INT_ST_REG (0x0100)



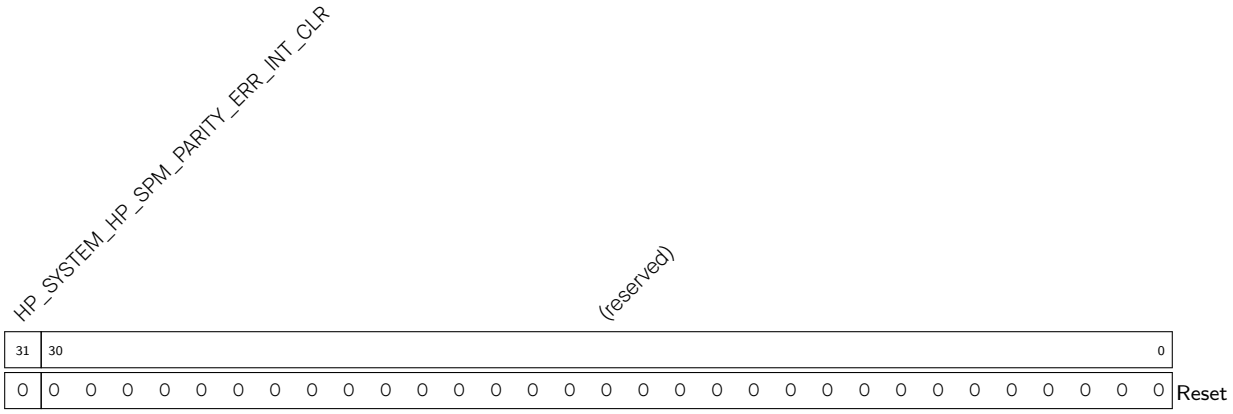
HP_SYSTEM_HP_SPM_PARITY_ERR_INT_ST HP_SPM_PARITY_ERR_INT 的屏蔽中断状态。(RO)

Register 21.9. HP_SYSTEM_HP_SPM_INT_ENA_REG (0x0104)



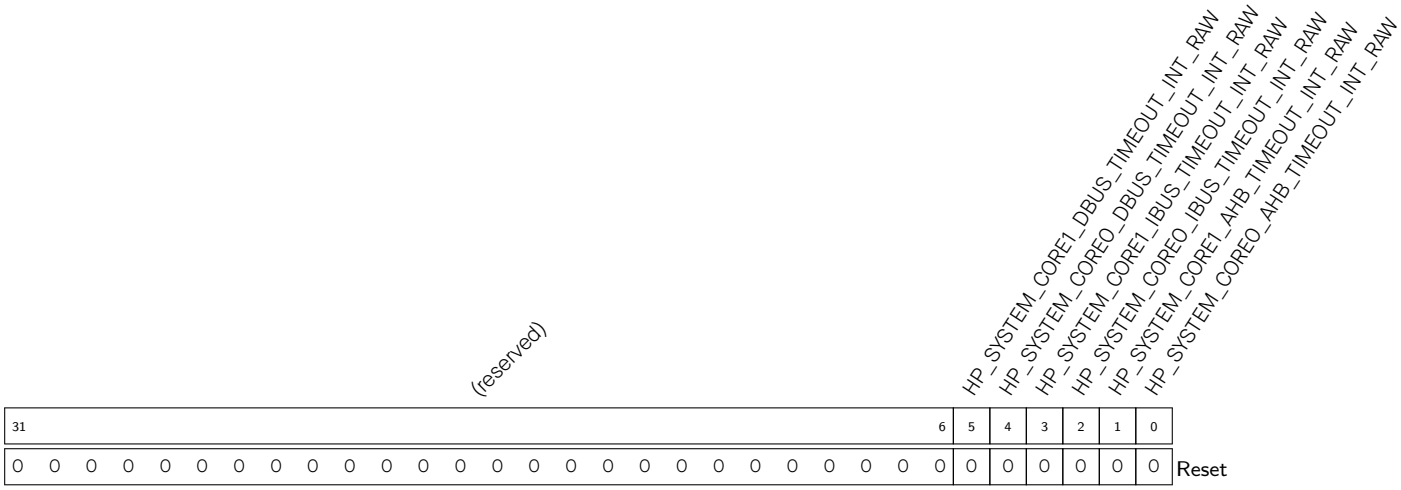
HP_SYSTEM_HP_SPM_PARITY_ERR_INT_ENA 写 1 使能 HP_SPM_PARITY_ERR_INT。(R/W)

Register 21.10. HP_SYSTEM_HP_SPM_INT_CLR_REG (0x0108)



HP_SYSTEM_HP_SPM_PARITY_ERR_INT_CLR 写 1 清除 HP_SPM_PARITY_ERR_INT。(WT)

Register 21.11. HP_SYSTEM_HP_CORE_TIMEOUT_INT_RAW_REG (0x01A8)



HP_SYSTEM_CORE0_AHB_TIMEOUT_INT_RAW HP_CORE0_AHB_TIMEOUT_INT 的原始中断状态。(R/WTC/SS)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_RAW HP_CORE1_AHB_TIMEOUT_INT 的原始中断状态。(R/WTC/SS)

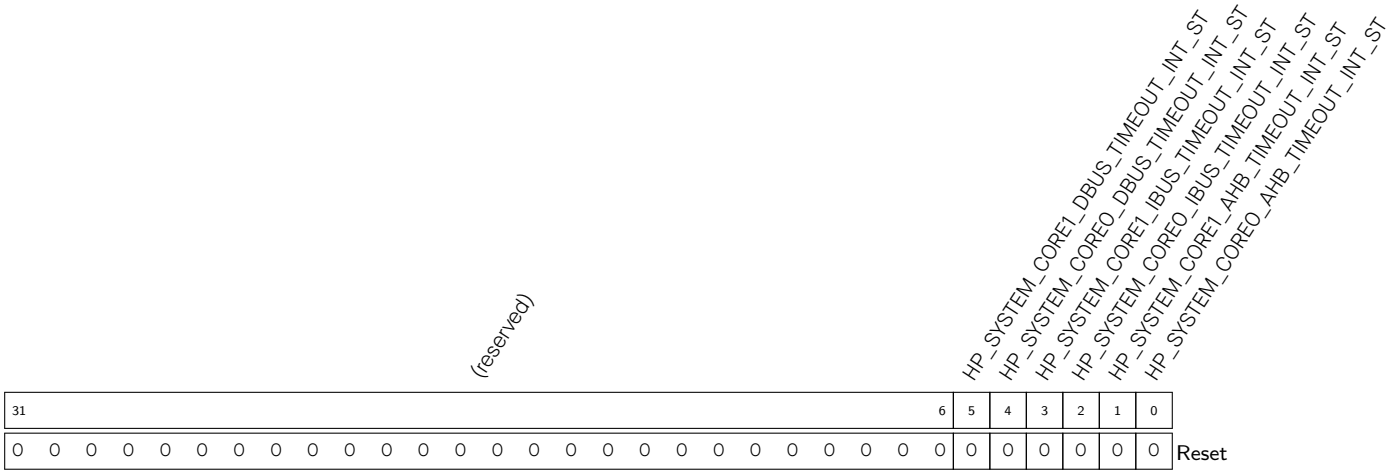
HP_SYSTEM_CORE0_IBUS_TIMEOUT_INT_RAW HP_CORE0_IBUS_TIMEOUT_INT 的原始中断状态。(R/WTC/SS)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_RAW HP_CORE1_IBUS_TIMEOUT_INT 的原始中断状态。(R/WTC/SS)

HP_SYSTEM_CORE0_DBUS_TIMEOUT_INT_RAW HP_CORE0_DBUS_TIMEOUT_INT 的原始中断状态。(R/WTC/SS)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_RAW HP_CORE1_DBUS_TIMEOUT_INT 的原始中断状态。(R/WTC/SS)

Register 21.12. HP_SYSTEM_HP_CORE_TIMEOUT_INT_ST_REG (0x01AC)



HP_SYSTEM_CORE0_AHB_TIMEOUT_INT_ST HP_CORE0_AHB_TIMEOUT_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_ST HP_CORE1_AHB_TIMEOUT_INT 的屏蔽中断状态。(RO)

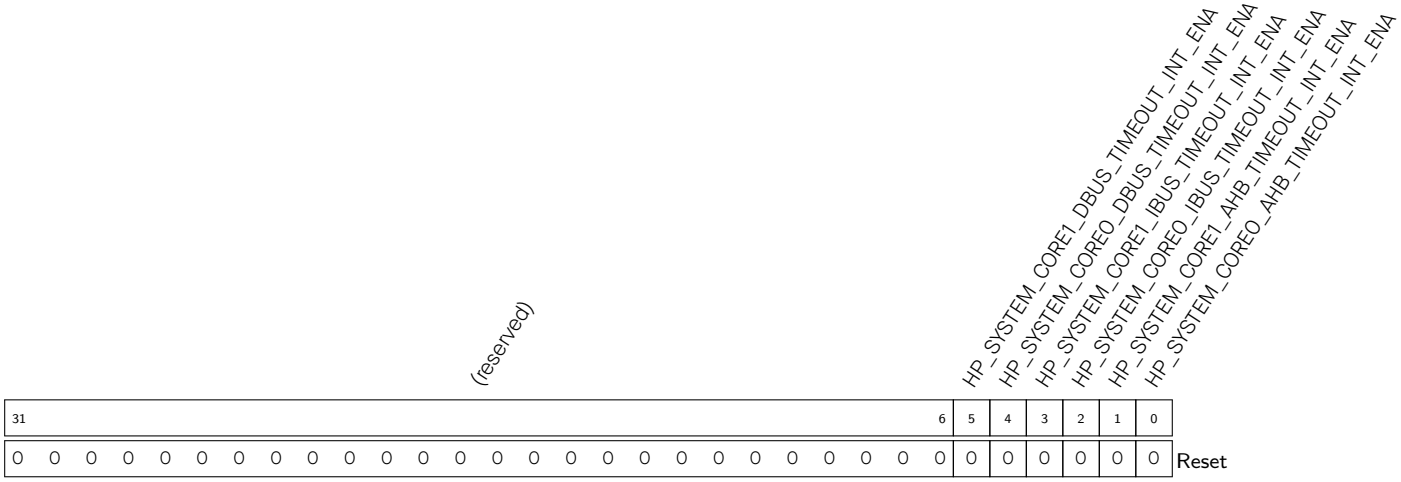
HP_SYSTEM_CORE0_IBUS_TIMEOUT_INT_ST HP_CORE0_IBUS_TIMEOUT_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_ST HP_CORE1_IBUS_TIMEOUT_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_CORE0_DBUS_TIMEOUT_INT_ST HP_CORE0_DBUS_TIMEOUT_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_ST HP_CORE1_DBUS_TIMEOUT_INT 的屏蔽中断状态。(RO)

Register 21.13. HP_SYSTEM_HP_CORE_TIMEOUT_INT_ENA_REG (0x01B0)



HP_SYSTEM_CORE0_AHB_TIMEOUT_INT_ENA 写 1 使能 HP_CORE0_AHB_TIMEOUT_INT。(R/W)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_ENA 写 1 使能 HP_CORE1_AHB_TIMEOUT_INT。(R/W)

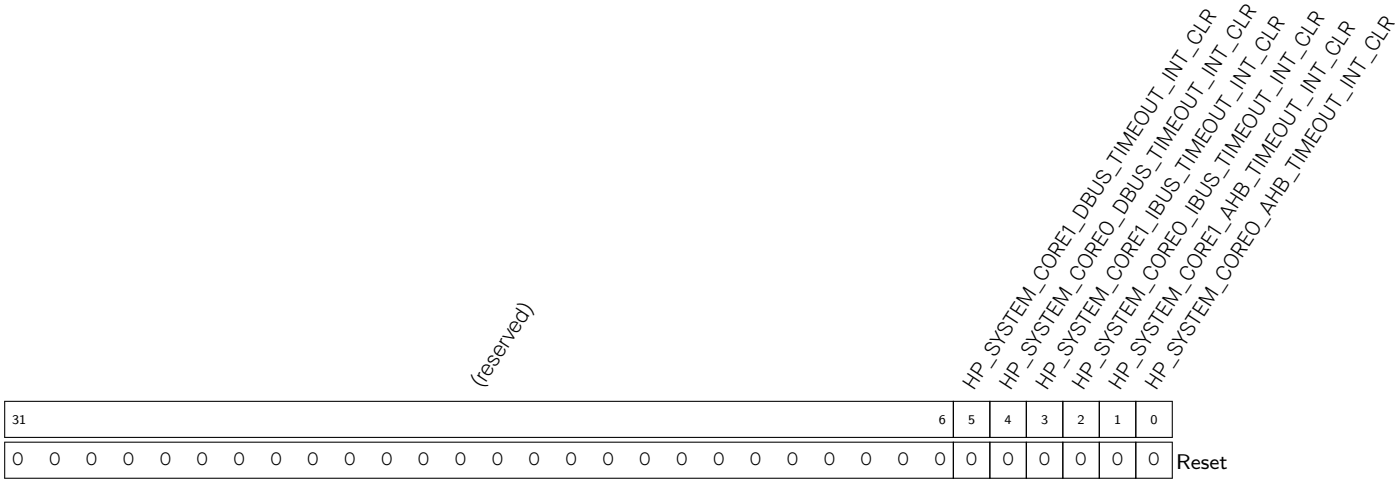
HP_SYSTEM_CORE0_IBUS_TIMEOUT_INT_ENA 写 1 使能 HP_CORE0_IBUS_TIMEOUT_INT。(R/W)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_ENA 写 1 使能 HP_CORE1_IBUS_TIMEOUT_INT。(R/W)

HP_SYSTEM_CORE0_DBUS_TIMEOUT_INT_ENA 写 1 使能 HP_CORE0_DBUS_TIMEOUT_INT。(R/W)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_ENA 写 1 使能 HP_CORE1_DBUS_TIMEOUT_INT。(R/W)

Register 21.14. HP_SYSTEM_HP_CORE_TIMEOUT_INT_CLR_REG (0x01B4)



HP_SYSTEM_CORE0_AHB_TIMEOUT_INT_CLR 写 1 清除 HP_CORE0_AHB_TIMEOUT_INT。(WT)

HP_SYSTEM_CORE1_AHB_TIMEOUT_INT_CLR 写 1 清除 HP_CORE1_AHB_TIMEOUT_INT。(WT)

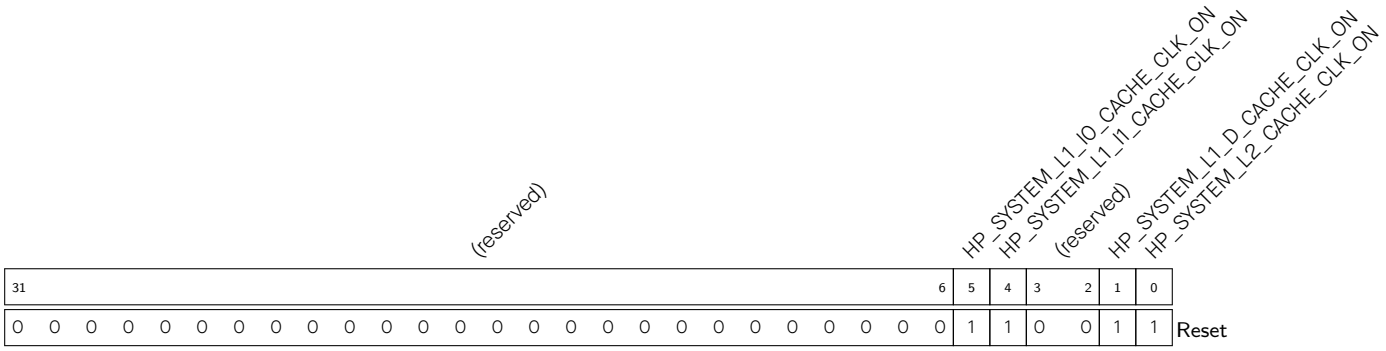
HP_SYSTEM_CORE0_IBUS_TIMEOUT_INT_CLR 写 1 清除 HP_CORE0_IBUS_TIMEOUT_INT。(WT)

HP_SYSTEM_CORE1_IBUS_TIMEOUT_INT_CLR 写 1 清除 HP_CORE1_IBUS_TIMEOUT_INT。(WT)

HP_SYSTEM_CORE0_DBUS_TIMEOUT_INT_CLR 写 1 清除 HP_CORE0_DBUS_TIMEOUT_INT。(WT)

HP_SYSTEM_CORE1_DBUS_TIMEOUT_INT_CLR 写 1 清除 HP_CORE1_DBUS_TIMEOUT_INT。(WT)

Register 21.15. HP_SYSTEM_HP_CACHE_CLK_CONFIG_REG (0x0020)



HP_SYSTEM_L2_CACHE_CLK_ON 配置是否启用 L2 cache 时钟。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L1_D_CACHE_CLK_ON 配置是否启用 L1 dcache 时钟。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L1_I1_CACHE_CLK_ON 配置是否启用 L1 ichache 1 时钟。

0：禁用

1：启用

(R/W)

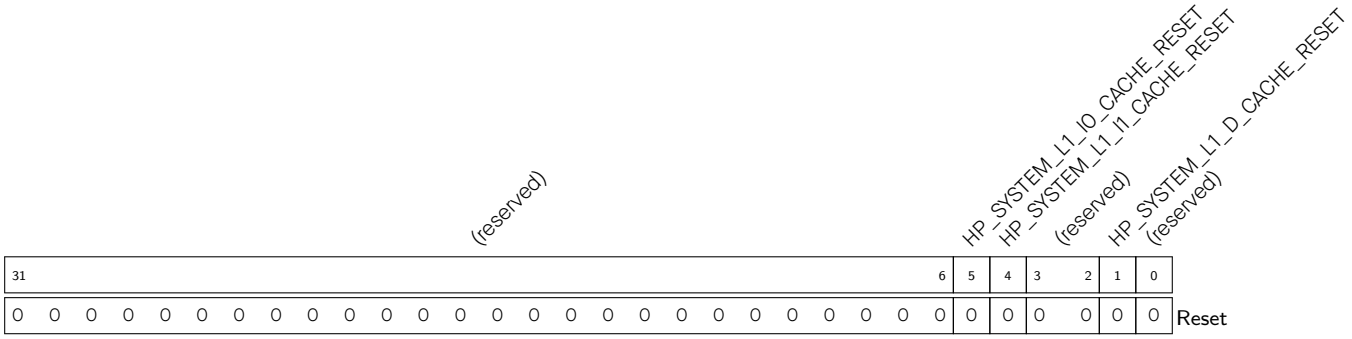
HP_SYSTEM_L1_I0_CACHE_CLK_ON 配置是否启用 L1 ichache 0 时钟。

0：禁用

1：启用

(R/W)

Register 21.16. HP_SYSTEM_CACHE_RESET_CONFIG_REG (0x0024)



HP_SYSTEM_L1_D_CACHE_RESET 配置是否复位 L1 dcache。

0：无影响

1：复位

(R/W)

HP_SYSTEM_L1_I1_CACHE_RESET 配置是否复位 L1 ichache 1。

0：无影响

1：复位

(R/W)

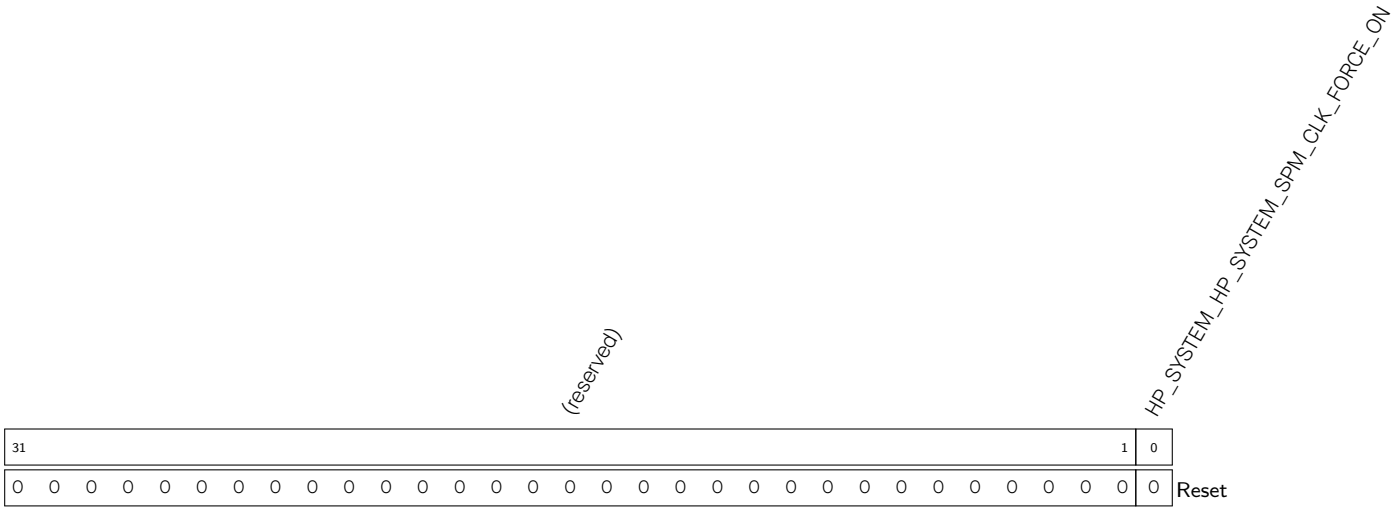
HP_SYSTEM_L1_I0_CACHE_RESET 配置是否复位 L1 ichache 0。

0：无影响

1：复位

(R/W)

Register 21.17. HP_SYSTEM_HP_SPM_RAM_PWR_CTRL0_REG (0x003C)



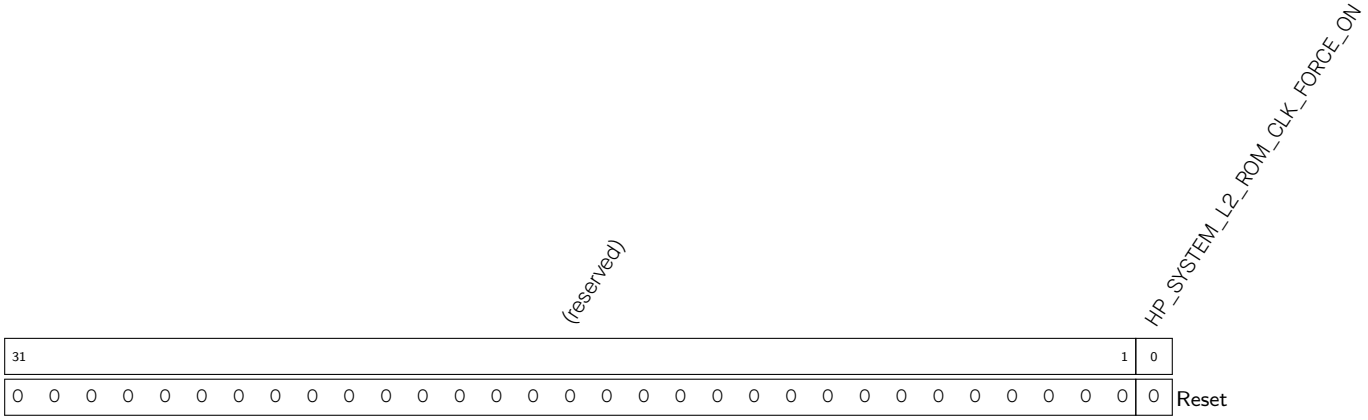
HP_SYSTEM_HP_SPM_CLK_FORCE_ON 配置是否强制开启 HP SPM 时钟。

0：无影响

1：强制开启

(R/W)

Register 21.18. HP_SYSTEM_L2_ROM_PWR_CTRL0_REG (0x0040)



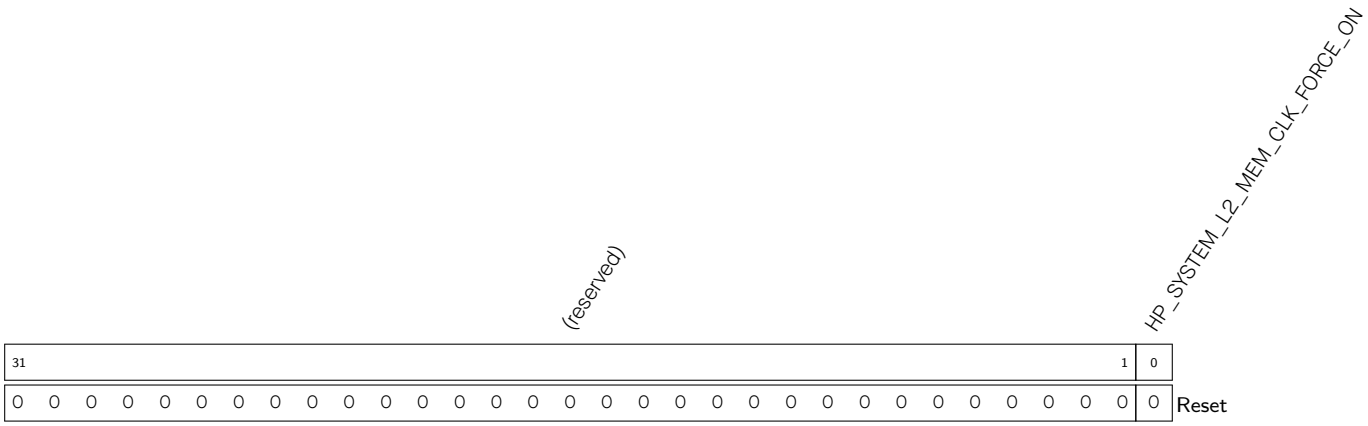
HP_SYSTEM_L2_ROM_CLK_FORCE_ON 配置是否强制开启 L2ROM 时钟。

0：无影响

1：强制开启

(R/W)

Register 21.19. HP_SYSTEM_HP_SYSTEM_L2_MEM_RAM_PWR_CTRL0_REG (0x0060)



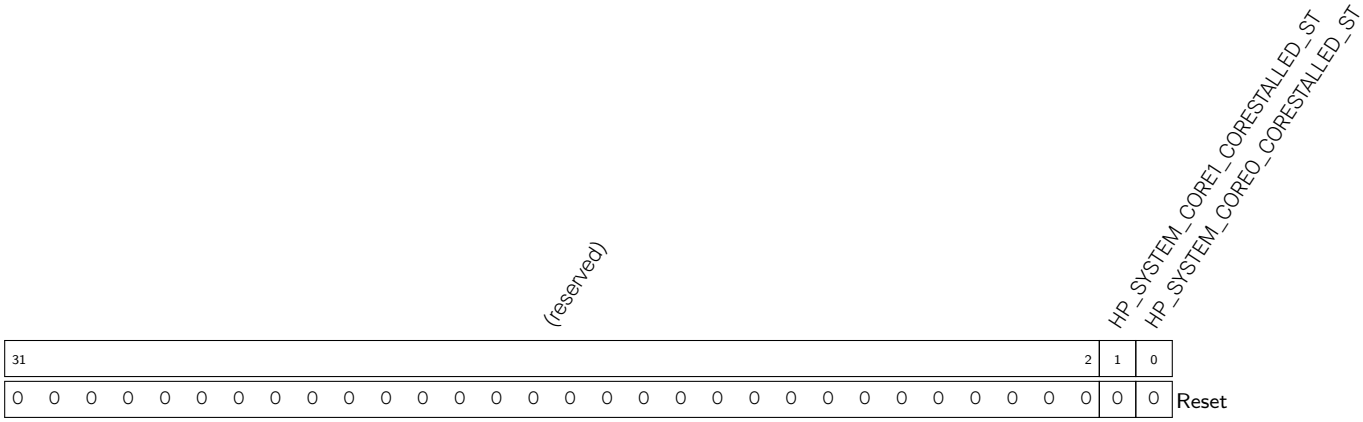
HP_SYSTEM_L2_MEM_CLK_FORCE_ON 配置是否强制开启 L2MEM 时钟。

0：无影响

1：强制开启

(R/W)

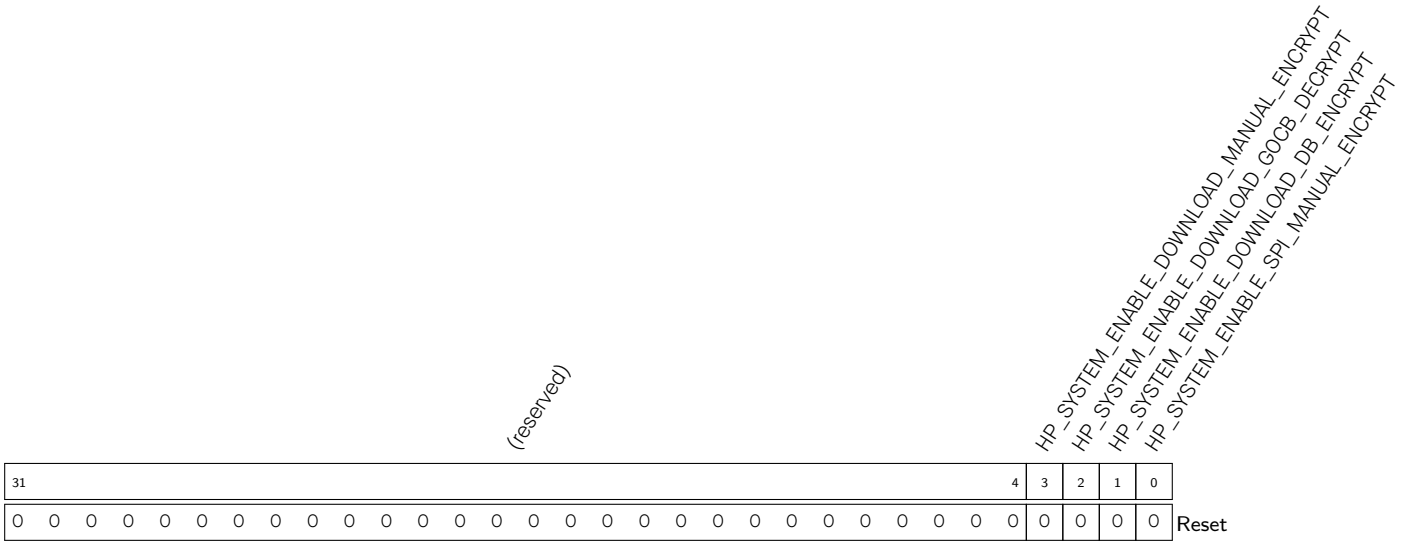
Register 21.20. HP_SYSTEM_CPU_CORESTALLED_ST_REG (0x0064)



HP_SYSTEM_CORE0_CORESTALLED_ST 表示 HP CPU0 的 stall 状态。(RO)

HP_SYSTEM_CORE1_CORESTALLED_ST 表示 HP CPU1 的 stall 状态。(RO)

Register 21.21. HP_SYSTEM_CRYPTO_CTRL_REG (0x0070)



HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 配置是否在 SPI Boot 模式下启用手动加密。

0：禁用

1：启用

(R/W)

HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT 配置是否在 Joint Download Boot 模式下启用自动加密。

0：禁用

1：启用

(R/W)

HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT 配置是否在 Joint Download Boot 模式下启用自动解密。

0：禁用

1：启用

(R/W)

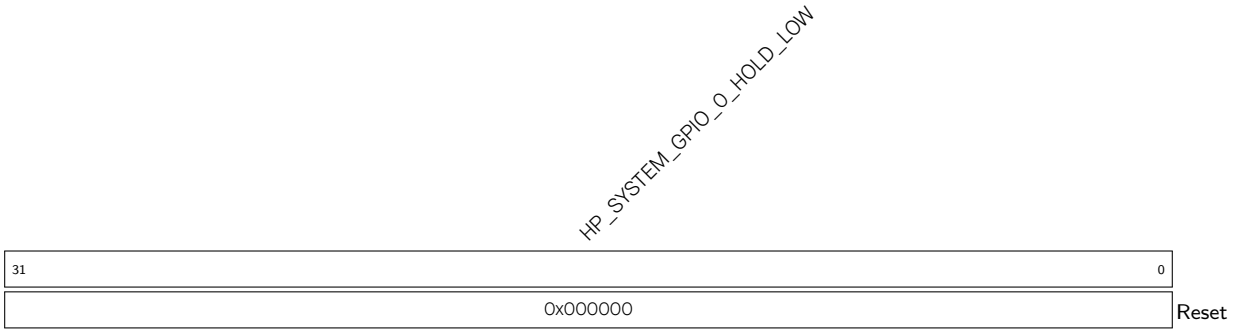
HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 配置是否在 Joint Download Boot 模式下启用手动加密。

0：禁用

1：启用

(R/W)

Register 21.22. HP_SYSTEM_GPIO_O_HOLD_CTRL0_REG (0x0074)



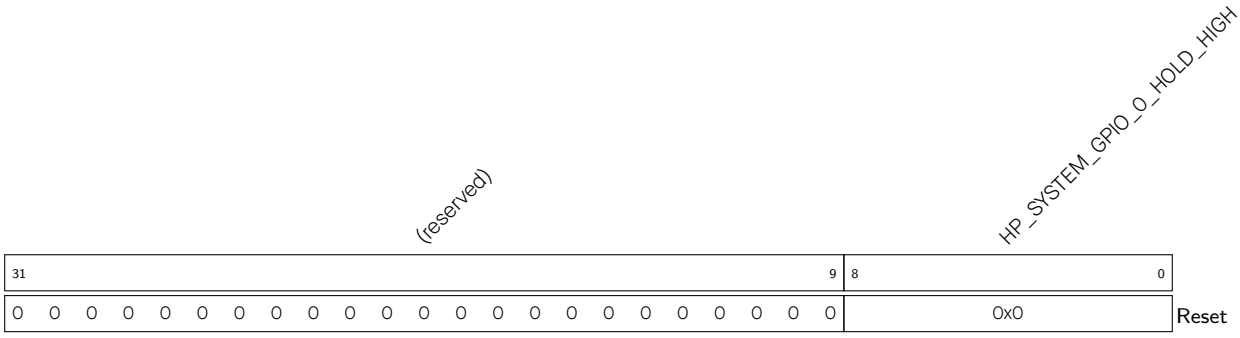
HP_SYSTEM_GPIO_O_HOLD_LOW 配置是否保持 HP GPIO16 ~ HP GPIO47 的状态。

0：不保持

1：保持

(R/W)

Register 21.23. HP_SYSTEM_GPIO_O_HOLD_CTRL1_REG (0x0078)



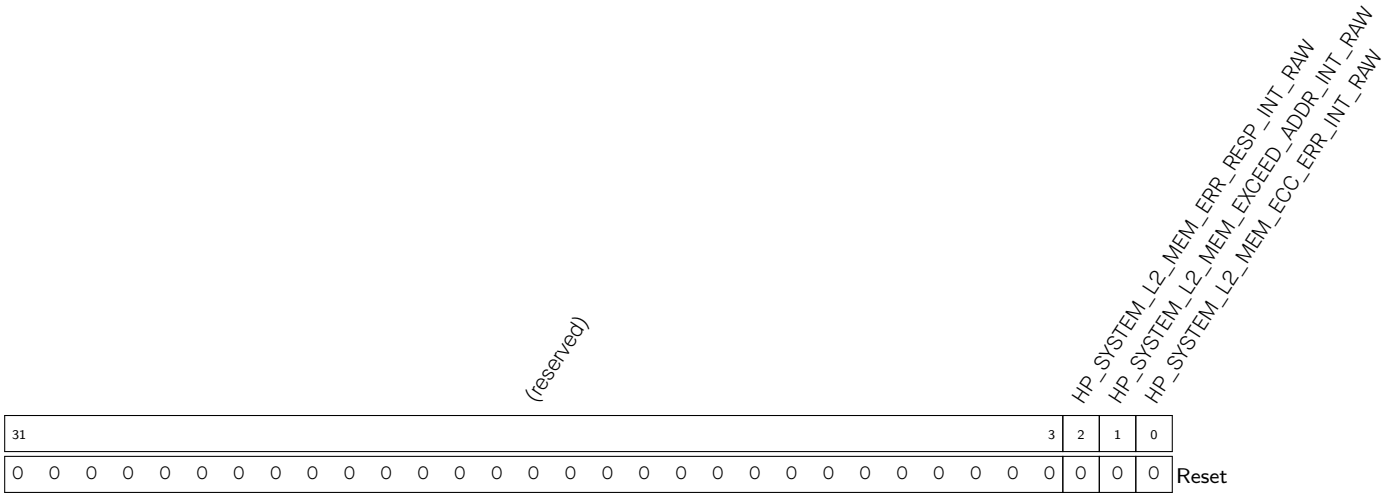
HP_SYSTEM_GPIO_O_HOLD_HIGH 配置是否保持 HP GPIO48 ~ HP GPIO56。

0：不保持

1：保持

(R/W)

Register 21.24. HP_SYSTEM_L2_MEM_INT_RAW_REG (0x009C)

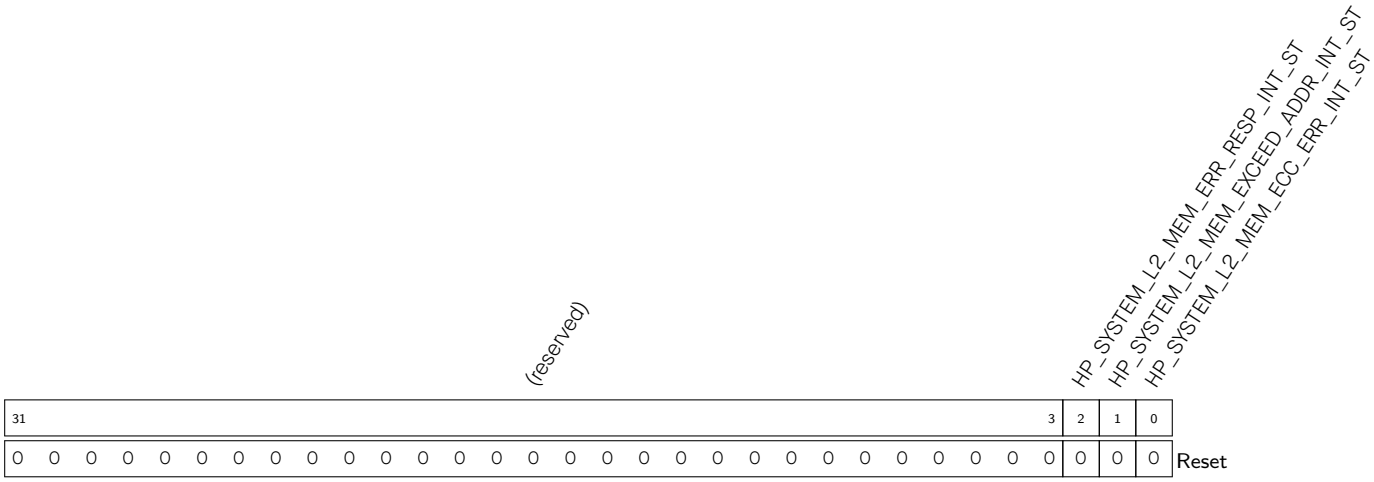


HP_SYSTEM_L2_MEM_ECC_ERR_INT_RAW L2_MEM_ECC_ERR_INT 的原始中断状态，当检测到并纠正 ECC 的一位翻转错误时触发。(R/WTC/SS)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_RAW L2_MEM_EXCEED_ADDR_INT 的原始中断状态，当访问地址在不同模式下超出限制时触发。
旁路模式：0xff9ffff
L2 cache 128 KB 模式：0xff80000
L2 cache 256 KB 模式：0xff60000
(R/WTC/SS)

HP_SYSTEM_L2_MEM_ERR_RESP_INT_RAW L2_MEM_ERR_RESP_INT 的原始中断状态，当发生错误响应时触发。(R/WTC/SS)

Register 21.25. HP_SYSTEM_L2_MEM_INT_ST_REG (0x00A0)

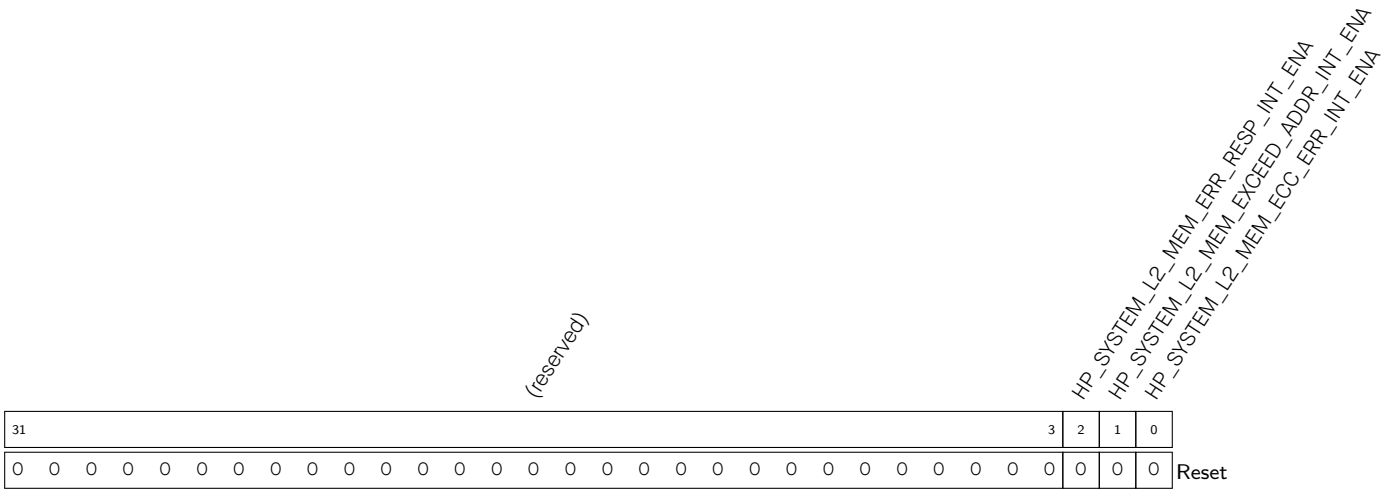


HP_SYSTEM_L2_MEM_ECC_ERR_INT_ST L2_MEM_ECC_ERR_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_ST L2_MEM_EXCEED_ADDR_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_L2_MEM_ERR_RESP_INT_ST L2_MEM_ERR_RESP_INT 的屏蔽中断状态。(RO)

Register 21.26. HP_SYSTEM_L2_MEM_INT_ENA_REG (0x00A4)

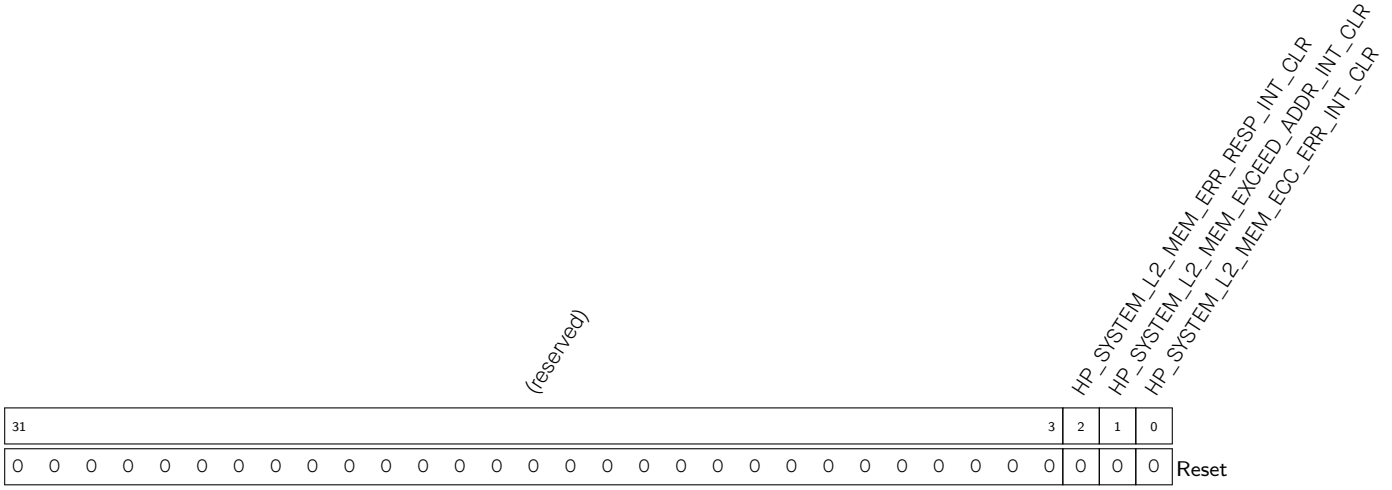


HP_SYSTEM_L2_MEM_ECC_ERR_INT_ENA 写 1 使能 L2_MEM_ECC_ERR_INT。(R/W)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_ENA 写 1 使能 L2_MEM_EXCEED_ADDR_INT。(R/W)

HP_SYSTEM_L2_MEM_ERR_RESP_INT_ENA 写 1 使能 L2_MEM_ERR_RESP_INT。(R/W)

Register 21.27. HP_SYSTEM_L2_MEM_INT_CLR_REG (0x00A8)

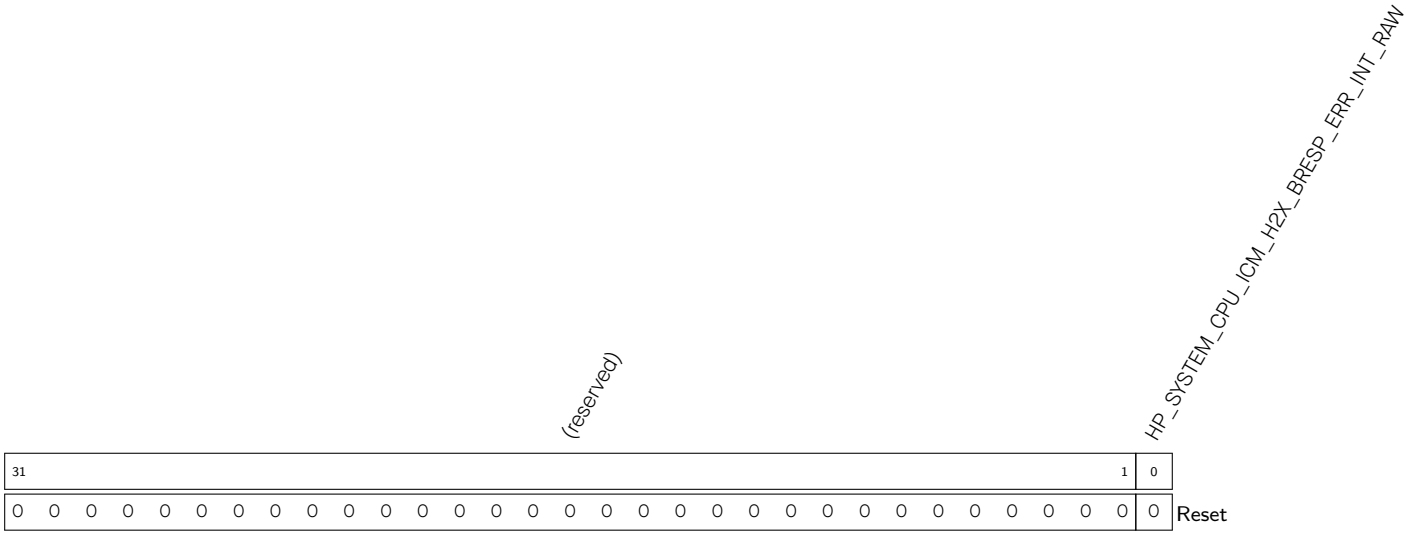


HP_SYSTEM_L2_MEM_ECC_ERR_INT_CLR 写 1 清除 L2_MEM_ECC_ERR_INT。(WT)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_CLR 写 1 清除 L2_MEM_EXCEED_ADDR_INT。(WT)

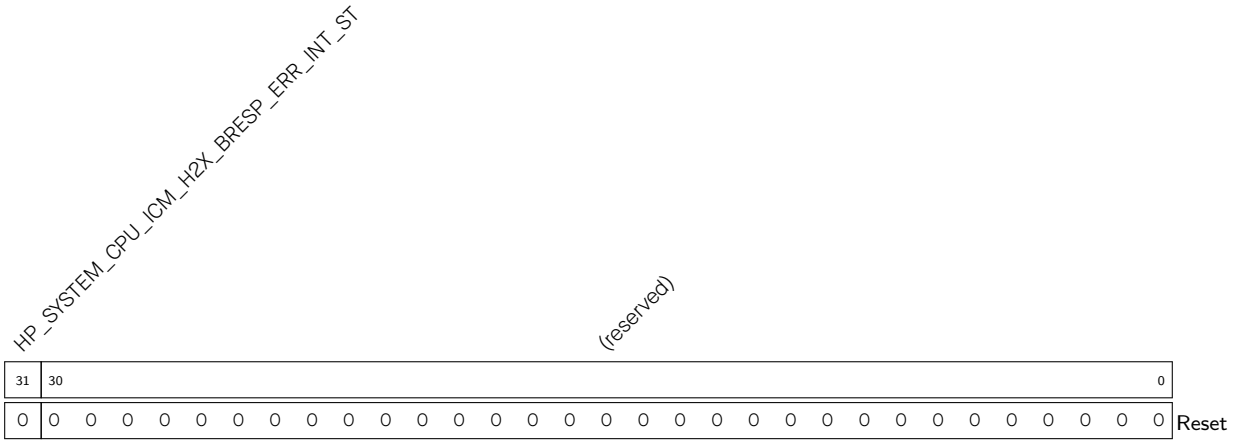
HP_SYSTEM_L2_MEM_ERR_RESP_INT_CLR 写 1 清除 L2_MEM_ERR_RESP_INT。(WT)

Register 21.28. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_RAW_REG (0x0188)



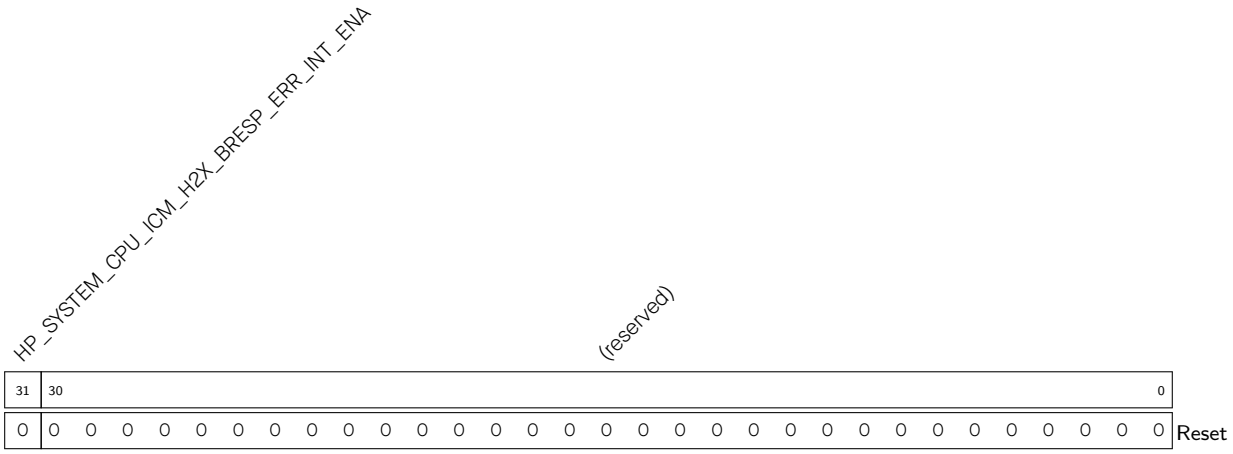
HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_RAW CPU_ICM_H2X_BRESP_ERR_INT 的原始中断状态，当 AHB2AXI 处于 Post Write 模式中发生 BRESP 错误时触发。(R/WTC/SS)

Register 21.29. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ST_REG (0x018C)



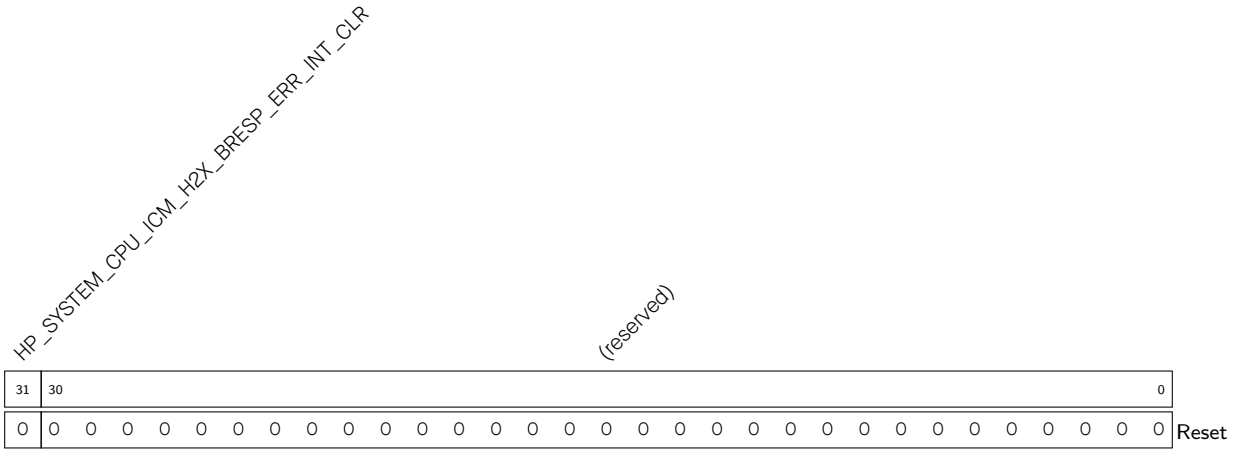
HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_ST CPU_ICM_H2X_BRESP_ERR_INT 的屏蔽中断状态。(RO)

Register 21.30. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_ENA_REG (0x0190)



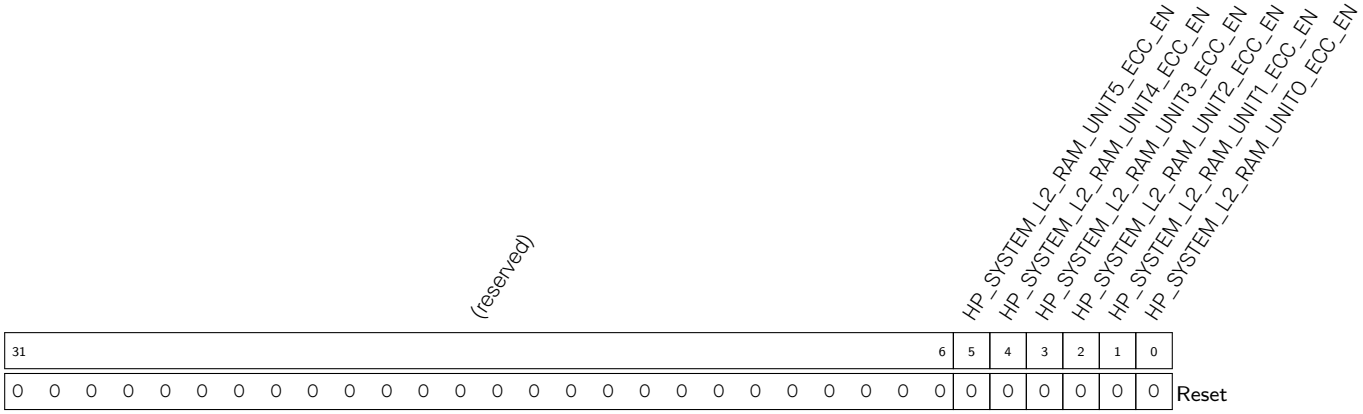
HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_ENA 写 1 使能 CPU_ICM_H2X_BRESP_ERR_INT。(R/W)

Register 21.31. HP_SYSTEM_AHB2AXI_BRESP_ERR_INT_CLR_REG (0x0194)



HP_SYSTEM_CPU_ICM_H2X_BRESP_ERR_INT_CLR 写 1 清除 CPU_ICM_H2X_BRESP_ERR_INT。(WT)

Register 21.32. HP_SYSTEM_L2_MEM_L2_RAM_ECC_REG (0x00AC)



HP_SYSTEM_L2_RAM_UNIT0_ECC_EN 配置是否启用 L2MEM unit0 的 ECC 检查。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_RAM_UNIT1_ECC_EN 配置是否启用 L2MEM unit1 的 ECC 检查。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_RAM_UNIT2_ECC_EN 配置是否启用 L2MEM unit2 的 ECC 检查。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_RAM_UNIT3_ECC_EN 配置是否启用 L2MEM unit3 的 ECC 检查。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_RAM_UNIT4_ECC_EN 配置是否启用 L2MEM unit4 的 ECC 检查。

0：禁用

1：启用

(R/W)

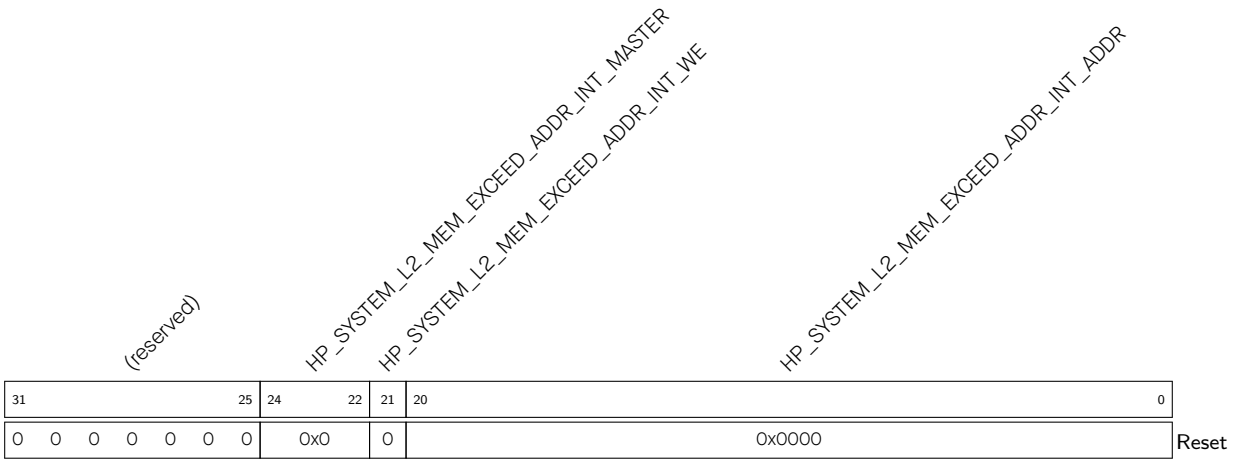
HP_SYSTEM_L2_RAM_UNIT5_ECC_EN 配置是否启用 L2MEM unit5 的 ECC 检查。

0：禁用

1：启用

(R/W)

Register 21.33. HP_SYSTEM_L2_MEM_INT_RECORD0_REG (0x00B0)



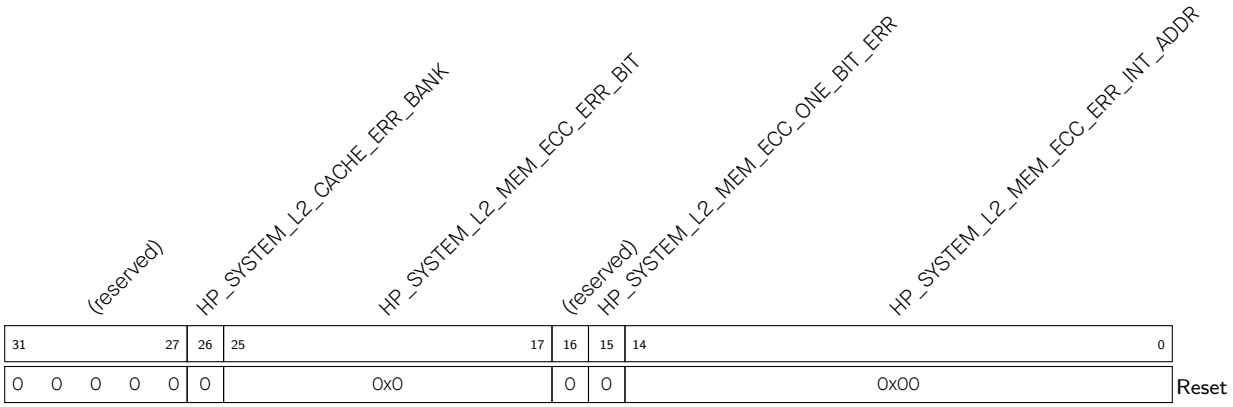
HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_ADDR 表示 L2_MEM_EXCEED_ADDR_INT 发生时的地址。(RO)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_WE 表示 L2_MEM_EXCEED_ADDR_INT 发生时的操作（写/读）。(RO)

HP_SYSTEM_L2_MEM_EXCEED_ADDR_INT_MASTER 表示触发 L2_MEM_EXCEED_ADDR_INT 的源。

0：无效值。没有作用
1：cachebus0
2：cachebus1
3：dma_axi
4：无效值。没有作用
5：无效值。没有作用
6：ahb
7：无效值。没有作用
(RO)

Register 21.34. HP_SYSTEM_L2_MEM_INT_RECORD1_REG (0x00B4)



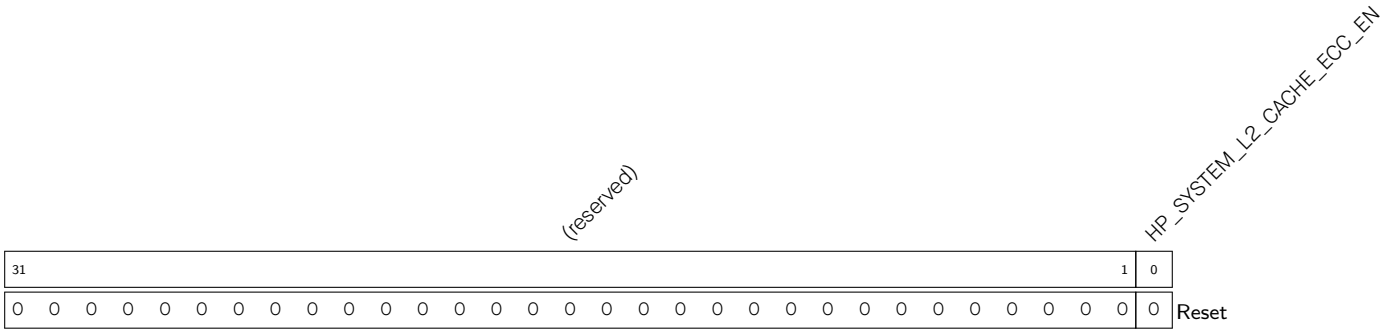
HP_SYSTEM_L2_MEM_ECC_ERR_INT_ADDR 表示 L2_MEM_ECC_ERR_INT 发生时的地址。(RO)

HP_SYSTEM_L2_MEM_ECC_ONE_BIT_ERR 表示 L2_MEM_ECC_ERR_INT 发生时的错误。(RO)

HP_SYSTEM_L2_MEM_ECC_ERR_BIT 表示触发 L2_MEM_ECC_ERR_INT 的翻转位。(RO)

HP_SYSTEM_L2_CACHE_ERR_BANK 表示 L2 cache ECC 错误的条编号。(RO)

Register 21.35. HP_SYSTEM_L2_MEM_L2_CACHE_ECC_REG (0x00C4)



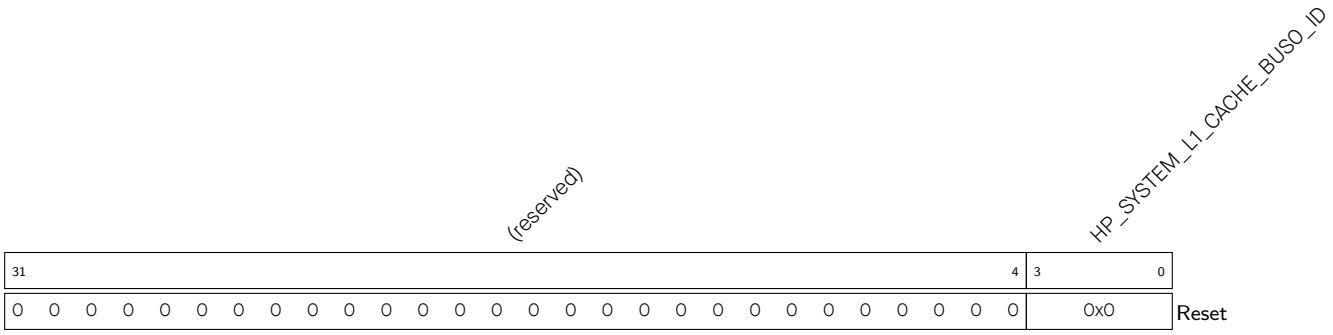
HP_SYSTEM_L2_CACHE_ECC_EN 配置是否启用 L2 cache 的 ECC 校验。

0：禁用

1：启用

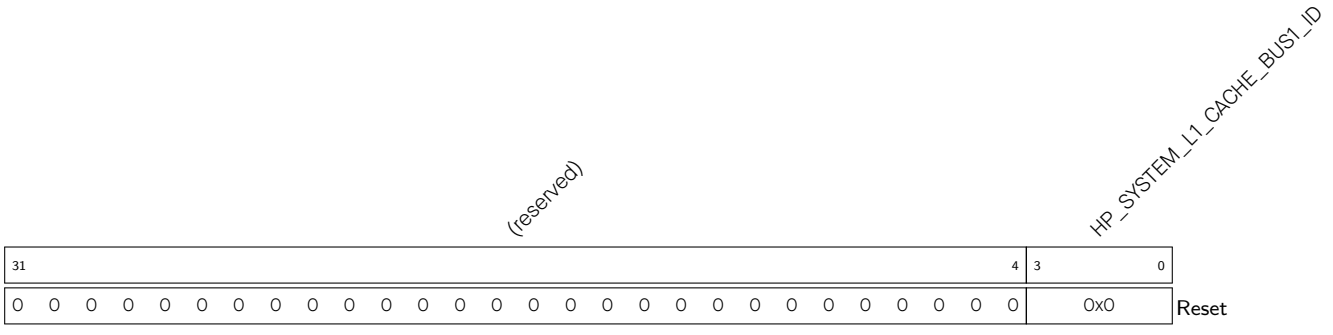
(R/W)

Register 21.36. HP_SYSTEM_HP_L1CACHE_BUS0_ID_REG (0x00C8)



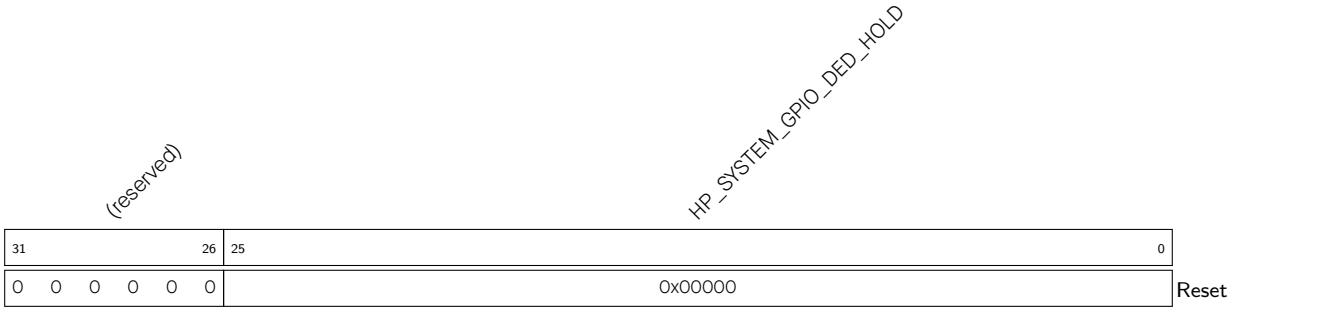
HP_SYSTEM_L1_CACHE_BUS0_ID 配置 L1 cache bus0 的 UID。(R/W)

Register 21.37. HP_SYSTEM_HP_L1CACHE_BUS1_ID_REG (0x00CC)



HP_SYSTEM_L1_CACHE_BUS1_ID 配置 L1 cache bus1 的 UID。(R/W)

Register 21.38. HP_SYSTEM_GPIO_DED_HOLD_CTRL_REG (0x00F0)



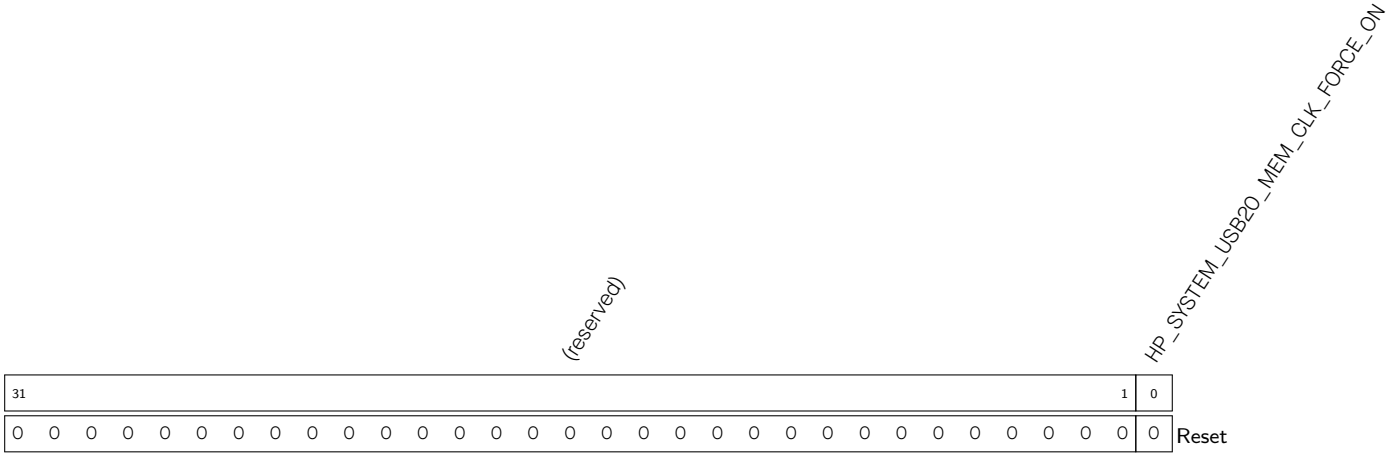
HP_SYSTEM_GPIO_DED_HOLD 配置是否保持管脚状态。

位 [25:6]：16 线 PSRAM 管脚

位 [5:0]：flash 管脚

(R/W)

Register 21.39. HP_SYSTEM_HP_USB20OTG_MEM_CTRL_REG (0x00F8)



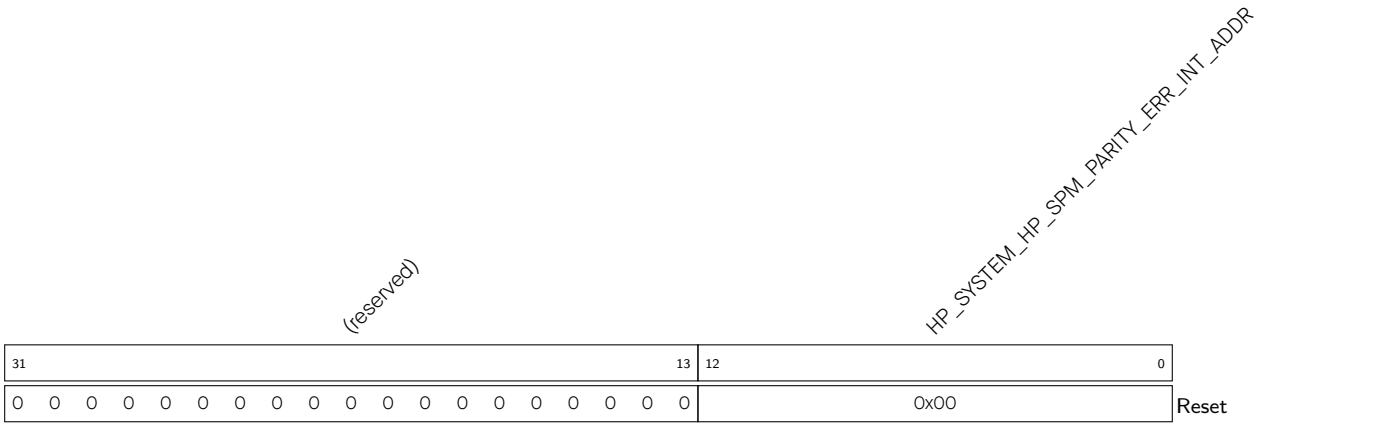
HP_SYSTEM_USB20_MEM_CLK_FORCE_ON 配置是否强制开启 USB OTGHS 中的内存时钟。

0：不强制开启

1：强制开启

(R/W)

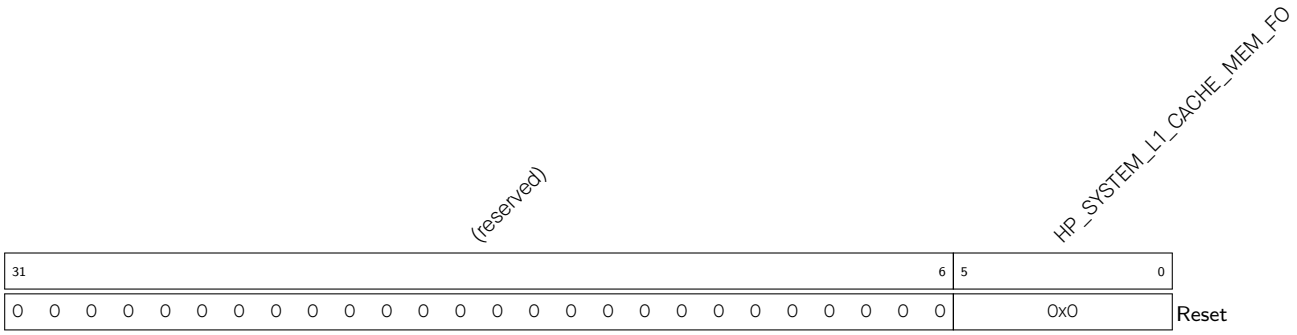
Register 21.40. HP_SYSTEM_HP_SPM_PARITY_INT_RECORD_REG (0x010C)



HP_SYSTEM_HP_SPM_PARITY_ERR_INT_ADDR 表示 HP_SPM_PARITY_ERR_INT 发生时的地址。

(RO)

Register 21.41. HP_SYSTEM_HP_L1_CACHE_PWR_CTRL_REG (0x0110)



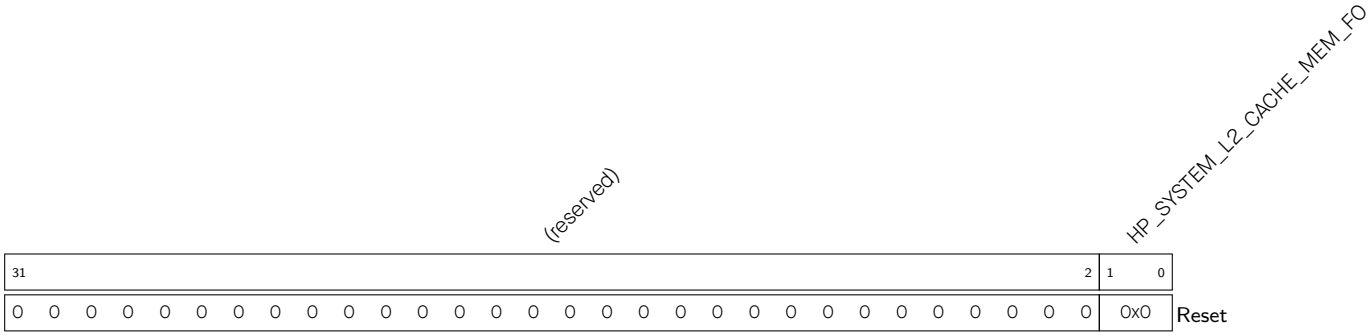
HP_SYSTEM_L1_CACHE_MEM_FO 配置是否强制开启 L1 cache。

0：不强制开启

1：强制开启

(R/W)

Register 21.42. HP_SYSTEM_HP_L2_CACHE_PWR_CTRL_REG (0x0114)



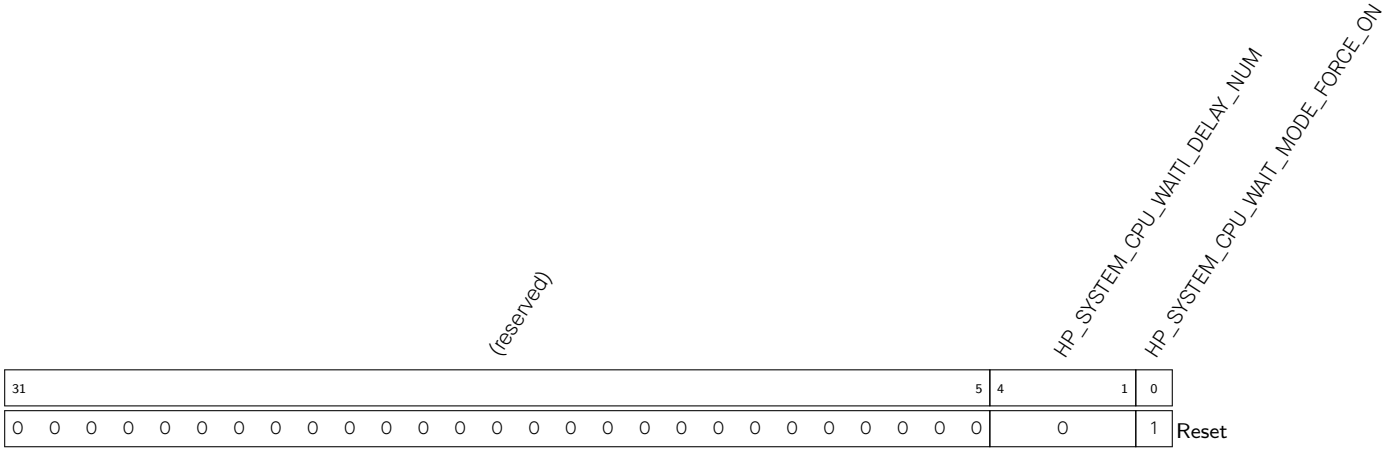
HP_SYSTEM_L2_CACHE_MEM_FO 配置是否强制开启 L2 cache。

0：不强制开启

1：强制开启

(R/W)

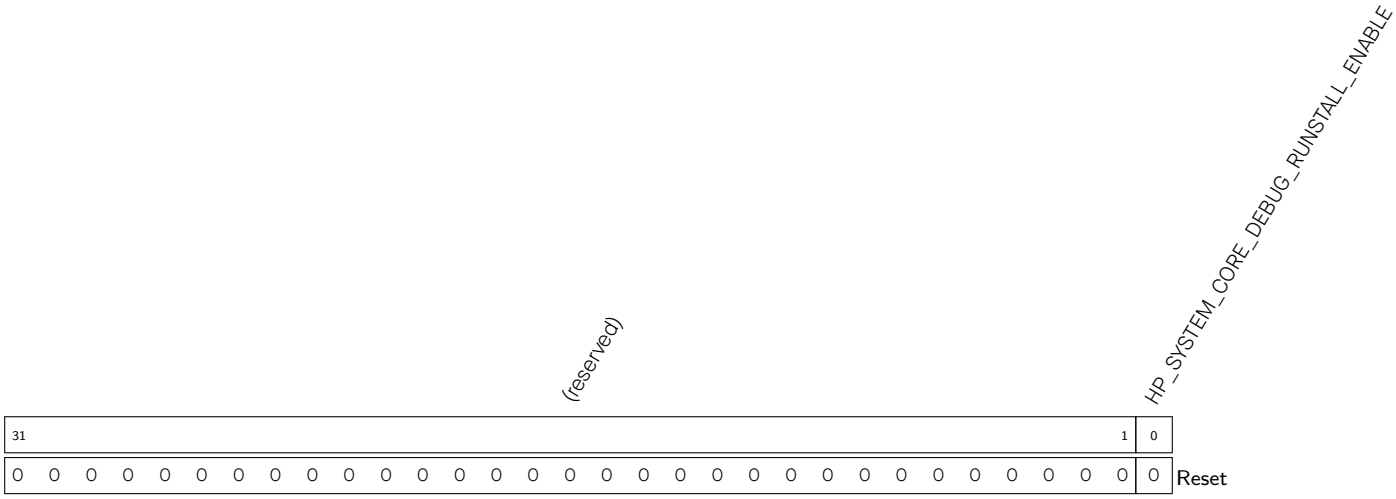
Register 21.43. HP_SYSTEM_HP_CPU_WAITI_CONF_REG (0x0118)



HP_SYSTEM_CPU_WAIT_MODE_FORCE_ON 配置是否强制开启 cpu_waiti_clk。(R/W)

HP_SYSTEM_CPU_WAITI_DELAY_NUM 配置 CPU 进入等待模式时的延迟周期。在配置的延迟后，cpu_waiti_clk 关闭。(R/W)

Register 21.44. HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_REG (0x011C)



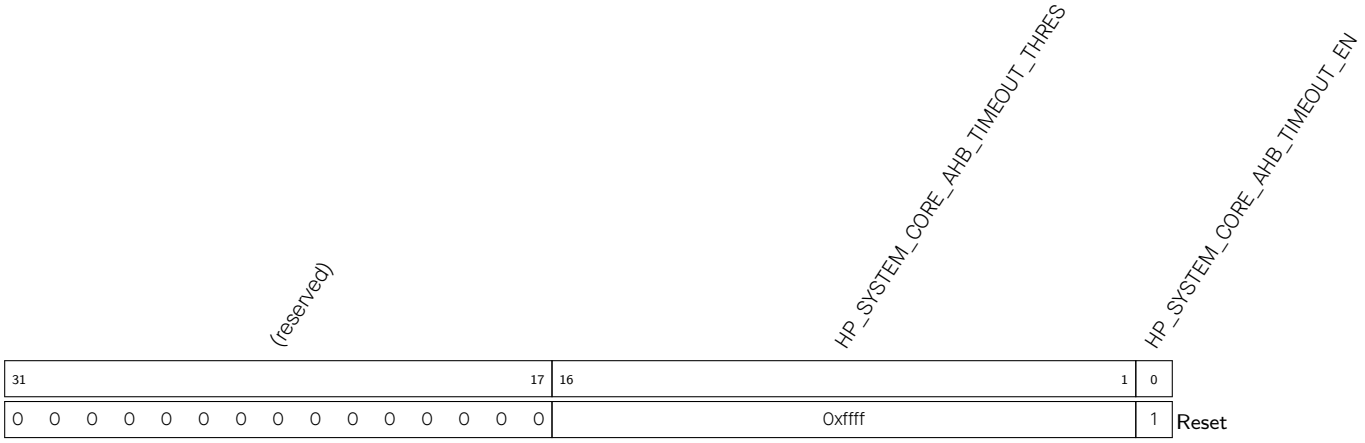
HP_SYSTEM_CORE_DEBUG_RUNSTALL_ENABLE 配置是否启用 HP CPU 和 LP CPU 之间的调试运行暂停功能。

0：禁用

1：启用

(R/W)

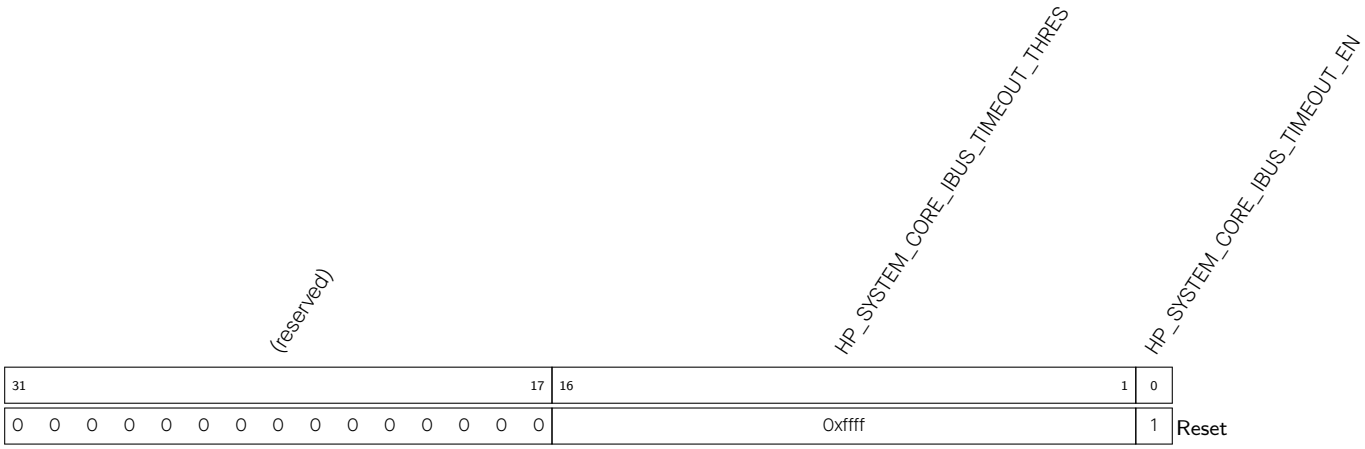
Register 21.45. HP_SYSTEM_HP_CORE_AHB_TIMEOUT_REG (0x0120)



HP_SYSTEM_CORE_AHB_TIMEOUT_EN 配置是否启用 HP CPU0/1 AHB 总线的超时保护。(R/W)

HP_SYSTEM_CORE_AHB_TIMEOUT_THRES 配置 HP CPU0/1 AHB 总线的超时阈值。(R/W)

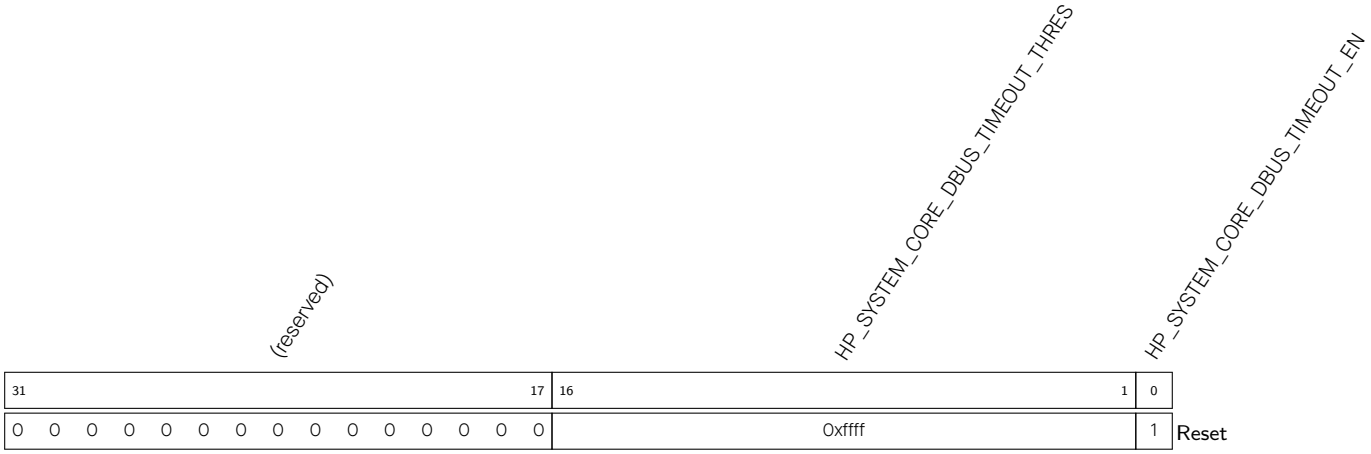
Register 21.46. HP_SYSTEM_HP_CORE_IBUS_TIMEOUT_REG (0x0124)



HP_SYSTEM_CORE_IBUS_TIMEOUT_EN 配置是否启用 HP CPU0/1 IBUS 的超时保护。(R/W)

HP_SYSTEM_CORE_IBUS_TIMEOUT_THRES 配置 HP CPU0/1 IBUS 的超时阈值。(R/W)

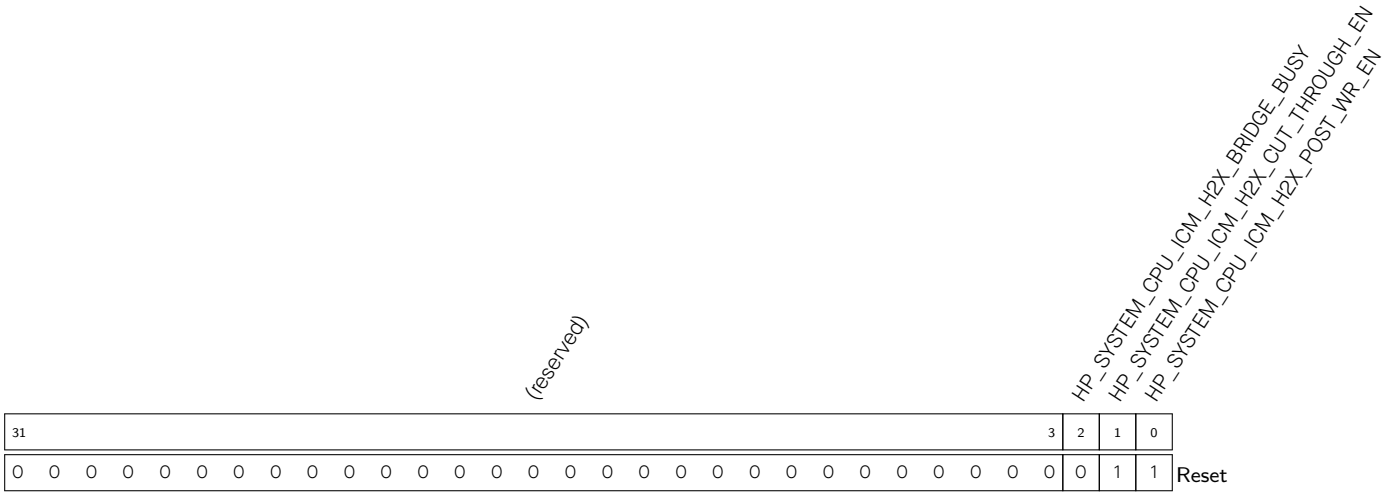
Register 21.47. HP_SYSTEM_HP_CORE_DBUS_TIMEOUT_REG (0x0128)



HP_SYSTEM_CORE_DBUS_TIMEOUT_EN 配置是否启用 HP CPU0/1 DBUS 的超时保护。(R/W)

HP_SYSTEM_CORE_DBUS_TIMEOUT_THRES 配置 HP CPU0/1 DBUS 的超时阈值。(R/W)

Register 21.48. HP_SYSTEM_HP_ICM_CPU_H2X_CFG_REG (0x0138)

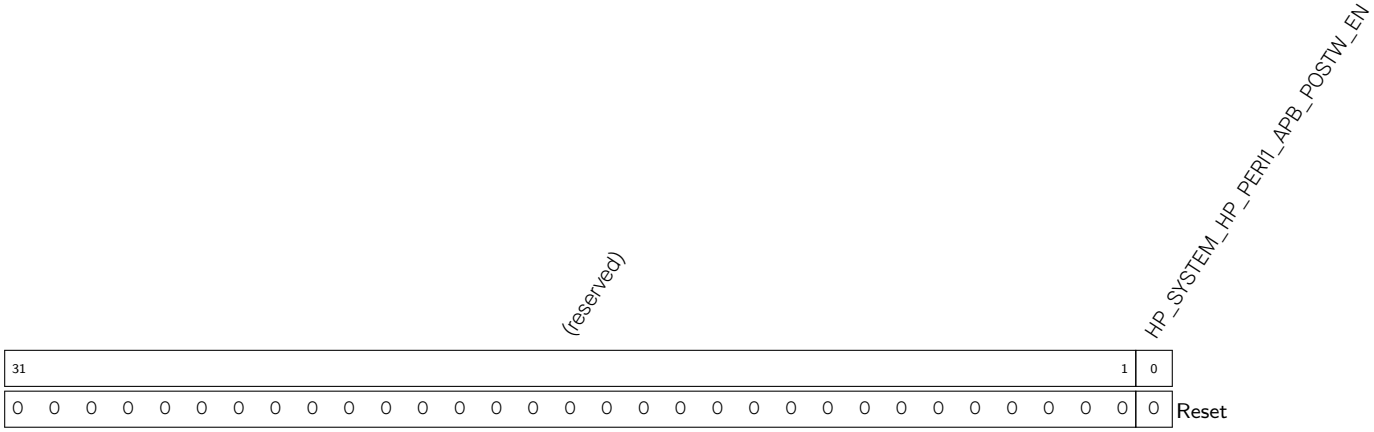


HP_SYSTEM_CPU_ICM_H2X_POST_WR_EN 配置是否在 AHB2AXI 桥中启用 Post Write 模式以提高写入性能。(R/W)

HP_SYSTEM_CPU_ICM_H2X_CUT_THROUGH_EN 配置是否在 AHB2AXI 桥中启用 CUT Through 模式以提高写入性能。(R/W)

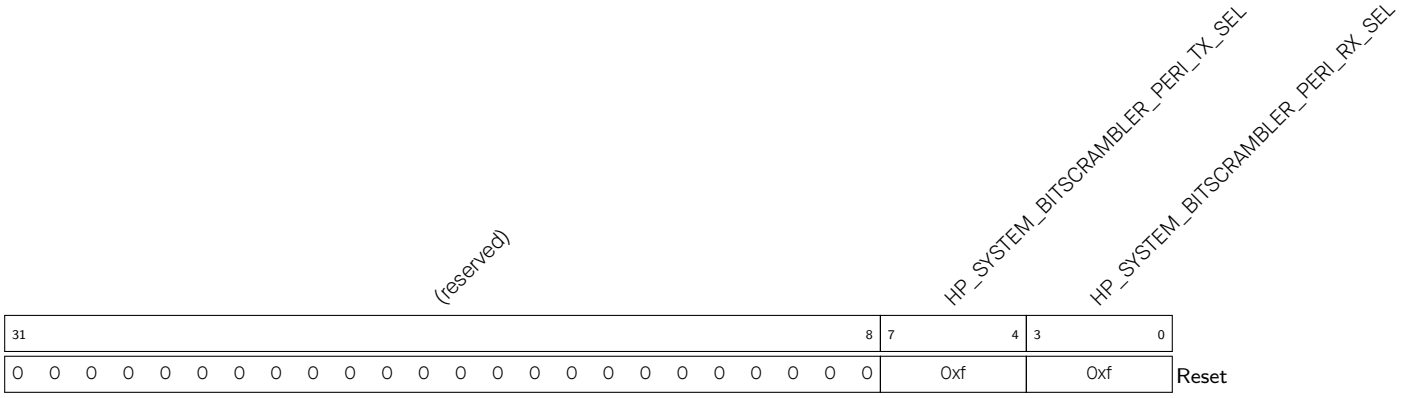
HP_SYSTEM_CPU_ICM_H2X_BRIDGE_BUSY 表示 AHB2AXI 桥何时处于忙碌状态。(RO)

Register 21.49. HP_SYSTEM_PERI1_APB_POSTW_EN_REG (0x013C)



HP_SYSTEM_HP_PERI1_APB_POSTW_EN 配置是否启用外设的 APB2APB 桥的 Post Write 模式。此模式加速写操作，但需要时间更新寄存器值。(R/W)

Register 21.50. HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG (0x0140)



HP_SYSTEM_BITSCRAMBLER_PERI_RX_SEL 配置选择接入比特调节器接收通道的 DMA 外设。

0：LCD_CAM
1：GPSPI2
2：GPSPI3
3：PARL_IO
4：AES
5：SHA
6：ADC
7：I2S0
8：I2S1
9：I2S2
10：I3C_MST
11：UHCI0
12：RMT
13：无效值。没有作用
14：无效值。没有作用
15：无效值。没有作用
(R/W)

见下页...

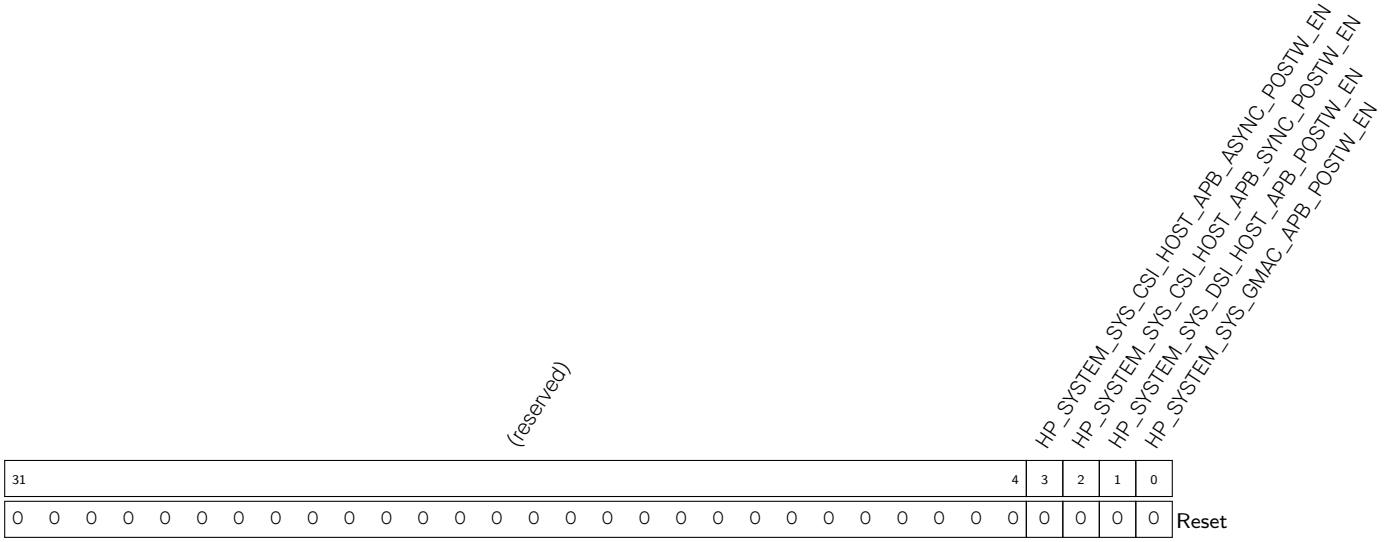
Register 21.50. HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG (0x0140)

接上页...

HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 配置选择接入比特调节器发送通道的 DMA 外设。

0：LCD_CAM
1：GPSPI2
2：GPSPI3
3：PARL_IO
4：AES
5：SHA
6：ADC
7：I2S0
8：I2S1
9：I2S2
10：I3C_MST
11：UHCI0
12：RMT
13：无效值。没有作用
14：无效值。没有作用
15：无效值。没有作用
(R/W)

Register 21.51. HP_SYSTEM_APB_SYNC_POSTW_EN_REG (0x0144)



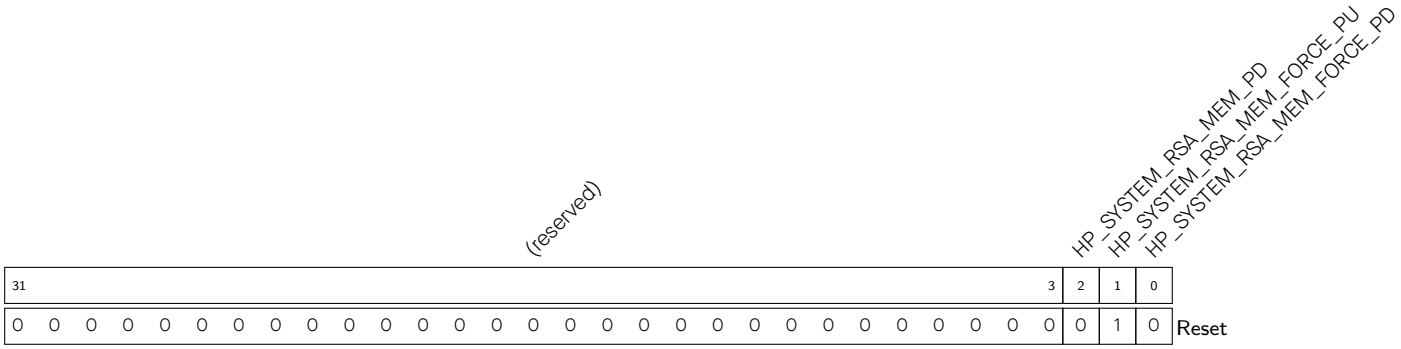
HP_SYSTEM_SYS_GMAC_APB_POSTW_EN 配置是否启用 GMAC 中同步 APB2APB 桥的 Post Write 模式。该模式加速写操作，但需要时间更新寄存器值。(R/W)

HP_SYSTEM_SYS_DSI_HOST_APB_POSTW_EN 配置是否启用 DSI Host 中同步 APB2APB 桥的 Post Write 模式。该模式加速写操作，但需要时间更新寄存器值。(R/W)

HP_SYSTEM_SYS_CSI_HOST_APB_SYNC_POSTW_EN 配置是否启用 CSI Host 中同步 APB2APB 桥的 Post Write 模式。该模式加速写操作，但需要时间更新寄存器值。(R/W)

HP_SYSTEM_SYS_CSI_HOST_APB_ASYNC_POSTW_EN 配置是否启用 CSI Host 中异步 APB2APB 桥的 Post Write 模式。该模式加速写操作，但需要时间更新寄存器值。(R/W)

Register 21.52. HP_SYSTEM_HP_RSA_PD_CTRL_REG (0x01D0)



HP_SYSTEM_RSA_MEM_FORCE_PD 配置是否强制关闭 RSA 内部存储器电源。此设置的优先级低于 HP_SYSTEM_RSA_MEM_FORCE_PU。

0：无影响

1：关闭电源

(R/W)

HP_SYSTEM_RSA_MEM_FORCE_PU 配置是否强制开启 RSA 内部存储器电源。

0：无影响

1：开启电源

(R/W)

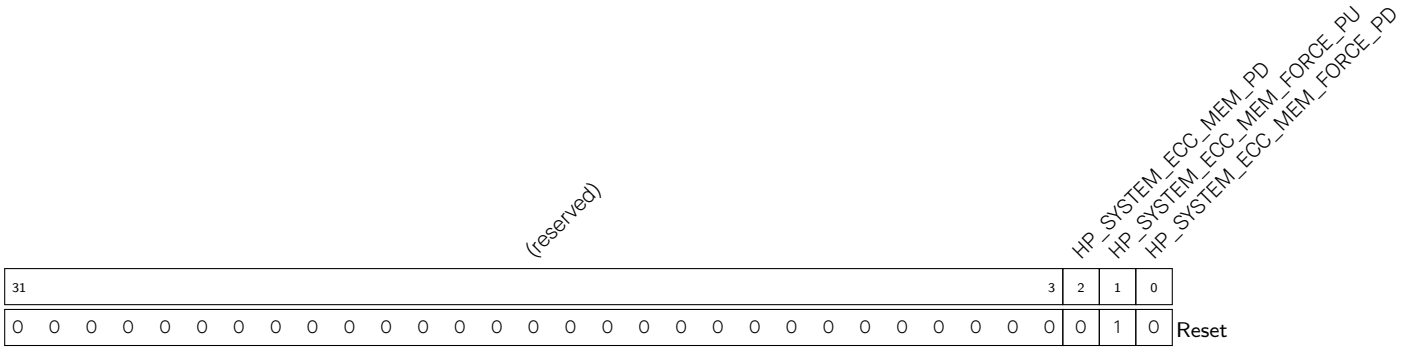
HP_SYSTEM_RSA_MEM_PD 配置是否关闭 RSA 内部存储器电源。

0：无影响

1：关闭电源

(R/W)

Register 21.53. HP_SYSTEM_HP_ECC_PD_CTRL_REG (0x01D4)



HP_SYSTEM_ECC_MEM_FORCE_PD 配置是否强制关闭 ECC 内部存储器电源。此设置的优先级低于 PMU_FORCE_TOP_RESET。

0：无影响

1：关闭电源

(R/W)

HP_SYSTEM_ECC_MEM_FORCE_PU 配置是否强制开启 ECC 内部存储器电源。

0：无影响

1：开启电源

(R/W)

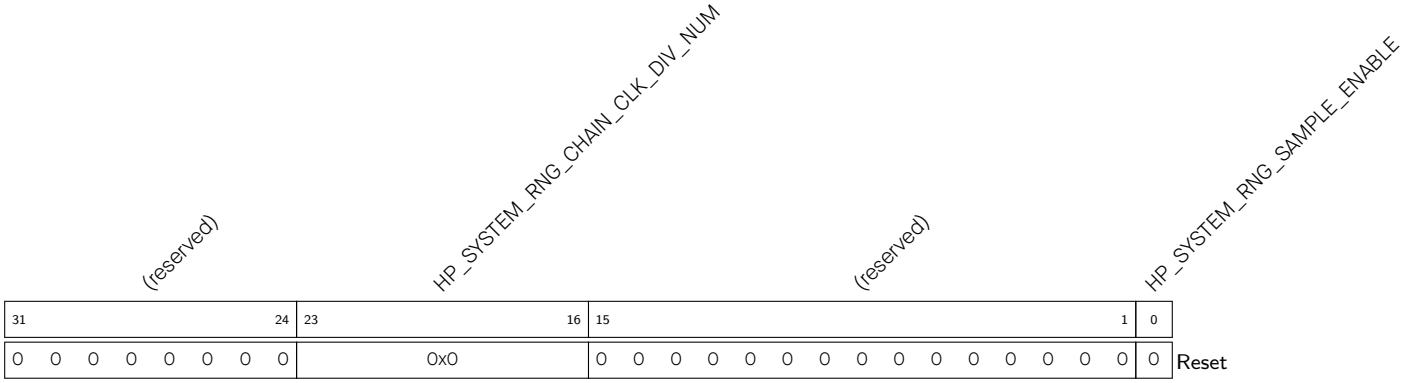
HP_SYSTEM_ECC_MEM_PD 配置是否关闭 ECC 内部存储器电源。

0：无影响

1：关闭电源

(R/W)

Register 21.54. HP_SYSTEM_HP_RNG_CFG_REG (0x01D8)



HP_SYSTEM_RNG_SAMPLE_ENABLE 配置是否启用 RNG 采样链。

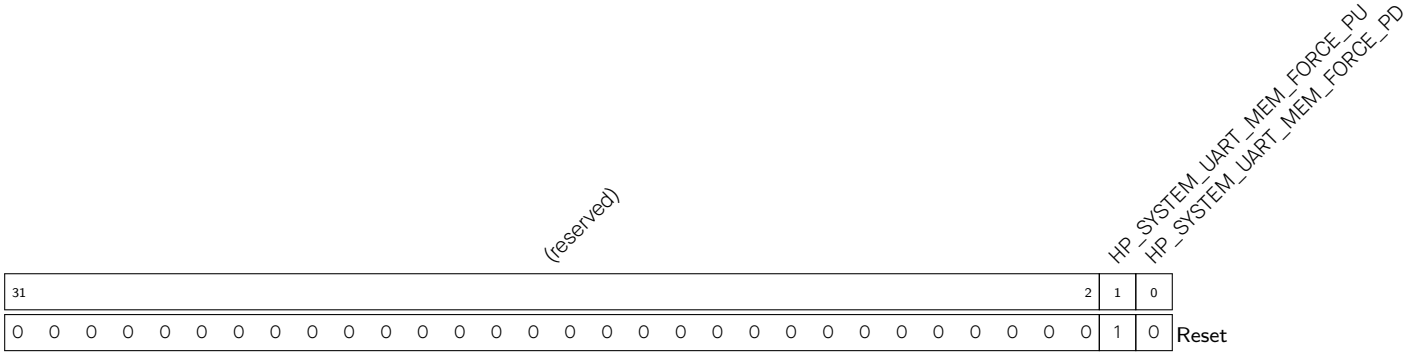
0：禁用

1：启用

(R/W)

HP_SYSTEM_RNG_CHAIN_CLK_DIV_NUM 配置用于调试的链时钟分频数。(R/W)

Register 21.55. HP_SYSTEM_HP_UART_PD_CTRL_REG (0x01DC)



HP_SYSTEM_UART_MEM_FORCE_PD 配置是否强制关闭 HP UART 内部存储器电源。

0：无影响

1：关闭电源

(R/W)

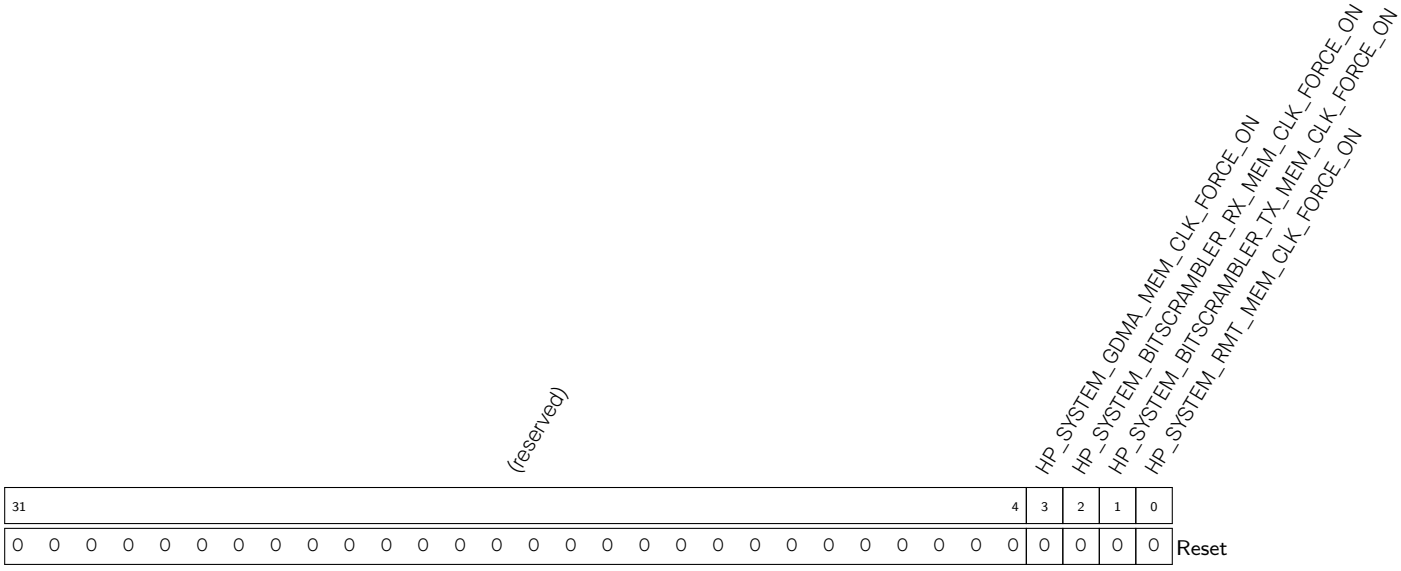
HP_SYSTEM_UART_MEM_FORCE_PU 配置是否强制开启 HP UART 内部存储器电源。

0：无影响

1：开启电源

(R/W)

Register 21.56. HP_SYSTEM_HP_PERI_MEM_CLK_FORCE_ON_REG (0x01E0)



HP_SYSTEM_RMT_MEM_CLK_FORCE_ON 配置是否强制开启 RMT 中的内存时钟门控。

0：无影响

1：强制开启

(R/W)

HP_SYSTEM_BITSCRAMBLER_TX_MEM_CLK_FORCE_ON 配置是否强制开启比特调节器中的 TX 内存时钟门控。

0：无影响

1：强制开启

(R/W)

HP_SYSTEM_BITSCRAMBLER_RX_MEM_CLK_FORCE_ON 配置是否强制开启比特调节器中的 RX 内存时钟门控。

0：无影响

1：强制开启

(R/W)

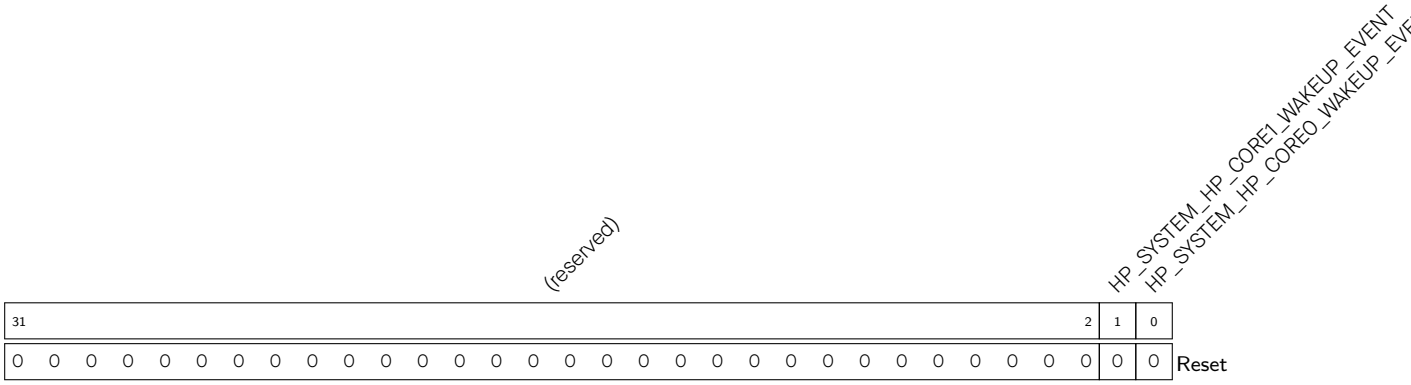
HP_SYSTEM_GDMA_MEM_CLK_FORCE_ON 配置是否强制开启 GDMA 中的内存时钟门控。

0：无影响

1：强制开启

(R/W)

Register 21.57. HP_SYSTEM_HP_CPU_WAKEUP_EVENT_REG (0x01EC)



HP_SYSTEM_HP_CORE0_WAKEUP_EVENT 配置是否唤醒 HP CPU0。

0：无作用

1：唤醒

(R/W)

HP_SYSTEM_HP_CORE1_WAKEUP_EVENT 配置是否唤醒 HP CPU1。

0：无作用

1：唤醒

(R/W)

Register 21.58. HP_SYSTEM_HP2LP_INTR_GROUP0_EN_REG (0x01F0)



HP_SYSTEM_H2LP_INTR_GROUP0_EN 写 1 使能对应外设到 LP CPU 的中断。(R/W)

Register 21.59. HP_SYSTEM_HP2LP_INTR_GROUP1_EN_REG (0x01F4)



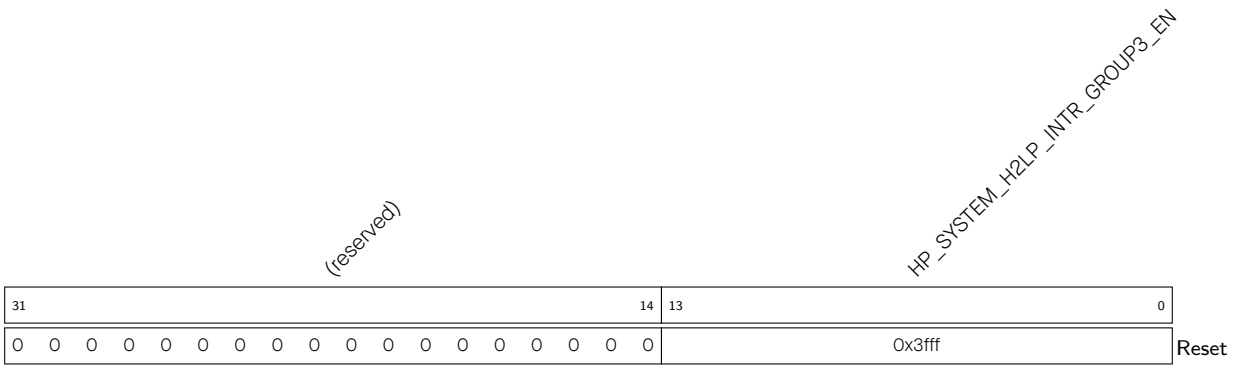
HP_SYSTEM_H2LP_INTR_GROUP1_EN 写 1 使能对应外设到 LP CPU 的中断。(R/W)

Register 21.60. HP_SYSTEM_HP2LP_INTR_GROUP2_EN_REG (0x01F8)



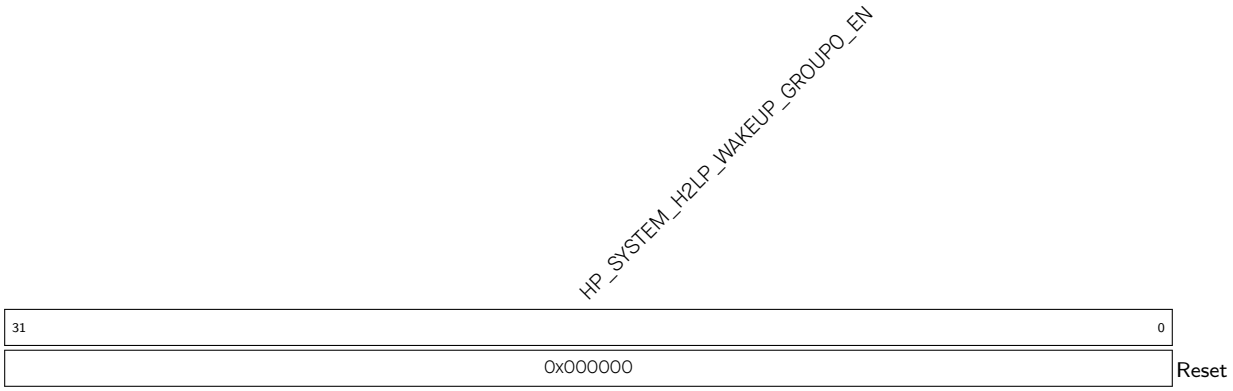
HP_SYSTEM_H2LP_INTR_GROUP2_EN 写 1 使能对应外设到 LP CPU 的中断。(R/W)

Register 21.61. HP_SYSTEM_HP2LP_INTR_GROUP3_EN_REG (0x01FC)



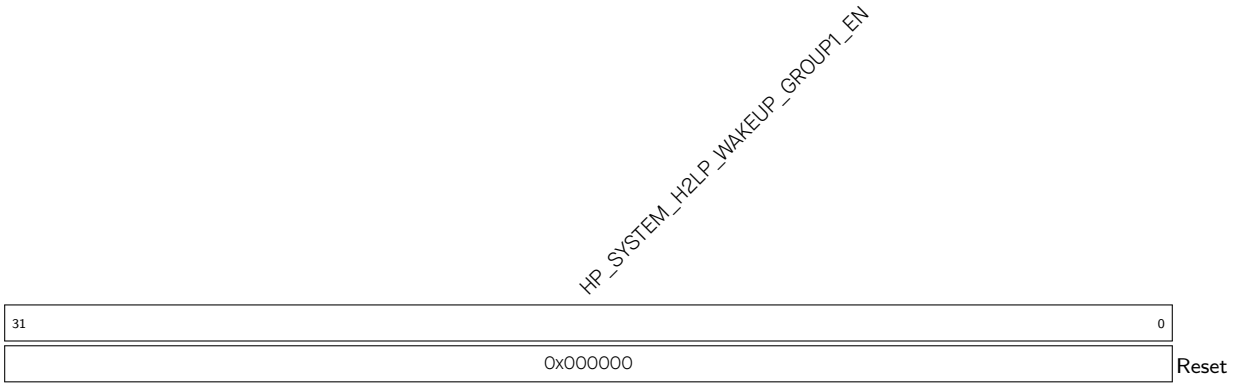
HP_SYSTEM_H2LP_INTR_GROUP3_EN 写 1 使能对应外设到 LP CPU 的中断。(R/W)

Register 21.62. HP_SYSTEM_HP2LP_WAKEUP_GROUP0_EN_REG (0x0210)



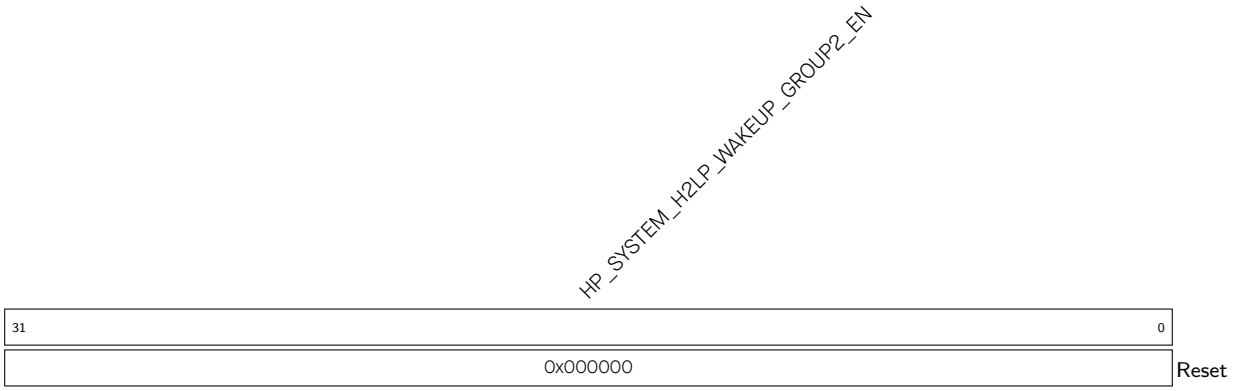
HP_SYSTEM_H2LP_WAKEUP_GROUP0_EN 写 1 使能对应外设到 PMU 的唤醒。(R/W)

Register 21.63. HP_SYSTEM_HP2LP_WAKEUP_GROUP1_EN_REG (0x0214)



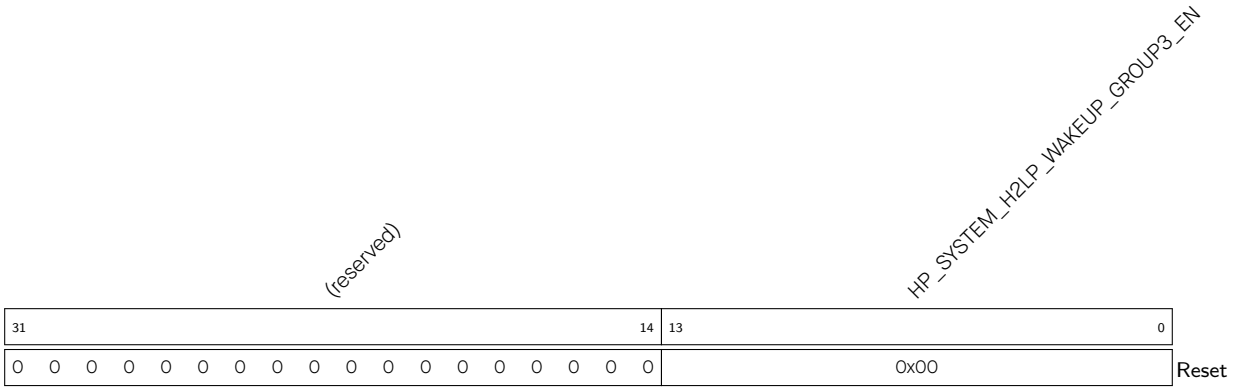
HP_SYSTEM_H2LP_WAKEUP_GROUP1_EN 写 1 使能对应外设到 PMU 的唤醒。(R/W)

Register 21.64. HP_SYSTEM_HP2LP_WAKEUP_GROUP2_EN_REG (0x0218)



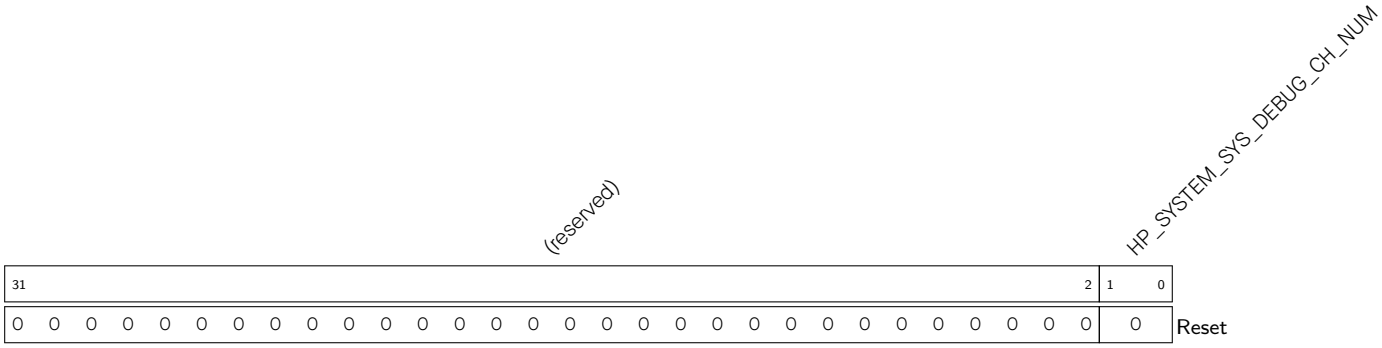
HP_SYSTEM_H2LP_WAKEUP_GROUP2_EN 写 1 使能对应外设到 PMU 的唤醒。(R/W)

Register 21.65. HP_SYSTEM_HP2LP_WAKEUP_GROUP3_EN_REG (0x021C)



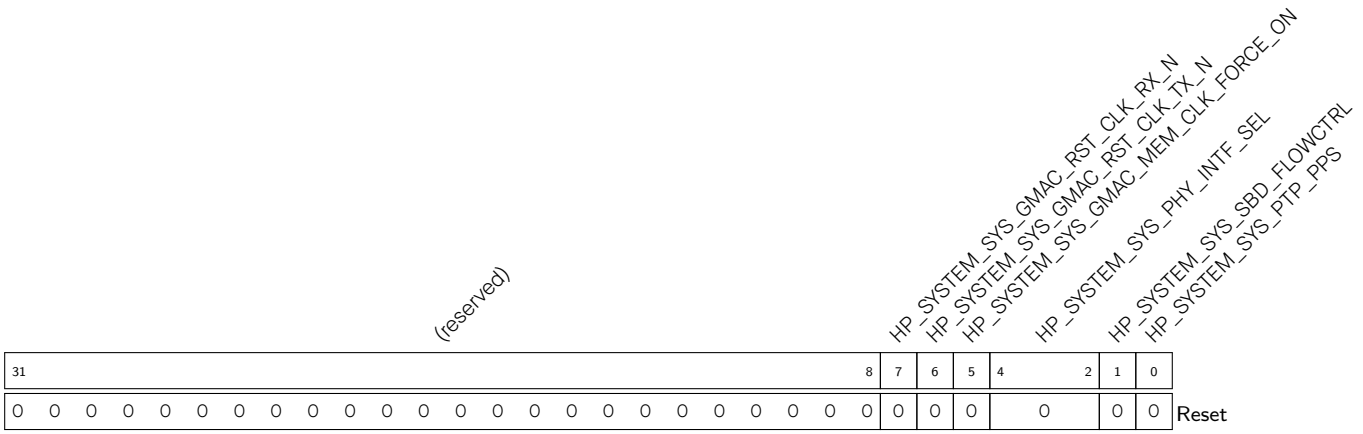
HP_SYSTEM_H2LP_WAKEUP_GROUP3_EN 写 1 使能对应外设到 PMU 的唤醒。(R/W)

Register 21.66. HP_SYSTEM_GDMA_CTRL_REG (0x0148)



HP_SYSTEM_SYS_DEBUG_CH_NUM 配置 GDMA 调试通道的数量。(R/W)

Register 21.67. HP_SYSTEM_GMAC_CTRL0_REG (0x014C)



HP_SYSTEM_SYS_PTP_PPS 表示以太网 MAC 的每秒脉冲输出。(RO)

HP_SYSTEM_SYS_SBD_FLOWCTRL 配置旁路流量控制信号。

0：无影响

1：指示以太网 MAC 在全双工模式下发送暂停帧，并在半双工模式下启用背压。

(R/W)

HP_SYSTEM_SYS_PHY_INTF_SEL 配置以太网 PHY 接口。

0：MII

1：无效值。没有作用

2：无效值。没有作用

3：无效值。没有作用

4：RMII

5：无效值。没有作用

6：无效值。没有作用

7：无效值。没有作用

(R/W)

HP_SYSTEM_SYS_GMAC_MEM_CLK_FORCE_ON 配置是否强制启用 GMAC 内存的内存时钟。

0：无影响

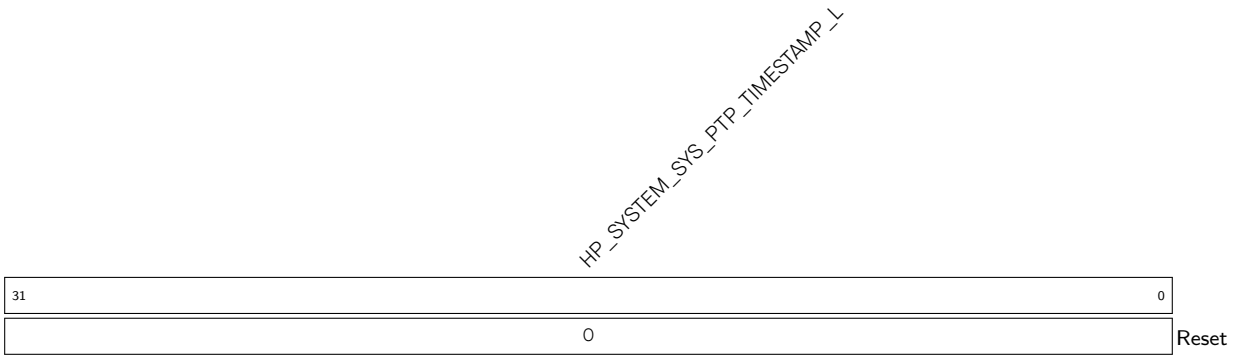
1：强制启用

(R/W)

HP_SYSTEM_SYS_GMAC_RST_CLK_TX_N 表示与以太网 MAC 发送时钟同步的低电平有效复位信号。(RO)

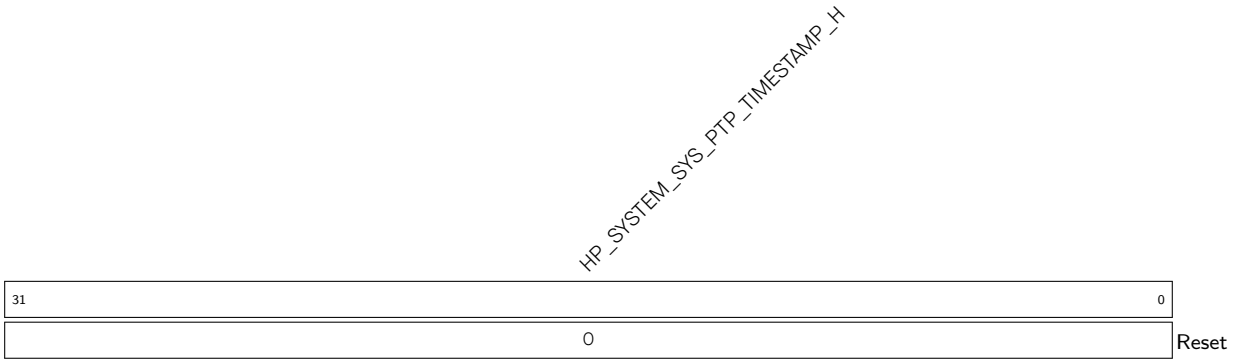
HP_SYSTEM_SYS_GMAC_RST_CLK_RX_N 表示与以太网 MAC 接收时钟同步的低电平有效复位信号。(RO)

Register 21.68. HP_SYSTEM_GMAC_CTRL1_REG (0x0150)



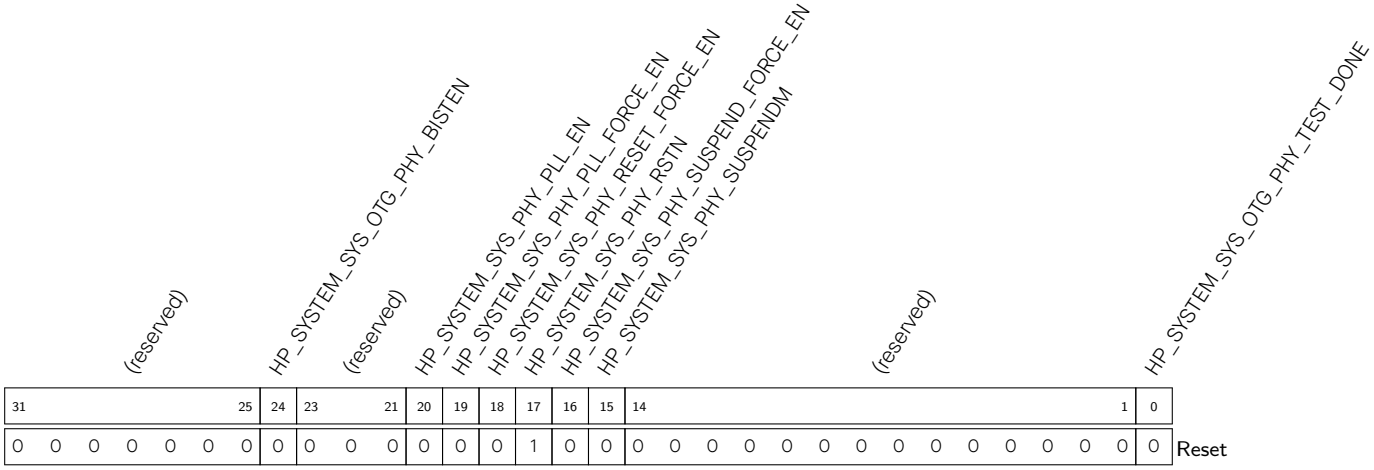
HP_SYSTEM_SYS_PTP_TIMESTAMP_L 表示从以太网 MAC 同步至 clk_ptp_ref 时钟的时间戳输出的低 32 位。(RO)

Register 21.69. HP_SYSTEM_GMAC_CTRL2_REG (0x0154)



HP_SYSTEM_SYS_PTP_TIMESTAMP_H 表示从以太网 MAC 同步至 clk_ptp_ref 时钟的时间戳输出的高 32 位。(RO)

Register 21.70. HP_SYSTEM_USBOTG20_CTRL_REG (0x015C)



HP_SYSTEM_SYS_OTG_PHY_TEST_DONE 表示 USB OTG_HS PHY BIST 测试的状态。

0：失败

1：通过

(RO)

HP_SYSTEM_SYS_PHY_SUSPENDM 配置是否使 USB OTG_HS PHY 进入 Suspend 模式。仅当 HP_SYSTEM_PHY_SUSPEND_FORCE_EN 设置为 1 时有效。

0：无影响

1：进入 Suspend 模式

(R/W)

HP_SYSTEM_SYS_PHY_SUSPEND_FORCE_EN 配置是否允许通过设置 HP_SYSTEM_PHY_SUSPENDM 强制使 USB OTG_HS PHY 进入 Suspend 模式。

0：无影响

1：允许

(R/W)

HP_SYSTEM_SYS_PHY_RSTN 配置是否复位 USB OTG_HS PHY。仅当 HP_SYSTEM_PHY_RESET_FORCE_EN 设置为 1 时有效。

0：复位

1：无作用

(R/W)

HP_SYSTEM_SYS_PHY_RESET_FORCE_EN 配置是否允许通过清除 HP_SYSTEM_PHY_RSTN 强制复位 USB OTG_HS PHY。

0：无影响

1：强制复位

(R/W)

见下页...

Register 21.70. HP_SYSTEM_USBOTG20_CTRL_REG (0x015C)

接上页...

HP_SYSTEM_SYS_PHY_PLL_FORCE_EN 配置是否允许通过设置 HP_SYSTEM_PHY_PLL_EN 强制启用 USB OTG_HS PHY PLL。

0：无影响

1：强制启用

(R/W)

HP_SYSTEM_SYS_PHY_PLL_EN 配置是否启用 USB OTG_HS PHY PLL。仅当 HP_SYSTEM_PHY_PLL_FORCE_EN 设置时有效。

0：无影响

1：启用

(R/W)

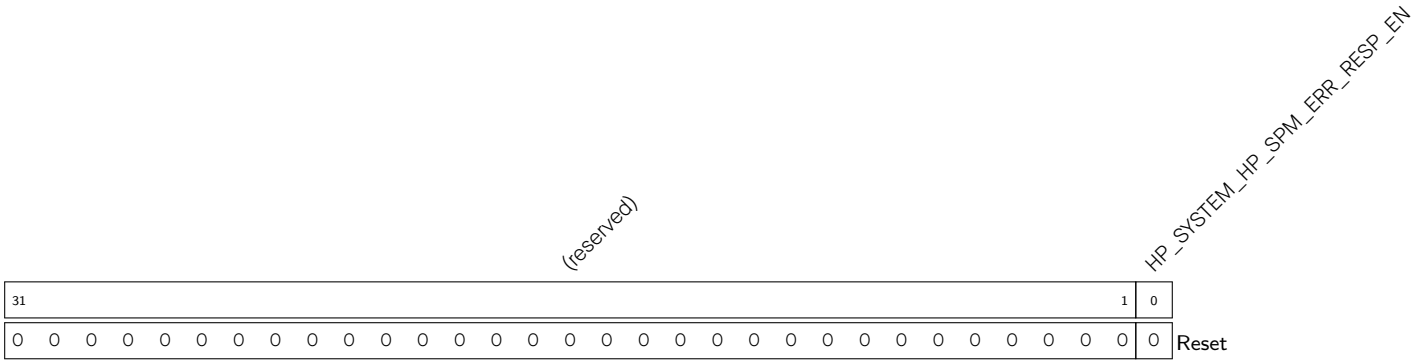
HP_SYSTEM_SYS_OTG_PHY_BISTEN 配置是否使 USB OTG_HS PHY 进入 BIST 模式并执行自检。

0：无影响

1：进入 BIST 模式

(R/W)

Register 21.71. HP_SYSTEM_HP_SPM_ERR_RESP_CTRL_REG (0x0160)



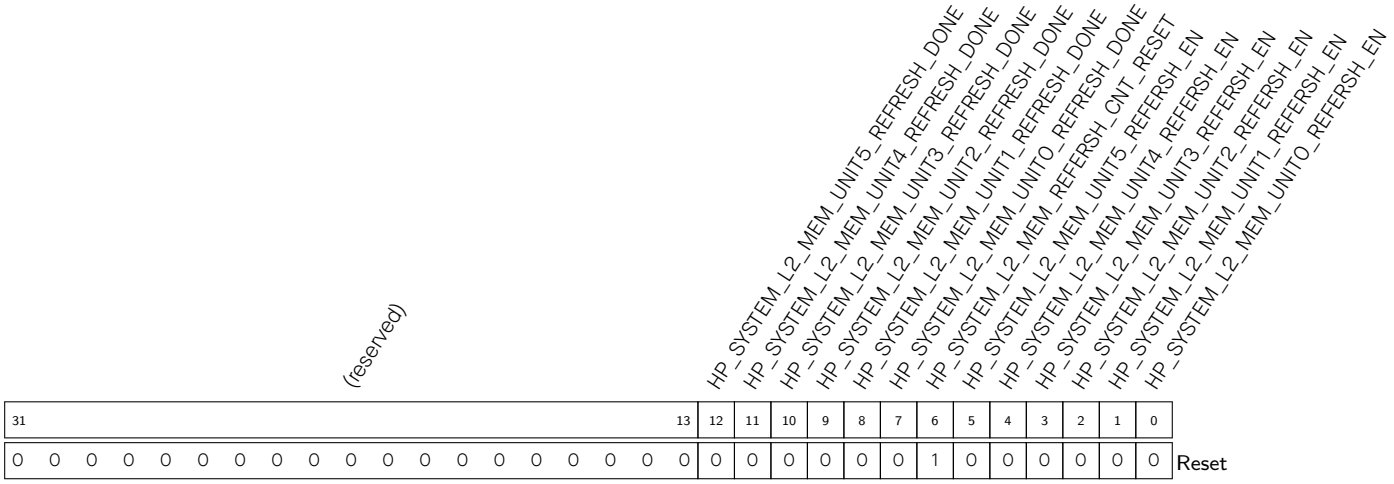
HP_SYSTEM_HP_SPM_ERR_RESP_EN 配置是否向 HP CPU 报告奇偶校验错误并触发异常。

0：禁用

1：启用

(R/W)

Register 21.72. HP_SYSTEM_L2_MEM_REFRESH_REG (0x0164)



HP_SYSTEM_L2_MEM_UNIT0_REFERSH_EN 配置是否刷新 L2MEM unit0 的 ECC 位。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_MEM_UNIT1_REFERSH_EN 配置是否刷新 L2MEM unit1 的 ECC 位。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_MEM_UNIT2_REFERSH_EN 配置是否刷新 L2MEM unit2 的 ECC 位。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_MEM_UNIT3_REFERSH_EN 配置是否刷新 L2MEM unit3 的 ECC 位。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_MEM_UNIT4_REFERSH_EN 配置是否刷新 L2MEM unit4 的 ECC 位。

0：禁用

1：启用

(R/W)

HP_SYSTEM_L2_MEM_UNIT5_REFERSH_EN 配置是否刷新 L2MEM unit5 的 ECC 位。

0：禁用

1：启用

(R/W)

见下页...

Register 21.72. HP_SYSTEM_L2_MEM_REFRESH_REG (0x0164)

接上页...

HP_SYSTEM_L2_MEM_REFERSH_CNT_RESET 配置是否复位 L2MEM 刷新计数。

0：无影响

1：复位

(R/W)

HP_SYSTEM_L2_MEM_UNIT0_REFRESH_DONE 表示 L2MEM unit0 ECC 位的刷新状态。(RO)

HP_SYSTEM_L2_MEM_UNIT1_REFRESH_DONE 表示 L2MEM unit1 ECC 位的刷新状态。(RO)

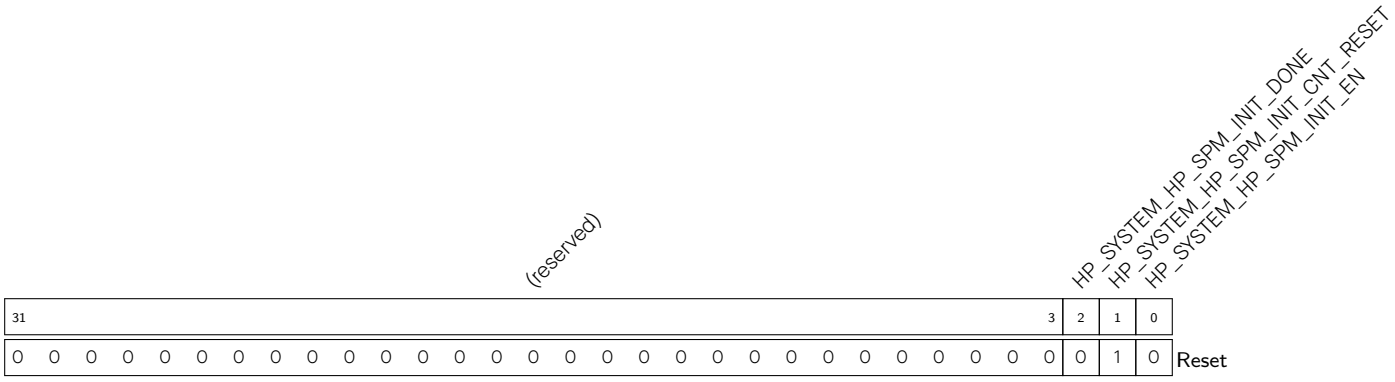
HP_SYSTEM_L2_MEM_UNIT2_REFRESH_DONE 表示 L2MEM unit2 ECC 位的刷新状态。(RO)

HP_SYSTEM_L2_MEM_UNIT3_REFRESH_DONE 表示 L2MEM unit3 ECC 位的刷新状态。(RO)

HP_SYSTEM_L2_MEM_UNIT4_REFRESH_DONE 表示 L2MEM unit4 ECC 位的刷新状态。(RO)

HP_SYSTEM_L2_MEM_UNIT5_REFRESH_DONE 表示 L2MEM unit5 ECC 位的刷新状态。(RO)

Register 21.73. HP_SYSTEM_HP_SPM_INIT_REG (0x0168)



HP_SYSTEM_HP_SPM_INIT_EN 配置是否启用 HP SPM 奇偶校验位初始化。

0：禁用

1：启用

(R/W)

HP_SYSTEM_HP_SPM_INIT_CNT_RESET 配置是否复位 SPM 初始化计数。

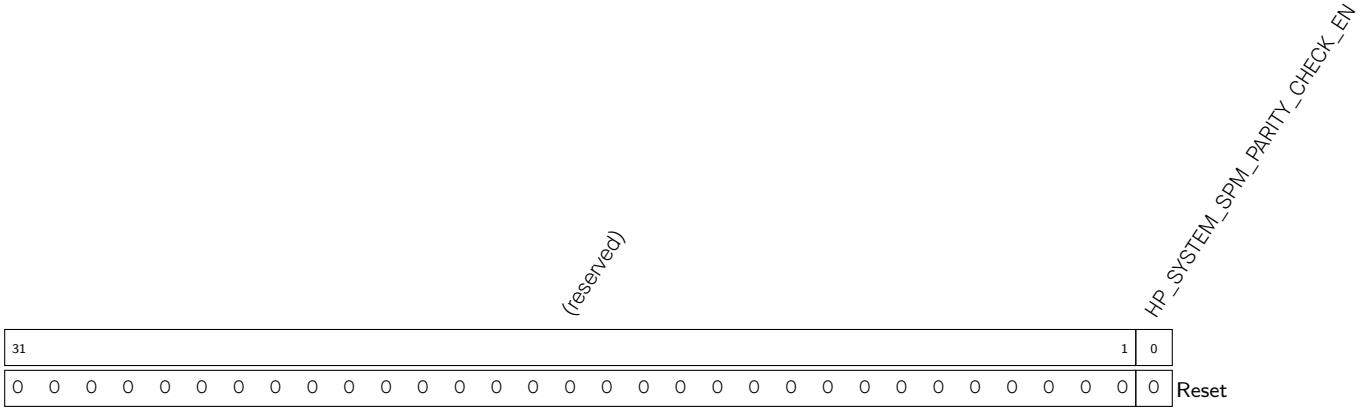
0：无影响

1：复位

(R/W)

HP_SYSTEM_HP_SPM_INIT_DONE 表示 HP SPM 奇偶校验位的初始化状态。(RO)

Register 21.74. HP_SYSTEM_HP_SPM_PARITY_CHECK_CTRL_REG (0x016C)



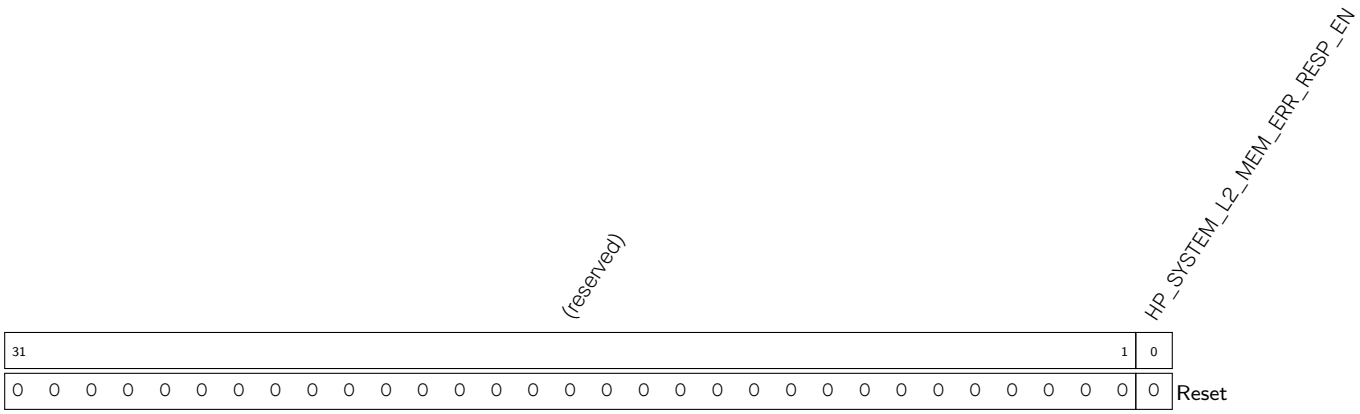
HP_SYSTEM_HP_SPM_PARITY_CHECK_EN 配置是否启用 HP SPM 的奇偶校验错误检查。

0：禁用

1：启用

(R/W)

Register 21.75. HP_SYSTEM_L2_MEM_ERR_RESP_CTRL_REG (0x0198)



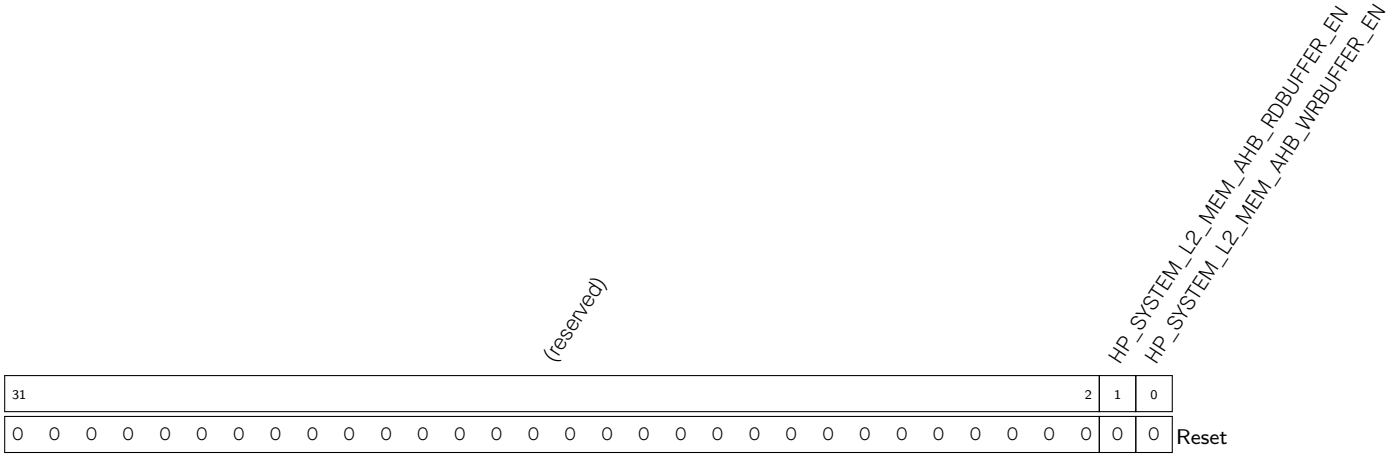
HP_SYSTEM_L2_MEM_ERR_RESP_EN 配置是否向 HP CPU 报告 ECC 错误并触发异常。

0：禁用

1：启用

(R/W)

Register 21.76. HP_SYSTEM_L2_MEM_AHB_BUFFER_CTRL_REG (0x019C)



HP_SYSTEM_L2_MEM_AHB_WRBUFFER_EN 配置是否启用 L2MEM AHB 写缓冲。

0：禁用

1：启用

(R/W)

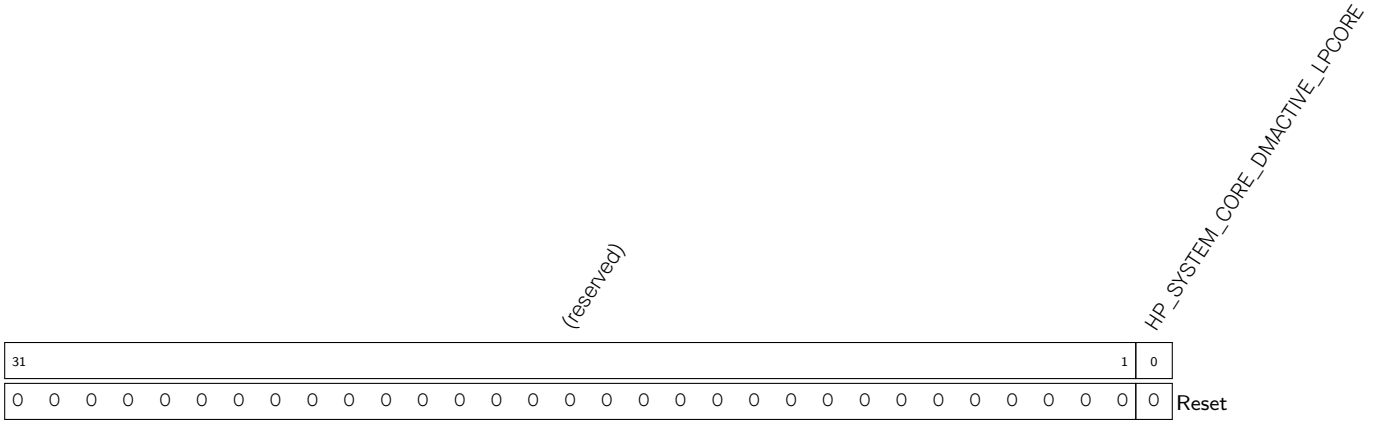
HP_SYSTEM_L2_MEM_AHB_RDBUFFER_EN 配置是否启用 L2MEM AHB 读缓冲。

0：禁用

1：启用

(R/W)

Register 21.77. HP_SYSTEM_HP_CORE_DMACTIVE_LPCORE_REG (0x01A0)



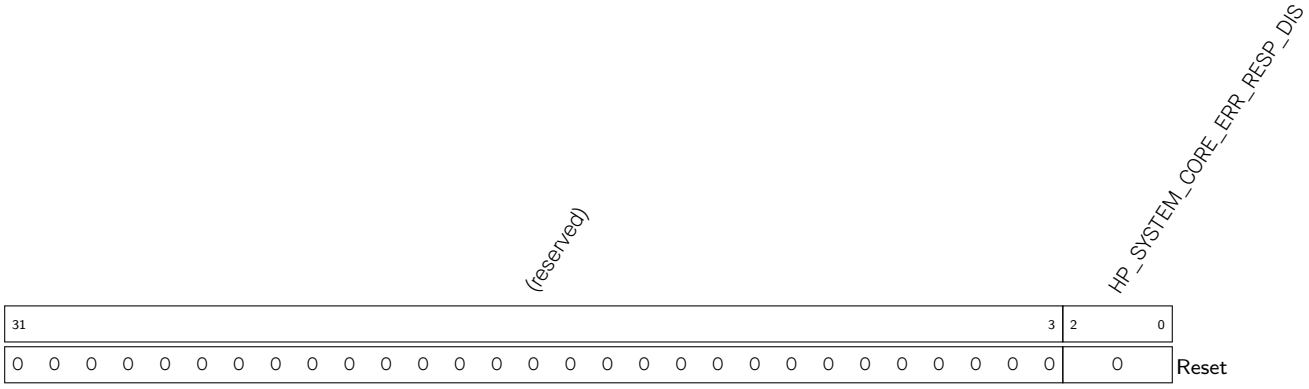
HP_SYSTEM_CORE_DMACTIVE_LPCORE 表示 LP CPU 调试模块的 dmactive 值。

0：表示 LP CPU 未连接到 JTAG 接口，或未处于调试模式。

1：表示 LP CPU 已连接到 JTAG 接口，并处于调试模式。

(RO)

Register 21.78. HP_SYSTEM_HP_CORE_ERR_RESP_DIS_REG (0x01A4)



HP_SYSTEM_CORE_ERR_RESP_DIS 配置是否禁用对应总线的错误响应。

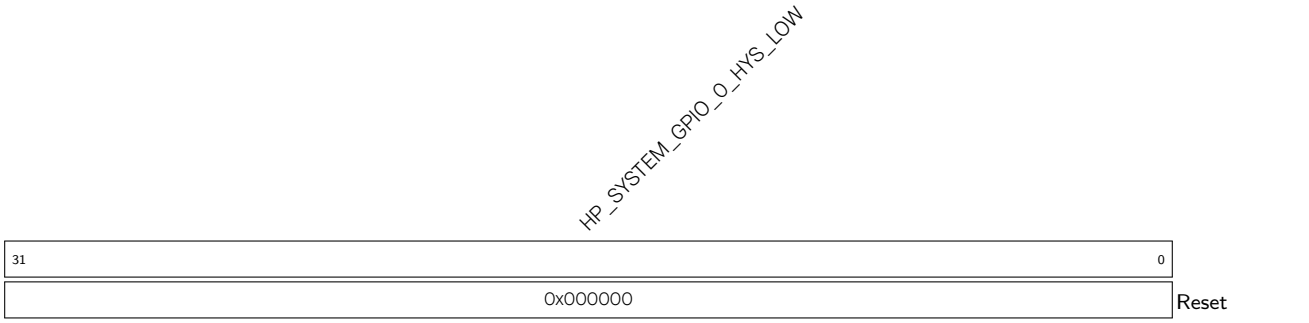
位 [2]：AHB 总线

位 [1]：DBUS

位 [0]：IBUS

(R/W)

Register 21.79. HP_SYSTEM_GPIO_O_HYS_CTRL0_REG (0x01C0)



HP_SYSTEM_GPIO_O_HYS_LOW 配置是否使能 HP GPIO16 ~ HP GPIO47 的迟滞功能。

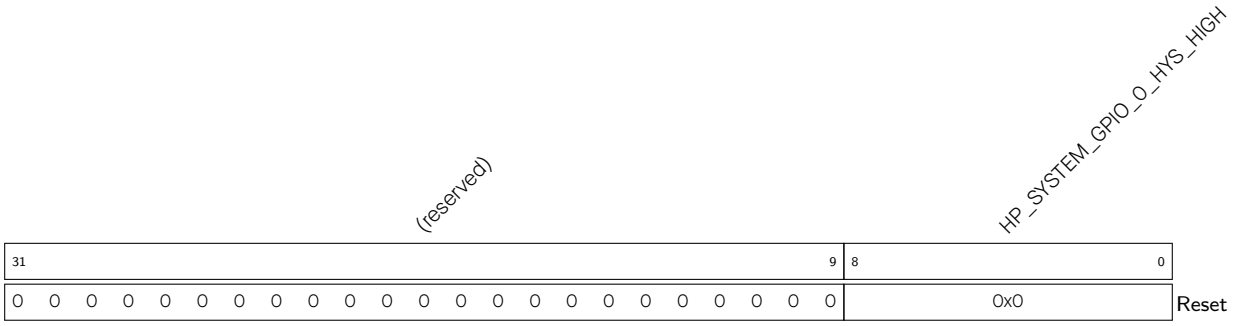
0：无影响

1：使能

详见章节 10 *GPIO 交换矩阵和 IO MUX* 中的 10.10 小节。

(R/W)

Register 21.80. HP_SYSTEM_GPIO_O_HYS_CTRL1_REG (0x01C4)



HP_SYSTEM_GPIO_O_HYS_HIGH 配置是否使能 HP GPIO48 ~ HP GPIO56 的迟滞功能。

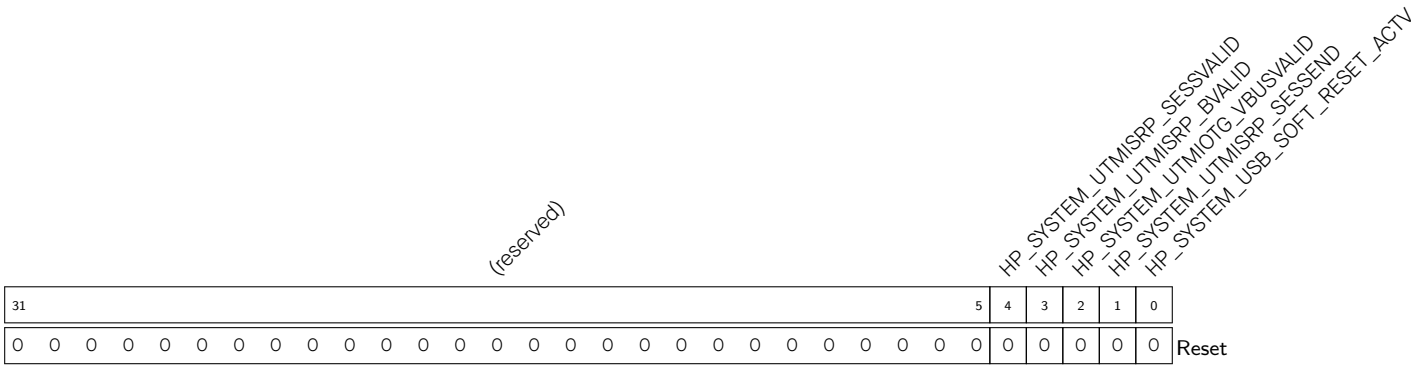
0：无影响

1：使能

详见章节 10 *GPIO 交换矩阵和 IO MUX* 中的 10.10 小节。

(R/W)

Register 21.81. HP_SYSTEM_USB_OTGHS_PHY_ST_REG (0x01E8)



HP_SYSTEM_USB_SOFT_RESET_ACTV_PDOMAIN 表示电源域中 USB 软复位激活状态。(RO)

HP_SYSTEM_UTMISRP_SESSEND 表示 USB OTG2.0 会话结束状态。(RO)

HP_SYSTEM_UTMIOTG_VBUSVALID 表示 USB OTG2.0 VBUS 有效状态。(RO)

HP_SYSTEM_UTMISRP_BVALID 表示 USB OTG2.0 B-valid 状态。(RO)

HP_SYSTEM_UTMISRP_SESSVALID 表示 USB OTG2.0 会话有效状态。(RO)

Register 21.82. HP_SYSTEM_HP2LP_INTR_GROUP0_ST_REG (0x0200)



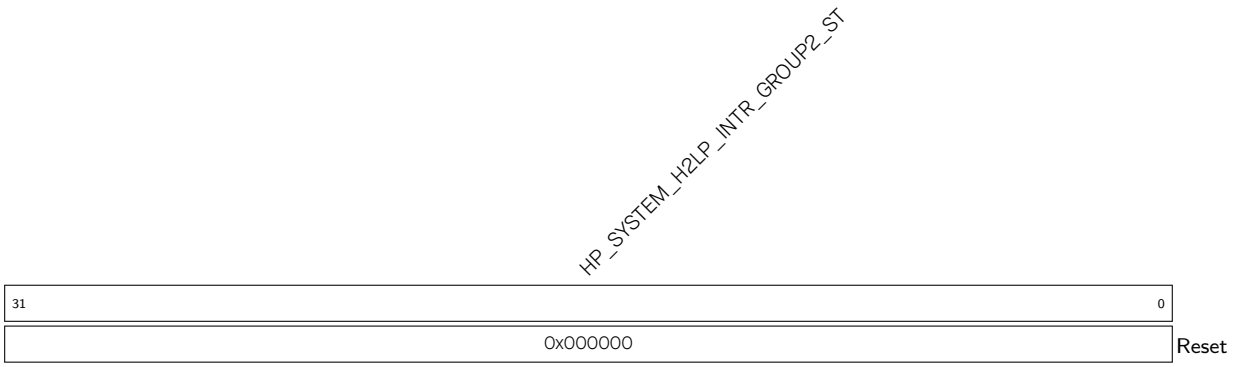
HP_SYSTEM_H2LP_INTR_GROUP0_ST 每一位表示对应外设到 LP CPU 的中断状态。(RO)

Register 21.83. HP_SYSTEM_HP2LP_INTR_GROUP1_ST_REG (0x0204)



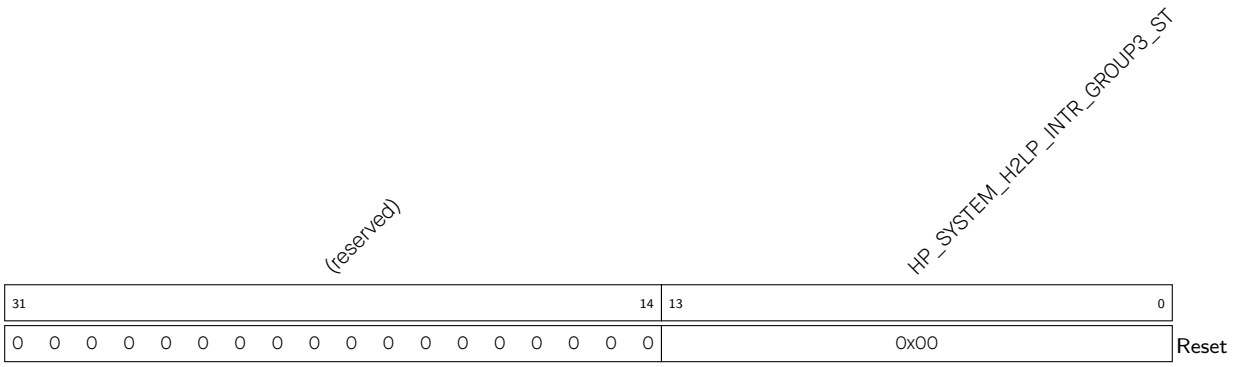
HP_SYSTEM_H2LP_INTR_GROUP1_ST 每一位表示对应外设到 LP CPU 的中断状态。(RO)

Register 21.84. HP_SYSTEM_HP2LP_INTR_GROUP2_ST_REG (0x0208)



HP_SYSTEM_H2LP_INTR_GROUP2_ST 每一位表示对应外设到 LP CPU 的中断状态。(RO)

Register 21.85. HP_SYSTEM_HP2LP_INTR_GROUP3_ST_REG (0x020C)



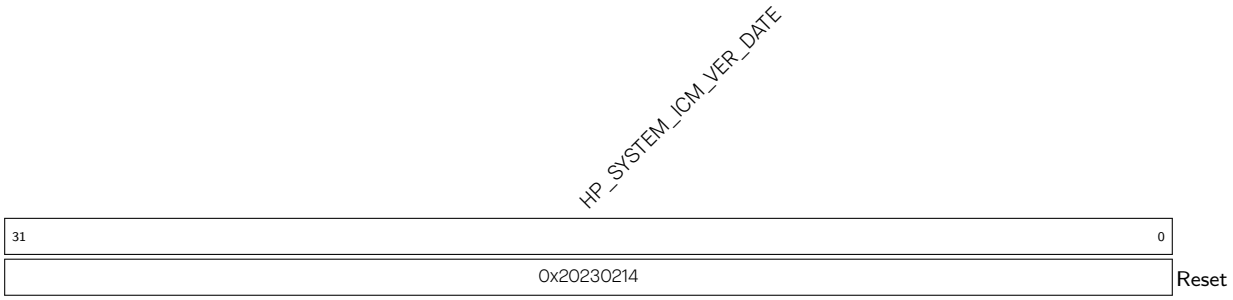
HP_SYSTEM_H2LP_INTR_GROUP3_ST 每一位表示对应外设到 LP CPU 的中断状态。(RO)

21.5.2 ICM Register

本小节的所有地址均为相对于 AXI ICM 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

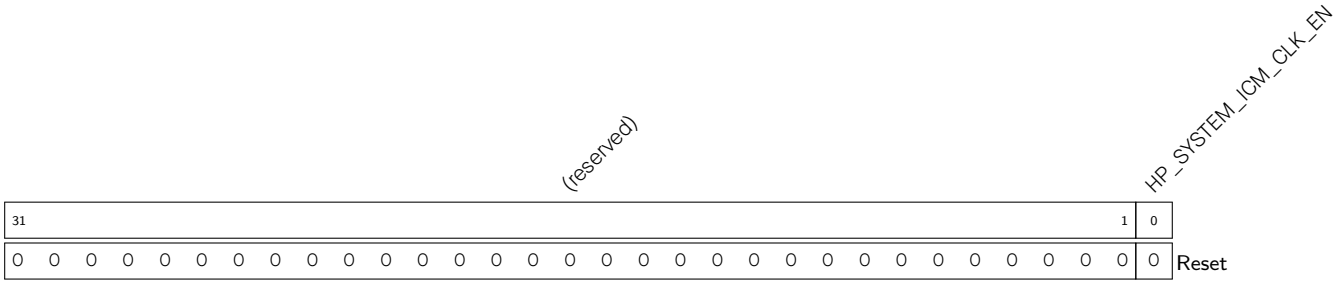
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 21.86. HP_SYSTEM_ICM_VER_DATE_REG (0x0000)



HP_SYSTEM_ICM_VER_DATE 版本控制寄存器。(R/W)

Register 21.87. HP_SYSTEM_ICM_CLK_EN_REG (0x0004)



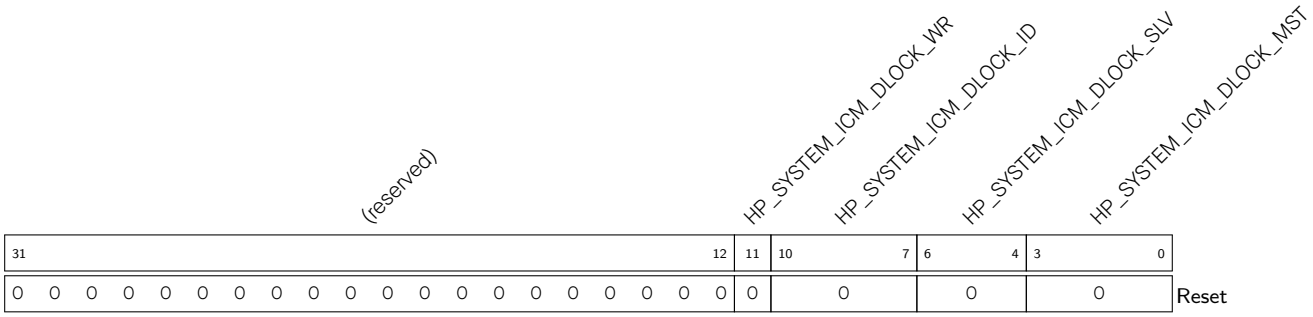
HP_SYSTEM_ICM_CLK_EN 配置是否强制开启寄存器时钟。

0：时钟仅在寄存器访问期间启用

1：强制开启寄存器时钟

(R/W)

Register 21.88. HP_SYSTEM_ICM_DLOCK_STATUS_REG (0x0008)



HP_SYSTEM_ICM_DLOCK_MST 表示 deadlock 中最低编号的主机。(RO)

HP_SYSTEM_ICM_DLOCK_SLV 表示发生 deadlock 的从机。(RO)

HP_SYSTEM_ICM_DLOCK_ID 表示 deadlock 事件的 AXI ID。(RO)

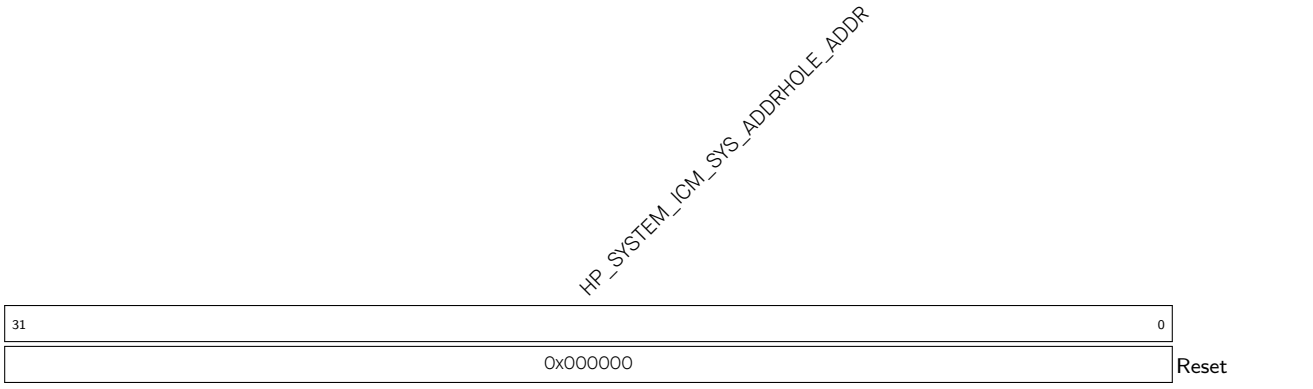
HP_SYSTEM_ICM_DLOCK_WR 表示 deadlock 事件是否为写操作。

0：非写操作

1：写操作

(RO)

Register 21.89. HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR_REG (0x0038)



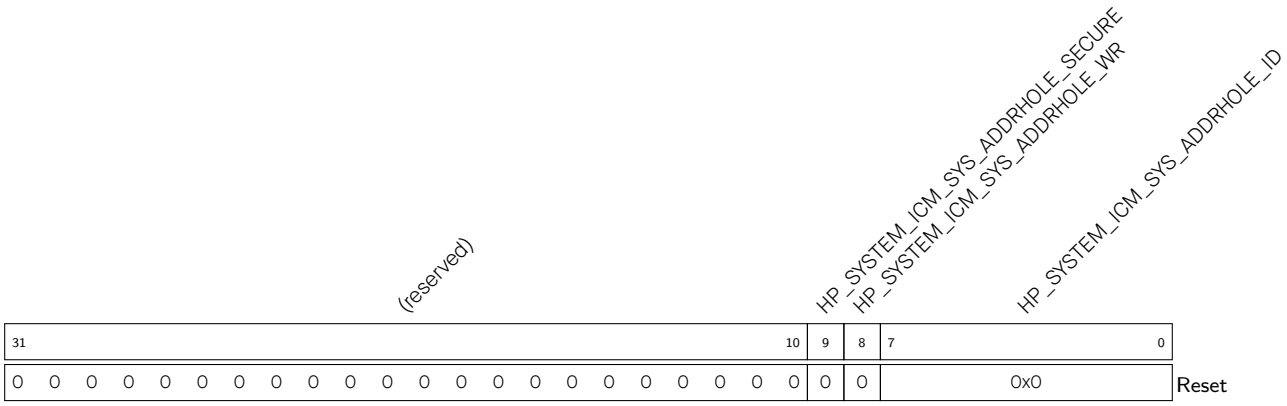
HP_SYSTEM_ICM_SYS_ADDRHOLE_ADDR 表示 ICM_SYS_ADDRHOLE_INT 发生时的地址。

若 HP_SYSTEM_ICM_SYS_ADDRHOLE_SECURE 为 0，则为未经授权的访问地址。

若 HP_SYSTEM_ICM_SYS_ADDRHOLE_SECURE 为 1，则为非法访问地址。

(RO)

Register 21.90. HP_SYSTEM_ICM_SYS_ADDRHOLE_INFO_REG (0x003C)



HP_SYSTEM_ICM_SYS_ADDRHOLE_ID 表示 ICM_SYS_ADDRHOLE_INT 发生时的主机 ID。此 ID 包含 4 位 CID 和 4 位 UID（详情请参阅相关 IP）。CID 用于 ICM 中的主机验证。CID 值对应以下主机：

1：CACHE

5：GDMA MST1

6：GDMA MST2

8：AXI GDMA

10：DMA2D

11：H264 MST1

12：H264 MST2

(RO)

HP_SYSTEM_ICM_SYS_ADDRHOLE_WR 表示 ICM_SYS_ADDRHOLE_INT 发生时的传输类型。

0：读传输

1：写传输

(RO)

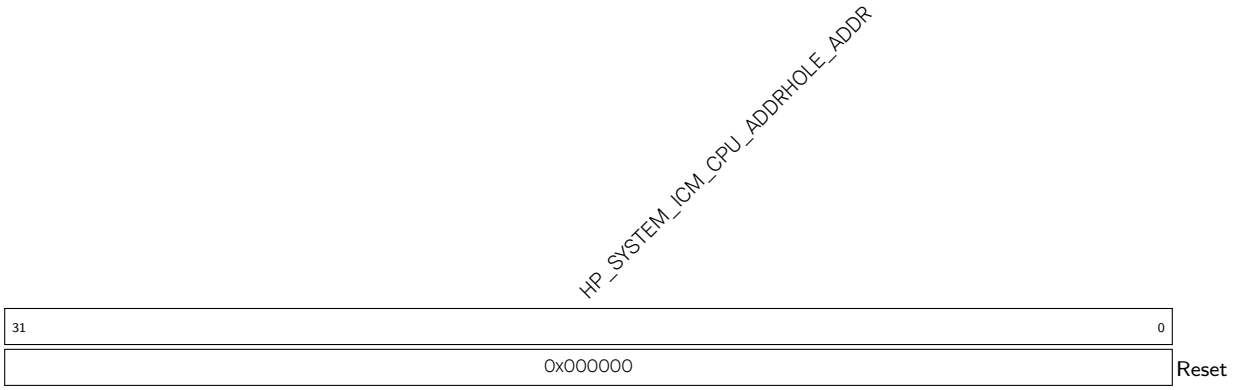
HP_SYSTEM_ICM_SYS_ADDRHOLE_SECURE 表示 ICM_SYS_ADDRHOLE_INT 发生时的访问错误类型。

0：未经授权的访问

1：非法访问

(RO)

Register 21.91. HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR_REG (0x0040)



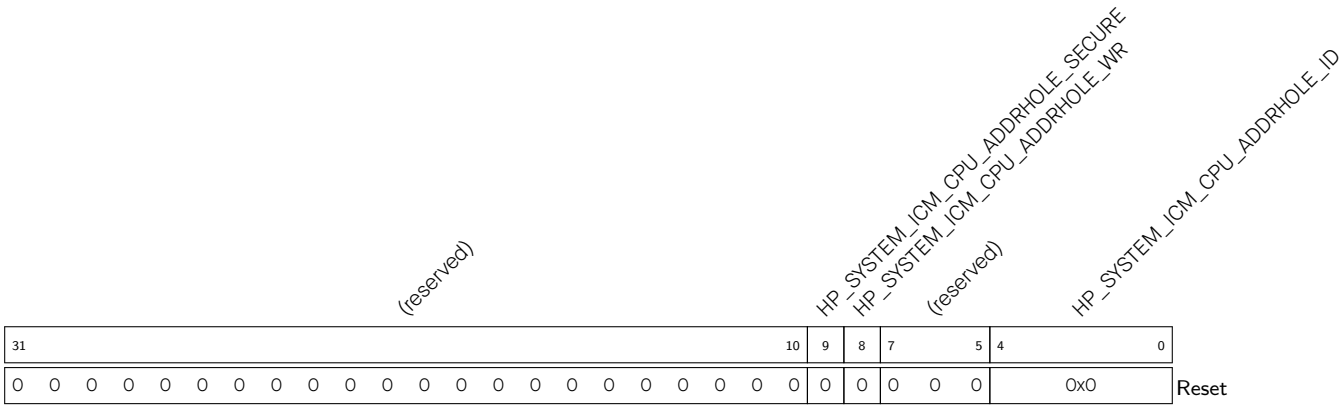
HP_SYSTEM_ICM_CPU_ADDRHOLE_ADDR 表示 ICM_CPU_ADDRHOLE_INT 发生时的地址。

若 HP_SYSTEM_ICM_CPU_ADDRHOLE_SECURE 为 0，则为未经授权的访问地址。

若 HP_SYSTEM_ICM_CPU_ADDRHOLE_SECURE 为 1，则为非法访问地址。

(RO)

Register 21.92. HP_SYSTEM_ICM_CPU_ADDRHOLE_INFO_REG (0x0044)



HP_SYSTEM_ICM_CPU_ADDRHOLE_ID 表示 ICM_CPU_ADDRHOLE_INT 发生时的主机 ID。

0：HP CPU0
1：HP CPU1
2：LP CPU
3：USBOTGFS
5：GMAC
6：SDMMC
7：USBOTGHS
8：TRACE0
9：TRACE1
10：SPM Monitor
11：L2MEM Monitor
16-31：AHB GDMA
(RO)

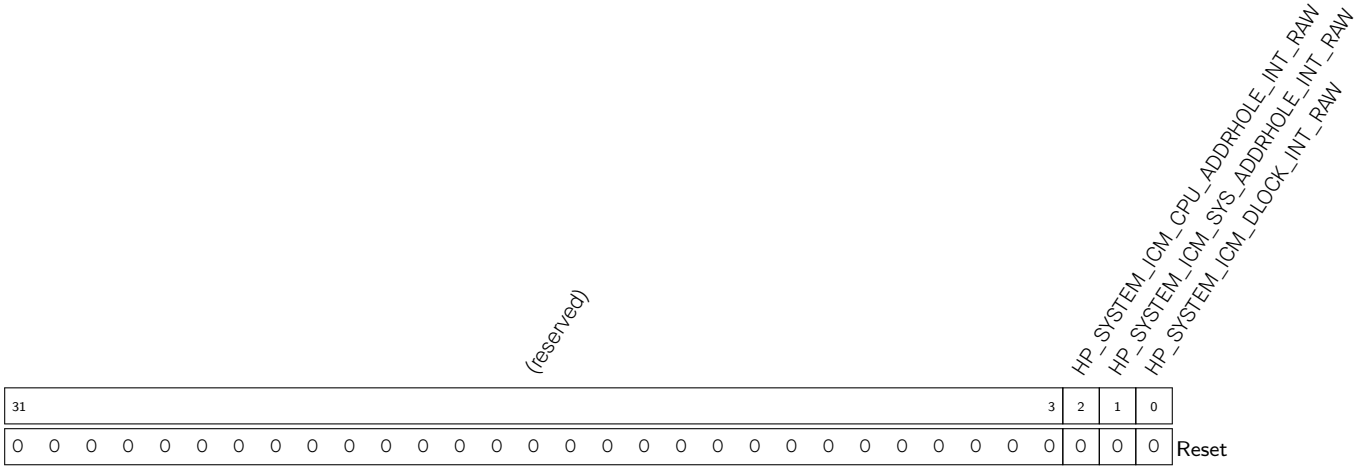
HP_SYSTEM_ICM_CPU_ADDRHOLE_WR 表示 ICM_CPU_ADDRHOLE_INT 发生时的传输类型。

0：读传输
1：写传输
(RO)

HP_SYSTEM_ICM_CPU_ADDRHOLE_SECURE 表示 ICM_CPU_ADDRHOLE_INT 发生时的访问错误类型。

0：未经授权的访问
1：非法访问
(RO)

Register 21.93. HP_SYSTEM_ICM_INT_RAW_REG (0x000C)

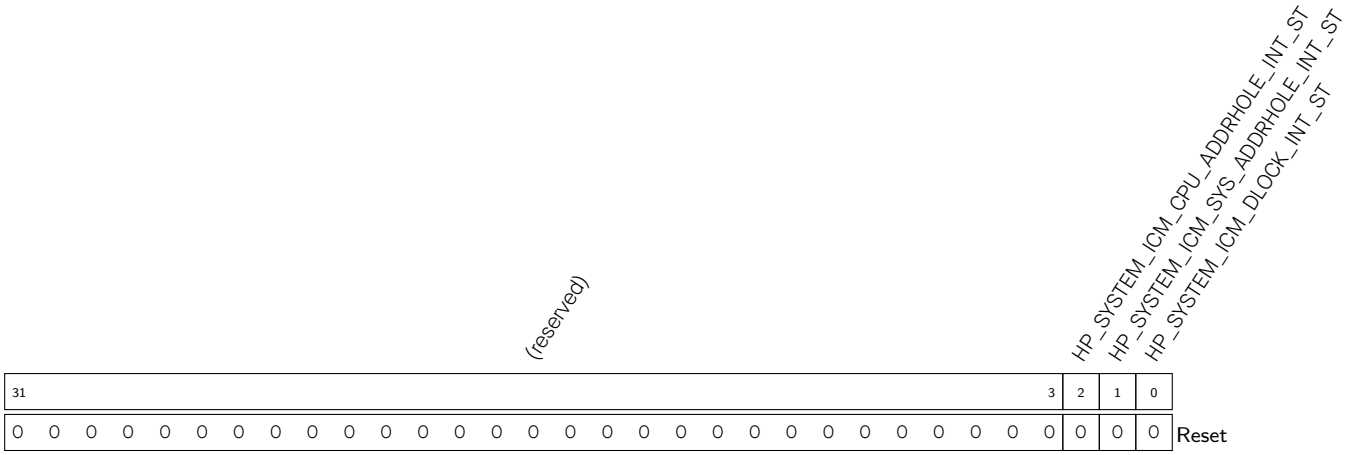


HP_SYSTEM_ICM_DLOCK_INT_RAW ICM_DLOCK_INT 的原始中断状态。(R/WTC/SS)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_RAW ICM_SYS_ADDRHOLE_INT 的原始中断状态。(R/WTC/SS)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_RAW ICM_CPU_ADDRHOLE_INT 的原始中断状态。(R/WTC/SS)

Register 21.94. HP_SYSTEM_ICM_INT_ST_REG (0x0010)

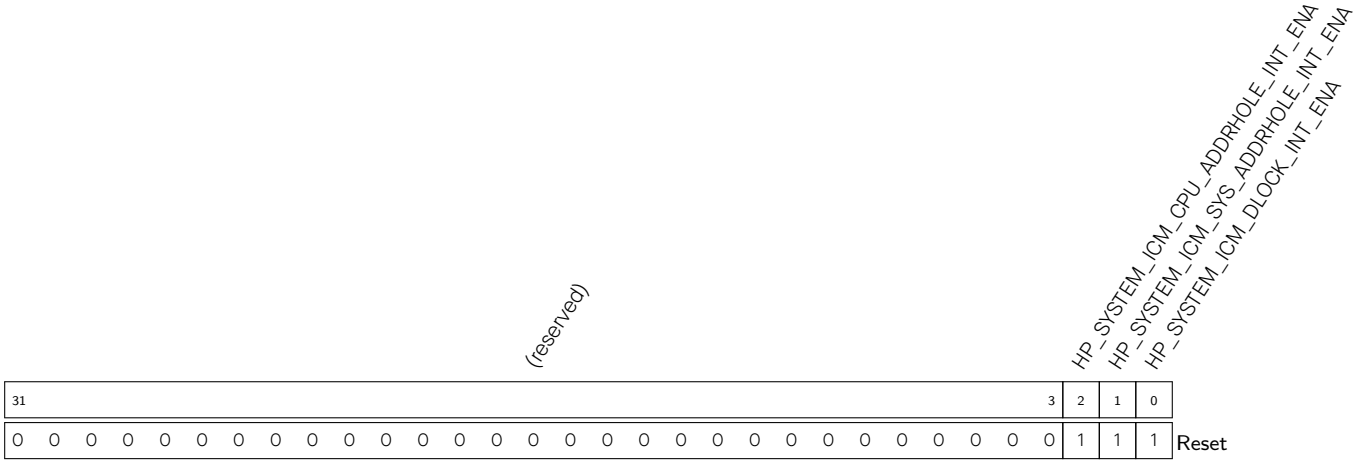


HP_SYSTEM_ICM_DLOCK_INT_ST DLOCK_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_ST ICM_SYS_ADDRHOLE_INT 的屏蔽中断状态。(RO)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_ST ICM_CPU_ADDRHOLE_INT 的屏蔽中断状态。(RO)

Register 21.95. HP_SYSTEM_ICM_INT_ENA_REG (0x0014)

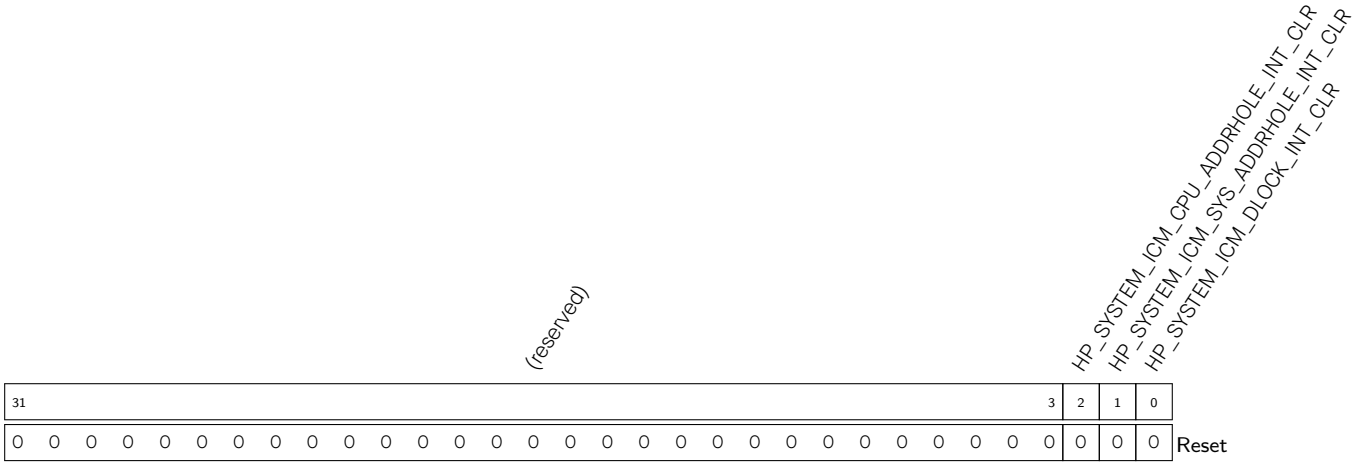


HP_SYSTEM_ICM_DLOCK_INT_ENA 写 1 使能 DLOCK_INT。(R/W)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_ENA 写 1 使能 ICM_SYS_ADDRHOLE_INT。(R/W)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_ENA 写 1 使能 ICM_CPU_ADDRHOLE_INT。(R/W)

Register 21.96. HP_SYSTEM_ICM_INT_CLR_REG (0x0018)

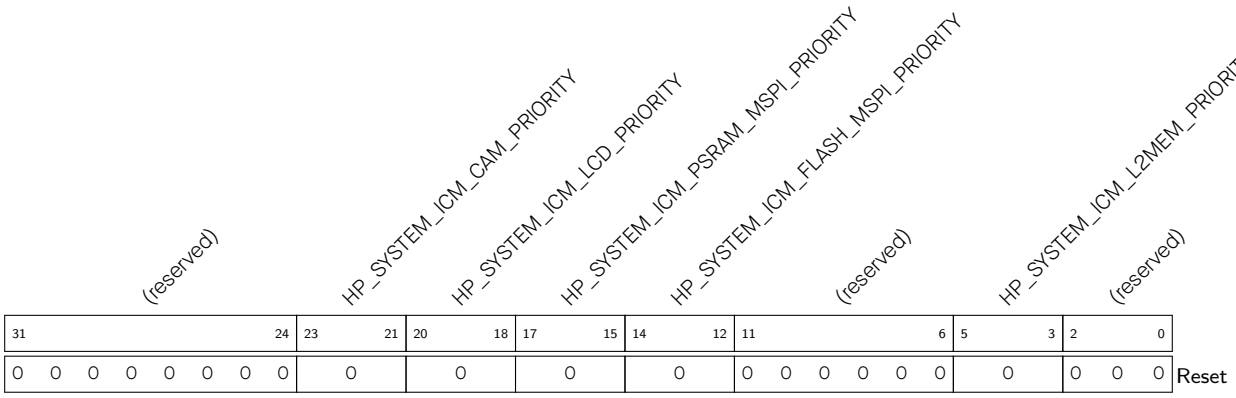


HP_SYSTEM_ICM_DLOCK_INT_CLR 写 1 清除 DLOCK_INT。(WT)

HP_SYSTEM_ICM_SYS_ADDRHOLE_INT_CLR 写 1 清除 ICM_SYS_ADDRHOLE_INT。(WT)

HP_SYSTEM_ICM_CPU_ADDRHOLE_INT_CLR 写 1 清除 ICM_CPU_ADDRHOLE_INT。(WT)

Register 21.97. HP_SYSTEM_ICM_SLV_ARB_PRIORITY_REG (0x0024)



HP_SYSTEM_ICM_L2MEM_PRIORITY 配置连接到 SYS ICM 的从机响应通道中 L2MEM 的仲裁优先级。(R/W)

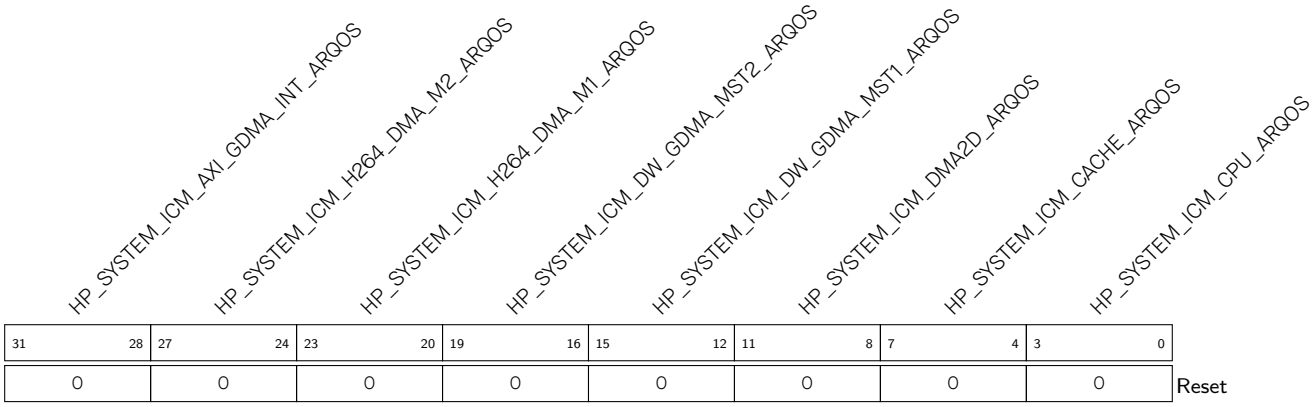
HP_SYSTEM_ICM_FLASH_MSPI_PRIORITY 配置连接到 SYS ICM 的从机响应通道中 FLASH MSPI 的仲裁优先级。(R/W)

HP_SYSTEM_ICM_PSRAM_MSPI_PRIORITY 配置连接到 SYS ICM 的从机响应通道中 PSRAM MSPI 的仲裁优先级。(R/W)

HP_SYSTEM_ICM_LCD_PRIORITY 配置连接到 SYS ICM 的从机响应通道中 MIPI_LCD 寄存器的仲裁优先级。(R/W)

HP_SYSTEM_ICM_CAM_PRIORITY 配置连接到 SYS ICM 的从机响应通道中 MIPI_CAM 寄存器的仲裁优先级。(R/W)

Register 21.98. HP_SYSTEM_ICM_MST_ARQOS_REG0_REG (0x0028)



HP_SYSTEM_ICM_CPU_ARQOS 配置 CPU ICM 的 ARQoS 值。(R/W)

HP_SYSTEM_ICM_CACHE_ARQOS 配置 CACHE 的 ARQoS 值。(R/W)

HP_SYSTEM_ICM_DMA2D_ARQOS 配置 DMA2D 的 ARQoS 值。(R/W)

HP_SYSTEM_ICM_DW_GDMA_MST1_ARQOS 配置 DW-GDMA MST1 的 ARQoS 值。(R/W)

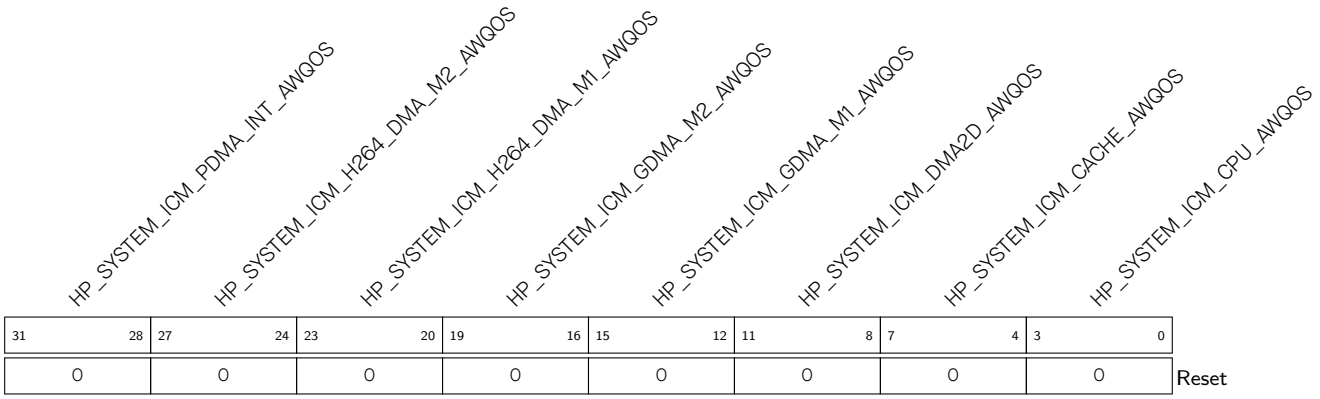
HP_SYSTEM_ICM_DW_GDMA_MST2_ARQOS 配置 DW-GDMA MST2 的 ARQoS 值。(R/W)

HP_SYSTEM_ICM_H264_DMA_M1_ARQOS 配置 H264 DMAMST1 的 ARQoS 值。(R/W)

HP_SYSTEM_ICM_H264_DMA_M2_ARQOS 配置 H264 DMAMST2 的 ARQoS 值。(R/W)

HP_SYSTEM_ICM_AXI_GDMA_INT_ARQOS 配置 AXI GDMA 的 ARQoS 值。(R/W)

Register 21.99. HP_SYSTEM_ICM_MST_AWQOS_REG0_REG (0x0030)



HP_SYSTEM_ICM_CPU_AWQOS 配置 CPU ICM 的 AWQoS 值。(R/W)

HP_SYSTEM_ICM_CACHE_AWQOS 配置 CACHE 的 AWQoS 值。(R/W)

HP_SYSTEM_ICM_DMA2D_AWQOS 配置 DMA2D 的 AWQoS 值。(R/W)

HP_SYSTEM_ICM_GDMA_M1_AWQOS 配置 DW-GDMA MST1 的 AWQoS 值。(R/W)

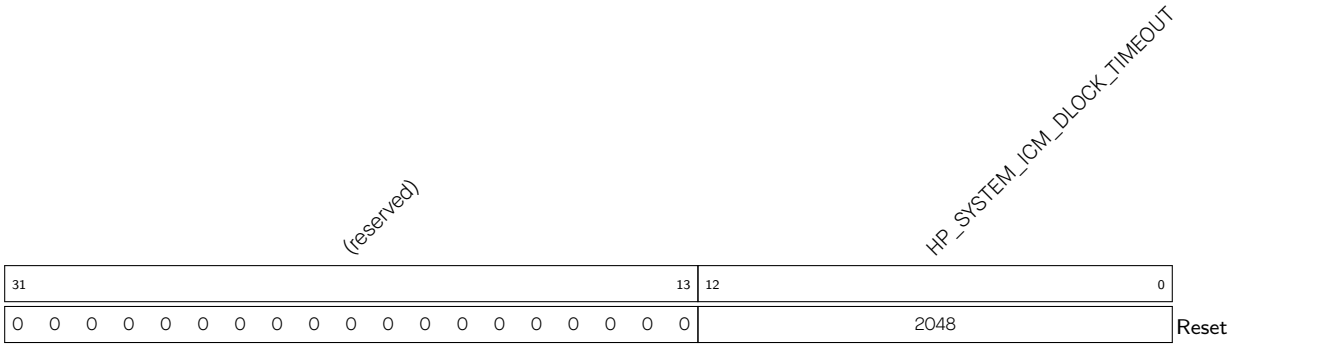
HP_SYSTEM_ICM_GDMA_M2_AWQOS 配置 DW-GDMA MST2 的 AWQoS 值。(R/W)

HP_SYSTEM_ICM_H264_DMA_M1_AWQOS 配置 H264 DMA MST1 的 AWQoS 值。(R/W)

HP_SYSTEM_ICM_H264_DMA_M2_AWQOS 配置 H264 DMA MST2 的 AWQoS 值。(R/W)

HP_SYSTEM_ICM_PDMA_INT_AWQOS 配置 AXI GDMA 的 AWQoS 值。(R/W)

Register 21.100. HP_SYSTEM_ICM_DLOCK_TIMEOUT_REG (0x0048)



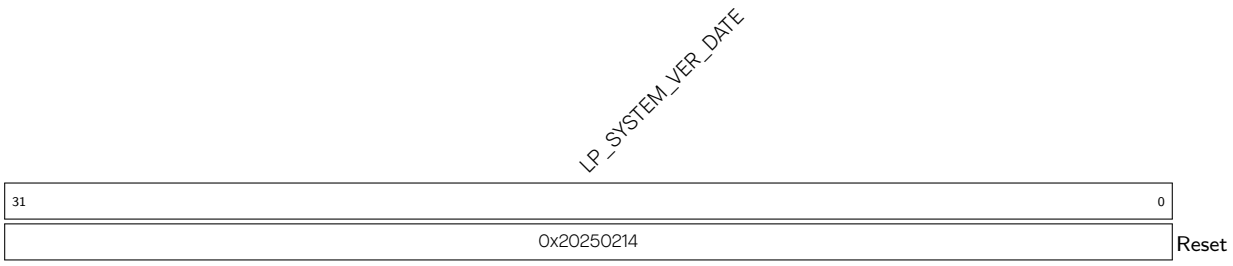
HP_SYSTEM_ICM_DLOCK_TIMEOUT 配置 deadlock 超时阈值。如果在配置的阈值内没有响应，将发生 deadlock。
(R/W)

21.5.3 LP 寄存器

本小节的所有地址均为相对于 LP 寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

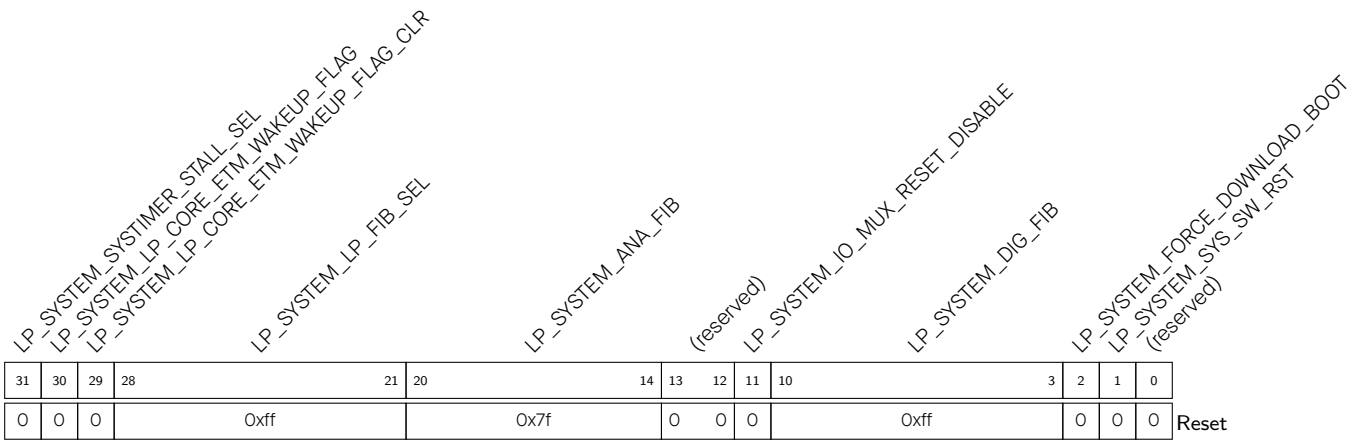
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 21.101. LP_SYSTEM_LP_SYS_VER_DATE_REG (0x0000)



LP_SYSTEM_VER_DATE 版本控制寄存器。(R/W)

Register 21.102. LP_SYSTEM_SYS_CTRL_REG (0x0008)



LP_SYSTEM_SYS_SW_RST 配置是否触发软件系统复位。(WT)

LP_SYSTEM_FORCE_DOWNLOAD_BOOT 配置是否在管脚启动模式为 SPI Boot 模式时，强制进入 Joint Download Boot 模式。

0：不强制进入 Joint Download Boot 模式

1：强制进入 Joint Download Boot 模式

(R/W)

LP_SYSTEM_DIG_FIB 配置软件 FIB。

位 [0]：未使用

位 [1]：软件 BOD 使能

位 [2]：软件 SWD 使能

位 [3]：未使用

位 [4]：软件 DCDC 开关

位 [5]：软件 DCDC VSET 使能

位 [6]：未使用

位 [7]：未使用

(R/W)

LP_SYSTEM_IO_MUX_RESET_DISABLE 配置是否禁用 HP IOMUX 复位。

0：启用 IOMUX 复位

1：禁用 IOMUX 复位

(R/W)

见下页...

Register 21.102. LP_SYSTEM_SYS_CTRL_REG (0x0008)

接上页...

LP_SYSTEM_ANA_FIB 表示模拟 FIB 状态。

位 [0]：未使用
位 [1]：模拟 BOD 使能
位 [2]：模拟 SWD 使能
位 [3]：未使用
位 [4]：未使用
位 [5]：未使用
位 [6]：未使用
位 [7]：未使用
(RO)

LP_SYSTEM_LP_FIB_SEL 配置 FIB 控制源选择。

位 [0]：未使用
位 [1]：配置 BOD 使能控制选择
0：由软件控制
1：由模拟控制
位 [2]：配置 SWD 使能控制选择
0：由软件控制
1：由模拟控制
位 [3]：未使用
位 [4]：未使用
位 [5]：未使用
位 [6]：未使用
位 [7]：未使用
(R/W)

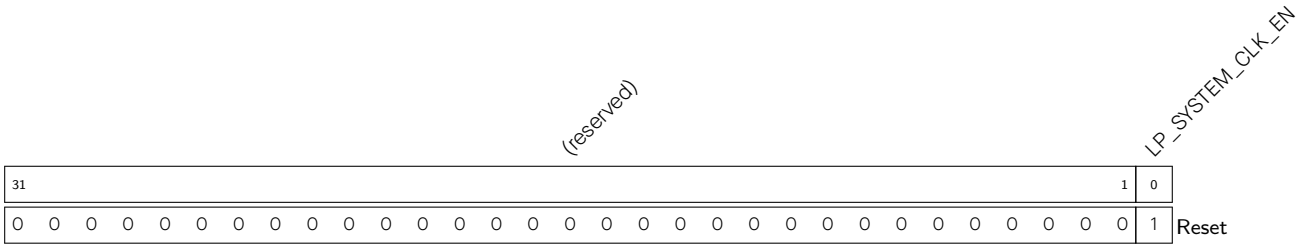
LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG_CLR 写 1 清除 LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG。
(WT)

LP_SYSTEM_LP_CORE_ETM_WAKEUP_FLAG 表示 ULP 唤醒的 ETM 任务。(R/WTC/SS)

LP_SYSTEM_SYSTIMER_STALL_SEL 配置 LP systimer_stall 信号是来自 HP CPU0 还是 HP CPU1。

0：来自 HP CPU0
1：来自 HP CPU1
(R/W)

Register 21.103. LP_SYSTEM_LP_CLK_CTRL_REG (0x000C)



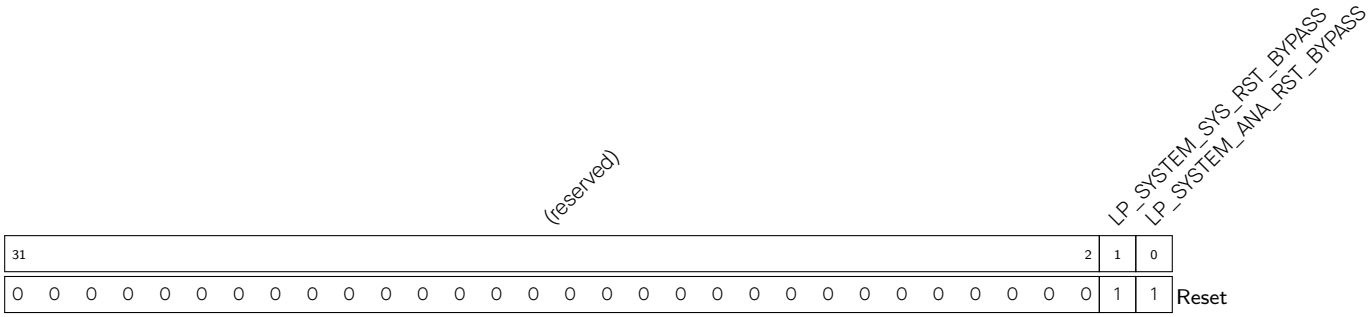
LP_SYSTEM_CLK_EN 配置是否始终开启寄存器时钟。

0：时钟仅在寄存器访问期间启用

1：时钟始终开启

(R/W)

Register 21.104. LP_SYSTEM_LP_RST_CTRL_REG (0x0010)



LP_SYSTEM_ANA_RST_BYPASS 配置是否绕过以下模拟复位源：欠压检测器、超级看门狗和电源故障检测器。

0：不绕过

1：绕过

(R/W)

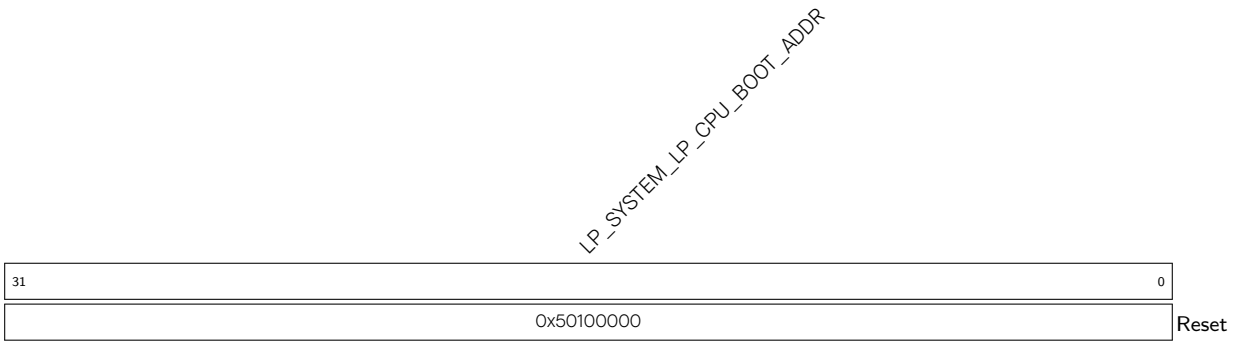
LP_SYSTEM_SYS_RST_BYPASS 配置是否绕过以下数字复位源：软件、HP 看门狗定时器、LP 看门狗定时器和 eFuse。

0：不绕过

1：绕过

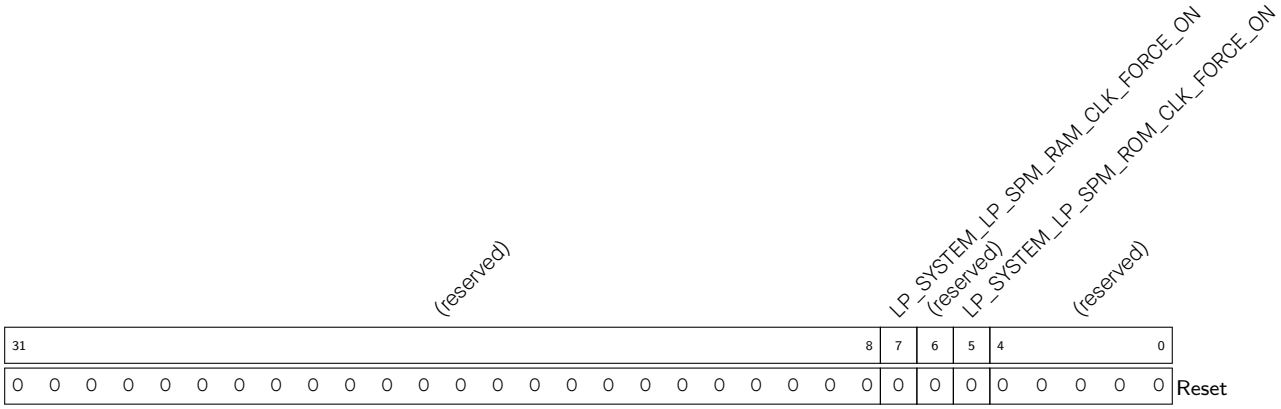
(R/W)

Register 21.105. LP_SYSTEM_LP_CORE_BOOT_ADDR_REG (0x0018)



LP_SYSTEM_LP_CPU_BOOT_ADDR 配置 LP CPU 启动地址。(R/W)

Register 21.106. LP_SYSTEM_LP_SPM_PWR_CTRL_REG (0x0024)



LP_SYSTEM_LP_SPM_ROM_CLK_FORCE_ON 配置当 LP CPU 时钟开启时是否强制开启 LP SPM_ROM 时钟。

0：不强制开启

1：强制开启

(R/W)

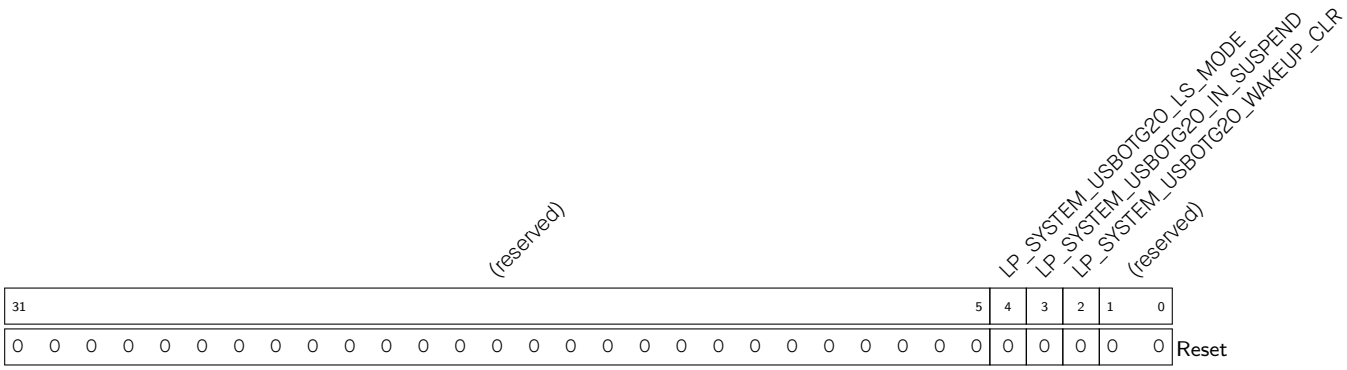
LP_SYSTEM_LP_SPM_RAM_CLK_FORCE_ON 配置是否强制开启 LP SPM_RAM 时钟。

0：不强制开启

1：强制开启

(R/W)

Register 21.107. LP_SYSTEM_USB_CTRL_REG (0x0100)



LP_SYSTEM_USBOTG20_WAKEUP_CLR 写 1 清除来自 USB OTG2.0 到 PMU 的唤醒信号。(WT)

LP_SYSTEM_USBOTG20_IN_SUSPEND 配置是否启用来自 USB OTG2.0 的唤醒信号。

0：禁用

1：启用

(R/W)

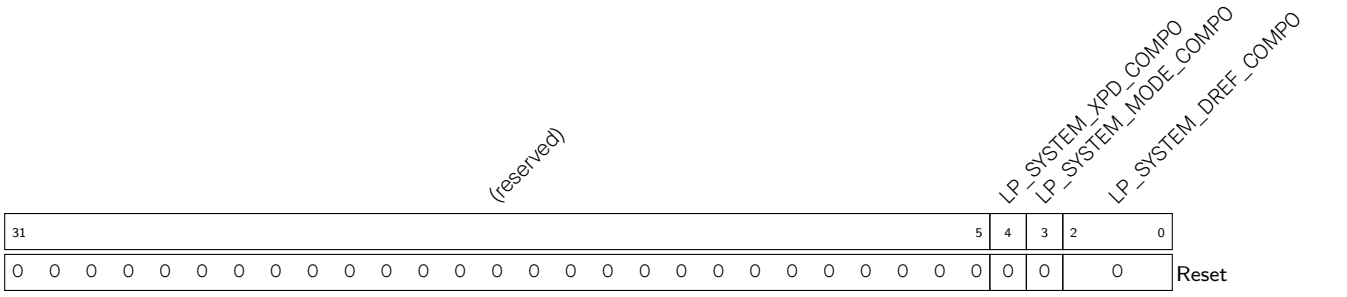
LP_SYSTEM_USBOTG20_LS_MODE 配置 USB OTG2.0 是否工作在低速模式。

0：非低速模式

1：低速模式

(R/W)

Register 21.108. LP_SYSTEM_PAD_COMP0_REG (0x0148)



LP_SYSTEM_DREF_COMP0 配置模拟电压比较器 0 的内部参考电压。(R/W)

LP_SYSTEM_MODE_COMP0 配置模拟电压比较器 0 的比较模式。

0：比较主电压与外部参考电压

1：比较主电压与内部参考电压

(R/W)

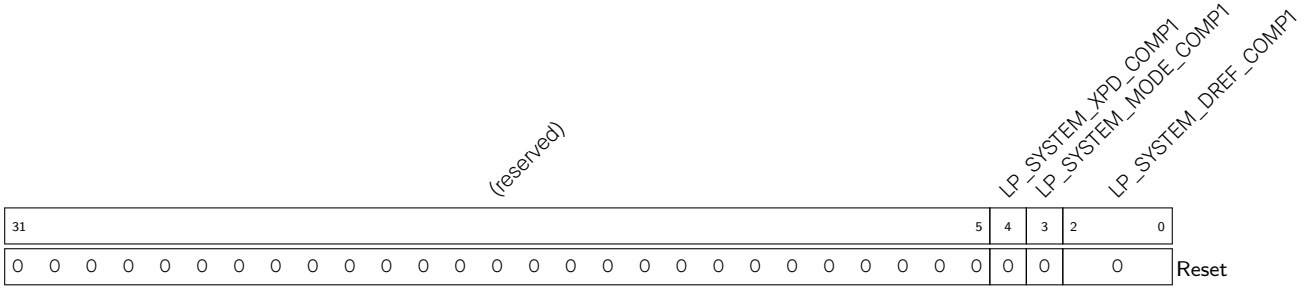
LP_SYSTEM_XPD_COMP0 配置是否启用模拟电压比较器 0。

0：禁用

1：启用

(R/W)

Register 21.109. LP_SYSTEM_PAD_COMP1_REG (0x014C)



LP_SYSTEM_DREF_COMP1 配置模拟电压比较器 1 的内部参考电压。(R/W)

LP_SYSTEM_MODE_COMP1 配置模拟电压比较器 1 的比较模式。

0：比较主电压与外部参考电压

1：比较主电压与内部参考电压

(R/W)

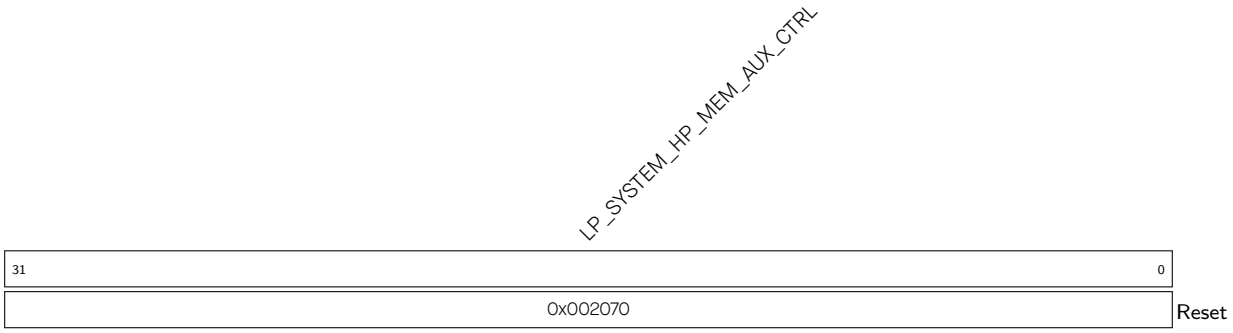
LP_SYSTEM_XPD_COMP1 配置是否启用模拟电压比较器 1。

0：禁用

1：启用

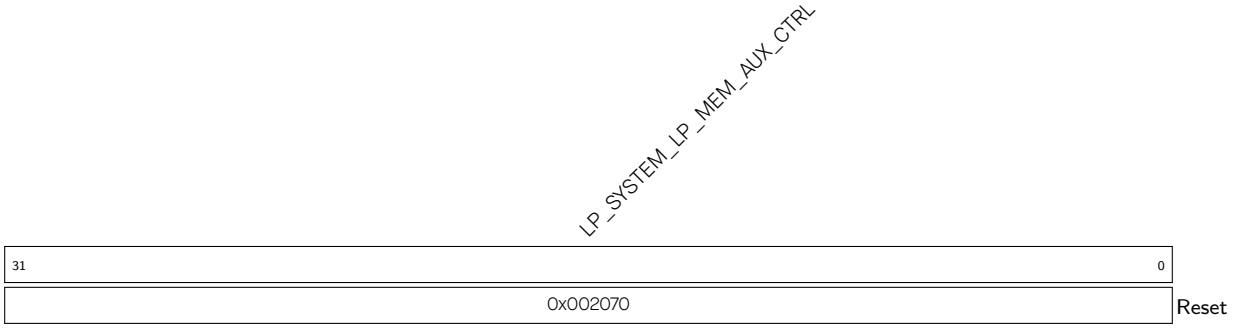
(R/W)

Register 21.110. LP_SYSTEM_HP_MEM_AUX_CTRL_REG (0x0180)



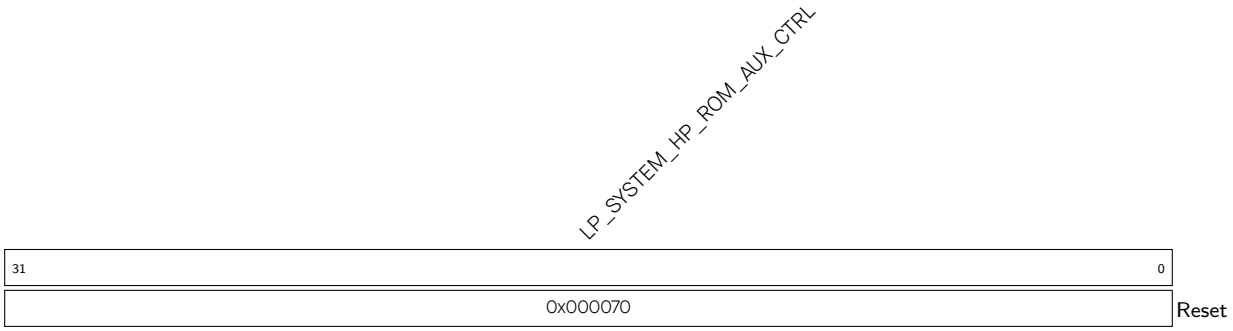
LP_SYSTEM_HP_MEM_AUX_CTRL 配置 HP 系统内存辅助控制值。(R/W)

Register 21.111. LP_SYSTEM_LP_MEM_AUX_CTRL_REG (0x0184)



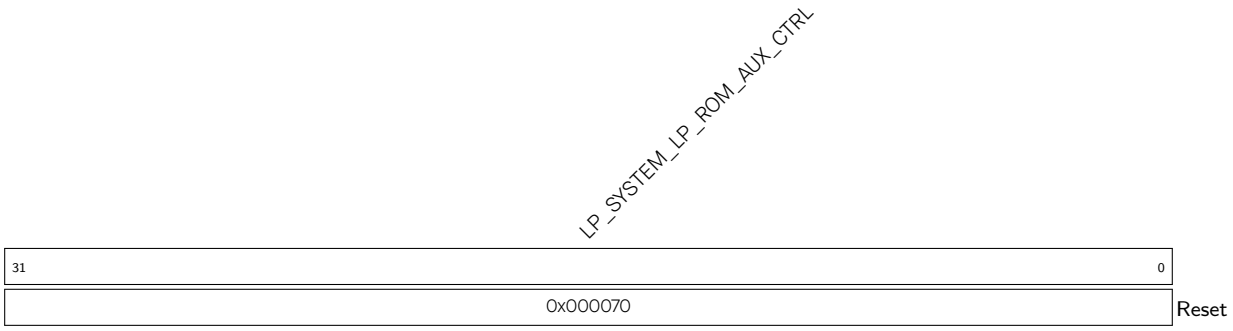
LP_SYSTEM_LP_MEM_AUX_CTRL 配置 LP 系统内存辅助控制值。(R/W)

Register 21.112. LP_SYSTEM_HP_ROM_AUX_CTRL_REG (0x0188)



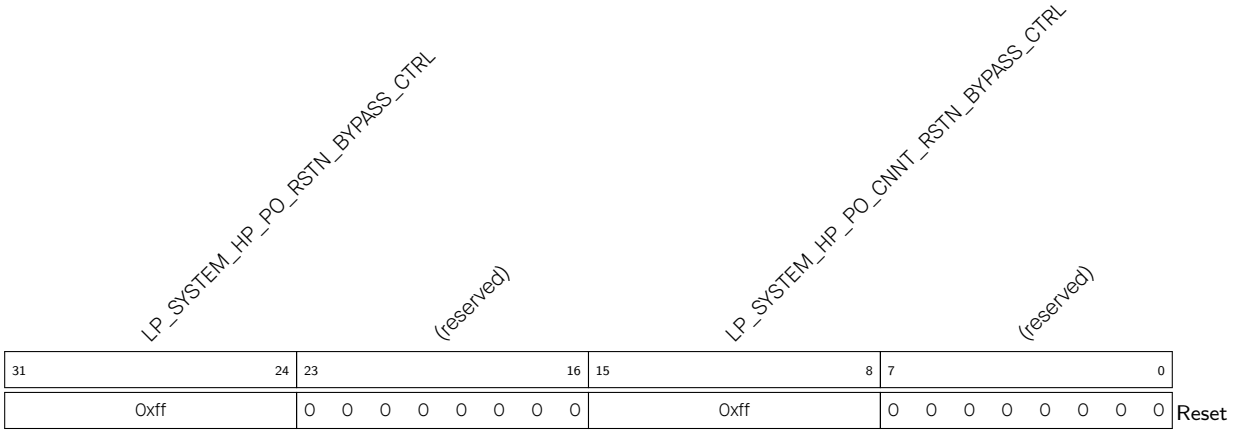
LP_SYSTEM_HP_ROM_AUX_CTRL 配置 L2ROM 辅助控制值。(R/W)

Register 21.113. LP_SYSTEM_LP_ROM_AUX_CTRL_REG (0x018C)



LP_SYSTEM_LP_ROM_AUX_CTRL 配置 LPROM 辅助控制值。(R/W)

Register 21.114. LP_SYSTEM_HP_POR_RST_BYPASS_CTRL_REG (0x01A0)



LP_SYSTEM_HP_PO_CNNT_RSTN_BYPASS_CTRL 配置在上电复位期间是否绕过 HP_CNNT 电源域的复位源。

位 [0]：配置是否绕过 eFuse 复位

0：无作用

1：绕过

位 [1]：配置是否绕过 LP 看门狗定时器复位

0：无作用

1：绕过

位 [2]：配置是否绕过 USB (JTAG) 复位

0：无作用

1：绕过

位 [3]：配置是否绕过 USB (UART) 复位

0：无作用

1：绕过

位 [4]：未使用

位 [5]：未使用

位 [6]：配置是否绕过 HP 看门狗定时器复位

0：无作用

1：绕过

位 [7]：配置是否绕过软件复位

0：无作用

1：绕过

(R/W)

见下页...

Register 21.114. LP_SYSTEM_HP_POR_RST_BYPASS_CTRL_REG (0x01A0)

接上页...

LP_SYSTEM_HP_PO_RSTN_BYPASS_CTRL 配置在上电复位期间是否绕过 HP 电源域的复位源。

位 [0]：配置是否绕过 eFuse 复位

0：无作用

1：绕过

位 [1]：配置是否绕过 LP 看门狗定时器复位

0：无作用

1：绕过

位 [2]：配置是否绕过 USB (JTAG) 复位

0：无作用

1：绕过

位 [3]：配置是否绕过 USB (UART) 复位

0：无作用

1：绕过

位 [4]：未使用

位 [5]：未使用

位 [6]：配置是否绕过 HP 看门狗定时器复位

0：无作用

1：绕过

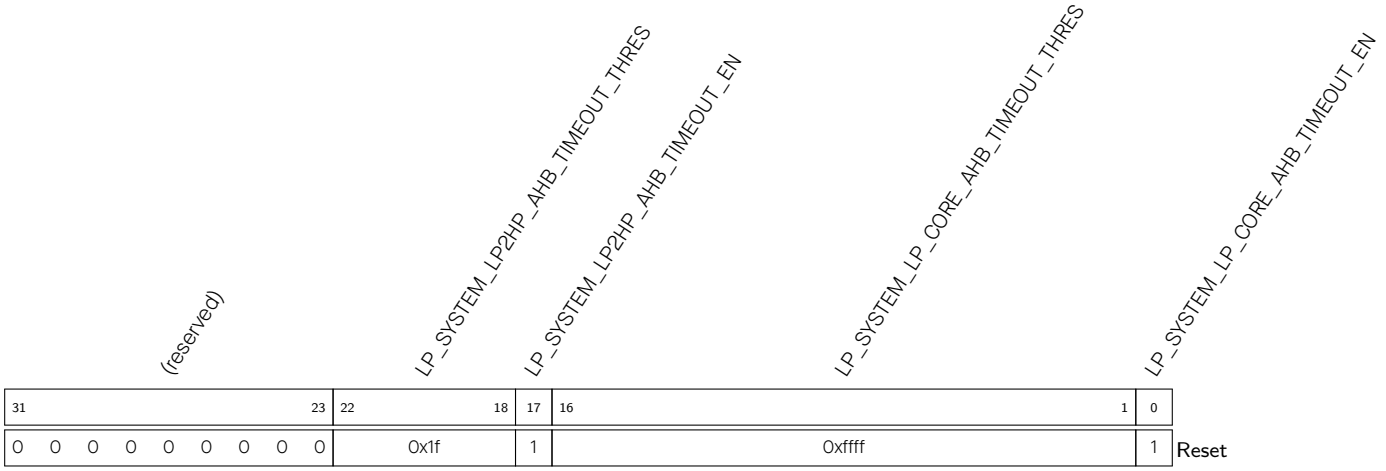
位 [7]：配置是否绕过软件复位

0：无作用

1：绕过

(R/W)

Register 21.115. LP_SYSTEM_LP_CORE_AHB_TIMEOUT_REG (0x01B0)



LP_SYSTEM_LP_CORE_AHB_TIMEOUT_EN 配置是否启用 LP AHB 总线的超时保护。

0：禁用

1：启用

(R/W)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_THRES 配置 LP_CORE_AHB_TIMEOUT_INT 阈值。(R/W)

LP_SYSTEM_LP2HP_AHB_TIMEOUT_EN 配置是否启用 LP2HP AHB 总线的超时保护。

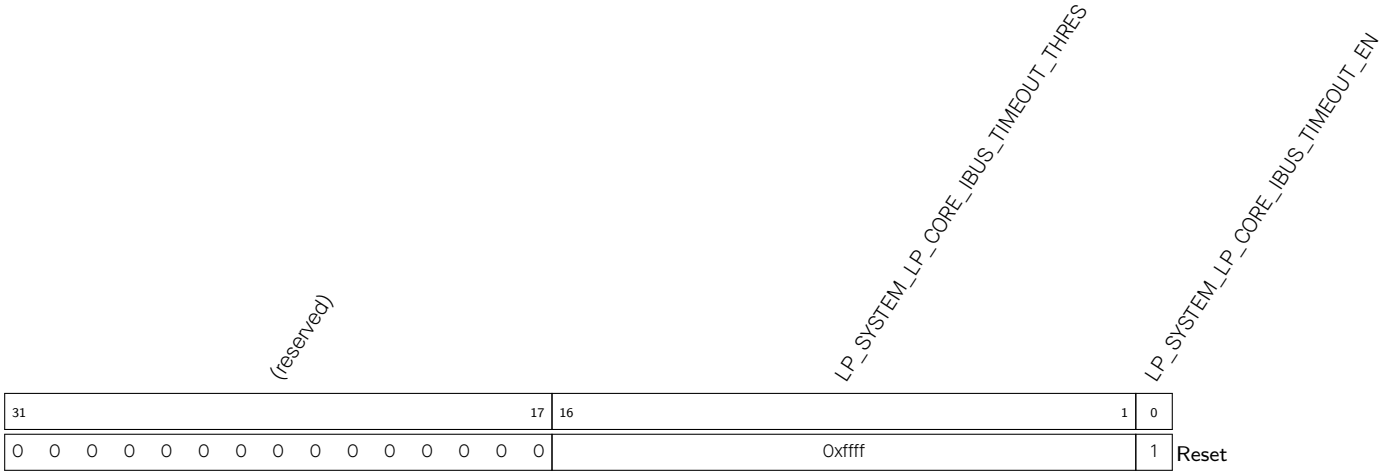
0：禁用

1：启用

(R/W)

LP_SYSTEM_LP2HP_AHB_TIMEOUT_THRES 配置 LP_CORE_AHB_TIMEOUT_INT 阈值。(R/W)

Register 21.116. LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_REG (0x01B4)



LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_EN 配置是否启用 LP CPU IBUS 的超时保护。

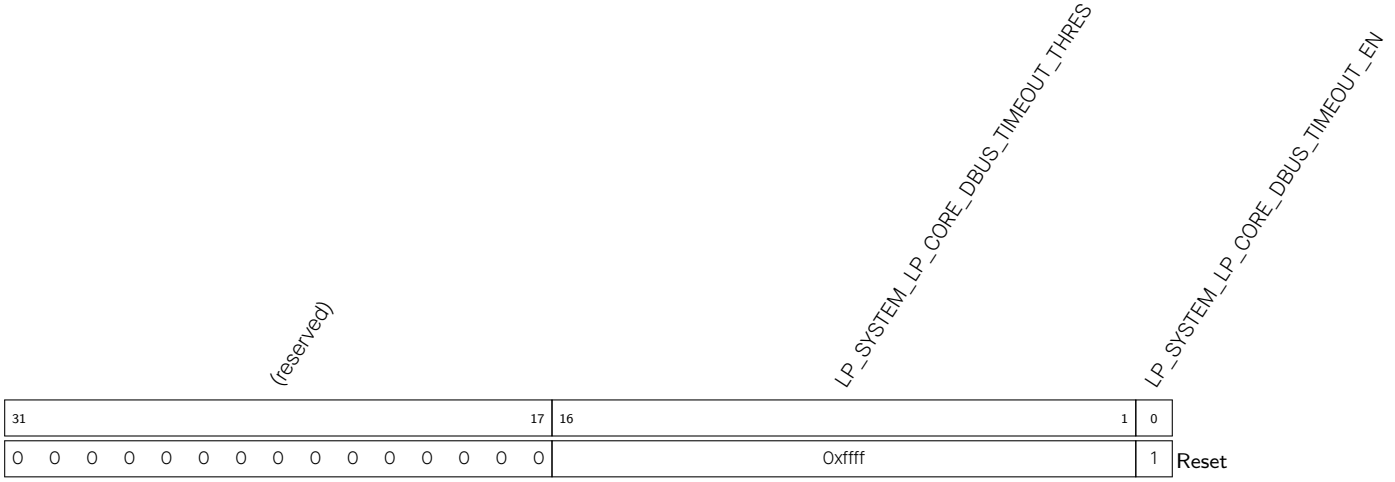
0：禁用

1：启用

(R/W)

LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_THRES 配置 LP_CORE_IBUS_TIMEOUT_INT 阈值。(R/W)

Register 21.117. LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_REG (0x01B8)



LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_EN 配置是否启用 LP CPU DBUS 的超时保护。

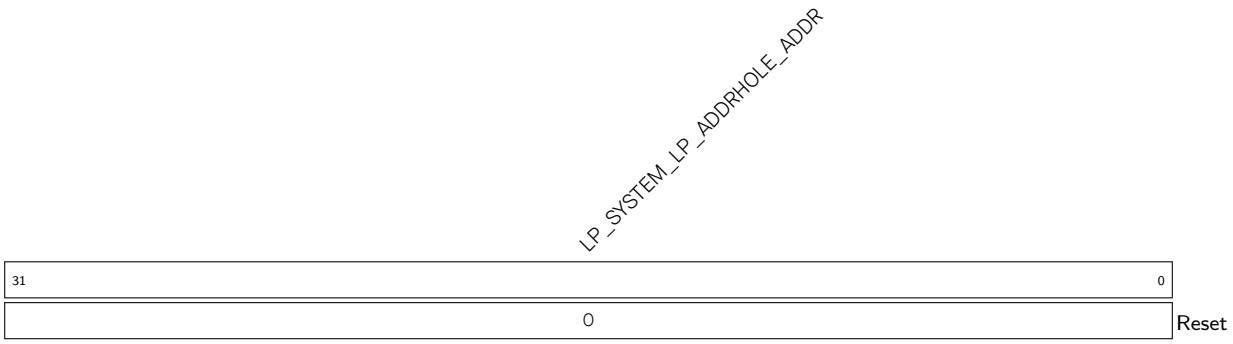
0：禁用

1：启用

(R/W)

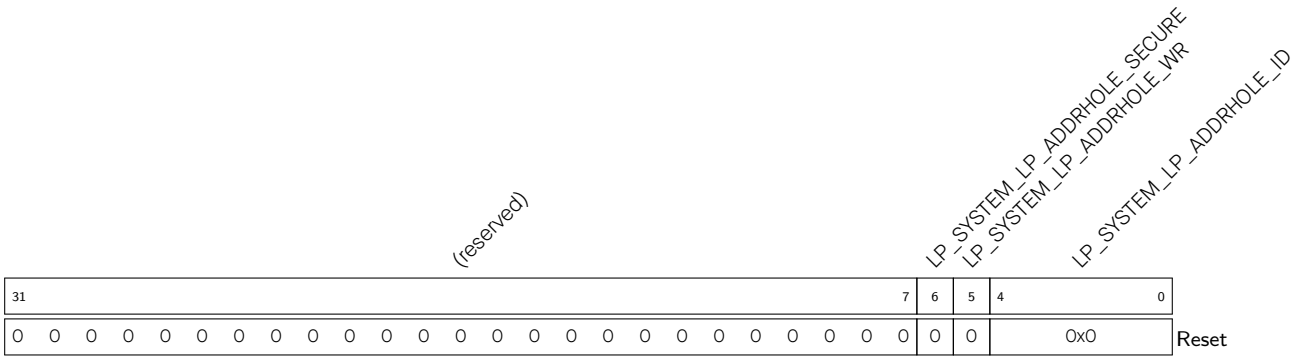
LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_THRES 配置 LP_CORE_DBUS_TIMEOUT_INT 阈值。(R/W)

Register 21.118. LP_SYSTEM_LP_ADDRHOLE_ADDR_REG (0x0168)



LP_SYSTEM_LP_ADDRHOLE_ADDR 表示 LP_ADDRHOLE_INT 发生时的传输地址。(RO)

Register 21.119. LP_SYSTEM_LP_ADDRHOLE_INFO_REG (0x016C)



LP_SYSTEM_LP_ADDRHOLE_ID 表示 LP_ADDRHOLE_INT 发生时的主 ID。

0：LP CPU0
1：LP CPU1
2：LP CPU
3：USB OTG11
4：保留
5：GMAC
6：SDMMC
7：USBOTG20
8：TRACE0
9：TRACE1
10：SPM 监视器
11：L2MEM 监视器
16~31：AHB PDMA
(RO)

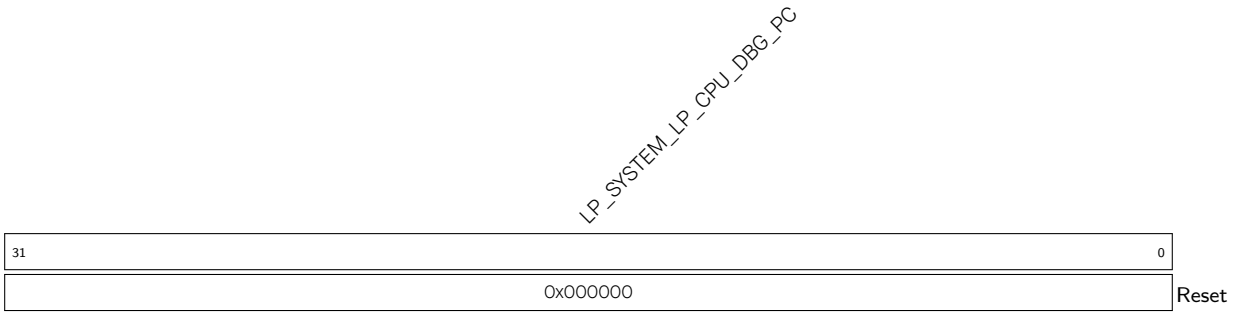
LP_SYSTEM_LP_ADDRHOLE_WR 表示 LP_ADDRHOLE_INT 发生时的传输方向。

1：写传输
0：读传输
(RO)

LP_SYSTEM_LP_ADDRHOLE_SECURE 表示 LP_ADDRHOLE_INT 发生时的地址孔类型。

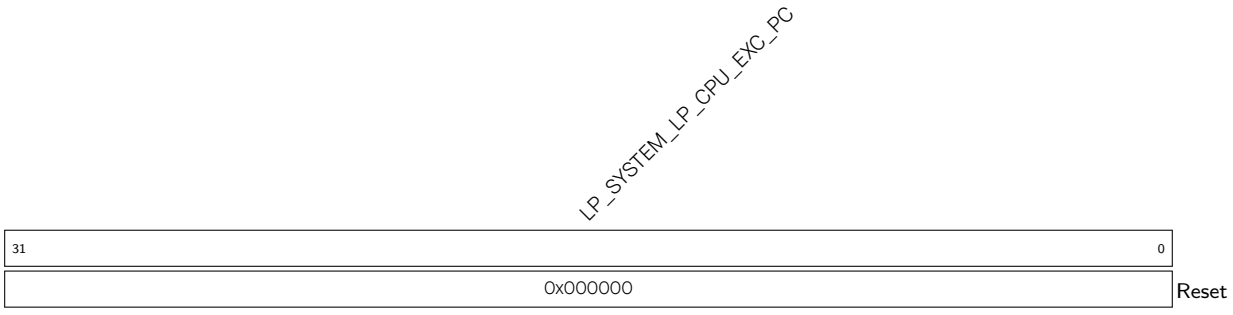
1：非法地址访问
0：无权限访问
(RO)

Register 21.120. LP_SYSTEM_LP_CPU_DBG_PC_REG (0x0190)



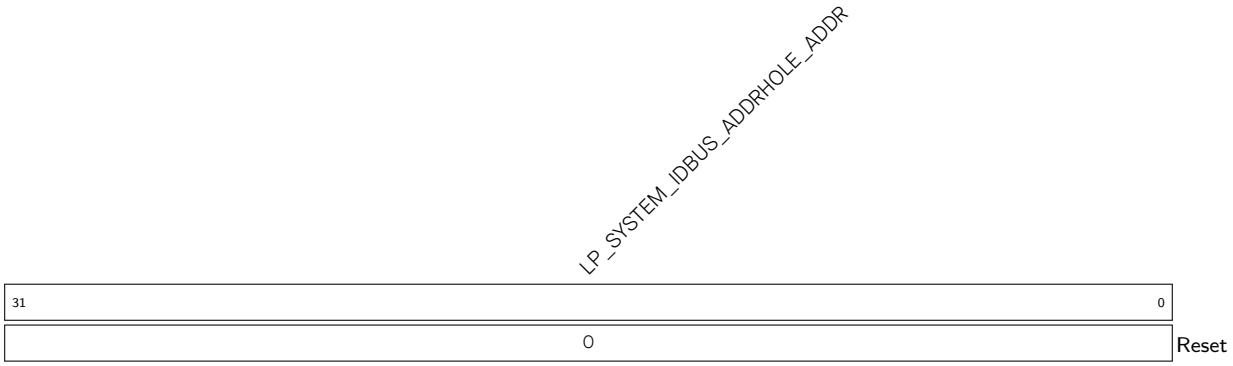
LP_SYSTEM_LP_CPU_DBG_PC 表示 LP CPU 当前 PC 地址。(RO)

Register 21.121. LP_SYSTEM_LP_CPU_EXC_PC_REG (0x0194)



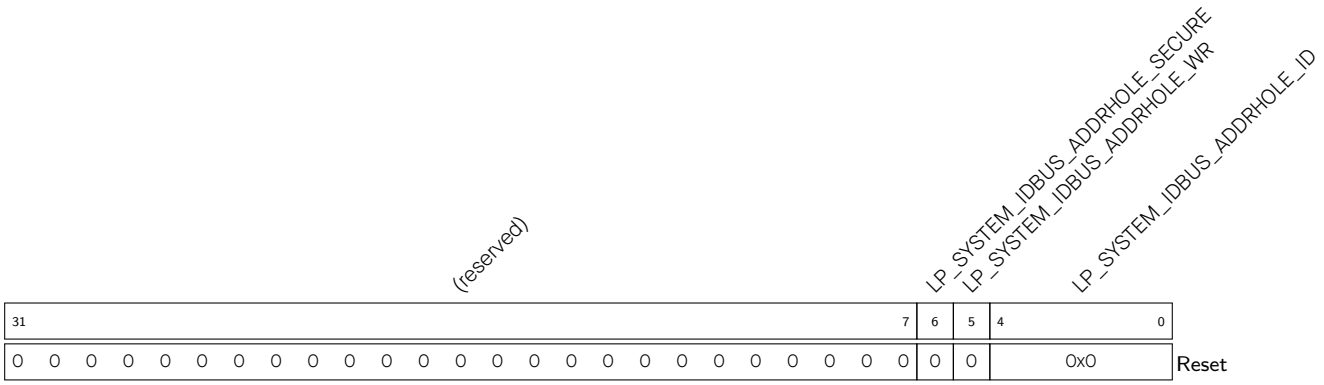
LP_SYSTEM_LP_CPU_EXC_PC 表示 LP CPU 异常 PC 地址。(RO)

Register 21.122. LP_SYSTEM_IDBUS_ADDRHOLE_ADDR_REG (0x0198)



LP_SYSTEM_IDBUS_ADDRHOLE_ADDR 表示 IDBUS_ADDRHOLE_INT 发生时的传输地址。(RO)

Register 21.123. LP_SYSTEM_IDBUS_ADDRHOLE_INFO_REG (0x019C)



LP_SYSTEM_IDBUS_ADDRHOLE_ID 表示 IDBUS_ADDRHOLE_INT 发生时的 LP CPU ID 值 (00010)。(RO)

LP_SYSTEM_IDBUS_ADDRHOLE_WR 表示 IDBUS_ADDRHOLE_INT 发生时的传输方向。

0：读传输

1：写传输

(RO)

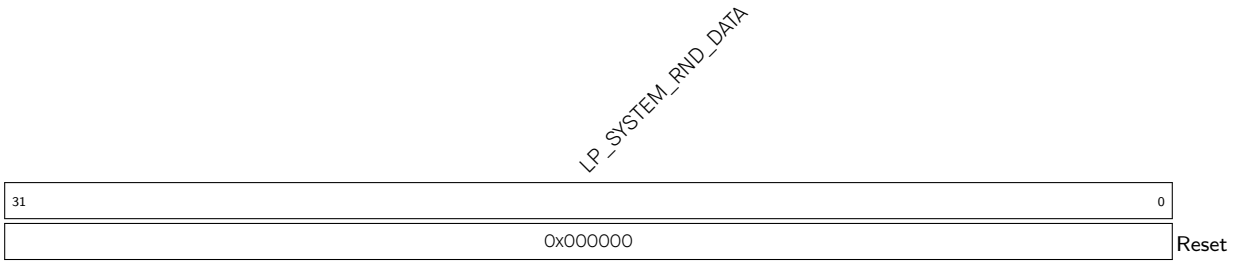
LP_SYSTEM_IDBUS_ADDRHOLE_SECURE 表示 IDBUS_ADDRHOLE_INT 发生时的地址孔类型。

0：无权限访问

1：非法地址访问

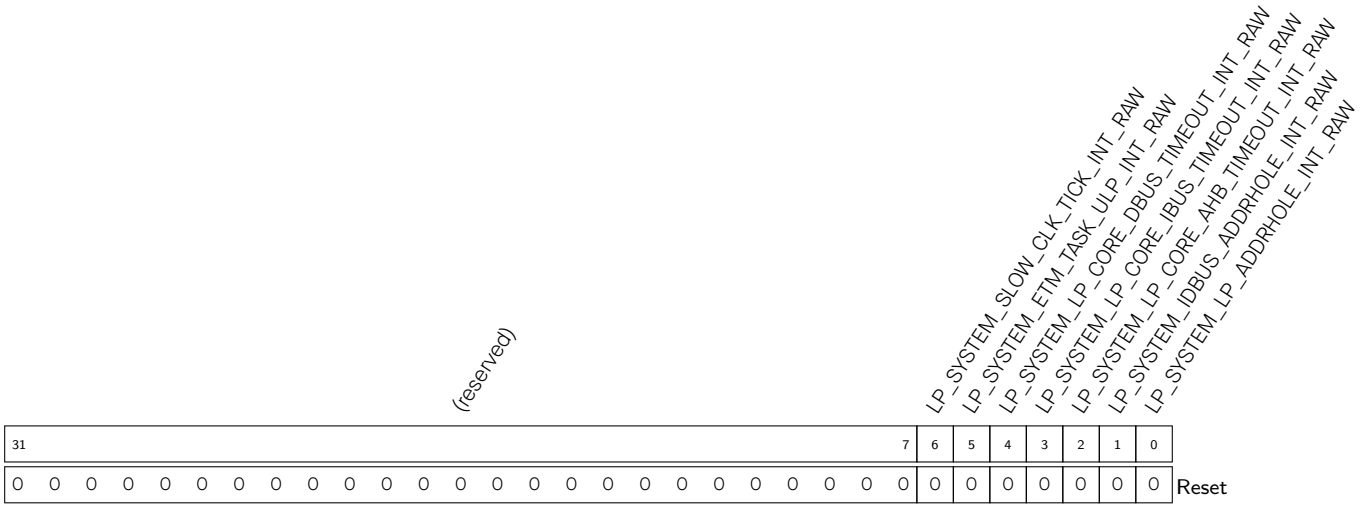
(RO)

Register 21.124. LP_SYSTEM_RNG_DATA_REG (0x01A4)



LP_SYSTEM_RND_DATA 表示 RNG 生成的随机数。(RO)

Register 21.125. LP_SYSTEM_INT_RAW_REG (0x0170)



LP_SYSTEM_LP_ADDRHOLE_INT_RAW LP_ADDRHOLE_INT 的原始中断状态。(R/SS/WTC)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_RAW IDBUS_ADDRHOLE_INT 的原始中断状态。(R/SS/WTC)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_RAW LP_CORE_AHB_TIMEOUT_INT 的原始中断状态。(R/SS/WTC)

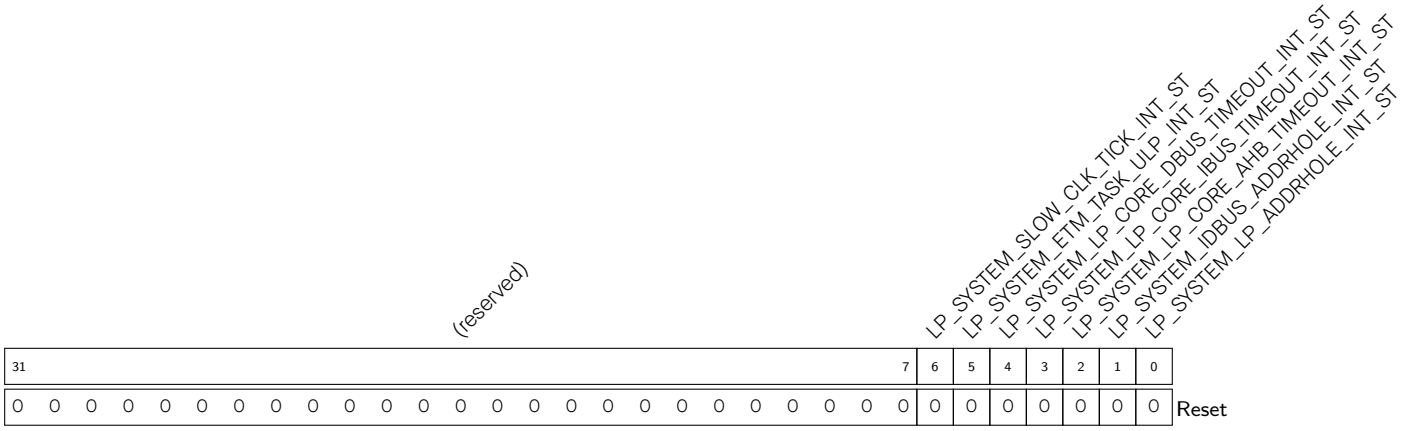
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_RAW LP_CORE_IBUS_TIMEOUT_INT 的原始中断状态。(R/SS/WTC)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_RAW LP_CORE_DBUS_TIMEOUT_INT 的原始中断状态。(R/SS/WTC)

LP_SYSTEM_ETM_TASK_ULP_INT_RAW ETM_TASK_ULP_INT 的原始中断状态。(R/SS/WTC)

LP_SYSTEM_SLOW_CLK_TICK_INT_RAW SLOW_CLK_TICK_INT 的原始中断状态。(R/SS/WTC)

Register 21.126. LP_SYSTEM_INT_ST_REG (0x0174)



LP_SYSTEM_LP_ADDRHOLE_INT_ST LP_ADDRHOLE_INT 的屏蔽中断状态。(RO)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_ST IDBUS_ADDRHOLE_INT 的屏蔽中断状态。(RO)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_ST LP_CORE_AHB_TIMEOUT_INT 的屏蔽中断状态。(RO)

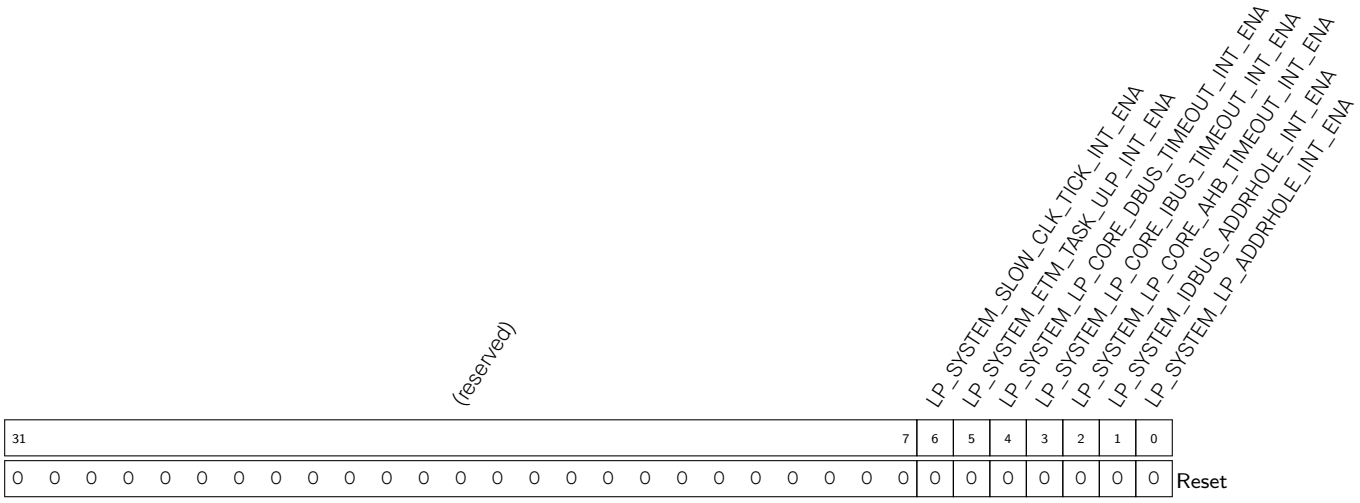
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_ST LP_CORE_IBUS_TIMEOUT_INT 的屏蔽中断状态。(RO)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_ST LP_CORE_DBUS_TIMEOUT_INT 的屏蔽中断状态。(RO)

LP_SYSTEM_ETM_TASK_ULP_INT_ST ETM_TASK_ULP_INT 的屏蔽中断状态。(RO)

LP_SYSTEM_SLOW_CLK_TICK_INT_ST SLOW_CLK_TICK_INT 的屏蔽中断状态。(RO)

Register 21.127. LP_SYSTEM_INT_ENA_REG (0x0178)



LP_SYSTEM_LP_ADDRHOLE_INT_ENA 写 1 使能 LP_ADDRHOLE_INT。(R/W)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_ENA 写 1 使能 IDBUS_ADDRHOLE_INT。(R/W)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_ENA 写 1 使能 LP_CORE_AHB_TIMEOUT_INT。(R/W)

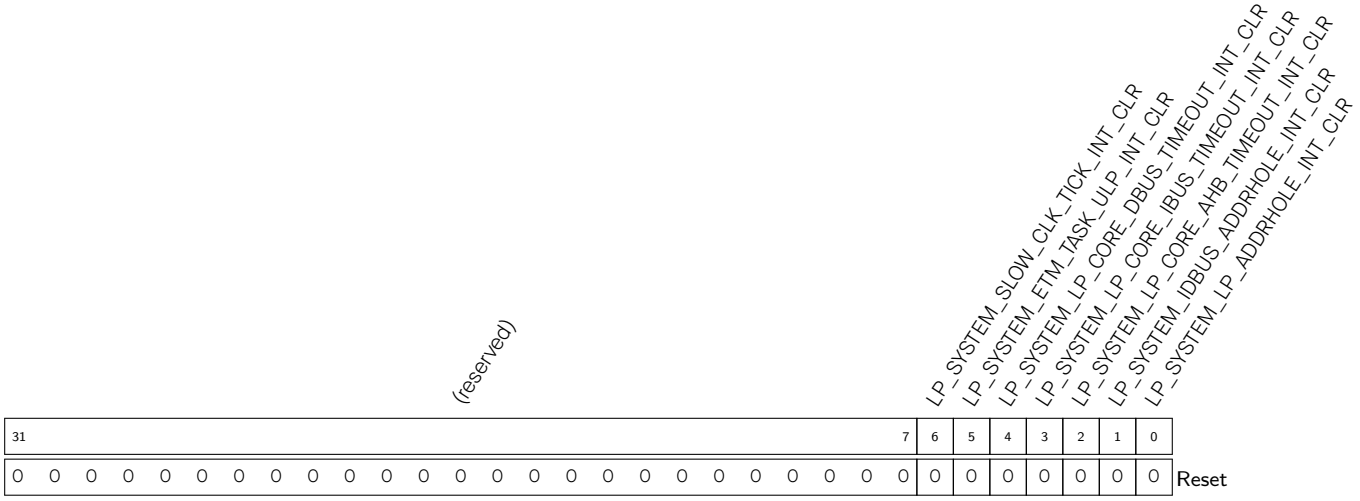
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_ENA 写 1 使能 LP_CORE_IBUS_TIMEOUT_INT。(R/W)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_ENA 写 1 使能 LP_CORE_DBUS_TIMEOUT_INT。(R/W)

LP_SYSTEM_ETM_TASK_ULP_INT_ENA 写 1 使能 ETM_TASK_ULP_INT。(R/W)

LP_SYSTEM_SLOW_CLK_TICK_INT_ENA 写 1 使能 SLOW_CLK_TICK_INT。(R/W)

Register 21.128. LP_SYSTEM_INT_CLR_REG (0x017C)



LP_SYSTEM_LP_ADDRHOLE_INT_CLR 写 1 清除 LP_ADDRHOLE_INT。(WT)

LP_SYSTEM_IDBUS_ADDRHOLE_INT_CLR 写 1 清除 IDBUS_ADDRHOLE_INT。(WT)

LP_SYSTEM_LP_CORE_AHB_TIMEOUT_INT_CLR 写 1 清除 LP_CORE_AHB_TIMEOUT_INT。(WT)

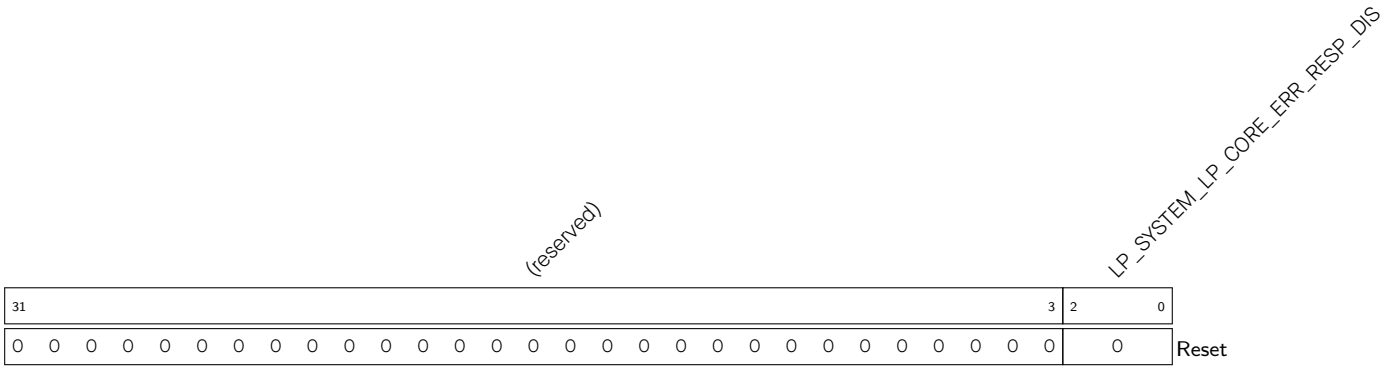
LP_SYSTEM_LP_CORE_IBUS_TIMEOUT_INT_CLR 写 1 清除 LP_CORE_IBUS_TIMEOUT_INT。(WT)

LP_SYSTEM_LP_CORE_DBUS_TIMEOUT_INT_CLR 写 1 清除 LP_CORE_DBUS_TIMEOUT_INT。(WT)

LP_SYSTEM_ETM_TASK_ULP_INT_CLR 写 1 清除 ETM_TASK_ULP_INT。(WT)

LP_SYSTEM_SLOW_CLK_TICK_INT_CLR 写 1 清除 SLOW_CLK_TICK_INT。(WT)

Register 21.129. LP_SYSTEM_LP_CORE_ERR_RESP_DIS_REG (0x01BC)



LP_SYSTEM_LP_CORE_ERR_RESP_DIS 配置是否禁用 LP CPU 总线的错误响应。

位 [0]：配置是否禁用 IBUS 的错误响应

0：启用错误响应

1：禁用错误响应

位 [1]：配置是否禁用 DBUS 的错误响应

0：启用错误响应

1：禁用错误响应

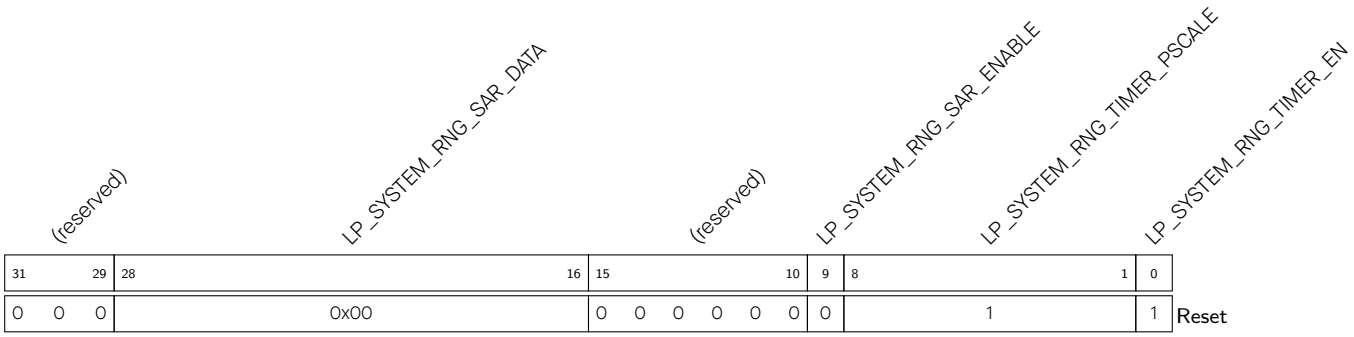
位 [2]：配置是否禁用 AHB 总线的错误响应

0：启用错误响应

1：禁用错误响应

(R/W)

Register 21.130. LP_SYSTEM_RNG_CFG_REG (0x01C0)



LP_SYSTEM_RNG_TIMER_EN 配置是否启用 RNG 定时器。

0：禁用

1：启用

(R/W)

LP_SYSTEM_RNG_TIMER_PSCALE 配置 RNG 定时器的时钟分频因子。(R/W)

LP_SYSTEM_RNG_SAR_ENABLE 配置是否启用 RNG_SARADC。

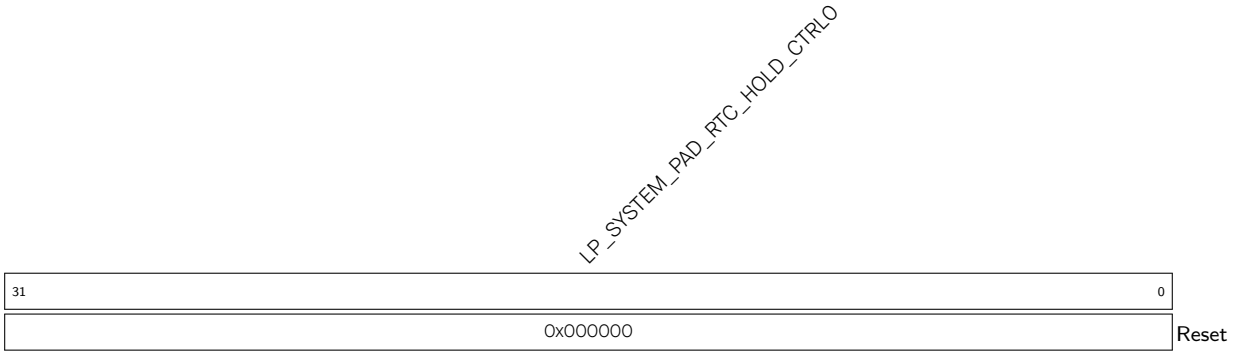
0：禁用

1：启用

(R/W)

LP_SYSTEM_RNG_SAR_DATA 表示模拟 SAR3_CAP 值，用于调试 RNG_SARADC 样本计数。(RO)

Register 21.131. LP_SYSTEM_PAD_RTC_HOLD_CTRL0_REG (0x01C4)



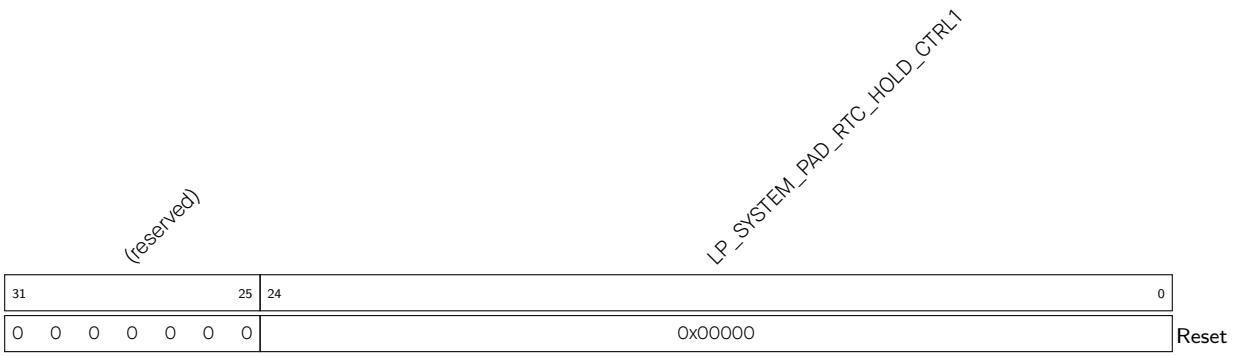
LP_SYSTEM_PAD_RTC_HOLD_CTRL0 配置是否保持 pad 0 ~ pad 31 的状态。

0：不保持

1：保持

(R/W)

Register 21.132. LP_SYSTEM_PAD_RTC_HOLD_CTRL1_REG (0x01C8)



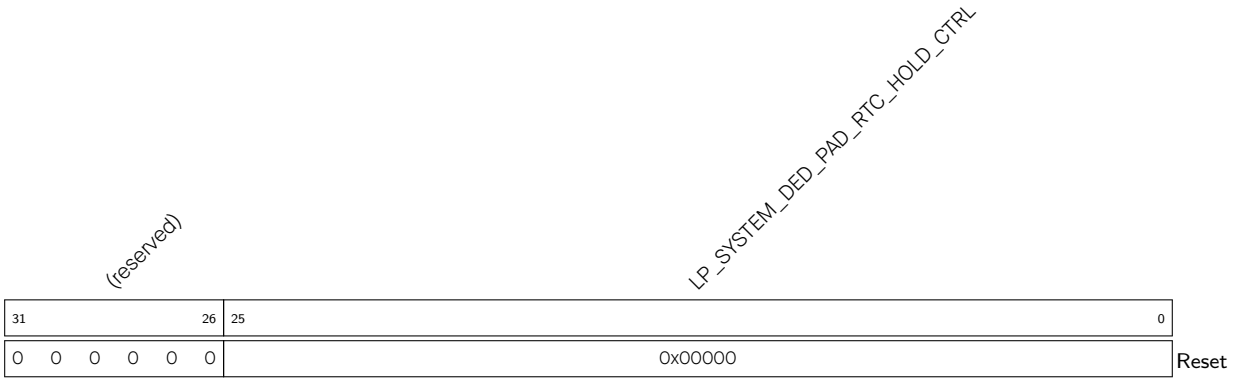
LP_SYSTEM_PAD_RTC_HOLD_CTRL1 配置是否保持 pad 32 ~ pad 56 的状态。

0：不保持

1：保持

(R/W)

Register 21.133. LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL_REG (0x01CC)



LP_SYSTEM_DED_PAD_RTC_HOLD_CTRL 配置是否保持 pad 状态。

位 [0:5]：Flash pad

0：不保持

1：保持

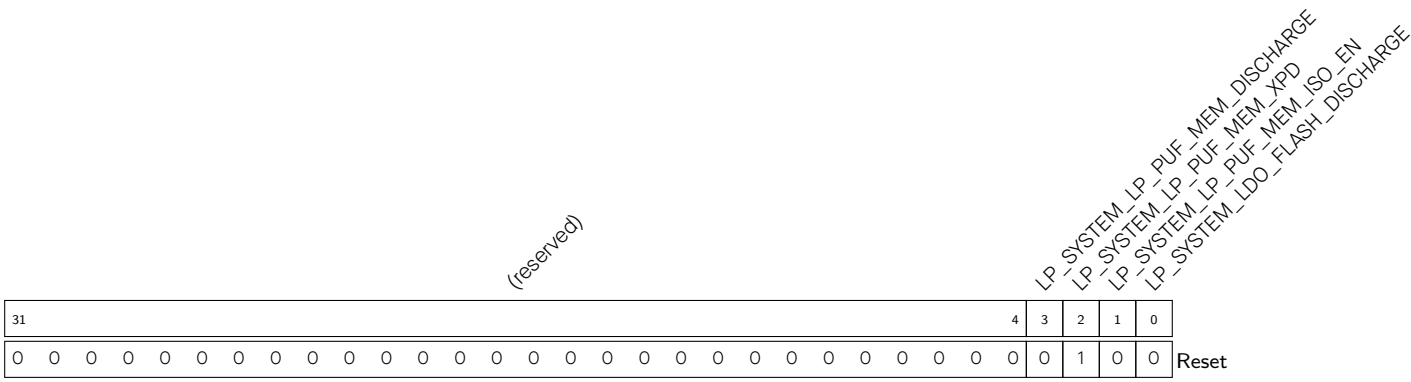
位 [6:25]：PSRAM pad

0：不保持

1：保持

(R/W)

Register 21.134. LP_SYSTEM_DISCHARGE_REG (0x0200)



LP_SYSTEM_LDO_FLASH_DISCHARGE 配置是否对 LDO flash 放电。

0：无作用

1：放电

(R/W)

LP_SYSTEM_LP_PUF_MEM_ISO_EN 配置是否使能 LP PUF 存储器隔离。

0：禁用

1：使能

(R/W)

LP_SYSTEM_LP_PUF_MEM_XPD 配置是否上电 LP PUF 存储器。

0：断电

1：上电

(R/W)

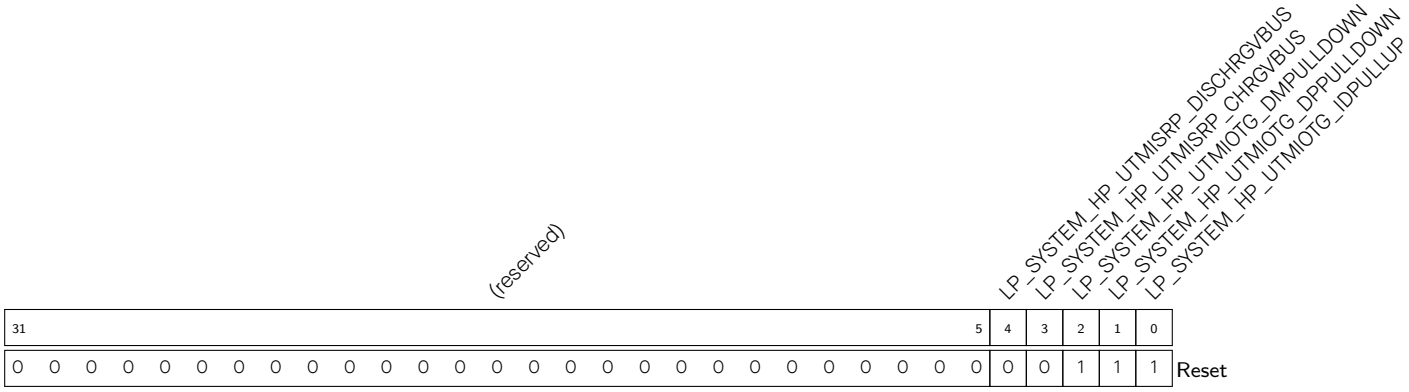
LP_SYSTEM_LP_PUF_MEM_DISCHARGE 配置是否对 LP PUF 存储器放电。

0：无作用

1：放电

(R/W)

Register 21.135. LP_SYSTEM_HP_USB_OTGHS_PHY_CTRL_REG (0x0204)



LP_SYSTEM_HP_UTMIOTG_IDPULLUP 配置是否上拉 USB OTG2.0 PHY ID。

0：不上拉

1：上拉

(R/W)

LP_SYSTEM_HP_UTMIOTG_DPPULLDOWN 配置是否下拉 USB OTG2.0 PHY DP。

0：不下拉

1：下拉

(R/W)

LP_SYSTEM_HP_UTMIOTG_DMPULLDOWN 配置是否下拉 USB OTG2.0 PHY DM。

0：不下拉

1：下拉

(R/W)

LP_SYSTEM_HP_UTMISRP_CHRGVBUS 配置是否对 USB OTG2.0 PHY VBUS 充电。

0：无作用

1：充电

(R/W)

LP_SYSTEM_HP_UTMISRP_DISCHRGVBUS 配置是否对 USB OTG2.0 PHY VBUS 放电。

0：无作用

1：放电

(R/W)

第 22 章

辅助调试

22.1 概述

辅助调试模块提供一套调试功能，可用于软件开发时进行调试定位问题所在。

22.2 主要特性

- **读写监测**：监测一个高性能双核处理器（High-Performance CPU, HP CPU0 和 HP CPU1）总线是否在限定的存储器地址范围内进行读写操作，若在该地址范围内发生读写操作则触发中断。
- **栈指针 (SP) 监测**：监测栈指针是否超出限定的范围，若超出范围则产生中断。
- **程序计数器 (PC) 记录**：记录 PC，可以获得上一次 HP CPU0 或 HP CPU1 复位时的 PC 值。
- **总线访问记录**：记录总线访问信息，当 HP CPU0、HP CPU1 或 DMA 写了某个特殊值时，会记录此次写操作的总线类型、地址和 PC 值（仅记录 HP CPU0 或 HP CPU1 写操作的 PC），并将这些信息记录到 HP L2MEM 中。

22.3 功能描述

22.3.1 区域读写监测

为确认 HP CPU0 是否在某段地址范围（即区域）进行过读写行为，辅助调试模块能够监测 HP CPU0 总线，当总线在监测地址范围进行读写操作时会触发中断，详情请见章节 2 高性能处理器 中的表 2.4-1。当 HP CPU0 总线访问数据空间时，本章节将其称为数据总线；而当 HP CPU0 总线访问 AHB 外设空间时，本章节将其称为外设总线。辅助调试模块可同时监测数据总线在两段地址范围内的读写行为，假设命名为 HP CPU0 数据总线区域 0 和 HP CPU0 数据总线区域 1，外设总线也可同时监测两段地址范围，假设命名为 HP CPU0 外设总线区域 0 和 HP CPU0 外设总线区域 1。区域 0 和区域 1 根据需要进行配置。

HP CPU1 同理。

22.3.2 栈指针监测

为防止 HP CPU0 栈溢出或者错误的压栈出栈，辅助调试模块能够监测栈指针。当 HP CPU0 栈指针超过 HP CPU0 栈指针监测区域限定的上下边界时，模块会记录 PC 指针并产生中断，边界由软件进行配置。

HP CPU1 同理。

22.3.3 PC 记录

在某些时候软件开发者希望知道上次 HP CPU0 复位时的 PC 指针。比如，在程序卡死只能复位时，开发者可能希望读取复位时的 PC 指针以便知道程序在哪里卡死，然后进行调试。辅助调试模块可以记录 HP CPU0 复位时的 PC，方便开发者进行调试。

HP CPU1 同理。

22.3.4 CPU/DMA 总线访问记录

辅助调试模块能够实时记录 HP CPU0 数据总线、HP CPU1 数据总线和 DMA 总线的写行为，当总线在某个范围内发生写操作或者在某个范围内写某个特定的值时，此次操作的总线类型、地址和 PC（仅记录 HP CPU0 或 HP CPU1 写操作的 PC）等信息会被记录下来，并按照一定的格式存储到 HP L2MEM 中。此处提到的地址范围只能处于 HP SPM 或 HP L2MEM 的范围区间。

22.4 中断

以下辅助调试模块的中断源会生成 ASSIST_DEBUG_INT 中断信号。

- ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT：当 HP CPU0 数据总线在 HP CPU0 数据总线区域 0 读操作时触发。
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT：当 HP CPU0 数据总线在 HP CPU0 数据总线区域 0 写操作时触发。
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT：当 HP CPU0 数据总线在 HP CPU0 数据总线区域 1 读操作时触发。
- ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT：当 HP CPU0 数据总线在 HP CPU0 数据总线区域 1 写操作时时触发。
- ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT：当 HP CPU0 外设总线在 HP CPU0 外设总线区域 0 读操作时触发。
- ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT：当 HP CPU0 外设总线在 HP CPU0 外设总线区域 0 写操作时时触发。
- ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT：当 HP CPU0 外设总线在 HP CPU0 外设总线区域 1 读操作时触发。
- ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT：当 HP CPU0 外设总线在 HP CPU0 外设总线区域 1 写操作时时触发。
- ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT：当 HP CPU0 栈指针小于 HP CPU0 栈指针监测区域的下边界时触发。
- ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT：当 HP CPU0 栈指针大于 HP CPU0 栈指针监测区域的上边界时触发。
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT：当 HP CPU1 数据总线在 HP CPU1 数据总线区域 0 读操作时触发。
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT：当 HP CPU1 数据总线在 HP CPU1 数据总线区域 0 写操作时触发。

- ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT：当 HP CPU1 数据总线在 HP CPU1 数据总线区域 1 读操作时触发。
- ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT：当 HP CPU1 数据总线在 HP CPU1 数据总线区域 1 写操作时时触发。
- ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT：当 HP CPU1 外设总线在 HP CPU1 外设总线区域 0 读操作时触发。
- ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT：当 HP CPU1 外设总线在 HP CPU1 外设总线区域 0 写操作时时触发。
- ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT：当 HP CPU1 外设总线在 HP CPU1 外设总线区域 1 读操作时触发。
- ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT：当 HP CPU1 外设总线在 HP CPU1 外设总线区域 1 写操作时时触发。
- ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT：当 HP CPU1 栈指针小于 HP CPU1 栈指针监测区域的下边界时触发。
- ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT：当 HP CPU1 栈指针大于 HP CPU1 栈指针监测区域的上边界时触发。

说明：
对于中断、中断信号、中断源及其关系的详细描述，可以查看章节 13 中断矩阵 > Section 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 22.6 寄存器列表。

22.5 工作流程

22.5.1 区域监测和栈指针监测配置

区域监测可监测 HP CPU0 和 HP CPU1 的数据总线和外设总线的读写行为，每个总线可同时监测两个区域。详细监测模式如下。

- 数据总线读写监测
  - 监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 0 是否进行读操作
  - 监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 0 是否进行写操作
  - 监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 1 是否进行读操作
  - 监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 1 是否进行写操作
  - 监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 0 是否进行读操作
  - 监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 0 是否进行写操作
  - 监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 1 是否进行读操作
  - 监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 1 是否进行写操作

- 外设总线读写监测
  - 监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 0 是否进行读操作
  - 监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 0 是否进行写操作
  - 监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 1 是否进行读操作
  - 监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 1 是否进行写操作
  - 监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 0 是否进行读操作
  - 监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 0 是否进行写操作
  - 监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 1 是否进行读操作
  - 监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 1 是否进行写操作
- 栈指针监测
  - 监测 HP CPU0 栈指针是否大于 HP CPU0 栈指针监测区域上边界
  - 监测 HP CPU0 栈指针是否小于 HP CPU0 栈指针监测区域下边界
  - 监测 HP CPU1 栈指针是否大于 HP CPU1 栈指针监测区域上边界
  - 监测 HP CPU1 栈指针是否小于 HP CPU1 栈指针监测区域下边界

区域监测和栈指针监测的配置流程如下：

1. 配置监测区域和栈指针
   - HP CPU0 数据总线区域 0 由 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG 进行配置。
   - HP CPU0 数据总线区域 1 由 ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG 进行配置。
   - HP CPU0 外设总线区域 0 由 ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG 进行配置。
   - HP CPU0 外设总线区域 1 由 ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG 进行配置。
   - HP CPU0 栈指针监测区域由 ASSIST_DEBUG_CORE_0_SP_MIN_REG 和 ASSIST_DEBUG_CORE_0_SP_MAX_REG 进行配置。
   - HP CPU1 数据总线区域 0 由 ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG 和 ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG 进行配置。
   - HP CPU1 数据总线区域 1 由 ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG 和 ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG 进行配置。
   - HP CPU1 外设总线区域 0 由 ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG 和 ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG 进行配置。
   - HP CPU1 外设总线区域 1 由 ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG 和 ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG 进行配置。
   - HP CPU1 栈指针监测区域由 ASSIST_DEBUG_CORE_1_SP_MIN_REG 和 ASSIST_DEBUG_CORE_1_SP_MAX_REG 进行配置。

2. 配置中断
    - 配置 ASSIST_DEBUG_CORE_0_INTR_ENA_REG 或 ASSIST_DEBUG_CORE_1_INTR_ENA_REG 用于使能不同模式的中断。
    - 读取 ASSIST_DEBUG_CORE_0_INTR_RAW_REG 或 ASSIST_DEBUG_CORE_1_INTR_RAW_REG 用于查询不同模式的原始中断状态。
    - 配置 ASSIST_DEBUG_CORE_0_INTR_CLR_REG 或 ASSIST_DEBUG_CORE_1_INTR_CLR_REG 用于清除不同模式的中断。
3. 配置 ASSIST_DEBUG_CORE_0_MONTR_ENA_REG 或 ASSIST_DEBUG_CORE_1_MONTR_ENA_REG 使能不同的监测模式，可同时使能。

比如，若想监测 HP CPU0 数据总线地址 A 到地址 B 范围内是否进行过写操作，可通过 HP CPU0 数据总线区域 0 或者区域 1 进行监测，以 HP CPU0 数据总线区域 0 为例：

1. 配置 ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 为 1，使能 HP CPU0 更新 PC 信号给辅助调试模块。
2. 配置 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG 为 A 地址。
3. 配置 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG 为 B 地址。
4. 配置 ASSIST_DEBUG_CORE_0_INTR_ENA_REG bit[1] 使能 HP CPU0 数据总线区域 0 写监测中断。
5. 配置 ASSIST_DEBUG_CORE_0_MONTR_ENA_REG bit[1] 使能 HP CPU0 数据总线区域 0 写监测。
6. 配置中断矩阵将 ASSIST_DEBUG_INTR 映射到 HP CPU0 中断或 HP CPU1 中断（参考章节 13 中断矩阵）。
7. 发生中断后
    - 读取 ASSIST_DEBUG_CORE_0_INTR_RAW_REG 和 ASSIST_DEBUG_CORE_1_INTR_RAW_REG 确认是什么操作触发的中断。
    - 如果是 HP CPU0 区域监测引发的中断，读取 ASSIST_DEBUG_CORE_0_AREA_PC 可获取触发中断时刻的 HP CPU0 PC 值，读取 ASSIST_DEBUG_CORE_0_AREA_SP 可获取触发中断时刻的 HP CPU0 SP 值。
    - 如果是栈指针监测引发的中断，读取 ASSIST_DEBUG_CORE_0_SP_PC 可获取触发中断时刻的 HP CPU0 PC 值。
    - 配置 ASSIST_DEBUG_CORE_0_INTR_CLR_REG 和 ASSIST_DEBUG_CORE_1_INTR_CLR_REG 不同位可清除不同的中断。

22.5.2 PC 记录配置

配置 ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 为 1，使能 HP CPU0 更新 PC 信号给辅助调试模块。同时当 ASSIST_DEBUG_CORE_0_RCD_RECORDEN 配置为 1 时，
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 会去记录 HP CPU0 PC 信号，同时
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG 会记录 HP CPU0 SP 值，否则保持原值。

当 HP CPU0 发生复位时，ASSIST_DEBUG_CORE_0_RCD_EN_REG 会被复位，但是
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 和 ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG 不会被复位，因此这两个寄存器会一直保持 HP CPU0 复位时刻的 PC 值和 SP 值。

HP CPU1 同理。

22.5.3 CPU/DMA 总线访问记录配置

22.5.3.1 HP CPU0/1 总线访问记录配置

HP CPU0/1 总线访问记录（SPM 监控功能）的配置流程如下：

1. 配置监测地址范围
   - 配置 SPM_MEM_MONITOR_LOG_MIN_REG 和 SPM_MEM_MONITOR_LOG_MAX_REG 确定地址范围。监测地址范围需处于 0x3010_0000 ~ 0x3010_1FFF 区间内。
2. 配置监测模式 (SPM_MEM_MONITOR_LOG_MODE)
   - 写监测（监测总线是否发生写操作）
   - 字 (word) 监测（监测总线是否写某个特定 word）
   - 半字 (halfword) 监测（监测总线是否写某个特定 halfword）
   - 字节 (byte) 监测（监测总线是否写某个特定 byte）
3. 配置需要监测的特殊值
   - 监测模式为 word 监测时，SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 即为监测的特定 word。
   - 监测模式为 halfword 监测时，SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 的 [15:0] 即为监测的特定 halfword。
   - 监测模式为 byte 监测时，SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 的 [7:0] 即为监测的特定 byte。
   - SPM_MEM_MONITOR_LOG_DATA_MASK_REG 用于屏蔽 SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 的相应 byte。当某一 byte 被屏蔽，表示该 byte 可为任意的值，比如 word 监测，SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 配置为 0x01020304，SPM_MEM_MONITOR_LOG_DATA_MASK_REG 配置为 0x1，因此只要总线写 0x010203XX 都会被记录。
4. 配置记录信息的存储地址范围
   - SPM_MEM_MONITOR_LOG_MEM_START_REG 和 SPM_MEM_MONITOR_LOG_MEM_END_REG 为存储地址范围配置寄存器。记录信息的存储地址范围为 0x4FF0_0000 ~ 0x4FFB_FFFF。
   - 置位 SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG，使 SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 更新为 SPM_MEM_MONITOR_LOG_MEM_START_REG。
   - 配置辅助调试模块的 SPM 监控功能对 HP L2MEM 的使用权限，只有开启了辅助调试模块对 HP L2MEM 的使用权限才能访问 HP L2MEM。关于如何配置，详情见章节 20 权限控制 *(PMS)*) 的 TEE_DMA_SPM_MON_PMS_W_REG 及相关内容。
5. 配置存储记录信息的内存的存储模式：loop 模式和非 loop 模式
   - loop 模式下，将记录信息循环写进配置地址范围内，当写到结束地址时回到起始地址开始写，覆盖之前记录的信息。置位 SPM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使用 loop 模式。
     例如，存储记录信息的地址范围为 0 ~ 4 ，总线访问过程中有 1 ~ 10 共十次写记录信息的操作，那么第 5 次写操作写到地址 4 后，第 6 次写操作会回到地址 0 开始写，第 6 ~ 10 写操作会覆盖之前的第 1 ~ 5 写操作的数据。

- 非 loop 模式下，当将记录信息写到结束地址后，会一直停留在结束地址，并不再写记录信息，不会覆盖最开始的记录信息，并且丢弃后面的记录信息。清零 SPM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使用非 loop 模式。
  例如，存储记录信息的地址范围为 0 ~ 4 ，总线访问过程中有 1 ~ 10 共十次写记录信息的操作，那么第 5 次写操作写到地址 4 后，第 6 ~ 10 次写操作的地址会一直在地址 4 上，但不再进行写操作，因此地址范围 0 ~ 4 保留前 5 次写操作信息，并且丢弃第 6 ~ 10 次写操作信息。

6. 配置总线使能

   - 配置 SPM_MEM_MONITOR_LOG_CORE_ENA 使能 HP CPU0 或 HP CPU1 总线访问记录，可同时使能。

辅助调试模块会先将 HP CPU0 或 HP CPU1 的记录信息缓存到一个内部 buffer A 里，然后从 buffer 取出记录信息再存储到配置的存储区域内。当监测行为被连续触发时，连续产生的记录数据包可能会使得 buffer 满而不能缓存新的记录数据包，此时模块会舍弃掉不能及时缓存到 buffer 的记录数据包，并在 buffer 不满的时候缓存一个 HP CPU LOST 数据包作为替代。然而仅能知道 HP CPU LOST 数据包生成时所舍弃的记录数据包的总线类型，不能知道在此之前舍弃了多少个记录数据包。

当总线访问记录结束后需要从内存中读取记录数据并进行解码。记录数据里只有三种包格式，HP CPU0 数据包、HP CPU1 数据包和 HP CPU LOST 数据包，HP CPU0/1 数据包分别对应 HP CPU0/1 数据总线记录信息，三种包格式如表 22.5-1 和 22.5-2 所示：

表 22.5-1. HP CPU0/1 包格式

Bit[63:33]	Bit[32]	Bit[31:3]	Bit[2:1]	Bit[0]
pc_offset	anchored(1)	addr_offset	format	anchored(0)

表 22.5-2. LOST 包格式

Bit[31:7]	Bit[6:3]	Bit[2:1]	Bit[0]
reserved	lost_source	format	anchored(0)

从包格式可以看出，HP CPU0/1 数据包共 64 位，HP CPU LOST 数据包共 32 位。下面介绍各个域的含义：

- **format** 表示此次数据包的类型，0 表示 HP CPU0 数据包，1 表示 HP CPU1 数据包，2 表示 HP CPU LOST 数据包，3 为保留值。
- **pc_offset** 记录了 HP CPU0/1 的 PC 指针的偏移量，实际 PC = pc_offset + 0x0000_0000。
- **addr_offset** 记录了此次写操作的地址偏移，实际地址 = addr_offset + {SPM_MEM_MONITOR_LOG_MIN_REG[31:4], 4'h0}。
- **lost_source** 记录了 HP CPU LOST 数据包生成时舍弃了哪些数据包：
  - Bit[4:3]：0 表示未舍弃 HP CPU0 数据包，3 表示舍弃了 HP CPU0 数据包，其他值为保留值。
  - Bit[6:5]：0 表示未舍弃 HP CPU1 数据包，3 表示舍弃了 HP CPU1 数据包，其他值为保留值。
- **anchored** 表示该 32 位数据在数据包中的位置，0 表示该 32 位数据是数据包的低 32 位，1 表示该 32 位数据是数据包的高 32 位。

模块内部 buffer 的数据宽度是 32 位。当同时使能 HP CPU0 和 HP CPU1 总线访问记录并且在某一时刻同时生成了记录数据，先将 HP CPU0 数据包缓存到 buffer 中，再缓存 HP CPU1 数据包。辅助调试模块会自动地将缓存数据从 buffer 取出，并将取出的数据再以 32 位数据宽度存入指定范围的 memory 里。当采用 loop 模式时，

若在配置的存储地址范围内循环了若干次，可能存在残留数据干扰解析，即 HP CPU0/1 数据包的低 32 位数据被覆盖而使得高 32 位数据成为残留数据，因此确定第一个有效数据包的位置时必须过滤可能存在的残留数据。使用 SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 确定了数据包的起始位置之后，检查数据包 anchored 位的数值，是 0 则保留，是 1 则舍弃。

数据包解析流程如下：

- SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG 确定数据是否溢出配置范围。
  - 如果未溢出，则数据读取的地址范围为 SPM_MEM_MONITOR_LOG_MEM_START_REG ~ SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4 ；
  - 若溢出并且开启了 loop 模式，则数据读取的地址范围为 SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ SPM_MEM_MONITOR_LOG_MEM_END_REG 和 SPM_MEM_MONITOR_LOG_MEM_START_REG ~ SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4；
  - 若溢出并且未开启 loop 模式，则数据读取的地址范围为 SPM_MEM_MONITOR_LOG_MEM_START_REG ~ SPM_MEM_MONITOR_LOG_MEM_END_REG。
- 从起始地址处开始读取数据并解析，每次读取 32 位。

数据解析完成之后需要通过配置 SPM_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 清除 SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

22.5.3.2 DMA 总线访问记录配置

辅助调试模块共支持 4 个 DMA 总线通道组，分别为 DMA_0 通道组、DMA_1 通道组、DMA_2 通道组和 DMA_3 通道组。其中 DMA_0 通道组和 DMA_1 通道组为保留通道组，不可使用；DMA_2 通道组的 DMA 总线是 AXI 矩阵访问 HP L2MEM 的总线，DMA_3 通道组的 DMA 总线是 AHB 矩阵访问 HP L2MEM 的总线，详情请见章节 9 系统和存储器 > 9.3.2 小节。DMA 总线访问记录（L2MEM 监控功能）的配置流程如下：

1. 配置监测地址范围
   - 配置 L2_MEM_MONITOR_LOG_MIN_REG 和 L2_MEM_MONITOR_LOG_MAX_REG 确定地址范围。监测地址范围需处于 0x4FF0_0000 ~ 0x4FFB_FFFF 或 0x8FF0_0000 - 0x8FFB_FFFF 区间内。
2. 配置监测模式 (L2_MEM_MONITOR_LOG_MODE)
   - 写监测（监测总线是否发生写操作）
   - 字 (word) 监测（监测总线是否写某个特定 word）
   - 半字 (halfword) 监测（监测总线是否写某个特定 halfword）
   - 字节 (byte) 监测（监测总线是否写某个特定 byte）
3. 配置需要监测的特殊值
   - 监测模式为 word 监测时，L2_MEM_MONITOR_LOG_CHECK_DATA_REG 即为监测的特定 word。
   - 监测模式为 halfword 监测时，L2_MEM_MONITOR_LOG_CHECK_DATA_REG 的 [15:0] 即为监测的特定 halfword。
   - 监测模式为 byte 监测时，L2_MEM_MONITOR_LOG_CHECK_DATA_REG 的 [7:0] 即为监测的特定 byte。

- L2_MEM_MONITOR_LOG_DATA_MASK_REG 用于屏蔽 L2_MEM_MONITOR_LOG_CHECK_DATA_REG 的相应 byte。当某一 byte 被屏蔽，表示该 byte 可为任意的值，比如 word 监测，L2_MEM_MONITOR_LOG_CHECK_DATA_REG 配置为 0x01020304，L2_MEM_MONITOR_LOG_DATA_MASK_REG 配置为 0x1，因此只要总线写 0x010203XX 都会被记录。

4. 配置记录信息的存储地址范围
   - L2_MEM_MONITOR_LOG_MEM_START_REG 和 L2_MEM_MONITOR_LOG_MEM_END_REG 为存储地址范围配置寄存器。记录信息的存储地址范围为 0x4FF0_0000 ~ 0x4FFB_FFFF。
   - 置位 L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG，使 L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 更新为 L2_MEM_MONITOR_LOG_MEM_START_REG。
   - 配置辅助调试模块的 L2MEM 监控功能对 HP L2MEM 的使用权限，只有开启了辅助调试模块对 HP L2MEM 的使用权限才能访问 HP L2MEM。关于如何配置，详情见章节 20 权限控制 *(PMS)* 的 TEE_DMA_L2MEM_MON_PMS_W_REG 及相关内容。
5. 配置存储记录信息的内存的存储模式：loop 模式和非 loop 模式
   - loop 模式下，将记录信息循环写进配置地址范围内，当写到结束地址时回到起始地址开始写，覆盖之前记录的信息。置位 L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使用 loop 模式。
   - 非 loop 模式下，当将记录信息写到结束地址后，会一直停留在结束地址，并不再写记录信息，不会覆盖最开始的记录信息，并且丢弃后面的记录信息。清零 L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使用非 loop 模式。
   - 示例同章节 22.5.3.1 的步骤 5。
6. 配置总线使能
   - 配置 L2_MEM_MONITOR_LOG_DMA_2_ENA 和 L2_MEM_MONITOR_LOG_DMA_3_ENA 分别使能 DMA_2 通道组和 DMA_3 通道组总线访问记录，可同时使能。

辅助调试模块会先将 DMA 记录信息缓存到一个内部 buffer B 里，然后从 buffer 取出记录信息再存储到配置的存储区域内。当监测行为被连续触发时，连续产生的记录数据包可能会使得 buffer 满而不能缓存新的记录数据包，此时模块会舍弃掉不能及时缓存到 buffer 的记录数据包，并在 buffer 不满的时候缓存一个 DMA LOST 数据包作为替代，此时仅能知道 DMA LOST 数据包生成时所舍弃的记录数据包的总线类型，但不能知道在此之前舍弃了多少个记录数据包。

当总线访问记录结束后需要从内存中读取记录数据并进行解码。记录数据里只有三种包格式，DMA_2 数据包、DMA_3 数据包和 DMA LOST 数据包，DMA_2 数据包对应 DMA_2 通道组数据总线记录信息，DMA_3 数据包对应 DMA_3 通道组数据总线记录信息，三种包格式如表 22.5-3、22.5-4 和 22.5-5 所示：

表 22.5-3. DMA_2 包格式

Bit[31:9]	Bit[8:4]	Bit[3:1]	Bit[0]
addr_offset0	reserved	format	anchored(0)

表 22.5-4. DMA_3 包格式

Bit[31:9]	Bit[8:4]	Bit[3:1]	Bit[0]
addr_offset1	reserved	format	anchored(0)

表 22.5-5. DMA LOST 包格式

Bit[31:10]	Bit[9:8]	Bit[7:4]	Bit[3:1]	Bit[0]
reserved	lost_source	reserved	format	anchored(0)

从包格式可以看出，DMA_2 、DMA_3 和 DMA LOST 数据包均为 32 位。下面介绍各个域的含义：

- **format** 表示此次数据包的类型，4 表示 DMA_2 数据包，5 表示 DMA_3 数据包，6 表示 DMA LOST 数据包，其他值为保留值。
- **addr_offset0** 记录了此次写操作的地址偏移，实际地址 = addr_offset0 + {L2_MEM_MONITOR_LOG_MIN_REG[31:5]，5'h0}。
- **addr_offset1** 记录了此次写操作的地址偏移，实际地址 = addr_offset1 + {L2_MEM_MONITOR_LOG_MIN_REG[31:2]，2'h0}。
- **lost_source** 记录了 DMA LOST 数据包生成时舍弃了哪些数据包：
  - Bit[8]：0 表示未舍弃 DMA_2 数据包，1 表示舍弃了 DMA_2 数据包。
  - Bit[9]：0 表示未舍弃 DMA_3 数据包，1 表示舍弃了 DMA_3 数据包。
- **anchored** 表示该 32 位数据在数据包中的位置，0 表示该 32 位数据是数据包的低 32 位。

模块内部 buffer 的数据宽度是 32 位。当同时使能 DMA_2 通道组或 DMA_3 通道组总线访问记录并且在某一时刻同时生成了记录数据，先将 DMA_2 的数据包缓存到 buffer 中，再缓存 DMA_3 的数据包。辅助调试模块会自动地将缓存数据从 buffer 取出，并将取出的数据再以 32 位数据宽度存入指定范围的 memory 里。

数据包解析流程如下：

- L2_MEM_MONITOR_LOG_MEM_FULL_FLAG 确定数据是否溢出配置范围。
  - 如果未溢出，则数据读取的地址范围为 L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4 ；
  - 若溢出并且开启了 loop 模式，则数据读取的地址范围为 L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ L2_MEM_MONITOR_LOG_MEM_END_REG 和 L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4；
  - 若溢出并且未开启 loop 模式，则数据读取的地址范围为 L2_MEM_MONITOR_LOG_MEM_START_REG ~ L2_MEM_MONITOR_LOG_MEM_END_REG。
- 从起始地址处开始读取数据并解析，每次读取 32 位。

数据解析完成之后需要通过配置 L2_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 清除 L2_MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

22.6 寄存器列表

本小节中 **HP CPU 总线记录配置寄存器**的地址为相对于 **SPM 监控**基地址的地址偏移量（相对地址），详见 22.6.1，**DMA 总线记录配置寄存器**的地址为相对于 **L2MEM 监控**基地址的地址偏移量（相对地址），详见 22.6.2，其它寄存器的所有地址均为相对于 **总线监控**基地址的地址偏移量（相对地址），详见 22.6.3。具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

22.6.1 HP CPU 总线记录配置寄存器列表

名称	描述	地址	访问
HP CPU 总线记录配置寄存器			
SPM_MEM_MONITOR_LOG_SETTING_REG	总线访问记录配置寄存器	0x0000	R/W
SPM_MEM_MONITOR_LOG_CHECK_DATA_REG	总线访问监测数据配置寄存器	0x0008	R/W
SPM_MEM_MONITOR_LOG_DATA_MASK_REG	总线访问监测数据屏蔽配置寄存器	0x000C	R/W
SPM_MEM_MONITOR_LOG_MIN_REG	总线访问监测范围配置寄存器	0x0010	R/W
SPM_MEM_MONITOR_LOG_MAX_REG	总线访问监测范围配置寄存器	0x0014	R/W
SPM_MEM_MONITOR_LOG_MEM_START_REG	记录数据写入内存的起始地址	0x0018	R/W
SPM_MEM_MONITOR_LOG_MEM_END_REG	记录数据写入内存的结束地址	0x001C	R/W
SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	表示下一次写内存的写地址	0x0020	RO
SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	更新下一次写内存的写地址为记录数据写入内存的起始地址	0x0024	WT
SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	记录溢出状态寄存器	0x0028	不定
时钟控制寄存器			
SPM_MEM_MONITOR_CLOCK_GATE_REG	寄存器时钟控制	0x002C	R/W
版本寄存器			
SPM_MEM_MONITOR_DATE_REG	版本寄存器	0x03FC	R/W

22.6.2 DMA 总线记录配置寄存器列表

名称	描述	地址	访问
DMA 总线记录配置寄存器			
L2_MEM_MONITOR_LOG_SETTING_REG	总线访问记录配置寄存器	0x0000	R/W
L2_MEM_MONITOR_LOG_SETTING1_REG	总线访问记录配置寄存器	0x0004	R/W
L2_MEM_MONITOR_LOG_CHECK_DATA_REG	总线访问监测数据配置寄存器	0x0008	R/W
L2_MEM_MONITOR_LOG_DATA_MASK_REG	总线访问监测数据屏蔽配置寄存器	0x000C	R/W
L2_MEM_MONITOR_LOG_MIN_REG	总线访问监测范围配置寄存器	0x0010	R/W
L2_MEM_MONITOR_LOG_MAX_REG	总线访问监测范围配置寄存器	0x0014	R/W
L2_MEM_MONITOR_LOG_MEM_START_REG	记录数据写入内存的起始地址	0x0018	R/W
L2_MEM_MONITOR_LOG_MEM_END_REG	记录数据写入内存的结束地址	0x001C	R/W
L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	表示下一次写内存的写地址	0x0020	RO

名称	描述	地址	访问
L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	更新下一次写内存的写地址为记录数据写入内存的起始地址	0x0024	WT
L2_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	记录溢出状态寄存器	0x0028	不定
时钟控制寄存器			
L2_MEM_MONITOR_CLOCK_GATE_REG	寄存器时钟控制	0x002C	R/W
版本寄存器			
L2_MEM_MONITOR_DATE_REG	版本寄存器	0x03FC	R/W

22.6.3 其它寄存器列表

名称	描述	地址	访问
监测配置寄存器			
ASSIST_DEBUG_CORE_0_MONTR_ENA_REG	HP CPU0 监测使能配置寄存器	0x0000	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG	HP CPU0 数据总线区域 0 监测地址下边界配置寄存器	0x0010	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG	HP CPU0 数据总线区域 0 监测地址上边界配置寄存器	0x0014	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG	HP CPU0 数据总线区域 1 监测地址下边界配置寄存器	0x0018	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG	HP CPU0 数据总线区域 1 监测地址上边界配置寄存器	0x001C	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG	HP CPU0 外设总线区域 0 监测地址下边界配置寄存器	0x0020	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG	HP CPU0 外设总线区域 0 监测地址上边界配置寄存器	0x0024	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG	HP CPU0 外设总线区域 1 监测地址下边界配置寄存器	0x0028	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG	HP CPU0 外设总线区域 1 监测地址上边界配置寄存器	0x002C	R/W
ASSIST_DEBUG_CORE_0_AREA_PC_REG	HP CPU0 区域监测 PC 状态寄存器	0x0030	RO
ASSIST_DEBUG_CORE_0_AREA_SP_REG	HP CPU0 区域监测 SP 状态寄存器	0x0034	RO
ASSIST_DEBUG_CORE_0_SP_MIN_REG	HP CPU0 栈指针监测地址下边界配置寄存器	0x0038	R/W
ASSIST_DEBUG_CORE_0_SP_MAX_REG	HP CPU0 栈指针监测地址上边界配置寄存器	0x003C	R/W
ASSIST_DEBUG_CORE_0_SP_PC_REG	HP CPU0 栈指针监测 PC 状态寄存器	0x0040	RO
ASSIST_DEBUG_CORE_1_MONTR_ENA_REG	HP CPU1 监测使能配置寄存器	0x0080	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG	HP CPU1 数据总线区域 0 监测地址下边界配置寄存器	0x0090	R/W

名称	描述	地址	访问
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG	HP CPU1 数据总线区域 0 监测地址上边界配置寄存器	0x0094	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG	HP CPU1 数据总线区域 1 监测地址下边界配置寄存器	0x0098	R/W
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG	HP CPU1 数据总线区域 1 监测地址上边界配置寄存器	0x009C	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG	HP CPU1 外设总线区域 0 监测地址下边界配置寄存器	0x00A0	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG	HP CPU1 外设总线区域 0 监测地址上边界配置寄存器	0x00A4	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG	HP CPU1 外设总线区域 1 监测地址下边界配置寄存器	0x00A8	R/W
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG	HP CPU1 外设总线区域 1 监测地址上边界配置寄存器	0x00AC	R/W
ASSIST_DEBUG_CORE_1_AREA_PC_REG	HP CPU1 区域监测 PC 状态寄存器	0x00B0	RO
ASSIST_DEBUG_CORE_1_AREA_SP_REG	HP CPU1 区域监测 SP 状态寄存器	0x00B4	RO
ASSIST_DEBUG_CORE_1_SP_MIN_REG	HP CPU1 栈指针监测地址下边界配置寄存器	0x00B8	R/W
ASSIST_DEBUG_CORE_1_SP_MAX_REG	HP CPU1 栈指针监测地址上边界配置寄存器	0x00BC	R/W
ASSIST_DEBUG_CORE_1_SP_PC_REG	HP CPU1 栈指针监测 PC 状态寄存器	0x00C0	RO
中断配置寄存器			
ASSIST_DEBUG_CORE_0_INTR_RAW_REG	HP CPU0 原始中断状态寄存器	0x0004	RO
ASSIST_DEBUG_CORE_0_INTR_ENA_REG	HP CPU0 中断使能配置寄存	0x0008	R/W
ASSIST_DEBUG_CORE_0_INTR_CLR_REG	HP CPU0 清除中断配置寄存器	0x000C	WT
ASSIST_DEBUG_CORE_1_INTR_RAW_REG	HP CPU1 原始中断状态寄存器	0x0084	RO
ASSIST_DEBUG_CORE_1_INTR_ENA_REG	HP CPU1 中断使能配置寄存	0x0088	R/W
ASSIST_DEBUG_CORE_1_INTR_CLR_REG	HP CPU1 清除中断配置寄存器	0x008C	WT
PC 记录配置寄存器			
ASSIST_DEBUG_CORE_0_RCD_EN_REG	HP CPU0 PC 记录使能配置寄存器	0x0044	R/W
ASSIST_DEBUG_CORE_1_RCD_EN_REG	HP CPU1 PC 记录使能配置寄存器	0x00C4	R/W
PC 记录状态寄存器			
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG	HP CPU0 PC 记录寄存器	0x0048	RO
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG	HP CPU0 SP 记录寄存器	0x004C	RO
ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG	HP CPU1 PC 记录寄存器	0x00C8	RO
ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP_REG	HP CPU1 SP 记录寄存器	0x00CC	RO
CPU 状态寄存器			

名称	描述	地址	访问
ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	HP CPU0 异常前的最后一条指令的 PC	0x0070	RO
ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG	HP CPU0 处于调试模式的标志寄存器	0x0074	RO
ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXCEPTION_REG	HP CPU1 异常前的最后一条指令的 PC	0x00F0	RO
ASSIST_DEBUG_CORE_1_DEBUG_MODE_REG	HP CPU1 处于调试模式的标志寄存器	0x00F4	RO
时钟控制寄存器			
ASSIST_DEBUG_CLOCK_GATE_REG	寄存器时钟控制	0x0108	R/W
版本寄存器			
ASSIST_DEBUG_DATE_REG	版本寄存器	0x03FC	R/W

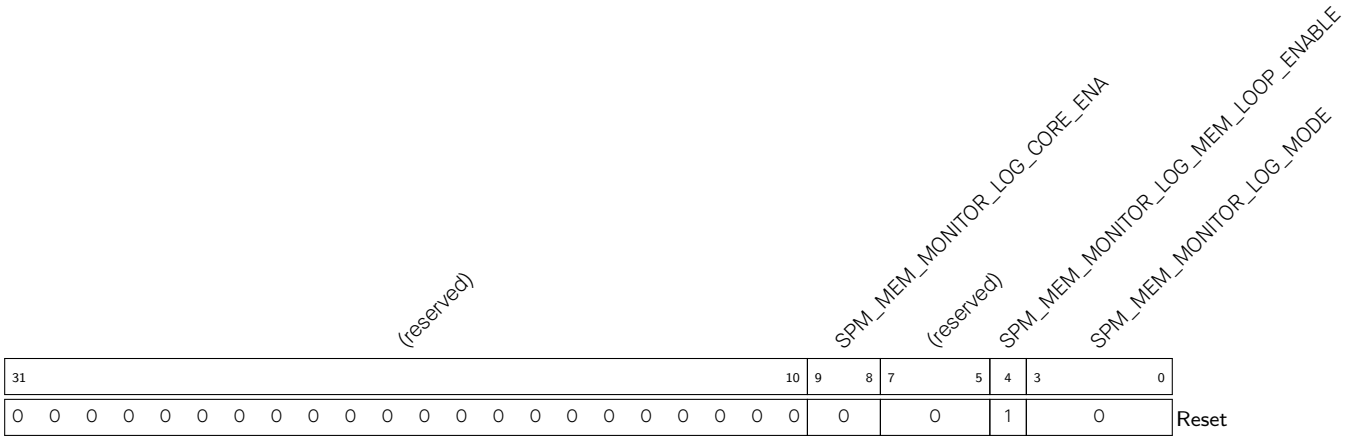
22.7 寄存器

本小节中 **HP CPU 总线记录配置寄存器**的地址为相对于 **SPM 监控**基地址的地址偏移量（相对地址），详见 22.7.1，**DMA 总线记录配置寄存器**的地址为相对于 **L2MEM 监控**基地址的地址偏移量（相对地址），详见 22.7.2，其它寄存器的所有地址均为相对于 **总线监控**基地址的地址偏移量（相对地址），详见 22.7.3，具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

22.7.1 HP CPU 总线记录配置寄存器

Register 22.1. SPM_MEM_MONITOR_LOG_SETTING_REG (0x0000)



SPM_MEM_MONITOR_LOG_MODE 配置监测模式。

bit[0]：写监测
0：不使能
1：使能
bit[1]：word 监测
0：不使能
1：使能
bit[2]：halfword 监测
0：不使能
1：使能
bit[3]：byte 监测
0：不使能
1：使能
(R/W)

见下页...

Register 22.1. SPM_MEM_MONITOR_LOG_SETTING_REG (0x0000)

接上页...

SPM_MEM_MONITOR_LOG_MEM_LOOP_ENABLE 配置写内存的存储模式。

1：loop 模式

0：非 loop 模式

(R/W)

SPM_MEM_MONITOR_LOG_CORE_ENA 配置是否使能 HP CPU0 或 HP CPU1 总线访问记录。

bit[0]：配置是否使能 HP CPU0 总线访问记录

0：不使能

1：使能

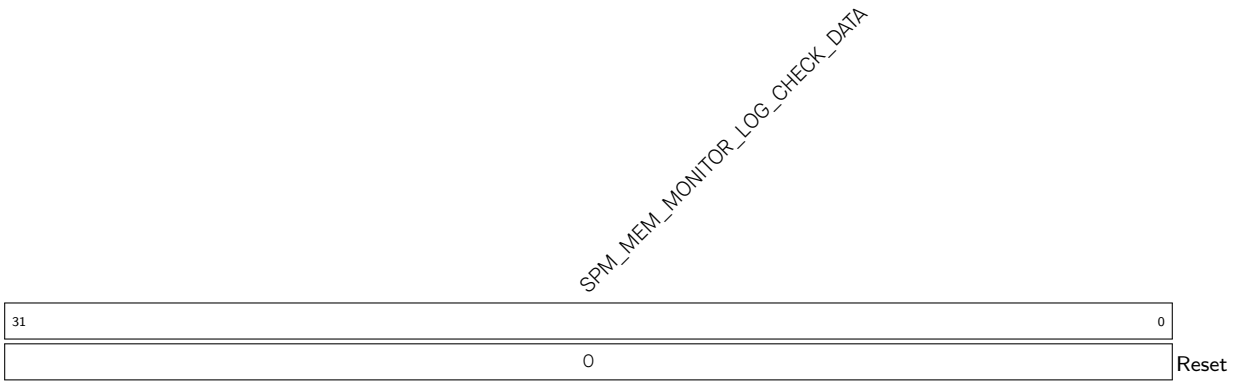
bit[1]：配置是否使能 HP CPU1 总线访问记录

0：不使能

1：使能

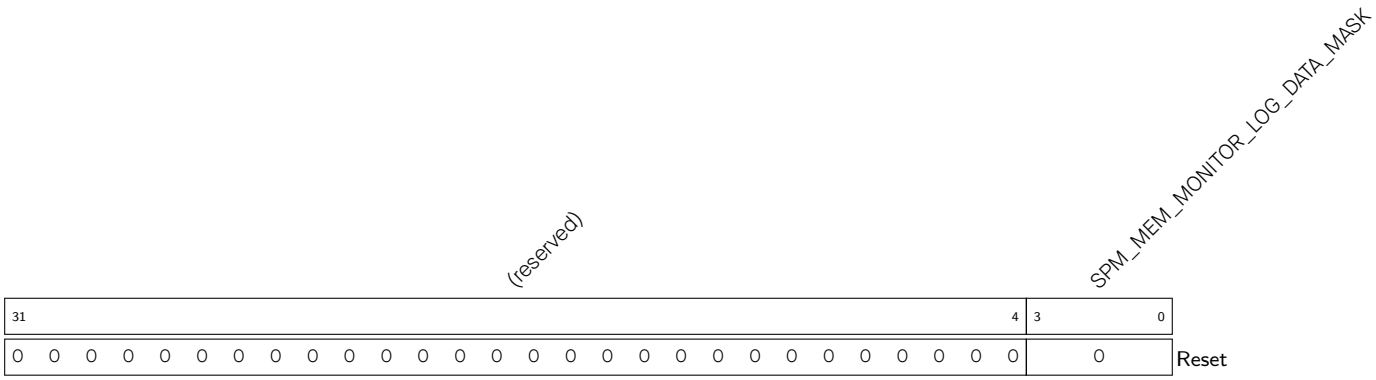
(R/W)

Register 22.2. SPM_MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



SPM_MEM_MONITOR_LOG_CHECK_DATA 配置总线访问监测的特殊值。(R/W)

Register 22.3. SPM_MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



SPM_MEM_MONITOR_LOG_DATA_MASK 是否屏蔽 SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 的相应字节。

bit[0]：是否屏蔽 SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 最低位字节。

0：不屏蔽

1：屏蔽

bit[1]：是否屏蔽 SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 第二个低位字节。

0：不屏蔽

1：屏蔽

bit[2]：是否屏蔽 SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 第二个高位字节。

0：不屏蔽

1：屏蔽

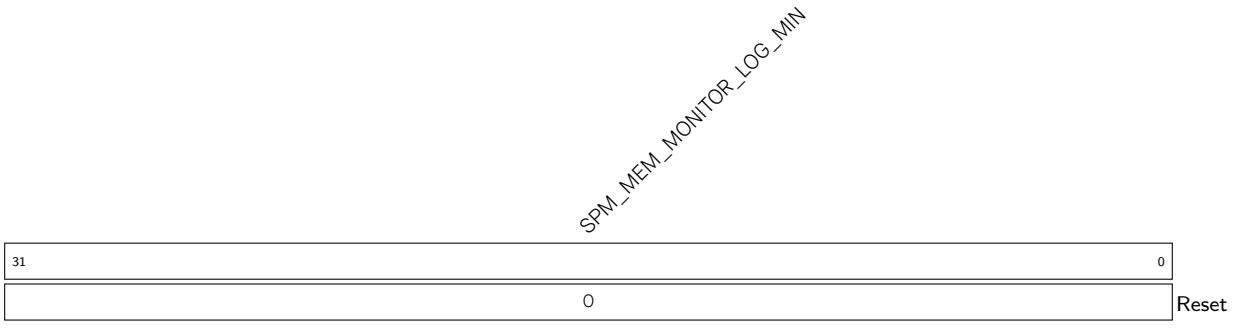
bit[3]：是否屏蔽 SPM_MEM_MONITOR_LOG_CHECK_DATA_REG 最高位字节。

0：不屏蔽

1：屏蔽

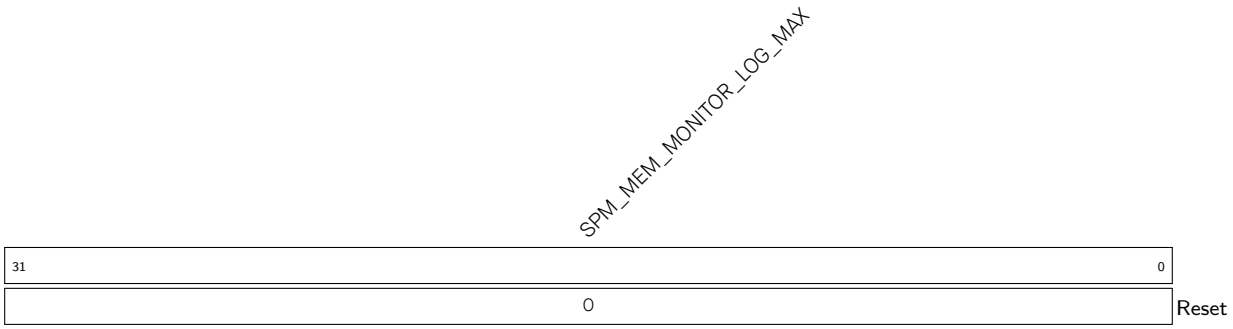
(R/W)

Register 22.4. SPM_MEM_MONITOR_LOG_MIN_REG (0x0010)



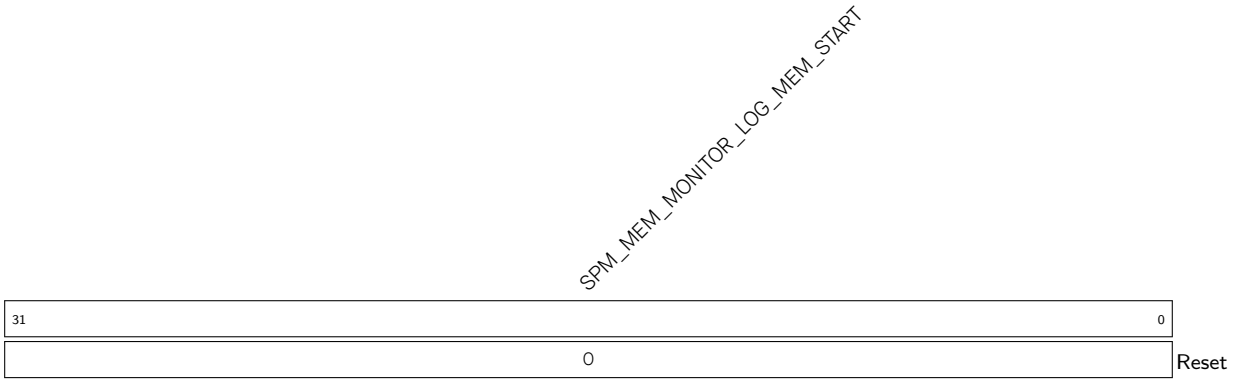
SPM_MEM_MONITOR_LOG_MIN 配置监测地址的下边界。(R/W)

Register 22.5. SPM_MEM_MONITOR_LOG_MAX_REG (0x0014)



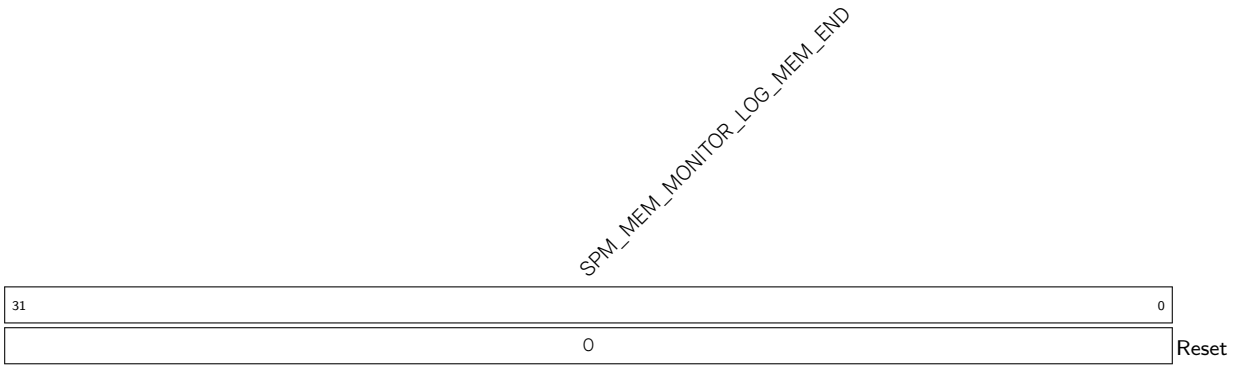
SPM_MEM_MONITOR_LOG_MAX 配置监测地址的上边界。(R/W)

Register 22.6. SPM_MEM_MONITOR_LOG_MEM_START_REG (0x0018)



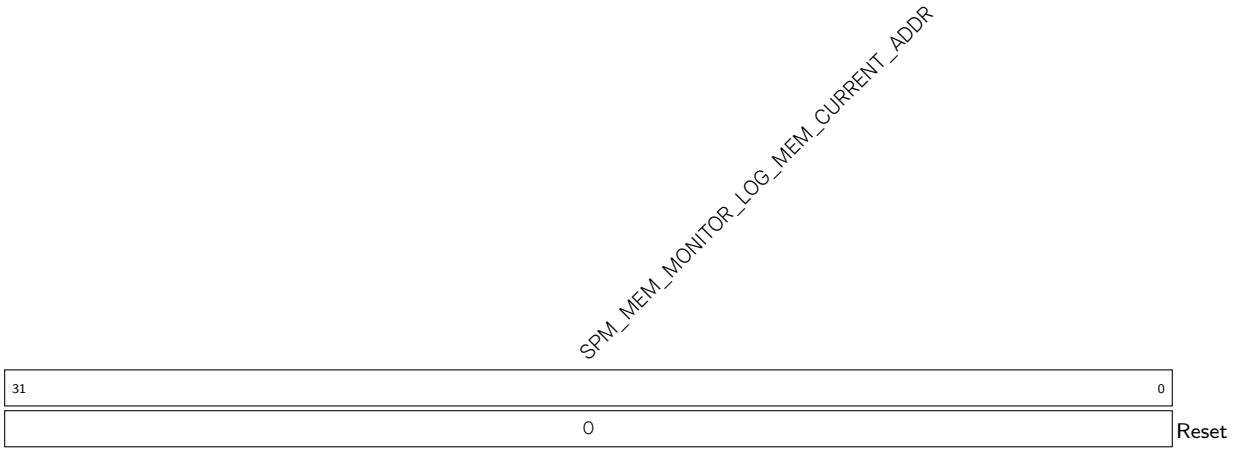
SPM_MEM_MONITOR_LOG_MEM_START 配置记录数据写入内存的起始地址。(R/W)

Register 22.7. SPM_MEM_MONITOR_LOG_MEM_END_REG (0x001C)



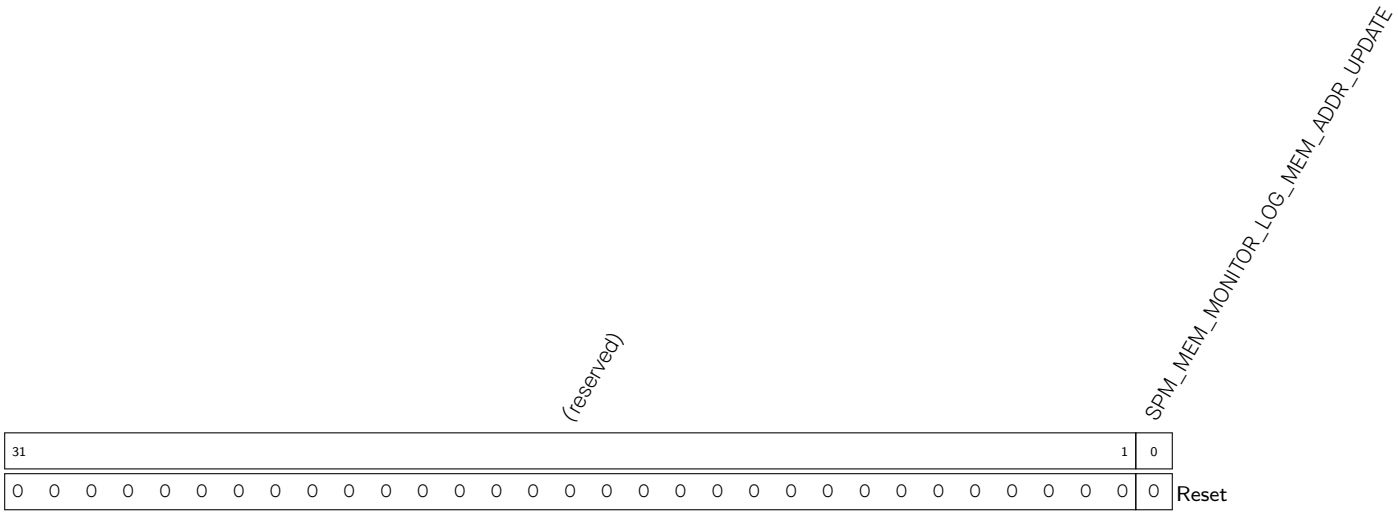
SPM_MEM_MONITOR_LOG_MEM_END 配置记录数据写入内存的结束地址。(R/W)

Register 22.8. SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0020)



SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR 表示下一次写内存的写地址。(RO)

Register 22.9. SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0024)



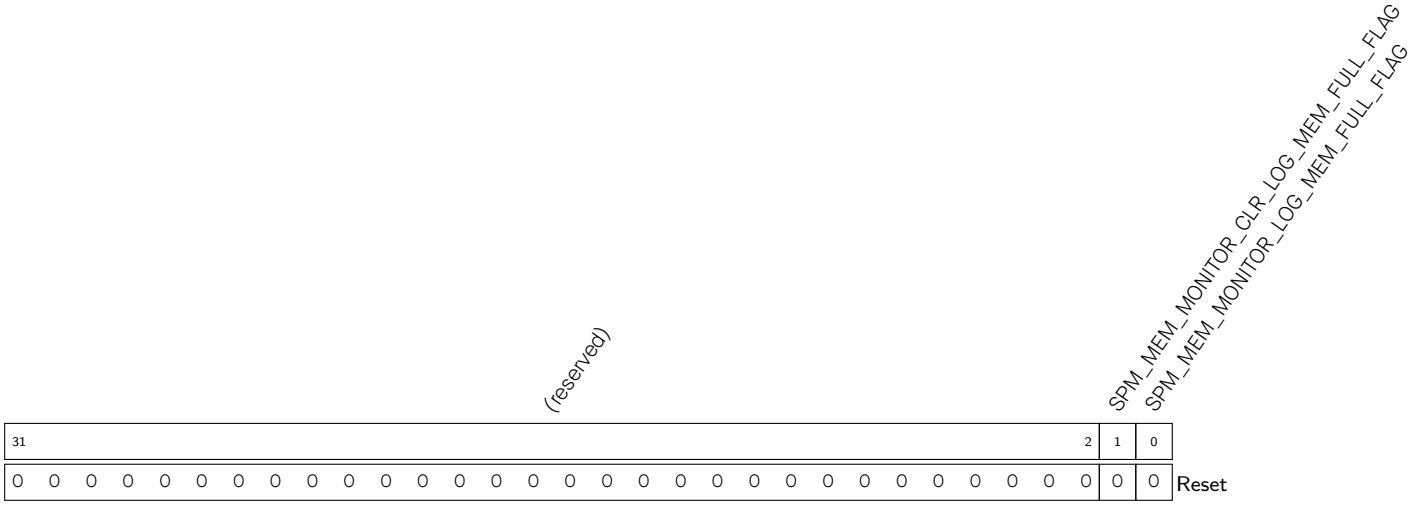
SPM_MEM_MONITOR_LOG_MEM_ADDR_UPDATE 配置是否将 SPM_MEM_MONITOR_LOG_MEM_START_REG 的值更新至 SPM_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG。

1：更新

0：不更新（默认值）

(WT)

Register 22.10. SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0028)



SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG 表示数据是否溢出存储地址范围。

0：未溢出

1：溢出

(RO)

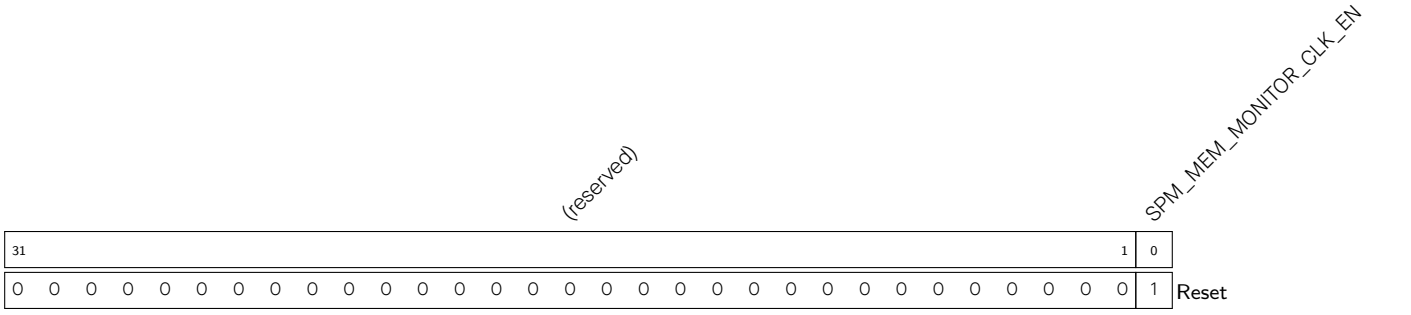
SPM_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 配置是否清除 SPM_MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

0：不清除（默认值）

1：清除

(WT)

Register 22.11. SPMMEM_MONITOR_CLOCK_GATE_REG (0x002C)



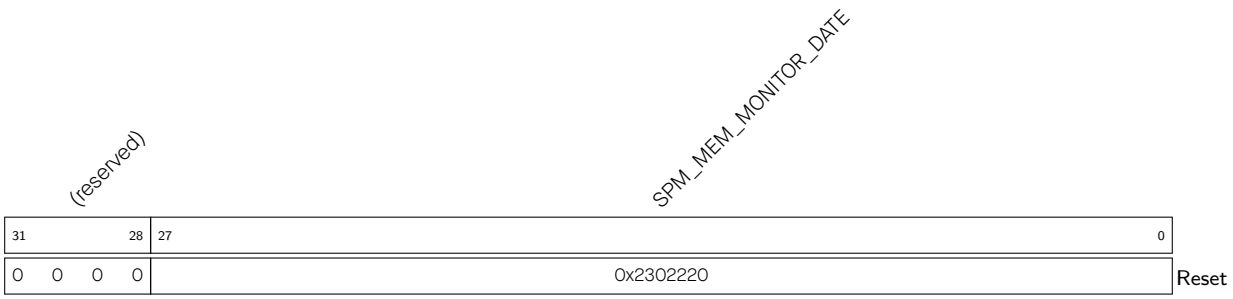
SPM_MEM_MONITOR_CLK_EN 配置是否使能寄存器时钟门控。

0：不使能

1：使能

(R/W)

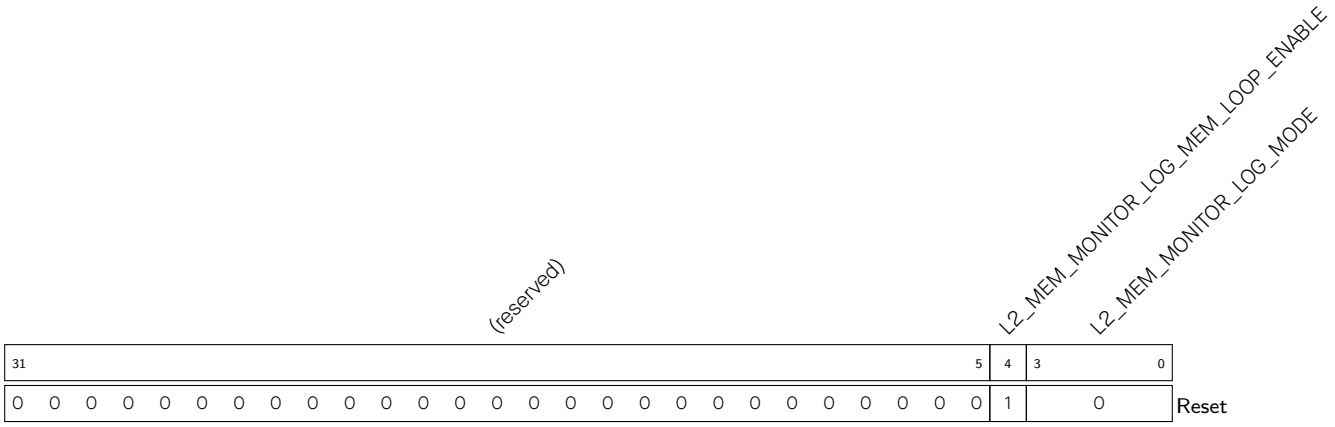
Register 22.12. SPM_MEM_MONITOR_DATE_REG (0x03FC)



SPM_MEM_MONITOR_DATE 版本控制寄存器。(R/W)

22.7.2 DMA 总线记录配置寄存器

Register 22.13. L2_MEM_MONITOR_LOG_SETTING_REG (0x0000)



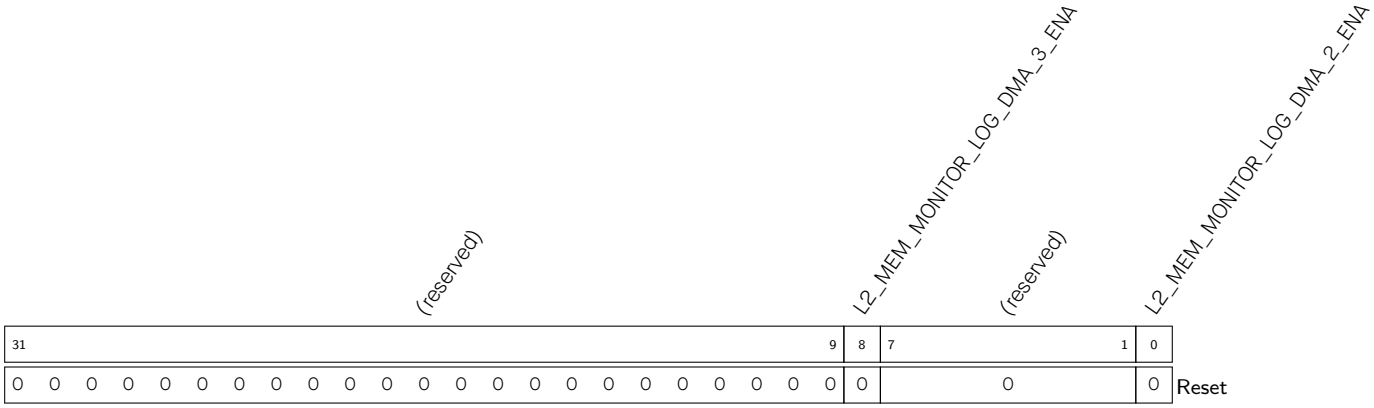
L2_MEM_MONITOR_LOG_MODE 配置监测模式。

bit[0]：写监测
0：不使能
1：使能
bit[1]：word 监测
0：不使能
1：使能
bit[2]：halfword 监测
0：不使能
1：使能
bit[3]：byte 监测
0：不使能
1：使能
(R/W)

L2_MEM_MONITOR_LOG_MEM_LOOP_ENABLE 配置写内存的存储模式。

1：loop 模式
0：非 loop 模式
(R/W)

Register 22.14. L2_MEM_MONITOR_LOG_SETTING1_REG (0x0004)



L2_MEM_MONITOR_LOG_DMA_2_ENA 配置是否使能 DMA_2 通道组总线访问记录。

0：不使能

1：使能

(R/W)

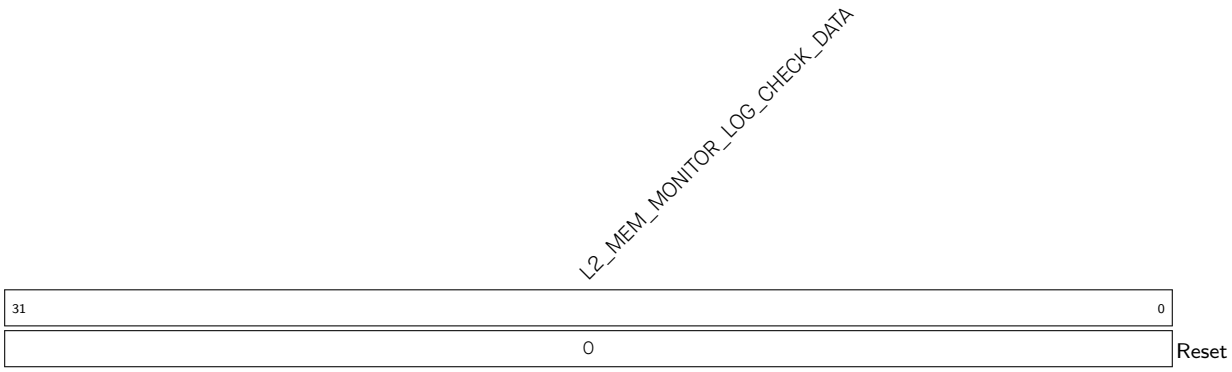
L2_MEM_MONITOR_LOG_DMA_3_ENA 配置是否使能 DMA_3 通道组总线访问记录。

0：不使能

1：使能

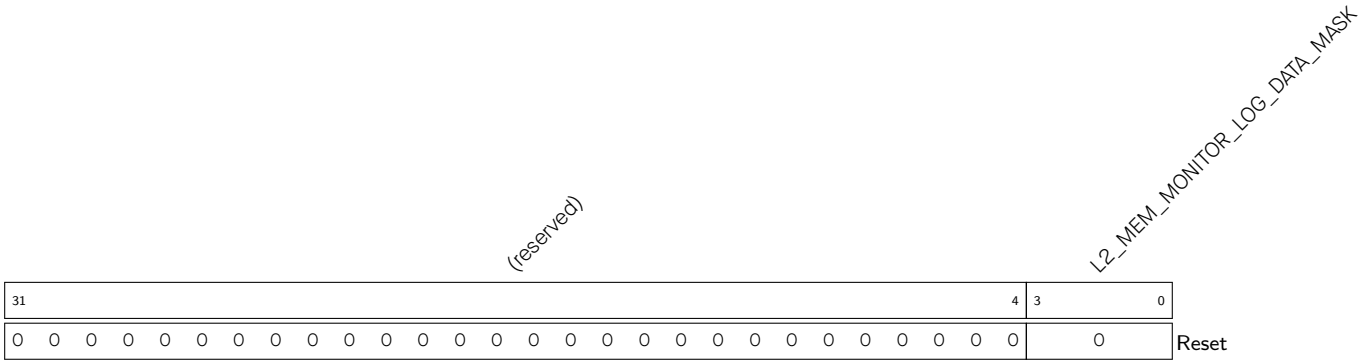
(R/W)

Register 22.15. L2_MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



L2_MEM_MONITOR_LOG_CHECK_DATA 配置总线访问监测的特殊值。(R/W)

Register 22.16. L2_MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



L2_MEM_MONITOR_LOG_DATA_MASK 是否屏蔽 L2_MEM_MONITOR_LOG_CHECK_DATA_REG 的相应字节。

bit[0]：是否屏蔽 L2_MEM_MONITOR_LOG_CHECK_DATA_REG 最低位字节。

0：不屏蔽

1：屏蔽

bit[1]：是否屏蔽 L2_MEM_MONITOR_LOG_CHECK_DATA_REG 第二个低位字节。

0：不屏蔽

1：屏蔽

bit[2]：是否屏蔽 L2_MEM_MONITOR_LOG_CHECK_DATA_REG 第二个高位字节。

0：不屏蔽

1：屏蔽

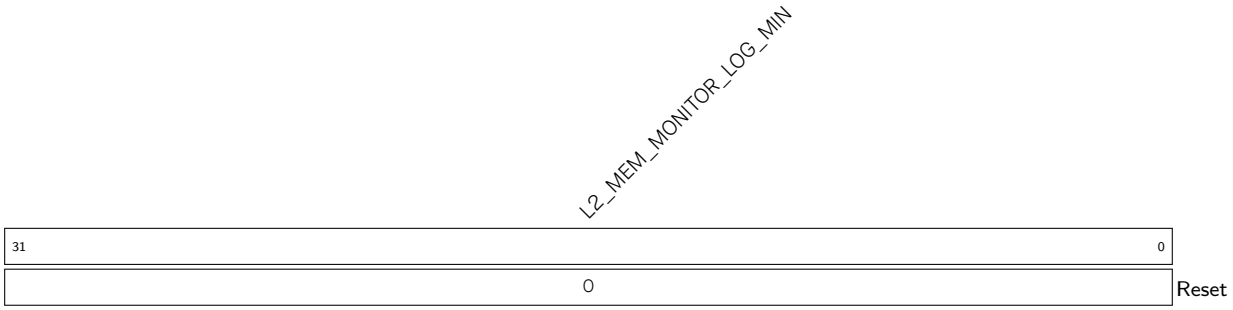
bit[3]：是否屏蔽 L2_MEM_MONITOR_LOG_CHECK_DATA_REG 最高位字节。

0：不屏蔽

1：屏蔽

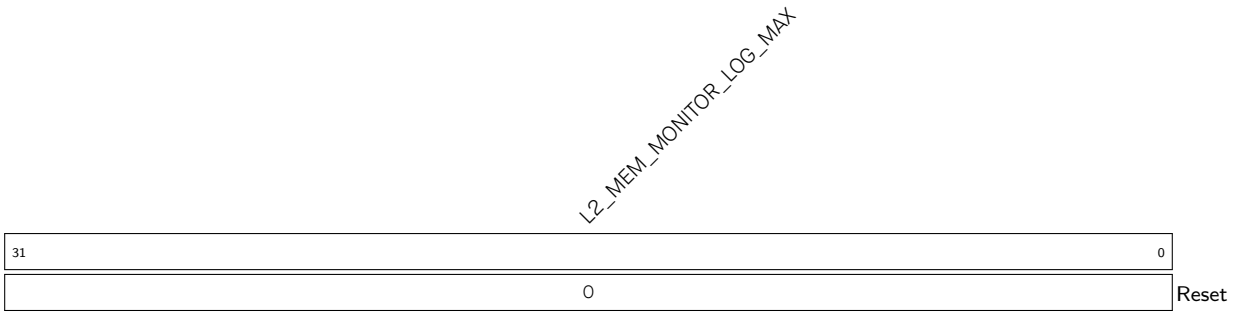
(R/W)

Register 22.17. L2_MEM_MONITOR_LOG_MIN_REG (0x0010)



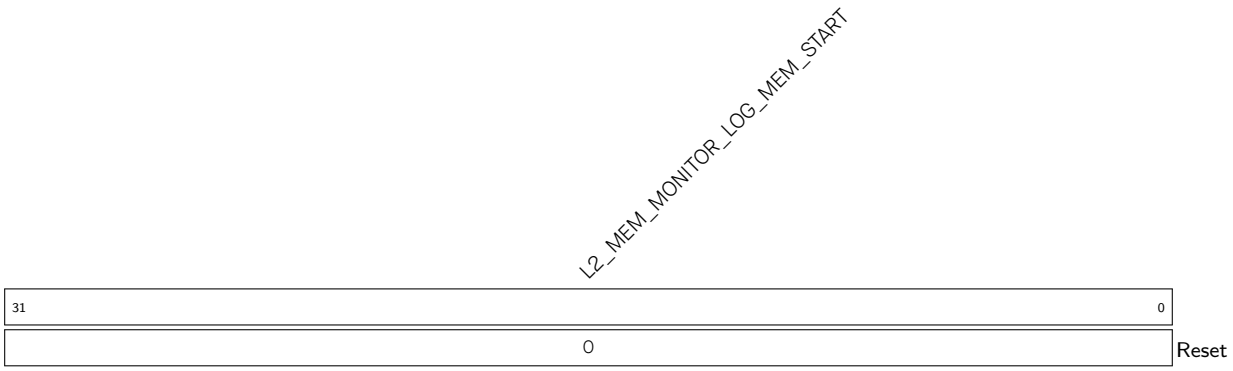
L2_MEM_MONITOR_LOG_MIN 配置监测地址的下边界。(R/W)

Register 22.18. L2_MEM_MONITOR_LOG_MAX_REG (0x0014)



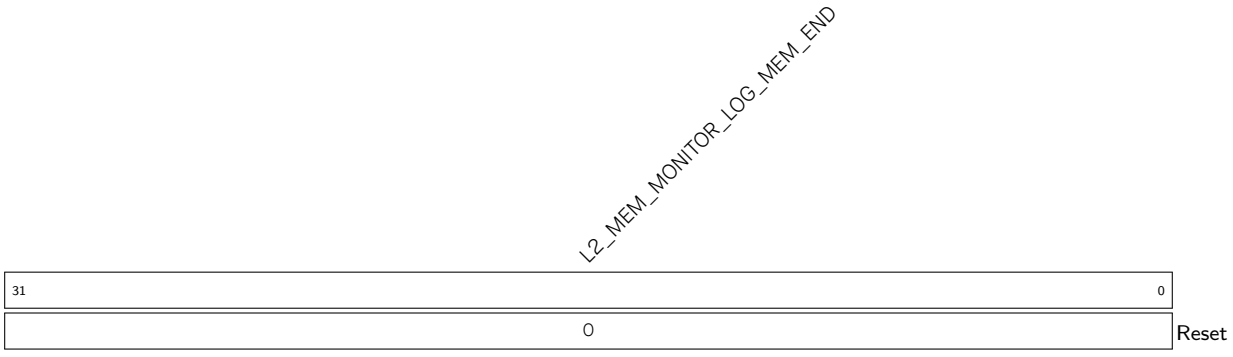
L2_MEM_MONITOR_LOG_MAX 配置监测地址的上边界。(R/W)

Register 22.19. L2_MEM_MONITOR_LOG_MEM_START_REG (0x0018)



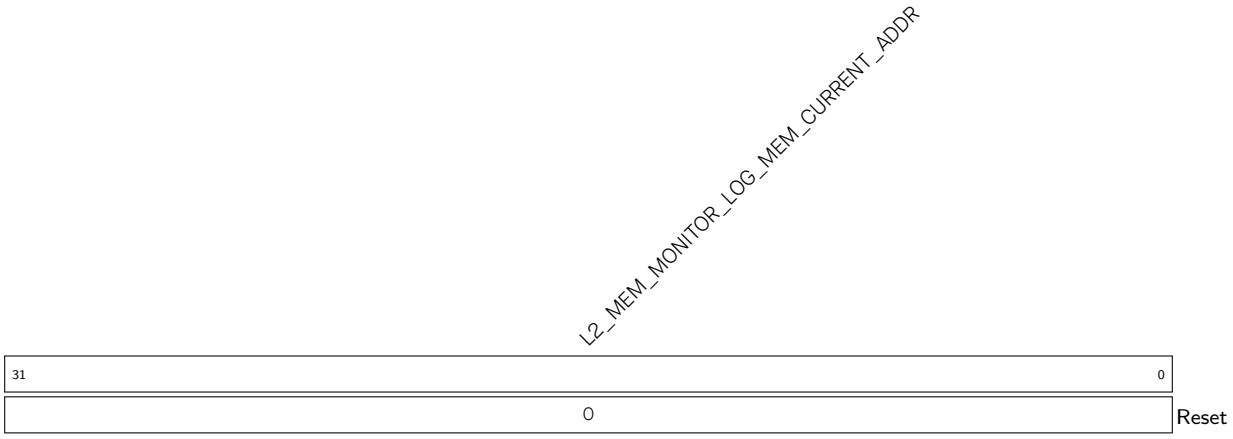
L2_MEM_MONITOR_LOG_MEM_START 配置记录数据写入内存的起始地址。(R/W)

Register 22.20. L2_MEM_MONITOR_LOG_MEM_END_REG (0x001C)



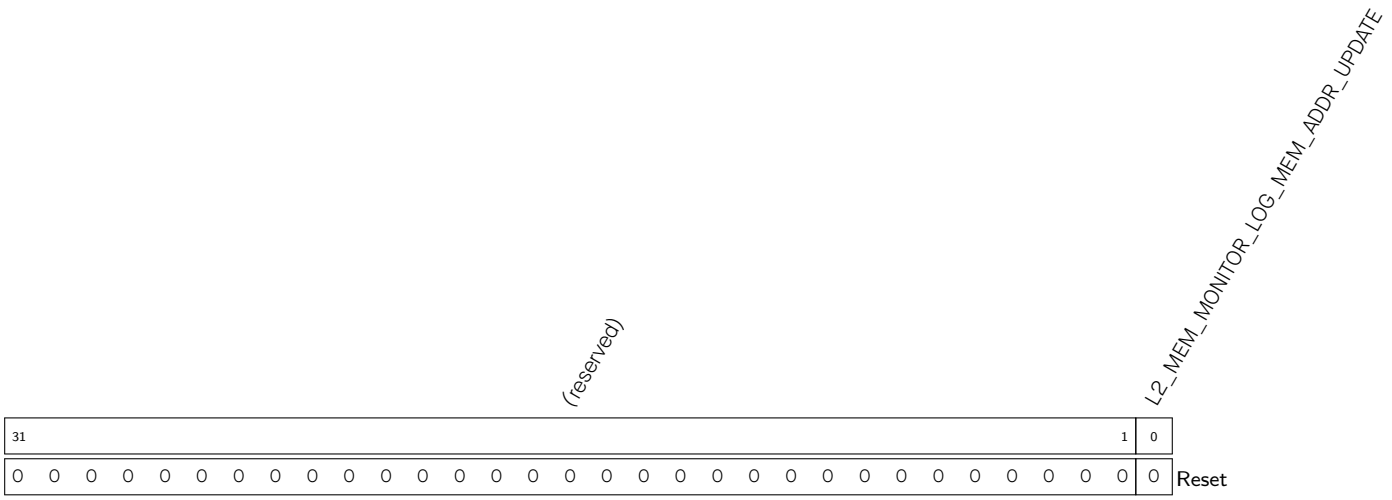
L2_MEM_MONITOR_LOG_MEM_END 配置记录数据写入内存的结束地址。(R/W)

Register 22.21. L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0020)



L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR 表示下一次写内存的写地址。(RO)

Register 22.22. L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0024)



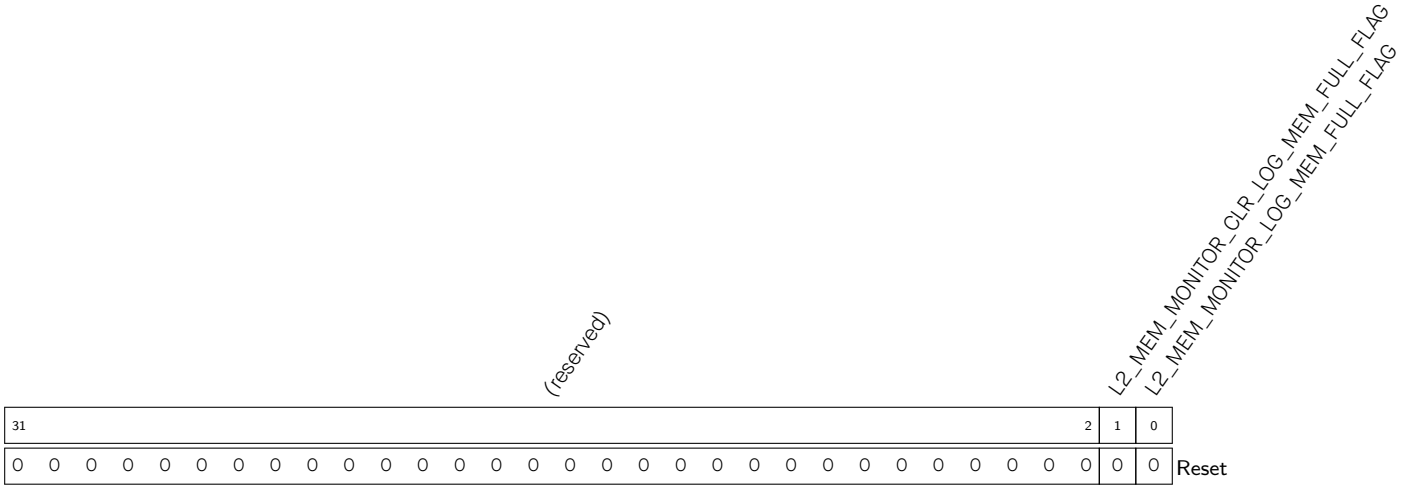
L2_MEM_MONITOR_LOG_MEM_ADDR_UPDATE 配置是否将 L2_MEM_MONITOR_LOG_MEM_START_REG 的值更新至 L2_MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG。

1：更新

0：不更新（默认值）

(WT)

Register 22.23. L2_MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0028)



L2_MEM_MONITOR_LOG_MEM_FULL_FLAG 表示数据是否溢出存储地址范围。

0：未溢出

1：溢出

(RO)

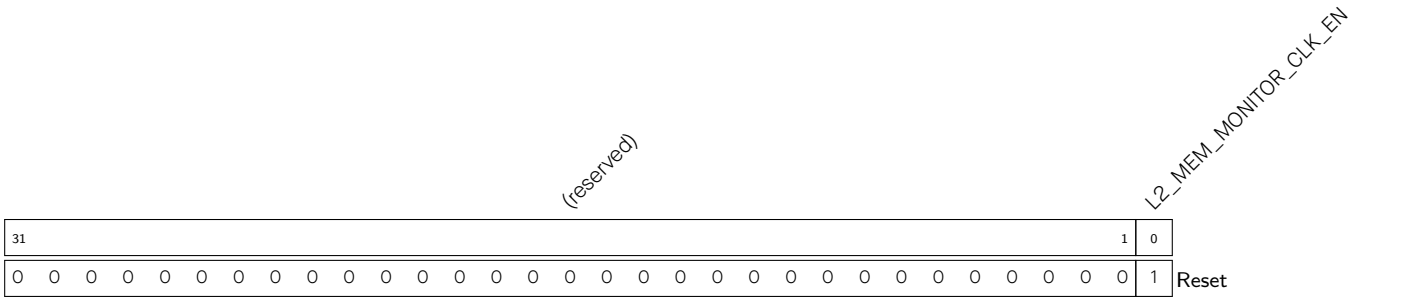
L2_MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 配置是否清除 L2_MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

0：不清除（默认值）

1：清除

(WT)

Register 22.24. L2_MEM_MONITOR_CLOCK_GATE_REG (0x002C)



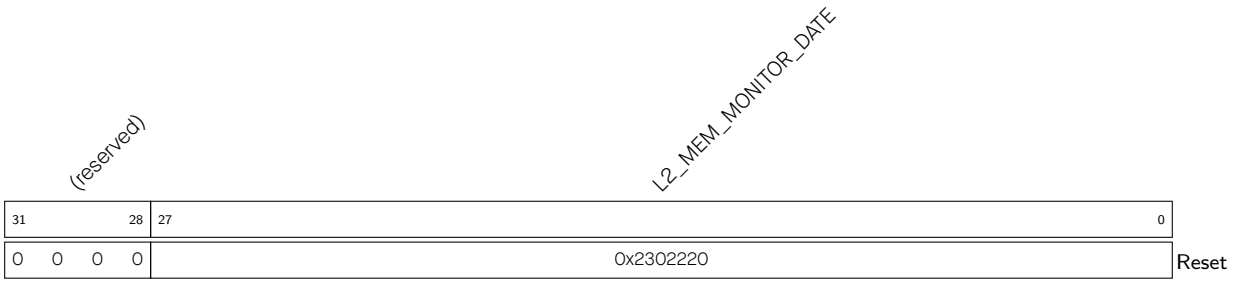
L2_MEM_MONITOR_CLK_EN 配置是否使能寄存器时钟门控。

0：不使能

1：使能

(R/W)

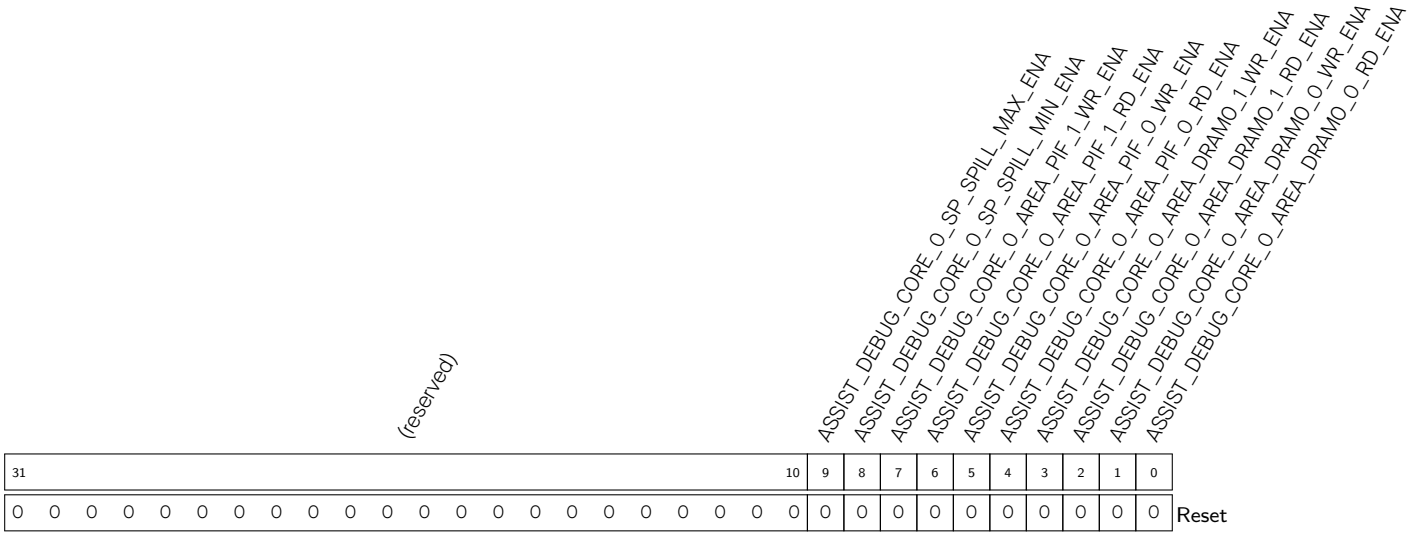
Register 22.25. L2_MEM_MONITOR_DATE_REG (0x03FC)



L2_MEM_MONITOR_DATE 版本控制寄存器。(R/W)

22.7.3 其它寄存器

Register 22.26. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_ENA 配置是否监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 0 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_ENA 配置是否监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 0 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_ENA 配置是否监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 1 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_ENA 配置是否监测 HP CPU0 数据总线在 HP CPU0 数据总线区域 1 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_ENA 配置是否监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 0 内读操作。

0：不监测

1：监测

(R/W)

见下页……

Register 22.26. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)

接上页……

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_ENA 配置是否监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 0 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_ENA 配置是否监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 1 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_ENA 配置是否监测 HP CPU0 外设总线在 HP CPU0 外设总线区域 1 内写操作。

0：不监测 1：监测 (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_ENA 配置是否监测 HP CPU0 栈指针小于 HP CPU0 栈指针监测区域的下边界。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_ENA 配置是否监测 HP CPU0 栈指针大于 HP CPU0 栈指针监测区域的上边界。

0：不监测

1：监测

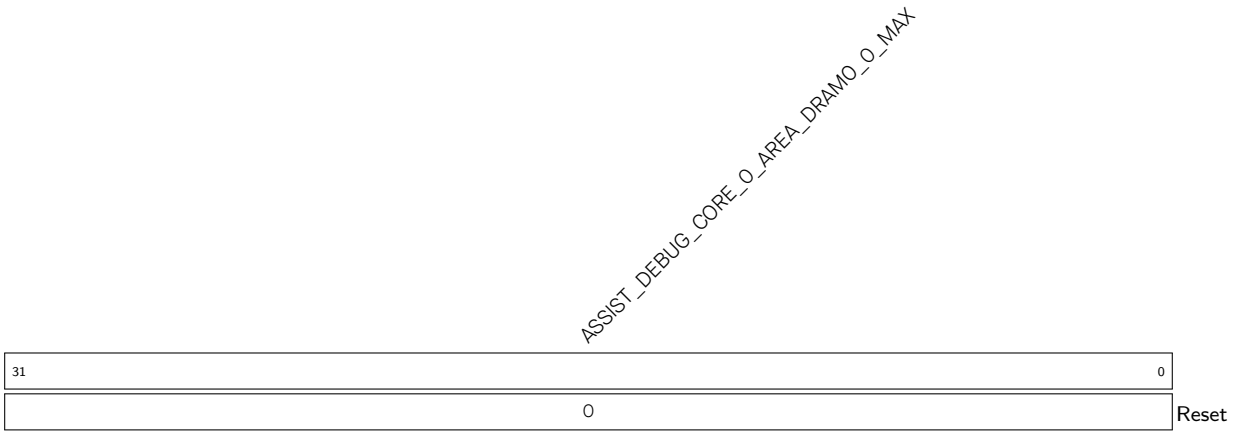
(R/W)

Register 22.27. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG (0x0010)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN 配置 HP CPU0 数据总线区域 0 的下边界。(R/W)

Register 22.28. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG (0x0014)



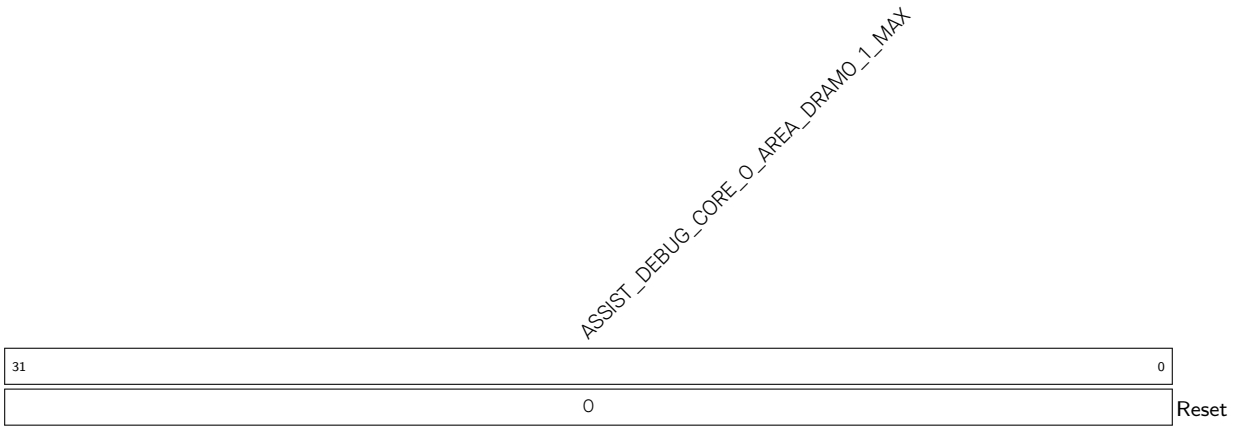
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX 配置 HP CPU0 数据总线区域 0 的上边界。(R/W)

Register 22.29. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG (0x0018)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN 配置 HP CPU0 数据总线区域 1 的下边界。(R/W)

Register 22.30. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG (0x001C)



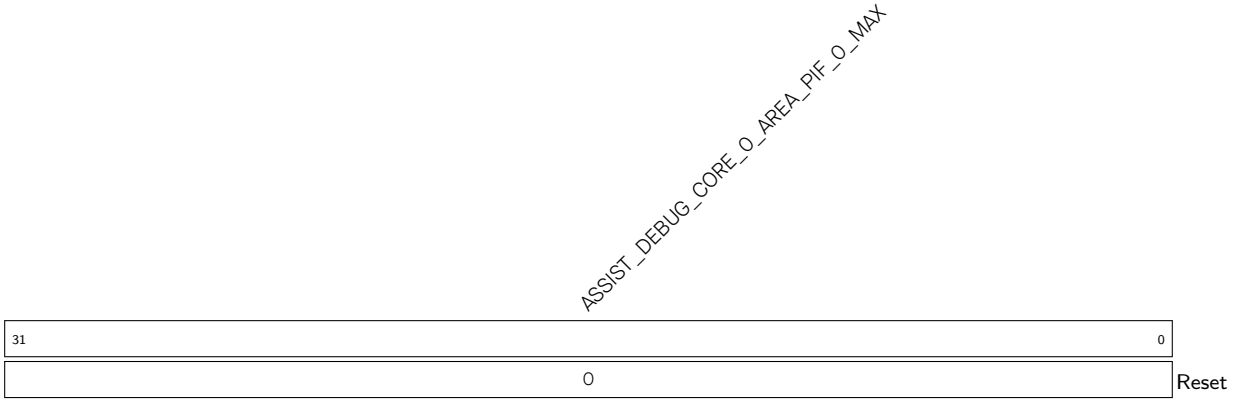
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX 配置 HP CPU0 数据总线区域 1 的上边界。(R/W)

Register 22.31. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG (0x0020)



ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN 配置 HP CPU0 外设总线区域 0 的下边界。(R/W)

Register 22.32. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG (0x0024)



ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX 配置 HP CPU0 外设总线区域 0 的上边界。(R/W)

Register 22.33. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG (0x0028)



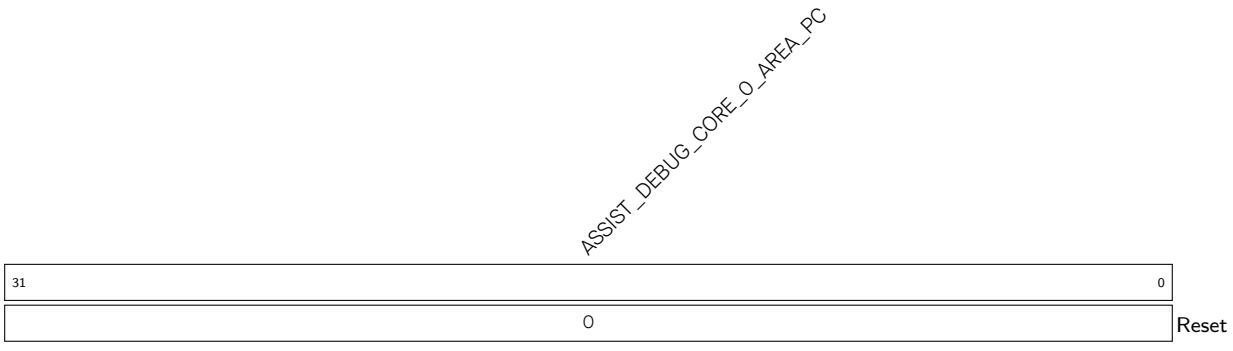
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN 配置 HP CPU0 外设总线区域 1 的下边界。(R/W)

Register 22.34. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG (0x002C)



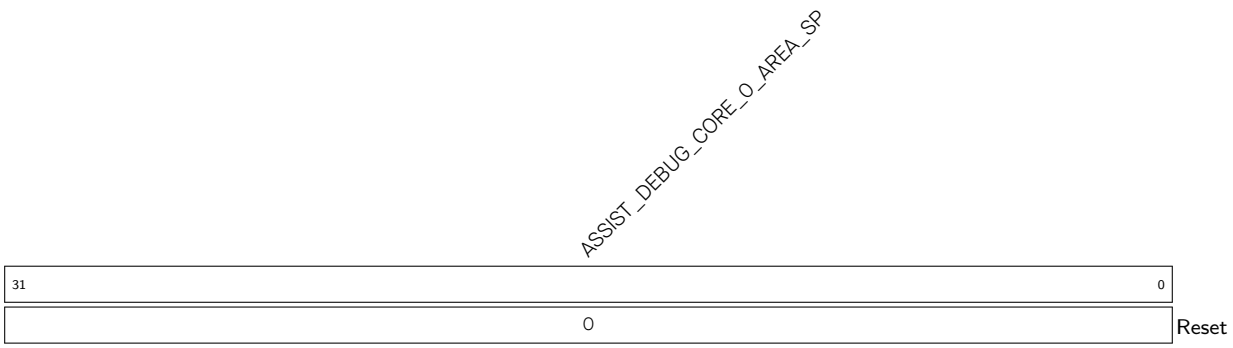
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX 配置 HP CPU0 外设总线区域 1 的上边界。(R/W)

Register 22.35. ASSIST_DEBUG_CORE_0_AREA_PC_REG (0x0030)



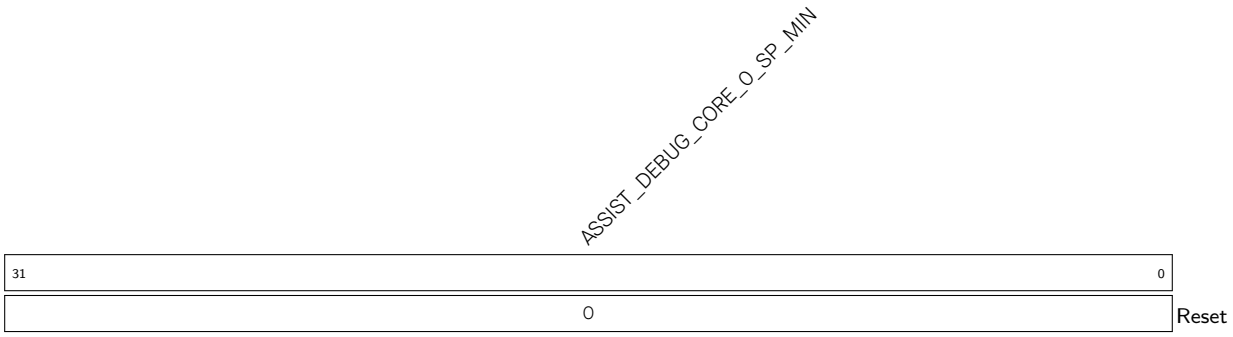
ASSIST_DEBUG_CORE_0_AREA_PC 表示区域监测下触发中断时刻的 HP CPU0 PC 值。(RO)

Register 22.36. ASSIST_DEBUG_CORE_0_AREA_SP_REG (0x0034)



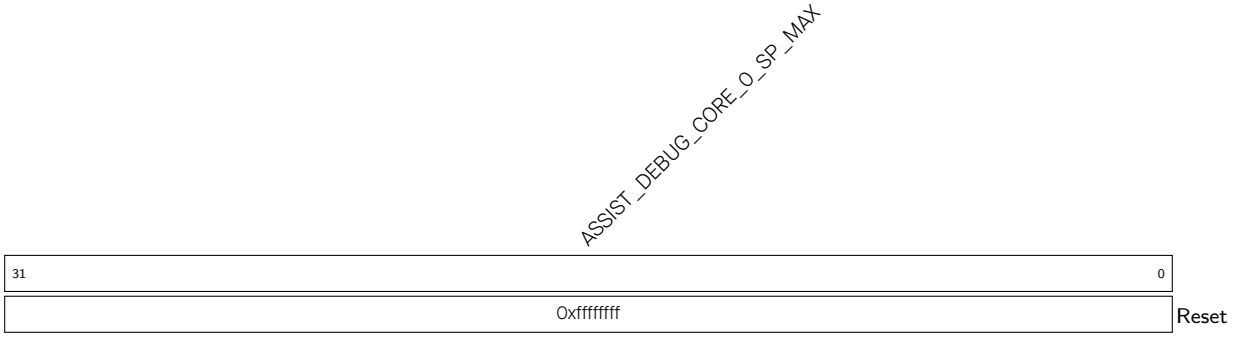
ASSIST_DEBUG_CORE_0_AREA_SP 表示区域监测下触发中断时刻的 HP CPU0 SP 值。(RO)

Register 22.37. ASSIST_DEBUG_CORE_0_SP_MIN_REG (0x0038)



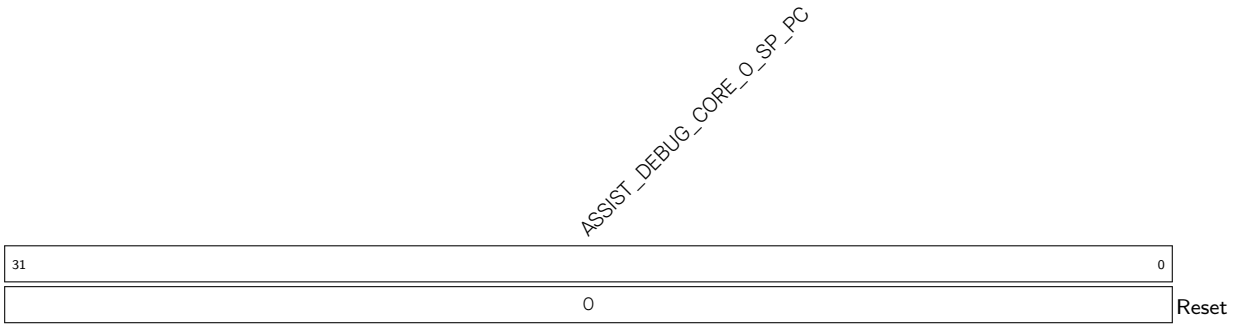
ASSIST_DEBUG_CORE_0_SP_MIN 配置 HP CPU0 栈指针监测区域的下边界。(R/W)

Register 22.38. ASSIST_DEBUG_CORE_0_SP_MAX_REG (0x003C)



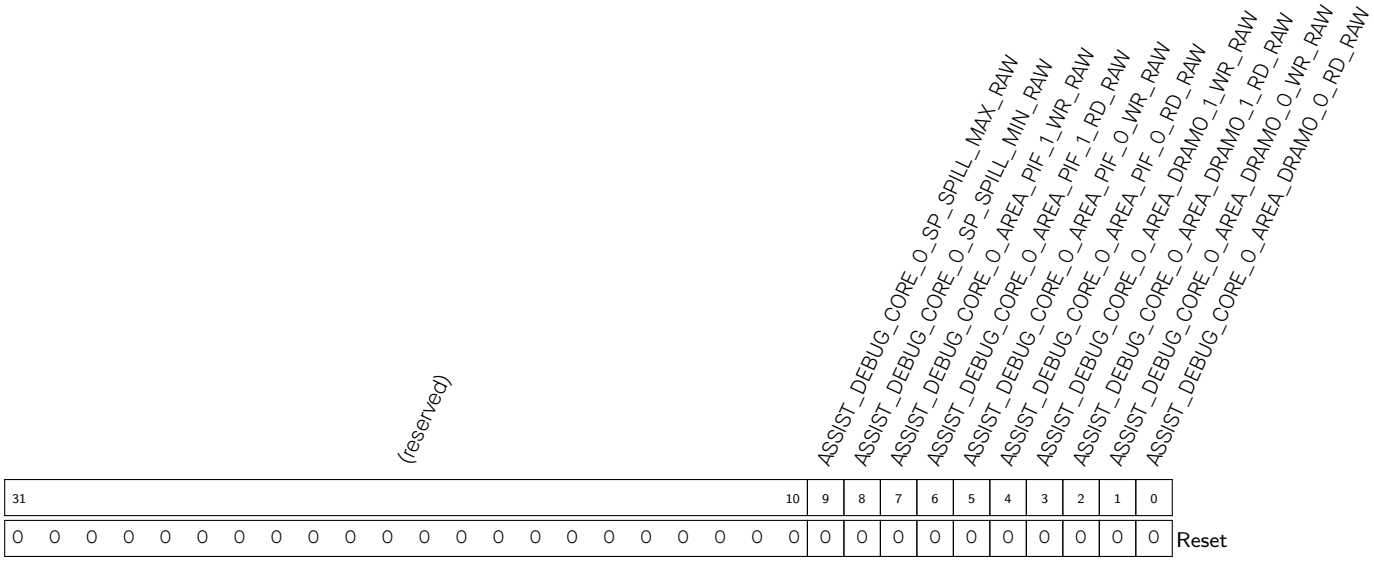
ASSIST_DEBUG_CORE_0_SP_MAX 配置 HP CPU0 栈指针监测区域的上边界。(R/W)

Register 22.39. ASSIST_DEBUG_CORE_0_SP_PC_REG (0x0040)



ASSIST_DEBUG_CORE_0_SP_PC 表示栈指针监测的 HP CPU0 PC 值。(RO)

Register 22.40. ASSIST_DEBUG_CORE_0_INTR_RAW_REG (0x0004)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_RAW ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_RAW ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_RAW ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_RAW ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_RAW ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_RAW ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT 的原始中断状态。(RO)

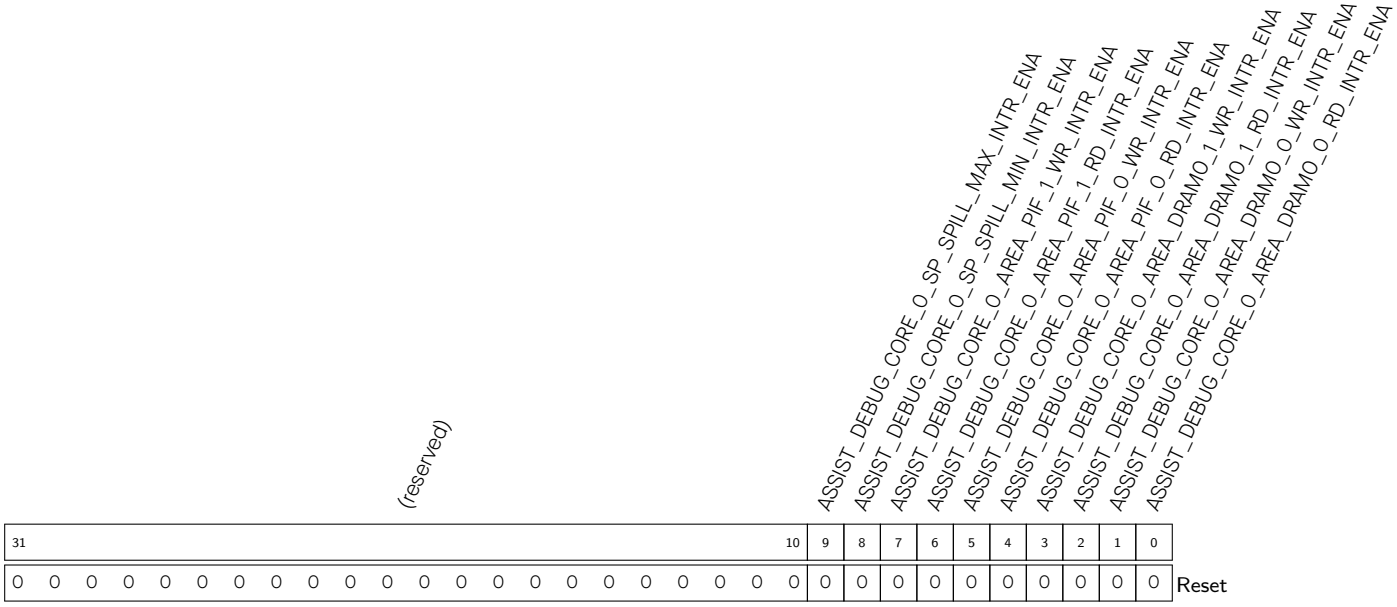
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_RAW ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_RAW ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_RAW ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_RAW ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT 的原始中断状态。(RO)

Register 22.41. ASSIST_DEBUG_CORE_0_INTR_ENA_REG (0x0008)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INT。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INT。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INT。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INT。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT。(R/W)

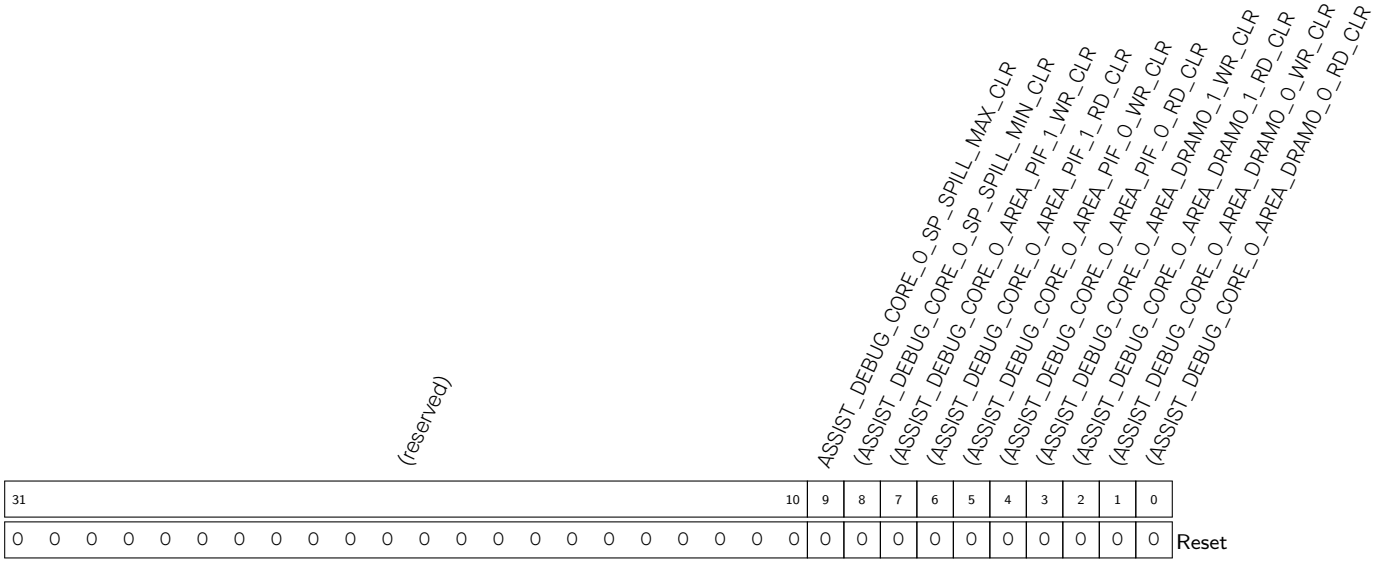
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INT。(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT。(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INTR_ENA 写 1 使能 ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT。(R/W)

Register 22.42. ASSIST_DEBUG_CORE_0_INTR_CLR_REG (0x000C)



ASSIST_DEBUG_CORE_0_AREA_DRAMO_0_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_DRAMO_0_RD_INT。(WT)

ASSIST_DEBUG_CORE_0_AREA_DRAMO_0_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_DRAMO_0_WR_INT。(WT)

ASSIST_DEBUG_CORE_0_AREA_DRAMO_1_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_DRAMO_1_RD_INT。(WT)

ASSIST_DEBUG_CORE_0_AREA_DRAMO_1_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_DRAMO_1_WR_INT。(WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INT。(WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT。(WT)

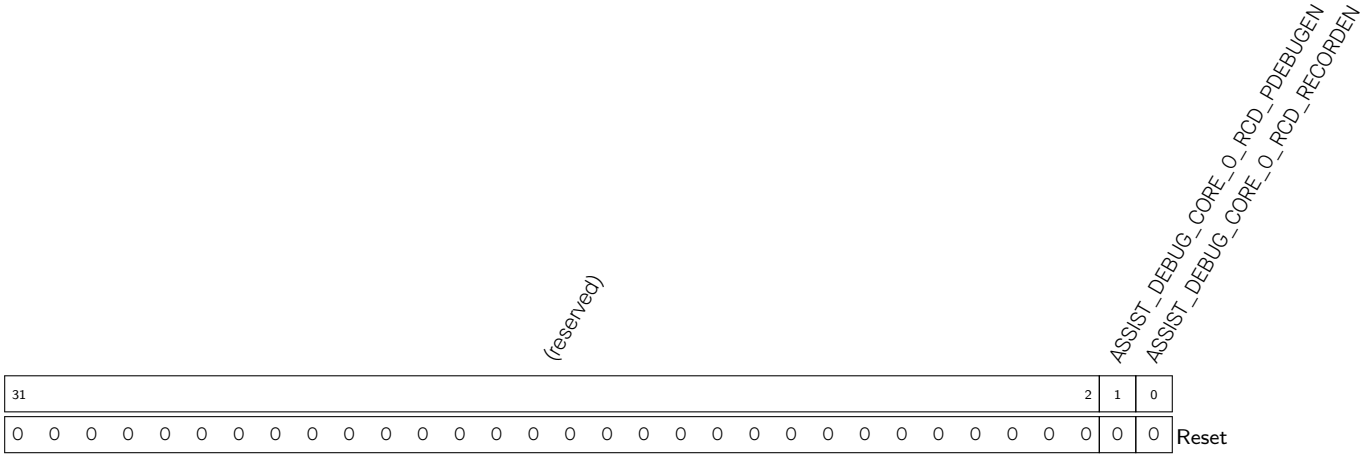
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INT。(WT)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INT。(WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INT。(WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_CLR 写 1 清除 ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INT。(WT)

Register 22.43. ASSIST_DEBUG_CORE_0_RCD_EN_REG (0x0044)



ASSIST_DEBUG_CORE_0_RCD_RECORDEN 配置是否使能 HP CPU0 PC 记录。

0：不使能

1：ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 开始实时记录 HP CPU0 PC

(R/W)

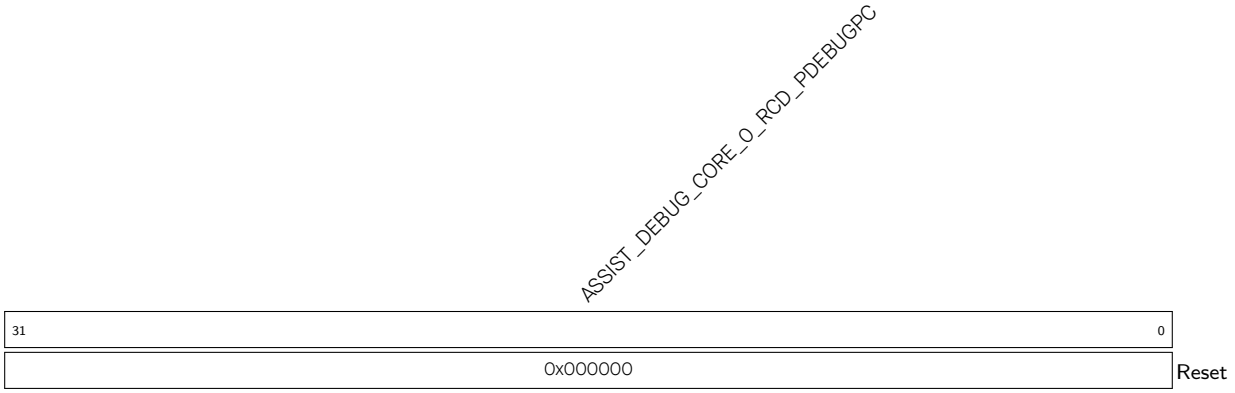
ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 配置是否使能 HP CPU0 调试。

0：不使能

1：HP CPU0 输出 HP CPU0 PC

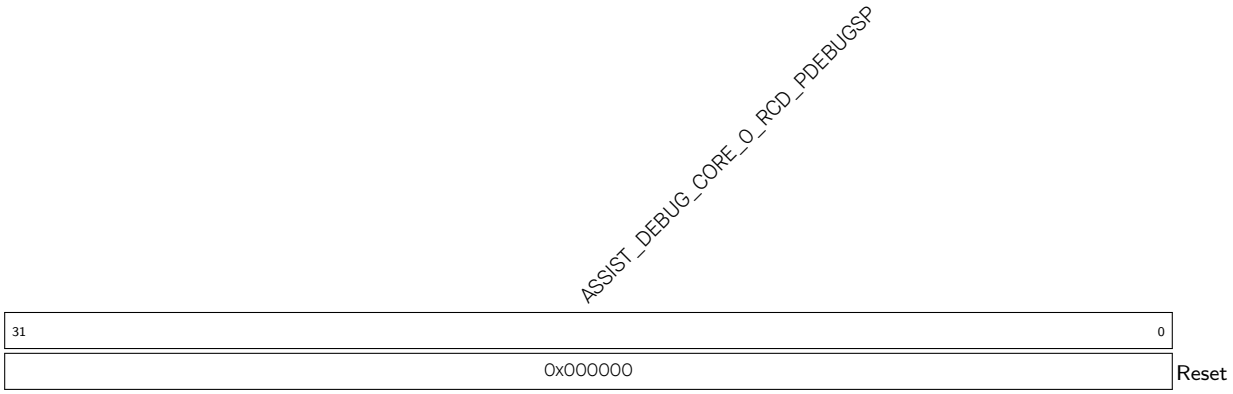
(R/W)

Register 22.44. ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG (0x0048)



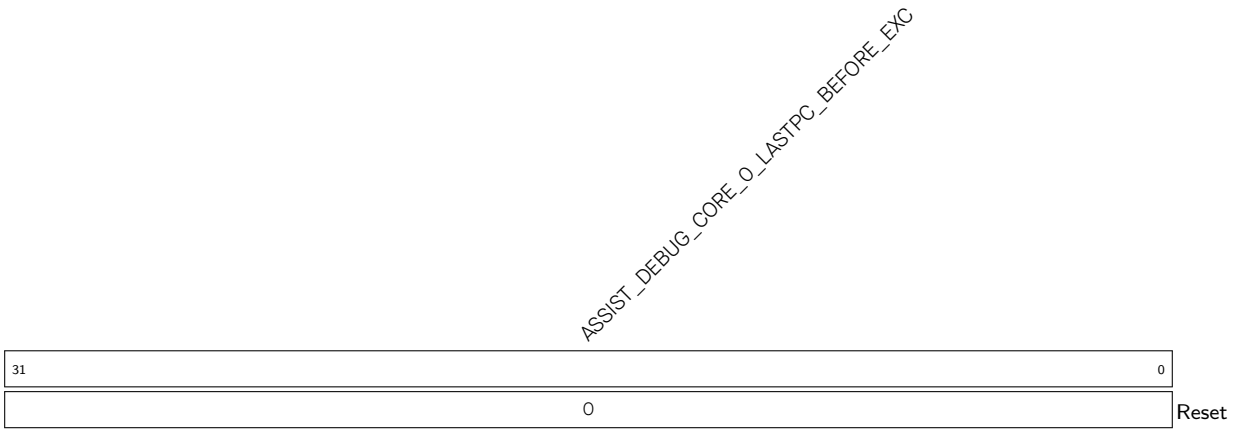
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC 表示 HP CPU0 复位时刻的 PC 值。(RO)

Register 22.45. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG (0x004C)



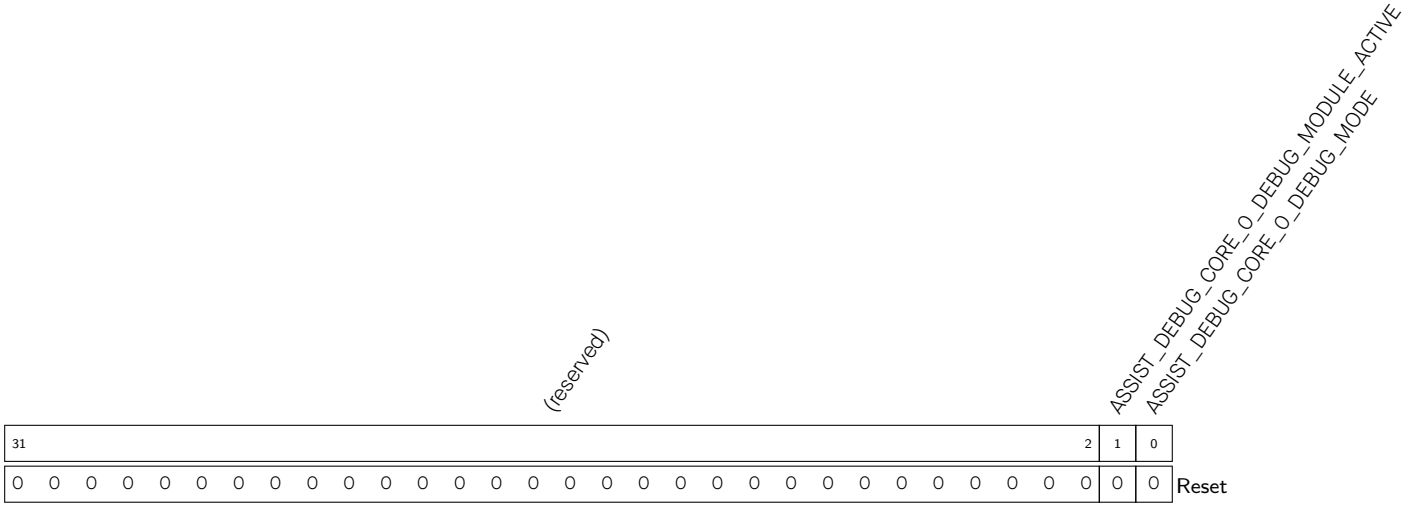
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP 表示 HP CPU0 SP。(RO)

Register 22.46. ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)



ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXC 表示 HP CPU0 异常前的最后一条指令的 HP CPU0 PC。(RO)

Register 22.47. ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG (0x0074)



ASSIST_DEBUG_CORE_0_DEBUG_MODE 表示 HP CPU0 是否处于调试模式。

1：处于调试模式

0：未处于调试模式

(RO)

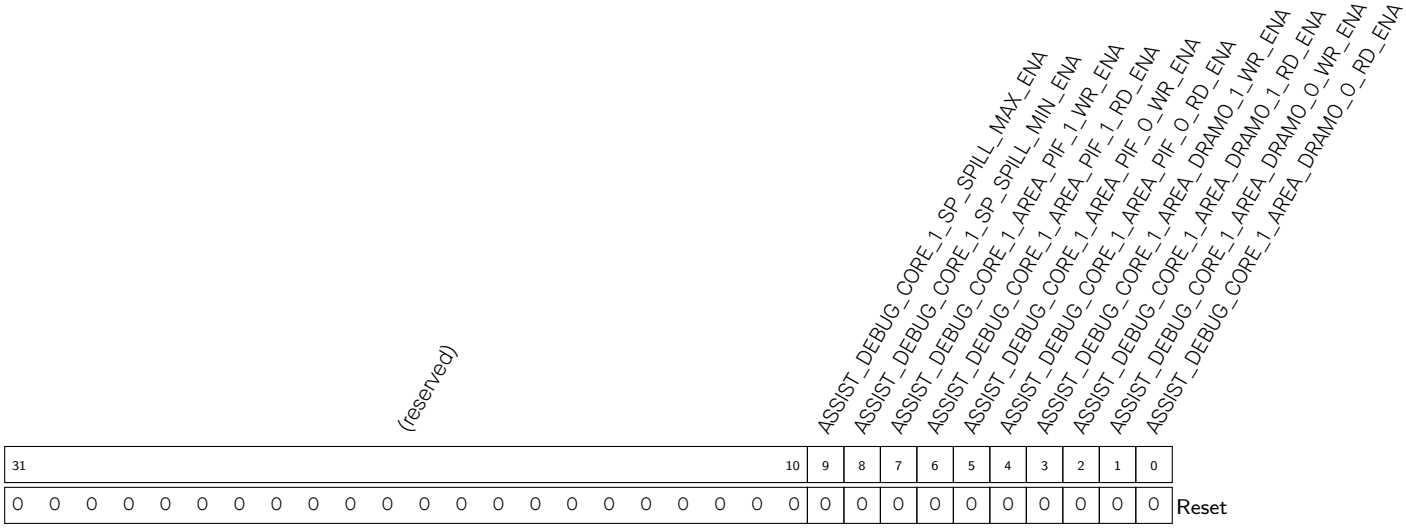
ASSIST_DEBUG_CORE_0_DEBUG_MODULE_ACTIVE 表示 HP CPU0 调试模块的工作状态。

1：处于工作状态

其它：未处于工作状态

(RO)

Register 22.48. ASSIST_DEBUG_CORE_1_MONTR_ENA_REG (0x0080)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_ENA 配置是否监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 0 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_ENA 配置是否监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 0 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_ENA 配置是否监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 1 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_ENA 配置是否监测 HP CPU1 数据总线在 HP CPU1 数据总线区域 1 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_ENA 配置是否监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 0 内读操作。

0：不监测

1：监测

(R/W)

见下页……

Register 22.48. ASSIST_DEBUG_CORE_1_MONTR_ENA_REG (0x0000)

接上页……

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_ENA 配置是否监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 0 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_ENA 配置是否监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 1 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_ENA 配置是否监测 HP CPU1 外设总线在 HP CPU1 外设总线区域 1 内写操作。

0：不监测 1：监测 (R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_ENA 配置是否监测 HP CPU1 栈指针小于 HP CPU1 栈指针监测区域的下边界。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_ENA 配置是否监测 HP CPU1 栈指针大于 HP CPU1 栈指针监测区域的上边界。

0：不监测

1：监测

(R/W)

Register 22.49. ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN_REG (0x0090)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MIN 配置 HP CPU1 数据总线区域 0 的下边界。(R/W)

Register 22.50. ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX_REG (0x0094)



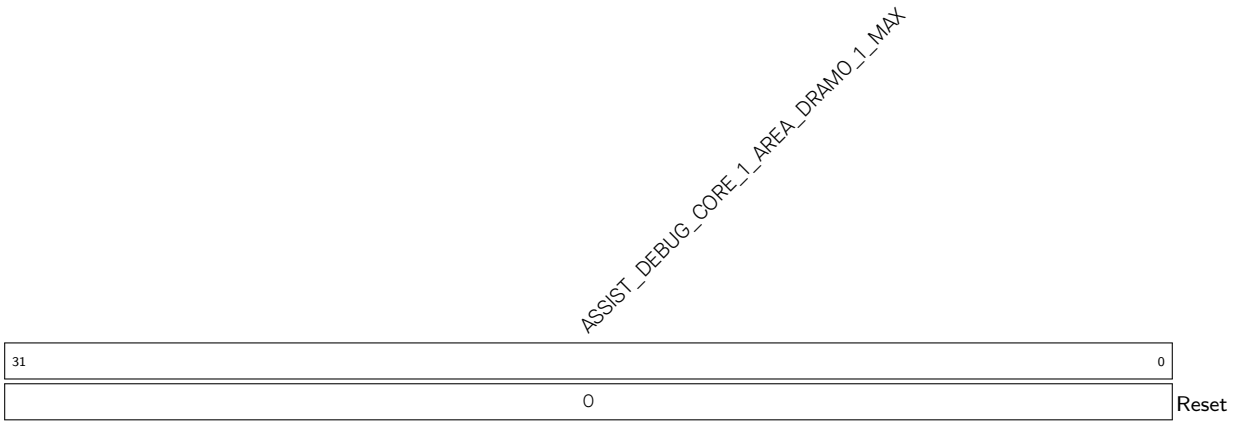
ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_MAX 配置 HP CPU1 数据总线区域 0 的上边界。(R/W)

Register 22.51. ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN_REG (0x0098)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MIN 配置 HP CPU1 数据总线区域 1 的下边界。(R/W)

Register 22.52. ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX_REG (0x009C)



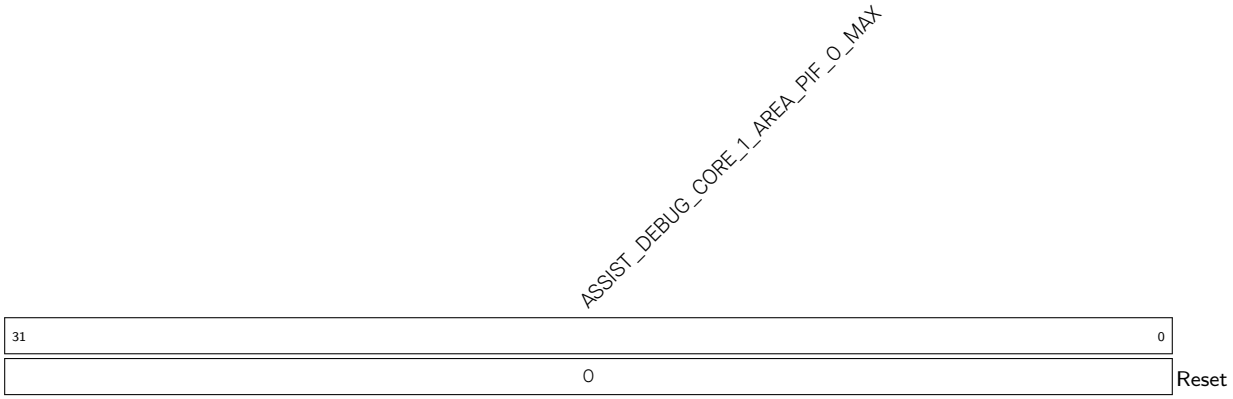
ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_MAX 配置 HP CPU1 数据总线区域 1 的上边界。(R/W)

Register 22.53. ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN_REG (0x00A0)



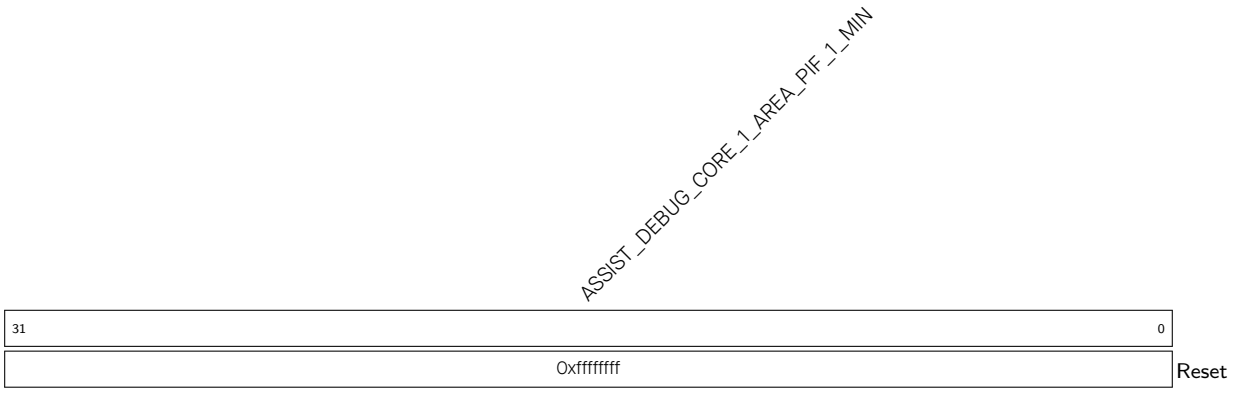
ASSIST_DEBUG_CORE_1_AREA_PIF_0_MIN 配置 HP CPU1 外设总线区域 0 的下边界。(R/W)

Register 22.54. ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX_REG (0x00A4)



ASSIST_DEBUG_CORE_1_AREA_PIF_0_MAX 配置 HP CPU1 外设总线区域 0 的上边界。(R/W)

Register 22.55. ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN_REG (0x00A8)



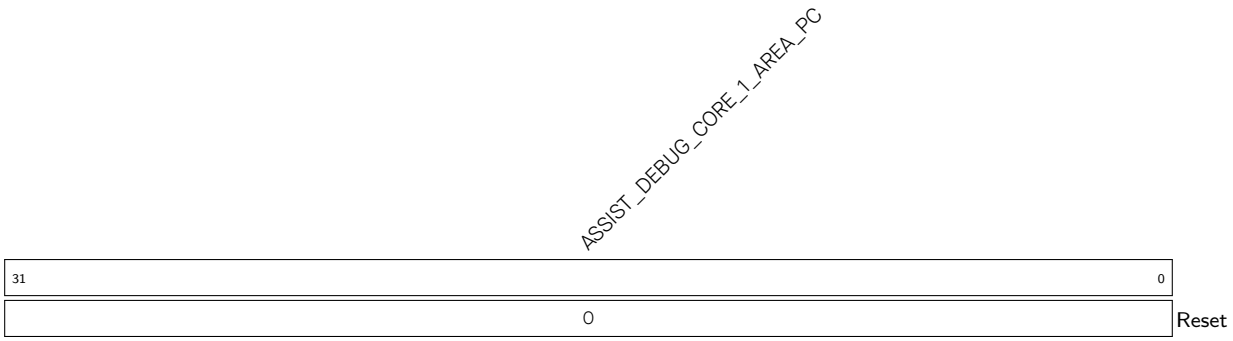
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MIN 配置 HP CPU1 外设总线区域 1 的下边界。(R/W)

Register 22.56. ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX_REG (0x00AC)



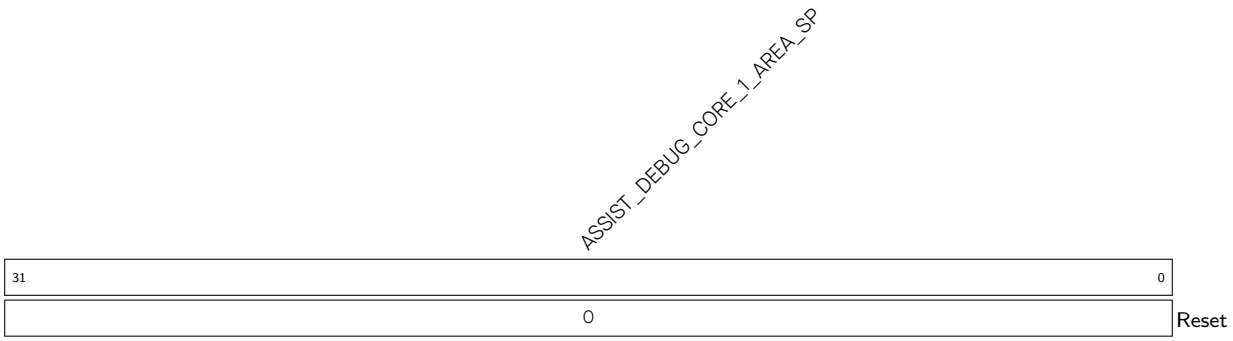
ASSIST_DEBUG_CORE_1_AREA_PIF_1_MAX 配置 HP CPU1 外设总线区域 1 的上边界。(R/W)

Register 22.57. ASSIST_DEBUG_CORE_1_AREA_PC_REG (0x00B0)



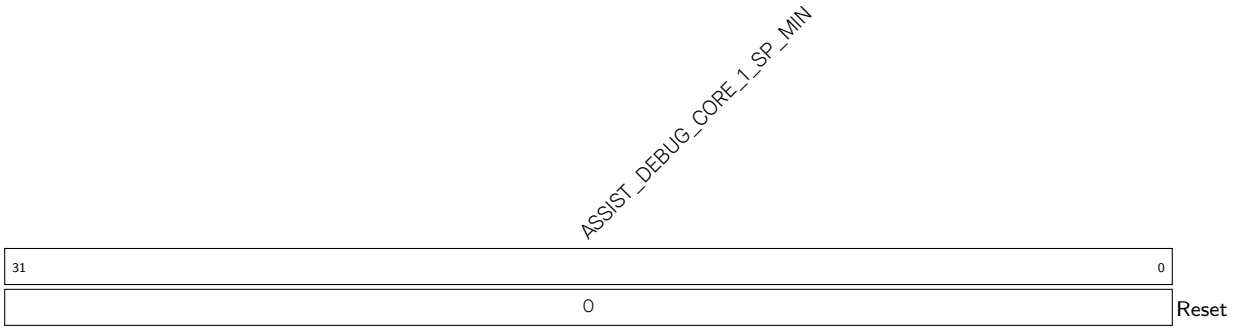
ASSIST_DEBUG_CORE_1_AREA_PC 表示区域监测下触发中断时刻的 HP CPU1 PC 值。(RO)

Register 22.58. ASSIST_DEBUG_CORE_1_AREA_SP_REG (0x00B4)



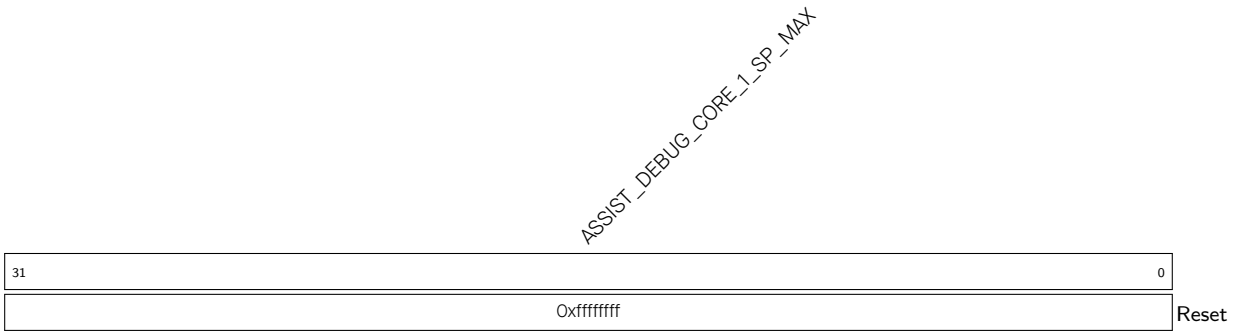
ASSIST_DEBUG_CORE_1_AREA_SP 表示区域监测下触发中断时刻的 HP CPU1 SP 值。(RO)

Register 22.59. ASSIST_DEBUG_CORE_1_SP_MIN_REG (0x00B8)



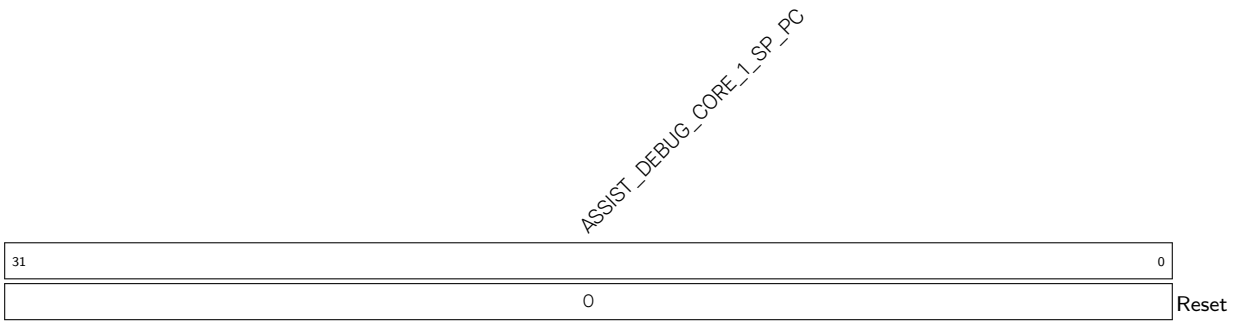
ASSIST_DEBUG_CORE_1_SP_MIN 配置 HP CPU1 栈指针监测区域的下边界。(R/W)

Register 22.60. ASSIST_DEBUG_CORE_1_SP_MAX_REG (0x00BC)



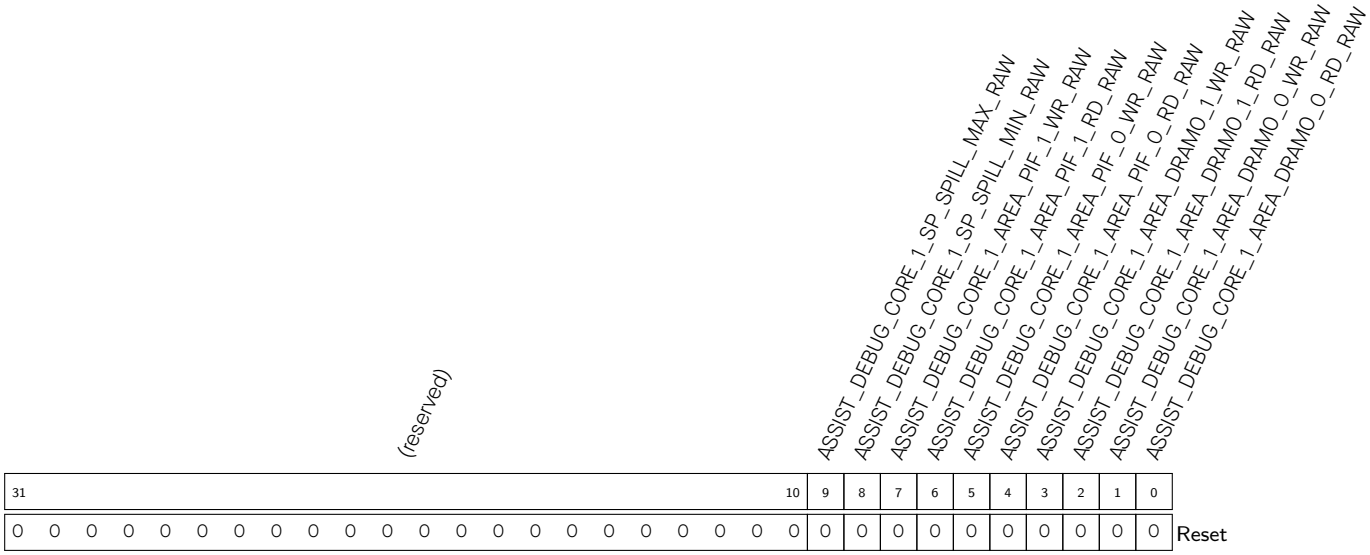
ASSIST_DEBUG_CORE_1_SP_MAX 配置 HP CPU1 栈指针监测区域的上边界。(R/W)

Register 22.61. ASSIST_DEBUG_CORE_1_SP_PC_REG (0x00C0)



ASSIST_DEBUG_CORE_1_SP_PC 表示栈指针监测的 HP CPU1 PC 值。(RO)

Register 22.62. ASSIST_DEBUG_CORE_1_INTR_RAW_REG (0x0084)



ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_RAW ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_RAW ASSIST_DEBUG_CORE_1_AREA_DRAM0_0_WR_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_RAW ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_RAW ASSIST_DEBUG_CORE_1_AREA_DRAM0_1_WR_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_RAW ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_RAW ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT 的原始中断状态。(RO)

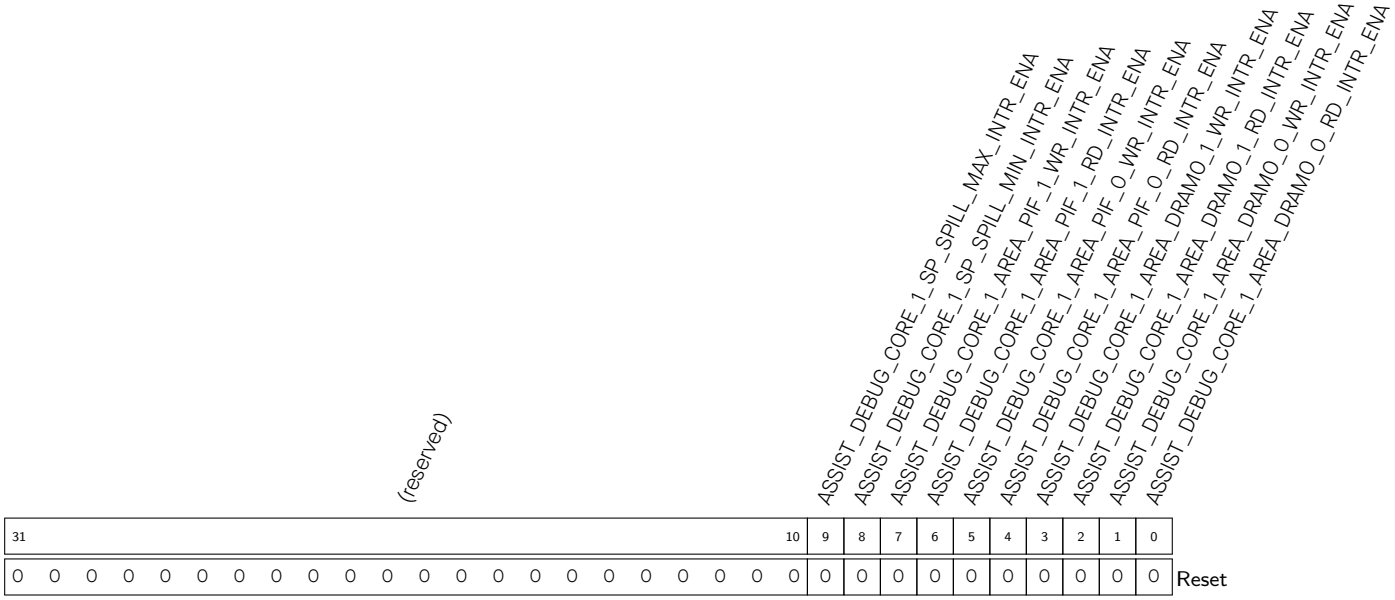
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_RAW ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_RAW ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_RAW ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT 的原始中断状态。(RO)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_RAW ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT 的原始中断状态。(RO)

Register 22.63. ASSIST_DEBUG_CORE_1_INTR_ENA_REG (0x0088)



ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_RD_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_DRAMO_0_RD_INT。(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_WR_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_DRAMO_0_WR_INT。(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_RD_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_DRAMO_1_RD_INT。(R/W)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_WR_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_DRAMO_1_WR_INT。(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT。(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT。(R/W)

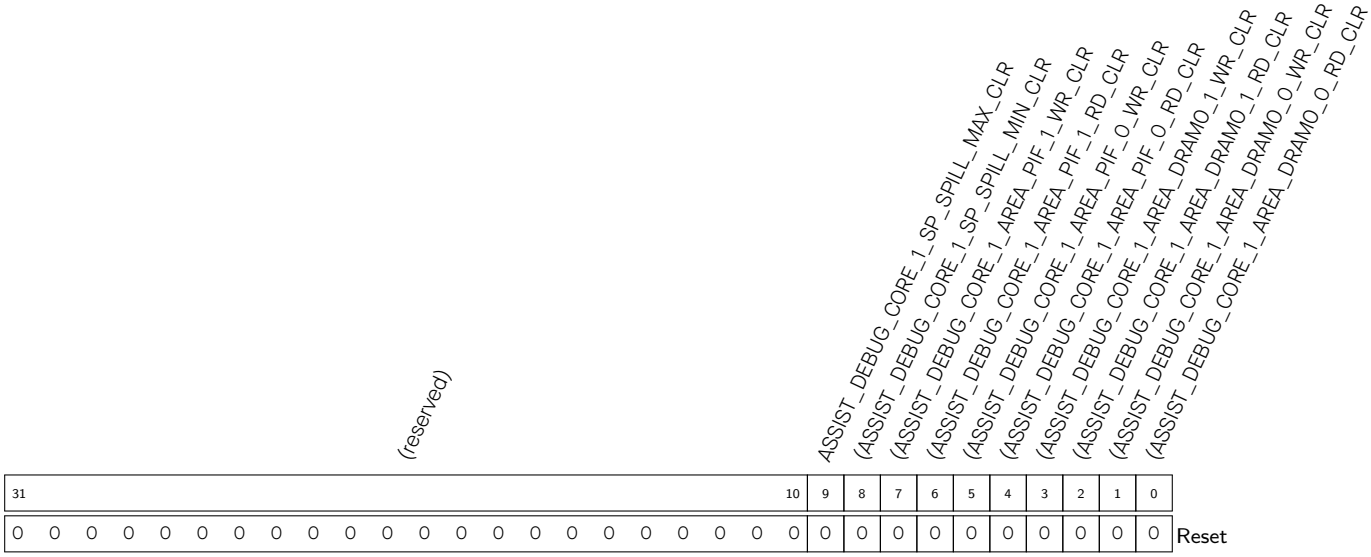
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT。(R/W)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT。(R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_SP_SPILL_MIN_INT。(R/W)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INTR_ENA 写 1 使 能 AS-SIST_DEBUG_CORE_1_SP_SPILL_MAX_INT。(R/W)

Register 22.64. ASSIST_DEBUG_CORE_1_INTR_CLR_REG (0x008C)



ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_RD_INT。(WT)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_DRAMO_0_WR_INT。(WT)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_RD_INT。(WT)

ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_DRAMO_1_WR_INT。(WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_PIF_0_RD_INT。(WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_PIF_0_WR_INT。(WT)

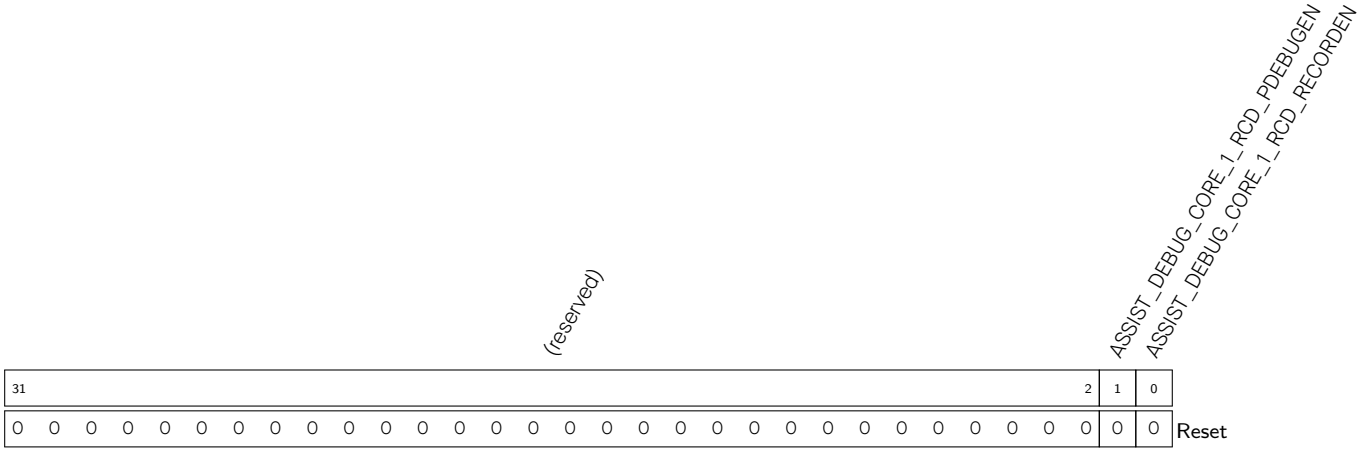
ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_PIF_1_RD_INT。(WT)

ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_AREA_PIF_1_WR_INT。(WT)

ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_SP_SPILL_MIN_INT。(WT)

ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_CLR 写 1 清除 ASSIST_DEBUG_CORE_1_SP_SPILL_MAX_INT。(WT)

Register 22.65. ASSIST_DEBUG_CORE_1_RCD_EN_REG (0x00C4)



ASSIST_DEBUG_CORE_1_RCD_RECORDEN 配置是否使能 HP CPU1 PC 记录。

0：不使能

1：ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG 开始实时记录 HP CPU1 PC

(R/W)

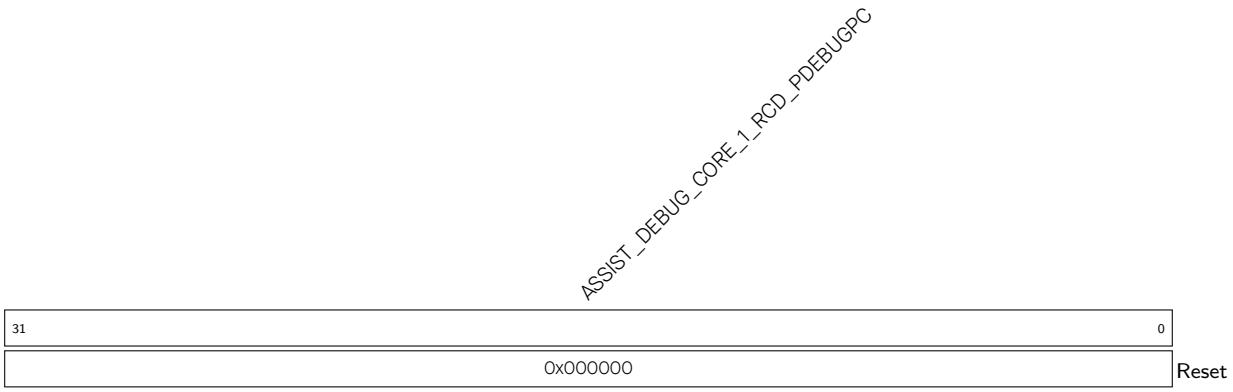
ASSIST_DEBUG_CORE_1_RCD_PDEBUGEN 配置是否使能 HP CPU1 调试。

0：不使能

1：HP CPU1 输出 HP CPU1 PC

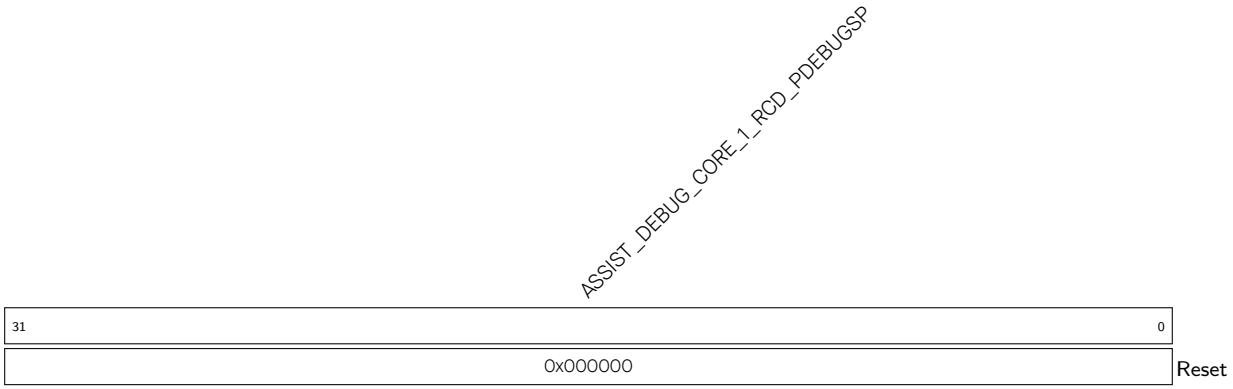
(R/W)

Register 22.66. ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC_REG (0x00C8)



ASSIST_DEBUG_CORE_1_RCD_PDEBUGPC 表示 HP CPU1 复位时刻的 PC 值。(RO)

Register 22.67. ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP_REG (0x00CC)



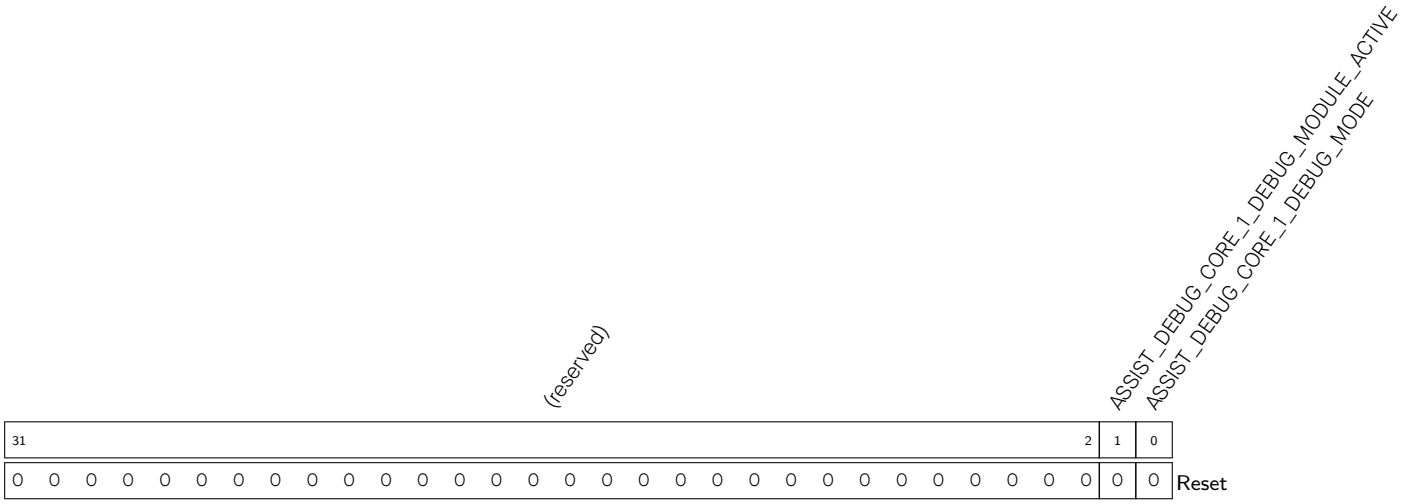
ASSIST_DEBUG_CORE_1_RCD_PDEBUGSP 表示 HP CPU1 SP。(RO)

Register 22.68. ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXCEPTION_REG (0x00F0)



ASSIST_DEBUG_CORE_1_LASTPC_BEFORE_EXC 表示 HP CPU1 异常前的最后一条指令的 HP CPU1 PC。(RO)

Register 22.69. ASSIST_DEBUG_CORE_1_DEBUG_MODE_REG (0x00F4)



ASSIST_DEBUG_CORE_1_DEBUG_MODE 表示 HP CPU1 是否处于调试模式。

1：处于调试模式

0：未处于调试模式

(RO)

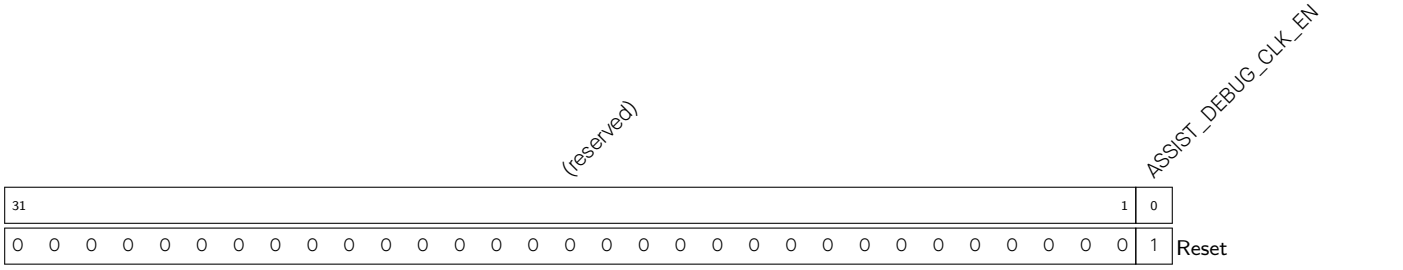
ASSIST_DEBUG_CORE_1_DEBUG_MODULE_ACTIVE 表示 HP CPU1 调试模块的工作状态。

1：处于工作状态

其它：未处于工作状态

(RO)

Register 22.70. ASSIST_DEBUG_CLOCK_GATE_REG (0x0108)



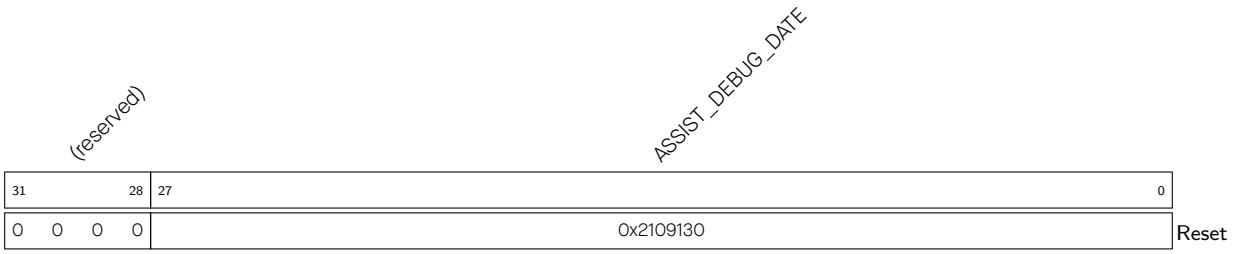
ASSIST_DEBUG_CLK_EN 配置是否使能寄存器时钟门控。

0：不使能

1：使能

(R/W)

Register 22.71. ASSIST_DEBUG_DATE_REG (0x03FC)



ASSIST_DEBUG_DATE 版本控制寄存器。(R/W)

第 23 章

AXI 性能监测器

23.1 概述

AXI 总线性能监测器 (Advanced eXtensible Interface Performance Monitor, AXI Perf Mon) 用于实时监测 AXI 总线上正在进行的传输行为，并根据监测的指标反馈监测结果。软件可根据监测结果了解总线传输当前状态，包括带宽、延迟等性能相关的数据，用于软件测试或开发时的调试。

AXI Perf Mon 有 9 个监测端口，每个端口对应一个 AXI matrix 上的设备，包括 cache、vdma_mst1、vdma_mst2、2d-dma、h264-dma_mst1、h264-dma_mst2、psram、camera 和 lcd。AXI Perf Mon 内部有 2 个监测单元，各监测单元完全相同且完全独立，同时被 9 个监测端口共享。监测启动前，用户可将各监测单元与任意一个监测端口绑定，监测对应端口的总线行为。

23.2 术语

为了更好地说明 AXI Perf Mon 的功能，本章使用了以下术语。

AXI 高级可扩展接口 (Advanced eXtensible Interface)，AMBA 3（Advanced Microcontroller Bus Architecture，高级微控制器总线架构）协议的一部分，为存储器和外设之间、存储器和存储器之间提供了高宽带、低延迟的数据传输。

agent AXI 总线上的设备，包括 AXI 总线上的主机接口 (Master) 和从机接口 (Slave)。

outstanding AXI 协议中，当一个 AXI 事务的地址已被从设备 (Slave) 接收，但最终响应 (BRESP/RRESP) 尚未返回主设备 (Master) 时，该事务即处于 Outstanding（未完成/在途）状态。

burst 在 AXI 协议中，Burst 传输是指在一次地址握手之后，通过连续的数据拍 (beat) 完成多次数据传输的机制。Burst 的引入旨在减少地址通道的重复开销，提高总线带宽利用率和传输效率。一次 Burst 传输由一个地址阶段和多个数据阶段组成，其中地址阶段仅在 Burst 开始时发生一次，后续每个数据拍按照既定规则连续传输，直到 Burst 结束。

AWVALID/AWREADY AXI 协议中的地址通道传输握手信号。AWVALID 拉高代表上游发起写命令传输请求，AWREADY 拉高代表下游响应了上游的写命令传输请求，握手成功即完成一笔写命令传输。ARVALID/ARREADY 同理，代表读传输命令的握手。

WVALID/WREADY AXI 协议中的数据通道传输握手信号。WVALID 拉高代表上游发起写数据传输请求，WREADY 拉高代表下游响应了上游的写数据传输请求，握手成功即完成一拍写数据传输。RVALID/RREADY 同理，但是读数据来源于下游，上游接收数据，握手成功即完成一拍读数据传输。

APB Advanced Peripheral Bus 是一种低带宽、低功耗的总线协议，主要用于连接低速外设（如 GPIO、定时器、UART、ADC 等）与系统的主处理器或其他高性能模块。APB 是 ARM AMBA (Advanced Microcontroller Bus Architecture) 总线家族的一部分，与 AHB (Advanced High-performance Bus) 和 AXI (Advanced eXtensible Interface) 一起组成了 AMBA 系统中的总线架构。

端口 模块边界上的逻辑接口，用于与外部逻辑进行信号通信。

单元 在较大的设计中实现特定功能的硬件模块。在本文中，特指负责性能监测的单元。

n/*m* 在有关寄存器的章节中，经常使用 *n*/*m* 来统一表示同类寄存器。*n* 表示监测单元 0 或 1；*m* 表示每个监测单元拥有多个同类寄存器，具体数量请参阅寄存器章节相关描述。

region 本文中 region 的含义与地址范围相关，但有所区别。在监测瞬时带宽时，可以通过定义 region 来过滤掉不相关的带宽统计信息。region 必须基于完整的从设备地址来定义，可以包含一个或多个完整的从设备地址范围，但不支持仅对单个从设备地址范围中的部分区间进行过滤。

23.3 主要特性

AXI Perf Mon 具有以下特性：

- 适配 AXI3 总线接口
- 2 个监测单元完全相同且独立
- 9 个监测端口
- 可通过软件配置各监测单元对应的监测端口
- 监测时支持传输事务通过传输 ID 进行过滤，被过滤的传输 ID 代表的传输事务将不再被统计
- 支持多个测量指标，用户根据需求选择对应指标进行监测
  - 读/写地址通道当前传输事务 outstanding 数
  - 读/写地址通道延迟
  - 读/写数据通道延迟
  - 单次传输事务中写数据通道空隙 cycle 总数
  - 读传输事务延迟
  - 写传输事务延迟
  - 平均带宽测量
  - 瞬时带宽测量
  - 瞬时带宽区间统计及异常瞬时带宽记录
- 每个监测单元最大支持 6 个测量指标计数器
- 各监测单元内部，每个指标计数器可配置统计任一指标，不允许多个指标计数器同时统计同一指标
- 每个监测单元最大支持 6 个区间统计指标计数器，即最大支持分成 6 个区间对瞬时带宽指标监测结果进行统计

23.4 结构概览

图 23.4-1 为 AXI Perf Mon 的基本架构图。

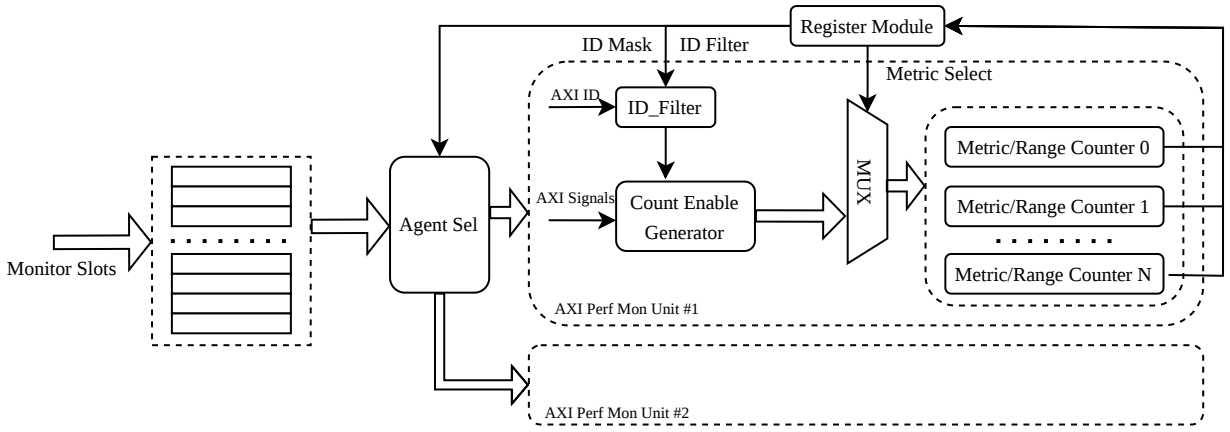


图 23.4-1. AXI Perf Mon 的架构

AXI Perf Mon 主要包含以下功能模块：

- **待监测端口选择**：从接入 AXI Perf Mon 的端口中选出接入监测单元的端口
- **ID 过滤**：对监测事件进行分类
- **计数使能生成器**：将用户通过寄存器配置选择的统计指标信息转换成计数器的开启和关闭指示
- **计数器**：执行统计计数。

23.5 功能描述

23.5.1 时钟

AXI Perf Mon 在 2 个独立的时钟域下运行，每个时钟域对应特定的接口：

- APB_CLK：用于配置 AXI Perf Mon 寄存器的 APB 总线时钟
- AXI_CLK：SoC 系统 AXI 总线时钟

AXI Perf Mon 属于系统辅助 debug 组件，因此在正常工作状态时会默认关闭时钟使能。如果要使用 AXI Perf Mon，需要先在时钟复位模块中打开时钟使能。AXI Perf Mon 监测单元的工作时钟与被监测总线的时钟完全同频同相，下文提到的时钟 cycle 均基于 AXI Perf Mon 的 AXI_CLK。

有关时钟的更多信息，请查看章节 11 复位和时钟。

时钟的详细配置流程，请查看章节 23.7 配置流程。

23.5.2 复位

AXI Perf Mon 支持软件复位，可复位整个模块，包括所有监测单元和寄存器。

可通过向 HP_SYS_CLKRST_REG_AXI_PERF_MON_SYS_RST_EN 字段先写 1 后写 0 实现复位。

23.5.3 传输 ID 过滤

- 在 AXI 传输事务中，每笔传输事务都存在一个对应的传输 ID，用于标记此笔传输事务。部分 Master 侧会将不同种类的传输事务分成不同的传输 ID。传输 ID 过滤器用于屏蔽一些用户不关注的传输事务，避免统

计结果受到干扰。

- ID 过滤可通过配置 AXI_PERF_MON_SEL_AG*n*_RD_ID_MASK 或 AXI_PERF_MON_SEL_AG*n*_WR_ID_MASK（ID 掩码）和 AXI_PERF_MON_SEL_AG*n*_RD_ID_FILTER 或 AXI_PERF_MON_SEL_AG*n*_WR_ID_FILTER （ID 过滤值）来实现。
- ID 掩码中，0 表示可忽略的位，1 表示用于过滤的位。若传输 ID 在掩码为 1 的位上与过滤值不符，则该传输事务被屏蔽，不计入统计。
- ID 过滤既可以作用于 Master 侧也可以作用于 Slave 侧。Master 侧 AXI 传输事务 ID 为 4 bit，Slave 侧传输事务 ID 为 8 bit。

23.5.4 监测指标

AXI Perf Mon 的端口监测功能由软件通过配置 AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG 指标选择寄存器实现。其中：

- *n* 表示该寄存器所属的 AXI Perf Mon 监测单元编号，取值范围为 0 到 1。
- *m* 表示该寄存器在每个监测单元内的索引编号，取值范围为 1 到 2。

每个寄存器被划分为四个 8-bit 的段，以 *i* 作为索引（取值范围为 0 到 3）。在每个段中，位 [8*i*+4:8*i*] 用于选择测量指标，位 [8*i*+7:8*i*+5] 用于选择区间统计指标。

指标选择寄存器的配置值与测量指标以及区间统计指标的对应关系见表 23.5-1 和表 23.5-2。

表 23.5-1. 指标选择寄存器中的测量指标配置

指标选择寄存器 [8*i*+4:8*i*] 配置值	测量指标
0	无效值
1	写事务 outstanding 数量统计
2	读事务 outstanding 数量统计
3	写地址通道延迟统计
4	读地址通道延迟统计
5	写数据通道延迟统计
6	读数据通道延迟统计
7	写数据通道 WVALID 空隙统计
8	读事务传输延迟统计
9	写事务传输延迟统计
10	平均带宽测量中读事务有效数据传输次数统计
11	平均带宽测量中读传输时间长度统计
12	平均带宽测量中写事务有效数据传输次数统计
13	平均带宽测量中写传输时间长度统计
14	瞬时带宽测量中读事务有效数据传输次数统计
15	瞬时带宽测量中读传输时间长度统计
16	瞬时带宽测量中写事务有效数据传输次数统计
17	瞬时带宽测量中写传输时间长度统计
18~31	无效值

表 23.5-2. 指标选择寄存器中的区间统计指标配置

指标选择寄存器 [8*i+7:8*i+5] 配置值	区间统计指标
0	无效值
1	读事务瞬时带宽区间统计
2	写事务瞬时带宽区间统计
3~7	无效值

通过配置 agent 选择寄存器（AXI_PERF_MON_AGENT_SELECT_REG），可以选择 AXI Perf Mon 各监测单元所监测的端口。

该寄存器被划分为两个段，以 *i* 作为索引（取值范围为 0 到 1），分别对应监测单元 *i* 的端口选择配置：

- 位 [4:0] 用于选择监测单元 0 所监测的 agent。
- 位 [9:5] 用于选择监测单元 1 所监测的 agent。

寄存器的配置值与实际 agent 的对应关系见表 23.5-3。

表 23.5-3. agent 选择寄存器配置值与实际 agent 对应关系表

agent 选择寄存器 [5*i+4:5*i] 配置值	agent
0	无效值
1	cache
2	vdma_mst1
3	vdma_mst2
4	2d-dma
5	h264-dma_mst1
6	h264-dma_mst2
7	psram
8	camera
9	lcd
10~31	无效值

23.5.4.1 读/写地址通道传输事务最大 outstanding 数量

outstanding 能力反映了 Master 侧发送 AXI 传输事务的并发能力，outstanding 能力越强，传输延迟越小。在实际传输过程中，outstanding 能力不仅取决于 Master 的配置，还受到总线组件以及上下游设备 outstanding 能力的限制。AXI Perf Mon 支持测量当前监测端口传输事务的 outstanding 数量。

- 对于读传输，当地址通道信息有效发出后，数据通道对应此传输的最后一笔数据传输未完成，则累加一笔 outstanding 传输；如数据传输完成，则 outstanding 数量减一。
- 对于写传输，当地址通道信息有效发出后，写响应通道对应此笔传输事务的响应没有完成，则累加一笔 outstanding 传输；如得到对应写传输的写响应，则 outstanding 数量减一。
- 传输过程中，outstanding 值一般会一直发生变化。AXI Perf Mon 会将出现的最大 outstanding 值写入结果寄存器中，供软件参考分析。

23.5.4.2 读/写地址通道延迟

地址通道延迟代表一笔传输命令发出后多长时间会被接收。AXI Perf Mon 支持对当前监测端口传输事务的读/写地址通道延迟进行测量。

- 对于读传输，当地址通道 ARVALID 拉高后，表示 Master 侧准备发出一笔读传输事务的命令，Slave 侧根据当前空闲情况选择是否拉高 ARREADY。ARREADY 拉高后，地址通道完成一笔传输命令的握手。读地址通道延迟是指该监测端口 ARVALID 拉高至 ARREADY 拉高的 cycle 数。
- 写地址通道延迟的测量方式和读地址通道延迟测量方式一致。
- 传输过程中，对于同一个监测端口，其延迟值一般也会一直发生变化。AXI Perf Mon 会将出现的最大/最小读/写地址通道延迟值写入结果寄存器中，供软件参考分析。

23.5.4.3 读/写数据通道延迟

对于 AXI 传输，一笔 burst 传输开始后不可中断。但在实际传输过程中，数据通道并非完全的背靠背传输，传输中可能存在空闲周期。AXI Perf Mon 支持测量当前监测端口传输事务的读/写数据通道延迟。

- 对于读传输，当数据通道 RVALID 拉高后，表示 Slave 侧准备发出一拍读传输事务的数据，Master 侧根据当前空闲情况选择是否拉高 RREADY。RREADY 拉高后，读数据通道完成一拍数据传输的握手。读数据通道延迟是指该监测端口 RVALID 拉高至 RREADY 拉高的 cycle 数。
- 写数据通道延迟的测量方式和读数据通道延迟测量方式一致。
- 传输过程中，对于同一个监测端口，其延迟值一般也会一直发生变化，AXI Perf Mon 会将出现的最大/最小读/写数据通道延迟值写入结果寄存器中，供软件参考分析。

23.5.4.4 写数据通道 WVALID 空隙

当 Master 侧开始发送一笔 burst 数据传输后，数据传输之间的空闲周期可能来源于 Slave 侧接收数据的等待、Master 侧 WVALID 的空隙，以及经过转换桥时产生的显著空隙。AXI Perf Mon 支持对当前监测端口传输事务中一笔传输事务的 WVALID 空隙进行统计。

- 一笔写传输事务中，如果传输还未完成，WVALID 一旦拉低，统计 WVALID 拉低的 cycle 总数。
- 传输过程中，对于同一个监测端口，其写数据通道 WVALID 空隙统计数量一般也会一直发生变化，AXI Perf Mon 会将出现的最大/最小写数据通道 WVALID 空隙写入结果寄存器中，供软件参考分析。

23.5.4.5 读/写事务传输延迟

在传输过程中，不仅关注地址通道和数据通道各自的延迟，还会关注传输事务中地址通道命令准备发送时到数据通道准备接收数据时的延迟。该延迟反映了从发出传输命令到开始数据传输的时间间隔，可辅助决定 Master 侧 AXI 传输缓存 buffer 的大小。AXI Perf Mon 支持统计当前监测端口传输事务中一笔传输事务的读/写事务传输延迟。

- 对于读传输，当一笔传输的 ARVALID 拉高后，指标测量计数器会开始统计延迟，直到这笔传输第一拍数据的 RVALID 信号拉高，计数器停止计数，得到读事务传输延迟。
- 对于写传输，当一笔传输的 AWVALID 拉高后，指标测量计数器会开始统计延迟，直到这笔传输的第一拍数据完成传输 (统计终点与读传输有区别)，计数器停止计数，得到写事务传输延迟。
- 监测端口可能存在多笔 outstanding 传输，但在统计读/写传输事务延迟指标中，指标计数器只有一个，因此每次只能统计一笔事务的延迟。当开始统计某笔事务后，其他并发事务将被忽略，直到该笔统计完成。为使统计数据具有代表性，优先选择传输时间较长的事务进行统计。

- 传输过程中，对于同一个监测端口，其传输事务延迟统计值一般也会一直发生变化，AXI Perf Mon 会将出现的最大/最小传输事务延迟值写入结果寄存器中，供软件参考分析。

23.5.4.6 平均带宽测量

带宽是评估总线性能最直观的指标之一。AXI Perf Mon 支持测量平均带宽。

- AXI Perf Mon 内部没有计算单元，并不直接计算带宽值，而是通过统计时间（总线 cycle 数量）和传输数据量（data 数量）来间接统计带宽。
- 由于需要观测时间和数据量，平均带宽测量需要同时使用两个指标计数器。带宽统计会忽略 data strobe 的影响，只统计数据量，即使是无效数据，也会被统计。
- 当使能平均带宽测量时，时间计数器将跟随总线 cycle 一起递增，而数据量根据当前数据通道握手情况递增。每完成一笔 burst 传输，会将当前时间和数据量统计结果刷新到结果寄存器中。如果数据传输完成后没有关闭平均带宽测量，数据通道一直处于 idle 状态，那么统计结果将一直保持在最后一笔 burst 传输结束时的时间和数据量统计结果，不会增加空白时间。软件需要根据总线频率，一个 beat 代表的数据 byte 数量来计算得到实际的平均带宽。

23.5.4.7 瞬时带宽测量

平均带宽体现了在较长一段时间内设备对带宽的需求，但是在评估总线性能时，还应关注较短时间窗口内的设备带宽变化，即瞬时带宽。AXI Pref Mon 也支持测量瞬时带宽。

- 瞬时带宽通过时间窗口滑动的方式来进行测量。当配置好时间窗口的长度（以总线 cycle 为基本单位）并使能瞬时带宽测量后，测量会以软件无感的方式进行。测量的起始和终止行为完全由硬件操作，当一个时间窗口测量结束后，下一个时间窗口的测量开始，进行连续的测量。
- 瞬时带宽的测量和平均带宽测量方法基本一致，都需要两个指标计数器，分别对时间长度和数据量进行测量。当时间计数器达到寄存器配置的时间窗口值时，表示一次瞬时带宽测量结束，将统计的数据量反馈给下一级的极值比较器，并清空数据量计数器和时间计数器，开始下一轮计数。
- 区间统计。瞬时带宽如果仅记录极值，不足以反映总线传输情况，因此引入区间统计。区间统计基于区间统计计数器，会有由软件配置的区间上限和区间下限，其数量和指标计数器一样。当指标计数器结果统计完成后，会将其结果与区间计数器对应的区间对比，进行分类统计，更能反映系统情况。
- 事件记录。瞬时带宽测量时，会将监测端口最近 4 次瞬时带宽测量值（当前未完成测量的瞬时带宽值不做统计）和最近 4 笔 AXI 接口传输信息（ID/Burst length）信息写入寄存器中。地址通道信息在地址通道握手时记录，而瞬时带宽测量值在测量窗口结束时记录，存在记录时间差异。这些信息是根据传输实际情况同步更新的，方便软件了解当前传输的基本情况。
- 异常反馈。瞬时带宽测量时还会对异常的带宽变化进行反馈。对监测端口设立最小瞬时带宽阈值（基于数据量），代表传输过程中认为此端口处于非异常情况的最小瞬时带宽值，一旦指标计数器统计结果小于此阈值时，触发中断标志信号，并广播到所有的 AXI Perf Mon 监测单元，停止所有端口瞬时带宽的测量，事件记录停止，保持现场，根据事件记录的结果，辅助判断哪个设备抢占了带宽。软件也可以通过配置 AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_EN 或 AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_EN 来暂停对应监测单元的事件记录。
- 地址过滤。一般情况下，访问内部存储不需要额外关注带宽。在瞬时带宽测量时，如果需要对于异常进行进行反馈，一般是多 Master 对于同一个外部存储如 PSRAM 同时发起访问。那么在异常结果分析时，需要排除其余 Master 对于其余 Slave 的访问，这种带宽对分析的场景属于无效带宽。因此增加地址过滤的功能，可以根据访问地址进行过滤，从而获得真正有竞争关系的多 Master 的瞬时带宽。地址过滤的基本单位是 Slave，即支持对不同 Slave 进行地址过滤，但不支持对同一个 Slave 的部分地址空间进行过滤。

- 监测恢复。在异常带宽行为被发现且广播后，各 AXI Perf Mon 会暂停对瞬时带宽的测量。当软件对当前现场结果分析完毕后，若希望在不对系统复位的情况下重新保持对瞬时带宽的监测，可通过配置 AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_CLR 或 AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_CLR 的动作来清除停滞状态测。

23.6 中断

ESP32-P4 的 AXI Perf Mon 可以生成以下中断信号并发送给中断矩阵。

- AXI_PERF_MON_INTR

此中断信号由 AXI Perf Mon 的内部中断源生成。表 23.6-1 列出了 AXI Perf Mon 的内部中断源、中断触发条件、以及生成的中断信号。

表 23.6-1. AXI PERF MON 的内部中断源

内部中断源	触发条件	中断信号
AXI_PERF_MON_SEL_AG*n*_RECORD_-CNT_UNDER_LIMIT_INT (*n* = 0, 1)	瞬时带宽测量值小于设置的最小瞬时带宽阈值	AXI_PERF_MON_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 23.8 寄存器列表。

23.7 配置流程

23.7.1 使用 AXI Perf Mon 测量 outstanding

利用 AXI Perf Mon 监测某待监测端口的 outstanding 能力：

1. 配置寄存器 AXI_PERF_MON_AGENT_SELECT_REG，每 5 bit 代表一个 agent 被对应的监测单元选中，即 [4:0] 表示监测单元 0 选择的待监测端口，[9:5] 代表监测单元 1 选择的待监测端口。
2. 配置寄存器 AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG (*n* 代表监测单元 0 或 1；*m* 代表每个监测单元存在多个此寄存器，可参照寄存器小节查看具体寄存器数量，此处 *m* = 1, 2) 选择监测的指标，每 8 bit 代表一个指标计数器对应的指标，AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT1 [4:0] 的值代表测量指标计数器 0 对应的监测指标，其 [7:5] 的值代表区间指标计数器 0 对应的监测指标。AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT2 [4:0] 的值代表测量指标计数器 4 对应的监测指标，其 [7:5] 的值代表区间指标计数器 4 对应的监测指标，以此类推。配置完成后，监测开始。
3. 通过读取结果寄存器中 AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_RESULT 的值获取状态信息，[15:0] 是统计的最小值，[31:16] 是统计的最大值。结果计数器序号与选择的指标计数器对应。
4. 将寄存器 AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG 中对应指标计数器的 8 bit 字段配置为 0，停止该指标计数器对此端口这一指标的监测，并将计数器复位。

outstanding 数量、地址通道延迟、数据通道延迟、读写事务传输延迟等均可以通过这个配置流程进行监测，配置寄存器 AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG 到期望监测的指标即可。

23.7.2 使用 AXI Perf Mon 测量读平均带宽

利用 AXI Perf Mon 监测某待监测端口在一段时间内的平均带宽：

1. 配置寄存器 AXI_PERF_MON_AGENT_SELECT_REG 选择待监测端口。
2. 配置寄存器 AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG 选择监测的指标为读平均带宽时间长度测量和数据量测量。
3. 配置 AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TIME_CNT_VALID_STROBE_NUM 和 AXI_PERF_MON_SEL_AG*n*_RD_BANDW_DATA_CNT_VALID_STROBE_NUM，选择时间和数据的计量步长。计量步长如果配成 3，那么每 4 个有效 cycle 使计数器加 1，每 4 个有效数据使计数器加 1。
4. 配置 AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_EN 或 AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_EN 使能平均带宽测量，配置 AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_STOP 或 AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_STOP 停止平均带宽测量。
5. 测量结果可通过寄存器 AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_RESULT_REG 读取，结果计数器序号与选择的指标计数器对应。结果寄存器中高 16 bit 代表平均带宽数据/时间统计量，如果为 16'hFFFF，代表结果统计溢出，结果失效，需配置 AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TIME_CNT_VALID_STROBE_NUM 和 AXI_PERF_MON_SEL_AG*n*_RD_BANDW_DATA_CNT_VALID_STROBE_NUM 增大统计步长，或减少统计时间和数据量。

23.7.3 使用 AXI Perf Mon 测量读瞬时带宽

利用 AXI Perf Mon 监测某待监测端口的瞬时带宽：

1. 配置寄存器 AXI_PERF_MON_AGENT_SELECT_REG 选择待监测端口。
2. 配置寄存器 AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG 选择监测的指标为读瞬时带宽时间长度测量和数据量测量。
3. 测量瞬时带宽时，由软件规定每一段测量的时间长度。达到测量时长后，读取数据统计结果寄存器里的值可得到该时间长度内传输的数据量，同时开始重新统计下一段测量时间内的数据量。配置寄存器 AXI_PERF_MON_SEL_AG*n*_RD_INS_BANDW_TIME_THR_REG 决定测量时间长度。配置 AXI_PERF_MON_SEL_AG0_RD_INS_BANDW_TEST_EN 或 AXI_PERF_MON_SEL_AG1_RD_INS_BANDW_TEST_EN 可以使能瞬时带宽测量。
4. 测量瞬时带宽时，若希望反馈端口在密集数据传输场景下带宽陡降的异常情况，可以配置 AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_RD_DATA_LIMIT 填写带宽数据量下限阈值。一旦统计的瞬时带宽数据量小于该下限，AXI Perf Mon 会产生中断并停止测量 AXI Perf Mon 监测单元瞬时带宽。
5. 瞬时带宽的事件记录信息可查询寄存器 AXI_PERF_MON_SEL_AG*n*_INS_BANDW_DATA*m*_REG 和 AXI_PERF_MON_SEL_AG*n*_RD_AXI_INFO_RECORD*m*_REG。这些寄存器的值根据实时统计情况刷新，当中断产生时，这些寄存器将暂停更新，以保留现场。可通过 AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_EN 或 AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_EN 来停止寄存器更新。通过

AXI_PERF_MON_SEL_AGO_SW_RECORD_STOP_CLR 或
AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_CLR 来清除停滞状态，恢复记录更新。

6. 测量瞬时带宽时，允许对数据进行地址过滤。地址过滤以 region 为单位，由 AXI_PERF_MON_SEL_AG*n*_RD_ADDR_MASK*m* 和 AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER*m* 过滤出三个 region，由 AXI_PERF_MON_SEL_AGO_RD_ADDR_REGION_SEL 或 AXI_PERF_MON_SEL_AG1_RD_ADDR_REGION_SEL 来选择哪些 region 是有效目的地址。
7. 瞬时带宽统计支持以区间统计的方式来系统地呈现带宽分布状态。配置寄存器 AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG 每 8 bit 的后 3 bit 用于指定区间计数器的统计指标。AXI_PERF_MON__SEL_AG*n*_METRIC_RANGE_CNT_LOW 和 AXI_PERF_MON__SEL_AG*n*_METRIC_RANGE_CNT_HIGH 用于指定每个区间的上下限。瞬时带宽的区间统计结果可以通过寄存器 AXI_PERF_MON_SEL_AG*n*_RANGE*m*_REG 查询。

23.8 寄存器列表

本小节的所有地址均为相对于 AXI 性能监测器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
AXI_PERF_MON_CLK_EN_REG	reserved	0x0000	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE0_REG	监测单元 0 对应的区间统计上下限配置寄存器 0	0x00C8	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE0_REG	监测单元 1 对应的区间统计上下限配置寄存器 0	0x00CC	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE1_REG	监测单元 0 对应的区间统计上下限配置寄存器 1	0x00D0	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE1_REG	监测单元 1 对应的区间统计上下限配置寄存器 1	0x00D4	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE2_REG	监测单元 0 对应的区间统计上下限配置寄存器 2	0x00D8	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE2_REG	监测单元 1 对应的区间统计上下限配置寄存器 2	0x00DC	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE3_REG	监测单元 0 对应的区间统计上下限配置寄存器 3	0x00E0	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE3_REG	监测单元 1 对应的区间统计上下限配置寄存器 3	0x00E4	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE4_REG	监测单元 0 对应的区间统计上下限配置寄存器 4	0x00E8	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE4_REG	监测单元 1 对应的区间统计上下限配置寄存器 4	0x00EC	R/W
AXI_PERF_MON_SEL_AG0_METRIC_RANGE5_REG	监测单元 0 对应的区间统计上下限配置寄存器 5	0x00F0	R/W
AXI_PERF_MON_SEL_AG1_METRIC_RANGE5_REG	监测单元 1 对应的区间统计上下限配置寄存器 5	0x00F4	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_MASK0_REG	监测单元 0 对应的读 region 0 地址过滤掩码	0x00F8	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_MASK0_REG	监测单元 1 对应的读 region 0 地址过滤掩码	0x00FC	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_MASK1_REG	监测单元 0 对应的读 region 1 地址过滤掩码	0x0100	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_MASK1_REG	监测单元 1 对应的读 region 1 地址过滤掩码	0x0104	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_MASK2_REG	监测单元 0 对应的读 region 2 地址过滤掩码	0x0108	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_MASK2_REG	监测单元 1 对应的读 region 2 地址过滤掩码	0x010C	R/W

名称	描述	地址	访问
AXI_PERF_MON_SEL_AG0_WR_ADDR_MASK0_REG	监测单元 0 对应的写 region 0 地址过滤掩码	0x0110	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_MASK0_REG	监测单元 1 对应的写 region 0 地址过滤掩码	0x0114	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_MASK1_REG	监测单元 0 对应的写 region 1 地址过滤掩码	0x0118	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_MASK1_REG	监测单元 1 对应的写 region 1 地址过滤掩码	0x011C	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_MASK2_REG	监测单元 0 对应的写 region 2 地址过滤掩码	0x0120	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_MASK2_REG	监测单元 1 对应的写 region 2 地址过滤掩码	0x0124	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_FILTER0_REG	监测单元 0 对应的读 region 0 地址过滤值	0x0128	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_FILTER0_REG	监测单元 1 对应的读 region 0 地址过滤值	0x012C	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_FILTER1_REG	监测单元 0 对应的读 region 1 地址过滤值	0x0130	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_FILTER1_REG	监测单元 1 对应的读 region 1 地址过滤值	0x0134	R/W
AXI_PERF_MON_SEL_AG0_RD_ADDR_FILTER2_REG	监测单元 0 对应的读 region 2 地址过滤值	0x0138	R/W
AXI_PERF_MON_SEL_AG1_RD_ADDR_FILTER2_REG	监测单元 1 对应的读 region 2 地址过滤值	0x013C	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_FILTER0_REG	监测单元 0 对应的写 region 0 地址过滤值	0x0140	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_FILTER0_REG	监测单元 1 对应的写 region 0 地址过滤值	0x0144	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_FILTER1_REG	监测单元 0 对应的写 region 1 地址过滤值	0x0148	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_FILTER1_REG	监测单元 1 对应的写 region 1 地址过滤值	0x014C	R/W
AXI_PERF_MON_SEL_AG0_WR_ADDR_FILTER2_REG	监测单元 0 对应的写 region 2 地址过滤值	0x0150	R/W
AXI_PERF_MON_SEL_AG1_WR_ADDR_FILTER2_REG	监测单元 1 对应的写 region 2 地址过滤值	0x0154	R/W
AXI_PERF_MON_SEL_AG0_METRIC_SELECT1_REG	监测单元 0 对应的指标选择寄存器 1	0x0158	R/W
AXI_PERF_MON_SEL_AG1_METRIC_SELECT1_REG	监测单元 1 对应的指标选择寄存器 1	0x015C	R/W
AXI_PERF_MON_SEL_AG0_METRIC_SELECT2_REG	监测单元 0 对应的指标选择寄存器 2	0x0160	R/W
AXI_PERF_MON_SEL_AG1_METRIC_SELECT2_REG	监测单元 1 对应的指标选择寄存器 2	0x0164	R/W

名称	描述	地址	访问
AXI_PERF_MON_SEL_AG_RD_ADDR_REGION_SEL_REG	地址过滤被选择的读 region	0x0168	R/W
AXI_PERF_MON_SEL_AG_WR_ADDR_REGION_SEL_REG	地址过滤被选择的写 region	0x016C	R/W
AXI_PERF_MON_SEL_AG_ADDR_FILTER_EN_REG	地址过滤使能	0x0170	R/W
AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_EN_REG	瞬时带宽监测事件记录暂停使能	0x0174	WT
AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_CLR_REG	瞬时带宽监测恢复事件记录	0x0178	WT
AXI_PERF_MON_SEL_AG_INS_BANDW_TEST_EN_REG	瞬时带宽测试使能	0x017C	R/W
AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_DATA_-LIMIT_REG	监测单元 0 瞬时带宽最小带宽数据量阈值	0x0180	R/W
AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_DATA_-LIMIT_REG	监测单元 1 瞬时带宽最小带宽数据量阈值	0x0184	R/W
AXI_PERF_MON_SEL_AG0_ID_MASK_REG	监测单元 0 ID 过滤掩码	0x0188	R/W
AXI_PERF_MON_SEL_AG1_ID_MASK_REG	监测单元 1 ID 过滤掩码	0x018C	R/W
AXI_PERF_MON_SEL_AG0_ID_FILTER_REG	监测单元 0 ID 过滤值	0x0190	R/W
AXI_PERF_MON_SEL_AG1_ID_FILTER_REG	监测单元 1 ID 过滤值	0x0194	R/W
AXI_PERF_MON_SEL_AG_BANDW_TEST_EN_REG	平均带宽测量使能	0x0198	R/W
AXI_PERF_MON_SEL_AG_BANDW_TEST_STOP_REG	平均带宽测量停止	0x019C	R/W
AXI_PERF_MON_SEL_AG0_BANDW_TRIGGER_IN_SEL_REG	监测单元 0 平均带宽测量触发信号源选择	0x01A0	R/W
AXI_PERF_MON_SEL_AG1_BANDW_TRIGGER_IN_SEL_REG	监测单元 1 平均带宽测量触发信号源选择	0x01A4	R/W
AXI_PERF_MON_SEL_AG0_INS_BANDW_TIME_THR_REG	监测单元 0 瞬时带宽测量时间长度	0x01B8	R/W
AXI_PERF_MON_SEL_AG1_INS_BANDW_TIME_THR_REG	监测单元 1 瞬时带宽测量时间长度	0x01BC	R/W
agent 选择寄存器			
AXI_PERF_MON_AGENT_SELECT_REG	选择待监测的 agent	0x0004	R/W
状态寄存器			
AXI_PERF_MON_SEL_AG0_COUNTER0_REG	监测单元 0 指标计数结果寄存器 0	0x0008	RO
AXI_PERF_MON_SEL_AG1_COUNTER0_REG	监测单元 1 指标计数结果寄存器 0	0x000C	RO
AXI_PERF_MON_SEL_AG0_COUNTER1_REG	监测单元 0 指标计数结果寄存器 1	0x0010	RO
AXI_PERF_MON_SEL_AG1_COUNTER1_REG	监测单元 1 指标计数结果寄存器 1	0x0014	RO
AXI_PERF_MON_SEL_AG0_COUNTER2_REG	监测单元 0 指标计数结果寄存器 2	0x0018	RO
AXI_PERF_MON_SEL_AG1_COUNTER2_REG	监测单元 1 指标计数结果寄存器 2	0x001C	RO
AXI_PERF_MON_SEL_AG0_COUNTER3_REG	监测单元 0 指标计数结果寄存器 3	0x0020	RO
AXI_PERF_MON_SEL_AG1_COUNTER3_REG	监测单元 1 指标计数结果寄存器 3	0x0024	RO

名称	描述	地址	访问
AXI_PERF_MON_SEL_AG0_COUNTER4_REG	监测单元 0 指标计数结果寄存器 4	0x0028	RO
AXI_PERF_MON_SEL_AG1_COUNTER4_REG	监测单元 1 指标计数结果寄存器 4	0x002C	RO
AXI_PERF_MON_SEL_AG0_COUNTER5_REG	监测单元 0 指标计数结果寄存器 5	0x0030	RO
AXI_PERF_MON_SEL_AG1_COUNTER5_REG	监测单元 1 指标计数结果寄存器 5	0x0034	RO
AXI_PERF_MON_SEL_AG0_RANGE0_REG	监测单元 0 区间统计结果寄存器 0	0x0038	RO
AXI_PERF_MON_SEL_AG1_RANGE0_REG	监测单元 1 区间统计结果寄存器 0	0x003C	RO
AXI_PERF_MON_SEL_AG0_RANGE1_REG	监测单元 0 区间统计结果寄存器 1	0x0040	RO
AXI_PERF_MON_SEL_AG1_RANGE1_REG	监测单元 1 区间统计结果寄存器 1	0x0044	RO
AXI_PERF_MON_SEL_AG0_RANGE2_REG	监测单元 0 区间统计结果寄存器 2	0x0048	RO
AXI_PERF_MON_SEL_AG1_RANGE2_REG	监测单元 1 区间统计结果寄存器 2	0x004C	RO
AXI_PERF_MON_SEL_AG0_RANGE3_REG	监测单元 0 区间统计结果寄存器 3	0x0050	RO
AXI_PERF_MON_SEL_AG1_RANGE3_REG	监测单元 1 区间统计结果寄存器 3	0x0054	RO
AXI_PERF_MON_SEL_AG0_RANGE4_REG	监测单元 0 区间统计结果寄存器 4	0x0058	RO
AXI_PERF_MON_SEL_AG1_RANGE4_REG	监测单元 1 区间统计结果寄存器 4	0x005C	RO
AXI_PERF_MON_SEL_AG0_RANGE5_REG	监测单元 0 区间统计结果寄存器 5	0x0060	RO
AXI_PERF_MON_SEL_AG1_RANGE5_REG	监测单元 1 区间统计结果寄存器 5	0x0064	RO
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD0_REG	监测单元 0 AXI 读传输信息记录寄存器 0	0x0068	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD0_REG	监测单元 1 AXI 读传输信息记录寄存器 0	0x006C	RO
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD1_REG	监测单元 0 AXI 读传输信息记录寄存器 1	0x0070	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD1_REG	监测单元 1 AXI 读传输信息记录寄存器 1	0x0074	RO
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD2_REG	监测单元 0 AXI 读传输信息记录寄存器 2	0x0078	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD2_REG	监测单元 1 AXI 读传输信息记录寄存器 2	0x007C	RO

名称	描述	地址	访问
AXI_PERF_MON_SEL_AG0_RD_AXI_INFO_RECORD3_REG	监测单元 0 AXI 读传输信息记录寄存器 3	0x0080	RO
AXI_PERF_MON_SEL_AG1_RD_AXI_INFO_RECORD3_REG	监测单元 1 AXI 读传输信息记录寄存器 3	0x0084	RO
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD0_REG	监测单元 0 AXI 写传输信息记录寄存器 0	0x0088	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD0_REG	监测单元 1 AXI 写传输信息记录寄存器 0	0x008C	RO
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD1_REG	监测单元 0 AXI 写传输信息记录寄存器 1	0x0090	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD1_REG	监测单元 1 AXI 写传输信息记录寄存器 1	0x0094	RO
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD2_REG	监测单元 0 AXI 写传输信息记录寄存器 2	0x0098	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD2_REG	监测单元 1 AXI 写传输信息记录寄存器 2	0x009C	RO
AXI_PERF_MON_SEL_AG0_WR_AXI_INFO_RECORD3_REG	监测单元 0 AXI 写传输信息记录寄存器 3	0x00A0	RO
AXI_PERF_MON_SEL_AG1_WR_AXI_INFO_RECORD3_REG	监测单元 1 AXI 写传输信息记录寄存器 3	0x00A4	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA0_REG	监测单元 0 瞬时带宽记录寄存器 0	0x00A8	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA0_REG	监测单元 1 瞬时带宽记录寄存器 0	0x00AC	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA1_REG	监测单元 0 瞬时带宽记录寄存器 1	0x00B0	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA1_REG	监测单元 1 瞬时带宽记录寄存器 1	0x00B4	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA2_REG	监测单元 0 瞬时带宽记录寄存器 2	0x00B8	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA2_REG	监测单元 1 瞬时带宽记录寄存器 2	0x00BC	RO
AXI_PERF_MON_SEL_AG0_INS_BANDW_DATA3_REG	监测单元 0 瞬时带宽记录寄存器 3	0x00C0	RO
AXI_PERF_MON_SEL_AG1_INS_BANDW_DATA3_REG	监测单元 1 瞬时带宽记录寄存器 3	0x00C4	RO
AXI_PERF_MON_SEL_AG0_WR_BANDW_CNT_VALID_-STROBE_NUM_REG	监测单元 0 写平均带宽计量步长寄存器	0x01A8	R/W
AXI_PERF_MON_SEL_AG1_WR_BANDW_CNT_VALID_-STROBE_NUM_REG	监测单元 1 写平均带宽计量步长寄存器	0x01AC	R/W
AXI_PERF_MON_SEL_AG0_RD_BANDW_CNT_VALID_-STROBE_NUM_REG	监测单元 0 读平均带宽计量步长寄存器	0x01B0	R/W
AXI_PERF_MON_SEL_AG1_RD_BANDW_CNT_VALID_-STROBE_NUM_REG	监测单元 1 读平均带宽计量步长寄存器	0x01B4	R/W

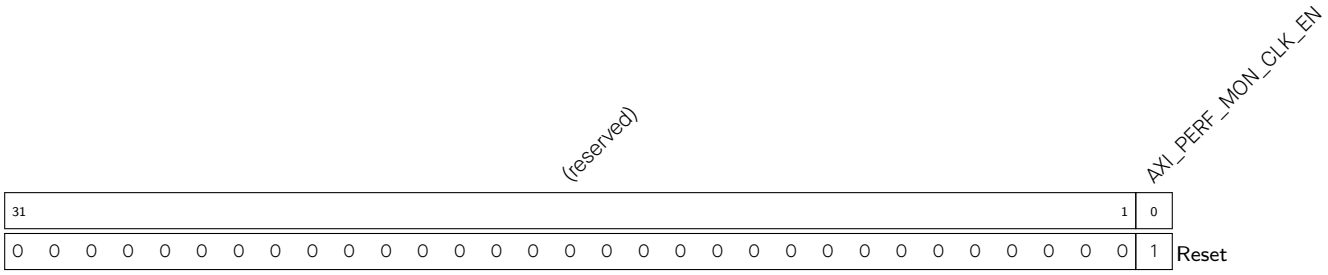
名称	描述	地址	访问
中断寄存器			
AXI_PERF_MON_SEL_AG_INT_RAW_REG	原始中断寄存器	0x01C0	R/ WTC/ SS
AXI_PERF_MON_SEL_AG_INT_ST_REG	中断状态寄存器	0x01C4	RO
AXI_PERF_MON_SEL_AG_INT_ENA_REG	中断使能寄存器	0x01C8	R/W
AXI_PERF_MON_SEL_AG_INT_CLR_REG	中断清除寄存器	0x01CC	WT

23.9 寄存器

本小节的所有地址均为相对于 AXI 性能监测器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 23.1. AXI_PERF_MON_CLK_EN_REG (0x0000)



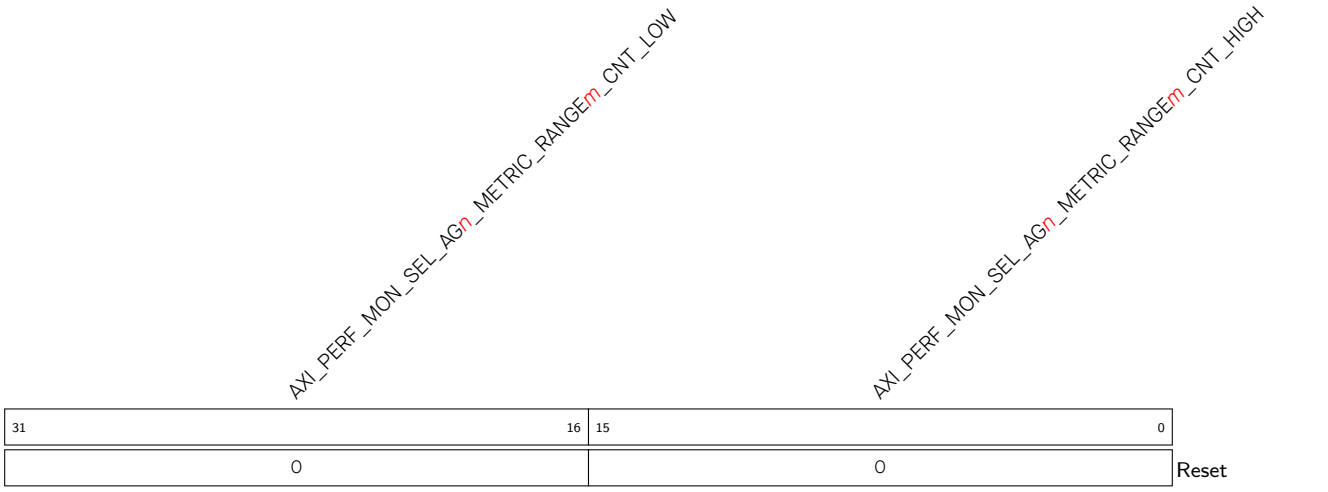
AXI_PERF_MON_CLK_EN 配置是否强制使能 AXI 性能监测器寄存器模块时钟。

0: AXI 性能监测器寄存器模块时钟自动开启和关闭

1: 强制使能 AXI 性能监测器寄存器模块时钟

(R/W)

Register 23.2. AXI_PERF_MON_SEL_AG*n*_METRIC_RANGE*m*_REG (*n*: 0-1, *m*: 0-5) (0x00C8+0x4**n*+0x8**m*)



AXI_PERF_MON_SEL_AG*n*_METRIC_RANGE*m*_CNT_HIGH (*n*: 0-1, *m*: 0-5) 配置选定监测单元中选定指标的区间统计范围上限。(R/W)

AXI_PERF_MON_SEL_AG*n*_METRIC_RANGE*m*_CNT_LOW (*n*: 0-1, *m*: 0-5) 配置选定监测单元中选定指标的区间统计范围下限。(R/W)

Register 23.3. AXI_PERF_MON_SEL_AG*n*_RD_ADDR_MASK*m*_REG (*n*: 0-1, *m*: 0-2) (0x00F8+0x4*n*+0x8*m*)



AXI_PERF_MON_SEL_AG*n*_RD_ADDR_MASK*m* (*n*: 0-1, *m*: 0-2) 配置选定监测单元地址过滤功能的读地址掩码。被掩码的位不参与比较。

仅在 AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER_EN 为 1 时有效。

(R/W)

Register 23.4. AXI_PERF_MON_SEL_AG*n*_WR_ADDR_MASK*m*_REG (*n*: 0-1, *m*: 0-2) (0x0110+0x4*n*+0x8*m*)

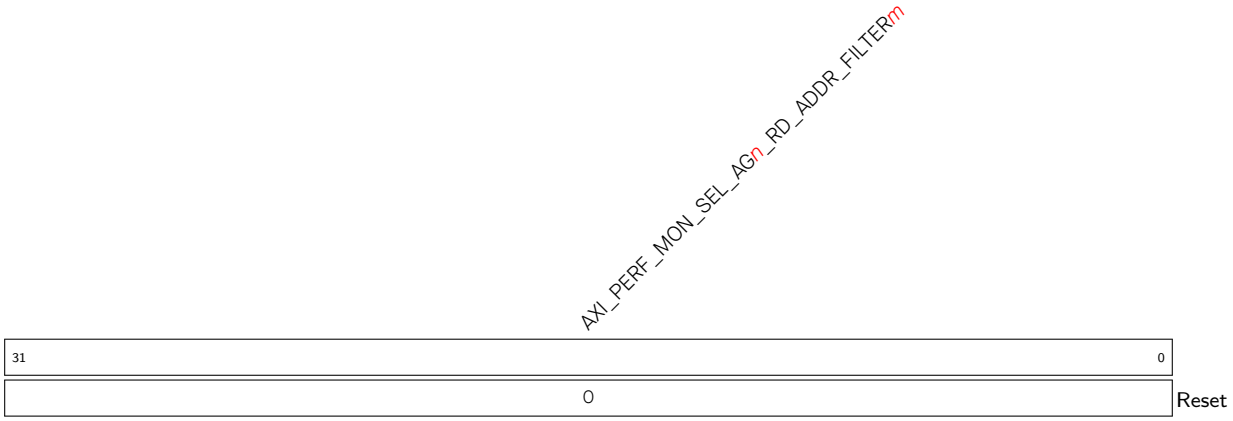


AXI_PERF_MON_SEL_AG*n*_WR_ADDR_MASK*m* (*n*: 0-1, *m*: 0-2) 配置选定监测单元地址过滤功能的写地址掩码。被掩码的位不参与比较。

仅在 AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER_EN 为 1 时有效。

(R/W)

Register 23.5. AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER*m*_REG (*n*: 0-1, *m*: 0-2) (0x0128+0x4*n*+0x8*m*)



AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER*m* (*n*: 0-1, *m*: 0-2) 配置选定监测单元的读地址过滤值。未被掩码的位会与该值进行比较。

仅在 AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER_EN 为 1 时有效。

(R/W)

Register 23.6. AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER*m*_REG (*n*: 0-1, *m*: 0-2) (0x0140+0x4*n*+0x8*m*)



AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER*m* (*n*: 0-1, *m*: 0-2) 配置选定监测单元的写地址过滤值。未被掩码的位会与该值进行比较。

仅在 AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER_EN 为 1 时有效。

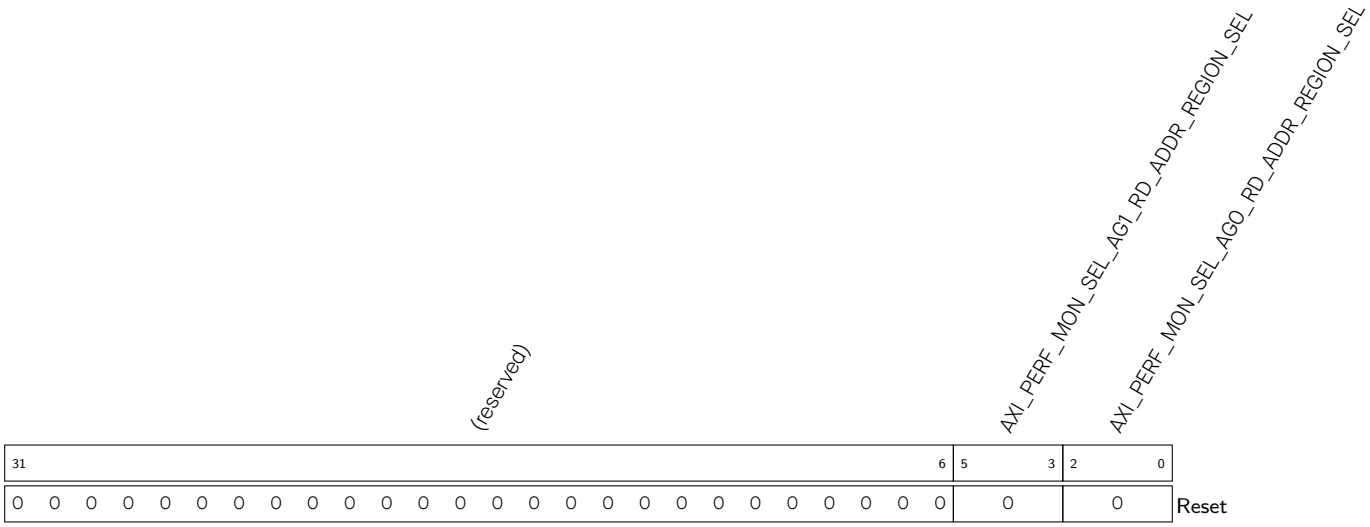
(R/W)

Register 23.7. AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m*_REG (*n*: 0-1, *m*: 1-2) (0x0158+0x4*n*+0x8*(*m*-1))



AXI_PERF_MON_SEL_AG*n*_METRIC_SELECT*m* (*n*: 0-1, *m*: 1-2) 配置选定监测单元的指标选择。每个计数器占用 8 位。在每组 8 位中，[4:0] 选择测量指标，[7:5] 选择区间统计指标。计数器可同时进行指标测量与区间统计。(R/W)

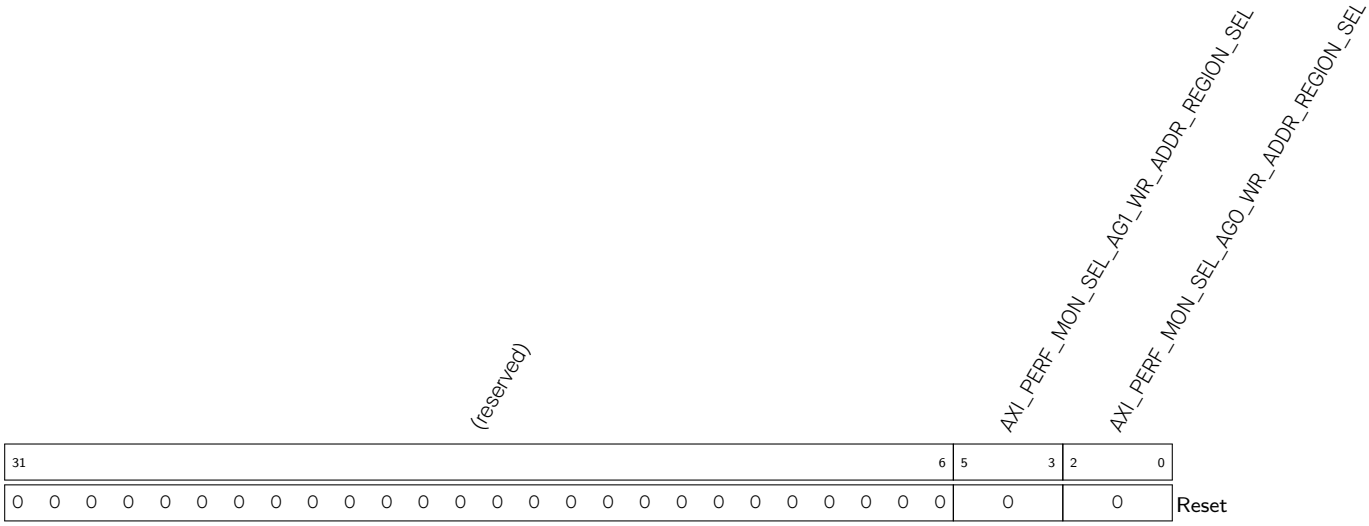
Register 23.8. AXI_PERF_MON_SEL_AG_RD_ADDR_REGION_SEL_REG (0x0168)



AXI_PERF_MON_SEL_AG0_RD_ADDR_REGION_SEL 配置监测单元 0 的读 region 选择。与掩码和过滤值配合使用。仅统计地址位于所选 region 内的事务。(R/W)

AXI_PERF_MON_SEL_AG1_RD_ADDR_REGION_SEL 配置监测单元 1 的读 region 选择。与掩码和过滤值配合使用。仅统计地址位于所选 region 内的事务。(R/W)

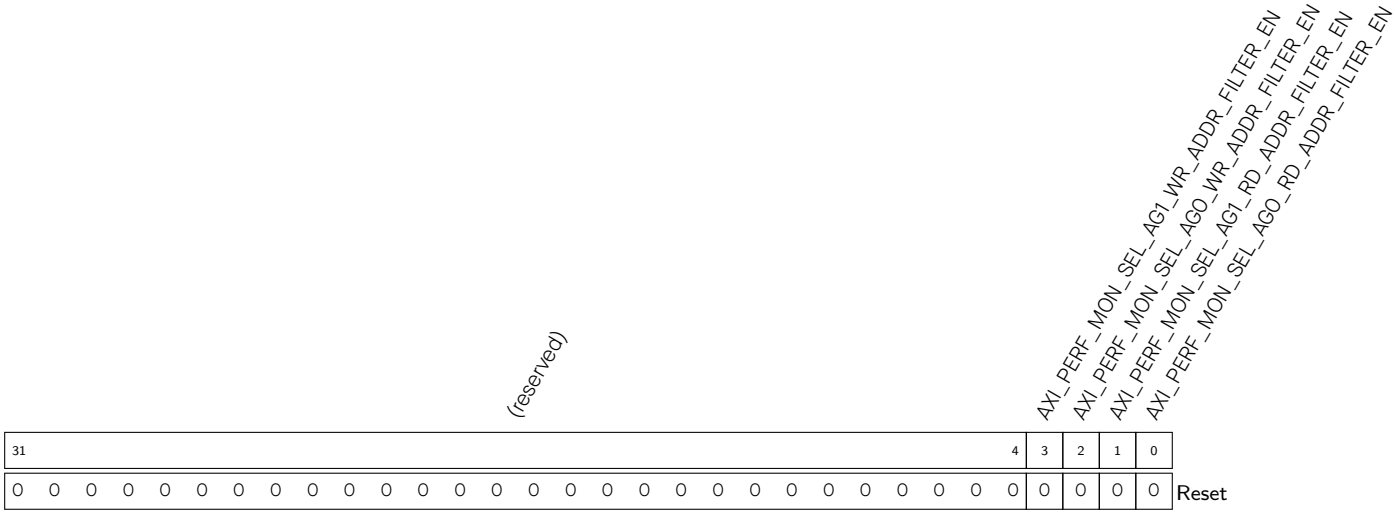
Register 23.9. AXI_PERF_MON_SEL_AG_WR_ADDR_REGION_SEL_REG (0x016C)



AXI_PERF_MON_SEL_AG0_WR_ADDR_REGION_SEL 配置监测单元 0 的写 region 选择。与掩码和过滤值配合使用。仅统计地址位于所选 region 内的事务。(R/W)

AXI_PERF_MON_SEL_AG1_WR_ADDR_REGION_SEL 配置监测单元 1 的写 region 选择。与掩码和过滤值配合使用。仅统计地址位于所选 region 内的事务。(R/W)

Register 23.10. AXI_PERF_MON_SEL_AG_ADDR_FILTER_EN_REG (0x0170)



AXI_PERF_MON_SEL_AG*n*_RD_ADDR_FILTER_EN (*n*: 0-1) 配置是否使能读地址过滤功能。

0: 禁用。记录所有读地址访问

1: 使能

(R/W)

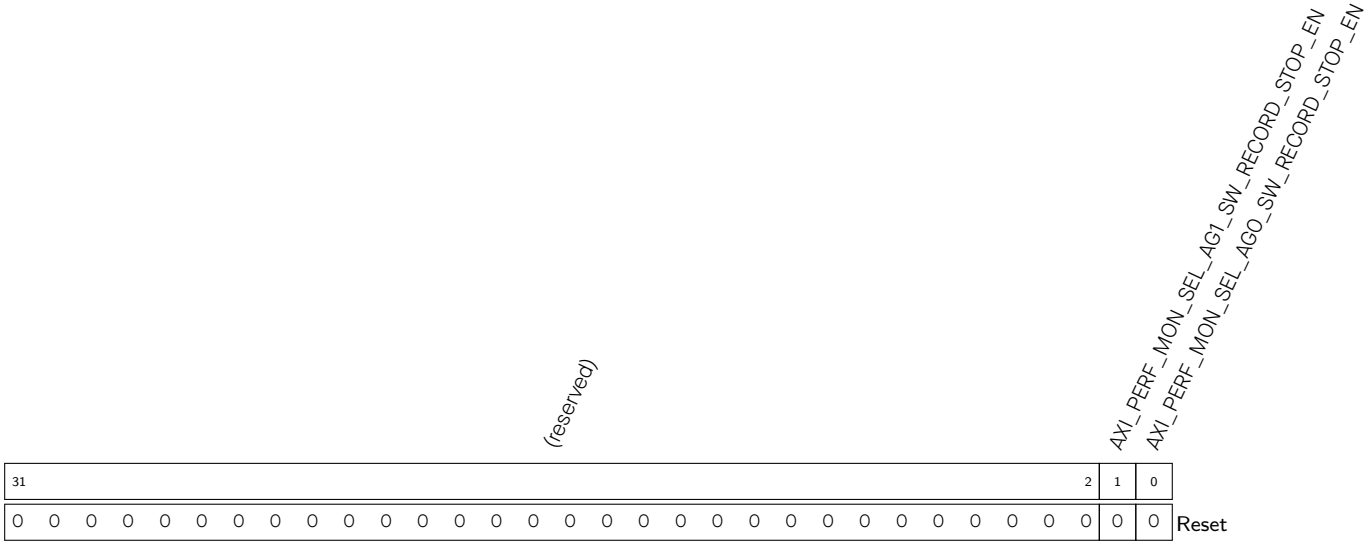
AXI_PERF_MON_SEL_AG*n*_WR_ADDR_FILTER_EN (*n*: 0-1) 配置是否使能写地址过滤功能。

0: 禁用。记录所有写地址访问

1: 使能

(R/W)

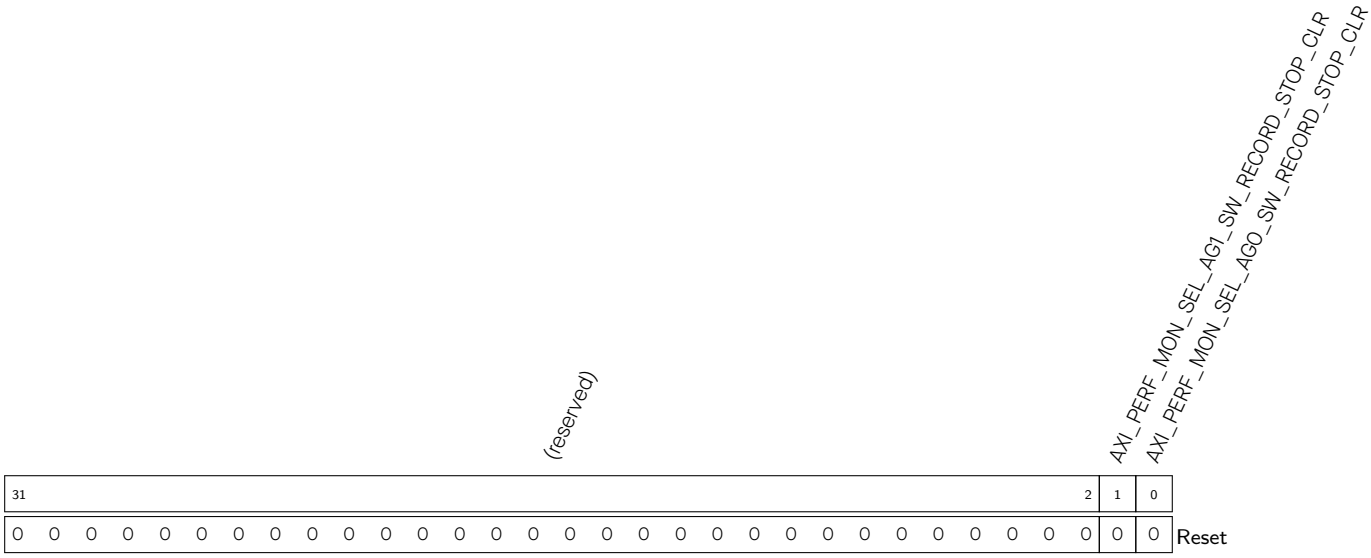
Register 23.11. AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_EN_REG (0x0174)



AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_EN 写 1 暂停事件日志功能。已记录的信息会被保留。(WT)

AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_EN 写 1 暂停事件日志功能。已记录的信息会被保留。(WT)

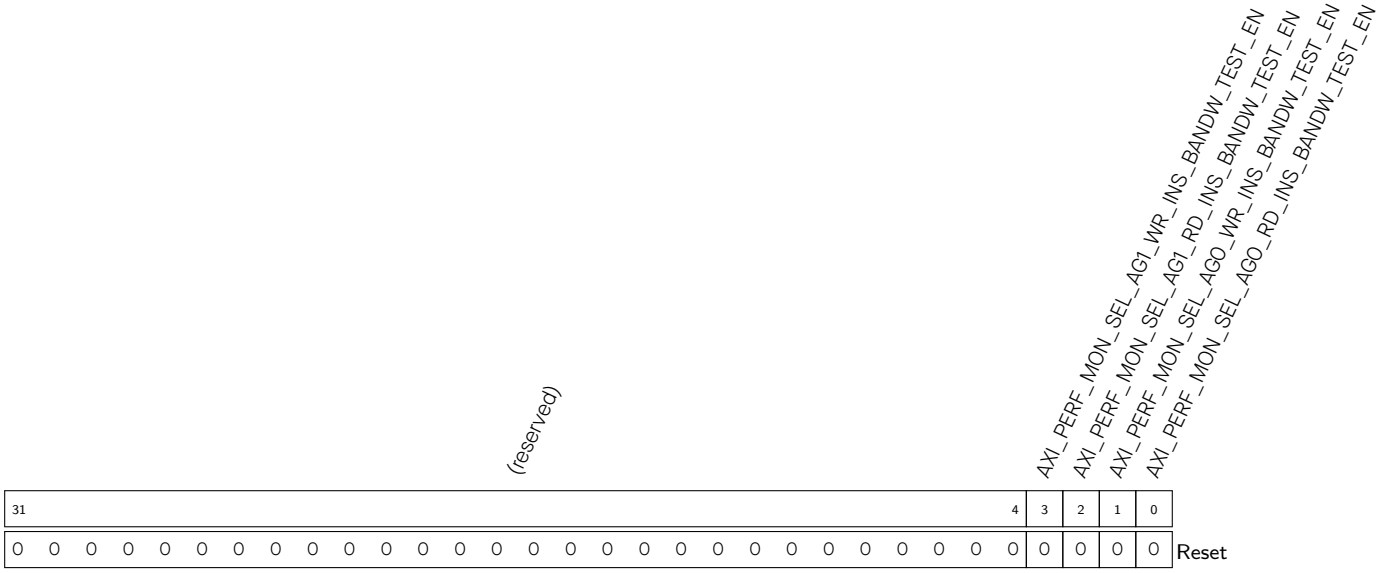
Register 23.12. AXI_PERF_MON_SEL_AG_SW_RECORD_STOP_CLR_REG (0x0178)



AXI_PERF_MON_SEL_AG0_SW_RECORD_STOP_CLR 写 1 清除事件日志暂停状态。从现在开始记录新的事务。(WT)

AXI_PERF_MON_SEL_AG1_SW_RECORD_STOP_CLR 写 1 清除事件日志暂停状态。从现在开始记录新的事务。(WT)

Register 23.13. AXI_PERF_MON_SEL_AG_INS_BANDW_TEST_EN_REG (0x017C)



AXI_PERF_MON_SEL_AG0_RD_INS_BANDW_TEST_EN 配置是否使能监测单元 0 的读瞬时带宽测试。

0: 禁用

1: 使能

(R/W)

AXI_PERF_MON_SEL_AG0_WR_INS_BANDW_TEST_EN 配置是否使能监测单元 0 的写瞬时带宽测试。

0: 禁用

1: 使能

(R/W)

AXI_PERF_MON_SEL_AG1_RD_INS_BANDW_TEST_EN 配置是否使能监测单元 1 的读瞬时带宽测试。

0: 禁用

1: 使能

(R/W)

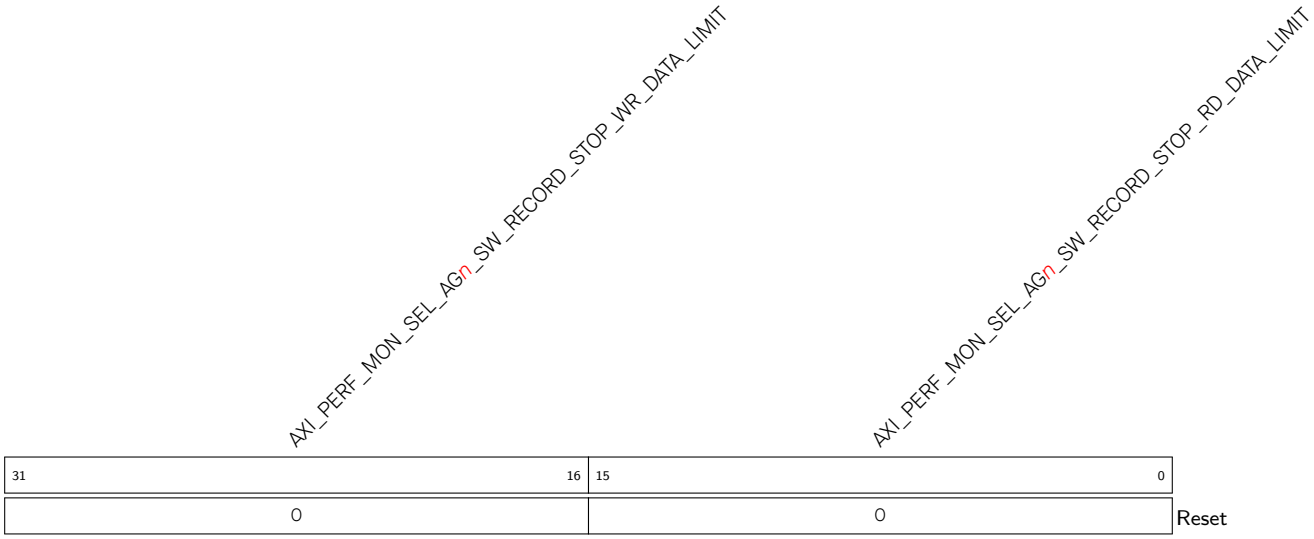
AXI_PERF_MON_SEL_AG1_WR_INS_BANDW_TEST_EN 配置是否使能监测单元 1 的写瞬时带宽测试。

0: 禁用

1: 使能

(R/W)

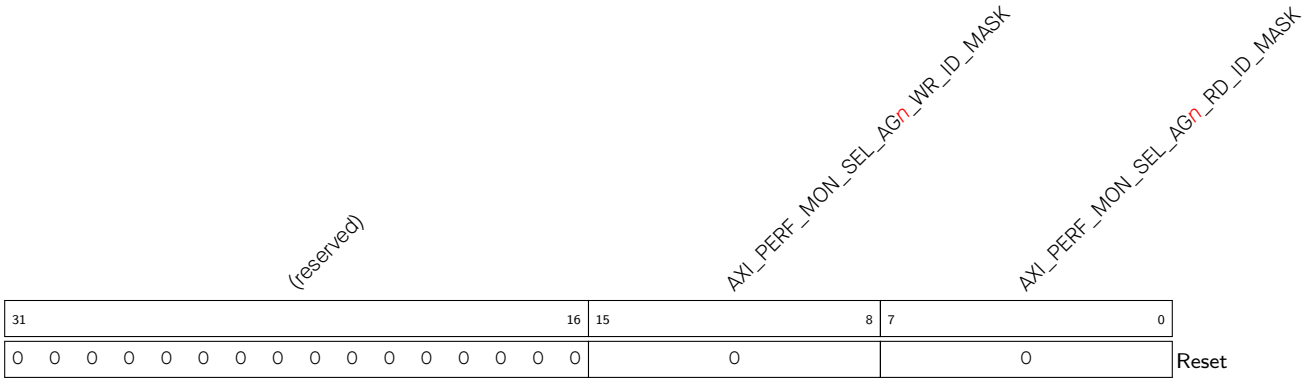
Register 23.14. AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_DATA_LIMIT_REG (*n*: 0-1) (0x0180+0x4*n*)



AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_RD_DATA_LIMIT (*n*: 0-1) 配置读瞬时带宽测试数据下限。当测量数据达到该下限时，将使能中断。(R/W)

AXI_PERF_MON_SEL_AG*n*_SW_RECORD_STOP_WR_DATA_LIMIT (*n*: 0-1) 配置写瞬时带宽测试数据下限。当测量数据达到该下限时，将使能中断。(R/W)

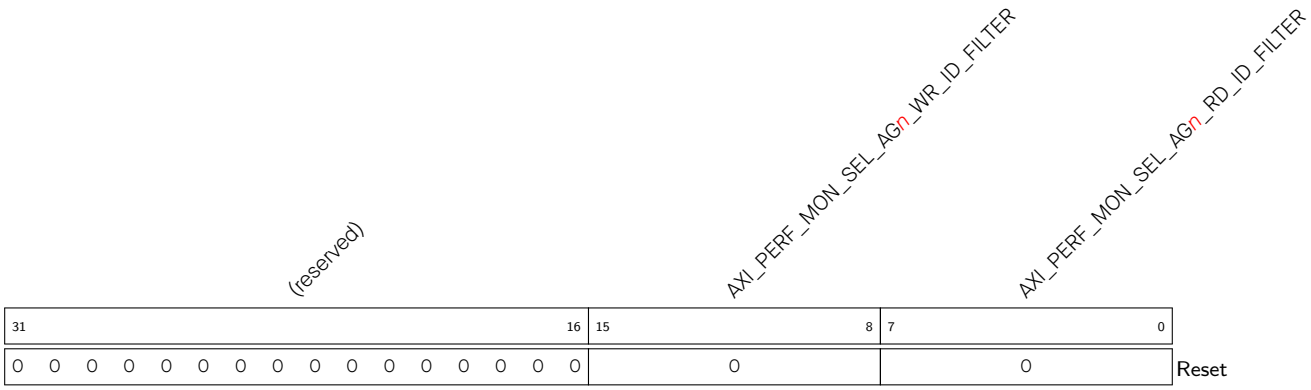
Register 23.15. AXI_PERF_MON_SEL_AG*n*_ID_MASK_REG (*n*: 0-1) (0x0188+0x4*n*)



AXI_PERF_MON_SEL_AG*n*_RD_ID_MASK (*n*: 0-1) 配置读 ID 掩码。被掩码的位不参与比较。(R/W)

AXI_PERF_MON_SEL_AG*n*_WR_ID_MASK (*n*: 0-1) 配置写 ID 掩码。被掩码的位不参与比较。(R/W)

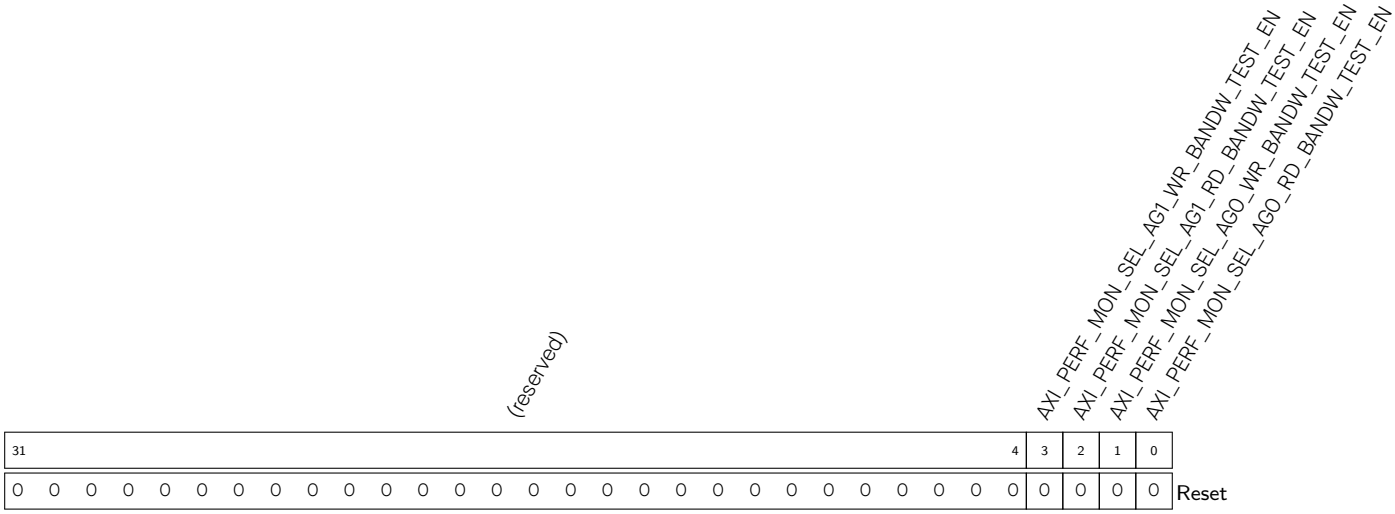
Register 23.16. AXI_PERF_MON_SEL_AG*n*_ID_FILTER_REG (*n*: 0-1) (0x0190+0x4**n*)



AXI_PERF_MON_SEL_AG*n*_RD_ID_FILTER (*n*: 0-1) 配置读 ID 过滤值。未被掩码的位必须与该值匹配才能通过过滤。(R/W)

AXI_PERF_MON_SEL_AG*n*_WR_ID_FILTER (*n*: 0-1) 配置写 ID 过滤值。未被掩码的位必须与该值匹配才能通过过滤。(R/W)

Register 23.17. AXI_PERF_MON_SEL_AG_BANDW_TEST_EN_REG (0x0198)



AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_EN 配置是否使能监测单元 0 的读平均带宽测试。

0: 禁用

1: 使能

(R/W)

AXI_PERF_MON_SEL_AG0_WR_BANDW_TEST_EN 配置是否使能监测单元 0 的写平均带宽测试。

0: 禁用

1: 使能

(R/W)

AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_EN 配置是否使能监测单元 1 的读平均带宽测试。

0: 禁用

1: 使能

(R/W)

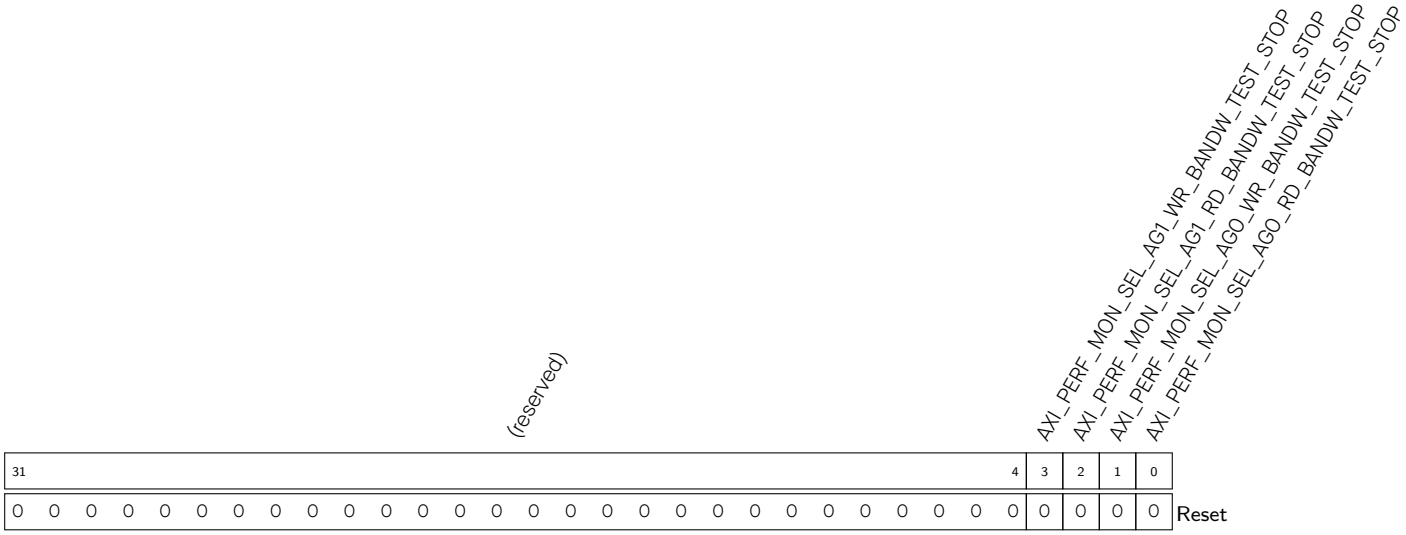
AXI_PERF_MON_SEL_AG1_WR_BANDW_TEST_EN 配置是否使能监测单元 1 的写平均带宽测试。

0: 禁用

1: 使能

(R/W)

Register 23.18. AXI_PERF_MON_SEL_AG_BANDW_TEST_STOP_REG (0x019C)



AXI_PERF_MON_SEL_AG0_RD_BANDW_TEST_STOP 配置是否停止监测单元 0 的读平均带宽测试。

0: 继续

1: 停止

(R/W)

AXI_PERF_MON_SEL_AG0_WR_BANDW_TEST_STOP 配置是否停止监测单元 0 的写平均带宽测试。

0: 继续

1: 停止

(R/W)

AXI_PERF_MON_SEL_AG1_RD_BANDW_TEST_STOP 配置是否停止监测单元 1 的读平均带宽测试。

0: 继续

1: 停止

(R/W)

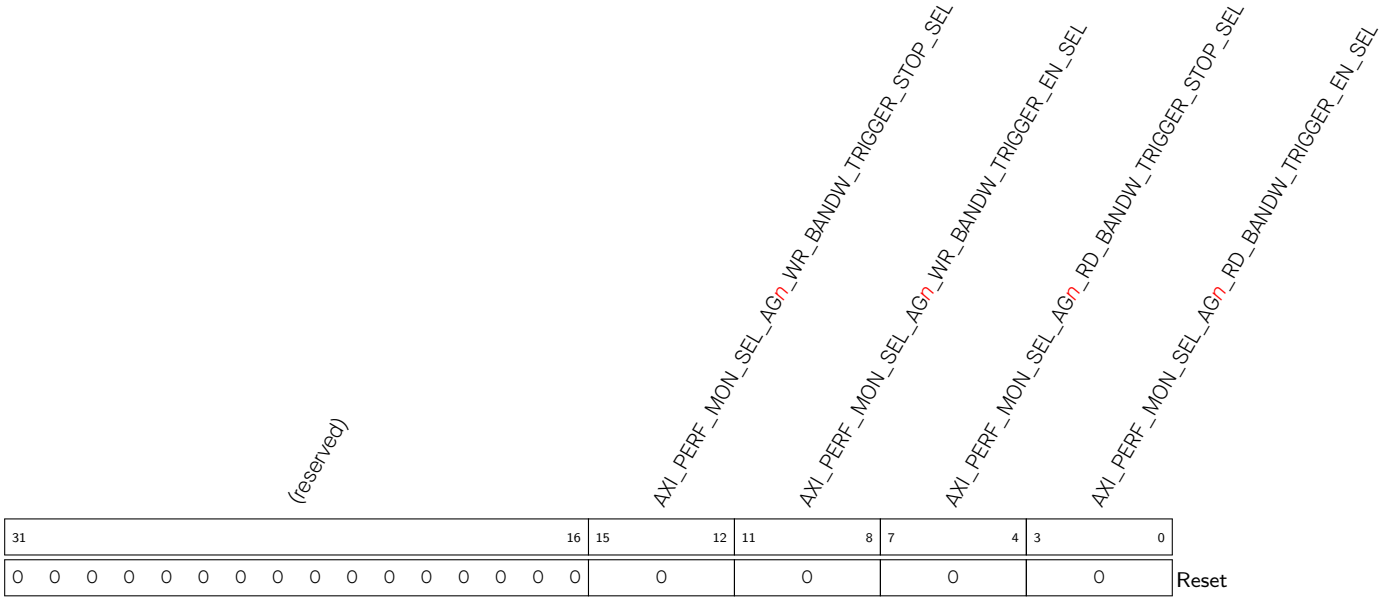
AXI_PERF_MON_SEL_AG1_WR_BANDW_TEST_STOP 配置是否停止监测单元 1 的写平均带宽测试。

0: 继续

1: 停止

(R/W)

Register 23.19. AXI_PERF_MON_SEL_AG*n*_BANDW_TRIGGER_IN_SEL_REG (*n*: 0-1) (0x01A0+0x4*n*)



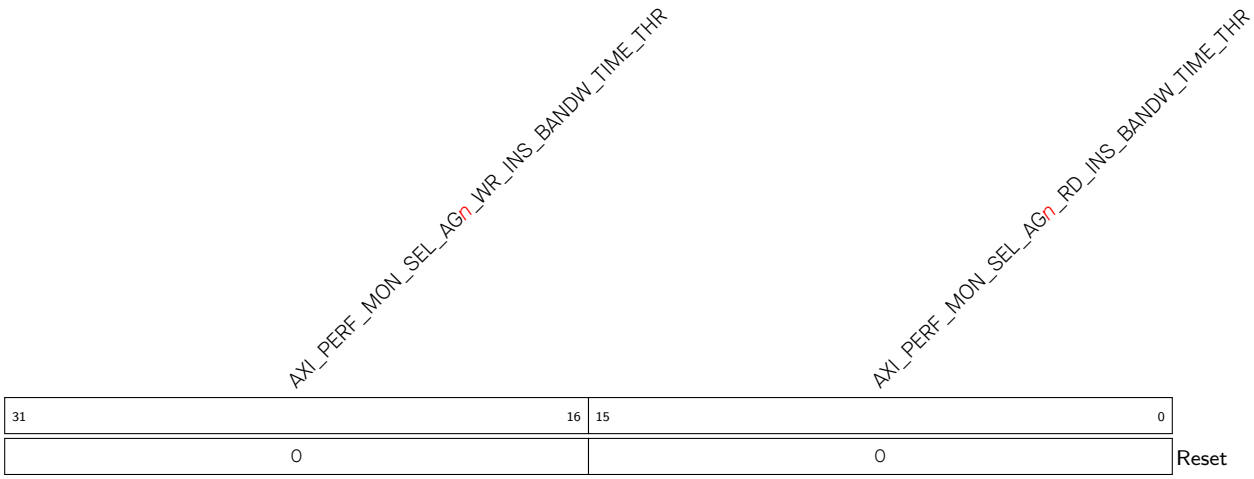
AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TRIGGER_EN_SEL (*n*: 0-1) 配置读平均带宽测试的触发使能源选择。触发使能可以来自 SoC 或软件配置。(R/W)

AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TRIGGER_STOP_SEL (*n*: 0-1) 配置读平均带宽测试的触发停止源选择。触发停止可以来自 SoC 或软件配置。(R/W)

AXI_PERF_MON_SEL_AG*n*_WR_BANDW_TRIGGER_EN_SEL (*n*: 0-1) 配置写平均带宽测试的触发使能源选择。触发使能可以来自 SoC 或软件配置。(R/W)

AXI_PERF_MON_SEL_AG*n*_WR_BANDW_TRIGGER_STOP_SEL (*n*: 0-1) 配置写平均带宽测试的触发停止源选择。触发停止可以来自 SoC 或软件配置。(R/W)

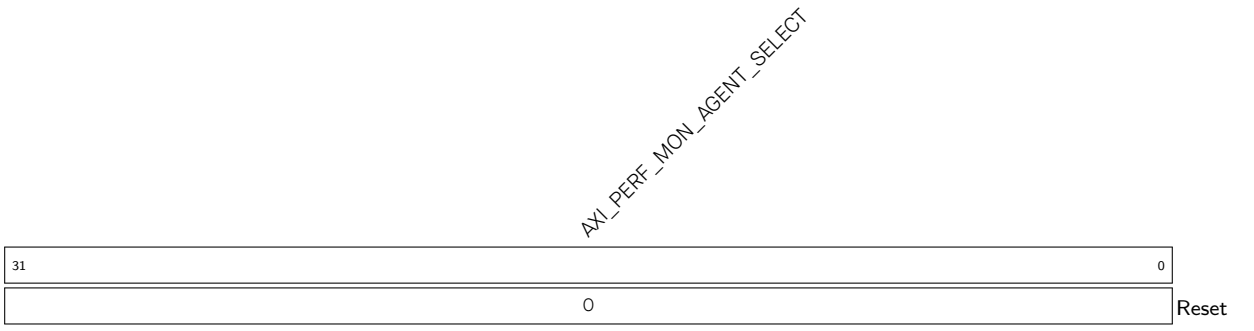
Register 23.20. AXI_PERF_MON_SEL_AG*n*_INS_BANDW_TIME_THR_REG (*n*: 0-1) (0x01B8+0x4**n*)



AXI_PERF_MON_SEL_AG*n*_RD_INS_BANDW_TIME_THR (*n*: 0-1) 配置读瞬时带宽测试的时间阈值。计数器在该时间窗口内记录数据量，然后刷新重新计数。(R/W)

AXI_PERF_MON_SEL_AG*n*_WR_INS_BANDW_TIME_THR (*n*: 0-1) 配置写瞬时带宽测试的时间阈值。计数器在该时间窗口内记录数据量，然后刷新重新计数。(R/W)

Register 23.21. AXI_PERF_MON_AGENT_SELECT_REG (0x0004)



AXI_PERF_MON_AGENT_SELECT 配置每个监测单元对应的 agent 选择。每个 agent 选择占 5 位。

- Bits [4:0]：AXI Perf Mon 监测单元 0 选择的待监测端口
- Bits [9:5]：AXI Perf Mon 监测单元 1 选择的待监测端口
- Bits [31:10]：保留

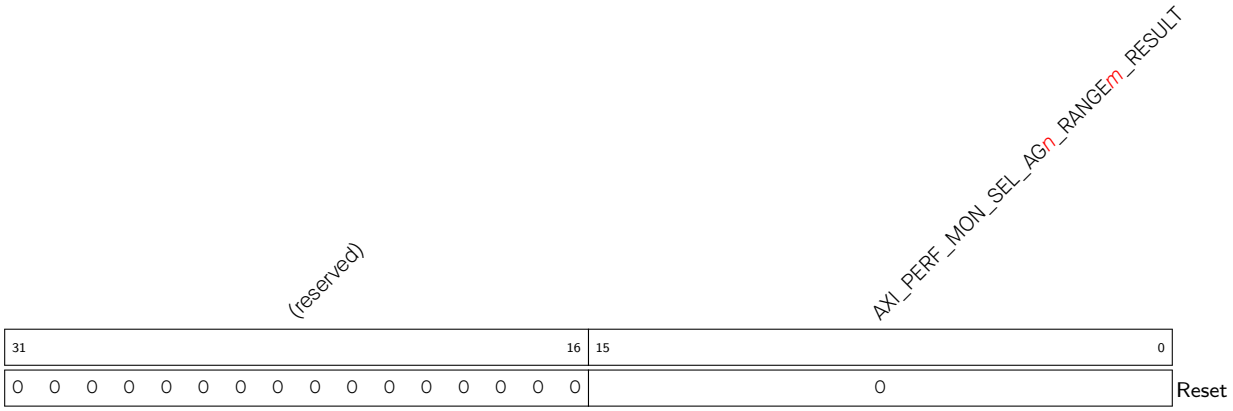
(R/W)

Register 23.22. AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_REG (*n*: 0-1, *m*: 0-5) (0x0008+0x4*n*+0x8*m*)



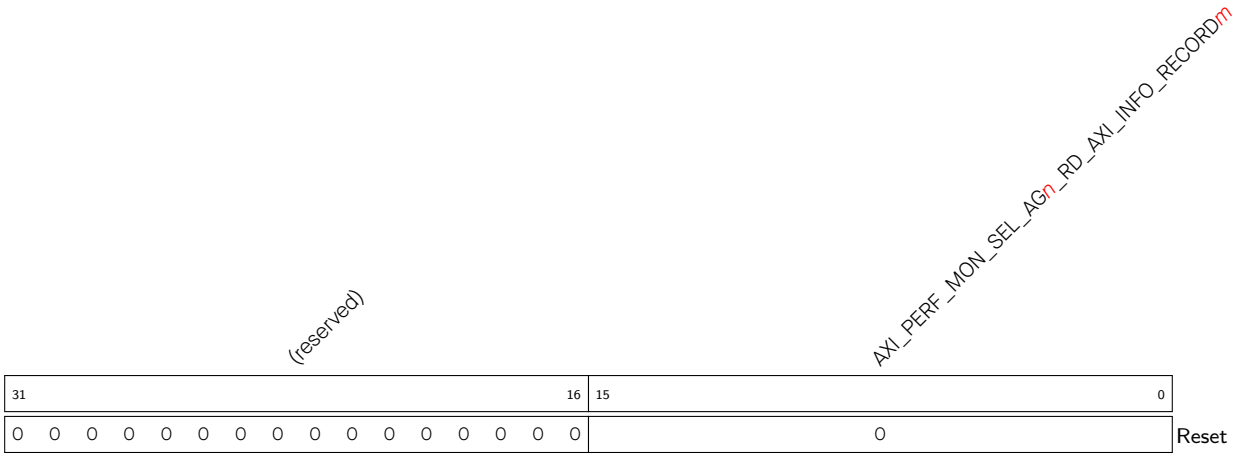
AXI_PERF_MON_SEL_AG*n*_COUNTER*m*_RESULT (*n*: 0-1, *m*: 0-5) 表示该计数器中选定监测单元的指标测量结果。[15:0] 为最小值，[31:16] 为最大值。(RO)

Register 23.23. AXI_PERF_MON_SEL_AG*n*_RANGE*m*_REG (*n*: 0-1, *m*: 0-5) (0x0038+0x4*n*+0x8*m*)



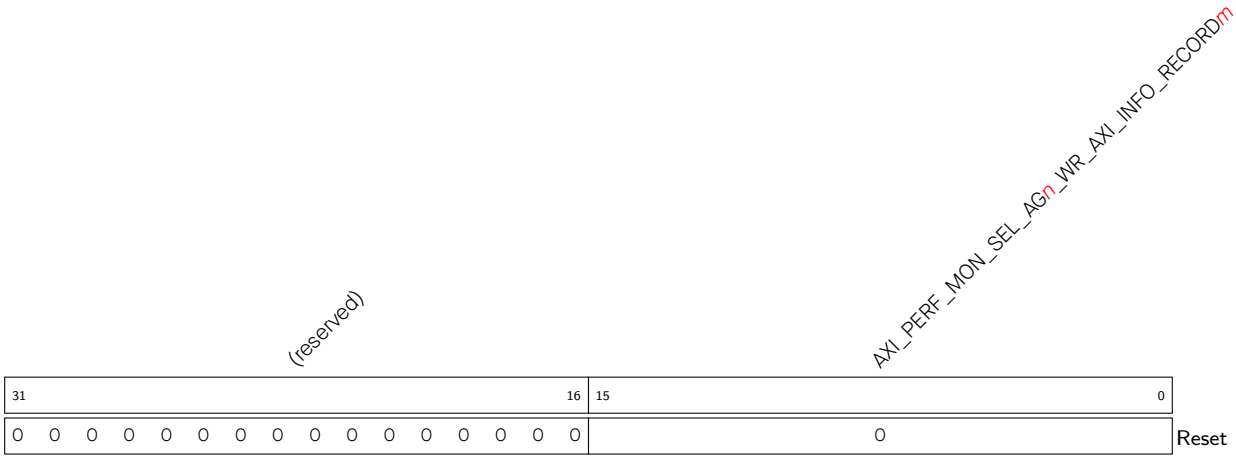
AXI_PERF_MON_SEL_AG*n*_RANGE*m*_RESULT (*n*: 0-1, *m*: 0-5) 表示该计数器中选定监测单元的区间统计结果。(RO)

Register 23.24. AXI_PERF_MON_SEL_AG*n*_RD_AXI_INFO_RECORD*m*_REG (*n*: 0-1, *m*: 0-3) (0x0068+0x4***n*+0x8***m*)



AXI_PERF_MON_SEL_AG*n*_RD_AXI_INFO_RECORD*m* (*n*: 0-1, *m*: 0-3) 表示选定监测单元的最新 AXI 读事务信息记录。[7:0] 为 ARLEN，[15:8] 为 ARID。(RO)

Register 23.25. AXI_PERF_MON_SEL_AG*n*_WR_AXI_INFO_RECORD*m*_REG (*n*: 0-1, *m*: 0-3) (0x0088+0x4***n*+0x8***m*)



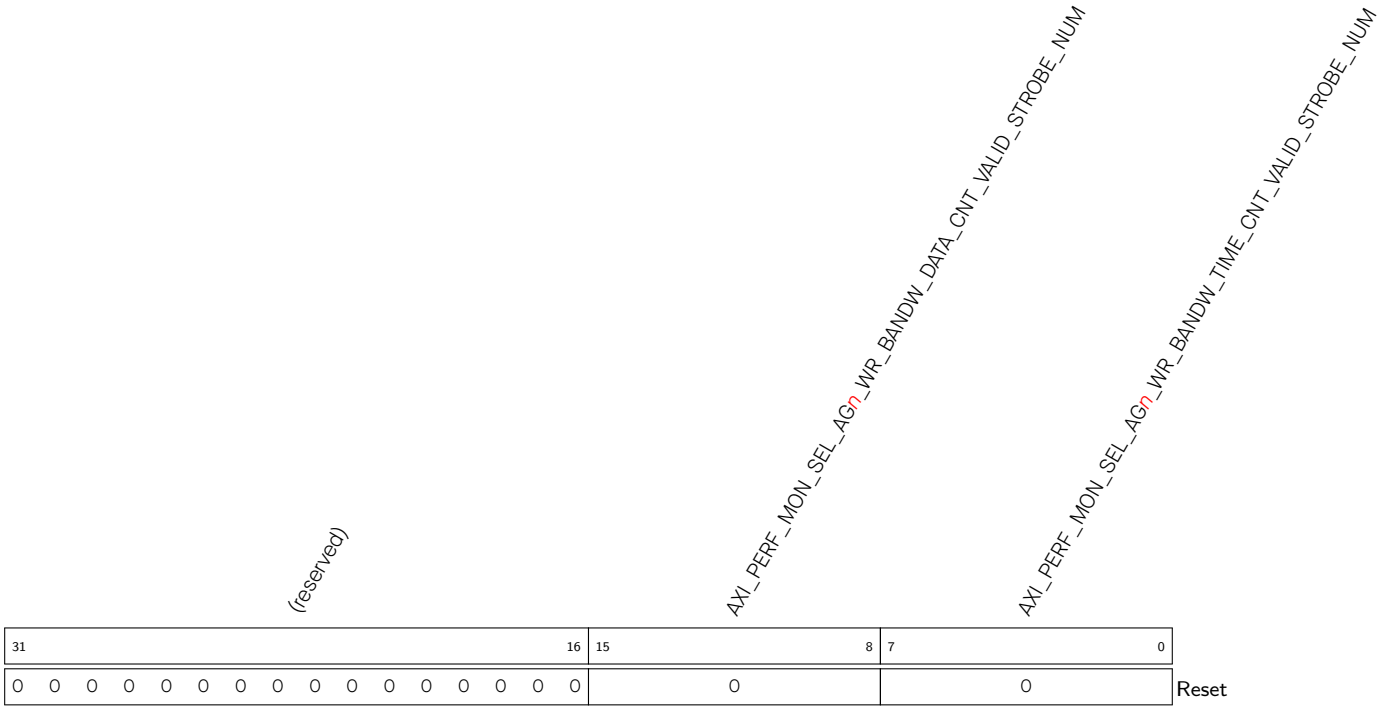
AXI_PERF_MON_SEL_AG*n*_WR_AXI_INFO_RECORD*m* (*n*: 0-1, *m*: 0-3) 表示选定监测单元的最新 AXI 写事务信息记录。[7:0] 为 AWLEN，[15:8] 为 AWID。(RO)

Register 23.26. AXI_PERF_MON_SEL_AG*n*_INS_BANDW_DATA*m*_REG (*n*: 0-1, *m*: 0-3) (0x00A8+0x4**n*+0x8**m*)



AXI_PERF_MON_SEL_AG*n*_INS_BANDW_DATA*m* (*n*: 0-1, *m*: 0-3) 表示选定监测单元在配置的时间窗口内记录的最新瞬时带宽统计数据量。[15:0] 为读，[31:16] 为写。(RO)

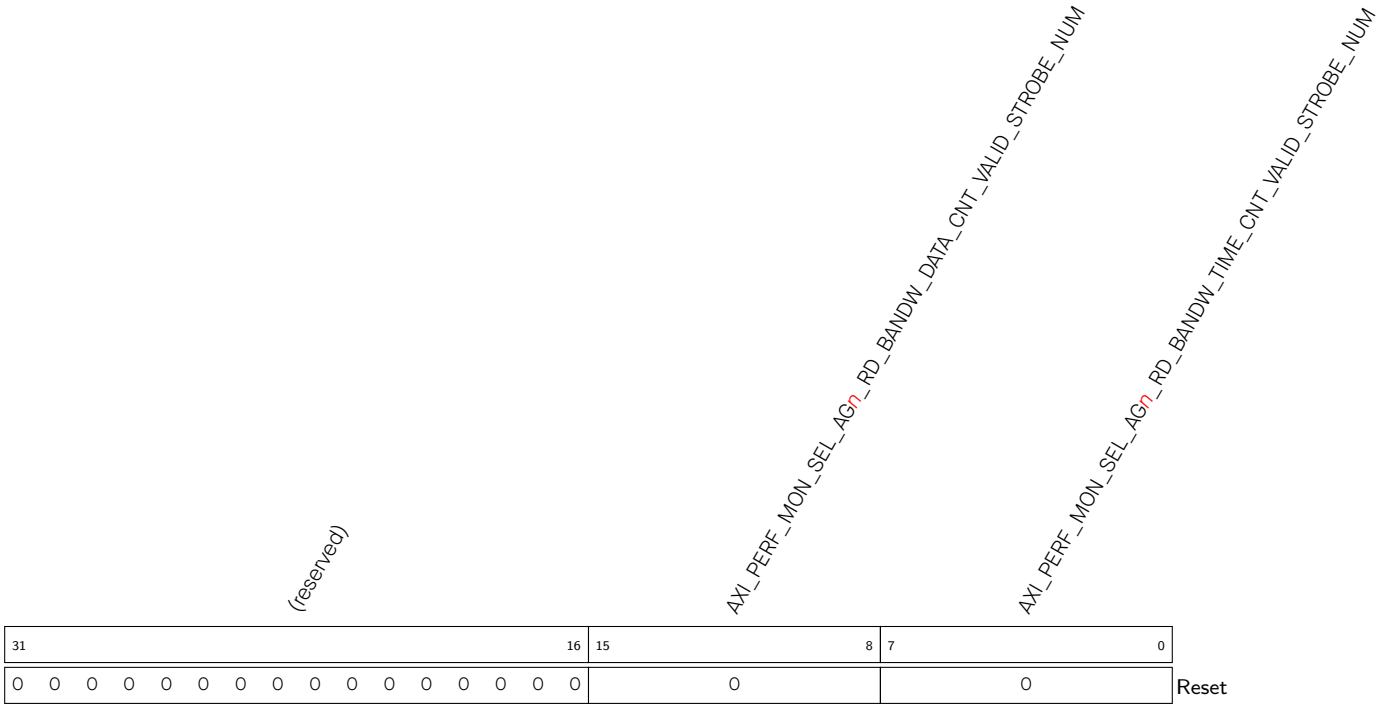
Register 23.27. AXI_PERF_MON_SEL_AG*n*_WR_BANDW_CNT_VALID_STROBE_NUM_REG (*n*: 0-1) (0x01A8+0x4**n*)



AXI_PERF_MON_SEL_AG*n*_WR_BANDW_TIME_CNT_VALID_STROBE_NUM (*n*: 0-1) 表示写带宽测试中时间计数器的缩放系数。(R/W)

AXI_PERF_MON_SEL_AG*n*_WR_BANDW_DATA_CNT_VALID_STROBE_NUM (*n*: 0-1) 表示写带宽测试中数据计数器的缩放系数。(R/W)

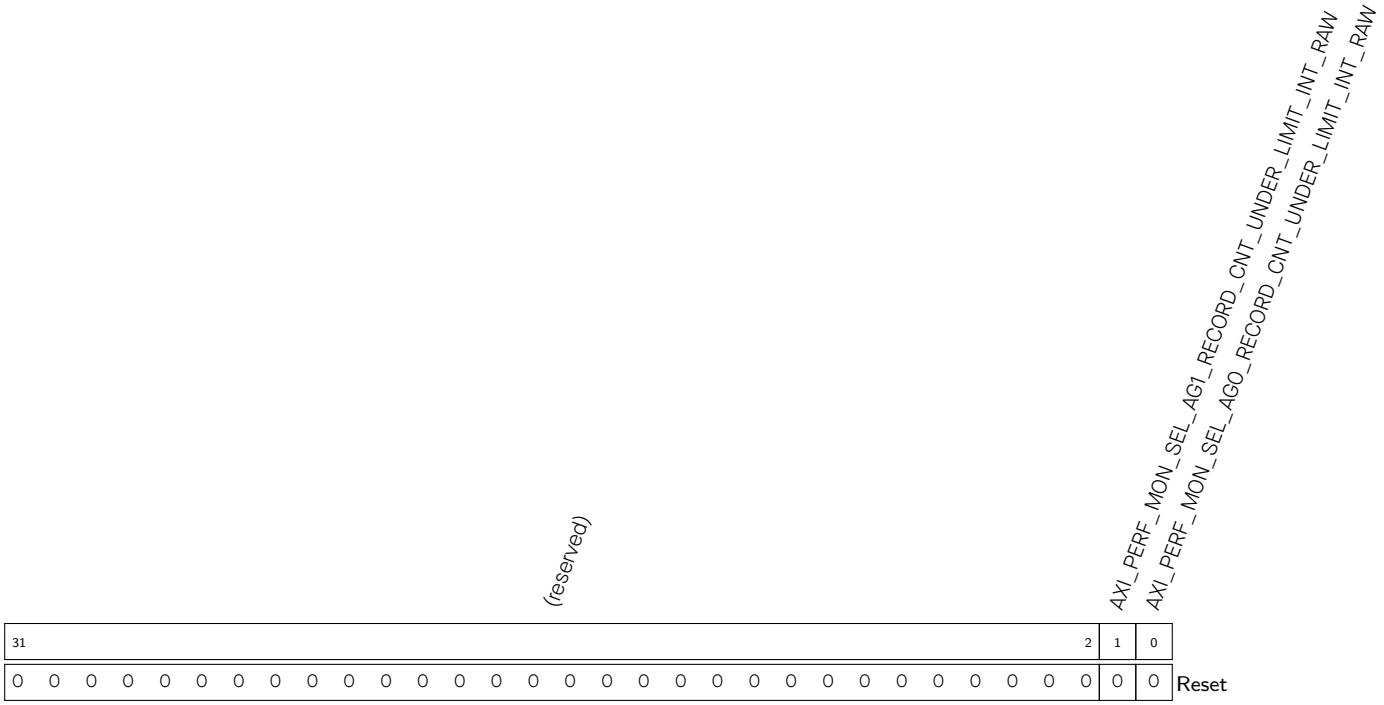
Register 23.28. AXI_PERF_MON_SEL_AG*n*_RD_BANDW_CNT_VALID_STROBE_NUM_REG (*n*: 0-1) (0x01B0+0x4**n*)



AXI_PERF_MON_SEL_AG*n*_RD_BANDW_TIME_CNT_VALID_STROBE_NUM (*n*: 0-1) 表示读带宽测试中时间计数器的缩放系数。(R/W)

AXI_PERF_MON_SEL_AG*n*_RD_BANDW_DATA_CNT_VALID_STROBE_NUM (*n*: 0-1) 表示读带宽测试中数据计数器的缩放系数。(R/W)

Register 23.29. AXI_PERF_MON_SEL_AG_INT_RAW_REG (0x01C0)



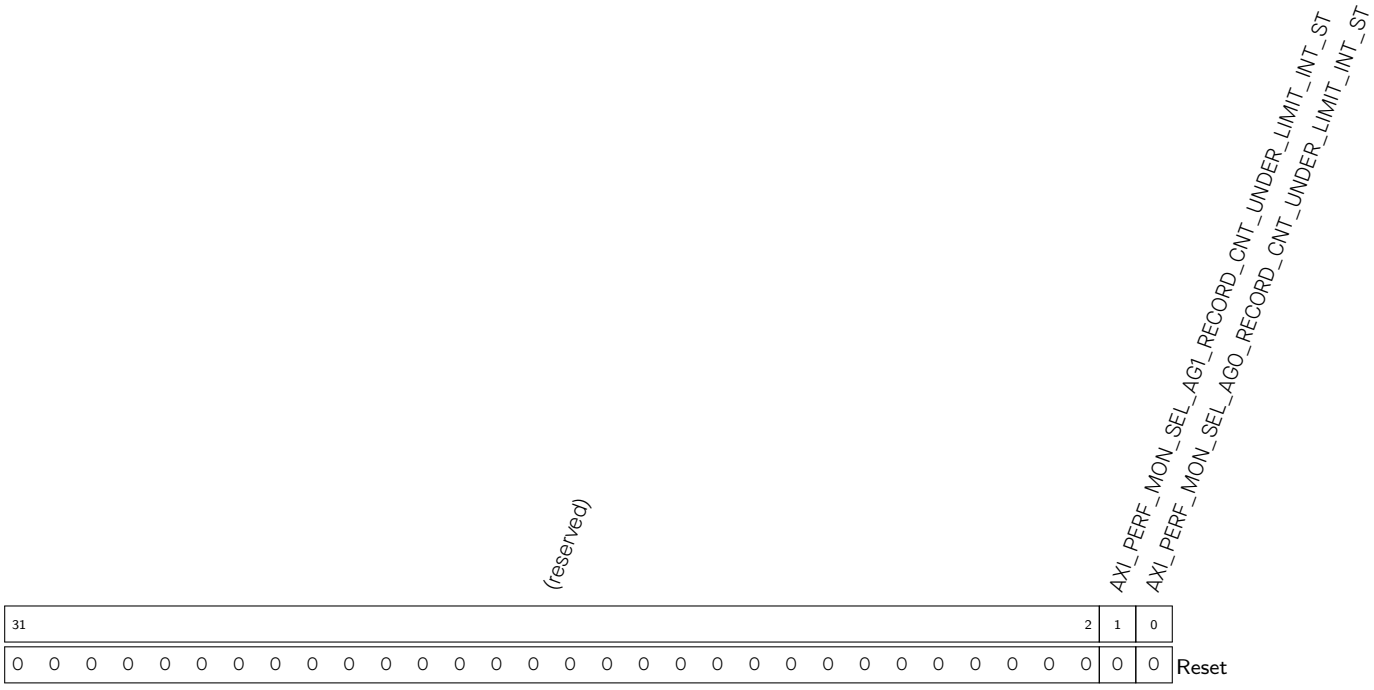
AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_RAW

AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT 的原始中断状态。(R/ WTC/ SS)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_RAW

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT 的原始中断状态。(R/ WTC/ SS)

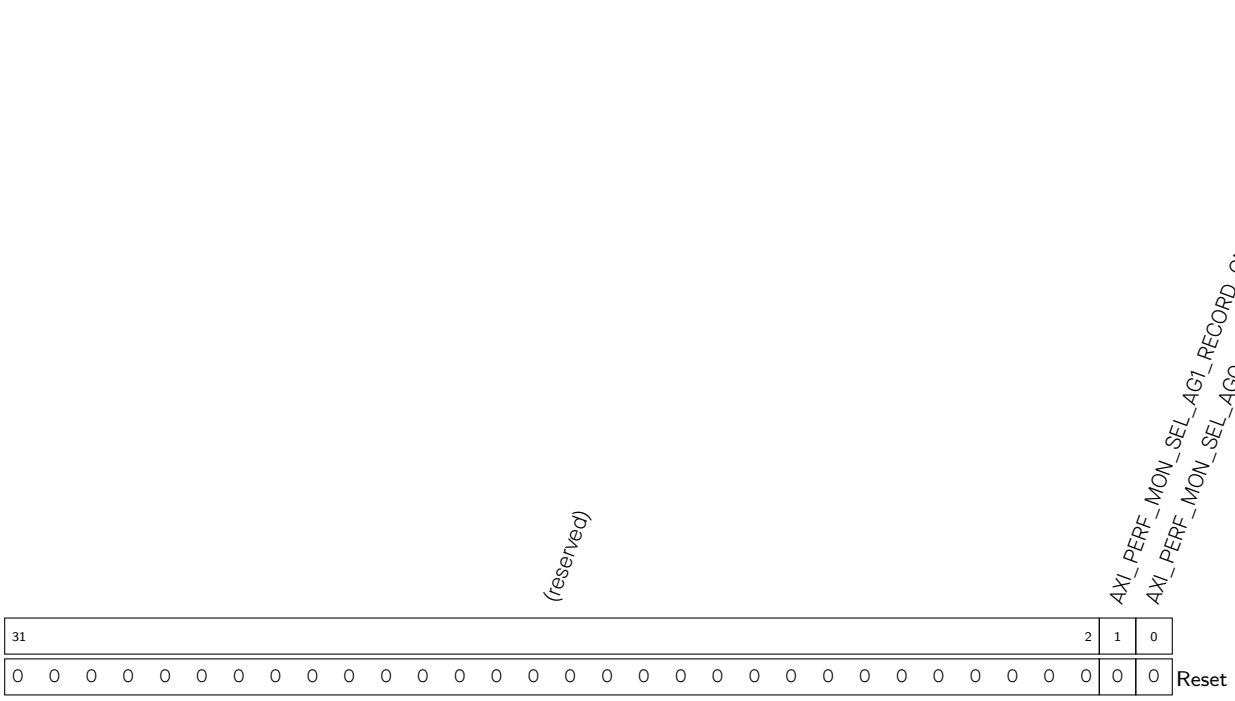
Register 23.30. AXI_PERF_MON_SEL_AG_INT_ST_REG (0x01C4)



AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_ST AXI_PERF_MON_SEL_AG0_-RECORD_CNT_UNDER_LIMIT_INT 的屏蔽中断状态。(RO)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_ST AXI_PERF_MON_SEL_AG1_-RECORD_CNT_UNDER_LIMIT_INT 的屏蔽中断状态。(RO)

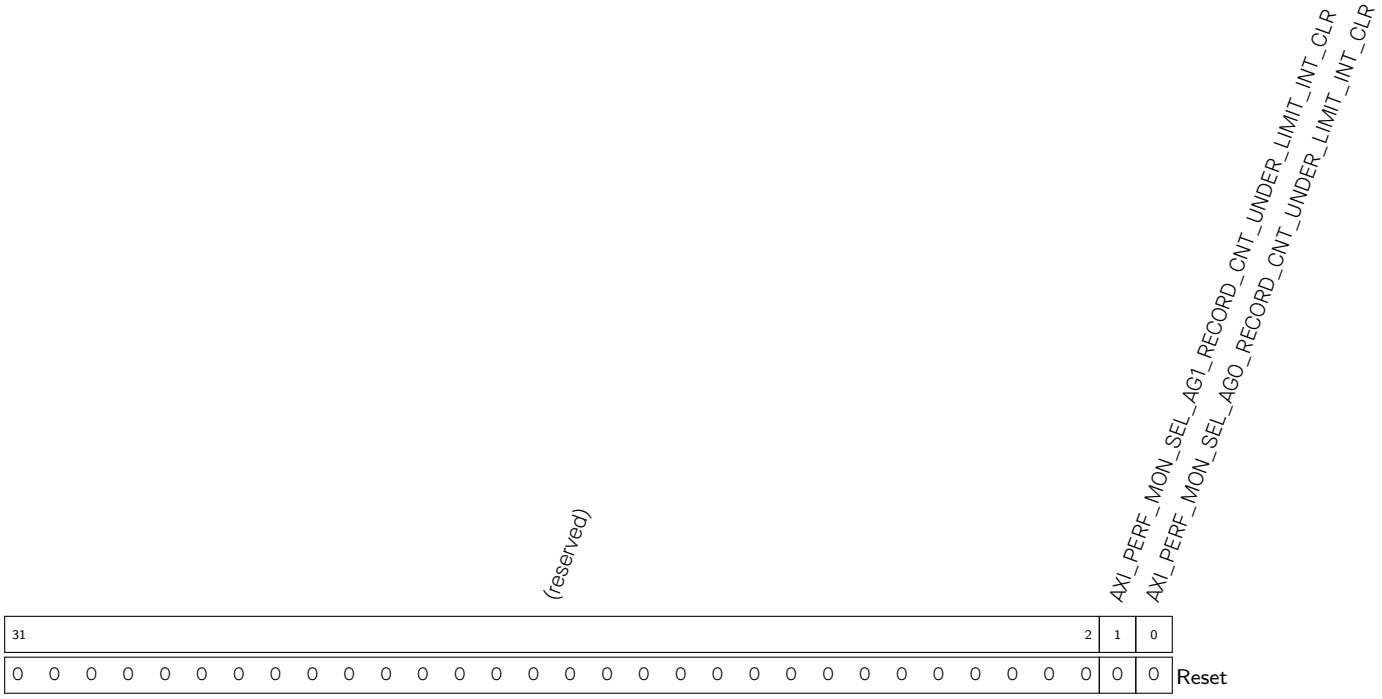
Register 23.31. AXI_PERF_MON_SEL_AG_INT_ENA_REG (0x01C8)



AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_ENA 写 1 使 能
AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT。(R/W)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_ENA 写 1 使 能
AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT。(R/W)

Register 23.32. AXI_PERF_MON_SEL_AG_INT_CLR_REG (0x01CC)



AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT_CLR 写 1 清除

AXI_PERF_MON_SEL_AG0_RECORD_CNT_UNDER_LIMIT_INT。(WT)

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT_CLR 写 1 清除

AXI_PERF_MON_SEL_AG1_RECORD_CNT_UNDER_LIMIT_INT。(WT)

第 24 章

LP 信箱控制器

24.1 概述

ESP32-P4 中包含一个 LP 信箱控制器模块，旨在通过硬件机制以实现 LP CPU 和 HP CUP0/1 之间高效的核间通信。LP 信箱控制器模块中包含 16 个 32 位信息寄存器可供 LP CPU 和 HP CPU0/1 存储并传递信息，并通过中断机制实现 LP CPU 和 HP CPU0/1 之间的核间通信。

24.2 特性

ESP32-P4 的 LP 信箱控制器模块具有以下特性：

- 支持多达 16 个 32 位信息寄存器用于核间通信
- 支持 LP CPU 外部中断信号 MB_LP_INTR
- 支持 HP CPU*x* (*x*: 0 ~ 1) 外部中断信号 MB_HP_INTR

24.3 功能描述

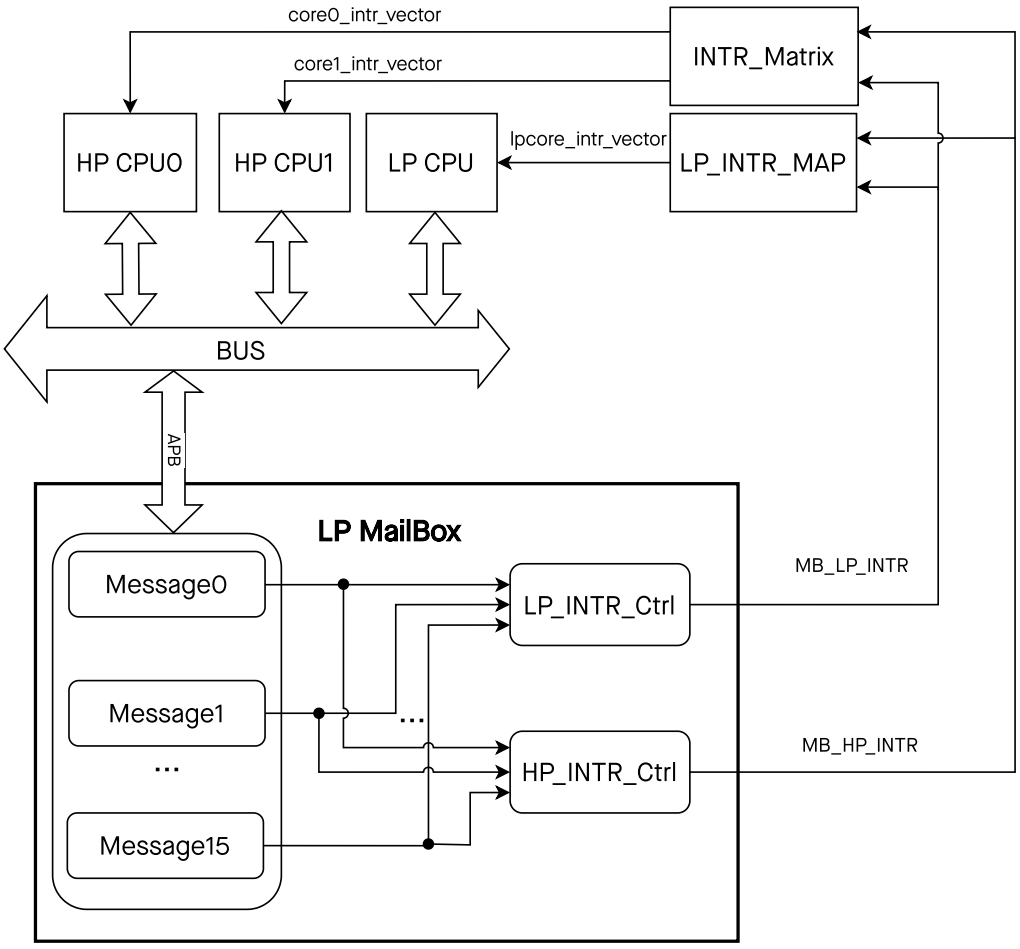


图 24.3-1. LP 信箱控制器系统框图

24.3.1 信息寄存器

LP 信箱控制器中包含了 MB_MESSAGE_0_REG ~ MB_MESSAGE_15_REG 共计 16 组 32 位信息寄存器，用于缓存核间通信信息。LP CPU 和 HP CPU0/1 均可读写这些信息寄存器以传递或获取信息。

读写信息寄存器通过 APB 总线来完成，使用 APB_CLK 时钟。该时钟为 LP 信箱控制器使用的唯一时钟，更多时钟信息，见章节 11 复位和时钟。

24.3.2 中断

ESP32-P4 的 LP 信箱控制器可以生成如下外部中断信号：

- MB_LP_INTR
- MB_HP_INTR

MB_LP_INTR 直接映射至 LP CPU（见章节 5 低功耗处理器），MB_HP_INTR 可通过中断矩阵映射至 HP CPU0/1。这些中断信号由 LP 信箱控制器模块的内部中断源生成。表 24.3-1 列出了 LP 信箱控制器的内部中断源、中断触发条件、以及生成的中断信号。

表 24.3-1. LP 信箱控制器的内部中断源

内部中断源	触发条件	中断信号
MB_HP_*m*_INT	写 MB_MESSAGE_*m*_REG	MB_HP_INTR
MB_LP_*n*_INT	写 MB_MESSAGE_*n*_REG	MB_LP_INTR

说明：

- *m* 和 *n* 可以是 0 ~ 15。
- MB_HP_INTR 同样可作为 LP CPU 的外部中断源，通过 LP 中断映射模块映射（LP_INTR_MAP，见图 24.3-1）至 LP CPU。有关 LP_INTR_MAP 的更多信息，见章节 5 低功耗处理器。
- MB_LP_INTR 同样可以作为 HP CPU*x* 的外部中断源，可通过中断矩阵将其映射至 HP CPU*x*。
- 当 LP 信箱控制器需要为 HP CPU0 和 HP CPU1 分配不同的中断源时，可通过中断矩阵将 MB_HP_INTR 和 MB_LP_INTR 分别映射至 HP CPU0 和 HP CPU1。

24.3.3 核间通信

当 LP CPU 与 HP CPU*x* 之间需要通信时，可以根据需求利用 LP 信箱控制器中部分或者全部信息寄存器来分享信息，并通过相应的中断来同步信息。在通信期间，需要软件保证仅发送方可写 LP 信箱控制器中的信息寄存器，防止信息寄存器中的数据被篡改。

- 以 HP CPU0 通过 MB_MESSAGE_*n*_REG 向 LP CPU 发起通信为例：
  - 置位 MB_LP_*n*_INT_ENA 使能 MB_LP_*n*_INT 中断。
  - HP CPU0 写 MB_MESSAGE_*n*_REG，缓存共享信息。
  - LP CPU 检测 MB_LP_INT_ST_REG 中断状态寄存器发现 MB_LP_*n*_ST 状态位置位。
  - LP CPU 读取 MB_MESSAGE_*n*_REG 获取共享信息。
- 以 LP CPU 向 HP CPU0 发起通信为例：
  - 置位 MB_HP_*m*_INT_ENA 使能 MB_HP_*m*_INT 中断。
  - 配置中断矩阵，将 MB_HP_INTR 映射至 HP CPU0（见章节 13 中断矩阵）。
  - LP CPU 写 MB_MESSAGE_*m*_REG，缓存共享信息。
  - HP CPU0 检测 MB_HP_INT_ST_REG 中断状态寄存器，发现 MB_HP_*m*_ST 状态位置位。
  - HP CPU0 读取 MB_MESSAGE_*m*_REG 获取共享信息。

24.4 寄存器列表

本小节的所有地址均为相对于 LP 信箱控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

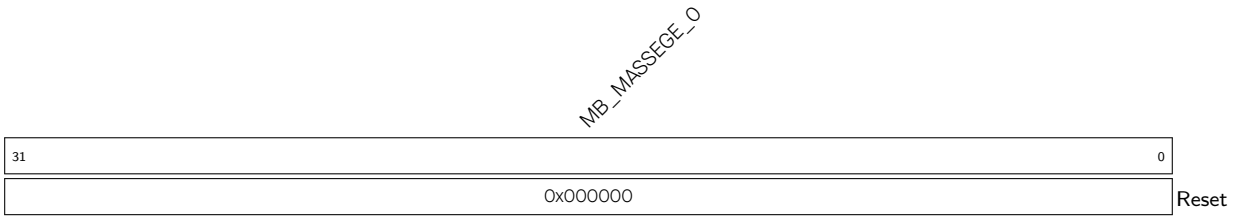
名称	描述	地址	访问
配置寄存器			
MB_MASSEGE_0_REG	信息寄存器 0	0x0000	R/W
MB_MASSEGE_1_REG	信息寄存器 1	0x0004	R/W
MB_MASSEGE_2_REG	信息寄存器 2	0x0008	R/W
MB_MASSEGE_3_REG	信息寄存器 3	0x000C	R/W
MB_MASSEGE_4_REG	信息寄存器 4	0x0010	R/W
MB_MASSEGE_5_REG	信息寄存器 5	0x0014	R/W
MB_MASSEGE_6_REG	信息寄存器 6	0x0018	R/W
MB_MASSEGE_7_REG	信息寄存器 7	0x001C	R/W
MB_MASSEGE_8_REG	信息寄存器 8	0x0020	R/W
MB_MASSEGE_9_REG	信息寄存器 9	0x0024	R/W
MB_MASSEGE_10_REG	信息寄存器 10	0x0028	R/W
MB_MASSEGE_11_REG	信息寄存器 11	0x002C	R/W
MB_MASSEGE_12_REG	信息寄存器 12	0x0030	R/W
MB_MASSEGE_13_REG	信息寄存器 13	0x0034	R/W
MB_MASSEGE_14_REG	信息寄存器 14	0x0038	R/W
MB_MASSEGE_15_REG	信息寄存器 15	0x003C	R/W
MB_REG_CLK_EN_REG	时钟门控寄存器	0x0060	R/W
中断寄存器			
MB_LP_INT_RAW_REG	MB_LP_INTR 原始中断位	0x0040	RO
MB_LP_INT_ST_REG	MB_LP_INTR 中断状态位	0x0044	RO
MB_LP_INT_ENA_REG	MB_LP_INTR 中断使能位	0x0048	R/W
MB_LP_INT_CLR_REG	MB_LP_INTR 中断清除位	0x004C	WO
MB_HP_INT_RAW_REG	MB_HP_INTR 原始中断位	0x0050	RO
MB_HP_INT_ST_REG	MB_HP_INTR 中断状态位	0x0054	RO
MB_HP_INT_ENA_REG	MB_HP_INTR 中断使能位	0x0058	R/W
MB_HP_INT_CLR_REG	MB_HP_INTR 中断清除位	0x005C	WO

24.5 寄存器

本小节的所有地址均为相对于 LP 信箱控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

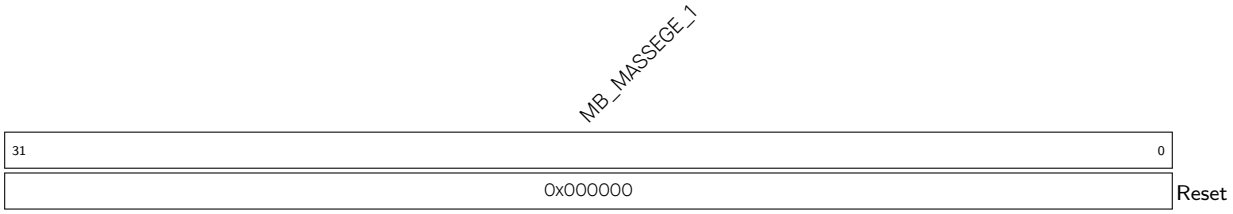
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 24.1. MB_MASSEGE_0_REG (0x0000)



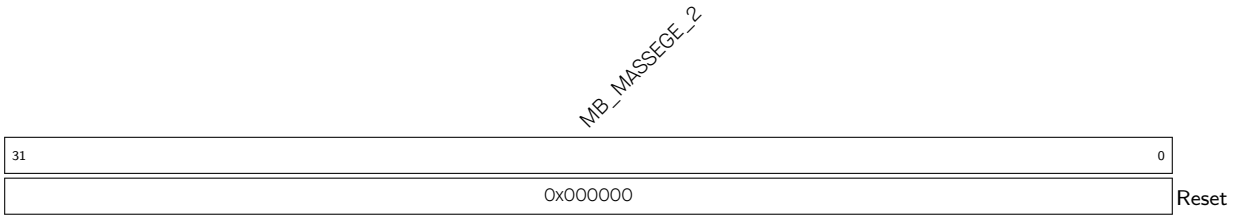
MB_MASSEGE_0 信息寄存器 0。(R/W)

Register 24.2. MB_MASSEGE_1_REG (0x0004)



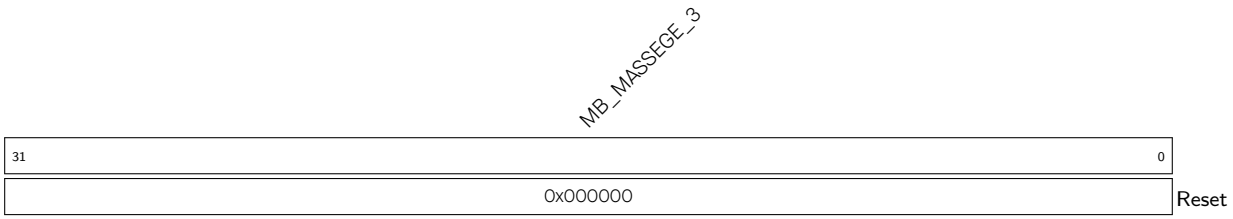
MB_MASSEGE_1 信息寄存器 1。(R/W)

Register 24.3. MB_MASSEGE_2_REG (0x0008)



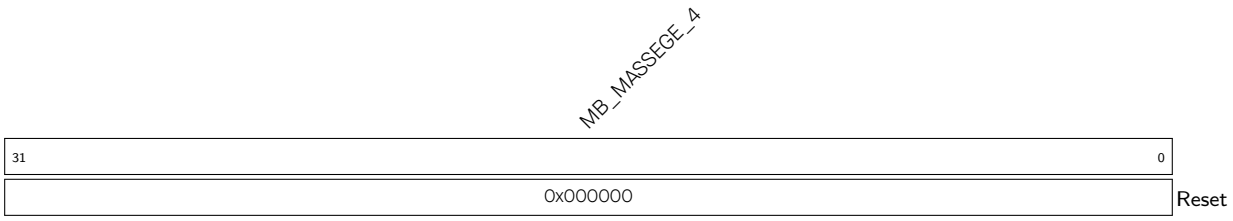
MB_MASSEGE_2 信息寄存器 2。(R/W)

Register 24.4. MB_MASSEGE_3_REG (0x000C)



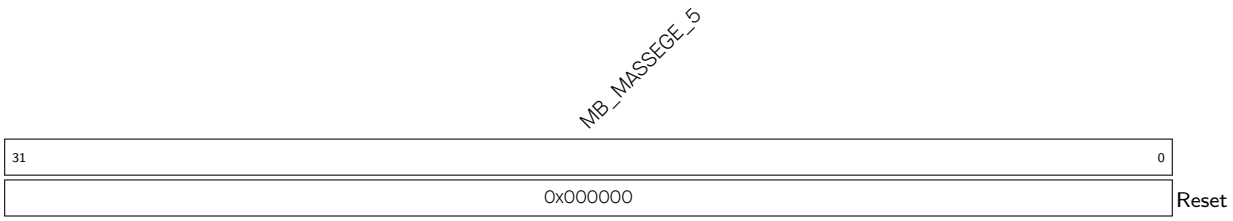
MB_MASSEGE_3 信息寄存器 3。(R/W)

Register 24.5. MB_MASSEGE_4_REG (0x0010)



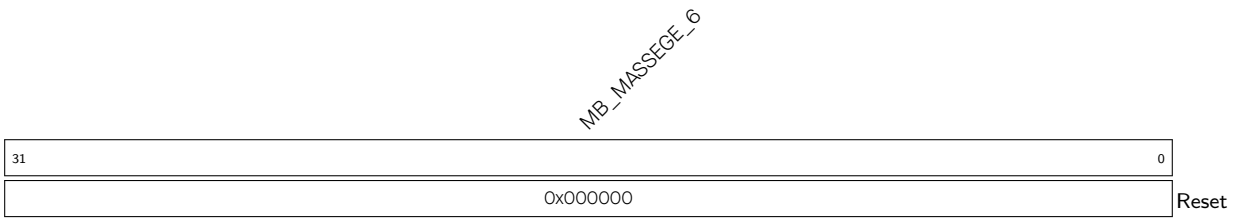
MB_MASSEGE_4 信息寄存器 4。(R/W)

Register 24.6. MB_MASSEGE_5_REG (0x0014)



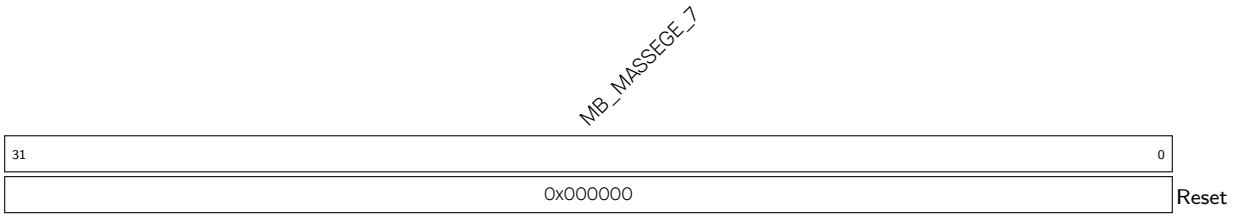
MB_MASSEGE_5 信息寄存器 5。(R/W)

Register 24.7. MB_MASSEGE_6_REG (0x0018)



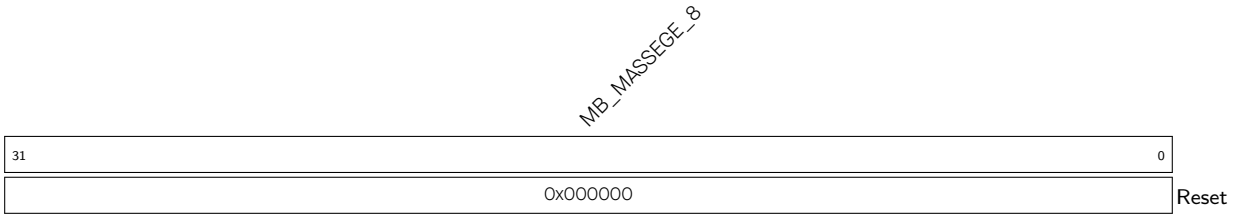
MB_MASSEGE_6 信息寄存器 6。(R/W)

Register 24.8. MB_MASSEGE_7_REG (0x001C)



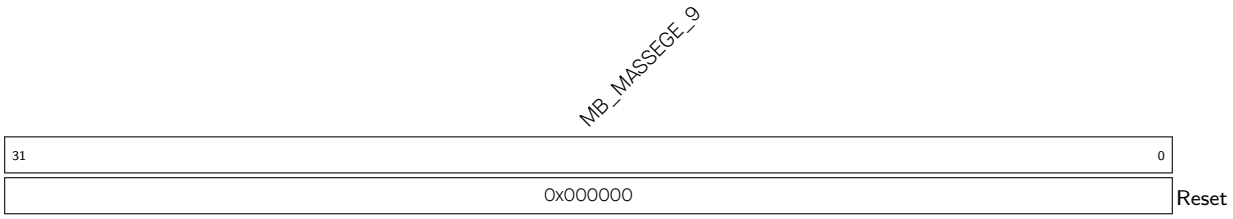
MB_MASSEGE_7 信息寄存器 7。(R/W)

Register 24.9. MB_MASSEGE_8_REG (0x0020)



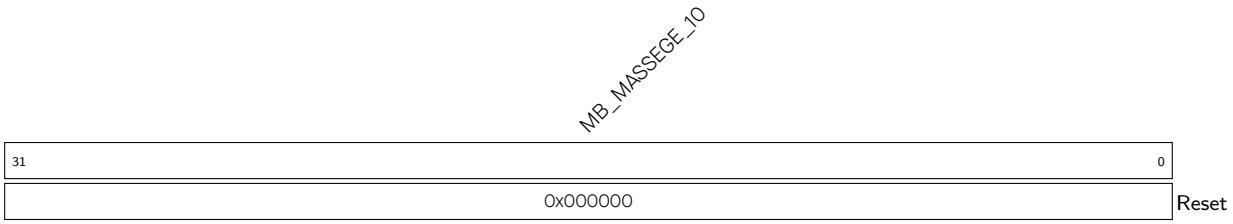
MB_MASSEGE_8 信息寄存器 8。(R/W)

Register 24.10. MB_MASSEGE_9_REG (0x0024)



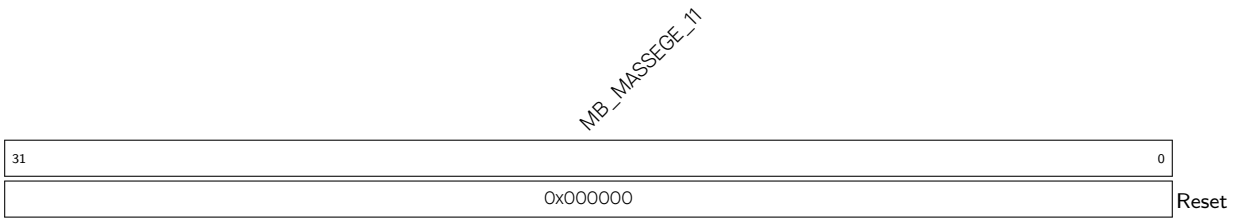
MB_MASSEGE_9 信息寄存器 9。(R/W)

Register 24.11. MB_MASSEGE_10_REG (0x0028)



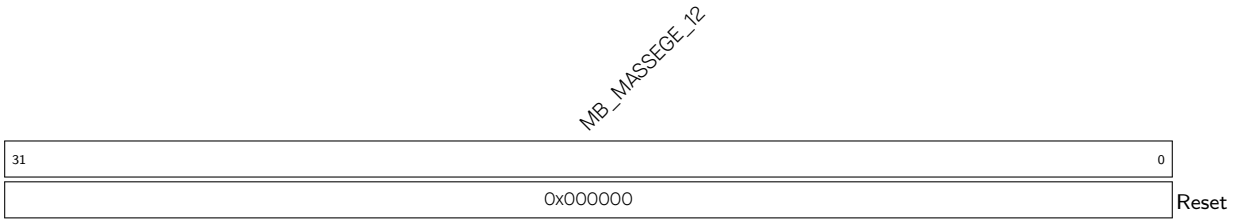
MB_MASSEGE_10 信息寄存器 10。(R/W)

Register 24.12. MB_MASSEGE_11_REG (0x002C)



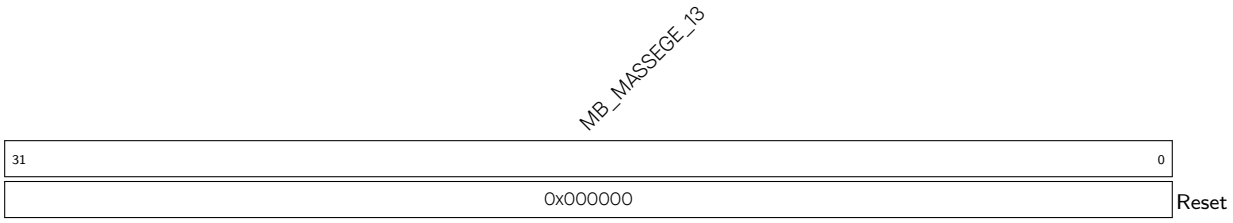
MB_MASSEGE_11 信息寄存器 11。(R/W)

Register 24.13. MB_MASSEGE_12_REG (0x0030)



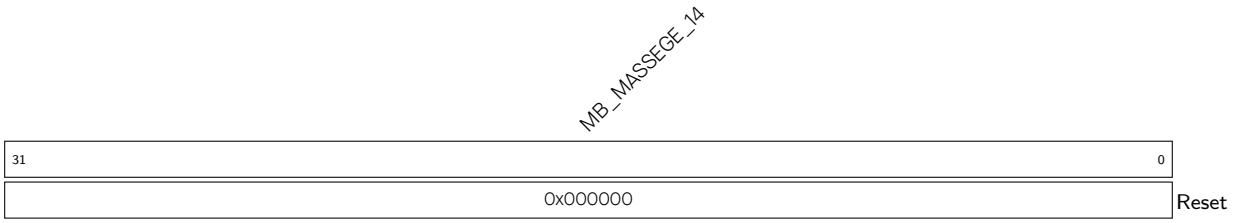
MB_MASSEGE_12 信息寄存器 12。(R/W)

Register 24.14. MB_MASSEGE_13_REG (0x0034)



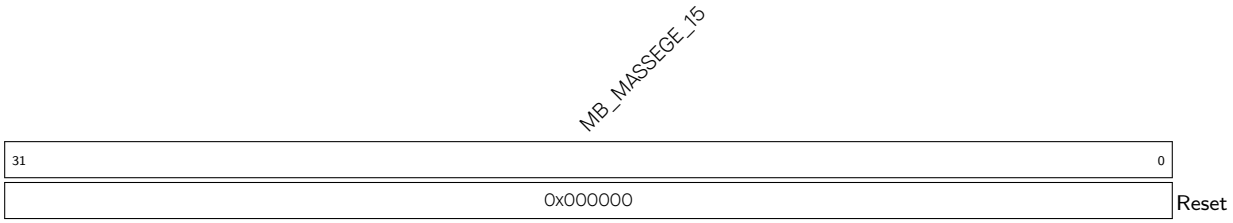
MB_MASSEGE_13 信息寄存器 13。(R/W)

Register 24.15. MB_MASSEGE_14_REG (0x0038)



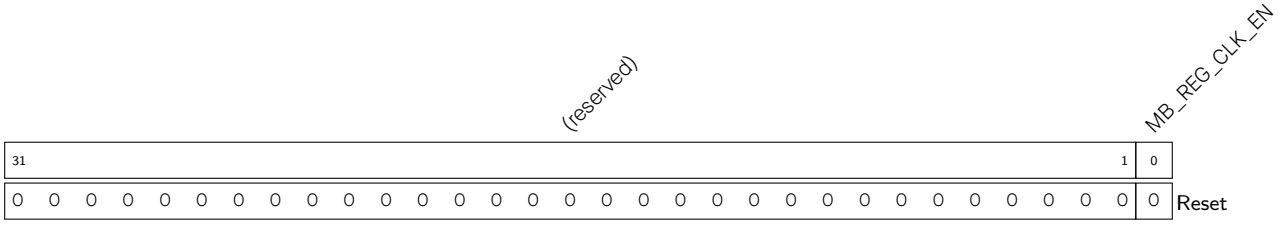
MB_MASSEGE_14 信息寄存器 14。(R/W)

Register 24.16. MB_MASSEGE_15_REG (0x003C)



MB_MASSEGE_15 信息寄存器 15。(R/W)

Register 24.17. MB_REG_CLK_EN_REG (0x0060)



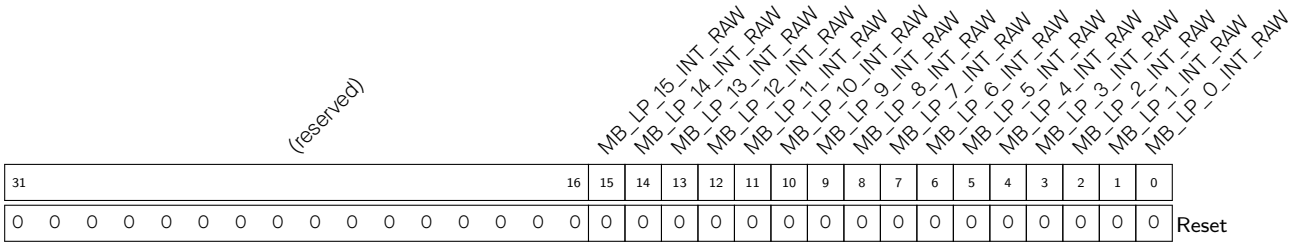
MB_REG_CLK_EN 配置寄存器时钟门控。

1：读写寄存器时，始终开启时钟。

0：仅在读写寄存器操作时，使能时钟。

(R/W)

Register 24.18. MB_LP_INT_RAW_REG (0x0040)



MB_LP_0_INT_RAW MB_LP_0_INT 的原始中断状态。(RO)

MB_LP_1_INT_RAW MB_LP_1_INT 的原始中断状态。(RO)

MB_LP_2_INT_RAW MB_LP_2_INT 的原始中断状态。(RO)

MB_LP_3_INT_RAW MB_LP_3_INT 的原始中断状态。(RO)

MB_LP_4_INT_RAW MB_LP_4_INT 的原始中断状态。(RO)

MB_LP_5_INT_RAW MB_LP_5_INT 的原始中断状态。(RO)

MB_LP_6_INT_RAW MB_LP_6_INT 的原始中断状态。(RO)

MB_LP_7_INT_RAW MB_LP_7_INT 的原始中断状态。(RO)

MB_LP_8_INT_RAW MB_LP_8_INT 的原始中断状态。(RO)

MB_LP_9_INT_RAW MB_LP_9_INT 的原始中断状态。(RO)

MB_LP_10_INT_RAW MB_LP_10_INT 的原始中断状态。(RO)

MB_LP_11_INT_RAW MB_LP_11_INT 的原始中断状态。(RO)

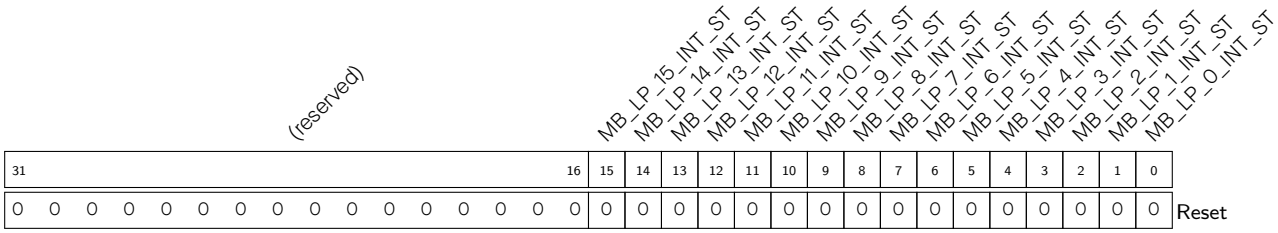
MB_LP_12_INT_RAW MB_LP_12_INT 的原始中断状态。(RO)

MB_LP_13_INT_RAW MB_LP_13_INT 的原始中断状态。(RO)

MB_LP_14_INT_RAW MB_LP_14_INT 的原始中断状态。(RO)

MB_LP_15_INT_RAW MB_LP_15_INT 的原始中断状态。(RO)

Register 24.19. MB_LP_INT_ST_REG (0x0044)



MB_LP_0_INT_ST MB_LP_0_INT 的屏蔽中断状态。(RO)

MB_LP_1_INT_ST MB_LP_1_INT 的屏蔽中断状态。(RO)

MB_LP_2_INT_ST MB_LP_2_INT 的屏蔽中断状态。(RO)

MB_LP_3_INT_ST MB_LP_3_INT 的屏蔽中断状态。(RO)

MB_LP_4_INT_ST MB_LP_4_INT 的屏蔽中断状态。(RO)

MB_LP_5_INT_ST MB_LP_5_INT 的屏蔽中断状态。(RO)

MB_LP_6_INT_ST MB_LP_6_INT 的屏蔽中断状态。(RO)

MB_LP_7_INT_ST MB_LP_7_INT 的屏蔽中断状态。(RO)

MB_LP_8_INT_ST MB_LP_8_INT 的屏蔽中断状态。(RO)

MB_LP_9_INT_ST MB_LP_9_INT 的屏蔽中断状态。(RO)

MB_LP_10_INT_ST MB_LP_10_INT 的屏蔽中断状态。(RO)

MB_LP_11_INT_ST MB_LP_11_INT 的屏蔽中断状态。(RO)

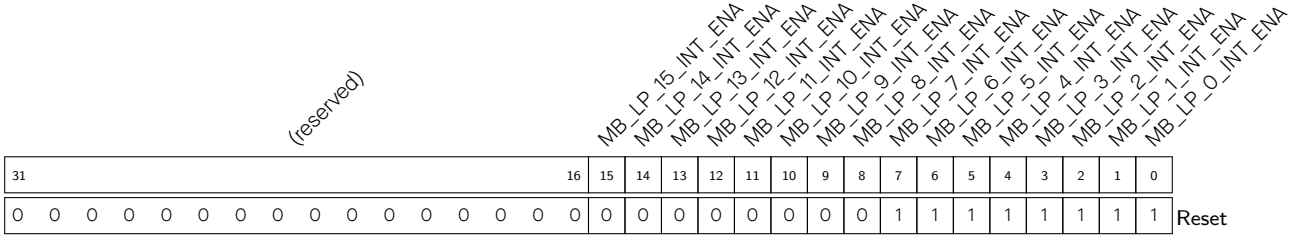
MB_LP_12_INT_ST MB_LP_12_INT 的屏蔽中断状态。(RO)

MB_LP_13_INT_ST MB_LP_13_INT 的屏蔽中断状态。(RO)

MB_LP_14_INT_ST MB_LP_14_INT 的屏蔽中断状态。(RO)

MB_LP_15_INT_ST MB_LP_15_INT 的屏蔽中断状态。(RO)

Register 24.20. MB_LP_INT_ENA_REG (0x0048)



MB_LP_0_INT_ENA 写 1 使能 MB_LP_0_INT 中断。(R/W)

MB_LP_1_INT_ENA 写 1 使能 MB_LP_1_INT 中断。(R/W)

MB_LP_2_INT_ENA 写 1 使能 MB_LP_2_INT 中断。(R/W)

MB_LP_3_INT_ENA 写 1 使能 MB_LP_3_INT 中断。(R/W)

MB_LP_4_INT_ENA 写 1 使能 MB_LP_4_INT 中断。(R/W)

MB_LP_5_INT_ENA 写 1 使能 MB_LP_5_INT 中断。(R/W)

MB_LP_6_INT_ENA 写 1 使能 MB_LP_6_INT 中断。(R/W)

MB_LP_7_INT_ENA 写 1 使能 MB_LP_7_INT 中断。(R/W)

MB_LP_8_INT_ENA 写 1 使能 MB_LP_8_INT 中断。(R/W)

MB_LP_9_INT_ENA 写 1 使能 MB_LP_9_INT 中断。(R/W)

MB_LP_10_INT_ENA 写 1 使能 MB_LP_10_INT 中断。(R/W)

MB_LP_11_INT_ENA 写 1 使能 MB_LP_11_INT 中断。(R/W)

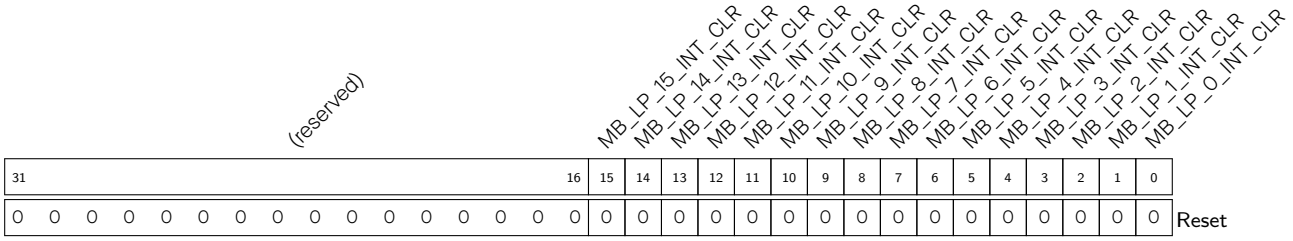
MB_LP_12_INT_ENA 写 1 使能 MB_LP_12_INT 中断。(R/W)

MB_LP_13_INT_ENA 写 1 使能 MB_LP_13_INT 中断。(R/W)

MB_LP_14_INT_ENA 写 1 使能 MB_LP_14_INT 中断。(R/W)

MB_LP_15_INT_ENA 写 1 使能 MB_LP_15_INT 中断。(R/W)

Register 24.21. MB_LP_INT_CLR_REG (0x004C)



MB_LP_0_INT_CLR 写 1 清除 MB_LP_0_INT 中断。(WO)

MB_LP_1_INT_CLR 写 1 清除 MB_LP_1_INT 中断。(WO)

MB_LP_2_INT_CLR 写 1 清除 MB_LP_2_INT 中断。(WO)

MB_LP_3_INT_CLR 写 1 清除 MB_LP_3_INT 中断。(WO)

MB_LP_4_INT_CLR 写 1 清除 MB_LP_4_INT 中断。(WO)

MB_LP_5_INT_CLR 写 1 清除 MB_LP_5_INT 中断。(WO)

MB_LP_6_INT_CLR 写 1 清除 MB_LP_6_INT 中断。(WO)

MB_LP_7_INT_CLR 写 1 清除 MB_LP_7_INT 中断。(WO)

MB_LP_8_INT_CLR 写 1 清除 MB_LP_8_INT 中断。(WO)

MB_LP_9_INT_CLR 写 1 清除 MB_LP_9_INT 中断。(WO)

MB_LP_10_INT_CLR 写 1 清除 MB_LP_10_INT 中断。(WO)

MB_LP_11_INT_CLR 写 1 清除 MB_LP_11_INT 中断。(WO)

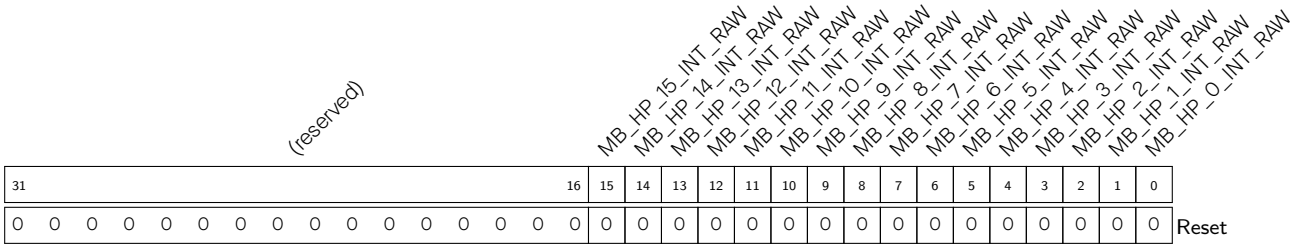
MB_LP_12_INT_CLR 写 1 清除 MB_LP_12_INT 中断。(WO)

MB_LP_13_INT_CLR 写 1 清除 MB_LP_13_INT 中断。(WO)

MB_LP_14_INT_CLR 写 1 清除 MB_LP_14_INT 中断。(WO)

MB_LP_15_INT_CLR 写 1 清除 MB_LP_15_INT 中断。(WO)

Register 24.22. MB_HP_INT_RAW_REG (0x0050)



MB_HP_0_INT_RAW MB_HP_0_INT 的原始中断状态。(RO)

MB_HP_1_INT_RAW MB_HP_1_INT 的原始中断状态。(RO)

MB_HP_2_INT_RAW MB_HP_2_INT 的原始中断状态。(RO)

MB_HP_3_INT_RAW MB_HP_3_INT 的原始中断状态。(RO)

MB_HP_4_INT_RAW MB_HP_4_INT 的原始中断状态。(RO)

MB_HP_5_INT_RAW MB_HP_5_INT 的原始中断状态。(RO)

MB_HP_6_INT_RAW MB_HP_6_INT 的原始中断状态。(RO)

MB_HP_7_INT_RAW MB_HP_7_INT 的原始中断状态。(RO)

MB_HP_8_INT_RAW MB_HP_8_INT 的原始中断状态。(RO)

MB_HP_9_INT_RAW MB_HP_9_INT 的原始中断状态。(RO)

MB_HP_10_INT_RAW MB_HP_10_INT 的原始中断状态。(RO)

MB_HP_11_INT_RAW MB_HP_11_INT 的原始中断状态。(RO)

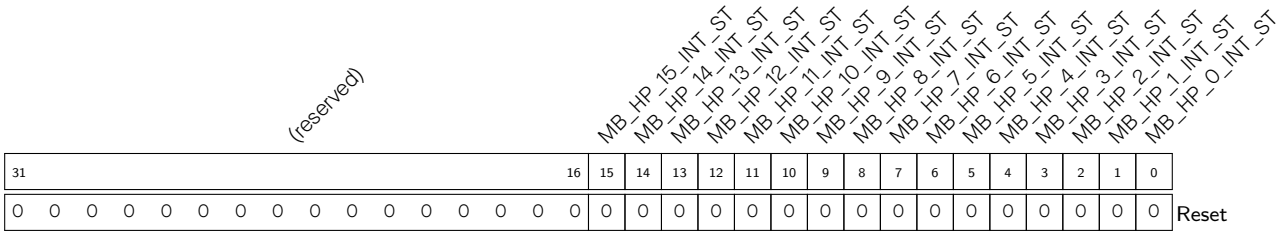
MB_HP_12_INT_RAW MB_HP_12_INT 的原始中断状态。(RO)

MB_HP_13_INT_RAW MB_HP_13_INT 的原始中断状态。(RO)

MB_HP_14_INT_RAW MB_HP_14_INT 的原始中断状态。(RO)

MB_HP_15_INT_RAW MB_HP_15_INT 的原始中断状态。(RO)

Register 24.23. MB_HP_INT_ST_REG (0x0054)



MB_HP_0_INT_ST MB_HP_0_INT 的屏蔽中断状态。(RO)

MB_HP_1_INT_ST MB_HP_1_INT 的屏蔽中断状态。(RO)

MB_HP_2_INT_ST MB_HP_2_INT 的屏蔽中断状态。(RO)

MB_HP_3_INT_ST MB_HP_3_INT 的屏蔽中断状态。(RO)

MB_HP_4_INT_ST MB_HP_4_INT 的屏蔽中断状态。(RO)

MB_HP_5_INT_ST MB_HP_5_INT 的屏蔽中断状态。(RO)

MB_HP_6_INT_ST MB_HP_6_INT 的屏蔽中断状态。(RO)

MB_HP_7_INT_ST MB_HP_7_INT 的屏蔽中断状态。(RO)

MB_HP_8_INT_ST MB_HP_8_INT 的屏蔽中断状态。(RO)

MB_HP_9_INT_ST MB_HP_9_INT 的屏蔽中断状态。(RO)

MB_HP_10_INT_ST MB_HP_10_INT 的屏蔽中断状态。(RO)

MB_HP_11_INT_ST MB_HP_11_INT 的屏蔽中断状态。(RO)

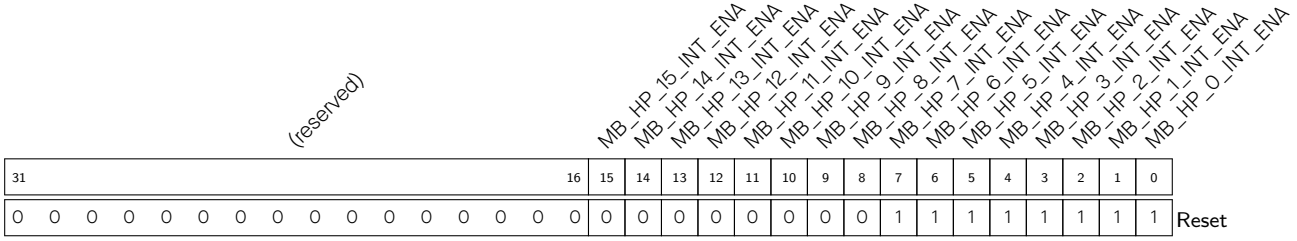
MB_HP_12_INT_ST MB_HP_12_INT 的屏蔽中断状态。(RO)

MB_HP_13_INT_ST MB_HP_13_INT 的屏蔽中断状态。(RO)

MB_HP_14_INT_ST MB_HP_14_INT 的屏蔽中断状态。(RO)

MB_HP_15_INT_ST MB_HP_15_INT 的屏蔽中断状态。(RO)

Register 24.24. MB_HP_INT_ENA_REG (0x0058)



MB_HP_0_INT_ENA 写 1 使能 MB_HP_0_INT 中断。(R/W)

MB_HP_1_INT_ENA 写 1 使能 MB_HP_1_INT 中断。(R/W)

MB_HP_2_INT_ENA 写 1 使能 MB_HP_2_INT 中断。(R/W)

MB_HP_3_INT_ENA 写 1 使能 MB_HP_3_INT 中断。(R/W)

MB_HP_4_INT_ENA 写 1 使能 MB_HP_4_INT 中断。(R/W)

MB_HP_5_INT_ENA 写 1 使能 MB_HP_5_INT 中断。(R/W)

MB_HP_6_INT_ENA 写 1 使能 MB_HP_6_INT 中断。(R/W)

MB_HP_7_INT_ENA 写 1 使能 MB_HP_7_INT 中断。(R/W)

MB_HP_8_INT_ENA 写 1 使能 MB_HP_8_INT 中断。(R/W)

MB_HP_9_INT_ENA 写 1 使能 MB_HP_9_INT 中断。(R/W)

MB_HP_10_INT_ENA 写 1 使能 MB_HP_10_INT 中断。(R/W)

MB_HP_11_INT_ENA 写 1 使能 MB_HP_11_INT 中断。(R/W)

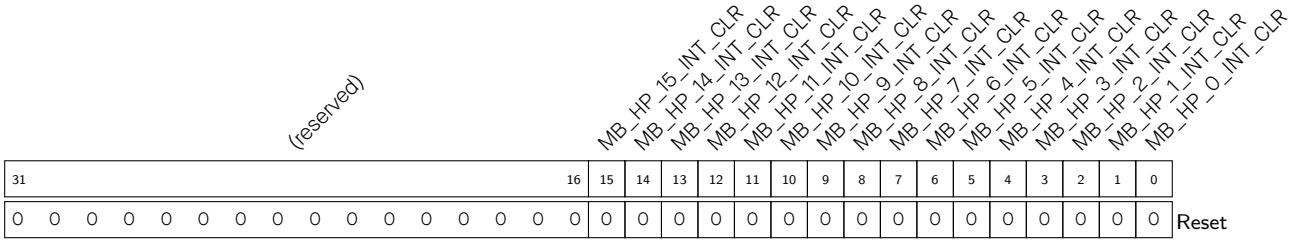
MB_HP_12_INT_ENA 写 1 使能 MB_HP_12_INT 中断。(R/W)

MB_HP_13_INT_ENA 写 1 使能 MB_HP_13_INT 中断。(R/W)

MB_HP_14_INT_ENA 写 1 使能 MB_HP_14_INT 中断。(R/W)

MB_HP_15_INT_ENA 写 1 使能 MB_HP_15_INT 中断。(R/W)

Register 24.25. MB_HP_INT_CLR_REG (0x005C)



MB_HP_0_INT_CLR 写 1 清除 MB_HP_0_INT 中断。(WO)

MB_HP_1_INT_CLR 写 1 清除 MB_HP_1_INT 中断。(WO)

MB_HP_2_INT_CLR 写 1 清除 MB_HP_2_INT 中断。(WO)

MB_HP_3_INT_CLR 写 1 清除 MB_HP_3_INT 中断。(WO)

MB_HP_4_INT_CLR 写 1 清除 MB_HP_4_INT 中断。(WO)

MB_HP_5_INT_CLR 写 1 清除 MB_HP_5_INT 中断。(WO)

MB_HP_6_INT_CLR 写 1 清除 MB_HP_6_INT 中断。(WO)

MB_HP_7_INT_CLR 写 1 清除 MB_HP_7_INT 中断。(WO)

MB_HP_8_INT_CLR 写 1 清除 MB_HP_8_INT 中断。(WO)

MB_HP_9_INT_CLR 写 1 清除 MB_HP_9_INT 中断。(WO)

MB_HP_10_INT_CLR 写 1 清除 MB_HP_10_INT 中断。(WO)

MB_HP_11_INT_CLR 写 1 清除 MB_HP_11_INT 中断。(WO)

MB_HP_12_INT_CLR 写 1 清除 MB_HP_12_INT 中断。(WO)

MB_HP_13_INT_CLR 写 1 清除 MB_HP_13_INT 中断。(WO)

MB_HP_14_INT_CLR 写 1 清除 MB_HP_14_INT 中断。(WO)

MB_HP_15_INT_CLR 写 1 清除 MB_HP_15_INT 中断。(WO)

第 25 章

欠压检测器

25.1 概述

ESP32-P4 的欠压检测器可以检查管脚 VDD_ANA，VDD_BAT 的电压，在电压快速下落至预设阈值（默认为 2.4 V）以下时发出触发信号，并进行相应处理，从而关闭部分耗电模块（主要是 flash 模块），为数字模块争取更多时间，用以保存、转移重要数据。

25.2 主要特性

- 支持 2 种管脚欠压检测
  - 支持模拟电源管脚 VDD_ANA 欠压检测
  - 支持电池电源管脚 VDD_BAT 欠压检测
- 支持两种检测模式
  - 模式 0：当欠压计数器达到设定的阈值后触发中断，并根据配置选择复位方式
  - 模式 1：欠压发生后直接触发系统复位
- 支持监控阈值与噪声过滤的配置
- 支持欠压后的复位等级选择

25.3 结构概览

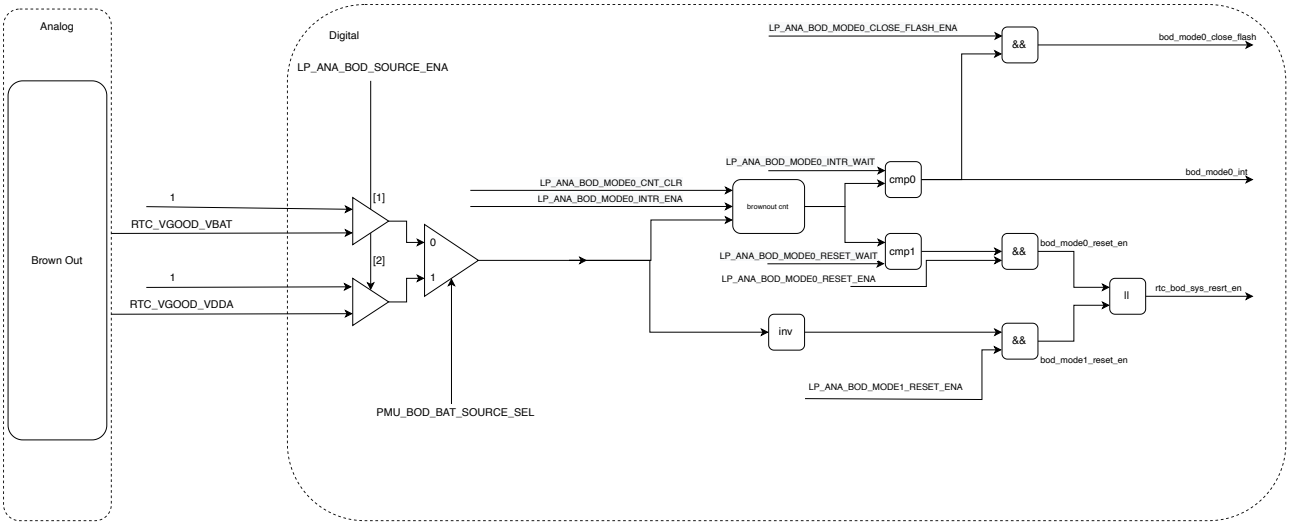


图 25.3-1. 欠压检测模块结构图

- 检测源头选择：欠压检测器可通过配置 LP_ANA_BOD_SOURCE_ENA 选择检测电源；其中 RTC_VGOOD_VBAT 是 VDD_BAT 的检测结果，RTC_VGOOD_VDDA 是 VDD_ANA 的检测结果。
- 欠压检测器 (Brown Out)：模拟电路，用于检测电源管脚电压并产生欠压信号，它将电源管脚电压与预设电压阈值进行比较，当电源管脚电压低于该阈值时，输出欠压信号。
- 欠压计数器 (brownout cnt)：用于过滤被检测电源管脚上的电压噪声，当接收到欠压信号时，该计数器将基于时钟 LP_DYN_FAST_CLK 进行计数，计数过程中若欠压信号失效，计数器将被直接清零。也可以通过配置 LP_ANA_BOD_MODE0_CNT_CLR 清空计数器。
- 中断比较器 (cmp0)：根据欠压计数器的值与配置的阈值比较产生中断，当欠压计数器的计数值大于配置的阈值后将产生中断信号 bod_mode0_int。如果使能了 LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA，将会触发 bod_mode0_close_flash 使 flash 进入 SUSPEND 状态。
- 复位比较器 (cmp1)：根据欠压计数器的值与配置的阈值比较产生复位，当欠压计数器的计数值大于配置的阈值后将产生复位信号。

25.4 功能描述

- 被检测管脚
  - VDD_ANA: 模拟电路供电管脚，支持模式 0 与模式 1。
  - VDD_BAT: 电池供电管脚，支持模式 0 与模式 1，且存在两个可配置的电压检测阈值:
    * VBAT_BOD：用作 VDD_BAT 的欠压检测
    * VBAT_CHARGER：用于检测 VDD_BAT 连接的电池是否需要充电

  这两个阈值分别触发不同的中断。
- 检测模式
  根据欠压检测器检测到欠压信号后的处理方法，分为两种检测模式：

  - 模式 0：使能该模式后，内置的欠压计数器将在模拟产生欠压信号后开始计数，当欠压计数器达到 LP_ANA_BOD_MODE0_RESET_WAIT 和 LP_ANA_BOD_MODE0_INTR_WAIT 中设定的阈值后将分别产生复位与中断信号，以此过滤掉一些被检测电源上的噪声。配置 LP_ANA_BOD_MODE0_RESET_SEL 可选择复位方式：
    * 0：将触发芯片复位。
    * 1：将触发系统复位。
  - 模式 1：直接触发系统复位，默认为该模式。

- 电压监控阈值
  电压监控阈值由模拟 I2C 控制器直接配置模拟电路实现。阈值配置寄存器如下：BIAS 为模拟 I2C 控制器的从机，从机地址为 0x6a。如下 REGXX 为 BIAS 的寄存器地址，[x:x] 为该寄存器的 bit 位。
  - BIAS_OR_DREFL_VBAT (REG08[7:5])：释放 VDD_BAT 欠压状态的阈值
  - BIAS_OR_DREFH_VBAT (REG08[4:2])：VDD_BAT 的欠压检测阈值
  - BIAS_OR_DREFL_VDDA (REG09[7:5])：释放 VDD_ANA 欠压状态的阈值
  - BIAS_OR_DREFH_VDDA (REG09[4:2])：VDD_ANA 的欠压检测阈值
  - BIAS_OR_DREFL_VBAT_CHARGER (REG10[7:5])：释放 VDD_BAT 连接电池欠压状态的阈值
  - BIAS_OR_DREFH_VBAT_CHARGER (REG10[4:2])：VDD_BAT 连接电池的欠压检测阈值

  VBAT 和 VDDA 阈值配置与电压的关系如下：
  - 0: 2.0 V
  - 1: 2.1 V
  - 2: 2.2 V
  - 3: 2.3 V
  - 4: 2.4 V
  - 5: 2.5 V
  - 6: 2.6 V
  - 7: 2.7 V

  VBAT_CHARGER 阈值配置与电压的关系如下：
  - 0: 2.2 V
  - 1: 2.3 V
  - 2: 2.4 V
  - 3: 2.5 V
  - 4: 2.6 V
  - 5: 2.7 V
  - 6: 2.8 V
  - 7: 2.9 V

- 电压监控滤波
  针对 VDD_BAT 的两个阈值检测输出结果添加了专用的毛刺过滤器，如下图所示：

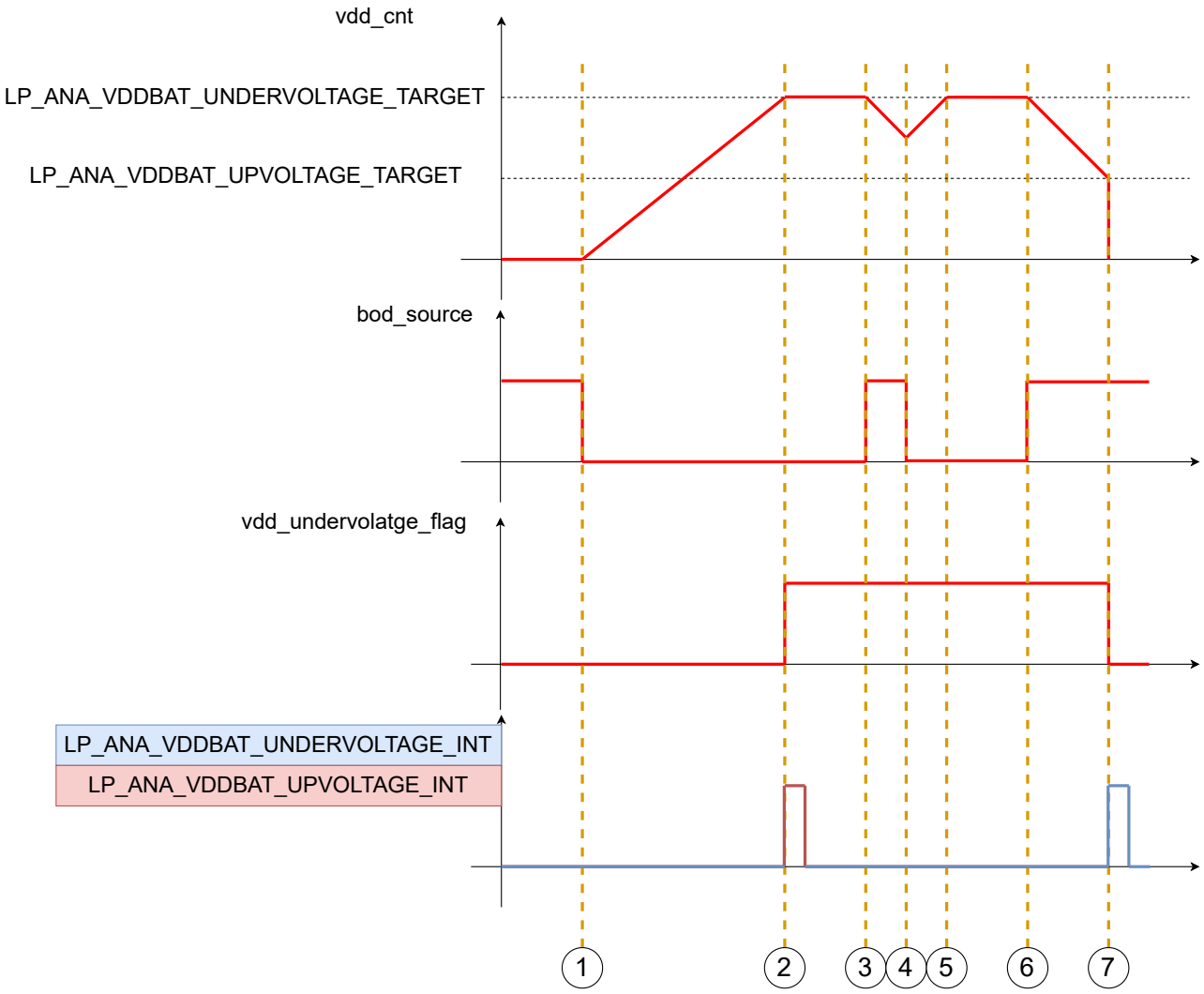


图 25.4-1. 欠压检测模块结构图

  - vdd_cnt：欠压计数器计数值。
  - bod_source：欠压检测源，这里指 VDD_BAT。
  - vdd_undervoltage_flag：欠压指示标志。
  - 阶段 0 ~ 1：当 bod_source 为 1，表示电压正常，欠压计数器（vdd_cnt）保持为 0。
  - 阶段 1 ~ 2：当 bod_source 为 0，表示电压异常，欠压计数器将开始计数，当计数器值达到欠压计数阈值后，拉高欠压标志信号，并产生中断。
  - 阶段 2 ~ 3：当欠压标志信号为高时，欠压计数器将不再计数，并稳定在寄存器 LP_ANA_VDDBAT_UNDERVOLTAGE_TARGET 的值。
  - 阶段 3 ~ 4 ~ 5：若 bod_source 短暂的恢复正常，欠压计数器将向下计数，但由于计数值未低于 LP_ANA_VDDBAT_UPVOLTAGE_TARGET，则欠压标志信号不翻转。
  - 阶段 6 ~ 7：当 bod_source 彻底恢复正常，欠压计数器向下计数到 LP_ANA_VDDBAT_UPVOLTAGE_TARGET，欠压标志信号拉低，计数器清零。

25.5 中断

ESP32-P4 的欠压检测模块可以生成以下中断信号并发送给中断矩阵。

- LP_ANAPERI_INT

上述中断信号由欠压检测模块的内部中断源生成。

- LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT：当管脚 VDD_BAT 的电压恢复至预设的充电阈值时触发。
- LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT：当管脚 VDD_BAT 的电压下降至预设的充电阈值时触发。
- LP_ANA_VDDBAT_UPVOLTAGE_INT：当管脚 VDD_BAT 的电压恢复至预设的欠压阈值时触发。
- LP_ANA_VDDBAT_UNDERVOLTAGE_INT：当管脚 VDD_BAT 的电压下降至预设的欠压阈值时触发。
- LP_ANA_BOD_MODE0_INT：模式 0 时，当欠压计数器达到设定的阈值时触发该中断。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 25.7 寄存器列表。

25.6 配置流程

- 模式 0
  - 配置 LP_ANA_BOD_MODE0_RESET_WAIT 决定产生复位的计数器阈值。
  - 配置 LP_ANA_BOD_MODE0_INTR_WAIT 决定产生中断的计数器阈值。
  - 配置 LP_ANA_BOD_MODE0_RESET_SEL 选择模式 0 下的复位方式。
  - 配置 LP_ANA_BOD_SOURCE_ENA 使能需要检测的管脚。
  - 置位 LP_ANA_BOD_MODE0_RESET_ENA 使能欠压复位功能。
  - 置位 LP_ANA_BOD_MODE0_INTR_ENA 使能噪声过滤计数器与中断功能。
- 模式 1
  - 配置 LP_ANA_BOD_SOURCE_ENA 使能需要检测的管脚。

25.7 寄存器列表

本小节的所有地址均为相对于欠压检测器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

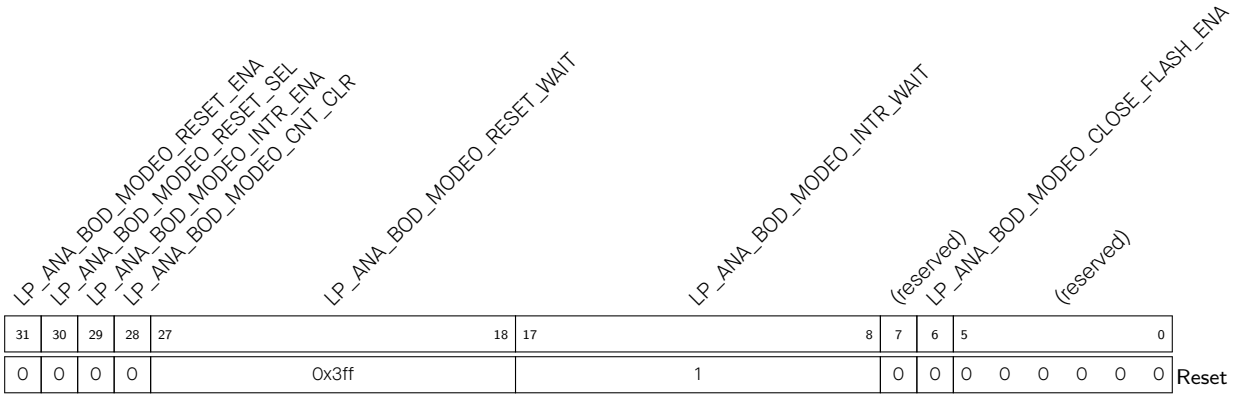
名称	描述	地址	类型
LP_ANA_BOD_MODE0_CNTL_REG	模式 0 控制寄存器。	0x0000	R/W
LP_ANA_VDD_SOURCE_CNTL_REG	检测源控制寄存器。	0x0008	varies
LP_ANA_VDDBAT_BOD_CNTL_REG	VDD_BAT 欠压检测控制寄存器。	0x000C	varies
LP_ANA_VDDBAT_CHARGE_CNTL_REG	VDD_BAT 欠压预充电控制寄存器。	0x0010	varies
LP_ANA_INT_RAW_REG	原始中断寄存器	0x0020	RO
LP_ANA_INT_ST_REG	中断状态寄存器	0x0024	RO
LP_ANA_INT_ENA_REG	中断使能寄存器	0x0028	R/W
LP_ANA_INT_CLR_REG	中断清除寄存器	0x002C	WT

25.8 寄存器

本小节的所有地址均为相对于欠压检测器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 25.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)



LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA 使能欠压发生时关闭 flash 功能。

0：关闭

1：使能

(R/W)

LP_ANA_BOD_MODE0_INTR_WAIT 配置模式 0 下触发中断的计数阈值。(R/W)

LP_ANA_BOD_MODE0_RESET_WAIT 配置模式 0 下触发复位的计数阈值。(R/W)

LP_ANA_BOD_MODE0_CNT_CLR 清零模式 0 下欠压滤波计数器值。(R/W)

LP_ANA_BOD_MODE0_INTR_ENA 使能欠压滤波计数器与中断。

0：关闭

1：使能

(R/W)

LP_ANA_BOD_MODE0_RESET_SEL 配置模式 0 下欠压触发时的复位方式。

0：将触发芯片复位

1：将触发系统复位

(R/W)

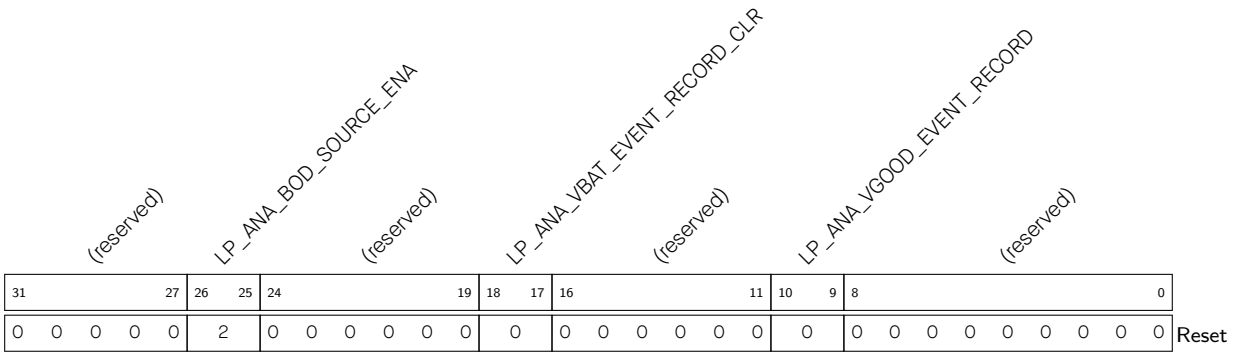
LP_ANA_BOD_MODE0_RESET_ENA 使能欠压复位功能。

0：关闭

1：使能

(R/W)

Register 25.2. LP_ANA_VDD_SOURCE_CNTL_REG (0x0008)



LP_ANA_VGOOD_EVENT_RECORD 记录曾发生欠压的标志位。

0：VDD_BAT 与 VDD_ANA 管脚均未发生欠压

1：VDD_BAT 管脚曾发生欠压

2：VDD_ANA 管脚曾发生欠压

3：VDD_BAT 与 VDD_ANA 管脚均发生欠压

(RO)

LP_ANA_VBAT_EVENT_RECORD_CLR 标志位清零信号。

0：不操作

1：清除 VDD_BAT 欠压标志位

2：清除 VDD_ANA 欠压标志位

3：无效值

(WT)

LP_ANA_BOD_SOURCE_ENA 检测源使能寄存器。

0：不检测任何管脚电压

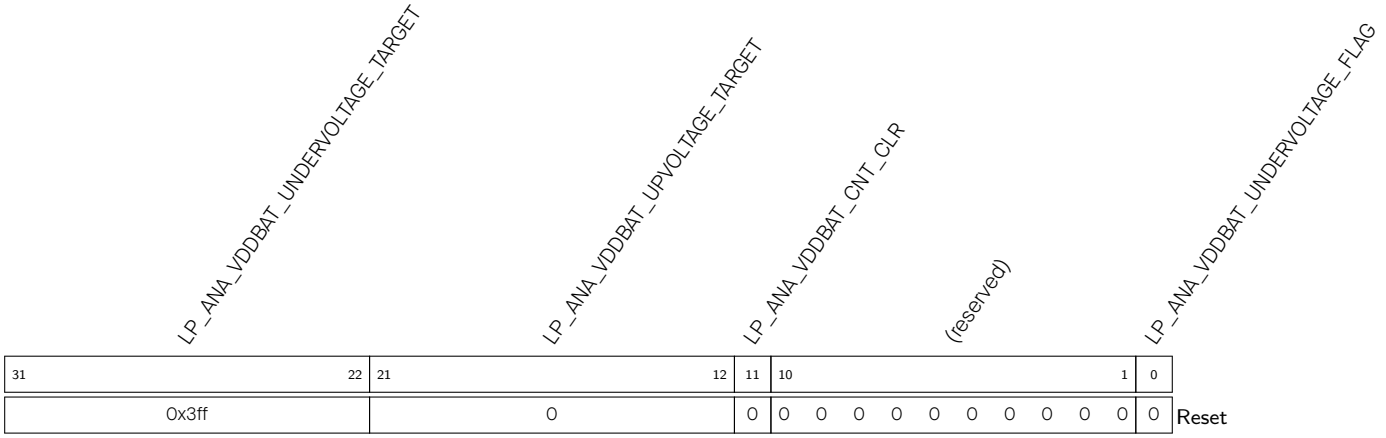
1：只使能 VDD_BAT 管脚检测

2：只使能 VDD_ANA 管脚检测

3：使能 VDD_BAT 与 VDD_ANA 管脚检测

(R/W)

Register 25.3. LP_ANA_VDDBAT_BOD_CNTL_REG (0x000C)



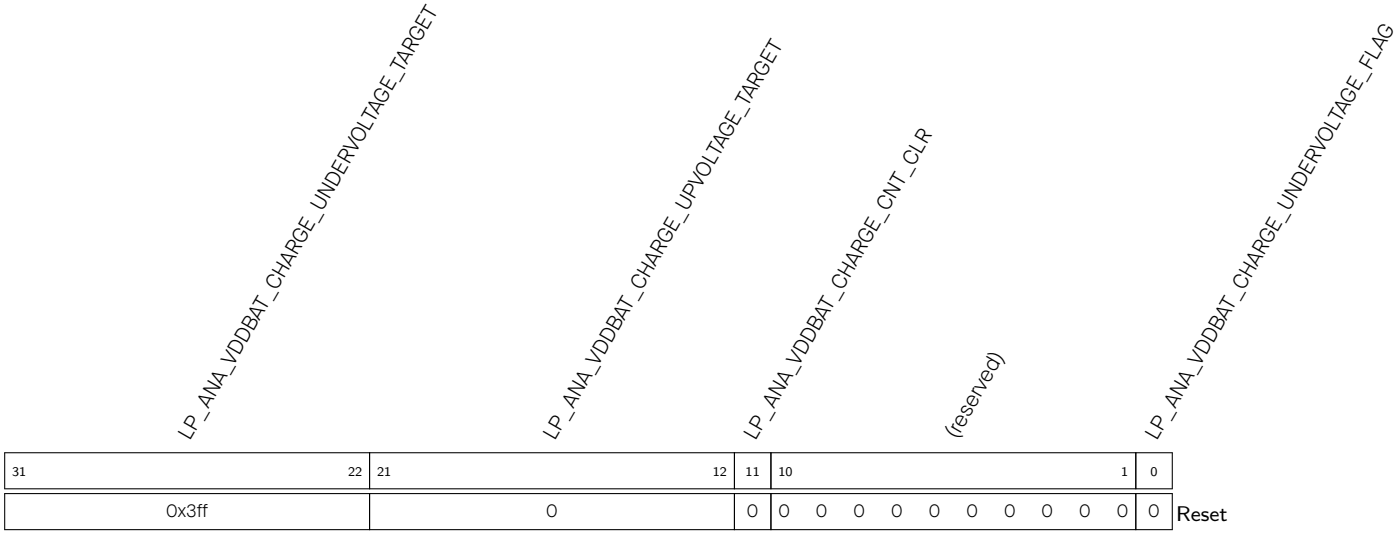
LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG VDD_BAT 管脚欠压标志信号。(RO)

LP_ANA_VDDBAT_CNT_CLR 清零 VDD_BAT 管脚欠压计数器。(WT)

LP_ANA_VDDBAT_UPVOLTAGE_TARGET 配置 VDD_BAT 管脚欠压计数器电压恢复阈值，当计数器向下减至该值时，LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG 清零。(R/W)

LP_ANA_VDDBAT_UNDERVOLTAGE_TARGET 配置 VDD_BAT 管脚欠压计数器欠压阈值，当计数器向上增至该值时，LP_ANA_VDDBAT_UNDERVOLTAGE_FLAG 置 1。(R/W)

Register 25.4. LP_ANA_VDDBAT_CHARGE_CNTL_REG (0x0010)



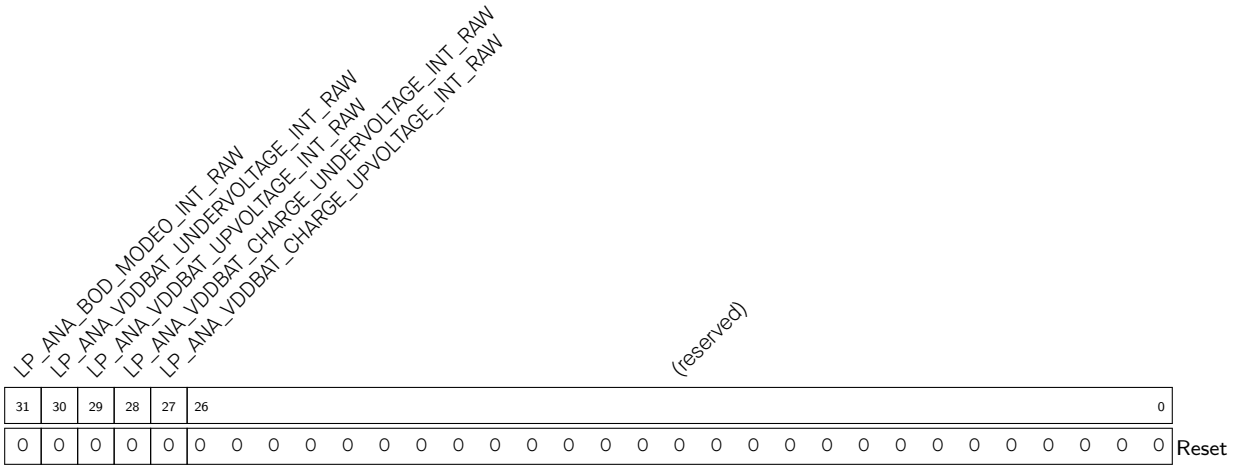
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG VDD_BAT 管脚充电标志信号。(RO)

LP_ANA_VDDBAT_CHARGE_CNT_CLR 清零 VDD_BAT 管脚欠压计数器。(WT)

LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_TARGET 配置 VDD_BAT 管脚充电计数器电压恢复阈值，当计数器向下减至该值时，LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG 清零。(R/W)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_TARGET 配置 VDD_BAT 管脚欠压计数器充电阈值，当计数器向上增至该值时，LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_FLAG 置 1。(R/W)

Register 25.5. LP_ANA_INT_RAW_REG (0x0020)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_RAW VDD_BAT 电压恢复至充电阈值电压时产生该中断。(RO)

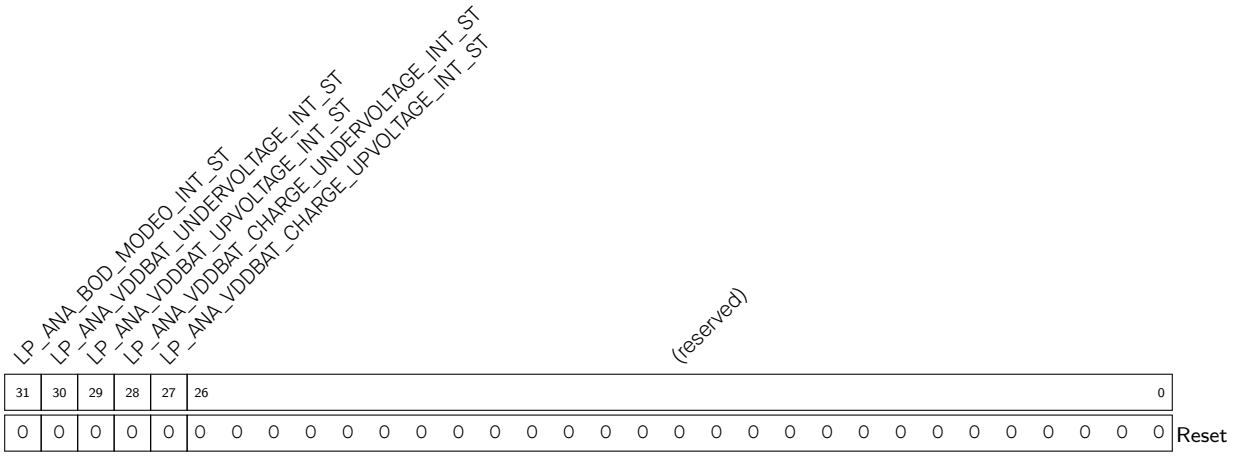
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_RAW VDD_BAT 电压降低至充电阈值电压时产生该中断。(RO)

LP_ANA_VDDBAT_UPVOLTAGE_INT_RAW VDD_BAT 电压恢复至欠压阈值电压时产生该中断。(RO)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_RAW VDD_BAT 电压降低至欠压阈值电压时产生该中断。(RO)

LP_ANA_BOD_MODE0_INT_RAW 模式 0 欠压中断。(RO)

Register 25.6. LP_ANA_INT_ST_REG (0x0024)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_ST VDD_BAT 电压恢复至充电阈值电压时产生并发送该中断给 CPU。(RO)

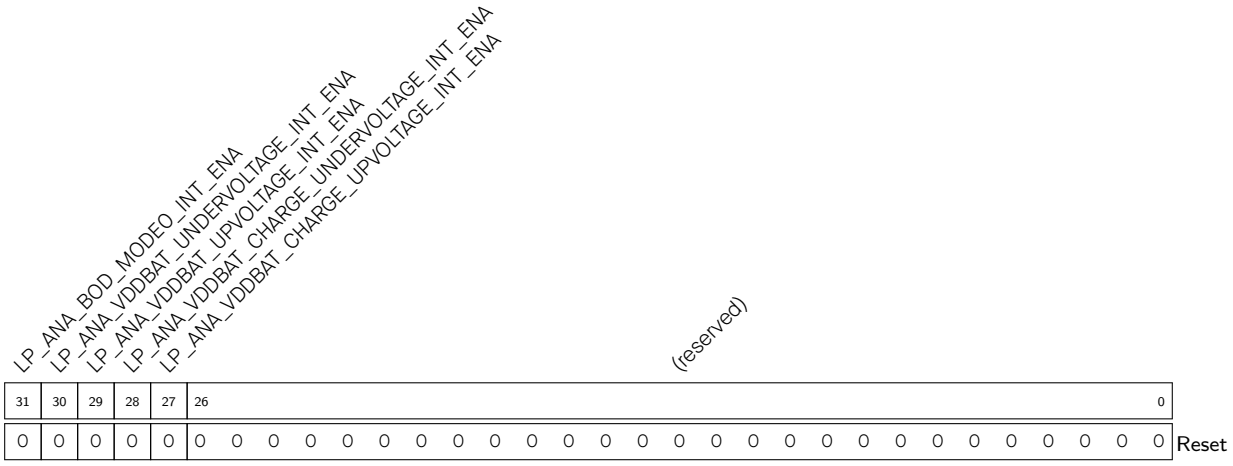
LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_ST VDD_BAT 电压降低至充电阈值电压时产生并发送该中断给 CPU。(RO)

LP_ANA_VDDBAT_UPVOLTAGE_INT_ST VDD_BAT 电压恢复至欠压阈值电压时产生并发送该中断给 CPU。(RO)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_ST VDD_BAT 电压降低至欠压阈值电压时产生并发送该中断给 CPU。(RO)

LP_ANA_BOD_MODE0_INT_ST 产生模式 0 欠压中断并发送该中断给 CPU。(RO)

Register 25.7. LP_ANA_INT_ENA_REG (0x0028)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_ENA 配置是否使能 VDD_BAT 电压恢复至充电阈值电压时产生的中断。

0：否

1：是

(R/W)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_ENA 配置是否使能 VDD_BAT 电压降低至充电阈值电压时产生的中断。

0：否

1：是

(R/W)

LP_ANA_VDDBAT_UPVOLTAGE_INT_ENA 配置是否使能 VDD_BAT 电压恢复至欠压阈值电压时产生的中断。

0：否

1：是

(R/W)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_ENA 配置是否使能 VDD_BAT 电压降低至欠压阈值电压时产生的中断。

0：否

1：是

(R/W)

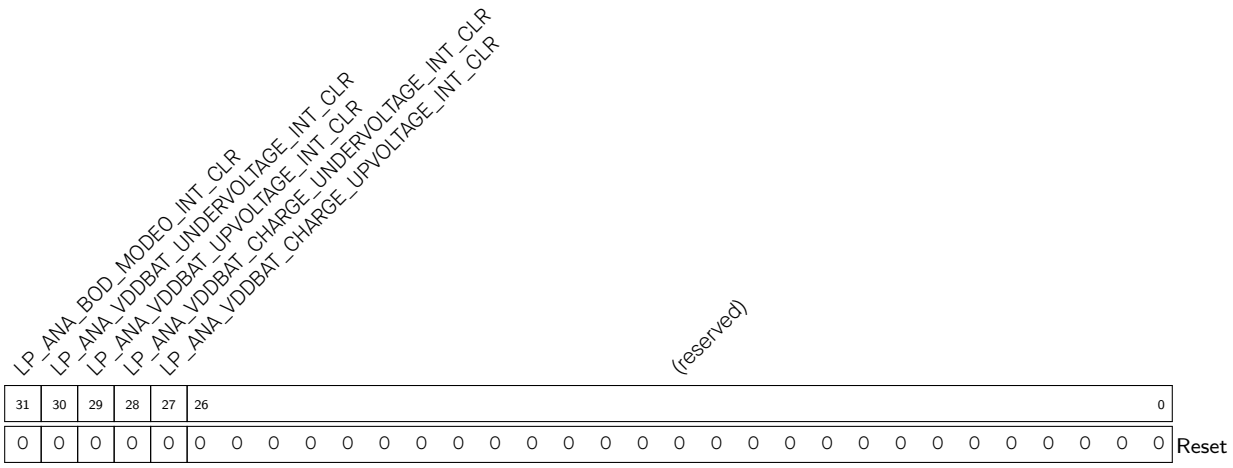
LP_ANA_BOD_MODE0_INT_ENA 配置是否使能模式 0 欠压中断。

0：否

1：是

(R/W)

Register 25.8. LP_ANA_INT_CLR_REG (0x002C)



LP_ANA_VDDBAT_CHARGE_UPVOLTAGE_INT_CLR 配置是否清除 VDD_BAT 电压恢复至充电阈值电压时产生的中断。(WT)

LP_ANA_VDDBAT_CHARGE_UNDERVOLTAGE_INT_CLR 配置是否清除 VDD_BAT 电压降低至充电阈值电压时产生的中断。(WT)

LP_ANA_VDDBAT_UPVOLTAGE_INT_CLR 配置是否清除 VDD_BAT 电压恢复至欠压阈值电压时产生的中断。(WT)

LP_ANA_VDDBAT_UNDERVOLTAGE_INT_CLR 配置是否清除 VDD_BAT 电压降低至欠压阈值电压时产生的中断。(WT)

LP_ANA_BOD_MODE0_INT_ST 配置是否清除模式 0 欠压中断。(WT)

第 26 章

电源检测器 (PSDET)

26.1 概述

ESP32-P4 芯片集成了电源检测器，用于监测片上 HP 系统调压器的电压。具体而言，该功能由一个电压毛刺检测器实现，通过实时监测供电电压，预防异常波动和毛刺攻击。电源检测器可以在 HP 系统工作期间持续工作。

26.2 主要特性

- 在芯片正常工作阶段保持监测，不需要软件干预
- 当监测到约 50 ns 以上的电压毛刺后，自动触发系统复位

26.3 功能描述

基于电压毛刺检测的电源检测器的结构如下图所示：



图 26.3-1. ESP32-P4 电源检测器结构

ESP32-P4 的电源检测器可以在芯片工作周期内保持检查供电网络上的电压，当监测到波动深度大于 100 mV，且持续时间大于 50 ns 的瞬时电压毛刺时，会自动触发系统复位，检测过程无需寄存器配置。由该功能触发的系统复位会被 HP CPU0 复位源LP_CLKRST_HPCORE0_RESET_CAUSE 记录，对应 0x1B 复位源。

该功能默认关闭，可通过烧写 EFUSE_PVT_GLITCH_EN 与 EFUSE_PVT_GLITCH_MODE 开启功能。更多有关 eFuse 的信息，请参考章节 9.4 *eFuse 控制器 (EFUSE)*。

更多有关复位的信息，请参考章节 11 复位和时钟。

第 V 卷

加密和安全组件

该部分专门介绍芯片的安全功能，包括 AES、ECC 等加密加速器，数字签名、随机数生成、加密/解密算法等安全特性，展示了芯片在加密和安全数据处理方面的性能。

第 27 章

AES 加速器 (AES)

27.1 概述

ESP32-P4 内置 AES（高级加密标准）硬件加速器可使用 AES 算法，完成数据的加解密运算，具有 Typical AES 和 DMA-AES 两种工作模式。整体而言，相比基于纯软件的 AES 运算，AES 硬件加速器能够极大地提高运算速度。

27.2 主要特性

ESP32-P4 支持以下特性：

- Typical AES 工作模式
  - AES-128/AES-256 加解密运算
- DMA-AES 工作模式
  - AES-128/AES-256 加解密运算
  - 块（加密）模式
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - 伽罗瓦/计数器模式 (Galois/Counter Mode, GCM)
  - 中断发生

27.3 时钟和复位

AES 加速器的激活需使能 HP_SYS_CLKRST_PERI_CLK_CTRL25_REG 寄存器中的 HP_SYS_CLKRST_CRYPTO_AES_CLK_EN 位，并同时清零HP_SYS_CLKRST_HP_RST_EN2_REG 寄存器中的 HP_SYS_CLKRST_RST_EN_AES 位。此外，由于密码学加速器模块之间的资源复用，用户还需要额外清零 HP_SYS_CLKRST_HP_RST_EN2_REG 寄存器中的 HP_SYS_CLKRST_RST_EN_DS 位和 HP_SYS_CLKRST_RST_EN_KM 位。

27.4 工作模式简介

ESP32-P4 内置的 AES 加速器支持 Typical AES 和 DMA-AES 两种工作模式。

- Typical AES 工作模式：
  - 支持使用 128 位或 256 位密钥进行加密与解密运算，即 NIST FIPS 197 标准中的 AES-128 和 AES-256 加解密运算。

  这种情况下，明文和密文的读写操作统一通过 CPU 访问完成。
- DMA-AES 工作模式支持：
  - 使用 128 位或 256 位密钥进行加密与解密运算，即 NIST FIPS 197 标准中的 AES-128 和 AES-256 加解密运算；
  - NIST SP 800-38A 标准中的 ECB、CBC、OFB、CTR、CFB8、CFB128 块加密模式运算。
  - NIST SP 800-38D 标准中的 GCM。

  在这种情况下，明文和密文的传输通过硬件上的 DMA 完成，运算完成时会有中断发生。

用户可通过配置 AES_DMA_ENABLE_REG 选择 AES 加速器的工作模式，具体参考表 27.4-1。

表 27.4-1. 工作模式

AES_DMA_ENABLE_REG	工作模式
0	Typical AES
1	DMA-AES

用户可通过配置 AES_MODE_REG 寄存器选择密钥长度和加解密方向，具体可参考表 27.4-2。

表 27.4-2. 密钥长度和加解密方向

AES_MODE_REG[2:0]	密钥长度和加解密方向
0	AES-128 加密
1	保留
2	AES-256 加密
3	保留
4	AES-128 解密
5	保留
6	AES-256 解密
7	保留

有关 Typical AES 和 DMA-AES 两种工作模式的具体介绍，请见章节 27.5 和 27.6。

注意：

ESP32-P4 的RSA 数字签名外设 (RSA_DS) 模块也会调用 AES 加速器。因此，在该模块工作时，用户无法正常访问 AES 加速器。

27.5 Typical AES 工作模式

在 Typical AES 工作模式下，AES 加速器的状态值可查看寄存器 AES_STATE_REG，具体见表 27.5-1 所示：

表 27.5-1. 状态返回值

返回值	描述	状态说明
0	IDLE	加速器空闲或运算完成
1	WORK	加速器忙于运算

27.5.1 密钥、明文、密文

寄存器 AES_KEY_*n*_REG 用于存放密钥，由 8 个 32 位寄存器组成。

- 如果为 AES-128 加解密运算，则 128 位密钥在寄存器 AES_KEY_0_REG ~ AES_KEY_3_REG 中。
- 如果为 AES-256 加解密运算，则 256 位密钥在寄存器 AES_KEY_0_REG ~ AES_KEY_7_REG 中。

寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG 用于存放明文和密文，各由 4 个 32 位寄存器组成。

- 如果为 AES-128 或 AES-256 加密运算，则运算开始之前用明文初始化寄存器 AES_TEXT_IN_*m*_REG。运算完成之后，AES 加速器将把密文更新入寄存器 AES_TEXT_OUT_*m*_REG。
- 如果为 AES-128 或 AES-256 解密运算，则运算开始之前用密文初始化寄存器 AES_TEXT_IN_*m*_REG。运算完成之后，AES 加速器将把明文更新入寄存器 AES_TEXT_OUT_*m*_REG。

27.5.2 字节序

文本字节序

在 Typical AES 工作模式下，AES 加速器可以使用密钥对 128 位的块进行加解密。在读写寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG 中的数据时，用户应遵循表 27.5-2 中定义的文本字节序。

表 27.5-2. Typical AES 文本字节序

明文/密文					
State[1]		c[2]			
		0	1	2	3
r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
	1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
	2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]
	3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

[1] 有关 “State（以及 c 和 r）” 的详细定义，请参考 NIST FIPS 197 中 3.4 The State 章节。
[2] 其中，*x* = IN 或 OUT。

密钥字节序

在 Typical AES 工作模式下，在向寄存器 AES_KEY_*n*_REG 中填入数据时，用户应遵循表 27.5-3 和表 27.5-4 中定义的文本字节序。

表 27.5-3. AES-128 密钥字节序

Bit[1]	w[0]	w[1]	w[2]	w[3][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

[1] Bit 列代表 w[0] ~ w[3] 每个 word 中的各个字节。

[2] w[0] ~ w[3] 符合标准 NIST FIPS 197 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

表 27.5-4. AES-256 密钥字节序

Bit[1]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

[1] Bit 列代表 w[0] ~ w[7] 每个 word 中的各个字节。

[2] w[0] ~ w[7] 符合标准 NIST FIPS 197 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

27.5.3 Typical AES 工作流程

单次运算

1. 对寄存器 AES_DMA_ENABLE_REG 写入 0。
2. 初始化寄存器 AES_MODE_REG、AES_KEY_*n*_REG、AES_TEXT_IN_*m*_REG。
3. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
4. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 0。
5. 从寄存器 AES_TEXT_OUT_*m*_REG 读取结果。

连续运算

在连续运算过程中，每次运算完成之后，只有寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG (*m*: 0-3) 会被 AES 加速器更新，而 AES_DMA_ENABLE_REG、AES_MODE_REG、AES_KEY_*n*_REG 等寄存器中的内容不会变化。所以进行连续运算时可以简化初始化操作。

1. 第一次运算之前对寄存器 AES_DMA_ENABLE_REG 写入 0。
2. 第一次运算之前初始化寄存器 AES_MODE_REG 和 AES_KEY_*n*_REG。
3. 更新寄存器 AES_TEXT_IN_*m*_REG。
4. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
5. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 0。
6. 从寄存器 AES_TEXT_OUT_*m*_REG 读取结果 。返回步骤 3，进行下一轮运算。

27.6 DMA-AES 工作模式

在 DMA-AES 工作模式下，AES 加速器可支持 ECB、CBC、OFB、CTR、CFB8、CFB128 六种块模式运算。用户可通过配置 AES_BLOCK_MODE_REG 寄存器选择具体运算类型，具体可参考表 27.6-1。

表 27.6-1. 块模式选择

AES_BLOCK_MODE_REG[2:0]	块模式
0	ECB (Electronic Code Book)
1	CBC (Cipher Block Chaining)
2	OFB (Output FeedBack)
3	CTR (Counter)
4	CFB8 (8-bit Cipher FeedBack)
5	CFB128 (128-bit Cipher FeedBack)
6	GCM (Galois/Counter Mode)
7	保留

AES 加速器的状态值可查看寄存器 AES_STATE_REG，具体见表 27.6-2 所示：

表 27.6-2. 状态返回值

返回值	描述	状态说明
0	IDLE	加速器空闲
1	WORK	加速器忙于运算
2	DONE	加速器运算完成

AES 加速器在 DMA-AES 工作模式下允许中断发生，软件清零。中断功能默认关闭，用户可通过将 AES_INT_ENA_REG 寄存器配置为 1 开启中断。如开启中断功能，AES 加速器在完成运算时，中断发生。

27.6.1 密钥、明文、密文

块运算模式

在块运算模式下，AES 加速器的源数据来自 DMA，结果数据也将被写入 DMA。

- 如果为加密运算，则 DMA 从存储器中读取明文数据流并将其传给 AES。AES 运算出密文后将密文写入 DMA。DMA 再将密文写入存储器。
- 如果为解密运算，则 DMA 从存储器中读取密文数据流并将其传给 AES。AES 运算出明文后将明文写入 DMA。DMA 再将明文写入存储器。

AES 加速器在进行块运算时，结果数据与源数据的大小保持一致。此时，DMA 的数据搬运过程和 AES 的运算过程有所交叠，因此总工作时间有所减少。

值得注意的是，AES 加速器在 DMA-AES 工作模式下要求源数据的大小必须是 128 位的整数倍，否则需要将原始明文封装为 128 位的整数倍，即在源数据末尾应尽可能少补 0，具体过程见表 27.6-3 所示。

表 27.6-3. TEXT-PADDING

函数： TEXT-PADDING( )
输入： X，比特串。 **输出：** Y = TEXT-PADDING(X)，长度为最接近 128 比特整数倍的值。
步骤： 假设 X 是一个数据流，可以按如下方式分成 n 个部分： $X = X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n$ 此处，X_1、X_2、$\cdots$、X_{n-1} 的长度均为 128 比特，而 X_n 的长度为 t (0<=t<=127)。 如果 $t = 0$，则 TEXT-PADDING(X) = X; 如果 $0 < t <= 127$，定义 X_n^* 块的长度为 128 比特，且 $X_n^* = X_n \| \| 0^{128-t}$，则 TEXT-PADDING($X$) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^*$ = $X \| \| 0^{128-t}$

27.6.2 字节序

在 DMA-AES 工作模式下，源数据和结果数据的传输完全由 DMA 完成，因此不支持字节序的控制调节，但要求它们在存储器中以一定的方式来存放，且要求数据量必须是块的整数倍。

举例说明，假设 DMA 需要搬运源数据：

- 十六进制：0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- 数据大小：相当于 2 个块

假设起始地址为 0x0280，则源数据在存储器中的存放位置如表 27.6-4 所示。结果数据也遵从相同的存放规则，在此不多做介绍。

表 27.6-4. DMA-AES 存储字节序

地址	字节	地址	字节	地址	字节	地址	字节
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

27.6.3 标准增量函数

AES 加速器在进行 CTR 块运算时，还可提供两种标准增量函数供用户选择：INC_{32} 和 INC_{128}。用户可通过将寄存器 AES_INC_SEL_REG 置为 0 或 1 选择 INC_{32} 或 INC_{128} 标准增量函数。更多有关标准增量函数的内容，请见 NIST SP 800-38A 标准中的 B.1 The Standard Incrementing Function 章节。

27.6.4 块个数

寄存器 AES_BLOCK_NUM_REG 存放明文或密文的块个数 (Block Number)，其值等于 length(**TEXT-PADDING**(P))/128，也等于 length(**TEXT-PADDING**(C))/128。这里的 P 指明文 (plaintext)，C 指密文 (ciphertext)。该寄存器仅在 DMA-AES 工作模式下有意义。

27.6.5 初始向量

存储器 AES_IV_MEM 的空间大小为 16 字节，仅在块运算模式下有效。对于 CBC/OFB/CFB8/CFB128 操作，AES_IV_MEM 用于存放初始向量 (Initialization Vector, IV) 的值。对于 CTR 操作，AES_IV_MEM 存放初始计数器 (Initial Counter Block, ICB) 的值。

IV 和 ICB 都是 128 位的比特串，从左向右被分割成 16 个字节 (Byte0, Byte1, Byte2, ⋯, Byte15)，构成一个字节序列，在 AES_IV_MEM 中存放时需要遵循表 27.6-4 中的字节序规则，即 Byte0 存放在 AES_IV_MEM 中的最低地址中，Byte15 存放在 AES_IV_MEM 中的最高地址中。

更多有关 IV 和 ICB 的信息，请参考 NIST SP 800-38A 标准。

27.6.6 块工作流程

1. 选择一条 DMA 通道与 AES 加速器连接，配置 DMA 链表，而后启动 DMA。详情请见章节 6 *通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*。
2. 初始化与 AES 加速器相关的寄存器：
    - 对寄存器 AES_DMA_ENABLE_REG 写入 1。

- 配置寄存器 AES_INT_ENA_REG 以开启或禁用中断。
- 初始化寄存器 AES_MODE_REG 和 AES_KEY_*n*_REG。
- 配置寄存器 AES_BLOCK_MODE_REG，选择具体块加密模式。详见表 27.6-1。
- 初始化寄存器 AES_BLOCK_NUM_REG，请参照章节 27.6.4。
- 初始化寄存器 AES_INC_SEL_REG（仅在 CTR 块模式下使用）。
- 初始化存储器 AES_IV_MEM（在 ECB 块模式下不使用）。

3. 对寄存器 AES_TRIGGER_REG 写入 1，启动运算。
4. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 2。如果开启了中断功能，也可以等待 AES_INT 中断产生。
5. 确认 DMA 完成从 AES 到内存的数据传输。此时，结果数据已经被 DMA 写入内存，可以直接读取。详情请参考章节 6 *通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*。
6. 如果开启了中断，当处理中断程序完成后，请及时对寄存器 AES_INT_CLR_REG 写 1 以清除中断。
7. 对寄存器 AES_DMA_EXIT_REG 写入 1 释放 AES 加速器，再读取寄存器 AES_STATE_REG 将读到 0。该步操作可以提前完成，但必须在步骤 4 之后完成。

27.6.7 GCM 工作流程

1. 配置 DMA 链并启动 DMA。有关 DMA 的详细信息，请见章节 6 *通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*。
2. 初始化与 AES 加速器相关的寄存器：
   - 对寄存器 AES_DMA_ENABLE_REG 写入 1。
   - 配置寄存器 AES_INT_ENA_REG 以开启或禁用中断。
   - 初始化寄存器 AES_MODE_REG 和 AES_KEY_*n*_REG。
   - 对寄存器 AES_BLOCK_MODE_REG 写入 6。
   - 初始化寄存器 AES_BLOCK_NUM_REG，详情请见章节 27.6.4 *块个数*。
   - 初始化寄存器 AES_AAD_BLOCK_NUM_REG，详情请见章节 27.7.4 *AAD 块个数*。
   - 初始化寄存器 AES_REMAINDER_BIT_NUM_REG ，详情请见章节 27.7.5 *不完整块的有效位数*。
3. 对寄存器 AES_TRIGGER_REG 写入 1，启动运算。
4. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 2。有关 AES 加速器工作状态的详细信息，请参考表 27.6-2 *状态返回值*。此步骤中不会产生中断。
5. 从存储器 AES_H_MEM 中读取 H。
6. 生成 J_0 并将其写入存储器 AES_J0_MEM。
7. 对寄存器 AES_CONTINUE_OP_REG 写入 1 以继续操作。
8. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 2。如果开启了中断功能，也可以等待 AES_INT 中断产生。有关 AES 加速器工作状态的详细信息，请参考表 27.6-2 *状态返回值*。
9. 从存储器 AES_T0_MEM 中读取 T_0。

10. 确认 DMA 完成从 AES 到内存的数据传输。此时，结果数据已经被 DMA 写入内存，可以直接读取。详情请参考章节 6 通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)。
11. 如果开启了中断，当处理中断程序完成后，请及时对寄存器 AES_INT_CLR_REG 写 1 以清除中断。
12. 对寄存器 AES_DMA_EXIT_REG 写入 1 退出 DMA，再读取寄存器 AES_STATE_REG 将读到 0。该步操作可以提前完成，但必须在步骤 8 之后完成。

27.7 GCM 算法

ESP32-P4 的 AES 加速器支持 GCM 算法。通常情况下，很少会使用超过 2^{32}-1 位长度的 AAD、C 和 P。因此，本章节规定 AAD、C 和 P 的长度不应超过 2^{32}-1。下图 27.7-1 为 ESP32-P4 AES 加速器中的 GCM 加密过程。

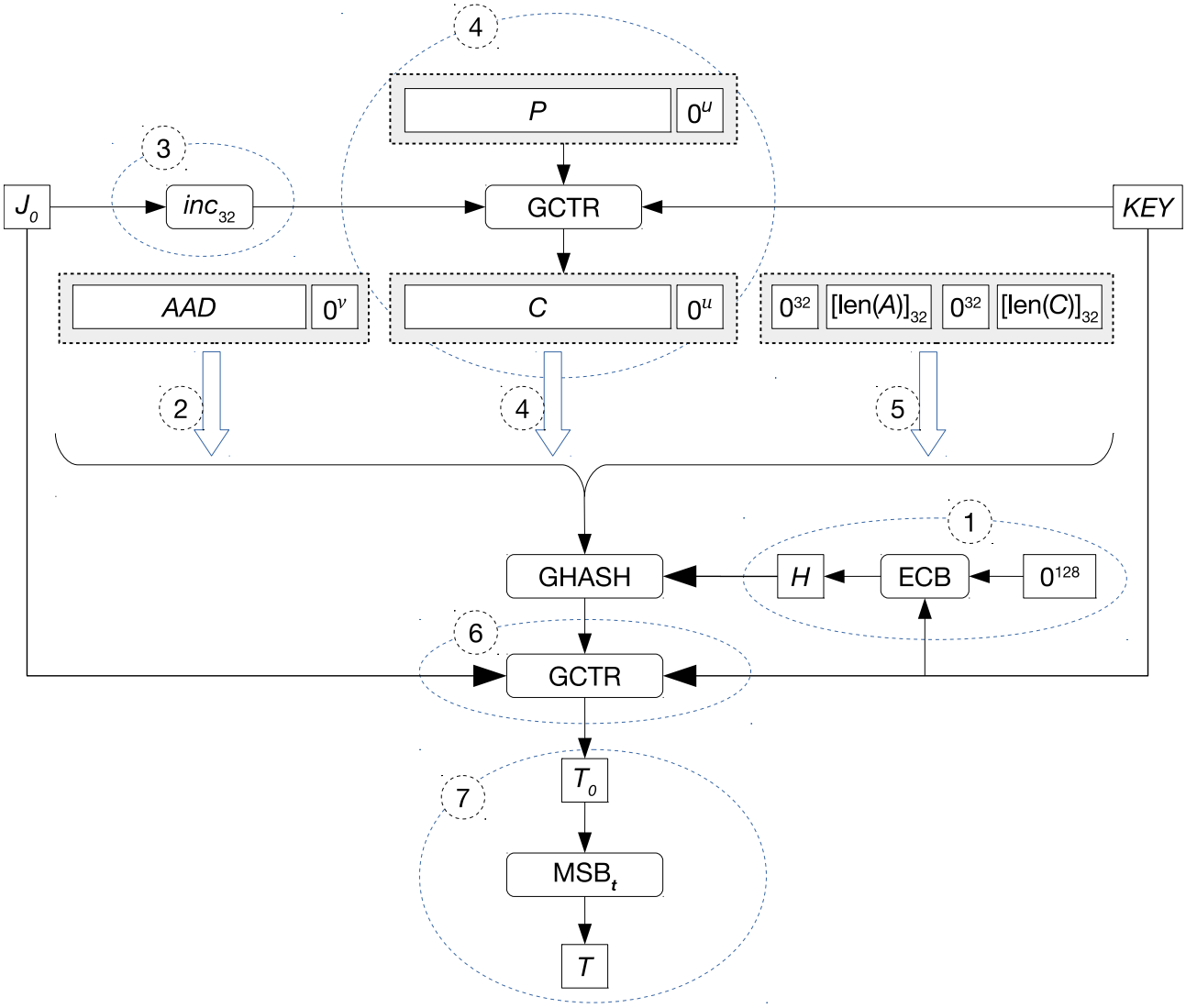


图 27.7-1. GCM 加密过程

GCM 加密过程如下：

1. 硬件执行 ECB 算法以获取哈希子密钥 H，该子密钥将用于哈希运算。
2. 硬件执行 GHASH 算法，使用 已填充的 AAD 进行哈希运算。

3. 硬件以 J_0 作为输入执行标准递增函数 INC_{32}，该结果将作为 CTR 加密的初始计数器值。
4. 硬件执行 GCTR 算法加密已填充的明文 P，然后执行 GHASH 算法使用已填充的密文 C 进行哈希运算。
5. 硬件执行 GHASH 算法对 AAD 块进行哈希运算，得到 128 位的哈希结果。
6. 硬件执行 GCTR 算法加密 J_0，得到 T_0。
7. 软件从硬件获取结果 T_0，执行 MSB_t 算法，得到最终的认证标签 T。

GCM 解密与 GCM 加密唯一的区别在于图 27.7-1 中的步骤 4。具体而言，在 GCM 解密中，AES 加速器不是执行 GCTR 算法加密已填充的明文，而是执行这一算法解密已填充的密文，详见 NIST SP 800-38D。

27.7.1 哈希子密钥

在 GCM 操作中，哈希子密钥 H 是硬件运算出的 128 位值，详见图 27.7-1 中的步骤 1。更多信息请参考 NIST SP 800-38D 的 7 GCM Specification 章节中关于 "Step 1. Let H = $\text{CIPH}_K(0^{128})$" 的描述。

哈希子密钥 H 存放在存储器 AES_H_MEM 中。与其他字节序相同，其最高有效字节（即最左侧字节）Byte0 存放在存储器的最低地址中，而最低有效字节（即最右侧字节）Byte15 存放在最高地址中，详见表 27.5-2。

27.7.2 J_0

J_0 是硬件运算出的 128 位值，用于图 27.7-1 的步骤 3 和步骤 6。有关生成 J_0 的详细信息，请参考 NIST SP 800-38D 的 7 GCM Specification 章节。

J_0 存放在存储器 AES_J0_MEM 中。与其他字节序相同，其最高有效字节（即最左侧字节）Byte0 存放在存储器的最低地址中，而最低有效字节（即最右侧字节）Byte15 存放在最高地址中，详见表 27.5-2。

27.7.3 认证标签

认证标签（下文简称标签）是 GCM 的关键输出之一，详见图 27.7-1 的步骤 7。标签值由其长度 t ($1 <= t <= 128$) 决定：

- 当 $t = 128$ 时，标签值等于 128 位的字符串 T_0，存放在 AES_T0_MEM 中。与其他字节序相同，其最高有效字节（即最左侧字节）Byte0 存放在存储器的最低地址中，而最低有效字节（即最右侧字节）Byte15 存放在最高地址中，详见表 27.5-2。
- 当 $1 <= t < 128$ 时，标签值等于 T_0 的 t 位最高有效位（即最左侧位）。在这种情况下的标签为 $\text{MSB}_t(T_0)$，代表 T_0 的 t 位最高有效位。例如，MSB_4(11011010) = 1110，MSB_5(11010011010) = 11010。有关 $\text{MSB}_t(\,)$ 函数的详细信息，请参考 NIST SP 800-38D 中的 6 Mathematical Components of GCM 章节。

27.7.4 AAD 块个数

寄存器 AES_AAD_BLOCK_NUM_REG 存储附加认证数据 (AAD) 的块个数。该寄存器的长度等于 **TEXT-PADDING**(AAD) 长度)/128。AES 加速器在 DMA-AES 模式下工作时才会使用此寄存器。

27.7.5 不完整块的有效位数

寄存器 AES_REMAINDER_BIT_NUM_REG 表示明文或密文中不完整块的有效位数。该寄存器中存储的值等于 P 长度/128 或 C 长度/128。AES 加速器在 DMA-AES 模式下工作时才会使用此寄存器。

AES_REMAINDER_BIT_NUM_REG 不影响明文或密文的相关运算结果，但会影响 T_0 的值，因此也会影响认证标签的值。

GCM 算法可以看作是 GCTR 操作和 GHASH 操作的结合，其中 GCTR 执行加密和解密，而 GHASH 运算认证标签的值。

请注意，AES_REMAINDER_BIT_NUM_REG 寄存器仅对 GCM 加密有效。具体而言：

- 对于 GCM 加密来说，硬件首先运算 C，然后将其以 **TEXT-PADDING**(C) 的形式传递给 GHASH 操作。此时，硬件根据 AES_REMAINDER_BIT_NUM_REG 的值确定在数据末尾应添加多少 0。
- GCM 解密使用 **TEXT-PADDING**(C) 函数完成填充。此时，AES_REMAINDER_BIT_NUM_REG 寄存器无效。

27.8 伪轮次抗 DPA 功能

ESP32-P4 AES 模块的伪轮次抗 DPA 功能可以增加芯片的抵御攻击的性能。当开启伪轮次功能时，AES 模块将在原始操作轮次之前和之后随机添加伪轮次。在一个伪轮次中，AES 将使用由硬件自动生成的伪密钥执行伪计算。伪轮次中的计算不会影响原始 AES 操作的结果。

伪轮次抗 DPA 功能的主要配置参数包括：

伪轮次功能的开关

用户可通过配置 AES_PSEUDO_EN 寄存器，选择是否开启伪轮次功能。

- 0：计算过程中不增加伪轮次；
- 1：计算过程中增加伪轮次。

伪轮次的总数

在每次 AES 计算中随机插入伪轮次的具体数量可通过以下寄存器配置：

- pseudo_base：等于 AES_PSEUDO_BASE 中配置的值，该值配置伪轮次功能的基本轮数。
- pseudo_inc：等于 AES_PSEUDO_INC 中配置的值，该值配置伪轮次功能的随机递增轮数。

伪轮次的总数将在

$$[pseudo_base, pseudo_base + (2^{pseudo_inc} - 1)]$$

范围内随机生成。

随机数更新频率

用户可通过配置 AES_PSEUDO_RNG_CNT 的值来确定伪轮次功能中随机密钥的更新频率。配置的值越大，代表更新频率越高。通常建议将此值配置为最大值 7。

27.9 中断

ESP32-P4 的 AES 可以生成以下中断信号并发送给 中断矩阵。

- AES_INTR

以上中断信号由 AES 的内部中断源生成。表 27.9-1 列出了 AES 的内部中断源、中断触发条件、以及生成的中断信号。

表 27.9-1. AES 的内部中断源

内部中断源	触发条件	中断信号
AES_DMA_DONE_INT	AES DMA 运算完成	AES_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

27.10 存储器列表

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	大小	起始地址	结束地址	访问
AES_IV_MEM	存储器 IV	16 字节	0x0050	0x005F	R/W
AES_H_MEM	存储器 H	16 字节	0x0060	0x006F	RO
AES_J0_MEM	存储器 J0	16 字节	0x0070	0x007F	R/W
AES_T0_MEM	存储器 T0	16 字节	0x0080	0x008F	RO

27.11 寄存器列表

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

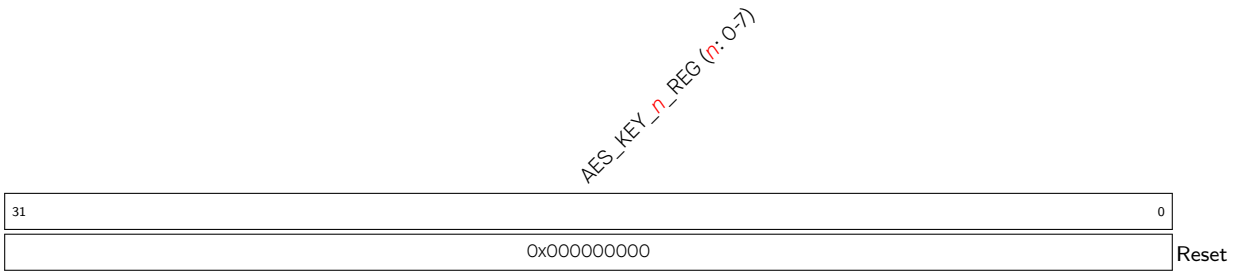
名称	描述	地址	访问
密钥寄存器			
AES_KEY_0_REG	AES 密钥资料寄存器 0	0x0000	R/W
AES_KEY_1_REG	AES 密钥资料寄存器 1	0x0004	R/W
AES_KEY_2_REG	AES 密钥资料寄存器 2	0x0008	R/W
AES_KEY_3_REG	AES 密钥资料寄存器 3	0x000C	R/W
AES_KEY_4_REG	AES 密钥资料寄存器 4	0x0010	R/W
AES_KEY_5_REG	AES 密钥资料寄存器 5	0x0014	R/W
AES_KEY_6_REG	AES 密钥资料寄存器 6	0x0018	R/W
AES_KEY_7_REG	AES 密钥资料寄存器 7	0x001C	R/W
TEXT_IN 寄存器			
AES_TEXT_IN_0_REG	源数据资料寄存器 0	0x0020	R/W
AES_TEXT_IN_1_REG	源数据资料寄存器 1	0x0024	R/W
AES_TEXT_IN_2_REG	源数据资料寄存器 2	0x0028	R/W
AES_TEXT_IN_3_REG	源数据资料寄存器 3	0x002C	R/W
TEXT_OUT 寄存器			
AES_TEXT_OUT_0_REG	结果数据资料寄存器 0	0x0030	RO
AES_TEXT_OUT_1_REG	结果数据资料寄存器 1	0x0034	RO
AES_TEXT_OUT_2_REG	结果数据资料寄存器 2	0x0038	RO
AES_TEXT_OUT_3_REG	结果数据资料寄存器 3	0x003C	RO
控制或配置寄存器			
AES_MODE_REG	配置密钥长度和加解密方向	0x0040	R/W
AES_DMA_ENABLE_REG	配置 AES 加速器工作模式	0x0090	R/W
AES_BLOCK_MODE_REG	配置 DMA-AES 下的块运算模式	0x0094	R/W
AES_BLOCK_NUM_REG	配置块数量	0x0098	R/W
AES_INC_SEL_REG	配置标准增量函数	0x009C	R/W
AES_TRIGGER_REG	开始运算	0x0048	WT
AES_CONTINUE_OP_REG	继续运算	0x00A8	WO
AES_DMA_EXIT_REG	退出运算	0x00B8	WO
AES_PSEUDO_REG	配置伪轮次抗攻击功能	0x00D0	R/W
状态寄存器			
AES_STATE_REG	运算状态	0x004C	RO
中断寄存器			
AES_INT_CLR_REG	DMA-AES 中断清除	0x00AC	WT
AES_INT_ENA_REG	DMA-AES 中断使能	0x00B0	R/W

27.12 寄存器

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

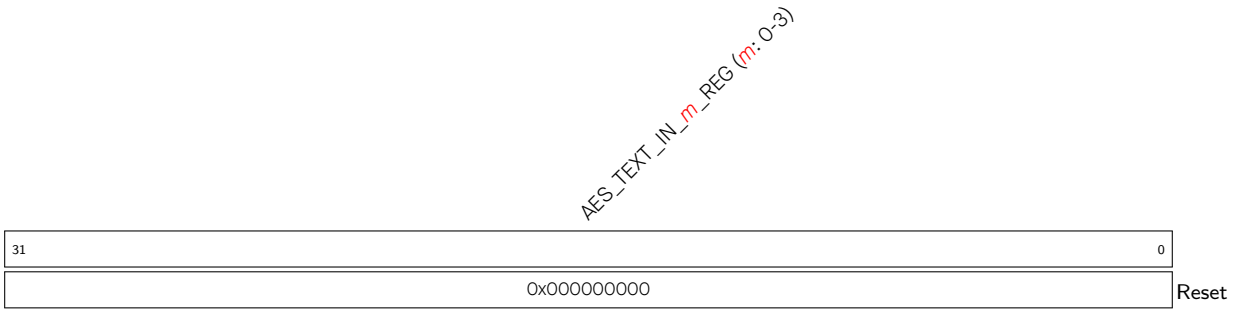
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 27.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4**n*)



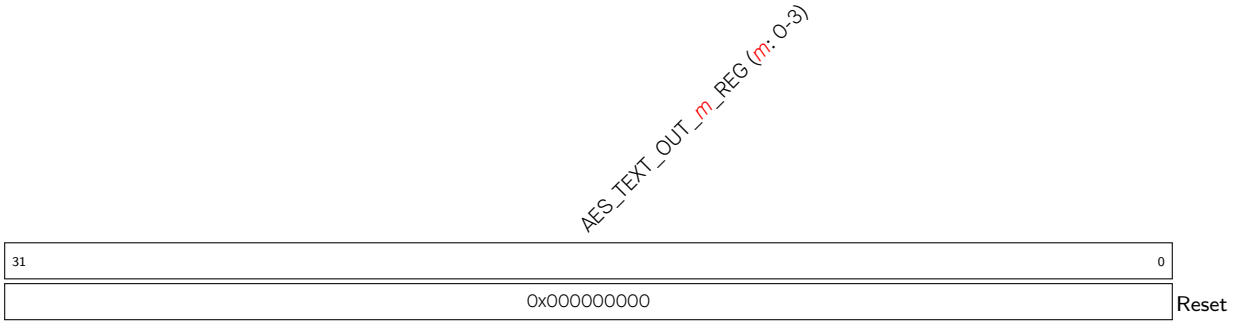
AES_KEY_*n*_REG (*n*: 0-7) 表示 AES 密钥资料。(R/W)

Register 27.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4**m*)



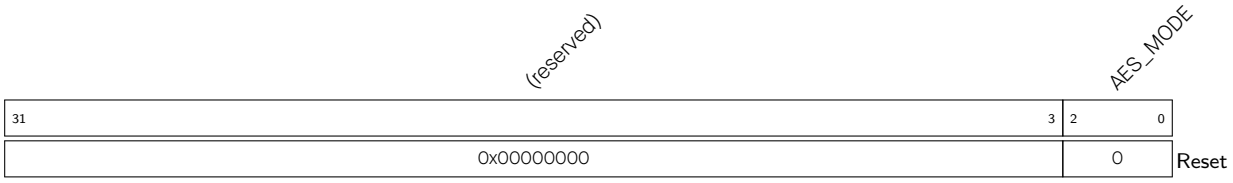
AES_TEXT_IN_*m*_REG (*m*: 0-3) 表示 Typical AES 工作模式下的输入文本。(R/W)

Register 27.3. AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4**m*)



AES_TEXT_OUT_*m*_REG (*m*: 0-3) 表示 Typical AES 工作模式下的输出文本。(RO)

Register 27.4. AES_MODE_REG (0x0040)



AES_MODE 配置 AES 加速器的密钥长度和加解密方向。

0：AES-128 加密

1：保留

2：AES-256 加密

3：保留

4：AES-128 解密

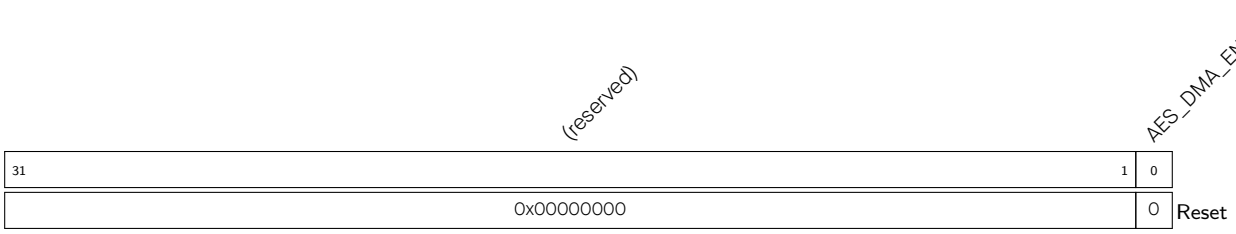
5：保留

6：AES-256 解密

7：保留

(R/W)

Register 27.5. AES_DMA_ENABLE_REG (0x0090)



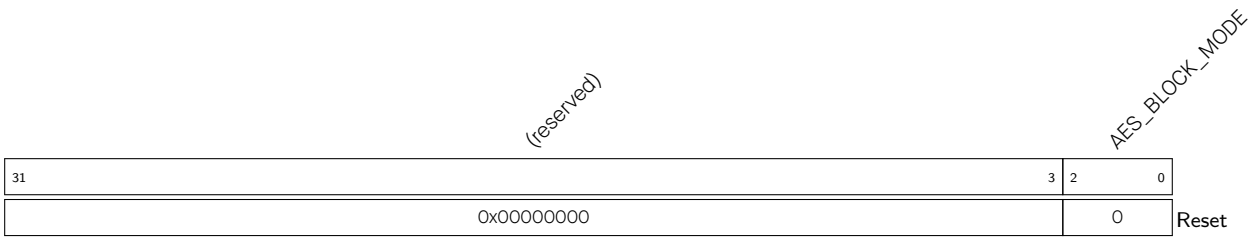
AES_DMA_ENABLE 配置 AES 加速器的工作模式。

0：Typical AES

1：DMA-AES

(R/W)

Register 27.6. AES_BLOCK_MODE_REG (0x0094)



AES_BLOCK_MODE 配置 AES 加速器在 DMA-AES 工作模式下的块模式。

0：ECB (Electronic Code Block)

1：CBC (Cipher Block Chaining)

2：OFB (Output FeedBack)

3：CTR (Counter)

4：CFB8 (8-bit Cipher FeedBack)

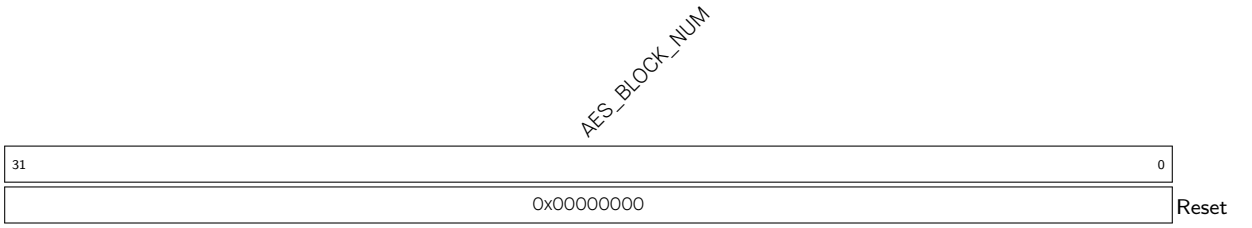
5：CFB128 (128-bit Cipher FeedBack)

6：GCM (Galois/Counter Mode)

7：保留

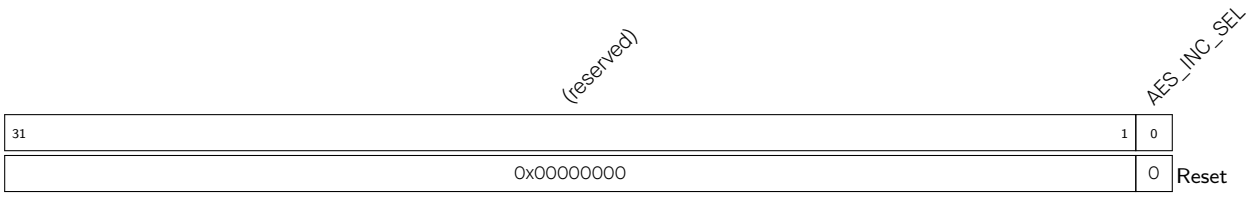
(R/W)

Register 27.7. AES_BLOCK_NUM_REG (0x0098)



AES_BLOCK_NUM 表示 DMA-AES 工作模式下待加解密的文本块数，详见章节 27.6.4。(R/W)

Register 27.8. AES_INC_SEL_REG (0x009C)



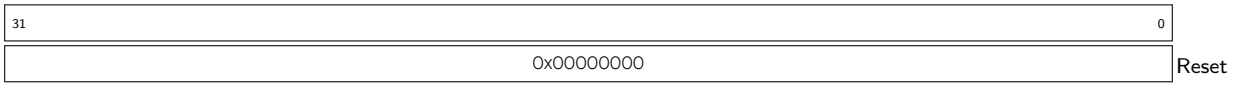
AES_INC_SEL 配置 CTR 块模式使用的标准增量函数。

0：INC_{32} 标准增量函数

1：INC_{128} 标准增量函数

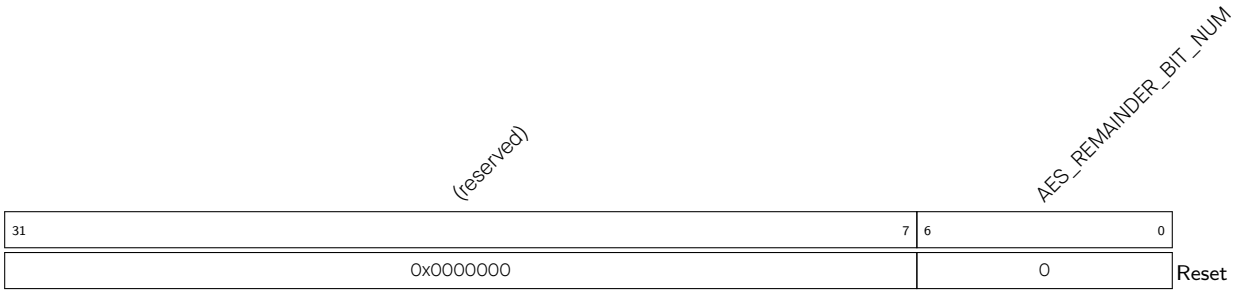
(R/W)

Register 27.9. AES_AAD_BLOCK_NUM_REG (0x00A0)



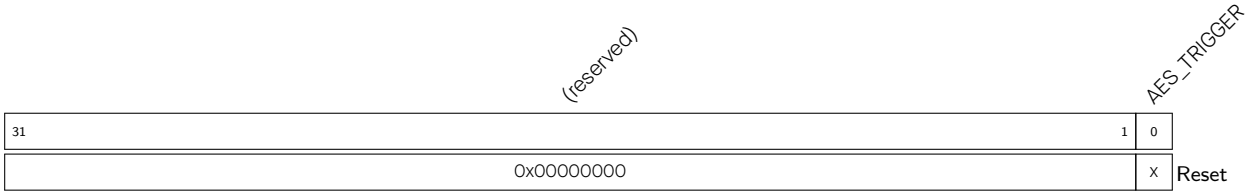
AES_AAD_BLOCK_NUM 表示 GCM 运算附加认证数据 (AAD) 的块个数，详见章节 27.7.4。(R/W)

Register 27.10. AES_REMAINDER_BIT_NUM_REG (0x00A4)



AES_REMAINDER_BIT_NUM 表示 GCM 运算不完整块的有效位数，详见章节 27.7.5。(R/W)

Register 27.11. AES_TRIGGER_REG (0x0048)



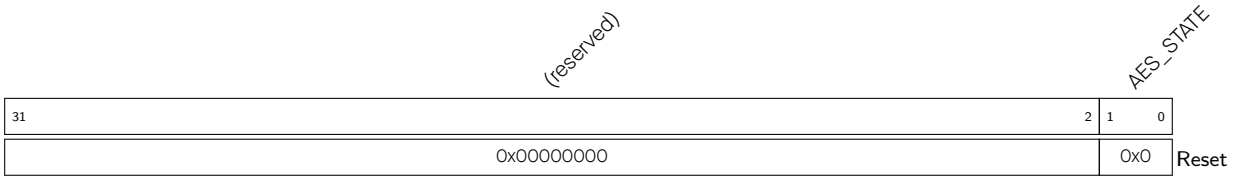
AES_TRIGGER 配置是否启动 AES 运算。

0：无效果

1：启动

(WT)

Register 27.12. AES_STATE_REG (0x004C)



AES_STATE 表示 AES 加速器的状态。

在 Typical AES 工作模式下，

0：IDLE

1：WORK

2：无效果

3：无效果

在 DMA-AES 工作模式下，

0：IDLE

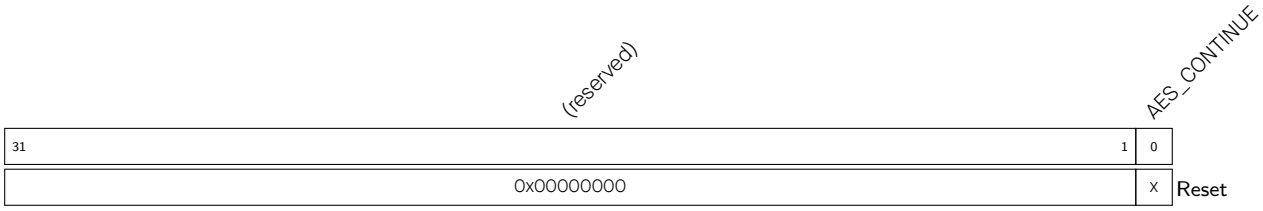
1：WORK

2：DONE

3：无效果

(RO)

Register 27.13. AES_CONTINUE_OP_REG (0x00A8)



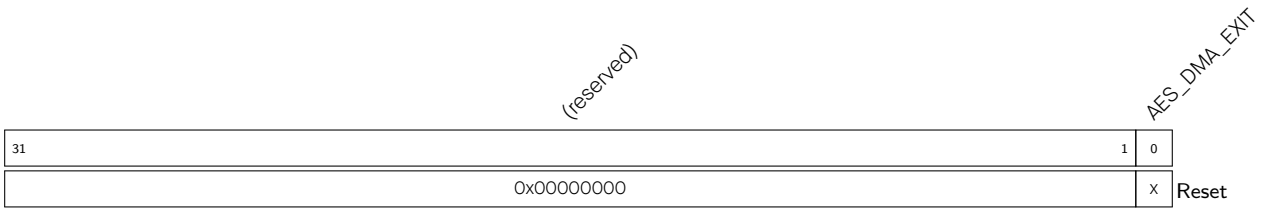
AES_CONTINUE_OP 配置是否继续 AES 运算。

0：无效果

1：继续

(WO)

Register 27.14. AES_DMA_EXIT_REG (0x00B8)



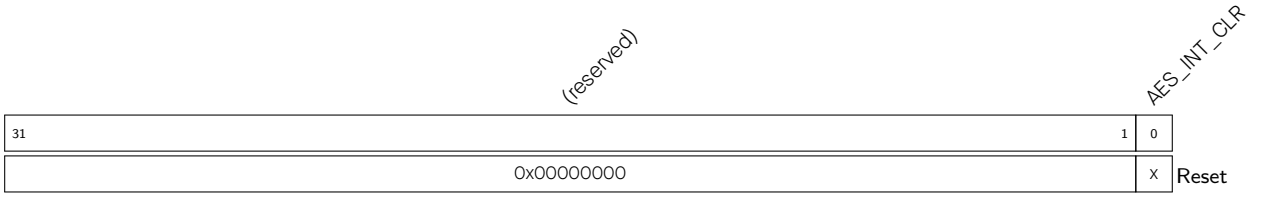
AES_DMA_EXIT 配置是否退出 AES 运算。

0：无效果

1：退出

仅在 DMA-AES 模式下有效。(WO)

Register 27.15. AES_INT_CLR_REG (0x00AC)



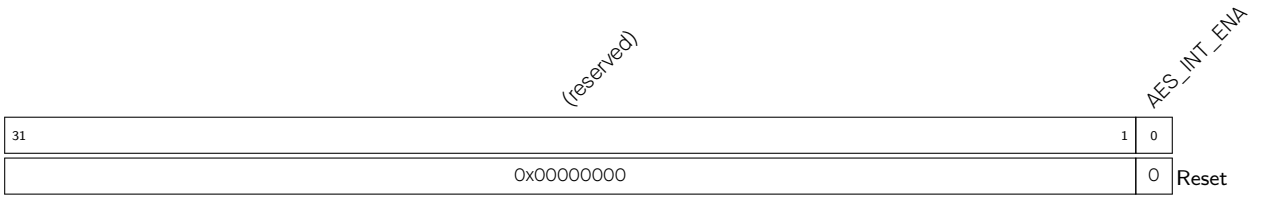
AES_INT_CLR 配置是否清除 AES 中断。

0：无效果

1：清除

(WT)

Register 27.16. AES_INT_ENA_REG (0x00B0)



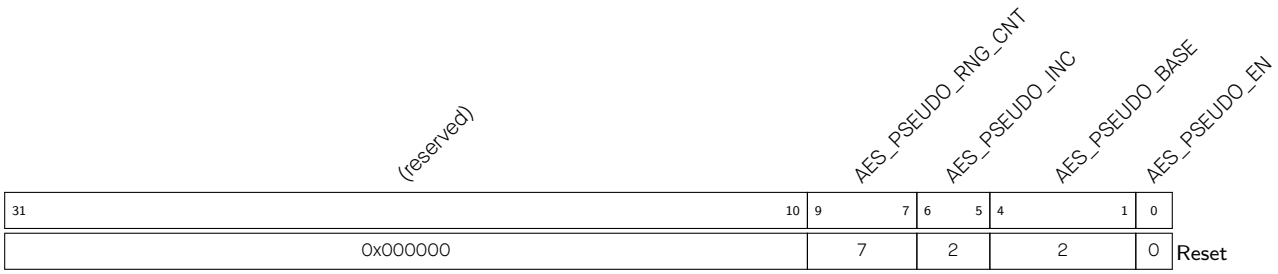
AES_INT_ENA 配置是否使能 AES 中断功能。

0：禁用

1：使能

(R/W)

Register 27.17. AES_PSEUDO_REG (0x00D0)



AES_PSEUDO_EN 配置增加伪轮次抗 DPA 计算的开启。

0：计算过程中不增加伪轮次

1：计算过程中增加伪轮次

(R/W)

AES_PSEUDO_BASE 配置伪轮次抗 DPA 计算中的基本轮数。(R/W)

AES_PSEUDO_INC 配置伪轮次抗 DPA 计算中的递增轮数。(R/W)

AES_PSEUDO_RNG_CNT 配置在伪轮次抗 DPA 计算中伪密钥的更新频率。(R/W)

第 28 章

ECC 加速器 (ECC)

28.1 概述

椭圆曲线密码学 (Elliptic Curve Cryptography, ECC) 是一种基于椭圆曲线数学的公开密钥加密演算法，其优势在于相对于 RSA 算法，使用较小长度的密钥就能够提供相当等级的加密安全性。

ESP32-P4 ECC 硬件加速器支持基于可选曲线的多种基础运算，用以实现对 ECC 基本运算及其衍生算法（如 ECDSA 等算法）的加速。

28.2 主要特性

ESP32-P4 ECC 硬件加速器具备以下特性：

- 支持三种可选 ECC 曲线，即 FIPS 186-5 中定义的 P-192、P-256 和 P-384
- 提供 11 种可选工作模式
- 提升抗攻击性能
- 提供计算完成的中断和中断控制

28.3 ECC 背景知识

本节介绍了与 ECC 硬件加速器相关的背景知识以及章节中会应用到的专业名词。

28.3.1 椭圆曲线与曲线上的点

ECC 是一种基于大素数的有限域椭圆曲线的算法，这一椭圆曲线的数学表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

其中，

- p 是素数，
- a 和 b 为两个小于 p 的非负整数，
- (x, y) 为满足该椭圆曲线方程的点。

28.3.2 仿射坐标系与 Jacobian 坐标系

一条椭圆曲线：

- 在仿射坐标系下的表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

- 在 Jacobian 坐标系下的表达式为：

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

一个曲线上的点在仿射坐标系下的表示 (x, y) 与在 Jacobian 坐标系下的表示 (X, Y, Z) 有如下转换关系：

- 从 Jacobian 坐标系到仿射坐标系的转换：

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- 从仿射坐标系到 Jacobian 坐标系的转换：

$$X = x$$

$$Y = y$$

$$Z = 1$$

28.3.3 内存块

ECC 硬件加速器的内存块用于存储 ECC 运算中所用到的输入数据和输出数据。

表 28.3-1. ECC 硬件加速器内存块

内存块	大小（字节）	起始地址*	结束地址*	访问权限
ECC_MULT_Mem_k	32	0x100	0x11F	R/W
ECC_MULT_Mem_Px	32	0x120	0x13F	R/W
ECC_MULT_Mem_Py	32	0x140	0x15F	R/W
ECC_MULT_Mem_Qx	32	0x160	0x17F	R/W
ECC_MULT_Mem_Qy	32	0x180	0x19F	R/W
ECC_MULT_Mem_Qz	32	0x1A0	0x1BF	R/W

* 采用相对于 ECC 加速器基地址的偏移量。详见章节 9 系统和存储器 中的表 9.3-2。

28.3.4 数据与数据块

在 ECC 硬件加速器模块中会用到的数据位宽可能为 192、256 或 384 位，这取决于进行运算的曲线。

假设一个数据为 $D[255:0]$，则其可以被划分为八个 32 位的数据块 $D[n][31:0](n = 0, 1, \cdots, 7)$，序号数低的数据块对应二进制低位，即：

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

28.3.5 数据存储

数据存储即将数据存储进一个内存块的操作，该数据为 ECC 算法的输入数据。具体而言，将数据写入一个 ECC 内存块相当于将该数据 $D[n][31:0](n = 0, 1, \cdots, 7)$ 依次写入“该内存块起始地址 + $4 \times n$”：

- 写入 $D[0]$ 至“起始地址”
- 写入 $D[1]$ 至“起始地址 + 4”
- $\cdots$
- 写入 $D[7]$ 至“起始地址 + 28”

说明：
进行数据存储操作时，只需要写入数据长度的数据块即可，不需要在高位补 0。例如，在写入 192 位长度的数据时，只需要存储 192 位的数据，不需要在数据的高位补 0。

28.3.6 数据读取

数据读取即将一个数据从一个内存块读出的操作，该数据为 ECC 算法的输出数据。具体来说，从一个 ECC 内存块读数据相当于从“该内存块起始地址 + $4 \times n$”依次读出 $D[n][31:0](n = 0, 1, \cdots, 7)$：

- 从“起始地址”读出 $D[0]$
- 从“起始地址 + 4”读出 $D[1]$
- $\cdots$
- 从“起始地址 + 28”读出 $D[7]$

说明：
进行数据读取操作时，只需要读取数据长度的数据块即可。例如，在读取 192 位长度的数据时，只需要读取低 192 位（即 6 个数据块）的数据。

28.3.7 标准运算与 Jacobian 运算

ESP32-P4 ECC 硬件加速器中，所有标准运算（包括标准点验证、标准点加和标准点乘）的输入数据以及输出数据中的点均在仿射坐标系中；相对应地，所有 Jacobian 运算（包括 Jacobian 点验证、Jacobian 点加和 Jacobian 点乘）的输入数据以及输出数据中的点均在 Jacobian 坐标系中。

28.4 功能描述

28.4.1 曲线模式

ESP32-P4 ECC 硬件加速器共支持两种曲线模式，每种曲线模式唯一对应一条椭圆曲线。用户通过配置 ECC_MULT_CURVE_MODE 可选定密钥长度模式，其与椭圆曲线的对应关系如表 28.4-1 所示。

表 28.4-1. ECC 加速器曲线模式控制

ECC_MULT_CURVE_MODE	对应椭圆曲线*
0	FIPS P-192
1	FIPS P-256
2	FIPS P-384

* FIPS P-192/P-256/P-384 的曲线定义，请参考 FIPS 186-5。

28.4.2 工作模式

ESP32-P4 ECC 硬件加速器共有 11 种工作模式，每种工作模式基于选定曲线进行不同操作。用户通过配置 ECC_MULT_WORK_MODE 可选定工作模式，其值与工作模式的对应关系如表 28.4-2 所示。

表 28.4-2. ECC 硬件加速器工作模式控制

ECC_MULT_WORK_MODE	对应模式
0	标准点乘
1	**保留项，不可用**
2	标准点验证
3	标准点验证 + 标准点乘
4	Jacobian 点乘
5	点加
6	Jacobian 点验证
7	标准点验证 + Jacobian 点乘
8	模加
9	模减
10	模乘
11	模除

说明：
请注意，Jacobian 点乘模式的运算速度比标准点乘模式约快 10%。

每种工作模式的具体计算和输入/输出数据，请参照下述子章节。

28.4.2.1 标准点乘模式

标准点乘模式计算公式如下：

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中，

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px、ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。

- 输出数据：Q_x 和 Q_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。

28.4.2.2 标准点验证模式

标准点验证模式用于计算点 (P_x, P_y) 是否在选定的椭圆曲线上。

- 输入数据：P_x 和 P_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
- 输出数据：点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。

28.4.2.3 标准点验证 + 标准点乘模式

标准点验证 + 标准点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上，如果其在选定的椭圆曲线上，则继续计算如下公式：

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中，

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px、ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。
- 输出数据：
  - 点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。
  - Q_x 和 Q_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - J_x、J_y 和 J_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

28.4.2.4 Jacobian 点乘模式

Jacobian 点乘模式计算公式如下：

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px、ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。
- 输出数据：Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

28.4.2.5 点加模式

点加模式计算公式如下：

$$R = (R_x, R_y) = (J_x, J_y, J_z) = (P_x, P_y, 1) + (Q_x, Q_y, Q_z)$$

其中，

- (R_x, R_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：
  - P_x 和 P_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。
- 输出数据：
  - R_x 和 R_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - J_x、J_y 和 J_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

28.4.2.6 Jacobian 点验证模式

Jacobian 点验证模式用于计算点 (Q_x, Q_y, Q_z) 是否在选定的椭圆曲线上。

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。
- 输出数据：点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。

28.4.2.7 标准点验证 + Jacobian 点乘模式

标准点验证 + Jacobian 点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上，如果其在选定的椭圆曲线上，则继续计算如下公式：

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px, ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。
- 输出数据：
  - 点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。
  - Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

28.4.2.8 模加模式

模加模式计算公式如下：

$$R = A + B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
    * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Px。

28.4.2.9 模减模式

模减模式计算公式如下：

$$R = A - B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
    * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Px。

28.4.2.10 模乘模式

模乘模式计算公式如下：

$$R = A \cdot B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
    * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Py。

28.4.2.11 模除模式

模除模式计算公式如下：

$$R = A \cdot B^{-1} \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：

  * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
  * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。

- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Py。

28.4.3 提升抗攻击性能

在执行不同类型的点乘计算（即模式 0、3、4 和 7）时，ESP32-P4 的 ECC 加速器提供了一个选项，可保证所有点乘运算都维持相同的最长运行时间，从而进一步提升其抗攻击性能。

要启用强抗攻击性能，请将 ECC_MULT_SECURITY_MODE 置 1。启用此设置后，ECC 加速器在每次进行点乘计算时：

- 对于每次操作，执行延迟保持恒定。
- 功耗波动最小化。

28.5 时钟与复位

ESP32-P4 ECC 硬件加速器模块仅有一个模块时钟 CRYPTO_ECC_CLK 和一个模块复位 CRYPTO_ECC_RST。

ECC 时钟与复位由电源/时钟/复位寄存器 (PCR) 处理，详见章节 11 复位和时钟。在使用 ECC 硬件加速器之前，需要配置 HP_SYS_CLKRST_CRYPTO_ECC_CLK_EN 来启用 ECC 时钟，并清除 HP_SYS_CLKRST_RST_EN_ECC 来释放 ECC 复位。此外，由于密码学加速器之间的资源复用，还需额外清除 HP_SYS_CLKRST_RST_EN_ECDSA。

28.6 中断

ESP32-P4 ECC 硬件加速器可产生一个中断信号 ECC_INTR，并将其发送给 中断矩阵。

中断信号 ECC_INTR 由 ECC 硬件加速器的内部中断源 ECC_MULT_CALC_DONE_INT 生成。ECC 硬件加速器运算完成即触发该中断源。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。中断源 ECC_MULT_CALC_DONE_INT 由以下寄存器位控制（更多信息请参考 28.8 寄存器列表）：

- ECC_MULT_CALC_DONE_INT_RAW：ECC 硬件加速器运算完成时置 1。
- ECC_MULT_CALC_DONE_INT_ST：表示 ECC 硬件加速器运算完成中断的状态，用 ECC_MULT_CALC_DONE_INT_ENA 使能或屏蔽 ECC_MULT_CALC_DONE_INT_RAW 来生成该位。
- ECC_MULT_CALC_DONE_INT_ENA：用于使能或屏蔽 ECC 硬件加速器运算完成中断状态位。
- ECC_MULT_CALC_DONE_INT_CLR：置 1 此位清除 ECC 硬件加速器运算完成中断，对应的 ECC_MULT_CALC_DONE_INT_RAW 和 ECC_MULT_CALC_DONE_INT_ST 位会清零。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

28.7 配置流程

软件配置 ECC 硬件加速器的流程如下：

1. 配置 ECC 硬件加速器的时钟与复位。配置细节请参考 28.5 小节。
2. 根据 28.4 小节的描述，按照需求配置 ECC 加速器密钥长度模式和工作模式。
3. 根据 28.6 小节的描述，使能 ECC_MULT_CALC_DONE_INT 中断。
4. 置位 ECC_MULT_START 以启动 ECC 运算。
5. 等待 ECC_MULT_CALC_DONE_INT 中断的产生，即 ECC 运算结束。
6. 根据 28.4 小节的描述，查看运算结果。

28.8 寄存器列表

本小节的所有地址均为相对于 ECC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

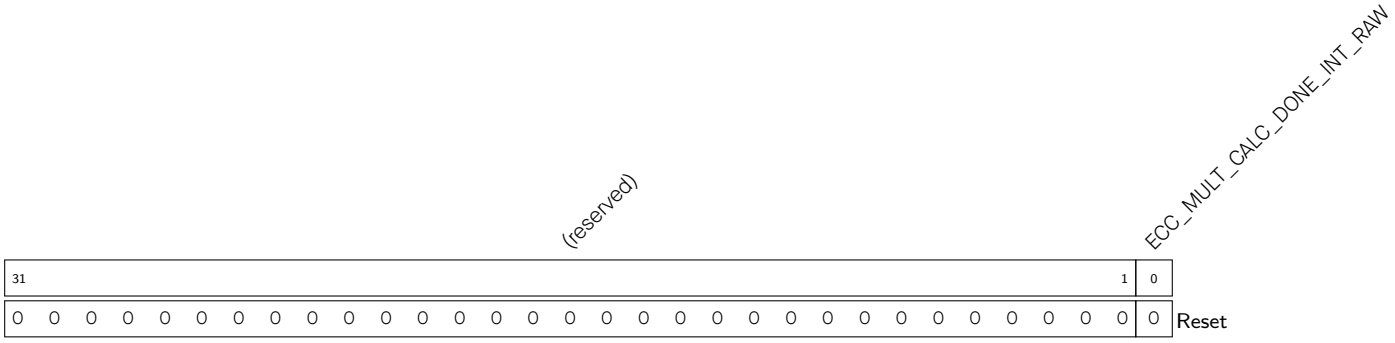
名称	描述	地址	访问权限
中断寄存器			
ECC_MULT_INT_RAW_REG	原始中断状态寄存器	0x000C	R/SS/WTC
ECC_MULT_INT_ST_REG	中断屏蔽状态寄存器	0x0010	RO
ECC_MULT_INT_ENA_REG	中断使能寄存器	0x0014	R/W
ECC_MULT_INT_CLR_REG	中断清除寄存器	0x0018	WT
配置寄存器			
ECC_MULT_CONF_REG	ECC 加速器配置寄存器	0x001C	varies
版本寄存器			
ECC_MULT_DATE_REG	版本控制寄存器	0x00FC	R/W

28.9 寄存器

本小节的所有地址均为相对于 ECC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

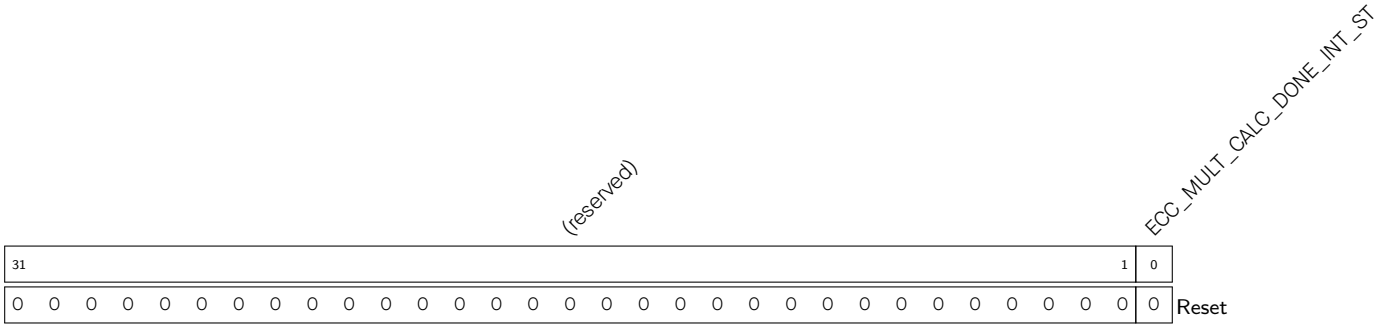
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 28.1. ECC_MULT_INT_RAW_REG (0x000C)



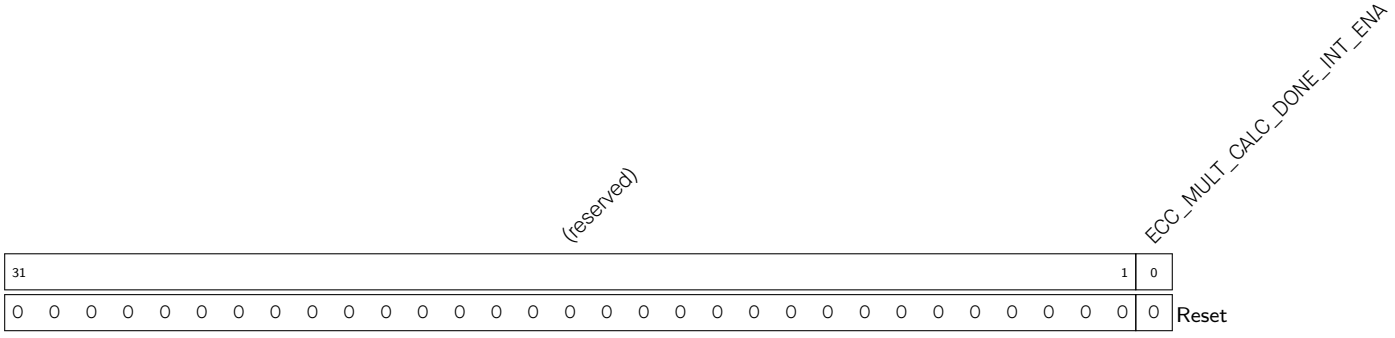
ECC_MULT_CALC_DONE_INT_RAW ECC_MULT_CALC_DONE_INT 的原始中断状态。(R/SS/WTC)

Register 28.2. ECC_MULT_INT_ST_REG (0x0010)



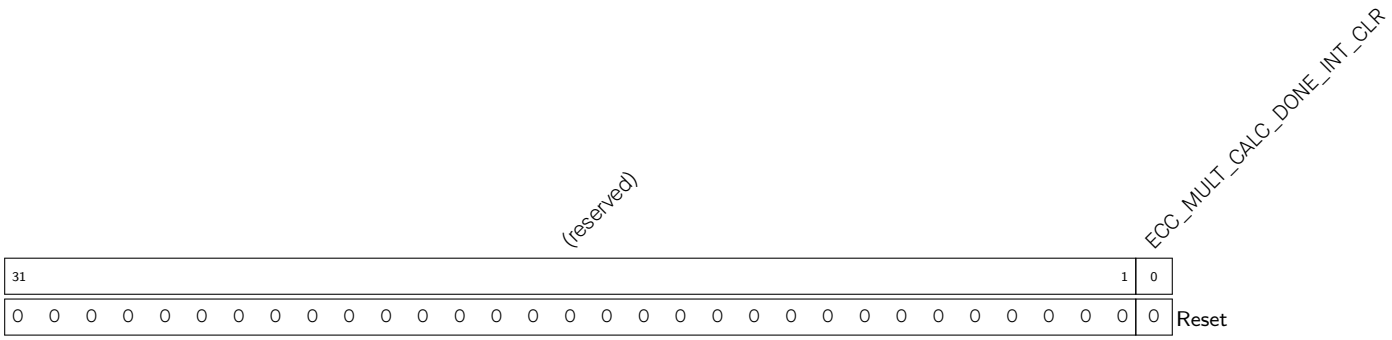
ECC_MULT_CALC_DONE_INT_ST ECC_MULT_CALC_DONE_INT 的屏蔽中断状态。(RO)

Register 28.3. ECC_MULT_INT_ENA_REG (0x0014)



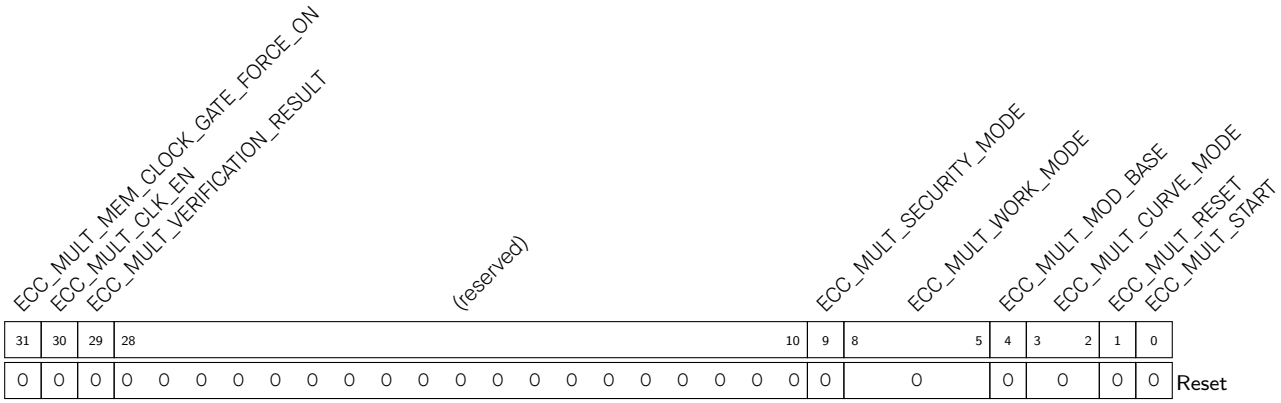
ECC_MULT_CALC_DONE_INT_ENA 写 1 使能 ECC_MULT_CALC_DONE_INT 中断。(R/W)

Register 28.4. ECC_MULT_INT_CLR_REG (0x0018)



ECC_MULT_CALC_DONE_INT_CLR 写 1 清除 ECC_MULT_CALC_DONE_INT 中断。(WT)

Register 28.5. ECC_MULT_CONF_REG (0x001C)



ECC_MULT_START 配置是否启动 ECC 加速器。此位运算结束后自动清 0。

0：无效

1：启动 ECC 加速器

(R/W/SC)

ECC_MULT_RESET 配置是否复位 ECC 加速器。

0：无效

1：复位

(WT)

ECC_MULT_CURVE_MODE 配置 ECC 加速器的曲线模式。

0：P-192

1：P-256

2：P-384

3：无效

(R/W)

见下页...

Register 28.5. ECC_MULT_CONF_REG (0x001C)

接上页...

ECC_MULT_MOD_BASE 配置在模运算操作中选择使用曲线模数或基点阶数。仅在工作模式 8-11 中有效。

0: n（基点阶数）

1: p（曲线模数）

(R/W)

ECC_MULT_WORK_MODE 配置 ECC 加速器的工作模式。

0：标准点乘

1：保留项，不可用

2：标准点验证

3：标准点验证 + 标准点乘

4：Jacobian 点乘

5：点加

6：Jacobian 点验证

7：标准点验证 + Jacobian 点乘

8：模加

9：模减

10：模乘

11：模除

(R/W)

ECC_MULT_SECURITY_MODE 配置是否针对点乘计算提升抗攻击性能。仅在工作模式 0、3、4 和 7 中有效。

0：禁用

1：启用

(R/W)

ECC_MULT_VERIFICATION_RESULT 表示 ECC 加速器的验证结果，仅在运算完成时有效。

0：验证失败

1：验证成功

(R/SS)

ECC_MULT_CLK_EN 配置是否强制打开寄存器门控。

0：无效

1：强制打开

(R/W)

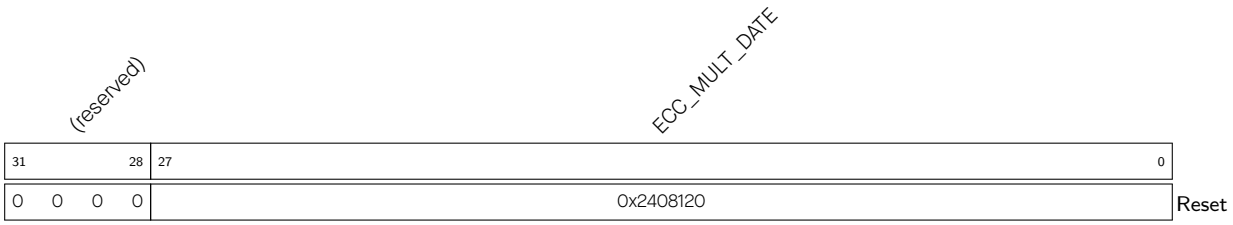
ECC_MULT_MEM_CLOCK_GATE_FORCE_ON 配置是否强制打开 ECC 内存时钟门控。

0：无效

1：强制打开

(R/W)

Register 28.6. ECC_MULT_DATE_REG (0x00FC)



ECC_MULT_DATE 版本控制寄存器。(R/W)

第 29 章

HMAC 加速器 (HMAC)

HMAC 模块通过 hash 算法 SHA-256 和密钥计算得到数据信息的信息认证码 (MAC)，采用 RFC 2104 标准。长度为 256 位的 HMAC 密钥存储在 eFuse 的密钥块中，可配置为读保护，即用户无法直接从 HMAC 加速器外部访问密钥。

29.1 主要特性

- 使用标准 HMAC-SHA-256 算法
- 基于 eFuse 密钥的 HMAC-SHA-256 算法计算，
  - 为提高安全性，软件在下行模式下无法访问其计算结果
  - 软件在上行模式下可访问其计算结果
- 下行模式下，支持生成 RSA 数字签名外设 (RSA_DS) 所需的密钥
- 下行模式下，支持重启软禁用的 JTAG

29.2 功能描述

HMAC 模块有两种工作模式，即上行模式和下行模式。上行模式中，由用户提供 HMAC 信息并回读计算结果；下行模式中，HMAC 模块为其他两个可能的内部硬件模块提供输入：一方面，如 JTAG 被暂时禁用，HMAC 可使能 JTAG；另一方面，HMAC 可用作存储在 RSA_DS 外设 flash 中数字签名参数的解密密钥。此外，这些计算在下行模式下自动进行，只要配置正确，就可以确保密钥及派生密钥材料的保密性。

说明：
芯片释放复位信号后，HMAC 模块将检查 eFuse 中是否烧写有 RSA_DS 外设所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行 RSA_DS 模式，完成相应密钥计算。

29.2.1 上行模式

在上行模式下计算 HMAC 值，用户应执行以下步骤：

1. 初始化 HMAC 模块，进入上行模式。
2. 将正确填充的信息写入 HMAC 中，一次写入一个块。
3. 从 HMAC 回读计算结果。

有关此过程的详细步骤，可参见章节 29.2.5。

说明：
使用上行模式的较多为支持 HMAC-SHA-256 算法的挑战-应答协议。假设挑战-应答协议中的通讯双方分别称为 A 和 B，想要交换的数据信息为 M，则协议的一般验证流程为：

- A 计算出一个特殊的随机数 M。
- A 将 M 发送给 B。
- B（通过 M 和密钥）计算 HMAC 结果，并将结果发送给 A。
- A（通过 M 和密钥）内部计算 HMAC 结果。
- A 比较两次计算结果。如结果相同，则验证通过 B 的身份。

29.2.2 下行模式 - JTAG 启动功能

JTAG 可被 eFuse 暂时关闭，关闭后，用户可使用 HMAC 模块重启 JTAG 功能。（详细步骤可参见章节 9.4 *eFuse 控制器 (EFUSE)*。）该模式下，用户应提供某 eFuse 密钥的 HMAC 计算结果，HMAC 模块将检查用户提供的数值与计算的数值是否匹配。如果二者相同，JTAG 将被启动，直到用户再次调用 HMAC 模块清除计算结果并关闭 JTAG。

用户可执行以下步骤重启 JTAG：

1. 通过初始化时钟和 HMAC 复位信号启动 HMAC 模块，并通过配置 HMAC_SET_PARA_PURPOSE_REG 位进入下行 JTAG 启动模式，等待计算完成。详细步骤可参见章节 29.2.5。
2. 将 1 写入 HMAC_SOFT_JTAG_CTRL_REG 寄存器，进入 JTAG 重启模式。
3. 将 256 位的 HMAC 值写入寄存器 HMAC_WR_JTAG_REG。该值是预先在本地使用 SHA-256 和已被写入 eFuse 的密钥对 32 字节的 0x00 进行 HMAC 计算得到的。需以大端字序分 8 次写入，每次 32 位。
4. 如果由 eFuse 中密钥计算的 HMAC 结果与用户在步骤 3 中写入的数值一致，则 JTAG 重启。否则，JTAG 保持关闭状态。
5. 在用户将 1 写入寄存器 HMAC_SET_INVALIDATE_JTAG_REG 或复位重启之前，JTAG 将保持关闭状态。如需再次重启 JTAG，请重复上述步骤。

29.2.3 下行模式 - RSA_DS 密钥派生功能

RSA 数字签名外设 (RSA_DS) 使用 AES-CBC 算法加密其参数。HMAC 模块作为密钥导出函数 (KDF) 导出解密上述参数的 AES 密钥。

在启用 RSA_DS 外设之前，用户必须先通过 HMAC 模块计算得到 RSA_DS 外设工作时所需的密钥。详细信息可参见章节 32 *RSA 数字签名外设 (RSA_DS)*。芯片上电后，HMAC 模块将检查 eFuse 中是否烧写有 RSA_DS 外设所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行 RSA_DS 模式，完成相应密钥计算。

29.2.4 HMAC eFuse 配置

每个烧写入 eFuse 密钥块的 HMAC 密钥都有其功能，指定该密钥用于实现哪一功能。HMAC 中，未配置匹配的功能数值的密钥无法执行计算。当前 HMAC 模块共支持三种功能：下行模式下的 JTAG 重启功能和 RSA_DS 密钥导出功能，以及上行模式下的 HMAC 计算功能。表 29.2-1 列出了上述各项功能，以及一种可指定密钥同时用于 JTAG 重启和 RSA_DS 密钥导出的功能。

启动 HMAC 模块进行计算之前，用户应先读取章节 9.4 *eFuse 控制器 (EFUSE)* 中 EFUSE_KEY_PURPOSE_x 的值（基于 eFuse 中共存在 6 个密钥，故 x 的值为 0 ~ 5，其中 EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_1 属于寄存器 EFUSE_RD_REPEAT_DATA1_REG，EFUSE_KEY_PURPOSE_2 ~ EFUSE_KEY_PURPOSE_5 属于寄存器 EFUSE_RD_REPEAT_DATA2_REG），检查 eFuse 中是否已烧写密钥。例如，在上行模式下，如果 EFUSE_KEY_PURPOSE_0~5 中没有 EFUSE_KEY_PURPOSE_HMAC_UP，则说明 eFuse 中没有上行模式可用的密钥。此时，可按照下述步骤将密钥烧写入 eFuse 中：

1. 准备一个 256 位 HMAC 密钥，将其烧写到空的 eFuse 密钥块 y 中。eFuse 中共有 6 个密钥块，分别为 4 ~ 9，因此 y 的值为 4 ~ 9。也就是说，若密钥为 key0，则对应的密钥块为 block4。接着，将功能对应数值烧写至 EFUSE_KEY_PURPOSE_$(y-4)$。例如在上行模式下，烧写密钥后，应将 EFUSE_KEY_PURPOSE_HMAC_UP（对应值为 8）写入 EFUSE_KEY_PURPOSE_$(y-4)$。更多烧写 eFuse 密钥的详细信息，可参见章节 9.4 *eFuse 控制器 (EFUSE)*。
2. 如有需要，配置 eFuse 密钥块读保护功能，使用户无法读取密钥值。用户应保留步骤中生成的随机密钥副本以便验证。

请注意，功能为 EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL 的密钥既可用于 JTAG 重启，也可用于 RSA_DS 密钥导出。

表 29.2-1. HMAC 功能及配置数值

功能	模式	数值	描述
JTAG 重启	下行模式	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
RSA_DS 密钥导出	下行模式	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC 计算	上行模式	8	EFUSE_KEY_PURPOSE_HMAC_UP
JTAG 重启和 RSA_DS 密钥导出	下行模式	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

选取 eFuse 的密钥块与 HMAC 功能

eFuse 控制器共有 6 个密钥块，即 KEY0 ~ 5 。用户将编号 n 写入寄存器 HMAC_SET_PARA_KEY_REG，表示选择 KEYn 作为本次 HMAC 模块运行时使用的密钥。

将所使用功能对应的数值写入寄存器 HMAC_SET_PARA_PURPOSE_REG 完成配置（可参见章节 29.2.5）。需要注意的是，eFuse 存储器中的密钥在烧写时都定义了功能用途，只有当 HMAC 的配置功能与 KEYn 定义的功能用途相匹配时，HMAC 模块才会执行配置好的计算任务。否则，HMAC 模块将返回匹配错误结果并结束当前计算任务。

比如，如果用户选择了 KEY3 作为本次计算的密钥，且烧写入 EFUSE_KEY_PURPOSE_3 中的数值为 6，即功能 EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG。参照表 29.2-1 可知，KEY3 是用于 JTAG 重启的密钥。如果此时寄存器 HMAC_SET_PARA_PURPOSE_REG 中配置的数值也是 6，HMAC 模块才会启动 JTAG 重启功能的计算。

从密钥管理器中选取密钥

用户可以在密钥管理器中部署 HMAC 的密钥，部署方式请参考章节 36 *密钥管理器*。要选择密钥管理器中的该密钥，请将密钥编号 7 写入注册表 HMAC_SET_PARA_KEY_REG。用户从密钥管理器中选取密钥时，注册表中 HMAC 操作的预定目标会自动设置为 “HMAC Calculation”，其他设置都不会生效。

29.2.5 调用 HMAC 流程（详细说明）

用户调用 ESP32-P4 中 HMAC 流程如下：

29.2.5.1 启动 HMAC 模块

1. 置位寄存器 HP_SYS_CLKRST_CRYPTO_HMAC_CLK_EN 中 HMAC 和 SHA 的外设时钟位，清除寄存器 HP_SYS_CLKRST_RST_EN_HMAC 中相应的外设复位位。关于如何配置时钟和重启信号，请参见章节 11 复位和时钟。
2. 将 1 写入寄存器 HMAC_SET_START_REG。

29.2.5.2 配置 HMAC 密钥和密钥功能

1. 将密钥功能 m 写入寄存器 HMAC_SET_PARA_PURPOSE_REG。表 29.2-1 列出了 m 对应的数值，可参见章节 29.2.4。
2. 将数值 n 写入寄存器 HMAC_SET_PARA_KEY_REG，选择 eFuse 存储器中的 KEYn 作为本次计算的密钥（n 的取值范围为 0 ~ 5，7），可参见章节 29.2.4。
3. 将 1 写入寄存器 HMAC_SET_PARA_FINISH_REG，完成配置。
4. 读取寄存器 HMAC_QUERY_ERROR_REG。如果返回值为 1，表明选取的密钥块与配置的密钥功能不匹配，本次计算任务中止。如果返回值为 0，表明选取的密钥块与配置的密钥功能匹配，可以执行计算流程。
5. 如果 HMAC_SET_PARA_PURPOSE_REG 的数值不为 8，表明 HMAC 模块当前处于下行模式，请参考章节 29.2.5.3 进行下一步配置。如果数值为 8，表明 HMAC 模块当前处于上行模式，请参考章节 29.2.5.4 进行下一步配置。

29.2.5.3 下行模式流程

1. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
2. 如需清除计算结果，并确保后续不再使用相关硬件（JTAG 或 RSA_DS），用户可将 1 写入寄存器 HMAC_SET_INVALIDATE_JTAG_REG，清除 JTAG 密钥生成的结果；也可将 1 写入寄存器 HMAC_SET_INVALIDATE_DS_REG，清除 RSA_DS_KEY 生成的结果。此后，如需重启 JTAG 或 RSA_DS 外设，需重新进行 HMAC 调用流程。

29.2.5.4 上行模式流程

1. 上行模式下传输数据：
   (a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
   (b) 用户根据章节 29.3.1 的描述对数据进行 SHA 附加填充并获取数据块 Block_n ($n >= 0$)
   (c) 根据数据块 n 的数量，选择其中一种操作：
   - 如果 Block_n 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位：
     i. 将 512 位的数据块 Block_n 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
     ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
     iii. 写 1 到寄存器 HMAC_ONE_BLOCK_REG 中，完成数据传输。

- 如果 Block_n 是最后一个填充数据块：
  i. 将 512 位的数据块 Block_0 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
  ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
  iii. 自动完成数据传输。
- 如果 Block_n 是倒数第二个填充数据块：
  i. 将 512 位的数据块 Block_n 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
  ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
  iii. 写 1 到寄存器 HMAC_SET_MESSAGE_PAD_REG 中，随后跳转到步骤 c 继续数据传输。
- 如果 Block_n 既不是最后一个也不是倒数第二个数据块：
  i. 将 512 位的数据块 Block_n 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
  ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
  iii. 写 1 到寄存器 HMAC_SET_MESSAGE_ING_REG 中，随后跳转到步骤 c 继续数据传输。

2. 读取上行模式下的结果 hash 值：
   (a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
   (b) 从寄存器 HMAC_RD_RESULT_*n*_REG (*n*: 0-7) 中读取 hash 结果数值。
   (c) 将 1 写入寄存器 HMAC_SET_RESULT_FINISH_REG，结束当次计算，并同时清除计算结果。
   (d) 上行模式下的操作完成。

说明：
RSA_DS 外设和 HMAC 模块可直接调用或在内部使用 SHA 加速器，但两者不能同时共享硬件资源。因此在 HMAC 模块运行过程中，SHA 模块无法被 CPU 和 RSA_DS 外设调用。

29.3 HMAC 算法细节

29.3.1 附加填充位

HMAC 模块中采用 SHA-256 作为加密 hash 算法。该算法中，若待输入数据的总位数不是 512 的倍数，用户须在软件中应用 SHA-256 附加填充算法。SHA-256 附加填充算法与 FIPS PUB 180-4 中章节 *Padding the Message* 所描述相同。下行模式下，用户无需输入数据或进行数据填充，HMAC 模块默认使用 32 字节 0x00 方式重启 JTAG，以及 32-byte 0xff 方式生成 RSA_DS 外设的 AES 密钥。

为方便阅读，以下将简要介绍数据填充的步骤。如图 29.3-1 所示，假设待处理数据长度为 m 个位，填充步骤如下：

1. 在待处理数据末尾附加一个 “1”。
2. 附加 k 个 “0”。其中，k 为满足 $m+1+k \equiv 448 (mod 512)$ 的最小非负数。

3. 附加一个 64 位的整数值作为二进制块。该二进制块的内容为待填充数据作为一个大端二进制整数值 m 的长度。

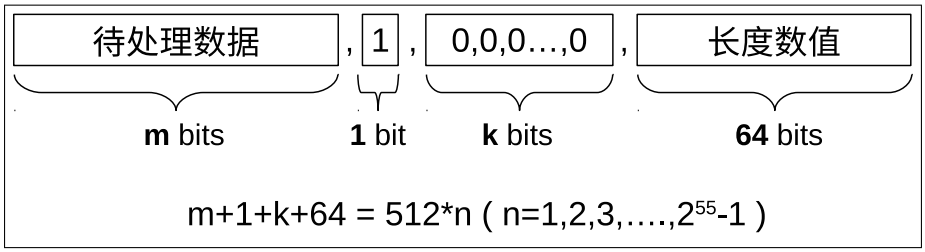


图 29.3-1. HMAC 附加填充位示意图

上行模式下，若待填充数据总位数是 512 的整数倍，则用户可通过将 1 写入 HMAC_SET_MESSGAE_END_REG 配置由硬件完成 SHA 附加填充操作，也可通过将 1 写入 HMAC_SET_MESSAGE_PAD_REG 自行完成填充操作。若待填充数据总位数不是 512 的整数倍，则用户只能自行完成 SHA 附加填充操作。数据填充操作完成后，用户应参照章节 29.2.5 完成后续配置。

29.3.2 HMAC 算法结构

HMAC 模块中应用的算法结构示意图如 29.3-2 所示。这是 RFC 2104 中描述的标准 HMAC 算法。

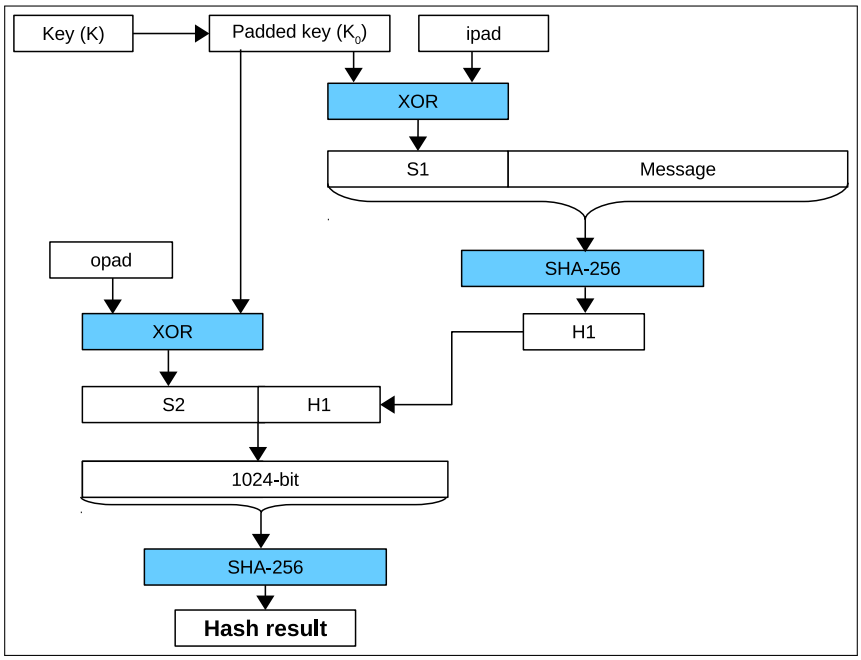


图 29.3-2. HMAC 结构示意图

图 29.3-2 中，

1. ipad 是由 64 个 0x36 字节组成的 512 位数据块。
2. opad 是由 64 个 0x5c 字节组成的 512 位数据块。

首先，HMAC 模块在 256 位的密钥 K 的位序列后附加上 256 位的 0 序列，得到 512 位的 K_0。再对 K_0 和 ipad 进行异或运算，得到 512 位的 S1。将总位数为 512 倍数的待输入数据附加到 512 位的 S1 数值后，使用 SHA-256 加密算法计算得到 256 位的 H1。

HMAC 模块通过对 K_0 和 opad 进行异或运算得到 S2，将 256 位的 hash 计算结果附加到 512 位的 S2 数值后，

得到 768 位长度的序列，使用章节 29.3.1 中描述的 SHA 附加填充算法将该序列填充成 1024 位的序列，最后使用 SHA-256 加密算法计算得到的最终 hash 结果（256 位）。

29.4 寄存器列表

本小节的所有地址均为相对于 HAMC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
控制/状态寄存器			
HMAC_SET_START_REG	HMAC 开始控制寄存器	0x0040	WO
HMAC_SET_PARA_FINISH_REG	HMAC 配置完成寄存器	0x004C	WO
HMAC_SET_MESSAGE_ONE_REG	HMAC 信息控制寄存器	0x0050	WO
HMAC_SET_MESSAGE_ING_REG	HMAC 信息继续寄存器	0x0054	WO
HMAC_SET_MESSAGE_END_REG	HMAC 信息终止寄存器	0x0058	WO
HMAC_SET_RESULT_FINISH_REG	HMAC 读取结果完成寄存器	0x005C	WO
HMAC_SET_INVALIDATE_JTAG_REG	注销 JTAG 结果寄存器	0x0060	WO
HMAC_SET_INVALIDATE_DS_REG	注销 RSA_DS 结果寄存器	0x0064	WO
HMAC_QUERY_ERROR_REG	存储用户配置的密钥和功能的配对结果	0x0068	RO
HMAC_QUERY_BUSY_REG	存储 HMAC 模块的忙碌状态	0x006C	RO
HMAC_SET_MESSAGE_PAD_REG	软件填充寄存器	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block 信息寄存器	0x00F4	WO
配置寄存器			
HMAC_SET_PARA_PURPOSE_REG	HMAC 参数配置寄存器	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC 密钥配置寄存器	0x0048	WO
HMAC_SOFT_JTAG_CTRL_REG	重启 JTAG 寄存器 0	0x00F8	WO
HMAC_WR_JTAG_REG	重启 JTAG 寄存器 1	0x00FC	WO
HMAC 信息块			
HMAC_WR_MESSAGE_0_REG	信息寄存器 0	0x0080	WO
HMAC_WR_MESSAGE_1_REG	信息寄存器 1	0x0084	WO
HMAC_WR_MESSAGE_2_REG	信息寄存器 2	0x0088	WO
HMAC_WR_MESSAGE_3_REG	信息寄存器 3	0x008C	WO
HMAC_WR_MESSAGE_4_REG	信息寄存器 4	0x0090	WO
HMAC_WR_MESSAGE_5_REG	信息寄存器 5	0x0094	WO
HMAC_WR_MESSAGE_6_REG	信息寄存器 6	0x0098	WO
HMAC_WR_MESSAGE_7_REG	信息寄存器 7	0x009C	WO
HMAC_WR_MESSAGE_8_REG	信息寄存器 8	0x00A0	WO
HMAC_WR_MESSAGE_9_REG	信息寄存器 9	0x00A4	WO
HMAC_WR_MESSAGE_10_REG	信息寄存器 10	0x00A8	WO
HMAC_WR_MESSAGE_11_REG	信息寄存器 11	0x00AC	WO
HMAC_WR_MESSAGE_12_REG	信息寄存器 12	0x00B0	WO
HMAC_WR_MESSAGE_13_REG	信息寄存器 13	0x00B4	WO
HMAC_WR_MESSAGE_14_REG	信息寄存器 14	0x00B8	WO
HMAC_WR_MESSAGE_15_REG	信息寄存器 15	0x00BC	WO
HMAC 上行结果			
HMAC_RD_RESULT_0_REG	Hash 结果寄存器 0	0x00C0	RO

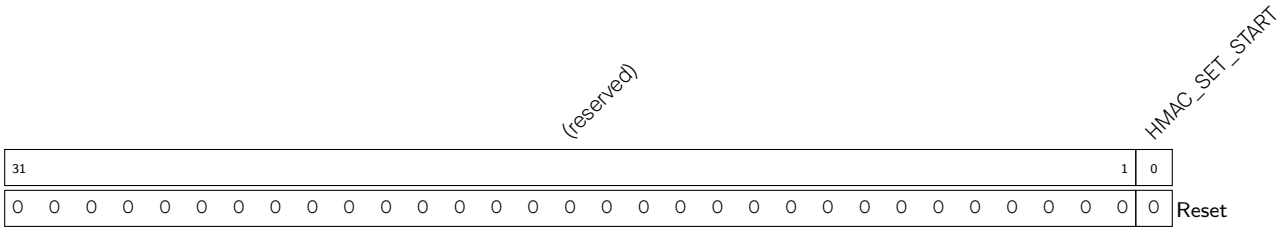
名称	描述	地址	访问
HMAC_RD_RESULT_1_REG	Hash 结果寄存器 1	0x00C4	RO
HMAC_RD_RESULT_2_REG	Hash 结果寄存器 2	0x00C8	RO
HMAC_RD_RESULT_3_REG	Hash 结果寄存器 3	0x00CC	RO
HMAC_RD_RESULT_4_REG	Hash 结果寄存器 4	0x00D0	RO
HMAC_RD_RESULT_5_REG	Hash 结果寄存器 5	0x00D4	RO
HMAC_RD_RESULT_6_REG	Hash 结果寄存器 6	0x00D8	RO
HMAC_RD_RESULT_7_REG	Hash 结果寄存器 7	0x00DC	RO
版本寄存器			
HMAC_DATE_REG	版本控制寄存器	0x01FC	R/W

29.5 寄存器

本小节的所有地址均为相对于 HAMC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 29.1. HMAC_SET_START_REG (0x0040)



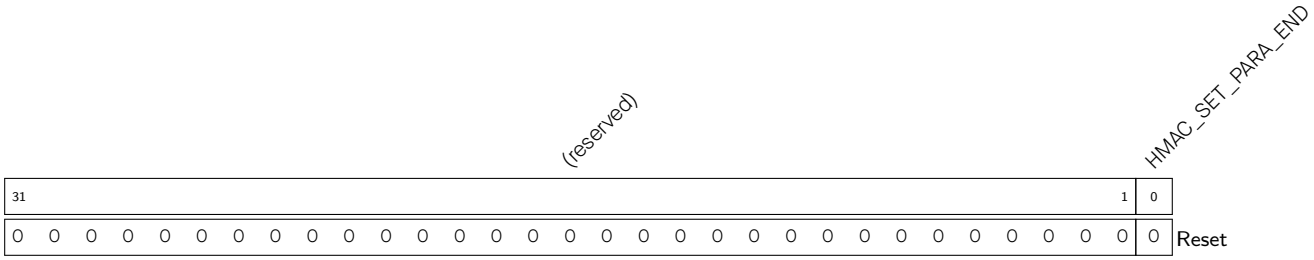
HMAC_SET_START 配置是否启动 HMAC。

0：关闭 HMAC

1：启动 HMAC

(WO)

Register 29.2. HMAC_SET_PARA_FINISH_REG (0x004C)



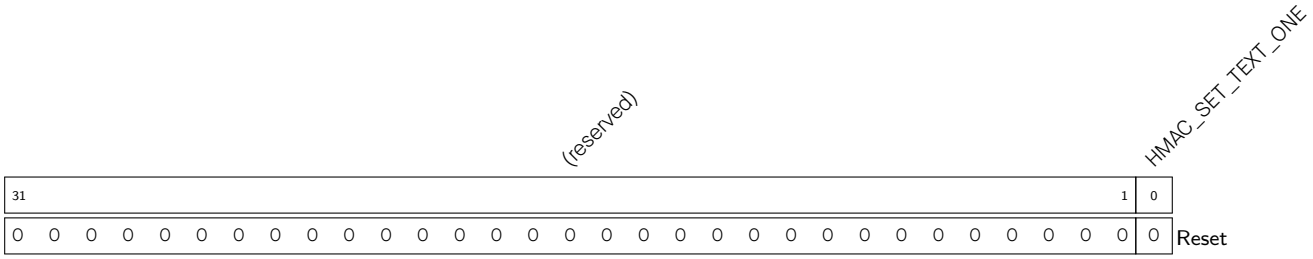
HMAC_SET_PARA_END 配置是否完成 HMAC 配置。

0：无效

1：完成配置

(WO)

Register 29.3. HMAC_SET_MESSAGE_ONE_REG (0x0050)



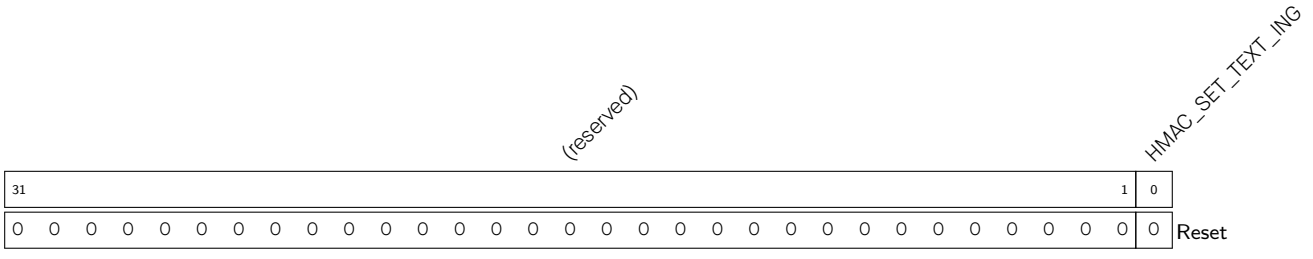
HMAC_SET_TEXT_ONE 调用 SHA 计算一个数据块。

0：无效

1：调用

(WO)

Register 29.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



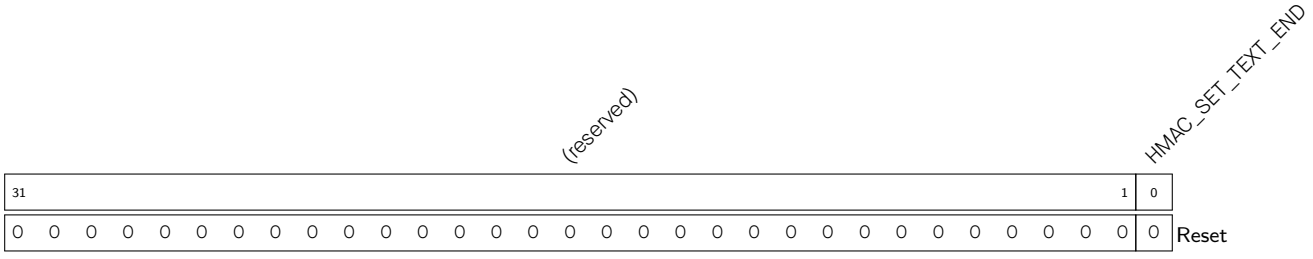
HMAC_SET_TEXT_ING 配置是否存在未处理的数据块。

0：不存在

1：存在

(WO)

Register 29.5. HMAC_SET_MESSAGE_END_REG (0x0058)



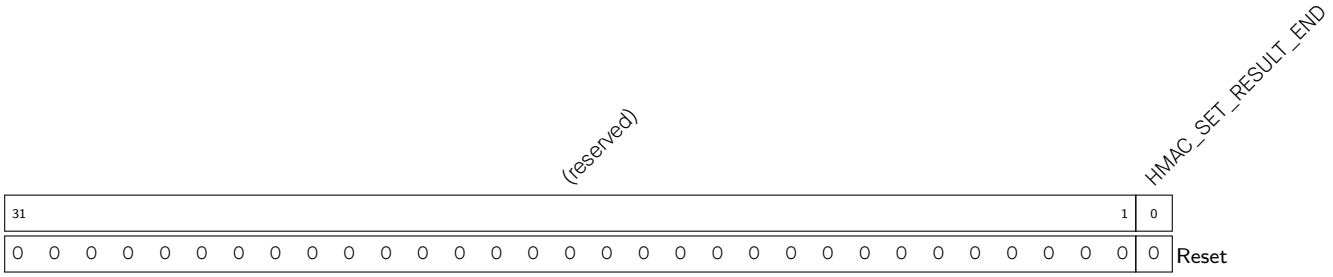
HMAC_SET_TEXT_END 配置是否开始硬件填充。

0：无效

1：开始硬件填充

(WO)

Register 29.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



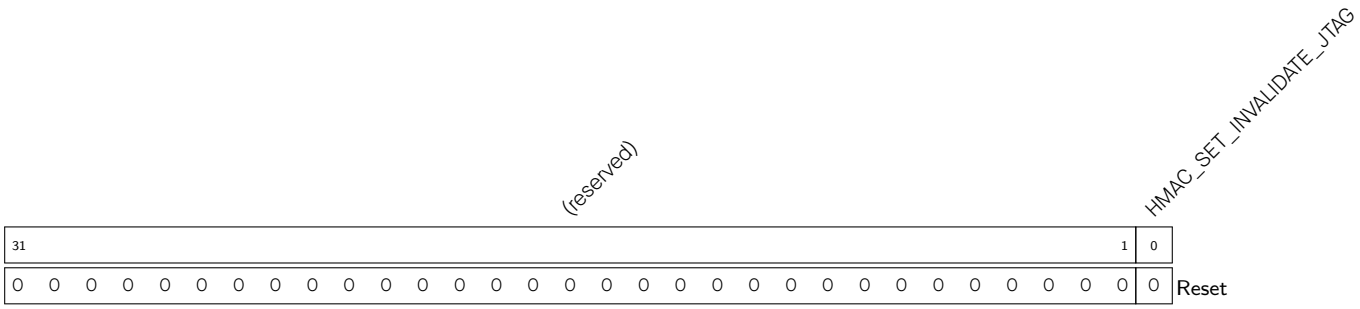
HMAC_SET_RESULT_END 配置是否退出上行模式并清空计算结果。

0：无效

1：退出上行模式并清空计算结果

(WO)

Register 29.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



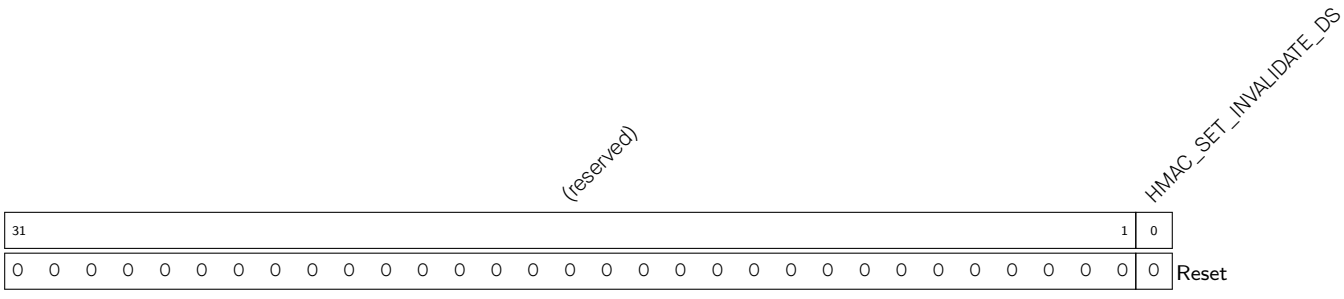
HMAC_SET_INVALIDATE_JTAG 配置是否清空下行模式下 JTAG 重启功能的计算结果。

0：无效

1：清空计算结果

(WO)

Register 29.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



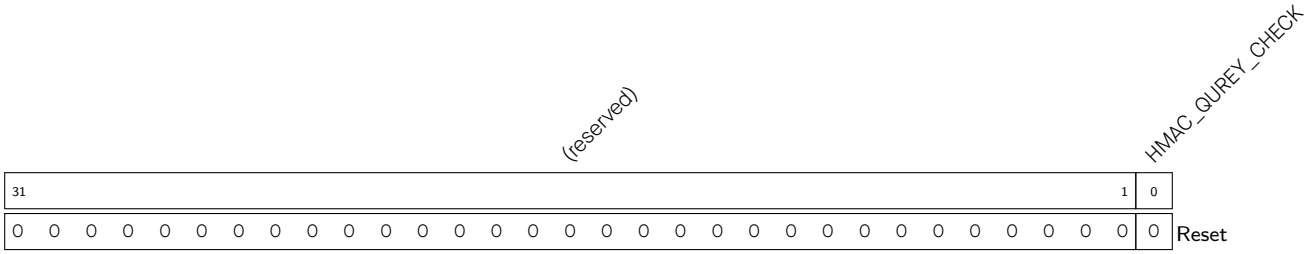
HMAC_SET_INVALIDATE_DS 配置是否清空下行模式下 RSA_DS 外设的计算结果。

0：无效

1：清空计算结果

(WO)

Register 29.9. HMAC_QUERY_ERROR_REG (0x0068)



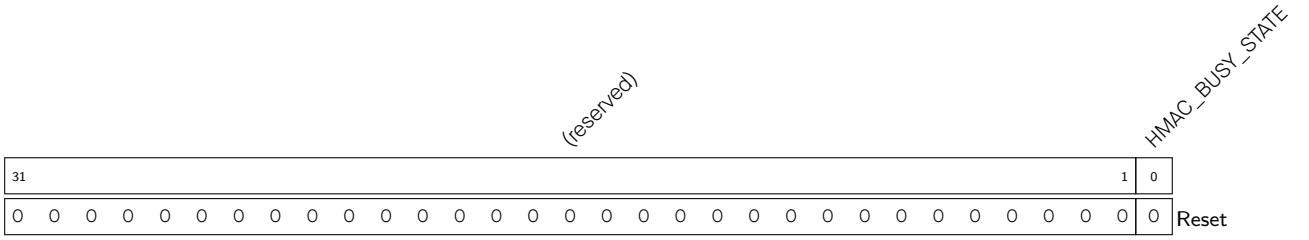
HMAC_QUREY_CHECK 表示 HMAC 密钥与功能是否匹配。

0：HMAC 密钥和功能匹配。

1：不匹配

(RO)

Register 29.10. HMAC_QUERY_BUSY_REG (0x006C)



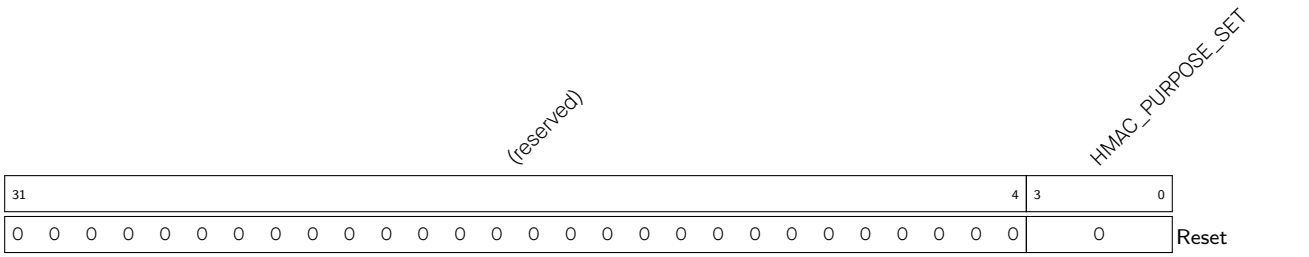
HMAC_BUSY_STATE 表示 HMAC 是否处于忙碌状态。执行计算任务之前，请确保 HMAC 已空闲。

0：空闲

1：HMAC 仍处于工作状态

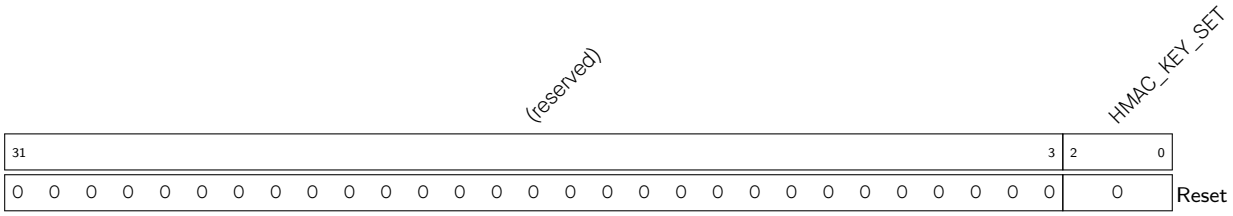
(RO)

Register 29.11. HMAC_SET_PARA_PURPOSE_REG (0x0044)



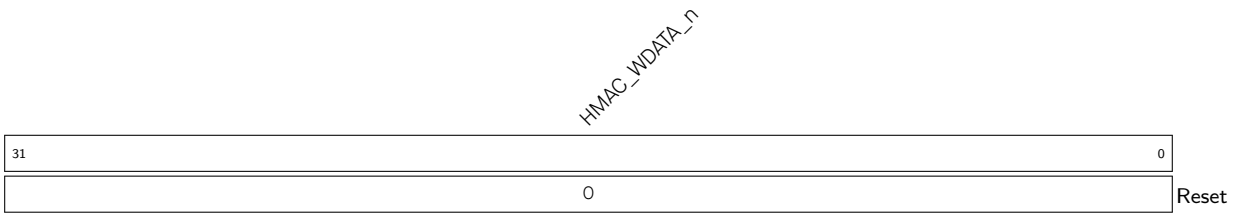
HMAC_PURPOSE_SET 设置 HMAC 功能，请参阅表 29.2-1。(WO)

Register 29.12. HMAC_SET_PARA_KEY_REG (0x0048)



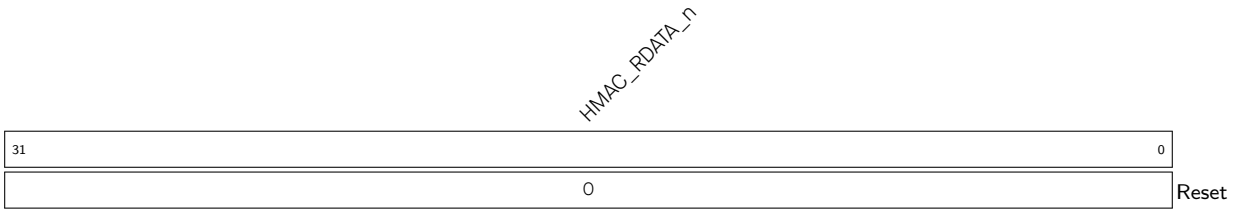
HMAC_KEY_SET 选择 HMAC 密钥。共有 7 个密钥，编号 0 至 5，和 7，将选择的密钥编号写入该字段即可。(WO)

Register 29.13. HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4**n*)



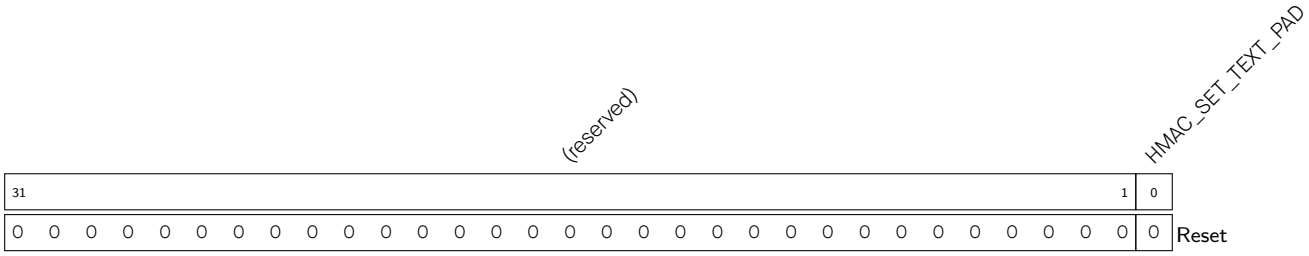
HMAC_WDATA_*n* 表示信息的第 *n* 个 32 位数据信息。(WO)

Register 29.14. HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4**n*)



HMAC_RDATA_*n* 读取 hash 结果的第 *n* 个 32 位。(RO)

Register 29.15. HMAC_SET_MESSAGE_PAD_REG (0x00F0)



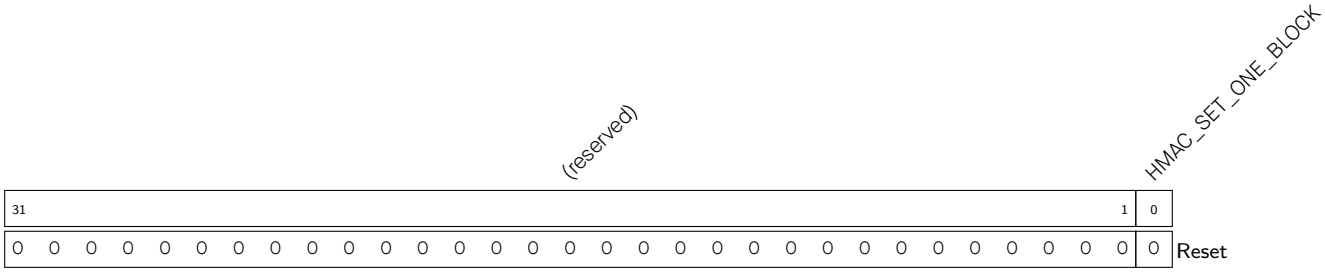
HMAC_SET_TEXT_PAD 表示是否由软件执行填充操作。

0：不由软件执行

1：由软件执行

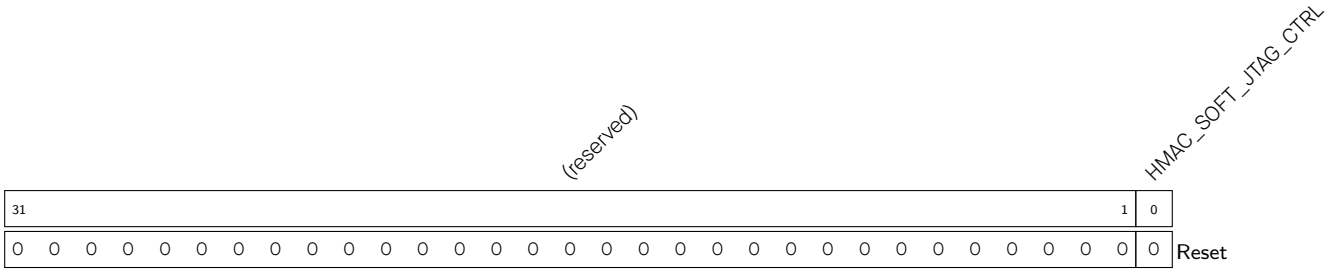
(WO)

Register 29.16. HMAC_ONE_BLOCK_REG (0x00F4)



HMAC_SET_ONE_BLOCK 置 1 表示 Block_n 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位，无需填充。(WO)

Register 29.17. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)



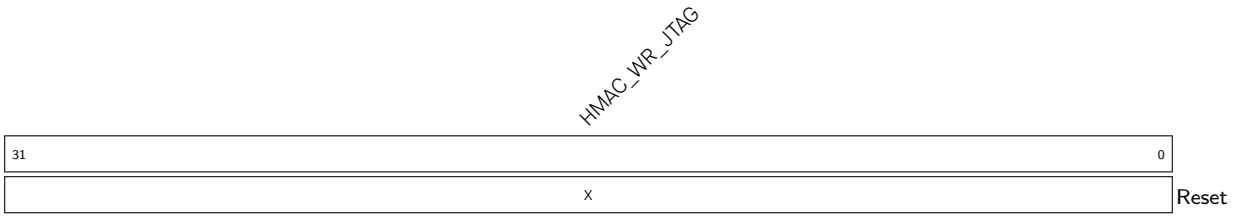
HMAC_SOFT_JTAG_CTRL 配置是否开启 JTAG 验证模式。

0：关闭

1：开启

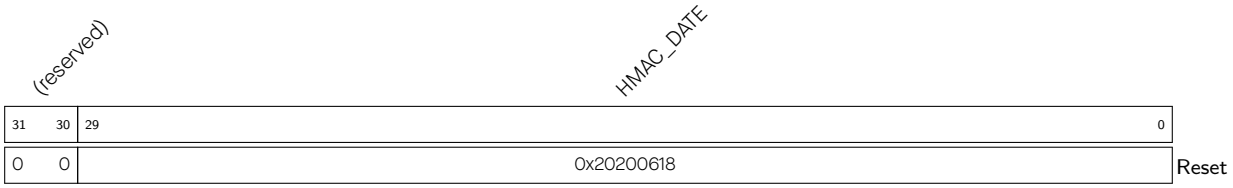
(WO)

Register 29.18. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG 写入用于重启 JTAG 的输入比较数值。(WO)

Register 29.19. HMAC_DATE_REG (0x01FC)



HMAC_DATE 版本控制寄存器。(R/W)

第 30 章

RSA 加速器 (RSA)

30.1 概述

RSA 加速器可为多种运用于“RSA 非对称式加密演算法”的高精度计算提供硬件支持，与纯软件 RSA 算法相比，能够极大地降低此类运算的运行时间和软件复杂度。RSA 加速器还支持多种“运算子长度”，具有很高的灵活性。

30.2 主要特性

RSA 加速器支持以下功能：

- 大数模幂运算（支持两个加速选项，可实现进一步加速）
- 大数模乘运算
- 大数乘法运算
- 多种运算子长度
- 支持在运算完成后触发中断

30.3 功能描述

RSA 加速器的激活需使能 HP_SYS_CLKRST_PERI_CLK_CTRL25_REG 外围时钟的 HP_SYS_CLKRST_CRYPTO_RSA_CLK_EN 位，并同时清零 HP_SYS_CLKRST_HP_RST_EN2_REG 中的 HP_SYS_CLKRST_RST_EN_RSA 位。此外，还需要清零 HP_SYS_CLKRST_RST_EN_DS 和 HP_SYS_CLKRST_RST_EN_ECDSA 位从而复位RSA 数字签名外设 (RSA_DS) 和ECDSA 数字签名外设 (ECDSA_DS)。

不过，RSA 加速器激活后还须等待 RSA 相关存储器 初始化完成后才能开始工作。具体来说，RSA_QUERY_CLEAN_REG 读 0 时初始化开始，读 1 时初始化完成。因此，在复位后首次使用 RSA 加速器时，软件需要先查询 RSA_QUERY_CLEAN_REG 的值是否为 1，以确保 RSA 加速器可正常工作。

此外，RSA 加速器支持中断功能，可对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。RSA 加速器的中断功能默认开启。

注意：

ESP32-P4 的 RSA 数字签名外设 (RSA_DS) 模块在工作期间会调用 RSA 加速器。此时，用户无法正常访问 RSA 加速器。

30.3.1 定义与表达

ESP32-P4 的 RSA 加速器支持运算子长度为 $N = 32 \times n$ 的运算。

为采用基于 b 的位置表示法，设进制数

$$b = 2^{32}$$

则运算子可以由若干个 b 进制数来表示：

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

其中 $Z_{n-1}\cdots Z_0$、$X_{n-1}\cdots X_0$、$Y_{n-1}\cdots Y_0$、$M_{n-1}\cdots M_0$、$\overline{r}_{n-1}\cdots \overline{r}_0$ 分别表示一个 b 进制数，位宽皆为 32。且 Z_{n-1}、X_{n-1}、Y_{n-1}、M_{n-1}、$\overline{r}_{n-1}$ 分别为 Z、X、Y、M、$\overline{r}$ 最高位的 b 进制数，而 Z_0、X_0、Y_0、M_0、$\overline{r}_0$ 分别为 Z、X、Y、M、$\overline{r}$ 最低位的 b 进制数。

另设 $R = b^n$，则计算得参数 $\overline{r} = R^2 \bmod M$。

另外 M' 可使用下方公式计算：

$$M' = -M^{-1} \bmod b$$

其中，M^{-1} 为 M 对 R 的模逆元，可使用扩展二进制 GCD 算法计算。

30.3.2 大数模幂运算

大数模幂运算的算法是 $Z = X^Y \bmod M$，它是基于 Montgomery Multiplication（蒙哥马利乘法）实现的。因此，对于大数模幂运算，除了需要运算子 X、Y、M 外，还需要额外两个运算子，即参数 $\overline{r}$ 和 M'。这两个参数需要通过软件提前运算得到。

RSA 加速器支持运算子长度为 $N = 32 \times n$ ($n \in \{1, 2, 3, \ldots, 128\}$) 的大数模幂运算。$Z$、$X$、$Y$、$M$ 和 $\overline{r}$ 的位宽为这 128 种中的任意一种，要求它们的位宽必须相同，而 M' 的位宽始终是 32。

大数模幂运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。
   (a) 对寄存器 RSA_MODE_REG 写入 ($\frac{N}{32} - 1$)。
   (b) 对寄存器 RSA_M_PRIME_REG 写入 M'。
   (c) 根据需要配置加速选项相关寄存器。请参照章节 30.3.5 获取详细信息。
3. 将 X_i、Y_i、M_i、$\overline{r}_i (i \in \{0, 1, \ldots, n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Y_MEM、RSA_M_MEM、RSA_Z_MEM。每块存储器的容量都是 128 字 (word)。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

   只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。

4. 对 RSA_SET_START_MODEXP 写入 1 启动计算。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生（如使能）。
6. 从存储器 RSA_Z_MEM 读出运算结果 $Z_i(i \in \{0, 1, \ldots, n-1\})$。
7. 若中断功能已开启，对 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_Y_MEM 中的 Y_i、存储器 RSA_M_MEM 中的 M_i、寄存器 RSA_M_PRIME_REG 中的 M' 都不会变化。但是，存储器 RSA_X_MEM 中的 X_i 与存储器 RSA_Z_MEM 中的 $\overline{r}_i$ 都已经被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

30.3.3 大数模乘运算

大数模乘运算 $Z = X \times Y \bmod M$ 也是基于 Montgomery Multiplication 实现的。因此，与大数模幂运算类似，也需要预先通过软件计算额外的两个运算子 $\overline{r}$ 和 M'。

RSA 加速器也支持运算子长度为 $N = 32 \times n$ $(n \in \{1, 2, 3, \ldots, 128\})$ 的大数模乘运算。

大数模乘运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。
   (a) 对寄存器 RSA_MODE_REG 写入 $(\frac{N}{32} - 1)$。
   (b) 对寄存器 RSA_M_PRIME_REG 写入 M'。
3. 将 X_i、Y_i、M_i、$\overline{r}_i$ $(i \in \{0, 1, \ldots, n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Y_MEM、RSA_M_MEM、RSA_Z_MEM。每块存储器的容量都是 128 字 (word)。

   每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

   只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 RSA_SET_START_MODMULT 写入 1。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生（如使能）。
6. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in \{0, 1, \ldots, n-1\})$。
7. 若中断功能已开启，对寄存器 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_X_MEM 中的 X_i、存储器 RSA_Y_MEM 中的 Y_i、存储器 RSA_M_MEM 中的 M_i、寄存器 RSA_M_PRIME_REG 中的 M' 都不会变化。但是，存储器 RSA_Z_MEM 中的 $\overline{r}_i$ 已经被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

30.3.4 大数乘法运算

大数乘法运算实现了 $Z = X \times Y$。其中 Z 的长度是运算子 X、Y 长度的两倍，所以 RSA 加速器只支持运算子 X、Y 长度为 $N = 32 \times n$ $(n \in \{1, 2, 3, \ldots, 64\})$ 的大数乘法运算。运算子 Z 的长度 $\hat{N}$ 为 $2 \times N$。

大数乘法运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。

2. 配置相关寄存器。对寄存器 RSA_MODE_REG 写入 $(\frac{\hat{N}}{32}-1)$，即 $(\frac{N}{16}-1)$。
3. 将 X_i、Y_i $(i \in \{0,1,\ldots,n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Z_MEM。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。n 为 $\frac{N}{32}$ 。

   X_i $(i \in \{0,1,\ldots,n-1\})$ 要填充到存储器 RSA_X_MEM 中的第 i 个字对应的地址中，但需要注意的是，Y_i $(i \in \{0,1,\ldots,n-1\})$ 并不是要填充到存储器 RSA_Z_MEM 中的第 i 个字对应的地址中，而是需要填充到存储器 RSA_Z_MEM 中的第 $n+i$ 个字对应的地址中，即存储器 RSA_Z_MEM 的基地址加上偏移量 $4\times(n+i)$。

   只需要根据运算子长度，将这两个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 RSA_SET_START_MULT 写入 1。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生（如使能）。
6. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in \{0,1,\ldots,\hat{n}-1\})$。$\hat{n}$ 为 $2\times n$ 。
7. 若中断功能已开启，对 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_X_MEM 中的 X_i 都不会变化。但是，存储器 RSA_Z_MEM 中的 Y_i 已经被覆盖。所以当需要连续运算时，只需要更新必需的寄存器与存储器即可。

30.3.5 控制加速

对于大数模幂运算，ESP32-P4 的 RSA 加速器还特别提供 SEARCH 和 CONSTANT_TIME 两个选项，可进一步提高运算速度。默认情况下，这两个选项均处于不额外加速状态，可以单独使用，也可以同时使用。值得注意的是，即使这两个选项均处于不额外加速状态，硬件 RSA 加速器也比纯软件实施的 RSA 算法快很多。

具体来说，当这两个选项均处于不额外加速状态时，求解 $Z=X^Y \bmod M$ 的时间开销完全由运算子长度决定。否则，只要有某个选项携带有额外加速效果，那么运算的时间开销还与 Y 的 0/1 分布有关。

为了更清楚地说明问题，首先假设 Y 的二进制表示为：

$$Y=(\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

其中，

- N 代表 Y 的长度，
- $\widetilde{Y}_t$ 的值为 1，
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ 的值均为 0，
- 且 $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ 中包括 m 个 0，其余 t-m 全部为 1，即 $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2},\cdots,\widetilde{Y}_0$ 的汉明重量 (Hamming weight) 为 $t-m$。

此时，当启动任一额外加速选项时：

- SEARCH 选项（RSA_SEARCH_ENABLE 置 1 开启额外加速）
  - RSA 加速器将忽略所有 $\widetilde{Y}_i$ $(i>\alpha)$ 位。其中，加速位置 α 可通过 RSA_SEARCH_POS_REG 寄存器配置。α 在配置时需小于 N-1，否则相当于没有额外加速；且不建议小于 t，否则无法正确求解

$Z = X^Y \bmod M$。当设置 α 为 t 时，加速效果最佳。此时，$\widetilde{Y}_{N-1}, \widetilde{Y}_{N-2}, ..., \widetilde{Y}_{t+1}$ 中的 0 位将在运算中全部被忽略。

- 注意，这个选项会忽略一些位，进而导致密钥长度变短，使安全性降低。因此，不应使用在安全要求高的应用中。

- CONSTANT_TIME 选项（RSA_CONSTANT_TIME_REG 置 0 开启额外加速）
  - RSA 加速器在运算过程中将简化对 Y 中 0 位的处理。因此不难想象，Y 中的 0 越多，加速效果越明显。
  - 注意，这个选项的加速效果与密钥中的 0/1 分布密切相关，这可能会被侧信道攻击 (SCA) 等攻击加以利用，使安全性降低。因此，不应使用在安全要求高的应用中。

为了直观地展示这两个选项带来的额外加速效果，下面通过一个典型实例加以说明。本实例中：

- $Z = X^Y \bmod M$
- N 等于 3072
- Y 等于 65537
- XM 取任意值
- 当 SEARCH 选项开启时，设定 α 为 16，即 RSA_SEARCH_POS_REG 配置为 16。

表 30.3-1 展示了在进行上述 $Z = X^Y \bmod M$ 运算时，开启不同加速选项组合分别对应的时间开销。

表 30.3-1. 加速效果

SEARCH 选项	CONSTANT_TIME 选项	时间开销（时钟周期）
不加速	不加速	174.7×10^6
加速	不加速	1.023×10^6
不加速	加速	0.546×10^6
加速	加速	0.540×10^6

如图 30.3-1 所示：

- 当两个选项均处于不额外加速状态时，时间开销最大。
- 当两个选项均处于额外加速状态时，时间开销最小。
- 相比于不额外加速状态，任一选项处于额外加速状态时的时间开销明显大幅度降低。

30.4 中断

ESP32-P4 的 RSA 加速器可以生成以下中断信号并发送给中断矩阵。

- RSA_INTR

这些中断信号由 RSA 加速器的内部中断源生成。表 30.4-1 列出了 RSA 加速器的内部中断源、中断触发条件、以及生成的中断信号。

表 30.4-1. RSA 加速器的内部中断源

内部中断源	触发条件	中断信号

RSA_CALC_DONE_INT	RSA 计算完成	RSA_INTR

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

30.5 存储器列表

请注意，这里的地址都是相对于 RSA 加速器基地址的地址偏移量（相对地址），详见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	大小（字节）	起始地址	结束地址	访问
RSA_M_MEM	存储器 M	512	0x0000	0x01FF	R/W
RSA_Z_MEM	存储器 Z	512	0x0200	0x03FF	R/W
RSA_Y_MEM	存储器 Y	512	0x0400	0x05FF	R/W
RSA_X_MEM	存储器 X	512	0x0600	0x07FF	R/W

30.6 寄存器列表

本小节的所有地址均为相对于 RSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

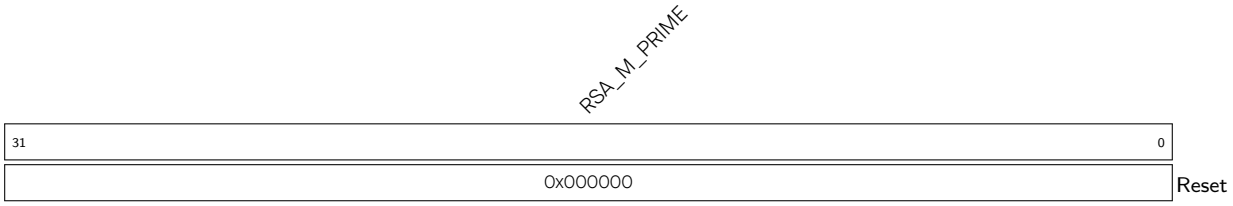
名称	描述	地址	访问
控制或配置寄存器			
RSA_M_PRIME_REG	代表 M’	0x0800	R/W
RSA_MODE_REG	配置 RSA 长度模式	0x0804	R/W
RSA_SET_START_MODEXP_REG	启动模幂运算	0x080C	WT
RSA_SET_START_MODMULT_REG	启动模乘运算	0x0810	WT
RSA_SET_START_MULT_REG	启动乘法运算	0x0814	WT
RSA_QUERY_IDLE_REG	代表 RSA 状态	0x0818	RO
RSA_CONSTANT_TIME_REG	配置 constant_time 选项	0x0820	R/W
RSA_SEARCH_ENABLE_REG	配置 search 选项	0x0824	R/W
RSA_SEARCH_POS_REG	配置 search 开始位置	0x0828	R/W
状态寄存器			
RSA_QUERY_CLEAN_REG	表示 RSA 内存初始化状态	0x0808	RO
中断寄存器			
RSA_INT_CLR_REG	清除 RSA 中断	0x081C	WT
RSA_INT_ENA_REG	使能 RSA 中断	0x082C	R/W
版本控制寄存器			
RSA_DATE_REG	版本控制寄存器	0x0830	R/W

30.7 寄存器

本小节的所有地址均为相对于 RSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

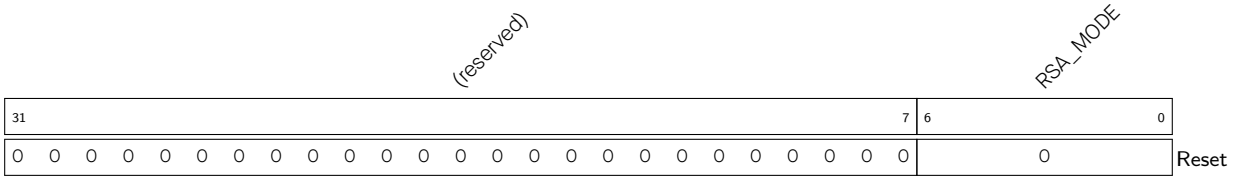
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 30.1. RSA_M_PRIME_REG (0x0800)



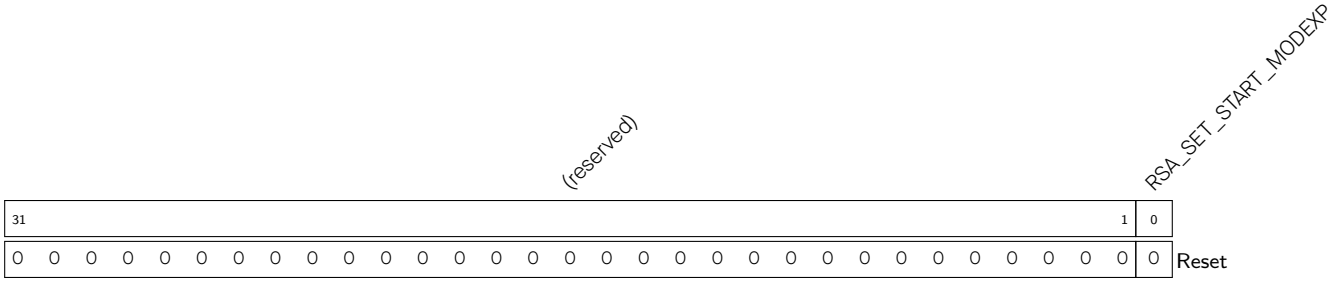
RSA_M_PRIME 代表 M'。(R/W)

Register 30.2. RSA_MODE_REG (0x0804)



RSA_MODE 配置 RSA 长度。(R/W)

Register 30.3. RSA_SET_START_MODEXP_REG (0x080C)



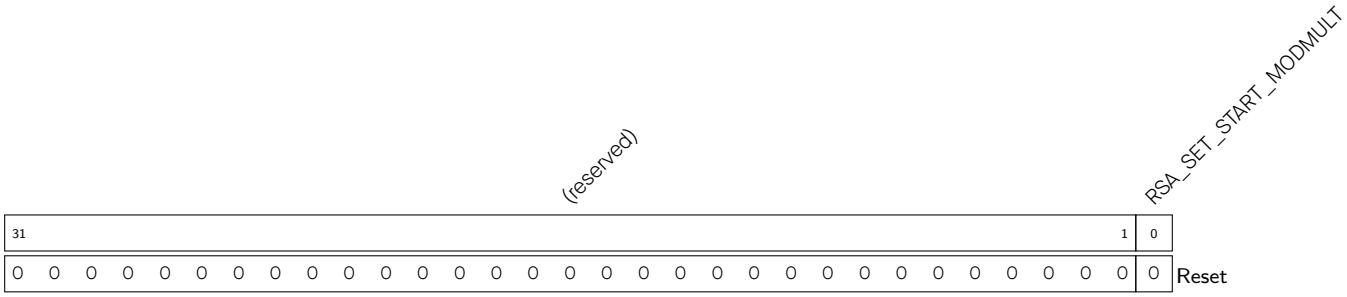
RSA_SET_START_MODEXP 配置是否启动模幂运算。

0：无效果

1：启动

(WT)

Register 30.4. RSA_SET_START_MODMULT_REG (0x0810)



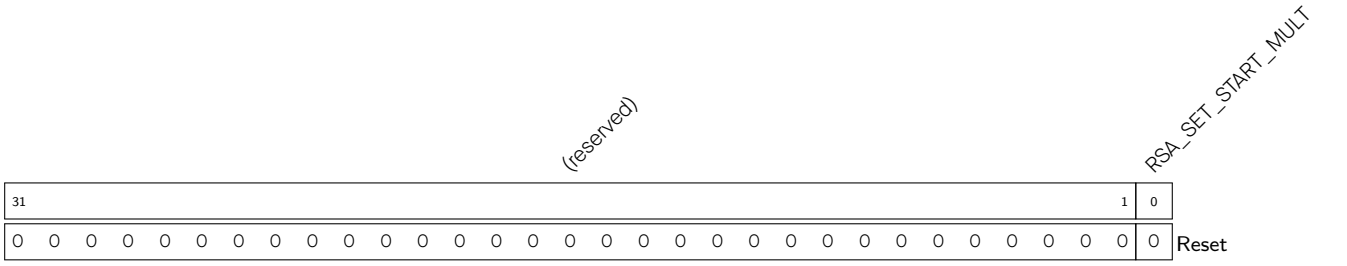
RSA_SET_START_MODMULT 配置是否启动模乘运算。

0：无效果

1：启动

(WT)

Register 30.5. RSA_SET_START_MULT_REG (0x0814)



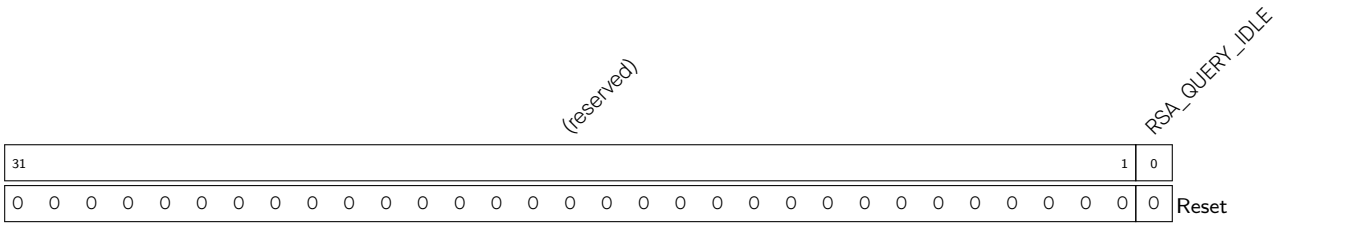
RSA_SET_START_MULT 配置是否启动乘法运算。

0：无效果

1：启动

(WT)

Register 30.6. RSA_QUERY_IDLE_REG (0x0818)



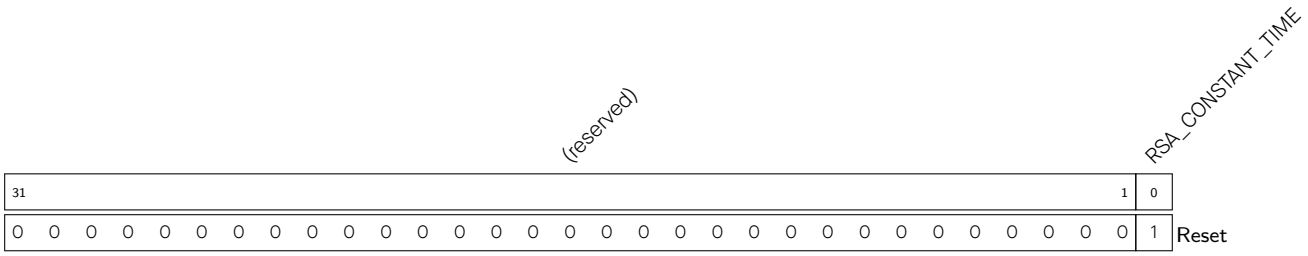
RSA_QUERY_IDLE 代表 RSA 状态。

0：忙碌

1：空闲

(RO)

Register 30.7. RSA_CONSTANT_TIME_REG (0x0820)



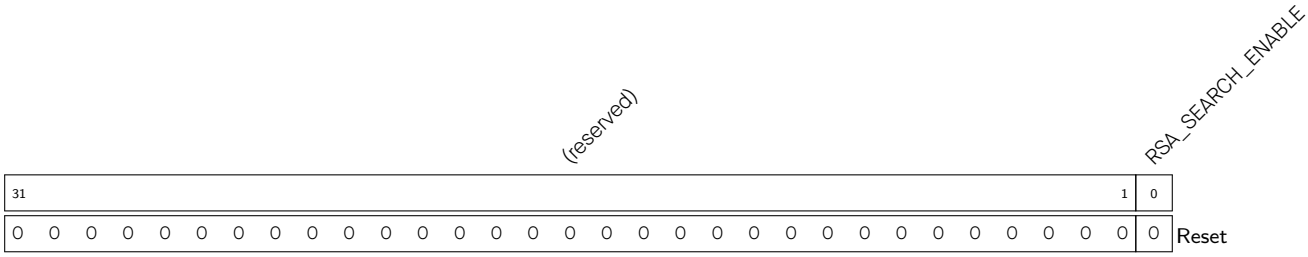
RSA_CONSTANT_TIME 配置 constant_time 选项。

0：加速

1：不加速（默认）

(R/W)

Register 30.8. RSA_SEARCH_ENABLE_REG (0x0824)



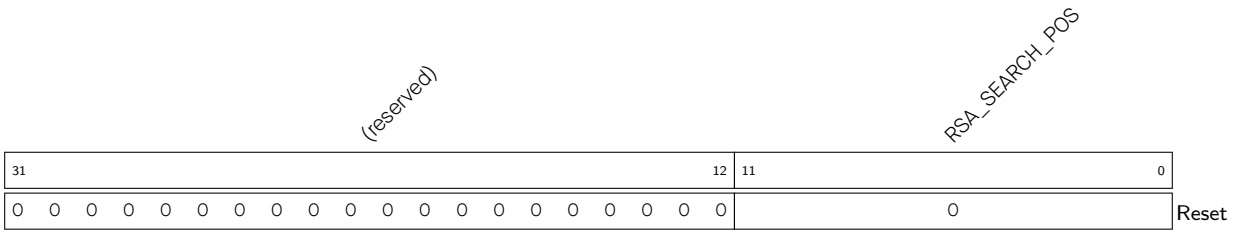
RSA_SEARCH_ENABLE 配置 search 选项。

0：不加速（默认）

1：加速

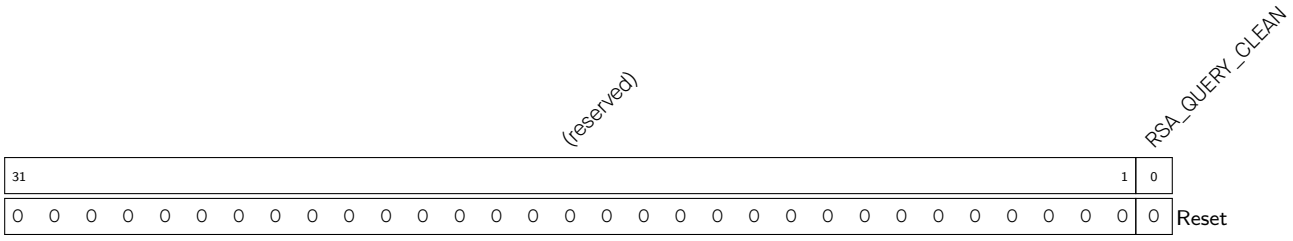
此选项需配合 RSA_SEARCH_POS_REG 使用。(R/W)

Register 30.9. RSA_SEARCH_POS_REG (0x0828)



RSA_SEARCH_POS 配置 search 的开始地址。此选项需配合 RSA_SEARCH_ENABLE_REG 使用，仅在 RSA_SEARCH_ENABLE 为 1 时有效。(R/W)

Register 30.10. RSA_QUERY_CLEAN_REG (0x0808)



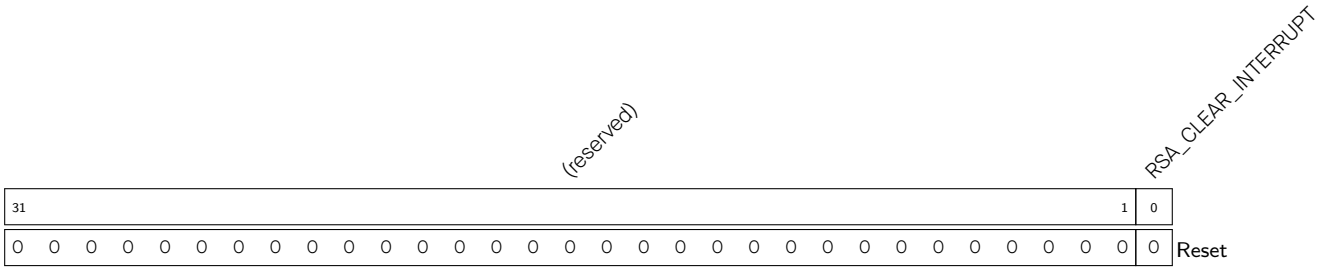
RSA_QUERY_CLEAN 代表 RSA 内存初始化的状态。

0：没有完成

1：已经完成

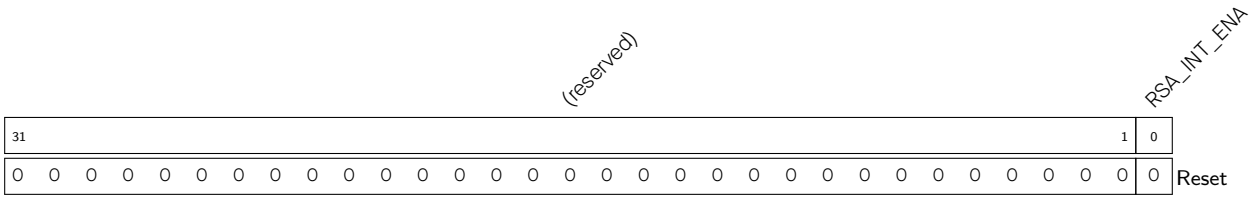
(RO)

Register 30.11. RSA_INT_CLR_REG (0x081C)



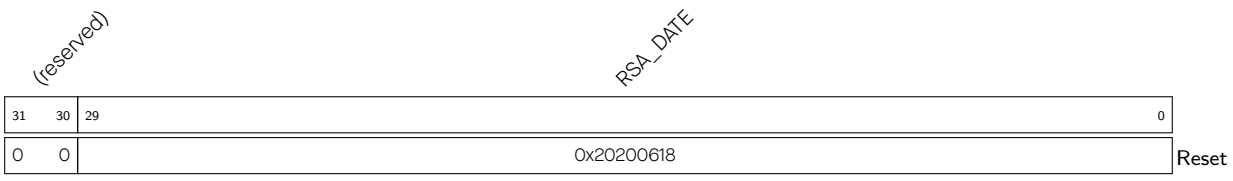
RSA_CLEAR_INTERRUPT 写 1 清除 RSA 中断。(WT)

Register 30.12. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA 写 1 使能 RSA 中断。(R/W)

Register 30.13. RSA_DATE_REG (0x0830)



RSA_DATE 版本控制寄存器。(R/W)

第 31 章

SHA 加速器 (SHA)

31.1 概述

ESP32-P4 内置的 SHA（安全哈希算法）硬件加速器可完成 SHA 运算，具有 Typical SHA 和 DMA-SHA 两种工作模式。整体而言，相比基于纯软件的 SHA 运算，SHA 硬件加速器能够极大地提高运算速度。

31.2 主要特性

ESP32-P4 的 SHA 硬件加速器：

- 支持 FIPS PUB 180-4 规范 中的以下运算标准
  - SHA-1 运算
  - SHA-224 运算
  - SHA-256 运算
  - SHA-384 运算
  - SHA-512 运算
  - SHA-512/224 运算
  - SHA-512/256 运算
  - SHA-512/*t* 运算
- 提供两种工作模式
  - Typical SHA 工作模式
  - DMA-SHA 工作模式
- 允许插入 (interleaved) 功能（仅限 Typical SHA 工作模式）
- 允许中断功能（仅限 DMA-SHA 工作模式）

31.3 工作模式简介

ESP32-P4 内置的 SHA 加速器支持两种工作模式。

- Typical SHA 工作模式：所有数据读写统一通过 CPU 访问完成。
- DMA-SHA 工作模式：所有读数据通过硬件上的 DMA 完成。具体来说，用户可配置 GDMA-AXI 控制器，由 GDMA-AXI 控制器提供 SHA 运算过程中所需的数据信息。因此，可以释放 CPU 执行其他任务。

SHA 加速器的激活仅需使能 HP_SYS_CLKRST_CRYPTO_SHA_CLK_EN 位，并同时清零 HP_SYS_CLKRST_RST_EN_SHA 位。此外，由于加密模块间存在硬件复用的关系，RSA 数字签名外设 (RSA_DS)，HMAC 加速器 (HMAC) 和ECDSA 数字签名外设 (ECDSA_DS) 模块的复位寄存器域置 1 会复位 SHA 模块，因此需要清零 HP_SYS_CLKRST_RST_EN_DS，HP_SYS_CLKRST_RST_EN_HMAC，和 HP_SYS_CLKRST_RST_EN_ECDSA 。

用户可通过配置 SHA_START_REG 或 SHA_DMA_START_REG 选择 SHA 加速器的工作模式，先配置的工作模式生效，具体请见表 31.3-1。

表 31.3-1. 工作模式选择

工作模式	选择方式
Typical SHA	SHA_START_REG 置 1
DMA-SHA	SHA_DMA_START_REG 置 1

用户可通过配置 SHA_MODE_REG 寄存器选择 SHA 加速器的运算标准，具体请见表 31.3-2。

表 31.3-2. 运算标准选择

哈希运算标准	SHA_MODE_REG 的配置
SHA-1	0
SHA-224	1
SHA-256	2
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/*t*	7

注意：

ESP32-P4 的 RSA 数字签名外设 (RSA_DS) 和 HMAC 加速器 (HMAC) 在工作时也会调用 SHA 加速器。此时，用户无法正常访问 SHA 加速器。

31.4 功能描述

SHA 加速器可以提取信息摘要 (message digest)，其主要工作流程分为两步：信息预处理 和哈希运算。

31.4.1 信息预处理

信息预处理分为三个主要步骤：附加填充比特、信息解析 和设置初始哈希值。

31.4.1.1 附加填充比特

根据选择的不同运算标准，SHA 加速器仅能处理长度为 512 位及其整倍数或 1024 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件操作将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 m 位，则填充步骤如下：

- **SHA-1、SHA-224 和 SHA-256**
  1. 首先，在待处理信息后填充 1 个 “1”；
  2. 随后，再填充 *k* 个 “0”。其中，*k* 为满足 $m+1+k \equiv 448 \ mod \ 512$ 的最小非负数解；
  3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 *m* 的值。
- **SHA-384、SHA-512、SHA-512/224、SHA-512/256 和 SHA-512/*t***
  1. 首先，在待处理信息后填充 1 个 “1”；
  2. 随后，再填充 *k* 个 “0”。其中，*k* 为满足 $m+1+k \equiv 896 \ mod \ 1024$ 的最小非负数解；
  3. 最后，在末尾填充一个 128 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 *m* 的值。

更多详情，请参考 《FIPS PUB 180-4 规范》 > 章节 “Padding the Message”。

31.4.1.2 信息解析

在完成信息填充后，还需将待处理信息（及其填充）解析为 N 个 512 位或 1024 位的信息块。SHA 加速器在工作时，所有的信息块都会写入 SHA_M_*n*_REG 中。

- 对于 **SHA-1、SHA-224 和 SHA-256**：

  待处理信息（及其填充）应解析为 N 个 512 位的信息块，即 $M^{(1)}$、$M^{(2)}$、...、$M^{(N)}$。一个 512 位信息块包括 16 个 32 位的字 (word)，则第一个 32 位字表示为 $M_0^{(i)}$，第二个 32 位字表示为 $M_1^{(i)}$，...，直到第 16 个 32 位字表示为 $M_{15}^{(i)}$。

  SHA 加速器在工作时，$M_0^{(i)}$ 会写入 SHA_M_0_REG 中，$M_1^{(i)}$ 会写入 SHA_M_1_REG，...，$M_{15}^{(i)}$ 会写入 SHA_M_15_REG 中。

- 对于 **SHA-384、SHA-512、SHA-512/224、SHA-512/256 和 SHA-512/*t***：

  待处理信息（及其填充）应解析为 N 个 1024 位的信息块，即 $M^{(1)}$、$M^{(2)}$、...、$M^{(N)}$。一个 1024 位信息块包括 16 个 64 位的字 (word)，则第 i 个信息块的第一个 64 位字表示为 $M_0^{(i)}$，第二个 64 位字表示为 $M_1^{(i)}$，...，第 16 个 64 位字表示为 $M_{15}^{(i)}$。

  SHA 加速器在工作时，$M_0^{(i)}$ 的高 32 位和低 32 位会分别写入 SHA_M_0_REG 和 SHA_M_1_REG 中，...，$M_{15}^{(i)}$ 的高 32 位和低 32 位会分别写入 SHA_M_30_REG 和 SHA_M_31_REG 中。

> **说明：**
> 有关 “信息块” 及相关概念的描述，请参考 《FIPS PUB 180-4 规范》 > 章节 “Glossary of Terms and Acronyms”。

31.4.1.3 哈希初始值 (Initial Hash Value)

在进行哈希运算前，首先必须设置哈希初始值 $H^{(0)}$。不同运算标准的哈希初始值设置要求不同，其中 SHA-1、SHA-224、SHA-256、SHA-384、SHA-512、SHA-512/224、SHA-512/256 等运算的哈希初始值为常量 C，且已经固定在硬件中，无需专门计算。

然而，SHA-512/*t* 对于不同的 t 均需要一个不同的哈希初始值。简单来说，SHA-512/*t* 是一种基于 SHA-512 的 t 位运算标准，其运算结果将截断至 t 位。其中，运算标准对 t 值的要求为 “大于 0 小于 512 且不等于 384 的正

整数”。对于不同 *t* 值的 SHA-512/*t* 运算，其哈希初始值可通过对 “SHA-512/t” 字符串的十六进制表示进行 SHA-512 运算获得。简言之，对于 *t* 取值不同的 SHA-512/*t* 运算标准，其不同之处仅在于 *t* 值不同。

因此，为了简化 SHA-512/*t* 的哈希初始值计算，可使用以下方法：

1. **计算 t_string 和 t_length**：其中，t_string 为 *t* 的字符串信息，长度为 32-bit。t_length 指明字符串长度信息，长度为 7-bit。根据 t 的取值范围不同，t_string 和 t_length 的计算过程如下：

   - 如果 $1 <= t <= 9$，则 $t_length = 7'h48$，t_string 需要按照如下格式封装：

$8'h30 + 8'ht_0$	$1'b1$	$23'b0$

     其中，$t_0 = t$。

     举例，如果 *t* = 8，则 t_0 = 8，t_string = $32'h38800000$。

   - 如果 $10 <= t <= 99$，则 $t_length = 7'h50$，t_string 需要按照如下格式封装：

$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$15'b0$

     其中，$t_0 = t\%10$，$t_1 = t/10$。

     举例，如果 *t* = 56，则 t_0 = 6，t_1 = 5，t_string = $32'h35368000$。

   - 如果 $100 <= t < 512$，则 $t_length = 7'h58$，t_string 需要按照如下格式封装：

$8'h30 + 8'ht_2$	$8'h30 + 8'ht_1$	$8'h30 + 8'ht_0$	$1'b1$	$7'b0$

     其中，$t_0 = t\%10$，$t_1 = (t/10)\%10$，$t_2 = t/100$。

     举例，如果 *t* = 231，则 t_0 = 1，t_1 = 3，t_2 = 2，t_string = $32'h32333180$。

2. **配置计算哈希初始值所需的寄存器**：用 t_string 和 t_length 初始化文本寄存器 SHA_T_STRING_REG 和 SHA_T_LENGTH_REG。

3. **计算得到哈希初始值**：对 SHA_MODE_REG 寄存器置 7 选择 SHA-512/*t* 运算，并对 SHA_START_REG 寄存器置 1，启动 SHA 加速器的运算即可。最后，轮询寄存器 SHA_BUSY_REG 结果为 0，则哈希初始值已计算完毕。

此外，也可以按照 FIPS PUB 180-4 Spec 中 “5.3.6 SHA-512/*t*” 章节的描述计算 SHA-512/*t* 的哈希初始值，也就是对 “SHA-512/*t*” 字符串的十六进制表示进行一次 “特殊” 的 SHA-512 运算，其运算得到的信息摘要即为所需的哈希初始值。这里的 “特殊” 指本次 SHA-512 运算的哈希初始值为 “SHA-512 运算标准的初始值常量 C 与 0xa5 每 8 位进行一次异或位运算后得到的结果”。

31.4.2 哈希运算流程

在完成信息预处理后，ESP32-P4 SHA 加速器将正式开始哈希运算，最终根据不同运算标准得到不同长度的信息摘要。正如上文所述，ESP32-P4 SHA 加速器支持 Typical SHA 和 DMA-SHA 两种工作模式，下面将对这两种工作模式的具体流程进行介绍。

31.4.2.1 Typical SHA 模式下的运算流程

通常情况下，ESP32-P4 的 SHA 会先处理完当前信息的所有信息块并生成该信息的信息摘要，之后再开始计算新的信息摘要。

不过，ESP32-P4 SHA 加速器还支持 “interleaved” 运算（仅 Typical SHA 支持），即在完成当前信息的所有运算前，用户在每个信息块计算完成后均可插入新的运算。

具体来说，用户可以将存储在 SHA_H_*n*_REG 寄存器中的信息摘要暂时保存到其他地方，然后让 SHA 加速器来完成其他优先级更高的运算任务。当插入的运算结束后，用户再将之前暂存的信息摘要重新写入 SHA_H_*n*_REG 中，并继续完成之前中断的计算。

Typical SHA 的具体运算流程（SHA-512/*t* 除外）

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器，设置运算标准。具体配置，请参考表 31.3-2。
2. 处理当前信息块。
   - 将当前信息块写入 SHA_M_*n*_REG 寄存器。
3. 启动 SHA 加速器 [1]。
   - 如果为首次运算，则对 SHA_START_REG 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器按照步骤 1 中选定的运算标准，使用硬件中固定的哈希初始值进行运算；
   - 如果非首次运算[2]，则对 SHA_CONTINUE_REG 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器使用 SHA_H_*n*_REG 寄存器中的值作为哈希初始值进行运算。
4. 查询当前信息块的处理进度。
   - 轮询寄存器 SHA_BUSY_REG 一直到读回的值为 0，代表 SHA 硬件加速器已完成对当前信息块的计算，进入 “空闲” 状态[3]。
5. 选择是否有后续的待处理信息块。
   - 如果存在后续待处理信息块，则跳回执行步骤 2。
   - 否则，继续执行。
6. 获取信息摘要：
   - 从寄存器堆 SHA_H_*n*_REG 取出信息摘要。

Typical SHA 的具体运算流程（SHA-512/*t*）

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器为 7 选择 SHA-512/*t* 运算标准。
2. 计算哈希初始值。
   (a) 配置 t_string 和 t_length，并将其写入 SHA_T_STRING_REG 和 SHA_T_LENGTH_REG 寄存器。具体请见 31.4.1.3 章节。
   (b) 对 SHA_START_REG 寄存器置 1，启动 SHA 加速器的运算。
   (c) 轮询寄存器 SHA_BUSY_REG 一直到读回的值为 0，代表 SHA 硬件加速器已完成哈希初始值的计算。
3. 处理当前信息块[1]。
   - 将当前信息块写入 SHA_M_*n*_REG 寄存器。
4. 启动 SHA 加速器。

- 对 SHA_CONTINUE_REG 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器使用 SHA_H_*n*_REG 寄存器中的值作为哈希初始值进行运算。

5. 查询当前信息块的处理进度。
   - 轮询寄存器 SHA_BUSY_REG 一直到读回的值为 0，代表 SHA 硬件加速器已完成对当前信息块的计算，进入“空闲”状态[3]。
6. 选择是否有后续的待处理信息块。
   - 如果存在后续待处理信息块，则跳回执行步骤 3。
   - 否则，继续执行。
7. 获取信息摘要：
   - 从寄存器堆 SHA_H_*n*_REG 读取信息摘要。

说明：

1. 这里，在 SHA 加速器进行硬件运算时，如果存在后续待处理信息块，软件还可以同时将后续信息块写入 SHA_M_*n*_REG 寄存器，以节省时间。
2. 比如重新启动 SHA 加速器完成之前暂停任务的情况。
3. 这里，用户可以选择是否需要插入其他任务。如需插入，请前往 插入任务工作流程 具体查看。

如上文所述，ESP32-P4 SHA 加速器**支持在 Typical SHA 模式下“插入”任务**。

具体工作流程如下。

1. 保存插入前任务的以下数据，准备将 SHA 加速器的使用权移交给插入的任务。
   - 读取并保存寄存器 SHA_MODE_REG 中的运算标准类型。
   - 读取并保存寄存器堆 SHA_H_*n*_REG 中的信息摘要。
2. 执行插入的任务。具体按照插入运算类型的不同，请见 Typical SHA 或 DMA-SHA 工作流程。
3. 恢复插入前任务的以下数据，准备将 SHA 加速器的使用权交还给插入前的任务。
   - 将获得使用权前保存的运算标准类型重新写入寄存器 SHA_MODE_REG;
   - 将获得使用权前保存的信息摘要写入寄存器堆 SHA_H_*n*_REG。
4. 将之前任务的下一个待处理信息块写入 SHA_M_*n*_REG 寄存器，并对 SHA_CONTINUE_REG 寄存器置 1，重新启动 SHA 加速器，完成之前暂停的任务。

31.4.2.2 DMA-SHA 模式下的运算流程

ESP32-P4 SHA 加速器在 DMA-SHA 工作模式下不支持“interleaved”运算，即用户必须在每次 DMA 运算（可能包括 1 个或多个信息块）全部结束后才能插入新的运算。这种情况下，用户如有插入运算需求，可将较大信息块进行拆分，并进行多次 DMA 运算。每次 DMA 运算之间允许插入其他运算标准的计算任务。

单次 DMA 运算最多可以处理 63 个信息块。与 Typical SHA 不同，SHA 在 DMA-SHA 工作模式下，运算过程中的数据搬运过程均由 GDMA-AXI 控制器完成。关于 GDMA-AXI 控制器的具体配置可见章节 *6 通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*。

DMA-SHA 的具体工作流程（SHA-512/*t* 除外）

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器，设置运算标准。具体配置，请参考表 31.3-2。
2. 选择是否启用中断。请将 SHA_INT_ENA_REG 寄存器配置为 1 以启动中断。
3. 配置块个数。
   - 将待加密数据的总块数 *M* 写入 SHA_DMA_BLOCK_NUM_REG 寄存器。
4. 开始 DMA-SHA 运算。
   - 如果当前 DMA-SHA 运算为接着另一次 DMA-SHA 的运算，需要提前将另一次计算得到的信息摘要写入寄存器堆 SHA_H_*n*_REG 中，随后将 1 写入寄存器 SHA_DMA_CONTINUE_REG；
   - 否则，只需要将 1 写入寄存器 SHA_DMA_START_REG。
5. 等待 DMA-SHA 运算结束。判断 DMA-SHA 运算结束有以下两种方法：
   - 轮询寄存器 SHA_BUSY_REG 结果为 0。
   - 等待中断信号产生。此时，应及时通过软件将 SHA_INT_CLEAR_REG 寄存器置为 1 以清除中断。
6. 获取信息摘要
   - 从寄存器堆 SHA_H_*n*_REG 取出信息摘要。

DMA-SHA 的具体工作流程（SHA-512/*t*）

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器为 7 选择 SHA-512/*t* 运算标准。
2. 选择是否启用中断。请将 SHA_INT_ENA_REG 寄存器配置为 1 以启动中断。
3. 计算哈希初始值。
   (a) 配置 t_string 和 t_length，并将其写入 SHA_T_STRING_REG 和 SHA_T_LENGTH_REG 寄存器。具体请见 31.4.1.3 章节。
   (b) 对 SHA_START_REG 寄存器置 1，启动 SHA 加速器的运算。
   (c) 轮询寄存器 SHA_BUSY_REG 一直到读回的值为 0，代表 SHA 硬件加速器已完成哈希初始值的计算。
4. 配置块个数。
   - 将待加密数据的总块数 M 写入 SHA_DMA_BLOCK_NUM_REG 寄存器。
5. 开始 DMA-SHA 运算。
   - 对 SHA_DMA_CONTINUE_REG 置 1 启动 SHA 加速器。
6. 等待 DMA-SHA 运算结束。判断 DMA-SHA 运算结束有以下两种方法：
   - 轮询寄存器 SHA_BUSY_REG 结果为 0。
   - 等待中断信号产生。此时，应及时通过软件将 SHA_INT_CLEAR_REG 寄存器置为 1 以清除中断。
7. 获取信息摘要
   - 从寄存器堆 SHA_H_*n*_REG 取出信息摘要。

31.4.3 信息摘要存储

哈希运算完成之后，计算得到的信息摘要被 SHA 加速器更新至对应的 SHA_H_*n*_REG (*n*: 0 ~ 15) 寄存器中。不同运算标准得到的信息摘要长度也不同，详情见表 31.4-4：

表 31.4-4. 不同运算标准信息摘要的寄存器占用情况

哈希运算标准	信息摘要长度（位）	寄存器占用情况[1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/*t*[2]	*t*	SHA_H_0_REG ~ SHA_H_*x*_REG

[1] 信息摘要从最高有效位至最低有效位存放，第一个 word 存放在寄存器 SHA_H_0_REG 中，第二个 word 存放在寄存器 SHA_H_1_REG 中，以此类推，具体可参考 31.4.1.2。

[2] SHA-512/*t* 运算标准使用的寄存器与 *t* 的取值有关。*x*+1 代表用于存储 *t* 位信息摘要的 32 位寄存器个数，因此 *x* = $roundup(t/32)$-1。举例：

- 当 *t* = 8 时，则 *x* = 0，代表最终的信息摘要长度为 8 位，存放在寄存器 SHA_H_0_REG 的高 8 位中；
- 当 *t* = 32 时，则 *x* = 0，代表最终的信息摘要长度为 32 位，存放在寄存器 SHA_H_0_REG 中；
- 当 *t* = 132 时，则 *x* = 4，代表最终的信息摘要长度为 132 位，存放在寄存器 SHA_H_0_REG、SHA_H_1_REG、SHA_H_2_REG、SHA_H_3_REG，及 SHA_H_4_REG 中。

31.5 中断

ESP32-P4 的 SHA 加速器可以生成以下中断信号并发送给中断矩阵。

- SHA_INTR

这些中断信号由 SHA 加速器的内部中断源生成。表 31.5-1 列出了 SHA 加速器的内部中断源、中断触发条件、以及生成的中断信号。

表 31.5-1. SHA 加速器的内部中断源

内部中断源	触发条件	中断信号
SHA_CALC_DONE_INT	DMA-SHA 模式下完成信息摘要的计算	SHA_INTR

说明：

- 中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。
- 与标准中断寄存器组不同，SHA 的中断寄存器组仅包含 INT_ENA(SHA_INT_ENA_REG)和 INT_CLR(SHA_INT_CLEAR_REG)

字段，并不支持用户读取 INT_RAW 和 INT_ST 字段。

31.6 寄存器列表

本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	权限
控制与配置寄存器			
SHA_MODE_REG	配置 SHA 加速器的运算标准	0x0000	R/W
SHA_T_STRING_REG	t_string，用于计算 SHA-512/t 的哈希初始值	0x0004	R/W
SHA_T_LENGTH_REG	t_length，用于计算 SHA-512/t 的哈希初始值	0x0008	R/W
SHA_CONTINUE_REG	继续 SHA 运算（仅用于 Typical SHA 模式）	0x0014	WO
SHA_DMA_START_REG	启动 SHA 加速器的 DMA-SHA 模式	0x001C	WO
SHA_START_REG	启动 SHA 加速器的 Typical SHA 模式	0x0010	WO
SHA_DMA_CONTINUE_REG	继续 SHA 运算（仅用于 DMA-SHA 模式）	0x0020	WO
SHA_DMA_BLOCK_NUM_REG	信息块个数寄存器（仅用于 DMA-SHA 模式）	0x000C	R/W
状态寄存器			
SHA_BUSY_REG	表示 SHA 加速器是否处于“忙碌”状态	0x0018	RO
中断寄存器			
SHA_INT_CLEAR_REG	DMA-SHA 中断清除寄存器	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA 中断使能寄存器	0x0028	R/W
数据寄存器			
SHA_H_0_REG	哈希值	0x0040	R/W
SHA_H_1_REG	哈希值	0x0044	R/W
SHA_H_2_REG	哈希值	0x0048	R/W
SHA_H_3_REG	哈希值	0x004C	R/W
SHA_H_4_REG	哈希值	0x0050	R/W
SHA_H_5_REG	哈希值	0x0054	R/W
SHA_H_6_REG	哈希值	0x0058	R/W
SHA_H_7_REG	哈希值	0x005C	R/W
SHA_H_8_REG	哈希值	0x0060	R/W
SHA_H_9_REG	哈希值	0x0064	R/W
SHA_H_10_REG	哈希值	0x0068	R/W
SHA_H_11_REG	哈希值	0x006C	R/W
SHA_H_12_REG	哈希值	0x0070	R/W
SHA_H_13_REG	哈希值	0x0074	R/W
SHA_H_14_REG	哈希值	0x0078	R/W
SHA_H_15_REG	哈希值	0x007C	R/W
SHA_M_0_REG	输入信息	0x0080	R/W
SHA_M_1_REG	输入信息	0x0084	R/W
SHA_M_2_REG	输入信息	0x0088	R/W
SHA_M_3_REG	输入信息	0x008C	R/W
SHA_M_4_REG	输入信息	0x0090	R/W
SHA_M_5_REG	输入信息	0x0094	R/W

名称	描述	地址	权限
SHA_M_6_REG	输入信息	0x0098	R/W
SHA_M_7_REG	输入信息	0x009C	R/W
SHA_M_8_REG	输入信息	0x00A0	R/W
SHA_M_9_REG	输入信息	0x00A4	R/W
SHA_M_10_REG	输入信息	0x00A8	R/W
SHA_M_11_REG	输入信息	0x00AC	R/W
SHA_M_12_REG	输入信息	0x00B0	R/W
SHA_M_13_REG	输入信息	0x00B4	R/W
SHA_M_14_REG	输入信息	0x00B8	R/W
SHA_M_15_REG	输入信息	0x00BC	R/W
SHA_M_16_REG	输入信息	0x00C0	R/W
SHA_M_17_REG	输入信息	0x00C4	R/W
SHA_M_18_REG	输入信息	0x00C8	R/W
SHA_M_19_REG	输入信息	0x00CC	R/W
SHA_M_20_REG	输入信息	0x00D0	R/W
SHA_M_21_REG	输入信息	0x00D4	R/W
SHA_M_22_REG	输入信息	0x00D8	R/W
SHA_M_23_REG	输入信息	0x00DC	R/W
SHA_M_24_REG	输入信息	0x00E0	R/W
SHA_M_25_REG	输入信息	0x00E4	R/W
SHA_M_26_REG	输入信息	0x00E8	R/W
SHA_M_27_REG	输入信息	0x00EC	R/W
SHA_M_28_REG	输入信息	0x00F0	R/W
SHA_M_29_REG	输入信息	0x00F4	R/W
SHA_M_30_REG	输入信息	0x00F8	R/W
SHA_M_31_REG	输入信息	0x00FC	R/W
版本寄存器			
SHA_DATE_REG	版本控制寄存器	0x002C	R/W

31.7 寄存器

本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

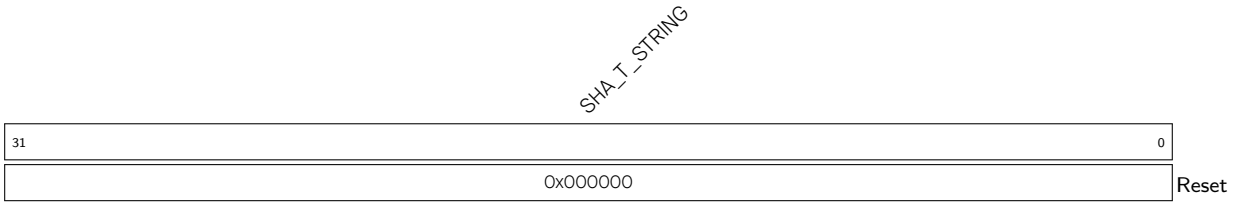
Register 31.1. SHA_MODE_REG (0x0000)



SHA_MODE 配置 SHA 加速器的运算标准。

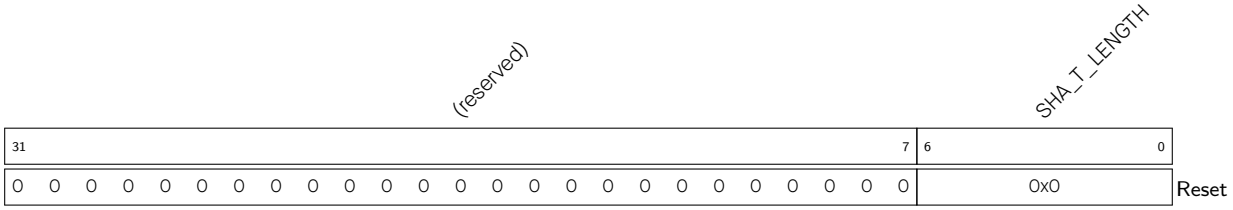
0：SHA-1
1：SHA-224
2：SHA-256
3：SHA-384
4：SHA-512
5：SHA-512/224
6：SHA-512/256
7：SHA-512/t
(R/W)

Register 31.2. SHA_T_STRING_REG (0x0004)



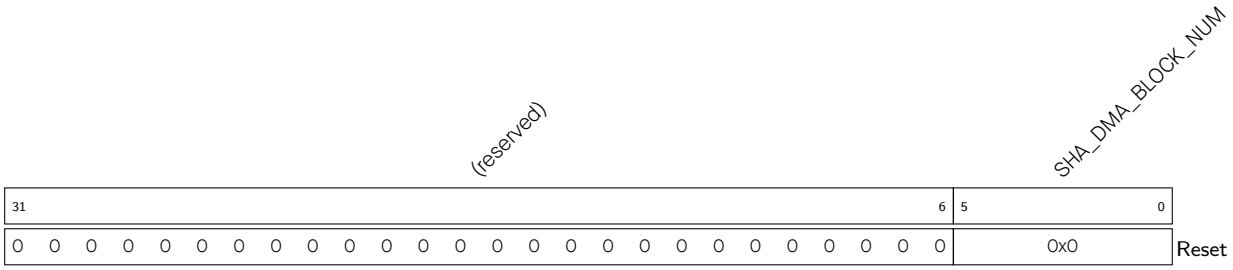
SHA_T_STRING 表示哈希字符串内容（仅用于计算 SHA-512/t 的哈希初始值）。(R/W)

Register 31.3. SHA_T_LENGTH_REG (0x0008)



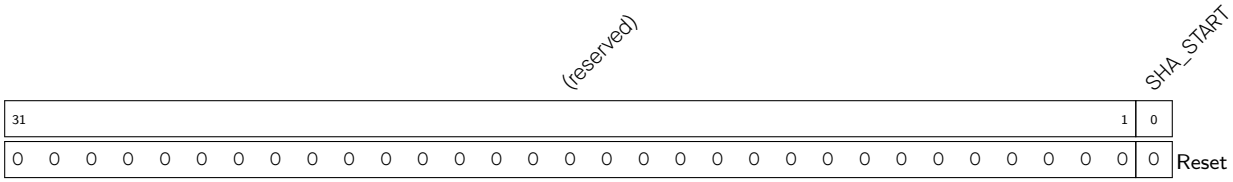
SHA_T_LENGTH 表示哈希字符串长度（仅用于计算 SHA-512/t 的哈希初始值）。(R/W)

Register 31.4. SHA_DMA_BLOCK_NUM_REG (0x000C)



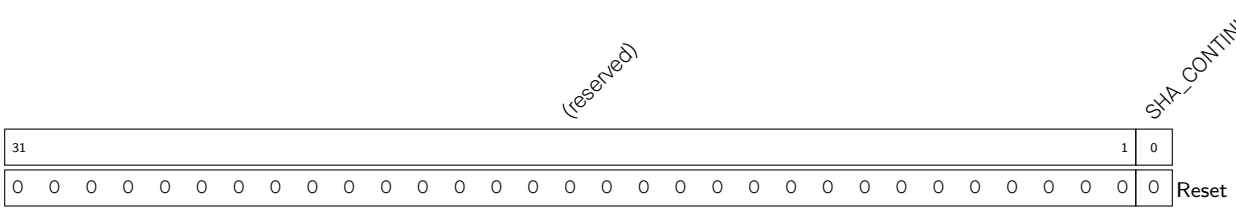
SHA_DMA_BLOCK_NUM 配置 DMA-SHA 工作模式下的信息块个数。(R/W)

Register 31.5. SHA_START_REG (0x0010)



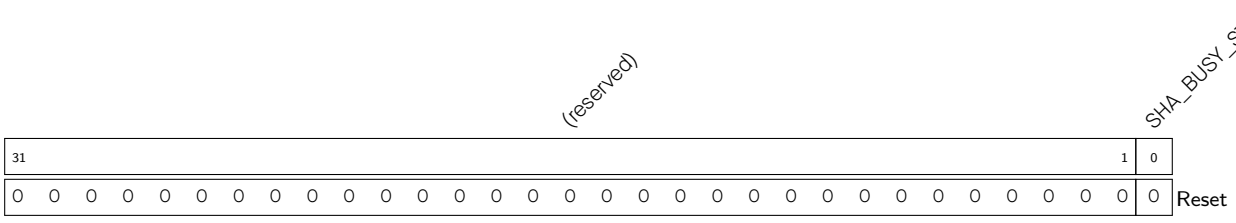
SHA_START 写 1 启动 SHA 加速器的 Typical SHA 模式。(WO)

Register 31.6. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE 写 1 继续 SHA 加速器的 Typical SHA 运算。(WO)

Register 31.7. SHA_BUSY_REG (0x0018)



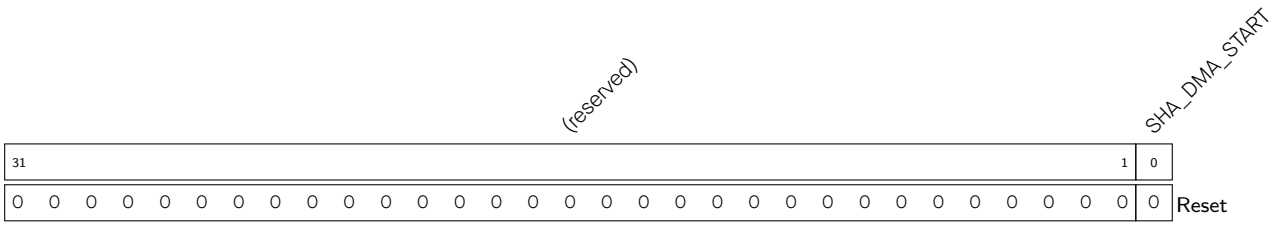
SHA_BUSY_STATE 表示 SHA 是否处于“忙碌”状态。

0：空闲

1：忙碌

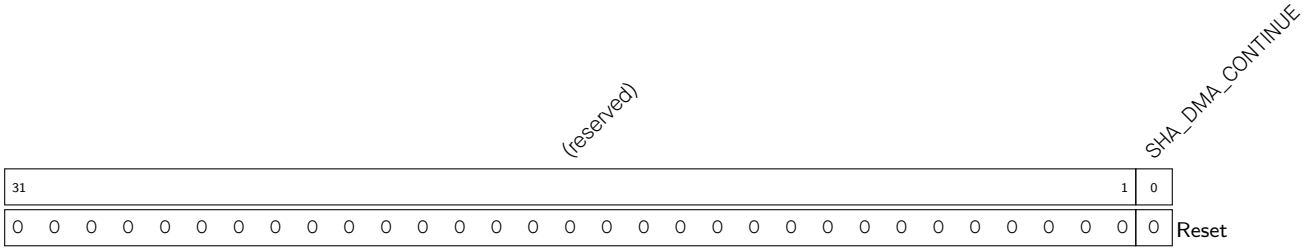
(RO)

Register 31.8. SHA_DMA_START_REG (0x001C)



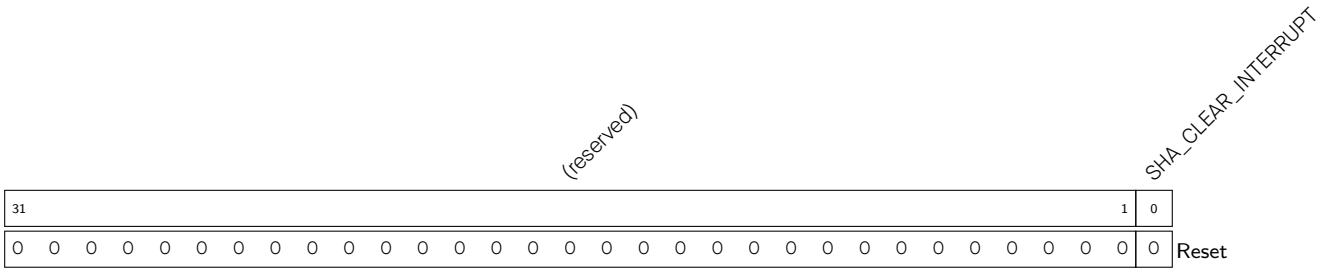
SHA_DMA_START 写 1 启动 SHA 加速器的 DMA-SHA 模式。(WO)

Register 31.9. SHA_DMA_CONTINUE_REG (0x0020)



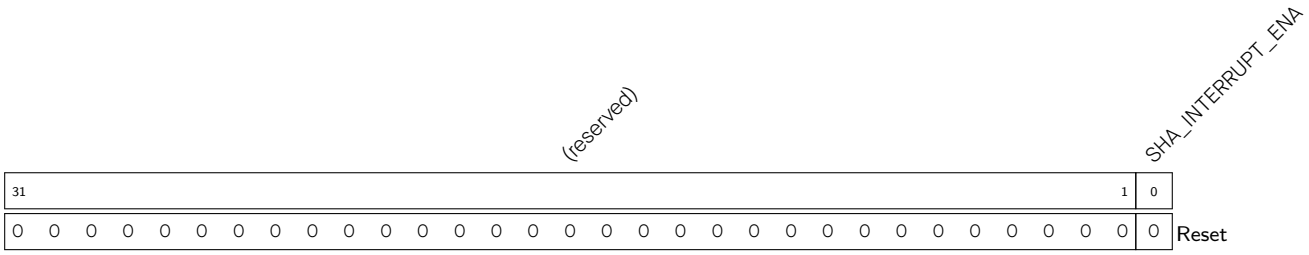
SHA_DMA_CONTINUE 写 1 继续 SHA 加速器的 DMA-SHA 运算。(WO)

Register 31.10. SHA_INT_CLEAR_REG (0x0024)



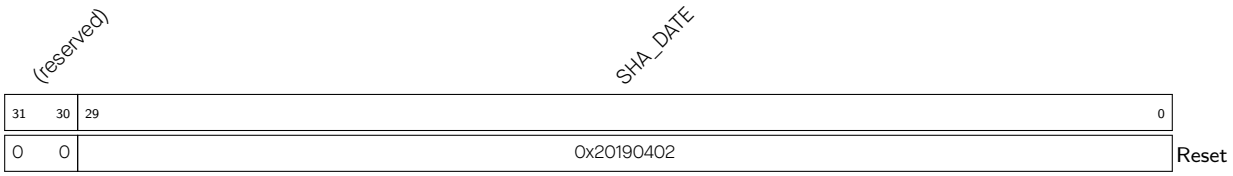
SHA_CLEAR_INTERRUPT 写 1 清除 DMA-SHA 中断。(WO)

Register 31.11. SHA_INT_ENA_REG (0x0028)



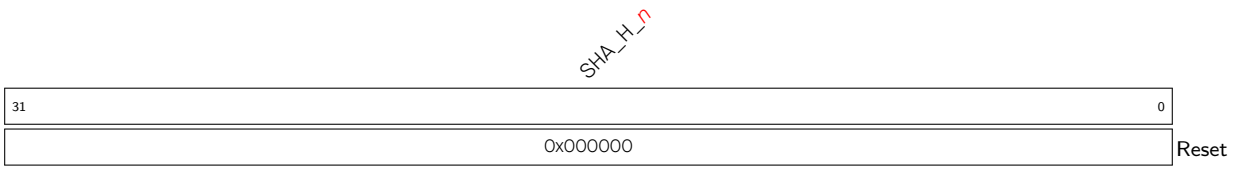
SHA_INTERRUPT_ENA 写 1 使能 DMA-SHA 中断。(R/W)

Register 31.12. SHA_DATE_REG (0x002C)



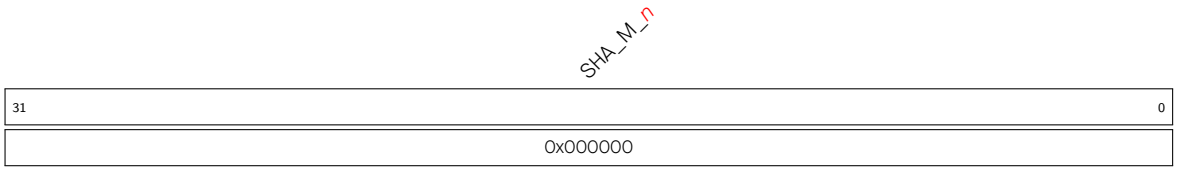
SHA_DATE 版本控制寄存器。(R/W)

Register 31.13. SHA_H_*n*_REG (*n*: 0-15) (0x0040+4**n*)



SHA_H_*n* 表示第 *n* 个 32 位哈希值。(R/W)

Register 31.14. SHA_M_*n*_REG (*n*: 0-31) (0x0080+4**n*)



SHA_M_*n* 表示第 *n* 个 32 位输入信息。(R/W)

第 32 章

RSA 数字签名外设 (RSA_DS)

32.1 概述

RSA 数字签名外设 (RSA_DS) 使用密码学算法，用于验证消息的真实性和完整性。该外设也可用于向服务器验证设备身份，或验证消息是否经过篡改。

ESP32-P4 包含一个 RSA 数字签名外设 (RSA_DS)，可提供硬件加速，高效生成基于 RSA 的数字签名。HMAC 作为密钥导出函数 (Key Derivation Function, KDF)，使用存储在 eFuse 或密钥管理器生成的密钥作为输入密钥，输出 RSA_DS_KEY。随后，RSA_DS 外设使用 RSA_DS_KEY 解密预先加密的参数，计算出签名。上述过程都发生在硬件层面，因此，在计算过程中的所有密钥都对用户不可见，保障了签名操作的安全性。

32.2 主要特性

- 支持长度最大为 4096 位的 RSA 数字签名密钥
- 支持仅限 RSA_DS 外设读取的加密私钥数据
- 支持 SHA-256 摘要，保护私钥数据免遭攻击者篡改

32.3 功能描述

32.3.1 概述

RSA_DS 外设计算 RSA 签名的方式为 $Z = X^Y \bmod M$，其中 Z 是签名，X 是输入消息，Y 和 M 是 RSA 私钥参数。

私钥参数以密文形式存储在 flash 中。用于解密私钥参数的 RSA_DS_KEY 可以来自于 HMAC 模块或密钥管理器模块。当该私钥：

- 由 RSA_DS 外设通过 HMAC 模块计算获取时，生成密钥所需的 HMAC 输入 $HMAC_KEY$ 存放在 eFuse 中，只允许被 HMAC 模块访问。也就是说，只有 RSA_DS 硬件才能解密私钥密文，软件无法获取私钥明文。关于 eFuse 和 HMAC 的相关细节请参照章节 9.4 *eFuse 控制器 (EFUSE)* 和章节 29 *HMAC 加速器 (HMAC)*。
- 由 RSA_DS 外设通过密钥管理器模块获取时，该密钥必须以安全的方式由密钥管理器部署而来。

需要签名时，软件直接将输入消息 X 发送到 RSA_DS 外设。RSA 签名计算完成后，软件将读取签名结果 Z。

为方便描述，此处约定几个符号，它们的作用域仅限于本章。

- 1^s: 表示一个完全由 “1” 组成的长度为 s 位的位串。

- $[x]_s$: 一个长度为 s 位的位串，要求 s 为 8 的整数倍。如果 x 是一个数 ($x < 2^s$)，那么其在位串中遵循小端字节序。x 可以是一个变量，例如 $[Y]_{4096}$，或一个十六进制的常数，比如 $[0x0C]_8$。根据需要，$[x]_t$ 右边可以加上$(s-t)$ 个 0，使字串长度扩展成 s 位，得到 $[x]_s$。例如：$[0x05]_8 = 00000101$，$[0x05]_{16} = 0000010100000000$，$[0x0005]_{16} = 0000000000000101$，$[0x13]_8 = 00010011$，$[0x13]_{16} = 0001001100000000$，$[0x0013]_{16} = 0000000000010011$。
- ||: 表示位串粘接操作符，用于将两个位串前后粘成一个较长的位串。

32.3.2 私钥运算子

私钥运算子 Y（私钥指数）和 M（密钥模数）由用户生成。它们具有特定的 RSA 密钥长度（最大为 4096 位）。RSA 签名操作还额外需要两个运算子，即参数 $\overline{r}$ 和 M'，由 Y 和 M 运算得到。

用户加密运算子 Y、M、$\overline{r}$、M' 与验证摘要，并以密文 C 的形式存储。密文 C 输入到 RSA_DS 外设之后，先由硬件解密，然后参与 RSA 签名运算。有关生成 C 的方法，请参考章节 32.3.3。

RSA_DS 外设支持运算子长度为 $N = 32 \times x$ ($x \in \{1, 2, 3, \ldots, 128\}$) 的 RSA 签名运算 $Z = X^Y \bmod M$。其中，参数的位宽需要满足 RSA 计算的运算子长度要求，即 Z、X、Y、M 和 $\overline{r}$ 的位宽必须相同，且为 N 中的任意值，而 M' 的位宽则始终是 32。更多 RSA 计算相关信息，请参考章节 30 *RSA 加速器 (RSA)* 中的 30.3.2 *大数模幂运算* 部分。

32.3.3 软件准备工作

如果用户想使用 RSA_DS 外设进行数字签名，则需要提前做好软件准备工作，如图 32.3-1 所示。图中左半部分列出了在硬件开始 RSA 签名计算之前软件需要做的准备工作，右半部分则列出了硬件在整个签名计算过程中的计算流程。

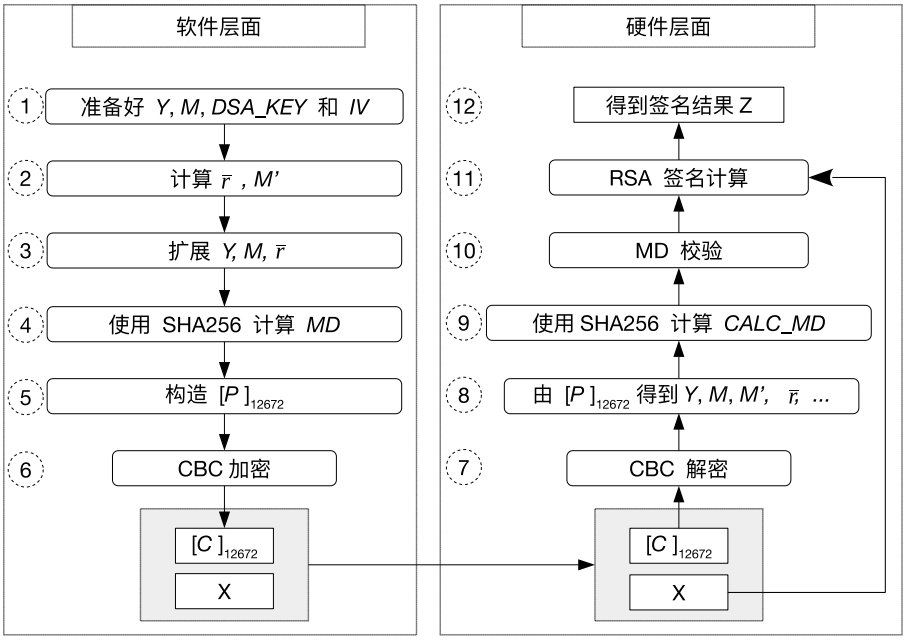


图 32.3-1. 软件准备工作与硬件工作流程

说明：

1. 即便有多次签名计算，也只需进行一次软件准备工作（图 32.3-1 左侧），但每一次签名计算都需要重新进行硬件

运算（图 32.3-1 右侧）。

2. 软件准备需要配置时钟复位，相关信息请参考章节 11 复位和时钟。

用户需要依照图 32.3-1 左侧中指定的步骤来计算 C。详细过程如下：

- **步骤 1**：按照章节 32.3.2 所述，准备好符合运算子长度要求的 RSA 私钥运算子 Y 和 M。定义 $[L]_{32} = \frac{N}{32} - 1$（比如，对于 RSA 4096，$[L]_{32}$ == [0x80-1]$_{32}$）。根据私钥的生成方法准备 $[RSA_DS_KEY]_{256}$：
  - 当使用 HMAC 作为密钥来源时：准备好 $[HMAC_KEY]_{256}$，并计算出 $[RSA_DS_KEY]_{256}$，即 RSA_DS_KEY = HMAC-SHA256($[HMAC_KEY]_{256}, 1^{256}$)。
  - 当使用密钥管理器作为密钥来源时：准备好 256 位的 RSA_DS_KEY。

  接着，生成一个符合 AES-CBC 块加密算法要求的随机 $[IV]_{128}$。有关 AES 的更多信息，请参考章节 27 AES 加速器 (AES)。
- **步骤 2**：根据 M 求解 $\overline{r}$ 和 M'。
- **步骤 3**：扩展 Y、M 和 $\overline{r}$，得到 $[Y]_{4096}$、$[M]_{4096}$ 和 $[\overline{r}]_{4096}$。此步骤是由于 Y、M 和 $\overline{r}$ 的最大位宽为 4096，故位宽小于 4096 的运算子需要扩展至 4096，而位宽等于 4096 的则不需要扩展。
- **步骤 4**：使用 SHA-256 计算 MD 校验码：$[MD]_{256}$ = SHA256 ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **步骤 5**：构造 $[P]_{12672}$ = ($[Y]_{4096}||[M]_{4096}||[\overline{r}]_{4096}||[Box]_{384}$)。其中，$[Box]_{384}$ = ($[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$)，$[\beta]_{64}$ 是符合 PKCS#7 封装方式的追加码，由 8 个字节码 0x80 组成，即 [0x0808080808080808]$_{64}$，目的在于使 P 的长度为 128 位的整数倍。
- **步骤 6**：计算密文形式的私钥参数 C = $[C]_{12672}$ = AES-CBC-ENC ($[P]_{12672}, [RSA_DS_KEY]_{256}, [IV]_{128}$)，长度为 1200 字节。$C$ 也可以表示为 C = $[C]_{12672}$ = ($[\widehat{Y}]_{4096}||[\widehat{M}]_{4096}||[\widehat{\overline{r}}]_{4096}||[\widehat{Box}]_{384}$)。其中，$[\widehat{Y}]_{4096}$、$[\widehat{M}]_{4096}$、$[\widehat{\overline{r}}]_{4096}$、$[\widehat{Box}]_{384}$ 是 C 的四个子参数，分别对应 $[Y]_{4096}$、$[M]_{4096}$、$[\overline{r}]_{4096}$、$[Box]_{384}$ 的密文形式。

32.3.4 RSA_DS 硬件工作流程

每次需要计算数字签名时，都会触发硬件操作。硬件需要三个输入信息：预先生成的私钥密文 C、唯一消息 X 以及 IV。

RSA_DS 外设的工作流程可以分为三个阶段：

1. **解析阶段，即图 32.3-1 中的步骤 7 和步骤 8**

   解析过程是图 32.3-1 中步骤 6 的逆过程。RSA_DS 外设将调用 AES 硬件加速器以 CBC 块模式对输入的密文信息 C 进行解密，获取明文信息。该过程可以表示为 P = AES-CBC-DEC (C, RSA_DS_KEY, IV)。其中，IV 就是 $[IV]_{128}$，由用户直接指定；$[RSA_DS_KEY]_{256}$ 由：
   - 硬件 HMAC 提供，通过存储在 eFuse 中的 $HMAC_KEY$ 得到，软件无法获取。
   - 硬件密钥管理器提供时，通过部署在密钥管理器中的 RSA_DS_KEY 得到，软件无法获取。

   通过 P，RSA_DS 外设能够解析出 $[Y]_{4096}$、$[M]_{4096}$、$[\overline{r}]_{4096}$、$[M']_{32}$、$[L]_{32}$、MD 校验码和追加码 $[\beta]_{64}$，这相当于步骤 5 的逆过程。
2. **校验阶段，即图 32.3-1 中的步骤 9 和步骤 10**

RSA_DS 外设会执行两种校验操作：MD 校验和填充 (padding) 校验。由于填充校验和 MD 校验同步进行，因此填充校验未在图 32.3-1 中体现。

- MD 校验：RSA_DS 外设调用 SHA-256 进行哈希计算获取哈希值 $[CALC_MD]_{256}$（即步骤 4）。然后，将 $[CALC_MD]_{256}$ 与 MD 校验码 $[MD]_{256}$ 作比较，当且仅当二者相同时，MD 校验通过。
- 填充校验：RSA_DS 外设检查解析阶段解析出的追加码 $[\beta]_{64}$ 是否符合 PKCS#7 标准，当且仅当符合标准时，填充校验通过。

仅当 MD 校验通过后，RSA_DS 外设才会执行后续计算，否则 RSA_DS 外设将拒绝执行。如果填充校验失败，将生成警告信息，但不会影响 RSA_DS 外设的后续操作。

3. **计算阶段，即图 32.3-1 中的步骤 11 和步骤 12**

   RSA_DS 外设将把用户输入的 X，以及解析得到的 Y、M 和 $\overline{r}$ 都视为大数，结合解析得到的 M'，构成了大数模幂运算 $X^Y \bmod M$ 的所有必要输入参数。大数模幂运算的运算长度由 L 的值唯一指定。RSA_DS 外设调用 RSA 完成大数模幂运算 $Z = X^Y \bmod M$，Z 为签名结果。

32.3.5 RSA_DS 软件工作流程

每次需要计算数字签名时，都应执行以下软件操作。输入内容为预先生成的私钥密文 C、唯一的消息 X、IV。本节中的软件步骤会触发章节 32.3.4 中描述的硬件工作流程。
下述流程基于一个假设：软件已经调用了 HMAC 模块，HMAC 已经根据 $HMAC_KEY$ 计算出了 RSA_DS_KEY。

1. **准备工作**：准备好 C、X、IV。具体方法请参考章节 32.3.3 。
2. **启动 RSA_DS 外设**：对寄存器 DSA_SET_START_REG 写 1。
3. **配置 DSA_KEY_SOURCE_REG 选择 RSA_DS_KEY 的来源，并检查是否已经准备好**：
   - 配置 DSA_KEY_SOURCE_REG 为 1，即选择密钥管理器中部署的 RSA_DS_KEY。
     检查 RSA_DS_KEY 已经成功部署，检查方式为：读取寄存器域 KEYMNG_KEY_DS_VLD，当其值为 1 时，代表 RSA_DS_KEY 已经成功部署并准备好。
   - 配置 DSA_KEY_SOURCE_REG 为 0，即选择从 HMAC 模块计算 RSA_DS_KEY。
     轮询寄存器 DSA_QUERY_BUSY_REG 直到读到 0。

   如果 DSA_QUERY_BUSY_REG 超过 1 ms 还没读到 0，则说明 HMAC 初始化出现了问题。此时，软件应当读寄存器 DSA_QUERY_KEY_WRONG_REG，根据返回值判断具体是下述哪一种情况：
   - 如果读到零值，说明 HMAC 未被调用。
   - 如果读到非零值 (1 ~ 15)，则说明 HMAC 被调用过，但是 RSA_DS 外设没有成功从 HMAC 模块获取到 RSA_DS_KEY。此种情况可能是因为受到了其他程序的干扰。
4. **配置寄存器**：将 IV block 中的内容写入寄存器 DSA_IV_*m*_REG (*m*: 0 ~ 3)。有关 IV block 的更多信息，请参考章节 *27 AES 加速器 (AES)*。
5. **将 X 写入存储器 DSA_X_MEM**：将 X_i ($i \in \{0, 1, \ldots, n-1\}$) 写入存储器 DSA_X_MEM，容量为 96 个字 (word)，其中 $n = \frac{N}{32}$ 。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。当 X 的长度小于 96 个字时，存储器 DSA_X_MEM 中有一部分空间未使用，该部分空间中的数据可以是任意值。
6. **将 C 写入存储器**：即将 C 中的四个子参数分别写入对应的存储器。
   - 将 $\widehat{Y}_i$ ($i \in \{0, 1, \ldots, 127\}$) 写入存储器 DSA_Y_MEM。

- 将 $\widehat{M_i}$ ($i \in \{0, 1, \ldots, 127\}$) 写入存储器 DSA_M_MEM。
- 将 $\widehat{r_i}$ ($i \in \{0, 1, \ldots, 127\}$) 写入存储器 DSA_RB_MEM。
- 将 $\widehat{Box_i}$ ($i \in \{0, 1, \ldots, 11\}$) 写入存储器 DSA_BOX_MEM。

  其中存储器 DSA_Y_MEM、DSA_M_MEM、DSA_RB_MEM 的容量均为 128 个字，而存储器 DSA_BOX_MEM 容量只有 12 个字。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

7. **启动计算**：对寄存器 DSA_SET_ME_REG 写入 1。
8. **等待运算结束**：轮询寄存器 DSA_QUERY_BUSY_REG 直到读到 0。
9. **检查校验结果**：读寄存器 DSA_QUERY_CHECK_REG，根据返回值决定后续操作。
   - 如果返回值为 0，则说明填充校验通过，MD 校验通过，可以继续读取 Z 结果值。
   - 如果返回值为 1，则说明填充校验通过，但 MD 校验失败。Z 结果值无效，跳至步骤 11。
   - 如果返回值为 2，则说明填充校验失败，但 MD 校验通过，用户可以继续读取 Z 结果值。但仍需注意的是，此时数据填充不符合 PKCS#7 封装方式。
   - 如果返回值为 3，则说明填充校验失败，且 MD 校验失败。这种情况说明有致命错误发生。Z 结果值无效。跳至步骤 11。
10. **读出运算结果**：从存储器 DSA_Z_MEM 读出运算结果 Z_i ($i \in \{0, 1, \ldots, n-1\}$)，其中 $n = \frac{N}{32}$ 。Z 以小端字节序存储在存储器中。
11. **退出计算环境**：对寄存器 DSA_SET_FINISH_REG 写入 1，然后轮询寄存器 DSA_QUERY_BUSY_REG 直到读到 0。

RSA_DS 外设退出计算环境后，所有输入/输出寄存器和存储器中的数据都已经被抹除（清零）。

32.4 存储器列表

请注意，这里的起始地址和结束地址都是相对于 RSA 数字签名外设基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

名称	描述	大小（字节）	起始地址	结束地址	访问
DSA_Y_MEM	存储器 Y	512	0x0000	0x01FF	WO
DSA_M_MEM	存储器 M	512	0x0200	0x03FF	WO
DSA_RB_MEM	存储器 r	512	0x0400	0x05FF	WO
DSA_BOX_MEM	存储器 Box	48	0x0600	0x062F	WO
DSA_X_MEM	存储器 X	512	0x0800	0x09FF	WO
DSA_Z_MEM	存储器 Z	512	0x0A00	0x0BFF	RO

32.5 寄存器列表

本小节的所有地址均为相对于 RSA 数字签名外设基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

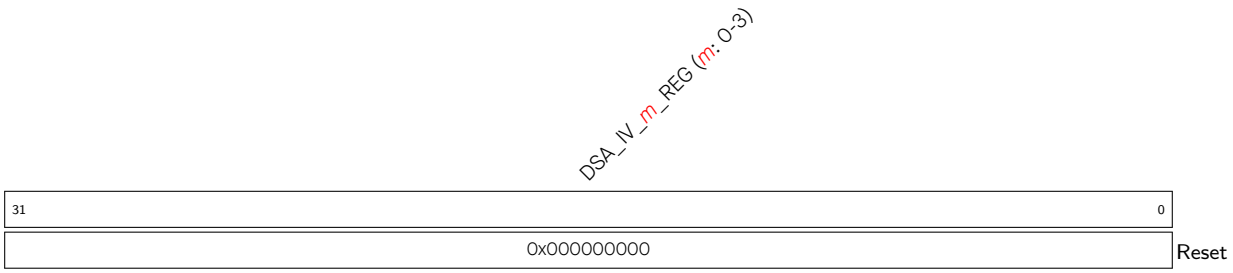
名称	描述	地址	访问
配置寄存器			
DSA_IV_0_REG	IV block 数据	0x0630	WO
DSA_IV_1_REG	IV block 数据	0x0634	WO
DSA_IV_2_REG	IV block 数据	0x0638	WO
DSA_IV_3_REG	IV block 数据	0x063C	WO
DSA_KEY_SOURCE_REG	配置 RSA_DS_KEY 来源	0x0E18	R/W
状态/控制寄存器			
DSA_SET_START_REG	启动 RSA_DS 外设	0x0E00	WO
DSA_SET_ME_REG	开始 RSA_DS 运算	0x0E04	WO
DSA_SET_FINISH_REG	结束 RSA_DS 运算	0x0E08	WO
DSA_QUERY_BUSY_REG	代表 RSA_DS 外设状态	0x0E0C	RO
DSA_QUERY_KEY_WRONG_REG	查询 RSA_DS_KEY 未准备好的原因	0x0E10	RO
DSA_QUERY_CHECK_REG	查询校验结果	0x0E14	RO
版本寄存器			
DSA_DATE_REG	版本控制寄存器	0x0E20	R/W

32.6 寄存器

本小节的所有地址均为相对于 RSA 数字签名外设基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

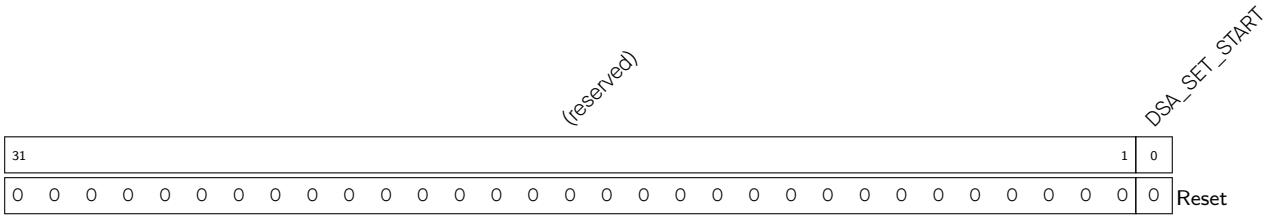
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 32.1. DSA_IV_*m*_REG (*m*: 0-3) (0x0630+4**m*)



DSA_IV_*m*_REG (*m*: 0-3) 写入 IV block 数据。(WO)

Register 32.2. DSA_SET_START_REG (0x0E00)



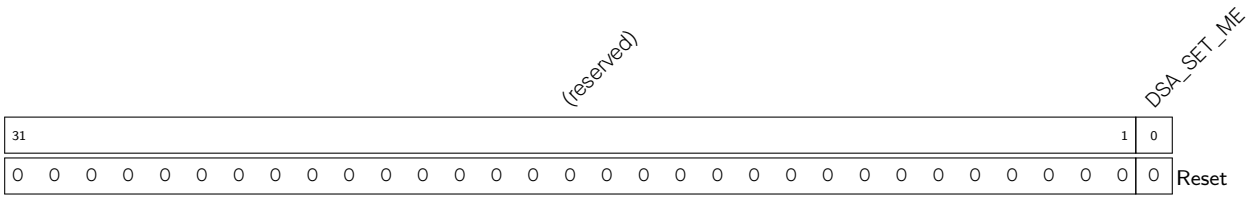
DSA_SET_START 配置是否启动 RSA_DS 外设。

0：无效

1：启动 RSA_DS 外设

(WO)

Register 32.3. DSA_SET_ME_REG (0x0E04)



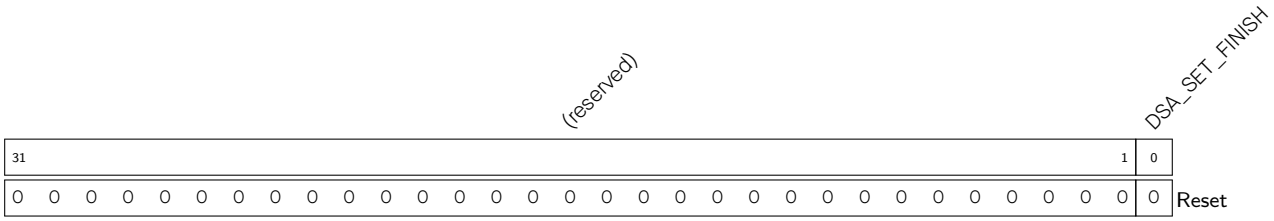
DSA_SET_ME 配置是否启动计算。

0：无效

1：启动计算

(WO)

Register 32.4. DSA_SET_FINISH_REG (0x0E08)



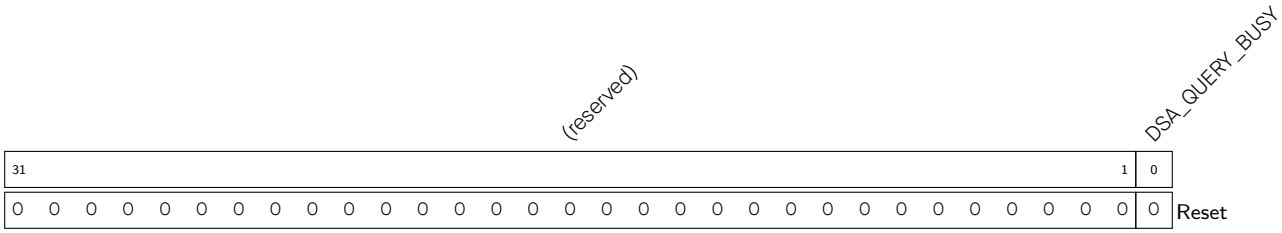
DSA_SET_FINISH 配置是否结束运算。

0：无效

1：结束运算

(WO)

Register 32.5. DSA_QUERY_BUSY_REG (0x0E0C)



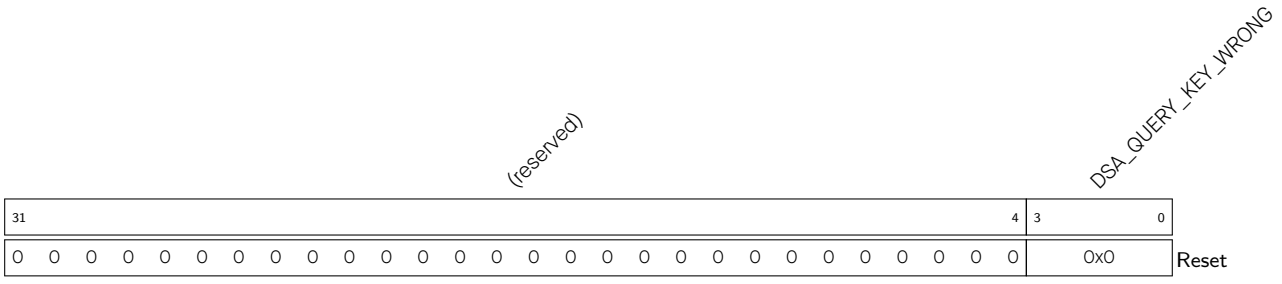
DSA_QUERY_BUSY 表示 RSA_DS 外设是否空闲。

0：RSA_DS 外设空闲

1：RSA_DS 外设正忙

(RO)

Register 32.6. DSA_QUERY_KEY_WRONG_REG (0x0E10)



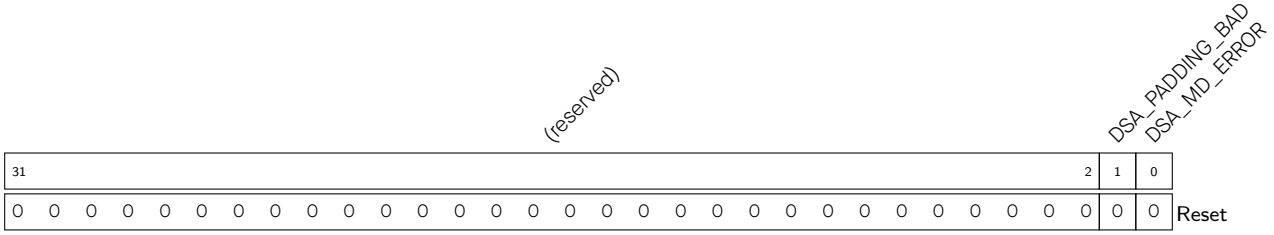
DSA_QUERY_KEY_WRONG 表示 HMAC 模块的错误情况。

0：HMAC 未被调用

1-15：HMAC 曾被调用，RSA_DS 外设没有成功从 HMAC 模块获取到 RSA_DS_KEY

(RO)

Register 32.7. DSA_QUERY_CHECK_REG (0x0E14)



DSA_PADDING_BAD 表示填充校验是否通过。

0：填充校验通过

1：填充校验失败

(RO)

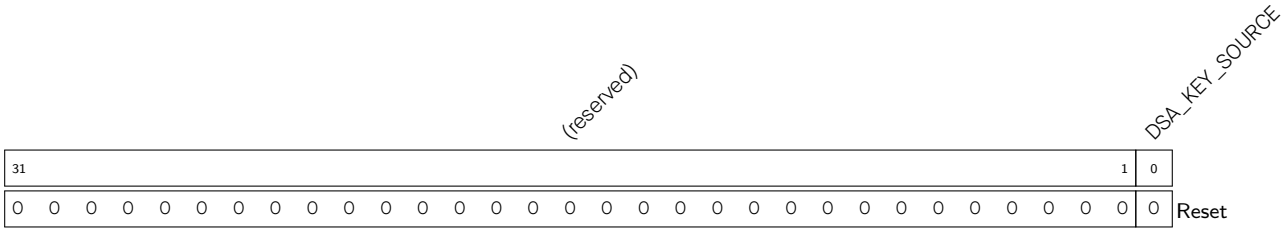
DSA_MD_ERROR 表示 MD 校验是否通过。

0：MD 校验通过

1：MD 校验失败

(RO)

Register 32.8. DSA_KEY_SOURCE_REG (0x0E18)



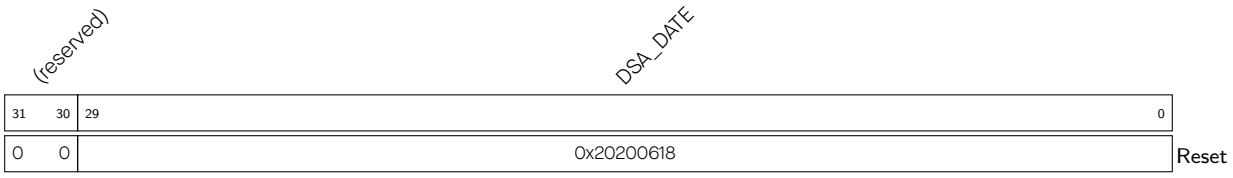
DSA_KEY_SOURCE 表示 RSA_DS_KEY 的来源。

0：HMAC

1：密钥管理器

(R/W)

Register 32.9. DSA_DATE_REG (0x0E20)



DSA_DATE 版本控制寄存器。(R/W)

第 33 章

ECDSA 数字签名外设 (ECDSA_DS)

33.1 概述

在密码学中，椭圆曲线数字签名算法 (ECDSA) 是数字签名算法 (DSA) 基于椭圆曲线密码学的一种变体。

ESP32-P4 的 ECDSA 数字签名外设 (ECDSA_DS) 可在安全高效的环境中计算 ECDSA 签名，不仅可保证签名过程的机密性，还实现了快速计算，非常适合对加密安全性和加密速度有高要求的应用。通过 ECDSA_DS 模块，用户可以在不牺牲性能的情况下保证数据得到保护。

33.2 主要特性

ESP32-P4 的 ECDSA_DS 模块支持：

- 数字签名的验证
- 三种椭圆曲线，分别是 P-192、P-256 和 P-384（具体定义见 FIPS 186-5 规范）
- 多种哈希算法，包括 SHA-224、SHA-256、SHA-384、SHA-512、SHA-512-224 和 SHA-512-256（具体定义见 FIPS PUB 180-4 规范）
- 基于不同工作状态的寄存器访问控制，以确保信息安全

33.3 ECDSA 背景知识

33.3.1 域参数

ECDSA 使用的参数通常被称为域参数，可用于生成密钥、生成签名和验证签名等运算，主要包括有限域中定义的椭圆曲线参数、曲线上的基点和基点阶数等。

ECDSA 使用的域参数主要包括两大类：

- 椭圆曲线域参数：
  - 质数模数 p，指定定义椭圆曲线的有限域的大小。
  - 系数 a 和 b，用于指定椭圆曲线方程。
  - 曲线上的基点 G，用于生成公钥。
  - 基点的阶数 n，代表在椭圆曲线上对基点 G 进行重复点加的次数。
- 哈希函数参数：
  - 哈希算法，用于将要签名的消息生成固定长度的哈希值。
  - 哈希值的长度，决定签名的大小。

这些参数共同定义了 ECDSA 域。

33.3.2 密钥生成

ECDSA 中的密钥生成过程如下所述：

1. 选择一个椭圆曲线域，定义如下参数：
   - 质数模数 p
   - 系数 a 和 b
   - 基点 G 和其阶数 n
2. 生成私钥：
   - 私钥 d 为在 1 到 n-1 之间随机选择的整数。注意，这里必须确保私钥是真正随机的（不可预测或复制）。因此，使用一个安全的随机数发生器 *(RNG)*非常重要。
   - 私钥用于生成签名，需妥善保密保管。
3. 计算公钥：
   - 公钥 Q 的计算方式为 $Q = dG$。
4. 导出公钥：
   - 公钥 Q 通常以 (Qx，Qy) 表示，可公开给他人。

33.3.3 签名生成

ECDSA 签名生成过程如下所述：

1. 选择一条消息 m，用于签名。
2. 计算消息 m 的哈希值 e：e = HASH(m)，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息哈希值 e 的摘要 z：令 z 为 e 的左端 L_n 位，其中 L_n 是基点阶数 n 的位长度。
4. 选择一个随机数 k：范围在 1 到 n-1 之间，其中 n 是椭圆曲线上基点的阶数。
5. 计算签名：签名计算如下：
   (a) 计算点 $(x, y) = kG$
   (b) 计算 $r = x \bmod n$。如果 r 等于零，返回上一步并选择新的 k 值。
   (c) 计算 $s = k^{-1} * (z + d * r) \bmod n$。如果 s 等于零，返回步骤 4 并选择新的 k 值。
   (d) 签名为 (r, s)。
6. 发送消息 m 和签名 (r, s) 给接收者。

33.3.4 签名验证

接收方可以使用密钥生成阶段导出的公钥，来验证对应私钥生成的签名。验证过程涉及检查签名是否由对应的私钥生成，以及签名是否对给定的消息有效。

ECDSA 签名验证过程如下所述：

1. 获得公钥 Q，接收消息 m 和签名 (r, s)。

2. 计算消息的哈希值：e 等于 HASH(m)，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息的摘要：令 z 为 e 的左端 L_n 位，其中 L_n 是基点阶数 n 的位长度。
4. 验证签名：签名验证如下所示：
   (a) 验证 r 和 s 是否是介于 1 到 n-1 之间的整数，其中 n 是椭圆曲线上基点的阶数。如果 r 或 s 超出此范围，则签名无效。
   (b) 计算 $u_1 = z * s^{-1} \bmod n$ 和 $u_2 = r * s^{-1} \bmod n$。
   (c) 计算点 $(x_1, y_1) = u_1 * G + u_2 * Q$，其中 G 是椭圆曲线上的基点，Q 是与用于签名生成的私钥相关联的公钥。
   (d) 验证 $r = x_1 \bmod n$。如果 r 不等于 $x_1 \bmod n$，则签名无效。
5. 接受或拒绝签名：如果签名有效，接收方可以确信消息没有被篡改，且消息来自预期的发送方，可以接收。否则，接收方将拒绝并判定为无效消息。

33.4 功能描述

本节描述了 ESP32-P4 的 ECDSA_DS 模块的详细信息。

33.4.1 ECDSA_DS 工作模式

ESP32-P4 内置的 ECDSA_DS 支持签名验证。

用户可以通过配置 ECDSA_ECC_CURVE，根据下表 33.4-1 选择 ECDSA_DS 模块使用的椭圆曲线。

表 33.4-1. ECDSA_DS 椭圆曲线选择

ECDSA_ECC_CURVE	椭圆曲线
0	P-192
1	P-256
2	P-384

用户可以通过配置 ECDSA_SHA_MODE，根据下表 33.4-2 选择用于消息哈希的 SHA 算法。

表 33.4-2. ECDSA_DS SHA 算法

ECDSA_SHA_MODE	SHA 算法
1	SHA-224
2	SHA-256
3	SHA-384
4	SHA-512
5	SHA-512-224
6	SHA-512-256
其他	无效

此外，用户可以通过查询 ECDSA_STATE_REG 寄存器并将返回值与下表 33.4-3 进行比较，来确认 ECDSA_DS 模块的工作状态。

表 33.4-3. ECDSA_DS 工作状态

ECDSA_STATE_REG	状态	描述
0	IDLE	空闲或已完成的运算，对应于 IDLE 阶段。
1	LOAD	等待用户将信息加载到 ECDSA_DS 中，对应于 LOAD 阶段。
2	保留	–
3	BUSY	硬件正在运行中，对应于 PREP、PROC 和 POST 阶段。

33.4.2 数据和数据块

ESP32-P4 的 ECDSA_DS 模块会用到的数据位宽为 192、256、384 或 512 位。

假设一个数据为 $D[255:0]$，则其可以被划分为八个 32 位的数据块 $D[n][31:0](n=0,1,\cdots,7)$，序号数低的数据块对应二进制低位。以 256 位的数据为例：

$$D[255:0]=D[7][31:0],D[6][31:0],D[5][31:0],D[4][31:0],D[3][31:0],D[2][31:0],D[1][31:0],D[0][31:0]$$

33.4.2.1 数据写入

数据写入即将数据写入进一个内存块的过程，该数据为 ECDSA_DS 的输入数据。具体而言，将数据写入一个 ECDSA_DS 内存块相当于将该数据 $D[n][31:0](n=0,1,\cdots,7)$ 依次写入“该内存块起始地址 + $4\times n$”。以 256 位的数据为例：

- 将 $D[0]$ 写入“起始地址”
- 将 $D[1]$ 写入“起始地址 + 4”
- $\cdots$
- 将 $D[7]$ 写入“起始地址 + 28”

说明：
进行数据写入操作时，只需要写入数据长度的数据块即可，不需要在数据的高位补 0。例如，在写入 192 位长度的数据时，不需要在数据的高位补 0，只需要存储 192 位的数据。

33.4.2.2 数据读取

数据读取即将一个数据从一个内存块读出的过程，该数据为 ECDSA_DS 的输出数据。具体来说，从一个 ECDSA_DS 内存块读数据相当于从“该内存块起始地址 + $4\times n$”依次读出 $D[n][31:0]$。以 256 位的数据为例：

- 从“起始地址”读出 $D[0]$
- 从“起始地址 + 4”读出 $D[1]$
- $\cdots$
- 从“起始地址 + 28”读出 $D[7]$

说明：
进行数据读取操作时，只需要读取数据长度的数据块即可，例如读取 192 位长度的数据时，只需要读取低 192 位（即 6 个数据块）的数据。

33.4.2.3 消息填充

SHA 加速器仅能处理长度为 512 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件运算将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 L_M 位，则填充步骤如下：

1. 首先，在待处理信息后填充 1 个 "1"；
2. 随后，再填充 L_A 个 "0"。其中，L_A 为满足 $L_M + 1 + L_A \equiv 448 \ mod \ 512$ 的最小非负数解；
3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 L_M 的值。

更多详情，请参考 《FIPS PUB 180-4 规范》 > 章节 "Padding the Message"。

33.4.2.4 解析消息

在完成信息填充后，我们还需将待处理信息（及其填充）解析为 N 个 512 位的信息块，即 $M^{(1)}$、$M^{(2)}$、...、$M^{(N)}$。

说明：
1. 有关 "解析消息" 的详细信息，请参考 FIPS PUB 180-4 规范 > 章节 "Parsing the Message"。
2. 有关 "信息块" 及相关概念的描述，请参考 FIPS PUB 180-4 规范 > 章节 "Glossary of Terms and Acronyms"。

33.4.3 安全功能

为确保运算过程的安全性，ECDSA_DS 模块实现了多种安全功能。

33.4.3.1 抗攻击性能

ESP32-P4 的 ECDSA_DS 模块每次执行操作时，都会利用 ECC 加速器的强抗攻击性能（参考小节 28.4.3 提升抗攻击性能）。这意味着 ECDSA_DS 在每次生成和验证签名时：

- 对于每次操作，执行延迟保持恒定。
- 功耗波动最小化。

这为 ESP32-P4 的 ECDSA_DS 模块提供了强大的抗攻击性能。

33.4.3.2 基于状态的寄存器访问控制

ESP32-P4 的 ECDSA_DS 模块实施了基于状态的寄存器访问控制，可有效防止运行过程中可能发生的非法配置篡改或数据访问，从而保证密钥的安全性。

具体来说，ECDSA_DS 的寄存器在不同工作状态下的访问权限是不同的。例如，ECDSA_CONF_REG 寄存器仅在 ECDSA_DS 处于 IDLE 状态时允许读写运算。这样一来，当 ECDSA_DS 处于其他状态（如 LOAD、BUSY）时，

该寄存器中的配置信息则不允许读取或写入，从而得到很好的保护。有关 ECDSA_DS 所有工作状态的详细信息，请见表 33.4-3。

有关基于状态的 ECDSA_DS 寄存器访问控制的详细信息，请参阅寄存器列表。

33.4.3.3 硬件占用

在运算过程中，ESP32-P4 的 ECDSA_DS 模块将占用以下硬件模块：

- SHA 加速器
- ECC 加速器

其中，SHA 加速器在触发 ECDSA_SHA_RELEASE_INT 时将释放，而 ECC 加速器在整个 ECDSA_DS 运算中将被占用。

说明：

硬件占用是一种保护复用模块和存储空间的机制。当一个模块被硬件占用时，用户将无法执行以下运算：

- 读取或写入模块的寄存器或存储器数据。
- 关闭模块时钟。
- 复位模块。

在硬件占用结束时，占用的模块将自动复位。此外，当用户对主模块执行软件复位时，所有占用的模块将同时复位。

33.5 配置流程

33.5.1 ECDSA_DS 流程

ECDSA_DS 的完整工作流程包括以下五个阶段。

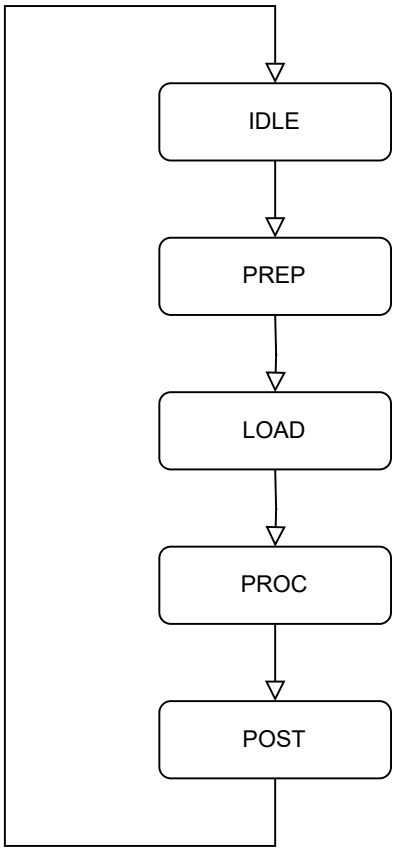


图 33.5-1. ECDSA_DS 流程

每个阶段的详细配置流程在下文中描述。

33.5.1.1 IDLE 阶段

在 IDLE 阶段：

1. 配置静态参数，包括 eFuse 位 ECDSA_KEY，即 ECDSA_DS 中的私钥 d 的值。为了正确配置 eFuse 中的密钥值，用户需要将密钥值写入 KEY*n*（n = 0 ~ 5），并将相应的 EFUSE_KEY_PURPOSE_*n* 设置为 ECDSA_KEY。有关更详细的配置步骤，请参阅第 9.4 章 *eFuse 控制器 (EFUSE)*。
2. 配置配置寄存器，包括以下字段：
   (a) ECDSA_ECC_CURVE：选择 ECDSA_DS 模块的椭圆曲线。
   (b) ECDSA_SOFTWARE_SET_Z：选择使用直接输入的 z。
3. 配置寄存器字段 ECDSA_START 以进入 PREP 阶段。

33.5.1.2 PREP 阶段

在 PREP 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA_DS 执行准备工作。

通过轮询 ECDSA_BUSY 的状态直到其不为 BUSY，表示 PREP 阶段已结束。然后 ECDSA_DS 将自动进入 LOAD 阶段。

33.5.1.3 LOAD 阶段

在 LOAD 阶段：

1. 使用以下选项之一将输入 z 提供给 ECDSA_DS：
   - 直接输入 z：将 z 写入 ECDSA_Z_MEM。
   - ECDSA_DS SHA 接口：从消息生成 z。有关详细信息，请参阅第 33.5.1.6 节。
2. 写入签名验证所需的签名和公钥内容：
   (a) 将签名（r，s）写入 ECDSA_R_MEM 和 ECDSA_S_MEM。
   (b) 将公钥（Qx，Qy）写入 ECDSA_QX_MEM 和 ECDSA_QY_MEM。
3. 将 1 写入 ECDSA_LOAD_DONE，表示配置完成。然后 ECDSA_DS 将自动进入 PROC 阶段。

33.5.1.4 PROC 阶段

在 PROC 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA_DS 执行 ECDSA 签名验证。

通过轮询 ECDSA_BUSY 直到其不为 BUSY，表示 PROC 阶段已结束。然后 ECDSA_DS 将自动进入 POST 阶段。

33.5.1.5 POST 阶段

在 POST 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA_DS 执行运算的收尾工作。

通过轮询 ECDSA_BUSY 直到不再为 BUSY，表示 POST 阶段已结束。然后 ECDSA_DS 将自动返回 IDLE 阶段。

33.5.1.6 ECDSA_DS SHA 接口

ESP32-P4 的 ECDSA_DS 模块可以自动执行哈希运算并基于直接输入的消息生成 z。

对于消息哈希，ECDSA_DS 模块支持 SHA 算法 SHA-224（仅当选择 P-192 作为椭圆曲线时有效）和 SHA-256。

要使用 ECDSA_DS SHA 接口，请完成以下步骤：

1. 根据第 33.4.2.3 节中的描述，对消息进行填充。
2. 将消息及其填充解析为消息块。详细信息请参见第 33.4.2.4 节。
3. 处理当前消息块。
   - 将当前消息块写入 ECDSA_MEM_M。
4. 启动 ECDSA_DS SHA 接口 [1]。
   - 如果这是第一次执行此步骤，请将 1 写入 ECDSA_SHA_START 以启动 ECDSA_DS SHA 接口；
   - 如果不是第一次执行此步骤 [2]，请将 1 写入 ECDSA_SHA_CONTINUE 以继续运算。
5. 查询当前信息块的处理进度。
   - 轮询寄存器 ECDSA_SHA_BUSY，直到其为 IDLE，表示接口已完成当前消息块的运算。
6. 处理下一个消息块：

- 如有下一个消息块，则返回到步骤 3。
- 如没有更多的消息块，退出。

说明：

1. 在此步骤中，软件还可以在接口启动 SHA 运算时将下一个要处理的消息块（如有）写入寄存器 ECDSA_MEM_M，以节省时间。
2. 即正在继续之前暂停的 ECDSA_DS SHA 接口运算。

33.5.2 时钟和复位

ESP32-P4 的 ECDSA_DS 模块有一个时钟模块和一个复位模块，其激活仅需使能 HP_SYS_CLKRST_PERI_CLK_CTRL25_REG 外围时钟的 HP_SYS_CLKRST_CRYPTO_ECDSA_CLK_EN 位，并同时清零 HP_SYS_CLKRST_HP_RST_EN2_REG 寄存器的 HP_SYS_CLKRST_RST_EN_ECDSA 位。有关如何配置 ECDSA_DS 时钟和复位的详细信息，请参考第 11 章*复位和时钟*。

33.6 中断

ESP32-P4 的 ECDSA_DS 模块可以生成一个中断信号 ECDSA_INTR 并发送给 中断矩阵。

ECDSA_DS 模块有四个中断源可产生中断信号 ECDSA_INTR，如下所示。每个中断源都有一组通用的配置寄存器，详见章节 *中断配置寄存器*。

- ECDSA_PREP_DONE_INT：在 PREP 阶段结束时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_PREP_DONE_INT_RAW：存储 ECDSA_PREP_DONE_INT 的原始中断状态。
  - ECDSA_PREP_DONE_INT_ST：代表 ECDSA_PREP_DONE_INT 中断的状态。通过 ECDSA_PREP_DONE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_PREP_DONE_INT_ENA：启用/禁用 ECDSA_PREP_DONE_INT 中断。
  - ECDSA_PREP_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_PREP_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_PREP_DONE_INT_RAW 和 ECDSA_PREP_DONE_INT_ST 将被清除。
- ECDSA_PROC_DONE_INT：在 PROC 阶段结束时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_PROC_DONE_INT_RAW：存储 ECDSA_PROC_DONE_INT 的原始中断状态。
  - ECDSA_PROC_DONE_INT_ST：代表 ECDSA_PROC_DONE_INT 中断的状态。通过 ECDSA_PROC_DONE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_PROC_DONE_INT_ENA：启用/禁用 ECDSA_PROC_DONE_INT 中断。
  - ECDSA_PROC_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_PROC_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_PROC_DONE_INT_RAW 和 ECDSA_PROC_DONE_INT_ST 将被清除。
- ECDSA_POST_DONE_INT：在 POST 阶段结束时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_POST_DONE_INT_RAW：存储 ECDSA_POST_DONE_INT 的原始中断状态。
  - ECDSA_POST_DONE_INT_ST：代表 ECDSA_POST_DONE_INT 中断的状态。通过 ECDSA_POST_DONE_INT_RAW 字段启用/禁用此字段生成。

  - ECDSA_POST_DONE_INT_ENA：启用/禁用 ECDSA_POST_DONE_INT 中断。
  - ECDSA_POST_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_POST_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_POST_DONE_INT_RAW 和 ECDSA_POST_DONE_INT_ST 将被清除。
- ECDSA_SHA_RELEASE_INT：在 SHA 释放时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_SHA_RELEASE_INT_RAW：存储 ECDSA_SHA_RELEASE_INT 的原始中断状态。
  - ECDSA_SHA_RELEASE_INT_ST：代表 ECDSA_SHA_RELEASE_INT 中断的状态。通过 ECDSA_SHA_RELEASE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_SHA_RELEASE_INT_ENA：启用/禁用 ECDSA_SHA_RELEASE_INT 中断。
  - ECDSA_SHA_RELEASE_INT_CLR：将此位设置为 1 以清除 ECDSA_SHA_RELEASE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_SHA_RELEASE_INT_RAW 和 ECDSA_SHA_RELEASE_INT_ST 将被清除。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

33.7 存储器块

表 33.7-1. ECDSA_DS 存储器块

存储器块	大小（字节）	起始地址*	结束地址*	访问权限
ECDSA_MEM_M	64	0x280	0x2BF	R/W
ECDSA_MEM_R	32	0xA00	0xA1F	R/W
ECDSA_MEM_S	32	0xA20	0xA3F	R/W
ECDSA_MEM_Z	32	0xA40	0xA5F	R/W
ECDSA_MEM_Qx	32	0xA60	0xA7F	R/W
ECDSA_MEM_Qy	32	0xA80	0xA9F	R/W

* 相对于 ECDSA_DS 基地址的地址偏移在第 9 章*系统和存储器* 中的表 9.3-2 中提供。

33.8 寄存器列表

本小节的所有地址均为相对于 ECDSA_DS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

为了提高安全性，在不同工作状态下，ECDSA_DS 寄存器的读写权限不同。以下是每个工作状态的缩写及其与阶段的对应关系：

- PI：IDLE 状态，对应 IDLE 阶段。
- PL：LOAD 状态，对应 LOAD 阶段。
- PB：BUSY 状态，对应 PREP、PROC 和 POST 阶段。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

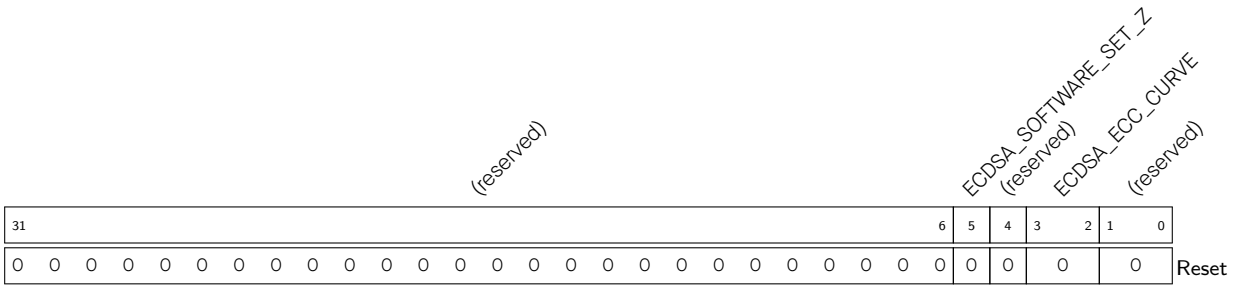
名称	描述	地址	访问权限			
			PI	PL	PG	PB
数据存储器	见表 33.7-1。					
配置寄存器						
ECDSA_CONF_REG	ECDSA_DS 配置寄存器	0x0004	R/W	N/A		
ECDSA_START_REG	ECDSA_DS 启动寄存器	0x001C	WT			N/A
时钟和复位寄存器						
ECDSA_CLK_REG	ECDSA_DS 时钟门控寄存器	0x0008	R/W	N/A		
中断寄存器						
ECDSA_INT_RAW_REG	ECDSA_DS 中断原始状态寄存器	0x000C	RO/WTC/SS			
ECDSA_INT_ST_REG	ECDSA_DS 中断状态寄存器	0x0010	RO			
ECDSA_INT_ENA_REG	ECDSA_DS 中断使能寄存器	0x0014	R/W			
ECDSA_INT_CLR_REG	ECDSA_DS 中断清除寄存器	0x0018	WT			
状态寄存器						
ECDSA_STATE_REG	ECDSA_DS 状态寄存器	0x0020	RO			
结果寄存器						
ECDSA_RESULT_REG	ECDSA_DS 结果寄存器	0x0024	RO/SS			N/A
SHA 寄存器						
ECDSA_SHA_MODE_REG	ECDSA_DS 控制 SHA 寄存器（哈希算法）	0x0200	N/A	R/W	N/A	
ECDSA_SHA_START_REG	ECDSA_DS 控制 SHA 寄存器（运算）	0x0210	N/A	WT	N/A	
ECDSA_SHA_CONTINUE_REG	ECDSA_DS 控制 SHA 寄存器（运算）	0x0214	N/A	WT	N/A	
ECDSA_SHA_BUSY_REG	ECDSA_DS 控制 SHA 状态寄存器	0x0218	N/A	RO	N/A	
版本寄存器						
ECDSA_DATE_REG	版本控制寄存器	0x00FC	R/W	N/A		

33.9 寄存器

本小节的所有地址均为相对于 ECDSA_DS 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 33.1. ECDSA_CONF_REG (0x0004)



ECDSA_ECC_CURVE 配置使用的椭圆曲线。

0：P-192

1：P-256

2：P-384

3：无效

(R/W)

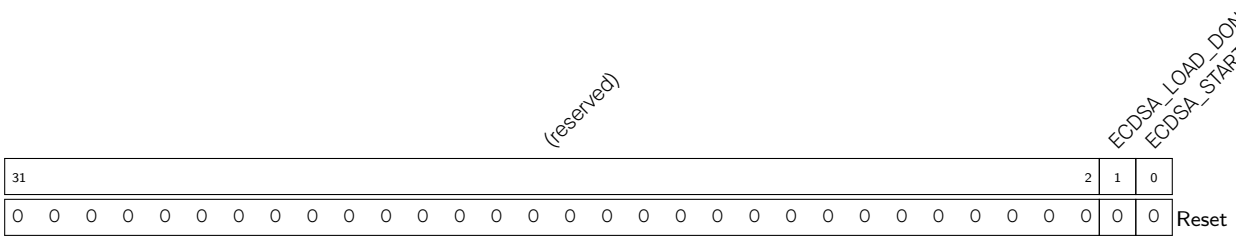
ECDSA_SOFTWARE_SET_Z 配置参数 z 的配置方式。

0：SHA 结果生成

1：软件写入

(R/W)

Register 33.2. ECDSA_START_REG (0x001C)



ECDSA_START 配置是否启动 ECDSA_DS 运算。此位配置后自动清 0。

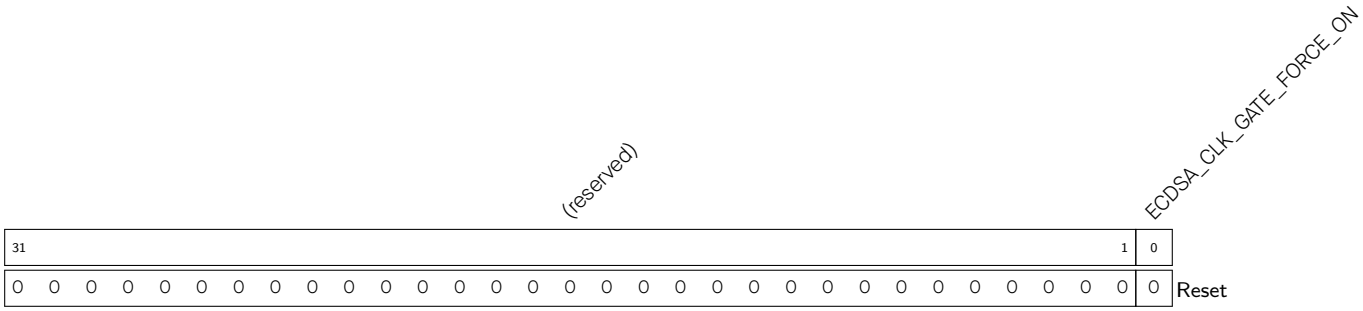
0：无效

1：启动 ECDSA_DS 运算

(WT)

ECDSA_LOAD_DONE 写 1 生成中断信号，表示 ECDSA_DS 的 LOAD 阶段已完成。此位配置后自动清 0。(WT)

Register 33.3. ECDSA_CLK_REG (0x0008)



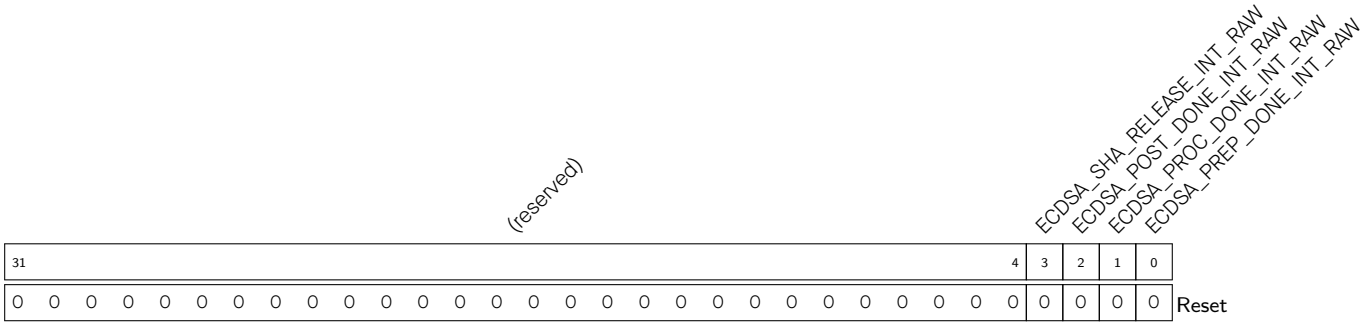
ECDSA_CLK_GATE_FORCE_ON 配置是否强制启动 ECC 内存的时钟门控。

0：无效

1：强制启动

(R/W)

Register 33.4. ECDSA_INT_RAW_REG (0x000C)



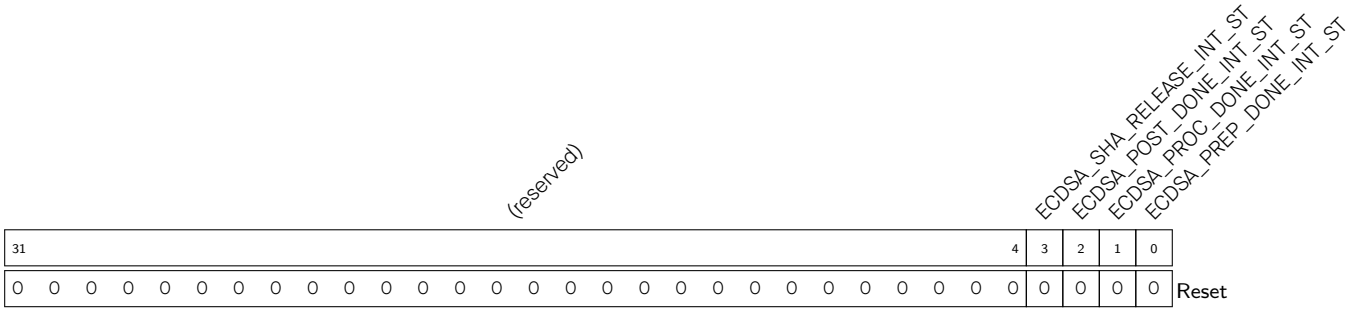
ECDSA_PREP_DONE_INT_RAW ECDSA_PREP_DONE_INT 的原始中断状态。(R/SS/WTC)

ECDSA_PROC_DONE_INT_RAW ECDSA_PROC_DONE_INT 的原始中断状态。(R/SS/WTC)

ECDSA_POST_DONE_INT_RAW ECDSA_POST_DONE_INT 的原始中断状态。(R/SS/WTC)

ECDSA_SHA_RELEASE_INT_RAW ECDSA_SHA_RELEASE_INT 的原始中断状态。(R/SS/WTC)

Register 33.5. ECDSA_INT_ST_REG (0x0010)



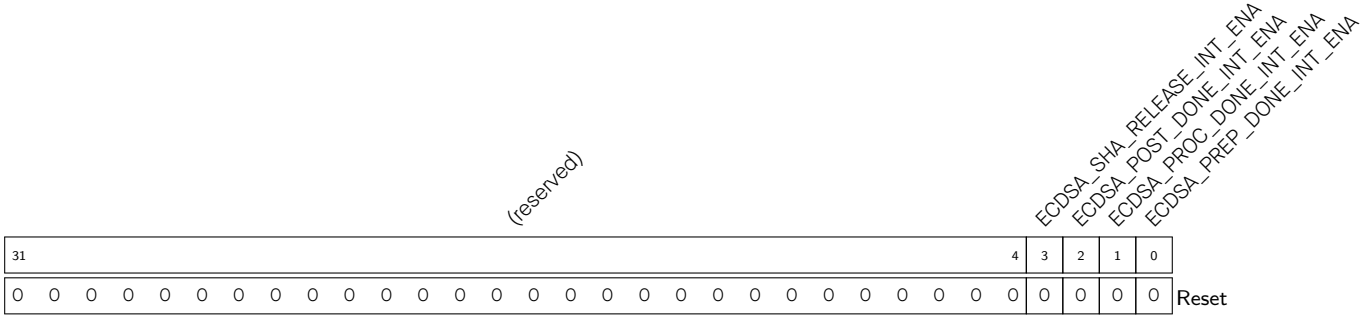
ECDSA_PREP_DONE_INT_ST ECDSA_PREP_DONE_INT 的屏蔽中断状态。(RO)

ECDSA_PROC_DONE_INT_ST ECDSA_PROC_DONE_INT 的屏蔽中断状态。(RO)

ECDSA_POST_DONE_INT_ST ECDSA_POST_DONE_INT 的屏蔽中断状态。(RO)

ECDSA_SHA_RELEASE_INT_ST ECDSA_SHA_RELEASE_INT 的屏蔽中断状态。(RO)

Register 33.6. ECDSA_INT_ENA_REG (0x0014)



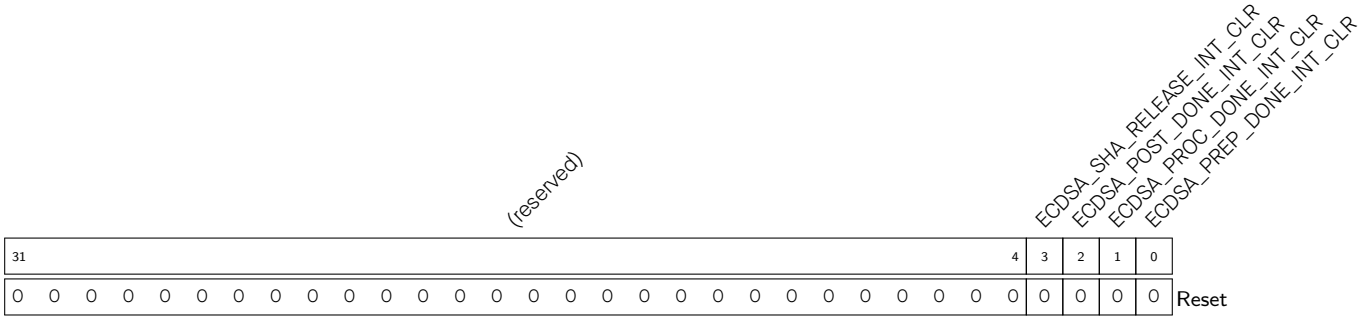
ECDSA_PREP_DONE_INT_ENA 写 1 使能 ECDSA_PREP_DONE_INT 中断。(R/W)

ECDSA_PROC_DONE_INT_ENA 写 1 使能 ECDSA_PROC_DONE_INT 中断。(R/W)

ECDSA_POST_DONE_INT_ENA 写 1 使能 ECDSA_POST_DONE_INT 中断。(R/W)

ECDSA_SHA_RELEASE_INT_ENA 写 1 使能 ECDSA_SHA_RELEASE_INT 中断。(R/W)

Register 33.7. ECDSA_INT_CLR_REG (0x0018)



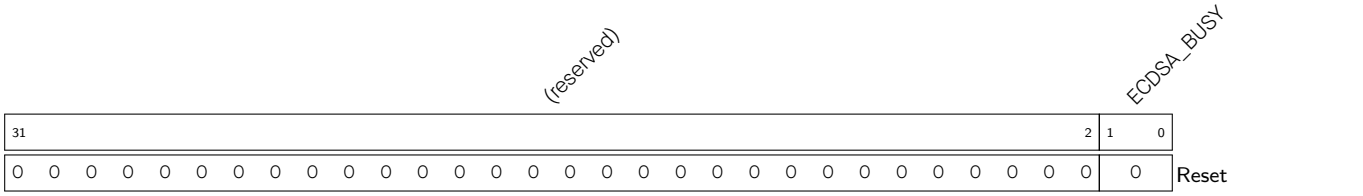
ECDSA_PREP_DONE_INT_CLR 写 1 清除 ECDSA_PREP_DONE_INT 中断。(WT)

ECDSA_PROC_DONE_INT_CLR 写 1 清除 ECDSA_PROC_DONE_INT 中断。(WT)

ECDSA_POST_DONE_INT_CLR 写 1 清除 ECDSA_POST_DONE_INT 中断。(WT)

ECDSA_SHA_RELEASE_INT_CLR 写 1 清除 ECDSA_SHA_RELEASE_INT 中断。(WT)

Register 33.8. ECDSA_STATE_REG (0x0020)



ECDSA_BUSY 表示 ECDSA_DS 模块的工作状态。

0：IDLE

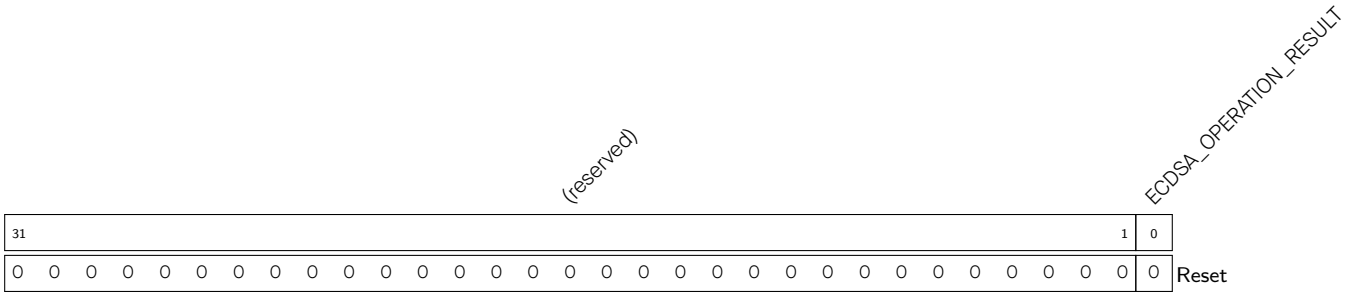
1：LOAD

2：保留

3：BUSY

(RO)

Register 33.9. ECDSA_RESULT_REG (0x0024)



ECDSA_OPERATION_RESULT 表示 ECDSA_DS 运算是否成功。

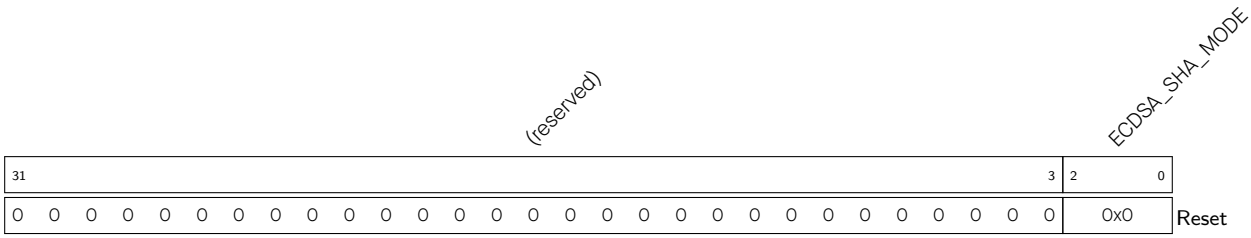
0：失败

1：成功

仅在 ECDSA_DS 运算完成后有效。

(RO/SS)

Register 33.10. ECDSA_SHA_MODE_REG (0x0200)



ECDSA_SHA_MODE 配置哈希运算的 SHA 算法。

1：SHA-224

2：SHA-256

3：SHA-384

4：SHA-512

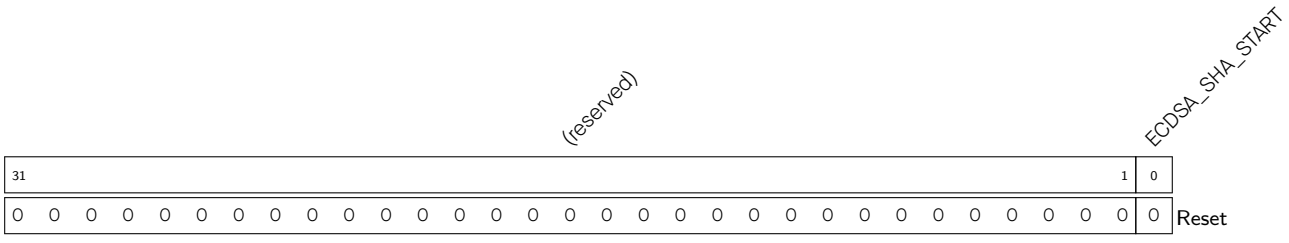
5：SHA-512-224

6：SHA-512-256

其他：无效

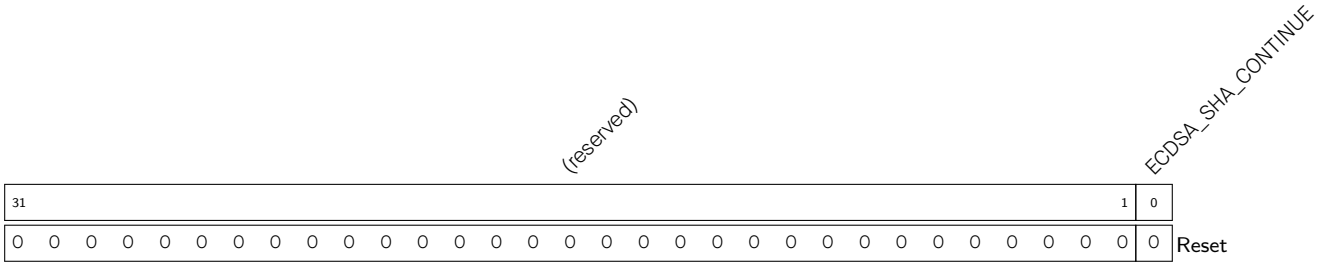
(R/W)

Register 33.11. ECDSA_SHA_START_REG (0x0210)



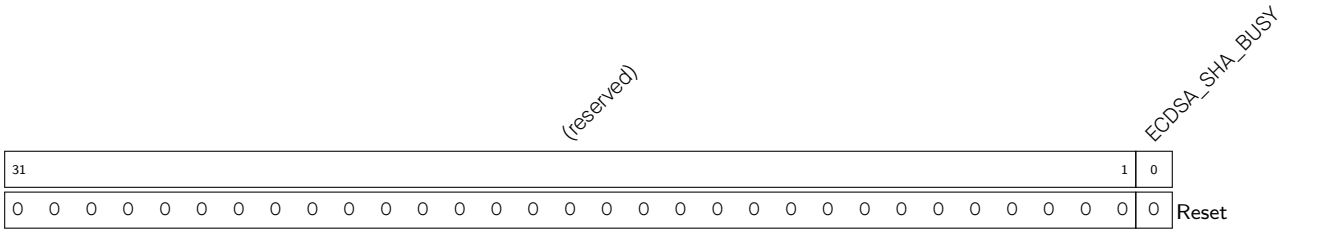
ECDSA_SHA_START 写 1 启动 ECDSA_DS 运算中的首个 SHA 运算。此位配置后自动清 0。(WT)

Register 33.12. ECDSA_SHA_CONTINUE_REG (0x0214)



ECDSA_SHA_CONTINUE 写 1 启动 ECDSA_DS 运算中的后续 SHA 运算。此位配置后自动清 0。(WT)

Register 33.13. ECDSA_SHA_BUSY_REG (0x0218)



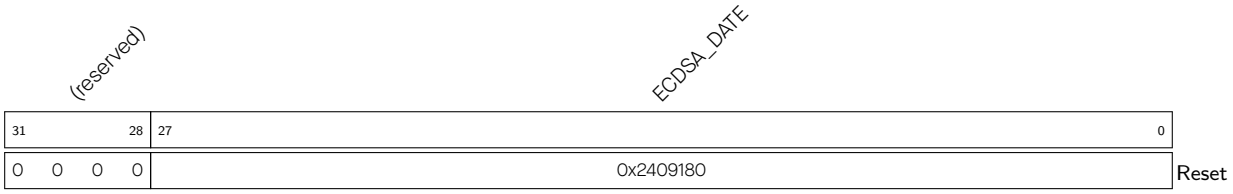
ECDSA_SHA_BUSY 表示 ECDSA_DS 运算过程中 SHA 加速器的工作状态。

0：IDLE

1：BUSY

(RO)

Register 33.14. ECDSA_DATE_REG (0x00FC)



ECDSA_DATE ECDSA_DS 版本控制寄存器。(R/W)

第 34 章

片外存储器加密与解密 (XTS_AES)

34.1 概述

ESP32-P4 芯片集成了片外存储器加密与解密模块，使用 IEEE Std 1619-2007 指定的 XTS-AES 标准算法，为用户存放在片外存储器（flash 和 RAM）的应用代码和数据提供了安全保障。用户可以将专有固件、敏感的用户数据（如用来访问私有网络的证书）存放在片外 flash 中，或将一般数据存放在片外 RAM 中。

34.2 主要特性

- 使用通用 XTS-AES 算法，符合 IEEE Std 1619-2007
- 支持手动加密，需要软件参与
- 支持高速自动加解密，无需软件参与
- 由寄存器配置、eFuse 参数、启动 (boot) 模式共同决定开启/关闭加解密功能
- 支持可配置的抗 DPA 攻击功能

34.3 模块结构

片外存储器加解密模组包含三个模块：手动加密 (Manual Encryption) 模块、自动加密 (Auto Encryption) 模块和自动解密 (Auto Decryption) 模块。模块结构如图 34.3-1 所示。

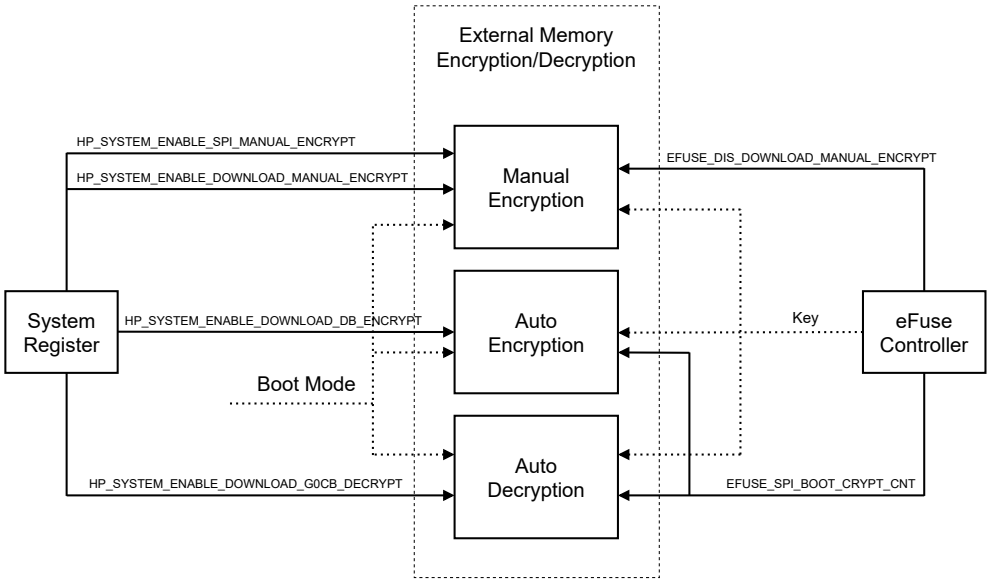


图 34.3-1. 片外存储器加解密结构

手动加密模块能够对指令和数据进行加密，加密后的指令/数据随即以密文状态由 SPI1 写入片外 flash。

当 CPU 通过 cache 向片外 RAM 写入数据时，自动加密模块会首先自动加密数据，然后将数据作为密文写入片外 RAM。

当 CPU 通过高速 cache 从片外 flash 或片外 RAM 读取数据时，自动解密模块将自动解密密文，从而检索指令和数据。

系统寄存器外设中（请参见章节 21 *系统寄存器 (SYSREG)*），寄存器 HP_SYSTEM_CRYPTO_CTRL_REG 内与片外存储器加解密相关的有以下 4 位：

- HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- HP_SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT
- HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

此外，片外存储器加解密模组还会从外设 eFuse 控制器中获取两个参数：EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 和 EFUSE_SPI_BOOT_CRYPT_CNT。更多详细信息，请参考章节 9.4 *eFuse 控制器 (EFUSE)*。

34.4 功能描述

34.4.1 XTS 算法

手动加密模块、自动加密模块和自动解密模块均使用 XTS 算法。具体实现中，XTS 算法使用 1024 位为一个数据单元 (data unit)，此处的“数据单元”由 *XTS-AES Tweakable Block Cipher* 标准中的章节 *XTS-AES encryption procedure* 定义。更多关于 XTS-AES 算法的信息，请参考 IEEE Std 1619-2007。

34.4.2 密钥

执行 XTS 算法时，手动加密模块、自动加密模块和自动解密模块共用同一密钥 Key。该密钥由硬件 eFuse 或密钥管理器提供，软件无法访问该状态。

Key 的长度为 256 位或 512 位。Key 的值和长度由 Block4 ~ Block9 中一个 eFuse 块的参数信息以及相应的 eFuse 参数决定。为方便阐述如何通过 eFuse 参数信息推导出 Key 的值，现定义：

- $Block_A$：Block4 ~ Block9 中密钥用途为 EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_1 的块（请参考表 9.4-2 密钥用途数值对应的含义）。如果 $Block_A$ 存在，那么 $Block_A$ 中存放着 256 位的 Key_A。
- $Block_B$：Block4 ~ Block9 中密钥用途为 EFUSE_KEY_PURPOSE_XTS_AES_256_KEY_2 的块（请参考表 9.4-2 密钥用途数值对应的含义）。如果 $Block_B$ 存在，那么 $Block_B$ 中存放着 256 位的 Key_B。
- $Block_C$：Block4 ~ Block9 中密钥用途为 EFUSE_KEY_PURPOSE_XTS_AES_128_KEY 的块（请参考表 9.4-2 密钥用途数值对应的含义）。如果 $Block_C$ 存在，那么 $Block_C$ 中存放着 256 位的 Key_C。

表 34.4-1. 根据 Key_A、Key_B 及 Key_C 生成的 Key 值

$Block_A$	$Block_B$	$Block_C$	Key	Key 长度（位）
存在	存在	无关项	$Key_A \| \| Key_B$	512
存在	不存在	无关项	$Key_A \| \| 0^{256}$	512
不存在	存在	无关项	$0^{256} \| \| Key_B$	512
不存在	不存在	存在	Key_C	256
不存在	不存在	不存在	0^{256}	256

说明：

- “存在”表示存在该 Block
- “不存在”表示不存在该 Block
- “0^{256}”表示由 256 个位 0 组成的位串
- “||”表示键合运算符，用于将一个位串连接到另一个位串

更多有关密钥用途的信息，请参考章节 9.4 *eFuse 控制器 (EFUSE)* 中的表 9.4-2 密钥用途数值对应的含义。

34.4.3 目标空间

目标空间指的是片外存储器中存放首次加密密文的一段连续地址空间。目标空间可由目标类型、目标大小和目标基地址唯一确定。这三个参数的定义如下：

- 目标类型：目标空间的类型 ($type$)，可为片外 flash 或片外 RAM。值为 0 时代表片外 flash，值为 1 则代表片外 RAM。
- 目标大小：目标空间的大小 ($size$)，即单次加密的数据量。以字节为单位，支持 16 字节、32 字节或 64 字节。
- 目标基地址：目标空间的基地址 ($base_addr$)。目标基地址为 30 位的物理地址，取值范围为 0x0000_0000 ~ 0x3FFF_FFFF，但要求以 $size$ 为单位对齐，即 $base_addr \% size == 0$。

例如，在一次加密操作中，需要将 16 字节的指令数据加密后写入片外 flash 中的地址段 0x130 ~ 0x13F，则目标空间为 0x130 ~ 0x13F，目标类型为 0（片外 flash），目标大小为 16（字节），目标基地址为 0x130。

对于任意长度（必须是 16 字节的整数倍）的明文指令或数据的加密，整个加密过程可拆分成多次进行，每次加密均具备相应的目标空间和相应参数。

对于自动加密和自动解密模块，上述参数由硬件自动定义。对于手动加密模块，上述参数需由用户主动配置。

说明：

IEEE Std 1619-2007 中的章节 *5.1 Data units and tweaks* 中定义的“tweak”是一个 128-bit 的非负整数（$tweak$），其值可以通过公式求出：$tweak$ = $type * 2^{30}$ + ($base_addr$ & 0x3FFFFF80)。$tweak$ 中低 7 位和高 97 位恒为零。

34.4.4 数据写入

对于自动加密和自动解密模块，数据写入由硬件自动完成。对于手动加密模块，数据写入需由用户主动配置。手动加密模块中包含一个由 16 个寄存器组成的寄存器块 XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 15)，专用于数据写入，一次可以存放最多 512 位明文指令/数据。

实际上，在手动加密模块中，相对于明文的来源，密文将要存放的位置更为重要。基于明文和密文之间严格的对应关系，为更好地描述明文在寄存器块中的存放方式，现假设明文从一开始就存放在目标空间中，在加密完成后替换为密文。因此，本节中将不再出现“明文”的概念，而用“目标空间”来代替。

目标空间映射到寄存器块的方法：

假设目标空间中某个字的存放地址为 $address$，记 $offset$ = $address$%64，n = $offset$/4，那么该字将被存放在寄存器 XTS_AES_PLAIN_n_REG 中。

例如，当目标大小为 64 时，寄存器块中的所有寄存器都将被使用，目标空间中的地址与寄存器块之间的映射关系如表 34.4-2 所示。

表 34.4-2. 目标空间与寄存器堆的映射关系

offset	寄存器	*offset*	寄存器
0x00	XTS_AES_PLAIN_0_REG	0x20	XTS_AES_PLAIN_8_REG
0x04	XTS_AES_PLAIN_1_REG	0x24	XTS_AES_PLAIN_9_REG
0x08	XTS_AES_PLAIN_2_REG	0x28	XTS_AES_PLAIN_10_REG
0x0C	XTS_AES_PLAIN_3_REG	0x2C	XTS_AES_PLAIN_11_REG
0x10	XTS_AES_PLAIN_4_REG	0x30	XTS_AES_PLAIN_12_REG
0x14	XTS_AES_PLAIN_5_REG	0x34	XTS_AES_PLAIN_13_REG
0x18	XTS_AES_PLAIN_6_REG	0x38	XTS_AES_PLAIN_14_REG
0x1C	XTS_AES_PLAIN_7_REG	0x3C	XTS_AES_PLAIN_15_REG

34.4.5 手动加密模块

手动加密模块是一个外设模块，自身带有寄存器，可以被 CPU 直接访问。模块内的寄存器、系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。需注意，手动加密模块仅支持加密存放在片外 flash 中的内容。

当且仅当满足下列对应条件时，手动加密模块才允许手动加密：

- SPI Boot 模式下：

  当寄存器 HP_SYSTEM_CRYPTO_CTRL_REG 的

HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 位为 1 时，手动加密模块拥有工作权限，否则无法工作。

- Joint Download Boot 模式下：

  当寄存器 HP_SYSTEM_CRYPTO_CTRL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 位为 1，且 eFuse 参数 EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 为 0 时，手动加密模块拥有工作权限，否则无法工作。

说明：

- 即使 CPU 可以越过 cache，直接读取片外存储器从而得到加密指令/数据，但用户依旧无法获取密钥 Key。

34.4.6 自动加密模块

自动加密模块并非常规外设模块，不带有寄存器，也无法被 CPU 直接访问。由系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，自动加密模块才允许自动加密：

- SPI Boot 模式下：

  当参数 EFUSE_SPI_BOOT_CRYPT_CNT（共 3 位）为 1，2，4 或 7 时（即二进制表示有奇数个 1 时），自动加密模块拥有工作权限，否则无法工作。

- Joint Download Boot 模式下：

  当寄存器 HP_SYSTEM_CRYPTO_CTRL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT 位为 1 时，自动加密模块拥有工作权限，否则无法工作。

说明：

- 当自动加密模块拥有工作权限时，如果 CPU 向片外 RAM 写入数据，自动加密模块将自动加密该数据，随后将密文写入片外 RAM。加密的整个过程无需软件参与，且对 cache 透明。加密算法过程中，密钥 Key 无法被用户获取。
- 当自动加密模块没有工作权限时，将忽略 CPU 对 cache 的访问请求，不处理数据。此时，数据将以明文形式直接写入片外 RAM。

34.4.7 自动解密模块

自动解密并非传统外设模块，自身不带寄存器，不能被 CPU 直接访问。系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，自动解密模块才允许自动解密：

- SPI Boot 模式下

  当参数 EFUSE_SPI_BOOT_CRYPT_CNT（共 3 位）为 1，2，4 或 7 时（即二进制表示有奇数个 1 时），自动解密模块拥有工作权限，否则无法工作。

- Joint Download Boot 模式下

  当寄存器 HP_SYSTEM_CRYPTO_CTRL_REG 的

HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT 位为 1 时，自动解密模块拥有工作权限，否则无法工作。

说明：

1. 当自动解密模块拥有工作权限时，如果 CPU 通过 cache 读取片外存储器中的指令/数据，自动解密将自动对读取到的密文进行解密以恢复指令/数据。解密的整个过程无需软件参与，且对 cache 透明。解密算法过程中，密钥 Key 无法被软件获取。
2. 当自动解密模块没有工作权限时，无论数据加密与否，自动解密模块不对片外存储器中的数据产生作用。此时，CPU 通过 cache 读取到的是片外存储器中的原始内容。

34.5 软件流程

手动加密模块工作时需要软件参与，软件流程为：

1. 配置 XTS_AES：
   - 将寄存器 XTS_AES_DESTINATION_REG 类型 $type$ 的值设置为 0。
   - 将寄存器 XTS_AES_PHYSICAL_ADDRESS_REG 的值设置为 $base_addr$。
   - 将寄存器 XTS_AES_LINESIZE_REG 的值设置为 $\frac{size}{32}$。

   关于 $base_addr$ 和 $size$ 的定义，请参考章节 34.4.3。
2. 将明文指令/数据写入至寄存器块 XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 15)。更多详细信息，请参考章节 34.4.4。
   请根据实际需求写入寄存器，未使用的寄存器可为任意值。
3. 等待手动加密模块进入空闲状态。轮询寄存器 XTS_AES_STATE_REG 直到软件读取到 0，表明手动加密模块已进入空闲状态。
4. 向寄存器 XTS_AES_TRIGGER_REG 写入 1，启动手动加密。
5. 等待加密完成。轮询寄存器 XTS_AES_STATE_REG，直到软件读取到 2。
   上述步骤为使用 Key 操作手动加密模块对明文指令/数据进行加密的过程。
6. 向寄存器 XTS_AES_RELEASE_REG 写入 1，使 SPI1 获得密文的访问权限。该操作完成后，寄存器 XTS_AES_STATE_REG 的值将为 3。
7. 调用 SPI1，将密文写入片外 flash（请参阅 ESP-IDF 编程指南 中的 Flash 加密 API 参考）。
8. 向寄存器 XTS_AES_DESTROY_REG 写入 1，销毁密文。该操作完成后，寄存器 XTS_AES_STATE_REG 的值将为 0。

请根据所需加密的明文指令/数据的数量，重复上述步骤。

34.6 抗 DPA 攻击

DPA（Differential Power Analysis, 差分能量分析）是密码学中的一种旁路攻击手段，攻击者可以统计分析从多个加密操作收集的数据，从而计算加密计算中的中间值。ESP32-P4 XTS_AES 模块支持抗 DPA 攻击功能，以抵御来自外界的 DPA 攻击。根据 IEEE Std 1619-2007，XTS-AES 算法可分为两步：

- 第一步：计算 T 的值。下文中，将此步骤称为“计算 T”。
- 第二步：计算密文/明文。下文中，将此步骤称为“计算 D”。

用户可以通过寄存器配置实现不同安全等级：

- 首先，为更好地描述各步骤，定义以下参数：
  - $select_reg$ = XTS_AES_CRYPT_DPA_SELECT_REGISTER
  - $reg_d_dpa_en$ = XTS_AES_CRYPT_CALC_D_DPA_EN
  - $efuse_dpa_en$ = EFUSE_CRYPT_DPA_ENABLE
  - $reg_anti_dpa_level$ = XTS_AES_CRYPT_SECURITY_LEVEL
  - $efuse_anti_dpa_level$ = 3
- 配置 XTS_AES 模块抗 DPA 攻击的安全等级：

  $$Anti_DPA_level = select_reg\ ?\ (reg_anti_dpa_level) : (efuse_dpa_en * efuse_anti_dpa_level)$$

  当 $Anti_DPA_level$ 为 0 时，关闭抗 DPA 攻击功能。$Anti_DPA_level$ 的值越大，抗 DPA 攻击的能力就越强。
- 配置是否在 XTS-AES 算法计算 D 时开启抗 DPA 功能：

  $$Anti_DPA_enabled_in_calc_D = select_reg\ ?\ reg_d_dpa_en : efuse_dpa_en$$

  在 $Anti_DPA_level$ 不为 0 的情况下，当 $Anti_DPA_enabled_in_calc_D$ 等于 1 时，XTS-AES 算法在计算 D 时开启抗 DPA 功能。
  在 $Anti_DPA_level$ 不为 0 的情况下，XTS-AES 算法计算 T 时总是开启抗 DPA 功能。

说明：

- 在 efuse_dpa_en 置 1 后，用户仍可通过配置 $select_reg$ = 1 及 $reg_anti_dpa_level$ = 0 关闭抗 DPA 功能。
- 配置是否开启抗 DPA 功能会对访问片外存储器的带宽造成影响：
  - 在计算 D 过程中配置开启抗 DPA 功能时，当抗攻击等级 >=4，读写带宽将降低超过 50%。
  - 在计算 D 过程中配置关闭抗 DPA 功能时，当抗攻击等级 >=6，读写带宽将降低超过 50%。

34.7 寄存器列表

本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

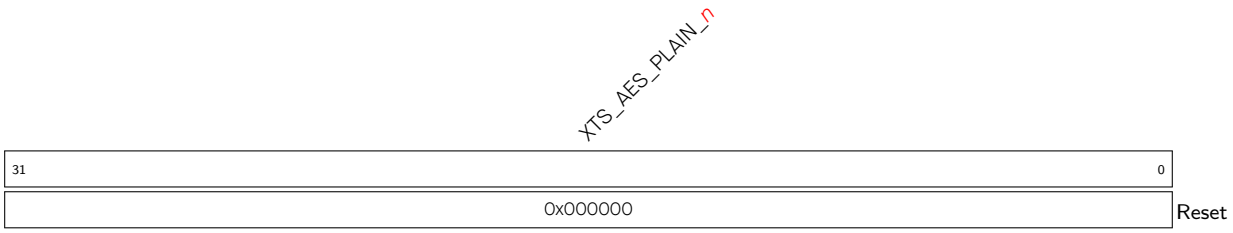
名称	描述	地址	访问
明文寄存器堆			
XTS_AES_PLAIN_0_REG	明文寄存器 0	0x0300	R/W
XTS_AES_PLAIN_1_REG	明文寄存器 1	0x0304	R/W
XTS_AES_PLAIN_2_REG	明文寄存器 2	0x0308	R/W
XTS_AES_PLAIN_3_REG	明文寄存器 3	0x030C	R/W
XTS_AES_PLAIN_4_REG	明文寄存器 4	0x0310	R/W
XTS_AES_PLAIN_5_REG	明文寄存器 5	0x0314	R/W
XTS_AES_PLAIN_6_REG	明文寄存器 6	0x0318	R/W
XTS_AES_PLAIN_7_REG	明文寄存器 7	0x031C	R/W
XTS_AES_PLAIN_8_REG	明文寄存器 8	0x0320	R/W
XTS_AES_PLAIN_9_REG	明文寄存器 9	0x0324	R/W
XTS_AES_PLAIN_10_REG	明文寄存器 10	0x0328	R/W
XTS_AES_PLAIN_11_REG	明文寄存器 11	0x032C	R/W
XTS_AES_PLAIN_12_REG	明文寄存器 12	0x0330	R/W
XTS_AES_PLAIN_13_REG	明文寄存器 13	0x0334	R/W
XTS_AES_PLAIN_14_REG	明文寄存器 14	0x0338	R/W
XTS_AES_PLAIN_15_REG	明文寄存器 15	0x033C	R/W
配置寄存器			
XTS_AES_LINESIZE_REG	配置目标空间的大小	0x0340	R/W
XTS_AES_DESTINATION_REG	配置片外存储器的类型	0x0344	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	配置物理地址	0x0348	R/W
XTS_AES_DPA_CTRL_REG	配置抗 DPA 功能	0x0388	R/W
控制/状态寄存器			
XTS_AES_TRIGGER_REG	启动 AES 算法	0x034C	WO
XTS_AES_RELEASE_REG	释放控制寄存器	0x0350	WO
XTS_AES_DESTROY_REG	销毁控制寄存器	0x0354	WO
XTS_AES_STATE_REG	状态寄存器	0x0358	RO
版本寄存器			
XTS_AES_DATE_REG	版本控制寄存器	0x035C	R/W

34.8 寄存器

本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

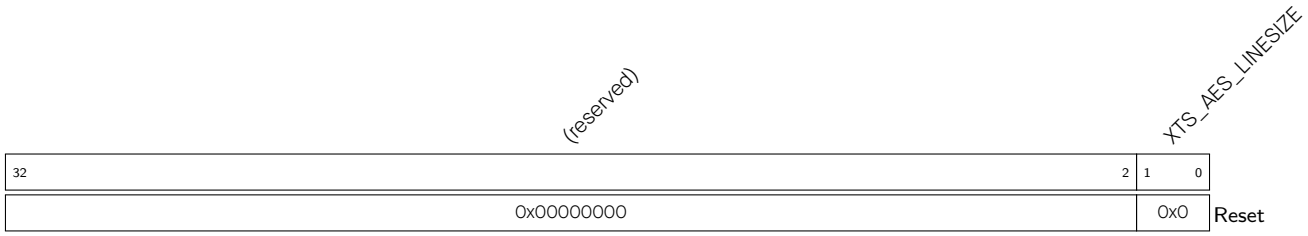
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 34.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0300+4**n*)



XTS_AES_PLAIN_*n* 配置明文的第 *n* 个 32 位部分。(R/W)

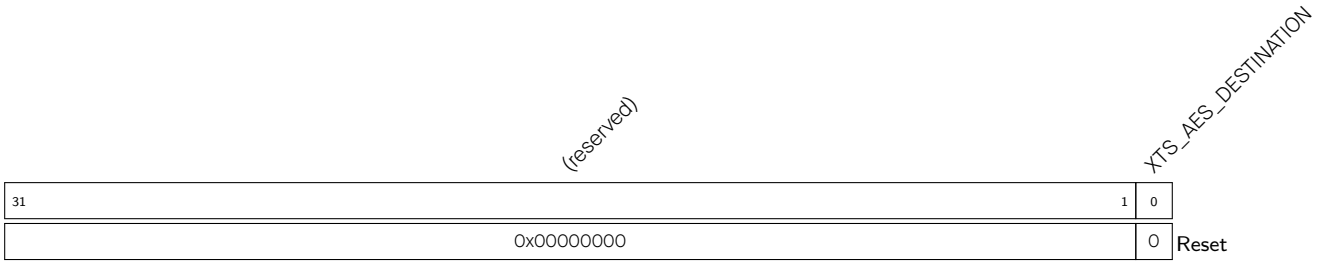
Register 34.2. XTS_AES_LINESIZE_REG (0x0340)



XTS_AES_LINESIZE 配置单次加密的数据大小。

0：加密 16 字节
1：加密 32 字节
2：加密 64 字节
3：无效
(R/W)

Register 34.3. XTS_AES_DESTINATION_REG (0x0344)



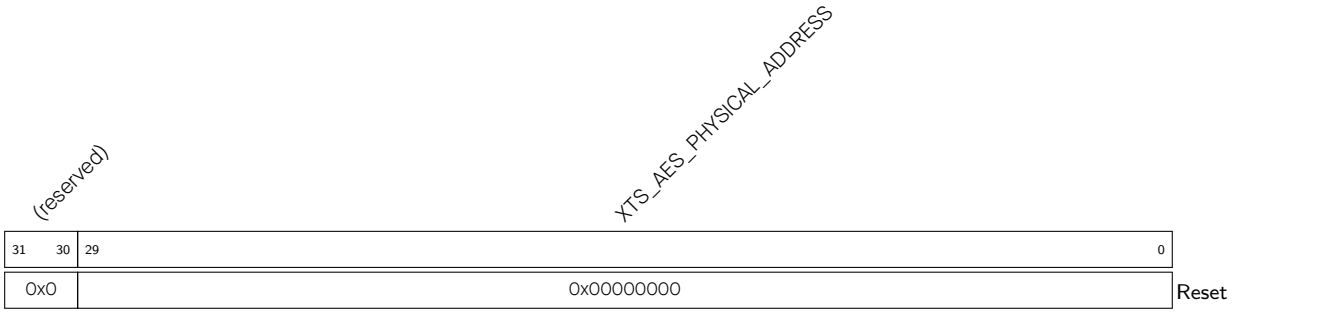
XTS_AES_DESTINATION 配置手动加密的存储器类型，目前只能手动加密 flash，所以只能为 0。用户不能写入 1，否则将发生错误。

0：加密 flash

1：加密片外 RAM

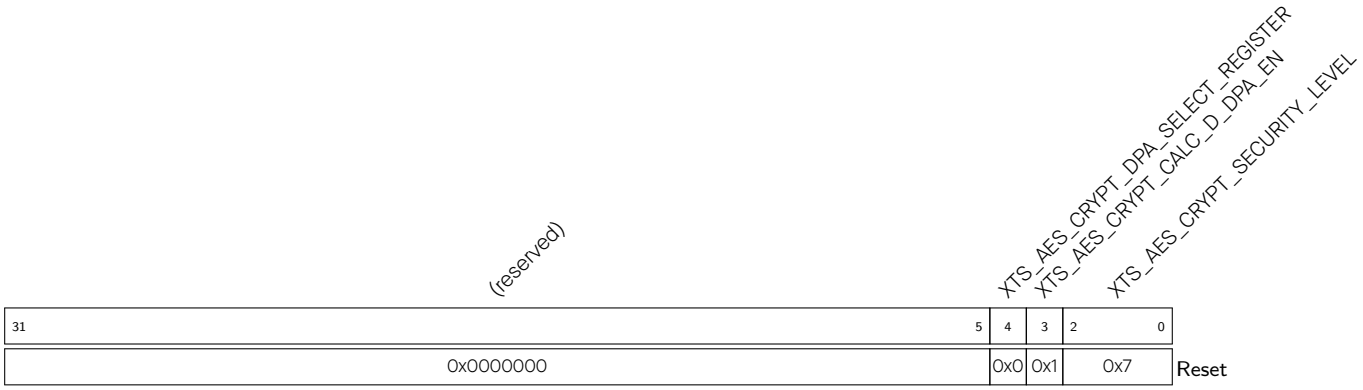
(R/W)

Register 34.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0348)



XTS_AES_PHYSICAL_ADDRESS 配置物理地址。请注意，该值范围必须为 0x0000_0000 ~ 0x00FF_FFFF。(R/W)

Register 34.5. XTS_AES_DPA_CTRL_REG (0x0388)



XTS_AES_CRYPT_DPA_SELECT_REGISTER 配置由 eFuse 或寄存器控制抗 DPA 功能。

0：由寄存器控制抗 DPA 功能

1：由 eFuse 控制抗 DPA 功能

(R/W)

XTS_AES_CRYPT_CALC_D_DPA_EN 配置是否在 XTS_AES 算法中启用抗 DPA 功能。

0：仅在计算 T 时启用抗 DPA 功能

1：计算 T 或是计算 D 时都启用抗 DPA 功能

请注意，此字段仅在 XTS_AES_CRYPT_SECURITY_LEVEL 不为 0 时有效。

(R/W)

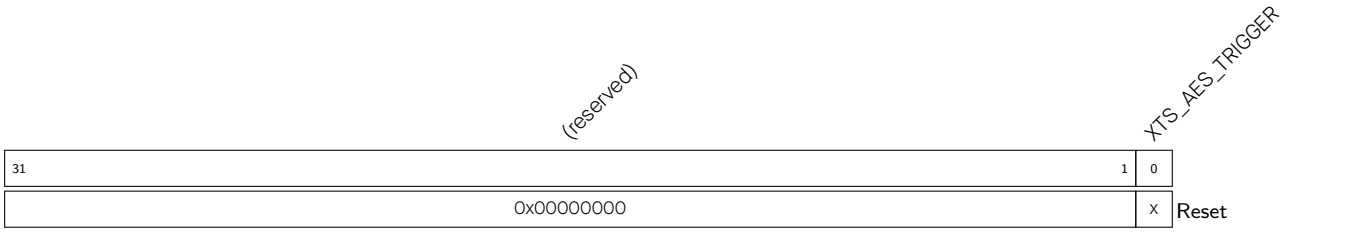
XTS_AES_CRYPT_SECURITY_LEVEL 配置外部加解密的安全等级。

0：抗 DPA 功能已关闭

1-7：数字越大，则加解密安全性更高

(R/W)

Register 34.6. XTS_AES_TRIGGER_REG (0x034C)



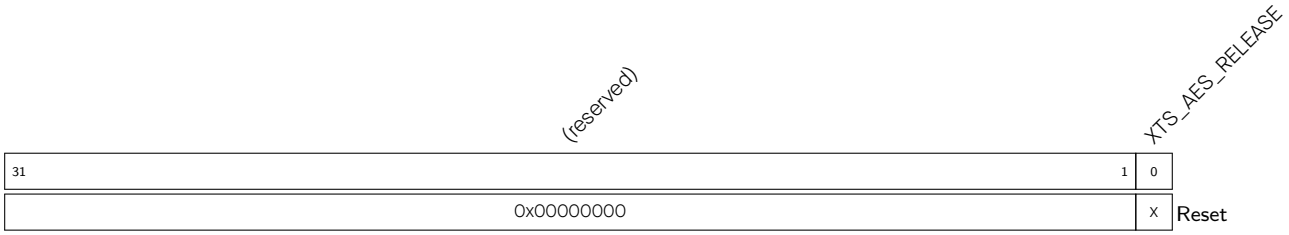
XTS_AES_TRIGGER 配置是否使能手动加密运算。

0：关闭手动加密运算

1：使能手动加密运算

(WO)

Register 34.7. XTS_AES_RELEASE_REG (0x0350)



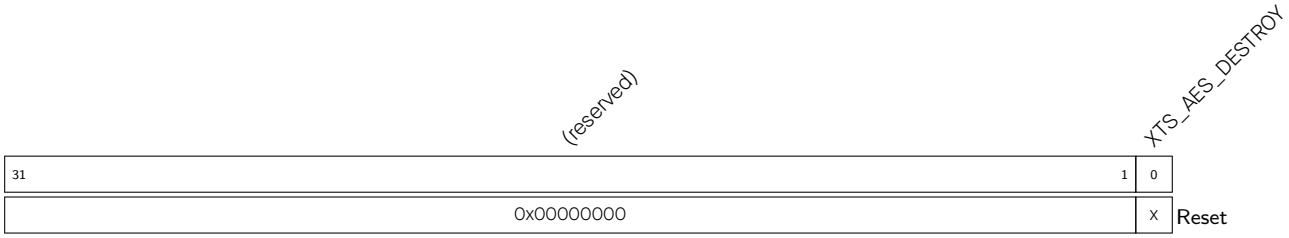
XTS_AES_RELEASE 配置是否使 SPI1 获取密文访问权限。

0：无效

1：使 SPI1 获取密文访问权限

(WO)

Register 34.8. XTS_AES_DESTROY_REG (0x0354)



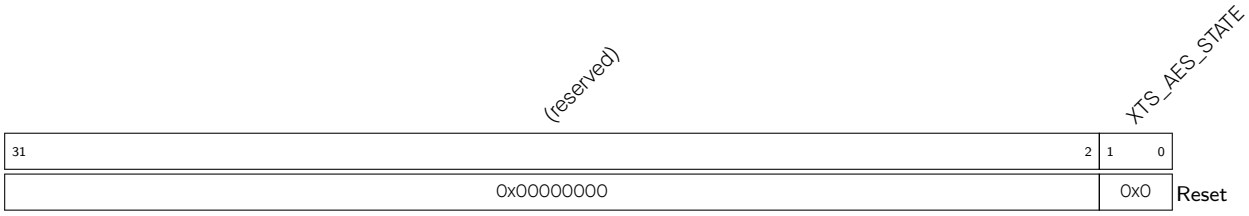
XTS_AES_DESTROY 配置是否销毁加密结果。

0：无效

1：销毁加密结果

(WO)

Register 34.9. XTS_AES_STATE_REG (0x0358)



XTS_AES_STATE 表示手动加密模块状态。

0 (XTS_AES_IDLE)：空闲

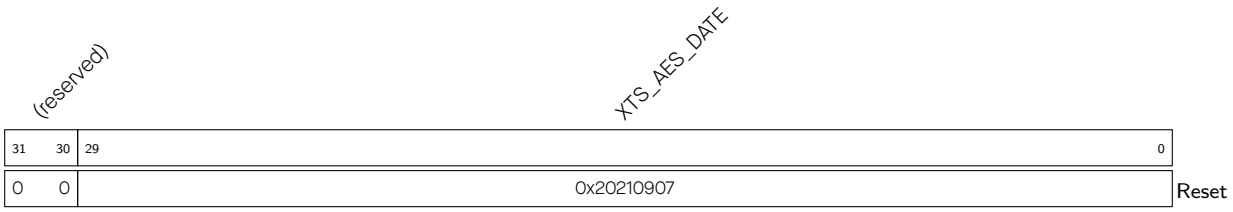
1 (XTS_AES_BUSY)：计算中

2 (XTS_AES_DONE)：计算完成，但手动加密结果数据对 SPI 不可见

3 (XTS_AES_RELEASE)：手动加密结果对 SPI 可见

(RO)

Register 34.10. XTS_AES_DATE_REG (0x035C)



XTS_AES_DATE 版本控制寄存器。(R/W)

第 35 章

随机数发生器 (RNG)

35.1 概述

ESP32-P4 内置一个真随机数发生器，其生成的 32 位随机数可作为加密等操作的基础。

35.2 主要特性

ESP32-P4 的随机数发生器可通过物理过程而非算法生成真随机数，所有生成的随机数在特定范围内出现的概率完全一样。

35.3 功能描述

系统可以从随机数发生器的寄存器 LPSYSREG_RNG_DATA_REG 中读取随机数，每个读到的 32 位随机数都是真随机数，噪声源为系统中的**热噪声**、**异步时钟**和**环形振荡器**。

- **热噪声**可以来自 SAR ADC。当芯片的 SAR ADC 工作时，就会产生比特流，并通过异或 (XOR) 逻辑运算作为随机数种子进入随机数生成器。
- RC_FAST_CLK 是一种**异步时钟源**，会产生电路亚稳态。这种亚稳态也可以作为随机数种子，进入随机数生成器。关于 RC_FAST_CLK，详见以下 说明。
- BUF_CELL (缓存单元) 实现的**环形振荡器** (BUF_CHAIN) 生成的计数值作为随机数种子进入随机数生成器。

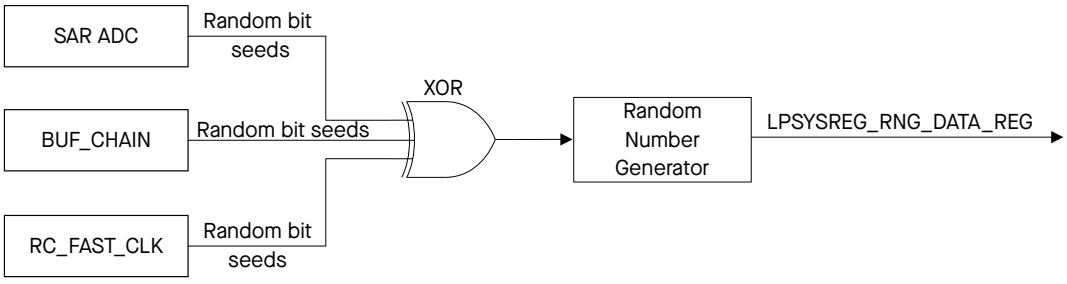


图 35.3-1. 噪声源

当 SAR ADC 打开时，每个 RC_FAST_CLK 时钟周期内（来自内部 RC 振荡器，详见 11 复位和时钟 章节），随机数发生器将获得 1 位的熵。因此，为了获得最大的熵值，建议读取 LPSYSREG_RNG_DATA_REG 寄存器时的速

率不超过 1 MHz。

35.4 编程指南

在使用 ESP32-P4 的随机数生成器时，应该至少保证 SAR ADC 或 RC_FAST_CLK 处于使能状态，否则可能会导致产生伪随机数，应注意避免。其中，

- 关于 SAR ADC，详见 65 *ADC 控制器 (ADC)* 章节。
- RC_FAST_CLK 可通过设置 PMU_HP_SLEEP_LP_CK_POWER_REG 寄存器中的 PMU_HP_SLEEP_XPD_FOSC_CLK 位使能。关于 RC_FAST_CLK，详见以下 说明。
- 通过设置 LPPERI_CLK_EN_REG 寄存器中的 LPPERI_CK_EN_RNG 位，对 RC_FAST_CLK 进行门控。
- BUF_CHAIN 可通过设置 RNG_CFG_REG 寄存器中的 RNG_SAMPLE_ENABLE 位使能。

说明：

RC_FAST_CLK 时钟仅可以提高随机数发生器的熵值。然而，为了保证随机数发生器可以获得足够大的熵值，仍建议在使用随机数发生器时至少保证 SAR ADC 处于工作状态。

在使用随机数生成器时，请多次读取 LPSYSREG_RNG_DATA_REG 寄存器的值，直至获得足够多的随机数。在读取寄存器时，注意控制速率不要超过上方第 35.3 小节的介绍。

35.5 寄存器列表

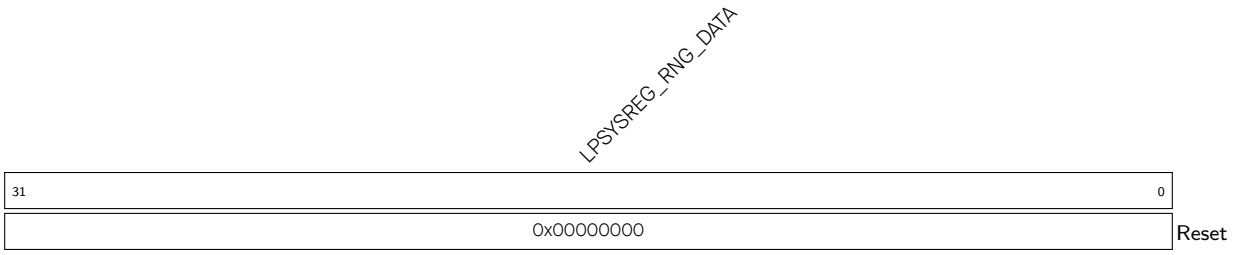
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
LPSYSREG_RNG_DATA_REG	随机数源数据	0x5011_01A4	RO
LPPERI_CLK_EN_REG	RC_FAST_CLK 的门控信号	0x5012_0000	R/W
RNG_CFG_REG	RNG 配置寄存器	0x5012_6000	R/W

35.6 寄存器

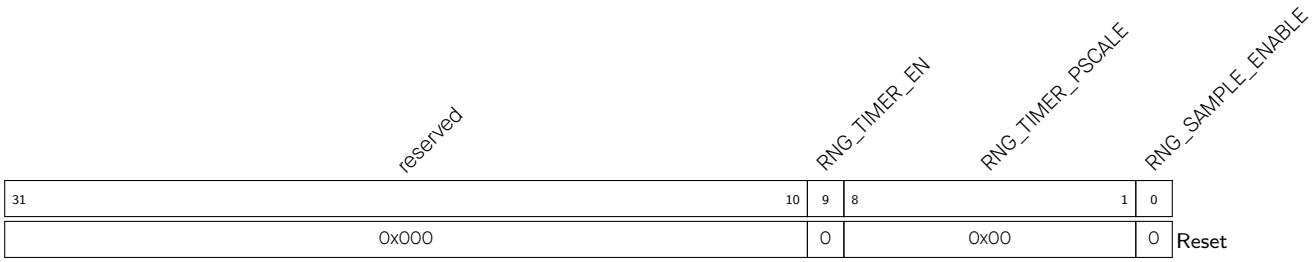
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 35.1. LPSYSREG_RNG_DATA_REG (0x5011_01A4)



LPSYSREG_RNG_DATA 随机数源数据。(RO)

Register 35.2. RNG_CFG_REG (0x5012_6000)



RNG_SAMPLE_ENABLE 配置是否使能环形振荡器噪声源。

1：使能

0：关闭

(R/W)

RNG_TIMER_PSCALE RNG 内置定时器的分频系数。(R/W)

RNG_RTC_TIMER_EN 配置是否使能 RNG 内置定时器。

1：使能并在生成随机数时使用该定时器的计数值

0：关闭定时器

(R/W)

第 36 章

密钥管理器

36.1 概述

密钥管理是系统的安全核心，利用密钥管理器和硬件唯一密钥 (HUK) 生成器，实现了密钥的安全存储和部署。使用芯片的 HUK 作为信任根，芯片的任何区域均不存储明文密钥，因此提高了芯片的整体安全性。

与使用 eFuse 存储密钥相比，密钥管理器和 HUK 生成器可以对系统进行以下改进：

- 环境适应性：支持密钥的安全部署，即使外部环境不可信任，依然可以安全地完成密钥部署。
- 抗攻击能力：每个芯片的 SRAM 都有一个独一无二的默认值，软件无法访问该默认值。明文密钥不存储在芯片的任何物理位置，因此可以抵御切片攻击和复制攻击。例如直接通过切片查看 eFuse 中密钥的方法将会失效。
- 密钥管理灵活性：密钥管理器允许用户将密钥信息（非明文）存储在外部存储器中。与 eFuse 不同，密钥管理器允许用户创建无限数量的密钥，进行动态密钥更改、密钥替换等。

36.2 密钥管理背景知识

SRAM PUF（物理不可克隆函数）利用 SRAM 单元上电时的天然初始状态，将这些初始状态用作芯片的“指纹”。芯片上电时，每个未初始化的 SRAM 位会因为晶体管在制造过程中的微小差异而倾向于变成 0 或 1。这些差异对每颗芯片都是独一无二的，因此生成的位模式也具有设备唯一性。

利用这一特性可以在芯片上生成一个硬件唯一密钥（HUK），且无需在芯片上存储这个密钥：芯片可以在每次启动时依靠物理特性重新生成密钥，而不需要从 flash 或 eFuse 中读取。HUK 随后可用作安全启动、设备身份与认证、加密密钥以及封装其他密钥的信任根。

但原始的 SRAM PUF 位并不完全稳定。大多数位在每次上电时都会产生相同的值，但某些位可能会因为温度、电压或噪声等因素而出现少量翻转。为了生成稳定的 HUK，芯片在 HUK 生成阶段，会生成具备纠错能力的辅助数据（huk_info）用于恢复 HUK，用户需要将这个辅助数据存储在 flash 中。

在后续的每次启动中，硬件会读取新的（带噪声的）PUF 响应，并利用辅助数据将任何翻转的位纠正回参考值，从而重新构造出部署阶段生成的完全相同的 HUK。随后，HUK 会被加载到加密硬件内部的隐藏密钥槽中（绝不会进入普通 RAM），使设备能够基于 SRAM PUF 获得一把可重复生成、每颗芯片唯一的秘密密钥，即使底层的 PUF 位在每次上电时并非 100% 相同。

36.3 术语

为了更好地说明 HUK 生成器和密钥管理器的功能，本章使用了以下术语及参数。

术语

HUK 硬件唯一密钥 (Hardware Unique Key, HUK)，使用 SRAM PUF 从设备上电时的默认初始值中派生出来的硬件密钥。

PUF 物理不可克隆函数 (Physical Unclonable Function, PUF) 基于 SRAM，属于一种安全技术，利用 SRAM 在上电时的独特且不可预测的行为为设备生成唯一的指纹或加密密钥。

HUK 生成模式 在该模式下，HUK 生成器生成新的 HUK。

HUK 恢复模式 在该模式下，HUK 生成器恢复已部署的 HUK。

随机部署模式 在该模式下，密钥管理器部署随机私钥。随机私钥由随机数生成器生成。

AES 部署模式 在该模式下，密钥管理器部署指定私钥。

ECDHO 部署模式 在该模式下，密钥管理器部署协商私钥 ($k1 * k2 * G$)。该私钥由密钥 $k1$ 和 $k2 * G$ 共同生成。

ECDH1 部署模式 在该模式下，密钥管理器部署协商私钥。该私钥由密钥 $k1$、$k2 * G$、和 $k2_info$ 共同生成。

私钥恢复模式 在该模式下，密钥管理器恢复已部署的私钥。

key_info 导出模式 在该模式下，密钥管理器重新生成 key_info，用户可将该 key_info 导出到密钥管理器的内部存储器中。

参数

huk_info 包含模糊提取信息和对 HUK 进行 RS 编码后的纠错码信息，不包含任何 HUK 本身的信息。该 huk_info 可用于恢复已部署的 HUK。

key_info 在密钥管理器的所有模式下，key_info 均等于 AES{AES{key,rand_num}||rand_num,HUK}。

$AES\{a, b\}$ 以 b 为密钥对明文 a 进行加密，算法为 AES_ECB，即 AES 模块的典型模式。

$rand_num$ 随机数

$a||b$ 将两段数据连接成一段数据。

$k1$ 指定私钥

$k2$ 辅助密钥，用于辅助密钥部署，在 AES 部署模式和 ECDH1 部署模式下使用。在 AES 部署模式下，其值不影响部署的密钥；在 ECDH1 部署模式下，其值会影响部署的协商密钥。

$k1_encrypted$ 使用 $k2$ 对 $k1$ 进行加密而生成的密钥。在 AES 部署模式和 ECDH1 部署模式下使用，$k1_encrypted$ = AES{k1,k2}

$k2_info$ $k2$ 的恢复信息，对应辅助密钥 $k2$。在 AES 部署模式和 ECDH1 部署模式下使用，$k2_info$ = AES{AES{k2,rand_num}||rand_num,init_key}。

$init_key$ 用于 $k2_info$ 的生成过程，以及从 $k2_info$ 恢复为 $k2$ 的过程。其值可以存储在用途为 EFUSE_KM_INIT_KEY 的 eFuse 块中，也可以来自于软件配置。

sw_init_key 软件配置的 $init_key$。

$k1 * k2 * G$ 协商密钥，通过在椭圆曲线上计算 $k1 * k2 * G$ 生成。

$k1 * G$ 在椭圆曲线上计算 $k1$ 生成的密钥。

$k2 * G$ 辅助公钥，通过在椭圆曲线上计算 $k2 * G$ 生成，该公钥用于生成 $k1 * k2 * G$。

$key_purpose$ 密钥的用途。

36.4 主要特性

HUK 生成器具有以下特性：

- HUK 生成模式：
  - 生成新 HUK 及其对应的 HUK 恢复信息
- HUK 恢复模式：
  - 使用 HUK 恢复信息来恢复 HUK
- HUK 恢复错误提示
- HUK 风险等级提示

密钥管理器具有以下特性：

- 密钥数量无限
- 指定私钥部署（AES 部署模式）
  - 用户可指定密钥的值
- 协商私钥部署（ECDHO 部署模式）
  - 最高安全模式，无需担心外部渠道的数据泄露
  - 获取私钥需初始化芯片
  - 密钥值为芯片与用户协商所得
- 协商私钥部署（ECDH1 部署模式）
  - 利用辅助密钥部署协商私钥
  - 获取私钥无需启动芯片
- 随机密钥部署（随机部署模式）
  - 部署硬件生成的随机密钥，无人知道其确切值
- 私钥恢复部署
  - 输入部署时生成的密钥信息可恢复完全相同的密钥
- 密钥信息导出
  - 为同一个密钥每次生成不同的密钥信息

36.5 结构概览

密钥管理器和 HUK 生成器的顶层架构如下图所示。

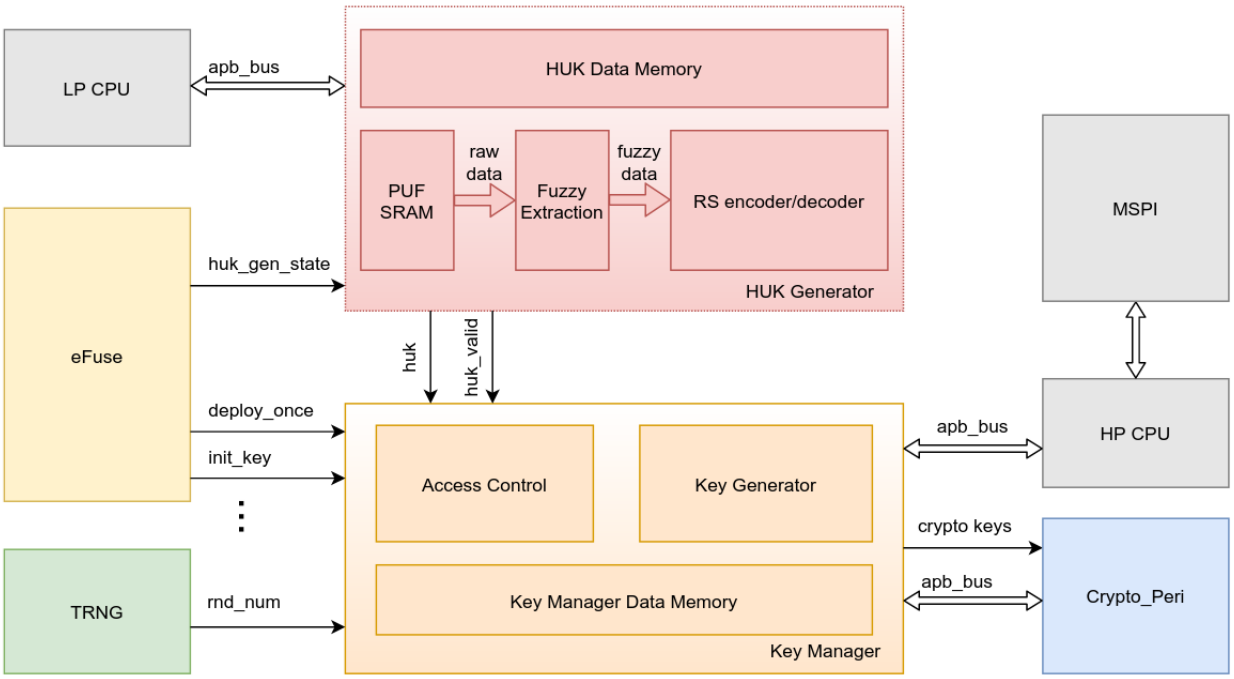


图 36.5-1. 密钥管理器 & HUK 生成器架构

密钥管理器与外部模块的交互路径如下：

- **eFuse 路径：** 使用 eFuse 位来控制密钥管理器的配置。
- **HP APB 路径：** 使用 HP APB 总线来配置寄存器和传输存储器数据。
- **HUK 路径：** 从 HUK 生成器接收 HUK。
- **Crypto 路径：** 使用 Crypto APB 总线控制 ECC/AES 等密码加速器实现密码运算。
- **TRNG 路径：** 从 TRNG 接收真随机数据。

HUK 生成器与外部模块的交互路径如下：

- **eFuse 路径：** 使用 eFuse 位来控制 HUK 生成器的配置。
- **LP APB 路径：** 使用 LP APB 总线来配置寄存器和传输存储器数据。
- **密钥管理器路径：** 发送 HUK 给密钥管理器。

36.6 功能描述

36.6.1 HUK 生成器

HUK 生成器基于 SRAM 物理不可克隆函数 (PUF) 为每个芯片生成独一无二的 HUK。软件无法以任何方式访问此 HUK，因此密钥管理器使用此 HUK 可以保证其安全性可靠。

36.6.1.1 HUK 生成过程

HUK 生成器具有以下模式：

- **HUK 生成模式：** 生成新的 HUK 及其对应的 HUK 恢复信息 (*huk_info*)
- **HUK 恢复模式：** 根据 *huk_info* 恢复 HUK

EFUSE_KM_HUK_GEN_STATE 和 HUK_MODE 共同控制 HUK 生成器的模式。

表 36.6-1. HUK 生成器工作模式

HUK_MODE	eFuse *	模式
1	配置成 1 的比特数量为偶数	HUK 生成模式
	配置成 1 的比特数量为奇数	HUK 恢复模式
0	—	HUK 恢复模式

* 这里指 EFUSE_KM_HUK_GEN_STATE 中所含 1 的总数。

说明：

- HUK 生成模式结束后，建议向 EFUSE_KM_HUK_GEN_STATE 写入一个 1，此时 eFuse 中配置成 1 的比特数量为奇数。这样，芯片下次启动时，HUK 生成器只能进入 HUK 恢复模式。此时 HUK 恢复模式下恢复的 HUK 将与本次 HUK 生成模式中生成的 HUK 相同。
- 如果在 HUK 生成模式结束后，芯片掉电前没有写入 EFUSE_KM_HUK_GEN_STATE，HUK 仍然可以配置为在下次上电时进入 HUK 生成模式，但新生成 HUK 的值不再与上次用于密钥部署的 HUK 相同。用户可以利用这一点来生成多个 HUK，同时利用其对应的 *huk_info* 恢复所需要的 HUK。
- *huk_info* 仅可被用于同一块芯片，即在芯片 A 生成的 *huk_info* 无法在芯片 B 中恢复 HUK。

36.6.1.2 HUK 状态检查

用户可读取 HUK_STATUS_REG 来判断 HUK 的当前状态，并决定是否需要重新生成 HUK。HUK_STATUS_REG 中各个字段的功能如下：

- HUK_STATUS：表示 HUK 当前的生成状态。
  - 0：HUK 尚未生成。
  - 1：HUK 已成功生成且有效。
  - 2：HUK 已生成但无效。
  - 3：保留。
- HUK_RISK_LEVEL：表示 HUK 的当前风险等级，值为 0~7，值越大，HUK 的风险越高。当值为 7 时，表示 HUK 不再可用。此时 HUK_STATUS 的值也将相应变为 2。

说明：

- HUK 生成器使用 SRAM PUF 的默认值来生成 HUK。HUK 的风险等级指的是 SRAM PUF 的当前默认值与创建 HUK 时 SRAM PUF 的默认值之间的差值。
- 在一些特殊的应用中，比如用户想弃用某个密钥，使对应的 *key_info* 变为无效时，可以选择通过更新 HUK 的方式来实现上述目的。HUK 一旦变更，用户将无法用新的 *huk_info* 结合之前的 *key_info* 恢复密钥，即之前部署的密钥及其对应的 *key_info* 均将变为无效。
- 强烈建议软件监控风险等级，并在风险值达到 4–6 时重新生成 HUK。

36.6.2 密钥管理器

说明：
每次使用密钥管理器进行密钥部署时，均需要确保 HUK 已经处于有效状态。可检查 HUK_STATUS_REG 来确定 HUK 的状态。

密钥管理器实现了包含四种模式的私钥部署流程、统一的私钥恢复流程以及 key_info 导出流程。

- **四种私钥部署模式**
  - 随机部署模式：部署随机私钥
  - AES 部署模式：部署指定私钥
  - ECDH0 部署模式：部署协商私钥
  - ECDH1 部署模式：部署协商私钥

  四种部署模式可以让用户根据需求，在不同安全等级的环境中选择最便捷的部署流程。
- **一种恢复模式**
  - 私钥恢复模式：恢复已部署的私钥
    该模式允许用户从任意数量的已部署的密钥中自由选择所需的密钥进行恢复。
- **一种导出模式**
  - 导出 key_info 模式：生成并输出已部署私钥的 key_info
    该模式允许用户为已部署的密钥重新生成 key_info。

36.6.2.1 随机部署模式

用户可使用该模式来部署随机密钥。该密钥由随机数生成器生成。

说明：
随机部署模式十分便捷，非常适用于不需要知道私钥具体值的应用场景。

36.6.2.2 AES 部署方式

用户可使用该模式来部署指定私钥。该模式需要以下密钥及参数：

- HUK：由 HUK 生成器生成，并发送给密钥管理器。
- $k1$：指定的私钥，待部署到芯片。
- $k2$：辅助密钥，用于加密 $k1$，保证在通讯链路上不存在 $k1$ 的明文，保障数据安全。
- $init_key$：初始辅助密钥，用于生成和解密 $k2_info$ 的密钥。
- $k2_info$：由 $k2$ 和 $init_key$ 生成，用于恢复得到 $k2$。计算公式为 $k2_info = AES\{AES\{k2, rand_num\}||rand_num, init_key\}$。

部署流程如下：

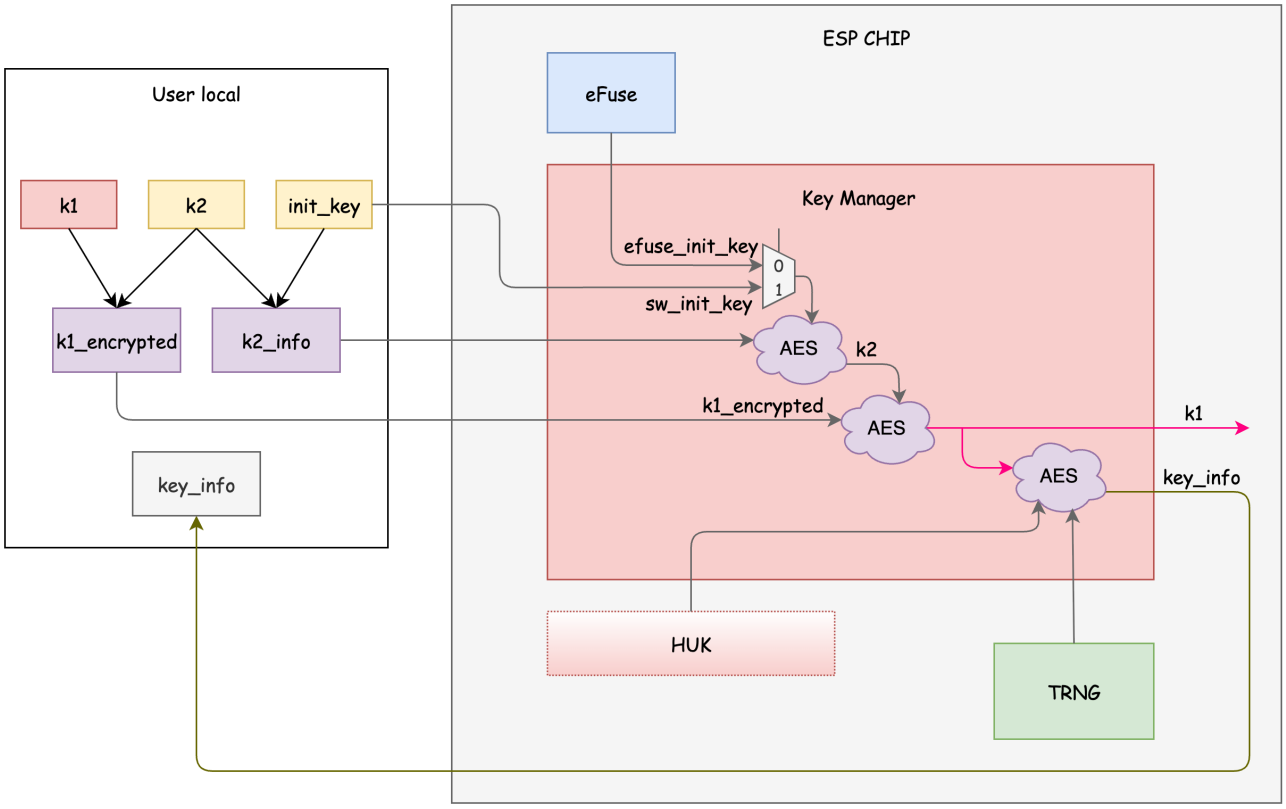


图 36.6-1. AES 部署模式

1. 用户在本地生成私钥 $k1$，辅助密钥 $k2$ 和初始辅助密钥 $init_key$。
2. 用户本地计算得到 $k2_info$ 和 $k1_encrypted$。
3. 在 ESP32-P4 密钥管理器中完成密钥部署过程：
   (a) 根据使用场景，将 $init_key$ 写入用途为 EFUSE_KM_INIT_KEY 的 eFuse 块或 sw_init_key 中。
   (b) 将 $k2_info$ 和 $k1_encrypted$ 写入密钥管理器内部存储器。
   (c) 等待密钥管理器完成部署。
   (d) 用户通过指定地址从密钥管理器内部存储器中读取 key_info。

注意：

- 由于芯片每次上电都需要重新部署密钥，如果下次上电时要使用 key_info 进行密钥恢复，则必须确保芯片断电后 key_info 能够继续保存。一种常见的方法就是将其存储在 flash 中。
- AES 部署模式适用于工厂生产：生产商可以向工厂端提供初始辅助密钥 $init_key$ 和使用 $k1$ 加密的固件，并在固件中存储 $k2_info$ 和 $k1_encrypted$。
  - 这种情况下，建议在固件中加入初次部署私钥的流程，存储 key_info，并在后续芯片上电启动中使用恢复部署模式，而不是每次都使用 AES 部署模式，以有效抵御针对 eFuse 的切片攻击。
  - 注意，这种部署方式需要保证工厂端可信，因为工厂可以获得初始辅助密钥 $init_key$ 的明文（用于烧录到 eFuse 中），如果工厂不可信，且知道如何使用 $init_key$ 将 $k2_info$ 解密为 $k2$，就可能会导致 $k1$ 的泄露。
- 在每一次使用 AES 部署模式部署密钥时，都需要保证 $init_key$，$k2_info$ 和 $k1_encrypted$ 的值不会同时泄露。

- 在使用 EFUSE_KM_INIT_KEY eFuse 块中存储的 $init_key$ 时，传输 $k2_info$ 和 $k1_encrypted$ 时可以处于不可信环境。但这意味着用户必须在本地保留第一次写入 EFUSE_KM_INIT_KEY eFuse 块中的 $init_key$。
- 在使用 sw_init_key 时，传输 sw_init_key，$k2_info$ 和 $k1_encrypted$ 时必须处于可信环境。

36.6.2.3 ECDHO 部署模式

用户可使用该模式来部署协商私钥。该模式需要以下密钥及参数：

- HUK：由 HUK 生成器生成，并发送给密钥管理器。
- $k1$：根密钥，用户自己的私钥，只存在于用户本地。
- $k1 * G$：在椭圆曲线上计算 $k1$*G 生成的密钥。
- $k2 * G$：辅助公钥，通过在椭圆曲线上计算 $k2 * G$ 生成。
- $k1 * k2 * G$：待部署的协商私钥，密钥管理器使用 $k1 * G$ 和 $k2$ 生成协商私钥，用户本地则使用 $k1$ 和 $k2 * G$ 生成协商私钥。
- key_info：$k1 * k2 * G$ 的恢复信息。

该部署过程中，任何用户本地和芯片之外的信息泄露，均不会影响 $k1 * k2 * G$ 的生成或产生泄露风险。部署流程如下：

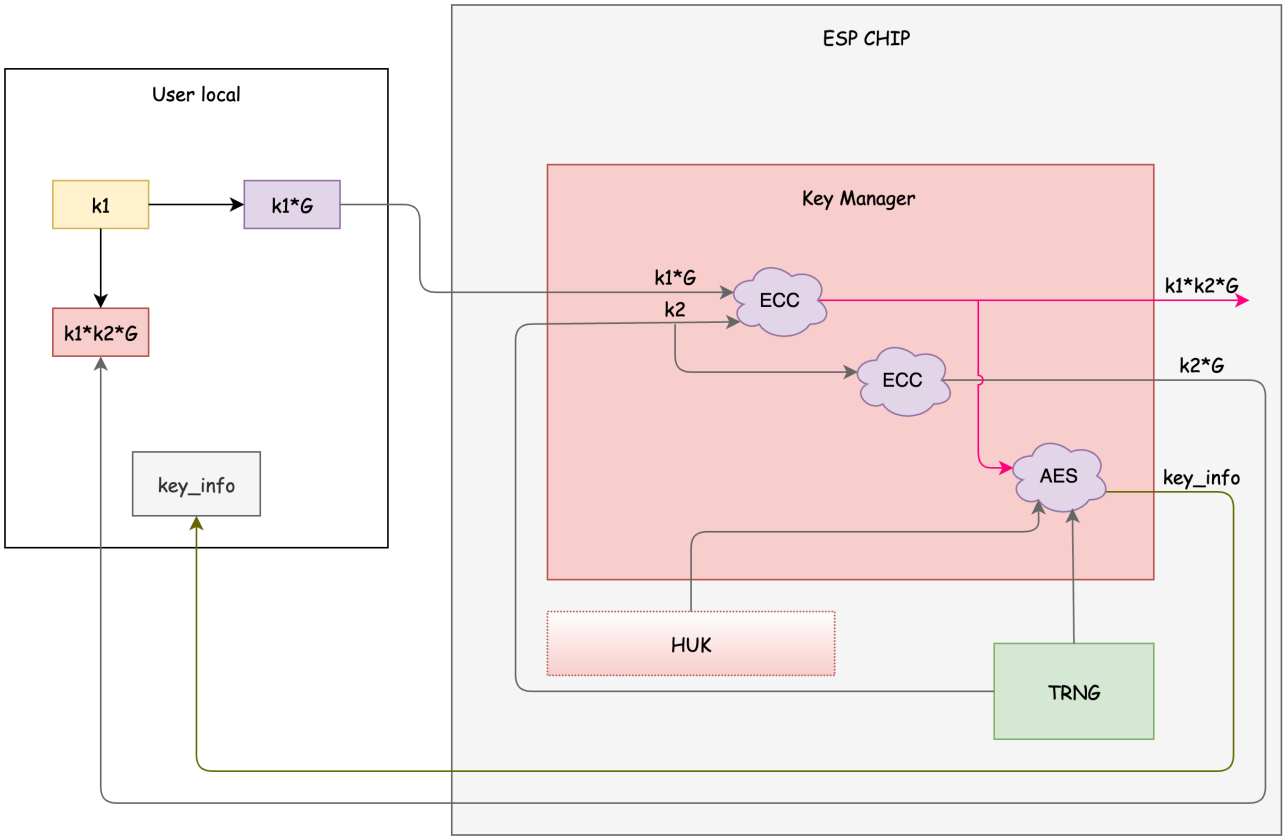


图 36.6-2. ECDHO 部署模式

1. 用户本地生成 $k1$ ，并计算得到 $k1 * G$。
2. 在 ESP32-P4 密钥管理器中完成密钥部署过程：

(a) 将 $k1 * G$ 写入密钥管理器的内部存储器。

(b) 等待密钥管理器完成部署。

(c) 用户通过指定地址从密钥管理器内部存储器中读取 $k2 * G$ 和协商私钥 $k1 * k2 * G$ 对应的 key_info。

3. 将读取的 $k2 * G$ 与用户本地私钥 $k1$ 一起生成协商密钥 $k1 * k2 * G$。

说明：

- ECDHO 部署模式采用 NIST-P256 曲线，所有椭圆曲线点乘运算均基于该曲线执行。
- ECDHO 部署模式可以保证协商私钥只能在用户本地和密钥管理器内部生成，部署过程中的任何信息泄露都不会导致协商私钥的泄露。因此，使用 ECDHO 部署模式时，传输数据的过程可以处在不可信环境。
- ECDHO 部署模式下，用户只能在运行 ESP32-P4 密钥管理器后才能获得协商好的私钥。不适用于需要提前获知私钥具体值的应用场景。

36.6.2.4 ECDH1 部署模式

用户可使用该模式来部署协商私钥。该模式需要以下密钥及参数：

- HUK：由 HUK 生成器生成，并发送给密钥管理器。
- $k1$：根密钥，用户自己的私钥，在 ECDH1 部署模式中只存在于用户本地。
- $k1 * G$：在椭圆曲线上计算 $k1$*G 生成的密钥。
- $k2$：辅助私钥，在 ECDH1 部署模式中只存在于用户本地和 ESP32-P4 密钥管理器内部。
- $k2 * G$：辅助公钥，通过在椭圆曲线上计算 $k2 * G$ 生成。
- $k2_info$：由 $k2$ 和 $init_key$ 生成，用于恢复得到 $k2$。计算公式为 $k2_info = AES\{AES\{k2, rand_num\}||rand_num, init_key\}$。
- $k1 * k2 * G$：待部署的协商私钥，密钥管理器使用 $k1 * G$ 和 $k2$ 生成协商私钥，用户本地则使用 $k1$ 和 $k2 * G$ 生成协商私钥。
- key_info：$k1 * k2 * G$ 的恢复信息。

部署流程如下：

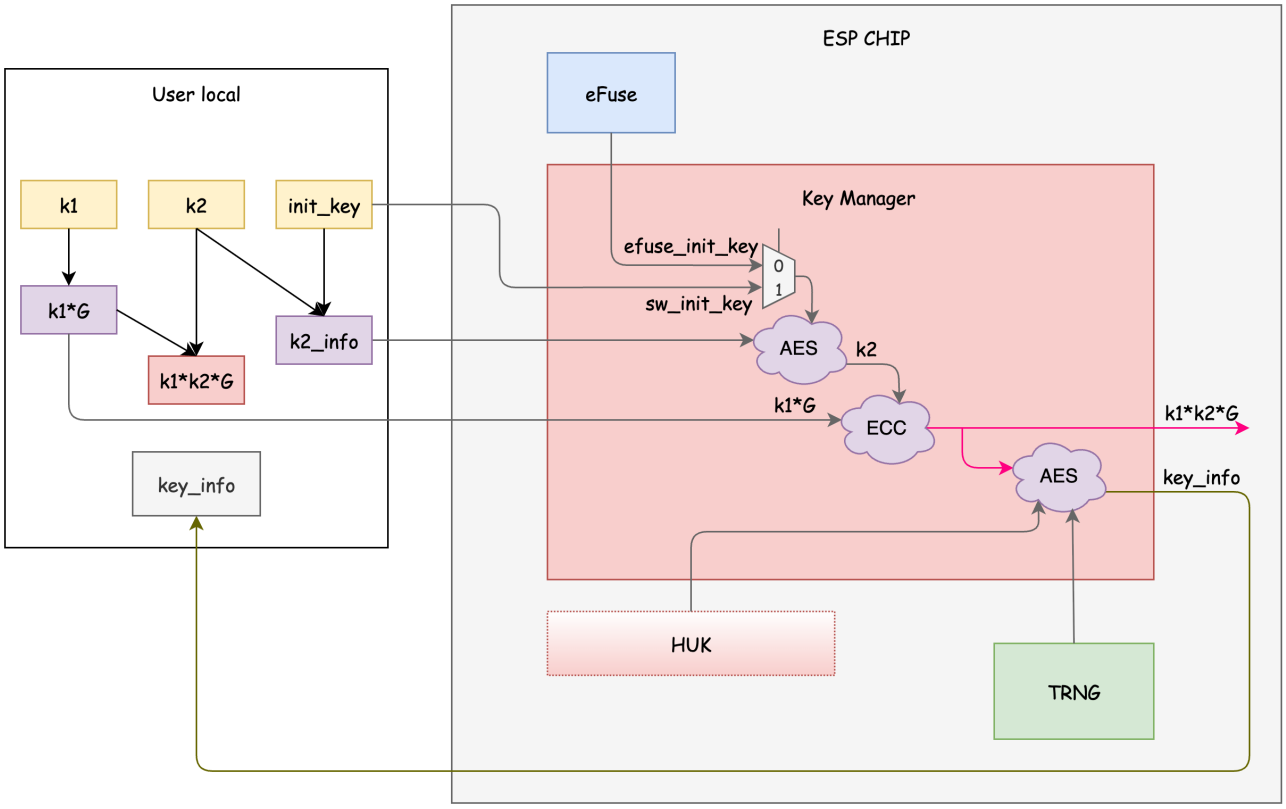


图 36.6-3. ECDH1 部署模式

1. 用户在本地生成私钥 $k1$，辅助密钥 $k2$ 和初始辅助密钥 $init_key$。
2. 用户本地计算得到 $k1 * G$，$k2 * G$，$k2_info$，以及协商私钥 $k1 * k2 * G$。
3. 在 ESP32-P4 密钥管理器中完成密钥部署过程：
   (a) 根据使用场景（见下文说明），将 $init_key$ 写入 EFUSE_KM_INIT_KEY eFuse 块或 sw_init_key 中。
   (b) 将 $k1 * G$ 和 $k2_info$ 写入密钥管理器的内部存储器。
   (c) 等待密钥管理器完成部署。
   (d) 用户通过指定地址从密钥管理器内部存储器中读取 $k2 * G$ 和协商私钥 $k1 * k2 * G$ 对应的 key_info。

说明：

- ECDH1 部署模式采用 NIST-P256 曲线，所有椭圆曲线点乘运算均基于该曲线执行。
- ECDH1 部署模式适用于工厂生产：生产商可以向工厂端提供初始辅助密钥 $init_key$ 和使用 $k1 * k2 * G$ 加密的固件，并在固件中存储 $k2_info$ 和 $k1 * G$。
  - 这种情况下，建议在固件中加入初次部署私钥的流程，存储 key_info，并在后续芯片上电启动中使用恢复部署模式，而不是每次都使用 ECDH1 部署模式，以有效抵御针对 eFuse 的切片攻击。
  - 注意，这种部署方式需要保证工厂端可信，因为工厂可以获得初始辅助密钥 $init_key$ 的明文（用于烧录到 eFuse 中），如果工厂不可信，且知道如何使用 $init_key$ 将 $k2_info$ 解密为 $k2$，就可能会导致 $k1 * k2 * G$ 的泄露。
- 与 AES 部署模式类似，在每一次使用 ECDH1 部署模式部署密钥时，都需要保证 $init_key$，$k2_info$ 和 $k1 * G$ 的值不会同时泄露。

  - 在使用 EFUSE_KM_INIT_KEY eFuse 块中存储的 $init_key$ 时，传输 $k2_info$ 和 $k1 * G$ 时可以处于不可信环境。但这意味着用户必须在本地保留第一次写入 EFUSE_KM_INIT_KEY eFuse 块中的 $init_key$。
  - 在使用 sw_init_key 时，传输 sw_init_key，$k2_info$ 和 $k1 * G$ 时必须处于可信环境。

- 使用 ECDH1 部署模式更新密钥时，比使用 AES 部署模式拥有更高的安全性能。由于 $k1 * G$ 的值与 $k2$、$init_key$ 均无关，因此在使用 ECDH1 部署模式更新协商密钥时，可以只更新 $k2_info$，而不更新 $k1 * G$。用户可以：
  1. 在可信环境下将 $k1 * G$ 写入固件。
  2. 使用 ECDH1 部署模式更新密钥时，只传输 $init_key$ 和 $k2_info$。这种情况下，用户可在不可信环境中使用 sw_init_key 进行 ECDH1 模式部署密钥。

36.6.2.5 私钥恢复模式

私钥部署完成后，用户可通过部署时生成的 key_info 来恢复该私钥。具体过程为：

1. 读取外部存储器中的 key_info 并将其存放到密钥管理器内部存储器的固定地址中。
2. 等待密钥管理器完成部署。
3. 重复上述步骤 1 ~ 步骤 2，直到恢复所有需要的私钥。注意，如果恢复了两个或多个相同 $key_purpose$ 的私钥，则最后一个私钥有效。关于 $key_purpose$ 的说明，请参考小节 4。

36.6.2.6 key_info 导出模式

私钥部署成功后，用户可使用此模式重新生成并输出 key_info。
具体过程为：

1. 选择部署成功的 $key_purpose$，进入 key_info 导出模式。
2. 等待密钥管理器完成部署。
3. 用户通过指定地址从密钥管理器内部存储器中读取 key_info。

说明：

- 该模式可用于对同一个密钥输出不同的 key_info。
- 利用该模式也可以只更新 HUK，而不更新私钥。具体操作为：
  - 先使用 HUK 生成模式更新 HUK
  - 再通过 key_info 导出模式更新 key_info。
    这样下次启动时可以使用新的 key_info 在新的 HUK 下恢复相同的私钥。

36.7 编程流程

36.7.1 HUK 生成器

HUK 生成过程分为六个阶段：

1. **IDLE 阶段**：此阶段用户配置模式和静态参数。

2. **PREP 阶段**：此阶段 HUK_STATE 处于 BUSY 状态，HUK 生成器执行 HUK 生成准备。用户只需等待此阶段结束，HUK 生成器将自动进入 LOAD 阶段。
3. **LOAD 阶段**：此阶段用户根据模式将 HUK 信息存入 HUK 生成器的内部存储器中。
4. **PROC 阶段**：此阶段 HUK_STATE 处于 BUSY 状态，HUK 生成器执行 HUK 生成操作。用户只需等待此阶段结束，HUK 生成器将自动进入 GAIN 阶段。
5. **GAIN 阶段**：此阶段用户根据模式从 HUK 生成器的内部存储器中读取所需的 HUK 信息。
6. **POST 阶段**：此阶段 HUK_STATE 处于 BUSY 状态，HUK 生成器执行 HUK 生成的后续工作。用户只需等待该阶段结束，HUK 生成器将自动回到 IDLE 阶段。

36.7.1.1 IDLE 阶段

用户检查 HUK_STATE，确认 HUK 生成器处于 IDLE 阶段。在 IDLE 阶段，用户需要进行如下配置：

1. **配置工作模式**：参考表 36.6-1 配置 HUK 生成器的工作模式。
2. **配置进入下一阶段**：置位 HUK_START，完成 IDLE 阶段，进入 PREP 阶段。

36.7.1.2 PREP 阶段

用户需要等待 PREP 阶段结束。可以通过以下任一方式完成此操作：

- **监测 HUK_STATE**：持续检查 HUK_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 HUK_PREP_DONE_INT，并通过中断函数进行处理。

36.7.1.3 LOAD 阶段

用户检查 HUK_STATE，确认 HUK 生成器处于 LOAD 阶段。在 LOAD 阶段，用户需要进行如下配置：

1. **根据模式进行配置：**
   - **HUK 生成模式**：无需配置。
   - **HUK 恢复模式**：将 HUK 恢复信息 huk_info（165 个字）写入 HUK_INFO_MEM 中。

   > **说明：**
   > 如果之前已经生成过多个 HUK 恢复信息并存储在外部存储器中，则在恢复模式下使用任意一个 HUK 恢复信息都会恢复当时对应的 HUK。

2. **配置进入下一阶段**：置位 HUK_CONTINUE，完成 LOAD 阶段，进入 PROC 阶段。

36.7.1.4 PROC 阶段

用户需要等待 PROC 阶段结束。可以通过以下任一方式完成此操作：

- **监测 HUK_STATE**：持续检查 HUK_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 HUK_PROC_DONE_INT，并通过中断函数进行处理。

36.7.1.5 GAIN 阶段

用户检查 HUK_STATE，确认 HUK 生成器处于 GAIN 阶段。在 GAIN 阶段，用户根据选择的模式提取所需的密钥信息：

1. **提取信息：**
   - **HUK 生成模式：** 从 HUK_INFO_MEM 中读取 huk_info（165 个字）。
   - **HUK 恢复模式：** 根据 HUK_UPDATE_REQ 判断是否需要更新 huk_info。
     - 需要更新：从 HUK_INFO_MEM 中读取 huk_info（165 个字）。
     - 无需更新：无需读取任何信息。
2. **配置进入下一阶段**：置位 HUK_CONTINUE，完成 GAIN 阶段，进入 POST 阶段。

36.7.1.6 POST 阶段

用户需要等待 POST 阶段结束。可以通过以下任一方式完成此操作：

- **监测 HUK_STATE**：持续检查 HUK_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 HUK_POST_DONE_INT，并通过中断函数进行处理。

36.7.2 密钥管理器

密钥管理器的密钥部署过程分为六个阶段：

1. **IDLE 阶段**：此阶段用户配置模式、密钥用途和静态参数。
2. **PREP 阶段**：此阶段 KEYMNG_STATE 处于 BUSY 状态，密钥管理器执行密钥部署准备工作。用户只需等待此阶段结束，密钥管理器将自动进入 LOAD 阶段。
3. **LOAD 阶段**：此阶段用户根据模式将密钥信息存入密钥管理器的内部存储器中。
4. **PROC 阶段**：此阶段 KEYMNG_STATE 处于 BUSY 状态，密钥管理器执行密钥部署操作。用户只需等待此阶段结束，密钥管理器将自动进入 GAIN 阶段。
5. **GAIN 阶段**：此阶段用户根据模式从密钥管理器的内部存储器中读取所需的密钥信息。
6. **POST 阶段**：此阶段 KEYMNG_STATE 处于 BUSY 状态，密钥管理器执行密钥部署的后续工作。用户只需等待该阶段结束，密钥管理器将自动回到 IDLE 阶段。

36.7.2.1 IDLE 阶段

用户检查 KEYMNG_STATE，确认密钥管理器处于 IDLE 阶段。在 IDLE 阶段，用户需要进行如下配置：

1. **完成 HUK 生成过程：**

   (a) 根据 HUK 是否已经生成，选择以下两种流程之一：
   - **HUK 尚未生成**：按照章节 36.7.1 所述完成 HUK 生成过程。
   - **HUK 已生成**：跳过 HUK 生成过程。

   (b) **验证 HUK 是否有效**：检查 KEYMNG_HUK_VALID：
   - 0，则 HUK 无效，返回步骤 1a，直到其值为 1。
   - 1，HUK 有效。
2. **正确初始化 TRNG**：用户需确保 TRNG 工作正常。
3. **配置静态参数**：包括 eFuse 位和静态参数寄存器及其对应的寄存器锁：

- **EFUSE_KM_INIT_KEY eFuse 块**：256 位初始密钥，密钥管理器在 AES 和 ECDH1 部署模式中会用到该密钥。
- EFUSE_KM_DEPLOY_ONLY_ONCE：控制对应的私钥是否只能在上电后部署一次。
  - **Bit[0]**：配置 ECDSA 数字签名外设 (ECDSA_DS) 密钥是否只能部署一次。
    - 0：可以部署多次
    - 1：只能部署一次
  - **Bit[1]**：配置 XTS 密钥是否只能部署一次。
    - 0：可以部署多次
    - 1：只能部署一次
- EFUSE_FORCE_USE_KEY_MANAGER_KEY：控制相应的私钥是否只能来自密钥管理器。
  - **Bit[0]**：表示 ECDSA_DS 密钥是否只能来自密钥管理器。
    - 0：不必来自密钥管理器
    - 1：必须来自密钥管理器
  - **Bit[1]**：表示 flash 加密密钥是否只能来自密钥管理器。
    - 0：不必来自密钥管理器
    - 1：必须来自密钥管理器
  - **Bit[2]**：表示 HMAC 密钥是否只能来自密钥管理器。
    - 0：不必来自密钥管理器
    - 1：必须来自密钥管理器
  - **Bit[3]**：表示 RSA 数字签名外设 (RSA_DS) 的密钥是否只能来自密钥管理器。
    - 0：不必来自密钥管理器
    - 1：必须来自密钥管理器
  - **Bit[4]**：表示 PSRAM 密钥是否只能来自密钥管理器。
    - 0：不必来自密钥管理器
    - 1：必须来自密钥管理器
- KEYMNG_USE_EFUSE_KEY：配置密钥的来源为 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的相应位为 0 时有效。
  - **Bit[0]**：配置 ECDSA_DS 密钥是否来自 eFuse 块。
    - 0：无效
    - 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[0] 为 0 时有效。
  - **Bit[1]**：配置 flash 加密密钥是否来自 eFuse 块
    - 0：无效
    - 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[1] 为 0 时有效。
  - **Bit[2]**：配置 HMAC 密钥是否来自 eFuse 块

    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[2] 为 0 时有效。
  - **Bit[3]**：配置 RSA_DS 的密钥是否来自 eFuse 块
    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[3] 为 0 时有效。
  - **Bit[4]**：配置 PSRAM 密钥是否来自 eFuse 块
    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[4] 为 0 时有效。
- **EFUSE_FORCE_DISABLE_SW_INIT_KEY**：控制是否永久禁止使用软件源 $init_key$。
  - 0：不禁用
  - 1：禁用软件源 $init_key$
- **KEYMNG_USE_SW_INIT_KEY**：配置使用软件配置的 sw_init_key 而不是用途为 EFUSE_KM_INIT_KEY 的 eFuse 块，仅当 EFUSE_FORCE_DISABLE_SW_INIT_KEY 为 0 时有效。
  - 0：使用用途为 EFUSE_KM_INIT_KEY 的 eFuse 块
  - 1：使用软件写入的初始密钥，即 sw_init_key
- **EFUSE_KM_RND_SWITCH_CYCLE**：配置密钥管理器切换随机数的周期。注意：此切换周期建议配置为 TRNG 模块的时钟周期所对应的密钥管理器时钟周期数。
  - 0：切换周期由 KEYMNG_RND_SWITCH_CYCLE 控制
  - 1：8 个密钥管理器时钟周期
  - 2：16 个密钥管理器时钟周期
  - 3：32 个密钥管理器时钟周期
- **KEYMNG_RND_SWITCH_CYCLE**：配置密钥管理器切换随机数的周期，单位为密钥管理器时钟周期。仅当 EFUSE_KM_RND_SWITCH_CYCLE 为 0 时生效。注意：此切换周期建议配置为 TRNG 模块的时钟周期所对应的密钥管理器时钟周期数。

4. **设置工作模式**：配置 KEYMNG_KGEN_MODE，完成密钥管理器的工作模式配置，具体配置见章节 36.6.2：
   - 0：随机部署模式
   - 1：AES 部署模式
   - 2：ECDH0 部署模式
   - 3：ECDH1 部署模式
   - 4：私钥恢复模式
   - 5：key_info 导出模式
   - 其他值：无效
5. **配置密钥**：配置 KEYMNG_KEY_PURPOSE 完成密钥用途 ($key_purpose$) 的配置：
   - 1: ecdsa_key_192

- 2: ecdsa_key_256
- 3: flash_256_1_key
- 4: flash_256_2_key
- 5: flash_128_key
- 6: hmac_key
- 7: ds_key
- 8: psram_256_1_key
- 9: psram_256_2_key
- 10: psram_128_key
- 11: ecdsa_key_384_l
- 12: ecdsa_key_384_h
- 其他值：无效

6. **配置进入下一阶段**：置位 KEYMNG_START，完成 IDLE 阶段，进入 PREP 阶段。

36.7.2.2 PREP 阶段

用户需要等待 PREP 阶段结束。可以通过以下任一方式完成此操作：

- **监测 KEYMNG_STATE**：持续检查 KEYMNG_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 KEYMNG_PREP_DONE_INT，并通过中断函数进行处理。

36.7.2.3 LOAD 阶段

用户检查 KEYMNG_STATE，确认密钥管理器处于 LOAD 阶段。在 LOAD 阶段，用户需要进行如下配置：

1. **根据工作模式进行配置：**
   - **随机部署模式**：无需配置。
   - **AES 部署模式**：
     (a) 如果使用软件配置 $init_key$ 的值：则将所需配置的值 sw_init_key（256 位）写入 KEYMNG_SW_INIT_KEY_MEM。
     (b) 将 $k2_info$（512 位）写入 KEYMNG_ASSIST_INFO_MEM。
     (c) 将 $k1_encrypted$（256 位）写入 KEYMNG_PUBLIC_INFO_MEM。
   - **ECDH0 部署模式：**
     (a) 将 $k1 * G$（512 位）写入 KEYMNG_PUBLIC_INFO_MEM。
   - **ECDH1 部署模式：**
     (a) 如果使用软件配置的 $init_key$：则将 sw_init_key（256 位）写入 KEYMNG_SW_INIT_KEY_MEM；
     (b) 将 $k2_info$（512 位）写入 KEYMNG_ASSIST_INFO_MEM。
     (c) 将 $k1 * G$（512 位）写入 KEYMNG_PUBLIC_INFO_MEM。

- **私钥恢复模式：**

  (a) 将 key_info（512 位）写入 KEYMNG_ASSIST_INFO_MEM。

- key_info **导出模式**：无需配置。

2. **配置进入下一阶段**：置位 KEYMNG_CONTINUE，完成 LOAD 阶段，进入 PROC 阶段。

36.7.2.4 PROC 阶段

用户需要等待 PROC 阶段结束。可以通过以下任一方式完成此操作：

- **监测 KEYMNG_STATE**：持续检查 KEYMNG_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 KEYMNG_PROC_DONE_INT，并通过中断函数进行处理。

36.7.2.5 GAIN 阶段

用户检查 KEYMNG_STATE，确认密钥管理器处于 GAIN 阶段。在 GAIN 阶段，用户根据选择的模式提取所需的密钥信息：

1. **提取信息：**

   - **随机部署模式：**

     (a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

   - **AES 部署模式：**

     (a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

   - **ECDHO 部署模式：**

     (a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

     (b) 从 KEYMNG_ASSIST_INFO_MEM 中读取 $k2 * G$（512 位）。

   - **ECDH1 部署模式：**

     (a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

   - **私钥恢复模式**：无需读取任何信息。

   - key_info **导出模式：**

     (a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

2. **配置进入下一阶段**：置位 KEYMNG_CONTINUE，完成 GAIN 阶段，进入 POST 阶段。

36.7.2.6 POST 阶段

用户需要等待 POST 阶段结束。可以通过以下任一方式完成此操作：

- **监测 KEYMNG_STATE**：持续检查 KEYMNG_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 KEYMNG_POST_DONE_INT，并通过中断函数进行处理。

36.8 编程示例

通过以上章节的介绍，可以总结出一套完整的芯片部署流程。本节主要介绍 HUK 生成器和密钥管理器在两种芯片典型启动过程中所需的操作步骤。

说明：
在本节中，将以简化的方式描述所有配置过程。有关详细信息，请参考第 36.7 小节。

36.8.1 芯片私钥部署阶段的软件流程示例

芯片首次启动和部署私钥对外部存储器进行加密/解密的过程：

1. 芯片进入下载模式。
2. **配置 HUK 生成器**：由于是第一次启动芯片，所以需要配置 HUK 生成器进入 HUK 生成模式：
   (a) 配置 IDLE 阶段，根据表 36.6-1 配置 HUK 工作模式为 HUK 生成模式，然后置位 HUK_START。
   (b) 进入 LOAD 阶段后，置位 HUK_CONTINUE。
   (c) 进入 GAIN 阶段后，读取 huk_info，然后置位 HUK_CONTINUE。
   (d) 等待 HUK 生成器回到 IDLE 阶段。
3. **配置 HUK 生成状态**：写入 EFUSE_KM_HUK_GEN_STATE，使其 1 的个数为奇数，以便实现下次启动时只能进入 HUK 恢复模式。
4. **部署密钥**：以 ECDHO 模式下的密钥部署为例：
   (a) 配置 IDLE 阶段，配置 KEYMNG_KEY_PURPOSE 为 XTS，然后置位 KEYMNG_START。
   (b) 进入 LOAD 阶段后，将本地生成的 $k1 * G$ 写入密钥管理器，然后置位 KEYMNG_CONTINUE。
   (c) 进入 GAIN 阶段后，读取 key_info 和 $k2 * G$，然后置位 KEYMNG_CONTINUE。
   (d) 等待密钥管理器回到 IDLE 阶段。
5. **上传密钥**：通过通讯接口上传 $k2 * G$ 给用户。
6. **生成并加密数据：**
   - 用户根据获得的 $k2 * G$ 和本地的 $k1$，生成协商私钥 $k1 * k2 * G$。
   - 对数据进行本地加密，最后将密文写入 flash。

36.8.2 芯片正常启动阶段的软件流程示例

当芯片部署完成后，正常启动的完整流程如下：

1. **芯片进入 SPI 启动模式：**
   芯片以 SPI 启动模式启动，首先进入未加密的程序段。
2. **配置 HUK 恢复模式：**
   由于 HUK 已经生成，下面配置 HUK 恢复模式：
   (a) 配置 IDLE 阶段，根据表 36.6-1 配置 HUK 生成器的工作模式为 HUK 恢复模式，然后置位 HUK_START。

(b) 进入 LOAD 阶段后，将 huk_info 写入 HUK 生成器，然后配置 HUK_CONTINUE。

(c) 进入 GAIN 阶段后，置位 HUK_CONTINUE。

(d) 等待 HUK 生成器回到 IDLE 阶段。

3. **配置密钥管理器为私钥恢复模式：**

(a) 配置 IDLE 阶段，配置 KEYMNG_KEY_PURPOSE 为 flash_128_key，然后置位 KEYMNG_START。

(b) 进入 LOAD 阶段后，将 key_info 写入密钥管理器，然后置位 KEYMNG_CONTINUE。

(c) 进入 GAIN 阶段后，置位 KEYMNG_CONTINUE。

(d) 等待密钥管理器回到 IDLE 阶段。

4. **进入加密程序段**：此时外部存储器中的的加/解密私钥已经生效，系统进入加密程序段。

36.9 中断

ESP32-P4 的密钥管理器和 HUK 生成器可以生成以下中断信号并发送给 中断矩阵。

- KEYMNG_INTR
- HUK_INTR

密钥管理器和 HUK 生成器有多个中断源可生成以上中断信号。表 36.9-1 列出了密钥管理器和 HUK 生成器内部的中断源及其触发条件。

表 36.9-1. 密钥管理器和 HUK 生成器内部中断源

内部中断源	触发条件	中断信号
KEYMNG_PREP_DONE_INT	密钥管理器 PREP 阶段完成	KEYMNG_INTR
KEYMNG_PROC_DONE_INT	密钥管理器 PROC 阶段完成	KEYMNG_INTR
KEYMNG_POST_DONE_INT	密钥管理器 POST 阶段完成	KEYMNG_INTR
HUK_PREP_DONE_INT	HUK 生成器 PREP 阶段完成	HUK_INTR
HUK_PROC_DONE_INT	HUK 生成器 PROC 阶段完成	HUK_INTR
HUK_POST_DONE_INT	HUK 生成器 POST 阶段完成	HUK_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

36.10 内存块

本小节的所有地址均为相对于密钥管理器或 HUK 生成器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

表 36.10-1. 密钥管理器内存块

名称	描述	容量（字节）	起始地址	结束地址	访问
KEYMNG_ASSIST_INFO_MEM	存储辅助信息	64	0x0100	0x013F	Varies

KEYMNG_PUBLIC_INFO_MEM	存储公共信息	64	0x0140	0x017F	Varies
KEYMNG_SW_INIT_KEY_MEM	存储 *sw_init_key*	32	0x0180	0x019F	Varies

表 36.10-2. HUK 生成器内存块

名称	描述	容量（字节）	起始地址	结束地址	访问
HUK_INFO_MEM	存储 *huk_info*	660	0x0100	0x027F	Varies

36.11 寄存器列表

36.11.1 HUK 寄存器列表

本小节的所有地址均为相对于 HUK 基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

请查看章节 *Access 寄存器的访问类型*，了解**访问**列缩写的含义。

名称	描述	地址	访问
时钟门控寄存器			
HUK_CLK_REG	时钟门控制寄存器	0x0004	R/W
中断寄存器			
HUK_INT_RAW_REG	原始中断寄存器（电平有效）	0x0008	RO/ WTC/ SS
HUK_INT_ST_REG	中断状态寄存器	0x000C	RO
HUK_INT_ENA_REG	中断使能寄存器	0x0010	R/W
HUK_INT_CLR_REG	中断清除寄存器	0x0014	WT
配置寄存器			
HUK_CONF_REG	配置寄存器	0x0020	R/W
控制寄存器			
HUK_START_REG	控制寄存器	0x0024	WT
状态寄存器			
HUK_STATE_REG	HUK 生成器状态寄存器	0x0028	RO
结果寄存器			
HUK_STATUS_REG	HUK 状态寄存器	0x0034	RO
版本寄存器			
HUK_DATE_REG	版本控制寄存器	0x00FC	R/W

36.11.2 密钥管理器寄存器列表

本小节的所有地址均为相对于密钥管理器基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

请查看章节 *Access 寄存器的访问类型*，了解**访问**列缩写的含义。

名称	描述	地址	访问
时钟门控寄存器			
KEYMNG_CLK_REG	时钟门控寄存器	0x0004	R/W
中断寄存器			
KEYMNG_INT_RAW_REG	原始中断寄存器（电平有效）	0x0008	RO/ WTC/ SS
KEYMNG_INT_ST_REG	中断状态寄存器	0x000C	RO
KEYMNG_INT_ENA_REG	中断使能寄存器	0x0010	R/W
KEYMNG_INT_CLR_REG	中断清除寄存器	0x0014	WT
静态配置寄存器			
KEYMNG_STATIC_REG	静态配置寄存器	0x0018	R/W
KEYMNG_LOCK_REG	静态配置锁寄存器	0x001C	R/W1
配置寄存器			
KEYMNG_CONF_REG	配置寄存器	0x0020	R/W
控制寄存器			
KEYMNG_START_REG	控制寄存器	0x0024	WT
状态寄存器			
KEYMNG_STATE_REG	状态寄存器	0x0028	RO
结果寄存器			
KEYMNG_RESULT_REG	运算结果寄存器	0x002C	RO/ SS
KEYMNG_KEY_VLD_REG	密钥状态寄存器	0x0030	RO
KEYMNG_HUK_VLD_REG	HUK 状态寄存器	0x0034	RO
版本寄存器			
KEYMNG_DATE_REG	版本控制寄存器	0x00FC	R/W

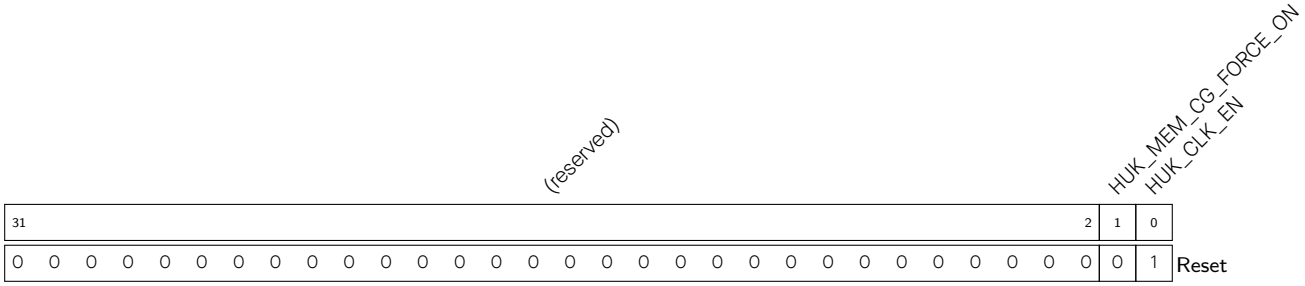
36.12 寄存器

36.12.1 HUK 寄存器

本小节的所有地址均为相对于 HUK 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 36.1. HUK_CLK_REG (0x0004)



HUK_CLK_EN 配置是否强制打开寄存器时钟门控。

0：不强制打开

1：强制打开

(R/W)

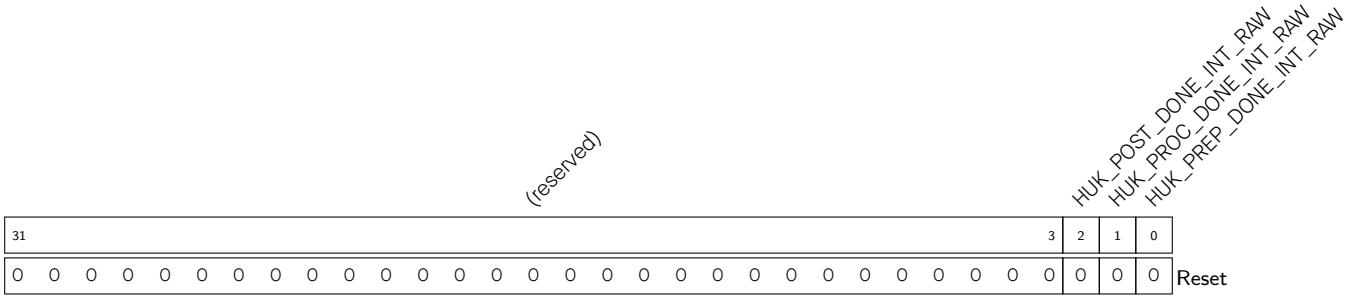
HUK_MEM_CG_FORCE_ON 配置是否强制打开存储器时钟门控。

0：不强制打开

1：强制打开

(R/W)

Register 36.2. HUK_INT_RAW_REG (0x0008)



HUK_PREP_DONE_INT_RAW HUK_PREP_DONE_INT 的原始中断状态。

(RO/WTC/SS)

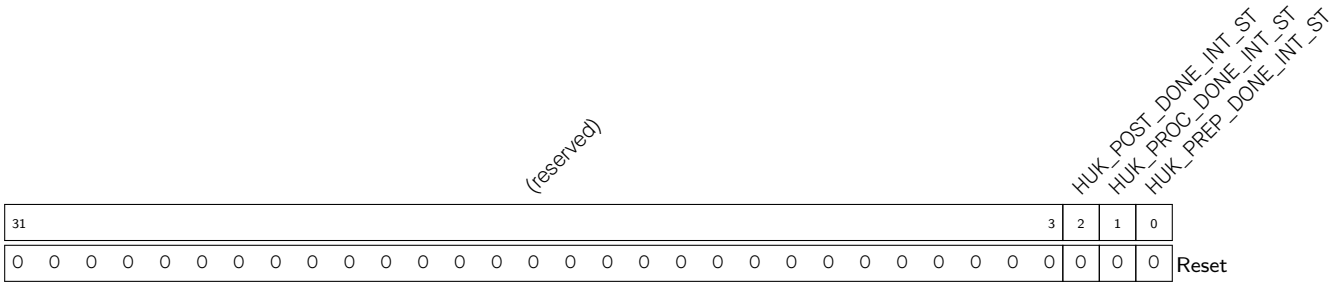
HUK_PROC_DONE_INT_RAW HUK_PROC_DONE_INT 的原始中断状态。

(RO/WTC/SS)

HUK_POST_DONE_INT_RAW HUK_POST_DONE_INT 的原始中断状态。

(RO/WTC/SS)

Register 36.3. HUK_INT_ST_REG (0x000C)

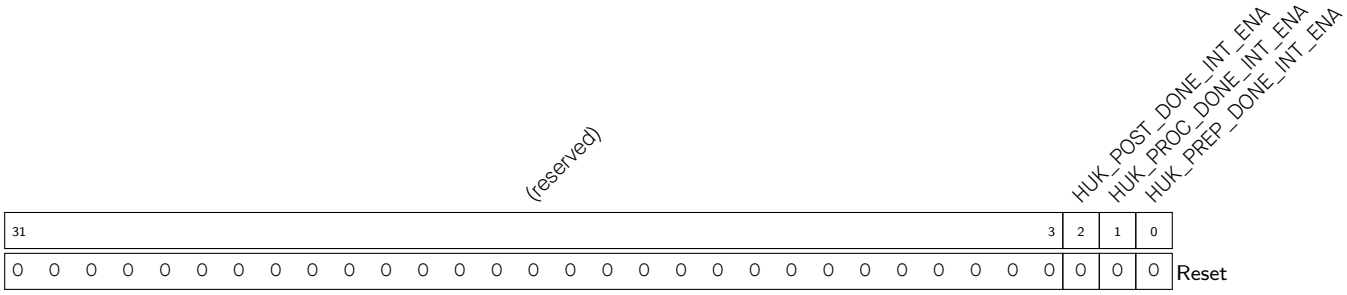


HUK_PREP_DONE_INT_ST HUK_PREP_DONE_INT 的屏蔽中断状态。
(RO)

HUK_PROC_DONE_INT_ST HUK_PROC_DONE_INT 的屏蔽中断状态。
(RO)

HUK_POST_DONE_INT_ST HUK_POST_DONE_INT 的屏蔽中断状态。
(RO)

Register 36.4. HUK_INT_ENA_REG (0x0010)

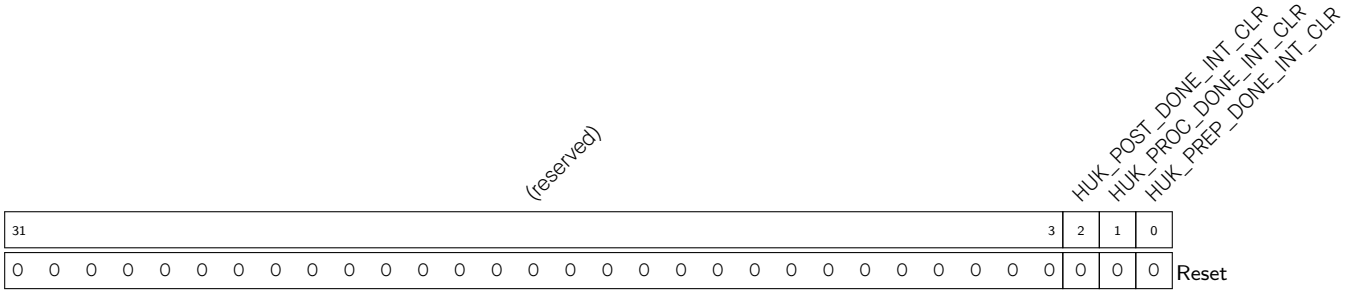


HUK_PREP_DONE_INT_ENA 写 1 使能 HUK_PREP_DONE_INT 中断。
(R/W)

HUK_PROC_DONE_INT_ENA 写 1 使能 HUK_PROC_DONE_INT 中断。
(R/W)

HUK_POST_DONE_INT_ENA 写 1 使能 HUK_POST_DONE_INT 中断。
(R/W)

Register 36.5. HUK_INT_CLR_REG (0x0014)



HUK_PREP_DONE_INT_CLR 写 1 清除 HUK_PREP_DONE_INT 中断。
(WT)

HUK_PROC_DONE_INT_CLR 写 1 清除 HUK_PROC_DONE_INT 中断。
(WT)

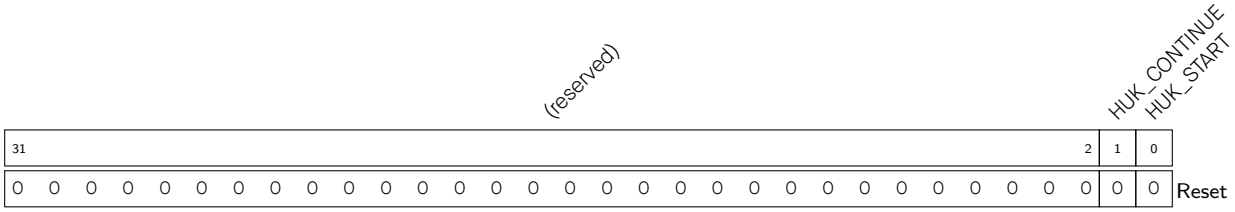
HUK_POST_DONE_INT_CLR 写 1 清除 HUK_POST_DONE_INT 中断。
(WT)

Register 36.6. HUK_CONF_REG (0x0020)



HUK_MODE 配置 HUK 模式。
0：HUK 恢复模式
1：HUK 生成模式
(R/W)

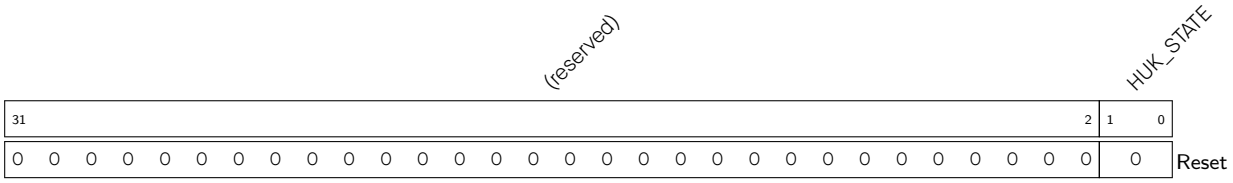
Register 36.7. HUK_START_REG (0x0024)



HUK_START 写入 1 以在 IDLE 状态下启动 HUK 生成器。
(WT)

HUK_CONTINUE 写入 1 继续 LOAD/GAIN 状态下的 HUK 生成器操作。
(WT)

Register 36.8. HUK_STATE_REG (0x0028)



HUK_STATE 表示 HUK 生成器的状态。

0: IDLE

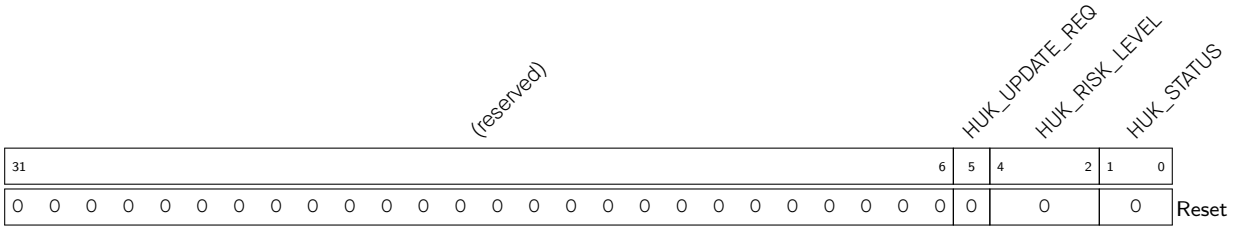
1: LOAD

2: GAIN

3: BUSY

(RO)

Register 36.9. HUK_STATUS_REG (0x0034)



HUK_STATUS 代表 HUK 生成状态。

0：未生成 HUK。

1：HUK 已生成且有效。

2：HUK 已生成但无效。

3：保留。

(RO)

HUK_RISK_LEVEL 代表 HUK 的风险等级。

0~6：风险等级越高，PUF SRAM 中的错误比特越多。

7：等级错误，HUK 无效。

(RO)

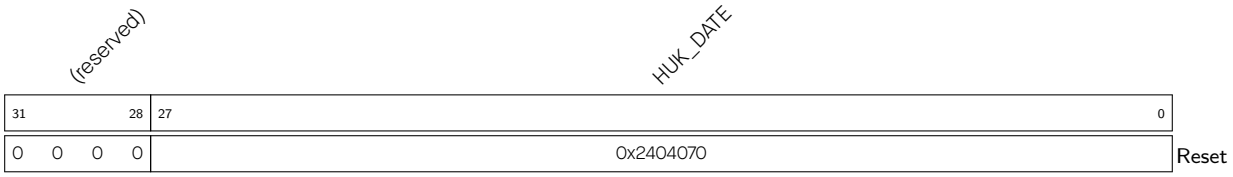
HUK_UPDATE_REQ 表示 huk_info 的更新要求。

0：用户可根据风险等级选择更新 huk_info。

1：HUK info 已过期，用户需更新 huk_info。

(RO)

Register 36.10. HUK_DATE_REG (0x00FC)



HUK_DATE 版本控制寄存器。

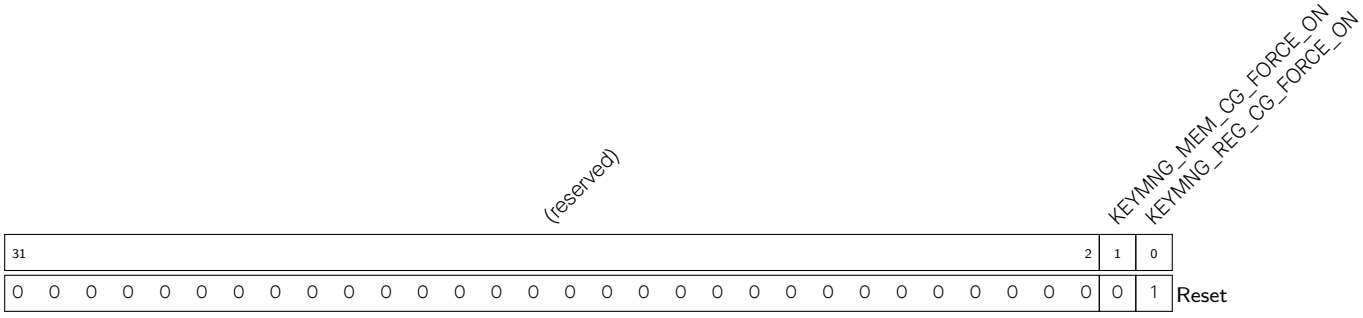
(R/W)

36.12.2 密钥管理器寄存器

本小节的所有地址均为相对于密钥管理器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 36.11. KEYMNG_CLK_REG (0x0004)



KEYMNG_REG_CG_FORCE_ON 配置是否强制打开寄存器时钟门控。

0：不强制打开

1：强制打开

(R/W)

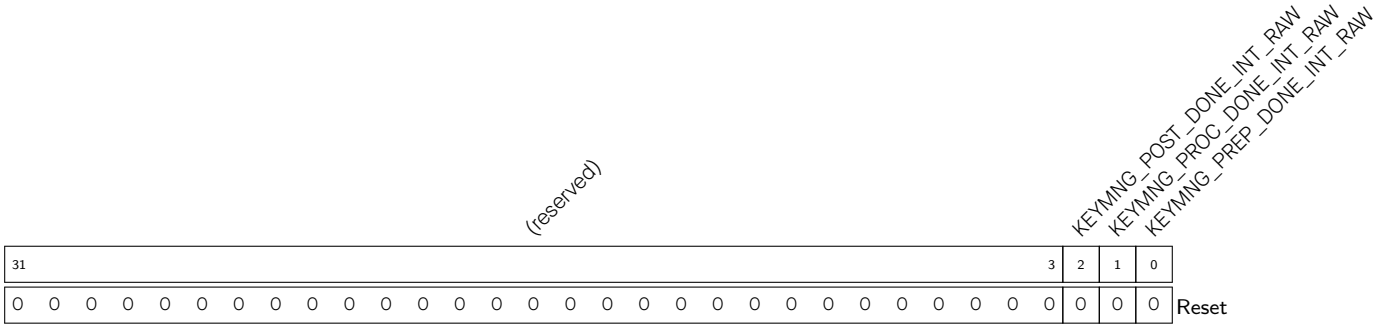
KEYMNG_MEM_CG_FORCE_ON 配置是否强制打开存储器时钟门控。

0：不强制打开

1：强制打开

(R/W)

Register 36.12. KEYMNG_INT_RAW_REG (0x0008)

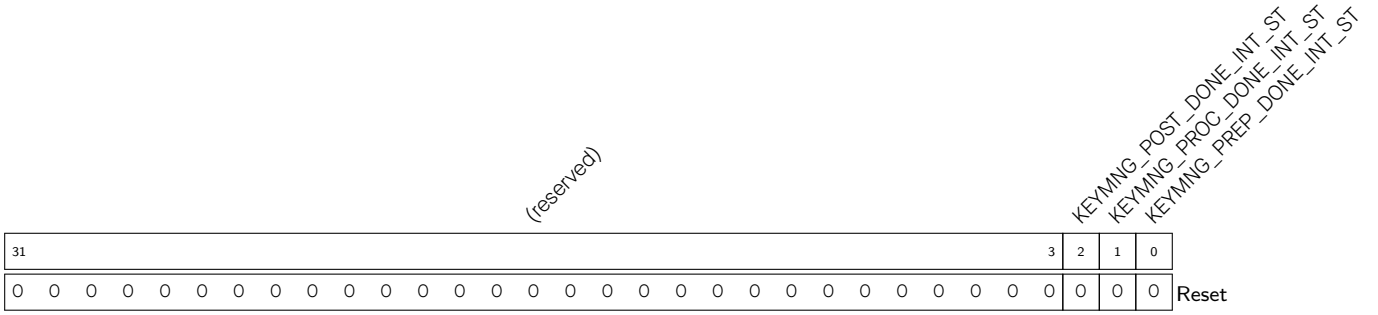


KEYMNG_PREP_DONE_INT_RAW KEYMNG_PREP_DONE_INT 的原始中断状态。
(RO/WTC/SS)

KEYMNG_PROC_DONE_INT_RAW KEYMNG_PROC_DONE_INT 的原始中断状态。
(RO/WTC/SS)

KEYMNG_POST_DONE_INT_RAW KEYMNG_POST_DONE_INT 的原始中断状态。
(RO/WTC/SS)

Register 36.13. KEYMNG_INT_ST_REG (0x000C)

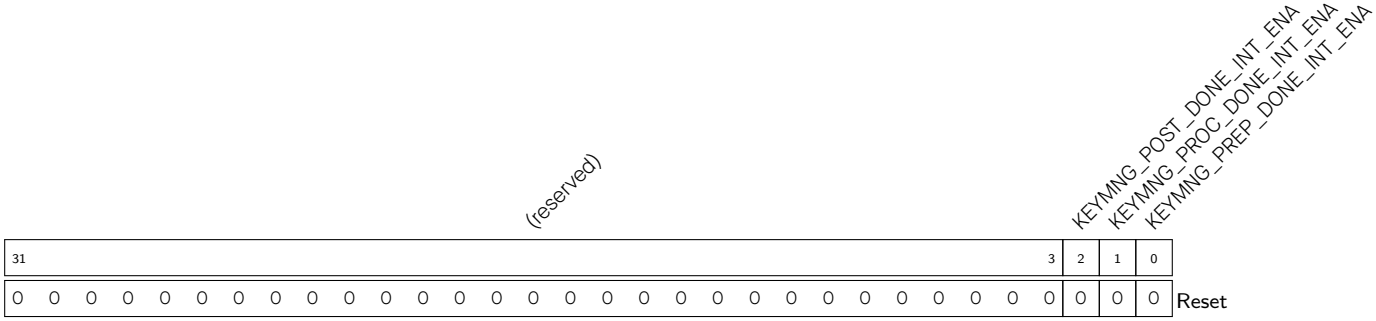


KEYMNG_PREP_DONE_INT_ST KEYMNG_PREP_DONE_INT 的屏蔽中断状态。
(RO)

KEYMNG_PROC_DONE_INT_ST KEYMNG_PROC_DONE_INT 的屏蔽中断状态。
(RO)

KEYMNG_POST_DONE_INT_ST KEYMNG_POST_DONE_INT 的屏蔽中断状态。
(RO)

Register 36.14. KEYMNG_INT_ENA_REG (0x0010)

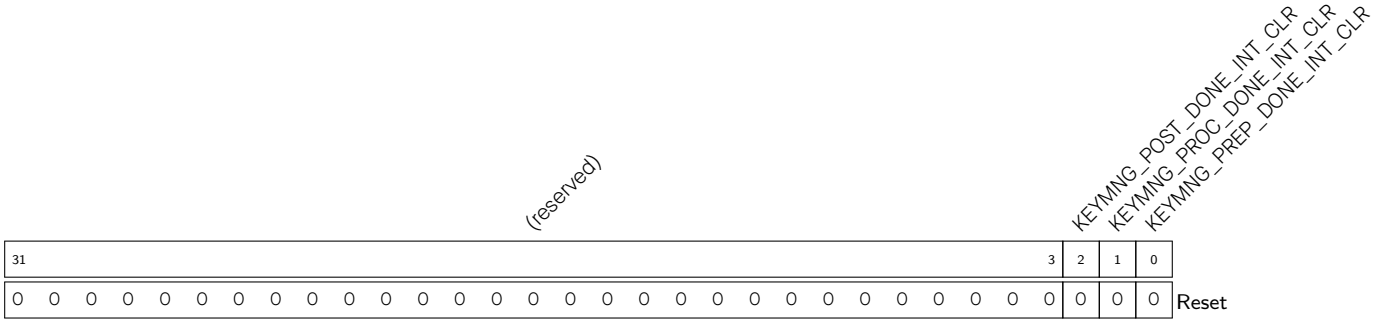


KEYMNG_PREP_DONE_INT_ENA 写 1 使能 KEYMNG_PREP_DONE_INT 中断。 (R/W)

KEYMNG_PROC_DONE_INT_ENA 写 1 使能 KEYMNG_PROC_DONE_INT 中断。 (R/W)

KEYMNG_POST_DONE_INT_ENA 写 1 使能 KEYMNG_POST_DONE_INT 中断。 (R/W)

Register 36.15. KEYMNG_INT_CLR_REG (0x0014)

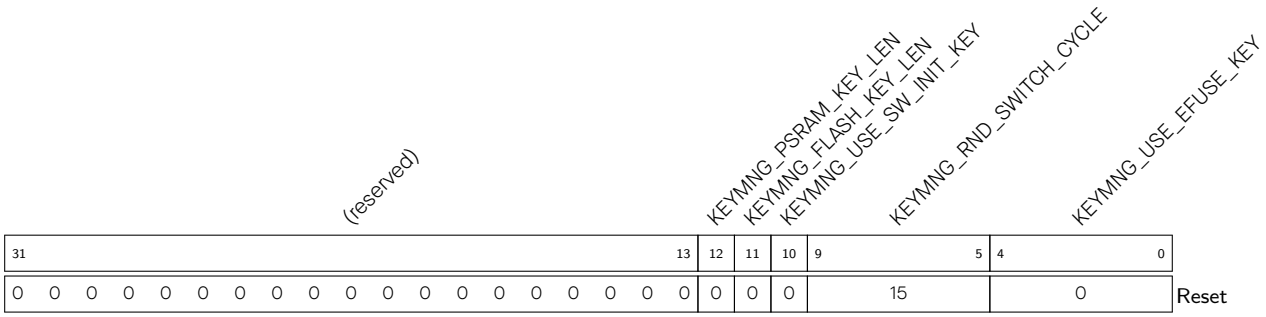


KEYMNG_PREP_DONE_INT_CLR 写 1 清除 KEYMNG_PREP_DONE_INT 中断。 (WT)

KEYMNG_PROC_DONE_INT_CLR 写 1 清除 KEYMNG_PROC_DONE_INT 中断。 (WT)

KEYMNG_POST_DONE_INT_CLR 写 1 清除 KEYMNG_POST_DONE_INT 中断。 (WT)

Register 36.16. KEYMNG_STATIC_REG (0x0018)



KEYMNG_USE_EFUSE_KEY 配置是否使用 eFuse 密钥，而不使用密钥管理器部署的密钥。

- Bit[0]：配置 ECDSA_DS 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[0] 为 0 时有效。
- Bit[1]：配置 flash 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[1] 为 0 时有效。
- Bit[2]：配置 HMAC 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[2] 为 0 时有效。
- Bit[3]：配置 RSA_DS_KEY 使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[3] 为 0 时有效。
- Bit[4]：配置 PSRAM 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[4] 为 0 时有效。

(R/W)

KEYMNG_RND_SWITCH_CYCLE 配置密钥管理器切换随机数的周期，单位为密钥管理器时钟周期。仅当 EFUSE_KM_RND_SWITCH_CYCLE 为 0 时生效。
注意：此切换周期建议配置为 TRNG 模块的时钟周期所对应的密钥管理器时钟周期数。
(R/W)

KEYMNG_USE_SW_INIT_KEY 配置使用软件配置的 sw_init_key 而不是 EFUSE_KM_INIT_KEY，仅当 EFUSE_FORCE_DISABLE_SW_INIT_KEY 为 0 时有效。
0：使用 EFUSE_KM_INIT_KEY
1：使用软件写入的初始密钥，即 sw_init_key
(R/W)

见下页

Register 36.16. KEYMNG_STATIC_REG (0x0018)

接上页

KEYMNG_FLASH_KEY_LEN 配置 flash 加密的密钥长度。

0：使用 xts-aes-128

1：使用 xts-aes-256

(R/W)

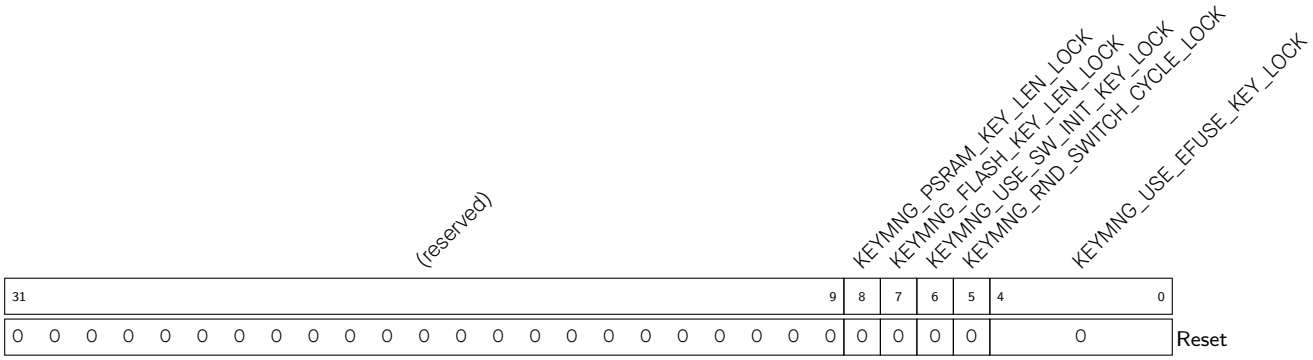
KEYMNG_PSRAM_KEY_LEN 配置 PSRAM 加密的密钥长度。

0：使用 xts-aes-128

1：使用 xts-aes-256

(R/W)

Register 36.17. KEYMNG_LOCK_REG (0x001C)



KEYMNG_USE_EFUSE_KEY_LOCK 写 1 锁定 KEYMNG_USE_EFUSE_KEY。

每一位分别锁定 KEYMNG_USE_EFUSE_KEY 的相应位。

(R/W1)

KEYMNG_RND_SWITCH_CYCLE_LOCK 写 1 锁定 KEYMNG_RND_SWITCH_CYCLE。

(R/W1)

KEYMNG_USE_SW_INIT_KEY_LOCK 写 1 锁定 KEYMNG_USE_SW_INIT_KEY。

(R/W1)

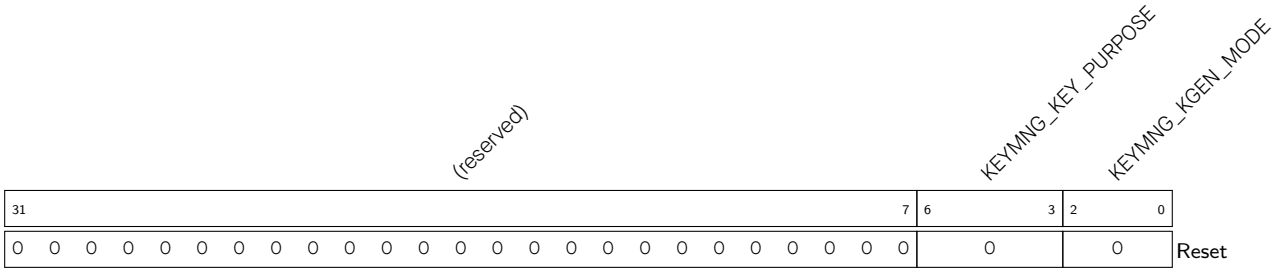
KEYMNG_FLASH_KEY_LEN_LOCK 写 1 锁定 KEYMNG_FLASH_KEY_LEN。

(R/W1)

KEYMNG_PSRAM_KEY_LEN_LOCK 写 1 锁定 KEYMNG_PSRAM_KEY_LEN。

(R/W1)

Register 36.18. KEYMNG_CONF_REG (0x0020)



KEYMNG_KGEN_MODE 配置密钥管理器的部署模式。

0：随机部署模式

1：AES 部署模式

2：ECDH0 部署模式

3：ECDH1 部署模式

4：私钥恢复模式

5：导出 key_info 模式

6~7：保留

(R/W)

KEYMNG_KEY_PURPOSE 配置密钥用途

1: ecdsa_key_192

2: ecdsa_key_256

3: flash_256_1_key

4: flash_256_2_key

5: flash_128_key

6: hmac_key

7: ds_key

8: psram_256_1_key

9: psram_256_2_key

10: psram_128_key

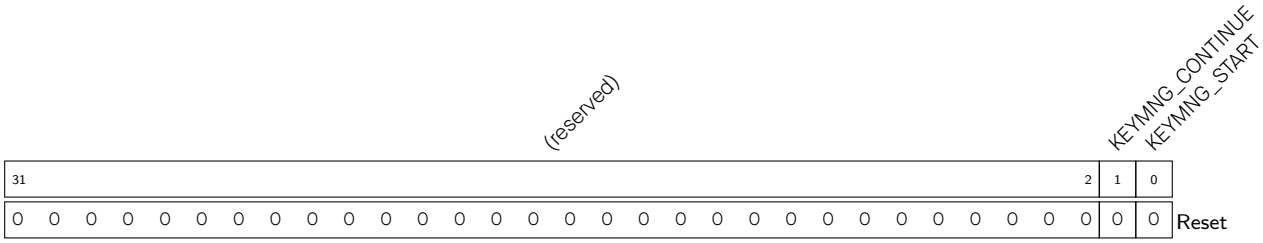
11: ecdsa_key_384_l

12: ecdsa_key_384_h

其他值：保留

(R/W)

Register 36.19. KEYMNG_START_REG (0x0024)



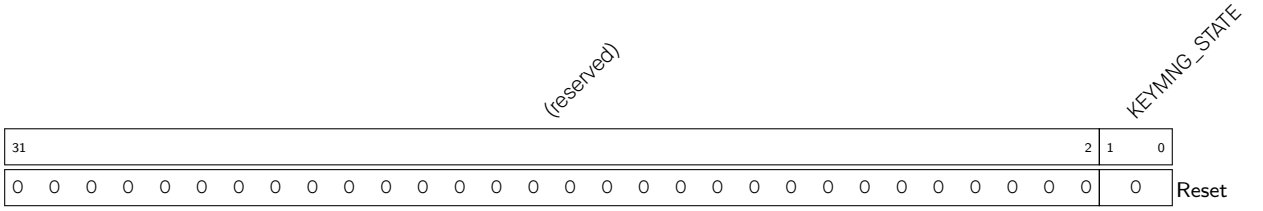
KEYMNG_START 写入 1 以在 IDLE 状态下启动密钥管理器。

(WT)

KEYMNG_CONTINUE 写入 1 继续 LOAD/GAIN 状态下的密钥管理器操作。

(WT)

Register 36.20. KEYMNG_STATE_REG (0x0028)



KEYMNG_STATE 表示密钥管理器的状态。

0: IDLE

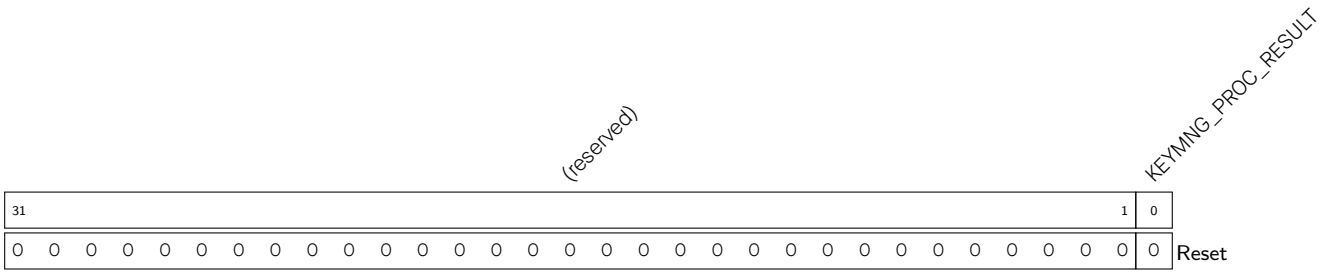
1: LOAD

2: GAIN

3: BUSY

(RO)

Register 36.21. KEYMNG_RESULT_REG (0x002C)



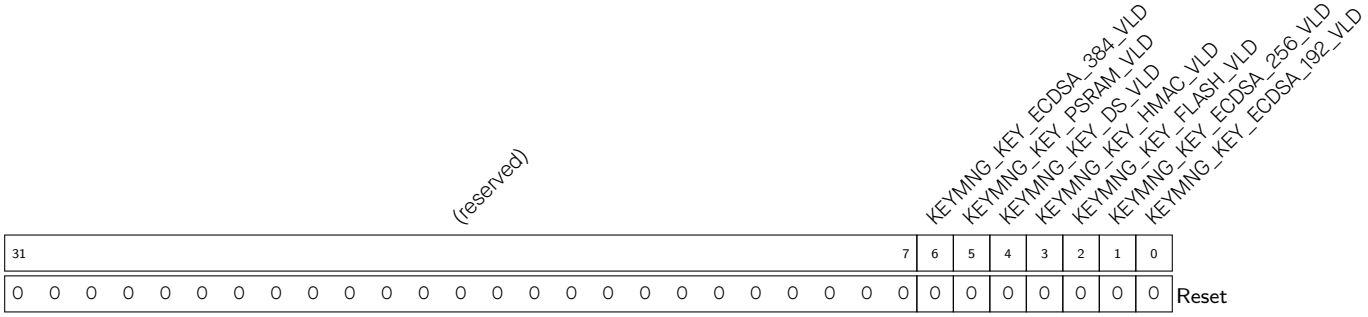
KEYMNG_PROC_RESULT 表示密钥管理器的处理结果，仅当密钥管理器处理完成后才有效。

0：密钥管理器程序失败。

1：密钥管理器程序成功。

(RO/SS)

Register 36.22. KEYMNG_KEY_VLD_REG (0x0030)



KEYMNG_KEY_ECDSA_192_VLD 表示 key_ecdsa_192 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_ECDSA_256_VLD 表示 key_ecdsa_256 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_FLASH_VLD 表示 key_flash 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_HMAC_VLD 表示 key_hmac 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_DS_VLD 表示 key_ds 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_PSRAM_VLD 表示 key_psram 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

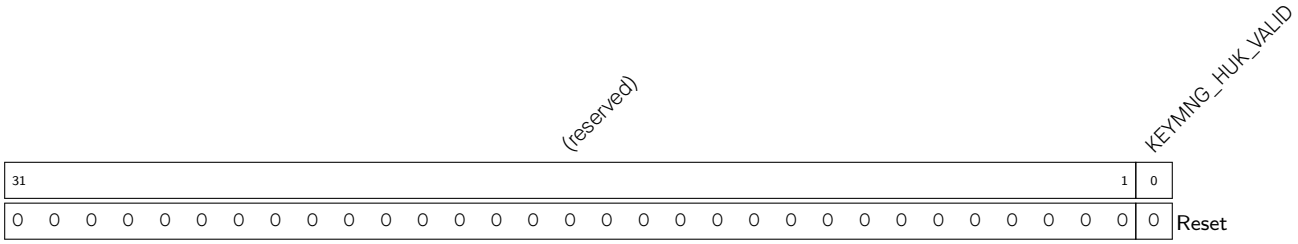
KEYMNG_KEY_ECDSA_384_VLD 表示 key_ecdsa_384 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

Register 36.23. KEYMNG_HUK_VLD_REG (0x0034)



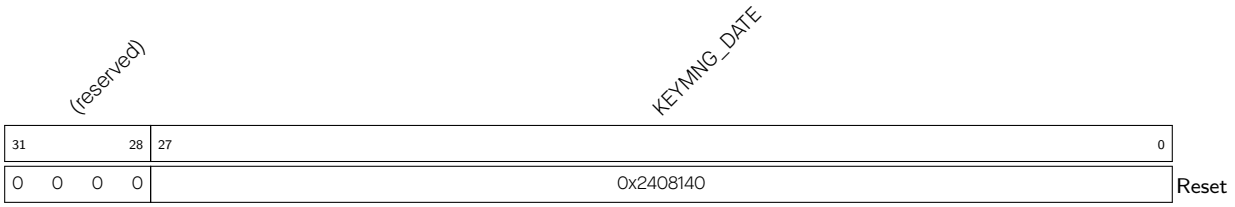
KEYMNG_HUK_VALID 代表 HUK 状态。

0：HUK 无效

1：HUK 有效

(RO)

Register 36.24. KEYMNG_DATE_REG (0x00FC)



KEYMNG_DATE 版本控制寄存器。

(R/W)

第 VI 卷

图像和语音处理

该部分涉及芯片的多媒体处理功能，介绍了与图像和语音处理相关的组件，包括摄像头控制器、视频编码器、图像信号处理器和语音活动检测。

第 37 章

JPEG 图像编解码器

ESP32-P4 的 JPEG 图像编解码器是一种基于 JPEG 基线标准的图像编解码器，可以对图像进行压缩（编码）和解压缩（解码），从而降低传输图像所需的带宽或存储图像所需的空间，可以处理高分辨率的图像。图像编码流程主要包括离散余弦变换 (DCT)、Z 形 (Zigzag) 扫描、量化和哈夫曼编码。图像解码流程是编码流程的逆操作，主要包括哈夫曼解码、反量化、反 Z 形扫描和反离散余弦变换。

37.1 术语

MCU	最小编码单元，即编码和解码的基础单位
基线 (baseline)	JPEG 协议中规定的一种基于 DCT 的顺序编码和解码处理，在所有基于 DCT 的解码处理中都需用到
图像 (image)	整数二维数组集
分量 (component)	二维数组组成的图像中的一维
数据单元 (data unit)	组成分量的 8 × 8 像素块
标记 (marker)	两字节码，第一个字节为十六进制的 0xFF，第二个字节为 1 至 0xFE 区间内的值，用于识别压缩数据格式的各个组成部分
恢复标记 (restart marker)	区分两个编码后数据段的标记，用于解码器再同步、错误恢复等特定用途
恢复间隔 (restart interval)	两个恢复标记间隔的整数个 MCU
扫描 (scan)	对一幅图像分量的一个或多个数据进行的单一浏览

图 37.1-1 为 JPEG 码流的组成结构图。图中虚线部分表示该部分码流的组成结构，黄色部分表示标记码。

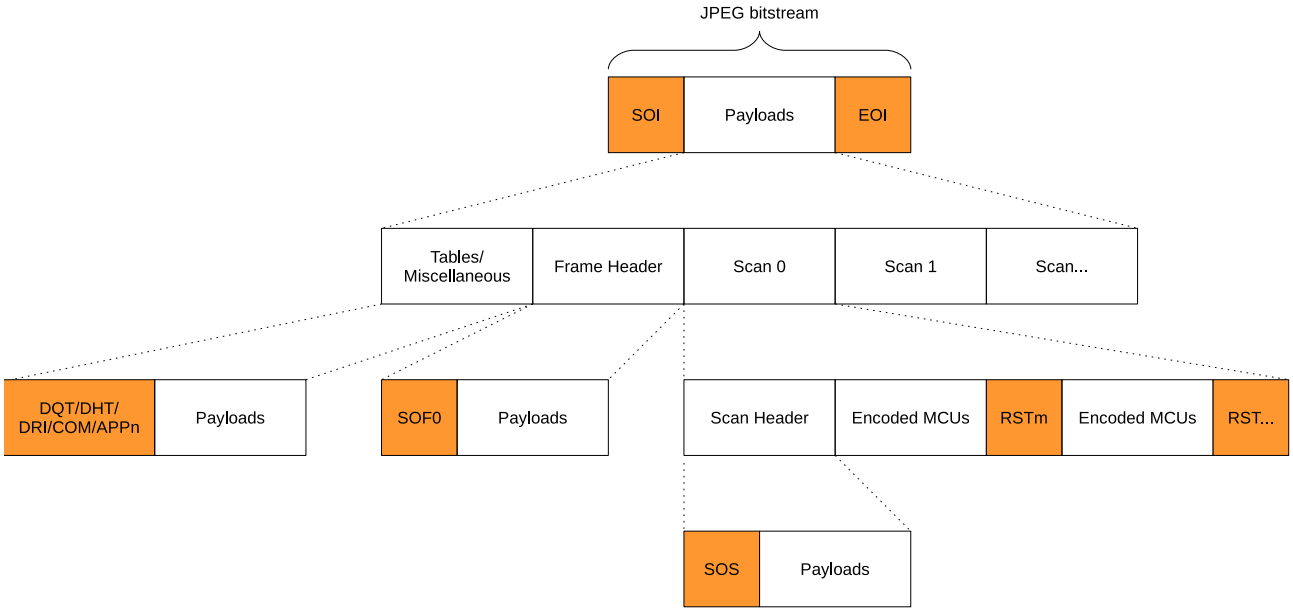


图 37.1-1. JPEG 码流格式

表 37.1-2 为码流中包含的各个标记码及其描述。

表 37.1-2. 标记码描述

代码	标记	描述
0xFFC0	SOF0	帧开始标记（基线 DCT）
0xFFC4	DHT	定义的哈夫曼表标记
0xFFDm	RSTm	恢复标记，模数为 8
0xFFD8	SOI	图像开始标记
0xFFD9	EOI	图像结束标记
0xFFDA	SOS	开始扫描标记
0xFFDB	DQT	定义的量化表标记
0xFFDD	DRI	定义的恢复间隔标记
0xFFEn	APPn	用于应用段的标记，“n” 为 0 到 F 区间内的值
0xFFFE	COM	注释标记

37.2 概述

ESP32-P4 的 JPEG 图像编解码器基于 JPEG 基线标准，该标准规定了编码处理需通过一次或多次扫描完成。一次扫描可以处理一个或多个 MCU。MCU 由数据单元序列构成，而数据单元序列取决于扫描图像分量的取样因子，也就是说不同图像格式的 MCU 可能由不同的数据单元序列构成，如表 37.2-1 所示。

表 37.2-1. 不同图像格式 MCU 包含的数据单位

图像格式	MCU[1]		
	亮度分量 Y	色度分量 U	色度分量 V
YUV444	1 × 1	1 × 1	1 × 1
YUV422	2 × 1	1 × 1	1 × 1

YUV420	2 × 2	1 × 1	1 × 1
GRAY	1 × 1	无	无

[1] 每个图像格式的 MCU 可以表示为每个分量水平方向的数据单元 × 垂直方向的数据单元。

在同一个时刻，JPEG 图像编解码器只能配置为编码器或解码器，即不能同时作为编码器和解码器工作：

- 作为编码器使用时，通过 2D DMA 获取待压缩图像，将编码后的码流发送给 2D DMA；
- 作为解码器使用时，通过 2D DMA 获取编码后的码流，将解码后的图像发送给 2D DMA。

JPEG 图像解码器模块的工作流程如图 37.2-1 所示。

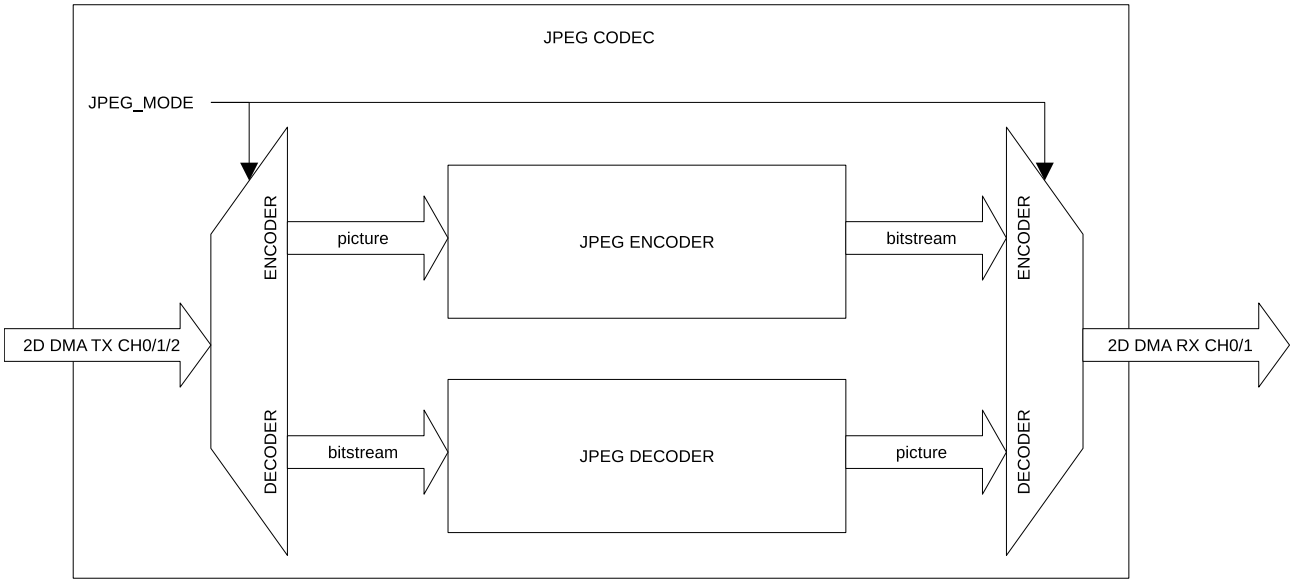


图 37.2-1. JPEG 图像编解码器的系统连接框图

JPEG 图像编解码器的功能时钟是 SYS_CLK。

37.3 主要特性

JPEG 图像编解码器作为编码器使用时，具有以下特性：

- 使用离散余弦变换算法
- 使用范式哈夫曼编码
- 原始输入图像格式支持 RGB888、RGB565、YUV422 和 GRAY
- 支持将 RGB888、RGB565 图像格式转换为 YUV444、YUV422 或 YUV420 格式的图像进行压缩（压缩仅支持 YUV444、YUV422 或 YUV420 格式）
- 支持 4 个 8 位或 16 位精度的可配置量化系数表
- 性能：
  - 静态图像压缩最大支持 4K 分辨率
  - 动态图像压缩最大支持 1080P@40fps，720P@70fps（不包括包头编码时间）

- 可自动填充零字节
- 可自动添加 EOI 标记

JPEG 图像编解码器作为解码器使用时，具有以下特性：

- 使用反离散余弦变换算法
- 使用哈夫曼解码
- 支持 YUV444、YUV422、YUV420、GRAY 图像格式的压缩码流解码
- 支持 4 个 8 位或 16 位精度的可配置量化系数表
- 支持 2 个 DC 和 2 个 AC 哈夫曼表
- 支持任意分辨率的图像解码，但输出的解码图像分辨率不同于输入图像格式：
  - YUV444、GRAY：输出的解码图像水平和垂直分辨率均为 8 的倍数，即 150 × 150 的图像输出分辨率为 152 × 152
  - YUV422：输出的解码图像水平分辨率为 16 的倍数，垂直分辨率为 8 的倍数，即 150 × 150 的图像输出分辨率为 160 × 152
  - YUV420：输出的解码图像水平和垂直分辨率均为 16 的倍数，即 150 × 150 的图像输出分辨率为 160 × 160
- 性能：
  - 静态图像解码最大支持 4K 分辨率
  - 动态图像解码最大支持 1080P@40fps，720P@70fps（不包括包头解析时间）

37.4 架构概述

JPEG 图像编解码器配置为编码器时，其架构如图 37.4-1 所示。

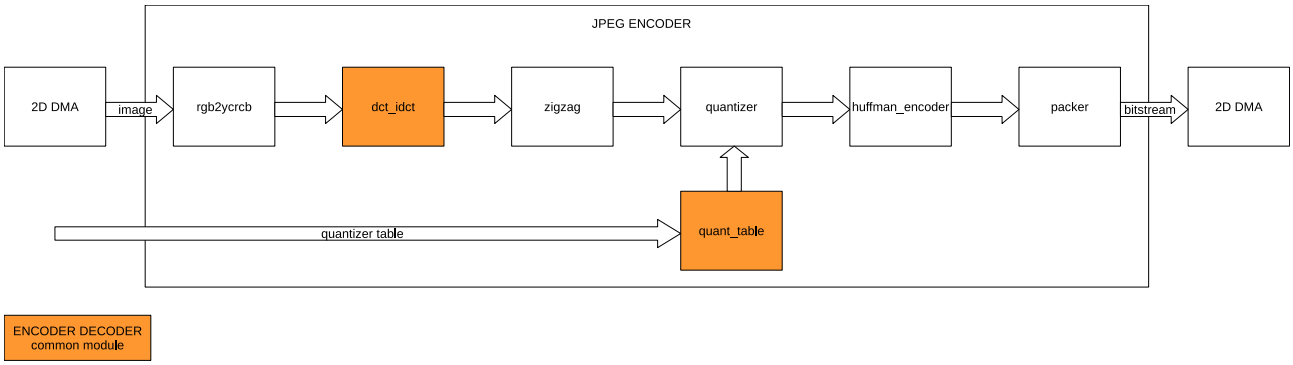


图 37.4-1. JPEG 编码器架构

- **rgb2ycrcb** 模块实现颜色空间转换功能，将原始图像转换为编解码器支持的压缩图像格式
- **dct_idct** 模块实现对数据单元的二维 DCT 变换
- **zigzag** 模块对 DCT 变换后的数据单元进行 Z 形扫描排序
- **quant_table** 模块存储软件下发的量化系数表
- **quantizer** 模块根据 **quant_table** 模块存储的量化系数表，对 Z 形排序后的数据进行量化

- **huffman_encoder** 模块对数据单元量化后结果进行哈夫曼编码
- **packer** 模块对哈夫曼编码后数据按照 32 位为一组进行打包，输出最终的编码码流

JPEG 图像编解码器配置为解码器时，其架构如图 37.4-2 所示。

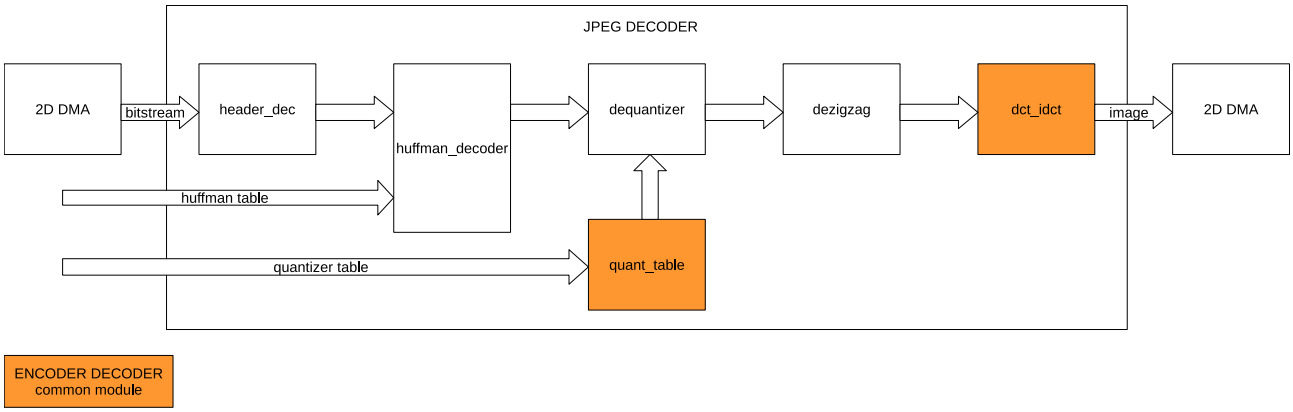


图 37.4-2. JPEG 解码器架构

- **header_dec** 模块实现码流的提取及 SOS 和 EOI 标记的解析，并控制解码流程
- **huffman_decoder** 模块存储软件下发的哈夫曼表，并实现对码流的哈夫曼解码
- **quant_table** 模块存储软件下发的量化系数表
- **dequantizer** 模块根据 **quant_table** 模块存储的量化系数表，对哈夫曼解码后的数据进行反量化
- **dezigzag** 模块对反量化后的数据进行反 Z 形扫描排序，组合成反量化后的数据单元
- **dct_idct** 模块对反量化后的数据单元进行二维反 DCT 变换，输出最终的解码后数据单元

由于 JPEG 图像编解码器不能同时作为编码器和解码器使用，编码器和解码器可以共用 **quant_table** 模块和 **dct_idct** 模块，节约硬件资源。

37.5 功能描述

JPEG 图像编解码器可配置为编码器或解码器使用。

37.5.1 JPEG 编码器

37.5.1.1 暂停

在执行其他优先级更高的任务时，JPEG 编码器可以暂停。该功能可通过 JPEG_PAUSE_EN 位控制。

- JPEG_PAUSE_EN 置 1 时，JPEG 编码器不再输出码流到 2D DMA 的接收通道
- JPEG_PAUSE_EN 清 0 时，JPEG 编码器恢复运行，正常输出码流

37.5.1.2 颜色空间转换

根据 JPEG 标准，JPEG 编码器支持的图像格式为 YUV444、YUV422、YUV420 和 GRAY。ESP32-P4 的 JPEG 编码器集成了颜色空间转换功能后，可以在编码前将 RGB888 和 RGB565 格式转换为编码器支持的上述图像格式。如此，JPEG 编码器便可以支持更多图像格式。

RGB 转换为 YUV 颜色空间的公式为：

$$Y = (0.299 \times R) + (0.587 \times G) + (0.114 \times B)$$
$$U = (-0.1687 \times R) + (-0.3313 \times G) + (0.5 \times B) + 128$$
$$V = (0.5 \times R) + (-0.4187 \times G) + (-0.0813 \times B) + 128$$

YUV 格式仅支持下采样 (downsampling)。例如，YUV444 可以保持原格式不变，或者降低采样数转换为 YUV422 或 YUV420；YUV422 可以保持原格式不变，或者降低采样数转换为 YUV420。

转换前的图像格式由格式和像素顺序表示。表 37.5-1 为颜色空间转换支持图像格式的表示方式：

表 37.5-1. 颜色空间转换支持的图像格式

JPEG_EXTD_COLOR_SPACE_EN	JPEG_COLOR_SPACE	JPEG_EXTD_COLOR_SPACE	格式[1]	JPEG_PIXEL_REV	像素顺序
0	0	N/A	RGB888	0	RGB
				1	BGR
	1		YUV422[2]	0	YUYV
				1	UYVY
	2		RGB565	0	RGB
				1	BGR
	3		GRAY[2]	N/A	YYY
1	N/A	0	YUV444	0	YUV
				1	VUY
		1	YUV420[2]	N/A	YYU/YYV[3]

[1] 可通过配置 JPEG_SAMPLE_SEL 字段选择编码器支持的转换格式：
- 0：YUV444
- 1：YUV422
- 2：YUV420

[2] 图像格式转换存在以下限制：
- YUV422：仅可保持不变或降低采样数至 YUV420。设置 JPEG_SAMPLE_SEL 为 0 无效。
- YUV420 或 GRAY：无法转换，不受 JPEG_SAMPLE_SEL 的配置控制。

[3] 像素排列方式按行交替：奇数行采用 YYU 模式，偶数行采用 YYV 模式。

37.5.1.3 可配置量化系数表

ESP32-P4 的 JPEG 编码器硬件支持四个 8 × 8 量化系数表，每个表可以通过软件配置 64 个系数值。

每个量化表的精度可通过 JPEG_QNR_PRECISION 配置：

- 0：低 8 位有效
- 1：低 16 位有效

量化表系数值的配置模式可通过 JPEG_QNR_FIFO_EN 选择：

- 0：非 FIFO 模式，系数写入特定地址
- 1：FIFO 模式，所有系数写入同一地址

表 37.5-2. 量化系数表中系数的位置编号（示例）

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
...							
56	57	58	59	60	61	62	63

不同模式下，量化系数表的配置方式会有所不同。注意，*n* (*n* = 0, 1, 2, 3) 表示每个量化系数表的编号。

- 非 FIFO 模式：量化系数表 *n* 中的所有量化系数必须按位置写入到特定地址，即 (JPEG 图像编解码器基地址 + *n* × 0x100 + 量化系数位置编号 × 0x4)，其中 量化系数位置编号 可以是 0、1、2、...、63（详见表 37.5-2）。在该模式下，由于每个系数都写入到特定地址，因此不需要按顺序配置这 64 个量化系数。
- FIFO 模式：量化系数表 *n* 中的 64 个量化系数全部写入相同的地址，即 JPEG_T*n*QNR_REG。在该模式下，由于所有系数都写入相同的地址，因此需要按顺序配置这 64 个量化系数。然后，硬件将自动按照表 37.5-2 中的顺序存储这 64 个量化系数。

如果量化系数表不足四个，则对应的量化表不配置。

要将特定量化系数表 *n* 用于亮度分量或色度分量，只需将相应的寄存器字段设置为 *n*：

- 亮度分量：JPEG_LQNR_TBL_SEL
- 色度分量：JPEG_CQNR_TBL_SEL

表 37.5-3 和表 37.5-4 分别为 JPEG 基线标准推荐用于亮度分量和色度分量的 8 位精度量化系数表。

表 37.5-3. 推荐用于亮度分量的 8 位精度量化系数表

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

表 37.5-4. 推荐用于色度分量的 8 位精度量化系数表

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

更多信息，请参阅 ITU T.81 Recommendation。调整量化系数表会改变压缩比（即未压缩图像大小与压缩图像大小之间的比率）和编码图像的质量（量化系数越大，压缩比越大，图像质量越差）。

37.5.1.4 填充零字节

JPEG 基线标准要求在哈夫曼编码段内避免使用任何标记，所有哈夫曼编码生成的 0xFF 字节后应“填充”零字节。

JPEG 编码器实现了该功能，可以在码流中哈夫曼编码段的 0xFF 字节后自动“填充”0x00 字节。

可通过 JPEG_FF_CHECK_EN 配置是否在 0xFF 字节后添加 0x00 字节：

- 0：不添加
- 1：添加（默认）

37.5.1.5 EOI 标记

图像结束 (EOI) 标记用于标记压缩图像码流的结束。EOI 标记代码为 0xFFD9。

JPEG 编码器可以在码流末尾自动添加 EOI 标记。

可通过 JPEG_TAILER_EN 配置是否添加 EOF 标记：

- 0：不添加
- 1：添加

37.5.2 JPEG 解码器

码流中的信息必须先由软件解析，然后通过 APB 配置总线发送给硬件。注意，硬件不支持解析除 SOS、RST 和 EOI 标记段之外的任何标记段。

37.5.2.1 多个色度分量

帧头包含有关图像色度分量数量、每个分量的 ID 以及每个分量水平和垂直方向采样因子的信息。

JPEG 解码器的软件必须解析这类信息，并将其传递给响应寄存器 JPEG_COMPONENT_NUM、JPEG_C*n*_ID、JPEG_C*n*_X_FACTOR 和 JPEG_C*n*_Y_FACTOR。此处 *n* 可以为 0、1、2，表示硬件支持最多 3 个色度分量。注意，如果分量不足 3 个，不需要配置其余寄存器。

硬件会参考每个色度分量寄存器 JPEG_C*n*_ID 的值，检查其是否与 Scan 头信息中的 ID 一致。采样因子表示一个 MCU 在水平和垂直方向上包含该色度分量的数据单元数。因此，根据 JPEG_COMPONENT_NUM、JPEG_C*n*_X_FACTOR 和 JPEG_C*n*_Y_FACTOR 字段的值，硬件可以获得各个色度分量的比例，即待解码图像的格式。详见表 37.5-5。

表 37.5-5. 采样因子和图像格式

JPEG_COMPONENT_NUM	JPEG_C*n*_X/Y_FACTOR	待解码图像的格式
0	-	无效
1	C0_X_FACTOR 为 1 C0_Y_FACTOR 为 1	GRAY
2	-	无效
	C0_X_FACTOR 为 1 C0_Y_FACTOR 为 1 C1_X_FACTOR 为 1 C1_Y_FACTOR 为 1	YUV444

	C2_X_FACTOR 为 1 C2_Y_FACTOR 为 1	
	C0_X_FACTOR 为 2 C0_Y_FACTOR 为 1 C1_X_FACTOR 为 1 C1_Y_FACTOR 为 1 C2_X_FACTOR 为 1 C2_Y_FACTOR 为 1	YUV422
	C0_X_FACTOR 为 2 C0_Y_FACTOR 为 2 C1_X_FACTOR 为 1 C1_Y_FACTOR 为 1 C2_X_FACTOR 为 1 C2_Y_FACTOR 为 1	YUV420

无论待解码图像是什么格式，解码器只支持一次扫描所有分量。解码后，图像会发送到 2D DMA，有关每种解码图像格式的布局请参考章节 8 *2D-DMA 控制器 (2D-DMA)*。

37.5.2.2 解析 RST 标记

Scan 段比特流中可能包含恢复 (RST) 标记。RST 标记的代码为 0xFFD*x* (*x* = 0, 1, ..., 7)。

JPEG 解码器的硬件可以解析 RST 标记，检查两个 RST 标记之间实际的 MCU 个数是否与 Tables/Miscellaneous 段中 DRI 定义的恢复间隔相同。

首先，软件会解析 DRI 段，并将定义的恢复间隔传递给 JPEG_RESTART_INTERVAL 字段。之后，硬件每解码 JPEG_RESTART_INTERVAL 个 MCU，就会检查后续码流是否包含 RST 标记。

如果 JPEG_RESTART_INTERVAL 配置为 0，则码流中不包含 RST 标记，此时硬件将不检查 RST 标记。

37.5.2.3 可配置量化系数表

ESP32-P4 JPEG 解码器同样支持 4 个可配置量化系数表。由于 JPEG 图像编解码器不能同时用作编码器和解码器，解码器可以和编码器共用相同的 4 个可配置系数表，从而节省硬件资源。

解码器量化系数表的配置与编码器的（详见章节 37.5.1.3）类似，但有以下不同：

- 软件需要解析 Tables/Miscellaneous 段中的 DQT 段，获取解码器的量化系数表。推荐的解码器量化系数表与编码器相同，如表 37.5-3 和表 37.5-4 所示。
- 量化系数表*n* 的精度可以通过 JPEG_T*n*_DQT_INFO 字段的高 4 位配置：
  - 0：低 8 位有效
  - 1：低 16 位有效

  精度也是从 Tables/Miscellaneous 段中的 DQT 段获取。
- 每个色度分量可选择不同的量化系数表。JPEG_T*n*_DQT_INFO 字段的低 4 位配置量化系数表的 ID，JPEG_C*n*_DQT_TBL_SEL 字段配置每个色度分量选择的量化系数表 ID。每个量化系数表的 ID 也从 Tables/Miscellaneous 段中的 DQT 段获取。

37.5.2.4 可配置哈夫曼表

ESP32-P4 支持最多 2 个 DC 哈夫曼表和 2 个 AC 哈夫曼表。可在限制范围内配置所需数量的哈夫曼表。

每个哈夫曼表可以包含三类信息，这三类信息可从 Tables/Miscellaneous 段中的 DHT 段获取：

- 长度为 1 到 16 位的码字数量
- 解码后码字对应的符号
  - DC 表：最多 16 个符号
  - AC 表：最多 256 个符号
- 长度为 1 到 16 位的最小码字

DC 和 AC 哈夫曼表都可以通过 JPEG_DHT_FIFO_EN 选择配置模式：

- 0：非 FIFO 模式
- 1：FIFO 模式

不同模式下，哈夫曼表的配置方式会有所不同。以下是 DC 表*n* (*n* = 0, 1) 的配置方式：

- 非 FIFO 模式：DC*n* 哈夫曼表的不同信息必须按信息类型写入到特定地址。
  - 长度为 1 到 16 位的码字数量：JPEG 图像编解码器基地址 + 0x500 + *n* × 0x80 + (码字长度 - 1) × 0x4，其中 码字长度 = 1, 2, ..., 16。
  - 解码后码字对应的符号：JPEG 图像编解码器基地址 + 0x600 + *n* × 0x40 + 符号位置 × 0x4，其中 符号位置 = 0, 1, ..., 15。
  - 长度为 1 到 16 位的最小码字：JPEG 图像编解码器基地址 + 0xF00 + *n* × 0x80 + (码字长度 - 1) × 0x4，其中 码字长度 = 1, 2, ..., 16。

  在该模式下，由于信息按类型都写入到特定地址，因此不需要按顺序配置。
- FIFO 模式：DC*n* 哈夫曼表同一类型的信息全部写入到同个地址。
  - 长度为 1 到 16 位的码字数量：JPEG_DHT_TOTLEN_DC*n*_REG。
  - 解码后码字对应的所有符号（DC 表最多 16 个符号）：JPEG_DHT_VAL_DC*n*_REG。
  - 长度为 1 到 16 位的最小码字：JPEG_DHT_CODEMIN_DC*n*_REG。

  在该模式下，由于同一类型的信息全部写入到同个地址，因此需要按顺序配置同一类型的信息。之后，硬件将自动按顺序存储这些信息。

AC*n* (*n* = 0, 1) 表的配置方式与 DC*n* (*n* = 0, 1) 表类似。以下是 AC*n* (*n* = 0, 1) 的配置方式：

- 非 FIFO 模式：AC*n* 哈夫曼表的不同信息必须按信息类型写入到特定地址。
  - 长度为 1 到 16 位的码字数量：JPEG 图像编解码器基地址 + 0x540 + *n* × 0x80 + (码字长度 - 1) × 0x4，其中 码字长度 = 1, 2, ..., 16。
  - 解码后码字对应的符号：JPEG 图像编解码器基地址 + 0x680 + *n* × 0x400 + 符号位置 × 0x4，其中 符号位置 = 0, 1, ..., 256。
  - 长度为 1 到 16 位的最小码字：JPEG 图像编解码器基地址 + 0xF40 + *n* × 0x80 + (码字长度 - 1) × 0x4，其中 码字长度 = 1, 2, ..., 16。

  在该模式下，由于信息按类型都写入到特定地址，因此不需要按顺序配置。

- FIFO 模式：AC*n* 哈夫曼表同一类型的信息全部写入到同个地址。
  - 长度为 1 到 16 位的码字数量：JPEG_DHT_TOTLEN_AC*n*_REG。
  - 解码后码字对应的所有符号（AC 表最多 256 个符号）：JPEG_DHT_VAL_AC*n*_REG。
  - 长度为 1 到 16 位的最小码字：JPEG_DHT_CODEMIN_AC*n*_REG。

  在该模式下，由于同一类型的信息全部写入到同个地址，因此需要按顺序配置同一类型的信息。之后，硬件将自动按顺序存储这些信息。

长度为 1 到 16 位的最小码字应该向左偏移到 16 位配置值的高位。如果没有码字长度，则配置值为 0xFFFF。例如，如果最小码字为 0x7C，则 16 位配置值应为 0x7C00。

JPEG 基线标准推荐的 DC 和 AC 哈夫曼表，可在 ITU T.81 Recommendation 中找到。

软件可以通过解析 Tables/Miscellaneous 段中的 DHT 段来获取 DC*n* 和 AC*n* 哈夫曼表的 ID，然后将 ID 传递给相应字段 JPEG_DC*n*_DHT_ID 和 JPEG_AC*n*_DHT_ID。

硬件会自动使用与 JPEG_DC*n*_DHT_ID 和 JPEG_AC*n*_DHT_ID ID 值相同的 DC*n* 和 AC*n* 哈夫曼表。

37.5.2.5 超时检测

JPEG 图像编解码器用作解码器时还支持超时检测。以下情况出现时，硬件会触发超时中断 JPEG_DECODE_TIMEOUT_INT：

- 解码器停止读取码流超过 $(2^{JPEG_DECODE_TIMEOUT_THRES} - 1)$ 个时钟周期
- 解码图像所用的时钟周期数超过 $(2^{JPEG_DECODE_TIMEOUT_THRES} - 1)$ 的值

超时中断触发后，解码器会根据 JPEG_DECODE_TIMEOUT_TASK_SEL 的配置复位：

- 0：硬件不会执行任何操作，需要软件配置 JPEG_SOFT_RST 或 JPEG_FSM_RST 来执行复位操作。
- 1：硬件将自动复位解码器，从而退出超时状态并进入初始状态。

37.6 中断

ESP32-P4 的 JPEG 图像编解码器可以生成 JPEG_INTR 中断信号并发送给 中断矩阵。

JPEG_INTR 中断信号由下列中断源生成：

- JPEG_DONE_INT：当一帧图像编码或解码结束时触发。
- JPEG_RLE_PARALLEL_ERR_INT：当游程编码出错时触发。
- JPEG_CID_ERR_INT：当解码得到的各个分量 ID 和软件配置的各个分量 ID 不同时触发。
- JPEG_C_DHT_DC_ID_ERR_INT：当解码得到的各个分量 DC 哈夫曼表 ID 不是软件配置的 DC0 哈夫曼表 ID 或 DC1 哈夫曼表 ID 时触发。
- JPEG_C_DHT_AC_ID_ERR_INT：当解码得到的各个分量 AC 哈夫曼表 ID 不是软件配置的 AC0 哈夫曼表 ID 或 AC1 哈夫曼表 ID 时触发。
- JPEG_C_DQT_ID_ERR_INT：当解码得到的各个分量量化表 ID 和软件配置的各个分量量化表 ID 不同时触发。
- JPEG_RST_UXP_ERR_INT：当软件配置的 JPEG_RESTART_INTERVAL 为 0 但解码器解析到 RST 标记时触发。

- JPEG_RST_CHECK_NONE_ERR_INT：当软件配置的 JPEG_RESTART_INTERVAL 非 0 但解码器解析不到 RST 标记时触发。
- JPEG_RST_CHECK_POS_ERR_INT：当两个解析到的 RST 标记之间的 MCU 数不等于软件配置的 JPEG_RESTART_INTERVAL 时触发。
- JPEG_OUT_EOF_INT：当从 2D DMA 的 TX 通道读取到 EOF 标志时触发。
- JPEG_SR_COLOR_MODE_ERR_INT：该中断无效。
- JPEG_DCT_DONE_INT：当一个数据单元 DCT 或 IDCT 计算完成时触发。
- JPEG_BS_LAST_BLOCK_EOF_INT：当最后一个数据单元编码完成时触发。
- JPEG_SCAN_CHECK_NONE_ERR_INT：当一帧图像有多个待解码 Scan 并且在任意 Scan 头信息中（JPEG_SOS_CHECK_BYTE_NUM + 1）个字节都未解析到 SOS 标记时触发。
- JPEG_SCAN_CHECK_POS_ERR_INT：当解码器解析到的 Scan 头信息位置错误时触发。
- JPEG_UXP_DET_INT：当解码器解析到的标记不是硬件支持的标记时触发。
- JPEG_EN_FRAME_EOF_ERR_INT：当从 2D DMA 读取到的待编码数据单元数量小于根据软件配置的图像分辨率计算出的数据单元数量时触发。
- JPEG_EN_FRAME_EOF_LACK_INT：当从 2D DMA 读取完一帧待编码图像但未读取到 EOF 标志时触发。
- JPEG_DE_FRAME_EOF_ERR_INT：当解码完一帧图像得到的数据单元数量和根据软件配置的图像分辨率计算出的数据单元数量不一致时触发。
- JPEG_DE_FRAME_EOF_LACK_INT：当从 2D DMA 读取完一帧图像的码流但未读到 EOF 标志或 EOI 标记时触发。
- JPEG_SOS_UNMATCH_ERR_INT：当解码器解析到的 Scan 头信息中的分量数为 0 或解析到的 Scan 头信息中的头长度与实际不符时触发。
- JPEG_MARKER_ERR_FST_SCAN_INT：当解码器解析到的第一个 Scan 头信息中有错时触发，错误包括触发 JPEG_CID_ERR_INT、JPEG_C_DHT_DC_ID_ERR_INT、JPEG_C_DHT_AC_ID_ERR_INT、JPEG_C_DQT_ID_ERR_INT 和 JPEG_SOS_UNMATCH_ERR_INT 的错误。
- JPEG_MARKER_ERR_OTHER_SCAN_INT：当解码器解析到的非第一个 Scan 头信息中有错时触发，错误类型和触发 JPEG_MARKER_ERR_FST_SCAN_INT 的错误相同。
- JPEG_UNDET_INT：当从 2D DMA 读取完一帧图像的码流但未读到过 SOS 标记时触发。
- JPEG_DECODE_TIMEOUT_INT：当解码器超时时触发。

上述中断源仅在对应中断使能寄存器 中断源名称_ENA 置 1 时才能触发。向 中断源名称_CLR 写 1 可清除中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 37.8 寄存器列表。

37.7 编程流程

37.7.1 JPEG 编码器

JPEG 图像编解码器作为编码器使用时，配置流程为：

1. 激活 JPEG 图像编解码器
   - 将 HP_SYS_CLKRST_JPEG_SYS_CLK_EN 置 1，使能 JPEG 图像编解码器时钟。
   - 将 HP_SYS_CLKRST_RST_EN_JPEG 置 0，释放 JPEG 图像编解码器系统级复位。
2. 配置 JPEG 图像编解码器
   - 将 JPEG_SOFT_RST 置 1，复位 JPEG 图像编解码器，再将 JPEG_SOFT_RST 置 0，释放 JPEG 图像编解码器复位。
   - 将 JPEG_MODE 置 0，把 JPEG 图像编解码器配置为编码器。
   - 配置以下字段，选择原始图像格式：
     - 格式：
       * JPEG_COLOR_SPACE
       * JPEG_EXTD_COLOR_SPACE_EN
       * JPEG_EXTD_COLOR_SPACE
     - 像素顺序：JPEG_PIXEL_REV

     详见表 37.5-1。
   - 配置 JPEG_TAILER_EN，选择是否在码流末尾添加 EOI 标记 (0xFFD9)：
     - 0：禁用
     - 1：使能
   - 配置 JPEG_SAMPLE_SEL 选择待压缩图像格式：
     - 0：YUV444
     - 1：YUV422
     - 2：YUV420

     注意：该字段仅在原始图像格式为 RGB、YUV444 或 YUV422 时有效。详见表 37.5-1。
   - 将 JPEG_HA 配置为原始图像宽度，JPEG_VA 配置为原始图像高度。
   - 参考章节 37.5.1.3 的说明配置量化系数表。
3. 配置 2D DMA（具体配置方法和相关概念介绍，请参考章节 8 *2D-DMA 控制器 (2D-DMA)*。）

   (a) 配置 2D DMA TX 通道的链表：
   - 将 2DEN 置 1，使能 2D 功能。
   - 将 Buffer address pointer 配置为存储原始图像的起始地址。
   - 将 pbyte 选择原始图像一个像素的字节数。

- 将 HA 配置为原始图像宽度，VA 配置为原始图像高度。
- 将 hb 配置为一个图像块水平方向的像素个数，vb 配置为一个图像块垂直方向的像素个数。
- 将 mod 配置为 1，表示连续读取 hb × vb 大小图像块。
- 将 (X，Y) 配置为起始图像块的第一个像素坐标。根据原始图像格式和待压缩图像的格式，hb 和 vb 有一定限制，如表37.7-1所示：

表 37.7-1. JPEG 编码器 hb 和 vb 配置

原始图像格式	待压缩图像格式	hb	vb
RGB/YUV444	YUV444	8 的倍数	8 的倍数
	YUV422	16 的倍数	8 的倍数
	YUV420	16 的倍数	16 的倍数
YUV422	YUV422	16 的倍数	8 的倍数
	YUV420	16 的倍数	16
YUV420	YUV420	16 的倍数	16
GRAY	GRAY	8 的倍数	8 的倍数

(b) 配置 2D DMA RX 通道的链表：

- 将 2DEN 置 1，使能 1D 功能。
- 将 Buffer address pointer 配置为存储编码码流的起始地址。
- 将 size 配置为存储编码码流空间的字节数。
- 将 mod 配置为 0，表示一次读取 hb × vb 大小图像块。

(c) 连接 JPEG 图像编解码器和 2D DMA 通道：

- 将 DMA2D_OUT_PERI_SEL_CH*n* 配置为 0，连接 JPEG 图像编解码器和 TX*n*；
- 将 DMA2D_IN_PERI_SEL_CH*n* 配置为 0，连接 JPEG 图像编解码器和 RX*n*。

(d) 复位 2D DMA 通道：

- 2D DMA TX 通道*n*：将 DMA2D_OUT_RST_CH*n* 置 1，复位 2D DMA TX 通道*n*，再将 DMA2D_OUT_RST_CH*n* 置 0，释放 2D DMA TX 通道*n* 复位。
- 2D DMA RX 通道*n*：将 DMA2D_IN_RST_CH*n* 置 1，复位 2D DMA RX 通道*n*，再将 DMA2D_IN_RST_CH*n* 置 0，释放 2D DMA RX 通道*n* 复位。

(e) 选择 2D DMA 通道的突发长度：

- 配置 DMA2D_OUT_MEM_BURST_LENGTH_CH*n*，选择 2D DMA TX 通道*n* 的突发长度。
- 配置 DMA2D_IN_MEM_BURST_LENGTH_CH*n*，选择 2D DMA RX 通道*n* 的突发长度。

(f) 根据原始图像的格式和待压缩图像的格式，配置 DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* 选择 2D DMA 输出图像块大小。详见表 37.7-2：

- 0：8 × 8
- 1：16 × 8
- 2：16 × 16

表 37.7-2. DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n* 配置

原始图像格式	待压缩图像格式	DMA2D_OUT_MACRO_BLOCK_SIZE_CH*n*
RGB/YUV444	YUV444	0
	YUV422	1
	YUV420	2
YUV422	YUV422	1
	YUV420	2
YUV420	YUV420	2
GRAY	GRAY	0

(g) 将 DMA2D_OUT_REORDER_EN_CH0 置 1 使能重排 (Reorder) 功能，提高带宽利用率。此功能仅适用于 2D DMA TX 通道 0，不适用于任何其他 TX 通道。

(h) 配置 2D DMA 通道链表地址：

- 2D DMA TX 通道：DMA2D_OUTLINK_ADDR_CH*n*
- 2D DMA RX 通道：DMA2D_INLINK_ADDR_CH*n*

4. 使能中断源：
   - DMA2D_OUT_INT_ENA_CH*n*_REG
   - DMA2D_IN_INT_ENA_CH*n*_REG
   - JPEG_INT_ENA_REG
5. 开启 2D DMA 通道传输：
   - 2D DMA TX 通道：将 DMA2D_OUTLINK_START_CH*n* 先置 1 再置 0
   - 2D DMA RX 通道：将 DMA2D_INLINK_START_CH*n* 先置 1 再置 0
6. 配置 JPEG_JPEG_START 字段，让 JPEG 图像编解码器开始编码。
7. 等待 DMA2D_IN_SUC_EOF_CH*n*_INT 变为 1，表示已将编码后的码流完全写入存储器。

37.7.2 JPEG 解码器

JPEG 图像编解码器作为解码器使用时，配置流程为：

1. 激活 JPEG 图像编解码器
   - 将 HP_SYS_CLKRST_JPEG_SYS_CLK_EN 置 1，使能 JPEG 图像编解码器时钟。
   - 将 HP_SYS_CLKRST_RST_EN_JPEG 置 0，释放 JPEG 图像编解码器系统级复位。
2. 配置 JPEG 图像编解码器
   - 将 JPEG_SOFT_RST 置 1，复位 JPEG 图像编解码器，再将 JPEG_SOFT_RST 置 0，释放 JPEG 图像编解码器复位。
   - 将 JPEG_MODE 置 1，把 JPEG 图像编解码器配置为解码器。
   - 参考章节 37.5.2.2 的说明，根据需要使能 RST 标记检测。
   - 参考章节 37.5.2.3 的说明配置量化系数表。

- 参考章节 37.5.2.4 的说明配置哈夫曼表。
- 配置解码图像的宽度和高度：
  - 宽度：将解码图像的宽度写入 JPEG_HA.
  - 高度：将解码图像的高度写入 JPEG_VA.
  - 根据不同的解码图像格式，JPEG_HA 和 JPEG_VA 必须满足表 37.7-3 的要求。

表 37.7-3. JPEG 解码器 JPEG_HA、JPEG_VA 的配置

解码图像格式	JPEG_HA	JPEG_VA
YUV444	8 的倍数	8
YUV422	16 的倍数	8
YUV420	16 的倍数	16
GRAY	8 的倍数	8

3. 配置 2D DMA（具体配置方法和相关概念介绍，请参考章节 *8 2D-DMA 控制器 (2D-DMA)*。）

(a) 配置 2D DMA TX 通道的链表：

- 将 2DEN 置 0，使能 1D 功能。
- 将 Buffer address pointer 配置为存储待解码码流的起始地址。
- 将 size 配置为存储待解码码流空间的字节数。
- 将 mod 配置为 0，表示一次读取 hb × vb 大小图像块。

(b) 配置 2D DMA RX 通道的链表：

- 将 2DEN 置 1，使能 2D 功能。
- 将 Buffer address pointer 配置为存储解码图像的起始地址。
- 将 pbyte 配置为解码并经过 2D DMA 颜色空间转换的图像一个像素的字节数。
- 将 HA 配置为输出解码图像宽度，VA 配置为输出解码图像高度，hb 配置为一个图像块水平方向的像素个数，vb 配置为一个图像块垂直方向的像素个数。
- 将 mod 配置为 1，表示连续读取 hb × vb 大小图像块。
- 将 (X，Y) 配置为起始图像块的第一个像素坐标。根据原始图像格式和待压缩图像的格式，hb 和 vb 有一定限制，如表 37.7-1 所示：

表 37.7-4. JPEG 解码器 hb 和 vb 配置

解码图像格式	hb	vb
YUV444	8 的倍数	8 的倍数
YUV422	16 的倍数	8 的倍数
YUV420	16 的倍数	16 的倍数
GRAY	8 的倍数	8 的倍数

(c) 连接 JPEG 图像编解码器和 2D DMA 通道：

- 将 DMA2D_OUT_PERI_SEL_CH*n* 配置为 0，连接 JPEG 图像编解码器和 TX 通道*n*；

- 将 DMA2D_IN_PERI_SEL_CH*n* 配置为 0，连接 JPEG 图像编解码器和 RX 通道*n*。

(d) 复位 2D DMA 通道：

- 2D DMA TX 通道*n*：将 DMA2D_OUT_RST_CH*n* 置 1，复位 2D DMA TX 通道*n*，再将 DMA2D_OUT_RST_CH*n* 置 0，释放 2D DMA TX 通道*n* 复位。
- 2D DMA RX 通道*n*：将DMA2D_IN_RST_CH*n* 置 1，复位 2D DMA RX 通道*n*，再将 DMA2D_IN_RST_CH*n* 置 0，释放 2D DMA RX 通道*n* 复位。

(e) 选择 2D DMA 通道的突发长度：

- 配置 DMA2D_OUT_MEM_BURST_LENGTH_CH*n*，选择 2D DMA TX 通道*n* 的突发长度。
- 配置 DMA2D_IN_MEM_BURST_LENGTH_CH*n*，选择 2D DMA RX 通道*n* 的突发长度。

(f) 根据解码图像的格式，配置 DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* 选择 2D DMA 接收图像块大小。详见表 37.7-5：

- 0：8 × 8
- 1：16 × 8
- 2：16 × 16

表 37.7-5. DMA2D_IN_MACRO_BLOCK_SIZE_CH*n* 配置

解码图像格式	DMA2D_IN_MACRO_BLOCK_SIZE_CH*n*
YUV444	0
YUV422	1
YUV420	2
GRAY	0

(g) 将 DMA2D_IN_REORDER_EN_CH0 置 1 使能重排 (Reorder) 功能，提高带宽利用率。此功能仅适用于 2D DMA RX 通道 0，不适用于任何其他 RX 通道。

(h) 配置 DMA2D_IN_SCRAMBLE_SEL_PRE_CH0，选择在 2D DMA 颜色空间转换前如何调换解码图像的像素顺序：

- 0：BYTE2-1-0
- 1：BYTE2-0-1
- 2：BYTE1-0-2
- 3：BYTE1-2-0
- 4：BYTE0-2-1
- 5：BYTE0-1-2

字段配置为 5 表示调换像素顺序，例如将 JPEG 解码器输出的 YUV444 调换为 VUY444。请注意，该功能仅适用于 2D DMA RX 通道 0，不适用于其他 RX 通道。

(i) 配置 DMA2D_IN_COLOR_INPUT_SEL_CH0、DMA2D_IN_COLOR_3B_PROC_EN_CH0 和 DMA2D_IN_COLOR_OUTPUT_SEL_CH0 选择颜色转换的方式。详见表 37.7-6：

表 37.7-6. 2D DMA 颜色空间转换配置

颜色空间转换方式	INPUT_SEL_CH0	3B_PROC_EN_CH0	OUTPUT_SEL_CH0
YUV420 → RGB	0	1	1
YUV422 → RGB	0	1	1
YUV444 → RGB	2	1	1
YUV444（仅调换像素顺序）	2	0	1
不转换颜色空间	7	N/A	N/A

请注意，该功能仅适用于 2D DMA RX 通道 0，不适用于其他 RX 通道。

(j) 配置颜色空间转换公式所需的参数，包括 DMA2D_IN_COLOR_PARAM_H0_CH0、DMA2D_IN_COLOR_PARAM_H1_CH0、DMA2D_IN_COLOR_PARAM_M0_CH0、DMA2D_IN_COLOR_PARAM_M1_CH0、DMA2D_IN_COLOR_PARAM_L0_CH0 和 DMA2D_IN_COLOR_PARAM_L1_CH0。请注意，该功能仅适用于 2D DMA RX 通道 0，不适用于其他 RX 通道。

(k) 配置 DMA2D_IN_SCRAMBLE_SEL_POST_CH0，选择在 2D DMA 颜色空间转换后如何调换图像的像素顺序：

- 0：BYTE2-1-0
- 1：BYTE2-0-1
- 2：BYTE1-0-2
- 3：BYTE1-2-0
- 4：BYTE0-2-1
- 5：BYTE0-1-2

字段配置为 5 表示翻转像素顺序，例如将 2D DMA 颜色空间转换后的 RGB 翻转为 BGR。请注意，该功能仅适用于 2D DMA RX 通道 0，不适用于其他 RX 通道。

(l) 配置 2D DMA 通道链表地址：

- 2D DMA TX 通道：DMA2D_OUTLINK_ADDR_CH*n*
- 2D DMA RX 通道：DMA2D_INLINK_ADDR_CH*n*

4. 使能中断源：
   - DMA2D_OUT_INT_ENA_CH*n*_REG
   - DMA2D_IN_INT_ENA_CH*n*_REG
   - JPEG_INT_ENA_REG
5. 开启 2D DMA 通道传输：
   - 2D DMA TX 通道：将 DMA2D_OUTLINK_START_CH*n* 先置 1 再置 0
   - 2D DMA RX 通道：将 DMA2D_INLINK_START_CH*n* 先置 1 再置 0
6. 配置 JPEG_JPEG_START 字段，让 JPEG 图像编解码器开始解码。
7. 等待 DMA2D_IN_SUC_EOF_CH*n*_INT 变为 1，表示解码图像已完全写入存储器。

37.7.3 复位

根据实际场景，可在编码或解码过程中 **复位** JPEG 图像编解码器的所有模块或部分模块：

- 复位 JPEG 图像编解码器的所有模块，包括所有 RAM、FIFO 和状态机：将 JPEG_SOFT_RST 置 1。此时寄存器配置不会复位。
- 复位 JPEG 图像编解码器的所有 RAM 和 FIFO，比如在更改量化系数表或哈夫曼表时：将 JPEG_FIFO_RST 置 1。
- 复位 JPEG 图像编解码器的所有状态机，比如终止当前编码或解码流程时：将 JPEG_FSM_RST 置 1。

37.8 寄存器列表

本小节的所有地址均为相对于 JPEG 图像编解码器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

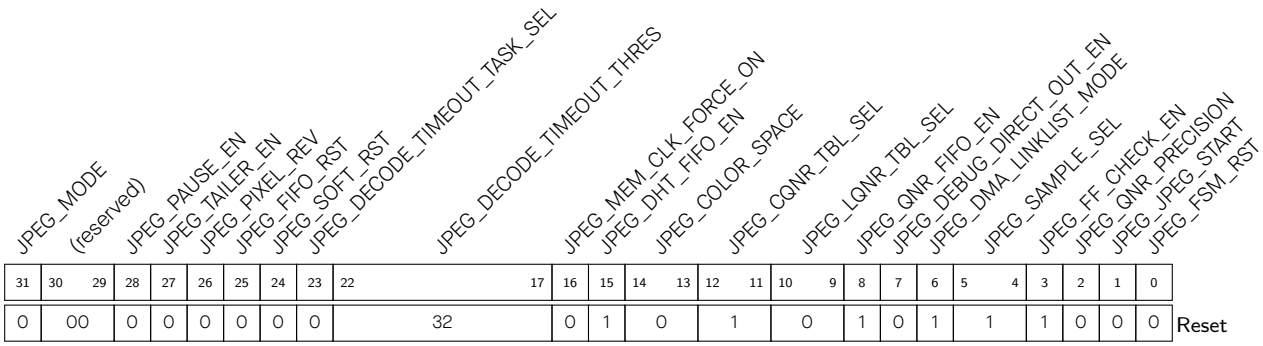
名称	描述	地址	访问
控制和配置寄存器			
JPEG_CONFIG_REG	系统级控制和配置寄存器	0x0000	varies
JPEG_DQT_INFO_REG	量化系数表信息寄存器	0x0004	R/W
JPEG_PIC_SIZE_REG	图像大小寄存器	0x0008	R/W
JPEG_EXTD_CONFIG_REG	颜色空间转换扩展格式配置寄存器	0x000C	R/W
JPEG_T0QNR_REG	量化系数表 0 寄存器	0x0010	HRO
JPEG_T1QNR_REG	量化系数表 1 寄存器	0x0014	HRO
JPEG_T2QNR_REG	量化系数表 2 寄存器	0x0018	HRO
JPEG_T3QNR_REG	量化系数表 3 寄存器	0x001C	HRO
JPEG_DECODE_CONF_REG	解码器配置寄存器	0x0020	varies
JPEG_C0_REG	颜色分量 0 寄存器	0x0024	R/W
JPEG_C1_REG	颜色分量 1 寄存器	0x0028	R/W
JPEG_C2_REG	颜色分量 2 寄存器	0x002C	R/W
JPEG_C3_REG	颜色分量 3 寄存器	0x0030	R/W
JPEG_DHT_INFO_REG	哈夫曼表信息寄存器	0x0034	R/W
JPEG_DHT_TOTLEN_DC0_REG	DC0 哈夫曼表码字长度寄存器	0x0058	HRO
JPEG_DHT_VAL_DC0_REG	DC0 哈夫曼表符号寄存器	0x005C	HRO
JPEG_DHT_TOTLEN_AC0_REG	AC0 哈夫曼表码字长度寄存器	0x0060	HRO
JPEG_DHT_VAL_AC0_REG	AC0 哈夫曼表符号寄存器	0x0064	HRO
JPEG_DHT_TOTLEN_DC1_REG	DC1 哈夫曼表码字长度寄存器	0x0068	HRO
JPEG_DHT_VAL_DC1_REG	DC1 哈夫曼表符号寄存器	0x006C	HRO
JPEG_DHT_TOTLEN_AC1_REG	AC1 哈夫曼表码字长度寄存器	0x0070	HRO
JPEG_DHT_VAL_AC1_REG	AC1 哈夫曼表符号寄存器	0x0074	HRO
JPEG_DHT_CODEMIN_DC0_REG	DC0 哈夫曼表最小码字寄存器	0x0078	HRO
JPEG_DHT_CODEMIN_AC0_REG	AC0 哈夫曼表最小码字寄存器	0x007C	HRO
JPEG_DHT_CODEMIN_DC1_REG	DC1 哈夫曼表最小码字寄存器	0x0080	HRO
JPEG_DHT_CODEMIN_AC1_REG	AC1 哈夫曼表最小码字寄存器	0x0084	HRO
JPEG_SYS_REG	系统配置寄存器	0x00F8	R/W
中断寄存器			
JPEG_INT_RAW_REG	中断原始状态寄存器	0x0038	R/WTC/SS
JPEG_INT_ENA_REG	中断使能寄存器	0x003C	R/W
JPEG_INT_ST_REG	中断屏蔽状态寄存器	0x0040	RO
JPEG_INT_CLR_REG	中断清除寄存器	0x0044	WT
版本寄存器			
JPEG_VERSION_REG	版本控制寄存器	0x00FC	R/W

37.9 寄存器

本小节的所有地址均为相对于 JPEG 图像编解码器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 37.1. JPEG_CONFIG_REG (0x0000)



31	30 29	28	27	26	25	24	23	22 17	16	15	14 13	12 11	10 9	8	7	6	5 4	3	2	1	0	
0	00	0	0	0	0	0	0	32	0	1	0	1	0	1	0	1	1	1	0	0	0	Reset

JPEG_FSM_RST 配置是否复位编解码器状态机。

0：无效值。没有作用

1：复位状态机

(WT)

JPEG_JPEG_START 配置是否开始压缩新图像。

0：无效值。没有作用

1：开始压缩新图像

(WT)

JPEG_QNR_PRECISION 配置编码器的量化系数表精度。

0：8 位精度

1：16 位精度

(R/W)

JPEG_FF_CHECK_EN 配置是否在“0xFF”后添加“0x00”。

0：不添加

1：添加

(R/W)

JPEG_SAMPLE_SEL 配置待压缩图像的格式。

0：YUV444

1：YUV422

2：YUV420

3：无效值。没有作用

(R/W)

见下页...

Register 37.1. JPEG_CONFIG_REG (0x0000)

接上页...

JPEG_DMA_LINKLIST_MODE 表示 2D DMA 链表模式。

0：无效值。没有作用

1：2D DMA 使用链表进行配置

(RO)

JPEG_DEBUG_DIRECT_OUT_EN 配置是否启用编码器的调试模式。

0：禁用调试模式，编码器将在正常模式下工作

1：启用调试模式，输入图像将直接由编码器输出

(R/W)

JPEG_QNR_FIFO_EN 配置是否启用 FIFO 模式配置量化系数表。

0：禁用。使用非 FIFO 模式

1：启用

(R/W)

JPEG_LQNR_TBL_SEL 配置编码器的亮度量化系数表 ID。(R/W)

JPEG_CQNR_TBL_SEL 配置编码器的色度量化系数表 ID。(R/W)

JPEG_COLOR_SPACE 配置原始图像的颜色空间。

0：RGB888

1：YUV422

2：RGB565

3：GRAY

(R/W)

JPEG_DHT_FIFO_EN 配置是否启用 FIFO 模式配置哈夫曼表。

0：禁用。使用非 FIFO 模式

1：启用

注意，读取哈夫曼表仅支持非 FIFO 模式。

(R/W)

JPEG_MEM_CLK_FORCE_ON 配置是否强制打开内存的时钟门。

0：不强制打开

1：强制打开

(R/W)

JPEG_DECODE_TIMEOUT_THRES 配置触发 JPEG_DECODE_TIMEOUT_INT_RAW 的解码器超时时间。超时时间为 $2^{JPEG_DECODE_TIMEOUT_THRES} - 1$ 个时钟周期。(R/W)

JPEG_DECODE_TIMEOUT_TASK_SEL 配置解码器超时时的复位模式。

0：软件使用复位来中止解码过程

1：硬件自动中止解码过程

(R/W)

见下页...

Register 37.1. JPEG_CONFIG_REG (0x0000)

接上页...

JPEG_SOFT_RST 配置是否对 JPEG 图像编解码器应用软复位。

0：软复位释放

1：对 JPEG 图像编解码器应用软复位（寄存器配置值不会被复位）

(R/W)

JPEG_FIFO_RST 配置是否复位 JPEG 图像编解码器的 FIFO。

0：复位释放

1：复位 JPEG 图像编解码器的 FIFO

(R/W)

JPEG_PIXEL_REV 配置是否调换原始图像的像素顺序。

0：不调换

1：调换

(R/W)

JPEG_TAILER_EN 配置是否在码流的末尾添加 EOI 标记 “0xFFD9”。

0：不添加

1：添加

(R/W)

JPEG_PAUSE_EN 配置是否暂停 JPEG 图像编解码器。

0：不暂停

1：暂停

(R/W)

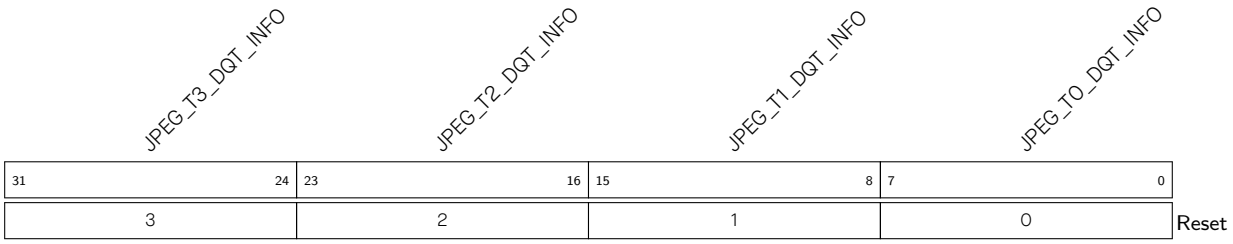
JPEG_MODE 配置 JPEG 图像编解码器的工作模式是编码器还是解码器。

0：编码器

1：解码器

(R/W)

Register 37.2. JPEG_DQT_INFO_REG (0x0004)



JPEG_T0_DQT_INFO 配置解码器量化系数表 0 的精度和 ID。

位 [7:4]：配置精度

位 [3:0]：配置 ID

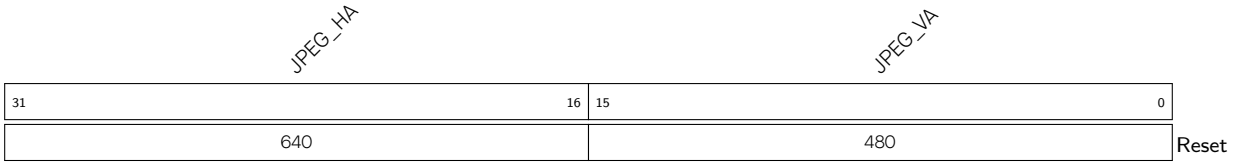
(R/W)

JPEG_T1_DQT_INFO 配置解码器量化系数表 1 的精度和 ID。详见 JPEG_T0_DQT_INFO。(R/W)

JPEG_T2_DQT_INFO 配置解码器量化系数表 2 的精度和 ID。详见 JPEG_T0_DQT_INFO。(R/W)

JPEG_T3_DQT_INFO 配置解码器量化系数表 3 的精度和 ID。详见 JPEG_T0_DQT_INFO。(R/W)

Register 37.3. JPEG_PIC_SIZE_REG (0x0008)



JPEG_VA 配置图像的高度。

当 JPEG 图像编解码器作为编码器工作时，最大可配置位数为 14。

当 JPEG 图像编解码器作为解码器工作时，最大可配置位数为 16。

(R/W)

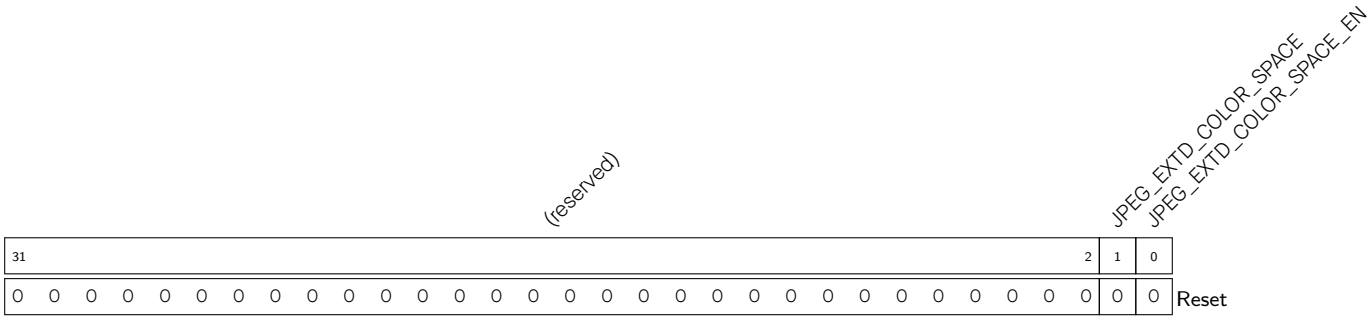
JPEG_HA 配置图像的宽度。

当 JPEG 图像编解码器作为编码器工作时，最大可配置位数为 14。

当 JPEG 图像编解码器作为解码器工作时，最大可配置位数为 16。

(R/W)

Register 37.4. JPEG_EXTD_CONFIG_REG (0x000C)



JPEG_EXTD_COLOR_SPACE_EN 配置是否扩展颜色空间转换支持的格式。

0：禁用

1：扩展

(R/W)

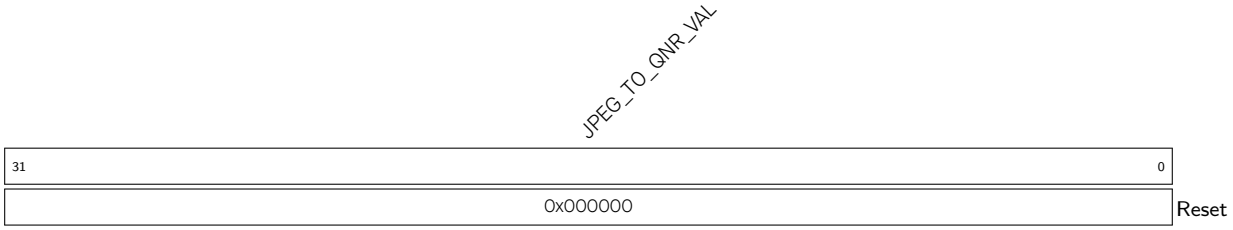
JPEG_EXTD_COLOR_SPACE 配置可转换颜色空间的扩展格式。仅在 JPEG_EXTD_COLOR_SPACE_EN 配置为 1 时有效。

0：YUV444

1：YUV420

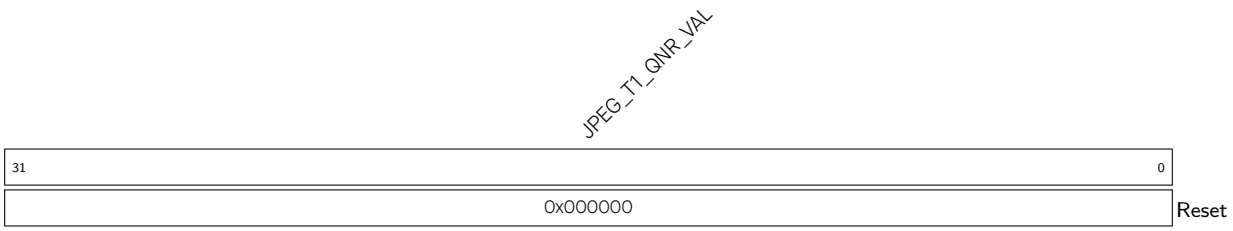
(R/W)

Register 37.5. JPEG_T0QNR_REG (0x0010)



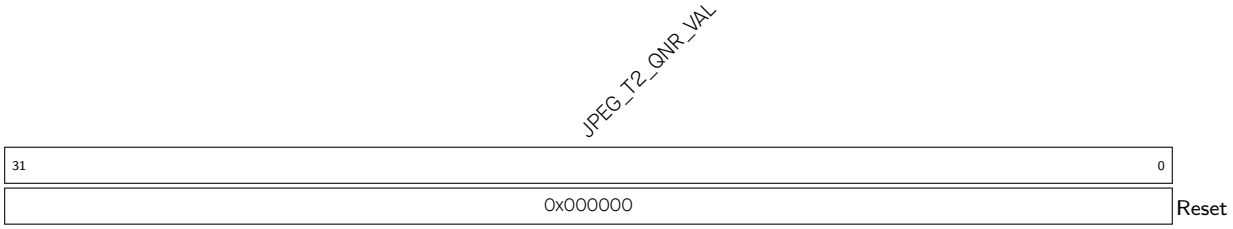
JPEG_T0_QNR_VAL 配置 FIFO 模式下量化系数表 0 的量化系数。(HRO)

Register 37.6. JPEG_T1QNR_REG (0x0014)



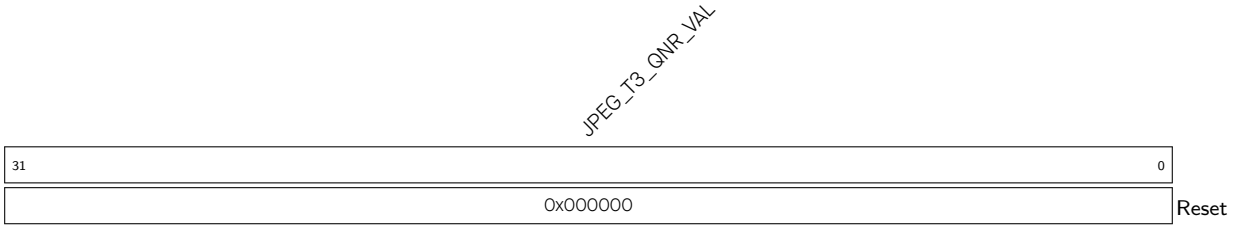
JPEG_T1_QNR_VAL 配置 FIFO 模式下量化系数表 1 的量化系数。(HRO)

Register 37.7. JPEG_T2QNR_REG (0x0018)



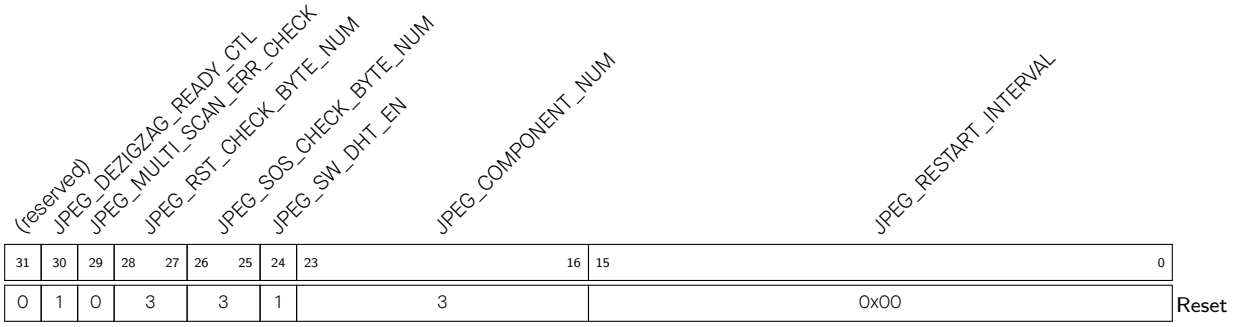
JPEG_T2_QNR_VAL 配置 FIFO 模式下量化系数表 2 的量化系数。(HRO)

Register 37.8. JPEG_T3QNR_REG (0x001C)



JPEG_T3_QNR_VAL 配置 FIFO 模式下量化系数表 3 的量化系数。(HRO)

Register 37.9. JPEG_DECODE_CONF_REG (0x0020)



JPEG_RESTART_INTERVAL 配置解码时 DRI 段的 RST 间隔。(R/W)

JPEG_COMPONENT_NUM 配置解码时图像中的色度分量数量。(R/W)

JPEG_SW_DHT_EN 表示由软件为解码器配置的哈夫曼表。该位的值始终为 1。(RO)

JPEG_SOS_CHECK_BYTE_NUM 配置解码需多次的图像时，一次扫描 SOS 标记检查的字节数。实际检查数为 JPEG_SOS_CHECK_BYTE_NUM + 1 (R/W)

JPEG_RST_CHECK_BYTE_NUM 配置解码一个 RST 间隔后，检查下一个 RST 标记的字节数。实际检查数为 JPEG_RST_CHECK_BYTE_NUM + 1。(R/W)

JPEG_MULTI_SCAN_ERR_CHECK 保留给解码器，请勿配置。(R/W)

JPEG_DEZIGZAG_READY_CTL 保留给解码器，请勿配置。(R/W)

Register 37.10. JPEG_C0_REG (0x0024)



JPEG_C0_DQT_TBL_SEL 配置解码器模式下分量 0 的选定量化系数表 ID。(R/W)

JPEG_C0_Y_FACTOR 配置解码器模式下分量 0 的垂直采样因子。(R/W)

JPEG_C0_X_FACTOR 配置解码器模式下分量 0 的水平采样因子。(R/W)

JPEG_C0_ID 配置解码器模式下分量 0 的 ID。(R/W)

Register 37.11. JPEG_C1_REG (0x0028)



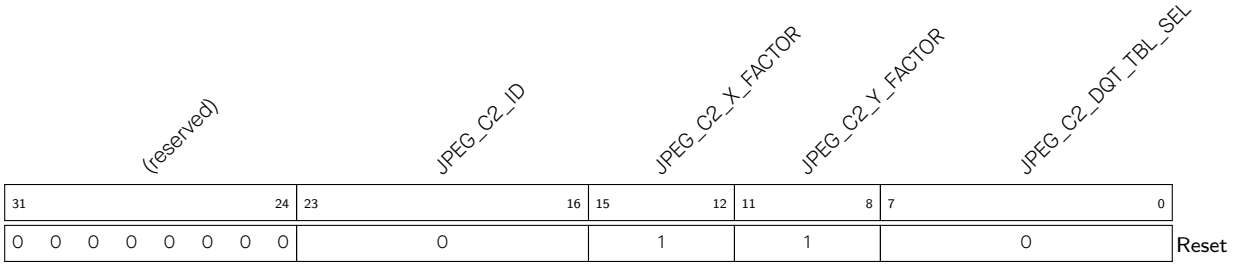
JPEG_C1_DQT_TBL_SEL 配置解码器模式下分量 1 的选定量化系数表 ID。(R/W)

JPEG_C1_Y_FACTOR 配置解码器模式下分量 1 的垂直采样因子。(R/W)

JPEG_C1_X_FACTOR 配置解码器模式下分量 1 的水平采样因子。(R/W)

JPEG_C1_ID 配置解码器模式下分量 1 的 ID。(R/W)

Register 37.12. JPEG_C2_REG (0x002C)



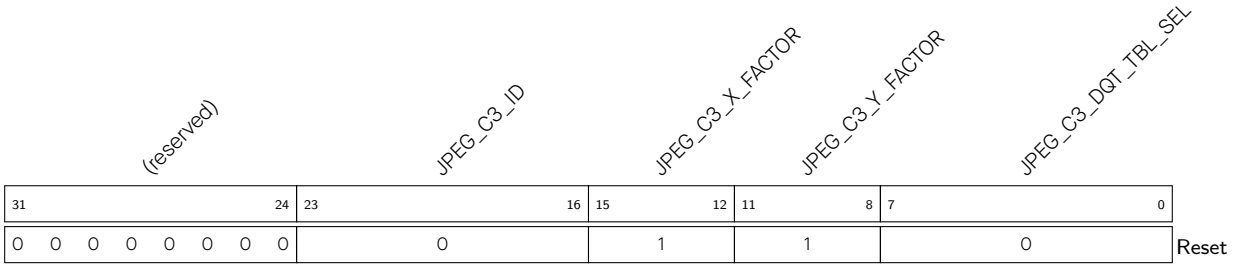
JPEG_C2_DQT_TBL_SEL 配置解码器模式下分量 2 的选定量化系数表 ID。(R/W)

JPEG_C2_Y_FACTOR 配置解码器模式下分量 2 的垂直采样因子。(R/W)

JPEG_C2_X_FACTOR 配置解码器模式下分量 2 的水平采样因子。(R/W)

JPEG_C2_ID 配置解码器模式下分量 2 的 ID。(R/W)

Register 37.13. JPEG_C3_REG (0x0030)



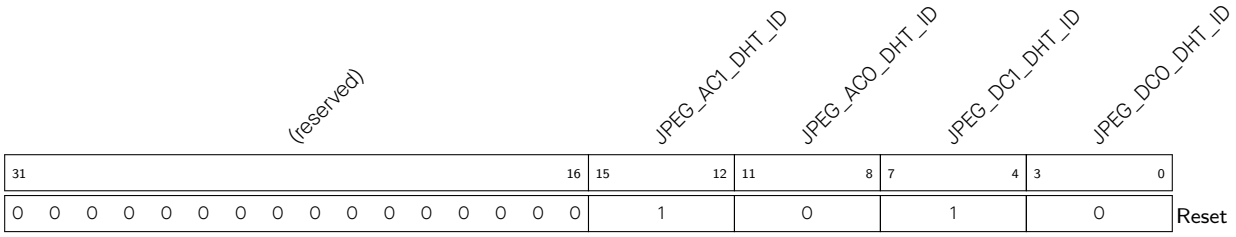
JPEG_C3_DQT_TBL_SEL 配置解码器模式下分量 3 的选定量化系数表 ID。(R/W)

JPEG_C3_Y_FACTOR 配置解码器模式下分量 3 的垂直采样因子。(R/W)

JPEG_C3_X_FACTOR 配置解码器模式下分量 3 的水平采样因子。(R/W)

JPEG_C3_ID 配置解码器模式下分量 3 的 ID。(R/W)

Register 37.14. JPEG_DHT_INFO_REG (0x0034)



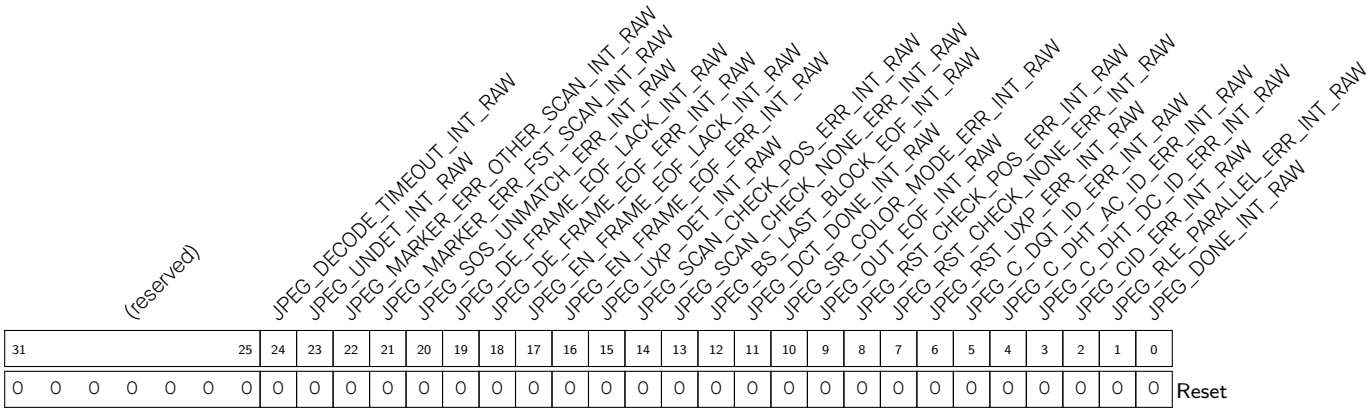
JPEG_DC0_DHT_ID 配置解码器模式下 DC0 哈夫曼表的 ID。(R/W)

JPEG_DC1_DHT_ID 配置解码器模式下 DC1 哈夫曼表的 ID。(R/W)

JPEG_AC0_DHT_ID 配置解码器模式下 AC0 哈夫曼表的 ID。(R/W)

JPEG_AC1_DHT_ID 配置解码器模式下 AC1 哈夫曼表的 ID。(R/W)

Register 37.15. JPEG_INT_RAW_REG (0x0038)



JPEG_DONE_INT_RAW JPEG_DONE_INT 的原始中断状态。(R/WTC/SS)

JPEG_RLE_PARALLEL_ERR_INT_RAW JPEG_RLE_PARALLEL_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_CID_ERR_INT_RAW JPEG_CID_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_C_DHT_DC_ID_ERR_INT_RAW JPEG_C_DHT_DC_ID_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_C_DHT_AC_ID_ERR_INT_RAW JPEG_C_DHT_AC_ID_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_C_DQT_ID_ERR_INT_RAW JPEG_C_DQT_ID_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_RST_UXP_ERR_INT_RAW JPEG_RST_UXP_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_RST_CHECK_NONE_ERR_INT_RAW JPEG_RST_CHECK_NONE_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_RST_CHECK_POS_ERR_INT_RAW JPEG_RST_CHECK_POS_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_OUT_EOF_INT_RAW JPEG_OUT_EOF_INT 的原始中断状态。(R/WTC/SS)

JPEG_SR_COLOR_MODE_ERR_INT_RAW 该中断无效。(R/WTC/SS)

JPEG_DCT_DONE_INT_RAW JPEG_DCT_DONE_INT 的原始中断状态。(R/WTC/SS)

JPEG_BS_LAST_BLOCK_EOF_INT_RAW JPEG_BS_LAST_BLOCK_EOF_INT 的原始中断状态。(R/WTC/SS)

JPEG_SCAN_CHECK_NONE_ERR_INT_RAW JPEG_SCAN_CHECK_NONE_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_SCAN_CHECK_POS_ERR_INT_RAW JPEG_SCAN_CHECK_POS_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_UXP_DET_INT_RAW JPEG_UXP_DET_INT 的原始中断状态。(R/WTC/SS)

见下页...

Register 37.15. JPEG_INT_RAW_REG (0x0038)

接上页...

JPEG_EN_FRAME_EOF_ERR_INT_RAW JPEG_EN_FRAME_EOF_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_EN_FRAME_EOF_LACK_INT_RAW JPEG_EN_FRAME_EOF_LACK_INT 的原始中断状态。(R/WTC/SS)

JPEG_DE_FRAME_EOF_ERR_INT_RAW JPEG_DE_FRAME_EOF_ERR_INT 的原始中断状态。(R/WTC/SS)

JPEG_DE_FRAME_EOF_LACK_INT_RAW JPEG_DE_FRAME_EOF_LACK_INT 的原始中断状态。(R/WTC/SS)

JPEG_SOS_UNMATCH_ERR_INT_RAW JPEG_SOS_UNMATCH_ERR_INT 的原始中断状态。(R/WTC/SS)

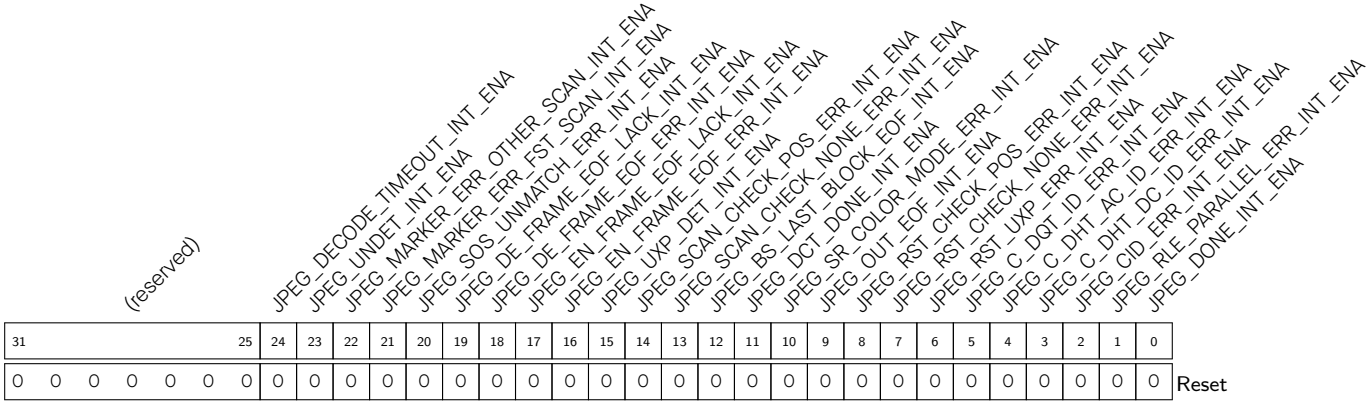
JPEG_MARKER_ERR_FST_SCAN_INT_RAW JPEG_MARKER_ERR_FST_SCAN_INT 的原始中断状态。(R/WTC/SS)

JPEG_MARKER_ERR_OTHER_SCAN_INT_RAW JPEG_MARKER_ERR_OTHER_SCAN_INT 的原始中断状态。(R/WTC/SS)

JPEG_UNDET_INT_RAW JPEG_UNDET_INT 的原始中断状态。(R/WTC/SS)

JPEG_DECODE_TIMEOUT_INT_RAW JPEG_DECODE_TIMEOUT_INT 的原始中断状态。(R/WTC/SS)

Register 37.16. JPEG_INT_ENA_REG (0x003C)



JPEG_DONE_INT_ENA 写 1 使能 JPEG_DONE_INT_ENA。(R/W)

JPEG_RLE_PARALLEL_ERR_INT_ENA 写 1 使能 JPEG_RLE_PARALLEL_ERR_INT。(R/W)

JPEG_CID_ERR_INT_ENA 写 1 使能 JPEG_CID_ERR_INT。(R/W)

JPEG_C_DHT_DC_ID_ERR_INT_ENA 写 1 使能 JPEG_C_DHT_DC_ID_ERR_INT。(R/W)

JPEG_C_DHT_AC_ID_ERR_INT_ENA 写 1 使能 JPEG_C_DHT_AC_ID_ERR_INT。(R/W)

JPEG_C_DQT_ID_ERR_INT_ENA 写 1 使能 JPEG_C_DQT_ID_ERR_INT。(R/W)

JPEG_RST_UXP_ERR_INT_ENA 写 1 使能 JPEG_RST_UXP_ERR_INT。(R/W)

JPEG_RST_CHECK_NONE_ERR_INT_ENA 写 1 使能 JPEG_RST_CHECK_NONE_ERR_INT。(R/W)

JPEG_RST_CHECK_POS_ERR_INT_ENA 写 1 使能 JPEG_RST_CHECK_POS_ERR_INT。(R/W)

JPEG_OUT_EOF_INT_ENA 写 1 使能 JPEG_OUT_EOF_INT。(R/W)

JPEG_SR_COLOR_MODE_ERR_INT_ENA 写 1 使能 JPEG_SR_COLOR_MODE_ERR_INT。(R/W)

JPEG_DCT_DONE_INT_ENA 写 1 使能 JPEG_DCT_DONE_INT。(R/W)

JPEG_BS_LAST_BLOCK_EOF_INT_ENA 写 1 使能 JPEG_BS_LAST_BLOCK_EOF_INT。(R/W)

JPEG_SCAN_CHECK_NONE_ERR_INT_ENA 写 1 使能 JPEG_SCAN_CHECK_NONE_ERR_INT。(R/W)

JPEG_SCAN_CHECK_POS_ERR_INT_ENA 写 1 使能 JPEG_SCAN_CHECK_POS_ERR_INT。(R/W)

JPEG_UXP_DET_INT_ENA 写 1 使能 JPEG_UXP_DET_INT。(R/W)

见下页...

Register 37.16. JPEG_INT_ENA_REG (0x003C)

接上页...

JPEG_EN_FRAME_EOF_ERR_INT_ENA 写 1 使能 JPEG_EN_FRAME_EOF_ERR_INT。(R/W)

JPEG_EN_FRAME_EOF_LACK_INT_ENA 写 1 使能 JPEG_EN_FRAME_EOF_LACK_INT。(R/W)

JPEG_DE_FRAME_EOF_ERR_INT_ENA 写 1 使能 JPEG_DE_FRAME_EOF_ERR_INT。(R/W)

JPEG_DE_FRAME_EOF_LACK_INT_ENA 写 1 使能 JPEG_DE_FRAME_EOF_LACK_INT。(R/W)

JPEG_SOS_UNMATCH_ERR_INT_ENA 写 1 使能 JPEG_SOS_UNMATCH_ERR_INT。(R/W)

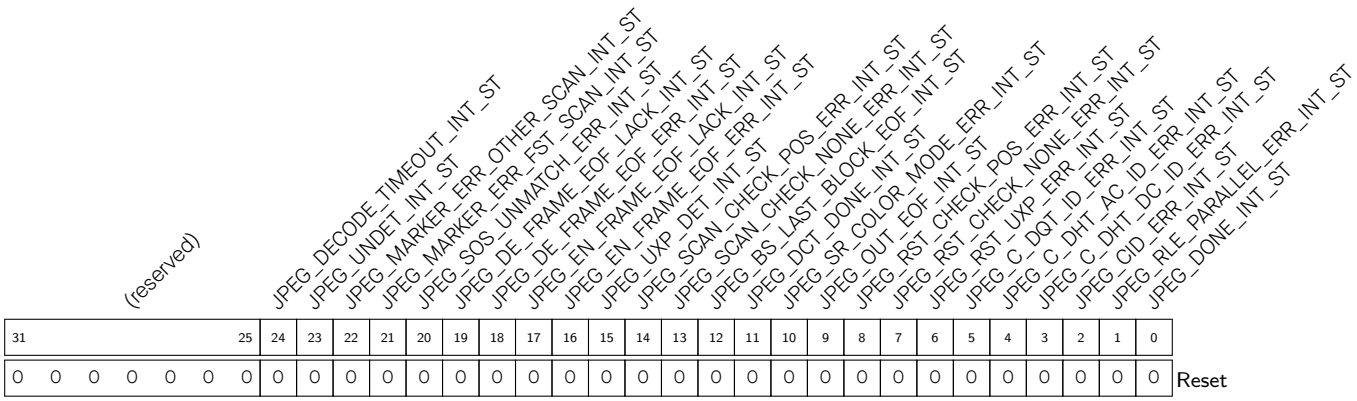
JPEG_MARKER_ERR_FST_SCAN_INT_ENA 写 1 使能 JPEG_MARKER_ERR_FST_SCAN_INT。(R/W)

JPEG_MARKER_ERR_OTHER_SCAN_INT_ENA 写 1 使能 JPEG_MARKER_ERR_OTHER_SCAN_INT。(R/W)

JPEG_UNDET_INT_ENA 写 1 使能 JPEG_UNDET_INT。(R/W)

JPEG_DECODE_TIMEOUT_INT_ENA 写 1 使能 JPEG_DECODE_TIMEOUT_INT。(R/W)

Register 37.17. JPEG_INT_ST_REG (0x0040)



JPEG_DONE_INT_ST JPEG_DONE_INT 的屏蔽中断状态。(RO)

JPEG_RLE_PARALLEL_ERR_INT_ST JPEG_RLE_PARALLEL_ERR_INT 的屏蔽中断状态。(RO)

JPEG_CID_ERR_INT_ST JPEG_CID_ERR_INT 的屏蔽中断状态。(RO)

JPEG_C_DHT_DC_ID_ERR_INT_ST JPEG_C_DHT_DC_ID_ERR_INT 的屏蔽中断状态。(RO)

JPEG_C_DHT_AC_ID_ERR_INT_ST JPEG_C_DHT_AC_ID_ERR_INT 的屏蔽中断状态。(RO)

JPEG_C_DQT_ID_ERR_INT_ST JPEG_C_DQT_ID_ERR_INT 的屏蔽中断状态。(RO)

JPEG_RST_UXP_ERR_INT_ST JPEG_RST_UXP_ERR_INT 的屏蔽中断状态。(RO)

JPEG_RST_CHECK_NONE_ERR_INT_ST JPEG_RST_CHECK_NONE_ERR_INT 的屏蔽中断状态。(RO)

JPEG_RST_CHECK_POS_ERR_INT_ST JPEG_RST_CHECK_POS_ERR_INT 的屏蔽中断状态。(RO)

JPEG_OUT_EOF_INT_ST JPEG_OUT_EOF_INT 的屏蔽中断状态。(RO)

JPEG_SR_COLOR_MODE_ERR_INT_ST JPEG_SR_COLOR_MODE_ERR_INT 的屏蔽中断状态。(RO)

JPEG_DCT_DONE_INT_ST JPEG_DCT_DONE_INT 的屏蔽中断状态。(RO)

JPEG_BS_LAST_BLOCK_EOF_INT_ST JPEG_BS_LAST_BLOCK_EOF_INT 的屏蔽中断状态。(RO)

JPEG_SCAN_CHECK_NONE_ERR_INT_ST JPEG_SCAN_CHECK_NONE_ERR_INT 的屏蔽中断状态。(RO)

JPEG_SCAN_CHECK_POS_ERR_INT_ST JPEG_SCAN_CHECK_POS_ERR_INT 的屏蔽中断状态。(RO)

JPEG_UXP_DET_INT_ST JPEG_UXP_DET_INT 的屏蔽中断状态。(RO)

见下页...

Register 37.17. JPEG_INT_ST_REG (0x0040)

接上页...

JPEG_EN_FRAME_EOF_ERR_INT_ST JPEG_EN_FRAME_EOF_ERR_INT 的屏蔽中断状态。(RO)

JPEG_EN_FRAME_EOF_LACK_INT_ST JPEG_EN_FRAME_EOF_LACK_INT 的屏蔽中断状态。(RO)

JPEG_DE_FRAME_EOF_ERR_INT_ST JPEG_DE_FRAME_EOF_ERR_INT 的屏蔽中断状态。(RO)

JPEG_DE_FRAME_EOF_LACK_INT_ST JPEG_DE_FRAME_EOF_LACK_INT 的屏蔽中断状态。(RO)

JPEG_SOS_UNMATCH_ERR_INT_ST JPEG_SOS_UNMATCH_ERR_INT 的屏蔽中断状态。(RO)

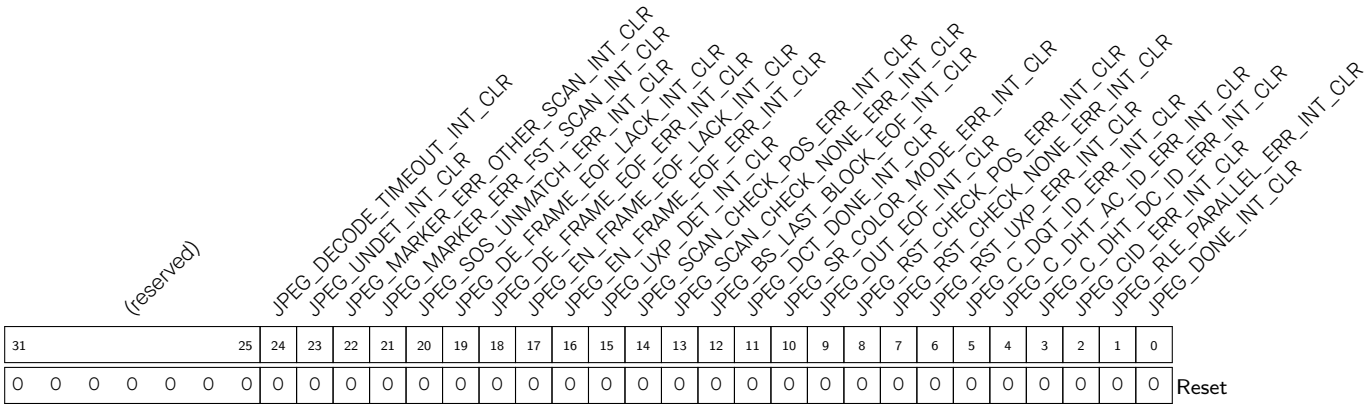
JPEG_MARKER_ERR_FST_SCAN_INT_ST JPEG_MARKER_ERR_FST_SCAN_INT 的屏蔽中断状态。(RO)

JPEG_MARKER_ERR_OTHER_SCAN_INT_ST JPEG_MARKER_ERR_OTHER_SCAN_INT 的屏蔽中断状态。(RO)

JPEG_UNDET_INT_ST JPEG_UNDET_INT 的屏蔽中断状态。(RO)

JPEG_DECODE_TIMEOUT_INT_ST JPEG_DECODE_TIMEOUT_INT 的屏蔽中断状态。(RO)

Register 37.18. JPEG_INT_CLR_REG (0x0044)



JPEG_DONE_INT_CLR 写 1 清除 JPEG_DONE_INT。(WT)

JPEG_RLE_PARALLEL_ERR_INT_CLR 写 1 清除 JPEG_RLE_PARALLEL_ERR_INT。(WT)

JPEG_CID_ERR_INT_CLR 写 1 清除 JPEG_CID_ERR_INT。(WT)

JPEG_C_DHT_DC_ID_ERR_INT_CLR 写 1 清除 JPEG_C_DHT_DC_ID_ERR_INT。(WT)

JPEG_C_DHT_AC_ID_ERR_INT_CLR 写 1 清除 JPEG_C_DHT_AC_ID_ERR_INT。(WT)

JPEG_C_DQT_ID_ERR_INT_CLR 写 1 清除 JPEG_C_DQT_ID_ERR_INT。(WT)

JPEG_RST_UXP_ERR_INT_CLR 写 1 清除 JPEG_RST_UXP_ERR_INT。(WT)

JPEG_RST_CHECK_NONE_ERR_INT_CLR 写 1 清除 JPEG_RST_CHECK_NONE_ERR_INT。(WT)

JPEG_RST_CHECK_POS_ERR_INT_CLR 写 1 清除 JPEG_RST_CHECK_POS_ERR_INT。(WT)

JPEG_OUT_EOF_INT_CLR 写 1 清除 JPEG_OUT_EOF_INT。(WT)

JPEG_SR_COLOR_MODE_ERR_INT_CLR 写 1 清除 JPEG_SR_COLOR_MODE_ERR_INT。(WT)

JPEG_DCT_DONE_INT_CLR 写 1 清除 JPEG_DCT_DONE_INT。(WT)

JPEG_BS_LAST_BLOCK_EOF_INT_CLR 写 1 清除 JPEG_BS_LAST_BLOCK_EOF_INT。(WT)

JPEG_SCAN_CHECK_NONE_ERR_INT_CLR 写 1 清除 JPEG_SCAN_CHECK_NONE_ERR_INT。(WT)

JPEG_SCAN_CHECK_POS_ERR_INT_CLR 写 1 清除 JPEG_SCAN_CHECK_POS_ERR_INT。(WT)

JPEG_UXP_DET_INT_CLR 写 1 清除 JPEG_UXP_DET_INT。(WT)

JPEG_EN_FRAME_EOF_ERR_INT_CLR 写 1 清除 JPEG_EN_FRAME_EOF_ERR_INT。(WT)

JPEG_EN_FRAME_EOF_LACK_INT_CLR 写 1 清除 JPEG_EN_FRAME_EOF_LACK_INT。(WT)

JPEG_DE_FRAME_EOF_ERR_INT_CLR 写 1 清除 JPEG_DE_FRAME_EOF_ERR_INT。(WT)

JPEG_DE_FRAME_EOF_LACK_INT_CLR 写 1 清除 JPEG_DE_FRAME_EOF_LACK_INT。(WT)

见下页...

Register 37.18. JPEG_INT_CLR_REG (0x0044)

接上页...

JPEG_SOS_UNMATCH_ERR_INT_CLR 写 1 清除 JPEG_SOS_UNMATCH_ERR_INT。(WT)

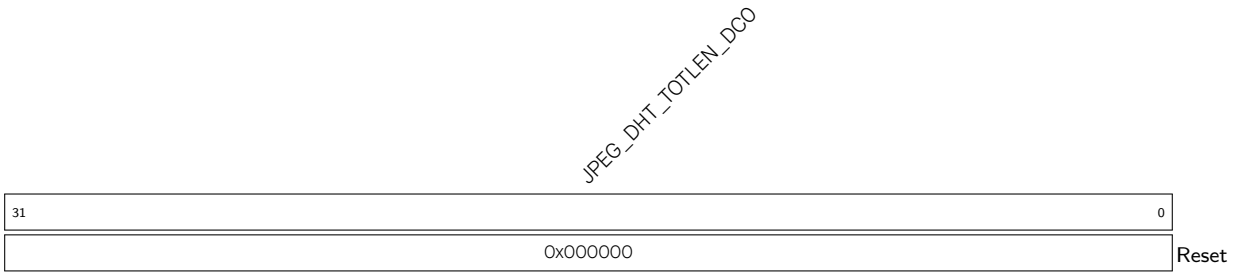
JPEG_MARKER_ERR_FST_SCAN_INT_CLR 写 1 清除 JPEG_MARKER_ERR_FST_SCAN_INT。(WT)

JPEG_MARKER_ERR_OTHER_SCAN_INT_CLR 写 1 清除 JPEG_MARKER_ERR_OTHER_SCAN_INT。(WT)

JPEG_UNDET_INT_CLR 写 1 清除 JPEG_UNDET_INT。(WT)

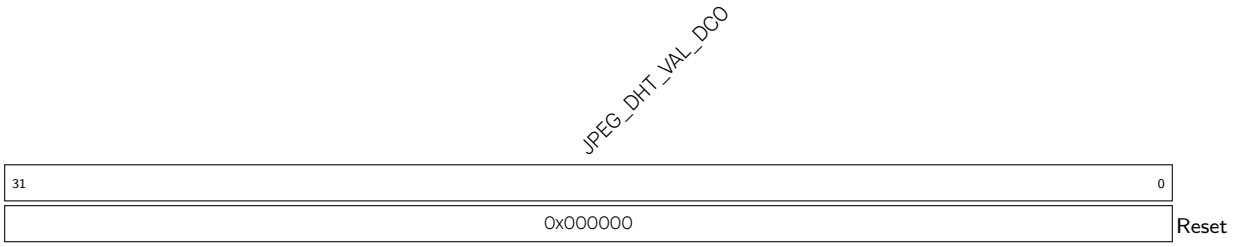
JPEG_DECODE_TIMEOUT_INT_CLR 写 1 清除 JPEG_DECODE_TIMEOUT_INT。(WT)

Register 37.19. JPEG_DHT_TOTLEN_DCO_REG (0x0058)



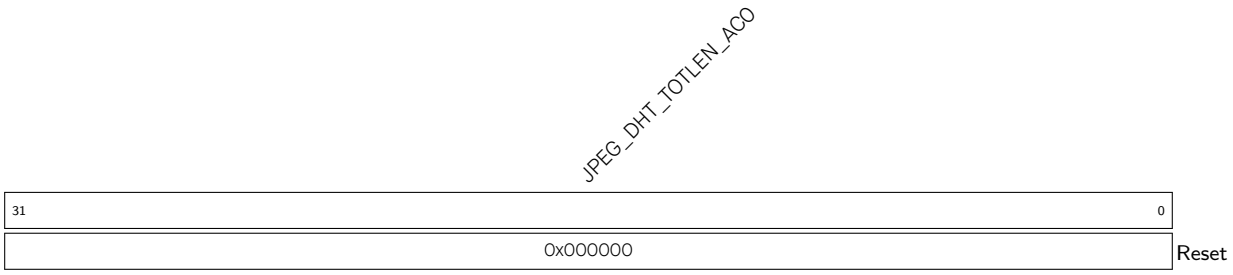
JPEG_DHT_TOTLEN_DCO 配置 FIFO 模式下 DCO 哈夫曼表长度为 1 到 16 位的码字数量。(HRO)

Register 37.20. JPEG_DHT_VAl_DCO_REG (0x005C)



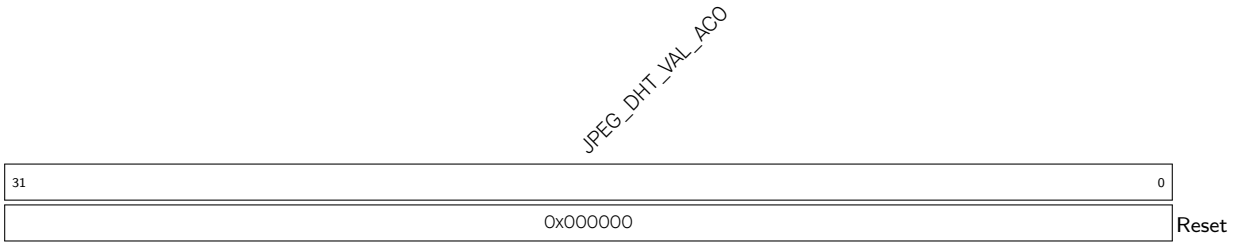
JPEG_DHT_VAL_DCO 配置 FIFO 模式下 DCO 哈夫曼表解码后码字对应的符号。(HRO)

Register 37.21. JPEG_DHT_TOTLEN_AC0_REG (0x0060)



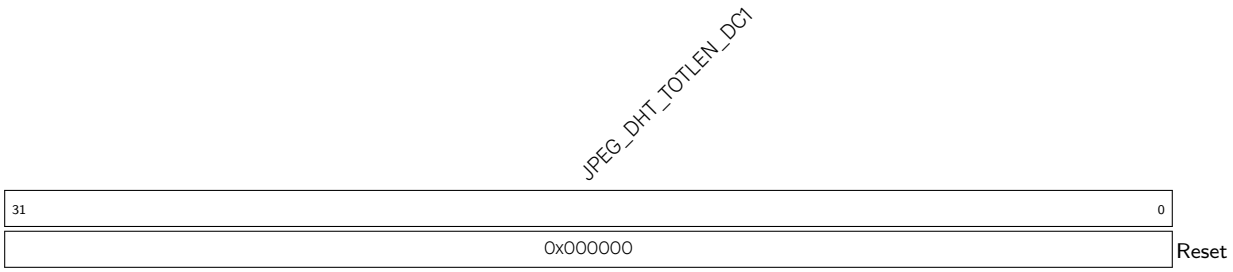
JPEG_DHT_TOTLEN_AC0 配置 FIFO 模式下 AC0 哈夫曼表长度为 1 到 16 位的码字数量。(HRO)

Register 37.22. JPEG_DHT_VAl_AC0_REG (0x0064)



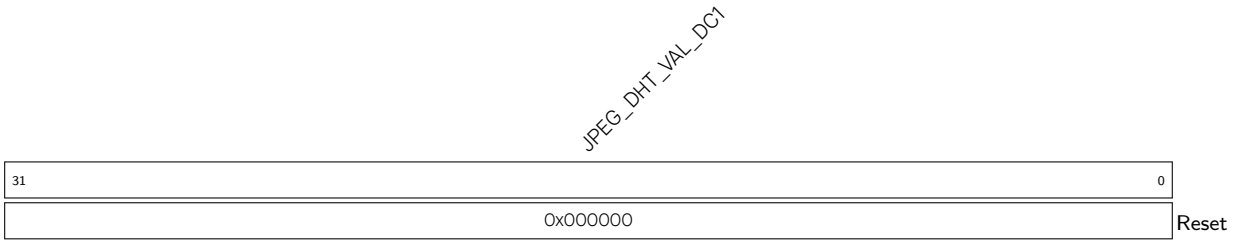
JPEG_DHT_VAL_AC0 配置 FIFO 模式下 AC0 哈夫曼表解码后码字对应的符号。(HRO)

Register 37.23. JPEG_DHT_TOTLEN_DC1_REG (0x0068)



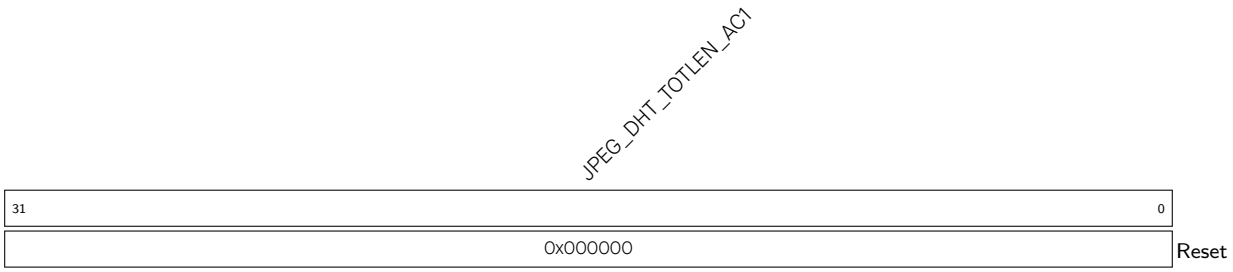
JPEG_DHT_TOTLEN_DC1 配置 FIFO 模式下 DC1 哈夫曼表长度为 1 到 16 位的码字数量。(HRO)

Register 37.24. JPEG_DHT_VAl_DC1_REG (0x006C)



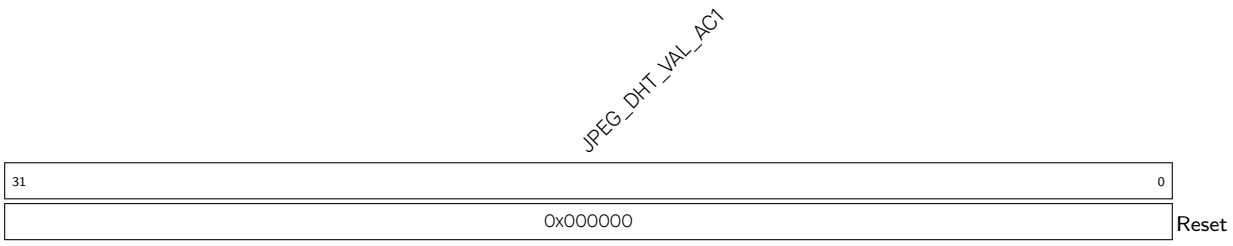
JPEG_DHT_VAL_DC1 配置 FIFO 模式下 DC1 哈夫曼表解码后码字对应的符号。(HRO)

Register 37.25. JPEG_DHT_TOTLEN_AC1_REG (0x0070)



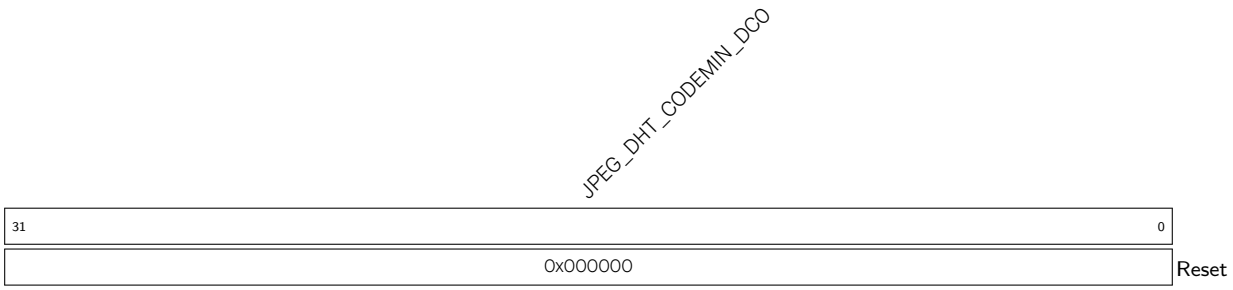
JPEG_DHT_TOTLEN_AC1 配置 FIFO 模式下 AC1 哈夫曼表长度为 1 到 16 位的码字数量。(HRO)

Register 37.26. JPEG_DHT_VAl_AC1_REG (0x0074)



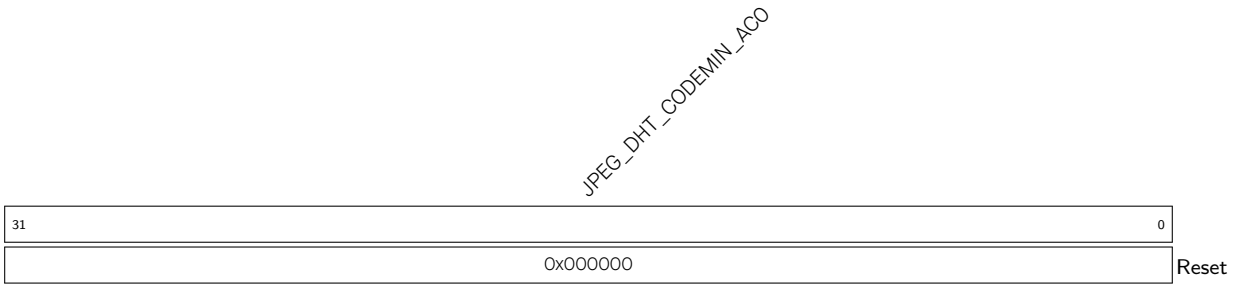
JPEG_DHT_VAL_AC1 配置 FIFO 模式下 AC1 哈夫曼表解码后码字对应的符号。(HRO)

Register 37.27. JPEG_DHT_CODEMIN_DCO_REG (0x0078)



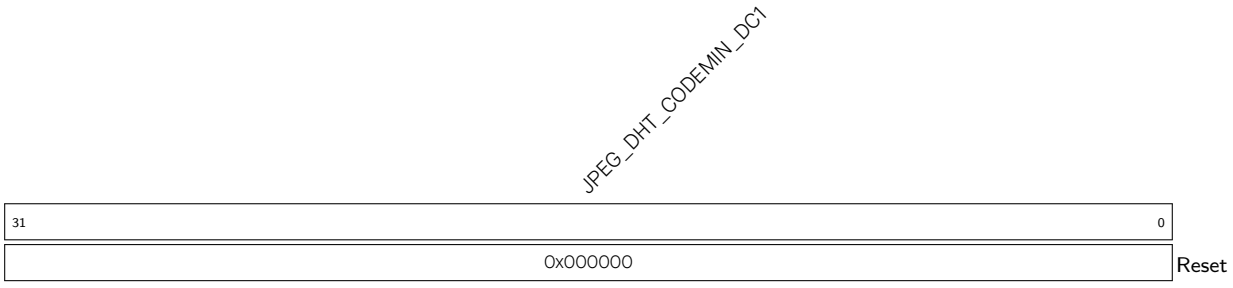
JPEG_DHT_CODEMIN_DCO 配置 FIFO 模式下 DCO 哈夫曼表长度为 1 到 16 位的最小码字。码字应向左偏移至 16 位的最高有效位。(HRO)

Register 37.28. JPEG_DHT_CODEMIN_AC0_REG (0x007C)



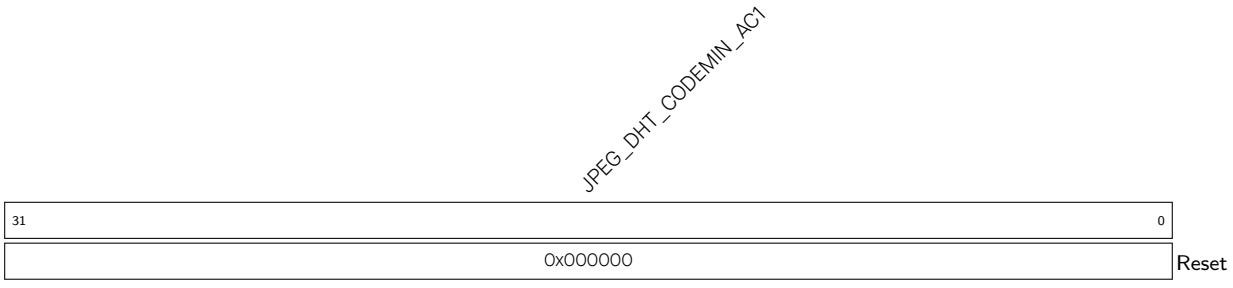
JPEG_DHT_CODEMIN_AC0 配置 FIFO 模式下 AC0 哈夫曼表长度为 1 到 16 位的最小码字。码字应向左偏移至 16 位的最高有效位。(HRO)

Register 37.29. JPEG_DHT_CODEMIN_DC1_REG (0x0080)



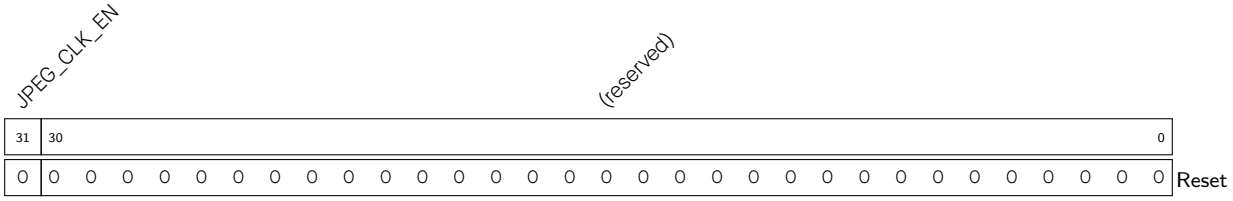
JPEG_DHT_CODEMIN_DC1 配置 FIFO 模式下 DC1 哈夫曼表长度为 1 到 16 位的最小码字。码字应向左偏移至 16 位的最高有效位。(HRO)

Register 37.30. JPEG_DHT_CODEMIN_AC1_REG (0x0084)



JPEG_DHT_CODEMIN_AC1 配置 FIFO 模式下 AC1 哈夫曼表长度为 1 到 16 位的最小码字。码字应向左偏移至 16 位的最高有效位。(HRO)

Register 37.31. JPEG_SYS_REG (0x00F8)



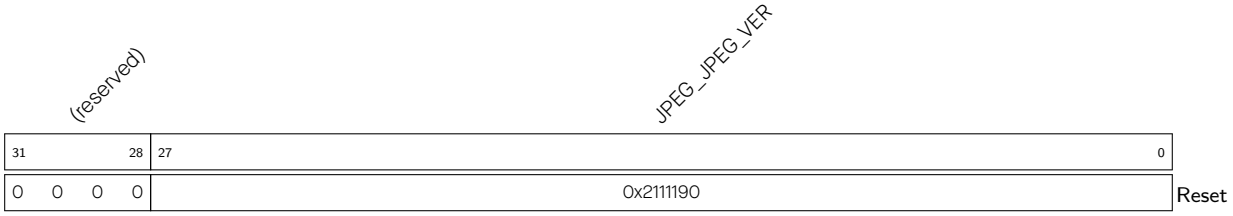
JPEG_CLK_EN 配置是否强制打开寄存器时钟门。

0：只有在应用程序写入寄存器时才打开时钟门

1：强制打开寄存器的时钟门

(R/W)

Register 37.32. JPEG_VERSION_REG (0x00FC)



JPEG_JPEG_VER 版本控制。(R/W)

第 38 章

图像信号处理器 (ISP)

38.1 概述

ESP32-P4 包含一个图像信号处理器 (ISP)，该模块由多种图像处理算法组成，以流水线方式处理图像数据。

ISP 可以从片外 DVP 摄像头接口、MIPI-CSI 摄像头接口或者系统存储器获取图像数据，处理完成后的数据经过 VDMA 再存储到系统存储器内。

ISP 的 SoC 架构如图 38.1-1 所示，ISP 必须与其他模块协同工作以完成图像数据的读写：

- 输出图像时，ISP 使用 CSI_Bridge 将数据转为 AXI 接口，并通过 VDMA 控制器将输出的图像存储到系统存储器。了解 VDMA 的更多信息，详见章节 *7 VDMA 控制器 (VDMA)*
- 当输入图像来自系统存储器，ISP 必须通过 VDMA 获取图像数据
- 当输入图像来自片外 MIPI-CSI 摄像头接口，
  - ISP 必须通过 CSI HOST 获取图像数据，详见章节 *42 MIPI CSI*
  - 可经过 CSI_CM 进行 8-bit 或 16-bit 翻转，或 RGB888、RGB565、YUV422 颜色通道调整，或 RGB888 下采样到 RGB565，或 YUV422 下采样到 YUV420
- 当输入图像来自片外 DVP 摄像头接口，ISP 数据直接从 DVP IO 输入，详见章节 *10 GPIO 交换矩阵和 IO MUX*

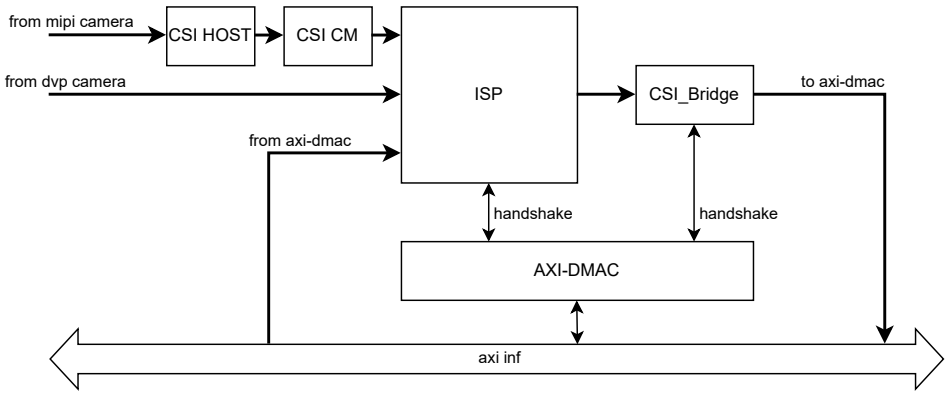


图 38.1-1. ESP32-P4 ISP SoC 架构

38.2 术语

MIPI-CSI 符合 MIPI 规范摄像头的高速串行接口

Image Interface CSI HOST 的输出接口
根据数据位宽分为 Image Interface 32 和 Image Interface 64

DVP 视频并行接口，一般由 vsync、hsync、de、data 信号组成

RAW 图像传感器直出的未经处理的数据，一般可分为 R、Gr、Gb、B 四个通道
根据数据位宽会分为 RAW8、RAW10、RAW12 等

RGB 由红、绿、蓝三色构成的彩色图像格式
根据每种颜色的位宽分为 RGB888、RGB565 等

YUV 由亮度、色度构成的彩色图像格式
根据数据排列方式分为 YUV444、YUV422、YUV420 等

38.3 主要特性

ISP 具有以下特性：

- 最大分辨率：1920 x 1080
- 支持三个数据输入源：MIPI-CSI、DVP、VDMA
- 输入格式支持：RAW8、RAW10、RAW12
- 输出格式支持：RAW8、RGB888、RGB565、YUV422、YUV420
- 流水线功能：
  - 黑电平矫正 (BLC)
  - 坏点矫正 (DPC)
  - 拜耳域降噪 (BF)
  - 镜头阴影矫正 (LSC)
  - 去马赛克
  - 白平衡增益 (WBG)
  - 颜色矫正矩阵 (CCM)
  - gamma 矫正
  - RGB 转 YUV (RGB2YUV)
  - 锐化 (sharpen)
  - 亮度、对比度、饱和度、色相调节 (COLOR)
  - YUV 范围调整
  - YUV 转 RGB (YUV2RGB)
  - 裁剪 (CROP)
  - 自动曝光统计 (AE)
  - 自动对焦统计 (AF)
  - 自动白平衡统计 (AWB)
  - 直方图统计 (HIST)

38.4 系统架构

图 38.4-1 为 ISP 架构图，主要包括 ISP_Header、ISP_Pipeline、ISP_Tail、CSI_Bridge 四部分。

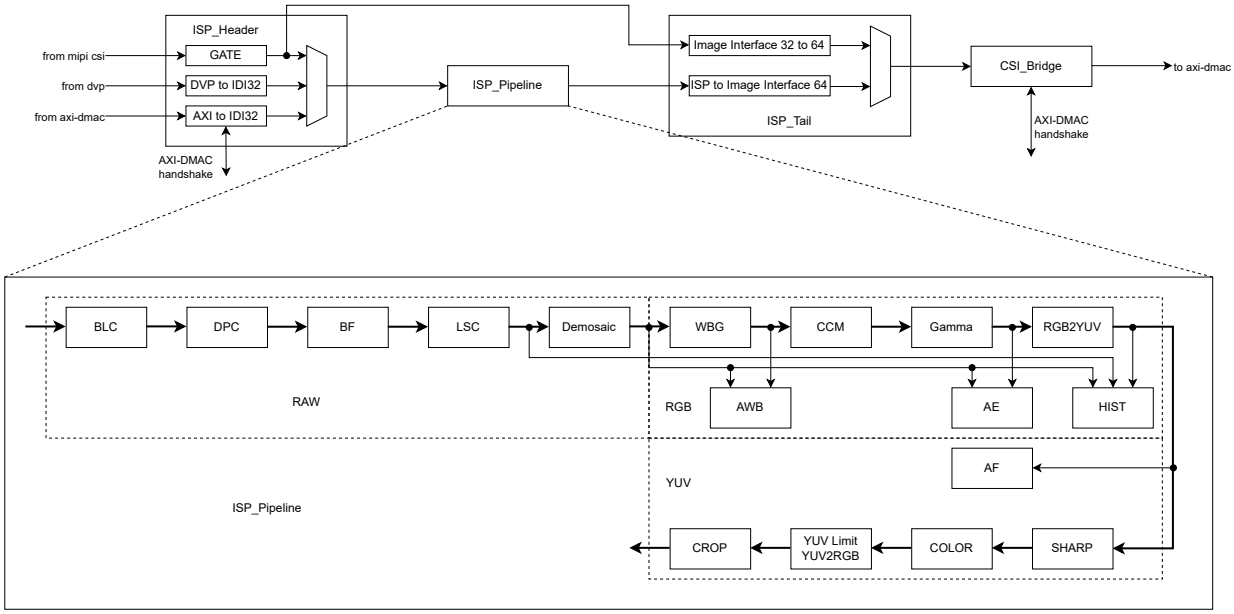


图 38.4-1. ISP IP 架构

- ISP_Header 根据配置从不同输入源选择数据并进行缓存，之后再按统一格式发送给 ISP_Pipeline。
- ISP_Pipeline 是 ISP 主要部分，所有图像处理算法均在此实现。
- ISP_Tail 包括两路输入：MIPI-CSI 以及 ISP_Pipeline。当 ISP 关闭时，MIPI-CSI 的 Image Interface 32 数据通过 ISP_Tail 进行数据格式转换，转为 Image Interface 64 数据输出到 CSI_Bridge；当 ISP 打开时，ISP_Pipeline 的输出会在 ISP_Tail 进行数据裁剪、数据格式转换，转为 Image Interface 64 再输出到 CSI_Bridge。
- CSI_Bridge 为 ISP 最终输出端口，该部分通过 VDMA 将 ISP 输出图像搬运到系统存储器。

38.5 功能描述

38.5.1 ISP_Header

ISP_Header 是 ISP 的输入级，主要功能包括：

- 根据 ISP_IN_SRC 配置选择图像数据作为输入源
- 缓存图像数据，根据配置的输入图像数据类型以及尺寸转为 ISP_Pipeline 所需格式
  - MIPI-CSI 输入时，直接通过 ISP_DATA_TYPE 设置输入类型
  - DVP 输入时，ISP_DATA_TYPE 设置为 0，再通过 ISP_CAM_DATA_TYPE 设置对应数据类型
  - VDMA 输入时，通过 ISP_DATA_TYPE 设置输入类型，同时 ISP_DMA_DATA_TYPE 也要进行对应配置

38.5.2 ISP_Pipeline

ISP_Pipeline 为 ISP 主体部分，包含了所有算法处理模块。每一个算法模块均可通过 ISP_CNTL_REG 对应位使能或禁用，且除了和颜色空间转换相关的模块外（去马赛克、RGB 转 YUV、YUV 转 RGB），其余模块的开关均支持动态开关，无需关闭 ISP 再重新配置。同时每一个算法模块开启后，均有一个帧结束中断源，能够在每一帧图像处理结束后触发中断。

38.5.2.1 黑电平矫正 (BLC)

本模块用于对输入像素做黑电平扣除。通过 ISP_BLC_Rx_VALUE 配置对应 Bayer 通道需要扣除的黑电平值。扣除黑电平值后像素范围将会缩减，可通过 ISP_BLC_Rx_STRETCH 配置是否将像素范围重新拉伸到 0~255。

一共 4 个 Bayer 通道，位置和通道编号对应如下：

- R0 对应左上位置通道
- R1 对应右上位置通道
- R2 对应左下位置通道
- R3 对应右下位置通道

38.5.2.2 坏点矫正 (DPC)

本模块用于去除输入图像的坏点。DPC 包括动态矫正和静态矫正，分别通过 ISP_STA_EN 和 ISP_DYN_EN 控制，两者可同时使能。

动态矫正根据输入像素内容自动判断坏点，并用中值进行替换。动态矫正支持以下两种坏点判断算法：

- 算法 0：统计当前像素周围 8 个像素的最小值 min8 以及最大值 max8。如果当前像素大于 max8 + ISP_DPC_THRESHOLD_H，或者小于 min8 - ISP_DPC_THRESHOLD_L，则认为该像素为坏点。
- 算法 1：设当前像素为 pix，计算周围 8 个像素的均值 est，差的绝对值 dif = |est − pix|。令 avg = (sum(8 个邻点) + 8*pix) / 16，检测分两步：
  - 第一步，选择周围 8 个像素最大值 max8，若 pix < max8 * ISP_DPC_THRESHOLD_H 且 pix > max8 * ISP_DPC_THRESHOLD_L，则认为非坏点，否则进入下一步
  - 第二步，若满足以下任一条件则认为是坏点
    - $dif > avg \times$ ISP_DPC_FACTOR_DARK and $est \geq pix$
    - $dif > (255 - avg) \times$ ISP_DPC_FACTOR_BRIG and $est < pix$

静态矫正通过初始化坏点坐标查找表（LUT）实现，LUT 存储的坏点坐标对应像素会被中值替换。坏点坐标需要通过静态标定获取，该功能不能和静态/动态矫正同时使能。

标定步骤如下：

- 黑图坏点标定
  - 配置 ISP_DPC_BLACK_EN = 1
  - 配置 ISP_DPC_CHECK_EN = 1
  - 配置 ISP_DPC_EN = 1
  - 输入黑图，等待完成

  - 配置 ISP_DPC_CHECK_EN = 0
  - 软件通过 LUT 寄存器读取所有黑图坏点坐标
- 白图坏点标定
  - 配置 ISP_DPC_BLACK_EN = 0
  - 配置 ISP_DPC_CHECK_EN = 1
  - 配置 ISP_DPC_EN = 1
  - 输入白图，等待完成
  - 配置 ISP_DPC_CHECK_EN = 0
  - 软件通过 LUT 寄存器读取所有白图坏点坐标
- 软件将得到的黑图坏点坐标和白图坏点坐标合并，去除重复坐标，并写回 LUT

LUT 读写流程可参考小节 38.5.2.4。其中，配置 ISP_LUT_NUM = 1 可选择 DPC 坏点坐标 LUT，该 LUT 深度为 512，即最多存储 512 个坏点。

38.5.2.3 拜耳域降噪 (BF)

本模块用于对输入图像在拜耳域进行降噪处理。通过 ISP_SIGMA 可以调整降噪强度，通过 ISP_GAU_TEMPLATExx 能够改变降噪模板微调处理效果。

38.5.2.4 镜头阴影矫正 (LSC)

本模块用于在拜耳域进行镜头阴影矫正。本模块对 R、Gr、Gb、B 四个通道进行独立矫正，每个通道都按照 32 x 32 进行窗口划分（相当于对整幅图像按照 64 x 64 划分），每个通道的矫正系数存储在查找表 (LUT) 内。

在启动该模块前，需要先进行标定，获取 LUT 矫正系数。具体表现为，在环境光线均匀场景下拍摄一张白色图像，将图像每个通道也按照 32 x 32 网格进行划分，取图像亮度中心亮度作为参考亮度，并由此计算每个网格顶点处的亮度增益。对于右、下方超出图像范围的网格，可计算图像边界点的增益，再根据上一个点的增益按照线性关系计算下一个网格边界增益值。之后按照从左到右、从上到下的顺序将增益存储到 LUT 完成初始化。

以 1920 x 1080 分辨率图像为例，此时每一个通道的分辨率则为 960 x 540。所有通道均按 32 x 32 的网格进行划分，水平方向可划分 30 个网格，垂直方向可划分 17 个网格。假定图像亮度中心就在图像几何中心，可取坐标 (480，270) 附近像素的亮度平均值作为参考亮度 ref_lum。接下来，按照从左往右、从上往下的顺序，依次取每个网格顶点的实际像素亮度 act_lum，通过公式 $gain = ref_lum/act_lum$ 获取矫正系数，并按顺序存入 LUT。

对于超出图像的网格，比如垂直方向最后一个网格的垂直坐标为 543，超过实际最大垂直坐标 539。此时可取坐标 511 的矫正系数 gain_0 和坐标 539 的矫正系数 gain_1，按照坐标比例 $gain = gain_0 + (gain_1 - gain_0)/(539 - 511) * (543 - 511)$ 得到 543 的增益。

LSC 有两个 LUT，分别用于存储 Gb、B 和 R、Gr 通道的增益。LUT 深度为 560，根据图像分辨率计算得到的每个通道的增益系数数量不得超过该值。增益为 10 位定点数（高 2 位为整数位，低 8 位为小数位），存储顺序如表 38.5-1 所示。LUT 可通过寄存器进行访问：

- 写操作会将 ISP_LUT_WDATA_REG 中的数据写入 LUT。
- 读操作会将从 LUT 读取的数据存入 ISP_LUT_RDATA_REG。

- 通过 ISP_LUT_CMD_REG 发起 LUT 操作命令，每次写该寄存器都会发起一次 LUT 读写命令，因此该寄存器所有域需同时写入。其中：
  - ISP_LUT_CMD：0 为读操作，1 为写操作
  - ISP_LUT_NUM：置 0 选择 LSC LUT
  - ISP_LUT_ADDR：配置访问的 LUT 地址，其中第 [11:10] 位为 0 时选择 LUT_Gb_B，第 [11:10] 位为 1 时选择 LUT_R_Gr

表 38.5-1. LSC LUT 增益存储顺序

LUT sel	bit [19:10]	bit [9:0]
LUT_Gb_B	Gb	B
LUT_R_Gr	R	Gr

LSC 开启前，还需配置 ISP_LSC_XTABLESIZE 设置水平方向矫正系数的数量。该值由公式 $ISP_LSC_XTABLESIZE = fix((line_width - 1)/2/32) + 2$ 得到，其中 $fix()$ 为取整操作。比如，输入 RAW 图像水平宽度为 1920，则 $ISP_LSC_XTABLESIZE = fix((1920 - 1)/2/32) + 2 = 29 + 2 = 31$。

38.5.2.5 去马赛克

去马赛克用于将拜耳图像转为 RGB888 图像。使用 ISP_DEMOSAIC_GRAD_RATIO 可以对转换效果进行微调，该寄存器域为 2 位整数，4 位小数，默认值为 1.0。

38.5.2.6 白平衡增益 (WBG)

本模块用于图像白平衡调整，原理是对应颜色通道的像素值乘以 ISP_WBG_R/B/G 增益系数。每个增益系数整数部分为 4 bit，小数部分为 8 bit。

38.5.2.7 颜色矫正矩阵 (CCM)

CCM 为图像像素的矫正矩阵，可对图像中每一个 RGB888 像素进行一定程度的调整，包括白平衡的调整也会用到该模块。模块的输出 R'、G'、B' 和输入 R、G、B 关系如下矩阵所示，其中，RR、RG、RB、GR、GG、GB、BR、BG、BB 可通过 ISP_CCM_xx 配置，每一个参数均为 13 位定点数。其中第 [12] 位为符号位，第 [11:0] 位为数值的绝对值，[11:8] 为整数部分，[7:0] 为小数部分。

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} RR & RG & RB \\ GR & GG & GB \\ BR & BG & BB \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{38.1}$$

38.5.2.8 gamma 矫正

本模块用于调整图像的 gamma 曲线。用户需要在使能 gamma 之前配置好 gamma 曲线。可通过寄存器实现 gamma 曲线配置，R、G、B 通道均有一个独立曲线，每个曲线有 16 个采样点。其中，X 分量为输入值，通过 ISP_GAMMA_R/G/B_Xn（n 为 00 至 0F）间接获得；Y 分量为输出值，可直接通过 ISP_GAMMA_R/G/B_Yn（n 为 00 - 0F）设置。配置完成后，通过向 ISP_GAMMA_UPDATE 写 1 生效。gamma 曲线示意如图 38.5-1 所示。

对于 X 分量，以 R 通道为例，需要满足以下条件：

$$X(n) - X(n-1) = 2^{ISP_GAMMA_R_X(n)}$$

$$sum(2^{ISP_GAMMA_R_X(n)}) = 256, n = 00\text{~}0F$$

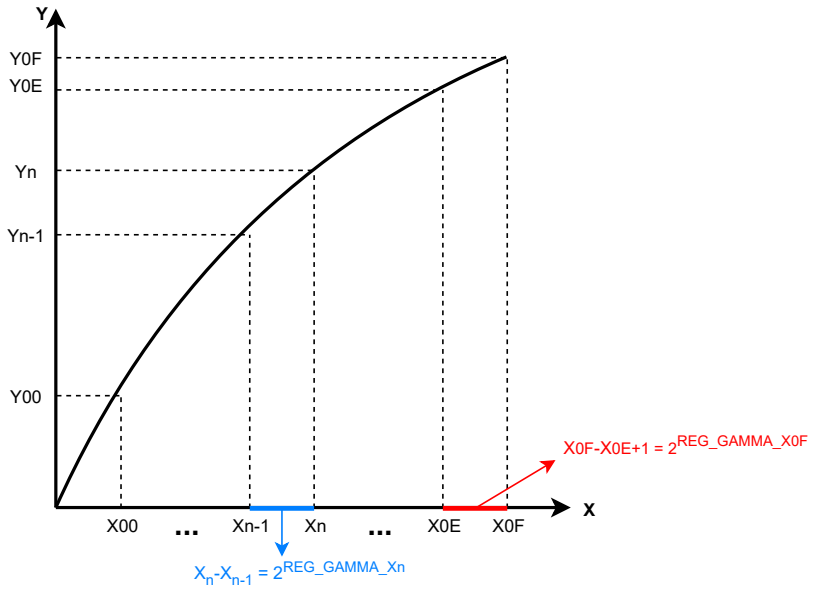


图 38.5-1. gamma 曲线示意

38.5.2.9 RGB 转 YUV (RGB2YUV)

RGB2YUV 将 RGB888 像素转为 full range 的 YUV422 格式，转换协议可通过 ISP_YUV_MODE 配置。

38.5.2.10 锐化 (sharpen)

该模块用于对图像进行锐化。该模块通过寄存器 ISP_SHARP_FILTER_COExx 配置的低通滤波器模板（如均值模板、高斯模板）分离图像的低频、高频分量图像。对于低频分量不做处理，对于高频分量图像，其每一个像素作如下处理以实现锐化：

- 低于 ISP_SHARP_THRESHOLD_LOW，将对应高频像素置 0。
- 高于 ISP_SHARP_THRESHOLD_LOW 但低于 ISP_SHARP_THRESHOLD_HIGH，将对应高频像素乘以系数 ISP_SHARP_AMOUNT_LOW。
- 高于 ISP_SHARP_THRESHOLD_HIGH，将对应高频像素乘以系数 ISP_SHARP_AMOUNT_HIGH。

每一帧图像结束后，可通过读取 ISP_SHARP_GRADIENT_MAX 获得该帧图像高频分量像素的最大值，以此可作为下一帧图像的配置参考。

38.5.2.11 对比度 / 色相 / 饱和度 / 亮度 (COLOR)

本模块用于调整图像的对比度、饱和度、色相和亮度。其中，对比度、饱和度、亮度通过 ISP_COLOR_CONTRAST、ISP_COLOR_SATURATION、ISP_COLOR_BRIGHTNESS 进行调整，色相通过将 ISP_COLOR_HUE_H 和 ISP_COLOR_HUE 组合成一个 9-bit 寄存器进行调整。

38.5.2.12 YUV 范围调整及 YUV 转 RGB (YUV2RGB)

YUV_Limit 将 full range YUV422 转为 limit range YUV422。该模块只有在 RGB2YUV 开启、YUV2RGB 关闭、且 ISP_YUV_RANGE 为 1 的情况下才会打开。此时 ISP 的输出类型为 YUV422 或 YUV420。

YUV2RGB 将 full range 的 YUV422 像素转为 RGB888 格式，转换协议可通过 ISP_YUV_MODE 配置。

> **说明：**
> 对于 full range YUV，Y、U、V 通道范围均为 0 到 255。
> 对于 limit range YUV，Y 通道范围为 16 到 235，U、V 通道范围为 16 到 240。

38.5.2.13 裁剪 (CROP)

CROP 用于图像裁剪，通过 ISP_CROP_X_START、ISP_CROP_X_END、ISP_CROP_Y_START、ISP_CROP_Y_END 分别配置 X、Y 方向的裁剪起始、结束坐标。起始坐标必须为偶数，结束坐标必须为奇数。

38.5.2.14 自动曝光统计 (AE)

AE 使用 25 个统计子窗口进行图像亮度统计，所有子窗口可以组成一个 5 x 5 的统计窗口，窗口尺寸通过寄存器 ISP_AE_BX/BY_REG 设置，子窗口编号如图 38.5-2 所示。每一个子窗口的 8 位平均亮度可以通过 ISP_AE_Bxx_MEAN 读取，用户可以根据该统计结果实现摄像头的自动曝光算法。

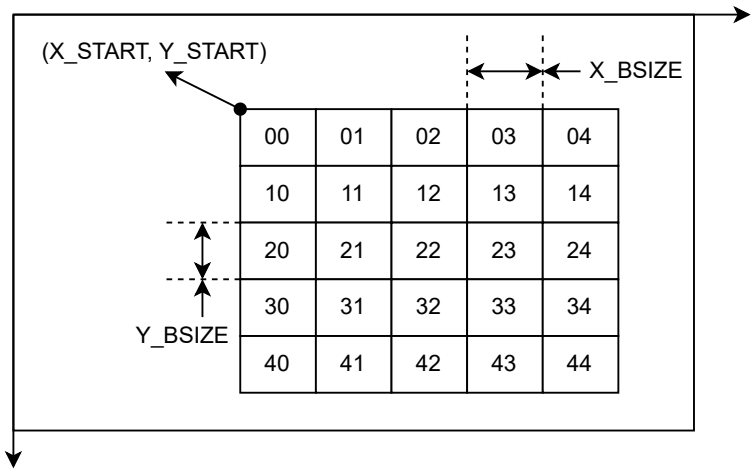


图 38.5-2. 子窗口编号示意图

AE 有两个数据采样点，分别为 demosaic 的输出和 gamma 的输出，可通过 ISP_AE_SELECT 进行配置。

AE 还包括一个简易的亮度监测模块，用户可自行选择使用该模块的硬件监测或使用软件实现监测。该模块每隔 ISP_AE_MONITO_RPERIOD 帧数会统计一次所有子窗口亮度均值。当统计结果不在 ISP_AE_MONITOR_TH/TL 设置的阈值内，就会触发 AE 亮度监测的中断。该亮度监测模块不支持各个子窗口设置不同权重，因此若要实现不同权重监测（如高光优先策略），需要软件定时触发手动统计并处理统计数据。

38.5.2.15 自动对焦统计 (AF)

AF 用于统计当前图像下统计窗口内的亮度、清晰度信息，采样点位于 sharpen 的输出。AF 有 3 个独立的统计窗口，每个统计窗口的大小、位置均可通过 ISP_AF_H/VSCALE_A/B/C_REG 配置。模块支持自动统计和手动

触发统计。每个窗口的亮度、清晰度统计结果可以通过 ISP_AF_LUMA/B/C、ISP_AF_SUMA/B/C 获取，用户可以根据窗口的统计结果实现摄像头的自动对焦算法。

AF 还包括一个对焦场景监测模块，用户可自行选择使用该模块的硬件监测或使用软件实现监测。该模块以当前所有统计窗口亮度、清晰度的累加值变化作为判据，每 ISP_AF_ENV_PERIOD 帧数进行一次统计。当连续两次统计的亮度、清晰度与基准值相差大于阈值，则触发对焦场景变化的中断，需要重新对焦。基准值在每完成一次自动统计或手动触发统计后会自动更新，阈值的设置有三种方案：

- 直接通过 ISP_AF_ENV_USER_THRESHOLD_LUM 和 ISP_AF_ENV_USER_THRESHOLD_SUM 分别设置亮度、清晰度阈值
- 当 ISP_AF_ENV_USER_THRESHOLD_SUM 为 0 时，使用对应的清晰度基准值乘以 ISP_AF_ENV_THRESHOLD（4 位小数）作为清晰度阈值
- 当 ISP_AF_ENV_USER_THRESHOLD_LUM 为 0 时，使用对应的亮度基准值乘以 ISP_AF_ENV_THRESHOLD（4 位小数）作为亮度阈值

38.5.2.16 自动白平衡统计 (AWB)

AWB 用于统计所设置的窗口下白点的白平衡信息。其中，窗口坐标可通过 ISP_AWB_H/VSCALE_REG 设置；白点通过亮度上下限 ISP_AWB_MAX/MIN_LUM、R/G 上下限 ISP_AWB_MAX/MIN_RG、B/G 上下限 ISP_AWB_MAX/MIN_BG 进行筛选。

统计结果包括白点数量以及所有白点的 R、G、B 分量累加值，可分别通过 ISP_AWB0_WHITE_CNT、ISP_AWB0_ACC_R/G/B 获取，用户可以根据统计结果实现摄像头的自动白平衡算法。

此外，AWB 还支持在窗口内设置 5 x 5 子窗口，通过 ISP_AWB_X_START、ISP_AWB_X_BSIZE、ISP_AWB_Y_START、ISP_AWB_Y_BSIZE 配置。子窗范围不能超过主窗口范围，如图 38.5-3 所示。统计结果会写入 AWB LUT 内，通过配置 ISP_LUT_NUM = 2 访问，具体流程可参考小节 38.5.2.4。统计结果按子窗从左往右、从上往下的顺序写入，每一个子窗结果占用 4 个地址，从低地址到高地址分别为白点数量、白点 R 通道累加值、白点 G 通道累加值、白点 B 通道累加值。

AWB 有两个数据采样点，分别为 Demosaic 的输出和 CCM 的输出，可通过 ISP_AWB_SAMPLE 进行配置。

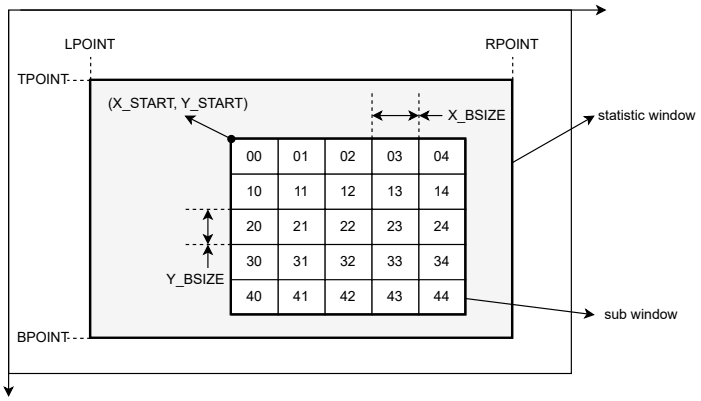


图 38.5-3. AWB 窗口示意图

38.5.2.17 直方图统计 (HIST)

HIST 也用于统计图像的亮度信息。该模块可设置一个统计窗口，统计窗口可分为 5 x 5 子窗口，子窗口编号和 AE 类似，通过寄存器 ISP_HIST_OFFS_REG 和 ISP_HIST_SIZE_REG 配置。每个子窗口权重均可通过 ISP_HIST_WEIGHT_xx 独立配置。

统计结果会按照一个 16 区间的直方图输出。直方图的横坐标通过 ISP_HIST_SEG_x_x 设置，最终统计结果通过 ISP_HIST_BIN_x 获取。

HIST 三个数据采样点分别为 BF、去马赛克、RGB2YUV，对应 RAW、RGB、YUV 数据采样点，可通过 ISP_HIST_MODE 进行配置。当采样点选择 RGB 时，可通过 ISP_HIST_COEFF_R/G/B 配置 RGB 加权系数。

38.5.3 ISP_Tail

ISP_Tail 为后处理部分，其输入数据有两个输入源。两路输入最终都以 Image Interface 64 格式输出到 CSI_Bridge。

当 ISP 关闭时，ISP_Tail 接收来自 MIPI CSI HOST 的 Image Interface 32 数据，将其转为 Image Interface 64 并输出到 CSI_Bridge。输入的 Image Interface 32 数据可以通过 ISP_BYTE_ENDIAN_ORDER 调整字节顺序。

当 ISP 打开时，ISP_Tail 接收来自 ISP_Pipeline 的数据，并进行进一步处理输出到 CSI_Bridge。这些处理包括：

- 根据寄存器配置，将 RGB888 转为 RGB565
- 根据寄存器配置，将 YUV422 转为 YUV420
- 将 ISP_Pipeline 的数据转为 Image Interface 64 格式

38.5.4 时序控制

对于存在矩阵操作的模块，需要缓存多行数据再开始操作。其中，图像的最后几行数据缓存，需要模块产生输出时序来推出。通过 ISP_BF/DEMOSAIC/SHARP_TAIL_PIXEN_PULSE_TH/TL 可以控制此时的输出时序，防止速度太快使后续的 CSI_Bridge 缓存溢出。该寄存器设置过大时可能会打乱正常数据时序，请谨慎操作。其中 TH 必须要小于 ISP_HADR_NUM-1，推荐将 ISP_BF/DEMOSAIC/SHARP_TAIL_PIXEN_PULSE_TH/TL 设置为小于等于 8。

38.5.5 影子寄存器

ISP 部分模块支持影子寄存器，具体模块如下：

- BLC
- DPC
- BF
- WBG
- CCM
- GAMMA
- SHARP

- COLOR

其中，GAMMA 的影子寄存器更新方式不可配，需严格按照小节 38.5.2.8 中的描述操作。其余模块可通过 ISP_SHADOW_UPDATE_SEL 配置影子寄存器更新方式。

38.5.6 CSI_CM

对于 MIPI CSI 输入，通过配置CSI_BRIG_HOST_CM_CTRL_REG，可以借助 CSI_CM 进行简单的数据转换，包括：

- 数据高低 8-bit 翻转
- 数据高低 16-bit 翻转
- RGB888 颜色通道顺序调整，行像素量必须为 4 的倍数
- RGB888 下采样到 RGB565，行像素量必须为 4 的倍数
- RGB565 颜色通道顺序调整，行像素量必须为 2 的倍数
- YUV422 颜色通道顺序调整，行像素量必须为 2 的倍数
- YUV422 下采样到 YUV420，行像素量必须为 16 的倍数

该模块结构如图 38.5-4 所示。

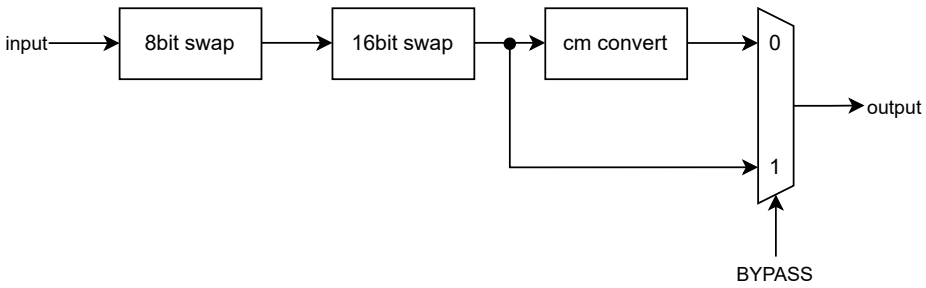


图 38.5-4. ESP32-P4 CSI CM 结构

38.5.7 CSI_Bridge

CSI_Bridge 是 ISP 和系统存储器之间的接口，该模块借助 VDMA 将来自 ISP 的数据存储到系统存储器。整幅图像可以作为一个 block 传输，也可以分成多个 block 传输，通过 CSI_BRIG_DMA_BURST_LEN 和 CSI_BRIG_DMABLK_SIZE 进行配置。

- CSI_BRIG_DMA_BURST_LEN 配置一次 VDMA 传输 burst 长度包含的 64 位数据量。
- CSI_BRIG_DMABLK_SIZE 配置一个 VDMA 块传输包含的 VDMA burst 传输数量。若一帧图像只用一个 block 传输，此时一个 block 内包含 n 个 VDMA burst，则将该寄存器域配置为大于 n 的值。建议配置为最大值 0x1FFF。

38.5.8 像素格式及字节排序

38.5.8.1 输出像素排列格式

ISP 使能关闭后，ISP_Tail 直接接收来自 MIPI CSI HOST 的 Image Interface 32 数据，其数据排列可参考章节 42 *MIPI CSI*。在该情况下，CSI HOST 输出的 Image Interface 32 数据在 ISP_Tail 和 CSI_Bridge 可经过两级字节排序，详细可参考 字节排序

在 ISP 使能之后，经过 ISP_Pipeline 处理后的输出数据，其排列格式如表 38.5-2 所示，并最终按照该格式由 ISP 输出。

表 38.5-2. ISP 输出像素排列格式

Color Mode	Addr + 7	Addr + 6	Addr + 5	Addr + 4	Addr + 3	Addr + 2	Addr + 1	Addr + 0
RAW8	RAW7[7:0]	RAW6[7:0]	RAW5[7:0]	RAW4[7:0]	RAW3[7:0]	RAW2[7:0]	RAW1[7:0]	RAW0[7:0]
RGB888	G2[7:0]	B2[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
	B5[7:0]	R4[7:0]	G4[7:0]	B4[7:0]	R3[7:0]	G3[7:0]	B3[7:0]	R2[7:0]
	R7[7:0]	G7[7:0]	B7[7:0]	R6[7:0]	G6[7:0]	B6[7:0]	R5[7:0]	G5[7:0]
RGB565	R3[4:0]G3[5:3]	G3[2:0]B3[4:0]	R2[4:0]G2[5:3]	G2[2:0]B2[4:0]	R1[4:0]G1[5:3]	G1[2:0]B1[4:0]	R0[4:0]G0[5:3]	G0[2:0]B0[4:0]
YUV422	Y3[7:0]	V2[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	V0[7:0]	Y0[7:0]	U0[7:0]
YUV420 (1、3、5... 等奇数行)	Y4[7:0]	U4[7:0]	Y3[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	Y0[7:0]	U0[7:0]
	U10[7:0]	Y9[7:0]	Y8[7:0]	U8[7:0]	Y7[7:0]	Y6[7:0]	U6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	U14[7:0]	Y13[7:0]	Y12[7:0]	U12[7:0]	Y11[7:0]	Y10[7:0]
YUV420 (2、4、6... 等偶数行)	Y4[7:0]	V4[7:0]	Y3[7:0]	Y2[7:0]	V2[7:0]	Y1[7:0]	Y0[7:0]	V0[7:0]
	V10[7:0]	Y9[7:0]	Y8[7:0]	V8[7:0]	Y7[7:0]	Y6[7:0]	V6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	V14[7:0]	Y13[7:0]	Y12[7:0]	V12[7:0]	Y11[7:0]	Y10[7:0]

38.5.8.2 字节排序

CSI_Bridge 可以通过 CSI_BRIG_BYTE_ENDIAN_ORDER 对输入的 64 位数据按照字节进行重排。在 ISP_Tail 内部也可通过 ISP_BYTE_ENDIAN_ORDER 对字节顺序进行重排。两者差异如下，作用效果如图 38.5-5 所示。

- ISP_Tail 的字节排序作用于其 Image Interface 32 输入，将 32 位的输入数据的高低字节进行翻转后，再拼成 64 位数据输出。这个步骤只在 ISP 关闭时起作用，主要用于处理 Image Interface 32 的 YUV422、YUV420 输入类型。
- CSI_Bridge 的字节排序作用于 64 位的输入数据（即 ISP_Tail 的输出数据），将其高低字节进行翻转。

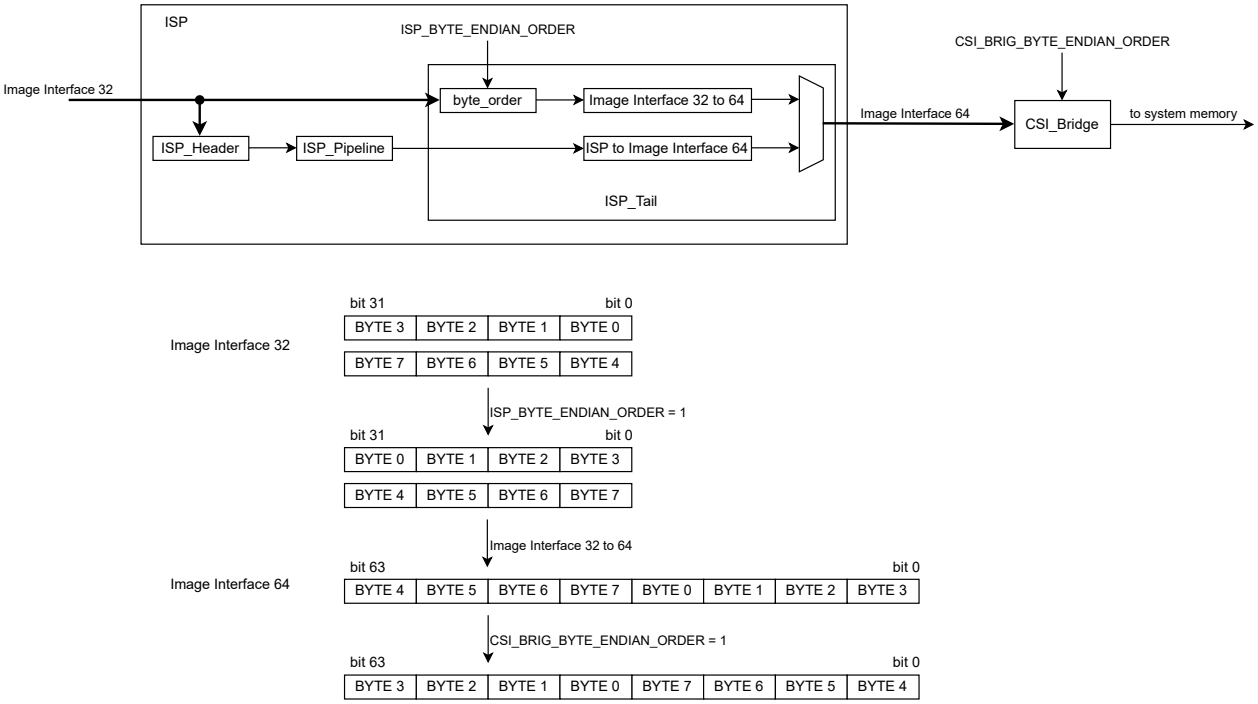


图 38.5-5. 字节顺序

38.6 中断

ESP32-P4 的 ISP 可以生成以下中断信号并发送给 中断矩阵。

- isp_interrupt
- csi_bridge_interrupt

这些中断信号由 ISP 的内部中断源生成。表 38.6-1 列出了 ISP 的内部中断源、中断触发条件、以及生成的中断信号。

表 38.6-1. ISP 内部中断源

内部中断源	触发条件	中断信号
ISP_CROP_ERR_INT	裁剪模块窗口配置错误	isp_interrupt
ISP_WBG_FRAME_INT	白平衡增益处理一帧结束	isp_interrupt
ISP_CROP_FRAME_INT	裁剪处理一帧结束	isp_interrupt
ISP_HEADER_IDI_FRAME_INT	MIPI-CSI 输入一帧结束。该中断不会受 ISP_MIPI_DATA_EN 开关影响	isp_interrupt
ISP_TAIL_IDI_FRAME_INT	ISP_Tail 输出一帧结束	isp_interrupt
ISP_YUV2RGB_FRAME_INT	YUV 转 RGB 处理一帧结束	isp_interrupt
ISP_COLOR_FRAME_INT	对比度 / 色相 / 饱和度 / 亮度处理一帧结束	isp_interrupt
ISP_SHARP_FRAME_INT	锐化处理一帧结束	isp_interrupt
ISP_RGB2YUV_FRAME_INT	RGB 转 YUV 处理一帧结束	isp_interrupt
ISP_GAMMA_FRAME_INT	gamma 处理一帧结束	isp_interrupt

见下页

表 38.6-1 – 接上页

内部中断源	触发条件	中断信号
ISP_CCM_FRAME_INT	颜色矫正矩阵处理一帧结束	isp_interrupt
ISP_DEMOSAIC_FRAME_INT	Demosaic 处理一帧结束	isp_interrupt
ISP_BF_FRAME_INT	拜耳域降噪处理一帧结束	isp_interrupt
ISP_DPC_FRAME_INT	坏点矫正处理一帧结束	isp_interrupt
ISP_LSC_FRAME_INT	镜头阴影矫正处理一帧结束	isp_interrupt
ISP_BLC_FRAME_INT	黑电平矫正处理一帧结束	isp_interrupt
ISP_FRAME_INT	ISP_Pipeline 输出一帧结束	isp_interrupt
ISP_HIST_FDONE_INT	直方图处理一帧结束	isp_interrupt
ISP_AWB_FDONE_INT	自动白平衡统计处理一帧结束	isp_interrupt
ISP_AF_ENV_INT	自动对焦统计监测到对焦场景变化	isp_interrupt
ISP_AF_FDONE_INT	自动对焦统计处理一帧结束	isp_interrupt
ISP_AE_FRAME_DONE_INT	自动曝光统计处理一帧结束	isp_interrupt
ISP_AE_MONITOR_INT	自动曝光统计监测的亮度发生变化	isp_interrupt
ISP_GAMMA_XCOORD_ERR_INT	GAMMA 参数设置错误，ISP_GAMMA_R/G/B_X00-0F 设置的 X 长度和不等于 256	isp_interrupt
ISP_DPC_CHECK_DONE_INT	DPC 静态标定一帧结束	isp_interrupt
ISP_MIPI_HNUM_UNMATCH_INT	宽度 HNUM 设置和 MIPI-CSI 实际输入不匹配	isp_interrupt
ISP_DATA_TYPE_SETTING_ERR_INT	DATA_TYPE 设置错误，和实际不匹配	isp_interrupt
ISP_HVNUM_SETTING_ERR_INT	宽度 HNUM 和高度 VNUM 设置错误：图像尺寸必须是偶数，且 RAW10 输入时，HNUM 需为 4 的倍数	isp_interrupt
ISP_BUF_FULL_INT	ISP_Header 的缓存已满	isp_interrupt
ISP_ASYNC_FIFO_OVF_INT	ISP_Header 的异步 FIFO 溢出	isp_interrupt
ISP_DATA_TYPE_ERR_INT	输入数据使用了非法的 DATA_TYPE（非 RAW8 / RAW10 / RAW12）	isp_interrupt
CSI_BRIG_DMA_CFG_HAS_UPDATED_INT	CSI_Bridge 的 VDMA 配置已经更新	csi_bridge_interrupt
CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT	CSI_Bridge 异步 FIFO 溢出	csi_bridge_interrupt
CSI_BRIG_CSI_BUF_OVERRUN_INT	CSI_Bridge 的缓存溢出	csi_bridge_interrupt
CSI_BRIG_DISCARD_INT	发生帧丢弃	csi_bridge_interrupt
CSI_BRIG_VADR_NUM_LT_INT	配置的 CSI_BRIG_VADR_NUM 小于实际值	csi_bridge_interrupt
CSI_BRIG_VADR_NUM_GT_INT	配置的 CSI_BRIG_VADR_NUM 大于实际值	csi_bridge_interrupt

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 38.8 寄存器列表 以及 38.8 寄存器列表。

38.7 配置流程

38.7.1 ISP 时钟复位配置

1. 打开 ISP、CSI_Bridge 模块的总线时钟；
   - 向 HP_SYS_CLKRST_CSI_BRG_SYS_CLK_EN 写 1 打开 ISP 和 CSI_Bridge 时钟
2. 配置 ISP 工作时钟；
   - 通过 HP_SYS_CLKRST_ISP_CLK_SRC_SEL 选择工作时钟源
   - 通过 HP_SYS_CLKRST_ISP_CLK_DIV_NUM 配置分频系数
   - 向 HP_SYS_CLKRST_ISP_CLK_EN 写 1 打开 ISP 工作时钟
3. 复位 ISP、CSI_Bridge 模块。
   - 向 HP_SYS_CLKRST_RST_EN_CSI_BRG 写 1 再 0 复位 ISP 和 CSI_Bridge

38.7.2 CSI_Bridge 配置

1. 通过 CSI_BRIG_HADR_NUM、CSI_BRIG_VADR_NUM 配置图像宽度和高度。其中，宽度单位为 “64 位”，高度单位为 “行”；
2. 通过 CSI_BRIG_DATA_TYPE_MIN、CSI_BRIG_DATA_TYPE_MAX 设置 DATA_TYPE 有效范围；
3. 通过 CSI_BRIG_DMA_BURST_LEN 配置 burst 长度，该配置需要与 VDMA 对应通道的 SRC_MSIZE（一次 burst 传输长度）以及 SRC_TR_WIDTH（数据宽度）匹配。其中，CSI_Bridge 的数据宽度按照 64 位计算；
4. 若将一幅图像分为多个 block 传输，通过 CSI_BRIG_DMABLK_SIZE 设置每个 block 最大会分成多少次 VDMA 的 burst；
5. 向 CSI_BRIG_CSI_BRIG_EN 写 1 使能 CSI_Bridge。

38.7.3 VDMA 配置

1. 配置对应通道寄存器；
   - SRC/DST_MULTBLK_TYPE 设置为 3（使用链表配置）
   - TT_FC 设置
     - 对于 CSI_Bridge，TT_FC 设置为 4（从外设搬运到 memory，外设作为流控）
     - 对于 memory 输入到 ISP，TT_FC 设置为 6（从 memory 搬运到外设，外设作为流控）
   - HS_SEL 设置
     - 对于 CSI_Bridge，HS_SEL_SRC 设置为 0（外设使用硬件握手），HS_SEL_DST 设置为 1（memory 使用软件握手）
     - 对于 memory 输入到 ISP，HS_SEL_SRC 设置为 1（memory 使用软件握手），HS_SEL_DST 设置为 0（外设使用硬件握手）

- 硬件握手通道设置
  - 对于 CSI_Bridge，SRC_PER 设置为 1
  - 对于 memory 输入到 ISP，DST_PER 设置为 2
- SRC/DST_TR_WIDTH 设置为 3（数据宽度 64）

2. 根据寄存器配置设置链表。

说明：
本章节中的 VDMA 配置只涉及一些关键步骤，详细流程可参考章节 *7 VDMA 控制器 (VDMA)*。
图像尺寸以及起始地址均需 8 字节对齐。

38.7.4 ISP 通用配置

1. 通过 ISP_HADR_NUM、ISP_VADR_NUM 配置输入图像尺寸；
2. 通过 ISP_BAYER_MODE 配置拜耳图像排列模式；
3. 通过 ISP_OUT_TYPE 配置 ISP 输出数据格式；
4. 完成 ISP_Pipeline 各模块配置，详细可参考 38.5.2 对应章节；
5. 通过寄存器 ISP_CNTL_REG 各域打开所需 Pipeline 模块的开关。其中，格式转换相关模块（Demosaic、RGB 转 YUV、YUV 转 RGB）必须在 ISP 使能前就打开相应开关，且必须与最终输出格式匹配。

38.7.5 从 MIPI-CSI 获取图像时打开 ISP

1. 参考小节 38.7.1 配置 ISP 和 CSI_Bridge 时钟复位；
2. 参考小节 38.7.2 配置 CSI_Bridge；
3. 参考小节 38.7.3 配置 VDMA；
4. 配置 MIPI CSI HOST，详见章节 42；
5. 参考小节 38.7.4 配置 ISP 公共配置；
6. 向 ISP_IN_SRC 写 0 选择输入源为 MIPI-CSI；
7. 通过 ISP_DATA_TYPE 配置输入数据类型；
8. 向 ISP_EN 写 1 使能 ISP；
9. 向 ISP_MIPI_DATA_EN 写 1 打开 MIPI-CSI 数据开关。

38.7.6 从 DVP 获取图像时打开 ISP

1. 参考小节 38.7.1 配置 ISP 和 CSI_Bridge 时钟复位；
2. 参考小节 38.7.2 配置 CSI_Bridge；
3. 参考小节 38.7.3 配置 VDMA；
4. 参考小节 38.7.4 配置 ISP 公共配置；
5. 向 ISP_IN_SRC 写 1 选择输入源为 DVP；

6. 通过 ISP_CAM_DATA_TYPE 配置输入数据类型，向 ISP_DATA_TYPE 写 0；
7. 向 ISP_EN 写 1 使能 ISP；
8. 通过 ISP_CAM_CONF_REG 配置 DVP 接口数据模式；
9. 向 ISP_CAM_RESET 写 1，等待一段时间再写 0，经过一段时间完成复位 DVP 接口；
10. 向 ISP_CAM_UPDATE_REG 写 1；
11. 向 ISP_CAM_EN 写 1 使能 DVP 接口；
12. 等待 ISP_CAM_UPDATE_REG 自动清零完成初始化。

38.7.7 从 VDMA 获取图像时打开 ISP

1. 参考小节 38.7.1 配置 ISP 和 CSI_Bridge 时钟复位；
2. 参考小节 38.7.2 配置 CSI_Bridge；
3. 参考小节 38.7.3 配置 VDMA；
4. 参考小节 38.7.4 配置 ISP 公共配置；
5. 向 ISP_IN_SRC 写 2 选择输入源为 VDMA；
6. 通过 ISP_DMA_DATA_TYPE、ISP_DATA_TYPE 配置输入数据类型；
7. 通过 ISP_DMA_RAW_NUM_TOTAL 配置一帧图像数据量，单位为 “64 位”，该值需要等于图像宽度 x 图像高度 x 每个像素位数/64，再向 ISP_DMA_RAW_NUM_TOTAL_SET 写 1 生效配置；
8. 通过 ISP_DMA_BURST_LEN 配置 burst 长度，该配置需要与 VDMA 配置一致；
9. 向 ISP_DMA_UPDATE_REG 写 1 更新 VDMA 配置，等待一段时间完成更新；
10. 向 ISP_DMA_EN 写 1 发送一帧数据。

38.7.8 LSC LUT 配置

LSC 可以在 ISP 开启后再进行配置。在打开 LSC 前需要初始化 LSC 的 LUT。

1. 写 LUT；
   - 向 ISP_LUT_WDATA_REG 写入将要存入 LUT 的数据
   - 写 ISP_LUT_CMD_REG 发起写操作，将数据写入对应地址
2. 读 LUT。
   - 写 ISP_LUT_CMD_REG 发起读操作，读取对应地址的数据
   - 读 ISP_LUT_RDATA_REG 获取读出数据

38.7.9 AE 配置

AE 可以在 ISP 开启后再进行配置。

1. 通过 ISP_AE_SELECT 选择采样点；
2. 设置统计窗口；

- ISP_AE_X_START 设置窗口水平方向起始点
- ISP_AE_X_BSIZE 设置每一个子窗口水平方向尺寸
- ISP_AE_Y_START 设置窗口垂直方向起始点
- ISP_AE_Y_BSIZE 设置每一个子窗口垂直方向尺寸
- ISP_AE_SUBWIN_PIXNUM 设置每一个子窗口包含的像素点个数
- ISP_AE_SUBWIN_RECIP 设置每一个子窗口包含的像素点个数的倒数，20 位小数

3. 设置亮度监测功能；
   - 若需要打开监测，将 ISP_AE_MONITOR_TH 和 ISP_AE_MONITOR_TL 设置为非 0
   - ISP_AE_MONITOR_PERIOD 设置监测统计周期，单位为帧数
4. 向 ISP_AE_EN 写 1 打开 AE；
5. 获取统计结果；
   - 向 ISP_AE_FRAME_DONE_INT_CLR 写 1 清空统计完成状态
   - 向 ISP_AE_UPDATE 写 1 发起一次统计。每次向该寄存器写 1 都会清空监测功能的帧数计数。推荐在需要手动统计时关闭监测功能，统计完成后再打开，以免未及时读出统计结果时，监测统计结果覆盖掉手动统计结果
   - 等待统计完成
     - 可等待 AE_FDONE_INT 导致的 ISP 中断
     - 或者查询 ISP_AE_FRAME_DONE_INT_RAW 状态
   - 读取 ISP_AE_Bxx_MEAN 获得各子窗口统计结果
6. 亮度监测处理。
   - 可等待 AE_MONITOR_INT 导致的 ISP 中断
   - 或者查询 ISP_AE_MONITOR_INT_RAW 状态
   - 触发亮度监测变化后，关闭亮度监测功能，重新进行摄像头曝光，之后再重新打开

38.7.10 AF 配置

AF 可以在 ISP 开启后再进行配置。

1. 设置 A、B、C 三个统计窗口，每个窗口都不要超过 1024 x 1024；
   - ISP_AF_LPOINT_A/B/C 设置三个窗口的左边界
   - ISP_AF_RPOINT_A/B/C 设置三个窗口的右边界
   - ISP_AF_TPOINT_A/B/C 设置三个窗口的上边界
   - ISP_AF_BPOINT_A/B/C 设置三个窗口的下边界
2. 通过 ISP_AF_THRESHOLD 设置清晰度阈值；
3. 设置对焦场景监测功能；
   - 若需打开对焦场景监测功能，将监测周期 ISP_AF_ENV_PERIOD 设置为非 0

- 设置阈值有如下两种方式
  - 若 ISP_AF_ENV_USER_THRESHOLD_SUM 或 ISP_AF_ENV_USER_THRESHOLD_LUM 不为 0，则使用对应设置值作为清晰度、亮度的变化阈值
  - 若 ISP_AF_ENV_USER_THRESHOLD_SUM 或 ISP_AF_ENV_USER_THRESHOLD_LUM 为 0，则使用最近统计结果的清晰度、亮度乘上 ISP_AF_ENV_THRESHOLD（4 位小数）作为变化阈值

4. 设置统计触发方式，每一次触发统计完成后都会更新监测的基准值；
   - 若 ISP_AF_AUTO_UPDATE 为 1，则为自动触发，在 AF 使能后每一帧都会进行一次统计。由于每一次统计都会清空一次监测功能的帧数计数，因此此时监测功能会失效
   - 若 ISP_AF_AUTO_UPDATE 为 0，使用手动触发。监测功能只有在此前提下才有效
5. 向 ISP_AF_EN 写 1 使能 AF；
6. 获取统计结果；
   - 向 ISP_AF_FDONE_INT_CLR 写 1 清空统计完成状态
   - 触发统计
     - 若使用自动触发，每一帧都会进行统计
     - 若使用手动统计，向 ISP_AF_MANUAL_UPDATE 写 1 触发一次统计
   - 等待统计完成
     - 可等待 AF_FDONE_INT 导致的 ISP 中断
     - 或者查询 ISP_AF_FDONE_INT_RAW 状态
   - 读取 ISP_AF_SUMA/B/C 获取清晰度统计结果，读取 ISP_AF_LUMA/B/C 获取亮度统计结果
7. 对焦场景监测处理。
   - 可等待 AF_ENV_INT 导致的 ISP 中断
   - 或者查询 ISP_AF_ENV_INT_RAW 状态
   - 触发对焦场景监测变化后，关闭监测功能，重新进行摄像头对焦，之后再重新打开

38.7.11 AWB 配置

AWB 可以在 ISP 开启后再进行配置。

1. 通过 ISP_AWB_SAMPLE 选择采样点，向 ISP_AWB_MODE 写 1；
2. 设置统计窗口；
   - ISP_AWB_LPOINT 设置窗口的左边界
   - ISP_AWB_RPOINT 设置窗口的右边界
   - ISP_AWB_TPOINT 设置窗口的上边界
   - ISP_AWB_BPOINT 设置窗口的下边界
3. 设置白点约束；
   - ISP_AWB_MAX_LUM 设置白点亮度上限

- ISP_AWB_MIN_LUM 设置白点亮度下限
- ISP_AWB_MAX_RG 设置白点 R/G 比值上限，2 位整数，8 位小数
- ISP_AWB_MIN_RG 设置白点 R/G 比值下限，2 位整数，8 位小数
- ISP_AWB_MAX_BG 设置白点 B/G 比值上限，2 位整数，8 位小数
- ISP_AWB_MIN_BG 设置白点 B/G 比值下限，2 位整数，8 位小数

4. 获取统计结果。
   - 向 ISP_AWB_FDONE_INT_CLR 写 1 清空统计完成状态
   - 向 ISP_AWB_EN 写 0 再写 1 触发统计
   - 等待统计完成
     - 可等待 AWB_FDONE_INT 导致的 ISP 中断
     - 或者查询 ISP_AWB_FDONE_INT_RAW 状态
   - 读取 ISP_AWBO_WHITE_CNT 获取统计窗口内白点数量，读取 ISP_AWBO_ACC_R/G/B 获取统计窗口内白点的 R、G、B 分量累加值

38.7.12 HIST 配置

HIST 可以在 ISP 开启后再进行配置。

1. 通过 ISP_HIST_MODE 选择采样点；
   - 若采样点设置为 RGB，则通过 ISP_HIST_COEFF_R/G/B 设置 RGB 权重，8 位小数
2. 设置统计窗口；
   - ISP_HIST_X_OFFS 设置窗口水平方向起始点
   - ISP_HIST_X_SIZE 设置每一个子窗口水平方向尺寸
   - ISP_HIST_Y_OFFS 设置窗口垂直方向起始点
   - ISP_HIST_Y_SIZE 设置每一个子窗口垂直方向尺寸
3. 设置直方图；
   - 通过 ISP_HIST_SEG_x_x 设置直方图每一个区间划分
   - 通过 ISP_HIST_WEIGHT_xx 设置每一个子窗口权重，8 位小数
4. 向 ISP_HIST_EN 写 1 使能 HIST；
5. 获取统计结果。
   - 向 ISP_HIST_FDONE_INT_CLR 写 1 清空统计完成状态
   - HIST 使能后，每一帧都会进行统计。等待统计完成
     - 可等待 HIST_FDONE_INT 导致的 ISP 中断
     - 或者查询 ISP_HIST_FDONE_INT_RAW 状态
   - 读取 ISP_HIST_BIN_x 获取各区间统计结果

38.7.13 更新 ISP_Pipeline 模块配置

1. 禁用 ISP_CNTL_REG 对应模块；
2. 等待模块中断；
3. 更新模块配置；
4. 重新使能模块。

38.8 寄存器列表

38.8.1 ISP 寄存器列表

本小节的所有地址均为相对于 ISP 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问权限
版本寄存器			
ISP_VER_DATE_REG	版本寄存器	0x0000	R/W
配置寄存器			
ISP_CLK_EN_REG	ISP 时钟控制寄存器	0x0004	R/W
ISP_CNTL_REG	ISP 模块使能寄存器	0x0008	R/W
ISP_HSYNC_CNT_REG	hsync 行间隔控制寄存器	0x000C	R/W
ISP_FRAME_CFG_REG	图像控制寄存器	0x0010	R/W
ISP_CCM_COEF0_REG	CCM 参数寄存器 0	0x0014	R/W
ISP_CCM_COEF1_REG	CCM 参数寄存器 1	0x0018	R/W
ISP_CCM_COEF3_REG	CCM 参数寄存器 3	0x001C	R/W
ISP_CCM_COEF4_REG	CCM 参数寄存器 4	0x0020	R/W
ISP_CCM_COEF5_REG	CCM 参数寄存器 5	0x0024	R/W
ISP_BF_MATRIX_CTRL_REG	BF 像素转矩阵控制寄存器	0x0028	R/W
ISP_BF_SIGMA_REG	BF 降噪控制强度寄存器	0x002C	R/W
ISP_BF_GAU0_REG	BF 降噪模板寄存器 0	0x0030	R/W
ISP_BF_GAU1_REG	BF 降噪模板寄存器 1	0x0034	R/W
ISP_DPC_CTRL_REG	DPC 控制寄存器	0x0038	R/W
ISP_DPC_CONF_REG	DPC 参数寄存器	0x003C	R/W
ISP_DPC_MATRIX_CTRL_REG	DPC 像素转矩阵控制寄存器	0x0040	R/W
ISP_LUT_CMD_REG	LUT 命令寄存器	0x0048	WT
ISP_LUT_WDATA_REG	LUT 写数据寄存器	0x004C	R/W
ISP_LSC_TABLESIZE_REG	LSC 参数控制寄存器	0x0054	R/W
ISP_DEMOSAIC_MATRIX_CTRL_REG	去马赛克像素转矩阵控制寄存器	0x0058	R/W
ISP_DEMOSAIC_GRAD_RATIO_REG	去马赛克参数控制寄存器	0x005C	R/W
ISP_GAMMA_CTRL_REG	gamma 控制寄存器	0x0074	R/W
ISP_GAMMA_RY1_REG	gamma 曲线 R 通道 Y 轴配置寄存器 1	0x0078	R/W
ISP_GAMMA_RY2_REG	gamma 曲线 R 通道 Y 轴配置寄存器 2	0x007C	R/W
ISP_GAMMA_RY3_REG	gamma 曲线 R 通道 Y 轴配置寄存器 3	0x0080	R/W
ISP_GAMMA_RY4_REG	gamma 曲线 R 通道 Y 轴配置寄存器 4	0x0084	R/W
ISP_GAMMA_GY1_REG	gamma 曲线 G 通道 Y 轴配置寄存器 1	0x0088	R/W
ISP_GAMMA_GY2_REG	gamma 曲线 G 通道 Y 轴配置寄存器 2	0x008C	R/W
ISP_GAMMA_GY3_REG	gamma 曲线 G 通道 Y 轴配置寄存器 3	0x0090	R/W
ISP_GAMMA_GY4_REG	gamma 曲线 G 通道 Y 轴配置寄存器 4	0x0094	R/W
ISP_GAMMA_BY1_REG	gamma 曲线 B 通道 Y 轴配置寄存器 1	0x0098	R/W

名称	描述	地址	访问权限
ISP_GAMMA_BY2_REG	gamma 曲线 B 通道 Y 轴配置寄存器 2	0x009C	R/W
ISP_GAMMA_BY3_REG	gamma 曲线 B 通道 Y 轴配置寄存器 3	0x00A0	R/W
ISP_GAMMA_BY4_REG	gamma 曲线 B 通道 Y 轴配置寄存器 4	0x00A4	R/W
ISP_GAMMA_RX1_REG	gamma 曲线 R 通道 X 轴配置寄存器 1	0x00A8	R/W
ISP_GAMMA_RX2_REG	gamma 曲线 R 通道 X 轴配置寄存器 2	0x00AC	R/W
ISP_GAMMA_GX1_REG	gamma 曲线 G 通道 X 轴配置寄存器 1	0x00B0	R/W
ISP_GAMMA_GX2_REG	gamma 曲线 G 通道 X 轴配置寄存器 2	0x00B4	R/W
ISP_GAMMA_BX1_REG	gamma 曲线 B 通道 X 轴配置寄存器 1	0x00B8	R/W
ISP_GAMMA_BX2_REG	gamma 曲线 B 通道 X 轴配置寄存器 2	0x00BC	R/W
ISP_AE_CTRL_REG	AE 控制寄存器	0x00C0	varies
ISP_AE_MONITOR_REG	AE 监测控制寄存器	0x00C4	R/W
ISP_AE_BX_REG	AE 统计窗口 X 方向配置寄存器	0x00C8	R/W
ISP_AE_BY_REG	AE 统计窗口 Y 方向配置寄存器	0x00CC	R/W
ISP_AE_WINPIXNUM_REG	AE 子窗口像素数量配置寄存器	0x00D0	R/W
ISP_AE_WIN_RECIPROCAL_REG	AE 子窗口像素数量倒数配置寄存器	0x00D4	R/W
ISP_SHARP_CTRL0_REG	锐化控制寄存器 0	0x00F4	R/W
ISP_SHARP_FILTER0_REG	锐化低频模板配置寄存器 0	0x00F8	R/W
ISP_SHARP_FILTER1_REG	锐化低频模板配置寄存器 1	0x00FC	R/W
ISP_SHARP_FILTER2_REG	锐化低频模板配置寄存器 2	0x0100	R/W
ISP_SHARP_MATRIX_CTRL_REG	锐化像素转矩阵控制寄存器	0x0104	R/W
ISP_SHARP_CTRL1_REG	锐化控制寄存器 1	0x0108	RO
ISP_DMA_CNTL_REG	ISP VDMA 输入控制寄存器	0x010C	varies
ISP_DMA_RAW_DATA_REG	ISP VDMA 输入数据量配置寄存器	0x0110	varies
ISP_CAM_CNTL_REG	ISP DVP 输入控制寄存器	0x0114	R/W
ISP_CAM_CONF_REG	ISP DVP 输入配置寄存器	0x0118	R/W
ISP_AF_CTRL0_REG	AF 控制寄存器 0	0x011C	varies
ISP_AF_ENV_USER_TH_SUM_REG	AF 场景监测清晰度阈值配置寄存器	0x0128	R/W
ISP_AF_ENV_USER_TH_LUM_REG	AF 场景监测亮度阈值配置寄存器	0x012C	R/W
ISP_AF_THRESHOLD_REG	AF 统计阈值配置寄存器	0x0130	varies
ISP_AF_HSCALE_A_REG	AF 统计窗口 A 水平方向配置寄存器	0x0134	R/W
ISP_AF_VSCALE_A_REG	AF 统计窗口 A 垂直方向配置寄存器	0x0138	R/W
ISP_AF_HSCALE_B_REG	AF 统计窗口 B 水平方向配置寄存器	0x013C	R/W
ISP_AF_VSCALE_B_REG	AF 统计窗口 B 垂直方向配置寄存器	0x0140	R/W
ISP_AF_HSCALE_C_REG	AF 统计窗口 C 水平方向配置寄存器	0x0144	R/W
ISP_AF_VSCALE_C_REG	AF 统计窗口 C 垂直方向配置寄存器	0x0148	R/W
ISP_AWB_MODE_REG	AWB 控制寄存器	0x0164	R/W
ISP_AWB_HSCALE_REG	AWB 统计窗口水平方向配置寄存器	0x0168	R/W
ISP_AWB_VSCALE_REG	AWB 统计窗口垂直方向配置寄存器	0x016C	R/W
ISP_AWB_TH_LUM_REG	AWB 白点亮度阈值配置寄存器	0x0170	R/W
ISP_AWB_TH_RG_REG	AWB 白点 R/G 阈值配置寄存器	0x0174	R/W
ISP_AWB_TH_BG_REG	AWB 白点 B/G 阈值配置寄存器	0x0178	R/W
ISP_COLOR_CTRL_REG	亮度、对比度、饱和度控制寄存器	0x018C	R/W

名称	描述	地址	访问权限
ISP_BLC_VALUE_REG	BLC 控制寄存器	0x0190	R/W
ISP_BLC_CTRL0_REG	BLC 像素拉伸寄存器	0x0194	R/W
ISP_HIST_MODE_REG	HIST 控制寄存器	0x01A4	R/W
ISP_HIST_COEFF_REG	HIST RGB 参数配置寄存器	0x01A8	R/W
ISP_HIST_OFFS_REG	HIST 统计窗口起始点配置寄存器	0x01AC	R/W
ISP_HIST_SIZE_REG	HIST 统计子窗口尺寸配置寄存器	0x01B0	R/W
ISP_HIST_SEG0_REG	HIST 直方图 X 坐标区间配置寄存器 0	0x01B4	R/W
ISP_HIST_SEG1_REG	HIST 直方图 X 坐标区间配置寄存器 1	0x01B8	R/W
ISP_HIST_SEG2_REG	HIST 直方图 X 坐标区间配置寄存器 2	0x01BC	R/W
ISP_HIST_SEG3_REG	HIST 直方图 X 坐标区间配置寄存器 3	0x01C0	R/W
ISP_HIST_WEIGHT0_REG	HIST 子窗口权重配置寄存器 0	0x01C4	R/W
ISP_HIST_WEIGHT1_REG	HIST 子窗口权重配置寄存器 1	0x01C8	R/W
ISP_HIST_WEIGHT2_REG	HIST 子窗口权重配置寄存器 2	0x01CC	R/W
ISP_HIST_WEIGHT3_REG	HIST 子窗口权重配置寄存器 3	0x01D0	R/W
ISP_HIST_WEIGHT4_REG	HIST 子窗口权重配置寄存器 4	0x01D4	R/W
ISP_HIST_WEIGHT5_REG	HIST 子窗口权重配置寄存器 5	0x01D8	R/W
ISP_HIST_WEIGHT6_REG	HIST 子窗口权重配置寄存器 6	0x01DC	R/W
ISP_YUV_FORMAT_REG	YUV 格式控制寄存器	0x0234	R/W
ISP_CROP_CTRL_REG	CROP 控制寄存器	0x0244	WT
ISP_CROP_Y_CAPTURE_REG	CROP Y 方向控制寄存器	0x0248	R/W
ISP_CROP_X_CAPTURE_REG	CROP X 方向控制寄存器	0x024C	R/W
ISP_CROP_ERR_ST_REG	CROP 错误状态寄存器	0x0250	RO
ISP_WBG_COEF_R_REG	WBG R 通道增益寄存器	0x0254	R/W
ISP_WBG_COEF_G_REG	WBG G 通道增益寄存器	0x0258	R/W
ISP_WBG_COEF_B_REG	WBG B 通道增益寄存器	0x025C	R/W
ISP_COLOR_HUE_CTRL_REG	色相控制寄存器	0x0260	R/W
ISP_AWB_BX_REG	AWB 子窗 X 方向控制寄存器	0x0264	R/W
ISP_AWB_BY_REG	AWB 子窗 Y 方向控制寄存器	0x0268	R/W
ISP_STATE_REG	ISP 状态寄存器	0x026C	RO
ISP_SHADOW_REG_CTRL_REG	影子寄存器控制寄存器	0x0270	R/W
状态寄存器			
ISP_DPC_DEADPIX_CNT_REG	DPC 静态标定坏点数量寄存器	0x0044	RO
ISP_LUT_RDATA_REG	LUT 读数据寄存器	0x0050	RO
ISP_AE_BLOCK_MEAN_0_REG	AE 统计结果寄存器 0	0x00D8	RO
ISP_AE_BLOCK_MEAN_1_REG	AE 统计结果寄存器 1	0x00DC	RO
ISP_AE_BLOCK_MEAN_2_REG	AE 统计结果寄存器 2	0x00E0	RO
ISP_AE_BLOCK_MEAN_3_REG	AE 统计结果寄存器 3	0x00E4	RO
ISP_AE_BLOCK_MEAN_4_REG	AE 统计结果寄存器 4	0x00E8	RO
ISP_AE_BLOCK_MEAN_5_REG	AE 统计结果寄存器 5	0x00EC	RO
ISP_AE_BLOCK_MEAN_6_REG	AE 统计结果寄存器 6	0x00F0	RO
ISP_AF_SUM_A_REG	AF 窗口 A 清晰度统计结果寄存器	0x014C	RO
ISP_AF_SUM_B_REG	AF 窗口 B 清晰度统计结果寄存器	0x0150	RO

名称	描述	地址	访问权限
ISP_AF_SUM_C_REG	AF 窗口 C 清晰度统计结果寄存器	0x0154	RO
ISP_AF_LUM_A_REG	AF 窗口 A 亮度统计结果寄存器	0x0158	RO
ISP_AF_LUM_B_REG	AF 窗口 B 亮度统计结果寄存器	0x015C	RO
ISP_AF_LUM_C_REG	AF 窗口 C 亮度统计结果寄存器	0x0160	RO
ISP_AWB0_WHITE_CNT_REG	AWB 白点数量统计结果寄存器	0x017C	RO
ISP_AWB0_ACC_R_REG	AWB R 通道统计结果寄存器	0x0180	RO
ISP_AWB0_ACC_G_REG	AWB G 通道统计结果寄存器	0x0184	RO
ISP_AWB0_ACC_B_REG	AWB B 通道统计结果寄存器	0x0188	RO
ISP_HIST_BIN0_REG	HIST 区间 0 统计结果寄存器	0x01E0	RO
ISP_HIST_BIN1_REG	HIST 区间 1 统计结果寄存器	0x01E4	RO
ISP_HIST_BIN2_REG	HIST 区间 2 统计结果寄存器	0x01E8	RO
ISP_HIST_BIN3_REG	HIST 区间 3 统计结果寄存器	0x01EC	RO
ISP_HIST_BIN4_REG	HIST 区间 4 统计结果寄存器	0x01F0	RO
ISP_HIST_BIN5_REG	HIST 区间 5 统计结果寄存器	0x01F4	RO
ISP_HIST_BIN6_REG	HIST 区间 6 统计结果寄存器	0x01F8	RO
ISP_HIST_BIN7_REG	HIST 区间 7 统计结果寄存器	0x01FC	RO
ISP_HIST_BIN8_REG	HIST 区间 8 统计结果寄存器	0x0200	RO
ISP_HIST_BIN9_REG	HIST 区间 9 统计结果寄存器	0x0204	RO
ISP_HIST_BIN10_REG	HIST 区间 10 统计结果寄存器	0x0208	RO
ISP_HIST_BIN11_REG	HIST 区间 11 统计结果寄存器	0x020C	RO
ISP_HIST_BIN12_REG	HIST 区间 12 统计结果寄存器	0x0210	RO
ISP_HIST_BIN13_REG	HIST 区间 13 统计结果寄存器	0x0214	RO
ISP_HIST_BIN14_REG	HIST 区间 14 统计结果寄存器	0x0218	RO
ISP_HIST_BIN15_REG	HIST 区间 15 统计结果寄存器	0x021C	RO
中断寄存器			
ISP_INT_RAW_REG	原始中断状态寄存器	0x0064	R/SS/WTC
ISP_INT_ST_REG	屏蔽中断状态寄存器	0x0068	RO
ISP_INT_ENA_REG	中断使能寄存器	0x006C	R/W
ISP_INT_CLR_REG	中断清除寄存器	0x0070	WT

38.8.2 MIPI CSI_Bridge 寄存器列表

本小节的所有地址均为相对于 MIPI CSI_Bridge 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问权限
时钟门控寄存器			
CSI_BRIG_CLK_EN_REG	CSI_Bridge 时钟门控寄存器	0x0000	R/W
CSI_Bridge 控制寄存器			

名称	描述	地址	访问权限
CSI_BRIG_CSI_EN_REG	CSI_Bridge 使能寄存器	0x0004	R/W
CSI_BRIG_BUF_FLOW_CTL_REG	CSI_Bridge 缓存控制寄存器	0x000C	varies
CSI_Bridge DMA 控制寄存器			
CSI_BRIG_DMA_REQ_CFG_REG	DMA 请求控制寄存器	0x0008	R/W
CSI_BRIG_DMA_REQ_INTERVAL_REG	DMA 请求间隔控制寄存器	0x002C	R/W
CSI_BRIG_DMABLK_SIZE_REG	DMA 数据块尺寸控制寄存器	0x0030	R/W
CSI_Bridge 帧格式配置寄存器			
CSI_BRIG_DATA_TYPE_CFG_REG	DATA_TYPE 控制寄存器	0x0010	R/W
CSI_BRIG_FRAME_CFG_REG	图像控制寄存器	0x0014	R/W
CSI_BRIG_ENDIAN_MODE_REG	字节排序控制寄存器	0x0018	R/W
CSI_Bridge 中断寄存器			
CSI_BRIG_INT_RAW_REG	CSI_Bridge 原始中断状态寄存器	0x001C	R/WTC/SS
CSI_BRIG_INT_CLR_REG	CSI_Bridge 中断清除寄存器	0x0020	WT
CSI_BRIG_INT_ST_REG	CSI_Bridge 屏蔽中断状态寄存器	0x0024	RO
CSI_BRIG_INT_ENA_REG	CSI_Bridge 中断使能寄存器	0x0028	R/W
CSI HOST 控制寄存器			
CSI_BRIG_HOST_CTRL_REG	CSI HOST 控制寄存器	0x0040	R/W
CSI HOST 颜色控制寄存器			
CSI_BRIG_HOST_CM_CTRL_REG	CSI CM 控制寄存器	0x0048	R/W
CSI_BRIG_HOST_SIZE_CTRL_REG	CSI CM 图像尺寸控制寄存器	0x004C	R/W

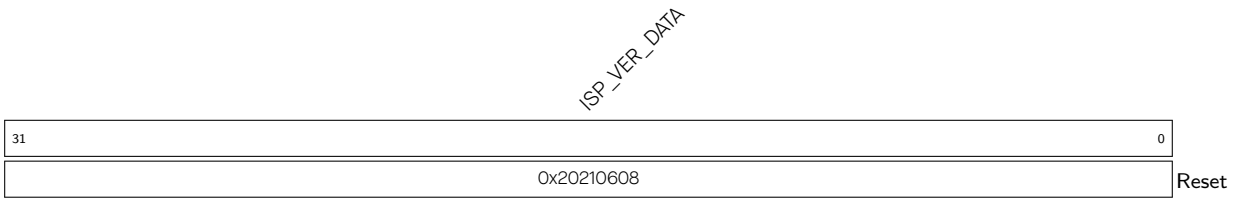
38.9 寄存器

38.9.1 ISP 寄存器

本小节的所有地址均为相对于 ISP 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

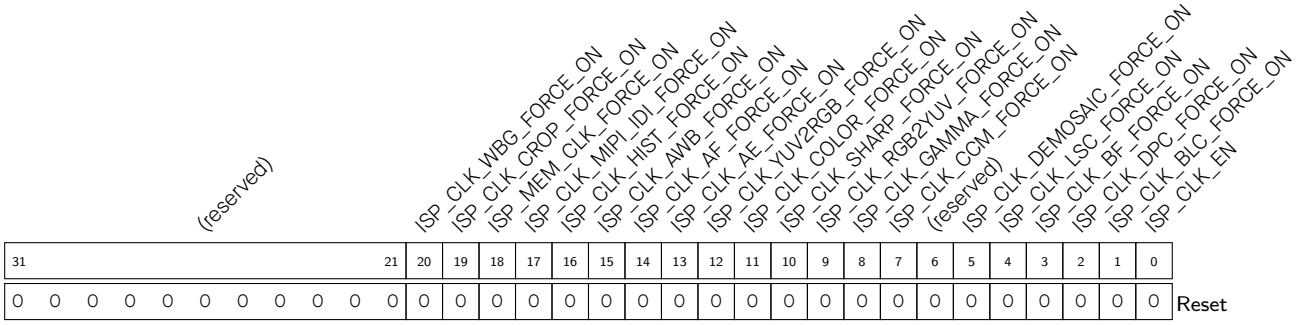
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 38.1. ISP_VER_DATE_REG (0x0000)



ISP_VER_DATA ISP 版本控制寄存器。(R/W)

Register 38.2. ISP_CLK_EN_REG (0x0004)



ISP_CLK_EN 配置是否常开 ISP 寄存器的时钟开关。

0：时钟只会在发生寄存器访问时打开

1：时钟常开

(R/W)

ISP_CLK_BF_FORCE_ON 配置是否强制打开 BF 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_LSC_FORCE_ON 配置是否强制打开 LSC 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_DEMOSAIC_FORCE_ON 配置是否强制打开去马赛克时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_CCM_FORCE_ON 配置是否强制打开 CCM 时钟。

0：不强制打开

1：强制打开

(R/W)

见下页...

Register 38.2. ISP_CLK_EN_REG (0x0004)

接上页...

ISP_CLK_GAMMA_FORCE_ON 配置是否强制打开 gamma 矫正时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_RGB2YUV_FORCE_ON 配置是否强制打开 RGB2YUV 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_SHARP_FORCE_ON 配置是否强制打开锐化时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_COLOR_FORCE_ON 配置是否强制打开对比度 / 色相 / 饱和度 / 亮度时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_YUV2RGB_FORCE_ON 配置是否强制打开 YUV2RGB 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_AE_FORCE_ON 配置是否强制打开 AE 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_AF_FORCE_ON 配置是否强制打开 AF 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_AWB_FORCE_ON 配置是否强制打开 AWB 时钟。

0：不强制打开

1：强制打开

(R/W)

见下页...

Register 38.2. ISP_CLK_EN_REG (0x0004)

接上页...

ISP_CLK_HIST_FORCE_ON 配置是否强制打开 HIST 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_MIPI_IDI_FORCE_ON 配置是否强制打开 MIPI Image Interface 32 输入时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_MEM_CLK_FORCE_ON 配置是否强制打开 ISP 所有 memory 时钟。

0：不强制打开

1：强制打开

(R/W)

ISP_CLK_CROP_FORCE_ON 配置是否强制打开 CROP 时钟。

0：不强制打开

1：强制打开

(R/W)

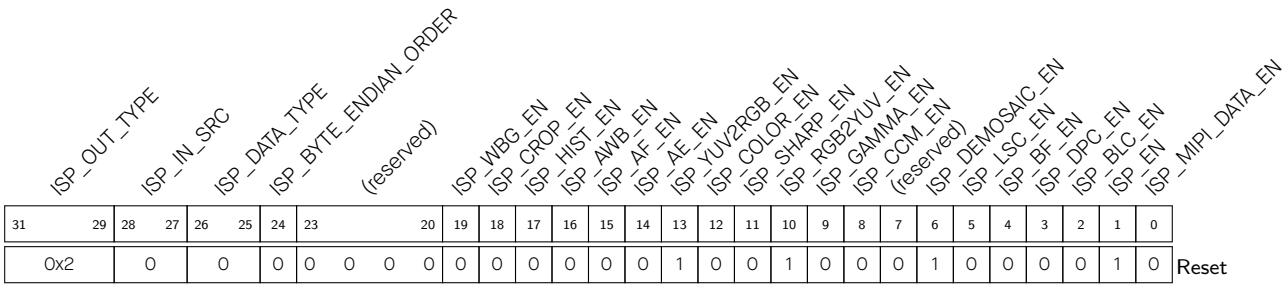
ISP_CLK_WBG_FORCE_ON 配置是否强制打开 WBG 时钟。

0：不强制打开

1：强制打开

(R/W)

Register 38.3. ISP_CNTL_REG (0x0008)



31 29	28 27	26 25	24	23 20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x2	0	0	0	0 0 0 0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	Reset

ISP_MIPI_DATA_EN 配置是否使能 MIPI Image Interface 32 输入。

0：禁用

1：使能

(R/W)

ISP_EN 配置是否使能 ISP。

0：禁用

1：使能

(R/W)

ISP_BLC_EN 配置是否使能 BLC 模块。

0：禁用

1：使能

(R/W)

ISP_DPC_EN 配置是否使能 DPC 模块。

0：禁用

1：使能

(R/W)

ISP_BF_EN 配置是否使能 BF 模块。

0：禁用

1：使能

(R/W)

ISP_LSC_EN 配置是否使能 LSC 模块。

0：禁用

1：使能

(R/W)

ISP_DEMOSAIC_EN 配置是否使能去马赛克模块。

0：禁用

1：使能

(R/W)

ISP_CCM_EN 配置是否使能 CCM 模块。

0：禁用

1：使能

(R/W)

见下页...

Register 38.3. ISP_CNTL_REG (0x0008)

接上页...

ISP_GAMMA_EN 配置是否使能 gamma 矫正模块。

0：禁用

1：使能

(R/W)

ISP_RGB2YUV_EN 配置是否使能 RGB2YUV 模块。

0：禁用

1：使能

(R/W)

ISP_SHARP_EN 配置是否使能锐化模块。

0：禁用

1：使能

(R/W)

ISP_COLOR_EN 配置是否使能对比度 / 色相 / 饱和度 / 亮度模块。

0：禁用

1：使能

(R/W)

ISP_YUV2RGB_EN 配置是否使能 YUV2RGB 模块。

0：禁用

1：使能

(R/W)

ISP_AE_EN 配置是否使能 AE 模块。

0：禁用

1：使能

(R/W)

ISP_AF_EN 配置是否使能 AF 模块。

0：禁用

1：使能

(R/W)

ISP_AWB_EN 配置是否使能 AWB 模块。

0：禁用

1：使能

(R/W)

ISP_HIST_EN 配置是否使能 HIST 模块。

0：禁用

1：使能

(R/W)

见下页...

Register 38.3. ISP_CNTL_REG (0x0008)

接上页...

ISP_CROP_EN 配置是否使能 CROP 模块。

0：禁用

1：使能

(R/W)

ISP_WBG_EN 配置是否使能 WBG 模块。

0：禁用

1：使能

(R/W)

ISP_BYTE_ENDIAN_ORDER 配置 ISP 关闭时的 MIPI Image Interface 32 输入字节排序。

0：csi_data[31:0]

1：[7:0], [15:8], [23:16], [31:24]

(R/W)

ISP_DATA_TYPE 配置输入数据类型。

0：RAW8

1：RAW10

2：RAW12

3：无效

(R/W)

ISP_IN_SRC 配置输入数据源。

0：来自 CSI HOST 的 MIPI Image Interface 32 数据

1：来自 DVP 的数据

2：来自 AXI-DMAC 的数据

3：无效

(R/W)

ISP_OUT_TYPE 配置输出数据类型。

0：RAW8

1：YUV422

2：RGB888

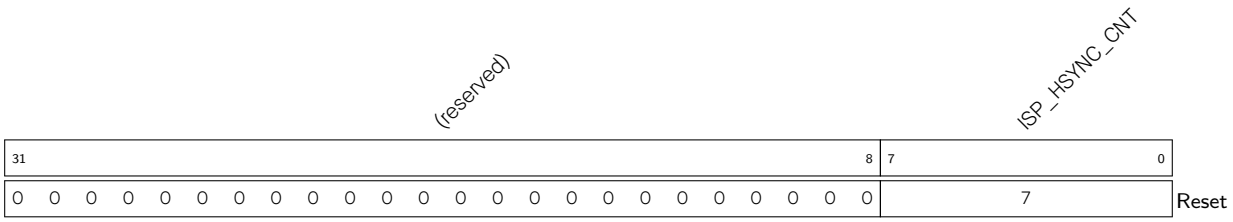
3：YUV420

4：RGB565

其他：无效

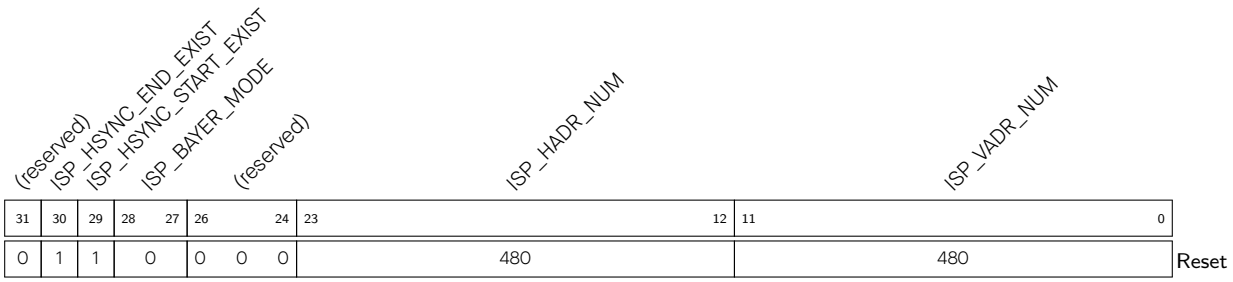
(R/W)

Register 38.4. ISP_HSYNC_CNT_REG (0x000C)



ISP_HSYNC_CNT 配置 Image Interface 32 转 ISP 数据时的 hsync 与之前的 vsync 或 line_end 的间隔，一般使用默认值即可。单位：ISP_CLK 时钟周期数。(R/W)

Register 38.5. ISP_FRAME_CFG_REG (0x0010)



ISP_VADR_NUM 配置输入图像高度，配置为实际行数 - 1。(R/W)

ISP_HADR_NUM 配置输入图像宽度，配置为实际行宽 - 1。(R/W)

ISP_BAYER_MODE 配置输入图像的拜耳模式。

0：BG/GR
1：GB/RG
2：GR/BG
3：RG/GB
(R/W)

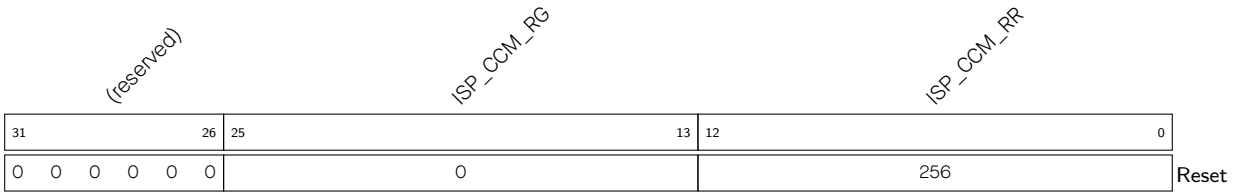
ISP_HSYNC_START_EXIST 配置 hsync_start 包是否存在。

0：不存在
1：存在
(R/W)

ISP_HSYNC_END_EXIST 配置 hsync_end 包是否存在，必须与 ISP_HSYNC_START_EXIST 值相同。

0：不存在
1：存在
(R/W)

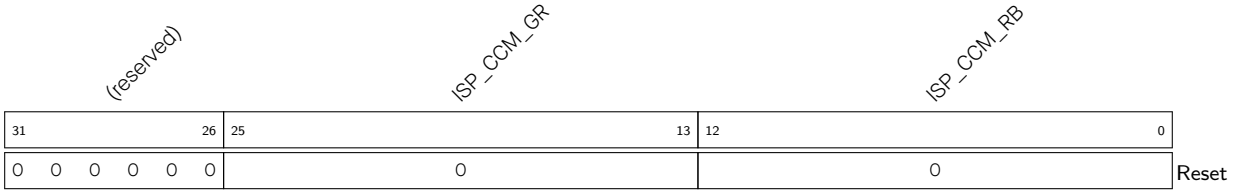
Register 38.6. ISP_CCM_COEF0_REG (0x0014)



ISP_CCM_RR 配置 CCM 的 RR 参数，第 [12] 位为符号位，第 [11:0] 为数值的绝对值，其中 [11:10] 为整数位，[9:0] 为小数位。(R/W)

ISP_CCM_RG 配置 CCM 的 RG 参数，描述详见 ISP_CCM_RR。(R/W)

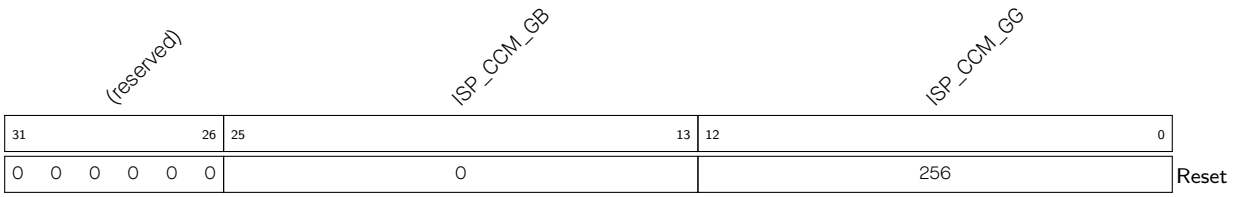
Register 38.7. ISP_CCM_COEF1_REG (0x0018)



ISP_CCM_RB 配置 CCM 的 RB 参数，描述详见 ISP_CCM_RR。(R/W)

ISP_CCM_GR 配置 CCM 的 GR 参数，描述详见 ISP_CCM_RR。(R/W)

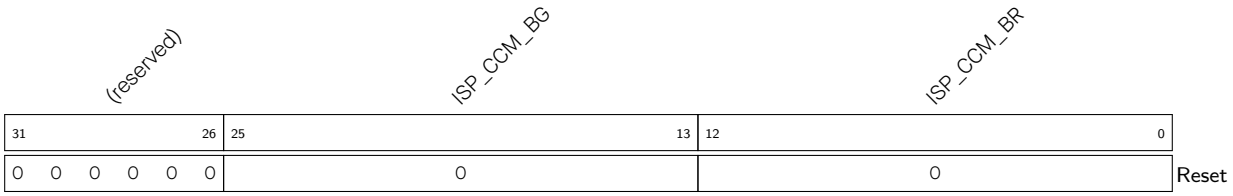
Register 38.8. ISP_CCM_COEF3_REG (0x001C)



ISP_CCM_GG 配置 CCM 的 GG 参数，描述详见 ISP_CCM_RR。(R/W)

ISP_CCM_GB 配置 CCM 的 GB 参数，描述详见 ISP_CCM_RR。(R/W)

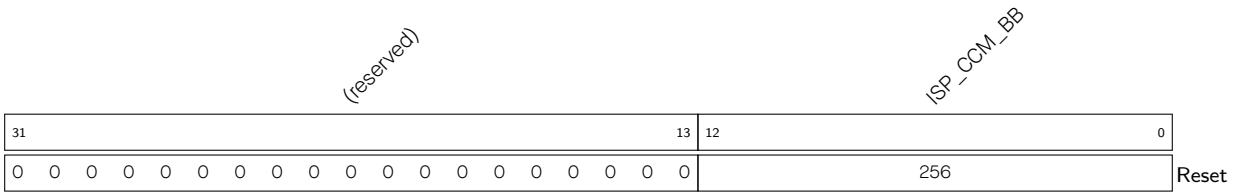
Register 38.9. ISP_CCM_COEF4_REG (0x0020)



ISP_CCM_BR 配置 CCM 的 BR 参数，描述详见 ISP_CCM_RR。(R/W)

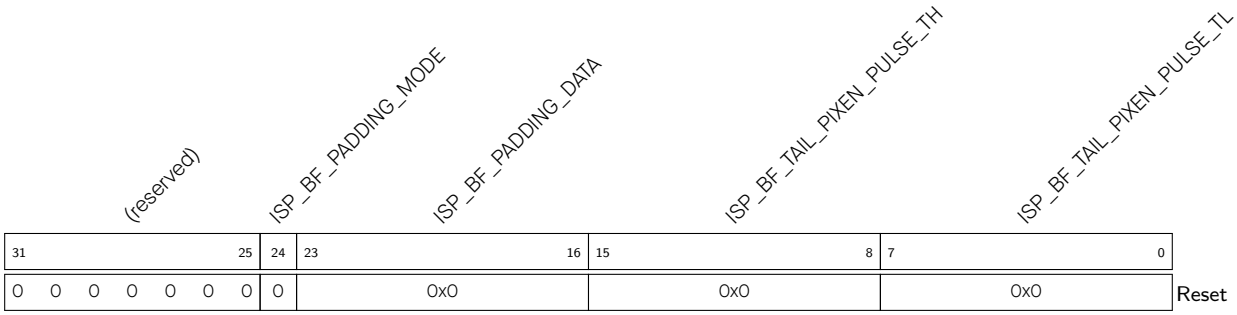
ISP_CCM_BG 配置 CCM 的 BG 参数，描述详见 ISP_CCM_RR。(R/W)

Register 38.10. ISP_CCM_COEF5_REG (0x0024)



ISP_CCM_BB 配置 CCM 的 BB 参数，描述详见 ISP_CCM_RR。(R/W)

Register 38.11. ISP_BF_MATRIX_CTRL_REG (0x0028)



ISP_BF_TAIL_PIXEN_PULSE_TL 配置 BF 像素转矩阵时末尾行数据速率的数据控制周期。

在 ISP_BF_TAIL_PIXEN_PULSE_TL 个时钟周期内，只有前 ISP_BF_TAIL_PIXEN_PULSE_TH 个周期输出有效。谨慎配置，防止配置过大，当前帧的末尾延伸到下一帧。

只有 ISP_BF_TAIL_PIXEN_PULSE_TL 与 ISP_BF_TAIL_PIXEN_PULSE_TH 均不为 0，且 ISP_BF_TAIL_PIXEN_PULSE_TH < ISP_BF_TAIL_PIXEN_PULSE_TL 才会开启该功能。(R/ W)

ISP_BF_TAIL_PIXEN_PULSE_TH 配置 BF 像素转矩阵时末尾行数据速率的数据有效周期，必须小于 ISP_HADR_NUM-1。(R/W)

ISP_BF_PADDING_DATA 配置 BF 像素转矩阵过程中图像边缘扩充时的填充数据。(R/W)

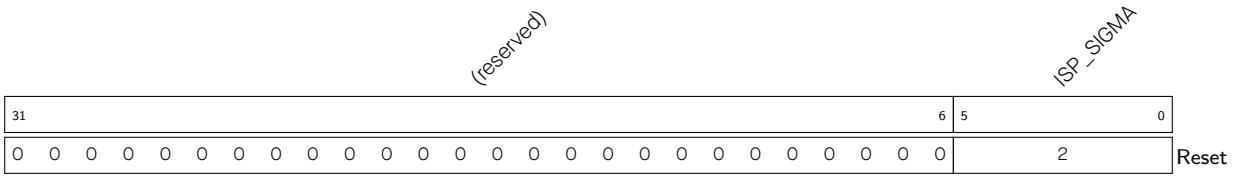
ISP_BF_PADDING_MODE 配置 BF 像素转矩阵过程中图像边缘的扩充方式。

0：使用图像边缘的像素值自动填充

1：使用 ISP_BF_PADDING_DATA 填充

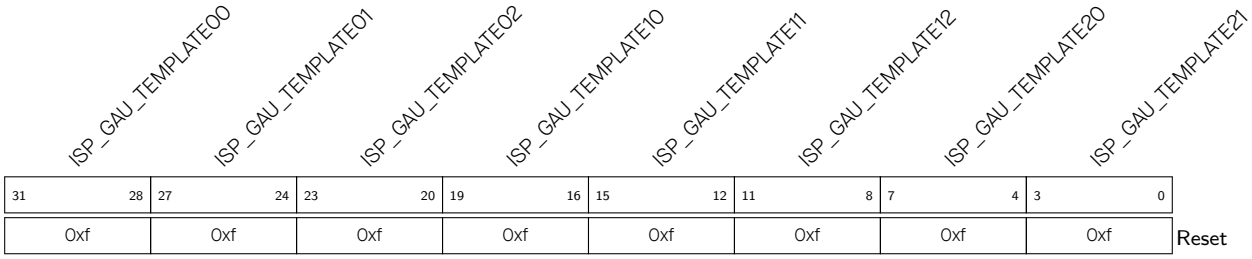
(R/W)

Register 38.12. ISP_BF_SIGMA_REG (0x002C)



ISP_SIGMA 配置 BF 降噪强度，数值 2 至 20 表示强度依次加强，其余值为降噪强度最高。(R/W)

Register 38.13. ISP_BF_GAU0_REG (0x0030)



ISP_GAU_TEMPLATE21 配置 BF 降噪模板坐标 21 的值。(R/W)

ISP_GAU_TEMPLATE20 配置 BF 降噪模板坐标 20 的值。(R/W)

ISP_GAU_TEMPLATE12 配置 BF 降噪模板坐标 12 的值。(R/W)

ISP_GAU_TEMPLATE11 配置 BF 降噪模板坐标 11 的值。(R/W)

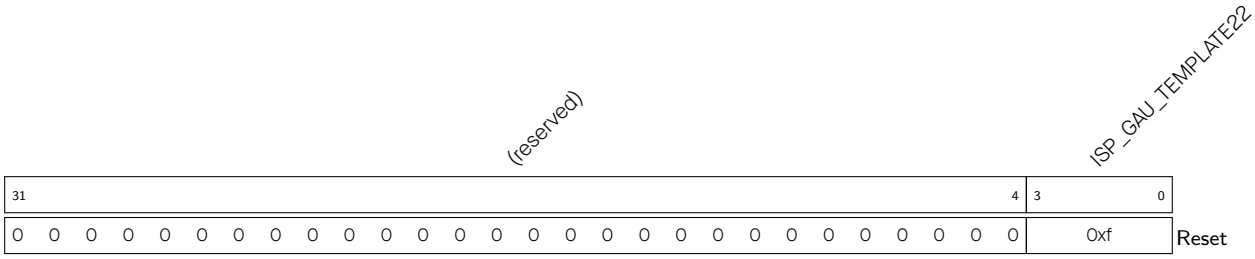
ISP_GAU_TEMPLATE10 配置 BF 降噪模板坐标 10 的值。(R/W)

ISP_GAU_TEMPLATE02 配置 BF 降噪模板坐标 02 的值。(R/W)

ISP_GAU_TEMPLATE01 配置 BF 降噪模板坐标 01 的值。(R/W)

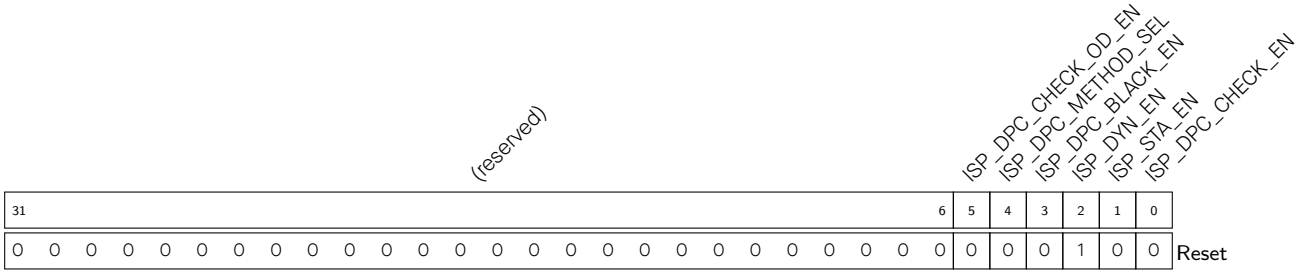
ISP_GAU_TEMPLATE00 配置 BF 降噪模板坐标 00 的值。(R/W)

Register 38.14. ISP_BF_GAU1_REG (0x0034)



ISP_GAU_TEMPLATE22 配置 BF 降噪模板坐标 22 的值。(R/W)

Register 38.15. ISP_DPC_CTRL_REG (0x0038)



ISP_DPC_CHECK_EN 配置是否使能静态标定功能。标定和矫正不能同时打开。

0：关闭

1：使能

(R/W)

ISP_STA_EN 配置是否使能静态矫正功能。

0：关闭

1：使能

(R/W)

ISP_DYN_EN 配置是否使能动态矫正功能。

0：关闭

1：使能

(R/W)

ISP_DPC_BLACK_EN 配置当前进行静态标定的图像类型。

0：白色图像

1：黑色图像

(R/W)

ISP_DPC_METHOD_SEL 配置动态矫正的算法选择。

0：选择算法 0

1：选择算法 1(R/W)

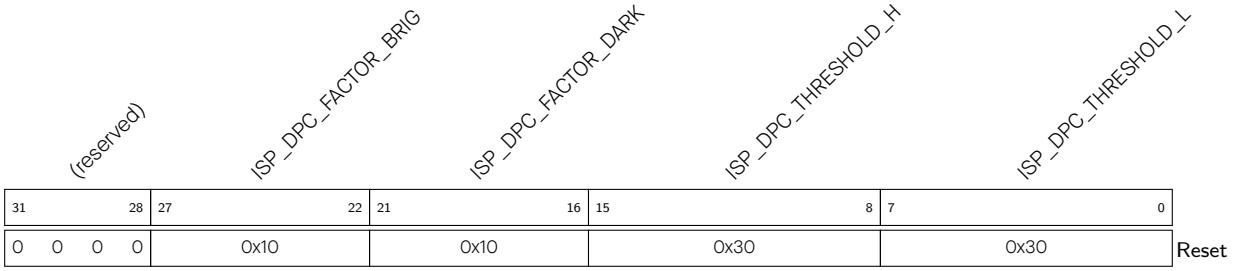
ISP_DPC_CHECK_OD_EN 配置静态标定过程是否输出图像。

0：不输出图像

1：输出图像

(R/W)

Register 38.16. ISP_DPC_CONF_REG (0x003C)



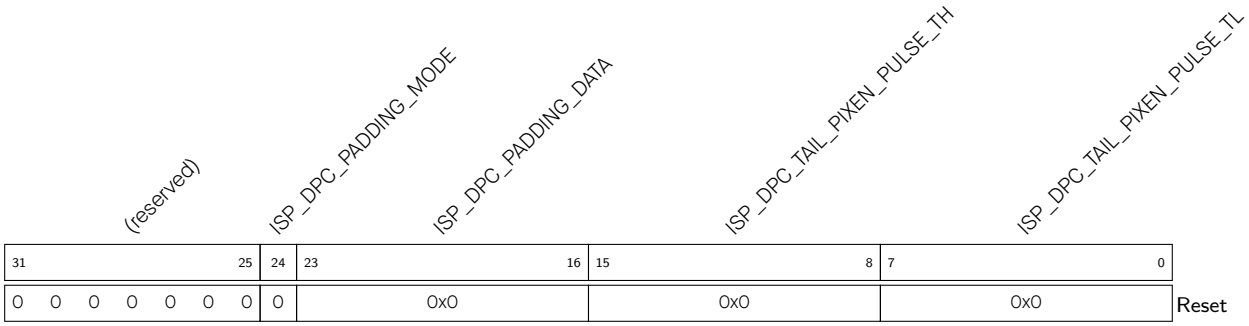
ISP_DPC_THRESHOLD_L 配置静态标定过程中的黑图坏点阈值，或者动态矫正算法 0 中的低阈值（8-bit 有效），或者动态矫正算法 1 中的低阈值（5-bit 有效，最大值 16）。详见小节 38.5.2.2。(R/W)

ISP_DPC_THRESHOLD_H 配置静态标定过程中的白图坏点阈值，或者动态矫正算法 0 中的高阈值（8-bit 有效），或者动态矫正算法 1 中的高阈值（5-bit 有效，最大值 16）。详见小节 38.5.2.2。(R/W)

ISP_DPC_FACTOR_DARK 配置动态矫正算法 1 中的暗点系数。(R/W)

ISP_DPC_FACTOR_BRIG 配置动态矫正算法 1 中的亮点系数。(R/W)

Register 38.17. ISP_DPC_MATRIX_CTRL_REG (0x0040)



ISP_DPC_TAIL_PIXEN_PULSE_TL 配置 DPC 像素转矩阵时末尾行数据速率的数据控制周期。

在 ISP_DPC_TAIL_PIXEN_PULSE_TL 个时钟周期内，只有前 ISP_DPC_TAIL_PIXEN_PULSE_TH 个周期输出有效。谨慎配置，防止配置过大，当前帧的末尾延伸到下一帧。

只有 ISP_DPC_TAIL_PIXEN_PULSE_TL 与 ISP_DPC_TAIL_PIXEN_PULSE_TH 均不为 0，且 ISP_DPC_TAIL_PIXEN_PULSE_TH < ISP_DPC_TAIL_PIXEN_PULSE_TL 才会开启该功能。(R/W)

ISP_DPC_TAIL_PIXEN_PULSE_TH 配置 DPC 像素转矩阵时末尾行数据速率的数据有效周期，必须小于 ISP_HADR_NUM - 1。(R/W)

ISP_DPC_PADDING_DATA 配置 DPC 像素转矩阵过程中图像边缘扩充时的填充数据。(R/W)

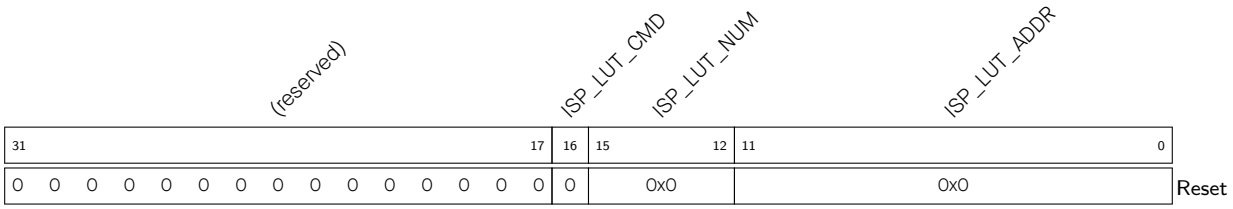
ISP_DPC_PADDING_MODE 配置 DPC 像素转矩阵过程中图像边缘的扩充方式。

0：使用图像边缘的像素值自动填充

1：使用 ISP_DPC_PADDING_DATA 填充

(R/W)

Register 38.18. ISP_LUT_CMD_REG (0x0048)

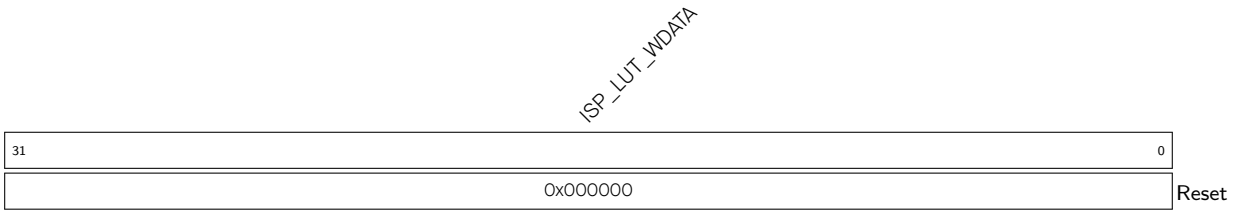


ISP_LUT_ADDR 配置 LUT 地址。当 ISP_LUT_NUM 选择 LSC LUT 时，[11:10] 为 0 表示选择 Gb_B LUT，为 1 表示选择 R_Gr LUT。[9:0] 表示 LUT 的实际地址。
(WT)

ISP_LUT_NUM 配置 LUT 选择。
0：选择 LSC LUT
其他：无效
(WT)

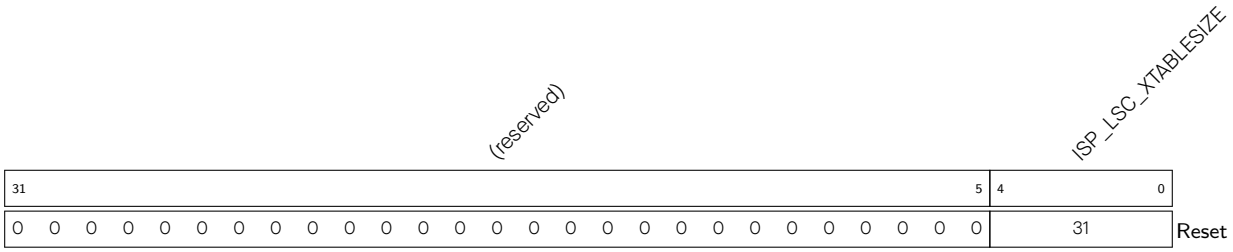
ISP_LUT_CMD 配置 LUT 读写操作类型。
0：读 LUT
1：写 LUT
(WT)

Register 38.19. ISP_LUT_WDATA_REG (0x004C)



ISP_LUT_WDATA 将要写入 LUT 的数据，需在写入 ISP_LUT_CMD_REG 前配置好本寄存器。(R/W)

Register 38.20. ISP_LSC_TABLESIZE_REG (0x0054)



ISP_LSC_XTABLESIZE 配置 LSC 水平方向网格数量。该值由 $fix((line_number-1)/2/32)+2$ 得到，其中 $fix()$ 为取整操作。(R/W)

Register 38.21. ISP_DEMOSAIC_MATRIX_CTRL_REG (0x0058)



ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL 配置去马赛克像素转矩阵时的末尾行数据速率的数据控制周期。

在 ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL 个时钟周期内，只有前 ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH 个周期输出有效。谨慎配置，防止配置过大，当前帧的末尾延伸到下一帧。

只有 ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL 与 ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH 均不为 0，且 ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH < ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TL 才会开启该功能。(R/W)

ISP_DEMOSAIC_TAIL_PIXEN_PULSE_TH 配置去马赛克像素转矩阵时的末尾行数据速率的数据有效周期，必须小于 ISP_HADR_NUM-1。(R/W)

ISP_DEMOSAIC_PADDING_DATA 配置去马赛克像素转矩阵时的图像边缘的扩充时的填充数据。(R/W)

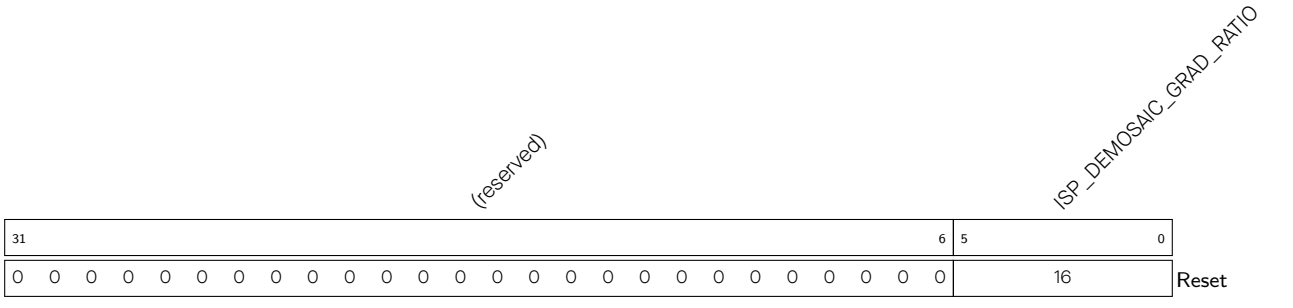
ISP_DEMOSAIC_PADDING_MODE 配置去马赛克像素转矩阵时的图像边缘扩充方式。

0：使用图像边缘的像素值自动填充

1：使用 ISP_DEMOSAIC_PADDING_DATA 进行填充

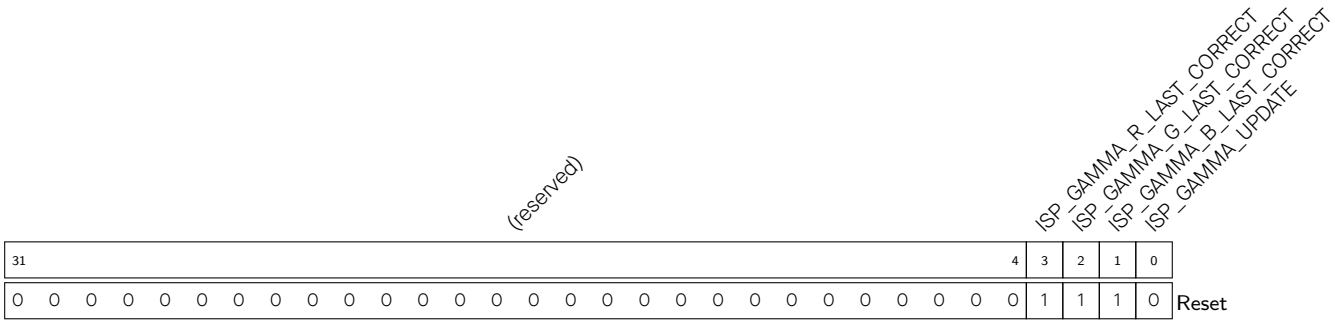
(R/W)

Register 38.22. ISP_DEMOSAIC_GRAD_RATIO_REG (0x005C)



ISP_DEMOSAIC_GRAD_RATIO 配置去马赛克效果，[5:4] 为整数，[3:0] 为小数。(R/W)

Register 38.23. ISP_GAMMA_CTRL_REG (0x0074)



ISP_GAMMA_UPDATE 配置 gamma 曲线更新。向该寄存器域写 1 更新 gamma 曲线参数，更新完成后自动清零。(R/W)

ISP_GAMMA_B_LAST_CORRECT 配置是否使能 B 通道 gamma 曲线 X 轴最后一段区间参数矫正。

0：不使能

1：使能

(R/W)

ISP_GAMMA_G_LAST_CORRECT 配置是否使能 G 通道 gamma 曲线 X 轴最后一段区间参数矫正。

0：不使能

1：使能

(R/W)

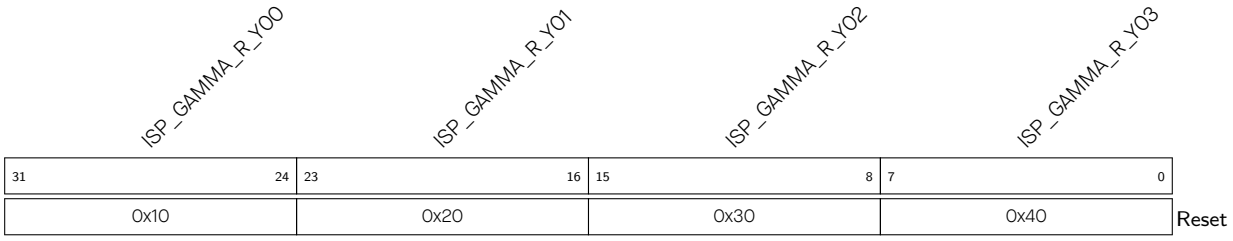
ISP_GAMMA_R_LAST_CORRECT 配置是否使能 R 通道 gamma 曲线 X 轴最后一段区间参数矫正。

0：不使能

1：使能

(R/W)

Register 38.24. ISP_GAMMA_RY1_REG (0x0078)



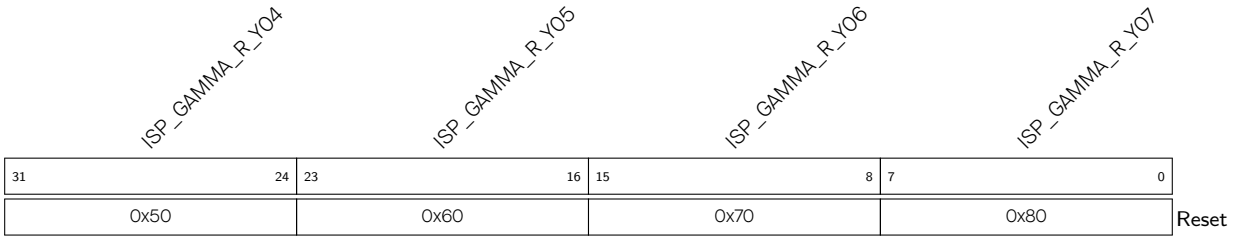
ISP_GAMMA_R_Y03 配置 R 通道 gamma 曲线 Y 轴第 3 个点的值。(R/W)

ISP_GAMMA_R_Y02 配置 R 通道 gamma 曲线 Y 轴第 2 个点的值。(R/W)

ISP_GAMMA_R_Y01 配置 R 通道 gamma 曲线 Y 轴第 1 个点的值。(R/W)

ISP_GAMMA_R_Y00 配置 R 通道 gamma 曲线 Y 轴第 0 个点的值。(R/W)

Register 38.25. ISP_GAMMA_RY2_REG (0x007C)



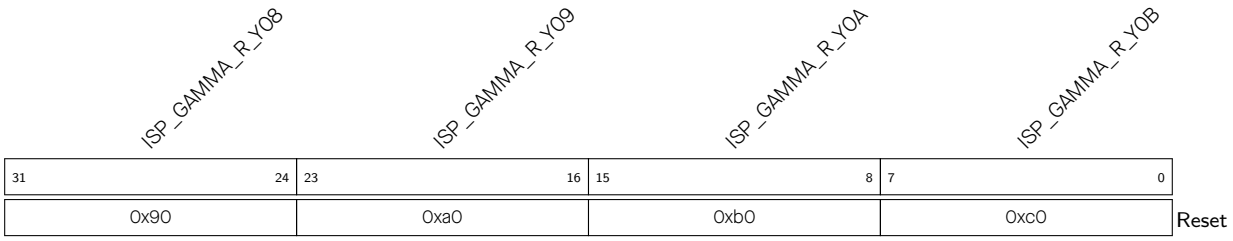
ISP_GAMMA_R_Y07 配置 R 通道 gamma 曲线 Y 轴第 7 个点的值。(R/W)

ISP_GAMMA_R_Y06 配置 R 通道 gamma 曲线 Y 轴第 6 个点的值。(R/W)

ISP_GAMMA_R_Y05 配置 R 通道 gamma 曲线 Y 轴第 5 个点的值。(R/W)

ISP_GAMMA_R_Y04 配置 R 通道 gamma 曲线 Y 轴第 4 个点的值。(R/W)

Register 38.26. ISP_GAMMA_RY3_REG (0x0080)



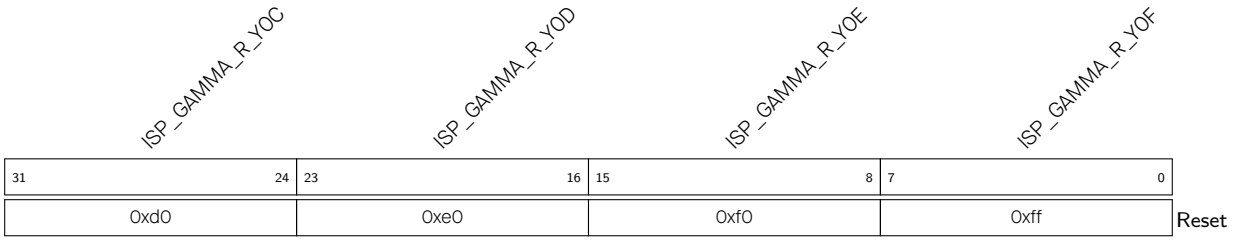
ISP_GAMMA_R_Y0B 配置 R 通道 gamma 曲线 Y 轴第 11 个点的值。(R/W)

ISP_GAMMA_R_Y0A 配置 R 通道 gamma 曲线 Y 轴第 10 个点的值。(R/W)

ISP_GAMMA_R_Y09 配置 R 通道 gamma 曲线 Y 轴第 9 个点的值。(R/W)

ISP_GAMMA_R_Y08 配置 R 通道 gamma 曲线 Y 轴第 8 个点的值。(R/W)

Register 38.27. ISP_GAMMA_RY4_REG (0x0084)



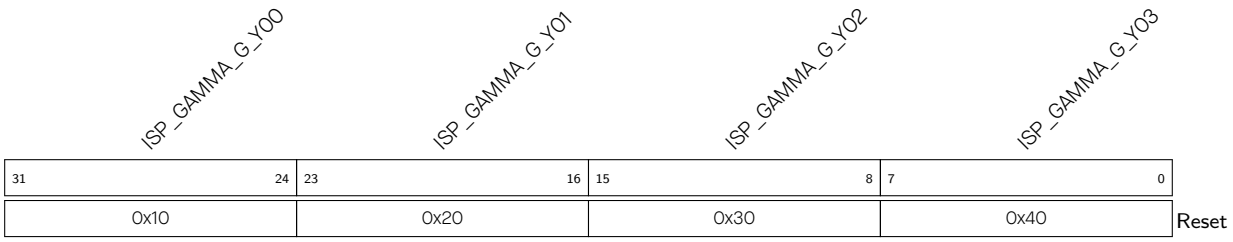
ISP_GAMMA_R_Y0F 配置 R 通道 gamma 曲线 Y 轴第 15 个点的值。(R/W)

ISP_GAMMA_R_Y0E 配置 R 通道 gamma 曲线 Y 轴第 14 个点的值。(R/W)

ISP_GAMMA_R_Y0D 配置 R 通道 gamma 曲线 Y 轴第 13 个点的值。(R/W)

ISP_GAMMA_R_Y0C 配置 R 通道 gamma 曲线 Y 轴第 12 个点的值。(R/W)

Register 38.28. ISP_GAMMA_GY1_REG (0x0088)



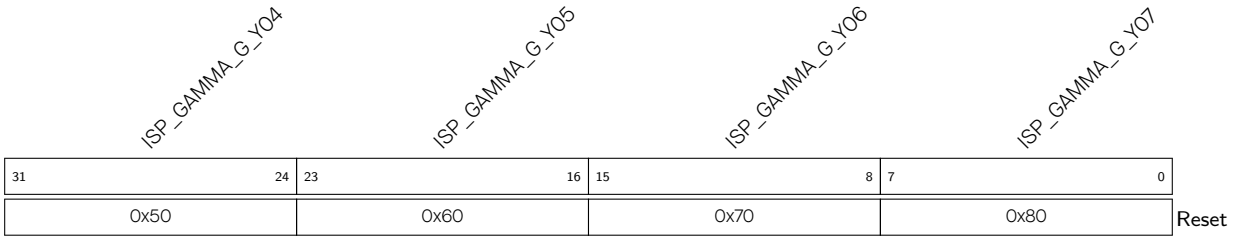
ISP_GAMMA_G_Y03 配置 G 通道 gamma 曲线 Y 轴第 3 个点的值。(R/W)

ISP_GAMMA_G_Y02 配置 G 通道 gamma 曲线 Y 轴第 2 个点的值。(R/W)

ISP_GAMMA_G_Y01 配置 G 通道 gamma 曲线 Y 轴第 1 个点的值。(R/W)

ISP_GAMMA_G_Y00 配置 G 通道 gamma 曲线 Y 轴第 0 个点的值。(R/W)

Register 38.29. ISP_GAMMA_GY2_REG (0x008C)



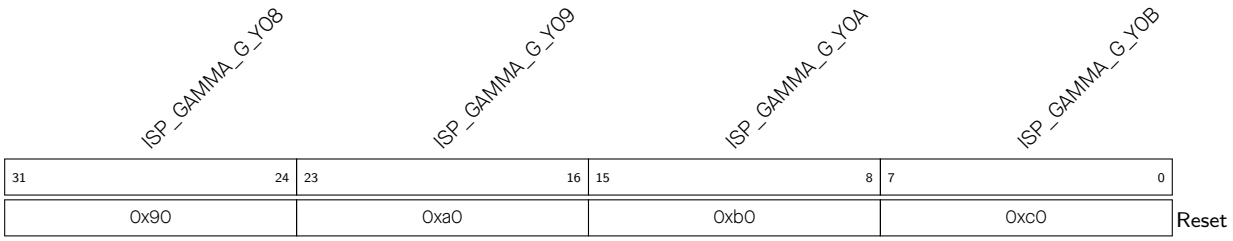
ISP_GAMMA_G_Y07 配置 G 通道 gamma 曲线 Y 轴第 7 个点的值。(R/W)

ISP_GAMMA_G_Y06 配置 G 通道 gamma 曲线 Y 轴第 6 个点的值。(R/W)

ISP_GAMMA_G_Y05 配置 G 通道 gamma 曲线 Y 轴第 5 个点的值。(R/W)

ISP_GAMMA_G_Y04 配置 G 通道 gamma 曲线 Y 轴第 4 个点的值。(R/W)

Register 38.30. ISP_GAMMA_GY3_REG (0x0090)



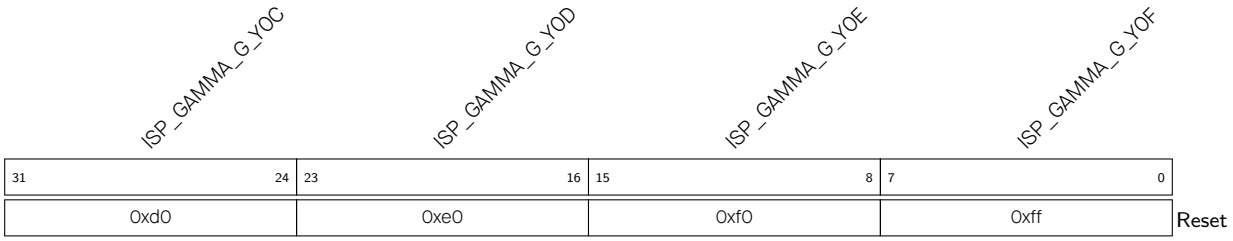
ISP_GAMMA_G_Y0B 配置 G 通道 gamma 曲线 Y 轴第 11 个点的值。(R/W)

ISP_GAMMA_G_Y0A 配置 G 通道 gamma 曲线 Y 轴第 10 个点的值。(R/W)

ISP_GAMMA_G_Y09 配置 G 通道 gamma 曲线 Y 轴第 9 个点的值。(R/W)

ISP_GAMMA_G_Y08 配置 G 通道 gamma 曲线 Y 轴第 8 个点的值。(R/W)

Register 38.31. ISP_GAMMA_GY4_REG (0x0094)



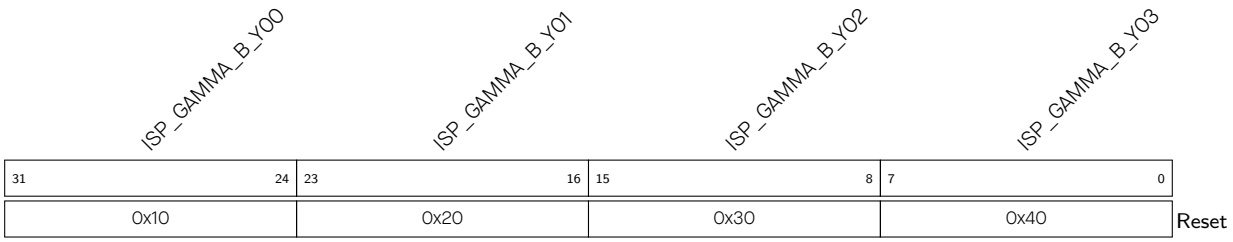
ISP_GAMMA_G_YOF 配置 G 通道 gamma 曲线 Y 轴第 15 个点的值。(R/W)

ISP_GAMMA_G_YOE 配置 G 通道 gamma 曲线 Y 轴第 14 个点的值。(R/W)

ISP_GAMMA_G_YOD 配置 G 通道 gamma 曲线 Y 轴第 13 个点的值。(R/W)

ISP_GAMMA_G_YOC 配置 G 通道 gamma 曲线 Y 轴第 12 个点的值。(R/W)

Register 38.32. ISP_GAMMA_BY1_REG (0x0098)



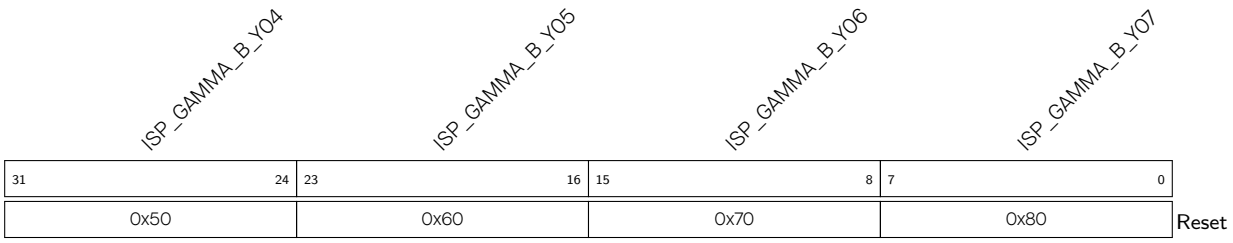
ISP_GAMMA_B_YO3 配置 B 通道 gamma 曲线 Y 轴第 3 个点的值。(R/W)

ISP_GAMMA_B_YO2 配置 B 通道 gamma 曲线 Y 轴第 2 个点的值。(R/W)

ISP_GAMMA_B_YO1 配置 B 通道 gamma 曲线 Y 轴第 1 个点的值。(R/W)

ISP_GAMMA_B_YO0 配置 B 通道 gamma 曲线 Y 轴第 0 个点的值。(R/W)

Register 38.33. ISP_GAMMA_BY2_REG (0x009C)



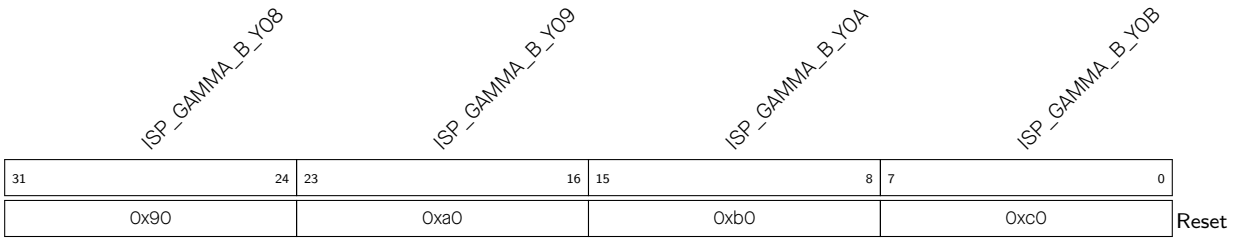
ISP_GAMMA_B_Y07 配置 B 通道 gamma 曲线 Y 轴第 7 个点的值。(R/W)

ISP_GAMMA_B_Y06 配置 B 通道 gamma 曲线 Y 轴第 6 个点的值。(R/W)

ISP_GAMMA_B_Y05 配置 B 通道 gamma 曲线 Y 轴第 5 个点的值。(R/W)

ISP_GAMMA_B_Y04 配置 B 通道 gamma 曲线 Y 轴第 4 个点的值。(R/W)

Register 38.34. ISP_GAMMA_BY3_REG (0x00A0)



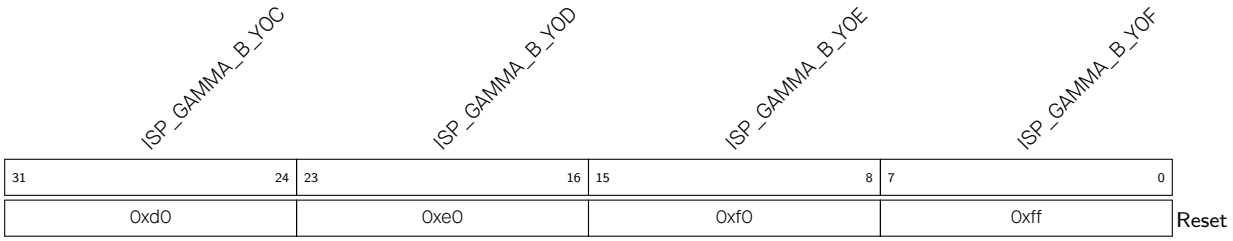
ISP_GAMMA_B_Y0B 配置 B 通道 gamma 曲线 Y 轴第 11 个点的值。(R/W)

ISP_GAMMA_B_Y0A 配置 B 通道 gamma 曲线 Y 轴第 10 个点的值。(R/W)

ISP_GAMMA_B_Y09 配置 B 通道 gamma 曲线 Y 轴第 9 个点的值。(R/W)

ISP_GAMMA_B_Y08 配置 B 通道 gamma 曲线 Y 轴第 8 个点的值。(R/W)

Register 38.35. ISP_GAMMA_BY4_REG (0x00A4)



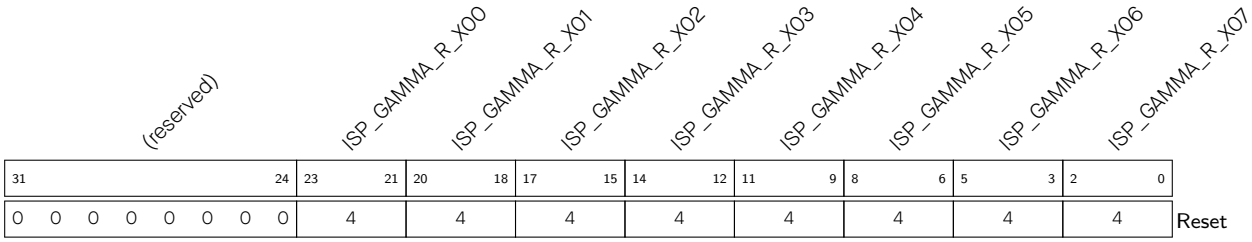
ISP_GAMMA_B_Y0F 配置 B 通道 gamma 曲线 Y 轴第 15 个点的值。(R/W)

ISP_GAMMA_B_Y0E 配置 B 通道 gamma 曲线 Y 轴第 14 个点的值。(R/W)

ISP_GAMMA_B_Y0D 配置 B 通道 gamma 曲线 Y 轴第 13 个点的值。(R/W)

ISP_GAMMA_B_Y0C 配置 B 通道 gamma 曲线 Y 轴第 12 个点的值。(R/W)

Register 38.36. ISP_GAMMA_RX1_REG (0x00A8)



ISP_GAMMA_R_X07 配置 R 通道 gamma 曲线 X 轴第 7 个区间宽度参数，具体配置可以参考 38.5.2.8 小节。(R/W)

ISP_GAMMA_R_X06 配置 R 通道 gamma 曲线 X 轴第 6 个区间宽度参数。(R/W)

ISP_GAMMA_R_X05 配置 R 通道 gamma 曲线 X 轴第 5 个区间宽度参数。(R/W)

ISP_GAMMA_R_X04 配置 R 通道 gamma 曲线 X 轴第 4 个区间宽度参数。(R/W)

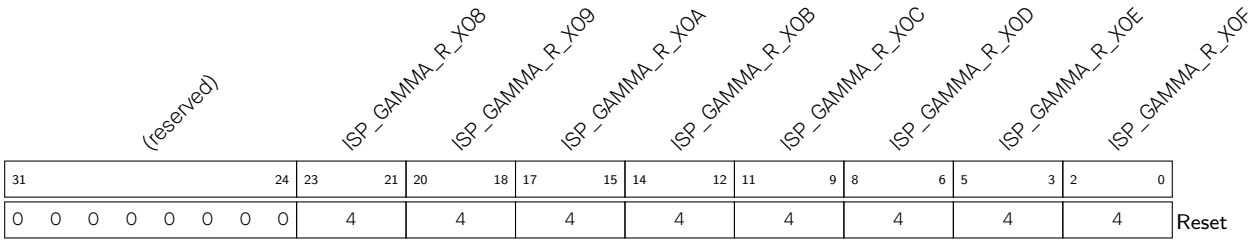
ISP_GAMMA_R_X03 配置 R 通道 gamma 曲线 X 轴第 3 个区间宽度参数。(R/W)

ISP_GAMMA_R_X02 配置 R 通道 gamma 曲线 X 轴第 2 个区间宽度参数。(R/W)

ISP_GAMMA_R_X01 配置 R 通道 gamma 曲线 X 轴第 1 个区间宽度参数。(R/W)

ISP_GAMMA_R_X00 配置 R 通道 gamma 曲线 X 轴第 0 个区间宽度参数。(R/W)

Register 38.37. ISP_GAMMA_RX2_REG (0x00AC)



ISP_GAMMA_R_X0F 配置 R 通道 gamma 曲线 X 轴第 15 个区间宽度参数，具体配置可以参考 38.5.2.8 小节。(R/W)

ISP_GAMMA_R_X0E 配置 R 通道 gamma 曲线 X 轴第 14 个区间宽度参数。(R/W)

ISP_GAMMA_R_X0D 配置 R 通道 gamma 曲线 X 轴第 13 个区间宽度参数。(R/W)

ISP_GAMMA_R_X0C 配置 R 通道 gamma 曲线 X 轴第 12 个区间宽度参数。(R/W)

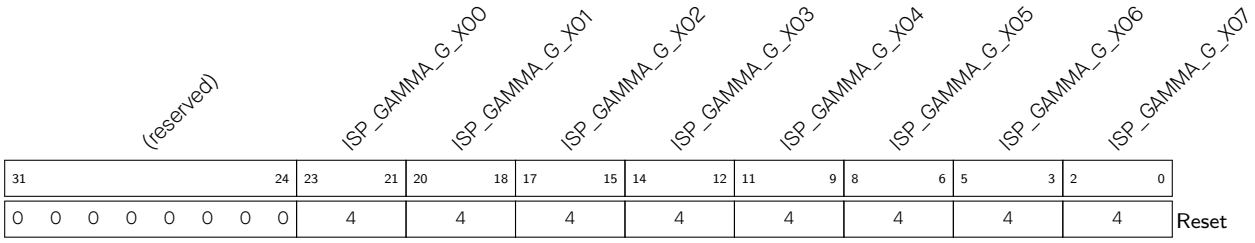
ISP_GAMMA_R_X0B 配置 R 通道 gamma 曲线 X 轴第 11 个区间宽度参数。(R/W)

ISP_GAMMA_R_X0A 配置 R 通道 gamma 曲线 X 轴第 10 个区间宽度参数。(R/W)

ISP_GAMMA_R_X09 配置 R 通道 gamma 曲线 X 轴第 9 个区间宽度参数。(R/W)

ISP_GAMMA_R_X08 配置 R 通道 gamma 曲线 X 轴第 8 个区间宽度参数。(R/W)

Register 38.38. ISP_GAMMA_GX1_REG (0x00B0)



ISP_GAMMA_G_X07 配置 G 通道 gamma 曲线 X 轴第 7 个区间宽度参数，具体配置可以参考 38.5.2.8 小节。(R/W)

ISP_GAMMA_G_X06 配置 G 通道 gamma 曲线 X 轴第 6 个区间宽度参数。(R/W)

ISP_GAMMA_G_X05 配置 G 通道 gamma 曲线 X 轴第 5 个区间宽度参数。(R/W)

ISP_GAMMA_G_X04 配置 G 通道 gamma 曲线 X 轴第 4 个区间宽度参数。(R/W)

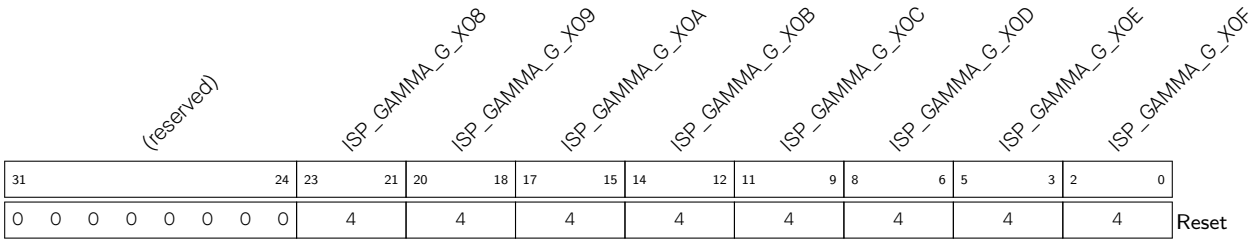
ISP_GAMMA_G_X03 配置 G 通道 gamma 曲线 X 轴第 3 个区间宽度参数。(R/W)

ISP_GAMMA_G_X02 配置 G 通道 gamma 曲线 X 轴第 2 个区间宽度参数。(R/W)

ISP_GAMMA_G_X01 配置 G 通道 gamma 曲线 X 轴第 1 个区间宽度参数。(R/W)

ISP_GAMMA_G_X00 配置 G 通道 gamma 曲线 X 轴第 0 个区间宽度参数。(R/W)

Register 38.39. ISP_GAMMA_GX2_REG (0x00B4)



ISP_GAMMA_G_X0F 配置 G 通道 gamma 曲线 X 轴第 15 个区间宽度参数，具体配置可以参考 38.5.2.8 小节。(R/W)

ISP_GAMMA_G_X0E 配置 G 通道 gamma 曲线 X 轴第 14 个区间宽度参数。(R/W)

ISP_GAMMA_G_X0D 配置 G 通道 gamma 曲线 X 轴第 13 个区间宽度参数。(R/W)

ISP_GAMMA_G_X0C 配置 G 通道 gamma 曲线 X 轴第 12 个区间宽度参数。(R/W)

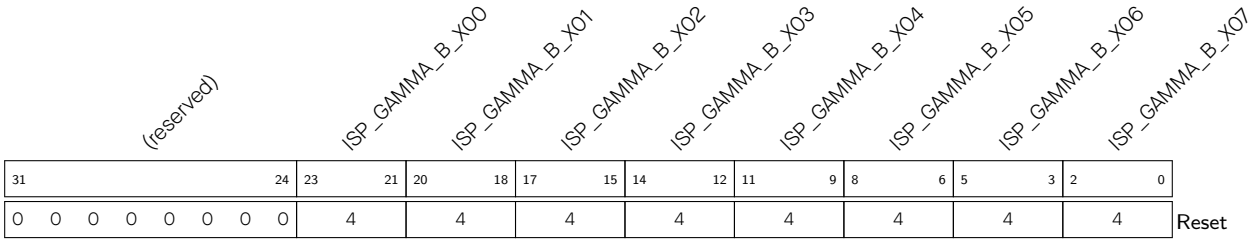
ISP_GAMMA_G_X0B 配置 G 通道 gamma 曲线 X 轴第 11 个区间宽度参数。(R/W)

ISP_GAMMA_G_X0A 配置 G 通道 gamma 曲线 X 轴第 10 个区间宽度参数。(R/W)

ISP_GAMMA_G_X09 配置 G 通道 gamma 曲线 X 轴第 9 个区间宽度参数。(R/W)

ISP_GAMMA_G_X08 配置 G 通道 gamma 曲线 X 轴第 8 个区间宽度参数。(R/W)

Register 38.40. ISP_GAMMA_BX1_REG (0x00B8)



ISP_GAMMA_B_X07 配置 B 通道 gamma 曲线 X 轴第 7 个区间宽度参数，具体配置可以参考 38.5.2.8 小节。(R/W)

ISP_GAMMA_B_X06 配置 B 通道 gamma 曲线 X 轴第 6 个区间宽度参数。(R/W)

ISP_GAMMA_B_X05 配置 B 通道 gamma 曲线 X 轴第 5 个区间宽度参数。(R/W)

ISP_GAMMA_B_X04 配置 B 通道 gamma 曲线 X 轴第 4 个区间宽度参数。(R/W)

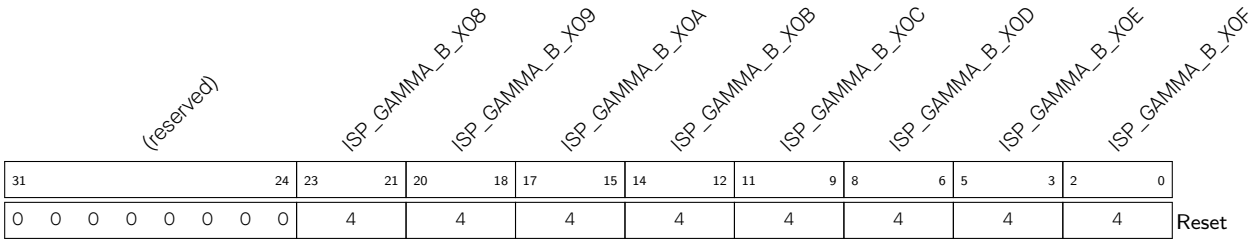
ISP_GAMMA_B_X03 配置 B 通道 gamma 曲线 X 轴第 3 个区间宽度参数。(R/W)

ISP_GAMMA_B_X02 配置 B 通道 gamma 曲线 X 轴第 2 个区间宽度参数。(R/W)

ISP_GAMMA_B_X01 配置 B 通道 gamma 曲线 X 轴第 1 个区间宽度参数。(R/W)

ISP_GAMMA_B_X00 配置 B 通道 gamma 曲线 X 轴第 0 个区间宽度参数。(R/W)

Register 38.41. ISP_GAMMA_BX2_REG (0x00BC)



ISP_GAMMA_B_X0F 配置 B 通道 gamma 曲线 X 轴第 15 个区间宽度参数，具体配置可以参考 38.5.2.8 小节。(R/W)

ISP_GAMMA_B_X0E 配置 B 通道 gamma 曲线 X 轴第 14 个区间宽度参数。(R/W)

ISP_GAMMA_B_X0D 配置 B 通道 gamma 曲线 X 轴第 13 个区间宽度参数。(R/W)

ISP_GAMMA_B_X0C 配置 B 通道 gamma 曲线 X 轴第 12 个区间宽度参数。(R/W)

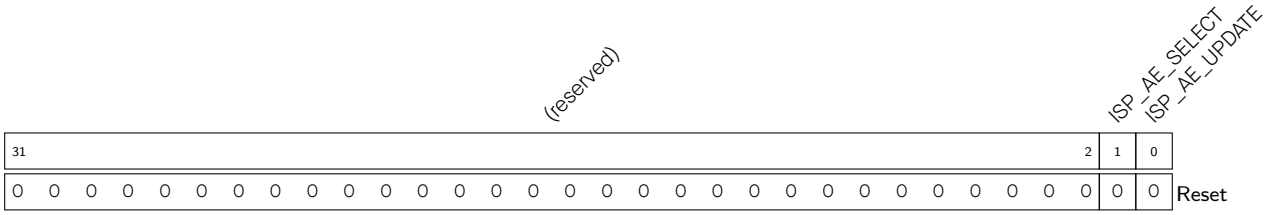
ISP_GAMMA_B_X0B 配置 B 通道 gamma 曲线 X 轴第 11 个区间宽度参数。(R/W)

ISP_GAMMA_B_X0A 配置 B 通道 gamma 曲线 X 轴第 10 个区间宽度参数。(R/W)

ISP_GAMMA_B_X09 配置 B 通道 gamma 曲线 X 轴第 9 个区间宽度参数。(R/W)

ISP_GAMMA_B_X08 配置 B 通道 gamma 曲线 X 轴第 8 个区间宽度参数。(R/W)

Register 38.42. ISP_AE_CTRL_REG (0x00C0)



ISP_AE_UPDATE 配置 AE 统计采样。AE 使能后，写 1 触发一次统计。(WT)

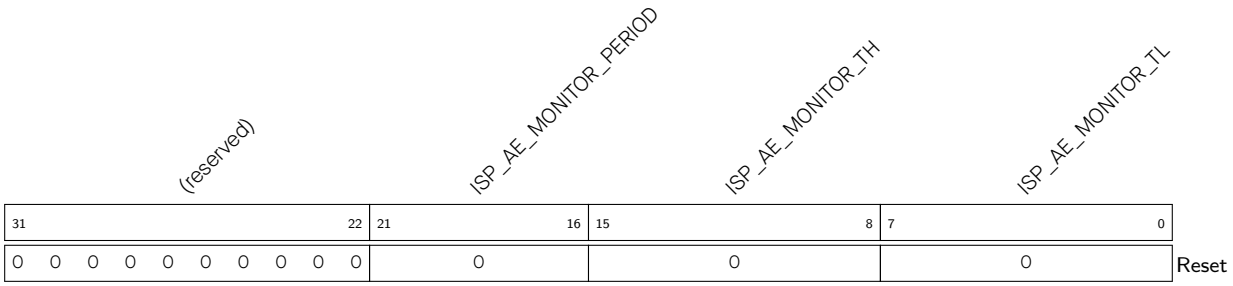
ISP_AE_SELECT 配置 AE 采样点。

0：使用去马赛克模块的输出数据

1：使用 gamma 矫正模块的输出数据

(R/W)

Register 38.43. ISP_AE_MONITOR_REG (0x00C4)

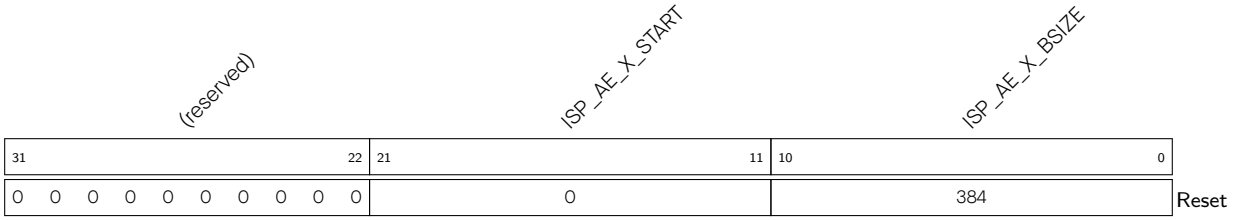


ISP_AE_MONITOR_TL 配置 AE 监测的亮度下阈值。(R/W)

ISP_AE_MONITOR_TH 配置 AE 监测的亮度上阈值。(R/W)

ISP_AE_MONITOR_PERIOD 配置 AE 监测的间隔帧数。(R/W)

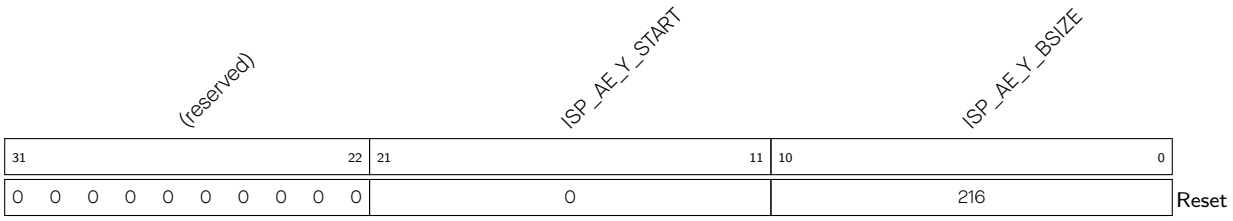
Register 38.44. ISP_AE_BX_REG (0x00C8)



ISP_AE_X_BSIZE 配置每一个子窗口的水平尺寸。(R/W)

ISP_AE_X_START 配置 AE 统计窗口水平方向起始坐标。(R/W)

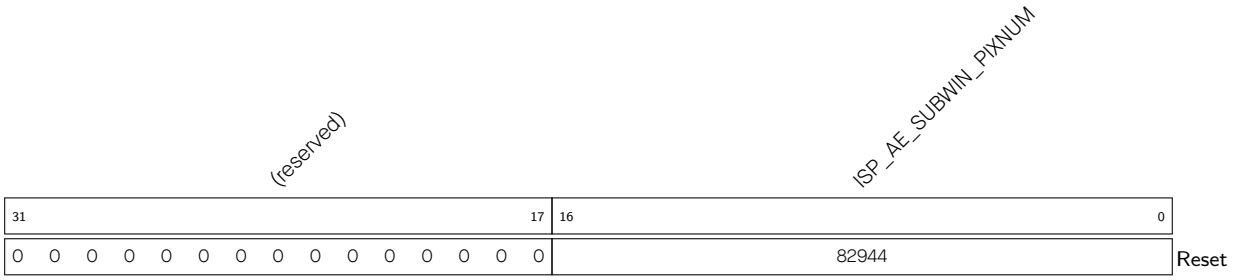
Register 38.45. ISP_AE_BY_REG (0x00CC)



ISP_AE_Y_BSIZE 配置每一个子窗口的垂直尺寸。(R/W)

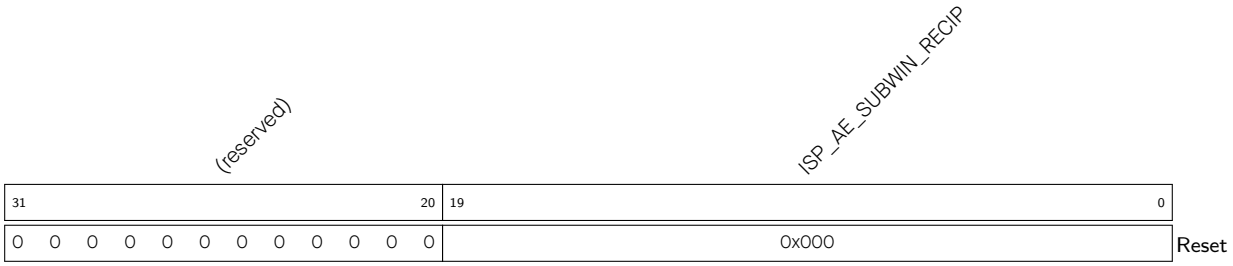
ISP_AE_Y_START 配置 AE 统计窗口垂直方向起始坐标。(R/W)

Register 38.46. ISP_AE_WINPIXNUM_REG (0x00D0)



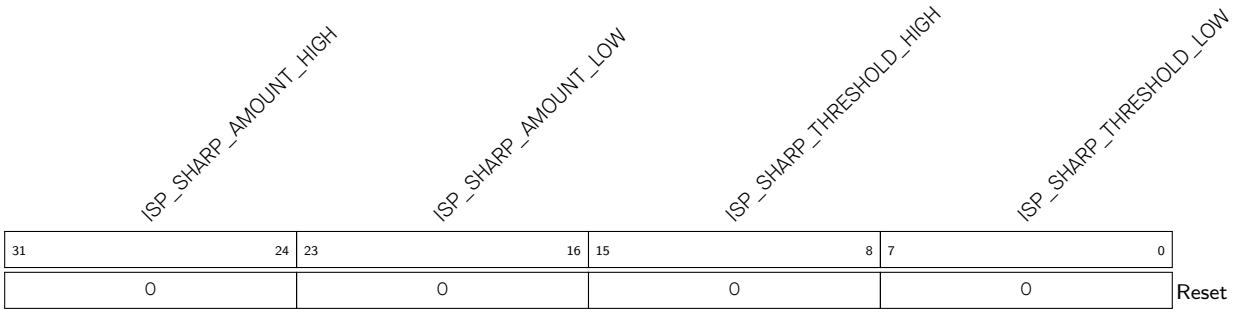
ISP_AE_SUBWIN_PIXNUM 配置每一个子窗口的像素数量。(R/W)

Register 38.47. ISP_AE_WIN_RECIPROCAL_REG (0x00D4)



ISP_AE_SUBWIN_RECIP 配置每一个子窗口的像素数量的倒数，20 位小数，[19:0] 为小数部分。(R/W)

Register 38.48. ISP_SHARP_CTRL0_REG (0x00F4)



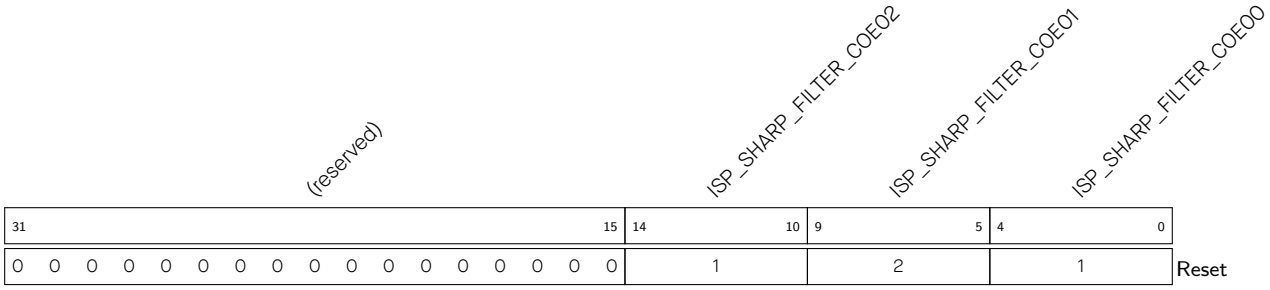
ISP_SHARP_THRESHOLD_LOW 配置锐化高频细节阈值，具体配置可参考章节 38.5.2.10。(R/W)

ISP_SHARP_THRESHOLD_HIGH 配置锐化高频边缘阈值，具体配置可参考章节 38.5.2.10。(R/W)

ISP_SHARP_AMOUNT_LOW 配置锐化高频细节增益，具体配置可参考章节 38.5.2.10。(R/W)

ISP_SHARP_AMOUNT_HIGH 配置锐化高频边缘增益，具体配置可参考章节 38.5.2.10。(R/W)

Register 38.49. ISP_SHARP_FILTER0_REG (0x00F8)

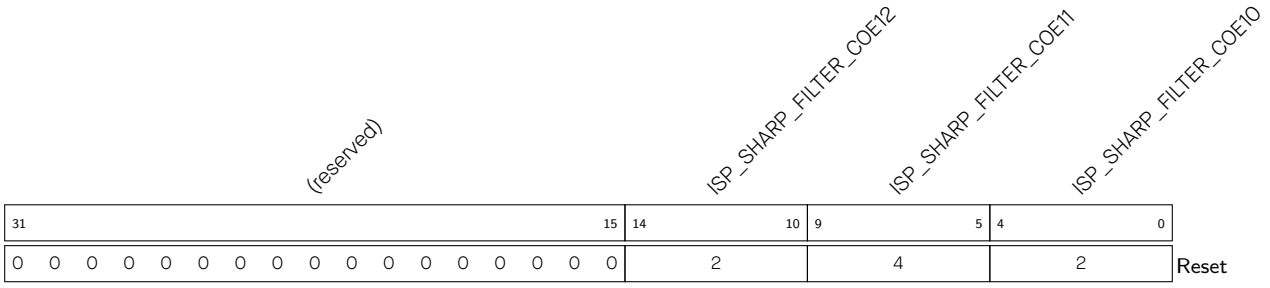


ISP_SHARP_FILTER_COE00 配置锐化低频滤波器模板坐标 00 的值。(R/W)

ISP_SHARP_FILTER_COE01 配置锐化低频滤波器模板坐标 01 的值。(R/W)

ISP_SHARP_FILTER_COE02 配置锐化低频滤波器模板坐标 02 的值。(R/W)

Register 38.50. ISP_SHARP_FILTER1_REG (0x00FC)

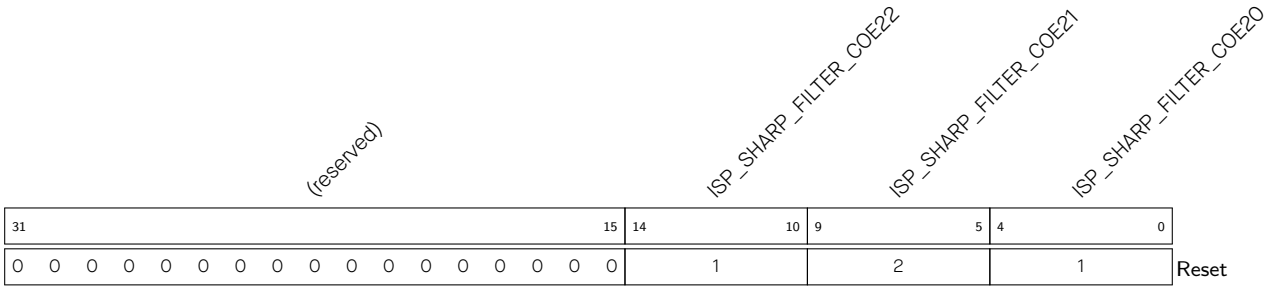


ISP_SHARP_FILTER_COE10 配置锐化低频滤波器模板坐标 10 的值。(R/W)

ISP_SHARP_FILTER_COE11 配置锐化低频滤波器模板坐标 11 的值。(R/W)

ISP_SHARP_FILTER_COE12 配置锐化低频滤波器模板坐标 12 的值。(R/W)

Register 38.51. ISP_SHARP_FILTER2_REG (0x0100)

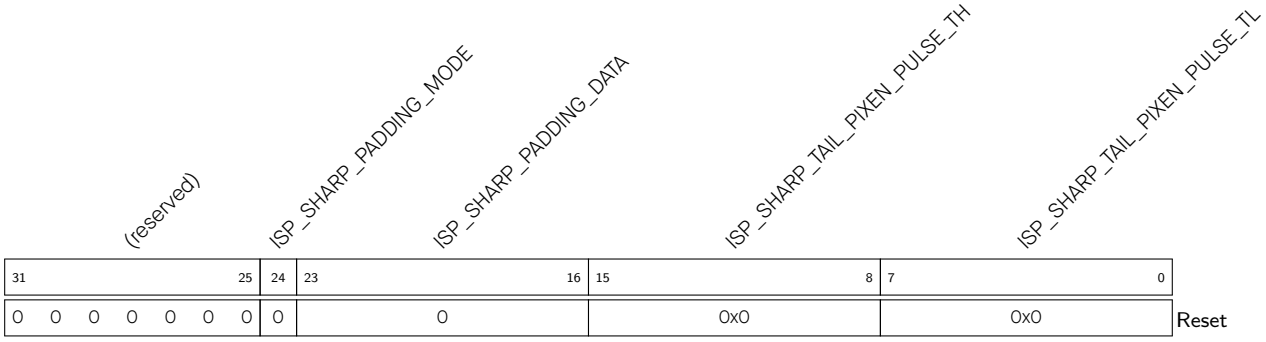


ISP_SHARP_FILTER_COE20 配置锐化低频滤波器模板坐标 20 的值。(R/W)

ISP_SHARP_FILTER_COE21 配置锐化低频滤波器模板坐标 21 的值。(R/W)

ISP_SHARP_FILTER_COE22 配置锐化低频滤波器模板坐标 22 的值。(R/W)

Register 38.52. ISP_SHARP_MATRIX_CTRL_REG (0x0104)



ISP_SHARP_TAIL_PIXEN_PULSE_TL 配置锐化模块像素转矩阵时的末尾行数据速率的数据控制周期。

在 ISP_SHARP_TAIL_PIXEN_PULSE_TL 个时钟周期内，只有前 ISP_SHARP_TAIL_PIXEN_PULSE_TH 个周期输出有效。谨慎配置，防止配置过大，当前帧的末尾延伸到下一帧。

只有 ISP_SHARP_TAIL_PIXEN_PULSE_TL 与 ISP_SHARP_TAIL_PIXEN_PULSE_TH 均不为 0，且 ISP_SHARP_TAIL_PIXEN_PULSE_TH < ISP_SHARP_TAIL_PIXEN_PULSE_TL 才会开启该功能。(R/W)

ISP_SHARP_TAIL_PIXEN_PULSE_TH 配置锐化模块像素转矩阵时的末尾行数据速率的数据有效周期，必须小于 ISP_HADR_NUM-1。(R/W)

ISP_SHARP_PADDING_DATA 配置锐化模块像素转矩阵时的图像边缘的扩充时的填充数据。(R/W)

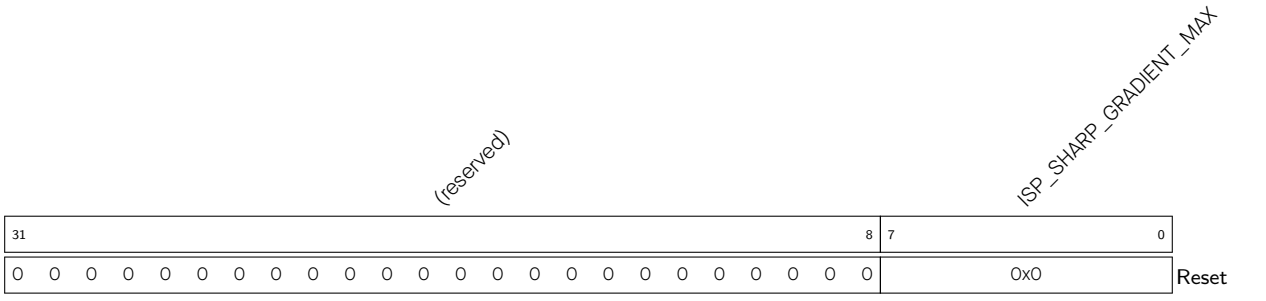
ISP_SHARP_PADDING_MODE 配置锐化模块像素转矩阵时的图像边缘扩充方式。

0：使用图像边缘的像素值自动填充

1：使用 ISP_SHARP_PADDING_DATA 进行填充

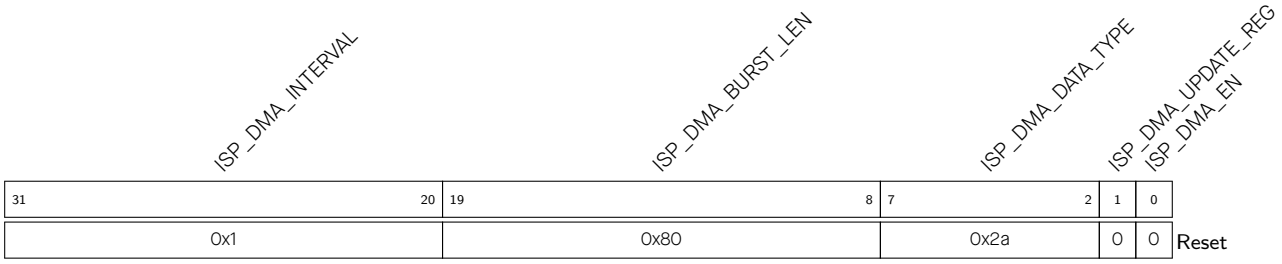
(R/W)

Register 38.53. ISP_SHARP_CTRL1_REG (0x0108)



ISP_SHARP_GRADIENT_MAX 表示锐化高频分量像素最大值，每一帧自动刷新。(RO)

Register 38.54. ISP_DMA_CNTL_REG (0x010C)



ISP_DMA_EN 配置是否触发 VDMA 搬运一帧数据。

0：不触发

1：触发

(WT)

ISP_DMA_UPDATE_REG 配置是否更新 ISP_DMA_DATA_TYPE、ISP_DMA_BURST_LEN、ISP_DMA_INTERVAL 配置。

0：不更新

1：更新，更新完成后自动清零

(R/W)

ISP_DMA_DATA_TYPE 配置搬运数据的 Image Interface data_type。

0x2A：RAW8

0x2B：RAW10

0x2C：RAW12

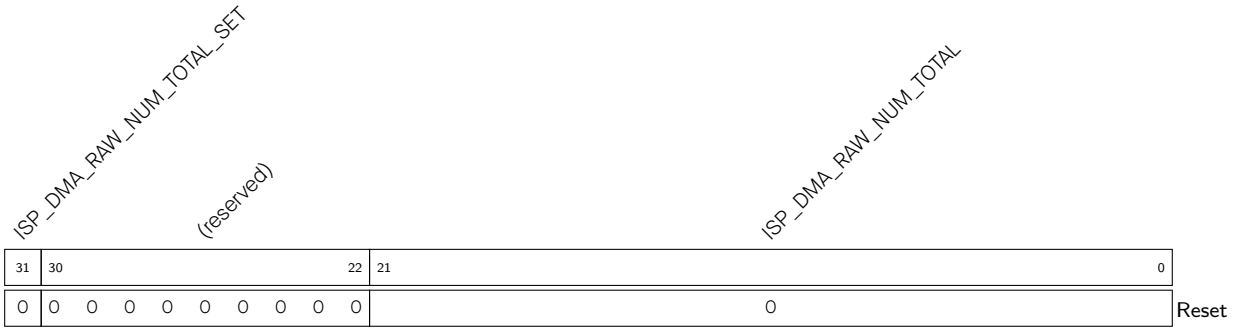
其他：无效

(R/W)

ISP_DMA_BURST_LEN 配置一次 VDMA 搬运的 burst 长度，需要与 VDMA 配置匹配。(R/W)

ISP_DMA_INTERVAL 配置 VDMA 请求间隔。12'b1：1 个周期，12'b11：2 个周期，以此类推。(R/W)

Register 38.55. ISP_DMA_RAW_DATA_REG (0x0110)



ISP_DMA_RAW_NUM_TOTAL 配置一帧图像包含多少个 64 位数据。(R/W)

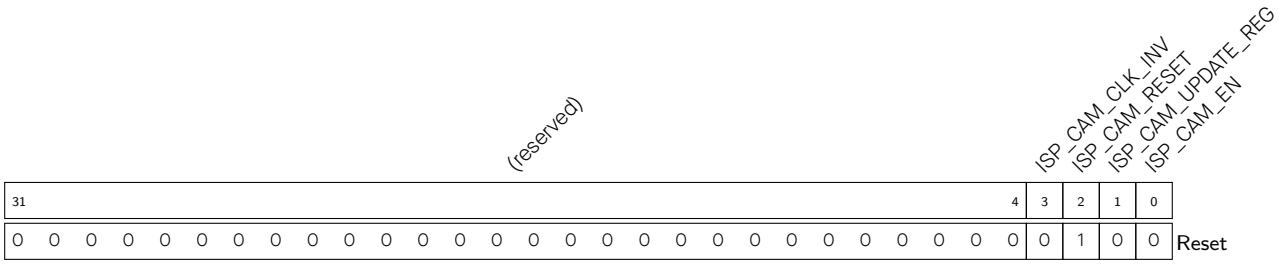
ISP_DMA_RAW_NUM_TOTAL_SET 配置是否更新 ISP_DMA_RAW_NUM_TOTAL 配置。

0：不更新

1：更新

(WT)

Register 38.56. ISP_CAM_CNTL_REG (0x0114)



ISP_CAM_EN 配置是否使能摄像头 DVP 接口输入。

0：禁用

1：使能，开始接收数据

(R/W)

ISP_CAM_UPDATE_REG 配置是否更新寄存器 ISP_CAM_CONF_REG 配置。

0：不更新

1：更新

(R/W)

ISP_CAM_RESET 配置是否复位摄像头 DVP 输入逻辑。

0：不复位

1：复位

(R/W)

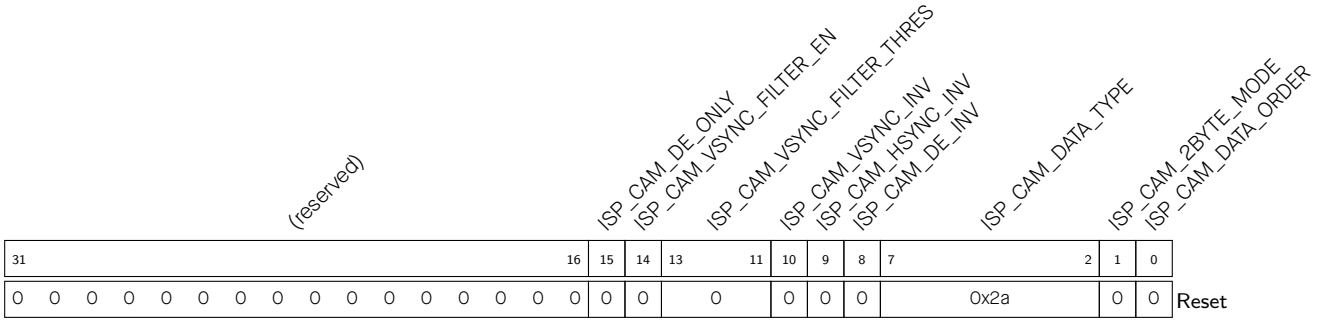
ISP_CAM_CLK_INV 配置是否反相 DVP 时钟。

0：保持不变

1：反相

(R/W)

Register 38.57. ISP_CAM_CONF_REG (0x0118)



ISP_CAM_DATA_ORDER 配置 DVP 输入数据的顺序。

0：cam_data_in

1：cam_data_in[7:0], cam_data_in[15:8]

(R/W)

ISP_CAM_2BYTE_MODE 配置 2 字节模式开关。

0：关闭 2 字节模式

1：打开 2 字节模式

(R/W)

ISP_CAM_DATA_TYPE 配置 DVP 输入数据格式。

0x2a：RAW8

0x2b：RAW10

0x2c：RAW12

(R/W)

ISP_CAM_DE_INV 配置是否反相 DVP de 信号。

0：保持不变

1：反相

(R/W)

ISP_CAM_HSYNC_INV 配置是否反相 DVP hsync 信号。

0：保持不变

1：反相

(R/W)

ISP_CAM_VSYNC_INV 配置是否反相 DVP vsync 信号。

0：保持不变

1：反相

(R/W)

ISP_CAM_VSYNC_FILTER_THRES 配置 vsync 滤波，低于该长度的 vsync 会被过滤。(R/W)

ISP_CAM_VSYNC_FILTER_EN 配置是否使能 vsync 滤波。

0：关闭滤波

1：开启滤波

(R/W)

见下页...

Register 38.57. ISP_CAM_CONF_REG (0x0118)

接上页...

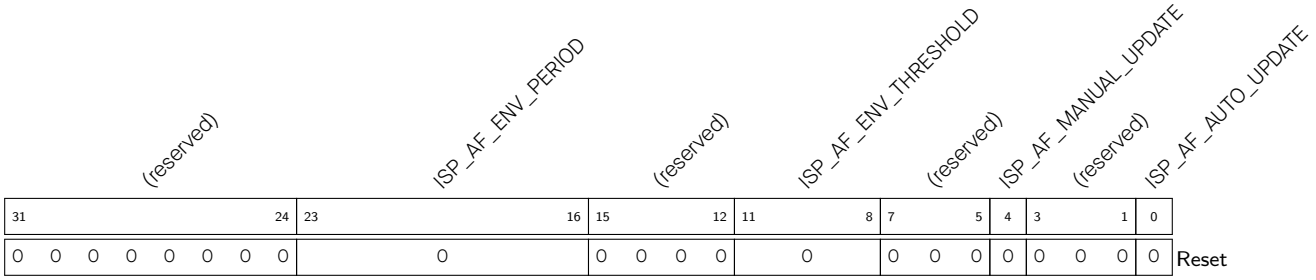
ISP_CAM_DE_ONLY 配置 DVP 接口是否只有 DE 信号，无 HSYNC 信号。

0：有 HSYNC

1：无 HSYNC

(R/W)

Register 38.58. ISP_AF_CTRL0_REG (0x011C)



ISP_AF_AUTO_UPDATE 配置是否打开 AF 自动统计。

0：关闭自动统计，需要手动触发统计

1：打开自动统计，AF 使能后每一帧都会进行统计

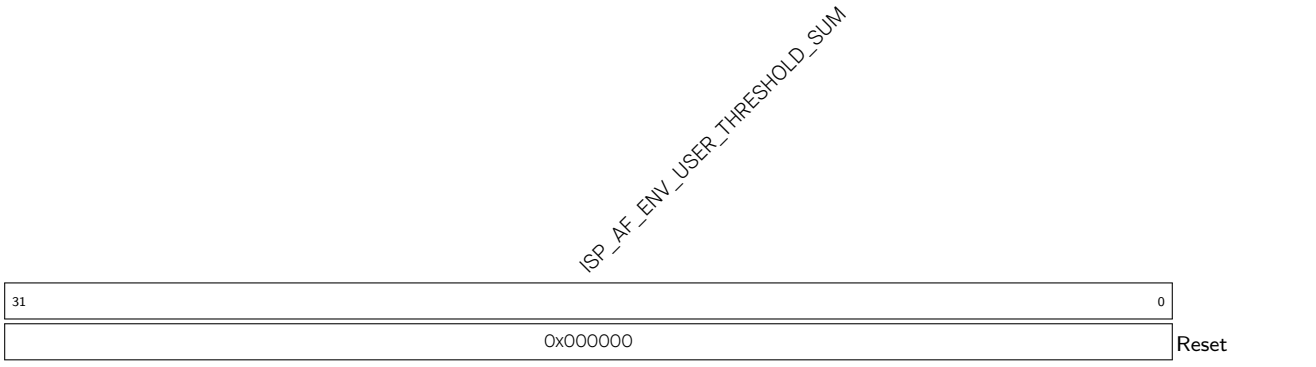
(R/W)

ISP_AF_MANUAL_UPDATE 配置 AF 手动统计。AF 自动统计关闭后，向该寄存器域写 1 触发一次统计。(WT)

ISP_AF_ENV_THRESHOLD 配置 AF 场景监测阈值。在 ISP_AF_ENV_USER_THRESHOLD_SUM 或 ISP_AF_ENV_USER_THRESHOLD_LUM 为 0 时，当清晰度、亮度的变化连续两次大于基准值与该寄存器域乘积，会触发场景监测中断。4 位小数。(R/W)

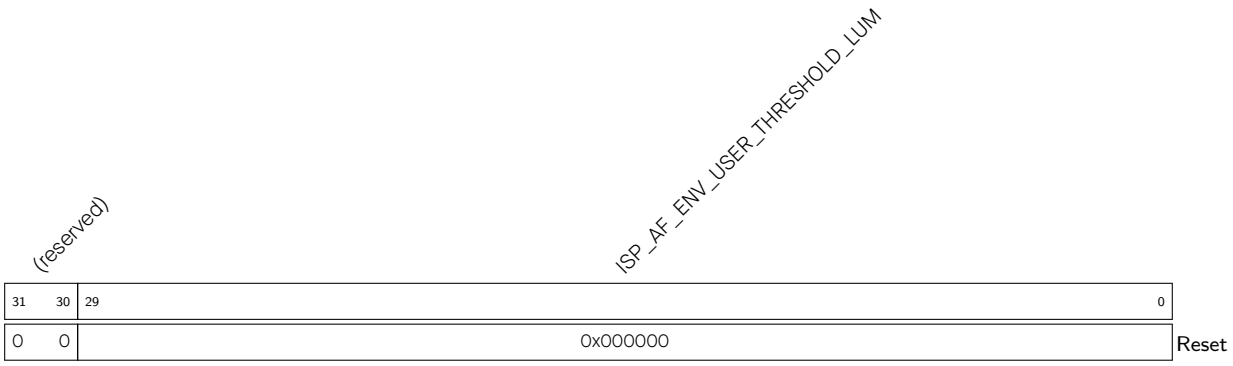
ISP_AF_ENV_PERIOD 配置 AF 场景监测的统计间隔帧数。设置为 0 关闭场景监测功能。(R/W)

Register 38.59. ISP_AF_ENV_USER_TH_SUM_REG (0x0128)



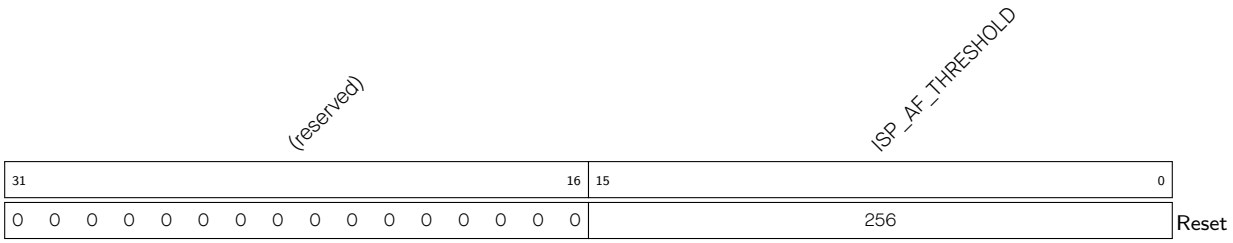
ISP_AF_ENV_USER_THRESHOLD_SUM 配置 AF 场景监测的清晰度变化绝对值阈值。(R/W)

Register 38.60. ISP_AF_ENV_USER_TH_LUM_REG (0x012C)



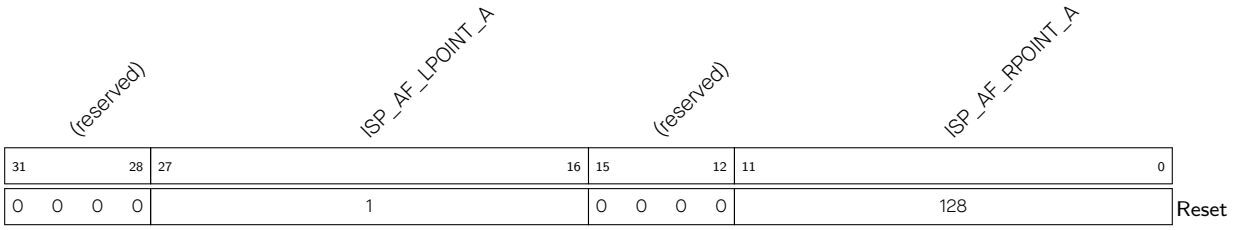
ISP_AF_ENV_USER_THRESHOLD_LUM 配置 AF 场景监测的亮度变化绝对值阈值。(R/W)

Register 38.61. ISP_AF_THRESHOLD_REG (0x0130)



ISP_AF_THRESHOLD 配置 AF 统计的清晰度阈值。(R/W)

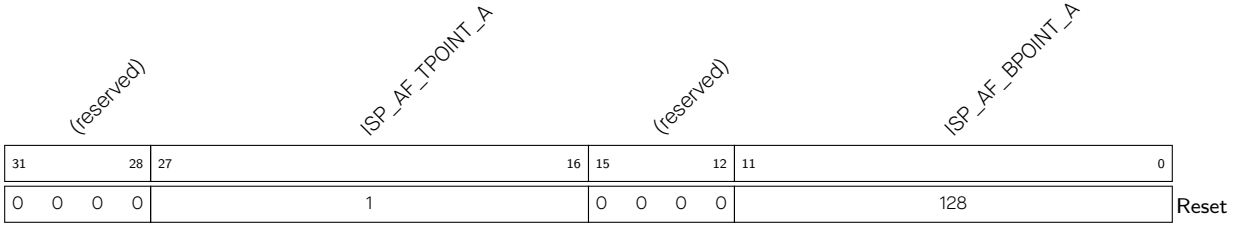
Register 38.62. ISP_AF_HSCALE_A_REG (0x0134)



ISP_AF_RPOINT_A 配置 AF 统计窗口 A 的右边界坐标，推荐小于 HNUM-1。(R/W)

ISP_AF_LPOINT_A 配置 AF 统计窗口 A 的左边界坐标，推荐大于等于 2。(R/W)

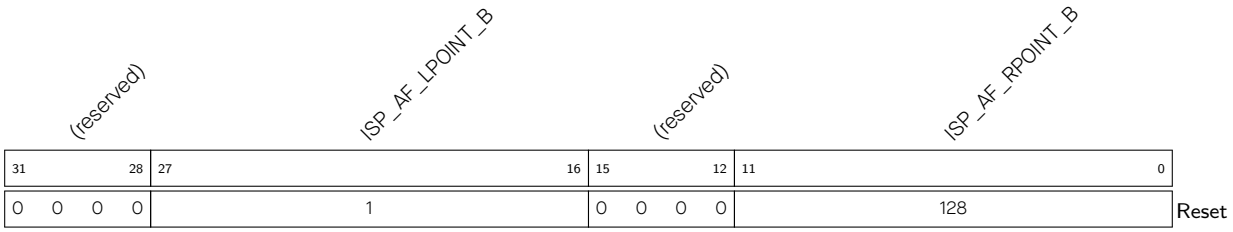
Register 38.63. ISP_AF_VSCALE_A_REG (0x0138)



ISP_AF_BPOINT_A 配置 AF 统计窗口 A 的下边界坐标，推荐小于 HNUM-1。(R/W)

ISP_AF_TPOINT_A 配置 AF 统计窗口 A 的上边界坐标，推荐大于等于 2。(R/W)

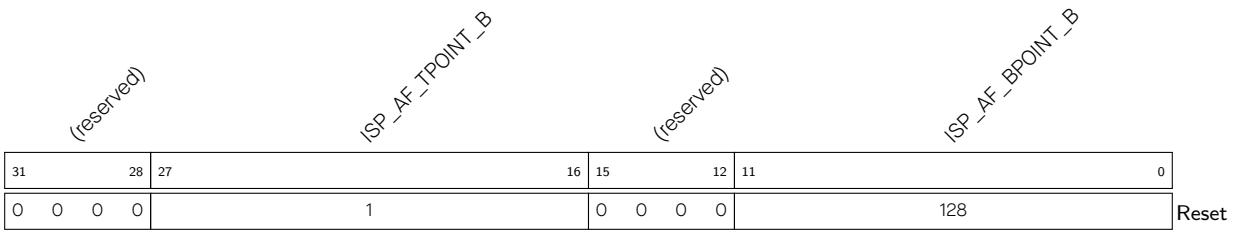
Register 38.64. ISP_AF_HSCALE_B_REG (0x013C)



ISP_AF_RPOINT_B 配置 AF 统计窗口 B 的右边界坐标，推荐小于 HNUM-1。(R/W)

ISP_AF_LPOINT_B 配置 AF 统计窗口 B 的左边界坐标，推荐大于等于 2。(R/W)

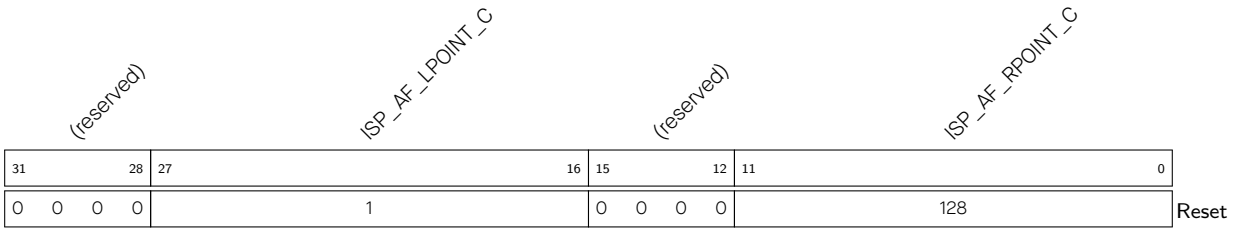
Register 38.65. ISP_AF_VSCALE_B_REG (0x0140)



ISP_AF_BPOINT_B 配置 AF 统计窗口 B 的下边界坐标，推荐小于 HNUM-1。(R/W)

ISP_AF_TPOINT_B 配置 AF 统计窗口 B 的上边界坐标，推荐大于等于 2。(R/W)

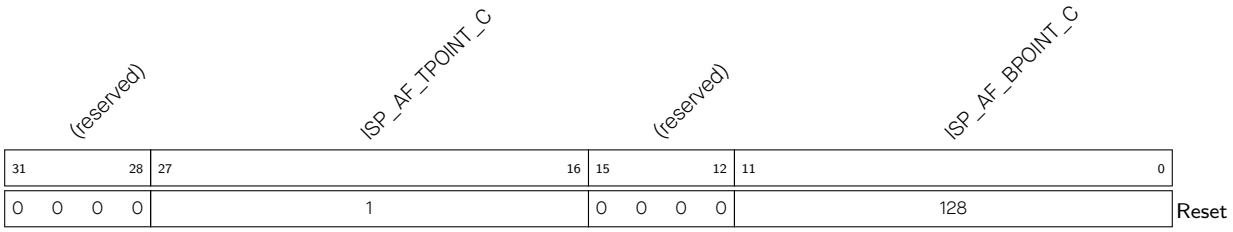
Register 38.66. ISP_AF_HSCALE_C_REG (0x0144)



ISP_AF_RPOINT_C 配置 AF 统计窗口 C 的右边界坐标，推荐小于 HNUM-1。(R/W)

ISP_AF_LPOINT_C 配置 AF 统计窗口 C 的左边界坐标，推荐大于等于 2。(R/W)

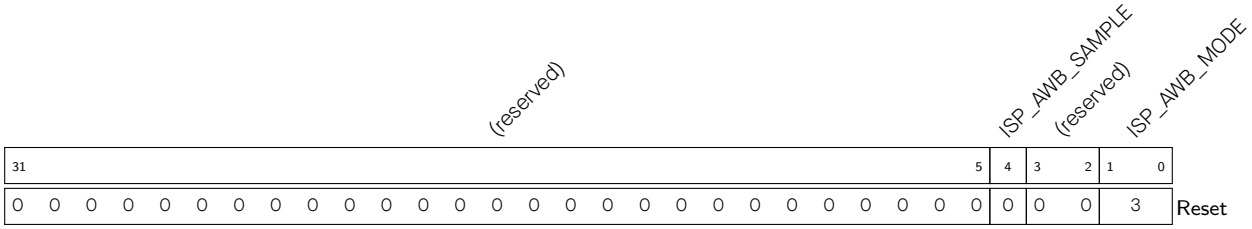
Register 38.67. ISP_AF_VSCALE_C_REG (0x0148)



ISP_AF_BPOINT_C 配置 AF 统计窗口 C 的下边界坐标，推荐小于 HNUM-1。(R/W)

ISP_AF_TPOINT_C 配置 AF 统计窗口 C 的上边界坐标，推荐大于等于 2。(R/W)

Register 38.68. ISP_AWB_MODE_REG (0x0164)



ISP_AWB_MODE 配置是否使能 AWB 算法。

1：使能 AWB 算法

其他：无效

(R/W)

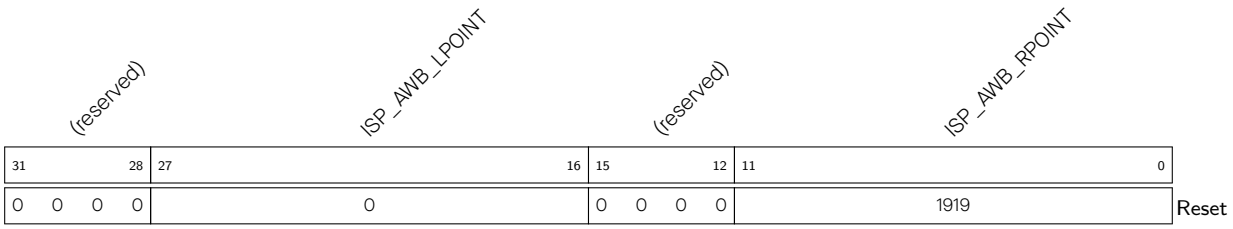
ISP_AWB_SAMPLE 配置 AWB 采样点。

0：使用 CCM 处理之前的数据

1：使用 CCM 处理之后的数据

(R/W)

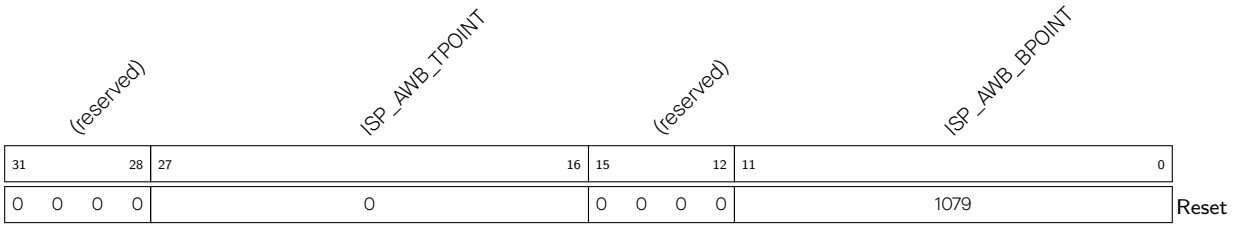
Register 38.69. ISP_AWB_HSCALE_REG (0x0168)



ISP_AWB_RPOINT 配置 AWB 统计窗口右边界坐标。(R/W)

ISP_AWB_LPOINT 配置 AWB 统计窗口左边界坐标。(R/W)

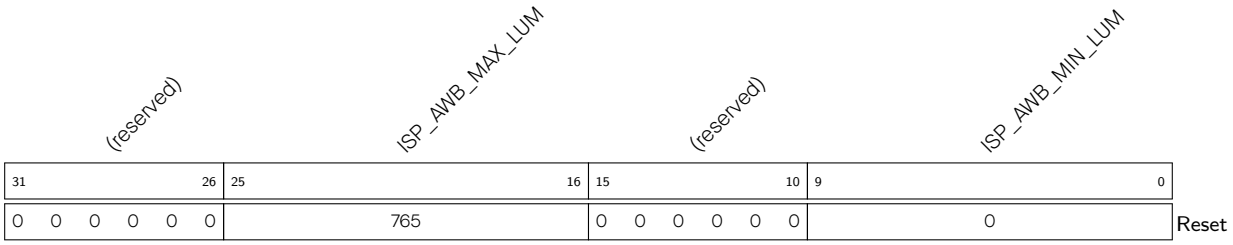
Register 38.70. ISP_AWB_VSCALE_REG (0x016C)



ISP_AWB_BPOINT 配置 AWB 统计窗口下边界坐标。(R/W)

ISP_AWB_TPOINT 配置 AWB 统计窗口上边界坐标。(R/W)

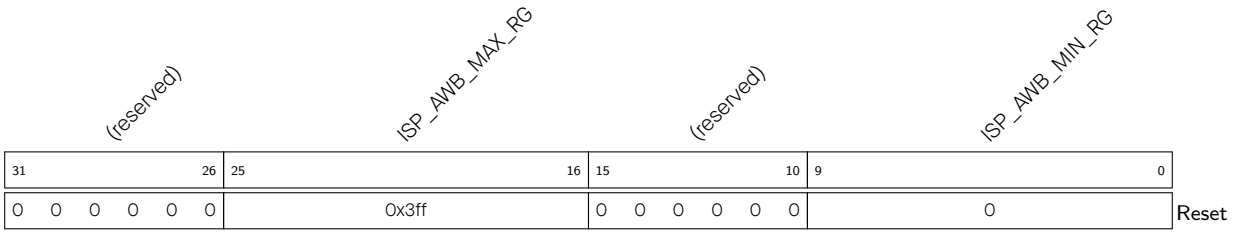
Register 38.71. ISP_AWB_TH_LUM_REG (0x0170)



ISP_AWB_MIN_LUM 配置 AWB 白点筛选 R+G+B 亮度下限。(R/W)

ISP_AWB_MAX_LUM 配置 AWB 白点筛选 R+G+B 亮度上限。(R/W)

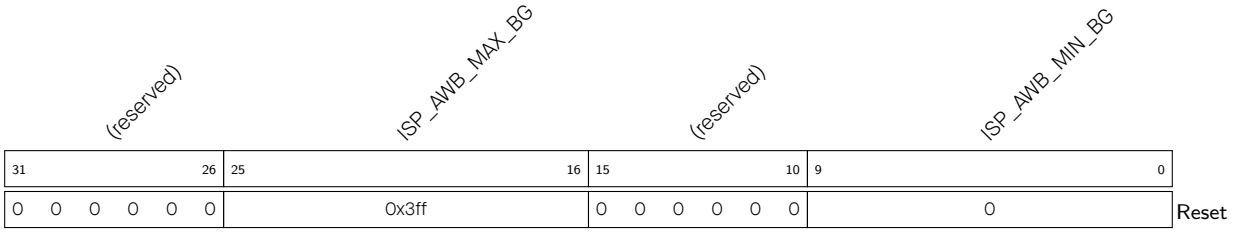
Register 38.72. ISP_AWB_TH_RG_REG (0x0174)



ISP_AWB_MIN_RG 配置 AWB 白点筛选 R/G 下限，[9:8] 为整数，[7:0] 为小数。(R/W)

ISP_AWB_MAX_RG 配置 AWB 白点筛选 R/G 上限，[9:8] 为整数，[7:0] 为小数。(R/W)

Register 38.73. ISP_AWB_TH_BG_REG (0x0178)



ISP_AWB_MIN_BG 配置 AWB 白点筛选 B/G 下限，[9:8] 为整数，[7:0] 为小数。(R/W)

ISP_AWB_MAX_BG 配置 AWB 白点筛选 B/G 上限，[9:8] 为整数，[7:0] 为小数。(R/W)

Register 38.74. ISP_COLOR_CTRL_REG (0x018C)



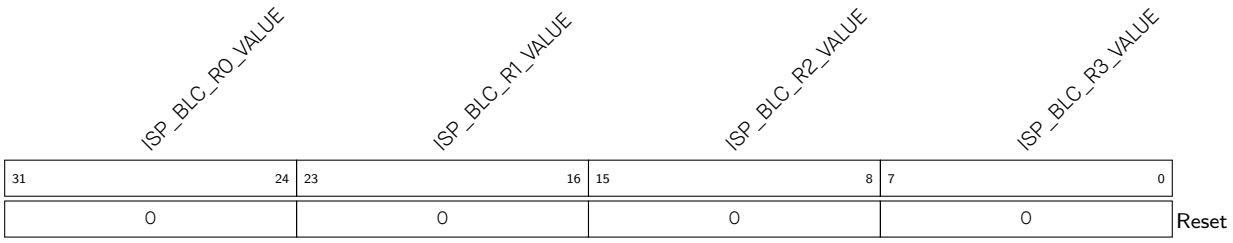
ISP_COLOR_SATURATION 配置饱和度，[7] 为整数，[6:0] 为小数。(R/W)

ISP_COLOR_HUE 配置色相的低 8 bit，总共 9 bit，最高 bit 为 ISP_COLOR_HUE_H。有效值 0-359。(R/W)

ISP_COLOR_CONTRAST 配置对比度，[7] 为整数，[6:0] 为小数。(R/W)

ISP_COLOR_BRIGHTNESS 配置亮度，二进制补码。(R/W)

Register 38.75. ISP_BLC_VALUE_REG (0x0190)



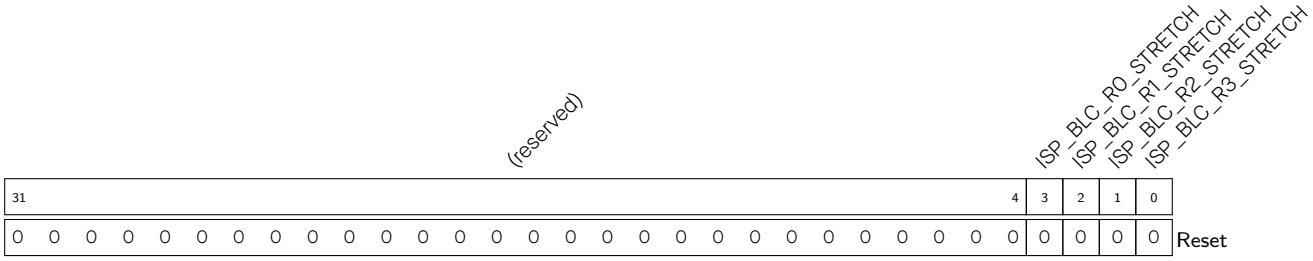
ISP_BLC_R3_VALUE 配置拜耳图像通道 3 的黑电平值。(R/W)

ISP_BLC_R2_VALUE 配置拜耳图像通道 2 的黑电平值。(R/W)

ISP_BLC_R1_VALUE 配置拜耳图像通道 1 的黑电平值。(R/W)

ISP_BLC_R0_VALUE 配置拜耳图像通道 0 的黑电平值。(R/W)

Register 38.76. ISP_BLC_CTRL0_REG (0x0194)



ISP_BLC_R3_STRETCH 配置是否使能通道 3 拉伸。

0：使能

1：关闭

(R/W)

ISP_BLC_R2_STRETCH 配置是否使能通道 2 拉伸。

0：使能

1：关闭

(R/W)

ISP_BLC_R1_STRETCH 配置是否使能通道 1 拉伸。

0：使能

1：关闭

(R/W)

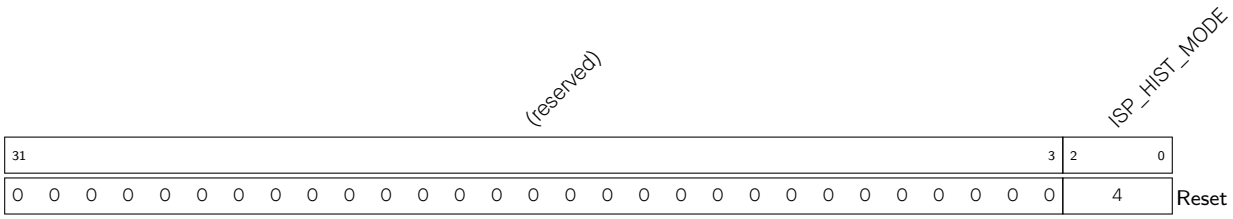
ISP_BLC_R0_STRETCH 配置是否使能通道 0 拉伸。

0：使能

1：关闭

(R/W)

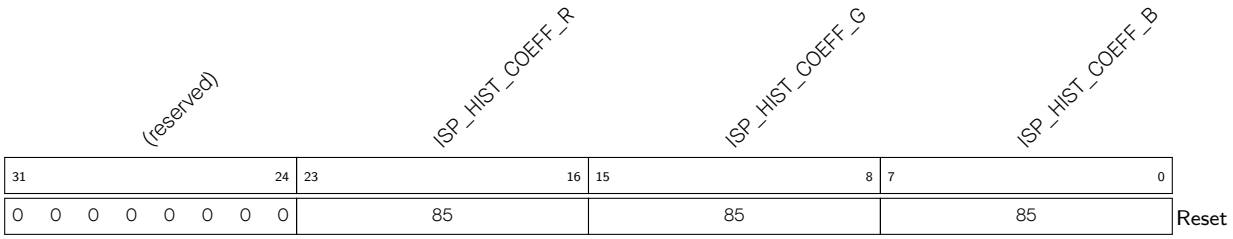
Register 38.77. ISP_HIST_MODE_REG (0x01A4)



ISP_HIST_MODE 配置直方图统计采样模式。

0：RAW_B
1：RAW_GB
2：RAW_GR
3：RAW_R
4：RGB
5：YUV_Y
6：YUV_U
7：YUV_V
(R/W)

Register 38.78. ISP_HIST_COEFF_REG (0x01A8)

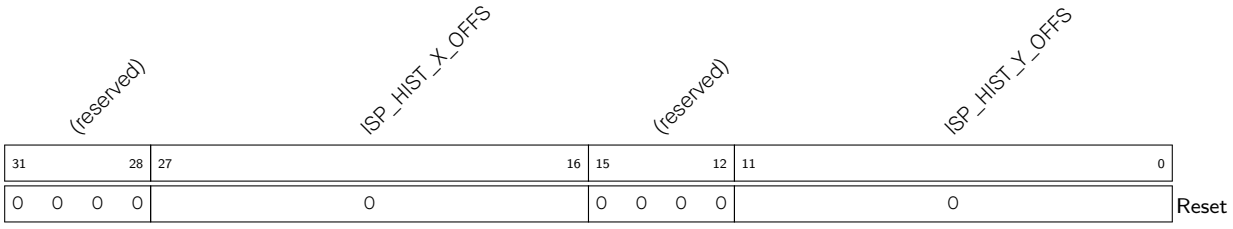


ISP_HIST_COEFF_B 配置 HIST 统计模式为 RGB 时的 RGB 转亮度的 B 权重，[7:0] 为小数。(R/W)

ISP_HIST_COEFF_G 配置 HIST 统计模式为 RGB 时的 RGB 转亮度的 G 权重，[7:0] 为小数。(R/W)

ISP_HIST_COEFF_R 配置 HIST 统计模式为 RGB 时的 RGB 转亮度的 R 权重，[7:0] 为小数。(R/W)

Register 38.79. ISP_HIST_OFFS_REG (0x01AC)



ISP_HIST_Y_OFFS 配置 HIST 统计窗口垂直方向起始坐标。(R/W)

ISP_HIST_X_OFFS 配置 HIST 统计窗口水平方向起始坐标。(R/W)

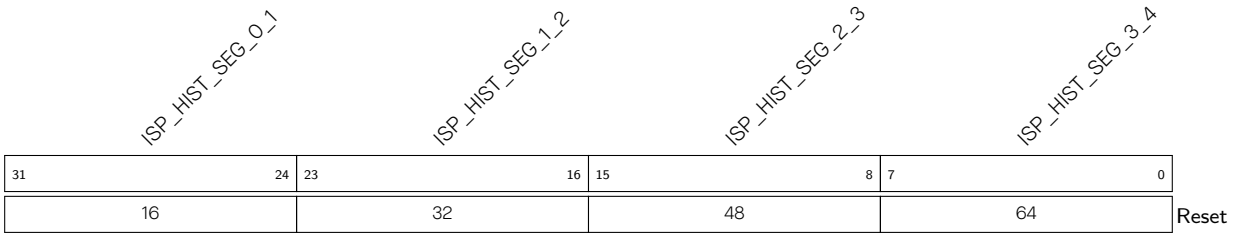
Register 38.80. ISP_HIST_SIZE_REG (0x01B0)



ISP_HIST_Y_SIZE 配置 HIST 子窗口垂直方向尺寸。(R/W)

ISP_HIST_X_SIZE 配置 HIST 子窗口水平方向尺寸。(R/W)

Register 38.81. ISP_HIST_SEG0_REG (0x01B4)



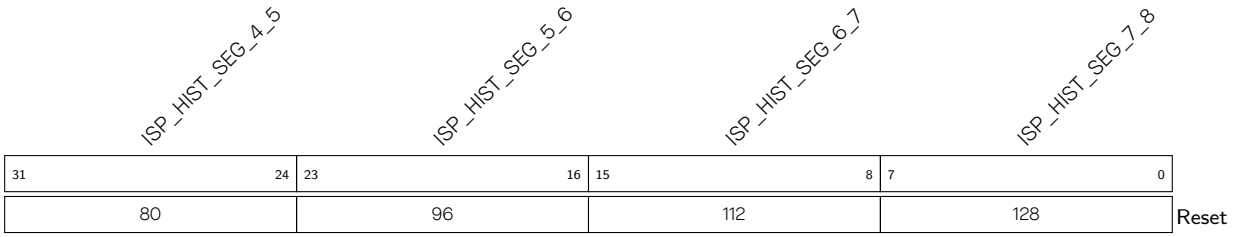
ISP_HIST_SEG_3_4 配置 HIST 直方图区间 3 和 4 分隔阈值。(R/W)

ISP_HIST_SEG_2_3 配置 HIST 直方图区间 2 和 3 分隔阈值。(R/W)

ISP_HIST_SEG_1_2 配置 HIST 直方图区间 1 和 2 分隔阈值。(R/W)

ISP_HIST_SEG_0_1 配置 HIST 直方图区间 0 和 1 分隔阈值。(R/W)

Register 38.82. ISP_HIST_SEG1_REG (0x01B8)



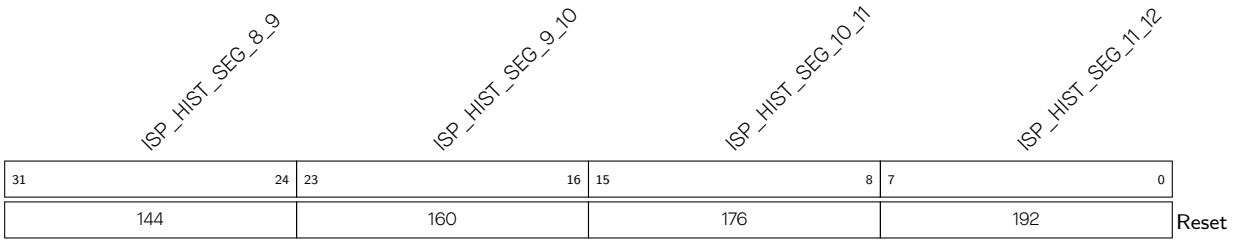
ISP_HIST_SEG_7_8 配置 HIST 直方图区间 7 和 8 分隔阈值。(R/W)

ISP_HIST_SEG_6_7 配置 HIST 直方图区间 6 和 7 分隔阈值。(R/W)

ISP_HIST_SEG_5_6 配置 HIST 直方图区间 5 和 6 分隔阈值。(R/W)

ISP_HIST_SEG_4_5 配置 HIST 直方图区间 4 和 5 分隔阈值。(R/W)

Register 38.83. ISP_HIST_SEG2_REG (0x01BC)



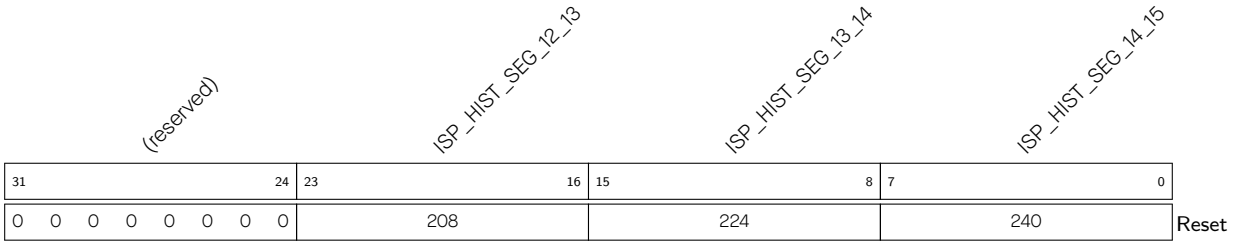
ISP_HIST_SEG_11_12 配置 HIST 直方图区间 11 和 12 分隔阈值。(R/W)

ISP_HIST_SEG_10_11 配置 HIST 直方图区间 10 和 11 分隔阈值。(R/W)

ISP_HIST_SEG_9_10 配置 HIST 直方图区间 9 和 10 分隔阈值。(R/W)

ISP_HIST_SEG_8_9 配置 HIST 直方图区间 8 和 9 分隔阈值。(R/W)

Register 38.84. ISP_HIST_SEG3_REG (0x01C0)

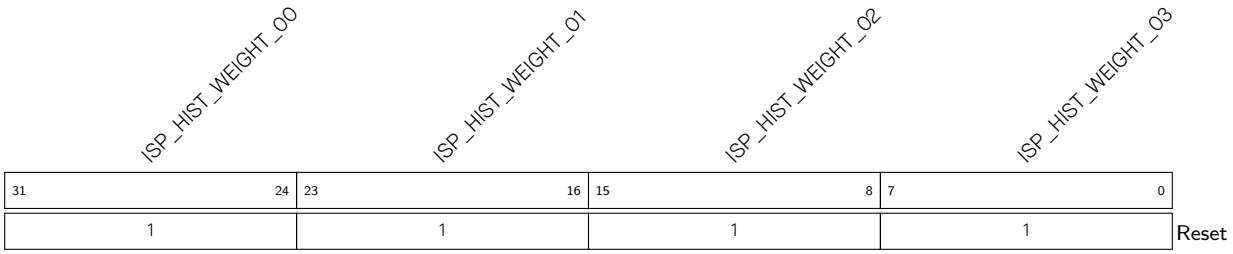


ISP_HIST_SEG_14_15 配置 HIST 直方图区间 14 和 15 分隔阈值。(R/W)

ISP_HIST_SEG_13_14 配置 HIST 直方图区间 13 和 14 分隔阈值。(R/W)

ISP_HIST_SEG_12_13 配置 HIST 直方图区间 12 和 13 分隔阈值。(R/W)

Register 38.85. ISP_HIST_WEIGHT0_REG (0x01C4)



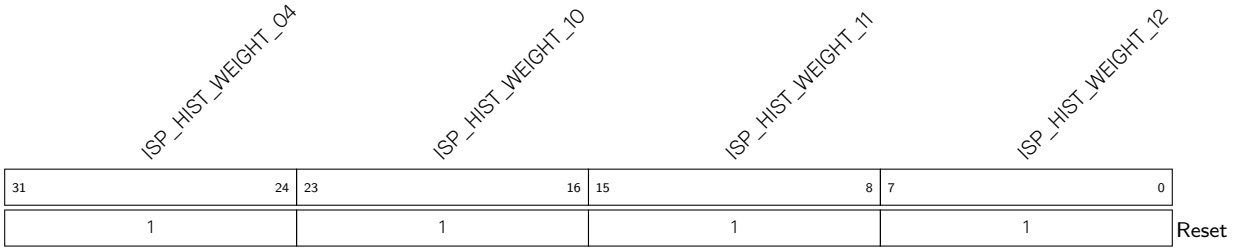
ISP_HIST_WEIGHT_03 配置 HIST 子窗口 03 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_02 配置 HIST 子窗口 02 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_01 配置 HIST 子窗口 01 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_00 配置 HIST 子窗口 00 权重，[7:0] 为小数。(R/W)

Register 38.86. ISP_HIST_WEIGHT1_REG (0x01C8)



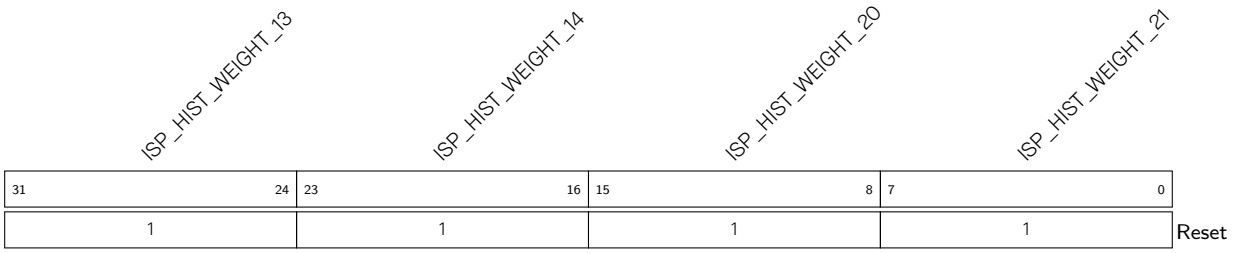
ISP_HIST_WEIGHT_12 配置 HIST 子窗口 12 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_11 配置 HIST 子窗口 11 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_10 配置 HIST 子窗口 10 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_04 配置 HIST 子窗口 04 权重，[7:0] 为小数。(R/W)

Register 38.87. ISP_HIST_WEIGHT2_REG (0x01CC)



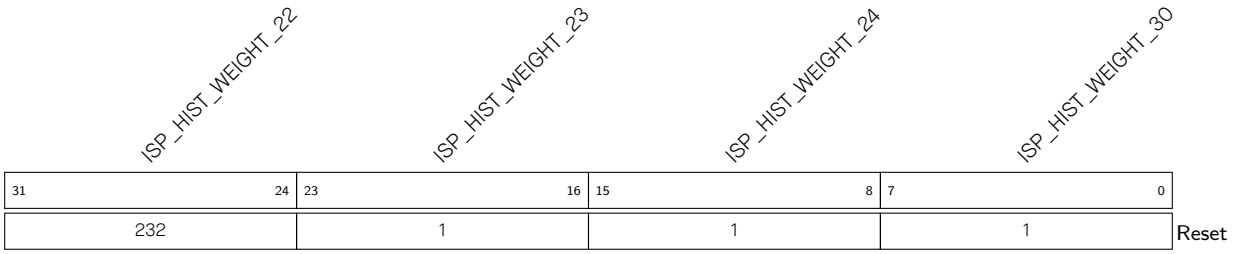
ISP_HIST_WEIGHT_21 配置 HIST 子窗口 21 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_20 配置 HIST 子窗口 20 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_14 配置 HIST 子窗口 14 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_13 配置 HIST 子窗口 13 权重，[7:0] 为小数。(R/W)

Register 38.88. ISP_HIST_WEIGHT3_REG (0x01D0)



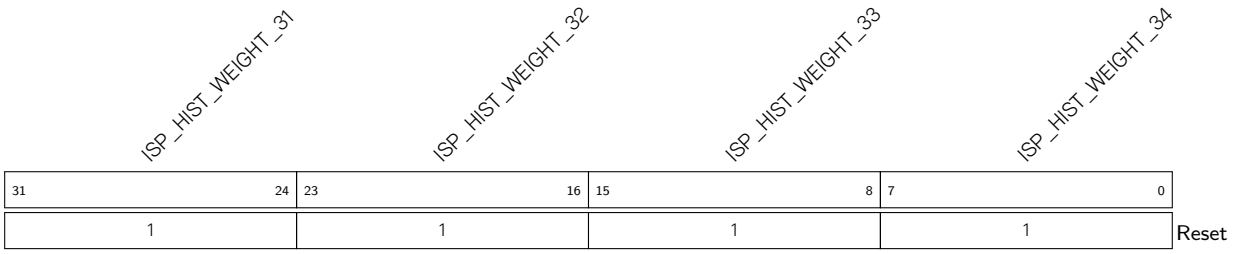
ISP_HIST_WEIGHT_30 配置 HIST 子窗口 30 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_24 配置 HIST 子窗口 24 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_23 配置 HIST 子窗口 23 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_22 配置 HIST 子窗口 22 权重，[7:0] 为小数。(R/W)

Register 38.89. ISP_HIST_WEIGHT4_REG (0x01D4)



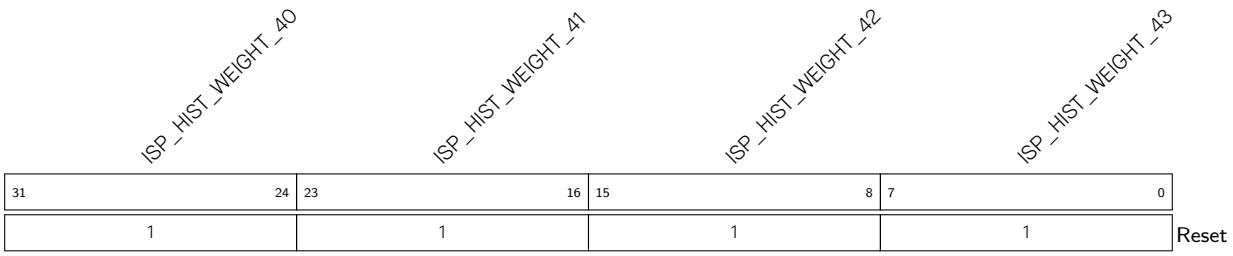
ISP_HIST_WEIGHT_34 配置 HIST 子窗口 34 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_33 配置 HIST 子窗口 33 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_32 配置 HIST 子窗口 32 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_31 配置 HIST 子窗口 31 权重，[7:0] 为小数。(R/W)

Register 38.90. ISP_HIST_WEIGHT5_REG (0x01D8)



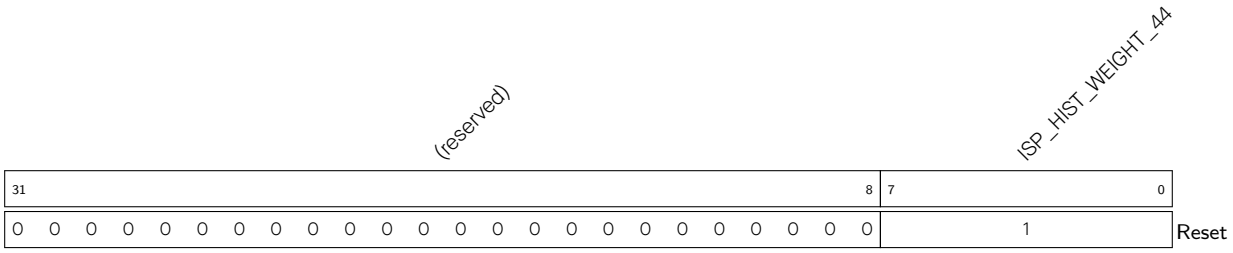
ISP_HIST_WEIGHT_43 配置 HIST 子窗口 43 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_42 配置 HIST 子窗口 42 权重，[7:0] 为小数。(R/W)

ISP_HIST_WEIGHT_41 配置 HIST 子窗口 41 权重，[7:0] 为小数。(R/W)

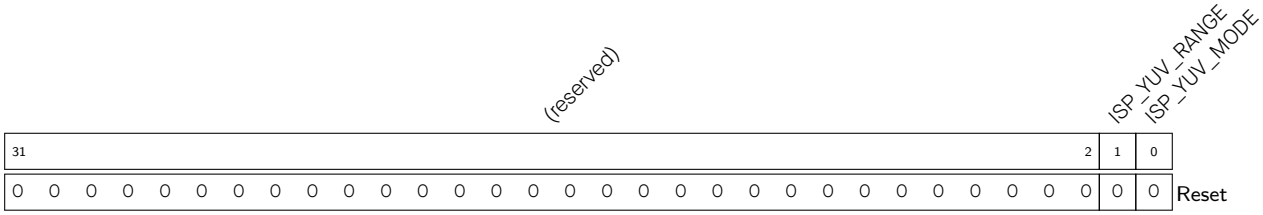
ISP_HIST_WEIGHT_40 配置 HIST 子窗口 40 权重，[7:0] 为小数。(R/W)

Register 38.91. ISP_HIST_WEIGHT6_REG (0x01DC)



ISP_HIST_WEIGHT_44 配置 HIST 子窗口 44 权重，[7:0] 为小数。(R/W)

Register 38.92. ISP_YUV_FORMAT_REG (0x0234)



ISP_YUV_MODE 配置 YUV 输出模式。

0：ITU-R BT.601

1：ITU-R BT.709

(R/W)

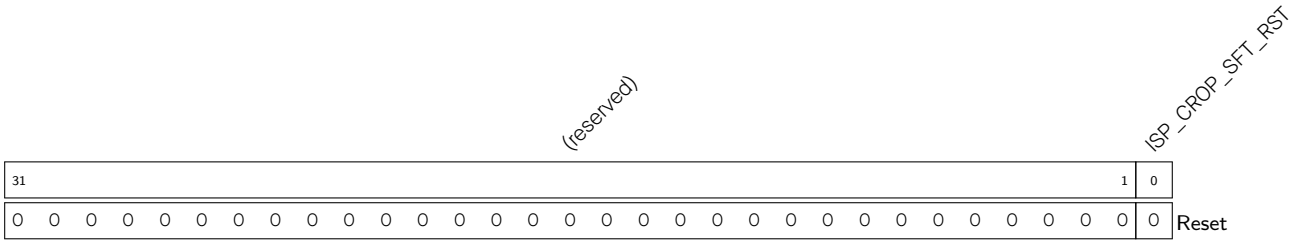
ISP_YUV_RANGE 配置 YUV 输出范围。

0：full range (YUV2RGB)

1：limit range (YUV_Limit)

(R/W)

Register 38.93. ISP_CROP_CTRL_REG (0x0244)



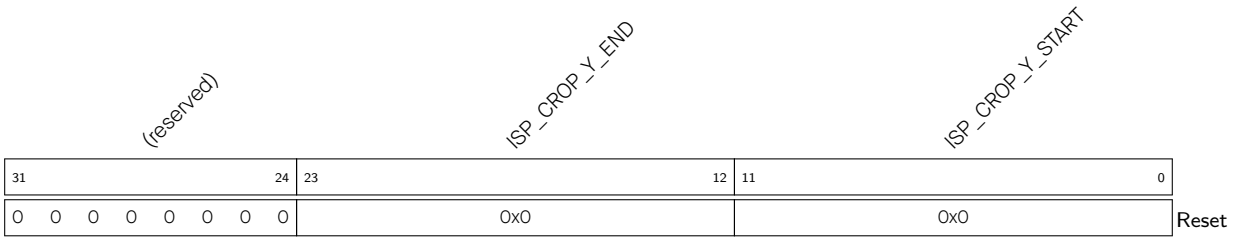
ISP_CROP_SFT_RST 配置是否复位 CROP 错误状态。

0：不复位

1：复位

(WT)

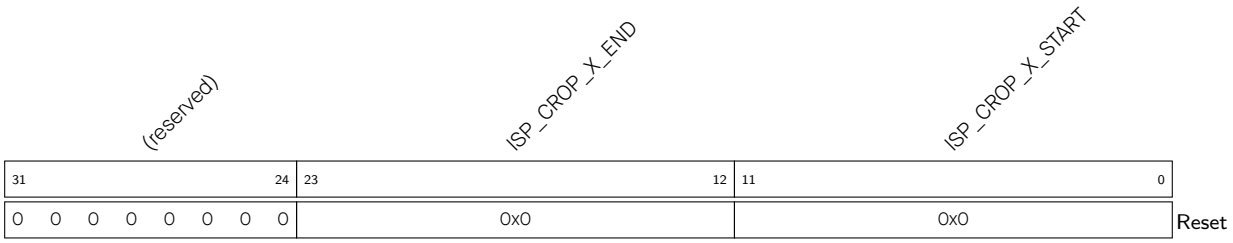
Register 38.94. ISP_CROP_Y_CAPTURE_REG (0x0248)



ISP_CROP_Y_START 配置 Y 方向起始坐标。(R/W)

ISP_CROP_Y_END 配置 Y 方向结束坐标。(R/W)

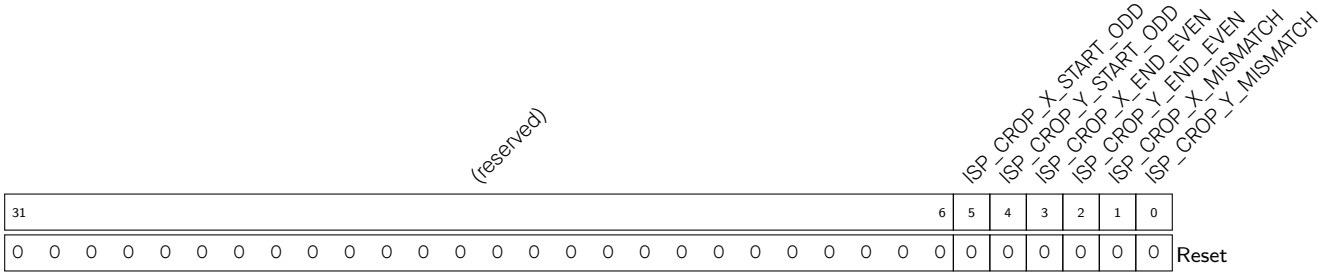
Register 38.95. ISP_CROP_X_CAPTURE_REG (0x024C)



ISP_CROP_X_START 配置 X 方向起始坐标。(R/W)

ISP_CROP_X_END 配置 X 方向结束坐标。(R/W)

Register 38.96. ISP_CROP_ERR_ST_REG (0x0250)



ISP_CROP_Y_MISMATCH 表示 CROP Y 方向配置坐标超过图像边界错误。(RO)

ISP_CROP_X_MISMATCH 表示 CROP X 方向配置坐标超过图像边界错误。(RO)

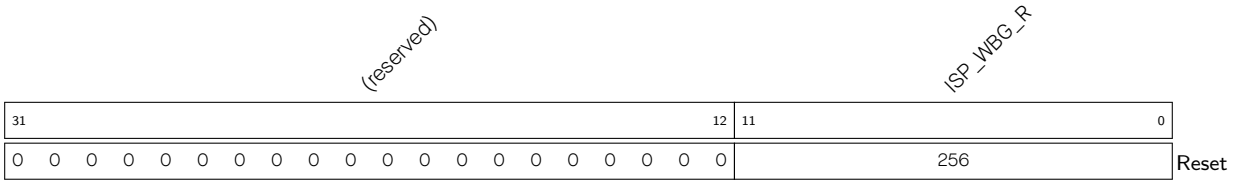
ISP_CROP_Y_END_EVEN 表示 Y 方向结束坐标为偶数错误。(RO)

ISP_CROP_X_END_EVEN 表示 X 方向结束坐标为偶数错误。(RO)

ISP_CROP_Y_START_ODD 表示 Y 方向起始坐标为奇数错误。(RO)

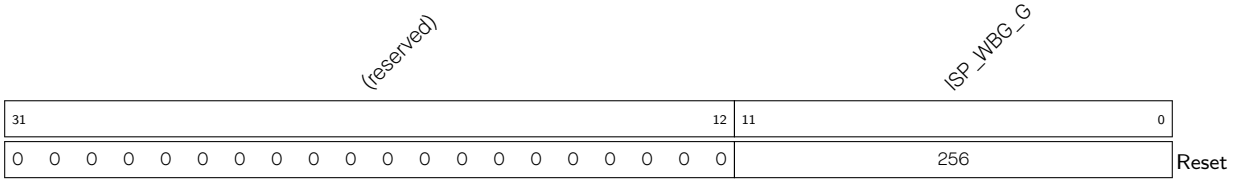
ISP_CROP_X_START_ODD 表示 X 方向起始坐标为奇数错误。(RO)

Register 38.97. ISP_WBG_COEF_R_REG (0x0254)



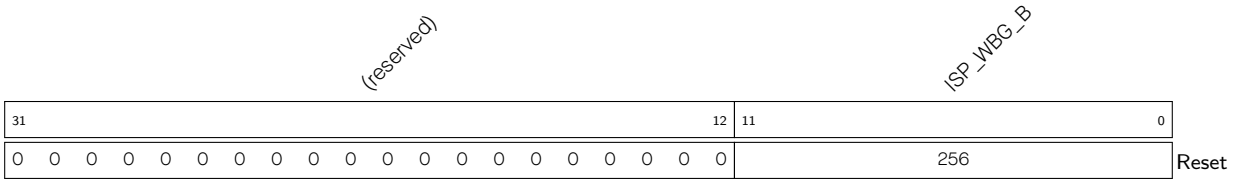
ISP_WBG_R 配置 WBG R 通道增益，4-bit 整数，8-bit 小数。(R/W)

Register 38.98. ISP_WBG_COEF_G_REG (0x0258)



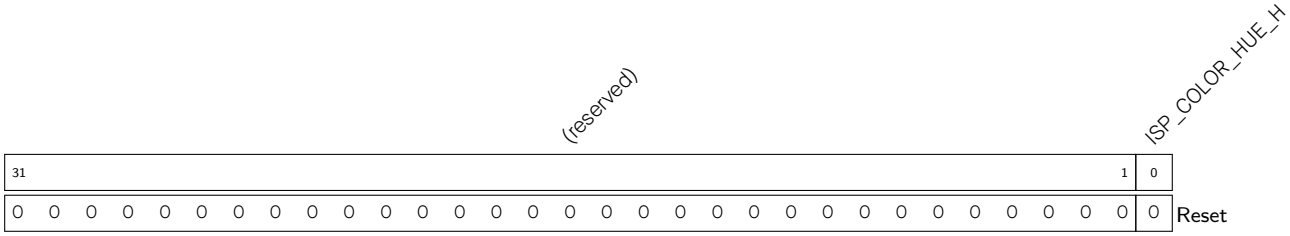
ISP_WBG_G 配置 WBG G 通道增益，4-bit 整数，8-bit 小数。(R/W)

Register 38.99. ISP_WBG_COEF_B_REG (0x025C)



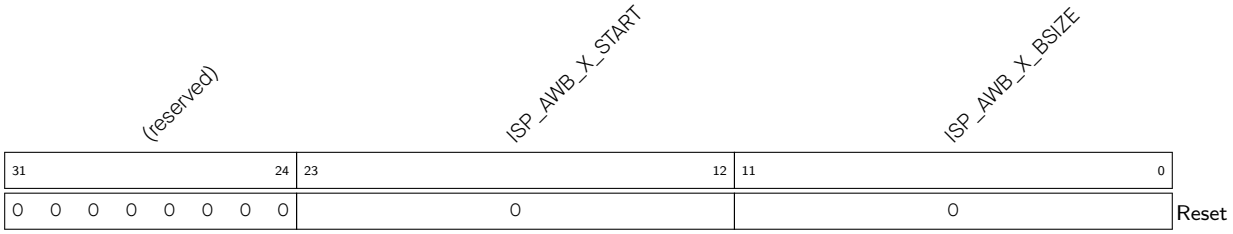
ISP_WBG_B 配置 WBG B 通道增益，4-bit 整数，8-bit 小数。(R/W)

Register 38.100. ISP_COLOR_HUE_CTRL_REG (0x0260)



ISP_COLOR_HUE_H 配置 HUE 最高比特。(R/W)

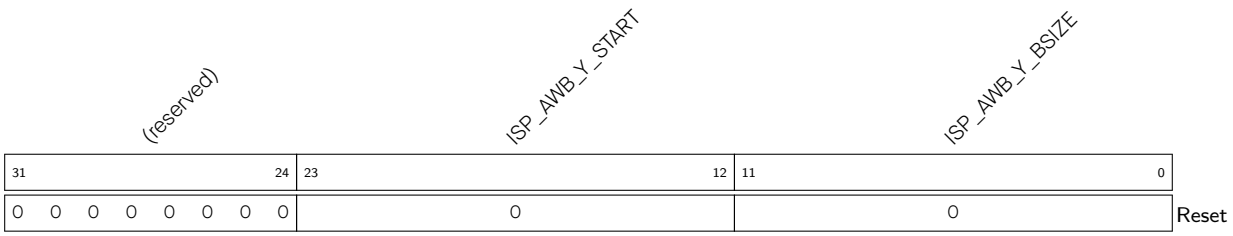
Register 38.101. ISP_AWB_BX_REG (0x0264)



ISP_AWB_X_BSIZE 配置 AWB 子窗 X 方向尺寸，最小为 4。(R/W)

ISP_AWB_X_START 配置 AWB 子窗 X 方向起始坐标。(R/W)

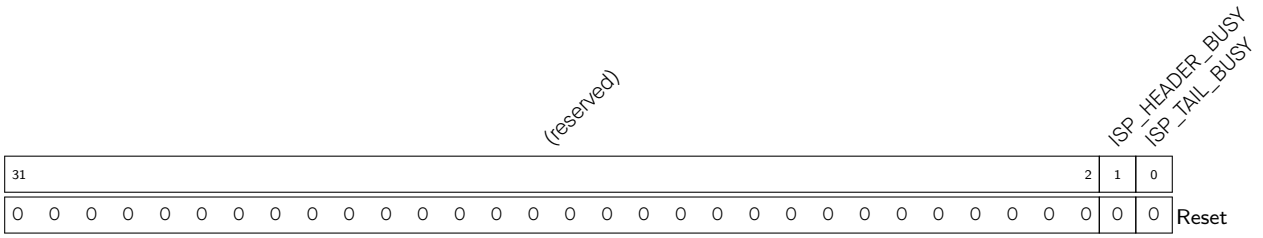
Register 38.102. ISP_AWB_BY_REG (0x0268)



ISP_AWB_Y_BSIZE 配置 AWB 子窗 Y 方向尺寸。(R/W)

ISP_AWB_Y_START 配置 AWB 子窗 Y 方向起始坐标。(R/W)

Register 38.103. ISP_STATE_REG (0x026C)



ISP_TAIL_BUSY 表示 isp_tail 是否忙状态。

0：空闲

1：忙碌

(RO)

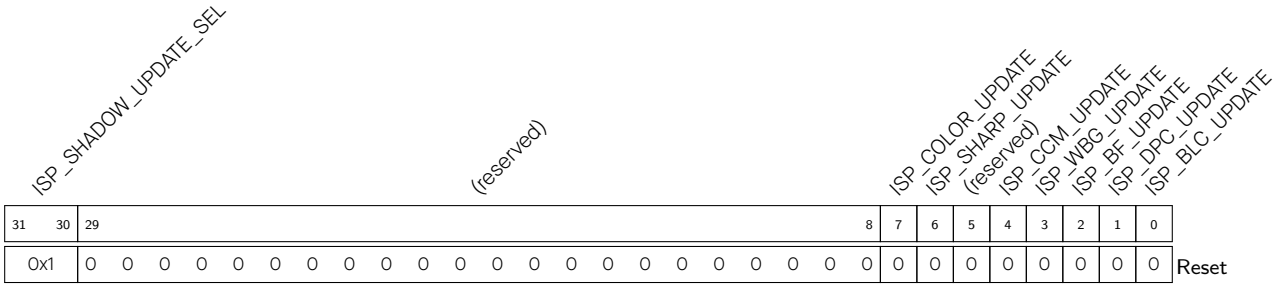
ISP_HEADER_BUSY 表示 isp_header 是否忙状态。

0：空闲

1：忙碌

(RO)

Register 38.104. ISP_SHADOW_REG_CTRL_REG (0x0270)



ISP_BLC_UPDATE 写 1 更新 BLC 寄存器。(R/W)

ISP_DPC_UPDATE 写 1 更新 DPC 寄存器。(R/W)

ISP_BF_UPDATE 写 1 更新 BF 寄存器。(R/W)

ISP_WBG_UPDATE 写 1 更新 WBG 寄存器。(R/W)

ISP_CCM_UPDATE 写 1 更新 CCM 寄存器。(R/W)

ISP_SHARP_UPDATE 写 1 更新 SHARP 寄存器。(R/W)

ISP_COLOR_UPDATE 写 1 更新 COLOR 寄存器。(R/W)

ISP_SHADOW_UPDATE_SEL 配置影子寄存器更新方式。

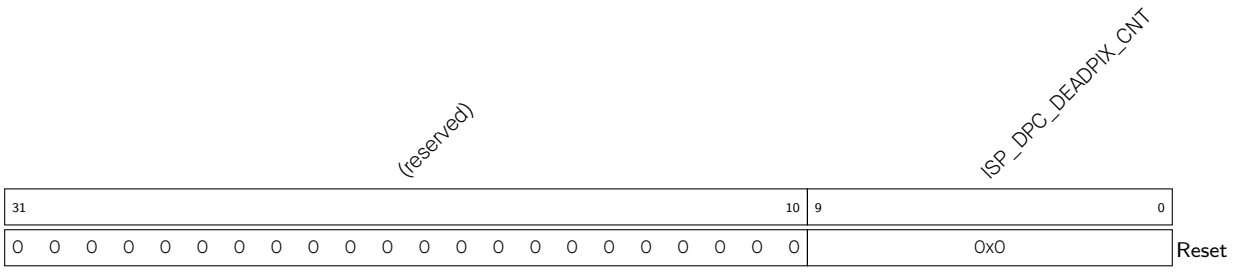
0：禁用影子寄存器

1：每一次 vsync 时自动更新

2：向对应更新寄存器写 1 后，下一次 vsync 自动更新

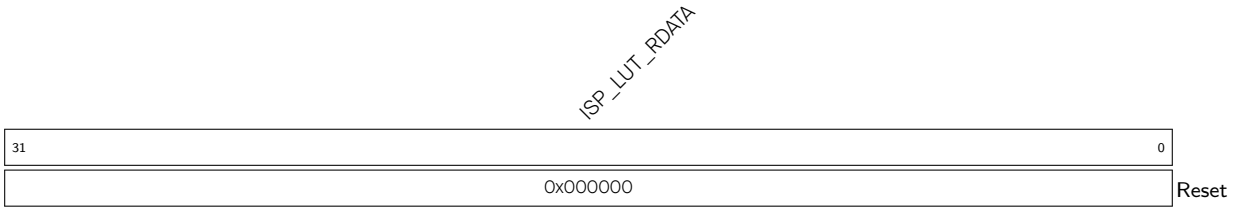
(R/W)

Register 38.105. ISP_DPC_DEADPIX_CNT_REG (0x0044)



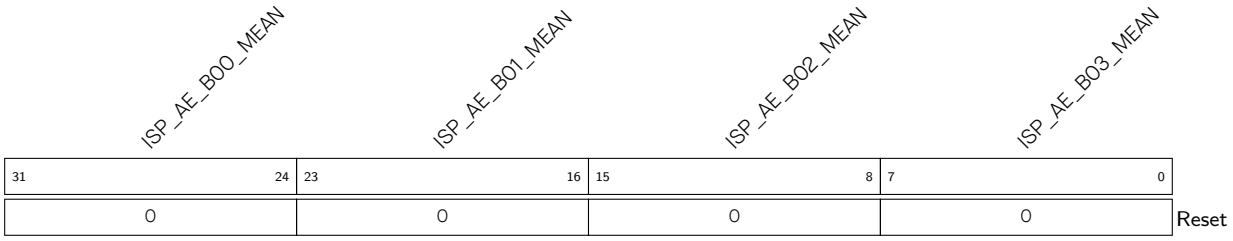
ISP_DPC_DEADPIX_CNT 表示静态标定统计出的坏点数量。(RO)

Register 38.106. ISP_LUT_RDATA_REG (0x0050)



ISP_LUT_RDATA 从 LUT 读出的数据。配置 ISP_LUT_CMD 后读该寄存器。(RO)

Register 38.107. ISP_AE_BLOCK_MEAN_0_REG (0x00D8)



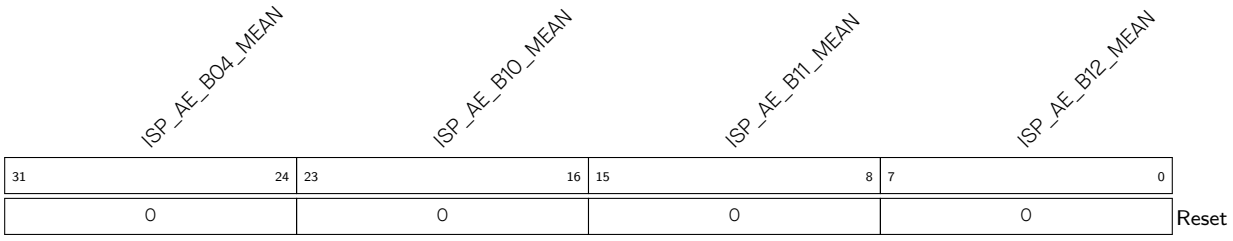
ISP_AE_B03_MEAN 表示 AE 子窗口 03 统计结果。(RO)

ISP_AE_B02_MEAN 表示 AE 子窗口 02 统计结果。(RO)

ISP_AE_B01_MEAN 表示 AE 子窗口 01 统计结果。(RO)

ISP_AE_B00_MEAN 表示 AE 子窗口 00 统计结果。(RO)

Register 38.108. ISP_AE_BLOCK_MEAN_1_REG (0x00DC)



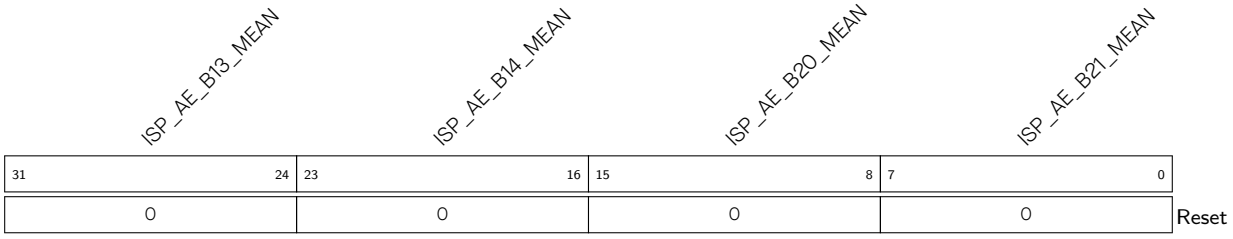
ISP_AE_B12_MEAN 表示 AE 子窗口 12 统计结果。(RO)

ISP_AE_B11_MEAN 表示 AE 子窗口 11 统计结果。(RO)

ISP_AE_B10_MEAN 表示 AE 子窗口 10 统计结果。(RO)

ISP_AE_B04_MEAN 表示 AE 子窗口 04 统计结果。(RO)

Register 38.109. ISP_AE_BLOCK_MEAN_2_REG (0x00E0)



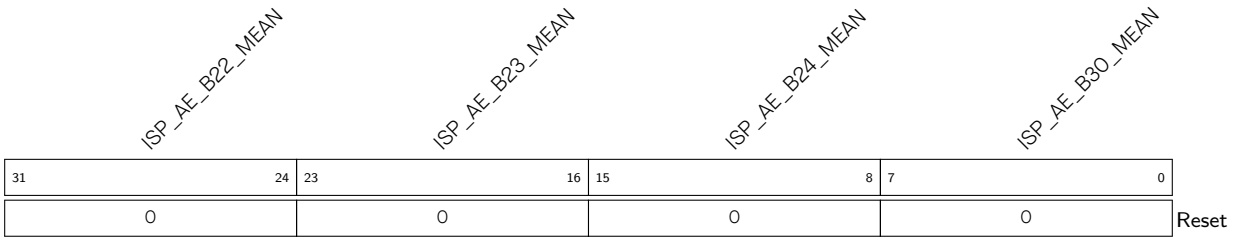
ISP_AE_B21_MEAN 表示 AE 子窗口 21 统计结果。(RO)

ISP_AE_B20_MEAN 表示 AE 子窗口 20 统计结果。(RO)

ISP_AE_B14_MEAN 表示 AE 子窗口 14 统计结果。(RO)

ISP_AE_B13_MEAN 表示 AE 子窗口 13 统计结果。(RO)

Register 38.110. ISP_AE_BLOCK_MEAN_3_REG (0x00E4)



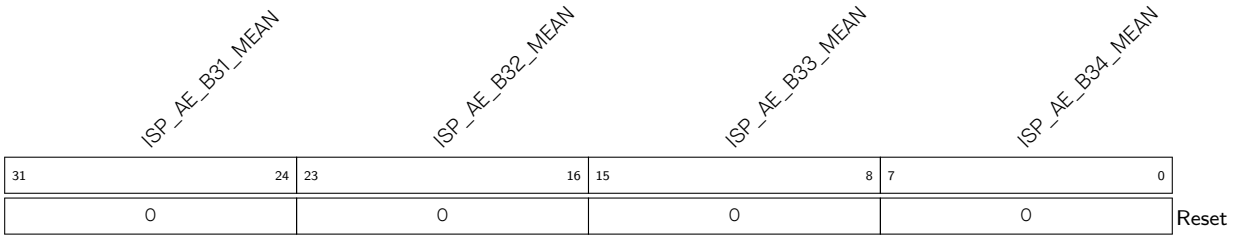
ISP_AE_B30_MEAN 表示 AE 子窗口 30 统计结果。(RO)

ISP_AE_B24_MEAN 表示 AE 子窗口 24 统计结果。(RO)

ISP_AE_B23_MEAN 表示 AE 子窗口 23 统计结果。(RO)

ISP_AE_B22_MEAN 表示 AE 子窗口 22 统计结果。(RO)

Register 38.111. ISP_AE_BLOCK_MEAN_4_REG (0x00E8)



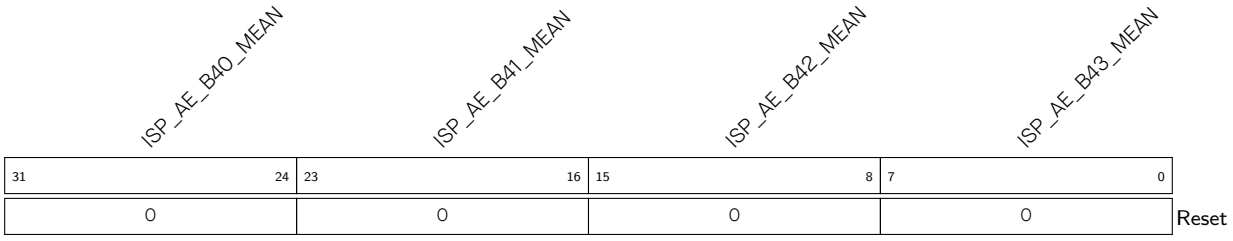
ISP_AE_B34_MEAN 表示 AE 子窗口 34 统计结果。(RO)

ISP_AE_B33_MEAN 表示 AE 子窗口 33 统计结果。(RO)

ISP_AE_B32_MEAN 表示 AE 子窗口 32 统计结果。(RO)

ISP_AE_B31_MEAN 表示 AE 子窗口 31 统计结果。(RO)

Register 38.112. ISP_AE_BLOCK_MEAN_5_REG (0x00EC)



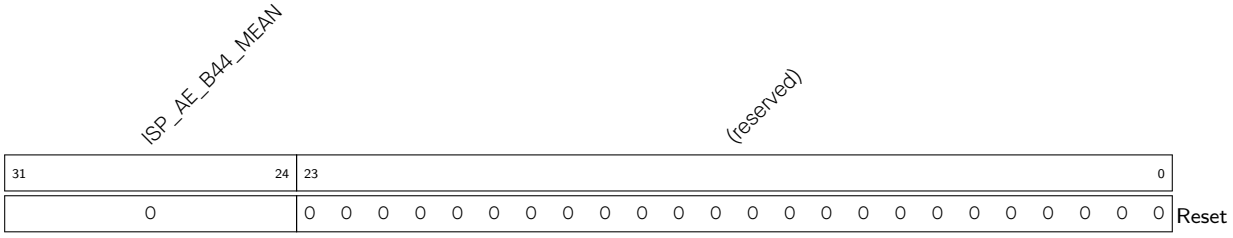
ISP_AE_B43_MEAN 表示 AE 子窗口 43 统计结果。(RO)

ISP_AE_B42_MEAN 表示 AE 子窗口 42 统计结果。(RO)

ISP_AE_B41_MEAN 表示 AE 子窗口 41 统计结果。(RO)

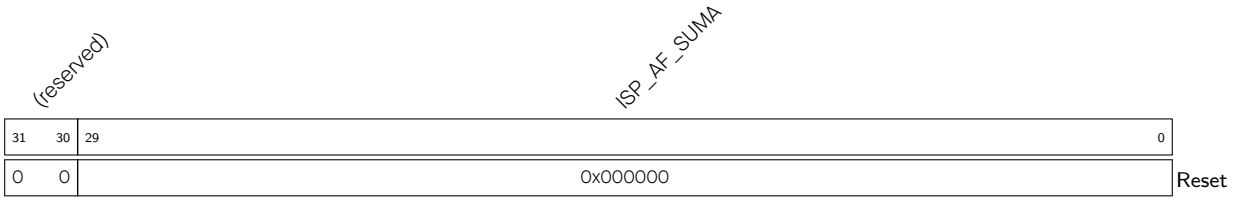
ISP_AE_B40_MEAN 表示 AE 子窗口 40 统计结果。(RO)

Register 38.113. ISP_AE_BLOCK_MEAN_6_REG (0x00F0)



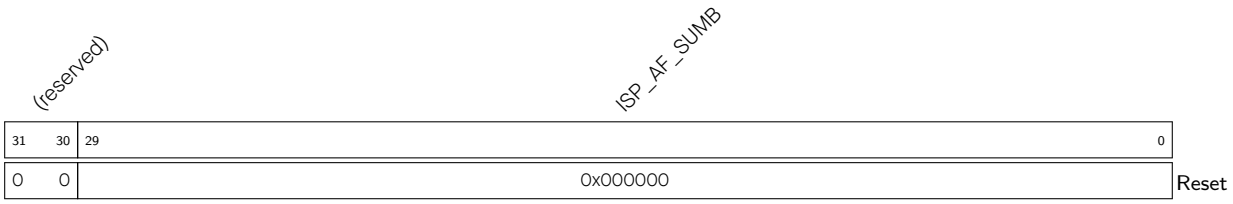
ISP_AE_B44_MEAN 表示 AE 子窗口 44 统计结果。(RO)

Register 38.114. ISP_AF_SUM_A_REG (0x014C)



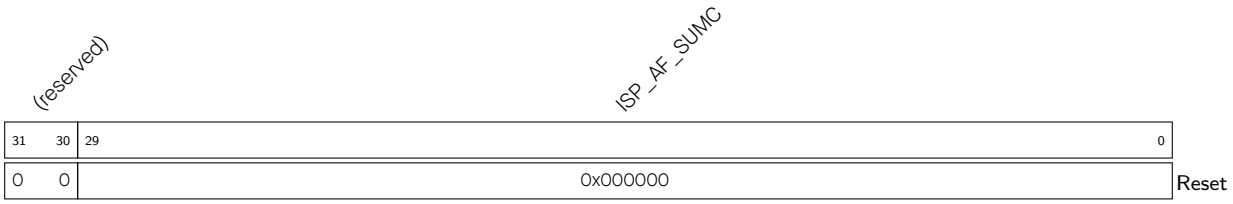
ISP_AF_SUMA 表示 AF 窗口 A 清晰度统计结果。(RO)

Register 38.115. ISP_AF_SUM_B_REG (0x0150)



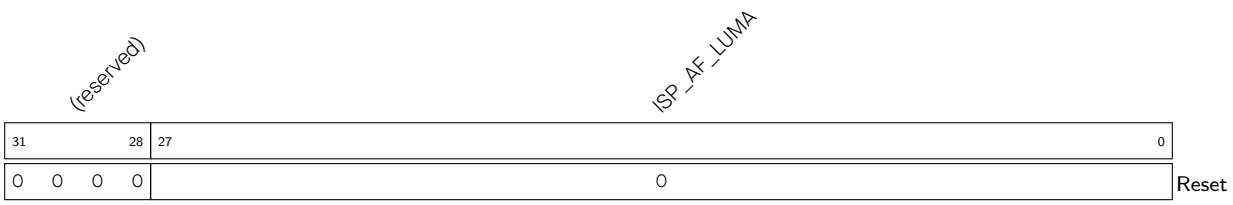
ISP_AF_SUMB 表示 AF 窗口 B 清晰度统计结果。(RO)

Register 38.116. ISP_AF_SUM_C_REG (0x0154)



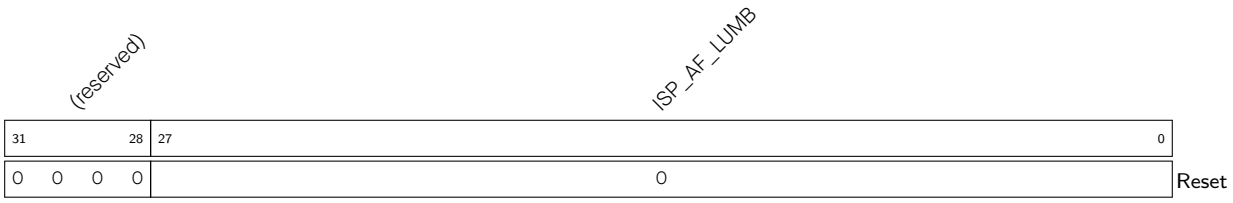
ISP_AF_SUMC 表示 AF 窗口 C 清晰度统计结果。(RO)

Register 38.117. ISP_AF_LUM_A_REG (0x0158)



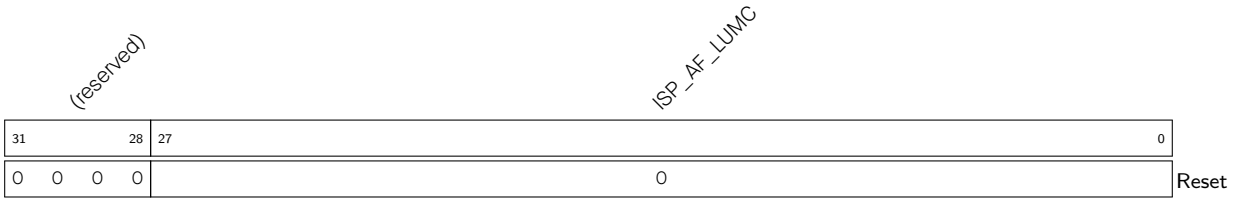
ISP_AF_LUMA 表示 AF 窗口 A 亮度统计结果。(RO)

Register 38.118. ISP_AF_LUM_B_REG (0x015C)



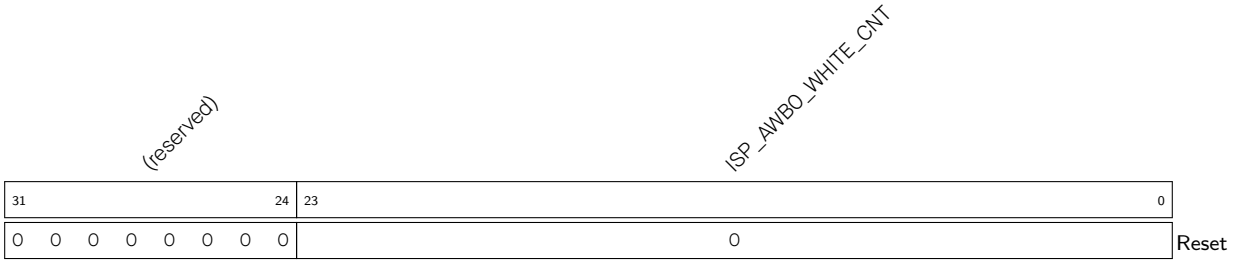
ISP_AF_LUMB 表示 AF 窗口 B 亮度统计结果。(RO)

Register 38.119. ISP_AF_LUM_C_REG (0x0160)



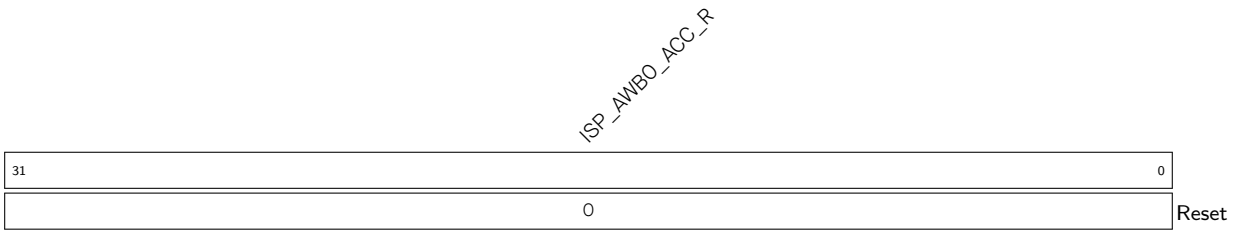
ISP_AF_LUMC 表示 AF 窗口 C 亮度统计结果。(RO)

Register 38.120. ISP_AWB0_WHITE_CNT_REG (0x017C)



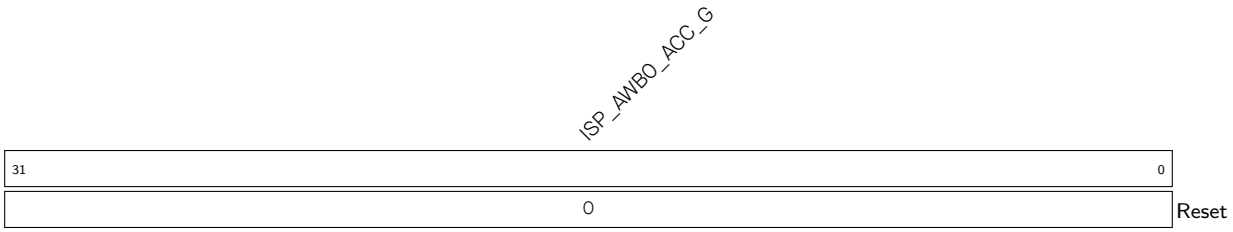
ISP_AWB0_WHITE_CNT 表示 AWB 统计窗口内白点数量。(RO)

Register 38.121. ISP_AWB0_ACC_R_REG (0x0180)



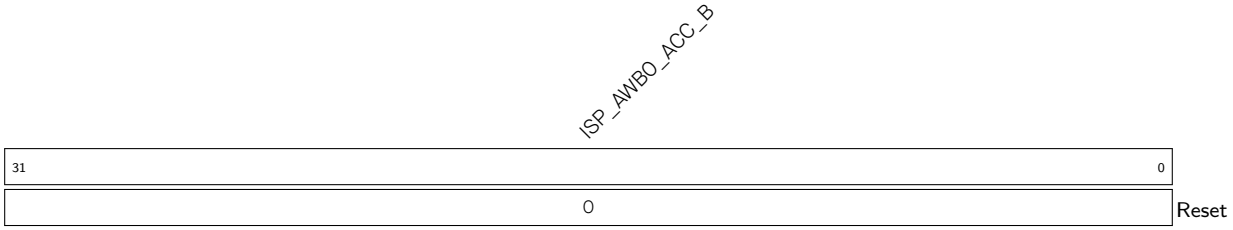
ISP_AWB0_ACC_R 表示 AWB 统计窗口内白点 R 通道累加值。(RO)

Register 38.122. ISP_AWB0_ACC_G_REG (0x0184)



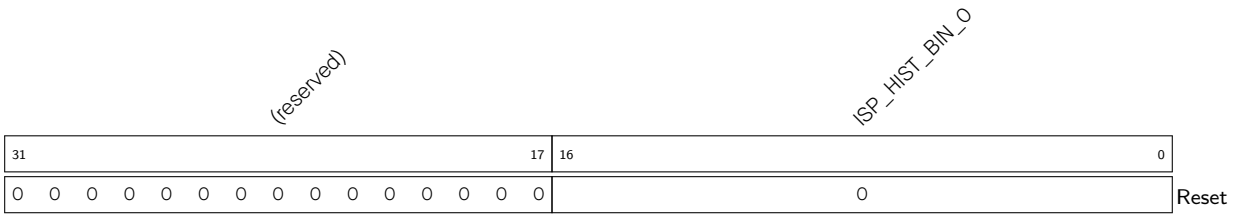
ISP_AWB0_ACC_G 表示 AWB 统计窗口内白点 G 通道累加值。(RO)

Register 38.123. ISP_AWB0_ACC_B_REG (0x0188)



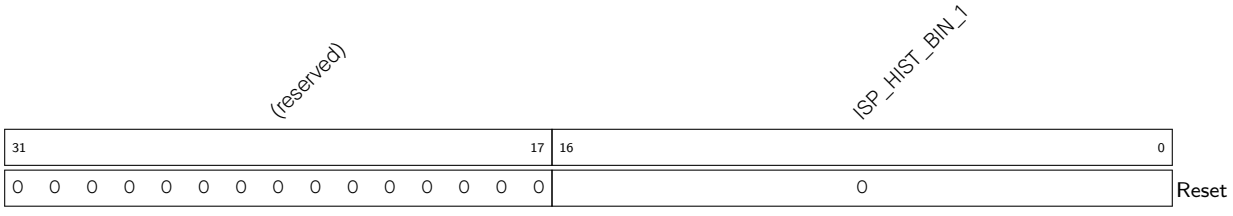
ISP_AWB0_ACC_B 表示 AWB 统计窗口内白点 B 通道累加值。(RO)

Register 38.124. ISP_HIST_BIN0_REG (0x01E0)



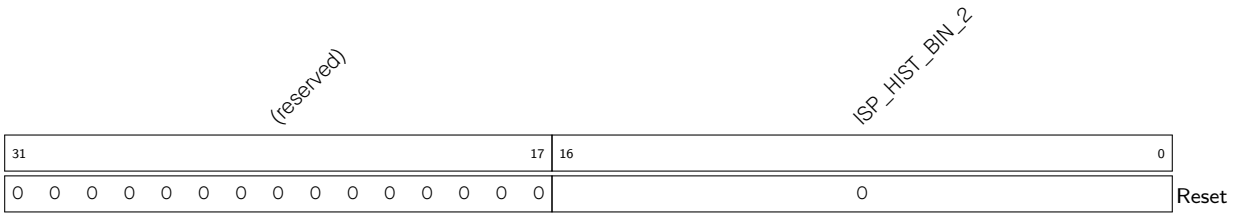
ISP_HIST_BIN_0 表示 HIST 统计窗口内亮度落在区间 0 的像素数量。(RO)

Register 38.125. ISP_HIST_BIN1_REG (0x01E4)



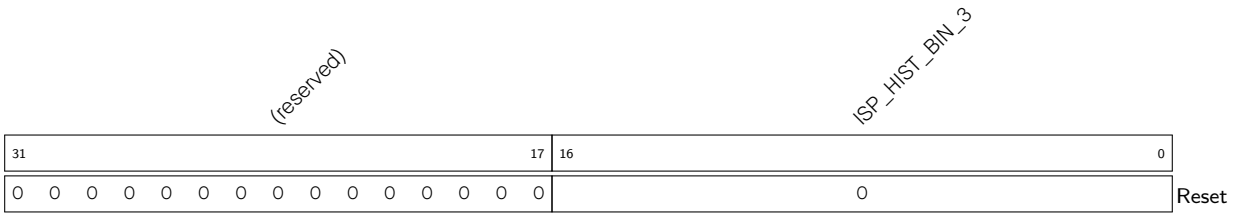
ISP_HIST_BIN_1 表示 HIST 统计窗口内亮度落在区间 1 的像素数量。(RO)

Register 38.126. ISP_HIST_BIN2_REG (0x01E8)



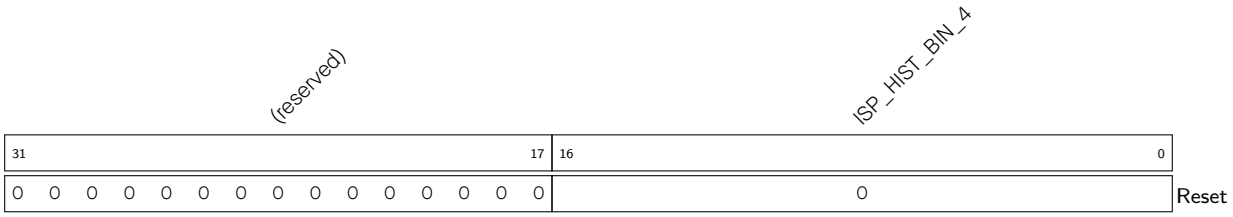
ISP_HIST_BIN_2 表示 HIST 统计窗口内亮度落在区间 2 的像素数量。(RO)

Register 38.127. ISP_HIST_BIN3_REG (0x01EC)



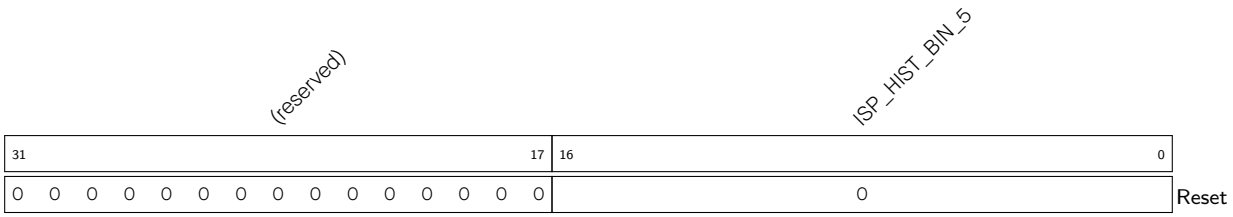
ISP_HIST_BIN_3 表示 HIST 统计窗口内亮度落在区间 3 的像素数量。(RO)

Register 38.128. ISP_HIST_BIN4_REG (0x01F0)



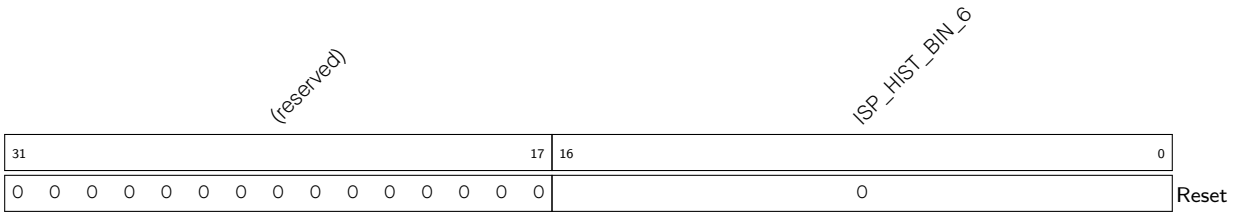
ISP_HIST_BIN_4 表示 HIST 统计窗口内亮度落在区间 4 的像素数量。(RO)

Register 38.129. ISP_HIST_BIN5_REG (0x01F4)



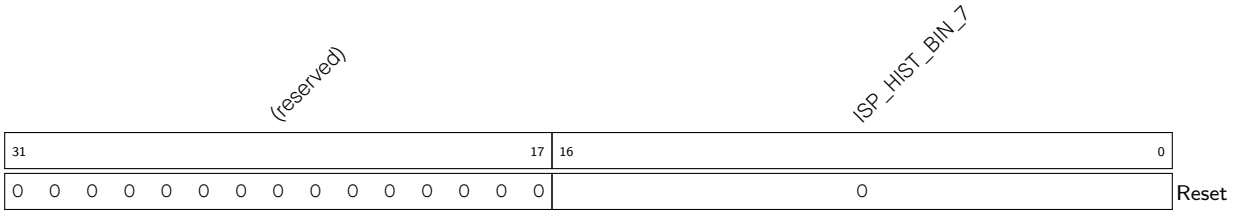
ISP_HIST_BIN_5 表示 HIST 统计窗口内亮度落在区间 5 的像素数量。(RO)

Register 38.130. ISP_HIST_BIN6_REG (0x01F8)



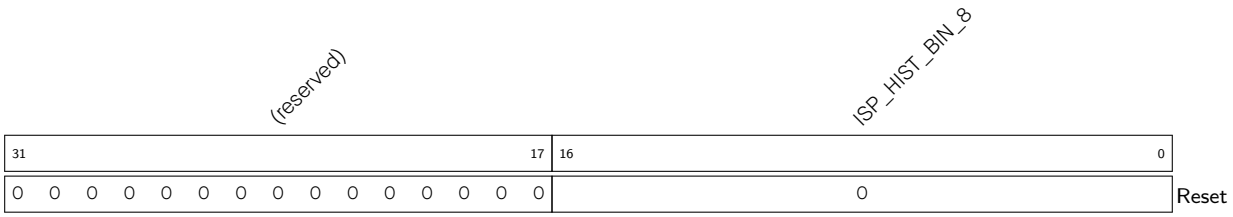
ISP_HIST_BIN_6 表示 HIST 统计窗口内亮度落在区间 6 的像素数量。(RO)

Register 38.131. ISP_HIST_BIN7_REG (0x01FC)



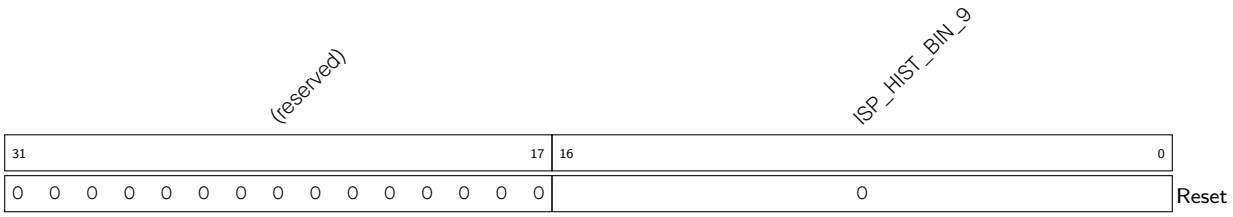
ISP_HIST_BIN_7 表示 HIST 统计窗口内亮度落在区间 7 的像素数量。(RO)

Register 38.132. ISP_HIST_BIN8_REG (0x0200)



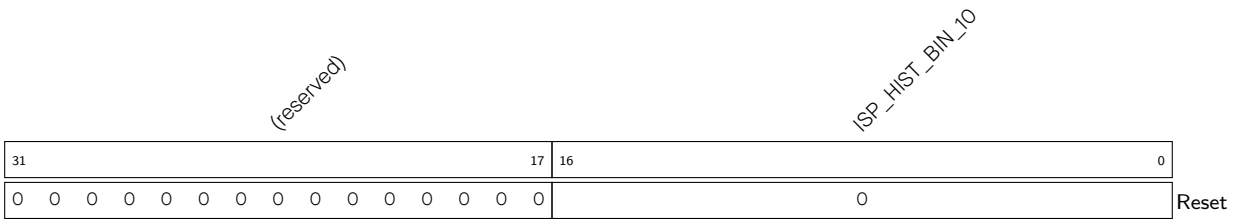
ISP_HIST_BIN_8 表示 HIST 统计窗口内亮度落在区间 8 的像素数量。(RO)

Register 38.133. ISP_HIST_BIN9_REG (0x0204)



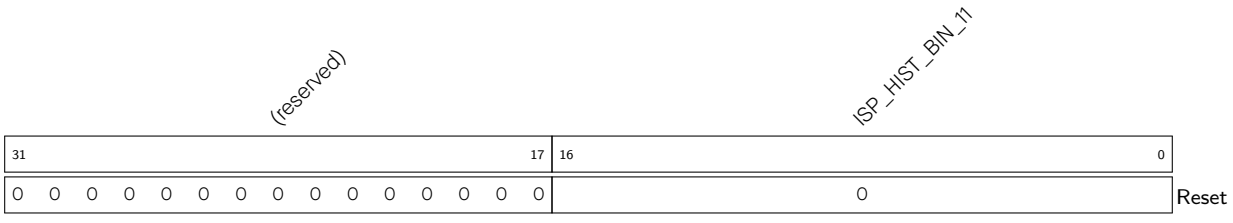
ISP_HIST_BIN_9 表示 HIST 统计窗口内亮度落在区间 9 的像素数量。(RO)

Register 38.134. ISP_HIST_BIN10_REG (0x0208)



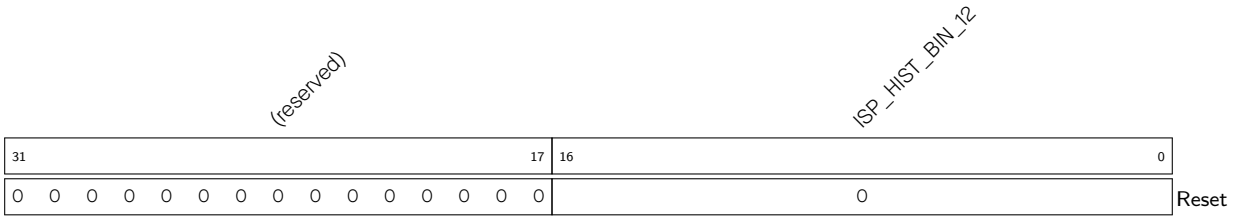
ISP_HIST_BIN_10 表示 HIST 统计窗口内亮度落在区间 10 的像素数量。(RO)

Register 38.135. ISP_HIST_BIN11_REG (0x020C)



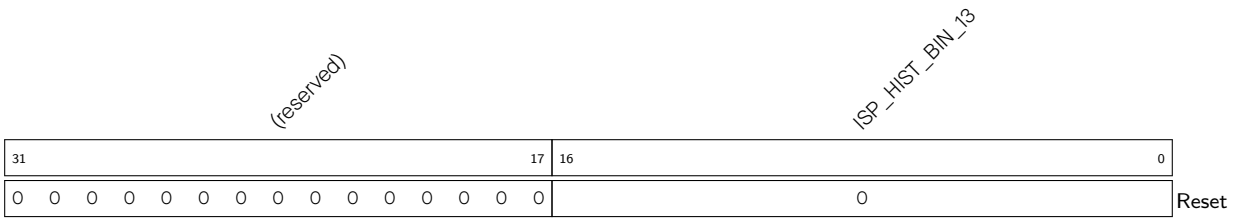
ISP_HIST_BIN_11 表示 HIST 统计窗口内亮度落在区间 11 的像素数量。(RO)

Register 38.136. ISP_HIST_BIN12_REG (0x0210)



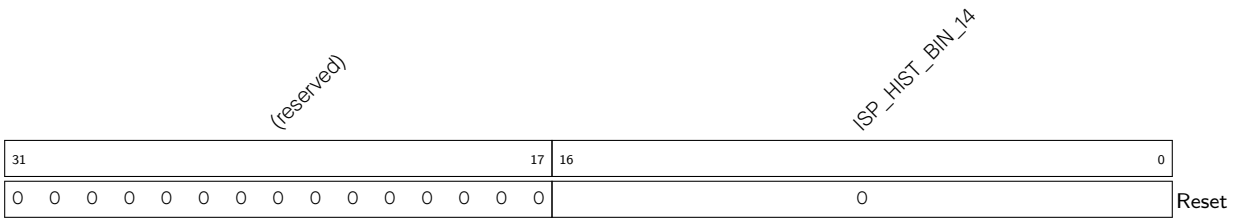
ISP_HIST_BIN_12 表示 HIST 统计窗口内亮度落在区间 12 的像素数量。(RO)

Register 38.137. ISP_HIST_BIN13_REG (0x0214)



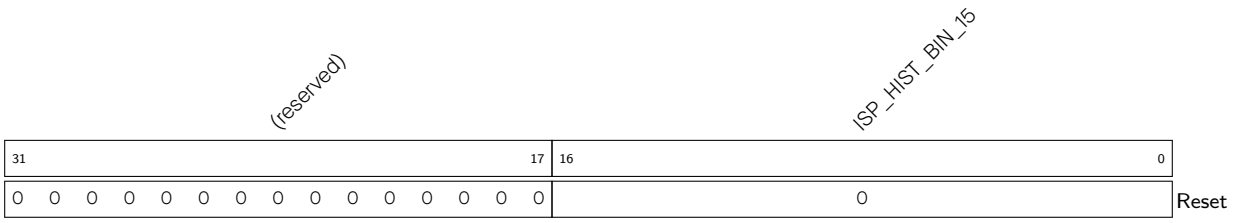
ISP_HIST_BIN_13 表示 HIST 统计窗口内亮度落在区间 13 的像素数量。(RO)

Register 38.138. ISP_HIST_BIN14_REG (0x0218)



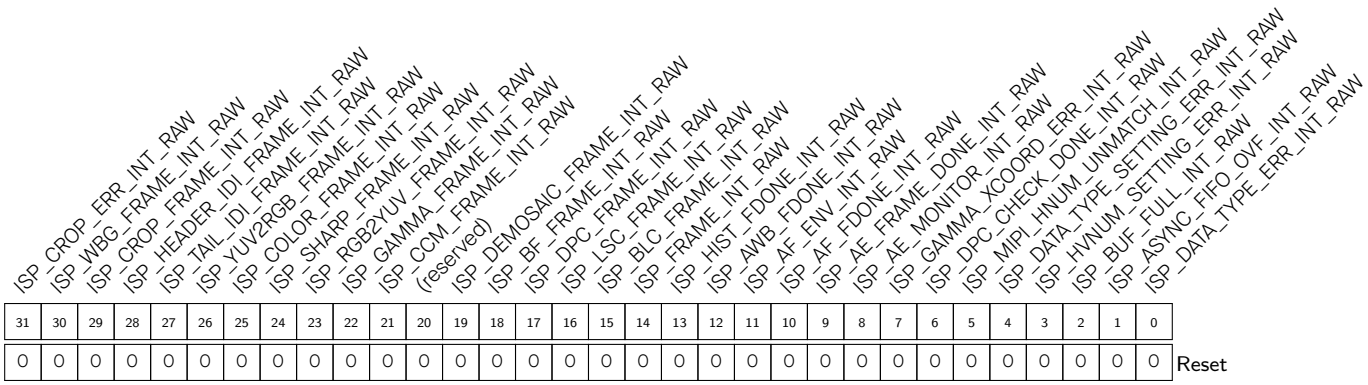
ISP_HIST_BIN_14 表示 HIST 统计窗口内亮度落在区间 14 的像素数量。(RO)

Register 38.139. ISP_HIST_BIN15_REG (0x021C)



ISP_HIST_BIN_15 表示 HIST 统计窗口内亮度落在区间 15 的像素数量。(RO)

Register 38.140. ISP_INT_RAW_REG (0x0064)



ISP_DATA_TYPE_ERR_INT_RAW ISP_DATA_TYPE_ERR_INT 的原始中断状态 (R/SS/WTC)

ISP_ASYNC_FIFO_OVF_INT_RAW ISP_ASYNC_FIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

ISP_BUF_FULL_INT_RAW ISP_BUF_FULL_INT 的原始中断状态。(R/SS/WTC)

ISP_HVNUM_SETTING_ERR_INT_RAW ISP_HVNUM_SETTING_ERR_INT 的原始中断状态。(R/SS/WTC)

ISP_DATA_TYPE_SETTING_ERR_INT_RAW ISP_DATA_TYPE_SETTING_ERR_INT 的原始中断状态。(R/SS/WTC)

ISP_MIPI_HNUM_UNMATCH_INT_RAW ISP_MIPI_HNUM_UNMATCH_INT 的原始中断状态。(R/SS/WTC)

ISP_DPC_CHECK_DONE_INT_RAW ISP_DPC_CHECK_DONE_INT 的原始中断状态。(R/SS/WTC)

ISP_GAMMA_XCOORD_ERR_INT_RAW ISP_GAMMA_XCOORD_ERR_INT 的原始中断状态。(R/SS/WTC)

ISP_AE_MONITOR_INT_RAW ISP_AE_MONITOR_INT 的原始中断状态。(R/SS/WTC)

ISP_AE_FRAME_DONE_INT_RAW ISP_AE_FRAME_DONE_INT 的原始中断状态。(R/SS/WTC)

ISP_AF_FDONE_INT_RAW ISP_AF_FDONE_INT 的原始中断状态。(R/SS/WTC)

ISP_AF_ENV_INT_RAW ISP_AF_ENV_INT 的原始中断状态。(R/SS/WTC)

ISP_AWB_FDONE_INT_RAW ISP_AWB_FDONE_INT 的原始中断状态。(R/SS/WTC)

ISP_HIST_FDONE_INT_RAW ISP_HIST_FDONE_INT 的原始中断状态。(R/SS/WTC)

ISP_FRAME_INT_RAW ISP_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_BLC_FRAME_INT_RAW ISP_BLC_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_LSC_FRAME_INT_RAW ISP_LSC_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_DPC_FRAME_INT_RAW ISP_DPC_FRAME_INT 的原始中断状态。(R/SS/WTC)

见下页...

Register 38.140. ISP_INT_RAW_REG (0x0064)

接上页...

ISP_BF_FRAME_INT_RAW ISP_BF_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_DEMOSAIC_FRAME_INT_RAW ISP_DEMOSAIC_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_CCM_FRAME_INT_RAW ISP_CCM_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_GAMMA_FRAME_INT_RAW ISP_GAMMA_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_RGB2YUV_FRAME_INT_RAW ISP_RGB2YUV_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_SHARP_FRAME_INT_RAW ISP_SHARP_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_COLOR_FRAME_INT_RAW ISP_COLOR_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_YUV2RGB_FRAME_INT_RAW ISP_YUV2RGB_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_TAIL_IDI_FRAME_INT_RAW ISP_TAIL_IDI_FRAME_INT 的原始中断状态。(R/SS/WTC)

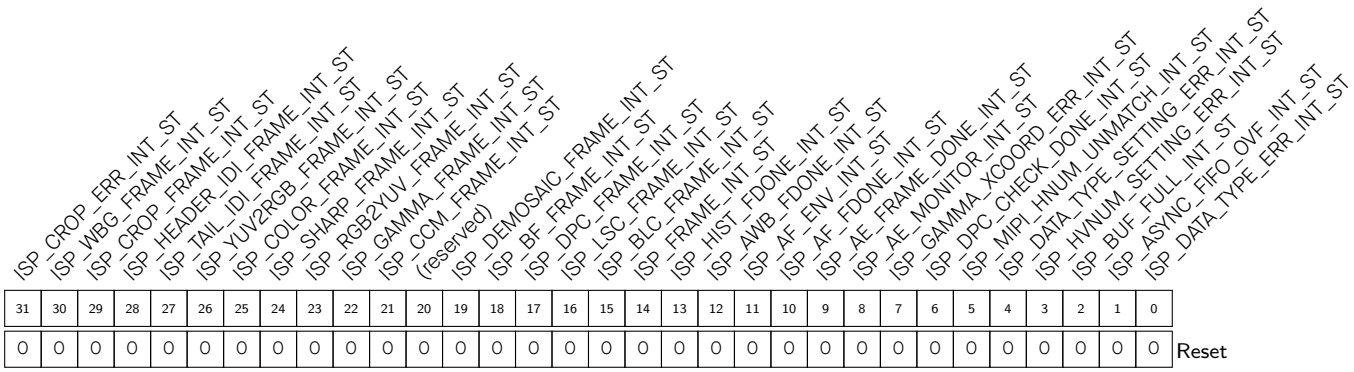
ISP_HEADER_IDI_FRAME_INT_RAW ISP_HEADER_IDI_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_CROP_FRAME_INT_RAW ISP_CROP_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_WBG_FRAME_INT_RAW ISP_WBG_FRAME_INT 的原始中断状态。(R/SS/WTC)

ISP_CROP_ERR_INT_RAW ISP_CROP_ERR_INT 的原始中断状态。(R/SS/WTC)

Register 38.141. ISP_INT_ST_REG (0x0068)



ISP_DATA_TYPE_ERR_INT_ST ISP_DATA_TYPE_ERR_INT 的可屏蔽中断状态。(RO)

ISP_ASYNC_FIFO_OVF_INT_ST ISP_ASYNC_FIFO_OVF_INT 的可屏蔽中断状态。(RO)

ISP_BUF_FULL_INT_ST ISP_BUF_FULL_INT 的可屏蔽中断状态。(RO)

ISP_HVNUM_SETTING_ERR_INT_ST ISP_HVNUM_SETTING_ERR_INT 的可屏蔽中断状态。(RO)

ISP_DATA_TYPE_SETTING_ERR_INT_ST ISP_DATA_TYPE_SETTING_ERR_INT 的可屏蔽中断状态。(RO)

ISP_MIPI_HNUM_UNMATCH_INT_ST ISP_MIPI_HNUM_UNMATCH_INT 的可屏蔽中断状态。(RO)

ISP_DPC_CHECK_DONE_INT_ST ISP_DPC_CHECK_DONE_INT 的可屏蔽中断状态。(RO)

ISP_GAMMA_XCOORD_ERR_INT_ST ISP_GAMMA_XCOORD_ERR_INT 的可屏蔽中断状态。(RO)

ISP_AE_MONITOR_INT_ST ISP_AE_MONITOR_INT 的可屏蔽中断状态。(RO)

ISP_AE_FRAME_DONE_INT_ST ISP_AE_FRAME_DONE_INT 的可屏蔽中断状态。(RO)

ISP_AF_FDONE_INT_ST ISP_AF_FDONE_INT 的可屏蔽中断状态。(RO)

ISP_AF_ENV_INT_ST ISP_AF_ENV_INT 的可屏蔽中断状态。(RO)

ISP_AWB_FDONE_INT_ST ISP_AWB_FDONE_INT 的可屏蔽中断状态。(RO)

ISP_HIST_FDONE_INT_ST ISP_HIST_FDONE_INT 的可屏蔽中断状态。(RO)

ISP_FRAME_INT_ST ISP_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_BLC_FRAME_INT_ST ISP_BLC_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_LSC_FRAME_INT_ST ISP_LSC_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_DPC_FRAME_INT_ST ISP_DPC_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_BF_FRAME_INT_ST ISP_BF_FRAME_INT 的可屏蔽中断状态。(RO)

见下页...

Register 38.141. ISP_INT_ST_REG (0x0068)

接上页...

ISP_DEMOSAIC_FRAME_INT_ST ISP_DEMOSAIC_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_CCM_FRAME_INT_ST ISP_CCM_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_GAMMA_FRAME_INT_ST ISP_GAMMA_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_RGB2YUV_FRAME_INT_ST ISP_RGB2YUV_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_SHARP_FRAME_INT_ST ISP_SHARP_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_COLOR_FRAME_INT_ST ISP_COLOR_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_YUV2RGB_FRAME_INT_ST ISP_YUV2RGB_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_TAIL_IDI_FRAME_INT_ST ISP_TAIL_IDI_FRAME_INT 的可屏蔽中断状态。(RO)

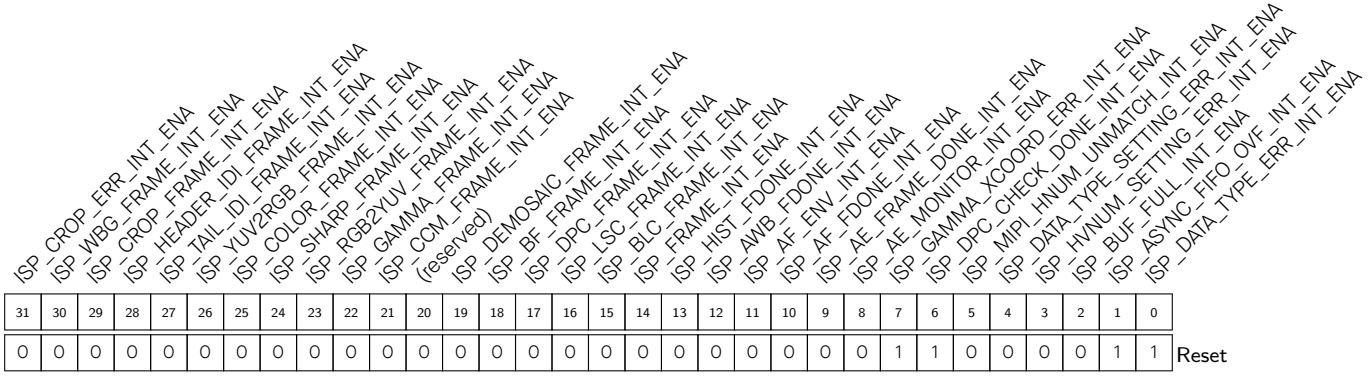
ISP_HEADER_IDI_FRAME_INT_ST ISP_HEADER_IDI_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_CROP_FRAME_INT_ST ISP_CROP_FRAME 的可屏蔽中断状态。(RO)

ISP_WBG_FRAME_INT_ST ISP_WBG_FRAME_INT 的可屏蔽中断状态。(RO)

ISP_CROP_ERR_INT_ST ISP_CROP_ERR_INT 的可屏蔽中断状态。(RO)

Register 38.142. ISP_INT_ENA_REG (0x006C)



ISP_DATA_TYPE_ERR_INT_ENA 写 1 使能 ISP_DATA_TYPE_ERR_INT。(R/W)

ISP_ASYNC_FIFO_OVF_INT_ENA 写 1 使能 ISP_ASYNC_FIFO_OVF_INT。(R/W)

ISP_BUF_FULL_INT_ENA 写 1 使能 ISP_BUF_FULL_INT。(R/W)

ISP_HVNUM_SETTING_ERR_INT_ENA 写 1 使能 ISP_HVNUM_SETTING_ERR_INT。(R/W)

ISP_DATA_TYPE_SETTING_ERR_INT_ENA 写 1 使能 ISP_DATA_TYPE_SETTING_ERR_INT。(R/W)

ISP_MIPI_HNUM_UNMATCH_INT_ENA 写 1 使能 ISP_MIPI_HNUM_UNMATCH_INT。(R/W)

ISP_DPC_CHECK_DONE_INT_ENA 写 1 使能 ISP_DPC_CHECK_DONE_INT。(R/W)

ISP_GAMMA_XCOORD_ERR_INT_ENA 写 1 使能 ISP_GAMMA_XCOORD_ERR_INT。(R/W)

ISP_AE_MONITOR_INT_ENA 写 1 使能 ISP_AE_MONITOR_INT。(R/W)

ISP_AE_FRAME_DONE_INT_ENA 写 1 使能 ISP_AE_FRAME_DONE_INT。(R/W)

ISP_AF_FDONE_INT_ENA 写 1 使能 ISP_AF_FDONE_INT。(R/W)

ISP_AF_ENV_INT_ENA 写 1 使能 ISP_AF_ENV_INT。(R/W)

ISP_AWB_FDONE_INT_ENA 写 1 使能 ISP_AWB_FDONE_INT。(R/W)

ISP_HIST_FDONE_INT_ENA 写 1 使能 ISP_HIST_FDONE_INT。(R/W)

ISP_FRAME_INT_ENA 写 1 使能 ISP_FRAME_INT。(R/W)

ISP_BLC_FRAME_INT_ENA 写 1 使能 ISP_BLC_FRAME_INT。(R/W)

ISP_LSC_FRAME_INT_ENA 写 1 使能 ISP_LSC_FRAME_INT。(R/W)

ISP_DPC_FRAME_INT_ENA 写 1 使能 ISP_DPC_FRAME_INT。(R/W)

ISP_BF_FRAME_INT_ENA 写 1 使能 ISP_BF_FRAME_INT。(R/W)

ISP_DEMOSAIC_FRAME_INT_ENA 写 1 使能 ISP_DEMOSAIC_FRAME_INT。(R/W)

ISP_CCM_FRAME_INT_ENA 写 1 使能 ISP_CCM_FRAME_INT。(R/W)

见下页...

Register 38.142. ISP_INT_ENA_REG (0x006C)

接上页...

ISP_GAMMA_FRAME_INT_ENA 写 1 使能 ISP_GAMMA_FRAME_INT。(R/W)

ISP_RGB2YUV_FRAME_INT_ENA 写 1 使能 ISP_RGB2YUV_FRAME_INT。(R/W)

ISP_SHARP_FRAME_INT_ENA 写 1 使能 ISP_SHARP_FRAME_INT。(R/W)

ISP_COLOR_FRAME_INT_ENA 写 1 使能 ISP_COLOR_FRAME_INT。(R/W)

ISP_YUV2RGB_FRAME_INT_ENA 写 1 使能 ISP_YUV2RGB_FRAME_INT。(R/W)

ISP_TAIL_IDI_FRAME_INT_ENA 写 1 使能 ISP_TAIL_IDI_FRAME_INT。(R/W)

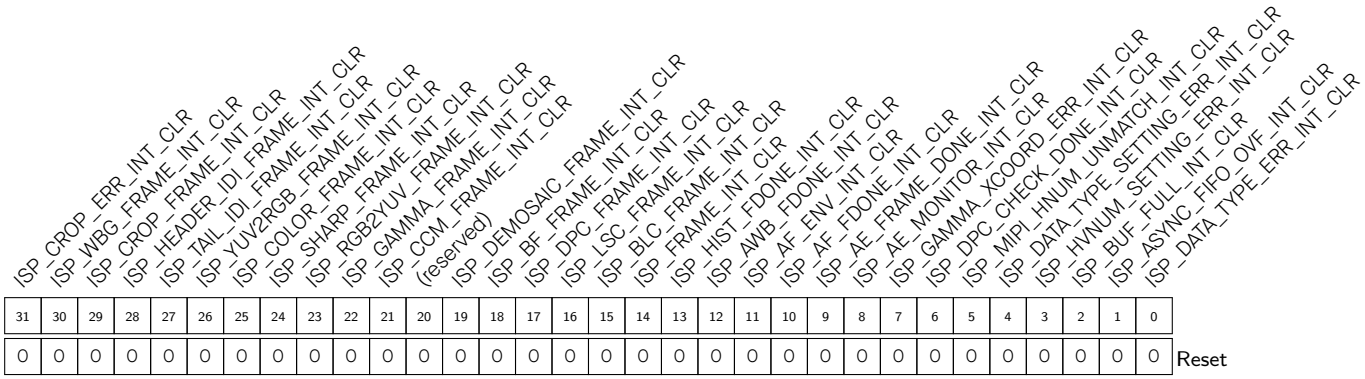
ISP_HEADER_IDI_FRAME_INT_ENA 写 1 使能 ISP_HEADER_IDI_FRAME_INT。(R/W)

ISP_CROP_FRAME_INT_ENA 写 1 使能 ISP_CROP_FRAME_INT。(R/W)

ISP_WBG_FRAME_INT_ENA 写 1 使能 ISP_WBG_FRAME_INT。(R/W)

ISP_CROP_ERR_INT_ENA 写 1 使能 ISP_CROP_ERR_INT。(R/W)

Register 38.143. ISP_INT_CLR_REG (0x0070)



ISP_DATA_TYPE_ERR_INT_CLR 写 1 清除 ISP_DATA_TYPE_ERR_INT。(WT)

ISP_ASYNC_FIFO_OVF_INT_CLR 写 1 清除 ISP_ASYNC_FIFO_OVF_INT。(WT)

ISP_BUF_FULL_INT_CLR 写 1 清除 ISP_BUF_FULL_INT。(WT)

ISP_HVNUM_SETTING_ERR_INT_CLR 写 1 清除 ISP_HVNUM_SETTING_ERR_INT。(WT)

ISP_DATA_TYPE_SETTING_ERR_INT_CLR 写 1 清除 ISP_DATA_TYPE_SETTING_ERR_INT。(WT)

ISP_MIPI_HNUM_UNMATCH_INT_CLR 写 1 清除 ISP_MIPI_HNUM_UNMATCH_INT。(WT)

ISP_DPC_CHECK_DONE_INT_CLR 写 1 清除 ISP_DPC_CHECK_DONE_INT。(WT)

ISP_GAMMA_XCOORD_ERR_INT_CLR 写 1 清除 ISP_GAMMA_XCOORD_ERR_INT。(WT)

ISP_AE_MONITOR_INT_CLR 写 1 清除 ISP_AE_MONITOR_INT。(WT)

ISP_AE_FRAME_DONE_INT_CLR 写 1 清除 ISP_AE_FRAME_DONE_INT。(WT)

ISP_AF_FDONE_INT_CLR 写 1 清除 ISP_AF_FDONE_INT。(WT)

ISP_AF_ENV_INT_CLR 写 1 清除 ISP_AF_ENV_INT。(WT)

ISP_AWB_FDONE_INT_CLR 写 1 清除 ISP_AWB_FDONE_INT。(WT)

ISP_HIST_FDONE_INT_CLR 写 1 清除 ISP_HIST_FDONE_INT。(WT)

ISP_FRAME_INT_CLR 写 1 清除 ISP_FRAME_INT。(WT)

ISP_BLC_FRAME_INT_CLR 写 1 清除 ISP_BLC_FRAME_INT。(WT)

ISP_LSC_FRAME_INT_CLR 写 1 清除 ISP_LSC_FRAME_INT。(WT)

ISP_DPC_FRAME_INT_CLR 写 1 清除 ISP_DPC_FRAME_INT。(WT)

ISP_BF_FRAME_INT_CLR 写 1 清除 ISP_BF_FRAME_INT。(WT)

ISP_DEMOSAIC_FRAME_INT_CLR 写 1 清除 ISP_DEMOSAIC_FRAME_INT。(WT)

ISP_CCM_FRAME_INT_CLR 写 1 清除 ISP_CCM_FRAME_INT。(WT)

见下页...

Register 38.143. ISP_INT_CLR_REG (0x0070)

接上页...

ISP_GAMMA_FRAME_INT_CLR 写 1 清除 ISP_GAMMA_FRAME_INT。(WT)

ISP_RGB2YUV_FRAME_INT_CLR 写 1 清除 ISP_RGB2YUV_FRAME_INT。(WT)

ISP_SHARP_FRAME_INT_CLR 写 1 清除 ISP_SHARP_FRAME_INT。(WT)

ISP_COLOR_FRAME_INT_CLR 写 1 清除 ISP_COLOR_FRAME_INT。(WT)

ISP_YUV2RGB_FRAME_INT_CLR 写 1 清除 ISP_YUV2RGB_FRAME_INT。(WT)

ISP_TAIL_IDI_FRAME_INT_CLR 写 1 清除 ISP_TAIL_IDI_FRAME_INT。(WT)

ISP_HEADER_IDI_FRAME_INT_CLR 写 1 清除 ISP_HEADER_IDI_FRAME_INT。(WT)

ISP_CROP_FRAME_INT_CLR 写 1 清除 ISP_CROP_FRAME_INT。(WT)

ISP_WBG_FRAME_INT_CLR 写 1 清除 ISP_WBG_FRAME_INT。(WT)

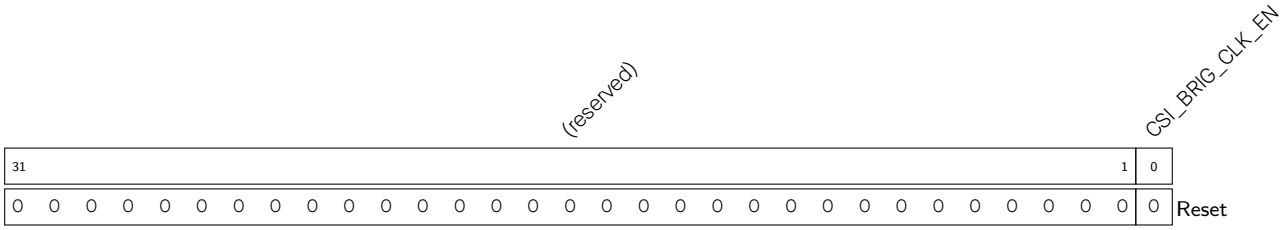
ISP_CROP_ERR_INT_CLR 写 1 清除 ISP_CROP_ERR_INT。(WT)

38.9.2 MIPI CSI_Bridge 寄存器

本小节的所有地址均为相对于 MIPI CSI_Bridge 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 38.144. CSI_BRIG_CLK_EN_REG (0x0000)



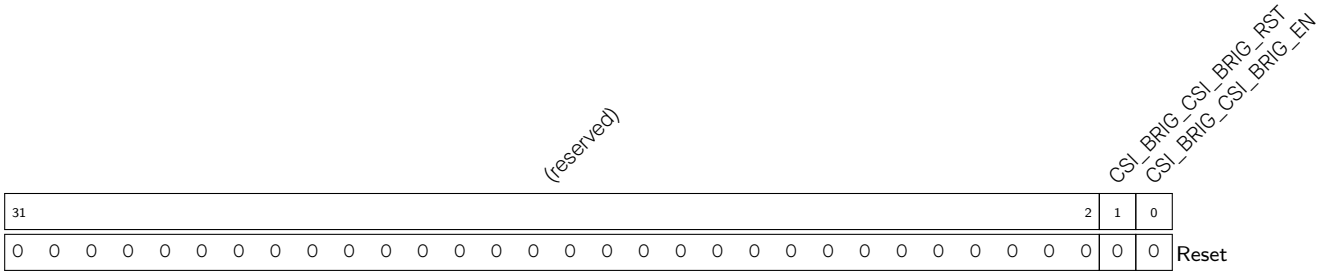
CSI_BRIG_CLK_EN 配置 CSI_Bridge 寄存器时钟是否常开。

0：时钟只在寄存器访问时打开

1：时钟常开

(R/W)

Register 38.145. CSI_BRIG_CSI_EN_REG (0x0004)



CSI_BRIG_CSI_BRIG_EN 配置是否使能 CSI_Bridge。

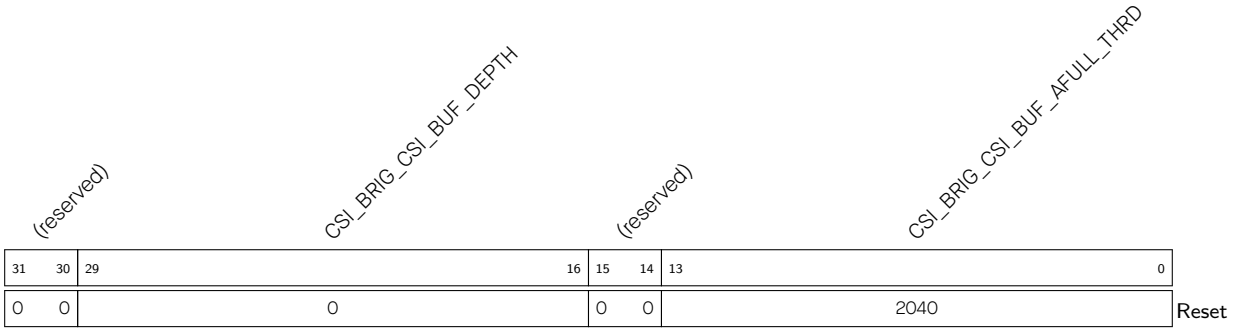
0：禁用

1：使能

(R/W)

CSI_BRIG_CSI_BRIG_RST 0: release csi bridge reset. 1: enable csi bridge reset. (R/W)

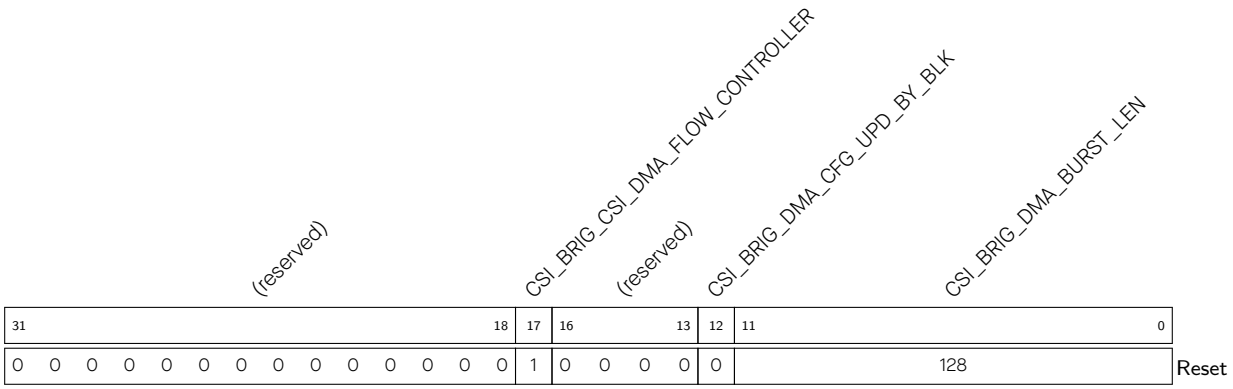
Register 38.146. CSI_BRIG_BUF_FLOW_CTL_REG (0x000C)



CSI_BRIG_CSI_BUF_AFULL_THRD 配置 CSI_Bridge 缓存即将填满的大小阈值。(R/W)

CSI_BRIG_CSI_BUF_DEPTH 表示 CSI_Bridge 缓存当前使用的大小。(RO)

Register 38.147. CSI_BRIG_DMA_REQ_CFG_REG (0x0008)



CSI_BRIG_DMA_BURST_LEN 配置一次 VDMA 传输 burst 长度包含的 64 位数据量。(R/W)

CSI_BRIG_DMA_CFG_UPD_BY_BLK 配置 CSI_BRIG_DMA_BURST_LEN 和 CSI_BRIG_DMABLK_SIZE 的更新时间。

0：每次 VDMA 块传输完成更新

1：每一帧传输结束更新

(R/W)

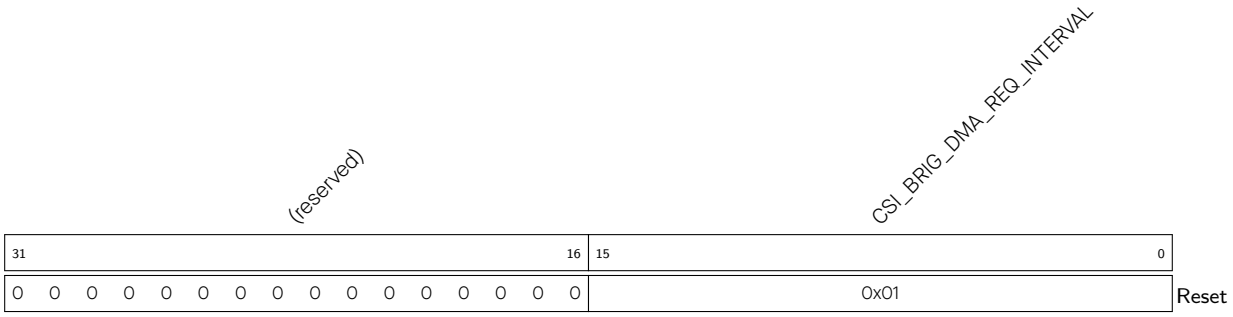
CSI_BRIG_CSI_DMA_FLOW_CONTROLLER 配置 DMA 流控制器。

0：DMA 作为流控制器

1：CSI_bridge 作为流控制器

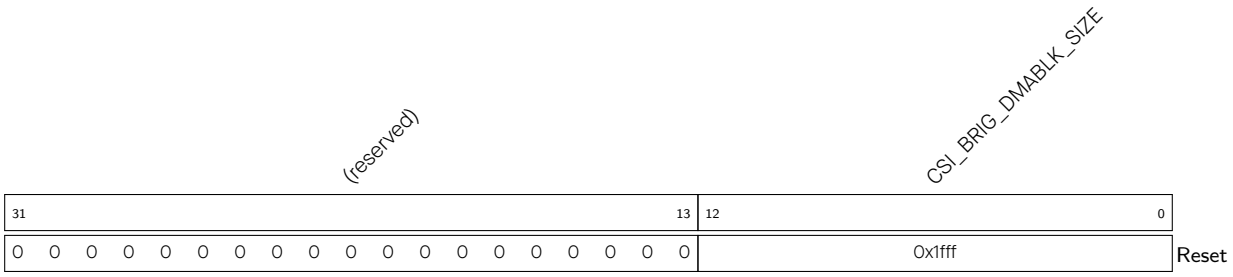
(R/W)

Register 38.148. CSI_BRIG_DMA_REQ_INTERVAL_REG (0x002C)



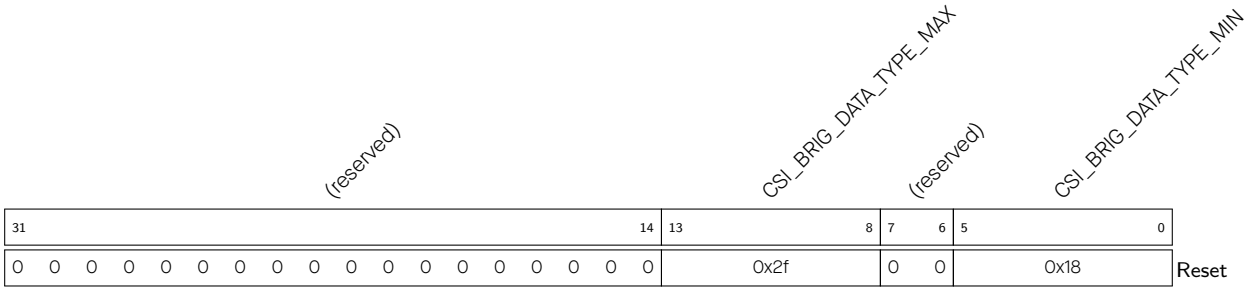
CSI_BRIG_DMA_REQ_INTERVAL 配置 VDMA 请求间隔。16'b1：1 个时钟周期，16'b11：2 个时钟周期，16'hFFFF：16 个时钟周期，以此类推。(R/W)

Register 38.149. CSI_BRIG_DMABLK_SIZE_REG (0x0030)



CSI_BRIG_DMABLK_SIZE 配置一个 VDMA 块传输包含的 VDMA burst 传输数量。(R/W)

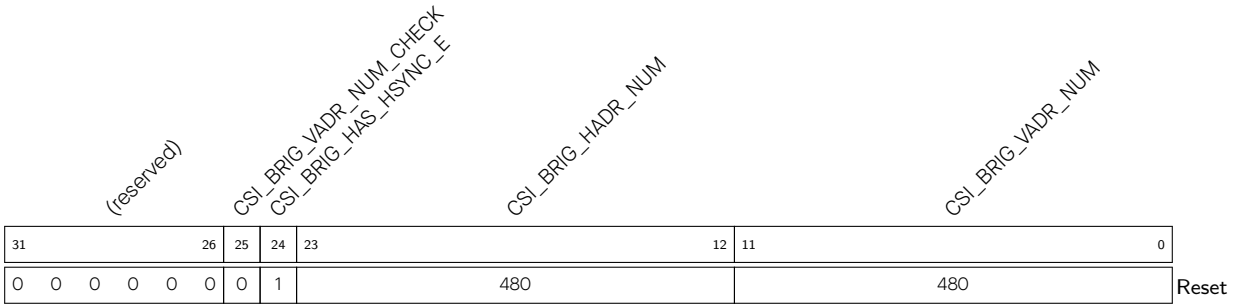
Register 38.150. CSI_BRIG_DATA_TYPE_CFG_REG (0x0010)



CSI_BRIG_DATA_TYPE_MIN 配置有效像素的 DATA_TYPE 最小值。(R/W)

CSI_BRIG_DATA_TYPE_MAX 配置有效像素的 DATA_TYPE 最大值。(R/W)

Register 38.151. CSI_BRIG_FRAME_CFG_REG (0x0014)



CSI_BRIG_VADR_NUM 配置一帧图像的高度，单位：行。(R/W)

CSI_BRIG_HADR_NUM 配置一帧图像宽度包含的 64 位数据数量。(R/W)

CSI_BRIG_HAS_HSYNC_E 配置 Image Interface 64 输入是否包含 HSYNC_START 和 HSYNC_END 包。

0：不包含

1：包含

(R/W)

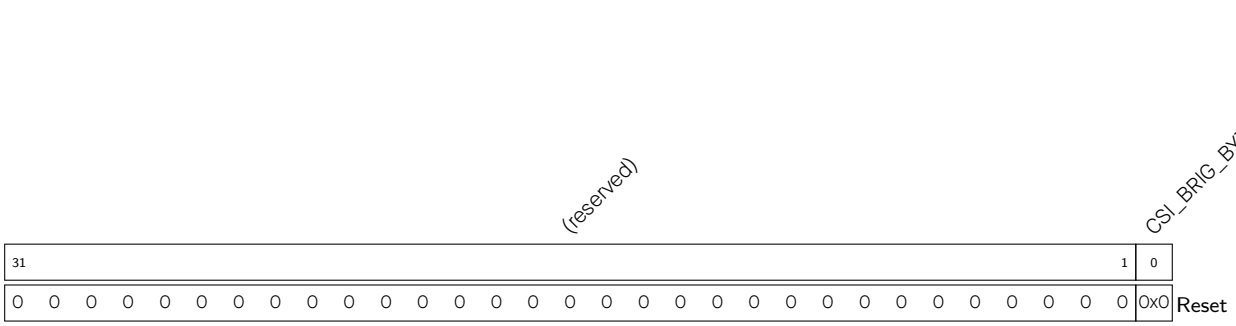
CSI_BRIG_VADR_NUM_CHECK 配置是否使能行数检查。

0：禁用

1：使能

(R/W)

Register 38.152. CSI_BRIG_ENDIAN_MODE_REG (0x0018)



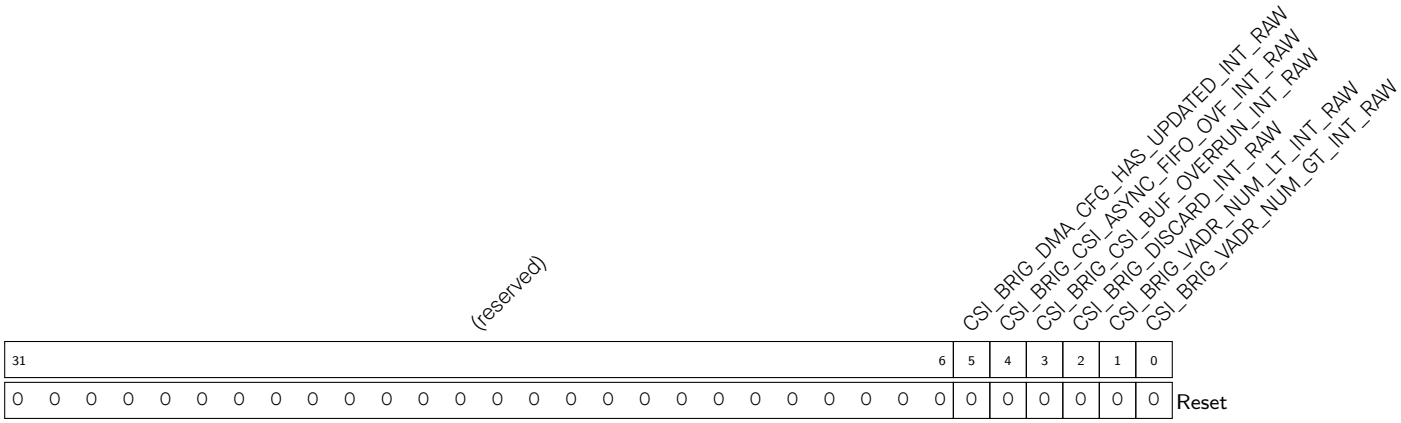
CSI_BRIG_BYTE_ENDIAN_ORDER 配置 64 位数据的字节顺序。

0：顺序不变

1：高低字节顺序翻转

(R/W)

Register 38.153. CSI_BRIG_INT_RAW_REG (0x001C)



CSI_BRIG_VADR_NUM_GT_INT_RAW CSI_BRIG_VADR_NUM_GT_INT 的原始中断状态。(R/WTC/SS)

CSI_BRIG_VADR_NUM_LT_INT_RAW CSI_BRIG_VADR_NUM_LT_INT 的原始中断状态。(R/WTC/SS)

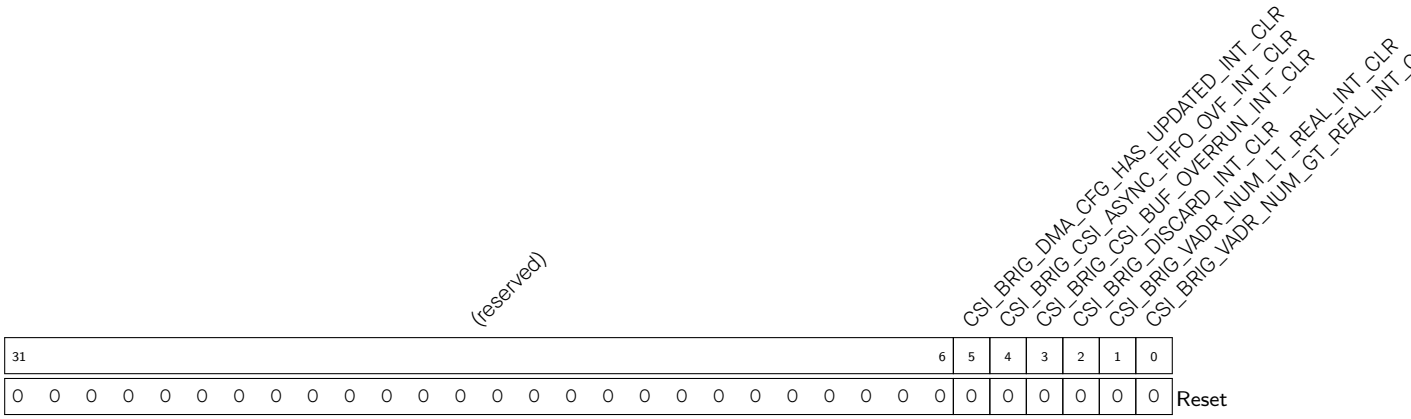
CSI_BRIG_DISCARD_INT_RAW CSI_BRIG_DISCARD_INT 的原始中断状态。(R/WTC/SS)

CSI_BRIG_CSI_BUF_OVERRUN_INT_RAW CSI_BRIG_CSI_BUF_OVERRUN_INT 的原始中断状态。(R/WTC/SS)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_RAW CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_RAW CSI_BRIG_DMA_CFG_HAS_UPDATED_INT 的原始中断状态。(R/WTC/SS)

Register 38.154. CSI_BRIG_INT_CLR_REG (0x0020)



CSI_BRIG_VADR_NUM_GT_REAL_INT_CLR 写 1 清除 CSI_BRIG_VADR_NUM_GT_REAL_INT 中断。(WT)

CSI_BRIG_VADR_NUM_LT_REAL_INT_CLR 写 1 清除 CSI_BRIG_VADR_NUM_LT_REAL_INT 中断。(WT)

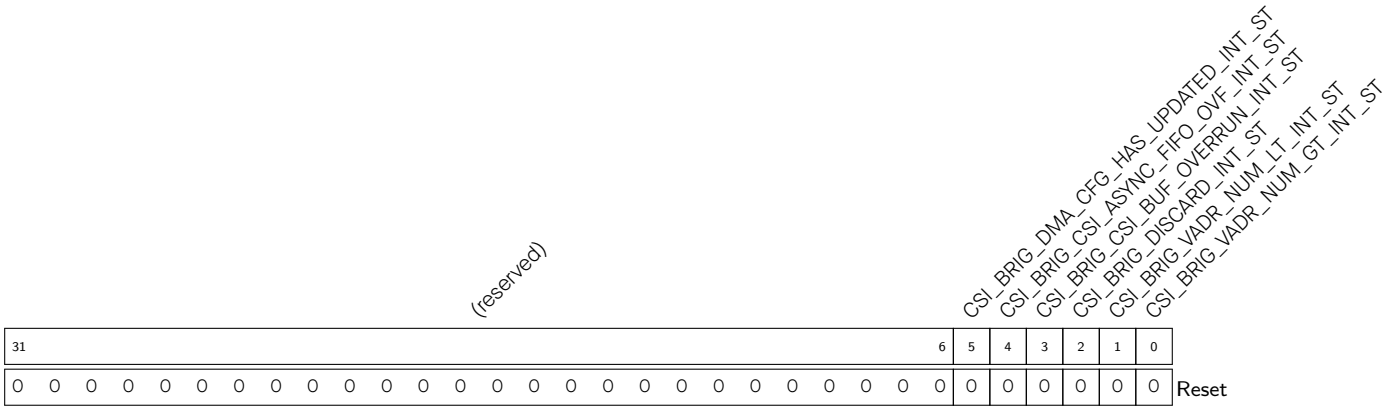
CSI_BRIG_DISCARD_INT_CLR 写 1 清除 CSI_BRIG_DISCARD_INT 中断。(WT)

CSI_BRIG_CSI_BUF_OVERRUN_INT_CLR 写 1 清除 CSI_BRIG_CSI_BUF_OVERRUN_INT 中断。(WT)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_CLR 写 1 清除 CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT 中断。(WT)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_CLR 写 1 清除 CSI_BRIG_DMA_CFG_HAS_UPDATED_INT 中断。(WT)

Register 38.155. CSI_BRIG_INT_ST_REG (0x0024)



CSI_BRIG_VADR_NUM_GT_INT_ST CSI_BRIG_VADR_NUM_GT_INT 的可屏蔽中断状态。(RO)

CSI_BRIG_VADR_NUM_LT_INT_ST CSI_BRIG_VADR_NUM_LT_INT 的可屏蔽中断状态。(RO)

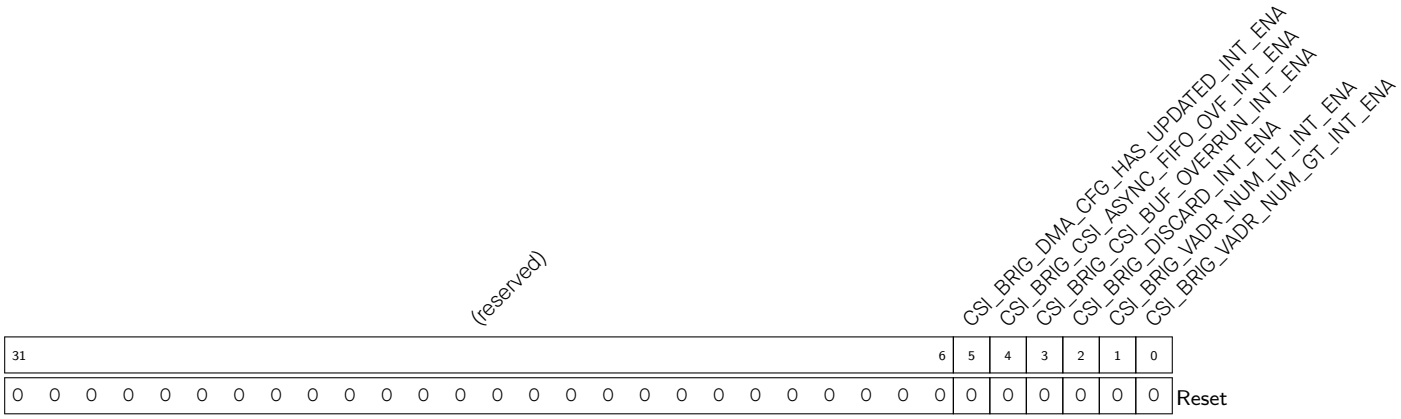
CSI_BRIG_DISCARD_INT_ST CSI_BRIG_DISCARD_INT 的可屏蔽中断状态。(RO)

CSI_BRIG_CSI_BUF_OVERRUN_INT_ST CSI_BRIG_CSI_BUF_OVERRUN_INT 的可屏蔽中断状态。(RO)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_ST CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT 的可屏蔽中断状态。(RO)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_ST CSI_BRIG_DMA_CFG_HAS_UPDATED_INT 的可屏蔽中断状态。(RO)

Register 38.156. CSI_BRIG_INT_ENA_REG (0x0028)



CSI_BRIG_VADR_NUM_GT_INT_ENA 写 1 使能 CSI_BRIG_VADR_NUM_GT_INT 中断。(R/W)

CSI_BRIG_VADR_NUM_LT_INT_ENA 写 1 使能 CSI_BRIG_VADR_NUM_LT_INT 中断。(R/W)

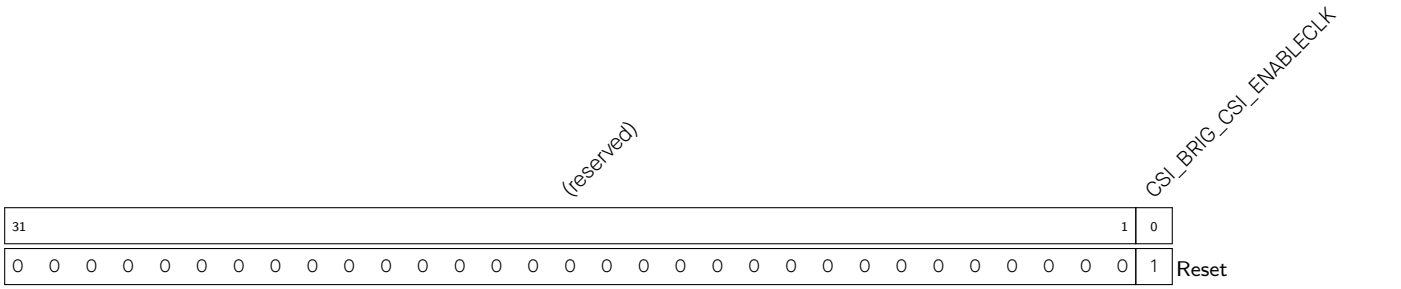
CSI_BRIG_DISCARD_INT_ENA 写 1 使能 CSI_BRIG_DISCARD_INT 中断。(R/W)

CSI_BRIG_CSI_BUF_OVERRUN_INT_ENA 写 1 使能 CSI_BRIG_CSI_BUF_OVERRUN_INT 中断。(R/W)

CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT_ENA 写 1 使能 CSI_BRIG_CSI_ASYNC_FIFO_OVF_INT 中断。(R/W)

CSI_BRIG_DMA_CFG_HAS_UPDATED_INT_ENA 写 1 使能 CSI_BRIG_DMA_CFG_HAS_UPDATED_INT 中断。(R/W)

Register 38.157. CSI_BRIG_HOST_CTRL_REG (0x0040)



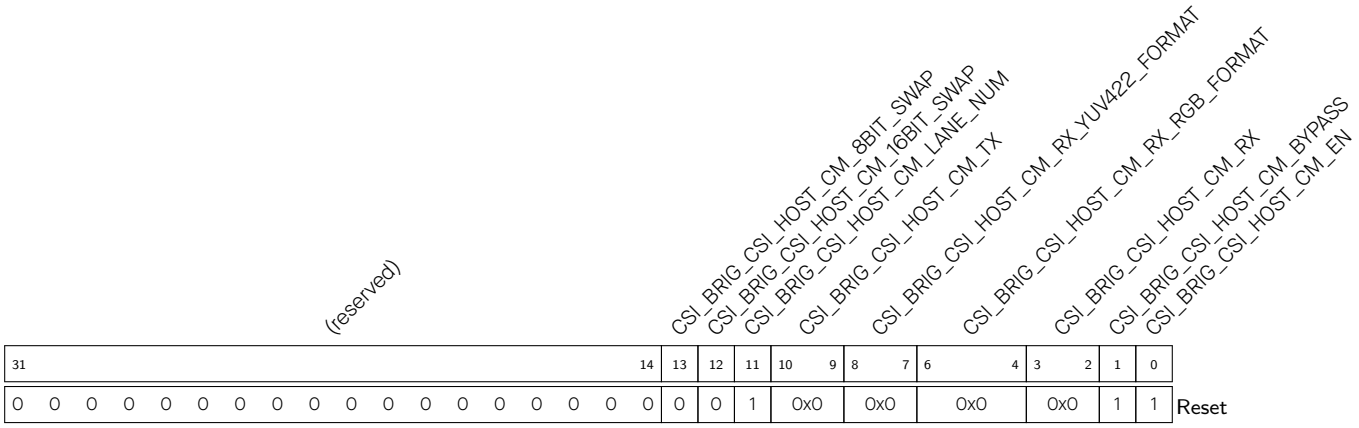
CSI_BRIG_CSI_ENABLECLK 配置是否使能 CSI PHY 的时钟通道。

0：禁用

1：使能

(R/W)

Register 38.158. CSI_BRIG_HOST_CM_CTRL_REG (0x0048)



CSI_BRIG_CSI_HOST_CM_EN 配置是否使能 CSI CM 功能。

0：关闭，此时 CSI CM 不会输出数据，后端 ISP 无法接收数据

1：使能

(R/W)

CSI_BRIG_CSI_HOST_CM_BYPASS 配置是否盘路 CSI CM 功能。

0：不盘路

1：盘路

(R/W)

CSI_BRIG_CSI_HOST_CM_RX 配置 CSI CM 输入图像格式。

0：RGB888

1：RGB565

2：YUV422

3：YUV420

(R/W)

CSI_BRIG_CSI_HOST_CM_RX_RGB_FORMAT 配置输入 RGB 的通道排序。

0：RGB

1：BGR

2：RGB

3：BRG

4：GRB

5：GBR

其他值：无效 (R/W)

CSI_BRIG_CSI_HOST_CM_RX_YUV422_FORMAT 配置输入 YUV422 的通道排序。

0：YVYU

1：YUYV

2：VYUY

3：UYVY

(R/W)

见下页...

Register 38.158. CSI_BRIG_HOST_CM_CTRL_REG (0x0048)

接上页...

CSI_BRIG_CSI_HOST_CM_TX 配置 CSI CM 输出图像格式。

0：RGB888

1：RGB565

2：YUV422

3：YUV420

(R/W)

CSI_BRIG_CSI_HOST_CM_LANE_NUM 配置 CSI 通道数量，只有输入输出均为 RGB888 的时候有效。

0：1 通道

1：2 通道

(R/W)

CSI_BRIG_CSI_HOST_CM_16BIT_SWAP 配置是否使能高低 16-bit 翻转。

0：关闭

1：使能

(R/W)

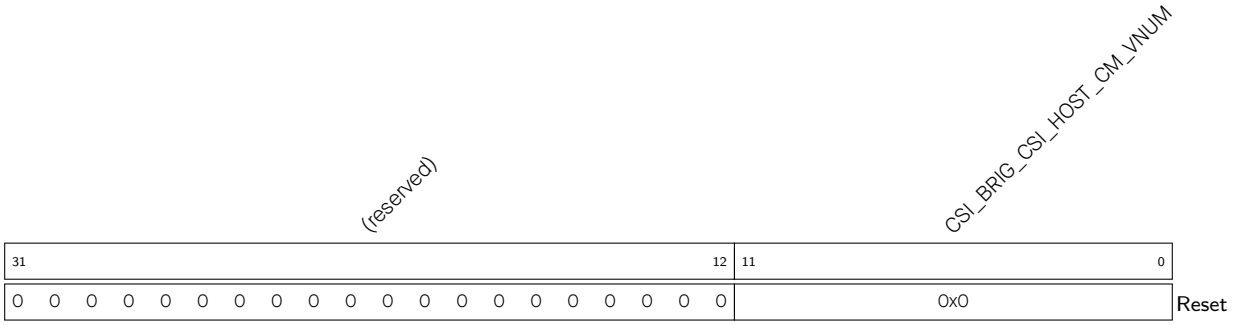
CSI_BRIG_CSI_HOST_CM_8BIT_SWAP 配置是否使能高低 8-bit 翻转。

0：关闭

1：使能

(R/W)

Register 38.159. CSI_BRIG_HOST_SIZE_CTRL_REG (0x004C)



CSI_BRIG_CSI_HOST_CM_HNUM 配置 MIPI CSI 输入图像接口的水平方向尺寸（以 32-bit 数据为单位）。其取值为 line_pix_num*bits_per_pix/32 - 1，仅在 YUV422 转换为 YUV420 时有效。(R/W)

第 39 章

像素处理加速器 (PPA)

39.1 概述

ESP32-P4 包括一个像素处理加速器 (PPA)。该模块为 2D 图像显示设计，用于对一些图像算法进行硬件加速，实现图像的旋转、缩放、镜像、叠加功能。

PPA 主要包括两大功能模块：旋转 - 缩放 - 镜像 (SRM) 和图层叠加 (BLEND)。SRM 模块实现图像旋转、缩放、XY 方向镜像功能，BLEND 模块实现根据 Alpha 通道（即透明度）对两个相同尺寸的图层进行叠加的功能。

PPA 按照像素块进行图像处理。像素块需要通过 2D-DMA 从内存里获取，PPA 与 2D-DMA 的连接关系如图 39.1-1 所示。

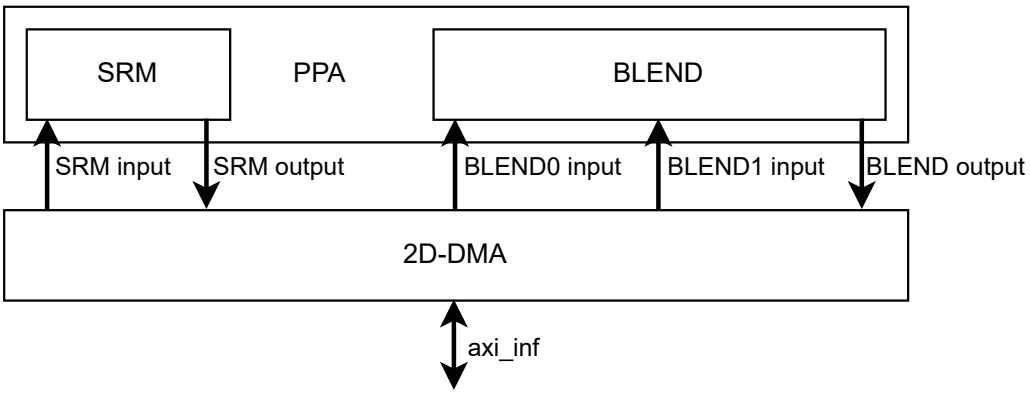


图 39.1-1. PPA 2D-DMA 连接关系

39.2 术语

为更好描述 PPA 功能，本章定义了以下术语：

图像 系统内存中存储的一幅完整图像
图像块 特指按一定尺寸从图像中截取的一块，最大尺寸等同于图像。即 2D-DMA 2D-MODO 下的 hb、vb 大小
像素块 特指从图像块中截取的一块，SRM 以像素块为单位读写数据

39.3 主要特性

- SRM 支持图像块旋转、缩放、镜像：
  - 输入格式支持 ARGB8888、RGB888、RGB565、YUV422、YUV420、GRAY
  - 输出格式支持 ARGB8888、RGB888、RGB565、YUV422、YUV420、GRAY

  - 逆时针旋转角度支持 0°、90°、180°、270°
  - 水平、垂直方向独立缩放支持 8 位整数及 4 位小数
  - 水平、垂直方向镜像
- BLEND 支持两个相同尺寸的图层叠加以及输出特定像素的填充图像：
  - 前景输入格式支持 ARGB8888、RGB888、RGB565、L4、L8、A4、A8
  - 背景输入格式支持 ARGB8888、RGB888、RGB565、YUV422、YUV420、GRAY、L4、L8
  - 输出格式支持 ARGB8888、RGB888、RGB565、YUV422、YUV420、GRAY
  - 基于 Alpha 通道的图层叠加，若图层没有 Alpha 通道，可通过寄存器配置提供
  - 前景和背景支持通过设置 color-key 范围实现特殊颜色抠图

39.4 结构概览

PPA 结构如图 39.4-1 所示，包括两个独立的功能模块：SRM 和 BLEND。

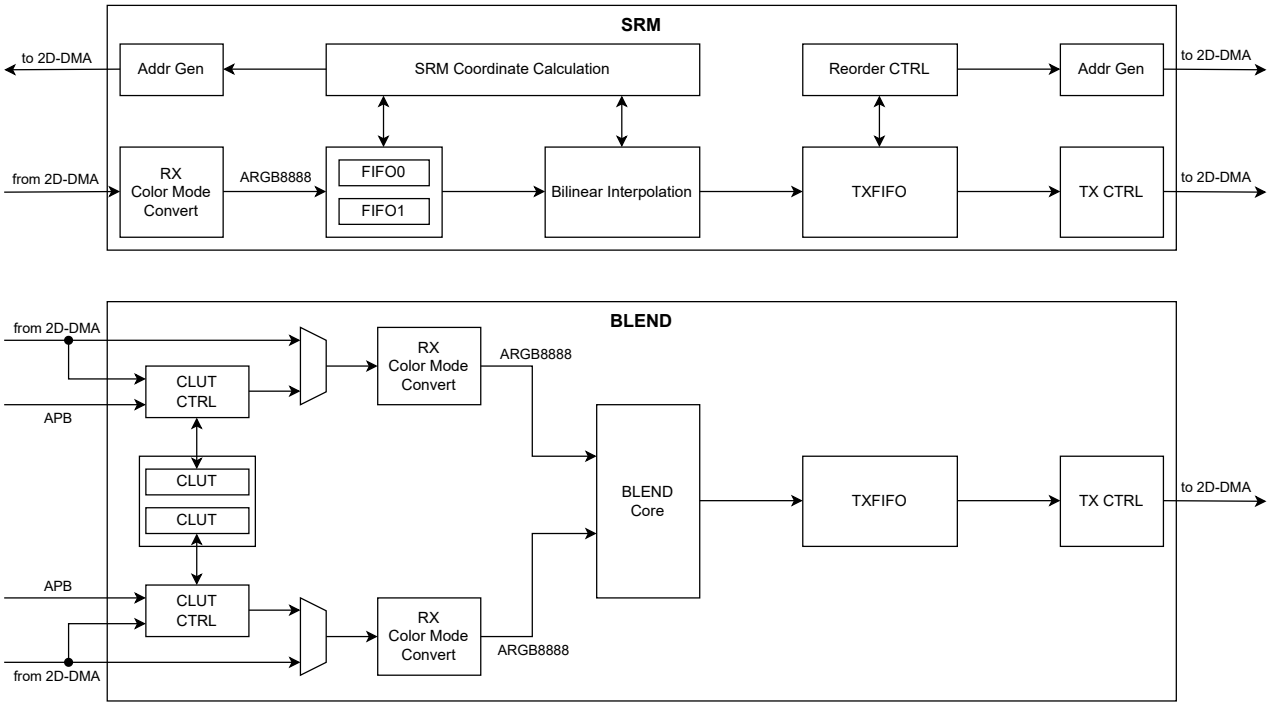


图 39.4-1. PPA 系统架构

对于 SRM，SRM Coordinate Calculation 对输出像素块坐标进行扫描，计算每个输出像素块所对应的输入像素块坐标。Addr Gen 再通过输入像素块坐标计算地址，发送给 2D-DMA。之后，PPA 从 2D-DMA 接受像素块数据，根据寄存器配置转成 ARGB8888 格式，并存入 RX FIFO（由两块 SRAM 组成的乒乓 FIFO，每一块都能完整存下整个像素块）。当一个完整像素块缓存完成后，Bilinear Interpolation 从 RX FIFO 中读取输入像素，经过插值运算得到输出像素并输出到 TX FIFO。Reorder CTRL 再根据寄存器配置，计算输出像素块坐标以及地址发送给 2D-DMA，同时按照输出像素块坐标从 TX FIFO 取出对应像素发送给 2D-DMA，由 2D-DMA 存到对应地址。

对于 BLEND，PPA 也会从 2D-DMA 接受数据，并根据输入数据的类型再转换为 ARGB8888 格式。转换后的像

素会直接进入算法模块 BLEND Core，完成后再进入 TX FIFO 缓存并通过 2D-DMA 输出到系统内存。与 SRM 不同的是，BLEND 不需要自己计算地址，完全由 2D-DMA 通过链表控制。

39.5 功能描述

39.5.1 PPA 颜色空间

PPA 支持真彩色与伪彩色。真彩色是指在组成一幅彩色图像的每个像素中，存在 R、G、B 三个基色分量，每个分量直接决定显示设备的基色强度。PPA 支持的真彩色格式包括 ARGB8888、RGB888、RGB565。

伪彩色是指一幅彩色图像中每个像素的颜色不是由基色分量数值直接决定，而是把像素值当作彩色查找表 (Color Look-Up Table, CLUT) 的入口，去查找一个像素对应的 A、R、G、B 值。PPA 支持入口地址 8 位或 4 位位宽的 CLUT。

除此之外，PPA 的 SRM 操作还支持 YUV422、YUV420、GRAY 数据格式。表 39.5-1 所示为支持颜色格式在内存中的存储顺序。其中，输入的 YUV422 存储格式可通过 SRM 或 BLEND 的 YUV422_BYTE_ORDER 寄存器配置，输出固定按照 YUV422_BYTE_ORDER = 1 存储。

表 39.5-1. 像素存储顺序

颜色格式	起始地址 + 3	起始地址 + 2	起始地址 + 1	起始地址 + 0
ARGB8888	**A0**[7:0]	**R0**[7:0]	**G0**[7:0]	**B0**[7:0]
RGB888	**B1**[7:0]	**R0**[7:0]	**G0**[7:0]	**B0**[7:0]
	G2[7:0]	**B2**[7:0]	**R1**[7:0]	**G1**[7:0]
	R3[7:0]	**G3**[7:0]	**B3**[7:0]	**R2**[7:0]
RGB565	**R1**[4:0]**G1**[5:3]	**G1**[2:0]**B1**[4:0]	**R0**[4:0]**G0**[5:3]	**G0**[2:0]**B0**[4:0]
L8	**L3**[7:0]	**L2**[7:0]	**L1**[7:0]	**L0**[7:0]
L4	**L7**[3:0]**L6**[3:0]	**L5**[3:0]**L4**[3:0]	**L3**[3:0]**L2**[3:0]	**L1**[3:0]**L0**[3:0]
A8	**A3**[7:0]	**A2**[7:0]	**A1**[7:0]	**A0**[7:0]
A4	**A7**[3:0]**A6**[3:0]	**A5**[3:0]**A4**[3:0]	**A3**[3:0]**A2**[3:0]	**A1**[3:0]**A0**[3:0]
YUV422 YUV422_BYTE_ORDER=0	**Y1**[7:0]	**V1**[7:0]	**Y0**[7:0]	**U0**[7:0]
YUV422 YUV422_BYTE_ORDER=1	**Y1**[7:0]	**U1**[7:0]	**Y0**[7:0]	**V0**[7:0]
YUV422 YUV422_BYTE_ORDER=2	**v1**[7:0]	**y1**[7:0]	**U0**[7:0]	**Y0**[7:0]
YUV422 YUV422_BYTE_ORDER=3	**U1**[7:0]	**Y1**[7:0]	**V0**[7:0]	**Y0**[7:0]
YUV420 (1、3、5... 等奇数行)	**U2**[7:0]	**Y1**[7:0]	**Y0**[7:0]	**U0**[7:0]
	Y4[7:0]	**U4**[7:0]	**Y3**[7:0]	**Y2**[7:0]
	Y7[7:0]	**Y6**[7:0]	**U6**[7:0]	**Y5**[7:0]
YUV420 (2、4、6... 等偶数行)	**V2**[7:0]	**Y1**[7:0]	**Y0**[7:0]	**V0**[7:0]

见下页

表 39.5-1 – 接上页

颜色格式	起始地址 + 3	起始地址 + 2	起始地址 + 1	起始地址 + 0
	Y4[7:0]	V4[7:0]	Y3[7:0]	Y2[7:0]
	Y7[7:0]	Y6[7:0]	V6[7:0]	Y5[7:0]
GRAY	P3[7:0]	P2[7:0]	P1[7:0]	P0[7:0]

39.5.1.1 SRM 颜色空间

SRM 的输入输出均支持 ARGB8888、RGB888、RGB565、YUV422、YUV420、GRAY 格式，输入格式由 PPA_SRM_RX_CM 配置，输出格式由 PPA_SRM_TX_CM 配置。其中 SRM 输入还有一系列像素处理流程，如图 39.5-1 所示。

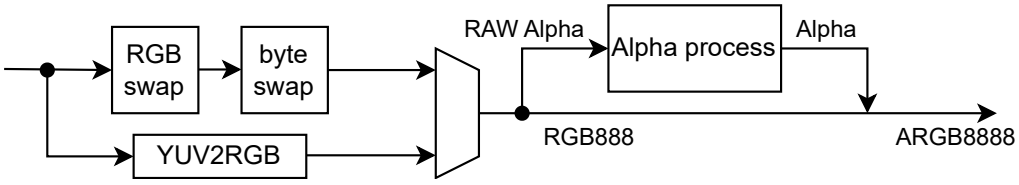


图 39.5-1. PPA SRM 输入像素处理

输入类型为 ARGB8888、RGB888、RGB565 时，支持进行 RGB 以及字节顺序翻转。具体配置分别如下：

- 向 PPA_SRM_RX_RGB_SWAP_EN 写 1 将 RGB 翻转为 BGR，将 ARGB 翻转为 BGRA。
- 向 PPA_SRM_RX_BYTE_SWAP_EN 写 1 分别将 0、1 字节和 2、3 字节翻转（即 {0，1，2，3} 翻转为 {1，0，3，2}）。由于 RGB888 一个像素只有 3 字节，因此不会受字节翻转寄存器配置影响。

SRM 内部均使用 ARGB8888 进行处理。当输入不带 Alpha 通道时，图像的 Alpha 通道值默认为 255。Alpha 通道还可以通过寄存器 PPA_SRM_FIX_ALPHA_REG 进一步配置，如图 39.5-2 所示。具体的配置效果可参考表 39.5-2。

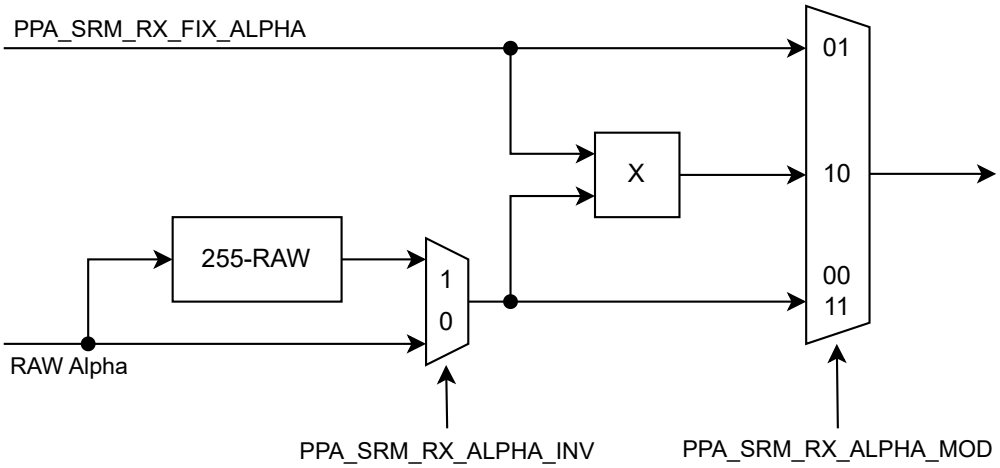


图 39.5-2. PPA SRM Alpha 配置逻辑

表 39.5-2. PPA SRM Alpha 配置效果

PPA_SRM_RX_ALPHA_INV	PPA_SRM_RX_ALPHA_MOD	输出结果
0	00/11	输出原始 Alpha 值
	01	输出 PPA_SRM_RX_FIX_ALPHA 配置的值
	10	输出原始 Alpha 值与 PPA_SRM_RX_FIX_ALPHA 相乘后的高 8 位
1	00/11	输出 255 减去原始 Alpha 值后得到的值
	01	输出 PPA_SRM_RX_FIX_ALPHA 配置值
	10	输出 255 减去原始 Alpha 值后再与 PPA_SRM_RX_FIX_ALPHA 相乘后的高 8 位

需注意，当输入或输出类型为 YUV 时，还需进一步通过寄存器配置 YUV、RGB 颜色转换。

当输入格式为 YUV420 时，需通过寄存器域 PPA_YUV_RX_RANGE 将输入 YUV 数据范围配置为 full-range 或 limit-range，并通过寄存器域 PPA_YUV2RGB_PROTOCAL 将 YUV 转为 RGB 使用的协议配置为 BT601 或 BT709。转换公式如下：

- YUV_{full} 转为 YUV_{limit}
  - $Y_{limit} = \frac{220}{256} Y_{full} + 16$
  - $U_{limit} = \frac{225}{256} U_{full} + 16$
  - $V_{limit} = \frac{225}{256} V_{full} + 16$
- YUV_{limit} 按照 BT601 转为 RGB
  - $R = \frac{298}{256} Y_{limit} + \frac{409}{256} V_{limit} - \frac{56906}{256}$
  - $G = \frac{298}{256} Y_{limit} - \frac{100}{256} U_{limit} - \frac{208}{256} V_{limit} + \frac{34707}{256}$
  - $B = \frac{298}{256} Y_{limit} + \frac{516}{256} U_{limit} - \frac{70836}{256}$
- YUV_{limit} 按照 BT709 转为 RGB
  - $R = \frac{298}{256} Y_{limit} + \frac{459}{256} V_{limit} - \frac{63367}{256}$
  - $G = \frac{298}{256} Y_{limit} - \frac{55}{256} U_{limit} - \frac{136}{256} V_{limit} + \frac{19681}{256}$
  - $B = \frac{298}{256} Y_{limit} + \frac{541}{256} U_{limit} - \frac{73918}{256}$

当输出格式选为 YUV420 时，需通过寄存器域 PPA_YUV_TX_RANGE 将输出 YUV 数据范围配置为 full-range 或 limit-range，通过寄存器域 PPA_RGB2YUV_PROTOCAL 将 RGB 转为 YUV 使用的协议配置为 BT601 或 BT709。转换公式如下：

- YUV_{limit} 转为 YUV_{full}
  - $Y_{full} = \frac{298}{256} Y_{limit} + \frac{4768}{256}$
  - $U_{full} = \frac{291}{256} U_{limit} + \frac{4550}{256}$
  - $V_{full} = \frac{291}{256} V_{limit} + \frac{4550}{256}$
- RGB 按照 BT601 转为 YUV_{limit}
  - $Y_{limit} = \frac{4096}{256} + \frac{66}{256} R + \frac{129}{256} G + \frac{25}{256} B$

  - $U_{limit} = \frac{32768}{256} - \frac{38}{256}R - \frac{74}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{94}{256}G - \frac{18}{256}B$
- RGB 按照 BT709 转为 YUV_{limit}
  - $Y_{limit} = \frac{4096}{256} + \frac{47}{256}R + \frac{157}{256}G + \frac{16}{256}B$
  - $U_{limit} = \frac{32768}{256} - \frac{26}{256}R - \frac{86}{256}G + \frac{112}{256}B$
  - $V_{limit} = \frac{32768}{256} + \frac{112}{256}R - \frac{102}{256}G - \frac{10}{256}B$

39.5.1.2 BLEND 颜色空间

BLEND 前景输入支持 ARGB8888、RGB888、RGB565、L4、L8、A4、A8，背景输入支持 ARGB8888、RGB888、RGB565、YUV422、YUV420、GRAY、L4、L8，可通过寄存器域 PPA_BLEND0/1_RX_CM 分别配置前景、背景图层的输入颜色格式。其中，A4、A8 只支持前景图层，且当输入类型为 L4、A4 时，图像块的尺寸 hb 以及在图像中的偏移 x 必须为偶数。BLEND 输出支持 ARGB8888、RGB888、RGB565，可通过寄存器域 PPA_BLEND_TX_CM 进行配置。

BLEND 的 RGB 翻转、字节翻转、Alpha 通道配置和 SRM 输入一致。另外，当前景图层数据类型为 A4、A8 时，输入数据为 Alpha 通道数值。此时若 PPA_BLEND_BYPASS 配置为 1，则前景图层输入的 Alpha 值会替换掉背景图层的 Alpha 通道并输出；配置为 0，则为正常的 BLEND 操作，前景图层的 RGB 值由 PPA_BLEND_RGB_REG 指定。

当输入数据类型为 L4、L8 时，需要先对 CLUT 进行初始化。BLEND 有两个深度为 256、宽度为 32 的 CLUT，初始化有两种方式，分别为 FIFO 模式、MEM 模式。

- 向 PPA_APB_FIFO_MASK 写 0 进入 FIFO 模式。
  FIFO 模式下，通过读写 PPA_RDWR_WORD_BLENDx_CLUT 实现访问 CLUT。其中 0 对应背景通道 CLUT，1 对应前景通道 CLUT。
  每次读/写该寄存器，会自动将对应的读/写地址加 1，可通过向 PPA_BLENDx_CLUT_MEM_RST 写 1 再写 0 复位读写地址，可通过向PPA_BLENDx_CLUT_MEM_RDADDR_RST 写 1 再写 0 复位读地址。
- 向 PPA_APB_FIFO_MASK 写 1 进入 MEM 模式。
  MEM 模式下，可直接通过地址访问 CLUT。在 PPA 的总线地址范围内，CLUT 地址如表 39.5-3 所示。

表 39.5-3. BLEND CLUT 地址

addr[11:10]	addr[9:2]	addr[1:0]
01：选择 BLEND0 CLUT 10：选择 BLEND1 CLUT	CLUT 实际地址，0 ~ 255	00

39.5.2 PPA 2D-DMA 链表配置

PPA 通过 2D-DMA 与系统内存进行数据交互。对于 SRM 或 BLEND，都需要一次性配置好所需操作图像块的 2D-DMA 链表信息。2D-DMA 链表如图 39.5-3，详细可参考 8 *2D-DMA 控制器 (2D-DMA)*：

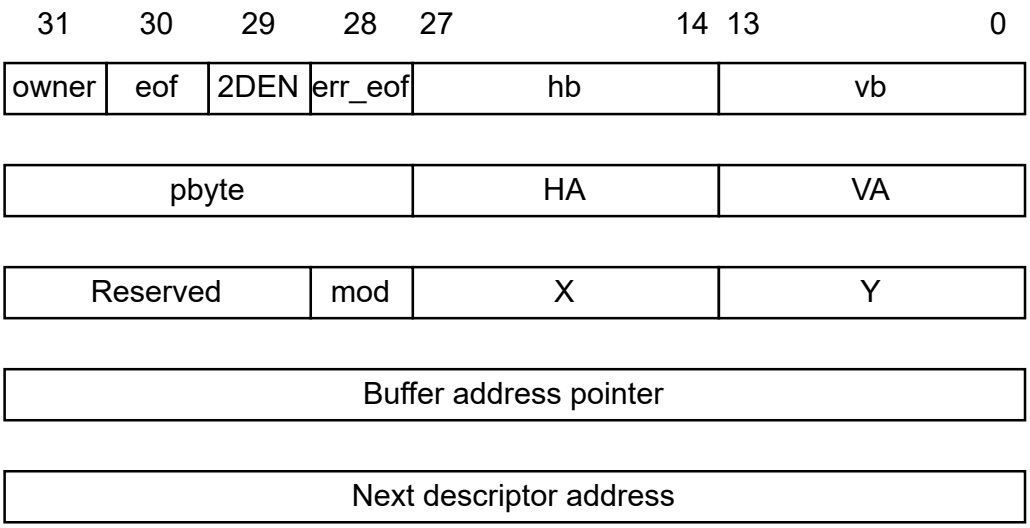


图 39.5-3. 2D-DMA 链表

对于 PPA，对应的链表参数如表 39.5-4 所示。

表 39.5-4. PPA 链表配置

域	配置
owner	1
eof	1
2DEN	1
err_eof	0
hb	图像块水平方向尺寸 若输入或输出数据为 YUV420，则必须为偶数
vb	图像块垂直方向尺寸 若输入或输出数据为 YUV420，则必须为偶数
pbyte	ARGB8888：5（4 字节/像素） RGB888：4（3 字节/像素） RGB565：3（2 字节/像素） YUV420：2（1.5 字节/像素） A8/L8：1（1 字节/像素） A4/L4：0（0.5 字节/像素）
HA	整幅图像水平方向尺寸 若输入或输出数据为 YUV420，则必须为偶数
VA	整幅图像垂直方向尺寸 若输入或输出数据为 YUV420，则必须为偶数
mod	0
X	图像块在图像中的水平方向偏移（偏移数从 0 开始） 若输入或输出数据为 YUV420，则必须为偶数
Y	图像块在图像中的垂直方向偏移（偏移数从 0 开始） 若输入或输出数据为 YUV420，则必须为偶数

见下页

表 39.5-4 – 接上页

域	配置
Buffer address pointer	图像起始地址
Next descriptor address	N/A

39.5.2.1 SRM 特殊配置

当配置好输入输出图像块的 2D-DMA 链表后，SRM 需要以像素块为单位，从输入图像块存储空间中选择出所需像素块，处理完成后再将其存储到输出图像块对应存储空间。这个过程中，每一个像素块的起始地址均需要受 SRM 的控制。为实现该功能，需要向 2D-DMA 对应通道的 DMA2D_IN/OUT_DSCR_PORT_EN_CHx 寄存器域写 1 以使能描述符接口（DSCR-PORT 模式），从而允许 PPA 实时控制下一个像素块的起始地址。

在数据输入方向，SRM 必须每次均能接收到一个完整的像素块。当像素块选择到达图像块边界时，需要 2D-DMA 将像素块超出边界部分自动补齐，可通过向 2D-DMA 对应通道的寄存器域 DMA2D_OUT_DSCR_PORT_BLK_V/H_CHx 配置为像素块大小实现。配置值根据图像格式而不同，具体如下表 39.5-5。

表 39.5-5. 自动补全块大小配置

PPA_SRM_BK_SIZE_SEL	像素格式	DMA2D_OUT_DSCR_PORT_BLK_H_CHx	DMA2D_OUT_DSCR_PORT_BLK_V_CHx
0	ARGB8888 RGB888 RGB565 GRAY	34	34
0	YUV422	36	34
0	YUV420	36	36
1	ARGB8888 RGB888 RGB565 GRAY	18	18
1	YUV422	20	18
1	YUV420	20	20

说明：
为了最优化存储访问性能，推荐使用默认配置 PPA_SRM_BK_SIZE_SEL = 0。

39.5.3 旋转 - 缩放 - 镜像 (SRM)

39.5.3.1 基本功能

SRM 实现缩放、旋转、镜像三个功能，先将图像基于坐标原点进行缩放，再基于中心点（X_{middle}，Y_{middle}）进行旋转，最后基于中心点（X_{target}，Y_{target}）进行水平、垂直方向镜像，如图 39.5-4 所示。缩放、旋转、镜像参数均通过寄存器进行配置：

- PPA_SRM_SCAL_X_INT 水平方向缩放系数，整数部分
- PPA_SRM_SCAL_X_FRAG 水平方向缩放系数，小数部分
- PPA_SRM_SCAL_Y_INT 垂直方向缩放系数，整数部分
- PPA_SRM_SCAL_Y_FRAG 垂直方向缩放系数，小数部分
- PPA_SRM_ROTATE_ANGLE 逆时针旋转角度
- PPA_SRM_MIRROR_X 水平方向镜像使能
- PPA_SRM_MIRROR_Y 垂直方向镜像使能

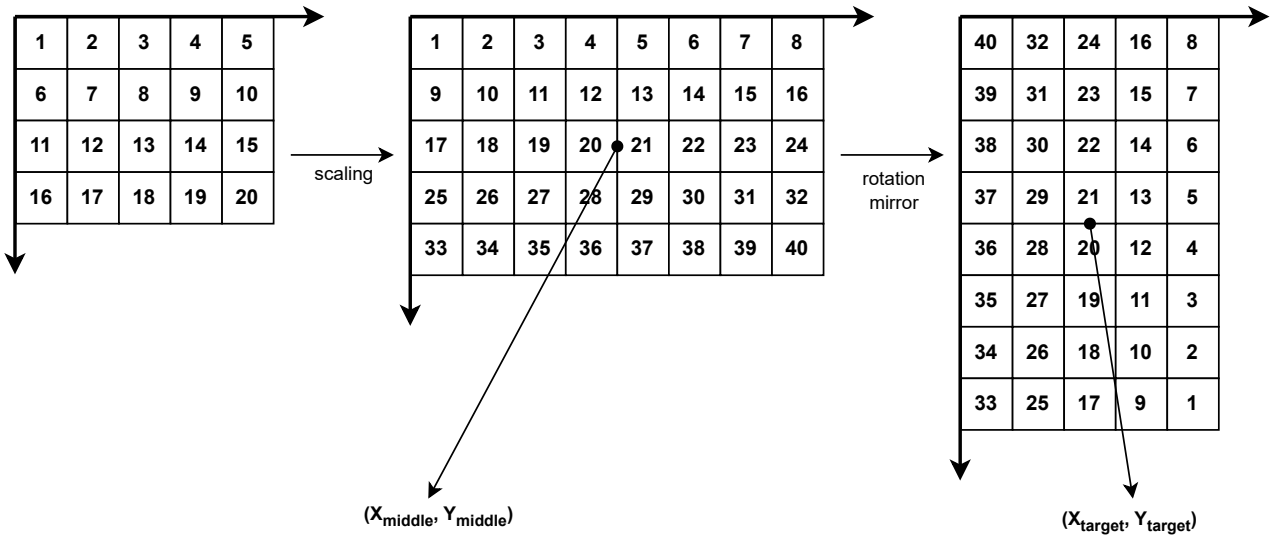


图 39.5-4. SRM 步骤示例

39.5.3.2 像素块重排

当输出像素块较小时，如果每一个像素块都发起一次 2D-DMA 传输，会导致总线带宽利用率较低。为了提高总线带宽利用率，SRM 能对输出像素块进行缓存拼接，拼接为一个更大的像素块后再一次性输出。该功能默认打开，可以通过将 PPA_SRM_MACRO_BK_RO_BYPASS 置 1 关闭。

像素块重排功能开启后，当输出像素块尺寸满足一定条件，会自动进行重排拼接，数量如表 39.5-6 所示。

- 当输出像素块为 ARGB8888/RGB888 格式时，若输出像素块垂直高度小于等于 32，且水平宽度小于等于 16 时，像素块重排功能会自动开启。
- 当输出像素块为 RGB565 格式时，若输出像素块垂直高度小于等于 32，且水平宽度小于等于 32 时，像素块重排功能会自动开启。
- 当输出像素块为 YUV420 格式时，若输出像素块垂直高度小于等于 32，且水平宽度小于等于 32 时，像素块重排功能会自动开启。

表 39.5-6. SRM 输出像素块重排数量

ARGB8888/RGB888		RGB565		YUV420	
水平尺寸	重排个数	水平尺寸	重排个数	水平尺寸（偶数）	重排个数
1	32	1	64	2	32
2	16	2	32	4	16
3	10	3	20	6	10
4	8	4	16	8	8
5	6	5	12	10	6
6	5	6	10	12	5
7	4	7	9	14	4
8	4	8	8	16	4
9	3	9	7	18	3
10	3	10	6	20-32	2
11	2	11	5		
12	2	12	5		
13	2	13	4		
14	2	14	4		
15	2	15	4		
16	2	16	4		
		17	3		
		18	3		
		19-32	2		

39.5.4 图层叠加 (BLEND)

39.5.4.1 基本功能

BLEND 把两幅同尺寸图像块通过 Alpha 通道进行叠加，再输出。叠加公式如下，其中 Ab 为背景图层 Alpha 通道，Af 为前景图层 Alpha 通道，Cb 对应背景图层 R、G、B 分量，Cf 对应前景图层 R、G、B 分量：

- $Aout = Ab + Af - Ab * Af$
- $Cout = (Cb * Ab * (1 - Af) + Cf * Af)/(Ab + Af - Ab * Af)$

同时，BLEND 支持基于像素颜色的 color-key 抠图，通过 PPA_CK_FG/BG_HIGH/LOW_REG 配置颜色范围，可将图像在逻辑上分为 4 个不同选区，并对每个选区执行不同的操作：

- 当前景像素在 color-key 范围内，背景像素在 color-key 范围外，此时输出背景像素。
- 当背景像素在 color-key 范围内，前景像素在 color-key 范围外，此时若 PPA_COLORKEY_FG_BG_REVERSE 为 0，则输出背景像素；为 1，则输出前景像素。
- 当背景、前景像素均在 color-key 范围内，此时输出 PPA_COLORKEY_DEFAULT_R/G/B 设置的颜色。
- 当背景、前景像素均不在 color-key 范围内，此时按照正常 Alpha Blending 流程输出。

当 PPA_BLEND_BYPASS 置为 1 后，BLEND 会直接输出背景图层的数据；若此时前景输入数据类型为 A4/A8，则前景输入数据会替换背景输入数据的 Alpha 通道再输出。

39.5.4.2 填充图像输出

BLEND 还支持填充图像输出。该功能开启后，BLEND 只有输出端会工作，连续不断输出填充图像。填充图像的尺寸以及数据可分别通过 PPA_BLEND_TX_SIZE_REG 和 PPA_BLEND_TX_FIX_PIXEL 设置。

39.6 中断

ESP32-P4 PPA 包含一个中断信号，该中断会输出到 13中断矩阵。

- PPA_INTR

PPA_INTR 中断由内部多个中断源产生，包括：

- PPA_SRM_PARAM_CFG_ERR_INT：当 SRM 配置了错误参数触发。用户可通过 PPA_SRM_PARAM_ERR_ST_REG 获取具体错误类型，如表 39.6-1 所示。
- PPA_BLEND_PARAM_CFG_ERR_INT：当 BLEND 配置了错误参数触发。用户可通过 PPA_BLEND_ST_REG 获取具体错误类型，如表 39.6-2 所示。
- PPA_SRM_EOF_INT：当 SRM 完成一次图像块处理触发。
- PPA_BLEND_EOF_INT：当 BLEND 完成一次图像块处理触发。

表 39.6-1. SRM 参数错误类型

错误类型	详情
PPA_TX_DSCR_VB_ERR_ST	SRM 输出图像块垂直尺寸加上输出链表配置的垂直偏移 Y 超出输出链表配置的整幅图像垂直尺寸 VA
PPA_TX_DSCR_HB_ERR_ST	SRM 输出图像块水平尺寸加上输出链表配置的水平偏移 X 超出输出链表配置的整幅图像水平尺寸 HA
PPA_Y_RX_SCAL_EQUAL_0_ERR_ST	SRM 输入图像块垂直方向缩放系数为 0
PPA_RX_DSCR_VB_ERR_ST	SRM 输入链表配置的垂直方向图像块尺寸 VB 加上垂直方向偏移 Y 大于整幅图像垂直尺寸 VA
PPA_YDST_LEN_TOO_SAMLL_ERR_ST	SRM 缩放后的图像块垂直方向尺寸为 0。例如，图像块垂直尺寸为 14，垂直缩放系数为 1/16，不进行旋转，此时缩放后的图像块垂直尺寸为 0
PPA_YDST_LEN_TOO_LARGE_ERR_ST	SRM 缩放后的图像块垂直方向尺寸大于 8191
PPA_X_RX_SCAL_EQUAL_0_ERR_ST	SRM 输入图像块水平方向缩放系数为 0
PPA_RX_DSCR_HB_ERR_ST	SRM 输入链表配置的水平方向图像块尺寸 HB 加上水平方向偏移 X 大于整幅图像水平尺寸 HA
PPA_XDST_LEN_TOO_SAMLL_ERR_ST	SRM 缩放后的图像块水平方向尺寸为 0
PPA_XDST_LEN_TOO_LARGE_ERR_ST	SRM 缩放后的图像块水平方向尺寸大于 8191
PPA_X_YUV420_RX_SCALE_ERR_ST	当 SRM 输入为 YUV420 时，输入链表的 HA/hb/X 参数为奇数
PPA_Y_YUV420_RX_SCALE_ERR_ST	当 SRM 输入为 YUV420 时，输入链表的 VA/vb/Y 参数为奇数
PPA_X_YUV420_TX_SCALE_ERR_ST	当 SRM 输出为 YUV420 时，输出链表的 HA/hb/X 参数为奇数
PPA_Y_YUV420_TX_SCALE_ERR_ST	当 SRM 输出为 YUV420 时，输出链表的 VA/vb/Y 参数为奇数

表 39.6-2. BLEND 参数错误类型

错误类型	详情
PPA_BLEND_SIZE_DIFF_ST	BLEND 前景和背景尺寸不一致
PPA_BLEND_YUV_X_SCALE_ERR_ST	BLEND 背景图像格式为 YUV422 或 YUV420 时，水平尺寸为奇数
PPA_BLEND_YUV_Y_SCALE_ERR_ST	BLEND 背景图像格式为 YUV420 时，垂直尺寸为奇数

39.7 配置流程

39.7.1 PPA 时钟复位配置流程

1. 打开 PPA、2D-DMA 模块的时钟：
   - 向 HP_SYS_CLKRST_PPA_SYS_CLK_EN 写 1 打开 PPA 时钟
   - 向 HP_SYS_CLKRST_DMA2D_SYS_CLK_EN 写 1 打开 2D-DMA 时钟
2. 复位 PPA、2D-DMA 模块：
   - 向 HP_SYS_CLKRST_RST_EN_PPA 写 1 再写 0 复位 PPA
   - 向 HP_SYS_CLKRST_RST_EN_DMA2D 写 1 再写 0 复位 2D-DMA

39.7.2 SRM 配置流程

1. 参考 39.7.1 打开 PPA、2D-DMA 时钟，复位 PPA、2D-DMA。
2. 配置 2D-DMA 输入输出链表，具体参数可参考表 39.5-4。
3. 选择 2D-DMA 输入输出通道：
   - 向 DMA2D_OUT_PERI_SEL_CHx 写 0x1，将一条 2D-DMA 输出通道分配给 PPA SRM 输入。
   - 向 DMA2D_IN_PERI_SEL_CHx 写 0x1，将一条 2D-DMA 输入通道分配给 PPA SRM 输出。
4. 对 PPA 使用的 2D-DMA 通道使能描述符接口：
   - 向 DMA2D_OUT_DSCR_PORT_EN_CHx 写 1 使能描述符接口。
   - 向 DMA2D_IN_DSCR_PORT_EN_CHx 写 1 使能描述符接口。
5. 配置 2D-DMA 描述符接口补全参数：
   - 配置 DMA2D_OUT_DSCR_PORT_BLK_H_CHx 水平补全参数，详见表 39.5-5。
   - 配置 DMA2D_OUT_DSCR_PORT_BLK_V_CHx 垂直补全参数，详见表 39.5-5。
6. 开启 2D-DMA 输入输出通道：
   - 向 DMA2D_OUTLINK_START_CHx 写 1 开启 2D-DMA 输出通道。
   - 向 DMA2D_INLINK_START_CHx 写 1 开启 2D-DMA 输入通道。
7. 配置 SRM 参数：
   - 配置 PPA_SRM_BK_SIZE_SEL 设置处理图像块尺寸。
   - 配置 PPA_SRM_RX_CM 设置 SRM 输入数据类型。

- 配置 PPA_SRM_TX_CM 设置 SRM 输出数据类型。
- 配置 PPA_SRM_SCAL_X_INT 设置水平方向缩放系数的整数部分。
- 配置 PPA_SRM_SCAL_X_FRAG 设置水平方向缩放系数的小数部分。
- 配置 PPA_SRM_SCAL_Y_INT 设置垂直方向缩放系数，整数部分。
- 配置 PPA_SRM_SCAL_Y_FRAG 设置垂直方向缩放系数，小数部分。
- 配置 PPA_SRM_ROTATE_ANGLE 设置逆时针旋转角度。
- 配置 PPA_SRM_MIRROR_X 设置水平方向镜像使能。
- 配置 PPA_SRM_MIRROR_Y 设置垂直方向镜像使能。

8. 向 PPA_SCAL_ROTATE_START 写 1 开启 SRM 工作。

39.7.3 BLEND CLUT 配置流程

1. FIFO 模式：
   - 向 PPA_APB_FIFO_MASK 写 0 进入 FIFO 模式
   - 写 PPA_RDWR_WORD_BLEND0_CLUT 初始化背景图层 CLUT
   - 写 PPA_RDWR_WORD_BLEND1_CLUT 初始化前景图层 CLUT
2. MEM 模式：
   - 向 PPA_APB_FIFO_MASK 写 1 进入 MEM 模式。
   - 直接向对应地址写数据初始化 CLUT，可参考表 39.5-3。

39.7.4 BLEND 配置流程

1. 参考 39.7.1 打开 PPA、2D-DMA 时钟，复位 PPA、2D-DMA。
2. 配置 2D-DMA 输入输出链表，具体参数可参考表 39.5-4。
3. 选择 2D-DMA 输入输出通道：
   - 向 DMA2D_OUT_PERI_SEL_CHx 写 0x2，将一条 2D-DMA 输出通道分配给 PPA BLEND 背景层输入。
   - 向 DMA2D_OUT_PERI_SEL_CHx 写 0x3，将一条 2D-DMA 输出通道分配给 PPA BLEND 前景层输入。
   - 向 DMA2D_IN_PERI_SEL_CHx 写 0x2，将一条 2D-DMA 输入通道分配给 PPA BLEND 输出。
4. 开启 2D-DMA 输入输出通道。其中，x 为分配给 BLEND 前景、背景的两条通道：
   - 向 DMA2D_OUTLINK_START_CHx 写 1 开启 2D-DMA 输出通道
   - 向 DMA2D_INLINK_START_CHx 写 1 开启 2D-DMA 输入通道
5. 配置 BLEND 参数：
   - 配置 PPA_BLEND0_RX_CM 设置 BLEND 背景图层输入数据类型。若为 L4/L8，需要先按照 39.7.3 初始化 CLUT
   - 配置 PPA_BLEND1_RX_CM 设置 BLEND 前景图层输入数据类型。若为 L4/L8，需要先按照 39.7.3 初始化 CLUT

- 配置 PPA_BLEND_TX_CM 设置 BLEND 输出数据类型
- 配置 PPA_CK_BG_HIGH_REG、PPA_CK_BG_LOW_REG 分别配置背景图层 color-key 上下限
- 配置 PPA_CK_FG_HIGH_REG、PPA_CK_FG_LOW_REG 分别配置前景图层 color-key 上下限
- 配置 PPA_CK_DEFAULT_REG 分别配置 color-key 默认值以及行为
- 向 PPA_BLEND_EN 写 1 使能 BLEND

6. 向 PPA_BLEND_TRANS_MODE_UPDATE 写 1 开启 BLEND 工作。

39.7.5 BLEND 图像填充配置流程

1. 参考 39.7.1 打开 PPA、2D-DMA 时钟，复位 PPA、2D-DMA。
2. 配置 2D-DMA 输入（PPA BLEND 输出）链表，具体参数可参考表 39.5-4。
3. 向 DMA2D_IN_PERI_SEL_CHx 写 0x2，将一条 2D-DMA 输入通道分配给 PPA BLEND 输出。
4. 向 DMA2D_INLINK_START_CHx 写 1 开启 2D-DMA 输入通道。
5. 配置 BLEND 参数：
   - 配置 PPA_BLEND_TX_CM 设置 BLEND 输出数据类型
   - 向 PPA_BLEND_FIX_PIXEL_FILL_EN 写 1 使能图像填充功能
   - 配置 PPA_BLEND_TX_FIX_PIXEL 设置填充图像像素值
   - 配置 PPA_BLEND_HB、PPA_BLEND_VB 设置填充图像尺寸
6. 向 PPA_BLEND_TRANS_MODE_UPDATE 写 1 开启 BLEND 工作。

39.7.6 错误处理流程

- 参考 39.7.1 复位整个 PPA，再重新配置
- 或者向 PPA_SCAL_ROTATE_RST、PPA_BLEND_RST 写 1 再写 0 分别复位 SRM、BLEND，再重新配置

说明：

- 本章节中所有 2D-DMA 配置只涉及一些关键步骤，详细流程可参考 8 *2D-DMA 控制器 (2D-DMA)*。
- 本章节中，DMA2D_OUT 方向对应 PPA 输入方向，DMA2D_IN 方向对应 PPA 输出方向。

39.8 寄存器列表

本小节的所有地址均为相对于像素处理加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问权限
配置寄存器			
PPA_BLEND0_CLUT_DATA_REG	BLEND 背景图层 CLUT SRAM 读写寄存器	0x0000	R/W
PPA_BLEND1_CLUT_DATA_REG	BLEND 前景图层 CLUT SRAM 读写寄存器	0x0004	R/W
PPA_CLUT_CONF_REG	BLEND CLUT 配置寄存器	0x000C	R/W
PPA_SRM_COLOR_MODE_REG	SRM 颜色空间配置寄存器	0x0020	R/W
PPA_BLEND_COLOR_MODE_REG	BLEND 颜色空间配置寄存器	0x0024	R/W
PPA_SRM_BYTE_ORDER_REG	SRM 字节顺序控制寄存器	0x0028	R/W
PPA_BLEND_BYTE_ORDER_REG	BLEND 字节顺序控制寄存器	0x002C	R/W
PPA_BLEND_TRANS_MODE_REG	BLEND 控制寄存器	0x0034	varies
PPA_SRM_FIX_ALPHA_REG	SRM 透明度通道配置寄存器	0x0038	R/W
PPA_BLEND_TX_SIZE_REG	BLEND 填充图像尺寸配置寄存器	0x003C	R/W
PPA_BLEND_FIX_ALPHA_REG	BLEND 透明度通道配置寄存器	0x0040	R/W
PPA_BLEND_RGB_REG	BLEND RGB 颜色配置寄存器	0x0048	R/W
PPA_BLEND_FIX_PIXEL_REG	BLEND 填充图像像素配置寄存器	0x004C	R/W
PPA_CK_FG_LOW_REG	BLEND 前景图层 color-key 下阈值配置寄存器	0x0050	R/W
PPA_CK_FG_HIGH_REG	BLEND 前景图层 color-key 上阈值配置寄存器	0x0054	R/W
PPA_CK_BG_LOW_REG	BLEND 背景图层 color-key 下阈值配置寄存器	0x0058	R/W
PPA_CK_BG_HIGH_REG	BLEND 背景图层 color-key 上阈值配置寄存器	0x005C	R/W
PPA_CK_DEFAULT_REG	BLEND color-key 默认值配置寄存器	0x0060	R/W
PPA_SRM_SCAL_ROTATE_REG	SRM 配置寄存器	0x0064	varies
PPA_SRM_MEM_PD_REG	SRM 存储器控制寄存器	0x0068	R/W
PPA_REG_CONF_REG	寄存器时钟门控控制	0x006C	R/W
PPA_RGB2GRAY_REG	RGB 转 GRAY 系数寄存器	0x0098	R/W
中断寄存器			
PPA_INT_RAW_REG	中断源状态寄存器	0x0010	R/WTC/SS
PPA_INT_ST_REG	可屏蔽中断状态寄存器	0x0014	RO
PPA_INT_ENA_REG	中断使能寄存器	0x0018	R/W
PPA_INT_CLR_REG	中断清除寄存器	0x001C	WT
状态寄存器			
PPA_CLUT_CNT_REG	BLEND CLUT 写计数器状态寄存器	0x0070	RO
PPA_BLEND_ST_REG	BLEND 状态寄存器	0x0074	RO
PPA_SRM_PARAM_ERR_ST_REG	SRM 配置错误状态寄存器	0x0078	RO
PPA_SRM_STATUS_REG	SRM 状态机状态寄存器	0x007C	RO
调试寄存器			

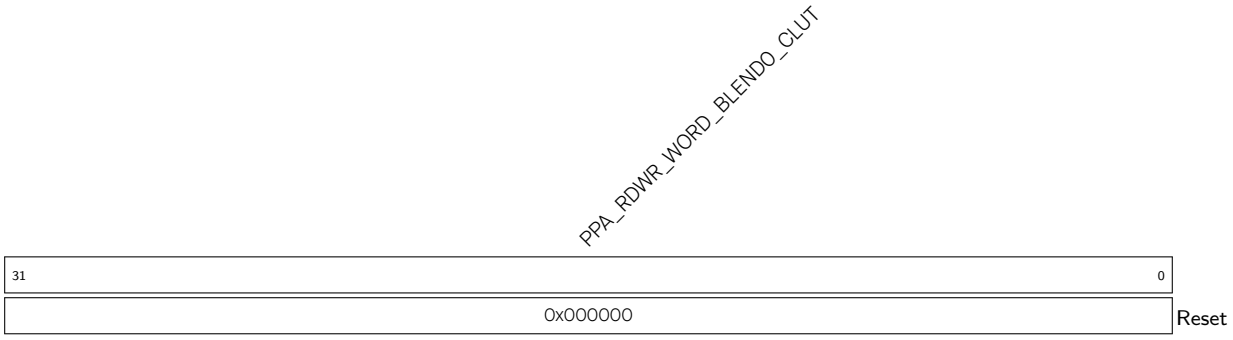
名称	描述	地址	访问权限
PPA_DEBUG_CTRL0_REG	调试寄存器	0x0090	R/W
PPA_DEBUG_CTRL1_REG	调试寄存器	0x0094	R/W
版本寄存器			
PPA_DATE_REG	PPA 版本寄存器	0x0100	R/W

39.9 寄存器

本小节的所有地址均为相对于像素处理加速器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

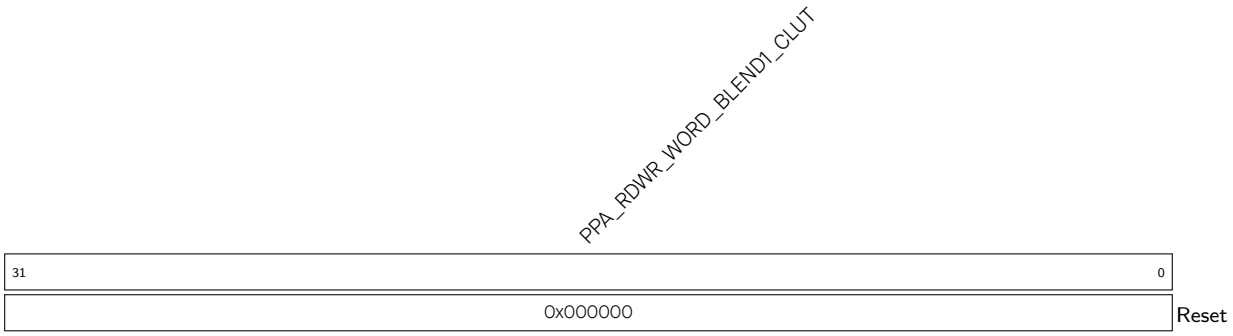
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 39.1. PPA_BLEND0_CLUT_DATA_REG (0x0000)



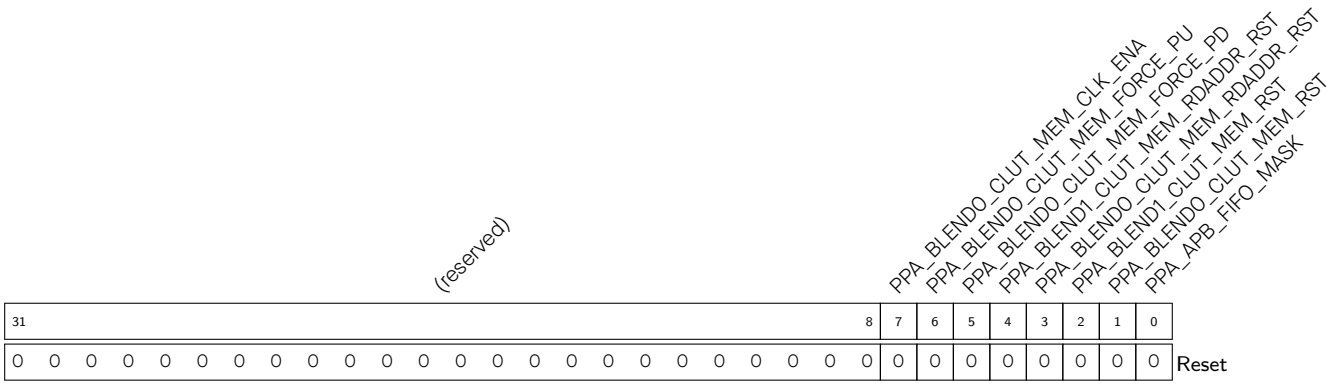
PPA_RDWR_WORD_BLEND0_CLUT FIFO 模式下 BLEND 背景图层 CLUT 访问寄存器。(R/W)

Register 39.2. PPA_BLEND1_CLUT_DATA_REG (0x0004)



PPA_RDWR_WORD_BLEND1_CLUT FIFO 模式下 BLEND 前景图层 CLUT 访问寄存器。(R/W)

Register 39.3. PPA_CLUT_CONF_REG (0x000C)



PPA_APB_FIFO_MASK 配置 BLEND CLUT 的访问方式。

1'b0: FIFO 模式访问 CLUT，具体可见 39.5.1.2。

1'b1: MEM 模式访问 CLUT，具体可见 39.5.1.2。

(R/W)

PPA_BLENDO_CLUT_MEM_RST 配置 BLENDO CLUT 逻辑的复位。

0：释放复位

1：复位

(R/W)

PPA_BLEND1_CLUT_MEM_RST 配置 BLEND1 CLUT 逻辑的复位。

0：释放复位

1：复位

(R/W)

PPA_BLENDO_CLUT_MEM_RDADDR_RST 配置 FIFO 模式下 BLENDO CLUT 的读地址复位。

0：释放复位

1：复位

(R/W)

PPA_BLEND1_CLUT_MEM_RDADDR_RST 配置 FIFO 模式下 BLEND1 CLUT 的读地址复位。

0：释放复位

1：复位

(R/W)

PPA_BLENDO_CLUT_MEM_FORCE_PD 配置 BLEND CLUT 强制掉电。

0：不强制掉电

1：强制掉电

(R/W)

PPA_BLENDO_CLUT_MEM_FORCE_PU 配置 BLEND CLUT 强制上电。

0：不强制上电

1：强制上电

(R/W)

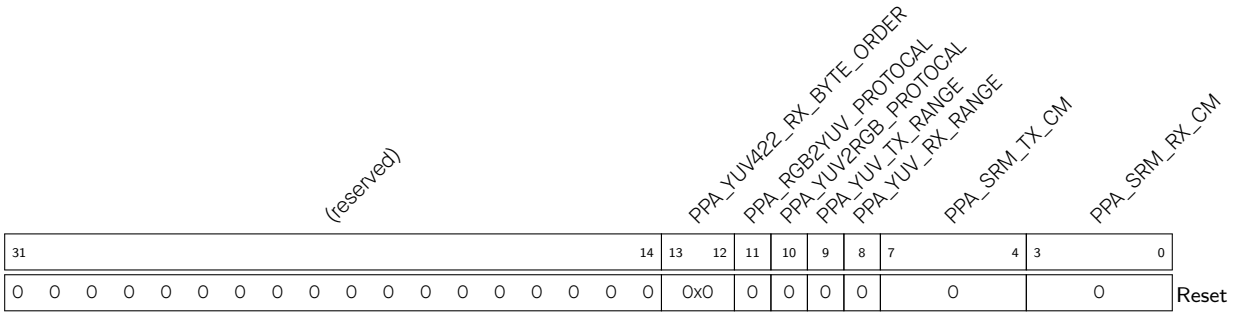
PPA_BLENDO_CLUT_MEM_CLK_ENA 配置 BLEND CLUT 时钟强制打开。

0：不强制打开

1：不强制打开

(R/W)

Register 39.4. PPA_SRM_COLOR_MODE_REG (0x0020)



PPA_SRM_RX_CM 配置 SRM 输入图像颜色格式。

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

其他：保留

(R/W)

PPA_SRM_TX_CM 配置 SRM 输出图像颜色格式。

0: ARGB8888

1: RGB888

2: RGB565

8: YUV420

其他：保留

(R/W)

PPA_YUV_RX_RANGE 配置 SRM 输入为 YUV 时的 YUV 范围。

0：限制范围

1：全范围

(R/W)

PPA_YUV_TX_RANGE 配置 SRM 输出为 YUV 时的 YUV 范围。

0：限制范围

1：全范围

(R/W)

PPA_YUV2RGB_PROTOCAL 配置 SRM 输入方向 YUV 转 RGB 使用协议。

0: BT601

1: BT709

(R/W)

PPA_RGB2YUV_PROTOCAL 配置 SRM 输出方向 RGB 转 YUV 使用协议。

0: BT601

1: BT709

(R/W)

见下页...

Register 39.4. PPA_SRM_COLOR_MODE_REG (0x0020)

接上页...

PPA_YUV422_RX_BYTE_ORDER 配置当 SRM 输入方向为 YUV422 时的排列格式。

0：YVYU

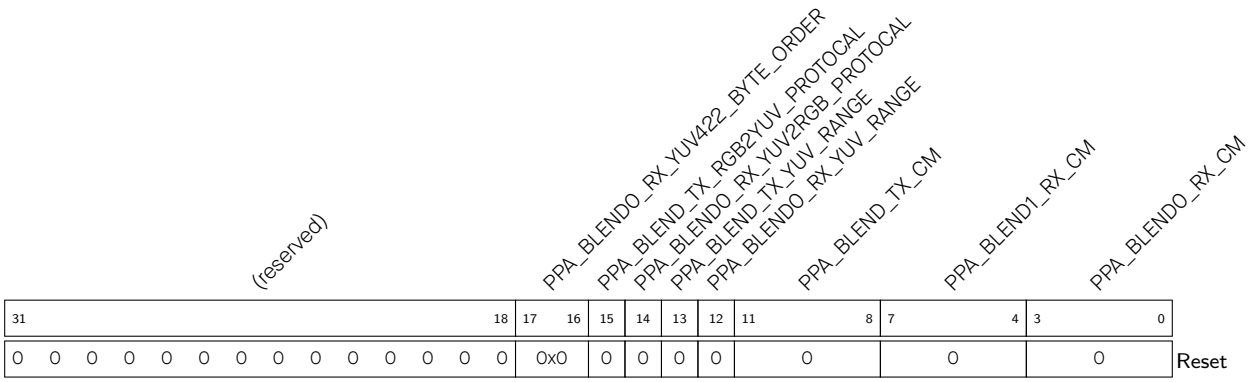
1：YUYV

2：VYUY

3：UYVY

(R/W)

Register 39.5. PPA_BLEND_COLOR_MODE_REG (0x0024)



PPA_BLEND0_RX_CM 配置 BLEND 背景图层输入颜色格式。

0: ARGB8888
1: RGB888
2: RGB565
4: L8
5: L4
8: YUV420
9: YUV422
12: GRAY
其他：保留
(R/W)

PPA_BLEND1_RX_CM 配置 BLEND 前景图层输入颜色格式。

0: ARGB8888
1: RGB888
2: RGB565
4: L8
5: L4
6: A8
7: A4
其他：保留
(R/W)

PPA_BLEND_TX_CM 配置 BLEND 输出颜色格式。

0: ARGB8888
1: RGB888
2: RGB565
8: YUV420
9: YUV422
12: GRAY
其他：保留
(R/W)

见下页...

Register 39.5. PPA_BLEND_COLOR_MODE_REG (0x0024)

接上页...

PPA_BLEND0_RX_YUV_RANGE 配置 BLEND 背景输入为 YUV 时的 YUV 范围。

0：限制范围

1：全范围

(R/W)

PPA_BLEND_TX_YUV_RANGE 配置 BLEND 输出为 YUV 时的 YUV 范围。

0：限制范围

1：全范围

(R/W)

PPA_BLEND0_RX_YUV2RGB_PROTOCAL 配置 BLEND 背景输入方向 YUV 转 RGB 使用协议。

0: BT601

1: BT709

(R/W)

PPA_BLEND_TX_RGB2YUV_PROTOCAL 配置 BLEND 输出方向 RGB 转 YUV 使用协议。

0: BT601

1: BT709

(R/W)

PPA_BLEND0_RX_YUV422_BYTE_ORDER 配置当 BLEND 背景输入方向为 YUV422 时的排列格式。

0：YVYU

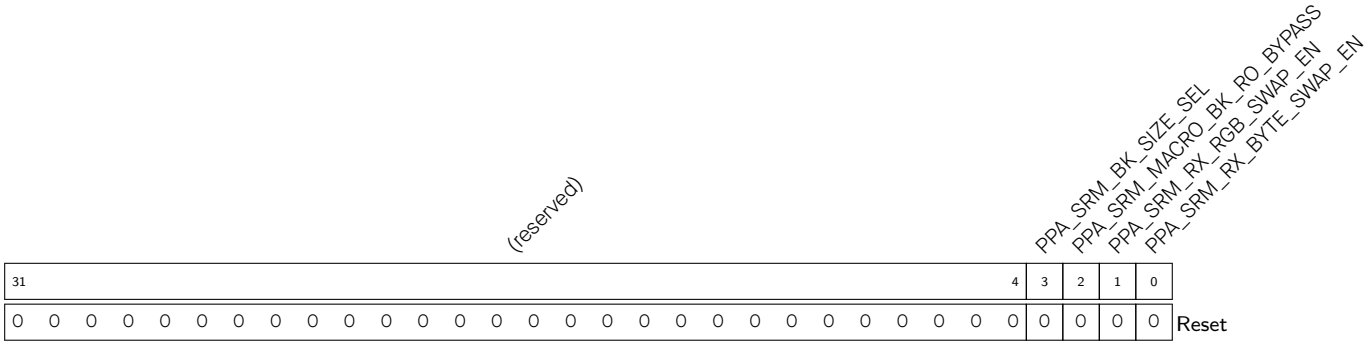
1：YUYV

2：VYUY

3：UYVY

(R/W)

Register 39.6. PPA_SRM_BYTE_ORDER_REG (0x0028)



PPA_SRM_RX_BYTE_SWAP_EN 配置 SRM 有效数据输入是否按字节进行翻转。

0：不翻转

1：翻转

(R/W)

PPA_SRM_RX_RGB_SWAP_EN 配置是否在 SRM 输入为 RGB 或 ARGB 时翻转各通道排列。

0：不翻转

1：RGB 翻转为 BGR，ARGB 翻转为 BGRA

(R/W)

PPA_SRM_MACRO_BK_RO_BYPASS 配置像素块重排功能。

0：打开像素块重排功能，自动计算是否需要重排

1：关闭像素块重排功能

(R/W)

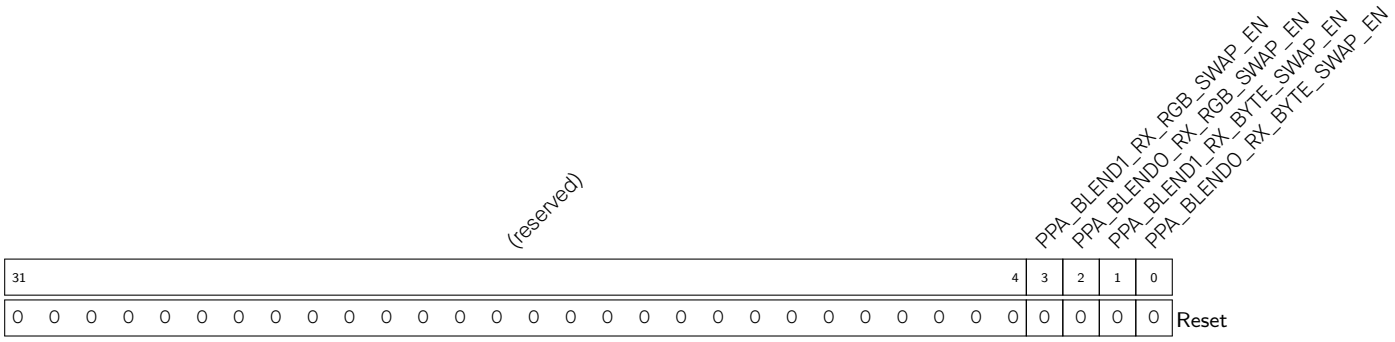
PPA_SRM_BK_SIZE_SEL 配置 SRM 像素块尺寸。

0：32x32

1：16x16

(R/W)

Register 39.7. PPA_BLEND_BYTE_ORDER_REG (0x002C)



PPA_BLEND0_RX_BYTE_SWAP_EN 配置 BLEND 背景图层有效数据输入是否按字节进行翻转。

0：不翻转

1：翻转

(R/W)

PPA_BLEND1_RX_BYTE_SWAP_EN 配置 BLEND 前景图层有效数据输入是否按字节进行翻转。

0：不翻转

1：翻转

(R/W)

PPA_BLEND0_RX_RGB_SWAP_EN 配置是否在 BLEND 背景图层输入为 RGB 或 ARGB 时翻转通道排列。

0：不翻转 RGB

1：RGB 翻转为 BGR，ARGB 翻转为 BGRA

(R/W)

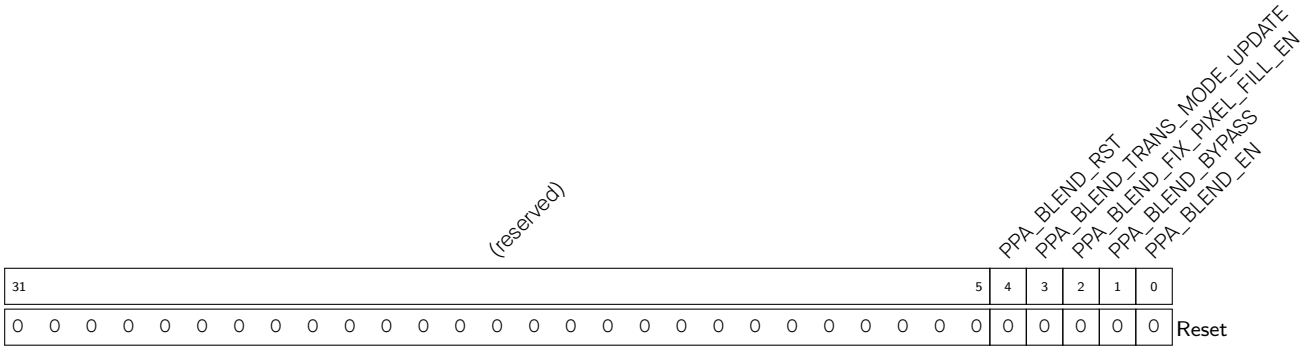
PPA_BLEND1_RX_RGB_SWAP_EN 配置是否在 BLEND 前景图层输入为 RGB 或 ARGB 时翻转通道排列。

0：不翻转 RGB

1：RGB 翻转为 BGR，ARGB 翻转为 BGRA

(R/W)

Register 39.8. PPA_BLEND_TRANS_MODE_REG (0x0034)



PPA_BLEND_EN 配置 BLEND 工作开关。

0：关闭 BLEND

1：打开 BLEND

(R/W)

PPA_BLEND_BYPASS 配置是否跳过 BLEND 直接输出背景图层。

0：跳过 BLEND

1：不跳过 BLEND

(R/W)

PPA_BLEND_FIX_PIXEL_FILL_EN 配置填充图像输出功能。

0：关闭填充图像输出功能

1：打开填充图像输出功能，此时只有 BLEND 输出工作，输出图像象素以及尺寸由 PPA_BLEND_TX_SIZE_REG 和 PPA_BLEND_TX_FIX_PIXEL 控制

(R/W)

PPA_BLEND_TRANS_MODE_UPDATE 写 1 更新传输模式，使 PPA_BLEND_FIX_PIXEL_FILL_EN 生效，并开启 BLEND 工作。(WT)

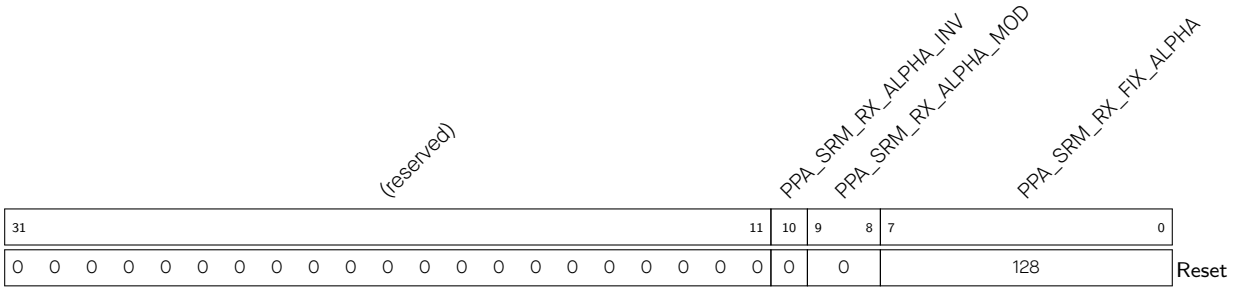
PPA_BLEND_RST 配置 BLEND 逻辑的复位。

0：释放复位

1：复位

(R/W)

Register 39.9. PPA_SRM_FIX_ALPHA_REG (0x0038)



PPA_SRM_RX_FIX_ALPHA 配置 SRM Alpha 通道值。(R/W)

PPA_SRM_RX_ALPHA_MOD 配置 SRM 输入数据的 Alpha 通道替换模式。

1：用 PPA_SRM_RX_FIX_ALPHA 作为最终 Alpha 值

2：原始 Alpha 值与 PPA_SRM_RX_FIX_ALPHA 相乘的高 8 位为最终 Alpha 值

其他：原始 Alpha 值为最终 Alpha 值

(R/W)

PPA_SRM_RX_ALPHA_INV 配置是否反相 SRM 输入数据的原始 Alpha 值。

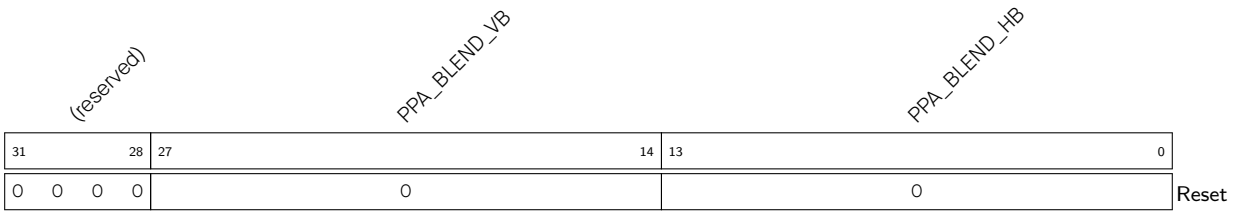
0：原始 Alpha 值不变

1：原始 Alpha 值反相，变为 255 减去原始 Alpha 值

对于不带 Alpha 通道的数据类型，原始 Alpha 值为 255

(R/W)

Register 39.10. PPA_BLEND_TX_SIZE_REG (0x003C)



PPA_BLEND_HB 配置填充图像输出功能下的水平方向尺寸，单位：像素。(R/W)

PPA_BLEND_VB 配置填充图像输出功能下的垂直方向尺寸，单位：像素。(R/W)

Register 39.11. PPA_BLEND_FIX_ALPHA_REG (0x0040)



PPA_BLEND0_RX_FIX_ALPHA 配置 BLEND 背景图层的 Alpha 通道值。(R/W)

PPA_BLEND1_RX_FIX_ALPHA 配置 BLEND 前景图层的 Alpha 通道值。(R/W)

PPA_BLEND0_RX_ALPHA_MOD 配置 BLEND 背景图层数据的 Alpha 通道替换模式。

1：用 PPA_BLEND0_RX_FIX_ALPHA 作为最终 Alpha 值

2：原始 Alpha 值与 PPA_BLEND0_RX_FIX_ALPHA 相乘的高 8 位为最终 Alpha 值

其他：原始 Alpha 值为最终 Alpha 值

(R/W)

PPA_BLEND1_RX_ALPHA_MOD 配置 BLEND 前景图层数据的 Alpha 通道替换模式。

1：用PPA_BLEND1_RX_FIX_ALPHA 作为最终 Alpha 值

2：原始 Alpha 值与 PPA_BLEND1_RX_FIX_ALPHA 相乘的高 8 位为最终 Alpha 值

其他：原始 Alpha 值为最终 Alpha 值

(R/W)

PPA_BLEND0_RX_ALPHA_INV 配置是否反相 BLEND 背景图层数据的原始 Alpha 值。

0：原始 Alpha 值不变

1：原始 Alpha 值反相，变为 255 减去原始 Alpha 值

对于不带 Alpha 通道的数据类型，原始 Alpha 值为 255。

(R/W)

PPA_BLEND1_RX_ALPHA_INV 配置是否反相 BLEND 前景图层数据的原始 Alpha 值。

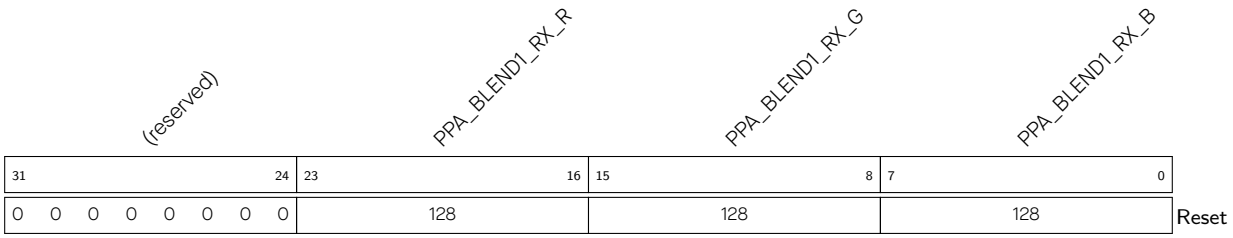
0：原始 Alpha 值不变

1：原始 Alpha 值反相，变为 255 减去原始 Alpha 值

对于不带 Alpha 通道的数据类型，原始 Alpha 值为 255。

(R/W)

Register 39.12. PPA_BLEND_RGB_REG (0x0048)

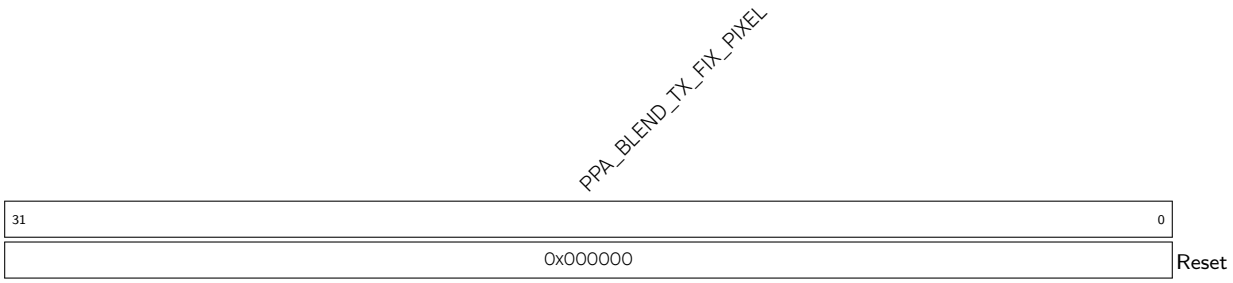


PPA_BLEND1_RX_B 配置 BLEND 前景图层为 A4/A8 时的 B 通道值。(R/W)

PPA_BLEND1_RX_G 配置 BLEND 前景图层为 A4/A8 时的 G 通道值。(R/W)

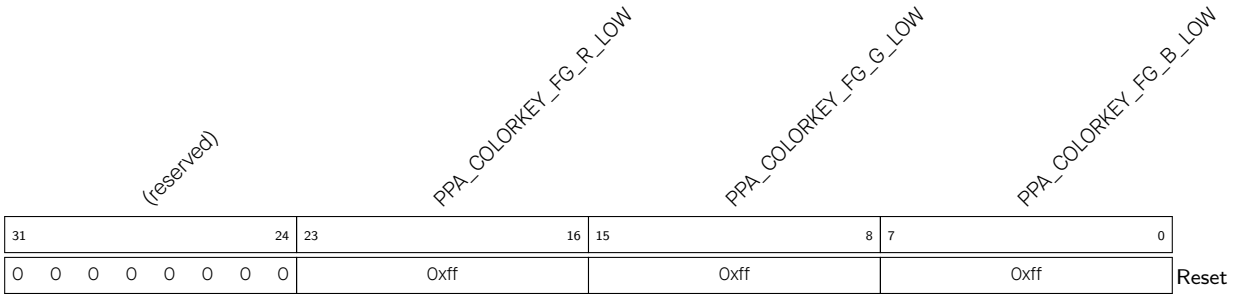
PPA_BLEND1_RX_R 配置 BLEND 前景图层为 A4/A8 时的 R 通道值。(R/W)

Register 39.13. PPA_BLEND_FIX_PIXEL_REG (0x004C)



PPA_BLEND_TX_FIX_PIXEL 配置 BLEND 填充图像输出模式下的像素值。
[31:24]：A，[23:16]：R，[15:8]：G，[7:0]：B。
(R/W)

Register 39.14. PPA_CK_FG_LOW_REG (0x0050)

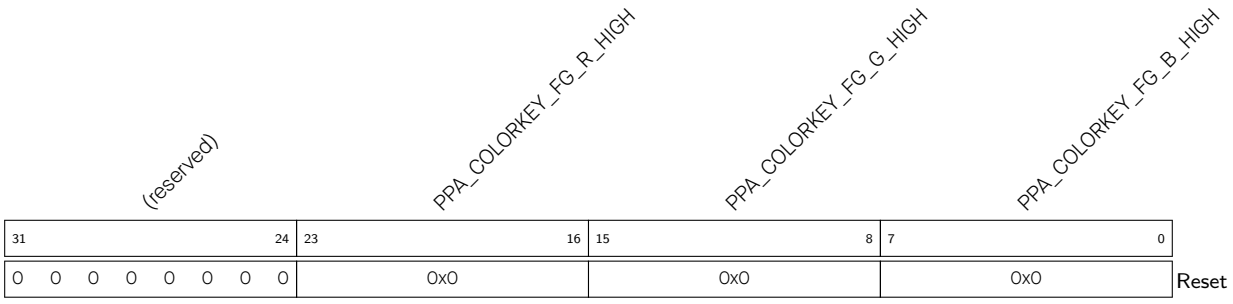


PPA_COLORKEY_FG_B_LOW 配置 BLEND 前景图层 B 通道 color-key 下阈值。(R/W)

PPA_COLORKEY_FG_G_LOW 配置 BLEND 前景图层 G 通道 color-key 下阈值。(R/W)

PPA_COLORKEY_FG_R_LOW 配置 BLEND 前景图层 R 通道 color-key 下阈值。(R/W)

Register 39.15. PPA_CK_FG_HIGH_REG (0x0054)

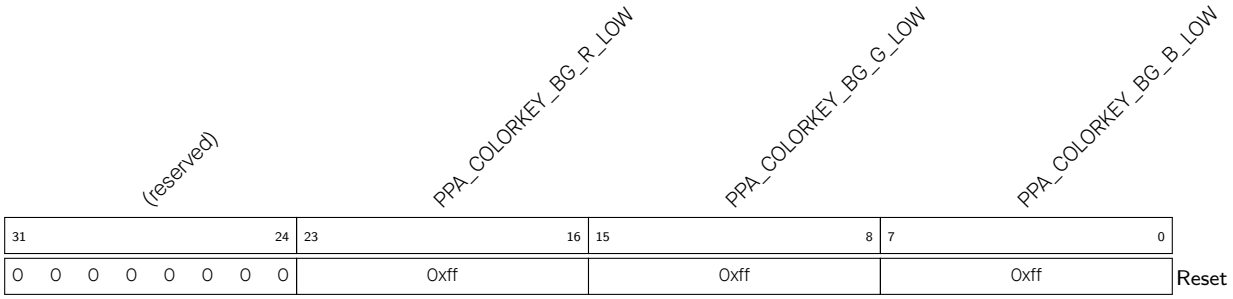


PPA_COLORKEY_FG_B_HIGH 配置 BLEND 前景图层 B 通道 color-key 上阈值。(R/W)

PPA_COLORKEY_FG_G_HIGH 配置 BLEND 前景图层 G 通道 color-key 上阈值。(R/W)

PPA_COLORKEY_FG_R_HIGH 配置 BLEND 前景图层 R 通道 color-key 上阈值。(R/W)

Register 39.16. PPA_CK_BG_LOW_REG (0x0058)

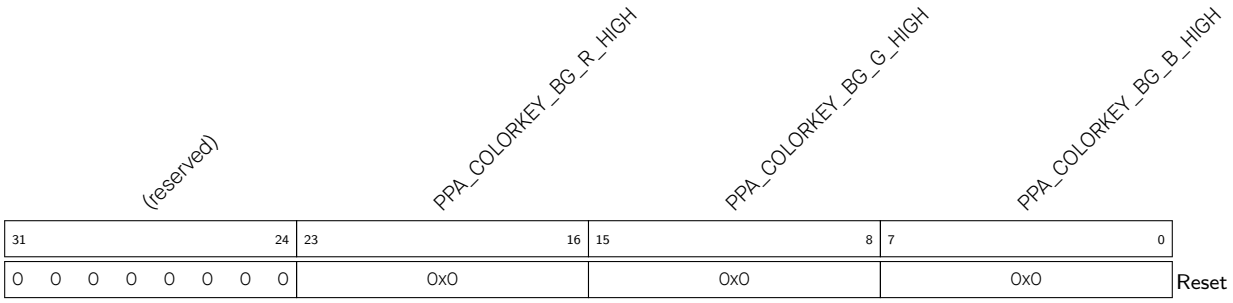


PPA_COLORKEY_BG_B_LOW 配置 BLEND 背景图层 B 通道 color-key 下阈值。(R/W)

PPA_COLORKEY_BG_G_LOW 配置 BLEND 背景图层 G 通道 color-key 下阈值。(R/W)

PPA_COLORKEY_BG_R_LOW 配置 BLEND 背景图层 R 通道 color-key 下阈值。(R/W)

Register 39.17. PPA_CK_BG_HIGH_REG (0x005C)



PPA_COLORKEY_BG_B_HIGH 配置 BLEND 背景图层 B 通道 color-key 上阈值。(R/W)

PPA_COLORKEY_BG_G_HIGH 配置 BLEND 背景图层 G 通道 color-key 上阈值。(R/W)

PPA_COLORKEY_BG_R_HIGH 配置 BLEND 背景图层 R 通道 color-key 上阈值。(R/W)

Register 39.18. PPA_CK_DEFAULT_REG (0x0060)



PPA_COLORKEY_DEFAULT_B 配置 BLEND color-key B 通道默认值。(R/W)

PPA_COLORKEY_DEFAULT_G 配置 BLEND color-key G 通道默认值。(R/W)

PPA_COLORKEY_DEFAULT_R 配置 BLEND color-key R 通道默认值。(R/W)

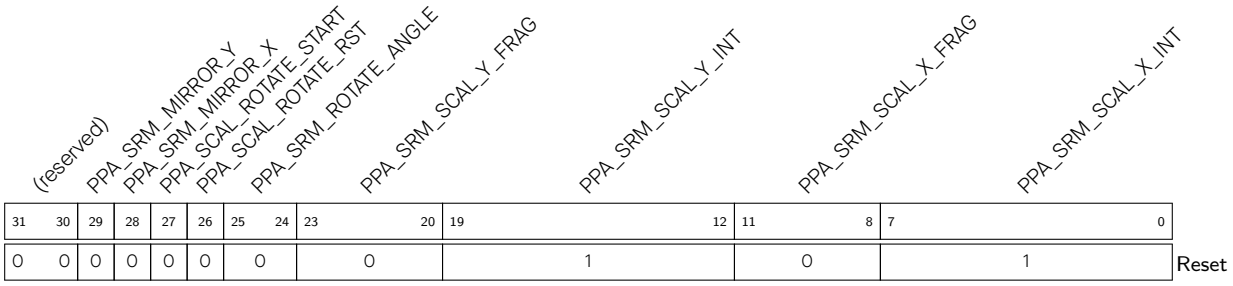
PPA_COLORKEY_FG_BG_REVERSE 配置当像素处于背景 color-key 范围但不在前景 color-key 范围时的处理方式

0：输出背景像素

1：输出前景像素

(R/W)

Register 39.19. PPA_SRM_SCAL_ROTATE_REG (0x0064)



PPA_SRM_SCAL_X_INT 配置 SRM 水平方向缩放系数整数部分。(R/W)

PPA_SRM_SCAL_X_FRAG 配置 SRM 水平方向缩放系数小数部分。(R/W)

PPA_SRM_SCAL_Y_INT 配置 SRM 垂直方向缩放系数整数部分。(R/W)

PPA_SRM_SCAL_Y_FRAG 配置 SRM 垂直方向缩放系数小数部分。(R/W)

PPA_SRM_ROTATE_ANGLE 配置 SRM 逆时针旋转角度

0: 0°
1: 90°
2: 180°
3: 270°
(R/W)

PPA_SCAL_ROTATE_RST 配置 SRM 逻辑的复位

0：释放复位
1：复位
(R/W)

PPA_SCAL_ROTATE_START 写 1 启动 SRM。(WT)

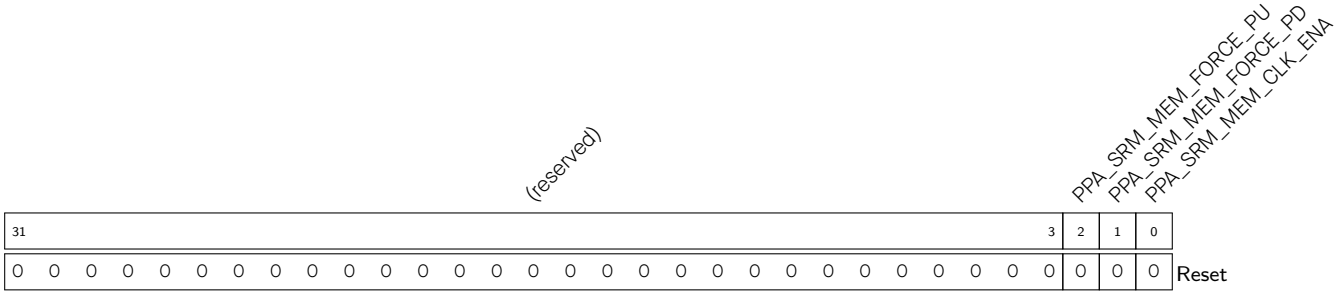
PPA_SRM_MIRROR_X 配置 SRM 水平方向镜像

0: 关闭水平镜像
1: 打开水平镜像
(R/W)

PPA_SRM_MIRROR_Y 配置 SRM 垂直方向镜像

0: 关闭垂直镜像
1: 打开垂直镜像
(R/W)

Register 39.20. PPA_SRM_MEM_PD_REG (0x0068)



PPA_SRM_MEM_CLK_ENA 配置是否强制打开 SRM MEM 时钟

0：不强制打开

1：强制打开

(R/W)

PPA_SRM_MEM_FORCE_PD 配置 SRM MEM 是否强制掉电

0：不强制掉电

1：强制掉电

(R/W)

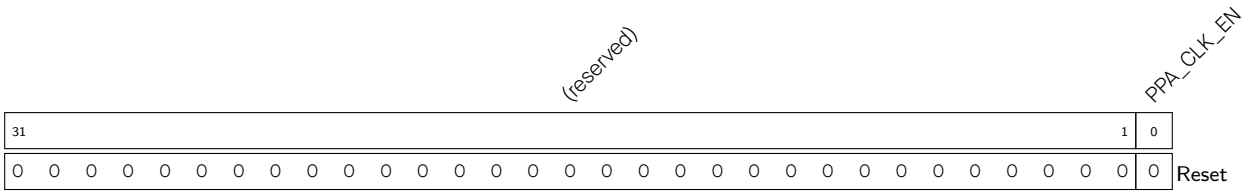
PPA_SRM_MEM_FORCE_PU 配置 SRM MEM 是否强制上电

0：不强制上电

1：强制上电

(R/W)

Register 39.21. PPA_REG_CONF_REG (0x006C)



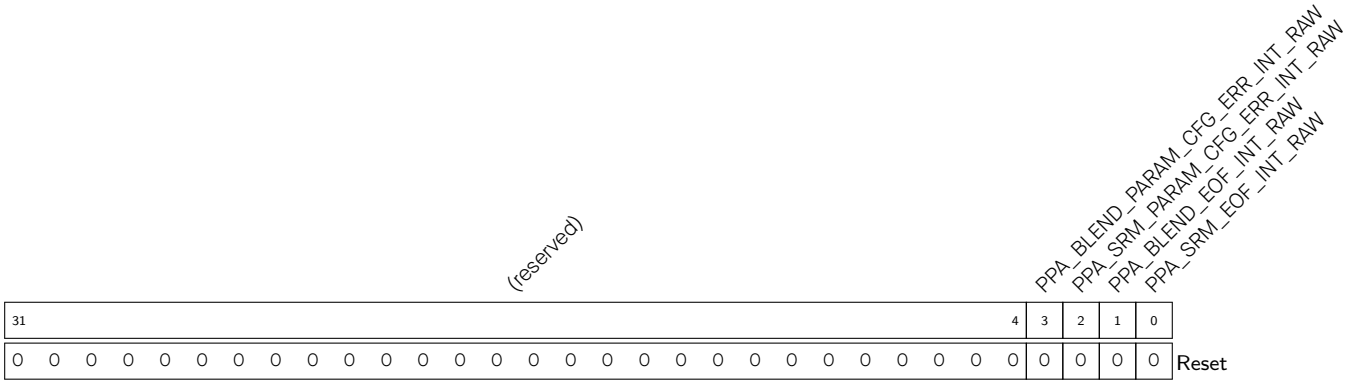
PPA_CLK_EN 配置是否常开 PPA 寄存器时钟。

0：时钟只会在发生寄存器访问时打开

1：时钟常开

(R/W)

Register 39.22. PPA_INT_RAW_REG (0x0010)



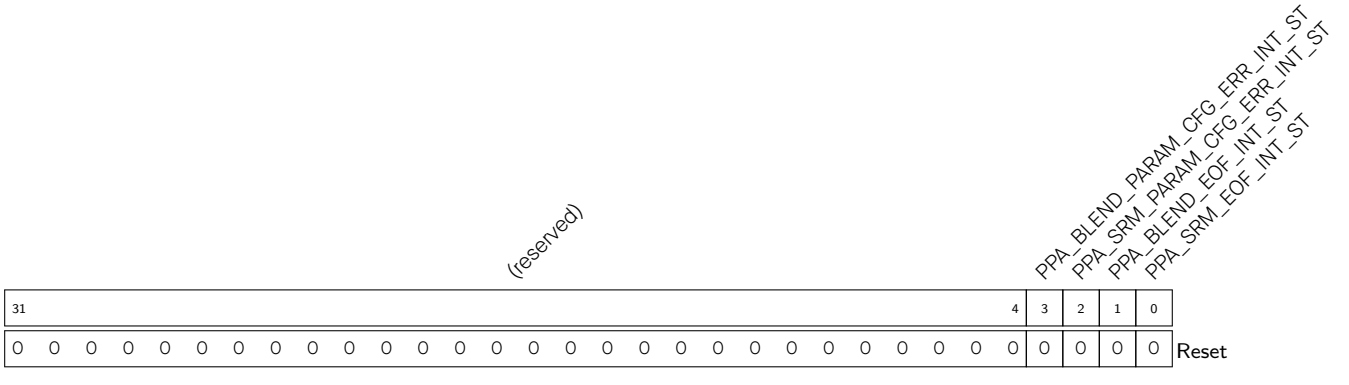
PPA_SRM_EOF_INT_RAW 表示 PPA_SRM_EOF_INT 的原始中断状态。(R/WTC/SS)

PPA_BLEND_EOF_INT_RAW 表示 PPA_BLEND_EOF_INT 的原始中断状态。(R/WTC/SS)

PPA_SRM_PARAM_CFG_ERR_INT_RAW 表示 PPA_SRM_PARAM_CFG_ERR_INT 的原始中断状态。可通过 PPA_SRM_PARAM_ERR_ST_REG 查看具体错误。(R/WTC/SS)

PPA_BLEND_PARAM_CFG_ERR_INT_RAW 表示 PPA_BLEND_PARAM_CFG_ERR_INT 的原始中断状态。可通过 PPA_BLEND_ST_REG 查看具体错误。(R/WTC/SS)

Register 39.23. PPA_INT_ST_REG (0x0014)



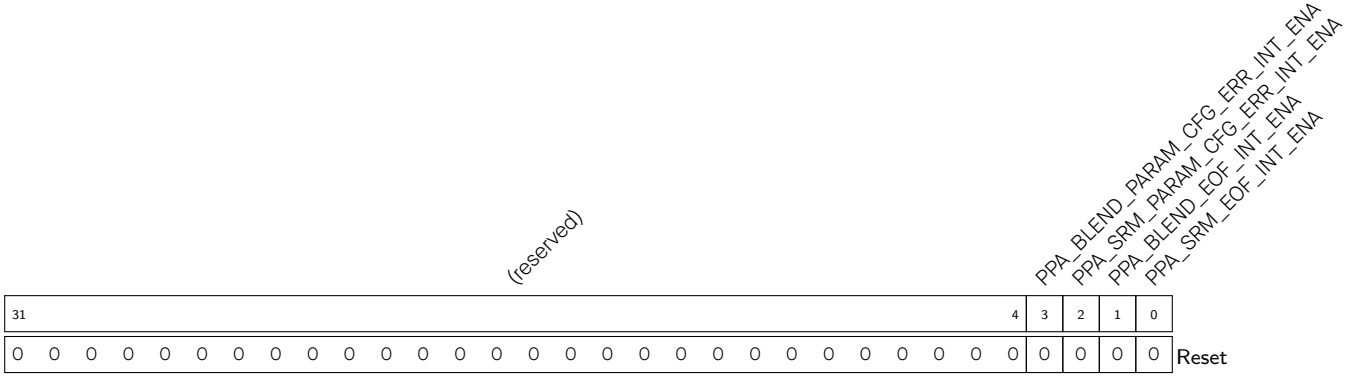
PPA_SRM_EOF_INT_ST 表示 PPA_SRM_EOF_INT 的屏蔽中断状态。(RO)

PPA_BLEND_EOF_INT_ST 表示 PPA_BLEND_EOF_INT 的屏蔽中断状态。(RO)

PPA_SRM_PARAM_CFG_ERR_INT_ST 表示 PPA_SRM_PARAM_CFG_ERR_INT 的屏蔽中断状态。(RO)

PPA_BLEND_PARAM_CFG_ERR_INT_ST 表示 PPA_BLEND_PARAM_CFG_ERR_INT 的屏蔽中断状态。(RO)

Register 39.24. PPA_INT_ENA_REG (0x0018)



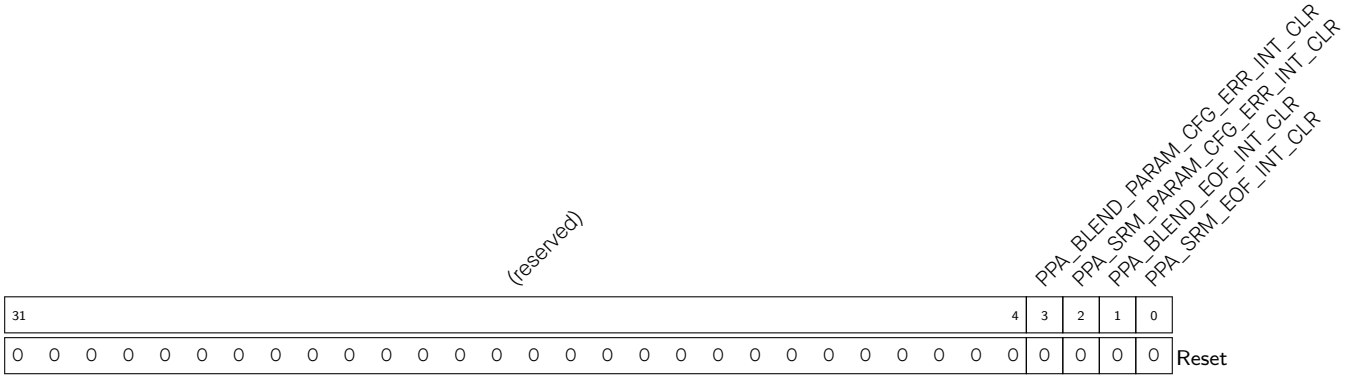
PPA_SRM_EOF_INT_ENA 写 1 使能 PPA_SRM_EOF_INT。
(R/W)

PPA_BLEND_EOF_INT_ENA 写 1 使能 PPA_BLEND_EOF_INT。
(R/W)

PPA_SRM_PARAM_CFG_ERR_INT_ENA 写 1 使能 PPA_SRM_PARAM_CFG_ERR_INT。
(R/W)

PPA_BLEND_PARAM_CFG_ERR_INT_ENA 写 1 使能 PPA_BLEND_PARAM_CFG_ERR_INT。
(R/W)

Register 39.25. PPA_INT_CLR_REG (0x001C)



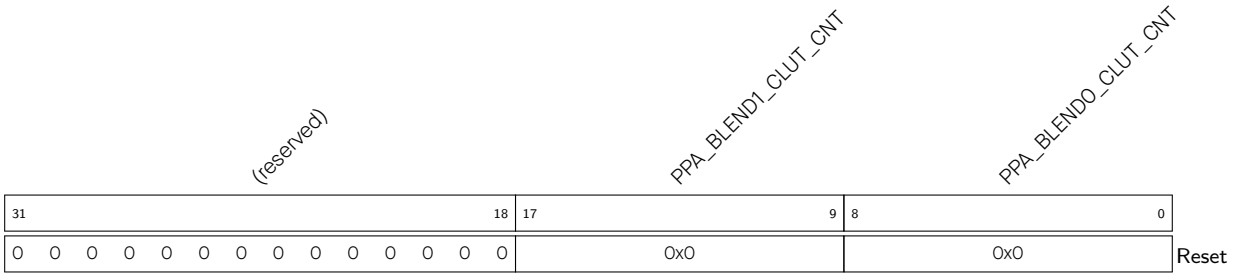
PPA_SRM_EOF_INT_CLR 写 1 清空 PPA_SRM_EOF_INT。(WT)

PPA_BLEND_EOF_INT_CLR 写 1 清空 PPA_BLEND_EOF_INT。(WT)

PPA_SRM_PARAM_CFG_ERR_INT_CLR 写 1 清空 PPA_SRM_PARAM_CFG_ERR_INT。(WT)

PPA_BLEND_PARAM_CFG_ERR_INT_CLR 写 1 清空 PPA_BLEND_PARAM_CFG_ERR_INT。(WT)

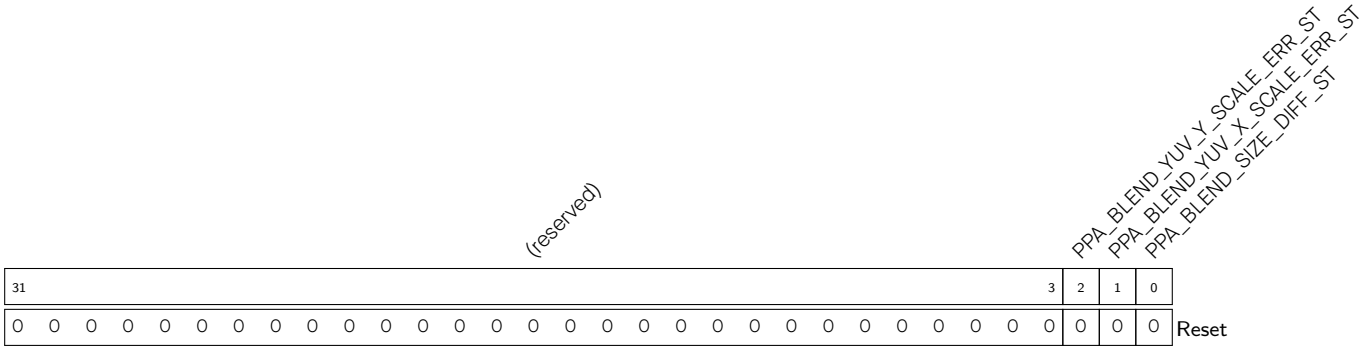
Register 39.26. PPA_CLUT_CNT_REG (0x0070)



PPA_BLEND0_CLUT_CNT 表示 BLEND 背景图层 CLUT 在 FIFO 模式下的当前写地址。(RO)

PPA_BLEND1_CLUT_CNT 表示 BLEND 前景图层 CLUT 在 FIFO 模式下的当前写地址。(RO)

Register 39.27. PPA_BLEND_ST_REG (0x0074)



PPA_BLEND_SIZE_DIFF_ST 表示 BLEND 前景、背景尺寸是否一致。

0：BLEND 前景、背景尺寸一致

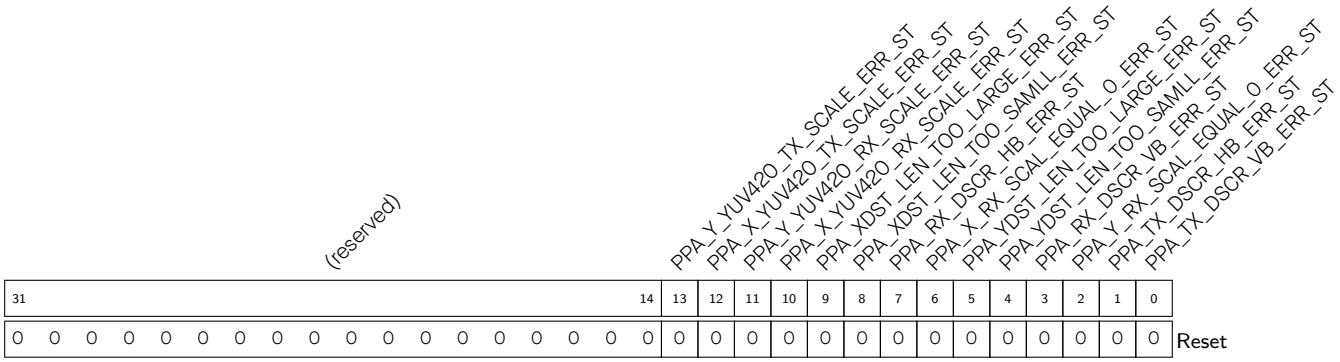
1：BLEND 前景、背景尺寸不一致

(RO)

PPA_BLEND_YUV_X_SCALE_ERR_ST 表示图像格式为 YUV422 或 YUV420 时 X 方向尺寸为奇数。(RO)

PPA_BLEND_YUV_Y_SCALE_ERR_ST 表示图像格式为 YUV420 时 Y 方向尺寸为奇数。(RO)

Register 39.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)



PPA_TX_DSCR_VB_ERR_ST 表示 SRM 输出图像块垂直尺寸加上输出链表配置的垂直偏移 Y 是否超出输出链表配置的整幅图像垂直尺寸 VA。

0：未发生该错误

1：发生该错误

(RO)

PPA_TX_DSCR_HB_ERR_ST 表示 SRM 输出图像块水平尺寸加上输出链表配置的水平偏移 X 是否超出输出链表配置的整幅图像水平尺寸 HA。

0：未发生该错误

1：发生该错误

(RO)

PPA_Y_RX_SCAL_EQUAL_0_ERR_ST 表示 SRM 输入图像块垂直方向缩放系数是否为 0。

0：未发生该错误

1：发生该错误

(RO)

PPA_RX_DSCR_VB_ERR_ST 表示 SRM 输入链表配置的垂直方向图像块尺寸 VB 加上垂直方向偏移 Y 是否大于整幅图像垂直尺寸 VA。

0：未发生该错误

1：发生该错误

(RO)

PPA_YDST_LEN_TOO_SAMLL_ERR_ST 表示 SRM 缩放后的图像块垂直方向尺寸是否为 0。

0：未发生该错误

1：发生该错误

(RO)

PPA_YDST_LEN_TOO_LARGE_ERR_ST 表示 SRM 缩放后的图像块垂直方向尺寸是否大于 8191。

0：未发生该错误

1：发生该错误

(RO)

PPA_X_RX_SCAL_EQUAL_0_ERR_ST 表示 SRM 输入图像块水平方向缩放系数是否为 0。

0：未发生该错误

1：发生该错误

(RO)

见下页...

Register 39.28. PPA_SRM_PARAM_ERR_ST_REG (0x0078)

接上页...

PPA_RX_DSCR_HB_ERR_ST 表示 SRM 输入链表配置的水平方向图像块尺寸 HB 加上水平方向偏移 X 是否大于整幅图像水平尺寸 HA。

0：未发生该错误

1：发生该错误

(RO)

PPA_XDST_LEN_TOO_SAMLL_ERR_ST 表示 SRM 缩放后的图像块水平方向尺寸是否为 0。

0：未发生该错误

1：发生该错误

(RO)

PPA_XDST_LEN_TOO_LARGE_ERR_ST 表示 SRM 缩放后的图像块水平方向尺寸是否大于 8191。

0：未发生该错误

1：发生该错误

(RO)

PPA_X_YUV420_RX_SCALE_ERR_ST 表示 SRM 输入链表的 HA/HB/X 参数在输入类型为 YUV420 时是否为奇数。

0：未发生该错误

1：发生该错误

(RO)

PPA_Y_YUV420_RX_SCALE_ERR_ST 表示 SRM 输入链表的 VA/VB/Y 参数在输入类型为 YUV420 时是否为奇数。

0：未发生该错误

1：发生该错误

(RO)

PPA_X_YUV420_TX_SCALE_ERR_ST 表示 SRM 输出链表的 HA/HB/X 参数在输出类型为 YUV420 时是否为奇数。

0：未发生该错误

1：发生该错误

(RO)

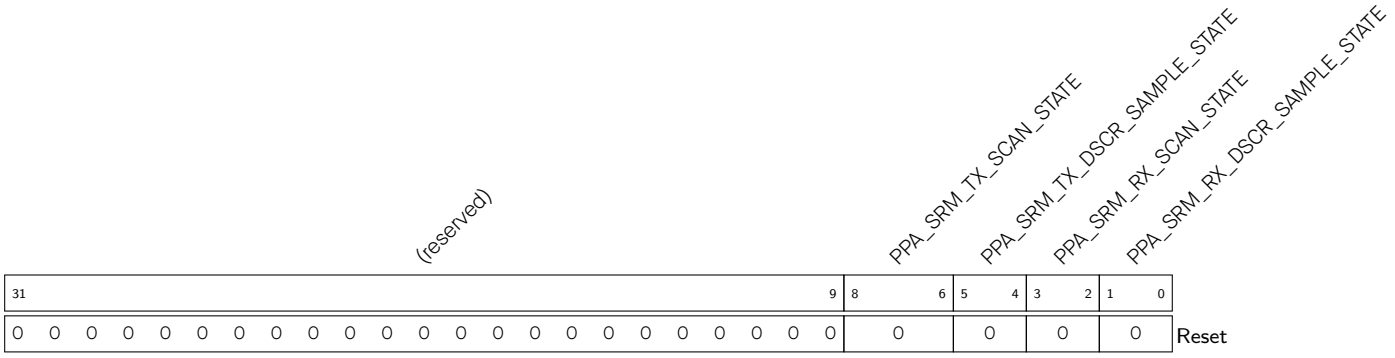
PPA_Y_YUV420_TX_SCALE_ERR_ST 表示 SRM 输出链表的 VA/VB/Y 参数在输出类型为 YUV420 时是否为奇数。

0：未发生该错误

1：发生该错误

(RO)

Register 39.29. PPA_SRM_STATUS_REG (0x007C)



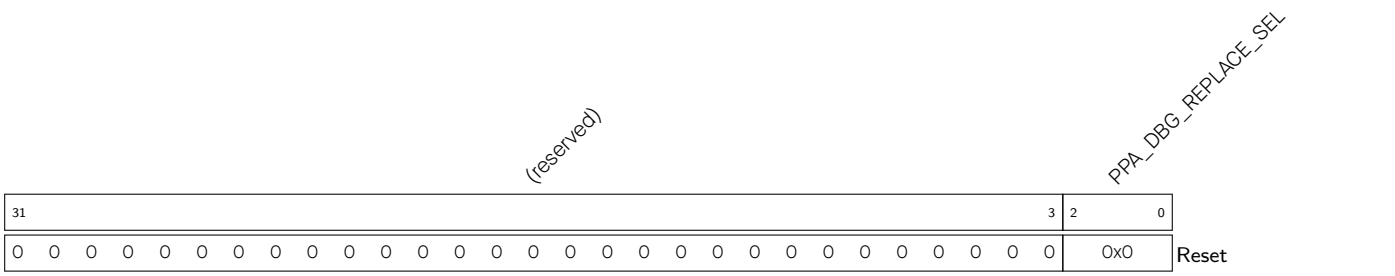
PPA_SRM_RX_DSCR_SAMPLE_STATE 表示 SRM RX 描述符状态。(RO)

PPA_SRM_RX_SCAN_STATE 表示 SRM RX 像素扫描状态。(RO)

PPA_SRM_TX_DSCR_SAMPLE_STATE 表示 SRM TX 描述符状态。(RO)

PPA_SRM_TX_SCAN_STATE 表示 SRM TX 像素扫描状态。(RO)

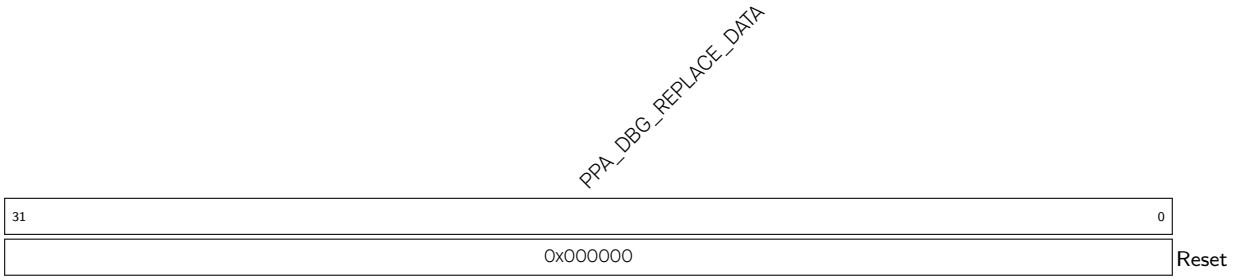
Register 39.30. PPA_DEBUG_CTRL0_REG (0x0090)



PPA_DBG_REPLACE_SEL 配置替换像素的位置。

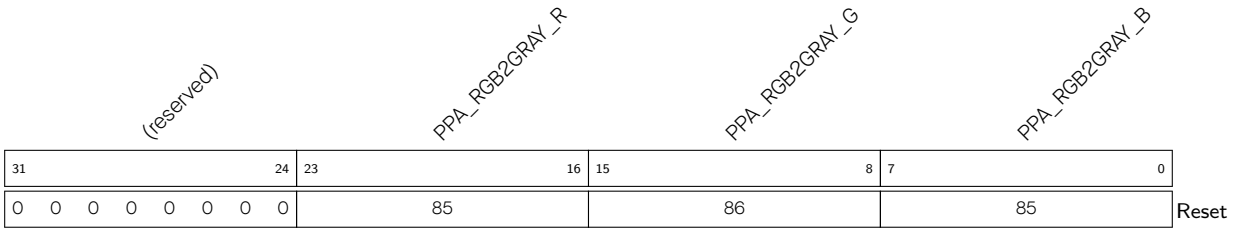
0：不替换
1：SRM RX 输入端
2：SRM 插值输出端
3: SRM TX 输出端
4：BLEND 前景输入端
5：BLEND 背景输入端
6：BLEND 输出端
(R/W)

Register 39.31. PPA_DEBUG_CTRL1_REG (0x0094)



PPA_DBG_REPLACE_DATA 配置替换像素的值。(R/W)

Register 39.32. PPA_RGB2GRAY_REG (0x0098)

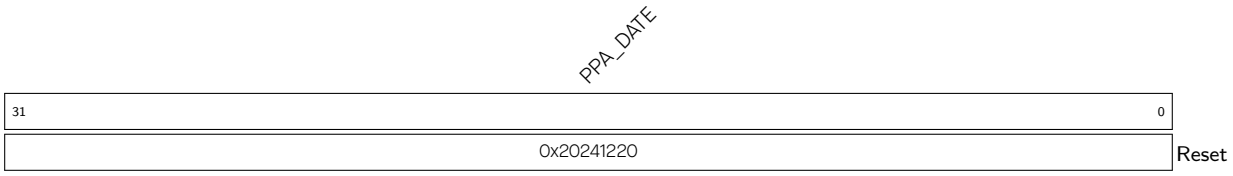


PPA_RGB2GRAY_B 配置 RGB 转 GRAY 时 B 的系数。(R/W)

PPA_RGB2GRAY_G 配置 RGB 转 GRAY 时 G 的系数。(R/W)

PPA_RGB2GRAY_R 配置 RGB 转 GRAY 时 R 的系数。(R/W)

Register 39.33. PPA_DATE_REG (0x0100)



PPA_DATE 版本控制寄存器。(R/W)

第 40 章

LCD 与 Camera 控制器 (LCD_CAM)

40.1 概述

ESP32-P4 的 LCD_CAM 控制器包含一个独立的 LCD 控制模块和 Camera（摄像头）控制模块，可以外接 LCD 和摄像头设备，功能灵活多样。

LCD 模块提供了一个 LCD 接口，可以连接兼容 RGB、Motorola 6800 (MOTO6800) 和 Intel 8080 (I8080) 的 LCD 设备。LCD 模块负责将视频数据发送到 LCD 设备。

Camera 模块提供一个并行摄像头接口，用于连接 DVP 摄像头，支持 8/16 位模式。Camera 模块可以管理与摄像头的通信，接收图像或视频帧。

ESP32-P4 的 LCD_CAM 模块特别适用于需要屏幕显示和视频处理的开发项目。

40.2 特性

LCD_CAM 模块具有以下主要特性：

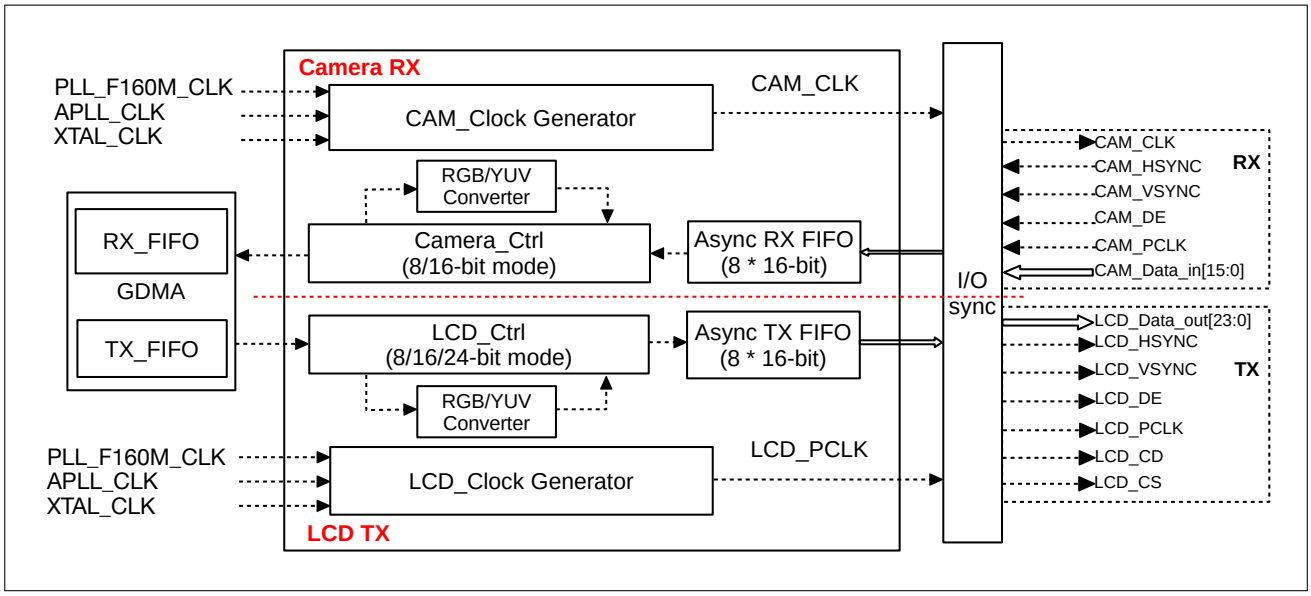
- 支持以下工作模式：
  - LCD 主机发送模式
  - Camera 从机接收模式
  - Camera 主机接收模式
- 支持同时外接 LCD 和摄像头设备
- 当外接 LCD 设备时，支持：
  - 8/16/24 位并行输出模式
  - RGB、MOTO6800、I8080 多种 LCD 模式
  - LCD 数据可由 GDMA 取自内部或外部存储器
- 当外接摄像头设备（即 DVP 图像传感器）时，支持：
  - 8/16 位并行输入模式
  - 视频数据可由 GDMA 存入内部或外部存储器
- 支持 LCD_CAM 接口中断

40.3 功能描述

40.3.1 功能框图

图 40.3-1 是 ESP32-P4 LCD_CAM 模块的结构框图，包含：

- 一个发送控制单元 (LCD_Ctrl)
- 一个接收控制单元 (Camera_Ctrl)
- 一个发送异步 FIFO (Async TX FIFO)，用于与外部设备如 LCD 进行交互
- 一个接收异步 FIFO (Async RX FIFO)，用于与外部设备如摄像头进行交互
- 两个时钟生成模块（LCD_Clock Generator 和 CAM_Clock Generator），用于生成对应模块的时钟
- 两个格式转换模块 (RGB/YUV Converter)，用于各种格式的视频数据互相转换



-→: 控制信号 =>: 数据流

图 40.3-1. LCD_CAM 功能框图

40.3.2 信号描述

表 40.3-1 列出了 LCD_CAM 工作所需的输入和输出信号。

三种工作模式的解释如下：

- **Camera 从机接收模式：**在此模式下，Camera 模块作为通讯接收方输入控制信号。
- **Camera 主机接收模式：**在此模式下，Camera 模块向从机发送时钟信号，从机收到时钟信号后向 Camera 模块发送数据。
- **LCD 主机发送模式：**在此模式下，LCD 模块作为通讯发起方向从机发送控制信号。

表 40.3-1. 信号描述

工作模式	信号[1]	方向	功能
Camera 从机接收模式	CAM_PCLK	输入	像素时钟输入信号
	CAM_VSYNC	输入	帧同步输入信号 (VSYNC)[3]

见下页

表 40.3-1 – 接上页

工作模式	信号[1]	方向	功能
	CAM_HSYNC	输入	行同步输入信号 (HSYNC)[3]
	CAM_DE	输入	数据有效输入信号 (DE)[3]
	CAM_Data_in[*N*:0][2]	输入	并行输入数据总线，支持 8/16 位并行数据输入
Camera 主机接收模式	CAM_PCLK	输入	像素时钟输入信号
	CAM_CLK	输出	主机时钟输出信号
	CAM_VSYNC	输入	帧同步输入信号[3]
	CAM_HSYNC	输入	行同步输入信号[3]
	CAM_DE	输入	数据有效输入信号[3]
	CAM_Data_in[*N*:0][2]	输入	并行输入数据总线，支持 8/16 位并行数据输入
LCD 主机发送模式	LCD_PCLK	输出	LCD 像素时钟输出信号
	LCD_HSYNC	输出	RGB 格式下，用作行同步信号
	LCD_VSYNC	输出	RGB 格式下，用作帧同步信号
	LCD_DE	输出	RGB 格式下，用作数据有效信号
	LCD_CD	输出	I8080 格式下，用作命令和数据信号 (CD)
	LCD_CS	输出	I8080/MOTO6800 格式下，用作片选信号 (CS)
	LCD_Data_out[*M*:0][2]	输出	并行输出数据总线，支持 8/16/24 位并行数据输出

[1] LCD_CAM 的所有信号均需要经过 GPIO 交换矩阵映射到芯片管脚。更多信息请参考章节 *10 GPIO 交换矩阵和 IO MUX*。

[2] 输入信号位宽 8 或 16 时，则 *N* 分别为 7 或 15。输入信号位宽 8、16 或 24 时，*M* 分别为 7、15 或 23。

[3] LCD_CAM_CAM_VH_DE_MODE_EN 为 1（VSYNC + HSYNC 模式）时：VSYNC、HSYNC、DE 线同时控制数据有效，因此用户需要接 VSYNC、HSYNC、DE 信号线。LCD_CAM_CAM_VH_DE_MODE_EN 为 0（DE 模式）时：VSYNC、DE 线同时控制数据有效，HSYNC 线可以不接。但是在此模式下，Camera 的 YUV-RGB 转换功能将不可用。

40.3.3 LCD_CAM 模块时钟

40.3.3.1 LCD 时钟

时钟源经时钟生成模块处理后生成 LCD 模块所需的时钟，如图 40.3-2 所示。其中，

- LCD_CLK 为 LCD 模块的主时钟，由时钟源分频获得；
- LCD_PCLK 为像素时钟，由 LCD_CLK 分频获得。

HP_SYS_CLKRST_LCD_CLK_EN 用于使能 LCD_CLK_SRC（时钟源），HP_SYS_CLKRST_LCD_CLK_SRC_SEL 用于选择时钟源：

- 0：选择 XTAL_CLK
- 1：选择 PLL_F160M_CLK
- 2：选择 APLL_CLK
- 3：关闭 LCD 时钟源

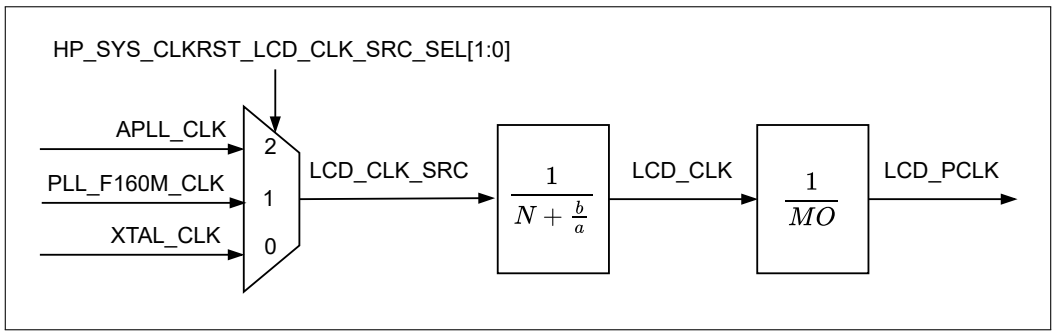


图 40.3-2. LCD 时钟

LCD_CLK 的频率 f_{LCD_CLK} 与分频器时钟源频率 $f_{LCD_CLK_SRC}$ 间的关系如下：

$$f_{LCD_CLK} = \frac{f_{LCD_CLK_SRC}}{N + \frac{b}{a}}$$

其中，*N*、*a*、*b* 与 HP_SYS_CLKRST_LCD_CLK_DIV_NUM、HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR、HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR 的值相关，具体为：

- HP_SYS_CLKRST_LCD_CLK_DIV_NUM = 0 时，*N* = 1，*a* 和 *b* 的值无需关注，始终为 1 分频。
- HP_SYS_CLKRST_LCD_CLK_DIV_NUM >=1 时，*N* = HP_SYS_CLKRST_LCD_CLK_DIV_NUM + 1。
  - 对于整数分频，请将 HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR 和 HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR 清零。
  - 对于小数分频，*b* = HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR，*a* = HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR，请保证 HP_SYS_CLKRST_LCD_CLK_DIV_NUMERATOR 的值小于 HP_SYS_CLKRST_LCD_CLK_DIV_DENOMINATOR 的值。

LCD 模块的像素时钟 LCD_PCLK 由 LCD_CLK 分频获得。即：

$$f_{LCD_PCLK} = \frac{f_{LCD_CLK}}{MO}$$

其中，MO 由 LCD_CAM_LCD_CLK_EQU_SYSCLK 和 LCD_CAM_LCD_CLKCNT_N 决定，具体为：

- LCD_CAM_LCD_CLK_EQU_SYSCLK = 1 时，MO = 1。
- LCD_CAM_LCD_CLK_EQU_SYSCLK = 0 时，MO = LCD_CAM_LCD_CLKCNT_N + 1。

注意：

- LCD_CAM_LCD_CLKCNT_N 不可配置为 0。
- 使用小数分频功能会产生时钟抖动。当 LCD_CLK 和 LCD_PCLK 无法通过 PLL_F160M_CLK 整数分频产生时，可使用 APLL_CLK 作为时钟源，详情请参考章节 11 复位和时钟。

40.3.3.2 Camera 时钟

时钟源经时钟生成模块处理后生成 Camera 模块所需的时钟，如图 40.3-3 所示。其中，

- CAM_CLK 为 Camera 模块的主机时钟输出，由时钟源分频获得。
- CAM_PCLK 为像素时钟，从 Camera 从机输入获得。

HP_SYS_CLKRST_CAM_CLK_EN 用于使能 CAM_CLK_SRC（时钟源），HP_SYS_CLKRST_CAM_CLK_SRC_SEL 用于选择时钟源：

- 0：XTAL_CLK
- 1：PLL_F160M_CLK
- 2：APLL_CLK
- 3：关闭 Camera 时钟源

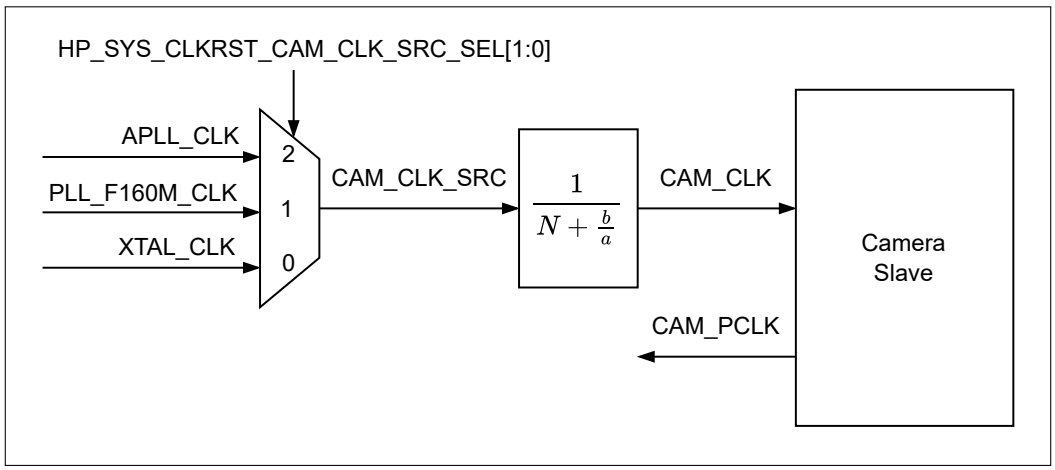


图 40.3-3. Camera 时钟

CAM_CLK 的频率 f_{CAM_CLK} 与分频器时钟源频率 $f_{CAM_CLK_SRC}$ 间的关系如下：

$$f_{CAM_CLK} = \frac{f_{CAM_CLK_SRC}}{N + \frac{b}{a}}$$

其中，2 <= *N* <= 256，*N* 对应为 HP_SYS_CLKRST_CAM_CLK_DIV_NUM 的值，具体为：

- HP_SYS_CLKRST_CAM_CLK_DIV_NUM = 0 时，*N* = 256。
- HP_SYS_CLKRST_CAM_CLK_DIV_NUM = 1 时，*N* = 2。
- HP_SYS_CLKRST_CAM_CLK_DIV_NUM 为其它值时，*N* = HP_SYS_CLKRST_CAM_CLK_DIV_NUM 的值。

b 对应 HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR 的值，*a* 对应 HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR 的值。

- 对于整数分频，HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR 和 HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR 都清零。
- 对于小数分频，HP_SYS_CLKRST_CAM_CLK_DIV_NUMERATOR 的值应小于 HP_SYS_CLKRST_CAM_CLK_DIV_DENOMINATOR 的值。

40.3.4 LCD_CAM 模块复位

LCD_CAM 模块共包含下列可应用于不同范围的复位寄存器：

- HP_SYS_CLKRST_RST_EN_LCDCAM：配置该复位寄存器，可以复位 LCD_CAM 模块中的所有子模块。
- LCD_CAM_LCD_RESET：配置该寄存器可复位所有 LCD 通路部分，包括发送控制单元 (LCD_Ctrl)、LCD 格式转换模块 (RGB/YUV Converter) 和 Async TX FIFO。配置该寄存器不会影响 Camera 部分的正常运行。

- LCD_CAM_CAM_RESET：配置该寄存器可复位所有 Camera 通路部分，包括接收控制单元 (Camera_Ctrl)、Camera 格式转换模块 (RGB/YUV Converter) 和 Async RX FIFO。配置该寄存器不会影响 LCD 通路的正常运行。
- LCD_CAM_LCD_AFIFO_RESET：配置该寄存器可以复位 Async TX FIFO，而不影响任何其他模块的运行。例如，可以在 LCD 每次发送完一帧时配置该复位寄存器，以防止前一帧的数据传输错误影响到下一帧的发送。
- LCD_CAM_CAM_AFIFO_RESET：配置该寄存器可以复位 Async RX FIFO，而不影响任何其他模块的运行。例如，可以在 Camera 每次接收完一帧时配置该复位寄存器，以防止前一帧的数据传输错误影响到下一帧的接收。

注意：

- 复位寄存器 HP_SYS_CLKRST_RST_EN_LCDCAM 不是硬件自清，即写 1 使能复位，写 0 释放复位。
- 上述复位寄存器除 HP_SYS_CLKRST_RST_EN_LCDCAM 以外均为硬件自清，即写 1 后，硬件会生成一个复位脉冲，并自动清零该寄存器。
- 在模块和 FIFO 复位之前，需要先配置模块时钟 LCD_CLK 和 CAM_CLK。

40.3.5 LCD_CAM 数据格式控制

40.3.5.1 LCD 数据格式控制

在使用 LCD 发送数据时，数据会经过三个阶段的处理过程：

1. 阶段一：数据预处理
   从 GDMA 获取输入数据，并进行数据预处理，处理完成后将数据输出给 YUV-RGB 格式转换模块。
2. 阶段二：数据色彩空间转换
   在 YUV-RGB 格式转换模块中进行数据处理（如果不开启 YUV-RGB 格式转换模块，这一阶段的数据将不发生任何改变）。
3. 阶段三：数据后处理
   LCD 模块从 YUV-RGB 格式转换模块取得数据，并进行数据后处理，处理完成后发送到 GPIO。

> **说明：**
> 每一阶段的输出数据，均为下一阶段的输入数据。

以下详细介绍每个阶段的配置。

阶段一：数据预处理

在阶段一中，配置以下寄存器可以调整来自 GDMA 的数据的位宽、数据位顺序和字节序：

- LCD_CAM_LCD_BYTE_MODE
  - 0：LCD 从 GDMA 输入的数据位宽为 8 个比特
  - 1：LCD 从 GDMA 输入的数据位宽为 16 个比特
  - 2：LCD 从 GDMA 输入的数据位宽为 24 个比特
- LCD_CAM_LCD_BIT_ORDER
  - 0：不反转

- 1：反转输入数据的数据位顺序

- LCD_CAM_LCD_BYTE_ORDER
  - 0：不反转
  - 1：反转数据字节顺序。仅在输入数据位宽为 16 比特或 24 比特时有效。

具体配置方式见下表。

表 40.3-2. LCD 数据格式控制 - 阶段一

GDMA 数据字节序列[1]	LCD_CAM_LCD_BYTE_MODE	LCD_CAM_LCD_BIT_ORDER	LCD_CAM_LCD_BYTE_ORDER[2]	阶段一输出数据序列[3,4]
B0,B1,B2,B3,B4,B5	0	0	0	{B0}{B1}{B2}{B3}{B4}{B5}
		1	0	{B0'}{B1'}{B2'}{B3'}{B4'}{B5'}
	1	0	0	{B1,B0}{B3,B2}{B5,B4}
			1	{B0,B1}{B2,B3}{B4,B5}
		1	0	{B1',B0'}{B3',B2'}{B5',B4'}
			1	{B0',B1'}{B2',B3'}{B4',B5'}
	2	0	0	{B2,B1,B0}{B5,B4,B3}
			1	{B0,B1,B2}{B3,B4,B5}
		1	0	{B2',B1',B0'}{B5',B4',B3'}
			1	{B0',B1',B2'}{B3',B4',B5'}

[1] GDMA 数据字节序列中的 B0 ~ B5 每一个代表一个字节的数据，靠左的数据对应低地址，即 B0 ~ B5 的地址为从低到高。

[2] 只有表格中的配置组合是有效的，所有表格所示之外的配置组合可能会导致不可预期的数据错误。

[3] 输出数据序列中，每个 {} 中的数据为大端并行数据，各个 {} 中的数据为串行关系，输出顺序为从左至右。以 {B0}{B1}{B2}{B3} 为例，{B0} 中的各个数据位之间为并行关系，而 {B0} 中的数据与 {B1}、{B2}、{B3} 的数据为串行关系。输出顺序为先输出 {B0}。

[4] 输出数据序列中，Bn'[7:0] = Bn[0:7]，其中 n = 0,1,2,3,4,5。

阶段二：数据色彩空间转换

在阶段二中，数据进行色彩空间转换，配置方式请参考章节 40.3.6。

> **说明：**
> 用户可以通过将 LCD_CAM_LCD_CONV_ENABLE 配置为 0 跳过阶段二，这种情况下，阶段一的输出将会直接成为阶段三的输入。

阶段三：数据后处理

在阶段三中，配置以下寄存器可以对将要输出的数据进行数据位顺序调整和字节重排列：

- LCD_CAM_LCD_WIRE_MODE
  - 0：LCD 输出到 GPIO 的数据位宽为 8 个比特
  - 1：LCD 输出到 GPIO 的数据位宽为 16 个比特
  - 2：LCD 输出到 GPIO 的数据位宽为 24 个比特
- LCD_CAM_LCD_DOUT_BIT_ORDER

  - 0：不反转
  - 1：反转输出数据每个字节的数据位顺序
- LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE
  - 0：不开启 LCD 输出数据字节重排列功能。
  - 1：开启 LCD 输出数据字节重排列功能，配置寄存器 LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE 以调整重排列模式，各个重排列模式的具体效果请参考表格 40.3-3。

表 40.3-3. LCD 输出数据重排列功能模式

阶段三输入数据字节序列[1]	LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE[2]	阶段三输出数据字节序列[3]
B0,B1,B2,B3,B4,B5	0	B1,B0,B3,B2,B5,B4
	1	B0,B2,B1,B3,B5,B4
	2	B1,B0,B2,B4,B3,B5
	3	B1,B2,B0,B4,B5,B3
	4	B2,B0,B1,B5,B3,B4
	5	B2,B1,B0,B5,B4,B3

[1] 输入数据字节序列中的 B0 ~ B5 每一个代表一个字节的数据，这表示把输入数据按照小端排列为字节序列。例如输入数据有效数据位宽为 16 比特时，原始输入数据序列 {B1,B0}{B3,B2}{B5,B4} 转换为输入数据字节序列后为 {B0,B1,B2,B3,B4,B5}。

[2] 只有表格中的配置值是有效的，所有表格所示之外的配置值可能会导致不可预期的数据错误。

[3] 最终发送到 GPIO 的发送数据序列可以由阶段三输出数据字节序列和发送数据位宽转换得到，例如阶段三输出数据字节序列为 B1,B0,B2,B4,B3,B5 时，若发送数据位宽为 16 比特，则发送数据序列为 {B0,B1}{B4,B2}{B5,B3}。其中，每个 {} 中的数据为大端并行数据，各个 {} 中的数据为串行关系，发送顺序为从左至右。在本例的发送数据序列中，{B0,B1} 中的各个数据位之间为并行关系，而 {B0,B1} 中的数据与 {B4,B2}、{B5,B3} 的数据为串行关系。发送顺序为先发送 {B0,B1}。

40.3.5.2 Camera 数据格式控制

在使用 Camera 接收数据时，配置以下寄存器可以调整送往 GDMA 的数据的数据位顺序和字节顺序：

- LCD_CAM_CAM_2BYTE_EN
  - 0：接收数据的位宽为 8 个比特
  - 1：接收数据的位宽为 16 个比特
- LCD_CAM_CAM_BIT_ORDER
  - 0：不反转
  - 1：反转数据位顺序：
    - 在 8 位模式下，CAM_DATA_in[7:0] 反转为 CAM_DATA_in[0:7]；
    - 在 16 位模式下，CAM_DATA_in[15:0] 反转为 CAM_DATA_in[0:15]。
- LCD_CAM_CAM_BYTE_ORDER
  - 0：不反转
  - 1：反转数据字节顺序。仅在 16 位模式下有效。

具体配置方式见下表。

表 40.3-4. Camera 数据格式控制

接收数据序列[1]	LCD_CAM_CAM_2BYTE_EN	LCD_CAM_CAM_BIT_ORDER[2]	LCD_CAM_CAM_BYTE_ORDER[2]	GDMA 数据序列[3,4]
{B0}{B1}{B2}{B3}	0	0	0	B0,B1,B2,B3
		1	0	B0',B1',B2',B3'
{B1,B0}{B3,B2}	1	0	0	B0,B1,B2,B3
			1	B1,B0,B3,B2
		1	0	B0',B1',B2',B3'
			1	B1',B0',B3',B2'

[1] 接收数据序列中，每个 {} 中的数据为大端并行数据，各个 {} 中的数据为串行关系，接收顺序为从左至右。以 {B0}{B1}{B2}{B3} 为例，{B0} 中的各个数据位之间为并行关系，而 {B0} 中的数据与 {B1}、{B2}、{B3} 的数据为串行关系。接收顺序为先接收 {B0}。

[2] 只有表格中的配置组合是有效的，所有表格所示之外的配置组合可能会导致不可预期的数据错误。

[3] GDMA 数据序列中的 B0 ~ B3 每一个代表一个字节的数据，靠左的数据对应低地址，即 B0 ~ B3 的地址为从低到高。

[4] GDMA 数据序列中，Bn'[7:0] = Bn[0:7]，其中 n = 0,1,2,3。

说明：
当接收序列为每次接收 1 个字节数据时，CAM_Data_in[7:0] 为有效数据，因此用户需要将 CAM_Data_in[7:0] 与主机相连。

40.3.6 YUV-RGB 数据格式转换

ESP32-P4 的 LCD_CAM 模块支持 YUV-RGB 数据格式互相转换。LCD 模块以及 Camera 模块各有一个数据格式转换模块，其功能包括：

- 支持在 BT601 和 BT709 两种标准下进行格式转换
- RGB565 格式到 YUV422/420/411 (full/limited range) 格式的转换
- YUV422/YUV411 (full/limited range) 格式到 RGB565 格式的转换
- YUV422/411 (full/limited range) 各种格式的转换

LCD 模块的 YUV-RGB 数据格式转换模块还支持：

- YUV422/411 (full/limited range) 格式到 RGB888 (full/limited range) 格式的转换
- RGB565 格式到 YUV444 (full/limited range) 格式的转换
- RGB888 (full/limited range) 格式与 RGB565 格式的转换
- YUV422/411 (full/limited range) 格式到 YUV444 (full/limited range) 格式的转换

40.3.6.1 YUV 数据流格式

在 ESP32-P4 LCD_CAM 模块中我们作以下规定：

假设 8 个依次待传输像素点对应的 YUV 数据为 $[Y_i, U_i, V_i]$ (i = 1 ~ 8)，则在：

- YUV422 模式下，LCD 依次发送或 Camera 依次接收如下数据：

Y_1	U_1	Y_2	V_2	Y_3	U_3	Y_4	V_4	Y_5	U_5	Y_6	V_6	Y_7	U_7	Y_8	V_8

- YUV420 模式下，LCD 和 CAM 不参与格式转换。
- YUV411 模式下，LCD 依次发送或 Camera 依次接收如下数据：

Y_1	U_1	Y_2	Y_3	V_3	Y_4	Y_5	U_5	Y_6	Y_7	V_7	Y_8

40.3.6.2 格式转换模块配置流程

Camera 模块中的格式转换模块与 LCD 模块中的格式转换模块几乎完全相同。因此，下文以 LCD 模块中的格式转换模块为例说明配置流程：

1. 置位 LCD_CAM_LCD_CONV_ENABLE 使能 YUV-RGB 格式转换模块；
2. 通过 LCD_CAM_LCD_CONV_MODE_8BITS_ON 配置输入数据的有效位宽：
   - 0：输入数据的有效数据位宽为 16 bit
   - 1：输入数据的有效数据位宽为 8 bit
3. 配置输出数据有效位宽：
   - 对于 Camera 模块中的格式转换模块：输出数据的有效数据位宽永远等于输入数据的有效数据位宽
   - 对于 LCD 模块中的格式转换模块，当 LCD_CAM_LCD_WIRE_MODE 的值为：
     - 2 时，格式转换模块的输出数据的有效数据位宽为 24 bit
     - 其他时，格式转换模块的输出数据的有效数据位宽等于输入数据的有效数据位宽
4. 通过 LCD_CAM_LCD_CONV_PROTOCOL_MODE 配置标准：
   - 0：使用 BT601 标准
   - 1：使用 BT709 标准
5. 配置转换模式：
   - 在 24 位输出时：

表 40.3-5. 24 位输出时的转换模式配置

转换模式	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]	RGB2RGB_MODE[4]
RGB565 → YUV444	1	-	3	-
RGB565 → RGB888	0	-	-	1
YUV422 → RGB888	0	0	3	-
YUV411 → RGB888	0	2	3	-
YUV422 → YUV444	1	0	0/1/2	-
YUV411 → YUV444	1	2	0/1/2	-

[1] 对应 LCD_CAM_LCD_CONV_TRANS_MODE 的值
[2] 对应 LCD_CAM_LCD_CONV_YUV_MODE 的值
[3] 对应 LCD_CAM_LCD_CONV_YUV2YUV_MODE 的值
[4] 对应 LCD_CAM_LCD_CONV_RGB2RGB_MODE 的值

- 在非 24 位输出时：

表 40.3-6. 非 24 位输出时的转换模式配置

转换模式	TRANS_MODE[1]	YUV_MODE[2]	YUV2YUV_MODE[3]	RGB2RGB_MODE[4]
RGB565 → YUV422	1	0	3	-
RGB565 → YUV420	1	1	3	-
RGB565 → YUV411	1	2	3	-
RGB888 → RGB565	0	-	-	1
YUV422 → RGB565	0	0	3	-
YUV411 → RGB565	0	2	3	-
YUV422 → YUV420	1	0	1	-
YUV422 → YUV411	1	0	2	-
YUV411 → YUV422	1	2	0	-
YUV411 → YUV420	1	2	1	-

[1] 对应 LCD_CAM_LCD_CONV_TRANS_MODE 的值
[2] 对应 LCD_CAM_LCD_CONV_YUV_MODE 的值
[3] 对应 LCD_CAM_LCD_CONV_YUV2YUV_MODE 的值
[4] 对应 LCD_CAM_LCD_CONV_RGB2RGB_MODE 的值

6. 配置输入数据色彩空间：配置 LCD_CAM_LCD_CONV_DATA_IN_MODE 为：
   - 0：有限色彩空间 (limited range)[1]
   - 1：全色彩空间 (full range)[2]
7. 配置输出数据色彩空间：配置 LCD_CAM_LCD_CONV_DATA_OUT_MODE 为：
   - 0：有限色彩空间 (limited range)[1]
   - 1：全色彩空间 (full range)[2]

说明：

1. 如果选择有限色彩空间，则
   - RGB 的取值范围为：16 ~ 240；
   - YUV 的取值范围为：
     - Y：16 ~ 240；
     - U-V：16 ~ 235。
2. 如果选择全色彩空间，则 RGB 或 YUV 的取值范围为：0 ~ 255；

40.3.7 LCD_CAM 时序

40.3.7.1 LCD 时序 (RGB 格式)

图 40.3-4 所示为 LCD RGB 格式的视频帧结构：

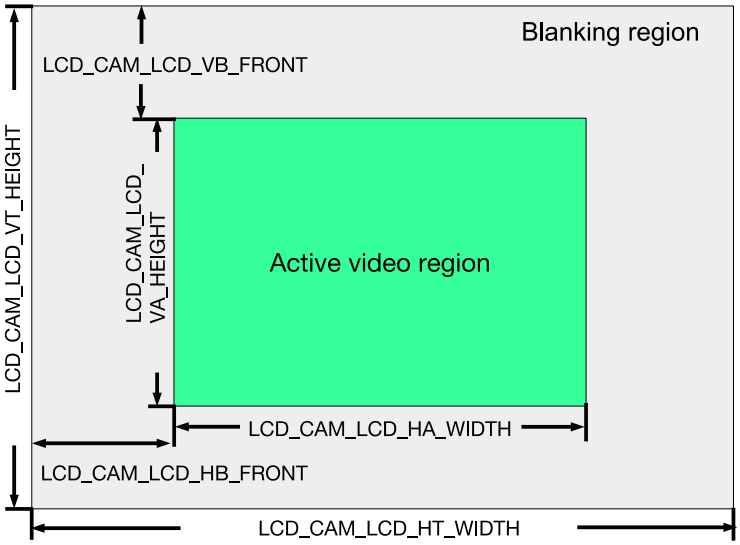


图 40.3-4. LCD 视频帧结构

如图所示，视频帧结构由以下寄存器配置：

- LCD_CAM_LCD_VT_HEIGHT
- LCD_CAM_LCD_VA_HEIGHT
- LCD_CAM_LCD_HB_FRONT
- LCD_CAM_LCD_HT_WIDTH
- LCD_CAM_LCD_HA_WIDTH
- LCD_CAM_VB_FRONT

图 40.3-5 显示了一个完整帧的时序。

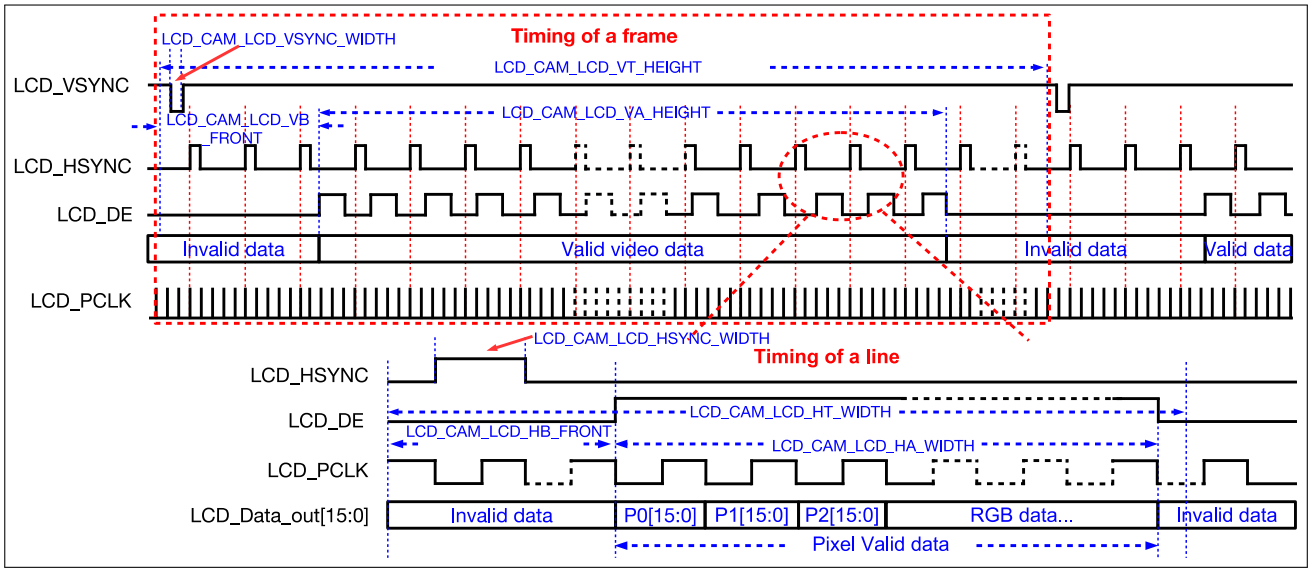


图 40.3-5. LCD 时序（RGB 格式）

说明：
配置上图所示的时序参数时，请特别注意参数值等于寄存器值 + 1。例如，如果希望 VSYNC 宽度为 1，则需要配置 LCD_CAM_LCD_VSYNC_WIDTH 为 0。更多信息见寄存器描述部分。

40.3.7.2 LCD 时序（I8080/MOTO6800 格式）

图 40.3-6 为 LCD I8080/MOTO6800 格式的时序图。

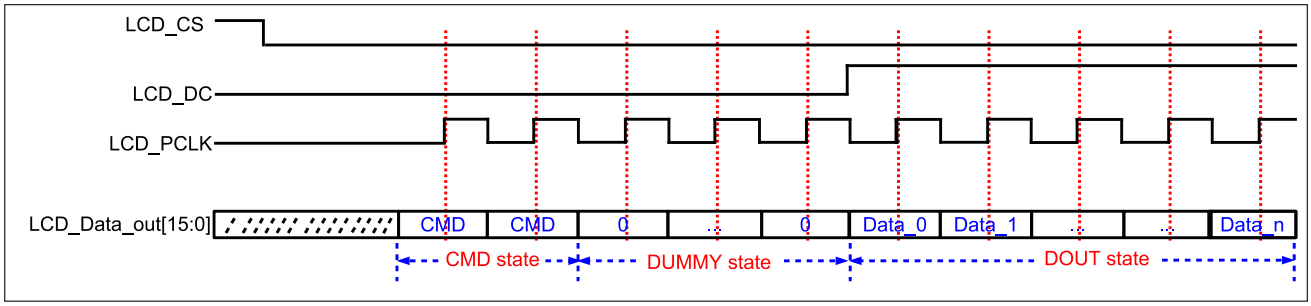


图 40.3-6. LCD 时序（I8080/MOTO6800 格式）

40.4 中断

ESP32-P4 的 LCD_CAM 可以生成以下中断信号并发送给中断矩阵。

- LCD_CAM_INTR

这个中断信号可由 LCD_CAM 的以下内部中断源生成：

- LCD_CAM_CAM_HS_INT：当 Camera 模块接收行数大于等于 LCD_CAM_CAM_LINE_INT_NUM + 1 时触发。
- LCD_CAM_CAM_VSYNC_INT：当 Camera 模块接收到帧指示信号 VSYNC（即接收完一帧数据）时触发。
- LCD_CAM_LCD_TRANS_DONE_INT：当 LCD 模块完成数据发送时触发。
- LCD_CAM_LCD_VSYNC_INT：当 LCD 模块发送帧指示信号 VSYNC（即发送完一帧数据）时触发。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 40.6 寄存器列表。

40.5 软件配置流程

说明：
LCD 模块和 Camera 模块的寄存器相关配置，均需要通过置位对应的 LCD_CAM_LCD_UPDATE 和 LCD_CAM_CAM_UPDATE 的方式来进行更新，从而将 LCD 寄存器数据从 APB 时钟域同步到 LCD/Camera 时钟域。见下文示例。

40.5.1 软件配置 LCD（RGB 格式）发送流程

软件配置 LCD 以 RGB 格式发送的流程如下：

1. 配置时钟，详见章节 40.3.3。
2. 配置信号管脚，详见表 40.3-1 和章节 10 *GPIO 交换矩阵和 IO MUX*。
3. 使能相应的中断，详见章节 40.4。
4. 置位 LCD_CAM_LCD_RGB_MODE_EN，使能 RGB 格式；
5. 配置帧结构，详见章节 40.3.7.1。
6. 置位 LCD_CAM_LCD_UPDATE 使配置生效。
7. 复位发送控制单元 LCD_Ctrl 和 Async TX FIFO，详见章节 40.3.4。
8. 配置 GDMA 发送链表，详见章节 7 *VDMA 控制器 (VDMA)*。
9. 开始发送数据：
   - 等待 LCD 从设备配置完成；
   - 置位 LCD_CAM_LCD_START 开始发送数据。
10. 等待步骤 3 设置的中断信号；
11. 在数据发送的帧间隔，检查 LCD_CAM_LCD_NEXT_FRAME_EN 是否被置位：
    - 如果被置位，则继续发送下一帧数据：置位 LCD_CAM_LCD_UPDATE，重复上述步骤；
    - 如果没有被置位，则 LCD 停止发送数据。
12. 若数据发送完成，则清零 LCD_CAM_LCD_START。

40.5.2 软件配置 LCD（I8080/MOTO6800 格式）发送流程

软件配置 LCD 以 I8080/MOTO6800 格式发送的流程如下：

1. 配置时钟，详见章节 40.3.3。
2. 配置信号管脚，详见表 40.3-1 和章节 10 *GPIO 交换矩阵和 IO MUX*。
3. 使能相应的中断，详见章节 40.4。
4. 清零 LCD_CAM_LCD_RGB_MODE_EN 关闭 RGB 格式。
5. 配置 CMD 阶段，包括 LCD_CAM_LCD_CMD、LCD_CAM_LCD_CMD_2_CYCLE_EN、LCD_CAM_LCD_FIRST_CMD_VALUE 和 LCD_CAM_LCD_LATTER_CMD_VALUE。
6. 配置 DUMMY 阶段，包括 LCD_CAM_LCD_DUMMY、LCD_CAM_LCD_DUMMY_CYCLELEN。
7. 配置 DOUT 阶段，包括 LCD_CAM_LCD_DOUT，根据输出模式不同进行以下配置：
   - 定长输出模式[a]，在 LCD_CAM_LCD_DOUT_CYCLELEN 中配置数据长度。
   - 持续输出模式[b]，则需置位 LCD_CAM_LCD_ALWAYS_OUT_EN 使能持续输出模式，不需要配置 LCD_CAM_LCD_DOUT_CYCLELEN。

说明：

(a) 定长输出模式是指：LCD 发送完成 LCD_CAM_LCD_DOUT_CYCLELEN 中配置的数据长度即结束 LCD 发送。

(b) 持续输出模式是指：LCD 持续发送数据，直到

i. LCD_CAM_LCD_START 被清零；

ii. 或 LCD_CAM_LCD_RESET 被置位；

iii. 或 GDMA 中所有数据被发送完。

8. 配置 CD 信号模式，包括 CD 信号的默认值以及在各阶段的值，详见寄存器 LCD_CAM_LCD_MISC_REG 描述。
9. 置位 LCD_CAM_LCD_UPDATE 使配置生效。
10. 复位发送控制单元 (LCD_Ctrl) 和 Async TX FIFO，详见章节 40.3.4。
11. 配置 GDMA 发送链表，详见章节 7 *VDMA 控制器 (VDMA)*。
12. 开始发送数据：
    - 等待 LCD 从设备配置完成；
    - 置位 LCD_CAM_LCD_START 开始发送数据。
13. 等待步骤 3 设置的中断信号；
14. 若数据发送完成，则清零 LCD_CAM_LCD_START。

注意：

无论 LCD 配置为 RGB 格式，还是 I8080/MOTO6800 格式，访问内部和外部存储器时均应注意以下限制：

- 如果 LCD 数据总线设置为 8 位并行输出时，则
  - 像素时钟 LCD_PCLK 频率需小于 80 MHz；
  - 如果同时使用了 YUV-RGB 格式转换，则 LCD_PCLK 频率需小于 60 MHz。
- 如果 LCD 数据总线设置为 16 位并行输出时，则
  - 像素时钟频率需小于 40 MHz；
  - 如果同时使用了 YUV-RGB 格式转换，则 LCD_PCLK 频率需小于 30 MHz。

40.5.3 软件配置 Camera 接收流程

软件配置 Camera 接收模式的流程如下：

1. 根据章节 40.3.3 的描述，配置时钟。注意，在从机模式下，模块时钟频率需要大于 2 倍的图形传感器的 PCLK 频率。
2. 配置信号管脚，详见表 40.3-1 和章节 10 *GPIO 交换矩阵和 IO MUX*。
3. 根据控制信号 HSYNC 置位或清零 LCD_CAM_CAM_VH_DE_MODE_EN。
4. 配置接收数据格式，详见章节 40.3.5.2。
5. 置位 LCD_CAM_CAM_UPDATE 使配置生效。

6. 复位接收控制单元 Camera_Ctrl 和 Async Rx FIFO，详见章节 40.3.4。
7. 使能相应的中断，详见章节 40.4。
8. 配置 GDMA 接收链表，并在 LCD_CAM_CAM_REC_DATA_BYTELEN 中配置接收数据长度。
9. 开始接收数据：
   - 在主机模式下，等待从机准备好后，置位 LCD_CAM_CAM_START 开始接收数据；
   - 在从机模式下，置位 LCD_CAM_CAM_START，等待主机提供时钟和控制信号后开始接收数据。
10. 接收数据，并存到 GDMA。最终产生步骤 7 中设置的中断。

注意：

- 无论 Camera 模块配置为主机模式还是从机模式，均应注意以下限制：
  - 如果选择 8 位并行输入，则
    * 像素时钟 CAM_PCLK 频率需小于 80 MHz；
    * 如果同时使用了 YUV-RGB 格式转换，则 CAM_PCLK 频率需小于 60 MHz。
  - 如果选择 16 位并行输入，则
    * CAM_PCLK 频率需小于 40 MHz；
    * 如果同时使用了 YUV-RGB 格式转换，则 CAM_PCLK 频率需小于 30 MHz。

说明：
对于 LCD_CAM 模块，如果同时外接了 LCD 和摄像头，则访问内部/外部存储器时，需保证接口上最大的数据吞吐率小于 GDMA 总数据带宽。此处默认 APB_CLK 频率为 80 MHz，更多有关 APB_CLK 的信息请见章节 11 复位和时钟。

40.6 寄存器列表

本小节的所有地址均为相对于 LCD_CAM 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

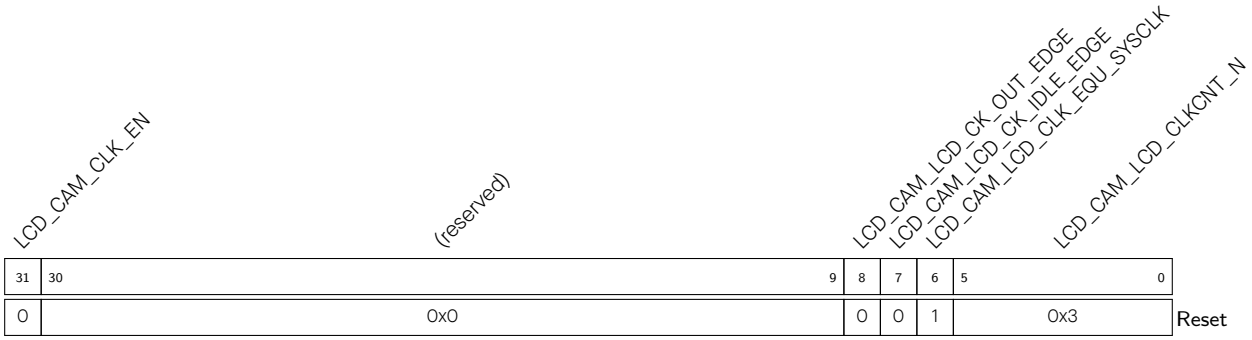
名称	描述	地址	访问
LCD 配置寄存器			
LCD_CAM_LCD_CLOCK_REG	LCD 时钟配置寄存器	0x0000	R/W
LCD_CAM_LCD_RGB_YUV_REG	LCD 数据格式转换配置寄存器	0x0010	R/W
LCD_CAM_LCD_USER_REG	LCD 用户配置寄存器	0x0014	varies
LCD_CAM_LCD_MISC_REG	LCD MISC 配置寄存器	0x0018	varies
LCD_CAM_LCD_CTRL_REG	LCD 信号配置寄存器	0x001C	R/W
LCD_CAM_LCD_CTRL1_REG	LCD 信号配置寄存器 1	0x0020	R/W
LCD_CAM_LCD_CTRL2_REG	LCD 信号配置寄存器 2	0x0024	R/W
LCD_CAM_LCD_FIRST_CMD_VAL_REG	LCD CMD 值配置寄存器	0x0028	R/W
LCD_CAM_LCD_LATTER_CMD_VAL_REG	LCD CMD 值配置寄存器	0x002C	R/W
LCD_CAM_LCD_DLY_MODE_CFG1_REG	LCD 数据/信号延迟模式配置寄存器	0x0030	R/W
LCD_CAM_LCD_DLY_MODE_CFG2_REG	LCD 数据延迟模式配置寄存器	0x0038	R/W
Camera 配置寄存器			
LCD_CAM_CAM_CTRL_REG	Camera 时钟配置寄存器	0x0004	R/W
LCD_CAM_CAM_CTRL1_REG	Camera 控制寄存器	0x0008	varies
LCD_CAM_CAM_RGB_YUV_REG	Camera 数据格式转换配置寄存器	0x000C	R/W
中断寄存器			
LCD_CAM_LC_DMA_INT_ENA_REG	LCD_CAM GDMA 中断使能寄存器	0x0064	R/W
LCD_CAM_LC_DMA_INT_RAW_REG	LCD_CAM GDMA 原始中断寄存器	0x0068	RO
LCD_CAM_LC_DMA_INT_ST_REG	LCD_CAM GDMA 屏蔽中断状态寄存器	0x006C	RO
LCD_CAM_LC_DMA_INT_CLR_REG	LCD_CAM GDMA 中断清除寄存器	0x0070	WO
版本寄存器			
LCD_CAM_LC_DATE_REG	版本控制寄存器	0x00FC	R/W

40.7 寄存器

本小节的所有地址均为相对于 LCD_CAM 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 40.1. LCD_CAM_LCD_CLOCK_REG (0x0000)



LCD_CAM_LCD_CLKCNT_N 与 LCD_CAM_LCD_CLK_EQU_SYSCLK 共同决定 LCD 模块的像素时钟 LCD_PCLK 的频率。

LCD_CAM_LCD_CLK_EQU_SYSCLK = 0 时，

$f_{LCD_PCLK} = f_{LCD_CLK}/(LCD_CAM_LCD_CLKCNT_N + 1)$。

注意，该字段不可配置为 0。

(R/W)

LCD_CAM_LCD_CLK_EQU_SYSCLK 与 LCD_CAM_LCD_CLK_EQU_SYSCLK 共同决定 LCD 模块的像素时钟 LCD_PCLK 的频率。

0：$f_{LCD_PCLK} = f_{LCD_CLK}/(LCD_CAM_LCD_CLKCNT_N + 1)$。

1：$f_{LCD_PCLK} = f_{LCD_CLK}$

(R/W)

LCD_CAM_LCD_CK_IDLE_EDGE 指示空闲状态下，LCD_PCLK 的电平。

0：低电平

1：高电平

(R/W)

LCD_CAM_LCD_CK_OUT_EDGE 指示前半个时钟周期内，LCD_PCLK 的电平。

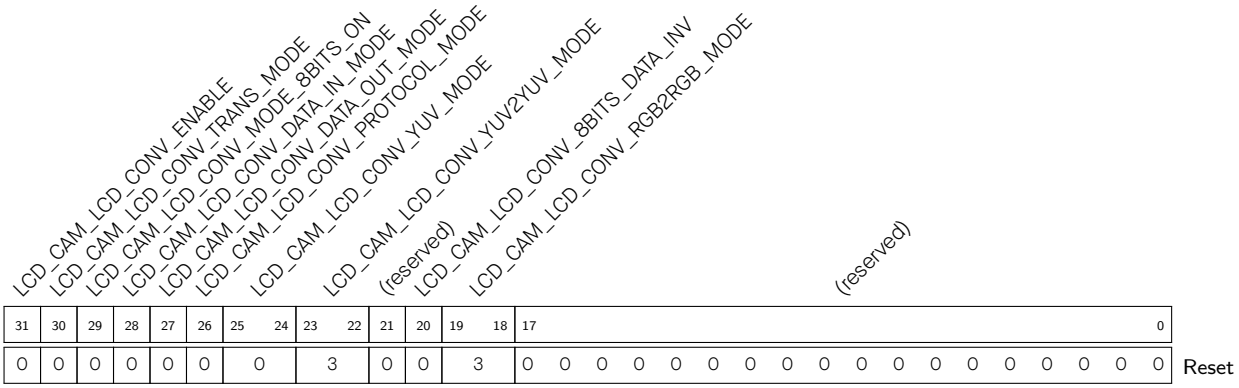
0：低电平

1：高电平

(R/W)

LCD_CAM_CLK_EN 置位此位，则强制打开所有配置寄存器的时钟，不再使用时钟门控。(R/W)

Register 40.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)



LCD_CAM_LCD_CONV_RGB2RGB_MODE 配置 RGB-to-RGB 模式下的数据转换格式。

0：数据被转换为 RGB565 格式

1：数据被转换为 RGB888 格式

2/3：不启用格式转换

(R/W)

LCD_CAM_LCD_CONV_8BITS_DATA_INV 配置是否交换两个字节的 GDMA 输入数据。

0：不交换

1：交换

(R/W)

LCD_CAM_LCD_CONV_YUV2YUV_MODE 配置 YUV-to-YUV 模式下的数据转换格式。

0：数据被转换为 YUV422 格式

1：数据被转换为 YUV420 格式

2：数据被转换为 YUV411 格式

3：不启用格式转换

仅在 LCD_CAM_LCD_CONV_TRANS_MODE = 1 时有效。

(R/W)

LCD_CAM_LCD_CONV_YUV_MODE 配置 YUV-to-YUV 模式或 YUV-to-RGB 模式下，输入数据具体的 YUV 格式。

0：输入数据为 YUV422 格式

1：输入数据为 YUV420 格式

2：输入数据为 YUV411 格式

(R/W)

LCD_CAM_LCD_CONV_PROTOCOL_MODE 配置格式转换的标准。

0：选择 BT601

1：选择 BT709

(R/W)

LCD_CAM_LCD_CONV_DATA_OUT_MODE 配置输出数据的色彩空间。

0：有限色彩空间

1：全色彩空间

(R/W)

见下页...

Register 40.2. LCD_CAM_LCD_RGB_YUV_REG (0x0010)

接上页...

LCD_CAM_LCD_CONV_DATA_IN_MODE 配置输入数据的色彩空间。

0：有限色彩空间

1：全色彩空间

(R/W)

LCD_CAM_LCD_CONV_MODE_8BITS_ON 配置 YUV-RGB 格式转换模块输入数据的有效位宽。

0：16 位模式

1：8 位模式

(R/W)

LCD_CAM_LCD_CONV_TRANS_MODE 配置格式转换模式。

0：转换成 RGB 格式

1：转换成 YUV 格式

(R/W)

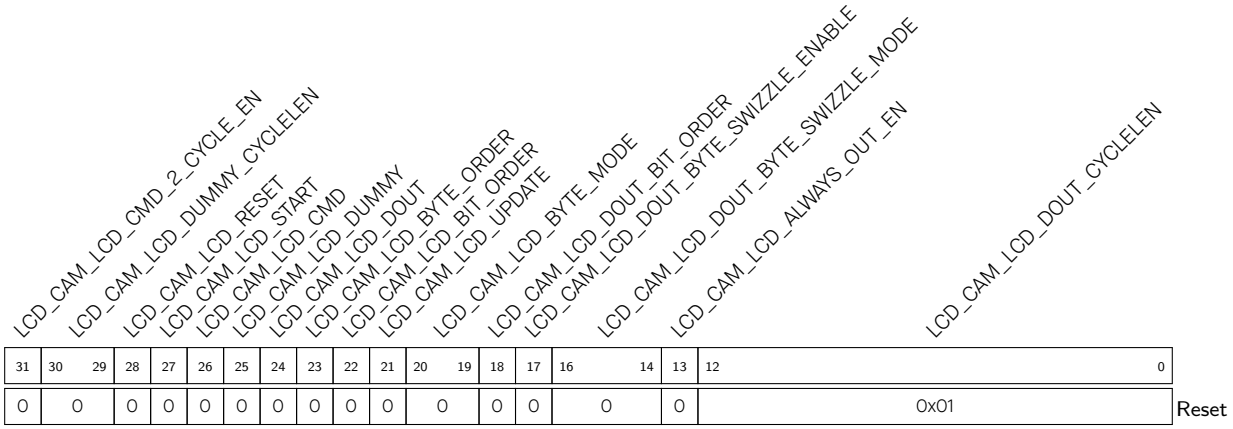
LCD_CAM_LCD_CONV_ENABLE 配置是否使能 YUV-RGB 格式转换模块。

0：旁路格式转换模块

1：使能格式转换模块

(R/W)

Register 40.3. LCD_CAM_LCD_USER_REG (0x0014)



LCD_CAM_LCD_DOUT_CYCLELEN 配置 LCD 模块输出数据阶段 (DOUT) 的周期时长。实际周期时长 = 本字段的值 + 1。(R/W)

LCD_CAM_LCD_ALWAYS_OUT_EN 使能 LCD 模块的持续输出模式。在该模式下，LCD 模块处于 DOUT 阶段时，持续输出数据，直至 LCD_CAM_LCD_START 被清除或 LCD_CAM_LCD_RESET 被置位。(R/W)

LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_MODE 配置 LCD 数据字节重排列模式，具体重排列方式请参考表格 40.3-3。(R/W)

LCD_CAM_LCD_DOUT_BYTE_SWIZZLE_ENABLE 配置是否开启 LCD 输出数据字节重排列功能。

0：不开启

1：开启

(R/W)

LCD_CAM_LCD_DOUT_BIT_ORDER 配置是否反转 LCD 输出数据的数据位顺序。

0：不反转

1：反转数据位顺序。在 n 位模式下，LCD_DATA_out[n-1:0] 反转为 LCD_DATA_out[0:n-1]。

(R/W)

LCD_CAM_LCD_BYTE_MODE 配置来自 GDMA 的数据的位宽。

0：8 位

1: 16 位

2: 24 位

(R/W)

LCD_CAM_LCD_UPDATE 配置是否更新 LCD 寄存器配置。

0：不更新

1：更新。将由硬件清除。

(R/W)

LCD_CAM_LCD_BIT_ORDER 配置是否反转 LCD 输入数据的数据位顺序。

0：不反转

1：反转数据位顺序。在 n 位模式下，LCD_DATA_in[n-1:0] 反转为 LCD_DATA_in[0:n-1]。

(R/W)

见下页...

Register 40.3. LCD_CAM_LCD_USER_REG (0x0014)

接上页...

LCD_CAM_LCD_BYTE_ORDER 配置是否反转 LCD 输入数据的字节序。

0：不反转

1：反转字节序。仅在 16 位和 24 位模式下有效。

(R/W)

LCD_CAM_LCD_DOUT 配置 LCD 发送数据的 DOUT 阶段。

0：禁用该功能

1：LCD 启动后，开始以 LCD RGB/I8080 模式发送数据

(R/W)

LCD_CAM_LCD_DUMMY 配置 LCD 发送数据的 DUMMY 阶段。

0：禁用该功能

1：LCD 启动后，使能 LCD 模式下的 DUMMY 阶段

(R/W)

LCD_CAM_LCD_CMD 配置 LCD 发送数据的 CMD 阶段。

0：禁用该功能

1：LCD 启动后，开始以 LCD 模式发送命令

(R/W)

LCD_CAM_LCD_START 置位则 LCD 开始发送数据。(R/W)

LCD_CAM_LCD_RESET LCD 模块复位寄存器。(WO)

LCD_CAM_LCD_DUMMY_CYCLELEN 设置 DUMMY 阶段的周期。实际周期 = 该字段的值 + 1。(R/W)

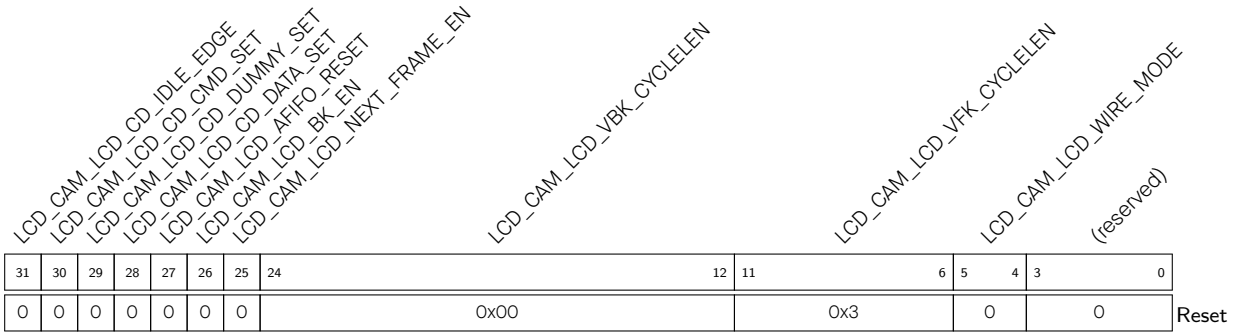
LCD_CAM_LCD_CMD_2_CYCLE_EN 设置 Command 阶段的周期。

0：一个周期

1：两个周期

(R/W)

Register 40.4. LCD_CAM_LCD_MISC_REG (0x0018)



LCD_CAM_LCD_WIRE_MODE 配置 LCD 输出到 GPIO 的数据位宽。

0: 8 位

1: 16 位

2: 24 位

(R/W)

LCD_CAM_LCD_VFK_CYCLELEN 在非 LCD RGB 模式下，设置建立 (Setup) 时钟周期。实际周期 = 该字段的值 + 1。(R/W)

LCD_CAM_LCD_VBK_CYCLELEN 在非 LCD RGB 模式下，设置保持 (Hold) 时钟周期。实际周期 = 该字段的值 + 1。(R/W)

LCD_CAM_LCD_NEXT_FRAME_EN 配置是否继续发送下一帧数据。

0：当前一帧数据发送完成后，LCD 停止工作。

1：当前一帧数据发送完成后，继续发送下一帧数据。

(R/W)

LCD_CAM_LCD_BK_EN 配置 LCD 发送数据时，是否允许有消隐区。

0：无消隐区

1：允许有消隐区

(R/W)

LCD_CAM_LCD_AFIFO_RESET 置位复位 Async TX FIFO。(WO)

LCD_CAM_LCD_CD_DATA_SET 配置 LCD I8080 模式下，处于 DOUT 阶段时，CD 线上的值。

0：LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1：LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_DUMMY_SET 配置 LCD I8080 模式下，处于 DUMMY 阶段时，CD 线上的值。

0：LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1：LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_CMD_SET 配置 LCD I8080 模式下，处于 CMD 阶段时，CD 线上的值。

0：LCD_CD = LCD_CAM_LCD_CD_IDLE_EDGE

1：LCD_CD = !LCD_CAM_LCD_CD_IDLE_EDGE

(R/W)

LCD_CAM_LCD_CD_IDLE_EDGE LCD_CD 的默认值。(R/W)

Register 40.5. LCD_CAM_LCD_CTRL_REG (0x001C)



LCD_CAM_LCD_HB_FRONT 配置 (HSYNC_POSITION + HSYNC_WIDTH + 水平后肩) 的值。(R/W)

LCD_CAM_LCD_VA_HEIGHT 配置帧有效垂直高度。(R/W)

LCD_CAM_LCD_VT_HEIGHT 配置帧总垂直高度。(R/W)

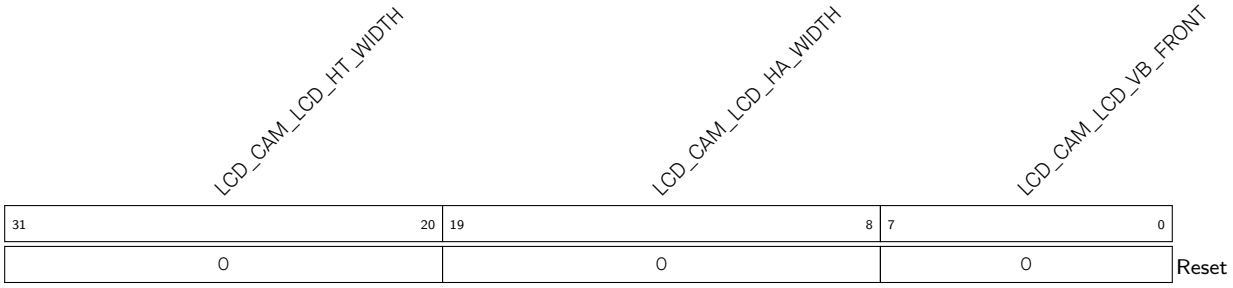
LCD_CAM_LCD_RGB_MODE_EN 配置是否使能 RGB 模式。

0：关闭 RGB 模式。

1：使能 RGB 模式，输入 VSYNC、HSYNC 和 DE 信号。

(R/W)

Register 40.6. LCD_CAM_LCD_CTRL1_REG (0x0020)

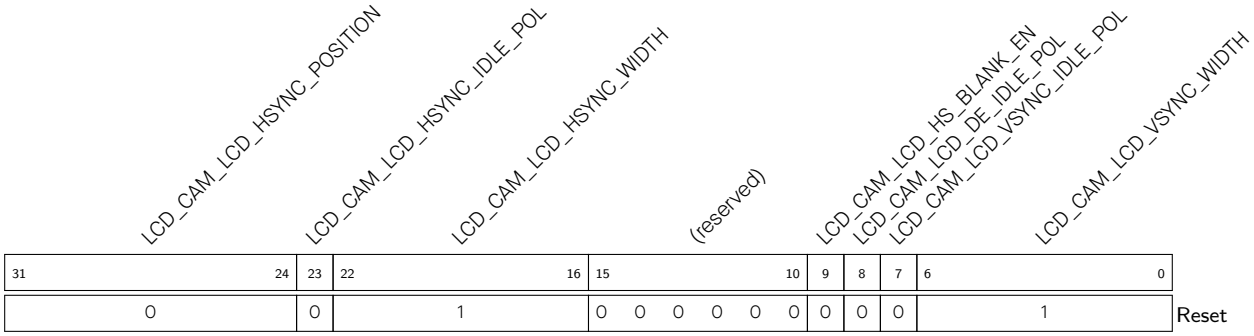


LCD_CAM_LCD_VB_FRONT 配置 (VSYNC_WIDTH + 垂直后肩) 的值。(R/W)

LCD_CAM_LCD_HA_WIDTH 配置帧有效水平宽度。(R/W)

LCD_CAM_LCD_HT_WIDTH 配置帧总水平宽度。(R/W)

Register 40.7. LCD_CAM_LCD_CTRL2_REG (0x0024)



LCD_CAM_LCD_VSYNC_WIDTH 配置 LCD_VSYNC 的有效脉冲宽度。参数值 = 寄存器值 + 1。(R/W)

LCD_CAM_LCD_VSYNC_IDLE_POL 配置 LCD_VSYNC 空闲值。

0：空闲值为低电平

1：空闲值为高电平

(R/W)

LCD_CAM_LCD_DE_IDLE_POL 配置 LCD_DE 空闲值。

0：空闲值为低电平

1：空闲值为高电平

(R/W)

LCD_CAM_LCD_HS_BLANK_EN 配置 RGB 模式下，LCD_HSYNC 的输出。

0：LCD_HSYNC 仅在有效的视频行输出。

1：LCD_HSYNC 可在场消隐行输出。

(R/W)

LCD_CAM_LCD_HSYNC_WIDTH 配置 LCD_HSYNC 有效脉冲宽度。参数值 = 寄存器值 + 1。(R/W)

LCD_CAM_LCD_HSYNC_IDLE_POL 配置 LCD_HSYNC 空闲值。

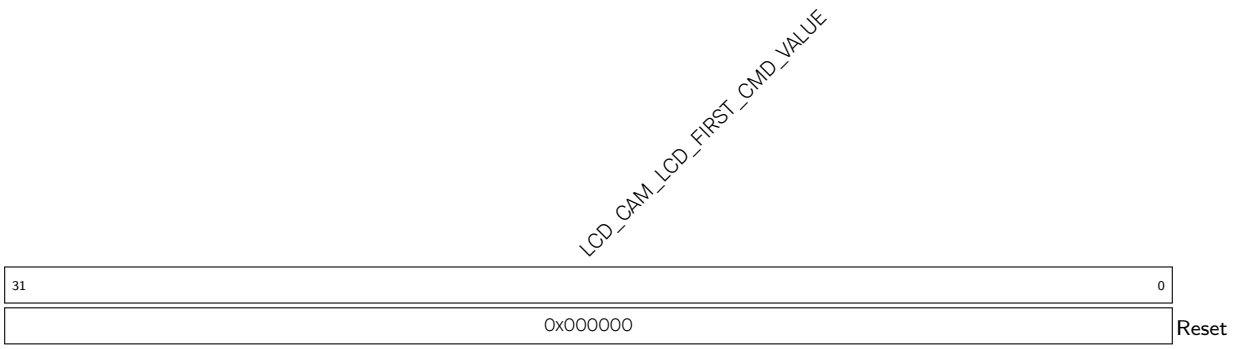
0：空闲值为低电平

1：空闲值为高电平

(R/W)

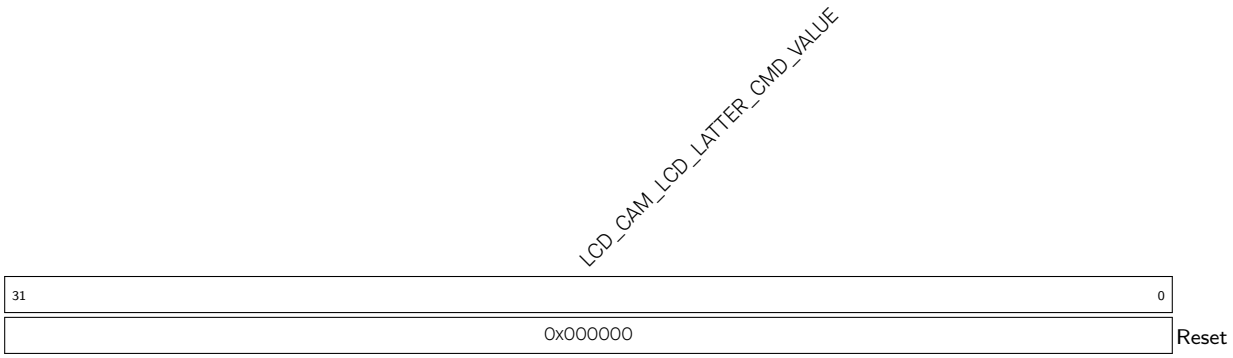
LCD_CAM_LCD_HSYNC_POSITION 配置 LCD_HSYNC 有效脉冲位置。参数值 = 寄存器值 + 1。单位为像素。(R/W)

Register 40.8. LCD_CAM_LCD_FIRST_CMD_VAL_REG (0x0028)



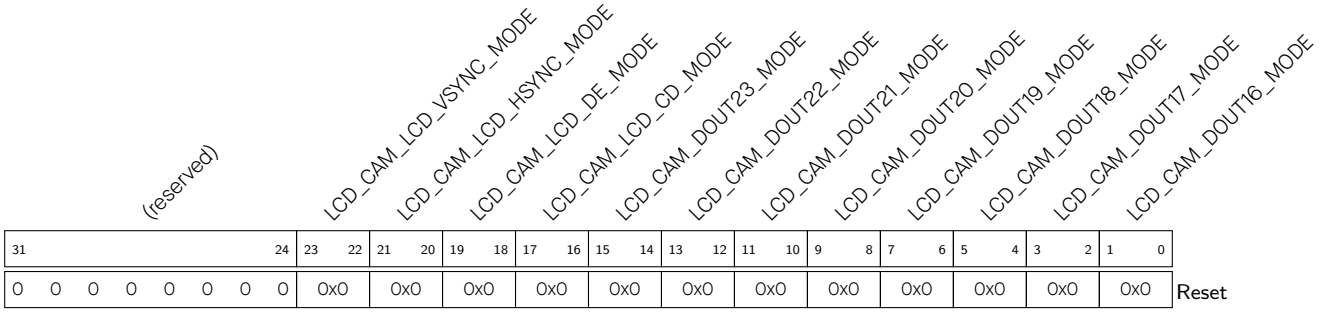
LCD_CAM_LCD_FIRST_CMD_VALUE 配置 LCD 处于 CMD 阶段第一个周期时，发送的命令值。(R/W)

Register 40.9. LCD_CAM_LCD_LATTER_CMD_VAL_REG (0x002C)



LCD_CAM_LCD_LATTER_CMD_VALUE 配置 LCD 处于 CMD 阶段第二个周期时，发送的命令值。(R/W)

Register 40.10. LCD_CAM_LCD_DLY_MODE_CFG1_REG (0x0030)



LCD_CAM_DOUT*n*_MODE (*n*: 16 - 23) 配置输出数据位 (*n*) 的延迟模式。

0：无延迟

1：在 LCD_CLK 上升沿延迟

2：在 LCD_CLK 下降沿延迟

(R/W)

LCD_CAM_LCD_CD_MODE 配置 LCD_CD 输出信号的延迟模式。

0：无延迟

1：在 LCD_CLK 上升沿延迟

2：在 LCD_CLK 下降沿延迟

(R/W)

LCD_CAM_LCD_DE_MODE 配置 LCD_DE 输出信号的延迟模式。

0：无延迟

1：在 LCD_CLK 上升沿延迟

2：在 LCD_CLK 下降沿延迟

(R/W)

LCD_CAM_LCD_HSYNC_MODE 配置 LCD_HSYNC 输出信号的延迟模式。

0：无延迟

1：在 LCD_CLK 上升沿延迟

2：在 LCD_CLK 下降沿延迟

(R/W)

LCD_CAM_LCD_VSYNC_MODE 配置 LCD_VSYNC 输出信号的延迟模式。

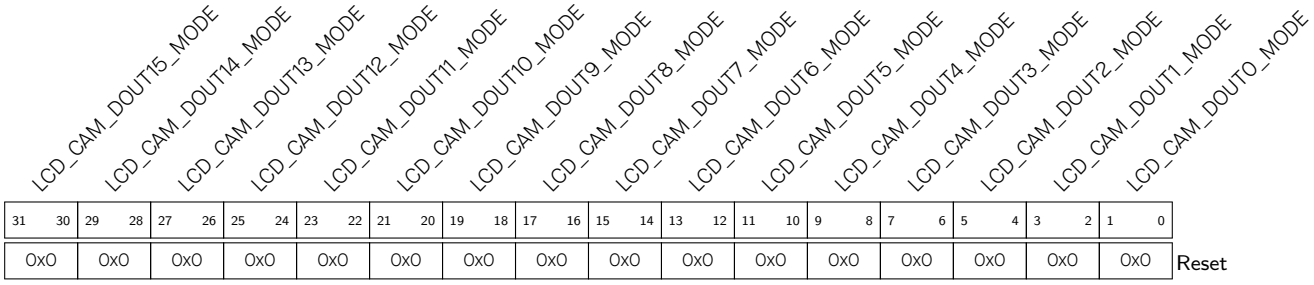
0：无延迟

1：在 LCD_CLK 上升沿延迟

2：在 LCD_CLK 下降沿延迟

(R/W)

Register 40.11. LCD_CAM_LCD_DLY_MODE_CFG2_REG (0x0038)



LCD_CAM_DOUT*n*_MODE (*n*: 0 - 15) 配置输出数据位 (*n*) 的延迟模式。

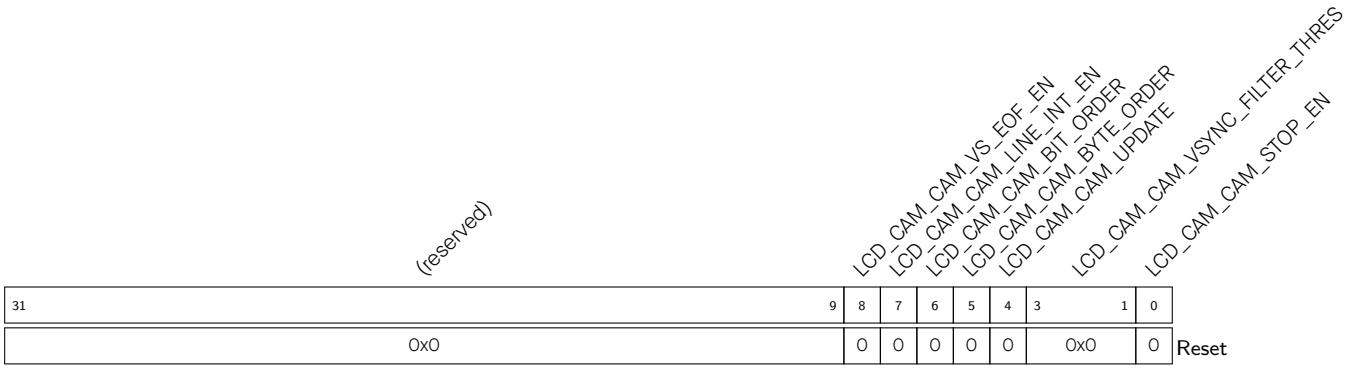
0：无延迟

1：在 LCD_CLK 上升沿延迟

2：在 LCD_CLK 下降沿延迟

(R/W)

Register 40.12. LCD_CAM_CAM_CTRL_REG (0x0004)



LCD_CAM_CAM_STOP_EN 配置是否使能 Camera 停止信号。

0：Camera 不停止工作。

1：GDMA RX FIFO 存满后，Camera 停止工作。

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_THRES 设置 CAM_VSYNC 信号的滤波阈值。(R/W)

LCD_CAM_CAM_UPDATE 配置是否更新 Camera 寄存器配置。

0：不更新

1：更新。将由硬件清除。

LCD_CAM_CAM_BYTE_ORDER 配置在 Camera 接收数据时是否反转数据字节顺序。

0：不反转

1：反转数据字节顺序。仅在 16 位模式下有效。

(R/W)

LCD_CAM_CAM_BIT_ORDER 配置在 Camera 接收数据时是否反转数据位顺序。

0：不反转

1：反转数据位顺序。在 8 位模式下，CAM_DATA_in[7:0] 反转为 CAM_DATA_in[0:7]；在 16 位模式下，CAM_DATA_in[15:0] 反转为 CAM_DATA_in[0:15]。

(R/W)

LCD_CAM_CAM_LINE_INT_EN 控制 LCD_CAM_CAM_HS_INT 中断原始值的生成。

0：不使能中断，即 LCD_CAM_CAM_HS_INT_RAW 无法生成。

1：使能中断，即 LCD_CAM_CAM_HS_INT_RAW 可以生成。

(R/W)

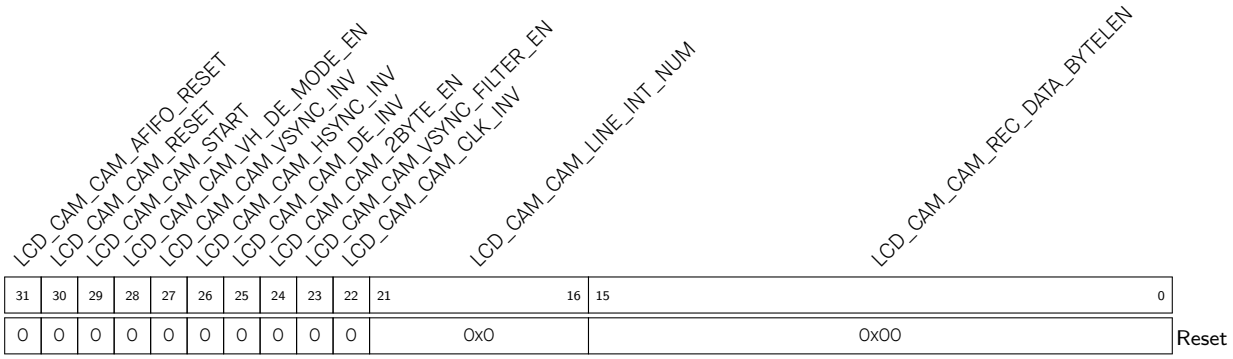
LCD_CAM_CAM_VS_EOF_EN 配置是否由 CAM_VSYNC 控制生成 in_suc_eof。

0：in_suc_eof 由 LCD_CAM_CAM_REC_DATA_BYTELEN 控制。

1：in_suc_eof 由 CAM_VSYNC 控制。

(R/W)

Register 40.13. LCD_CAM_CAM_CTRL1_REG (0x0008)



LCD_CAM_CAM_REC_DATA_BYTELEN 设置 Camera 触发 GDMA in_suc_eof_int 中断的接收数据字节长度。接收的数据长度 = 该字段的值 + 1。(R/W)

LCD_CAM_CAM_LINE_INT_NUM 设置触发 LCD_CAM_CAM_HS_INT 中断的视频行数。视频行数 = 该字段 + 1。(R/W)

LCD_CAM_CAM_CLK_INV 配置是否反转 CAM_PCLK 输入信号。

0：不反转

1：反转

(R/W)

LCD_CAM_CAM_VSYNC_FILTER_EN 配置是否使能 CAM_VSYNC 滤波功能。

0：不使用该功能

1：使能 CAM_VSYNC 滤波功能

(R/W)

LCD_CAM_CAM_2BYTE_EN 配置接收数据的位宽。

0：8 位

1：16 位

(R/W)

LCD_CAM_CAM_DE_INV 配置是否反转 CAM_DE 输入信号。

0：不反转

1：反转

(R/W)

LCD_CAM_CAM_HSYNC_INV 配置是否反转 CAM_HSYNC 输入信号。

0：不反转

1：反转

(R/W)

LCD_CAM_CAM_VSYNC_INV 配置是否反转 CAM_VSYNC 输入信号。

0：不反转

1：反转

(R/W)

见下页...

Register 40.13. LCD_CAM_CAM_CTRL1_REG (0x0008)

接上页...

LCD_CAM_CAM_VH_DE_MODE_EN 配置输入控制信号。

0：VSYNC、DE 线同时控制数据有效，HSYNC 线可以不接。但是在此模式下，Camera 的 YUV-RGB 转换功能将不可用。

1：VSYNC、HSYNC、DE 线同时控制数据有效，因此用户需要接 VSYNC、HSYNC、DE 信号线。(R/W)

LCD_CAM_CAM_START Camera 模块使能信号。(R/W)

LCD_CAM_CAM_RESET 置位复位 Camera 模块复位。(WO)

LCD_CAM_CAM_AFIFO_RESET Camera Async RX FIFO 复位信号。(WO)

Register 40.14. LCD_CAM_CAM_RGB_YUV_REG (0x000C)

31	30	29	28	27	26	25 24	23 22	21	20 0	
LCD_CAM_CAM_CONV_ENABLE	LCD_CAM_CAM_CONV_TRANS_MODE	LCD_CAM_CAM_CONV_MODE_8BITS_ON	LCD_CAM_CAM_CONV_DATA_IN_MODE	LCD_CAM_CAM_CONV_DATA_OUT_MODE	LCD_CAM_CAM_CONV_PROTOCOL_MODE	LCD_CAM_CAM_CONV_YUV_MODE	LCD_CAM_CAM_CONV_YUV2YUV_MODE	LCD_CAM_CAM_CONV_8BITS_DATA_INV	(reserved)	
0	0	0	0	0	0	0	3	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

LCD_CAM_CAM_CONV_8BITS_DATA_INV 配置是否反转两个字节的输入数据。

0：不交换

1：交换

(R/W)

LCD_CAM_CAM_CONV_YUV2YUV_MODE 配置 YUV-to-YUV 模式下的数据转换格式。

0：数据被转换为 YUV422 格式

1：数据被转换为 YUV420 格式

2：数据被转换为 YUV411 格式

3：不启用格式转换。

仅在 LCD_CAM_CAM_CONV_TRANS_MODE = 1 时有效。

(R/W)

LCD_CAM_CAM_CONV_YUV_MODE 配置在 YUV-to-YUV 模式下或 YUV-to-RGB 模式下，输入数据具体的 YUV 格式。

0：输入数据为 YUV422 格式

1：输入数据为 YUV420 格式

2：输入数据为 YUV411 格式

(R/W)

LCD_CAM_CAM_CONV_PROTOCOL_MODE 配置格式转换的标准。

0：选择 BT601

1：选择 BT709

(R/W)

LCD_CAM_CAM_CONV_DATA_OUT_MODE 配置 YUV-RGB 格式转换模块输出数据的色彩空间。

0：有限色彩空间

1：全色彩空间

(R/W)

LCD_CAM_CAM_CONV_DATA_IN_MODE 配置 YUV-RGB 格式转换模块输入数据的色彩空间。

0：有限色彩空间

1：全色彩空间

(R/W)

见下页...

Register 40.14. LCD_CAM_CAM_RGB_YUV_REG (0x000C)

接上页...

LCD_CAM_CAM_CONV_MODE_8BITS_ON 配置输入数据位宽。

0：16 位模式

1：8 位模式

(R/W)

LCD_CAM_CAM_CONV_TRANS_MODE 配置格式转换模式。

0：转换成 RGB 格式

1：转换成 YUV 格式

(R/W)

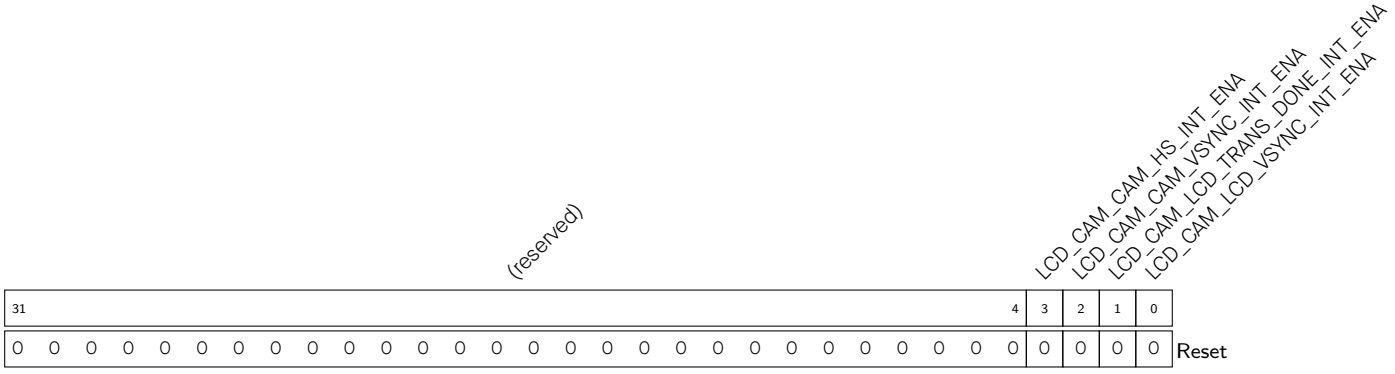
LCD_CAM_CAM_CONV_ENABLE 配置是否使能 YUV-RGB 格式转换模块。

0：旁路格式转换模块

1：使能格式转换模块

(R/W)

Register 40.15. LCD_CAM_LC_DMA_INT_ENA_REG (0x0064)



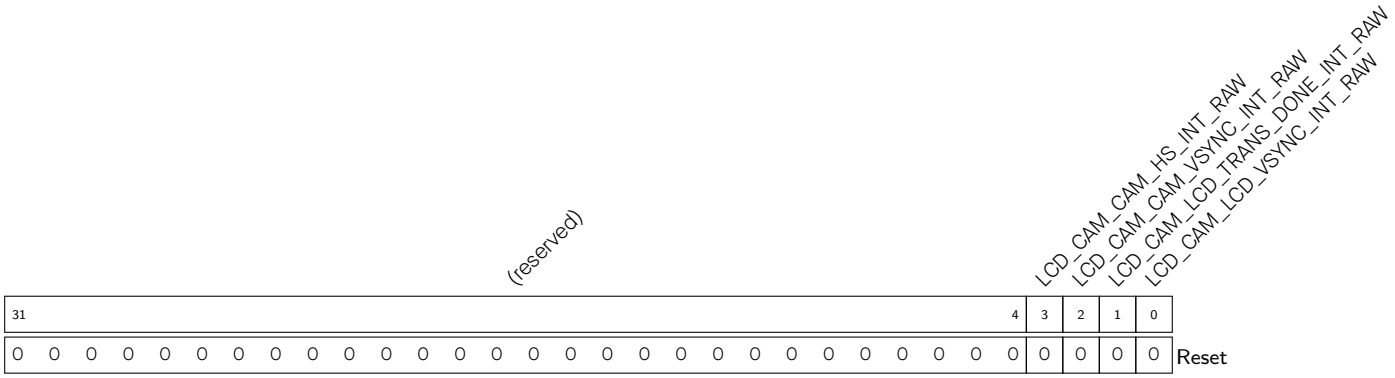
LCD_CAM_LCD_VSYNC_INT_ENA 写 1 使能 LCD_CAM_LCD_VSYNC_INT。(R/W)

LCD_CAM_LCD_TRANS_DONE_INT_ENA 写 1 使能 LCD_CAM_LCD_TRANS_DONE_INT。(R/W)

LCD_CAM_CAM_VSYNC_INT_ENA 写 1 使能 LCD_CAM_CAM_VSYNC_INT。(R/W)

LCD_CAM_CAM_HS_INT_ENA 写 1 使能 LCD_CAM_CAM_HS_INT。(R/W)

Register 40.16. LCD_CAM_LC_DMA_INT_RAW_REG (0x0068)



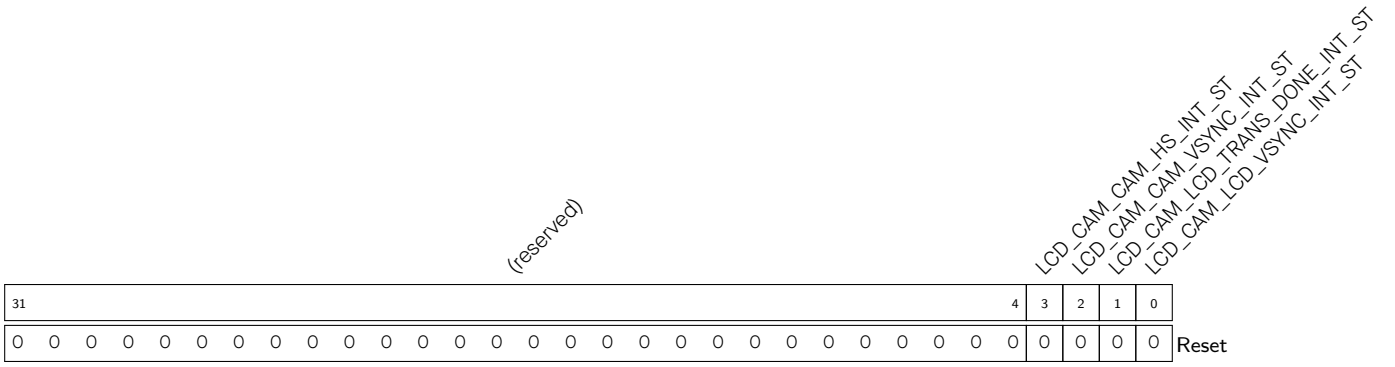
LCD_CAM_LCD_VSYNC_INT_RAW LCD_CAM_LCD_VSYNC_INT 的原始中断状态。(RO)

LCD_CAM_LCD_TRANS_DONE_INT_RAW LCD_CAM_LCD_TRANS_DONE_INT 的原始中断状态。(RO)

LCD_CAM_CAM_VSYNC_INT_RAW LCD_CAM_CAM_VSYNC_INT 的原始中断状态。(RO)

LCD_CAM_CAM_HS_INT_RAW LCD_CAM_CAM_HS_INT 的原始中断状态。(RO)

Register 40.17. LCD_CAM_LC_DMA_INT_ST_REG (0x006C)



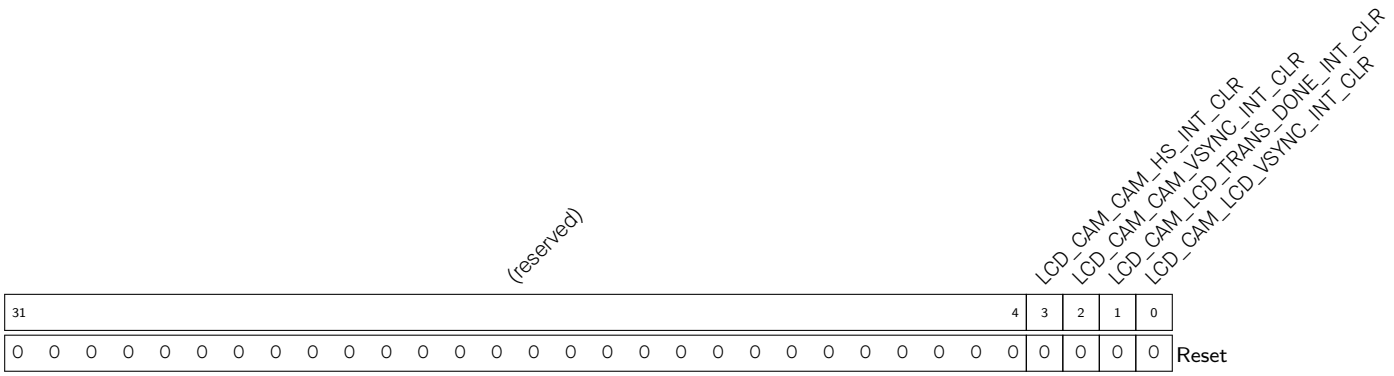
LCD_CAM_LCD_VSYNC_INT_ST LCD_CAM_LCD_VSYNC_INT 的屏蔽中断状态。(RO)

LCD_CAM_LCD_TRANS_DONE_INT_ST LCD_CAM_LCD_TRANS_DONE_INT 的屏蔽中断状态。(RO)

LCD_CAM_CAM_VSYNC_INT_ST LCD_CAM_CAM_VSYNC_INT 的屏蔽中断状态。(RO)

LCD_CAM_CAM_HS_INT_ST LCD_CAM_CAM_HS_INT 的屏蔽中断状态。(RO)

Register 40.18. LCD_CAM_LC_DMA_INT_CLR_REG (0x0070)



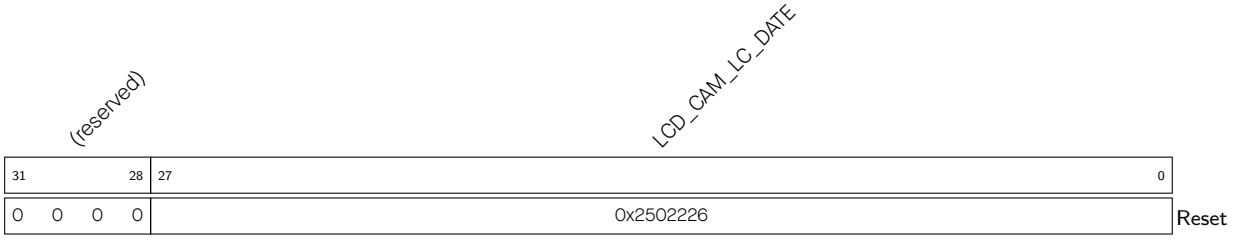
LCD_CAM_LCD_VSYNC_INT_CLR 写 1 清除 LCD_CAM_LCD_VSYNC_INT。(WO)

LCD_CAM_LCD_TRANS_DONE_INT_CLR 写 1 清除 LCD_CAM_LCD_TRANS_DONE_INT。(WO)

LCD_CAM_CAM_VSYNC_INT_CLR 写 1 清除 LCD_CAM_CAM_VSYNC_INT。(WO)

LCD_CAM_CAM_HS_INT_CLR 写 1 清除 LCD_CAM_CAM_HS_INT。(WO)

Register 40.19. LCD_CAM_LC_DATE_REG (0x00FC)



LCD_CAM_LC_DATE 版本控制寄存器。(R/W)

第 41 章

H264 编码器

41.1 概述

H264 编码器基于 H264 标准，用于视频序列的实时压缩。使用 H264 编码器可以显著减少数据总量，同时尽量减少视频质量的损失。

41.2 术语

比特流 表示一个或多个编码视频序列的比特序列。

配置文件 由 H264 标准规定的整个比特流语法的子集。

级别 在每个配置文件中指定，用于约束比特流中语法元素的值。

宏块 即 Macro Block，简称 MB。对于一个具有三个样本阵列的图像，其 MB 由一个 16 × 16 像素的亮度样本块和两个对应的色度样本块组成。

切片 每个切片包含整数个宏块。

I 帧 采用帧内编码的帧。

P 帧 采用帧间预测编码的帧。

I 切片 I 帧中的切片。

P 切片 P 帧中的切片。

P-skip 宏块 在 P 帧中跳过编码的宏块。

CAVLC 基于上下文的自适应可变长度编码。

MV 运动矢量。

GOP 图像组。在一个 GOP 中，第一帧为 I 帧，后续帧为 P 帧。

QP 量化参数。

ROI 感兴趣区域。

AXI 高级可扩展接口。

TX 发送。

RX 接收。

DB 去块滤波器。

NUM_H_MB 图像中水平宏块的数量。

41.3 特性

H264 编码器具有如下特性：

- 基于基线配置文件，支持级别 4.1。
- 编码视频色彩空间：8 位 YUV420。

- 原始视频色彩空间：RGB888/RGB565/8 位 YUV444/8 位 YUV422/8 位 YUV420（硬件可将其转换为 8 位 YUV420 用于编码）。
- 逐行视频，最大分辨率达 1920 × 1088 @ 30 fps。
- 支持 I 帧和 P 帧编码。
- 提供 GOP 和双流两种工作模式。在双流模式下，两个待编码的视频序列的总带宽不应超过 1920 × 1088 @ 30 fps。
- 帧内亮度宏块支持 4 × 4 和 16 × 16 的分割。
- 帧内预测中，4 × 4 的亮度宏块支持 9 种预测模式，16 × 16 的亮度宏块支持 4 种预测模式。
- 帧内预测中，色度宏块支持 4 种预测模式。
- 帧间预测中，亮度宏块支持多种分割大小：4 × 4、4 × 8、8 × 4、8 × 8、8 × 16、16 × 8、16 × 16。
- 运动估算精度达 1/2 和 1/4 像素。
- 水平运动搜索范围为 [-29.75, +16.75]，垂直运动搜索范围为 [-13.75, +13.75]。
- 配备去块滤波器。
- 使用 CAVLC 编码。
- 支持 P-skip 宏块。
- 支持在 P 片中包含 I 宏块。
- 支持亮度和色度分量量化结果降采样操作。
- 可配置固定 QP，并支持宏块级别的码率控制。
- 提供 MV 合并功能。当宏块的 MV 大于 0 时，此 MV 会输出到内存中。
- 支持 ROI 功能，最多可配置 8 个矩形 ROI，位置任意（允许重叠，优先级固定）。每个 ROI 可配置一个常量 QP 或 QP 偏移，非 ROI 区域可以配置 QP 偏移。
- 调试功能：可通过配置寄存器值替换原始视频像素数据、MV 合并结果数据、比特流数据、去块滤波中间数据和去块滤波结果数据。

41.4 架构

H264 编码器的架构如图 41.4-1 所示。

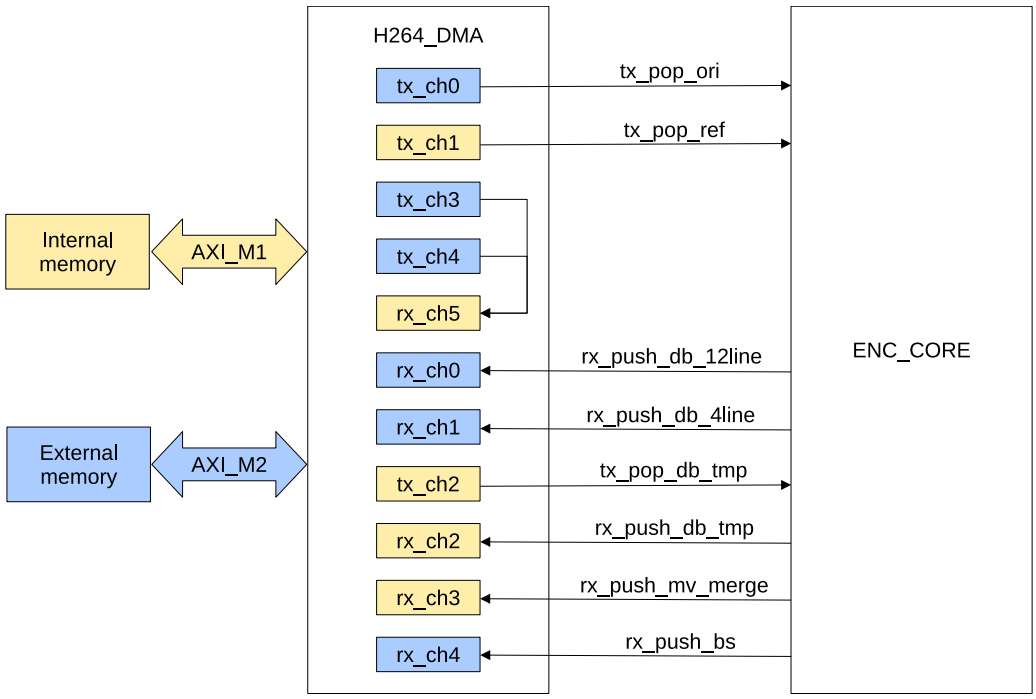


图 41.4-1. H264 编码器架构

H264 编码器由两部分组成：H264 专用 DMA (H264_DMA) 和编码器算法核 (ENC_CORE)。H264_DMA 用于数据传输，通过两组 AXI 总线与内部存储器和外部存储器连接，同时通过多个 DMA 发送/接收通道与 ENC_CORE 进行通信。具体而言，H264_DMA 配备 5 个 TX 通道 *n* 和 6 个 RX 通道 *n*，其中 TX 通道 3、TX 通道 4 和 RX 通道 5 为 H264_DMA 的内部通道，不与 ENC_CORE 连接。ENC_CORE 用于视频序列的压缩处理。

H264_DMA 的各个发送通道和接收通道功能如下：

- **TX 通道 0** 将原始图像发送到 ENC_CORE。
- **TX 通道 1** 将参考图像发送到 ENC_CORE。
- **TX 通道 2** 将去块滤波器中间数据发送到 ENC_CORE。
- **TX 通道 3** 和 **TX 通道 4** 将参考图像（即上一次去块滤波后的图像）发送到 **RX 通道 5**。
- **RX 通道 0** 和 **RX 通道 1** 将去块滤波器的结果（即参考图像）发送到外部存储器。
- **RX 通道 2** 将去块滤波器中间数据发送到内部存储器。
- **RX 通道 3** 将 MV 合并结果发送到内部存储器。
- **RX 通道 4** 将编码的比特流发送到外部存储器。
- **RX 通道 5** 将参考图像（即上一次去块滤波后的图像）发送到内部存储器。

根据描述可知，H264_DMA 通道可分为两类：

- 访问内部存储（通过 AXI_M1 总线）：TX 通道 1/2，RX 通道 2/3/5
- 访问外部存储（通过 AXI_M2 总线）：TX 通道 0/3/4，RX 通道 0/1/4

41.5 功能描述

41.5.1 编码器算法核 (ENC_CORE)

编码器算法核根据软件配置对视频序列进行压缩和编码。

41.5.1.1 架构

编码器算法核的架构如图 41.5-1 *ENC_CORE 架构* 所示。

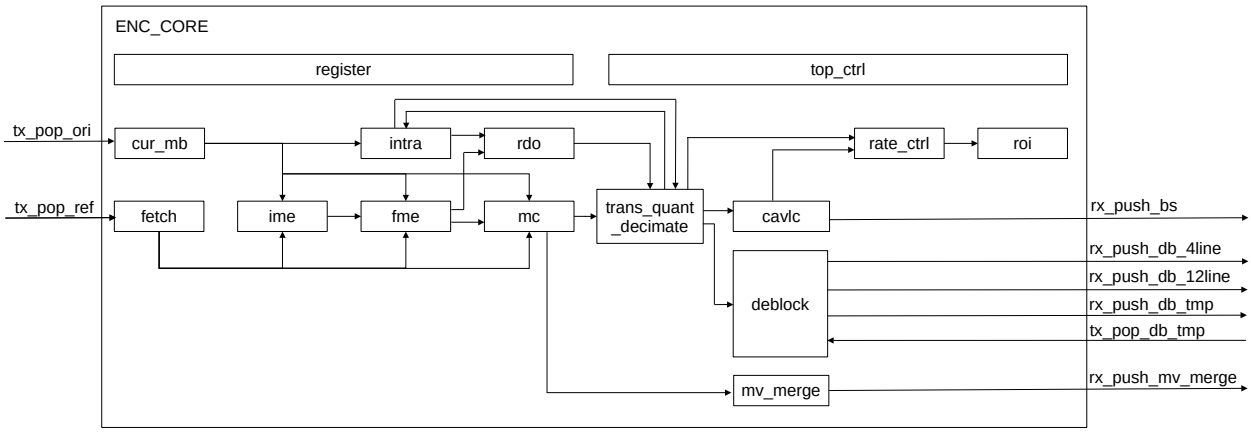


图 41.5-1. ENC_CORE 架构

每个模块的功能如下：

- register：选择和配置寄存器。
- top_ctrl：控制和调度编码器算法核的每个模块。
- cur_mb：从 H264_DMA 的 TX 通道 0 (tx_pop_ori) 中读取原始图像宏块，将其色彩空间转换为 8 位 YUV420，并分发给 intra、ime、fme 和 mc 模块。
- intra：使用 trans_quant_decimate 模块计算的重建结果对宏块进行帧内预测，将帧内预测的结果和残差发送给 trans_quant_decimate 模块，并将帧内预测成本发送给 rdo 模块。
- fetch：从 H264_DMA 的 TX 通道 1 (tx_pop_ref) 中读取搜索范围内的参考图像宏块，并将其分发给 ime、fme 和 mc 模块。
- ime：接收来自 cur_mb 模块的原始图像宏块，接收来自 fetch 模块的参考图像宏块，在搜索范围内进行整数运动估计，并将整数运动估计结果发送到 fme 模块。
- fme：接收来自 cur_mb 模块的原始图像宏块，接收来自 fetch 模块的参考图像宏块，接收来自 ime 模块的整数运动估计结果，并在整数运动估计的基础上进行分数运动估计。将分数运动估计结果发送到 mc 模块，并将帧间预测成本发送给 rdo 模块。
- mc：接收来自 cur_mb 模块的原始图像宏块，接收来自 fetch 模块的参考图像宏块，接收来自 fme 模块的分数运动估计结果，根据分数运动估计结果进行运动补偿。将帧间预测结果和残差之和发送到 trans_quant_decimate 模块，并将运动矢量发送到 mv_merge 模块。
- rdo：接收来自 intra 模块的帧内预测成本，接收来自 fme 模块的帧间预测成本，选择当前宏块的最终类型，支持 P 切片中的 I 宏块功能，并将最终的宏块类型发送到 trans_quant_decimate 模块。

- trans_quant_decimate：先接收来自 rdo 模块的最终宏块类型，接收来自 intra 模块的帧内预测和残差结果，接收来自 mc 模块的帧间预测和残差结果，然后进行 DCT、量化、反量化、反 DCT 和量化结果降采样，最后将残差信息发送到 rate_ctrl 模块，将降采样的量化结果发送到 cavlc 模块，并将重建结果发送到 deblock 模块。
- cavlc：接收来自 trans_quant_decimate 模块的量化结果和当前宏块的预测信息，进行上下文自适应变长编码（CAVLC），并将编码流写入 H264_DMA 的 RX 通道 4 (rx_push_bs) 中。
- deblock：接收来自 trans_quant_decimate 模块的重建结果和当前宏块的预测信息，进行去块滤波，并将去块滤波结果（即参考图像）写入 H264_DMA 的 RX 通道 0 (rx_push_db_12line) 和 RX 通道 1 (rx_push_db_4line) 中。此模块通过 H264_DMA 的 TX 通道 2（tx_pop_db_tmp）和 RX 通道 2 通道（rx_push_db_tmp）读写去块滤波的中间数据。
- mv_merge：接收来自 mc 模块的运动矢量 MV，进行 MV 合并，并将合并结果（宏块运动信息）写入 H264_DMA 的 RX 通道 3 (rx_push_mv_merge)。
- rate_ctrl：接收来自 trans_quant_decimate 模块的残差信息，接收来自 cavlc 模块的编码信息，以及 register 模块配置的帧级信息，进行宏块级码率控制，并将计算出的量化参数 QP 传递给 roi 模块。
- roi：接收来自 rate_ctrl 模块的量化参数，接收由 register 模块配置的 ROI 信息，调整 QP，并使用调整后的 QP 处理当前宏块。

41.5.1.2 原始视频色彩空间转换

由于原始视频色彩空间可以是 RGB888/RGB565/8-bit YUV444/8-bit YUV422/8-bit YUV420，而 ENC_CORE 只能编码 8-bit YUV420 视频，因此 ENC_CORE 需要将其转换为 8-bit YUV420。

转换步骤如下：

- 通过在低位补零转换为 RGB888（仅适用于 RGB565 原始视频；对于 RGB888/8-bit YUV444/8-bit YUV422/8-bit YUV420 原始视频，跳过此步骤）：

  RGB888_R={RGB565_R,3-bit 0}

  RGB888_G={RGB565_G,2-bit 0}

  RGB888_B={RGB565_B,3-bit 0}

- 通过以下公式转换为 8-bit YUV444（仅适用于 RGB565/RGB888 原始视频；对于 8-bit YUV444/8-bit YUV422/8-bit YUV420 原始视频，跳过此步骤）：

  YUV444_Y=(RGB888_R × 4899 + RGB888_G × 9617 + RGB888_B × 1868)»14

  YUV444_U=(128 × 16384 + RGB888_B × 8192 - RGB888_R × 2764 - RGB888_G × 5428)»14

  YUV444_V=(128 × 16384 + RGB888_R × 8192 - RGB888_G × 6860 - RGB888_B × 1332)»14

- 将 8-bit YUV444/8-bit YUV422 下采样为 8-bit YUV420（仅适用于 RGB565/RGB888/8-bit YUV444/8-bit YUV422 原始视频；对于 8-bit YUV420 原始视频，跳过此步骤）

原始视频色彩空间通过寄存器 H264_*x*_ORI_COLOR_SPACE 进行配置。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B 的寄存器。

41.5.1.3 量化结果降采样

ENC_CORE 根据每个宏块的最终量化结果的大小计算宏块的得分，然后将得分与可配置的阈值寄存器进行比较。如果低于阈值，量化结果将被设置为 0。否则，量化结果将保持不变，以进一步提高压缩比。对每个宏块的

亮度和色度分量独立执行上述过程。不同类型的宏块有不同的比较方法，如下所示：

- 亮度分量的 I 宏块，采用 16 × 16 分区：

  $$I_score_mb_cmp + H264_x_I16{\times}16_DECI_SCORE_OFFSET < H264_x_L_DECI_SCORE$$

  其中，I_score_mb_cmp 是 ENC_CORE 计算得出的当前宏块亮度分量的得分。H264_x_I16×16_DECI_SCORE_OFFSET 为可配置的偏移寄存器。H264_x_L_DECI_SCORE 是可配置的阈值寄存器。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。当满足以上条件时，当前亮度分量宏块的量化结果将被降采样为 0。

- 亮度分量 I 宏块，4 × 4 分区：当前亮度分量宏块的量化结果不会被减少为 0。

- 色度分量 I 宏块：

  $$c_score_mb_cmp + H264_x_I_chroma_DECI_SCORE_OFFSET < H264_x_C_DECI_SCORE$$

  其中，c_score_mb_cmp 是 ENC_CORE 计算的当前色度分量宏块的得分。H264_x_I_chroma_DECI_SCORE_OFFSET 是可配置的偏移寄存器。H264_x_C_DECI_SCORE 是可配置的阈值寄存器。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。当满足以上条件时，当前色度分量宏块的量化结果将被降采样为 0。

- 亮度分量 P 宏块:

  $$I_score_mb_cmp + H264_x_P16{\times}16_DECI_SCORE_OFFSET < H264_x_L_DECI_SCORE$$

  其中，I_score_mb_cmp 是由 ENC_CORE 计算的当前亮度分量宏块的得分。H264_x_P16×16_DECI_SCORE_OFFSET 是可配置的偏移寄存器。H264_x_L_DECI_SCORE 是可配置的阈值寄存器。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。当满足以上条件时，当前亮度分量宏块的量化结果将被降采样为 0。

- 色度分量 P 宏块:

  $$c_score_mb_cmp + H264_x_P_chroma_DECI_SCORE_OFFSET < H264_x_C_DECI_SCORE$$

  其中，c_score_mb_cmp 是由 ENC_CORE 计算的当前色度分量宏块的得分。H264_x_P_chroma_DECI_SCORE_OFFSET 是可配置的偏移寄存器。H264_x_C_DECI_SCORE 是可配置的阈值寄存器。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。当满足以上条件时，当前色度分量宏块的量化结果将被降采样为 0。

41.5.1.4 运动矢量 (MV) 合并

ENC_CORE 根据与 MV 合并相关的寄存器配置，合并亮度宏块的 MV，并将其写出到 H264_DMA 的 RX 通道 3 中 (rx_push_mv_merge)，最终存储在内部存储器中供上层应用使用，例如运动检测等。

与 MV 合并相关的寄存器包括：

- H264_x_MV_MERGE_EN：此寄存器控制 MV 合并功能的启用。将此寄存器设置为 1 表示启用相应视频序列的 MV 合并功能。将此寄存器设置为 0 表示关闭相应视频序列的 MV 合并功能。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。即使启用了 MV 合并功能，I 宏块也不会向存储器输出任何数据，因为 I 宏块没有 MV。

- H264_MV_MERGE_TYPE：此寄存器控制 MV 合并的类型。将此寄存器配置为 0，表示只保留最终类型为 P 宏块且亮度分量分区为 16 × 16 的宏块的 MV。将此寄存器配置为 1，表示只保留最终类型为 P 宏块的模最小的 MV。将此寄存器配置为 2，表示只保留最终类型为 P 宏块的模最大的 MV。3 是无效值。当寄存器 H264_x_MV_MERGE_EN 设置为 1 时，此寄存器的配置生效。

- H264_INT_MV_OUT_EN：此寄存器控制最终存储在内存中的合并 MV 的类型。将此寄存器设置为 0，表示如果保留的合并 MV 不为 0 时将被存储在内存中，否则将被丢弃。将此寄存器设置为 1，表示如果保留的合并 MV 的整数部分不为 0 时将被存储在内存中，否则将被丢弃。当寄存器 H264_*x*_MV_MERGE_EN 设置为 1 时，此寄存器的配置生效。
- H264_MB_VALID_NUM：此寄存器表示存储在内存中的当前图像 MV 合并结果的数量。一个宏块对应一个 MV 合并结果。当寄存器 H264_*x*_MV_MERGE_EN 设置为 1 时，此寄存器的值有意义，否则此寄存器的值无意义且保持为 0。

41.5.1.5 宏块级码率控制

ENC_CORE 根据码率控制相关的寄存器的配置、当前宏块的编码位数和残差的平均绝对值 (MAD)，计算出每个宏块的 QP 调整量。然后使用计算得出的 QP 调整量和寄存器中配置的帧级 QP，为每个宏块确定最终的 QP，从而实现编码位数的调整。

宏块级码率控制相关的寄存器包括：

- H264_*x*_MB_RATE_CTRL_EN：此寄存器控制是否启用宏块级码率控制。将此寄存器设置为 1，表示启用相应视频序列的宏块级码率控制。将此寄存器设置为 0，表示关闭相应视频序列的宏块级码率控制。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。注意，由于 I 帧使用恒定 QP 编码，因此即使启用了宏块级码率控制，I 帧中每个宏块的 QP 值也不会改变。
- H264_*x*_RATE_CTRL_U：此寄存器的配置值由软件中的帧级码率控制计算得出。ENC_CORE 需要此值来计算每个宏块的 QP 调整量。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。此寄存器配置值的计算方法属于软件中的帧级码率控制算法范畴，本章不再讨论。
- H264_*x*_MAD_FRAME_PRED：此寄存器是帧级 MAD 的预测值，由软件中的帧级码率控制进行配置。ENC_CORE 利用此值来计算每个宏块的 QP 调整量。*x*可以是 A 或 B，表示对应于视频序列 A 或 B。此寄存器配置值的计算方法属于软件中的帧级码率控制算法的范畴，本章不再讨论。
- H264_*x*_QP：此寄存器是帧级 QP 值，值范围是 10 到 51，由软件中的帧级码率控制进行配置。ENC_CORE 根据此值来调整每个宏块的 QP，从而实现宏块级别的码率控制。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。此寄存器的配置值的计算方法属于软件中的帧级码率控制算法的范畴，本章不再讨论。
- H264_*x*_QP_MAX：此寄存器用于限制宏块级码率控制最终计算的 QP 的最大值，值范围为 10 到 51。当宏块级码率控制最终计算的 QP 大于此寄存器配置值时，将被限制为此寄存器的配置值，否则保留原值。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。
- H264_*x*_QP_MIN：此寄存器用于限制宏块级码率控制最终计算的 QP 的最小值，值范围为 10 到 51。如果宏块级码率控制最终计算的 QP 小于此寄存器配置值，将被限制为此寄存器的配置值，否则保留原值。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。
- H264_FRAME_MAD_SUM：此寄存器表示当前帧中所有宏块的 MAD 累积和。软件需要在完成一帧视频的编码后读取此寄存器的值，进行帧级码率控制相关的计算。软件中的帧级码率控制算法超出了本章的范畴，不再讨论。
- H264_FRAME_ENC_BITS：此寄存器表示当前帧中所有宏块的编码位数的总和。软件需要在完成一帧视频的编码后读取此寄存器的值，进行帧级码率控制。软件中的帧级码率控制算法超出了本章的范畴，不再讨论。
- H264_FRAME_QP_SUM：此寄存器表示当前帧中所有宏块的 QP 总和。软件需要在完成一帧视频的编码后读取此寄存器的值，进行帧级码率控制。软件中的帧级码率控制算法超出了本章的范畴，不再讨论。

41.5.1.6 感兴趣区域 (ROI)

根据相关寄存器的配置，ENC_CORE 在一帧视频中最多支持 8 个可重叠的固定优先级矩形 ROI。通过调整 QP（由宏块级码率控制计算得出），每个 ROI 的编码图像质量各不相同。

与 ROI 相关的寄存器如下：

- H264_*x*_ROI_EN：此寄存器用于开启或关闭 ROI 功能。将其设置为 1 以启用 ROI 功能，设置为 0 禁用 ROI 功能。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。
- H264_*x*_ROI_MODE：此寄存器用于指定 ROI 的工作模式。如果配置为 0，ROI 将以固定 QP 模式工作。如果配置为 1，ROI 将以 QP 偏移模式工作。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。
- H264_*x*_ROI_REGION*n*_REG：此寄存器存储每个矩形 ROI 的启用和位置信息。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。*n* 可以是 0 到 7，表示 8 个 ROI。从 ROI 0 到 ROI 7，优先级依次降低。当存在重叠区域时，优先级最高的 ROI 的配置生效。此寄存器包含以下字段：
  - H264_*x*_ROI_REGION*n*_EN：设置为 1 以启用 ROI *n*，设置为 0 以关闭 ROI *n*。当寄存器 H264_*x*_ROI_EN 设置为 1 时，此寄存器可以控制 ROI *n* 的开关，否则此寄存器无效。
  - H264_*x*_ROI_REGION*n*_X：配置一个视频帧中 ROI *n* 的水平起始宏块坐标。
  - H264_*x*_ROI_REGION*n*_Y：配置一个视频帧中 ROI *n* 的垂直起始宏块坐标。
  - H264_*x*_ROI_REGION*n*_X_LEN：配置一个视频帧中 ROI *n* 的水平宏块数量。
  - H264_*x*_ROI_REGION*n*_Y_LEN：配置一个视频帧中 ROI *n* 的垂直宏块数量。
- H264_*x*_ROI_REGION*n*_QP：此寄存器用于配置每个矩形 ROI 中宏块的 QP 调整量。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。*n* 可以是 0 到 7，分别指代 8 个 ROI 之一。此寄存器的配置方法与 H264_*x*_ROI_MODE 寄存器的配置值有关：
  - 当 H264_*x*_ROI_MODE 设置为 0 时：表示 ROI 模式为固定 QP 模式。此时，H264_*x*_ROI_REGION*n*_QP 寄存器的值直接作为对应 ROI 中宏块的 QP 值，范围为 10 到 51。
  - 当 H264_*x*_ROI_MODE 设置为 1 时：表示 ROI 模式为 QP 偏移模式。此时，H264_*x*_ROI_REGION*n*_QP 寄存器的第 0 位到第 5 位表示 QP 偏移，范围为 0 到 51。第 6 位表示偏移的方向，0 表示正偏移，1 表示负偏移。
- H264_*x*_NO_ROI_REGION_QP：此寄存器用于配置 ROI 以外宏块的 QP 偏移。第 0 位到第 5 位表示 QP 偏移量，范围为 0 到 51。第 6 位表示偏移的方向，0 表示正偏移，1 表示负偏移。*x* 可以是 A 或 B，表示对应于视频序列 A 或 B。当 ROI 功能被禁用时，即 H264_*x*_ROI_EN 寄存器设置为 0 时，此寄存器不生效，ROI 以外宏块的 QP 不会改变。

注意，任何 ROI 调整后的 QP 值都不应小于 10。

无论宏块是在 ROI 中还是 ROI 以外，其最终调整的 QP 值都限制在 10 到 51 之间，以确保不超出 H264 标准范围。

41.5.1.7 调试用数据替换

为排查 ENC_CORE 处理流程中哪部分数据出错，可通过配置寄存器值替换原始视频像素数据、MV 合并结果数据、比特流数据、去块滤波中间数据和去块滤波结果数据（H264_*x*_DBG_REPLACE_ORI_DATA、H264_*x*_DBG_REPLACE_MV_MERGE_DATA、H264_*x*_DBG_REPLACE_WR_BS_DATA、H264_*x*_DBG_REPLACE_WR_DB_TEMP_DATA、H264_*x*_DBG_REPLACE_RD_DB_TEMP_DATA、H264_*x*_DBG_REPLACE_WR_DB_DATA、H264_*x*_DBG_REPLACE_REF_DATA）。

- 原始视频像素数据替换：将寄存器 H264_*x*_DBG_REPLACE_ORI_DATA_EN 设置为 1 以启用此功能。从 H264_DMA 读取的原始视频像素数据将被 H264_*x*_DBG_REPLACE_ORI_DATA 替换。
- MV 合并结果数据替换：将寄存器 H264_*x*_DBG_REPLACE_MV_MERGE_DATA_EN 设置为 1 以启用此功能。写入 H264_DMA 的 MV 合并数据将被 H264_*x*_DBG_REPLACE_MV_MERGE_DATA 替换。
- 比特流数据替换：将寄存器 H264_*x*_DBG_REPLACE_WR_BS_DATA_EN 设置为 1 以启用此功能。写入 H264_DMA 的比特流数据将被 H264_*x*_DBG_REPLACE_WR_BS_DATA 替换。
- 去块滤波中间数据替换：将寄存器 H264_*x*_DBG_REPLACE_WR_DB_TEMP_DATA_EN 设置为 1，以启用写入 H264_DMA 的去块滤波中间数据被 H264_*x*_DBG_REPLACE_WR_DB_TEMP_DATA 替换的功能。将寄存器 H264_*x*_DBG_REPLACE_RD_DB_TEMP_DATA_EN 设置为 1，以启用从 H264_DMA 读取的去块滤波中间数据被 H264_*x*_DBG_REPLACE_RD_DB_TEMP_DATA 替换的功能。
- 去块滤波结果数据替换：将寄存器 H264_*x*_DBG_REPLACE_WR_DB_DATA_EN 设置为 1，以启用写入 H264_DMA 的去块滤波结果数据被 H264_*x*_DBG_REPLACE_WR_DB_DATA 替换的功能。将寄存器 H264_*x*_DBG_REPLACE_REF_DATA_EN 设置为 1，以启用从 H264_DMA 读取的参考图像像素数据（即去块滤波结果数据）被 H264_*x*_DBG_REPLACE_REF_DATA 替换的功能。

41.5.2 H264 专用 DMA

H264 专用 DMA 能够根据软件配置，控制编码器算法核与内存之间的数据传输。

41.5.2.1 架构

H264_DMA 的架构如图 41.5-2 所示：

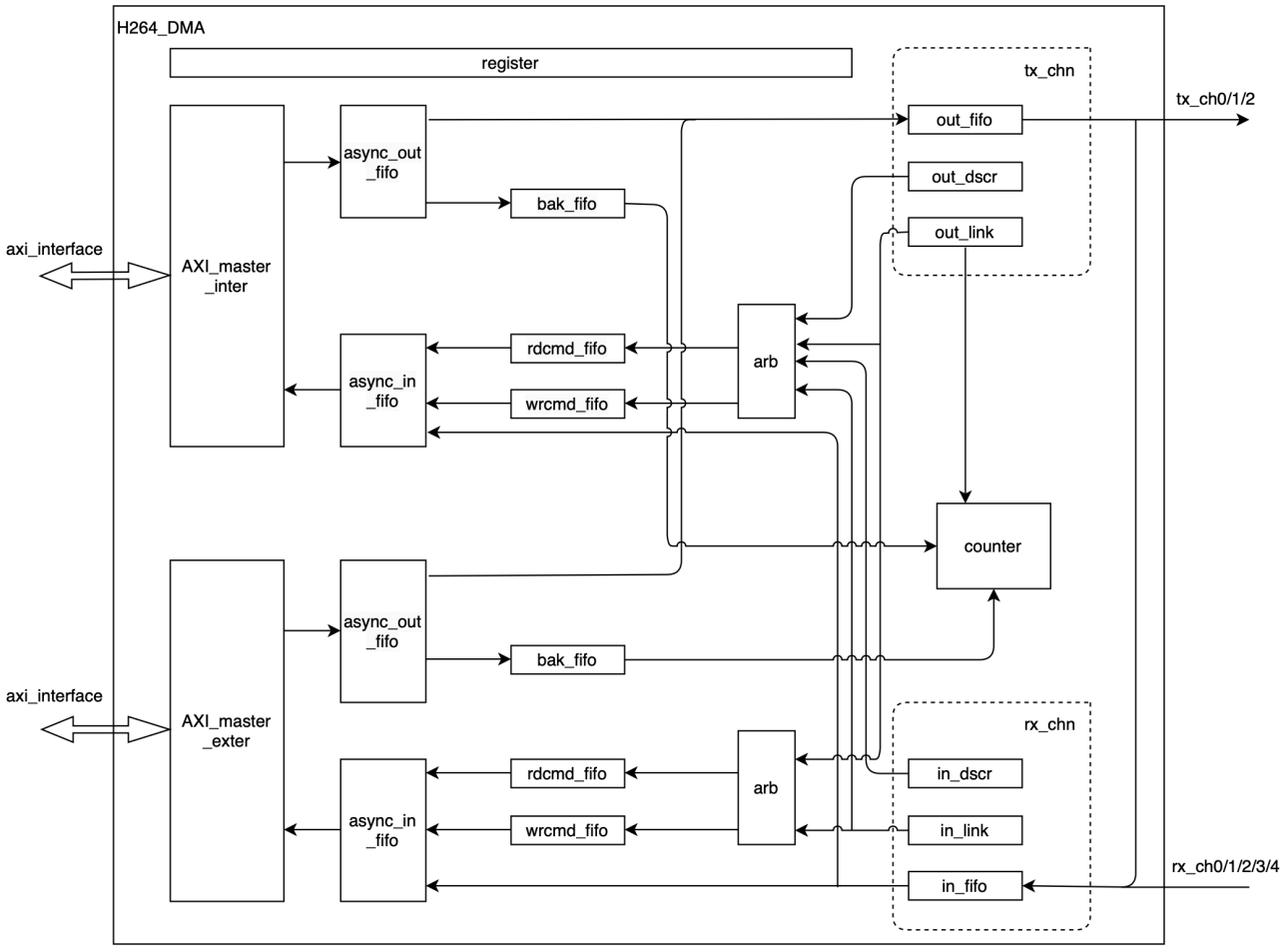


图 41.5-2. H264_DMA 架构

每个模块的功能如下：

- register：选择和配置寄存器。
- in_dscr：向内部存储器发起一个 AXI 读取请求，获取 RX 通道的链表描述符，并将描述符发送到相应 RX 通道的 in_link 模块。
- in_link：接收对应通道的 in_dscr 模块中获取的描述符，并将其拆分为多个 AXI 写数据请求，以引导相应的 in_fifo 模块将数据写入内部或外部存储器。RX 通道 0/1/4 用于将数据写入外部存储器，RX 通道 2/3/5 用于将数据写入内部存储器。
- out_dscr：向内部存储器发起一个 AXI 读取请求，获取 TX 通道的链表描述符，并将描述符发送到相应 TX 通道的 out_link 模块。
- out_link：接收对应通道的 out_dscr 模块中获取的描述符，并将其拆分为多个 AXI 读数据请求，来从内部或外部存储器获取数据。TX 通道 0/3/4 从外部存储器读取数据，TX 通道 1/2 从内部存储器读取数据。
- out_fifo：接收 AXI 读请求返回的数据，并等待 h264 或 RX 通道 5 读取。TX 通道 0/1/2 用于 h264 读取，TX 通道 3/4 用于 RX 通道 5 读取。
- arb：接收对外部存储器或内部存储器的读/写请求，并对这些读写请求进行仲裁。经过仲裁后，将相关请求分别发送到外部存储器的 rdcmd_fifo 和 wrcmd_fifo 模块，以及内部存储器的 rdcmd_fifo 和 wrcmd_fifo 模块。

- wrcmd_fifo：缓存由 arb 模块仲裁的 AXI 写请求，并将写请求和数据切割成多个 8 字节数据块，然后发送到 async_in_fifo 模块。
- rdcmd_fifo：缓存由 arb 模块仲裁的 AXI 读请求，并将读请求发送到 async_in_fifo 模块。
- bak_fifo：缓存由 arb 模块仲裁的读/写请求信息，包括通道 id、AXI_id、读/写请求数据长度，读/写请求请求源和读请求起始地址。该模块根据 async_out_fifo 模块返回的 AXI_id 找到请求的对应信息，并将请求的完成信息和 async_out_fifo 模块中读取的数据发送到相应通道。
- async_in_fifo：接收来自 wrcmd_fifo 模块和 rdcmd_fifo 模块的 AXI 请求，并将 H264_DMA 内核时钟域的请求同步到 AXI 时钟域，等待 AXI 主机读取。
- async_out_fifo：接收由 AXI_master_inter/AXI_master_exter 模块返回的读数据和读/写请求的完成信息，并将其从 AXI 时钟域同步到 H264_DMA 内核的时钟域。
- AXI_master_inter/AXI_master_exter：从 async_in_fifo 模块读取读/写请求，并将其转换为 AXI 总线协议格式输出。接收由 AXI 从机返回的读数据和读/写完成信息，并将其转换为内部私有接口格式，写入 async_out_fifo 模块。AXI_master_inter 模块用于访问内部存储空间，AXI_master_exter 模块用于访问外部存储空间。

41.5.2.2 链表描述符

H264_DMA 链表描述符如图 41.5-3 所示：

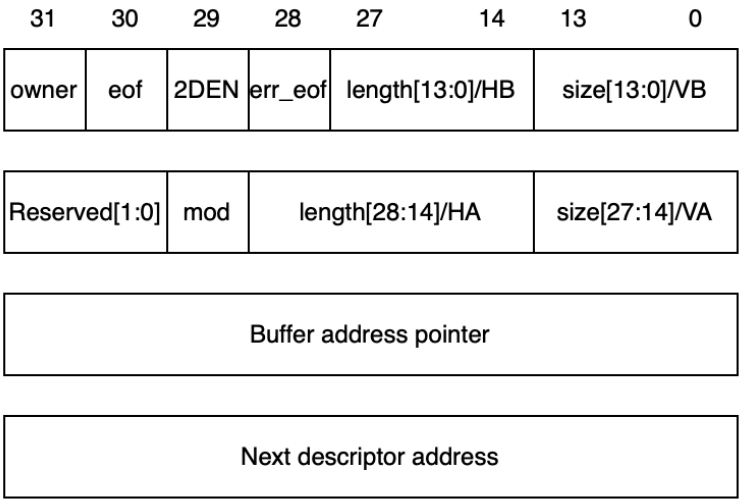


图 41.5-3. H264_DMA 链表描述符

- owner：表示与当前描述符对应的缓冲区允许的操作者。
  - 0：允许操作者为 CPU。
  - 1：允许操作者为 H264_DMA。H264_DMA 在使用描述符后，会清除此位。
  - 对发送描述符，由于发送描述符默认使用后不会写回，所以默认的 owner 位不会被清除。如果需要此描述符的 owner 位在使用后自动清除，需配置 H264_DMA_OUT_AUTO_WRBACK 启用写回功能。
- suc_eof：结束标志。
  - 0：当前描述符不是最后一个描述符。
  - 1：当前描述符是最后一个描述符。

- mod：数据块读取模式。
  - 0：H264_DMA 仅读取/写入一个水平宽度为 hb、垂直高度为 vb 的图像块一次。
  - 1：H264_DMA 读取/写入水平宽度为 hb、垂直高度为 vb 的图像块多次。直到读/写完具有 HA 水平宽度和 VA 垂直高度的图像数据。注意，hb/vb/HA/VA 的单位是像素。
  - 在 1D 模式下，此字段必须为 0。
- err_eof：接收结束错误标志。
  - 对于接收描述符，硬件在接收到一个数据包并检测到接收数据错误后，会将此位设置为 1。
  - 对于发送描述符，此位固定为 0。
- 2DEN：2D 功能使能信号。
  - 0：H264_DMA 的功能与一般 DMA 功能相同。此时，可以用来传输连续的数据段。
  - 1：启用 H264_DMA 功能，并根据配置的图像块进行数据传输。
- length[13:0]/HB：
  - 2DEN 为 0 时，表示传输数据的长度（最低 14 位）。对于数据接收，此字段在硬件接收到一帧数据后填充。对于数据传输，此字段表示要发送的数据长度，单位为字节。
  - 2DEN 为 1 时，表示图像块的水平宽度 (HB)，单位为像素。
- size[13:0]/VB：
  - 2DEN 为 0 时，表示为传输数据分配的内存空间大小（最低 14 位），单位为字节。
  - 2DEN 为 1 时，表示图像块的垂直高度 (VB)，单位是像素。
- length[28:14]/HA：
  - 2DEN 为 0 时，表示传输数据的长度（最高 14 位，只使用 14 ~ 27 位）。对于数据接收，此字段在硬件接收到一帧数据后填充。对于数据传输，此字段表示要发送的数据长度，单位是字节。
  - 2DEN 为 1 时，表示描述符对应图像的水平宽度 (HA)，单位是像素。
- size[27:14]/VA：
  - 2DEN 为 0 时，表示传输数据的内存空间大小（最高 14 位），单位是字节。
  - 2DEN 为 1 时，表示描述符对应图像的垂直高度 (VA)，单位是像素。
- Buffer address pointer：
  - 2DEN 为 0 时，表示 buffer 地址指针。
  - 2DEN 为 1 时，表示 buffer 中 HA × VA 图像块原点坐标的地址指针。
- Next descriptor address：下一个描述符地址的指针。

41.5.2.3 传输启动

H264_DMA 根据与通道链表描述符相关的寄存器配置，从内部存储器读取链表到本地，使得相应的通道可以用这些链表传输数据，从而启动相应的通道。

与链表描述符相关的寄存器包括：

- H264_DMA_OUTLINK/INLINK_START_CH*x*：此寄存器控制 H264_DMA 的相应通道是否开始获取描述符。将此寄存器设置为 1，表示相应的通道开始获取描述符。获取描述符后，硬件自动将此寄存器设置为 0，然后硬件自动获取后续的描述符。*x* 可以为 0、1、2、3、4，表示对应通道 0、1、2、3、4 的寄存器。
- H264_DMA_OUTLINK/INLINK_ADDR_CH*x*：此寄存器表示存储第一个描述符的地址，需要 8 字节对齐。

41.5.2.4 传输复位

为确保 H264_DMA 安全复位，防止通道卡住，H264_DMA 通道的复位须按照以下步骤进行：

1. 向相应通道的 H264_DMA_OUT/IN_CMD_DISABLE_CH*x* 寄存器写入 1。
2. 等待相应通道的 H264_DMA_OUT/IN_RESET_AVAIL_CH*x* 寄存器置位。
3. 向相应通道的 H264_DMA_OUT/IN_RST_CH*x* 寄存器写入 1。
4. 向相应通道的 H264_DMA_OUT/IN_RST_CH*x* 寄存器写入 0。

与通道复位相关的寄存器如下：

- H264_DMA_OUT/IN_CMD_DISABLE_CH*x*：此寄存器控制 H264_DMA 的相应通道是否停止向 AXI 发起读/写请求。将此寄存器设置为 1，表示相应通道停止向 AXI 发起读/写请求。将此寄存器设置为 0，表示相应通道可以向 AXI 发起读/写请求。*x* 在 TX 通道中可以是 0、1 和 2，表示对应于通道 0、1 和 2。在 RX 通道中，*x* 可以是 0、1、2、3、4 和 5，表示对应通道 0、1、2、3、4 和 5。
- H264_DMA_OUT/IN_RESET_AVAIL_CH*x*：此寄存器指示 H264_DMA 的相应通道是否可以安全复位。如果此寄存器为 1，表示相应的通道可以安全复位。如果此寄存器为 0，表示相应的通道无法安全复位。
- H264_DMA_OUT/IN_RST_CH*x*：此寄存器控制 H264_DMA 的相应通道是否执行软复位。将此寄存器设置为 1，表示相应的通道执行软复位。将此寄存器设置为 0，表示相应的通道不执行软复位。

注意：

- 由于 TX 通道 3、TX 通道 4 和 RX 通道 5 互相绑定，这三个通道只有一组 H264_DMA_OUT/IN_CMD_DISABLE_CH5、H264_DMA_OUT/IN_RESET_AVAIL_CH5 和 H264_DMA_OUT/IN_RST_CH5 寄存器。在复位这三个通道时，只需操作这一组寄存器，即可对这三个 H264_DMA 通道进行安全复位。
- 为确保数据传输的完成，在复位 RX 通道 0、RX 通道 1、RX 通道 3 和 RX 通道 4 前，需要检查相应的通道链表描述符是否写回成功。只有在通道的链表描述符写回成功后，即通道的所有数据传输完成后，才能复位通道。但是，当未启用 MV 合并功能，或 GOP 中的视频帧数为 1 时（即寄存器 H264_GOP_NUM 配置为 1 时），无需检查 RX 通道 3 的链表描述符是否写回成功。

41.5.2.5 调试用数据替换

H264_DMA 各个通道支持用调试数据替换各个通道的原始数据。

- 通过 H264_DMA_CH_DBG_EN_REG 控制是否开启对应通道的调试数据替换
- 调试数据替换开启后，对应通道的数据会被 H264_DMA_IN/OUT_CHx_DBG_DATA_H/L_REG 替换。其中 TX 通道 0 用来读取原始图像，其替换数据有效域根据 pbyte 配置变换而不同，其余通道都是 64bit 直接替换。TX 通道 0 的替换如下：
  - H264_DMA_ORI_PBYTE=0，H264_OUT_CH0_DBG_DATA_L 的 3:0 位有效
  - H264_DMA_ORI_PBYTE=1，H264_OUT_CH0_DBG_DATA_L 的 7:0 位有效
  - H264_DMA_ORI_PBYTE=2，H264_OUT_CH0_DBG_DATA_L 的 23:0 位有效

- H264_DMA_ORI_PBYTE=3，H264_OUT_CH0_DBG_DATA_L 的 15:0 位有效
- H264_DMA_ORI_PBYTE=4，H264_OUT_CH0_DBG_DATA_L 的 23:0 位有效

41.5.2.6 通道配置

在软件启动 H264_DMA 前，需要先对 H264_DMA 进行配置，包括寄存器配置和链表描述符配置。

链表描述符配置：需要配置描述符的通道包括 TX 通道 0 ~ TX 通道 4，RX 通道 0 ~ RX 通道 4。链表描述符均为循环链表。

1. 配置读取原始图像的通道（TX 通道 0）：
   - 模式：2D，mode1。
   - 长度/大小：HA 代表图像的水平像素数，VA 代表图像的垂直像素数，HB 为 16 × 4，VB 为 16。
2. 配置读取参考图像的通道（TX 通道 1）：
   - 模式：2D，mode1。
   - 长度/大小：HA 为 NUM_H_MB × 16 × 16，VA 为 3，HB 为 16 × 16，VB 为 3。
3. 配置读取 DB 中间数据的通道（TX 通道 2）：
   - 模式：1D。
   - 长度/大小：读取空间大小为 NUM_H_MB × (16 × 4 + 8 × 4 × 2)。
4. 配置读取 DB 数据前 12 行的通道（TX 通道 3）：
   - 模式：1D。
   - 长度/大小：读取空间大小为 (NUM_H_MB × (图像中的垂直宏块行数 - 1) × (16 × 12 + 8 × 4 × 2)) + (NUM_H_MB × 16 × 16 × 1.5)。
5. 配置读取 DB 数据的最后 4 行的通道（TX 通道 4）：
   - 模式：1D。
   - 长度/大小：读取空间大小是 (NUM_H_MB × (图像中的垂直宏块行数 - 1) × (16 × 4 + 8 × 4 × 2))。
6. 配置写入 DB 数据前 12 行的通道（RX 通道 0）：
   - 模式：1D。
   - 长度/大小：写入空间的大小与 TX 通道 3 的大小相同。
7. 配置写入 DB 数据的最后 4 行的通道（RX 通道 1）：
   - 模式：1D。
   - 长度/大小：写入空间的大小与 TX 通道 4 的大小相同。
8. 配置写入 DB 中间数据的通道（RX 通道 2）：
   - 模式：1D。
   - 长度/大小：写入空间的大小与 TX 通道 2 的大小相同。
9. 配置写入 MV 合并数据的通道（RX 通道 3）：
   - 模式：1D。

- 长度/大小：写入空间大小是 NUM_H_MB × 垂直宏块行数 × 4。

10. 配置写入比特流数据的通道（RX 通道 4）：
    - 模式：1D。
    - 长度/大小：写入空间的大小是图像的水平像素数 × 垂直像素行数 × 1.5。

寄存器配置：

- H264_DMA_ORI_PBYTE：此寄存器配置原始图像中每个像素的字节数。
- H264_DMA_OUT/IN_MEM_BURST_LENGTH_CH*x*：此寄存器控制相应通道的 AXI 突发长度。
  - 0：8 字节。1：16 字节。2：32 字节。3：64 字节。4：128 字节。
  - 在 TX 通道中，*x* 可以是 0 ~ 4。在 RX 通道中，*x* 可以是 0 ~ 5。
  - TX 通道 2 和 RX 通道 2 的突发长度配置必须相同。
- H264_DMA_OUT_REORDER_EN_CH0：此寄存器控制 TX 通道 0 是否启用数据重排序。当 TX 通道 0 的链表描述符配置 HB 不为 16 时，必须将此寄存器配置为 1。
- H264_DMA_BLOCK_START_ADDR_CH5：此寄存器控制 RX 通道 5 的数据起始地址。此寄存器配置必须与 TX 通道 1 链表描述符中的数据缓冲区地址一致。
- H264_DMA_BLOCK_ROW_LENGTH_4LINE_CH5：此寄存器指定一行 4 行块中包含的字节数，为 (NUM_H_MB × (16 × 4 + 8 × 4 × 2))。
- H264_DMA_BLOCK_ROW_LENGTH_12LINE_CH5：此寄存器指定一行 12 行块中包含的字节数，为 (NUM_H_MB × (16 × 12 + 8 × 4 × 2))。
- H264_DMA_BLOCK_LENGTH_4LINE_CH5：此寄存器指定一个 4 行块包含的字节数，为 (16 × 4 + 8 × 4 × 2)。
- H264_DMA_BLOCK_LENGTH_12LINE_CH5：此寄存器指定一个 12 行块包含的字节数，为 (16 × 12 + 8 × 4 × 2)。

41.6 中断

ESP32-P4 的 H264 编码器可以生成以下中断信号，并将其发送到中断矩阵。

- H264_REG_INT
- H264_DMA_IN_CH0_INT
- H264_DMA_IN_CH1_INT
- H264_DMA_IN_CH2_INT
- H264_DMA_IN_CH3_INT
- H264_DMA_IN_CH4_INT
- H264_DMA_IN_CH5_INT
- H264_DMA_OUT_CH0_INT
- H264_DMA_OUT_CH1_INT
- H264_DMA_OUT_CH2_INT

- H264_DMA_OUT_CH3_INT
- H264_DMA_OUT_CH4_INT

H264 编码器有多个内部中断源可以生成上述中断信号。

H264_REG_INT 中断由以下内部中断源生成：

- H264_BS_BUFFER_OVERFLOW_INT：当 ENC_CORE 一个 4 × 4 子块的编码字节数超过H264_BS_BUFFER_THRESHOLD时，触发此中断。
- H264_DMA_MOVE_2MB_LINE_DONE_INT：当 H264_DMA 将两个宏块行的参考图像从外部存储移动到内部存储时，触发此中断。
- H264_FRAME_DONE_INT：当 ENC_CORE 完成一帧视频的压缩和编码时，触发此中断。
- H264_REC_READY_INT：当 ENC_CORE 将 H264_*x*_REC_READY_TRIGGER_MB_LINES 行宏块的去块滤波结果（即参考图像）写入外部存储后，触发此中断。*x* 可以是 A 或 B，表示该寄存器对应于视频序列 A 或 B。
- H264_DB_TMP_READY_INT：当 ENC_CORE 将 H264_*x*_DB_TMP_READY_TRIGGER_MB_NUM 个宏块的去块滤波中间数据写入内部存储后，触发此中断。*x* 可以是 A 或 B，表示该寄存器对应于视频序列 A 或 B。

要触发上述中断源，必须先将中断使能寄存器 *XXX*_ENA 设置为 1。要清除中断，将寄存器 *XXX*_CLR 设置为 1（*XXX* 为中断源名称）。

H264_DMA_IN_CH*n*_INT（*n* 为 0 ~ 5）中断由以下中断源生成：

- H264_DMA_IN_DONE_CH*n*_INT：当 RX 通道 *n* (0 ~ 5) 接收到与接收描述符对应的所有数据时，触发此中断。
- H264_DMA_IN_SUC_EOF_CH*n*_INT：当接收描述符中的 suc_eof 位为 1，并且 RX 通道 *n* (0 ~ 5) 接收到与此接收描述符对应的数据时（即已接收到与 inlink 对应的数据帧或数据包），触发此中断。
- H264_DMA_IN_ERR_EOF_CH*n*_INT：当 RX 通道 *n* (0 ~ 4) 接收的数据帧或数据包中检测到错误时，触发此中断。
- H264_DMA_IN_DSCR_EMPTY_CH*n*_INT：当接收描述符指向的缓冲区大小小于 RX 通道 *n* (0 ~ 4) 要接收的数据长度时，触发此中断。

INT_MV_OUT_EN 配置合并后 MV 的输出类型。
0：输出整数部分或小数部分非零的合并后 MV
1：只输出整数部分非零的合并后 MV
(R/W)

- H264_DMA_INFIFO_OVF_L1_CH*n*_INT：当 RX 通道 *n* (0 ~ 5) 的 L1 RX FIFO 溢出时，触发此中断。
- H264_DMA_INFIFO_OVF_L2_CH*n*_INT：当 RX 通道 *n* (0 ~ 4) 的 L2 RX FIFO 溢出时，触发此中断。
- H264_DMA_INFIFO_UDF_L1_CH*n*_INT：当 RX 通道 *n* (0 ~ 5) 的 L1 RX FIFO 下溢时，触发此中断。
- H264_DMA_INFIFO_UDF_L2_CH*n*_INT：当 RX 通道*n* (0 ~ 4) 的 L2 RX FIFO 下溢时，触发此中断。
- H264_DMA_IN_DSCR_TASK_OVF_CH*n*_INT：当 RX 通道*n* (0 ~ 4) 的描述符就绪任务 FIFO 溢出时，触发此中断。
- H264_DMA_FETCH_MB_COL_CNT_OVF_CH*n*_INT：当 RX 通道 *n* (*n* = 5) 获取宏块列计数溢出时，触发此中断。

H264_DMA_OUT_CH*n*_INT（*n* 为 0 ~ 4）信号由以下中断源生成：

- H264_DMA_OUT_DONE_CH*n*_INT：当与发送描述符对应的所有数据都通过 TX 通道 *n* 发送后，触发此中断。
- H264_DMA_OUT_EOF_CH*n*_INT：当发送描述符中的 suc_eof 位为 1，且与此描述符对应的数据已通过 TX 通道 *n* 发送时，触发此中断。如果 H264_DMA_OUT_EOF_MODE_CH*n* 为 0，此中断将在与此描述符对应的数据的最后一个字节进入 H264_DMA 的 TX 通道时触发；如果 H264_DMA_OUT_EOF_MODE_CH*n* 为 1，此中断将在从 H264_DMA 的 TX 通道取出最后一个字节时触发。
- H264_DMA_OUT_DSCR_ERR_CH*n*_INT：当 TX 通道 *n* 上的发送描述符未通过两项描述符检查中的任意一项时，触发此中断。
- H264_DMA_OUT_TOTAL_EOF_CH*n*_INT：当通过 TX 通道 *n* 发送了与链表（包括多个描述符）对应的所有数据时，触发此中断。
- H264_DMA_OUTFIFO_OVF_L1_CH*n*_INT：当 H264_DMA 的 L1 TX FIFO 溢出时，触发此中断。
- H264_DMA_OUTFIFO_OVF_L2_CH*n*_INT：当 H264_DMA 的 L2 TX FIFO 溢出时，触发此中断。
- H264_DMA_OUTFIFO_UDF_L1_CH*n*_INT：当 H264_DMA 的 L1 TX FIFO 下溢时，触发此中断。
- H264_DMA_OUTFIFO_UDF_L2_CH*n*_INT：当 H264_DMA 的 L2 TX FIFO 下溢时，触发此中断。
- H264_DMA_OUT_DSCR_TASK_OVF_CH*n*_INT：当 TX 通道 *n* 的描述符就绪任务 FIFO 溢出时，触发此中断。

41.7 配置流程

H264 编码器具有两种工作模式：GOP 模式和双流模式。在 GOP 模式下，编码器在指定的 P 帧间隔中插入一个 I 帧。在双流模式下，编码器轮流处理两个视频序列，硬件会自动切换相应的 ENC_CORE 配置。

41.7.1 GOP 模式

在 GOP 模式下，编码单个视频序列 A。以一个包含 3 个图像的 GOP 为例，图像的编码顺序如图 41.7-1 所示：

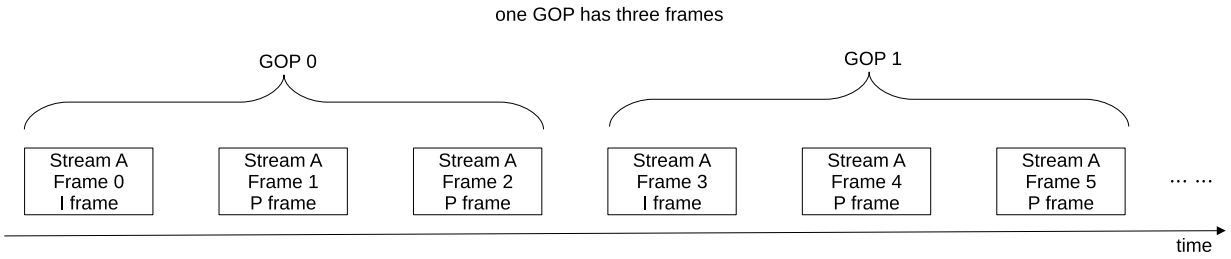


图 41.7-1. GOP 模式下的图像编码顺序

在 GOP 模式下，前缀为 H264_A_ 的寄存器有效，前缀为 H264_B_ 的寄存器无效。

GOP 模式的配置过程如下：

1. 参考章节 41.5.2.6 配置每个 H264_DMA 通道：
   (a) 配置 H264_DMA TX 通道 0。

(b) 配置 H264_DMA TX 通道 1。

(c) 配置 H264_DMA TX 通道 2。

(d) 配置 H264_DMA TX 通道 3 和 TX 通道 4。

(e) 配置 H264_DMA RX 通道 0 和 RX 通道 1。

(f) 配置 H264_DMA RX 通道 2。

(g) 配置 H264_DMA RX 通道 3。

(h) 配置 H264_DMA RX 通道 4。

(i) 配置 H264_DMA RX 通道 5。

2. 根据所需功能配置 ENC_CORE：

(a) 配置 GOP 模式：将寄存器 H264_FRAME_MODE 和寄存器 H264_DUAL_STREAM_MODE 都设置为 0。

(b) 配置每个 GOP 包含的图像数量：将寄存器 H264_GOP_NUM 配置为 n，则表示一个 GOP 包含 n 个图像。当此寄存器配置为 0 时，GOP 的第一帧是 I 帧，后续帧始终是 P 帧，直到编码器被重置为止。

(c) 配置图像的宏块分辨率：将寄存器 H264_A_SYS_TOTAL_MB_X 配置为图像水平方向的宏块数量，将寄存器 H264_A_SYS_TOTAL_MB_Y 配置为图像垂直方向的宏块数量。水平方向的宏块数量等于视频宽度除以 16 并向上取整的值，垂直方向的宏块数量等于视频高度除以 16 并向上取整的值。

(d) 配置图像初始 QP 值：将寄存器 H264_A_QP 配置为初始 QP 值，范围为 10 到 51。

(e) 配置去块滤波器：将寄存器 H264_A_BYPASS_DB_FILTER 设置为 0 以启用去块滤波器，将寄存器 H264_A_BYPASS_DB_FILTER 设置为 1 以禁用去块滤波器。

(f) 配置切片头信息：根据 H264 标准，切片头需包含一些必要信息，这些信息可在以下寄存器中进行配置：H264_SLICE_REMAIN_BIT、H264_SLICE_REMAIN_BITLENGTH、H264_SLICE_BYTE_LENGTH、H264_SLICE_BYTE_LSB 和 H264_SLICE_BYTE_MSB。例如切片头是一个 41 位变量，最高有效位是切片头的第一位，那么这些寄存器应该按以下方式设置：

- H264_SLICE_BYTE_MSB：切片头的第 41 位到第 10 位
- H264_SLICE_BYTE_LSB：切片头的第 9 位到第 2 位
- H264_SLICE_REMAIN_BIT：切片头的第 1 位
- H264_SLICE_BYTE_LENGTH：5
- H264_SLICE_REMAIN_BITLENGTH：1

(g) 配置原始视频色彩空间：将寄存器 H264_A_ORI_COLOR_SPACE 配置为对应的原始视频色彩空间。

(h) 如需使用量化结果降采样功能，请参考 41.5.1.3 章节配置相关寄存器。

(i) 如需使用宏块级码率控制功能，请参考 41.5.1.5 章节配置相关寄存器。

(j) 如需使用 ROI 功能，请参考 41.5.1.6 章节配置相关寄存器。

(k) 如需使用 MV 合并功能，请参考 41.5.1.4 章节配置相关寄存器。

(l) 如需使用调试功能，请参考 41.5.1.7 章节配置相关寄存器。

3. 根据当前要编码的视频帧的类型，执行相应的操作。（注意，在一个 GOP 中，第一帧是 I 帧，后续帧都是 P 帧）。

- 当前待编码帧为 I 帧：
  (a) 启动 H264_DMA TX 通道 0：请参考 41.5.2.3 章节进行配置。
  (b) 启动 H264_DMA RX 通道 0 和 RX 通道 1：请参考 41.5.2.3 章节进行配置。
  (c) 启动 H264_DMA RX 通道 2：请参考 41.5.2.3 章节进行配置。
  (d) 如要使用 MV 合并功能，启动 H264_DMA RX 通道 3：请参考 41.5.2.3 进行配置。
  (e) 启动 H264_DMA RX 通道 4：请参考 41.5.2.3 章节进行配置。
  (f) 清除 H264 中断并启用中断。
  (g) 开始图像编码：将寄存器 H264_FRAME_START 写为 1。
  (h) 等待 H264_DB_TMP_READY_INT 中断。中断触发后，启动 H264_DMA TX 通道 2。请参考 41.5.2.3 章节进行配置。
  (i) 等待 H264_REC_READY_INT 中断。中断触发后，启动 H264_DMA RX 通道 5 并启动 H264_DMA TX 通道 3 和 TX 通道 4。请参考 41.5.2.3 章节进行配置。
  (j) 启动 H264_DMA 将参考图像从外部存储移动到内部存储：将寄存器 H264_DMA_MOVE_START 写为 1。
  (k) 等待 H264_DMA_MOVE_2MB_LINE_DONE_INT 中断。中断触发后，启动 H264_DMA TX 通道 1。配置方法请参考 41.5.2.3 章节的描述。
  (l) 等待 H264_FRAME_DONE_INT 中断。中断触发表示图像的编码已完成。
  (m) 重置 H264_DMA TX 通道 2 和 RX 通道 2：配置方法请参考 41.5.2.4 章节的描述。
  (n) 重置 H264_DMA 中解块滤波器中间数据的读写次数计数器：将寄存器 H264_DMA_RX_CH2_INTER_COUNTER_RST 设置为 1，然后设置为 0。
- 当前待编码帧为 P 帧：
  (a) 清除 H264 中断并启用中断。
  (b) 启动 H264_DMA RX 通道 2：配置方法请参考 41.5.2.3 章节的描述。
  (c) 开始图像编码：将寄存器 H264_FRAME_START 写为 1。
  (d) 等待 H264_DB_TMP_READY_INT 中断。中断触发时，启动 H264_DMA TX 通道 2。配置方法请参考 41.5.2.3 章节的描述。
  (e) 等待 H264_FRAME_DONE_INT 中断。中断触发时，表示图像编码已完成。
  (f) 重置 H264_DMA TX 通道 2 和 RX 通道 2：配置方法请参考 41.5.2.4 章节的描述。
  (g) 重置 H264_DMA 中去块滤波器中间数据的读写次数计数器：将寄存器 H264_DMA_RX_CH2_INTER_COUNTER_RST 设置为 1，然后设置为 0。
- 可以在任何帧编码结束后重新配置以下寄存器：
  - 与量化结果降采样功能相关的寄存器，参考 41.5.1.3 章节的描述。
  - 与宏块级别码率控制功能相关的寄存器，参考41.5.1.5章节的描述。
  - 去块滤波器使能寄存器：H264_A_BYPASS_DB_FILTER。
  - ROI 功能相关的寄存器，参考 41.5.1.6 章节的描述。

- 与切片头信息相关的寄存器：H264_SLICE_REMAIN_BIT、H264_SLICE_REMAIN_BITLENGTH、H264_SLICE_BYTE_LENGTH、H264_SLICE_BYTE_LSB 和 H264_SLICE_BYTE_MSB。
- 与 MV 合并功能相关的寄存器，参考 41.5.1.4 章节的描述。
- 与调试功能相关的寄存器，参考 41.5.1.7 章节的描述。

4. 重复步骤 3，直到一个 GOP 中的所有视频帧都完成编码。
5. 确保数据完整写入。方法请参考 41.5.2.4 章节的描述。
6. 重置 H264_DMA 的所有发送通道和接收通道，参考 41.5.2.4 章节的描述。
7. 重置 H264_DMA 中去块滤波器结果数据的读写次数计数器，以及参考图像数据的读写次数计数器。具体而言，将寄存器 H264_DMA_RX_CH0_EXTER_COUNTER_RST 设置为 1，然后设置为 0；将寄存器 H264_DMA_RX_CH1_EXTER_COUNTER_RST 设置为 1，然后设置为 0；将寄存器H264_DMA_RX_CH5_INTER_COUNTER_RST 设置为 1，然后设置为 0。
8. 从步骤 2 开始，重新配置并编码下一个 GOP。

41.7.2 双流模式

在双流模式下，编码两个视频序列 A 和 B，图像的编码顺序如图 41.7-2 所示（一个 GOP 包含 3 个图像）。

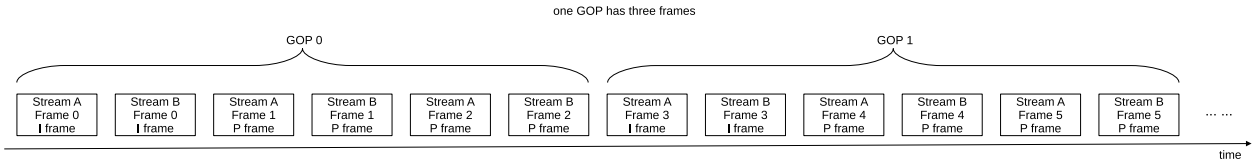


图 41.7-2. 双流模式下图像编码的顺序

在双流模式下，H264 编码帧的顺序固定，为 A1，B1，A2，B2，A3，B3，…，依此类推。在处理视频序列 A 的帧时，自动应用前缀为 H264_A_ 的寄存器，在处理视频序列 B 的帧时，自动应用前缀为 H264_B_ 的寄存器。其他前缀的寄存器对两个视频序列的帧都有效。

双流模式的配置过程如下：

1. 根据所需功能，配置 ENC_CORE：
   (a) 配置双流模式：将寄存器 H264_FRAME_MODE 设置为 1，并将寄存器 H264_DUAL_STREAM_MODE 设置为 1。
   (b) 配置 GOP（视频序列 A 和 B 的 GOP 配置必须相同）：将寄存器 H264_GOP_NUM 配置为 n，表示一个 GOP 包含 n 个图像。当寄存器配置为 0 时，GOP 的第一帧是 I 帧，后续帧始终是 P 帧，直到编码器复位为止。
   (c) 配置图像的宏块分辨率：将寄存器 H264_A_SYS_TOTAL_MB_X 配置为视频序列 A 在水平方向的宏块数，将寄存器 H264_A_SYS_TOTAL_MB_Y 配置为视频序列 A 在垂直方向的宏块数。将寄存器 H264_B_SYS_TOTAL_MB_X 配置为视频序列 B 在水平方向的宏块数，将寄存器 H264_B_SYS_TOTAL_MB_Y 配置为视频序列 B 在垂直方向的宏块数。水平方向的宏块数等于视频宽度除以 16 并向上取整，垂直方向的宏块数等于视频高度除以 16 并向上取整。
   (d) 配置图像的初始 QP：将寄存器 H264_A_QP 配置为视频序列 A 的初始 QP，范围为 10 到 51。将寄存器 H264_B_QP 配置为视频序列 B 的初始 QP，范围为 10 到 51。

(e) 配置去块滤波器：将寄存器 H264_A_BYPASS_DB_FILTER 设置为 0，以启用视频序列 A 的去块滤波器。将寄存器 H264_A_BYPASS_DB_FILTER 设置为 1，以禁用视频序列 A 的去块滤波器。将寄存器 H264_B_BYPASS_DB_FILTER 设置为 0，以启用视频序列 B 的去块滤波器。将寄存器 H264_B_BYPASS_DB_FILTER 设置为 1，以禁用视频序列 B 的去块滤波器。

(f) 配置切片头信息：根据 H264 标准，切片头需包含一些必要信息，配置这些信息的相关寄存器包括：H264_SLICE_REMAIN_BIT、H264_SLICE_REMAIN_BITLENGTH、H264_SLICE_BYTE_LENGTH、H264_SLICE_BYTE_LSB 和 H264_SLICE_BYTE_MSB。例如，如果切片头是一个 41 位变量，最高有效位是切片头的第一位，那么这些寄存器应按以下方式设置：

- H264_SLICE_BYTE_MSB：切片头的第 41 位到第 10 位
- H264_SLICE_BYTE_LSB：切片头的第 9 位到第 2 位
- H264_SLICE_REMAIN_BIT：切片头的第 1 位
- H264_SLICE_BYTE_LENGTH：5
- H264_SLICE_REMAIN_BITLENGTH：1

(g) 配置原始视频色彩空间：将寄存器 H264_A_ORI_COLOR_SPACE 配置为视频序列 A 对应的原始视频色彩空间。将寄存器 H264_B_ORI_COLOR_SPACE 配置为视频序列 B 对应的原始视频色彩空间。

(h) 如需使用量化结果降采样功能，请参考 41.5.1.3 章节配置相关寄存器。

(i) 如需使用宏块级码率控制功能，请参考 41.5.1.5 章节配置相关寄存器。

(j) 如需使用 ROI 功能，请参考 41.5.1.6 章节配置相关寄存器。

(k) 如需使用 MV 合并功能，请参考 41.5.1.4 章节配置相关寄存器。

(l) 如需使用调试功能，请参考 41.5.1.7 章节配置相关寄存器。

2. 根据待编码的帧所属的视频序列（A 或 B），使用与视频序列对应的信息，并参考 41.5.2.6 章节的描述配置每个 H264_DMA 通道：

(a) 配置 H264_DMA TX 通道 0。

(b) 配置 H264_DMA TX 通道 1。

(c) 配置 H264_DMA TX 通道 2。

(d) 配置 H264_DMA TX 通道 3 和 TX 通道 4。

(e) 配置 H264_DMA RX 通道 0 和 RX 通道 1。

(f) 配置 H264_DMA RX 通道 2。

(g) 配置 H264_DMA RX 通道 3。

(h) 配置 H264_DMA RX 通道 4。

(i) 配置 H264_DMA RX 通道 5。

3. 根据当前要编码的视频帧的类型，执行相应的操作。视频序列 A 和 B 交替编码，即首先编码视频序列 A 中的一帧，然后编码视频序列 B 中的一帧，交替进行。对于任何视频序列，在一个 GOP 中，第一帧是 I 帧，后续帧是 P 帧。

- 当前待编码帧为 I 帧：

(a) 启动 H264_DMA TX 通道 0：配置方法请参考 41.5.2.3 章节的描述。

(b) 启动 H264_DMA RX 通道 0 和 RX 通道 1：配置方法请参考 41.5.2.3 章节的描述。

(c) 启动 H264_DMA RX 通道 2：配置方法请参考 41.5.2.3 章节的描述。

(d) 启动 H264_DMA RX 通道 4：配置方法请参考 41.5.2.3 章节的描述。

(e) 清除 H264 中断并启用中断。

(f) 开始图像编码：将寄存器 H264_FRAME_START 写为 1。

(g) 等待 H264_DB_TMP_READY_INT 中断。中断触发时，启动 H264_DMA TX 通道 2。配置方法请参考 41.5.2.3 章节的描述。

(h) 等待 H264_FRAME_DONE_INT 中断。中断触发时，表示图像的编码已完成。

(i) 确保数据完全写入。方法请参考 41.5.2.4 章节的描述。

(j) 重置 H264_DMA 的每个发送和接收通道，参考 41.5.2.4 章节的描述。

(k) 重置 H264_DMA 中去块滤波器中间数据的读写计数器，重置参考图像数据的读写计数器。将寄存器 H264_DMA_RX_CH2_INTER_COUNTER_RST 设置为 1，然后设置为 0。将寄存器 H264_DMA_RX_CH5_INTER_COUNTER_RST 设置为 1，然后设置为 0。

- 当前待编码帧为 P 帧：

(a) 清除 H264 中断并启用中断。

(b) 启动 H264_DMA TX 通道 0：配置方法请参考 41.5.2.3 章节的描述。

(c) 启动 H264_DMA RX 通道 0 和 RX 通道 1：配置方法请参考 41.5.2.3 章节的描述。

(d) 启动 H264_DMA RX 通道 2：配置方法请参考 41.5.2.3 章节的描述。

(e) 启动 H264_DMA RX 通道 4：配置方法请参考 41.5.2.3 章节的描述。

(f) 启动 H264_DMA RX 通道 5：配置方法请参考 41.5.2.3 章节的描述。

(g) 启动 H264_DMA TX 通道 3 和 TX 通道 4：配置方法请参考 41.5.2.3 章节的描述。

(h) 如需使用 MV 合并功能，应启动 H264_DMA RX 通道 3：配置方法请参考 41.5.2.3 章节的描述。

(i) 启动 H264_DMA，将参考图像从外部存储移动到内部存储：将寄存器 H264_DMA_MOVE_START 写为 1。

(j) 等待 H264_DMA_MOVE_2MB_LINE_DONE_INT 中断。中断触发时，启动 H264_DMA TX 通道 1。配置方法请参考 41.5.2.3 章节的描述。

(k) 开始图像编码：将寄存器 H264_FRAME_START 写为 1。

(l) 等待 H264_DB_TMP_READY_INT 中断。中断触发时，启动 H264_DMA TX 通道 2。配置方法请参考 41.5.2.3 章节的描述。

(m) 等待 H264_FRAME_DONE_INT 中断。中断触发时，表示图像编码已完成。

(n) 确保数据完全写入。方法请参考 41.5.2.4 章节的描述。

(o) 重置 H264_DMA 的每个发送和接收通道，参考 41.5.2.4 章节的描述。

(p) 重置 H264_DMA 中去块滤波器中间数据的读写计数器和参考图像数据的读写计数器。将寄存器 H264_DMA_RX_CH2_INTER_COUNTER_RST 设置为 1，然后设置为 0。将寄存器 H264_DMA_RX_CH5_INTER_COUNTER_RST 设置为 1，然后设置为 0。

- 在任何帧编码结束后，可以重新配置以下与当前编码的视频序列对应的寄存器：
  - 与量化结果降采样函数相关的寄存器，请参考 41.5.1.3 章节的描述。
  - 与宏块级别码率控制功能相关的寄存器，请参考 41.5.1.5 章节的描述。
  - 去块滤波器使能寄存器：H264_A_BYPASS_DB_FILTER。
  - ROI 功能相关寄存器，请参考 41.5.1.6 章节的描述。
  - 与切片头信息相关的寄存器：H264_SLICE_REMAIN_BIT、H264_SLICE_REMAIN_BITLENGTH、H264_SLICE_BYTE_LENGTH、H264_SLICE_BYTE_LSB 和 H264_SLICE_BYTE_MSB。
  - 与 MV 合并功能相关的寄存器，请参考 41.5.1.4 章节的描述。
  - 与调试功能相关的寄存器，请参考 41.5.1.7 章节的描述。

4. 重复步骤 2 和 3，直到两个视频序列都完成一个 GOP 编码。
5. 重置 H264_DMA 中去块滤波器结果数据的读写次数计数器。将寄存器 H264_DMA_RX_CH0_EXTER_COUNTER_RST设置为 1，然后设置为 0。将寄存器H264_DMA_RX_CH1_EXTER_COUNTER_RST设置为 1，然后设置为 0。
6. 从步骤 1 开始，重新配置并编码下一个 GOP。

41.7.3 软重置

无论 H264 处于何种工作模式，按照以下流程重置，即可以随时停止编码。

1. 重置 ENC_CORE：在寄存器 H264_SYS_RST_PULSE 中写入 1。
2. 重置 H264_DMA 的每个发送和接收通道，参考章节 41.5.2.4。
3. 重置 H264_DMA 中的每个读写次数计数器。将寄存器 H264_DMA_RX_CH0_EXTER_COUNTER_RST 设置为 1，然后设置为 0。将寄存器H264_DMA_RX_CH1_EXTER_COUNTER_RST 设置为 1，然后设置为 0。将寄存器H264_DMA_RX_CH2_INTER_COUNTER_RST 设置为 1，然后设置为 0。将寄存器 H264_DMA_RX_CH5_INTER_COUNTER_RST 设置为 1，然后设置为 0。
4. 此时，H264 编码器返回到初始状态，可以根据需要重新配置。

41.8 寄存器列表

本节中的地址是相对于 **H264 编码器**基地址和 **H264 DMA** 基地址的相对地址。如需了解基地址详细信息，请参考 9 系统和存储器 章节，表 9.3-2 模块/外设地址空间映射表。

参考章节 寄存器访问类型，了解 **访问**列中的缩写含义。

41.8.1 H264 编码器寄存器列表

名称	描述	地址	访问
配置寄存器			
H264_SYS_CTRL_REG	H264 系统级控制寄存器	0x0000	varies
H264_GOP_CONF_REG	GOP 相关配置寄存器	0x0004	R/W
H264_SLICE_HEADER_REMAIN_REG	帧切片头剩余位寄存器	0x00AC	R/W
H264_SLICE_HEADER_BYTE_LENGTH_REG	帧切片头字节长度寄存器	0x00B0	R/W
H264_BS_THRESHOLD_REG	比特流缓冲区溢出阈值寄存器	0x00B4	R/W
H264_SLICE_HEADER_BYTE0_REG	帧切片头字节低 32 位寄存器	0x00B8	R/W
H264_SLICE_HEADER_BYTE1_REG	帧切片头字节高 32 位寄存器	0x00BC	R/W
H264_CONF_REG	通用配置寄存器	0x00D0	R/W
H264_MV_MERGE_CONFIG_REG	MV 合并配置寄存器	0x00D4	varies
视频序列 A 配置寄存器			
H264_A_SYS_MB_RES_REG	视频序列 A 的水平和垂直 MB 分辨率寄存器	0x0008	R/W
H264_A_SYS_CONF_REG	视频序列 A 的系统级配置寄存器	0x000C	R/W
H264_A_DECI_SCORE_REG	视频序列 A 的亮度和色度 MB 降采样分数寄存器	0x0010	R/W
H264_A_DECI_SCORE_OFFSET_REG	视频序列 A 的亮度和色度 MB 降采样分数偏移寄存器	0x0014	R/W
H264_A_RC_CONF0_REG	视频序列 A 的码率控制配置寄存器 0	0x0018	R/W
H264_A_RC_CONF1_REG	视频序列 A 的码率控制配置寄存器 1	0x001C	R/W
H264_A_DB_BYPASS_REG	视频序列 A 的去块绕过寄存器	0x0020	R/W
H264_A_ROI_REGION0_REG	视频序列 A 的 H264 ROI0 范围配置寄存器	0x0024	R/W
H264_A_ROI_REGION1_REG	视频序列 A 的 H264 ROI1 范围配置寄存器	0x0028	R/W
H264_A_ROI_REGION2_REG	视频序列 A 的 H264 ROI2 范围配置寄存器	0x002C	R/W
H264_A_ROI_REGION3_REG	视频序列 A 的 H264 ROI3 范围配置寄存器	0x0030	R/W
H264_A_ROI_REGION4_REG	视频序列 A 的 H264 ROI4 范围配置寄存器	0x0034	R/W
H264_A_ROI_REGION5_REG	视频序列 A 的 H264 ROI5 范围配置寄存器	0x0038	R/W
H264_A_ROI_REGION6_REG	视频序列 A 的 H264 ROI6 范围配置寄存器	0x003C	R/W

名称	描述	地址	访问
H264_A_ROI_REGION7_REG	视频序列 A 的 H264 ROI7 范围配置寄存器	0x0040	R/W
H264_A_ROI_REGION0_3_QP_REG	视频序列 A 的 H264 ROI0-3 QP 寄存器	0x0044	R/W
H264_A_ROI_REGION4_7_QP_REG	视频序列 A 的 H264 ROI4-7 QP 寄存器	0x0048	R/W
H264_A_NO_ROI_REGION_QP_OFFSET_REG	视频序列 A 的 H264 无 ROI QP 寄存器	0x004C	R/W
H264_A_ROI_CONFIG_REG	视频序列 A 的 H264 ROI 配置寄存器	0x0050	R/W
视频序列 B 配置寄存器			
H264_B_SYS_MB_RES_REG	视频序列 B 的水平和垂直 MB 分辨率寄存器	0x0054	R/W
H264_B_SYS_CONF_REG	视频序列 B 的系统级配置寄存器	0x0058	R/W
H264_B_DECI_SCORE_REG	视频序列 B 的亮度和色度 MB 降采样分数寄存器	0x005C	R/W
H264_B_DECI_SCORE_OFFSET_REG	视频序列 B 的亮度和色度 MB 降采样分数偏移寄存器	0x0060	R/W
H264_B_RC_CONF0_REG	视频序列 B 的码率控制配置寄存器 0	0x0064	R/W
H264_B_RC_CONF1_REG	视频序列 B 的码率控制配置寄存器 1	0x0068	R/W
H264_B_DB_BYPASS_REG	视频序列 B 的去块绕过寄存器	0x006C	R/W
H264_B_ROI_REGION0_REG	视频序列 B 的 H264 ROI0 范围配置寄存器	0x0070	R/W
H264_B_ROI_REGION1_REG	视频序列 B 的 H264 ROI1 范围配置寄存器	0x0074	R/W
H264_B_ROI_REGION2_REG	视频序列 B 的 H264 ROI2 范围配置寄存器	0x0078	R/W
H264_B_ROI_REGION3_REG	视频序列 B 的 H264 ROI3 范围配置寄存器	0x007C	R/W
H264_B_ROI_REGION4_REG	视频序列 B 的 H264 ROI4 范围配置寄存器	0x0080	R/W
H264_B_ROI_REGION5_REG	视频序列 B 的 H264 ROI5 范围配置寄存器	0x0084	R/W
H264_B_ROI_REGION6_REG	视频序列 B 的 H264 ROI6 范围配置寄存器	0x0088	R/W
H264_B_ROI_REGION7_REG	视频序列 B 的 H264 ROI7 范围配置寄存器	0x008C	R/W
H264_B_ROI_REGION0_3_QP_REG	视频序列 B 的 H264 ROI0-3 QP 寄存器	0x0090	R/W
H264_B_ROI_REGION4_7_QP_REG	视频序列 B 的 H264 ROI4-7 QP 寄存器	0x0094	R/W
H264_B_NO_ROI_REGION_QP_OFFSET_REG	视频序列 B 的 H264 无 ROI QP 寄存器	0x0098	R/W
H264_B_ROI_CONFIG_REG	视频序列 B 的 H264 ROI 配置寄存器	0x009C	R/W
H264_A_ORI_CONF_REG	视频序列 A 的原始图像配置寄存器	0x00F4	R/W
H264_B_ORI_CONF_REG	视频序列 B 的原始图像配置寄存器	0x00F8	R/W
H264_ORI_DEBUG_CONF_REG	原始图像调试配置寄存器	0x00FC	R/W
H264_MV_MERGE_DEBUG_CONF_REG	MV 合并调试配置寄存器	0x0100	R/W
H264_BS_DEBUG_CONG_REG	编码比特流调试配置寄存器	0x0104	R/W
H264_DB_WR_TEMP_DEBUG_CONG_REG	去块滤波器写临时数据调试配置寄存器	0x0108	R/W
H264_DB_RD_TEMP_DEBUG_CONG_REG	去块滤波器读临时数据调试配置寄存器	0x010C	R/W

名称	描述	地址	访问
H264_DB_WR_DEBUG_CONG_REG	去块滤波器最终数据调试配置寄存器	0x0110	R/W
H264_REF_DEBUG_CONG_REG	参考帧调试配置寄存器	0x0114	R/W
状态寄存器			
H264_RC_STATUS0_REG	码率控制状态寄存器 0	0x00A0	RO
H264_RC_STATUS1_REG	码率控制状态寄存器 1	0x00A4	RO
H264_RC_STATUS2_REG	码率控制状态寄存器 2	0x00A8	RO
H264_SYS_STATUS_REG	系统状态寄存器	0x00DC	RO
H264_FRAME_CODE_LENGTH_REG	帧代码字节长度寄存器	0x00E0	RO
中断寄存器			
H264_INT_RAW_REG	中断原始状态寄存器	0x00C0	R/WTC/SS
H264_INT_ST_REG	中断屏蔽状态寄存器	0x00C4	RO
H264_INT_ENA_REG	中断使能寄存器	0x00C8	R/W
H264_INT_CLR_REG	中断清除寄存器	0x00CC	WT
版本寄存器			
H264_DATE_REG	版本控制寄存器	0x00F0	R/W

41.8.2 H264 DMA 寄存器列表

名称	描述	地址	访问
配置寄存器			
TX 配置寄存器			
H264_DMA_OUT_CONF0_CH0_REG	TX CH0 config0 寄存器	0x0000	R/W
H264_DMA_OUT_PUSH_CH0_REG	TX CH0 输出 FIFO 推送寄存器	0x0018	varies
H264_DMA_OUT_LINK_CONF_CH0_REG	TX CH0 out_link 配置寄存器	0x001C	varies
H264_DMA_OUT_LINK_ADDR_CH0_REG	TX CH0 out_link 描述符地址寄存器	0x0020	R/W
H264_DMA_OUT_ARB_CH0_REG	TX CH0 仲裁配置寄存器	0x003C	R/W
H264_DMA_OUT_RO_PD_CONF_CH0_REG	TX CH0 重排序 RAM 配置寄存器	0x0044	R/W
H264_DMA_OUT_MODE_ENABLE_CH0_REG	TX CH0 测试模式启用寄存器	0x0050	R/W
H264_DMA_OUT_MODE_YUV_CH0_REG	TX CH0 测试模式 yuv 值寄存器	0x0054	R/W
H264_DMA_OUT_ETM_CONF_CH0_REG	TX CH0 ETM 配置寄存器	0x0068	R/W
H264_DMA_OUT_CONF0_CH1_REG	TX CH1 config0 寄存器	0x0100	R/W
H264_DMA_OUT_PUSH_CH1_REG	TX CH1 输出 FIFO 推送寄存器	0x0118	varies
H264_DMA_OUT_LINK_CONF_CH1_REG	TX CH1 out_link 控制寄存器	0x011C	varies
H264_DMA_OUT_LINK_ADDR_CH1_REG	TX CH1 out_link 描述符地址寄存器	0x0120	R/W
H264_DMA_OUT_ARB_CH1_REG	TX CH1 仲裁配置寄存器	0x013C	R/W
H264_DMA_OUT_ETM_CONF_CH1_REG	TX CH1 ETM 配置寄存器	0x0168	R/W
H264_DMA_OUT_CONF0_CH2_REG	TX CH2 config0 寄存器	0x0200	R/W
H264_DMA_OUT_PUSH_CH2_REG	TX CH2 输出 FIFO 推送寄存器	0x0218	varies
H264_DMA_OUT_LINK_CONF_CH2_REG	TX CH2 out_link 控制寄存器	0x021C	varies
H264_DMA_OUT_LINK_ADDR_CH2_REG	TX CH2 out_link 描述符地址寄存器	0x0220	R/W
H264_DMA_OUT_ARB_CH2_REG	TX CH2 仲裁配置寄存器	0x023C	R/W

名称	描述	地址	访问
H264_DMA_OUT_ETM_CONF_CH2_REG	TX CH2 ETM 配置寄存器	0x0268	R/W
H264_DMA_OUT_CONF0_CH3_REG	TX CH3 config0 寄存器	0x0300	R/W
H264_DMA_OUT_PUSH_CH3_REG	TX CH3 输出 FIFO 推送寄存器	0x0318	varies
H264_DMA_OUT_LINK_CONF_CH3_REG	TX CH3 out_link 控制寄存器	0x031C	varies
H264_DMA_OUT_LINK_ADDR_CH3_REG	TX CH3 out_link 描述符地址寄存器	0x0320	R/W
H264_DMA_OUT_ARB_CH3_REG	TX CH3 仲裁配置寄存器	0x033C	R/W
H264_DMA_OUT_ETM_CONF_CH3_REG	TX CH3 ETM 配置寄存器	0x0368	R/W
H264_DMA_OUT_CONF0_CH4_REG	TX CH4 config0 寄存器	0x0400	R/W
H264_DMA_OUT_PUSH_CH4_REG	TX CH4 输出 FIFO 推送寄存器	0x0418	varies
H264_DMA_OUT_LINK_CONF_CH4_REG	TX CH4 out_link 控制寄存器	0x041C	varies
H264_DMA_OUT_LINK_ADDR_CH4_REG	TX CH4 out_link 描述符地址寄存器	0x0420	R/W
H264_DMA_OUT_ARB_CH4_REG	TX CH4 仲裁配置寄存器	0x043C	R/W
H264_DMA_OUT_ETM_CONF_CH4_REG	TX CH4 ETM 配置寄存器	0x0468	R/W
RX 配置寄存器			
H264_DMA_IN_CONF0_CH0_REG	RX CH0 config0 寄存器	0x0500	R/W
H264_DMA_IN_POP_CH0_REG	RX CH0 输入 FIFO 弹出寄存器	0x0518	varies
H264_DMA_IN_LINK_CONF_CH0_REG	RX CH0 in_link 控制寄存器	0x051C	varies
H264_DMA_IN_LINK_ADDR_CH0_REG	RX CH0 in_link 地址寄存器	0x0520	R/W
H264_DMA_IN_ARB_CH0_REG	RX CH0 仲裁配置寄存器	0x0540	R/W
H264_DMA_IN_RO_PD_CONF_CH0_REG	RX CH0 重排序电源配置寄存器	0x0548	R/W
H264_DMA_IN_ETM_CONF_CH0_REG	RX CH0 ETM 配置寄存器	0x056C	R/W
H264_DMA_IN_CONF0_CH1_REG	RX CH1 config0 寄存器	0x0600	R/W
H264_DMA_IN_POP_CH1_REG	RX CH1 输入 FIFO 弹出寄存器	0x0618	varies
H264_DMA_IN_LINK_CONF_CH1_REG	RX CH1 in_link 控制寄存器	0x061C	varies
H264_DMA_IN_LINK_ADDR_CH1_REG	RX CH1 in_link 地址寄存器	0x0620	R/W
H264_DMA_IN_ARB_CH1_REG	RX CH1 仲裁配置寄存器	0x0640	R/W
H264_DMA_IN_ETM_CONF_CH1_REG	RX CH1 ETM 配置寄存器	0x0648	R/W
H264_DMA_IN_CONF0_CH2_REG	RX CH2 config0 寄存器	0x0700	R/W
H264_DMA_IN_POP_CH2_REG	RX CH2 INFIFO pop 寄存器	0x0718	varies
H264_DMA_IN_LINK_CONF_CH2_REG	RX CH2 in_link 控制寄存器	0x071C	varies
H264_DMA_IN_LINK_ADDR_CH2_REG	RX CH2 in_link 地址寄存器	0x0720	R/W
H264_DMA_IN_ARB_CH2_REG	RX CH2 仲裁配置寄存器	0x0740	R/W
H264_DMA_IN_ETM_CONF_CH2_REG	RX CH2 ETM 配置寄存器	0x0748	R/W
H264_DMA_IN_CONF0_CH3_REG	RX CH3 config0 寄存器	0x0800	R/W
H264_DMA_IN_POP_CH3_REG	RX CH3 INFIFO pop 寄存器	0x0818	varies
H264_DMA_IN_LINK_CONF_CH3_REG	RX CH3 in_link 控制寄存器	0x081C	varies
H264_DMA_IN_LINK_ADDR_CH3_REG	RX CH3 in_link 地址寄存器	0x0820	R/W
H264_DMA_IN_ARB_CH3_REG	RX CH3 仲裁配置寄存器	0x0840	R/W
H264_DMA_IN_ETM_CONF_CH3_REG	RX CH3 ETM 配置寄存器	0x0848	R/W
H264_DMA_IN_CONF0_CH4_REG	RX CH4 config0 寄存器	0x0900	R/W
H264_DMA_IN_POP_CH4_REG	RX CH4 INFIFO pop 寄存器	0x0918	varies
H264_DMA_IN_LINK_CONF_CH4_REG	RX CH4 in_link 控制寄存器	0x091C	varies
H264_DMA_IN_LINK_ADDR_CH4_REG	RX CH4 in_link 地址寄存器	0x0920	R/W

名称	描述	地址	访问
H264_DMA_IN_ARB_CH4_REG	RX CH4 仲裁配置寄存器	0x0940	R/W
H264_DMA_IN_ETM_CONF_CH4_REG	RX CH4 ETM 配置寄存器	0x0948	R/W
H264_DMA_IN_CONF0_CH5_REG	RX CH5 config0 寄存器	0x0A00	R/W
H264_DMA_IN_CONF1_CH5_REG	RX CH5 config1 寄存器	0x0A04	R/W
H264_DMA_IN_CONF2_CH5_REG	RX CH5 config2 寄存器	0x0A08	R/W
H264_DMA_IN_CONF3_CH5_REG	RX CH5 config3 寄存器	0x0A0C	R/W
H264_DMA_IN_POP_CH5_REG	RX CH5 INFIFO pop 寄存器	0x0A24	varies
H264_DMA_IN_ARB_CH5_REG	RX CH5 仲裁配置寄存器	0x0A40	R/W
其他配置寄存器			
H264_DMA_RST_CONF_REG	AXI 重置配置寄存器	0x0B08	R/W
H264_DMA_INTER_MEM_START_ADDR0_REG	inter memory range0 寄存器的起始地址	0x0B0C	R/W
H264_DMA_INTER_MEM_END_ADDR0_REG	inter memory range0 寄存器的结束地址	0x0B10	R/W
H264_DMA_INTER_MEM_START_ADDR1_REG	inter memory range1 寄存器的起始地址	0x0B14	R/W
H264_DMA_INTER_MEM_END_ADDR1_REG	inter memory range1 寄存器的结束地址	0x0B18	R/W
H264_DMA_EXTER_MEM_START_ADDR0_REG	exter memory range0 寄存器的起始地址	0x0B20	R/W
H264_DMA_EXTER_MEM_END_ADDR0_REG	exter memory range0 寄存器的结束地址	0x0B24	R/W
H264_DMA_EXTER_MEM_START_ADDR1_REG	exter memory range1 寄存器的起始地址	0x0B28	R/W
H264_DMA_EXTER_MEM_END_ADDR1_REG	exter memory range1 寄存器的结束地址	0x0B2C	R/W
H264_DMA_OUT_ARB_CONFIG_REG	保留	0x0B30	R/W
H264_DMA_IN_ARB_CONFIG_REG	保留	0x0B34	R/W
H264_DMA_COUNTER_RST_REG	计数器重置寄存器	0x0B50	R/W
H264_DMA_PBYTE_REG	图像 pbyte 寄存器	0x0B64	R/W
H264_DMA_CH_DBG_EN_REG	通道调试使能寄存器	0x0B68	R/W
H264_DMA_OUT_CH0_DBG_DATA_L_REG	输出通道 0 调试数据寄存器	0x0B6C	R/W
H264_DMA_OUT_CH0_DBG_DATA_H_REG	输出通道 0 调试数据寄存器	0x0B70	R/W
H264_DMA_OUT_CH1_DBG_DATA_L_REG	输出通道 1 调试数据寄存器	0x0B74	R/W
H264_DMA_OUT_CH1_DBG_DATA_H_REG	输出通道 1 调试数据寄存器	0x0B78	R/W
H264_DMA_OUT_CH2_DBG_DATA_L_REG	输出通道 2 调试数据寄存器	0x0B7C	R/W
H264_DMA_OUT_CH2_DBG_DATA_H_REG	输出通道 2 调试数据寄存器	0x0B80	R/W
H264_DMA_OUT_CH3_DBG_DATA_L_REG	输出通道 3 调试数据寄存器	0x0B84	R/W
H264_DMA_OUT_CH3_DBG_DATA_H_REG	输出通道 3 调试数据寄存器	0x0B88	R/W
H264_DMA_OUT_CH4_DBG_DATA_L_REG	输出通道 4 调试数据寄存器	0x0B8C	R/W
H264_DMA_OUT_CH4_DBG_DATA_H_REG	输出通道 4 调试数据寄存器	0x0B90	R/W
H264_DMA_IN_CH0_DBG_DATA_L_REG	输入通道 0 调试数据寄存器	0x0B94	R/W
H264_DMA_IN_CH0_DBG_DATA_H_REG	输入通道 0 调试数据寄存器	0x0B98	R/W

名称	描述	地址	访问
H264_DMA_IN_CH1_DBG_DATA_L_REG	输入通道 1 调试数据寄存器	0x0B9C	R/W
H264_DMA_IN_CH1_DBG_DATA_H_REG	输入通道 1 调试数据寄存器	0x0BA0	R/W
H264_DMA_IN_CH2_DBG_DATA_L_REG	输入通道 2 调试数据寄存器	0x0BA4	R/W
H264_DMA_IN_CH2_DBG_DATA_H_REG	输入通道 2 调试数据寄存器	0x0BA8	R/W
H264_DMA_IN_CH3_DBG_DATA_L_REG	输入通道 3 调试数据寄存器	0x0BAC	R/W
H264_DMA_IN_CH3_DBG_DATA_H_REG	输入通道 3 调试数据寄存器	0x0BB0	R/W
H264_DMA_IN_CH4_DBG_DATA_L_REG	输入通道 4 调试数据寄存器	0x0BB4	R/W
H264_DMA_IN_CH4_DBG_DATA_H_REG	输入通道 4 调试数据寄存器	0x0BB8	R/W
H264_DMA_IN_CH5_DBG_DATA_L_REG	输入通道 5 调试数据寄存器	0x0BBC	R/W
H264_DMA_IN_CH5_DBG_DATA_H_REG	输入通道 5 调试数据寄存器	0x0BC0	R/W
中断寄存器			
TX 中断寄存器			
H264_DMA_OUT_INT_RAW_CH0_REG	TX CH0 中断原始寄存器	0x0004	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH0_REG	TX CH0 中断使能寄存器	0x0008	R/W
H264_DMA_OUT_INT_ST_CH0_REG	TX CH0 中断状态寄存器	0x000C	RO
H264_DMA_OUT_INT_CLR_CH0_REG	TX CH0 中断清除寄存器	0x0010	WT
H264_DMA_OUT_INT_RAW_CH1_REG	TX CH1 中断原始寄存器	0x0104	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH1_REG	TX CH1 中断使能寄存器	0x0108	R/W
H264_DMA_OUT_INT_ST_CH1_REG	TX CH1 中断状态寄存器	0x010C	RO
H264_DMA_OUT_INT_CLR_CH1_REG	TX CH1 中断清除寄存器	0x0110	WT
H264_DMA_OUT_INT_RAW_CH2_REG	TX CH2 中断原始寄存器	0x0204	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH2_REG	TX CH2 中断使能寄存器	0x0208	R/W
H264_DMA_OUT_INT_ST_CH2_REG	TX CH2 中断状态寄存器	0x020C	RO
H264_DMA_OUT_INT_CLR_CH2_REG	TX CH2 中断清除寄存器	0x0210	WT
H264_DMA_OUT_INT_RAW_CH3_REG	TX CH3 中断原始寄存器	0x0304	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH3_REG	TX CH3 中断使能寄存器	0x0308	R/W
H264_DMA_OUT_INT_ST_CH3_REG	TX CH3 中断状态寄存器	0x030C	RO
H264_DMA_OUT_INT_CLR_CH3_REG	TX CH3 中断清除寄存器	0x0310	WT
H264_DMA_OUT_INT_RAW_CH4_REG	TX CH4 中断原始寄存器	0x0404	R/WTC/SS
H264_DMA_OUT_INT_ENA_CH4_REG	TX CH4 中断使能寄存器	0x0408	R/W
H264_DMA_OUT_INT_ST_CH4_REG	TX CH4 中断状态寄存器	0x040C	RO
H264_DMA_OUT_INT_CLR_CH4_REG	TX CH4 中断清除寄存器	0x0410	WT
RX 中断寄存器			

名称	描述	地址	访问
H264_DMA_IN_INT_RAW_CH0_REG	RX CH0 中断原始寄存器	0x0504	R/WTC/SS
H264_DMA_IN_INT_ENA_CH0_REG	RX CH0 中断使能寄存器	0x0508	R/W
H264_DMA_IN_INT_ST_CH0_REG	RX CH0 中断状态寄存器	0x050C	RO
H264_DMA_IN_INT_CLR_CH0_REG	RX CH0 中断清除寄存器	0x0510	WT
H264_DMA_IN_INT_RAW_CH1_REG	RX CH1 中断原始寄存器	0x0604	R/WTC/SS
H264_DMA_IN_INT_ENA_CH1_REG	RX CH1 中断使能寄存器	0x0608	R/W
H264_DMA_IN_INT_ST_CH1_REG	RX CH1 中断状态寄存器	0x060C	RO
H264_DMA_IN_INT_CLR_CH1_REG	RX CH1 中断清除寄存器	0x0610	WT
H264_DMA_IN_INT_RAW_CH2_REG	RX CH2 中断原始寄存器	0x0704	R/WTC/SS
H264_DMA_IN_INT_ENA_CH2_REG	RX CH2 中断使能寄存器	0x0708	R/W
H264_DMA_IN_INT_ST_CH2_REG	RX CH2 中断状态寄存器	0x070C	RO
H264_DMA_IN_INT_CLR_CH2_REG	RX CH2 中断清除寄存器	0x0710	WT
H264_DMA_IN_INT_RAW_CH3_REG	RX CH3 中断原始寄存器	0x0804	R/WTC/SS
H264_DMA_IN_INT_ENA_CH3_REG	RX CH3 中断使能寄存器	0x0808	R/W
H264_DMA_IN_INT_ST_CH3_REG	RX CH3 中断状态寄存器	0x080C	RO
H264_DMA_IN_INT_CLR_CH3_REG	RX CH3 中断清除寄存器	0x0810	WT
H264_DMA_IN_INT_RAW_CH4_REG	RX CH4 中断原始寄存器	0x0904	R/WTC/SS
H264_DMA_IN_INT_ENA_CH4_REG	RX CH4 中断使能寄存器	0x0908	R/W
H264_DMA_IN_INT_ST_CH4_REG	RX CH4 中断状态寄存器	0x090C	RO
H264_DMA_IN_INT_CLR_CH4_REG	RX CH4 中断清除寄存器	0x0910	WT
H264_DMA_IN_INT_RAW_CH5_REG	RX CH5 中断原始寄存器	0x0A10	R/WTC/SS
H264_DMA_IN_INT_ENA_CH5_REG	RX CH5 中断使能寄存器	0x0A14	R/W
H264_DMA_IN_INT_ST_CH5_REG	RX CH5 中断状态寄存器	0x0A18	RO
H264_DMA_IN_INT_CLR_CH5_REG	RX CH5 中断清除寄存器	0x0A1C	WT
状态寄存器			
TX 状态寄存器			
H264_DMA_OUTFIFO_STATUS_CH0_REG	TX CH0 outfifo 状态寄存器	0x0014	RO
H264_DMA_OUT_STATE_CH0_REG	TX CH0 状态寄存器	0x0024	RO
H264_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX CH0 EOF 描述符地址寄存器	0x0028	RO
H264_DMA_OUT_DSCR_CH0_REG	TX CH0 下一个描述符地址寄存器	0x002C	RO
H264_DMA_OUT_DSCR_BF0_CH0_REG	TX CH0 最后一个描述符地址寄存器	0x0030	RO

名称	描述	地址	访问
H264_DMA_OUT_DSCR_BF1_CH0_REG	TX CH0 倒数第二个描述符地址寄存器	0x0034	RO
H264_DMA_OUT_RO_STATUS_CH0_REG	TX CH0 重排序状态寄存器	0x0040	RO
H264_DMA_OUT_BUF_LEN_CH0_REG	TX CH0 缓冲区长度寄存器	0x0070	RO
H264_DMA_OUT_FIFO_BCNT_CH0_REG	TX CH0 fifo 字节计数寄存器	0x0074	RO
H264_DMA_OUT_PUSH_BYTECNT_CH0_REG	TX CH0 推送字节计数寄存器	0x0078	RO
H264_DMA_OUT_XADDR_CH0_REG	TX CH0 xaddr 寄存器	0x007C	RO
H264_DMA_OUTFIFO_STATUS_CH1_REG	TX CH1 outfifo 状态寄存器	0x0114	RO
H264_DMA_OUT_STATE_CH1_REG	TX CH1 状态寄存器	0x0124	RO
H264_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX CH1 描述符地址寄存器	0x0128	RO
H264_DMA_OUT_DSCR_CH1_REG	TX CH1 下一个描述符地址寄存器	0x012C	RO
H264_DMA_OUT_DSCR_BF0_CH1_REG	TX CH1 最后一个描述符地址寄存器	0x0130	RO
H264_DMA_OUT_DSCR_BF1_CH1_REG	TX CH1 倒数第二个描述符地址寄存器	0x0134	RO
H264_DMA_OUT_BUF_LEN_CH1_REG	TX CH1 缓冲区长度寄存器	0x0170	RO
H264_DMA_OUT_FIFO_BCNT_CH1_REG	TX CH1 fifo 字节计数寄存器	0x0174	RO
H264_DMA_OUT_PUSH_BYTECNT_CH1_REG	TX CH1 推送字节计数寄存器	0x0178	RO
H264_DMA_OUT_XADDR_CH1_REG	TX CH1 xaddr 寄存器	0x017C	RO
H264_DMA_OUTFIFO_STATUS_CH2_REG	TX CH2 outfifo 状态寄存器	0x0214	RO
H264_DMA_OUT_STATE_CH2_REG	TX CH2 状态寄存器	0x0224	RO
H264_DMA_OUT_EOF_DES_ADDR_CH2_REG	TX CH2 EOF 描述符地址寄存器	0x0228	RO
H264_DMA_OUT_DSCR_CH2_REG	TX CH2 下一个描述符地址寄存器	0x022C	RO
H264_DMA_OUT_DSCR_BF0_CH2_REG	TX CH2 最后一个描述符地址寄存器	0x0230	RO
H264_DMA_OUT_DSCR_BF1_CH2_REG	TX CH2 倒数第二个描述符地址寄存器	0x0234	RO
H264_DMA_OUT_BUF_LEN_CH2_REG	TX CH2 缓冲区长度寄存器	0x0270	RO
H264_DMA_OUT_FIFO_BCNT_CH2_REG	TX CH2 FIFO 字节计数寄存器	0x0274	RO
H264_DMA_OUT_PUSH_BYTECNT_CH2_REG	TX CH2 推送字节计数寄存器	0x0278	RO
H264_DMA_OUT_XADDR_CH2_REG	TX CH2 xaddr 寄存器	0x027C	RO
H264_DMA_OUTFIFO_STATUS_CH3_REG	TX CH3 outfifo 状态寄存器	0x0314	RO
H264_DMA_OUT_STATE_CH3_REG	TX CH3 状态寄存器	0x0324	RO
H264_DMA_OUT_EOF_DES_ADDR_CH3_REG	TX CH3 EOF 描述符地址寄存器	0x0328	RO
H264_DMA_OUT_DSCR_CH3_REG	TX CH3 下一个描述符地址寄存器	0x032C	RO
H264_DMA_OUT_DSCR_BF0_CH3_REG	TX CH3 最后一个描述符地址寄存器	0x0330	RO
H264_DMA_OUT_DSCR_BF1_CH3_REG	TX CH3 倒数第二个描述符地址寄存器	0x0334	RO
H264_DMA_OUT_BUF_LEN_CH3_REG	TX CH3 缓冲区长度寄存器	0x0370	RO
H264_DMA_OUT_FIFO_BCNT_CH3_REG	TX CH3 FIFO 字节计数寄存器	0x0374	RO
H264_DMA_OUT_PUSH_BYTECNT_CH3_REG	TX CH3 推送字节计数寄存器	0x0378	RO
H264_DMA_OUT_XADDR_CH3_REG	TX CH3 xaddr 寄存器	0x037C	RO
H264_DMA_OUT_BLOCK_BUF_LEN_CH3_REG	TX CH3 block 缓冲区长度寄存器	0x0380	RO
H264_DMA_OUTFIFO_STATUS_CH4_REG	TX CH4 outfifo 状态寄存器	0x0414	RO
H264_DMA_OUT_STATE_CH4_REG	TX CH4 状态寄存器	0x0424	RO
H264_DMA_OUT_EOF_DES_ADDR_CH4_REG	TX CH4 EOF 描述符地址寄存器	0x0428	RO

名称	描述	地址	访问
H264_DMA_OUT_DSCR_CH4_REG	TX CH4 下一个描述符地址寄存器	0x042C	RO
H264_DMA_OUT_DSCR_BF0_CH4_REG	TX CH4 最后一个描述符地址寄存器	0x0430	RO
H264_DMA_OUT_DSCR_BF1_CH4_REG	TX CH4 倒数第二个描述符地址寄存器	0x0434	RO
H264_DMA_OUT_BUF_LEN_CH4_REG	TX CH4 buf 长度寄存器	0x0470	RO
H264_DMA_OUT_FIFO_BCNT_CH4_REG	TX CH4 fifo 字节计数寄存器	0x0474	RO
H264_DMA_OUT_PUSH_BYTECNT_CH4_REG	TX CH4 推送字节计数寄存器	0x0478	RO
H264_DMA_OUT_XADDR_CH4_REG	TX CH4 xaddr 寄存器	0x047C	RO
H264_DMA_OUT_BLOCK_BUF_LEN_CH4_REG	TX CH4 块 buf 长度寄存器	0x0480	RO
RX 状态寄存器			
H264_DMA_INFIFO_STATUS_CH0_REG	RX CH0 INFIFO 状态寄存器	0x0514	RO
H264_DMA_IN_STATE_CH0_REG	RX CH0 状态寄存器	0x0524	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX CH0 EOF 描述符地址寄存器	0x0528	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX CH0 错误 EOF 描述符地址寄存器	0x052C	RO
H264_DMA_IN_DSCR_CH0_REG	RX CH0 下一个描述符地址寄存器	0x0530	RO
H264_DMA_IN_DSCR_BF0_CH0_REG	RX CH0 最后一个描述符地址寄存器	0x0534	RO
H264_DMA_IN_DSCR_BF1_CH0_REG	RX CH0 倒数第二个描述符地址寄存器	0x0538	RO
H264_DMA_IN_FIFO_CNT_CH0_REG	RX CH0 fifo 计数寄存器	0x0580	RO
H264_DMA_IN_POP_DATA_CNT_CH0_REG	RX CH0 pop 数据计数寄存器	0x0584	RO
H264_DMA_IN_XADDR_CH0_REG	RX CH0 xaddr 寄存器	0x0588	RO
H264_DMA_IN_BUF_HB_RCV_CH0_REG	RX CH0 buf 长度 hb rcv 寄存器	0x058C	RO
H264_DMA_INFIFO_STATUS_CH1_REG	RX CH1 INFIFO 状态寄存器	0x0614	RO
H264_DMA_IN_STATE_CH1_REG	RX CH1 状态寄存器	0x0624	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX CH1 EOF 描述符地址寄存器	0x0628	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX CH1 错误描述符地址寄存器	0x062C	RO
H264_DMA_IN_DSCR_CH1_REG	RX CH1 下一个 dscr 地址寄存器	0x0630	RO
H264_DMA_IN_DSCR_BF0_CH1_REG	RX CH1 最后一个 dscr 地址寄存器	0x0634	RO
H264_DMA_IN_DSCR_BF1_CH1_REG	RX CH1 倒数第二个 dscr 地址寄存器	0x0638	RO
H264_DMA_IN_FIFO_CNT_CH1_REG	RX CH1 fifo 计数寄存器	0x0680	RO
H264_DMA_IN_POP_DATA_CNT_CH1_REG	RX CH1 pop 数据计数寄存器	0x0684	RO
H264_DMA_IN_XADDR_CH1_REG	RX CH1 xaddr 寄存器	0x0688	RO
H264_DMA_IN_BUF_HB_RCV_CH1_REG	RX CH1 缓冲区长度 HB 接收寄存器	0x068C	RO
H264_DMA_INFIFO_STATUS_CH2_REG	RX CH2 INFIFO 状态寄存器	0x0714	RO
H264_DMA_IN_STATE_CH2_REG	RX CH2 状态寄存器	0x0724	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX CH2 成功 EOF 描述符地址寄存器	0x0728	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	RX CH2 错误 EOF 描述符地址寄存器	0x072C	RO
H264_DMA_IN_DSCR_CH2_REG	RX CH2 下一个描述符地址寄存器	0x0730	RO
H264_DMA_IN_DSCR_BF0_CH2_REG	RX CH2 最后一个描述符地址寄存器	0x0734	RO
H264_DMA_IN_DSCR_BF1_CH2_REG	RX CH2 倒数第二个 dscr addr 寄存器	0x0738	RO
H264_DMA_IN_FIFO_CNT_CH2_REG	RX CH2 fifo cnt 寄存器	0x0780	RO
H264_DMA_IN_POP_DATA_CNT_CH2_REG	RX CH2 pop data cnt 寄存器	0x0784	RO
H264_DMA_IN_XADDR_CH2_REG	RX CH2 xaddr 寄存器	0x0788	RO

名称	描述	地址	访问
H264_DMA_IN_BUF_HB_RCV_CH2_REG	RX CH2 缓冲区长度 HB 接收寄存器	0x078C	RO
H264_DMA_INFIFO_STATUS_CH3_REG	RX CH3 INFIFO 状态寄存器	0x0814	RO
H264_DMA_IN_STATE_CH3_REG	RX CH3 状态寄存器	0x0824	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH3_REG	RX CH3 成功 EOF 描述符地址寄存器	0x0828	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH3_REG	RX CH3 错误 EOF 描述符地址寄存器	0x082C	RO
H264_DMA_IN_DSCR_CH3_REG	RX CH3 下一个描述符地址寄存器	0x0830	RO
H264_DMA_IN_DSCR_BF0_CH3_REG	RX CH3 最后一个描述符地址寄存器	0x0834	RO
H264_DMA_IN_DSCR_BF1_CH3_REG	RX CH3 倒数第二个描述符地址寄存器	0x0838	RO
H264_DMA_IN_FIFO_CNT_CH3_REG	RX CH3 FIFO 计数器寄存器	0x0880	RO
H264_DMA_IN_POP_DATA_CNT_CH3_REG	RX CH3 pop data cnt 寄存器	0x0884	RO
H264_DMA_IN_XADDR_CH3_REG	RX CH3 xaddr 寄存器	0x0888	RO
H264_DMA_IN_BUF_HB_RCV_CH3_REG	RX CH3 缓冲区长度 HB 接收寄存器	0x088C	RO
H264_DMA_INFIFO_STATUS_CH4_REG	RX CH4 INFIFO 状态寄存器	0x0914	RO
H264_DMA_IN_STATE_CH4_REG	RX CH4 状态寄存器	0x0924	RO
H264_DMA_IN_SUC_EOF_DES_ADDR_CH4_REG	RX CH4 EOF 描述符地址寄存器	0x0928	RO
H264_DMA_IN_ERR_EOF_DES_ADDR_CH4_REG	RX CH4 错误 EOF 描述符地址寄存器	0x092C	RO
H264_DMA_IN_DSCR_CH4_REG	RX CH4 下一个描述符地址寄存器	0x0930	RO
H264_DMA_IN_DSCR_BF0_CH4_REG	RX CH4 最后一个描述符地址寄存器	0x0934	RO
H264_DMA_IN_DSCR_BF1_CH4_REG	RX CH4 倒数第二个描述符地址寄存器	0x0938	RO
H264_DMA_IN_FIFO_CNT_CH4_REG	RX CH4 FIFO 计数器寄存器	0x0980	RO
H264_DMA_IN_POP_DATA_CNT_CH4_REG	RX CH4 pop 数据计数器寄存器	0x0984	RO
H264_DMA_IN_XADDR_CH4_REG	RX CH4 xaddr 寄存器	0x0988	RO
H264_DMA_IN_BUF_HB_RCV_CH4_REG	RX CH4 缓冲区长度 HB 接收寄存器	0x098C	RO
H264_DMA_INFIFO_STATUS_CH5_REG	RX CH5 INFIFO 状态寄存器	0x0A20	RO
H264_DMA_IN_STATE_CH5_REG	RX CH5 状态寄存器	0x0A28	RO
H264_DMA_IN_FIFO_CNT_CH5_REG	RX CH5 FIFO 计数器寄存器	0x0A80	RO
H264_DMA_IN_POP_DATA_CNT_CH5_REG	RX CH5 pop 数据计数器寄存器	0x0A84	RO
H264_DMA_IN_XADDR_CH5_REG	RX CH5 xaddr 寄存器	0x0A88	RO
H264_DMA_IN_BUF_HB_RCV_CH5_REG	RX CH5 缓冲区长度 HB 接收寄存器	0x0A8C	RO
H264_DMA_INTER_AXI_ERR_REG	inter memory AXI 错误寄存器	0x0B00	RO
H264_DMA_EXTER_AXI_ERR_REG	exter memory AXI 错误寄存器	0x0B04	RO
H264_DMA_RX_CH0_COUNTER_REG	RX ch0 计数器寄存器	0x0B54	RO
H264_DMA_RX_CH1_COUNTER_REG	RX ch1 计数器寄存器	0x0B58	RO
H264_DMA_RX_CH2_COUNTER_REG	RX ch2 计数器寄存器	0x0B5C	RO
H264_DMA_RX_CH5_COUNTER_REG	RX ch5 计数器寄存器	0x0B60	RO
版本寄存器			
H264_DMA_DATE_REG	版本控制寄存器	0x0B3C	R/W

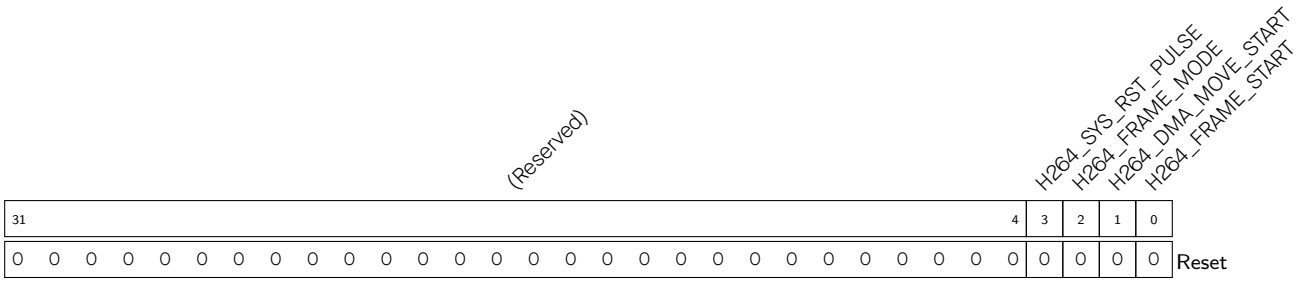
41.9 寄存器

本节中的地址是相对于 **H264 编码器**基地址和 **H264 DMA** 基地址的相对地址。如需了解基地址详细信息，请参考 9 系统和存储器 章节，表 9.3-2 模块/外设地址空间映射表。

如需了解如何对保留字段进行编程，请参考章节 寄存器保留字段编程。

41.9.1 H264 编码器寄存器

Register 41.1. H264_SYS_CTRL_REG (0x0000)



H264_FRAME_START 配置是否开始编码一帧图像。

0：无效值。没有作用

1：开始编码一帧图像

(WT)

H264_DMA_MOVE_START 配置是否开始从外部存储移动参考数据。

0：无效值。没有作用

1：H264 开始从外部存储向内部存储移动两行 MB 的参考帧

(WT)

H264_FRAME_MODE 配置 H264 运行模式。当字段 H264_DUAL_STREAM_MODE 设置为 1 时，此字段也必须设置为 1。

0：GOP 模式。在每个 GOP 开始之前，需要重新配置 H264_DMA

1：帧模式。在每帧开始之前，需要重新配置 H264_DMA

(R/W)

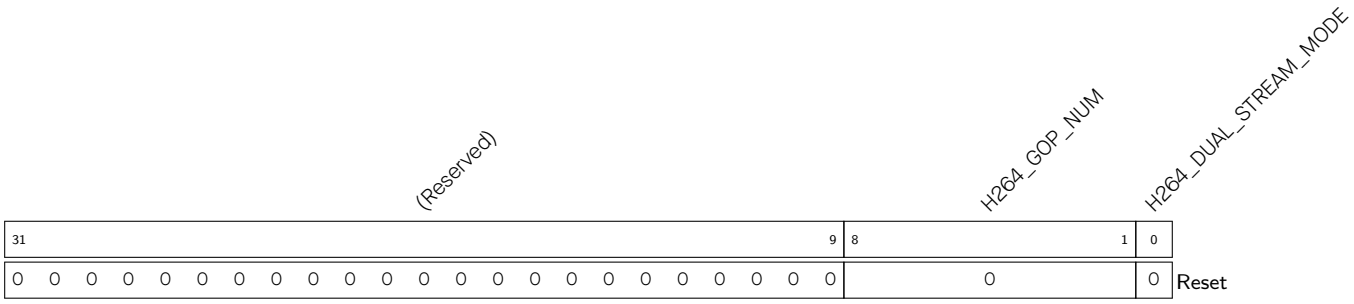
H264_SYS_RST_PULSE 配置是否重置 H264 encoder。

0：无效值。没有作用

1：重置 H264 encoder

(WT)

Register 41.2. H264_GOP_CONF_REG (0x0004)



H264_DUAL_STREAM_MODE 配置是否启用双流模式。当此字段设置为 1 时，H264_FRAME_MODE 字段也必须设置为 1。

0：GOP 模式

1：双流模式

(R/W)

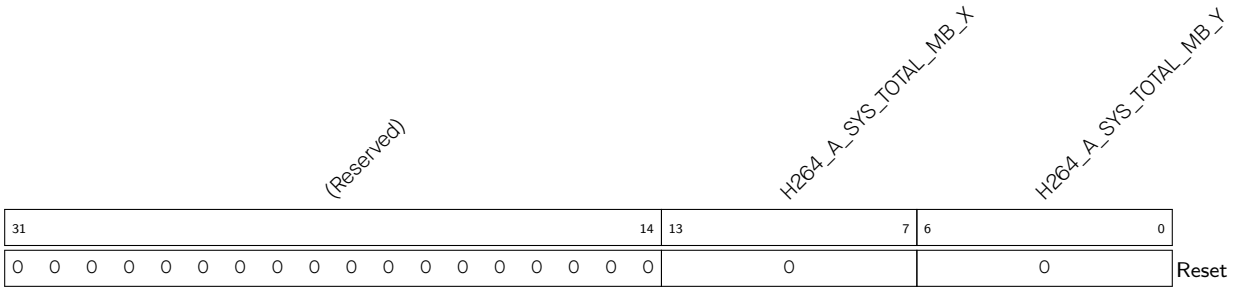
H264_GOP_NUM 配置一个 GOP 的帧数。

0：不限制 GOP 帧数

其他：一个 GOP 的实际帧数

(R/W)

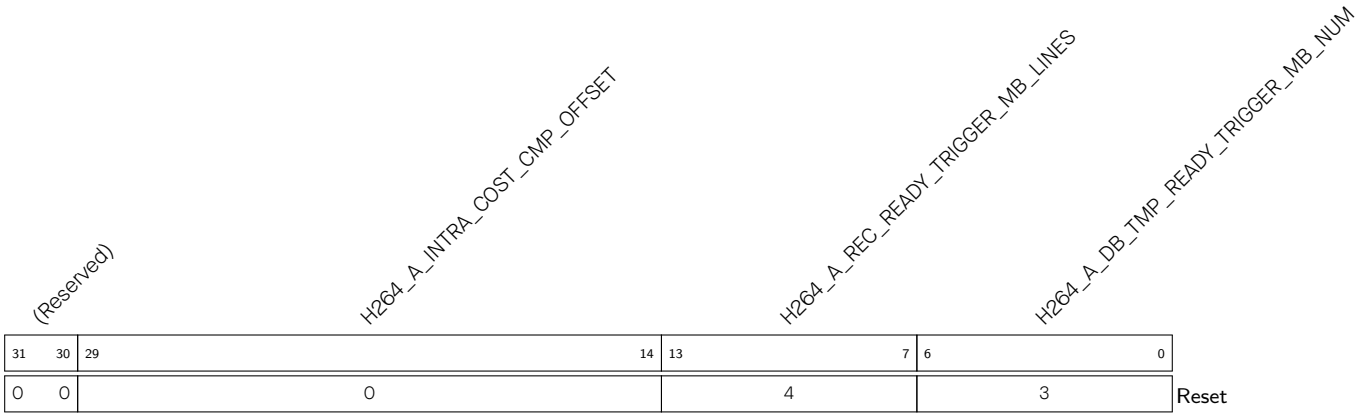
Register 41.3. H264_A_SYS_MB_RES_REG (0x0008)



H264_A_SYS_TOTAL_MB_Y 配置视频序列 A 的垂直 MB 分辨率。(R/W)

H264_A_SYS_TOTAL_MB_X 配置视频序列 A 的水平 MB 分辨率。(R/W)

Register 41.4. H264_A_SYS_CONF_REG (0x000C)

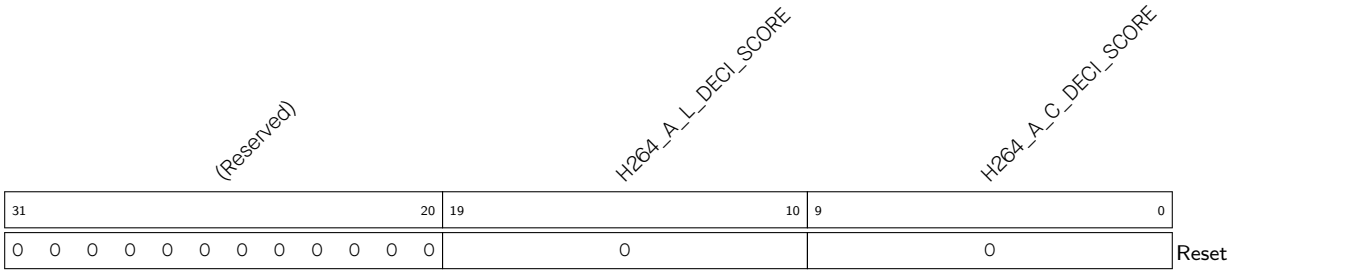


H264_A_DB_TMP_READY_TRIGGER_MB_NUM 配置何时触发视频序列 A 的 H264_DB_TMP_READY_INT。当（写入 db temp 的 MB 数量 +1）大于第一行 MB 中的此字段时，触发 H264_DB_TMP_READY_INT。最小值为 3。(R/W)

H264_A_REC_READY_TRIGGER_MB_LINES 配置何时触发视频序列 A 的 H264_REC_READY_INT。当生成的重建像素的 MB 行数大于此字段时，触发 H264_REC_READY_INT。最小值为 4。(R/W)

H264_A_INTRA_COST_CMP_OFFSET 配置视频序列 A 的 I MB 与 P MB 比较时的帧内成本偏移。(R/W)

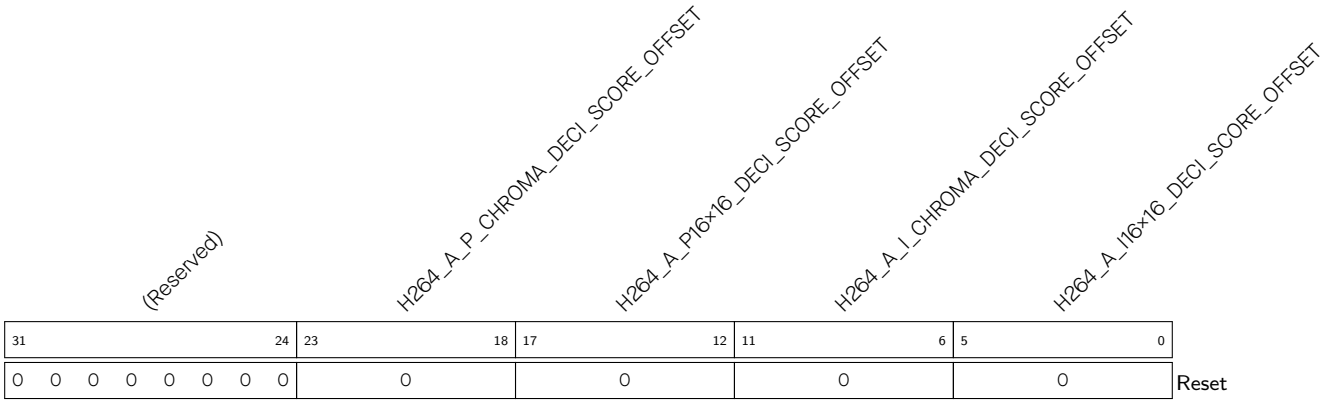
Register 41.5. H264_A_DECI_SCORE_REG (0x0010)



H264_A_C_DECI_SCORE 配置视频序列 A 的色度 MB 降采样分数。当色度分数小于此值时，将启用色度降采样。(R/W)

H264_A_L_DECI_SCORE 配置视频序列 A 的亮度 MB 降采样分数。当亮度分数小于此值时，将启用亮度降采样。(R/W)

Register 41.6. H264_A_DECI_SCORE_OFFSET_REG (0x0014)



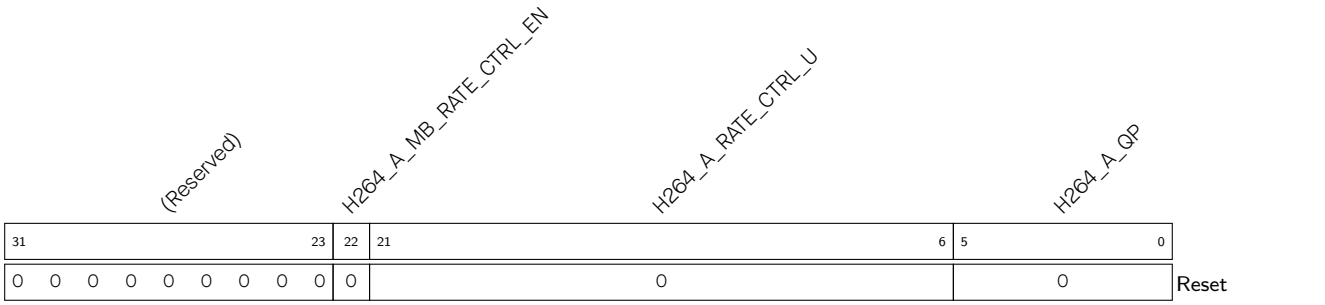
H264_A_I16×16_DECI_SCORE_OFFSET 配置视频序列 A 的 i16 × 16 MB 降采样分数偏移。此偏移将添加到 i16 × 16 MB 分数中。(R/W)

H264_A_I_CHROMA_DECI_SCORE_OFFSET 配置视频序列 A 的 I 色度 MB 降采样分数偏移。此偏移将添加到 I 色度 MB 分数中。(R/W)

H264_A_P16×16_DECI_SCORE_OFFSET 配置视频序列 A 的 p16 × 16 MB 降采样分数偏移。此偏移将添加到 p16 × 16 MB 分数中。(R/W)

H264_A_P_CHROMA_DECI_SCORE_OFFSET 配置视频序列 A 的 p 色度 MB 降采样分数偏移。此偏移将添加到 p 色度 MB 分数中。(R/W)

Register 41.7. H264_A_RC_CONF0_REG (0x0018)



H264_A_QP 配置视频序列 A 帧级初始亮度 QP 值。(R/W)

H264_A_RATE_CTRL_U 配置视频序列 A 的参数 U 值。U = int((float) u « 8)。(R/W)

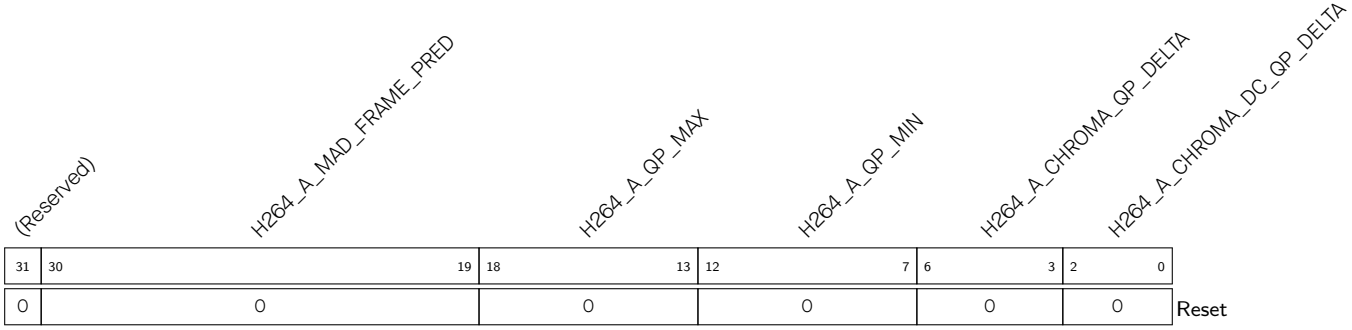
H264_A_MB_RATE_CTRL_EN 配置视频序列 A 是否启用 MB 级速率控制。

0：禁用 MB 级速率控制

1：启用 MB 级速率控制

(R/W)

Register 41.8. H264_A_RC_CONF1_REG (0x001C)



H264_A_CHROMA_DC_QP_DELTA 基于 Chroma QP 配置视频序列 A 的色度 DC QP 偏移。色度 DC QP = Chroma QP（映射后）+ reg_chroma_dc_QP_delta。(R/W)

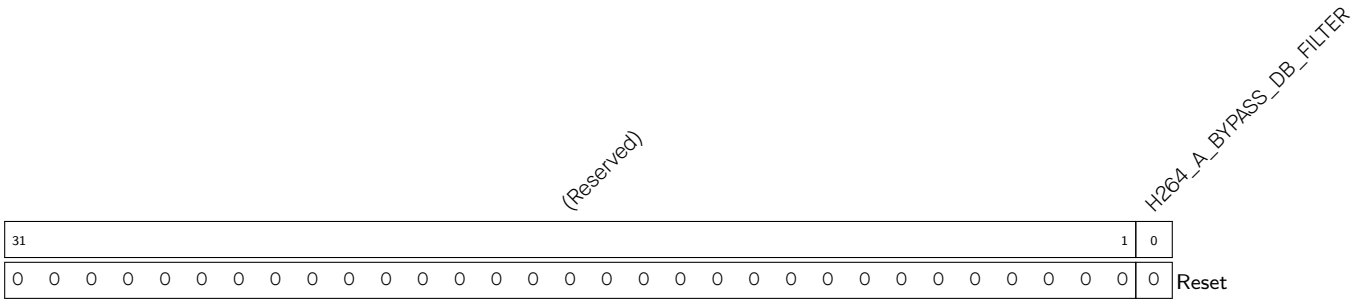
H264_A_CHROMA_QP_DELTA 基于亮度 QP 配置视频序列 A 的色度 QP 偏移。色度 QP（映射前）= 亮度 QP + reg_chroma_QP_delta。(R/W)

H264_A_QP_MIN 配置视频序列 A 允许的亮度 QP 最小值。(R/W)

H264_A_QP_MAX 配置视频序列 A 允许的亮度 QP 最大值。(R/W)

H264_A_MAD_FRAME_PRED 配置视频序列 A 帧级预测 MB MAD 值。(R/W)

Register 41.9. H264_A_DB_BYPASS_REG (0x0020)



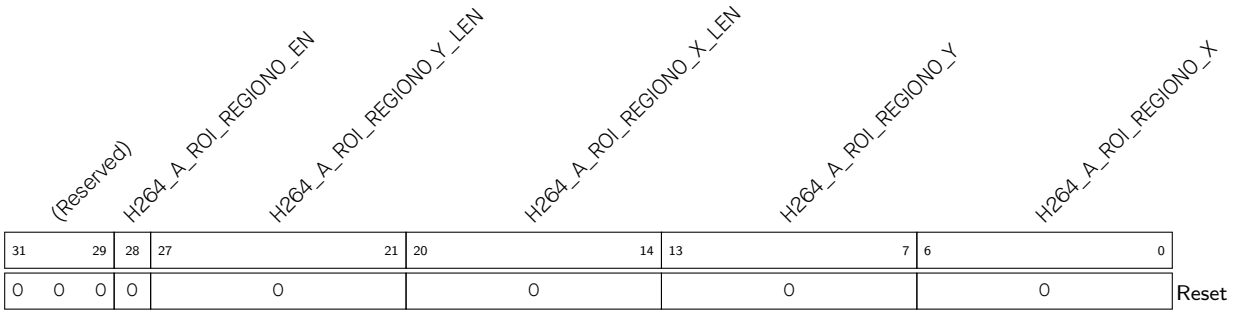
H264_A_BYPASS_DB_FILTER 配置是否绕过视频序列 A 的去块滤波器。

0：不绕过去块滤波器

1：绕过去块滤波器

(R/W)

Register 41.10. H264_A_ROI_REGION0_REG (0x0024)



H264_A_ROI_REGION0_X 配置视频序列 A 中 ROI 0 的水平起始 MB。(R/W)

H264_A_ROI_REGION0_Y 配置视频序列 A 中 ROI 0 的垂直起始 MB。(R/W)

H264_A_ROI_REGION0_X_LEN 配置视频序列 A 中 ROI 0 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION0_Y_LEN 配置视频序列 A 中 ROI 0 的垂直方向 MB 数量。(R/W)

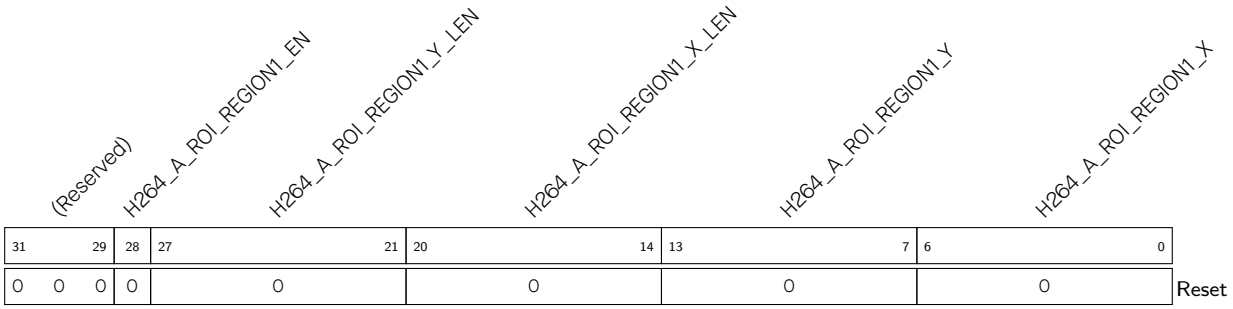
H264_A_ROI_REGION0_EN 配置是否启用视频序列 A 的 ROI 0。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.11. H264_A_ROI_REGION1_REG (0x0028)



H264_A_ROI_REGION1_X 配置视频序列 A 中 ROI 1 的水平起始 MB。(R/W)

H264_A_ROI_REGION1_Y 配置视频序列 A 中 ROI 1 的垂直起始 MB。(R/W)

H264_A_ROI_REGION1_X_LEN 配置视频序列 A 中 ROI 1 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION1_Y_LEN 配置视频序列 A 中 ROI 1 的垂直方向 MB 数量。(R/W)

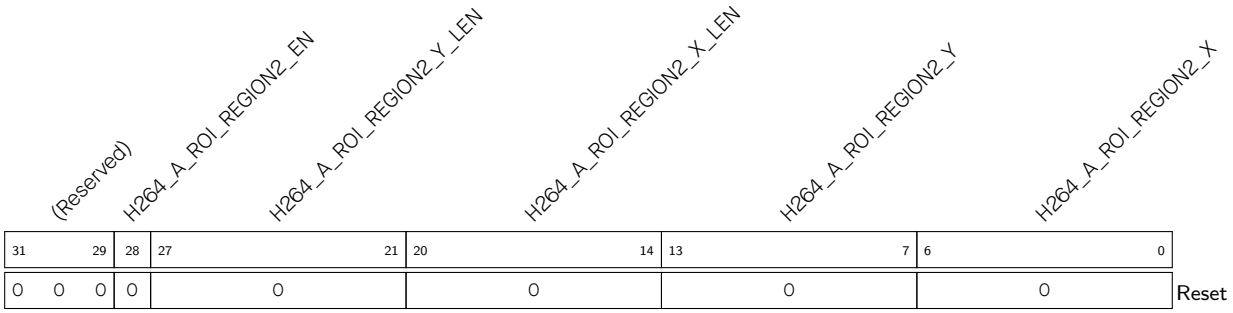
H264_A_ROI_REGION1_EN 配置是否启用视频序列 A 的 ROI 1。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.12. H264_A_ROI_REGION2_REG (0x002C)



H264_A_ROI_REGION2_X 配置视频序列 A 中 ROI 2 的水平起始 MB。(R/W)

H264_A_ROI_REGION2_Y 配置视频序列 A 中 ROI 2 的垂直起始 MB。(R/W)

H264_A_ROI_REGION2_X_LEN 配置视频序列 A 中 ROI 2 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION2_Y_LEN 配置视频序列 A 中 ROI 2 的垂直方向 MB 数量。(R/W)

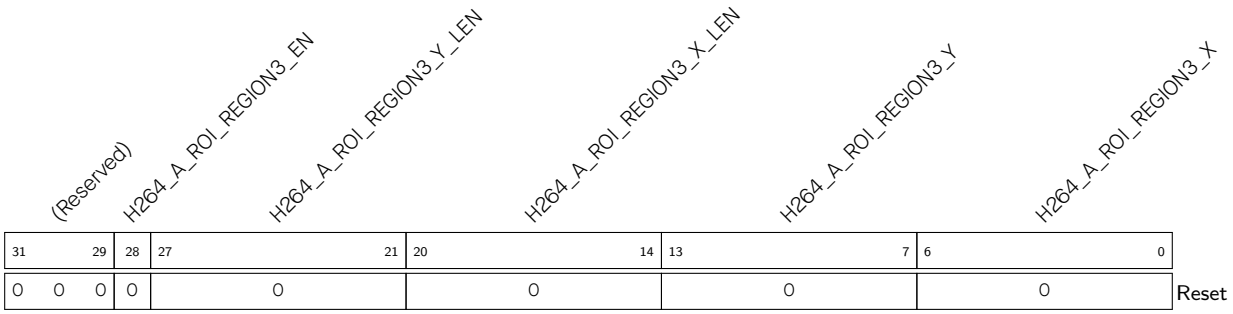
H264_A_ROI_REGION2_EN 配置是否启用视频序列 A 的 ROI 2。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.13. H264_A_ROI_REGION3_REG (0x0030)



H264_A_ROI_REGION3_X 配置视频序列 A 中 ROI 3 的水平起始 MB。(R/W)

H264_A_ROI_REGION3_Y 配置视频序列 A 中 ROI 3 的垂直起始 MB。(R/W)

H264_A_ROI_REGION3_X_LEN 配置视频序列 A 中 ROI 3 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION3_Y_LEN 配置视频序列 A 中 ROI 3 的垂直方向 MB 数量。(R/W)

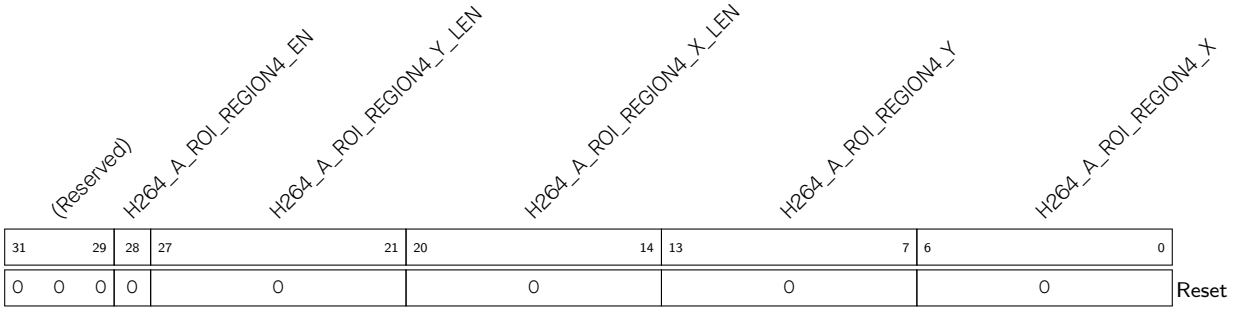
H264_A_ROI_REGION3_EN 配置是否启用视频序列 A 的 ROI 3。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.14. H264_A_ROI_REGION4_REG (0x0034)



H264_A_ROI_REGION4_X 配置视频序列 A 中 ROI 4 的水平起始 MB。(R/W)

H264_A_ROI_REGION4_Y 配置视频序列 A 中 ROI 4 的垂直起始 MB。(R/W)

H264_A_ROI_REGION4_X_LEN 配置视频序列 A 中 ROI 4 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION4_Y_LEN 配置视频序列 A 中 ROI 4 的垂直方向 MB 数量。(R/W)

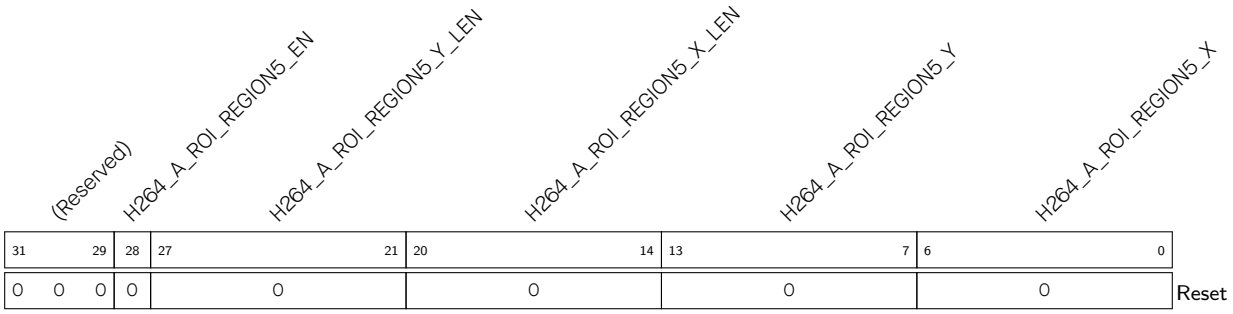
H264_A_ROI_REGION4_EN 配置是否启用视频序列 A 的 ROI 4。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.15. H264_A_ROI_REGION5_REG (0x0038)



H264_A_ROI_REGION5_X 配置视频序列 A 中 ROI 5 的水平起始 MB。(R/W)

H264_A_ROI_REGION5_Y 配置视频序列 A 中 ROI 5 的垂直起始 MB。(R/W)

H264_A_ROI_REGION5_X_LEN 配置视频序列 A 中 ROI 5 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION5_Y_LEN 配置视频序列 A 中 ROI 5 的垂直方向 MB 数量。(R/W)

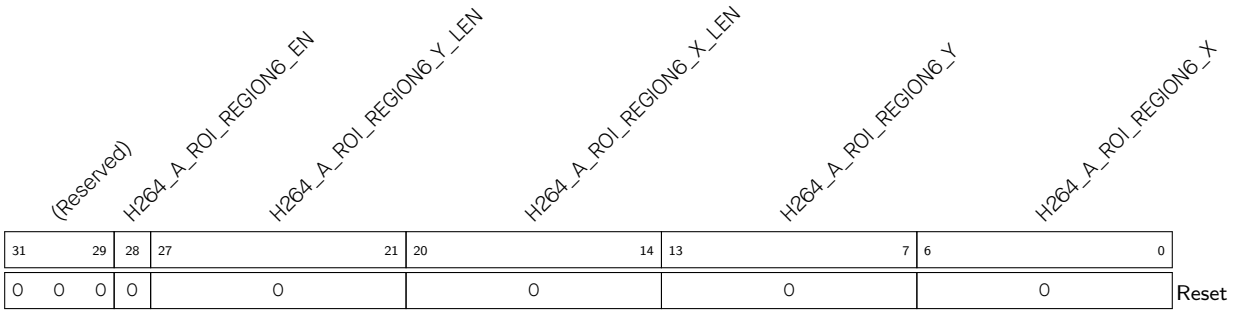
H264_A_ROI_REGION5_EN 配置是否启用视频序列 A 的 ROI 5。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.16. H264_A_ROI_REGION6_REG (0x003C)



H264_A_ROI_REGION6_X 配置视频序列 A 中 ROI 6 的水平起始 MB。(R/W)

H264_A_ROI_REGION6_Y 配置视频序列 A 中 ROI 6 的垂直起始 MB。(R/W)

H264_A_ROI_REGION6_X_LEN 配置视频序列 A 中 ROI 6 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION6_Y_LEN 配置视频序列 A 中 ROI 6 的垂直方向 MB 数量。(R/W)

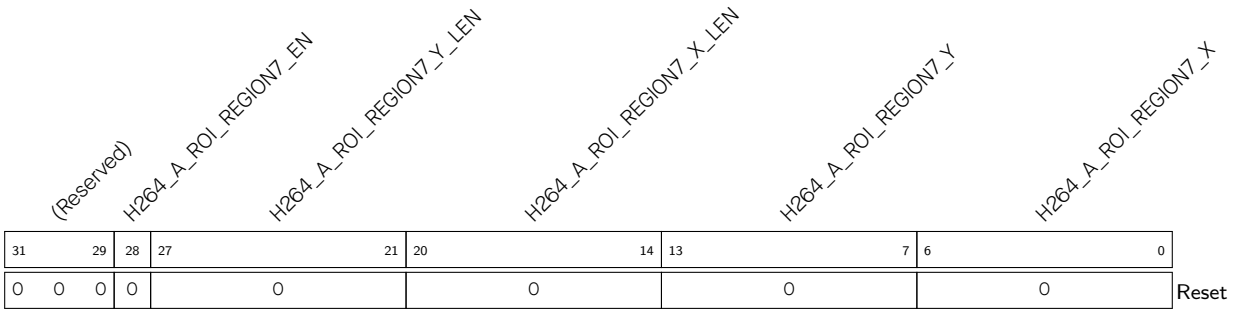
H264_A_ROI_REGION6_EN 配置是否启用视频序列 A 的 ROI 6。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.17. H264_A_ROI_REGION7_REG (0x0040)



H264_A_ROI_REGION7_X 配置视频序列 A 中 ROI 7 的水平起始 MB。(R/W)

H264_A_ROI_REGION7_Y 配置视频序列 A 中 ROI 7 的垂直起始 MB。(R/W)

H264_A_ROI_REGION7_X_LEN 配置视频序列 A 中 ROI 7 的水平方向 MB 数量。(R/W)

H264_A_ROI_REGION7_Y_LEN 配置视频序列 A 中 ROI 7 的垂直方向 MB 数量。(R/W)

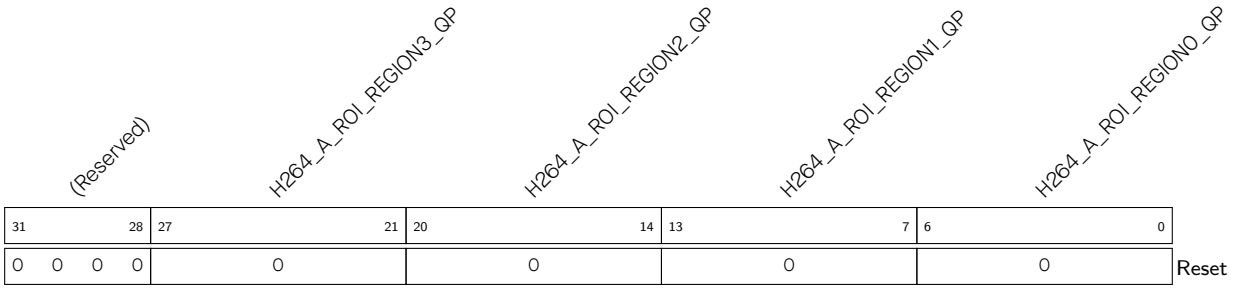
H264_A_ROI_REGION7_EN 配置是否启用视频序列 A 的 ROI 7。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.18. H264_A_ROI_REGION0_3_QP_REG (0x0044)



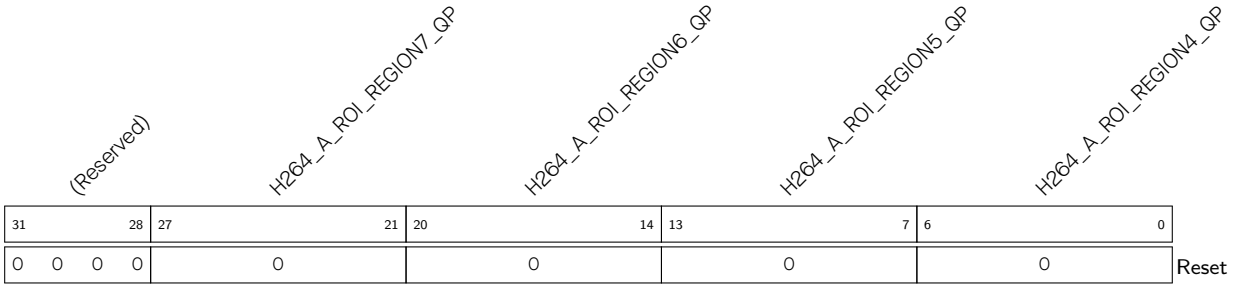
H264_A_ROI_REGION0_QP 配置视频序列 A 中 H264 ROI 0 的 QP 调整量。(R/W)

H264_A_ROI_REGION1_QP 配置视频序列 A 中 H264 ROI 1 的 QP 调整量。(R/W)

H264_A_ROI_REGION2_QP 配置视频序列 A 中 H264 ROI 2 的 QP 调整量。(R/W)

H264_A_ROI_REGION3_QP 配置视频序列 A 中 H264 ROI 3 的 QP 调整量。(R/W)

Register 41.19. H264_A_ROI_REGION4_7_QP_REG (0x0048)



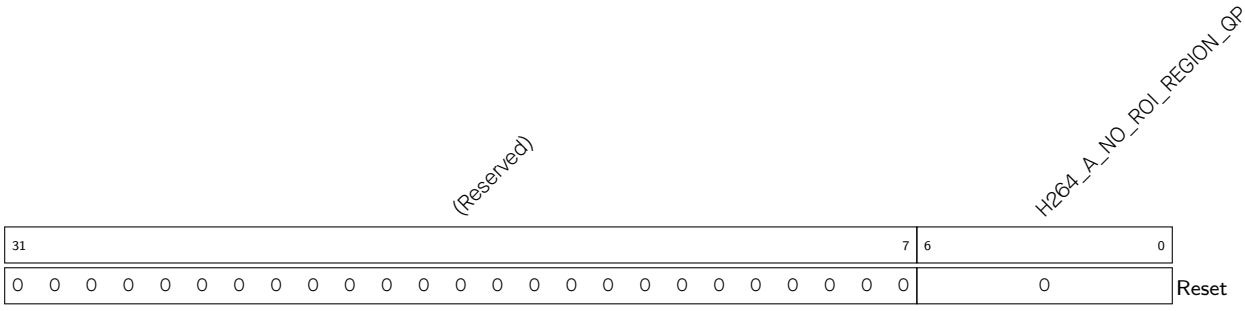
H264_A_ROI_REGION4_QP 配置视频序列 A 中 H264 ROI 4 的 QP 调整量。(R/W)

H264_A_ROI_REGION5_QP 配置视频序列 A 中 H264 ROI 5 的 QP 调整量。(R/W)

H264_A_ROI_REGION6_QP 配置视频序列 A 中 H264 ROI 6 的 QP 调整量。(R/W)

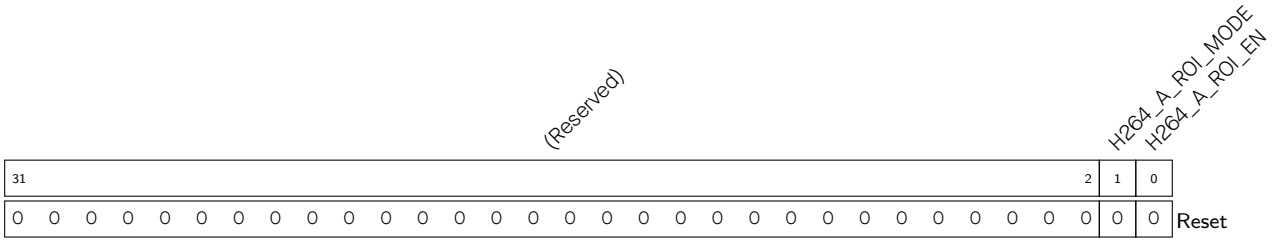
H264_A_ROI_REGION7_QP 配置视频序列 A 中 H264 ROI 7 的 QP 调整量。(R/W)

Register 41.20. H264_A_NO_ROI_REGION_QP_OFFSET_REG (0x004C)



H264_A_NO_ROI_REGION_QP 配置视频序列 A 中 H264 非 ROI 区域的 QP，delta QP。(R/W)

Register 41.21. H264_A_ROI_CONFIG_REG (0x0050)



H264_A_ROI_EN 配置是否在视频序列 A 中启用 ROI。

0：禁用 ROI

1：启用 ROI

(R/W)

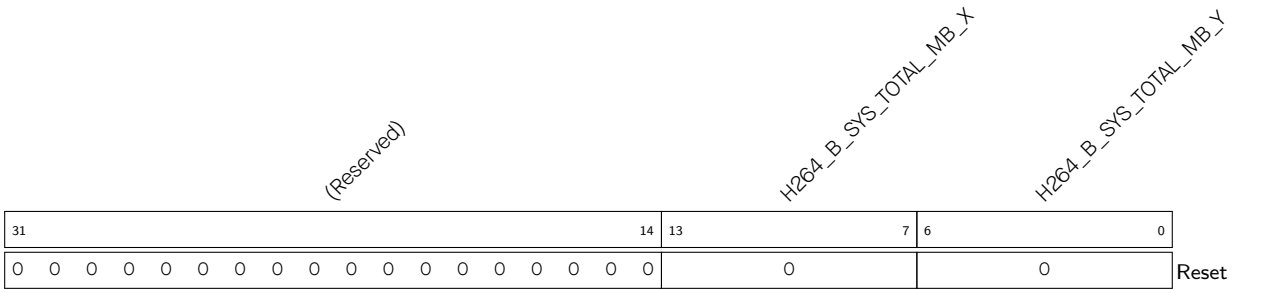
H264_A_ROI_MODE 配置视频序列 A 中 ROI 的模式。

0：固定 QP

1：Delta QP

(R/W)

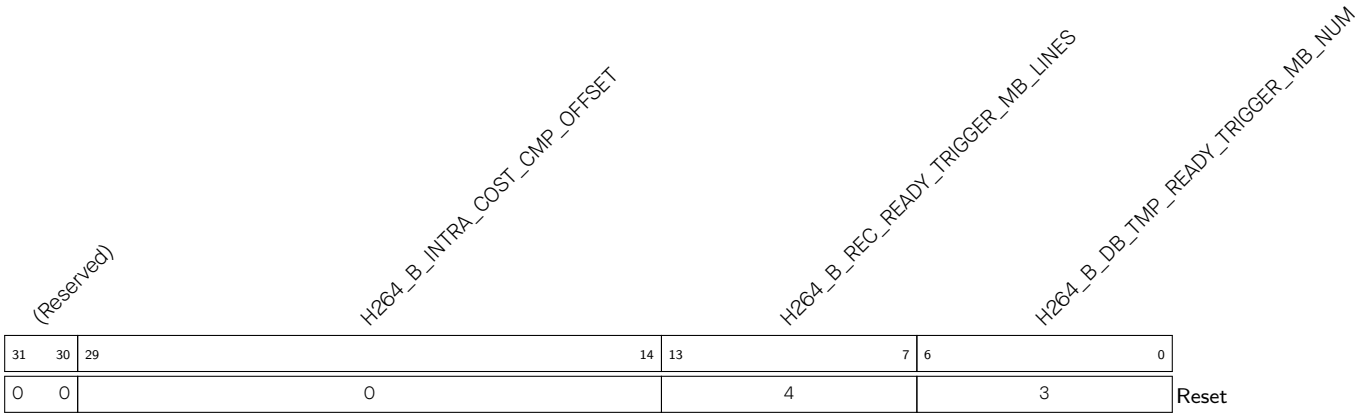
Register 41.22. H264_B_SYS_MB_RES_REG (0x0054)



H264_B_SYS_TOTAL_MB_Y 配置视频序列 B 的垂直 MB 分辨率。(R/W)

H264_B_SYS_TOTAL_MB_X 配置视频序列 B 的水平 MB 分辨率。(R/W)

Register 41.23. H264_B_SYS_CONF_REG (0x0058)

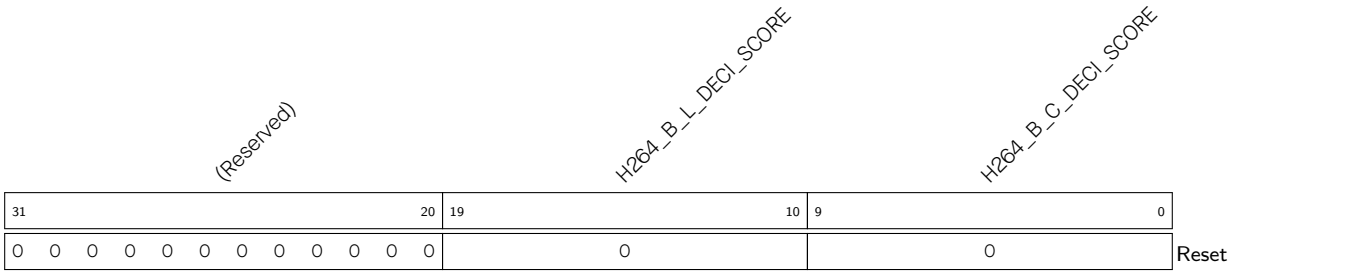


H264_B_DB_TMP_READY_TRIGGER_MB_NUM 配置何时触发视频序列 B 的 H264_DB_TMP_READY_INT。当第一行 MB 的写入 db temp 的 MB 数量 +1 大于此字段时，触发 H264_DB_TMP_READY_INT。最小值为 3。(R/W)

H264_B_REC_READY_TRIGGER_MB_LINES 配置何时触发视频序列 B 的 H264_REC_READY_INT。当生成的重建像素的 MB 行号大于此字段时，触发 H264_REC_READY_INT。最小值为 4。(R/W)

H264_B_INTRA_COST_CMP_OFFSET 配置视频序列 B 中 I MB 与 P MB 比较时的 intra cost 偏移。(R/W)

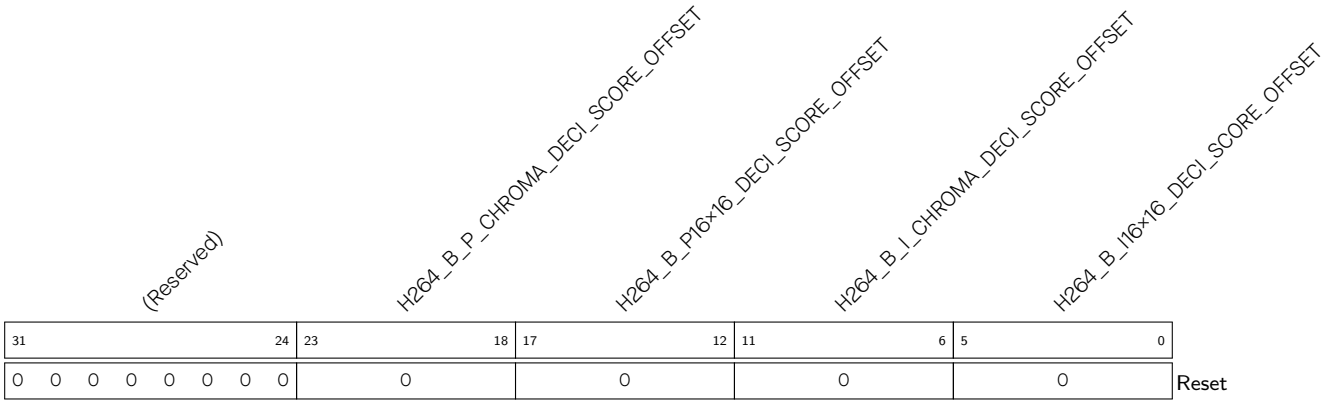
Register 41.24. H264_B_DECI_SCORE_REG (0x005C)



H264_B_C_DECI_SCORE 配置视频序列 B 色度 MB 降采样得分。当色度得分小于此值时，将启用色度降采样。(R/W)

H264_B_L_DECI_SCORE 配置视频序列 B 亮度 MB 降采样得分。当亮度得分小于此值时，将启用亮度降采样。(R/W)

Register 41.25. H264_B_DECI_SCORE_OFFSET_REG (0x0060)



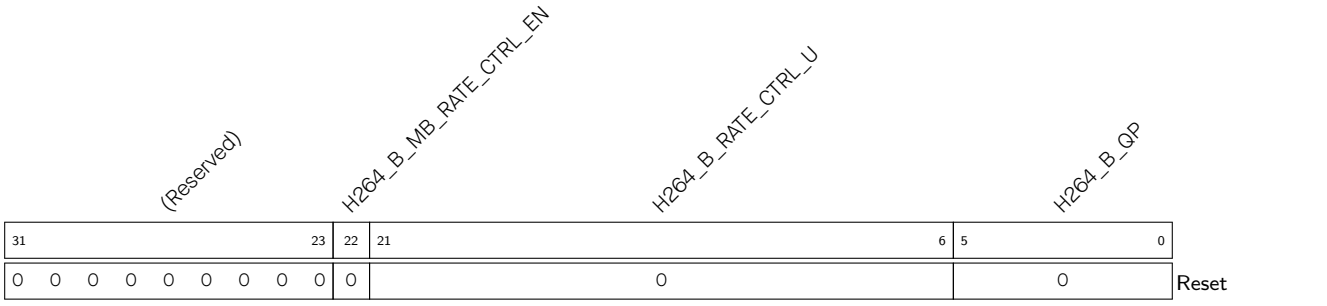
H264_B_I16×16_DECI_SCORE_OFFSET 配置视频序列 B i16 × 16 MB 降采样得分偏移。此偏移将加到 i16 × 16 MB 得分中。(R/W)

H264_B_I_CHROMA_DECI_SCORE_OFFSET 配置视频序列 B I 色度 MB 降采样得分偏移。此偏移将加到 I 色度 MB 得分中。(R/W)

H264_B_P16×16_DECI_SCORE_OFFSET 配置视频序列 B p16 × 16 MB 降采样得分偏移。此偏移将加到 p16 × 16 MB 得分中。(R/W)

H264_B_P_CHROMA_DECI_SCORE_OFFSET 配置视频序列 B p 色度 MB 降采样得分偏移。此偏移将加到 p 色度 MB 得分中。(R/W)

Register 41.26. H264_B_RC_CONFO_REG (0x0064)



H264_B_QP 配置视频序列 B 帧级初始亮度 QP 值。(R/W)

H264_B_RATE_CTRL_U 配置视频序列 B 参数 U 值。U = int((float) u « 8)。(R/W)

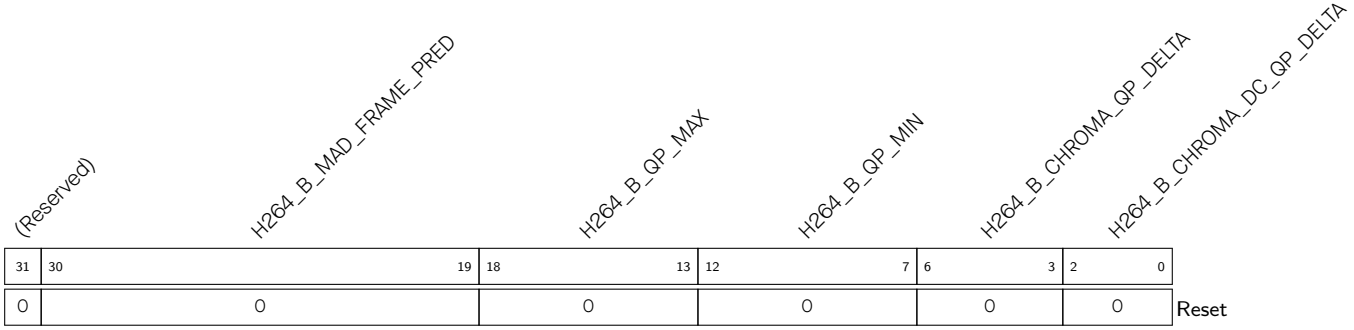
H264_B_MB_RATE_CTRL_EN 配置是否启用视频序列 B 的 MB 级速率控制。

0：禁用 MB 级速率控制

1：启用 MB 级速率控制

(R/W)

Register 41.27. H264_B_RC_CONF1_REG (0x0068)



H264_B_CHROMA_DC_QP_DELTA 配置基于色度 QP 的视频序列 B 色度 DC QP 偏移。色度 DC QP = 色度 QP（映射后）+ reg_chroma_dc_QP_delta。(R/W)

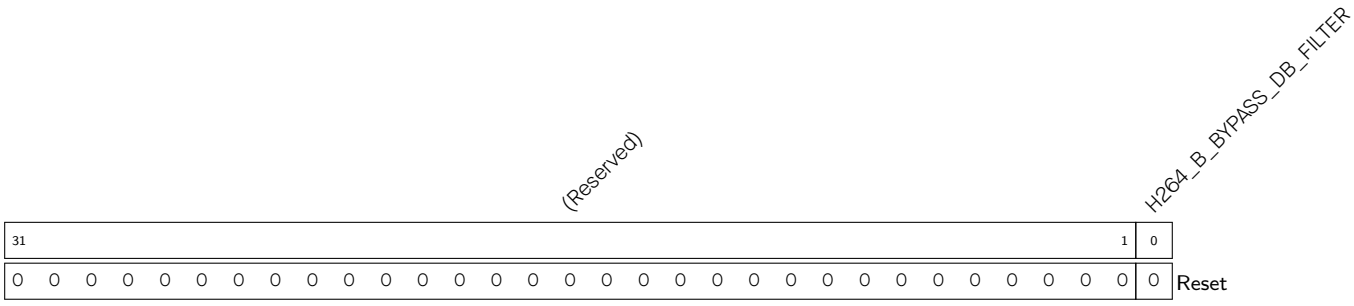
H264_B_CHROMA_QP_DELTA 配置基于亮度 QP 的视频序列 B 色度 QP 偏移。色度 QP（映射前）= 亮度 QP + reg_chroma_QP_delta。(R/W)

H264_B_QP_MIN 配置视频序列 B 允许的最小亮度 QP 值。(R/W)

H264_B_QP_MAX 配置视频序列 B 允许的最大亮度 QP 值。(R/W)

H264_B_MAD_FRAME_PRED 配置视频序列 B 帧级预测 MB 的 MAD 值。(R/W)

Register 41.28. H264_B_DB_BYPASS_REG (0x006C)



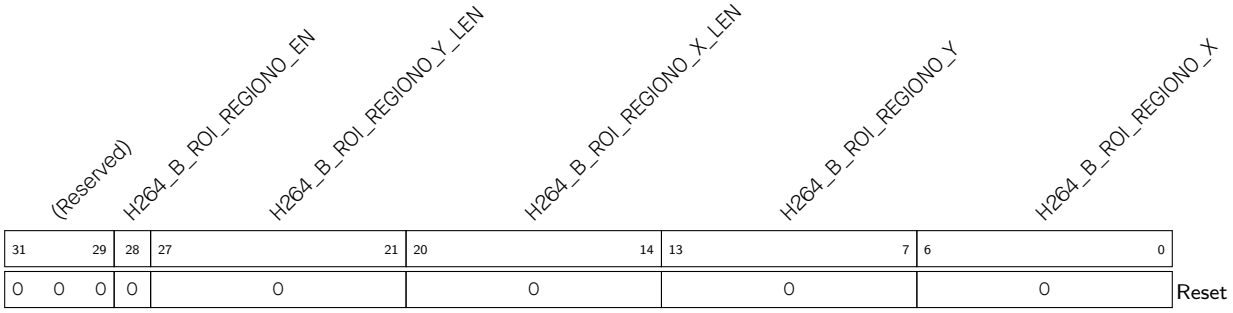
H264_B_BYPASS_DB_FILTER 配置是否绕过视频序列 B 的去块滤波器。

0：不绕过去块滤波器

1：绕过去块滤波器

(R/W)

Register 41.29. H264_B_ROI_REGION0_REG (0x0070)



H264_B_ROI_REGION0_X 配置视频序列 B 中 ROI 0 的水平起始 MB。(R/W)

H264_B_ROI_REGION0_Y 配置视频序列 B 中 ROI 0 的垂直起始 MB。(R/W)

H264_B_ROI_REGION0_X_LEN 配置视频序列 B 中 ROI 0 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION0_Y_LEN 配置视频序列 B 中 ROI 0 的垂直方向 MB 数量。(R/W)

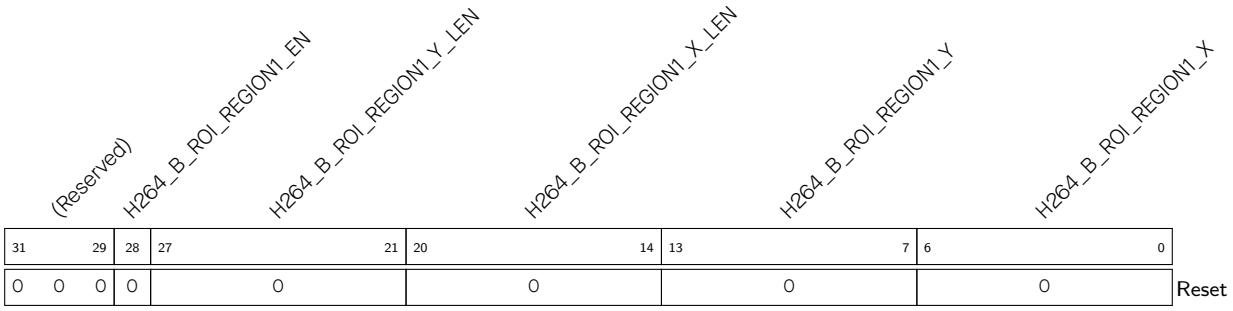
H264_B_ROI_REGION0_EN 配置是否启用视频序列 B 的 ROI 0。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.30. H264_B_ROI_REGION1_REG (0x0074)



H264_B_ROI_REGION1_X 配置视频序列 B 中 ROI 1 的水平起始 MB。(R/W)

H264_B_ROI_REGION1_Y 配置视频序列 B 中 ROI 1 的垂直起始 MB。(R/W)

H264_B_ROI_REGION1_X_LEN 配置视频序列 B 中 ROI 1 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION1_Y_LEN 配置视频序列 B 中 ROI 1 的垂直方向 MB 数量。(R/W)

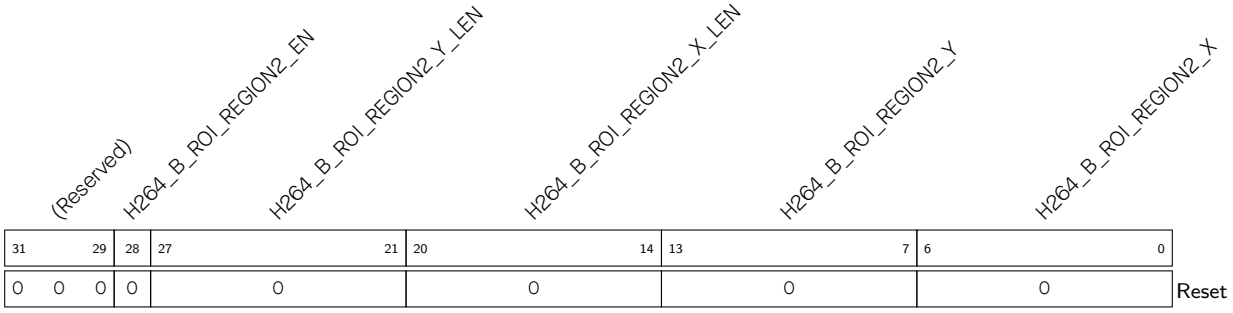
H264_B_ROI_REGION1_EN 配置是否启用视频序列 B 的 ROI 1。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.31. H264_B_ROI_REGION2_REG (0x0078)



H264_B_ROI_REGION2_X 配置视频序列 B 中 ROI 2 的水平起始 MB。(R/W)

H264_B_ROI_REGION2_Y 配置视频序列 B 中 ROI 2 的垂直起始 MB。(R/W)

H264_B_ROI_REGION2_X_LEN 配置视频序列 B 中 ROI 2 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION2_Y_LEN 配置视频序列 B 中 ROI 2 的垂直方向 MB 数量。(R/W)

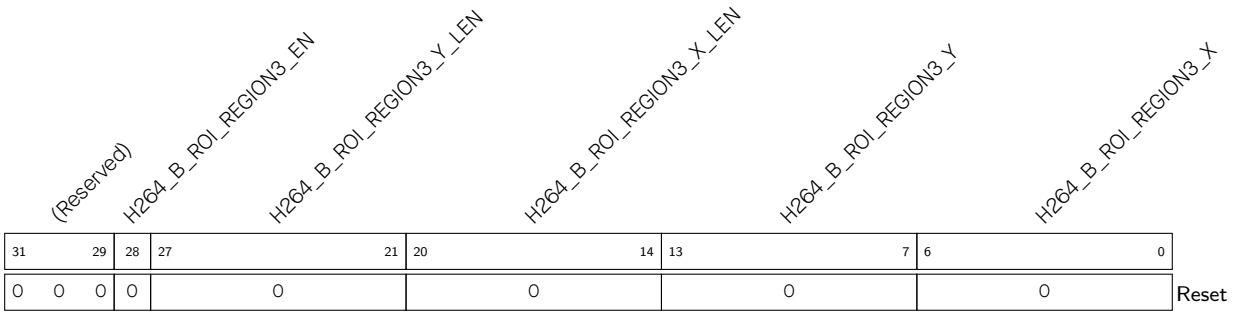
H264_B_ROI_REGION2_EN 配置是否启用视频序列 B 的 ROI 2。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.32. H264_B_ROI_REGION3_REG (0x007C)



H264_B_ROI_REGION3_X 配置视频序列 B 中 ROI 3 的水平起始 MB。(R/W)

H264_B_ROI_REGION3_Y 配置视频序列 B 中 ROI 3 的垂直起始 MB。(R/W)

H264_B_ROI_REGION3_X_LEN 配置视频序列 B 中 ROI 3 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION3_Y_LEN 配置视频序列 B 中 ROI 3 的垂直方向 MB 数量。(R/W)

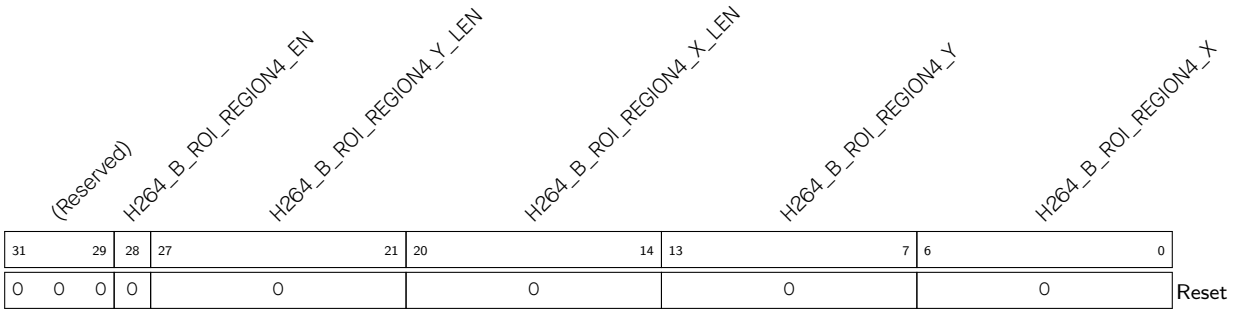
H264_B_ROI_REGION3_EN 配置是否启用视频序列 B 的 ROI 3。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.33. H264_B_ROI_REGION4_REG (0x0080)



H264_B_ROI_REGION4_X 配置视频序列 B 中 ROI 4 的水平起始 MB。(R/W)

H264_B_ROI_REGION4_Y 配置视频序列 B 中 ROI 4 的垂直起始 MB。(R/W)

H264_B_ROI_REGION4_X_LEN 配置视频序列 B 中 ROI 4 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION4_Y_LEN 配置视频序列 B 中 ROI 4 的垂直方向 MB 数量。(R/W)

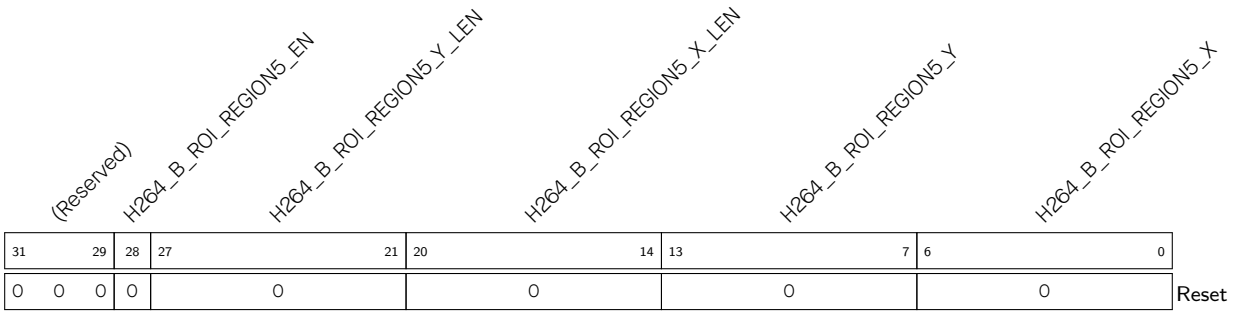
H264_B_ROI_REGION4_EN 配置是否启用视频序列 B 的 ROI 4。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.34. H264_B_ROI_REGION5_REG (0x0084)



H264_B_ROI_REGION5_X 配置视频序列 B 中 ROI 5 的水平起始 MB。(R/W)

H264_B_ROI_REGION5_Y 配置视频序列 B 中 ROI 5 的垂直起始 MB。(R/W)

H264_B_ROI_REGION5_X_LEN 配置视频序列 B 中 ROI 5 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION5_Y_LEN 配置视频序列 B 中 ROI 5 的垂直方向 MB 数量。(R/W)

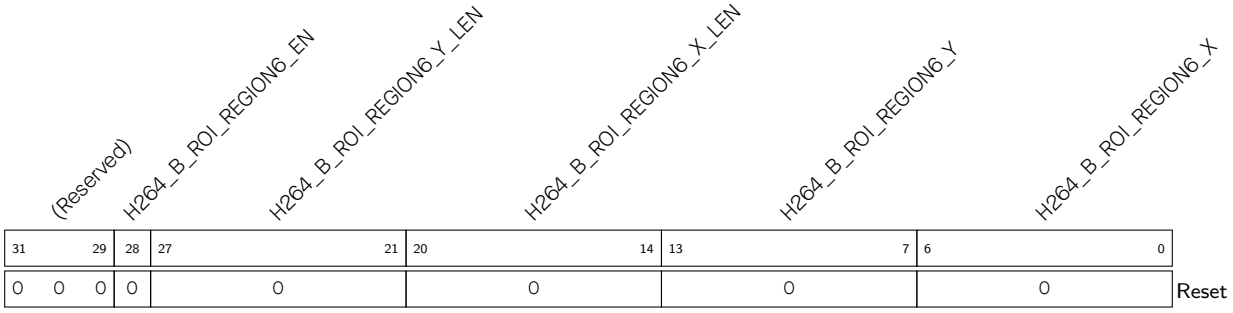
H264_B_ROI_REGION5_EN 配置是否启用视频序列 B 的 ROI 5。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.35. H264_B_ROI_REGION6_REG (0x0088)



H264_B_ROI_REGION6_X 配置视频序列 B 中 ROI 6 的水平起始 MB。(R/W)

H264_B_ROI_REGION6_Y 配置视频序列 B 中 ROI 6 的垂直起始 MB。(R/W)

H264_B_ROI_REGION6_X_LEN 配置视频序列 B 中 ROI 6 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION6_Y_LEN 配置视频序列 B 中 ROI 6 的垂直方向 MB 数量。(R/W)

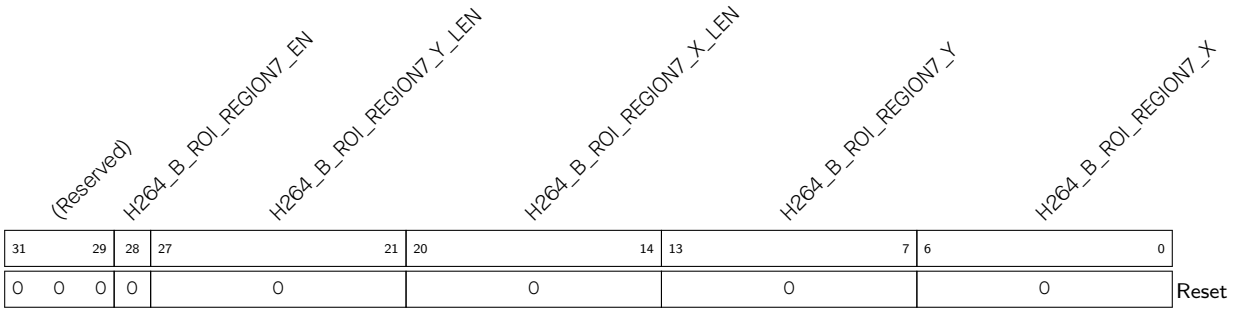
H264_B_ROI_REGION6_EN 配置是否启用视频序列 B 的 ROI 6。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.36. H264_B_ROI_REGION7_REG (0x008C)



H264_B_ROI_REGION7_X 配置视频序列 B 中 ROI 7 的水平起始 MB。(R/W)

H264_B_ROI_REGION7_Y 配置视频序列 B 中 ROI 7 的垂直起始 MB。(R/W)

H264_B_ROI_REGION7_X_LEN 配置视频序列 B 中 ROI 7 的水平方向 MB 数量。(R/W)

H264_B_ROI_REGION7_Y_LEN 配置视频序列 B 中 ROI 7 的垂直方向 MB 数量。(R/W)

H264_B_ROI_REGION7_EN 配置是否启用视频序列 B 的 ROI 7。

0：禁用 ROI

1：启用 ROI

(R/W)

Register 41.37. H264_B_ROI_REGION0_3_QP_REG (0x0090)



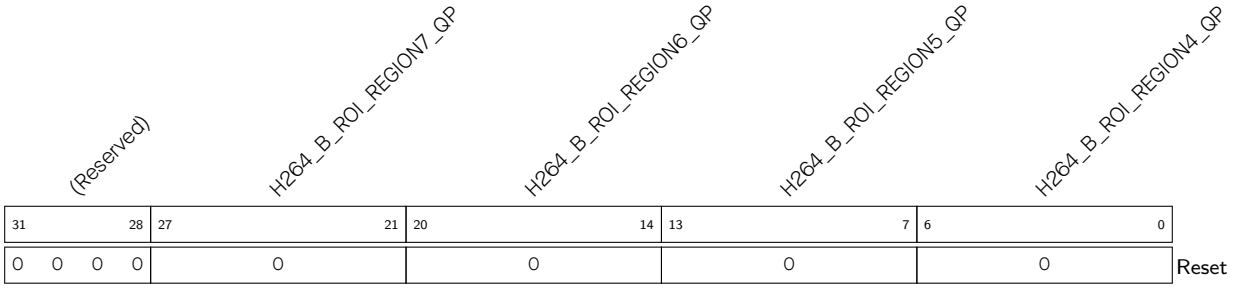
H264_B_ROI_REGION0_QP 配置视频序列 B 中 H264 ROI 0 的 QP 调整量。(R/W)

H264_B_ROI_REGION1_QP 配置视频序列 B 中 H264 ROI 1 的 QP 调整量。(R/W)

H264_B_ROI_REGION2_QP 配置视频序列 B 中 H264 ROI 2 的 QP 调整量。(R/W)

H264_B_ROI_REGION3_QP 配置视频序列 B 中 H264 ROI 3 的 QP 调整量。(R/W)

Register 41.38. H264_B_ROI_REGION4_7_QP_REG (0x0094)



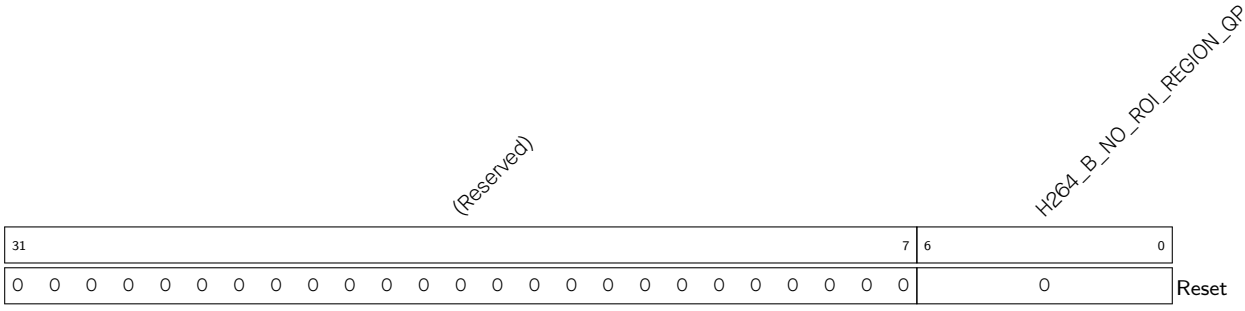
H264_B_ROI_REGION4_QP 配置视频序列 B 中 H264 ROI 4 的 QP 调整量。(R/W)

H264_B_ROI_REGION5_QP 配置视频序列 B 中 H264 ROI 5 的 QP 调整量。(R/W)

H264_B_ROI_REGION6_QP 配置视频序列 B 中 H264 ROI 6 的 QP 调整量。(R/W)

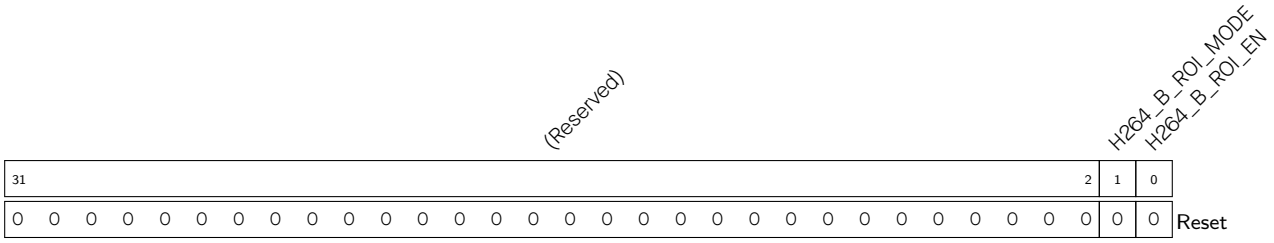
H264_B_ROI_REGION7_QP 配置视频序列 B 中 H264 ROI 7 的 QP 调整量。(R/W)

Register 41.39. H264_B_NO_ROI_REGION_QP_OFFSET_REG (0x0098)



H264_B_NO_ROI_REGION_QP 配置视频序列 B 中 H264 非 ROI 区域的 QP，delta QP。(R/W)

Register 41.40. H264_B_ROI_CONFIG_REG (0x009C)



H264_B_ROI_EN 配置是否在视频序列 B 中启用 ROI。

0：禁用 ROI

1：启用 ROI

(R/W)

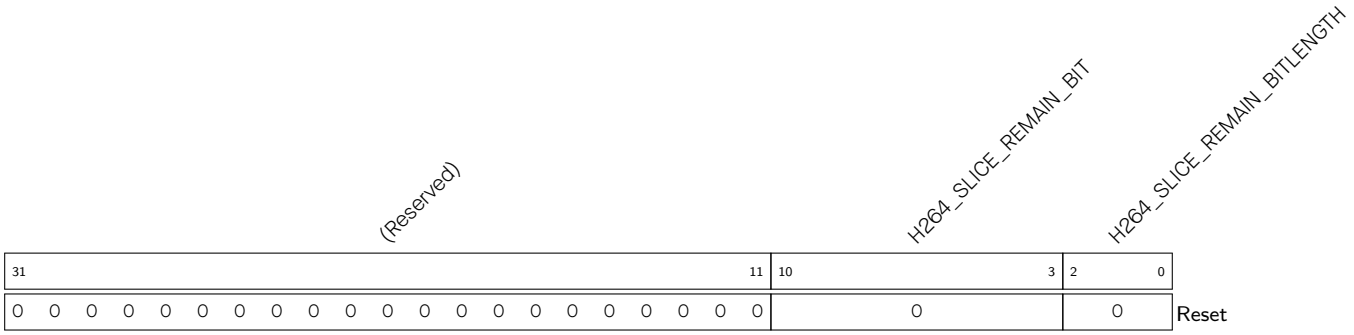
H264_B_ROI_MODE 配置视频序列 B 中 ROI 的模式。

0：固定 QP

1：Delta QP

(R/W)

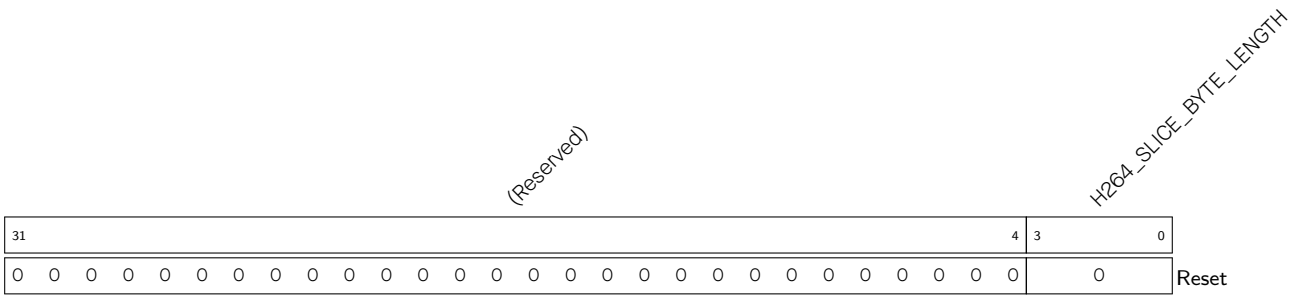
Register 41.41. H264_SLICE_HEADER_REMAIN_REG (0x00AC)



H264_SLICE_REMAIN_BITLENGTH 配置切片头剩余位数。(R/W)

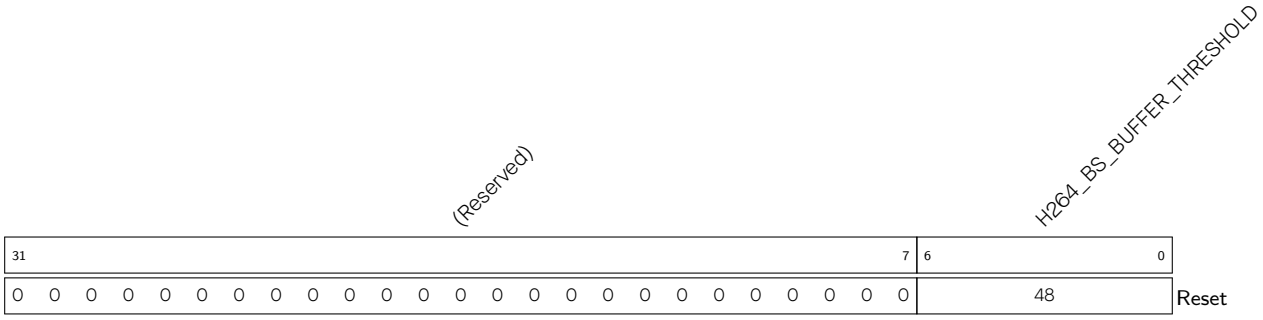
H264_SLICE_REMAIN_BIT 配置切片头剩余位。(R/W)

Register 41.42. H264_SLICE_HEADER_BYTE_LENGTH_REG (0x00B0)



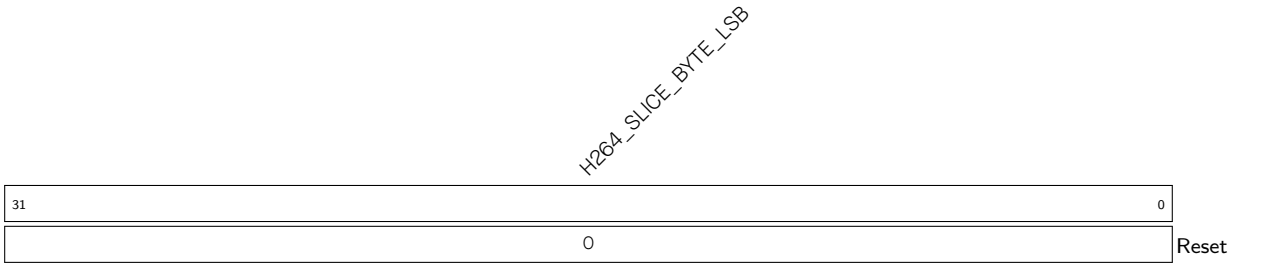
H264_SLICE_BYTE_LENGTH 配置切片头字节数。(R/W)

Register 41.43. H264_BS_THRESHOLD_REG (0x00B4)



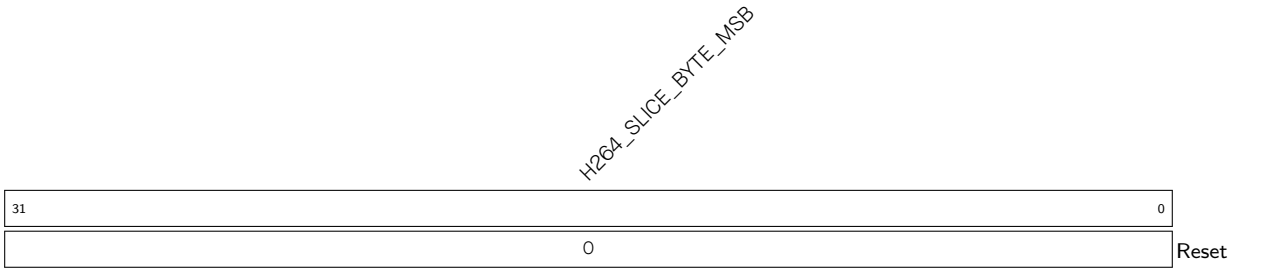
H264_BS_BUFFER_THRESHOLD 配置比特流缓冲区溢出阈值。此值应大于一个 4 × 4 子宏块的编码字节数。(R/W)

Register 41.44. H264_SLICE_HEADER_BYTE0_REG (0x00B8)



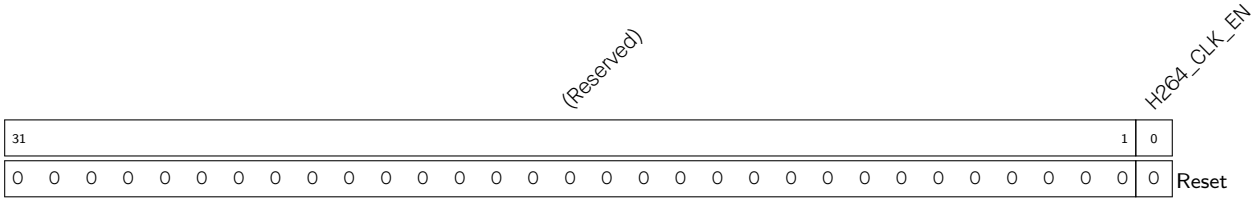
H264_SLICE_BYTE_LSB 配置切片头低 32 位。(R/W)

Register 41.45. H264_SLICE_HEADER_BYTE1_REG (0x00BC)



H264_SLICE_BYTE_MSB 配置切片头高 32 位。(R/W)

Register 41.46. H264_CONF_REG (0x00D0)



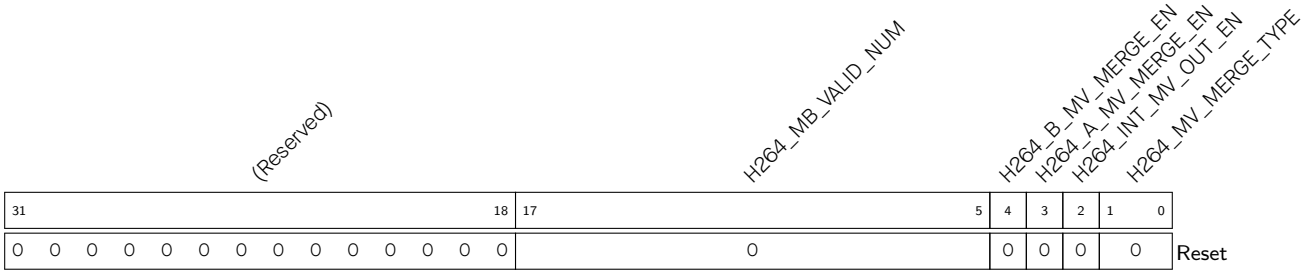
H264_CLK_EN 配置是否启用寄存器时钟门。

0：仅在应用程序写入寄存器时启用时钟门

1：强制启用寄存器的时钟门

(R/W)

Register 41.47. H264_MV_MERGE_CONFIG_REG (0x00D4)



H264_MV_MERGE_TYPE 配置 MV 合并类型。

0：合并 p16 × 16 MV

1：合并最小 MV

2：合并最大 MV

3：无效值。没有作用

(R/W)

H264_INT_MV_OUT_EN 配置合并后 MV 的输出类型。

0：输出所有部分（整数和小数）均非零的合并后 MV

1：只输出整数部分非零的合并后 MV

(R/W)

H264_A_MV_MERGE_EN 配置是否启用视频序列 A 的 MV 合并。

0：禁用

1：启用

(R/W)

H264_B_MV_MERGE_EN 配置是否启用视频序列 B 的 MV 合并。

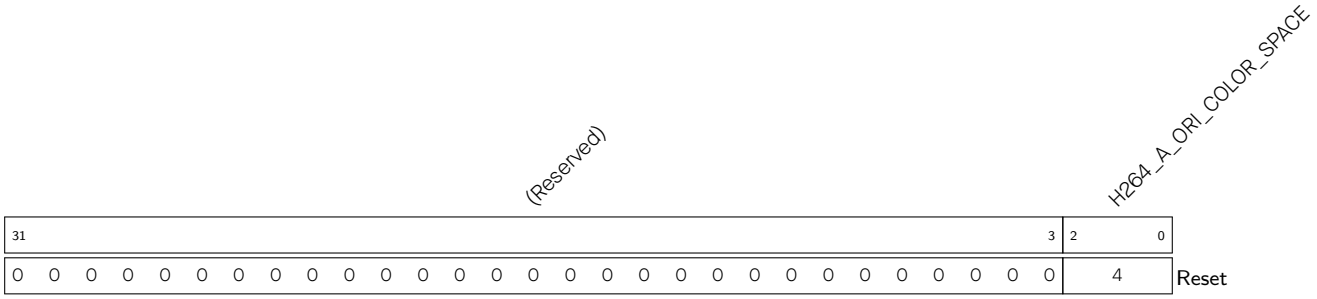
0：禁用

1：启用

(R/W)

H264_MB_VALID_NUM 表示输出有效的合并后 MV 的 MB 数量。(RO)

Register 41.48. H264_A_ORI_CONF_REG (0x00F4)



H264_A_ORI_COLOR_SPACE 配置视频序列 A 的原始图像色彩空间。

0：RGB888

1：RGB565

2：YUV444

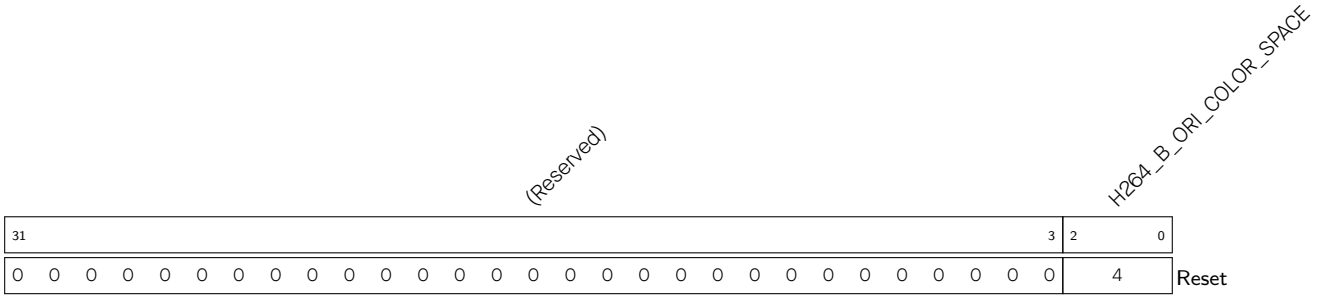
3：YUV422

4：YUV420

其他：无效值。没有作用

(R/W)

Register 41.49. H264_B_ORI_CONF_REG (0x00F8)



H264_B_ORI_COLOR_SPACE 配置视频序列 B 的原始图像色彩空间。

0：RGB888

1：RGB565

2：YUV444

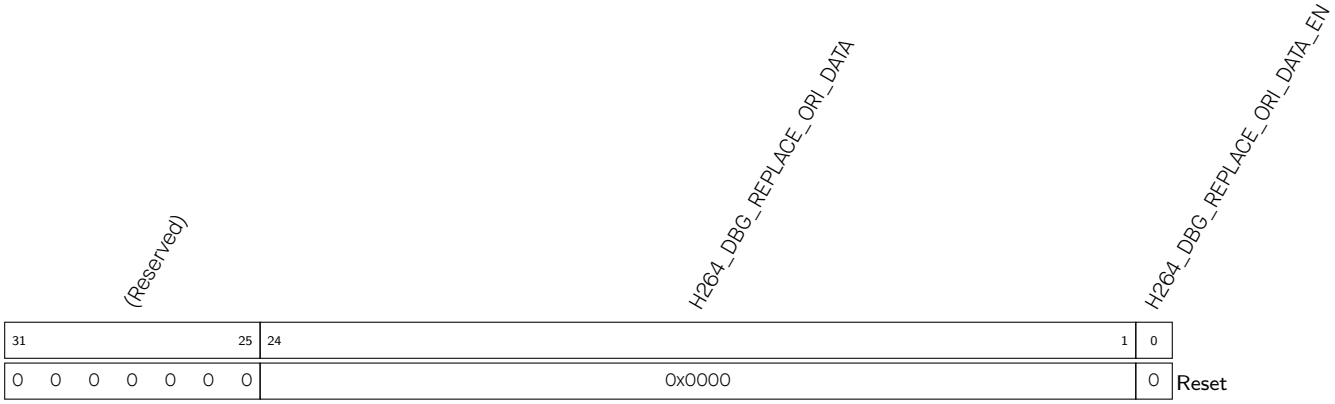
3：YUV422

4：YUV420

其他：无效值。没有作用

(R/W)

Register 41.50. H264_ORI_DEBUG_CONF_REG (0x00FC)



H264_DBG_REPLACE_ORI_DATA_EN 配置是否替换原始图像像素。

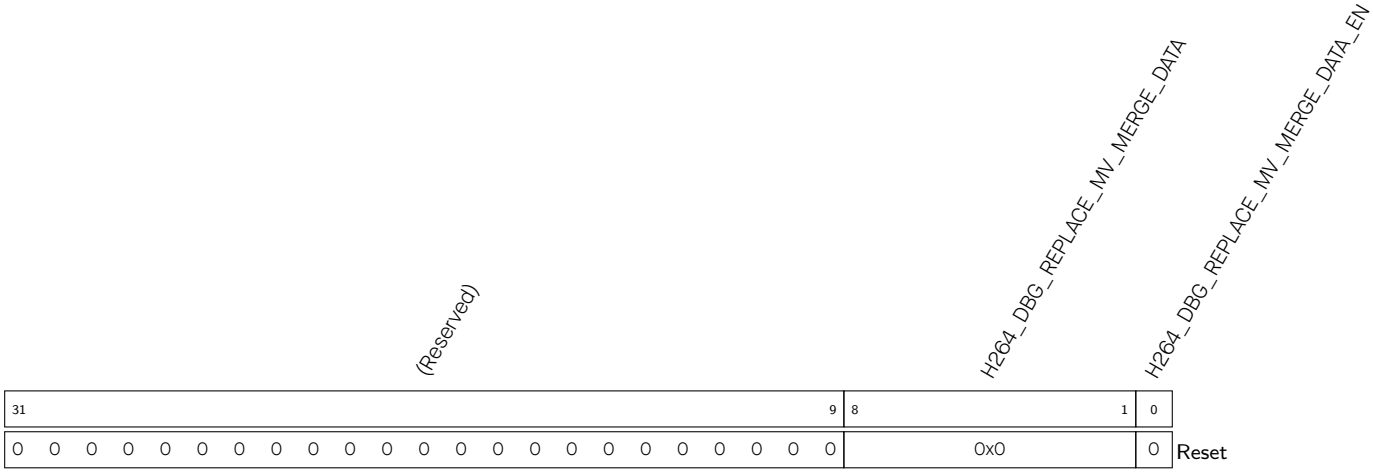
0：不替换

1：替换

(R/W)

H264_DBG_REPLACE_ORI_DATA 配置要替换的原始图像像素。当原始图像色彩空间为 RGB 时，byte0 2 为 BGR。当色彩空间为 YUV 时，byte0 2 为 VUY。(R/W)

Register 41.51. H264_MV_MERGE_DEBUG_CONF_REG (0x0100)



H264_DBG_REPLACE_MV_MERGE_DATA_EN 配置是否替换 MV 合并数据。

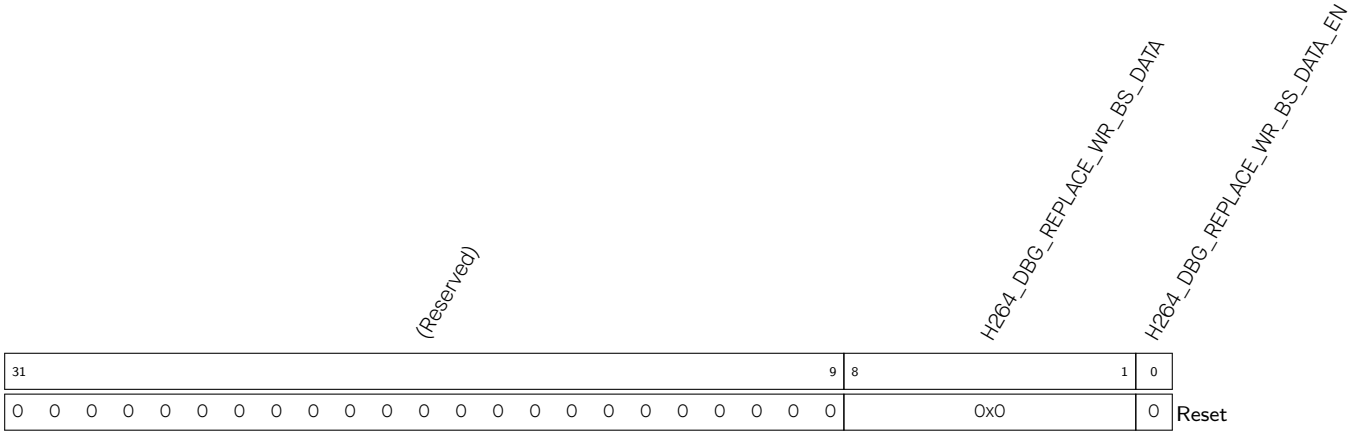
0：不替换

1：替换

(R/W)

H264_DBG_REPLACE_MV_MERGE_DATA 配置要替换的 MV 合并数据。(R/W)

Register 41.52. H264_BS_DEBUG_CONG_REG (0x0104)



H264_DBG_REPLACE_WR_BS_DATA_EN 配置是否替换比特流数据。

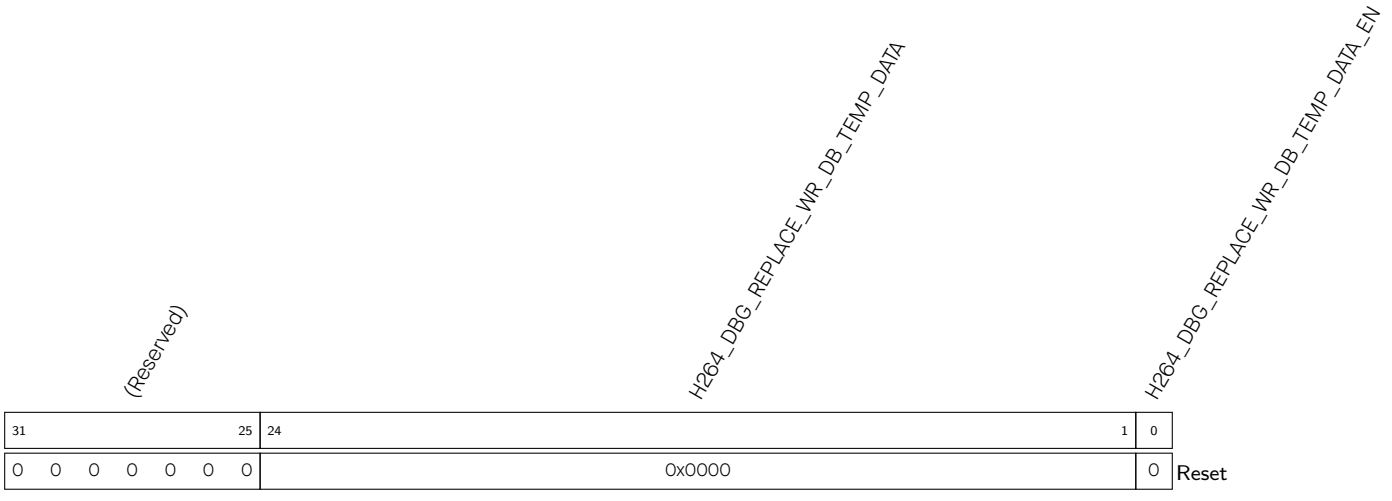
0：不替换

1：替换

(R/W)

H264_DBG_REPLACE_WR_BS_DATA 配置要替换的比特流数据。(R/W)

Register 41.53. H264_DB_WR_TEMP_DEBUG_CONG_REG (0x0108)



H264_DBG_REPLACE_WR_DB_TEMP_DATA_EN 配置去块滤波器是否替换写入的临时数据。

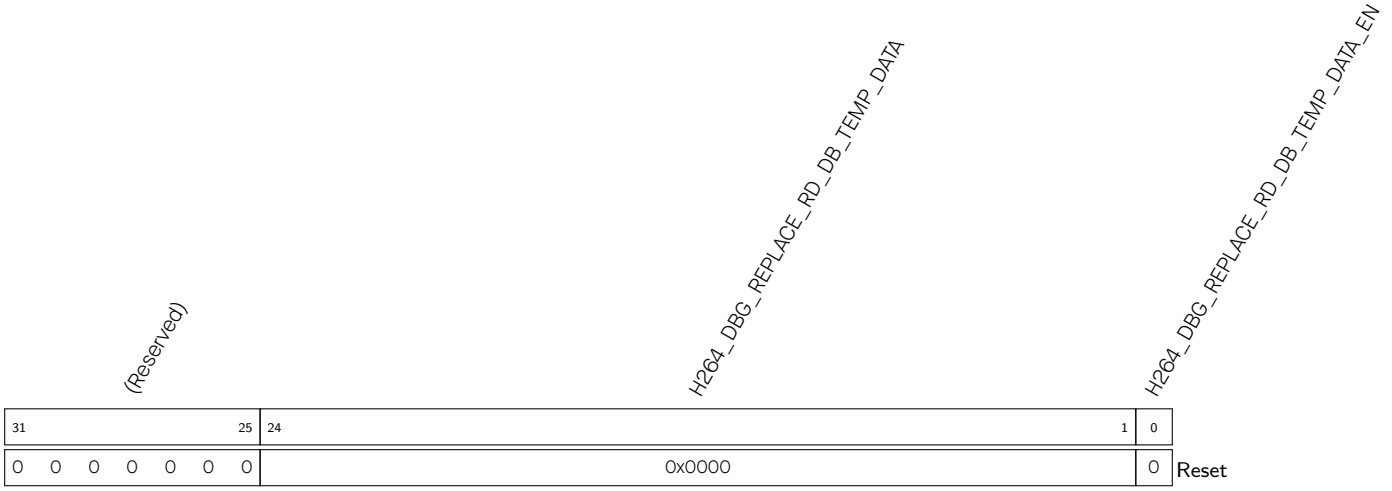
0：不替换

1：替换

(R/W)

H264_DBG_REPLACE_WR_DB_TEMP_DATA 配置去块滤波器要替换的写入临时数据。byte0 2 为 VUY。(R/W)

Register 41.54. H264_DB_RD_TEMP_DEBUG_CONG_REG (0x010C)



H264_DBG_REPLACE_RD_DB_TEMP_DATA_EN 配置去块滤波器是否替换读取的临时数据。

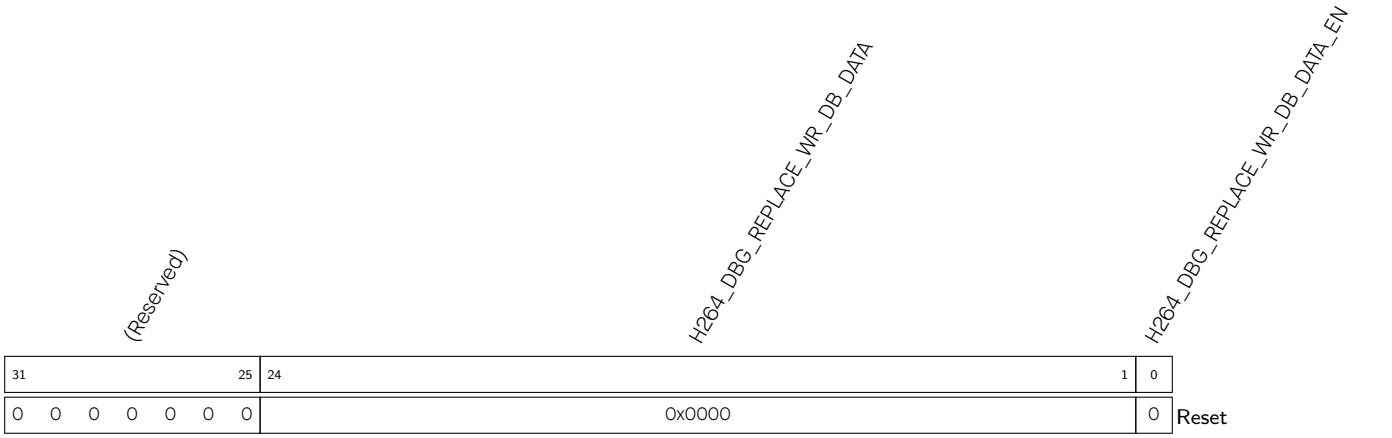
0：不替换

1：替换

(R/W)

H264_DBG_REPLACE_RD_DB_TEMP_DATA 配置去块滤波器要替换的读取临时数据。byte0 2 为 VUY。(R/W)

Register 41.55. H264_DB_WR_DEBUG_CONG_REG (0x0110)



H264_DBG_REPLACE_WR_DB_DATA_EN 配置去块滤波器是否替换写入数据。

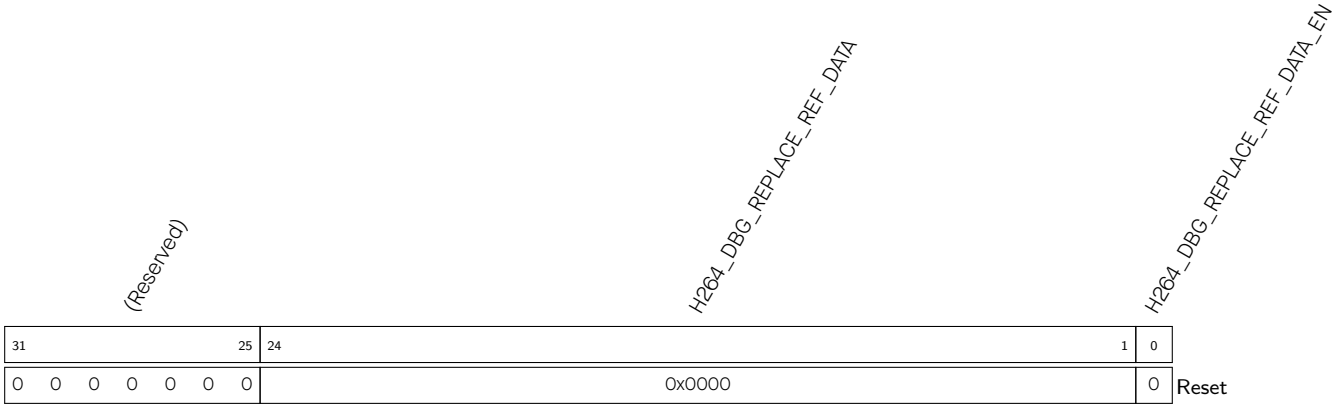
0：不替换

1：替换

(R/W)

H264_DBG_REPLACE_WR_DB_DATA 配置去块滤波器要替换的写入数据。byte0 2 为 VUY。(R/W)

Register 41.56. H264_REF_DEBUG_CONG_REG (0x0114)



H264_DBG_REPLACE_REF_DATA_EN 配置是否替换参考图像像素。

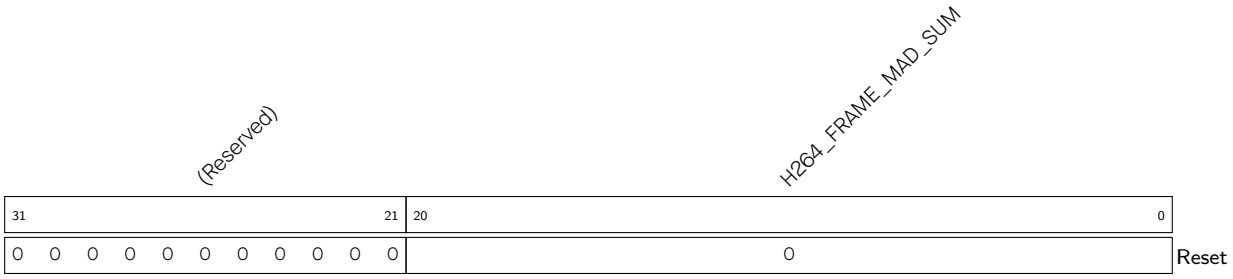
0：不替换

1：替换

(R/W)

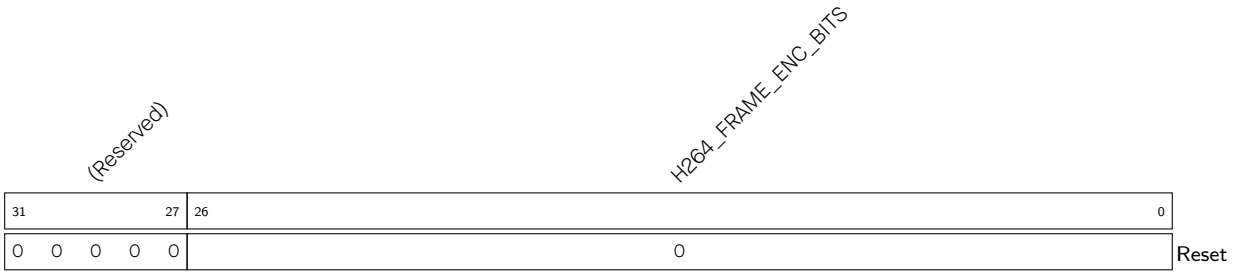
H264_DBG_REPLACE_REF_DATA 配置要替换的参考图像像素。byte0 2 为 VUY。(R/W)

Register 41.57. H264_RC_STATUS0_REG (0x00A0)



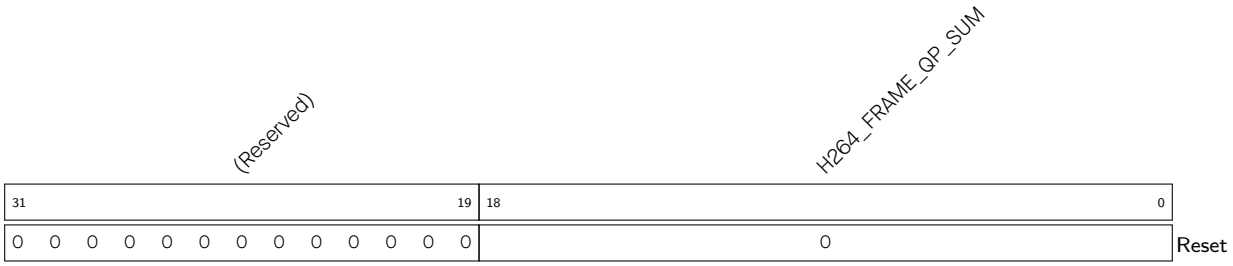
H264_FRAME_MAD_SUM 表示一帧中所有 MB 实际 MAD 的总和。(RO)

Register 41.58. H264_RC_STATUS1_REG (0x00A4)



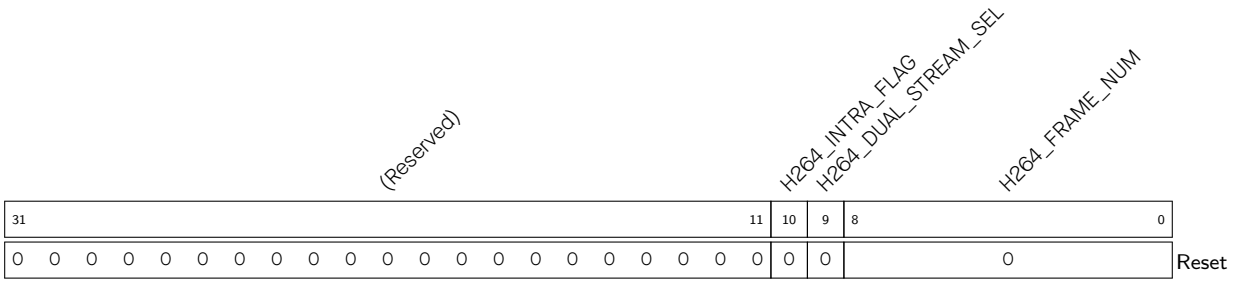
H264_FRAME_ENC_BITS 表示一帧中所有 MB 实际编码比特数的总和。(RO)

Register 41.59. H264_RC_STATUS2_REG (0x00A8)



H264_FRAME_QP_SUM 表示一帧中所有 MB 实际亮度 QP 总和值。(RO)

Register 41.60. H264_SYS_STATUS_REG (0x00DC)



H264_FRAME_NUM 表示当前帧号。(RO)

H264_DUAL_STREAM_SEL 表示当前帧使用的寄存器组。

0：使用寄存器组 A

1：使用寄存器组 B

(RO)

H264_INTRA_FLAG 表示当前编码帧的类型。

0：P 帧

1：I 帧

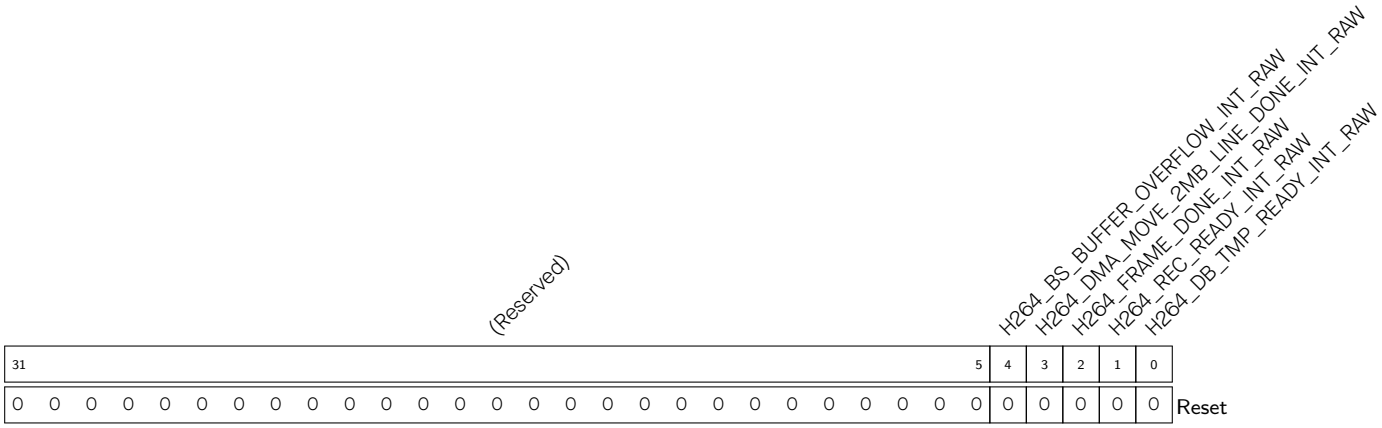
(RO)

Register 41.61. H264_FRAME_CODE_LENGTH_REG (0x00E0)



H264_FRAME_CODE_LENGTH 表示当前帧代码的字节长度。(RO)

Register 41.62. H264_INT_RAW_REG (0x00C0)



H264_DB_TMP_READY_INT_RAW H264_DB_TMP_READY_INT 的原始中断状态。(R/WTC/SS)

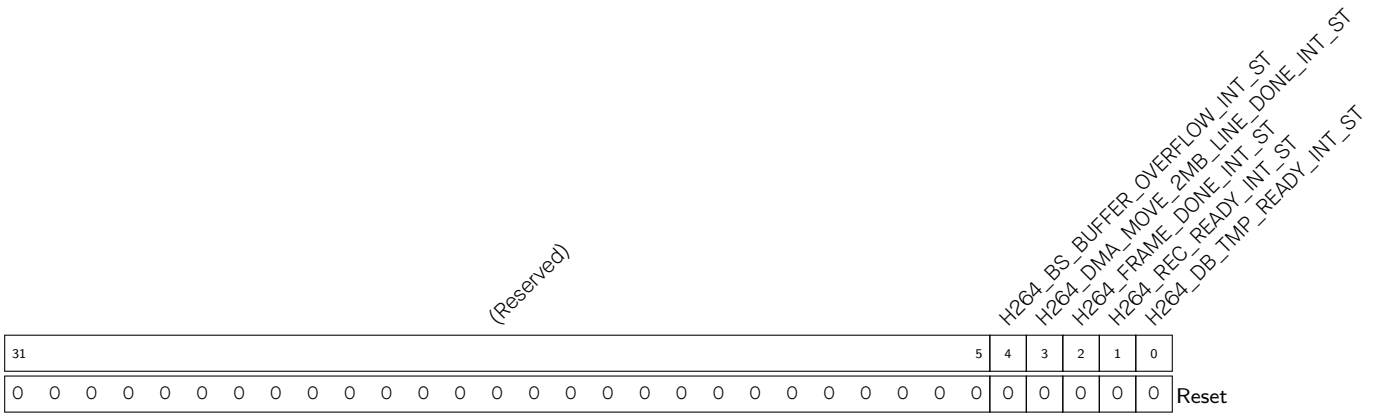
H264_REC_READY_INT_RAW H264_REC_READY_INT 的原始中断状态。(R/WTC/SS)

H264_FRAME_DONE_INT_RAW H264_FRAME_DONE_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_MOVE_2MB_LINE_DONE_INT_RAW H264_DMA_MOVE_2MB_LINE_DONE_INT 的原始中断状态。(R/WTC/SS)

H264_BS_BUFFER_OVERFLOW_INT_RAW H264_BS_BUFFER_OVERFLOW_INT 的原始中断状态。(R/WTC/SS)

Register 41.63. H264_INT_ST_REG (0x00C4)



H264_DB_TMP_READY_INT_ST H264_DB_TMP_READY_INT 的屏蔽中断状态。(RO)

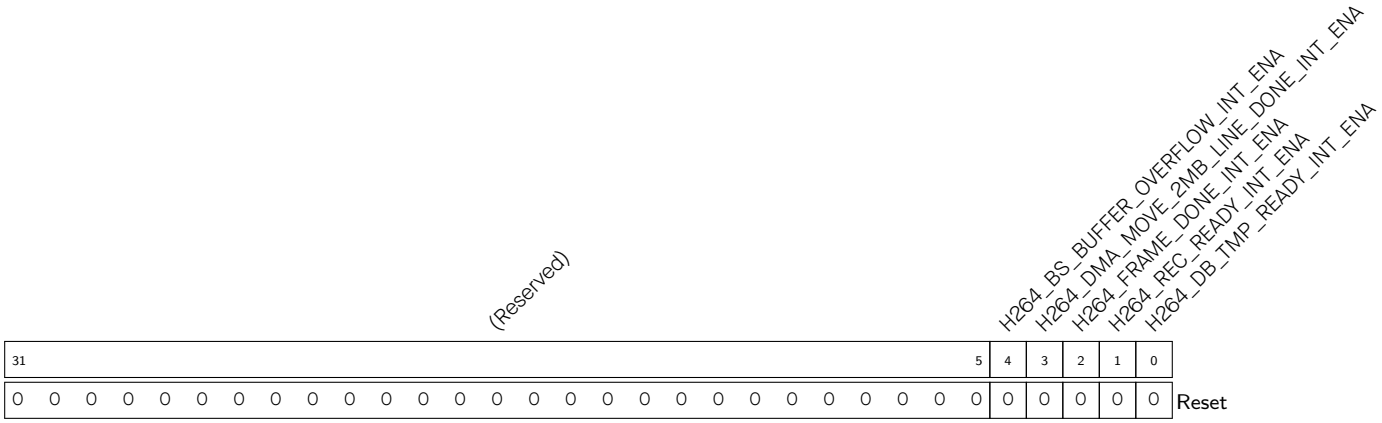
H264_REC_READY_INT_ST H264_REC_READY_INT 的屏蔽中断状态。(RO)

H264_FRAME_DONE_INT_ST H264_FRAME_DONE_INT 的屏蔽中断状态。(RO)

H264_DMA_MOVE_2MB_LINE_DONE_INT_ST H264_DMA_MOVE_2MB_LINE_DONE_INT 的屏蔽中断状态。(RO)

H264_BS_BUFFER_OVERFLOW_INT_ST H264_BS_BUFFER_OVERFLOW_INT 的屏蔽中断状态。(RO)

Register 41.64. H264_INT_ENA_REG (0x00C8)



H264_DB_TMP_READY_INT_ENA 写 1 以启用 H264_DB_TMP_READY_INT。(R/W)

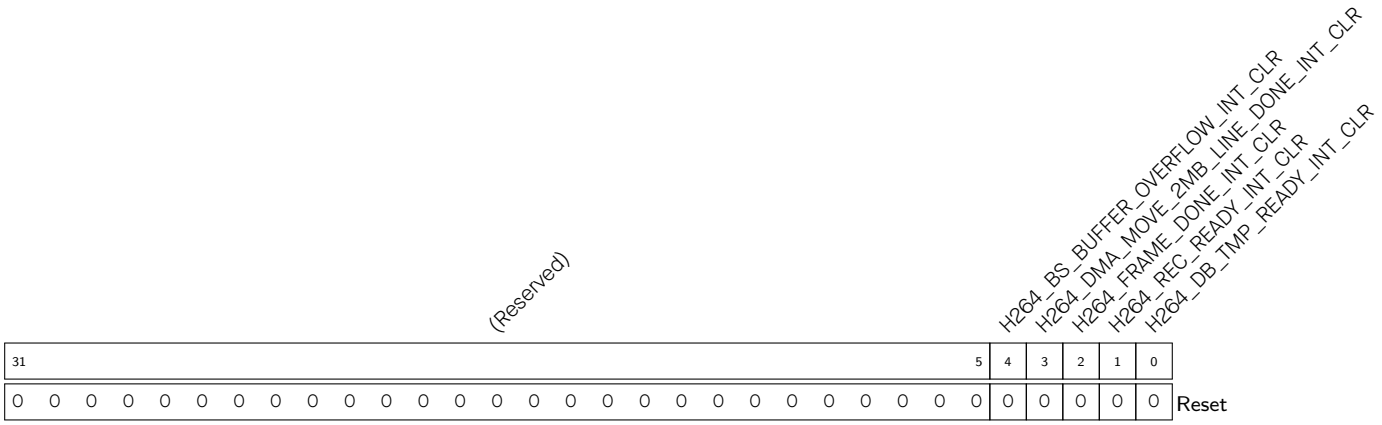
H264_REC_READY_INT_ENA 写 1 以启用 H264_REC_READY_INT。(R/W)

H264_FRAME_DONE_INT_ENA 写 1 以启用 H264_FRAME_DONE_INT。(R/W)

H264_DMA_MOVE_2MB_LINE_DONE_INT_ENA 写 1 以启用 H264_DMA_MOVE_2MB_LINE_-DONE_INT。(R/W)

H264_BS_BUFFER_OVERFLOW_INT_ENA 写 1 以启用 H264_BS_BUFFER_OVERFLOW_INT。(R/W)

Register 41.65. H264_INT_CLR_REG (0x00CC)



H264_DB_TMP_READY_INT_CLR 写 1 清除 H264_DB_TMP_READY_INT。(WT)

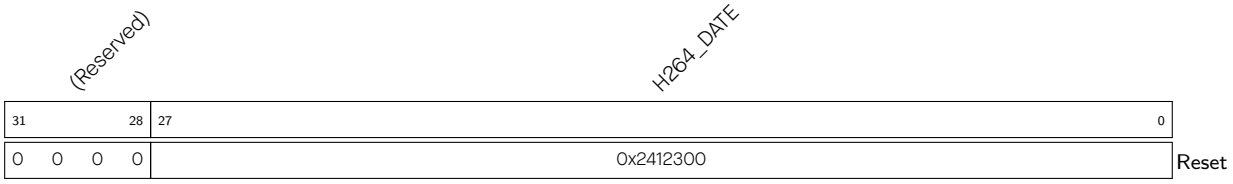
H264_REC_READY_INT_CLR 写 1 清除 H264_REC_READY_INT。(WT)

H264_FRAME_DONE_INT_CLR 写 1 清除 H264_FRAME_DONE_INT。(WT)

H264_DMA_MOVE_2MB_LINE_DONE_INT_CLR 写 1 清除 H264_DMA_MOVE_2MB_LINE_-DONE_INT。(WT)

H264_BS_BUFFER_OVERFLOW_INT_CLR 写 1 清除 H264_BS_BUFFER_OVERFLOW_INT。(WT)

Register 41.66. H264_DATE_REG (0x00F0)



H264_DATE 版本控制寄存器。(R/W)

41.9.2 H264 DMA 寄存器

Register 41.67. H264_DMA_OUT_CONF0_CH0_REG (0x0000)

31–27	26	25	24	23–17	16	15–13	12	11–9	8–6	5	4	3	2	1	0	
(Reserved)	H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH0	H264_DMA_OUT_CMD_DISABLE_CH0	H264_DMA_OUT_RST_CH0	(Reserved)	H264_DMA_OUT_REORDER_EN_CH0	(Reserved)	H264_DMA_OUT_PAGE_BOUND_EN_CH0	(Reserved)	H264_DMA_OUT_MEM_BURST_LENGTH_CH0	(Reserved)	H264_DMA_OUT_CHECK_OWNER_CH0	H264_DMA_OUT_ECC_AES_EN_CH0	H264_DMA_OUTDSCR_BURST_EN_CH0	H264_DMA_OUT_EOF_MODE_CH0	H264_DMA_OUT_AUTO_WRBACK_CH0	
0 0 0 0 0	0	0	0	0 0 0 0 0 0 0	0	0 0 0	0	0 0 0	0x0	0	0	0	0	1	0	Reset

H264_DMA_OUT_AUTO_WRBACK_CH0 配置是否在接收到由 outlink 描述符指向的所有数据时启用自动 outlink 写回。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_EOF_MODE_CH0 为 TX 通道 0 配置 EOF 标志生成模式。

0：当最后的数据已被推入 DMA 的 FIFO 时生成 EOF 标志

1：当最后的数据已从 DMA 的 FIFO 中弹出时生成 EOF 标志

(R/W)

H264_DMA_OUTDSCR_BURST_EN_CH0 配置是否启用 TX 通道 0 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ECC_AES_EN_CH0 配置 DMA 是否可以通过 TX 通道 0 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问 ECC/AES 区域。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_OUT_CHECK_OWNER_CH0 配置是否启用 TX 通道 0 的 owner 位检查。

0：禁用

1：启用

(R/W)

见下页...

Register 41.67. H264_DMA_OUT_CONF0_CH0_REG (0x0000)

接上页...

H264_DMA_OUT_MEM_BURST_LENGTH_CH0 配置 TX 通道 0 的突发长度。

0：8 字节
1：16 字节
2：32 字节
3：64 字节
4：128 字节
5 ~ 7：无效
(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH0 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界
1：AXI 读取的数据不会跨越地址边界
(R/W)

H264_DMA_OUT_REORDER_EN_CH0 配置是否启用 TX 通道 0 对宏块进行重排序。

0：禁用
1：启用
(R/W)

H264_DMA_OUT_RST_CH0 写入 1 然后写入 0 以重置 TX 通道 0。(R/W)

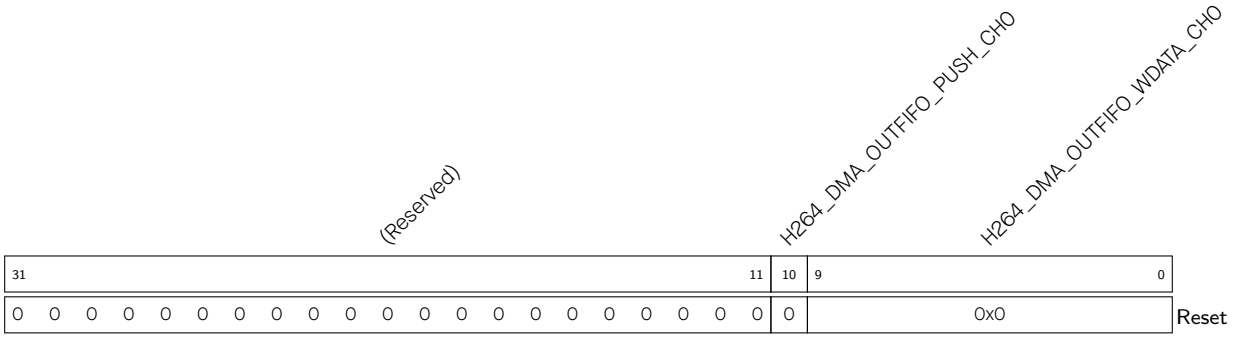
H264_DMA_OUT_CMD_DISABLE_CH0 配置是否禁用 TX 通道 0 上的命令。

0：禁用
1：启用
(R/W)

H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH0 配置是否禁用 TX 通道 0 的权重仲裁。

0：启用
1：禁用
(R/W)

Register 41.68. H264_DMA_OUT_PUSH_CH0_REG (0x0018)



H264_DMA_OUTFIFO_WDATA_CH0 表示需要推入 DMA TX FIFO 的数据。(R/W)

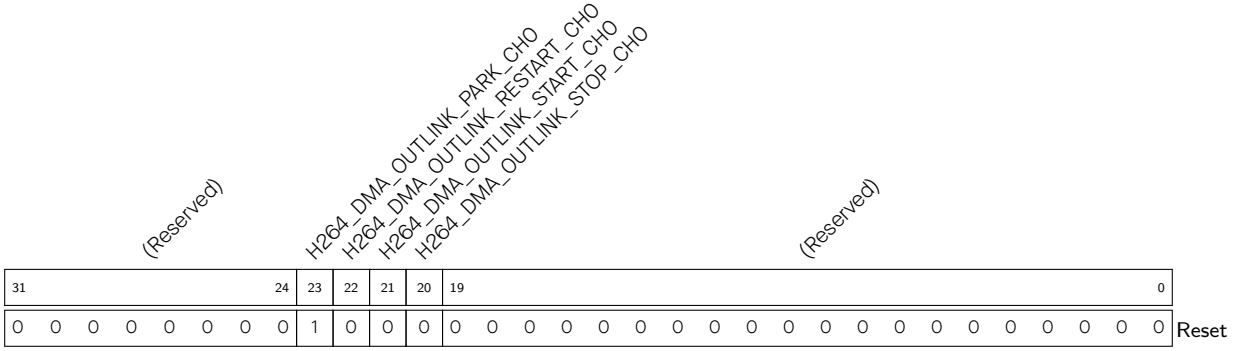
H264_DMA_OUTFIFO_PUSH_CH0 配置是否将数据推入 DMA TX FIFO。

0：无效

1：推送

(R/W/SC)

Register 41.69. H264_DMA_OUT_LINK_CONF_CH0_REG (0x001C)



H264_DMA_OUTLINK_STOP_CH0 配置是否停止处理 outlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_OUTLINK_START_CH0 配置是否开始处理 outlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH0 配置是否从最后一个地址重新开始一个新的 outlink。

0：无效

1：重新开始

(R/W/SC)

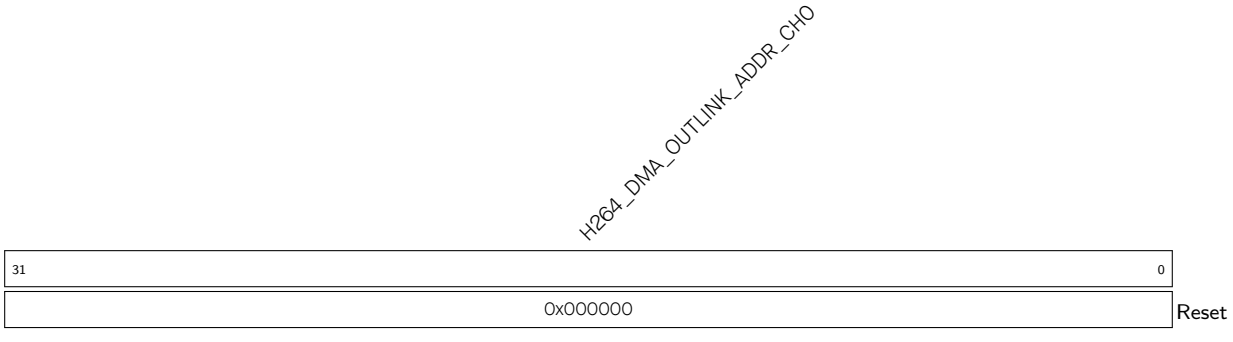
H264_DMA_OUTLINK_PARK_CH0 表示传输描述符的 FSM 的状态。

0：outlink 描述符的 FSM 正在工作

1：outlink 描述符的 FSM 处于空闲状态

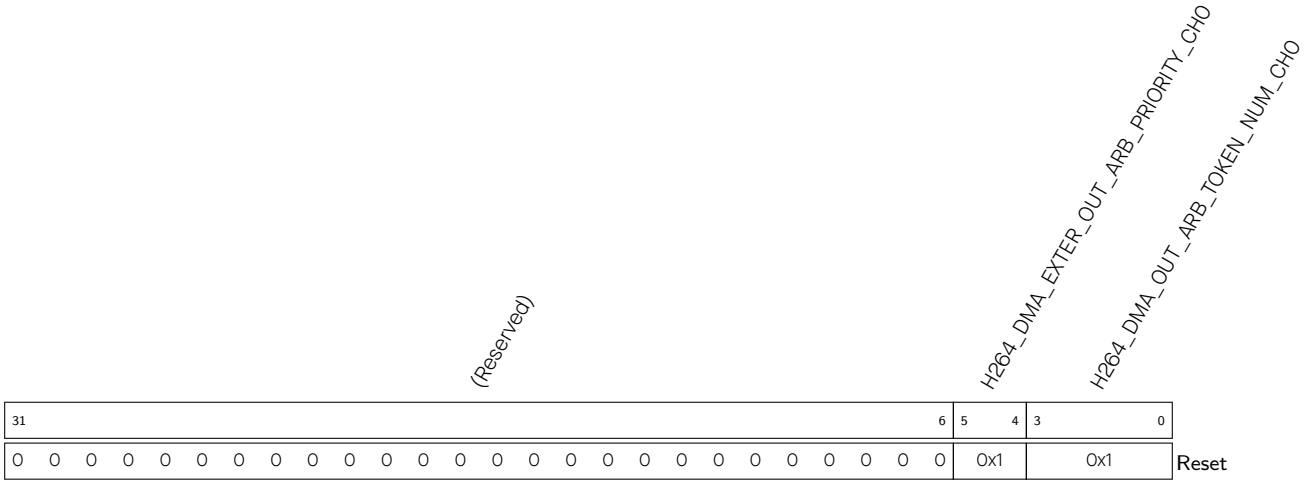
(RO)

Register 41.70. H264_DMA_OUT_LINK_ADDR_CH0_REG (0x0020)



H264_DMA_OUTLINK_ADDR_CH0 配置第一个 outlink 描述符的地址。(R/W)

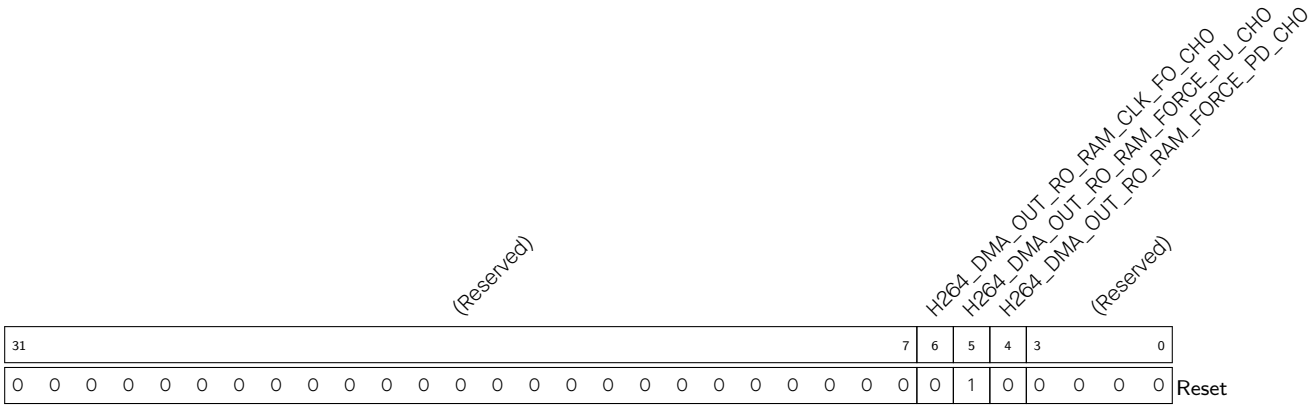
Register 41.71. H264_DMA_OUT_ARB_CH0_REG (0x003C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH0 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_EXTER_OUT_ARB_PRIORITY_CH0 配置 TX 通道 0 的外部存储器访问优先级。(R/W)

Register 41.72. H264_DMA_OUT_RO_PD_CONF_CH0_REG (0x0044)



H264_DMA_OUT_RO_RAM_FORCE_PD_CH0 配置是否强制为 DMA 重新排序 RAM 断电。

0：不强制断电

1：强制断电

(R/W)

H264_DMA_OUT_RO_RAM_FORCE_PU_CH0 配置是否强制为 DMA 重新排序 RAM 上电。

0：不强制上电

1：强制上电

(R/W)

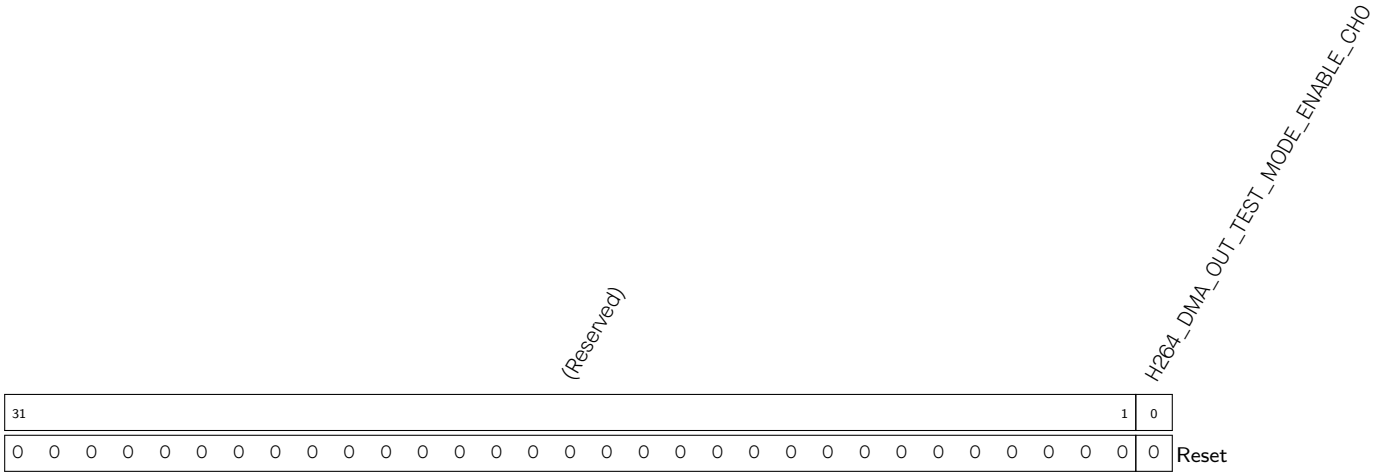
H264_DMA_OUT_RO_RAM_CLK_FO_CH0 配置是否强制启用时钟门控对 DMA TX 重新排序 RAM。

0：访问 DMA 中的 RAM 时将使用时钟门控

1：访问 DMA 中的 RAM 时强制打开时钟并绕过时钟门控

(R/W)

Register 41.73. H264_DMA_OUT_MODE_ENABLE_CH0_REG (0x0050)



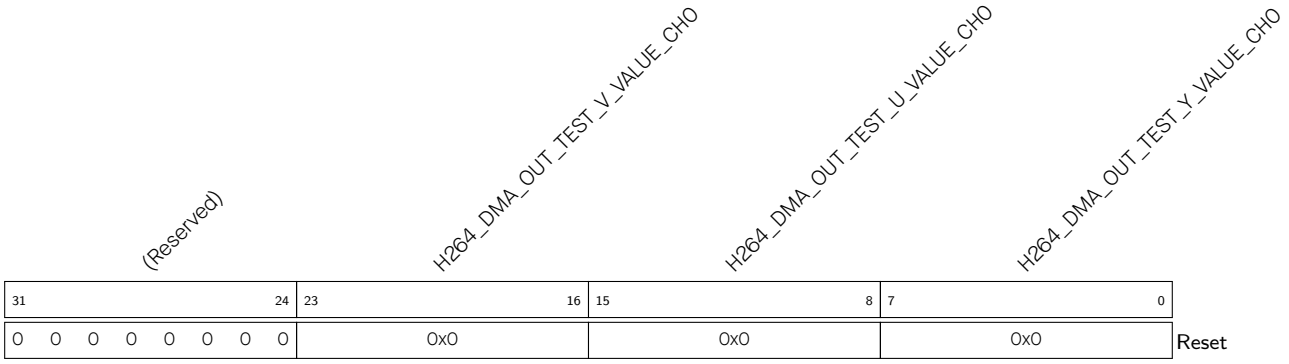
H264_DMA_OUT_TEST_MODE_ENABLE_CH0 配置是否启用 TX 通道 0 的测试模式。

0：TX 通道 0 在正常模式下工作

1：TX 通道 0 在测试模式下工作

(R/W)

Register 41.74. H264_DMA_OUT_MODE_YUV_CH0_REG (0x0054)

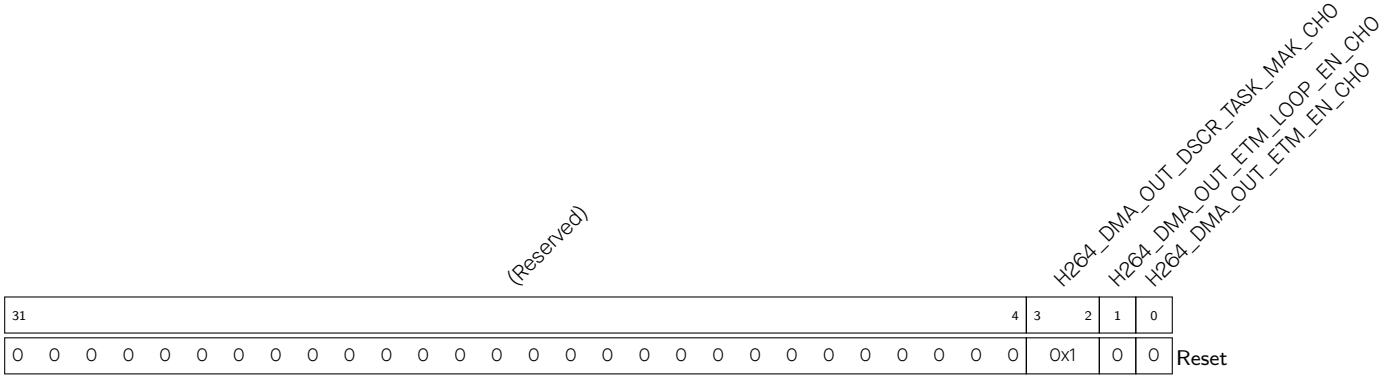


H264_DMA_OUT_TEST_Y_VALUE_CH0 在测试模式下配置 y 值。(R/W)

H264_DMA_OUT_TEST_U_VALUE_CH0 在测试模式下配置 u 值。(R/W)

H264_DMA_OUT_TEST_V_VALUE_CH0 在测试模式下配置 v 值。(R/W)

Register 41.75. H264_DMA_OUT_ETM_CONF_CH0_REG (0x0068)



H264_DMA_OUT_ETM_EN_CH0 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH0 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH0 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.76. H264_DMA_OUT_CONF0_CH1_REG (0x0100)

Bits	Field	Reset
31–27	(Reserved)	0 0 0 0 0
26	H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH1	0
25	H264_DMA_OUT_CMD_DISABLE_CH1	0
24	H264_DMA_OUT_RST_CH1	0
23–13	(Reserved)	0 0 0 0 0 0 0 0 0 0 0
12	H264_DMA_OUT_PAGE_BOUND_EN_CH1	0
11–9	(Reserved)	0 0 0
8–6	H264_DMA_OUT_MEM_BURST_LENGTH_CH1	0x0
5	(Reserved)	0
4	H264_DMA_OUT_CHECK_OWNER_CH1	0
3	H264_DMA_OUT_ECC_AES_EN_CH1	0
2	H264_DMA_OUTDSCR_BURST_EN_CH1	0
1	H264_DMA_OUT_EOF_MODE_CH1	1
0	H264_DMA_OUT_AUTO_WRBACK_CH1	0

H264_DMA_OUT_AUTO_WRBACK_CH1 配置是否在接收到由 outlink 描述符指向的所有数据时启用自动 outlink 写回。
- 0：禁用
- 1：启用

(R/W)

H264_DMA_OUT_EOF_MODE_CH1 为 TX 通道 1 配置 EOF 标志生成模式。
- 0：当最后的数据已被推入 DMA 的 FIFO 时生成 EOF 标志
- 1：当最后的数据已从 DMA 的 FIFO 中弹出时生成 EOF 标志

(R/W)

H264_DMA_OUTDSCR_BURST_EN_CH1 配置是否启用 TX 通道 1 读取描述符的 INCR 突发传输。
- 0：禁用
- 1：启用

(R/W)

H264_DMA_OUT_ECC_AES_EN_CH1 配置 DMA 是否可以通过 TX 通道 1 访问外部存储空间以进行 ECC 和 AES。
- 0：不访问
- 1：访问 ECC/AES 区域。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_OUT_CHECK_OWNER_CH1 配置是否启用 TX 通道 1 的 owner 位检查。
- 0：禁用
- 1：启用

(R/W)

见下页...

Register 41.76. H264_DMA_OUT_CONF0_CH1_REG (0x0100)

接上页...

H264_DMA_OUT_MEM_BURST_LENGTH_CH1 配置 TX 通道 1 的突发长度。

0：8 字节
1：16 字节
2：32 字节
3：64 字节
4：128 字节
5 ~ 7：无效
(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH1 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界
1：AXI 读取的数据不会跨越地址边界
(R/W)

H264_DMA_OUT_RST_CH1 写入 1 然后写入 0 以重置 TX 通道 1。(R/W)

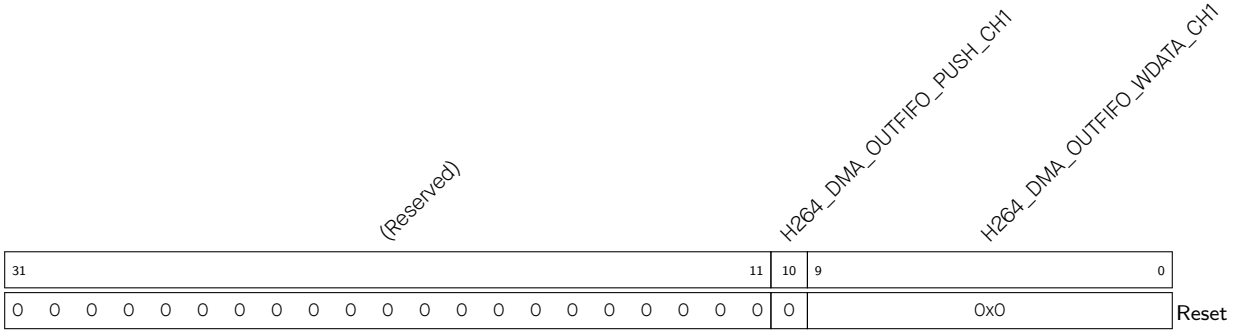
H264_DMA_OUT_CMD_DISABLE_CH1 配置是否禁用 TX 通道 1 上的命令。

0：禁用
1：启用
(R/W)

H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH1 配置是否禁用 TX 通道 1 的权重仲裁。

0：启用
1：禁用
(R/W)

Register 41.77. H264_DMA_OUT_PUSH_CH1_REG (0x0118)



H264_DMA_OUTFIFO_WDATA_CH1 表示需要推入 DMA TX FIFO 的数据。(R/W)

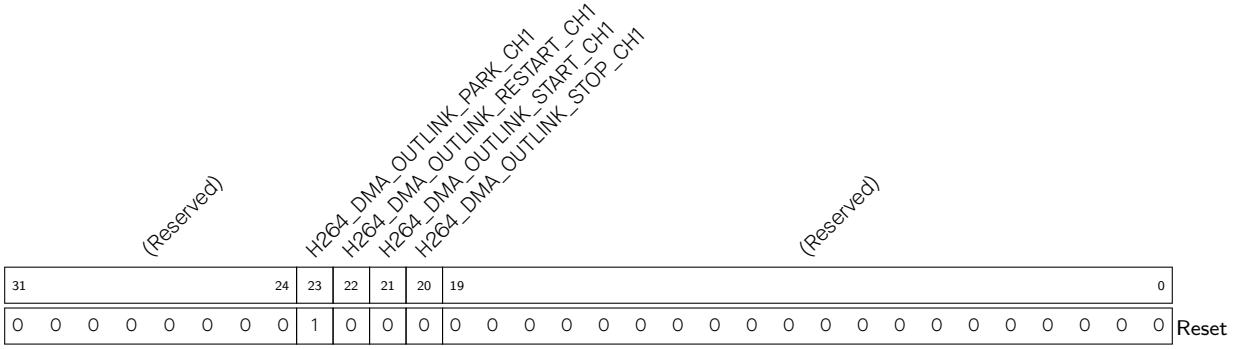
H264_DMA_OUTFIFO_PUSH_CH1 配置是否将数据推入 DMA TX FIFO。

0：无效

1：推送

(R/W/SC)

Register 41.78. H264_DMA_OUT_LINK_CONF_CH1_REG (0x011C)



H264_DMA_OUTLINK_STOP_CH1 配置是否停止处理 outlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_OUTLINK_START_CH1 配置是否开始处理 outlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH1 配置是否从最后一个地址重新开始一个新的 outlink。

0：无效

1：重新开始

(R/W/SC)

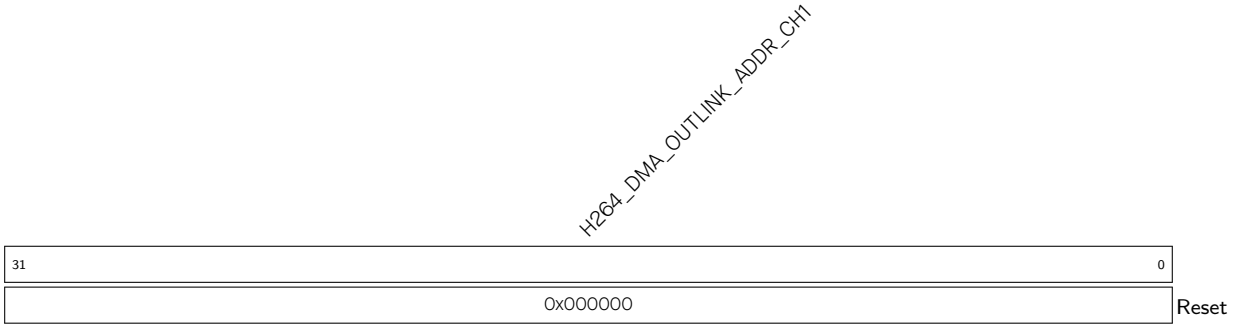
H264_DMA_OUTLINK_PARK_CH1 表示传输描述符的 FSM 的状态。

0：outlink 描述符的 FSM 正在工作

1：outlink 描述符的 FSM 处于空闲状态

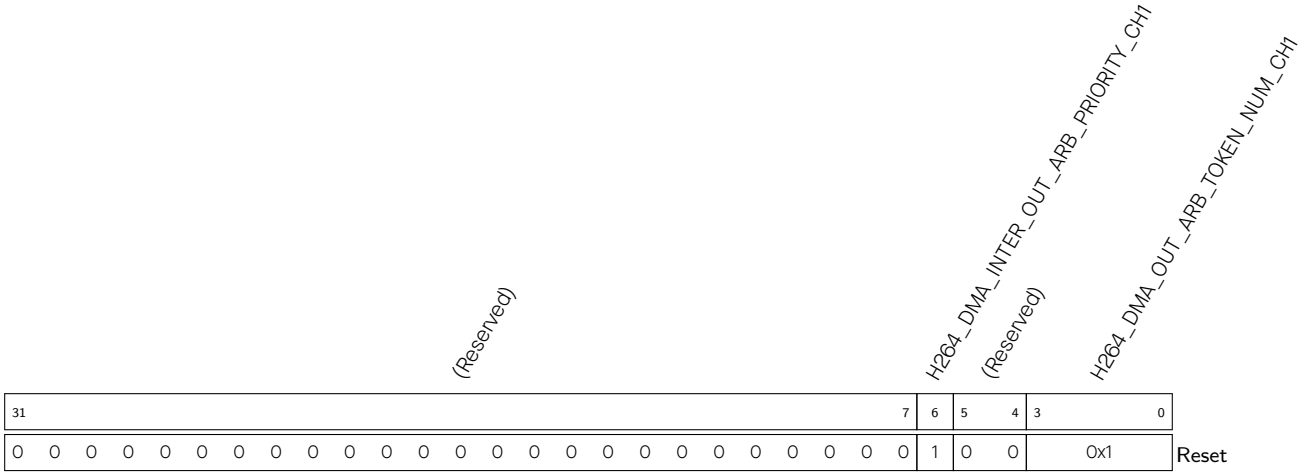
(RO)

Register 41.79. H264_DMA_OUT_LINK_ADDR_CH1_REG (0x0120)



H264_DMA_OUTLINK_ADDR_CH1 配置第一个 outlink 描述符的地址。(R/W)

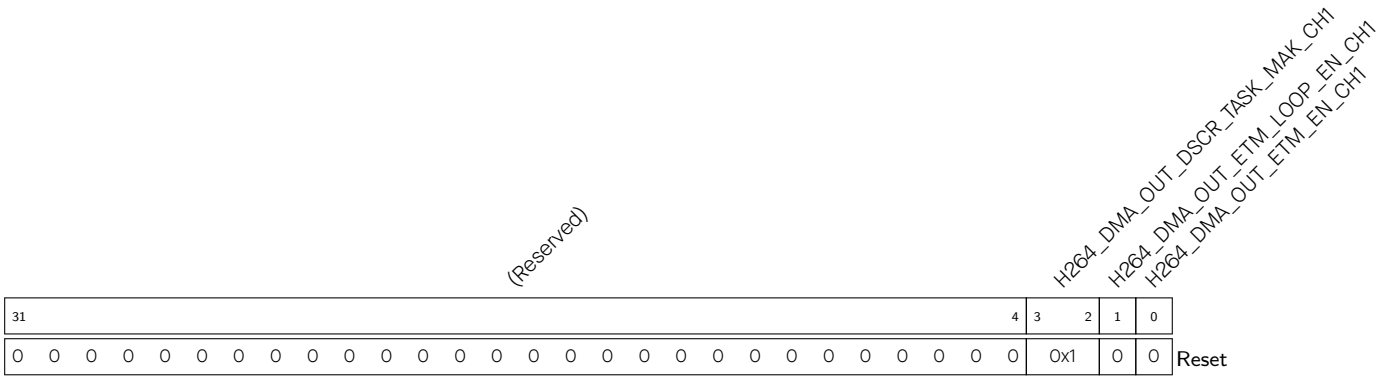
Register 41.80. H264_DMA_OUT_ARB_CH1_REG (0x013C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH1 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_INTER_OUT_ARB_PRIORITY_CH1 配置 TX 通道 1 的内部存储器访问优先级。(R/W)

Register 41.81. H264_DMA_OUT_ETM_CONF_CH1_REG (0x0168)



H264_DMA_OUT_ETM_EN_CH1 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH1 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH1 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.82. H264_DMA_OUT_CONF0_CH2_REG (0x0200)

(Reserved)	H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH2	H264_DMA_OUT_CMD_DISABLE_CH2	H264_DMA_OUT_RST_CH2	(Reserved)	H264_DMA_OUT_PAGE_BOUND_EN_CH2	(Reserved)	H264_DMA_OUT_MEM_BURST_LENGTH_CH2	(Reserved)	H264_DMA_OUT_CHECK_OWNER_CH2	H264_DMA_OUT_ECC_AES_EN_CH2	H264_DMA_OUTDSCR_BURST_EN_CH2	H264_DMA_OUT_EOF_MODE_CH2	H264_DMA_OUT_AUTO_WRBACK_CH2	
31 27	26	25	24	23 13	12	11 9	8 6	5	4	3	2	1	0	
0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0x0	0	0	0	0	1	0	Reset

H264_DMA_OUT_AUTO_WRBACK_CH2 配置是否在接收到由 outlink 描述符指向的所有数据时启用自动 outlink 写回。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_EOF_MODE_CH2 为 TX 通道 2 配置 EOF 标志生成模式。

0：当最后的数据已被推入 DMA 的 FIFO 时生成 EOF 标志

1：当最后的数据已从 DMA 的 FIFO 中弹出时生成 EOF 标志

(R/W)

H264_DMA_OUTDSCR_BURST_EN_CH2 配置是否启用 TX 通道 2 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ECC_AES_EN_CH2 配置 DMA 是否可以通过 TX 通道 2 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问 ECC/AES 区域。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_OUT_CHECK_OWNER_CH2 配置是否启用 TX 通道 2 的 owner 位检查。

0：禁用

1：启用

(R/W)

见下页...

Register 41.82. H264_DMA_OUT_CONF0_CH2_REG (0x0200)

接上页...

H264_DMA_OUT_MEM_BURST_LENGTH_CH2 配置 TX 通道 2 的突发长度。

0：8 字节
1：16 字节
2：32 字节
3：64 字节
4：128 字节
5 ~ 7：无效
(R/W)

H264_DMA_OUT_PAGE_BOUND_EN_CH2 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界
1：AXI 读取的数据不会跨越地址边界
(R/W)

H264_DMA_OUT_RST_CH2 写入 1 然后写入 0 以重置 TX 通道 2。(R/W)

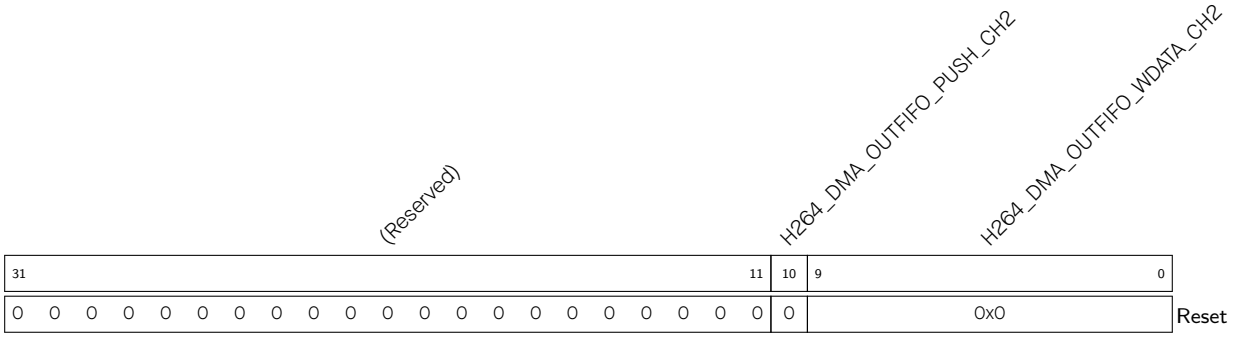
H264_DMA_OUT_CMD_DISABLE_CH2 配置是否禁用 TX 通道 2 上的命令。

0：禁用
1：启用
(R/W)

H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH2 配置是否禁用 TX 通道 2 的权重仲裁。

0：启用
1：禁用
(R/W)

Register 41.83. H264_DMA_OUT_PUSH_CH2_REG (0x0218)



H264_DMA_OUTFIFO_WDATA_CH2 表示需要推入 DMA TX FIFO 的数据。(R/W)

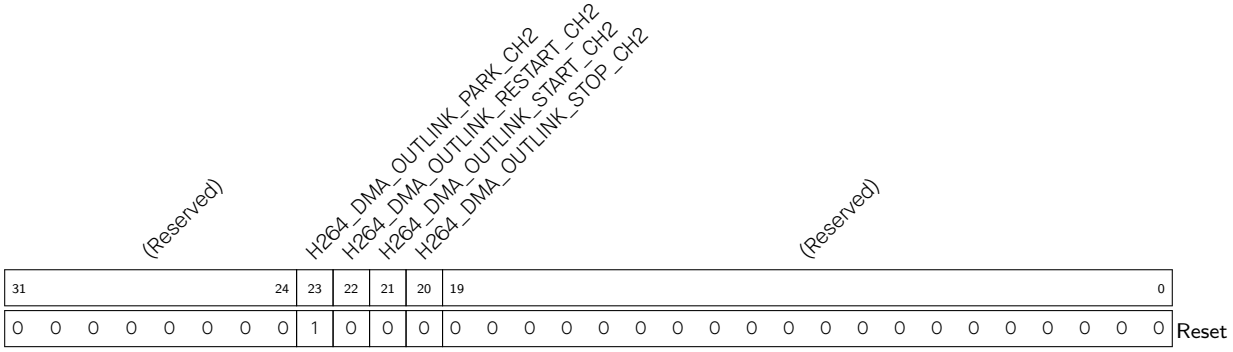
H264_DMA_OUTFIFO_PUSH_CH2 配置是否将数据推入 DMA TX FIFO。

0：无效

1：推送

(R/W/SC)

Register 41.84. H264_DMA_OUT_LINK_CONF_CH2_REG (0x021C)



H264_DMA_OUTLINK_STOP_CH2 配置是否停止处理 outlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_OUTLINK_START_CH2 配置是否开始处理 outlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH2 配置是否从最后一个地址重新开始一个新的 outlink。

0：无效

1：重新开始

(R/W/SC)

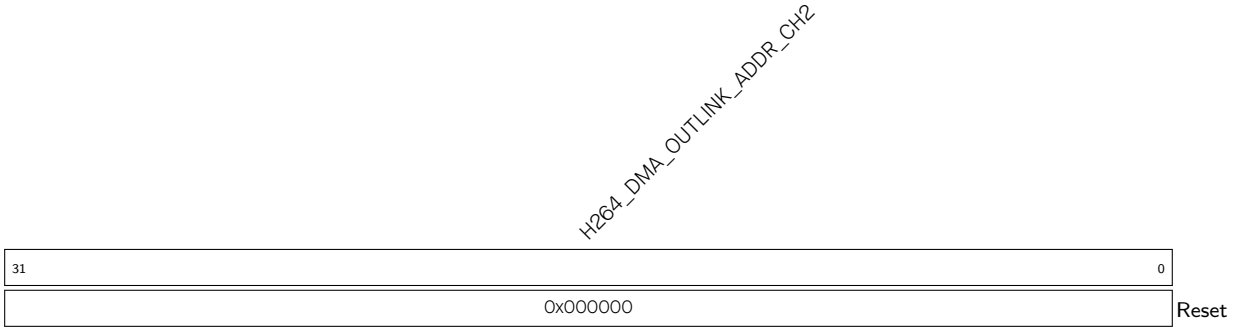
H264_DMA_OUTLINK_PARK_CH2 表示传输描述符的 FSM 的状态。

0：outlink 描述符的 FSM 正在工作

1：outlink 描述符的 FSM 处于空闲状态

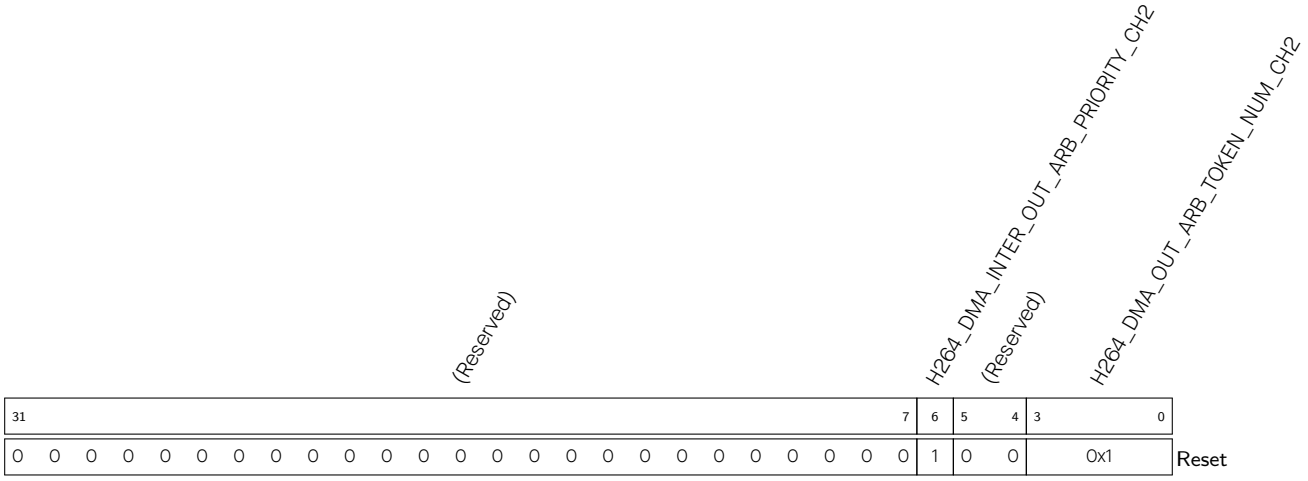
(RO)

Register 41.85. H264_DMA_OUT_LINK_ADDR_CH2_REG (0x0220)



H264_DMA_OUTLINK_ADDR_CH2 配置第一个 outlink 描述符的地址。(R/W)

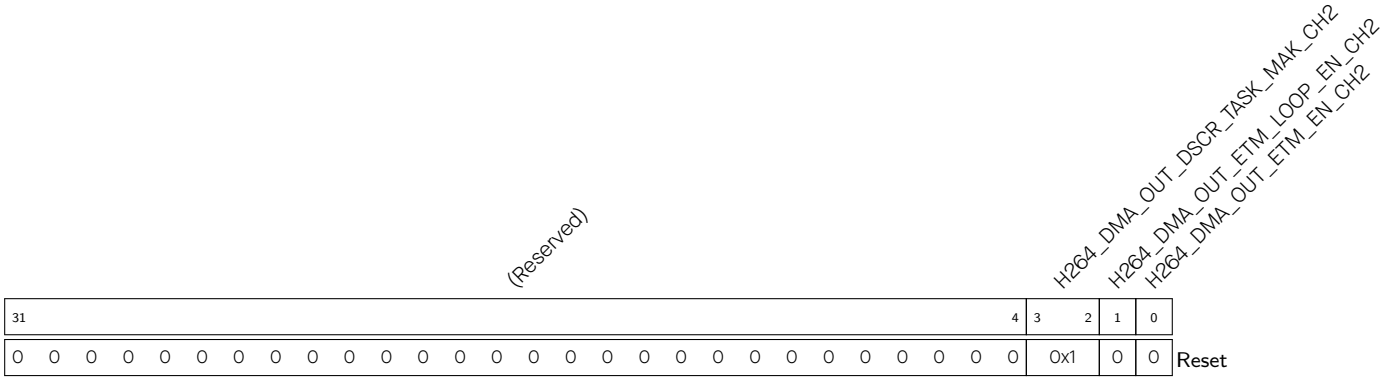
Register 41.86. H264_DMA_OUT_ARB_CH2_REG (0x023C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH2 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_INTER_OUT_ARB_PRIORITY_CH2 配置 TX 通道 2 的内部存储器访问优先级。

Register 41.87. H264_DMA_OUT_ETM_CONF_CH2_REG (0x0268)



H264_DMA_OUT_ETM_EN_CH2 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH2 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH2 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.88. H264_DMA_OUT_CONF0_CH3_REG (0x0300)

31–27	26	25–13	12	11–9	8–6	5	4	3	2	1	0	
(Reserved)	H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH3	(Reserved)	H264_DMA_OUT_PAGE_BOUND_EN_CH3	(Reserved)	H264_DMA_OUT_MEM_BURST_LENGTH_CH3	(Reserved)	H264_DMA_OUT_CHECK_OWNER_CH3	H264_DMA_OUT_ECC_AES_EN_CH3	H264_DMA_OUTDSCR_BURST_EN_CH3	H264_DMA_OUT_EOF_MODE_CH3	H264_DMA_OUT_AUTO_WRBACK_CH3	
0 0 0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0x0	0	0	0	0	1	0	Reset

H264_DMA_OUT_AUTO_WRBACK_CH3 配置是否在接收到由 outlink 描述符指向的所有数据时启用自动 outlink 写回。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_EOF_MODE_CH3 为 TX 通道 3 配置 EOF 标志生成模式。

0：当最后的数据已被推入 DMA 的 FIFO 时生成 EOF 标志

1：当最后的数据已从 DMA 的 FIFO 中弹出时生成 EOF 标志

(R/W)

H264_DMA_OUTDSCR_BURST_EN_CH3 配置是否启用 TX 通道 3 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ECC_AES_EN_CH3 配置 DMA 是否可以通过 TX 通道 3 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问 ECC/AES 区域。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_OUT_CHECK_OWNER_CH3 配置是否启用 TX 通道 3 的 owner 位检查。

0：禁用

1：启用

(R/W)

见下页...

Register 41.88. H264_DMA_OUT_CONF0_CH3_REG (0x0300)

接上页...

H264_DMA_OUT_MEM_BURST_LENGTH_CH3 配置 TX 通道 3 的突发长度。

0：8 字节
1：16 字节
2：32 字节
3：64 字节
4：128 字节
5 ~ 7：无效
(R/W)

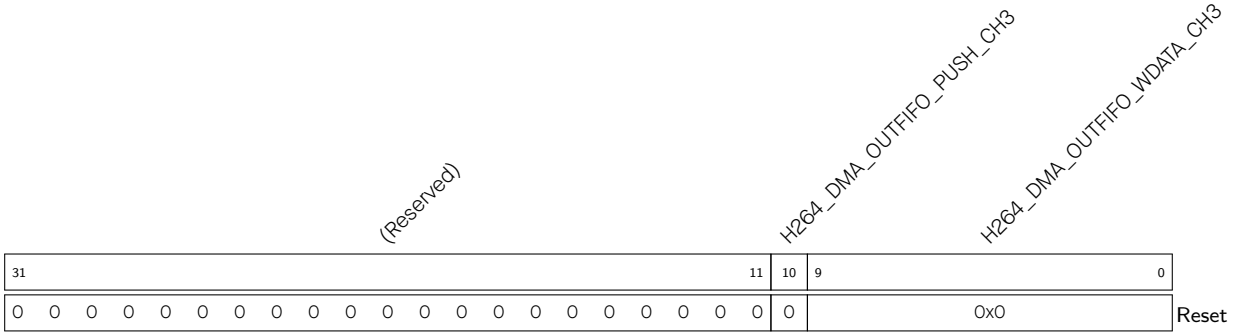
H264_DMA_OUT_PAGE_BOUND_EN_CH3 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界
1：AXI 读取的数据不会跨越地址边界
(R/W)

H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH3 配置是否禁用 TX 通道 3 的权重仲裁。

0：启用
1：禁用
(R/W)

Register 41.89. H264_DMA_OUT_PUSH_CH3_REG (0x0318)

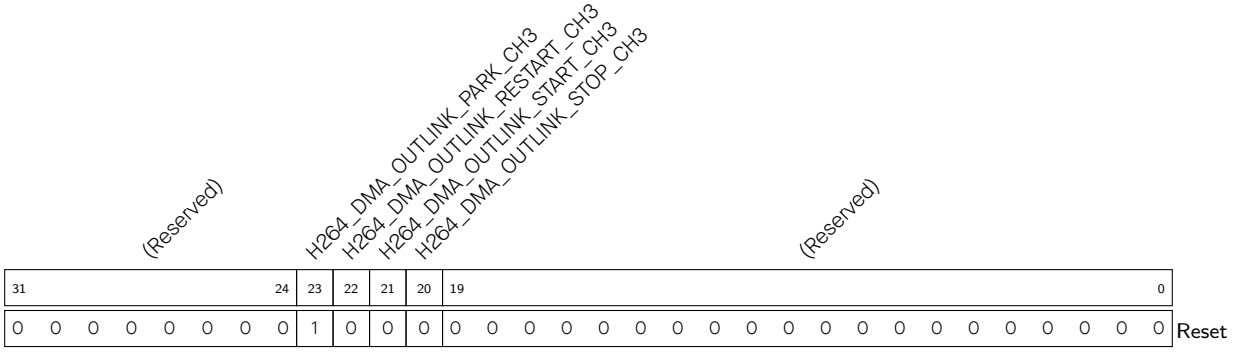


H264_DMA_OUTFIFO_WDATA_CH3 表示需要推入 DMA TX FIFO 的数据。(R/W)

H264_DMA_OUTFIFO_PUSH_CH3 配置是否将数据推入 DMA TX FIFO。

0：无效
1：推送
(R/W/SC)

Register 41.90. H264_DMA_OUT_LINK_CONF_CH3_REG (0x031C)



H264_DMA_OUTLINK_STOP_CH3 配置是否停止处理 outlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_OUTLINK_START_CH3 配置是否开始处理 outlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH3 配置是否从最后一个地址重新开始一个新的 outlink。

0：无效

1：重新开始

(R/W/SC)

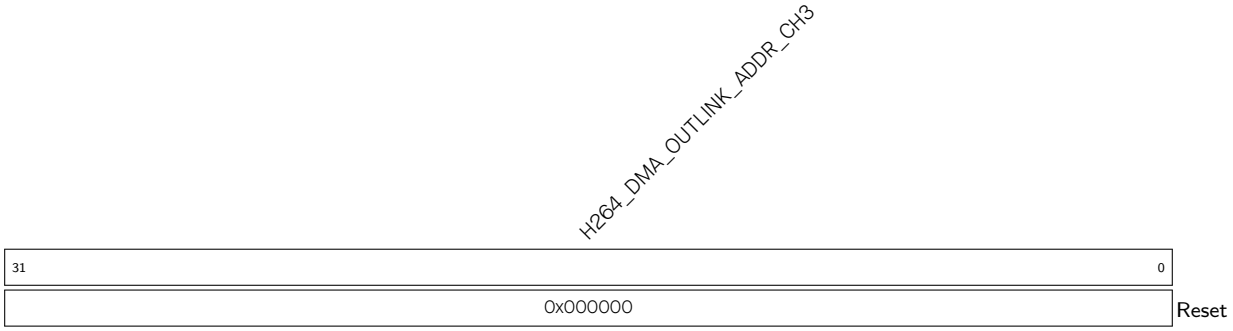
H264_DMA_OUTLINK_PARK_CH3 表示传输描述符的 FSM 的状态。

0：outlink 描述符的 FSM 正在工作

1：outlink 描述符的 FSM 处于空闲状态

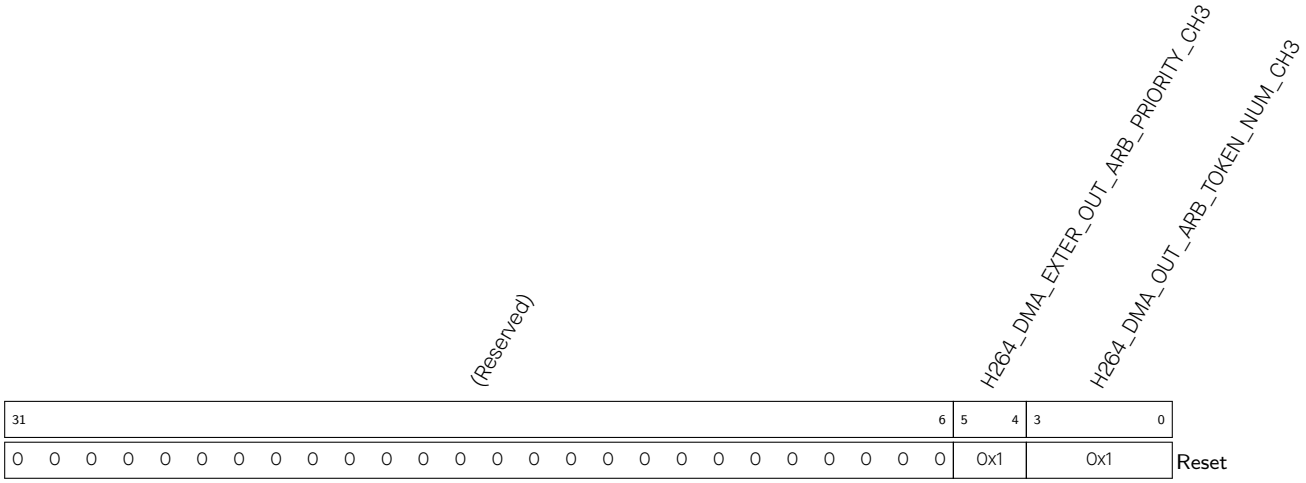
(RO)

Register 41.91. H264_DMA_OUT_LINK_ADDR_CH3_REG (0x0320)



H264_DMA_OUTLINK_ADDR_CH3 配置第一个 outlink 描述符的地址。(R/W)

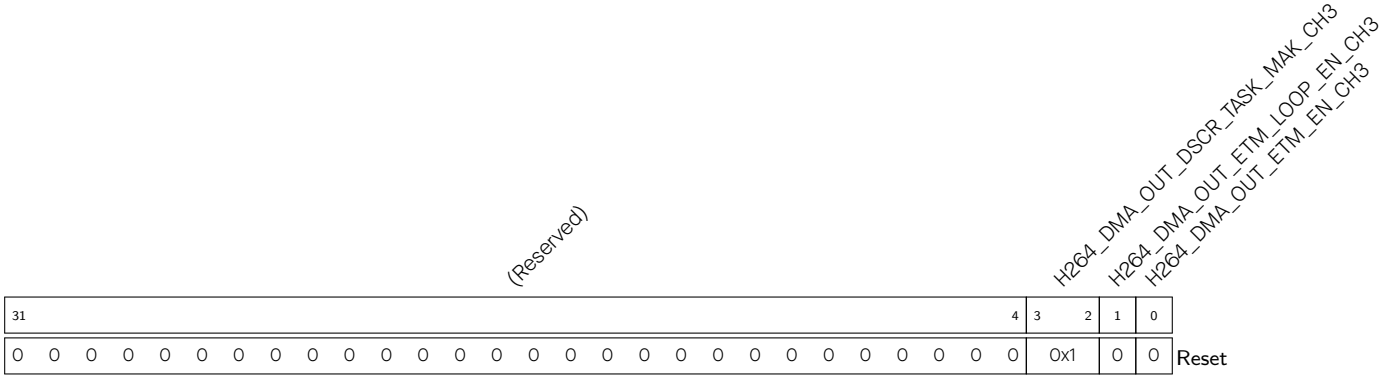
Register 41.92. H264_DMA_OUT_ARB_CH3_REG (0x033C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH3 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_EXTER_OUT_ARB_PRIORITY_CH3 配置 TX 通道 3 的外部存储器访问优先级。(R/W)

Register 41.93. H264_DMA_OUT_ETM_CONF_CH3_REG (0x0368)



H264_DMA_OUT_ETM_EN_CH3 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH3 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH3 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.94. H264_DMA_OUT_CONF0_CH4_REG (0x0400)

31–27	26	25–13	12	11–9	8–6	5	4	3	2	1	0	
(Reserved)	H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH4	(Reserved)	H264_DMA_OUT_PAGE_BOUND_EN_CH4	(Reserved)	H264_DMA_OUT_MEM_BURST_LENGTH_CH4	(Reserved)	H264_DMA_OUT_CHECK_OWNER_CH4	H264_DMA_OUT_ECC_AES_EN_CH4	H264_DMA_OUTDSCR_BURST_EN_CH4	H264_DMA_OUT_EOF_MODE_CH4	H264_DMA_OUT_AUTO_WRBACK_CH4	
0 0 0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0x0	0	0	0	0	1	0	Reset

H264_DMA_OUT_AUTO_WRBACK_CH4 配置是否在接收到由 outlink 描述符指向的所有数据时启用自动 outlink 写回。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_EOF_MODE_CH4 为 TX 通道 4 配置 EOF 标志生成模式。

0：当最后的数据已被推入 DMA 的 FIFO 时生成 EOF 标志

1：当最后的数据已从 DMA 的 FIFO 中弹出时生成 EOF 标志

(R/W)

H264_DMA_OUTDSCR_BURST_EN_CH4 配置是否启用 TX 通道 4 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ECC_AES_EN_CH4 配置 DMA 是否可以通过 TX 通道 4 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问 ECC/AES 区域。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_OUT_CHECK_OWNER_CH4 配置是否启用 TX 通道 4 的 owner 位检查。

0：禁用

1：启用

(R/W)

见下页...

Register 41.94. H264_DMA_OUT_CONF0_CH4_REG (0x0400)

接上页...

H264_DMA_OUT_MEM_BURST_LENGTH_CH4 配置 TX 通道 4 的突发长度。

0：8 字节
1：16 字节
2：32 字节
3：64 字节
4：128 字节
5 ~ 7：无效
(R/W)

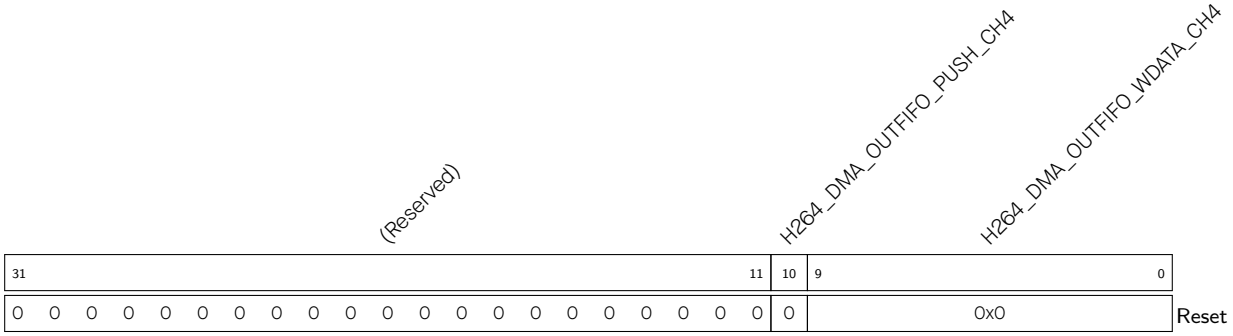
H264_DMA_OUT_PAGE_BOUND_EN_CH4 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界
1：AXI 读取的数据不会跨越地址边界
(R/W)

H264_DMA_OUT_ARB_WEIGHT_OPT_DIS_CH4 配置是否禁用 TX 通道 4 的权重仲裁。

0：启用
1：禁用
(R/W)

Register 41.95. H264_DMA_OUT_PUSH_CH4_REG (0x0418)

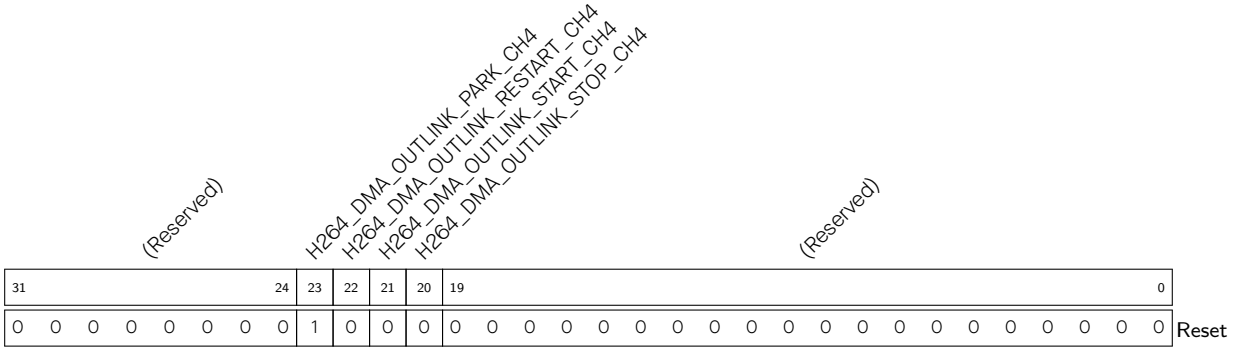


H264_DMA_OUTFIFO_WDATA_CH4 表示需要推入 DMA TX FIFO 的数据。(R/W)

H264_DMA_OUTFIFO_PUSH_CH4 配置是否将数据推入 DMA TX FIFO。

0：无效
1：推送
(R/W/SC)

Register 41.96. H264_DMA_OUT_LINK_CONF_CH4_REG (0x041C)



H264_DMA_OUTLINK_STOP_CH4 配置是否停止处理 outlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_OUTLINK_START_CH4 配置是否开始处理 outlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_OUTLINK_RESTART_CH4 配置是否从最后一个地址重新开始一个新的 outlink。

0：无效

1：重新开始

(R/W/SC)

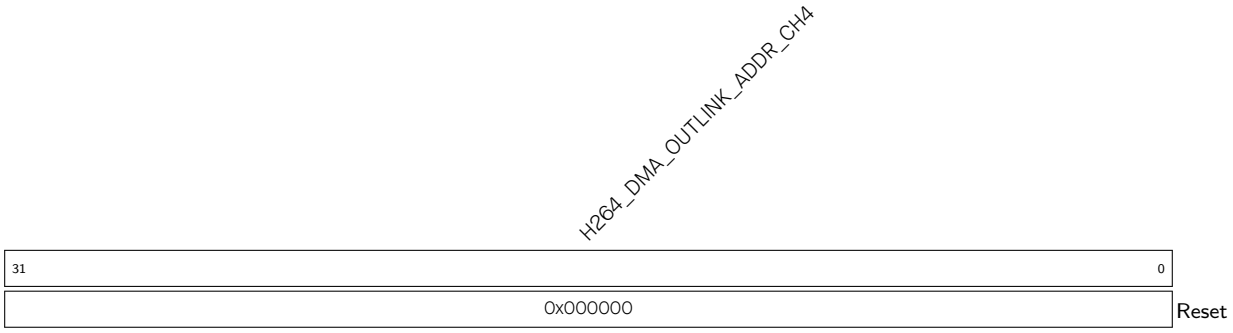
H264_DMA_OUTLINK_PARK_CH4 表示传输描述符的 FSM 的状态。

0：outlink 描述符的 FSM 正在工作

1：outlink 描述符的 FSM 处于空闲状态

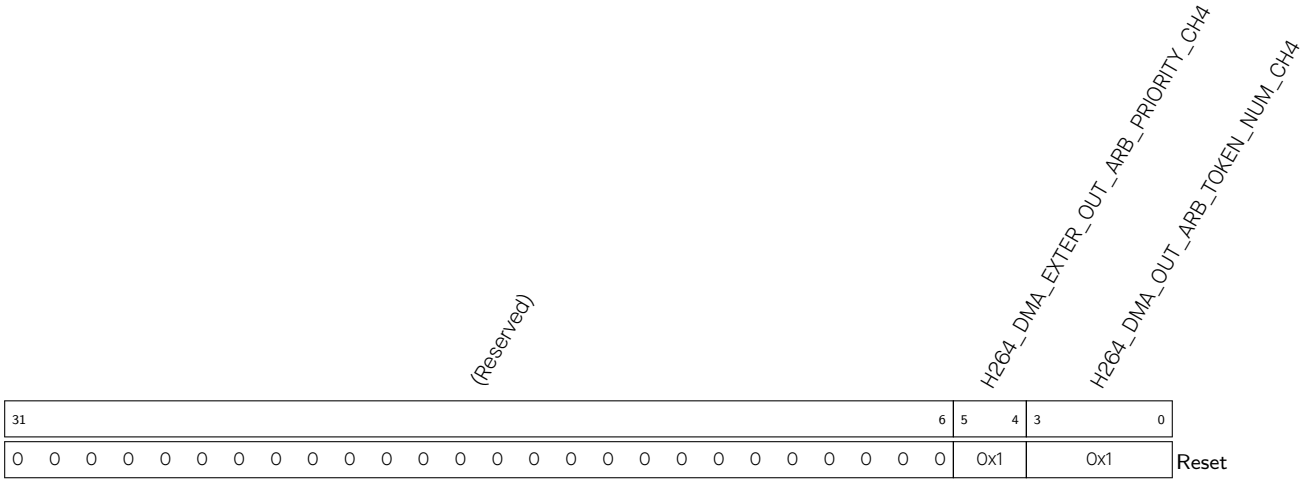
(RO)

Register 41.97. H264_DMA_OUT_LINK_ADDR_CH4_REG (0x0420)



H264_DMA_OUTLINK_ADDR_CH4 配置第一个 outlink 描述符的地址。(R/W)

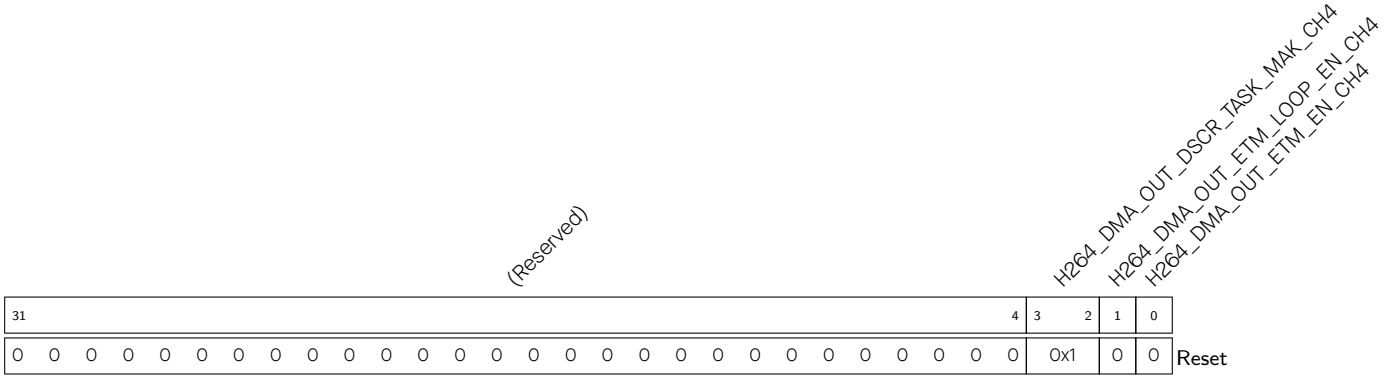
Register 41.98. H264_DMA_OUT_ARB_CH4_REG (0x043C)



H264_DMA_OUT_ARB_TOKEN_NUM_CH4 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_EXTER_OUT_ARB_PRIORITY_CH4 配置 TX 通道 4 的外部存储器访问优先级。(R/W)

Register 41.99. H264_DMA_OUT_ETM_CONF_CH4_REG (0x0468)



H264_DMA_OUT_ETM_EN_CH4 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_OUT_ETM_LOOP_EN_CH4 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_OUT_DSCR_TASK_MAK_CH4 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.100. H264_DMA_IN_CONF0_CH0_REG (0x0500)

位	字段	Reset
31:27	(Reserved)	0 0 0 0 0
26	H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH0	0
25	H264_DMA_IN_CMD_DISABLE_CH0	0
24	H264_DMA_IN_RST_CH0	0
23:13	(Reserved)	0 0 0 0 0 0 0 0 0 0 0
12	H264_DMA_IN_PAGE_BOUND_EN_CH0	0
11:9	(Reserved)	0 0 0
8:6	H264_DMA_IN_MEM_BURST_LENGTH_CH0	0x0
5	(Reserved)	0
4	H264_DMA_IN_CHECK_OWNER_CH0	0
3	H264_DMA_IN_ECC_AES_EN_CH0	0
2	H264_DMA_INDSCR_BURST_EN_CH0	0
1:0	(Reserved)	0 0

H264_DMA_INDSCR_BURST_EN_CH0 配置是否启用 RX 通道 0 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ECC_AES_EN_CH0 配置 DMA 是否可以通过 RX 通道 0 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_IN_CHECK_OWNER_CH0 配置是否启用 RX 通道 0 的 owner 位检查。

0：禁用

1：启用

(R/W)

H264_DMA_IN_MEM_BURST_LENGTH_CH0 配置 RX 通道 0 的突发长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

H264_DMA_IN_PAGE_BOUND_EN_CH0 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界

1：AXI 读取的数据不会跨越地址边界

(R/W)

H264_DMA_IN_RST_CH0 写入 1 然后写入 0 以重置 RX 通道 0。(R/W)

见下页...

Register 41.100. H264_DMA_IN_CONF0_CH0_REG (0x0500)

接上页...

H264_DMA_IN_CMD_DISABLE_CH0 配置是否禁用 RX 通道 0 上的命令。

0：禁用

1：启用

(R/W)

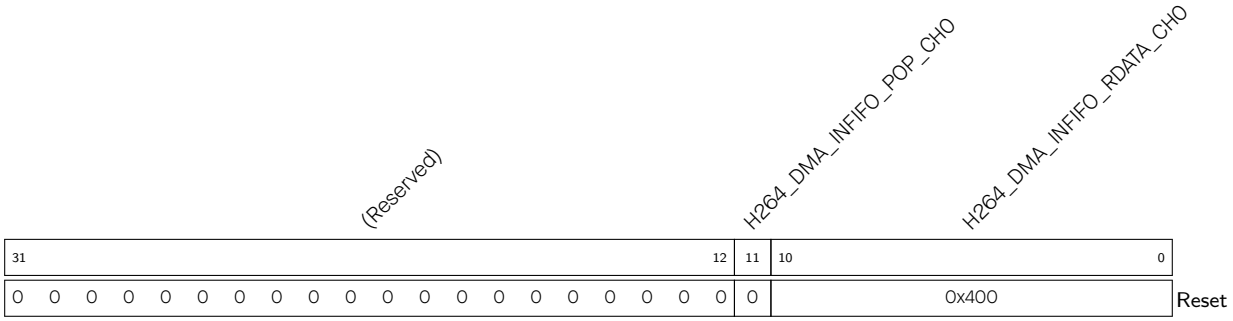
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH0 配置是否禁用 RX 通道 0 的权重仲裁。

0：启用

1：禁用

(R/W)

Register 41.101. H264_DMA_IN_POP_CH0_REG (0x0518)



H264_DMA_INFIFO_RDATA_CH0 表示从 DMA RX FIFO 弹出的数据。(RO)

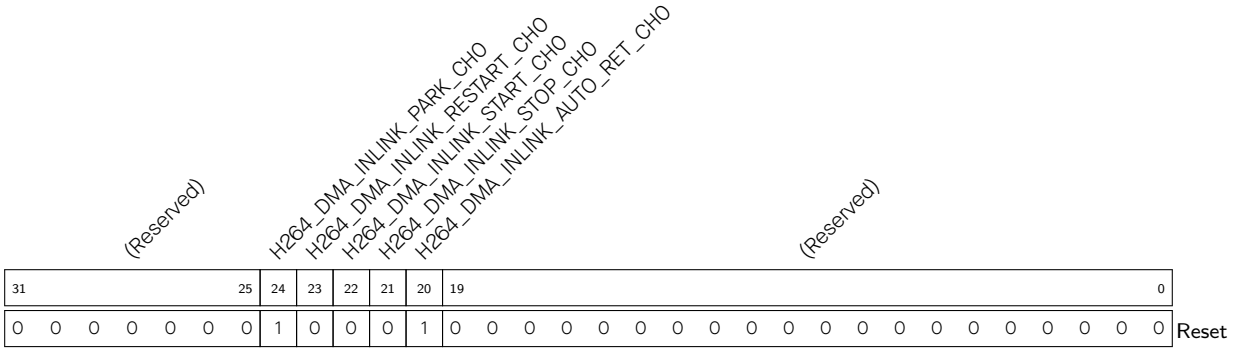
H264_DMA_INFIFO_POP_CH0 配置是否从 DMA RX FIFO 弹出数据。

0：无效

1：弹出

(R/W/SC)

Register 41.102. H264_DMA_IN_LINK_CONF_CH0_REG (0x051C)



H264_DMA_INLINK_AUTO_RET_CH0 配置在数据接收过程中出现错误时是否返回到当前接收描述符的地址。

0：不返回

1：返回

(R/W)

H264_DMA_INLINK_STOP_CH0 配置是否停止处理 inlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_INLINK_START_CH0 配置是否开始处理 inlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_INLINK_RESTART_CH0 配置是否从最后一个地址重新开始一个新的 inlink。

0：无效

1：重新开始

(R/W/SC)

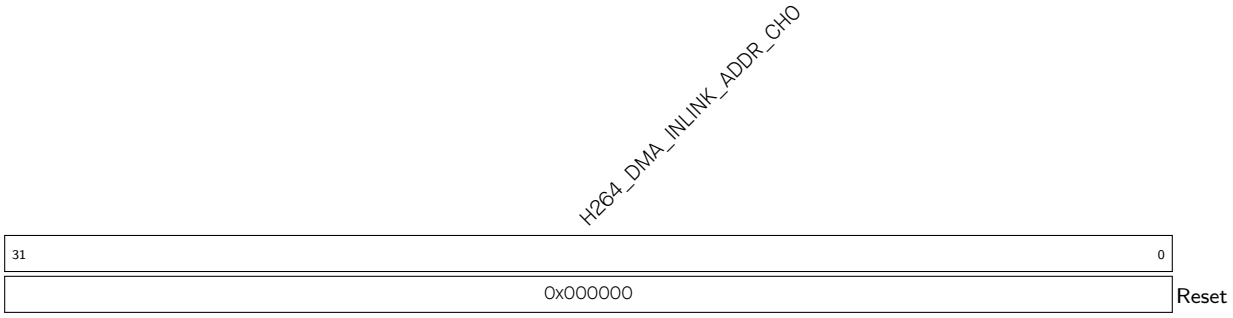
H264_DMA_INLINK_PARK_CH0 表示 inlink 描述符的 FSM 的状态。

0：inlink 描述符的 FSM 正在工作

1：inlink 描述符的 FSM 处于空闲状态

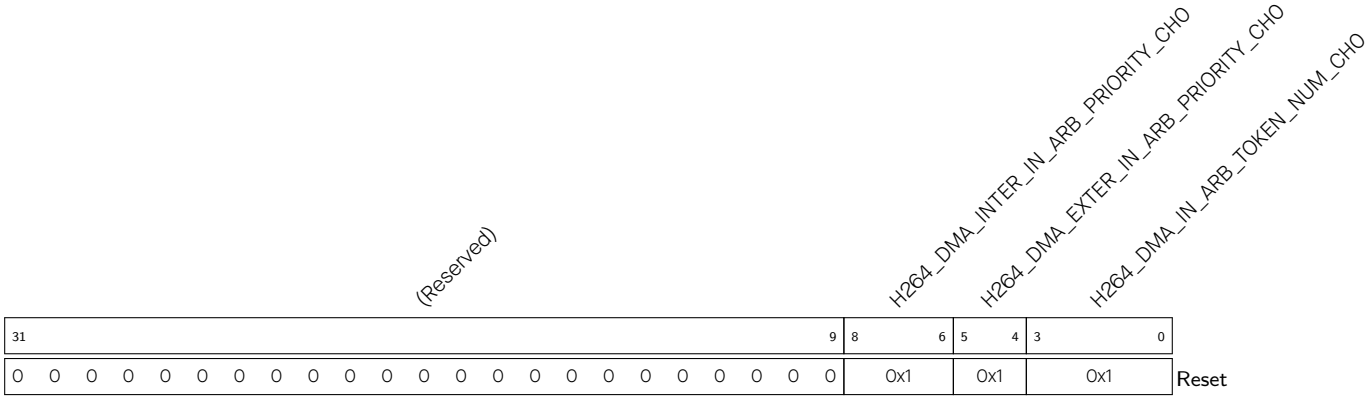
(RO)

Register 41.103. H264_DMA_IN_LINK_ADDR_CH0_REG (0x0520)



H264_DMA_INLINK_ADDR_CH0 配置第一个 inlink 描述符的地址。(R/W)

Register 41.104. H264_DMA_IN_ARB_CH0_REG (0x0540)

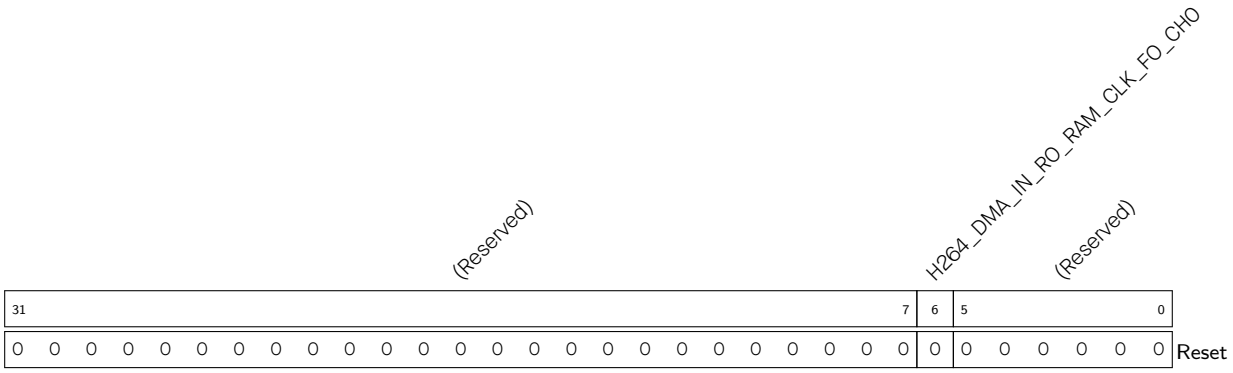


H264_DMA_IN_ARB_TOKEN_NUM_CH0 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_EXTER_IN_ARB_PRIORITY_CH0 配置 RX 通道 0 的外部存储器访问优先级。(R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH0 配置 RX 通道 0 的内部存储器访问优先级。(R/W)

Register 41.105. H264_DMA_IN_RO_PD_CONF_CH0_REG (0x0548)



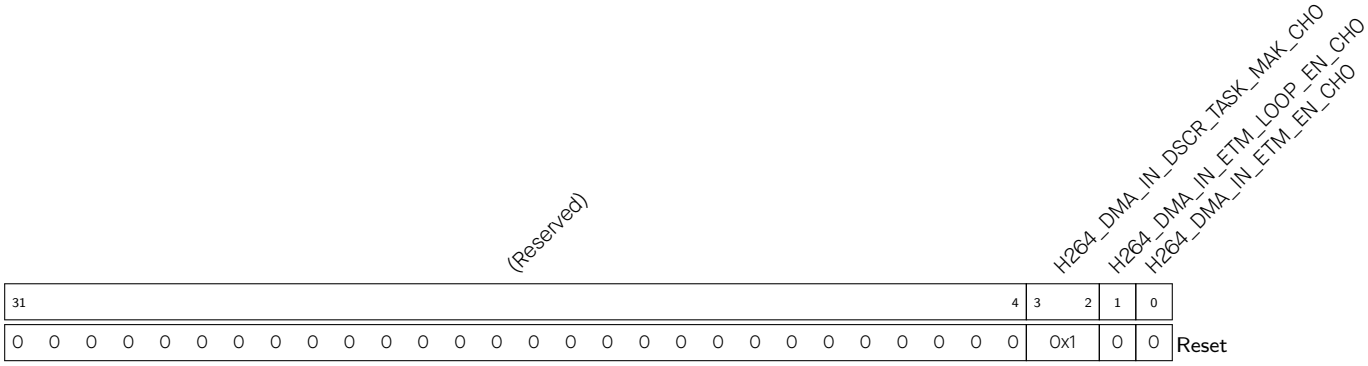
H264_DMA_IN_RO_RAM_CLK_FO_CH0 配置是否强制启用 DMA RX 重新排序 RAM 的时钟门控。

0：访问 DMA 中的 RAM 时将使用时钟门控

1：访问 DMA 中的 RAM 时强制打开时钟并绕过时钟门控

(R/W)

Register 41.106. H264_DMA_IN_ETM_CONF_CH0_REG (0x056C)



H264_DMA_IN_ETM_EN_CH0 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ETM_LOOP_EN_CH0 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH0 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.107. H264_DMA_IN_CONF0_CH1_REG (0x0600)

Bits	Field	Reset
31–27	(Reserved)	0 0 0 0 0
26	H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH1	0
25	H264_DMA_IN_CMD_DISABLE_CH1	0
24	H264_DMA_IN_RST_CH1	0
23–13	(Reserved)	0 0 0 0 0 0 0 0 0 0 0
12	H264_DMA_IN_PAGE_BOUND_EN_CH1	0
11–9	(Reserved)	0 0 0
8–6	H264_DMA_IN_MEM_BURST_LENGTH_CH1	0x0
5	(Reserved)	0
4	H264_DMA_IN_CHECK_OWNER_CH1	0
3	H264_DMA_IN_ECC_AES_EN_CH1	0
2	H264_DMA_INDSCR_BURST_EN_CH1	0
1–0	(Reserved)	0 0

H264_DMA_INDSCR_BURST_EN_CH1 配置是否启用 RX 通道 1 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ECC_AES_EN_CH1 配置 DMA 是否可以通过 RX 通道 1 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_IN_CHECK_OWNER_CH1 配置是否启用 RX 通道 1 的 owner 位检查。

0：禁用

1：启用

(R/W)

H264_DMA_IN_MEM_BURST_LENGTH_CH1 配置 RX 通道 1 的突发长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

H264_DMA_IN_PAGE_BOUND_EN_CH1 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界

1：AXI 读取的数据不会跨越地址边界

(R/W)

H264_DMA_IN_RST_CH1 写入 1 然后写入 0 以重置 RX 通道 1。(R/W)

见下页...

Register 41.107. H264_DMA_IN_CONF0_CH1_REG (0x0600)

接上页...

H264_DMA_IN_CMD_DISABLE_CH1 配置是否禁用 RX 通道 1 上的命令。

0：禁用

1：启用

(R/W)

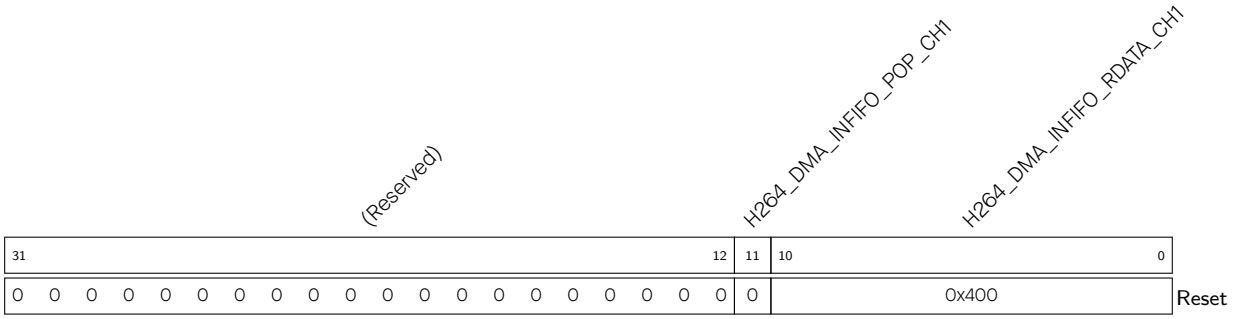
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH1 配置是否禁用 RX 通道 1 的权重仲裁。

0：启用

1：禁用

(R/W)

Register 41.108. H264_DMA_IN_POP_CH1_REG (0x0618)



H264_DMA_INFIFO_RDATA_CH1 表示从 DMA RX FIFO 弹出的数据。(RO)

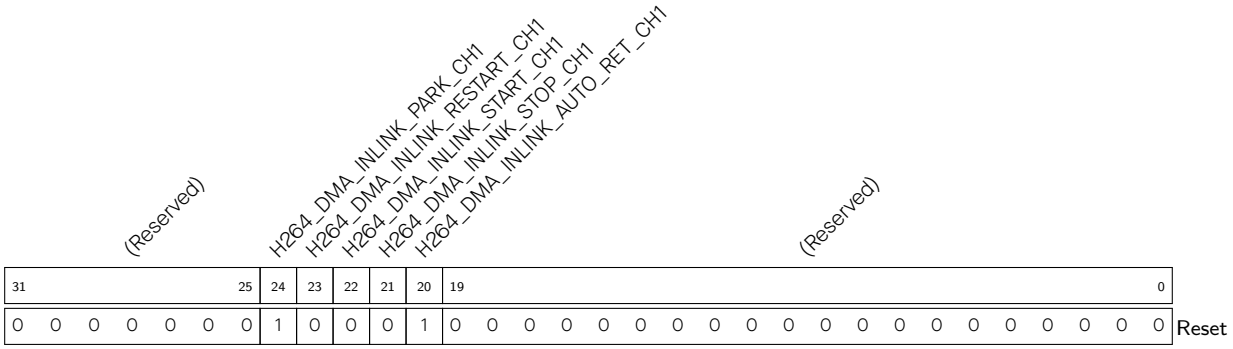
H264_DMA_INFIFO_POP_CH1 配置是否从 DMA RX FIFO 弹出数据。

0：无效

1：弹出

(R/W/SC)

Register 41.109. H264_DMA_IN_LINK_CONF_CH1_REG (0x061C)



H264_DMA_INLINK_AUTO_RET_CH1 配置在数据接收过程中出现错误时是否返回到当前接收描述符的地址。

0：不返回

1：返回

(R/W)

H264_DMA_INLINK_STOP_CH1 配置是否停止处理 inlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_INLINK_START_CH1 配置是否开始处理 inlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_INLINK_RESTART_CH1 配置是否从最后一个地址重新开始一个新的 inlink。

0：无效

1：重新开始

(R/W/SC)

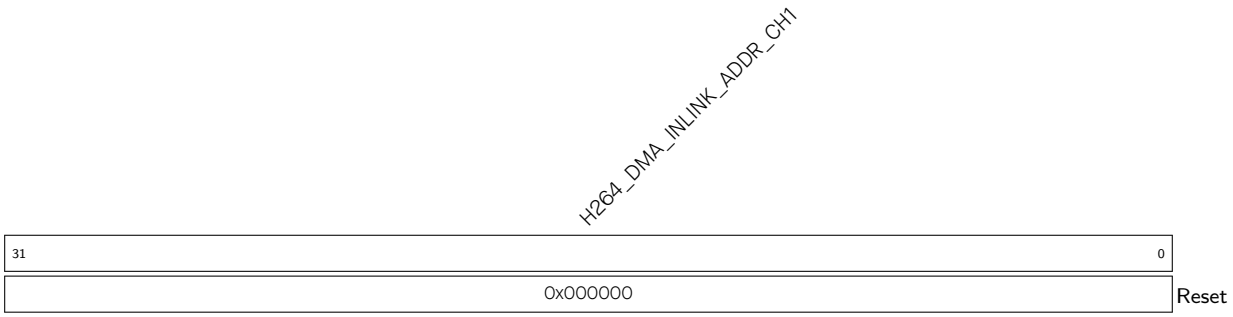
H264_DMA_INLINK_PARK_CH1 表示 inlink 描述符的 FSM 的状态。

0：inlink 描述符的 FSM 正在工作

1：inlink 描述符的 FSM 处于空闲状态

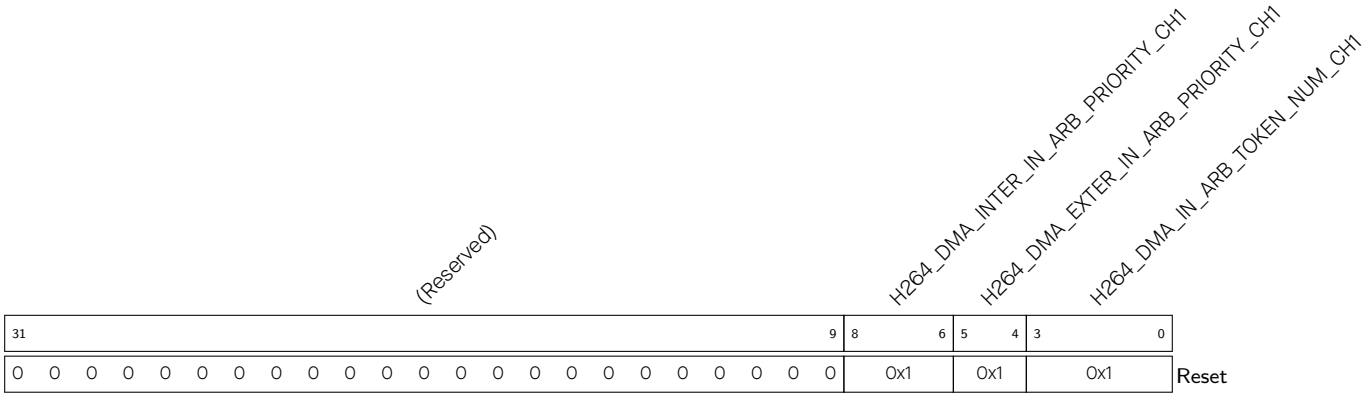
(RO)

Register 41.110. H264_DMA_IN_LINK_ADDR_CH1_REG (0x0620)



H264_DMA_INLINK_ADDR_CH1 配置第一个 inlink 描述符的地址。(R/W)

Register 41.111. H264_DMA_IN_ARB_CH1_REG (0x0640)

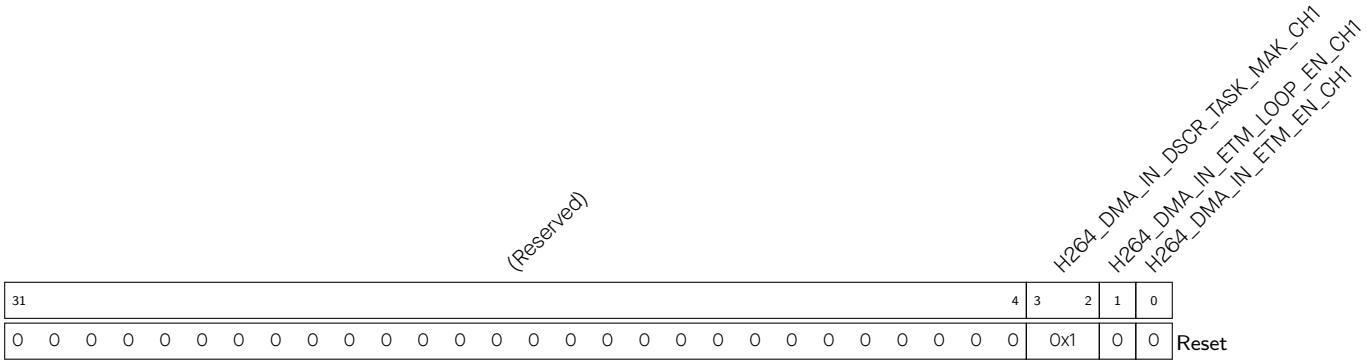


H264_DMA_IN_ARB_TOKEN_NUM_CH1 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_EXTER_IN_ARB_PRIORITY_CH1 配置 RX 通道 1 的外部存储器访问优先级。(R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH1 配置 RX 通道 1 的内部存储器访问优先级。(R/W)

Register 41.112. H264_DMA_IN_ETM_CONF_CH1_REG (0x0648)



H264_DMA_IN_ETM_EN_CH1 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ETM_LOOP_EN_CH1 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH1 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.113. H264_DMA_IN_CONF0_CH2_REG (0x0700)

Bits	31–27	26	25	24	23–13	12	11–9	8–6	5	4	3	2	1–0
Field	(Reserved)	H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH2	H264_DMA_IN_CMD_DISABLE_CH2	H264_DMA_IN_RST_CH2	(Reserved)	H264_DMA_IN_PAGE_BOUND_EN_CH2	(Reserved)	H264_DMA_IN_MEM_BURST_LENGTH_CH2	(Reserved)	H264_DMA_IN_CHECK_OWNER_CH2	H264_DMA_IN_ECC_AES_EN_CH2	H264_DMA_INDSCR_BURST_EN_CH2	(Reserved)
Reset	0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0x0	0	0	0	0	0 0

H264_DMA_INDSCR_BURST_EN_CH2 配置是否启用 RX 通道 2 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ECC_AES_EN_CH2 配置 DMA 是否可以通过 RX 通道 2 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_IN_CHECK_OWNER_CH2 配置是否启用 RX 通道 2 的 owner 位检查。

0：禁用

1：启用

(R/W)

H264_DMA_IN_MEM_BURST_LENGTH_CH2 配置 RX 通道 2 的突发长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

H264_DMA_IN_PAGE_BOUND_EN_CH2 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界

1：AXI 读取的数据不会跨越地址边界

(R/W)

H264_DMA_IN_RST_CH2 写入 1 然后写入 0 以重置 RX 通道 2。(R/W)

见下页...

Register 41.113. H264_DMA_IN_CONF0_CH2_REG (0x0700)

接上页...

H264_DMA_IN_CMD_DISABLE_CH2 配置是否禁用 RX 通道 2 上的命令。

0：禁用

1：启用

(R/W)

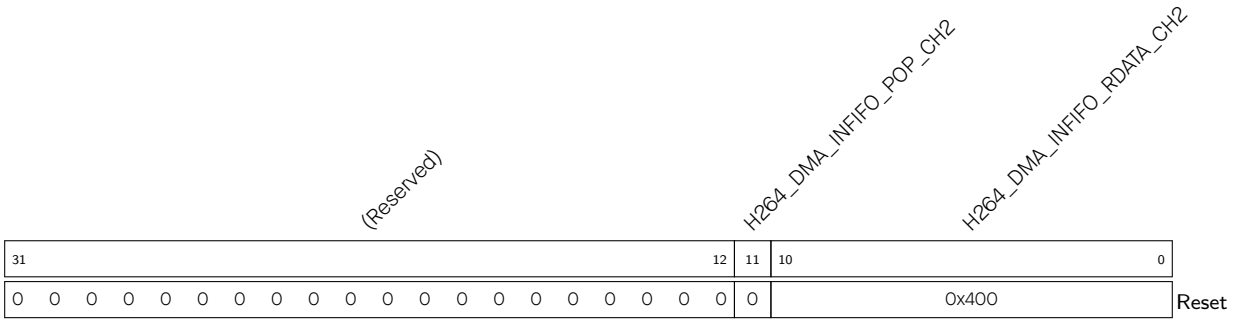
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH2 配置是否禁用 RX 通道 2 的权重仲裁。

0：启用

1：禁用

(R/W)

Register 41.114. H264_DMA_IN_POP_CH2_REG (0x0718)



H264_DMA_INFIFO_RDATA_CH2 表示从 DMA RX FIFO 弹出的数据。(RO)

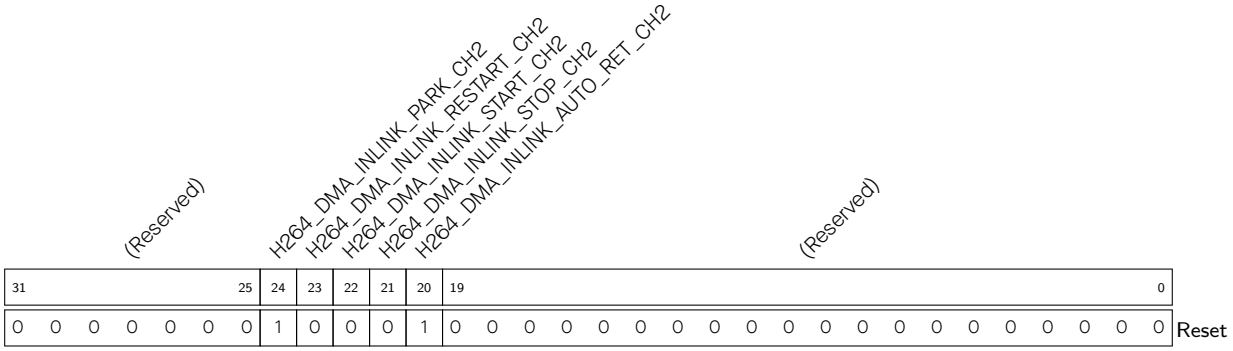
H264_DMA_INFIFO_POP_CH2 配置是否从 DMA RX FIFO 弹出数据。

0：无效

1：弹出

(R/W/SC)

Register 41.115. H264_DMA_IN_LINK_CONF_CH2_REG (0x071C)



H264_DMA_INLINK_AUTO_RET_CH2 配置在数据接收过程中出现错误时是否返回到当前接收描述符的地址。

0：不返回

1：返回

(R/W)

H264_DMA_INLINK_STOP_CH2 配置是否停止处理 inlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_INLINK_START_CH2 配置是否开始处理 inlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_INLINK_RESTART_CH2 配置是否从最后一个地址重新开始一个新的 inlink。

0：无效

1：重新开始

(R/W/SC)

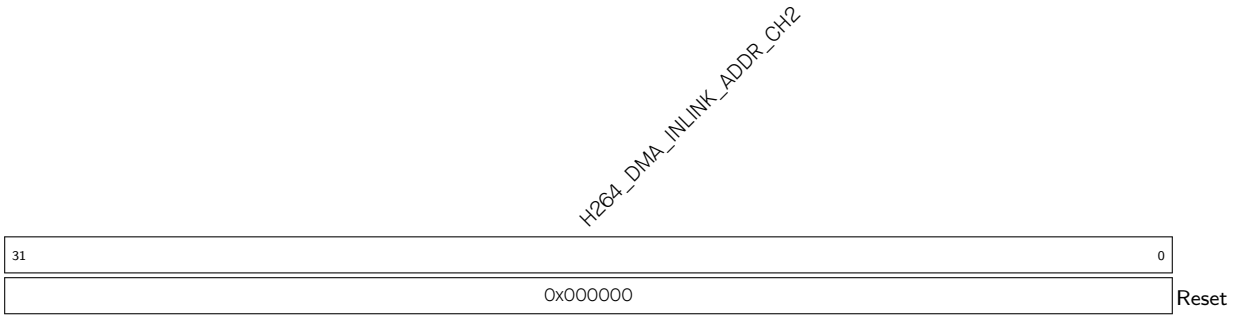
H264_DMA_INLINK_PARK_CH2 表示 inlink 描述符的 FSM 的状态。

0：inlink 描述符的 FSM 正在工作

1：inlink 描述符的 FSM 处于空闲状态

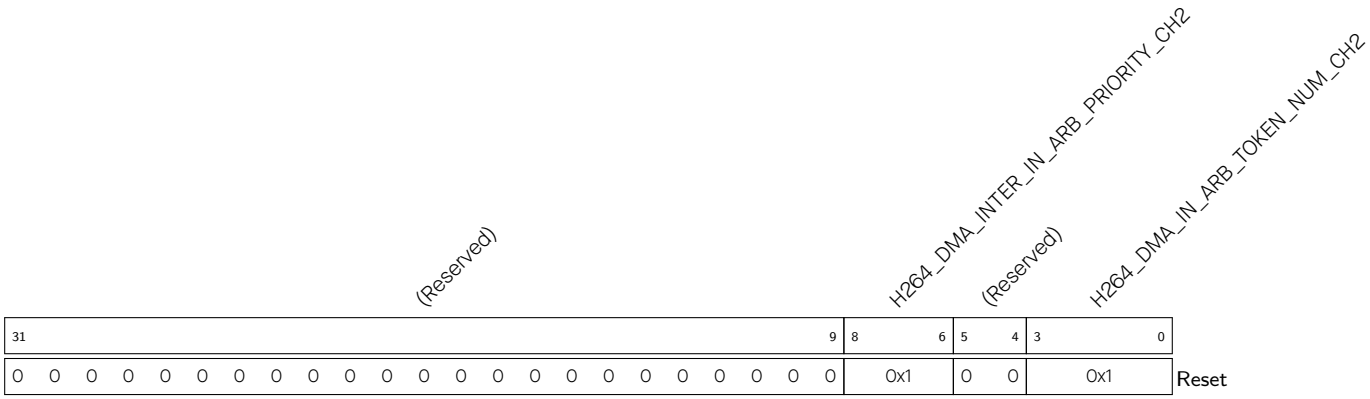
(RO)

Register 41.116. H264_DMA_IN_LINK_ADDR_CH2_REG (0x0720)



H264_DMA_INLINK_ADDR_CH2 配置第一个 inlink 描述符的地址。(R/W)

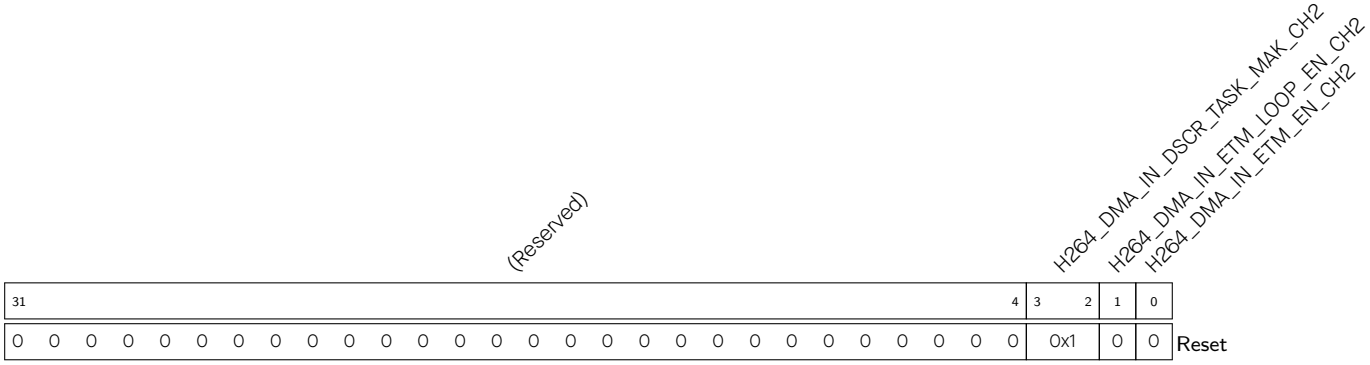
Register 41.117. H264_DMA_IN_ARB_CH2_REG (0x0740)



H264_DMA_IN_ARB_TOKEN_NUM_CH2 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH2 配置 RX 通道 2 的内部存储器访问优先级。(R/W)

Register 41.118. H264_DMA_IN_ETM_CONF_CH2_REG (0x0748)



H264_DMA_IN_ETM_EN_CH2 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ETM_LOOP_EN_CH2 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH2 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.119. H264_DMA_IN_CONF0_CH3_REG (0x0800)

Bits	31–27	26	25	24	23–13	12	11–9	8–6	5	4	3	2	1–0
Field	(Reserved)	H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH3	H264_DMA_IN_CMD_DISABLE_CH3	H264_DMA_IN_RST_CH3	(Reserved)	H264_DMA_IN_PAGE_BOUND_EN_CH3	(Reserved)	H264_DMA_IN_MEM_BURST_LENGTH_CH3	(Reserved)	H264_DMA_IN_CHECK_OWNER_CH3	H264_DMA_IN_ECC_AES_EN_CH3	H264_DMA_INDSCR_BURST_EN_CH3	(Reserved)
Reset	0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0x0	0	0	0	0	0 0

H264_DMA_INDSCR_BURST_EN_CH3 配置是否启用 RX 通道 3 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ECC_AES_EN_CH3 配置 DMA 是否可以通过 RX 通道 3 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_IN_CHECK_OWNER_CH3 配置是否启用 RX 通道 3 的 owner 位检查。

0：禁用

1：启用

(R/W)

H264_DMA_IN_MEM_BURST_LENGTH_CH3 配置 RX 通道 3 的突发长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

H264_DMA_IN_PAGE_BOUND_EN_CH3 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界

1：AXI 读取的数据不会跨越地址边界

(R/W)

H264_DMA_IN_RST_CH3 写入 1 然后写入 0 以重置 RX 通道 3。(R/W)

见下页...

Register 41.119. H264_DMA_IN_CONF0_CH3_REG (0x0800)

接上页...

H264_DMA_IN_CMD_DISABLE_CH3 配置是否禁用 RX 通道 3 上的命令。

0：禁用

1：启用

(R/W)

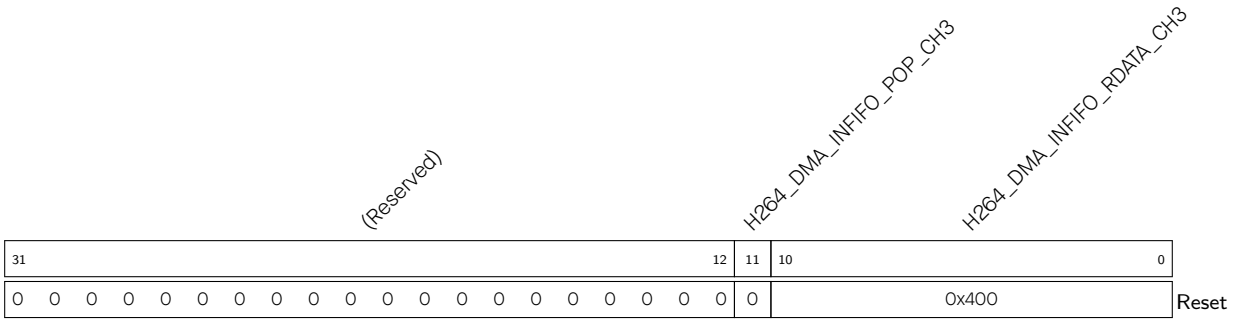
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH3 配置是否禁用 RX 通道 3 的权重仲裁。

0：启用

1：禁用

(R/W)

Register 41.120. H264_DMA_IN_POP_CH3_REG (0x0818)



H264_DMA_INFIFO_RDATA_CH3 表示从 DMA RX FIFO 弹出的数据。(RO)

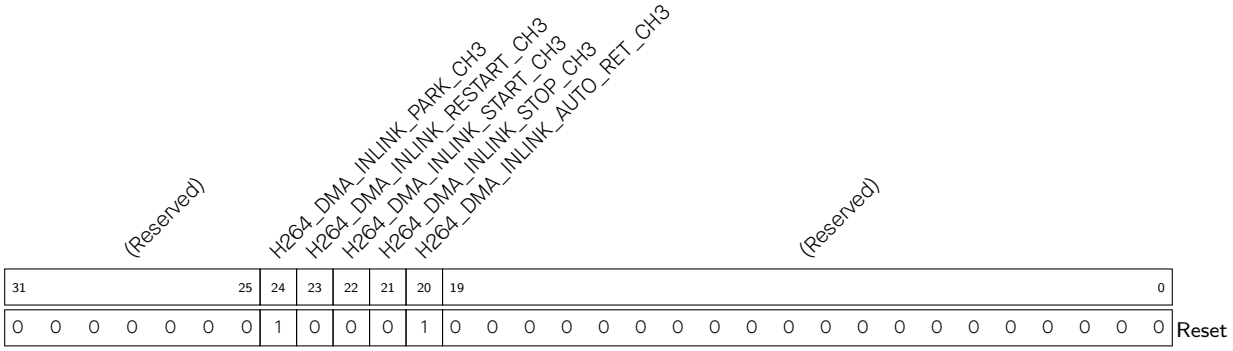
H264_DMA_INFIFO_POP_CH3 配置是否从 DMA RX FIFO 弹出数据。

0：无效

1：弹出

(R/W/SC)

Register 41.121. H264_DMA_IN_LINK_CONF_CH3_REG (0x081C)



H264_DMA_INLINK_AUTO_RET_CH3 配置在数据接收过程中出现错误时是否返回到当前接收描述符的地址。

0：不返回

1：返回

(R/W)

H264_DMA_INLINK_STOP_CH3 配置是否停止处理 inlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_INLINK_START_CH3 配置是否开始处理 inlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_INLINK_RESTART_CH3 配置是否从最后一个地址重新开始一个新的 inlink。

0：无效

1：重新开始

(R/W/SC)

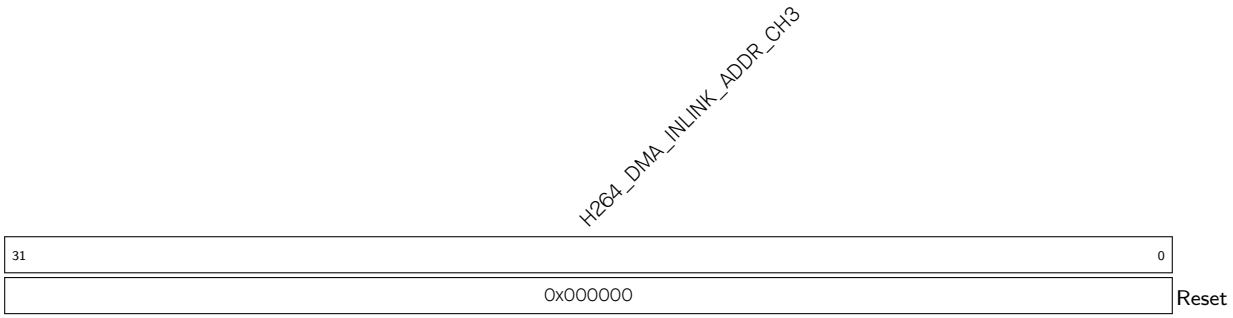
H264_DMA_INLINK_PARK_CH3 表示 inlink 描述符的 FSM 的状态。

0：inlink 描述符的 FSM 正在工作

1：inlink 描述符的 FSM 处于空闲状态

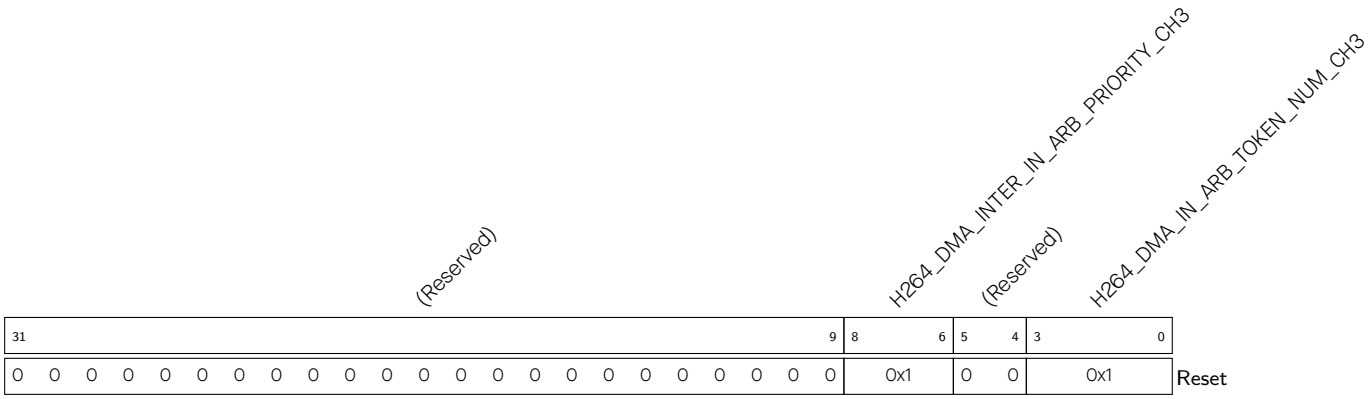
(RO)

Register 41.122. H264_DMA_IN_LINK_ADDR_CH3_REG (0x0820)



H264_DMA_INLINK_ADDR_CH3 配置第一个 inlink 描述符的地址。(R/W)

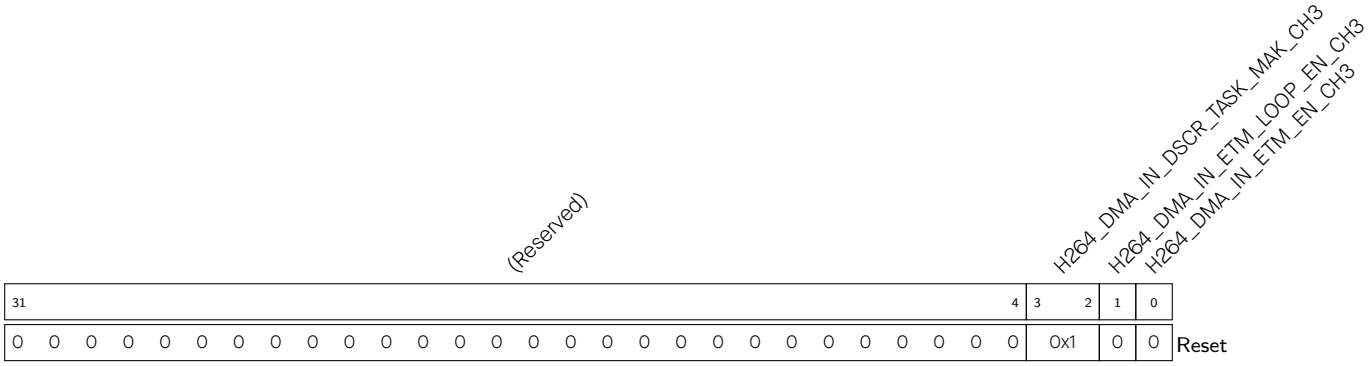
Register 41.123. H264_DMA_IN_ARB_CH3_REG (0x0840)



H264_DMA_IN_ARB_TOKEN_NUM_CH3 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH3 配置 RX 通道 3 的内部存储器访问优先级。(R/W)

Register 41.124. H264_DMA_IN_ETM_CONF_CH3_REG (0x0848)



H264_DMA_IN_ETM_EN_CH3 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ETM_LOOP_EN_CH3 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH3 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.125. H264_DMA_IN_CONF0_CH4_REG (0x0900)

Bits	31–27	26	25	24	23–13	12	11–9	8–6	5	4	3	2	1–0
Field	(Reserved)	H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH4	H264_DMA_IN_CMD_DISABLE_CH4	H264_DMA_IN_RST_CH4	(Reserved)	H264_DMA_IN_PAGE_BOUND_EN_CH4	(Reserved)	H264_DMA_IN_MEM_BURST_LENGTH_CH4	(Reserved)	H264_DMA_IN_CHECK_OWNER_CH4	H264_DMA_IN_ECC_AES_EN_CH4	H264_DMA_INDSCR_BURST_EN_CH4	(Reserved)
Reset	0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0x0	0	0	0	0	0 0

H264_DMA_INDSCR_BURST_EN_CH4 配置是否启用 RX 通道 4 读取描述符的 INCR 突发传输。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ECC_AES_EN_CH4 配置 DMA 是否可以通过 RX 通道 4 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_IN_CHECK_OWNER_CH4 配置是否启用 RX 通道 4 的 owner 位检查。

0：禁用

1：启用

(R/W)

H264_DMA_IN_MEM_BURST_LENGTH_CH4 配置 RX 通道 4 的突发长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

H264_DMA_IN_PAGE_BOUND_EN_CH4 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界

1：AXI 读取的数据不会跨越地址边界

(R/W)

H264_DMA_IN_RST_CH4 写入 1 然后写入 0 以重置 RX 通道 4。(R/W)

见下页...

Register 41.125. H264_DMA_IN_CONFO_CH4_REG (0x0900)

接上页...

H264_DMA_IN_CMD_DISABLE_CH4 配置是否禁用 RX 通道 4 上的命令。

0：禁用

1：启用

(R/W)

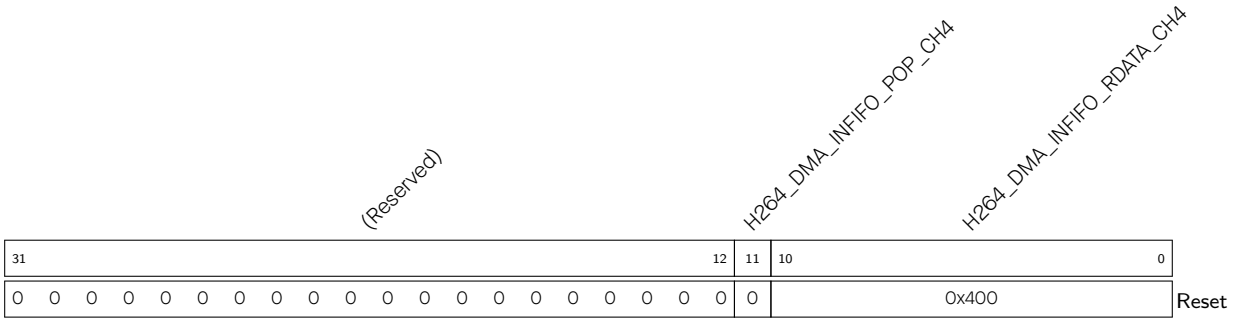
H264_DMA_IN_ARB_WEIGHT_OPT_DIS_CH4 配置是否禁用 RX 通道 4 的权重仲裁。

0：启用

1：禁用

(R/W)

Register 41.126. H264_DMA_IN_POP_CH4_REG (0x0918)



H264_DMA_INFIFO_RDATA_CH4 表示从 DMA RX FIFO 弹出的数据。(RO)

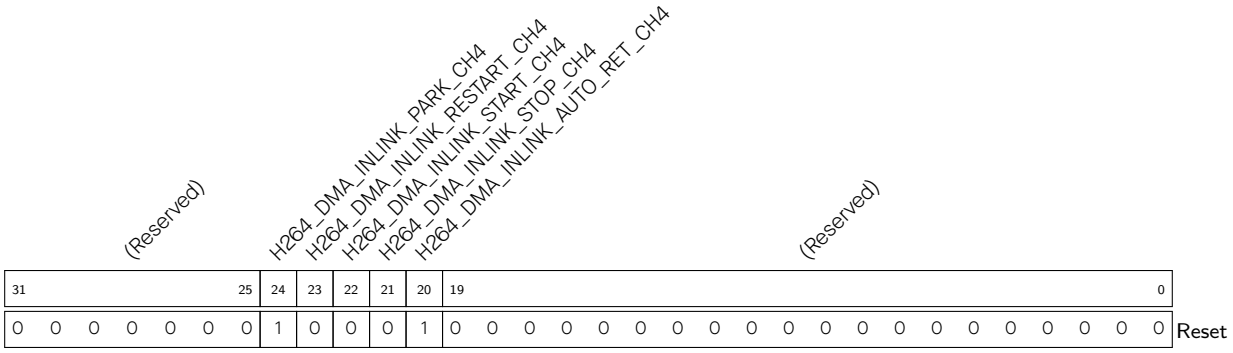
H264_DMA_INFIFO_POP_CH4 配置是否从 DMA RX FIFO 弹出数据。

0：无效

1：弹出

(R/W/SC)

Register 41.127. H264_DMA_IN_LINK_CONF_CH4_REG (0x091C)



H264_DMA_INLINK_AUTO_RET_CH4 配置在数据接收过程中出现错误时是否返回到当前接收描述符的地址。

0：不返回

1：返回

(R/W)

H264_DMA_INLINK_STOP_CH4 配置是否停止处理 inlink 描述符。

0：无效

1：停止

(R/W/SC)

H264_DMA_INLINK_START_CH4 配置是否开始处理 inlink 描述符。

0：无效

1：开始

(R/W/SC)

H264_DMA_INLINK_RESTART_CH4 配置是否从最后一个地址重新开始一个新的 inlink。

0：无效

1：重新开始

(R/W/SC)

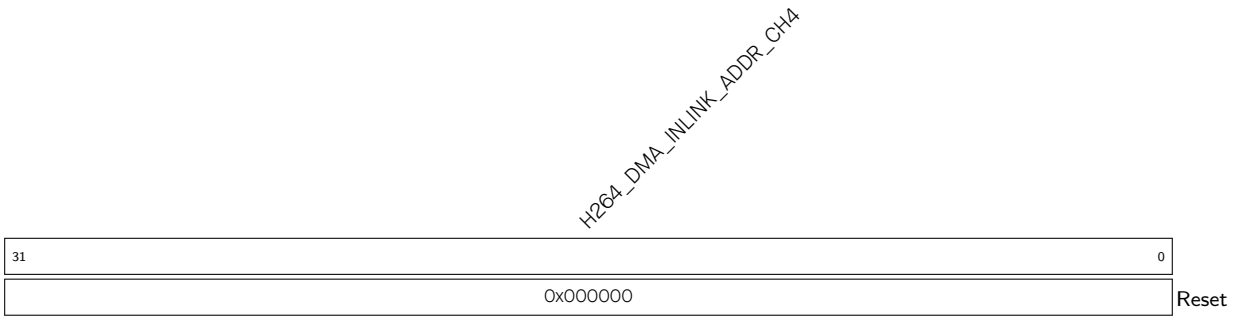
H264_DMA_INLINK_PARK_CH4 表示 inlink 描述符的 FSM 的状态。

0：inlink 描述符的 FSM 正在工作

1：inlink 描述符的 FSM 处于空闲状态

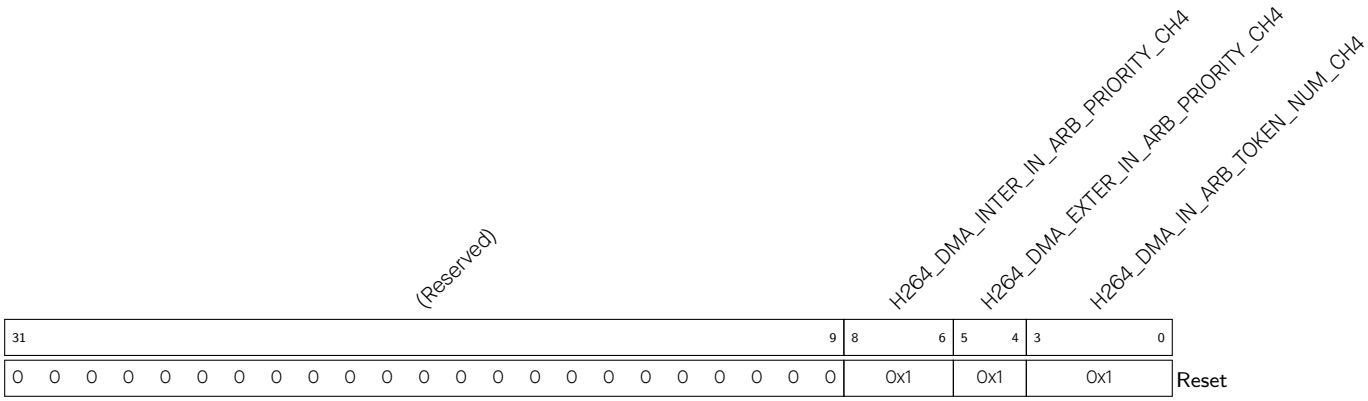
(RO)

Register 41.128. H264_DMA_IN_LINK_ADDR_CH4_REG (0x0920)



H264_DMA_INLINK_ADDR_CH4 配置第一个 inlink 描述符的地址。(R/W)

Register 41.129. H264_DMA_IN_ARB_CH4_REG (0x0940)

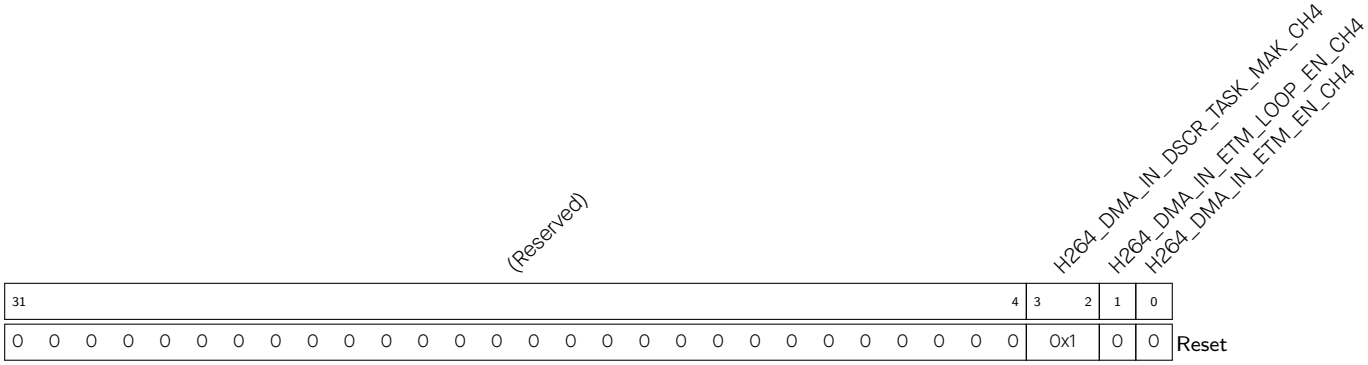


H264_DMA_IN_ARB_TOKEN_NUM_CH4 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_EXTER_IN_ARB_PRIORITY_CH4 配置 RX 通道 4 的外部存储器访问优先级。(R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH4 配置 RX 通道 4 的内部存储器访问优先级。(R/W)

Register 41.130. H264_DMA_IN_ETM_CONF_CH4_REG (0x0948)



H264_DMA_IN_ETM_EN_CH4 配置是否启用 ETM 任务功能。

0：禁用

1：启用

(R/W)

H264_DMA_IN_ETM_LOOP_EN_CH4 配置为 1 时，接收到任务后即可处理描述符。(R/W)

H264_DMA_IN_DSCR_TASK_MAK_CH4 ETM dscr_ready 的最大缓存数量。(R/W)

Register 41.131. H264_DMA_IN_CONF0_CH5_REG (0x0A00)

Bits	31–26	25	24	23–13	12	11–9	8–6	5–4	3	2–0	
Field	(Reserved)	H264_DMA_IN_CMD_DISABLE_CH5	H264_DMA_IN_RST_CH5	(Reserved)	H264_DMA_IN_PAGE_BOUND_EN_CH5	(Reserved)	H264_DMA_IN_MEM_BURST_LENGTH_CH5	(Reserved)	H264_DMA_IN_ECC_AES_EN_CH5	(Reserved)	
Value	0 0 0 0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0x0	0 0	0	0 0 0	Reset

H264_DMA_IN_ECC_AES_EN_CH5 配置 DMA 是否可以通过 RX 通道 5 访问外部存储空间以进行 ECC 和 AES。

0：不访问

1：访问。在这种情况下，存储区域的起始地址应为 16 字节对齐，其宽度乘以一个像素的字节数也应为 16 字节对齐。

(R/W)

H264_DMA_IN_MEM_BURST_LENGTH_CH5 配置 RX 通道 5 的突发长度。

0：8 字节

1：16 字节

2：32 字节

3：64 字节

4：128 字节

5 ~ 7：无效

(R/W)

H264_DMA_IN_PAGE_BOUND_EN_CH5 配置是否确保 AXI 读取的数据不会跨越由 mem_burst_length 定义的地址边界。

0：AXI 读取的数据可以跨越地址边界

1：AXI 读取的数据不会跨越地址边界

(R/W)

H264_DMA_IN_RST_CH5 写入 1 然后写入 0 以重置 RX 通道 5。(R/W)

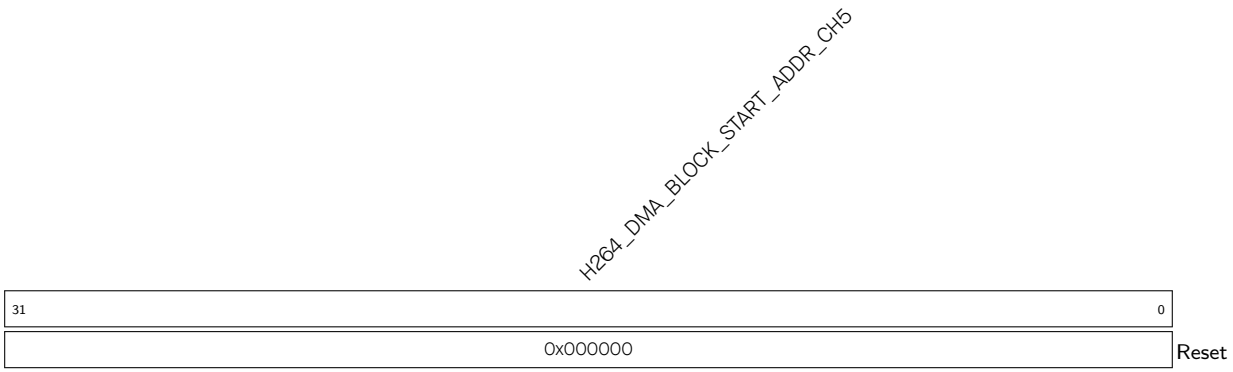
H264_DMA_IN_CMD_DISABLE_CH5 配置是否在 RX 通道 5 上禁用命令。

0：禁用

1：启用

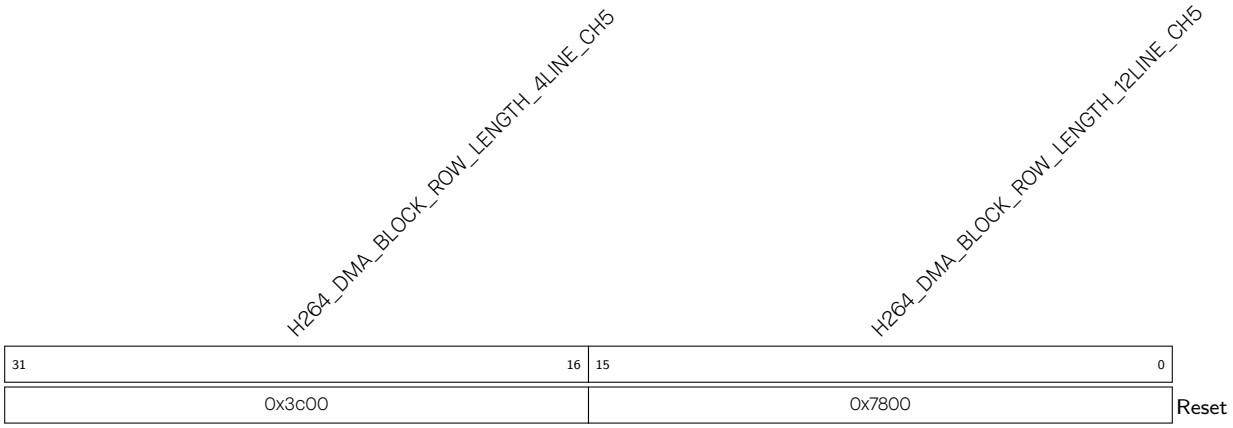
(R/W)

Register 41.132. H264_DMA_IN_CONF1_CH5_REG (0x0A04)



H264_DMA_BLOCK_START_ADDR_CH5 配置 RX 通道 5 的目标起始地址。(R/W)

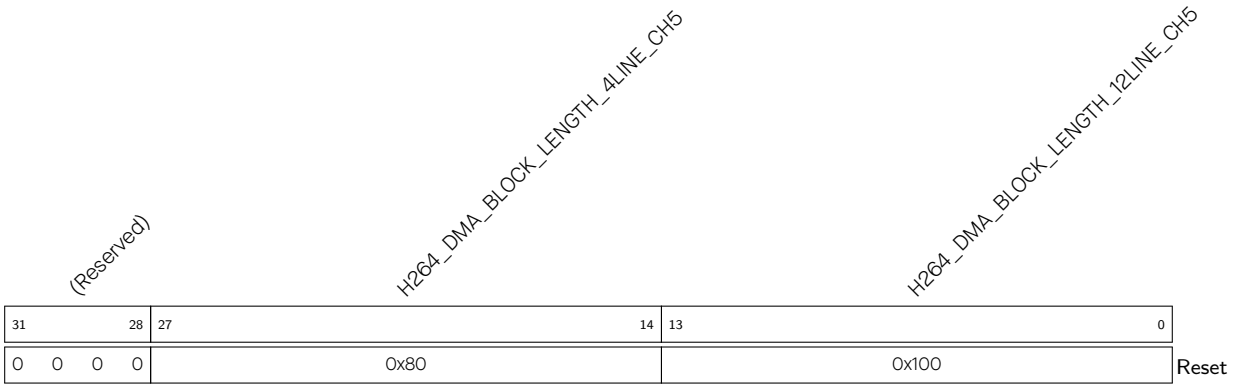
Register 41.133. H264_DMA_IN_CONF2_CH5_REG (0x0A08)



H264_DMA_BLOCK_ROW_LENGTH_12LINE_CH5 配置 12 行块中包含的字节数。(R/W)

H264_DMA_BLOCK_ROW_LENGTH_4LINE_CH5 配置 4 行块中包含的字节数。(R/W)

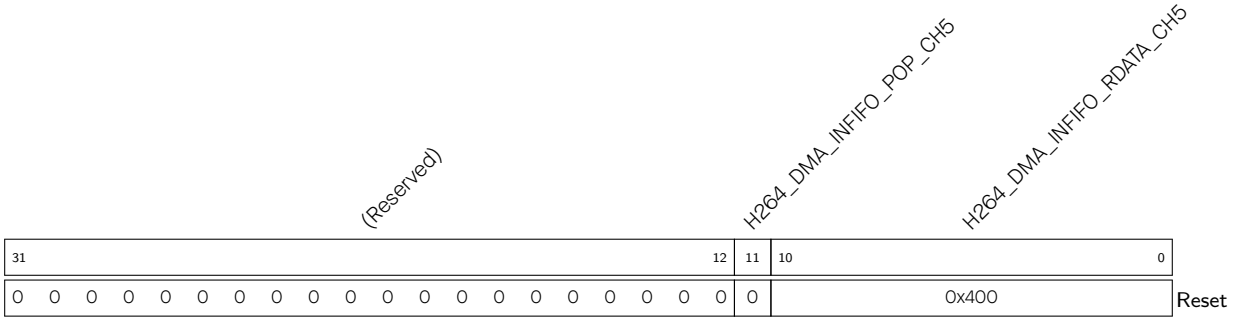
Register 41.134. H264_DMA_IN_CONF3_CH5_REG (0x0A0C)



H264_DMA_BLOCK_LENGTH_12LINE_CH5 配置一个 12 行块中包含的字节数 (R/W)

H264_DMA_BLOCK_LENGTH_4LINE_CH5 配置一个 4 行块中包含的字节数 (R/W)

Register 41.135. H264_DMA_IN_POP_CH5_REG (0x0A24)



H264_DMA_INFIFO_RDATA_CH5 表示从 DMA RX FIFO 弹出的数据。(RO)

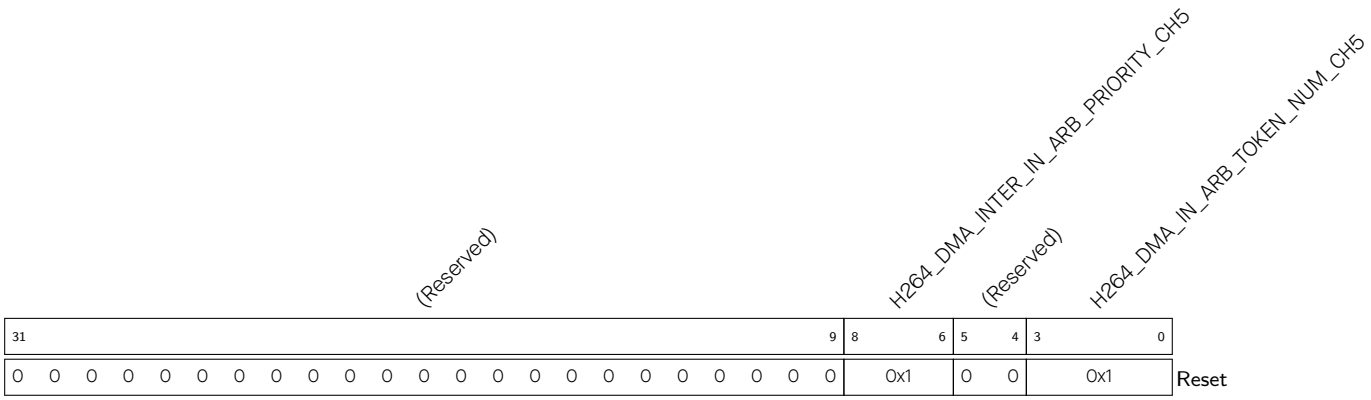
H264_DMA_INFIFO_POP_CH5 配置是否从 DMA RX FIFO 弹出数据。

0：无效

1：弹出

(R/W/SC)

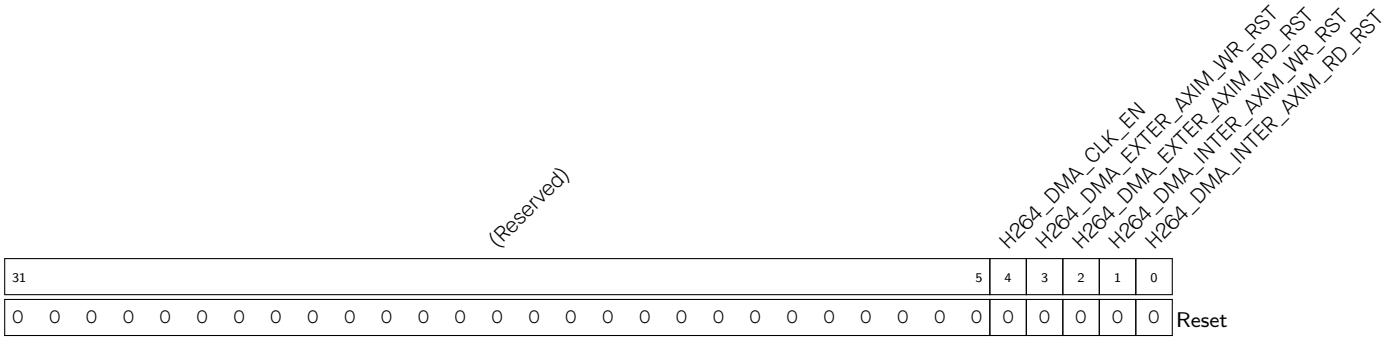
Register 41.136. H264_DMA_IN_ARB_CH5_REG (0x0A40)



H264_DMA_IN_ARB_TOKEN_NUM_CH5 配置仲裁器配置的最大令牌计数。(R/W)

H264_DMA_INTER_IN_ARB_PRIORITY_CH5 配置 RX 通道 5 的内部存储器访问优先级。(R/W)

Register 41.137. H264_DMA_RST_CONF_REG (0x0B08)



H264_DMA_INTER_AXIM_RD_RST 写入 1 然后写入 0 以重置内部 AXI 读取 FSM。(R/W)

H264_DMA_INTER_AXIM_WR_RST 写入 1 然后写入 0 以重置内部 AXI 写入 FSM。(R/W)

H264_DMA_EXTER_AXIM_RD_RST 写入 1 然后写入 0 以重置外部 AXI 读取 FSM。(R/W)

H264_DMA_EXTER_AXIM_WR_RST 写入 1 然后写入 0 以重置外部 AXI 写入 FSM。(R/W)

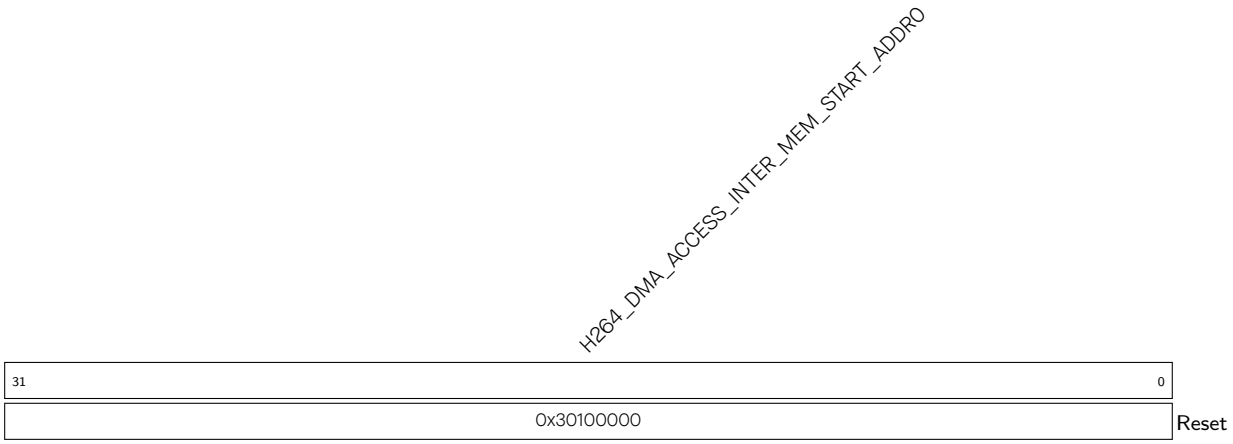
H264_DMA_CLK_EN 配置 DMA 时钟门控。

0：仅在应用程序写入寄存器时启用时钟

1：始终为寄存器启用时钟

(R/W)

Register 41.138. H264_DMA_INTER_MEM_START_ADDR0_REG (0x0B0C)



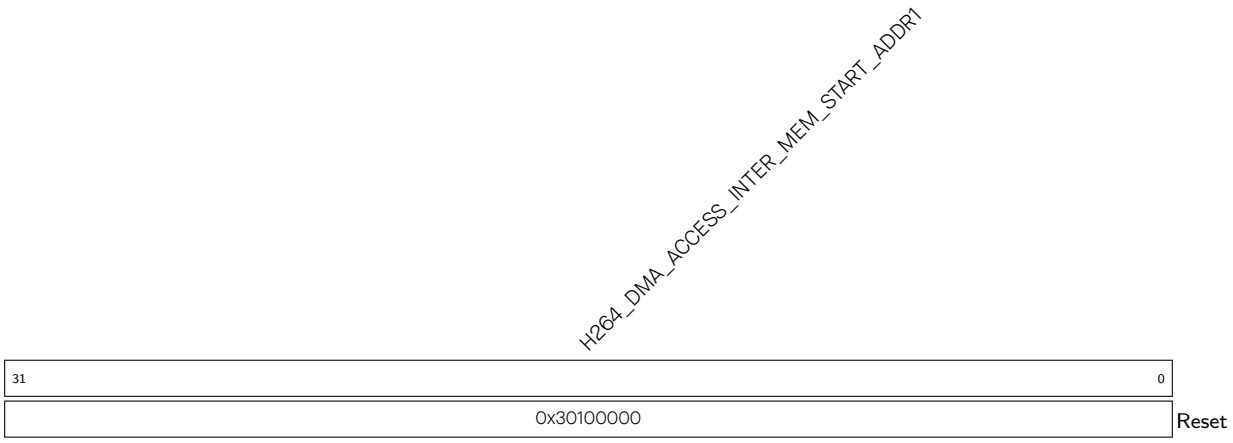
H264_DMA_ACCESS_INTER_MEM_START_ADDR0 配置可访问的内部存储地址 0 空间的起始地址。(R/W)

Register 41.139. H264_DMA_INTER_MEM_END_ADDR0_REG (0x0B10)



H264_DMA_ACCESS_INTER_MEM_END_ADDR0 配置可访问的内部存储地址 0 空间的结束地址。(R/W)

Register 41.140. H264_DMA_INTER_MEM_START_ADDR1_REG (0x0B14)



H264_DMA_ACCESS_INTER_MEM_START_ADDR1 配置可访问的内部存储地址 1 空间的起始地址。(R/W)

Register 41.141. H264_DMA_INTER_MEM_END_ADDR1_REG (0x0B18)



H264_DMA_ACCESS_INTER_MEM_END_ADDR1 配置可访问的内部存储地址 1 空间的结束地址。(R/W)

Register 41.142. H264_DMA_EXTER_MEM_START_ADDR0_REG (0x0B20)



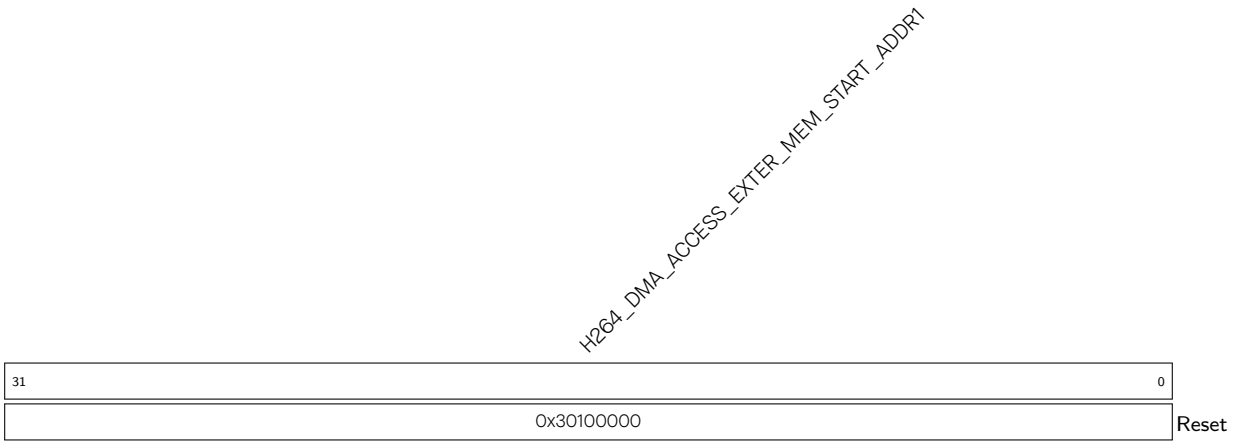
H264_DMA_ACCESS_EXTER_MEM_START_ADDR0 配置可访问的外部存储地址 0 空间的起始地址。(R/W)

Register 41.143. H264_DMA_EXTER_MEM_END_ADDR0_REG (0x0B24)



H264_DMA_ACCESS_EXTER_MEM_END_ADDR0 配置可访问的外部存储地址 0 空间的结束地址。(R/W)

Register 41.144. H264_DMA_EXTER_MEM_START_ADDR1_REG (0x0B28)



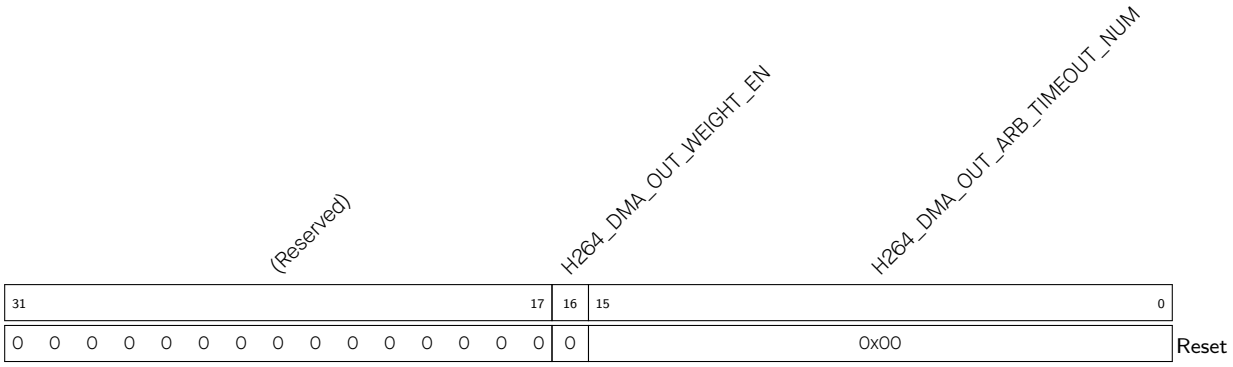
H264_DMA_ACCESS_EXTER_MEM_START_ADDR1 配置可访问的外部存储地址 1 空间的起始地址。(R/W)

Register 41.145. H264_DMA_EXTER_MEM_END_ADDR1_REG (0x0B2C)



H264_DMA_ACCESS_EXTER_MEM_END_ADDR1 配置可访问的外部存储地址 1 空间的结束地址。(R/W)

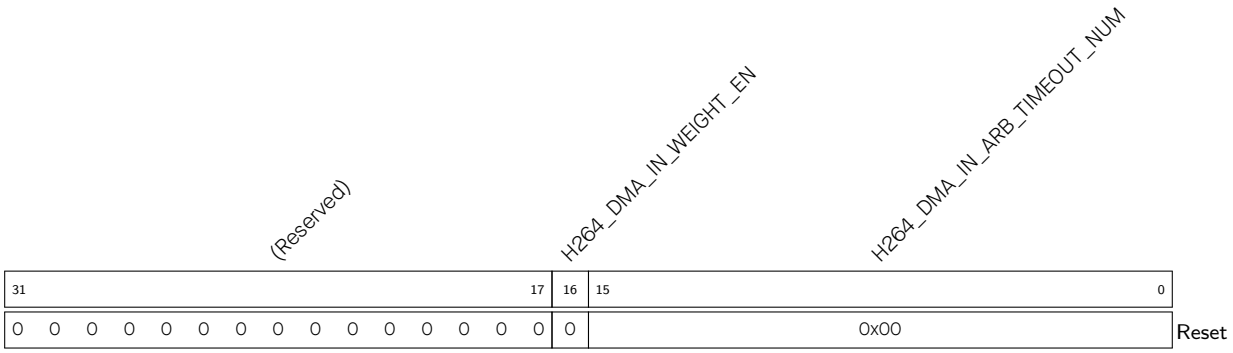
Register 41.146. H264_DMA_OUT_ARB_CONFIG_REG (0x0B30)



H264_DMA_OUT_ARB_TIMEOUT_NUM 配置仲裁器的最大超时计数。(R/W)

H264_DMA_OUT_WEIGHT_EN 配置是否启用 TX 的权重仲裁。(R/W)

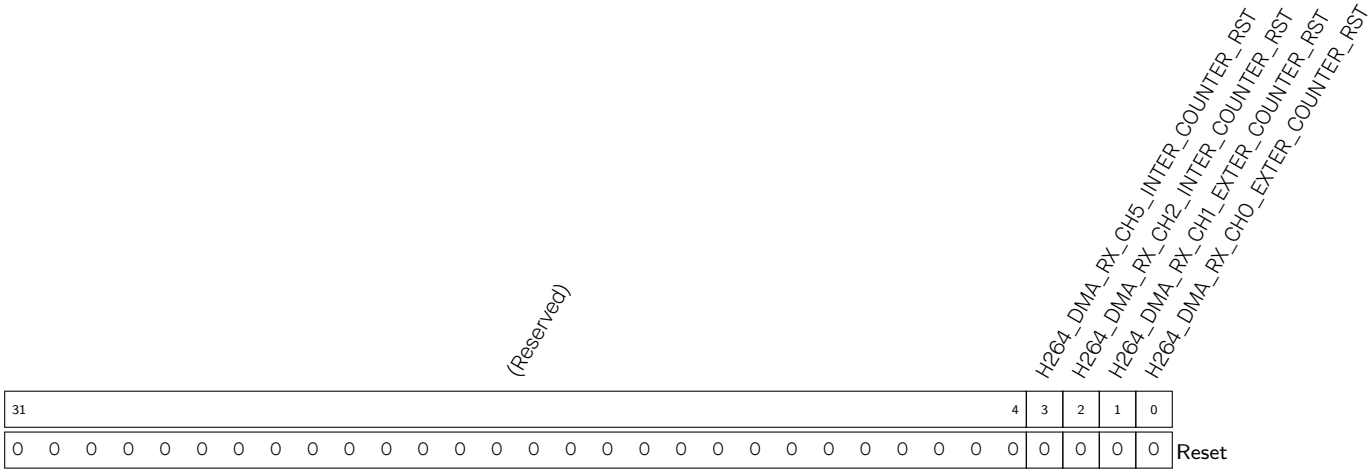
Register 41.147. H264_DMA_IN_ARB_CONFIG_REG (0x0B34)



H264_DMA_IN_ARB_TIMEOUT_NUM 配置仲裁器的最大超时计数。(R/W)

H264_DMA_IN_WEIGHT_EN 配置是否启用 RX 的权重仲裁。(R/W)

Register 41.148. H264_DMA_COUNTER_RST_REG (0x0B50)



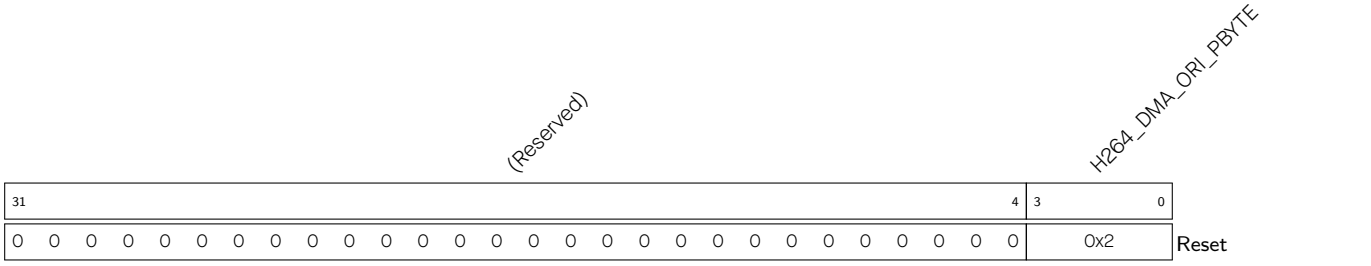
H264_DMA_RX_CH0_EXTER_COUNTER_RST 写入 1 然后写入 0 以重置 RX ch0 计数器。(R/W)

H264_DMA_RX_CH1_EXTER_COUNTER_RST 写入 1 然后写入 0 以重置 RX ch1 计数器。(R/W)

H264_DMA_RX_CH2_INTER_COUNTER_RST 写入 1 然后写入 0 以重置 RX ch2 计数器。(R/W)

H264_DMA_RX_CH5_INTER_COUNTER_RST 写入 1 然后写入 0 以重置 RX ch5 计数器。(R/W)

Register 41.149. H264_DMA_PBYTE_REG (0x0B64)



H264_DMA_ORI_PBYTE 配置原始图像每个像素的字节数。

0：0.5 字节/像素

1：1 字节/像素

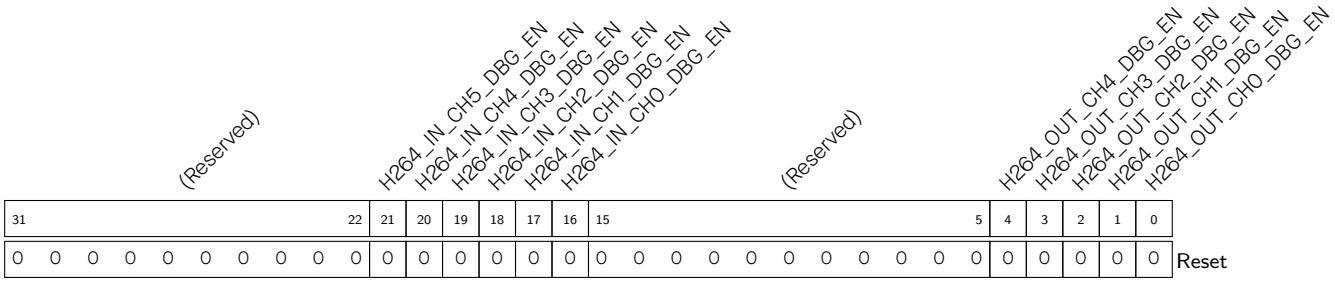
2：1.5 字节/像素

3：2 字节/像素

4：3 字节/像素

(R/W)

Register 41.150. H264_DMA_CH_DBG_EN_REG (0x0B68)



H264_OUT_CH0_DBG_EN 配置是否启用输出通道 0 调试。

0：禁用

1：启用

(R/W)

H264_OUT_CH1_DBG_EN 配置是否启用输出通道 1 调试。

0：禁用

1：启用

(R/W)

H264_OUT_CH2_DBG_EN 配置是否启用输出通道 2 调试。

0：禁用

1：启用

(R/W)

H264_OUT_CH3_DBG_EN 配置是否启用输出通道 3 调试。

0：禁用

1：启用

(R/W)

H264_OUT_CH4_DBG_EN 配置是否启用输出通道 4 调试。

0：禁用

1：启用

(R/W)

H264_IN_CH0_DBG_EN 配置是否启用输入通道 0 调试。

0：禁用

1：启用

(R/W)

H264_IN_CH1_DBG_EN 配置是否启用输入通道 1 调试。

0：禁用

1：启用

(R/W)

H264_IN_CH2_DBG_EN 配置是否启用输入通道 2 调试。

0：禁用

1：启用

(R/W)

H264_IN_CH3_DBG_EN 配置是否启用输入通道 3 调试。

0：禁用

1：启用

(R/W)

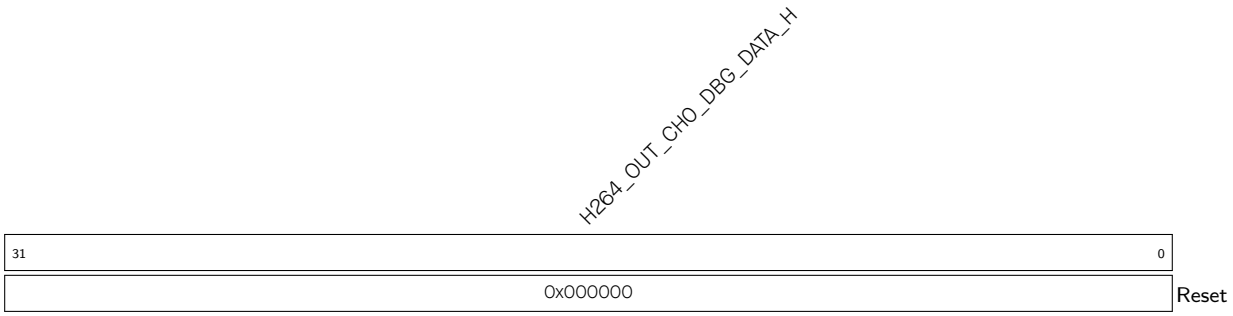
H264_IN_CH4_DBG_EN 配置是否启用输入通道 4 调试。

Register 41.151. H264_DMA_OUT_CH0_DBG_DATA_L_REG (0x0B6C)



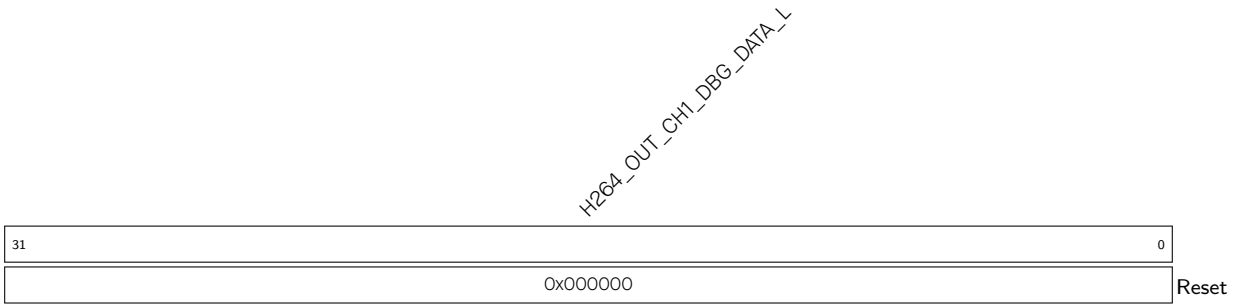
H264_OUT_CH0_DBG_DATA_L 配置输出通道 0 调试数据的第 31-0 位。(R/W)

Register 41.152. H264_DMA_OUT_CH0_DBG_DATA_H_REG (0x0B70)



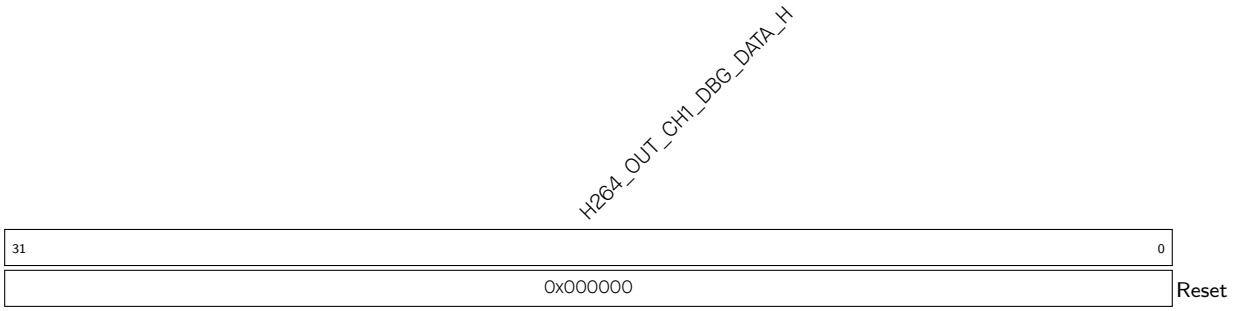
H264_OUT_CH0_DBG_DATA_H 配置输出通道 0 调试数据的第 63-32 位。(R/W)

Register 41.153. H264_DMA_OUT_CH1_DBG_DATA_L_REG (0x0B74)



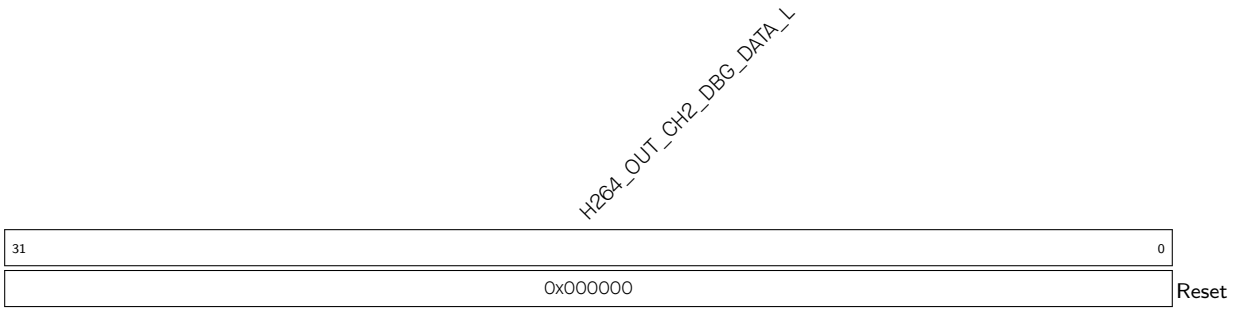
H264_OUT_CH1_DBG_DATA_L 配置输出通道 1 调试数据的第 31-0 位。(R/W)

Register 41.154. H264_DMA_OUT_CH1_DBG_DATA_H_REG (0x0B78)



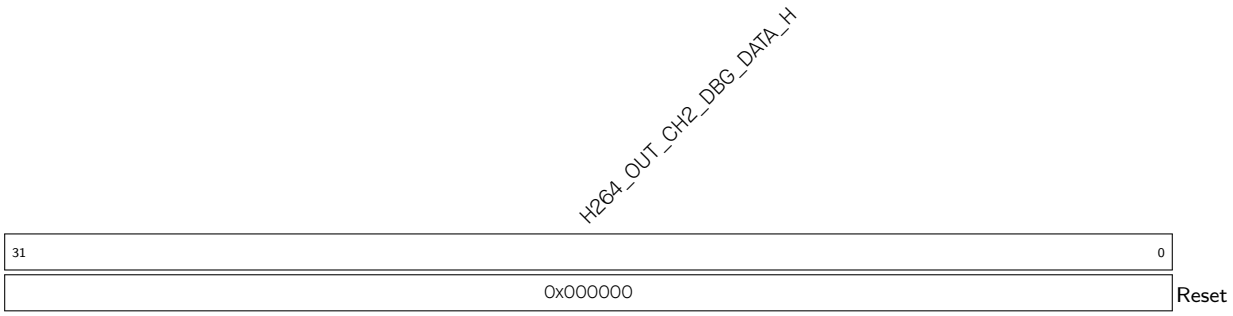
H264_OUT_CH1_DBG_DATA_H 配置输出通道 1 调试数据的第 63-32 位。(R/W)

Register 41.155. H264_DMA_OUT_CH2_DBG_DATA_L_REG (0x0B7C)



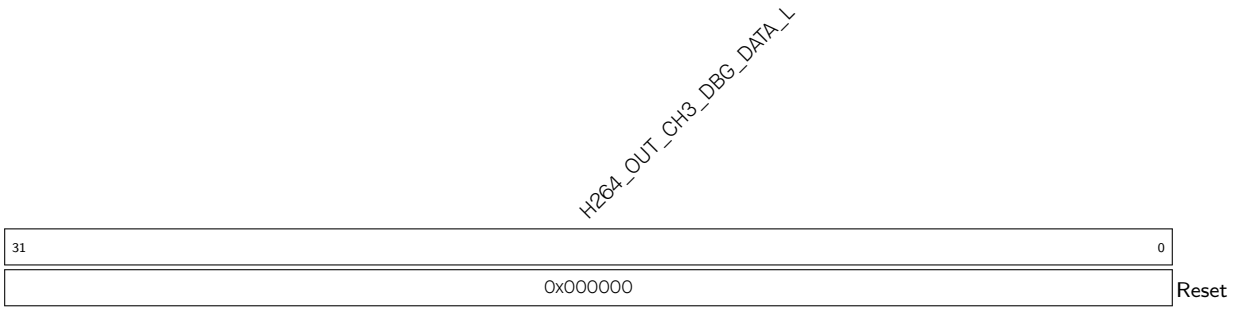
H264_OUT_CH2_DBG_DATA_L 配置输出通道 2 调试数据的第 31-0 位。(R/W)

Register 41.156. H264_DMA_OUT_CH2_DBG_DATA_H_REG (0x0B80)



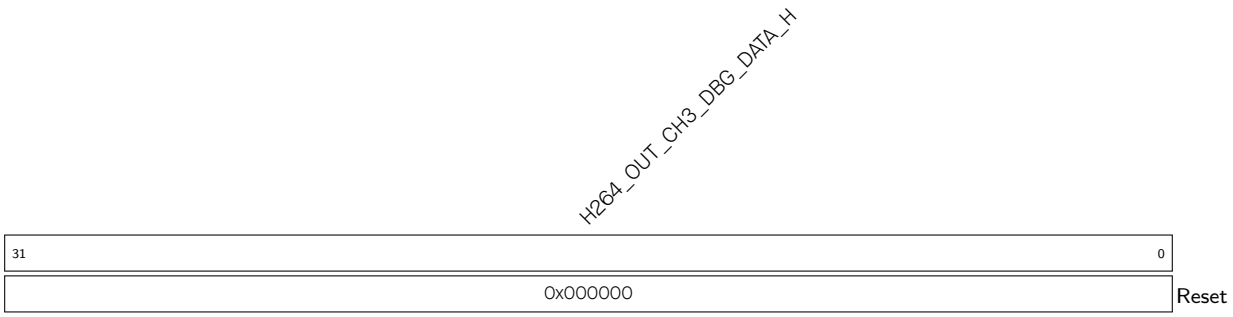
H264_OUT_CH2_DBG_DATA_H 配置输出通道 2 调试数据的第 63-32 位。(R/W)

Register 41.157. H264_DMA_OUT_CH3_DBG_DATA_L_REG (0x0B84)



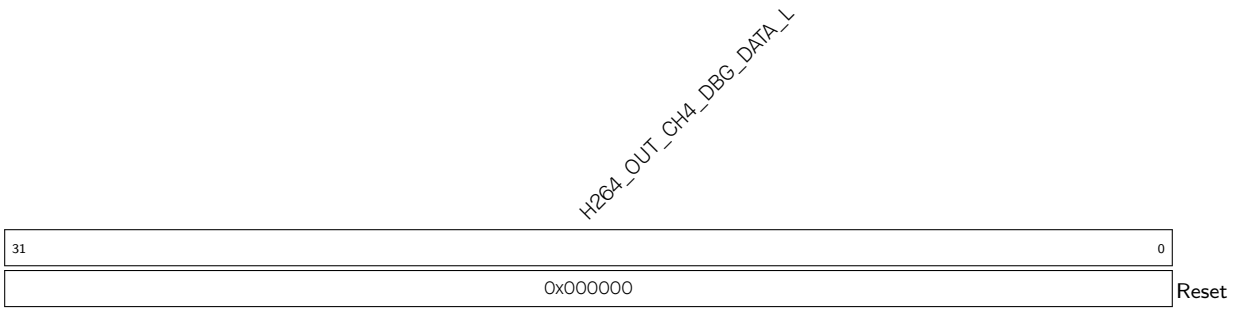
H264_OUT_CH3_DBG_DATA_L 配置输出通道 3 调试数据的第 31-0 位。(R/W)

Register 41.158. H264_DMA_OUT_CH3_DBG_DATA_H_REG (0x0B88)



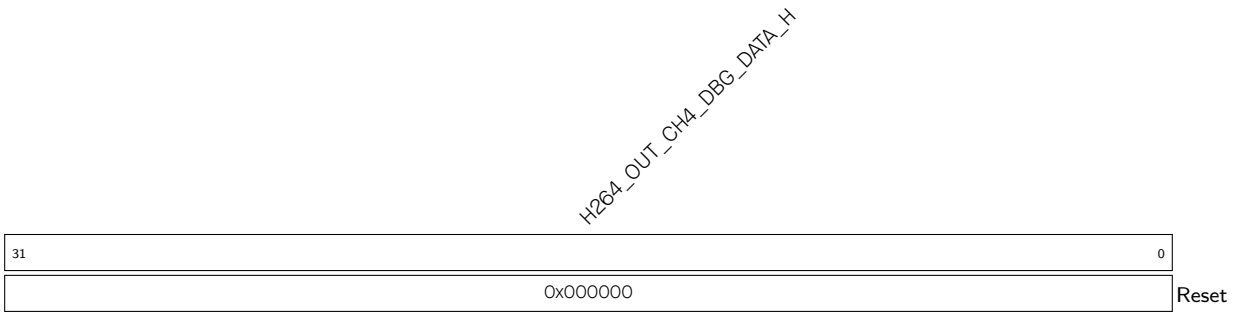
H264_OUT_CH3_DBG_DATA_H 配置输出通道 3 调试数据的第 63-32 位。(R/W)

Register 41.159. H264_DMA_OUT_CH4_DBG_DATA_L_REG (0x0B8C)



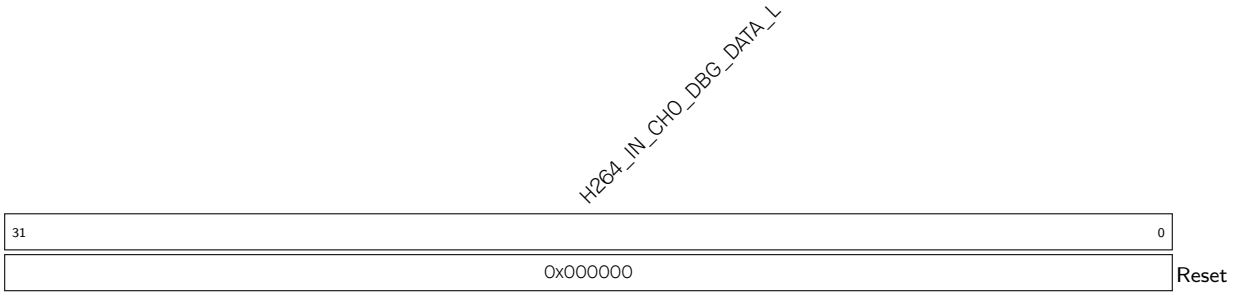
H264_OUT_CH4_DBG_DATA_L 配置输出通道 4 调试数据的第 31-0 位。(R/W)

Register 41.160. H264_DMA_OUT_CH4_DBG_DATA_H_REG (0x0B90)



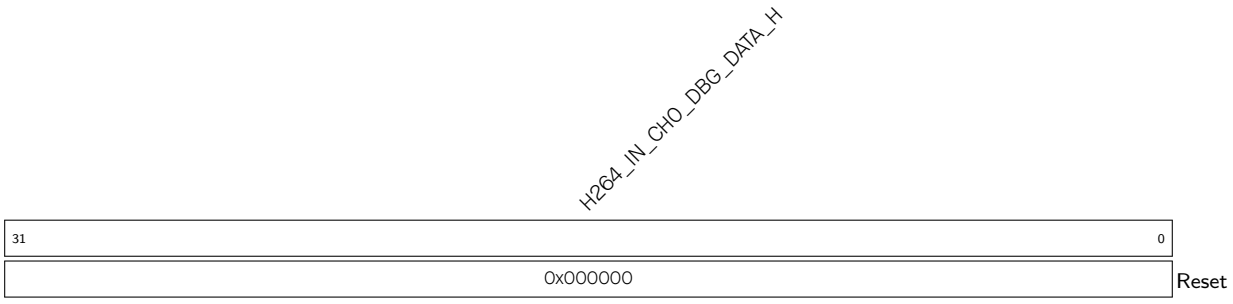
H264_OUT_CH4_DBG_DATA_H 配置输出通道 4 调试数据的第 63-32 位。(R/W)

Register 41.161. H264_DMA_IN_CH0_DBG_DATA_L_REG (0x0B94)



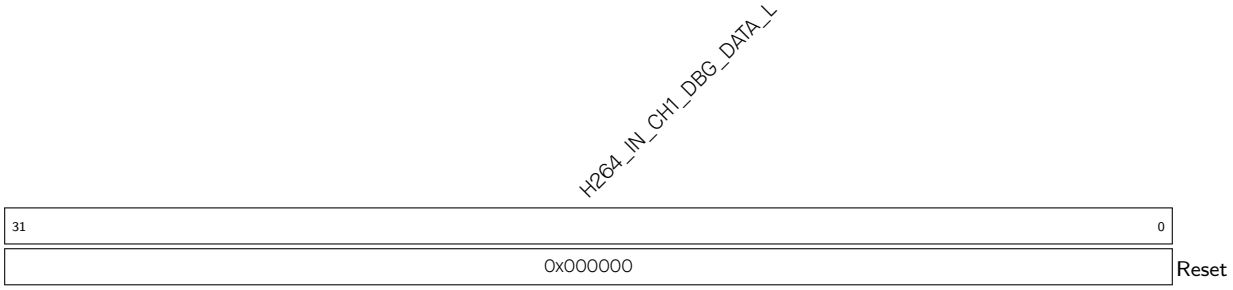
H264_IN_CH0_DBG_DATA_L 配置输入通道 0 调试数据的第 31-0 位。(R/W)

Register 41.162. H264_DMA_IN_CH0_DBG_DATA_H_REG (0x0B98)



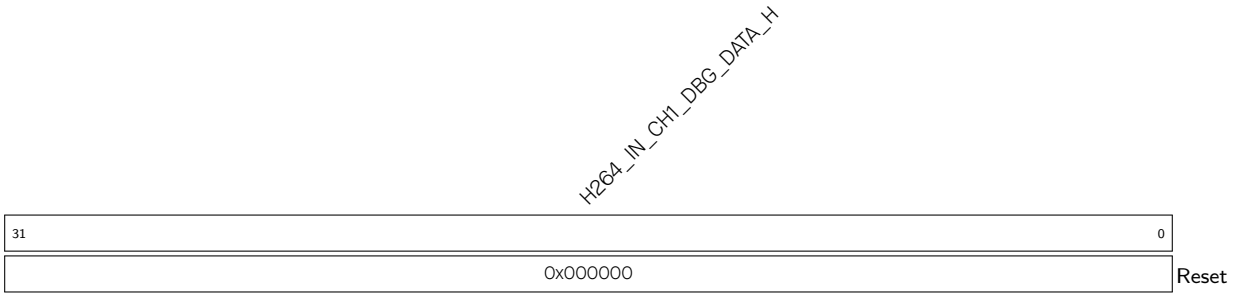
H264_IN_CH0_DBG_DATA_H 配置输入通道 0 调试数据的第 63-32 位。(R/W)

Register 41.163. H264_DMA_IN_CH1_DBG_DATA_L_REG (0x0B9C)



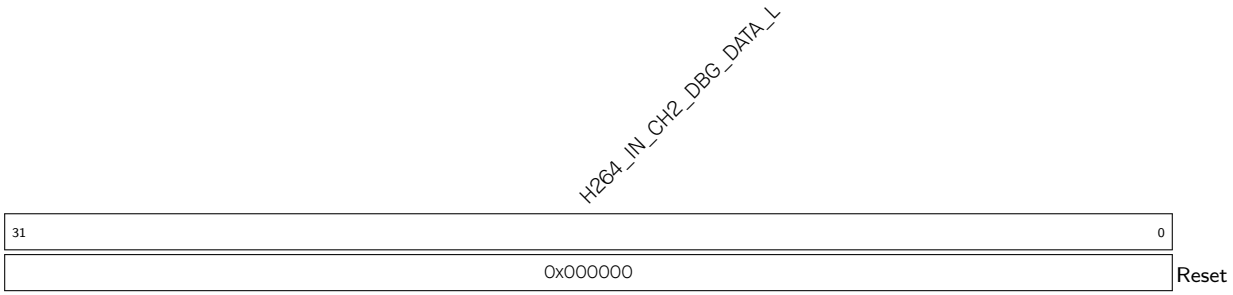
H264_IN_CH1_DBG_DATA_L 配置输入通道 1 调试数据的第 31-0 位。(R/W)

Register 41.164. H264_DMA_IN_CH1_DBG_DATA_H_REG (0x0BA0)



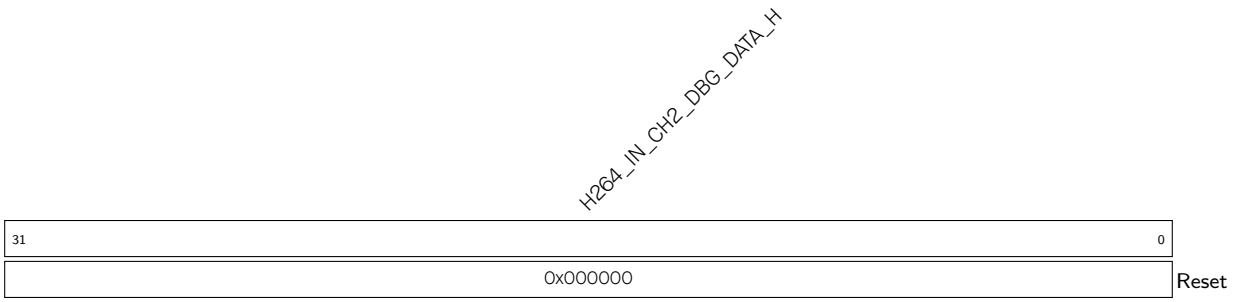
H264_IN_CH1_DBG_DATA_H 配置输入通道 1 调试数据的第 63-32 位。(R/W)

Register 41.165. H264_DMA_IN_CH2_DBG_DATA_L_REG (0x0BA4)



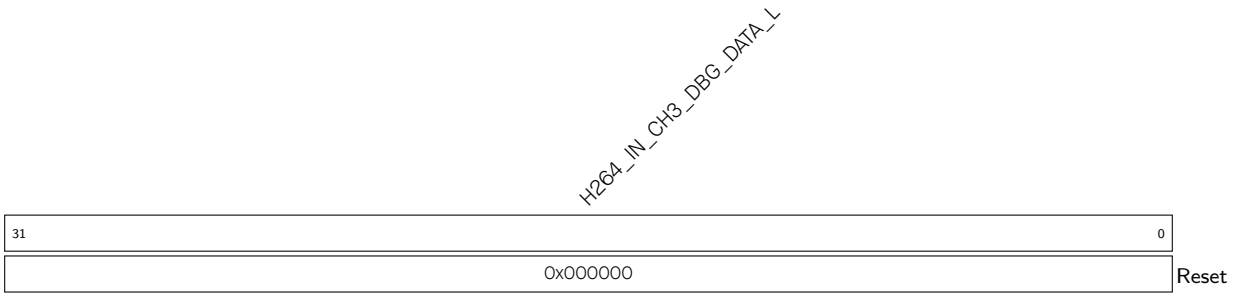
H264_IN_CH2_DBG_DATA_L 配置输入通道 2 调试数据的第 31-0 位。(R/W)

Register 41.166. H264_DMA_IN_CH2_DBG_DATA_H_REG (0x0BA8)



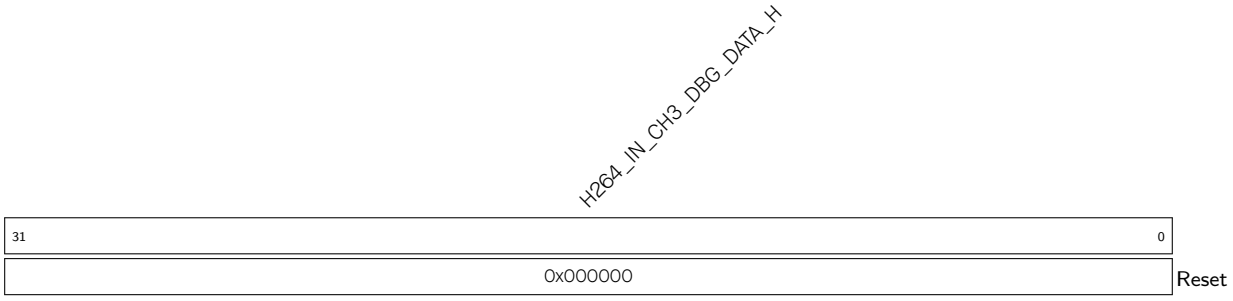
H264_IN_CH2_DBG_DATA_H 配置输入通道 2 调试数据的第 63-32 位。(R/W)

Register 41.167. H264_DMA_IN_CH3_DBG_DATA_L_REG (0x0BAC)



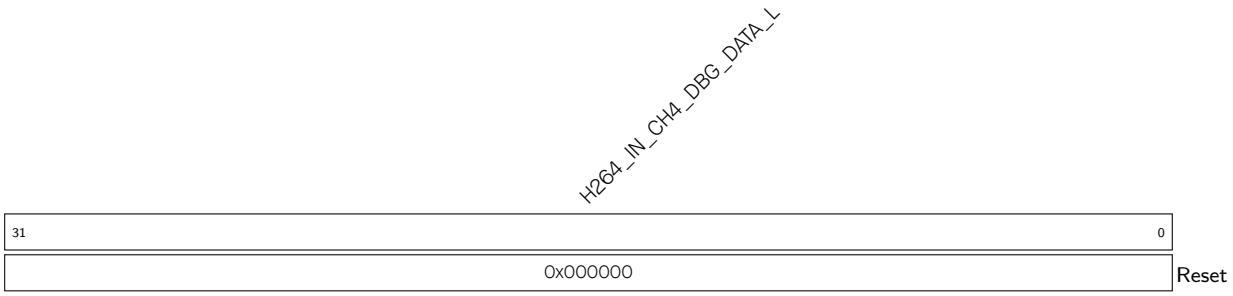
H264_IN_CH3_DBG_DATA_L 配置输入通道 3 调试数据的第 31-0 位。(R/W)

Register 41.168. H264_DMA_IN_CH3_DBG_DATA_H_REG (0x0BB0)



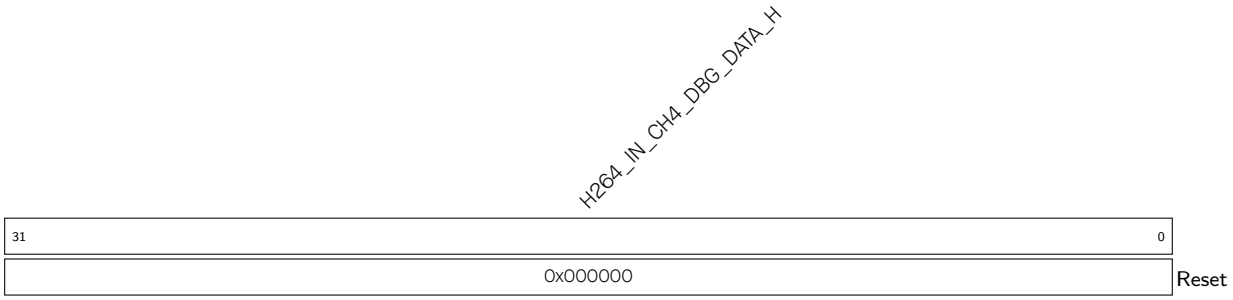
H264_IN_CH3_DBG_DATA_H 配置输入通道 3 调试数据的第 63-32 位。(R/W)

Register 41.169. H264_DMA_IN_CH4_DBG_DATA_L_REG (0x0BB4)



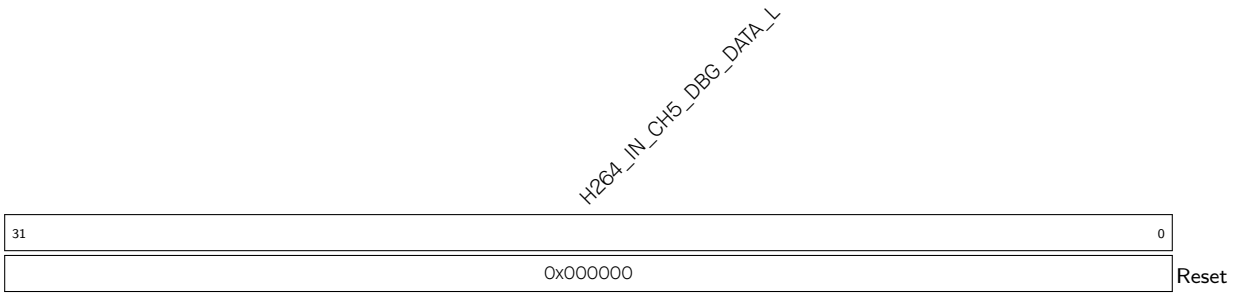
H264_IN_CH4_DBG_DATA_L 配置输入通道 4 调试数据的第 31-0 位。(R/W)

Register 41.170. H264_DMA_IN_CH4_DBG_DATA_H_REG (0x0BB8)



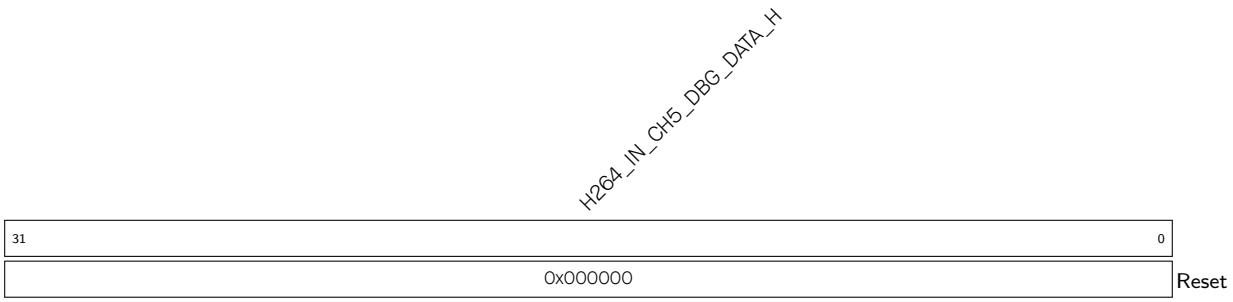
H264_IN_CH4_DBG_DATA_H 配置输入通道 4 调试数据的第 63-32 位。(R/W)

Register 41.171. H264_DMA_IN_CH5_DBG_DATA_L_REG (0x0BBC)



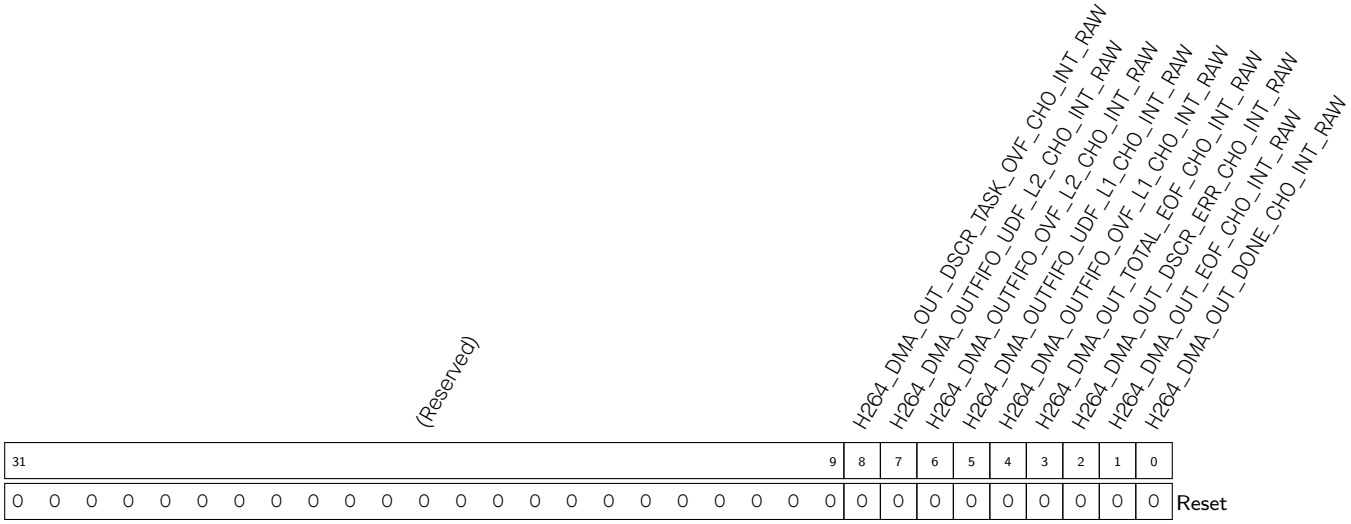
H264_IN_CH5_DBG_DATA_L 配置输入通道 5 调试数据的第 31-0 位。(R/W)

Register 41.172. H264_DMA_IN_CH5_DBG_DATA_H_REG (0x0BC0)



H264_IN_CH5_DBG_DATA_H 配置输入通道 5 调试数据的第 63-32 位。(R/W)

Register 41.173. H264_DMA_OUT_INT_RAW_CH0_REG (0x0004)



H264_DMA_OUT_DONE_CH0_INT_RAW H264_DMA_OUT_DONE_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_EOF_CH0_INT_RAW H264_DMA_OUT_EOF_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH0_INT_RAW H264_DMA_OUT_DSCR_ERR_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_RAW H264_DMA_OUT_TOTAL_EOF_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_RAW H264_DMA_OUTFIFO_OVF_L1_CH0_INT 的原始中断状态。(R/WTC/SS)

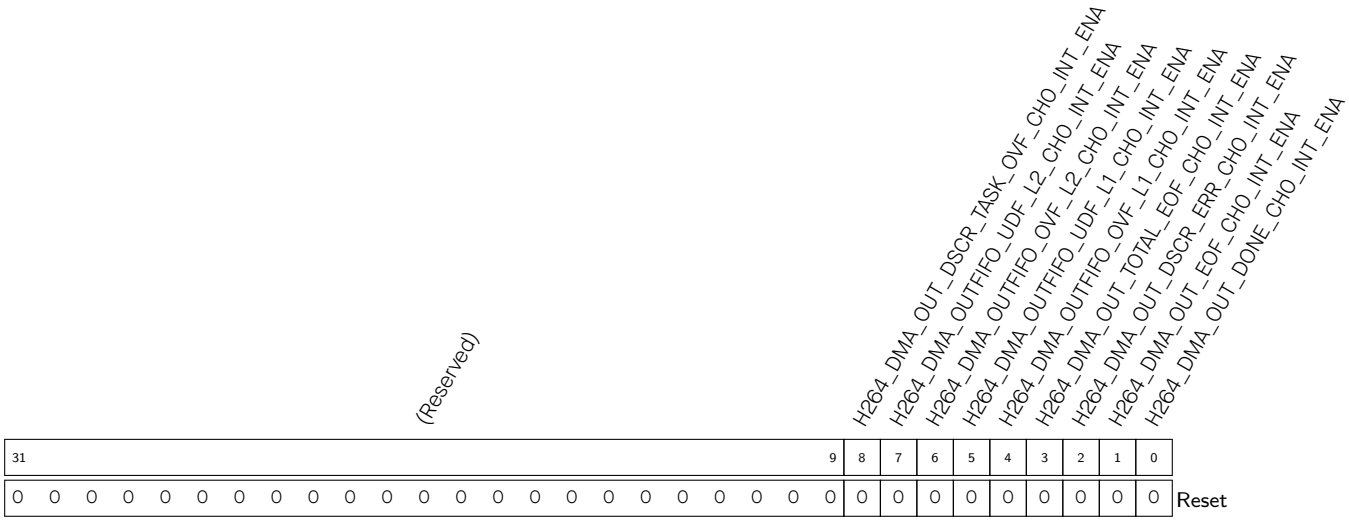
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_RAW H264_DMA_OUTFIFO_UDF_L1_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_RAW H264_DMA_OUTFIFO_OVF_L2_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_RAW H264_DMA_OUTFIFO_UDF_L2_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_RAW H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT 的原始中断状态。(R/WTC/SS)

Register 41.174. H264_DMA_OUT_INT_ENA_CH0_REG (0x0008)



H264_DMA_OUT_DONE_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUT_DONE_CH0_INT。(R/W)

H264_DMA_OUT_EOF_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUT_EOF_CH0_INT。(R/W)

H264_DMA_OUT_DSCR_ERR_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_ERR_CH0_INT。(R/W)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUT_TOTAL_EOF_CH0_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L1_CH0_INT。(R/W)

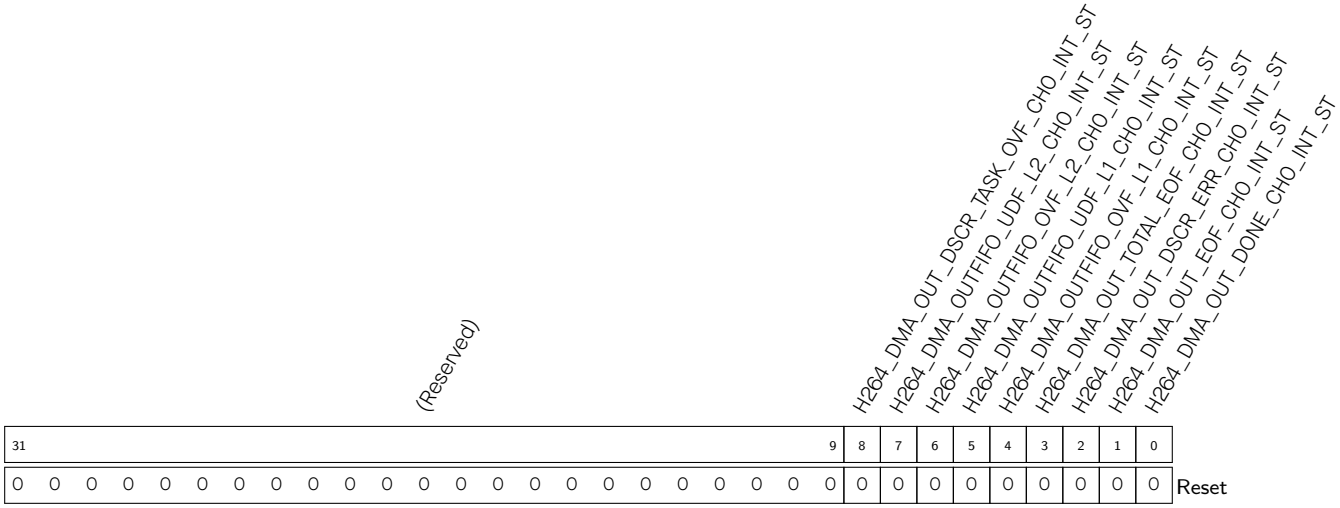
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L1_CH0_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L2_CH0_INT。(R/W)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L2_CH0_INT。(R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT。(R/W)

Register 41.175. H264_DMA_OUT_INT_ST_CH0_REG (0x000C)



H264_DMA_OUT_DONE_CH0_INT_ST H264_DMA_OUT_DONE_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_EOF_CH0_INT_ST H264_DMA_OUT_EOF_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_ERR_CH0_INT_ST H264_DMA_OUT_DSCR_ERR_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_ST H264_DMA_OUT_TOTAL_EOF_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_ST H264_DMA_OUTFIFO_OVF_L1_CH0_INT 的屏蔽中断状态。(RO)

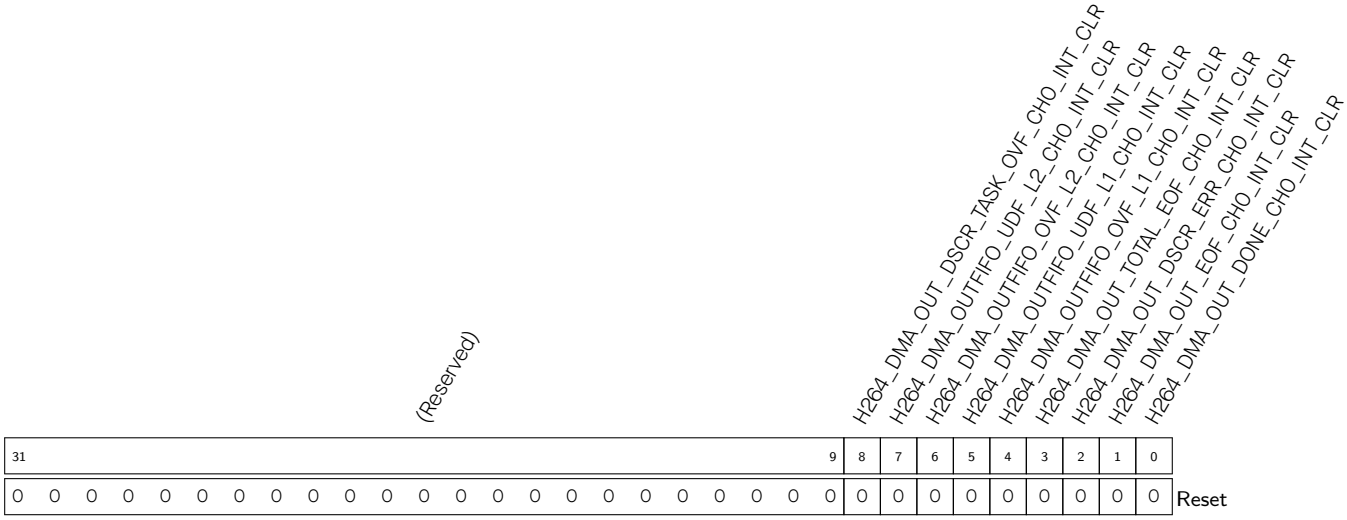
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_ST H264_DMA_OUTFIFO_UDF_L1_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_ST H264_DMA_OUTFIFO_OVF_L2_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_ST H264_DMA_OUTFIFO_UDF_L2_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_ST H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT 的屏蔽中断状态。(RO)

Register 41.176. H264_DMA_OUT_INT_CLR_CH0_REG (0x0010)



H264_DMA_OUT_DONE_CH0_INT_CLR 写 1 清除 H264_DMA_OUT_DONE_CH0_INT。(WT)

H264_DMA_OUT_EOF_CH0_INT_CLR 写 1 清除 H264_DMA_OUT_EOF_CH0_INT。(WT)

H264_DMA_OUT_DSCR_ERR_CH0_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_ERR_CH0_INT。(WT)

H264_DMA_OUT_TOTAL_EOF_CH0_INT_CLR 写 1 清除 H264_DMA_OUT_TOTAL_EOF_CH0_INT。(WT)

H264_DMA_OUTFIFO_OVF_L1_CH0_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L1_CH0_INT。(WT)

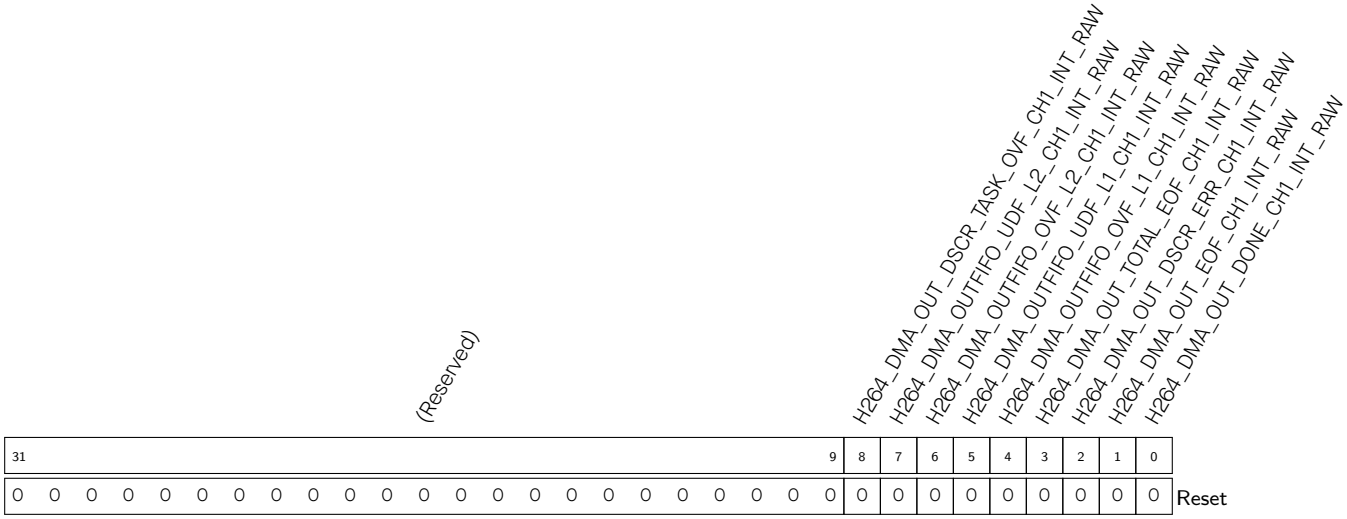
H264_DMA_OUTFIFO_UDF_L1_CH0_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L1_CH0_INT。(WT)

H264_DMA_OUTFIFO_OVF_L2_CH0_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L2_CH0_INT。(WT)

H264_DMA_OUTFIFO_UDF_L2_CH0_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L2_CH0_INT。(WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_TASK_OVF_CH0_INT。(WT)

Register 41.177. H264_DMA_OUT_INT_RAW_CH1_REG (0x0104)



H264_DMA_OUT_DONE_CH1_INT_RAW H264_DMA_OUT_DONE_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_EOF_CH1_INT_RAW H264_DMA_OUT_EOF_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH1_INT_RAW H264_DMA_OUT_DSCR_ERR_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_RAW H264_DMA_OUT_TOTAL_EOF_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_RAW H264_DMA_OUTFIFO_OVF_L1_CH1_INT 的原始中断状态。(R/WTC/SS)

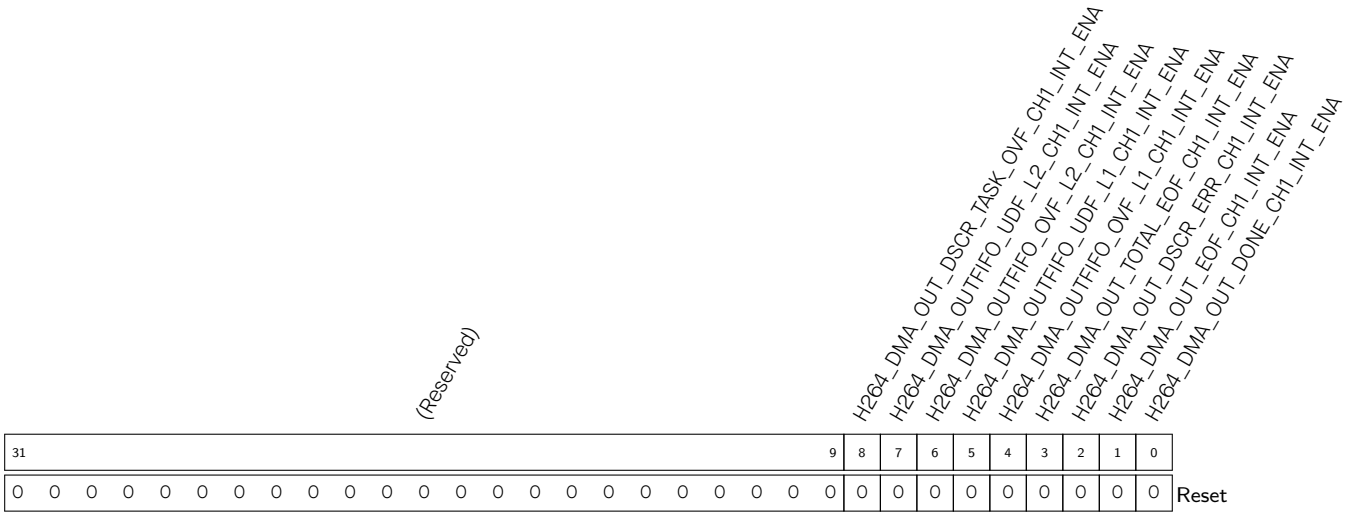
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_RAW H264_DMA_OUTFIFO_UDF_L1_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_RAW H264_DMA_OUTFIFO_OVF_L2_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_RAW H264_DMA_OUTFIFO_UDF_L2_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_RAW H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT 的原始中断状态。(R/WTC/SS)

Register 41.178. H264_DMA_OUT_INT_ENA_CH1_REG (0x0108)



H264_DMA_OUT_DONE_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUT_DONE_CH1_INT。(R/W)

H264_DMA_OUT_EOF_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUT_EOF_CH1_INT。(R/W)

H264_DMA_OUT_DSCR_ERR_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_ERR_CH1_INT。(R/W)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUT_TOTAL_EOF_CH1_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L1_CH1_INT。(R/W)

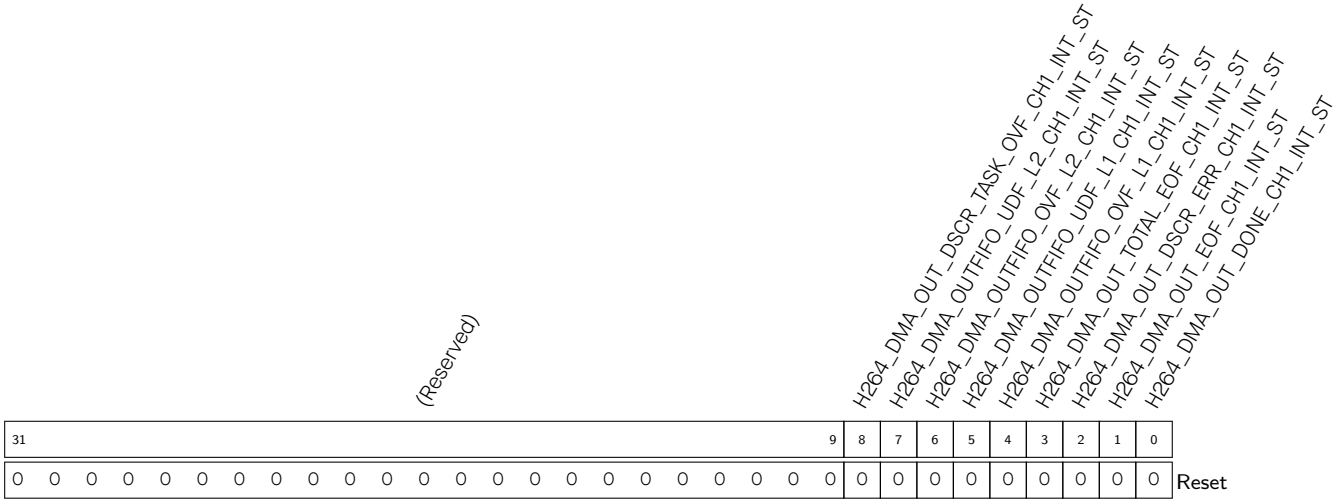
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L1_CH1_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L2_CH1_INT。(R/W)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L2_CH1_INT。(R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT。(R/W)

Register 41.179. H264_DMA_OUT_INT_ST_CH1_REG (0x010C)



H264_DMA_OUT_DONE_CH1_INT_ST H264_DMA_OUT_DONE_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_EOF_CH1_INT_ST H264_DMA_OUT_EOF_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_ERR_CH1_INT_ST H264_DMA_OUT_DSCR_ERR_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_ST H264_DMA_OUT_TOTAL_EOF_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_ST H264_DMA_OUTFIFO_OVF_L1_CH1_INT 的屏蔽中断状态。(RO)

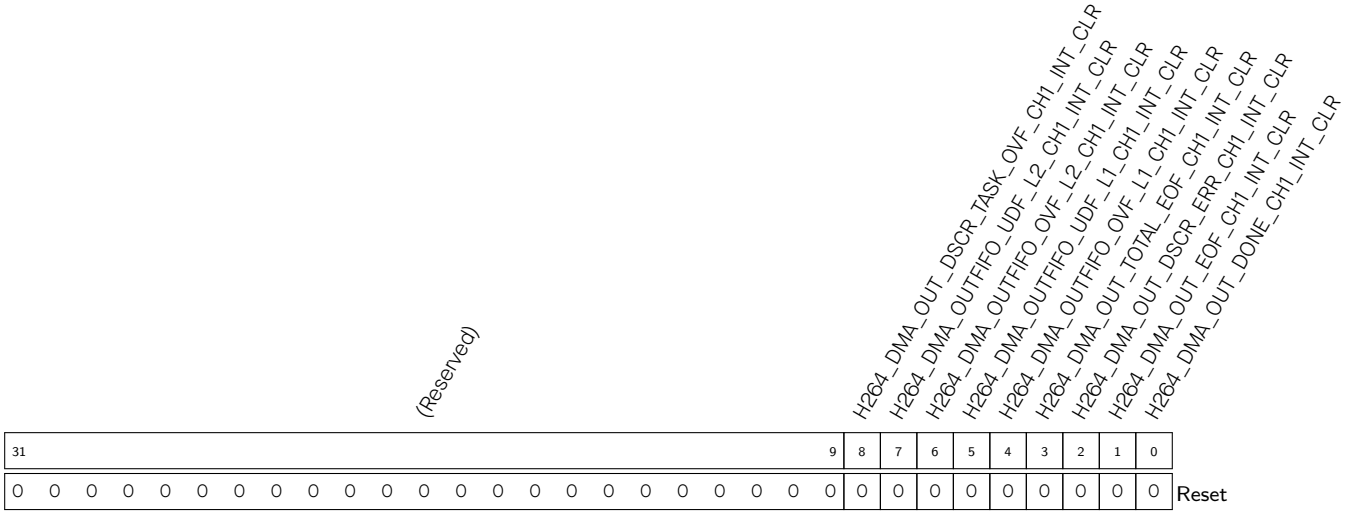
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_ST H264_DMA_OUTFIFO_UDF_L1_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_ST H264_DMA_OUTFIFO_OVF_L2_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_ST H264_DMA_OUTFIFO_UDF_L2_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_ST H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT 的屏蔽中断状态。(RO)

Register 41.180. H264_DMA_OUT_INT_CLR_CH1_REG (0x0110)



H264_DMA_OUT_DONE_CH1_INT_CLR 写 1 清除 H264_DMA_OUT_DONE_CH1_INT。(WT)

H264_DMA_OUT_EOF_CH1_INT_CLR 写 1 清除 H264_DMA_OUT_EOF_CH1_INT。(WT)

H264_DMA_OUT_DSCR_ERR_CH1_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_ERR_CH1_INT。(WT)

H264_DMA_OUT_TOTAL_EOF_CH1_INT_CLR 写 1 清除 H264_DMA_OUT_TOTAL_EOF_CH1_INT。(WT)

H264_DMA_OUTFIFO_OVF_L1_CH1_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L1_CH1_INT。(WT)

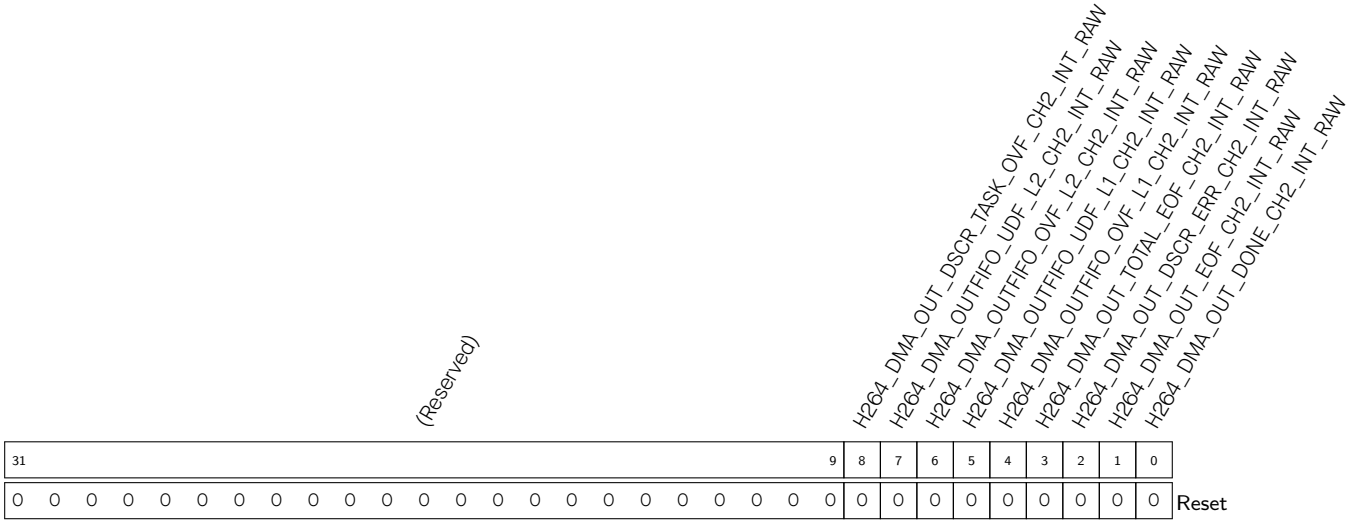
H264_DMA_OUTFIFO_UDF_L1_CH1_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L1_CH1_INT。(WT)

H264_DMA_OUTFIFO_OVF_L2_CH1_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L2_CH1_INT。(WT)

H264_DMA_OUTFIFO_UDF_L2_CH1_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L2_CH1_INT。(WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_TASK_OVF_CH1_INT。(WT)

Register 41.181. H264_DMA_OUT_INT_RAW_CH2_REG (0x0204)



H264_DMA_OUT_DONE_CH2_INT_RAW H264_DMA_OUT_DONE_CH2_INT 的原始中断状态。当所有对应于传输描述符的数据通过 TX 通道 2 发送时触发。(R/WTC/SS)

H264_DMA_OUT_EOF_CH2_INT_RAW H264_DMA_OUT_EOF_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH2_INT_RAW H264_DMA_OUT_DSCR_ERR_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_RAW H264_DMA_OUT_TOTAL_EOF_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_RAW H264_DMA_OUTFIFO_OVF_L1_CH2_INT 的原始中断状态。(R/WTC/SS)

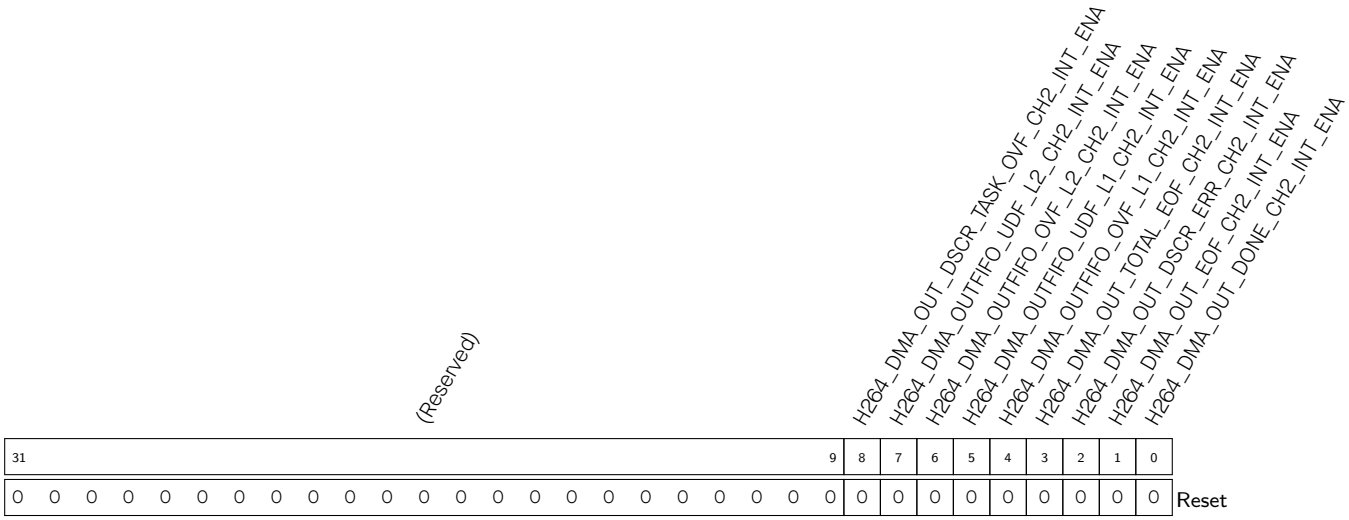
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_RAW H264_DMA_OUTFIFO_UDF_L1_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_RAW H264_DMA_OUTFIFO_OVF_L2_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_RAW H264_DMA_OUTFIFO_UDF_L2_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_RAW H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT 的原始中断状态。(R/WTC/SS)

Register 41.182. H264_DMA_OUT_INT_ENA_CH2_REG (0x0208)



H264_DMA_OUT_DONE_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUT_DONE_CH2_INT。(R/W)

H264_DMA_OUT_EOF_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUT_EOF_CH2_INT。(R/W)

H264_DMA_OUT_DSCR_ERR_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_ERR_CH2_INT。(R/W)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUT_TOTAL_EOF_CH2_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L1_CH2_INT。(R/W)

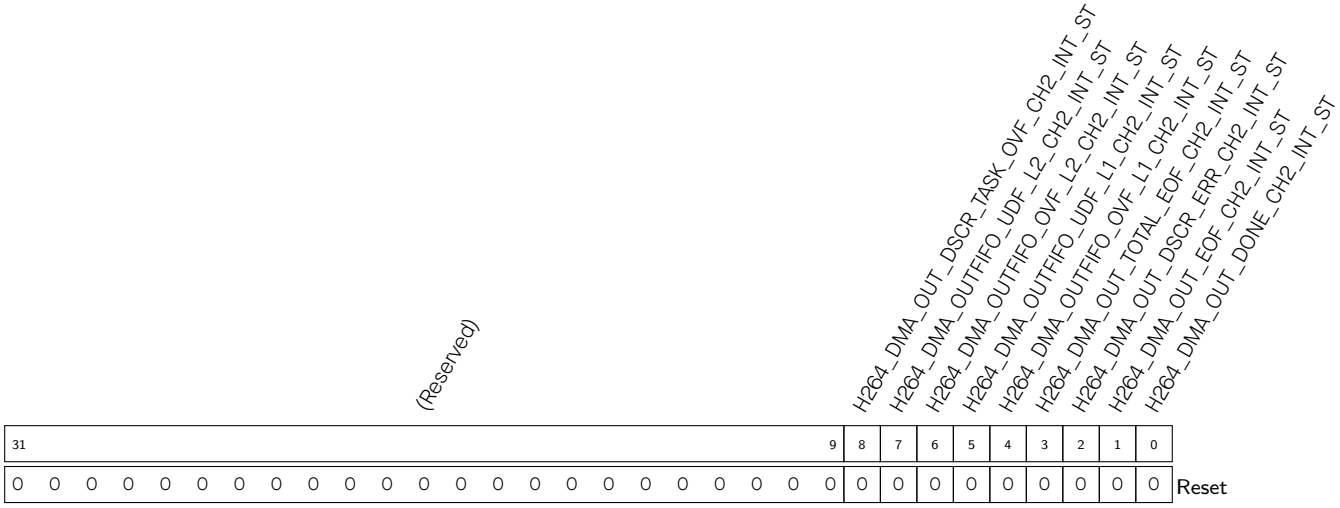
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L1_CH2_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L2_CH2_INT。(R/W)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L2_CH2_INT。(R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT。(R/W)

Register 41.183. H264_DMA_OUT_INT_ST_CH2_REG (0x020C)



H264_DMA_OUT_DONE_CH2_INT_ST H264_DMA_OUT_DONE_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_EOF_CH2_INT_ST H264_DMA_OUT_EOF_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_ERR_CH2_INT_ST H264_DMA_OUT_DSCR_ERR_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_ST H264_DMA_OUT_TOTAL_EOF_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_ST H264_DMA_OUTFIFO_OVF_L1_CH2_INT 的屏蔽中断状态。(RO)

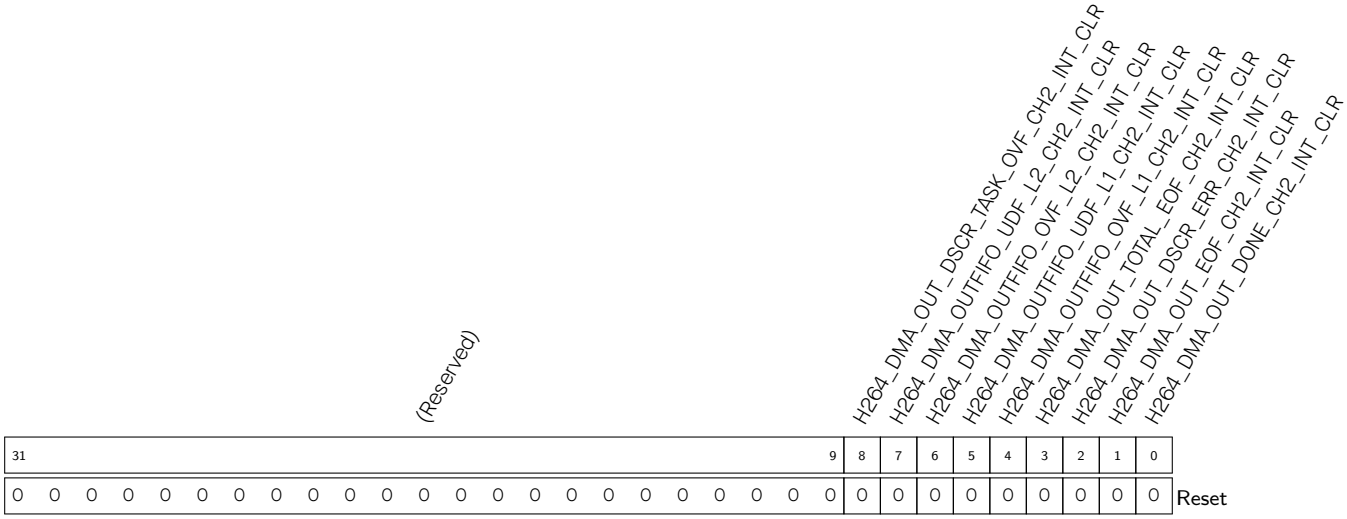
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_ST H264_DMA_OUTFIFO_UDF_L1_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_ST H264_DMA_OUTFIFO_OVF_L2_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_ST H264_DMA_OUTFIFO_UDF_L2_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_ST H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT 的屏蔽中断状态。(RO)

Register 41.184. H264_DMA_OUT_INT_CLR_CH2_REG (0x0210)



H264_DMA_OUT_DONE_CH2_INT_CLR 写 1 清除 H264_DMA_OUT_DONE_CH2_INT。(WT)

H264_DMA_OUT_EOF_CH2_INT_CLR 写 1 清除 H264_DMA_OUT_EOF_CH2_INT。(WT)

H264_DMA_OUT_DSCR_ERR_CH2_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_ERR_CH2_INT。(WT)

H264_DMA_OUT_TOTAL_EOF_CH2_INT_CLR 写 1 清除 H264_DMA_OUT_TOTAL_EOF_CH2_INT。(WT)

H264_DMA_OUTFIFO_OVF_L1_CH2_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L1_CH2_-INT。(WT)

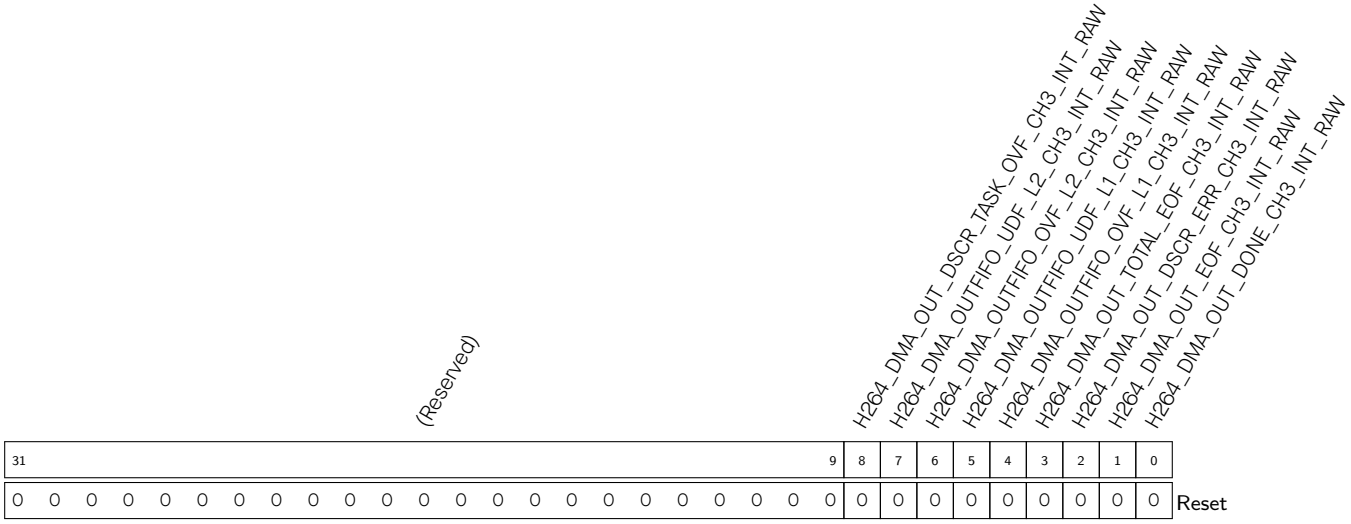
H264_DMA_OUTFIFO_UDF_L1_CH2_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L1_CH2_-INT。(WT)

H264_DMA_OUTFIFO_OVF_L2_CH2_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L2_CH2_-INT。(WT)

H264_DMA_OUTFIFO_UDF_L2_CH2_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L2_CH2_-INT。(WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH2_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_TASK_-OVF_CH2_INT。(WT)

Register 41.185. H264_DMA_OUT_INT_RAW_CH3_REG (0x0304)



H264_DMA_OUT_DONE_CH3_INT_RAW H264_DMA_OUT_DONE_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_EOF_CH3_INT_RAW H264_DMA_OUT_EOF_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH3_INT_RAW H264_DMA_OUT_DSCR_ERR_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_RAW H264_DMA_OUT_TOTAL_EOF_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_RAW H264_DMA_OUTFIFO_OVF_L1_CH3_INT 的原始中断状态。(R/WTC/SS)

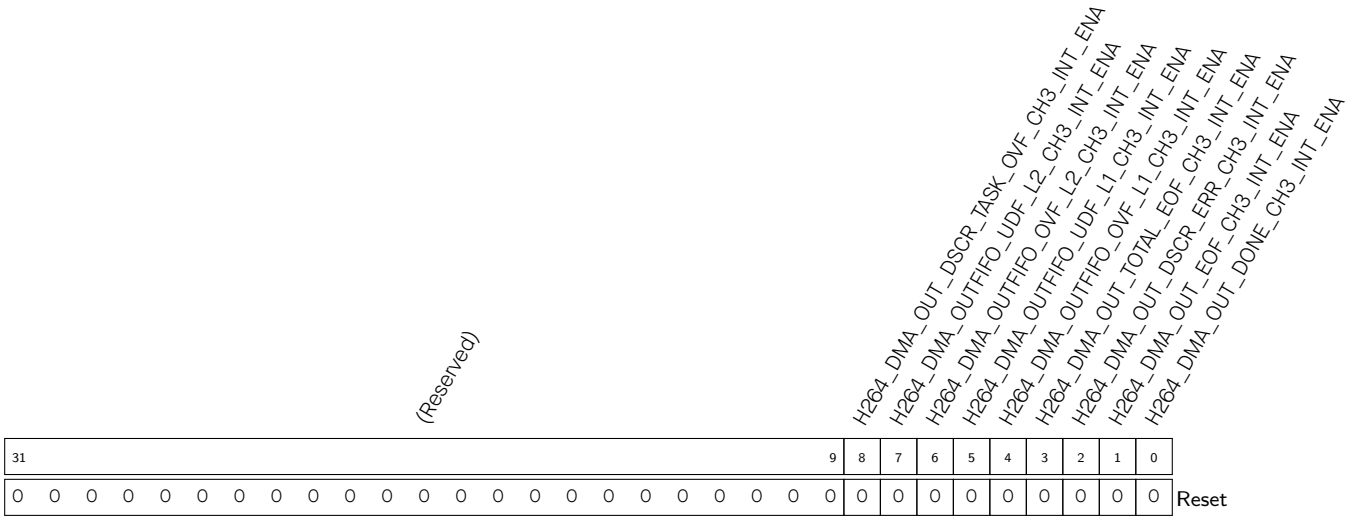
H264_DMA_OUTFIFO_UDF_L1_CH3_INT_RAW H264_DMA_OUTFIFO_UDF_L1_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_RAW H264_DMA_OUTFIFO_OVF_L2_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_RAW H264_DMA_OUTFIFO_UDF_L2_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_RAW H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT 的原始中断状态。(R/WTC/SS)

Register 41.186. H264_DMA_OUT_INT_ENA_CH3_REG (0x0308)



H264_DMA_OUT_DONE_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUT_DONE_CH3_INT。(R/W)

H264_DMA_OUT_EOF_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUT_EOF_CH3_INT。(R/W)

H264_DMA_OUT_DSCR_ERR_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_ERR_CH3_INT。(R/W)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUT_TOTAL_EOF_CH3_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L1_CH3_INT。(R/W)

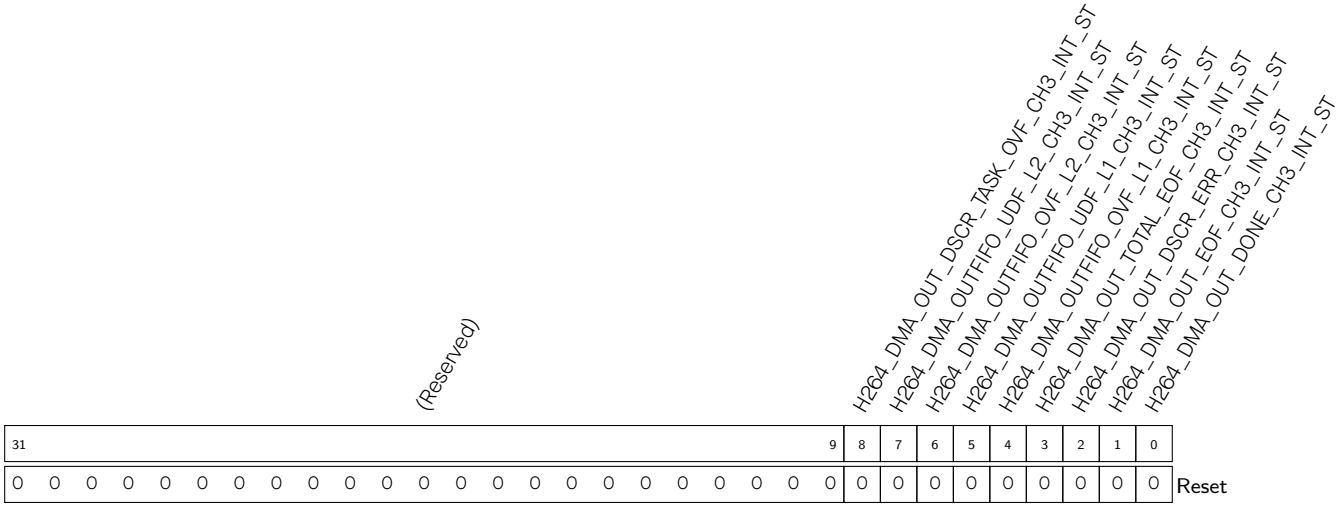
H264_DMA_OUTFIFO_UDF_L1_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L1_CH3_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L2_CH3_INT。(R/W)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L2_CH3_INT。(R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT。(R/W)

Register 41.187. H264_DMA_OUT_INT_ST_CH3_REG (0x030C)



H264_DMA_OUT_DONE_CH3_INT_ST H264_DMA_OUT_DONE_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_EOF_CH3_INT_ST H264_DMA_OUT_EOF_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_ERR_CH3_INT_ST H264_DMA_OUT_DSCR_ERR_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_ST H264_DMA_OUT_TOTAL_EOF_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_ST H264_DMA_OUTFIFO_OVF_L1_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_UDF_L1_CH3_INT_ST H264_DMA_OUTFIFO_UDF_L1_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_ST H264_DMA_OUTFIFO_OVF_L2_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_ST H264_DMA_OUTFIFO_UDF_L2_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_ST H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT 的屏蔽中断状态。(RO)

Register 41.188. H264_DMA_OUT_INT_CLR_CH3_REG (0x0310)

31–9	8	7	6	5	4	3	2	1	0	
(Reserved)	H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_CLR	H264_DMA_OUTFIFO_UDF_L2_CH3_INT_CLR	H264_DMA_OUTFIFO_OVF_L2_CH3_INT_CLR	H264_DMA_OUTFIFO_UDF_L1_CH3_INT_CLR	H264_DMA_OUTFIFO_OVF_L1_CH3_INT_CLR	H264_DMA_OUT_TOTAL_EOF_CH3_INT_CLR	H264_DMA_OUT_DSCR_ERR_CH3_INT_CLR	H264_DMA_OUT_EOF_CH3_INT_CLR	H264_DMA_OUT_DONE_CH3_INT_CLR	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

H264_DMA_OUT_DONE_CH3_INT_CLR 写 1 清除 H264_DMA_OUT_DONE_CH3_INT。(WT)

H264_DMA_OUT_EOF_CH3_INT_CLR 写 1 清除 H264_DMA_OUT_EOF_CH3_INT。(WT)

H264_DMA_OUT_DSCR_ERR_CH3_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_ERR_CH3_INT。(WT)

H264_DMA_OUT_TOTAL_EOF_CH3_INT_CLR 写 1 清除 H264_DMA_OUT_TOTAL_EOF_CH3_INT。(WT)

H264_DMA_OUTFIFO_OVF_L1_CH3_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L1_CH3_INT。(WT)

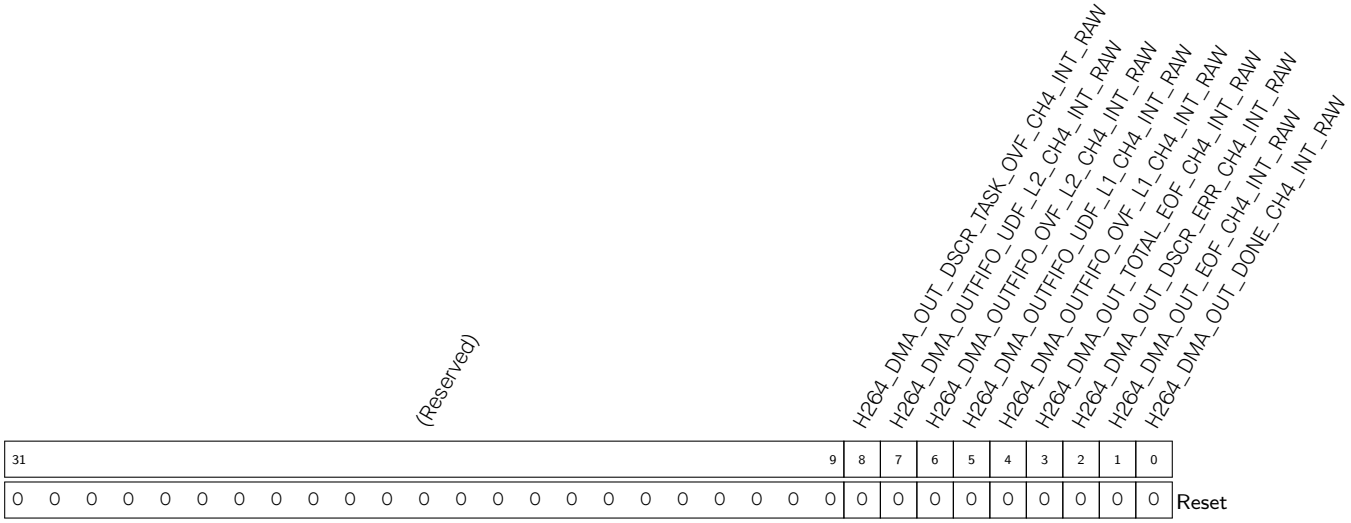
H264_DMA_OUTFIFO_UDF_L1_CH3_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L1_CH3_INT。(WT)

H264_DMA_OUTFIFO_OVF_L2_CH3_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L2_CH3_INT。(WT)

H264_DMA_OUTFIFO_UDF_L2_CH3_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L2_CH3_INT。(WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_TASK_OVF_CH3_INT。(WT)

Register 41.189. H264_DMA_OUT_INT_RAW_CH4_REG (0x0404)



H264_DMA_OUT_DONE_CH4_INT_RAW H264_DMA_OUT_DONE_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_EOF_CH4_INT_RAW H264_DMA_OUT_EOF_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_ERR_CH4_INT_RAW H264_DMA_OUT_DSCR_ERR_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_RAW H264_DMA_OUT_TOTAL_EOF_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_RAW H264_DMA_OUTFIFO_OVF_L1_CH4_INT 的原始中断状态。(R/WTC/SS)

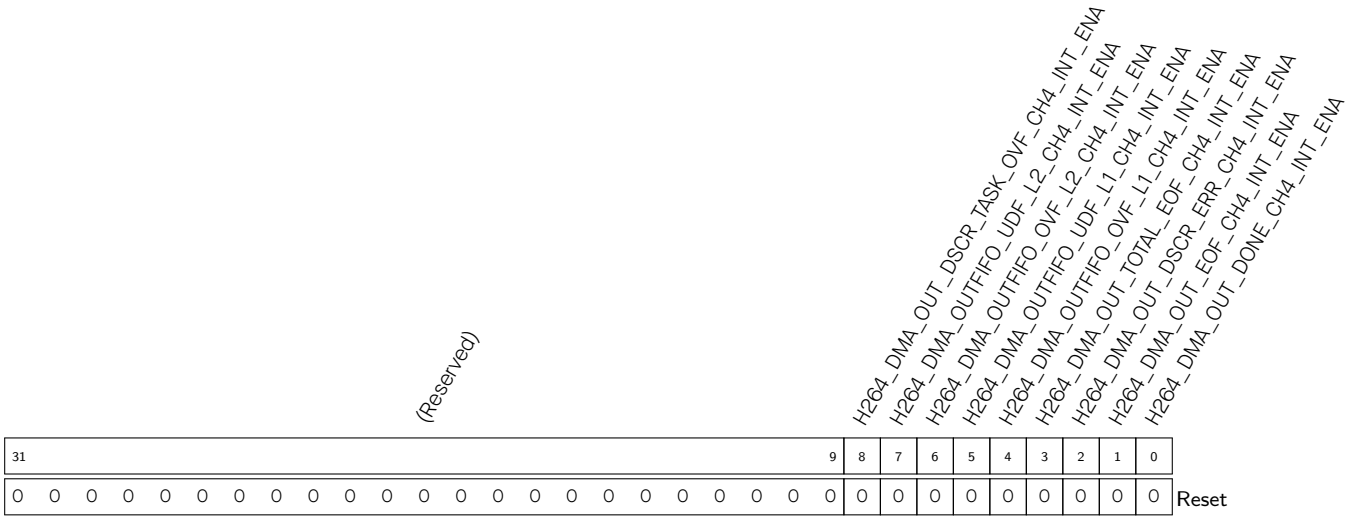
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_RAW H264_DMA_OUTFIFO_UDF_L1_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_RAW H264_DMA_OUTFIFO_OVF_L2_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_RAW H264_DMA_OUTFIFO_UDF_L2_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_RAW H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT 的原始中断状态。(R/WTC/SS)

Register 41.190. H264_DMA_OUT_INT_ENA_CH4_REG (0x0408)



H264_DMA_OUT_DONE_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUT_DONE_CH4_INT。(R/W)

H264_DMA_OUT_EOF_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUT_EOF_CH4_INT。(R/W)

H264_DMA_OUT_DSCR_ERR_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_ERR_CH4_INT。(R/W)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUT_TOTAL_EOF_CH4_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L1_CH4_INT。(R/W)

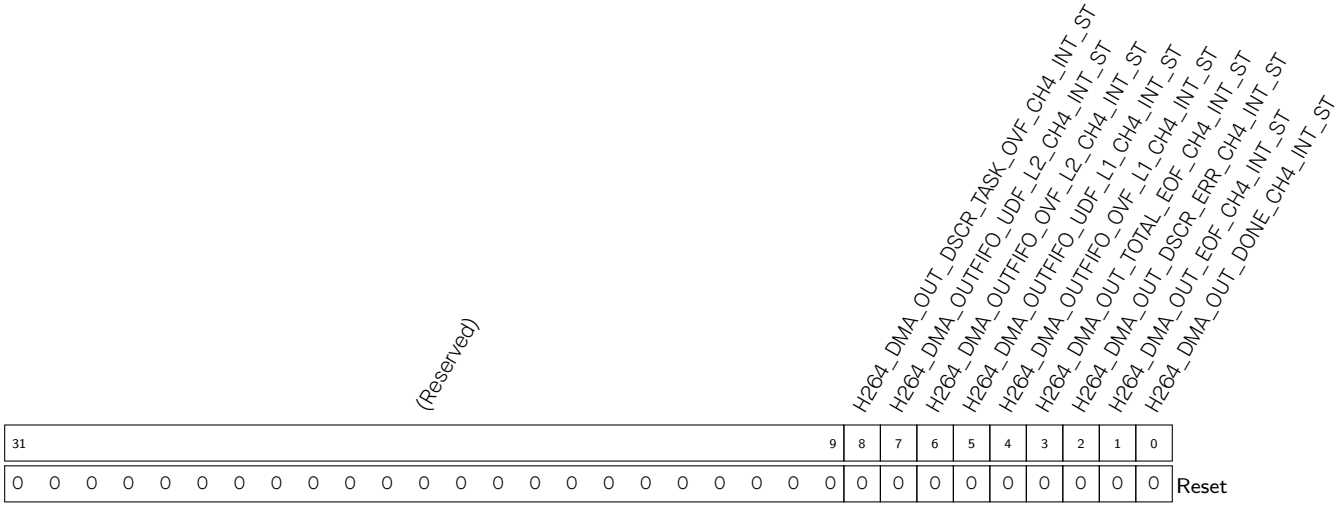
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L1_CH4_INT。(R/W)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_OVF_L2_CH4_INT。(R/W)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUTFIFO_UDF_L2_CH4_INT。(R/W)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_ENA 写入 1 以启用 H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT。(R/W)

Register 41.191. H264_DMA_OUT_INT_ST_CH4_REG (0x040C)



H264_DMA_OUT_DONE_CH4_INT_ST H264_DMA_OUT_DONE_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_EOF_CH4_INT_ST H264_DMA_OUT_EOF_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_ERR_CH4_INT_ST H264_DMA_OUT_DSCR_ERR_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_ST H264_DMA_OUT_TOTAL_EOF_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_ST H264_DMA_OUTFIFO_OVF_L1_CH4_INT 的屏蔽中断状态。(RO)

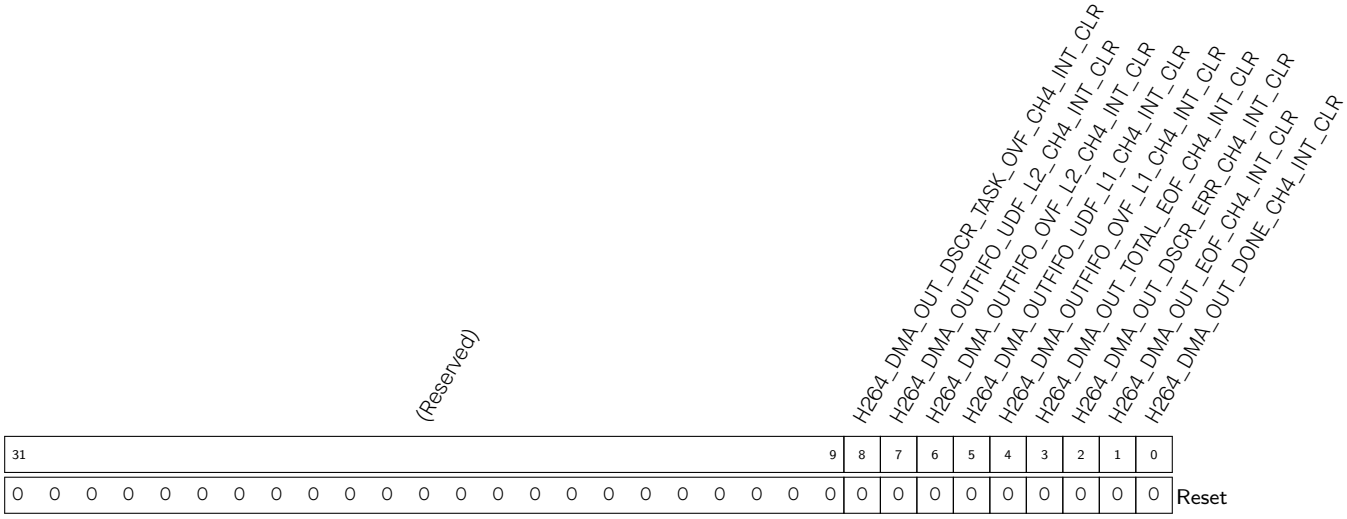
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_ST H264_DMA_OUTFIFO_UDF_L1_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_ST H264_DMA_OUTFIFO_OVF_L2_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_ST H264_DMA_OUTFIFO_UDF_L2_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_ST H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT 的屏蔽中断状态。(RO)

Register 41.192. H264_DMA_OUT_INT_CLR_CH4_REG (0x0410)



H264_DMA_OUT_DONE_CH4_INT_CLR 写 1 清除 H264_DMA_OUT_DONE_CH4_INT。(WT)

H264_DMA_OUT_EOF_CH4_INT_CLR 写 1 清除 H264_DMA_OUT_EOF_CH4_INT。(WT)

H264_DMA_OUT_DSCR_ERR_CH4_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_ERR_CH4_INT。(WT)

H264_DMA_OUT_TOTAL_EOF_CH4_INT_CLR 写 1 清除 H264_DMA_OUT_TOTAL_EOF_CH4_INT。(WT)

H264_DMA_OUTFIFO_OVF_L1_CH4_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L1_CH4_INT。(WT)

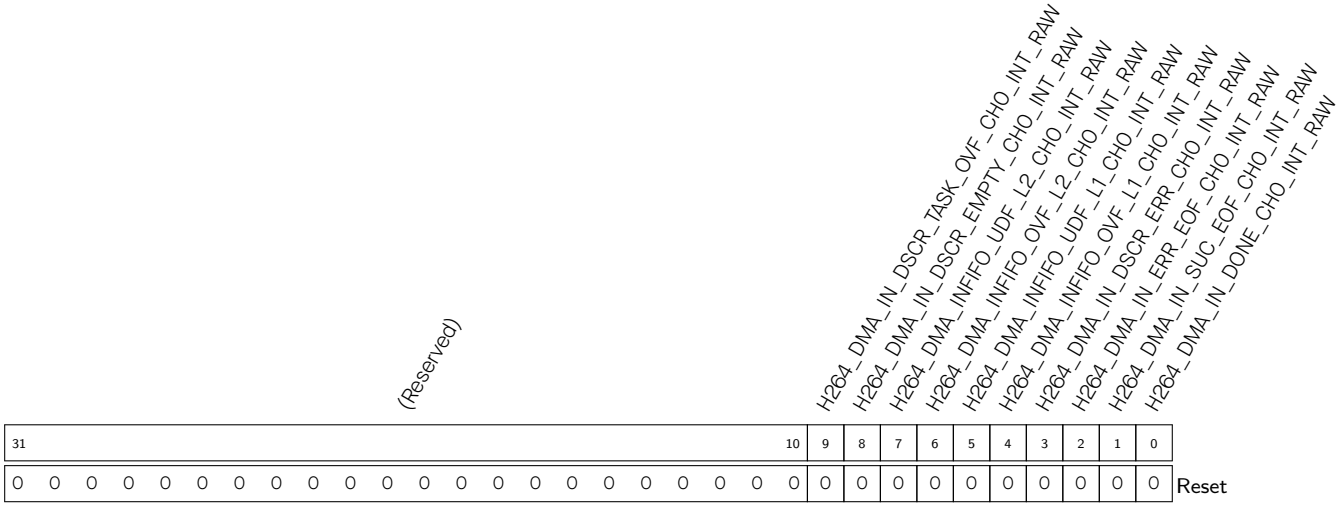
H264_DMA_OUTFIFO_UDF_L1_CH4_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L1_CH4_INT。(WT)

H264_DMA_OUTFIFO_OVF_L2_CH4_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_OVF_L2_CH4_INT。(WT)

H264_DMA_OUTFIFO_UDF_L2_CH4_INT_CLR 写 1 清除 H264_DMA_OUTFIFO_UDF_L2_CH4_INT。(WT)

H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT_CLR 写 1 清除 H264_DMA_OUT_DSCR_TASK_OVF_CH4_INT。(WT)

Register 41.193. H264_DMA_IN_INT_RAW_CH0_REG (0x0504)



H264_DMA_IN_DONE_CH0_INT_RAW H264_DMA_IN_DONE_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH0_INT_RAW H264_DMA_IN_SUC_EOF_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH0_INT_RAW H264_DMA_IN_ERR_EOF_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH0_INT_RAW H264_DMA_IN_DSCR_ERR_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH0_INT_RAW H264_DMA_INFIFO_OVF_L1_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH0_INT_RAW H264_DMA_INFIFO_UDF_L1_CH0_INT 的原始中断状态。(R/WTC/SS)

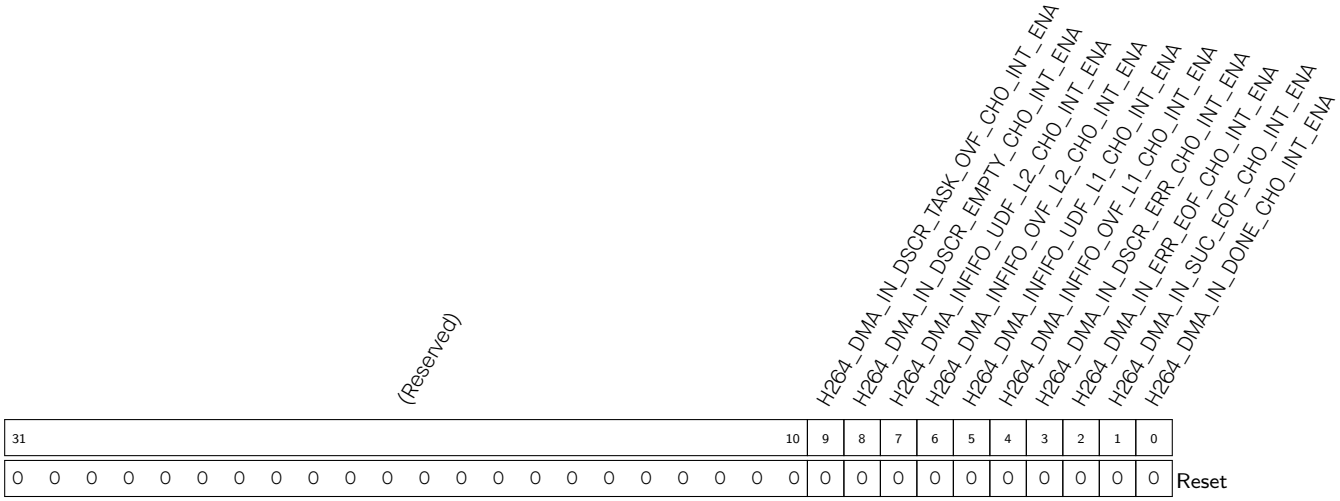
H264_DMA_INFIFO_OVF_L2_CH0_INT_RAW H264_DMA_INFIFO_OVF_L2_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH0_INT_RAW H264_DMA_INFIFO_UDF_L2_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_RAW H264_DMA_IN_DSCR_EMPTY_CH0_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_RAW H264_DMA_IN_DSCR_TASK_OVF_CH0_INT 的原始中断状态。(R/WTC/SS)

Register 41.194. H264_DMA_IN_INT_ENA_CH0_REG (0x0508)



H264_DMA_IN_DONE_CH0_INT_ENA 写入 1 以启用 H264_DMA_IN_DONE_CH0_INT。(R/W)

H264_DMA_IN_SUC_EOF_CH0_INT_ENA 写入 1 以启用 H264_DMA_IN_SUC_EOF_CH0_INT。(R/W)

H264_DMA_IN_ERR_EOF_CH0_INT_ENA 写入 1 以启用 H264_DMA_IN_ERR_EOF_CH0_INT。(R/W)

H264_DMA_IN_DSCR_ERR_CH0_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_ERR_CH0_INT。(R/W)

H264_DMA_INFIFO_OVF_L1_CH0_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L1_CH0_INT。(R/W)

H264_DMA_INFIFO_UDF_L1_CH0_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L1_CH0_INT。(R/W)

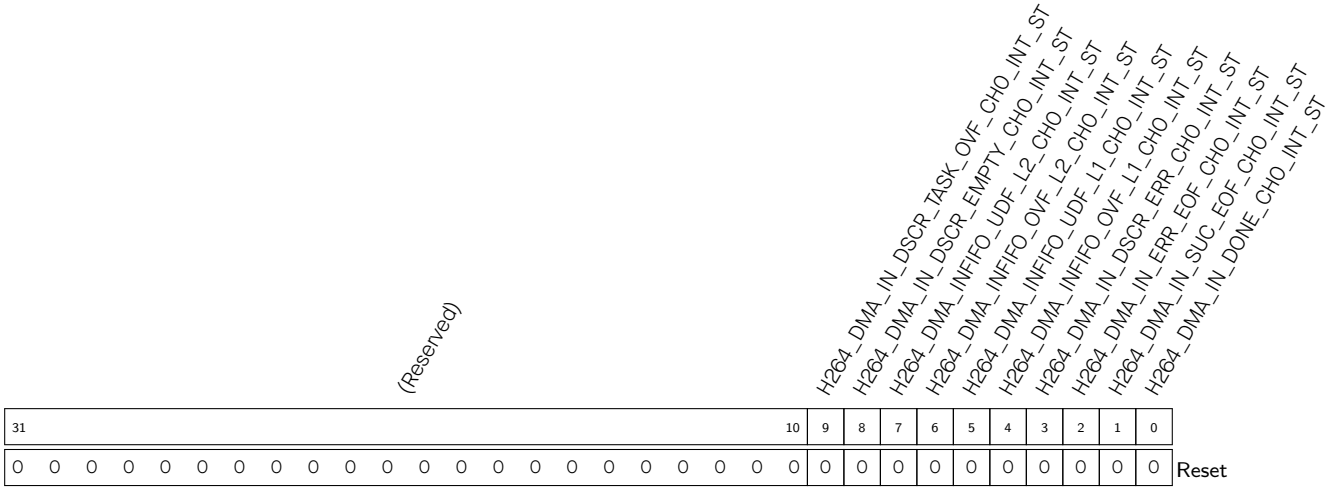
H264_DMA_INFIFO_OVF_L2_CH0_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L2_CH0_INT。(R/W)

H264_DMA_INFIFO_UDF_L2_CH0_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L2_CH0_INT。(R/W)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_EMPTY_CH0_INT。(R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_TASK_OVF_CH0_INT。(R/W)

Register 41.195. H264_DMA_IN_INT_ST_CH0_REG (0x050C)



H264_DMA_IN_DONE_CH0_INT_ST H264_DMA_IN_DONE_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_SUC_EOF_CH0_INT_ST H264_DMA_IN_SUC_EOF_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_ERR_EOF_CH0_INT_ST H264_DMA_IN_ERR_EOF_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_ERR_CH0_INT_ST H264_DMA_IN_DSCR_ERR_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_OVF_L1_CH0_INT_ST H264_DMA_INFIFO_OVF_L1_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L1_CH0_INT_ST H264_DMA_INFIFO_UDF_L1_CH0_INT 的屏蔽中断状态。(RO)

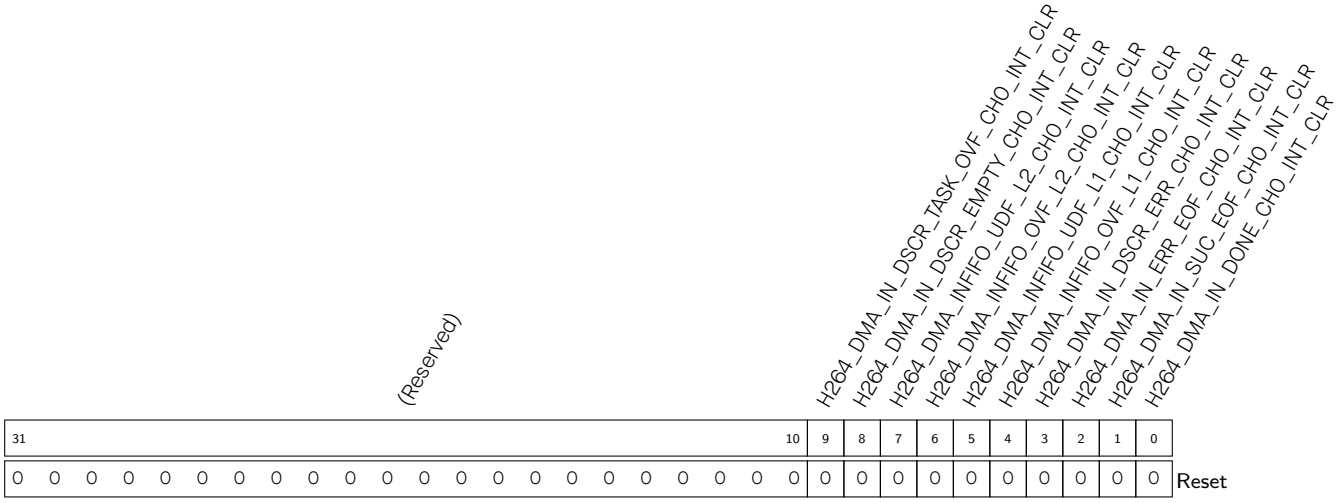
H264_DMA_INFIFO_OVF_L2_CH0_INT_ST H264_DMA_INFIFO_OVF_L2_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L2_CH0_INT_ST H264_DMA_INFIFO_UDF_L2_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_ST H264_DMA_IN_DSCR_EMPTY_CH0_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_ST H264_DMA_IN_DSCR_TASK_OVF_CH0_INT 的屏蔽中断状态。(RO)

Register 41.196. H264_DMA_IN_INT_CLR_CH0_REG (0x0510)



H264_DMA_IN_DONE_CH0_INT_CLR 写 1 清除 H264_DMA_IN_DONE_CH0_INT。(WT)

H264_DMA_IN_SUC_EOF_CH0_INT_CLR 写 1 清除 H264_DMA_IN_SUC_EOF_CH0_INT。(WT)

H264_DMA_IN_ERR_EOF_CH0_INT_CLR 写 1 清除 H264_DMA_IN_ERR_EOF_CH0_INT。(WT)

H264_DMA_IN_DSCR_ERR_CH0_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_ERR_CH0_INT。(WT)

H264_DMA_INFIFO_OVF_L1_CH0_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L1_CH0_INT。(WT)

H264_DMA_INFIFO_UDF_L1_CH0_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L1_CH0_INT。(WT)

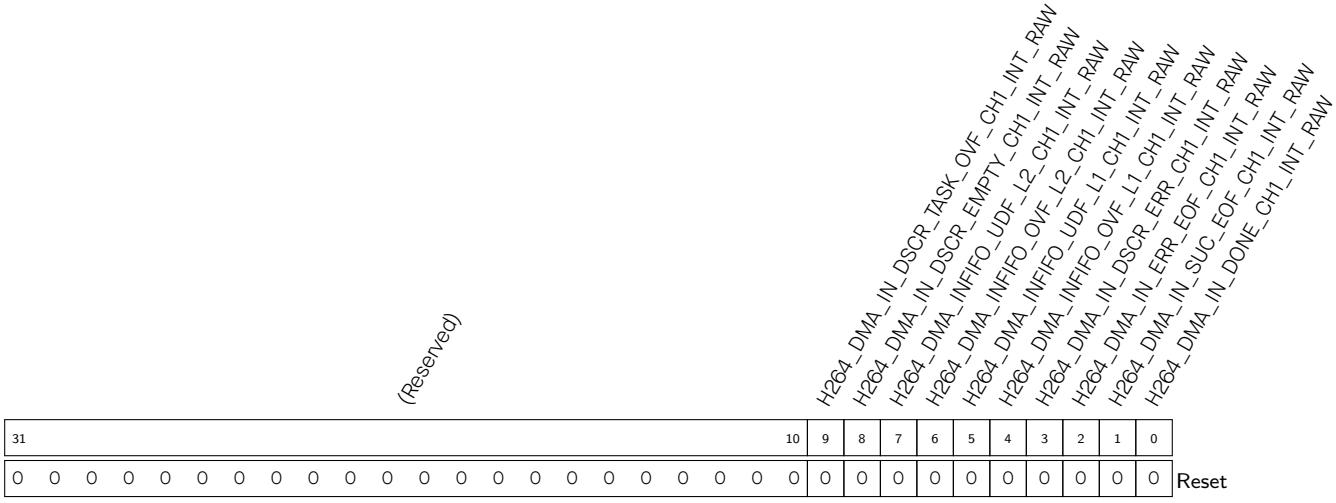
H264_DMA_INFIFO_OVF_L2_CH0_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L2_CH0_INT。(WT)

H264_DMA_INFIFO_UDF_L2_CH0_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L2_CH0_INT。(WT)

H264_DMA_IN_DSCR_EMPTY_CH0_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_EMPTY_CH0_INT。(WT)

H264_DMA_IN_DSCR_TASK_OVF_CH0_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_TASK_OVF_CH0_INT。(WT)

Register 41.197. H264_DMA_IN_INT_RAW_CH1_REG (0x0604)



H264_DMA_IN_DONE_CH1_INT_RAW H264_DMA_IN_DONE_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH1_INT_RAW H264_DMA_IN_SUC_EOF_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH1_INT_RAW H264_DMA_IN_ERR_EOF_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH1_INT_RAW H264_DMA_IN_DSCR_ERR_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH1_INT_RAW H264_DMA_INFIFO_OVF_L1_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH1_INT_RAW H264_DMA_INFIFO_UDF_L1_CH1_INT 的原始中断状态。(R/WTC/SS)

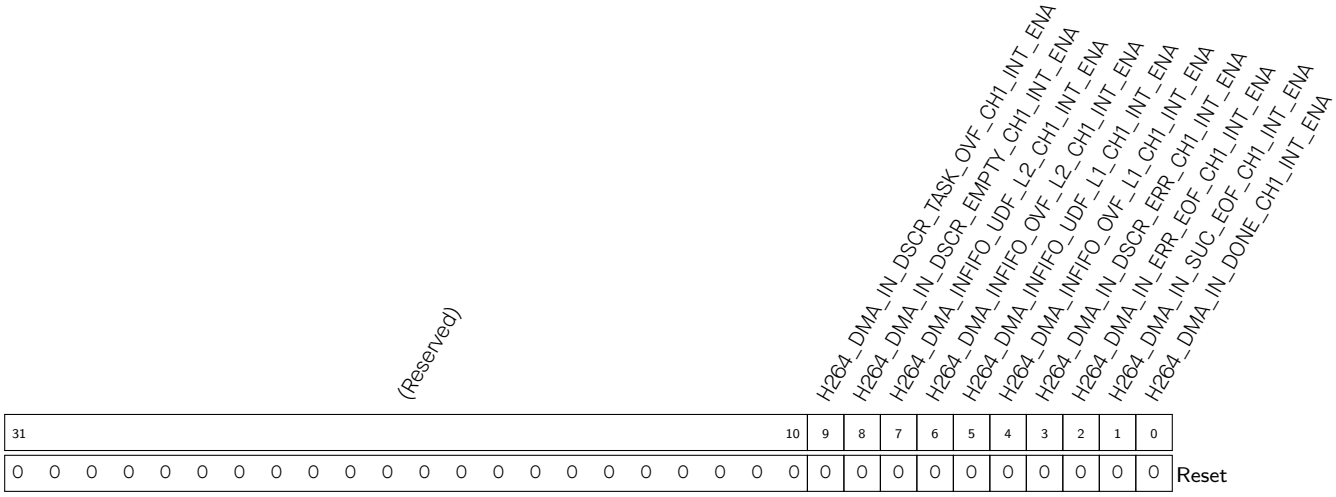
H264_DMA_INFIFO_OVF_L2_CH1_INT_RAW H264_DMA_INFIFO_OVF_L2_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH1_INT_RAW H264_DMA_INFIFO_UDF_L2_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_RAW H264_DMA_IN_DSCR_EMPTY_CH1_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_RAW H264_DMA_IN_DSCR_TASK_OVF_CH1_INT 的原始中断状态。(R/WTC/SS)

Register 41.198. H264_DMA_IN_INT_ENA_CH1_REG (0x0608)



H264_DMA_IN_DONE_CH1_INT_ENA 写入 1 以启用 H264_DMA_IN_DONE_CH1_INT。(R/W)

H264_DMA_IN_SUC_EOF_CH1_INT_ENA 写入 1 以启用 H264_DMA_IN_SUC_EOF_CH1_INT。(R/W)

H264_DMA_IN_ERR_EOF_CH1_INT_ENA 写入 1 以启用 H264_DMA_IN_ERR_EOF_CH1_INT。(R/W)

H264_DMA_IN_DSCR_ERR_CH1_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_ERR_CH1_INT。(R/W)

H264_DMA_INFIFO_OVF_L1_CH1_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L1_CH1_INT。(R/W)

H264_DMA_INFIFO_UDF_L1_CH1_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L1_CH1_INT。(R/W)

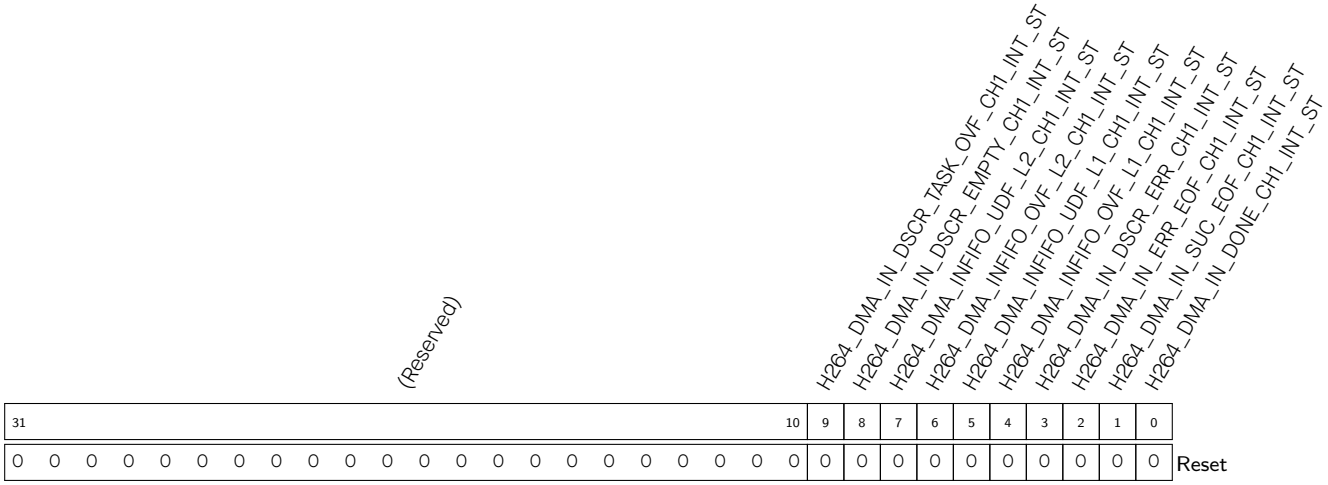
H264_DMA_INFIFO_OVF_L2_CH1_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L2_CH1_INT。(R/W)

H264_DMA_INFIFO_UDF_L2_CH1_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L2_CH1_INT。(R/W)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_EMPTY_CH1_INT。(R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_TASK_OVF_CH1_INT。(R/W)

Register 41.199. H264_DMA_IN_INT_ST_CH1_REG (0x060C)



H264_DMA_IN_DONE_CH1_INT_ST H264_DMA_IN_DONE_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_SUC_EOF_CH1_INT_ST H264_DMA_IN_SUC_EOF_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_ERR_EOF_CH1_INT_ST H264_DMA_IN_ERR_EOF_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_ERR_CH1_INT_ST H264_DMA_IN_DSCR_ERR_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_OVF_L1_CH1_INT_ST H264_DMA_INFIFO_OVF_L1_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L1_CH1_INT_ST H264_DMA_INFIFO_UDF_L1_CH1_INT 的屏蔽中断状态。(RO)

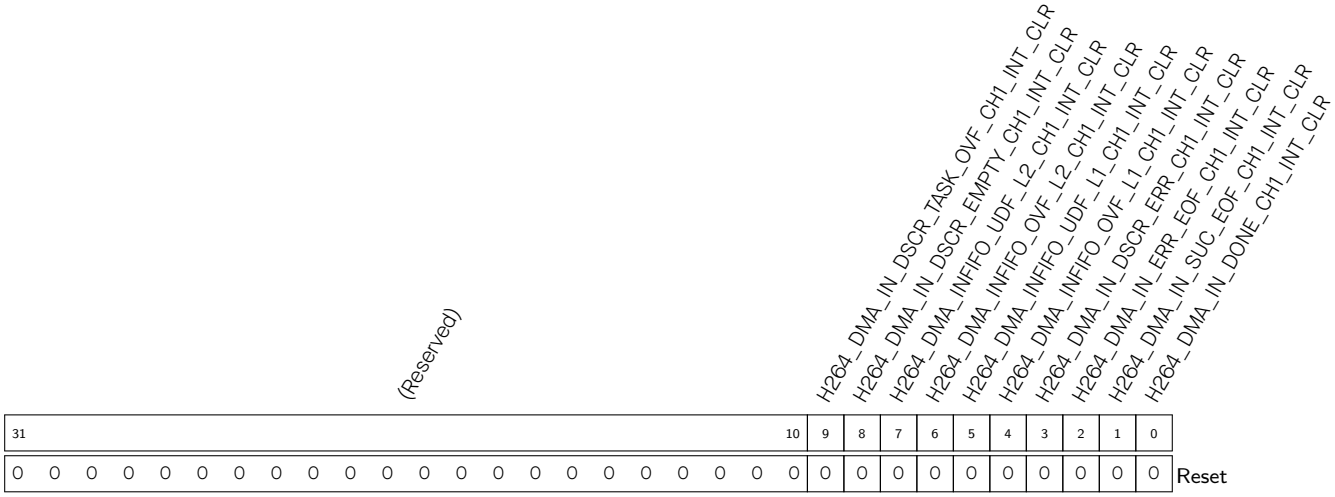
H264_DMA_INFIFO_OVF_L2_CH1_INT_ST H264_DMA_INFIFO_OVF_L2_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L2_CH1_INT_ST H264_DMA_INFIFO_UDF_L2_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_ST H264_DMA_IN_DSCR_EMPTY_CH1_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_ST H264_DMA_IN_DSCR_TASK_OVF_CH1_INT 的屏蔽中断状态。(RO)

Register 41.200. H264_DMA_IN_INT_CLR_CH1_REG (0x0610)



H264_DMA_IN_DONE_CH1_INT_CLR 写 1 清除 H264_DMA_IN_DONE_CH1_INT。(WT)

H264_DMA_IN_SUC_EOF_CH1_INT_CLR 写 1 清除 H264_DMA_IN_SUC_EOF_CH1_INT。(WT)

H264_DMA_IN_ERR_EOF_CH1_INT_CLR 写 1 清除 H264_DMA_IN_ERR_EOF_CH1_INT。(WT)

H264_DMA_IN_DSCR_ERR_CH1_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_ERR_CH1_INT。(WT)

H264_DMA_INFIFO_OVF_L1_CH1_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L1_CH1_INT。(WT)

H264_DMA_INFIFO_UDF_L1_CH1_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L1_CH1_INT。(WT)

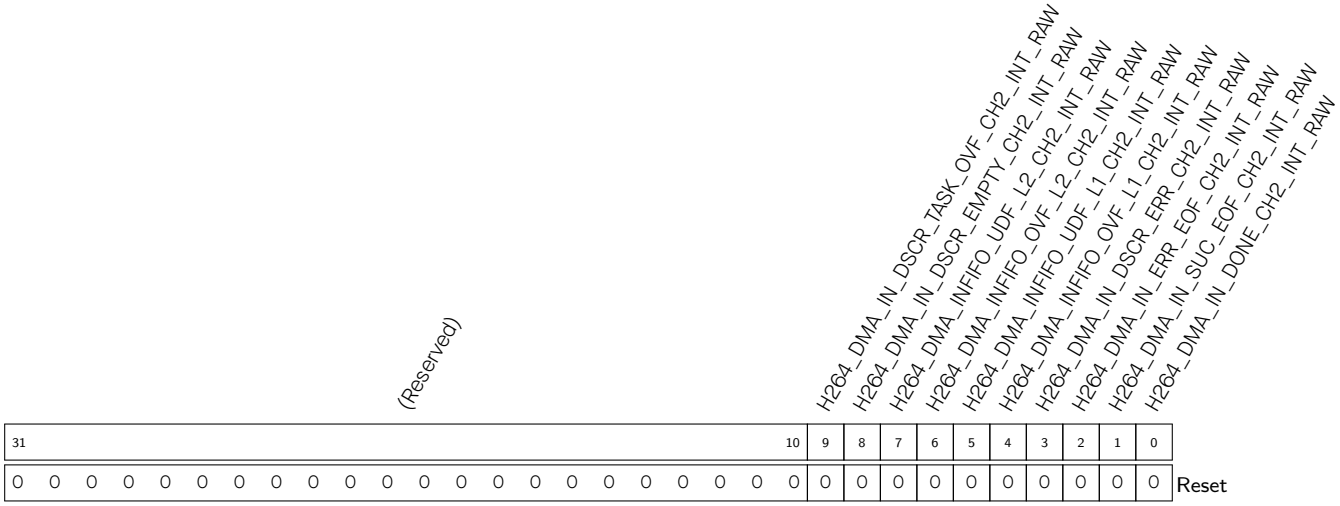
H264_DMA_INFIFO_OVF_L2_CH1_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L2_CH1_INT。(WT)

H264_DMA_INFIFO_UDF_L2_CH1_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L2_CH1_INT。(WT)

H264_DMA_IN_DSCR_EMPTY_CH1_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_EMPTY_CH1_INT。(WT)

H264_DMA_IN_DSCR_TASK_OVF_CH1_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_TASK_OVF_-CH1_INT。(WT)

Register 41.201. H264_DMA_IN_INT_RAW_CH2_REG (0x0704)



H264_DMA_IN_DONE_CH2_INT_RAW H264_DMA_IN_DONE_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH2_INT_RAW H264_DMA_IN_SUC_EOF_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH2_INT_RAW H264_DMA_IN_ERR_EOF_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH2_INT_RAW H264_DMA_IN_DSCR_ERR_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH2_INT_RAW H264_DMA_INFIFO_OVF_L1_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH2_INT_RAW H264_DMA_INFIFO_UDF_L1_CH2_INT 的原始中断状态。(R/WTC/SS)

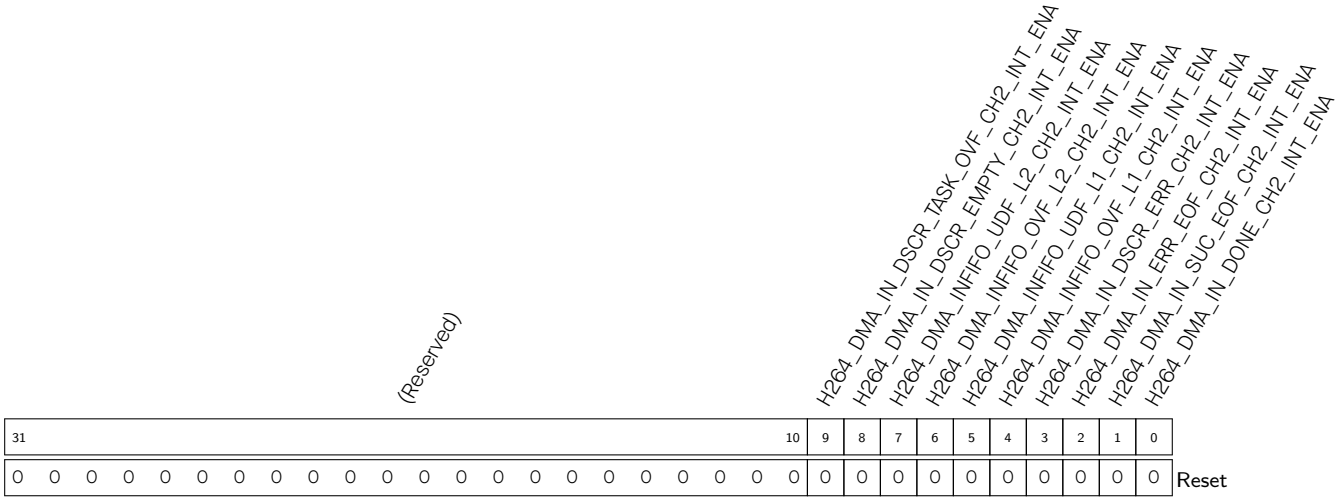
H264_DMA_INFIFO_OVF_L2_CH2_INT_RAW H264_DMA_INFIFO_OVF_L2_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH2_INT_RAW H264_DMA_INFIFO_UDF_L2_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_RAW H264_DMA_IN_DSCR_EMPTY_CH2_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_RAW H264_DMA_IN_DSCR_TASK_OVF_CH2_INT 的原始中断状态。(R/WTC/SS)

Register 41.202. H264_DMA_IN_INT_ENA_CH2_REG (0x0708)



H264_DMA_IN_DONE_CH2_INT_ENA 写入 1 以启用 H264_DMA_IN_DONE_CH2_INT。(R/W)

H264_DMA_IN_SUC_EOF_CH2_INT_ENA 写入 1 以启用 H264_DMA_IN_SUC_EOF_CH2_INT。(R/W)

H264_DMA_IN_ERR_EOF_CH2_INT_ENA 写入 1 以启用 H264_DMA_IN_ERR_EOF_CH2_INT。(R/W)

H264_DMA_IN_DSCR_ERR_CH2_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_ERR_CH2_INT。(R/W)

H264_DMA_INFIFO_OVF_L1_CH2_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L1_CH2_INT。(R/W)

H264_DMA_INFIFO_UDF_L1_CH2_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L1_CH2_INT。(R/W)

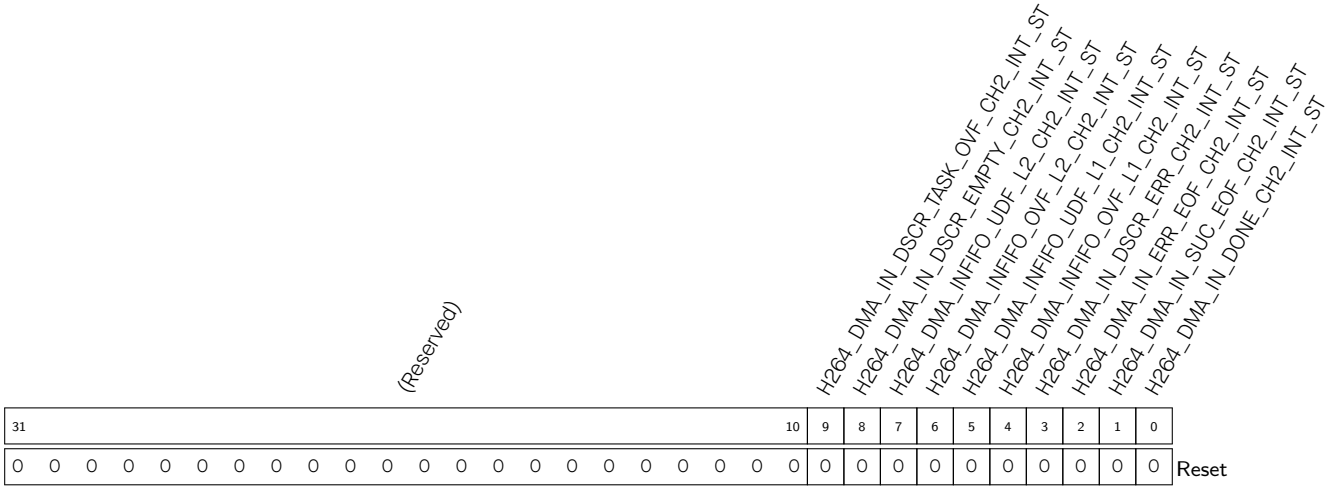
H264_DMA_INFIFO_OVF_L2_CH2_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L2_CH2_INT。(R/W)

H264_DMA_INFIFO_UDF_L2_CH2_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L2_CH2_INT。(R/W)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_EMPTY_CH2_INT。(R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_TASK_OVF_CH2_INT。(R/W)

Register 41.203. H264_DMA_IN_INT_ST_CH2_REG (0x070C)



H264_DMA_IN_DONE_CH2_INT_ST H264_DMA_IN_DONE_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_SUC_EOF_CH2_INT_ST H264_DMA_IN_SUC_EOF_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_ERR_EOF_CH2_INT_ST H264_DMA_IN_ERR_EOF_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_ERR_CH2_INT_ST H264_DMA_IN_DSCR_ERR_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_OVF_L1_CH2_INT_ST H264_DMA_INFIFO_OVF_L1_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L1_CH2_INT_ST H264_DMA_INFIFO_UDF_L1_CH2_INT 的屏蔽中断状态。(RO)

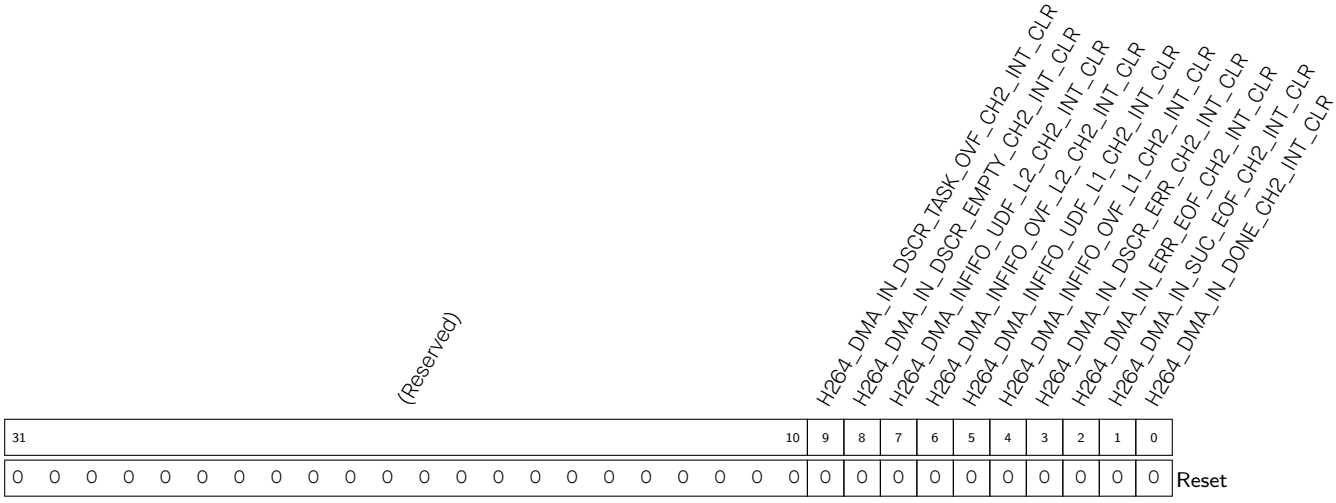
H264_DMA_INFIFO_OVF_L2_CH2_INT_ST H264_DMA_INFIFO_OVF_L2_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L2_CH2_INT_ST H264_DMA_INFIFO_UDF_L2_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_ST H264_DMA_IN_DSCR_EMPTY_CH2_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_ST H264_DMA_IN_DSCR_TASK_OVF_CH2_INT 的屏蔽中断状态。(RO)

Register 41.204. H264_DMA_IN_INT_CLR_CH2_REG (0x0710)



H264_DMA_IN_DONE_CH2_INT_CLR 写 1 清除 H264_DMA_IN_DONE_CH2_INT。(WT)

H264_DMA_IN_SUC_EOF_CH2_INT_CLR 写 1 清除 H264_DMA_IN_SUC_EOF_CH2_INT。(WT)

H264_DMA_IN_ERR_EOF_CH2_INT_CLR 写 1 清除 H264_DMA_IN_ERR_EOF_CH2_INT。(WT)

H264_DMA_IN_DSCR_ERR_CH2_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_ERR_CH2_INT。(WT)

H264_DMA_INFIFO_OVF_L1_CH2_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L1_CH2_INT。(WT)

H264_DMA_INFIFO_UDF_L1_CH2_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L1_CH2_INT。(WT)

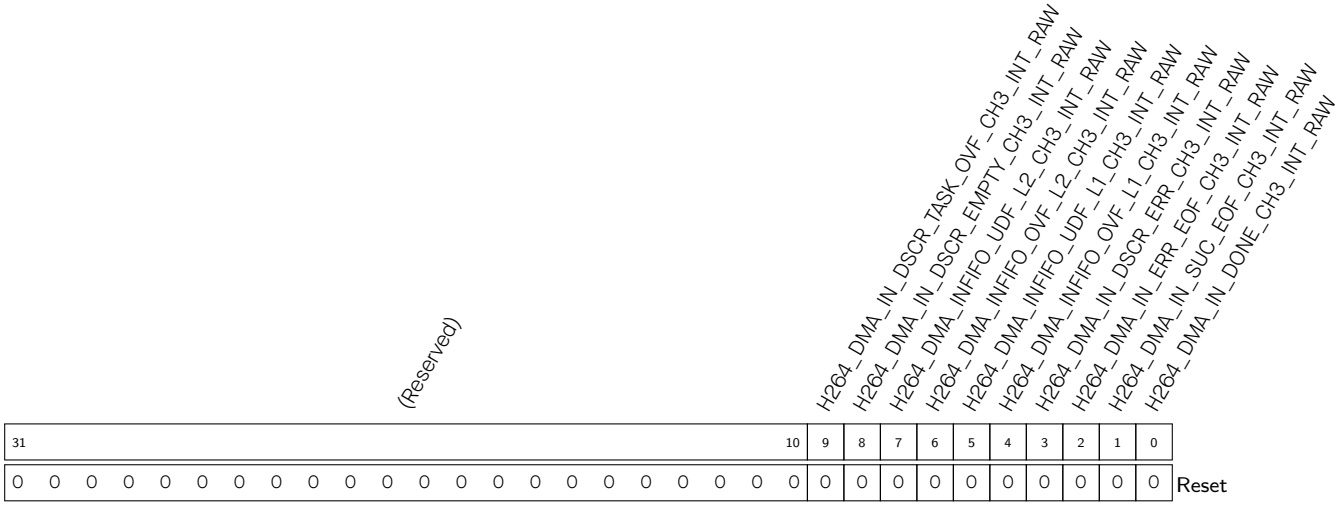
H264_DMA_INFIFO_OVF_L2_CH2_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L2_CH2_INT。(WT)

H264_DMA_INFIFO_UDF_L2_CH2_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L2_CH2_INT。(WT)

H264_DMA_IN_DSCR_EMPTY_CH2_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_EMPTY_CH2_INT。(WT)

H264_DMA_IN_DSCR_TASK_OVF_CH2_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_TASK_OVF_CH2_INT。(WT)

Register 41.205. H264_DMA_IN_INT_RAW_CH3_REG (0x0804)



H264_DMA_IN_DONE_CH3_INT_RAW H264_DMA_IN_DONE_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH3_INT_RAW H264_DMA_IN_SUC_EOF_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH3_INT_RAW H264_DMA_IN_ERR_EOF_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH3_INT_RAW H264_DMA_IN_DSCR_ERR_CH3_INT 通道 3 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH3_INT_RAW H264_DMA_INFIFO_OVF_L1_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH3_INT_RAW H264_DMA_INFIFO_UDF_L1_CH3_INT 的原始中断状态。(R/WTC/SS)

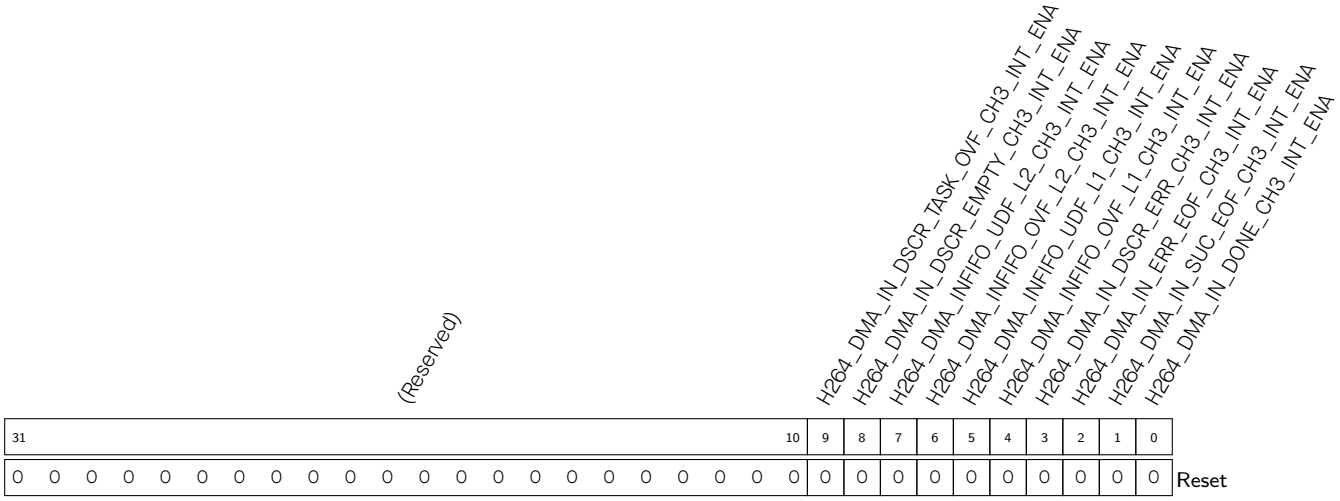
H264_DMA_INFIFO_OVF_L2_CH3_INT_RAW H264_DMA_INFIFO_OVF_L2_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH3_INT_RAW H264_DMA_INFIFO_UDF_L2_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_RAW H264_DMA_IN_DSCR_EMPTY_CH3_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_RAW H264_DMA_IN_DSCR_TASK_OVF_CH3_INT 的原始中断状态。(R/WTC/SS)

Register 41.206. H264_DMA_IN_INT_ENA_CH3_REG (0x0808)



H264_DMA_IN_DONE_CH3_INT_ENA 写入 1 以启用 H264_DMA_IN_DONE_CH3_INT。(R/W)

H264_DMA_IN_SUC_EOF_CH3_INT_ENA 写入 1 以启用 H264_DMA_IN_SUC_EOF_CH3_INT。(R/W)

H264_DMA_IN_ERR_EOF_CH3_INT_ENA 写入 1 以启用 H264_DMA_IN_ERR_EOF_CH3_INT。(R/W)

H264_DMA_IN_DSCR_ERR_CH3_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_ERR_CH3_INT。(R/W)

H264_DMA_INFIFO_OVF_L1_CH3_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L1_CH3_INT。(R/W)

H264_DMA_INFIFO_UDF_L1_CH3_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L1_CH3_INT。(R/W)

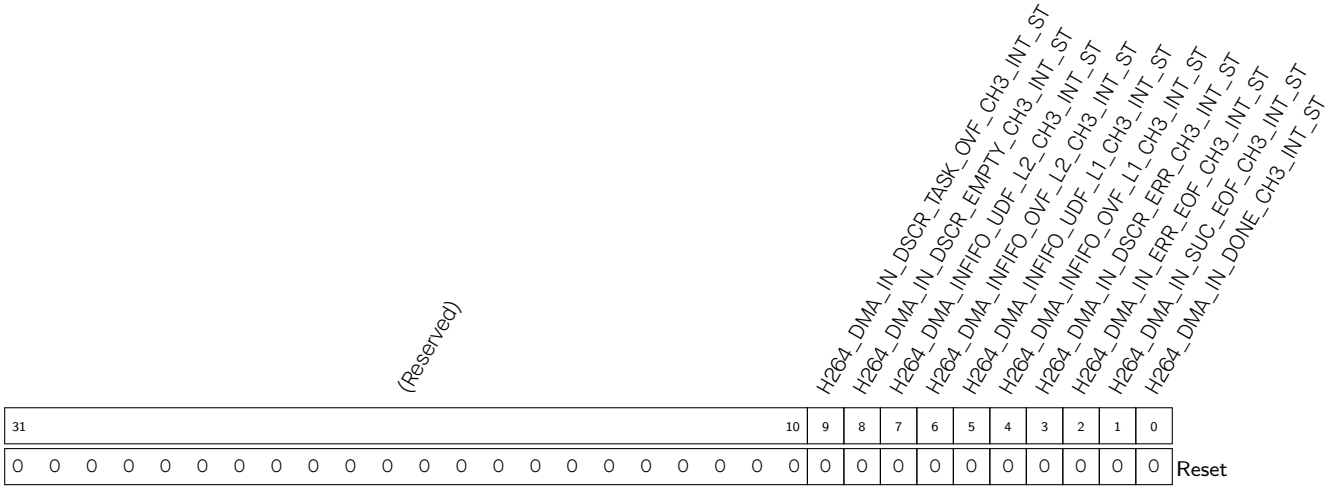
H264_DMA_INFIFO_OVF_L2_CH3_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L2_CH3_INT。(R/W)

H264_DMA_INFIFO_UDF_L2_CH3_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L2_CH3_INT。(R/W)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_EMPTY_CH3_INT。(R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_TASK_OVF_CH3_INT。(R/W)

Register 41.207. H264_DMA_IN_INT_ST_CH3_REG (0x080C)



H264_DMA_IN_DONE_CH3_INT_ST H264_DMA_IN_DONE_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_SUC_EOF_CH3_INT_ST H264_DMA_IN_SUC_EOF_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_ERR_EOF_CH3_INT_ST H264_DMA_IN_ERR_EOF_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_ERR_CH3_INT_ST H264_DMA_IN_DSCR_ERR_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_OVF_L1_CH3_INT_ST H264_DMA_INFIFO_OVF_L1_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L1_CH3_INT_ST H264_DMA_INFIFO_UDF_L1_CH3_INT 的屏蔽中断状态。(RO)

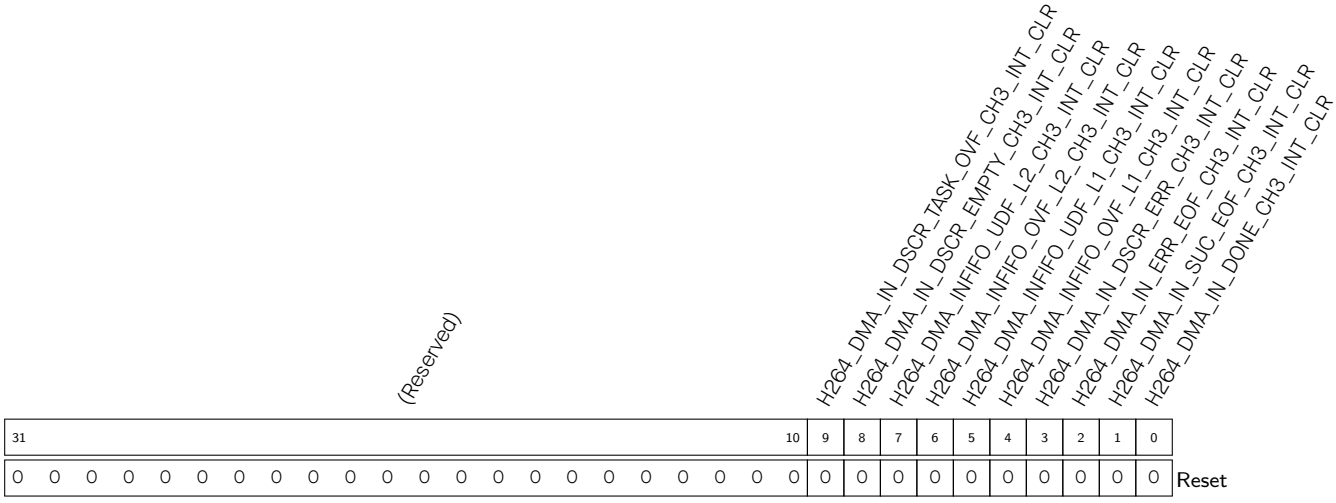
H264_DMA_INFIFO_OVF_L2_CH3_INT_ST H264_DMA_INFIFO_OVF_L2_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L2_CH3_INT_ST H264_DMA_INFIFO_UDF_L2_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_ST H264_DMA_IN_DSCR_EMPTY_CH3_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_ST H264_DMA_IN_DSCR_TASK_OVF_CH3_INT 的屏蔽中断状态。(RO)

Register 41.208. H264_DMA_IN_INT_CLR_CH3_REG (0x0810)



H264_DMA_IN_DONE_CH3_INT_CLR 写 1 清除 H264_DMA_IN_DONE_CH3_INT。(WT)

H264_DMA_IN_SUC_EOF_CH3_INT_CLR 写 1 清除 H264_DMA_IN_SUC_EOF_CH3_INT。(WT)

H264_DMA_IN_ERR_EOF_CH3_INT_CLR 写 1 清除 H264_DMA_IN_ERR_EOF_CH3_INT。(WT)

H264_DMA_IN_DSCR_ERR_CH3_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_ERR_CH3_INT。(WT)

H264_DMA_INFIFO_OVF_L1_CH3_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L1_CH3_INT。(WT)

H264_DMA_INFIFO_UDF_L1_CH3_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L1_CH3_INT。(WT)

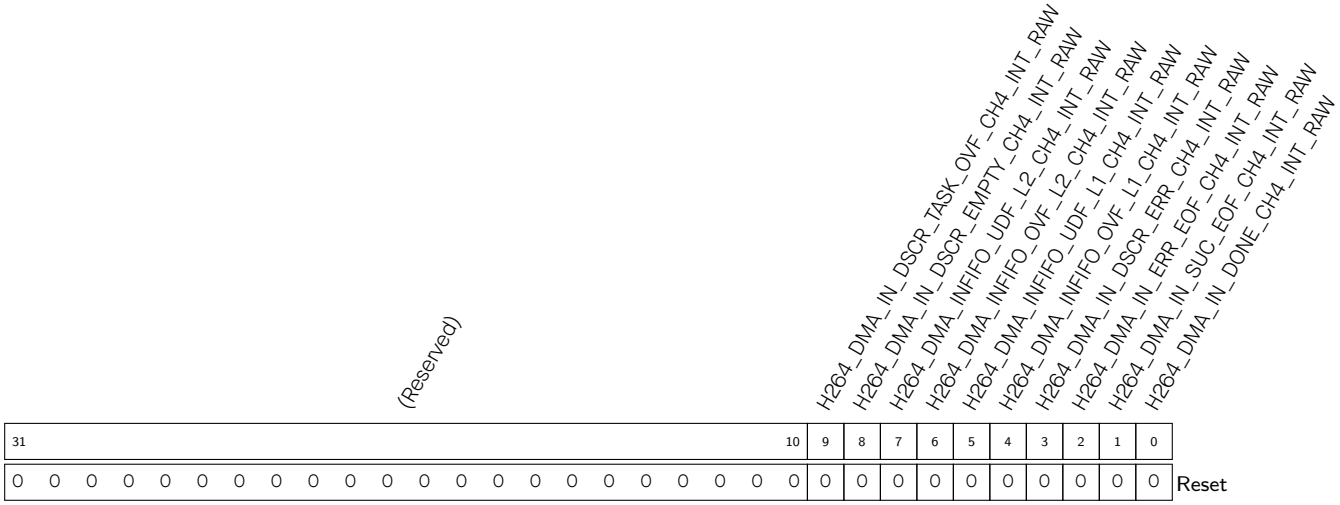
H264_DMA_INFIFO_OVF_L2_CH3_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L2_CH3_INT。(WT)

H264_DMA_INFIFO_UDF_L2_CH3_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L2_CH3_INT。(WT)

H264_DMA_IN_DSCR_EMPTY_CH3_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_EMPTY_CH3_INT。(WT)

H264_DMA_IN_DSCR_TASK_OVF_CH3_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_TASK_OVF_CH3_INT。(WT)

Register 41.209. H264_DMA_IN_INT_RAW_CH4_REG (0x0904)



H264_DMA_IN_DONE_CH4_INT_RAW H264_DMA_IN_DONE_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_SUC_EOF_CH4_INT_RAW H264_DMA_IN_SUC_EOF_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_ERR_EOF_CH4_INT_RAW H264_DMA_IN_ERR_EOF_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_ERR_CH4_INT_RAW H264_DMA_IN_DSCR_ERR_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH4_INT_RAW H264_DMA_INFIFO_OVF_L1_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH4_INT_RAW H264_DMA_INFIFO_UDF_L1_CH4_INT 的原始中断状态。(R/WTC/SS)

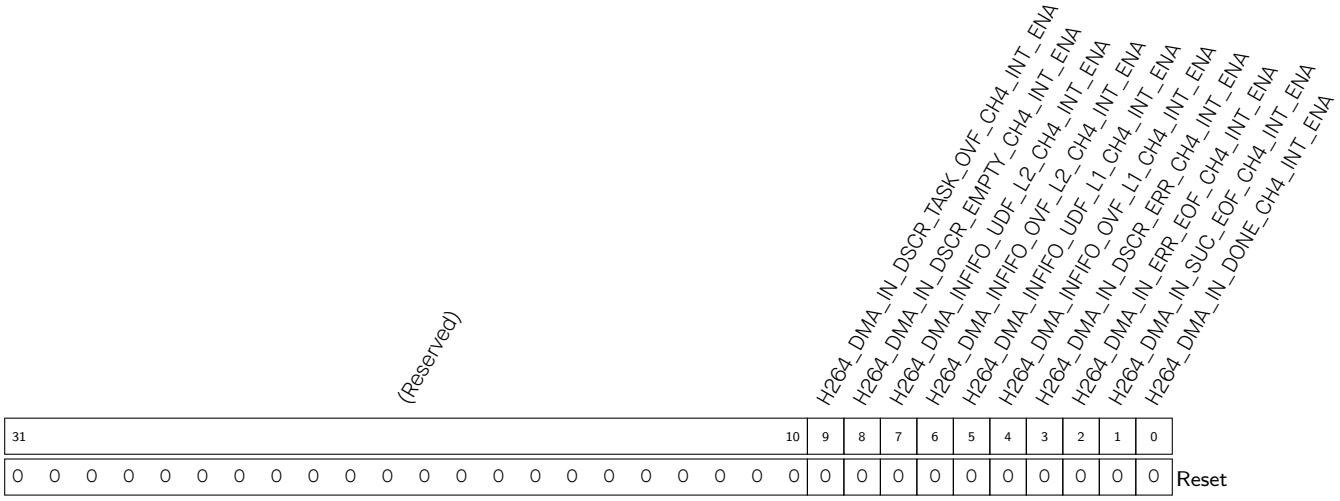
H264_DMA_INFIFO_OVF_L2_CH4_INT_RAW H264_DMA_INFIFO_OVF_L2_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L2_CH4_INT_RAW H264_DMA_INFIFO_UDF_L2_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_RAW H264_DMA_IN_DSCR_EMPTY_CH4_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_RAW H264_DMA_IN_DSCR_TASK_OVF_CH4_INT 的原始中断状态。(R/WTC/SS)

Register 41.210. H264_DMA_IN_INT_ENA_CH4_REG (0x0908)



H264_DMA_IN_DONE_CH4_INT_ENA 写入 1 以启用 H264_DMA_IN_DONE_CH4_INT。(R/W)

H264_DMA_IN_SUC_EOF_CH4_INT_ENA 写入 1 以启用 H264_DMA_IN_SUC_EOF_CH4_INT。(R/W)

H264_DMA_IN_ERR_EOF_CH4_INT_ENA 写入 1 以启用 H264_DMA_IN_ERR_EOF_CH4_INT。(R/W)

H264_DMA_IN_DSCR_ERR_CH4_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_ERR_CH4_INT。(R/W)

H264_DMA_INFIFO_OVF_L1_CH4_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L1_CH4_INT。(R/W)

H264_DMA_INFIFO_UDF_L1_CH4_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L1_CH4_INT。(R/W)

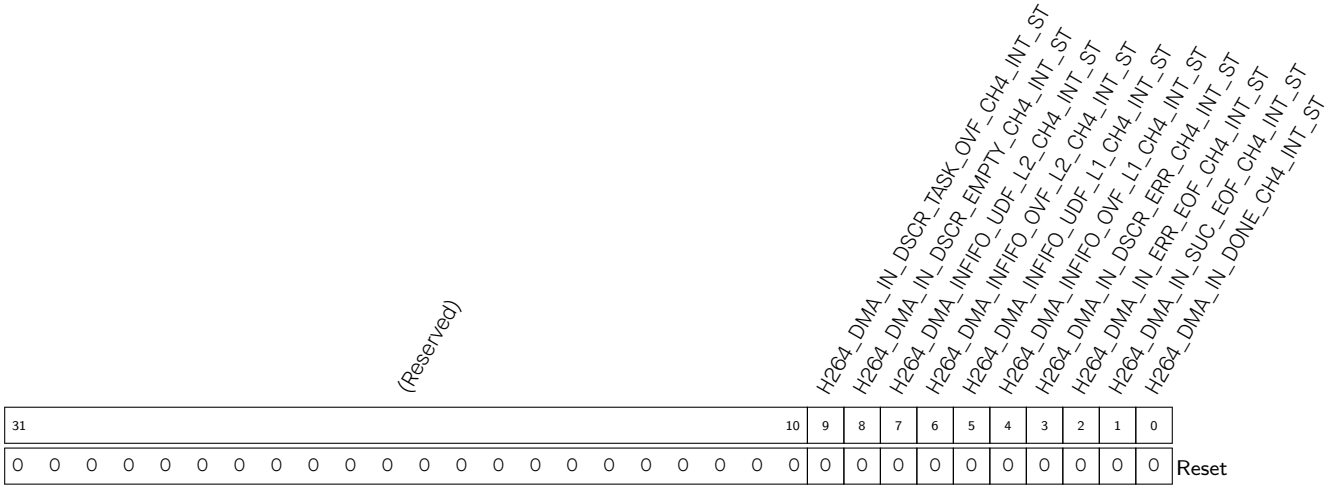
H264_DMA_INFIFO_OVF_L2_CH4_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L2_CH4_INT。(R/W)

H264_DMA_INFIFO_UDF_L2_CH4_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L2_CH4_INT。(R/W)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_EMPTY_CH4_INT。(R/W)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_ENA 写入 1 以启用 H264_DMA_IN_DSCR_TASK_OVF_CH4_INT。(R/W)

Register 41.211. H264_DMA_IN_INT_ST_CH4_REG (0x090C)



H264_DMA_IN_DONE_CH4_INT_ST H264_DMA_IN_DONE_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_SUC_EOF_CH4_INT_ST H264_DMA_IN_SUC_EOF_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_ERR_EOF_CH4_INT_ST H264_DMA_IN_ERR_EOF_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_ERR_CH4_INT_ST H264_DMA_IN_DSCR_ERR_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_OVF_L1_CH4_INT_ST H264_DMA_INFIFO_OVF_L1_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L1_CH4_INT_ST H264_DMA_INFIFO_UDF_L1_CH4_INT 的屏蔽中断状态。(RO)

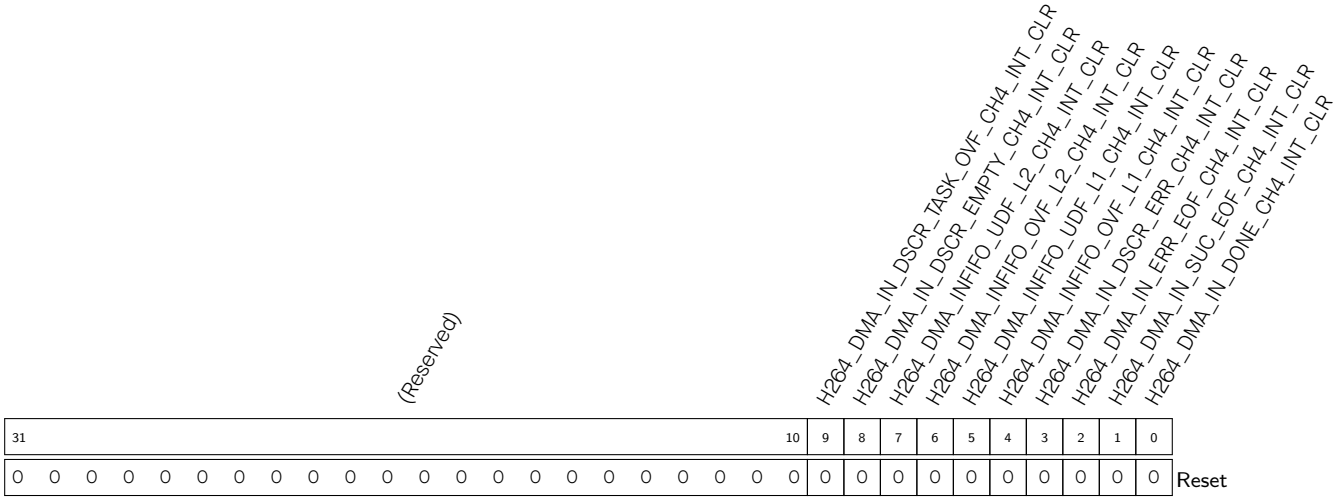
H264_DMA_INFIFO_OVF_L2_CH4_INT_ST H264_DMA_INFIFO_OVF_L2_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L2_CH4_INT_ST H264_DMA_INFIFO_UDF_L2_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_ST H264_DMA_IN_DSCR_EMPTY_CH4_INT 的屏蔽中断状态。(RO)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_ST H264_DMA_IN_DSCR_TASK_OVF_CH4_INT 的屏蔽中断状态。(RO)

Register 41.212. H264_DMA_IN_INT_CLR_CH4_REG (0x0910)



H264_DMA_IN_DONE_CH4_INT_CLR 写 1 清除 H264_DMA_IN_DONE_CH4_INT。(WT)

H264_DMA_IN_SUC_EOF_CH4_INT_CLR 写 1 清除 H264_DMA_IN_SUC_EOF_CH4_INT。(WT)

H264_DMA_IN_ERR_EOF_CH4_INT_CLR 写 1 清除 H264_DMA_IN_ERR_EOF_CH4_INT。(WT)

H264_DMA_IN_DSCR_ERR_CH4_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_ERR_CH4_INT。(WT)

H264_DMA_INFIFO_OVF_L1_CH4_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L1_CH4_INT。(WT)

H264_DMA_INFIFO_UDF_L1_CH4_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L1_CH4_INT。(WT)

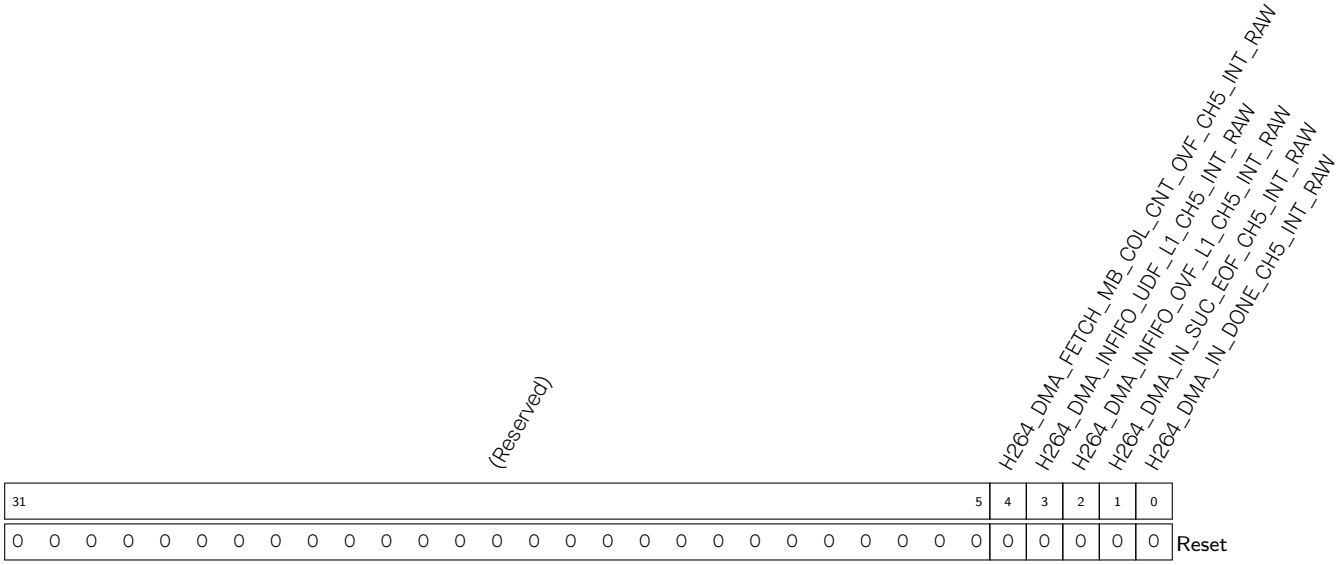
H264_DMA_INFIFO_OVF_L2_CH4_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L2_CH4_INT。(WT)

H264_DMA_INFIFO_UDF_L2_CH4_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L2_CH4_INT。(WT)

H264_DMA_IN_DSCR_EMPTY_CH4_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_EMPTY_CH4_INT。(WT)

H264_DMA_IN_DSCR_TASK_OVF_CH4_INT_CLR 写 1 清除 H264_DMA_IN_DSCR_TASK_OVF_-CH4_INT。(WT)

Register 41.213. H264_DMA_IN_INT_RAW_CH5_REG (0x0A10)



H264_DMA_IN_DONE_CH5_INT_RAW H264_DMA_IN_DONE_CH5_INT 的原始中断状态。(R/WTC/SS)

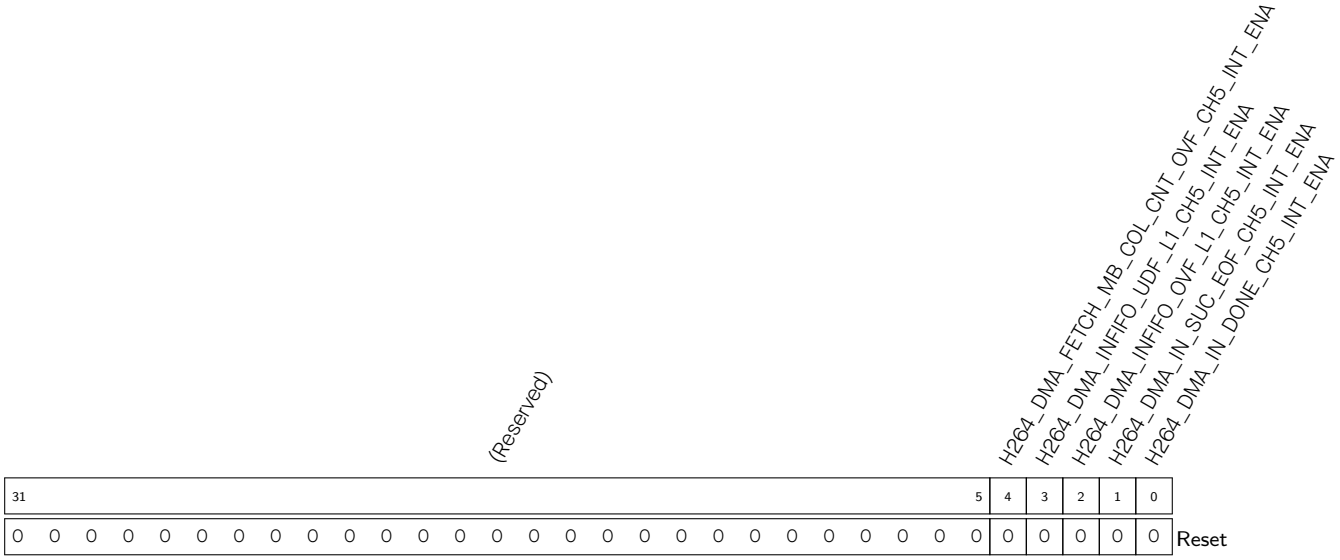
H264_DMA_IN_SUC_EOF_CH5_INT_RAW H264_DMA_IN_SUC_EOF_CH5_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_OVF_L1_CH5_INT_RAW H264_DMA_INFIFO_OVF_L1_CH5_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_INFIFO_UDF_L1_CH5_INT_RAW H264_DMA_INFIFO_UDF_L1_CH5_INT 的原始中断状态。(R/WTC/SS)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_RAW H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT 的原始中断状态。(R/WTC/SS)

Register 41.214. H264_DMA_IN_INT_ENA_CH5_REG (0x0A14)



H264_DMA_IN_DONE_CH5_INT_ENA 写入 1 以启用 H264_DMA_IN_DONE_CH5_INT。(R/W)

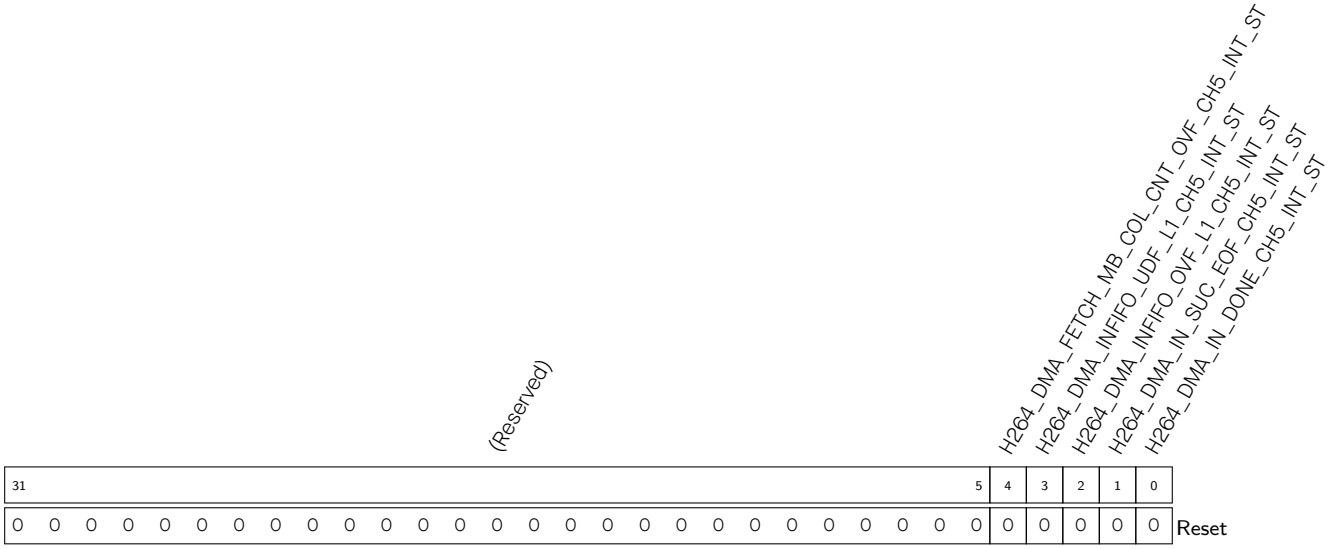
H264_DMA_IN_SUC_EOF_CH5_INT_ENA 写入 1 以启用 H264_DMA_IN_SUC_EOF_CH5_INT。(R/W)

H264_DMA_INFIFO_OVF_L1_CH5_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_OVF_L1_CH5_INT。(R/W)

H264_DMA_INFIFO_UDF_L1_CH5_INT_ENA 写入 1 以启用 H264_DMA_INFIFO_UDF_L1_CH5_INT。(R/W)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_ENA 写入 1 以启用 H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT。(R/W)

Register 41.215. H264_DMA_IN_INT_ST_CH5_REG (0x0A18)



H264_DMA_IN_DONE_CH5_INT_ST H264_DMA_IN_DONE_CH5_INT 的屏蔽中断状态。(RO)

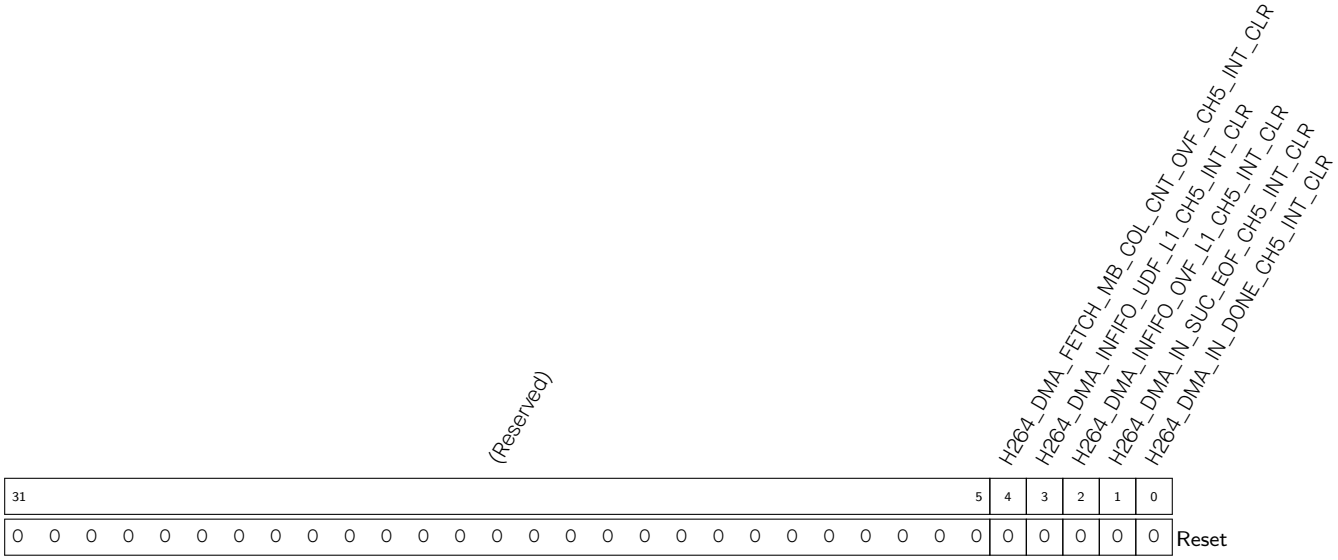
H264_DMA_IN_SUC_EOF_CH5_INT_ST H264_DMA_IN_SUC_EOF_CH5_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_OVF_L1_CH5_INT_ST H264_DMA_INFIFO_OVF_L1_CH5_INT 的屏蔽中断状态。(RO)

H264_DMA_INFIFO_UDF_L1_CH5_INT_ST H264_DMA_INFIFO_UDF_L1_CH5_INT 的屏蔽中断状态。(RO)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_ST H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT 的屏蔽中断状态。(RO)

Register 41.216. H264_DMA_IN_INT_CLR_CH5_REG (0x0A1C)



H264_DMA_IN_DONE_CH5_INT_CLR 写 1 清除 H264_DMA_IN_DONE_CH5_INT。(WT)

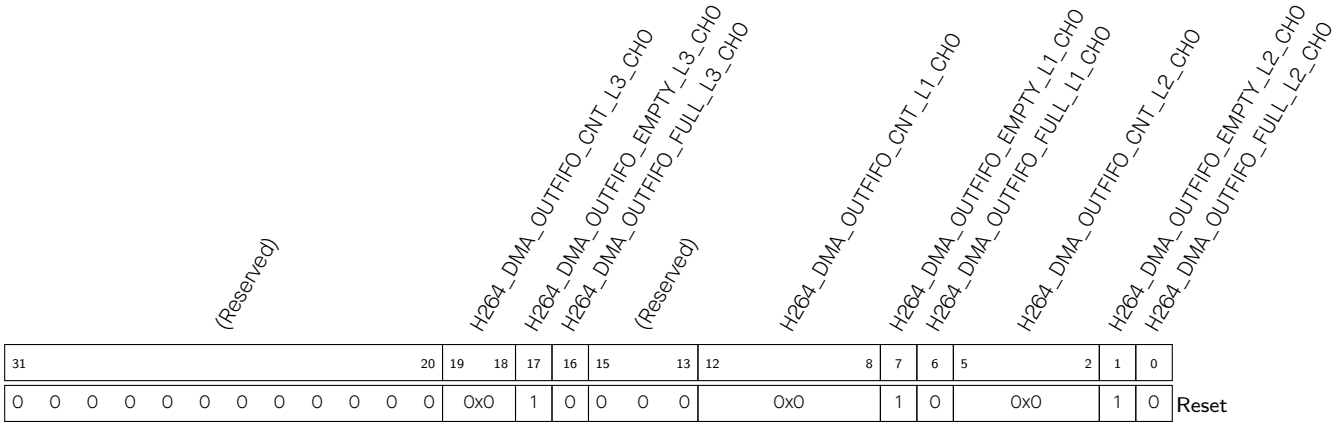
H264_DMA_IN_SUC_EOF_CH5_INT_CLR 写 1 清除 H264_DMA_IN_SUC_EOF_CH5_INT。(WT)

H264_DMA_INFIFO_OVF_L1_CH5_INT_CLR 写 1 清除 H264_DMA_INFIFO_OVF_L1_CH5_INT。(WT)

H264_DMA_INFIFO_UDF_L1_CH5_INT_CLR 写 1 清除 H264_DMA_INFIFO_UDF_L1_CH5_INT。(WT)

H264_DMA_FETCH_MB_COL_CNT_OVF_CH5_INT_CLR 写 1 清除 H264_DMA_FETCH_MB_COL_CNT_OVF_CH5。(WT)

Register 41.217. H264_DMA_OUTFIFO_STATUS_CH0_REG (0x0014)



H264_DMA_OUTFIFO_FULL_L2_CH0 表示 TX 通道 0 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH0 表示 TX 通道 0 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH0 表示 TX 通道 0 的 l2 FIFO 中的数据量。测量单位：16 字节。

(RO)

H264_DMA_OUTFIFO_FULL_L1_CH0 表示 TX 通道 0 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH0 表示 TX 通道 0 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH0 表示 TX 通道 0 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_OUTFIFO_FULL_L3_CH0 表示 TX 通道 0 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH0 表示 TX 通道 0 的 l3 FIFO 是否为空。

0：不为空

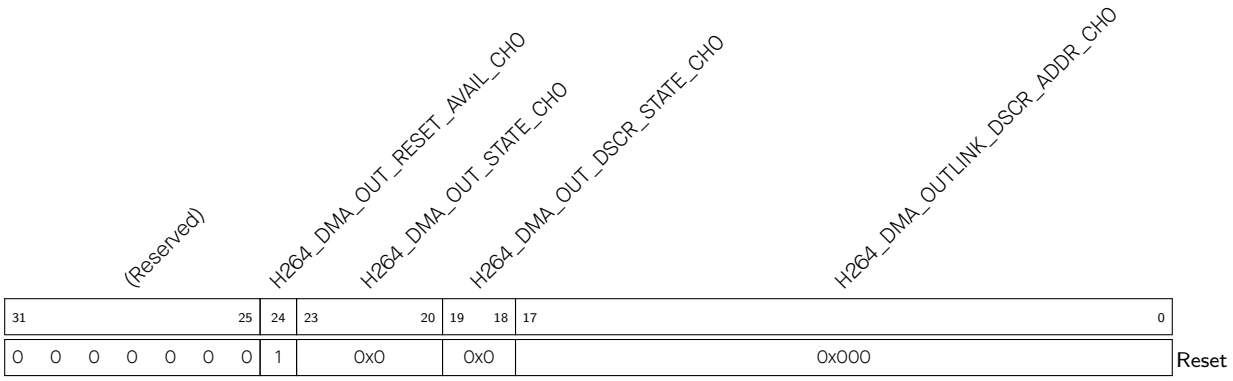
1：空

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH0 表示 TX 通道 0 的 l3 FIFO 中的数据量。测量单位：8 字节。

(RO)

Register 41.218. H264_DMA_OUT_STATE_CH0_REG (0x0024)



H264_DMA_OUTLINK_DSCR_ADDR_CH0 表示 TX 通道 0 的当前 outlink 描述符的地址。(RO)

H264_DMA_OUT_DSCR_STATE_CH0 表示 TX 通道 0 的描述符状态机的当前状态。(RO)

H264_DMA_OUT_STATE_CH0 表示 TX 通道 0 的当前控制模块状态机状态。(RO)

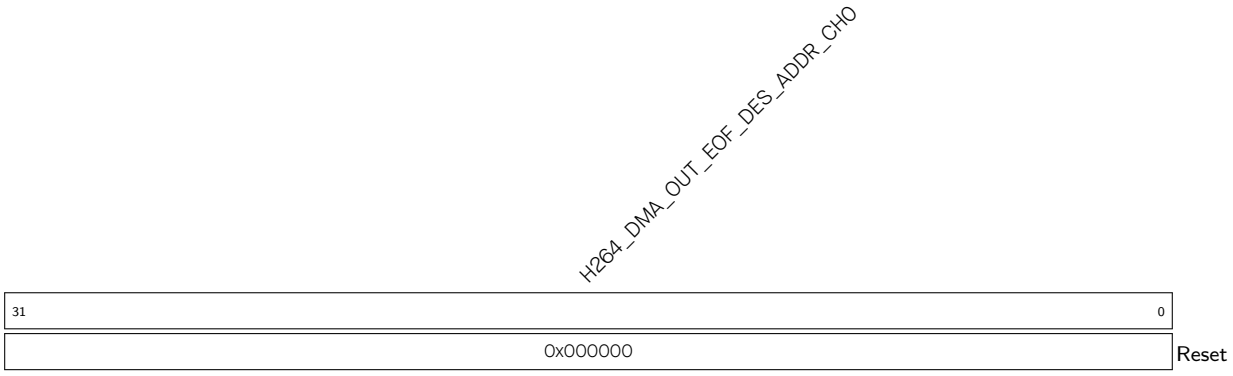
H264_DMA_OUT_RESET_AVAIL_CH0 表示是否可以安全地重置通道。

0：不安全

1：安全

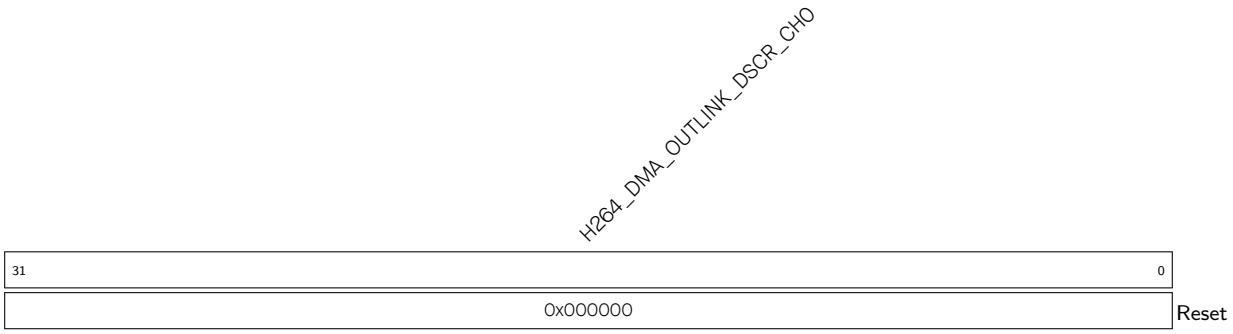
(RO)

Register 41.219. H264_DMA_OUT_EOF_DES_ADDR_CH0_REG (0x0028)



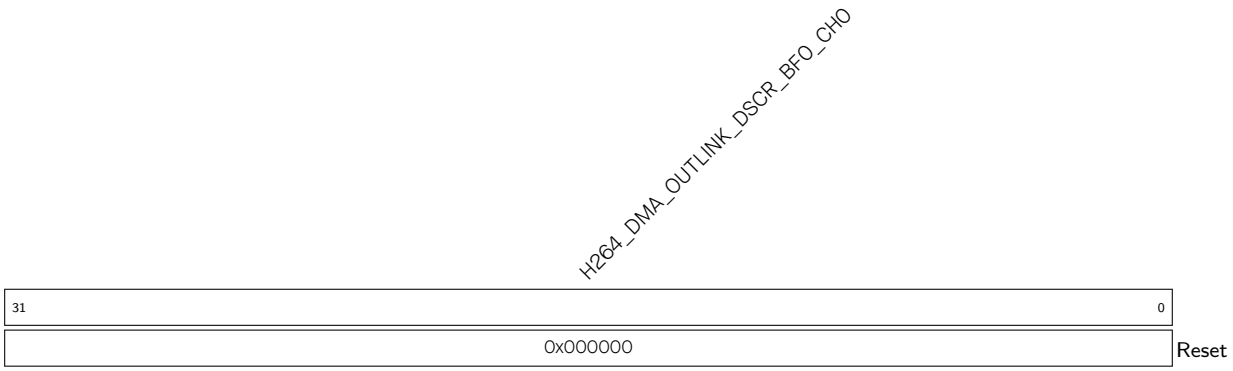
H264_DMA_OUT_EOF_DES_ADDR_CH0 表示当此描述符中的 EOF 位为 1 时，outlink 描述符的地址。(RO)

Register 41.220. H264_DMA_OUT_DSCR_CH0_REG (0x002C)



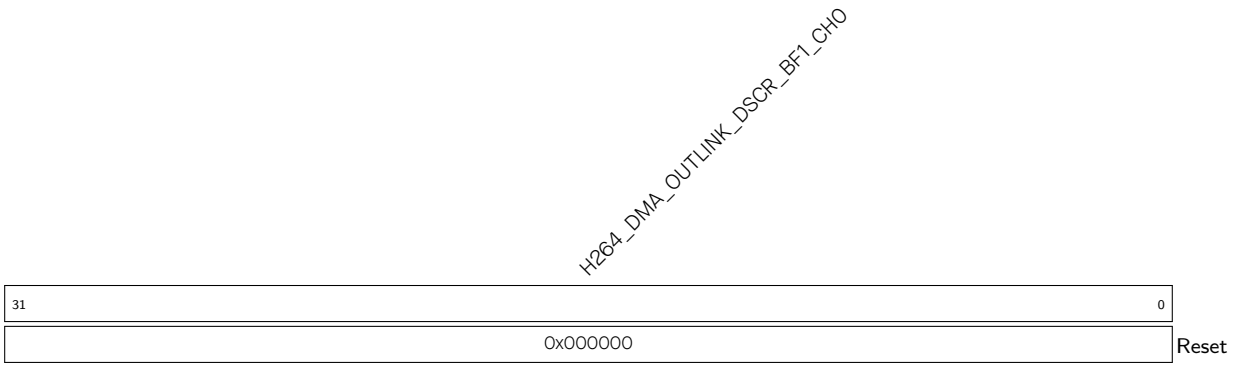
H264_DMA_OUTLINK_DSCR_CH0 表示由当前预读描述符指向的下一个描述符 y+1 的地址。(RO)

Register 41.221. H264_DMA_OUT_DSCR_BF0_CH0_REG (0x0030)



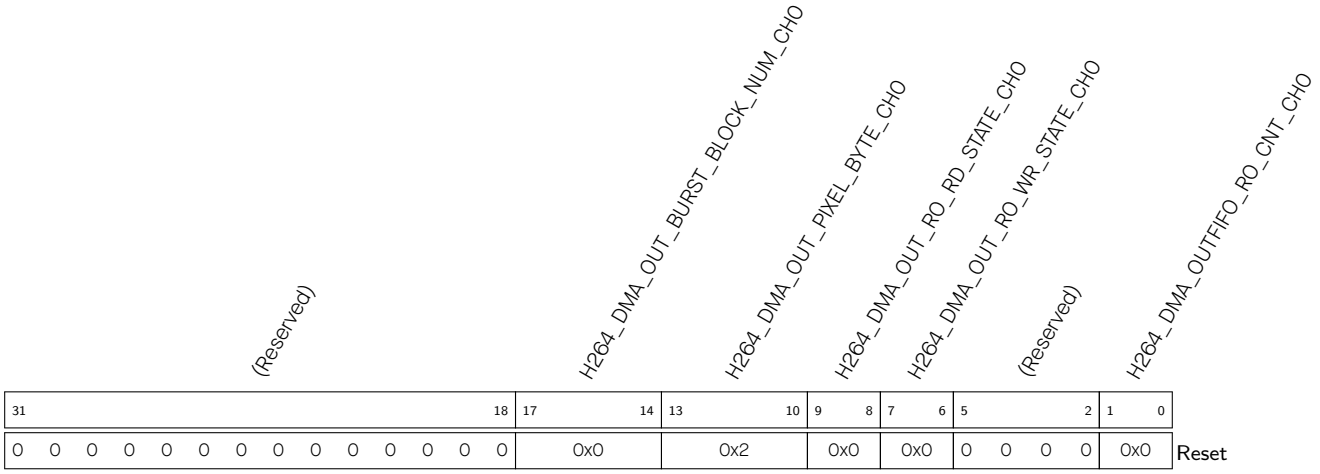
H264_DMA_OUTLINK_DSCR_BF0_CH0 表示预读的当前描述符 y 的地址。(RO)

Register 41.222. H264_DMA_OUT_DSCR_BF1_CH0_REG (0x0034)



H264_DMA_OUTLINK_DSCR_BF1_CH0 表示预读的前一个描述符 y-1 的地址。(RO)

Register 41.223. H264_DMA_OUT_RO_STATUS_CH0_REG (0x0040)



H264_DMA_OUTFIFO_RO_CNT_CH0 表示通道 0 的重排序 TX FIFO 中的数据的 8 字节数量。(RO)

H264_DMA_OUT_RO_WR_STATE_CH0 表示重排序 RAM 的写入状态。(RO)

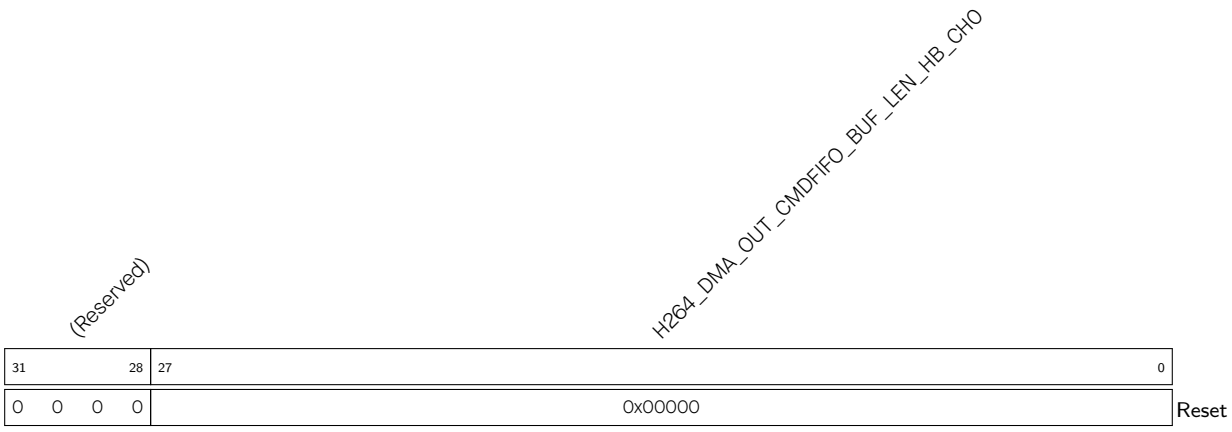
H264_DMA_OUT_RO_RD_STATE_CH0 表示重排序 RAM 的读取状态。(RO)

H264_DMA_OUT_PIXEL_BYTE_CH0 表示 TX 通道 0 中一个像素包含的字节数。

0：1 字节
1：1.5 字节
2：2 字节
3：2.5 字节
4：3 字节
5：4 字节
(RO)

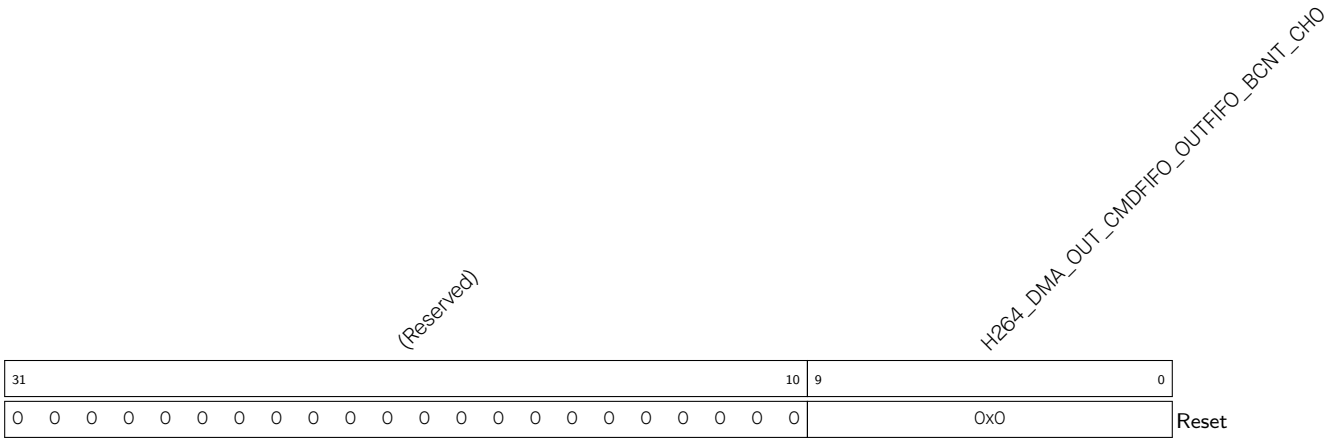
H264_DMA_OUT_BURST_BLOCK_NUM_CH0 保留。(RO)

Register 41.224. H264_DMA_OUT_BUF_LEN_CH0_REG (0x0070)



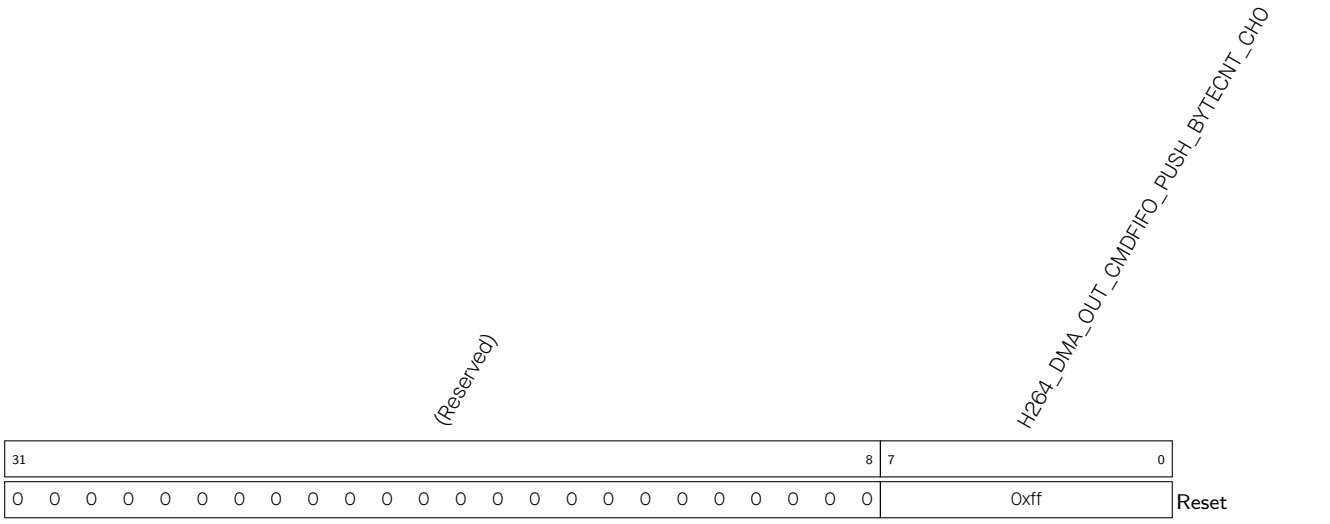
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH0 表示剩余的请求数据。(RO)

Register 41.225. H264_DMA_OUT_FIFO_BCNT_CH0_REG (0x0074)



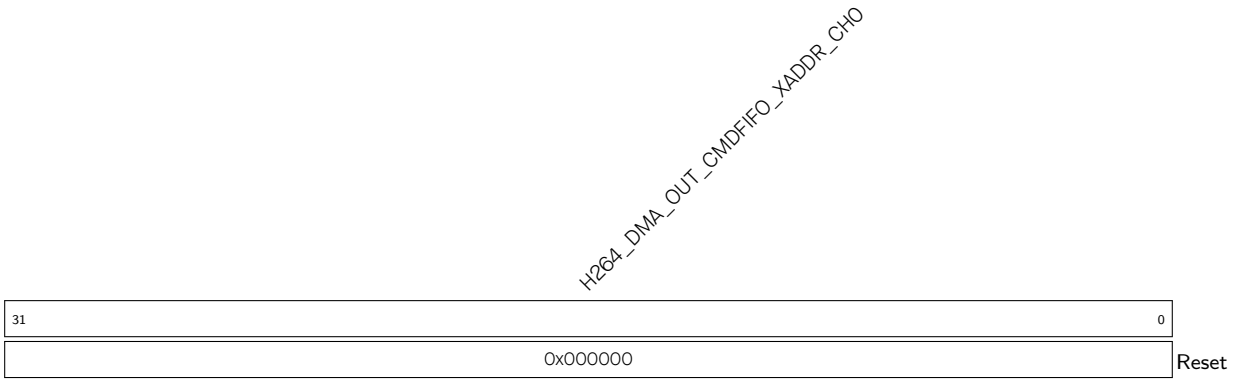
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH0 表示已在 FIFO 中使用的数据空间。(RO)

Register 41.226. H264_DMA_OUT_PUSH_BYTECNT_CH0_REG (0x0078)



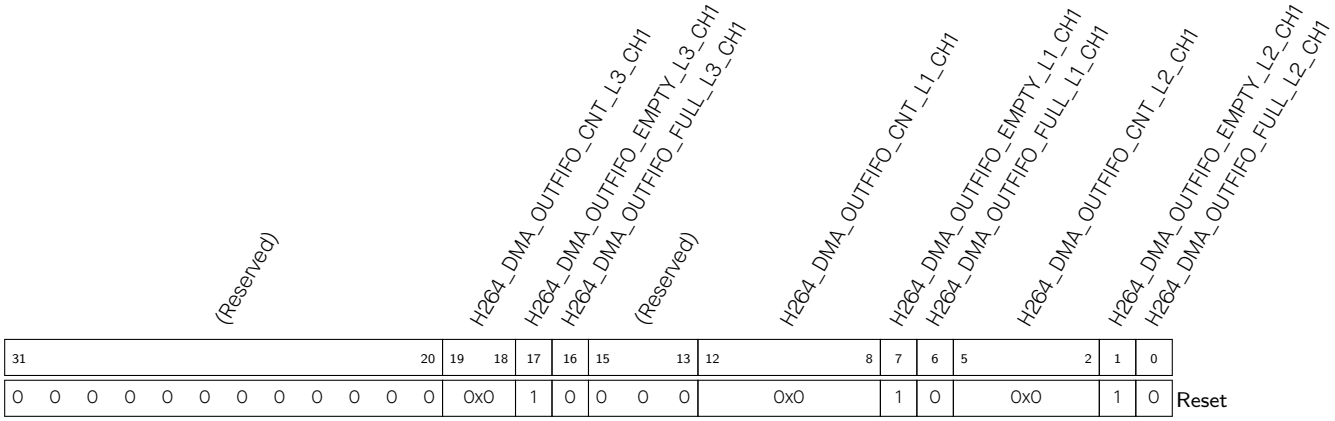
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH0 表示当前 AXI 请求中的数据量。(RO)

Register 41.227. H264_DMA_OUT_XADDR_CH0_REG (0x007C)



H264_DMA_OUT_CMDFIFO_XADDR_CH0 表示当前 AXI 请求的地址。(RO)

Register 41.228. H264_DMA_OUTFIFO_STATUS_CH1_REG (0x0114)



H264_DMA_OUTFIFO_FULL_L2_CH1 表示 TX 通道 1 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH1 表示 TX 通道 1 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH1 表示 TX 通道 1 的 l2 FIFO 中的数据量。测量单位：16 字节。

(RO)

H264_DMA_OUTFIFO_FULL_L1_CH1 表示 TX 通道 1 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH1 表示 TX 通道 1 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH1 表示 TX 通道 1 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_OUTFIFO_FULL_L3_CH1 表示 TX 通道 1 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH1 表示 TX 通道 1 的 l3 FIFO 是否为空。

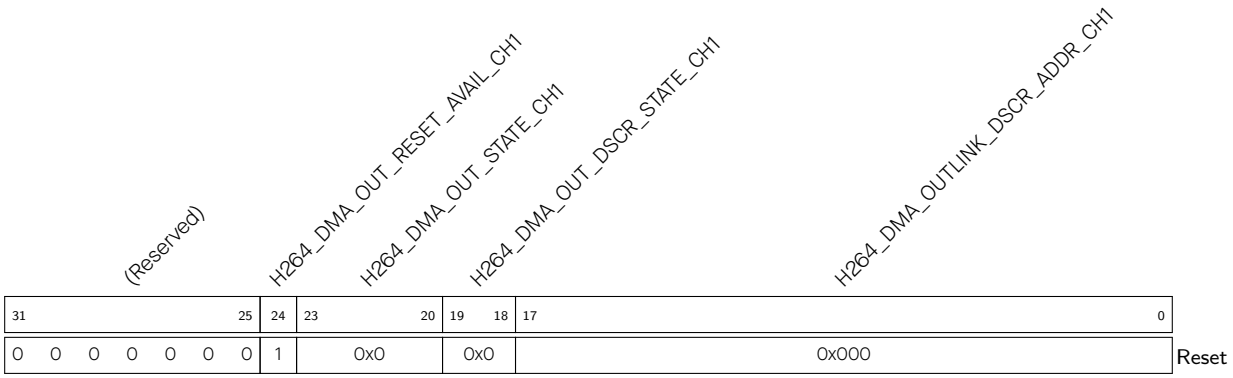
0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH1 表示 TX 通道 1 的 l3 FIFO 中的数据量。测量单位：8 字节。(RO)

Register 41.229. H264_DMA_OUT_STATE_CH1_REG (0x0124)



H264_DMA_OUTLINK_DSCR_ADDR_CH1 表示 TX 通道 1 的当前 outlink 描述符的地址。(RO)

H264_DMA_OUT_DSCR_STATE_CH1 表示 TX 通道 1 的描述符状态机的当前状态。(RO)

H264_DMA_OUT_STATE_CH1 表示 TX 通道 1 的当前控制模块状态机状态。(RO)

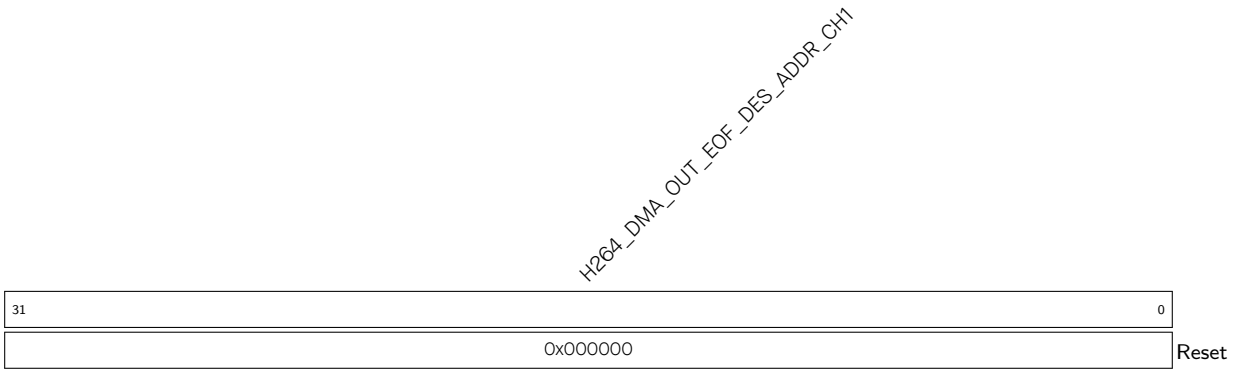
H264_DMA_OUT_RESET_AVAIL_CH1 表示是否可以安全地重置通道。

0：不安全

1：安全

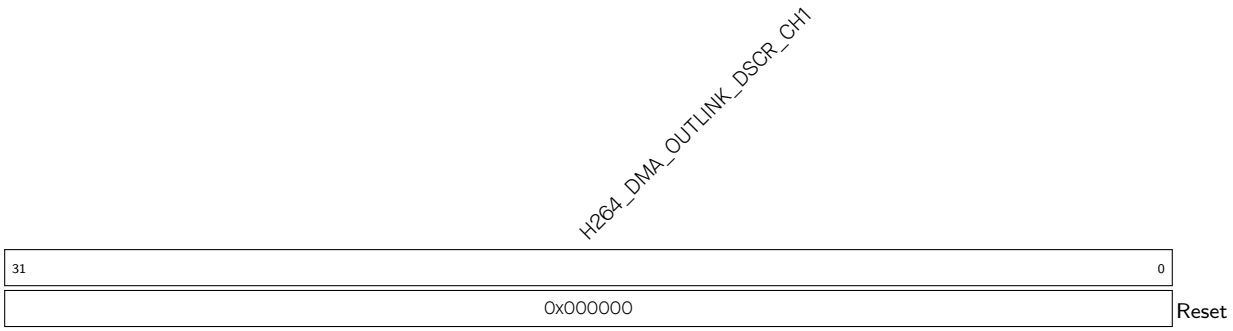
(RO)

Register 41.230. H264_DMA_OUT_EOF_DES_ADDR_CH1_REG (0x0128)



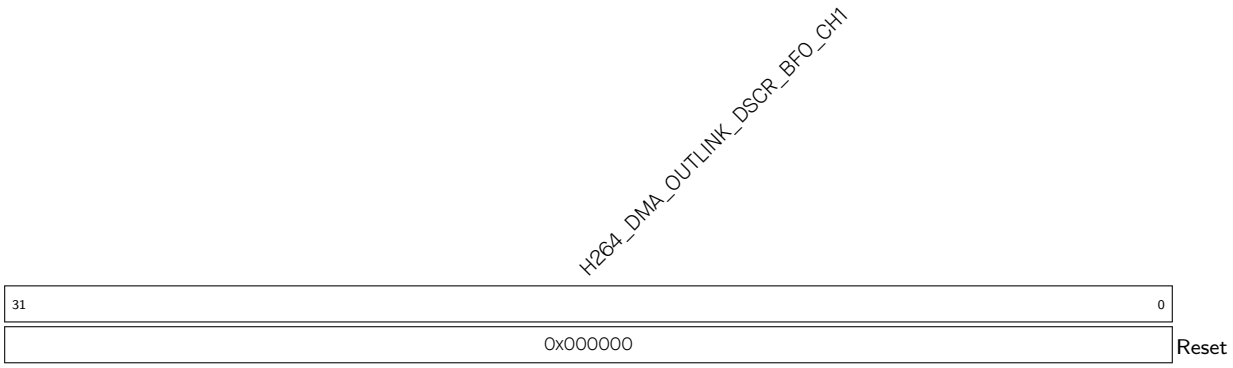
H264_DMA_OUT_EOF_DES_ADDR_CH1 表示当此描述符中的 EOF 位为 1 时，outlink 描述符的地址。(RO)

Register 41.231. H264_DMA_OUT_DSCR_CH1_REG (0x012C)



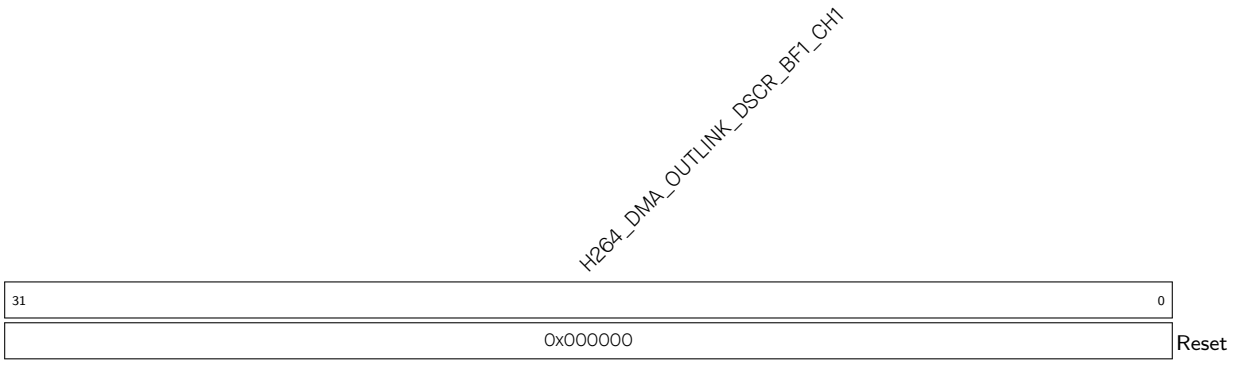
H264_DMA_OUTLINK_DSCR_CH1 表示由当前预读描述符指向的下一个描述符 y+1 的地址。(RO)

Register 41.232. H264_DMA_OUT_DSCR_BF0_CH1_REG (0x0130)



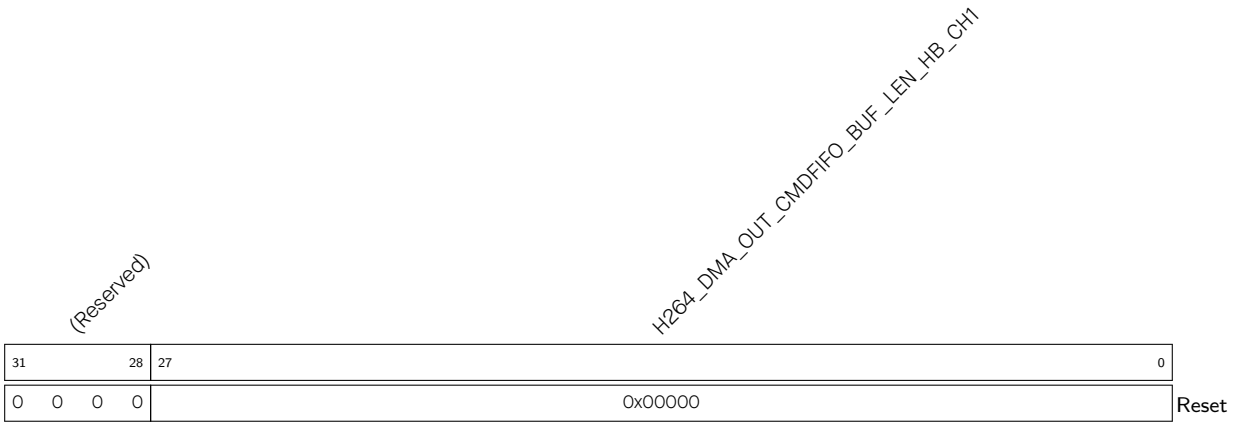
H264_DMA_OUTLINK_DSCR_BF0_CH1 表示预读的当前描述符 y 的地址。(RO)

Register 41.233. H264_DMA_OUT_DSCR_BF1_CH1_REG (0x0134)



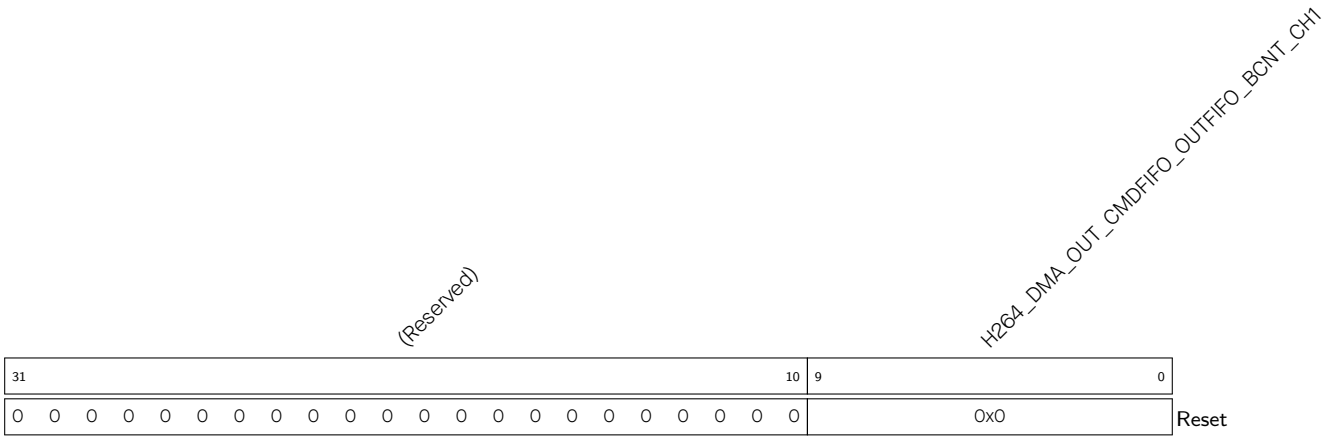
H264_DMA_OUTLINK_DSCR_BF1_CH1 表示预读的前一个描述符 y-1 的地址。(RO)

Register 41.234. H264_DMA_OUT_BUF_LEN_CH1_REG (0x0170)



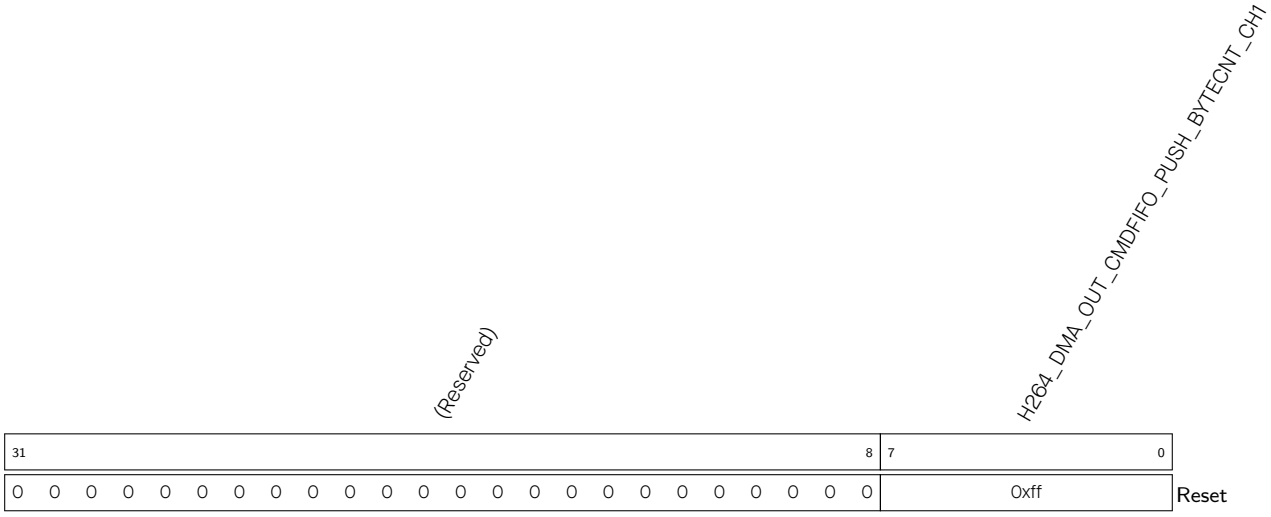
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH1 表示剩余的请求数据。(RO)

Register 41.235. H264_DMA_OUT_FIFO_BCNT_CH1_REG (0x0174)



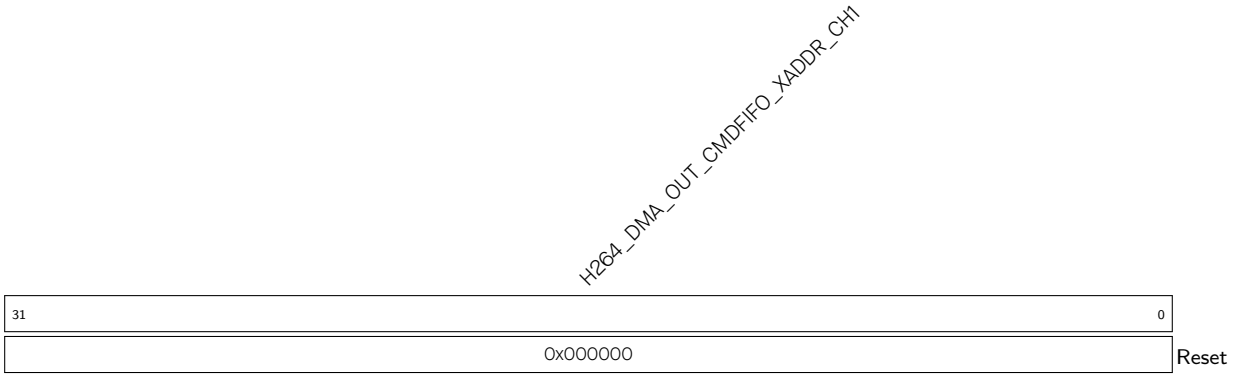
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH1 表示已在 FIFO 中使用的数据空间。(RO)

Register 41.236. H264_DMA_OUT_PUSH_BYTECNT_CH1_REG (0x0178)



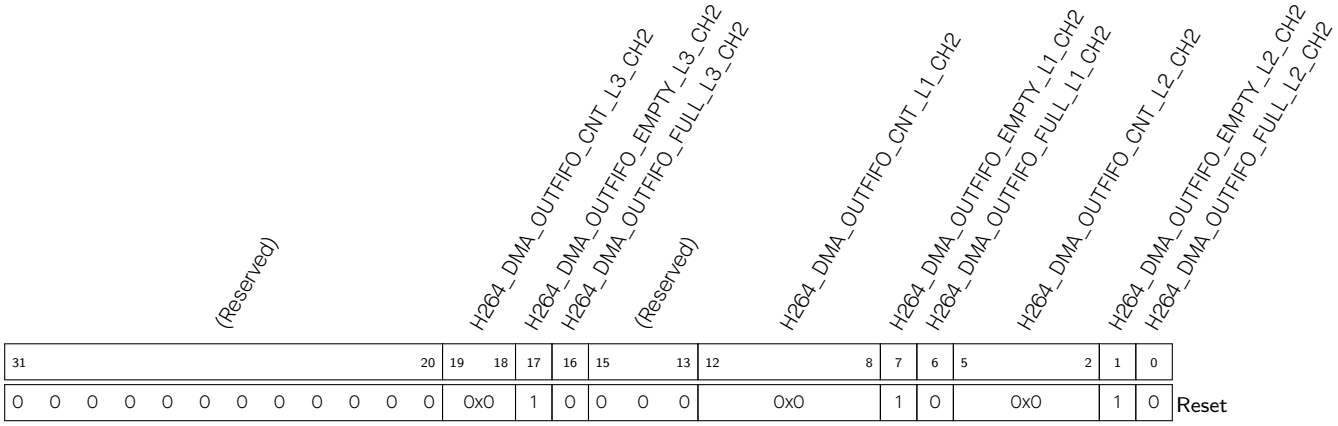
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH1 表示当前 AXI 请求中的数据量。(RO)

Register 41.237. H264_DMA_OUT_XADDR_CH1_REG (0x017C)



H264_DMA_OUT_CMDFIFO_XADDR_CH1 表示当前 AXI 请求的地址。(RO)

Register 41.238. H264_DMA_OUTFIFO_STATUS_CH2_REG (0x0214)



H264_DMA_OUTFIFO_FULL_L2_CH2 表示 TX 通道 2 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH2 表示 TX 通道 2 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH2 表示 TX 通道 2 的 l2 FIFO 中的数据量。测量单位：16 字节。

(RO)

H264_DMA_OUTFIFO_FULL_L1_CH2 表示 TX 通道 2 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH2 表示 TX 通道 2 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH2 表示 TX 通道 2 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_OUTFIFO_FULL_L3_CH2 表示 TX 通道 2 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH2 表示 TX 通道 2 的 l3 FIFO 是否为空。

0：不为空

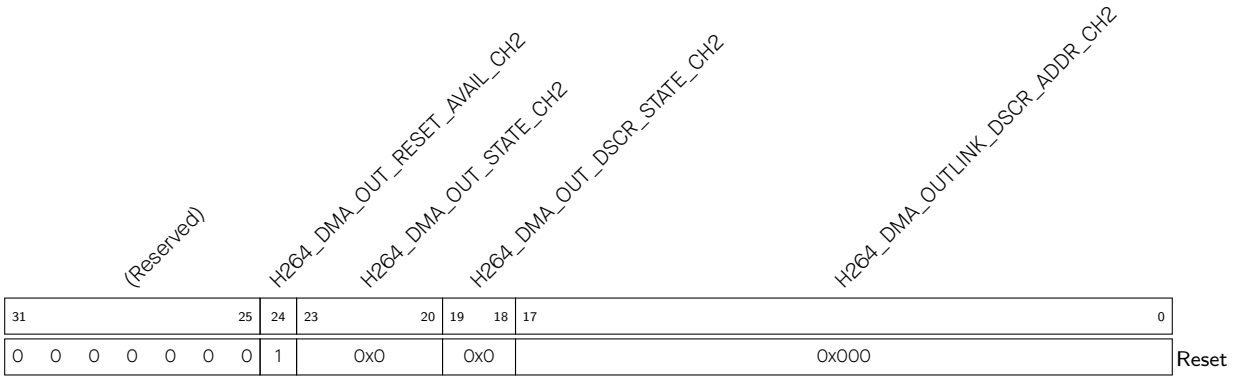
1：空

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH2 表示 TX 通道 2 的 l3 FIFO 中的数据量。测量单位：8 字节。

(RO)

Register 41.239. H264_DMA_OUT_STATE_CH2_REG (0x0224)



H264_DMA_OUTLINK_DSCR_ADDR_CH2 表示 TX 通道 2 的当前 outlink 描述符的地址。(RO)

H264_DMA_OUT_DSCR_STATE_CH2 表示 TX 通道 2 的描述符状态机的当前状态。(RO)

H264_DMA_OUT_STATE_CH2 表示 TX 通道 2 的当前控制模块状态机状态。(RO)

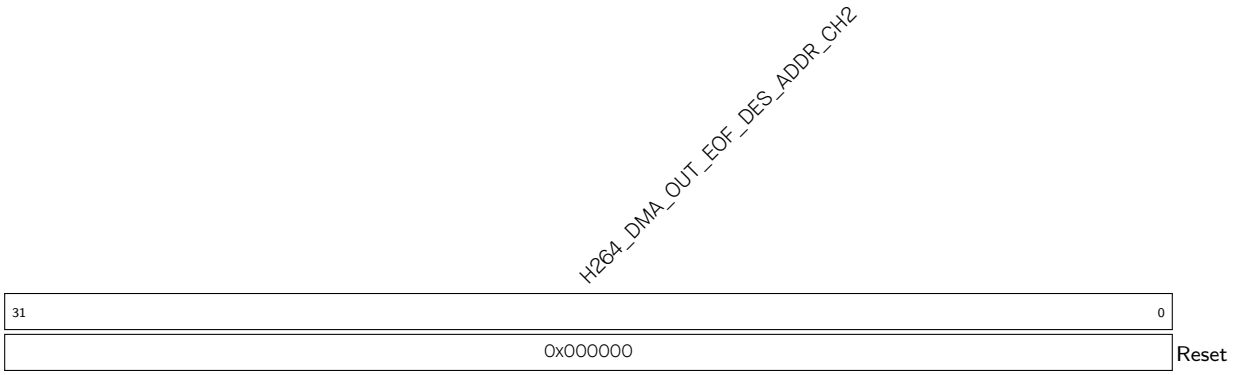
H264_DMA_OUT_RESET_AVAIL_CH2 表示是否可以安全地重置通道。

0：不安全

1：安全

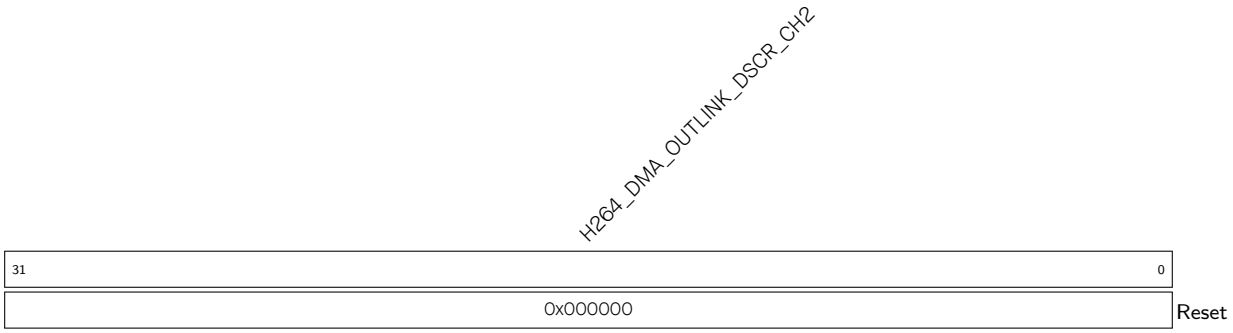
(RO)

Register 41.240. H264_DMA_OUT_EOF_DES_ADDR_CH2_REG (0x0228)



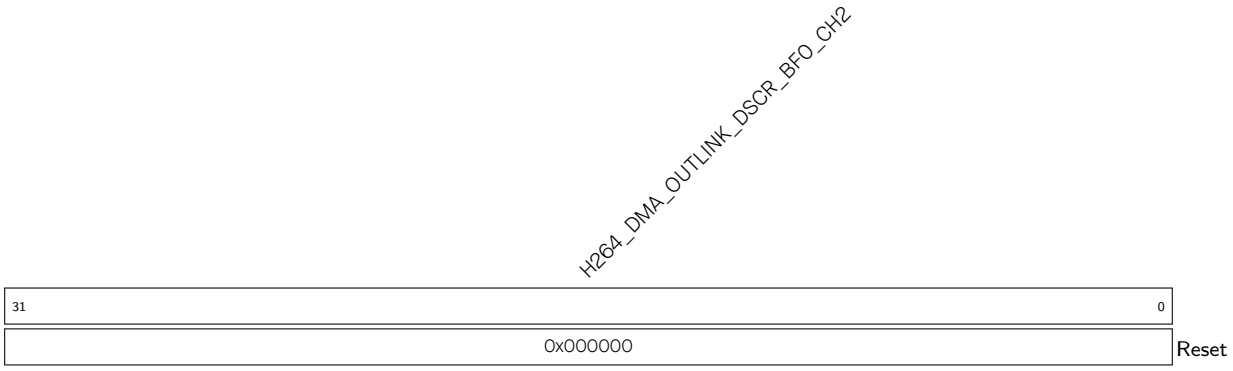
H264_DMA_OUT_EOF_DES_ADDR_CH2 当此描述符中的 EOF 位为 1 时，表示 inlink 描述符的地址。(RO)

Register 41.241. H264_DMA_OUT_DSCR_CH2_REG (0x022C)



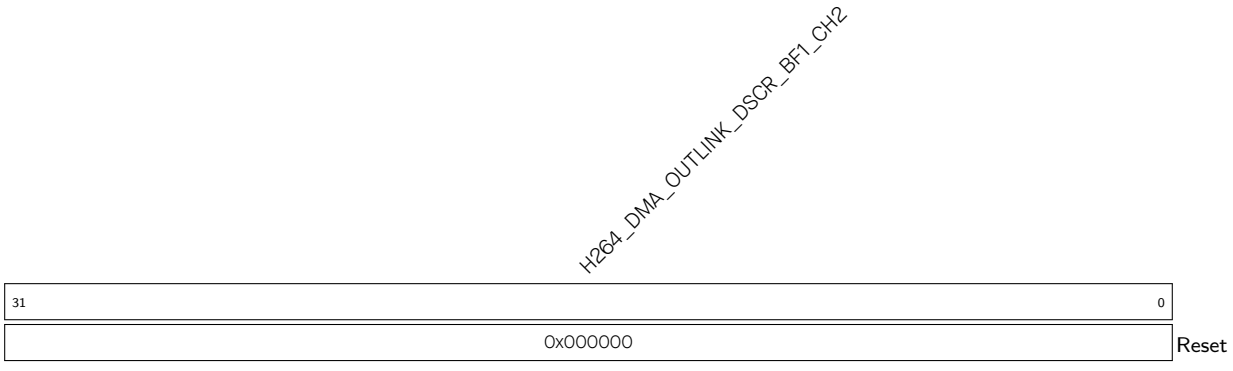
H264_DMA_OUTLINK_DSCR_CH2 表示由当前预读描述符指向的下一个描述符 y+1 的地址。(RO)

Register 41.242. H264_DMA_OUT_DSCR_BF0_CH2_REG (0x0230)



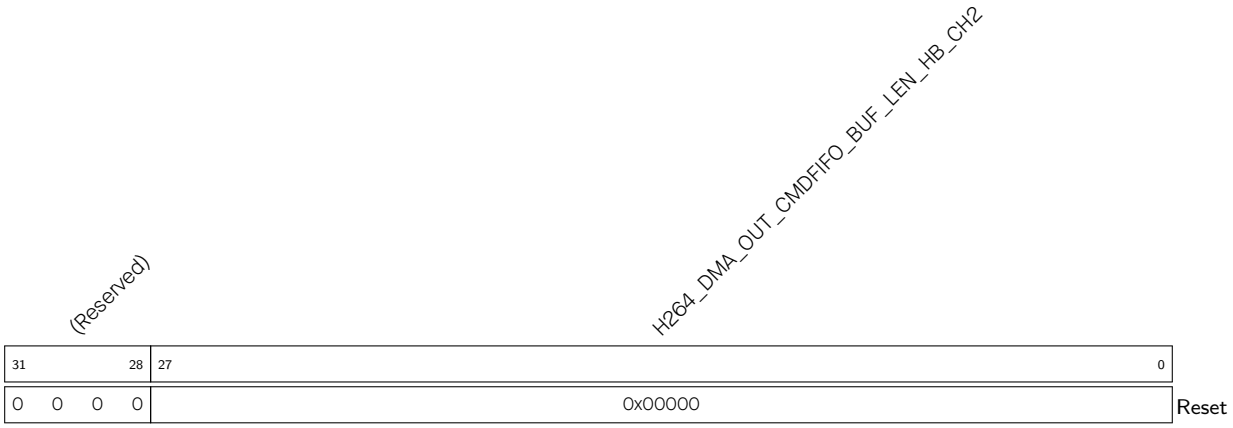
H264_DMA_OUTLINK_DSCR_BF0_CH2 表示预读的当前描述符 y 的地址。(RO)

Register 41.243. H264_DMA_OUT_DSCR_BF1_CH2_REG (0x0234)



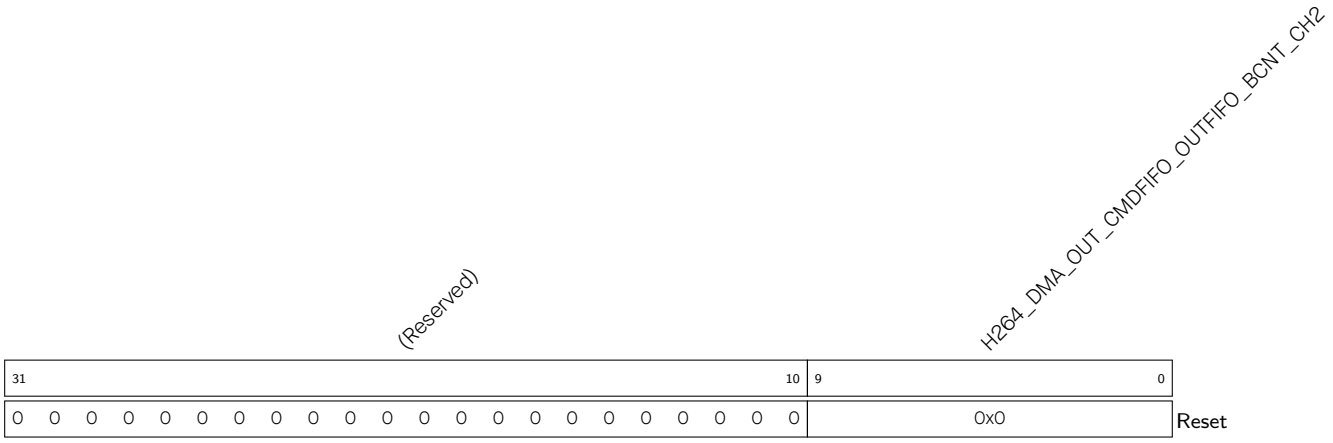
H264_DMA_OUTLINK_DSCR_BF1_CH2 表示预读的前一个描述符 y-1 的地址。(RO)

Register 41.244. H264_DMA_OUT_BUF_LEN_CH2_REG (0x0270)



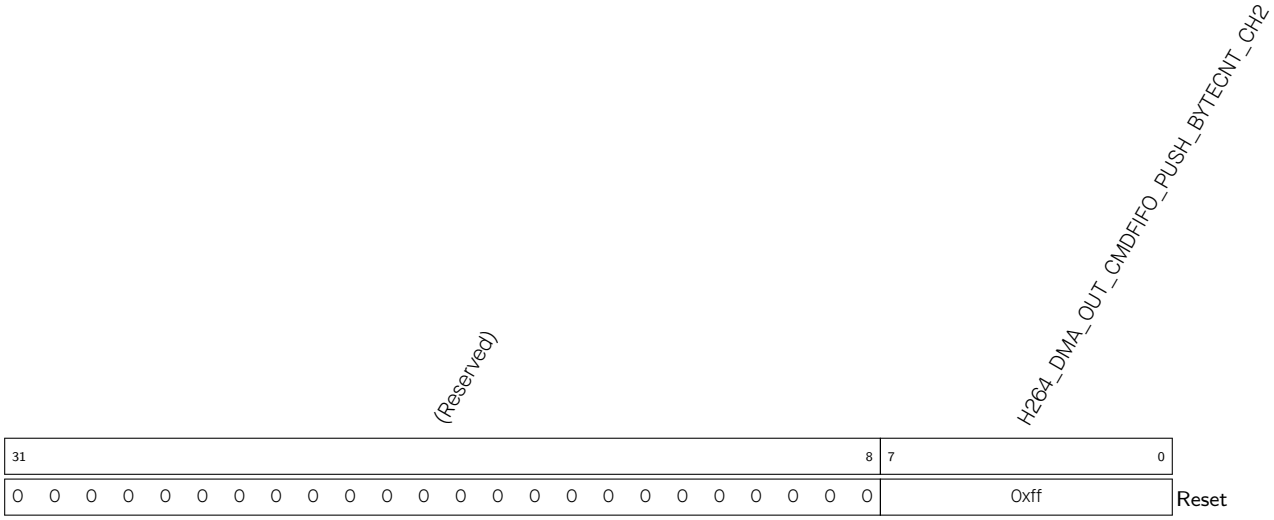
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH2 表示剩余的请求数据。(RO)

Register 41.245. H264_DMA_OUT_FIFO_BCNT_CH2_REG (0x0274)



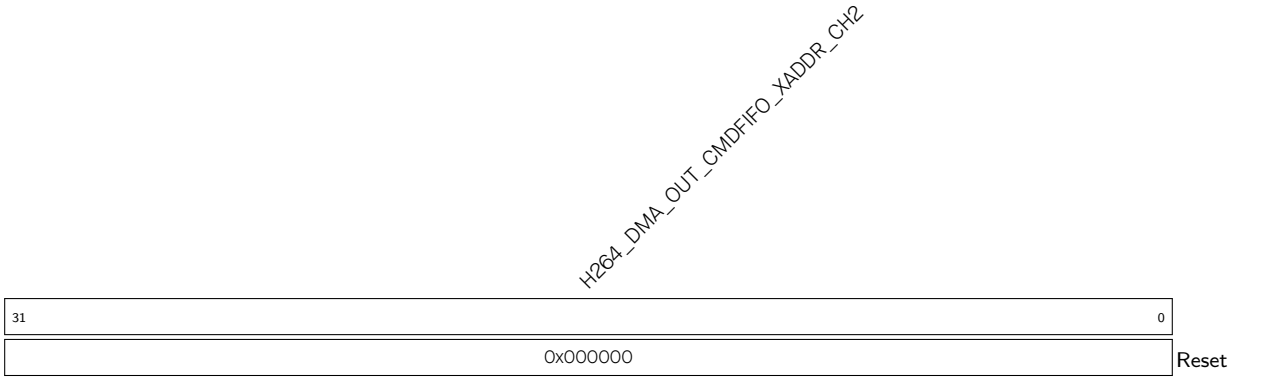
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH2 表示已在 FIFO 中使用的数据空间。(RO)

Register 41.246. H264_DMA_OUT_PUSH_BYTECNT_CH2_REG (0x0278)



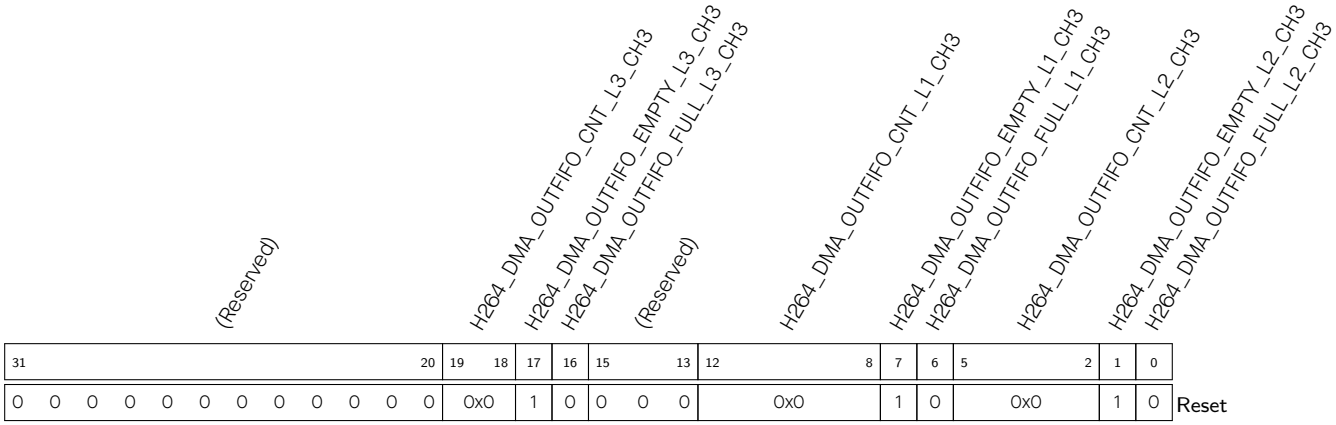
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH2 表示当前 AXI 请求中的数据量。(RO)

Register 41.247. H264_DMA_OUT_XADDR_CH2_REG (0x027C)



H264_DMA_OUT_CMDFIFO_XADDR_CH2 表示当前 AXI 请求的地址。(RO)

Register 41.248. H264_DMA_OUTFIFO_STATUS_CH3_REG (0x0314)



H264_DMA_OUTFIFO_FULL_L2_CH3 表示 TX 通道 3 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH3 表示 TX 通道 3 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH3 表示 TX 通道 3 的 l2 FIFO 中的数据量。测量单位：16 字节。

(RO)

H264_DMA_OUTFIFO_FULL_L1_CH3 表示 TX 通道 3 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH3 表示 TX 通道 3 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH3 表示 TX 通道 3 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_OUTFIFO_FULL_L3_CH3 表示 TX 通道 3 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH3 表示 TX 通道 3 的 l3 FIFO 是否为空。

0：不为空

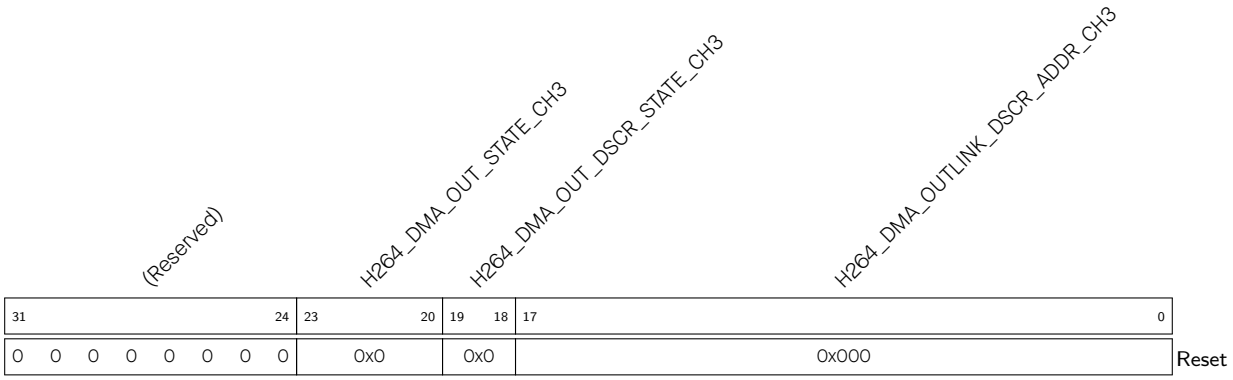
1：空

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH3 表示 TX 通道 3 的 l3 FIFO 中的数据量。测量单位：8 字节。

(RO)

Register 41.249. H264_DMA_OUT_STATE_CH3_REG (0x0324)

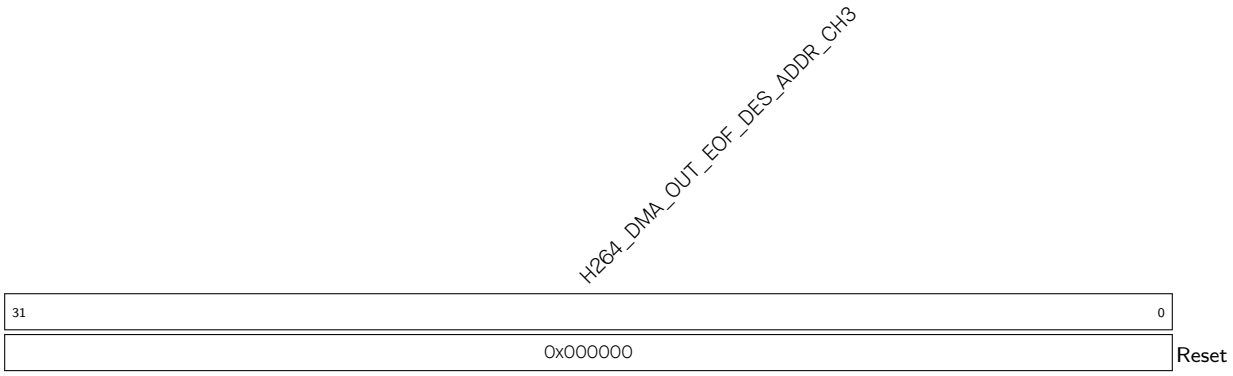


H264_DMA_OUTLINK_DSCR_ADDR_CH3 表示 TX 通道 3 的当前 outlink 描述符的地址。(RO)

H264_DMA_OUT_DSCR_STATE_CH3 表示 TX 通道 3 的描述符状态机的当前状态。(RO)

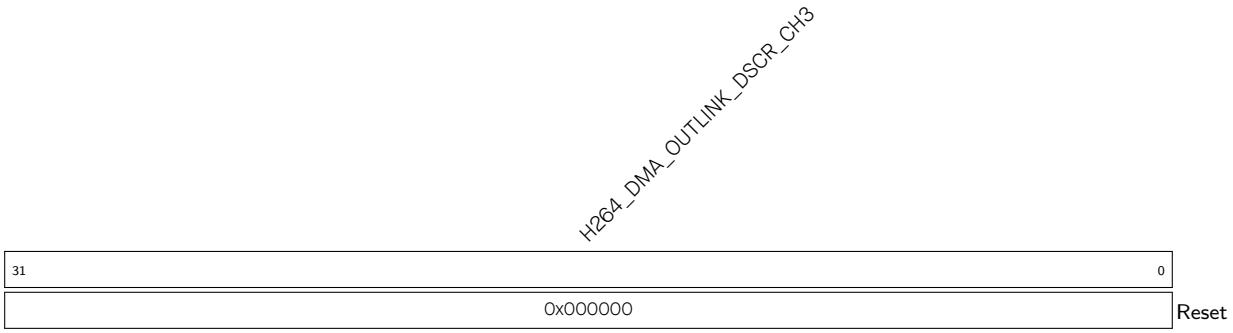
H264_DMA_OUT_STATE_CH3 表示 TX 通道 3 的当前控制模块状态机状态。(RO)

Register 41.250. H264_DMA_OUT_EOF_DES_ADDR_CH3_REG (0x0328)



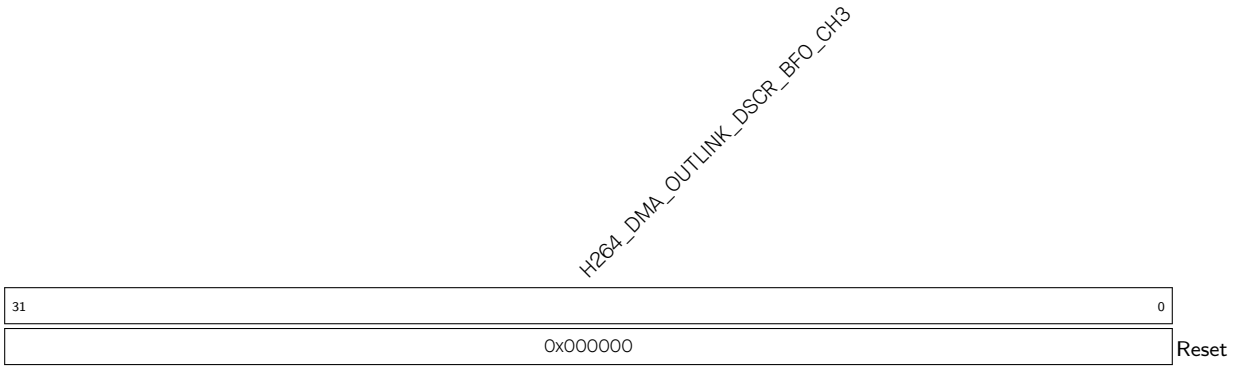
H264_DMA_OUT_EOF_DES_ADDR_CH3 当此描述符中的 EOF 位为 1 时，表示 inlink 描述符的地址。(RO)

Register 41.251. H264_DMA_OUT_DSCR_CH3_REG (0x032C)



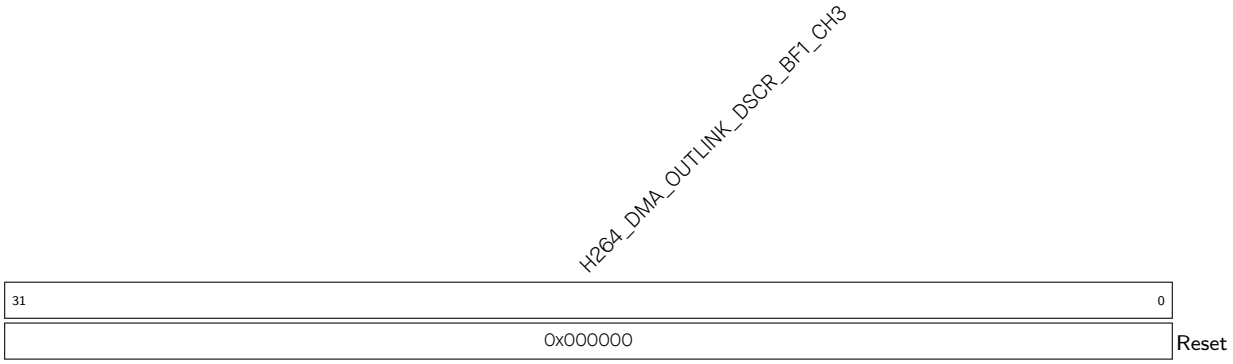
H264_DMA_OUTLINK_DSCR_CH3 表示由当前预读描述符指向的下一个描述符 y+1 的地址。(RO)

Register 41.252. H264_DMA_OUT_DSCR_BF0_CH3_REG (0x0330)



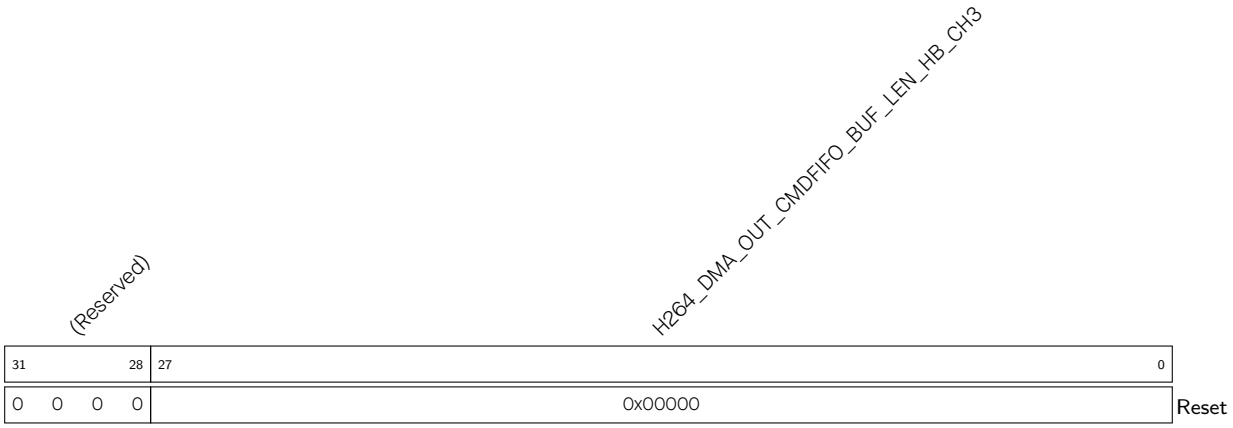
H264_DMA_OUTLINK_DSCR_BF0_CH3 表示预读的当前描述符 y 的地址。(RO)

Register 41.253. H264_DMA_OUT_DSCR_BF1_CH3_REG (0x0334)



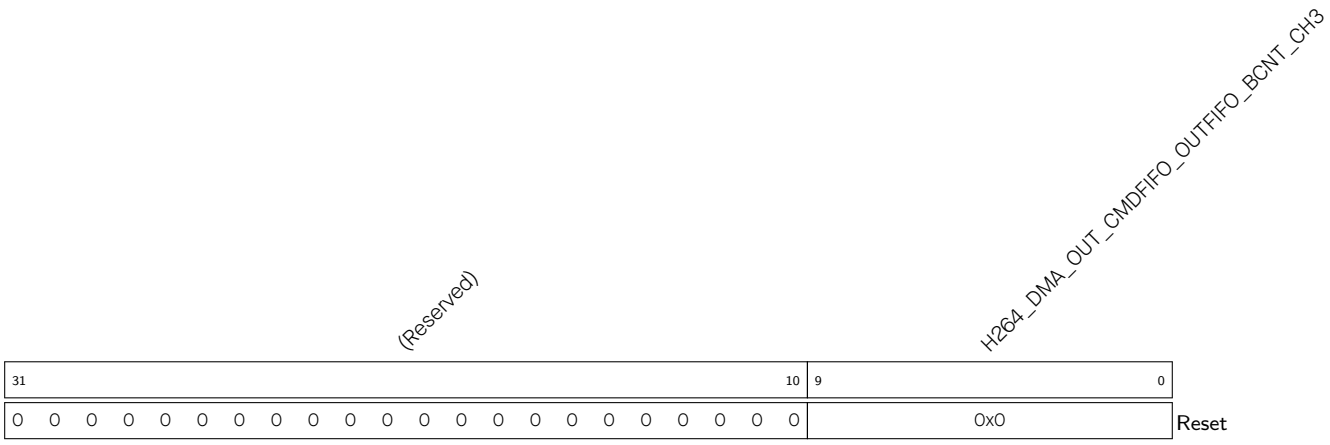
H264_DMA_OUTLINK_DSCR_BF1_CH3 表示预读的前一个描述符 y-1 的地址。(RO)

Register 41.254. H264_DMA_OUT_BUF_LEN_CH3_REG (0x0370)



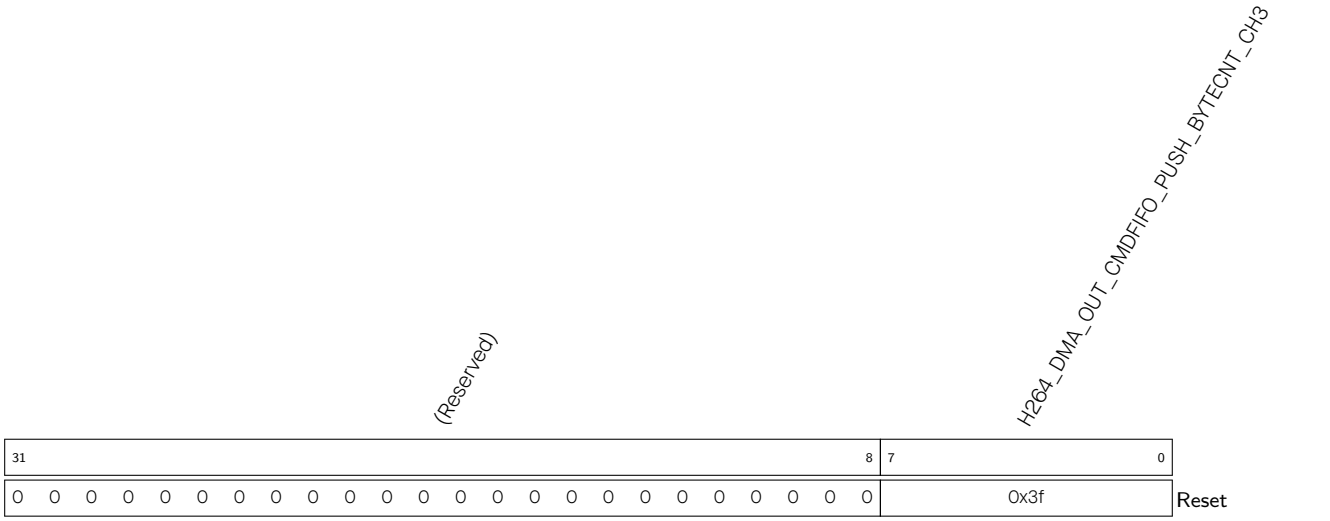
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH3 表示剩余的请求数据。(RO)

Register 41.255. H264_DMA_OUT_FIFO_BCNT_CH3_REG (0x0374)



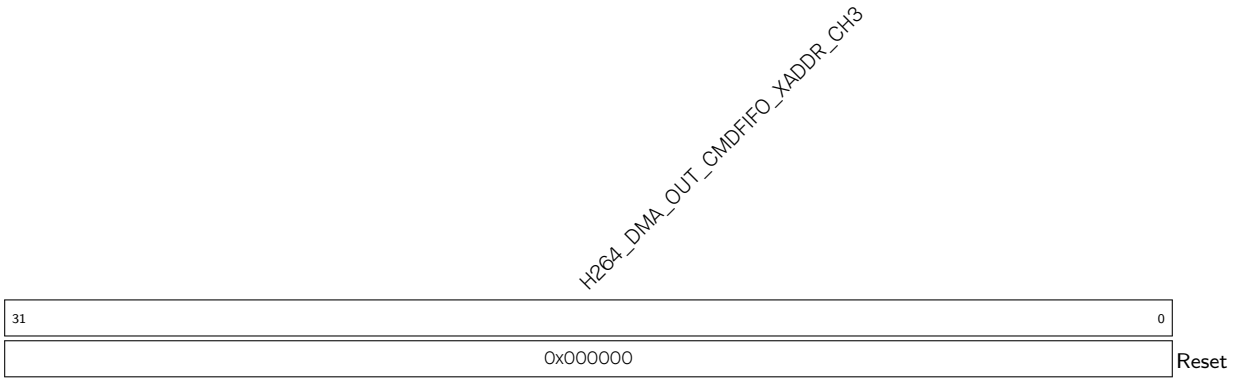
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH3 表示已在 FIFO 中使用的数据空间。(RO)

Register 41.256. H264_DMA_OUT_PUSH_BYTECNT_CH3_REG (0x0378)



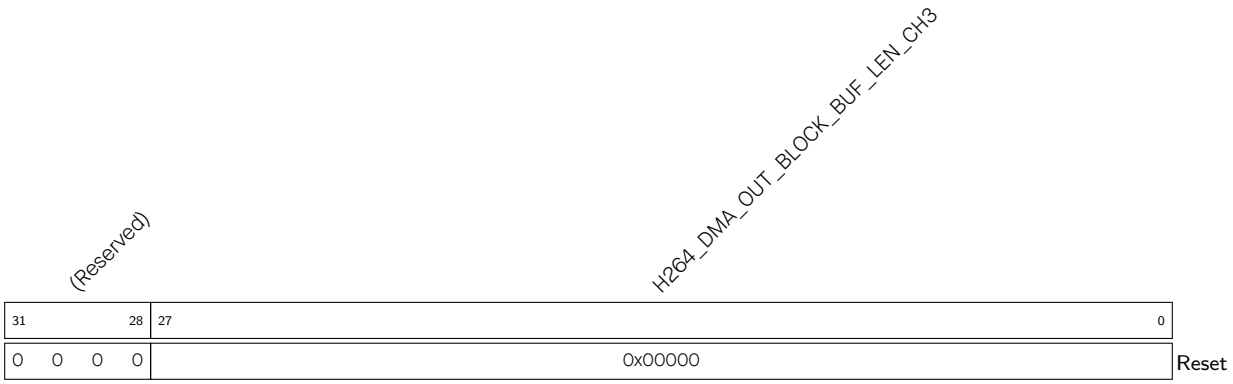
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH3 表示当前 AXI 请求中的数据量。(RO)

Register 41.257. H264_DMA_OUT_XADDR_CH3_REG (0x037C)



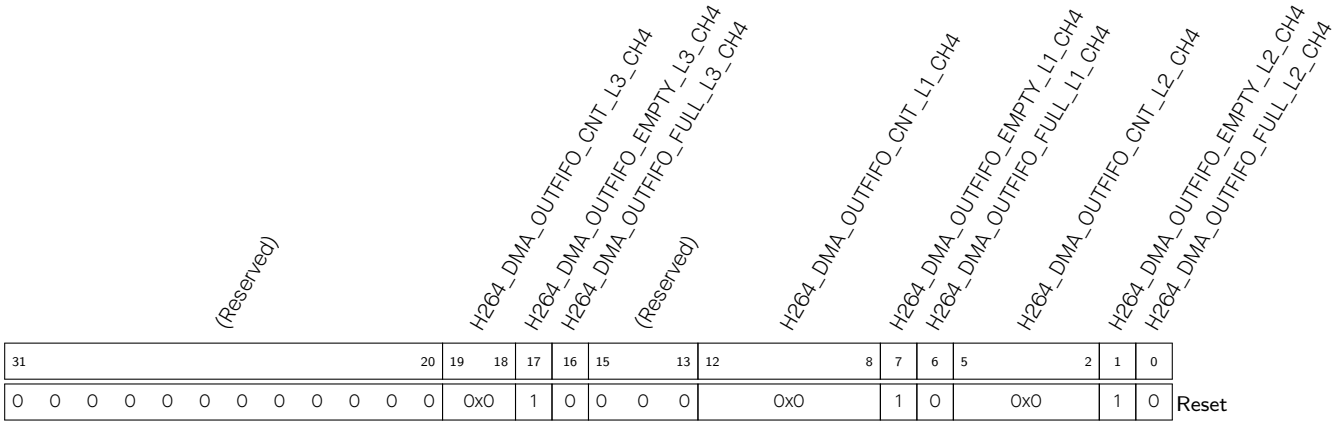
H264_DMA_OUT_CMDFIFO_XADDR_CH3 表示当前 AXI 请求的地址。(RO)

Register 41.258. H264_DMA_OUT_BLOCK_BUF_LEN_CH3_REG (0x0380)



H264_DMA_OUT_BLOCK_BUF_LEN_CH3 表示剩余的块请求数据。(RO)

Register 41.259. H264_DMA_OUTFIFO_STATUS_CH4_REG (0x0414)



H264_DMA_OUTFIFO_FULL_L2_CH4 表示 TX 通道 4 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L2_CH4 表示 TX 通道 4 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L2_CH4 表示 TX 通道 4 的 l2 FIFO 中的数据量。测量单位：16 字节。

(RO)

H264_DMA_OUTFIFO_FULL_L1_CH4 表示 TX 通道 4 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L1_CH4 表示 TX 通道 4 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_OUTFIFO_CNT_L1_CH4 表示 TX 通道 4 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_OUTFIFO_FULL_L3_CH4 表示 TX 通道 4 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_OUTFIFO_EMPTY_L3_CH4 表示 TX 通道 4 的 l3 FIFO 是否为空。

0：不为空

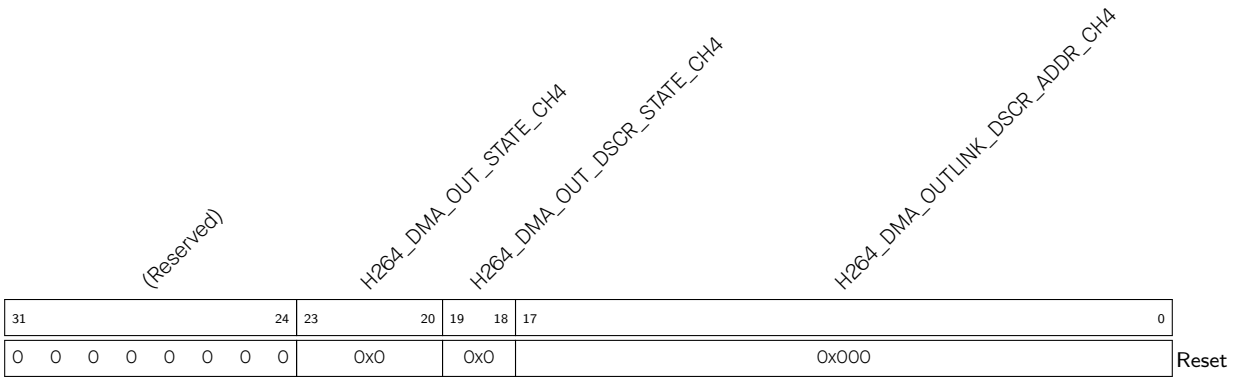
1：空

(RO)

H264_DMA_OUTFIFO_CNT_L3_CH4 表示 TX 通道 4 的 l3 FIFO 中的数据量。测量单位：8 字节。

(RO)

Register 41.260. H264_DMA_OUT_STATE_CH4_REG (0x0424)

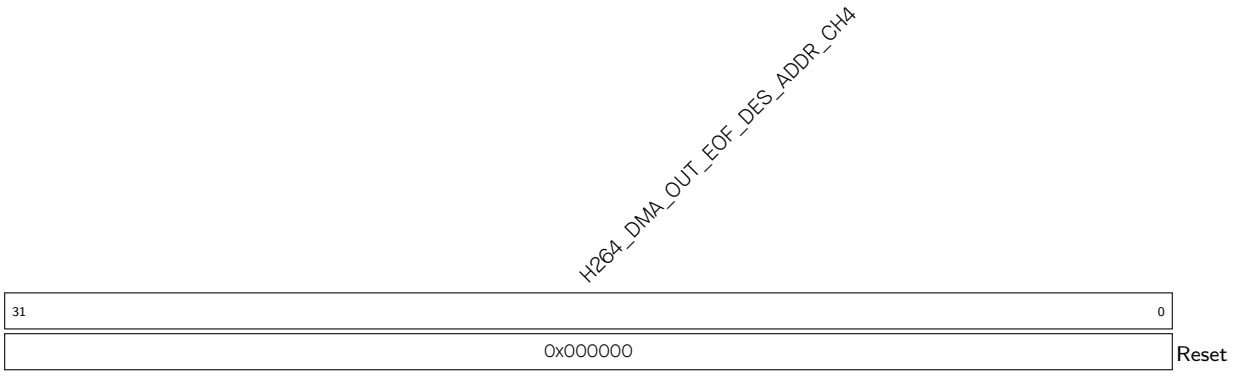


H264_DMA_OUTLINK_DSCR_ADDR_CH4 表示 TX 通道 4 的当前 outlink 描述符的地址。(RO)

H264_DMA_OUT_DSCR_STATE_CH4 表示 TX 通道 4 的描述符状态机的当前状态。(RO)

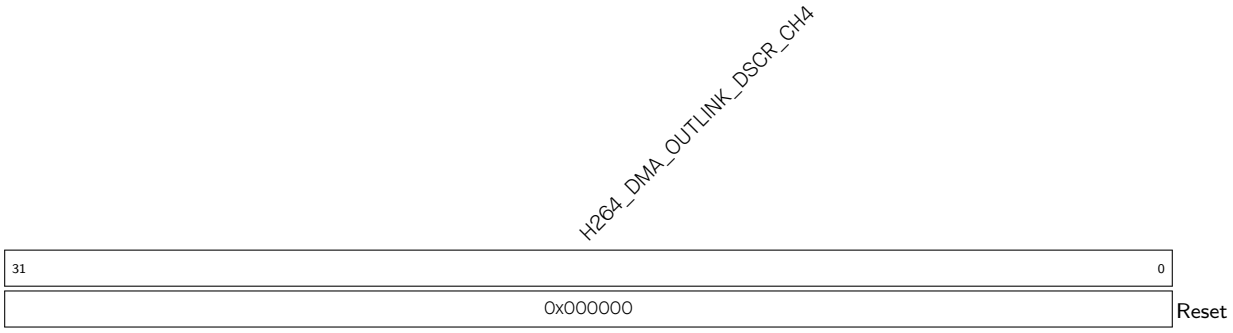
H264_DMA_OUT_STATE_CH4 表示 TX 通道 4 的当前控制模块状态机状态。(RO)

Register 41.261. H264_DMA_OUT_EOF_DES_ADDR_CH4_REG (0x0428)



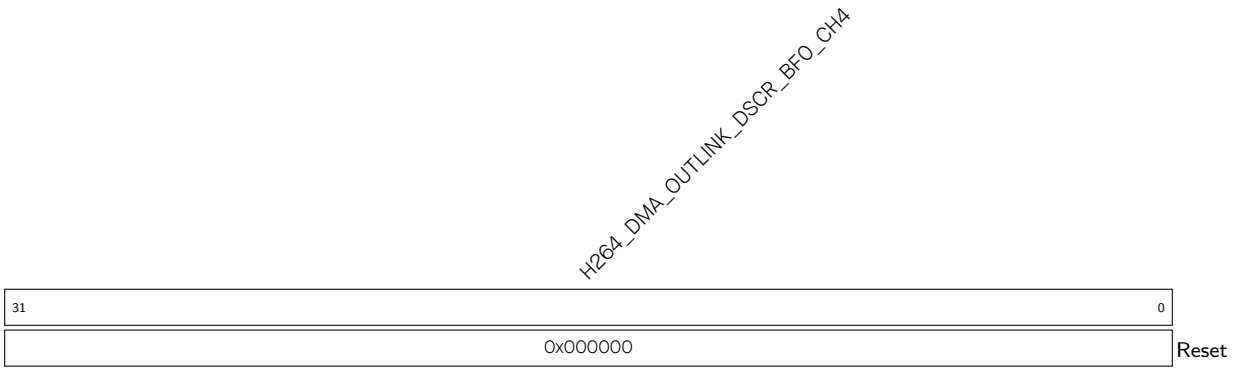
H264_DMA_OUT_EOF_DES_ADDR_CH4 当此描述符中的 EOF 位为 1 时，表示 inlink 描述符的地址。(RO)

Register 41.262. H264_DMA_OUT_DSCR_CH4_REG (0x042C)



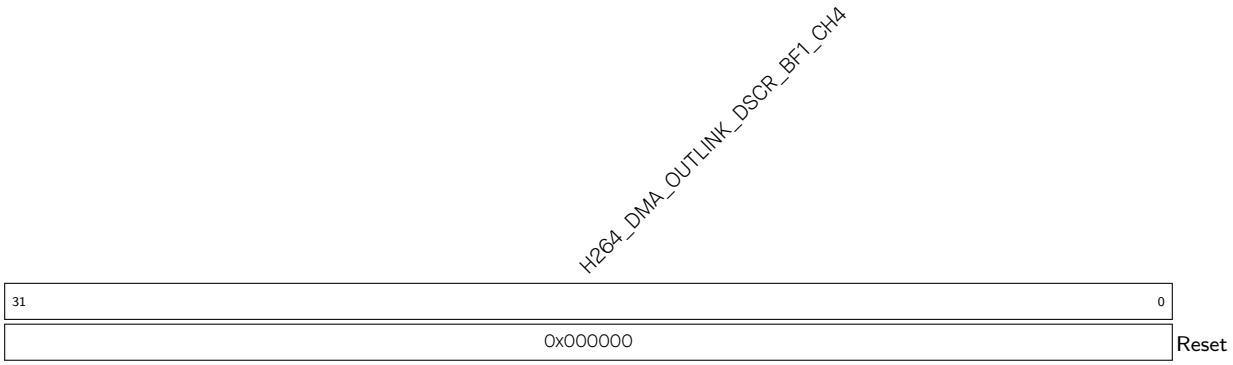
H264_DMA_OUTLINK_DSCR_CH4 表示由当前预读描述符指向的下一个描述符 y+1 的地址。(RO)

Register 41.263. H264_DMA_OUT_DSCR_BF0_CH4_REG (0x0430)



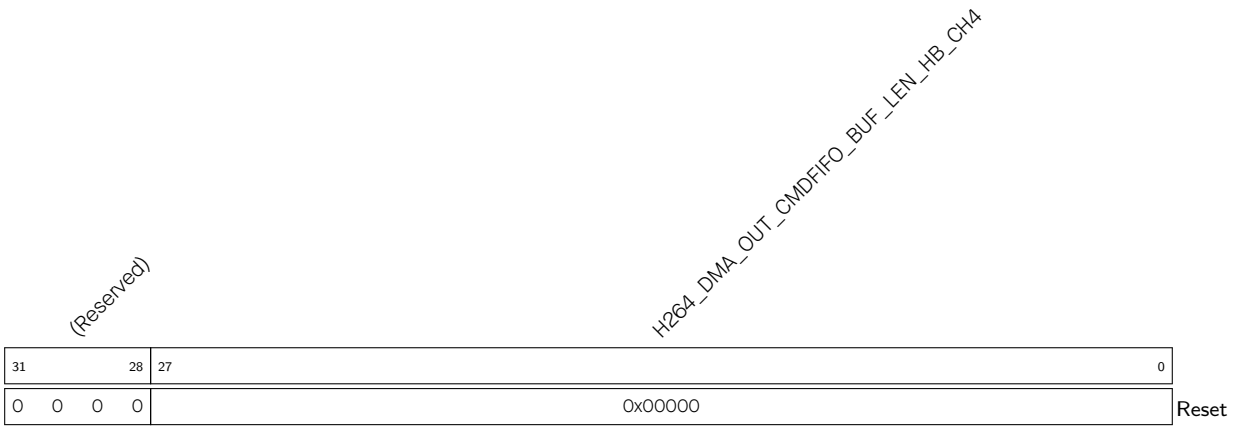
H264_DMA_OUTLINK_DSCR_BF0_CH4 表示预读的当前描述符 y 的地址。(RO)

Register 41.264. H264_DMA_OUT_DSCR_BF1_CH4_REG (0x0434)



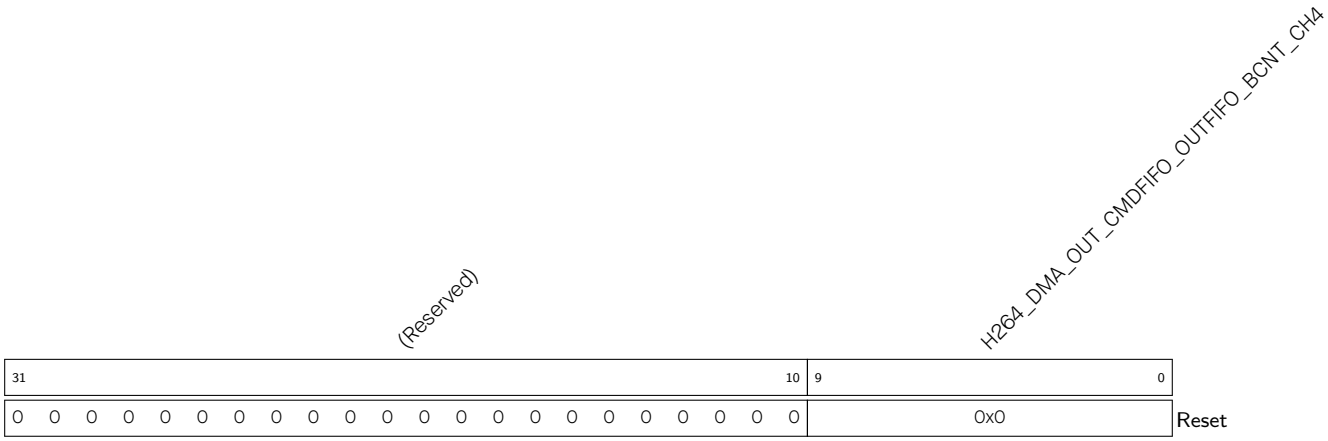
H264_DMA_OUTLINK_DSCR_BF1_CH4 表示预读的前一个描述符 y-1 的地址。(RO)

Register 41.265. H264_DMA_OUT_BUF_LEN_CH4_REG (0x0470)



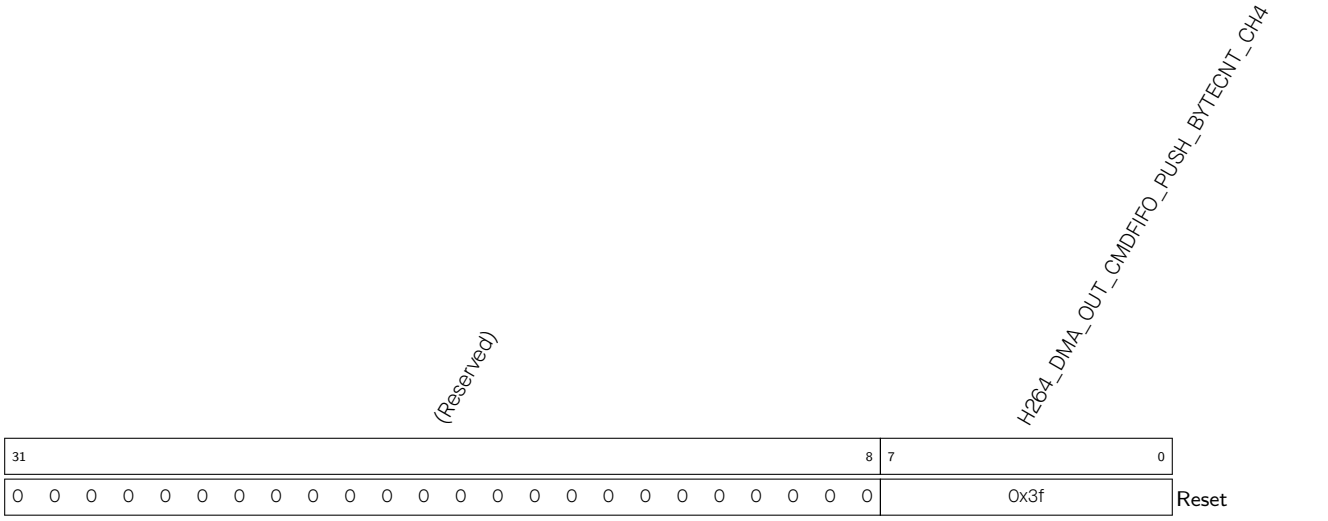
H264_DMA_OUT_CMDFIFO_BUF_LEN_HB_CH4 表示剩余的请求数据。(RO)

Register 41.266. H264_DMA_OUT_FIFO_BCNT_CH4_REG (0x0474)



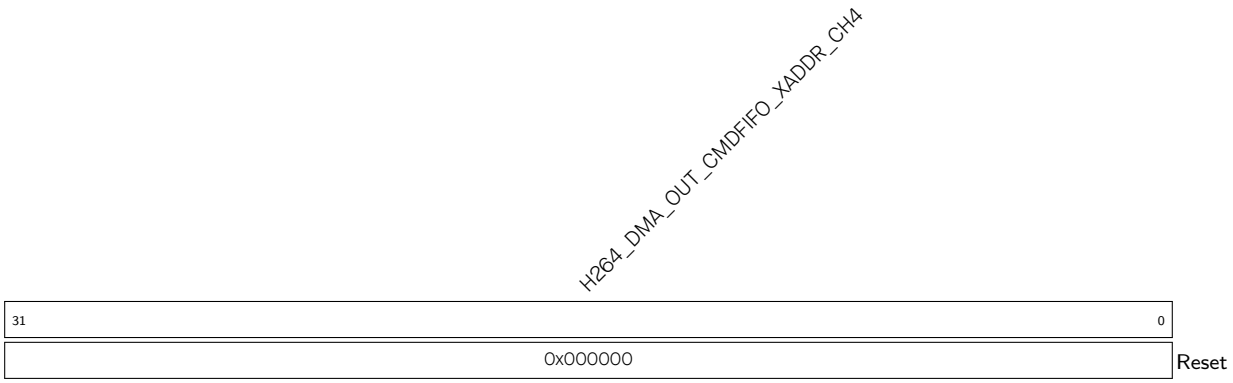
H264_DMA_OUT_CMDFIFO_OUTFIFO_BCNT_CH4 表示已在 FIFO 中使用的数据空间。(RO)

Register 41.267. H264_DMA_OUT_PUSH_BYTECNT_CH4_REG (0x0478)



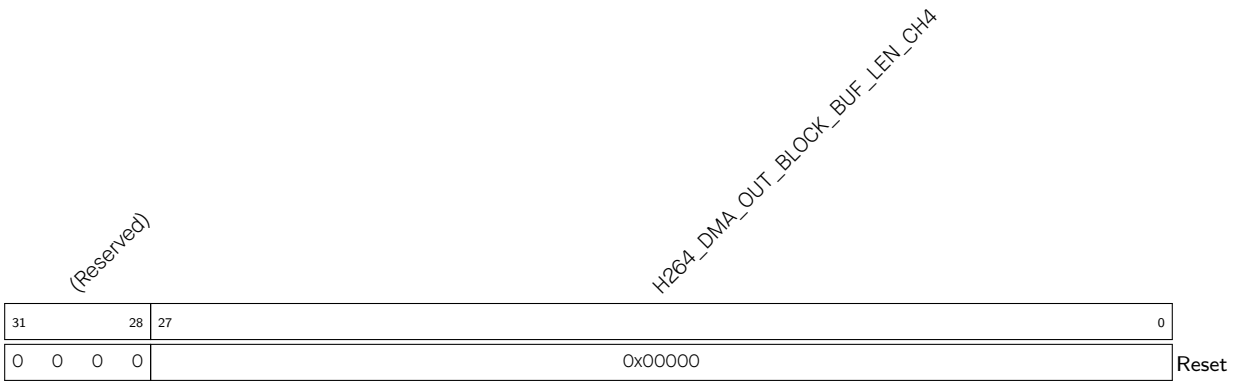
H264_DMA_OUT_CMDFIFO_PUSH_BYTECNT_CH4 表示当前 AXI 请求中的数据量。(RO)

Register 41.268. H264_DMA_OUT_XADDR_CH4_REG (0x047C)



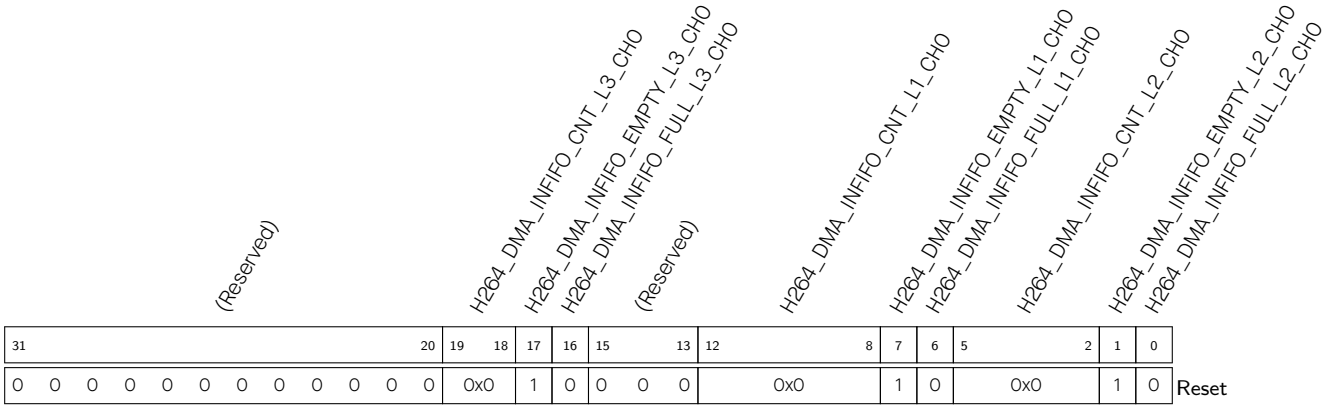
H264_DMA_OUT_CMDFIFO_XADDR_CH4 表示当前 AXI 请求的地址。(RO)

Register 41.269. H264_DMA_OUT_BLOCK_BUF_LEN_CH4_REG (0x0480)



H264_DMA_OUT_BLOCK_BUF_LEN_CH4 表示剩余的块请求数据。(RO)

Register 41.270. H264_DMA_INFIFO_STATUS_CH0_REG (0x0514)



H264_DMA_INFIFO_FULL_L2_CH0 表示 RX 通道 0 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH0 表示 RX 通道 0 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L2_CH0 表示 RX 通道 0 的 l2 FIFO 中的数据量。测量单位：16 字节。(RO)

H264_DMA_INFIFO_FULL_L1_CH0 表示 RX 通道 0 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH0 表示 RX 通道 0 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L1_CH0 表示 RX 通道 0 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_INFIFO_FULL_L3_CH0 表示 RX 通道 0 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH0 表示 RX 通道 0 的 l3 FIFO 是否为空。

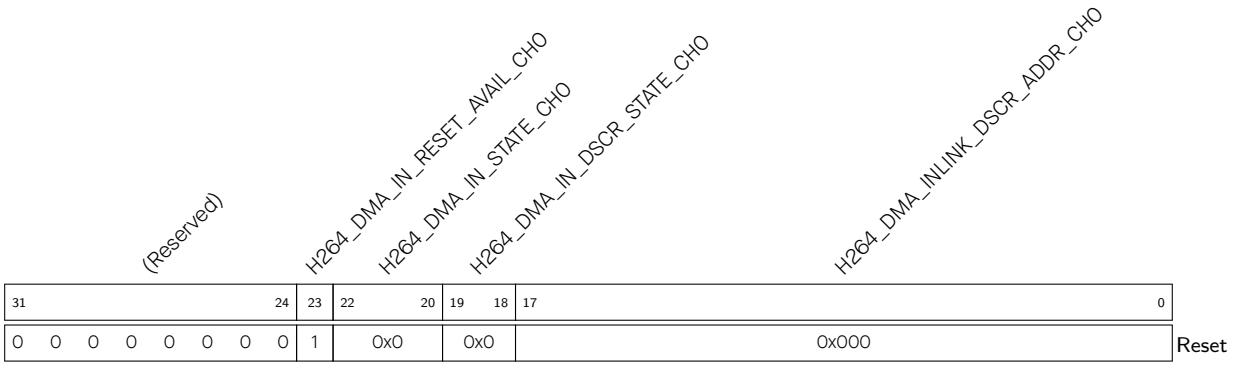
0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L3_CH0 表示 RX 通道 0 的 l3 FIFO 中的数据量。测量单位：8 字节。(RO)

Register 41.271. H264_DMA_IN_STATE_CH0_REG (0x0524)



H264_DMA_INLINK_DSCR_ADDR_CH0 表示 RX 通道 0 的当前 inlink 描述符的地址。(RO)

H264_DMA_IN_DSCR_STATE_CH0 表示 RX 通道 0 的描述符状态机的当前状态。(RO)

H264_DMA_IN_STATE_CH0 表示 RX 通道 0 的当前控制模块状态机状态。(RO)

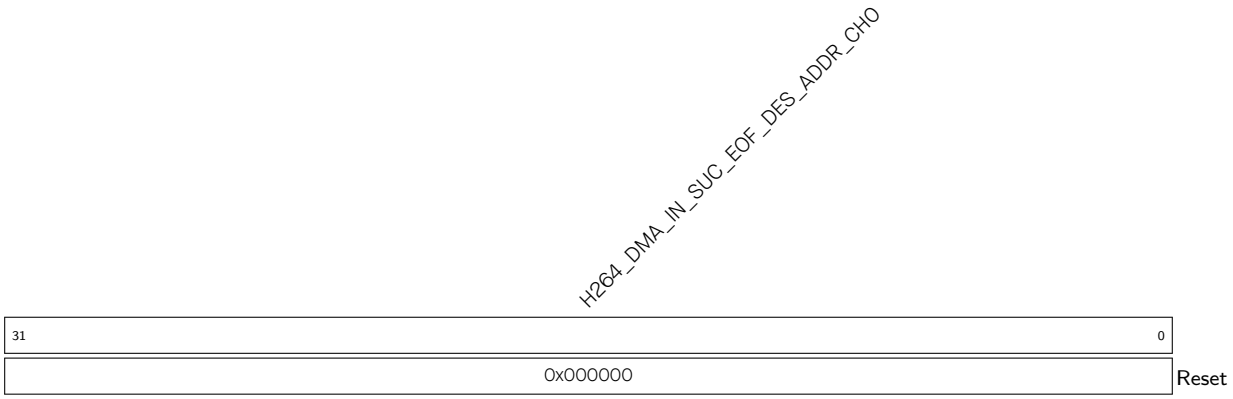
H264_DMA_IN_RESET_AVAIL_CH0 表示是否可以安全地重置通道。

0：不安全

1：安全

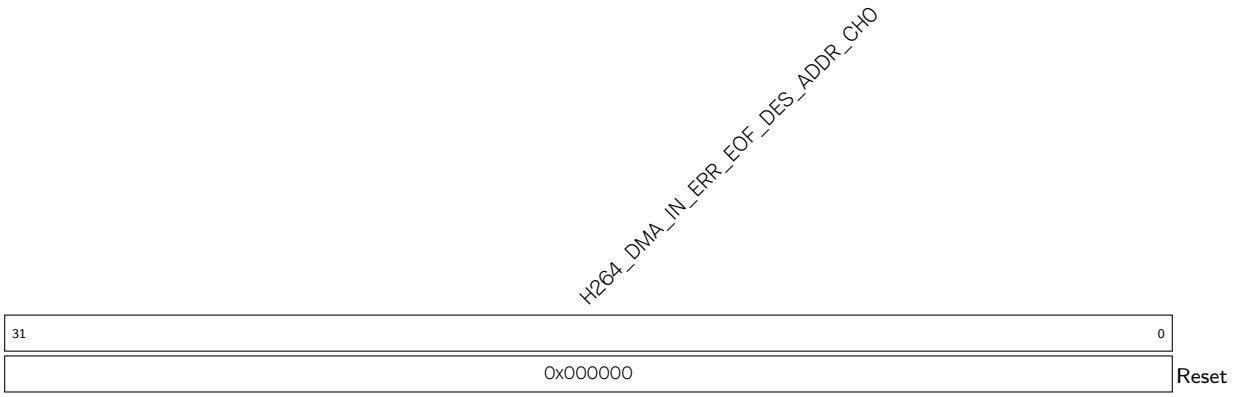
(RO)

Register 41.272. H264_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG (0x0528)



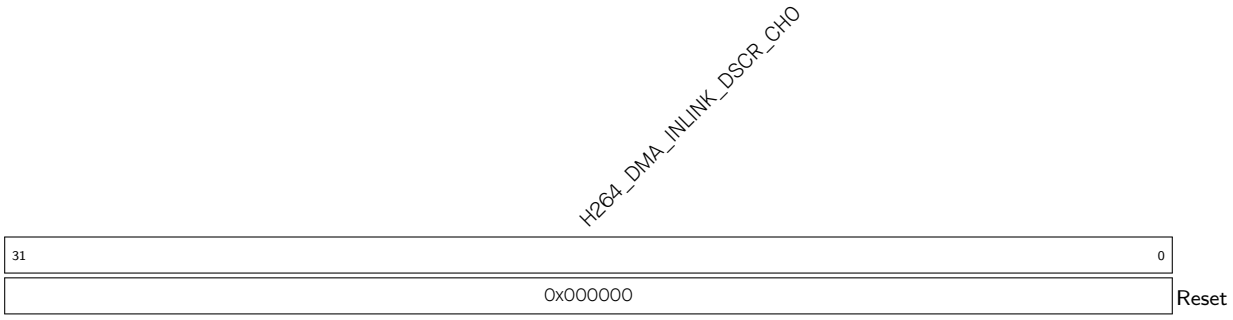
H264_DMA_IN_SUC_EOF_DES_ADDR_CH0 表示当此描述符中的 EOF 位为 1 时的 inlink 描述符的地址。(RO)

Register 41.273. H264_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG (0x052C)



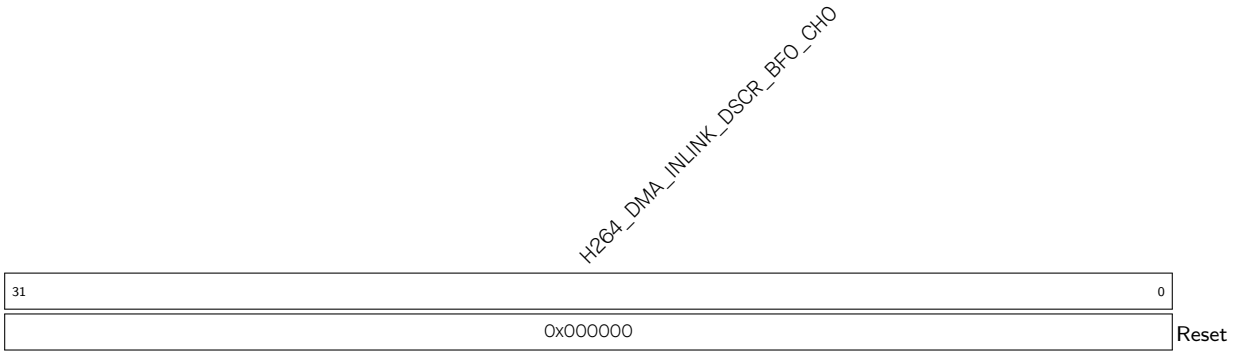
H264_DMA_IN_ERR_EOF_DES_ADDR_CH0 表示当前数据接收过程中出现错误时 inlink 描述符的地址。(RO)

Register 41.274. H264_DMA_IN_DSCR_CH0_REG (0x0530)



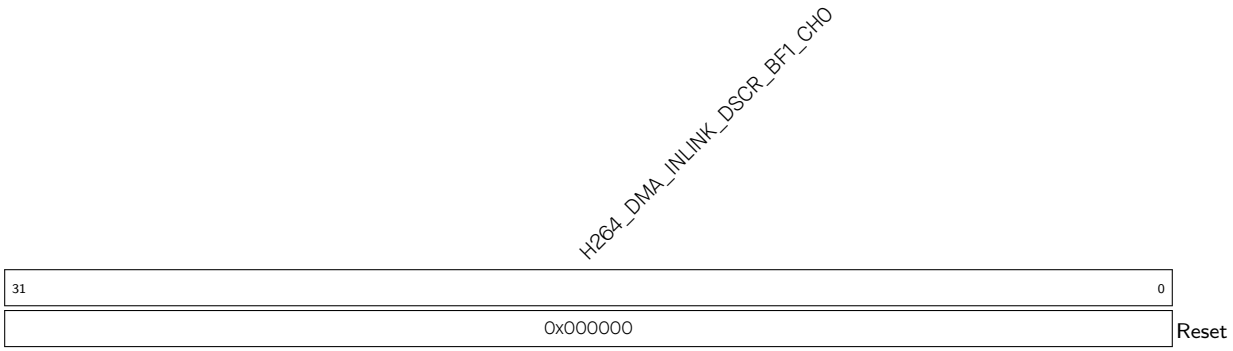
H264_DMA_INLINK_DSCR_CH0 表示由当前预读描述符指向的下一个描述符 x+1 的地址。(RO)

Register 41.275. H264_DMA_IN_DSCR_BF0_CH0_REG (0x0534)



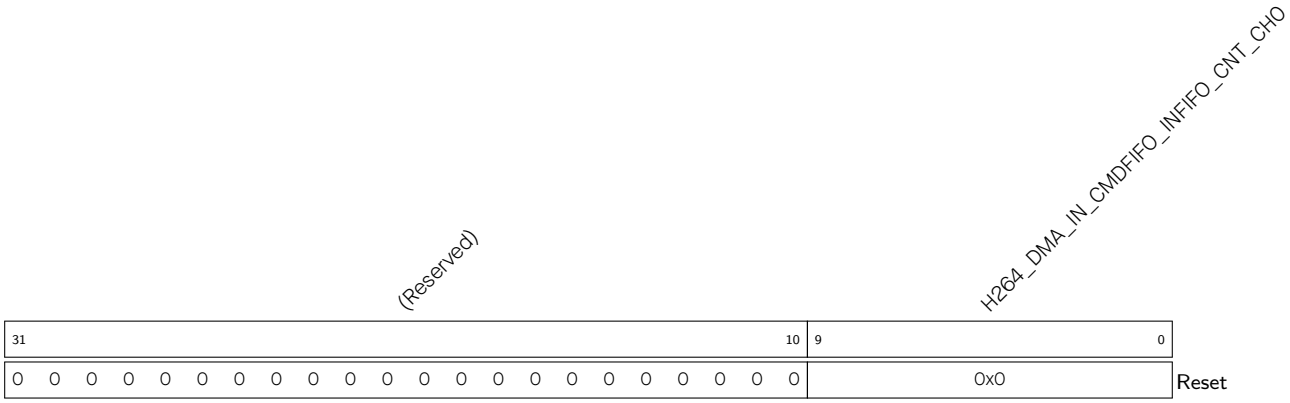
H264_DMA_INLINK_DSCR_BF0_CH0 表示预读的当前描述符 x 的地址。(RO)

Register 41.276. H264_DMA_IN_DSCR_BF1_CH0_REG (0x0538)



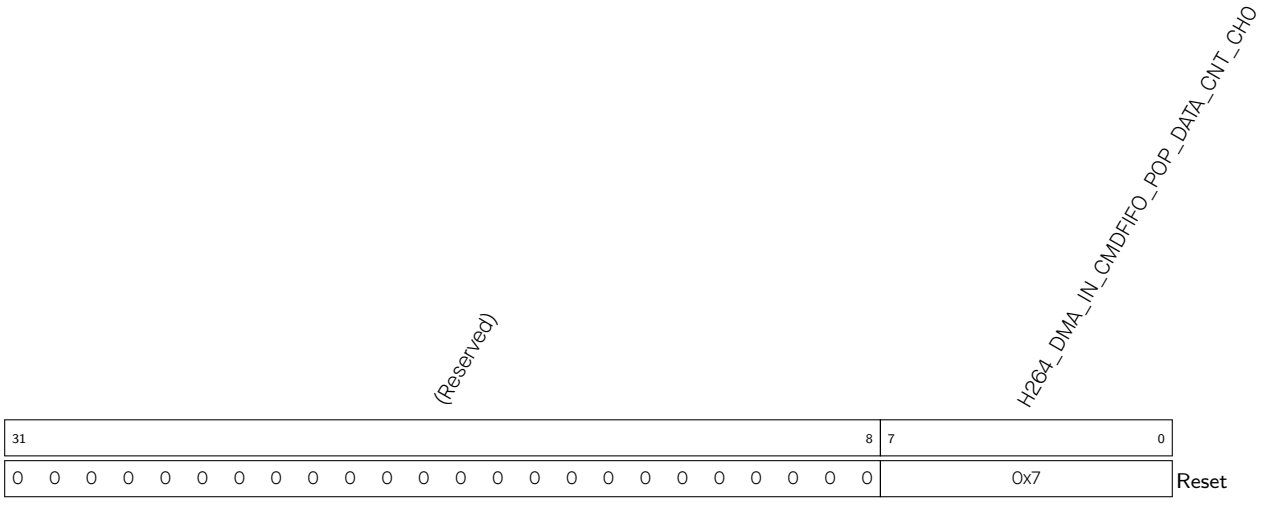
H264_DMA_INLINK_DSCR_BF1_CH0 表示预读的前一个描述符 x-1 的地址。(RO)

Register 41.277. H264_DMA_IN_FIFO_CNT_CH0_REG (0x0580)



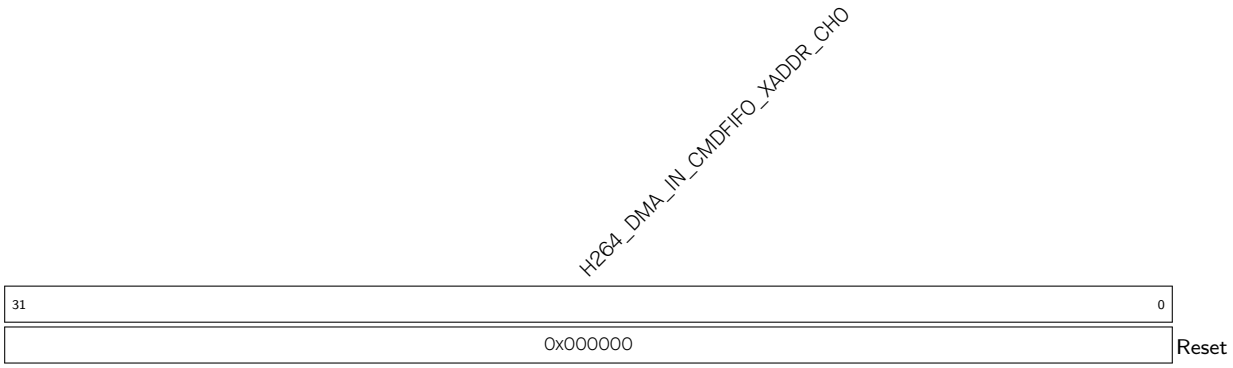
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH0 表示 FIFO 中剩余的请求数据。(RO)

Register 41.278. H264_DMA_IN_POP_DATA_CNT_CH0_REG (0x0584)



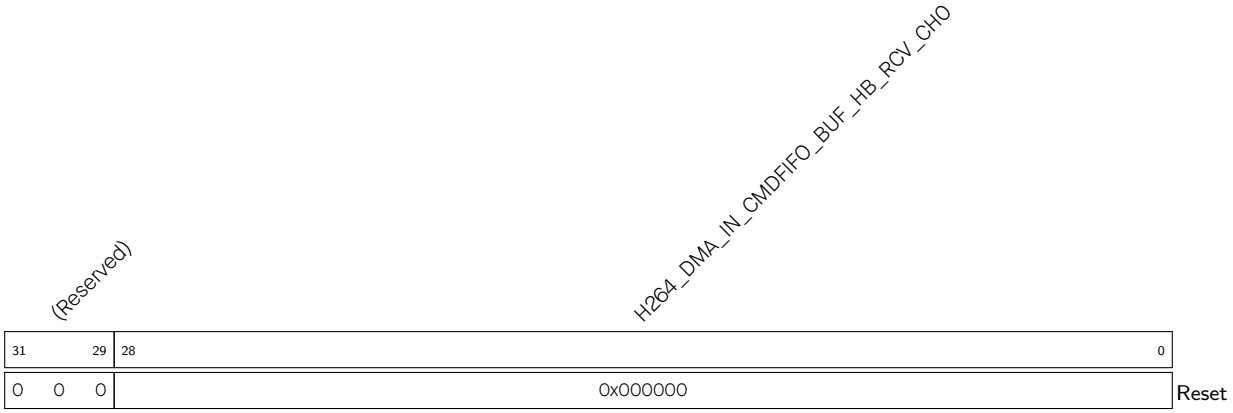
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH0 表示当前 AXI 请求中的数据量。(RO)

Register 41.279. H264_DMA_IN_XADDR_CH0_REG (0x0588)



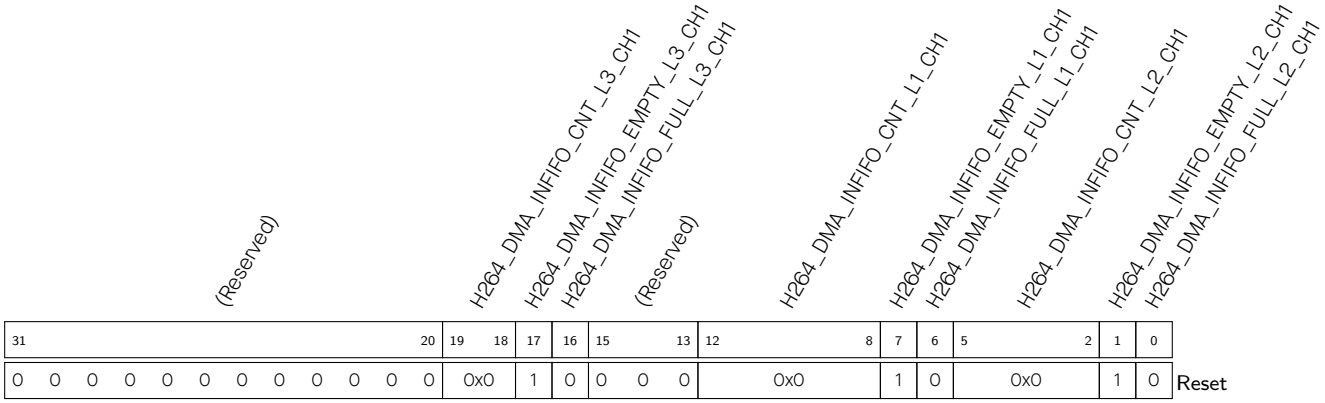
H264_DMA_IN_CMDFIFO_XADDR_CH0 表示当前 AXI 请求的地址。(RO)

Register 41.280. H264_DMA_IN_BUF_HB_RCV_CH0_REG (0x058C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH0 表示已请求的数据量。(RO)

Register 41.281. H264_DMA_INFIFO_STATUS_CH1_REG (0x0614)



H264_DMA_INFIFO_FULL_L2_CH1 表示 RX 通道 1 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH1 表示 RX 通道 1 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L2_CH1 表示 RX 通道 1 的 l2 FIFO 中的数据量。测量单位：16 字节。(RO)

H264_DMA_INFIFO_FULL_L1_CH1 表示 RX 通道 1 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH1 表示 RX 通道 1 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L1_CH1 表示 RX 通道 1 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_INFIFO_FULL_L3_CH1 表示 RX 通道 1 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH1 表示 RX 通道 1 的 l3 FIFO 是否为空。

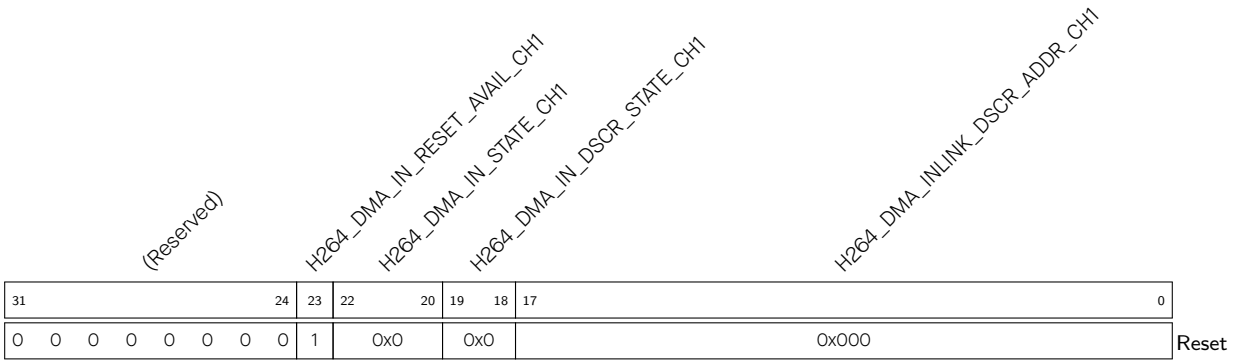
0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L3_CH1 表示 RX 通道 1 的 l3 FIFO 中的数据量。测量单位：8 字节。(RO)

Register 41.282. H264_DMA_IN_STATE_CH1_REG (0x0624)



H264_DMA_INLINK_DSCR_ADDR_CH1 表示 RX 通道 1 的当前 inlink 描述符的地址。(RO)

H264_DMA_IN_DSCR_STATE_CH1 表示 RX 通道 1 的描述符状态机的当前状态。(RO)

H264_DMA_IN_STATE_CH1 表示 RX 通道 1 的当前控制模块状态机状态。(RO)

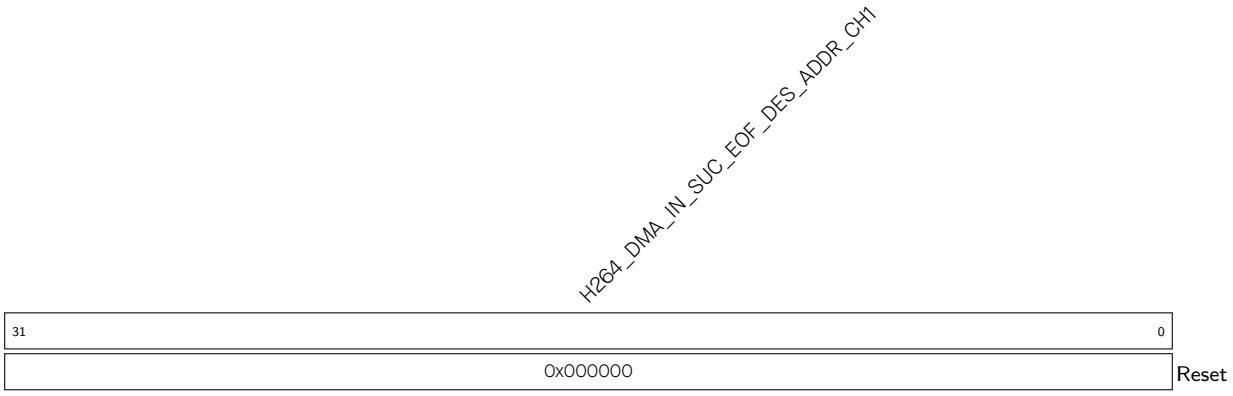
H264_DMA_IN_RESET_AVAIL_CH1 表示是否可以安全地重置通道。

0：不安全

1：安全

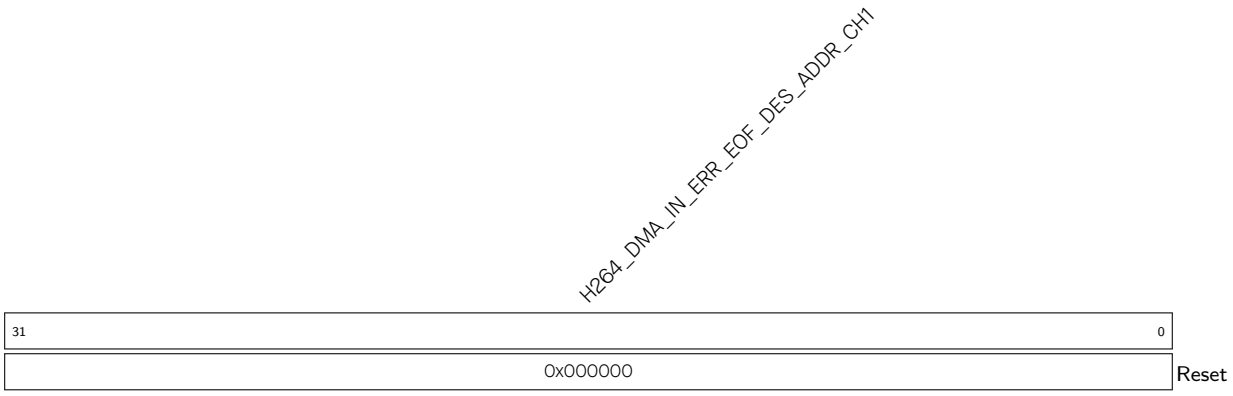
(RO)

Register 41.283. H264_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG (0x0628)



H264_DMA_IN_SUC_EOF_DES_ADDR_CH1 表示当此描述符中的 EOF 位为 1 时的 inlink 描述符的地址。(RO)

Register 41.284. H264_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG (0x062C)



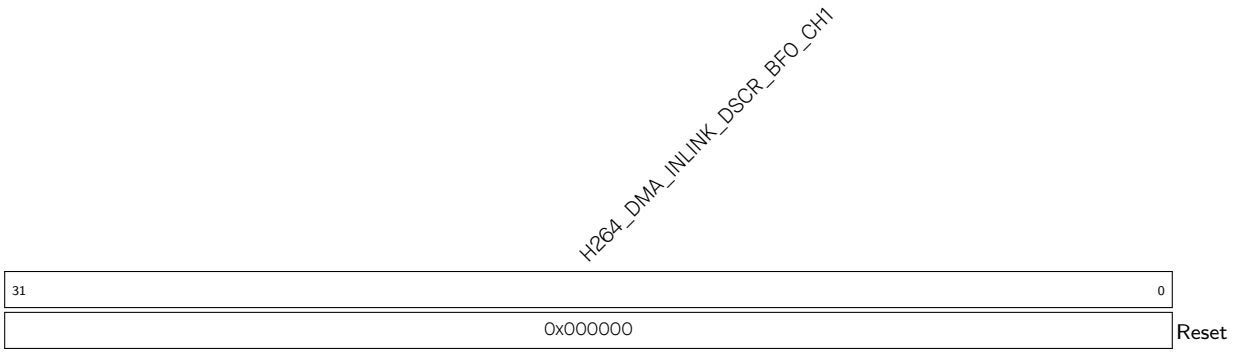
H264_DMA_IN_ERR_EOF_DES_ADDR_CH1 表示当前数据接收过程中出现错误时 inlink 描述符的地址。(RO)

Register 41.285. H264_DMA_IN_DSCR_CH1_REG (0x0630)



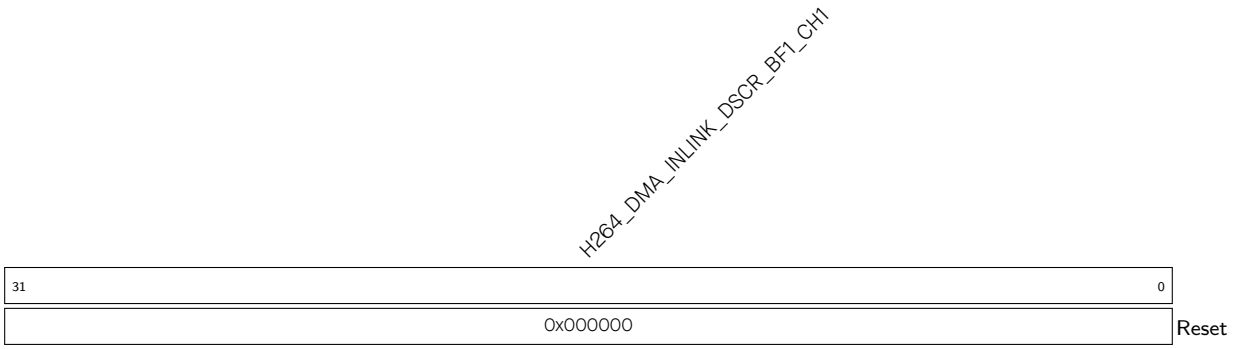
H264_DMA_INLINK_DSCR_CH1 表示由当前预读描述符指向的下一个描述符 x+1 的地址。(RO)

Register 41.286. H264_DMA_IN_DSCR_BF0_CH1_REG (0x0634)



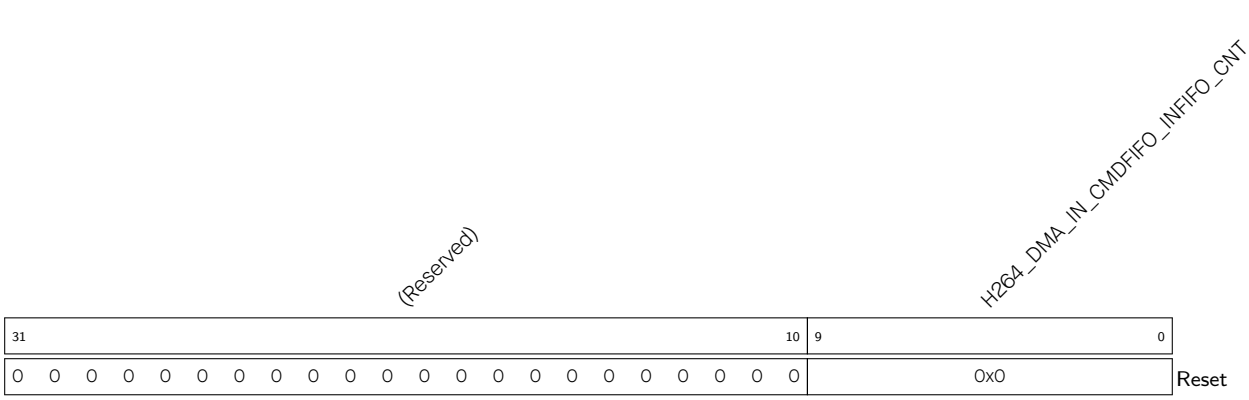
H264_DMA_INLINK_DSCR_BF0_CH1 表示预读的当前描述符 x 的地址。(RO)

Register 41.287. H264_DMA_IN_DSCR_BF1_CH1_REG (0x0638)



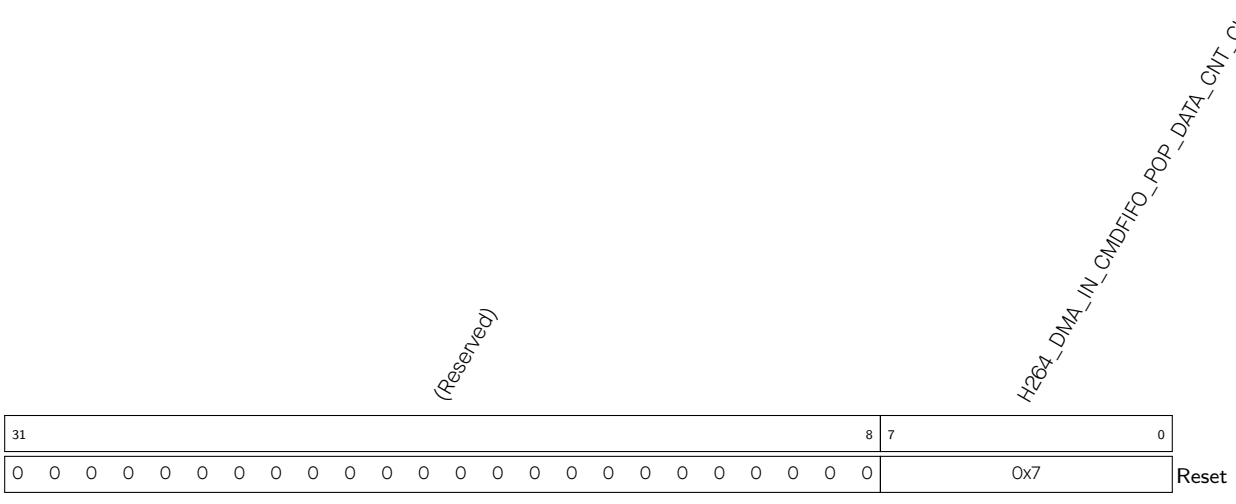
H264_DMA_INLINK_DSCR_BF1_CH1 表示预读的前一个描述符 x-1 的地址。(RO)

Register 41.288. H264_DMA_IN_FIFO_CNT_CH1_REG (0x0680)



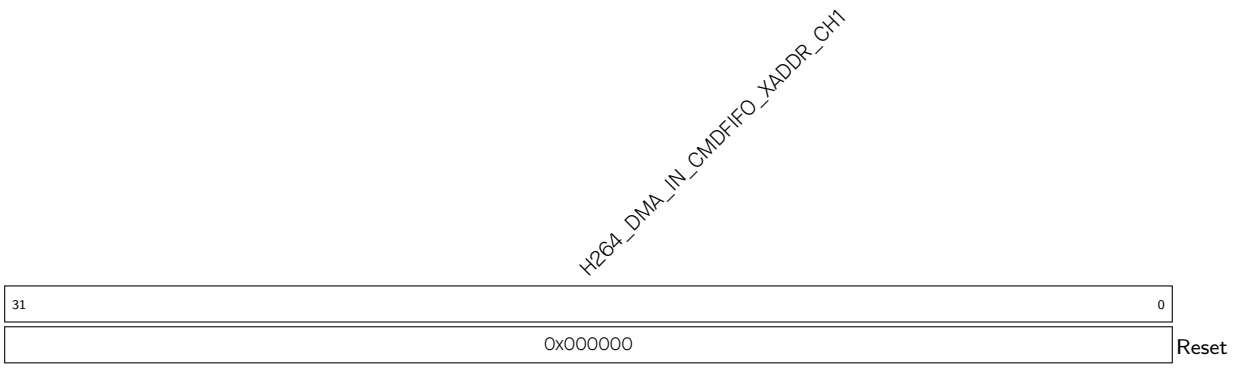
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH1 表示 FIFO 中剩余的请求数据。(RO)

Register 41.289. H264_DMA_IN_POP_DATA_CNT_CH1_REG (0x0684)



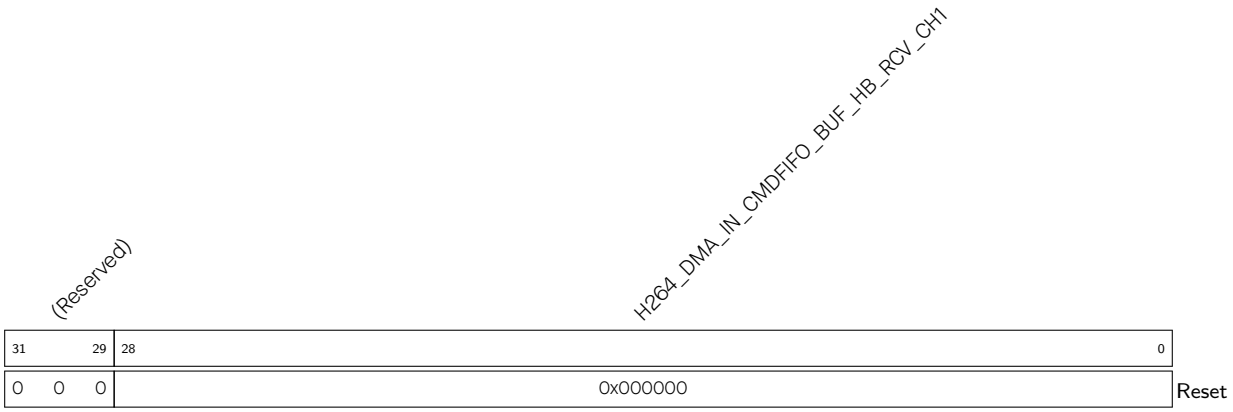
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH1 表示当前 AXI 请求中的数据量。(RO)

Register 41.290. H264_DMA_IN_XADDR_CH1_REG (0x0688)



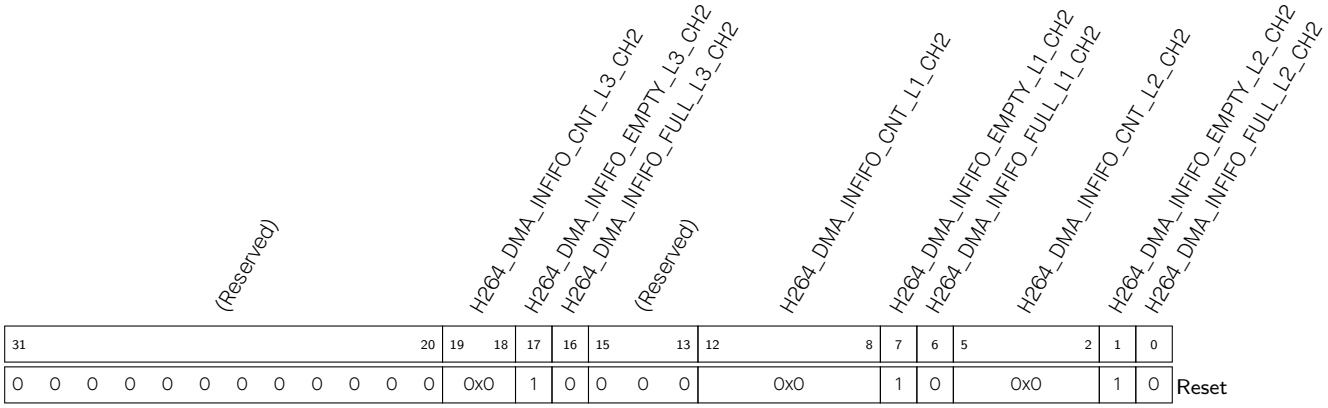
H264_DMA_IN_CMDFIFO_XADDR_CH1 表示当前 AXI 请求的地址。(RO)

Register 41.291. H264_DMA_IN_BUF_HB_RCV_CH1_REG (0x068C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH1 表示已请求的数据量。(RO)

Register 41.292. H264_DMA_INFIFO_STATUS_CH2_REG (0x0714)



H264_DMA_INFIFO_FULL_L2_CH2 表示 RX 通道 2 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH2 表示 RX 通道 2 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L2_CH2 表示 RX 通道 2 的 l2 FIFO 中的数据量。测量单位：16 字节。(RO)

H264_DMA_INFIFO_FULL_L1_CH2 表示 RX 通道 2 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH2 表示 RX 通道 2 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L1_CH2 表示 RX 通道 2 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_INFIFO_FULL_L3_CH2 表示 RX 通道 2 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH2 表示 RX 通道 2 的 l3 FIFO 是否为空。

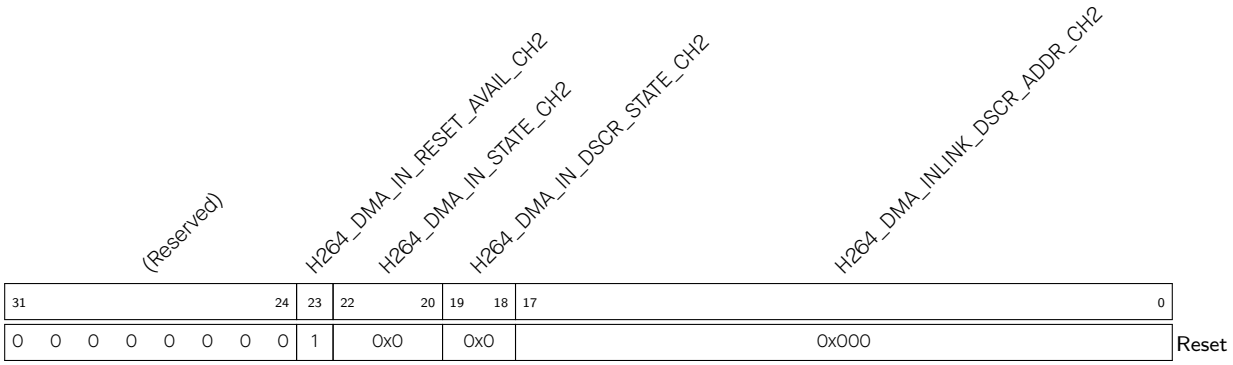
0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L3_CH2 表示 RX 通道 2 的 l3 FIFO 中的数据量。测量单位：8 字节。(RO)

Register 41.293. H264_DMA_IN_STATE_CH2_REG (0x0724)



H264_DMA_INLINK_DSCR_ADDR_CH2 表示 RX 通道 2 的当前 inlink 描述符的地址。(RO)

H264_DMA_IN_DSCR_STATE_CH2 表示 RX 通道 2 的描述符状态机的当前状态。(RO)

H264_DMA_IN_STATE_CH2 表示 RX 通道 2 的当前控制模块状态机状态。(RO)

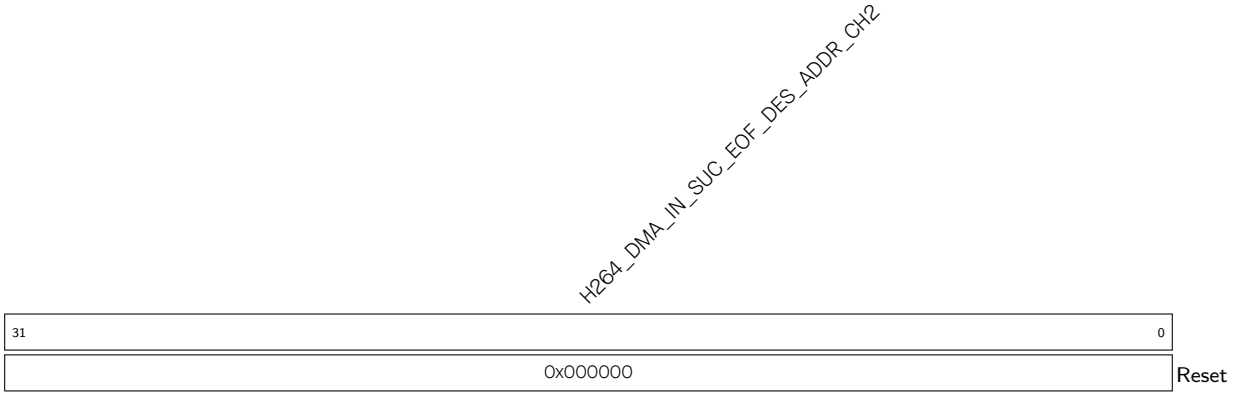
H264_DMA_IN_RESET_AVAIL_CH2 表示是否可以安全地重置通道。

0：不安全

1：安全

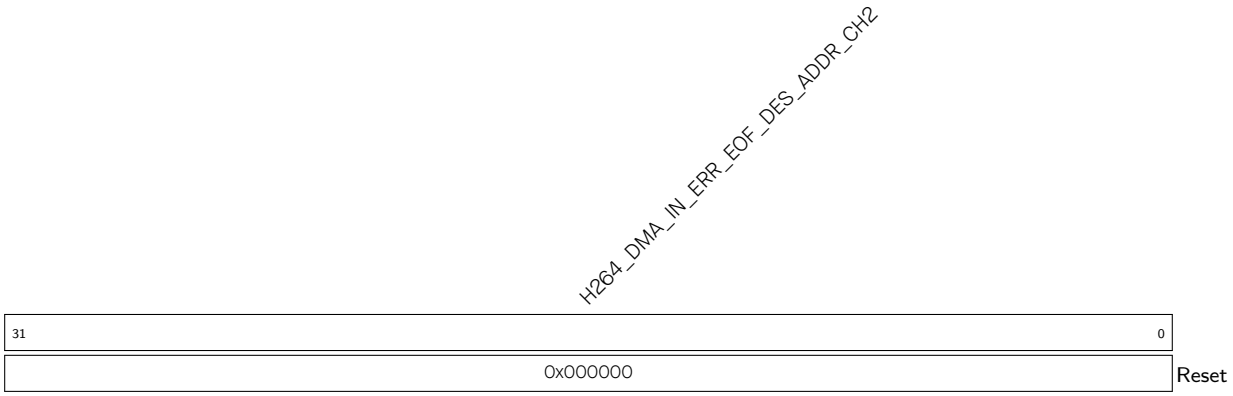
(RO)

Register 41.294. H264_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG (0x0728)



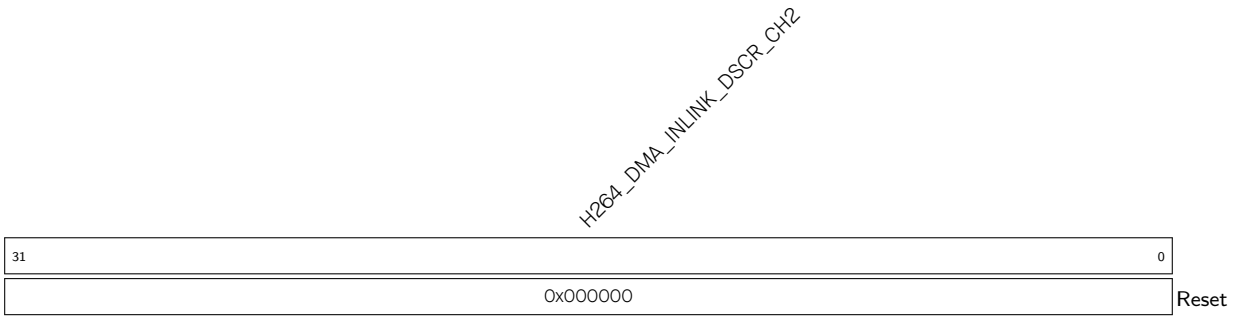
H264_DMA_IN_SUC_EOF_DES_ADDR_CH2 表示当此描述符中的 EOF 位为 1 时的 inlink 描述符的地址。(RO)

Register 41.295. H264_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG (0x072C)



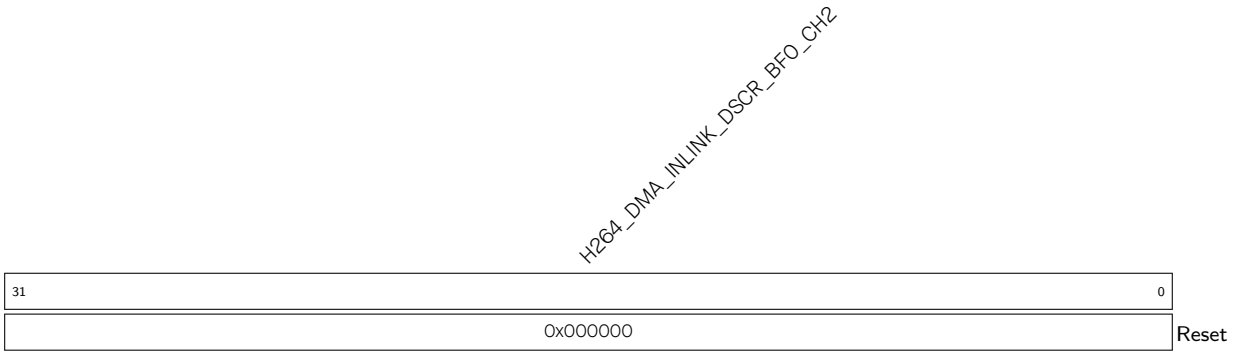
H264_DMA_IN_ERR_EOF_DES_ADDR_CH2 表示当前数据接收过程中出现错误时 inlink 描述符的地址。(RO)

Register 41.296. H264_DMA_IN_DSCR_CH2_REG (0x0730)



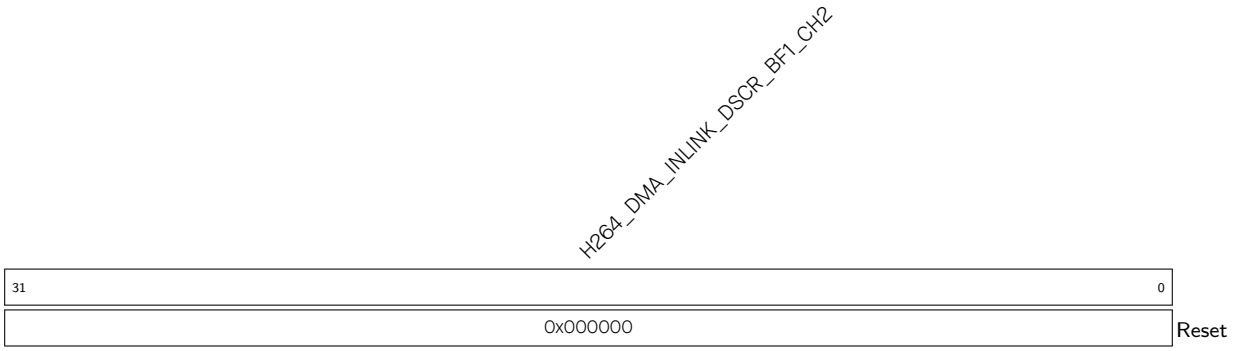
H264_DMA_INLINK_DSCR_CH2 表示由当前预读描述符指向的下一个描述符 x+1 的地址。(RO)

Register 41.297. H264_DMA_IN_DSCR_BF0_CH2_REG (0x0734)



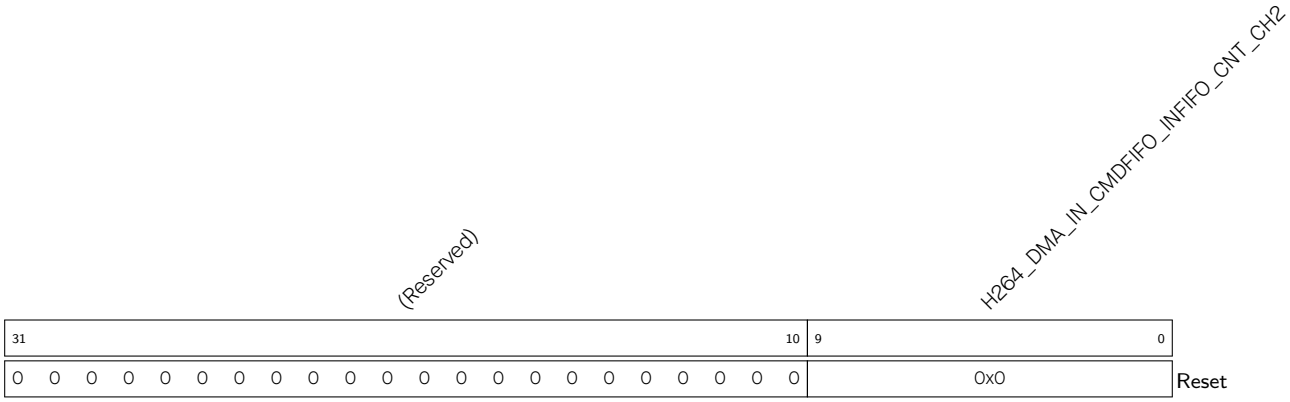
H264_DMA_INLINK_DSCR_BF0_CH2 表示预读的当前描述符 x 的地址。(RO)

Register 41.298. H264_DMA_IN_DSCR_BF1_CH2_REG (0x0738)



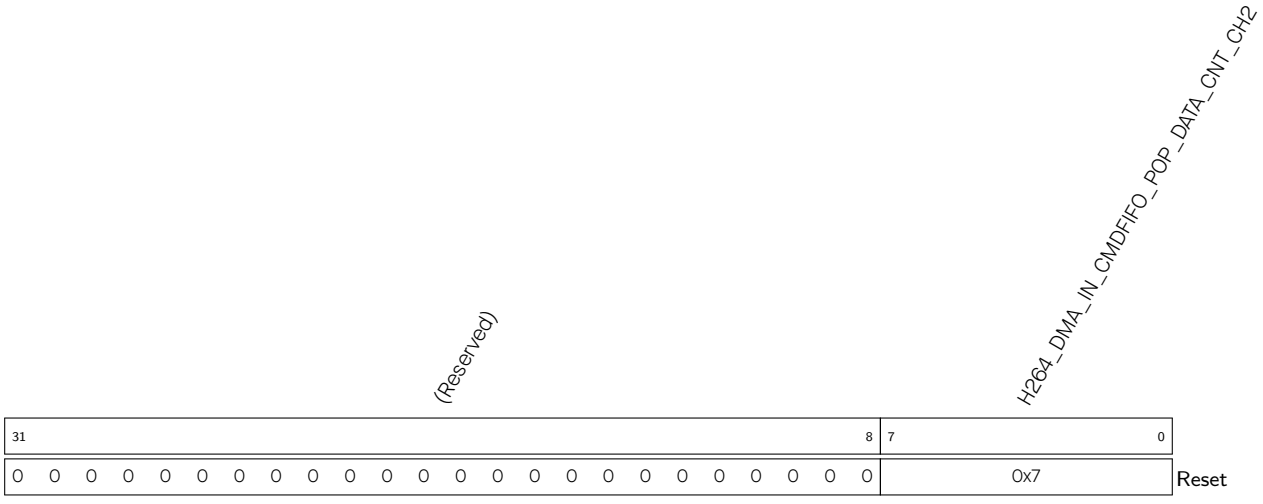
H264_DMA_INLINK_DSCR_BF1_CH2 表示预读的前一个描述符 x-1 的地址。(RO)

Register 41.299. H264_DMA_IN_FIFO_CNT_CH2_REG (0x0780)



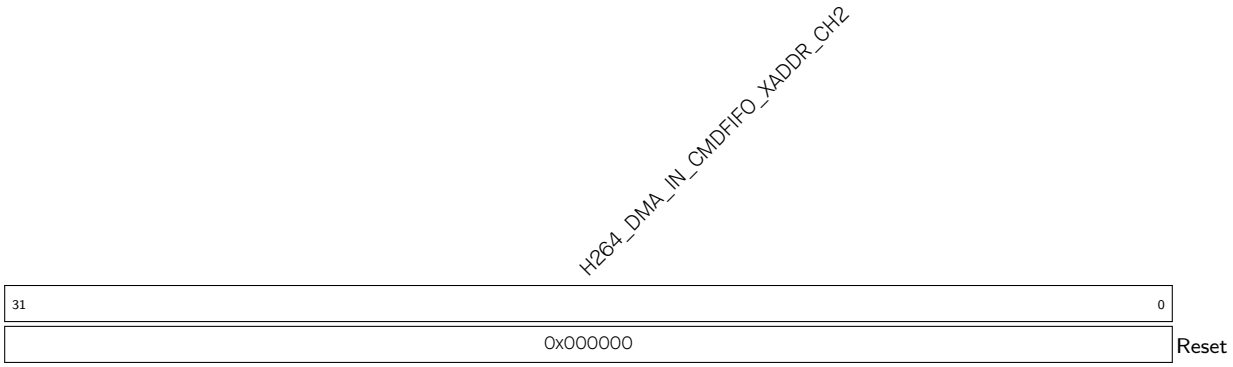
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH2 表示 FIFO 中剩余的请求数据。(RO)

Register 41.300. H264_DMA_IN_POP_DATA_CNT_CH2_REG (0x0784)



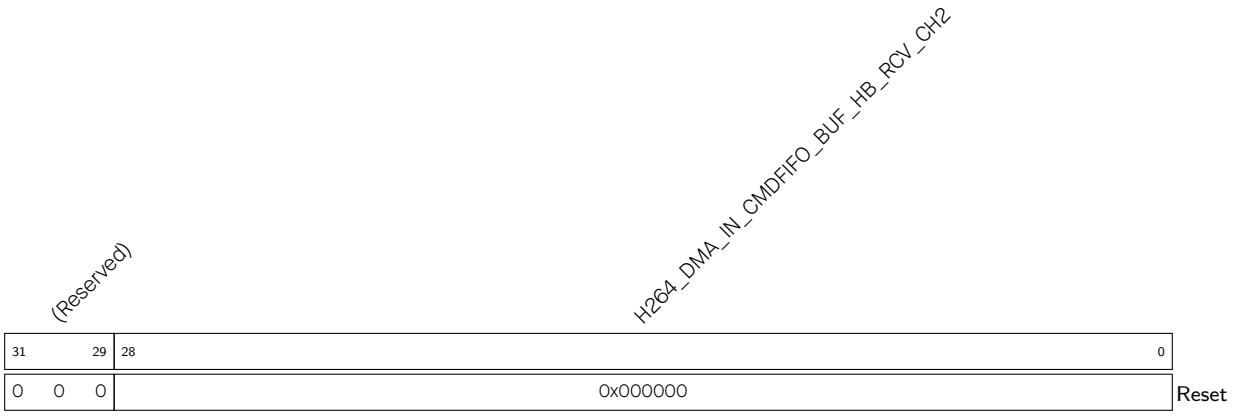
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH2 表示当前 AXI 请求中的数据量。(RO)

Register 41.301. H264_DMA_IN_XADDR_CH2_REG (0x0788)



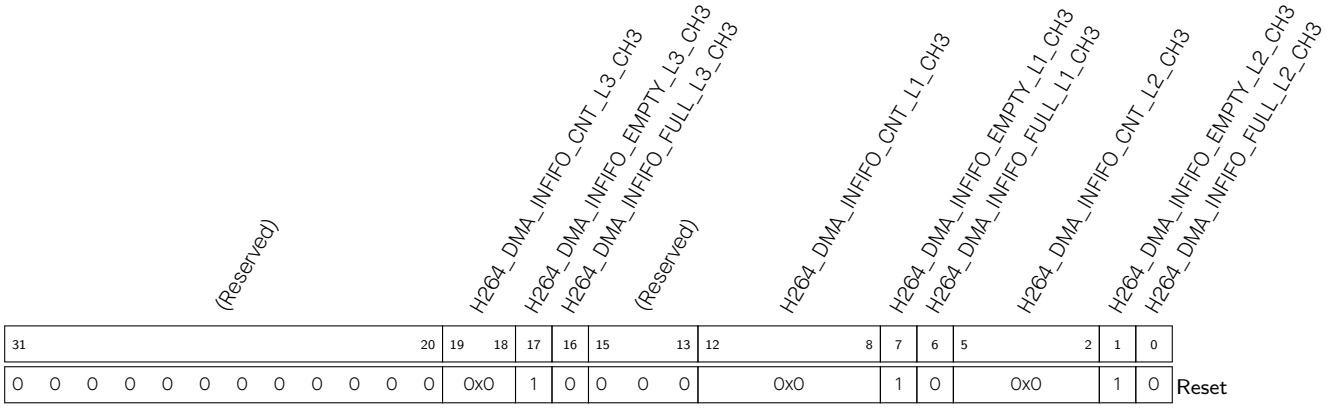
H264_DMA_IN_CMDFIFO_XADDR_CH2 表示当前 AXI 请求的地址。(RO)

Register 41.302. H264_DMA_IN_BUF_HB_RCV_CH2_REG (0x078C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH2 表示已请求的数据量。(RO)

Register 41.303. H264_DMA_INFIFO_STATUS_CH3_REG (0x0814)



H264_DMA_INFIFO_FULL_L2_CH3 表示 RX 通道 3 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH3 表示 RX 通道 3 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L2_CH3 表示 RX 通道 3 的 l2 FIFO 中的数据量。测量单位：16 字节。(RO)

H264_DMA_INFIFO_FULL_L1_CH3 表示 RX 通道 3 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH3 表示 RX 通道 3 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L1_CH3 表示 RX 通道 3 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_INFIFO_FULL_L3_CH3 表示 RX 通道 3 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH3 表示 RX 通道 3 的 l3 FIFO 是否为空。

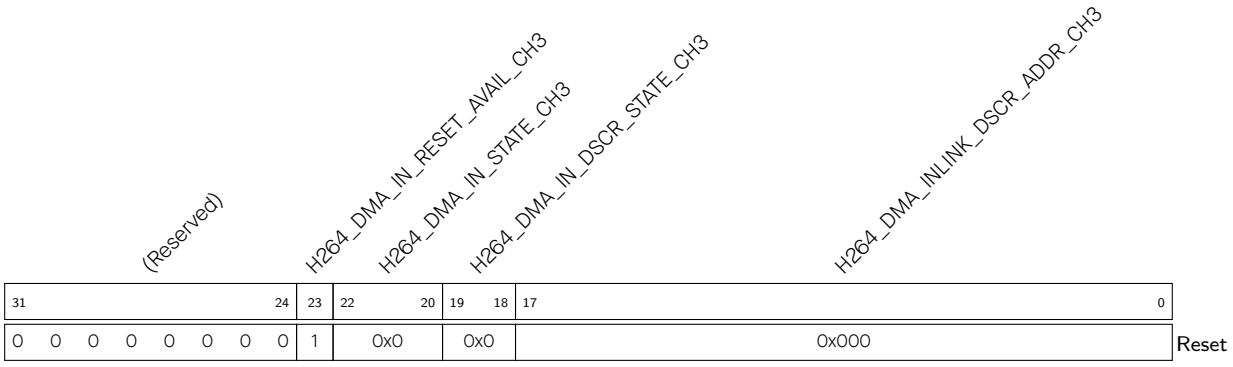
0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L3_CH3 表示 RX 通道 3 的 l3 FIFO 中的数据量。测量单位：8 字节。(RO)

Register 41.304. H264_DMA_IN_STATE_CH3_REG (0x0824)



H264_DMA_INLINK_DSCR_ADDR_CH3 表示 RX 通道 3 的当前 inlink 描述符的地址。(RO)

H264_DMA_IN_DSCR_STATE_CH3 表示 RX 通道 3 的描述符状态机的当前状态。(RO)

H264_DMA_IN_STATE_CH3 表示 RX 通道 3 的当前控制模块状态机状态。(RO)

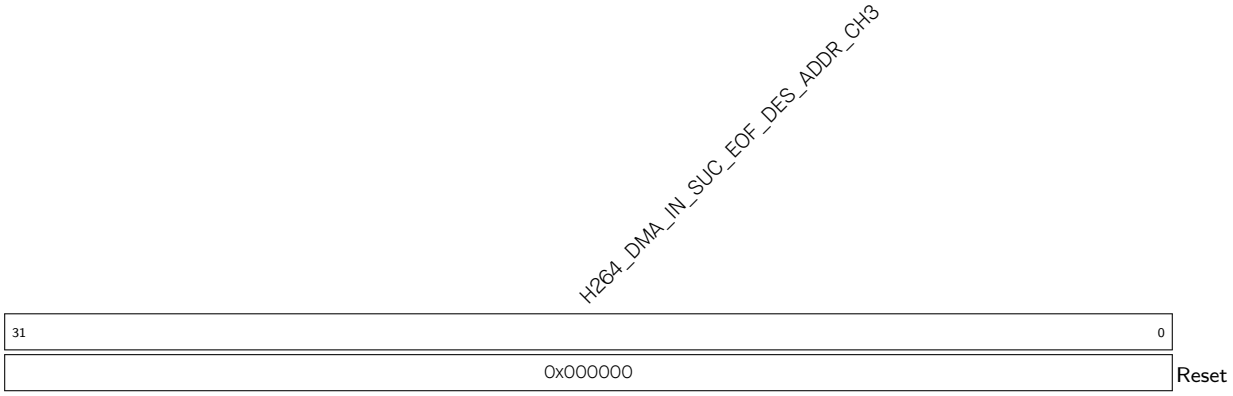
H264_DMA_IN_RESET_AVAIL_CH3 表示是否可以安全地重置通道。

0：不安全

1：安全

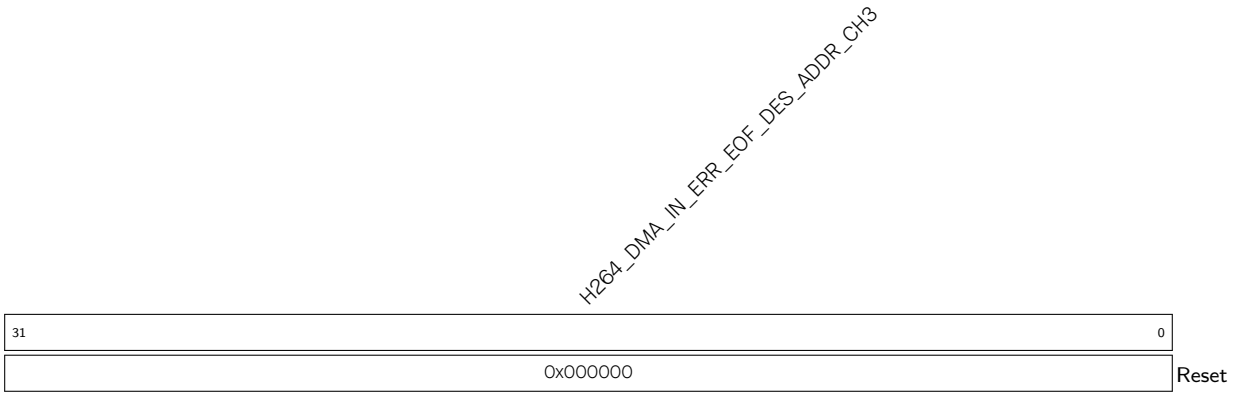
(RO)

Register 41.305. H264_DMA_IN_SUC_EOF_DES_ADDR_CH3_REG (0x0828)



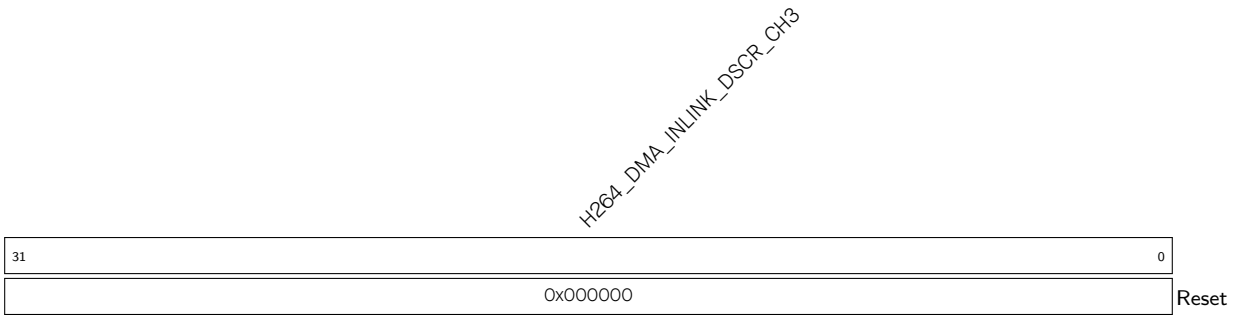
H264_DMA_IN_SUC_EOF_DES_ADDR_CH3 表示当此描述符中的 EOF 位为 1 时的 inlink 描述符的地址。(RO)

Register 41.306. H264_DMA_IN_ERR_EOF_DES_ADDR_CH3_REG (0x082C)



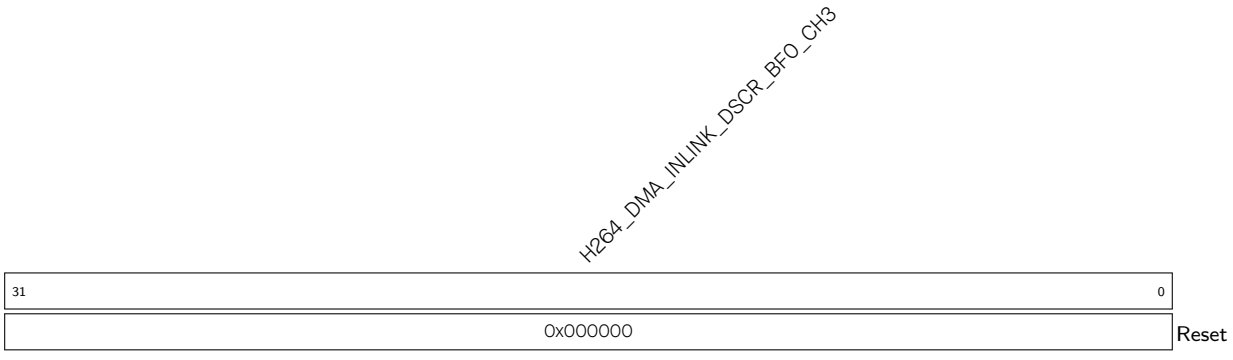
H264_DMA_IN_ERR_EOF_DES_ADDR_CH3 表示当前数据接收过程中出现错误时 inlink 描述符的地址。(RO)

Register 41.307. H264_DMA_IN_DSCR_CH3_REG (0x0830)



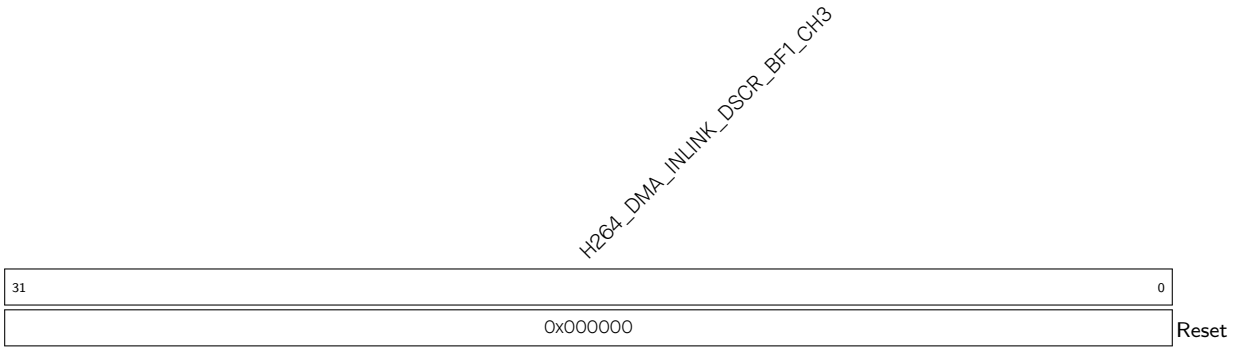
H264_DMA_INLINK_DSCR_CH3 表示由当前预读描述符指向的下一个描述符 x+1 的地址。(RO)

Register 41.308. H264_DMA_IN_DSCR_BF0_CH3_REG (0x0834)



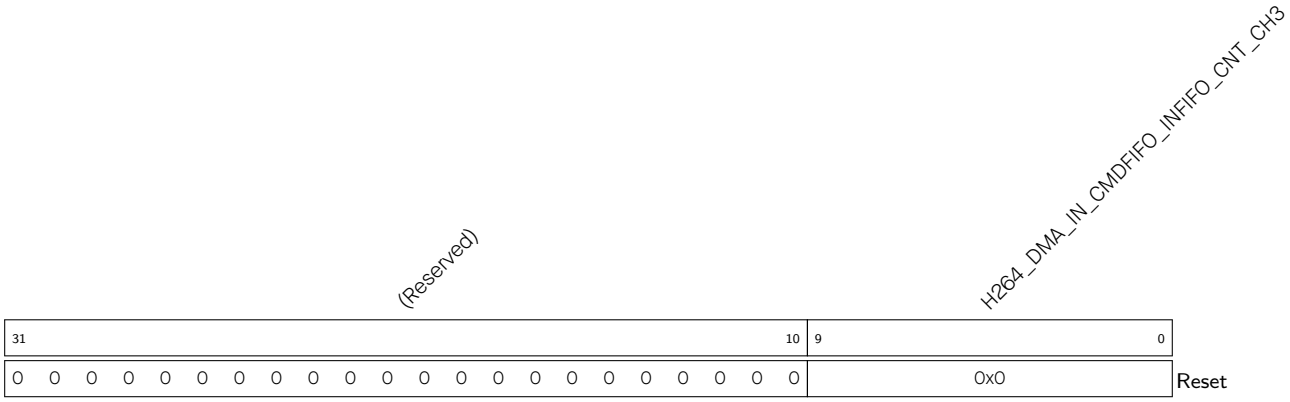
H264_DMA_INLINK_DSCR_BF0_CH3 表示预读的当前描述符 x 的地址。(RO)

Register 41.309. H264_DMA_IN_DSCR_BF1_CH3_REG (0x0838)



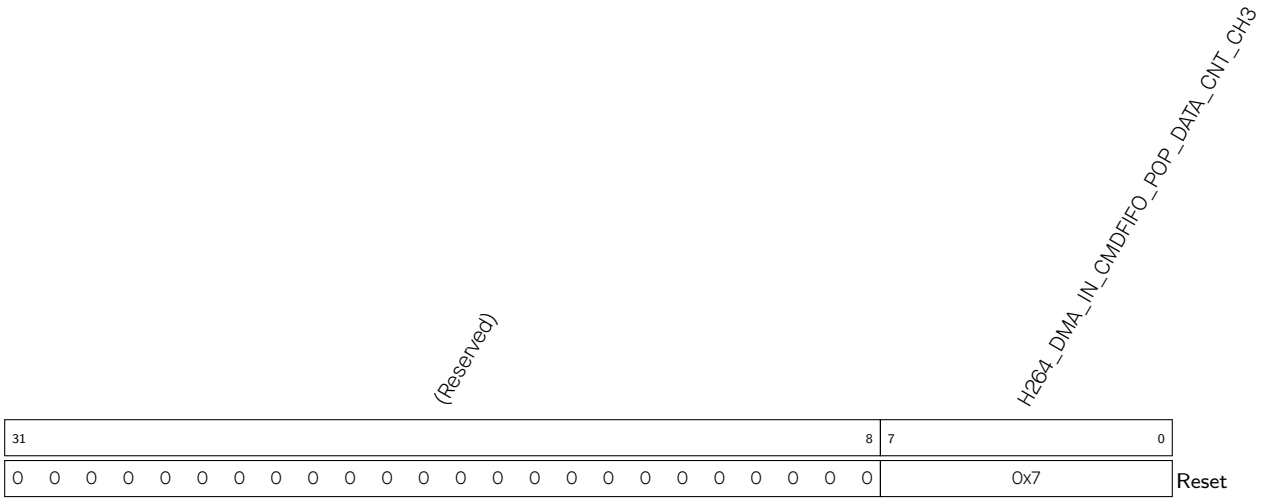
H264_DMA_INLINK_DSCR_BF1_CH3 表示预读的前一个描述符 x-1 的地址。(RO)

Register 41.310. H264_DMA_IN_FIFO_CNT_CH3_REG (0x0880)



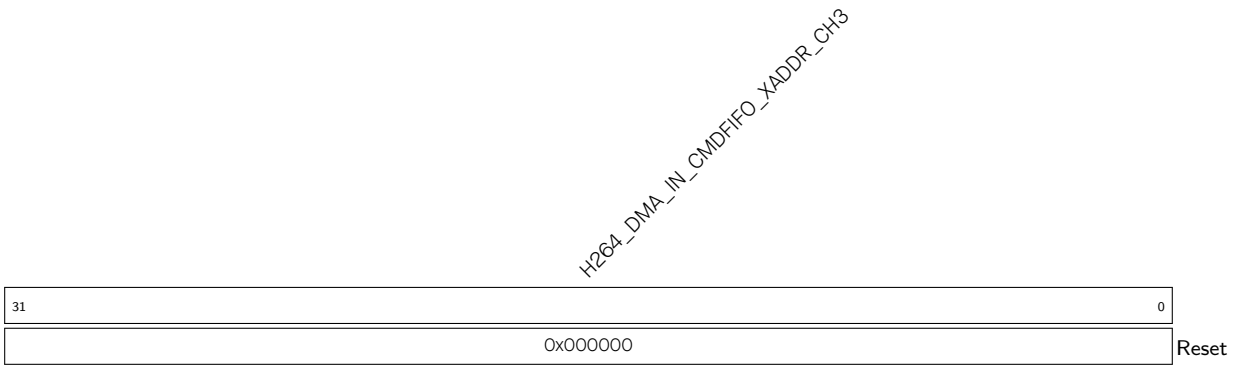
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH3 表示 FIFO 中剩余的请求数据。(RO)

Register 41.311. H264_DMA_IN_POP_DATA_CNT_CH3_REG (0x0884)



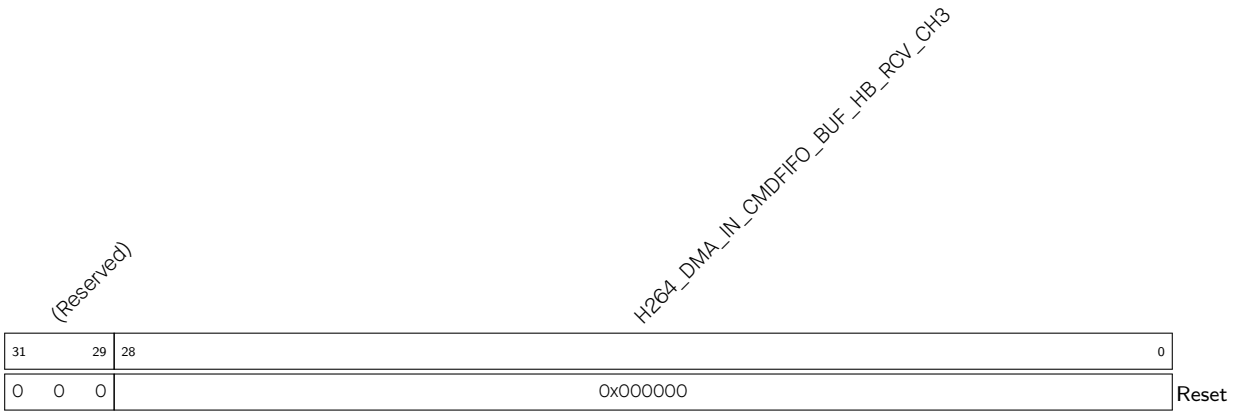
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH3 表示当前 AXI 请求中的数据量。(RO)

Register 41.312. H264_DMA_IN_XADDR_CH3_REG (0x0888)



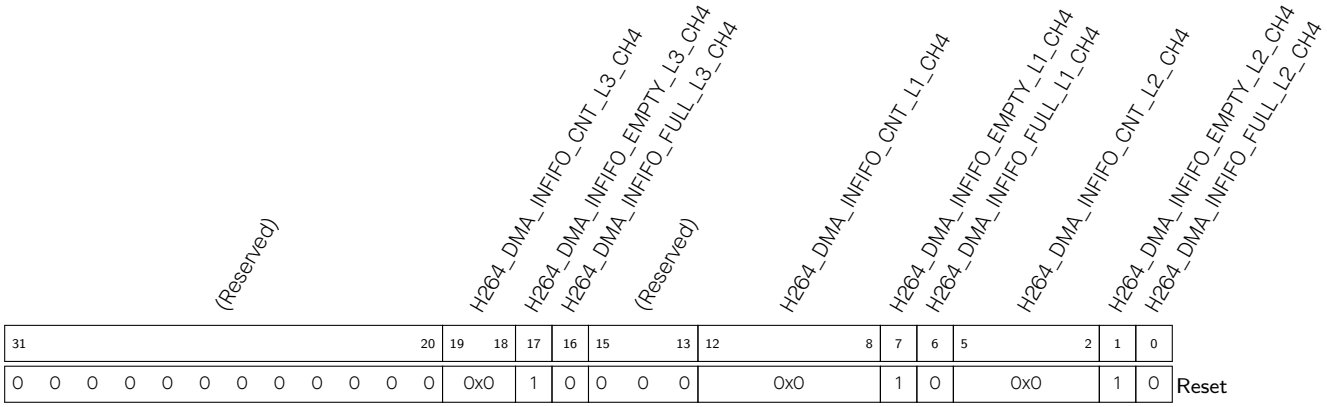
H264_DMA_IN_CMDFIFO_XADDR_CH3 表示当前 AXI 请求的地址。(RO)

Register 41.313. H264_DMA_IN_BUF_HB_RCV_CH3_REG (0x088C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH3 表示已请求的数据量。(RO)

Register 41.314. H264_DMA_INFIFO_STATUS_CH4_REG (0x0914)



H264_DMA_INFIFO_FULL_L2_CH4 表示 RX 通道 4 的 l2 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L2_CH4 表示 RX 通道 4 的 l2 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L2_CH4 表示 RX 通道 4 的 l2 FIFO 中的数据量。测量单位：16 字节。(RO)

H264_DMA_INFIFO_FULL_L1_CH4 表示 RX 通道 4 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH4 表示 RX 通道 4 的 l1 FIFO 是否为空。

0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L1_CH4 表示 RX 通道 4 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

H264_DMA_INFIFO_FULL_L3_CH4 表示 RX 通道 4 的 l3 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L3_CH4 表示 RX 通道 4 的 l3 FIFO 是否为空。

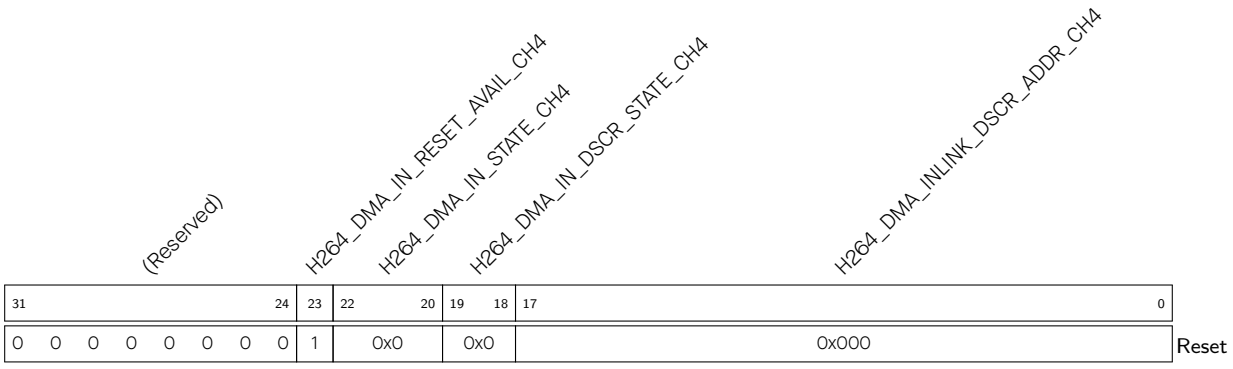
0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L3_CH4 表示 RX 通道 4 的 l3 FIFO 中的数据量。测量单位：8 字节。(RO)

Register 41.315. H264_DMA_IN_STATE_CH4_REG (0x0924)



H264_DMA_INLINK_DSCR_ADDR_CH4 表示 RX 通道 4 的当前 inlink 描述符的地址。(RO)

H264_DMA_IN_DSCR_STATE_CH4 表示 RX 通道 4 的描述符状态机的当前状态。(RO)

H264_DMA_IN_STATE_CH4 表示 RX 通道 4 的当前控制模块状态机状态。(RO)

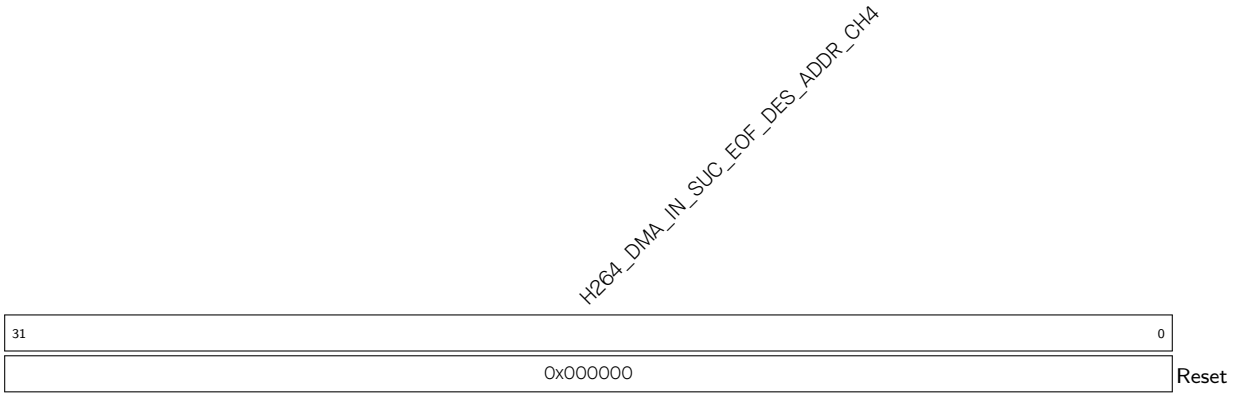
H264_DMA_IN_RESET_AVAIL_CH4 表示是否可以安全地重置通道。

0：不安全

1：安全

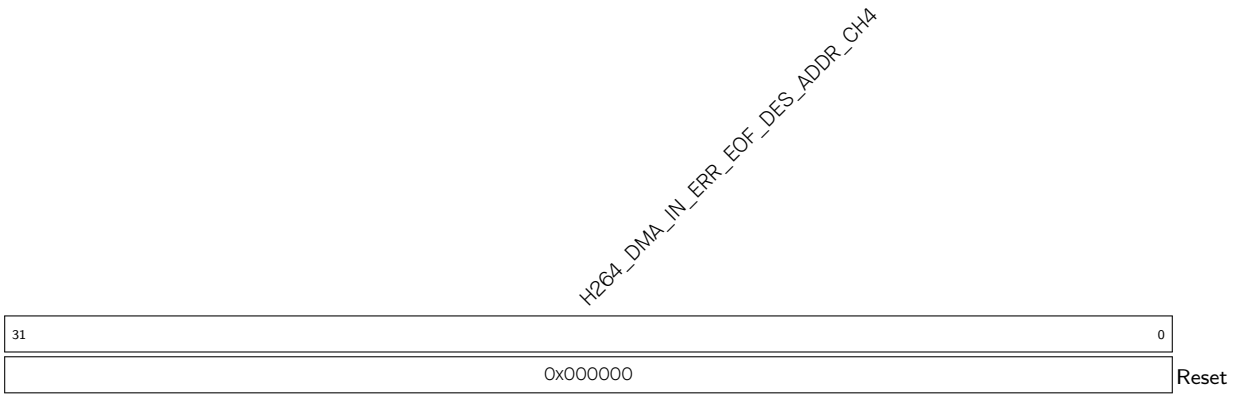
(RO)

Register 41.316. H264_DMA_IN_SUC_EOF_DES_ADDR_CH4_REG (0x0928)



H264_DMA_IN_SUC_EOF_DES_ADDR_CH4 表示当此描述符中的 EOF 位为 1 时的 inlink 描述符的地址。(RO)

Register 41.317. H264_DMA_IN_ERR_EOF_DES_ADDR_CH4_REG (0x092C)



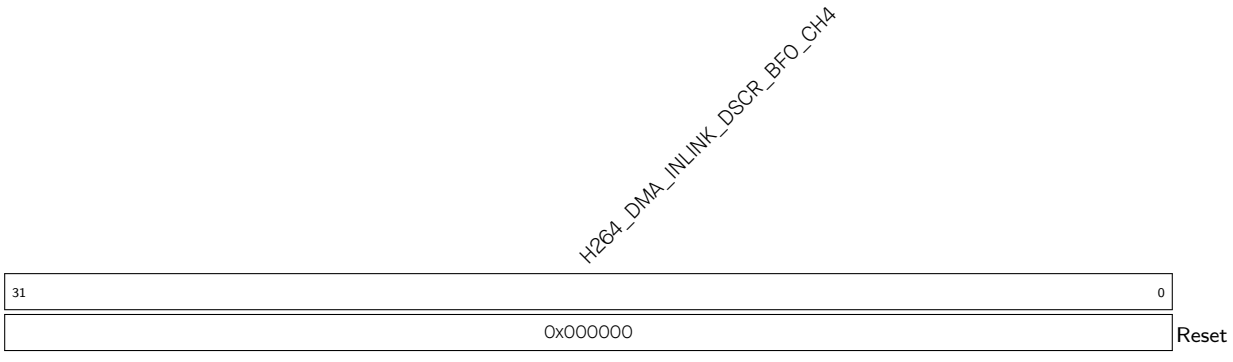
H264_DMA_IN_ERR_EOF_DES_ADDR_CH4 表示当前数据接收过程中出现错误时 inlink 描述符的地址。(RO)

Register 41.318. H264_DMA_IN_DSCR_CH4_REG (0x0930)



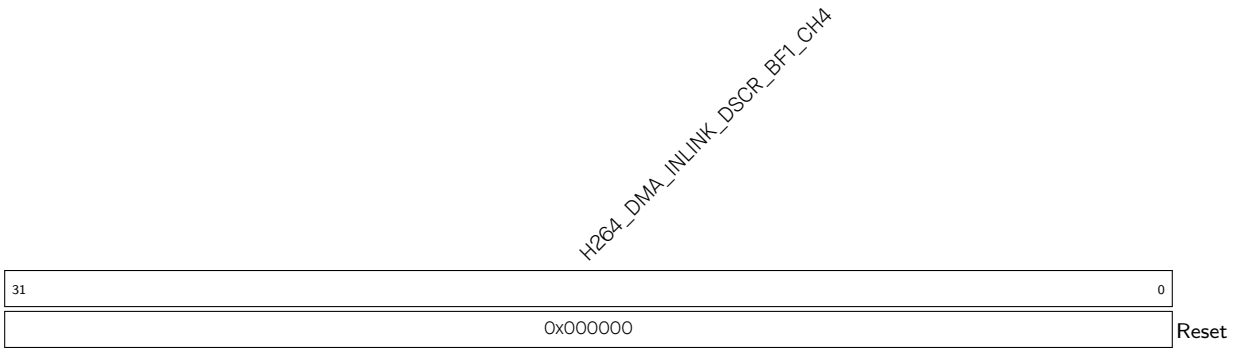
H264_DMA_INLINK_DSCR_CH4 表示由当前预读描述符指向的下一个描述符 x+1 的地址。(RO)

Register 41.319. H264_DMA_IN_DSCR_BF0_CH4_REG (0x0934)



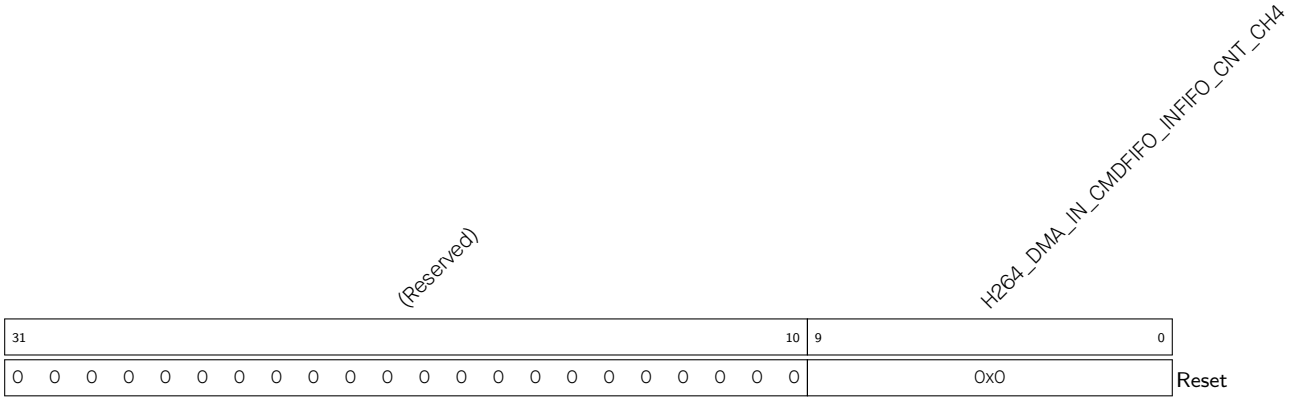
H264_DMA_INLINK_DSCR_BF0_CH4 表示预读的当前描述符 x 的地址。(RO)

Register 41.320. H264_DMA_IN_DSCR_BF1_CH4_REG (0x0938)



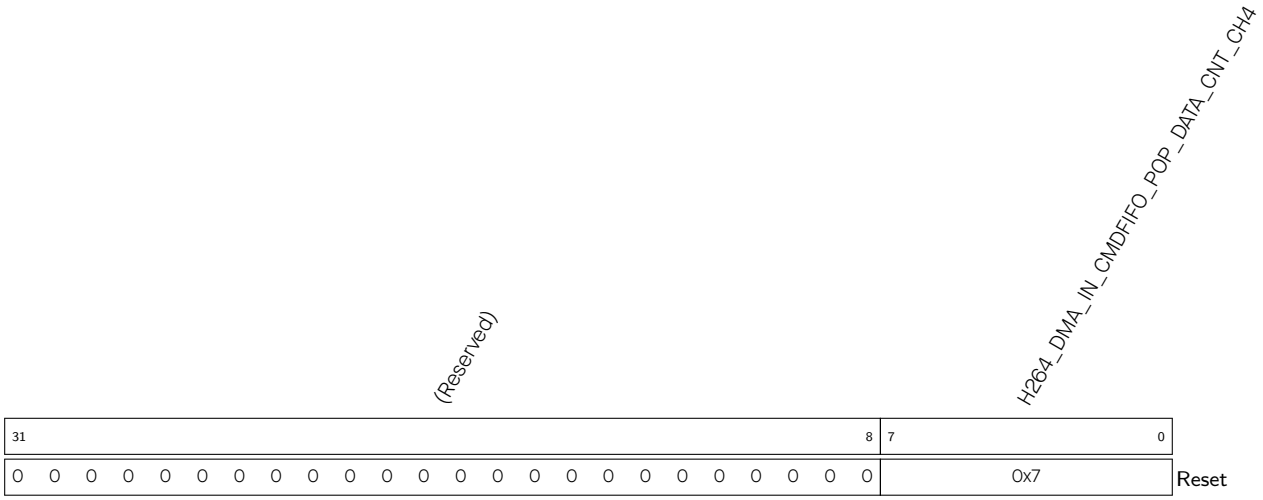
H264_DMA_INLINK_DSCR_BF1_CH4 表示预读的前一个描述符 x-1 的地址。(RO)

Register 41.321. H264_DMA_IN_FIFO_CNT_CH4_REG (0x0980)



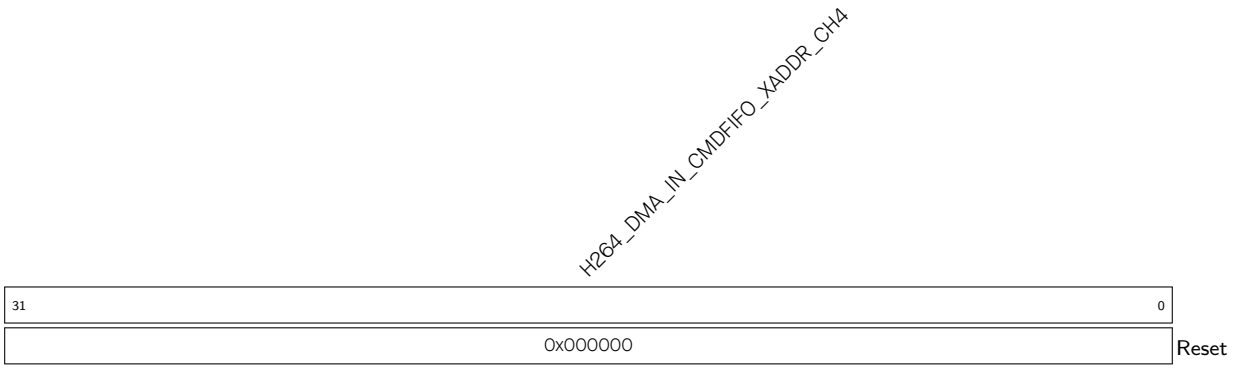
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH4 表示 FIFO 中剩余的请求数据。(RO)

Register 41.322. H264_DMA_IN_POP_DATA_CNT_CH4_REG (0x0984)



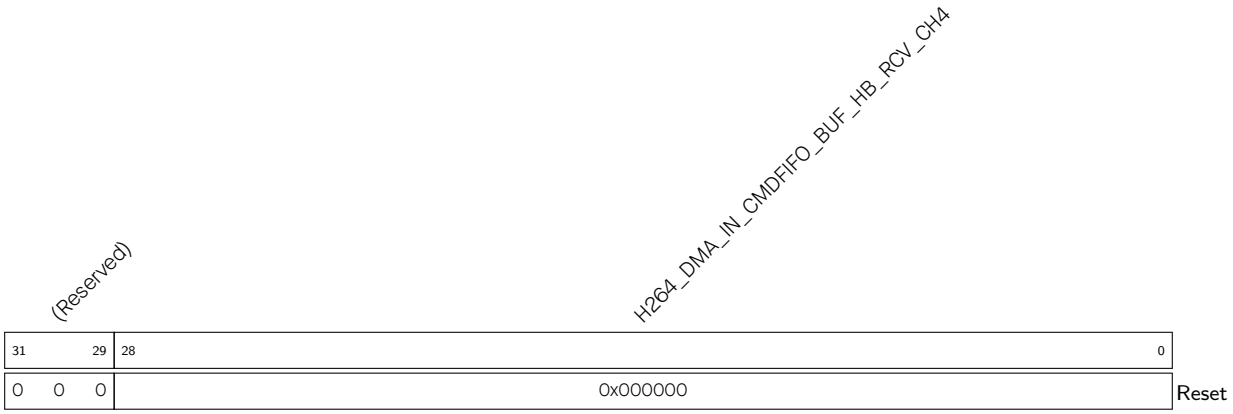
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH4 表示当前 AXI 请求中的数据量。(RO)

Register 41.323. H264_DMA_IN_XADDR_CH4_REG (0x0988)



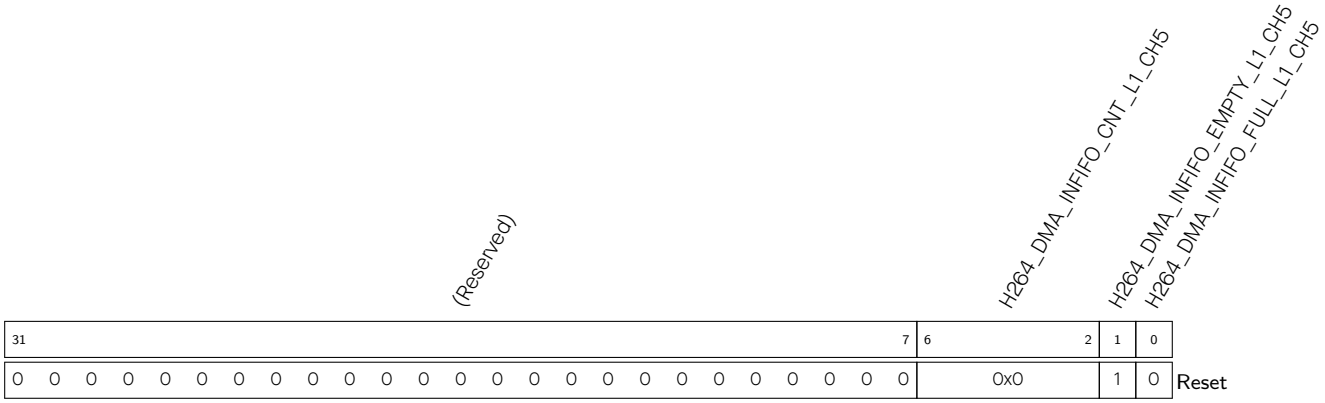
H264_DMA_IN_CMDFIFO_XADDR_CH4 表示当前 AXI 请求的地址。(RO)

Register 41.324. H264_DMA_IN_BUF_HB_RCV_CH4_REG (0x098C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH4 表示已请求的数据量。(RO)

Register 41.325. H264_DMA_INFIFO_STATUS_CH5_REG (0x0A20)



H264_DMA_INFIFO_FULL_L1_CH5 表示 RX 通道 5 的 l1 FIFO 是否已满。

0：未满

1：已满

(RO)

H264_DMA_INFIFO_EMPTY_L1_CH5 表示 RX 通道 5 的 l1 FIFO 是否为空。

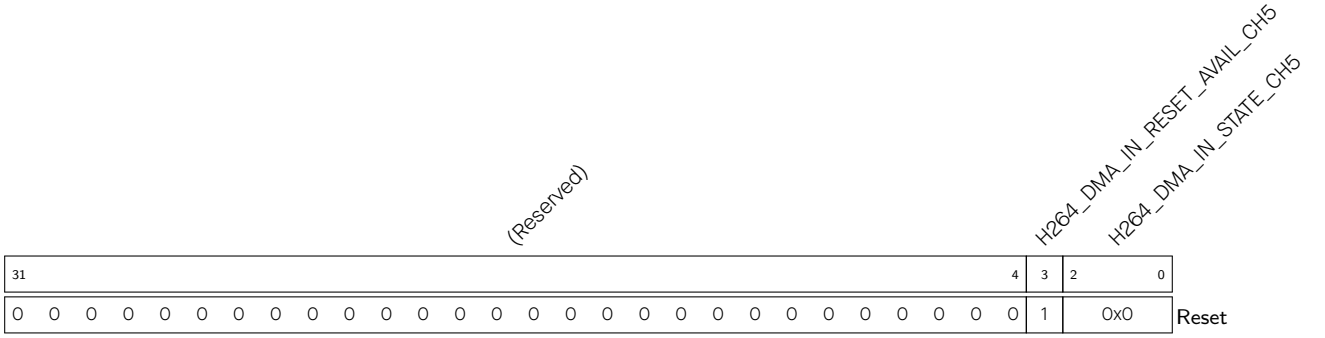
0：不为空

1：空

(RO)

H264_DMA_INFIFO_CNT_L1_CH5 表示 RX 通道 5 的 l1 FIFO 中的数据量。测量单位：字节。(RO)

Register 41.326. H264_DMA_IN_STATE_CH5_REG (0x0A28)



H264_DMA_IN_STATE_CH5 表示 RX 通道 5 的当前控制模块状态机状态。(RO)

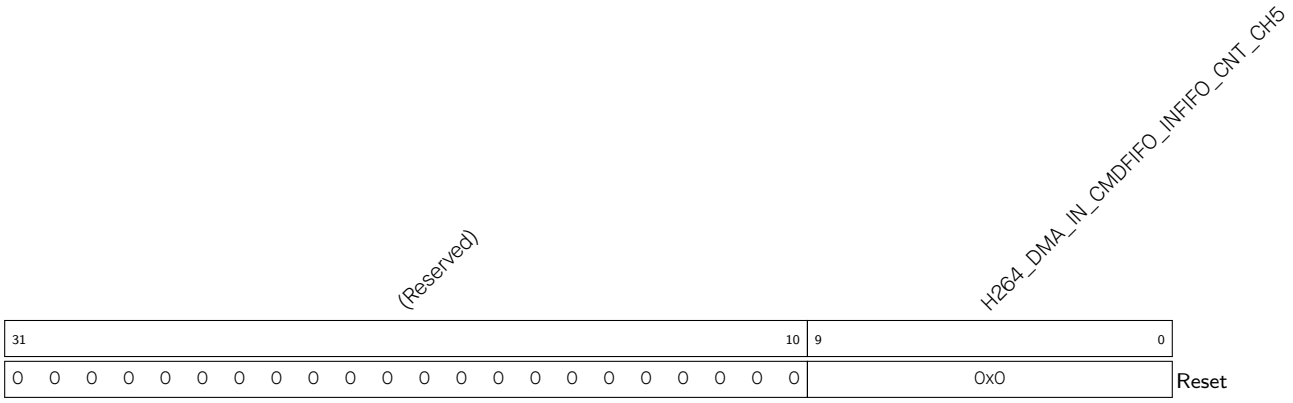
H264_DMA_IN_RESET_AVAIL_CH5 表示是否可以安全地重置通道。

0：不安全

1：安全

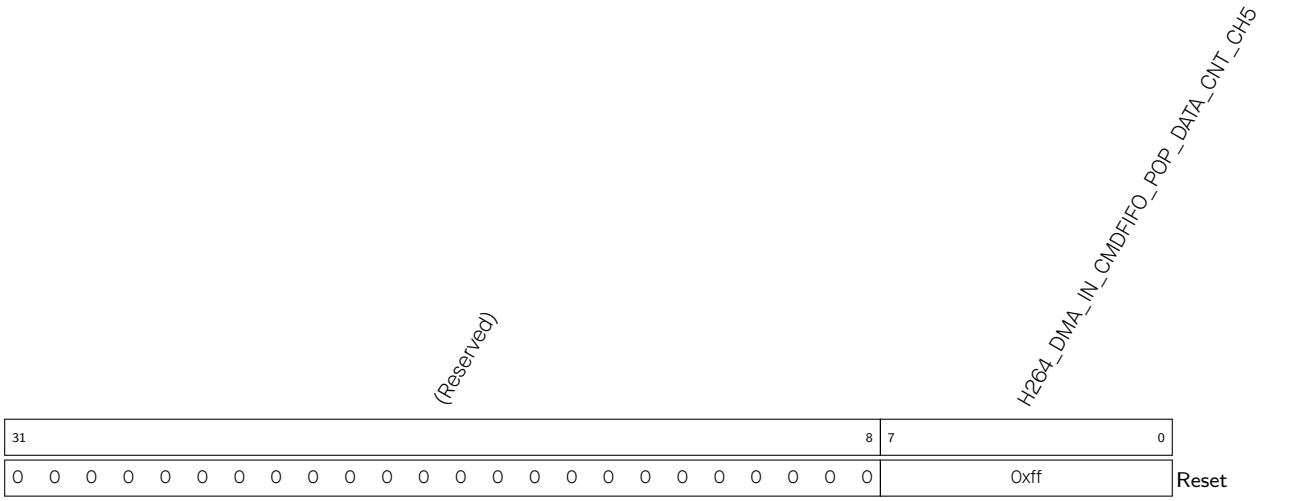
(RO)

Register 41.327. H264_DMA_IN_FIFO_CNT_CH5_REG (0x0A80)



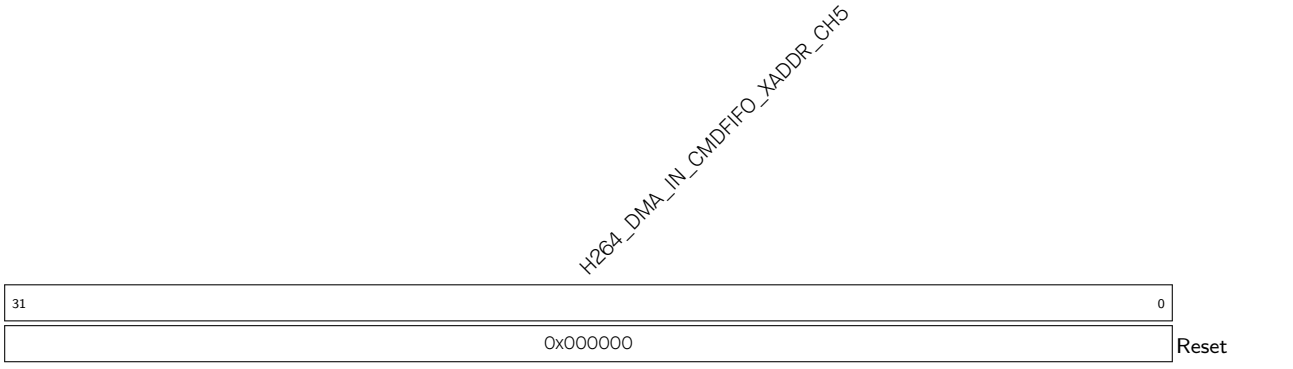
H264_DMA_IN_CMDFIFO_INFIFO_CNT_CH5 表示 FIFO 中剩余的请求数据。(RO)

Register 41.328. H264_DMA_IN_POP_DATA_CNT_CH5_REG (0x0A84)



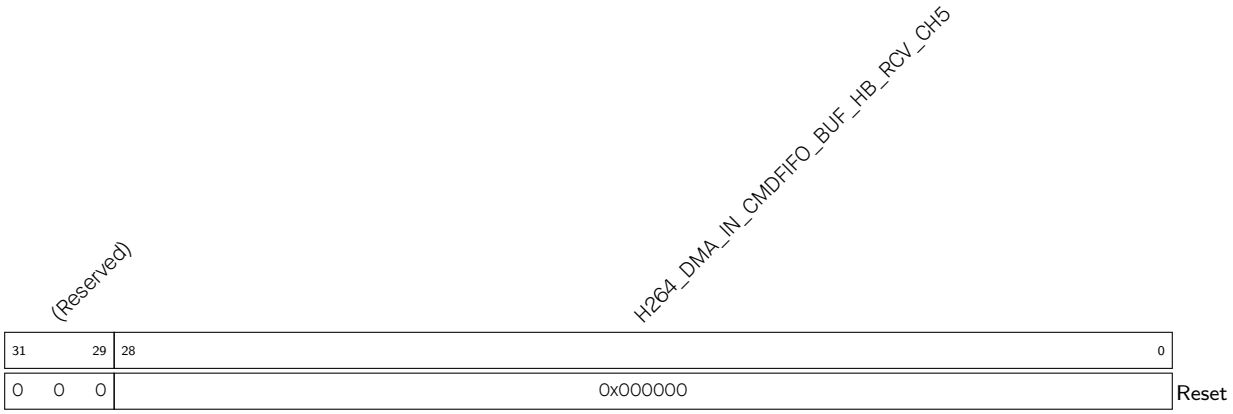
H264_DMA_IN_CMDFIFO_POP_DATA_CNT_CH5 表示当前 AXI 请求中的数据量。(RO)

Register 41.329. H264_DMA_IN_XADDR_CH5_REG (0x0A88)



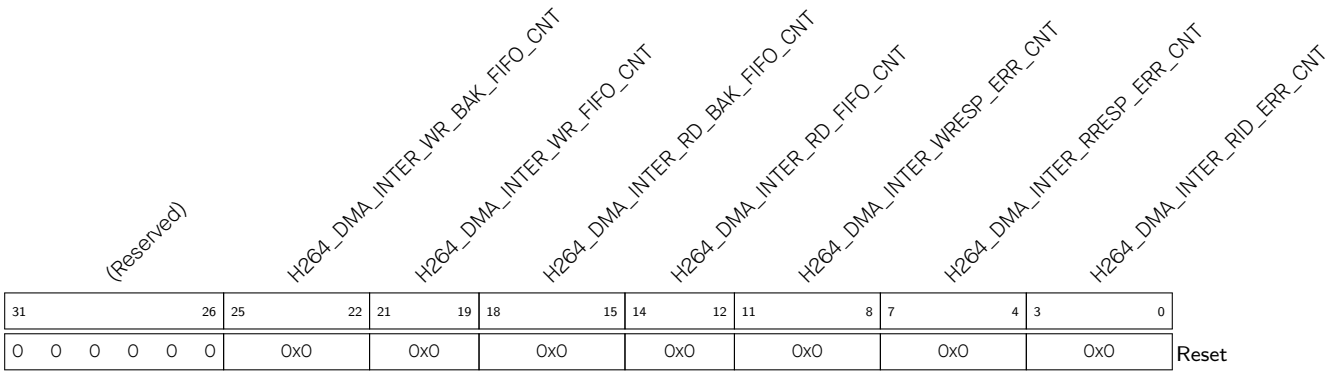
H264_DMA_IN_CMDFIFO_XADDR_CH5 表示当前 AXI 请求的地址。(RO)

Register 41.330. H264_DMA_IN_BUF_HB_RCV_CH5_REG (0x0A8C)



H264_DMA_IN_CMDFIFO_BUF_HB_RCV_CH5 表示已请求的数据量。(RO)

Register 41.331. H264_DMA_INTER_AXI_ERR_REG (0x0B00)



H264_DMA_INTER_RID_ERR_CNT 表示 AXI 读操作中内部存储器 ID 的错误计数。(RO)

H264_DMA_INTER_RRESP_ERR_CNT 表示 AXI 读操作中内部存储器响应的错误计数。(RO)

H264_DMA_INTER_WRESP_ERR_CNT 表示 AXI 写内部存储器响应错误的计数。(RO)

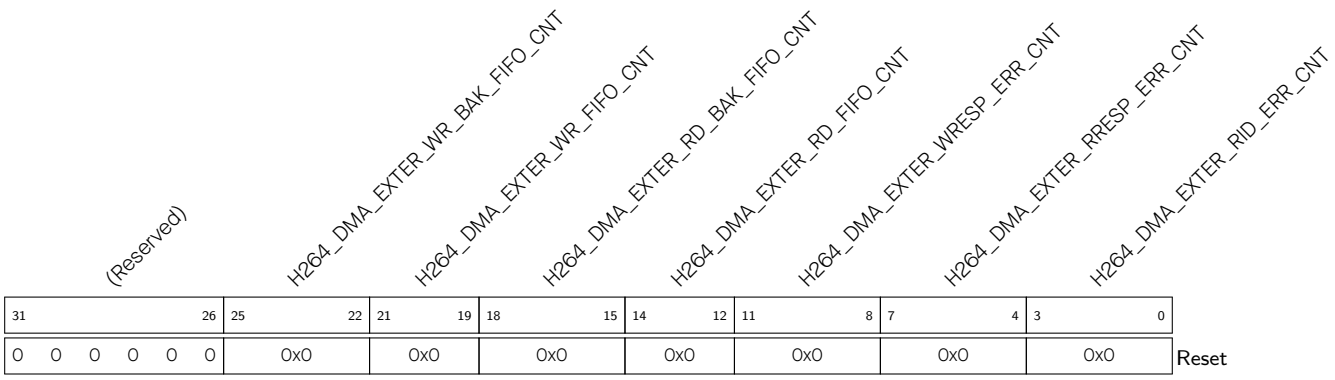
H264_DMA_INTER_RD_FIFO_CNT 表示 AXI 读内部存储器模块中 cmd FIFO 剩余 cmd 的计数。(RO)

H264_DMA_INTER_RD_BAK_FIFO_CNT 表示 AXI 读内部存储器模块中备份 cmd FIFO 剩余 cmd 的计数。(RO)

H264_DMA_INTER_WR_FIFO_CNT 表示 AXI 写内部存储器模块中 cmd FIFO 剩余 cmd 的计数。(RO)

H264_DMA_INTER_WR_BAK_FIFO_CNT 表示 AXI 写内部存储器模块中备份 cmd FIFO 剩余 cmd 的计数。(RO)

Register 41.332. H264_DMA_EXTER_AXI_ERR_REG (0x0B04)



H264_DMA_EXTER_RID_ERR_CNT 表示 AXI 读取外部存储 ID 中的错误计数。(RO)

H264_DMA_EXTER_RRESP_ERR_CNT 表示在 AXI 读取操作中外部存储响应的错误计数。(RO)

H264_DMA_EXTER_WRESP_ERR_CNT 表示 AXI 写入外部存储响应错误的计数。(RO)

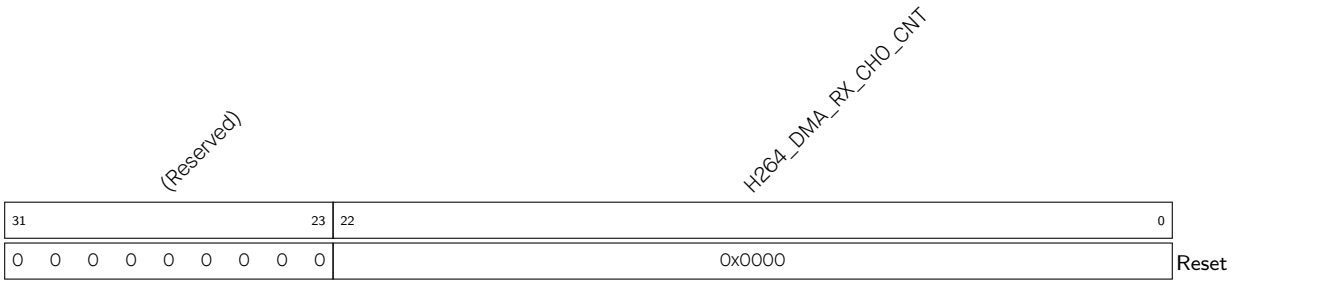
H264_DMA_EXTER_RD_FIFO_CNT 表示在 AXI 读取外部存储模块中命令 FIFO 剩余命令的计数。(RO)

H264_DMA_EXTER_RD_BAK_FIFO_CNT 表示在 AXI 读取外部存储模块中备份命令 FIFO 剩余命令的计数。(RO)

H264_DMA_EXTER_WR_FIFO_CNT 表示在 AXI 写入外部存储模块中命令 FIFO 剩余命令的计数。(RO)

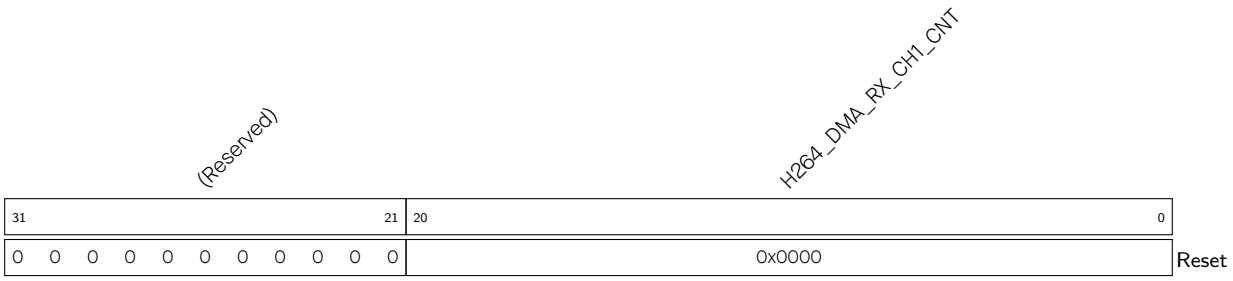
H264_DMA_EXTER_WR_BAK_FIFO_CNT 表示在 AXI 写入外部存储模块中备份命令 FIFO 剩余命令的计数。(RO)

Register 41.333. H264_DMA_RX_CH0_COUNTER_REG (0x0B54)



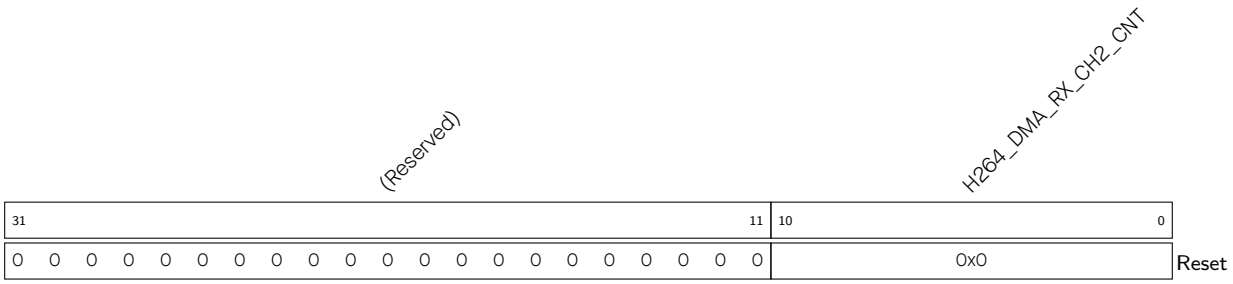
H264_DMA_RX_CH0_CNT 表示 RX 通道 0 计数器寄存器。(RO)

Register 41.334. H264_DMA_RX_CH1_COUNTER_REG (0x0B58)



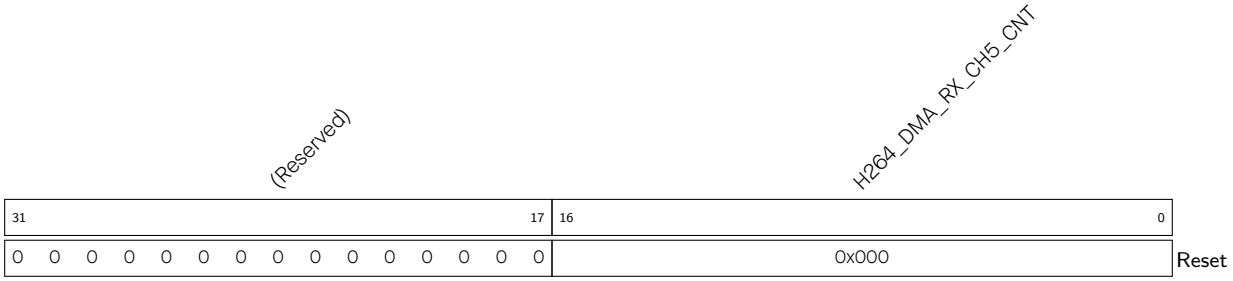
H264_DMA_RX_CH1_CNT 表示 RX 通道 1 计数器寄存器。(RO)

Register 41.335. H264_DMA_RX_CH2_COUNTER_REG (0x0B5C)



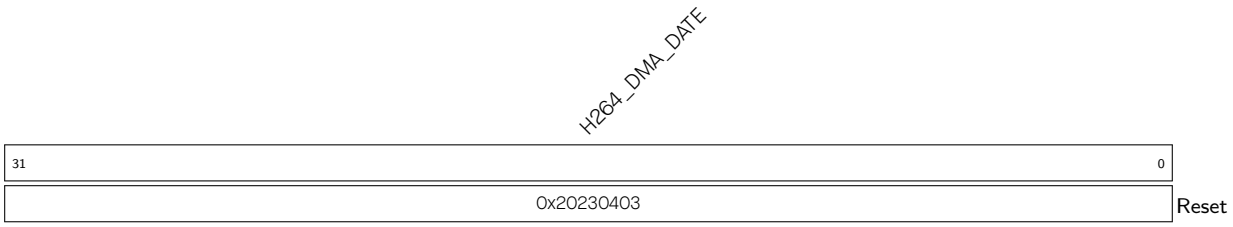
H264_DMA_RX_CH2_CNT 表示 RX 通道 2 计数器寄存器。(RO)

Register 41.336. H264_DMA_RX_CH5_COUNTER_REG (0x0B60)



H264_DMA_RX_CH5_CNT 表示 RX 通道 5 计数器寄存器。(RO)

Register 41.337. H264_DMA_DATE_REG (0x0B3C)



H264_DMA_DATE 版本控制寄存器。(R/W)

第 42 章

MIPI CSI

42.1 概述

ESP32-P4 的 MIPI CSI 模块为智能手机、平板电脑及其他便携式电子设备提供与摄像头模块连接的标准接口。该接口连接摄像头与移动设备应用处理器，旨在实现高速数据传输，确保摄像头与主机系统之间的高效通信。

ESP32-P4 MIPI CSI 由 MIPI RX D-PHY 接口和 MIPI CSI-2 主机控制器组成，提供一个时钟通道和两个数据通道，每个通道支持高达 1.5 Gbps 的数据传输速率，可与符合 MIPI CSI-2 规范 的摄像头传感器通信。

说明：

MIPI CSI-2 规范 定义了图像应用处理器与摄像头之间的通信。它是 MIPI 联盟标准定义的通信协议的一部分，用于移动系统中的芯片间通信。

42.2 术语

本小节介绍描述 MIPI CSI 功能的术语。

MIPI CSI-2	MIPI 联盟摄像头串行接口 2 (CSI-2) 规范。该接口通过 PHY 协议接口 (PPI) 从 MIPI RX D-PHY 接收摄像头数据，并通过 32 位图像接口将数据输出到图像信号处理器（ISP）。
MIPI D-PHY	MIPI 联盟规范的源同步物理层。该接口实现了 MIPI D-PHY 接口的通用通道物理层，通过一条时钟通道和两条数据通道从 MIPI 摄像头传感器接收时钟和数据。
PPI	MIPI 联盟规范中定义的 PHY 协议接口。该接口连接 CSI-2 主机控制器和 D-PHY。
ISP	图像信号处理器，用于处理从摄像头接收的图像。有关 ISP 的详细信息，请参阅章节 38 图像信号处理器 *(ISP)*。
发射器	发射数据的设备，通常为 MIPI 摄像头。
接收器	接收数据的设备，在本章中为 ESP32-P4 芯片。
时钟通道	MIPI D-PHY 接口的时钟通道，用于传输高速差分时钟信号。
数据通道	MIPI D-PHY 接口的数据通道，用于传输数据和命令。
线路	MIPI D-PHY 接口数据通道的线路，每一条数据通道都包含 P、N 两条线路。
序列	通过数据通道接收的一串状态码，控制 MIPI RX D-PHY 从控制模式进入其他模式。
控制模式	MIPI RX D-PHY 的工作模式之一，此模式用于接收来自发射器的请求。所有其他模式结束后都会回到该模式
高速数据接收模式	MIPI RX D-PHY 的工作模式之一，此模式下以突发传输的方式接收数据。

Escape 模式 MIPI RX D-PHY 的工作模式之一，此模式支持数据通道进行低速异步通信，以保持较低功耗状态。

超低功耗状态 (ULPS) MIPI RX D-PHY 的工作模式之一，在此模式下，时钟通道和数据通道均处于空闲状态，处于除关断外的最低功耗。

Test Code 用于管理和测试 MIPI D-PHY 的特定控制或配置代码，例如可用于设置工作模式。

42.3 主要特性

ESP32-P4 MIPI CSI 支持以下特性：

- MIPI RX D-PHY 符合 MIPI D-PHY 接口规范修订版 1.1
- MIPI CSI-2 主机符合 MIPI CSI-2 规范
- 具有两个数据通道，每个通道支持 80 Mbps ~ 1.5 Gbps 的速率范围
- Escape 模式下支持超低功耗状态 (ULPS)
- 支持与 ISP 连接的 32 位图像接口
- 支持多种输入图像格式：
  - RGB888/RGB666/RGB565
  - YUV422/YUV420
  - RAW8/RAW10/RAW12
- 支持 PHY 层、数据包层、线路层和帧层的错误检测与纠正
- 支持通过数据加扰以减少电磁干扰 (EMI)

说明：
ESP32-P4 不支持 MIPI 规范中定义的不同虚拟通道，因此数据包头中的虚拟通道号将被忽略。

42.4 结构概览

图 42.4-1 展示了 MIPI CSI 的架构及其与其他系统组件的连接。MIPI CSI 包括以下组件：

- **MIPI RX D-PHY**：实现物理链路层，包括一条时钟通道 (CP/CN) 和两条数据通道 (D0P/D0N, D1P/D1N) 用于连接摄像头。
- **MIPI CSI-2 主机**：实现 CSI-2 协议，从 D-PHY 接收摄像头数据并通过图像接口输出，包括：
  - **流水线**：寄存 PPI 信号。
  - **解扰器**：启用后可将 PPI 加扰数据转换为原始值。
  - **PHY 适配层**：管理 PHY 接口，包括 PHY 错误处理。
  - **数据包分析器**：处理从通道接收的数据，包括数据通道合并、头部解码、各种错误检测和纠正等。
  - **图像接口**：将像素重新排序为 32 位图像接口数据，并生成时序精确的视频同步信号。

  - **错误管理**：监测并通知 CSI-2 链路上的错误状态。
  - **寄存器库**：提供对配置和控制寄存器的访问。
- **PPI 接口**：连接 MIPI RX D-PHY 和 MIPI CSI-2 主机控制器。
- **Test Code 接口**：将配置细节从 MIPI CSI-2 主机控制器发送到 MIPI RX D-PHY。

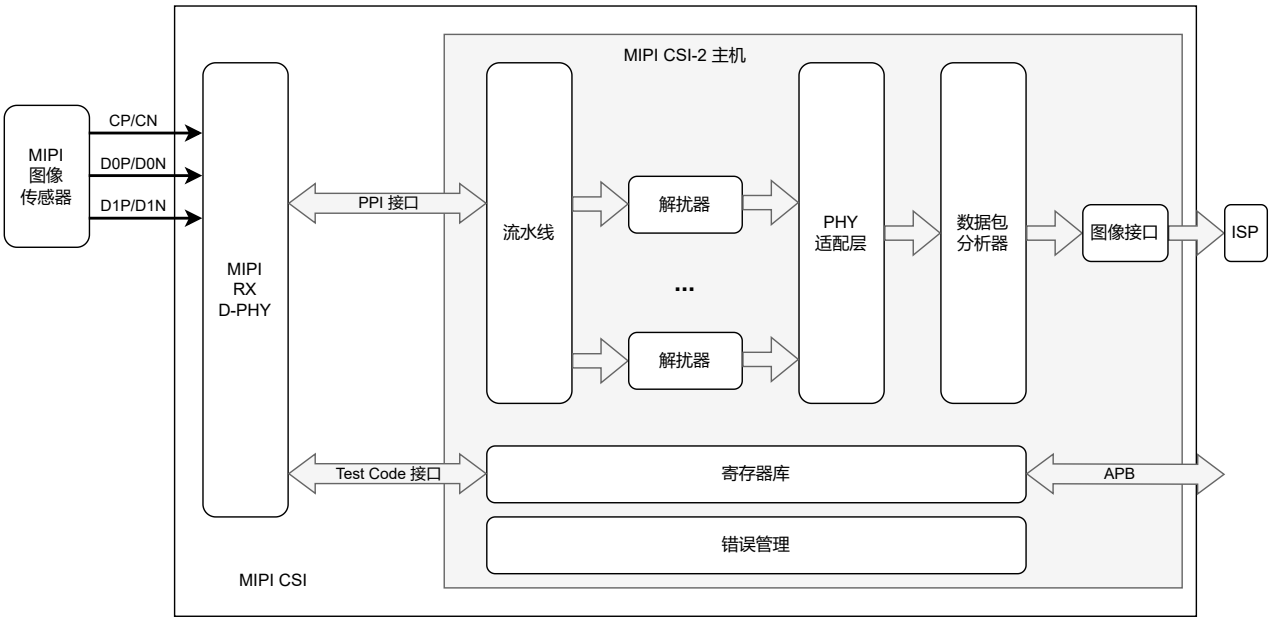


图 42.4-1. MIPI CSI 架构

42.5 功能描述

本小节介绍 MIPI RX D-PHY 的操作状态和模式，以及支持 MIPI CSI 使用的主要功能。有关详细配置，请参阅小节 42.7 配置流程。

42.5.1 通道状态

发射器通过驱动特定的线电平来决定通道状态。在正常操作中，通道由 HS-TX 或 LP-TX 驱动。

- HS-TX：高速传输。HS-TX 以差分方式驱动通道，具有两种状态：HS-0 和 HS-1。当 HS-TX 驱动通道时，通道始终处于高速数据接收模式。
  - HS-0：高速数据接收模式下传输差分信号 0。
  - HS-1：高速数据接收模式下传输差分信号 1。
- LP-TX：低功耗传输。两个 LP-TX 以单端方式分别独立驱动通道的两条线路，具有四种状态：LP-00、LP-01、LP-10 和 LP-11。当 LP-TX 驱动通道时，通道处于控制模式或 Escape 模式。
  - LP-00：在控制模式和 Escape 模式下，其他 LP 状态 (LP-01, LP-10, LP-11) 之间的过渡状态。在超低功耗状态 (ULPS) 下，通道也是该状态。
  - LP-01：在 Escape 模式下传输信号 0；或从控制模式进入高速数据接收模式的序列的开始信号。进入高速数据接收模式的完整序列为：LP-11 → LP-01 → LP-00

- LP-10：在 Escape 模式下传输信号 1；或从控制模式进入 Escape 模式或 Turnaround 模式的序列的开始信号。
  进入 Escape 模式的完整序列为：LP-11 → LP-10 → LP-00 → LP-01 → LP-00。
  进入 Turnaround 模式的完整序列为：LP-11 → LP-10 → LP-00 → LP-10 → LP-00。
- LP-11：停止状态，该状态为通道的空闲状态。

说明：
Turnaround 模式用于交换数据传输方向，此模式在 ESP32-P4 MIPI CSI 中未被使用，因此本章节不做进一步说明。

图 42.5-1 展示了高速 (HS) 和低功耗 (LP) 传输的通道线路电平。

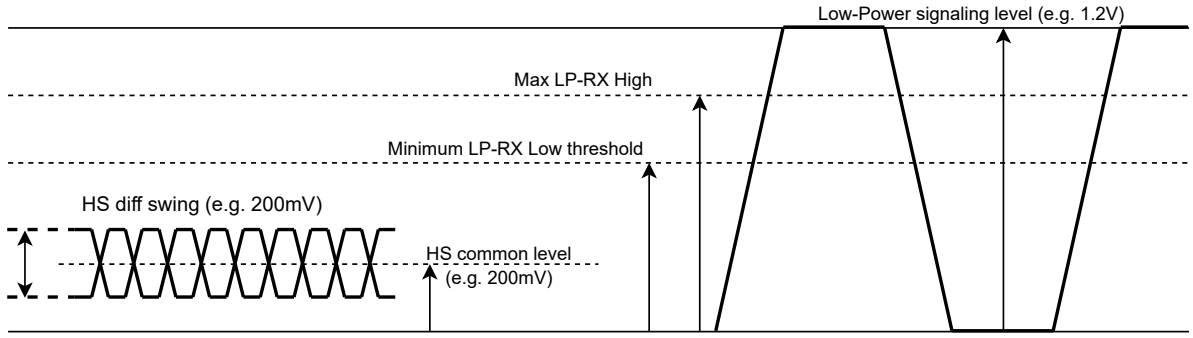


图 42.5-1. MIPI 通道线路电平

表 42.5-1 展示了可用的线路状态。

表 42.5-1. 线路状态描述

状态代码	DP 线	DN 线	不同模式下的对应状态		
			高速数据接收模式	控制模式	Escape 模式
HS-0	HS Low	HS High	0	N/A	N/A
HS-1	HS High	HS Low	1	N/A	N/A
LP-00	LP Low	LP Low	N/A	Bridge	Space
LP-01	LP Low	LP High	N/A	HS-Rqst	0
LP-10	LP High	LP Low	N/A	LP-Rqst	1
LP-11	LP High	LP High	N/A	Stop	N/A

42.5.2 MIPI RX D-PHY 操作

图 42.5-2 展示了 MIPI RX D-PHY 在初始化和激活状态下的各种状态和模式。以下各节详细描述每种状态和模式。

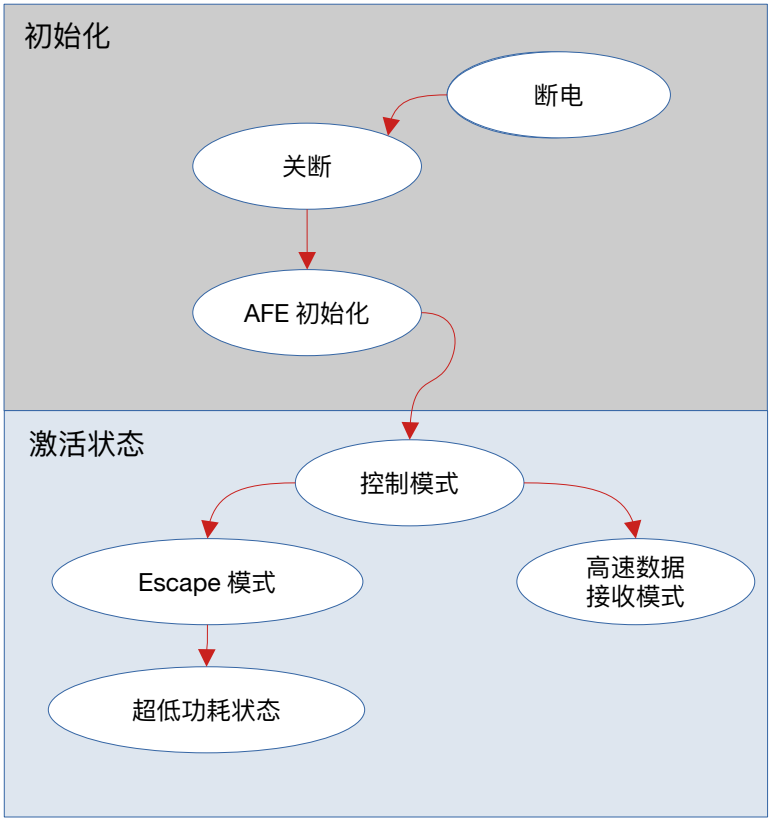


图 42.5-2. MIPI RX D-PHY 操作状态和模式

42.5.2.1 断电状态

在断电状态 (No-Power state) 下，PHY 无供电电压。

当 VDD_HP_*n* (n：0-3) 和 VDD_MIPI_DPHY 电源关闭时，MIPI CSI 的 MIPI RX D-PHY 处于断电状态。

42.5.2.2 关断状态

关断状态 (Shut-Down state) 时，PHY 处于空闲状态。

在关断状态下，PHY 禁用所有模拟模块，重置所有数字逻辑，并将 D0P/D0N、D1P/D1N 和 CP/CN 通道设置为高阻抗 (Hi-Z)。激活关断状态可确保最低功耗。将 CSI_HOST_DPHY_RSTZ 和 CSI_HOST_PHY_SHUTDOWNZ 设置为 0 可启用 PHY 的关断状态。

在退出关断状态之前，需要配置一些参数，包括要使用的通道数量和通道工作频率范围。

MIPI RX D-PHY 由两条数据通道组成。配置 CSI_HOST_N_LANES，选择使用一条或两条数据通道。在退出关断状态之前，这些配置应保持静态或至少稳定。

根据 MIPI RX D-PHY 的具体要求，可能需要执行额外的配置步骤。默认情况下，PHY 在 80 ~ 110 Mbps 的较低范围内工作。为了适配更高的比特率，用户可以在 Test Code 通道 0 的 HS RX 控制 中使用适当的代码调整 HS 频率范围（hsfreqrange 字段）。控制接口与 MIPI RX D-PHY 的其余部分相互独立，因此建议在关断状态下进行频率调整。

表 42.5-2 列出了 MIPI RX D-PHY 支持的不同频率范围。hsfreqrange 字段表示 HS 操作频率范围，可通过测 Test Code 0x44 访问。有关详细配置，请参阅小节 42.5.3 *MIPI RX D-PHY 配置*。

表 42.5-2. MIPI RX D-PHY 支持频率范围

频率范围 (Mbps)	hsfreqrange[5:0]*
80-89	000000
90-99	010000
100-109	100000
110-129	000001
130-139	010001
140-149	100001
150-169	000010
170-179	010010
180-199	100010
200-219	000011
220-239	010011
240-249	100011
250-269	000100
270-299	010100
300-329	000101
330-359	010101
360-399	100101
400-449	000110
450-499	010110
500-549	000111
550-599	010111
600-649	001000
650-699	011000
700-749	001001
750-799	011001
800-849	101001
850-899	111001
900-949	001010
950-999	011010
1000-1049	101010
1050-1099	111010
1100-1149	001011
1150-1199	011011
1200-1249	101011
1250-1299	111011
1300-1349	001100
1350-1399	011100
1400-1449	101100
1450-1500	111100

将 CSI_HOST_DPHY_RSTZ 和 CSI_HOST_PHY_SHUTDOWNZ 置 1 后，MIPI RX D-PHY 退出关断状态，并开始初始化过程。

42.5.2.3 AFE 初始化

在 CSI_HOST_DPHY_RSTZ 和 CSI_HOST_PHY_SHUTDOWNZ 释放后，PHY 开始初始化序列，进入正常运行。建议先释放 CSI_HOST_DPHY_RSTZ，再释放 CSI_HOST_PHY_SHUTDOWNZ。此外，请确保此时 CSI_DPHY_CFG_CLK（详细信息请参考章节 11 复位和时钟 > 表 11.2-5）可用且稳定。

初始化序列包含启用内部模块和执行内部校准等多个步骤。完成这些步骤后，控制权转交给通道，通道通过启用或禁用相应的接收器，处理来自发射器的 LP/HS 请求的功耗管理。

将 CSI_HOST_PHY_STOPSTATEDATA_*n* 和 CSI_HOST_PHY_STOPSTATECLK 设置为 1，执行所有初始化步骤。在此状态下，时钟和数据通道均处于停止状态 (LP-11)。

初始化周期 (TINIT) 是一个依赖于协议的参数，根据规范定义，其最短持续时间为 100 μs。设置 CSI_HOST_PHY_STOPSTATEDATA_*n* 和 CSI_HOST_PHY_STOPSTATECLK 为 1 后，软件应等待超过 100 μs，然后开始摄像头数据接收。

42.5.2.4 控制模式

AFE 初始化完成后，MIPI RX D-PHY 默认保持在控制模式 (Control Mode)，同时在线路上接收 LP-11 状态码（停止状态），直到接收到新请求。在接收请求后，PHY 进入高速数据接收模式、Escape 模式或超低功耗状态。所有请求开始和结束时，线路必须处于停止状态。

42.5.2.5 高速数据接收模式

在高速数据接收模式 (High-Speed Data Reception mode) 下，PHY 以突发传输的方式接收数据。通道仅在突发传输期间以高速模式运行。

如图 42.5-3 所示，一次突发传输包括三部分：低功耗初始化序列、高速数据有效载荷和接收结束序列。

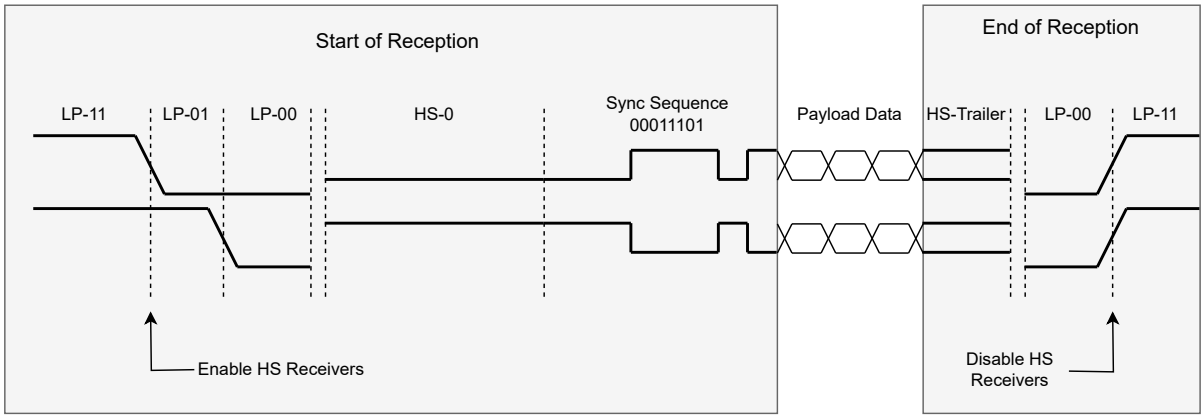


图 42.5-3. MIPI CSI HS 数据接收序列

接收器在接收到 [LP-11 → LP-01 → LP-00] 的低功耗序列状态码后，进入高速模式，然后通过 “00011101” 同步序列实现同步。同步完成后，PHY 接收高速数据，直到在通道上检测到从 LP-00 到 LP-11 的转换。

42.5.2.6 Escape 模式

Escape 模式支持数据通道进行低速异步通信。在 Escape 模式下，通道保持在低功耗状态。在收到 Escape 模式请求序列 [LP-11 → LP-10 → LP-00 → LP-01 → LP-00] 后，数据通道进入此模式。如果通道在达到 LP-00 之

前检测到 LP-11 状态码，请求将被中止，通道将返回到停止状态。进入 Escape 模式后，发射器会发送一个 8 位命令以指示请求的操作。

MIPI CSI 仅支持 Escape 模式下的超低功耗状态 (ULPS)。进入 ULPS 的 8 位命令为 “b00011110”。无效的进入命令会被忽略，接收器将等待，直到发射器返回到停止状态。

超低功耗状态

超低功耗状态 (Ultra Low Power State) 是除关断状态外功耗最低的状态。

发送超低功耗状态进入命令 “00011110”，可在数据通道上激活此模式。此后，时钟通道和数据通道均处于空闲状态 (LP-00)。

图 42.5-4 展示了时钟和数据通道的 ULPS 序列。

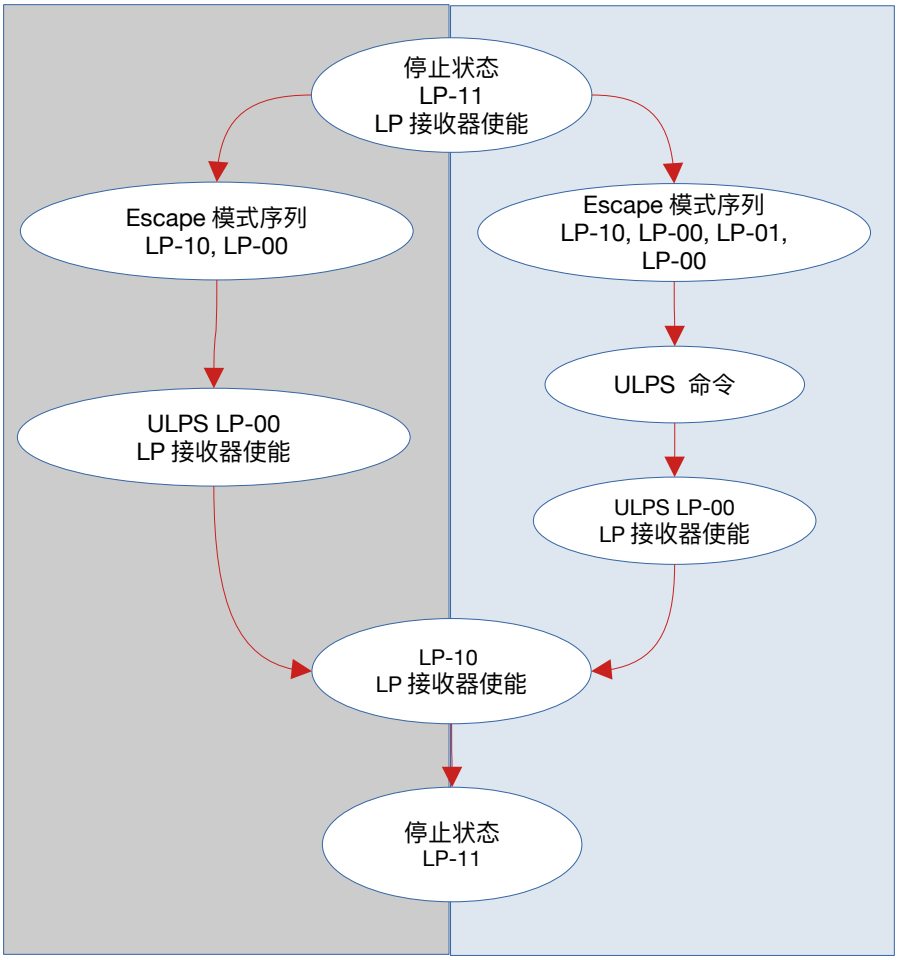


图 42.5-4. 时钟和数据通道 ULPS 序列

42.5.3 MIPI RX D-PHY 配置

MIPI RX D-PHY 包含一组 Test Code，用于配置 MIPI RX D-PHY 操作。所有支持的 Test Code 请参阅小节 42.5.3.2。

42.5.3.1 MIPI RX D-PHY 编码接口

MIPI RX D-PHY 的 Test Code 编程包括两个步骤。首先，将所需的 Test Code 编程到寄存器中，然后将相应的 Test Data 提供给测试仪的输入端。

在配置 Test Code 之前，执行以下步骤：

- 将 CSI_HOST_PHY_SHUTDOWNZ 和 CSI_HOST_DPHY_RSTZ 置 0 以关闭和复位 PHY。
- （推荐）将 CSI_HOST_PHY_TESTCLR 置 1，以在首次配置任何 Test Code 之前应用重置脉冲。

然后，按照以下步骤配置 Test Code：

- 设置所需的 Test Code。

  当 CSI_HOST_PHY_TESTEN 为 1 且 CSI_HOST_PHY_TESTCLK 处于下降沿时，Test Code 被采样锁存。

  - 确保 CSI_HOST_PHY_TESTCLK 为 1。
  - 将 CSI_HOST_PHY_TESTDIN 设置为 8 位 Test Code。
  - 将 CSI_HOST_PHY_TESTEN 设置为 1。
  - 将 CSI_HOST_PHY_TESTCLK 设置为 0。在 CSI_HOST_PHY_TESTCLK 处于下降沿时，CSI_HOST_PHY_TESTDIN[7:0] 的值会被内部锁存为当前的 Test Code。
  - 将 CSI_HOST_PHY_TESTEN 设置为 0。

- 输入 Test Data。

  当 CSI_HOST_PHY_TESTEN 为 0 且 CSI_HOST_PHY_TESTCLK 处于上升沿时，Test Data 将被编程到内部。

  - 设置 CSI_HOST_PHY_TESTCLK 为 0（如果尚未完成）。
  - 设置 CSI_HOST_PHY_TESTDIN 为 8 位 Test Data。
  - 设置 CSI_HOST_PHY_TESTCLK 为 1。当 CSI_HOST_PHY_TESTCLK 为上升沿时，Test Data 将被编程到内部。
  - 重复上述步骤，添加更多 Test Data。

将 CSI_HOST_PHY_TESTCLR 设置为 1 可复位 Test Code。仅在第一次编程操作之前，或希望将 PHY 的配置重置为默认值时需要执行此步骤。

图 42.5-5 展示了 MIPI RX D-PHY 控制接口的时序图。Test Code 成功编程后，CSI_HOST_PHY_TESTDOUT 将异步输出与特定 Test Code 相关的数据。这些数据可能包括回读信息或其他有意义的信号。

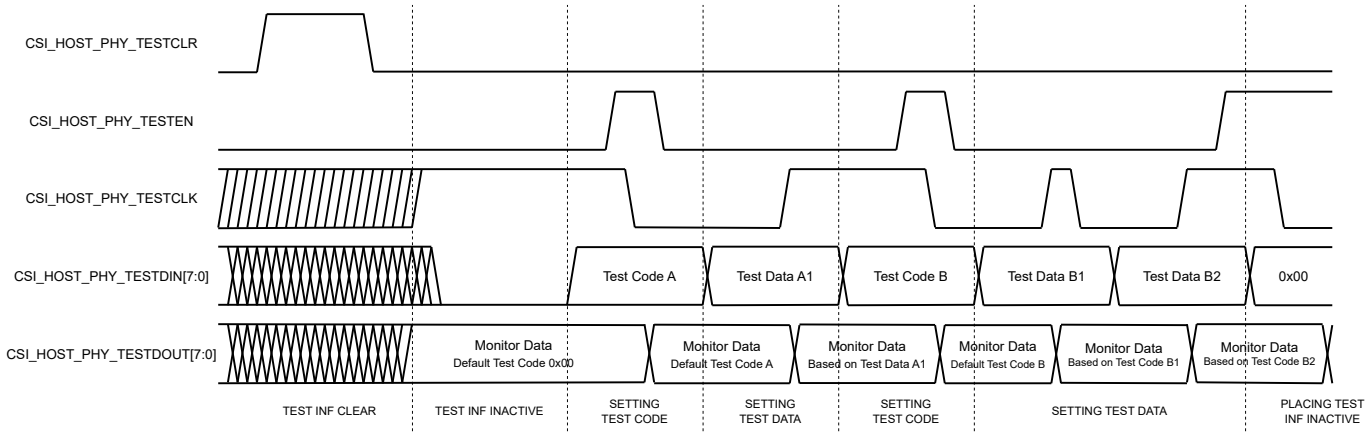


图 42.5-5. 控制接口时序图

说明：

某些 Test Code 需要两次写数据操作。在第一次操作中，CSI_HOST_PHY_TESTCLK 的值从 1 变为 0。请注意，当 CSI_HOST_PHY_TESTEN 设置为 1 时，应避免时钟的下降沿，否则当前的 Test Data 将被锁存为错误的 Test Code。

强烈建议在空闲时通过编程 Test Code 0x00 将测试接口设置为非激活状态，如图 42.5-5 中的最终序列所示。

42.5.3.2 MIPI RX D-PHY Test Code

可通过 CSI_HOST_PHY_TEST_CTRL0_REG 和 CSI_HOST_PHY_TEST_CTRL1_REG 配置 MIPI RX D-PHY 以执行 Test Code。详细信息请参阅小节 42.5.3.1 *MIPI RX D-PHY 编码接口*。

常规操作流程

使用以下 Test Code 启用 PHY 的常规操作流程和测试接口的唤醒状态。在此模式下，测试接口处于非激活状态。

- **Test Code**：0x00
- **Test Data**：无

通道 0 HS RX 控制

使用以下 Test Code 选择 HS 操作频率范围 (hsfreqrange)。

- **Test Code**：0x44
- **Test Data**：

表 42.5-3. 通道 0 HS RX 控制下的测试数据 (Test Data)

w-1'b0	w-6'b0	w-1'b0
程序选择器	HS 工作频率范围选择 (hsfreqrange)	保留

当程序选择器为 0 时，Test Data[6:1] 编程 hsfreqrange。

- **Test Dout**：

表 42.5-4. 通道 0 HS RX 控制下的测试数据输出 (Test Dout)

r-1'b0	r-6'b0	r-1'b0
1'b0	HS 工作频率范围选择 (hsfreqrange)	1'b0

当 CSI_HOST_PHY_TESTDIN[7] 为 0 时，Test Dout[6:1] 为编程的 hsfreqrange 的值。

其他

其他 Test Code 保留。

42.5.4 解扰器

数据加扰使用随机化技术来减轻电磁干扰 (EMI) 和射频 (RF) 自干扰，可以将链路的传输能量扩展到更宽的频率范围。

传输数据通过伪随机比特流 (PRBS) 加扰，以减少通道级别上出现重复模式的可能性。PRBS 由线性反馈移位寄存器 (LFSR) 生成，生成多项式为：

$$G(x) = x^{16} + x^5 + x^4 + x^3 + 1 \tag{42.1}$$

每条数据通道都有一个解扰模块。将 CSI_HOST_SCRAMBLE_ENABLE 设置为 1 可启用此模块。每条通道可以通过 CSI_HOST_SCRAMBLE_SEED_LANE*n* 配置不同的 16 位 LFSR 种子值。

表 42.5-5 列出了控制器复位后通道的默认种子值。

表 42.5-5. 通道 1 和通道 2 加扰器 PRBS 初始种子值

通道	初始种子值
1	0x1008
2	0x1188

42.5.5 错误管理

MIPI CSI-2 主机分析接收到的数据包，并确定是否存在协议错误。可能出现的错误如表 42.5-6 所示。这些错误作为中断源可以生成 CSI_INTR 信号。详细信息请参阅小节 42.6 中断。

表 42.5-6. MIPI CSI-2 主机可检测到的错误

错误/中断源	描述	对应寄存器字段
PHY_ERRSOTSYNCHS_*n* (*n*: 0-1)	数据通道 *n* 传输启动错误(未同步)	CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n* (*n*: 0-1)
ERR_ECC_DOUBLE	标头 ECC 包含至少两个错误（无法恢复）	CSI_HOST_ST_ERR_ECC_DOUBLE
SHORTER_PAYLOAD	报告的字数大于接收的字数（无法恢复）	CSI_HOST_ST_SHORTER_PAYLOAD
ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15)	虚拟通道 *n* 的帧开始与帧结束不匹配	CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n* (*n*: 0-15)

ERR_F_SEQ_VC*n* (*n*：0-15)	虚拟通道 *n* 上检测到不正确帧序列	CSI_HOST_ST_ERR_F_SEQ_VC*n* (*n*：0-15)
ERR_FRAME_DATA_VC*n* (*n*：0-15)	虚拟通道 *n* 的最后一个接收帧中至少有一个 CRC 错误	CSI_HOST_ST_ERR_FRAME_DATA_VC*n* (*n*：0-15)
ERR_CRC_VC*n* (*n*：0-15)	虚拟通道 *n* 上检测到有效载荷 CRC 错误	CSI_HOST_ST_ERR_CRC_VC*n* (*n*：0-15)
ERR_ID_VC*n* (*n*：0-15)	虚拟通道 *n* 上检测到未识别或未实现的数据类型	CSI_HOST_ST_ERR_ID_VC*n* (*n*：0-15)
ERR_ECC_CORRECTED_VC*n* (*n*：0-15)	虚拟通道 *n* 上检测到校验和错误	CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n* (*n*：0-15)
PHY_ERRSOTHS_*n* (*n*：0-1)	数据通道 *n* 传输启动错误（仍可实现同步）	CSI_HOST_ST_PHY_ERRSOTHS_*n* (*n*：0-1)
PHY_ERRESC_*n* (*n*：0-1)	数据通道 *n* Escape 进入错误 (ULPS)	CSI_HOST_ST_PHY_ERRESC_*n* (*n*：0-1)

说明：

MIPI CSI 不支持虚拟通道交错。在开始传输之前，请确认正在使用的虚拟通道。请从指定的虚拟通道读取错误信息以检测错误。

对于所有 "*_FATAL_REG" 寄存器中的错误，建议执行复位操作。

42.6 中断

ESP32-P4 的 MIPI CSI 模块可以生成以下中断信号并发送给中断矩阵。

CSI_HOST_INT_ST_<group> 寄存器组会报告错误条件，并触发 CSI_INTR 中断信号。

通过 CSI_HOST_INT_MSK_<group> 寄存器组可以屏蔽中断管脚的触发，默认情况下，所有错误都被屏蔽。将这些寄存器的任意位设置为 1 可启用特定的错误中断。发生错误时，CSI_HOST_INT_ST_<group> 寄存器组相应的位会被置为 1。CSI_HOST_INT_ST_<group> 寄存器组在每次读操作后会被清除。

激活相应错误条件后，设置 CSI_HOST_INT_FORCE_<group> 寄存器组的任何位为 1 将触发相应的中断。这些寄存器仅用于测试。

图 42.6-1 展示了 MIPI CSI 的中断原理：

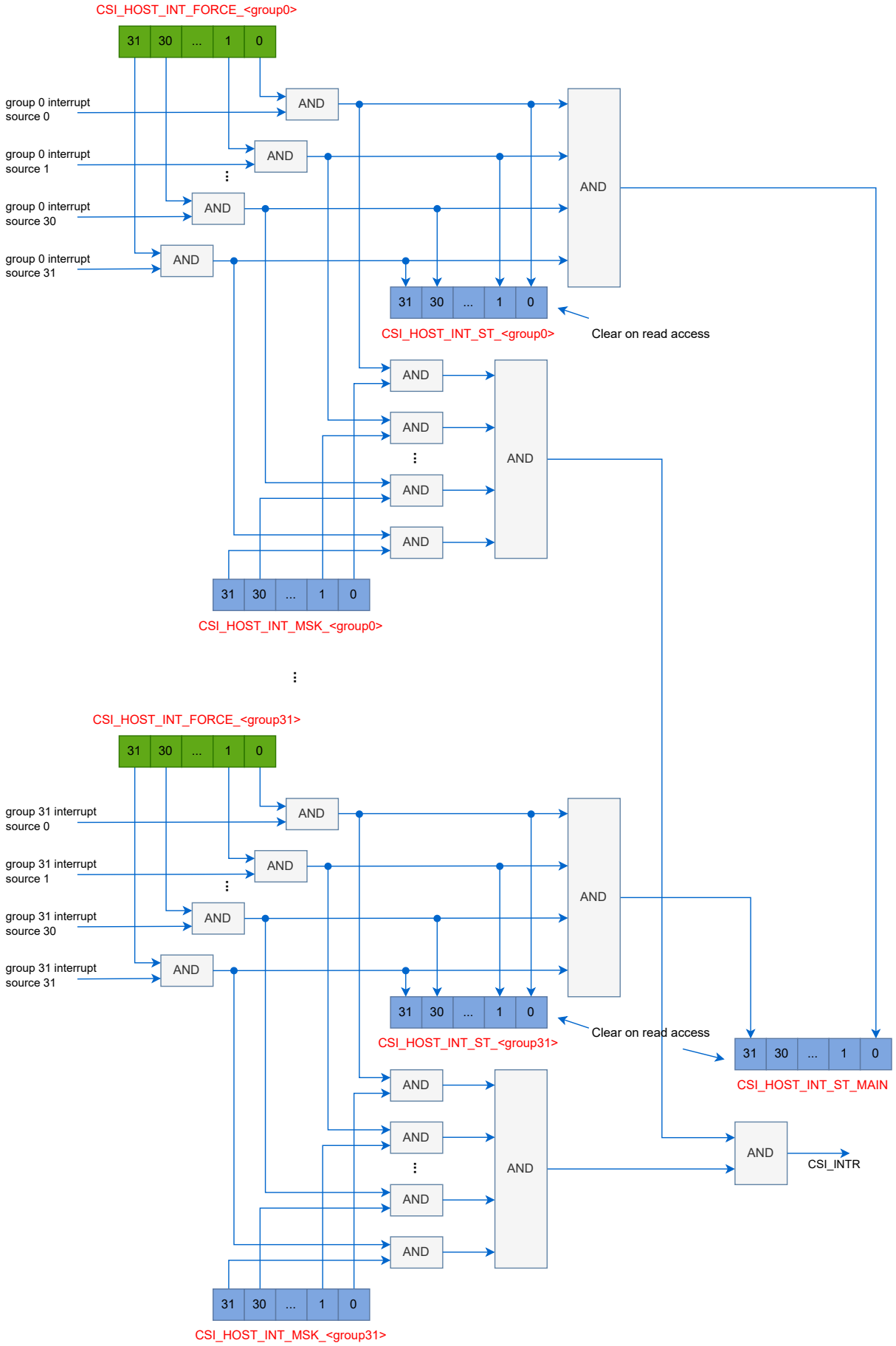


图 42.6-1. MIPI CSI 中断原理

CSI_INTR 中断信号由 MIPI CSI 的以下内部中断源生成（详细中断源描述请参阅表 42.5-6 *MIPI CSI-2 主机可检测到的错误*）：

- PHY 致命中断：
  - PHY_ERRSOTSYNCHS_*n* (*n*：0-1)
- 数据包致命中断：
  - ERR_ECC_DOUBLE
  - SHORTER_PAYLOAD
- 边界帧致命中断：
  - ERR_F_BNDRY_MATCH_VC*n* (*n*：0-15)
- 序列帧致命中断：
  - ERR_F_SEQ_VC*n* (*n*：0-15)
- CRC 帧致命中断：
  - ERR_FRAME_DATA_VC*n* (*n*：0-15)
- 有效载荷 CRC 致命中断：
  - ERR_CRC_VC*n* (*n*：0-15)
- 数据 ID 中断：
  - ERR_ID_VC*n* (*n*：0-15)
- ECC 纠正中断：
  - ERR_ECC_CORRECTED_VC*n* (*n*：0-15)
- PHY 中断：
  - PHY_ERRSOTHS_*n* (*n*：0-1)
  - PHY_ERRESC_*n* (*n*：0-1)

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

42.7 配置流程

本小节介绍如何配置 MIPI RX D-PHY 和 MIPI CSI-2 主机以接收图像数据。MIPI CSI 仅在高速数据接收模式下接收图像数据。

42.7.1 开始高速数据接收

42.7.1.1 时钟与复位

- 设置 HP_SYS_CLKRST_CSI_HOST_SYS_CLK_EN 为 1，启用 MIPI CSI 时钟。
- 设置 HP_SYS_CLKRST_CSI_HOST_SYS_RST_EN 为 0，释放 MIPI CSI 复位。

- 配置 HP_SYS_CLKRST_MIPI_CSI_DPHY_CLK_SRC_SEL，为 MIPI RX D-PHY 选择合适的时钟源。
- 设置 HP_SYS_CLKRST_MIPI_CSI_DPHY_CFG_CLK_EN 为 1，启用 MIPI RX D-PHY 的配置时钟。
- 设置 HP_SYS_CLKRST_RST_EN_CSI_BRG 为 1，释放 ISP 的 CSI_Bridge 模块并释放 ISP 复位。

42.7.1.2 MIPI RX D-PHY 和 MIPI 主机初始化

- 清除 CSI_HOST_PHY_TESTCLR，复位 CSI_HOST 主机控制器逻辑。
- 设置 CSI_HOST_PHY_TESTCLR 为 1，然后清除 CSI_HOST_PHY_TESTCLR 以复位 Test Code 逻辑。
- 在 MIPI RX D-PHY 编程接口中用 Test Code 0x44 配置 MIPI RX D-PHY 频率范围。
- 配置 CSI_HOST_N_LANES，选择系统中的活跃通道数。
- 设置 CSI_HOST_PHY_SHUTDOWNZ 和 CSI_HOST_DPHY_RSTZ 为 1，释放 MIPI RX D-PHY 复位。
- 将 CSI_HOST_INT_MSK_*_REG 寄存器的对应位设置为 1，启用错误中断。
- （如需）配置解扰功能。
- 设置 CSI_HOST_CSI2_RESETN 为 1，释放 CSI_HOST 复位。
- 等待 CSI_HOST_PHY_STOPSTATEDATA_*n* 和 CSI_HOST_PHY_STOPSTATECLK 变为 1。
- 启动摄像头图像捕获。

42.7.2 停止高速数据接收

- 禁用 VDMA 中的相关通道并等待 VDMA 被禁用。请参阅章节 *7 VDMA 控制器 (VDMA)*。
- 复位 ISP。请参阅章节 *38 图像信号处理器 (ISP)*。
- 设置 HP_SYS_CLKRST_RST_EN_CSI_BRG 为 0，复位 MIPI CSI 模块。

42.8 寄存器列表

本小节的所有地址均为相对于 CSI 主机基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
版本寄存器			
CSI_HOST_VERSION_REG	CSI_HOST 版本控制寄存器	0x0000	RO
配置寄存器			
CSI_HOST_N_LANES_REG	活跃通道配置寄存器	0x0004	R/W
CSI_HOST_CSI2_RESETN_REG	CSI_HOST 复位控制寄存器	0x0008	R/W
CSI_HOST_PHY_SHUTDOWNZ_REG	RX D-PHY 关断状态控制寄存器	0x0040	R/W
CSI_HOST_DPHY_RSTZ_REG	RX D-PHY 复位控制寄存器	0x0044	R/W
CSI_HOST_PHY_RX_REG	RX D-PHY RX 相关信号状态寄存器	0x0048	RO
CSI_HOST_PHY_TEST_CTRL0_REG	RX D-PHY Test Code 接口控制寄存器 0	0x0050	R/W
CSI_HOST_PHY_TEST_CTRL1_REG	RX D-PHY Test Code 接口控制寄存器 1	0x0054	varies
CSI_HOST_SCRAMBLING_REG	解扰器使能寄存器	0x0300	R/W
CSI_HOST_SCRAMBLING_SEED1_REG	数据通道 0 解扰器种子值控制寄存器	0x0304	R/W
CSI_HOST_SCRAMBLING_SEED2_REG	数据通道 1 解扰器种子值控制寄存器	0x0308	R/W
中断寄存器			
CSI_HOST_INT_ST_MAIN_REG	主要中断状态寄存器	0x000C	RC
CSI_HOST_INT_ST_PHY_FATAL_REG	RX D-PHY 致命中断状态寄存器	0x00E0	RC
CSI_HOST_INT_MSK_PHY_FATAL_REG	RX D-PHY 致命中断掩码寄存器	0x00E4	R/W
CSI_HOST_INT_FORCE_PHY_FATAL_REG	RX D-PHY 致命中断强制寄存器	0x00E8	R/W
CSI_HOST_INT_ST_PKT_FATAL_REG	数据包致命中断状态寄存器	0x00F0	RC
CSI_HOST_INT_MSK_PKT_FATAL_REG	数据包致命中断掩码寄存器	0x00F4	R/W
CSI_HOST_INT_FORCE_PKT_FATAL_REG	数据包致命中断强制寄存器	0x00F8	R/W
CSI_HOST_INT_ST_PHY_REG	RX D-PHY 中断状态寄存器	0x0110	RC
CSI_HOST_INT_MSK_PHY_REG	RX D-PHY 中断掩码寄存器	0x0114	R/W
CSI_HOST_INT_FORCE_PHY_REG	RX D-PHY 中断强制寄存器	0x0118	R/W
CSI_HOST_INT_ST_BNDRY_FRAME_FATAL_REG	帧边界致命中断状态寄存器	0x0280	RC
CSI_HOST_INT_MSK_BNDRY_FRAME_FATAL_REG	帧边界致命中断掩码寄存器	0x0284	R/W
CSI_HOST_INT_FORCE_BNDRY_FRAME_FATAL_REG	帧边界致命中断强制寄存器	0x0288	R/W
CSI_HOST_INT_ST_SEQ_FRAME_FATAL_REG	帧序列致命中断状态寄存器	0x0290	RC
CSI_HOST_INT_MSK_SEQ_FRAME_FATAL_REG	帧序列致命中断掩码寄存器	0x0294	R/W
CSI_HOST_INT_FORCE_SEQ_FRAME_FATAL_REG	帧序列致命中断强制寄存器	0x0298	R/W
CSI_HOST_INT_ST_CRC_FRAME_FATAL_REG	帧 CRC 致命中断状态寄存器	0x02A0	RC
CSI_HOST_INT_MSK_CRC_FRAME_FATAL_REG	帧 CRC 致命中断掩码寄存器	0x02A4	R/W
CSI_HOST_INT_FORCE_CRC_FRAME_FATAL_REG	帧 CRC 致命中断强制寄存器	0x02A8	R/W
CSI_HOST_INT_ST_PLD_CRC_FATAL_REG	有效载荷 CRC 致命中断状态寄存器	0x02B0	RC
CSI_HOST_INT_MSK_PLD_CRC_FATAL_REG	有效载荷 CRC 致命中断掩码寄存器	0x02B4	R/W
CSI_HOST_INT_FORCE_PLD_CRC_FATAL_REG	有效载荷 CRC 致命中断强制寄存器	0x02B8	R/W
CSI_HOST_INT_ST_DATA_ID_REG	数据 ID 状态寄存器	0x02C0	RC

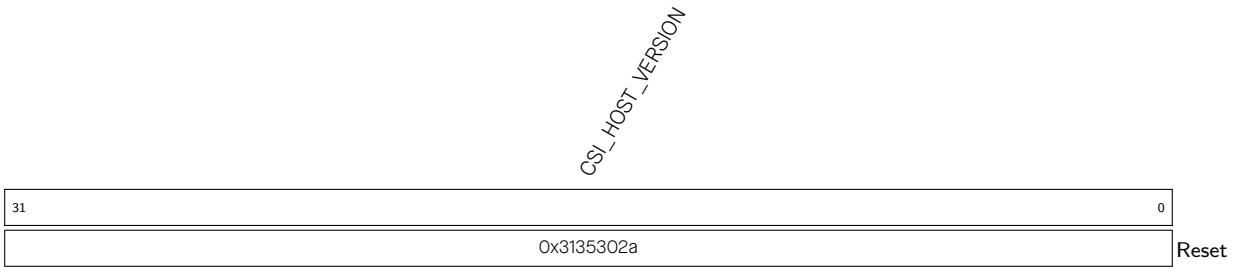
名称	描述	地址	访问
CSI_HOST_INT_MSK_DATA_ID_REG	数据 ID 中断掩码寄存器	0x02C4	R/W
CSI_HOST_INT_FORCE_DATA_ID_REG	数据 ID 中断强制寄存器	0x02C8	R/W
CSI_HOST_INT_ST_ECC_CORRECTED_REG	标头错误检测与纠正中断状态寄存器	0x02D0	RC
CSI_HOST_INT_MSK_ECC_CORRECTED_REG	标头错误检测与纠正中断掩码寄存器	0x02D4	R/W
CSI_HOST_INT_FORCE_ECC_CORRECTED_REG	标头错误检测与纠正中断强制寄存器	0x02D8	R/W
状态寄存器			
CSI_HOST_PHY_STOPSTATE_REG	RX D-PHY 停止状态信号状态寄存器	0x004C	RO

42.9 寄存器

本小节的所有地址均为相对于 CSI 主机基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

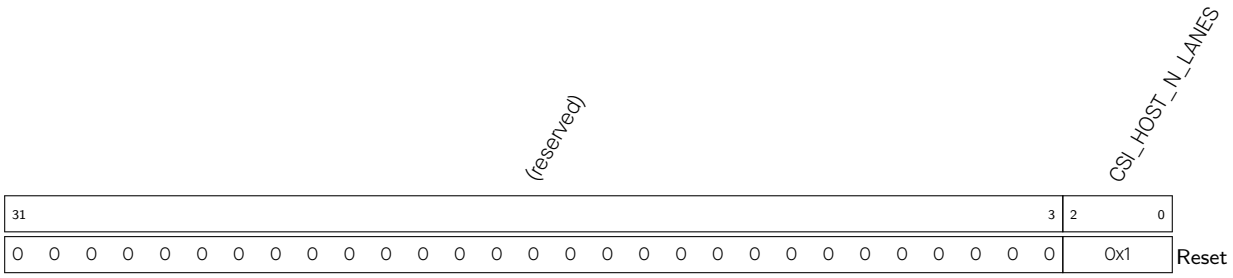
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 42.1. CSI_HOST_VERSION_REG (0x0000)



CSI_HOST_VERSION 版本控制寄存器。(RO)

Register 42.2. CSI_HOST_N_LANES_REG (0x0004)



CSI_HOST_N_LANES 配置 MIPI CSI-2 主机用于接收摄像头设备数据的活跃通道数。只有当 RX D-PHY 通道处于停止状态时，才能更新此字段。

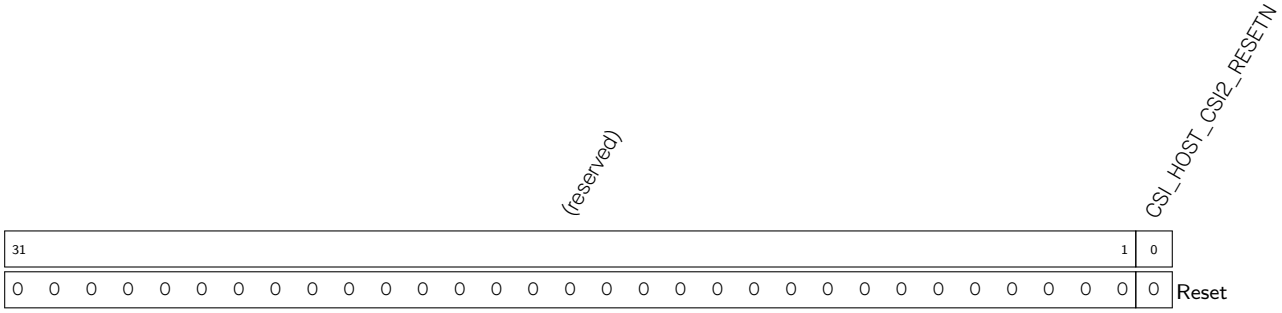
0：数据通道 0 活跃

1：数据通道 0 和数据通道 1 都活跃

其他值：无效

(R/W)

Register 42.3. CSI_HOST_CSI2_RESETN_REG (0x0008)



CSI_HOST_CSI2_RESETN 配置是否复位 MIPI CSI-2 主机控制器的内部逻辑。

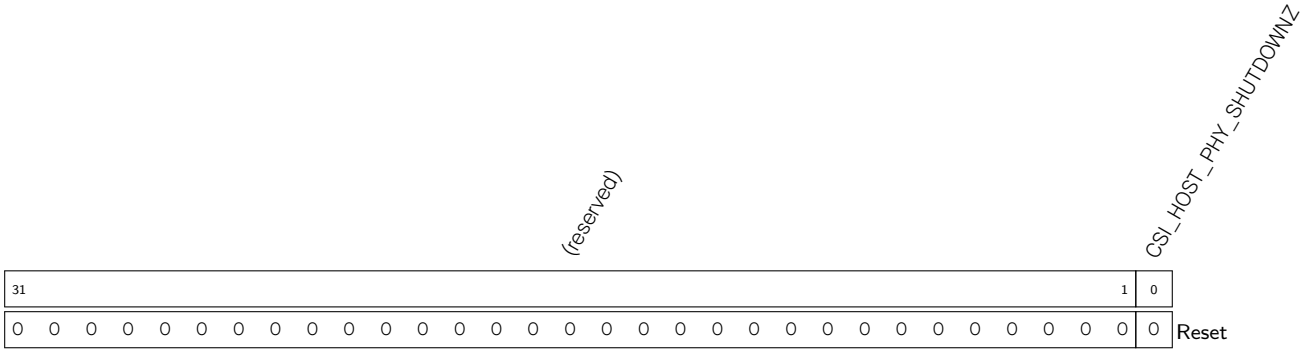
0：复位

1：释放复位

此寄存器仅影响控制器的内部逻辑，不会将配置复位为默认值。

(R/W)

Register 42.4. CSI_HOST_PHY_SHUTDOWNZ_REG (0x0040)



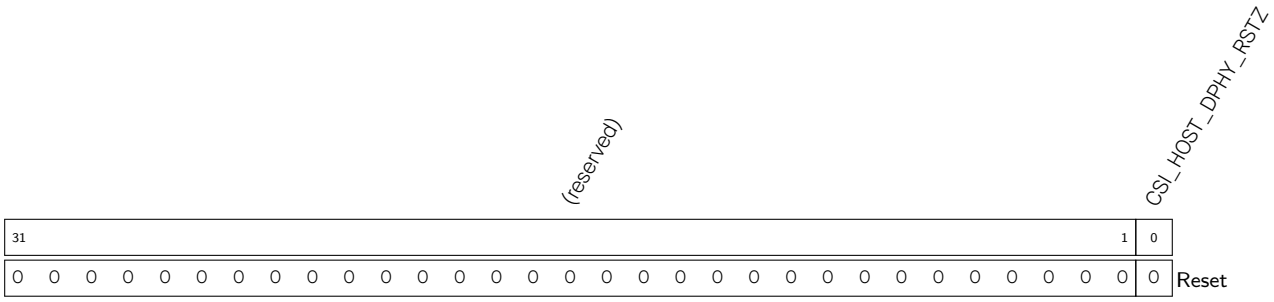
CSI_HOST_PHY_SHUTDOWNZ 配置是否启用 RX D-PHY 的关断状态。

0：启用

1：禁用

(R/W)

Register 42.5. CSI_HOST_DPHY_RSTZ_REG (0x0044)



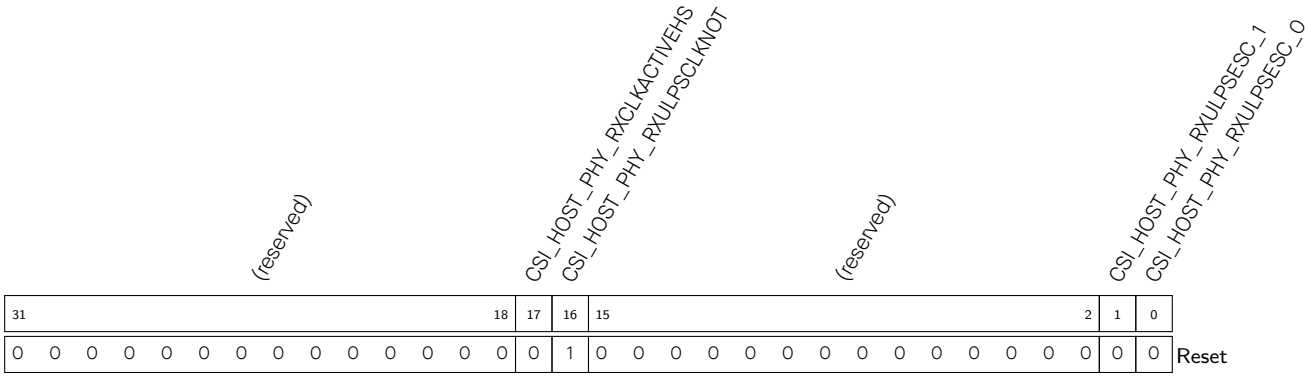
CSI_HOST_DPHY_RSTZ 配置是否复位 RX D-PHY 数字电路。

0：复位

1：释放复位

(R/W)

Register 42.6. CSI_HOST_PHY_RX_REG (0x0048)



CSI_HOST_PHY_RXULPSESC_*n* (*n*：0-1) 表示数据通道 *n* 是否进入超低功耗状态。

0：进入

1：未进入

(RO)

CSI_HOST_PHY_RXULPSCLKNOT 表示 RX D-PHY 时钟通道是否进入超低功耗状态。

0：进入

1：未进入

(RO)

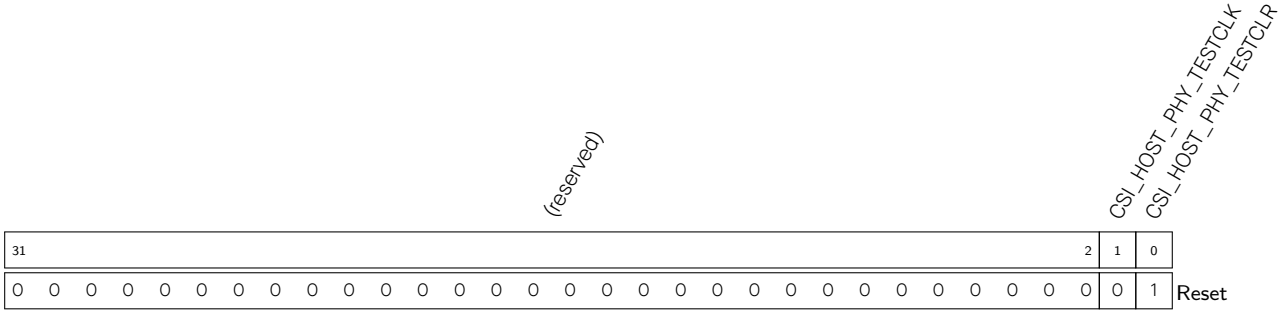
CSI_HOST_PHY_RXCLKACTIVEHS 表示 RX D-PHY 时钟通道是否正在主动接收 DDR 时钟。

0：未主动接收

1：正在主动接收

(RO)

Register 42.7. CSI_HOST_PHY_TEST_CTRL0_REG (0x0050)



CSI_HOST_PHY_TESTCLR 配置是否初始化 Test Code 接口。

0：不初始化

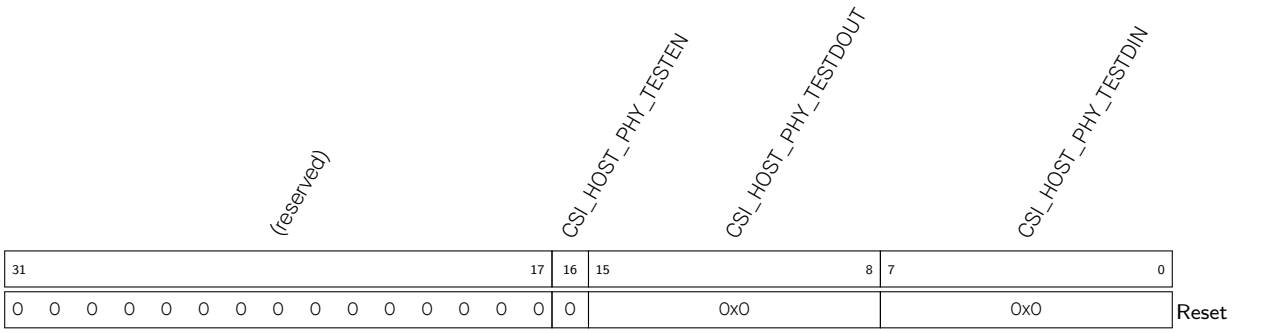
1：初始化

为确保模拟可编程默认值已预设，此位必须在上电后设置为 1，然后清除为 0。

(R/W)

CSI_HOST_PHY_TESTCLK 配置时钟以捕获 CSI_HOST_PHY_TESTDIN 中的总线内容到 D-PHY Test Code 逻辑中，通过 CSI_HOST_PHY_TESTEN 控制操作选择。(R/W)

Register 42.8. CSI_HOST_PHY_TEST_CTRL1_REG (0x0054)



CSI_HOST_PHY_TESTDIN 配置 Test Code 接口的 8 位数据输入，用于编程内部寄存器和访问测试功能。(R/W)

CSI_HOST_PHY_TESTDOUT 代表 Test Code 接口的 8 位数据输出，用于读取数据和其他检测功能。(RO)

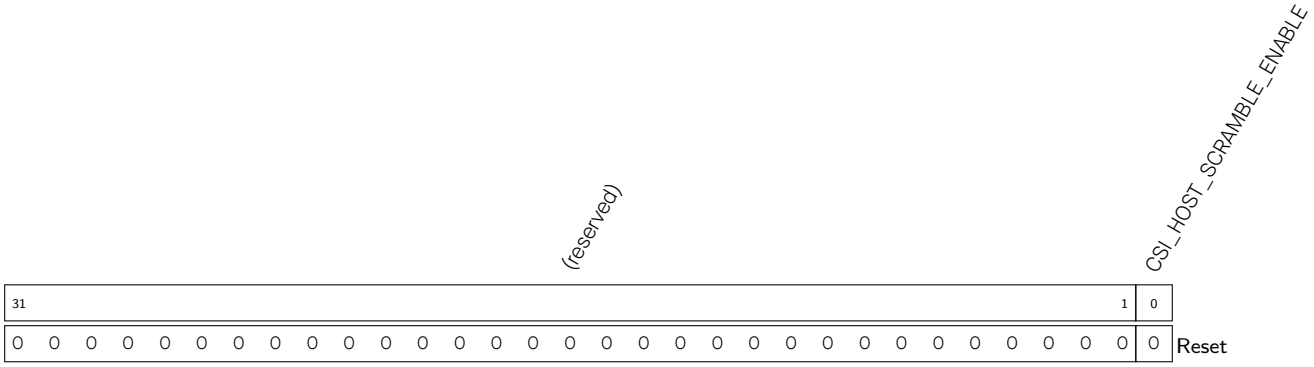
CSI_HOST_PHY_TESTEN 配置在 CSI_HOST_PHY_TESTCLK 的上升沿还是下降沿执行写入操作。

0：上升沿

1：下降沿

(R/W)

Register 42.9. CSI_HOST_SCRAMBLING_REG (0x0300)



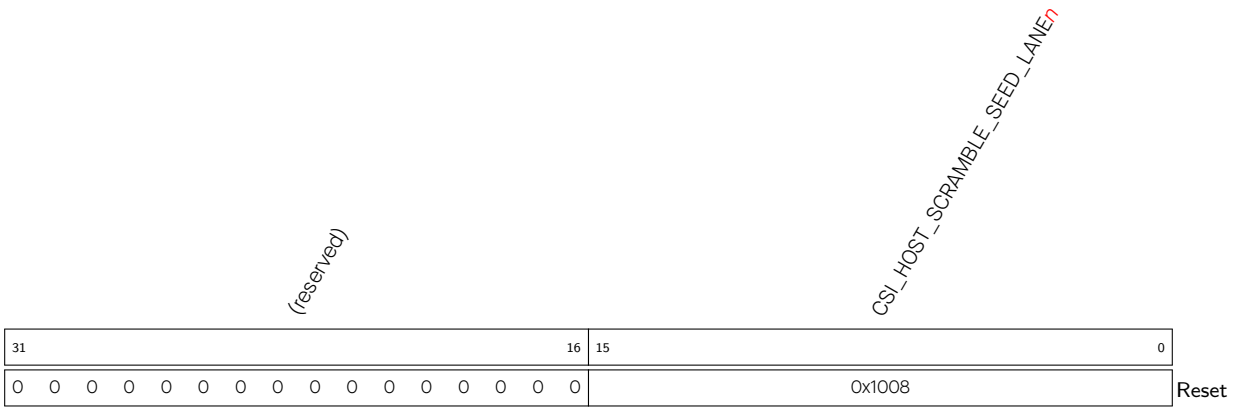
CSI_HOST_SCRAMBLE_ENABLE 配置是否启用解扰器。

0：禁用

1：启用

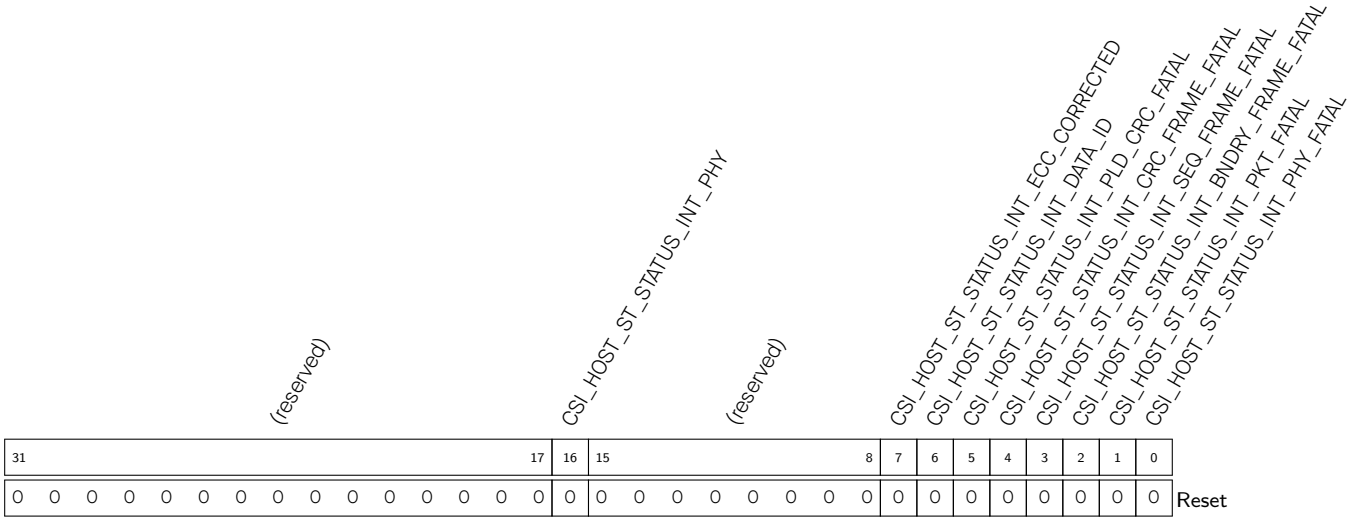
(R/W)

Register 42.10. CSI_HOST_SCRAMBLING_SEED*n*_REG (*n*: 1-2) (0x0300 + 0x04**n*)



CSI_HOST_SCRAMBLE_SEED_LANE*n-1* 配置解扰器用于通道 *n-1* 的种子值。(R/W)

Register 42.11. CSI_HOST_INT_ST_MAIN_REG (0x000C)



CSI_HOST_ST_STATUS_INT_PHY_FATAL 代表 CSI_HOST_INT_ST_PHY_FATAL_REG 的状态。(RC)

CSI_HOST_ST_STATUS_INT_PKT_FATAL 代表 CSI_HOST_INT_ST_PKT_FATAL_REG 的状态。(RC)

CSI_HOST_ST_STATUS_INT_BNDRY_FRAME_FATAL 代表 CSI_HOST_INT_ST_BNDRY_FRAME_FATAL_REG 的状态。(RC)

CSI_HOST_ST_STATUS_INT_SEQ_FRAME_FATAL 代表 CSI_HOST_INT_ST_SEQ_FRAME_FATAL_REG 的状态。(RC)

CSI_HOST_ST_STATUS_INT_CRC_FRAME_FATAL 代表 CSI_HOST_INT_ST_CRC_FRAME_FATAL_REG 的状态。(RC)

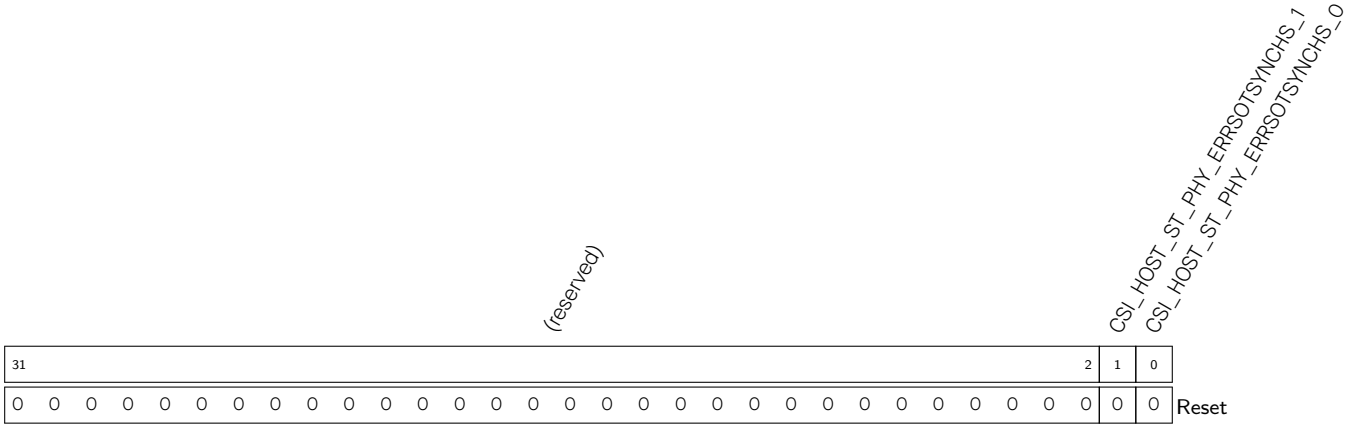
CSI_HOST_ST_STATUS_INT_PLD_CRC_FATAL 代表 CSI_HOST_INT_ST_PLD_CRC_FATAL_REG 的状态。(RC)

CSI_HOST_ST_STATUS_INT_DATA_ID 代表 CSI_HOST_INT_ST_DATA_ID_REG 的状态。(RC)

CSI_HOST_ST_STATUS_INT_ECC_CORRECTED 代表 CSI_HOST_INT_ST_ECC_CORRECTED_REG 的状态。(RC)

CSI_HOST_ST_STATUS_INT_PHY 代表 CSI_HOST_INT_PHY_REG 的状态。(RC)

Register 42.12. CSI_HOST_INT_ST_PHY_FATAL_REG (0x00E0)



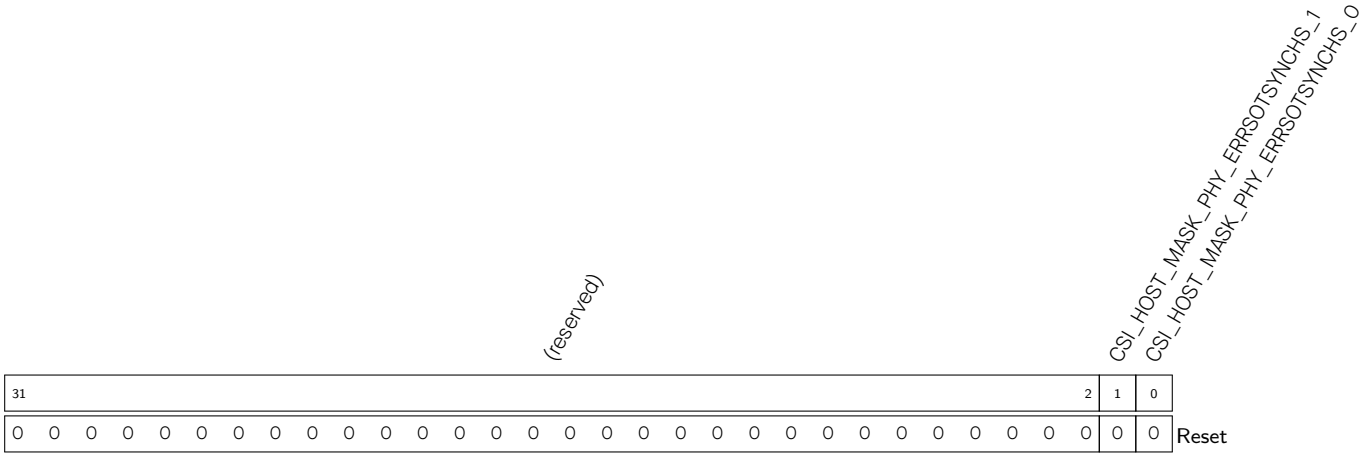
CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n* (*n*：0-1) 代表 PHY_ERRSOTSYNCHS_*n* 错误是否发生。

0：未发生

1：发生

(RC)

Register 42.13. CSI_HOST_INT_MSK_PHY_FATAL_REG (0x00E4)



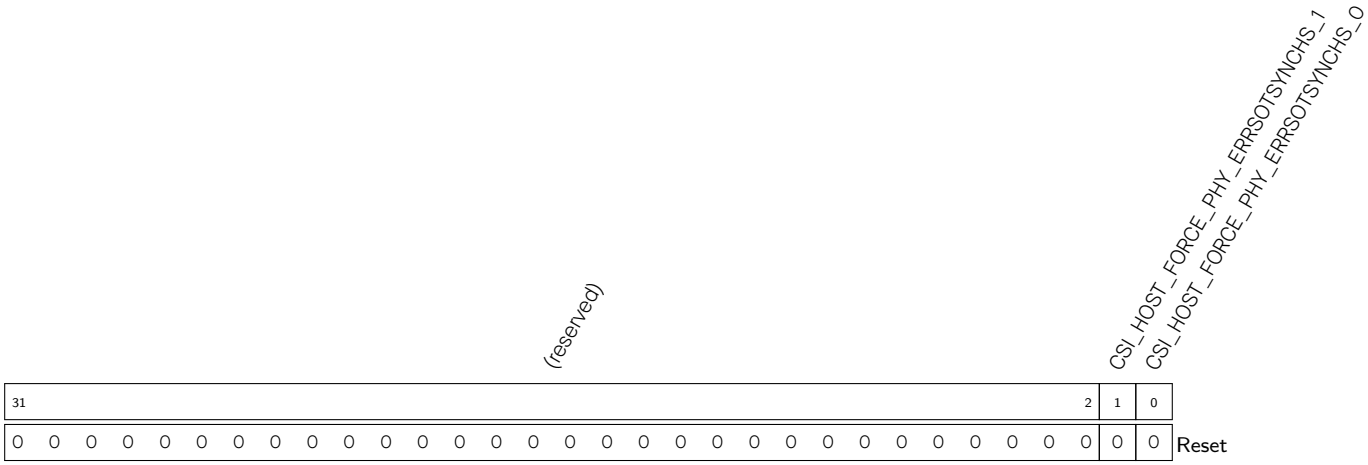
CSI_HOST_MASK_PHY_ERRSOTSYNCHS_*n* (*n*：0-1) 配置是否屏蔽 CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n*。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.14. CSI_HOST_INT_FORCE_PHY_FATAL_REG (0x00E8)



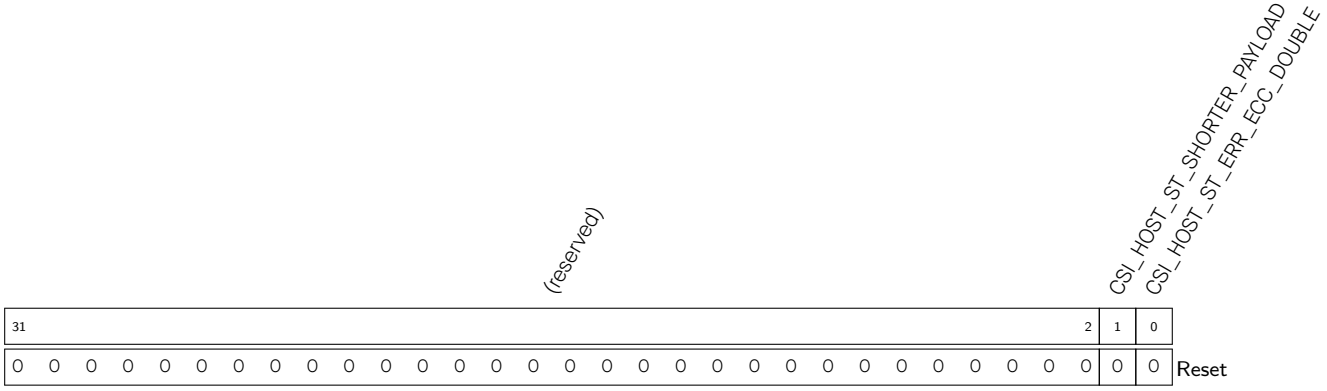
CSI_HOST_FORCE_PHY_ERRSOTSYNCHS_*n* (*n*：0-1) 配置是否强制将 CSI_HOST_ST_PHY_ERRSOTSYNCHS_*n* 设为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.15. CSI_HOST_INT_ST_PKT_FATAL_REG (0x00F0)



CSI_HOST_ST_ERR_ECC_DOUBLE 表示是否发生 ERR_ECC_DOUBLE 错误。

0：未发生

1：发生

(RO)

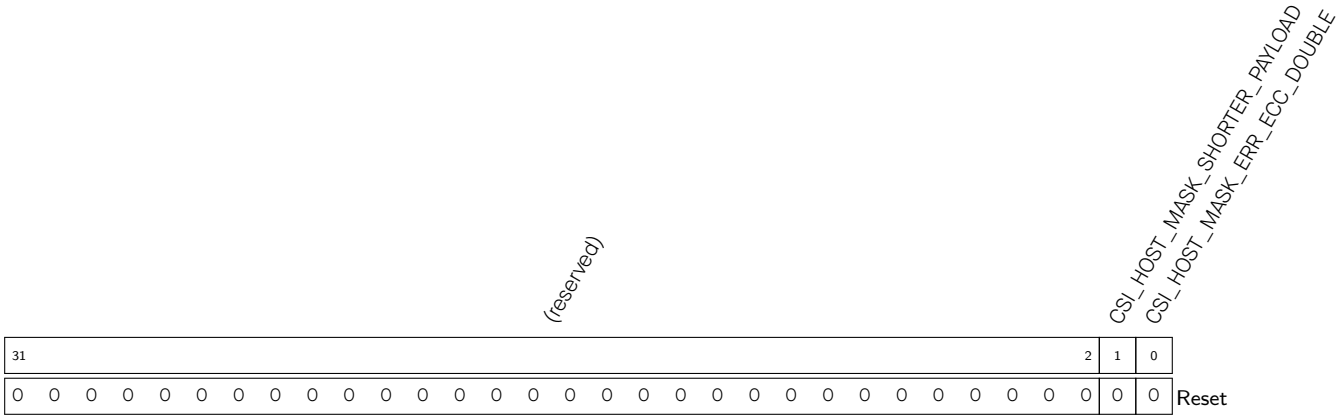
CSI_HOST_ST_SHORTER_PAYLOAD 表示是否发生 SHORTER_PAYLOAD 错误。

0：未发生

1：发生

(RO)

Register 42.16. CSI_HOST_INT_MSK_PKT_FATAL_REG (0x00F4)



CSI_HOST_MASK_ERR_ECC_DOUBLE 配置是否屏蔽 CSI_HOST_ST_ERR_ECC_DOUBLE。

0：屏蔽错误中断

1：启用错误中断

(R/W)

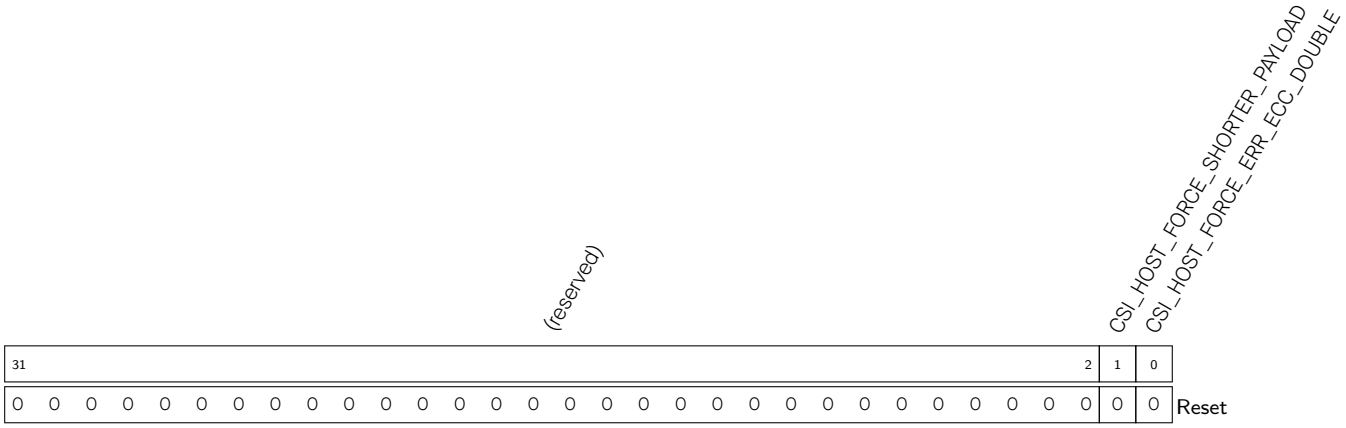
CSI_HOST_MASK_SHORTER_PAYLOAD 配置是否屏蔽 CSI_HOST_ST_SHORTER_PAYLOAD。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.17. CSI_HOST_INT_FORCE_PKT_FATAL_REG (0x00F8)



CSI_HOST_FORCE_ERR_ECC_DOUBLE 配置是否强制将 CSI_HOST_ST_ERR_ECC_DOUBLE 设置为 1。

0：不强制设置

1：强制设置

(R/W)

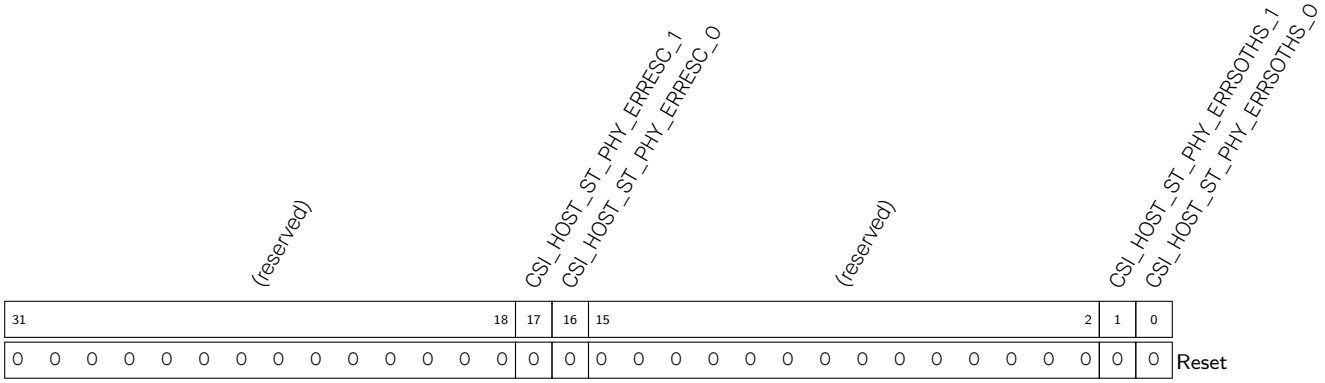
CSI_HOST_FORCE_SHORTER_PAYLOAD 配置是否强制将 CSI_HOST_ST_SHORTER_PAYLOAD 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.18. CSI_HOST_INT_ST_PHY_REG (0x0110)



CSI_HOST_ST_PHY_ERRSOTHS_*n* (*n*：0-1) 表示 PHY_ERRSOTHS_*n* 错误是否发生。

0：未发生

1：发生

(RC)

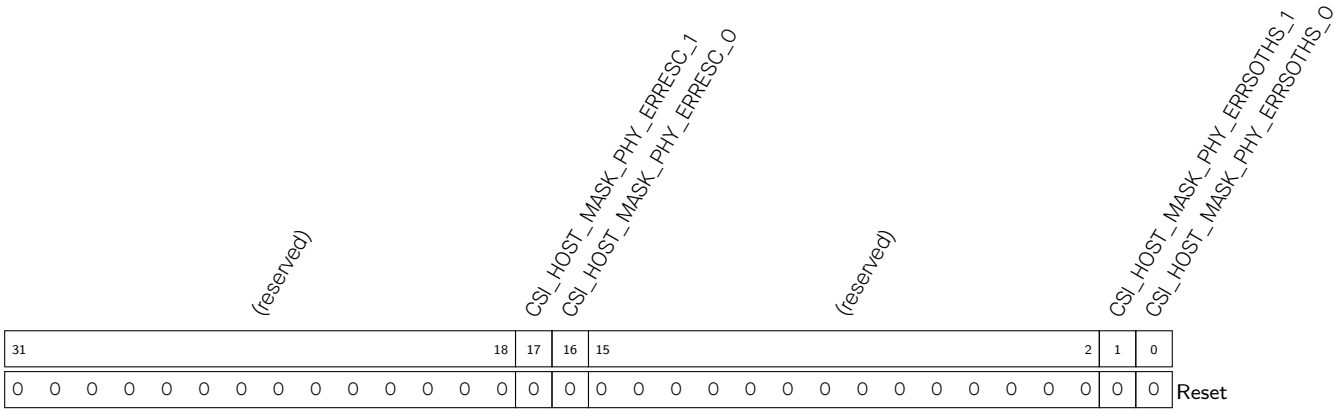
CSI_HOST_ST_PHY_ERRESC_*n* (*n*：0-1) 表示 PHY_ERRESC_*n* 错误是否发生。

0：未发生

1：发生

(RC)

Register 42.19. CSI_HOST_INT_MSK_PHY_REG (0x0114)



CSI_HOST_MASK_PHY_ERRSOTHS_*n* (*n*：0-1) 配置是否屏蔽 CSI_HOST_ST_PHY_ERRSOTHS_*n*。

0：屏蔽错误中断

1：启用错误中断

(R/W)

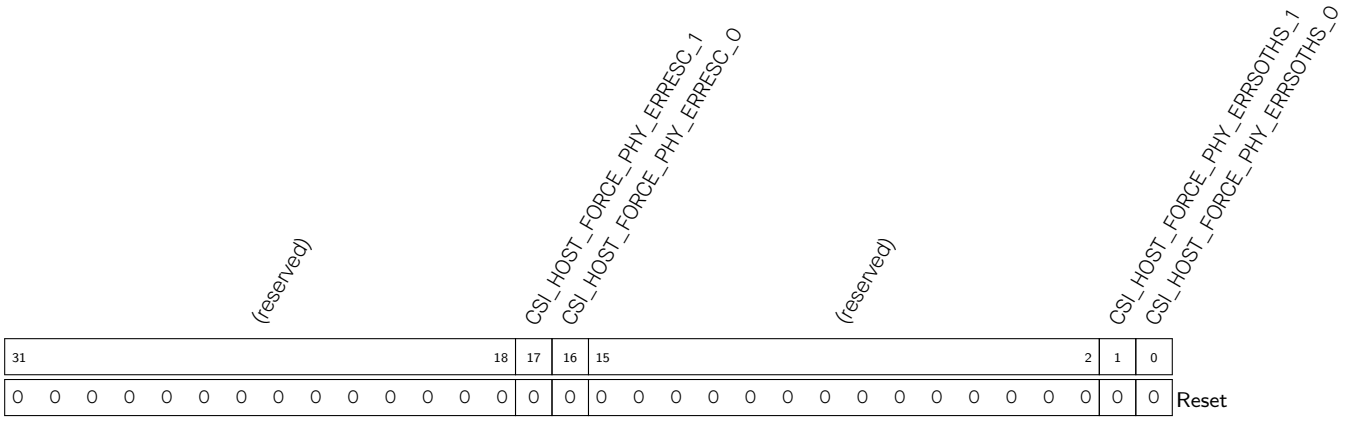
CSI_HOST_MASK_PHY_ERRESC_*n* (*n*：0-1) 配置是否屏蔽 CSI_HOST_ST_PHY_ERRESC_*n*。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.20. CSI_HOST_INT_FORCE_PHY_REG (0x0118)



CSI_HOST_FORCE_PHY_ERRSOTHS_*n* (*n*：0-1) 配置是否强制将 CSI_HOST_ST_PHY_ERRSOTHS_*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

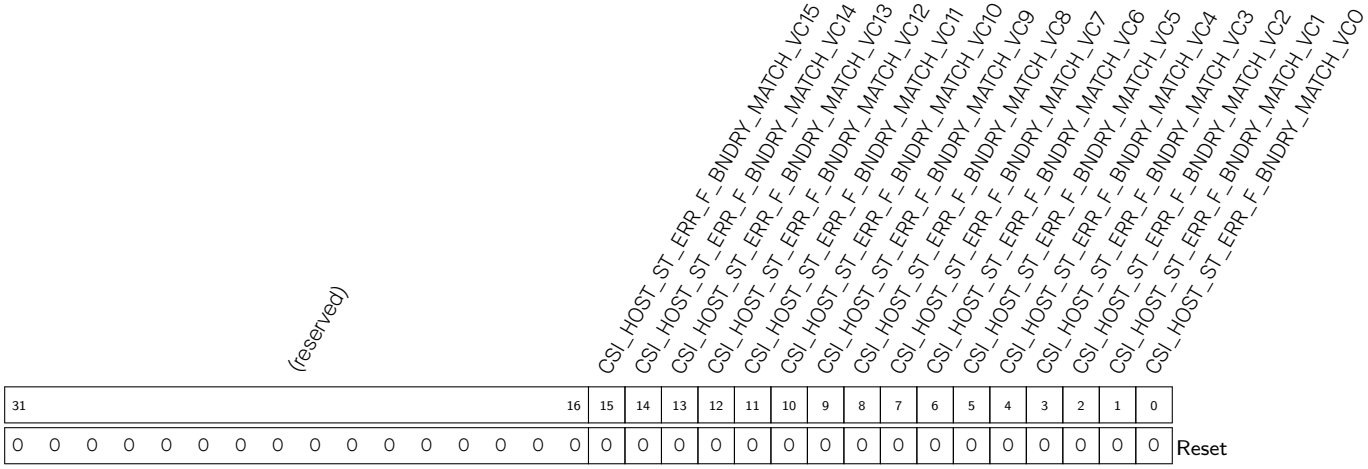
CSI_HOST_FORCE_PHY_ERRESC_*n* (*n*：0-1) 配置是否强制将 CSI_HOST_ST_PHY_ERRESC_*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.21. CSI_HOST_INT_ST_BNDRY_FRAME_FATAL_REG (0x0280)



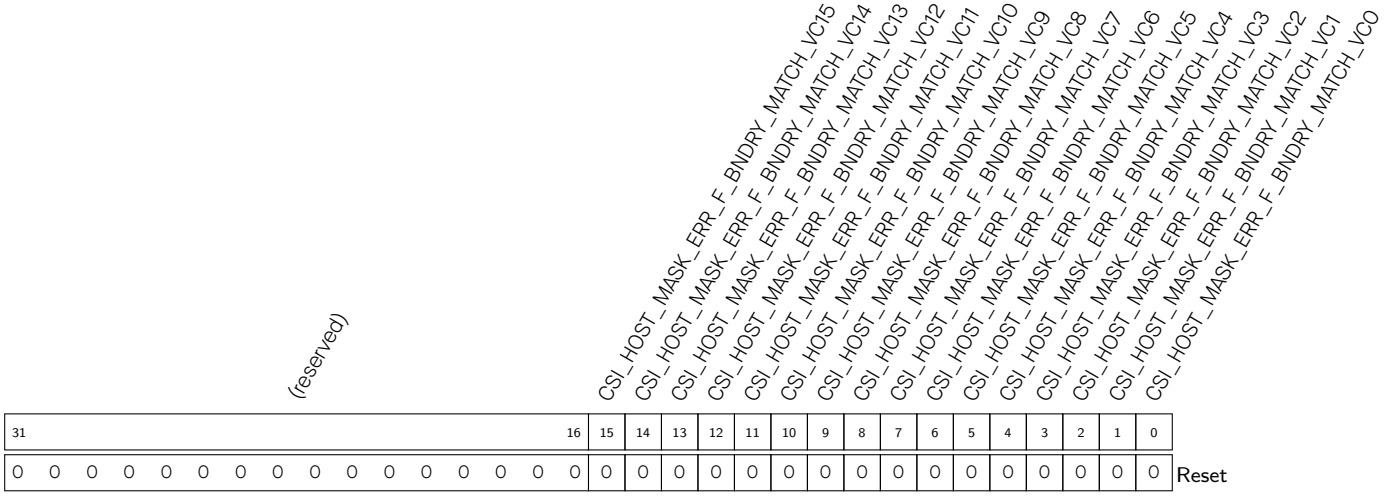
CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n* (*n*：0-15) 表示是否发生 ERR_F_BNDRY_MATCH_VC*n* 错误。

0：未发生

1：发生

(RO)

Register 42.22. CSI_HOST_INT_MSK_BNDRY_FRAME_FATAL_REG (0x0284)



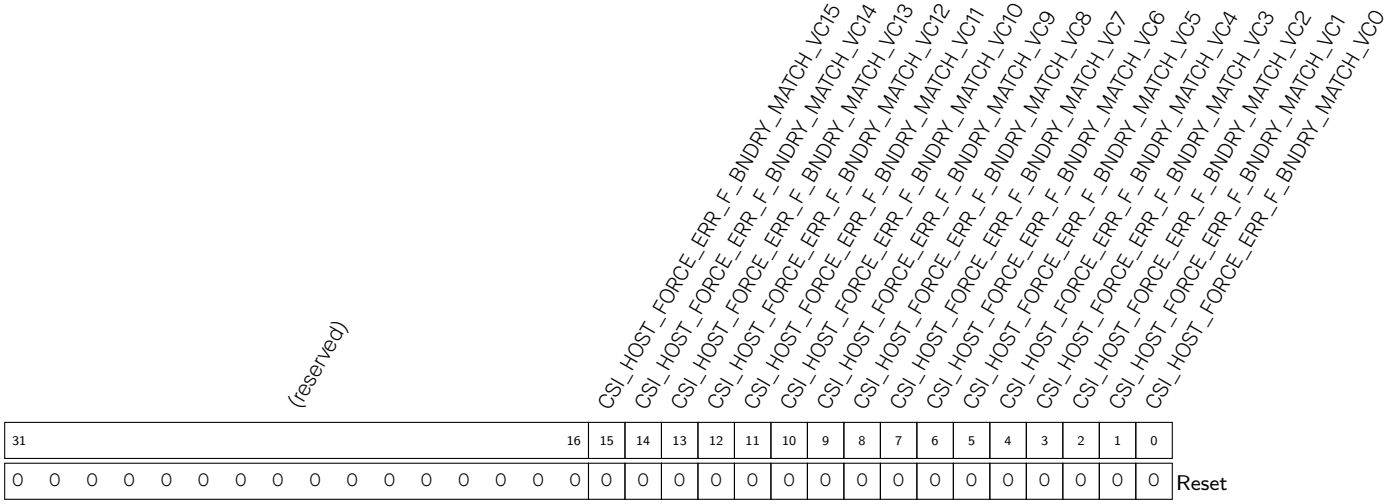
CSI_HOST_MASK_ERR_F_BNDRY_MATCH_VC*n* (*n*：0-15) 配 置 是 否 屏 蔽 CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n* 错误中断。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.23. CSI_HOST_INT_FORCE_BNDRY_FRAME_FATAL_REG (0x0288)



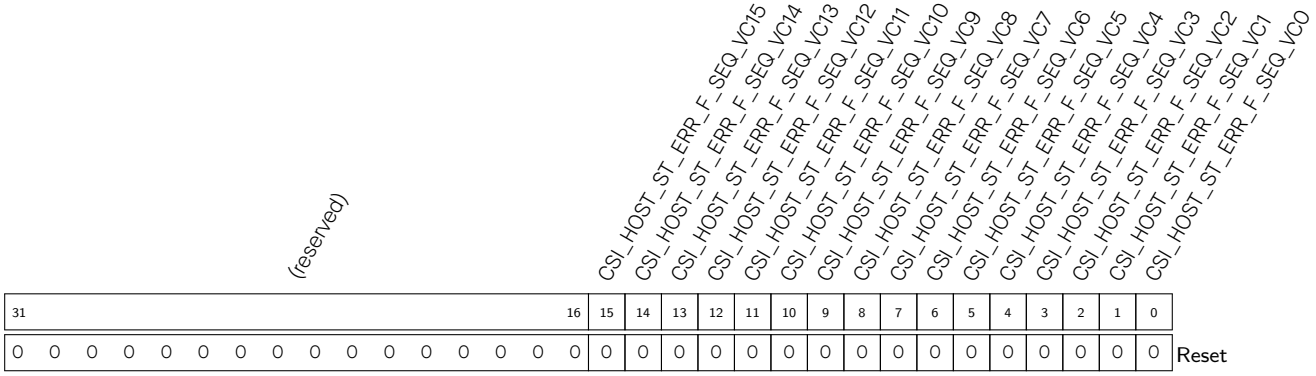
CSI_HOST_FORCE_ERR_F_BNDRY_MATCH_VC*n* (*n*：0-15) 配 置 是 否 强 制 将 CSI_HOST_ST_ERR_F_BNDRY_MATCH_VC*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.24. CSI_HOST_INT_ST_SEQ_FRAME_FATAL_REG (0x0290)



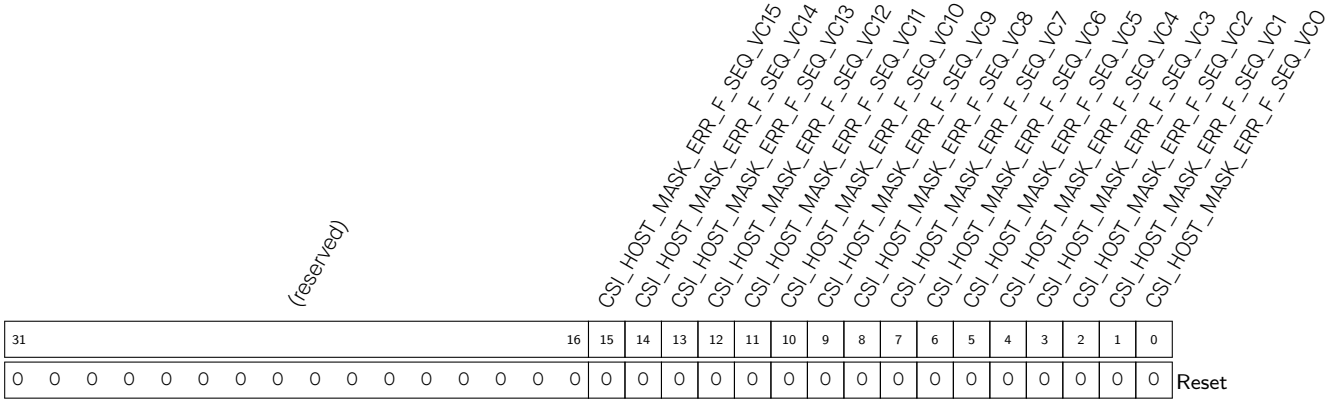
CSI_HOST_ST_ERR_F_SEQ_VC*n* (***n***：0-15) 表示是否发生 ERR_F_SEQ_VC*n* 错误。

0：未发生

1：发生

(RC)

Register 42.25. CSI_HOST_INT_MSK_SEQ_FRAME_FATAL_REG (0x0294)



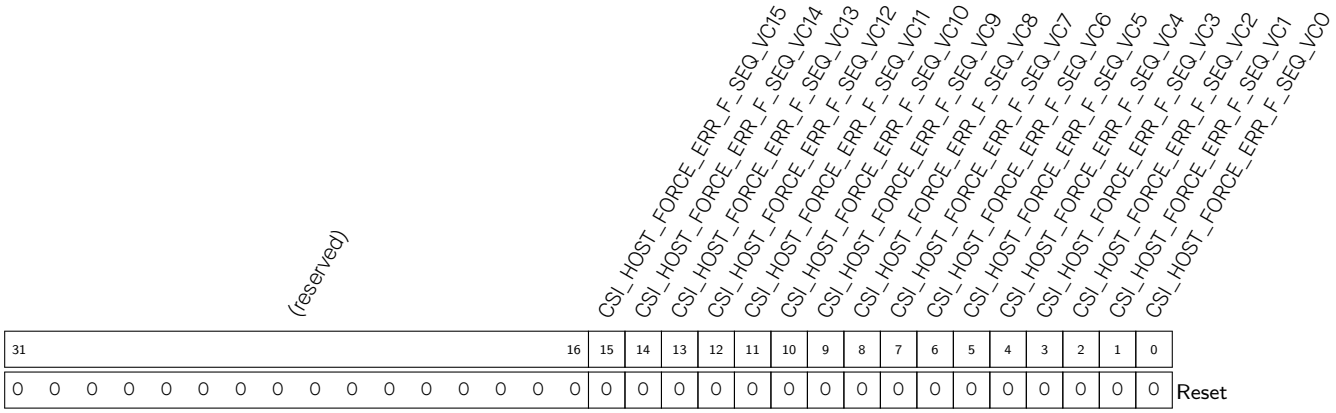
CSI_HOST_MASK_ERR_F_SEQ_VC*n* (***n***：0-15) 配置是否屏蔽 CSI_HOST_ST_ERR_F_SEQ_VC*n* 的错误中断。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.26. CSI_HOST_INT_FORCE_SEQ_FRAME_FATAL_REG (0x0298)



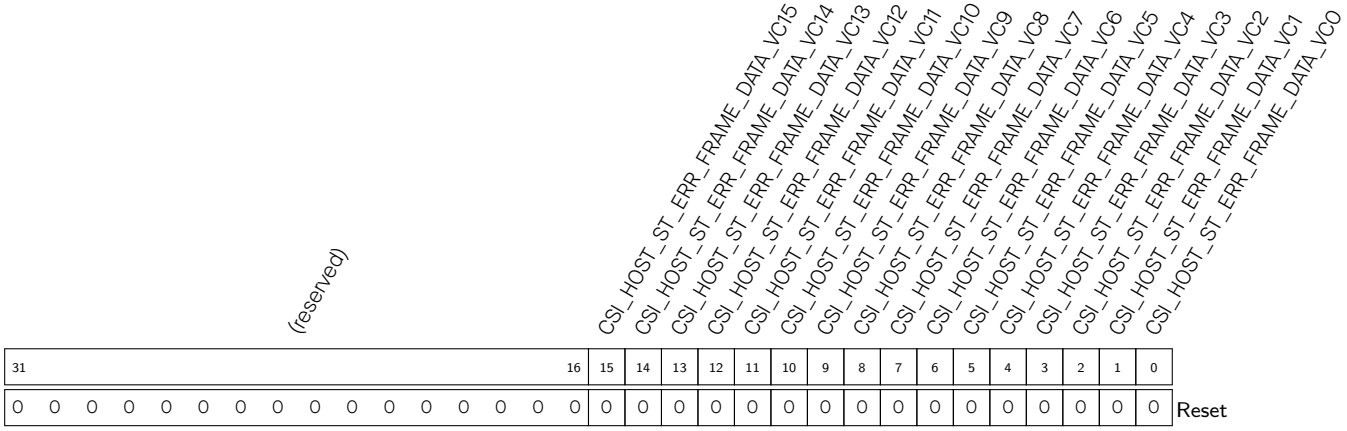
CSI_HOST_FORCE_ERR_F_SEQ_VC*n* (*n*：0-15) 配置是否强制将 CSI_HOST_ST_ERR_F_SEQ_VC*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.27. CSI_HOST_INT_ST_CRC_FRAME_FATAL_REG (0x02A0)



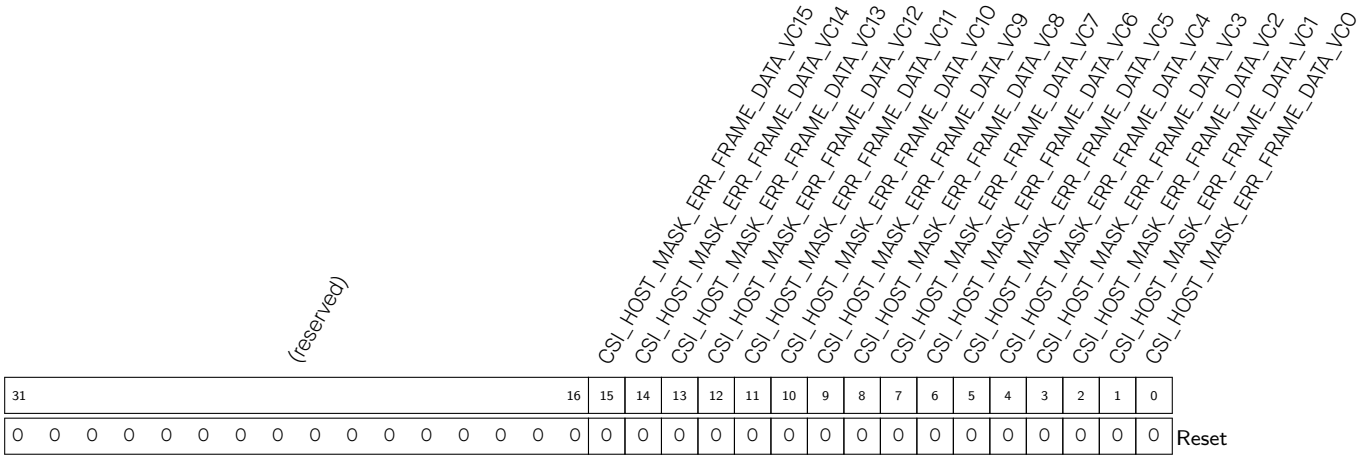
CSI_HOST_ST_ERR_FRAME_DATA_VC*n* (*n*：0-15) 表示是否发生 ERR_FRAME_DATA_VC*n* 错误。

0：未发生

1：发生

(RO)

Register 42.28. CSI_HOST_INT_MSK_CRC_FRAME_FATAL_REG (0x02A4)



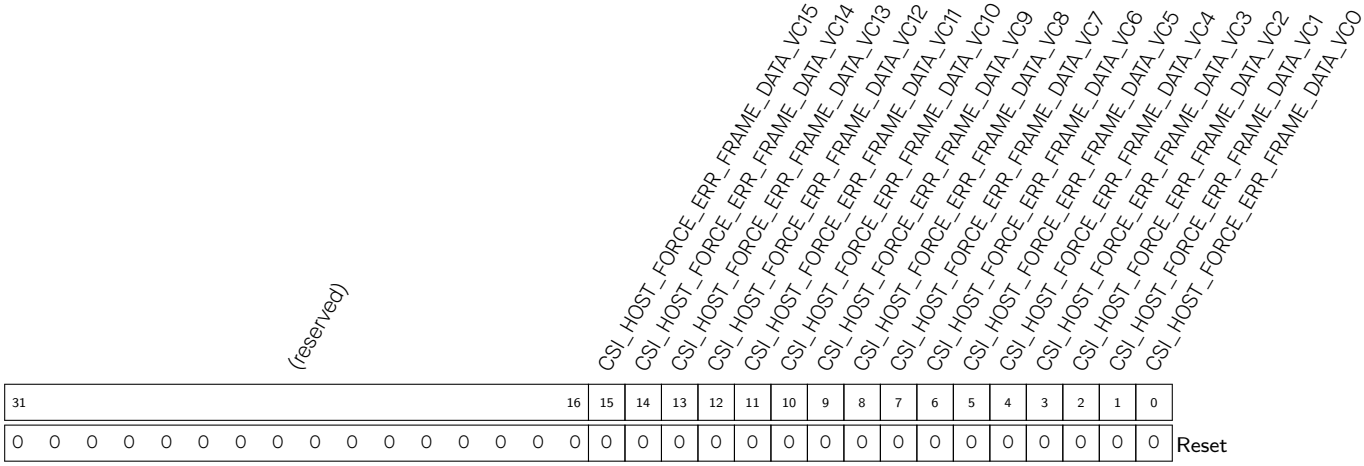
CSI_HOST_MASK_ERR_FRAME_DATA_VC*n* (*n*：0-15) 配置是否屏蔽 CSI_HOST_ST_ERR_FRAME_DATA_VC*n* 错误中断。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.29. CSI_HOST_INT_FORCE_CRC_FRAME_FATAL_REG (0x02A8)



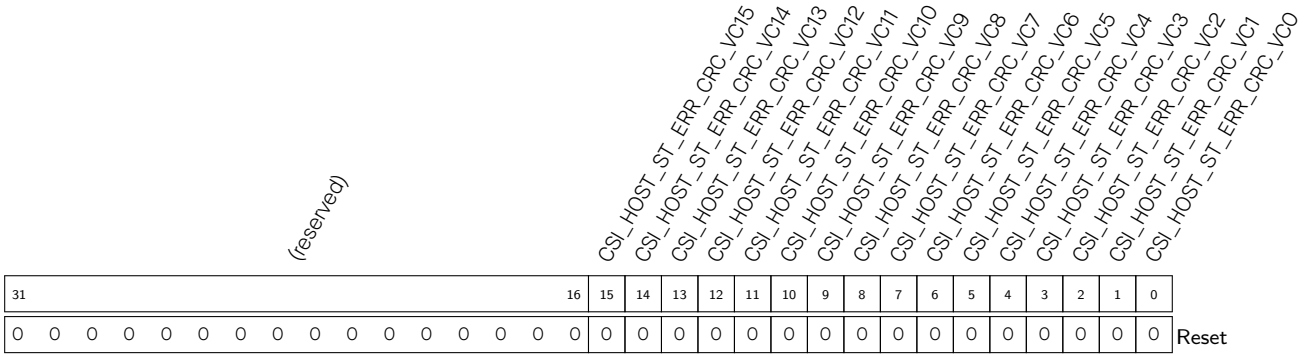
CSI_HOST_FORCE_ERR_FRAME_DATA_VC*n* (*n*：0-15) 配置是否强制将 CSI_HOST_ST_ERR_FRAME_DATA_VC*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.30. CSI_HOST_INT_ST_PLD_CRC_FATAL_REG (0x02B0)



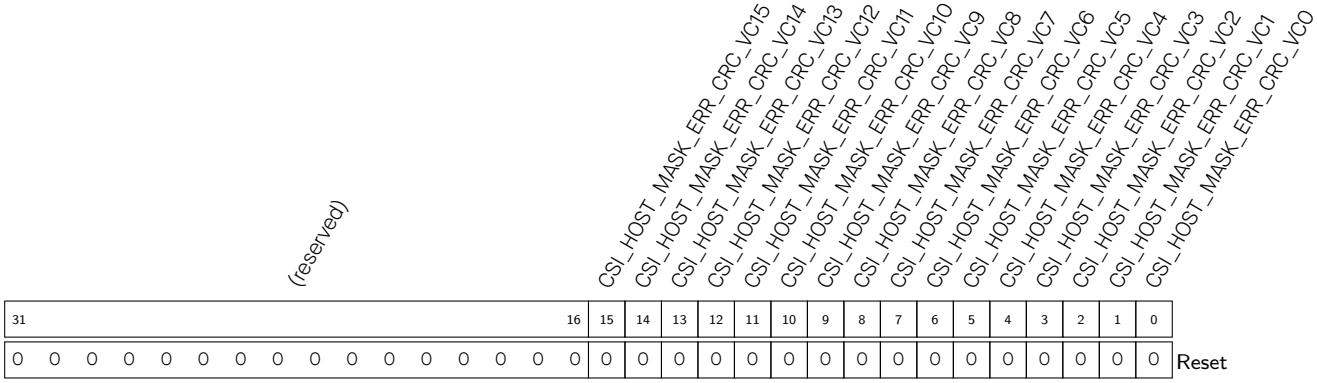
CSI_HOST_ST_ERR_CRC_VC*n* (*n*：0-15) 表示是否发生 ERR_CRC_VC*n* 错误。

0：未发生

1：发生

(RO)

Register 42.31. CSI_HOST_INT_MSK_PLD_CRC_FATAL_REG (0x02B4)



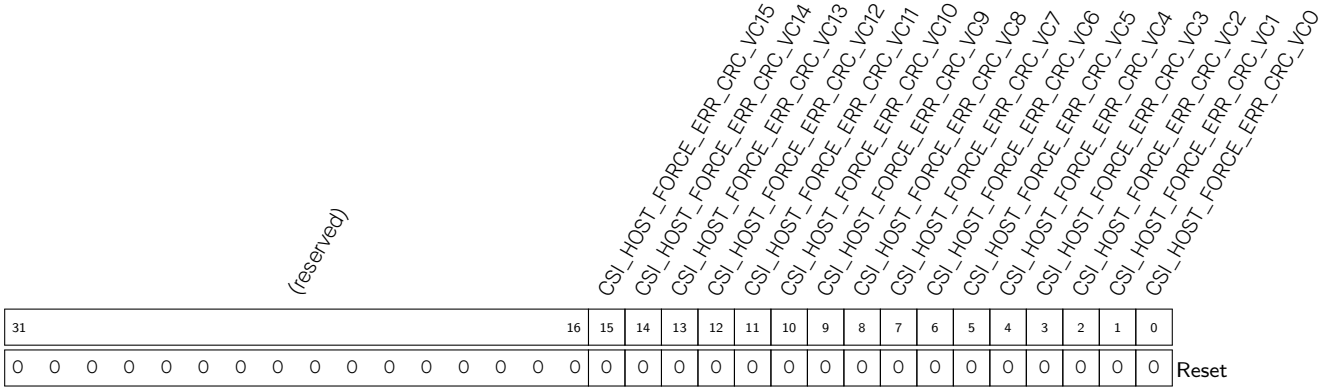
CSI_HOST_MASK_ERR_CRC_VC*n* (*n*：0-15) 配置是否屏蔽 CSI_HOST_ST_ERR_CRC_VC*n* 错误中断。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.32. CSI_HOST_INT_FORCE_PLD_CRC_FATAL_REG (0x02B8)



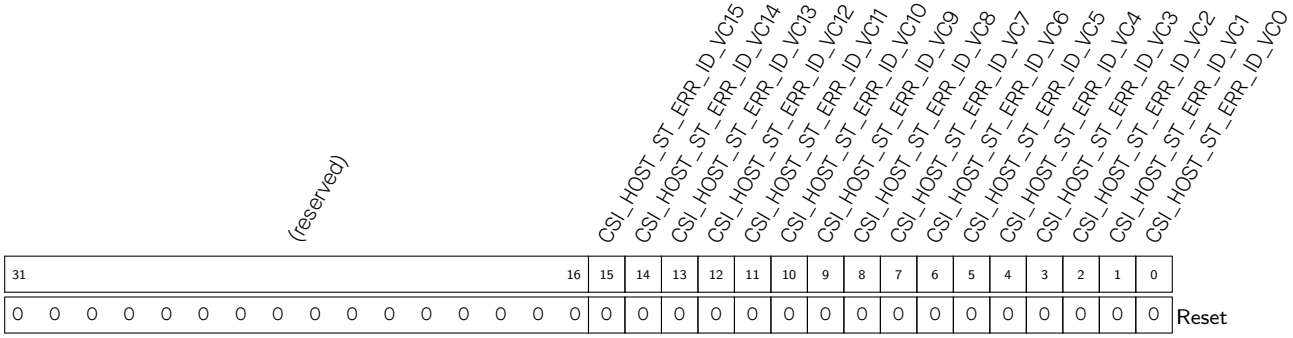
CSI_HOST_FORCE_ERR_CRC_VC*n* (*n*：0-15) 配置是否强制将 CSI_HOST_ST_ERR_CRC_VC*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.33. CSI_HOST_INT_ST_DATA_ID_REG (0x02C0)



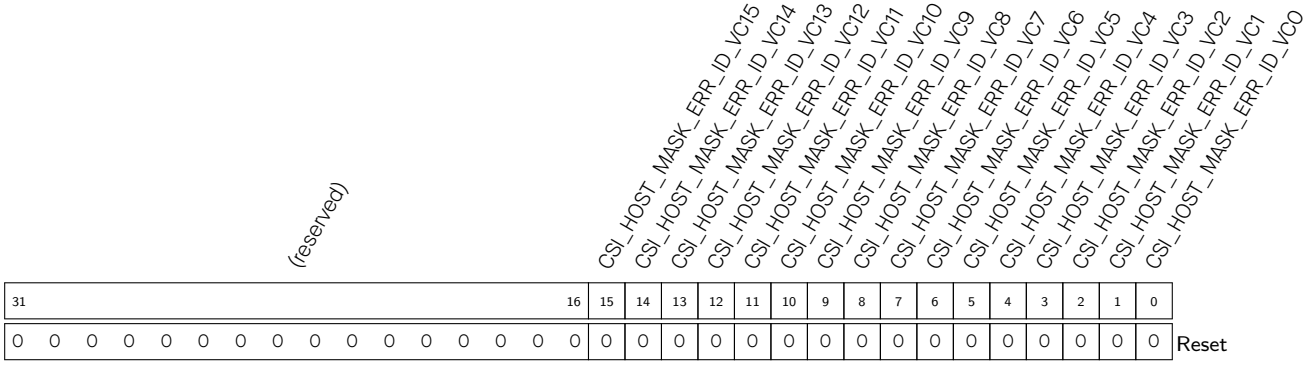
CSI_HOST_ST_ERR_ID_VC*n* (*n*：0-15) 表示 ERR_ID_VC*n* 错误是否发生。

0：未发生

1：发生

(RC)

Register 42.34. CSI_HOST_INT_MSK_DATA_ID_REG (0x02C4)



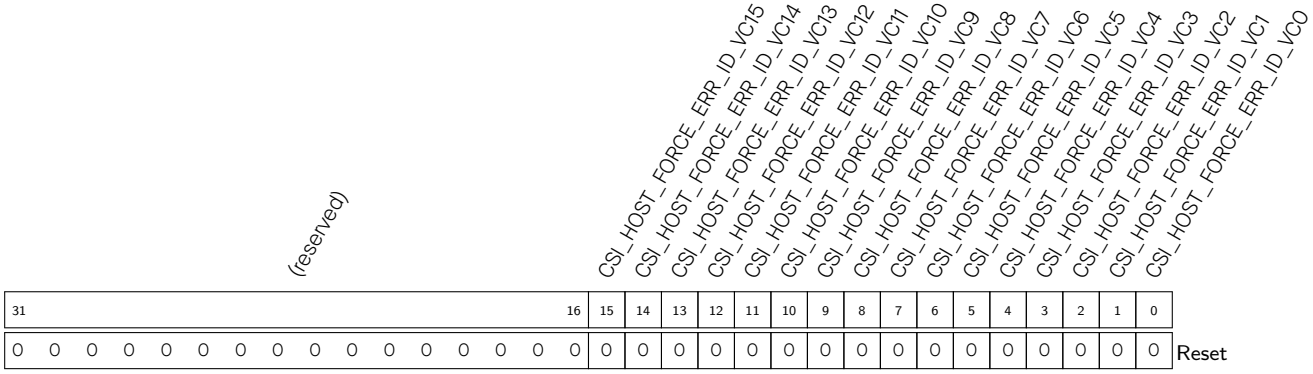
CSI_HOST_MASK_ERR_ID_VC*n* (*n*：0-15) 配置是否屏蔽 CSI_HOST_ST_ERR_ID_VC*n*。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.35. CSI_HOST_INT_FORCE_DATA_ID_REG (0x02C8)



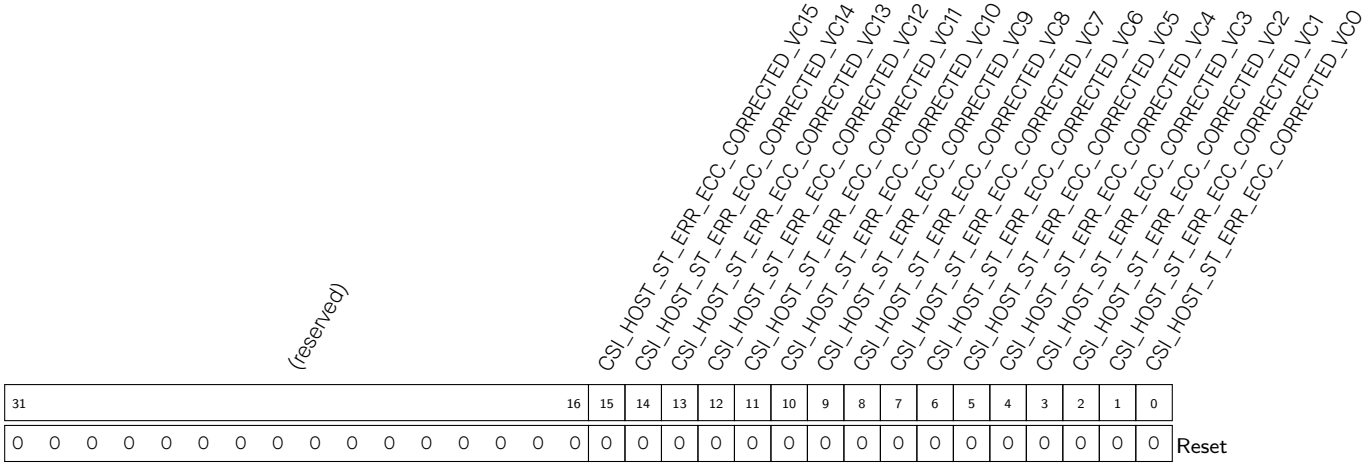
CSI_HOST_FORCE_ERR_ID_VC*n* (*n*：0-15) 配置是否强制将 CSI_HOST_ST_ERR_ID_VC*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.36. CSI_HOST_INT_ST_ECC_CORRECTED_REG (0x02D0)



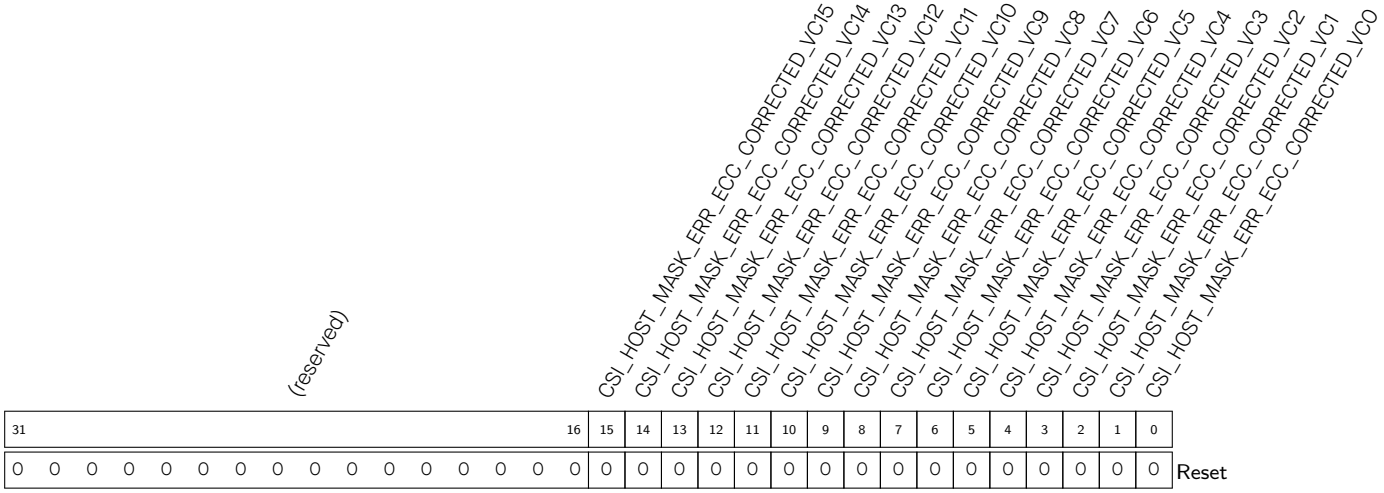
CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n* (*n*：0-15) 表示是否发生 ERR_ECC_CORRECTED_VC*n* 错误。

0：未发生

1：发生

(RO)

Register 42.37. CSI_HOST_INT_MSK_ECC_CORRECTED_REG (0x02D4)



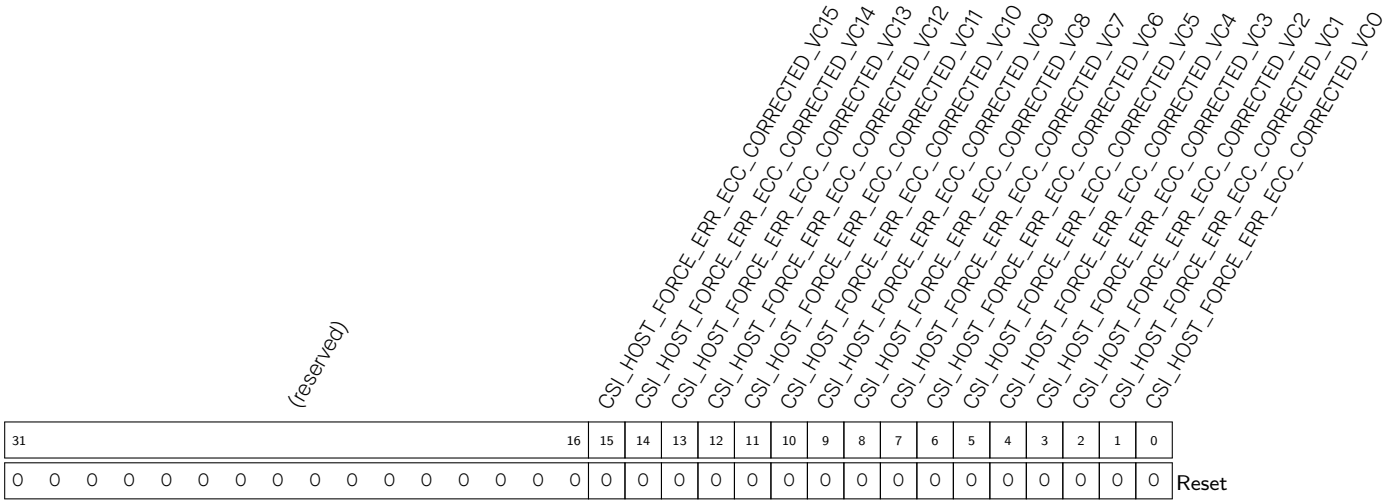
CSI_HOST_MASK_ERR_ECC_CORRECTED_VC*n* (*n*：0-15) 配置是否屏蔽 CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n* 错误中断。

0：屏蔽错误中断

1：启用错误中断

(R/W)

Register 42.38. CSI_HOST_INT_FORCE_ECC_CORRECTED_REG (0x02D8)



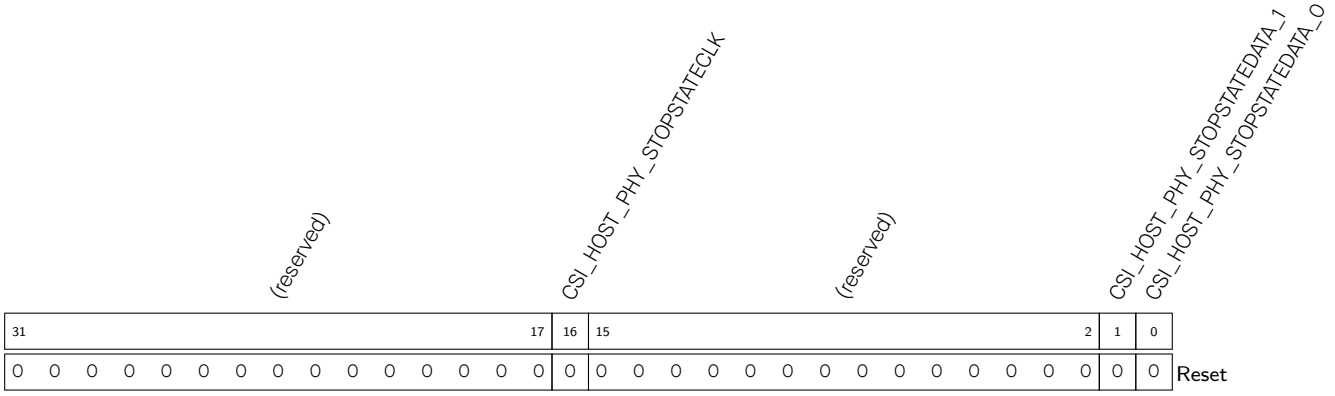
CSI_HOST_FORCE_ERR_ECC_CORRECTED_VC*n* (*n*：0-15) 配置是否强制将 CSI_HOST_ST_ERR_ECC_CORRECTED_VC*n* 设置为 1。

0：不强制设置

1：强制设置

(R/W)

Register 42.39. CSI_HOST_PHY_STOPSTATE_REG (0x004C)



CSI_HOST_PHY_STOPSTATEDATA_*n* (*n*：0-1) 表示数据通道 *n* 是否处于停止状态。

0：未处于停止状态

1：处于停止状态

(RO)

CSI_HOST_PHY_STOPSTATECLK 表示时钟通道是否处于停止状态。

0：未处于停止状态

1：处于停止状态

(RO)

第 43 章

MIPI DSI

43.1 介绍

MIPI DSI (Mobile Industry Processor Interface Display Serial Interface) 代表移动行业处理器接口显示串行接口，是一个标准接口协议，用于将显示器连接到移动设备，如智能手机、平板电脑和其他嵌入式系统。在 ESP32-P4 中，MIPI DSI 由三个部分组成：一个 DSI 桥接器，一个 MIPI DSI 主控制器，以及一个 MIPI 双向 D-PHY。它提供一个时钟通道和两条数据通道，并支持每个通道最高 1.5 Gbps 的通信速率，支持与符合 DSI 标准的显示器通信。DSI 属于 MIPI 联盟定义的一组通信协议。

DSI 桥接器提供系统内存与 MIPI DSI 主控制器之间的接口。它通过 VDMA 从系统内存接收图像数据，并通过显示像素接口 (DPI) 将图像数据输出到 MIPI DSI 主控制器中。MIPI DSI 主控制器实现了 MIPI DSI 规范中定义的协议功能。它从 DSI 桥接器接收图像数据，并通过 PHY 协议接口 (PPI) 将图像数据输出到 MIPI 双向 D-PHY。MIPI 双向 D-PHY 实现了 MIPI D-PHY 接口的双向通用通道的物理层，提供一个时钟通道和两个数据通道。PHY 仅用作主设备。

43.2 功能列表

MIPI DSI 在 ESP32-P4 中支持以下功能：

- 在从内存读取图像数据时，将 DSI 桥或 VDMA 配置为流控制器
- 支持在 DSI 桥中配置 DPI 视频时序
- 支持多种输入图像格式：RGB888、RGB666、RGB565、YUV422、YUV420、GRAY
- 支持各种 DPI 图像输出格式：RGB888、RGB666、RGB565
- 最大 DPI 频率：240 MHz
- MIPI DSI 主机符合 MIPI DSI 规范
- 符合 MIPI D-PHY 接口规范，修订版 1.1 的 PHY
- PHY 包含 PLL
- 连续和非连续时钟通道模式
- 支持通过数据通道 0 进入逃逸模式
- 在 HS TX 中，通道操作范围为 80 Mbps 到 1.5 Gbps
- 支持突发和非突发视频模式
- 支持通过 APB 接口的命令模式访问显示寄存器
- 支持 MIPI 联盟规范中定义的显示命令集 (DCS)

- 视频图案生成器
- 支持超低功耗模式

43.3 架构概述

图 43.3-1 显示了 MIPI DSI 的架构。

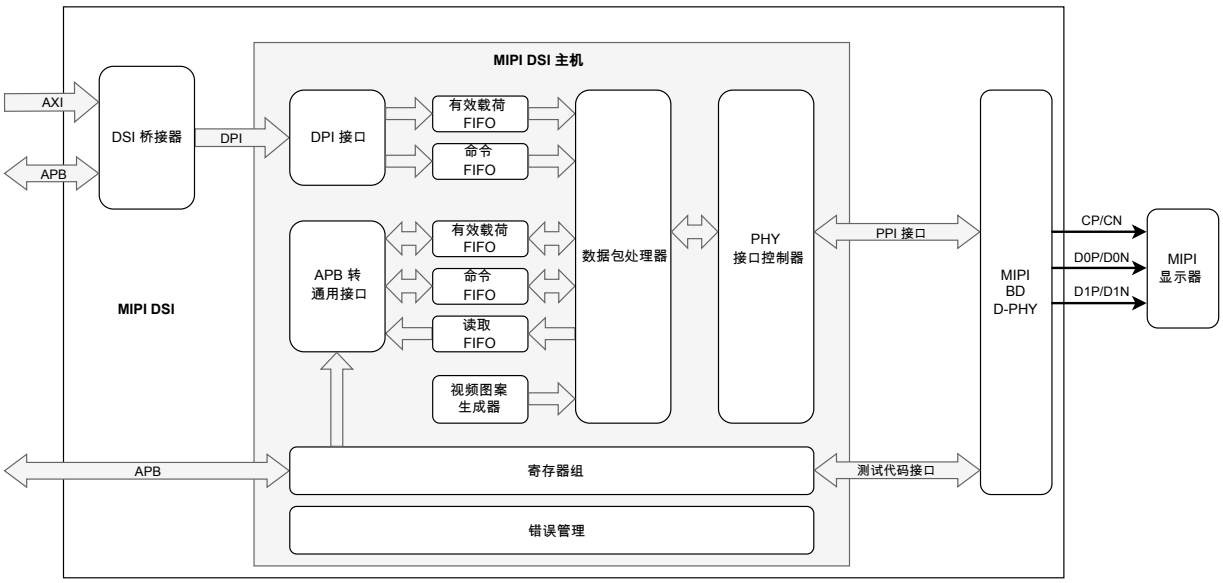


图 43.3-1. MIPI DSI 架构

- **DSI 桥接器**：DSI 桥接器通过 VDMA 从系统内存接收图像数据，将其转换为 DPI 数据格式，然后输出到 MIPI DSI 主机。
- **MIPI DSI 主机**：MIPI DSI 主机从两个来源接收图像数据：DSI 桥接器或视频图案生成器。然后，接收到的图像数据由数据包处理器根据 MIPI DSI 规范进行打包，并发送到 PHY 接口控制器。最后，图像数据被转换为 PPI 数据格式并输出到 MIPI DSI 双向 D-PHY。MIPI DSI 主机包括：
  - **DPI 接口**：从 DSI 桥接器接收像素数据并将其缓冲到 FIFO。
  - **寄存器组**：提供对配置和控制寄存器的访问。
  - **APB 到通用接口**：将 APB 操作转换为通用命令。
  - **视频图案生成器**：生成视频测试图案。
  - **数据包处理器**：安排链路内部的活动，包括数据包构建、生成 ECC CRC 代码、数据包接收等。
  - **PHY 接口控制器**：管理 PHY PPI 接口，包括启用低功耗 (LP) 传输/接收、高速 (HS) 传输等。
  - **错误管理**：监控并通知 DSI 链路上的错误状态。
- **MIPI DSI D-PHY**：MIPI DSI 双向 D-PHY 实现了具有一个时钟通道和两个数据通道的物理层，连接到外部显示驱动器。MIPI DSI 双向 D-PHY 还包括一个 PLL 以生成 HS TX 时钟。

43.4 功能描述

本节介绍了 MIPI 双向 (BD) D-PHY 的不同操作状态和模式，以及一些有助于 MIPI DSI 使用的主要功能。

43.4.1 DSI 桥接器

DSI Bridge 通过 VDMA 从系统内存接收图像数据。数据传输流控制器可以通过 DSI_BRG_DSI_DMA_FLOW_CONTROLLER 配置为 VDMA 或 DSI Bridge。建议使用 VDMA 作为流控制器以获得更好的性能。

当使用 VDMA 作为流控制器接收图像数据时，可以将图像拆分为多个块传输。要启用此功能，请将 DSI_BRG_DMA_MULTIBLK_EN 设置为 1，并将 DSI_BRG_DMA_FLOW_MULTIBLK_NUM 配置为所需值。有关块传输的详细说明，请参见 *7 VDMA 控制器 (VDMA)* 的 *传输层次结构*小节。

DSI Bridge 支持 RGB888、RGB666、RGB565、YUV422、YUV420 和 GRAY 数据类型的输入。对于 YUV422，DSI Bridge 支持 8 种不同的像素排列。用户可以通过配置 DSI_BRG_DATA_IN_TYPE 寄存器来选择输入数据类型是否为 YUV，然后使用 DSI_BRG_RAW_TYPE 指定具体的输入格式。对于发送到 DSI Host 的输出格式，使用 DSI_BRG_DPI_TYPE 指定 RGB 格式。

所有支持的输入像素排列，参见表 43.4-1。

表 43.4-1. 输入像素排列

颜色模式	Addr + 7	Addr + 6	Addr + 5	Addr + 4	Addr + 3	Addr + 2	Addr + 1	Addr + 0
RGB888	G2[7:0]	B2[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
	B5[7:0]	R4[7:0]	G4[7:0]	B4[7:0]	R3[7:0]	G3[7:0]	B3[7:0]	R2[7:0]
	R7[7:0]	G7[7:0]	B7[7:0]	R6[7:0]	G6[7:0]	B6[7:0]	R5[7:0]	G5[7:0]
RGB565	R3[4:0]G3[5:3]	G3[2:0]B3[4:0]	R2[4:0]G2[5:3]	G2[2:0]B2[4:0]	R1[4:0]G1[5:3]	G1[2:0]B1[4:0]	R0[4:0]G0[5:3]	G0[2:0]B0[4:0]
YUV422 DSI_BRG_YUV422_FORMAT = 0 DSI_BRG_YUV_PIX_ENDIAN = 0	Y3[7:0]	U3[7:0]	Y2[7:0]	V2[7:0]	Y1[7:0]	U1[7:0]	Y0[7:0]	V0[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 0 DSI_BRG_YUV_PIX_ENDIAN = 1	Y0[7:0]	U0[7:0]	Y1[7:0]	V1[7:0]	Y2[7:0]	U2[7:0]	Y3[7:0]	V3[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 1 DSI_BRG_YUV_PIX_ENDIAN = 0	Y3[7:0]	V3[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	V1[7:0]	Y0[7:0]	U0[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 1 DSI_BRG_YUV_PIX_ENDIAN = 1	Y0[7:0]	V0[7:0]	Y1[7:0]	U1[7:0]	Y2[7:0]	V2[7:0]	Y3[7:0]	U3[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 2 DSI_BRG_YUV_PIX_ENDIAN = 0	U3[7:0]	Y3[7:0]	V2[7:0]	Y2[7:0]	U1[7:0]	Y1[7:0]	V0[7:0]	Y0[7:0]
YUV422 DSI_BRG_YUV422_FORMAT = 2 DSI_BRG_YUV_PIX_ENDIAN = 1	U0[7:0]	Y0[7:0]	V1[7:0]	Y1[7:0]	U2[7:0]	Y2[7:0]	V3[7:0]	Y3[7:0]

见下页

表 43.4-1 – 接上页

颜色模式	Addr + 7	Addr + 6	Addr + 5	Addr + 4	Addr + 3	Addr + 2	Addr + 1	Addr + 0
YUV422 DSI_BRG_YUV422 _FORMAT = 3 DSI_BRG_YUV _PIX_ENDIAN = 0	V3[7:0]	Y3[7:0]	U2[7:0]	Y2[7:0]	V1[7:0]	Y1[7:0]	U0[7:0]	Y0[7:0]
YUV422 DSI_BRG_YUV422 _FORMAT = 3 DSI_BRG_YUV _PIX_ENDIAN = 1	V0[7:0]	Y0[7:0]	U1[7:0]	Y1[7:0]	V2[7:0]	Y2[7:0]	U3[7:0]	Y3[7:0]
YUV420 Odd lines (1, 3, 5...)	Y4[7:0]	U4[7:0]	Y3[7:0]	Y2[7:0]	U2[7:0]	Y1[7:0]	Y0[7:0]	U0[7:0]
	U10[7:0]	Y9[7:0]	Y8[7:0]	U8[7:0]	Y7[7:0]	Y6[7:0]	U6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	U14[7:0]	Y13[7:0]	Y12[7:0]	U12[7:0]	Y11[7:0]	Y10[7:0]
YUV420 Even lines (2, 4, 6...)	Y4[7:0]	V4[7:0]	Y3[7:0]	Y2[7:0]	V2[7:0]	Y1[7:0]	Y0[7:0]	V0[7:0]
	V10[7:0]	Y9[7:0]	Y8[7:0]	V8[7:0]	Y7[7:0]	Y6[7:0]	V6[7:0]	Y5[7:0]
	Y15[7:0]	Y14[7:0]	V14[7:0]	Y13[7:0]	Y12[7:0]	V12[7:0]	Y11[7:0]	Y10[7:0]
GRAY	P7[7:0]	P6[7:0]	P5[7:0]	P4[7:0]	P3[7:0]	P2[7:0]	P1[7:0]	P0[7:0]

DSI Bridge 通过 DPI 接口输出数据，并支持由 DSI_BRG_DPI_CONFIG 指定的几种 RGB 输出位排列。所有支持的位排列，请参见表 43.4-2。

表 43.4-2. DPI 位排列

	RGB565			RGB666		RGB888
DSI_BRG_DPI_CONFIG	0	1	2	0	1	0
bit-23						R7
bit-22						R6
bit-21			R4		R5	R5
bit-20		R4	R3		R4	R4
bit-19		R3	R2		R3	R3
bit-18		R2	R1		R2	R2
bit-17		R1	R0	R5	R1	R1
bit-16		R0		R4	R0	R0
bit-15	R4			R3		G7
bit-14	R3			R2		G6
bit-13	R2	G5	G5	R1	G5	G5
bit-12	R1	G4	G4	R0	G4	G4
bit-11	R0	G3	G3	G5	G3	G3
bit-10	G5	G2	G2	G4	G2	G2
bit-9	G4	G1	G1	G3	G1	G1
bit-8	G3	G0	G0	G2	G0	G0
bit-7	G2			G1		B7
bit-6	G1			G0		B6
bit-5	G0		B4	B5	B5	B5
bit-4	B4	B4	B3	B4	B4	B4
bit-3	B3	B3	B2	B3	B3	B3
bit-2	B2	B2	B1	B2	B2	B2
bit-1	B1	B1	B0	B1	B1	B1
bit-0	B0	B0		B0	B0	B0

43.4.2 DSI 主机

43.4.2.1 视频传输

DSI Host 使用 DPI 作为输入接口。它用于在视频模式下从 HOST 向外设传输实时像素流。

有两种视频传输模式：

- **突发模式**：在突发模式下，一行像素会被缓冲到 FIFO 中，并在一个单独的数据包中无中断地传输。这使得像素行能够在短时间内发送出去，然后让通道回到低功耗 (LP) 状态，从而降低功耗，并为其他数据传

输节省时间。

- **非突发模式**：在非突发模式下，一行像素被分成几个数据包。这种模式用于匹配像素带宽与 DSI 链路带宽，并减少主机和外设的内存需求。非突发模式有两种：
  - 带同步脉冲的非突发模式：允许外设准确重建原始视频时序，包括同步脉冲宽度。
  - 带同步事件的非突发模式：不需要准确重建同步脉冲。

如果主机和外设满足以下要求，建议选择突发模式。否则，像素数据可能会丢失，导致显示外设发生故障。

- 显示外设支持在一个单一块中接收一行完整像素，并且接收缓冲区不会溢出。
- DSI 输出带宽远高于 DPI 输入带宽，该带宽差异使传输通道能够在每一行数据传输完成后进入低功耗 (LP) 状态。

如果无法满足任何要求，则应选择非突发模式。在非突发模式下，DSI 主机应配置为匹配 DSI 输出比率和 DPI 输入比率。以下方程显示了传输参数应如何配置（左侧表示 DSI 输出比率，右侧表示 DPI 输入比率）：

- 如果启用了空数据包 (DSI_HOST_VID_NULL_SIZE $\neq$ 0)：

$$\begin{aligned} &lanebyteclk_period \times DSI_HOST_VID_NUM_CHUNKS \\ &\quad \times (DSI_HOST_VID_PKT_SIZE \times bytes_per_pixel + 12 \\ &\quad + DSI_HOST_VID_NULL_SIZE)/number_of_lanes \\ &= dpiclk_period \times pixels_per_line \end{aligned}$$

- 如果禁用了空数据包 (DSI_HOST_VID_NULL_SIZE $=$ 0)：

$$\begin{aligned} &lanebyteclk_period \times DSI_HOST_VID_NUM_CHUNKS \\ &\quad \times (DSI_HOST_VID_PKT_SIZE \times bytes_per_pixel + 6) \\ &/number_of_lanes = dpiclk_period \times pixels_per_line \end{aligned}$$

有关 lanebyteclk 和 dpiclk 的定义，请参阅 43.4.4。

43.4.2.2 命令传输

DSI 主机允许通过 APB 从接口，以命令模式传输 DSI 规范中厂商自定义的通用命令。

命令数据包使用 APB 寄存器构建：

- **DSI_HOST_GEN_PLD_DATA_REG**：向该寄存器写入数据会将数据作为命令数据包的有效载荷发送。读取该寄存器会返回回读数据包的有效载荷。
- **DSI_HOST_GEN_HDR_REG**：包含命令数据包头类型和头数据。写入此寄存器会触发命令传输。

有效数据包如下：

- 通用写短包 0 参数
- 通用写短包 1 参数
- 通用写短包 2 参数
- 通用读短包 0 参数
- 通用读短包 1 参数

- 通用读短包 2 参数
- 最大读包配置
- 通用长写包

在视频模式的消隐期 (Blanking) 或低功耗期 (BLLP) 内，命令数据包可以在高速 (HS) 或低速 (LP) 通道状态下传输。BLLP 对应于图 43.4-1 中的阴影区域。

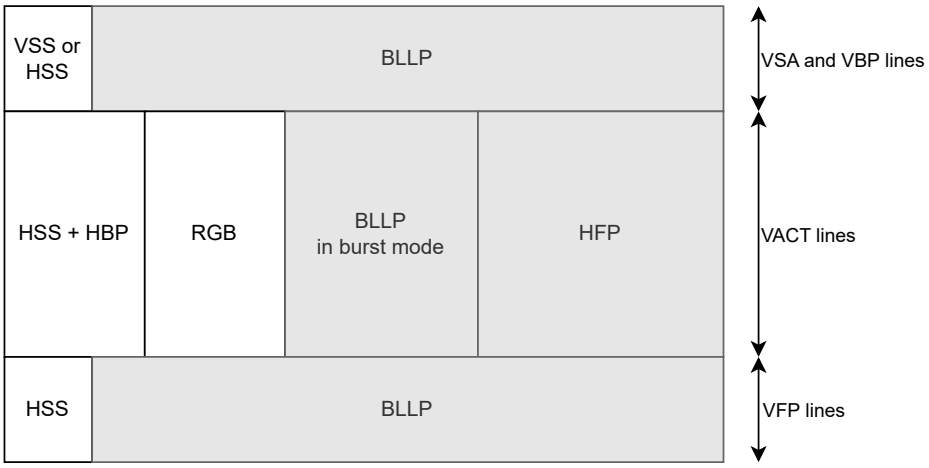


图 43.4-1. 命令传输周期

43.4.2.2.1 LP 模式下的命令传输

要在视频传输期间以 LP 模式发送命令，请将 DSI_HOST_LP_CMD_EN 设置为 1。在这种情况下，即使 DSI_HOST_VID_MODE_CFG_REG 的第 8 到 13 位设置为 0，仍然可以发送命令。

- DSI_HOST_OUTVACT_LPCMD_TIME 配置在 VSA、VBP 和 VFP 期间以 LP 模式传输命令的可用时间。此时间基于逃逸时钟确定，单位为字节。非突发带同步脉冲的 outvact_lpcmd_time 如图 43.4-2 所示；突发或非突发带同步事件的 outvact_lpcmd_time 如图 43.4-3 所示。

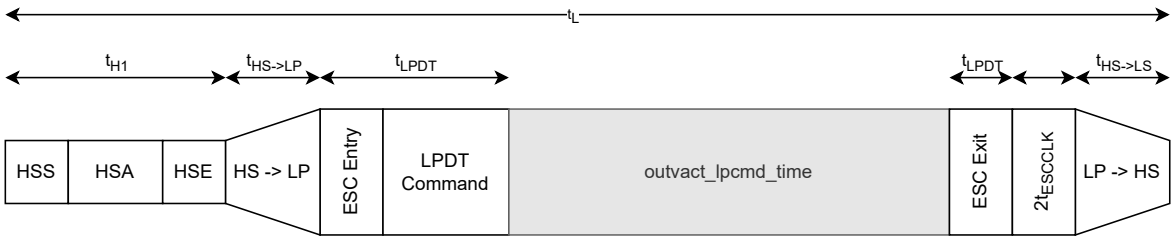


图 43.4-2. 非突发带同步脉冲的 outvact_lpcmd_time

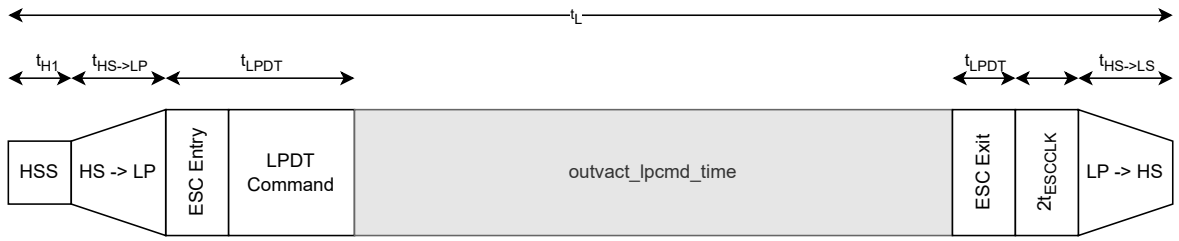


图 43.4-3. 突发或非突发带同步事件的 outvact_lpcmd_time

outvact_lpcmd_time 的计算公式为：

$$outvact_lpcmd_time = t_L - (t_{H1} + t_{HS \to LP} + t_{LP \to HS} + t_{LPDT} + 2 \times t_{ESCCLK}) / (2 \times 8 \times t_{ESCCLK})$$

- DSI_HOST_INVACT_LPCMD_TIME 配置在 HFP 期间以 LP 模式传输命令的可用时间。此时间基于逃逸时钟确定，单位为字节。非突发带同步脉冲的 invact_lpcmd_time 如图 43.4-4 所示；非突发带同步事件的 invact_lpcmd_time 如图 43.4-5 所示。

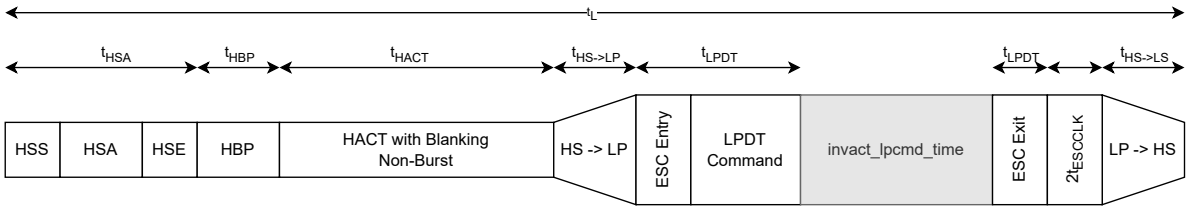


图 43.4-4. 非突发带同步脉冲的 invact_lpcmd_time

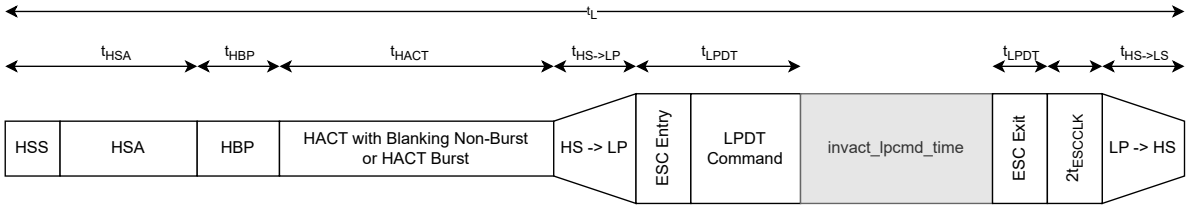


图 43.4-5. 突发或非突发带同步事件的 invact_lpcmd_time

invact_lpcmd_time 的计算公式为：

$$invact_lpcmd_time = t_L - (t_{HSA} + t_{HBP} + t_{HACT} + t_{HS \to LP} + t_{LP \to HS} + t_{LPDT} + 2 \times t_{ESCCLK}) / (2 \times 8 \times t_{ESCCLK})$$

43.4.2.2.2 HS 模式下的命令传输

如果 DSI_HOST_LP_CMD_EN 设置为 0，则在视频传输期间以 HS 模式发送命令。在这种情况下，不需要配置任何寄存器。

43.4.2.3 视频图案生成器

视频图案生成器可以输出水平或垂直的色带或 BER 测试图案。有关图案分辨率，请参阅 时钟和显示分辨率。以下寄存器字段用于配置视频图案生成器：

- DSI_HOST_VPG_ORIENTATION：指示色带的方向
- DSI_HOST_VPG_MODE：选择图案
- DSI_HOST_VPG_EN：启用视频图案生成器

43.4.2.3.1 色带图案

色带有 8 个颜色：白色、黄色、青色、绿色、品红色、红色、蓝色和黑色。每个颜色的宽度可以通过将图案大小除以 8 来计算。如果大小不能被 8 整除，则最后一个颜色黑色包含剩余的大小。图 43.4-6 和 43.4-7 分别显示了垂直和水平色带。

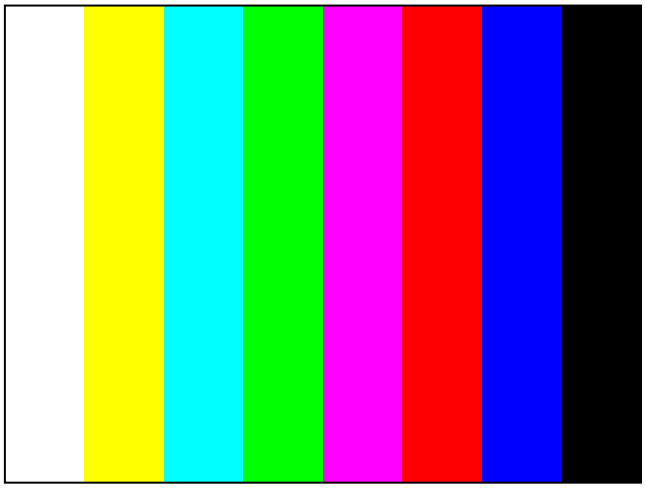


图 43.4-6. 垂直色带

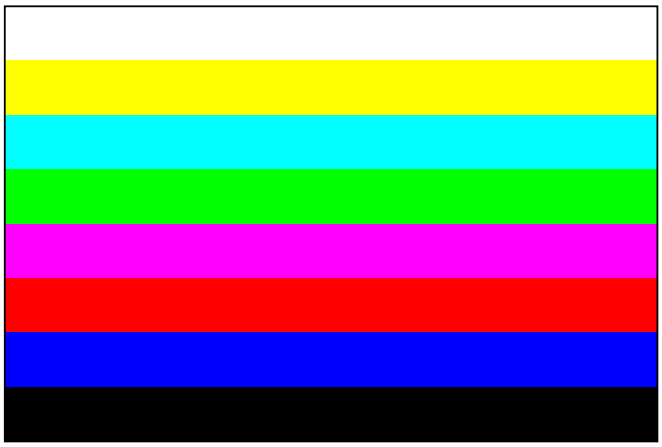


图 43.4-7. 水平色带

表 43.4-3 显示了色带图案的 RGB 编码。

表 43.4-3. 色带编码

	白色	黄色	青色	绿色	品红色	红色	蓝色	黑色
R	高	高	低	低	高	高	低	低
G	高	高	高	高	低	低	低	低
B	高	低	高	低	高	低	高	低

43.4.2.3.2 BER 测试图案

BER 测试图案用于测试设备 RX PHY 是否正确接收数据。生成的数据图案包括：

- X 字节的 0xAA（高频图案）
- X 字节的 0x33（中频图案）
- X 字节的 0xF0（低频图案）
- X 字节的 0x7F（孤立 0 图案）
- X 字节的 0x55（高频图案）
- X 字节的 0xCC（中频图案）
- X 字节的 0x0F（低频图案）
- Y 字节的 0x80（孤立 1 图案）

X 和 Y 取决于图案分辨率，在大多数情况下 Y 等于 X。使用 RGB888 颜色编码和八的倍数的水平分辨率，图 43.4-8 显示了在 DSI 显示器上出现的图案：

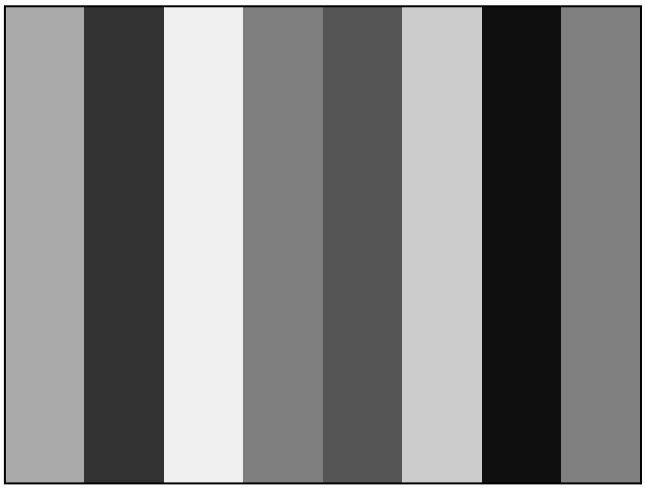


图 43.4-8. RGB888 BER 测试图案

43.4.2.4 错误报告

DSI_HOST_INT_ST0/1_REG 寄存器指示 DSI 主机错误报告。这些寄存器还可以触发中断以通知系统。要触发错误中断，应该清除相应的掩码寄存器 DSI_HOST_INT_MSK0/1_REG 中的位。DSI_HOST_INT_ST0/1_REG 在读

取操作后将被清除。

43.4.3 D-PHY

本节描述了 D-PHY 操作模式和 PLL 配置。

43.4.3.1 D-PHY 操作模式

图 43.4-9 显示了 PHY 初始化模式和运行模式。

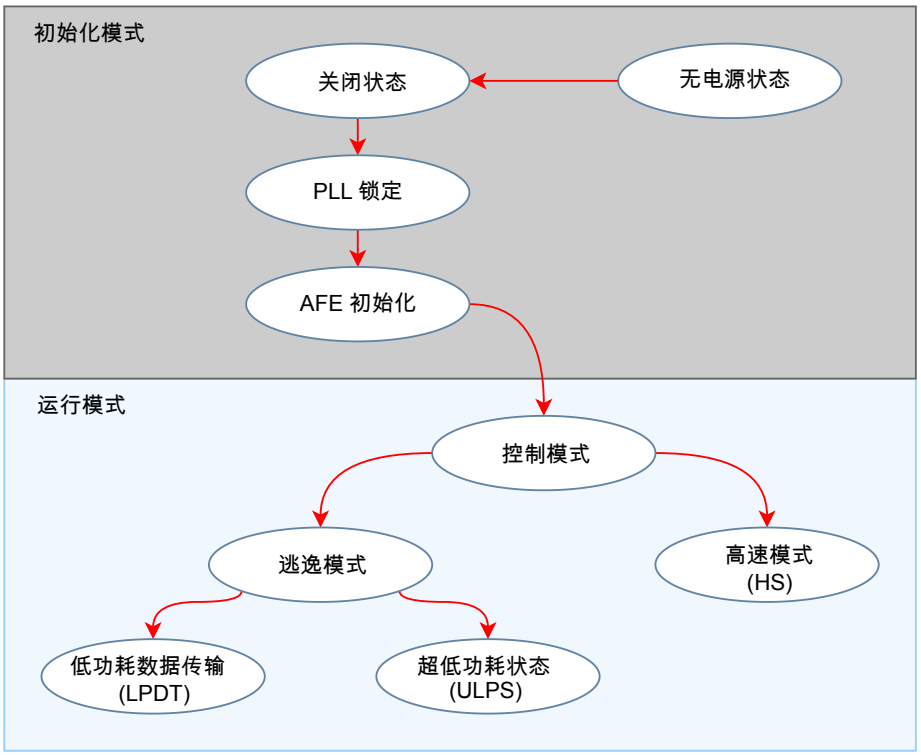


图 43.4-9. PHY 运行模式

43.4.3.1.1 无电源状态 (No-Power)

无电源 (No-Power) 状态的特征是没有对 PHY 提供任何供电电压。

对于 ESP32-P4 的 MIPI DSI，当电源 VDD_HP_*n* (n: 0-3) 和 VDD_MIPI_DPHY 关闭时，MIPI BD D-PHY 处于无电源状态。

43.4.3.1.2 关闭状态 (Shutdown)

在关闭状态下，会禁用所有模拟模块，重置所有数字逻辑，DP/DN 和 CP/CN 通道处于高阻抗 (Hi-Z)。要启用 PHY 的关闭模式，将 DSI_HOST_PHY_RSTZ 和 DSI_HOST_PHY_SHUTDOWNZ 设置为 0。

在离开关闭状态之前应配置一些参数，包括要使用的通道数量、通道操作频率范围，和 PLL 配置。

MIPI BD D-PHY 由两个数据通道组成。配置 DSI_HOST_N_LANES 以选择使用一个或两个数据通道。

默认情况下，PHY 设置为在 80-110 Mbps 的较低范围内操作。为了适应更高的比特率，可以使用适当的代码，在测试代码 测试代码 0x44 中调整 HS 频率范围（hsfreqrange 字段）。如果 PHY 始终在相同的比特率下工作，建议在关闭状态下进行此调整，因为控制接口独立于 MIPI BD D-PHY 的其它部分。支持的范围如表 43.4-4 所示：

表 43.4-4. MIPI BD D-PHY 支持的频率范围

频率范围 (Mbps)	hsfreqrange[5:0]*
80-89	000000
90-99	010000
100-109	100000
110-129	000001
130-139	010001
140-149	100001
150-169	000010
170-179	010010
180-199	100010
200-219	000011
220-239	010011
240-249	100011
250-269	000100
270-299	010100
300-329	000101
330-359	010101
360-399	100101
400-449	000110
450-499	010110
500-549	000111
550-599	010111
600-649	001000
650-699	011000
700-749	001001
750-799	011001
800-849	101001
850-899	111001
900-949	001010
950-999	011010
1000-1049	101010
1050-1099	111010
1100-1149	001011
1150-1199	011011
1200-1249	101011
1250-1299	111011
1300-1349	001100
1350-1399	011100

见下页

表 43.4-4 – 接上页

频率范围 (Mbps)	hsfreqrange[5:0]*
1400-1449	101100
1450-1500	111100

还需要通过测试代码接口配置 PLL 输入和输出频率。有关 PLL 参数的详细描述，请参阅 PLL 参数。

当 DSI_HOST_PHY_RSTZ 和 DSI_HOST_PHY_SHUTDOWNZ 设置为 1 时，MIPI BD D-PHY 离开关闭状态并开始初始化。

43.4.3.1.3 PLL 锁定和 AFE 初始化

在释放 DSI_HOST_PHY_RSTZ 和 DSI_HOST_PHY_SHUTDOWNZ 后，PHY 开始初始化序列以启用其正常操作。建议在释放 DSI_HOST_PHY_RSTZ 之前释放 DSI_HOST_PHY_SHUTDOWNZ。此外，确保 DSI_DPHY_CFG_CLK 和 DSI_DPHY_PLL_REFCLK（有关详细信息，请参见第 11 章中的表 11.2-5 复位和时钟）在此时可用且稳定。

之后，PLL 变为活动状态，PHY 等待 PLL 锁定。锁定状态由 DSI_HOST_PHY_LOCK 指示。

在 PLL 锁定后，开始 AFE 初始化。当 DSI_HOST_PHY_STOPSTATECLKLANE DSI_HOST_PHY_STOPSTATEnLANE 被置位时，所有初始化步骤完成，通道处于停止状态 (LP-11)。

43.4.3.1.4 控制模式 (Control)

AFE 初始化完成后，MIPI BD D-PHY 默认保持在控制模式，同时在线路上驱动停止状态 LP-11，直至有请求发出。任何请求都必须从该状态开始，并在该状态结束。请求发出后，通道可以进入高速数据传输模式、逃逸模式、超低功耗状态或转向状态。

43.4.3.1.5 高速数据传输

DSI 主机在收到来自 DSI 桥接器的数据后，会发送高速数据传输请求。在高速数据传输模式下，物理层 (PHY) 以突发方式传输图像数据。通道仅在这些突发传输期间以高速模式运行。

如图 43.4-10 所示，通道上的突发由三个部分组成：低功耗初始化序列、高速数据载荷和传输结束序列。

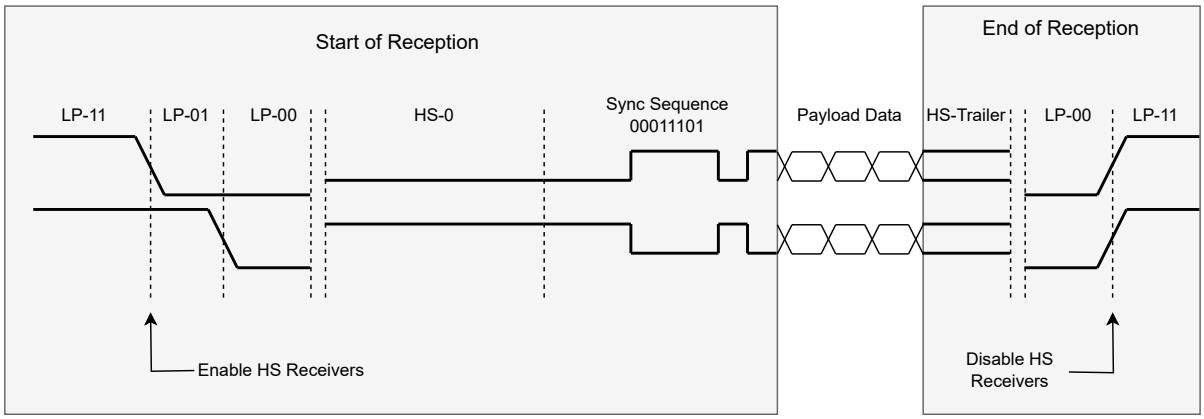


图 43.4-10. MIPI DSI HS 数据传输序列

43.4.3.1.6 逃逸模式 (Escape)

逃逸模式允许在数据通道上以低速进行异步通信。在逃逸模式下，通道保持在低功耗状态。ESP32-P4 中的 MIPI DSI 在逃逸模式下支持低功耗数据传输 (LPDT) 和超低功耗状态 (ULPS)。

如果 DSI_HOST_LP_CMD_EN 设置为 1，命令将在 LPDT 中发送。

当 DSI 空闲时，可以通过 DSI_HOST_PHY_ULPS_CTRL_REG 将 PHY 驱动到 ULPS 以节省功耗。通道状态可以通过 DSI_HOST_PHY_STATUS_REG 观察得知。

43.4.3.1.7 转向

PHY 可以通过转向交换传输方向。这用于接收来自 DSI 设备的命令响应。

43.4.3.2 PLL 参数

PLL 的 VCO 输出频率是输入参考时钟 DSI_DPHY_PLL_REFCLK 的频率以及倍增比率 M 和分频比率 N 的函数：

$$f_{VCO} = \frac{M}{N} \times f_{REFCLK}$$

该函数必须满足以下限制：

$$40\,\text{MHz} \geq \frac{f_{REFCLK}}{N} \geq 5\,\text{MHz}$$

此外，M 必须是偶数。M 和 N 的值可以通过 测试代码 0x17 和 测试代码 0x18 进行配置，配置值等于实际值减 1，如表 43.4-5 所示。之后，测试代码 0x19 的第 5 位和第 4 位应设置为 1，使 M 和 N 配置生效。

表 43.4-5. M 和 N

m[8:0] = M-1	M	n[6:0] = N-1	N
9'h1	2	7'h0	1
9'h3	4	7'h1	2
...	...	...	...

VCO 的 f_{VCO} 范围也可以通过 测试代码 0x10 进行选择。有效的 VCO 范围参见表 43.4-6。

表 43.4-6. VCO 的范围

vcorange[2:0]	VCO 频率 (MHz)
000	80-200
001	200-300
010	300-500
011	500-700
100	700-900
101	900-1100
110	1100-1300

见下页

表 43.4-6 – 接上页

vcorange[2:0]	VCO 频率 (MHz)
111	1300-1500

环路带宽应应根据输出频率通过测试代码接口进行配置，以确保 PLL 的正常运行。相关参数如下所示，有效值见表 43.4-7。

- VCO 电容 (vcocap[1:0])，通过 测试代码 0x10 配置。
- CP 电流 (icpctrl[3:0])，通过 测试代码 0x11 配置。
- LPF 特性 (lpfctrl[5:0])，通过 测试代码 0x12 配置。

表 43.4-7. PLL CP 和 LPF 控制位

VCO fmin (MHz)	VCO fmax (MHz)	vcorange[2:0]	icpctrl[3:0]	lpfctrl[5:0]	vcocap[1:0]
80	110	000	0100	010000	00
110	150		1000	001000	00
150	200		1100	001000	00
200	250	001	1100	001000	00
250	300		0010	001000	00
300	400	010	0101	001000	00
400	500		0101	000001	00
500	600	011	0110	010000	00
600	700		0110	000100	00
700	900	100	0110	000100	00
900	1100	101	0111	010000	00
1100	1300	110	0111	001000	00
1300	1500	111	0111	001000	00

说明：

vcorange, icpctrl, lpfctrl 在对应的 hsfreqrange 为 0 时会自动设置为默认值。

- 测试代码 0x10，第 7 位，清 0 以绕过 vcorange
- 测试代码 0x12，第 7 位，清 0 以绕过 icpctrl
- 测试代码 0x12，第 6 位，清 0 以绕过 lpfctrl

43.4.3.3 测试代码

MIPI BD D-PHY 提供了一组测试代码，用于配置 MIPI BD D-PHY 操作。配置可以使用 DSI_HOST_PHY_TST_CTRL0_REG 和 DSI_HOST_PHY_TST_CTRL1_REG 通过测试代码接口完成。有关测试代码接口的详细信息，请参阅 *42 MIPI CSI* 章 42.5.3.1 小节。

支持的所有测试代码包括：

43.4.3.3.1 正常运行

此模式表示 PHY 的正常运行以及测试接口的唤醒状态。在此模式下，测试接口处于非活动状态。

- **测试代码**：0x00
- **测试数据**：无

43.4.3.3.2 PLL 偏置电流选择器 / 滤波电容控制 / VCO 控制

控制 PLL 的偏置和 VCO 的传递函数。

- **测试代码**：0x10
- **测试数据**

表 43.4-8. 0x10 的测试数据

w-1'b0	w-1'b0	w-3'b000	w-2'b00	w-1'b1
跳过 VCO 范围	保留	VCO 范围控制	VCO 内部电容控制	参考偏置电流选择器

  - 位 7: 跳过 VCO 范围
    - 0 - VCO 范围使用对应 hsfreqrange 的默认值进行配置
    - 1 - VCO 范围使用位 5-3 进行配置
  - 位 6: 保留
  - 位 5-3: VCO 范围控制，详细信息请参见表 43.4-6
  - 位 2-1: VCO 内部电容控制
    - 00 - 默认电容
    - 01 - 低电容（1/4 默认值）
    - 10 - 高电容（2 倍默认值）
    - 11 - 不允许
  - 位 0: 参考偏置电流选择器
    - 0 - 内部偏置
    - 1 - 20uA 清洁的外部电流源

- **测试数据输出**

表 43.4-9. 0x10 的测试数据输出

r-1'b0	r-1'b0	r-3'b000	r-2'b00	r-1'b1
跳过 VCO 范围回环	1'b0	VCO 范围控制回环	VCO 内部电容控制回环	参考偏置电流选择器回环

43.4.3.3.3 用于初始化和超低功耗模式的 PLL CP 控制/PLL 锁定旁路

控制充电泵 (CP) 电流和 PLL 锁定旁路，用于初始化和 ULP 退出。

- **测试代码**：0x11

- **测试数据**

表 43.4-10. 0x11 的测试数据

w-4'b0000	w-4'b0010
PLL 锁定跳过	CP 电流

  - 位 7-4: PLL 锁定跳过
    * xxx1 - 在初始化期间跳过 PLL 锁定
    * x11x - 在退出 ULP 后跳过 PLL 锁定
  - 位 3-0: CP 电流 (uA)
    * 0000 - 1.5
    * 0001 - 3
    * 0010 - 4.5
    * 0011 - 6
    * 0100 - 2.5
    * 0101 - 5
    * 0110 - 7.5
    * 0111 - 10
    * 1000 - 3
    * 1001 - 6
    * 1010 - 9
    * 1011 - 12
    * 1100 - 4
    * 1101 - 8
    * 1110 - 12
    * 1111 - 16

- **测试数据输出**

表 43.4-11. 0x11 的测试数据输出

r-1'b0	r-1'b0
PLL 锁定跳过回环	CP 电流回环

43.4.3.3.4 PLL LPF 和 CP 控制

控制充电泵 (CP) 电流和低通滤波器 (LPF) 电阻。

- **测试代码**: 0x12

- **测试数据**

表 43.4-12. 0x12 的测试数据

w-1'b0	w-1'b0	w-6'b000010
跳过 CP 默认值	跳过 LPF 默认值	LPF 电阻

  - 位 7: 跳过 CP 默认值
    - 0 - CP 使用对应 hsfreqrange 的默认值进行配置
    - 1 - CP 使用测试代码 0x11 的位 3-0 进行配置
  - 位 6: 跳过 LPF 默认值
    - 0 - LPF 使用对应 hsfreqrange 的默认值进行配置
    - 1 - LPF 使用位 5-0 进行配置
  - 位 5-0: LPF 电阻 ($k\Omega$)
    - 000000 - 18
    - 000001 - 15.6
    - 000010 - 15
    - 000100 - 14.4
    - 001000 - 12.8
    - 010000 - 11.4
    - 100000 - 10.5
    - 其他 - 保留

- **测试数据输出**

表 43.4-13. 0x12 的测试数据输出

r-1'b0	r-1'b0	r-6'b000010
CP 跳过回环	LPF 跳过回环	LPF 电阻回环

43.4.3.3.5 PLL 输入分频比

控制 PLL 的输入分频器 N。$N = n + 1$，其中 N 是实际的分频比，n 是测试代码中的配置值。

要启用这一配置值，请确保 测试代码 0x19 的位 4 设置为 1。

- **测试代码**：0x17
- **测试数据**

表 43.4-14. 0x17 的测试数据

w-1'b0	w-7'b0000001
保留	输入分频器

- **测试数据输出**

表 43.4-15. 0x17 的测试数据输出

r-1'b0	r-7'b0000001
保留	输入分频器回环

43.4.3.3.6 PLL 循环分频比

控制 PLL 的反馈分频器 M。$M = m + 1$，其中 M 是实际的反馈比，m 是测试代码中的配置值。

要启用配置值，请确保 测试代码 0x19 的位 5 设置为 1。

- **测试代码**：0x18
- **测试数据**

表 43.4-16. 0x18 的测试数据

w-1'b0	w-2'b00	w-5'b00101
设置 0	保留	反馈分频器 [4:0]
w-1'b1	**w-3'b00**	**w-4'b0000**
设置 1	保留	反馈分频器 [8:5]

  - 位 7: 配置选择器
    * 0 - 设置 0 被配置
    * 1 - 设置 1 被配置
  - 位 6-5: 保留
  - 位 4-0: 反馈分频器

- **测试数据输出**
  TESTDIN[7] - 设置 0

表 43.4-17. 0x18 的测试数据输出

r-3'b000	r-5'b00101
3'b000	反馈分频器 [4:0] 回环

  TESTDIN[7] - 设置 1

表 43.4-18. 0x18 的测试数据输出

r-4'b0000	r-4'b0000
4'b0000	反馈分频器 [8:5] 回环

43.4.3.3.7 PLL 输入和循环分频比控制

控制 PLL 输入和循环分频比。

- **测试代码**：0x19

- **测试数据**

表 43.4-19. 0x19 的测试数据

w-2'b00	w-1'b0	w-1'b0	w-4'b0000
保留	跳过 PLL 循环分频器默认值	跳过 PLL 输入分频器默认值	保留

  - 位 7-6: 保留
  - 位 5: 跳过 PLL 循环分频器默认值
    * 0 - PLL 循环分频器使用对应 hsfreqrange 的默认值进行配置
    * 1 - PLL 循环分频器使用 测试代码 0x18 进行配置
  - 位 4: 跳过 PLL 输入分频器默认值
    * 0 - PLL 输入分频器使用对应 hsfreqrange 的默认值进行配置
    * 1 - PLL 输入分频器使用 测试代码 0x17 进行配置
  - 位 3-0: 保留

- **测试数据输出**

表 43.4-20. 0x19 的测试数据输出

r-2'b00	r-1'b0	r-1'b0	r-4'b0000
保留	跳过 PLL 循环分频器默认回环	跳过 PLL 输入分频器默认回环	保留

43.4.3.3.8 AFE/BIAS/Bandgap 模拟可编程性

控制 PHY 中的各种参数。

- **测试代码**：0x22
- **测试数据**：

表 43.4-21. 0x22 的测试数据

w-1'b0	w-7'b0000000	
设置 0	模拟可编程性 [6:0]	
w-1'b1	w-3'b00	w-4'b0000
设置 1	保留	模拟可编程性 [10:7]

  - 位 7：配置选择器
    * 0 - 配置设置 0（同时选择位 6:0 用于可观测性）
    * 1 - 配置设置 1（同时选择位 10:7 用于可观测性）
  - 位 6-0：模拟可编程性
    * 位 10：1.5 Gbps 支持。对于高于 1 Gbps 且不超过 1.5 Gbps 的速率，应启用此位
      · 0 - 禁用 1.5 Gbps 模拟电路支持
      · 1 - 启用 1.5 Gbps 模拟电路支持

* 位 9-7：调整带隙参考电压（典型角）
  - 000 - 97.07%
  - 001 - 98.05%
  - 010 - 99.02%
  - 011 - 100.00%
  - 100 - 93.17%
  - 101 - 94.15%
  - 110 - 95.12%
  - 111 - 96.10%
* 位 6：调整 LP RX 偏置电流
  - 0 - 100%
  - 1 - 50%
* 位 5-3：调整 LP TX 偏置电流
  - 000 - 76.92%
  - 001 - 83.33%
  - 010 - 90.91%
  - 011 - 100.00%
  - 100 - 111.10%
  - 101 - 125.00%
  - 110 - 142.80%
  - 111 - 166.70%
* 位 2-0：Biasextr：调整内部电阻
  - 000 - 87.1%
  - 001 - 91.5%
  - 010 - 95.9%
  - 011 - 100%
  - 100 - 105.94%
  - 101 - 111.88%
  - 110 - 118.8%
  - 111 - 127.7%

- **测试数据输出**：
  TESTDIN[7] - 设置 0

表 43.4-22. 0x22 的测试数据输出

r-1'b0	r-7'b0000000
1'b0	模拟可编程性 [6:0] 回环

TESTDIN[7] - 设置 1

表 43.4-23. 0x22 的测试数据输出

r-4'b0000	r-4'b0000
4'b0000	模拟可编程性 [10:7] 回环

43.4.3.3.9 HS RX 控制通道 0

控制 hsfreqrange。

- **测试代码**：0x44
- **测试数据**

表 43.4-24. 0x44 的测试数据

w-1'b0	w-6'b000000	w-1'b0
设置 2	HS 操作频率范围选择 (hsfreqrange)	跳过 HS-RX 稳定滤波器

  - 位 7: 配置选择器
    - 0 - 设置 2 被配置
    - 1 - 无效
  - 位 6-1: HS 频率范围选择。详细值请参见表 43.4-4。
  - 位 0: 跳过 HS 稳定滤波器

- **测试数据输出**

表 43.4-25. 0x44 的测试数据输出

r-1'b0	r-6'b000000	r-1'b0
1'b0	HS 运行的频率范围选择回环 (hsfreqrange)	跳过 HS-RX 稳定滤波器回环

43.4.4 时钟和显示分辨率

DSI 的时钟架构如图 43.4-11 所示。所有时钟为：

- **PCLK**：APB 总线时钟
- **ACLK**：AXI 总线时钟
- **DSI_DPICLK**：DPI 接口时钟
- **DSI_DPHY_CFG_CLK**：DPHY 配置时钟

- **DSI_DPHY_PLL_REFCLK**：DPHY PLL 输入参考时钟
- **TX_HS_CLK**：DPHY PLL 输出时钟
- **TX_BYTE_CLK**：高速 (HS) 传输字节时钟，TX_HS_CLK 除以 8。
- **TX_ESC_CLK**：低功耗 (LP) 逃逸模式传输时钟，从 TX_BYTE_CLK 分出。分频比可以通过 DSI_HOST_TO_CLK_DIVISION 配置，允许的最大频率为 20 MHz。

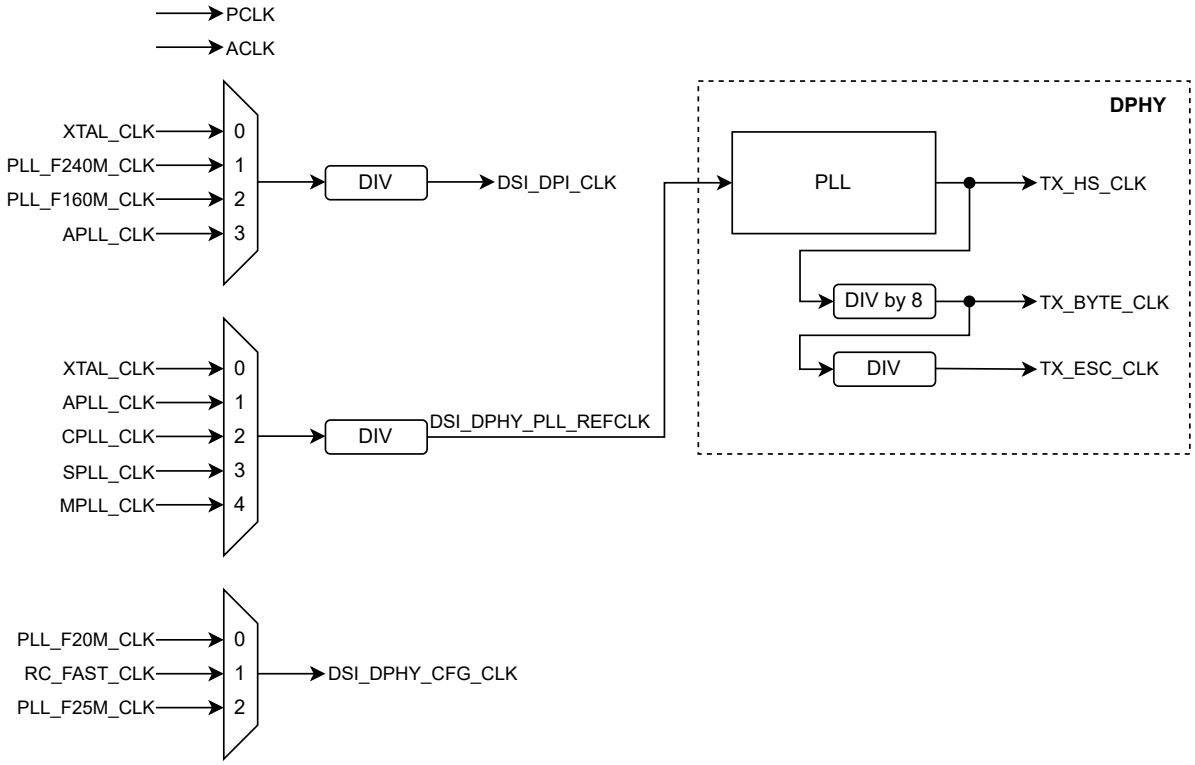


图 43.4-11. MIPI DSI 时钟架构

DSI 支持连续和非连续时钟通道。

- 对于连续时钟通道，需要把 DSI_HOST_AUTO_CLKLANE_CTRL 配置成 1，把 DSI_HOST_PHY_TXREQUESTCLKHS 配置成 1
- 对于非连续时钟通道，需要把 DSI_HOST_AUTO_CLKLANE_CTRL 配置成 0，把 DSI_HOST_PHY_TXREQUESTCLKHS 配置成 1

在视频模式传输中，视频时序参数如图 43.4-12 所示。

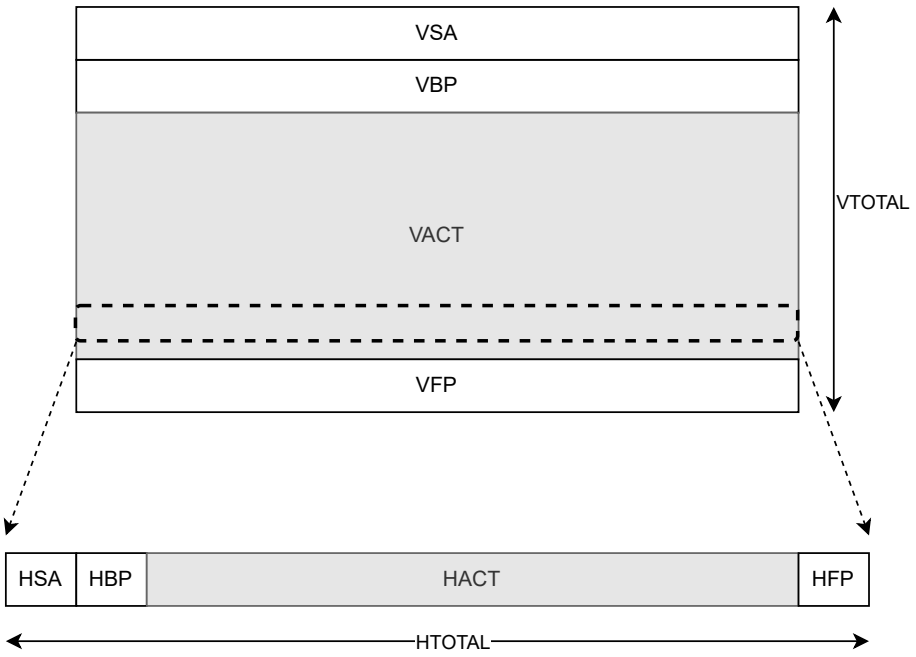


图 43.4-12. 视频时序

对于 DPI，视频时序和分辨率可以通过以下 DSI 桥接寄存器字段进行配置。之后，向 DSI_BRG_DPI_CONFIG_UPDATE 写入 1 使配置生效。

- DSI_BRG_HTOTAL：以 DSI_DPICLK 为单位的总水平长度 (HTOTAL)
- DSI_BRG_HDISP：以 DSI_DPICLK 为单位的有效水平长度 (HACT)
- DSI_BRG_HBANK：以 DSI_DPICLK 为单位的水平后沿长度 (HBP)
- DSI_BRG_HSYNC：以 DSI_DPICLK 为单位的水平同步长度 (HSA)
- DSI_BRG_VTOTAL：以行数为单位的总垂直长度 (VTOTAL)
- DSI_BRG_VDISP：以行数为单位的有效垂直长度 (VACT)
- DSI_BRG_VBANK：以行数为单位的垂直后沿长度 (VBP)
- DSI_BRG_VSYNC：以行数为单位的垂直同步长度 (VSA)

DSI 也需要类似的视频时序和分辨率配置，相关寄存器字段如下：

- DSI_HOST_VID_HSA_TIME：以 TX_BYTE_CLK 为单位的水平同步长度 (HSA)
- DSI_HOST_VID_HBP_TIME：以 TX_BYTE_CLK 为单位的水平后沿长度 (HBP)
- DSI_HOST_VID_HLINE_TIME：以 TX_BYTE_CLK 为单位的总水平长度 (HTOTAL)
- DSI_HOST_VSA_LINES：以行数为单位的垂直同步长度 (VSA)
- DSI_HOST_VBP_LINES：以行数为单位的垂直后沿长度 (VBP)
- DSI_HOST_VFP_LINES：以行数为单位的垂直前沿长度 (VFP)
- DSI_HOST_VACTIVE_LINES：以行数为单位的有效垂直长度 (VACT)

DPI 和 DSI 的视频时序配置应匹配，以避免行缓冲区溢出。DPI 和 DSI 中每行的实际时序应相同。

43.5 中断

ESP32-P4 的 MIPI DSI 可以生成以下中断信号，这些信号将发送到 中断矩阵。

- dsi_bridge_interrupt
- dsi_interrupt

以下中断源可以生成 dsi_bridge_interrupt 中断信号：

表 43.5-1. DSI 桥中断

中断源	详细信息
DSI_BRG_UNDERRUN_INT	DSI 桥缓冲下溢时触发

以下中断源可以生成 dsi_interrupt 中断信号：

表 43.5-2. DSI 主机中断

名称	原因	推荐处理方法
DSI_HOST_ACK_WITH_ERR_0	确认错误报告中的 SoT 错误	如果数据包很关键，请重新传输数据包
DSI_HOST_ACK_WITH_ERR_1	确认错误报告中的 SoT 同步错误	请重新传输数据包
DSI_HOST_ACK_WITH_ERR_2	确认错误报告中的 EoT 同步错误	请重新传输数据包
DSI_HOST_ACK_WITH_ERR_3	确认错误报告中的转义模式入口命令错误，设备未识别转义模式入口代码	检查设备能力
DSI_HOST_ACK_WITH_ERR_4	确认错误报告中的低功耗传输同步错误	请重新传输数据包
DSI_HOST_ACK_WITH_ERR_5	确认错误报告中的外设超时错误	检查主机超时计数器配置是否与设备规格一致；主机超时计数器配置应比设备更长。此错误会自动恢复，但不能保证传输或接收完成
DSI_HOST_ACK_WITH_ERR_6	确认错误报告中的虚假控制错误	重置 DSI 主机和 DPHY
DSI_HOST_ACK_WITH_ERR_7	确认错误报告中的保留项（特定于设备）	检查设备能力和连接性并重新传输
DSI_HOST_ACK_WITH_ERR_8	确认错误报告中的 ECC 错误，单比特（已检测并纠正）	无需采取行动。如果此错误频繁发生，请分析信号完整性或链路的噪声条件
DSI_HOST_ACK_WITH_ERR_9	确认错误报告中的 ECC 错误，多比特（已检测未纠正）	如果数据包很关键，请重新传输数据包。如果此错误频繁发生，请分析信号完整性或链路的噪声条件

见下页...

表 43.5-2 – ... 接上页

名称	原因	推荐处理方法
DSI_HOST_ACK_WITH_ERR_10	确认错误报告中的校验和错误（仅限长数据包）	如果数据包很关键，请重新传输数据包。如果此错误频繁发生，请检查连接
DSI_HOST_ACK_WITH_ERR_11	确认错误报告中的未识别 DSI 数据类型。某些数据包可能不受支持	检查设备能力
DSI_HOST_ACK_WITH_ERR_13	确认错误报告中的无效传输长度	请重新传输数据包。如果此错误频繁发生，请检查连接
DSI_HOST_ACK_WITH_ERR_14	确认错误报告中的保留项（特定于设备）	检查设备规格以找出可能的原因和措施
DSI_HOST_ACK_WITH_ERR_15	确认错误报告中的 DSI 协议违规	检查显示规格。发生此错误时，设备应有另一个读回命令，报告附加错误信息
DSI_HOST_DPHY_ERRORS_0	来自通道 0 的 ErrEsc 转义入口错误，未识别转义入口代码	重复接收
DSI_HOST_DPHY_ERRORS_1	来自通道 0 的 ErrSyncEsc 低功耗数据传输同步错误	重置 DSI 主机并重复传输
DSI_HOST_DPHY_ERRORS_2	来自通道 0 的控制错误 ErrControl	重置 DSI 主机和 DPHY
DSI_HOST_DPHY_ERRORS_3	来自通道 0 的 LP0 争用错误 ErrContentionLP0	重置 DSI 主机并重复传输
DSI_HOST_DPHY_ERRORS_4	来自通道 0 的 LP1 争用错误 ErrContentionLP1	重置 DSI 主机并重复传输
DSI_HOST_TO_HS_TX	高速传输超时错误	检查超时配置。主机超时计数器配置应比设备更长
DSI_HOST_TO_LP_RX	低功耗接收超时错误	检查超时配置。主机超时计数器配置应比设备更长
DSI_HOST_ECC_SINGLE_ERR	在接收的数据包中检测到并纠正了 ECC 单个错误	无需采取任何行动
DSI_HOST_ECC_MILTI_ERR	在接收的数据包中检测到 ECC 多个错误	重复传输
DSI_HOST_CRC_ERR	在接收的数据包有效载荷中检测到 CRC 错误	重置 DSI 主机并重复传输
DSI_HOST_PKT_SIZE_ERR	在数据包接收期间检测到数据包大小错误	重置 DSI 主机并重复传输
DSI_HOST_EOPT_ERR	在外设传输结束时未接收到 EoTp 数据包	此错误对接收数据包的数据完整性没有关键影响。检查设备是否支持 EoTp 数据包
DSI_HOST_DPI_PLD_WR_ERR	在 DPI 像素行存储期间，有效载荷 FIFO 变满且存储的数据已损坏	检查视频模式配置，重置 DSI 主机并重新启动视频传输

见下页...

表 43.5-2 – ... 接上页

名称	原因	推荐处理方法
DSI_HOST_GEN_CMD_WR_ERR	系统尝试通过通用接口写入命令，但 FIFO 已满。因此，命令未写入	重置 DSI 主机并重复传输
DSI_HOST_GEN_PLD_WR_ERR	系统尝试通过通用接口写入有效载荷数据，但 FIFO 已满。因此，有效载荷未写入	重置 DSI 主机并重复传输
DSI_HOST_GEN_PLD_SEND_ERR	在通用接口数据包构建期间，有效载荷 FIFO 变空且发送了损坏的数据	重置 DSI 主机并重复传输
DSI_HOST_GEN_PLD_RD_ERR	在 DCS 读取数据期间，有效载荷 FIFO 变空且发送到接口的数据已损坏	重置 DSI 主机并重复传输
DSI_HOST_GEN_PLD_RECEV_ERR	在通用接口数据包读回期间，有效载荷 FIFO 变满且接收到的数据已损坏	重置 DSI 主机并重复传输
DSI_HOST_DPI_BUFF_PLD_UNDER	在读取有效载荷以构建视频模式的 DSI 数据包时发生了下溢	重置 DSI 主机并重复传输

43.6 配置过程

43.6.1 DSI 桥接配置

- 配置 DSI Bridge 时钟并释放复位。
- 配置输入和输出颜色模式。详细信息请参阅 DSI Bridge。
- 配置视频时序。有关详细信息，请参阅 时钟和显示分辨率。
- 将 0 写入 DSI_BRG_DSI_DMA_FLOW_CONTROLLER 设置 VDMA 为流控制器。
- 配置 DSI_BRG_DMA_BURST_LEN DMA 突发长度以匹配 VDMA 的配置。
- 配置 DSI_BRG_DSI_RAW_BUF_ALMOST_EMPTY_THRD 为 $1024 - DSI_BRG_DMA_BURST_LEN$ 获得更好的性能。
- 将 1 写入 DSI_BRG_DPI_EN 以启用 DPI。
- 将 1 写入 DSI_BRG_DSI_EN 以启用 DSI Bridge。
- 配置 VDMA 并将 VDMA 设置为流控制器。详细信息请参阅 7。

43.6.2 D-PHY 配置

- 将 DSI_HOST_PHY_SHUTDOWNZ 和 DSI_HOST_PHY_RSTZ 设置为 0，将 DSI_HOST_PHY_TESTCLR 设置为 1。
- 应用适当的 DSI_DPHY_CFG_CLK 和 DSI_DPHY_PLL_REFCLK。

- 通过 测试代码 0x44 配置 hsfreqrange。详细值请参见表 43.4-4。
- 配置 PLL 参数。详细信息参见 PLL 参数。
- 配置 DSI_HOST_TO_CLK_DIVISION 以应用适当的 TX_ESC_CLK 分频比。
- 将 DSI_HOST_PHY_SHUTDOWNZ 和 DSI_HOST_PHY_RSTZ 设置为 1。
- 等待 DSI_HOST_PHY_STOPSTATE0LANE 和 DSI_HOST_PHY_STOPSTATE1LANE 信号置位。

43.6.3 MIPI DSI 通用配置

- 释放全局复位。
- 配置 DSI_HOST_N_LANES 以定义要使用的通道数量。
- 配置视频时序设置。有关详细信息，请参阅 时钟和显示分辨率。
- 通过 D-PHY 配置 初始化 DPHY。
- 将 DSI_HOST_SHUTDOWNZ 写入 1 以释放 DSI 主机的关机状态。

43.6.4 系统内存数据输出流配置

- 参考 DSI Bridge 配置 中的方法配置 DSI Bridge。
- 参考 MIPI DSI 通用配置 配置 DSI Host。

43.6.5 视频模式输出流配置

- 通过寄存器字段 DSI_HOST_VPG_ORIENTATION、DSI_HOST_VPG_MODE 和 DSI_HOST_VPG_EN 配置模式。有关详细信息，请参阅 视频图案生成器。
- 通过 MIPI DSI 通用配置 配置 DSI 主机。

43.6.6 使用 DCS 命令操作显示寄存器

- 写入显示
  - 将数据包有效载荷写入 DSI_HOST_GEN_PLD_DATA_REG。
  - 将数据包头写入 DSI_HOST_GEN_HDR_REG。
- 从显示中读取
  - 将数据包头写入 DSI_HOST_GEN_HDR_REG。
  - 从 DSI_HOST_GEN_PLD_DATA_REG 读取响应数据。

43.7 寄存器列表

43.7.1 MIPI DSI 桥接寄存器列表

本节中的地址为相对于 MIPI DSI BRIDGE 的基地址，该地址在第 9 章的表 9.3-2 *系统和存储器* 中提供。

Access 列中的缩写在 *寄存器的访问类型* 部分进行了说明。

名称	描述	地址	访问
配置寄存器			
DSI_BRG_CLK_EN_REG	DSI 桥接器时钟使能寄存器	0x0000	R/W
DSI_BRG_EN_REG	DSI 桥接器使能寄存器	0x0004	R/W
DSI_BRG_DMA_REQ_CFG_REG	DSI 桥接器 DMA 突发长度寄存器	0x0008	R/W
DSI_BRG_RAW_NUM_CFG_REG	DSI 桥接器原始像素数量控制寄存器	0x000C	varies
DSI_BRG_RAW_BUF_CREDIT_CTL_REG	DSI 桥接器原始缓冲区信用寄存器	0x0010	R/W
DSI_BRG_PIXEL_TYPE_REG	DSI 桥接器 DPI 像素类型控制寄存器	0x0018	R/W
DSI_BRG_DMA_BLOCK_INTERVAL_REG	DSI 桥接器 DMA 块间隔控制寄存器	0x001C	R/W
DSI_BRG_DMA_REQ_INTERVAL_REG	DSI 桥接器 DMA 请求间隔控制寄存器	0x0020	R/W
DSI_BRG_DPI_LCD_CTL_REG	DSI 桥接器 DPI 信号控制寄存器	0x0024	R/W
DSI_BRG_DPI_RSV_DPI_DATA_REG	DSI 桥接器 DPI 保留数据寄存器	0x0028	R/W
DSI_BRG_DPI_V_CFG0_REG	DSI 桥接器 DPI 垂直时序配置寄存器 0	0x0030	R/W
DSI_BRG_DPI_V_CFG1_REG	DSI 桥接器 DPI 垂直时序配置寄存器 1	0x0034	R/W
DSI_BRG_DPI_H_CFG0_REG	DSI 桥接器 DPI 水平时序配置寄存器 0	0x0038	R/W
DSI_BRG_DPI_H_CFG1_REG	DSI 桥接器 DPI 水平时序配置寄存器 1	0x003C	R/W
DSI_BRG_DPI_MISC_CONFIG_REG	DSI 桥接器 DPI 杂项配置寄存器	0x0040	R/W
DSI_BRG_DPI_CONFIG_UPDATE_REG	DSI 桥接器 DPI 配置更新寄存器	0x0044	WT
DSI_BRG_HOST_TRIGGER_REV_REG	DSI 桥接器主机触发反向控制寄存器	0x0064	R/W
DSI_BRG_BLK_RAW_NUM_CFG_REG	DSI 桥接器块原始像素数量控制寄存器	0x0068	varies
DSI_BRG_DMA_FRAME_INTERVAL_REG	DSI 桥接器 DMA 帧间隔控制寄存器	0x006C	R/W
DSI_BRG_MEM_AUX_CTRL_REG	DSI 桥接器存储器辅助控制寄存器	0x0070	R/W
DSI_BRG_RDN_ECO_LOW_REG	DSI 桥接器 RDN ECO 低寄存器	0x0078	R/W
DSI_BRG_RDN_ECO_HIGH_REG	DSI 桥接器 RDN ECO 高寄存器	0x007C	R/W
DSI_BRG_HOST_CTRL_REG	DSI 桥接器主机控制寄存器	0x0080	R/W
DSI_BRG_MEM_CLK_CTRL_REG	DSI 桥接器存储器时钟强制开启控制寄存器	0x0084	R/W
DSI_BRG_DMA_FLOW_CTRL_REG	DSI 桥接器 DMA 流控制器寄存器	0x0088	R/W
DSI_BRG_RAW_BUF_ALMOST_EMPTY_THRD_REG	DSI 桥接器原始缓冲区近空阈值寄存器	0x008C	R/W
DSI_BRG_YUV_CFG_REG	DSI 桥接器 YUV 格式配置寄存器	0x0090	R/W
DSI_BRG_PHY_LP_LOOPBACK_CTRL_REG	DSI PHY LP 回环测试控制寄存器	0x0094	varies
DSI_BRG_PHY_HS_LOOPBACK_CTRL_REG	DSI PHY HS 回环测试控制寄存器	0x0098	varies
DSI_BRG_PHY_LOOPBACK_CNT_REG	DSI PHY 回环测试计数器寄存器	0x009C	R/W
状态寄存器			
DSI_BRG_FIFO_FLOW_STATUS_REG	DSI 桥接器 FIFO 流状态寄存器	0x0014	RO
DSI_BRG_HOST_BIST_CTL_REG	DSI 桥接器主机 BIST 控制寄存器	0x0060	varies

名称	描述	地址	访问
DSI_BRG_RDN_ECO_CS_REG	DSI 桥接器 RDN ECO 片选寄存器	0x0074	varies
中断寄存器			
DSI_BRG_INT_ENA_REG	DSI 桥接器中断使能寄存器	0x0050	R/W
DSI_BRG_INT_CLR_REG	DSI 桥接器中断清除寄存器	0x0054	WT
DSI_BRG_INT_RAW_REG	DSI 桥接器原始中断状态寄存器	0x0058	R/WTC/SS
DSI_BRG_INT_ST_REG	DSI 桥接器屏蔽中断状态寄存器	0x005C	RO
版本寄存器			
DSI_BRG_VER_DATE_REG	DSI 桥接器版本日期寄存器	0x0100	R/W

43.7.2 MIPI DSI 主机寄存器列表

本节中的地址为相对于 MIPI DSI HOST 基地址，该地址在第 9 章的表 9.3-2 中提供 *系统和存储器*。

列 **访问**中给出的缩写在 *寄存器的访问类型* 部分中进行了说明。

名称	描述	地址	访问
版本寄存器			
DSI_HOST_VERSION_REG	版本寄存器	0x0000	RO
配置寄存器			
DSI_HOST_PWR_UP_REG	DSI Host 上电控制寄存器	0x0004	R/W
DSI_HOST_CLKMGR_CFG_REG	内部分频控制寄存器	0x0008	R/W
DSI_HOST_DPI_VCID_REG	DPI 虚拟通道 ID 配置寄存器	0x000C	R/W
DSI_HOST_DPI_COLOR_CODING_REG	DPI 色彩编码配置寄存器	0x0010	R/W
DSI_HOST_DPI_CFG_POL_REG	DPI 信号极性配置寄存器	0x0014	R/W
DSI_HOST_DPI_LP_CMD_TIM_REG	低功耗命令时序配置寄存器	0x0018	R/W
DSI_HOST_PCKHDL_CFG_REG	数据包头配置寄存器，配置 EoTp、BTA、CRC、ECC 的使用方式	0x002C	R/W
DSI_HOST_GEN_VCID_REG	通用接口读响应虚拟通道 ID 配置寄存器	0x0030	R/W
DSI_HOST_MODE_CFG_REG	视频模式或命令模式配置寄存器	0x0034	R/W
DSI_HOST_VID_MODE_CFG_REG	视频模式配置寄存器	0x0038	R/W
DSI_HOST_VID_PKT_SIZE_REG	视频数据包大小配置寄存器	0x003C	R/W
DSI_HOST_VID_NUM_CHUNKS_REG	数据块数量配置寄存器	0x0040	R/W
DSI_HOST_VID_NULL_SIZE_REG	空包大小配置寄存器	0x0044	R/W
DSI_HOST_VID_HSA_TIME_REG	HSA 配置寄存器	0x0048	R/W
DSI_HOST_VID_HBP_TIME_REG	HBP 配置寄存器	0x004C	R/W
DSI_HOST_VID_HLINE_TIME_REG	HTOTAL 配置寄存器	0x0050	R/W
DSI_HOST_VID_VSA_LINES_REG	VSA 配置寄存器	0x0054	R/W
DSI_HOST_VID_VBP_LINES_REG	VBP 配置寄存器	0x0058	R/W
DSI_HOST_VID_VFP_LINES_REG	VFP 配置寄存器	0x005C	R/W
DSI_HOST_VID_VACTIVE_LINES_REG	VACT 配置寄存器	0x0060	R/W
DSI_HOST_CMD_MODE_CFG_REG	命令模式配置寄存器	0x0068	R/W
DSI_HOST_GEN_HDR_REG	通用接口数据包头寄存器	0x006C	R/W

名称	描述	地址	访问
DSI_HOST_GEN_PLD_DATA_REG	通用接口数据包有效载荷寄存器	0x0070	R/W
DSI_HOST_TO_CNT_CFG_REG	超时计数器配置寄存器	0x0078	R/W
DSI_HOST_HS_RD_TO_CNT_REG	HS 读超时计数器寄存器	0x007C	R/W
DSI_HOST_LP_RD_TO_CNT_REG	LP 读超时计数器寄存器	0x0080	R/W
DSI_HOST_HS_WR_TO_CNT_REG	HS 写超时计数器寄存器	0x0084	R/W
DSI_HOST_LP_WR_TO_CNT_REG	LP 写超时计数器寄存器	0x0088	R/W
DSI_HOST_BTA_TO_CNT_REG	总线翻转（BTA）超时计数器寄存器	0x008C	R/W
DSI_HOST_LPCLK_CTRL_REG	非连续时钟配置寄存器	0x0094	R/W
DSI_HOST_PHY_TMR_LPCLK_CFG_REG	CLK 通道 HS 与 LP 切换时序配置寄存器	0x0098	R/W
DSI_HOST_PHY_TMR_CFG_REG	数据通道 HS 与 LP 切换时序配置寄存器	0x009C	R/W
DSI_HOST_PHY_RSTZ_REG	DPHY 复位控制寄存器	0x00A0	R/W
DSI_HOST_PHY_IF_CFG_REG	DPHY 通道数量及停止状态配置寄存器	0x00A4	R/W
DSI_HOST_PHY_ULPS_CTRL_REG	DPHY ULPS 控制寄存器	0x00A8	R/W
DSI_HOST_PHY_TX_TRIGGERS_REG	DPHY 触发寄存器	0x00AC	R/W
DSI_HOST_PHY_TST_CTRL0_REG	DPHY 测试码时钟及清除控制寄存器	0x00B4	R/W
DSI_HOST_PHY_TST_CTRL1_REG	DPHY 测试码数据及使能控制寄存器	0x00B8	varies
DSI_HOST_PHY_CAL_REG	DPHY 偏斜校准控制寄存器	0x00CC	R/W
DSI_HOST_PHY_TMR_RD_CFG_REG	DPHY 读取计时器配置寄存器	0x00F4	R/W
DSI_HOST_VID_SHADOW_CTRL_REG	DPI 影子寄存器控制寄存器	0x0100	R/W
状态寄存器			
DSI_HOST_CMD_PKT_STATUS_REG	通用接口 FIFO 状态寄存器	0x0074	RO
DSI_HOST_PHY_STATUS_REG	DPHY 状态寄存器	0x00B0	RO
DSI_HOST_DPI_VCID_ACT_REG	实际 VCID 值寄存器	0x010C	RO
DSI_HOST_DPI_COLOR_CODING_ACT_REG	实际 DPI_COLOR_CODING 值寄存器	0x0110	RO
DSI_HOST_DPI_LP_CMD_TIM_ACT_REG	实际 DPI_LP_CMD_TIM 值寄存器	0x0118	RO
DSI_HOST_VID_MODE_CFG_ACT_REG	实际 VID_MODE_CFG 值寄存器	0x0138	RO
DSI_HOST_VID_PKT_SIZE_ACT_REG	实际 VID_PKT_SIZE 值寄存器	0x013C	RO
DSI_HOST_VID_NUM_CHUNKS_ACT_REG	实际 VID_NUM_CHUNKS 值寄存器	0x0140	RO
DSI_HOST_VID_NULL_SIZE_ACT_REG	实际 VID_NULL_SIZE 值寄存器	0x0144	RO
DSI_HOST_VID_HSA_TIME_ACT_REG	实际 HSA 值寄存器	0x0148	RO
DSI_HOST_VID_HBP_TIME_ACT_REG	实际 HBP 值寄存器	0x014C	RO
DSI_HOST_VID_HLINE_TIME_ACT_REG	实际 HLINE 值寄存器	0x0150	RO
DSI_HOST_VID_VSA_LINES_ACT_REG	实际 VSA 值寄存器	0x0154	RO
DSI_HOST_VID_VBP_LINES_ACT_REG	实际 VBP 值寄存器	0x0158	RO
DSI_HOST_VID_VFP_LINES_ACT_REG	实际 VFP 值寄存器	0x015C	RO
DSI_HOST_VID_VACTIVE_LINES_ACT_REG	实际 VACT 值寄存器	0x0160	RO
DSI_HOST_VID_PKT_STATUS_REG	DPI FIFO 状态寄存器	0x0168	RO
中断寄存器			
DSI_HOST_INT_ST0_REG	中断状态寄存器	0x00BC	RO
DSI_HOST_INT_ST1_REG	中断状态寄存器	0x00C0	RO
DSI_HOST_INT_MSK0_REG	中断屏蔽寄存器	0x00C4	R/W
DSI_HOST_INT_MSK1_REG	中断屏蔽寄存器	0x00C8	R/W

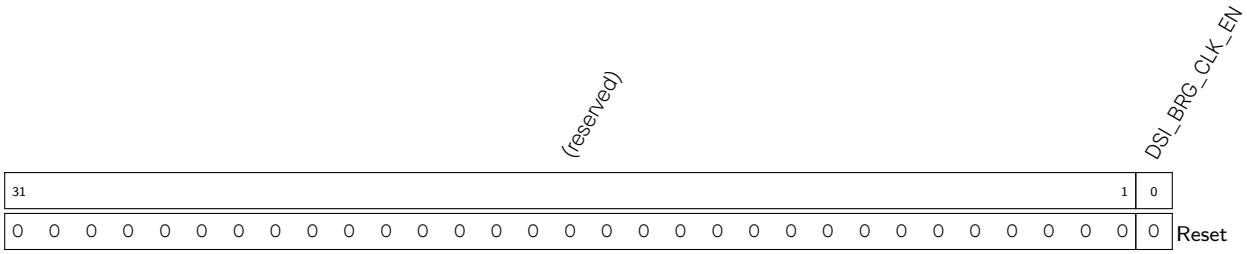
43.8 寄存器

43.8.1 MIPI DSI 桥接寄存器

本节中的地址相对于表 9.3-2 中提供的 MIPI DSI Bridge 基地址，该表位于第 9 章 *系统和存储器*。

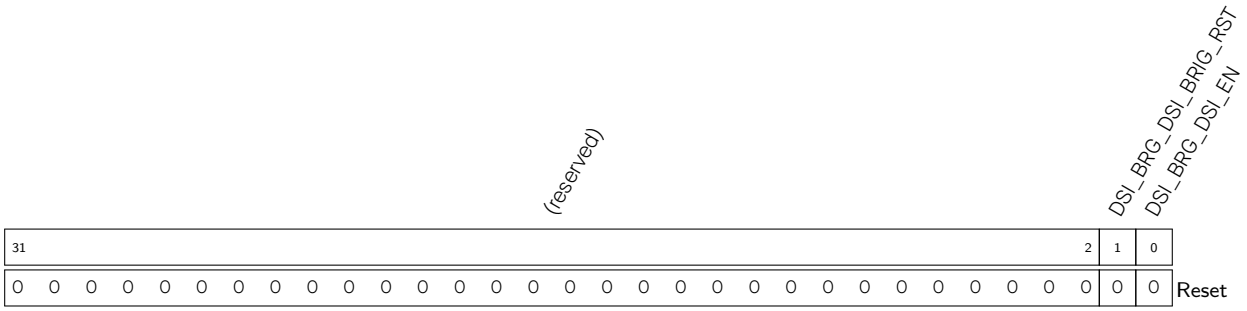
有关如何配置保留字段，请参阅 配置保留寄存器字段 部分。

Register 43.1. DSI_BRG_CLK_EN_REG (0x0000)



DSI_BRG_CLK_EN 配置 dsi_bridge 寄存器时钟门的 force_on。(R/W)

Register 43.2. DSI_BRG_EN_REG (0x0004)



DSI_BRG_DSI_EN 配置是否使能 dsi_bridge 模块。

0: 禁用

1: 使能

(R/W)

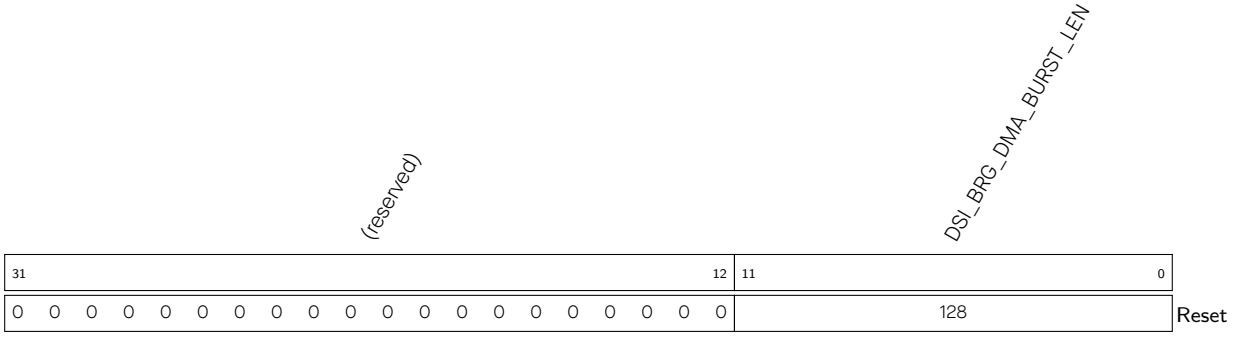
DSI_BRG_DSI_BRIG_RST 配置 dsi_bridge 软件复位。

0: 释放复位

1: 复位

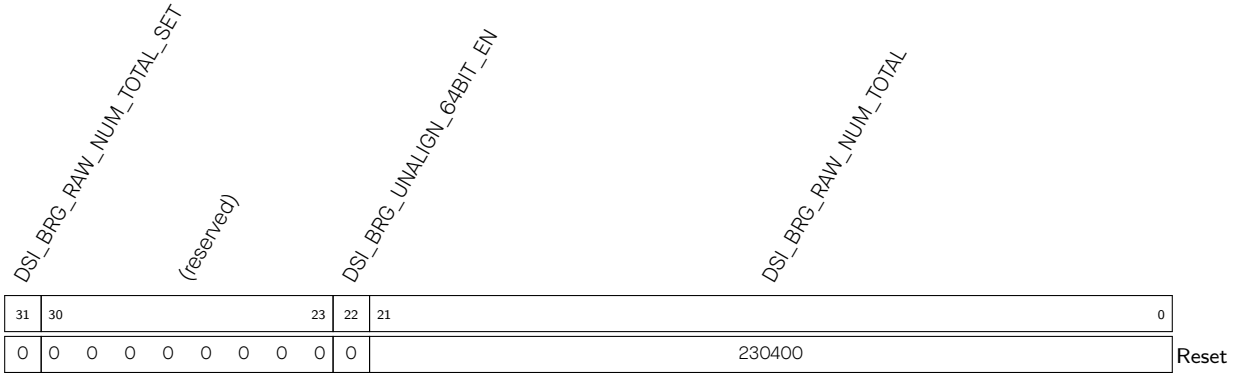
(R/W)

Register 43.3. DSI_BRG_DMA_REQ_CFG_REG (0x0008)



DSI_BRG_DMA_BURST_LEN 配置单次 DMA 突发传输中包含的 64 位数据单元数量。此值仅在 DSI 桥接器作为流控制器时有效。(R/W)

Register 43.4. DSI_BRG_RAW_NUM_CFG_REG (0x000C)



DSI_BRG_RAW_NUM_TOTAL 配置总像素位数除以 64 的值。(R/W)

DSI_BRG_UNALIGN_64BIT_EN 配置总像素位数是否为 64 位的倍数。

0: 对齐到 64 位

1: 不对齐到 64 位

(R/W)

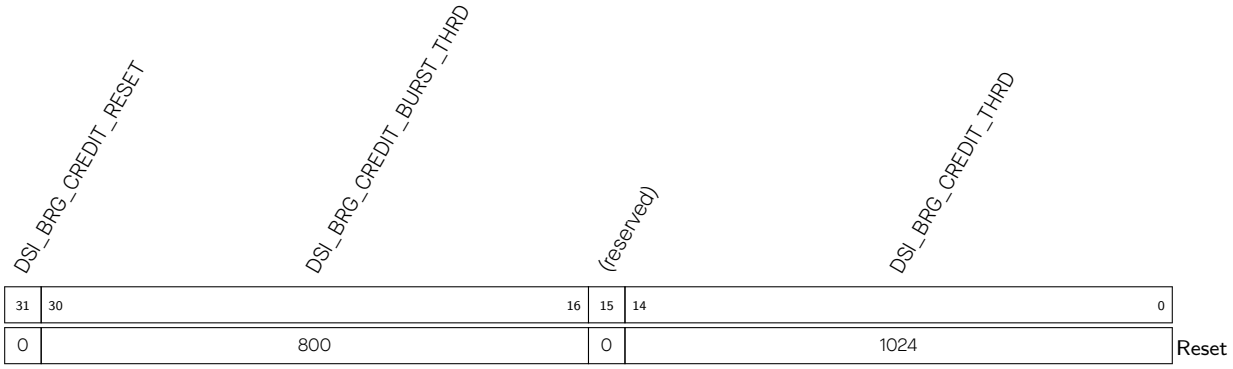
DSI_BRG_RAW_NUM_TOTAL_SET 配置是否使能将 reg_raw_num_total 重新加载到内部计数器。此值仅在 dsi_bridge 作为流控制器时有效。

0: 禁用

1: 使能

(WT)

Register 43.5. DSI_BRG_RAW_BUF_CREDIT_CTL_REG (0x0010)



DSI_BRG_CREDIT_THRD 配置 dsi_bridge FIFO 是否可以接收更多 64 位数据的阈值。此值仅在 dsi_bridge 作为流控制器时有效。(R/W)

DSI_BRG_CREDIT_BURST_THRD 配置 dsi_bridge FIFO 是否可以接收更多 DMA 突发的阈值。此值仅在 dsi_bridge 作为流控制器时有效。(R/W)

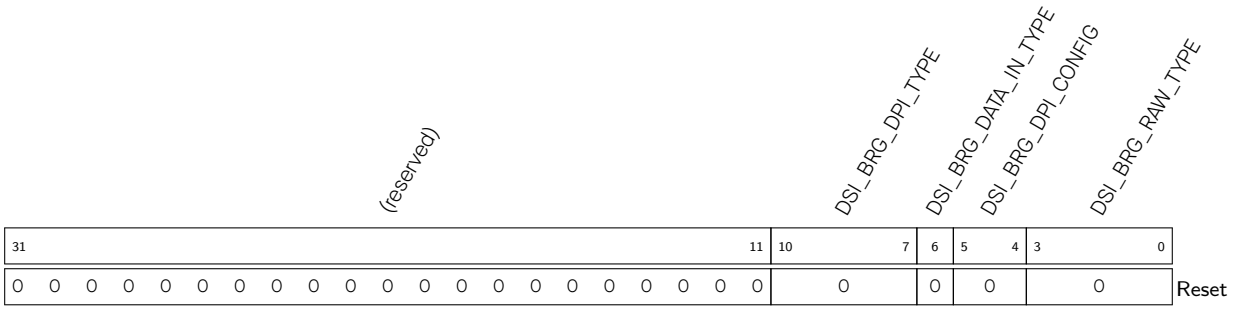
DSI_BRG_CREDIT_RESET 配置内部信用计数器的清除功能。此值仅在 dsi_bridge 作为流控制器时有效。

0: 无操作

1: 复位

(R/W)

Register 43.6. DSI_BRG_PIXEL_TYPE_REG (0x0018)



DSI_BRG_RAW_TYPE 配置输入像素类型。

0: RGB888

1: RGB666

2: RGB565

8: YUV444

9: YUV422

10: YUV420

12: GRAY

(R/W)

DSI_BRG_DPI_CONFIG 配置 DPI 接口的像素排列类型。详见表格 43.4-2 DPI 位排列。(R/W)

DSI_BRG_DATA_IN_TYPE 配置输入数据类型。

0: RGB

1: YUV

(R/W)

DSI_BRG_DPI_TYPE 配置 DPI 输出像素类型。

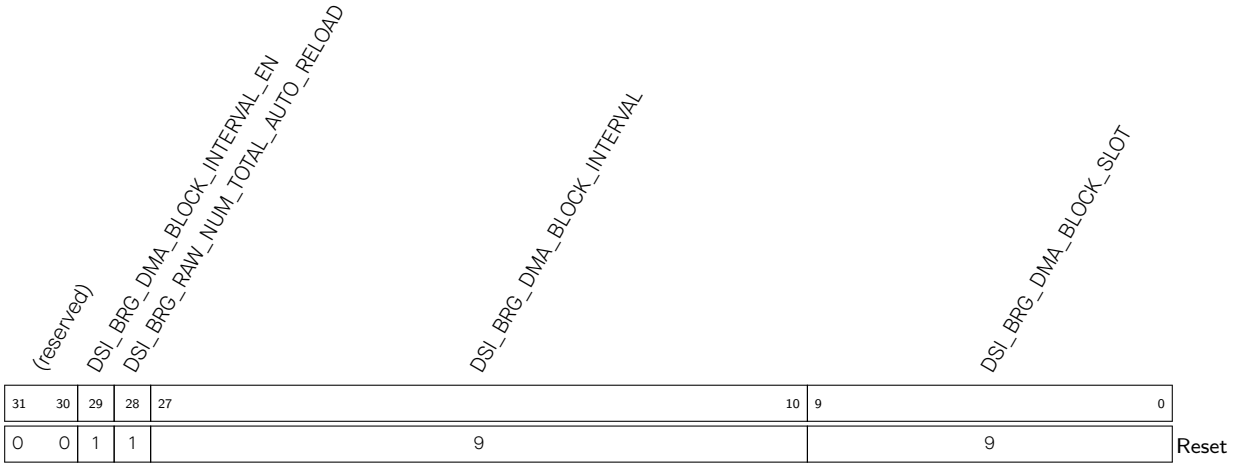
0: RGB888

1: RGB666

2: RGB565

(R/W)

Register 43.7. DSI_BRG_DMA_BLOCK_INTERVAL_REG (0x001C)



DSI_BRG_DMA_BLOCK_SLOT 配置最大 block_slot_cnt 值。(R/W)

DSI_BRG_DMA_BLOCK_INTERVAL 配置最大 block_interval_cnt 值。当 block_slot_cnt 满时，block_interval_cnt 加 1。(R/W)

DSI_BRG_RAW_NUM_TOTAL_AUTO_RELOAD 配置是否使能自动重新加载 reg_raw_num_total。

0: 禁用

1: 使能

(R/W)

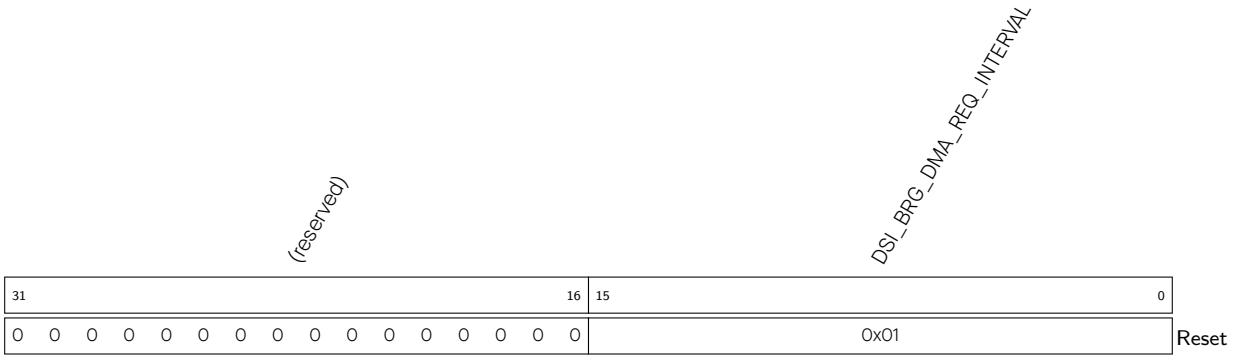
DSI_BRG_DMA_BLOCK_INTERVAL_EN 配置是否使能 DMA 块传输之间的间隔。

0: 禁用

1: 使能

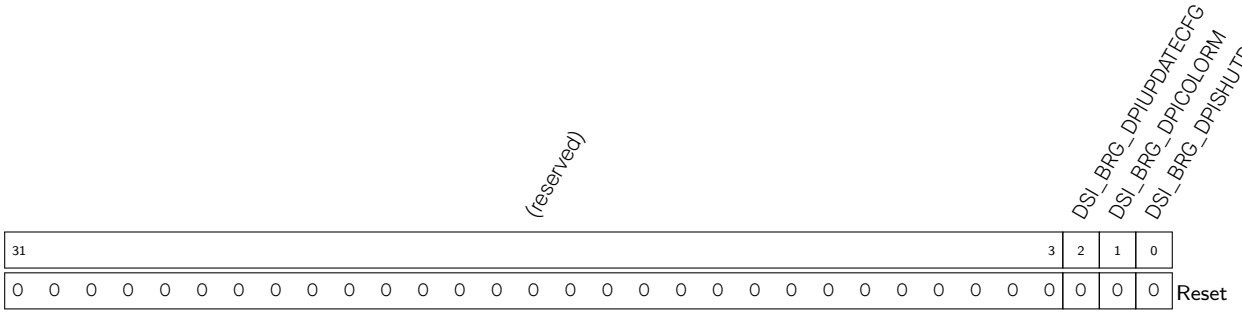
(R/W)

Register 43.8. DSI_BRG_DMA_REQ_INTERVAL_REG (0x0020)



DSI_BRG_DMA_REQ_INTERVAL 配置 DMA 请求事件之间的间隔。(R/W)

Register 43.9. DSI_BRG_DPI_LCD_CTL_REG (0x0024)

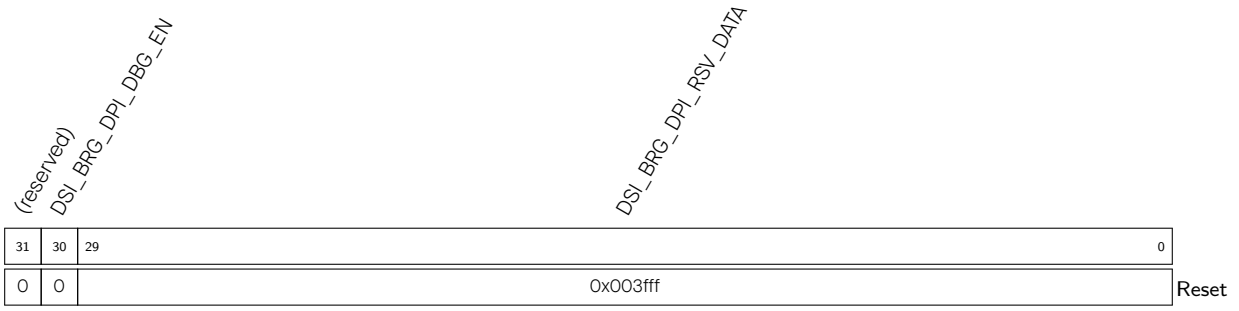


DSI_BRG_DPISHUTDN 配置 DPI 接口中的 dpishutdn 信号。(R/W)

DSI_BRG_DPICOLORM 配置 DPI 接口中的 dpicolorm 信号。(R/W)

DSI_BRG_DPIUPDATECFG 配置 DPI 接口中的 dpiupdatecfg 信号。(R/W)

Register 43.10. DSI_BRG_DPI_RSV_DPI_DATA_REG (0x0028)



DSI_BRG_DPI_RSV_DATA 配置当 dsi_bridge FIFO 下溢时发送到 dsi_host 的像素数据。(R/W)

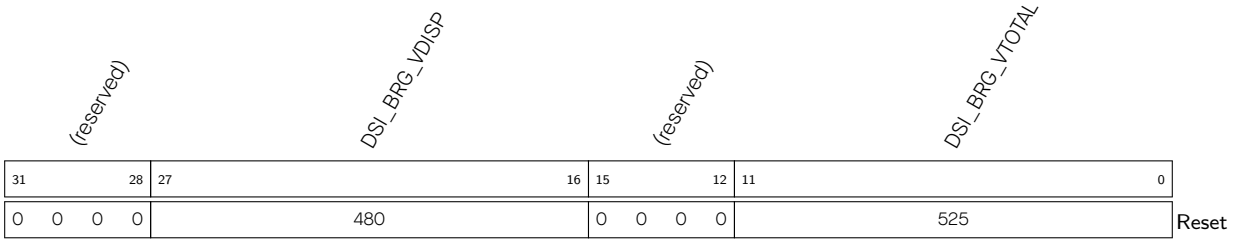
DSI_BRG_DPI_DBG_EN 配置数据调试功能使能。

0: 禁用

1: 使能

(R/W)

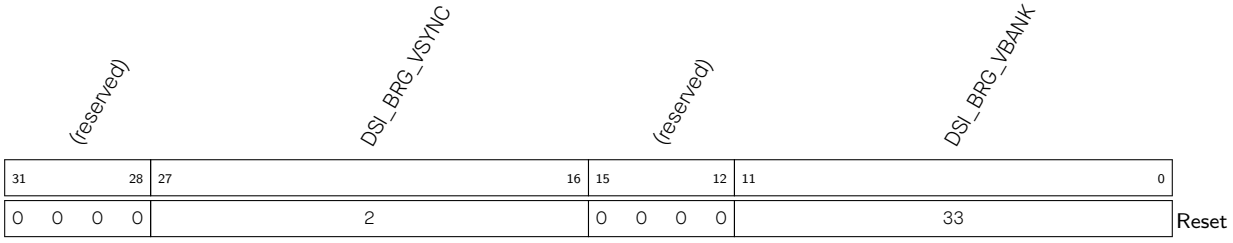
Register 43.11. DSI_BRG_DPI_V_CFG0_REG (0x0030)



DSI_BRG_VTOTAL 配置 DPI 输出的一帧总长度（按行）。必须满足：reg_vtotal > reg_vdisp+reg_vsync+reg_vbank。(R/W)

DSI_BRG_VDISP 配置 DPI 输出的有效行长度（按行）。(R/W)

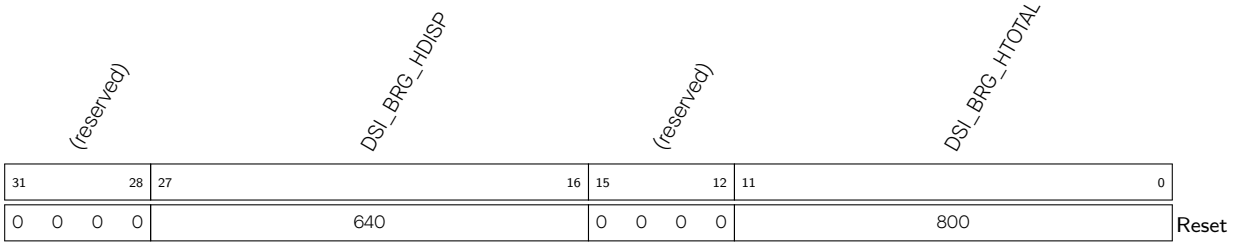
Register 43.12. DSI_BRG_DPI_V_CFG1_REG (0x0034)



DSI_BRG_VBANK 配置 DPI 输出中 vsync 和有效行之间的长度（按行）。(R/W)

DSI_BRG_VSYNC 配置 DPI 输出的 vsync 长度（按行）。(R/W)

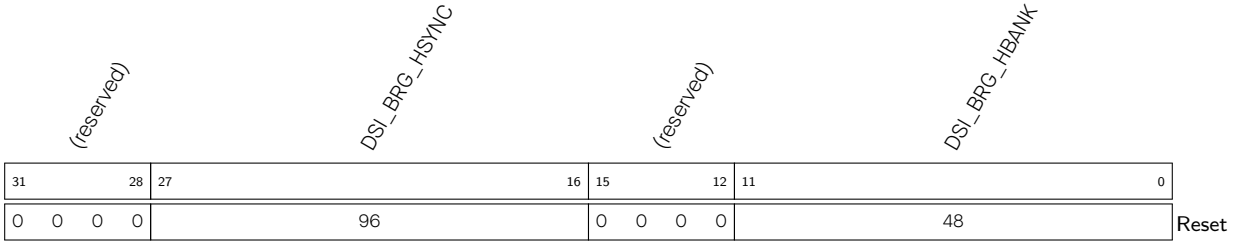
Register 43.13. DSI_BRG_DPI_H_CFG0_REG (0x0038)



DSI_BRG_HTOTAL 配置 DPI 输出的一行总长度（按像素数），必须满足：reg_htotal > reg_hdisp+reg_hsync+reg_hbank。(R/W)

DSI_BRG_HDISP 配置 DPI 输出的有效像素数据长度（按像素数）。(R/W)

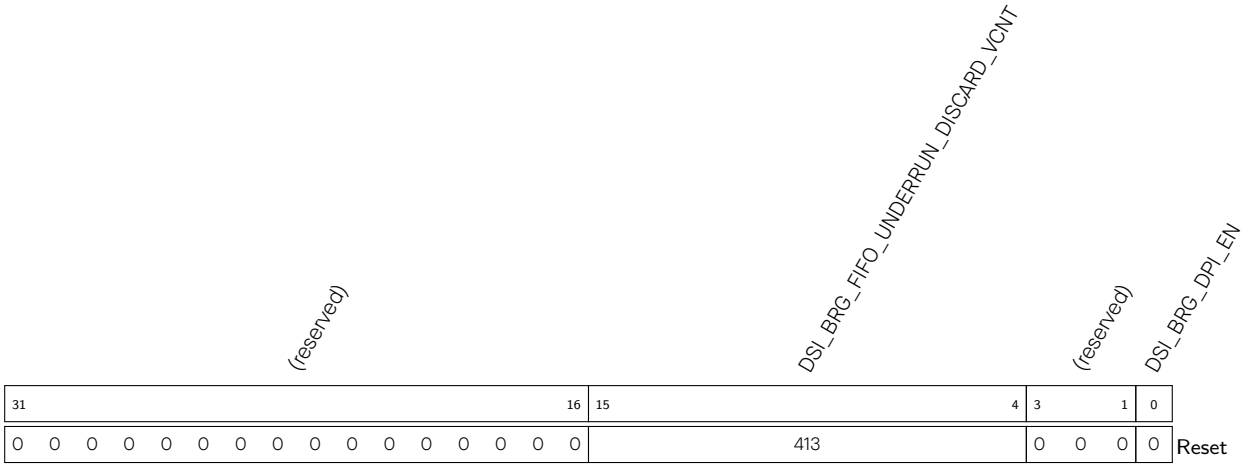
Register 43.14. DSI_BRG_DPI_H_CFG1_REG (0x003C)



DSI_BRG_HBANK 配置 DPI 输出中 hsync 和像素数据有效之间的长度（按像素数）。(R/W)

DSI_BRG_HSYNC 配置 DPI 输出的 hsync 长度（按像素数）。(R/W)

Register 43.15. DSI_BRG_DPI_MISC_CONFIG_REG (0x0040)



DSI_BRG_DPI_EN 配置是否使能 DPI 输出。

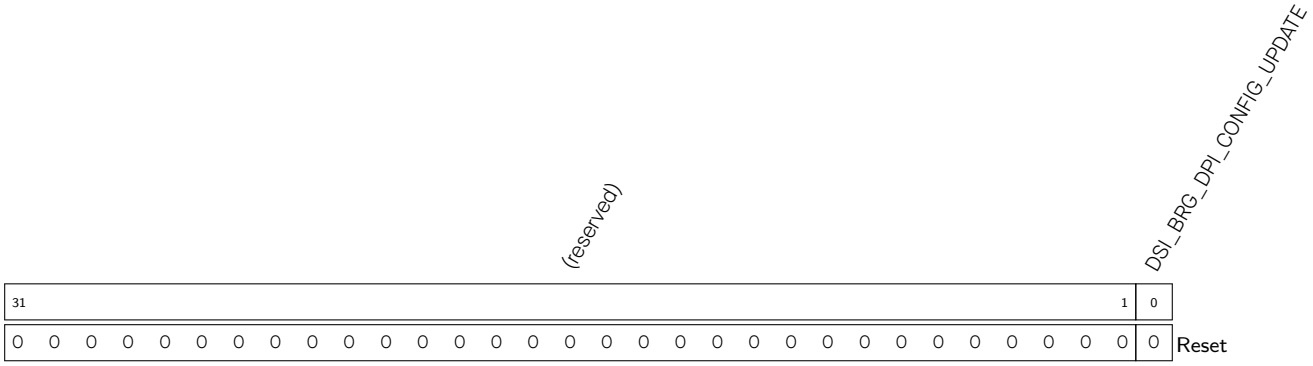
0: 禁用

1: 使能

(R/W)

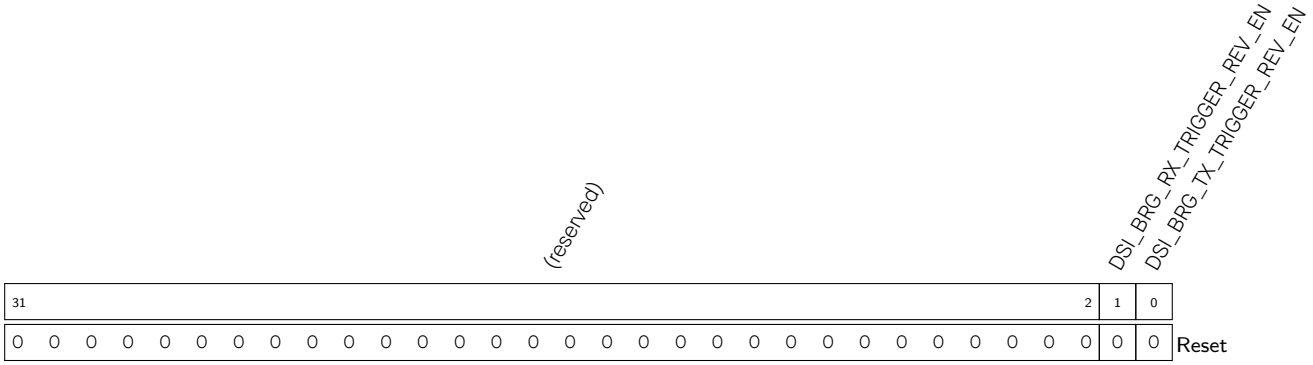
DSI_BRG_FIFO_UNDERRUN_DISCARD_VCNT 配置下溢中断掩码。当下溢发生且行计数小于此字段时，中断被抑制。(R/W)

Register 43.16. DSI_BRG_DPI_CONFIG_UPDATE_REG (0x0044)



DSI_BRG_DPI_CONFIG_UPDATE 写入 1 以更新 DPI 配置寄存器 DSI_BRG_DPI_*。(WT)

Register 43.17. DSI_BRG_HOST_TRIGGER_REV_REG (0x0064)



DSI_BRG_TX_TRIGGER_REV_EN 配置 DSI tx_trigger 反向。

0: 禁用

1: 使能

(R/W)

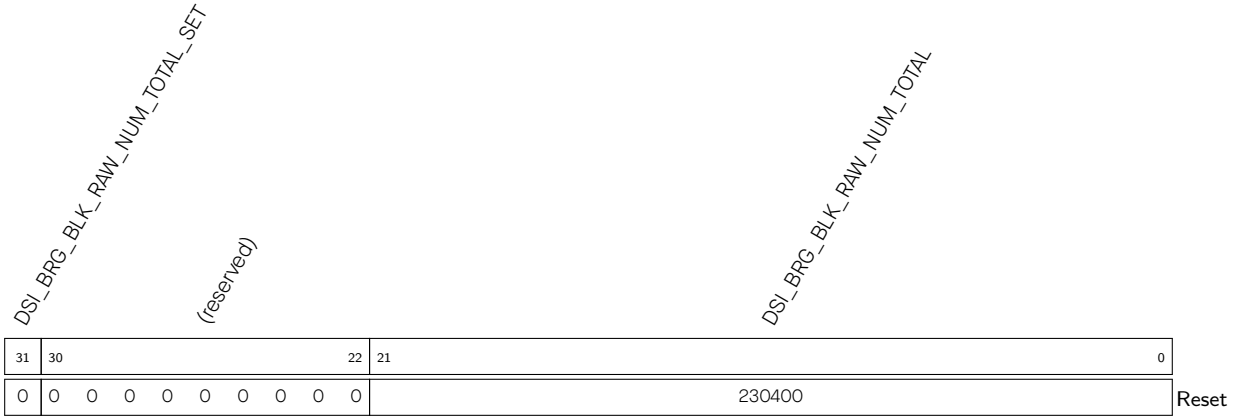
DSI_BRG_RX_TRIGGER_REV_EN 配置 DSI rx_trigger 反向。

0: 禁用

1: 使能

(R/W)

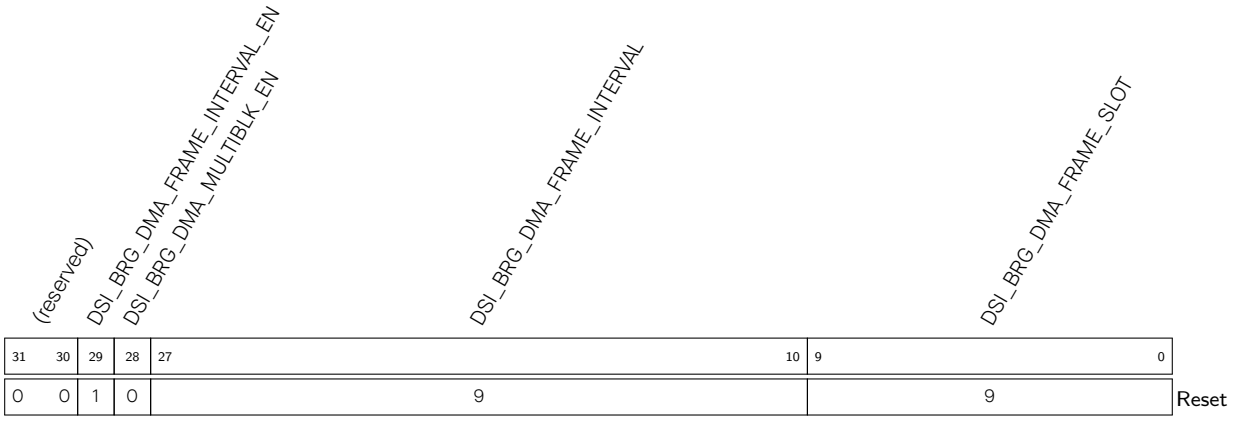
Register 43.18. DSI_BRG_BLK_RAW_NUM_CFG_REG (0x0068)



DSI_BRG_BLK_RAW_NUM_TOTAL 配置总块像素位数除以 64 的值。(R/W)

DSI_BRG_BLK_RAW_NUM_TOTAL_SET 写入 1 将 reg_blk_raw_num_total 重新加载到内部计数器。(WT)

Register 43.19. DSI_BRG_DMA_FRAME_INTERVAL_REG (0x006C)



DSI_BRG_DMA_FRAME_SLOT 配置最大 frame_slot_cnt 值。(R/W)

DSI_BRG_DMA_FRAME_INTERVAL 配置最大 frame_interval_cnt 值。当 frame_slot_cnt 满时，frame_interval_cnt 加 1。(R/W)

DSI_BRG_DMA_MULTIBLK_EN 配置是否使能多块传输。

0: 禁用

1: 使能

(R/W)

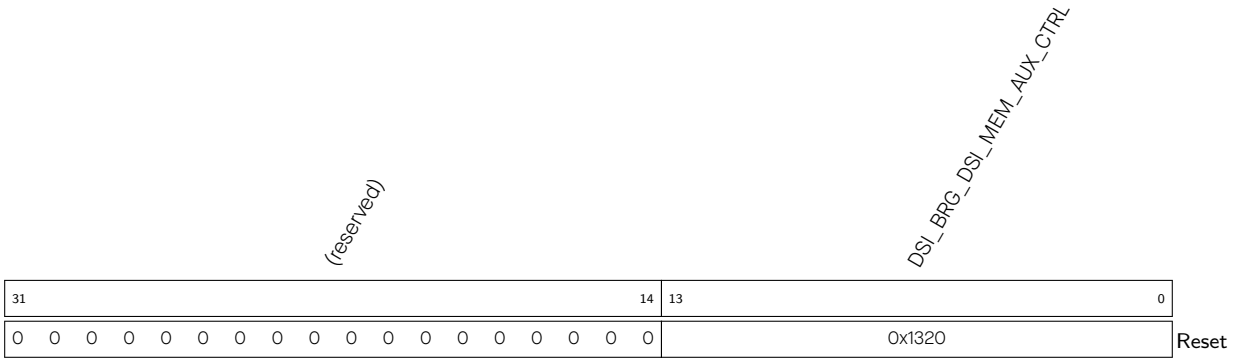
DSI_BRG_DMA_FRAME_INTERVAL_EN 配置是否使能帧传输之间的间隔。

0: 禁用

1: 使能

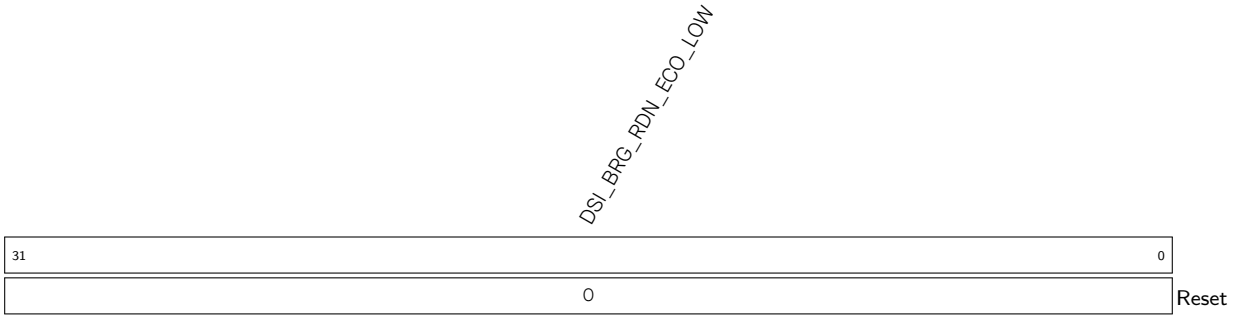
(R/W)

Register 43.20. DSI_BRG_MEM_AUX_CTRL_REG (0x0070)



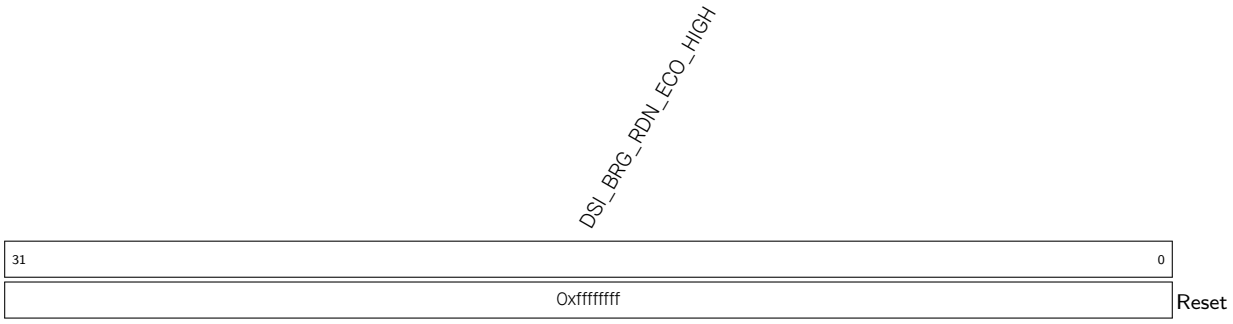
DSI_BRG_DSI_MEM_AUX_CTRL 配置 dsi_bridge FIFO 存储器的辅助控制。(R/W)

Register 43.21. DSI_BRG_RDN_ECO_LOW_REG (0x0078)



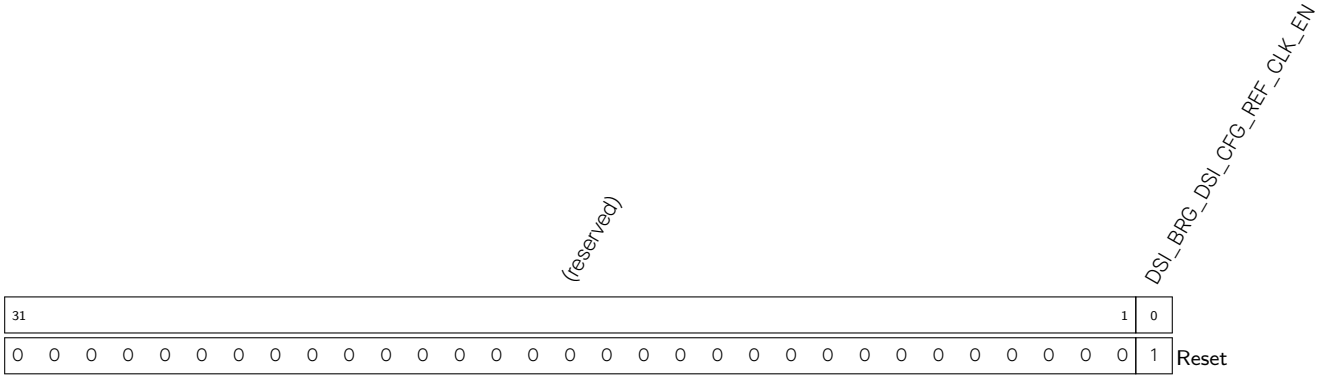
DSI_BRG_RDN_ECO_LOW RDN ECO 低寄存器值。(R/W)

Register 43.22. DSI_BRG_RDN_ECO_HIGH_REG (0x007C)



DSI_BRG_RDN_ECO_HIGH RDN ECO 高寄存器值。(R/W)

Register 43.23. DSI_BRG_HOST_CTRL_REG (0x0080)



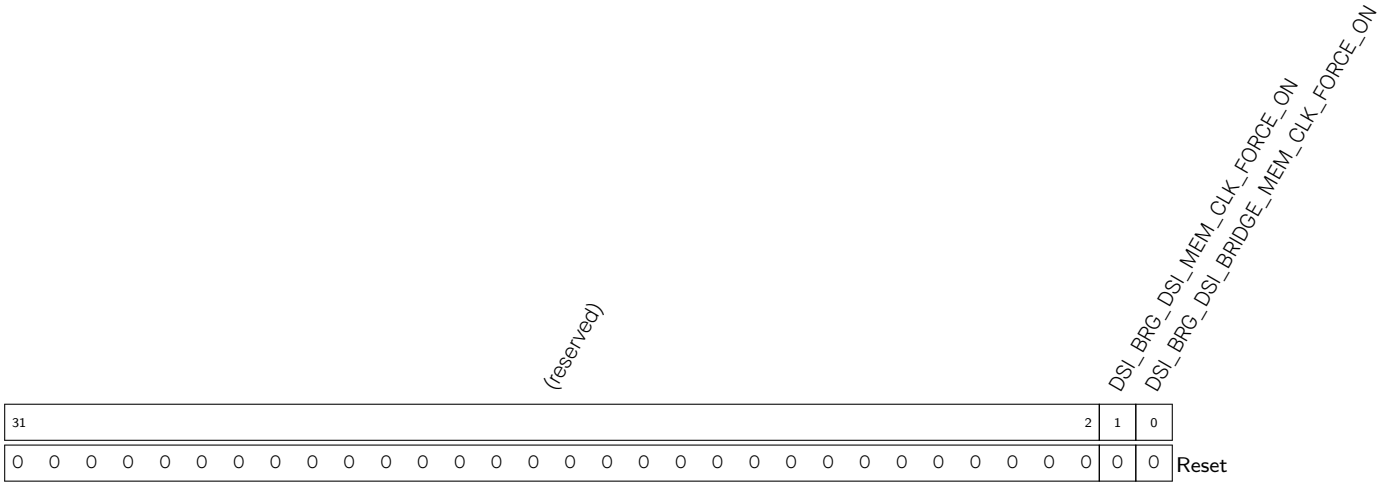
DSI_BRG_DSI_CFG_REF_CLK_EN 配置是否使能 dsi_host 的 refclk 和 cfg_clk 时钟。

0: 禁用

1: 使能

(R/W)

Register 43.24. DSI_BRG_MEM_CLK_CTRL_REG (0x0084)



DSI_BRG_DSI_BRIDGE_MEM_CLK_FORCE_ON 配置是否强制开启 dsi_bridge FIFO 存储器的时钟。

0: 禁用

1: 强制开启

(R/W)

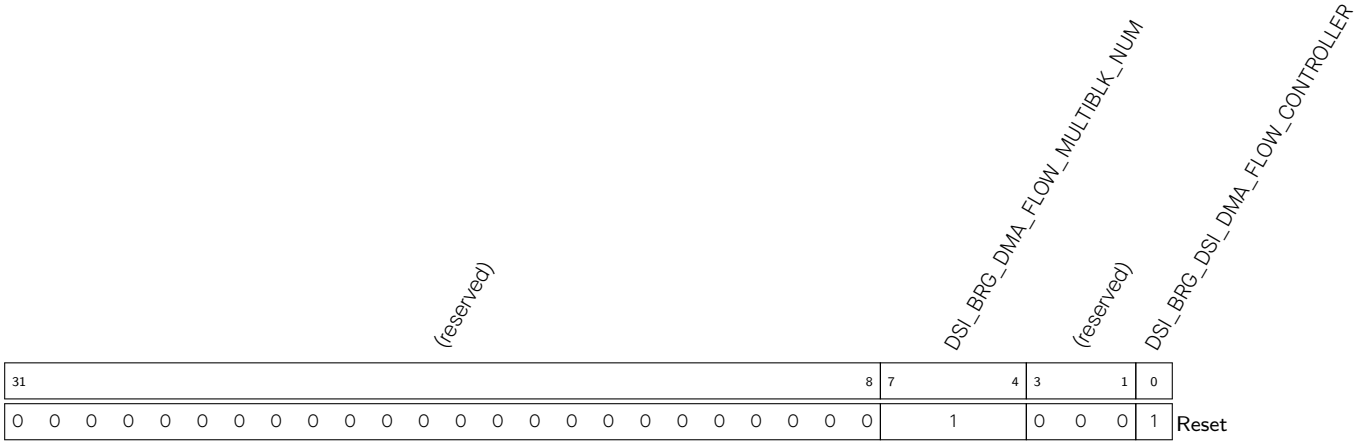
DSI_BRG_DSI_MEM_CLK_FORCE_ON 配置是否强制开启 DPI FIFO 存储器的时钟。

0: 禁用

1: 强制开启

(R/W)

Register 43.25. DSI_BRG_DMA_FLOW_CTRL_REG (0x0088)



DSI_BRG_DSI_DMA_FLOW_CONTROLLER 配置流控制器。

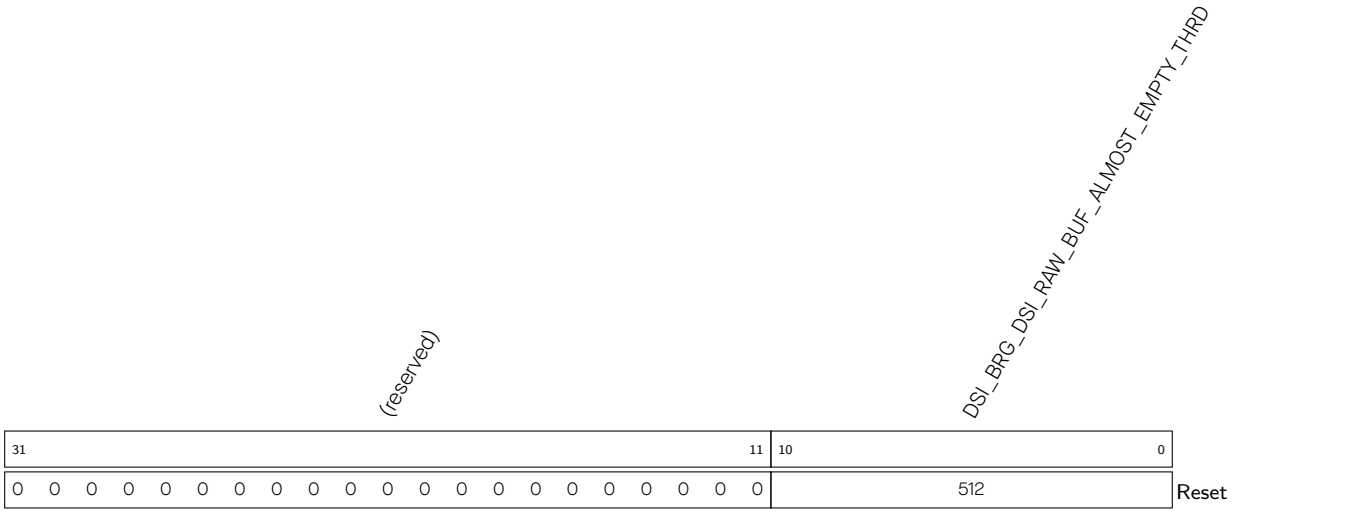
0: dmac 作为流控制器

1: dsi_bridge 作为流控制器

(R/W)

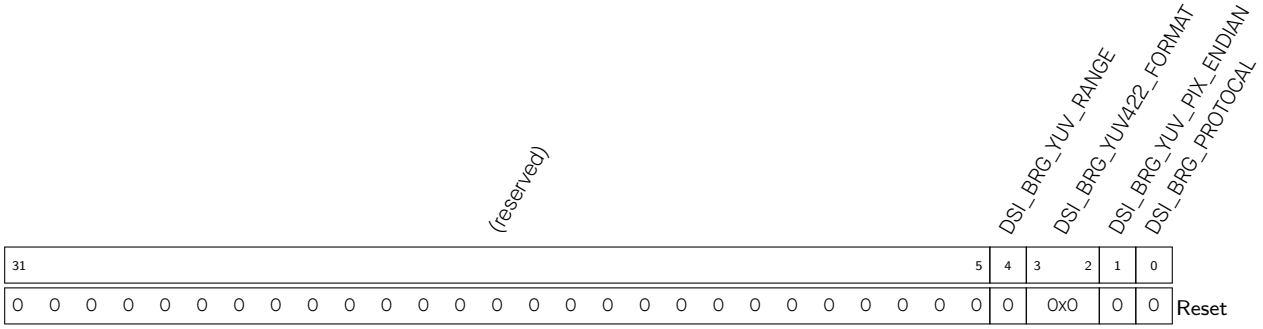
DSI_BRG_DMA_FLOW_MULTIBLK_NUM 配置当多块使能且 dmac 作为流控制器时的块数量。(R/W)

Register 43.26. DSI_BRG_RAW_BUF_ALMOST_EMPTY_THRD_REG (0x008C)



DSI_BRG_DSI_RAW_BUF_ALMOST_EMPTY_THRD 配置 FIFO 几乎空阈值。此值仅在 DMAC 作为流控制器时有效。(R/W)

Register 43.27. DSI_BRG_YUV_CFG_REG (0x0090)



DSI_BRG_PROTOCAL 配置 YUV 到 RGB 协议。

0: BT.601

1: BT.709

(R/W)

DSI_BRG_YUV_PIX_ENDIAN 配置 YUV 像素字节序。

0: Y0U0Y1V1Y2U2Y3V3

1: Y3U3Y2V2Y1U1Y0V0

(R/W)

DSI_BRG_YUV422_FORMAT 配置 YUV422 存储格式。

0: YUYV

1: YVYU

2: UYVY

3: VYUY

(R/W)

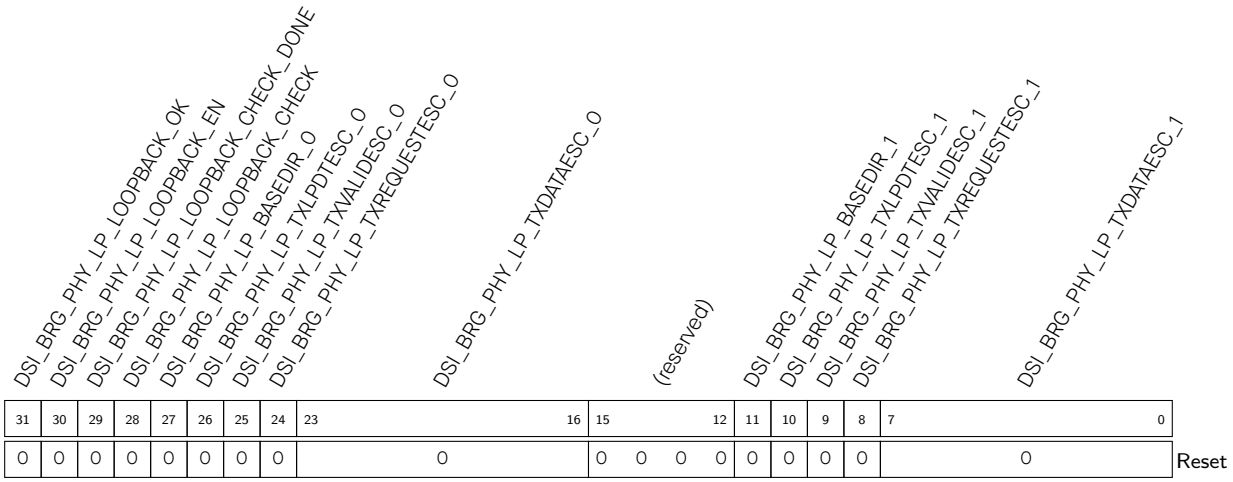
DSI_BRG_YUV_RANGE 配置 YUV 像素范围。

0: 限制范围

1: 全范围

(R/W)

Register 43.28. DSI_BRG_PHY_LP_LOOPBACK_CTRL_REG (0x0094)



DSI_BRG_PHY_LP_TXDATAESC_1 当使能 DSI PHY 低功耗回环测试时控制 txdataesc_1 (R/W)

DSI_BRG_PHY_LP_TXREQUESTESC_1 当使能 DSI PHY 低功耗回环测试时控制 txrequestesc_1 (R/W)

DSI_BRG_PHY_LP_TXVALIDESC_1 当使能 DSI PHY 低功耗回环测试时控制 txvalidesc_1 (R/W)

DSI_BRG_PHY_LP_TXLPDTESC_1 当使能 DSI PHY 低功耗回环测试时控制 txlpdtesc_1 (R/W)

DSI_BRG_PHY_LP_BASEDIR_1 当使能 DSI PHY 低功耗回环测试时控制 basedir_1 (R/W)

DSI_BRG_PHY_LP_TXDATAESC_0 当使能 DSI PHY 低功耗回环测试时控制 txdataesc_0 (R/W)

DSI_BRG_PHY_LP_TXREQUESTESC_0 当使能 DSI PHY 低功耗回环测试时控制 txrequestesc_0 (R/W)

DSI_BRG_PHY_LP_TXVALIDESC_0 当使能 DSI PHY 低功耗回环测试时控制 txvalidesc_0 (R/W)

DSI_BRG_PHY_LP_TXLPDTESC_0 当使能 DSI PHY 低功耗回环测试时控制 txlpdtesc_0 (R/W)

DSI_BRG_PHY_LP_BASEDIR_0 当使能 DSI PHY 低功耗回环测试时控制 basedir_0 (R/W)

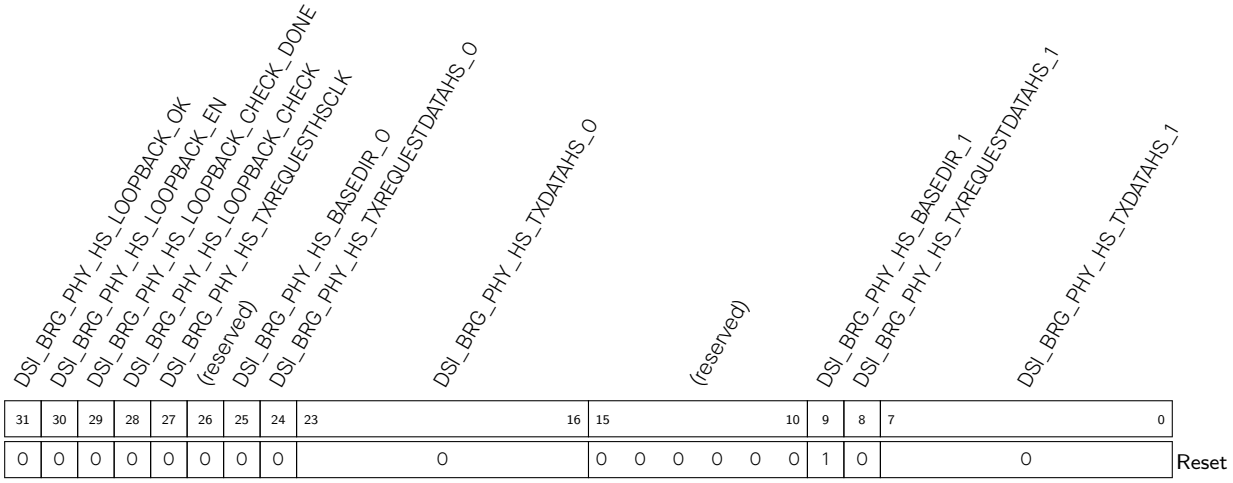
DSI_BRG_PHY_LP_LOOPBACK_CHECK DSI PHY 低功耗回环测试开始检查 (WT)

DSI_BRG_PHY_LP_LOOPBACK_CHECK_DONE DSI PHY 低功耗回环测试检查完成 (RO)

DSI_BRG_PHY_LP_LOOPBACK_EN DSI PHY 低功耗回环控制使能 (R/W)

DSI_BRG_PHY_LP_LOOPBACK_OK DSI PHY 低功耗回环测试结果 (RO)

Register 43.29. DSI_BRG_PHY_HS_LOOPBACK_CTRL_REG (0x0098)



DSI_BRG_PHY_HS_TXDATAHS_1 当使能 DSI PHY 高速回环测试时控制 txdatahs_1 (R/W)

DSI_BRG_PHY_HS_TXREQUESTDATAHS_1 当使能 DSI PHY 高速回环测试时控制 txrequestdatahs_1 (R/W)

DSI_BRG_PHY_HS_BASEDIR_1 当使能 DSI PHY 高速回环测试时控制 basedir_1 (R/W)

DSI_BRG_PHY_HS_TXDATAHS_0 当使能 DSI PHY 高速回环测试时控制 txdatahs_0 (R/W)

DSI_BRG_PHY_HS_TXREQUESTDATAHS_0 当使能 DSI PHY 高速回环测试时控制 txrequestdatahs_0 (R/W)

DSI_BRG_PHY_HS_BASEDIR_0 当使能 DSI PHY 高速回环测试时控制 basedir_0 (R/W)

DSI_BRG_PHY_HS_TXREQUESTHSCLK 当使能 DSI PHY 高速回环测试时控制 txrequesthsclk (R/W)

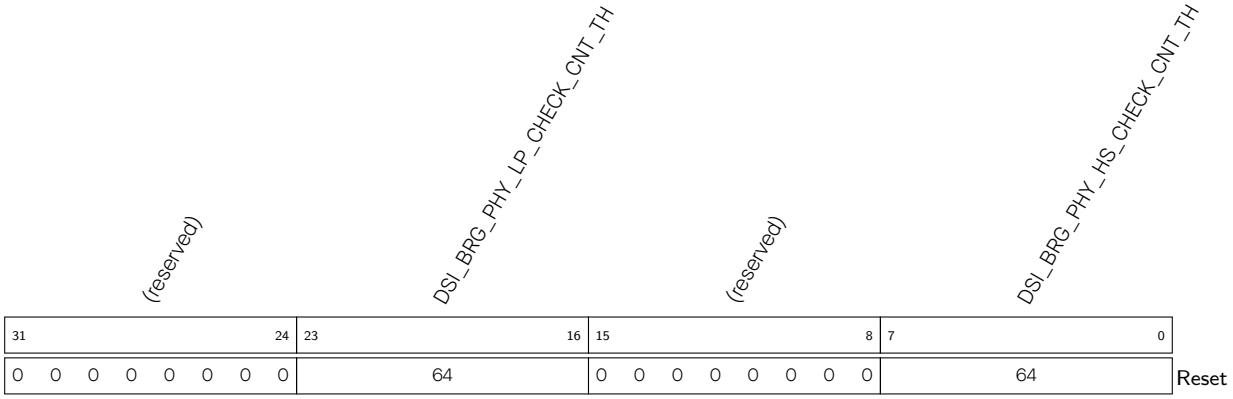
DSI_BRG_PHY_HS_LOOPBACK_CHECK DSI PHY 高速回环测试开始检查 (WT)

DSI_BRG_PHY_HS_LOOPBACK_CHECK_DONE DSI PHY 高速回环测试检查完成 (RO)

DSI_BRG_PHY_HS_LOOPBACK_EN DSI PHY 高速回环控制使能 (R/W)

DSI_BRG_PHY_HS_LOOPBACK_OK DSI PHY 高速回环测试结果 (RO)

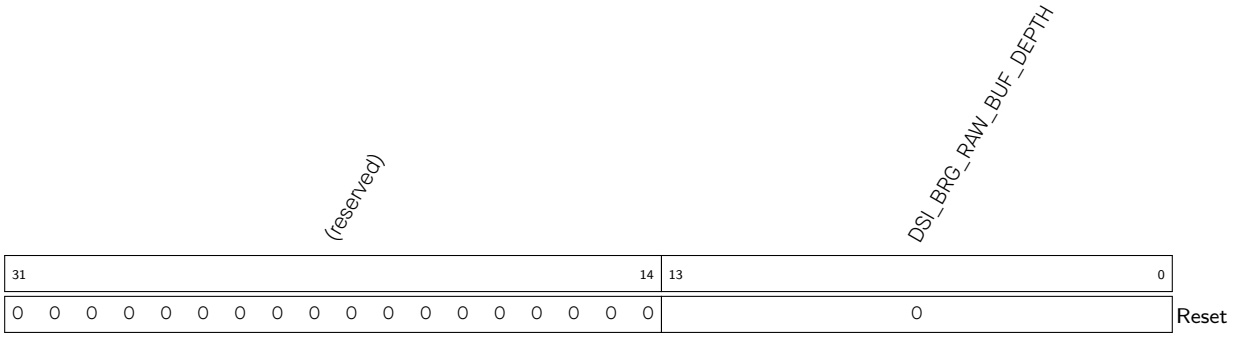
Register 43.30. DSI_BRG_PHY_LOOPBACK_CNT_REG (0x009C)



DSI_BRG_PHY_HS_CHECK_CNT_TH 高速回环测试检查计数。(R/W)

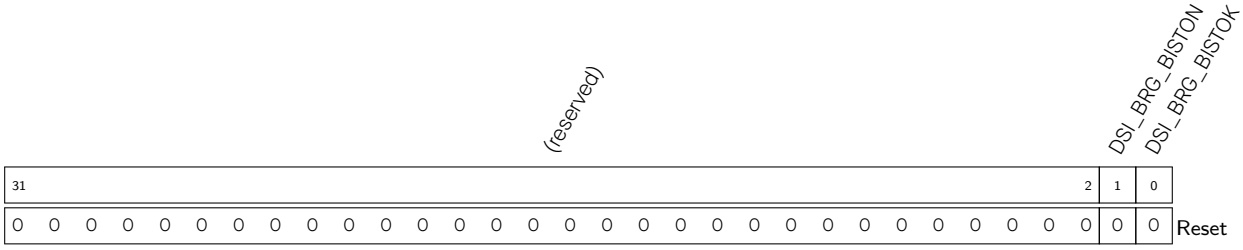
DSI_BRG_PHY_LP_CHECK_CNT_TH 低功耗回环测试检查计数。(R/W)

Register 43.31. DSI_BRG_FIFO_FLOW_STATUS_REG (0x0014)



DSI_BRG_RAW_BUF_DEPTH 表示 dsi_bridge FIFO 的当前深度。(RO)

Register 43.32. DSI_BRG_HOST_BIST_CTL_REG (0x0060)



DSI_BRG_BISTOK BIST 结果指示。(RO)

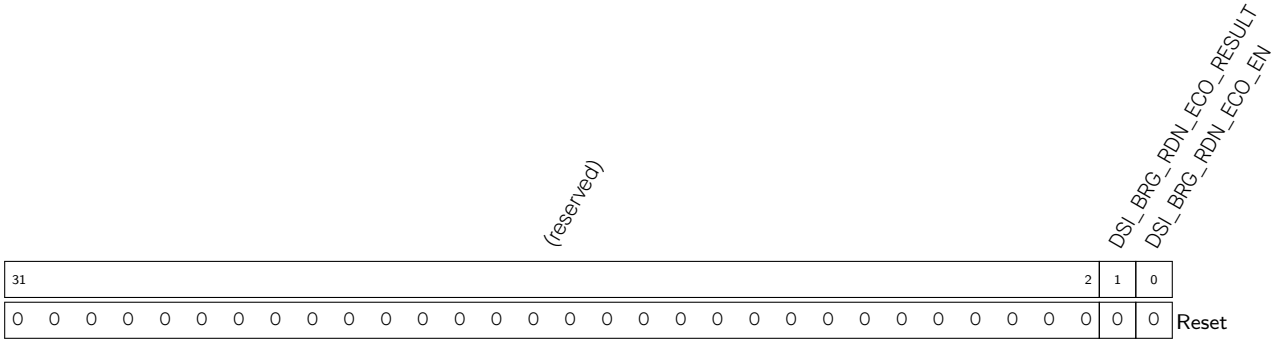
DSI_BRG_BISTON 配置 BIST 使能。

0: 禁用

1: 使能

(R/W)

Register 43.33. DSI_BRG_RDN_ECO_CS_REG (0x0074)



DSI_BRG_RDN_ECO_EN 配置 RDN ECO 使能。

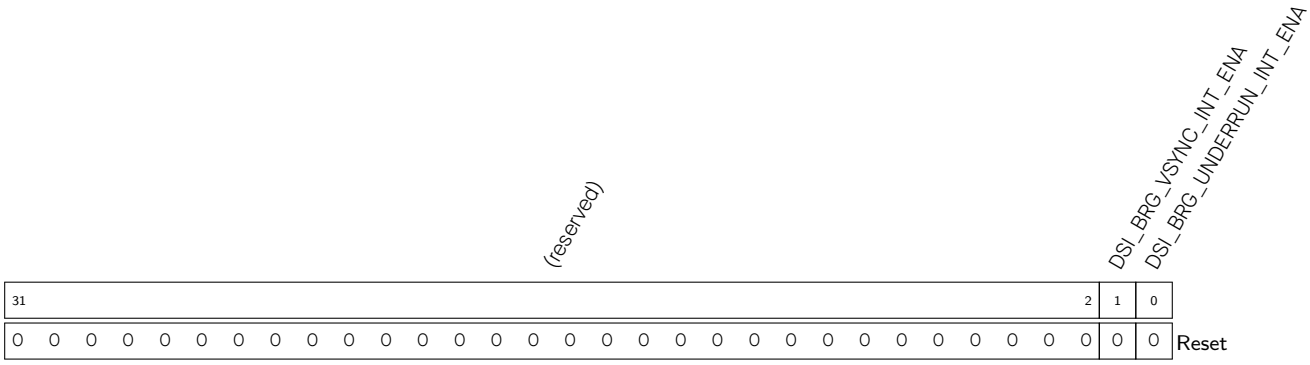
0: 禁用

1: 使能

(R/W)

DSI_BRG_RDN_ECO_RESULT RDN ECO 结果。(RO)

Register 43.34. DSI_BRG_INT_ENA_REG (0x0050)



DSI_BRG_UNDERRUN_INT_ENA 配置 DSI_BRG_UNDERRUN_INT 使能。

0: 禁用

1: 使能

(R/W)

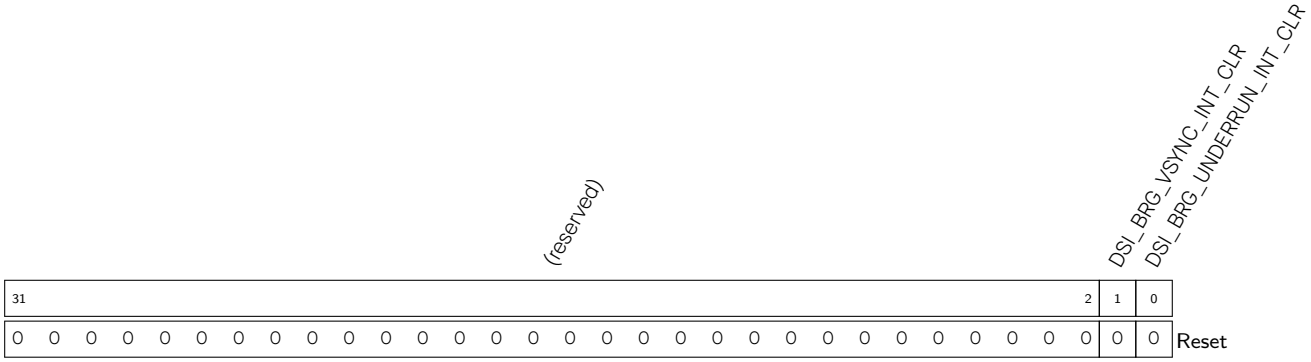
DSI_BRG_VSYNC_INT_ENA 配置 DSI_BRG_VSYNC_INT 使能。

0: 禁用

1: 使能

(R/W)

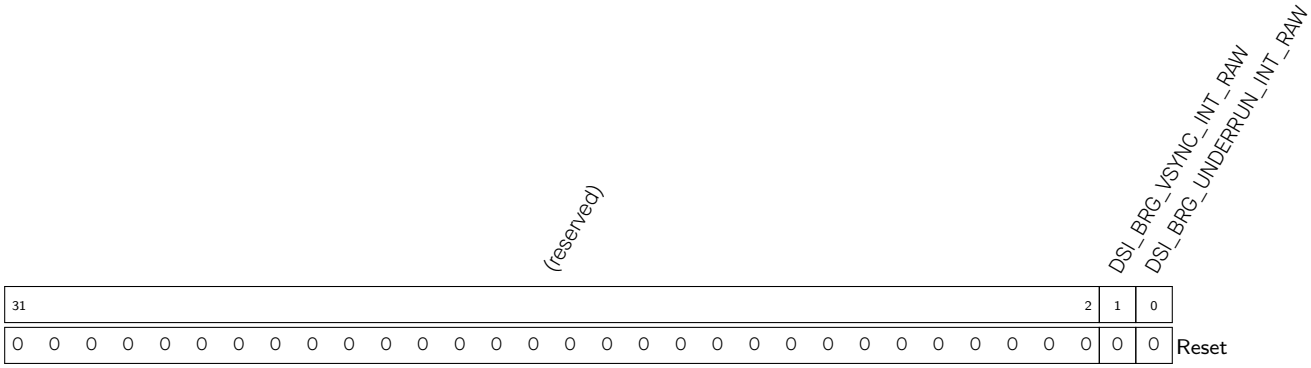
Register 43.35. DSI_BRG_INT_CLR_REG (0x0054)



DSI_BRG_UNDERRUN_INT_CLR 写入 1 以清除 DSI_BRG_INT_RAW_REG 中的 DSI_BRG_-UNDERRUN_INT_RAW 字段。(WT)

DSI_BRG_VSYNC_INT_CLR 写入 1 以清除 DSI_BRG_INT_RAW_REG 中的 DSI_BRG_VSYNC_INT_-RAW 字段。(WT)

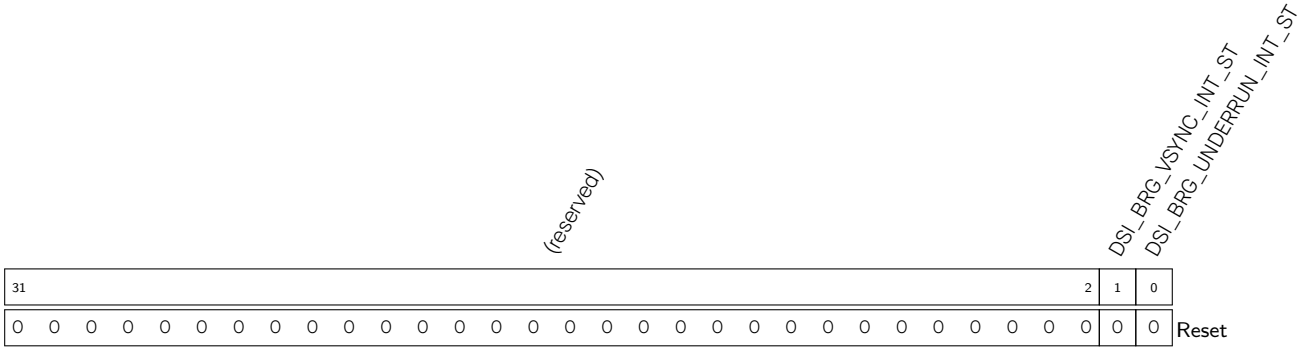
Register 43.36. DSI_BRG_INT_RAW_REG (0x0058)



DSI_BRG_UNDERRUN_INT_RAW dpi_underrun 的原始中断状态。(R/WTC/SS)

DSI_BRG_VSYNC_INT_RAW dpi_vsync 的原始中断状态。(R/WTC/SS)

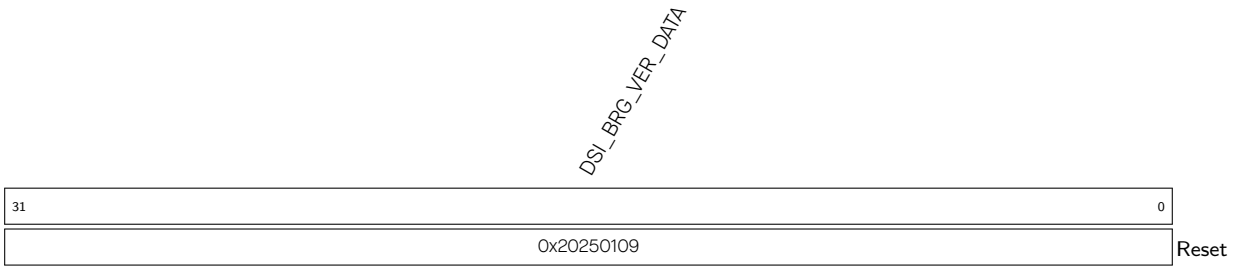
Register 43.37. DSI_BRG_INT_ST_REG (0x005C)



DSI_BRG_UNDERRUN_INT_ST 表示 dpi_underrun 的掩码中断状态。(RO)

DSI_BRG_VSYNC_INT_ST 表示 dpi_vsync 的掩码中断状态。(RO)

Register 43.38. DSI_BRG_VER_DATE_REG (0x0100)



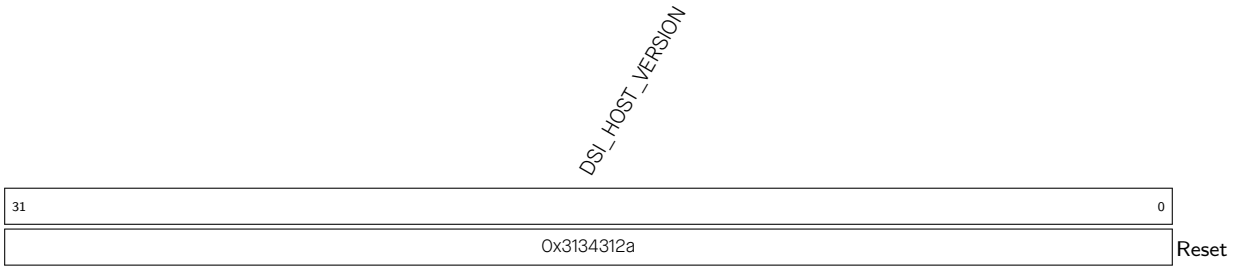
DSI_BRG_VER_DATA 表示此寄存器文件的 CSV 版本日期。(R/W)

43.8.2 MIPI DSI 主机寄存器

本节中的地址相对于 MIPI DSI Host 基地址，该地址在第 9 章的表 9.3-2 中提供 *系统和存储器*。

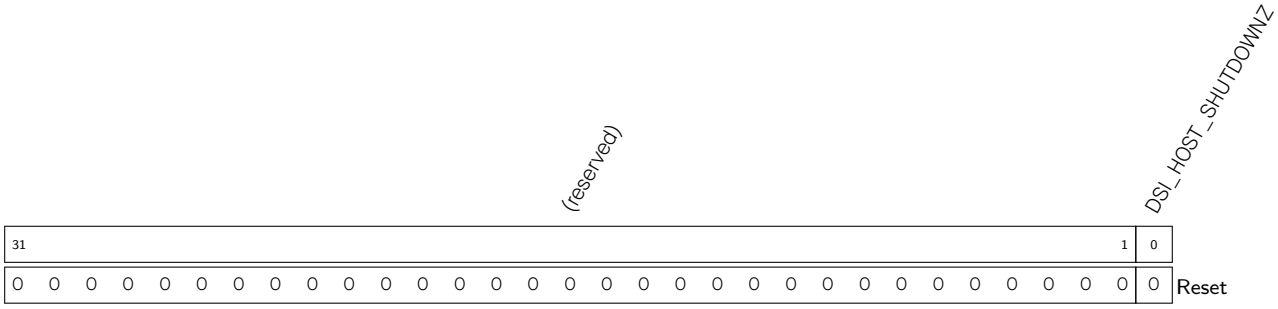
有关如何配置保留字段，请参阅 配置保留寄存器字段 部分。

Register 43.39. DSI_HOST_VERSION_REG (0x0000)



DSI_HOST_VERSION 表示 DSI Host 版本。(RO)

Register 43.40. DSI_HOST_PWR_UP_REG (0x0004)



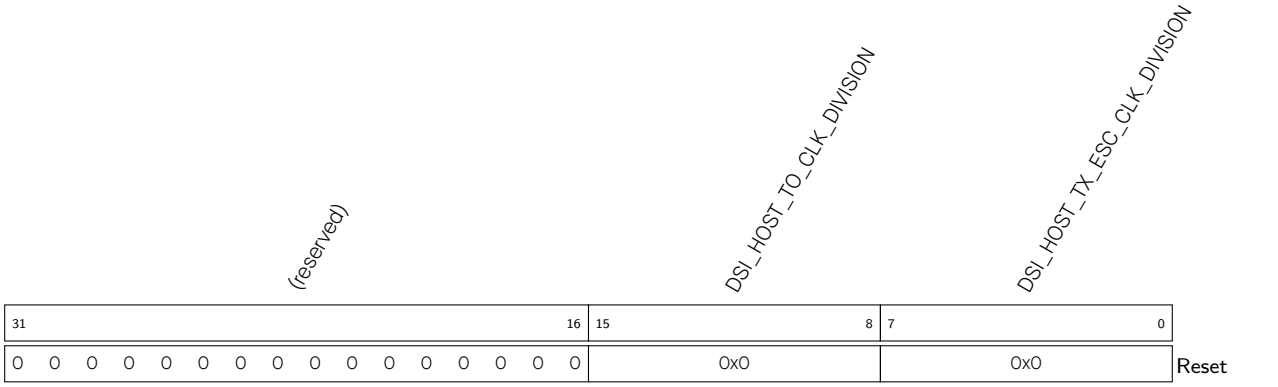
DSI_HOST_SHUTDOWNZ 配置 DSI Host 电源。

0: 复位

1: 上电

(R/W)

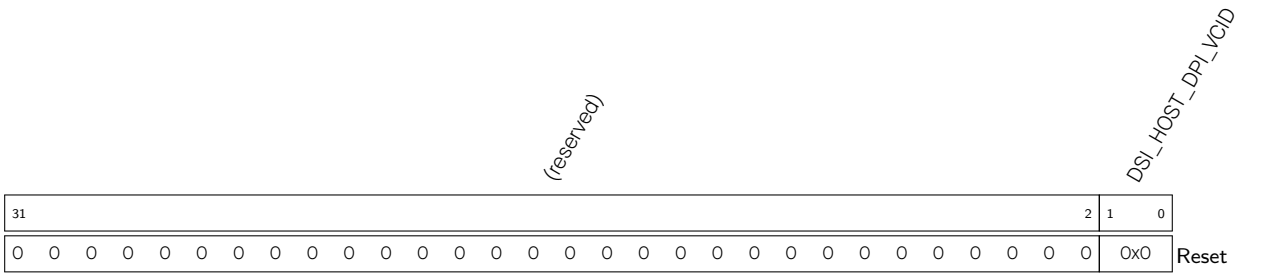
Register 43.41. DSI_HOST_CLKMGR_CFG_REG (0x0008)



DSI_HOST_TX_ESC_CLK_DIVISION 配置 TX Escape 时钟的分频因子。值为 0 或 1 将停止 TX Escape 时钟。(R/W)

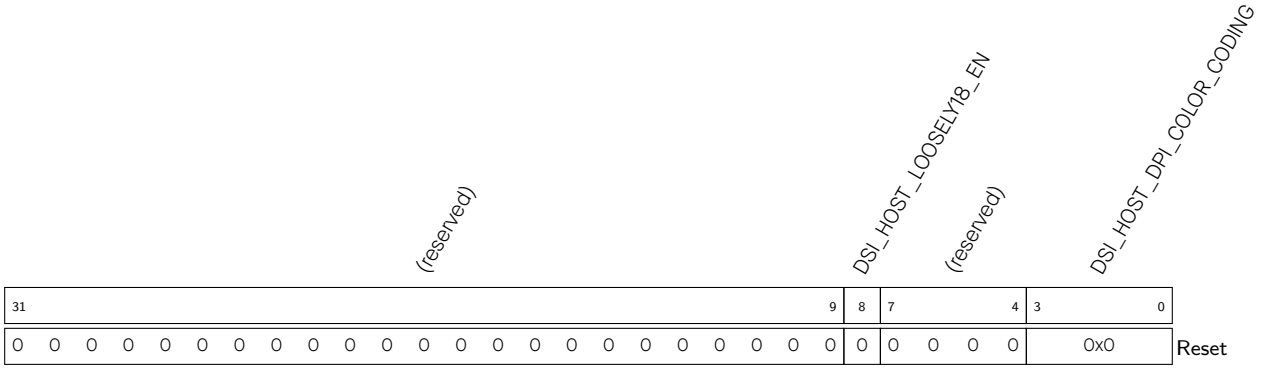
DSI_HOST_TO_CLK_DIVISION 配置从 HS 到 LP 和从 LP 到 HS 的超时计数器时钟的分频因子。(R/W)

Register 43.42. DSI_HOST_DPI_VCID_REG (0x000C)



DSI_HOST_DPI_VCID 配置 DPI 虚拟通道 ID。(R/W)

Register 43.43. DSI_HOST_DPI_COLOR_CODING_REG (0x0010)



DSI_HOST_DPI_COLOR_CODING 配置 DPI 颜色编码，详细信息请参考表 43.4-2。

0: RGB565 配置 0

1: RGB565 配置 1

2: RGB565 配置 2

3: RGB666 配置 0

4: RGB666 配置 1

5: RGB888

others: 保留

(R/W)

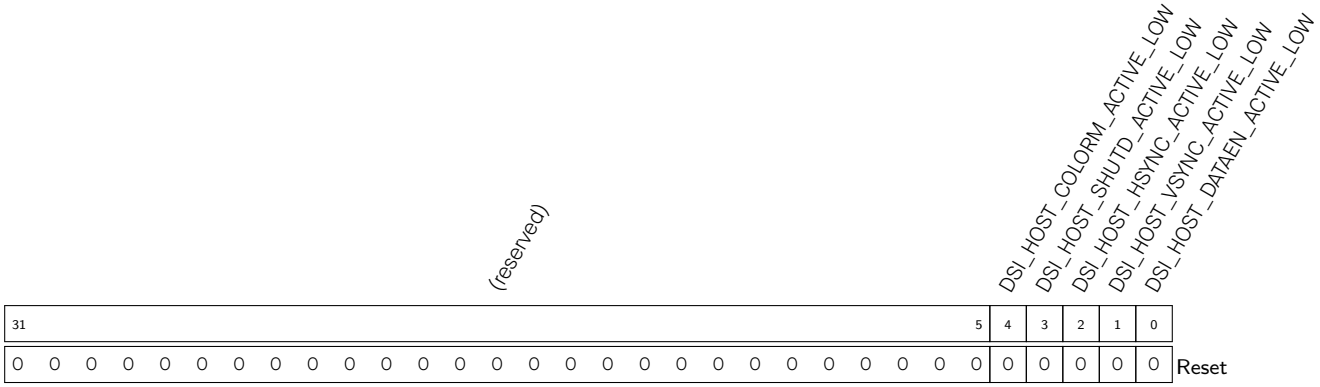
DSI_HOST_LOOSELY18_EN 配置 DSI 主机在 18 位模式下是否启用松散打包模式。

0: 禁用

1: 使能

(R/W)

Register 43.44. DSI_HOST_DPI_CFG_POL_REG (0x0014)



DSI_HOST_DATAEN_ACTIVE_LOW 配置 DPI dpidataen 信号极性。

0: 高电平有效

1: 低电平有效

(R/W)

DSI_HOST_VSYNC_ACTIVE_LOW 配置 DPI dpivsync 信号极性。

0: 高电平有效

1: 低电平有效

(R/W)

DSI_HOST_HSYNC_ACTIVE_LOW 配置 DPI dpihsync 信号极性。

0: 高电平有效

1: 低电平有效

(R/W)

DSI_HOST_SHUTD_ACTIVE_LOW 配置 DPI dpishutdn 信号极性。

0: 高电平有效

1: 低电平有效

(R/W)

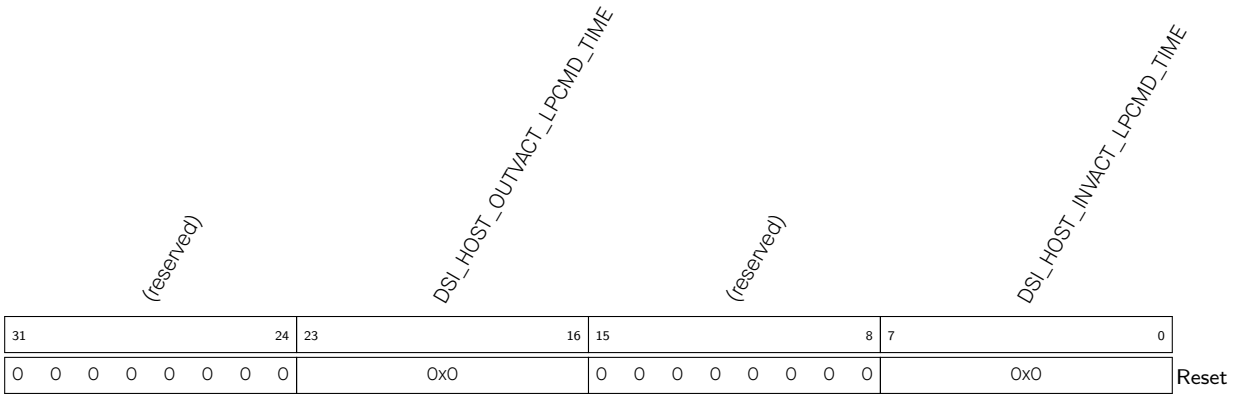
DSI_HOST_COLORM_ACTIVE_LOW 配置 DPI dpicolorm 信号极性。

0: 高电平有效

1: 低电平有效

(R/W)

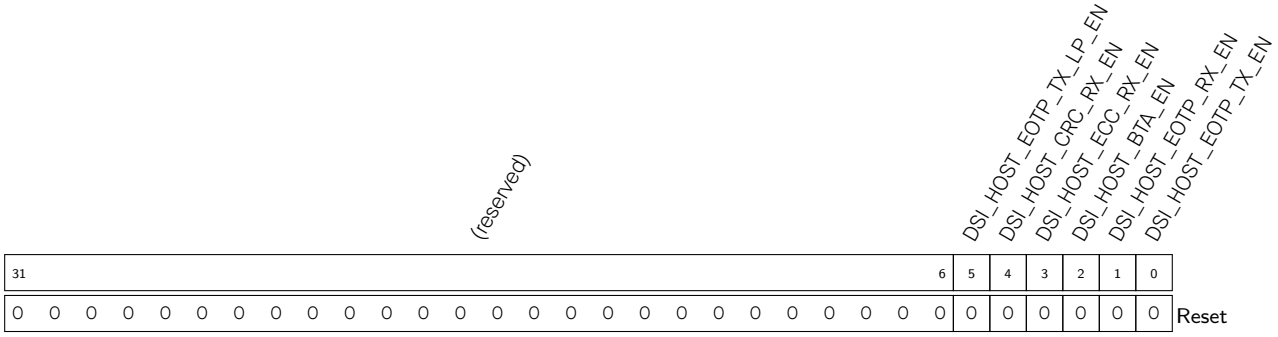
Register 43.45. DSI_HOST_DPI_LP_CMD_TIM_REG (0x0018)



DSI_HOST_INVACT_LPCMD_TIME 配置在 VACT 区域期间 LP 命令传输时一行中可以容纳的最大数据包大小（以字节为单位）。(R/W)

DSI_HOST_OUTVACT_LPCMD_TIME 配置在 VSA VBP VFP 区域期间 LP 命令传输时一行中可以容纳的最大数据包大小（以字节为单位）。(R/W)

Register 43.46. DSI_HOST_PCKHDL_CFG_REG (0x002C)



DSI_HOST_EOTP_TX_EN 配置是否使能 HS 中 EoTp 传输。

0: 禁用

1: 使能

(R/W)

DSI_HOST_EOTP_RX_EN 配置是否使能 EoTp 接收。

0: 禁用

1: 使能

(R/W)

DSI_HOST_BTA_EN 配置是否使能总线转向 (BTA) 请求。

0: 禁用

1: 使能

(R/W)

DSI_HOST_ECC_RX_EN 配置是否使能 ECC 接收、纠错和报告。

0: 禁用

1: 使能

(R/W)

DSI_HOST_CRC_RX_EN 配置是否使能 CRC 接收和错误报告。

0: 禁用

1: 使能

(R/W)

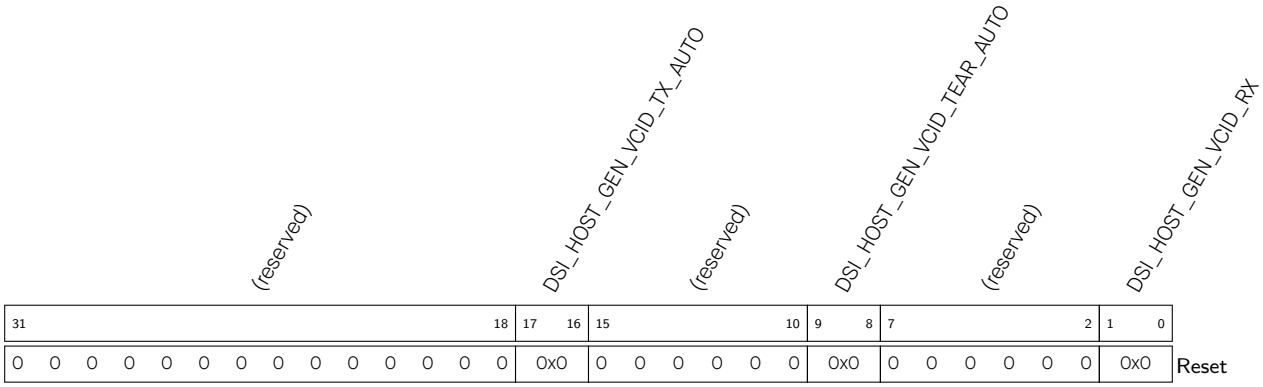
DSI_HOST_EOTP_TX_LP_EN 配置是否使能 LP 中 EoTp 传输。

0: 禁用

1: 使能

(R/W)

Register 43.47. DSI_HOST_GEN_VCID_REG (0x0030)

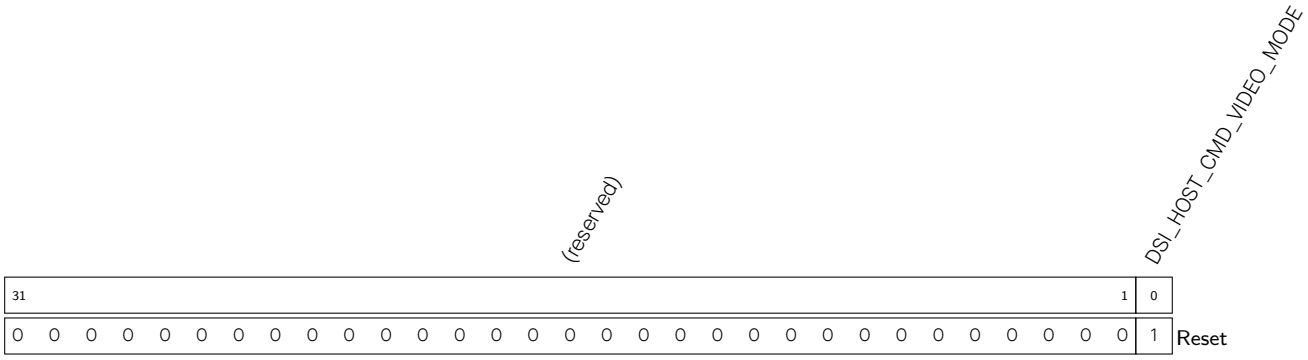


DSI_HOST_GEN_VCID_RX 配置通用接口回读虚拟通道 ID。(R/W)

DSI_HOST_GEN_VCID_TEAR_AUTO 配置硬件撕裂效应虚拟通道 ID。(R/W)

DSI_HOST_GEN_VCID_TX_AUTO 配置通用接口自动生成或传输的数据包虚拟通道 ID。(R/W)

Register 43.48. DSI_HOST_MODE_CFG_REG (0x0034)



DSI_HOST_CMD_VIDEO_MODE 配置操作模式。

0: 视频模式

1: 命令模式

(R/W)

Register 43.49. DSI_HOST_VID_MODE_CFG_REG (0x0038)

(reserved)	DSI_HOST_VPG_ORIENTATION	(reserved)	DSI_HOST_VPG_MODE	(reserved)	DSI_HOST_VPG_EN	DSI_HOST_LP_CMD_EN	DSI_HOST_FRAME_BTA_ACK_EN	DSI_HOST_LP_HFP_EN	DSI_HOST_LP_HBP_EN	DSI_HOST_LP_VACT_EN	DSI_HOST_LP_VFP_EN	DSI_HOST_LP_VBP_EN	DSI_HOST_LP_VSA_EN	(reserved)	DSI_HOST_VID_MODE_TYPE	
31 25	24	23 21	20	19 17	16	15	14	13	12	11	10	9	8	7 2	1 0	
0 0 0 0 0 0 0	0	0 0 0	0	0 0 0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0	0x0	Reset

DSI_HOST_VID_MODE_TYPE 配置视频传输类型。(R/W)

0: 带同步脉冲的非突发模式

1: 带同步事件的非突发模式

2: 突发模式

3: 突发模式

DSI_HOST_LP_VSA_EN 配置是否使能在时序允许时在 VSA 周期内返回 LP。

0: 禁用

1: 使能

(R/W)

DSI_HOST_LP_VBP_EN 配置是否使能在时序允许时在 VBP 周期内返回 LP。

0: 禁用

1: 使能

(R/W)

DSI_HOST_LP_VFP_EN 配置是否使能在时序允许时在 VFP 周期内返回 LP。

0: 禁用

1: 使能

(R/W)

DSI_HOST_LP_VACT_EN 配置是否使能在时序允许时在 VACT 周期内返回 LP。

0: 禁用

1: 使能

(R/W)

DSI_HOST_LP_HBP_EN 配置是否使能在时序允许时在 HBP 周期内返回 LP。

0: 禁用

1: 使能

(R/W)

DSI_HOST_LP_HFP_EN 配置是否使能在时序允许时在 HFP 周期内返回 LP。

0: 禁用

1: 使能

(R/W)

见下页...

Register 43.49. DSI_HOST_VID_MODE_CFG_REG (0x0038)

接上页...

DSI_HOST_FRAME_BTA_ACK_EN 配置是否在帧结束时启用确认响应请求。

0: 禁用

1: 使能

(R/W)

DSI_HOST_LP_CMD_EN 配置是否仅在 LP 模式下启用命令传输。

0: 禁用

1: 使能

(R/W)

DSI_HOST_VPG_EN 配置是否使能视频图案生成器。

0: 禁用

1: 使能

(R/W)

DSI_HOST_VPG_MODE 配置图案类型。

0: 彩条

1: 误码图案

(R/W)

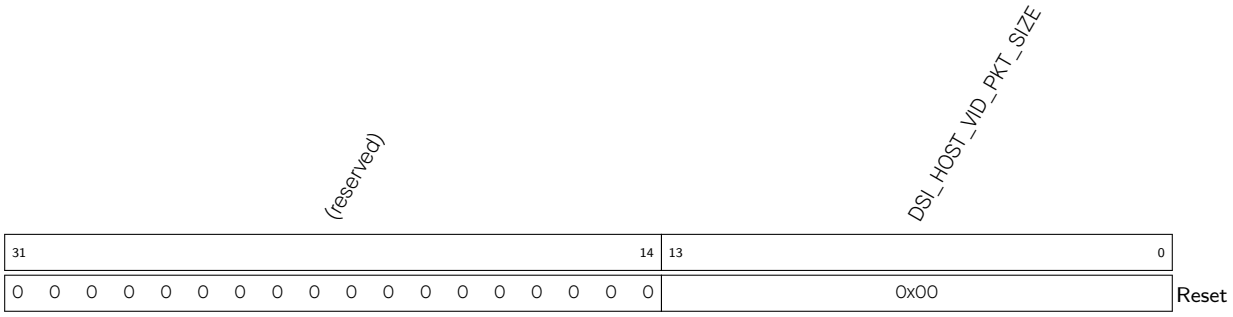
DSI_HOST_VPG_ORIENTATION 配置彩条方向。

0: 垂直

1: 水平

(R/W)

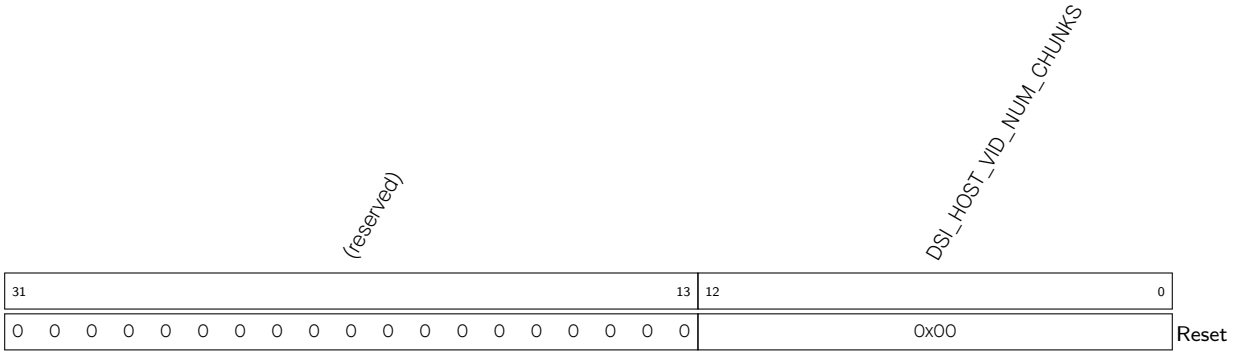
Register 43.50. DSI_HOST_VID_PKT_SIZE_REG (0x003C)



DSI_HOST_VID_PKT_SIZE 配置单个视频数据包中的像素数。(R/W)

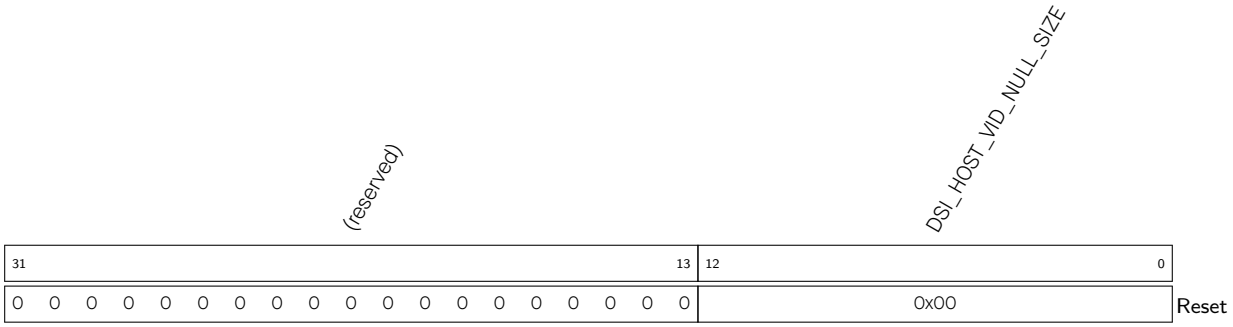
对于 18 位非松散打包数据，该值必须是 4 的倍数。

Register 43.51. DSI_HOST_VID_NUM_CHUNKS_REG (0x0040)



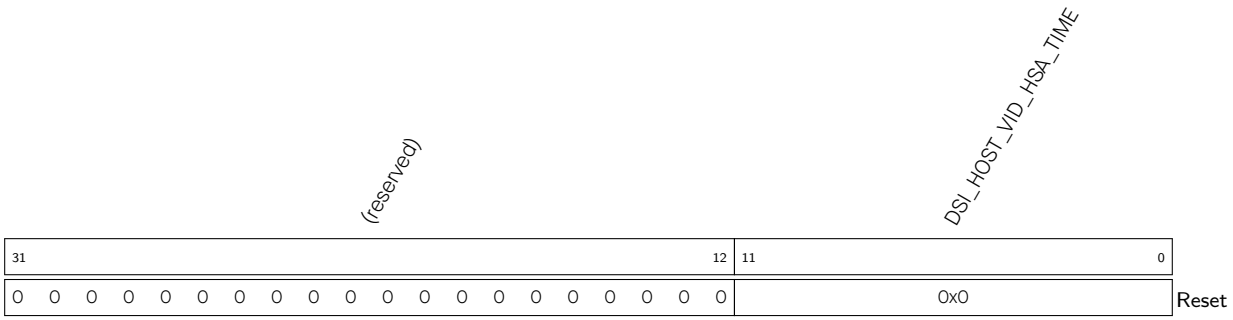
DSI_HOST_VID_NUM_CHUNKS 配置在一行中传输的块数。一个块由视频数据包和空数据包组成。(R/W)

Register 43.52. DSI_HOST_VID_NULL_SIZE_REG (0x0044)



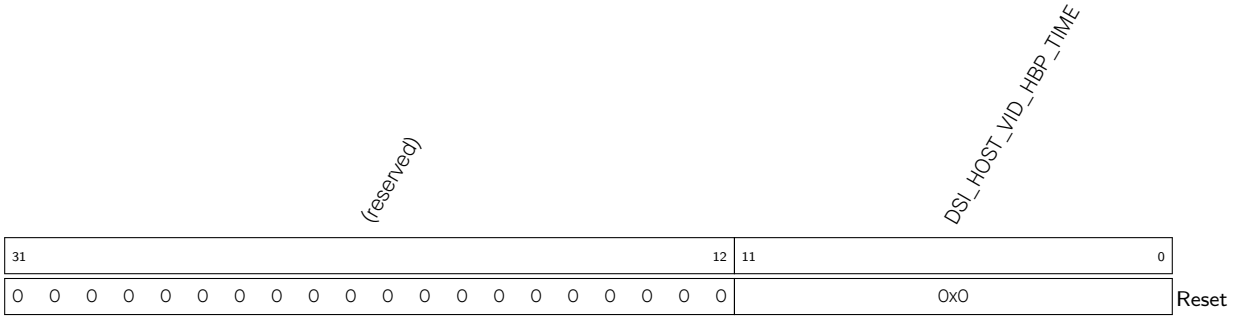
DSI_HOST_VID_NULL_SIZE 配置空数据包内的字节数。设置为 0 以禁用空数据包。(R/W)

Register 43.53. DSI_HOST_VID_HSA_TIME_REG (0x0048)



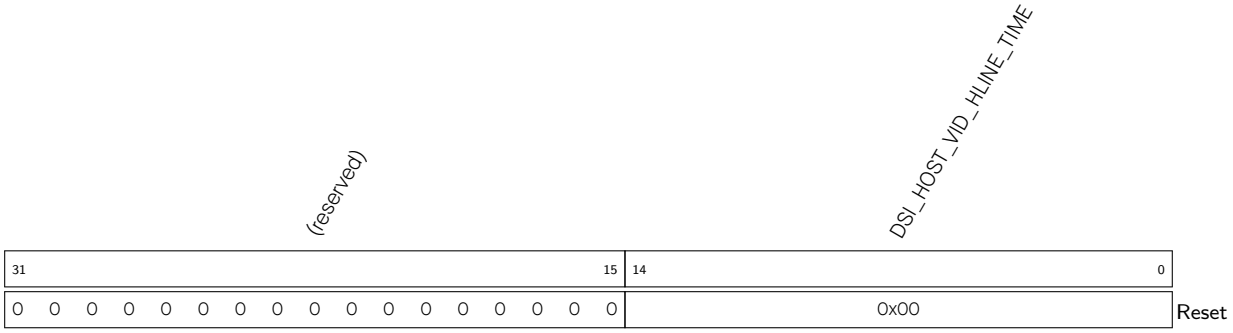
DSI_HOST_VID_HSA_TIME 配置 HSA 周期（以通道字节时钟周期为单位）。(R/W)

Register 43.54. DSI_HOST_VID_HBP_TIME_REG (0x004C)



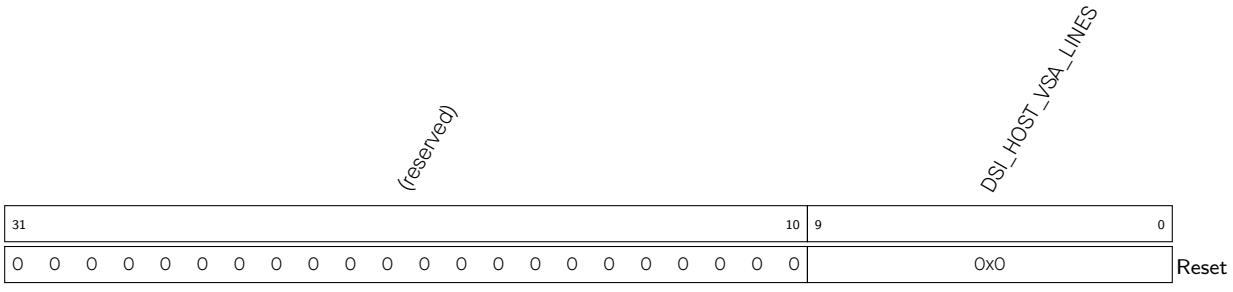
DSI_HOST_VID_HBP_TIME 配置 HBP 周期（以通道字节时钟周期为单位）。(R/W)

Register 43.55. DSI_HOST_VID_HLINE_TIME_REG (0x0050)



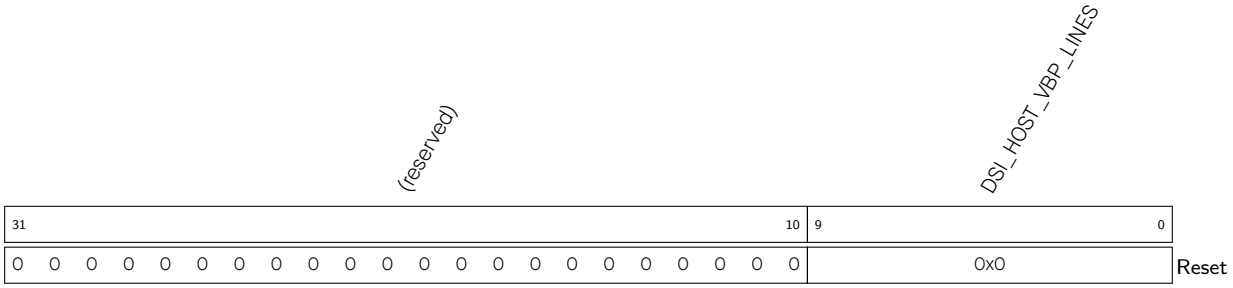
DSI_HOST_VID_HLINE_TIME 配置 HTOTAL 时间（HSA+HBP+HACT+HFP，以通道字节时钟周期为单位）。(R/W)

Register 43.56. DSI_HOST_VID_VSA_LINES_REG (0x0054)



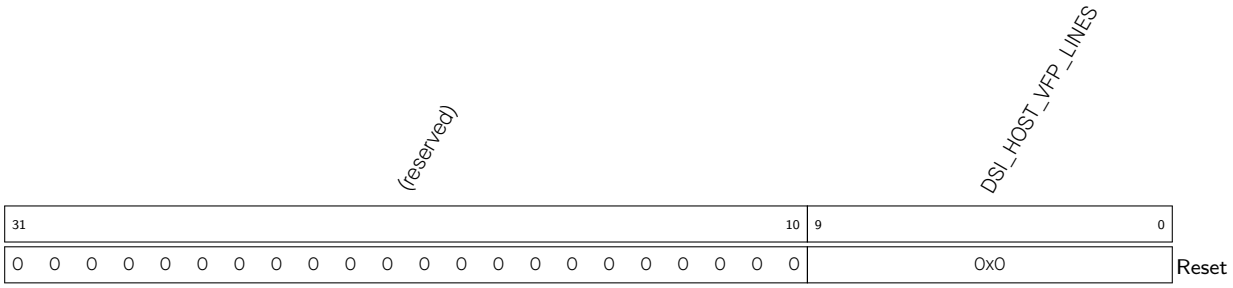
DSI_HOST_VSA_LINES 配置 VSA 周期（以行数表示）。(R/W)

Register 43.57. DSI_HOST_VID_VBP_LINES_REG (0x0058)



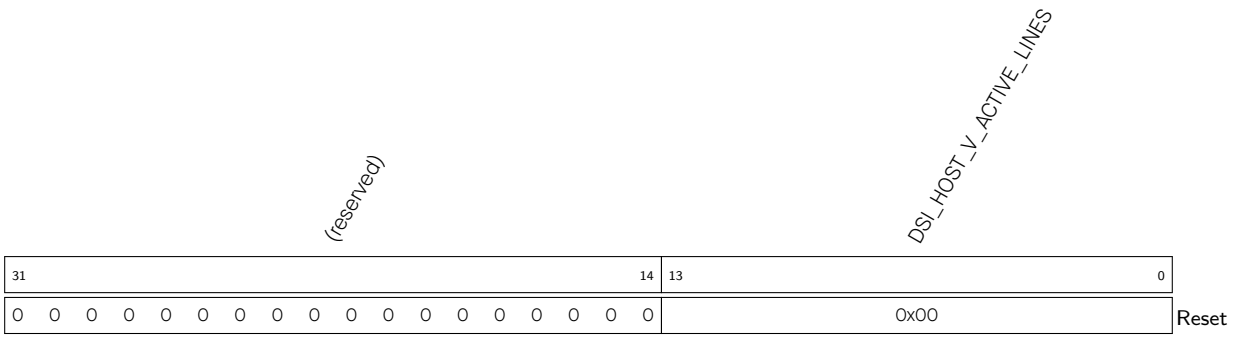
DSI_HOST_VBP_LINES 配置 VBP 周期（以行数表示）。(R/W)

Register 43.58. DSI_HOST_VID_VFP_LINES_REG (0x005C)



DSI_HOST_VFP_LINES 配置 VFP 周期（以行数表示）。(R/W)

Register 43.59. DSI_HOST_VID_VACTIVE_LINES_REG (0x0060)



DSI_HOST_V_ACTIVE_LINES 配置 VACT 周期（以行数表示）。(R/W)

Register 43.60. DSI_HOST_CMD_MODE_CFG_REG (0x0068)

Bits	Field	Reset
31–25	(reserved)	0 0 0 0 0 0 0
24	DSI_HOST_MAX_RD_PKT_SIZE	0
23–20	(reserved)	0 0 0 0
19	DSI_HOST_DCS_LW_TX	0
18	DSI_HOST_DCS_SR_0P_TX	0
17	DSI_HOST_DCS_SW_1P_TX	0
16	DSI_HOST_DCS_SW_0P_TX	0
15	(reserved)	0
14	DSI_HOST_GEN_LW_TX	0
13	DSI_HOST_GEN_SR_2P_TX	0
12	DSI_HOST_GEN_SR_1P_TX	0
11	DSI_HOST_GEN_SR_0P_TX	0
10	DSI_HOST_GEN_SW_2P_TX	0
9	DSI_HOST_GEN_SW_1P_TX	0
8	DSI_HOST_GEN_SW_0P_TX	0
7–2	(reserved)	0 0 0 0 0 0
1	DSI_HOST_ACK_RQST_EN	0
0	DSI_HOST_TEAR_FX_EN	0

DSI_HOST_TEAR_FX_EN 配置是否启用撕裂效应确认请求。

0: 禁用

1: 使能

(R/W)

DSI_HOST_ACK_RQST_EN 配置是否在每次数据包传输后启用确认请求。

0: 禁用

1: 使能

(R/W)

DSI_HOST_GEN_SW_0P_TX 配置零参数通用短写数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_GEN_SW_1P_TX 配置单参数通用短写数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_GEN_SW_2P_TX 配置双参数通用短写数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_GEN_SR_0P_TX 配置零参数通用短读数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_GEN_SR_1P_TX 配置单参数通用短读数据包命令传输类型。

0: HS

1: LP

(R/W)

见下页...

Register 43.60. DSI_HOST_CMD_MODE_CFG_REG (0x0068)

接上页...

DSI_HOST_GEN_SR_2P_TX 配置双参数通用短读数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_GEN_LW_TX 配置通用长写数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_DCS_SW_0P_TX 配置零参数 DCS 短写数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_DCS_SW_1P_TX 配置单参数 DCS 短写数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_DCS_SR_0P_TX 配置零参数 DCS 短读数据包命令传输类型。

0: HS

1: LP

(R/W)

DSI_HOST_DCS_LW_TX 配置 DCS 长写数据包命令传输类型。

0: HS

1: LP

(R/W)

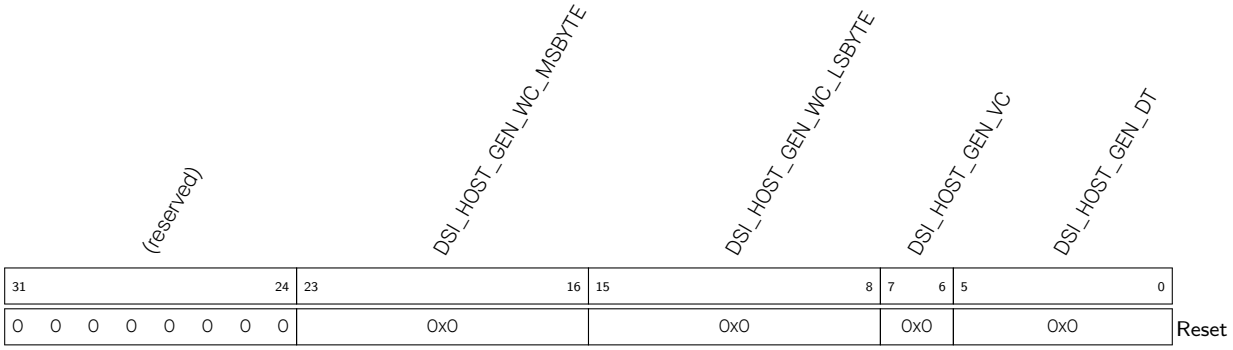
DSI_HOST_MAX_RD_PKT_SIZE 配置最大读数据包大小命令传输类型。

0: HS

1: LP

(R/W)

Register 43.61. DSI_HOST_GEN_HDR_REG (0x006C)



DSI_HOST_GEN_DT 配置数据包头的数据类型。(R/W)

DSI_HOST_GEN_VC 配置数据包头的虚拟通道 ID。(R/W)

DSI_HOST_GEN_WC_LSBYTE 配置长数据包的字计数字节或短数据包的数据 0（数据包头）。(R/W)

DSI_HOST_GEN_WC_MSBYTE 配置长数据包的字计数最高有效字节或短数据包的数据 1（数据包头）。(R/W)

Register 43.62. DSI_HOST_GEN_PLD_DATA_REG (0x0070)



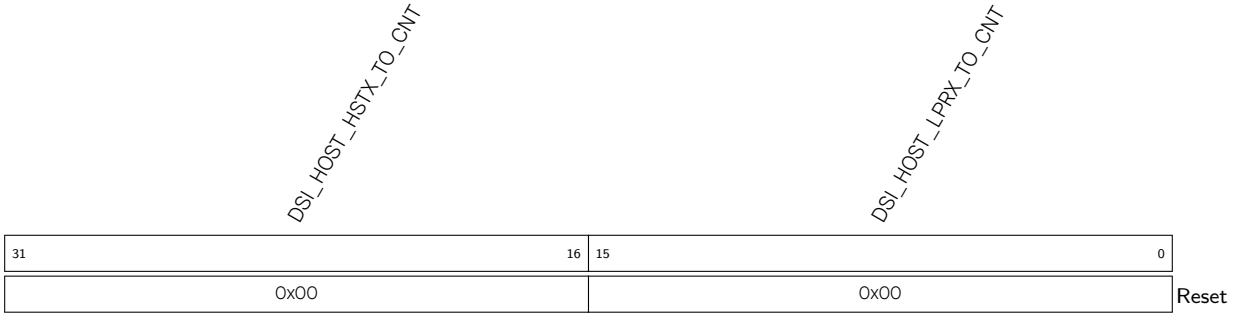
DSI_HOST_GEN_PLD_B1 配置数据包有效载荷的字节 1。(R/W)

DSI_HOST_GEN_PLD_B2 配置数据包有效载荷的字节 2。(R/W)

DSI_HOST_GEN_PLD_B3 配置数据包有效载荷的字节 3。(R/W)

DSI_HOST_GEN_PLD_B4 配置数据包有效载荷的字节 4。(R/W)

Register 43.63. DSI_HOST_TO_CNT_CFG_REG (0x0078)



DSI_HOST_LPRX_TO_CNT 配置触发 LP 接收超时竞争检测的超时计数器（以TO_CLK_DIVISION周期为单位）。(R/W)

DSI_HOST_HSTX_TO_CNT 配置触发 HS 传输超时竞争检测的超时计数器（以 TO_CLK_DIVISION 周期为单位）。

对于非突发模式，如果在 HFP 期间没有足够的时间从 HS 切换到 LP 并返回，DSI 链路每帧返回 LP 一次。配置应满足：

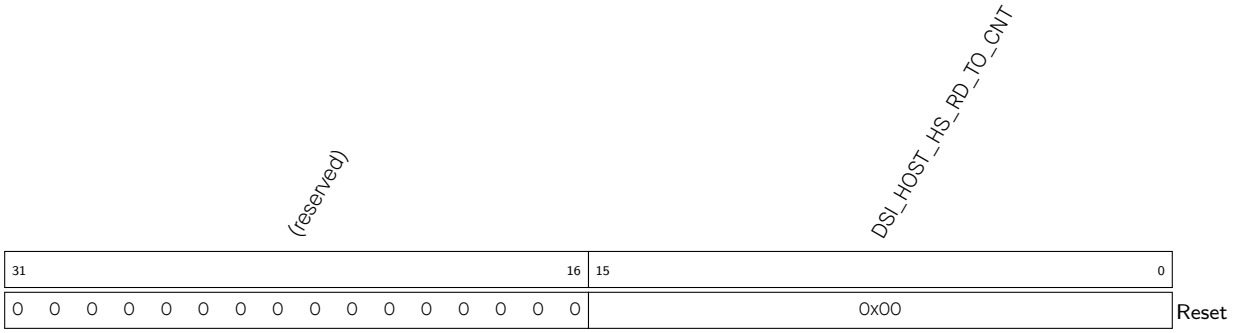
TO_CLK_DIVISION×DSI_HOST_LPRX_TO_CNT×lane_byte_clk_period ≥ 1.1×一帧数据传输时间

对于突发模式，像素数据包被时间压缩，在 HFP 期间留下更多时间从 HS 切换到 LP 并返回。如果选择该模式，配置应满足：

TO_CLK_DIVISION×DSI_HOST_LPRX_TO_CNT×lane_byte_clk_period ≥ 1.1×一行数据传输时间

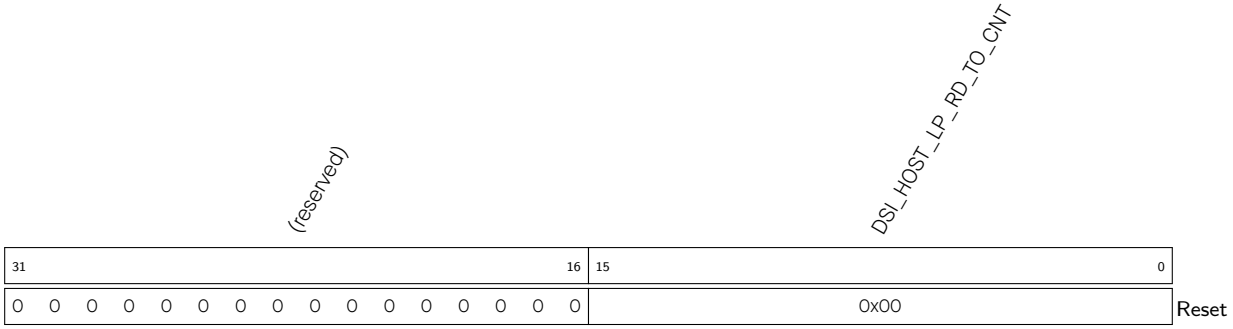
(R/W)

Register 43.64. DSI_HOST_HS_RD_TO_CNT_REG (0x007C)



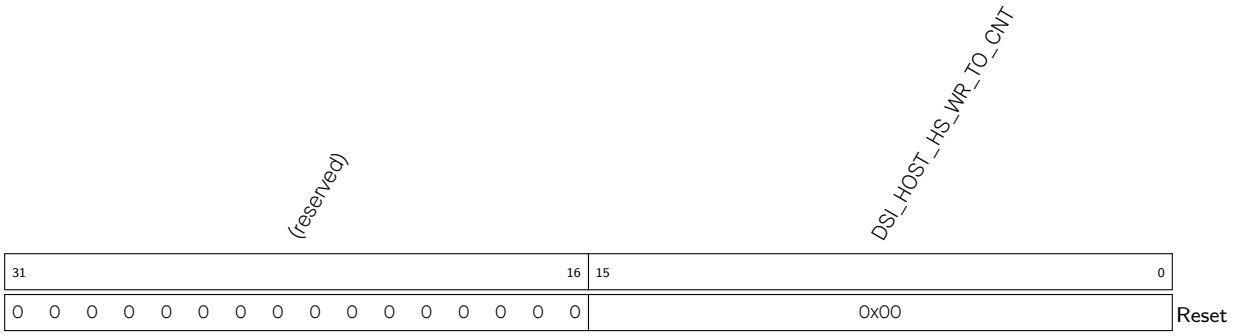
DSI_HOST_HS_RD_TO_CNT 配置 DSI Host 在发送 HS 读后保持链路静止的时间。以通道字节时钟周期为单位。计数从 DPHY 进入停止状态时开始，不会产生中断。(R/W)

Register 43.65. DSI_HOST_LP_RD_TO_CNT_REG (0x0080)



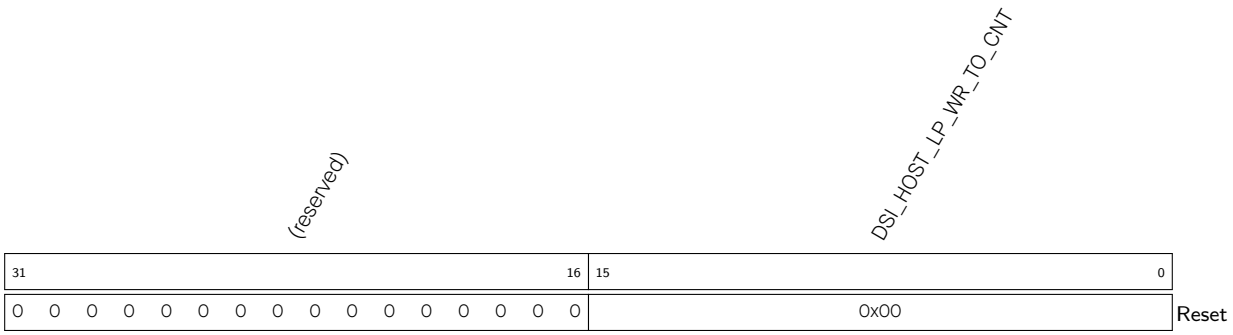
DSI_HOST_LP_RD_TO_CNT 配置 DSI Host 在发送 LP 读后保持链路静止的时间。以通道字节时钟周期为单位。计数从 DPHY 进入停止状态时开始，不会产生中断。(R/W)

Register 43.66. DSI_HOST_HS_WR_TO_CNT_REG (0x0084)



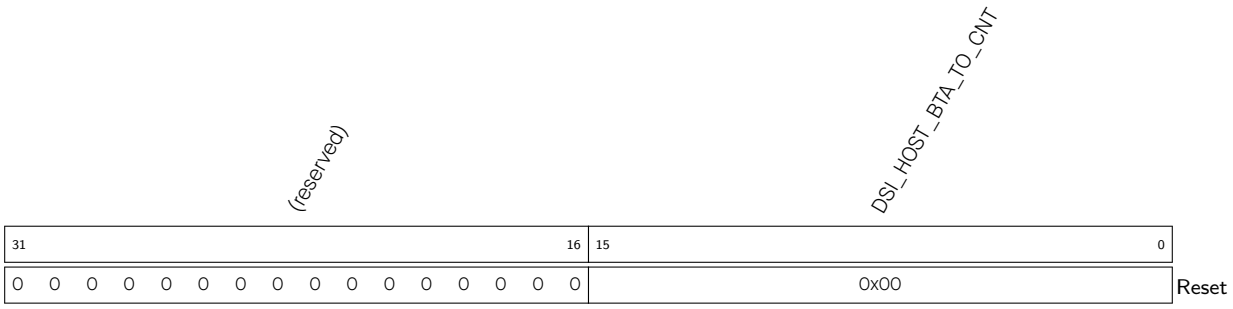
DSI_HOST_HS_WR_TO_CNT 配置 DSI Host 在发送 HS 写后保持链路静止的时间。以通道字节时钟周期为单位。计数从 DPHY 进入停止状态时开始，不会产生中断。(R/W)

Register 43.67. DSI_HOST_LP_WR_TO_CNT_REG (0x0088)



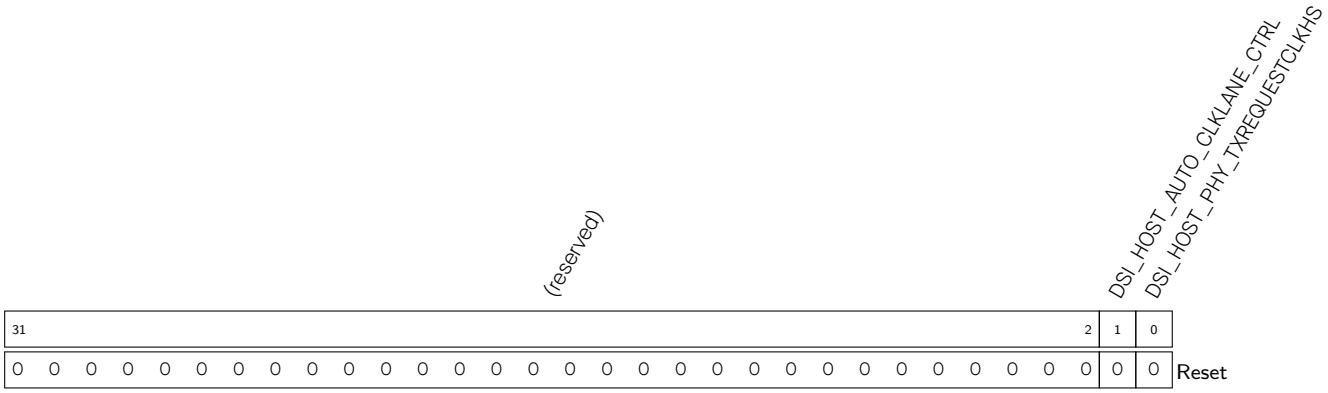
DSI_HOST_LP_WR_TO_CNT 配置 DSI Host 在发送 LP 写后保持链路静止的时间。以通道字节时钟周期为单位。计数从 DPHY 进入停止状态时开始，不会产生中断。(R/W)

Register 43.68. DSI_HOST_BTA_TO_CNT_REG (0x008C)



DSI_HOST_BTA_TO_CNT 配置 DSI Host 在发送 BTA 后保持链路静止的时间。以通道字节时钟周期为单位。计数从 DPHY 进入停止状态时开始，不会产生中断。(R/W)

Register 43.69. DSI_HOST_LPCLK_CTRL_REG (0x0094)



DSI_HOST_PHY_TXREQUESTCLKHS 控制 DPHY PPI txrequestclkhs 信号。(R/W)

DSI_HOST_AUTO_CLKLANE_CTRL 配置是否在时间允许时启用自动机制以停止在时钟通道中提供时钟。(R/W)

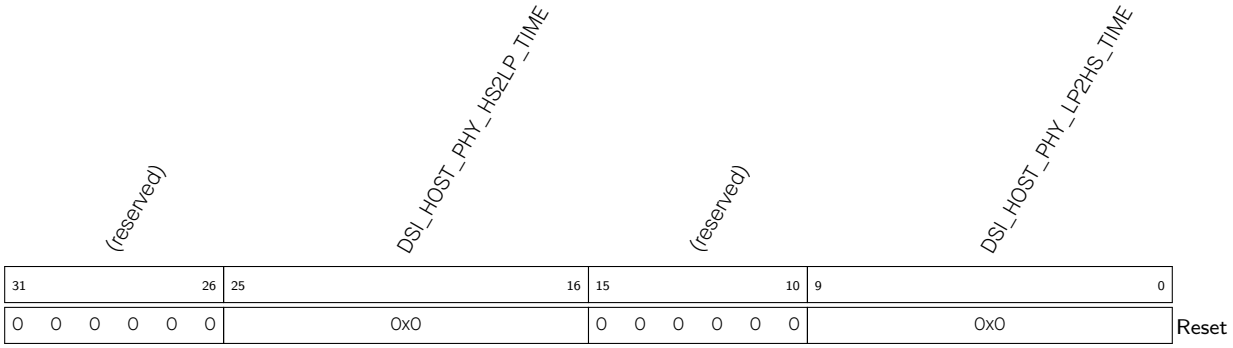
Register 43.70. DSI_HOST_PHY_TMR_LPCLK_CFG_REG (0x0098)



DSI_HOST_PHY_CLKLP2HS_TIME 配置 DPHY 时钟通道从 LP 到 HS 所需的最大时间（以通道字节时钟周期为单位）。(R/W)

DSI_HOST_PHY_CLKHS2LP_TIME 配置 DPHY 时钟通道从 HS 到 LP 所需的最大时间（以通道字节时钟周期为单位）。(R/W)

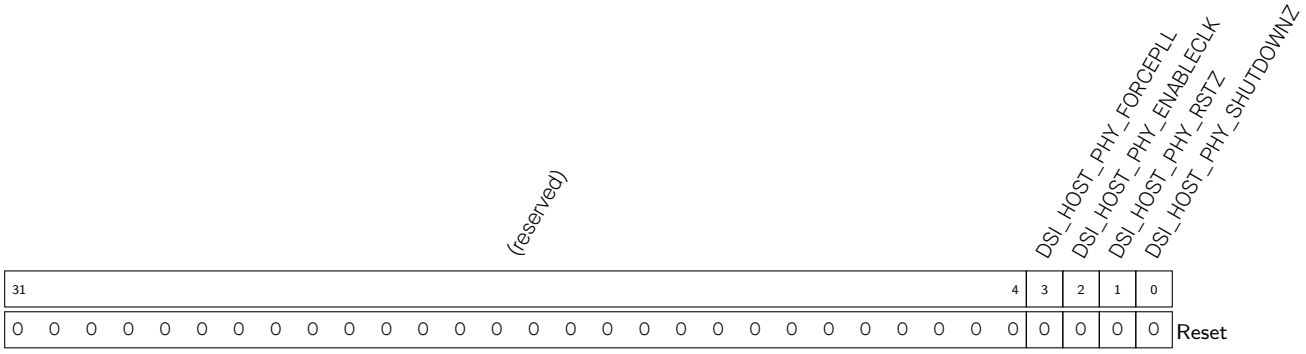
Register 43.71. DSI_HOST_PHY_TMR_CFG_REG (0x009C)



DSI_HOST_PHY_LP2HS_TIME 配置 DPHY 数据通道从 LP 到 HS 所需的最大时间（以通道字节时钟周期为单位）。(R/W)

DSI_HOST_PHY_HS2LP_TIME 配置 DPHY 数据通道从 HS 到 LP 所需的最大时间（以通道字节时钟周期为单位）。(R/W)

Register 43.72. DSI_HOST_PHY_RSTZ_REG (0x00A0)



DSI_HOST_PHY_SHUTDOWNZ 配置 DPHY 宏电源状态。

0: 下电

1: 上电

(R/W)

DSI_HOST_PHY_RSTZ 配置 DPHY 数字逻辑复位。

0: 复位

1: 释放复位

(R/W)

DSI_HOST_PHY_ENABLECLK 配置是否使能 DPHY 时钟通道模块。

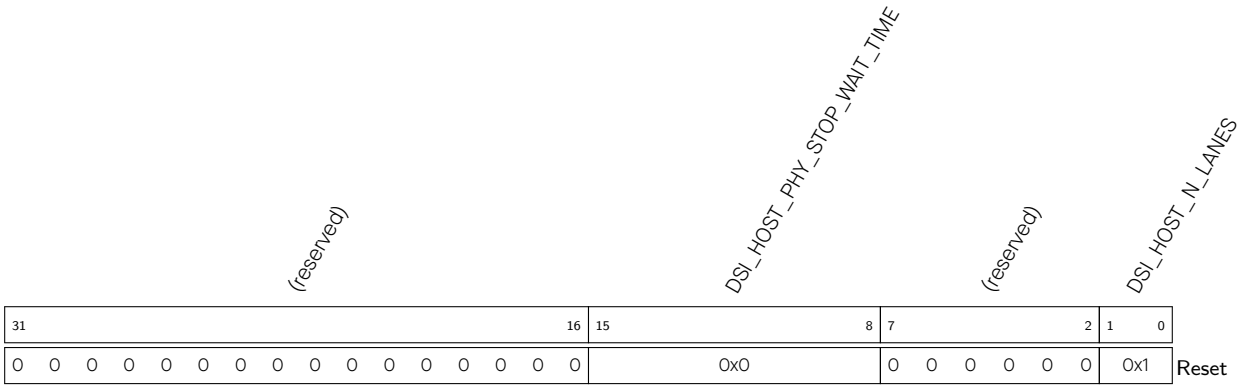
0: 禁用

1: 使能

(R/W)

DSI_HOST_PHY_FORCEPLL 当 DPHY 处于 ULPS 时配置 DPHY PLL。(R/W)

Register 43.73. DSI_HOST_PHY_IF_CFG_REG (0x00A4)



DSI_HOST_N_LANES 配置活动数据通道数。

0: 通道 0

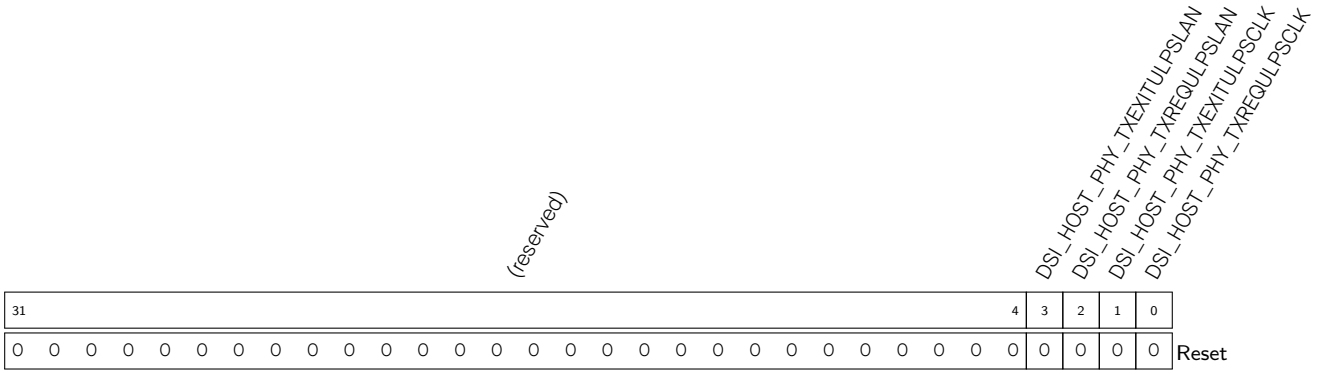
1: 通道 0 和 1

others: 保留

(R/W)

DSI_HOST_PHY_STOP_WAIT_TIME 配置在请求进入 HS 传输前 PHY 所需保持在 StopState 的最小时间 (R/W)

Register 43.74. DSI_HOST_PHY_ULPS_CTRL_REG (0x00A8)



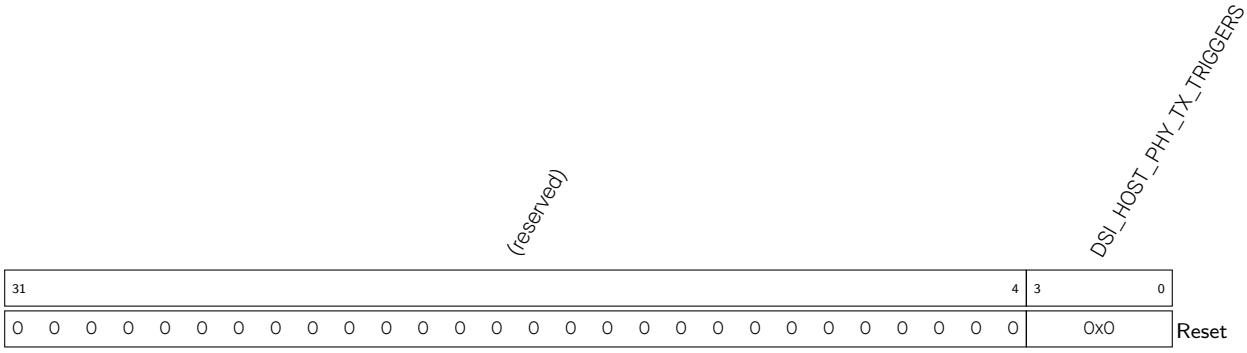
DSI_HOST_PHY_TXREQULPSCLK 控制时钟通道上的 ULPS 模式请求。(R/W)

DSI_HOST_PHY_TXEXITULPSCLK 控制时钟通道上的 ULPS 模式退出。(R/W)

DSI_HOST_PHY_TXREQULPSLAN 控制所有活动数据通道上的 ULPS 模式请求。(R/W)

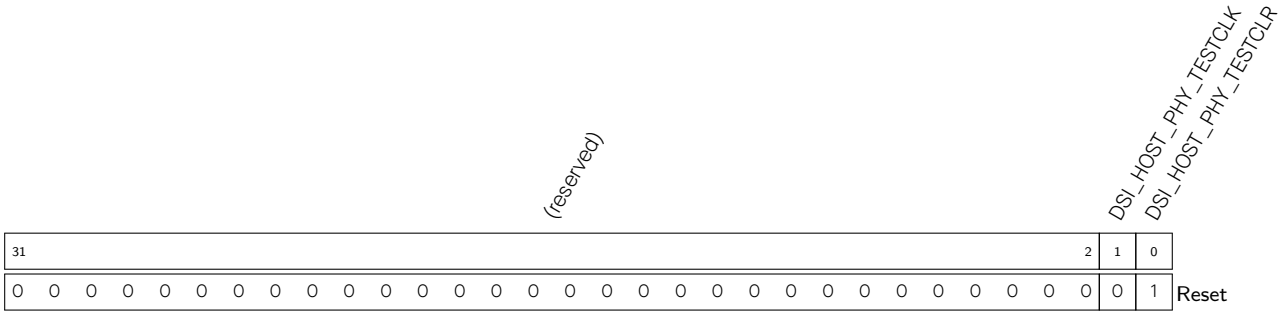
DSI_HOST_PHY_TXEXITULPSLAN 控制所有活动数据通道上的 ULPS 模式退出。(R/W)

Register 43.75. DSI_HOST_PHY_TX_TRIGGERS_REG (0x00AC)



DSI_HOST_PHY_TX_TRIGGERS 控制触发传输。(R/W)

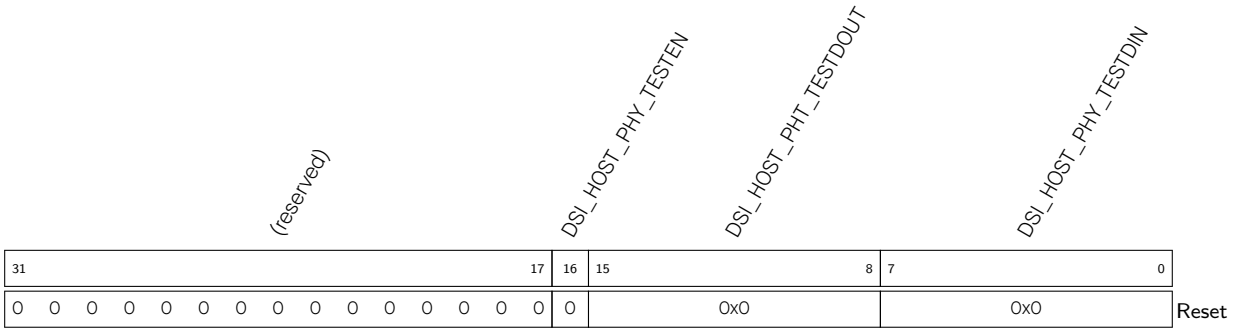
Register 43.76. DSI_HOST_PHY_TST_CTRL0_REG (0x00B4)



DSI_HOST_PHY_TESTCLR 控制测试代码接口 TESTCLR。(R/W)

DSI_HOST_PHY_TESTCLK 控制测试代码接口 TESTCLK。(R/W)

Register 43.77. DSI_HOST_PHY_TST_CTRL1_REG (0x00B8)

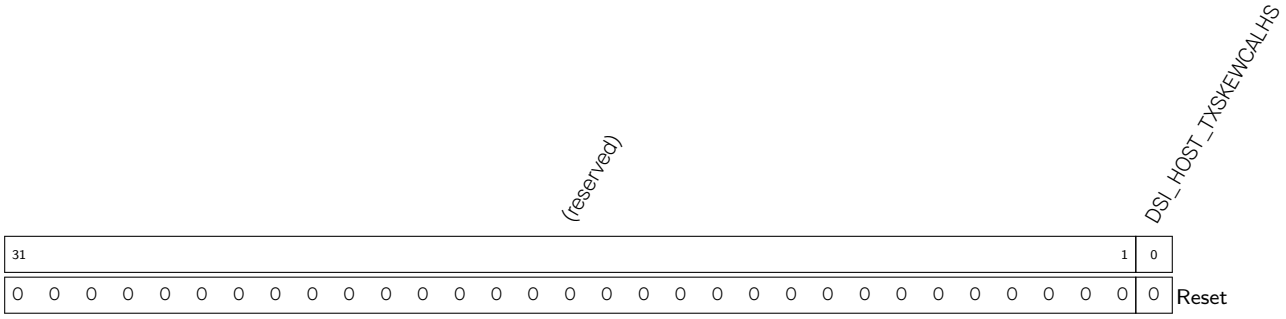


DSI_HOST_PHY_TESTDIN 控制测试代码接口 TESTDIN。(R/W)

DSI_HOST_PHT_TESTDOUT 表示测试代码接口 TESTDOUT。(RO)

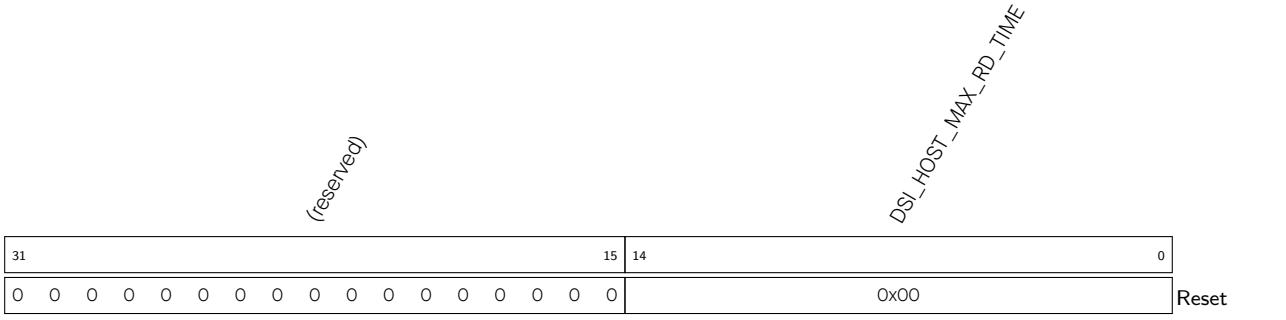
DSI_HOST_PHY_TESTEN 控制测试代码接口 TESTEN。(R/W)

Register 43.78. DSI_HOST_PHY_CAL_REG (0x00CC)



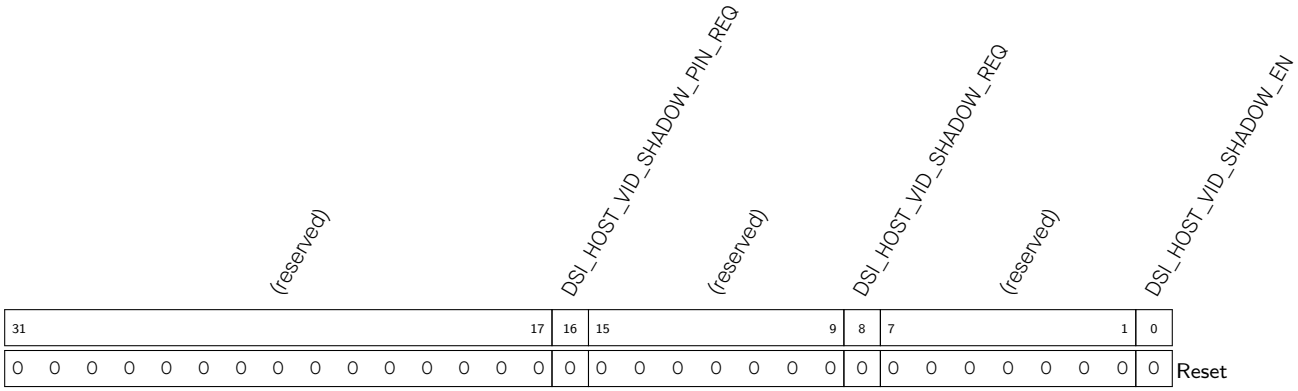
DSI_HOST_TXSKEWCALHS 配置 HS 偏斜校准（假设 DPHY 处于停止状态）。(R/W)

Register 43.79. DSI_HOST_PHY_TMR_RD_CFG_REG (0x00F4)



DSI_HOST_MAX_RD_TIME 配置执行读命令所需的最大时间（以通道字节时钟周期为单位）。此寄存器只能在没有任何读命令进行时修改。(R/W)

Register 43.80. DSI_HOST_VID_SHADOW_CTRL_REG (0x0100)

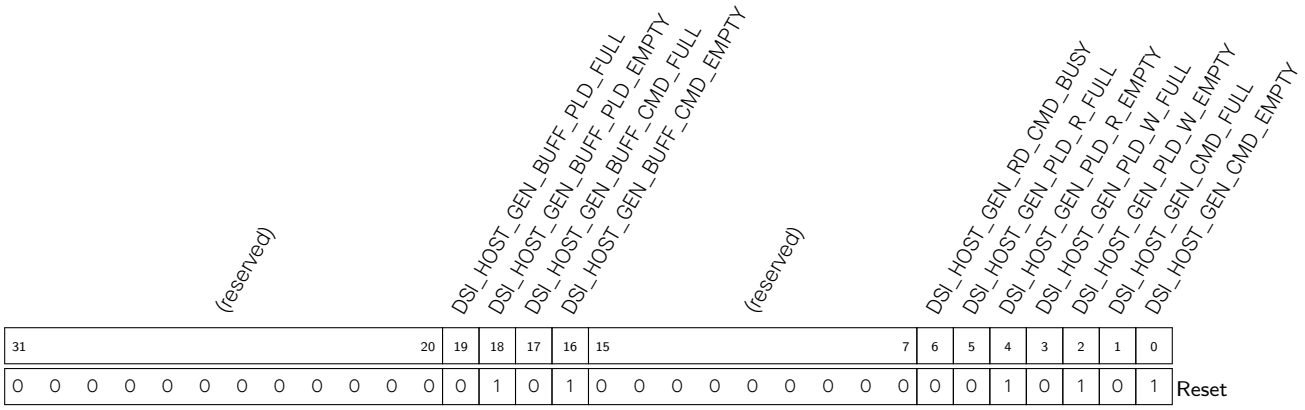


DSI_HOST_VID_SHADOW_EN 控制 DPI 是否从影子寄存器接收活动配置。(R/W)

DSI_HOST_VID_SHADOW_REQ 控制从寄存器组更新 DPI 寄存器到影子寄存器。请求完成后自动清除。(R/W)

DSI_HOST_VID_SHADOW_PIN_REQ 控制视频请求由外部引脚完成。设置为 1 时，忽略 DSI_HOST_VID_SHADOW_REQ。(R/W)

Register 43.81. DSI_HOST_CMD_PKT_STATUS_REG (0x0074)



DSI_HOST_GEN_CMD_EMPTY 表示通用命令 FIFO 的空状态。(RO)

DSI_HOST_GEN_CMD_FULL 表示通用命令 FIFO 的满状态。(RO)

DSI_HOST_GEN_PLD_W_EMPTY 表示通用写有效载荷 FIFO 的空状态。(RO)

DSI_HOST_GEN_PLD_W_FULL 表示通用写有效载荷 FIFO 的满状态。(RO)

DSI_HOST_GEN_PLD_R_EMPTY 表示通用读有效载荷 FIFO 的空状态。(RO)

DSI_HOST_GEN_PLD_R_FULL 表示通用读有效载荷 FIFO 的满状态。(RO)

DSI_HOST_GEN_RD_CMD_BUSY 表示已发出读命令，当整个响应存储在通用接口的 FIFO 中时清除。(RO)

DSI_HOST_GEN_BUFF_CMD_EMPTY 表示通用命令内部缓冲区的空状态。(RO)

DSI_HOST_GEN_BUFF_CMD_FULL 表示通用命令内部缓冲区的满状态。(RO)

DSI_HOST_GEN_BUFF_PLD_EMPTY 表示通用有效载荷内部缓冲区的空状态。(RO)

DSI_HOST_GEN_BUFF_PLD_FULL 表示通用有效载荷内部缓冲区的满状态。(RO)

Register 43.82. DSI_HOST_PHY_STATUS_REG (0x00B0)

Bits	31–9	8	7	6	5	4	3	2	1	0
Field	(reserved)	DSI_HOST_PHY_ULPSACTIVENOT1LANE	DSI_HOST_PHY_STOPSTATE1LANE	DSI_HOST_PHY_RXULPSESC0LANE	DSI_HOST_PHY_ULPSACTIVENOT0LANE	DSI_HOST_PHY_STOPSTATE0LANE	DSI_HOST_PHY_ULPSACTIVENOTCLK	DSI_HOST_PHY_STOPSTATECLKLANE	DSI_HOST_PHY_DIRECTION	DSI_HOST_PHY_LOCK
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	1	0	0	0	0	0	0

DSI_HOST_PHY_LOCK 表示 DPHY phylock 的状态。(RO)

DSI_HOST_PHY_DIRECTION 表示 phydirection 的状态。(RO)

DSI_HOST_PHY_STOPSTATECLKLANE 表示 DPHY phystopstateclklane 的状态。(RO)

DSI_HOST_PHY_ULPSACTIVENOTCLK 表示 DPHY phyulpsactivenotclk 的状态。(RO)

DSI_HOST_PHY_STOPSTATE0LANE 表示 DPHY phystopstate0lane 的状态。(RO)

DSI_HOST_PHY_ULPSACTIVENOT0LANE 表示 DPHY ulpsactivenot0lane 的状态。(RO)

DSI_HOST_PHY_RXULPSESC0LANE 表示 DPHY rxulpsesc0lane 的状态。(RO)

DSI_HOST_PHY_STOPSTATE1LANE 表示 DPHY phystopstate1lane 的状态。(RO)

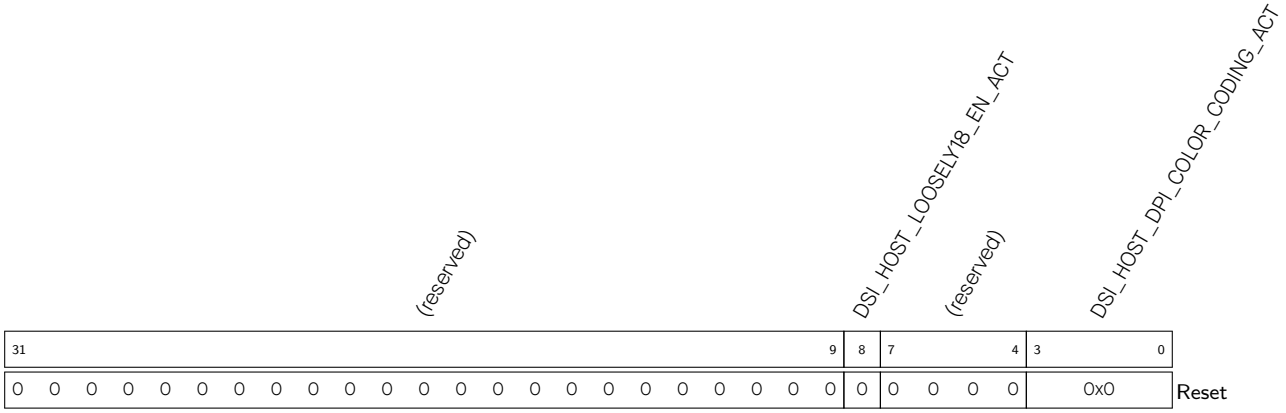
DSI_HOST_PHY_ULPSACTIVENOT1LANE 表示 DPHY ulpsactivenot1lane 的状态。(RO)

Register 43.83. DSI_HOST_DPI_VCID_ACT_REG (0x010C)

Bits	31–2	1–0
Field	(reserved)	DSI_HOST_DPI_VCID_ACT
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0

DSI_HOST_DPI_VCID_ACT 表示实际的 DPI 虚拟通道 ID。(RO)

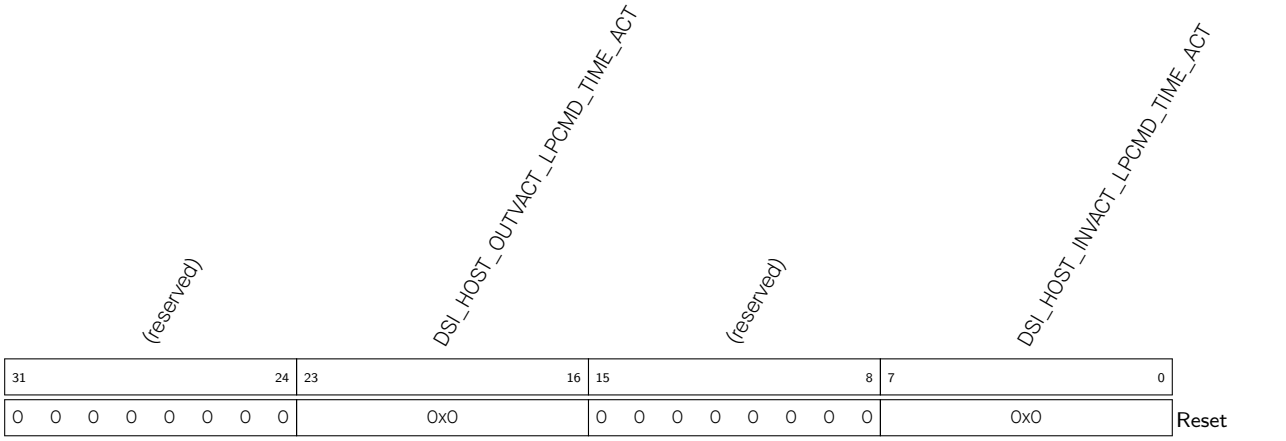
Register 43.84. DSI_HOST_DPI_COLOR_CODING_ACT_REG (0x0110)



DSI_HOST_DPI_COLOR_CODING_ACT 表示实际的 DPI 颜色编码。(RO)

DSI_HOST_LOOSELY18_EN_ACT 表示 18 位松散打包模式在硬件中的实际生效状态。(RO)

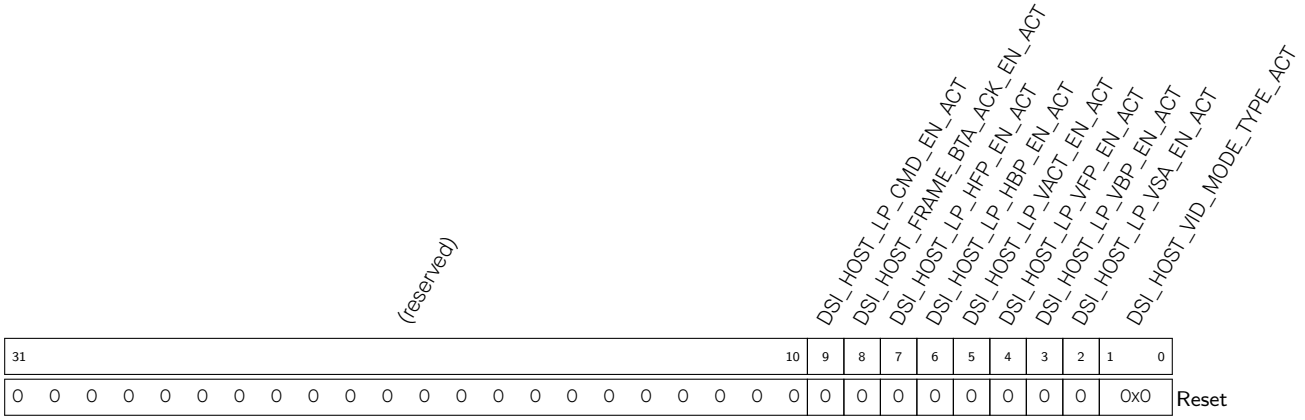
Register 43.85. DSI_HOST_DPI_LP_CMD_TIM_ACT_REG (0x0118)



DSI_HOST_INVACT_LPCMD_TIME_ACT 表示 INVACT_LPCMD_TIME 的实际值。(RO)

DSI_HOST_OUTVACT_LPCMD_TIME_ACT 表示 OUT_VACT_LPCMD_TIME 的实际值。(RO)

Register 43.86. DSI_HOST_VID_MODE_CFG_ACT_REG (0x0138)



DSI_HOST_VID_MODE_TYPE_ACT 表示 VID_MODE_TYPE 的实际值。(RO)

DSI_HOST_LP_VSA_EN_ACT 表示 LP_VSA_EN 的实际值。(RO)

DSI_HOST_LP_VBP_EN_ACT 表示 LP_VBP_EN 的实际值。(RO)

DSI_HOST_LP_VFP_EN_ACT 表示 LP_VFP_EN 的实际值。(RO)

DSI_HOST_LP_VACT_EN_ACT 表示 LP_VACT_EN 的实际值。(RO)

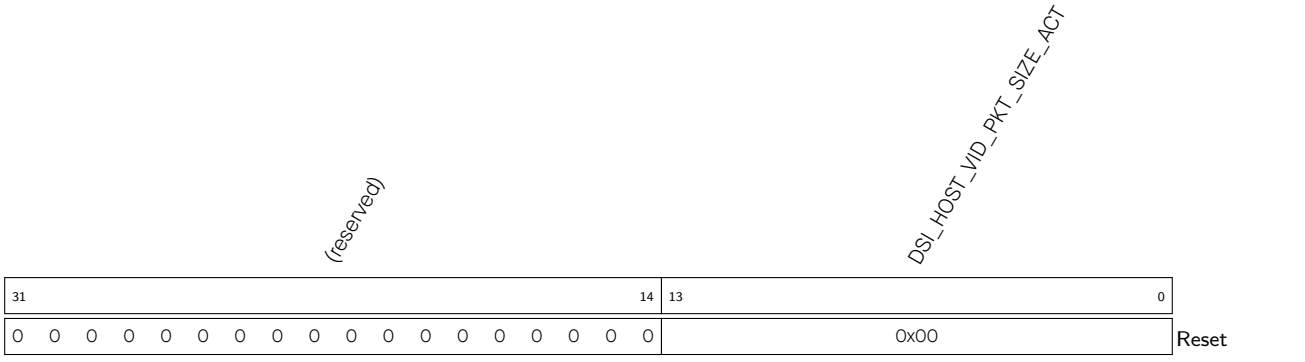
DSI_HOST_LP_HBP_EN_ACT 表示 LP_HBP_EN 的实际值。(RO)

DSI_HOST_LP_HFP_EN_ACT 表示 LP_HFP_EN 的实际值。(RO)

DSI_HOST_FRAME_BTA_ACK_EN_ACT 表示 FRAME_BTA_ACK_EN 的实际值。(RO)

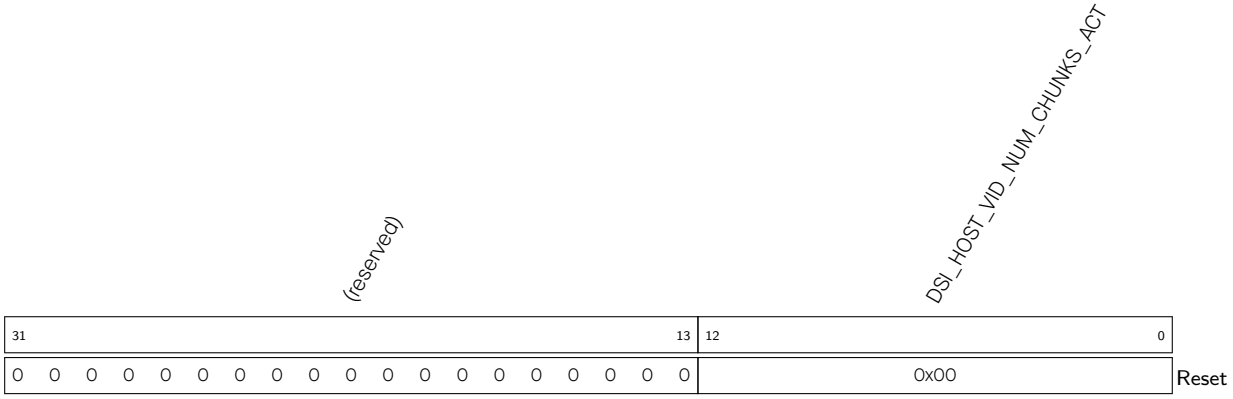
DSI_HOST_LP_CMD_EN_ACT 表示 LP_CMD_EN 的实际值。(RO)

Register 43.87. DSI_HOST_VID_PKT_SIZE_ACT_REG (0x013C)



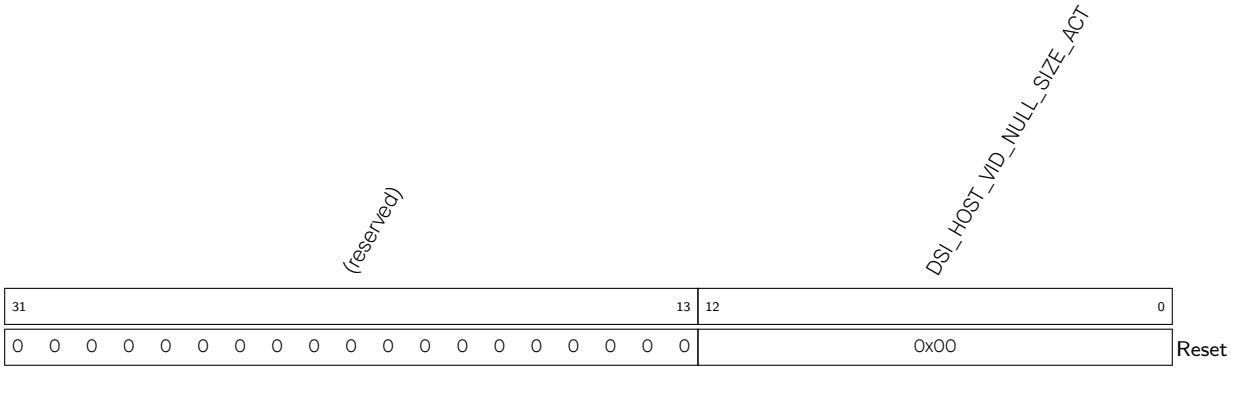
DSI_HOST_VID_PKT_SIZE_ACT 表示 VID_PKT_SIZE 的实际值。(RO)

Register 43.88. DSI_HOST_VID_NUM_CHUNKS_ACT_REG (0x0140)



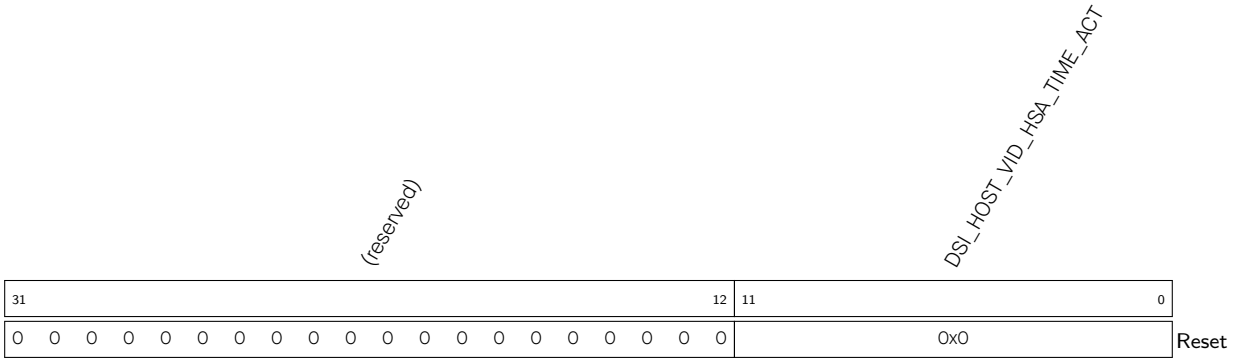
DSI_HOST_VID_NUM_CHUNKS_ACT 表示 VID_NUM_CHUNKS_ACT 的实际值。(RO)

Register 43.89. DSI_HOST_VID_NULL_SIZE_ACT_REG (0x0144)



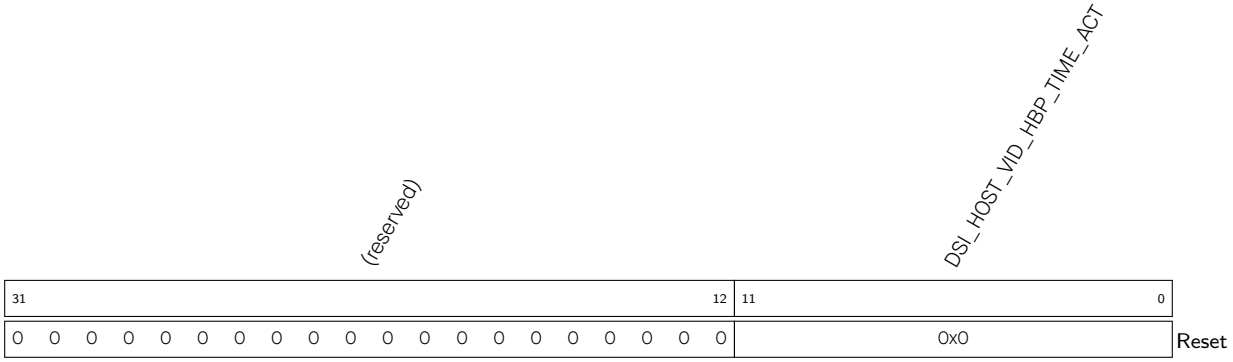
DSI_HOST_VID_NULL_SIZE_ACT 表示 VID_NULL_SIZE 的实际值。(RO)

Register 43.90. DSI_HOST_VID_HSA_TIME_ACT_REG (0x0148)



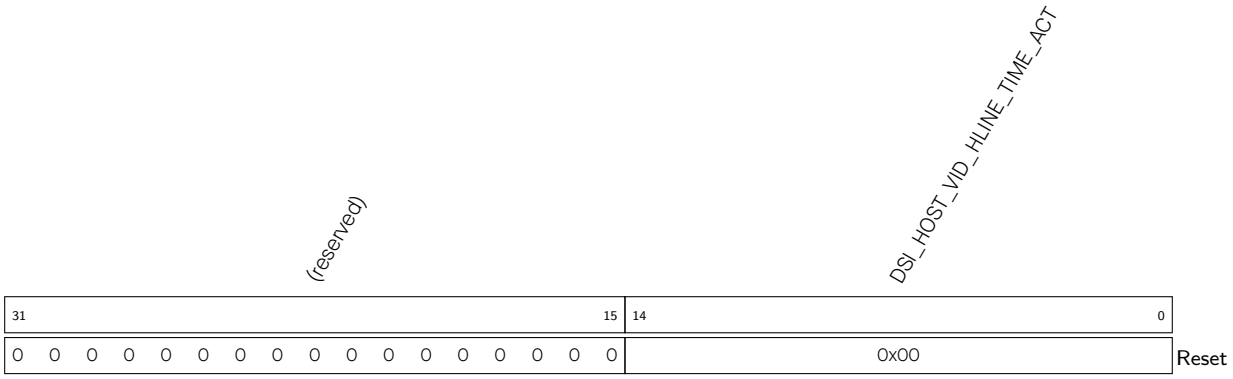
DSI_HOST_VID_HSA_TIME_ACT 表示 VID_HSA_TIME 的实际值。(RO)

Register 43.91. DSI_HOST_VID_HBP_TIME_ACT_REG (0x014C)



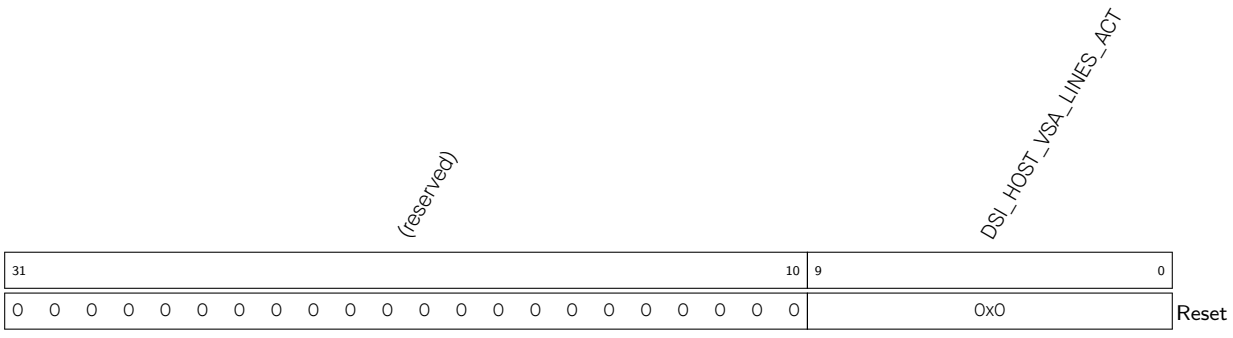
DSI_HOST_VID_HBP_TIME_ACT 表示 VID_HBP_TIME 的实际值。(RO)

Register 43.92. DSI_HOST_VID_HLINE_TIME_ACT_REG (0x0150)



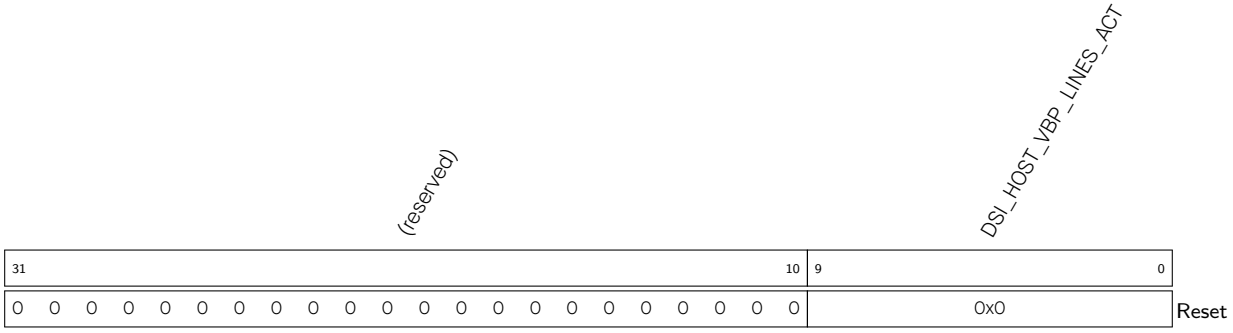
DSI_HOST_VID_HLINE_TIME_ACT 表示 VID_HLINE_TIME 的实际值。(RO)

Register 43.93. DSI_HOST_VID_VSA_LINES_ACT_REG (0x0154)



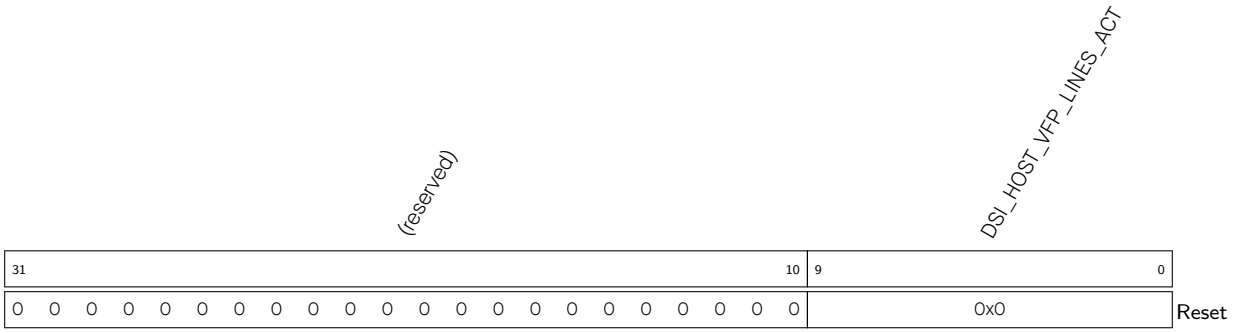
DSI_HOST_VSA_LINES_ACT 表示 VID_VSA_TIME 的实际值。(RO)

Register 43.94. DSI_HOST_VID_VBP_LINES_ACT_REG (0x0158)



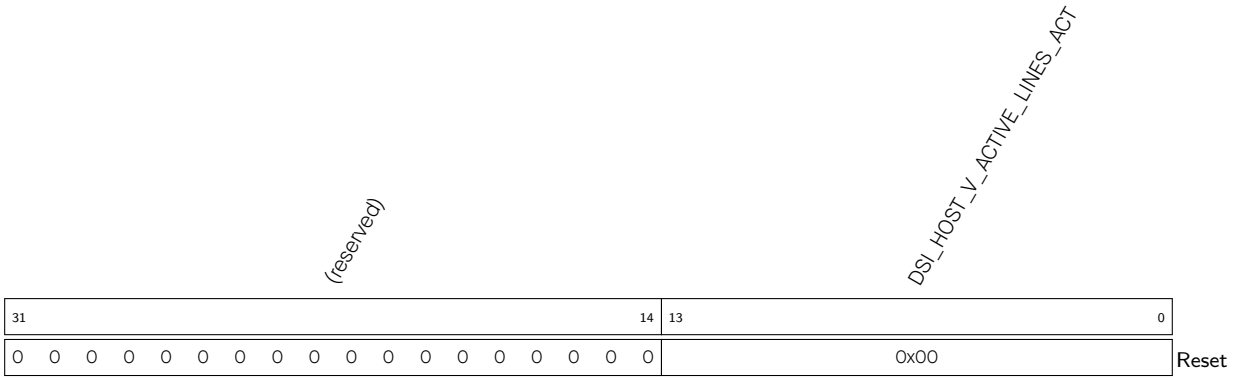
DSI_HOST_VBP_LINES_ACT 表示 VID_VBP_TIME 的实际值。(RO)

Register 43.95. DSI_HOST_VID_VFP_LINES_ACT_REG (0x015C)



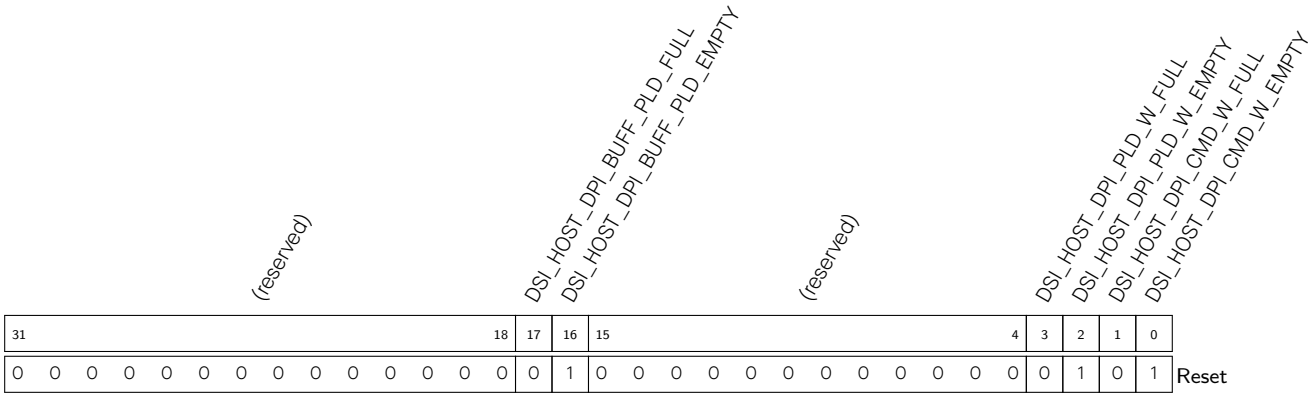
DSI_HOST_VFP_LINES_ACT 表示 VID_VFP_TIME 的实际值。(RO)

Register 43.96. DSI_HOST_VID_VACTIVE_LINES_ACT_REG (0x0160)



DSI_HOST_V_ACTIVE_LINES_ACT 表示 VID_VACTIVE_TIME 的实际值。(RO)

Register 43.97. DSI_HOST_VID_PKT_STATUS_REG (0x0168)



DSI_HOST_DPI_CMD_W_EMPTY 表示视频模式写命令 FIFO 的空状态。(RO)

DSI_HOST_DPI_CMD_W_FULL 表示视频模式写命令 FIFO 的满状态。(RO)

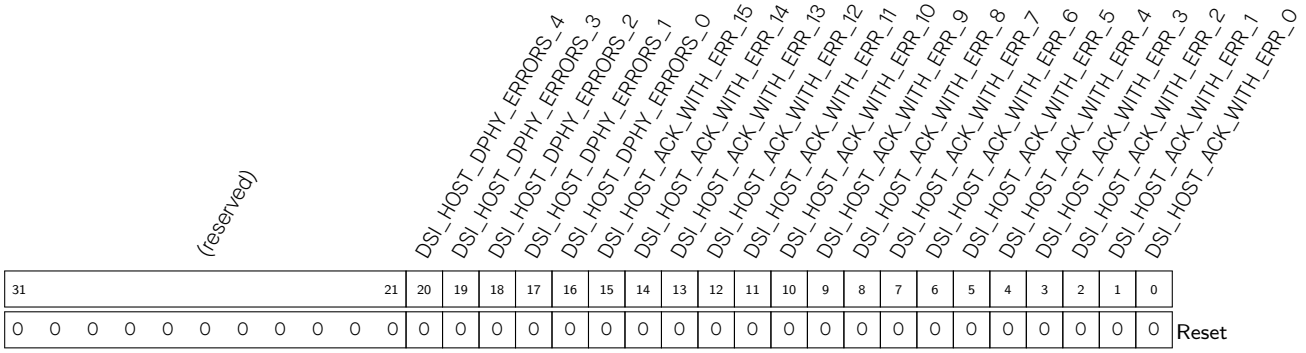
DSI_HOST_DPI_PLD_W_EMPTY 表示视频模式写有效载荷 FIFO 的空状态。(RO)

DSI_HOST_DPI_PLD_W_FULL 表示视频模式写有效载荷 FIFO 的满状态。(RO)

DSI_HOST_DPI_BUFF_PLD_EMPTY 表示视频模式有效载荷内部缓冲区的空状态。(RO)

DSI_HOST_DPI_BUFF_PLD_FULL 表示视频模式有效载荷内部缓冲区的满状态。(RO)

Register 43.98. DSI_HOST_INT_ST0_REG (0x00BC)



DSI_HOST_ACK_WITH_ERR_0 表示来自确认报告的 SoT 错误。(RO)

DSI_HOST_ACK_WITH_ERR_1 表示来自确认报告的 SoT 同步错误。(RO)

DSI_HOST_ACK_WITH_ERR_2 表示来自确认报告的 EoT 错误。(RO)

DSI_HOST_ACK_WITH_ERR_3 表示来自确认报告的转义模式进入命令错误。(RO)

DSI_HOST_ACK_WITH_ERR_4 表示来自确认报告的 LP 传输同步错误。(RO)

DSI_HOST_ACK_WITH_ERR_5 表示来自确认报告的外设超时错误。(RO)

DSI_HOST_ACK_WITH_ERR_6 表示来自确认报告的虚假控制错误。(RO)

DSI_HOST_ACK_WITH_ERR_7 表示来自确认报告的保留字段。(RO)

DSI_HOST_ACK_WITH_ERR_8 表示来自确认报告的 ECC 单比特错误。(RO)

DSI_HOST_ACK_WITH_ERR_9 表示来自确认报告的 ECC 多比特错误。(RO)

DSI_HOST_ACK_WITH_ERR_10 表示来自确认报告的校验和错误（仅长数据包）。(RO)

DSI_HOST_ACK_WITH_ERR_11 表示来自确认报告的未识别的 DSI 数据类型错误。(RO)

DSI_HOST_ACK_WITH_ERR_12 表示来自确认报告的无效 DSI VCID 错误。(RO)

DSI_HOST_ACK_WITH_ERR_13 表示来自确认报告的无效传输长度错误。(RO)

DSI_HOST_ACK_WITH_ERR_14 表示来自确认报告的保留字段。(RO)

DSI_HOST_ACK_WITH_ERR_15 表示来自确认报告的 DSI 协议违反错误。(RO)

DSI_HOST_DPHY_ERRORS_0 表示来自通道 0 的 ErrEsc 转义进入错误。(RO)

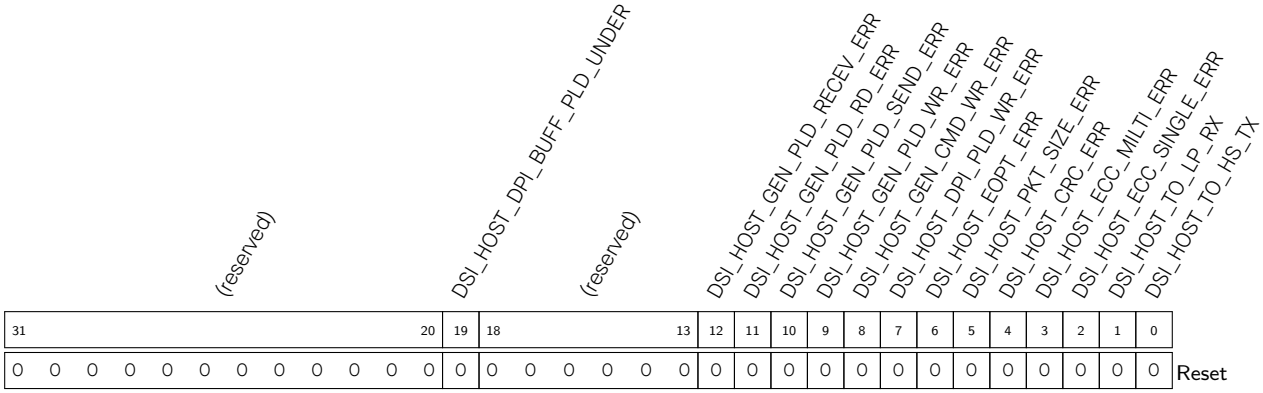
DSI_HOST_DPHY_ERRORS_1 表示来自通道 0 的 ErrSyncEsc LP 数据传输同步错误。(RO)

DSI_HOST_DPHY_ERRORS_2 表示来自通道 0 的 ErrControl 控制错误。(RO)

DSI_HOST_DPHY_ERRORS_3 表示来自通道 0 的 ErrContentionLP0 LP0 竞争错误。(RO)

DSI_HOST_DPHY_ERRORS_4 表示来自通道 0 的 ErrContentionLP1 LP1 竞争错误。(RO)

Register 43.99. DSI_HOST_INT_ST1_REG (0x00C0)



DSI_HOST_TO_HS_TX 表示 HS 传输超时计数器已达到结束且已检测到竞争。(RO)

DSI_HOST_TO_LP_RX 表示 LP 接收超时计数器已达到结束且已检测到竞争。(RO)

DSI_HOST_ECC_SINGLE_ERR 表示在接收的数据包中已检测到并纠正了 ECC 单比特错误。(RO)

DSI_HOST_ECC_MILTI_ERR 表示在接收的数据包中已检测到 ECC 多比特错误。(RO)

DSI_HOST_CRC_ERR 表示在接收的数据包中已检测到 CRC 错误。(RO)

DSI_HOST_PKT_SIZE_ERR 表示在数据包接收期间已检测到数据包大小错误。(RO)

DSI_HOST_EOPT_ERR 表示在传入外设传输结束时未收到 EoTp 数据包。(RO)

DSI_HOST_DPI_PLD_WR_ERR 表示在 DPI 像素行存储期间有效载荷 FIFO 变满且数据损坏。(RO)

DSI_HOST_GEN_CMD_WR_ERR 表示系统尝试通过通用接口写入命令，但 FIFO 已满。(RO)

DSI_HOST_GEN_PLD_WR_ERR 表示系统尝试通过通用接口写入有效载荷数据，但 FIFO 已满。(RO)

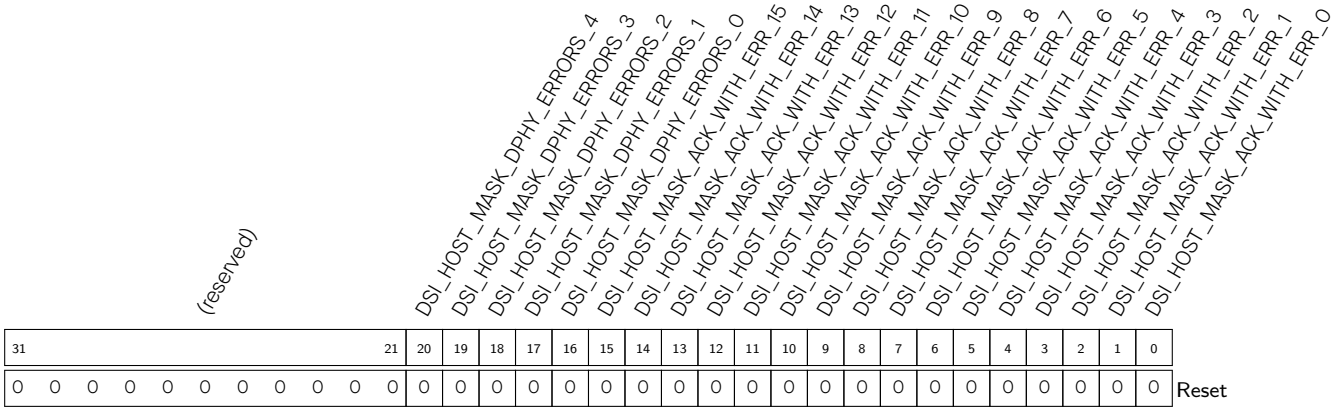
DSI_HOST_GEN_PLD_SEND_ERR 表示在通用接口数据包构建期间有效载荷 FIFO 变空并发送损坏的数据。(RO)

DSI_HOST_GEN_PLD_RD_ERR 表示在 DCS 读数据期间有效载荷 FIFO 变空，发送到接口的数据损坏。(RO)

DSI_HOST_GEN_PLD_RECEV_ERR 表示在通用接口数据包回读期间有效载荷 FIFO 变满，接收的数据损坏。(RO)

DSI_HOST_DPI_BUFF_PLD_UNDER 表示在读取有效载荷以构建视频模式的 DSI 数据包时发生了下溢。(RO)

Register 43.100. DSI_HOST_INT_MSK0_REG (0x00C4)



DSI_HOST_MASK_ACK_WITH_ERR_0 控制 ACK_WITH_ERR_0 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_1 控制 ACK_WITH_ERR_1 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_2 控制 ACK_WITH_ERR_2 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_3 控制 ACK_WITH_ERR_3 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_4 控制 ACK_WITH_ERR_4 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_5 控制 ACK_WITH_ERR_5 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_6 控制 ACK_WITH_ERR_6 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_7 控制 ACK_WITH_ERR_7 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_8 控制 ACK_WITH_ERR_8 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_9 控制 ACK_WITH_ERR_9 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_10 控制 ACK_WITH_ERR_10 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_11 控制 ACK_WITH_ERR_11 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_12 控制 ACK_WITH_ERR_12 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_13 控制 ACK_WITH_ERR_13 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_14 控制 ACK_WITH_ERR_14 的掩码。(R/W)

DSI_HOST_MASK_ACK_WITH_ERR_15 控制 ACK_WITH_ERR_15 的掩码。(R/W)

DSI_HOST_MASK_DPHY_ERRORS_0 控制 DPHY_ERRORS_0 的掩码。(R/W)

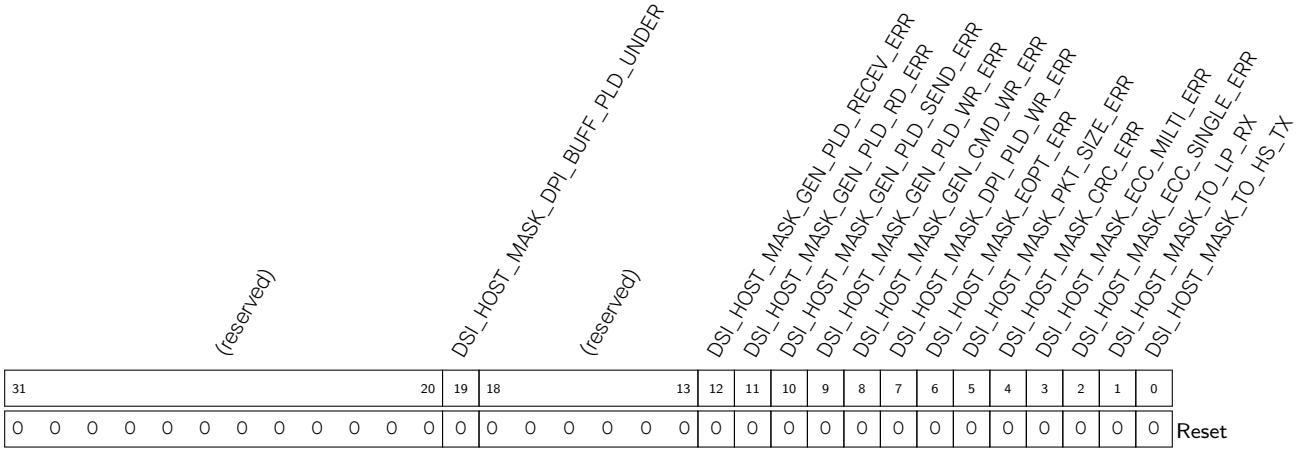
DSI_HOST_MASK_DPHY_ERRORS_1 控制 DPHY_ERRORS_1 的掩码。(R/W)

DSI_HOST_MASK_DPHY_ERRORS_2 控制 DPHY_ERRORS_2 的掩码。(R/W)

DSI_HOST_MASK_DPHY_ERRORS_3 控制 DPHY_ERRORS_3 的掩码。(R/W)

DSI_HOST_MASK_DPHY_ERRORS_4 控制 DPHY_ERRORS_4 的掩码。(R/W)

Register 43.101. DSI_HOST_INT_MSK1_REG (0x00C8)



DSI_HOST_MASK_TO_HS_TX 控制 TO_HS_TX 的掩码。(R/W)

DSI_HOST_MASK_TO_LP_RX 控制 TO_LP_RX 的掩码。(R/W)

DSI_HOST_MASK_ECC_SINGLE_ERR 控制 ECC_SINGLE_ERR 的掩码。(R/W)

DSI_HOST_MASK_ECC_MILTI_ERR 控制 ECC_MULTI_ERR 的掩码。(R/W)

DSI_HOST_MASK_CRC_ERR 控制 CRC_ERR 的掩码。(R/W)

DSI_HOST_MASK_PKT_SIZE_ERR 控制 PKT_SIZE_ERR 的掩码。(R/W)

DSI_HOST_MASK_EOPT_ERR 控制 EOPT_ERR 的掩码。(R/W)

DSI_HOST_MASK_DPI_PLD_WR_ERR 控制 DPI_PLD_WR_ERR 的掩码。(R/W)

DSI_HOST_MASK_GEN_CMD_WR_ERR 控制 GEN_CMD_WR_ERR 的掩码。(R/W)

DSI_HOST_MASK_GEN_PLD_WR_ERR 控制 GEN_PLD_WR_ERR 的掩码。(R/W)

DSI_HOST_MASK_GEN_PLD_SEND_ERR 控制 GEN_PLD_SEND_ERR 的掩码。(R/W)

DSI_HOST_MASK_GEN_PLD_RD_ERR 控制 GEN_PLD_RD_ERR 的掩码。(R/W)

DSI_HOST_MASK_GEN_PLD_RECEV_ERR 控制 GEN_PLD_RECEV_ERR 的掩码。(R/W)

DSI_HOST_MASK_DPI_BUFF_PLD_UNDER 控制 DPI_BUFF_PLD_UNDER 的掩码。(R/W)

第 44 章

语音活动检测 (VAD)

44.1 概述

ESP32-P4 集成了语音活动检测 (VAD) 模块，硬件实现了语音唤醒等多媒体功能的第一阶段算法。此外，它还为低功耗语音唤醒解决方案提供硬件支持。

44.2 主要特性

VAD 模块支持以下功能：

- VAD 算法按帧处理音频数据，每帧 256 个数据点，数据采样率为 8 kHz，位宽 16 位
- 2 KB 缓存，可保留多达 4 帧数据
- 独立的系统唤醒源
- 可配置中断源
- 算法参数可灵活配置

44.3 系统架构

图 44.3-1 为 ESP32-P4 VAD 模块的结构框图。ESP32-P4 VAD 模块包含：

- 能量阈值检测模块
- 快速傅里叶变换 (FFT) 计算模块
- 长时频谱散度 (LTSD) 计算模块

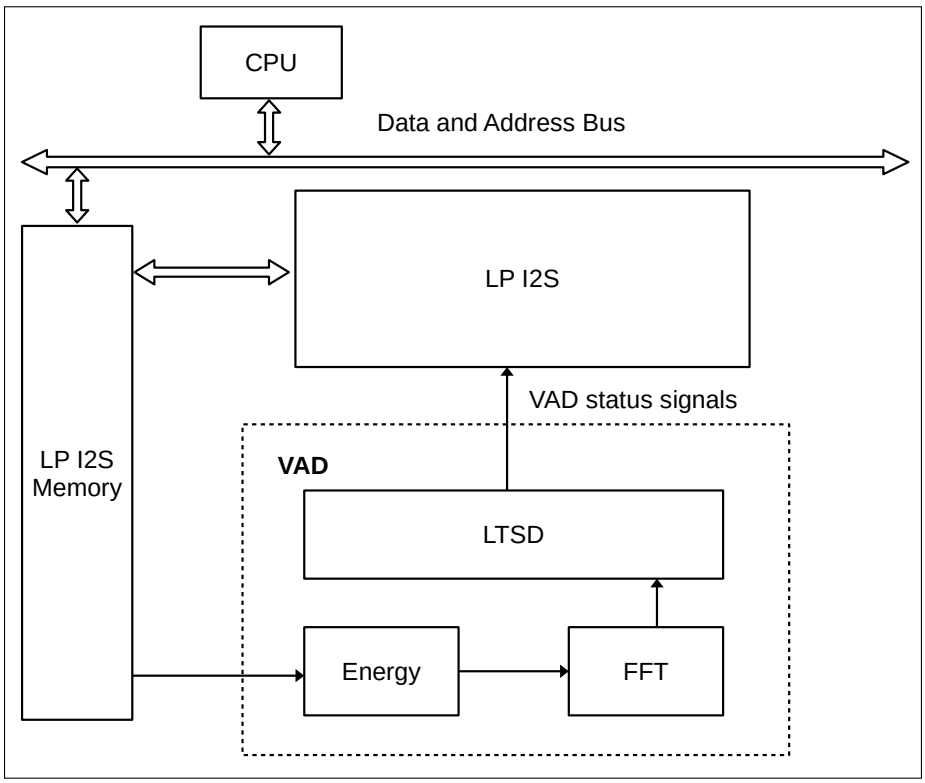


图 44.3-1. ESP32-P4 VAD 系统架构

VAD 模块从 LP I2S memory 读取 8kHz 采样率，16 位宽的音频数据，通过能量检测、FFT、LTSD 三级运算，检测语音活动状态，并将 VAD 状态信号发送给 LP I2S 模块，用以生成对应中断和唤醒信号。

44.4 功能描述

ESP32-P4 VAD 模块的功能灵活可配置。用户可以通过配置算法参数和唤醒源，满足实际场景需求。

44.4.1 算法参数配置

初始帧个数

在 VAD 模块开始运行后，将首先进入初始化阶段，并自动学习噪声参数，初始化阶段的耗时取决于用户配置的初始帧个数。合适的初始帧设置将很好地抑制噪声，显著降低误触发率，而过长的初始帧设置会使得初始帧内的语音活动被遗漏。请根据实际情况，配置合适的初始帧个数。

用户应尽量在初始化阶段使设备处于目标环境噪声中，设定 100 - 200 帧长度的初始帧。

用户可以通过配置 LP_I2S_PARAM_INIT_FRAME_NUM 来配置初始帧个数。

> **说明：**
> 建议在初始化阶段让 VAD 模块学习噪声参数，但 VAD 模块同样会在运行期间不断学习并更新噪声参数。因此，在不关心初始帧中误触发率的场景下，也可以减少初始帧的个数，直接让 VAD 模块进入运行状态。

能量阈值

VAD 模块在语音活动检测前会先判定能量阈值。设定能量阈值可以过滤能量过小的语音活动（例如远处的语音活动），避免触发唤醒。设定合适的能量阈值能够降低误触发率，并节省功耗。

VAD 模块定义语音帧的能量值为一帧数据的平方均值，并实时计算一帧数据的平方和，当该平方和小于能量阈值 ×256 时，将不会继续后续计算。用户可以通过 LP_I2S_PARAM_MIN_ENERGY 来配置能量阈值。

通带能量检查

VAD 通带能量检查判定通带能量在总频域中的占比是否达到要求。在不同的环境中，通带能量检查可能降低误触发率，但也可能增大漏检率。请根据实际环境判断是否需要开启通带能量检查。

用户可以通过 LP_I2S_PARAM_SKIP_BAND_ENERGY 来开启或关闭通带能量检查。

- 0：开启通带能量检查
- 1：关闭通带能量检查

语音活动检测

VAD 模块的语音活动状态分为语音活动状态和非语音活动状态，内部包含语音帧计数器 (speech_count) 和非语音帧计数器 (silent_count) 来控制状态之间的跳转。两种状态的跳转关系如下：

- 非语音活动状态下：
  - 当前帧有语音活动时，语音帧计数器 (speech_count) 加一；
  - 当前帧无语音活动时，语音帧计数器 (speech_count) 减一（到 0 为止）；
  - 当语音帧计数器 (speech_count) 的值大于 LP_I2S_PARAM_HANGOVER_SPEECH 时，VAD 模块进入语音活动状态，同时归零非语音帧计数器 (silent_count)；
  - VAD 模块在非语音活动状态下，会不断学习并更新噪声信息。
- 语音活动状态下：
  - 当前帧有语音活动时，语音帧计数器 (speech_count) 加一，同时非语音帧计数器 (silent_count) 减一（到 0 为止）；
  - 当前帧无语音活动时，非语音帧计数器 (silent_count) 加一，此时：
    * 如果语音帧计数器 (speech_count) 大于 LP_I2S_PARAM_MIN_SPEECH_COUNT，且
      · 非语音帧计数器 (silent_count) 大于 LP_I2S_PARAM_HANGOVER_SILENT，则 VAD 模块跳转进入非语音活动状态，同时清零语音帧计数器 (speech_count) 和非语音帧计数器 (silent_count)；
      · 非语音帧计数器 (silent_count) 小于 LP_I2S_PARAM_HANGOVER_SILENT，则 VAD 模块维持语音活动状态；
    * 如果语音帧计数器 (speech_count) 小于等于 LP_I2S_PARAM_MIN_SPEECH_COUNT，则 VAD 模块直接跳转进入非语音活动状态，同时清零语音帧计数器 (speech_count)；
  - VAD 模块在语音活动状态下，会暂停更新噪声信息，而当语音帧计数器 (speech_count) 大于 LP_I2S_PARAM_MAX_SPEECH_COUNT 时，视为进入了稳定的新环境中，会继续学习并更新噪声信息。

状态寄存器

VAD 模块将一些重要运算变量作为状态寄存器（只读），以方便用户进行调试，状态寄存器包含：

- LP_I2S_VAD_FLAG：表示当前 VAD 模块的语音活动状态。1 代表当前 VAD 模块处于语音活动状态下，0 代表当前 VAD 模块处于非语音活动状态下。
- LP_I2S_ENERGY_ENOUGH：表示当前帧的能量阈值检测是否通过。1 代表当前帧通过了能量阈值检测，0 代表当前帧未通过能量阈值检测。
- LP_I2S_SPEECH_COUNT_OB：表示语音帧计数器 (speech_count) 的值。
- LP_I2S_SILENT_COUNT_OB：表示非语音帧计数器 (silent_count) 的值。

44.4.2 唤醒源配置

当 VAD 模块处于语音活动状态时，会触发唤醒源 LP_I2S_WAKEUP。用户可以通过唤醒源配置（请参考章节 5 低功耗处理器），使 VAD 成为 ESP32-P4 的唤醒源之一。

44.5 中断

ESP32-P4 VAD 与 LP I2S 模块共用一个中断信号 LP_I2S_INTR。VAD 会生成以下两个中断源：

- LP_I2S_VAD_DONE_INT：当 VAD 结束一帧数据的运算时，触发此中断。
- LP_I2S_VAD_RESET_DONE_INT：当 VAD 完成复位操作时，触发此中断。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请参考章节 44.7 寄存器列表。

44.6 配置流程

44.6.1 自动运算模式

1. 根据 44.4.1 小节中的描述配置 VAD 参数。
2. 开启 VAD 自动检测模式，向 LP_I2S_VAD_EN 写 1。
3. 配置 LP I2S 处于 16 位接收模式，并开始接收数据，接收的数据将存入 LP I2S memory。
4. VAD 模块会在每次 LP I2S 接收满一帧数据后，自动对该帧数据进行 VAD 运算。
5. （可选）当 VAD 模块触发唤醒源/中断信号后，LP I2S memory 中已经存储触发 VAD 唤醒的语音帧，和当前语音帧的前 3 帧语音帧。用户可以通过 LP core 读取 LP I2S memory 中的该 4 帧语音数据，以进行后续语音信息处理。关于 LP I2S memory 的大小和数据读取方式，请参考章节 50 *LP I2S 控制器* > 小节 50.8.3 内部存储器。

44.6.2 手动运算模式

1. 根据 44.4.1 小节中的描述配置 VAD 参数。
2. 用户手动向 LP I2S memory 中存储语音数据，存储步骤请参考章节 50 *LP I2S 控制器* > 小节 50.8.3 内部存储器。
3. 存储完语音数据后，向 LP_I2S_VAD_FORCE_START 写 1，VAD 将会运行一帧数据的运算。
4. VAD 结束一帧数据的运算时，将生成 LP_I2S_VAD_DONE_INT 中断。

44.7 寄存器列表

本小节的所有地址均为相对于 VAD 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

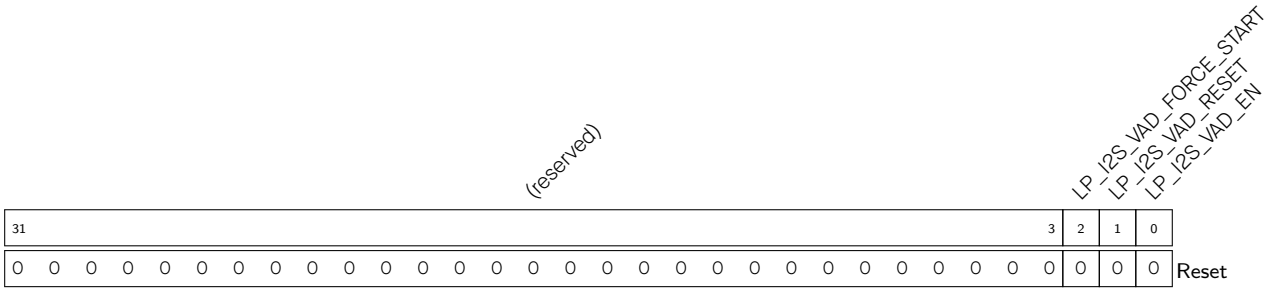
名称	描述	地址	访问
VAD 寄存器			
LP_I2S_VAD_CONF_REG	运算控制寄存器	0x0000	varies
LP_I2S_VAD_RESULT_REG	语音活动状态寄存器	0x0004	RO
LP_I2S_VAD_PARAM0_REG	参数配置寄存器 0	0x0080	R/W
LP_I2S_VAD_PARAM1_REG	参数配置寄存器 1	0x0084	R/W
LP_I2S_VAD_OBO_REG	重要运算变量状态寄存器	0x00B0	RO

44.8 寄存器

本小节的所有地址均为相对于 VAD 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 44.1. LP_I2S_VAD_CONF_REG (0x0000)



LP_I2S_VAD_EN 配置是否开启自动运算模式。

0：关闭

1：开启

(R/W)

LP_I2S_VAD_RESET 配置是否触发复位流程。

0：无效

1：触发

(WT)

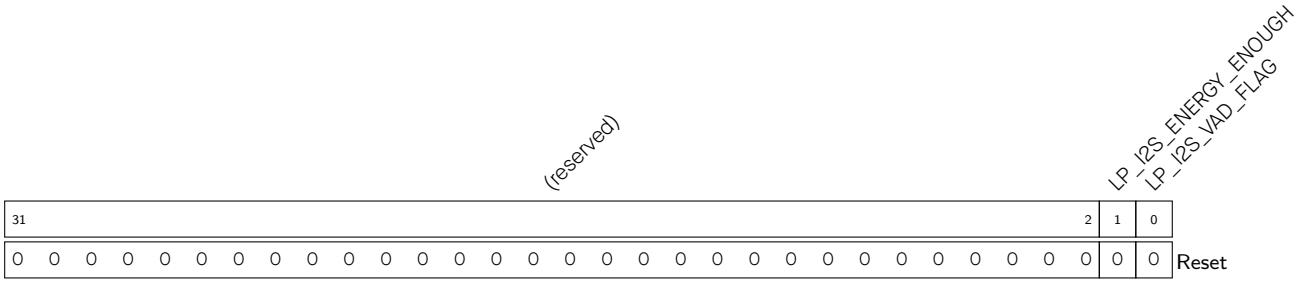
LP_I2S_VAD_FORCE_START 配置是否触发手动运算一帧数据。

0：无效

1：触发

(WT)

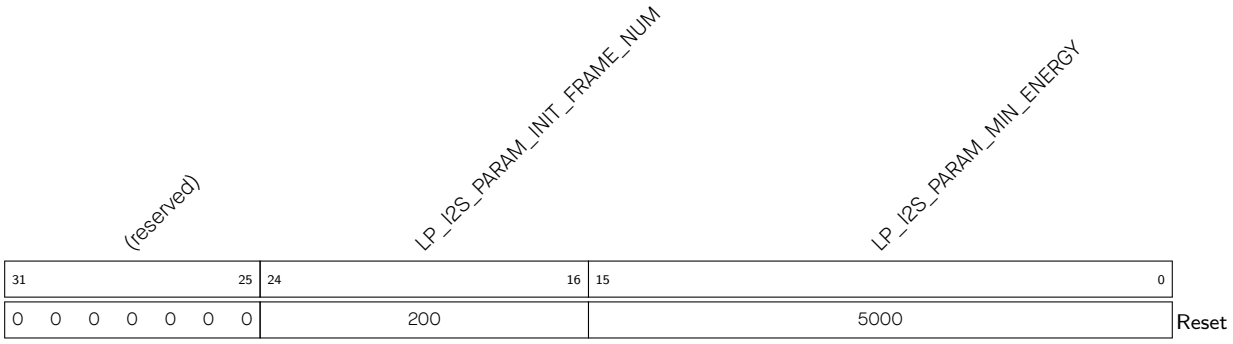
Register 44.2. LP_I2S_VAD_RESULT_REG (0x0004)



LP_I2S_VAD_FLAG 表示语音活动状态。具体请参考章节 44.4.1。(RO)

LP_I2S_ENERGY_ENOUGH 表示当前帧的能量检测是否通过。具体请参考章节 44.4.1。(RO)

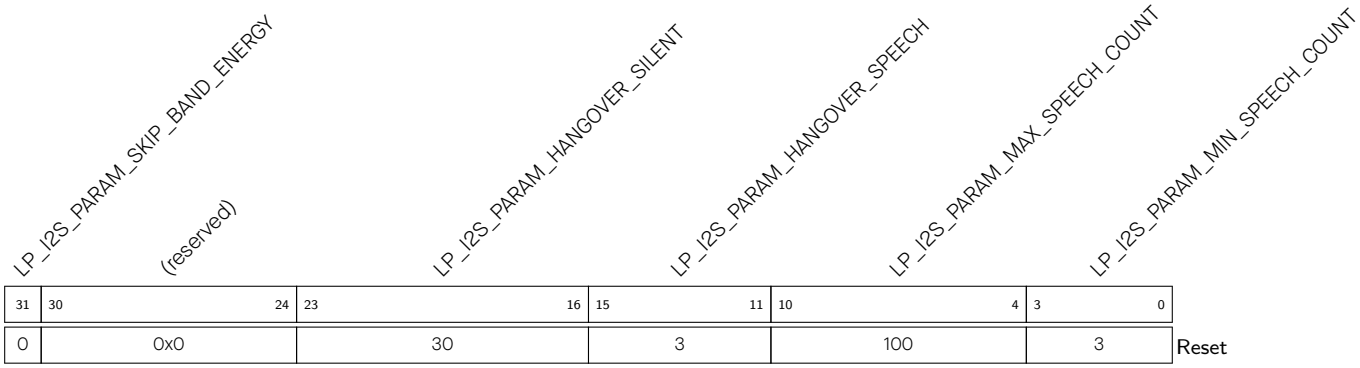
Register 44.3. LP_I2S_VAD_PARAM0_REG (0x0080)



LP_I2S_PARAM_MIN_ENERGY 配置能量阈值。具体请参考章节 44.4.1。(R/W)

LP_I2S_PARAM_INIT_FRAME_NUM 配置初始帧个数。具体请参考章节 44.4.1。(R/W)

Register 44.4. LP_I2S_VAD_PARAM1_REG (0x0084)



LP_I2S_PARAM_MIN_SPEECH_COUNT 配置语音活动状态跳转。具体请参考章节 44.4.1。(R/W)

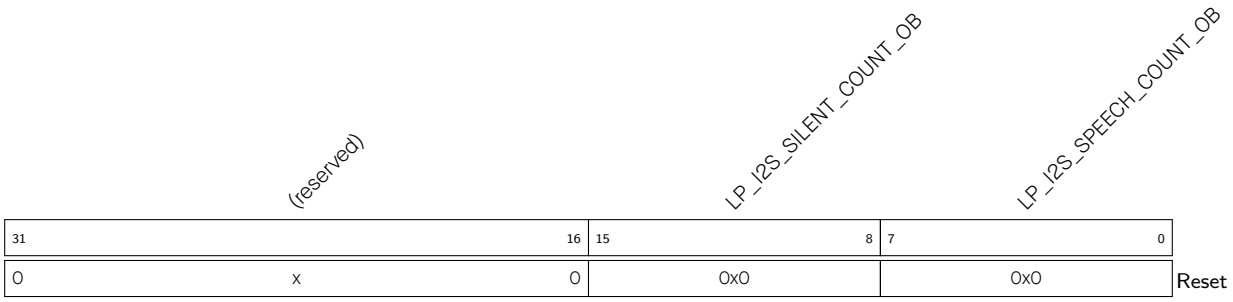
LP_I2S_PARAM_MAX_SPEECH_COUNT 配置语音活动状态跳转。具体请参考章节 44.4.1。(R/W)

LP_I2S_PARAM_HANGOVER_SPEECH 配置语音活动状态跳转。具体请参考章节 44.4.1。(R/W)

LP_I2S_PARAM_HANGOVER_SILENT 配置语音活动状态跳转。具体请参考章节 44.4.1。(R/W)

LP_I2S_PARAM_SKIP_BAND_ENERGY 配置是否开启通带能量检测。具体请参考章节 44.4.1。(R/W)

Register 44.5. LP_I2S_VAD_OB0_REG (0x00B0)



LP_I2S_SPEECH_COUNT_OB 表示语音帧计数器 (speech_count) 的值。具体请参考章节 44.4.1。(RO)

LP_I2S_SILENT_COUNT_OB 表示非语音帧计数器 (silent_count) 的值。具体请参考章节 44.4.1。(RO)

第 VII 卷

通讯接口

该部分涉及系统的连接性，介绍各种带有通讯接口的模块，如 I2C、I2S、SPI、UART、USB 等，同时还包括其他用于遥控、电机控制、LED 控制等功能的信号生成接口。

第 45 章

UART 控制器 (UART)

45.1 概述

UART 是一种以字符为导向、用于实现设备间通信的异步传输接口。异步传输的意思是不需要在发送数据上添加时钟信息。这也要求发送端和接收端的速率、停止位、奇偶校验位等都要相同，通信才能成功。

一个典型的 UART 帧开始于一个起始位，紧接着是有效数据，然后是奇偶校验位（可有可无），最后是停止位。ESP32-P4 上的 UART 控制器支持多种字符长度和停止位。另外，控制器还支持软硬件流控和 GDMA，可以实现高速的数据传输。开发者可以使用多个 UART 端口，同时又能保证很少的软件开销。

ESP32-P4 芯片中共有六个 UART 控制器，包含五个正常功能的 UART 和一个满足低功耗需求的 LP UART。UART 控制器可以兼容不同的 UART 设备。另外，UART 还可以用于红外数据交换 (IrDA) 或 RS485 调制解调器。本文以 UART*n* 指代五个 UART 控制器，*n* 为 0、1、2、3、4。LP UART 是 UART 的功能剪裁版本，且单独控制一组寄存器，通过表 45.2-1 可以区分 UART 和 LP UART 的特性。

45.2 主要特性

UART 控制器的特性以及与 LP UART 的特性区分见下表：

表 45.2-1. UART 和 LP UART 特性区分

UART 特性	LP UART 特性
可编程收发波特率，最大为 5 MBaud	
一个 UART 的发送通道和接收通道分别有 128 x 8 bit RAM 存储空间	LP UART 的发送 FIFO 和接收 FIFO 共用 20 x 8 bit 存储空间
全双工异步通信	
数据位（5 到 8 位）	
停止位（1、1.5 或 2 位）	
奇偶校验位	
AT_CMD 特殊字符检测	
RS485 协议	—
IrDA 协议	—
GDMA 高速数据通信	—
接收超时	
UART 唤醒模式	
软件流控和硬件流控	

见下页

表 45.2-1 – 接上页

UART 特性	LP_UART 特性
三个可分频的时钟源： 1. XTAL_CLK 2. RC_FAST_CLK 3. PLL_F80M_CLK	三个可分频的时钟源： 1. RC_FAST_CLK 2. XTAL_DIV_CLK 3. PLL_F8M_CLK

下文主要针对正常功能的 UART 进行叙述。

45.3 UART 架构

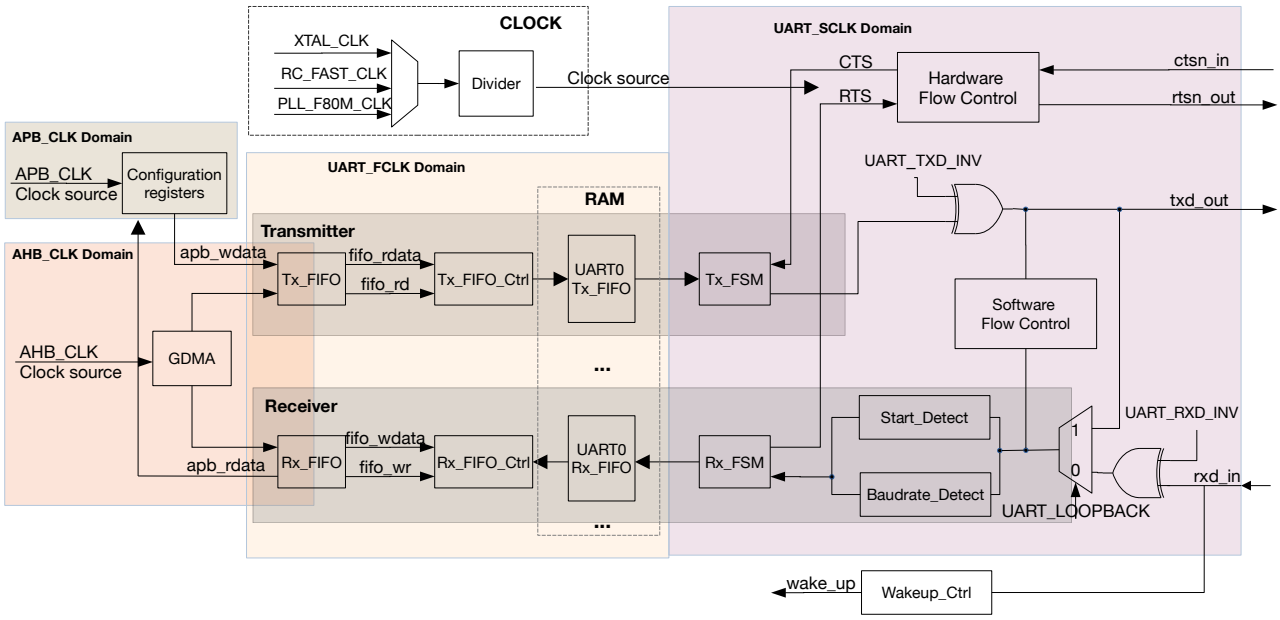


图 45.3-1. UART 基本架构图

图 45.3-1 为 UART 基本架构图。UART 模块工作在 4 个时钟域，分别为 APB_CLK、AHB_CLK、UART_SCLK、UART_FCLK。其中 APB_CLK 与 AHB_CLK 时钟关系为同步不同频（APB_CLK 由 AHB_CLK 分频得到），UART_SCLK 与 UART_FCLK 时钟关系为同步不同频（UART_SCLK 由 UART_FCLK 分频得到）。UART_FCLK 有三个时钟源：PLL_F80M_CLK、RC_FAST_CLK 以及晶振时钟 XTAL_CLK（详情请参考章节 11 复位和时钟）。可以通过配置 HP_SYS_UART*n*_CLK_SRC_SEL 来选择时钟源，之后使用分频器对时钟源进行分频，然后产生时钟信号 UART_SCLK。分频系数的整数部分由 HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUM 配置，小数部分的分母由 HP_SYS_CLKRST_UART*n*_SCLK_DIV_DENOMINATOR 配置，小数部分的分子由 HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUMERATOR 配置。支持的分频范围为 1 ~ 256。只有正常功能的 UART 包含此分频器，LP UART 不包含。

UART 控制器可以分为两个功能块：发送块和接收块。

发送块包含一个发送 FIFO 用于缓存待发送的数据。软件可以通过 APB 总线向 Tx_FIFO 写数据，也可以通过 GDMA 将数据搬入 Tx_FIFO。Tx_FIFO_Ctrl 用于控制 Tx_FIFO 的读写过程，当 Tx_FIFO 非空时，Tx_FSM 通过 Tx_FIFO_Ctrl 读取数据，并将数据按照配置的帧格式转化成比特流。比特流输出信号 txd_out 可以通过配置 UART_TXD_INV 寄存器实现取反功能。

接收块包含一个接收 FIFO 用于缓存待处理的数据。输入比特流 rxd_in 可以输入到 UART 控制器。可以通过 UART_RXD_INV 寄存器实现取反。Baudrate_Detect 通过检测最小比特流输入信号的脉宽来测量输入信号的波特率。Start_Detect 用于检测数据的起始位，当检测到起始位之后，Rx_FSM 通过 Rx_FIFO_Ctrl 将帧解析后的数据存入 Rx_FIFO 中。软件可以通过 APB 总线读取 Rx_FIFO 中的数据，也可以使用 GDMA 进行数据接收。

HW_Flow_Ctrl 通过标准 UART RTS 和 CTS（rtsn_out 和 ctsn_in）流控信号来控制 rxd_in 和 txd_out 的数据流。SW_Flow_Ctrl 通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来进行数据流的控制。当 UART 处于 Light-sleep 模式（详情请参考章节 15 低功耗管理）时，可以通过四种不同的方式产生 wake_up 信号给 RTC 模块，由 RTC 来唤醒 ESP32-P4 芯片。具体唤醒方式请参考章节 45.4.8。

45.4 功能描述

45.4.1 时钟与复位

UART 为异步外设。其寄存器配置模块工作在 APB_CLK 时钟域，TX/RX FIFO 作为跨时钟域模块工作在 AHB_CLK 和 UART_FCLK 时钟域。UART RAM 控制单元工作在 UART_FCLK 时钟域，控制 UART 发送与接收的模块工作在 UART_SCLK 时钟域，即 UART Core 模块的时钟域。

在 UART_SCLK 时钟频率满足生成波特率的需求下，可通过预分频使 UART Core 模块工作在较小的时钟频率，从而减小 UART 外设的功耗。通常情况下，UART Core 模块时钟小于 APB_CLK 时钟，并且在满足 UART 波特率的情况下，UART Core 时钟分频系数可以配置到最大值。UART 也支持 UART Core 模块时钟大于 APB_CLK 时钟，此时，UART Core 模块时钟最大为 APB_CLK 的 3 倍。另外，UART Core 的 TX/RX 时钟可以被单独控制。置位 UART_TX_SCLK_EN 使能 UART Core 的 TX 时钟；置位 UART_RX_SCLK_EN 使能 UART Core 的 RX 时钟。

为确保配置寄存器的值成功从 APB_CLK 时钟域同步到 UART_SCLK 时钟域，寄存器配置需要遵循一定的流程，详情请参考章节 45.6。

对整个 UART 的复位，需要遵循如下配置流程：

- 将 HP_SYS_CLKRST_UART*n*_CLK_EN 置 1 打开 UART*n* Core 时钟。
- 向 HP_SYS_CLKRST_RST_EN_UART*n*_CORE 位写 1。
- 将 HP_SYS_CLKRST_RST_EN_UART*n*_CORE 位清 0。

45.4.2 UART FIFO

UART 控制器的发送块和接收块分别使用一块 128 x 8 位 RAM 存储空间，并且分别通过一个 4 x 8 位的异步 FIFO 接口访问对应的 RAM。发送块和接收块的存储空间和异步 FIFO 接口相互独立，不可复用。

UART 的 Tx_FIFO 可以通过置位 UART_TXFIFO_RST 来复位，Rx_FIFO 可以通过置位 UART_RXFIFO_RST 来复位。

对于 TX FIFO，可以通过 APB 总线或 GDMA 向其写入数据，硬件 Tx_FSM 自动从其中读取数据，数据将按照配置的帧格式转换成比特流；对于 RX FIFO，可以通过 APB 总线或 GDMA 读取其中的数据，并存储到内存，硬件 Rx_FSM 将接收到的比特流转换成字节并写入 RX FIFO。五个 UART 共享同一个 GDMA 通道。

配置 UART_TXFIFO_EMPTY_THRHD 可以设置 Tx_FIFO 空信号阈值，当存储在 Tx_FIFO 中的数据量小于 UART_TXFIFO_EMPTY_THRHD 时会产生中断 UART_TXFIFO_EMPTY_INT；配置 UART_RXFIFO_FULL_THRHD 可以设置 Rx_FIFO 满信号阈值，当储存在 Rx_FIFO 中的数据量大于或等于 UART_RXFIFO_FULL_THRHD 会产生中断 UART_RXFIFO_FULL_INT。另外，当 Rx_FIFO 中储存的数据量超过其能存储的最大值时，会产生 UART_RXFIFO_OVF_INT 中断。

UART*n* 可以通过寄存器 UART_FIFO_REG 访问 FIFO。您可以写 UART_RXFIFO_RD_BYTE 将数据存入 TX FIFO，也可以读 UART_RXFIFO_RD_BYTE 获取 RX FIFO 中的数据。

45.4.3 波特率产生与检测

45.4.3.1 波特率产生

在 UART 发送或接收数据之前，需要配置寄存器来设置波特率。波特率发生器主要通过对输入时钟源的分频来实现，支持小数分频。UART_CLKDIV_SYNC_REG 将分频系数分成两个部分：UART_CLKDIV 用于配置整数部分，

UART_CLKDIV_FRAG 用于配置小数部分。在输入时钟为 80 MHz 的情况下，UART 能支持的最大波特率为 5 MBaud。

波特率分频器系数为

$$UART_CLKDIV + \frac{UART_CLKDIV_FRAG}{16}$$

也就是说，最终波特率为

$$\frac{INPUT_FREQ}{UART_CLKDIV + \frac{UART_CLKDIV_FRAG}{16}}$$

其中，INPUT_FREQ 为 UART Core 时钟。例如，若 UART_CLKDIV = 694，UART_CLKDIV_FRAG = 7，则分频系数为

$$694 + \frac{7}{16} = 694.4375$$

UART_CLKDIV_FRAG 为 0 时，分频器为整数分频，每 UART_CLKDIV 个输入脉冲都会产生一个输出脉冲。

UART_CLKDIV_FRAG 不为 0 时，分频器为小数分频，输出波特率脉冲不完全统一。如图 45.4-1 所示，每 16 个输出脉冲，波特率发生器分频 (UART_CLKDIV + 1) 个输入脉冲或 UART_CLKDIV 个输入脉冲。分频 (UART_CLKDIV + 1) 个输入脉冲产生 UART_CLKDIV_FRAG 个输出脉冲，分频 UART_CLKDIV 个输入脉冲产生剩余的 (16 - UART_CLKDIV_FRAG) 个输出脉冲。

如图 45.4-1 所示，输出脉冲相互交错，使得输出时序更加统一。

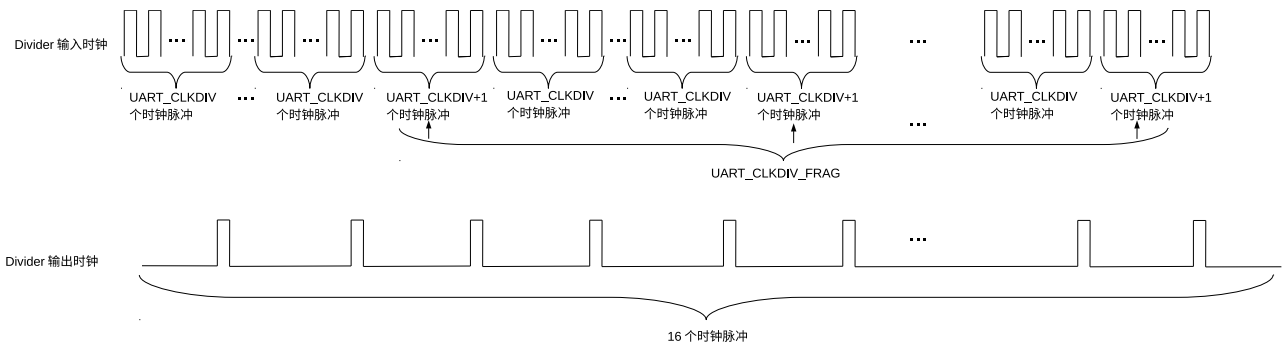


图 45.4-1. UART 控制器分频

为了支持 IrDA（详情见章节 45.4.7），IrDA 小数分频器会产生 16 × UART_CLKDIV_SYNC_REG 分频的时钟用于 IrDA 数据传输。产生 IrDA 数据传输时钟的小数分频器原理与上述小数分频器一样，取UART_CLKDIV/16 作为分频值的整数部分，取 UART_CLKDIV 的低 4 比特作为小数部分。

45.4.3.2 波特率检测

置位 UART_AUTOBAUD_EN 可以开启 UART 波特率自检测功能。图 45.4-2 中的 Baudrate_Detect 可以滤除信号脉宽小于 UART_GLITCH_FILT 的噪声。

在 UART 双方进行通信之前，可以通过发送随机数据，让具有波特率检测功能的数据接收方进行波特率分析。UART_LOWPULSE_MIN_CNT 存储了最小低电平脉冲宽度，UART_HIGHPULSE_MIN_CNT 存储了最小高电平脉冲宽度，UART_POSEDGE_MIN_CNT 存储了两个上升沿之间的最小脉冲宽度，UART_NEGEDGE_MIN_CNT 存储了两个下降沿之间最小的脉冲宽度。软件可以通过读取这四个寄存器获取发送方的波特率。

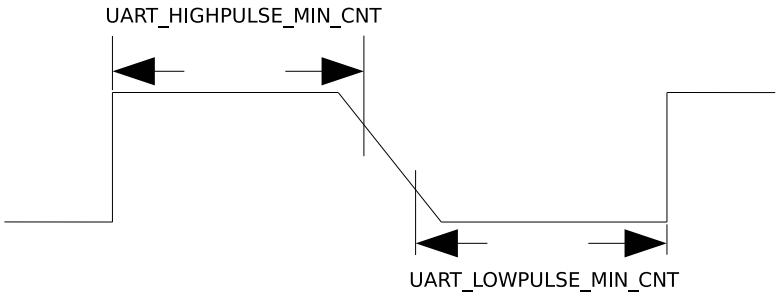


图 45.4-2. UART 信号下降沿较差时序图

波特率的计算分为三种情况：

1. 正常情况下，为防止因亚稳态在上升沿或下降沿附近采样数据错误而导致 UART_LOWPULSE_MIN_CNT 或者 UART_HIGHPULSE_MIN_CNT 不准确，单比特脉冲宽度可以通过将这两个值相加取平均消除误差。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART_LOWPULSE_MIN_CNT} + \text{UART_HIGHPULSE_MIN_CNT} + 2)/2}$$

2. 对于 UART 信号的下降沿信号比较差的情况，如图45.4-2 所示，这时通过取 UART_LOWPULSE_MIN_CNT 与 UART_HIGHPULSE_MIN_CNT 的和平均得到的值不准确，可以通过 UART_POSEDGE_MIN_CNT 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART_POSEDGE_MIN_CNT} + 1)/2}$$

3. 对于 UART 信号的上升沿信号比较差的情况，可以通过 UART_NEGEDGE_MIN_CNT 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART_NEGEDGE_MIN_CNT} + 1)/2}$$

45.4.4 UART 数据帧

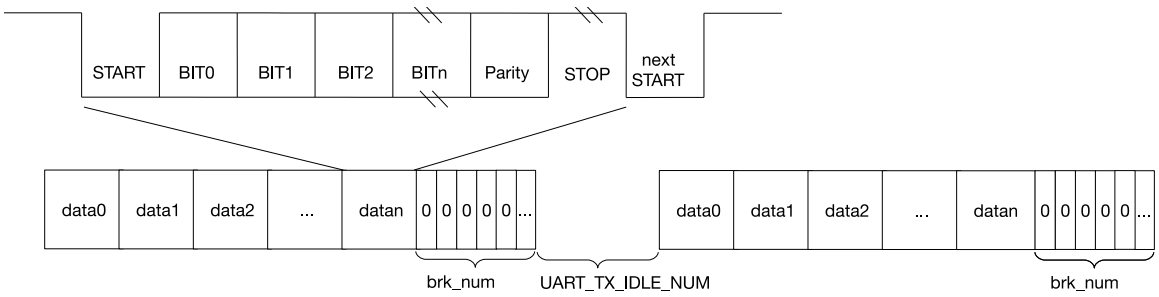


图 45.4-3. UART 数据帧结构

图 45.4-3 所示为基本数据帧格式，数据帧从起始位开始，以停止位结束。起始位占用 1 位，停止位可以通过配置 UART_STOP_BIT_NUM 实现 1、1.5、2 位宽（RS485 模式下可增加转换延时，详见章节 45.4.6.2）。起始位为低电平，停止位为高电平。

数据位宽 (BIT0 ~ BITn) 为 5 ~ 8 位，可以通过 UART_BIT_NUM 进行配置。当置位 UART_PARITY_EN 时，数据帧会在数据之后添加一位奇偶校验位。UART_PARITY 用于选择奇校验或是偶校验。当接收器检测到输入数据的

校验位错误时会产生 UART_PARITY_ERR_INT 中断，输入数据仍会存入 Rx_FIFO。当接收器检测到数据数据帧格式错误时会产生 UART_FRM_ERR_INT 中断，默认情况下，输入数据会被存入 Rx_FIFO。

Tx_FIFO 中数据都发送完成后会产生 UART_TX_DONE_INT 中断。置位 UART_TXD_BRK 时，Tx_FIFO 中数据发送完成后，发送端会进入终止状态 (break condition)，继续发送几个连续的特殊数据帧 NULL。在 NULL 数据帧，TX 数据线输出为低电平。NULL 数据帧的数量可由 UART_TX_BRK_NUM 进行配置。发送器发送完所有的 NULL 数据帧之后会产生 UART_TX_BRK_DONE_INT 中断。数据帧之间可以通过配置 UART_TX_IDLE_NUM 保持最小间隔时间。当一帧数据之后的空闲时间大于等于 UART_TX_IDLE_NUM 寄存器的配置值时则产生 UART_TX_BRK_IDLE_DONE_INT 中断。

在传输一个 NULL 数据帧所需的时间内，接收端 RX 数据线若一直检测到低电平，接收端会检测为终止状态，并触发 UART_BRK_DET_INT 中断表示终止状态已结束。

接收端通过 UART_RXFIFO_TOUT_INT 中断检测总线状态。接收端接收到至少一个字节数据后，总线处于空闲状态超过 UART_RX_TOUT_THRHD 位时间时，触发 UART_RXFIFO_TOUT_INT 中断。您可用此中断检测发送端是否已经发送所有数据。

45.4.5 AT_CMD 字符格式

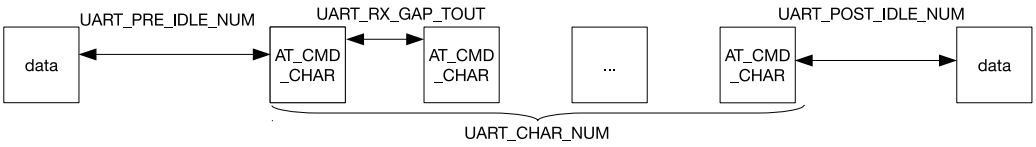


图 45.4-4. AT_CMD 字符格式

图 45.4-4 为一种特殊的 AT_CMD 字符格式。当接收器连续收到 AT_CMD_CHAR 字符且字符之间满足如下条件时将会产生 UART_AT_CMD_CHAR_DET_INT 中断。

- 接收到的第一个 AT_CMD_CHAR 与上一个非 AT_CMD_CHAR 之间间隔至少 UART_PRE_IDLE_NUM 个波特率周期。
- AT_CMD_CHAR 字符之间间隔小于 UART_RX_GAP_TOUT 个波特率周期。
- 接收的 AT_CMD_CHAR 字符个数必须大于等于 UART_CHAR_NUM。
- 接收到的最后一个 AT_CMD_CHAR 字符与下一个非 AT_CMD_CHAR 之间间隔至少 UART_POST_IDLE_NUM 个波特率周期。

注意：由于 AT_CMD_CHAR 字符之间间隔小于 UART_RX_GAP_TOUT 个波特率周期的限制，建议用户使用该功能时，配置的 APB 时钟频率不要低于 8 MHz。

45.4.6 RS485

五个 UART 控制器支持 RS485 通讯模式，RS485 因使用差分信号传输数据，相比于 RS232 具有更远的传输距离及更高的传输速率。RS485 有两线半双工及四线全双工两种选择，UART 模块采用两线半双工模式，并支持侦听总线的功能。

45.4.6.1 驱动控制

如图 45.4-5 所示，RS485 两线 multidrop 系统中，需要一个外部 RS485 收发器实现单端信号与差分信号的转换。RS485 收发器包括一个驱动器与一个接收器。当 UART 不作为发送器时，通过关闭驱动器来断开与差分传

输线的连接。DE 为 1 时，使能驱动器；DE 为 0 关闭驱动器。

UART 接收端通过外部接收器将差分信号转为单端信号。RE 作为接收器的使能控制信号，RE 为 0，使能接收器；RE 为 1，关闭接收器。如果 RE 被配置为 0，从而允许 UART 保持侦听总线上的数据，包括 UART 发送的数据。

DE 信号的控制分为软件控制和硬件控制两种方法。为减少软件的开销，DE 信号采用硬件来控制。图 45.4-5 所示，DE 与 UART 的 dtrn_out 相连（详见 45.4.9.1 小节）。

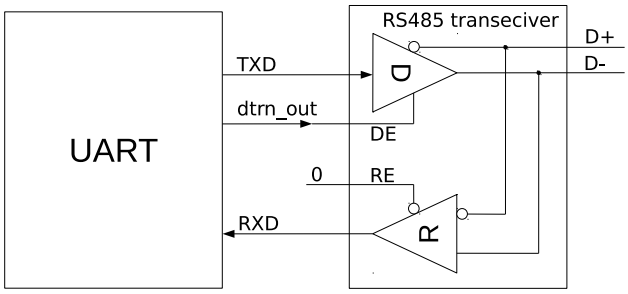


图 45.4-5. RS485 模式驱动控制结构图

45.4.6.2 转换延时

默认情况下，UART 处于接收状态。当从发送转为接收状态时，为保证发送数据被稳定接收，RS485 协议要求在发送停止位之后增加一个波特率的转换延时。UART 发送模块支持在起始位之前或在停止位之后增加一个波特率的延时。置位 UART_DLO_EN，在起始位之前增加一个波特率周期延时；置位 UART_DL1_EN，在停止位之后增加一个波特率周期延时。

45.4.6.3 总线侦听

RS485 两线 multidrop 系统中，当外部 RS485 收发器的 RE 被配置为 0 时，UART 支持侦听总线。默认情况下，不允许 UART 在发送数据时接收数据。置位 UART_RS485TX_RX_EN，允许在发送数据时接收数据，配合图 45.4-5 中外部 RS485 收发器的配置，UART 保持侦听传输总线。另外，默认情况下，不允许 UART 在接收数据时发送数据。置位 UART_RS485RXBY_TX_EN，允许在接收数据时发送数据。

UART 支持侦听 UART 发送的数据。UART 处于发送状态下，当侦听到 UART 发送的数据与 UART 接收的数据冲突时，触发 UART_RS485_CLASH_INT 中断；侦听到发送的数据帧错误时，触发 UART_RS485_FRM_ERR_INT 中断；侦听到发送数据校验位错误时，触发 UART_RS485_PARITY_ERR_INT 中断。

45.4.7 IrDA

IrDA 数据协议由物理层，链路接入层和链路管理层三个基本层协议组成。UART 实现了其物理层协议。在 IrDA 编码模式下，支持最大信号速率到 115.2 Kbit/s，即 SIR 模式。如图 45.4-6 所示，IrDA 编码器将来自 UART 的非归零编码 (NRZ) 信号采用反向归零编码 (RZI) 并输出给外部驱动和红外 LED，用 3/16 Bit Time 的脉宽调制信号表示逻辑 “0”，用低电平表示逻辑 “1”。IrDA 解码器接收来自红外接收器的信号并输出为 UART 的 NRZ 编码。一般情况下，接收端信号空闲时为高电平，编码器输出校验位与解码器输入校验位相反。当检测到低脉冲表示接收到开始信号。

IrDA 使能时，一个位被划分为 16 个时钟周期，在其第 9、10、11 个时钟周期中，当需要发送的比特为 0 时，IrDA 输出为高。

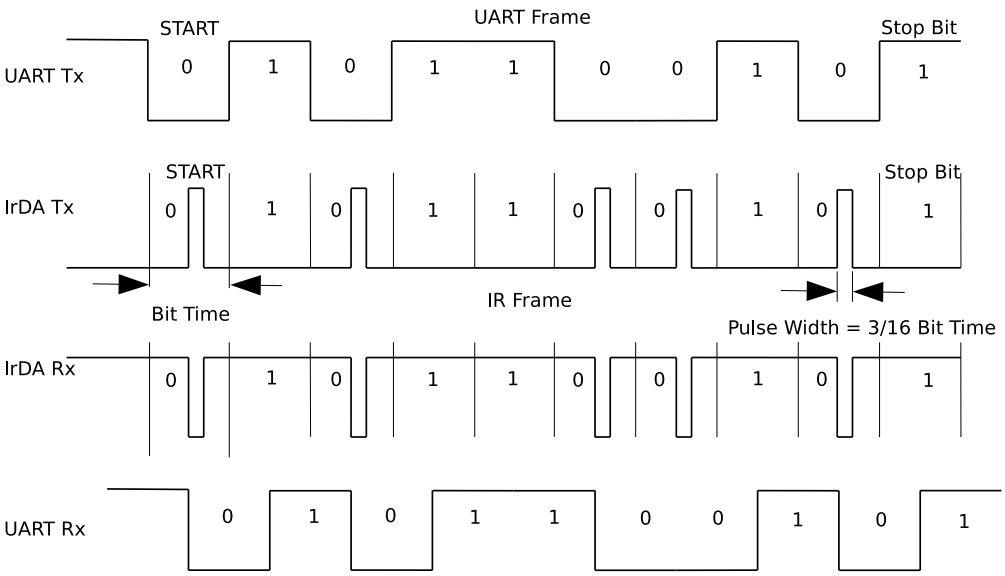


图 45.4-6. SIR 模式编解码时序图

IrDA 为半双工传输协议，不允许同时进行收发。如图45.4-7 所示，置位 UART_IRDA_EN 使能 IrDA 功能。置位 UART_IRDA_TX_EN（置 1）使能 IrDA 发送数据，这时不允许 IrDA 接收数据；复位 UART_IRDA_TX_EN（清 0）使能 IrDA 接收数据，这时不允许 IrDA 发送数据。

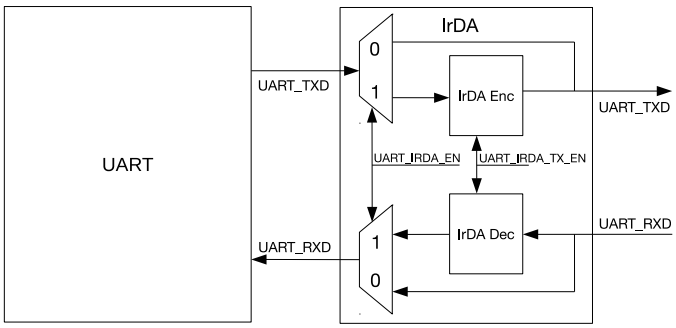


图 45.4-7. IrDA 编解码结构图

45.4.8 唤醒

UART 支持唤醒功能。当 UART 处于 Light-sleep 模式时，可以通过四种不同的方式产生 wake_up 信号给 RTC 模块，由 RTC 来唤醒芯片。

- UART_WK_MODE_SEL=0，当所有的时钟都关闭时，此时可以通过使 RXD 翻转若干周期，当上升沿个数大于等于 UART_ACTIVE_THRESHOLD +3 时唤醒芯片。
- UART_WK_MODE_SEL=1，由于 UART Core 时钟保持工作，因此 UART RX 仍然可以接收数据并将数据暂存在 Rx FIFO 中。当 Rx FIFO 中的数据量超过 UART_RX_WAKE_UP_THRHD 时，可以将芯片从 Light-sleep 中唤醒。
- UART_WK_MODE_SEL=2，当 UART RX 监测到起始位后，唤醒芯片。
- UART_WK_MODE_SEL=3，当 UART RX 接收到特定字符序列后，唤醒芯片。用户通过配置 UART_WK_CHAR0、UART_WK_CHAR1、UART_WK_CHAR2、UART_WK_CHAR3、UART_WK_CHAR4 自定义唤醒字符。UART 监测到由 CHAR0/CHAR1/CHAR2/CHAR3/CHAR4 组成的字符序列流之后唤醒芯片。

可以通过配置 UART_CHAR_NUM 和 UART_WK_CHAR_MASK 来设置不同的字符序列。如下表所示，对于最后一个配置，UART 将依次检测 CHAR0 ~ CHAR4。

表 45.4-1. UART_CHAR_WAKEUP 模式配置

UART_CHAR_NAME	UART_WP_CHAR_MASK	检测字符序列
1	0xF	CHAR4
2	0x7	CHAR3/CHAR4
3	0x3	CHAR2/CHAR3/CHAR4
4	0x1	CHAR1/CHAR2/CHAR3/CHAR4
5	0x0	CHAR0/CHAR1/CHAR2/CHAR3/CHAR4

使用 UART 唤醒之后，需要通过在 Active 模式下向 UART 传输数据或是复位整个 UART 模块清除 wake_up 信号，否则下一次唤醒所需的上升沿个数将减少。

45.4.9 流控

UART 控制器有两种数据流控方式：硬件流控和软件流控。硬件流控主要通过输出信号 rtsn_out 以及输入信号 ctsn_in 进行数据流控制。软件流控主要通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来实现数据流控功能。

45.4.9.1 硬件流控

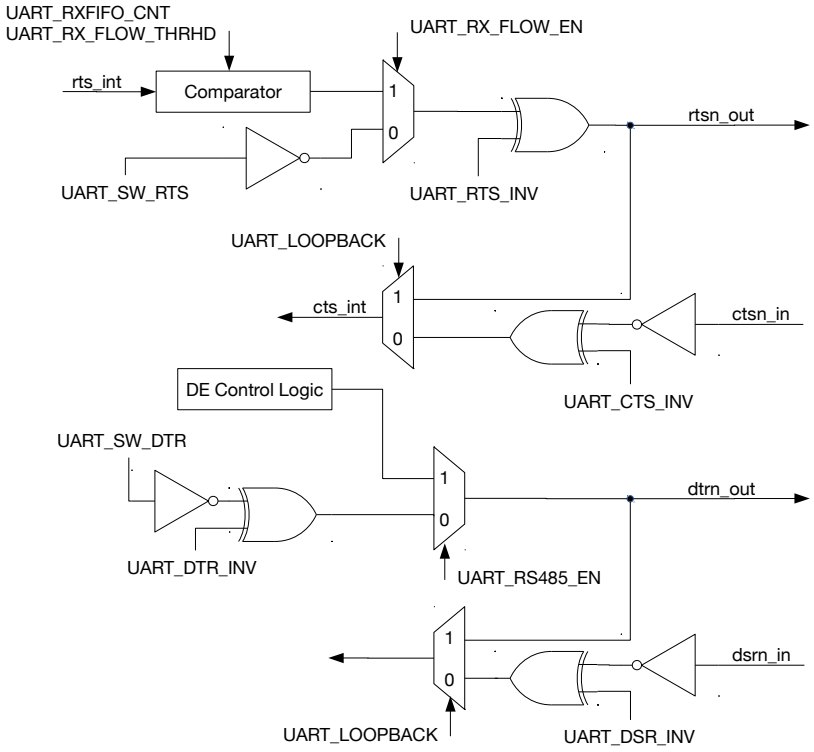


图 45.4-8. 硬件流控图

图 45.4-8 为 UART 硬件流控图。硬件流控的控制信号为输出信号 rtsn_out 及输入信号 ctsn_in。图 45.4-9 为两个 UART 之间硬件流控信号连接图。记 ESP32-P4 UART 为 IUO，External UART 为 EUO，下文将使用这两个标

记来区分两个 UART。输出信号 rtsn_out (IUO) 为低电平表示允许对方 (EUO) 发送数据，rtsn_out (IUO) 为高电平表示通知对方 (EUO) 中止数据发送直到 rtsn_out (IUO) 恢复低电平。rtsn_out 输出信号的控制有两种方式。

- 软件控制：将 UART_RX_FLOW_EN 置 0 进入该模式。该模式下通过软件配置 UART_SW_RTS 改变 rtsn_out 的电平。
- 硬件控制：将 UART_RX_FLOW_EN 置 1 进入该模式。该模式下硬件会当 Rx_FIFO 中的数据大于 UART_RX_FLOW_THRHD 时拉高 rtsn_out 的电平。

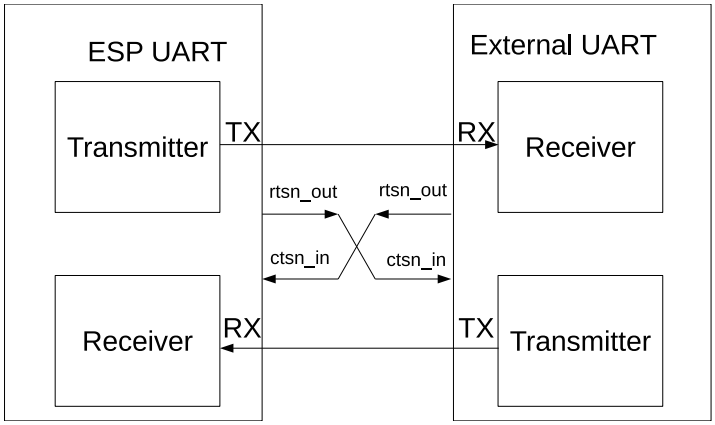


图 45.4-9. 硬件流控信号连接图

输入信号 ctsn_in (IUO) 为低电平表示允许发送端 (IUO) 发送数据；ctsn_in (IUO) 为高电平表示禁止发送端 (IUO) 发送数据。当 UART 检测到输入信号 ctsn_in (IUO) 的沿变化时会产生 UART_CTS_CHG_INT 中断。

UART 发送设备 (IUO) 输出信号 dtrn_out 为高电平表示发送数据已经准备完毕，处于可用状态。dtrn_out 通过配置寄存器 UART_SW_DTR 产生。UART 接收设备 (IUO) 在检测到输入信号 dsrn_in 的沿变化时会产生 UART_DSR_CHG_INT 中断。软件在检测到中断后，通过读取 UART_DSRN 可以获取 dsrn_in 的输入信号电平，UART_DSRN 为高电平时，表示对方设备 (EUO) 处于可用状态。

对于 RS485 两线 multidrop 系统，使用 dtrn_out 来收发转换。置位 UART_RS485_EN 使能 RS485 功能，dtrn_out 由硬件产生。数据开始发送时，dtrn_out 拉高，使能外部驱动器；数据最后一位发送完成后，dtrn_out 拉低，关闭外部驱动器。注意，当使能停止位之后增加一个波特率延时时，dtrn_out 会在延时结束后才拉低。

置位 UART_LOOPBACK 即开启 UART 的回环测试功能。此时 UART 的输出信号 txd_out 和其输入信号 rxd_in 相连，rtsn_out 和 ctsn_in 相连，dtrn_out 和 dsrn_out 相连。当接收的数据与发送的数据相同时表明 UART 能够正常发送和接收数据。

45.4.9.2 软件流控

软件流控不使用硬件的 CTS/RTS 控制线，而是在发送数据流中嵌入 XON/XOFF 字符来通知对方是否可以使用数据发送来实现流控。将 UART_SW_FLOW_CON_EN 置 1 使能软件流控。

在使用软件流控后，硬件会自动检测接收数据流中是否有 XON/XOFF 字符，在检测到相应的字符后会产生 UART_SW_XOFF_INT 或 UART_SW_XON_INT 中断。在检测到接收数据流中有 XOFF 字符后，发送器将会在发送完当前数据后停止发送；在检测到接收数据流中有 XON 字符后，将会使能发送器发送数据。另外，软件可以通过置位 UART_FORCE_XOFF 来强制发送器停止发送数据，发送器会在发送完当前字节后停止发送；也可以通过置位 UART_FORCE_XON 来使能发送器发送数据。

软件可以根据 Rx_FIFO 中剩余空间大小决定流控字符的发送。置位 UART_SEND_XOFF，发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 UART_XOFF_CHAR 配置；置位 UART_SEND_XON，发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 UART_XON_CHAR 配置。另外，当 UART 接收 FIFO 中的数据量超过 UART_XOFF_THRESHOLD 时，硬件会置位 UART_SEND_XOFF，UART 发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 UART_XOFF_CHAR 配置。当 UART 接收 FIFO 中的数据量小于 UART_XON_THRESHOLD 时，硬件会置位 UART_SEND_XON，UART 发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 UART_XON_CHAR 配置。

当 UART 工作在全双工模式时，如果 UART RX 接收到 XOFF，此时尽管 UART RX 接受的数据超过阈值，UART TX 将被禁止发送包括 XOFF 在内的任何数据。为了避免该种情况导致的软流控死锁或者 overflow，可以使能 UART_XON_XOFF_STILL_SEND，此时允许 UART TX 在被禁止发送时，仍然可以发送一个 XOFF 字符。

45.4.10 GDMA 模式

ESP32-P4 中的五个 UART 接口通过通用主机控制器接口 (UHCI) 共用 1 组 GDMA TX/RX 通道。在 GDMA 模式下，支持对 HCI 协议数据包的解析 (decoder) 及数据包封装 (encoder)。UHCI_UART_SEL 字段用于选择哪个串口占用 GDMA 通道。

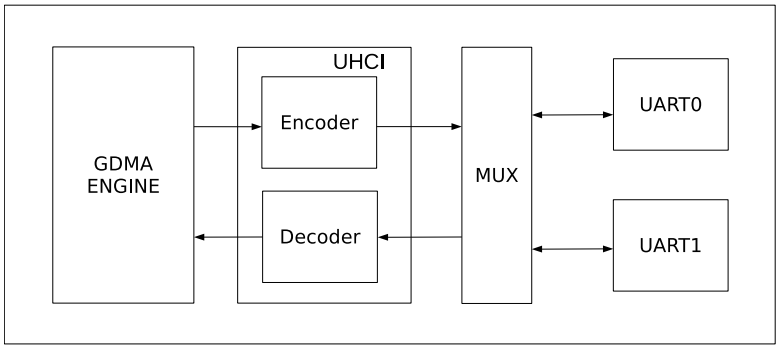


图 45.4-10. GDMA 模式数据传输

图 45.4-10 为 GDMA 方式数据传输图。在 GDMA Rx 通道接收数据前，软件将接收链表准备好。GDMA_INLINK_ADDR_CH*n* 用于指向第一个接收链表描述符。置位 GDMA_INLINK_START_CH*n* 之后，通用主机控制器接口 (UHCI) 会将 UART 接收到的数据传送给 decoder。经过 decoder 解析之后的数据在 GDMA 通道的控制下存入接收链表指定的 RAM 空间。

在 GDMA Tx 通道发送数据前，软件需要将发送链表和发送数据准备好，GDMA_OUTLINK_ADDR_CH*n* 用于指向第一个发送链表描述符。置位 GDMA_OUTLINK_START_CH*n* 之后，GDMA 引擎即从链表中指定的 RAM 地址读取数据，并通过 encoder 进行数据包封装，然后经 UART 的发送模块串行发送出去。

HCI 的数据包格式为（分隔符 + 数据 + 分隔符）。encoder 用于在数据前后加上分隔符，并将数据中和分隔符一样的数据替换为特殊字符。decoder 用于去除数据包前后分隔符，并将数据中的特殊字符替换为分隔符。数据前后的分隔符可以有连续多个。分隔符可由 UHCI_SEPER_CHAR 进行配置，默认值为 0xC0。数据中与分隔符一样的数据可以用 UHCI_ESC_SEQ0_CHAR0（默认为 0xDB）和 UHCI_ESC_SEQ0_CHAR1（默认为 0xDD）进行替换。当数据全部发送完成后，会产生 GDMA_OUT_TOTAL_EOF_CH*n*_INT 中断。当数据接收完成后，会产生 GDMA_IN_SUC_EOF_CH*n*_INT 中断。

45.5 中断

ESP32-P4 的 UART*n* 和 UHCI 可以生成以下中断信号并发送给中断矩阵。

- UART*n*_INTR
- UHCI_INTR

这些中断信号分别由 UART*n* 和 UHCI 的内部中断源生成。

UART*n* 中断源如下所列：

- UART_AT_CMD_CHAR_DET_INT：当接收器检测到 AT_CMD 字符时触发此中断。
- UART_RS485_CLASH_INT：在 RS485 模式下检测到发送器和接收器之间的冲突时触发此中断。
- UART_RS485_FRM_ERR_INT：在 RS485 模式下检测到发送块发送的数据帧错误时触发此中断。
- UART_RS485_PARITY_ERR_INT：在 RS485 模式下检测到发送块发送的数据校验位错误时触发此中断。
- UART_TX_DONE_INT：当发送器发送完 FIFO 中的所有数据时触发此中断。
- UART_TX_BRK_IDLE_DONE_INT：当发送器在最后一个数据发送后保持了最短的间隔时间时触发此中断。
- UART_TX_BRK_DONE_INT：当发送 FIFO 中的数据发送完之后发送器完成了发送 NULL 则触发此中断。
- UART_GLITCH_DET_INT：当接收器在起始位的中点处检测到毛刺时触发此中断。
- UART_SW_XOFF_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XOFF 字符时触发此中断。
- UART_SW_XON_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XON 字符时触发此中断。
- UART_RXFIFO_TOUT_INT：当接收器至少已经接收一个字节的数据，且之后在 UART_RX_TOUT_THRHD 位的时间内，总线一直处于 idle 状态将触发此中断。
- UART_BRK_DET_INT：当接收器在停止位之后检测到一个 NULL（即传输一个 NULL 的时间内保持逻辑低电平）时触发此中断。
- UART_CTS_CHG_INT：当接收器检测到 CTSn 信号的沿变化时触发此中断。
- UART_DSR_CHG_INT：当接收器检测到 DSRn 信号的沿变化时触发此中断。
- UART_RXFIFO_OVF_INT：当接收器接收到的数据量多于 FIFO 的存储量时触发此中断。
- UART_FRM_ERR_INT：当接收器检测到数据帧错误时触发此中断。
- UART_PARITY_ERR_INT：当接收器检测到校验位错误时触发此中断。
- UART_TXFIFO_EMPTY_INT：当发送 FIFO 中的数据量少于 UART_TXFIFO_EMPTY_THRHD 所指定的值时触发此中断。
- UART_RXFIFO_FULL_INT：当接收器接收到的数据多于 UART_RXFIFO_FULL_THRHD 所指定的值时触发此中断。
- UART_WAKEUP_INT：UART 被唤醒时产生此中断。

UHCI 中断源如下所列：

- UHCI_APP_CTRL1_INT：软件置位 UHCI_APP_CTRL1_INT_RAW 时触发此中断。
- UHCI_APP_CTRL0_INT：软件置位 UHCI_APP_CTRL0_INT_RAW 时触发此中断。
- UHCI_OUTLINK_EOF_ERR_INT：当检测到发送链表描述符中的 EOF 有错误时触发此中断。
- UHCI_SEND_A_REG_Q_INT：当使用 always_send 发送一串短包，UHCI 发送了短包后触发此中断。
- UHCI_SEND_S_REG_Q_INT：当使用 single_send 发送一串短包，UHCI 发送了短包后触发此中断。

- UHCI_TX_HUNG_INT：当 UHCI 利用 GDMA Tx 通道从 RAM 中读取数据的时间过长时触发此中断。
- UHCI_RX_HUNG_INT：当 UHCI 利用 GDMA Rx 通道接收数据的时间过长时触发此中断。
- UHCI_TX_START_INT：当检测到分隔符时触发此中断。
- UHCI_RX_START_INT：当分隔符已发送时触发此中断。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 45.7 寄存器列表。

45.6 编程流程

45.6.1 寄存器类型

UART 的所有寄存器都处于 APB_CLK 时钟域。

UART 上的配置寄存器分为两类：第一类寄存器作用在 APB/AHB 时钟域，该类寄存器配置后不需要额外的操作；第二类寄存器作用在 UART_SCLK 时钟域，因此需要进行跨时钟域处理。在配置完该类寄存器后，需要通过写寄存器 UART_REG_UPDATE 使配置的值同步到 Core 时钟域。当同步完成后，硬件会将 UART_REG_UPDATE 自动清零。推荐用户在配置需要同步的寄存器之后，检查 UART_REG_UPDATE 是否为 0，确保上一次的同步操作完成之后再执行接下来的寄存器配置。

为了方便用户区分这两类寄存器，章节 45.7 中将需要做跨时钟域操作的寄存器放在一起，该类寄存器名称附带 _SYNC 后缀。不需要做跨时钟域操作的寄存器放在一起，该类寄存器名称不带 _SYNC 后缀。

45.6.2 具体步骤

图45.6-1 显示了 UART 模块的编程流程。主要包括：初始化、寄存器配置、启动 UART TX/RX 和数据传输结束。

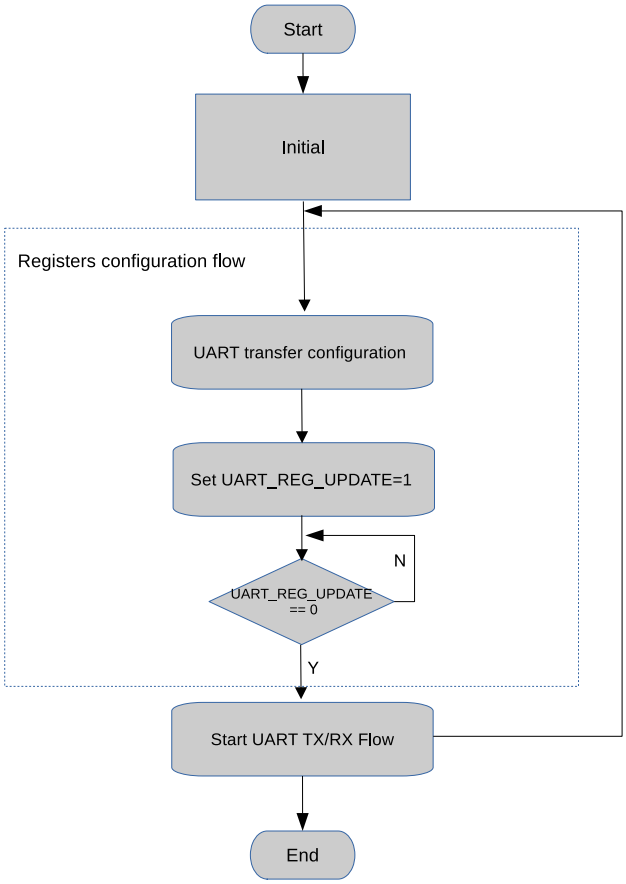


图 45.6-1. UART 编程流程

45.6.2.1 UART*n* 模块初始化

UART*n* 模块初始化流程如下：

- HP_SYS_CLKRST_RST_EN_UART*n*_APB 位写 1；
- HP_SYS_CLKRST_RST_EN_UART*n*_APB 位清零。
- HP_SYS_CLKRST_RST_EN_UART*n*_CORE 位写 1；
- HP_SYS_CLKRST_RST_EN_UART*n*_CORE 位清零。

45.6.2.2 UART*n* 通信配置

UART*n* 通信配置流程如下：

- 等待 UART_REG_UPDATE 为 0，确保上一次同步已经完成；
- 配置 HP_SYS_CLKRST_UART*n*_CLK_SRC_SEL 选择时钟源；
- 配置 HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUM、HP_SYS_CLKRST_UART*n*_SCLK_DIV_DENOMINATOR、HP_SYS_CLKRST_UART*n*_SCLK_DIV_NUMERATOR 设置预分频器系数；

- 配置 UART_CLKDIV、UART_CLKDIV_FRAG 设置发送波特率；
- 配置 UART_BIT_NUM 设置数据长度；
- 配置 UART_PARITY_EN、UART_PARITY 设置奇偶校验；
- 可选步骤，根据应用不同存在差异；
- 向 UART_REG_UPDATE 写 1，将配置的值同步到 Core 时钟域。

45.6.2.3 启动 UART*n*

启动 UART*n* TX 发送数据：

- 配置 UART_TXFIFO_EMPTY_THRHD，设置 TX FIFO 空阈值；
- 对 UART_TXFIFO_EMPTY_INT_ENA 置 0，关闭 UART_TXFIFO_EMPTY_INT 中断；
- 向 UART_RXFIFO_RD_BYTE 写入需要发送的数据；
- 置位 UART_TXFIFO_EMPTY_INT_CLR，清除 UART_TXFIFO_EMPTY_INT 中断；
- 置位 UART_TXFIFO_EMPTY_INT_ENA，使能 UART_TXFIFO_EMPTY_INT 中断；
- 检测 UART_TXFIFO_EMPTY_INT_ST，等待发送数据结束。

启动 UART*n* RX 数据接收：

- 配置 UART_RXFIFO_FULL_THRHD，设置 RX FIFO 满阈值；
- 置位 UART_RXFIFO_FULL_INT_ENA，使能 UART_RXFIFO_FULL_INT 中断；
- 检测 UART_RXFIFO_FULL_INT_ST，等待 RX FIFO 接收数据满；
- 通过读 UART_RXFIFO_RD_BYTE，从 RX FIFO 中读出数据，并可通过 UART_RXFIFO_CNT 获得当前 RX FIFO 中的接收数据量。

45.7 寄存器列表

45.7.1 UART 寄存器列表

本小节的所有地址均为相对于 UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 配置			
UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART 阈值和分配配置	0x0064	R/W
UART 中断寄存器			
UART_INT_RAW_REG	原始中断状态	0x0004	R/WTC/SS
UART_INT_ST_REG	屏蔽中断状态	0x0008	RO
UART_INT_ENA_REG	中断使能位	0x000C	R/W
UART_INT_CLR_REG	中断清除位	0x0010	WT
配置寄存器			
UART_CLKDIV_SYNC_REG	时钟分频配置	0x0014	R/W
UART_RX_FILT_REG	RX 滤波器配置	0x0018	R/W
UART_CONF0_SYNC_REG	配置寄存器 0	0x0020	R/W
UART_CONF1_REG	配置寄存器 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	硬件流控配置	0x002C	R/W
UART_SLEEP_CONF0_REG	UART 睡眠配置寄存器 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART 睡眠配置寄存器 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART 睡眠配置寄存器 2	0x0038	R/W
UART_SWFC_CONF0_SYNC_REG	软件流控字符配置	0x003C	varies
UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX 断开字符配置	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	帧结束空闲配置	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 模式配置	0x004C	R/W
UART_CLK_CONF_REG	UART core 时钟配置	0x0088	R/W
UART_REG_UPDATE_REG	UART 寄存器配置更新	0x0098	R/W/SC
UART_ID_REG	UART ID 寄存器	0x009C	R/W
状态寄存器			
UART_STATUS_REG	UART 状态寄存器	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0068	RO
UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x006C	RO
UART_FSM_STATUS_REG	UART 发送和接收状态	0x0070	RO
UART_AFIFO_STATUS_REG	UART 异步 FIFO 状态	0x0090	RO
AT 转义序列检测配置			
UART_AT_CMD_PRECNT_SYNC_REG	序列发送前的时序配置	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	序列发送后的时序配置	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	超时配置	0x0058	R/W

名称	描述	地址	访问
UART_AT_CMD_CHAR_SYNC_REG	AT 转义序列检测配置	0x005C	R/W
自动波特率检测寄存器			
UART_POSPULSE_REG	自动波特率检测高电平脉冲寄存器	0x0074	RO
UART_NEGPULSE_REG	自动波特率检测低电平脉冲寄存器	0x0078	RO
UART_LOWPULSE_REG	自动波特率检测最短低电平脉冲持续时间寄存器	0x007C	RO
UART_HIGHPULSE_REG	自动波特率检测最短高电平脉冲持续时间寄存器	0x0080	RO
UART_RXD_CNT_REG	自动波特率检测沿变化计数寄存器	0x0084	RO
版本寄存器			
UART_DATE_REG	UART 版本控制寄存器	0x008C	R/W

45.7.2 LP UART 寄存器列表

本小节的所有地址均为相对于 LP UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 配置			
LP_UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
LP_UART_TOUT_CONF_SYNC_REG	LP UART 阈值和分配配置	0x0064	R/W
LP UART 中断寄存器			
LP_UART_INT_RAW_REG	原始中断状态	0x0004	R/WTC/SS
LP_UART_INT_ST_REG	屏蔽中断状态	0x0008	RO
LP_UART_INT_ENA_REG	中断使能位	0x000C	R/W
LP_UART_INT_CLR_REG	中断清除位	0x0010	WT
配置寄存器			
LP_UART_CLKDIV_SYNC_REG	时钟分频配置	0x0014	R/W
LP_UART_RX_FILT_REG	RX 滤波器配置	0x0018	R/W
LP_UART_CONF0_SYNC_REG	配置寄存器 0	0x0020	R/W
LP_UART_CONF1_REG	配置寄存器 1	0x0024	R/W
LP_UART_HWFC_CONF_SYNC_REG	硬件流控配置	0x002C	R/W
LP_UART_SLEEP_CONF0_REG	LP UART 睡眠配置寄存器 0	0x0030	R/W
LP_UART_SLEEP_CONF1_REG	LP UART 睡眠配置寄存器 1	0x0034	R/W
LP_UART_SLEEP_CONF2_REG	LP UART 睡眠配置寄存器 2	0x0038	R/W
LP_UART_SWFC_CONF0_SYNC_REG	软件流控字符配置	0x003C	varies
LP_UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	R/W
LP_UART_TXBRK_CONF_SYNC_REG	TX 断开字符配置	0x0044	R/W
LP_UART_IDLE_CONF_SYNC_REG	帧结束空闲配置	0x0048	R/W
LP_UART_CLK_CONF_REG	LP UART core 时钟配置	0x0088	R/W
LP_UART_REG_UPDATE_REG	LP UART 寄存器配置更新	0x0098	R/W/SC
LP_UART_ID_REG	LP UART ID 寄存器	0x009C	R/W

名称	描述	地址	访问
状态寄存器			
LP_UART_STATUS_REG	LP UART 状态寄存器	0x001C	RO
LP_UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0068	RO
LP_UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x006C	RO
LP_UART_FSM_STATUS_REG	LP UART 发送和接收状态	0x0070	RO
LP_UART_AFIFO_STATUS_REG	LP UART 异步 FIFO 状态	0x0090	RO
AT 转义序列检测配置配置			
LP_UART_AT_CMD_PRECNT_SYNC_REG	序列发送前的时序配置配置	0x0050	R/W
LP_UART_AT_CMD_POSTCNT_SYNC_REG	序列发送前后的时序配置配置	0x0054	R/W
LP_UART_AT_CMD_GAPTOUT_SYNC_REG	超时配置	0x0058	R/W
LP_UART_AT_CMD_CHAR_SYNC_REG	AT 转义序列检测配置配置	0x005C	R/W
版本寄存器			
LP_UART_DATE_REG	LP UART 版本控制寄存器	0x008C	R/W

45.7.3 UHCI 寄存器列表

本小节的所有地址均为相对于 UHCI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
UHCI_CONF0_REG	UHCI 配置寄存器	0x0000	R/W
UHCI_CONF1_REG	UHCI 配置寄存器	0x0014	varies
UHCI_ESCAPE_CONF_REG	转义符配置	0x0020	R/W
UHCI_HUNG_CONF_REG	超时配置	0x0024	R/W
UHCI_ACK_NUM_REG	配置 UHCI ACK 值	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI 快速发送配置寄存器	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 快速发送寄存器	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 快速发送寄存器	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 快速发送寄存器	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 快速发送寄存器	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 快速发送寄存器	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 快速发送寄存器	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 快速发送寄存器	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 快速发送寄存器	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 快速发送寄存器	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 快速发送寄存器	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5 WORD0 快速发送寄存器	0x005C	R/W
UHCI_REG_Q5_WORD1_REG	Q5 WORD1 快速发送寄存器	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6 WORD0 快速发送寄存器	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6 WORD1 快速发送寄存器	0x0068	R/W
UHCI_ESC_CONF0_REG	转义序列配置寄存器 0	0x006C	R/W

名称	描述	地址	访问
UHCI_ESC_CONF1_REG	转义序列配置寄存器 1	0x0070	R/W
UHCI_ESC_CONF2_REG	转义序列配置寄存器 2	0x0074	R/W
UHCI_ESC_CONF3_REG	转义序列配置寄存器 3	0x0078	R/W
UHCI_PKT_THRES_REG	包长度配置寄存器	0x007C	R/W
UHCI 中断寄存器			
UHCI_INT_RAW_REG	原始中断状态	0x0004	varies
UHCI_INT_ST_REG	屏蔽中断状态	0x0008	RO
UHCI_INT_ENA_REG	中断使能位	0x000C	R/W
UHCI_INT_CLR_REG	中断清除位	0x0010	WT
UHCI 状态寄存器			
UHCI_STATE0_REG	UHCI 接收状态	0x0018	RO
UHCI_STATE1_REG	UHCI 发送状态	0x001C	RO
UHCI_RX_HEAD_REG	UHCI 包报头寄存器	0x002C	RO
版本寄存器			
UHCI_DATE_REG	UHCI 版本控制寄存器	0x0080	R/W

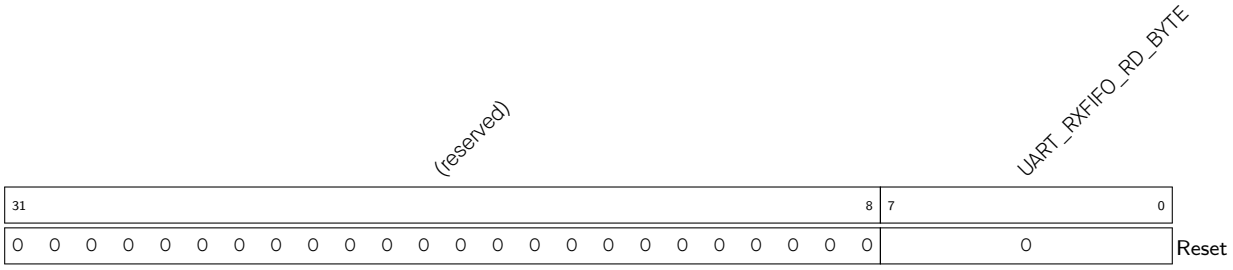
45.8 寄存器

45.8.1 UART 寄存器

本小节的所有地址均为相对于 UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

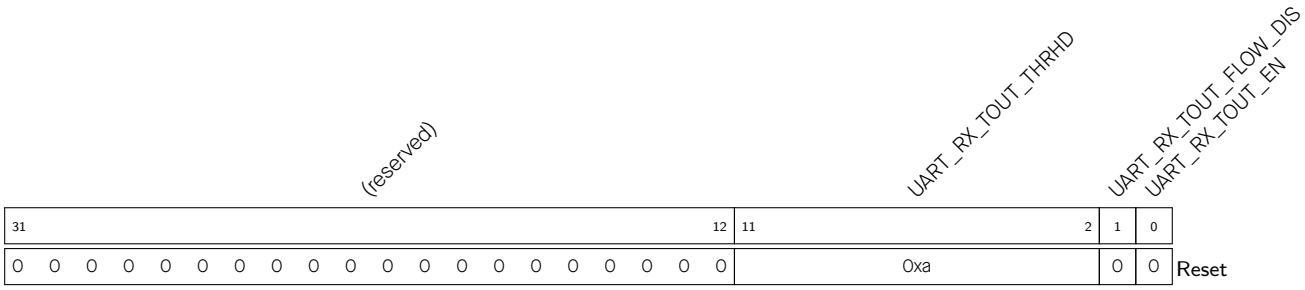
Register 45.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE 表示 UART *n* 从 FIFO 读取的数据。

单位：字节。(RO)

Register 45.2. UART_TOUT_CONF_SYNC_REG (0x0064)



UART_RX_TOUT_EN 配置是否开启 UART 接收器的超时功能。

0：关闭

1：开启

(R/W)

UART_RX_TOUT_FLOW_DIS 配置开启硬件流控时是否关闭 idle 状态计数器计数。

0：开启

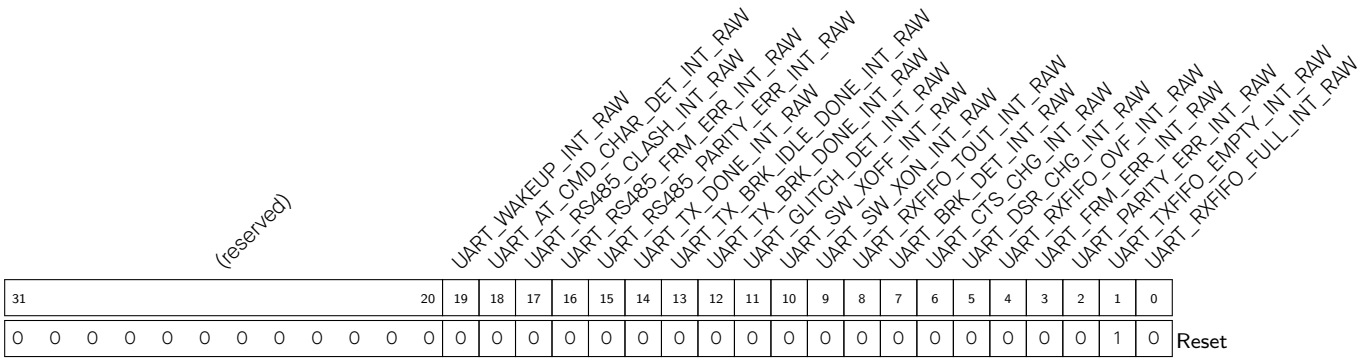
1：关闭

(R/W)

UART_RX_TOUT_THRHD 配置触发超时前总线可保持空闲状态的时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 45.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW UART_RXFIFO_FULL_INT 的原始中断状态。(R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW UART_TXFIFO_EMPTY_INT 的原始中断状态。(R/WTC/SS)

UART_PARITY_ERR_INT_RAW UART_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_FRM_ERR_INT_RAW UART_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW UART_RXFIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

UART_DSR_CHG_INT_RAW UART_DSR_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_CTS_CHG_INT_RAW UART_CTS_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_BRK_DET_INT_RAW UART_BRK_DET_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_TOUT_INT_RAW UART_RXFIFO_TOUT_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XON_INT_RAW UART_SW_XON_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XOFF_INT_RAW UART_SW_XOFF_INT 的原始中断状态。(R/WTC/SS)

UART_GLITCH_DET_INT_RAW UART_GLITCH_DET_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_DONE_INT_RAW UART_TX_BRK_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_IDLE_DONE_INT_RAW UART_TX_BRK_IDLE_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_DONE_INT_RAW UART_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_PARITY_ERR_INT_RAW UART_RS485_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

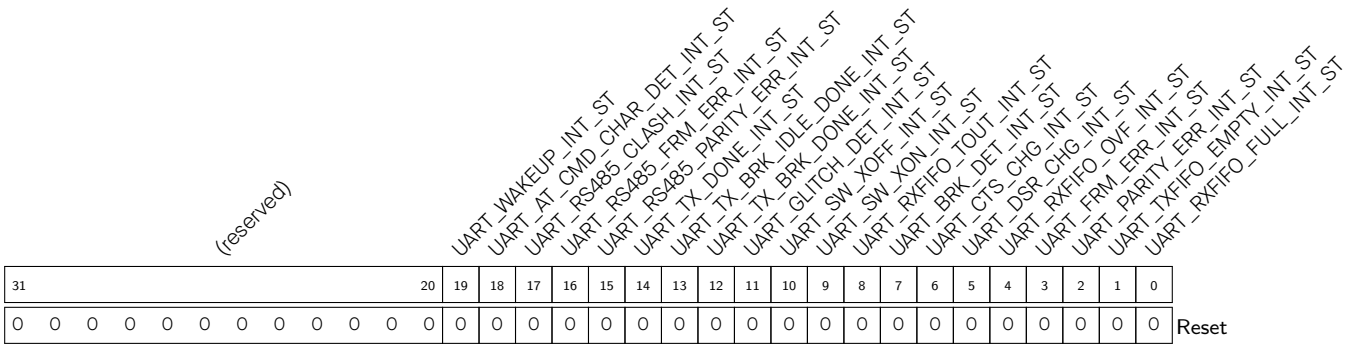
UART_RS485_FRM_ERR_INT_RAW UART_RS485_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_CLASH_INT_RAW UART_RS485_CLASH_INT 的原始中断状态。(R/WTC/SS)

UART_AT_CMD_CHAR_DET_INT_RAW UART_AT_CMD_CHAR_DET_INT 的原始中断状态。(R/WTC/SS)

UART_WAKEUP_INT_RAW UART_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 45.4. UART_INT_ST_REG (0x0008)



UART_RXFIFO_FULL_INT_ST UART_RXFIFO_FULL_INT 的屏蔽中断状态。(RO)

UART_TXFIFO_EMPTY_INT_ST UART_TXFIFO_EMPTY_INT 的屏蔽中断状态。(RO)

UART_PARITY_ERR_INT_ST UART_PARITY_ERR_INT 的屏蔽中断状态。(RO)

UART_FRM_ERR_INT_ST UART_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_OVF_INT_ST UART_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

UART_DSR_CHG_INT_ST UART_DSR_CHG_INT 的屏蔽中断状态。(RO)

UART_CTS_CHG_INT_ST UART_CTS_CHG_INT 的屏蔽中断状态。(RO)

UART_BRK_DET_INT_ST UART_BRK_DET_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_TOUT_INT_ST UART_RXFIFO_TOUT_INT 的屏蔽中断状态。(RO)

UART_SW_XON_INT_ST UART_SW_XON_INT 的屏蔽中断状态。(RO)

UART_SW_XOFF_INT_ST UART_SW_XOFF_INT 的屏蔽中断状态。(RO)

UART_GLITCH_DET_INT_ST UART_GLITCH_DET_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_DONE_INT_ST UART_TX_BRK_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_IDLE_DONE_INT_ST UART_TX_BRK_IDLE_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_DONE_INT_ST UART_TX_DONE_INT 的屏蔽中断状态。(RO)

UART_RS485_PARITY_ERR_INT_ST UART_RS485_PARITY_ERR_INT 的屏蔽中断状态。(RO)

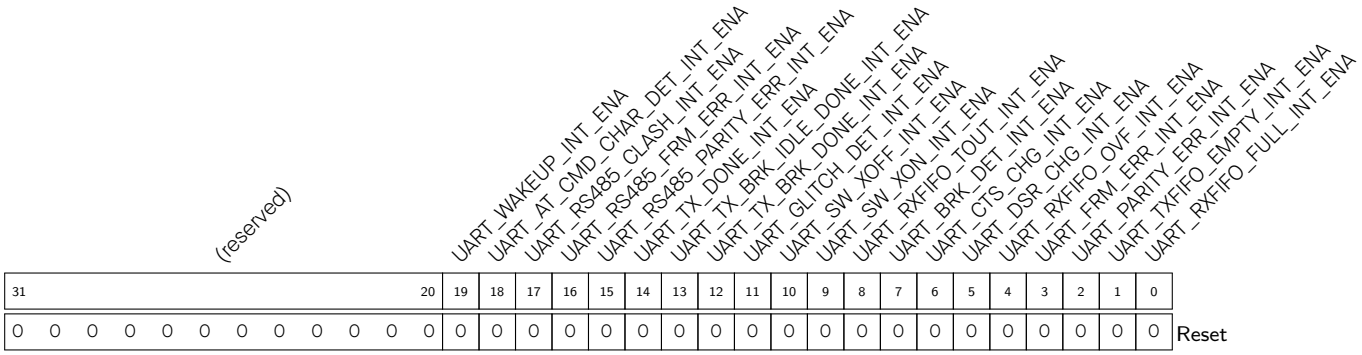
UART_RS485_FRM_ERR_INT_ST UART_RS485_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RS485_CLASH_INT_ST UART_RS485_CLASH_INT 的屏蔽中断状态。(RO)

UART_AT_CMD_CHAR_DET_INT_ST UART_AT_CMD_CHAR_DET_INT 的屏蔽中断状态。(RO)

UART_WAKEUP_INT_ST UART_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 45.5. UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA 写 1 使能 UART_RXFIFO_FULL_INT。(R/W)

UART_TXFIFO_EMPTY_INT_ENA 写 1 使能 UART_TXFIFO_EMPTY_INT。(R/W)

UART_PARITY_ERR_INT_ENA 写 1 使能 UART_PARITY_ERR_INT。(R/W)

UART_FRM_ERR_INT_ENA 写 1 使能 UART_FRM_ERR_INT。(R/W)

UART_RXFIFO_OVF_INT_ENA 写 1 使能 UART_RXFIFO_OVF_INT。(R/W)

UART_DSR_CHG_INT_ENA 写 1 使能 UART_DSR_CHG_INT。(R/W)

UART_CTS_CHG_INT_ENA 写 1 使能 UART_CTS_CHG_INT。(R/W)

UART_BRK_DET_INT_ENA 写 1 使能 UART_BRK_DET_INT。(R/W)

UART_RXFIFO_TOUT_INT_ENA 写 1 使能 UART_RXFIFO_TOUT_INT。(R/W)

UART_SW_XON_INT_ENA 写 1 使能 UART_SW_XON_INT。(R/W)

UART_SW_XOFF_INT_ENA 写 1 使能 UART_SW_XOFF_INT。(R/W)

UART_GLITCH_DET_INT_ENA 写 1 使能 UART_GLITCH_DET_INT。(R/W)

UART_TX_BRK_DONE_INT_ENA 写 1 使能 UART_TX_BRK_DONE_INT。(R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA 写 1 使能 UART_TX_BRK_IDLE_DONE_INT。(R/W)

UART_TX_DONE_INT_ENA 写 1 使能 UART_TX_DONE_INT。(R/W)

UART_RS485_PARITY_ERR_INT_ENA 写 1 使能 UART_RS485_PARITY_ERR_INT。(R/W)

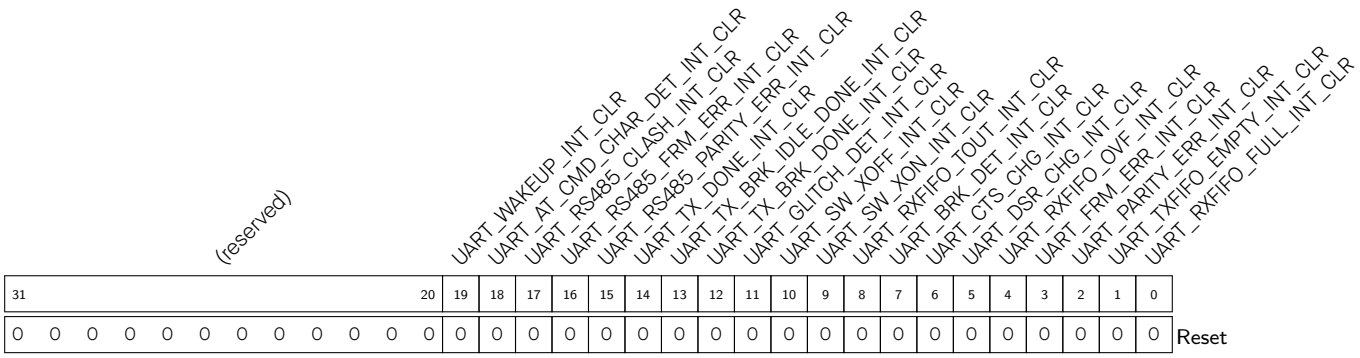
UART_RS485_FRM_ERR_INT_ENA 写 1 使能 UART_RS485_FRM_ERR_INT。(R/W)

UART_RS485_CLASH_INT_ENA 写 1 使能 UART_RS485_CLASH_INT。(R/W)

UART_AT_CMD_CHAR_DET_INT_ENA 写 1 使能 UART_AT_CMD_CHAR_DET_INT。(R/W)

UART_WAKEUP_INT_ENA 写 1 使能 UART_WAKEUP_INT。(R/W)

Register 45.6. UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR 写 1 清除 UART_RXFIFO_FULL_INT。(WT)

UART_TXFIFO_EMPTY_INT_CLR 写 1 清除 UART_TXFIFO_EMPTY_INT。(WT)

UART_PARITY_ERR_INT_CLR 写 1 清除 UART_PARITY_ERR_INT。(WT)

UART_FRM_ERR_INT_CLR 写 1 清除 UART_FRM_ERR_INT。(WT)

UART_RXFIFO_OVF_INT_CLR 写 1 清除 UART_RXFIFO_OVF_INT。(WT)

UART_DSR_CHG_INT_CLR 写 1 清除 UART_DSR_CHG_INT。(WT)

UART_CTS_CHG_INT_CLR 写 1 清除 UART_CTS_CHG_INT。(WT)

UART_BRK_DET_INT_CLR 写 1 清除 UART_BRK_DET_INT。(WT)

UART_RXFIFO_TOUT_INT_CLR 写 1 清除 UART_RXFIFO_TOUT_INT。(WT)

UART_SW_XON_INT_CLR 写 1 清除 UART_SW_XON_INT。(WT)

UART_SW_XOFF_INT_CLR 写 1 清除 UART_SW_XOFF_INT。(WT)

UART_GLITCH_DET_INT_CLR 写 1 清除 UART_GLITCH_DET_INT。(WT)

UART_TX_BRK_DONE_INT_CLR 写 1 清除 UART_TX_BRK_DONE_INT。(WT)

UART_TX_BRK_IDLE_DONE_INT_CLR 写 1 清除 UART_TX_BRK_IDLE_DONE_INT。(WT)

UART_TX_DONE_INT_CLR 写 1 清除 UART_TX_DONE_INT。(WT)

UART_RS485_PARITY_ERR_INT_CLR 写 1 清除 UART_RS485_PARITY_ERR_INT。(WT)

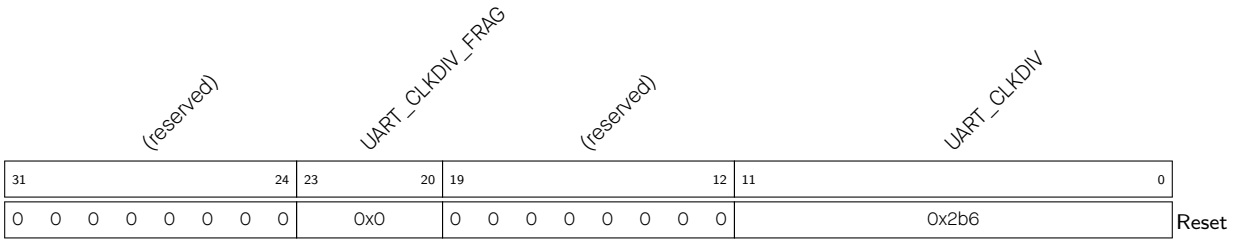
UART_RS485_FRM_ERR_INT_CLR 写 1 清除 UART_RS485_FRM_ERR_INT。(WT)

UART_RS485_CLASH_INT_CLR 写 1 清除 UART_RS485_CLASH_INT。(WT)

UART_AT_CMD_CHAR_DET_INT_CLR 写 1 清除 UART_AT_CMD_CHAR_DET_INT。(WT)

UART_WAKEUP_INT_CLR 写 1 清除 UART_WAKEUP_INT。(WT)

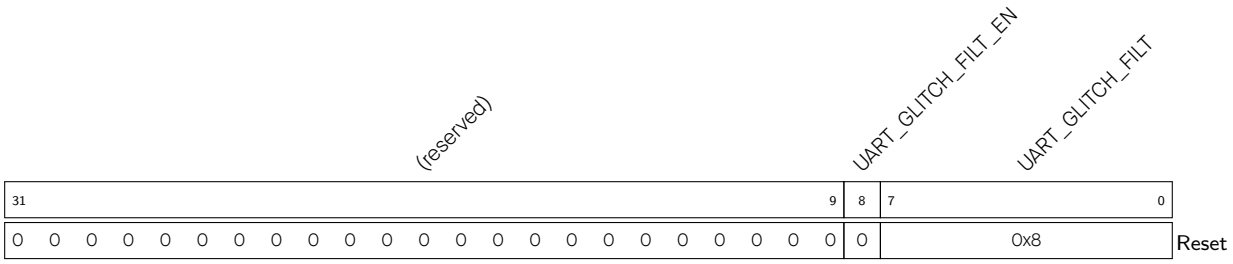
Register 45.7. UART_CLKDIV_SYNC_REG (0x0014)



UART_CLKDIV 配置波特率分频系数的整数部分。(R/W)

UART_CLKDIV_FRAG 配置波特率分频系数的小数部分。(R/W)

Register 45.8. UART_RX_FILT_REG (0x0018)



UART_GLITCH_FILT 配置滤波长度。

单位：UART Core 时钟周期。

宽度小于该数值的输入脉冲会被忽略。(R/W)

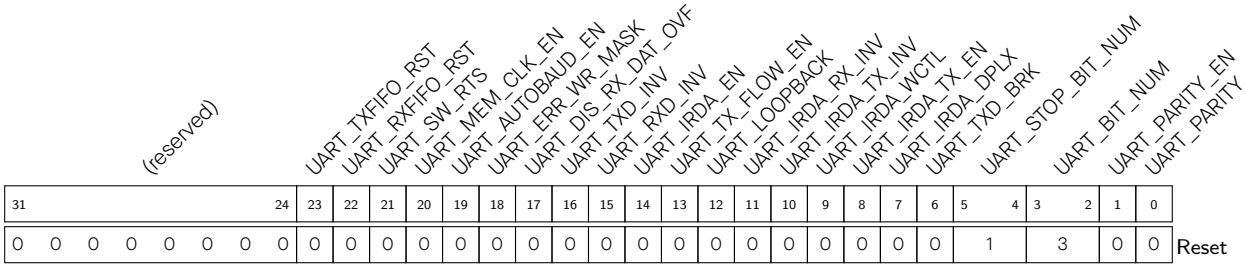
UART_GLITCH_FILT_EN 配置是否开启接收信号滤波器。

0：关闭

1：开启

(R/W)

Register 45.9. UART_CONF0_SYNC_REG (0x0020)



UART_PARITY 配置奇偶检验方式。

0：偶校验

1：奇校验

(R/W)

UART_PARITY_EN 配置是否开启 UART 奇偶校验。

0：关闭

1：开启

(R/W)

UART_BIT_NUM 配置数据位长度。

0：5 位

1：6 位

2：7 位

3：8 位

(R/W)

UART_STOP_BIT_NUM 配置停止位长度。

0: 无效值，没有作用

1: 1 位

2: 1.5 位

3: 2 位

(R/W)

UART_TXD_BRK 配置完成数据发送后，发送器是否发送 NULL。

0：不发送

1：发送

(R/W)

UART_IRDA_DPLX 配置是否开启 IrDA 回环测试模式。

0：关闭

1：开启

(R/W)

UART_IRDA_TX_EN 配置是否开启 IrDA 发送器。

0：关闭

1：开启

(R/W)

见下页...

Register 45.9. UART_CONF0_SYNC_REG (0x0020)

接上页...

UART_IRDA_WCTL 配置 IrDA 发送器的第 11 位。

0：该位为 0

1：该位与第 10 位相同

(R/W)

UART_IRDA_TX_INV 配置是否翻转 IrDA 发送器的电平。

0：不翻转

1：翻转

(R/W)

UART_IRDA_RX_INV 配置是否翻转 IrDA 接收器的电平。

0：不翻转

1：翻转

(R/W)

UART_LOOPBACK 配置是否开启 UART 回环测试模式。

0：关闭

1：开启

(R/W)

UART_TX_FLOW_EN 配置是否开启发送器的流控功能。

0：关闭

1：开启

(R/W)

UART_IRDA_EN 配置是否开启 IrDA。

0：关闭

1：开启

(R/W)

UART_RXD_INV 配置是否翻转 UART RXD 信号电平。

0：不翻转

1：翻转

(R/W)

UART_TXD_INV 配置是否翻转 UART TXD 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DIS_RX_DAT_OVF 配置是否关闭 UART 接收器数据溢出检测。

0：开启

1：关闭

(R/W)

见下页...

Register 45.9. UART_CONF0_SYNC_REG (0x0020)

接上页...

UART_ERR_WR_MASK 配置接收数据有错误时，是否仍将其存储至 FIFO。

0：存储

1：不存储

(R/W)

UART_AUTOBAUD_EN 配置是否开启波特率检测。

0：关闭

1：开启

(R/W)

UART_MEM_CLK_EN 配置是否开启 UART 存储器门控。

0：关闭

1：开启

(R/W)

UART_SW_RTS 配置软件流控使用的 RTS 信号。

0：禁止发送端发送数据

1：允许发送端发送数据

(R/W)

UART_RXFIFO_RST 配置是否复位 UART RX FIFO。

0：不复位

1：复位

(R/W)

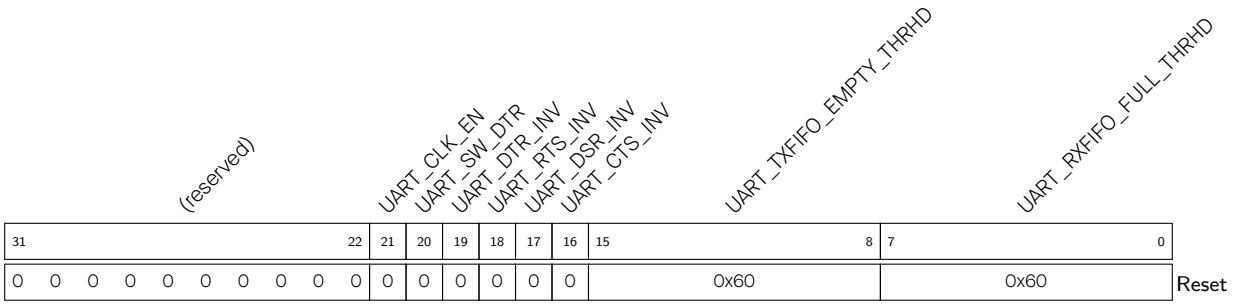
UART_TXFIFO_RST 配置是否复位 UART TX FIFO。

0：不复位

1：复位

(R/W)

Register 45.10. UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD 配置 RX FIFO 为满的阈值。

单位：字节。(R/W)

UART_TXFIFO_EMPTY_THRHD 配置 TX FIFO 为空的阈值。

单位：字节。(R/W)

UART_CTS_INV 配置是否翻转 UART CTS 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DSR_INV 配置是否翻转 UART DSR 信号电平。

0：不翻转

1：翻转

(R/W)

UART_RTS_INV 配置是否翻转 UART RTS 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DTR_INV 配置是否翻转 UART DTR 信号电平。

0：不翻转

1：翻转

(R/W)

UART_SW_DTR 配置软件流控使用的 DTR 信号。

0: 发送数据未准备完毕，处于不可用状态

1: 发送数据准备完毕，处于可用状态

(R/W)

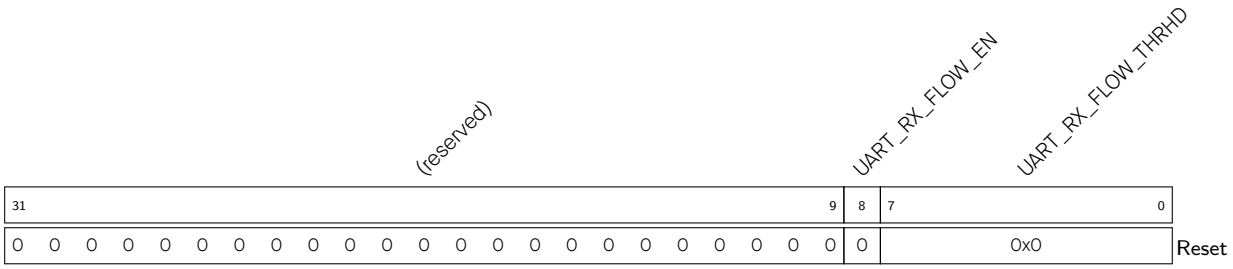
UART_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 45.11. UART_HWFC_CONF_SYNC_REG (0x002C)



UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大数。

单位：字节。(R/W)

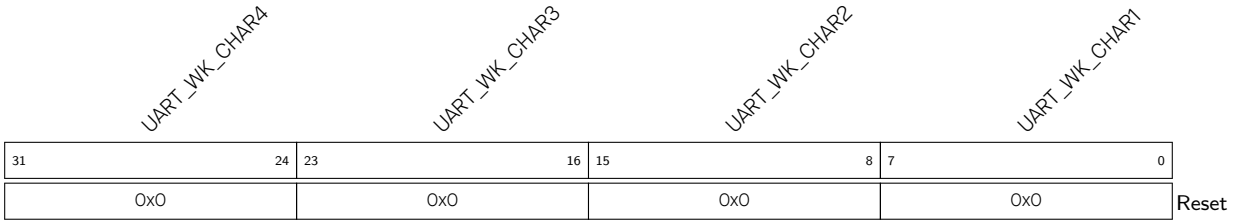
UART_RX_FLOW_EN 配置是否开启 UART 接收器。

0：关闭

1：开启

(R/W)

Register 45.12. UART_SLEEP_CONF0_REG (0x0030)



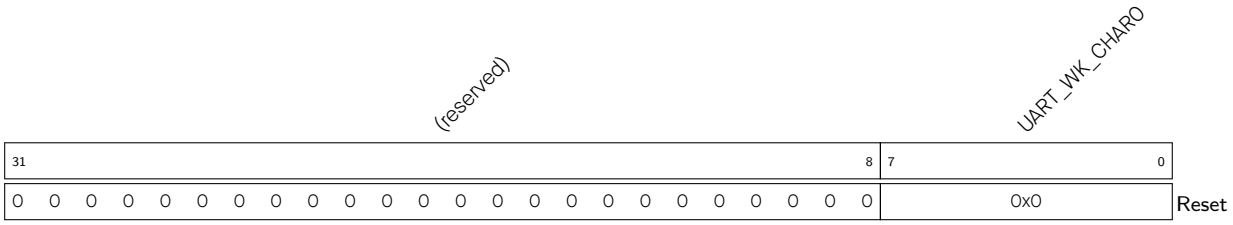
UART_WK_CHAR1 配置唤醒芯片的特定字符 1。(R/W)

UART_WK_CHAR2 配置唤醒芯片的特定字符 2。(R/W)

UART_WK_CHAR3 配置唤醒芯片的特定字符 3。(R/W)

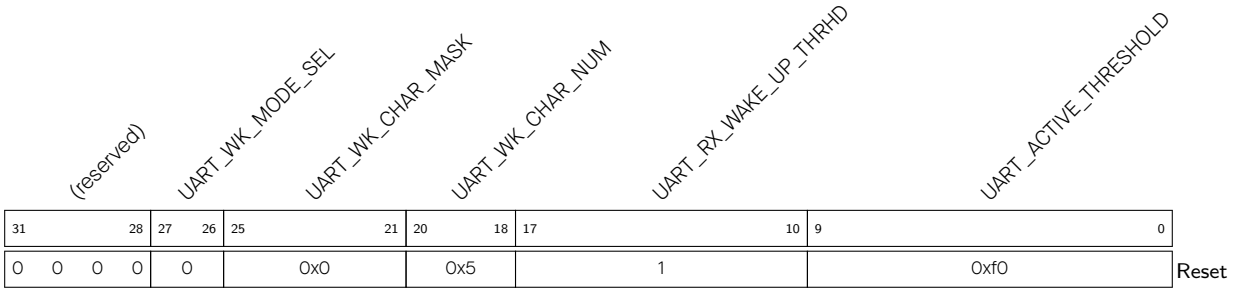
UART_WK_CHAR4 配置唤醒芯片的特定字符 4。(R/W)

Register 45.13. UART_SLEEP_CONF1_REG (0x0034)



UART_WK_CHAR0 配置唤醒芯片的特定字符 0。(R/W)

Register 45.14. UART_SLEEP_CONF2_REG (0x0038)



UART_ACTIVE_THRESHOLD 配置唤醒模式 0 唤醒芯片所需的 RXD 沿变化次数。(R/W)

UART_RX_WAKE_UP_THRHD 配置唤醒模式 1 唤醒芯片所需接收的数据量。

单位：字节。(R/W)

UART_WK_CHAR_NUM 配置选取多少个唤醒字符。(R/W)

UART_WK_CHAR_MASK 配置是否屏蔽唤醒字符。

0：不屏蔽

1：屏蔽

(R/W)

UART_WK_MODE_SEL 配置选取哪个唤醒模式。不同模式的含义详见 45.4.8 唤醒 小节。

0：唤醒模式 0

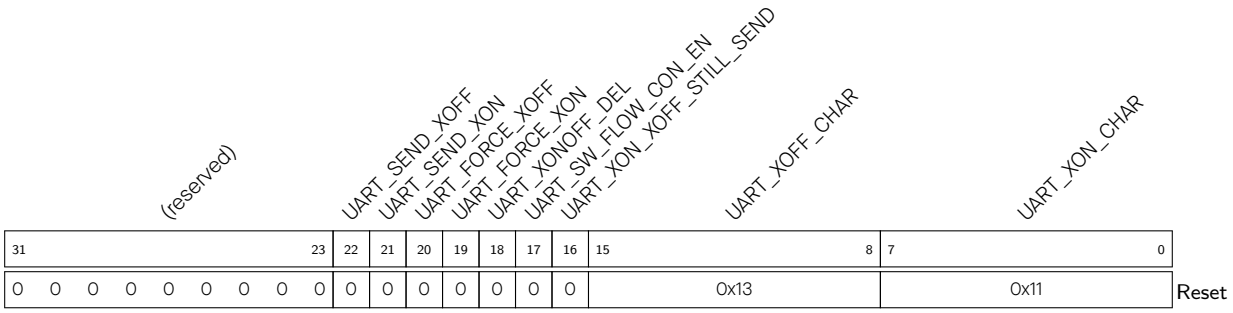
1：唤醒模式 1

2：唤醒模式 2

3：唤醒模式 3

(R/W)

Register 45.15. UART_SWFC_CONF0_SYNC_REG (0x003C)



UART_XON_CHAR 配置 XON 流控字符。(R/W)

UART_XOFF_CHAR 配置 XOFF 流控字符。(R/W)

UART_XON_XOFF_STILL_SEND 配置 UART 发送器关闭时，是否仍能发送 XON 或 XOFF 字符。

0：不能发送

1：可以发送

(R/W)

UART_SW_FLOW_CON_EN 配置是否开启软件流控。

0：关闭

1：开启

(R/W)

UART_XONOFF_DEL 配置是否移除接收数据中的流控字符。

0：不移除

1：移除

(R/W)

UART_FORCE_XON 配置是否让发送器继续发送数据。

0：不发送

1：继续发送

(R/W)

UART_FORCE_XOFF 配置是否让发送器停止发送数据。

0：不停止

1：停止

(R/W)

UART_SEND_XON 配置是否发送 XON 字符。

0：不发送

1：发送

(R/W/SS/SC)

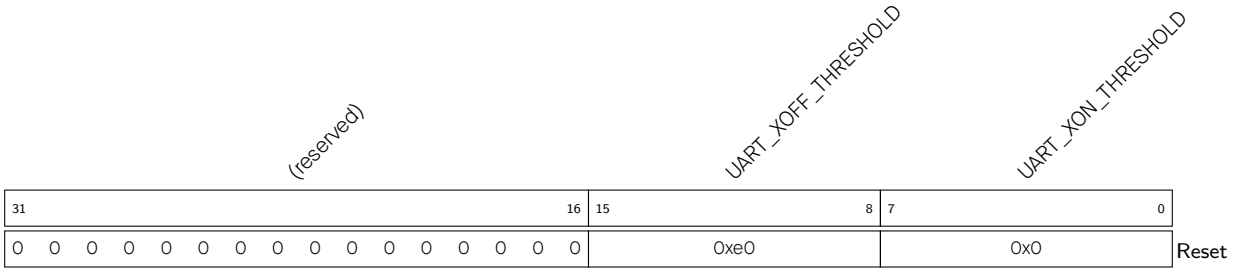
UART_SEND_XOFF 配置是否发送 XOFF 字符。

0：不发送

1：发送

(R/W/SS/SC)

Register 45.16. UART_SWFC_CONF1_REG (0x0040)



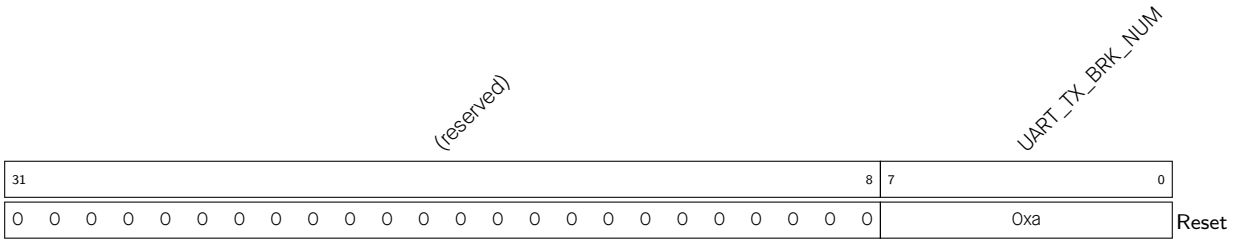
UART_XON_THRESHOLD 配置软件流控时，发送 XON 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

UART_XOFF_THRESHOLD 配置软件流控时，发送 XOFF 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

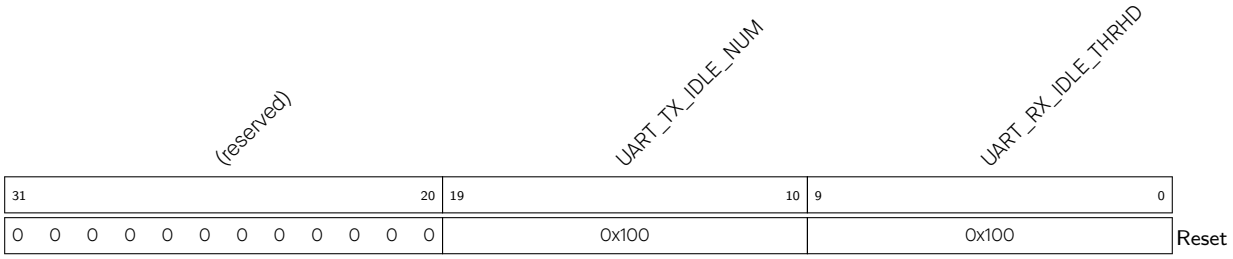
Register 45.17. UART_TXBRK_CONF_SYNC_REG (0x0044)



UART_TX_BRK_NUM 配置完成数据发送后待发 NULL 字符的数量。

仅在 UART_TXD_BRK 为 1 时有效。(R/W)

Register 45.18. UART_IDLE_CONF_SYNC_REG (0x0048)



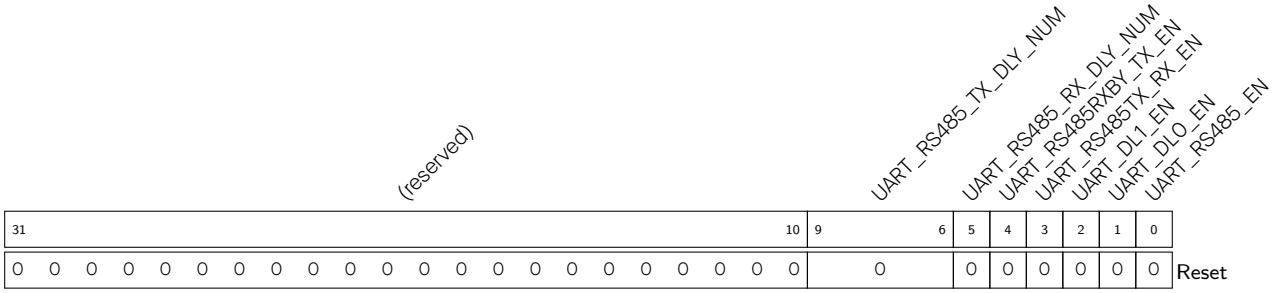
UART_RX_IDLE_THRHD 配置接收一字节数据时间过长、产生帧结束信号的阈值。

单位：位时间（传输一位所需的时间）。(R/W)

UART_TX_IDLE_NUM 配置两次数据传输的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 45.19. UART_RS485_CONF_SYNC_REG (0x004C)



UART_RS485_EN 配置是否开启 RS485 模式。

0：关闭

1：开启

(R/W)

UART_DLO_EN 配置是否在起始位之前增加一位延时。

0：不增加

1：增加

(R/W)

UART_DL1_EN 配置是否在停止位之后增加一位延时。

0：不增加

1：增加

(R/W)

UART_RS485TX_RX_EN 配置发送器在 RS485 模式下发送数据时，是否开启接收器接收数据。

0：关闭

1：开启

(R/W)

UART_RS485RXBY_TX_EN 配置 RS485 接收器线路繁忙时，是否开启 RS485 发送器发送数据。

0：关闭

1：开启

(R/W)

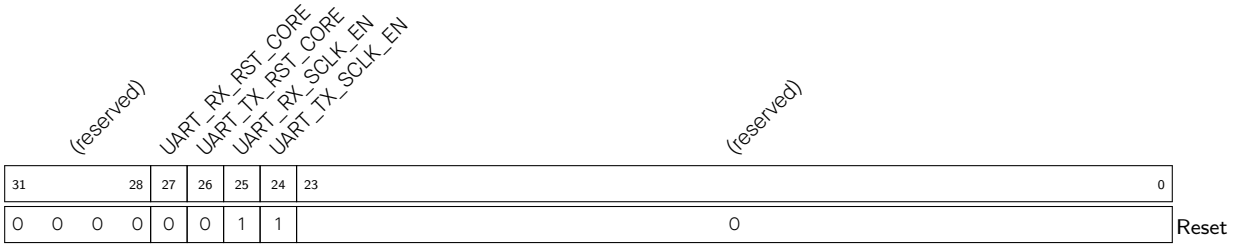
UART_RS485_RX_DLY_NUM 配置接收器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

UART_RS485_TX_DLY_NUM 配置发送器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

Register 45.20. UART_CLK_CONF_REG (0x0088)



UART_TX_SCLK_EN 配置是否开启 UART TX 时钟。

0：关闭

1：开启

(R/W)

UART_RX_SCLK_EN 配置是否开启 UART RX 时钟。

0：关闭

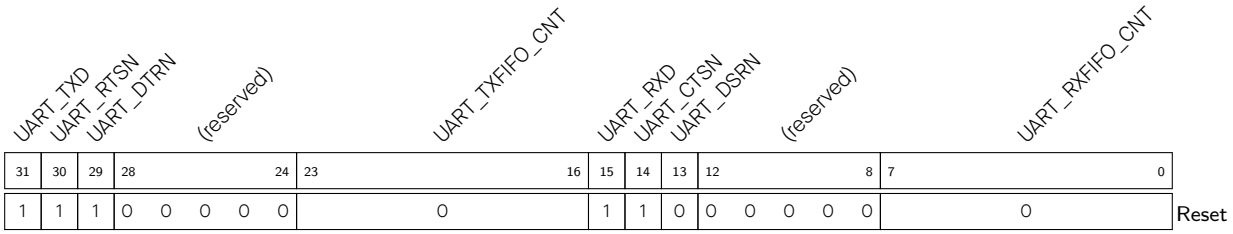
1：开启

(R/W)

UART_TX_RST_CORE 向此位先写 1 后写 0，复位 UART TX。(R/W)

UART_RX_RST_CORE 向此位先写 1 后写 0，复位 UART RX。(R/W)

Register 45.21. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT 表示 RX FIFO 中有效数据量。

单位：字节。(RO)

UART_DSRN 表示内部 UART DSR 信号的电平值。(RO)

UART_CTSN 表示内部 UART CTS 信号的电平值。(RO)

UART_RXD 表示内部 UART RXD 信号的电平值。(RO)

UART_TXFIFO_CNT 表示 TX FIFO 中的数据量。

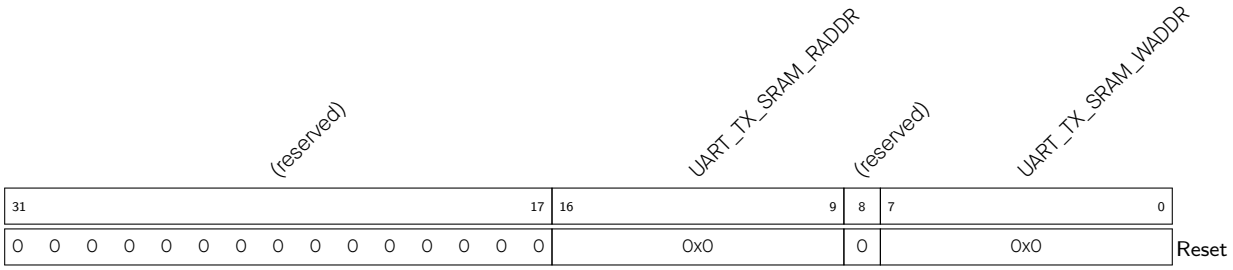
单位：字节。(RO)

UART_DTRN 表示内部 UART DTR 信号的电平。(RO)

UART_RTSN 表示内部 UART RTS 信号的电平。(RO)

UART_TXD 表示内部 UART TXD 信号的电平。(RO)

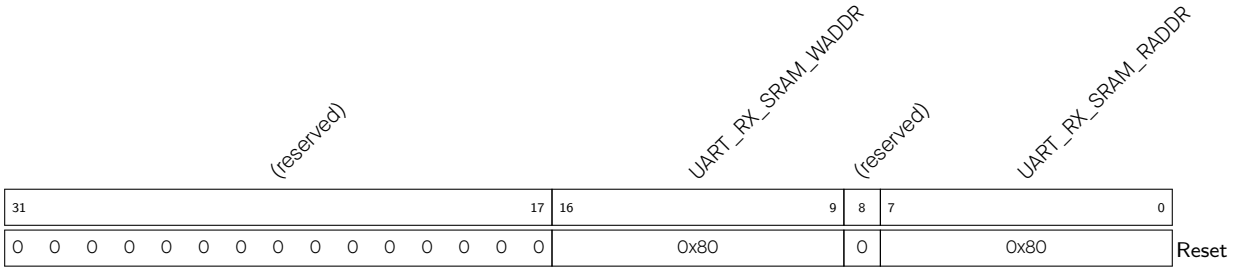
Register 45.22. UART_MEM_TX_STATUS_REG (0x0068)



UART_TX_SRAM_WADDR 表示写 TX FIFO 的偏移地址。(RO)

UART_TX_SRAM_RADDR 表示读 TX FIFO 的偏移地址。(RO)

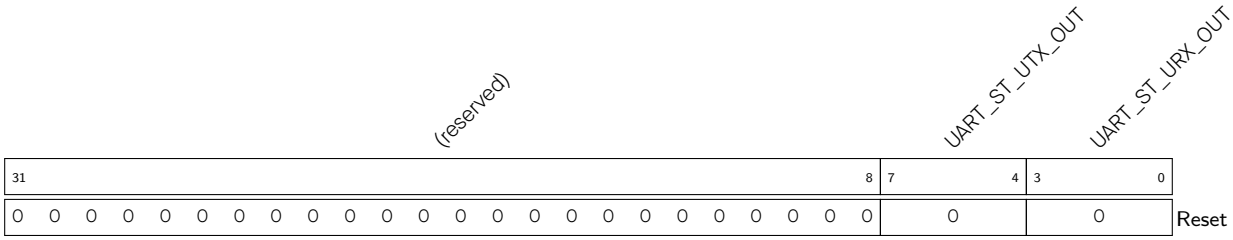
Register 45.23. UART_MEM_RX_STATUS_REG (0x006C)



UART_RX_SRAM_RADDR 表示读 RX FIFO 的偏移地址。(RO)

UART_RX_SRAM_WADDR 表示写 RX FIFO 的偏移地址。(RO)

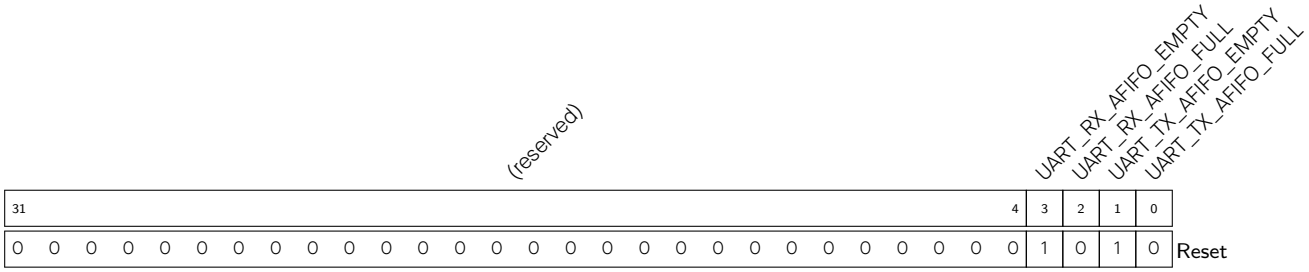
Register 45.24. UART_FSM_STATUS_REG (0x0070)



UART_ST_URX_OUT 表示 UART 接收器的状态。(RO)

UART_ST_UTX_OUT 表示 UART 发送器的状态。(RO)

Register 45.25. UART_AFIFO_STATUS_REG (0x0090)



UART_TX_AFIFO_FULL 表示 APB TX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

UART_TX_AFIFO_EMPTY 表示 APB TX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

UART_RX_AFIFO_FULL 表示 APB RX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

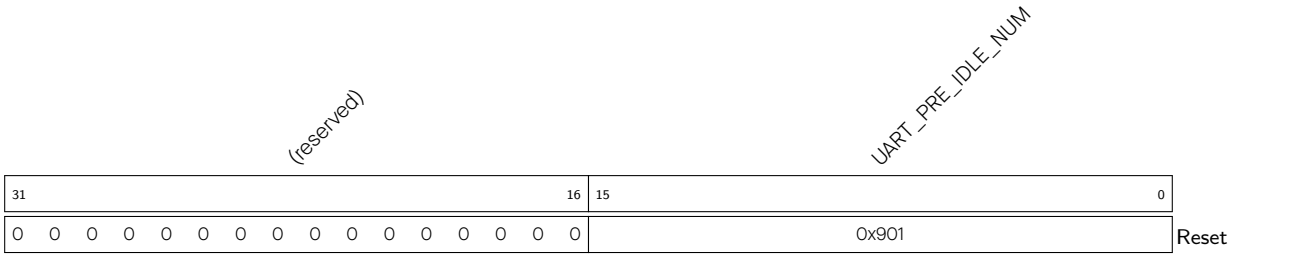
UART_RX_AFIFO_EMPTY 表示 APB RX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

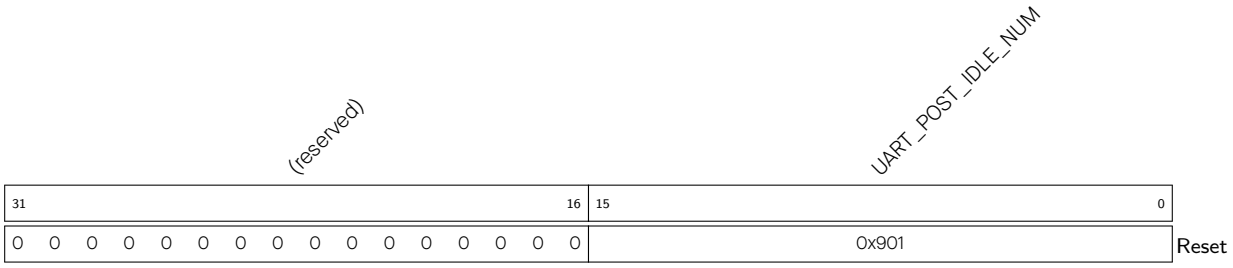
Register 45.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



UART_PRE_IDLE_NUM 配置接收器接收到第一个 AT_CMD 之前的空闲时间。

单位：位时间（传输一位所需的时间）。(R/W)

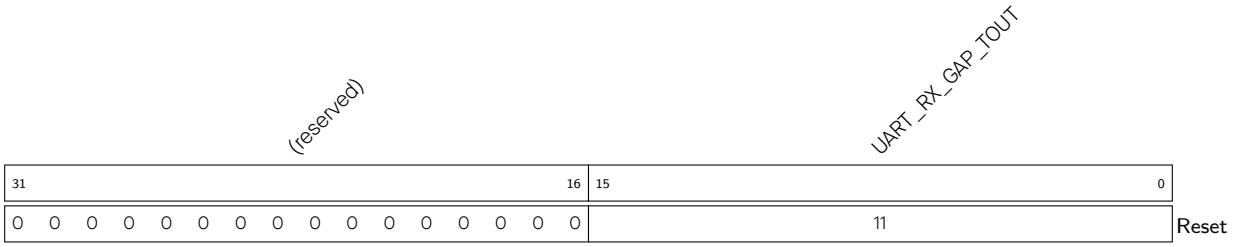
Register 45.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



UART_POST_IDLE_NUM 配置最后一个 AT_CMD 和后续数据之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

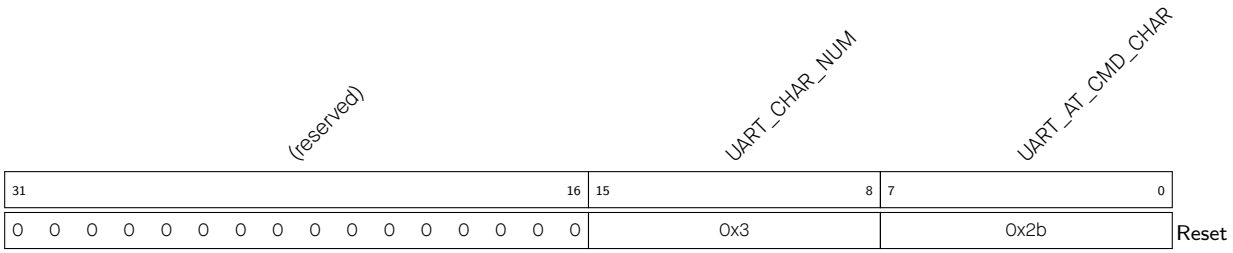
Register 45.28. UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



UART_RX_GAP_TOUT 配置 AT_CMD 之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

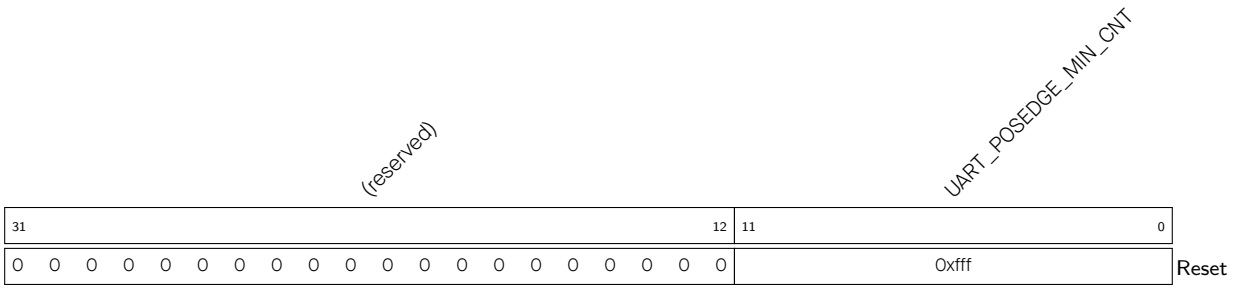
Register 45.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)



UART_AT_CMD_CHAR 配置 AT_CMD 字符。(R/W)

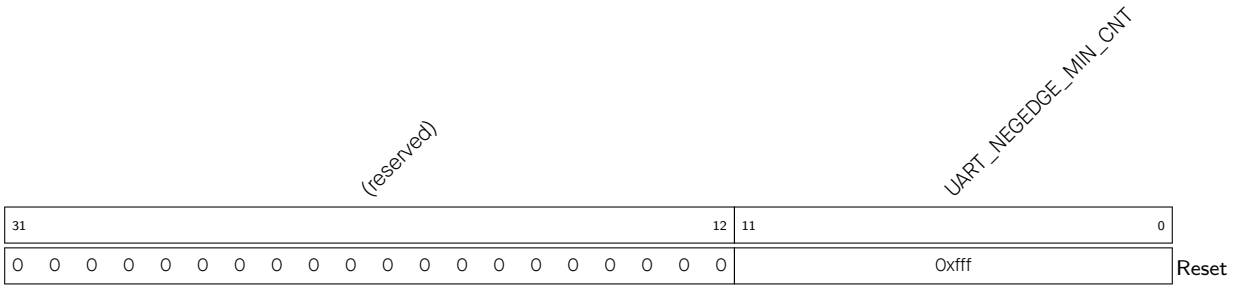
UART_CHAR_NUM 配置接收器可连续接收的 AT_CMD 字符数量。(R/W)

Register 45.30. UART_POSPULSE_REG (0x0074)



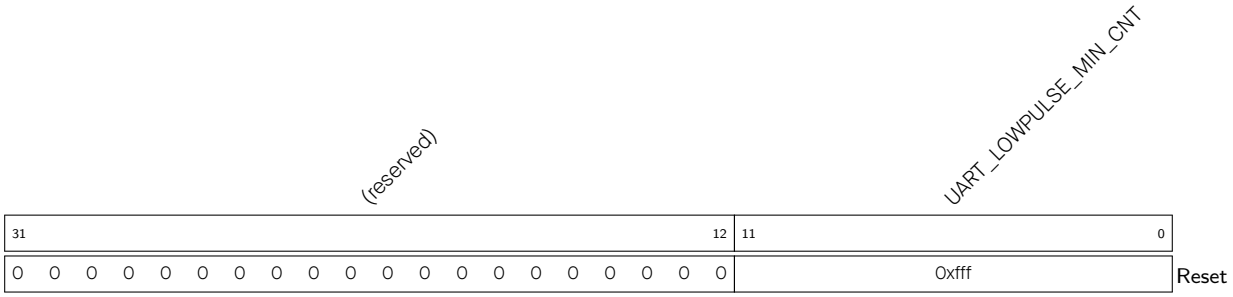
UART_POSEDGE_MIN_CNT 表示两个上升沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 45.31. UART_NEGPULSE_REG (0x0078)



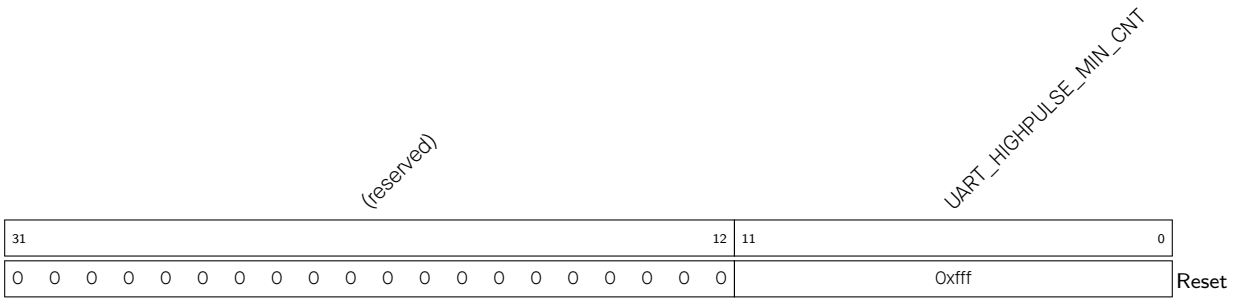
UART_NEGEDGE_MIN_CNT 两个下降沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 45.32. UART_LOWPULSE_REG (0x007C)



UART_LOWPULSE_MIN_CNT 表示低电平脉冲的最短持续时间，用于波特率检测。
单位：APB_CLK 时钟周期。(RO)

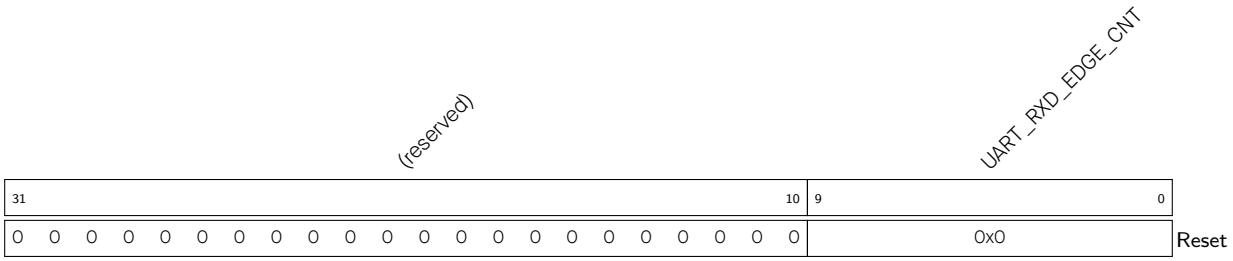
Register 45.33. UART_HIGHPULSE_REG (0x0080)



UART_HIGHPULSE_MIN_CNT 表示高电平脉冲最长持续时间，用于波特率检测。

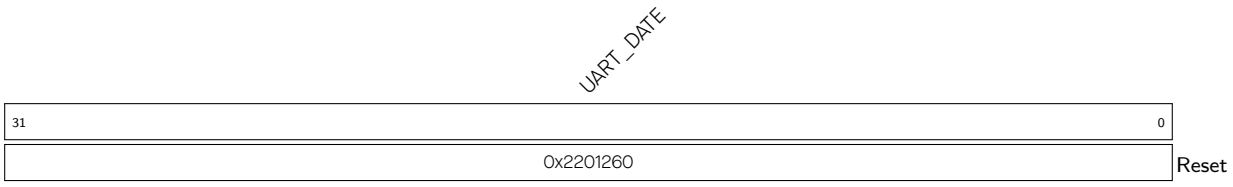
单位：APB_CLK 时钟周期。(RO)

Register 45.34. UART_RXD_CNT_REG (0x0084)



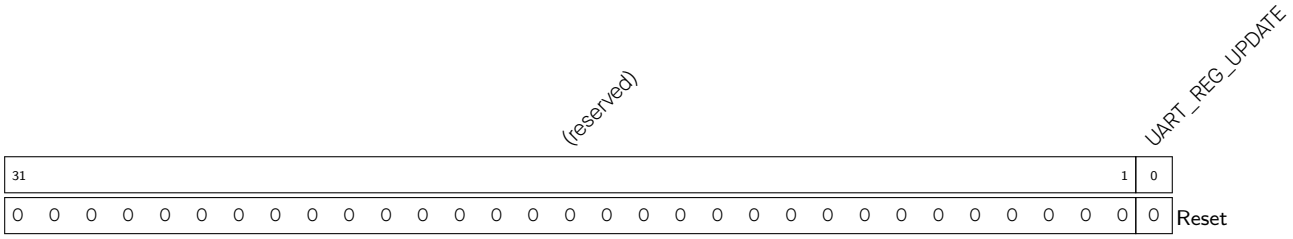
UART_RXD_EDGE_CNT 表示 RXD 沿变化的次数。用于波特率检测。(RO)

Register 45.35. UART_DATE_REG (0x008C)



UART_DATE 版本控制寄存器。(R/W)

Register 45.36. UART_REG_UPDATE_REG (0x0098)



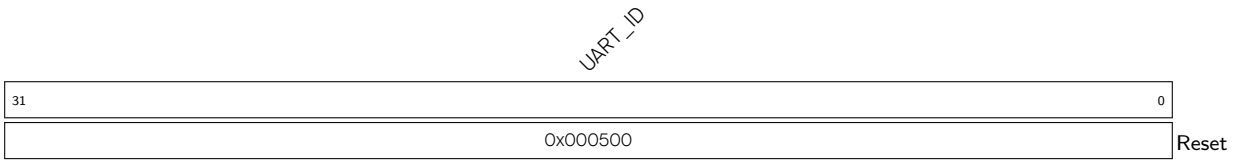
UART_REG_UPDATE 配置是否同步寄存器

0：不同步

1：同步

(R/W/SC)

Register 45.37. UART_ID_REG (0x009C)



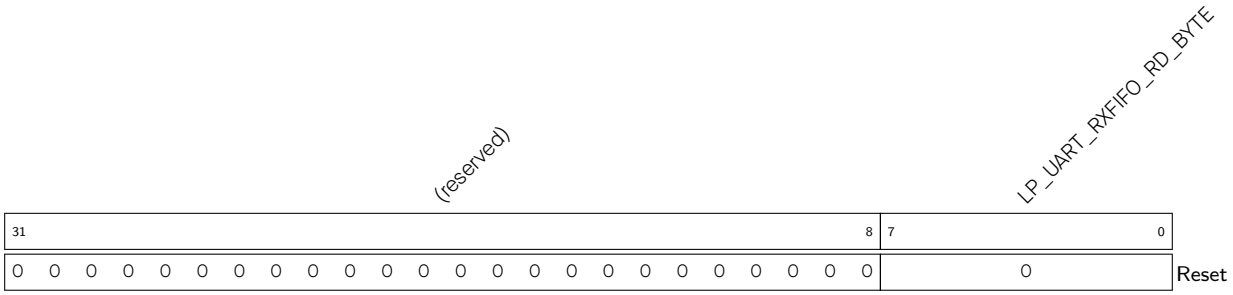
UART_ID 配置 UART ID。(R/W)

45.8.2 LP UART 寄存器

本小节的所有地址均为相对于 LP UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

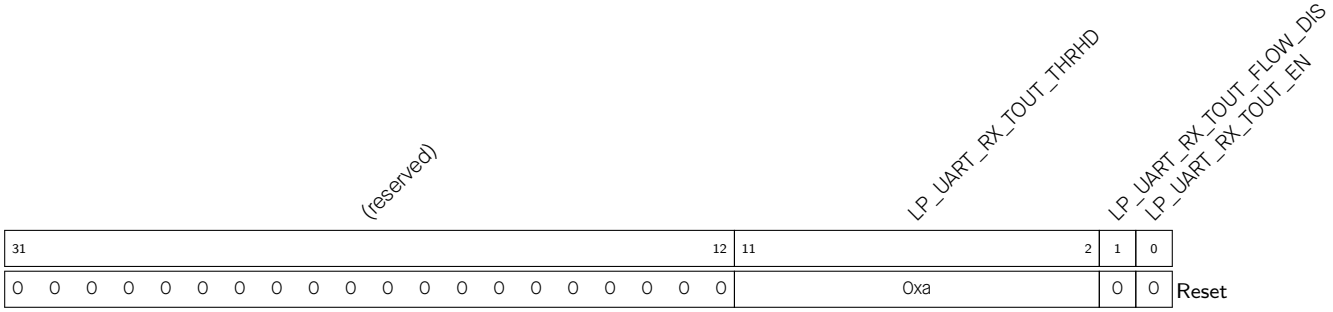
Register 45.38. LP_UART_FIFO_REG (0x0000)



LP_UART_RXFIFO_RD_BYTE 表示 LP UART *n* 从 FIFO 读取的数据。

单位：字节。(RO)

Register 45.39. LP_UART_TOUT_CONF_SYNC_REG (0x0064)



LP_UART_RX_TOUT_EN 配置是否开启 LP UART 接收器的超时功能。

0：关闭

1：开启

(R/W)

LP_UART_RX_TOUT_FLOW_DIS 配置开启硬件流控时是否关闭 idle 状态计数器计数。

0：无效值，没有作用

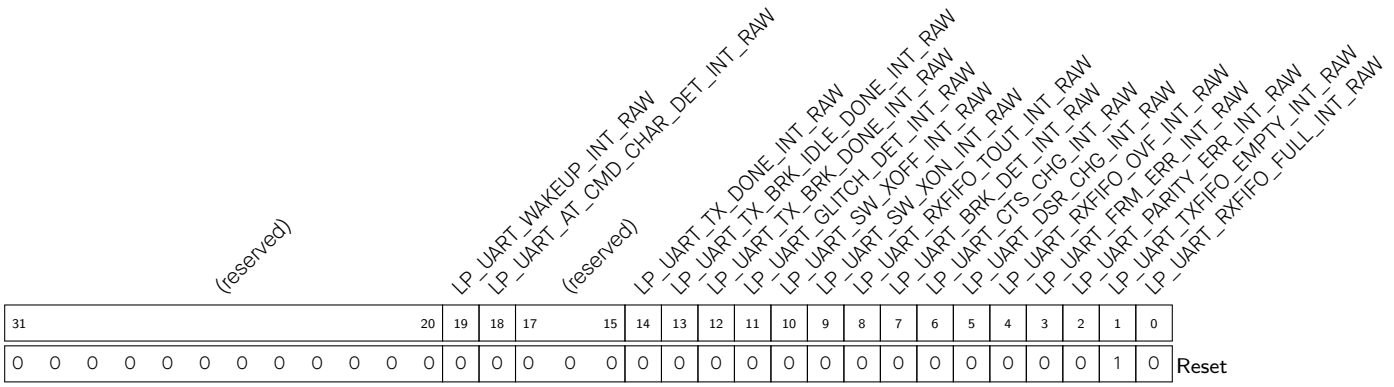
1：关闭

(R/W)

LP_UART_RX_TOUT_THRHD 配置触发超时前总线可保持空闲状态的时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 45.40. LP_UART_INT_RAW_REG (0x0004)



LP_UART_RXFIFO_FULL_INT_RAW LP_UART_RXFIFO_FULL_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TXFIFO_EMPTY_INT_RAW LP_UART_TXFIFO_EMPTY_INT 的原始中断状态。(R/ WTC/SS)

LP_UART_PARITY_ERR_INT_RAW LP_UART_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

LP_UART_FRM_ERR_INT_RAW LP_UART_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

LP_UART_RXFIFO_OVF_INT_RAW LP_UART_RXFIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

LP_UART_DSR_CHG_INT_RAW LP_UART_DSR_CHG_INT 的原始中断状态。(R/WTC/SS)

LP_UART_CTS_CHG_INT_RAW LP_UART_CTS_CHG_INT 的原始中断状态。(R/WTC/SS)

LP_UART_BRK_DET_INT_RAW LP_UART_BRK_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_RXFIFO_TOUT_INT_RAW UART_RXFIFO_TOUT_INT 的原始中断状态。(R/WTC/SS)

LP_UART_SW_XON_INT_RAW LP_UART_SW_XON_INT 的原始中断状态。(R/WTC/SS)

LP_UART_SW_XOFF_INT_RAW LP_UART_SW_XOFF_INT 的原始中断状态。(R/WTC/SS)

LP_UART_GLITCH_DET_INT_RAW LP_UART_GLITCH_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TX_BRK_DONE_INT_RAW UART_TX_BRK_DONE_INT 的原始中断状态。(R/WTC/SS)

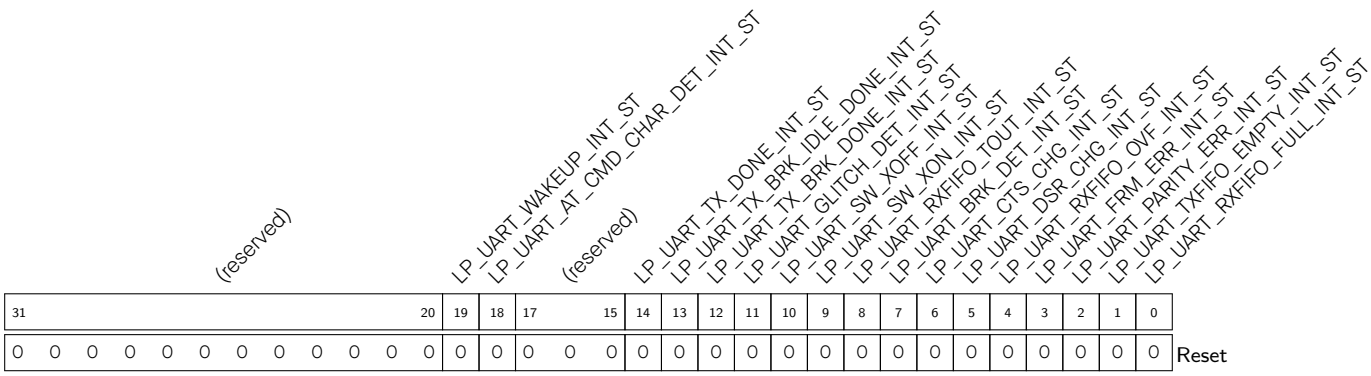
LP_UART_TX_BRK_IDLE_DONE_INT_RAW LP_UART_TX_BRK_IDLE_DONE_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TX_DONE_INT_RAW LP_UART_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

LP_UART_AT_CMD_CHAR_DET_INT_RAW LP_UART_AT_CMD_CHAR_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_WAKEUP_INT_RAW LP_UART_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 45.41. LP_UART_INT_ST_REG (0x0008)



LP_UART_RXFIFO_FULL_INT_ST LP_UART_RXFIFO_FULL_INT 的屏蔽中断状态。(RO)

LP_UART_TXFIFO_EMPTY_INT_ST LP_UART_TXFIFO_EMPTY_INT 的屏蔽中断状态。(RO)

LP_UART_PARITY_ERR_INT_ST LP_UART_PARITY_ERR_INT 的屏蔽中断状态。(RO)

LP_UART_FRM_ERR_INT_ST LP_UART_FRM_ERR_INT 的屏蔽中断状态。(RO)

LP_UART_RXFIFO_OVF_INT_ST LP_UART_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

LP_UART_DSR_CHG_INT_ST LP_UART_DSR_CHG_INT 的屏蔽中断状态。(RO)

LP_UART_CTS_CHG_INT_ST LP_UART_CTS_CHG_INT 的屏蔽中断状态。(RO)

LP_UART_BRK_DET_INT_ST LP_UART_BRK_DET_INT 的屏蔽中断状态。(RO)

LP_UART_RXFIFO_TOUT_INT_ST LP_UART_RXFIFO_TOUT_INT 的屏蔽中断状态。(RO)

LP_UART_SW_XON_INT_ST LP_UART_SW_XON_INT 的屏蔽中断状态。(RO)

LP_UART_SW_XOFF_INT_ST LP_UART_SW_XOFF_INT 的屏蔽中断状态。(RO)

LP_UART_GLITCH_DET_INT_ST LP_UART_GLITCH_DET_INT 的屏蔽中断状态。(RO)

LP_UART_TX_BRK_DONE_INT_ST LP_UART_TX_BRK_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_TX_BRK_IDLE_DONE_INT_ST LP_UART_TX_BRK_IDLE_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_TX_DONE_INT_ST LP_UART_TX_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_AT_CMD_CHAR_DET_INT_ST LP_UART_AT_CMD_CHAR_DET_INT 的屏蔽中断状态。(RO)

LP_UART_WAKEUP_INT_ST LP_UART_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 45.42. LP_UART_INT_ENA_REG (0x000C)



LP_UART_RXFIFO_FULL_INT_ENA 写 1 使能 LP_UART_RXFIFO_FULL_INT。(R/W)

LP_UART_TXFIFO_EMPTY_INT_ENA 写 1 使能 LP_UART_TXFIFO_EMPTY_INT。(R/W)

LP_UART_PARITY_ERR_INT_ENA 写 1 使能 LP_UART_PARITY_ERR_INT。(R/W)

LP_UART_FRM_ERR_INT_ENA 写 1 使能 LP_UART_FRM_ERR_INT。(R/W)

LP_UART_RXFIFO_OVF_INT_ENA 写 1 使能 LP_UART_RXFIFO_OVF_INT。(R/W)

LP_UART_DSR_CHG_INT_ENA 写 1 使能 LP_UART_DSR_CHG_INT。(R/W)

LP_UART_CTS_CHG_INT_ENA 写 1 使能 LP_UART_CTS_CHG_INT。(R/W)

LP_UART_BRK_DET_INT_ENA 写 1 使能 LP_UART_BRK_DET_INT。(R/W)

LP_UART_RXFIFO_TOUT_INT_ENA 写 1 使能 LP_UART_RXFIFO_TOUT_INT。(R/W)

LP_UART_SW_XON_INT_ENA 写 1 使能 LP_UART_SW_XON_INT。(R/W)

LP_UART_SW_XOFF_INT_ENA 写 1 使能 LP_UART_SW_XOFF_INT。(R/W)

LP_UART_GLITCH_DET_INT_ENA 写 1 使能 LP_UART_GLITCH_DET_INT。(R/W)

LP_UART_TX_BRK_DONE_INT_ENA 写 1 使能 LP_UART_TX_BRK_DONE_INT。(R/W)

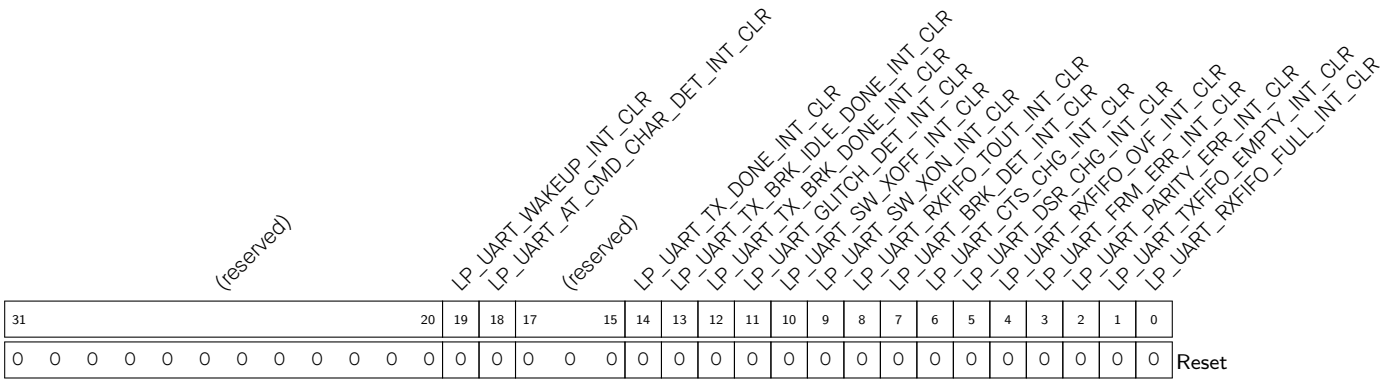
LP_UART_TX_BRK_IDLE_DONE_INT_ENA 写 1 使能 LP_UART_TX_BRK_IDLE_DONE_INT。(R/W)

LP_UART_TX_DONE_INT_ENA 写 1 使能 LP_UART_TX_DONE_INT。(R/W)

LP_UART_AT_CMD_CHAR_DET_INT_ENA 写 1 使能 LP_UART_AT_CMD_CHAR_DET_INT。(R/W)

LP_UART_WAKEUP_INT_ENA 写 1 使能 LP_UART_WAKEUP_INT。(R/W)

Register 45.43. LP_UART_INT_CLR_REG (0x0010)



LP_UART_RXFIFO_FULL_INT_CLR 写 1 清除 LP_UART_RXFIFO_FULL_INT。(WT)

LP_UART_TXFIFO_EMPTY_INT_CLR 写 1 清除 LP_UART_TXFIFO_EMPTY_INT。(WT)

LP_UART_PARITY_ERR_INT_CLR 写 1 清除 LP_UART_PARITY_ERR_INT。(WT)

LP_UART_FRM_ERR_INT_CLR 写 1 清除 LP_UART_FRM_ERR_INT。(WT)

LP_UART_RXFIFO_OVF_INT_CLR 写 1 清除 LP_UART_RXFIFO_OVF_INT。(WT)

LP_UART_DSR_CHG_INT_CLR 写 1 清除 LP_UART_DSR_CHG_INT。(WT)

LP_UART_CTS_CHG_INT_CLR 写 1 清除 LP_UART_CTS_CHG_INT。(WT)

LP_UART_BRK_DET_INT_CLR 写 1 清除 LP_UART_BRK_DET_INT。(WT)

LP_UART_RXFIFO_TOUT_INT_CLR 写 1 清除 LP_UART_RXFIFO_TOUT_INT。(WT)

LP_UART_SW_XON_INT_CLR 写 1 清除 LP_UART_SW_XON_INT。(WT)

LP_UART_SW_XOFF_INT_CLR 写 1 清除 LP_UART_SW_XOFF_INT。(WT)

LP_UART_GLITCH_DET_INT_CLR 写 1 清除 LP_UART_GLITCH_DET_INT。(WT)

LP_UART_TX_BRK_DONE_INT_CLR 写 1 清除 LP_UART_TX_BRK_DONE_INT。(WT)

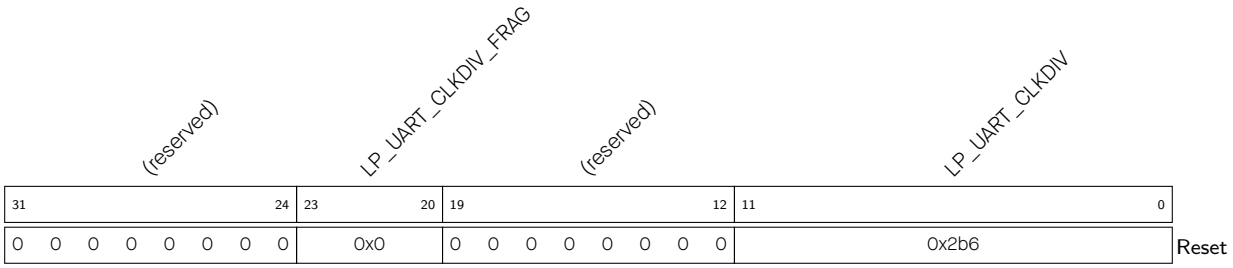
LP_UART_TX_BRK_IDLE_DONE_INT_CLR 写 1 清除 LP_UART_TX_BRK_IDLE_DONE_INT。(WT)

LP_UART_TX_DONE_INT_CLR 写 1 清除 LP_UART_TX_DONE_INT。(WT)

LP_UART_AT_CMD_CHAR_DET_INT_CLR 写 1 清除 LP_UART_AT_CMD_CHAR_DET_INT。(WT)

LP_UART_WAKEUP_INT_CLR 写 1 清除 LP_UART_WAKEUP_INT。(WT)

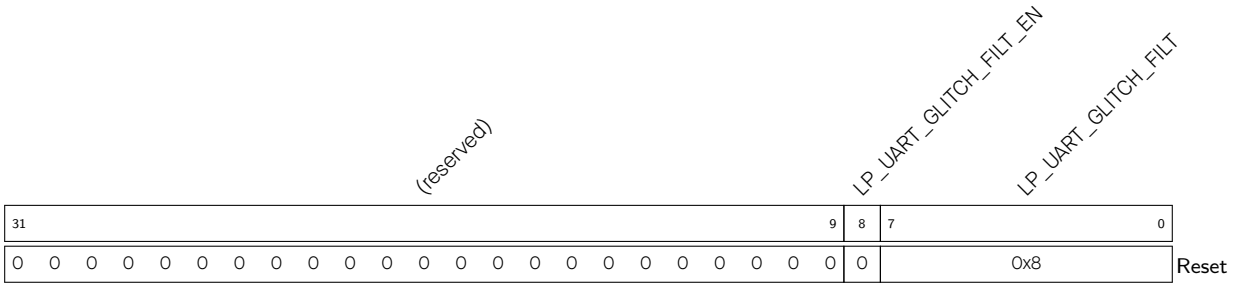
Register 45.44. LP_UART_CLKDIV_SYNC_REG (0x0014)



LP_UART_CLKDIV 配置波特率分频系数的整数部分。(R/W)

LP_UART_CLKDIV_FRAG 配置波特率分频系数的小数部分。(R/W)

Register 45.45. LP_UART_RX_FILT_REG (0x0018)



LP_UART_GLITCH_FILT 配置滤波长度。

单位：UART Core 时钟周期。

宽度小于该数值的输入脉冲会被忽略。(R/W)

LP_UART_GLITCH_FILT_EN 配置是否开启接收信号滤波器。

0：关闭

1：开启

(R/W)

Register 45.46. LP_UART_CONF0_SYNC_REG (0x0020)

Bits	Field	Reset
31:24	(reserved)	0 0 0 0 0 0 0 0
23	LP_UART_TXFIFO_RST	0
22	LP_UART_RXFIFO_RST	0
21	LP_UART_SW_RTS	0
20	LP_UART_MEM_CLK_EN	0
19	(reserved)	0
18	LP_UART_ERR_WR_MASK	0
17	LP_UART_DIS_RX_DAT_OVF	0
16	LP_UART_TXD_INV	0
15	LP_UART_RXD_INV	0
14	(reserved)	0
13	LP_UART_TX_FLOW_EN	0
12	LP_UART_LOOPBACK	0
11:7	(reserved)	0
6	LP_UART_TXD_BRK	0
5:4	LP_UART_STOP_BIT_NUM	1
3:2	LP_UART_BIT_NUM	3
1	LP_UART_PARITY_EN	0
0	LP_UART_PARITY	0

LP_UART_PARITY 配置奇偶检验方式。

0：偶校验

1：奇校验

(R/W)

LP_UART_PARITY_EN 配置是否开启 LP UART 奇偶校验。

0：关闭

1：开启

(R/W)

LP_UART_BIT_NUM 配置数据位长度。

0：5 位

1：6 位

2：7 位

3：8 位

(R/W)

LP_UART_STOP_BIT_NUM 配置停止位长度。

0: 无效值，没有作用

1: 1 位

2: 1.5 位

3: 2 位

(R/W)

LP_UART_TXD_BRK 配置完成数据发送后，发送器是否发送 NULL。

0：不发送

1：发送

(R/W)

LP_UART_LOOPBACK 配置是否开启 LP UART 回环测试模式。

0：关闭

1：开启

(R/W)

LP_UART_TX_FLOW_EN 配置是否开启发送器的流控功能。

0：关闭

1：开启

(R/W)

见下页...

Register 45.46. LP_UART_CONF0_SYNC_REG (0x0020)

接上页...

LP_UART_RXD_INV 配置是否翻转 LP UART RXD 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_TXD_INV 配置是否翻转 LP UART TXD 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_DIS_RX_DAT_OVF 配置是否关闭 LP UART 接收器数据溢出检测。

0：开启

1：关闭

(R/W)

LP_UART_ERR_WR_MASK 配置接收数据有错误时，是否扔将其存储至 FIFO。

0：存储

1：不存储

(R/W)

LP_UART_MEM_CLK_EN 配置是否开启 LP UART 存储器门控。

0：关闭

1：开启

(R/W)

LP_UART_SW_RTS 配置软件流控使用的 RTS 信号。

0：禁止发送端发送数据

1：允许发送端发送数据

(R/W)

LP_UART_RXFIFO_RST 配置是否复位 LP UART RX FIFO。

0：不复位

1：复位

(R/W)

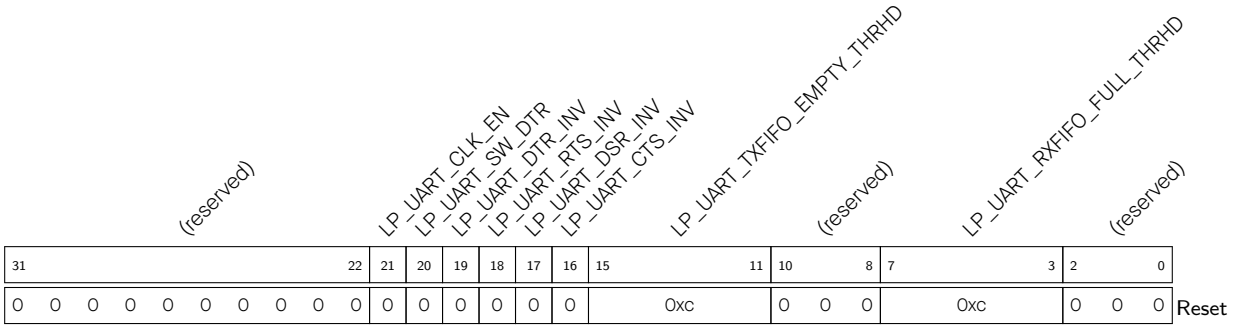
LP_UART_TXFIFO_RST 配置是否复位 LP UART TX FIFO。

0：不复位

1：复位

(R/W)

Register 45.47. LP_UART_CONF1_REG (0x0024)



LP_UART_RXFIFO_FULL_THRHD 配置 RX FIFO 为满的阈值。

单位：字节。(R/W)

LP_UART_TXFIFO_EMPTY_THRHD 配置 TX FIFO 为空的阈值。

单位：字节。(R/W)

LP_UART_CTS_INV 配置是否翻转 LP UART CTS 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_DSR_INV 配置是否翻转 LP UART DSR 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_RTS_INV 配置是否翻转 LP UART RTS 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_DTR_INV 配置是否翻转 LP UART DTR 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_SW_DTR 配置软件流控使用的 DTR 信号。

0: 发送数据未准备完毕，处于不可用状态

1: 发送数据准备完毕，处于可用状态

(R/W)

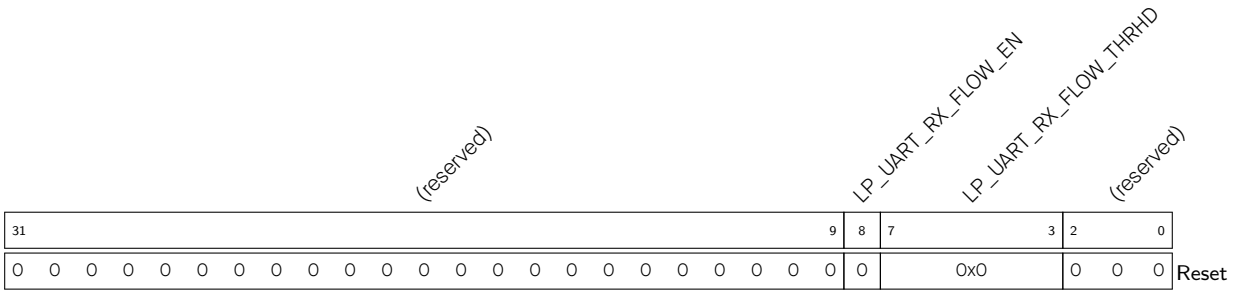
LP_UART_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 45.48. LP_UART_HWFC_CONF_SYNC_REG (0x002C)



LP_UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大数。

单位：字节。(R/W)

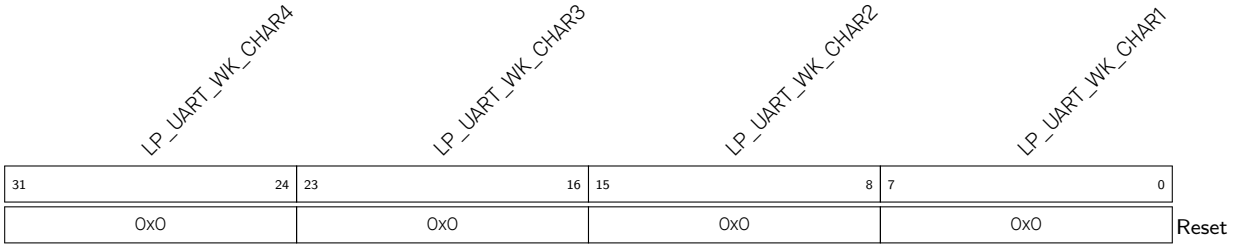
LP_UART_RX_FLOW_EN 配置是否开启 LP UART 接收器。

0：关闭

1：开启

(R/W)

Register 45.49. LP_UART_SLEEP_CONF0_REG (0x0030)



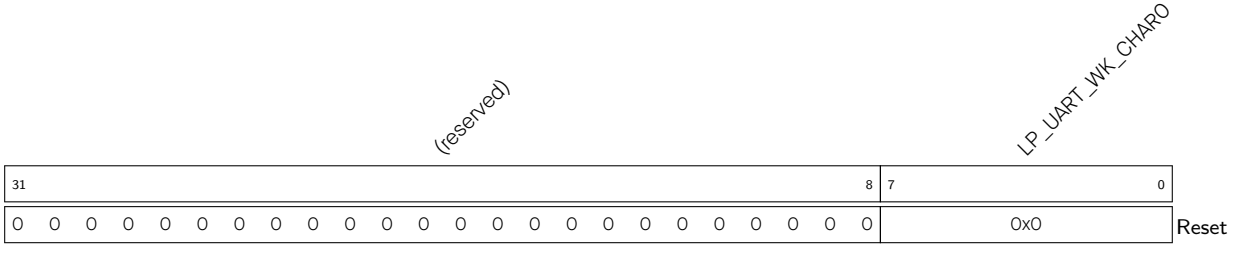
LP_UART_WK_CHAR1 配置唤醒芯片的特定字符 1。(R/W)

LP_UART_WK_CHAR2 配置唤醒芯片的特定字符 2。(R/W)

LP_UART_WK_CHAR3 配置唤醒芯片的特定字符 3。(R/W)

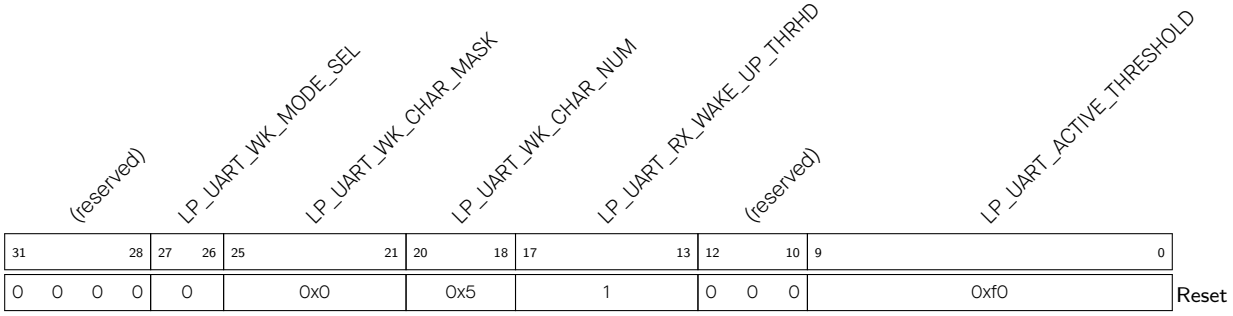
LP_UART_WK_CHAR4 配置唤醒芯片的特定字符 4。(R/W)

Register 45.50. LP_UART_SLEEP_CONF1_REG (0x0034)



LP_UART_WK_CHAR0 配置唤醒芯片的特定字符 0。(R/W)

Register 45.51. LP_UART_SLEEP_CONF2_REG (0x0038)



LP_UART_ACTIVE_THRESHOLD 配置唤醒模式 0 唤醒芯片所需的 RXD 沿变化次数。(R/W)

LP_UART_RX_WAKE_UP_THRHD 配置唤醒模式 1 唤醒芯片所需接收的数据量。

单位：字节。(R/W)

LP_UART_WK_CHAR_NUM 配置选取多少个唤醒字符。(R/W)

LP_UART_WK_CHAR_MASK 配置是否屏蔽唤醒字符。

0：不屏蔽

1：屏蔽

(R/W)

LP_UART_WK_MODE_SEL 配置选取哪个唤醒模式。

0：唤醒模式 0

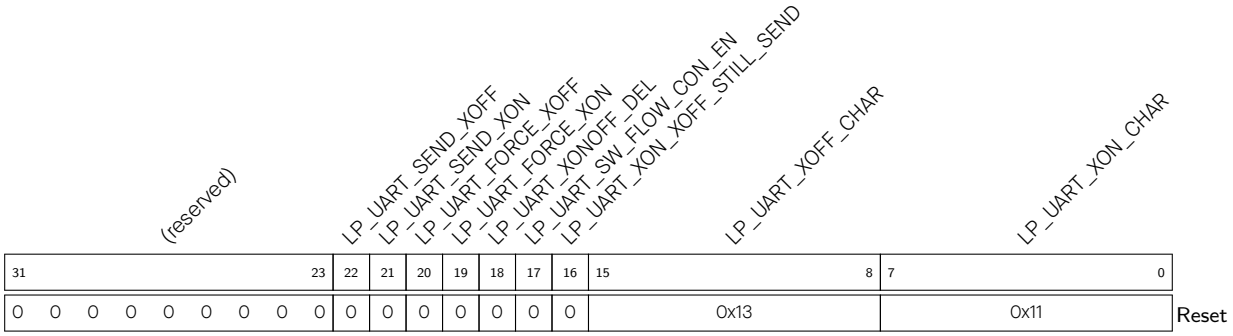
1：唤醒模式 1

2：唤醒模式 2

3：唤醒模式 3

(R/W)

Register 45.52. LP_UART_SWFC_CONF0_SYNC_REG (0x003C)



LP_UART_XON_CHAR 配置 XON 流控字符。(R/W)

LP_UART_XOFF_CHAR 配置 XOFF 流控字符。(R/W)

LP_UART_XON_XOFF_STILL_SEND 配置 LP UART 发送器关闭时，是否仍能发送 XON 或 XOFF 字符。

0：不能发送

1：可以发送

(R/W)

LP_UART_SW_FLOW_CON_EN 配置是否开启软件流控。

0：关闭

1：开启

(R/W)

LP_UART_XONOFF_DEL 配置是否移除接收数据中的流控字符。

0：不移除

1：移除

(R/W)

LP_UART_FORCE_XON 配置是否让发送器继续发送数据。

0：不发送

1：继续发送

(R/W)

LP_UART_FORCE_XOFF 配置是否让发送器停止发送数据。

0：不停止

1：停止

(R/W)

LP_UART_SEND_XON 配置是否发送 XON 字符。

0：不发送

1：发送

(R/W/SS/SC)

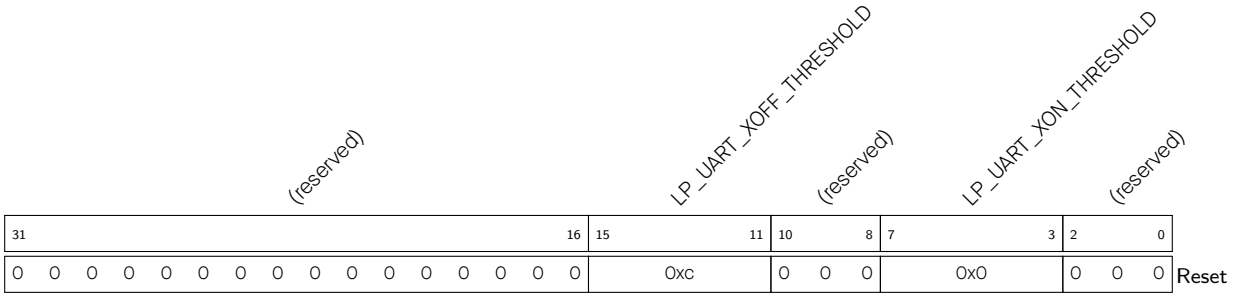
LP_UART_SEND_XOFF 配置是否发送 XOFF 字符。

0：不发送

1：发送

(R/W/SS/SC)

Register 45.53. LP_UART_SWFC_CONF1_REG (0x0040)



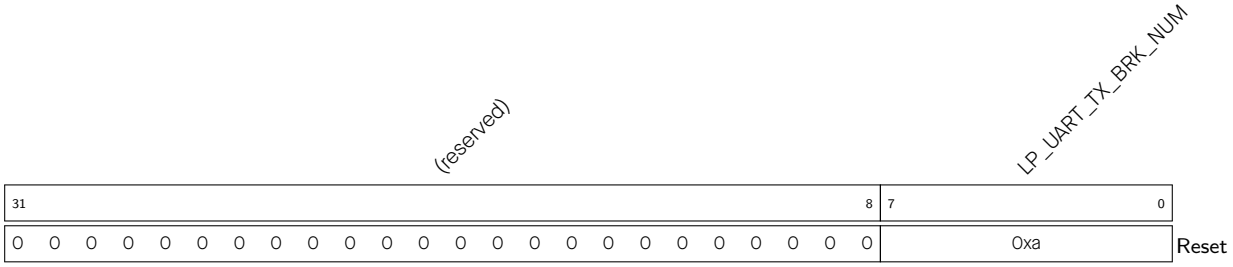
LP_UART_XON_THRESHOLD 配置软件流控时，发送 XON 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

LP_UART_XOFF_THRESHOLD 配置软件流控时，发送 XOFF 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

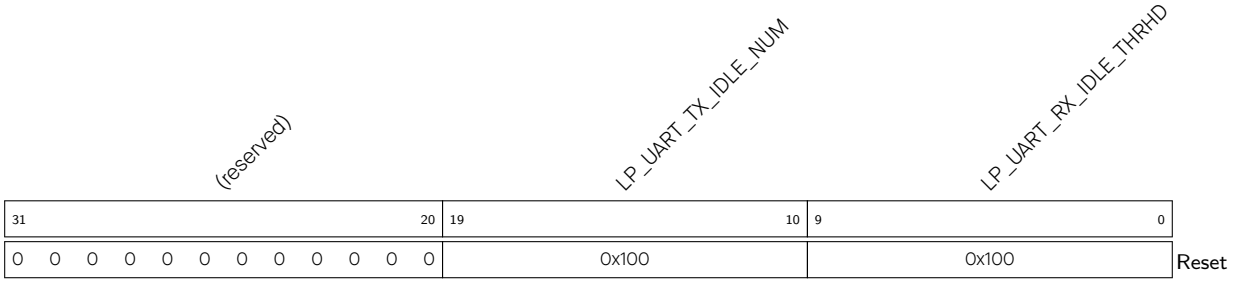
Register 45.54. LP_UART_TXBRK_CONF_SYNC_REG (0x0044)



LP_UART_TX_BRK_NUM 配置完成数据发送后待发 NULL 字符的数量。

仅在 LP_UART_TXD_BRK 为 1 时有效。(R/W)

Register 45.55. LP_UART_IDLE_CONF_SYNC_REG (0x0048)



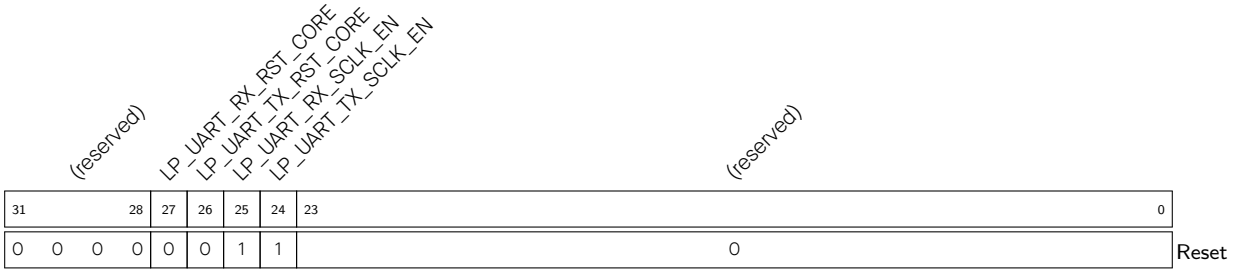
LP_UART_RX_IDLE_THRHD 配置接收一字节数据时间过长、产生帧结束信号的阈值。

单位：位时间（传输一位所需的时间）。(R/W)

LP_UART_TX_IDLE_NUM 配置两次数据传输的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 45.56. LP_UART_CLK_CONF_REG (0x0088)



LP_UART_TX_SCLK_EN 配置是否开启 LP UART TX 时钟。

0：关闭

1：开启

(R/W)

LP_UART_RX_SCLK_EN 配置是否开启 LP UART RX 时钟。

0：关闭

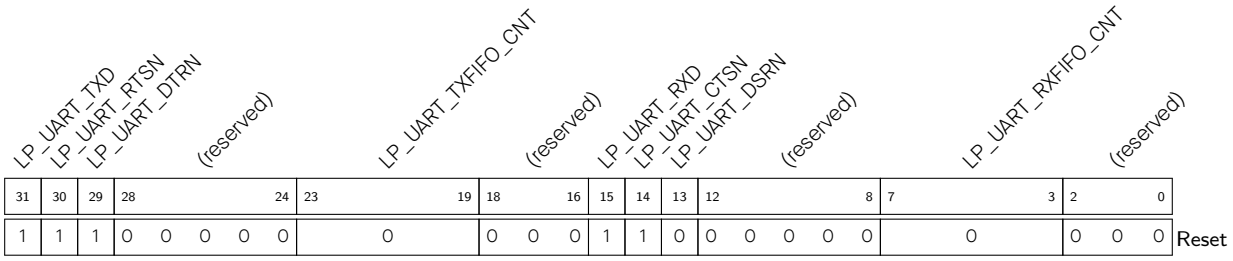
1：开启

(R/W)

LP_UART_TX_RST_CORE 向此位先写 1 后写 0，复位 LP UART TX。(R/W)

LP_UART_RX_RST_CORE 向此位先写 1 后写 0，复位 LP UART RX。(R/W)

Register 45.57. LP_UART_STATUS_REG (0x001C)



LP_UART_RXFIFO_CNT 表示 RX FIFO 中有效数据量。

单位：字节。(RO)

LP_UART_DSRN 表示内部 LP UART DSR 信号的电平值。(RO)

LP_UART_CTSN 表示内部 LP UART CTS 信号的电平值。(RO)

LP_UART_RXD 表示内部 LP UART RXD 信号的电平值。(RO)

LP_UART_TXFIFO_CNT 表示 TX FIFO 中的数据量。

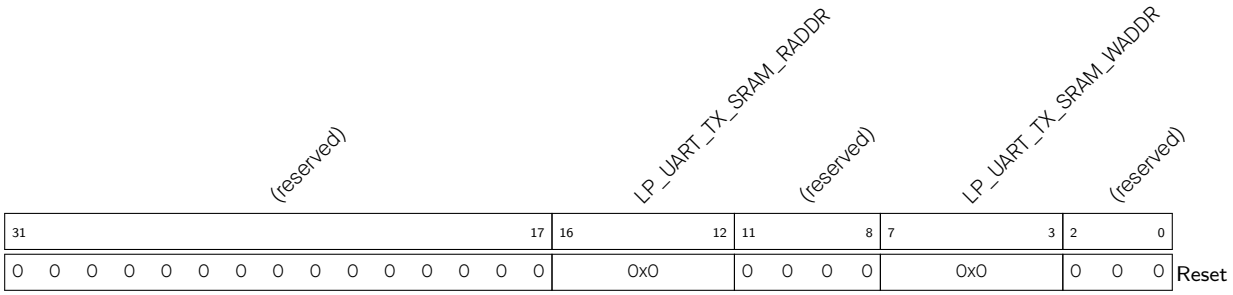
单位：字节。(RO)

LP_UART_DTRN 表示内部 LP UART DTR 信号的电平。(RO)

LP_UART_RTSN 表示内部 LP UART RTS 信号的电平。(RO)

LP_UART_TXD 表示内部 LP UART TXD 信号的电平。(RO)

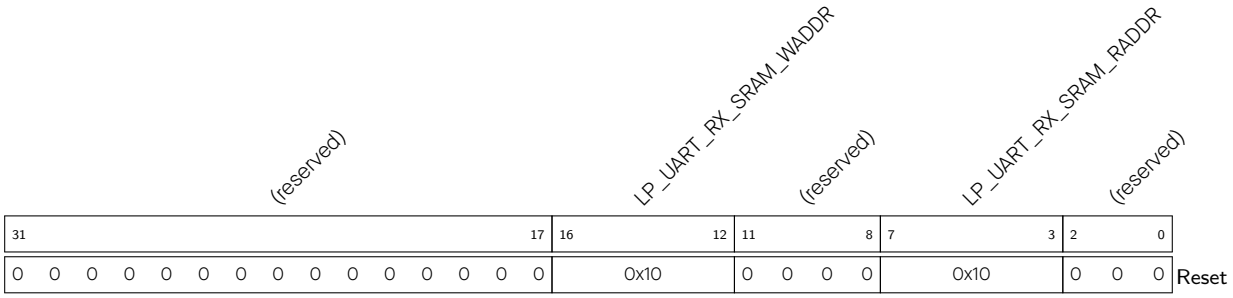
Register 45.58. LP_UART_MEM_TX_STATUS_REG (0x0068)



LP_UART_TX_SRAM_WADDR 表示写 TX FIFO 的偏移地址。(RO)

LP_UART_TX_SRAM_RADDR 表示读 TX FIFO 的偏移地址。(RO)

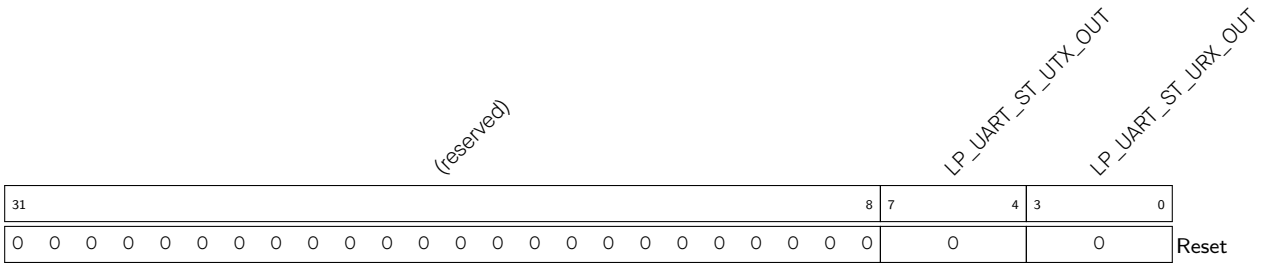
Register 45.59. LP_UART_MEM_RX_STATUS_REG (0x006C)



LP_UART_RX_SRAM_RADDR 表示读 RX FIFO 的偏移地址。(RO)

LP_UART_RX_SRAM_WADDR 表示写 RX FIFO 的偏移地址。(RO)

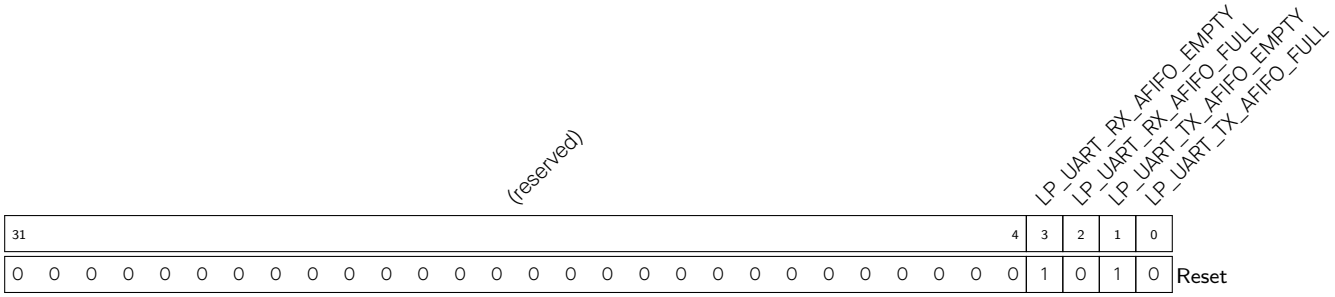
Register 45.60. LP_UART_FSM_STATUS_REG (0x0070)



LP_UART_ST_URX_OUT 表示 LP UART 接收器的状态。(RO)

LP_UART_ST_UTX_OUT 表示 LP UART 发送器的状态。(RO)

Register 45.61. LP_UART_AFIFO_STATUS_REG (0x0090)



LP_UART_TX_AFIFO_FULL 表示 APB TX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

LP_UART_TX_AFIFO_EMPTY 表示 APB TX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

LP_UART_RX_AFIFO_FULL 表示 APB RX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

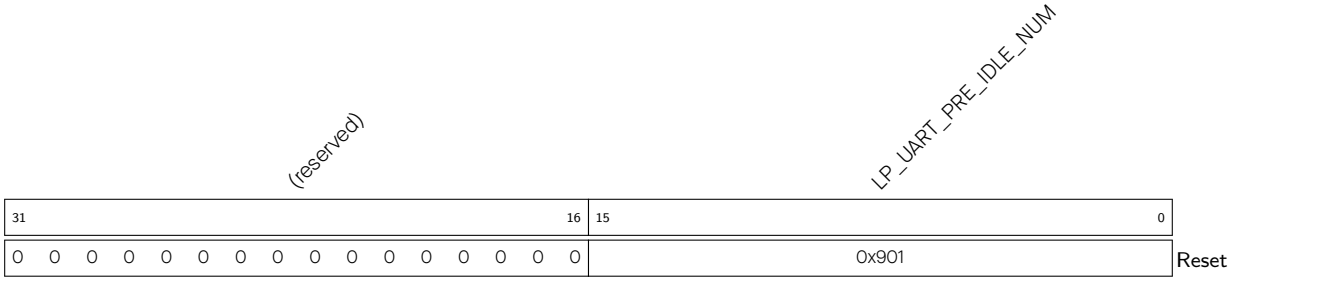
LP_UART_RX_AFIFO_EMPTY 表示 APB RX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

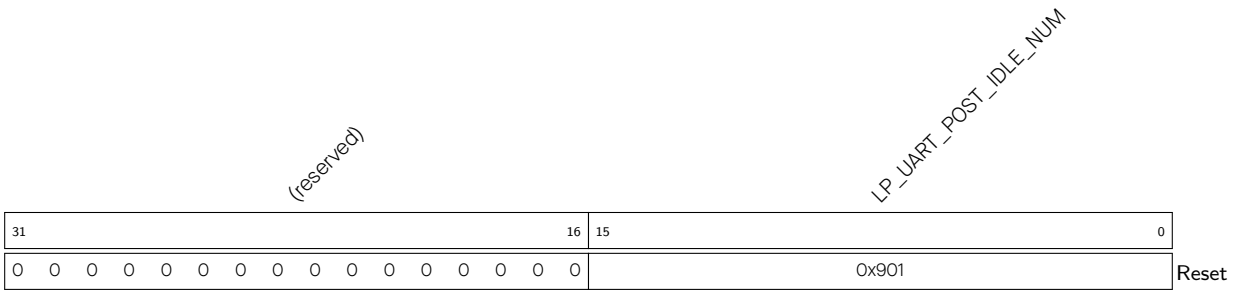
Register 45.62. LP_UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



LP_UART_PRE_IDLE_NUM 配置接收器接收到第一个 AT_CMD 之前的空闲时间。

单位：位时间（传输一位所需的时间）。(R/W)

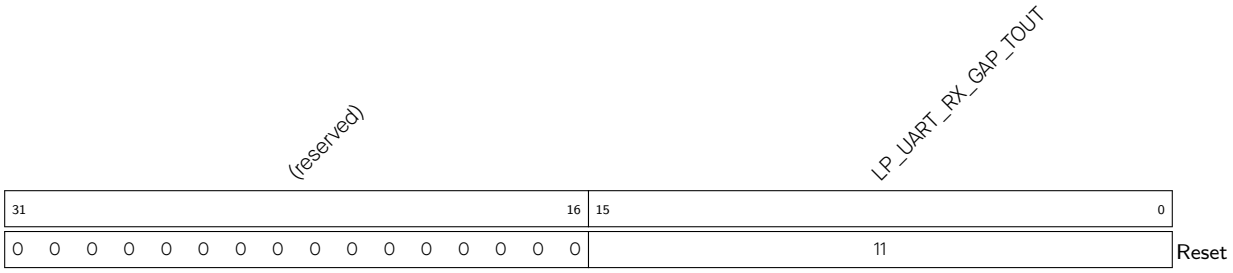
Register 45.63. LP_UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



LP_UART_POST_IDLE_NUM 配置最后一个 AT_CMD 和后续数据之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

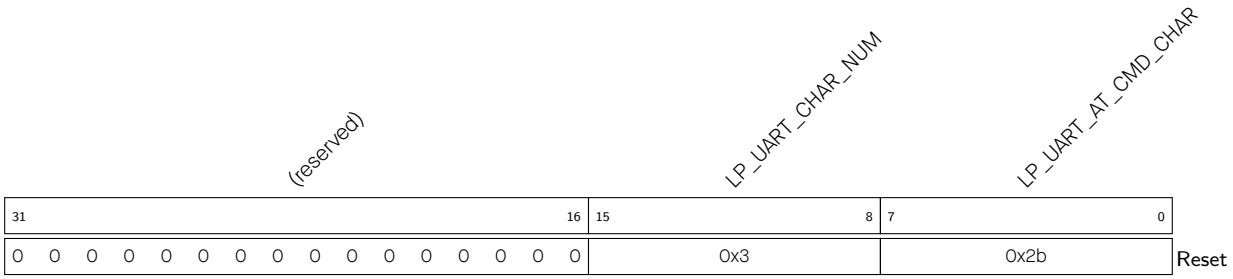
Register 45.64. LP_UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



LP_UART_RX_GAP_TOUT 配置 AT_CMD 之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

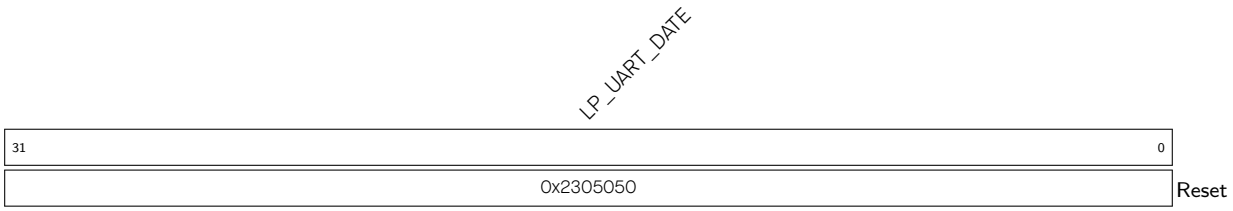
Register 45.65. LP_UART_AT_CMD_CHAR_SYNC_REG (0x005C)



LP_UART_AT_CMD_CHAR 配置 AT_CMD 字符。(R/W)

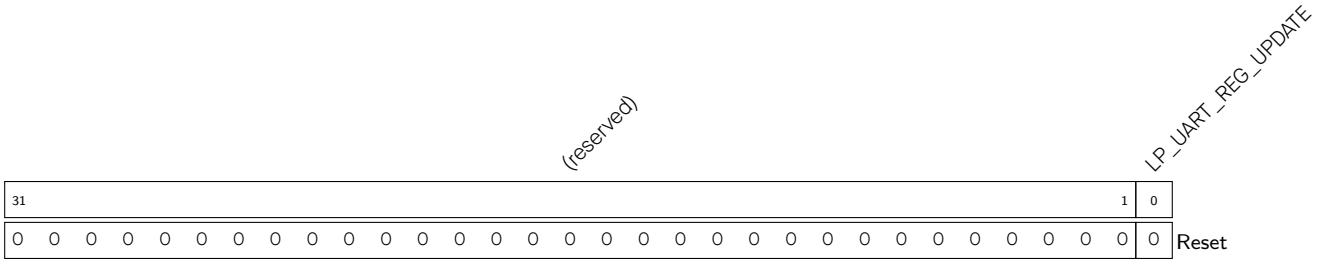
LP_UART_CHAR_NUM 配置接收器可连续接收的 AT_CMD 字符数量。(R/W)

Register 45.66. LP_UART_DATE_REG (0x008C)



LP_UART_DATE 版本控制寄存器。(R/W)

Register 45.67. LP_UART_REG_UPDATE_REG (0x0098)



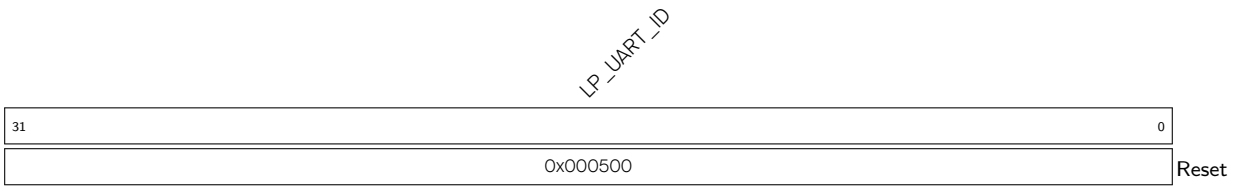
LP_UART_REG_UPDATE 配置是否同步寄存器

0：不同步

1：同步

(R/W/SC)

Register 45.68. LP_UART_ID_REG (0x009C)



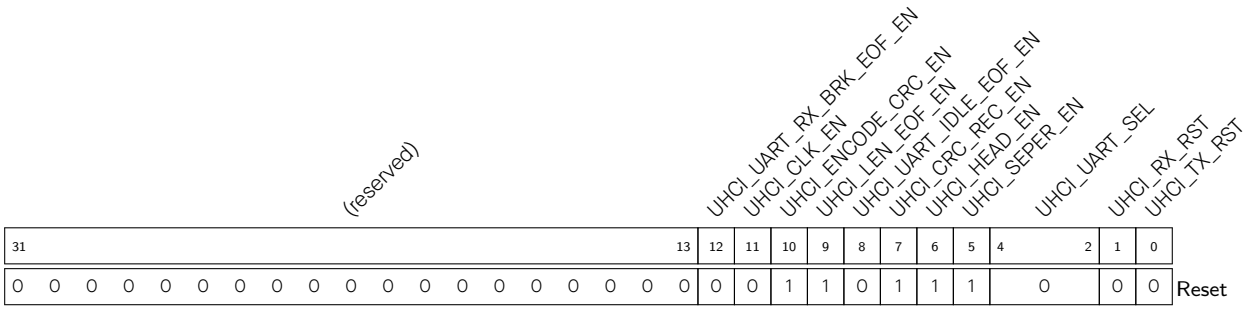
LP_UART_ID 配置 LP UART ID。(R/W)

45.8.3 UHCI 寄存器

本小节的所有地址均为相对于 UHCI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 45.69. UHCI_CONF0_REG (0x0000)



UHCI_TX_RST 先写 1 再写 0 复位解码器状态机。(R/W)

UHCI_RX_RST 先写 1 再写 0 复位编码器状态机。(R/W)

UHCI_UART_SEL 从 UART 0 ~ 4 中选择其一连接到 UHCI。

0：选择 UART0

1：选择 UART1

2：选择 UART2

3：选择 UART3

4：选择 UART4

5 ~ 7：无效值，没有 UART 连接到 UHCI

(R/W)

UHCI_SEPER_EN 配置是否使用特殊字符分隔数据帧。

0：不分隔

1：分隔

(R/W)

UHCI_HEAD_EN 配置是否用格式报头编码数据包。

0：不使用报头

1：使用报头

(R/W)

UHCI_CRC_REC_EN 配置 UHCI 是否开启接收 16 位 CRC。

0：关闭

1：开启

(R/W)

见下页...

Register 45.69. UHCI_CONF0_REG (0x0000)

接上页...

UHCI_UART_IDLE_EOF_EN 配置 UART 空闲时 UHCI 是否停止接收数据。

0：不停止

1：停止

(R/W)

UHCI_LEN_EOF_EN 配置 UHCI 解码器停止接收数据的条件。

0：接收到 0xC0 后停止

1：接收字节数达到指定值时停止。UHCI_HEAD_EN 为 1 时，指定值是 UCHI 数据包报头指定的数据长度。UHCI_HEAD_EN 为 0 时，指定值为配置值。

(R/W)

UHCI_ENCODE_CRC_EN 配置是否开启数据完整性检测，在数据末尾加 16 位 CCITT-CRC。

0：关闭

1：开启

(R/W)

UHCI_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

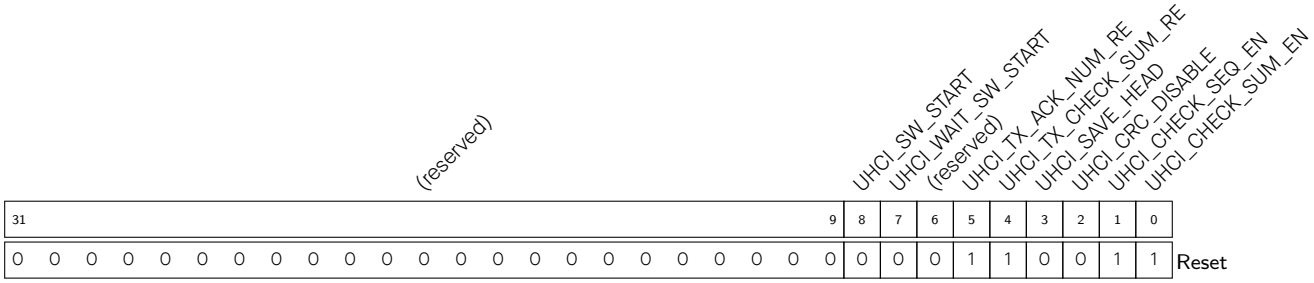
UHCI_UART_RX_BRK_EOF_EN 配置 UART 收到 NULL 帧后，UHCI 是否停止接收数据。

0：不停止

1：停止

(R/W)

Register 45.70. UHCI_CONF1_REG (0x0014)



UHCI_CHECK_SUM_EN 配置 UHCI 接收数据包时，是否开启报头校验和检查。

0：关闭

1：开启

(R/W)

UHCI_CHECK_SEQ_EN 配置 UHCI 接收数据包时，是否开启序列号检查。

0：关闭

1：开启

(R/W)

UHCI_CRC_DISABLE 配置是否开启 CRC 计算。

0：关闭

1：开启

仅在 UHCI 包中的数据完整性检测位为 1 时有效。

(R/W)

UHCI_SAVE_HEAD 配置 UHCI 接收数据包时，是否保存数据包报头。

0：不保存

1：保存

(R/W)

UHCI_TX_CHECK_SUM_RE 配置用校验和编码数据包。

0：不使用

1：使用

(R/W)

UHCI_TX_ACK_NUM_RE 配置准备发送可靠数据包时，是否用 ACK 编码该数据包。

0：不使用

1：使用

(R/W)

UHCI_WAIT_SW_START 配置是否将 UHCI 编码器的状态机调至 ST_SW_WAIT 状态。

0：否

1：是

(R/W)

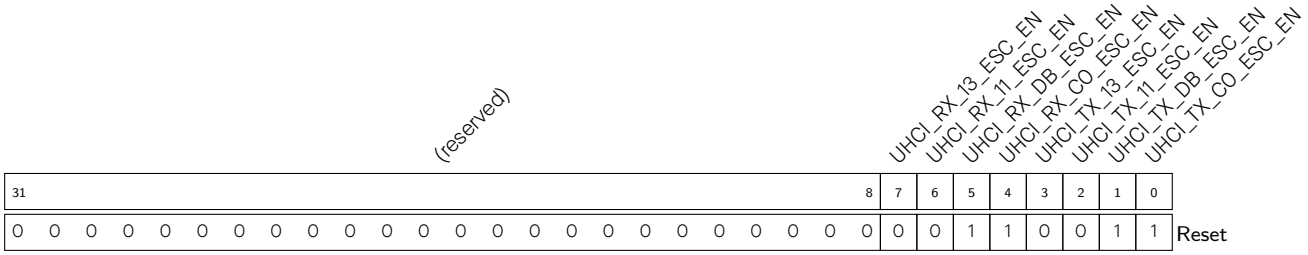
UHCI_SW_START 编码器的状态机为 ST_SW_WAIT 状态时，配置 UHCI 是否开始发送数据包。

0：不发送

1：开始发送

(R/W/SC)

Register 45.71. UHCI_ESCAPE_CONF_REG (0x0020)



UHCI_TX_C0_ESC_EN 配置 DMA 接收数据时，是否开启 0xC0 字符解码。

0：关闭

1：开启

(R/W)

UHCI_TX_DB_ESC_EN 配置 DMA 接收数据时，是否开启 0xDB 字符解码。

0：关闭

1：开启

(R/W)

UHCI_TX_11_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x11 解码。

0：关闭

1：开启

(R/W)

UHCI_TX_13_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x13 解码。

0：关闭

1：开启

(R/W)

UHCI_RX_C0_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换 0xC0。

0：不替换

1：替换

(R/W)

UHCI_RX_DB_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换 0xDB。

0：不替换

1：替换

(R/W)

UHCI_RX_11_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x11。

0：不替换

1：替换

(R/W)

UHCI_RX_13_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x13。

0：不替换

1：替换

(R/W)

Register 45.72. UHCI_HUNG_CONF_REG (0x0024)

(reserved)	UHCI_RXFIFO_TIMEOUT_ENA	UHCI_RXFIFO_TIMEOUT_SHIFT	UHCI_RXFIFO_TIMEOUT	UHCI_TXFIFO_TIMEOUT_ENA	UHCI_TXFIFO_TIMEOUT_SHIFT	UHCI_TXFIFO_TIMEOUT	
31 24	23	22 20	19 12	11	10 8	7 0	
0 0 0 0 0 0 0 0	1	0	0x10	1	0	0x10	Reset

UHCI_TXFIFO_TIMEOUT 配置 DMA 接收数据的超时值。

单位：毫秒。(R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT 配置 TX FIFO 超时计数器的阈值上限。(R/W)

UHCI_TXFIFO_TIMEOUT_ENA 配置是否开启 TX FIFO 接收数据超时。

0：关闭

1：开启

(R/W)

UHCI_RXFIFO_TIMEOUT 配置 DMA 从 RAM 读取数据的超时值。单位：毫秒。(R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT 配置 RX FIFO 超时计数器的阈值上限。(R/W)

UHCI_RXFIFO_TIMEOUT_ENA 配置是否开启 DMA 发送数据超时。

0：关闭

1：开启

(R/W)

Register 45.73. UHCI_ACK_NUM_REG (0x0028)

(reserved)	UHCI_ACK_NUM_LOAD	UHCI_ACK_NUM	
31 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0x0	Reset

UHCI_ACK_NUM 配置软件流控中使用的 ACK 数量。(R/W)

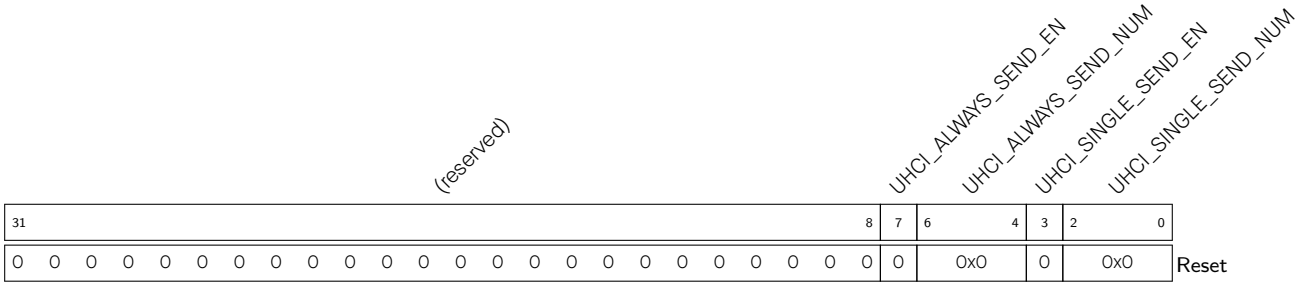
UHCI_ACK_NUM_LOAD 配置是否加载 ACK。

0：不加载

1：加载

(WT)

Register 45.74. UHCI_QUICK_SENT_REG (0x0030)



UHCI_SINGLE_SEND_NUM 配置 single_send 模式下发送哪个寄存器中的数据。

0：Q0 寄存器
1：Q1 寄存器
2：Q2 寄存器
3：Q3 寄存器
4：Q4 寄存器
5：Q5 寄存器
6：Q6 寄存器
7：无效值，没有作用
(R/W)

UHCI_SINGLE_SEND_EN 配置是否开启 single_send 模式。

0：关闭
1：开启
(R/W/SC)

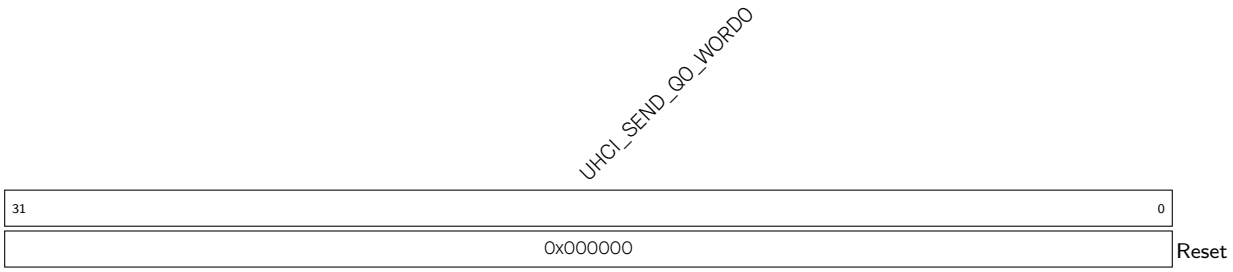
UHCI_ALWAYS_SEND_NUM 配置 always_send 模式下发送哪个寄存器中的数据。

0：Q0 寄存器
1：Q1 寄存器
2：Q2 寄存器
3：Q3 寄存器
4：Q4 寄存器
5：Q5 寄存器
6：Q6 寄存器
7：无效值，没有作用
(R/W)

UHCI_ALWAYS_SEND_EN 配置是否开启 always_send 模式。

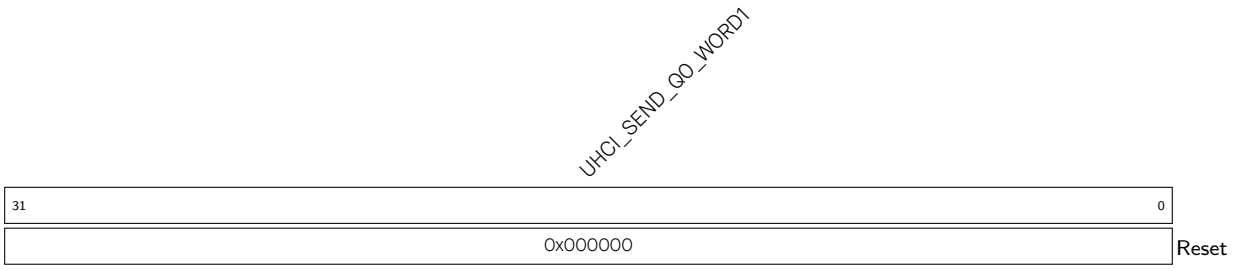
0：关闭
1：开启
(R/W)

Register 45.75. UHCI_REG_Q0_WORD0_REG (0x0034)



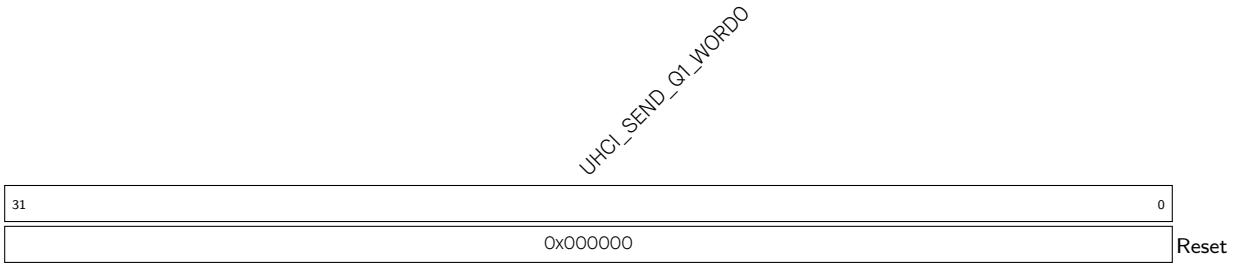
UHCI_SEND_Q0_WORD0 Q0 寄存器待发送数据。(R/W)

Register 45.76. UHCI_REG_Q0_WORD1_REG (0x0038)



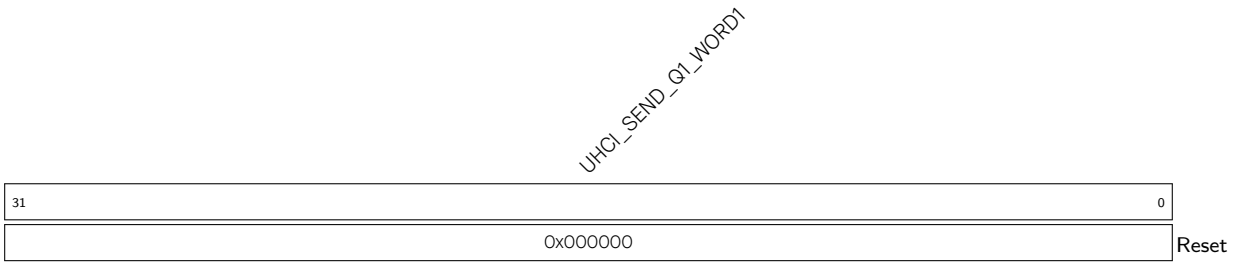
UHCI_SEND_Q0_WORD1 Q0 寄存器待发送数据。(R/W)

Register 45.77. UHCI_REG_Q1_WORD0_REG (0x003C)



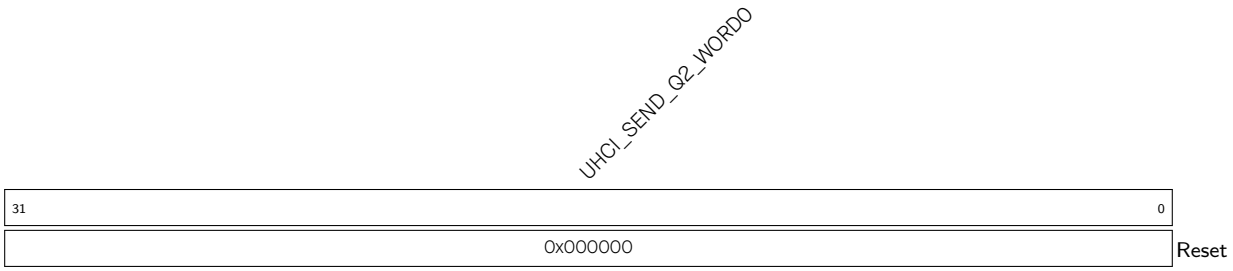
UHCI_SEND_Q1_WORD0 Q1 寄存器待发送数据。(R/W)

Register 45.78. UHCI_REG_Q1_WORD1_REG (0x0040)



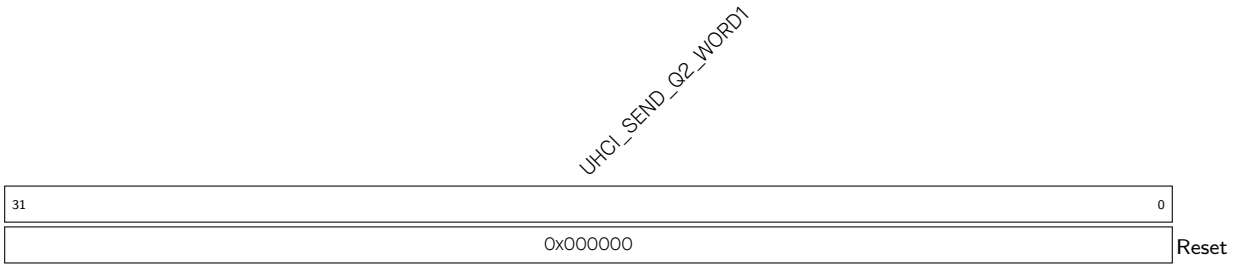
UHCI_SEND_Q1_WORD1 Q1 寄存器待发送数据。(R/W)

Register 45.79. UHCI_REG_Q2_WORD0_REG (0x0044)



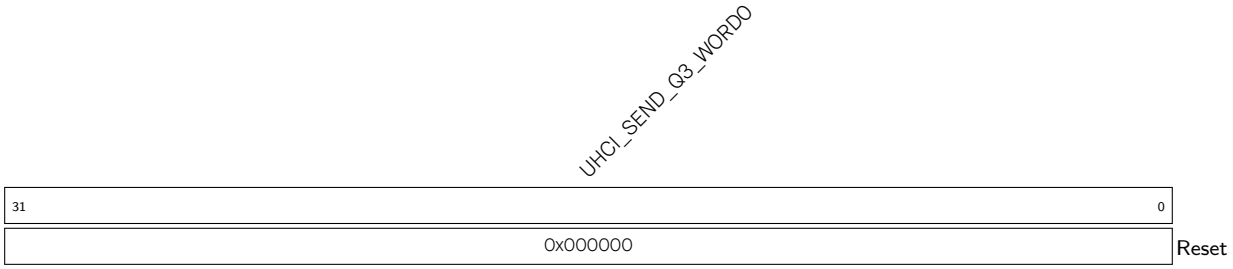
UHCI_SEND_Q2_WORD0 Q2 寄存器待发送数据。(R/W)

Register 45.80. UHCI_REG_Q2_WORD1_REG (0x0048)



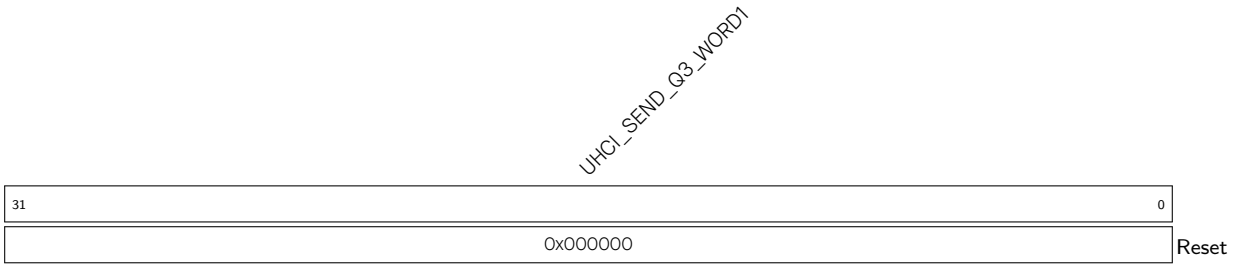
UHCI_SEND_Q2_WORD1 Q2 寄存器待发送数据。(R/W)

Register 45.81. UHCI_REG_Q3_WORD0_REG (0x004C)



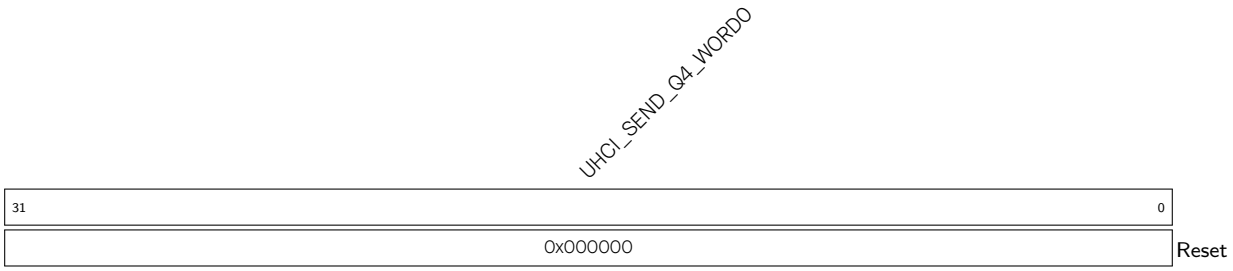
UHCI_SEND_Q3_WORD0 Q3 寄存器待发送数据。(R/W)

Register 45.82. UHCI_REG_Q3_WORD1_REG (0x0050)



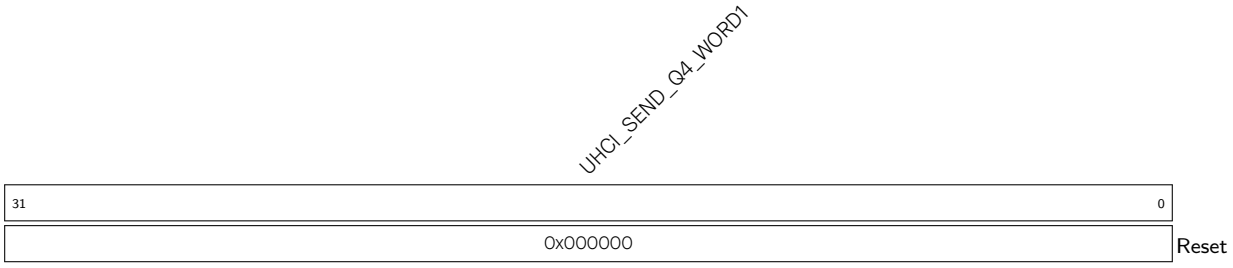
UHCI_SEND_Q3_WORD1 Q3 寄存器待发送数据。(R/W)

Register 45.83. UHCI_REG_Q4_WORD0_REG (0x0054)



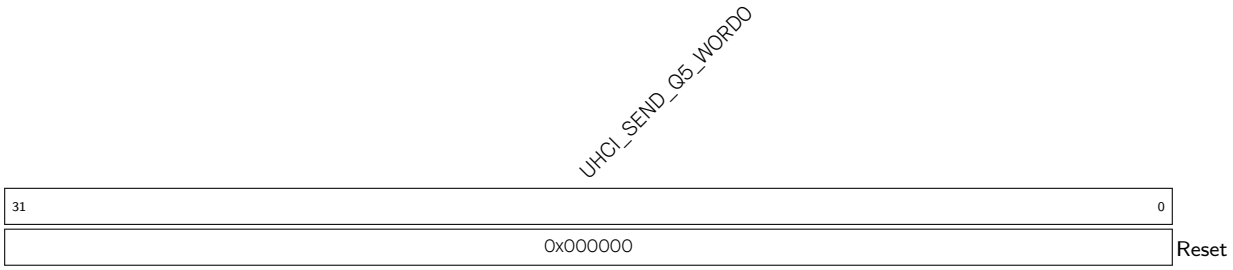
UHCI_SEND_Q4_WORD0 Q4 寄存器待发送数据。(R/W)

Register 45.84. UHCI_REG_Q4_WORD1_REG (0x0058)



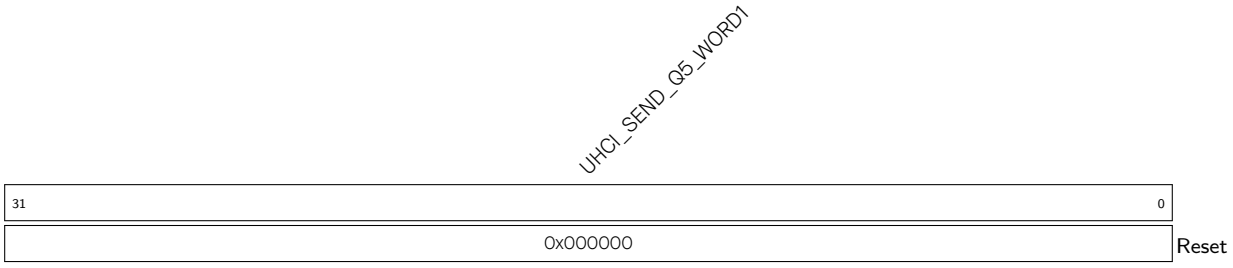
UHCI_SEND_Q4_WORD1 Q4 寄存器待发送数据。(R/W)

Register 45.85. UHCI_REG_Q5_WORD0_REG (0x005C)



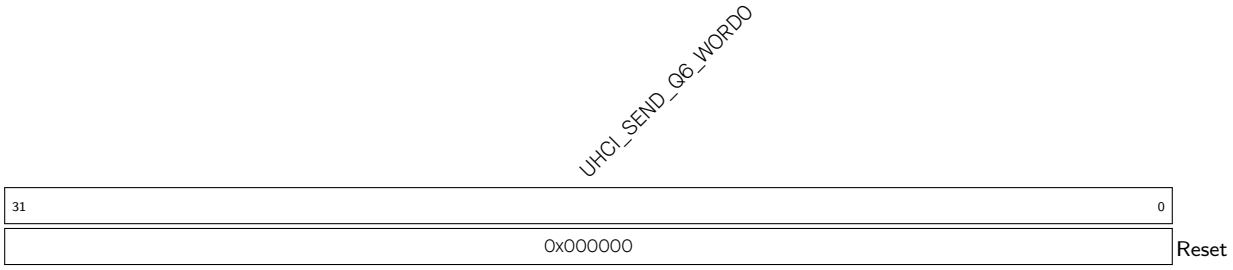
UHCI_SEND_Q5_WORD0 Q5 寄存器待发送数据。(R/W)

Register 45.86. UHCI_REG_Q5_WORD1_REG (0x0060)



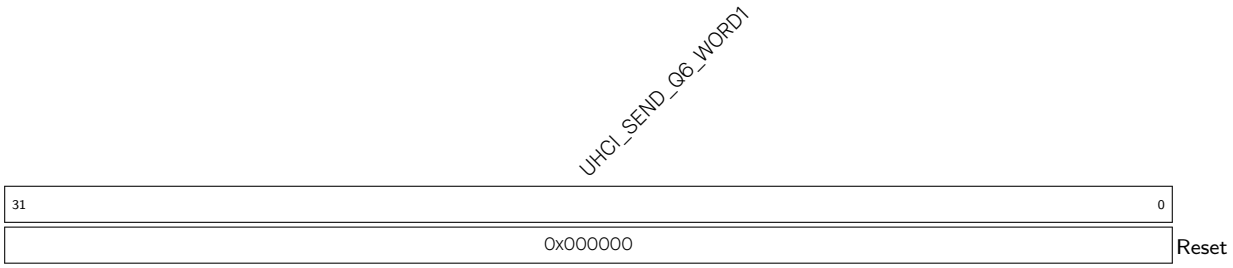
UHCI_SEND_Q5_WORD1 Q5 寄存器待发送数据。(R/W)

Register 45.87. UHCI_REG_Q6_WORD0_REG (0x0064)



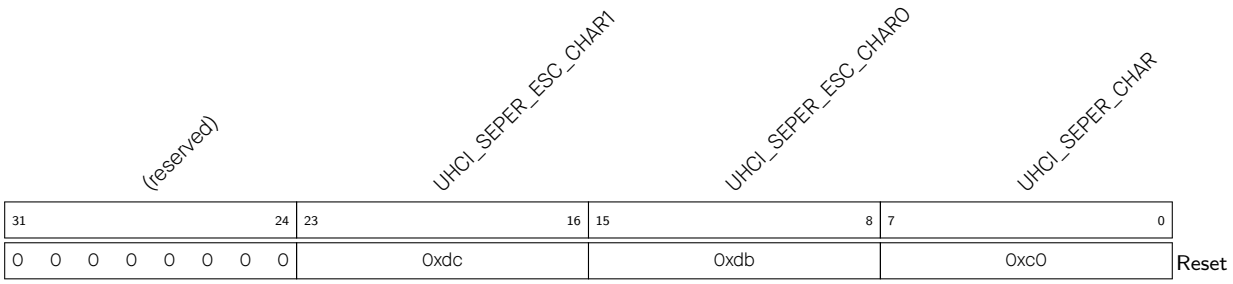
UHCI_SEND_Q6_WORD0 Q6 寄存器待发送数据。(R/W)

Register 45.88. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Q6 寄存器待发送数据。(R/W)

Register 45.89. UHCI_ESC_CONF0_REG (0x006C)

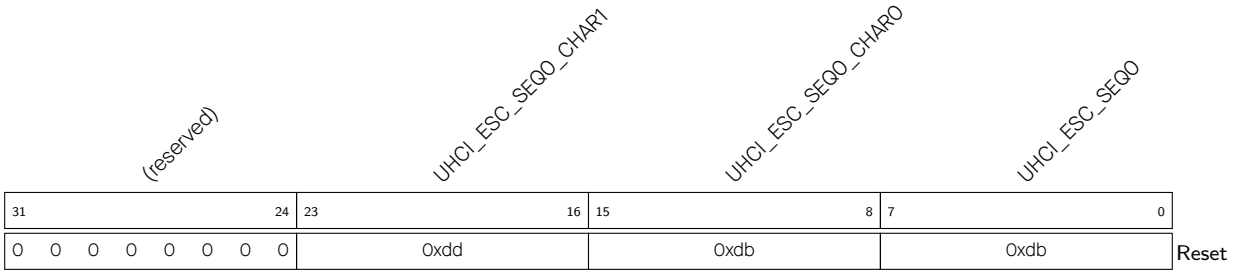


UHCI_SEPER_CHAR 配置编码数据包的分隔符，默认为 0xC0。(R/W)

UHCI_SEPER_ESC_CHAR0 配置 SLIP 转义序列的第一个字符，默认为 0xDB。(R/W)

UHCI_SEPER_ESC_CHAR1 配置 SLIP 转义序列的第二个字符，默认为 0xDC。(R/W)

Register 45.90. UHCI_ESC_CONF1_REG (0x0070)

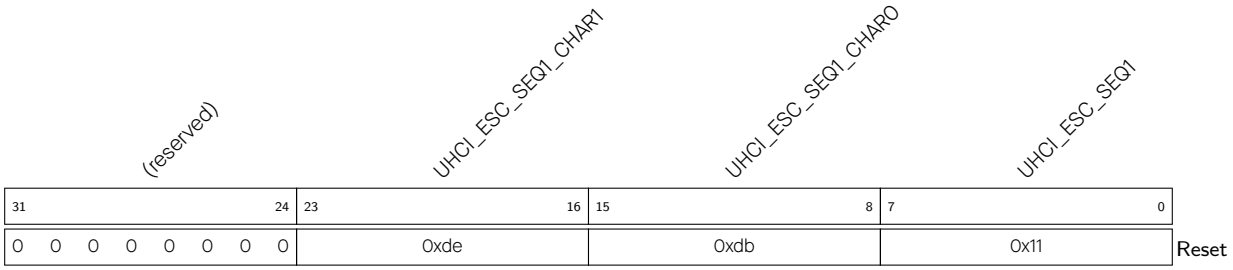


UHCI_ESC_SEQ0 配置需编码的字符，默认为用作 SLIP 转义序列第一个字符的 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ0 默认为 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ0 默认为 0xDD。(R/W)

Register 45.91. UHCI_ESC_CONF2_REG (0x0074)

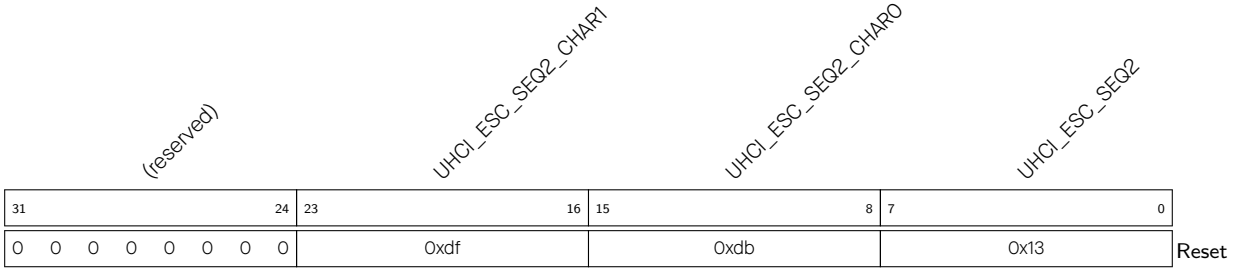


UHCI_ESC_SEQ1 配置需编码的字符，默认为流控字符的 0x11。(R/W)

UHCI_ESC_SEQ1_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ1 默认为 0xDB。(R/W)

UHCI_ESC_SEQ1_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ1 默认为 0xDE。(R/W)

Register 45.92. UHCI_ESC_CONF3_REG (0x0078)

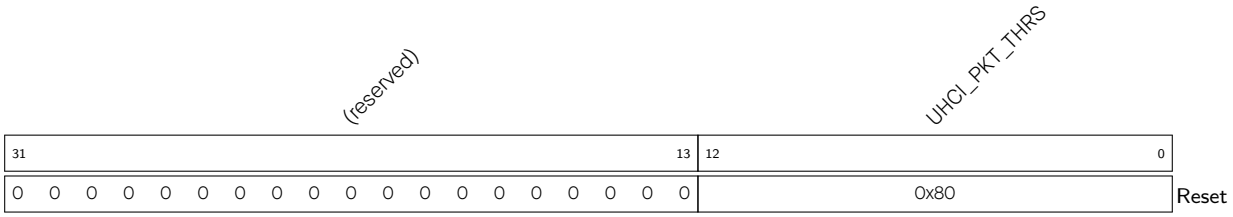


UHCI_ESC_SEQ2 配置需编码的字符，默认为流控字符的 0x13。(R/W)

UHCI_ESC_SEQ2_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ2 默认为 0xDB。(R/W)

UHCI_ESC_SEQ2_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ2 默认为 0xDF。(R/W)

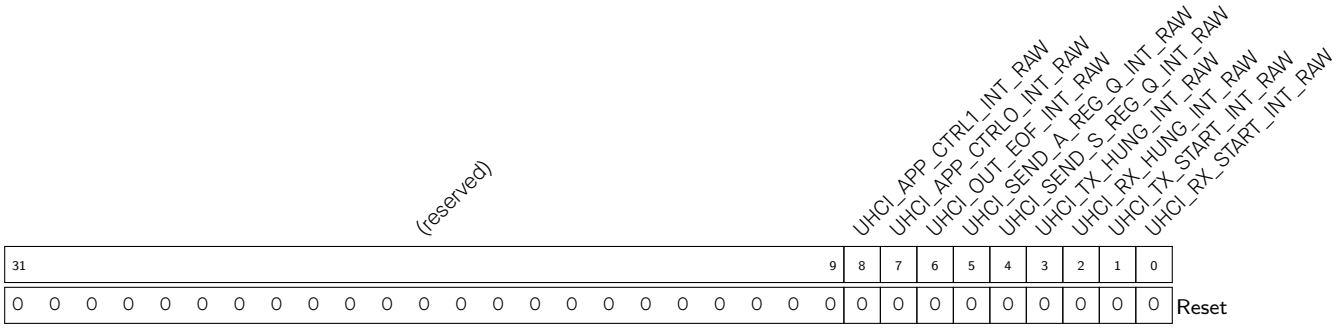
Register 45.93. UHCI_PKT_THRES_REG (0x007C)



UHCI_PKT_THRS 配置数据包的最大长度。

单位：字节。仅在 UHCI_HEAD_EN 为 0 时有效。(R/W)

Register 45.94. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW UHCI_RX_START_INT 的原始中断状态。(R/WTC/SS)

UHCI_TX_START_INT_RAW UHCI_TX_START_INT 的原始中断状态。(R/WTC/SS)

UHCI_RX_HUNG_INT_RAW UHCI_RX_HUNG_INT 的原始中断状态。(R/WTC/SS)

UHCI_TX_HUNG_INT_RAW UHCI_TX_HUNG_INT 的原始中断状态。(R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW UHCI_SEND_S_REG_Q_INT 的原始中断状态。(R/WTC/SS)

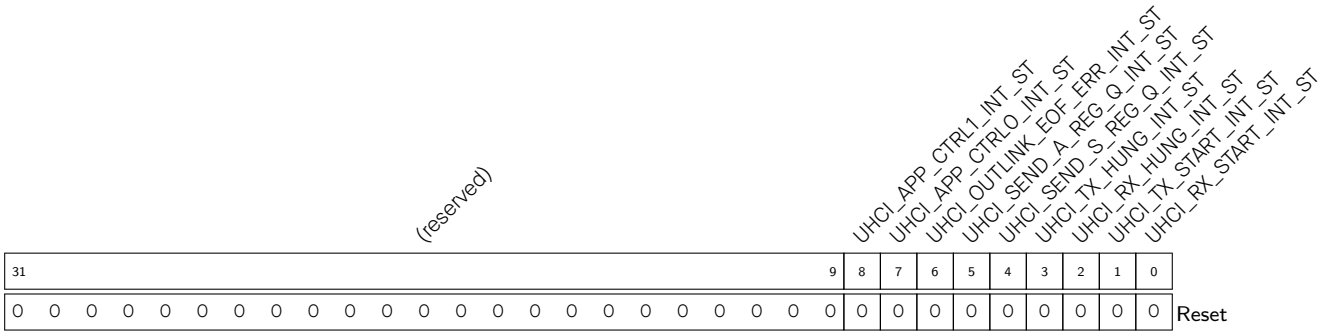
UHCI_SEND_A_REG_Q_INT_RAW UHCI_SEND_A_REG_Q_INT 的原始中断状态。(R/WTC/SS)

UHCI_OUT_EOF_INT_RAW UHCI_OUT_EOF_INT 的原始中断状态。(R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW UHCI_APP_CTRL0_INT 的原始中断状态。(R/W)

UHCI_APP_CTRL1_INT_RAW UHCI_APP_CTRL1_INT 的原始中断状态。(R/W)

Register 45.95. UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST UHCI_RX_START_INT 的屏蔽中断状态。(RO)

UHCI_TX_START_INT_ST UHCI_TX_START_INT 的屏蔽中断状态。(RO)

UHCI_RX_HUNG_INT_ST UHCI_RX_HUNG_INT 的屏蔽中断状态。(RO)

UHCI_TX_HUNG_INT_ST UHCI_TX_HUNG_INT 的屏蔽中断状态。(RO)

UHCI_SEND_S_REG_Q_INT_ST UHCI_SEND_S_REG_Q_INT 的屏蔽中断状态。(RO)

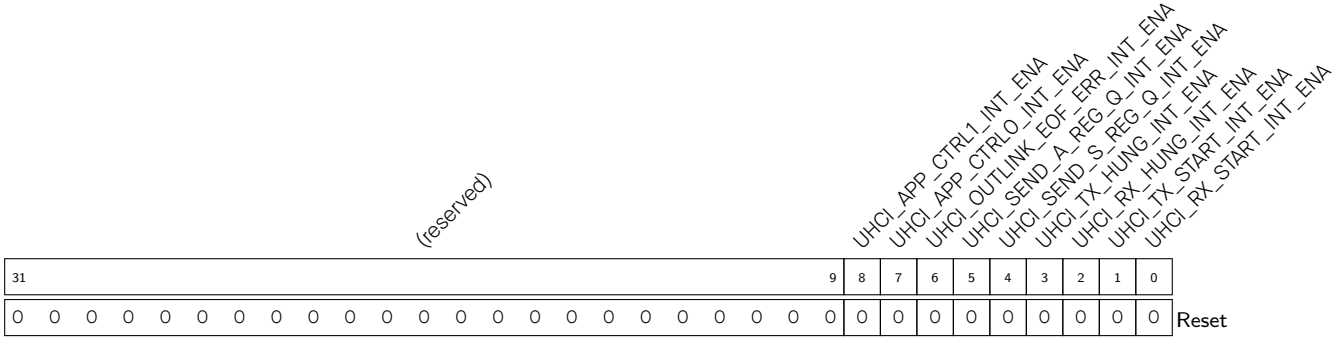
UHCI_SEND_A_REG_Q_INT_ST UHCI_SEND_A_REG_Q_INT 的屏蔽中断状态。(RO)

UHCI_OUTLINK_EOF_ERR_INT_ST UHCI_OUTLINK_EOF_ERR_INT 的屏蔽中断状态。(RO)

UHCI_APP_CTRL0_INT_ST UHCI_APP_CTRL0_INT 的屏蔽中断状态。(RO)

UHCI_APP_CTRL1_INT_ST UHCI_APP_CTRL1_INT 的屏蔽中断状态。(RO)

Register 45.96. UHCI_INT_ENA_REG (0x000C)



UHCI_RX_START_INT_ENA 写 1 使能 UHCI_RX_START_INT。(R/W)

UHCI_TX_START_INT_ENA 写 1 使能 UHCI_TX_START_INT。(R/W)

UHCI_RX_HUNG_INT_ENA 写 1 使能 UHCI_RX_HUNG_INT。(R/W)

UHCI_TX_HUNG_INT_ENA 写 1 使能 UHCI_TX_HUNG_INT。(R/W)

UHCI_SEND_S_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_S_REG_Q_INT。(R/W)

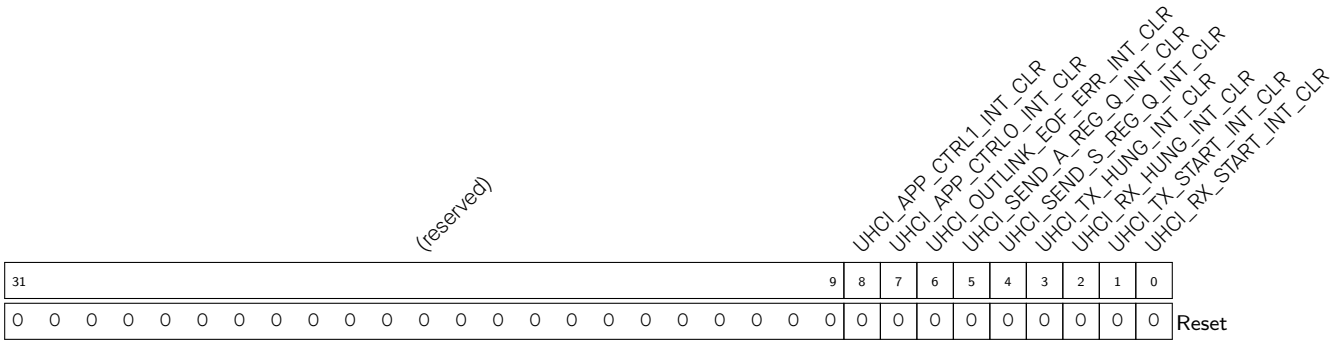
UHCI_SEND_A_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_A_REG_Q_INT。(R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA 写 1 使能 UHCI_OUTLINK_EOF_ERR_INT。(R/W)

UHCI_APP_CTRL0_INT_ENA 写 1 使能 UHCI_APP_CTRL0_INT。(R/W)

UHCI_APP_CTRL1_INT_ENA 写 1 使能 UHCI_APP_CTRL1_INT。(R/W)

Register 45.97. UHCI_INT_CLR_REG (0x0010)



UHCI_RX_START_INT_CLR 写 1 清除 UHCI_RX_START_INT。(WT)

UHCI_TX_START_INT_CLR 写 1 清除 UHCI_TX_START_INT。(WT)

UHCI_RX_HUNG_INT_CLR 写 1 清除 UHCI_RX_HUNG_INT。(WT)

UHCI_TX_HUNG_INT_CLR 写 1 清除 UHCI_TX_HUNG_INT。(WT)

UHCI_SEND_S_REG_Q_INT_CLR 写 1 清除 UHCI_SEND_S_REG_Q_INT。(WT)

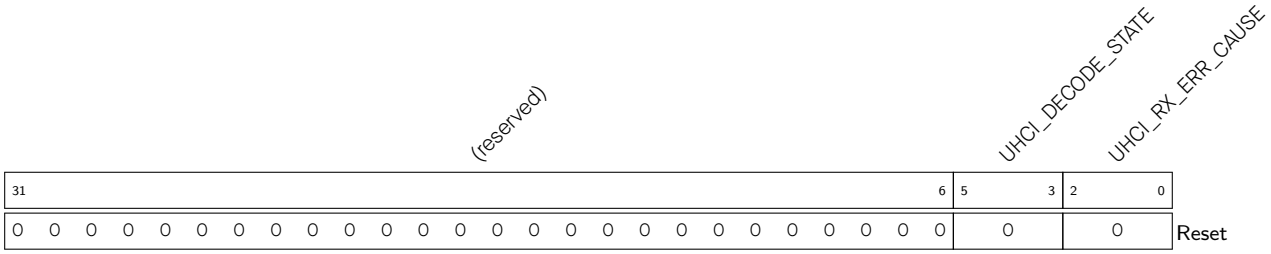
UHCI_SEND_A_REG_Q_INT_CLR 写 1 清除 UHCI_SEND_A_REG_Q_INT。(WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR 写 1 清除 UHCI_OUTLINK_EOF_ERR_INT。(WT)

UHCI_APP_CTRL0_INT_CLR 写 1 清除 UHCI_APP_CTRL0_INT。(WT)

UHCI_APP_CTRL1_INT_CLR 写 1 清除 UHCI_APP_CTRL1_INT。(WT)

Register 45.98. UHCI_STATE0_REG (0x0018)

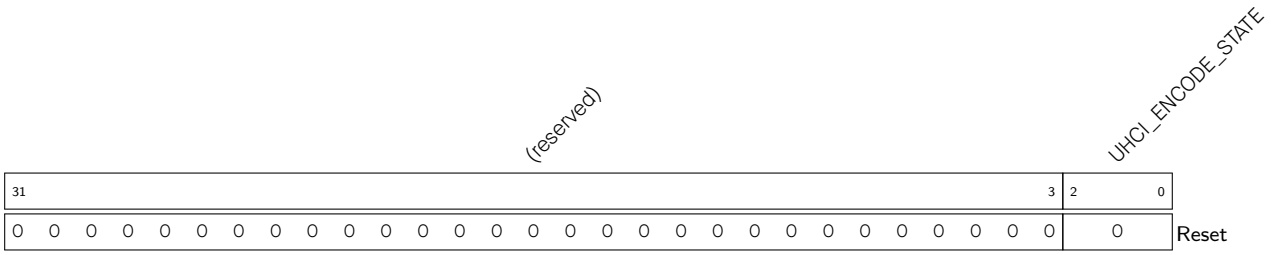


UHCI_RX_ERR_CAUSE 表示 DMA 接收数据错误的原因。

0：无效，没有作用
1：HCI 包校验和错误
2：HCI 包序列号错误
3：HCI 包 CRC 位错误
4：找到 0xC0 但接收的 HCI 包不完整
5：未找到 0xC0 但接收的 HCI 包完整
6：CRC 检测错误
7：无效，没有作用
(RO)

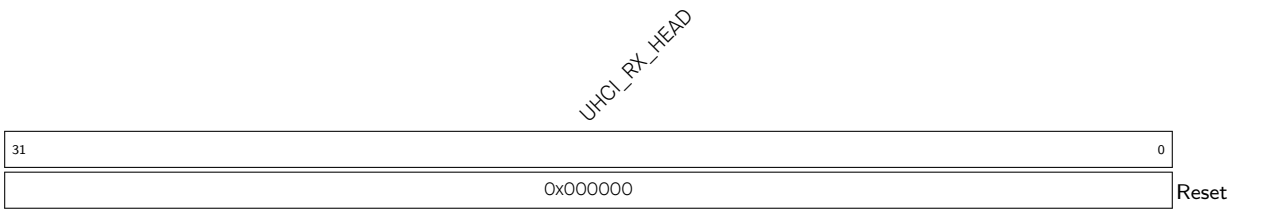
UHCI_DECODE_STATE 表示 UHCI 解码器状态。(RO)

Register 45.99. UHCI_STATE1_REG (0x001C)



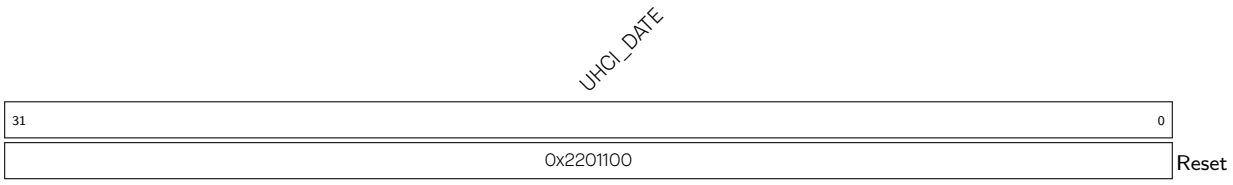
UHCI_ENCODE_STATE 表示 UHCI 编码器状态。(RO)

Register 45.100. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD 表示当前接收数据包的报头。(RO)

Register 45.101. UHCI_DATE_REG (0x0080)



UHCI_DATE 版本控制寄存器。(R/W)

第 46 章

SPI 控制器 (SPI)

说明：
本章仅适用于 ESP32-P4 芯片版本 v1.3。最新芯片版本的文档仍在开发中，将在后续 TRM 版本中发布。

46.1 概述

串行外设接口 (SPI) 是一种同步串行接口，可用于与外围设备进行通信。ESP32-P4 芯片集成了四个 SPI 控制器：

- MSPI 控制器，简称 MSPI，包括：
  - FLASH MSPI 控制器
    - FLASH MSPI SPI0
    - FLASH MSPI SPI1
  - PSRAM MSPI 控制器
    - PSRAM MSPI SPI0
    - PSRAM MSPI SPI1
- 通用 SPI2，简称 GP-SPI2
- 通用 SPI3，简称 GP-SPI3
- 低功耗 SPI，简称 LP-SPI

MSPI 控制器主要供内部使用以访问外部 flash 及 PSRAM，本章节不做介绍。本章节主要介绍 GP-SPI 控制器和 LP-SPI 控制器。**除非另有说明，否则在本章节中，GP-SPI 同时指 GP-SPI2 和 GP-SPI3**。

46.2 术语

为了更好地说明 GP-SPI 和 LP-SPI 的功能，本章使用了以下术语。

主机模式 GP-SPI 或 LP-SPI 用作 SPI 主机，发起 SPI 传输事务。

从机模式 GP-SPI 或 LP-SPI 用作 SPI 从机，当其 CS 被拉低时，与 SPI 主机进行数据传输。

MISO 主机输入，从机输出。数据从从机发送至主机。

MOSI 主机输出，从机输入。数据从主机发送至从机。

传输事务 一次完整的传输事务：主机拉低从机的 CS 线，开始传输数据，然后再拉高从机的 CS 线。传输事务为原子操作，即不可打断。

SPI 传输 SPI 主机与从机完成数据交换的一次完整过程。一次 SPI 传输可以包含一个或多个 SPI 传输事务。

单次传输 在这种传输模式下，仅包含一次传输事务。

CPU 控制的传输 由 CPU 控制，SPI_W0_REG~SPI_W15_REG（或 LP_SPI_W0_REG~LP_SPI_W15_REG）与 SPI 设备之间的数据传输。

DMA 控制的传输 由 DMA 引擎控制，DMA 与 SPI 设备之间的数据传输。

分段配置传输 GP-SPI2 用作主机时，DMA 控制的数据传输。此类传输包含多个传输事务（分段），每个传输事务均可独立配置。

从机连续传输 GP-SPI 用作从机时，DMA 控制的数据传输。此类传输包含多个传输事务（分段）。

全双工 主机与从机之间的发送线和接收线各自独立，发送数据和接收数据同时进行。

半双工 主机和从机只能有一方先发送数据，另一方接收数据。发送数据和接收数据不能同时进行。

四线全双工 四线包括：时钟线、片选线和两条数据线。其中，可使用两条数据线同时发送和接收数据。

四线半双工 四线包括：时钟线、片选线和两条数据线。其中，分时使用两条数据线，不可同时使用。

三线半双工 三线包括：时钟线、片选线和一条数据线。使用数据线分时发送和接收数据。

1-bit SPI 一个时钟周期传输一位数据。

(2-bit) Dual SPI 一个时钟周期传输两个数据位。

Dual Output Read Dual SPI 的一种数据模式，一个时钟周期可传输一位命令、或一位地址、或两位数据。

Dual I/O Read Dual SPI 的另外一种数据模式，一个时钟周期可传输一位命令、或两位地址、或两位数据。

(4-bit) Quad SPI 一个时钟周期传输四个数据位。

Quad Output Read Quad SPI 的一种数据模式，一个时钟周期可传输一位命令、或一位地址、或四位数据。

Quad I/O Read Quad SPI 的一种数据模式，一个时钟周期可传输一位命令、或四位地址、或四位数据。

QPI 一个时钟周期可传输四位命令、或四位地址、或四位数据。

(8-bit) Octal SPI 一个时钟周期传输八个数据位。

Octal Output Read Octal SPI 的一种数据模式，一个时钟周期可传输一位命令、或一位地址、或八位数据。

Octal I/O Read Octal SPI 的另外一种数据模式，一个时钟周期可传输一位命令、或八位地址、或八位数据。

OPI 一个时钟周期可传输八位命令、或八位地址、或八位数据。

SPI2 GP-SPI2 输入输出信号的前缀。SPI2 总线信号可通过 HP GPIO 交换矩阵或 HP IO MUX 与 GPIO 管脚相连。

SPI3 GP-SPI3 输入输出信号的前缀。SPI3 总线信号仅可通过 HP GPIO 交换矩阵连接到 GPIO 管脚。

LP_SPI LP-SPI 输入输出信号的前缀。LP_SPI 总线信号仅可通过 LP GPIO 交换矩阵连接到 GPIO 管脚。

46.3 特性

GP-SPI 具有以下特性：

- 用作主机或用作从机
- 支持半双工通信和全双工通信
- 支持 CPU 控制的传输类型以及 DMA 控制的传输类型
- 支持多种数据模式：
  - **GP-SPI2**
    - 1-bit SPI 模式
    - 2-bit Dual SPI 模式
    - 4-bit Quad SPI 模式
    - QPI 模式
    - 8-bit Octal SPI 模式（仅用于主机）
    - OPI 模式（仅用于主机）
  - **GP-SPI3**
    - 1-bit SPI 模式
    - 2-bit Dual SPI 模式
    - 4-bit Quad SPI 模式
    - QPI 模式
- 时钟频率可配置
  - 用作主机时：时钟频率可达 80 MHz
  - 用作从机时：时钟频率可达 60 MHz
- 数据长度可配置
  - 在 CPU 控制的主机和从机传输中：数据长度为 1~64 字节
  - 在 DMA 控制的主机单次传输中：数据长度为 1~32 KB
  - 在 DMA 控制的主机分段配置传输中：数据长度字节数无限制
  - 在 DMA 控制的从机单次或连续传输中：数据长度字节数无限制
- 读写数据的比特位顺序可配置
- 为 CPU 控制的传输和 DMA 控制的传输分别提供独立中断
- 时钟极性和相位可配置
- 四种 SPI 时钟模式：模式 0~ 模式 3
- 用作主机时，提供多条 CS 线
  - **GP-SPI2**：CS0~CS5
  - **GP-SPI3**：CS0~CS2

- 支持访问 SPI 接口的传感器、显示屏控制器、flash 或 RAM 芯片

LP-SPI 为 GP-SPI 的精简版，其功能为 GP-SPI 功能的子集，具有以下特性：

- 用作主机或用作从机
- 支持半双工通信和全双工通信
- 仅支持 CPU 控制的传输类型
- 仅支持 1-bit SPI 数据模式
- 时钟频率可配置
  - 用作主机时：时钟频率可达 40 MHz
  - 用作从机时：时钟频率可达 40 MHZ
- 数据长度可配置
  - 在 CPU 控制的主机和从机传输中：数据长度为 1~64 字节
- 读写数据的比特位顺序可配置
- 为 CPU 控制的传输提供中断
- 时钟极性和相位可配置
- 四种 SPI 时钟模式：模式 0~ 模式 3
- 用作主机时，仅提供 1 条 CS 线：CS0
- 用作从机时，支持唤醒功能（相较于 GP-SPI，属唯一新增功能）

46.4 架构概览

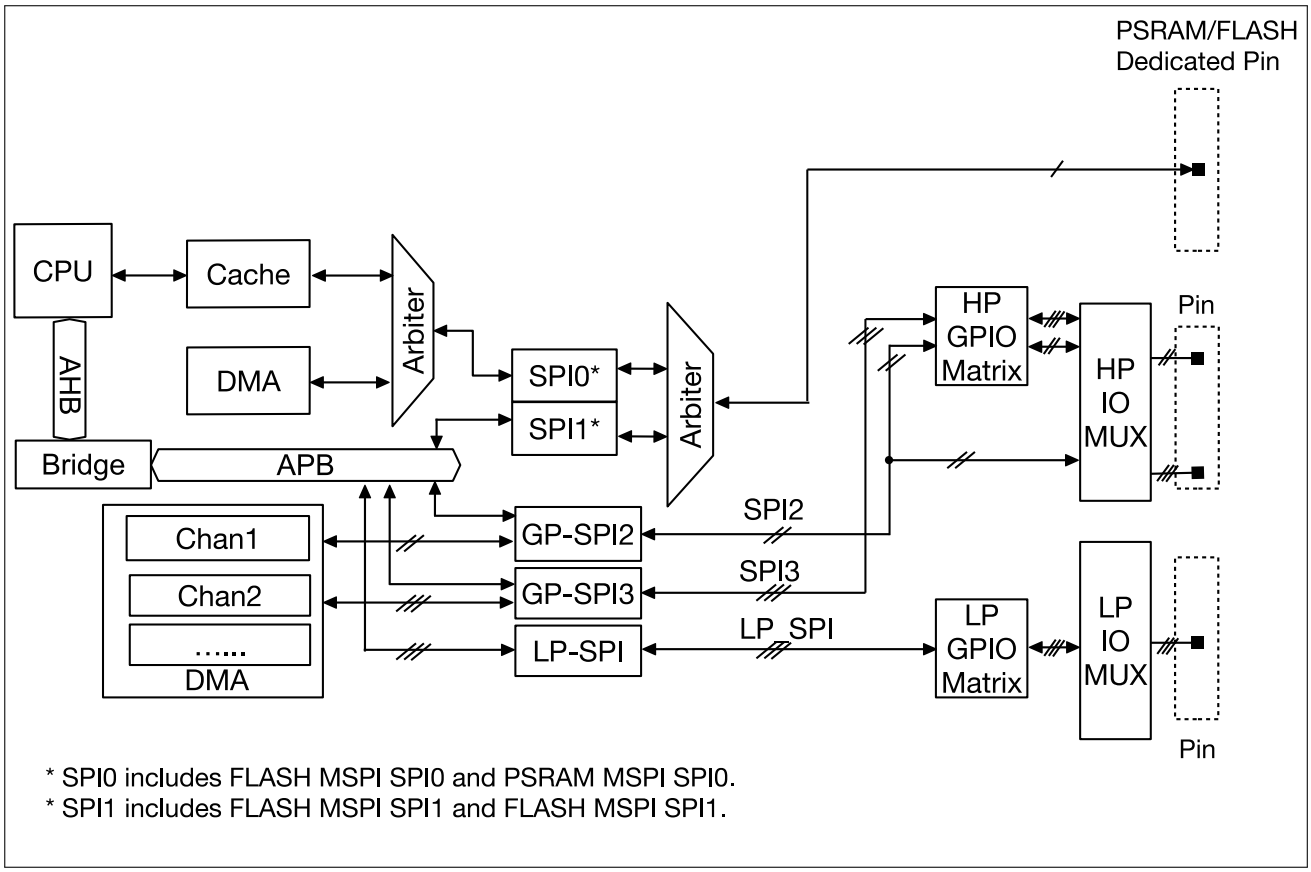


图 46.4-1. SPI 模块概览

图 46.4-1 所示为 SPI 模块的概览。GP-SPI2、GP-SPI3、LP-SPI 通过以下方式与 SPI 设备进行数据交换：

- 在 CPU 控制的传输中：
  - CPU <-> LP-SPI <-> SPI 设备
  - CPU <-> GP-SPI2 (GP-SPI3) <-> SPI 设备
- 在 DMA 控制的传输中：DMA <-> GP-SPI2 (GP-SPI3) <-> SPI 设备

本款芯片中，使用 “SPI2”、“SPI3” 和 “LP_SPI” 分别作为 GP-SPI2、GP-SPI3 和 LP-SPI 输入输出信号总线的前缀，用以说明 SPI 的总线功能。SPI2 总线信号可通过 HP GPIO 交换矩阵或 HP IO MUX 与 GPIO 管脚相连。SPI3 总线信号仅可通过 HP GPIO 交换矩阵连接到 GPIO 管脚，LP_SPI 总线信号仅可通过 LP GPIO 交换矩阵连接到 GPIO 管脚。更多信息，见章节 10 *GPIO 交换矩阵和 IO MUX*。

GP-SPI3 与 GP-SPI2 功能基本相同，LP-SPI 功能是 GP-SPI 功能的子集。GP-SPI2 功能描述见章节 46.5。GP-SPI2 和 GP-SPI3 及 LP-SPI 的功能差异见章节 46.5.1 和章节 46.10。

46.5 功能描述

46.5.1 数据模式

GP-SPI 和 LP-SPI 可配置成主机或从机，采用表 46.5-1 所示的数据模式与其他 SPI 设备进行通信。GP-SPI 和 LP-SPI 用作主机时，表中列出的各种数据模式见章节 46.5.9；用作从机时，表中列出的各种数据模式见章节 46.5.10。

表 46.5-1. GP-SPI 和 LP-SPI 支持的数据模式

数据模式		命令阶段	地址阶段	数据阶段	GP-SPI2	GP-SPI3	LP-SPI
1-bit SPI		1-bit	1-bit	1-bit	Y	Y	Y
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit	Y	Y	—
	Dual I/O Read	1-bit	2-bit	2-bit	Y	Y	—
Quad SPI	Quad Output Read	1-bit	1-bit	4-bit	Y	Y	—
	Quad I/O Read	1-bit	4-bit	4-bit	Y	Y	—
Octal SPI	Octal Output Read	1-bit	1-bit	8-bit	Y	—	—
	Octal I/O Read	1-bit	8-bit	8-bit	Y	—	—
QPI		4-bit	4-bit	4-bit	Y	Y	—
OPI		8-bit	8-bit	8-bit	Y	—	—

46.5.2 总线信号描述

SPI2/SPI3 总线信号和 LP_SPI 总线信号的功能描述如表 46.5-2。各种 SPI 模式下使用到的信号见表 46.5-3、表 46.5-4 和表 46.5-5。

表 46.5-2. SPI2/SPI3 总线信号和 LP_SPI 总线信号功能描述

SPI2 总线信号	SPI3 总线信号	LP-SPI 总线信号	描述
SPI2CLK	SPI3_CLK	LP_SPI_CLK	用作主机或从机时，输入输出时钟
SPI2CS0	SPI3_CS0	LP_SPI_CS	用作主机或从机时，输入输出片选信号
SPI2CS1~5	SPI3_CS1~2	—	用作主机时，输出片选信号
SPI2D	SPI3_D	LP_SPI_D	MOSI/SIO0：串行输入输出数据，比特 0
SPI2Q	SPI3_Q	LP_SPI_Q	MISO/SIO1：串行输入输出数据，比特 1
SPI2WP	SPI3_WP	—	SIO2：串行输入输出数据，比特 2
SPI2HD	SPI3_HD	—	SIO3：串行输入输出数据，比特 3
SPI2D4~7	—	—	SIO4~7：串行输入输出数据，比特 4~7
SPI2DQS	—	—	用作主机时，输出数据屏蔽信号

表 46.5-3. 各种 SPI 模式下使用到的 SPI2 总线信号

SPI2 总线信号	用作主机								用作从机					
	1-bit SPI			Dual SPI	Quad SPI	QPI	Octal SPI	OPI	1-bit SPI			Dual SPI	Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD						FD	3-line HD	4-line HD			
SPI2CLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI2CS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI2CS1	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS2	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS3	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS4	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2CS5	Y	Y	Y	Y	Y	Y	Y	Y						
SPI2D	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
SPI2Q	Y		(Y)[3]	Y[4]	Y[5]	Y	Y	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
SPI2WP					Y[5]	Y	Y	Y					Y[8]	Y
SPI2HD					Y[5]	Y	Y	Y					Y[8]	Y
SPI2D4~7							Y	Y						
SPI2DQS							Y	Y						

[1] FD：全双工

[2] HD：半双工

[3] 一次只使用两个信号中的一个

[4] 两个信号并行使用

[5] 四个信号并行使用

[6] 一次只使用两个信号中的一个

[7] 两个信号并行使用

[8] 四个信号并行使用

表 46.5-4. 各种 SPI 模式下使用到的 SPI3 总线信号

SPI3 总线信号	用作主机						用作从机					
	1-bit SPI			Dual SPI	Quad SPI	QPI	1-bit SPI			Dual SPI	Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD				FD	3-line HD	4-line HD			
SPI3_CLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI3_CS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPI3_CS1	Y	Y	Y	Y	Y	Y						
SPI3_CS2	Y	Y	Y	Y	Y	Y						
SPI3_D	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
SPI3_Q	Y		(Y)[3]	Y[4]	Y[5]	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
SPI3_WP					Y[5]	Y					Y[8]	Y
SPI3_HD					Y[5]	Y					Y[8]	Y

[1] FD：全双工

[2] HD：半双工

[3] 一次只使用两个信号中的一个

[4] 两个信号并行使用

[5] 四个信号并行使用

[6] 一次只使用两个信号中的一个

[7] 两个信号并行使用

[8] 四个信号并行使用

表 46.5-5. 各种 SPI 模式下使用到的 LP_SPI 总线信号

LP_SPI 总线信号	用作主机 (1-bit SPI)			用作从机 (1-bit SPI)		
	FD[1]	3-line HD[2]	4-line HD	FD	3-line HD	4-line HD
LP_SPI_CLK	Y	Y	Y	Y	Y	Y
LP_SPI_CS	Y	Y	Y	Y	Y	Y
LP_SPI_D	Y	Y	(Y)[3]	Y	Y	(Y)[3]
LP_SPI_Q	Y		(Y)[3]	Y		(Y)[3]

[1] FD：全双工
[2] HD：半双工
[3] 一次只使用两个信号中的一个

46.5.3 数据位读/写顺序控制

- GP-SPI
  - 用作主机时：
    * GP-SPI 主机发送的命令、地址和数据的位顺序由 SPI_WR_BIT_ORDER 控制；
    * 接收数据的位顺序由 SPI_RD_BIT_ORDER 控制。
  - 用作从机时：
    * GP-SPI 从机发送数据的位顺序由 SPI_WR_BIT_ORDER 控制；
    * 从机接收的命令、地址和数据的位顺序由 SPI_RD_BIT_ORDER 控制。
- LP-SPI
  - 用作主机时：
    * LP-SPI 主机发送的命令、地址和数据的位顺序由 LP_SPI_WR_BIT_ORDER 控制；
    * 接收数据的位顺序由 LP_SPI_RD_BIT_ORDER 控制。
  - 用作从机时：
    * LP-SPI 从机发送数据的位顺序由 LP_SPI_WR_BIT_ORDER 控制；
    * 从机接收的命令、地址和数据的位顺序由 LP_SPI_RD_BIT_ORDER 控制。

表 46.5-6 和表 46.5-7 所示为 SPI_RD/WR_BIT_ORDER 和 LP_SPI_RD/WR_BIT_ORDER 的功能。

表 46.5-6. GP-SPI 用作主机和从机时的数据位控制（以 GP-SPI2 为例）

位模式	SPI2 总线信号	SPI_RD/WR_BIT_ORDER = 0 (MSB)	SPI_RD/WR_BIT_ORDER = 2 (MSB)	SPI_RD/WR_BIT_ORDER = 1 (LSB)	SPI_RD/WR_BIT_ORDER = 3 (LSB)
1-bit 模式	SPI2D 或 SPI2Q	B7→B6→B5→B4→B3→B2→B1→B0	B7→B6→B5→B4→B3→B2→B1→B0	B0→B1→B2→B3→B4→B5→B6→B7	B0→B1→B2→B3→B4→B5→B6→B7
2-bit 模式	SPI2Q	B7→B5→B3→B1	B6→B4→B2→B0	B1→B3→B5→B7	B0→B2→B4→B6
	SPI2D	B6→B4→B2→B0	B7→B5→B3→B1	B0→B2→B4→B6	B1→B3→B5→B7
4-bit 模式	SPI2HD	B7→B3	B4→B0	B3→B7	B0→B4
	SPI2WP	B6→B2	B5→B1	B2→B6	B1→B5
	SPI2Q	B5→B1	B6→B2	B1→B5	B2→B6
	SPI2D	B4→B0	B7→B3	B0→B4	B3→B7
8-bit 模式	SPI2D7	B7	B7	B0	B0
	SPI2D6	B6	B6	B1	B1
	SPI2D5	B5	B5	B2	B2
	SPI2D4	B4	B4	B3	B3
	SPI2HD	B3	B3	B4	B4
	SPI2WP	B2	B2	B5	B5
	SPI2Q	B1	B1	B6	B6
	SPI2D	B0	B0	B7	B7

表 46.5-7. LP-SPI 用作主机和从机时的数据位控制

位模式	LP_SPI 总线信号	LP_SPI_RD/WR_BIT_ORDER = 0 (MSB)	LP_SPI_RD/WR_BIT_ORDER = 1 (LSB)
1-bit 模式	LP_SPI_D 或 LP_SPI_Q	B7→B6→B5→B4→B3→B2→B1→B0	B0→B1→B2→B3→B4→B5→B6→B7

46.5.4 非字节对齐传输

- 用作主机时，GP-SPI 主机可以按比特收发数据，比特长度等于 SPI_MS_DATA_BITLEN + 1，并且比特长度在 1/2/4/8 线模式下分别是 1/2/4/8 的倍数。
  - 发送长度不是 8 比特整数倍的数据时，软件需要填充最后不满 8 比特的数据为完整的 1 字节数据。
  - 接收长度不是 8 比特整数倍的数据时，不足 1 字节的部分依然按照 1 字节进行接收。
- 用作从机时，GP-SPI 从机可以按比特收发数据，并且比特长度在 1/2/4/8 线模式下分别是 1/2/4/8 的倍数。
  - 发送长度不是 8 比特整数倍的数据时，软件需要填充最后不满 8 比特的数据为完整的 1 字节数据。
  - 接收长度不是 8 比特整数倍的数据时，不足 1 字节的部分依然按照 1 字节进行接收，从机接收总长度可以通过 SPI_SLV_DATA_BITLEN 获取，并且最后 1 字节中的有效比特可以通过 SPI_SLV_LAST_BYTE_STRB 确定。

说明：

- 无论 GP-SPI 用作主机还是从机，不足 1 字节部分的接收/发送比特顺序仍然按照配置进行。
- LP-SPI 不支持非字节对齐传输。

46.5.5 传输类型

GP-SPI 用作主机或从机时支持的传输类型见下表。

表 46.5-8. GP-SPI 用作主机或从机时支持的传输类型

模式		CPU 控制的单次传输 [1]	DMA 控制的单次传输	DMA 控制的分段配置传输 [2]	DMA 控制的从机连续传输
主机	全双工	Y	Y	Y	—
	半双工	Y	Y	Y	—
从机	全双工	Y	Y	—	Y
	半双工	Y	Y	—	Y

[1] LP-SPI 用作主机或从机时仅支持 CPU 控制的单次传输。
[2] GP-SPI3 不支持 DMA 控制的分段配置传输。

以下章节将详细介绍上表中所列的各种传输类型。

46.5.6 CPU 控制的数据传输

说明：
GP-SPI 和 LP-SPI 均提供了 16 个 32-bit 的数据 buffer：

- GP-SPI：SPI_W0_REG~SPI_W15_REG
- LP-SPI：LP_SPI_W0_REG~LP_SPI_W15_REG

下文以 GP-SPI 数据 buffer 为例进行说明。

数据 buffer 结构见图 46.5-1。CPU 控制的传输表示在该次传输中发送的数据来自数据 buffer 或接收的数据存入数据 buffer。在这种传输类型中，每次传输事务均需要先配置相关寄存器，然后由 CPU 来触发。因此，CPU 控制的传输只能是单次传输，即仅包含一次传输事务。CPU 控制的传输支持全双工通信和半双工通信。

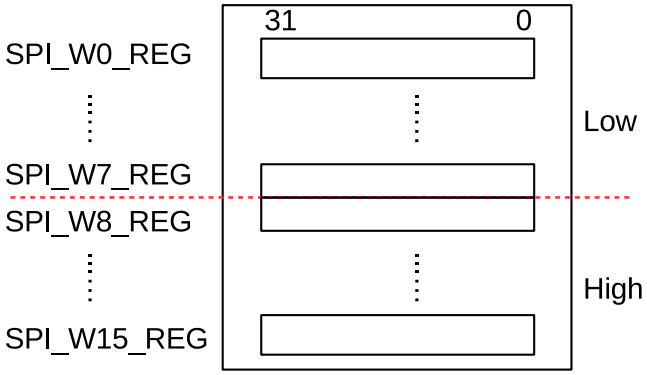


图 46.5-1. CPU 控制的传输中使用的数据 Buffer

46.5.6.1 CPU 控制的主机传输

GP-SPI 用作主机时，无论全双工通信还是半双工通信，CPU 控制的数据传输均通过 SPI_W0_REG~SPI_W15_REG 完成。此外，用户可通过配置 SPI_USR_MOSI_HIGHPART 和 SPI_USR_MISO_HIGHPART 选择“高位模式”仅使用其中部分寄存器，具体见下方描述。

- **TX 数据**
  - 未使能高位模式（SPI_USR_MOSI_HIGHPART 置 0）：此时，TX 数据取自 SPI_W0_REG~SPI_W15_REG，且每传输一个字节，取 TX 数据的地址即递增 1。**如果数据长度大于 64 字节，则 SPI_W0_REG~SPI_W15_REG 中的数据可能会被多次发送。**
    以 256 个字节为一个周期：
    * 字节 0~ 字节 63 依次发送 SPI_W0_REG~SPI_W15_REG 的数据。
    * 字节 64~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
    * 字节 256~ 字节 319（另一组数据的前 64 个字节）重新依次发送 SPI_W0_REG~SPI_W15_REG 的数据。以此类推。

    例如：发送 258 个字节（字节 0~ 字节 257），则：
    * 字节 0~ 字节 63 依次发送 SPI_W0_REG~SPI_W15_REG 的数据。
    * 字节 64~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
    * 字节 256~ 字节 257 重新依次发送地址 0 (SPI_W0_REG[7:0]) 和地址 1 (SPI_W0_REG[15:8]) 的数据：
      · 字节 256 取数据的地址为 256 对 64 取模的结果（256 % 64 = 0），即地址 0 (SPI_W0_REG[7:0])。
      · 字节 257 取数据的地址为 257 对 64 取模的结果（257 % 64 = 1），即地址 1 (SPI_W0_REG[15:8])。

- 使能高位模式（SPI_USR_MOSI_HIGHPART 置 1）：此时，TX 数据取自 SPI_W8_REG～SPI_W15_REG，且每传输一个字节，取 TX 数据的地址即递增 1。**如果数据长度大于 32 字节，则 SPI_W8_REG～SPI_W15_REG 中的数据将被多次发送**。
  以 256 个字节为一个周期：
  * 字节 0～ 字节 31 依次发送 SPI_W8_REG～SPI_W15_REG 中的数据。
  * 字节 32～ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
  * 字节 256～ 字节 287（另一组数据的前 32 个字节）重新依次发送 SPI_W8_REG～SPI_W15_REG 中的数据。以此类推。

  例如：发送 258 个字节（字节 0～ 字节 257），则：
  * 字节 0～ 字节 31 依次发送 SPI_W8_REG～SPI_W15_REG 的数据。
  * 字节 32～ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
  * 字节 256～ 字节 257 重新依次发送地址 0 (SPI_W8_REG[7:0]) 和地址 1 (SPI_W8_REG[15:8]) 的数据：
    · 字节 256 取数据的地址为 256 对 32 取模的结果（256 % 32 = 0），即地址 0 (SPI_W8_REG[7:0])。
    · 字节 257 取数据的地址为 257 对 32 取模的结果（257 % 32 = 1），即地址 1 (SPI_W8_REG[15:8])。

- **RX 数据**
  - 未使能高位模式（SPI_USR_MISO_HIGHPART 置 0）：此时，RX 数据存入 SPI_WO_REG～SPI_W15_REG，且每传输一个字节，存 RX 数据的地址即递增 1。**如果数据长度大于 64 字节，SPI_WO_REG～SPI_W15_REG 中的数据可能被覆盖**。
    以 256 个字节为一个周期：
    * 字节 0～ 字节 63 依次存入 SPI_WO_REG～SPI_W15_REG。
    * 字节 64～ 字节 255 重复存入 SPI_W15_REG[31:24]。
    * 字节 256～ 字节 319（另一组数据的前 64 个字节）重新依次存入 SPI_WO_REG～SPI_W15_REG。以此类推。

    例如：接收 258 个字节（字节 0～ 字节 257），则：
    * 字节 0～ 字节 63 依次存入 SPI_WO_REG～SPI_W15_REG。
    * 字节 64～ 字节 255 重复存入 SPI_W15_REG[31:24]。
    * 字节 256～ 字节 257 依次存入地址 0 (SPI_WO_REG[7:0]) 和地址 1 (SPI_WO_REG[15:8])：
      · 字节 256 存数据的地址为 256 对 64 取模的结果 (256 % 64 = 0)，即地址 0 (SPI_WO_REG[7:0])。
      · 字节 257 存数据的地址为 257 对 64 取模的结果 (257 % 64 = 1)，即地址 1 (SPI_WO_REG[15:8])。
  - 使能高位模式（SPI_USR_MISO_HIGHPART 置 1）：此时，RX 数据存入 SPI_W8_REG～SPI_W15_REG，且每传输一个字节，存 RX 数据的地址即递增 1。**如果数据长度大于 32 字节，则 SPI_W8_REG～SPI_W15_REG 中的数据将被覆盖**。
    以 256 个字节为一个周期：

* 字节 0~ 字节 31 依次存入 SPI_W8_REG~SPI_W15_REG。
* 字节 32~ 字节 255 重复存入 SPI_W15_REG[31:24]。
* 字节 256~ 字节 287（另一组数据的前 32 个字节）依次存入 SPI_W8_REG~SPI_W15_REG。以此类推。

例如： 接收 258 个字节（字节 0~ 字节 257），则：

* 字节 0~ 字节 31 依次存入 SPI_W8_REG~SPI_W15_REG。
* 字节 32~ 字节 255 重复存入 SPI_W15_REG[31:24]。
* 字节 256~ 字节 257 依次存入地址 0 (SPI_W8_REG[7:0]) 和地址 1 (SPI_W8_REG[15:8])：
  - 字节 256 存数据的地址为 256 对 32 取模的结果 (256 % 32 = 0)，即地址 0 (SPI_W8_REG[7:0])。
  - 字节 257 存数据的地址为 257 对 32 取模的结果 (257 % 32 = 1)，即地址 1 (SPI_W8_REG[15:8])。

说明：

- 上述的 TX/RX 数据均按字节寻址。
  - 如果未使能 “高位模式”，地址 0 表示 SPI_W0_REG[7:0]，地址 1 表示 SPI_W0_REG[15:8]，以此类推。
  - 如果使能了 “高位模式”，地址 0 表示 SPI_W8_REG[7:0]，地址 1 表示 SPI_W8_REG[15:8]，以此类推。

  最大地址为 SPI_W15_REG[31:24]。
- 为避免 TX/RX 数据传输错误，如 TX 数据重复发送或 RX 数据被覆盖等问题，请确保寄存器配置正确。
- LP-SPI 功能类似，但使用的是 LP-SPI 寄存器和 LP-SPI 数据 buffer，见章节 46.12.3。

46.5.6.2 CPU 控制的从机传输

GP-SPI 用作从机时，无论全双工通信或半双工通信，CPU 控制的数据传输均通过 SPI_W0_REG~SPI_W15_REG 完成，均采用按字节寻址。

- 全双工通信方式下：SPI_W0_REG~SPI_W15_REG 地址从 0 开始，且每传输一个字节，地址即递增 1。如果数据地址大于 63，则 SPI_W0_REG~SPI_W15_REG 中的数据会被覆盖。
- 半双工通信方式下：传输格式中 ADDR 的值即为 RX 数据或 TX 数据的起始地址，对应 SPI_W0_REG~SPI_W15_REG。每传输一个字节，则 RX 或 TX 地址即递增 1。如果地址大于 63（即大于最高地址：SPI_W15_REG[31:24]），SPI_W8_REG~SPI_W15_REG 中的数据会被覆盖。覆盖规律同主机模式下的 “高位模式”。

用户可根据具体应用，将 SPI_W0_REG~SPI_W15_REG

- 全部用作数据 buffer
- 部分用作数据 buffer，部分用作状态 buffer
- 全部用作状态 buffer

46.5.7 DMA 控制的数据传输

说明：
LP-SPI 不支持 DMA 控制的数据传输。下文描述仅适用于 GP-SPI。

在 DMA 控制的传输中，DMA RX 模块接收数据，DMA TX 模块发送数据。GP-SPI 用作主机和从机时均支持这种传输类型。

DMA 控制的传输可以是：

- 一次单次传输，仅包含一次传输事务。GP-SPI 用作主机和从机时均支持这种单次传输。
- 分段配置传输，包含多个传输事务（即多个分段）。仅有 GP-SPI2 用作主机时支持这种分段配置传输。更多信息，见章节 46.5.9.5。
- 从机连续传输，包含多次传输事务。仅有 GP-SPI 用作从机时支持这种从机连续传输。更多信息，见章节 46.5.10.3。

DMA 控制的传输只需由 CPU 触发一次即可完成多次传输事务。此类传输一旦被触发，DMA 引擎从 DMA 链接的内存中发送数据，或将收到的数据存入 DMA 链接的内存中，无需 CPU 的干预。

DMA 控制的传输支持全双工通信、半双工通信以及章节 46.5.9 和章节 46.5.10 所描述的功能。同时，DMA RX 模块与 DMA TX 模块互不影响，即支持四种全双工通信：

- 在 DMA 控制模式下接收数据，并在 DMA 控制模式下发送数据；
- 在 DMA 控制模式下接收数据，但在 CPU 控制模式下发送数据；
- 在 CPU 控制模式下接收数据，但在 DMA 控制模式下发送数据；
- 在 CPU 控制模式下接收数据，并在 CPU 控制模式下发送数据。

46.5.7.1 DMA 配置

- 选择 DMA 通道 *n*，并配置 GDMA TX/RX 描述符，见章节 6 *通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*；
- 置位 AXI_DMA_INLINK_START_CH*n* 或 AXI_DMA_OUTLINK_START_CH*n* 启动 GDMA RX/TX 引擎；
- 如果置位 AXI_DMA_OUTLINK_RESTART_CH*n*，则在所有 DMA TX buffer 用完之前，或在 DMA TX 引擎重置之前，新的 TX buffer 将会被添加到最后使用中的 TX buffer 结尾；
- DMA RX buffer 的链接方式与 DMA TX buffer 的链接方式相同，可通过置位 AXI_DMA_INLINK_START_CH*n* 或 AXI_DMA_INLINK_RESTART_CH*n* 来实现；
- TX 数据长度和 RX 数据长度分别由 DMA TX buffer 和 RX buffer 决定，TX 数据长度和 RX 数据长度范围为 0~32 KB；
- 启动 DMA 前，先初始化 DMA 接收链表 (inlink) 和发送链表 (outlink)。请置位寄存器 SPI_DMA_CONF_REG 中的 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA 位，否则读/写数据将存至或取自寄存器 SPI_W0_REG~SPI_W15_REG。

GP-SPI 用作主机时，如果置位了 AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA，则一次单次传输结束或一次分段配置传输结束，就会触发 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。

GP-SPI 用作从机时，如果置位了 AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA，则下表中任一情况均可触发 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。

表 46.5-9. GP-SPI 用作从机时数据传输的中断触发条件

传输类型	控制位[1]	控制位[2]	触发条件
从机单次传输	0	0	一次单次传输结束即触发该中断。
	1	0	一次单次传输结束，或接收的数据长度等于 SPI_MS_DATA_BITLEN + 1，即触发该中断。
从机连续传输	0	1	正确接收 CMD7 或 End_SEG_TRANS 命令即触发该中断。
	1	1	正确接收 CMD7 或 End_SEG_TRANS 命令、或接收的数据长度等于 SPI_MS_DATA_BITLEN + 1，即触发该中断。

[1] SPI_RX_EOF_EN

[2] SPI_DMA_SLV_SEG_TRANS_EN

46.5.7.2 DMA TX/RX Buffer 长度控制

配置的 DMA TX/RX buffer 长度最好应等于实际传输数据的长度。

- 如果配置的 DMA TX buffer 长度小于实际传输的数据长度，则多出来的数据将与最后传输的 TX buffer 数据相同。同时触发 SPI_OUTFIFO_EMPTY_ERR_INT 和 AXI_DMA_OUT_EOF_CH*n*_INT 中断。
- 如果配置的 DMA TX buffer 长度大于实际传输的数据长度，则 TX buffer 中的数据未被完全使用，即使稍后链接了新的 TX buffer，上个 TX buffer 中剩余的数据也将参与后续传输。请特别注意上述情况，或保存未使用的数据并复位 DMA。
- 如果配置的 DMA RX buffer 长度小于实际传输的数据长度，则多出来的数据将会丢失。同时触发 SPI_INFIFO_FULL_ERR_INT 和 SPI_TRANS_DONE_INT 中断。但不会触发 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。
- 如果配置的 DMA RX buffer 长度大于实际传输的数据长度，则 RX buffer 未被使用的部分被丢弃，下次传输直接使用后面链接的 RX buffer。

46.5.8 主机和从机数据流控制（以 GP-SPI 为例）

GP-SPI 用作主机和从机时均支持 CPU 控制的数据传输和 DMA 控制的数据传输。CPU 控制的数据传输发生在 SPI_W0_REG~SPI_W15_REG 和外围 SPI 设备之间。DMA 控制的数据传输发生在配置好的 DMA TX/RX buffer 和外围 SPI 设备之间。用户可在传输开始之前，配置 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA 来选择需要的传输类型。

46.5.8.1 GP-SPI 功能块图

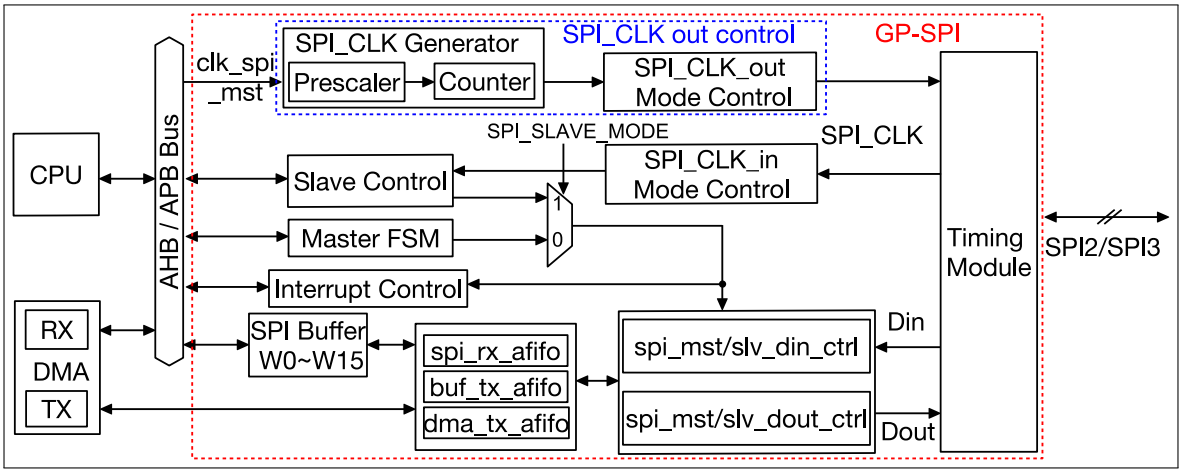


图 46.5-2. GP-SPI 功能块图

图 46.5-2 所示为 GP-SPI 主要的功能模块，包括：

- Master FSM：GP-SPI 的主机状态机。GP-SPI 用作主机时支持的所有功能，均由该状态机与寄存器共同控制。
- SPI Buffer: SPI_W0_REG~SPI_W15_REG，见图 46.5-1。CPU 控制的传输，其数据在 SPI buffer 中准备。
- 时序模块 (Timing Module)：捕获 SPI2/SPI3 总线上的数据。
- spi_mst/slv_din_ctrl 和 spi_mst/slv_dout_ctrl：用于将 TX/RX 数据转换成字节。
- spi_rx_afifo：暂存接收到的数据。
- buf_tx_afifo：暂存待发送的数据。
- dma_tx_afifo：暂存来自 DMA 的数据。
- clk_spi_mst：GP-SPI 模块时钟，在 GP-SPI 用作主机时，用于生成数据传输以及从机所需的 SPI_CLK 信号。
- SPI_CLK 生成器 (SPI_CLK Generator)：对 clk_spi_mst 进行分频生成 SPI_CLK 信号。分频系数由 SPI_CLKCNT_N 和 SPI_CLKDIV_PRE 共同决定。
- SPI_CLK_out Mode Control：发送数据传输以及从机所需的 SPI_CLK 信号。
- SPI_CLK_in Mode Control：当 GP-SPI 用作从机时，用于捕获 SPI 主机发出的 SPI_CLK 信号。

46.5.8.2 GP-SPI 用作主机时的数据流控制

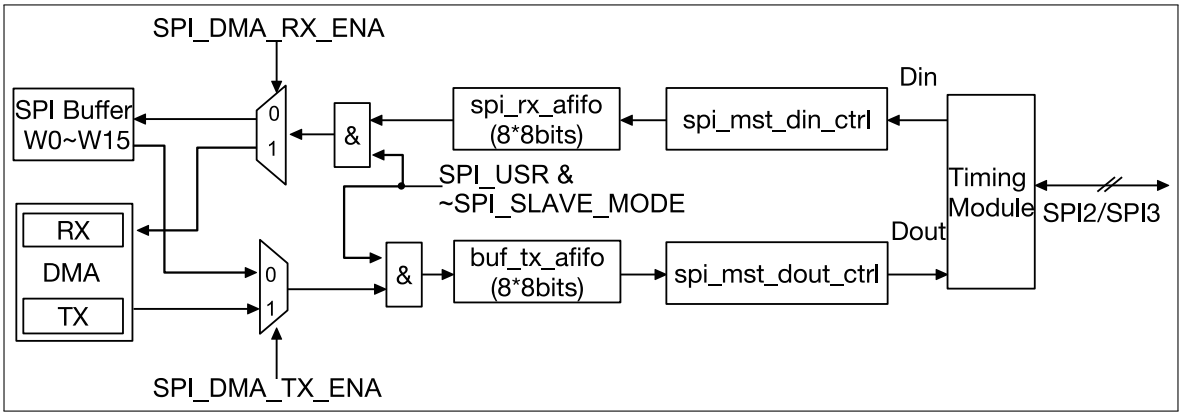


图 46.5-3. GP-SPI 用作主机时的数据流控制

图 46.5-3 所示为 GP-SPI 用作主机时的数据流。其控制逻辑如下：

- RX 数据：时序模块捕获 SPI2/SPI3 总线上的数据，然后 *spi_mst_din_ctrl* 模块将比特数据转化为字节数据，暂存于 *spi_rx_afifo* 中，此后根据控制方式转存至不同的接收位置：
  - CPU 控制：转存至 SPI_W0_REG ~SPI_W15_REG。
  - DMA 控制：转存至 DMA RX buffer。
- TX 数据：*buf_tx_afifo* 模块暂存待发送数据。根据控制方式不同，待发送数据来自不同的位置：
  - CPU 控制：TX 数据来自 SPI_W0_REG~SPI_W15_REG。
  - DMA 控制：TX 数据来自 DMA TX buffer。

buf_tx_afifo 中的数据会由时序模块以 1/2/4/(8)-bit 的模式发送出去。具体数据模式由 GP-SPI 状态机控制。时序模块可用于时序补偿。更多信息，见章节 46.8。

46.5.8.3 GP-SPI 用作从机时的数据流控制

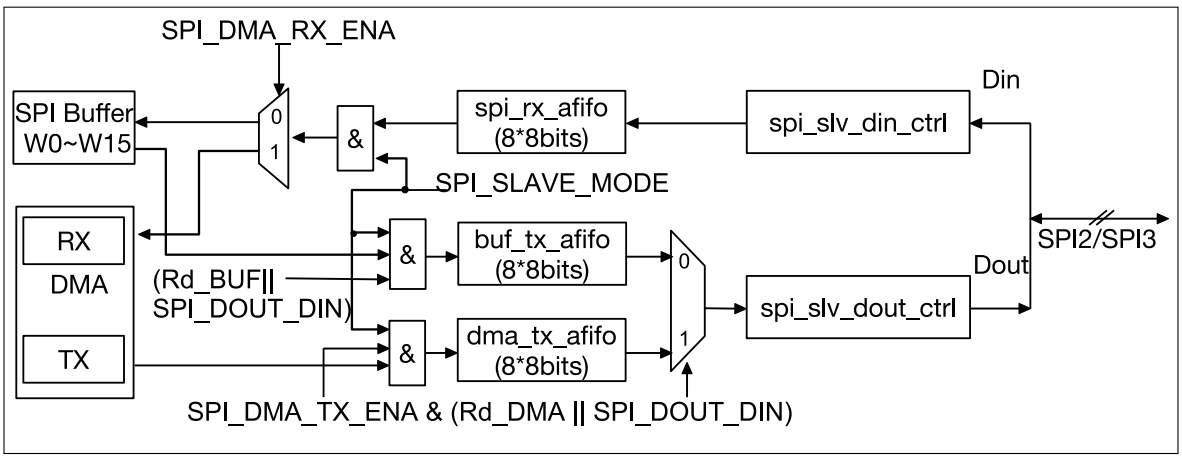


图 46.5-4. GP-SPI 用作从机时的数据流控制

图 46.5-4 所示为 GP-SPI 用作从机时的数据流控制。其控制逻辑如下：

- 在 CPU/DMA 控制的全双工/半双工传输下，当外部 SPI 主机发起 SPI 传输后，SPI2/SPI3 总线上的数据将被捕获，然后由 *spi_slv_din_ctrl* 模块转换为字节，暂存于 *spi_rx_afifo* 中。
  - 在 CPU 控制的全双工传输中，暂存于 *spi_rx_afifo* 中的 RX 数据之后会被转存到 SPI_W0_REG~SPI_W15_REG。
  - 在半双工 Wr_BUF 传输中，收到地址值 (SLV_ADDR[7:0]) 后，*spi_rx_afifo* 中暂存的 RX 数据将转存至寄存器 SPI_W0_REG ~SPI_W15_REG 的相应地址中。
  - 在 DMA 控制的全双工传输中，或在半双工 Wr_DMA 传输中，*spi_rx_afifo* 中暂存的 RX 数据将转存至配置好的 DMA RX buffer 中。
- 在 CPU 控制的全双工/半双工传输中，待发送的数据暂存在 *buf_tx_afifo* 中；而在 DMA 控制的全双工/半双工传输中，待发送的数据暂存在 *dma_tx_afifo* 中。因此，在一次从机连续传输中，CPU 控制的 Rd_BUF 传输事务和 DMA 控制的 Rd_DMA 传输事务可同时发生。
  - 在 CPU 控制的全双工传输中，如果置位了 SPI_SLAVE_MODE 和 SPI_DOUTDIN，同时清零了 SPI_DMA_TX_ENA，则 SPI_W0_REG~SPI_W15_REG 中的数据将被转存至 *buf_tx_afifo* 中。
  - 在 CPU 控制的半双工传输中，如果置位了 SPI_SLAVE_MODE，清零了 SPI_DOUTDIN，且收到指令 Rd_BUF 和地址 SLV_ADDR[7:0]，则 SPI_W0_REG~SPI_W15_REG 相应地址中的数据将被转存至 *buf_tx_afifo* 中。
  - 在 DMA 控制的全双工传输中，如果置位了 SPI_SLAVE_MODE、SPI_DOUTDIN 和 SPI_DMA_TX_ENA，则 DMA TX buffer 中的数据将被转存至 *dma_tx_afifo* 中。
  - 在 DMA 控制的半双工传输中，如果置位了 SPI_SLAVE_MODE，清零了 SPI_DOUTDIN，且收到指令 Rd_DMA，则 DMA TX buffer 中的数据将被转存至 *dma_tx_afifo* 中。

  buf_tx_afifo 或 *dma_tx_afifo* 中的数据将由 *spi_slv_dout_ctrl* 模块以 1/2/4/(8)-bit 的模式发送出去。

46.5.9 用作主机（以 GP-SPI 为例）

清零 SPI_SLAVE_REG 中 SPI_SLAVE_MODE 位可将 GP-SPI 配置成主机。此时，GP-SPI 提供时钟信号（GP-SPI 模块时钟的分频时钟）和 CS 信号。

46.5.9.1 主机状态机

GP-SPI 用作主机时，状态机在数据传输中控制其各个阶段，包括配置阶段 (CONF)、准备阶段 (PREP)、命令阶段 (CMD)、地址阶段 (ADDR)、空闲阶段 (DUMMY)、发送数据阶段 (DOUT) 和接收数据阶段 (DIN)。GP-SPI 主要用于访问 1/2/4/8-bit SPI 设备，如 flash、外部 RAM 等。因此，GP-SPI 各个阶段的命名规则应与 flash 以及外部 RAM 的时序名称保持一致。每个阶段的描述如下，GP-SPI 状态机的工作流程见图 46.5-5。

1. 空闲阶段 (IDLE)：GP-SPI 未处于工作状态或正在用作从机。
2. 配置阶段 (CONF)：仅用于 DMA 控制的分段配置传输（仅对 GP-SPI2 有效）。置位 SPI_USR 和 SPI_USR_CONF 使能该阶段。如果未使能该阶段，则说明当前传输为单次传输。
3. 准备阶段 (PREP)：准备 SPI 传输事务，控制 SPI CS 建立时间。置位 SPI_USR 和 SPI_CS_SETUP 使能该阶段。
4. 命令阶段 (CMD)：发送命令序列。置位 SPI_USR 和 SPI_USR_COMMAND 使能该阶段。
5. 地址阶段 (ADDR)：发送地址序列。置位 SPI_USR 和 SPI_USR_ADDR 使能该阶段。
6. 等待阶段 (DUMMY)：发送 DUMMY 序列。置位 SPI_USR 和 SPI_USR_DUMMY 使能该阶段。

7. 传输数据阶段 (DATA)：传输数据。
    - DOUT：发送数据。置位 SPI_USR 和 SPI_USR_MOSI 使能该阶段。
    - DIN：接收数据。置位 SPI_USR 和 SPI_USR_MISO 使能该阶段。
8. 结束阶段 (DONE)：控制 SPI CS 保持时间。置位 SPI_USR 使能该阶段。

注意：

主机状态机启动必须配置 SPI_USR。SPI_MST_FD_WAIT_DMA_TX_DATA 可以控制 SPI_USR 生效的时刻：

- 当 SPI_MST_FD_WAIT_DMA_TX_DATA 为 0 时，状态机相关状态在配置 SPI_USR 和其他控制寄存器后立即生效。
- 当 SPI_MST_FD_WAIT_DMA_TX_DATA 为 1 时，如果状态机也配置了 DOUT 状态，则会等待 *buf_tx_fifo* 中至少有数据后，SPI_USR 和其他控制寄存器才会生效，状态机开始启动。

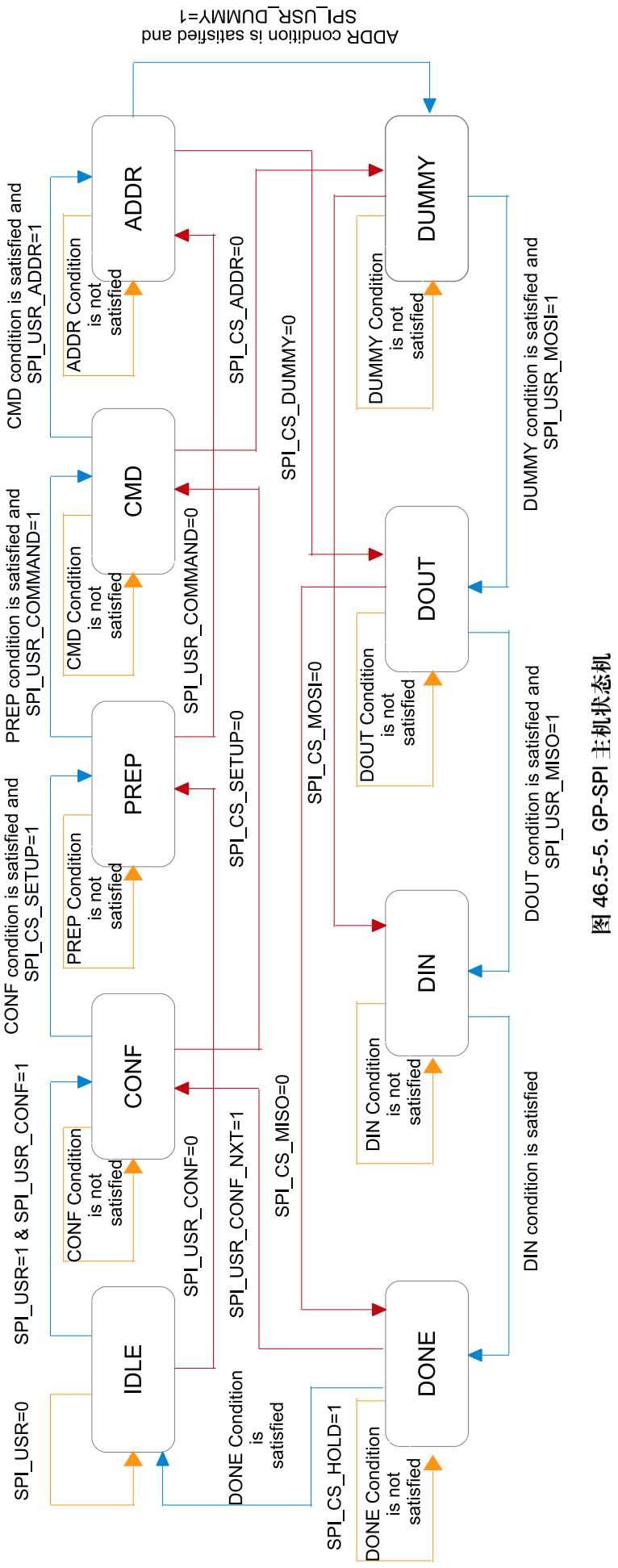


图 46.5-5. GP-SPI 主机状态机

图标说明：

- —: 表示相应的状态条件不满足，重复当前状态。
- —: 表示相应的寄存器已配置，状态条件已满足，将进行下一个状态。
- —: 表示相应的寄存器未配置，跳过下一个状态，或跳过后续多个状态。

上图中的各个状态条件描述如下：

- CONF condition: gpc[17:0] >= SPI_CONF_BITLEN[17:0]
- PREP condition: gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]
- CMD condition: gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]
- ADDR condition: gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]
- DUMMY condition: gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]
- DOUT condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DIN condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DONE condition: (gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)

状态机中用到了一个计数器 (gpc[17:0]) 来控制每个状态的周期长度。CONF、PREP、CMD、ADDR、DUMMY、DOUT 和 DIN 各状态可单独使能或禁用，也可以单独配置其周期长度。

46.5.9.2 状态控制和位模式控制寄存器

概述

表 46.5-10 列出了与 GP-SPI 状态控制相关的寄存器配置。如需使能 GP-SPI 的 QPI 模式，请置位寄存器 SPI_USER_REG 中 SPI_QPI_MODE 位。

表 46.5-10. 1/2/4/8-bit 模式下状态控制寄存器

状态	1-bit SPI2/SPI3 总线控制寄存器	2-bit SPI2/SPI3 总线控制寄存器	4-bit SPI2/SPI3 总线控制寄存器	8-bit SPI2 总线控制寄存器
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_OCT SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_OCT
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_MS_DATA_BITLEN	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_DUAL	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_QUAD	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_OCT
DOUT	SPI_USR_MOSI SPI_MS_DATA_BITLEN	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_DUAL	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_QUAD	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_OCT

如表 46.5-10 所示，如果希望在表格第一栏所示的状态中将 SPI2/SPI3 总线设置为相应的位模式（见表头），则需要配置该行中每一单元格的寄存器。

配置

例如，当 GP-SPI2 读取数据时，且希望实现：

- CMD 为 1-bit 模式
- ADDR 为 2-bit 模式
- DUMMY 为 8 个时钟周期
- DIN 为 4-bit 模式

则具体的寄存器配置如下：

1. 配置 CMD 状态相关寄存器。
   - 配置 SPI_USR_COMMAND_VALUE 为需要的命令值；
   - 配置 SPI_USR_COMMAND_BITLEN。SPI_USR_COMMAND_BITLEN 为所需要的命令位长 – 1；
   - 置位 SPI_USR_COMMAND；
   - 清除 SPI_FCMD_DUAL、SPI_FCMD_QUAD 和 SPI_FCMD_OCT。
2. 配置 ADDR 状态相关寄存器。
   - 配置 SPI_USR_ADDR_VALUE 为需要的地址值；
   - 配置 SPI_USR_ADDR_BITLEN。SPI_USR_ADDR_BITLEN 为所需要的地址位长 – 1；
   - 置位 SPI_USR_ADDR 和 SPI_FADDR_DUAL；
   - 清除 SPI_FADDR_QUAD 和 SPI_FADDR_OCT。
3. 配置 DUMMY 状态相关寄存器。
   - 在 SPI_USR_DUMMY_CYCLELEN 中配置 DUMMY 周期，其中 SPI_USR_DUMMY_CYCLELEN 的值等于 DUMMY 阶段所需要的时钟周期数 – 1；
   - 置位 SPI_USR_DUMMY。
4. 配置 DIN 状态相关寄存器。
   - 在 SPI_MS_DATA_BITLEN 中配置读数据的位长。SPI_MS_DATA_BITLEN 的值等于所需要的位长 – 1；
   - 置位 SPI_FREAD_QUAD 和 SPI_USR_MISO；
   - 清除 SPI_FREAD_DUAL 和 SPI_FREAD_OCT；
   - 如果选择了 DAM 控制的传输模式，则需要配置 DMA。如果选择了 CPU 控制的传输模式，则无需任何操作。
5. 清除 SPI_USR_MOSI，置位 SPI_UPDATE，硬件自动将目前为止所有配置更新到 clk_spi_mst 时钟域，并自动清除 SPI_UPDATE。
6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer。
7. 置位 SPI_USR 开始 GP-SPI2 传输。

说明：
本章节涉及主机模式下的寄存器配置，都需要置位 SPI_UPDATE 以将配置数据从 AHB_CLK 更新到 clk_spi_mst 时钟域，从机模式无需关注。具体的配置方式如上例。

写数据时 (DOUT)，需要配置 SPI_USR_MOSI，同时清除 SPI_USR_MISO。输出数据的位长等于 SPI_MS_DATA_BITLEN 加 1。在 CPU 控制的传输类型中，需要在数据 buffer (SPI_W0_REG~SPI_W15_REG) 中准备数据；在 DMA 控制的数据传输中，需要在 DMA TX buffer 中准备输出数据。字节顺序从 LSB (byte 0) 到 MSB 递增。

需特别注意 SPI_USR_COMMAND_VALUE 中的命令值以及 SPI_USR_ADDR_VALUE 中的地址值。

命令值的配置如下：

表 46.5-11. 命令值的发送顺序

COMMAND_BITLEN[1]	COMMAND_VALUE[2]	BIT_ORDER[3]	命令值的发送顺序
0~7	[7:0]	1	先发送 COMMAND_VALUE[COMMAND_BITLEN:0]。
		0	先发送 COMMAND_VALUE[7:7 – COMMAND_BITLEN]。
8~15	[15:0]	1	先发送 COMMAND_VALUE[7:0]，再发送 COMMAND_VALUE[COMMAND_BITLEN:8]。
		0	先发送 COMMAND_VALUE[7:0]，再发送 COMMAND_VALUE[15:15 – COMMAND_BITLEN]。

[1] SPI_USR_COMMAND_BITLEN：用于配置命令的位长。
[2] SPI_USR_COMMAND_VALUE：命令值写入的字段，见上表。
[3] SPI_WR_BIT_ORDER：0：先发送 LSB；1：先发送 MSB。

地址值配置如下：

表 46.5-12. 地址值的发送顺序

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	地址值的发送顺序
0~7	[31:24]	1	先发送 COMMAND_VALUE[ADDR_BITLEN + 24:24]。
		0	先发送 ADDR_VALUE[31:31 – ADDR_BITLEN]。
8~15	[31:16]	1	先发送 ADDR_VALUE[31:24]，再发送 ADDR_VALUE[ADDR_BITLEN + 8:16]。
		0	先发送 ADDR_VALUE[31:24]，再发送 ADDR_VALUE[23:31 - ADDR_BITLEN]。
16~23	[31:8]	1	先发送 ADDR_VALUE[31:16]，再发送 ADDR_VALUE[ADDR_BITLEN – 8:8]。
		0	先发送 ADDR_VALUE[31:16]，再发送 ADDR_VALUE[15:31 – ADDR_BITLEN]。
24~31	[31:0]	1	先发送 ADDR_VALUE[31:8]，再发送 ADDR_VALUE[ADDR_BITLEN – 24:0]。

		0	先发送 ADDR_VALUE[31:8]，再发送 ADDR_VALUE[7:31 – ADDR_BITLEN]。

[1] SPI_USR_ADDR_BITLEN：用于配置地址值的位长。
[2] SPI_USR_ADDR_VALUE：地址值写入的字段，见上表。
[3] SPI_WR_BIT_ORDER：0：先发送 LSB；1：先发送 MSB。

46.5.9.3 主机全双工通信（仅支持 1-bit 模式）

概述

GP-SPI 支持 SPI 全双工通信。在这种通信方式下，SPI 主机提供 CLK 和 CS 信号，然后与从机使用 1-bit 模式同时交换数据：MOSI（SPI2D/SPI3_D，发送），MISO（SPI2Q/SPI3_Q，接收）。用户可通过置位寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位使能全双工通信。GP-SPI2 与从机使用全双工通信时的连接方式见图 46.5-6。

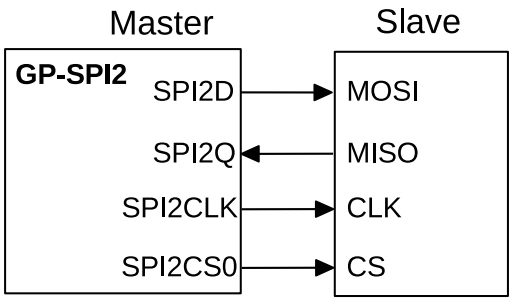


图 46.5-6. GP-SPI2 主机使用全双工与 SPI 从机通信

在全双工通信中，CMD、ADDR、DUMMY、DOUT 和 DIN 各个状态的具体行为可配置。通常，全双工模式跳过 CMD、ADDR 和 DUMMY 状态。传输数据的位长可在 SPI_MS_DATA_BITLEN 中配置。通信中使用的实际位长等于 (SPI_MS_DATA_BITLEN + 1)。

配置（以 GP-SPI2 为例）

按照以下操作步骤，开始数据传输：

- 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
- 配置 AHB 时钟（即 AHB_CLK，见章节 11 复位和时钟），并为 GP-SPI 配置模块时钟 (clk_spi_mst)；
- 置位 SPI_DOUTDIN 同时清除 SPI_SLAVE_MODE，使能主机全双工通信；
- 配置表 46.5-10 中所列的 GP-SPI2 寄存器；
- 配置 SPI CS 建立时间和保持时间，见章节 46.6；
- 设置 SPI2CLK 的极性，见章节 46.7；
- 根据选定的传输类型准备数据：
  - 如果选择的传输类型为 CPU 控制的 MOSI 传输，则需要在 SPI_W0_REG~SPI_W15_REG 中准备数据。
  - 如果选择了 DAM 控制的传输类型，则需要：
    * 配置 SPI_DMA_TX_ENA/SPI_DMA_RX_ENA；
    * 配置 DMA TX/RX 链表；

* 启动 DMA TX/RX 引擎，更多描述见章节 46.5.7 和章节 46.5.8。

- 配置中断，然后等待 SPI 从机做好传输准备；
- 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
- 置位寄存器 SPI_CMD_REG 中 SPI_USR 位，开始数据传输，然后等待之前配置的中断。

46.5.9.4 主机半双工通信（支持 1/2/4/(8)-bit 模式）

概述

在半双工通信下，GP-SPI 发送 CLK 和 CS 信号。在同一时刻，SPI 主机或从机只能有一个可以发送数据，另一个接收数据。用户可通过清除寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位使能半双工通信。SPI 半双工通信的通用格式为 CMD + [ADDR +] [DUMMY +] [DOUT 或 DIN]。其中，ADDR、DUMMY、DOUT 和 DIN 状态非必选，可单独禁用或启用。

如章节 46.5.9.2 所述，CMD、ADDR、DUMMY、DOUT 和 DIN 各个状态的周期、具体值和并行总线位模式等可独立配置。更多寄存器配置信息，见表 46.5-10。

半双工 GP-SPI 的详细属性如下：

1. CMD：0~16 位，主机发送，从机接收 (MOSI)。
2. ADDR：0~32 位，主机发送，从机接收。
3. DUMMY：0~256 个 SPI2CLK/SPI3_CLK 周期，主机发送，从机接收。
4. DOUT：在 CPU 控制的传输中，可传输 0~512 位（64 字节）数据；在 DMA 控制的单次传输中，可传输 0~256 Kbit (32 KB)；在 DMA 控制的分段配置传输中，可传输的数据长度无限制。主机发送，从机接收。
5. DIN：在 CPU 控制的传输中，可传输 0~512 位（64 字节）数据；在 DMA 控制的单次传输中，可传输 0~256 Kbit (32 KB)；在 DMA 控制的分段配置传输中，可传输的数据长度无限制。从机发送，主机接收 (MISO)。

GP-SPI2 还支持 1/2/4/8 位双倍数据速率模式 (Double Transfer Rate, DTR)，即在一个 SPI 时钟周期内，在上升沿和下降沿均收发数据：

- 置位 SPI_CMD_DTR_EN，则 CMD 值将在 DTR 模式下发送；否则，CMD 阶段将处于单倍数据速率模式 (Single Transfer Rate, STR)。
- 置位 SPI_ADDR_DTR_EN，则 ADDR 值将在 DTR 模式下发送；否则，ADDR 阶段将处于 STR 模式。
- 置位 SPI_DATA_DTR_EN，则 DIN 阶段的输入数据和 DOUT 阶段的输出数据将在 DTR 模式下发送；否则，数据将在 STR 模式下发送。

SPI_CMD_DTR_EN、SPI_ADDR_DTR_EN 和 SPI_DATA_DTR_EN 等控制位可分别单独配置，即 CMD 可配置成 STR 模式，而 ADDR 和 DOUT/DIN 可配置成 DTR 模式。

GP-SPI2 只能输出 SPI2DQS 信号，但无法接收该信号。因此，仅支持 DUMMY 固定的 flash 或者外部 RAM，即在一个读操作中，DUMMY 周期固定。

配置（以 GP-SPI2 为例）

则具体的寄存器配置如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟 (AHB_CLK)，并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；

3. 清除 SPI_DOUTDIN 和 SPI_SLAVE_MODE 位，使能主机半双工通信方式；
4. 配置表 46.5-10 中所列的 GP-SPI2 寄存器；
5. 配置 SPI CS 建立时间和保持时间，见章节 46.6；
6. 设置 SPI2CLK 的极性，见章节 46.7；
7. 根据选定的传输类型准备数据：
   - 如果选择的传输类型为 CPU 控制的 MOSI 传输，则需要在 SPI_W0_REG~SPI_W15_REG 中准备数据。
   - 如果选择了 DAM 控制的传输类型，则需要：
     - 配置 SPI_DMA_TX_ENA/SPI_DMA_RX_ENA；
     - 配置 DMA TX/RX 链表；
     - 启动 DMA TX/RX 引擎，更多描述见章节 46.5.7 和章节 46.5.8。
8. 配置中断，然后等待 SPI 从机做好传输准备；
9. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
10. 置位寄存器 SPI_CMD_REG 中 SPI_USR 位，开始数据传输，然后等待之前配置的中断。

应用示例（以 GP-SPI2 为例）

以下示例展示了 GP-SPI2 如何在主机半双工通信下访问 flash 和外部 RAM。

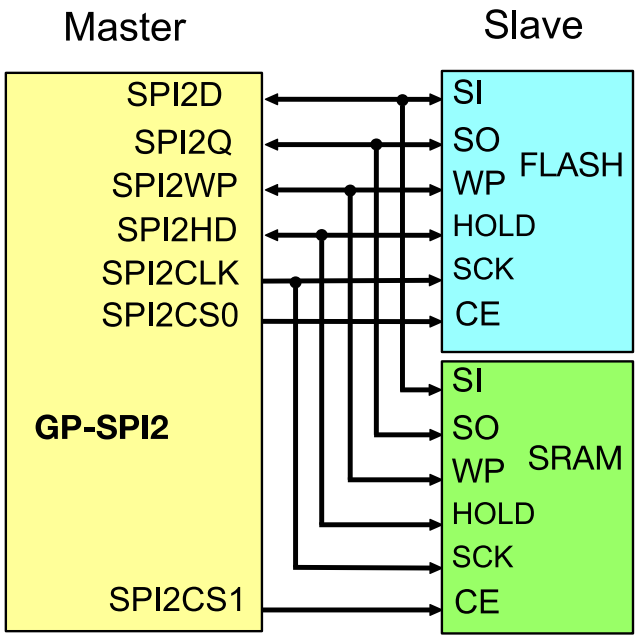


图 46.5-7. 4-bit 模式下 GP-SPI2 与 Flash 以及外部 RAM 的连接方式

图 46.5-8 所示为 GP-SPI2 按照标准 flash 规范进行 Quad I/O Read 操作。其他 GP-SPI 命令序列可以根据 SPI 从机的要求实现。

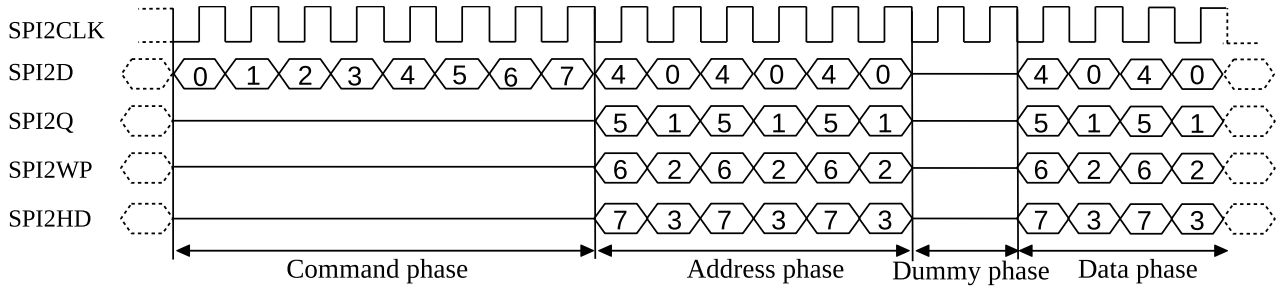


图 46.5-8. GP-SPI2 发送到 Flash 的 SPI Quad I/O Read 命令序列

46.5.9.5 DMA 控制的分段配置传输

说明：

- LP-SPI 和 GP-SPI3 均不支持 DMA 控制的分段配置传输。
- 由于跳过 CONF 阶段即可实现单次传输，因此不再另起章节单独介绍如何在 GP-SPI 用作主机时配置单次传输。

概述

GP-SPI2 用作主机时，可采用 DMA 控制的分段配置传输。

DMA 控制的主机传输可以是：

- 一次单次传输，仅包含一次传输事务。
- 分段配置传输，包含多个传输事务（即多个分段）。

如果选择了分段配置传输，则在每个分段中，寄存器均可单独配置。在分段配置传输中，仅需 CPU 触发一次，即可完成多次传输事务。具体工作流程见图 46.5-9。

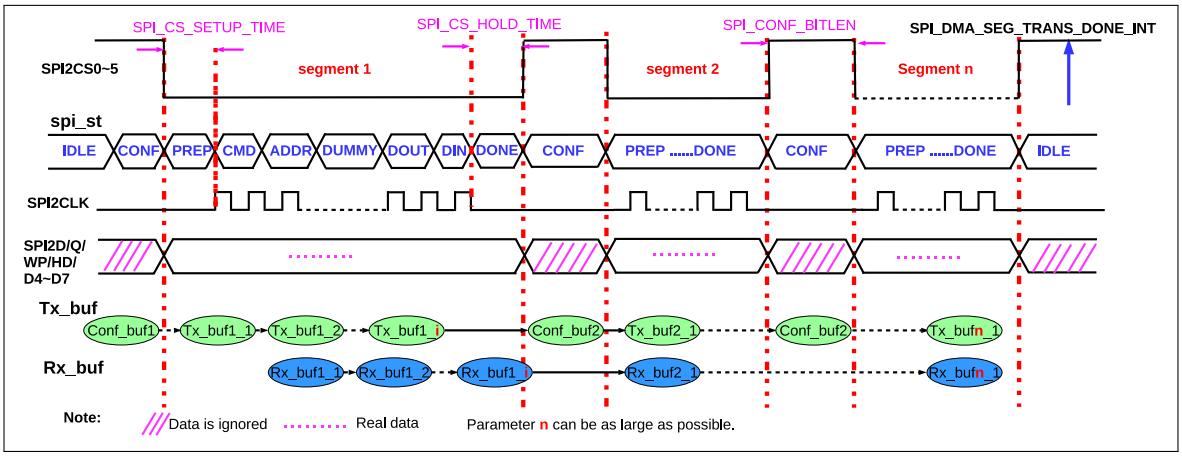


图 46.5-9. DAM 控制的分段配置传输

如图 46.5-9 所示，在分段配置传输中的某个单次传输事务 (segment *n*) 开始前，GP-SPI2 可在 CONF 阶段将寄存器重新按照 Conf_buf*n* 定义的内容进行配置。

建议为每个传输事务的 CONF 阶段提供单独的 DMA CONF 链表和 CONF buffer（即图 46.5-9 中的 Conf_buf*i*）。DMA TX 链表将所有的 CONF buffer 和 TX data buffer（即图 46.5-9 中的 Tx_buf*i*）链接起来，因此可以独立控制每个传输事务中的 SPI2 总线行为。

例如，在一次完整的分段配置传输中，传输事务 *i*、传输事务 *j* 和传输事务 *k* 可分别配置为全双工、半双工 MISO 和半双工 MOSI 模式。*i*、*j*、*k* 代表传输事务的编号。

同时，每个传输事务中，GP-SPI2 所使用到的各个阶段、各个阶段的相关值和 SPI2 总线周期长、以及 DMA 行为等，均可独立配置。当整个 DMA 控制的传输（包括多个传输事务）完成后，即触发 GP-SPI2 中断 SPI_DMA_SEG_TRANS_DONE_INT。

配置

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟 (AHB_CLK)，并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；
3. 清除 SPI_DOUTDIN 和 SPI_SLAVE_MODE 位，使能主机半双工通信；
4. 配置表 46.5-10 中所列的 GP-SPI2 寄存器；
5. 配置 SPI CS 建立时间和保持时间，见章节 46.6；
6. 设置 SPI2CLK 的极性，见章节 46.7；
7. 为每个传输事务准备 DMA CONF buffer 描述符和 TX data 描述符（可选）。把 CONF buffer 描述符和几次传输事务需要的 TX buffer 链接成一个链表；
8. 同样，为每个传输事务准备 RX buffer 描述符，并链接成一个链表；
9. 在该 DMA 控制的分段配置传输开始之前，为每个传输事务配置所需的 CONF buffer、TX buffer 和 RX buffer；
10. 配置 AXI_DMA_OUTLINK_ADDR_CH*n* 指向 CONF 和 TX buffer 描述符链表的首地址，之后置位 AXI_DMA_OUTLINK_START_CH*n*，启动 TX DMA；
11. 清除 SPI_DMA_CONF_REG 中 SPI_RX_EOF_EN 位。配置 AXI_DMA_INLINK_ADDR_CH*n* 指向 RX buffer 描述符链表的首地址，之后置位 AXI_DMA_INLINK_START_CH*n* 启动 RX DMA；
12. 置位 SPI_USR_CONF 使能 CONF 阶段；
13. 置位 SPI_DMA_SEG_TRANS_DONE_INT_ENA 使能 SPI_DMA_SEG_TRANS_DONE_INT 中断。如需配置其他中断，请参考章节 46.11；
14. 等待所有从机做好传输准备；
15. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
16. 置位 SPI_USR 开始本次 DMA 控制的分段配置传输；
17. 等待 SPI_DMA_SEG_TRANS_DONE_INT 中断，即 DMA 分段配置传输结束，数据已存储至相应内存。

说明：
在 PREP 阶段需要准备好本次分段传输发送的数据。应当保证：

$$(SPI_CS_SETUP_TIME + 1) \times T_{SPI_CLK} >= 4 \times (T_{AHB_CLK} + Tclk_spi_mst)$$

配置 CONF Buffer 和 Magic 值

在 GP-SPI2 分段配置传输中，仅有较上次传输事务有变动的寄存器会在 CONF 阶段被重新配置。为节省时间和芯片资源，其他寄存器配置则保持不变。

DMA CONF buffer*i* 中第一个字，即 SPI_BIT_MAP_WORD，记录传输事务 *i* 中，寄存器是否有改动。SPI_BIT_MAP_WORD 和待更新的 GP-SPI2 寄存器的对应关系见表 46.5-13，即位图 (BM) 表。如果位图表中某一位为 1，则在此次传输事务中，该位对应寄存器的值将被更新。如果其他寄存器不需要修改，则位图表中相应位应置为 0。

表 46.5-13. CONF 阶段 BM 位图

BM 位	寄存器	BM 位	寄存器
0	SPI_ADDR_REG	7	SPI_MISC_REG
1	SPI_CTRL_REG	8	SPI_DIN_MODE_REG
2	SPI_CLOCK_REG	9	SPI_DIN_NUM_REG
3	SPI_USER_REG	10	SPI_DOUT_MODE_REG
4	SPI_USER1_REG	11	SPI_DMA_CONF_REG
5	SPI_USER2_REG	12	SPI_DMA_INT_ENA_REG
6	SPI_MS_DLEN_REG	13	SPI_DMA_INT_CLR_REG

所有待修改寄存器的新值应紧跟在 SPI_BIT_MAP_WORD 之后，在 CONF buffer 中用连续的字表示。

为确保每个 CONF buffer 中内容正确，SPI_BIT_MAP_WORD[31:28] 位将用作 Magic 值，与寄存器 SPI_SLAVE_REG 中 SPI_DMA_SEG_MAGIC_VALUE 的值进行比较。SPI_DMA_SEG_MAGIC_VALUE 的值应在此 DMA 控制的分段配置传输开始之前配置，且在任何传输事务过程中均不可更改。

- 经比较，如果 SPI_BIT_MAP_WORD[31:28] == SPI_DMA_SEG_MAGIC_VALUE，则分段配置传输继续正常进行，整个传输过程结束则触发 SPI_DMA_SEG_TRANS_DONE_INT 中断。
- 如果 SPI_BIT_MAP_WORD[31:28] != SPI_DMA_SEG_MAGIC_VALUE，则 GP-SPI2 状态，即 spi_st 将返回至 IDLE 状态，分段配置传输立即结束。同时触发 SPI_DMA_SEG_TRANS_DONE_INT 中断，SPI_SEG_MAGIC_ERR_INT_RAW 位也将置 1。

CONF Buffer 配置示例

在一次分段配置传输中，传输事务 *i* 有 SPI_ADDR_REG、SPI_CTRL_REG、SPI_CLOCK_REG、SPI_USER_REG 和 SPI_USER1_REG 五个寄存器需要更新，则其 CONF buffer*i* 具体的配置示例见表 46.5-14 和表 46.5-15。

表 46.5-14. 传输事务 *i* 中 CONF buffer*i* 配置示例

CONF buffer*i*	说明
SPI_BIT_MAP_WORD	Buffer 中的第一个字。如果 SPI_DMA_SEG_MAGIC_VALUE 设置为 0xA，则本示例中该字的值为 0xA000001F。由表 46.5-15 可知，被置 1 的位有第 0、1、2、3 和 4 位，表示下列寄存器将被更新
SPI_ADDR_REG	CONF buffer*i* 的第二个字，存储 SPI_ADDR_REG 寄存器的更新值
SPI_CTRL_REG	CONF buffer*i* 的第三个字，存储 SPI_CTRL_REG 寄存器的更新值
SPI_CLOCK_REG	CONF buffer*i* 的第四个字，存储 SPI_CLOCK_REG 寄存器的更新值
SPI_USER_REG	CONF buffer*i* 的第五个字，存储 SPI_USER_REG 寄存器的更新值
SPI_USER1_REG	CONF buffer*i* 的第六个字，存储 SPI_USER1_REG 寄存器的更新值

表 46.5-15. BM 位图与待更新的寄存器

BM 位	值	寄存器	BM 位	值	寄存器
0	1	SPI_ADDR_REG	7	0	SPI_MISC_REG
1	1	SPI_CTRL_REG	8	0	SPI_DIN_MODE_REG
2	1	SPI_CLOCK_REG	9	0	SPI_DIN_NUM_REG
3	1	SPI_USER_REG	10	0	SPI_DOUT_MODE_REG
4	1	SPI_USER1_REG	11	0	SPI_DMA_CONF_REG
5	0	SPI_USER2_REG	12	0	SPI_DMA_INT_ENA_REG
6	0	SPI_MS_DLEN_REG	13	0	SPI_DMA_INT_CLR_REG

说明

使用 DMA 分段配置传输功能时，应注意以下寄存器相关位：

- SPI_USR_CONF：在置位 SPI_USR 之前，需先置位 SPI_USR_CONF，以使能本次传输。
- SPI_USR_CONF_NXT：如果传输事务 *i* 不是本次 DMA 控制的分段配置传输中的最后一次传输事务，则需要置位 SPI_USR_CONF_NXT。
- SPI_CONF_BITLEN：此外，在每个单独的传输事务中，GP-SPI2 的 CS 建立时间和保持时间可独立编程，更多配置信息见章节 46.6。在每次传输事务中，CS 保持高电平的时长约为：

$$(SPI_CONF_BITLEN + 5) \times T_{AHB_CLK}$$

  f_{AHB_CLK} 为 80 MHz 时，CONF 阶段的 CS 高电平时长可配置为 62.5 ns~3.2768 ms。如果 SPI_CONF_BITLEN 大于 0x3FFFA，(SPI_CONF_BITLEN + 5) 将溢出 (0x40000 - SPI_CONF_BITLEN - 5)。

46.5.10 用作从机（以 GP-SPI 为例）

GP-SPI 可用作从机与另一 SPI 主机进行通信。用作从机时，GP-SPI 支持特定格式的 1-bit SPI、2-bit Dual SPI、4-bit Quad SPI 和 QPI 模式。用户可置位寄存器 SPI_SLAVE_REG 中 SPI_SLAVE_MODE 位使能 GP-SPI 从机传输。

在传输过程中，CS 信号应保持低电平，CS 信号的下降沿和上升沿代表一次单次传输或从机连续传输的开始和结束。以 CS 低电平有效为例，GP-SPI 用作从机时，每次 SPI 传输之间的 CS 高电平无效持续时间应当不小于 8 T_{AHB_CLK}，保证每次传输都能正常结束。

46.5.10.1 可配置的通信格式

GP-SPI 用作从机时支持全双工通信和半双工通信。用户可配置寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位选择需要的通信方式。

全双工通信下，传输一开始，则数据同时输入和输出。此时，所有数据位均被视为输入/输出数据，即不需要命令、地址或 DUMMY 阶段。传输结束即触发 SPI_TRANS_DONE_INT 中断。

在半双工通信模式下，通信格式为 CMD+ADDR+DUMMY+DATA（DIN 或 DOUT）。

- “DIN” 表示 SPI 主机从 GP-SPI 中读取数据；
- “DOUT” 表示 SPI 主机向 GP-SPI 中写入数据。

每个阶段的详细特性如下：

1. CMD：
   - 表明 SPI 从机用于何种功能；
   - 一个字节，主机输出，从机输入；
   - 仅支持表 46.5-16 和表 46.5-17 所列的命令值；
   - 以 1-bit SPI 模式或 4-bit QPI 模式发送。
2. ADDR：
   - 在 CPU 控制的传输中，可以为 Wr_BUF 和 Rd_BUF 命令提供地址，或在其他命令中用作占位符，具体由应用定义；
   - 一个字节，主机输出，从机输入；
   - 可根据命令，以 1-bit，2-bit 或 4-bit 模式发送。
3. DUMMY：
   - DUMMY 的值无实际意义；SPI 从机在这个阶段准备数据；
   - SPI2 总线的位模式在这里也没有实际意义：
   - 持续八个 SPI_CLK 时钟周期。
4. DIN 或 DOUT：
   - 在 CPU 控制的传输中，可传输 0~64 字节数据；在 DMA 控制的传输中，传输数据长度无限制。
   - 可根据具体的 CMD 值，以 1-bit、2-bit 或 4-bit 模式发送。

说明：
半双工通信模式下，ADDR 和 DUMMY 阶段不可跳过。

半双工传输结束后，传输的 CMD 和 ADDR 的值分别锁存至 SPI_SLV_LAST_COMMAND 和 SPI_SLV_LAST_ADDR。如果 GP-SPI 不支持传输的 CMD 值，SPI_SLV_CMD_ERR_INT_RAW 将被置位。SPI_SLV_CMD_ERR_INT_RAW 仅可由软件清零。

46.5.10.2 半双工通信支持的 CMD 值

在半双工传输中，CMD 定义的值将决定传输类型。不支持的 CMD 值及其相关数据传输均被忽略，且 SPI_SLV_CMD_ERR_INT_RAW 将被置 1。传输格式为：CMD（8 位）+ ADDR（8 位）+ DUMMY（8 个 SPI_CLK 周期）+ DATA（单位：字节），CMD[3:0] 的详细说明如下：

- 0x1 (Wr_BUF)：CPU 控制的写操作。主机发送数据，GP-SPI 接收数据。数据将存储至相应地址的寄存器 SPI_W0_REG~SPI_W15_REG。
- 0x2 (Rd_BUF)：CPU 控制的读操作。主机接收 GP-SPI 发送的数据。数据来自相应地址的寄存器 SPI_W0_REG~SPI_W15_REG。
- 0x3 (Wr_DMA)：DMA 控制的写操作。主机发送数据，GP-SPI 接收数据。数据将存储至 GP-SPI 的 DMA RX buffer 中。
- 0x4 (Rd_DMA)：DMA 控制的读操作。主机接收 GP-SPI 发送的数据。数据来自 GP-SPI 接口的 DMA TX buffer。

- 0x7 (CMD7)：用于生成 SPI_SLV_CMD7_INT 中断。在从机连续传输下，使用 DMA RX 链表时，也可用于生成 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。但不会结束 GP-SPI 的从机连续传输。
- 0x8 (CMD8)：仅用于生成 SPI_SLV_CMD8_INT 中断，但不会结束 GP-SPI 的从机连续传输。
- 0x9 (CMD9)：仅用于生成 SPI_SLV_CMD9_INT 中断，但不会结束 GP-SPI 的从机连续传输。
- 0xA (CMDA)：仅用于生成 SPI_SLV_CMDA_INT 中断，但不会结束 GP-SPI 的从机连续传输。

CMD7、CMD8、CMD9 和 CMDA 的具体用途可由用户自定义。这些命令可用作握手信号、某些特定功能的密码、或某些用户自定义操作的触发信号等。

CMD、ADDR 和 DATA 阶段均支持 1/2/4-bit 模式，具体由 CMD[7:4] 决定。DUMMY 仅支持 1-bit 模式，且持续八个 SPI_CLK 时钟周期。CMD[7:4] 的具体定义如下：

- 0x0：CMD、ADDR 和 DATA 阶段均为 1-bit 模式。
- 0x1：CMD 和 ADDR 均为 1-bit 模式。DATA 为 2-bit 模式。
- 0x2：CMD 和 ADDR 均为 1-bit 模式。DATA 为 4-bit 模式。
- 0x5：CMD 为 1-bit 模式。ADDR 和 DATA 均为 2-bit 模式。
- 0xA：CMD 为 1-bit 模式，ADDR 和 DATA 均为 4-bit 模式。或 QPI 模式。

此外，CMD[7:0] 的值为 0x05、0xA5、0x06 和 0xDD 时，将跳过 DUMMY 和 DATA 阶段。CMD[7:0] 的具体定义如下：

- 0x05 (End_SEG_TRANS)：主机发送 0x05 命令，结束 SPI 模式下从机连续传输。
- 0xA5 (End_SEG_TRANS)：主机发送 0xA5 命令，结束 QPI 模式下从机连续传输。
- 0x06 (En_QPI)：GP-SPI 接收到 0x06 命令后，进入 QPI 模式。此时，寄存器 SPI_USER_REG 中 SPI_QPI_MODE 置位。
- 0xDD (Ex_QPI)：GP-SPI 接收到 0xDD 命令后，退出 QPI 模式。此时，SPI_QPI_MODE 位清零。

GP-SPI 支持的所有 CMD 值见表 46.5-16 和表 46.5-17。注意，DUMMY 仅支持 1-bit 模式，且持续八个 SPI_CLK 时钟周期。

表 46.5-16. GP-SPI 从机 SPI 模式支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
Wr_BUF	0x01	1-bit 模式	1-bit 模式	1-bit 模式
	0x11	1-bit 模式	1-bit 模式	2-bit 模式
	0x21	1-bit 模式	1-bit 模式	4-bit 模式
	0x51	1-bit 模式	2-bit 模式	2-bit 模式
	0xA1	1-bit 模式	4-bit 模式	4-bit 模式
Rd_BUF	0x02	1-bit 模式	1-bit 模式	1-bit 模式
	0x12	1-bit 模式	1-bit 模式	2-bit 模式
	0x22	1-bit 模式	1-bit 模式	4-bit 模式
	0x52	1-bit 模式	2-bit 模式	2-bit 模式
	0xA2	1-bit 模式	4-bit 模式	4-bit 模式
Wr_DMA	0x03	1-bit 模式	1-bit 模式	1-bit 模式
	0x13	1-bit 模式	1-bit 模式	2-bit 模式
	0x23	1-bit 模式	1-bit 模式	4-bit 模式

表 46.5-16. GP-SPI 从机 SPI 模式支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
	0x53	1-bit 模式	2-bit 模式	2-bit 模式
	0xA3	1-bit 模式	4-bit 模式	4-bit 模式
Rd_DMA	0x04	1-bit 模式	1-bit 模式	1-bit 模式
	0x14	1-bit 模式	1-bit 模式	2-bit 模式
	0x24	1-bit 模式	1-bit 模式	4-bit 模式
	0x54	1-bit 模式	2-bit 模式	2-bit 模式
	0xA4	1-bit 模式	4-bit 模式	4-bit 模式
CMD7	0x07	1-bit 模式	1-bit 模式	—
	0x17	1-bit 模式	1-bit 模式	—
	0x27	1-bit 模式	1-bit 模式	—
	0x57	1-bit 模式	2-bit 模式	—
	0xA7	1-bit 模式	4-bit 模式	—
CMD8	0x08	1-bit 模式	1-bit 模式	—
	0x18	1-bit 模式	1-bit 模式	—
	0x28	1-bit 模式	1-bit 模式	—
	0x58	1-bit 模式	2-bit 模式	—
	0xA8	1-bit 模式	4-bit 模式	—
CMD9	0x09	1-bit 模式	1-bit 模式	—
	0x19	1-bit 模式	1-bit 模式	—
	0x29	1-bit 模式	1-bit 模式	—
	0x59	1-bit 模式	2-bit 模式	—
	0xA9	1-bit 模式	4-bit 模式	—
CMDA	0x0A	1-bit 模式	1-bit 模式	—
	0x1A	1-bit 模式	1-bit 模式	—
	0x2A	1-bit 模式	1-bit 模式	—
	0x5A	1-bit 模式	2-bit 模式	—
	0xAA	1-bit 模式	4-bit 模式	—
End_SEG_TRANS	0x05	1-bit 模式	-	—
En_QPI	0x06	1-bit 模式	-	—

表 46.5-17. GP-SPI 从机 QPI 支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
Wr_BUF	0xA1	4-bit 模式	4-bit 模式	4-bit 模式
Rd_BUF	0xA2	4-bit 模式	4-bit 模式	4-bit 模式
Wr_DMA	0xA3	4-bit 模式	4-bit 模式	4-bit 模式
Rd_DMA	0xA4	4-bit 模式	4-bit 模式	4-bit 模式
CMD7	0xA7	4-bit 模式	4-bit 模式	—
CMD8	0xA8	4-bit 模式	4-bit 模式	—
CMD9	0xA9	4-bit 模式	4-bit 模式	—
CMDA	0xAA	4-bit 模式	4-bit 模式	—
End_SEG_TRANS	0xA5	4-bit 模式	4-bit 模式	—

表 46.5-17. GP-SPI 从机 QPI 支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
Ex_QPI	0xDD	4-bit 模式	4-bit 模式	—

GP-SPI 收到主机发送的 0x06 CMD (En_QPI) 命令后，将进入 QPI 模式。GP-SPI 在 QPI 模式下支持的传输类型，其后续所有阶段均为 4-bit 模式。如果收到 0xDD CMD (Ex_QPI)，则 GP-SPI 从机将返回到 SPI 模式。

未在表 46.5-16 和表 46.5-17 中列出的传输类型将被忽略掉。但如果 CS 低电平持续时长大于 2 个 APB_CLK 时钟周期，则将触发 SPI_TRANS_DONE_INT 中断。有关传输结束时触发的中断信息，请参考章节 46.11。

46.5.10.3 从机单次传输和从机连读传输

GP-SPI 用作从机时，支持由 DMA 和 CPU 控制的全双工和半双工通信。DMA 控制的从机传输，可以是一次单次传输，也可以是从机连续传输（包含多次传输事务）。CPU 控制的传输只能是单次传输，因为每次传输均需由 CPU 触发。

一次从机连续传输包含多个传输事务，每个传输事务可以是表 46.5-16 和 46.5-17 列出的任一传输类型。即在一次完整的连续传输过程中，可以包含 CPU 控制的数据传输，也可以包含 DMA 控制的数据传输。

在一次完整的连续传输过程中，推荐操作如下：

- CPU 控制的数据传输可用于握手通信以及少量数据传输。
- DMA 控制的数据传输可用于大量数据传输。

46.5.10.4 配置从机单次传输

用作从机时，GP-SPI 支持 CPU 控制的和 DMA 控制的全/半双工单次传输。

具体的寄存器配置如下（以 GP-SPI2 为例）：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟 (AHB_CLK)；
3. 置位 SPI_SLAVE_MODE 使能从机模式；
4. 配置 SPI_DOUTDIN：
   - 1：使能全双工通信。
   - 0：使能半双工通信。
5. 准备数据：
   - 如果选择的传输类型为 CPU 控制的传输，且 GP-SPI 发送数据，则在寄存器 SPI_W0_REG~SPI_W15_REG 中准备数据。
   - 如果选择的传输类型为 DMA 控制的传输，则需要：
     - 配置 SPI_DMA_TX_ENA/SPI_DMA_RX_ENA 和 SPI_RX_EOF_EN；
     - 配置 DMA TX/RX 链表；
     - 启动 DMA TX/RX 引擎，更多描述见章节 46.5.7 和章节 46.5.8。
6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；

7. 清零寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN 使能从机单次传输；
8. 置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_TRANS_DONE_INT_ENA，使能中断，并等待 SPI_TRANS_DONE_INT。在 DMA 控制模式下，使用 DMA RX buffer 时，推荐等待 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断，即数据已存储至相应内存。其他中断见章节 46.11。

46.5.10.5 配置半双工通信下从机连续传输

此模式必须使用 DMA。具体的寄存器配置如下（以 GP-SPI2 为例）：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 APB 时钟 (APB_CLK)；
3. 置位 SPI_SLAVE_MODE 使能从机模式；
4. 清除 SPI_DOUTDIN 使能半双工通信方式；
5. 根据需求，确定是否需要在寄存器 SPI_W0_REG~SPI_W15_REG 中准备数据；
6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
7. 置位 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA。清零 SPI_RX_EOF_EN。配置 DMA TX/RX 链表，并启动 DMA TX/RX 引擎，更多描述见章节 46.5.7 和章节 46.5.8；
8. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN，使能从机连续传输；
9. 置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_DMA_SEG_TRANS_DONE_INT_ENA，使能中断，并等待 SPI_DMA_SEG_TRANS_DONE_INT 中断。中断发生，即表明从机连续传输已结束，且数据已放入相应的内存中。其他中断见章节 46.11。

GP-SPI 收到 End_SEG_TRANS 命令（SPI 模式下为 0x05，QPI 模式下为 0xA5），从机连续传输结束，并触发 SPI_DMA_SEG_TRANS_DONE_INT 中断。

46.5.10.6 配置全双工通信下从机连续传输

在这一传输中，必须使用 DMA。数据从 DMA buffer 中输入输出。传输结束，触发 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。

具体的配置程序如下（以 GP-SPI2 为例）：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 APB 时钟 (APB_CLK)；
3. 置位 SPI_SLAVE_MODE 和 SPI_DOUTDIN，使能从机全双工通信；
4. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
5. 置位 SPI_DMA_TX_ENA/SPI_DMA_RX_ENA。配置 DMA TX/RX 链表，并启动 DMA TX/RX 引擎，更多描述见章节 46.5.7 和章节 46.5.8；
6. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_RX_EOF_EN 位。在寄存器 SPI_MS_DLEN_REG 的 SPI_MS_DATA_BITLEN[17:0] 中配置 DMA 接收数据长度（单位：比特）；
7. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN，使能从机连续传输；
8. 置位 AXI_DMA_IN_SUC_EOF_CH*n*_INT_ENA 使能中断，然后等待 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。

46.6 CS 建立时间和保持时间控制

SPI CS 建立时间和保持时间对于满足各种 SPI 设备（如 flash 或 PSRAM）的时序要求非常重要。

CS 建立时间为 CS 下降沿至 SPI_CLK 第一个锁存边沿的时间。模式 0 和模式 3 的第一锁存边沿为上升沿，模式 2 和模式 4 的第一锁存边沿为下降沿。

CS 保持时间为 SPI_CLK 最后一个锁存边沿到 CS 上升沿之间的时间。

从机模式下，CS 建立时间和保持时间应大于 0.5 x T_SPI_CLK，否则 SPI 传输可能出错。这里的 T_SPI_CLK 指 SPI_CLK 时钟周期。

主机模式下，CS 建立时间由寄存器 SPI_USER_REG 中的 SPI_CS_SETUP 位和寄存器 SPI_USER1_REG 中的 SPI_CS_SETUP_TIME 位控制：

- 清零 SPI_CS_SETUP，则 SPI CS 建立时间为 0.5 x T_SPI_CLK。
- 置位 SPI_CS_SETUP，则 SPI CS 建立时间为 (SPI_CS_SETUP_TIME + 1.5) x T_SPI_CLK。

CS 保持时间由寄存器 SPI_USER_REG 中的 SPI_CS_HOLD 位和寄存器 SPI_USER1_REG 中的 SPI_CS_HOLD_TIME 位控制：

- 清零 SPI_CS_HOLD，则 SPI CS 保持时间为 0.5 x T_SPI_CLK。
- 置位 SPI_CS_HOLD，则 SPI CS 保持时间为 (SPI_CS_HOLD_TIME + 1.5) x T_SPI_CLK。

图 46.6-1 和图 46.6-2 所示为访问外部 RAM 和 flash 时推荐的 CS 时序配置和寄存器配置。

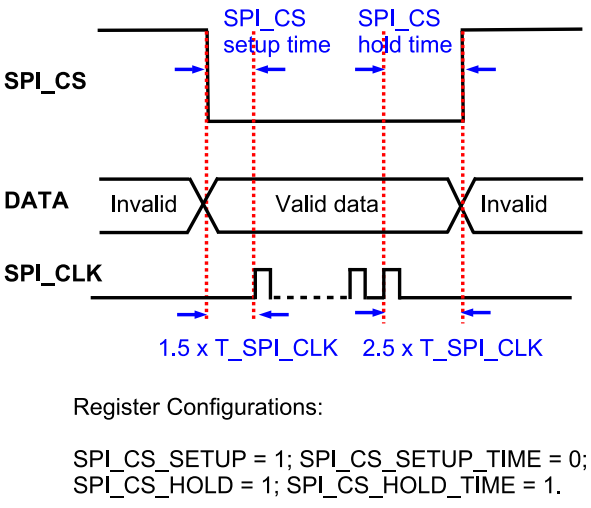
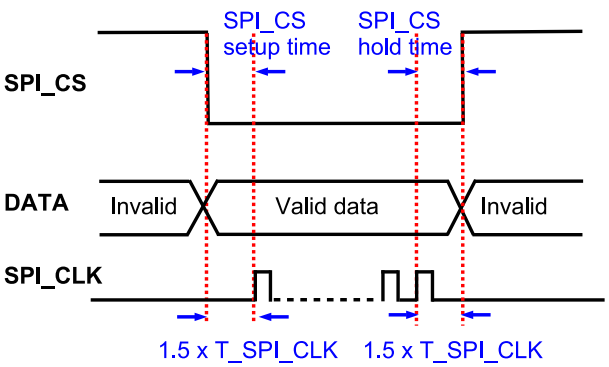


图 46.6-1. GP-SPI 访问外部 RAM 时推荐的 CS 时序配置



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 0.

图 46.6-2. GP-SPI 访问 Flash 时推荐的 CS 时序配置

46.7 时钟控制

46.7.1 GP-SPI 时钟控制

GP-SPI 中有以下四个时钟：

- clk_spi_mst：GP-SPI 模块时钟，可由多个时钟源分频所得，实际分频系数由 HP_SYS_CLKRST_GPSPI2/3_MST_CLK_DIV_NUM + 1 决定，具体使用的时钟源由 HP_SYS_CLKRST_GPSPI2/3_CLK_SRC_SEL 决定：
  - 0：XTAL_CLK
  - 1：RC_FAST_CLK
  - 2：SDIO_PLLO_CLK
  - 3：APLL_CLK
  - 4：SPLL_CLK (480 MHz)

  HP_SYS_CLKRST_GPSPI2/3_SYS_CLK_EN 用于使能所选择的时钟源。clk_spi_mst 在 GP-SPI 主机传输下用于生成数据传输以及从机所需的 SPI_CLK 信号。
- clk_hclk：GP-SPI 模块的时序补偿时钟，是 clk_spi_mst 的同源倍频时钟（2 倍频），具体时钟源同样由 HP_SYS_CLKRST_GPSPI2/3_CLK_SRC_SEL 决定。使用该时钟需要置位 HP_SYS_CLKRST_GPSPI2/3_HS_CLK_EN，并且实际分频系数由 HP_SYS_CLKRST_GPSPI2/3_HS_CLK_DIV_NUM + 1 决定。
- SPI_CLK：主机模式下的输出时钟，从机模式下的 IO 输入时钟。
- AHB_CLK：用于寄存器配置的时钟。

用作主机时，GP-SPI 最高输出时钟频率为 $f_{clk_spi_mst}$。如果需要较低的时钟频率，可以采用如下分频方式：

$$f_{SPI_CLK} = \frac{f_{clk_spi_mst}}{(SPI_CLKCNT_N + 1)(SPI_CLKDIV_PRE + 1)}$$

用户可配置寄存器 SPI_CLOCK_REG 中 SPI_CLKCNT_N 和 SPI_CLKDIV_PRE 设置分频系数。寄存器 SPI_CLOCK_REG 中 SPI_CLK_EQU_SYSCLK 置 1 时，GP-SPI 的输出时钟频率为 $f_{clk_spi_mst}$。如果采用其他整数分频，则 SPI_CLK_EQU_SYSCLK 应置 0。用作从机时，GP-SPI 支持的输入时钟频率 (f_{SPI_CLK}) 为：

- 如果 f_{AHB_CLK} >= 60 MHz，则输入时钟频率为：f_{SPI_CLK} <= 60 MHz。
- 如果 f_{AHB_CLK} < 60 MHz，则输入时钟频率为：f_{SPI_CLK} <= f_{AHB_CLK}。

46.7.2 LP-SPI 时钟控制

LP-SPI 中有以下时钟：

- lp_spi_pclk：用作寄存器配置的时钟，同时也用作 LP-SPI 模块时钟。在主机模式下用于生成数据传输以及从机所需的 LP_SPI_CLK。配置 LPPERI_CK_EN_LP_SPI 使能 lp_spi_pclk。
- LP_SPI_CLK：主机输出时钟。

46.7.3 时钟相位和极性

SPI 协议支持四种时钟模式，即模式 0~3，见图 46.7-1 和图 46.7-2。注，图片来源于 SPI 协议。

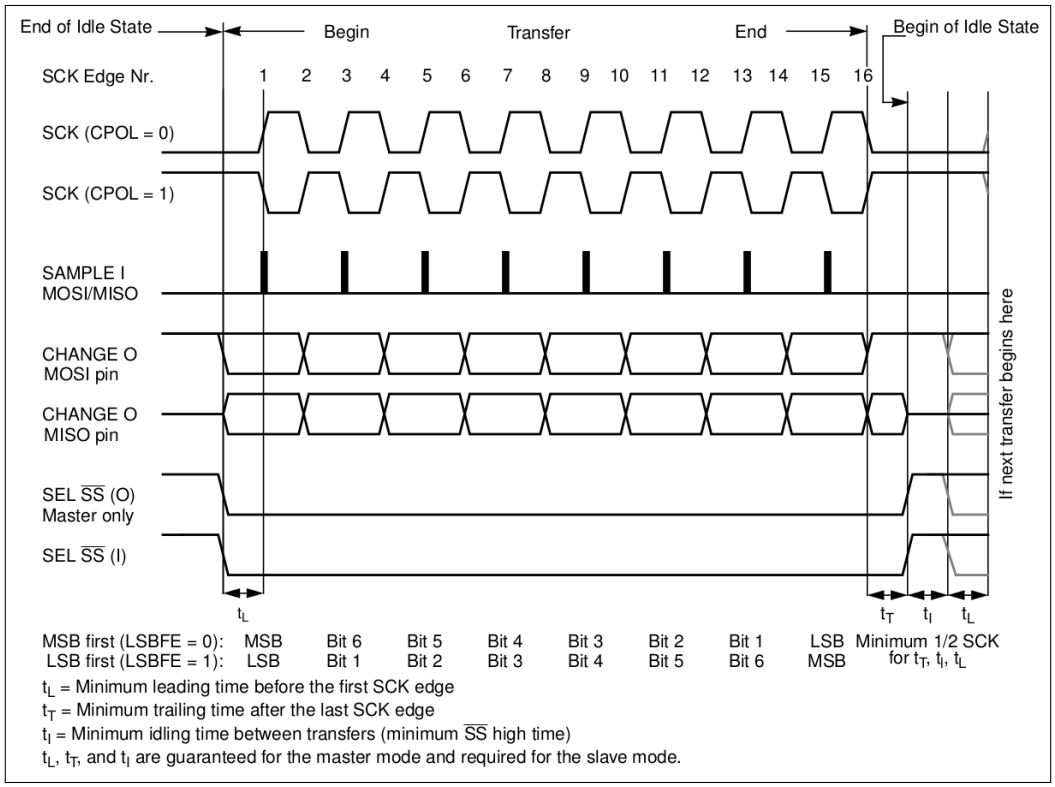


图 46.7-1. SPI 时钟模式 0 和时钟模式 2

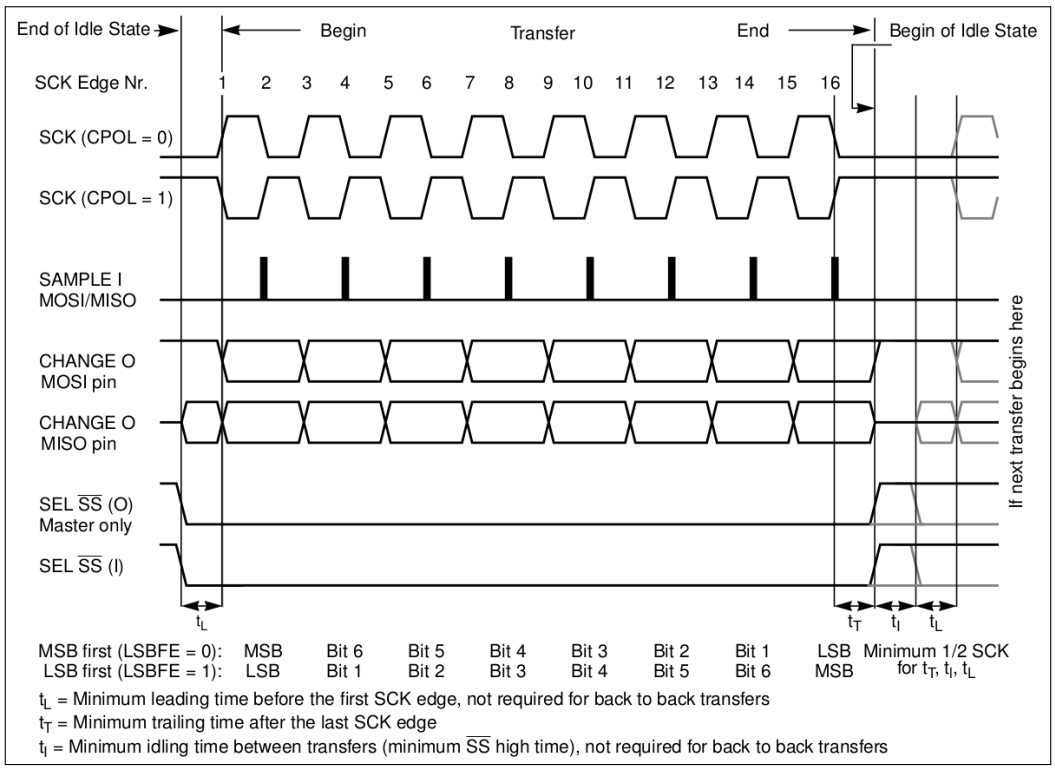


图 46.7-2. SPI 时钟模式 1 和时钟模式 3

- 模式 0：CPOL = 0，CPHA = 0；SPI 处于空闲模式时，SCK 为 0；数据在 SCK 下降沿变化，在上升沿采样。第一个数据在 SCK 的第一个下降沿之前被移出。
- 模式 1：CPOL = 0，CPHA = 1；SPI 处于空闲模式时，SCK 为 0；数据在 SCK 上升沿变化，在下降沿采样。
- 模式 2：CPOL = 1，CPHA = 0；SPI 处于空闲模式时，SCK 为 1；数据在 SCK 上升沿变化，在下降沿采样。第一个数据在 SCK 的第一个上升沿之前被移出。
- 模式 3：CPOL = 1，CPHA = 1；SPI 处于空闲模式时，SCK 为 1；数据在 SCK 下降沿变化，在上升沿采样。

46.7.4 主机时钟控制

GP-SPI 主机支持多种 SPI 时钟模式：模式 0~3。GP-SPI 极性和相位由寄存器 SPI_MISC_REG 中 SPI_CK_IDLE_EDGE 位和寄存器 SPI_USER_REG 中 SPI_CK_OUT_EDGE 位控制。SPI 时钟模式 0~3 的寄存器配置见表 46.7-1，可根据应用的路径延迟进行更改。

表 46.7-1. 主机时钟相位和极性配置

寄存器控制位	模式 0	模式 1	模式 2	模式 3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0

此外，SPI_CLK_MODE 可用于选择 CS 拉高时 SPI_CLK 的上升沿个数：0、1、2 或 SPI_CLK 一直有效。

说明：

SPI_CLK_MODE 配置成 1 或 2 时，必须置位 SPI_CS_HOLD 且 SPI_CS_HOLD_TIME 的值需大于 1。

46.7.5 从机时钟控制

GP-SPI 从机也支持四种 SPI 时钟模式：即模式 0~3。寄存器 SPI_USER_REG 中 SPI_TSCK_I_EDGE 和 SPI_RSCK_I_EDGE 位可用于配置时钟极性和相位。数据的输出沿则由寄存器 SPI_SLAVE_REG 中的 SPI_CLK_MODE_13 位控制。寄存器具体配置见表 46.7-2。

表 46.7-2. 从机时钟相位和极性配置

寄存器控制位	模式 0	模式 1	模式 2	模式 3
SPI_TSCK_I_EDGE	0	1	1	0
SPI_RSCK_I_EDGE	0	1	1	0
SPI_CLK_MODE_13	0	1	0	1

46.8 GP-SPI 时序补偿

概述

SPI 输入输出信号可通过 GPIO 矩阵或 IO MUX 映射到芯片管脚，但 IO MUX 不支持时序调整。输入输出数据在 GPIO 矩阵模块中，可在上升沿或下降沿延迟 1 或 2 个 IO MUX 运行时钟周期。更多寄存器配置信息，见章节 *10 GPIO 交换矩阵和 IO MUX*。仅 GP-SPI 支持时序补偿。

图 46.8-1 所示为 GP-SPI 用作主机时的时序补偿控制，包括以下路径：

- “CLK”：GP-SPI 总线时钟信号的输出路径。时钟由 SPI_CLK 输出控制模块发送，经过 GPIO 矩阵或 IO MUX，然后到达外部 SPI 设备。
- “IN”：GP-SPI 的数据输入路径，即图 46.8-1 中路径 3（紫色）。来自外部 SPI 设备的输入数据通过 GPIO 矩阵或 IO MUX，由时序模块（见图 46.5-2）进行调整，最后存储到 *spi_rx_afifo*。
- “OUT”：GP-SPI 的数据输出路径，即图 46.8-1 中路径 2（玫红色）。输出数据发送到时序模块，经过 GPIO 矩阵或 IO MUX，最后由外部 SPI 设备捕获。

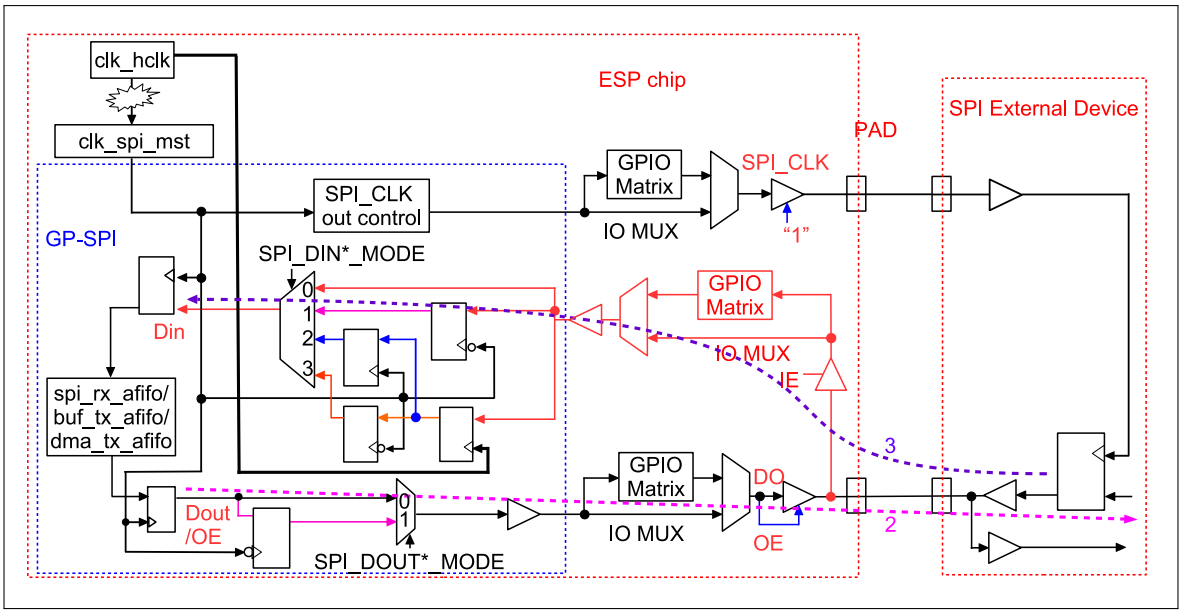


图 46.8-1. GP-SPI 主机时序补偿控制图

时序调制模块同时适用于输入数据和输出数据，数据可在时钟上升沿或下降沿延迟整数个 clk_spi_mst 周期，即整数个 $T_{clk_spi_mst}$。

关键寄存器

- SPI_DIN_MODE_REG：用于选择输入数据的锁存沿
- SPI_DIN_NUM_REG：用于选择输入数据的延迟周期
- SPI_DOUT_MODE_REG：用于选择输出数据的锁存沿

时序补偿应用示例（以 GP-SPI2 为例）

图 46.8-2 所示为 GP-SPI2 用作主机时的时序补偿示例。同时，DUMMY 周期长可更改，用于补偿实际的 I/O 线路延迟，从而提高 GP-SPI2 性能。

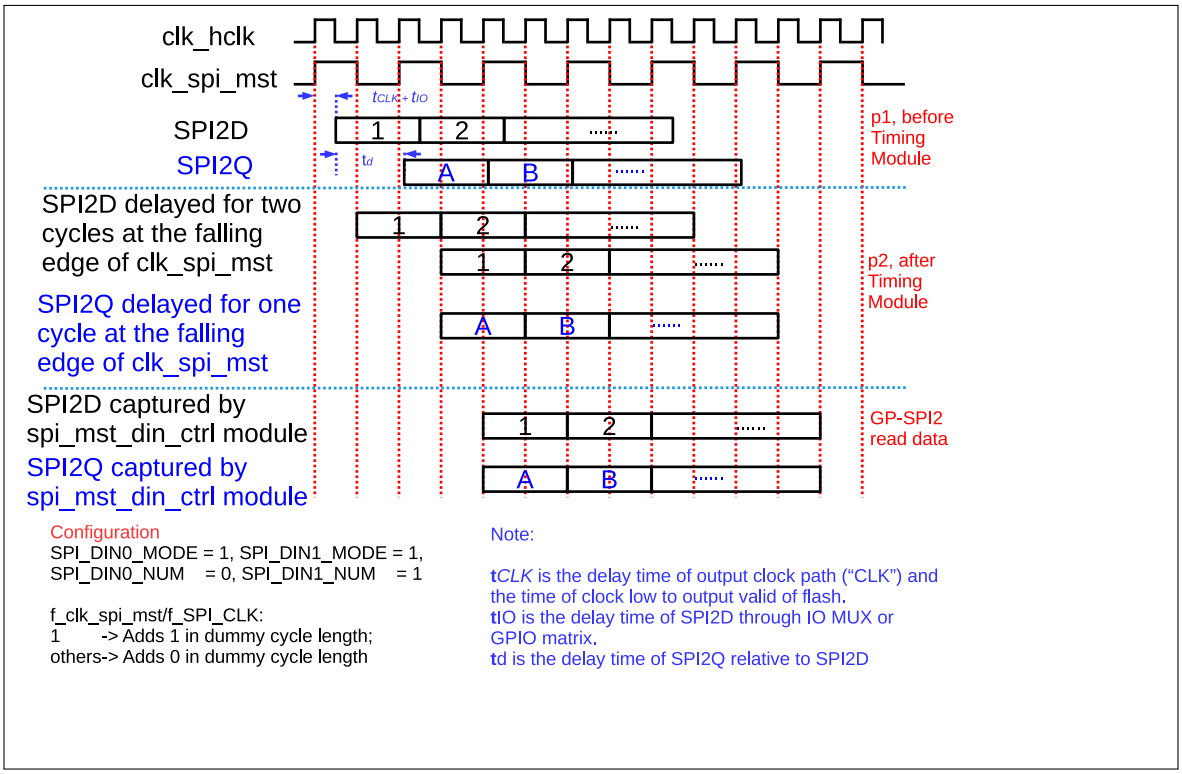


图 46.8-2. GP-SPI2 主机模式下时序补偿

图 46.8-2 中，"p1" 为时序模块输入数据的时间点；"p2" 为输出数据的时间点。由于 SPI2Q 输入数据并未与 SPI2D 输入数据对齐，如果没有时间补偿的话，GP-SPI2 读取数据将会出错。

为了正确读取数据，需要进行如下配置。其中，假设 $f_{clk_spi_mst}$ 等于 f_{SPI_CLK}：

- 将 SPI2D 在 clk_spi_mst 下降沿延迟两个时钟周期
- 将 SPI2Q 在 clk_spi_mst 下降沿延迟一个时钟周期
- 增加一个额外的 DUMMY 周期

GP-SPI2 用作从机时，如果寄存器 SPI_SLAVE_REG 中 SPI_RSCK_DATA_OUT 置 1，则在锁存沿发送输出数据，即提前半个 SPI 时钟周期。上述功能可用于从机模式时序补偿。

46.9 LP-SPI 唤醒

LP-SPI 用作从机时支持唤醒功能。当 LP-SPI 处于睡眠状态时（配置 LP_SPI_SLEEP_EN 为 1），可以通过 LP_SPI_SLV_WK_MODE_SEL 选择唤醒方式产生 wake_up 信号，触发 PMU 唤醒芯片。具体的唤醒流程，请参考章节 15 低功耗管理。

- LP_SPI_SLV_WK_MODE_SEL = 0 时，LP-SPI 从机监测到起始位后，唤醒芯片。
- LP_SPI_SLV_WK_MODE_SEL = 1 时，LP-SPI 从机监测收到特定字符序列后，唤醒芯片。
  - 通过 LP_SPI_SLEEP_WK_DATA_SEL 选择从 CMD 阶段或 DIN 阶段开始监测特定字符
  - 通过 LP_SPI_SLEEP_DIS_RXFIFO_WR_EN 选择在唤醒前是否缓存数据
  - 通过 LP_SPI_SLV_WK_CHAR0~LP_SPI_SLV_WK_CHAR4 自定义唤醒字符。LP-SPI 从机在检测到由 CHAR0/CHAR1/CHAR2/CHAR3/CHAR4 组成的字符序列流之后唤醒芯片。
  - 通过 LP_SPI_SLV_WK_CHAR_NUM 和 LP_SPI_SLV_WK_CHAR_MASK 来设置不同的字符序列，如表46.9-1 所示。同样也支持模糊检测，例如 LP_SPI_WK_SLV_CHAR_NUM = 5，且 LP_SPI_SLV_WK_MASK = 0x15（二进制 10101）时，检测字符序列为 */CHAR1/*/CHAR3/*。

表 46.9-1. LP-SPI 从机唤醒模式配置

LP_SPI_WK_SLV_CHAR_NUM	LP_SPI_SLV_WK_MASK	检测字符序列
1	0xF	CHAR4
2	0x7	CHAR3/CHAR4
3	0x3	CHAR2/CHAR3/CHAR4
4	0x1	CHAR1/CHAR2/CHAR3/CHAR4
5	0x0	CHAR0/CHAR1/CHAR2/CHAR3/CHAR4

46.10 GP-SPI2 与 GP-SPI3 和 LP-SPI 之间的差异

46.10.1 功能差异

表 46.10-1 列出了 GP-SPI2、GP-SPI3 和 LP-SPI 之间的功能差异。

表 46.10-1. GP-SPI2 与 GP-SPI3 和 LP-SPI 的功能差异

功能差异	GP-SPI2	GP-SPI3	LP-SPI
用作主机时，CMD、ADDR、DOUT 或 DIN 阶段支持的数据模式	支持 1/2/4/8-bit 模式	支持 1/2/4-bit 模式	支持 1-bit 模式
用作从机时，CMD、ADDR、DOUT 或 DIN 阶段支持的数据模式	支持 1/2/4-bit 模式	支持 1/2/4-bit 模式	支持 1-bit 模式
DMA 控制的相关传输	支持所有 DMA 控制的相关传输	不支持 DMA 控制的分段配置传输，支持其他 DMA 控制的传输	不支持任何 DMA 控制的传输

见下页

表 46.10-1 – 接上页

功能差异	GP-SPI2	GP-SPI3	LP-SPI
I/O 映射到管脚的方式	可通过 HP GPIO 交换矩阵或 HP IO MUX 映射到管脚	仅可通过 HP GPIO 交换矩阵映射到管脚	可通过 LP GPIO 交换矩阵或 LP IO MUX 映射到管脚
用作主机时提供的 CS 数量	6	3	1
非字节对齐传输功能	支持	支持	不支持
DTR 功能	支持	不支持	不支持
用作从机时的唤醒功能	不支持	不支持	支持
时序补偿	支持	支持	不支持

46.10.2 寄存器差异

表 46.10-2 列出了 GP-SPI2、GP-SPI3 和 LP-SPI 之间存在差异的寄存器及字段。

表 46.10-2. 寄存器差异

差异寄存器	差异字段	GP-SPI2	GP-SPI3	LP-SPI
CMD_REG	CONF_BITLEN	Y	—	—
USER_REG	OPI_MODE	Y	Y	—
	FWRITE_DUAL	Y	Y	—
	FWRITE_DUAD	Y	Y	—
	FWRITE_OCT	Y	—	—
	USR_CONF_NEXT	Y	—	—
CTRL_REG	FADDR_DUAL	Y	Y	—
	FADDR_QUAD	Y	Y	—
	FADDR_OCT	Y	—	—
	FCMD_DUAL	Y	Y	—
	FCMD_QUAD	Y	Y	—
	FCMD_OCT	Y	—	—
	FREAD_DUAL	Y	Y	—
	FREAD_QUAD	Y	Y	—
	FREAD_OCT	Y	—	—
	HOLD_POL	Y	Y	—
	WP_POL	Y	Y	—
MISC_REG	CS1_DIS	Y	Y	—
	CS2_DIS	Y	Y	—
	CS3_DIS	Y	—	—
	CS4_DIS	Y	—	—
	CS5_DIS	Y	—	—
	CLK_DATA_DTR_EN	Y	—	—
	DATA_DTR_EN	Y	—	—
	ADDR_DTR_EN	Y	—	—
	DQS_IDLE_EDGE	Y	—	—
	QUAD_DIN_PIN_SWAP	Y	—	—

见下页

表 46.10-2 – 接上页

差异寄存器	差异字段	GP-SPI2	GP-SPI3	LP-SPI
DMA_CONF_REG	DMA_OUTFIFO_EMPTY	Y	Y	—
	DMA_INFIFO_FULL	Y	Y	—
	DMA_SLV_SEG_TRANS_EN	Y	Y	—
	SLV_RX_SEG_TRANS_CLR_EN	Y	Y	—
	SLV_TX_SEG_TRANS_CLR_EN	Y	Y	—
	RX_EOF_EN	Y	Y	—
	DMA_RX_ENA	Y	Y	—
	DMA_TX_ENA	Y	Y	—
	DMA_AFIFO_RST	Y	Y	—
SLEEP_CONF0_REG	SLV_WK_CHAR0	—	—	Y
	SLV_WK_CHAR_NUM	—	—	Y
	SLV_WK_CHAR_MASK	—	—	Y
	SLV_WK_MODE_SEL	—	—	Y
	SLEEP_EN	—	—	Y
	SLEEP_DIS_RXFIFO_WR_EN	—	—	Y
	SLEEP_WK_DATA_SEL	—	—	Y
SLEEP_CONF1_REG	SLV_WK_CHAR1	—	—	Y
	SLV_WK_CHAR2	—	—	Y
	SLV_WK_CHAR3	—	—	Y
	SLV_WK_CHAR4	—	—	Y
SLAVE_REG	SLV_RDDMA_BITLEN_EN	Y	Y	—
	SLV_WRDMA_BITLEN_EN	Y	Y	—
	SLV_LAST_BYTE_STRB	Y	Y	—
	DMA_SEG_MAGIC_VALUE	Y	—	—
	USR_CONF	Y	—	—
	MST_FD_WAIT_DMA_TX_DATA	Y	Y	—
DIN_MODE_REG	DIN4_MODE	Y	—	—
	DIN5_MODE	Y	—	—
	DIN6_MODE	Y	—	—
	DIN7_MODE	Y	—	—
DIN_NUM_REG	DIN4_NUM	Y	—	—
	DIN5_NUM	Y	—	—
	DIN6_NUM	Y	—	—
	DIN7_NUM	Y	—	—
DOUT_MODE_REG	DOUT4_MODE	Y	—	—
	DOUT5_MODE	Y	—	—
	DOUT6_MODE	Y	—	—
	DOUT7_MODE	Y	—	—
	D_DQS_MODE	Y	—	—

见下页

表 46.10-2 – 接上页

差异寄存器	差异字段	GP-SPI2	GP-SPI3	LP-SPI
DMA_INT_ENA_REG	DMA_INFIFO_FULL_ERR_INT_ENA	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_ENA	Y	Y	—
	SLV_EX_QPI_INT_ENA	Y	Y	—
	SLV_EN_QPI_INT_ENA	Y	Y	—
	SLV_CMD7_INT_ENA	Y	Y	—
	SLV_CMD8_INT_ENA	Y	Y	—
	SLV_CMD9_INT_ENA	Y	Y	—
	SLV_CMDA_INT_ENA	Y	Y	—
	SLV_RD_DMA_DONE_INT_ENA	Y	Y	—
	SLV_WR_DMA_DONE_INT_ENA	Y	Y	—
	LP_SPI_WAKEUP_INT_ENA	—	—	Y
	DMA_SEG_TRANS_DONE_INT_ENA	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_ENA	Y	—	—
DMA_INT_CLR_REG	DMA_INFIFO_FULL_ERR_INT_CLR	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_CLR	Y	Y	—
	SLV_EX_QPI_INT_CLR	Y	Y	—
	SLV_EN_QPI_INT_CLR	Y	Y	—
	SLV_CMD7_INT_CLR	Y	Y	—
	SLV_CMD8_INT_CLR	Y	Y	—
	SLV_CMD9_INT_CLR	Y	Y	—
	SLV_CMDA_INT_CLR	Y	Y	—
	SLV_RD_DMA_DONE_INT_CLR	Y	Y	—
	SLV_WR_DMA_DONE_INT_CLR	Y	Y	—
	LP_SPI_WAKEUP_INT_CLR	—	—	Y
	DMA_SEG_TRANS_DONE_INT_CLR	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_CLR	Y	—	—
DMA_INT_RAW_REG	DMA_INFIFO_FULL_ERR_INT_RAW	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_RAW	Y	Y	—
	SLV_EX_QPI_INT_RAW	Y	Y	—
	SLV_EN_QPI_INT_RAW	Y	Y	—
	SLV_CMD7_INT_RAW	Y	Y	—
	SLV_CMD8_INT_RAW	Y	Y	—
	SLV_CMD9_INT_RAW	Y	Y	—
	SLV_CMDA_INT_RAW	Y	Y	—
	SLV_RD_DMA_DONE_INT_RAW	Y	Y	—
	SLV_WR_DMA_DONE_INT_RAW	Y	Y	—
	LP_SPI_WAKEUP_INT_RAW	—	—	Y
	DMA_SEG_TRANS_DONE_INT_RAW	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_RAW	Y	—	—

见下页

表 46.10-2 – 接上页

差异寄存器	差异字段	GP-SPI2	GP-SPI3	LP-SPI
DMA_INT_ST_REG	DMA_INFIFO_FULL_ERR_INT_ST	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_ST	Y	Y	—
	SLV_EX_QPI_INT_ST	Y	Y	—
	SLV_EN_QPI_INT_ST	Y	Y	—
	SLV_CMD7_INT_ST	Y	Y	—
	SLV_CMD8_INT_ST	Y	Y	—
	SLV_CMD9_INT_ST	Y	Y	—
	SLV_CMDA_INT_ST	Y	Y	—
	SLV_RD_DMA_DONE_INT_ST	Y	Y	—
	SLV_WR_DMA_DONE_INT_ST	Y	Y	—
	LP_SPI_WAKEUP_INT_ST	—	—	Y
	DMA_SEG_TRANS_DONE_INT_ST	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_ST	Y	—	—
DMA_INT_SET_REG	DMA_INFIFO_FULL_ERR_INT_SET	Y	Y	—
	DMA_OUTFIFO_EMPTY_ERR_INT_SET	Y	Y	—
	SLV_EX_QPI_INT_SET	Y	Y	—
	SLV_EN_QPI_INT_SET	Y	Y	—
	SLV_CMD7_INT_SET	Y	Y	—
	SLV_CMD8_INT_SET	Y	Y	—
	SLV_CMD9_INT_SET	Y	Y	—
	SLV_CMDA_INT_SET	Y	Y	—
	SLV_RD_DMA_DONE_INT_SET	Y	Y	—
	SLV_WR_DMA_DONE_INT_SET	Y	Y	—
	LP_SPI_WAKEUP_INT_SET	—	—	Y
	DMA_SEG_TRANS_DONE_INT_SET	Y	Y	—
	SPI_SEG_MAGIC_ERR_INT_SET	Y	—	—

46.10.3 中断差异

表 46.10-3 列出了 GP-SPI2、GP-SPI3 和 LP-SPI 支持的中断。

表 46.10-3. 中断差异

中断	GP-SPI2	GP-SPI3	LP-SPI
SPI_DMA_INFIFO_FULL_ERR_INT	Y	Y	—
SPI_DMA_OUTFIFO_EMPTY_ERR_INT	Y	Y	—
SPI_SLV_EX_QPI_INT	Y	Y	—
SPI_SLV_EN_QPI_INT	Y	Y	—
SPI_SLV_CMD7_INT	Y	Y	—
SPI_SLV_CMD8_INT	Y	Y	—
SPI_SLV_CMD9_INT	Y	Y	—
SPI_SLV_CMDA_INT	Y	Y	—

表 46.10-3. 中断差异

中断	GP-SPI2	GP-SPI3	LP-SPI
SPI_SLV_RD_DMA_DONE_INT	Y	Y	—
SPI_SLV_WR_DMA_DONE_INT	Y	Y	—
SPI_SLV_RD_BUF_DONE_INT	Y	Y	Y
SPI_SLV_WR_BUF_DONE_INT	Y	Y	Y
SPI_TRANS_DONE_INT	Y	Y	Y
SPI_DMA_SEG_TRANS_DONE_INT	Y	Y	—
SPI_SEG_MAGIC_ERR_INT	Y	—	—
SPI_MST_RX_AFIFO_WFULL_ERR_INT	Y	Y	Y
SPI_MST_TX_AFIFO_REMPTY_ERR_INT	Y	Y	Y
SPI_SLV_CMD_ERR_INT	Y	Y	Y
SPI_APP2_INT	Y	Y	Y
SPI_APP1_INT	Y	Y	Y
LP_SPI_WAKEUP_INT	—	—	Y

46.11 中断

46.11.1 中断描述

ESP32-P4 LP-SPI 和 GP-SPI 可以生成如下外部中断信号：

- LP-SPI: LP_SPI_INTR
- GP-SPI2: SPI_INTR_2
- GP-SPI3: SPI_INTR_3

LP_SPI_INTR 直接映射至 LP CPU（见章节 5 低功耗处理器），SPI_INTR_2 和 SPI_INTR_3 可通过中断矩阵映射至 HP CPU0/1。这些中断信号由 LP-SPI 或 GP-SPI 模块的内部中断源生成。表 46.11-1 列出了 GP-SPI 和 LP-SPI 的内部中断源、中断触发条件、以及生成的中断信号。

表 46.11-1. LP-SPI 和 GP-SPI 的内部中断源

内部中断源	触发条件	中断信号
SPI_DMA_INFIFO_FULL_ERR_INT	DMA RX FIFO 小于实际传输的数据长度	SPI_INTR_2/SPI_INTR_3
SPI_DMA_OUTFIFO_EMPTY_ERR_INT	DMA TX FIFO 小于实际传输的数据长度	SPI_INTR_2/SPI_INTR_3
SPI_SLV_EX_QPI_INT	GP-SPI 从机模式下，正确接收 Ex_QPI 命令，且 SPI 传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SLV_EN_QPI_INT	GP-SPI 从机模式下，正确接收 En_QPI 命令，且 SPI 传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SLV_CMD7_INT	GP-SPI 从机模式下，正确接收 CMD7 命令，且 SPI 传输结束	SPI_INTR_2/SPI_INTR_3

见下页

表 46.11-1 – 接上页

内部中断源	触发条件	中断信号
SPI_SLV_CMD8_INT	GP-SPI 从机模式下，正确接收 CMD8 命令，且 SPI 传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SLV_CMD9_INT	GP-SPI 从机模式下，正确接收 CMD9 命令，且 SPI 传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SLV_CMDA_INT	GP-SPI 从机模式下，正确接收 CMDA 命令，且 SPI 传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SLV_RD_DMA_DONE_INT	从机模式下，Rd_DMA 传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SLV_WR_DMA_DONE_INT	从机模式下，Wr_DMA 传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SLV_RD_BUF_DONE_INT	从机模式下，Rd_BUF 传输结束	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
SPI_SLV_WR_BUF_DONE_INT	从机模式下，Wr_BUF 传输结束	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
SPI_TRANS_DONE_INT	主从机模式下，SPI 总线传输结束	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
SPI_DMA_SEG_TRANS_DONE _INT	GP-SPI 从机连续传输模式下，END_SEG_TRANS 传输结束。主机模式下，分段配置传输结束	SPI_INTR_2/SPI_INTR_3
SPI_SEG_MAGIC_ERR_INT	在主机分段配置传输模式下，CONF buffer 中的 Magic 值有误	SPI_INTR_2
SPI_MST_RX_AFIFO_WFULL _ERR_INT	GP-SPI 主机模式下，发生 RX AFIFO write-full 错误	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
SPI_MST_TX_AFIFO_REMPTY _ERR_INT	GP-SPI 主机模式下，发生 TX AFIFO read-empty 错误	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
SPI_SLV_CMD_ERR_INT	GP-SPI 从机模式下，接收到 GP-SPI 不支持的命令值	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
SPI_APP2_INT	置位 SPI_APP2_INT_SET（仅用于用户自定义的功能）	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
SPI_APP1_INT	置位 SPI_APP1_INT_SET（仅用于用户自定义的功能）	LP_SPI_INTR/ SPI_INTR_2/SPI_INTR_3
LP_SPI_WAKEUP_INT	LP_SPI 监测到配置的字符，产生唤醒信号	LP_SPI_INTR

其中 LP-SPI 是 GP-SPI 的精简版本，仅包含部分 GP-SPI 中断源。每个中断源都有一组通用的配置寄存器，详见章节寄存器列表中的中断寄存器。

46.11.2 主机中断和从机中断（以 GP-SPI 为例）

表 46.11-2 和表 46.11-3 分别列出了 GP-SPI 在主机模式下和从机模式下用到的中断。通过这两张表，可以查询不同工作模式下，需要关注的主要中断信息。置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_*_INT_ENA 位，使能相应中断，并等待中断。传输结束时，将触发相关中断。注意，在下次传输之前，需软件清除中断。

表 46.11-2. GP-SPI 主机中断表

传输类型	通信模式	控制方式	中断
单次传输	全双工	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	半双工 MOSI 模式	DMA	SPI_TRANS_DONE_INT
		CPU	SPI_TRANS_DONE_INT
	半双工 MISO 模式	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT
		CPU	SPI_TRANS_DONE_INT
分段配置传输	全双工	DMA	SPI_DMA_SEG_TRANS_DONE_INT [3]
		CPU	不支持
	半双工 MOSI 模式	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	不支持
	半双工 MISO 模式	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	不支持

[1] 如果触发了 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断，则表示 GP-SPI 的所有 RX 数据已保存至 RX buffer，且所有 TX 数据已发送至从机。

[2] CS 拉高，则将触发 SPI_TRANS_DONE_INT 中断，表明主机与从机已完成 SPI_W0_REG~SPI_W15_REG 的数据交换。

[3] 如果触发了 SPI_DMA_SEG_TRANS_DONE_INT 中断，则表明整个分段配置传输，包括若干个传输事务，已完成。即 RX 数据已全部存入 RX buffer 且所有 TX 数据已发送完毕。

表 46.11-3. GP-SPI 从机中断表

传输类型	通信模式	控制方式	中断
单次传输	全双工	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	半双工 MOSI 模式	DMA (Wr_DMA)	AXI_DMA_IN_SUC_EOF_CH*n*_INT [3]
		CPU (Wr_BUF)	SPI_TRANS_DONE_INT [4]
	半双工 MISO 模式	DMA (Rd_DMA)	SPI_TRANS_DONE_INT [5]
		CPU (Rd_BUF)	SPI_TRANS_DONE_INT [6]
从机连续传输	全双工	DMA	AXI_DMA_IN_SUC_EOF_CH*n*_INT [7]
		CPU	不支持 [8]
	半双工 MOSI 模式	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [9]
		CPU (Wr_BUF)	不支持 [10]
	半双工 MISO 模式	DMA (Rd_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [11]
		CPU (Rd_BUF)	不支持 [12]

[1] 如果触发了 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断，则表示所有 RX 数据已保存至 RX buffer，且所有 TX 数据已发送至从机。
[2] CS 拉高，则将触发 SPI_TRANS_DONE_INT 中断，表明主机与从机已完成 SPI_W0_REG~SPI_W15_REG 的数据交换。
[3] 触发 SPI_SLV_WR_DMA_DONE_INT 中断仅表示 SPI 总线上的数据传输已完成，但并不能保证所有入栈数据已存至 RX buffer。因此，推荐使用 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。
[4] 或等待 SPI_SLV_WR_BUF_DONE_INT 中断。
[5] 或等待 SPI_SLV_RD_DMA_DONE_INT 中断。
[6] 或等待 SPI_SLV_RD_BUF_DONE_INT 中断。
[7] 传输开始前，从机应在 SPI_MS_DATA_BITLEN 中设置读数据的总长度。并在中断程序结束前，置位 SPI_RX_EOF_EN。
[8] 主机和从机需定义连续传输结束的方式，比如配置 GPIO 用作中断等。
[9] 主机发送 End_SEG_TRAN 结束连续传输，或从机在 SPI_MS_DATA_BITLEN 中配置总的读数据长度，然后等待 AXI_DMA_IN_SUC_EOF_CH*n*_INT 中断。
[10] 半双工 Wr_BUF 单次传输也可用于 DMA 控制的从机连续传输中。
[11] 主机发送 End_SEG_TRAN 结束从机连续传输。
[12] 半双工 Rd_BUF 单次传输也可用于 DMA 控制的从机连续传输中。

46.12 寄存器列表

46.12.1 GP-SPI2 寄存器列表

本小节的所有地址均为相对于 GP-SPI2 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	地址	访问
自定义控制命令			
SPI_CMD_REG	命令控制寄存器	0x0000	varies
SPI_ADDR_REG	地址值寄存器	0x0004	R/W
SPI_USER_REG	SPI 用户控制寄存器	0x0010	varies
SPI_USER1_REG	SPI 用户控制寄存器 1	0x0014	R/W
SPI_USER2_REG	SPI 用户控制寄存器 2	0x0018	R/W
控制和配置寄存器			
SPI_CTRL_REG	SPI 控制寄存器	0x0008	R/W
SPI_MS_DLEN_REG	SPI 数据位长控制寄存器	0x001C	R/W
SPI_MISC_REG	SPI MISC 寄存器	0x0020	R/W
SPI_DMA_CONF_REG	SPI DMA 控制寄存器	0x0030	varies
SPI_SLAVE_REG	SPI 从机控制寄存器	0x00E0	varies
SPI_SLAVE1_REG	SPI 从机控制寄存器 1	0x00E4	R/W/SS
时钟控制寄存器			
SPI_CLOCK_REG	SPI 时钟控制寄存器	0x000C	R/W
SPI_CLK_GATE_REG	SPI 模块时钟和寄存器时钟控制寄存器	0x00E8	R/W

名称	描述	地址	访问
时序寄存器			
SPI_DIN_MODE_REG	SPI 输入延迟模式配置寄存器	0x0024	R/W
SPI_DIN_NUM_REG	SPI 输入延迟周期配置寄存器	0x0028	R/W
SPI_DOUT_MODE_REG	SPI 输出延迟模式配置寄存器	0x002C	R/W
中断寄存器			
SPI_DMA_INT_ENA_REG	SPI 中断使能寄存器	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI 中断清除寄存器	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI 原始中断寄存器	0x003C	R/WTC/SS
SPI_DMA_INT_ST_REG	SPI 中断状态寄存器	0x0040	RO
SPI_DMA_INT_SET_REG	SPI 中断软件置位寄存器	0x0044	WT
CPU 数据 buffer			
SPI_W0_REG	SPI CPU 控制的 buffer 0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU 控制的 buffer 1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU 控制的 buffer 2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU 控制的 buffer 3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU 控制的 buffer 4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU 控制的 buffer 5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU 控制的 buffer 6	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU 控制的 buffer 7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU 控制的 buffer 8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU 控制的 buffer 9	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU 控制的 buffer 10	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU 控制的 buffer 11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU 控制的 buffer 12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU 控制的 buffer 13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU 控制的 buffer 14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU 控制的 buffer 15	0x00D4	R/W/SS
版本寄存器			
SPI_DATE_REG	版本控制寄存器	0x00F0	R/W

46.12.2 GP-SPI3 寄存器列表

本小节的所有地址均为相对于 GP-SPI3 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	地址	访问
自定义控制寄存器			
SPI_CMD_REG	命令控制寄存器	0x0000	varies
SPI_ADDR_REG	地址值寄存器	0x0004	R/W
SPI_USER_REG	SPI 用户控制寄存器	0x0010	varies
SPI_USER1_REG	SPI 用户控制寄存器 1	0x0014	R/W
SPI_USER2_REG	SPI 用户控制寄存器 2	0x0018	R/W

名称	描述	地址	访问
控制和配置寄存器			
SPI_CTRL_REG	SPI 控制寄存器	0x0008	R/W
SPI_MS_DLEN_REG	SPI 数据位长控制寄存器	0x001C	R/W
SPI_MISC_REG	SPI MISC 寄存器	0x0020	R/W
SPI_DMA_CONF_REG	SPI DMA 控制寄存器	0x0030	varies
SPI_SLAVE_REG	SPI 从机控制寄存器	0x00E0	varies
SPI_SLAVE1_REG	SPI 从机控制寄存器 1	0x00E4	R/W/SS
时钟控制寄存器			
SPI_CLOCK_REG	SPI 时钟控制寄存器	0x000C	R/W
SPI_CLK_GATE_REG	SPI 模块时钟和寄存器时钟控制寄存器	0x00E8	R/W
时序寄存器			
SPI_DIN_MODE_REG	SPI 输入延迟模式配置寄存器	0x0024	R/W
SPI_DIN_NUM_REG	SPI 输入延迟周期配置寄存器	0x0028	R/W
SPI_DOUT_MODE_REG	SPI 输出延迟模式配置寄存器	0x002C	R/W
中断寄存器			
SPI_DMA_INT_ENA_REG	SPI 中断使能寄存器	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI 中断清除寄存器	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI 原始中断寄存器	0x003C	R/WTC/SS
SPI_DMA_INT_ST_REG	SPI 中断状态寄存器	0x0040	RO
SPI_DMA_INT_SET_REG	SPI 中断软件置位寄存器	0x0044	WT
CPU 数据 buffer			
SPI_W0_REG	SPI CPU 控制的 buffer 0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU 控制的 buffer 1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU 控制的 buffer 2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU 控制的 buffer 3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU 控制的 buffer 4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU 控制的 buffer 5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU 控制的 buffer 6	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU 控制的 buffer 7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU 控制的 buffer 8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU 控制的 buffer 9	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU 控制的 buffer 10	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU 控制的 buffer 11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU 控制的 buffer 12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU 控制的 buffer 13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU 控制的 buffer 14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU 控制的 buffer 15	0x00D4	R/W/SS
版本寄存器			
SPI_DATE_REG	版本控制寄存器	0x00F0	R/W

46.12.3 LP-SPI 寄存器列表

本小节的所有地址均为相对于 LP-SPI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
自定义控制寄存器			
LP_SPI_CMD_REG	命令控制寄存器	0x0000	varies
LP_SPI_ADDR_REG	地址值寄存器	0x0004	R/W
LP_SPI_USER_REG	SPI 用户控制寄存器	0x0010	R/W
LP_SPI_USER1_REG	SPI 用户控制寄存器 1	0x0014	R/W
LP_SPI_USER2_REG	SPI 用户控制寄存器 2	0x0018	R/W
控制和配置寄存器			
LP_SPI_CTRL_REG	SPI 控制寄存器	0x0008	R/W
LP_SPI_MS_DLEN_REG	SPI 数据位长控制寄存器	0x001C	R/W
LP_SPI_MISC_REG	SPI MISC 寄存器	0x0020	R/W
LP_SPI_DMA_CONF_REG	SPI DMA 控制寄存器	0x0030	WT
LP_SPI_SLEEP_CONF0_REG	LP-SPI 睡眠配置寄存器 0	0x0044	R/W
LP_SPI_SLEEP_CONF1_REG	LP-SPI 睡眠配置寄存器 1	0x0048	R/W
LP_SPI_SLAVE_REG	SPI 从机控制寄存器	0x00E0	varies
LP_SPI_SLAVE1_REG	SPI 从机控制寄存器 1	0x00E4	R/W/SS
时钟控制寄存器			
LP_SPI_CLOCK_REG	SPI 时钟控制寄存器	0x000C	R/W
LP_SPI_CLK_GATE_REG	SPI 模块时钟和寄存器时钟控制寄存器	0x00E8	R/W
时序寄存器			
LP_SPI_DIN_MODE_REG	SPI 输入延迟模式配置寄存器	0x0024	R/W
LP_SPI_DIN_NUM_REG	SPI 输入延迟周期配置寄存器	0x0028	R/W
LP_SPI_DOUT_MODE_REG	SPI 输出延迟模式配置寄存器	0x002C	R/W
中断寄存器			
LP_SPI_DMA_INT_ENA_REG	SPI 中断使能寄存器	0x0034	R/W
LP_SPI_DMA_INT_CLR_REG	SPI 中断清除寄存器	0x0038	WT
LP_SPI_DMA_INT_RAW_REG	SPI 原始中断寄存器	0x003C	R/WTC/SS
LP_SPI_DMA_INT_ST_REG	SPI 中断状态寄存器	0x0040	RO
LP_SPI_DMA_INT_SET_REG	SPI 中断软件置位寄存器	0x004C	WT
CPU 数据 buffer			
LP_SPI_W0_REG	SPI CPU 控制的 buffer 0	0x0098	R/W/SS
LP_SPI_W1_REG	SPI CPU 控制的 buffer 1	0x009C	R/W/SS
LP_SPI_W2_REG	SPI CPU 控制的 buffer 2	0x00A0	R/W/SS
LP_SPI_W3_REG	SPI CPU 控制的 buffer 3	0x00A4	R/W/SS
LP_SPI_W4_REG	SPI CPU 控制的 buffer 4	0x00A8	R/W/SS
LP_SPI_W5_REG	SPI CPU 控制的 buffer 5	0x00AC	R/W/SS
LP_SPI_W6_REG	SPI CPU 控制的 buffer 6	0x00B0	R/W/SS
LP_SPI_W7_REG	SPI CPU 控制的 buffer 7	0x00B4	R/W/SS
LP_SPI_W8_REG	SPI CPU 控制的 buffer 8	0x00B8	R/W/SS
LP_SPI_W9_REG	SPI CPU 控制的 buffer 9	0x00BC	R/W/SS
LP_SPI_W10_REG	SPI CPU 控制的 buffer 10	0x00C0	R/W/SS
LP_SPI_W11_REG	SPI CPU 控制的 buffer 11	0x00C4	R/W/SS
LP_SPI_W12_REG	SPI CPU 控制的 buffer 12	0x00C8	R/W/SS

名称	描述	地址	访问
LP_SPI_W13_REG	SPI CPU 控制的 buffer 13	0x00CC	R/W/SS
LP_SPI_W14_REG	SPI CPU 控制的 buffer 14	0x00D0	R/W/SS
LP_SPI_W15_REG	SPI CPU 控制的 buffer 15	0x00D4	R/W/SS
版本寄存器			
LP_SPI_DATE_REG	版本控制寄存器	0x00F0	R/W

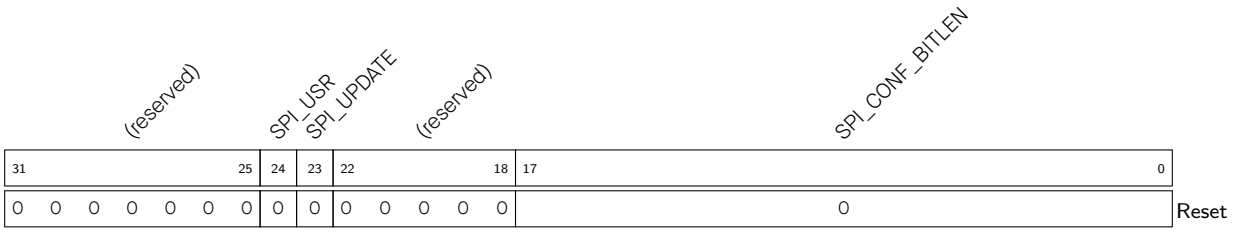
46.13 寄存器

46.13.1 GP-SPI2 寄存器

本小节的所有地址均为相对于 GP-SPI2 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 46.1. SPI_CMD_REG (0x0000)



SPI_CONF_BITLEN 配置 SPI CONF 阶段的 SPI_CLK 周期。

单位：SPI_CLK 时钟周期。

可在 CONF 阶段配置。

(R/W)

SPI_UPDATE 配置是否将 SPI 寄存器从 APB 时钟域同步到 SPI 模块时钟域。

0：不同步

1：同步

仅用于 SPI 主机。

(WT)

SPI_USR 配置是否使能用户自定义命令。

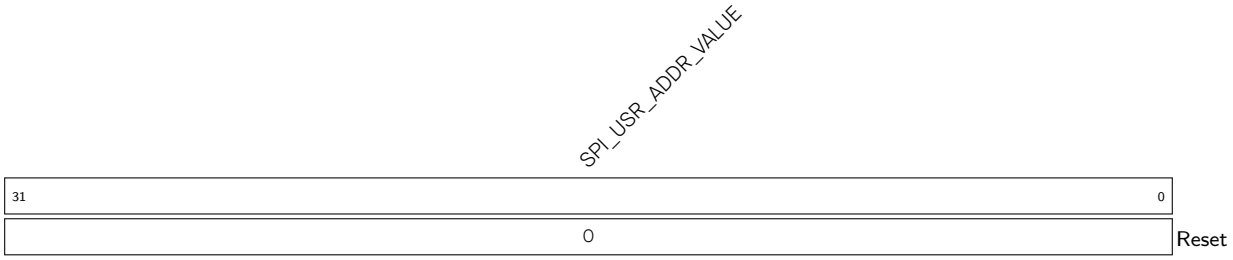
0：不使能

1：使能

置位此位将触发一次 SPI 操作。操作结束后此位被自动清零。CONF_buf 不可更改该配置。

(R/W/SC)

Register 46.2. SPI_ADDR_REG (0x0004)

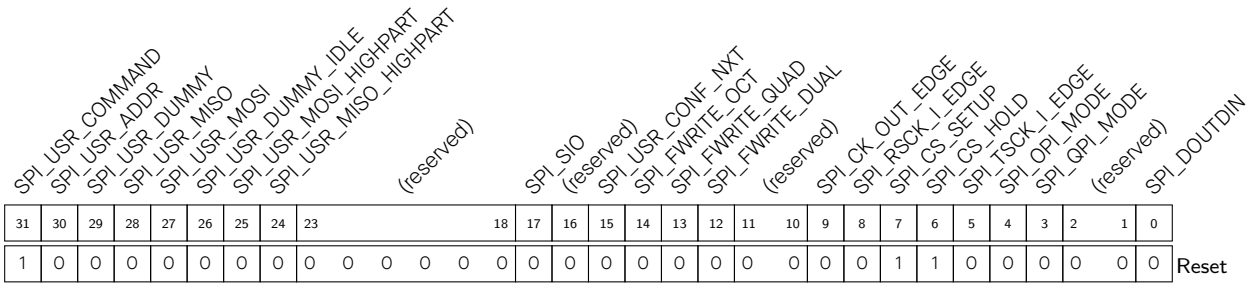


SPI_USR_ADDR_VALUE 配置从机地址。

可在 CONF 阶段配置。

(R/W)

Register 46.3. SPI_USER_REG (0x0010)



SPI_DOUTDIN 配置是否使能全双工通信。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_QPI_MODE 配置是否使能 QPI 模式。

0：不使能

1：使能

SPI 主机和从机均支持该配置。可在 CONF 阶段配置。

(R/W/SS/SC)

SPI_OPI_MODE 配置是否使能 OPI 模式。

0：不使能。SPI 控制器运行其他模式。

1：使能。SPI 控制器运行 OPI 模式，所有阶段均为 8 位模式。

该配置仅在 SPI 控制器用作主机时有效。可在 CONF 阶段配置。

(R/W)

SPI_TSCK_I_EDGE 配置 SPI 从机传输中是否更改 TSCK 极性。

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

SPI_CS_HOLD 配置在 SPI 处于完成 (DONE) 阶段时是否保持 SPI CS 拉低。

0：不保持拉低

1：保持拉低

可在 CONF 阶段配置。

(R/W)

SPI_CS_SETUP 配置在 SPI 处于准备 (PREP) 阶段时是否使能 SPI CS。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_RSCK_I_EDGE 配置 SPI 从机传输中是否更改 RSCK 极性。

0: RSCK = !SPI_CK_I

1: RSCK = SPI_CK_I

(R/W)

见下页

Register 46.3. SPI_USER_REG (0x0010)

接上页

SPI_CK_OUT_EDGE 配置 SPI 时钟模式，与 SPI_CK_IDLE_EDGE 搭配使用。

可在 CONF 阶段配置。更多信息见章节 46.7.4。

(R/W)

SPI_FWRITE_DUAL 配置在写操作 (DOUT) 阶段时读数据的方式是否为 2-bit 方式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FWRITE_QUAD 配置在写操作 (DOUT) 阶段时读数据的方式是否为 4-bit 方式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FWRITE_OCT 配置在写操作 (DOUT) 阶段时读数据的方式是否为 8-bit 方式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_CONF_NXT 配置是否使能下一次传输事务的 CONF 阶段。

0：当前传输事务结束后，本次分段配置传输结束。或者，当前的传输模式不是分段配置传输。

1：本次分段配置传输继续进行，开始下一次传输事务。

可在 CONF 阶段配置。

(R/W)

SPI_SIO 配置是否使能 3 线半双工通信，其中 MOSI 和 MISO 信号共享一个管脚。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_MISO_HIGHPART 配置在读数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.3. SPI_USER_REG (0x0010)

接上页

SPI_USR_MOSI_HIGHPART 配置在写数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_DUMMY_IDLE 配置是否在 DUMMY 阶段禁用 SPI 时钟。

0：不禁用

1：禁用

可在 CONF 阶段配置。

(R/W)

SPI_USR_MOSI 配置是否使能一次操作的写数据 (DOUT) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_MISO 配置是否使能一次操作的读数据 (DIN) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_DUMMY 配置是否使能一次操作的 DUMMY 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_ADDR 配置是否使能一次操作的地址 (ADDR) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_COMMAND 配置是否使能一次操作的命令 (CMD) 阶段。

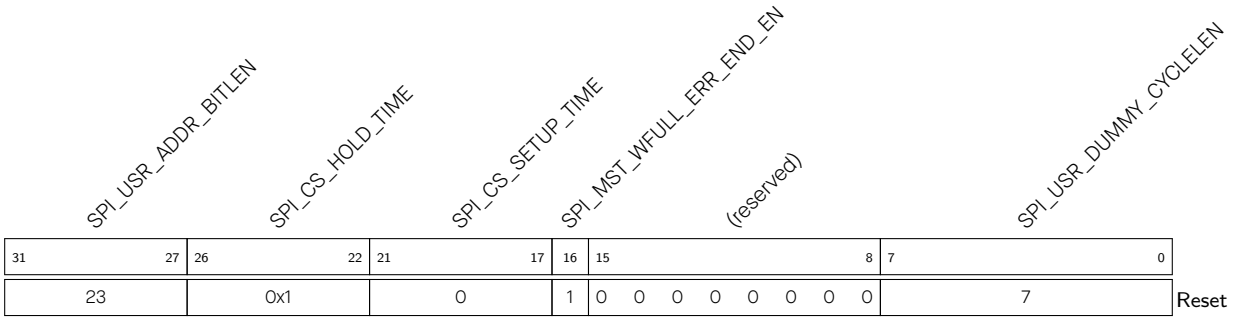
0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

Register 46.4. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN 配置 DUMMY 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。

可在 CONF 阶段配置。

(R/W)

SPI_MST_WFULL_ERR_END_EN 配置在主机全双工或半双工通信中，如果发生 RX AFIFO 满错误，是否终止 SPI 传输。

0：不终止

1：终止

(R/W)

SPI_CS_SETUP_TIME 配置准备 (PREP) 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。此字段与 SPI_CS_SETUP 搭配使用。

可在 CONF 阶段配置。

(R/W)

SPI_CS_HOLD_TIME 配置 CS 管脚的延迟周期。

单位：SPI_CLK 时钟周期。

此字段与 SPI_CS_HOLD 搭配使用。

可在 CONF 阶段配置。

(R/W)

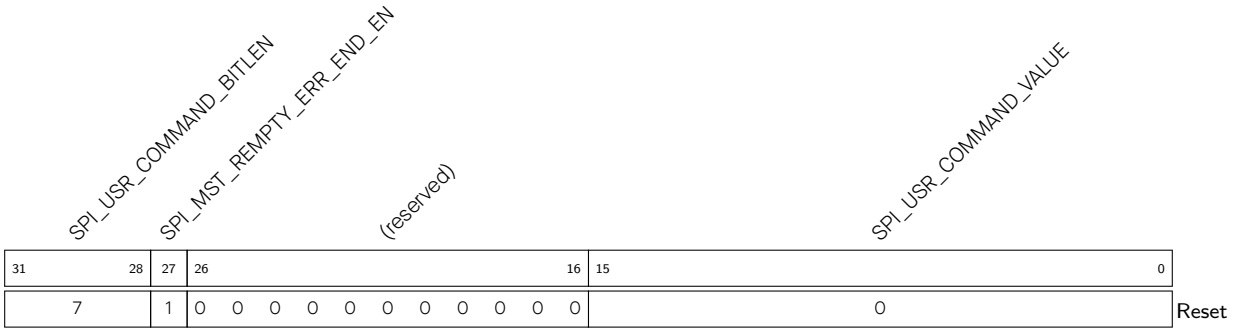
SPI_USR_ADDR_BITLEN 配置地址阶段的位长。

该值等于预期值 - 1。

可在 CONF 阶段配置。

(R/W)

Register 46.5. SPI_USER2_REG (0x0018)



SPI_USR_COMMAND_VALUE 配置命令值。

可在 CONF 阶段配置。

(R/W)

SPI_MST_REMPTY_ERR_END_EN 配置在主机全双工或半双工通信下，如果发生 TX AFIFO 空错误，是否终止 SPI 传输。

0：不终止

1：终止

(R/W)

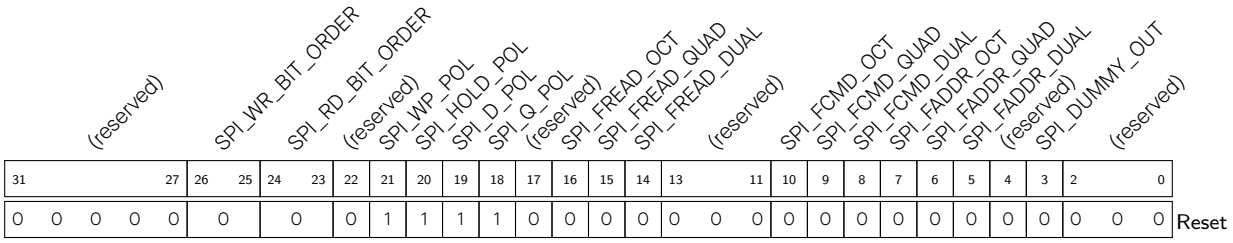
SPI_USR_COMMAND_BITLEN 配置命令阶段的位长。

该值等于预期值 - 1。

可在 CONF 阶段配置。

(R/W)

Register 46.6. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT 配置在 DUMMY 阶段是否输出 SPI2 总线信号。

0：不输出

1：输出

可在 CONF 阶段配置。

(R/W)

SPI_FADDR_DUAL 配置在地址 (ADDR) 阶段时是否使能 2-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FADDR_QUAD 配置在地址 (ADDR) 阶段时是否使能 4-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FADDR_OCT 配置在地址 (ADDR) 阶段时是否使能 8-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FCMD_DUAL 配置在命令 (CMD) 阶段时是否使能 2-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FCMD_QUAD 配置在命令 (CMD) 阶段时是否使能 4-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.6. SPI_CTRL_REG (0x0008)

接上页

SPI_FCMD_OCT 配置在命令 (CMD) 阶段时是否使能 8-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FREAD_DUAL 配置在读数据 (DIN) 阶段是否使能 2-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FREAD_QUAD 配置在读数据 (DIN) 阶段是否使能 4-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FREAD_OCT 配置在读数据 (DIN) 阶段是否使能 8-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_Q_POL 配置 MISO 的极性。

0：低

1：高

可在 CONF 阶段配置。

(R/W)

SPI_D_POL 配置 MOSI 的极性。

0：低

1：高

可在 CONF 阶段配置。

(R/W)

SPI_HOLD_POL 配置在 SPI 空闲状态下 SPI_HOLD 的输出值。

0：输出低电平

1：输出高电平

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.6. SPI_CTRL_REG (0x0008)

接上页

SPI_WP_POL 配置在 SPI 空闲状态下 WP 信号的输出值。

0：输出低电平

1：输出高电平

可在 CONF 阶段配置。

(R/W)

SPI_RD_BIT_ORDER 配置读数据 (MISO) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

2：先读高有效位

3：先读低有效位

可在 CONF 阶段配置，详细配置值参考表 46.5-6。

(R/W)

SPI_WR_BIT_ORDER 配置命令 (CMD)、地址 (ADDR) 和写数据 (MOSI) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

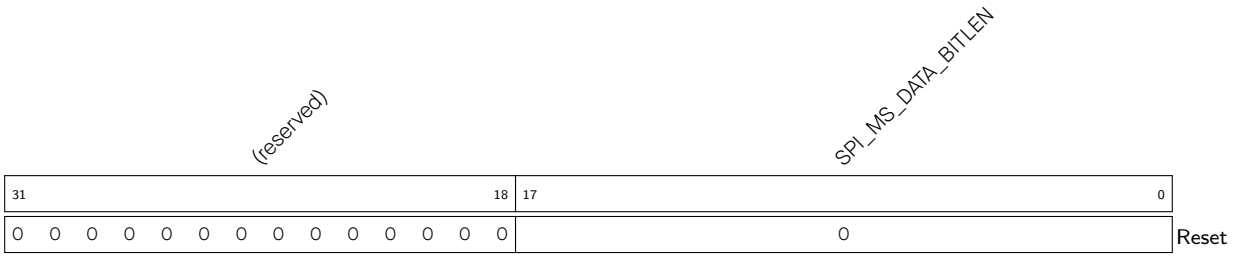
2：先读高有效位

3：先读低有效位

可在 CONF 阶段配置，详细配置值参考表 46.5-6。

(R/W)

Register 46.7. SPI_MS_DLEN_REG (0x001C)



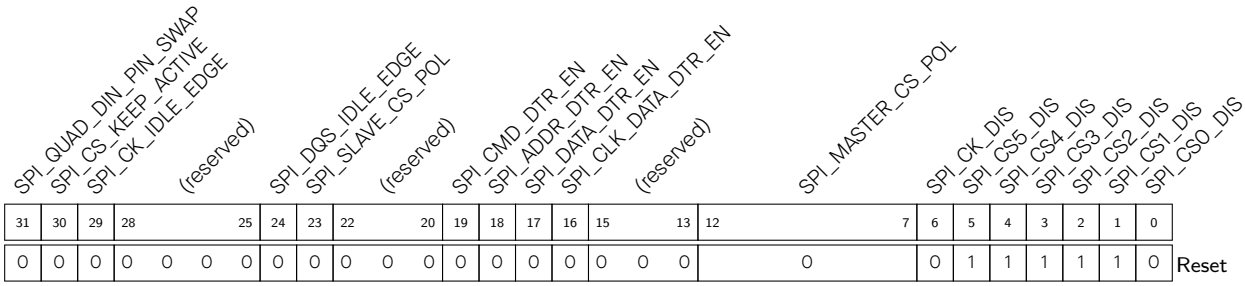
SPI_MS_DATA_BITLEN 配置 DMA 控制或 CPU 控制的主机传输的数据位长。或配置 DMA 控制的从机传输中接收数据的位长。

该值等于需要的位长 - 1。

可在 CONF 阶段配置。

(R/W)

Register 46.8. SPI_MISC_REG (0x0020)



SPI_CS0_DIS 配置是否禁用 SPI_CS0 管脚。

0：SPI_CS0 信号来自 SPI_CS0 管脚或输出至 SPI_CS0 管脚

1：禁用 SPI_CS0 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS1_DIS 配置是否禁用 SPI_CS1 管脚。

0：SPI_CS1 信号来自 SPI_CS1 管脚或输出至 SPI_CS1 管脚

1：禁用 SPI_CS1 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS2_DIS 配置是否禁用 SPI_CS2 管脚。

0：SPI_CS2 信号来自 SPI_CS2 管脚或输出至 SPI_CS2 管脚

1：禁用 SPI_CS2 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS3_DIS 配置是否禁用 SPI_CS3 管脚。

0：SPI_CS3 信号来自 SPI_CS3 管脚或输出至 SPI_CS3 管脚

1：禁用 SPI_CS3 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS4_DIS 配置是否禁用 SPI_CS4 管脚。

0：SPI_CS4 信号来自 SPI_CS4 管脚或输出至 SPI_CS4 管脚

1：禁用 SPI_CS4 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS5_DIS 配置是否禁用 SPI_CS5 管脚。

0：SPI_CS5 信号来自 SPI_CS5 管脚或输出至 SPI_CS5 管脚

1：禁用 SPI_CS5 管脚

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.8. SPI_MISC_REG (0x0020)

接上页

SPI_CK_DIS 配置是否禁用 SPI_CLK 输出信号。

0：不禁用

1：禁用

可在 CONF 阶段配置。

(R/W)

SPI_MASTER_CS_POL 配置主机 SPI_CS*n* (*n* = 0 ∼ 5) 的极性。

0：SPI_CS*n* 低电平有效

1：SPI_CS*n* 高电平有效

可在 CONF 阶段配置。

(R/W)

SPI_CLK_DATA_DTR_EN 配置在 SPI 主机模式下，SPI_CLK、DATA 和 SPI_DQS 是否使能 DTR 模式。

0：仅 SPI_DQS 使能 DTR 模式

1：SPI_CLK、DATA 和 SPI_DQS 均使能 DTR 模式

与 SPI_DATA_DTR_EN、SPI_ADDR_DTR_EN 和 SPI_CMD_DTR_EN 搭配使用。

(R/W)

SPI_DATA_DTR_EN 配置 SPI 在主机 1/2/4/8-bit 模式下，DIN 和 DOUT 阶段的时钟和数据是否使能 DTR 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_ADDR_DTR_EN 配置 SPI 在主机 1/2/4/8-bit 模式下，ADDR 阶段的时钟和数据是否使能 DTR 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_CMD_DTR_EN 配置 SPI 在主机 1/2/4/8-bit 模式下，CMD 阶段的时钟和数据是否使能 DTR 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.8. SPI_MISC_REG (0x0020)

接上页

SPI_SLAVE_CS_POL 配置是否反相 SPI 从机输入信号 CS 的极性。

0：保持不变

1：反相

可在 CONF 阶段配置。

(R/W)

SPI_DQS_IDLE_EDGE 配置 SPI_DQS 的默认值。

0：低电平

1：高电平

可在 CONF 阶段配置。

(R/W)

SPI_CK_IDLE_EDGE 配置 SPI_CLK 线在 GP-SPI2 空闲状态时的电平。

0：低电平

1：高电平

可在 CONF 阶段配置。

(R/W)

SPI_CS_KEEP_ACTIVE 配置 SPI_CS 是否保持低电平。

0：不保持低电平

1：保持低电平

可在 CONF 阶段配置。

(R/W)

SPI_QUAD_DIN_PIN_SWAP 配置是否交换 SPI Quad 输入管脚。

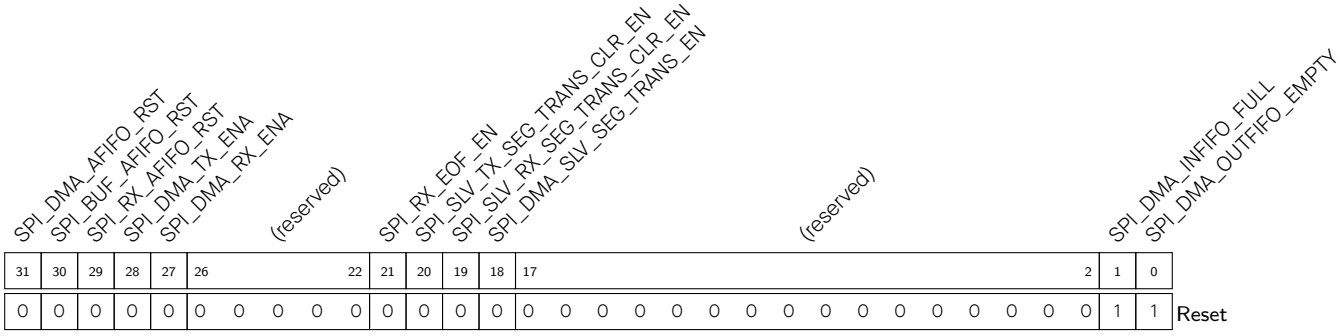
0：不交换

1：SPI2D 与 SPI2Q 交换，SPI2WP 与 SPI2HD 交换

可在 CONF 阶段配置。

(R/W)

Register 46.9. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY 表示 DMA TX FIFO 是否就绪。

0：DMA TX FIFO 已就绪，可以发送数据

1：DMA TX FIFO 尚未就绪，不能发送数据

(RO)

SPI_DMA_INFIFO_FULL 表示 DMA RX FIFO 是否就绪。

0：DMA RX FIFO 已就绪，可以接收数据

1：DMA RX FIFO 尚未就绪，不能接收数据

(RO)

SPI_DMA_SLV_SEG_TRANS_EN 配置是否使能半双工通信方式下，DMA 控制的从机连续传输。

0：不使能

1：使能

(R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA RX buffer 小于实际接收的数据长度：

1：后续 Wr_DMA 传输事务中传输的数据都不接收

0：当前 Wr_DMA 传输事务中传输的数据不接收，但在后续的 Wr_DMA 传输事务中：

- 如果 DMA RX buffer 长度不为 0，则后续 Wr_DMA 传输事务中传输的数据会被接收。
- 如果 DMA RX buffer 长度为 0，则后续 Wr_DMA 传输事务中传输的数据不会被接收。

(R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA TX buffer 小于实际发送的数据长度：

1：后续传输事务中传输的数据都不更新，即旧数据被重复发送

0：当前传输事务中传输的数据不更新，但在后续的传输事务中：

- 如果有新的数据填充到 DMA TX FIFO，则将发送新数据。
- 如果没有新的数据填充到 DMA TX FIFO，则没有新数据被发送。

(R/W)

SPI_RX_EOF_EN 配置 GDMA_IN_SUC_EOF_CH*n*_INT_RAW 中断的触发源。

0：在单次传输中，GDMA_IN_SUC_EOF_CH*n*_INT_RAW 由 SPI_TRANS_DONE_INT 事件置位；或在分段配置传输及从机连续传输中，由 SPI_DMA_SEG_TRANS_DONE_INT 事件置位。

1：在 DMA 控制的数据传输过程中，如果 DMA 传输的数据比特数等于 (SPI_MS_DATA_BITLEN + 1)，则硬件会置位 GDMA_IN_SUC_EOF_CH*n*_INT_RAW。

(R/W)

见下页

Register 46.9. SPI_DMA_CONF_REG (0x0030)

接上页

SPI_DMA_RX_ENA 配置是否使能 DMA 控制的接收数据模式。

0：不使能

1：使能

(R/W)

SPI_DMA_TX_ENA 配置是否使能 DMA 控制的发送数据模式。

0：不使能

1：使能

(R/W)

SPI_RX_AFIFO_RST 配置是否复位 spi_rx_afifo。spi_rx_afifo 将在 SPI 主机和从机传输中用于接收数据，见图 46.5-3 和图 46.5-4。

0：不复位

1：复位

(WT)

SPI_BUF_AFIFO_RST 配置是否复位 buf_tx_afifo。buf_tx_afifo 将在 CPU 控制的从机传输或主机传输中用于发送数据，见图 46.5-3 和图 46.5-4。

0：不复位

1：复位

(WT)

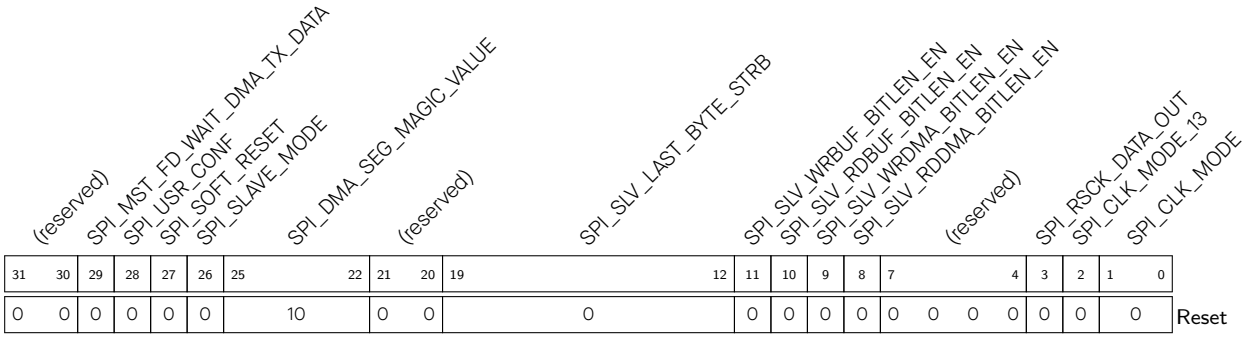
SPI_DMA_AFIFO_RST 配置是否复位 dma_tx_afifo。dma_tx_afifo 在 DMA 控制的从机传输中用于发送数据，见图 46.5-3 和图 46.5-4。

0：不复位

1：复位

(WT)

Register 46.10. SPI_SLAVE_REG (0x00E0)



SPI_CLK_MODE 配置 SPI 时钟模式。

0：CS 信号无效时，SPI 时钟关闭

1：CS 信号无效后，SPI 时钟延迟一个时钟周期

2：CS 信号无效后，SPI 时钟延迟两个时钟周期

3：SPI 时钟一直有效

可在 CONF 阶段配置。

(R/W)

SPI_CLK_MODE_13 配置时钟模式。

0：支持 SPI 时钟模式 0 或 2，见表 46.7-2。

1：支持 SPI 时钟模式 1 或 3，见表 46.7-2。

(R/W)

SPI_RSCK_DATA_OUT 配置输出数据的时钟沿。

0：在 TSCK 上升沿输出数据

1：在 RSCK 上升沿输出数据

(R/W)

SPI_SLV_RDDMA_BITLEN_EN 配置在 DMA 控制的 Rd_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

SPI_SLV_WRDMA_BITLEN_EN 配置在 DMA 控制的 Wr_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_SLV_RDBUF_BITLEN_EN 配置在 CPU 控制的 Rd_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

见下页

Register 46.10. SPI_SLAVE_REG (0x00E0)

接上页

SPI_SLV_WRBUF_BITLEN_EN 配置在 CPU 控制的 Wr_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_SLV_LAST_BYTE_STRB 代表在 SPI 主机全双工传输和半双工传输中，上次接收的字节数据的有效位。

(R/SS)

SPI_DMA_SEG_MAGIC_VALUE 配置 DMA 控制的分段配置传输中位图表的 Magic 值。

(R/W)

SPI_SLAVE_MODE 配置 SPI 工作模式。

0：主机

1：从机

(R/W)

SPI_SOFT_RESET 配置是否软件复位 SPI 时钟线、CS 线和数据线。

0：不复位

1：复位

可在 CONF 阶段配置。

(WT)

SPI_USR_CONF 配置是否使能当前 DMA 控制分段配置传输的 CONF 阶段。

0：无效，表明当前传输不是分段配置传输

1：使能，开始分段配置传输

(R/W)

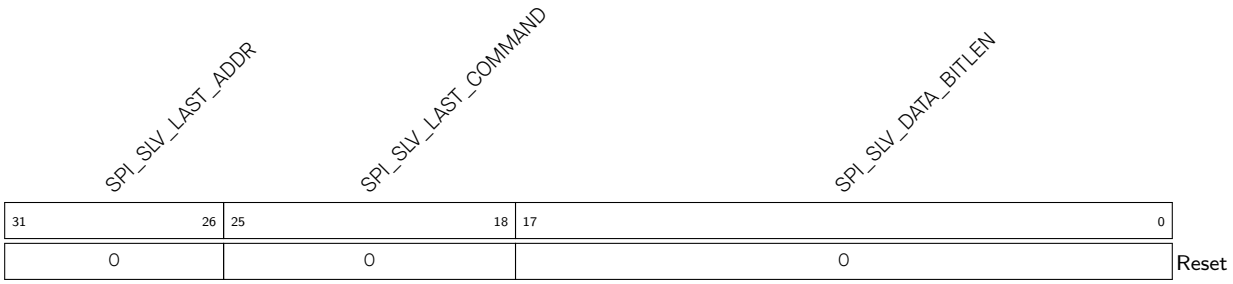
SPI_MST_FD_WAIT_DMA_TX_DATA 配置在主机全双工通信下 SPI 是否先等待 DMA TX 数据准备好之后，再开始 SPI 传输。

0：无需等待 DMA TX 数据即可开始 SPI 传输

1：等待

(R/W)

Register 46.11. SPI_SLAVE1_REG (0x00E4)

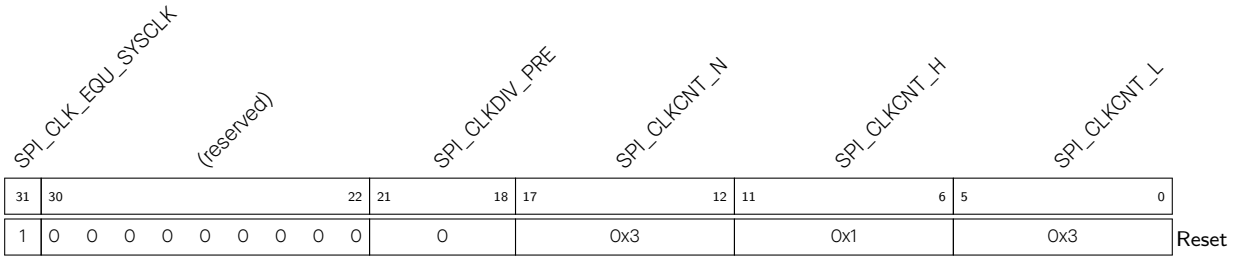


SPI_SLV_DATA_BITLEN 配置在 SPI 从机全双工和半双工传输中，传输的数据位长。
(R/W/SS)

SPI_SLV_LAST_COMMAND 配置从机的命令值。
(R/W/SS)

SPI_SLV_LAST_ADDR 配置从机的地址值。
(R/W/SS)

Register 46.12. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L 配置时钟占空比，与 SPI_CLKCNT_H 和 SPI_CLKCNT_N 配合使用。

用作主机时，必须与 SPI_CLKCNT_N 相等。

在从机模式下，必须为 0。

可在 CONF 阶段配置。

(R/W)

SPI_CLKCNT_H 配置用作主机时 SPI_CLK（高电平）的占空比。

建议将此值配置为 floor((SPI_CLKCNT_N + 1)/2 - 1)。floor() 表示向下取整值，例如 floor(2.2) = 2。

用作从机时，必须配置为 0。

可在 CONF 阶段配置。

(R/W)

SPI_CLKCNT_N 配置用作主机时 SPI_CLK 的分频系数。

SPI_CLK 频率为 $f_{clk_spi_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1)。

可在 CONF 阶段配置。

(R/W)

SPI_CLKDIV_PRE 配置用作主机时 SPI_CLK 的预分频系数。

可在 CONF 阶段配置。

(R/W)

SPI_CLK_EQU_SYSCLK 配置用作主机时 SPI_CLK 频率是否与 clk_spi_mst 频率相同。

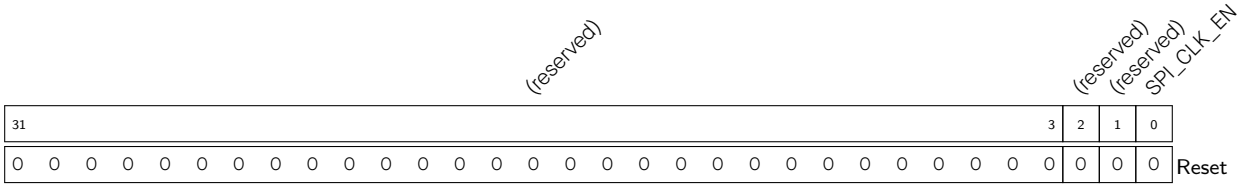
0：SPI_CLK 为 clk_spi_mst 的分频时钟

1：SPI_CLK 与 clk_spi_mst 频率相同

可在 CONF 阶段配置。

(R/W)

Register 46.13. SPI_CLK_GATE_REG (0x00E8)



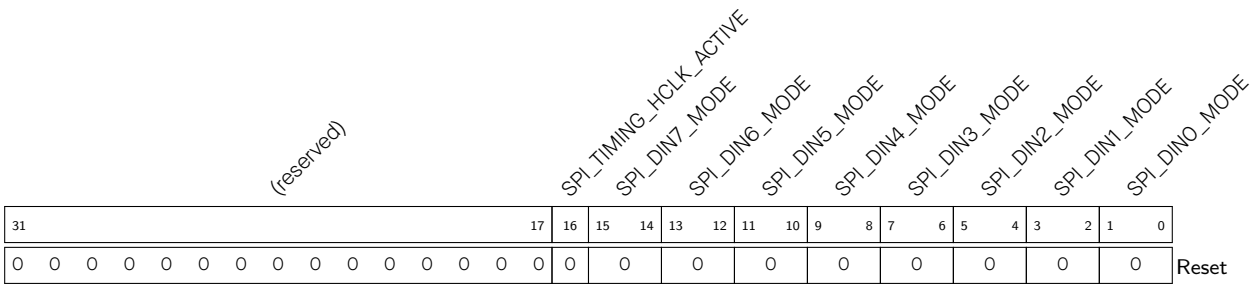
SPI_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能

(R/W)

Register 46.14. SPI_DIN_MODE_REG (0x0024)



SPI_DIN0_MODE 配置 SPI2D 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN0_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN1_MODE 配置 SPI2Q 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN1_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN2_MODE 配置 SPI2WP 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN2_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.14. SPI_DIN_MODE_REG (0x0024)

接上页

SPI_DIN3_MODE 配置 SPI2HD 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN3_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN3_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN3_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN4_MODE 配置 SPI2D4 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN4_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN4_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN4_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN5_MODE 配置 SPI2D5 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN5_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN5_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN5_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN6_MODE 配置 SPI2D6 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN6_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN6_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN6_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.14. SPI_DIN_MODE_REG (0x0024)

接上页

SPI_DIN7_MODE 配置 SPI2D7 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN7_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN7_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN7_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_TIMING_HCLK_ACTIVE 配置是否使能 SPI 输入信号时序模块的高频时钟 HCLK。

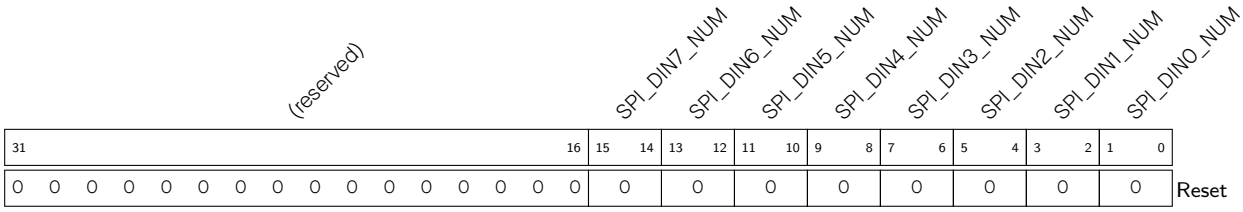
0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

Register 46.15. SPI_DIN_NUM_REG (0x0028)



SPI_DIN0_NUM 配置输入信号 SPI2D 的延迟周期数。具体的延迟模式由 SPI_DIN0_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN1_NUM 配置输入信号 SPI2Q 的延迟周期数。具体的延迟模式由 SPI_DIN1_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN2_NUM 配置输入信号 SPI2WP 的延迟周期数。具体的延迟模式由 SPI_DIN2_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN3_NUM 配置输入信号 SPI2HD 的延迟周期数。具体的延迟模式由 SPI_DIN3_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN4_NUM 配置输入信号 SPI2D4 的延迟周期数。具体的延迟模式由 SPI_DIN4_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

见下页

Register 46.15. SPI_DIN_NUM_REG (0x0028)

接上页

SPI_DIN5_NUM 配置输入信号 SPI2D5 的延迟周期数。具体的延迟模式由 SPI_DIN5_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN6_NUM 配置输入信号 SPI2D6 的延迟周期数。具体的延迟模式由 SPI_DIN6_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN7_NUM 配置输入信号 SPI2D7 的延迟周期数。具体的延迟模式由 SPI_DIN7_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

Register 46.16. SPI_DOUT_MODE_REG (0x002C)

(reserved)	SPI_D_DQS_MODE	SPI_DOUT7_MODE	SPI_DOUT6_MODE	SPI_DOUT5_MODE	SPI_DOUT4_MODE	SPI_DOUT3_MODE	SPI_DOUT2_MODE	SPI_DOUT1_MODE	SPI_DOUT0_MODE	
31 … 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_DOUT0_MODE 配置 SPI2D 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT1_MODE 配置 SPI2Q 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT2_MODE 配置 SPI2WP 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT3_MODE 配置 SPI2HD 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT4_MODE 配置 SPI2D4 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT5_MODE 配置 SPI2D5 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

见下页

Register 46.16. SPI_DOUT_MODE_REG (0x002C)

接上页

SPI_DOUT6_MODE 配置 SPI2D6 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT7_MODE 配置 SPI2D7 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_D_DQS_MODE 配置 SPI_DQS 信号的输出模式。

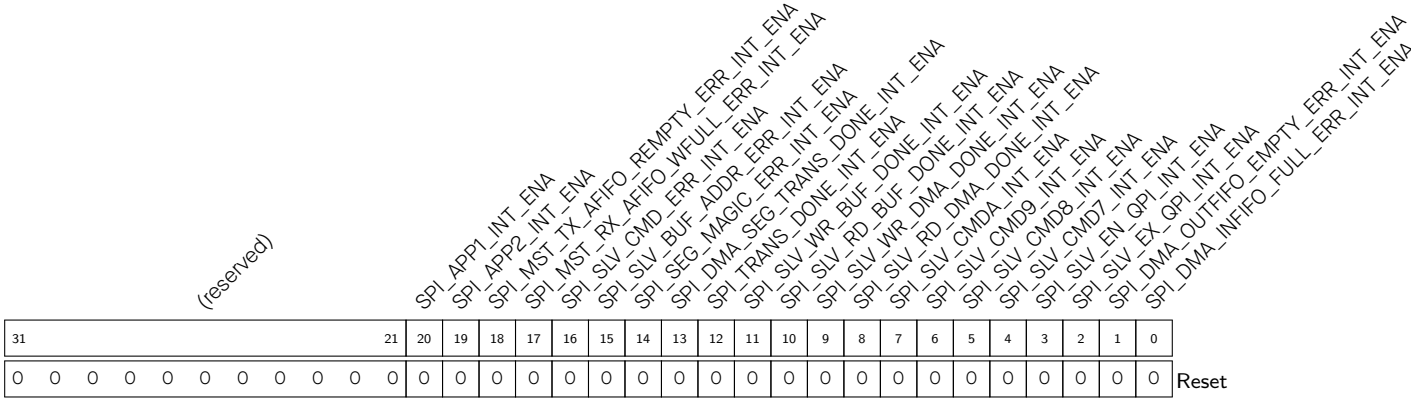
0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

Register 46.17. SPI_DMA_INT_ENA_REG (0x0034)



SPI_DMA_INFIFO_FULL_ERR_INT_ENA 写 1 使能 SPI_DMA_INFIFO_FULL_ERR_INT 中断。
(R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA 写 1 使能 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。
(R/W)

SPI_SLV_EX_QPI_INT_ENA 写 1 使能 SPI_SLV_EX_QPI_INT 中断。
(R/W)

SPI_SLV_EN_QPI_INT_ENA 写 1 使能 SPI_SLV_EN_QPI_INT 中断。
(R/W)

SPI_SLV_CMD7_INT_ENA 写 1 使能 SPI_SLV_CMD7_INT 中断。
(R/W)

SPI_SLV_CMD8_INT_ENA 写 1 使能 SPI_SLV_CMD8_INT 中断。
(R/W)

SPI_SLV_CMD9_INT_ENA 写 1 使能 SPI_SLV_CMD9_INT 中断。
(R/W)

SPI_SLV_CMDA_INT_ENA 写 1 使能 SPI_SLV_CMDA_INT 中断。
(R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_DMA_DONE_INT 中断。
(R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_DMA_DONE_INT 中断。
(R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_BUF_DONE_INT 中断。
(R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_BUF_DONE_INT 中断。
(R/W)

SPI_TRANS_DONE_INT_ENA 写 1 使能 SPI_TRANS_DONE_INT 中断。
(R/W)

见下页

Register 46.17. SPI_DMA_INT_ENA_REG (0x0034)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_ENA 写 1 使能 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(R/W)

SPI_SEG_MAGIC_ERR_INT_ENA 写 1 使能 SPI_SEG_MAGIC_ERR_INT 中断。
(R/W)

SPI_SLV_BUF_ADDR_ERR_INT_ENA 写 1 使能 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(R/W)

SPI_SLV_CMD_ERR_INT_ENA 写 1 使能 SPI_SLV_CMD_ERR_INT 中断。
(R/W)

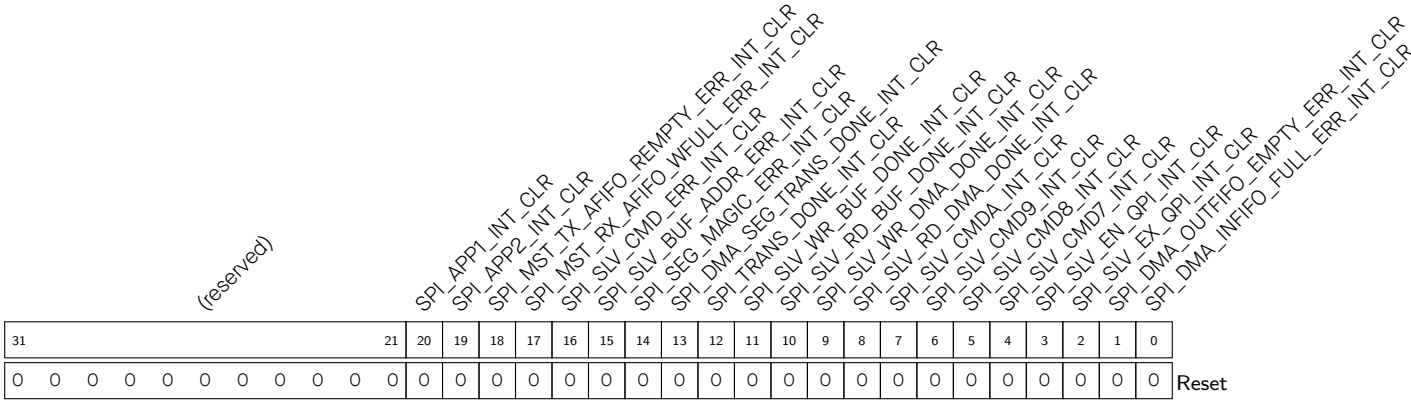
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA 写 1 使能 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA 写 1 使能 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(R/W)

SPI_APP2_INT_ENA 写 1 使能 SPI_APP2_INT 中断。
(R/W)

SPI_APP1_INT_ENA 写 1 使能 SPI_APP1_INT 中断。
(R/W)

Register 46.18. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR 写 1 清除 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR 写 1 清除 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(WT)

SPI_SLV_EX_QPI_INT_CLR 写 1 清除 SPI_SLV_EX_QPI_INT 中断。(WT)

SPI_SLV_EN_QPI_INT_CLR 写 1 清除 SPI_SLV_EN_QPI_INT 中断。(WT)

SPI_SLV_CMD7_INT_CLR 写 1 清除 SPI_SLV_CMD7_INT 中断。(WT)

SPI_SLV_CMD8_INT_CLR 写 1 清除 SPI_SLV_CMD8_INT 中断。(WT)

SPI_SLV_CMD9_INT_CLR 写 1 清除 SPI_SLV_CMD9_INT 中断。(WT)

SPI_SLV_CMDA_INT_CLR 写 1 清除 SPI_SLV_CMDA_INT 中断。(WT)

SPI_SLV_RD_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_DMA_DONE_INT 中断。(WT)

SPI_SLV_WR_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_DMA_DONE_INT 中断。(WT)

SPI_SLV_RD_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_BUF_DONE_INT 中断。(WT)

SPI_SLV_WR_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_BUF_DONE_INT 中断。(WT)

SPI_TRANS_DONE_INT_CLR 写 1 清除 SPI_TRANS_DONE_INT 中断。(WT)

见下页

Register 46.18. SPI_DMA_INT_CLR_REG (0x0038)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_CLR 写 1 清除 SPI_DMA_SEG_TRANS_DONE_INT 中断。(WT)

SPI_SEG_MAGIC_ERR_INT_CLR 写 1 清除 SPI_SEG_MAGIC_ERR_INT 中断。(WT)

SPI_SLV_BUF_ADDR_ERR_INT_CLR 写 1 清除 SPI_SLV_BUF_ADDR_ERR_INT 中断。(WT)

SPI_SLV_CMD_ERR_INT_CLR 写 1 清除 SPI_SLV_CMD_ERR_INT 中断。(WT)

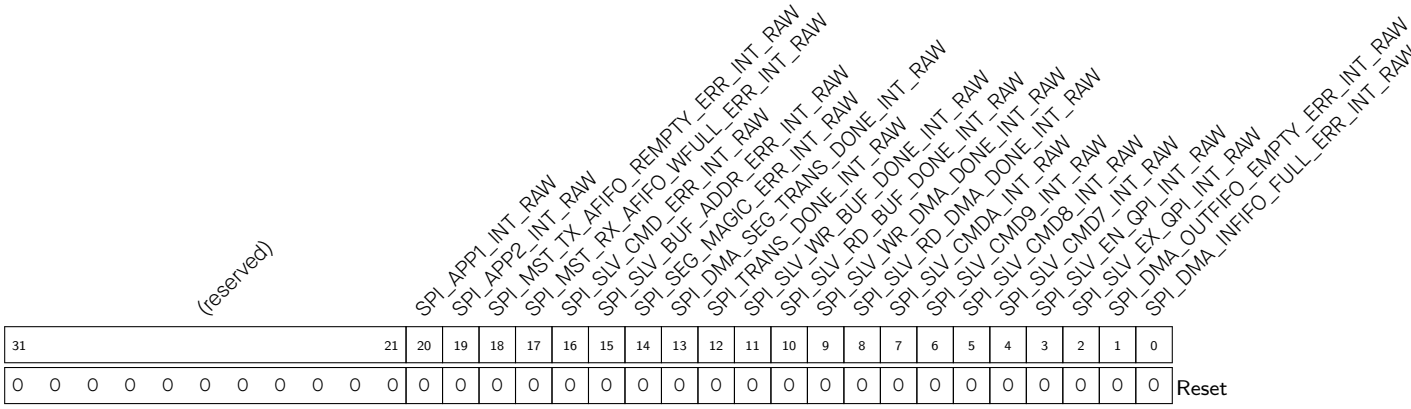
SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR 写 1 清除 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR 写 1 清除 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。(WT)

SPI_APP2_INT_CLR 写 1 清除 SPI_APP2_INT 中断。(WT)

SPI_APP1_INT_CLR 写 1 清除 SPI_APP1_INT 中断。(WT)

Register 46.19. SPI_DMA_INT_RAW_REG (0x003C)



SPI_DMA_INFIFO_FULL_ERR_INT_RAW SPI_DMA_INFIFO_FULL_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW SPI_SLV_EX_QPI_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW SPI_SLV_EN_QPI_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD7_INT_RAW SPI_SLV_CMD7_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD8_INT_RAW SPI_SLV_CMD8_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD9_INT_RAW SPI_SLV_CMD9_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMDA_INT_RAW SPI_SLV_CMDA_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW SPI_SLV_RD_DMA_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW SPI_SLV_WR_DMA_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW SPI_SLV_RD_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_WR_BUF_DONE_INT_RAW SPI_SLV_WR_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

见下页

Register 46.19. SPI_DMA_INT_RAW_REG (0x003C)

接上页

SPI_TRANS_DONE_INT_RAW SPI_TRANS_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_DMA_SEG_TRANS_DONE_INT_RAW SPI_DMA_SEG_TRANS_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW SPI_SEG_MAGIC_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_BUF_ADDR_ERR_INT_RAW SPI_SLV_BUF_ADDR_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW SPI_SLV_CMD_ERR_INT 的原始中断状态。
(R/WTC/SS)

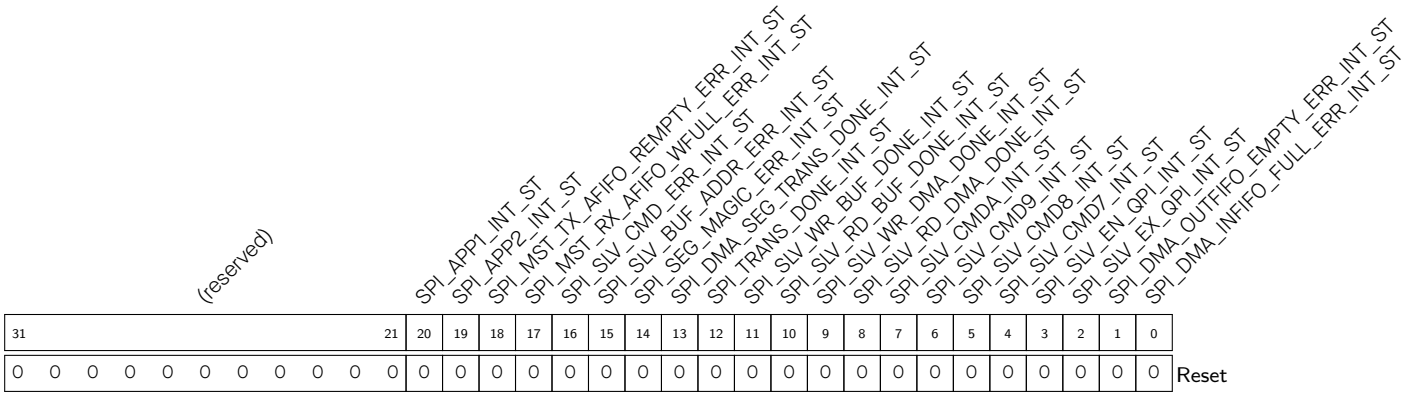
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW SPI_MST_RX_AFIFO_WFULL_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_APP2_INT_RAW SPI_APP2_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

SPI_APP1_INT_RAW SPI_APP1_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

Register 46.20. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST SPI_DMA_INFIFO_FULL_ERR_INT 的中断状态位。
(RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的中断状态位。
(RO)

SPI_SLV_EX_QPI_INT_ST SPI_SLV_EX_QPI_INT 的中断状态位。
(RO)

SPI_SLV_EN_QPI_INT_ST SPI_SLV_EN_QPI_INT 的中断状态位。
(RO)

SPI_SLV_CMD7_INT_ST SPI_SLV_CMD7_INT 的中断状态位。
(RO)

SPI_SLV_CMD8_INT_ST SPI_SLV_CMD8_INT 的中断状态位。
(RO)

SPI_SLV_CMD9_INT_ST SPI_SLV_CMD9_INT 的中断状态位。
(RO)

SPI_SLV_CMDA_INT_ST SPI_SLV_CMDA_INT 的中断状态位。
(RO)

SPI_SLV_RD_DMA_DONE_INT_ST SPI_SLV_RD_DMA_DONE_INT 的中断状态位。
(RO)

SPI_SLV_WR_DMA_DONE_INT_ST SPI_SLV_WR_DMA_DONE_INT 的中断状态位。
(RO)

SPI_SLV_RD_BUF_DONE_INT_ST SPI_SLV_RD_BUF_DONE_INT 的中断状态位。
(RO)

SPI_SLV_WR_BUF_DONE_INT_ST SPI_SLV_WR_BUF_DONE_INT 的中断状态位。
(RO)

SPI_TRANS_DONE_INT_ST SPI_TRANS_DONE_INT 的中断状态位。
(RO)

见下页

Register 46.20. SPI_DMA_INT_ST_REG (0x0040)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_ST SPI_DMA_SEG_TRANS_DONE_INT 的中断状态位。
(RO)

SPI_SEG_MAGIC_ERR_INT_ST SPI_SEG_MAGIC_ERR_INT 的中断状态位。
(RO)

SPI_SLV_BUF_ADDR_ERR_INT_ST SPI_SLV_BUF_ADDR_ERR_INT 的中断状态位。
(RO)

SPI_SLV_CMD_ERR_INT_ST SPI_SLV_CMD_ERR_INT 的中断状态位。
(RO)

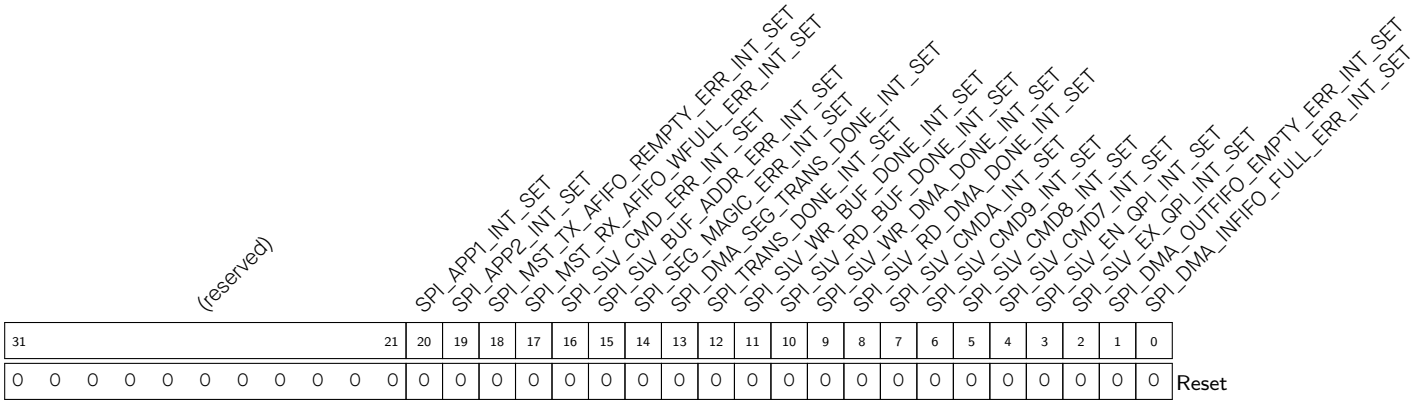
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断状态位。
(RO)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断状态位。
(RO)

SPI_APP2_INT_ST SPI_APP2_INT 的中断状态位。
(RO)

SPI_APP1_INT_ST SPI_APP1_INT 的中断状态位。
(RO)

Register 46.21. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET 写 1 置位 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET 写 1 置位 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(WT)

SPI_SLV_EX_QPI_INT_SET 写 1 置位 SPI_SLV_EX_QPI_INT 中断。(WT)

SPI_SLV_EN_QPI_INT_SET 写 1 置位 SPI_SLV_EN_QPI_INT 中断。(WT)

SPI_SLV_CMD7_INT_SET 写 1 置位 SPI_SLV_CMD7_INT 中断。(WT)

SPI_SLV_CMD8_INT_SET 写 1 置位 SPI_SLV_CMD8_INT 中断。(WT)

SPI_SLV_CMD9_INT_SET 写 1 置位 SPI_SLV_CMD9_INT 中断。(WT)

SPI_SLV_CMDA_INT_SET 写 1 置位 SPI_SLV_CMDA_INT 中断。(WT)

SPI_SLV_RD_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_RD_DMA_DONE_INT 中断。(WT)

SPI_SLV_WR_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_WR_DMA_DONE_INT 中断。(WT)

SPI_SLV_RD_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_RD_BUF_DONE_INT 中断。(WT)

SPI_SLV_WR_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_WR_BUF_DONE_INT 中断。(WT)

见下页

Register 46.21. SPI_DMA_INT_SET_REG (0x0044)

接上页

SPI_TRANS_DONE_INT_SET 写 1 置位 SPI_TRANS_DONE_INT 中断。
(WT)

SPI_DMA_SEG_TRANS_DONE_INT_SET 写 1 置位 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(WT)

SPI_SEG_MAGIC_ERR_INT_SET 写 1 置位 SPI_SEG_MAGIC_ERR_INT 中断。
(WT)

SPI_SLV_BUF_ADDR_ERR_INT_SET 写 1 置位 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(WT)

SPI_SLV_CMD_ERR_INT_SET 写 1 置位 SPI_SLV_CMD_ERR_INT 中断。
(WT)

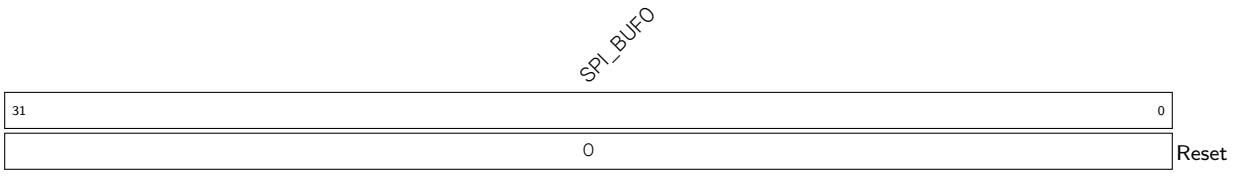
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET 写 1 置位 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET 写 1 置位 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(WT)

SPI_APP2_INT_SET 写 1 置位 SPI_APP2_INT 中断。
(WT)

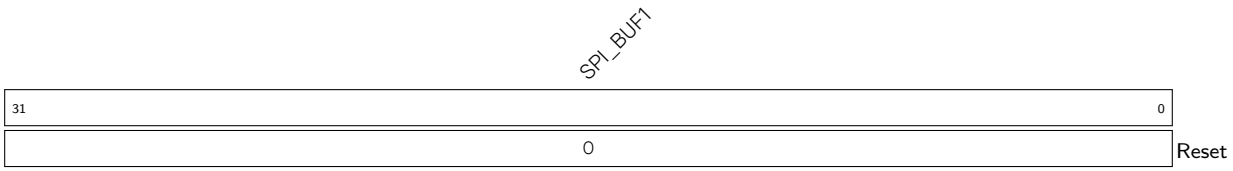
SPI_APP1_INT_SET 写 1 置位 SPI_APP1_INT 中断。
(WT)

Register 46.22. SPI_W0_REG (0x0098)



SPI_BUF0 数据 buffer 0，32 位。
(R/W/SS)

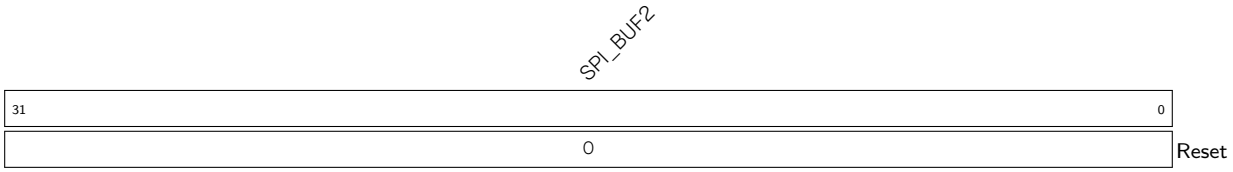
Register 46.23. SPI_W1_REG (0x009C)



SPI_BUF1 数据 buffer 1，32 位。

(R/W/SS)

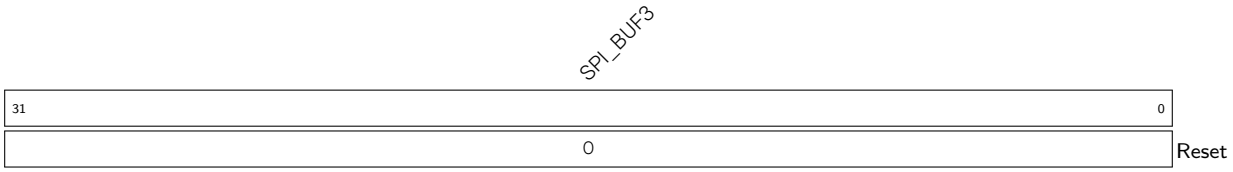
Register 46.24. SPI_W2_REG (0x00A0)



SPI_BUF2 数据 buffer 2，32 位。

(R/W/SS)

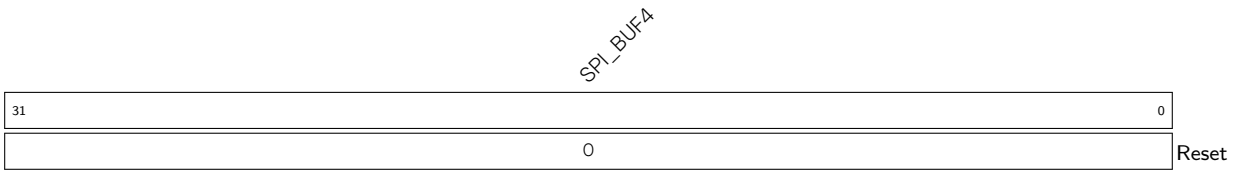
Register 46.25. SPI_W3_REG (0x00A4)



SPI_BUF3 数据 buffer 3，32 位。

(R/W/SS)

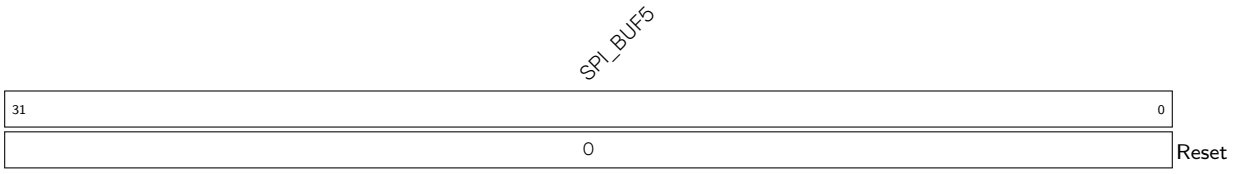
Register 46.26. SPI_W4_REG (0x00A8)



SPI_BUF4 数据 buffer 4，32 位。

(R/W/SS)

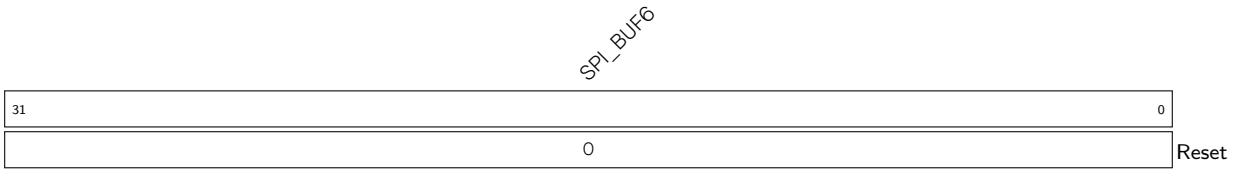
Register 46.27. SPI_W5_REG (0x00AC)



SPI_BUF5 数据 buffer 5，32 位。

(R/W/SS)

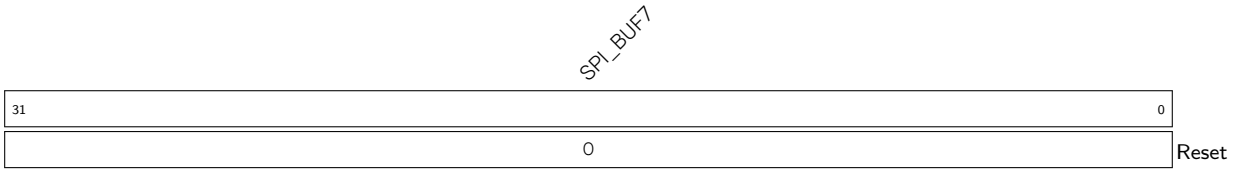
Register 46.28. SPI_W6_REG (0x00B0)



SPI_BUF6 数据 buffer 6，32 位。

(R/W/SS)

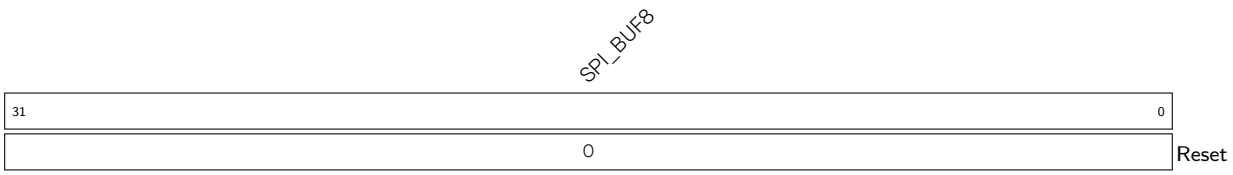
Register 46.29. SPI_W7_REG (0x00B4)



SPI_BUF7 数据 buffer 7，32 位。

(R/W/SS)

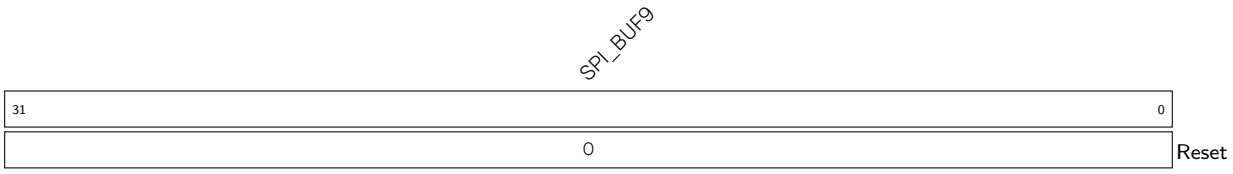
Register 46.30. SPI_W8_REG (0x00B8)



SPI_BUF8 数据 buffer 8，32 位。

(R/W/SS)

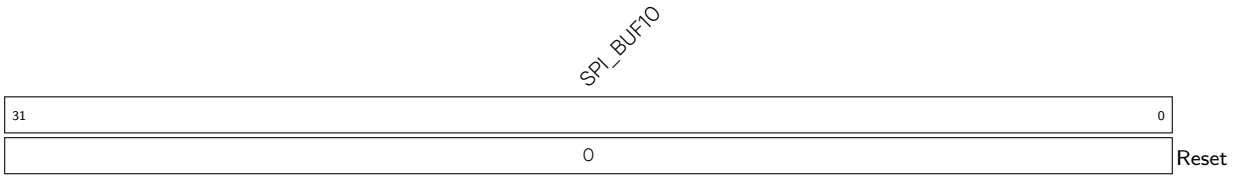
Register 46.31. SPI_W9_REG (0x00BC)



SPI_BUF9 数据 buffer 9，32 位。

(R/W/SS)

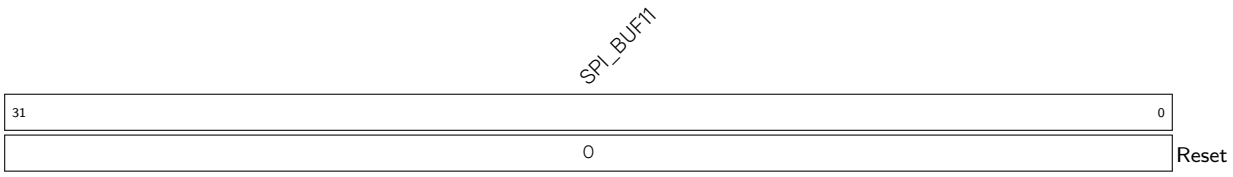
Register 46.32. SPI_W10_REG (0x00C0)



SPI_BUF10 数据 buffer 10，32 位。

(R/W/SS)

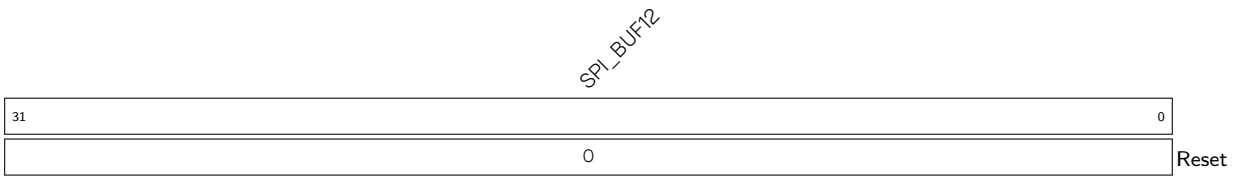
Register 46.33. SPI_W11_REG (0x00C4)



SPI_BUF11 数据 buffer 11，32 位。

(R/W/SS)

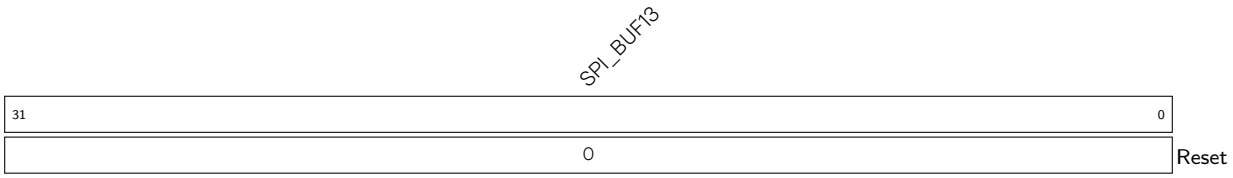
Register 46.34. SPI_W12_REG (0x00C8)



SPI_BUF12 数据 buffer 12，32 位。

(R/W/SS)

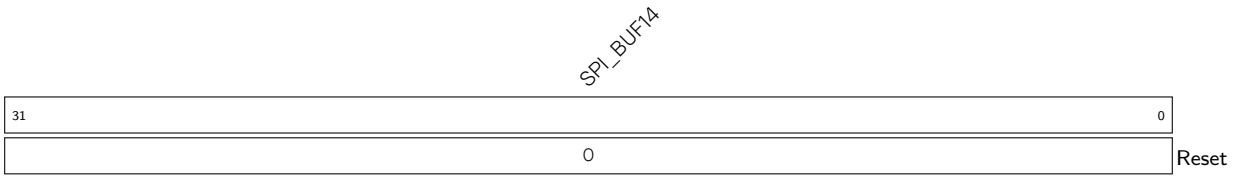
Register 46.35. SPI_W13_REG (0x00CC)



SPI_BUF13 数据 buffer 13，32 位。

(R/W/SS)

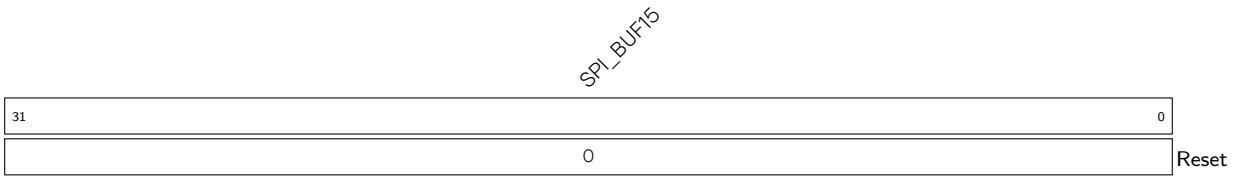
Register 46.36. SPI_W14_REG (0x00D0)



SPI_BUF14 数据 buffer 14，32 位。

(R/W/SS)

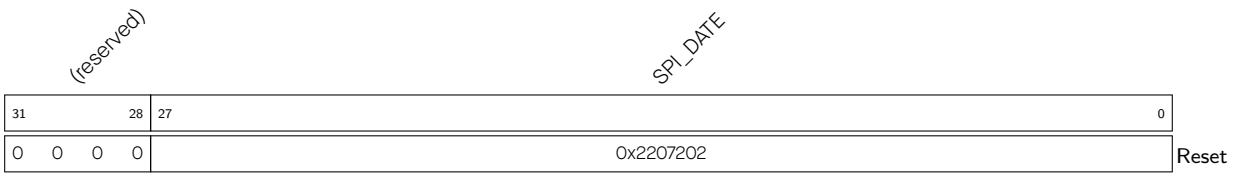
Register 46.37. SPI_W15_REG (0x00D4)



SPI_BUF15 数据 buffer 15，32 位。

(R/W/SS)

Register 46.38. SPI_DATE_REG (0x00F0)



SPI_DATE 版本控制寄存器。

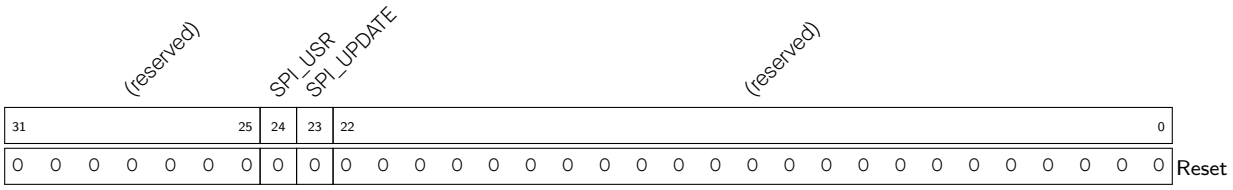
(R/W)

46.13.2 GP-SPI3 寄存器

本小节的所有地址均为相对于 GP-SPI3 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 46.39. SPI_CMD_REG (0x0000)



SPI_UPDATE 配置是否将 SPI 寄存器从 APB 时钟域同步到 SPI 模块时钟域。

0：不同步

1：同步

仅用于 SPI 主机。

(WT)

SPI_USR 配置是否使能用户自定义命令。

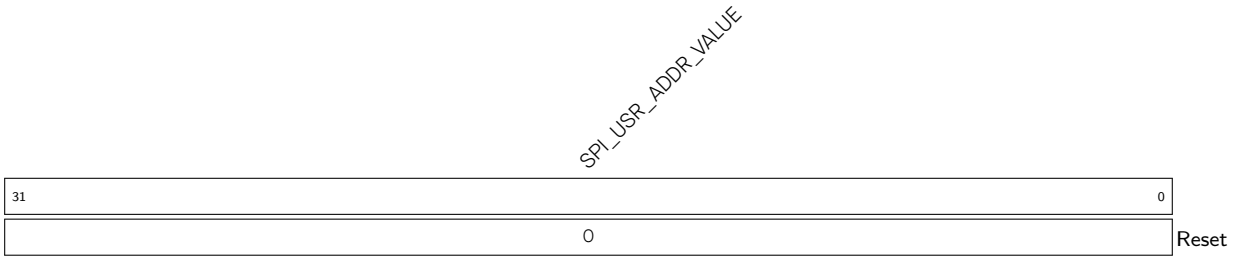
0：不使能

1：使能

置位此位将触发一次 SPI 操作。操作结束后此位被自动清零。

(R/W/SC)

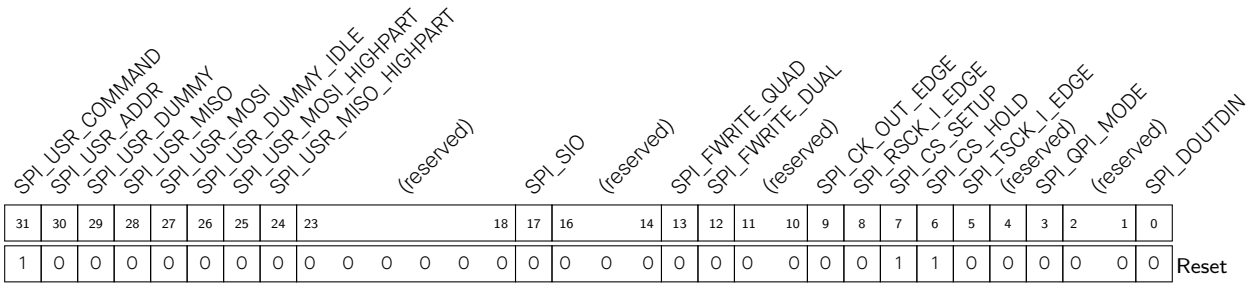
Register 46.40. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE 配置从机地址。

(R/W)

Register 46.41. SPI_USER_REG (0x0010)



SPI_DOUTDIN 配置是否使能全双工通信。

0：不使能

1：使能

(R/W)

SPI_QPI_MODE 配置是否使能 QPI 模式。

0：不使能

1：使能

SPI 主机和从机均支持该配置。(R/W/SS/SC)

SPI_TSCK_I_EDGE 配置 SPI 从机传输中是否更改 TSCK 极性。

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

SPI_CS_HOLD 配置在 SPI 处于完成 (DONE) 阶段时是否保持 SPI CS 拉低。

0：不保持拉低

1：保持拉低

(R/W)

SPI_CS_SETUP 配置在 SPI 处于准备 (PREP) 阶段时是否使能 SPI CS。

0：不使能

1：使能

(R/W)

SPI_RSCK_I_EDGE 配置 SPI 从机传输中是否更改 RSCK 极性。

0: RSCK = !SPI_CK_I

1: RSCK = SPI_CK_I

(R/W)

SPI_CK_OUT_EDGE 配置 SPI 时钟模式，与 SPI_CK_IDLE_EDGE 搭配使用。

更多信息见章节 46.7.4。

(R/W)

见下页

Register 46.41. SPI_USER_REG (0x0010)

接上页

SPI_FWRITE_DUAL 配置在写操作 (DOUT) 阶段时读数据的方式是否为 2-bit 方式。

0：不使能

1：使能

(R/W)

SPI_FWRITE_QUAD 配置在写操作 (DOUT) 阶段时读数据的方式是否为 4-bit 方式。

0：不使能

1：使能

(R/W)

SPI_SIO 配置是否使能 3 线半双工通信，其中 MOSI 和 MISO 信号共享一个管脚。

0：不使能

1：使能

(R/W)

SPI_USR_MISO_HIGHPART 配置在读数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

(R/W)

SPI_USR_MOSI_HIGHPART 配置在写数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

(R/W)

SPI_USR_DUMMY_IDLE 配置是否在 DUMMY 阶段禁用 SPI 时钟。

0：不禁用

1：禁用

(R/W)

SPI_USR_MOSI 配置是否使能一次操作的写数据 (DOUT) 阶段。

0：不使能

1：使能

(R/W)

见下页

Register 46.41. SPI_USER_REG (0x0010)

接上页

SPI_USR_MISO 配置是否使能一次操作的读数据 (DIN) 阶段。

0：不使能

1：使能

SPI_USR_DUMMY 配置是否使能一次操作的 DUMMY 阶段。

0：不使能

1：使能

(R/W)

SPI_USR_ADDR 配置是否使能一次操作的地址 (ADDR) 阶段。

0：不使能

1：使能

(R/W)

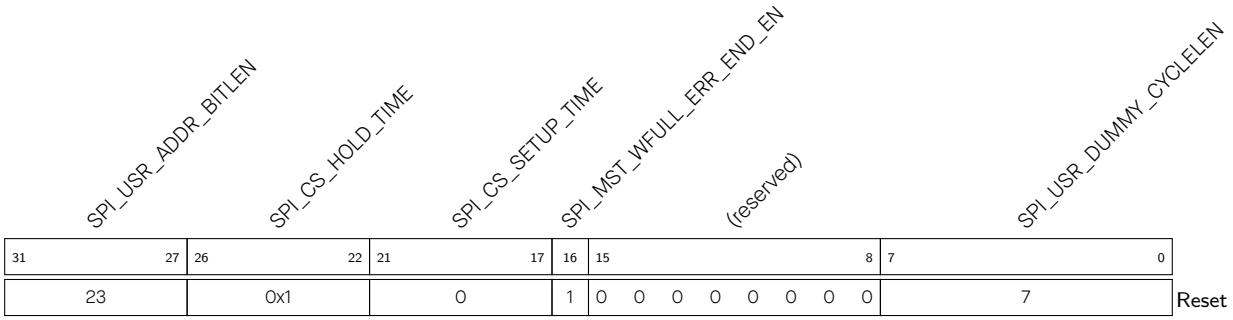
SPI_USR_COMMAND 配置是否使能一次操作的命令 (CMD) 阶段。

0：不使能

1：使能

(R/W)

Register 46.42. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN 配置 DUMMY 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。

(R/W)

SPI_MST_WFULL_ERR_END_EN 配置在主机全双工或半双工通信中，如果发生 RX AFIFO 满错误，是否终止 SPI 传输。

0：不终止

1：终止

(R/W)

SPI_CS_SETUP_TIME 配置准备 (PREP) 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。此字段与 SPI_CS_SETUP 搭配使用。

(R/W)

SPI_CS_HOLD_TIME 配置 CS 管脚的延迟周期。

单位：SPI_CLK 时钟周期。

此字段与 SPI_CS_HOLD 搭配使用。

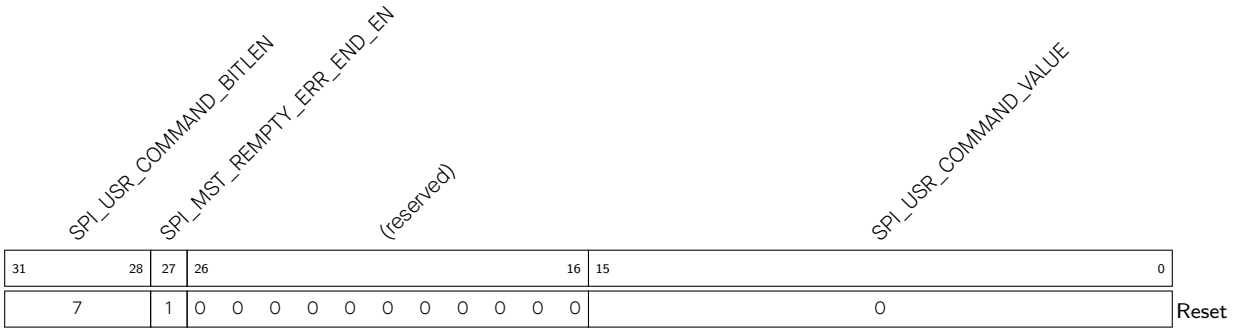
(R/W)

SPI_USR_ADDR_BITLEN 配置地址阶段的位长。

该值等于预期值 - 1。

(R/W)

Register 46.43. SPI_USER2_REG (0x0018)



SPI_USR_COMMAND_VALUE 配置命令值。

(R/W)

SPI_MST_REMPTY_ERR_END_EN 配置在主机全双工或半双工通信下，如果发生 TX AFIFO 空错误，是否终止 SPI 传输。

0：不终止

1：终止

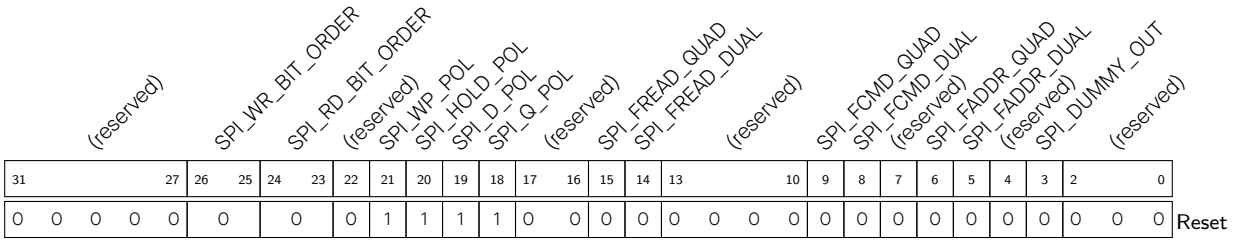
(R/W)

SPI_USR_COMMAND_BITLEN 配置命令阶段的位长。

该值等于预期值 - 1。

(R/W)

Register 46.44. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT 配置在 DUMMY 阶段是否输出 SPI3 总线信号。

0：不输出

1：输出

(R/W)

SPI_FADDR_DUAL 配置在地址 (ADDR) 阶段时是否使能 2-bit 模式。

0：不使能

1：使能

(R/W)

SPI_FADDR_QUAD 配置在地址 (ADDR) 阶段时是否使能 4-bit 模式。

0：不使能

1：使能

(R/W)

SPI_FCMD_DUAL 配置在命令 (CMD) 阶段时是否使能 2-bit 模式。

0：不使能

1：使能

(R/W)

SPI_FCMD_QUAD 配置在命令 (CMD) 阶段时是否使能 4-bit 模式。

0：不使能

1：使能

(R/W)

SPI_FREAD_DUAL 配置在读数据 (DIN) 阶段是否使能 2-bit 模式。

0：不使能

1：使能

(R/W)

SPI_FREAD_QUAD 配置在读数据 (DIN) 阶段是否使能 4-bit 模式。

0：不使能

1：使能

(R/W)

见下页

Register 46.44. SPI_CTRL_REG (0x0008)

接上页

SPI_Q_POL 配置 MISO 的极性。

0：低

1：高

(R/W)

SPI_D_POL 配置 MOSI 的极性。

0：低

1：高

(R/W)

SPI_HOLD_POL 配置在 SPI 空闲状态下 SPI_HOLD 的输出值。

0：输出低电平

1：输出高电平

(R/W)

SPI_WP_POL 配置在 SPI 空闲状态下 WP 信号的输出值。

0：输出低电平

1：输出高电平

(R/W)

SPI_RD_BIT_ORDER 配置读数据 (MISO) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

2：先读高有效位

3：先读低有效位

(R/W)

SPI_WR_BIT_ORDER 配置命令 (CMD)、地址 (ADDR) 和写数据 (MOSI) 阶段的比特顺序。

0：先读高有效位

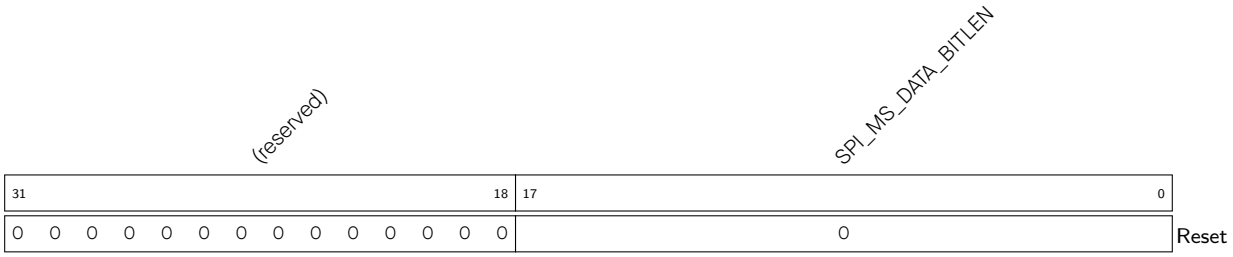
1：先读低有效位

2：先读高有效位

3：先读低有效位

(R/W)

Register 46.45. SPI_MS_DLEN_REG (0x001C)

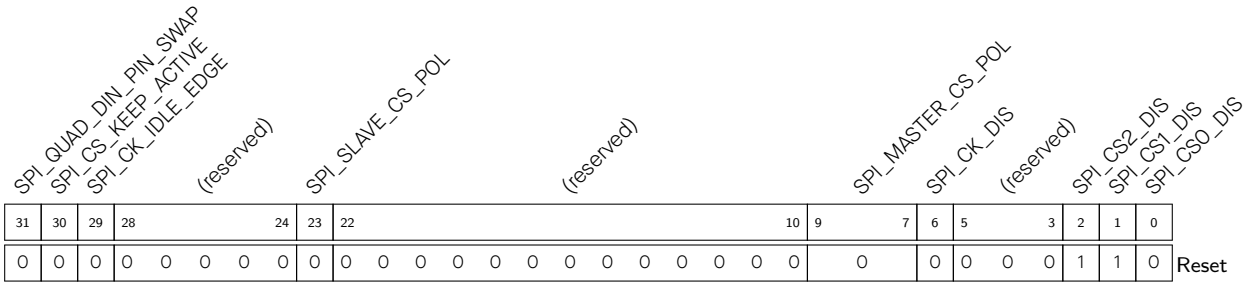


SPI_MS_DATA_BITLEN CPU 控制的主机传输的数据位长或配置 DMA 控制的从机传输中接收数据的位长。

该值等于需要的位长 - 1。

(R/W)

Register 46.46. SPI_MISC_REG (0x0020)



SPI_CS0_DIS 配置是否禁用 SPI_CS0 管脚。

0：SPI_CS0 信号来自 SPI_CS0 管脚或输出至 SPI_CS0 管脚

1：禁用 SPI_CS0 管脚

(R/W)

SPI_CS1_DIS 配置是否禁用 SPI_CS1 管脚。

0：SPI_CS1 信号来自 SPI_CS1 管脚或输出至 SPI_CS1 管脚

1：禁用 SPI_CS1 管脚

(R/W)

SPI_CS2_DIS 配置是否禁用 SPI_CS2 管脚。

0：SPI_CS2 信号来自 SPI_CS2 管脚或输出至 SPI_CS2 管脚

1：禁用 SPI_CS2 管脚

(R/W)

SPI_CK_DIS 配置是否禁用 SPI_CLK 输出信号。

0：使能

1：禁用

(R/W)

SPI_MASTER_CS_POL 配置主机 SPI_CS*n* (*n* = 0 ~ 2) 的极性。

0：SPI_CS*n* 低电平有效

1：SPI_CS*n* 高电平有效

(R/W)

SPI_SLAVE_CS_POL 配置是否反相 SPI 从机输入信号 CS 的极性。

0：保持不变

1：反相

(R/W)

见下页

Register 46.46. SPI_MISC_REG (0x0020)

接上页

SPI_CK_IDLE_EDGE 配置 SPI_CLK 线在 GP-SPI3 空闲状态时的电平。

0：低电平

1：高电平

(R/W)

SPI_CS_KEEP_ACTIVE 配置 SPI_CS 是否保持低电平。

0：不保持低电平

1：保持低电平

(R/W)

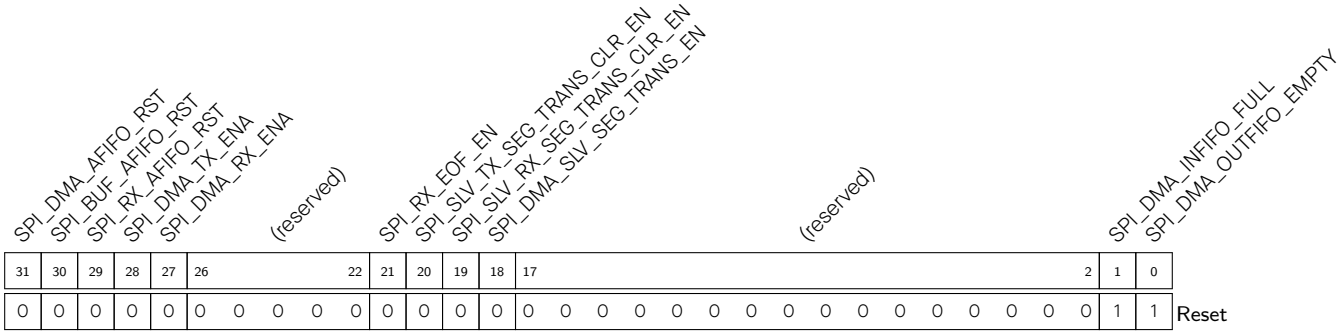
SPI_QUAD_DIN_PIN_SWAP 配置是否交换 SPI Quad 输入管脚。

0：不交换

1：SPI3_D 与 SPI3_Q 交换，SPI3_WP 与 SPI3_HD 交换

(R/W)

Register 46.47. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY 表示 DMA TX FIFO 是否就绪。

0：DMA TX FIFO 已就绪，可以发送数据

1：DMA TX FIFO 尚未就绪，不能发送数据

(RO)

SPI_DMA_INFIFO_FULL 表示 DMA RX FIFO 是否就绪。

0：DMA RX FIFO 已就绪，可以接收数据

1：DMA RX FIFO 尚未就绪，不能接收数据

(RO)

SPI_DMA_SLV_SEG_TRANS_EN 配置是否使能半双工通信方式下，DMA 控制的从机连续传输。

0：不使能

1：使能

(R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA RX buffer 小于实际接收的数据长度：

1：后续 Wr_DMA 传输事务中传输的数据都不接收

0：当前 Wr_DMA 传输事务中传输的数据不接收，但在后续的 Wr_DMA 传输事务中：

- 如果 DMA RX buffer 长度不为 0，则后续 Wr_DMA 传输事务中传输的数据会被接收。
- 如果 DMA RX buffer 长度为 0，则后续 Wr_DMA 传输事务中传输的数据不会被接收。

(R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA TX buffer 小于实际发送的数据长度：

1：后续传输事务中传输的数据都不更新，即旧数据被重复发送

0：当前传输事务中传输的数据不更新，但在后续的传输事务中：

- 如果有新的数据填充到 DMA TX FIFO，则将发送新数据。
- 如果没有新的数据填充到 DMA TX FIFO，则没有新数据被发送。

(R/W)

SPI_RX_EOF_EN 配置 GDMA_IN_SUC_EOF_CH*n*_INT_RAW 中断的触发源。

0：在单次传输中，GDMA_IN_SUC_EOF_CH*n*_INT_RAW 由 SPI_TRANS_DONE_INT 事件置位；或在从机连续传输中，由 SPI_DMA_SEG_TRANS_DONE_INT 事件置位。

1：在 DMA 控制的数据传输过程中，如果 DMA 传输的数据比特数等于 (SPI_MS_DATA_BITLEN + 1)，则硬件会置位 GDMA_IN_SUC_EOF_CH*n*_INT_RAW。

(R/W)

见下页

Register 46.47. SPI_DMA_CONF_REG (0x0030)

接上页

SPI_DMA_RX_ENA 配置是否使能 DMA 控制的接收数据模式。

0：不使能

1：使能

(R/W)

SPI_DMA_TX_ENA 配置是否使能 DMA 控制的发送数据模式。

0：不使能

1：使能

(R/W)

SPI_RX_AFIFO_RST 配置是否复位 spi_rx_afifo。spi_rx_afifo 将在 SPI 主机和从机传输中用于接收数据，见图 46.5-3 和图 46.5-4。

0：不复位

1：复位

(WT)

SPI_BUF_AFIFO_RST 配置是否复位 buf_tx_afifo。buf_tx_afifo 将在 CPU 控制的从机传输或主机传输中用于发送数据，见图 46.5-3 和图 46.5-4。

0：不复位

1：复位

(WT)

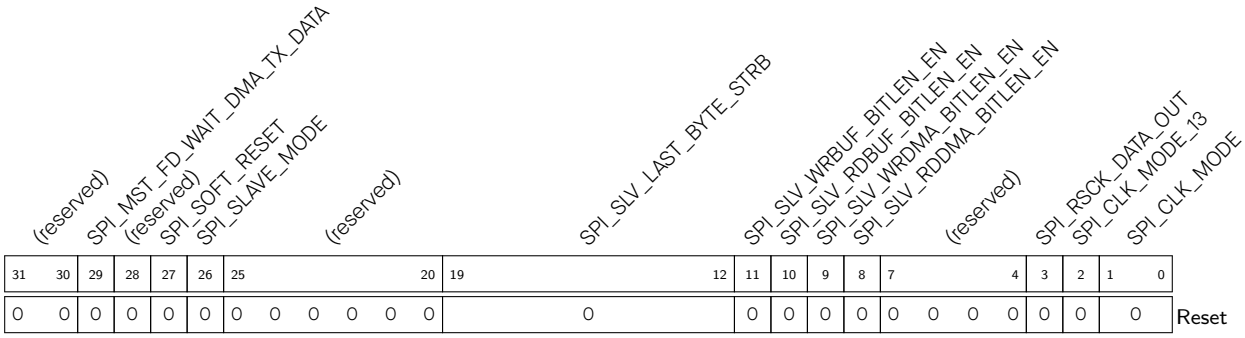
SPI_DMA_AFIFO_RST 配置是否复位 dma_tx_afifo。dma_tx_afifo 在 DMA 控制的从机传输中用于发送数据，见图 46.5-3 和图 46.5-4。

0：不复位

1：复位

(WT)

Register 46.48. SPI_SLAVE_REG (0x00E0)



SPI_CLK_MODE 配置 SPI 时钟模式。

0：CS 信号无效时，SPI 时钟关闭

1：CS 信号无效后，SPI 时钟延迟一个时钟周期

2：CS 信号无效后，SPI 时钟延迟两个时钟周期

3：SPI 时钟一直有效

(R/W)

SPI_CLK_MODE_13 配置时钟模式。

0：支持 SPI 时钟模式 0 或 2，见表 46.7-2。

1：支持 SPI 时钟模式 1 或 3，见表 46.7-2。

(R/W)

SPI_RSCK_DATA_OUT 配置输出数据的时钟沿。

0：在 TSCK 上升沿输出数据

1：在 RSCK 上升沿输出数据

(R/W)

SPI_SLV_RDDMA_BITLEN_EN 配置在 DMA 控制的 Rd_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

SPI_SLV_WRDMA_BITLEN_EN 配置在 DMA 控制的 Wr_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_SLV_RDBUF_BITLEN_EN 配置在 CPU 控制的 Rd_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

见下页

Register 46.48. SPI_SLAVE_REG (0x00E0)

接上页

SPI_SLV_WRBUF_BITLEN_EN 配置在 CPU 控制的 Wr_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_SLV_LAST_BYTE_STRB 代表在 SPI 主机全双工传输和半双工传输中，上次接收的字节数据的有效位。(R/SS)

SPI_SLAVE_MODE 配置 SPI 工作模式。

0：主机

1：从机

(R/W)

SPI_SOFT_RESET 配置是否软件复位 SPI 时钟线、CS 线和数据线。

0：不复位

1：复位

(WT)

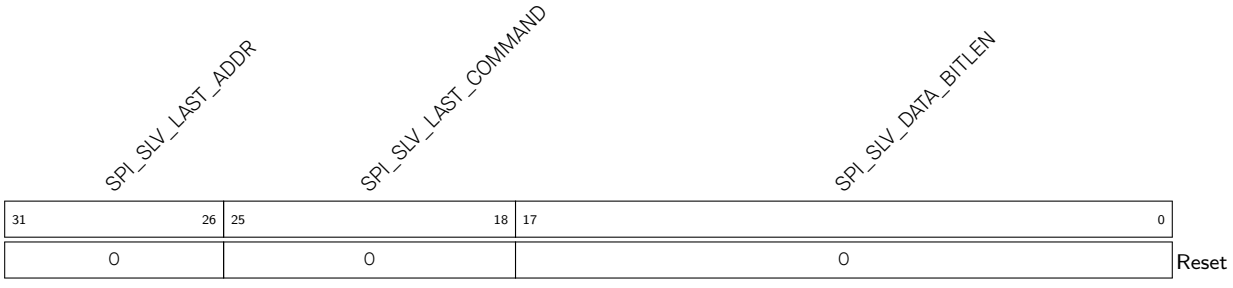
SPI_MST_FD_WAIT_DMA_TX_DATA 配置在主机全双工通信下 SPI 是否先等待 DMA TX 数据准备好之后，再开始 SPI 传输。

0：无需等待 DMA TX 数据即可开始 SPI 传输

1：等待

(R/W)

Register 46.49. SPI_SLAVE1_REG (0x00E4)



SPI_SLV_DATA_BITLEN 配置在 SPI 从机全双工和半双工传输中，传输的数据位长。

(R/W/SS)

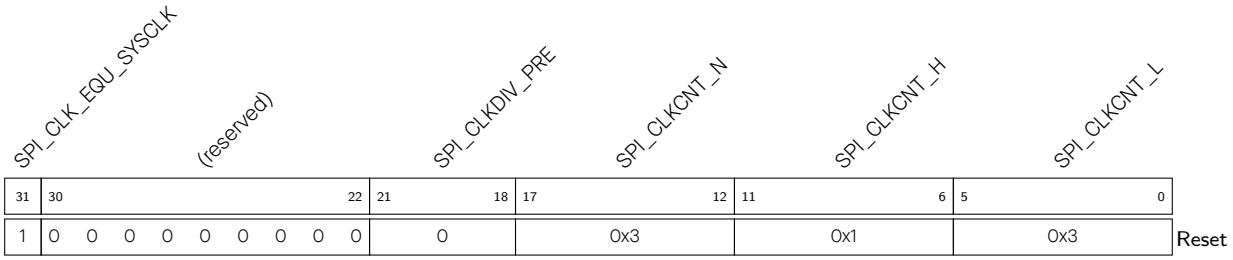
SPI_SLV_LAST_COMMAND 配置从机的命令值。

(R/W/SS)

SPI_SLV_LAST_ADDR 配置从机的地址值。

(R/W/SS)

Register 46.50. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L 配置时钟占空比，与 SPI_CLKCNT_H 和 SPI_CLKCNT_N 配合使用。

用作主机时，必须与 SPI_CLKCNT_N 相等。

在从机模式下，必须为 0。

可在 CONF 阶段配置。

(R/W)

SPI_CLKCNT_H 配置用作主机时 SPI_CLK（高电平）的占空比。

建议将此值配置为 floor((SPI_CLKCNT_N + 1)/2 - 1)。floor() 表示向下取整值，例如 floor(2.2) = 2。

用作从机时，必须配置为 0。

(R/W)

SPI_CLKCNT_N 配置用作主机时 SPI_CLK 的分频系数。

SPI_CLK 频率为 $f_{clk_spi_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1)。

(R/W)

SPI_CLKDIV_PRE 配置用作主机时 SPI_CLK 的预分频系数。

(R/W)

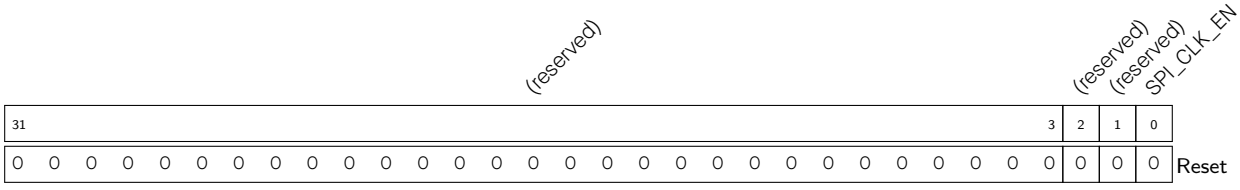
SPI_CLK_EQU_SYSCLK 配置用作主机时 SPI_CLK 频率是否与 clk_spi_mst 频率相同。

0：SPI_CLK 为 clk_spi_mst 的分频时钟

1：SPI_CLK 与 clk_spi_mst 频率相同

(R/W)

Register 46.51. SPI_CLK_GATE_REG (0x00E8)



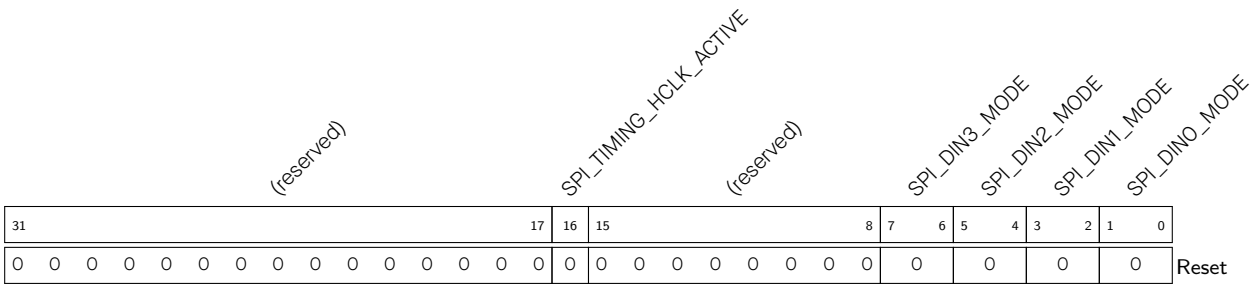
SPI_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能

(R/W)

Register 46.52. SPI_DIN_MODE_REG (0x0024)



SPI_DIN0_MODE 配置 SPI3_D 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN0_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

(R/W)

SPI_DIN1_MODE 配置 SPI3_Q 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN1_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

(R/W)

SPI_DIN2_MODE 配置 SPI3_WP 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN2_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

(R/W)

SPI_DIN3_MODE 配置 SPI3_HD 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN3_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN3_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN3_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

(R/W)

见下页

Register 46.52. SPI_DIN_MODE_REG (0x0024)

接上页

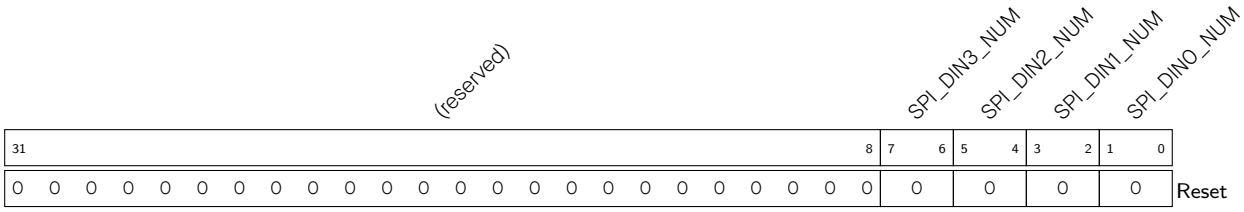
SPI_TIMING_HCLK_ACTIVE 配置是否使能 SPI 输入信号时序模块的高频时钟 HCLK。

0：不使能

1：使能

(R/W)

Register 46.53. SPI_DIN_NUM_REG (0x0028)



SPI_DIN0_NUM 配置输入信号 SPI3_D 的延迟周期数。具体的延迟模式由 SPI_DIN0_MODE 配置。

0：延迟 1 个时钟周期

1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

SPI_DIN1_NUM 配置输入信号 SPI3_Q 的延迟周期数。具体的延迟模式由 SPI_DIN1_MODE 配置。

0：延迟 1 个时钟周期

1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

SPI_DIN2_NUM 配置输入信号 SPI3_WP 的延迟周期数。具体的延迟模式由 SPI_DIN2_MODE 配置。

0：延迟 1 个时钟周期

1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

SPI_DIN3_NUM 配置输入信号 SPI3_HD 的延迟周期数。具体的延迟模式由 SPI_DIN3_MODE 配置。

0：延迟 1 个时钟周期

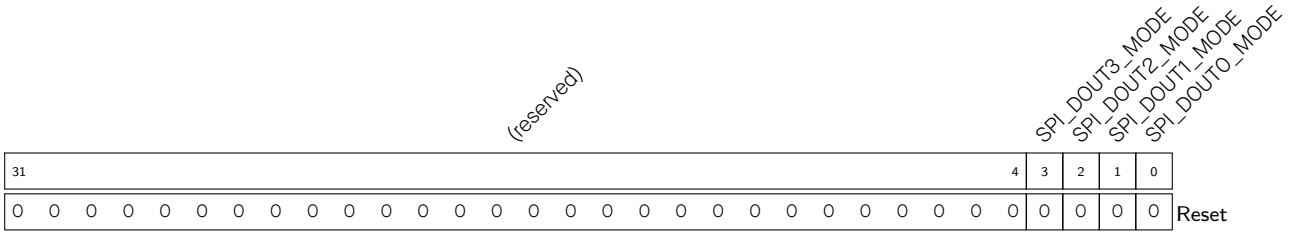
1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

Register 46.54. SPI_DOUT_MODE_REG (0x002C)



SPI_DOUT0_MODE 配置 SPI3_D 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

SPI_DOUT1_MODE 配置 SPI3_Q 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

SPI_DOUT2_MODE 配置 SPI3_WP 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

SPI_DOUT3_MODE 配置 SPI3_HD 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

Register 46.55. SPI_DMA_INT_ENA_REG (0x0034)



SPI_DMA_INFIFO_FULL_ERR_INT_ENA 写 1 使能 SPI_DMA_INFIFO_FULL_ERR_INT 中断。
(R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA 写 1 使能 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。
(R/W)

SPI_SLV_EX_QPI_INT_ENA 写 1 使能 SPI_SLV_EX_QPI_INT 中断。
(R/W)

SPI_SLV_EN_QPI_INT_ENA 写 1 使能 SPI_SLV_EN_QPI_INT 中断。
(R/W)

SPI_SLV_CMD7_INT_ENA 写 1 使能 SPI_SLV_CMD7_INT 中断。
(R/W)

SPI_SLV_CMD8_INT_ENA 写 1 使能 SPI_SLV_CMD8_INT 中断。
(R/W)

SPI_SLV_CMD9_INT_ENA 写 1 使能 SPI_SLV_CMD9_INT 中断。
(R/W)

SPI_SLV_CMDA_INT_ENA 写 1 使能 SPI_SLV_CMDA_INT 中断。
(R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_DMA_DONE_INT 中断。
(R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_DMA_DONE_INT 中断。
(R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_BUF_DONE_INT 中断。
(R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_BUF_DONE_INT 中断。
(R/W)

SPI_TRANS_DONE_INT_ENA 写 1 使能 SPI_TRANS_DONE_INT 中断。
(R/W)

见下页

Register 46.55. SPI_DMA_INT_ENA_REG (0x0034)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_ENA 写 1 使能 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(R/W)

SPI_SLV_BUF_ADDR_ERR_INT_ENA 写 1 使能 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(R/W)

SPI_SLV_CMD_ERR_INT_ENA 写 1 使能 SPI_SLV_CMD_ERR_INT 中断。
(R/W)

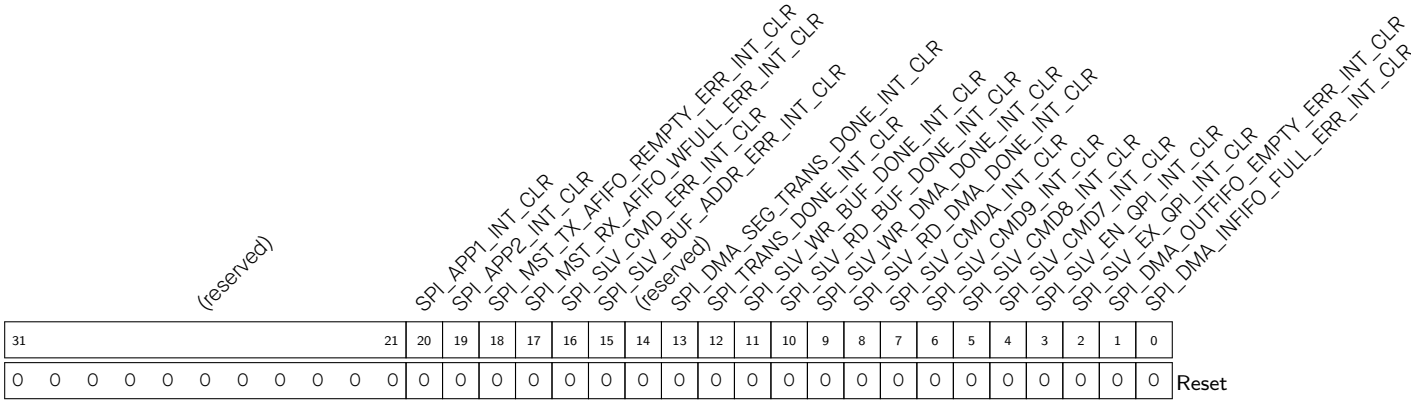
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA 写 1 使能 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA 写 1 使能 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(R/W)

SPI_APP2_INT_ENA 写 1 使能 SPI_APP2_INT 中断。
(R/W)

SPI_APP1_INT_ENA 写 1 使能 SPI_APP1_INT 中断。
(R/W)

Register 46.56. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR 写 1 清除 SPI_DMA_INFIFO_FULL_ERR_INT 中断。 (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR 写 1 清除 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。 (WT)

SPI_SLV_EX_QPI_INT_CLR 写 1 清除 SPI_SLV_EX_QPI_INT 中断。 (WT)

SPI_SLV_EN_QPI_INT_CLR 写 1 清除 SPI_SLV_EN_QPI_INT 中断。 (WT)

SPI_SLV_CMD7_INT_CLR 写 1 清除 SPI_SLV_CMD7_INT 中断。 (WT)

SPI_SLV_CMD8_INT_CLR 写 1 清除 SPI_SLV_CMD8_INT 中断。 (WT)

SPI_SLV_CMD9_INT_CLR 写 1 清除 SPI_SLV_CMD9_INT 中断。 (WT)

SPI_SLV_CMDA_INT_CLR 写 1 清除 SPI_SLV_CMDA_INT 中断。 (WT)

SPI_SLV_RD_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_DMA_DONE_INT 中断。 (WT)

SPI_SLV_WR_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_DMA_DONE_INT 中断。 (WT)

SPI_SLV_RD_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_BUF_DONE_INT 中断。 (WT)

SPI_SLV_WR_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_BUF_DONE_INT 中断。 (WT)

SPI_TRANS_DONE_INT_CLR 写 1 清除 SPI_TRANS_DONE_INT 中断。 (WT)

见下页

Register 46.56. SPI_DMA_INT_CLR_REG (0x0038)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_CLR 写 1 清除 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(WT)

SPI_SLV_BUF_ADDR_ERR_INT_CLR 写 1 清除 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(WT)

SPI_SLV_CMD_ERR_INT_CLR 写 1 清除 SPI_SLV_CMD_ERR_INT 中断。
(WT)

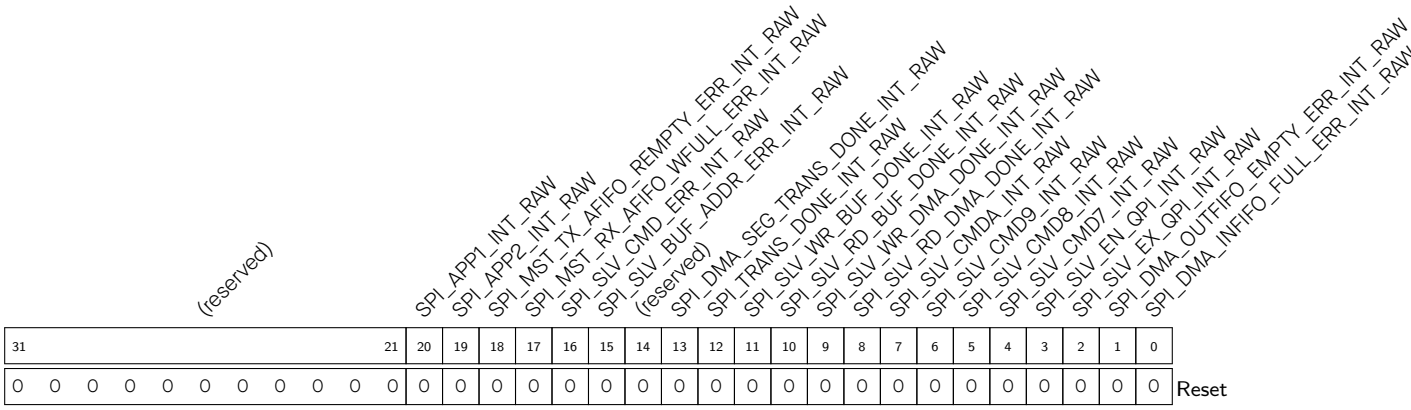
SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR 写 1 清除 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR 写 1 清除 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(WT)

SPI_APP2_INT_CLR 写 1 清除 SPI_APP2_INT 中断。
(WT)

SPI_APP1_INT_CLR 写 1 清除 SPI_APP1_INT 中断。
(WT)

Register 46.57. SPI_DMA_INT_RAW_REG (0x003C)



SPI_DMA_INFIFO_FULL_ERR_INT_RAW SPI_DMA_INFIFO_FULL_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW SPI_SLV_EX_QPI_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW SPI_SLV_EN_QPI_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD7_INT_RAW SPI_SLV_CMD7_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD8_INT_RAW SPI_SLV_CMD8_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD9_INT_RAW SPI_SLV_CMD9_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMDA_INT_RAW SPI_SLV_CMDA_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW SPI_SLV_RD_DMA_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW SPI_SLV_WR_DMA_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW SPI_SLV_RD_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_WR_BUF_DONE_INT_RAW SPI_SLV_WR_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_TRANS_DONE_INT_RAW SPI_TRANS_DONE_INT 的原始中断状态。
(R/WTC/SS)

见下页

Register 46.57. SPI_DMA_INT_RAW_REG (0x003C)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_RAW SPI_DMA_SEG_TRANS_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_BUF_ADDR_ERR_INT_RAW SPI_SLV_BUF_ADDR_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW SPI_SLV_CMD_ERR_INT 的原始中断状态。
(R/WTC/SS)

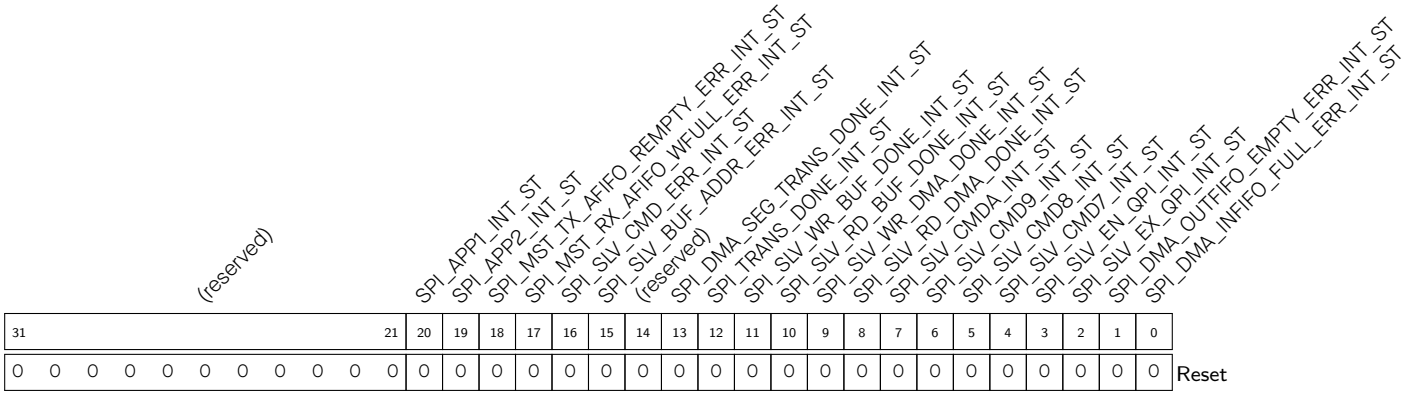
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW SPI_MST_RX_AFIFO_WFULL_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_APP2_INT_RAW SPI_APP2_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

SPI_APP1_INT_RAW SPI_APP1_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

Register 46.58. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST SPI_DMA_INFIFO_FULL_ERR_INT 的中断状态位。
(RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的中断状态位。
(RO)

SPI_SLV_EX_QPI_INT_ST SPI_SLV_EX_QPI_INT 的中断状态位。
(RO)

SPI_SLV_EN_QPI_INT_ST SPI_SLV_EN_QPI_INT 的中断状态位。
(RO)

SPI_SLV_CMD7_INT_ST SPI_SLV_CMD7_INT 的中断状态位。
(RO)

SPI_SLV_CMD8_INT_ST SPI_SLV_CMD8_INT 的中断状态位。
(RO)

SPI_SLV_CMD9_INT_ST SPI_SLV_CMD9_INT 的中断状态位。
(RO)

SPI_SLV_CMDA_INT_ST SPI_SLV_CMDA_INT 的中断状态位。
(RO)

SPI_SLV_RD_DMA_DONE_INT_ST SPI_SLV_RD_DMA_DONE_INT 的中断状态位。
(RO)

SPI_SLV_WR_DMA_DONE_INT_ST SPI_SLV_WR_DMA_DONE_INT 的中断状态位。
(RO)

SPI_SLV_RD_BUF_DONE_INT_ST SPI_SLV_RD_BUF_DONE_INT 的中断状态位。
(RO)

SPI_SLV_WR_BUF_DONE_INT_ST SPI_SLV_WR_BUF_DONE_INT 的中断状态位。
(RO)

SPI_TRANS_DONE_INT_ST SPI_TRANS_DONE_INT 的中断状态位。
(RO)

见下页

Register 46.58. SPI_DMA_INT_ST_REG (0x0040)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_ST SPI_DMA_SEG_TRANS_DONE_INT 的中断状态位。
(RO)

SPI_SLV_BUF_ADDR_ERR_INT_ST SPI_SLV_BUF_ADDR_ERR_INT 的中断状态位。
(RO)

SPI_SLV_CMD_ERR_INT_ST SPI_SLV_CMD_ERR_INT 的中断状态位。
(RO)

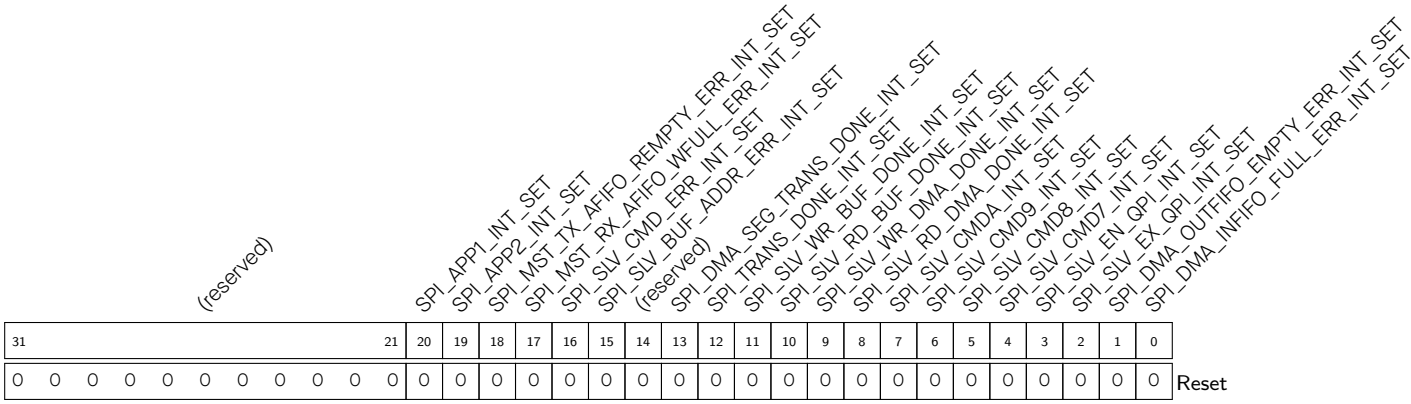
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断状态位。
(RO)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断状态位。
(RO)

SPI_APP2_INT_ST SPI_APP2_INT 的中断状态位。
(RO)

SPI_APP1_INT_ST SPI_APP1_INT 的中断状态位。
(RO)

Register 46.59. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET 写 1 置位 SPI_DMA_INFIFO_FULL_ERR_INT 中断。 (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET 写 1 置位 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。 (WT)

SPI_SLV_EX_QPI_INT_SET 写 1 置位 SPI_SLV_EX_QPI_INT 中断。 (WT)

SPI_SLV_EN_QPI_INT_SET 写 1 置位 SPI_SLV_EN_QPI_INT 中断。 (WT)

SPI_SLV_CMD7_INT_SET 写 1 置位 SPI_SLV_CMD7_INT 中断。 (WT)

SPI_SLV_CMD8_INT_SET 写 1 置位 SPI_SLV_CMD8_INT 中断。 (WT)

SPI_SLV_CMD9_INT_SET 写 1 置位 SPI_SLV_CMD9_INT 中断。 (WT)

SPI_SLV_CMDA_INT_SET 写 1 置位 SPI_SLV_CMDA_INT 中断。 (WT)

SPI_SLV_RD_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_RD_DMA_DONE_INT 中断。 (WT)

SPI_SLV_WR_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_WR_DMA_DONE_INT 中断。 (WT)

SPI_SLV_RD_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_RD_BUF_DONE_INT 中断。 (WT)

SPI_SLV_WR_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_WR_BUF_DONE_INT 中断。 (WT)

SPI_TRANS_DONE_INT_SET 写 1 置位 SPI_TRANS_DONE_INT 中断。 (WT)

见下页

Register 46.59. SPI_DMA_INT_SET_REG (0x0044)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_SET 写 1 置位 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(WT)

SPI_SLV_BUF_ADDR_ERR_INT_SET 写 1 置位 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(WT)

SPI_SLV_CMD_ERR_INT_SET 写 1 置位 SPI_SLV_CMD_ERR_INT 中断。
(WT)

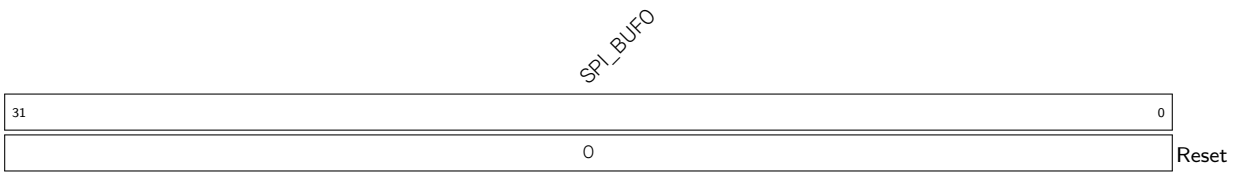
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET 写 1 置位 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET 写 1 置位 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(WT)

SPI_APP2_INT_SET 写 1 置位 SPI_APP2_INT 中断。
(WT)

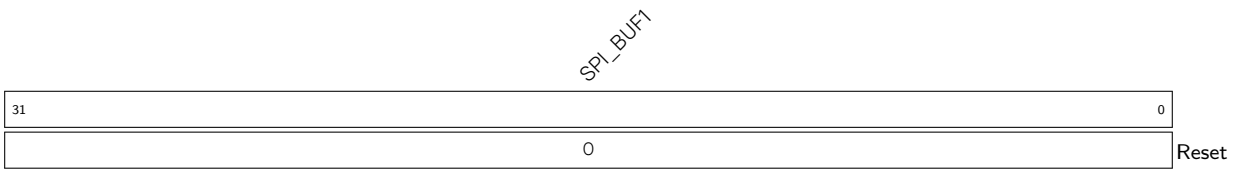
SPI_APP1_INT_SET 写 1 置位 SPI_APP1_INT 中断。
(WT)

Register 46.60. SPI_W0_REG (0x0098)



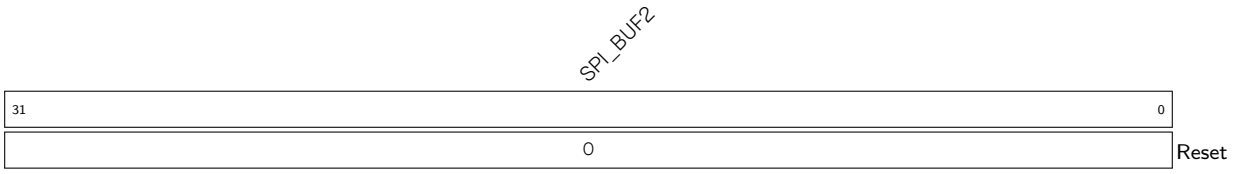
SPI_BUF0 数据 buffer 0，32 位。
(R/W/SS)

Register 46.61. SPI_W1_REG (0x009C)



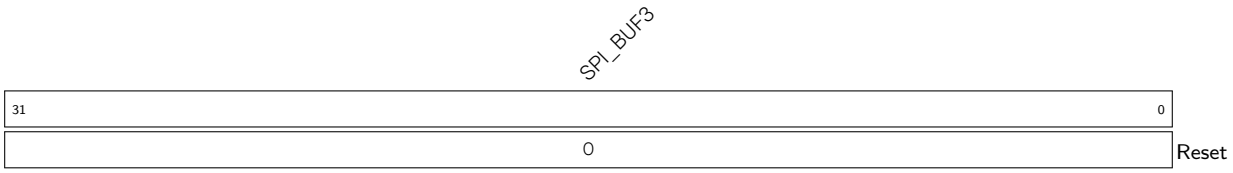
SPI_BUF1 数据 buffer 1，32 位。
(R/W/SS)

Register 46.62. SPI_W2_REG (0x00A0)



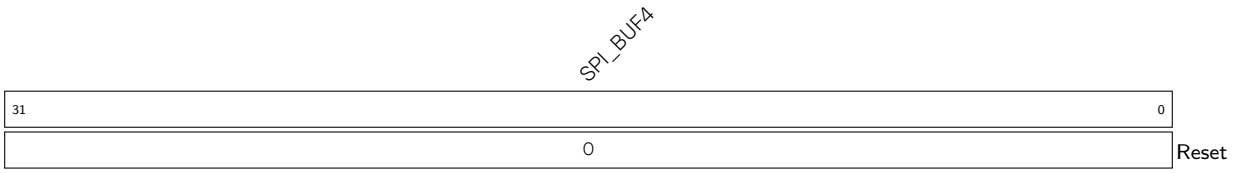
SPI_BUF2 数据 buffer 2，32 位。
(R/W/SS)

Register 46.63. SPI_W3_REG (0x00A4)



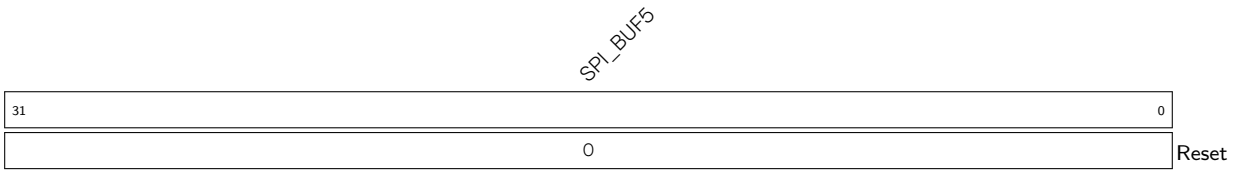
SPI_BUF3 数据 buffer 3，32 位。
(R/W/SS)

Register 46.64. SPI_W4_REG (0x00A8)



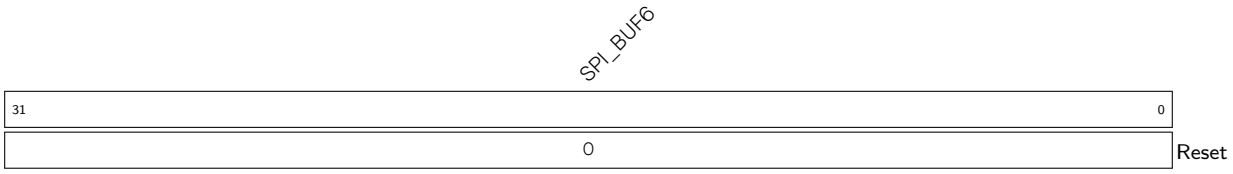
SPI_BUF4 数据 buffer 4，32 位。
(R/W/SS)

Register 46.65. SPI_W5_REG (0x00AC)



SPI_BUF5 数据 buffer 5，32 位。
(R/W/SS)

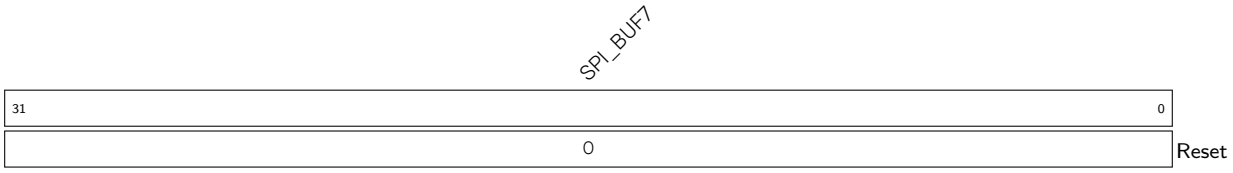
Register 46.66. SPI_W6_REG (0x00B0)



SPI_BUF6 数据 buffer 6，32 位。

(R/W/SS)

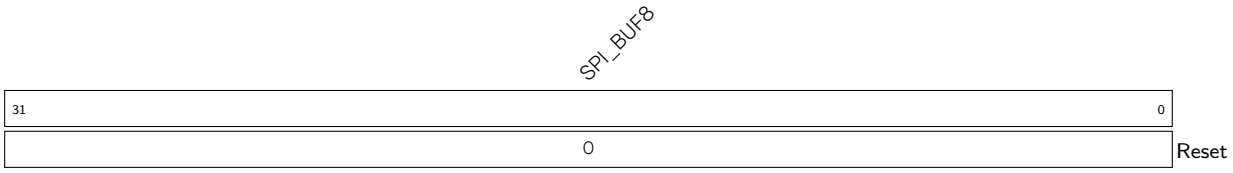
Register 46.67. SPI_W7_REG (0x00B4)



SPI_BUF7 数据 buffer 7，32 位。

(R/W/SS)

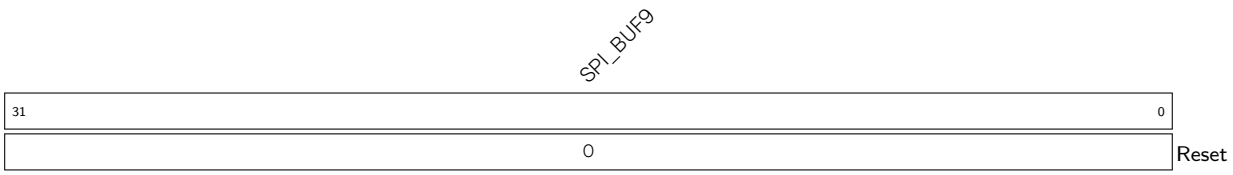
Register 46.68. SPI_W8_REG (0x00B8)



SPI_BUF8 数据 buffer 8，32 位。

(R/W/SS)

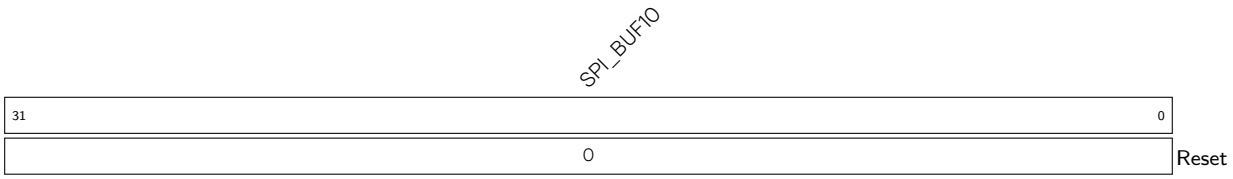
Register 46.69. SPI_W9_REG (0x00BC)



SPI_BUF9 数据 buffer 9，32 位。

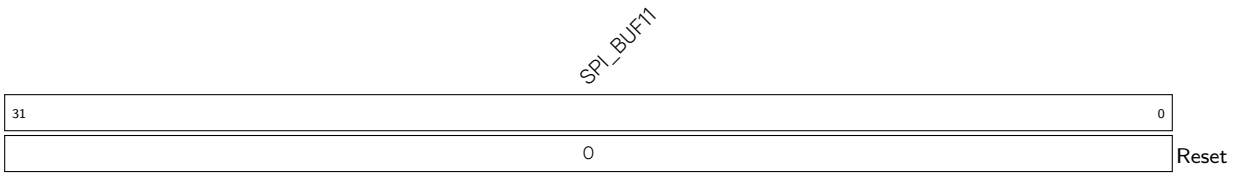
(R/W/SS)

Register 46.70. SPI_W10_REG (0x00C0)



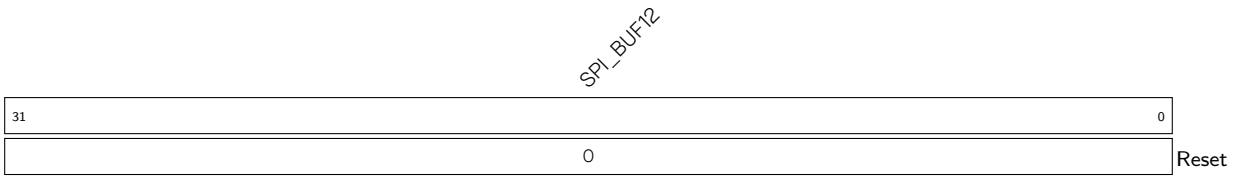
SPI_BUF10 数据 buffer 10，32 位。
(R/W/SS)

Register 46.71. SPI_W11_REG (0x00C4)



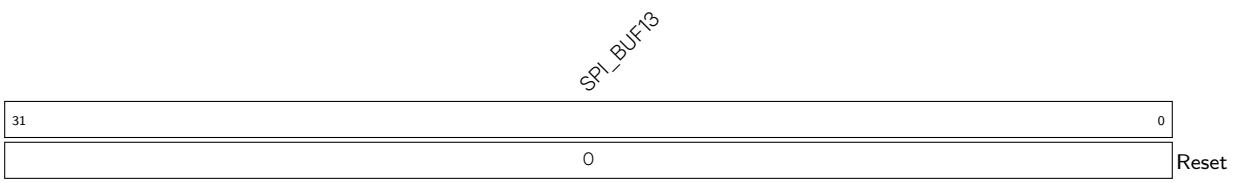
SPI_BUF11 数据 buffer 11，32 位。
(R/W/SS)

Register 46.72. SPI_W12_REG (0x00C8)



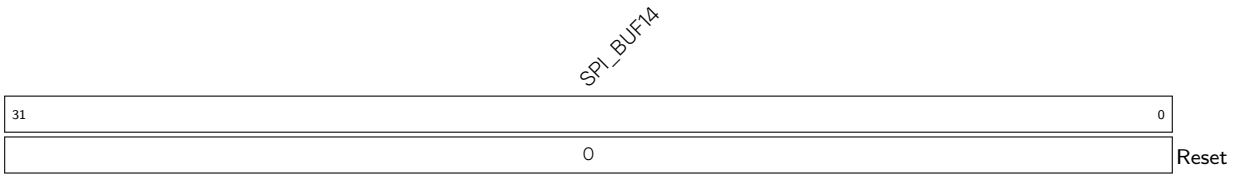
SPI_BUF12 数据 buffer 12，32 位。
(R/W/SS)

Register 46.73. SPI_W13_REG (0x00CC)



SPI_BUF13 数据 buffer 13，32 位。
(R/W/SS)

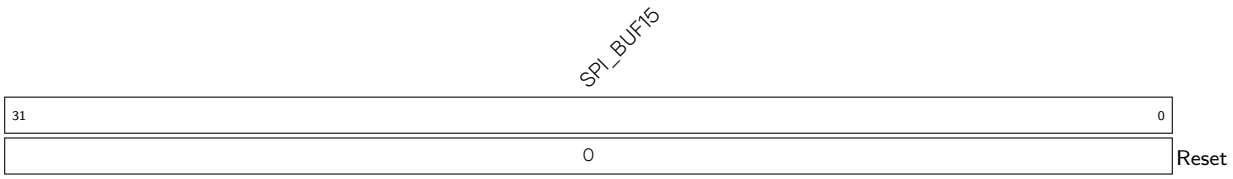
Register 46.74. SPI_W14_REG (0x00D0)



SPI_BUF14 数据 buffer 14，32 位。

(R/W/SS)

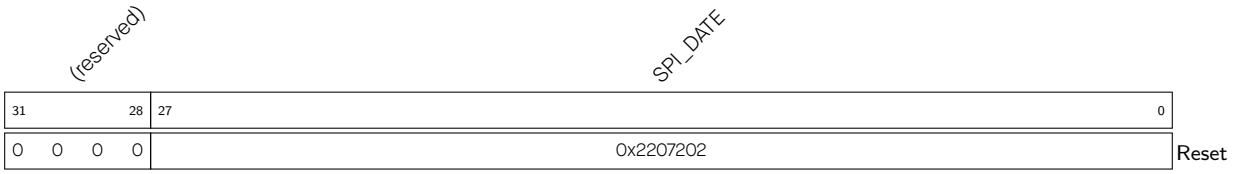
Register 46.75. SPI_W15_REG (0x00D4)



SPI_BUF15 数据 buffer 15，32 位。

(R/W/SS)

Register 46.76. SPI_DATE_REG (0x00F0)



SPI_DATE 版本控制寄存器。

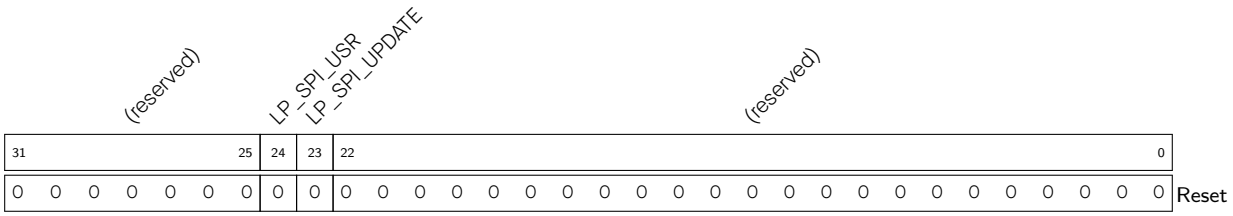
(R/W)

46.13.3 LP-SPI 寄存器

本小节的所有地址均为相对于 LP SPI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 *系统和存储器* 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 *如何配置寄存器的保留域*。

Register 46.77. LP_SPI_CMD_REG (0x0000)



LP_SPI_UPDATE 配置是否将 LP-SPI 寄存器从 APB 时钟域同步到 LP-SPI 模块时钟域。

0：不同步

1：同步

仅用于 LP-SPI 主机。

(WT)

LP_SPI_USR 配置是否使能用户自定义命令。

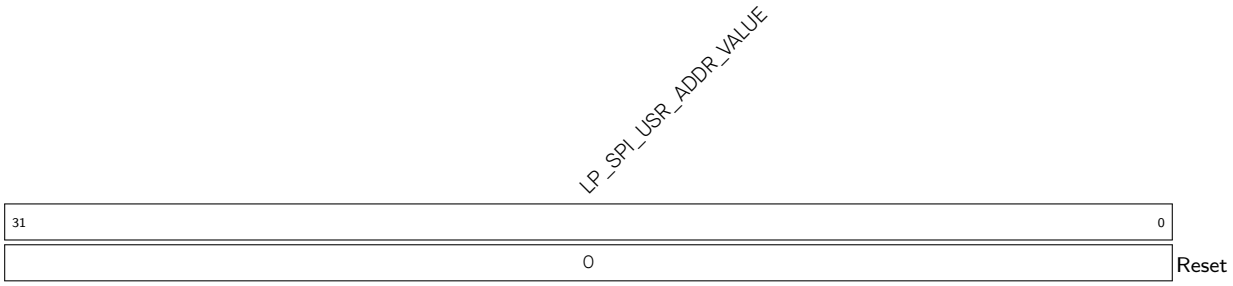
0：不使能

1：使能

置位此位将触发一次 SPI 操作。操作结束后此位被自动清零。

(R/W/SC)

Register 46.78. LP_SPI_ADDR_REG (0x0004)



LP_SPI_USR_ADDR_VALUE 配置从机地址。

(R/W)

Register 46.79. LP_SPI_USER_REG (0x0010)

Bits	Field	Reset
31	LP_SPI_USR_COMMAND	1
30	LP_SPI_USR_ADDR	0
29	LP_SPI_USR_DUMMY	0
28	LP_SPI_USR_MISO	0
27	LP_SPI_USR_MOSI	0
26	LP_SPI_USR_DUMMY_IDLE	0
25	LP_SPI_USR_MOSI_HIGHPART	0
24	LP_SPI_USR_MISO_HIGHPART	0
23–18	(reserved)	0 0 0 0 0 0
17	LP_SPI_SIO	0
16–10	(reserved)	0 0 0 0 0 0 0
9	LP_SPI_CK_OUT_EDGE	0
8	LP_SPI_RSCK_I_EDGE	0
7	LP_SPI_CS_SETUP	1
6	LP_SPI_CS_HOLD	1
5	LP_SPI_TSCK_I_EDGE	0
4–1	(reserved)	0 0 0 0
0	LP_SPI_DOUTDIN	0

LP_SPI_DOUTDIN 配置是否使能全双工通信。

0：不使能

1：使能

(R/W)

LP_SPI_TSCK_I_EDGE 配置 SPI 从机传输中是否更改 TSCK 极性。

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

LP_SPI_CS_HOLD 配置在 SPI 处于完成 (DONE) 阶段时是否保持 SPI CS 拉低。

0：不保持拉低

1：保持拉低

(R/W)

LP_SPI_CS_SETUP 配置在 SPI 处于准备 (PREP) 阶段时是否使能 SPI CS。

0：不使能

1：使能

(R/W)

LP_SPI_RSCK_I_EDGE 配置 SPI 从机传输中是否更改 RSCK 极性。

0: RSCK = !SPI_CK_I

1: RSCK = SPI_CK_I

(R/W)

LP_SPI_CK_OUT_EDGE 配置 SPI 时钟模式，与 LP_SPI_CK_IDLE_EDGE 搭配使用。

更多信息见章节 46.7.4。

(R/W)

LP_SPI_SIO 配置是否使能 3 线半双工通信，其中 MOSI 和 MISO 信号共享一个管脚。

0：不使能

1：使能

(R/W)

见下页

Register 46.79. LP_SPI_USER_REG (0x0010)

接上页

LP_SPI_USR_MISO_HIGHPART 配置在读数据阶段是否使能“高位模式”，即仅访问高位 buffer：LP_SPI_W8_REG ~ LP_SPI_W15_REG。

0：不使能

1：使能

(R/W)

LP_SPI_USR_MOSI_HIGHPART 配置在写数据阶段是否使能“高位模式”，即仅访问高位 buffer：LP_SPI_W8_REG ~ LP_SPI_W15_REG。

0：不使能

1：使能

(R/W)

LP_SPI_USR_DUMMY_IDLE 配置是否在 DUMMY 阶段禁用 SPI 时钟。

0：不禁用

1：禁用

(R/W)

LP_SPI_USR_MOSI 配置是否使能一次操作的写数据 (DOUT) 阶段。

0：不使能

1：使能

(R/W)

LP_SPI_USR_MISO 配置是否使能一次操作的读数据 (DIN) 阶段。

0：不使能

1：使能

(R/W)

LP_SPI_USR_DUMMY 配置是否使能一次操作的 DUMMY 阶段。

0：不使能

1：使能

(R/W)

LP_SPI_USR_ADDR 配置是否使能一次操作的地址 (ADDR) 阶段。

0：不使能

1：使能

(R/W)

见下页

Register 46.79. LP_SPI_USER_REG (0x0010)

接上页

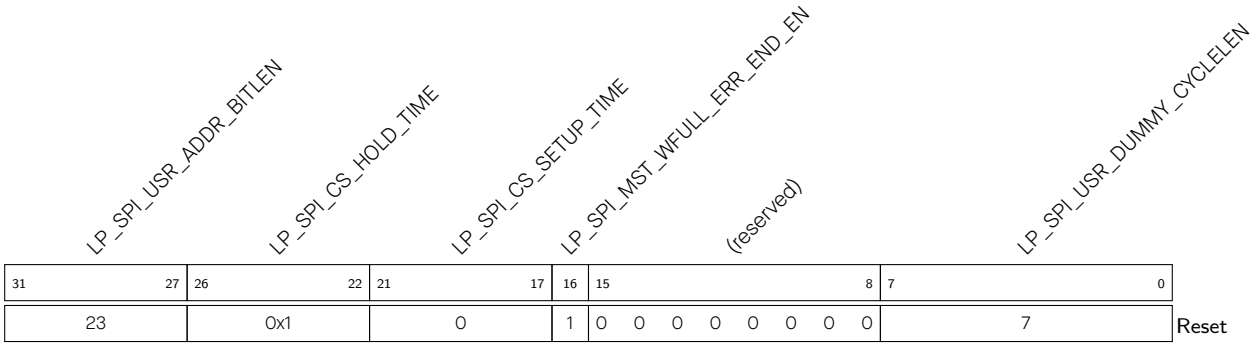
LP_SPI_USR_COMMAND 配置是否使能一次操作的命令 (CMD) 阶段。

0：不使能

1：使能

(R/W)

Register 46.80. LP_SPI_USER1_REG (0x0014)



LP_SPI_USR_DUMMY_CYCLELEN 配置 DUMMY 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。

(R/W)

LP_SPI_MST_WFULL_ERR_END_EN 配置在主机全双工或半双工通信中，如果发生 RX AFIFO 满错误，是否终止 LP-SPI 传输。

0：不终止

1：终止

(R/W)

LP_SPI_CS_SETUP_TIME 配置准备 (PREP) 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。此字段与 LP_SPI_CS_SETUP 搭配使用。

(R/W)

LP_SPI_CS_HOLD_TIME 配置 CS 管脚的延迟周期。

单位：SPI_CLK 时钟周期。

此字段与 LP_SPI_CS_HOLD 搭配使用。

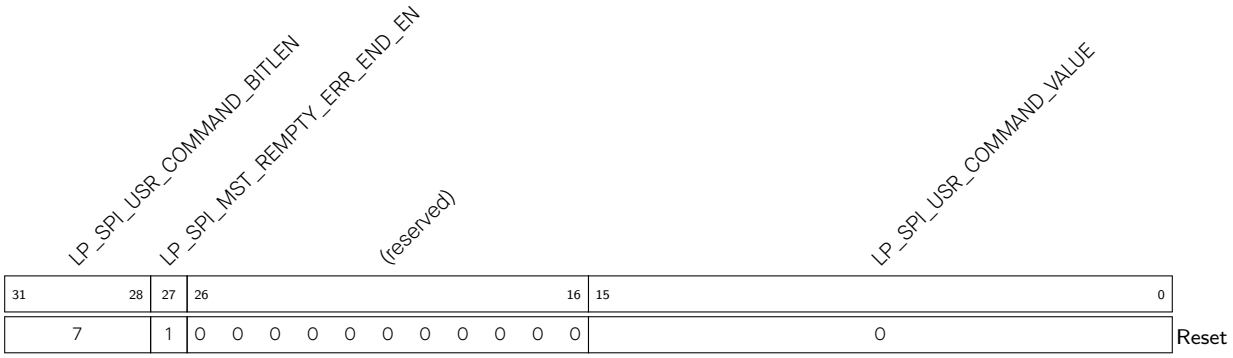
(R/W)

LP_SPI_USR_ADDR_BITLEN 配置地址阶段的位长。

该值等于预期值 - 1。

(R/W)

Register 46.81. LP_SPI_USER2_REG (0x0018)



LP_SPI_USR_COMMAND_VALUE 配置命令值。

(R/W)

LP_SPI_MST_REMPTY_ERR_END_EN 配置在主机全双工或半双工通信下，如果发生 TX AFIFO 空错误，是否终止 LP-SPI 传输。

0：不终止

1：终止

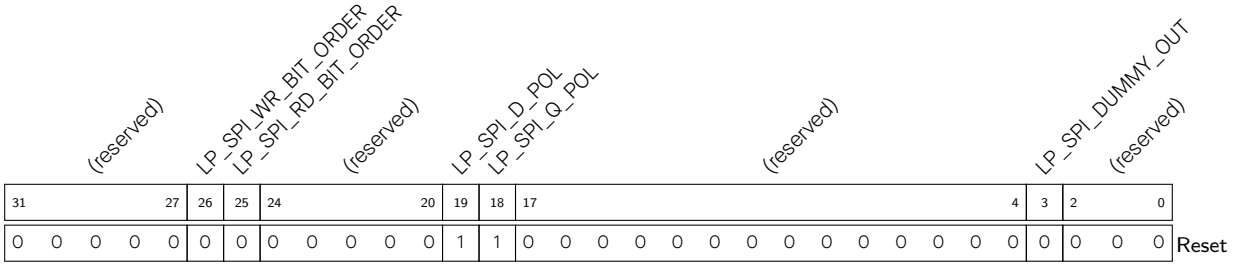
(R/W)

LP_SPI_USR_COMMAND_BITLEN 配置命令阶段的位长。

该值等于预期值 - 1。

(R/W)

Register 46.82. LP_SPI_CTRL_REG (0x0008)



LP_SPI_DUMMY_OUT 配置在 DUMMY 阶段是否输出 LP-SPI 总线信号。

0：不输出

1：输出

(R/W)

LP_SPI_Q_POL 配置 MISO 的极性。

0：低

1：高

(R/W)

LP_SPI_D_POL 配置 MOSI 的极性。

0：低

1：高

(R/W)

LP_SPI_RD_BIT_ORDER 配置读数据 (MISO) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

(R/W)

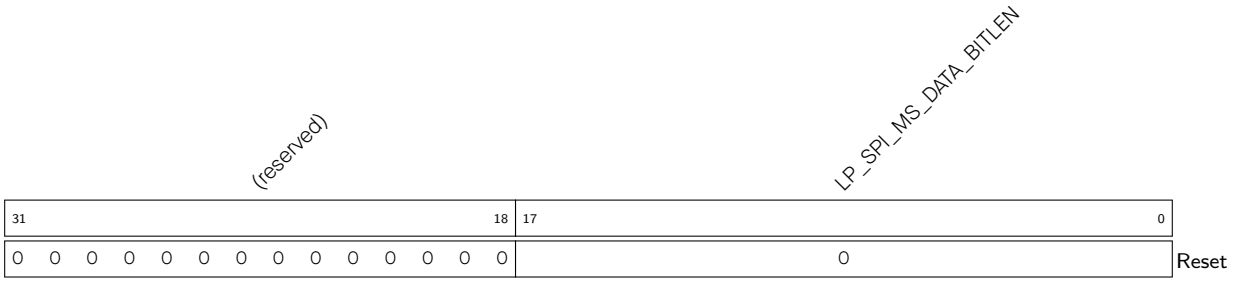
LP_SPI_WR_BIT_ORDER 配置命令 (CMD)、地址 (ADDR) 和写数据 (MOSI) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

(R/W)

Register 46.83. LP_SPI_MS_DLEN_REG (0x001C)



LP_SPI_MS_DATA_BITLEN 配置 CPU 控制的主机传输的数据位长。

该值等于需要的位长 - 1。

(R/W)

Register 46.84. LP_SPI_MISC_REG (0x0020)

(reserved)	LP_SPI_CS_KEEP_ACTIVE	LP_SPI_CK_IDLE_EDGE	(reserved)	LP_SPI_SLAVE_CS_POL	(reserved)	LP_SPI_MASTER_CS_POL	LP_SPI_CK_DIS	(reserved)	LP_SPI_CS0_DIS	
31	30	29	28 24	23	22 10	9 7	6	5 1	0	
0	0	0	0 0 0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0	0	Reset

LP_SPI_CS0_DIS 配置是否禁用 SPI_CS0 管脚。

0：SPI_CS0 信号来自 SPI_CS0 管脚或输出至 SPI_CS0 管脚

1：禁用 SPI_CS0 管脚

(R/W)

LP_SPI_CK_DIS 配置是否禁用 SPI_CLK 输出信号。

0：使能

1：禁用

(R/W)

LP_SPI_MASTER_CS_POL 配置主机 SPI_CS0 的极性。

0：SPI_CS0 低电平有效

1：SPI_CS0 高电平有效

(R/W)

LP_SPI_SLAVE_CS_POL 配置是否反相 SPI 从机输入信号 CS 的极性。

0：保持不变

1：反相

(R/W)

LP_SPI_CK_IDLE_EDGE 配置 SPI_CLK 线在 LP-SPI 空闲状态时的电平。

0：低电平

1：高电平

(R/W)

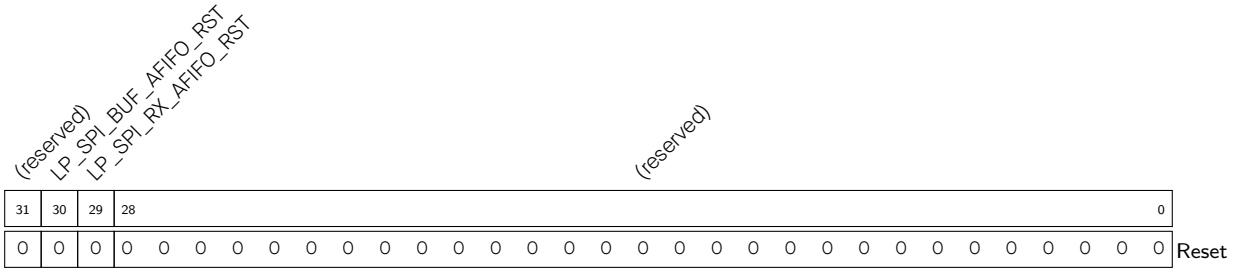
LP_SPI_CS_KEEP_ACTIVE 配置 SPI_CS 是否保持低电平。

0：不保持低电平

1：保持低电平

(R/W)

Register 46.85. LP_SPI_DMA_CONF_REG (0x0030)



LP_SPI_RX_AFIFO_RST 配置是否复位图 46.5-3 和图 46.5-4 中的 spi_rx_afifo。spi_rx_afifo 将在 LP-SPI 主机和从机传输中用于接收数据。

0：不复位

1：复位

(WT)

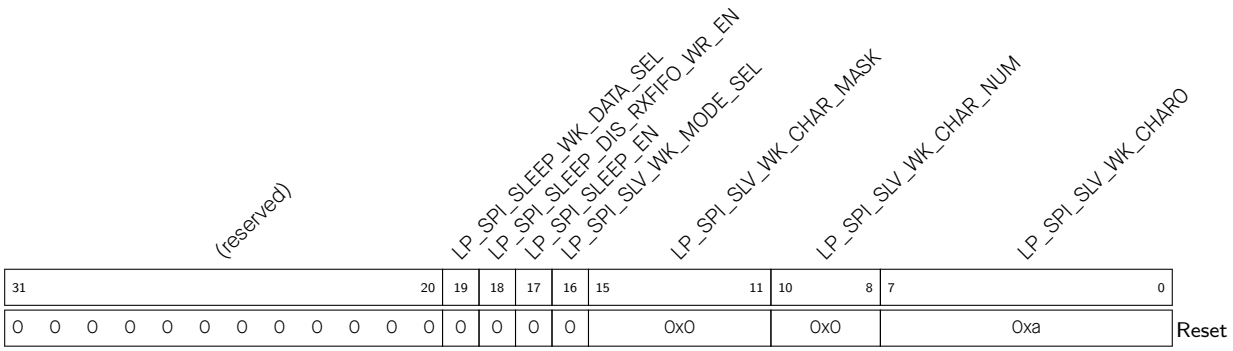
LP_SPI_BUF_AFIFO_RST 配置是否复位图 46.5-3 和图 46.5-4 中的 buf_tx_afifo。buf_tx_afifo 将在 CPU 控制的 LP-SPI 从机传输或主机传输中用于发送数据。

0：不复位

1：复位

(WT)

Register 46.86. LP_SPI_SLEEP_CONF0_REG (0x0044)



LP_SPI_SLV_WK_CHAR0 配置唤醒字符 0。

(R/W)

LP_SPI_SLV_WK_CHAR_NUM 配置启用的唤醒字符数量。

(R/W)

LP_SPI_SLV_WK_CHAR_MASK 配置需要屏蔽的唤醒字符位图。

比特 0 ~ 比特 4 分别对应唤醒字符 0 ~ 唤醒字符 4。

比特值为 0：对应的唤醒字符未被屏蔽

1：对应的唤醒字符被屏蔽

(R/W)

LP_SPI_SLV_WK_MODE_SEL 配置唤醒方式。

0：LP-SPI 从机监测到起始位后，唤醒芯片

1：LP-SPI 从机监测到特定字符序列后，唤醒芯片

(R/W)

LP_SPI_SLEEP_EN 配置是否使能睡眠模式。

0：不使能

1：使能

(R/W)

LP_SPI_SLEEP_DIS_RXFIFO_WR_EN 配置在 LP-SPI 唤醒前是否缓存数据。

0：缓存

1：不缓存

(R/W)

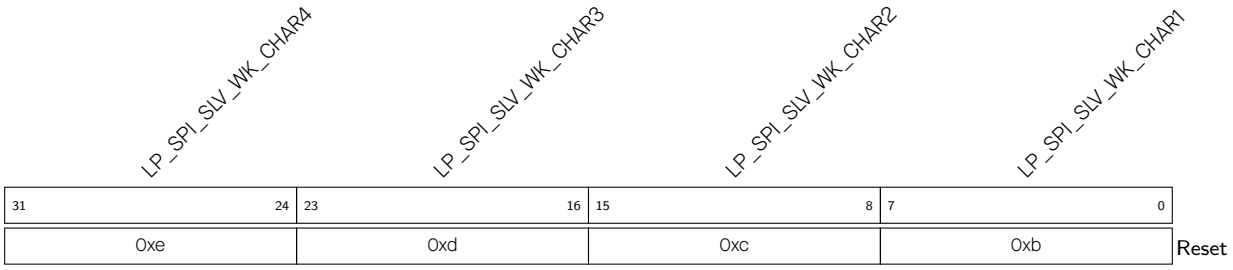
LP_SPI_SLEEP_WK_DATA_SEL 配置需要监测唤醒字符的阶段

0：仅监测 DATA 阶段的 RX 数据

1：监测所有阶段的 RX 数据

(R/W)

Register 46.87. LP_SPI_SLEEP_CONF1_REG (0x0048)



LP_SPI_SLV_WK_CHAR1 配置唤醒字符 1。
(R/W)

LP_SPI_SLV_WK_CHAR2 配置唤醒字符 2。
(R/W)

LP_SPI_SLV_WK_CHAR3 配置唤醒字符 3。
(R/W)

LP_SPI_SLV_WK_CHAR4 配置唤醒字符 4。
(R/W)

Register 46.88. LP_SPI_SLAVE_REG (0x00E0)

(reserved)	LP_SPI_SOFT_RESET	LP_SPI_SLAVE_MODE	(reserved)	LP_SPI_SLV_WRBUF_BITLEN_EN	LP_SPI_SLV_RDBUF_BITLEN_EN	(reserved)	LP_SPI_RSCK_DATA_OUT	LP_SPI_CLK_MODE_13	LP_SPI_CLK_MODE	
31 28	27	26	25 12	11	10	9 4	3	2	1 0	
0 0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0	0	0	0	Reset

LP_SPI_CLK_MODE 配置 LP-SPI 时钟模式。

0：CS 信号无效时，LP-SPI 时钟关闭

1：CS 信号无效后，LP-SPI 时钟延迟一个时钟周期

2：CS 信号无效后，LP-SPI 时钟延迟两个时钟周期

3：LP-SPI 时钟一直有效

(R/W)

LP_SPI_CLK_MODE_13 配置时钟模式。

0：支持 SPI 时钟模式 0 或 2，见表 46.7-2。

1：支持 SPI 时钟模式 1 或 3，见表 46.7-2。

(R/W)

LP_SPI_RSCK_DATA_OUT 配置输出数据的时钟沿。

0：在 TSCK 上升沿输出数据

1：在 RSCK 上升沿输出数据

(R/W)

LP_SPI_SLV_RDBUF_BITLEN_EN 配置在 CPU 控制的 Rd_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

LP_SPI_SLV_WRBUF_BITLEN_EN 配置在 CPU 控制的 Wr_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

LP_SPI_SLAVE_MODE 配置 SPI 工作模式。

0：主机

1：从机

(R/W)

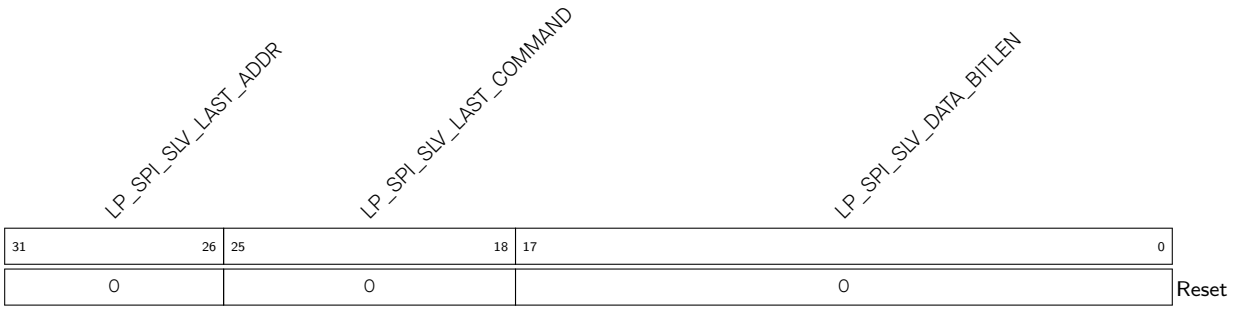
LP_SPI_SOFT_RESET 配置是否软件复位 SPI 时钟线、CS 线和数据线。

0：不复位

1：复位

(WT)

Register 46.89. LP_SPI_SLAVE1_REG (0x00E4)

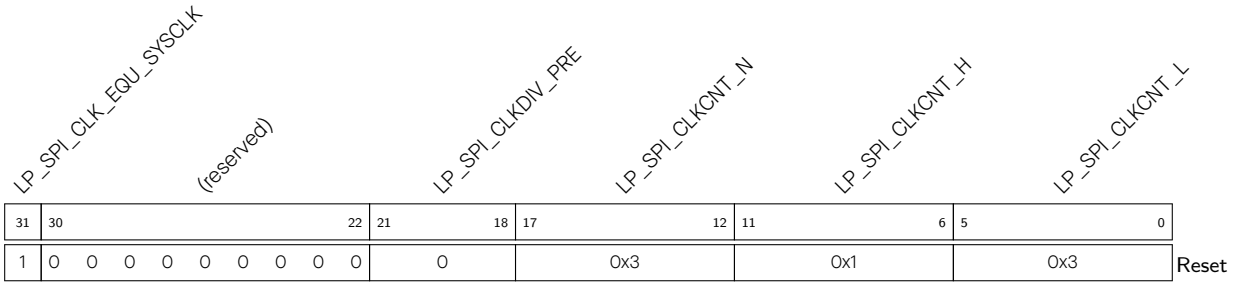


LP_SPI_SLV_DATA_BITLEN 配置在 SPI 从机全双工和半双工传输中，传输的数据位长。(R/W/SS)

LP_SPI_SLV_LAST_COMMAND 配置从机的命令值。(R/W/SS)

LP_SPI_SLV_LAST_ADDR 配置从机的地址值。(R/W/SS)

Register 46.90. LP_SPI_CLOCK_REG (0x000C)



LP_SPI_CLKCNT_L 配置时钟占空比，与 LP_SPI_CLKCNT_H 和 LP_SPI_CLKCNT_N 配合使用。

用作主机时，必须与 LP_SPI_CLKCNT_N 相等。

在从机模式下，必须为 0。

(R/W)

LP_SPI_CLKCNT_H 配置用作主机时 LP_SPI_CLK（高电平）的占空比。

建议将此值配置为 floor((LP_SPI_CLKCNT_N + 1)/2 - 1)。floor() 表示向下取整值，例如 floor(2.2) = 2。

用作从机时，必须配置为 0。

(R/W)

LP_SPI_CLKCNT_N 配置用作主机时 LP_SPI_CLK 的分频系数。

LP_SPI_CLK 频率为 $f_{lp_spi_pclk}$/(LP_SPI_CLKDIV_PRE + 1)/(LP_SPI_CLKCNT_N + 1)。

(R/W)

LP_SPI_CLKDIV_PRE 配置用作主机时 LP_SPI_CLK 的预分频系数。

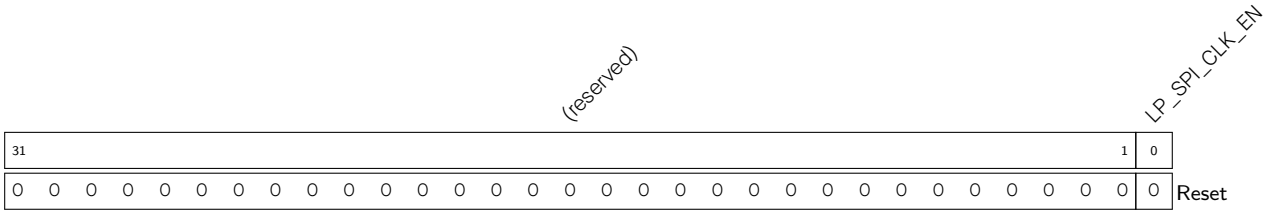
(R/W)

LP_SPI_CLK_EQU_SYSCLK 配置用作主机时 LP_SPI_CLK 频率是否与 lp_spi_pclk 频率相同。0：LP_SPI_CLK 为 lp_spi_pclk 的分频时钟

1：LP_SPI_CLK 与 lp_spi_pclk 频率相同

(R/W)

Register 46.91. LP_SPI_CLK_GATE_REG (0x00E8)



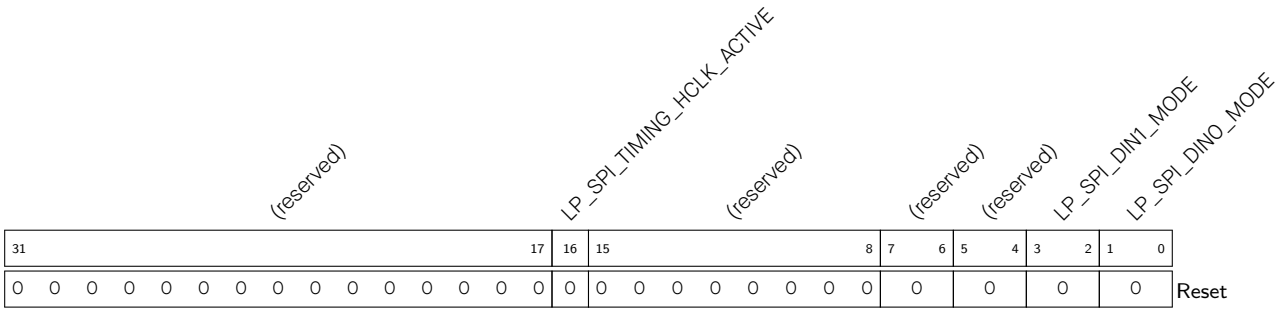
LP_SPI_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能

(R/W)

Register 46.92. LP_SPI_DIN_MODE_REG (0x0024)



LP_SPI_DIN0_MODE 配置 LP_SPI_D 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (LP_SPI_DIN0_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (LP_SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (LP_SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

(R/W)

LP_SPI_DIN1_MODE 配置 LP_SPI_Q 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (LP_SPI_DIN1_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (LP_SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (LP_SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

(R/W)

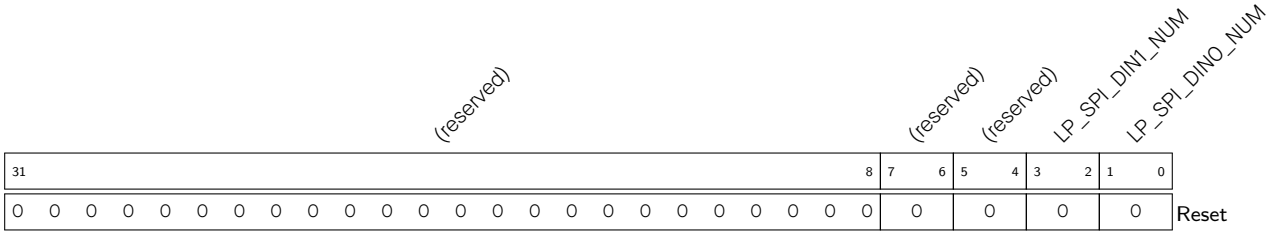
LP_SPI_TIMING_HCLK_ACTIVE 配置是否使能 LP-SPI 输入信号时序模块的高频时钟 HCLK。

0：不使能

1：使能

(R/W)

Register 46.93. LP_SPI_DIN_NUM_REG (0x0028)



LP_SPI_DIN0_NUM 配置输入信号 LP_SPI_D 的延迟周期数。具体的延迟模式由 LP_SPI_DIN0_MODE 配置。

0：延迟 1 个时钟周期

1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

LP_SPI_DIN1_NUM 配置输入信号 LP_SPI_Q 的延迟周期数。具体的延迟模式由 LP_SPI_DIN1_MODE 配置。

0：延迟 1 个时钟周期

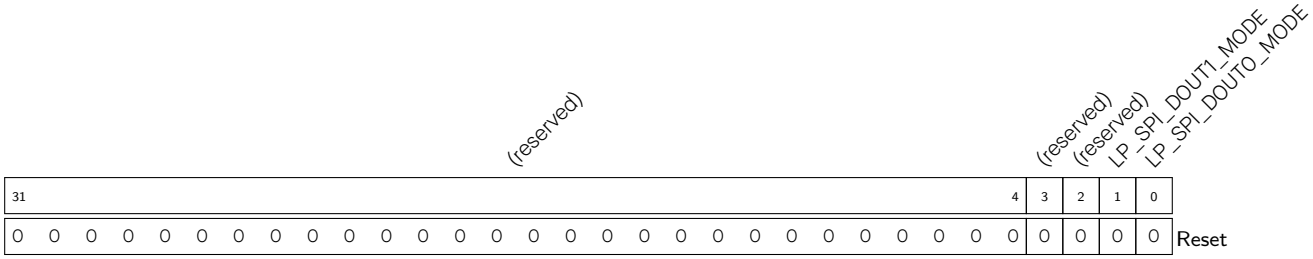
1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

Register 46.94. LP_SPI_DOUT_MODE_REG (0x002C)



LP_SPI_DOUT0_MODE 配置 LP_SPI_D 信号的输出模式。

0：无输出延迟

1：在 LP-SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

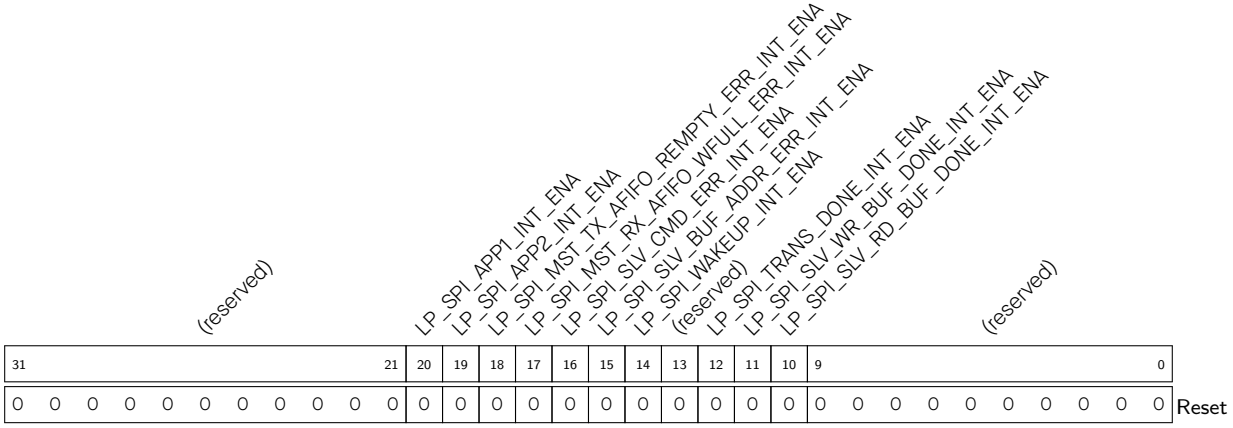
LP_SPI_DOUT1_MODE 配置 LP_SPI_Q 信号的输出模式。

0：无输出延迟

1：在 LP-SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

Register 46.95. LP_SPI_DMA_INT_ENA_REG (0x0034)



LP_SPI_SLV_RD_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_BUF_DONE_INT 中断。
(R/W)

LP_SPI_SLV_WR_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_BUF_DONE_INT 中断。
(R/W)

LP_SPI_TRANS_DONE_INT_ENA 写 1 使能 SPI_TRANS_DONE_INT 中断。
(R/W)

LP_SPI_WAKEUP_INT_ENA 写 1 使能 LP_SPI_WAKEUP_INT 中断。
(R/W)

LP_SPI_SLV_BUF_ADDR_ERR_INT_ENA 写 1 使能 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(R/W)

LP_SPI_SLV_CMD_ERR_INT_ENA 写 1 使能 SPI_SLV_CMD_ERR_INT 中断。
(R/W)

LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA 写 1 使能 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(R/W)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA 写 1 使能 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(R/W)

LP_SPI_APP2_INT_ENA 写 1 使能 SPI_APP2_INT 中断。
(R/W)

LP_SPI_APP1_INT_ENA 写 1 使能 SPI_APP1_INT 中断。
(R/W)

Register 46.96. LP_SPI_DMA_INT_CLR_REG (0x0038)

Bits	31–21	20	19	18	17	16	15	14	13	12	11	10	9–0
Field	(reserved)	LP_SPI_APP1_INT_CLR	LP_SPI_APP2_INT_CLR	LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR	LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR	LP_SPI_SLV_CMD_ERR_INT_CLR	LP_SPI_SLV_BUF_ADDR_ERR_INT_CLR	LP_SPI_WAKEUP_INT_CLR	(reserved)	LP_SPI_TRANS_DONE_INT_CLR	LP_SPI_SLV_WR_BUF_DONE_INT_CLR	LP_SPI_SLV_RD_BUF_DONE_INT_CLR	(reserved)
Reset	0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0

LP_SPI_SLV_RD_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_BUF_DONE_INT 中断。
(WT)

LP_SPI_SLV_WR_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_BUF_DONE_INT 中断。
(WT)

LP_SPI_TRANS_DONE_INT_CLR 写 1 清除 SPI_TRANS_DONE_INT 中断。
(WT)

LP_SPI_WAKEUP_INT_CLR 写 1 清除 LP_SPI_WAKEUP_INT 中断。
(WT)

LP_SPI_SLV_BUF_ADDR_ERR_INT_CLR 写 1 清除 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(WT)

LP_SPI_SLV_CMD_ERR_INT_CLR 写 1 清除 SPI_SLV_CMD_ERR_INT 中断。
(WT)

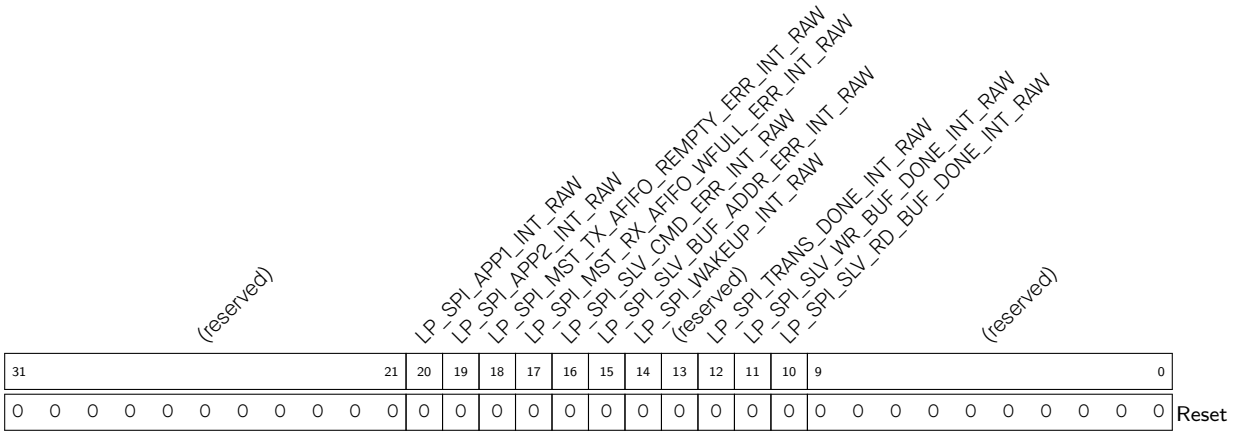
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR 写 1 清除 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(WT)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR 写 1 清除 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(WT)

LP_SPI_APP2_INT_CLR 写 1 清除 SPI_APP2_INT 中断。
(WT)

LP_SPI_APP1_INT_CLR 写 1 清除 SPI_APP1_INT 中断。
(WT)

Register 46.97. LP_SPI_DMA_INT_RAW_REG (0x003C)



LP_SPI_SLV_RD_BUF_DONE_INT_RAW SPI_SLV_RD_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

LP_SPI_SLV_WR_BUF_DONE_INT_RAW SPI_SLV_WR_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

LP_SPI_TRANS_DONE_INT_RAW SPI_TRANS_DONE_INT 的原始中断状态。
(R/WTC/SS)

LP_SPI_WAKEUP_INT_RAW LP_SPI_LP SPI_WAKEUP_INT 的原始中断状态。
(R/WTC/SS)

LP_SPI_SLV_BUF_ADDR_ERR_INT_RAW SPI_SLV_BUF_ADDR_ERR_INT 的原始中断状态。
(R/WTC/SS)

LP_SPI_SLV_CMD_ERR_INT_RAW SPI_SLV_CMD_ERR_INT 的原始中断状态。
(R/WTC/SS)

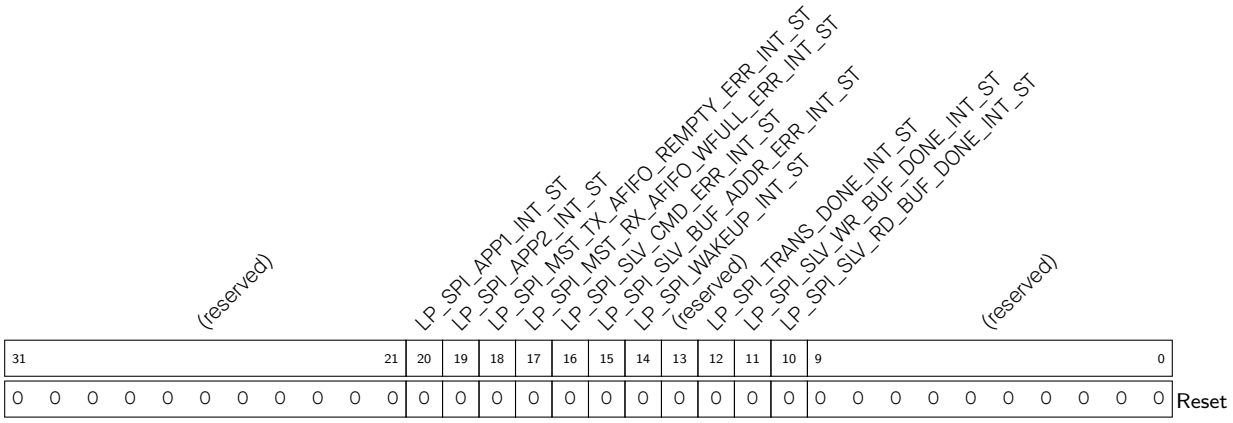
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW SPI_MST_RX_AFIFO_WFULL_ERR_INT 的原始中断状态。
(R/WTC/SS)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的原始中断状态。
(R/WTC/SS)

LP_SPI_APP2_INT_RAW SPI_APP2_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

LP_SPI_APP1_INT_RAW SPI_APP1_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

Register 46.98. LP_SPI_DMA_INT_ST_REG (0x0040)



LP_SPI_SLV_RD_BUF_DONE_INT_ST SPI_SLV_RD_BUF_DONE_INT 的中断状态位。(RO)

LP_SPI_SLV_WR_BUF_DONE_INT_ST SPI_SLV_WR_BUF_DONE_INT 的中断状态位。(RO)

LP_SPI_TRANS_DONE_INT_ST SPI_TRANS_DONE_INT 的中断状态位。(RO)

LP_SPI_WAKEUP_INT_ST LP_SPI_WAKEUP_INT 的中断状态位。(RO)

LP_SPI_SLV_BUF_ADDR_ERR_INT_ST SPI_SLV_BUF_ADDR_ERR_INT 的中断状态位。(RO)

LP_SPI_SLV_CMD_ERR_INT_ST SPI_SLV_CMD_ERR_INT 的中断状态位。(RO)

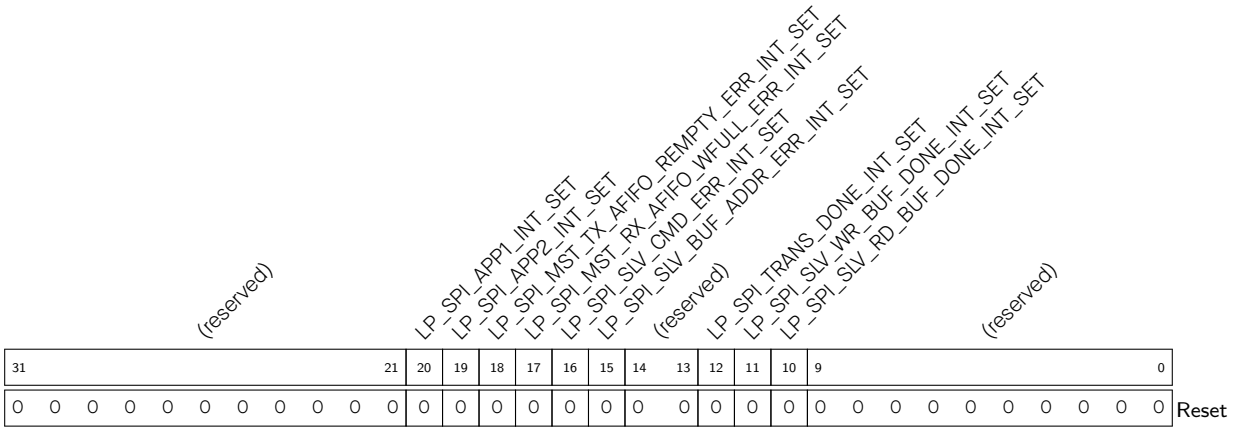
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断状态位。(RO)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断状态位。(RO)

LP_SPI_APP2_INT_ST SPI_APP2_INT 的中断状态位。(RO)

LP_SPI_APP1_INT_ST SPI_APP1_INT 的中断状态位。(RO)

Register 46.99. LP_SPI_DMA_INT_SET_REG (0x004C)



LP_SPI_SLV_RD_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_RD_BUF_DONE_INT 中断。
(WT)

LP_SPI_SLV_WR_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_WR_BUF_DONE_INT 中断。
(WT)

LP_SPI_TRANS_DONE_INT_SET 写 1 置位 SPI_TRANS_DONE_INT 中断。
(WT)

LP_SPI_SLV_BUF_ADDR_ERR_INT_SET 写 1 置位 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(WT)

LP_SPI_SLV_CMD_ERR_INT_SET 写 1 置位 SPI_SLV_CMD_ERR_INT 中断。
(WT)

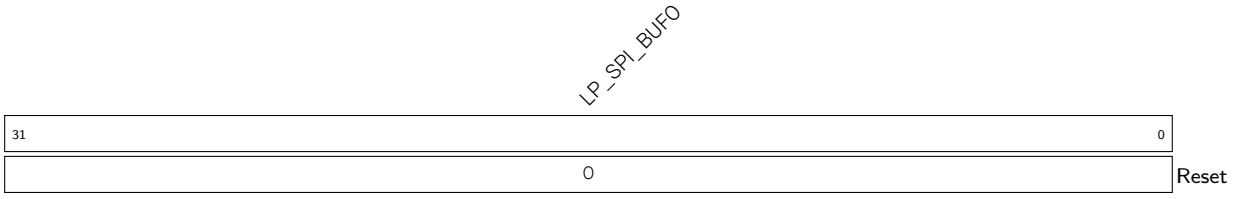
LP_SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET 写 1 置位 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(WT)

LP_SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET 写 1 置位 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(WT)

LP_SPI_APP2_INT_SET 写 1 置位 SPI_APP2_INT 中断。
(WT)

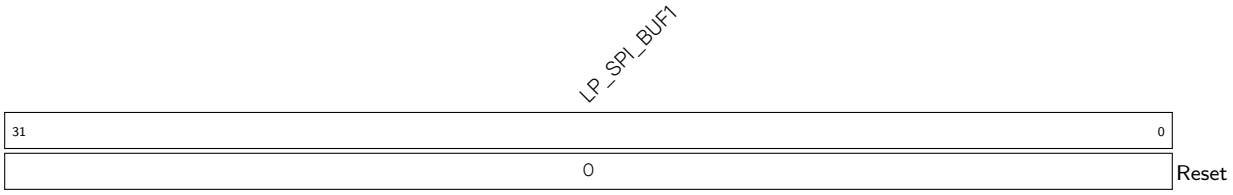
LP_SPI_APP1_INT_SET 写 1 置位 SPI_APP1_INT 中断。
(WT)

Register 46.100. LP_SPI_W0_REG (0x0098)



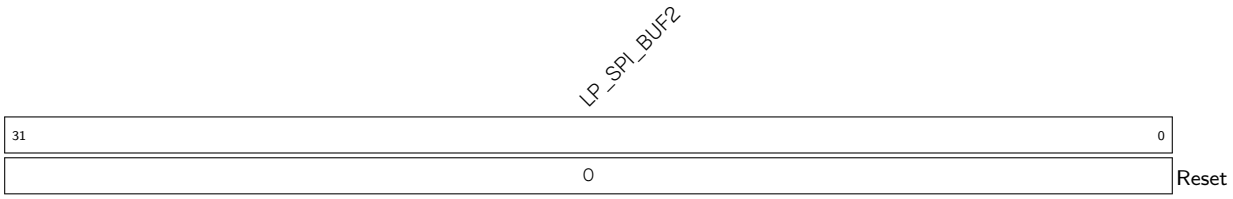
LP_SPI_BUF0 数据 buffer 0，32 位。
(R/W/SS)

Register 46.101. LP_SPI_W1_REG (0x009C)



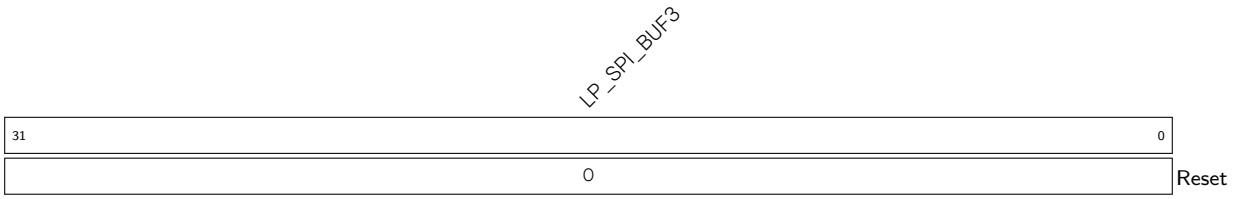
LP_SPI_BUF1 数据 buffer 1，32 位。
(R/W/SS)

Register 46.102. LP_SPI_W2_REG (0x00A0)



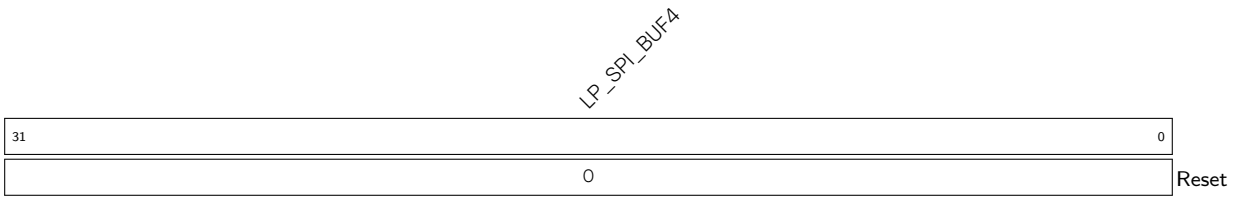
LP_SPI_BUF2 数据 buffer 2，32 位。
(R/W/SS)

Register 46.103. LP_SPI_W3_REG (0x00A4)



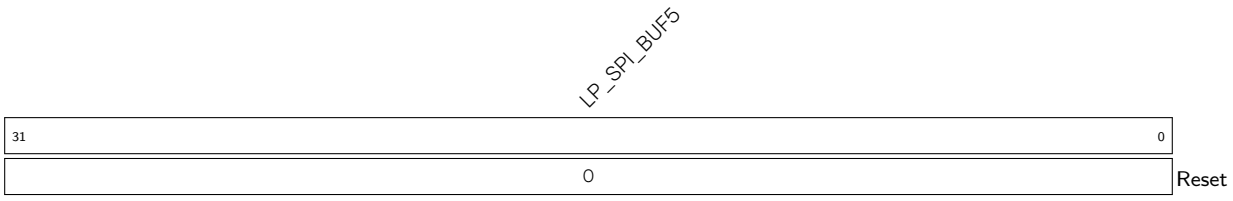
LP_SPI_BUF3 数据 buffer 3，32 位。
(R/W/SS)

Register 46.104. LP_SPI_W4_REG (0x00A8)



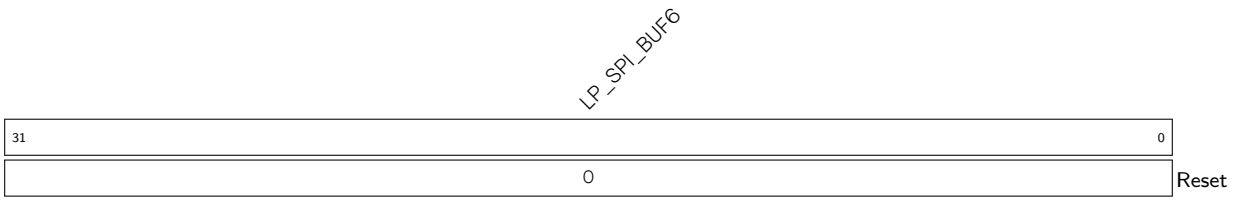
LP_SPI_BUF4 数据 buffer 4，32 位。
(R/W/SS)

Register 46.105. LP_SPI_W5_REG (0x00AC)



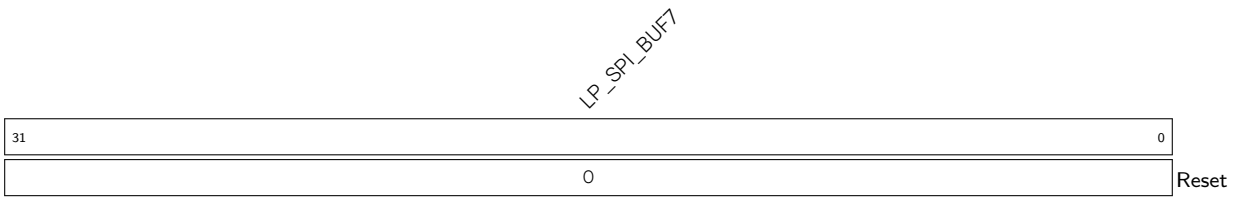
LP_SPI_BUF5 数据 buffer 5，32 位。
(R/W/SS)

Register 46.106. LP_SPI_W6_REG (0x00B0)



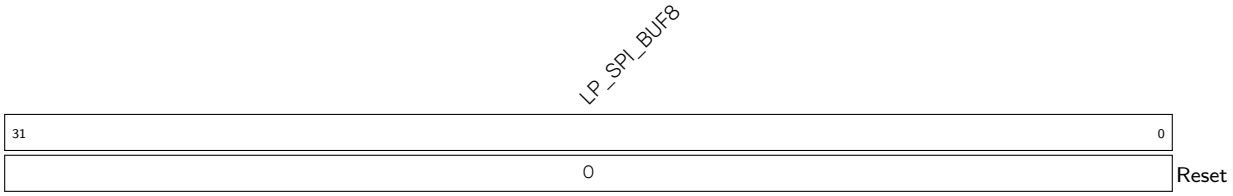
LP_SPI_BUF6 数据 buffer 6，32 位。
(R/W/SS)

Register 46.107. LP_SPI_W7_REG (0x00B4)



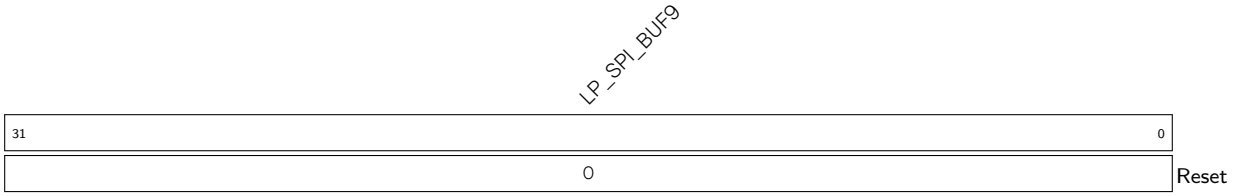
LP_SPI_BUF7 数据 buffer 7，32 位。
(R/W/SS)

Register 46.108. LP_SPI_W8_REG (0x00B8)



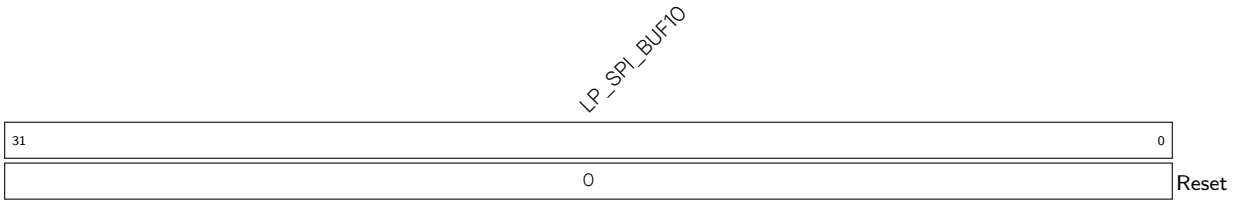
LP_SPI_BUF8 数据 buffer 8，32 位。
(R/W/SS)

Register 46.109. LP_SPI_W9_REG (0x00BC)



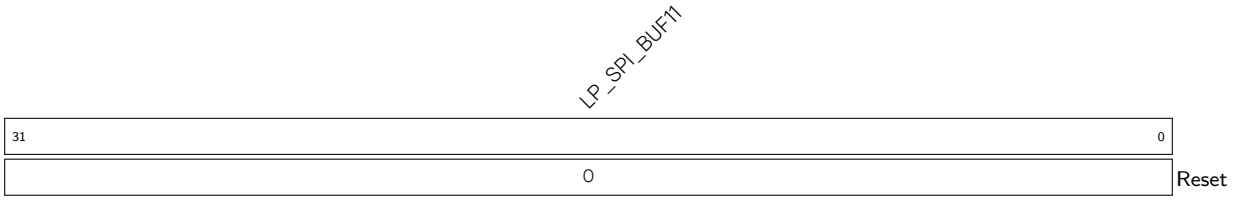
LP_SPI_BUF9 数据 buffer 9，32 位。
(R/W/SS)

Register 46.110. LP_SPI_W10_REG (0x00C0)



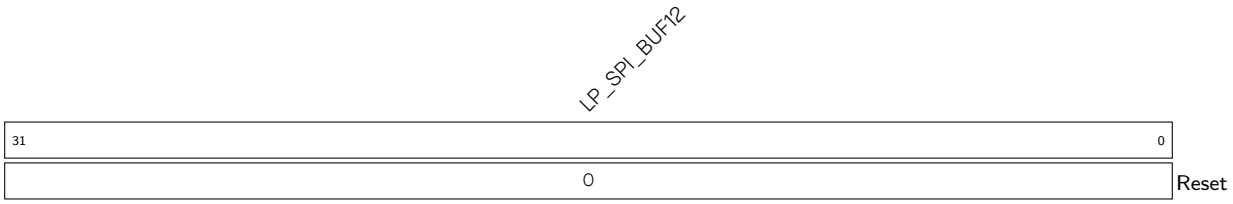
LP_SPI_BUF10 数据 buffer 10，32 位。
(R/W/SS)

Register 46.111. LP_SPI_W11_REG (0x00C4)



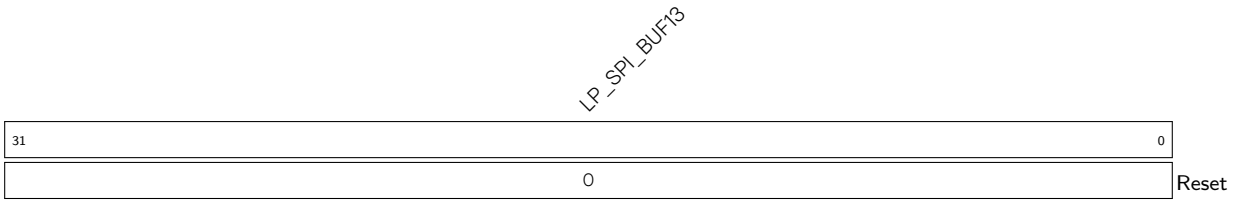
LP_SPI_BUF11 数据 buffer 11，32 位。
(R/W/SS)

Register 46.112. LP_SPI_W12_REG (0x00C8)



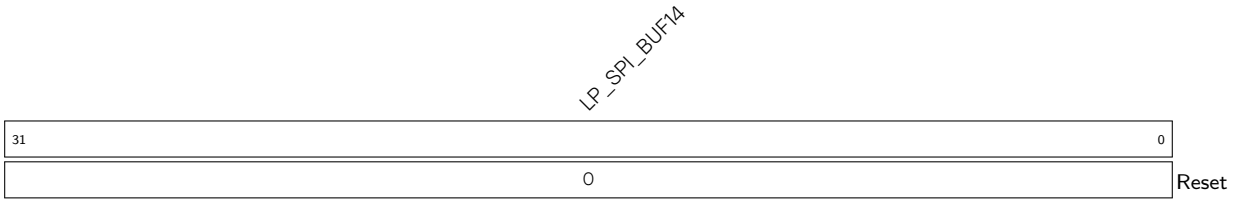
LP_SPI_BUF12 数据 buffer 12，32 位。
(R/W/SS)

Register 46.113. LP_SPI_W13_REG (0x00CC)



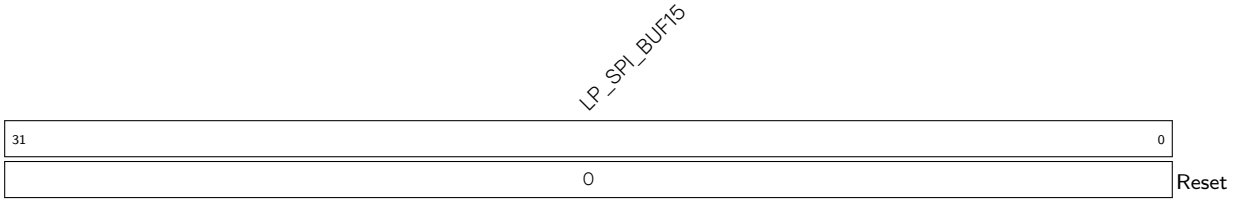
LP_SPI_BUF13 数据 buffer 13，32 位。
(R/W/SS)

Register 46.114. LP_SPI_W14_REG (0x00D0)



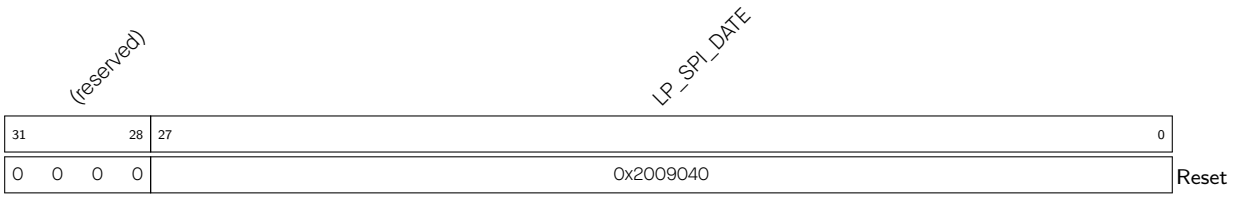
LP_SPI_BUF14 数据 buffer 14，32 位。
(R/W/SS)

Register 46.115. LP_SPI_W15_REG (0x00D4)



LP_SPI_BUF15 数据 buffer 15，32 位。
(R/W/SS)

Register 46.116. LP_SPI_DATE_REG (0x00F0)



LP_SPI_DATE 版本控制寄存器。

(R/W)

第 47 章

I2C 控制器 (I2C)

I2C (Inter-Integrated Circuit) 总线用于使 ESP32-P4 和多个外部设备进行通信。多个外部设备可以共用一个 I2C 总线。ESP32-P4 在主系统有两个 I2C 控制器，在低功耗系统有一个 I2C 控制器。其中，主系统中的 I2C 控制器既可作为主机又可作为从机（下文以 I2C 指代），低功耗系统中的 I2C 控制器则只可作为主机使用，在主系统休眠时仍能工作（下文以 LP_I2C 指代）。

47.1 概述

I2C 是一个两线总线，由 SDA 线和 SCL 线构成。这些线设置为漏极开漏 (open-drain) 输出。因此，I2C 总线上可以挂载多个外设，通常是和一个或多个主机以及一个或多个从机。但同一时刻只有一个主机能占用总线访问一个从机。

主机发出开始信号，即主机在 SCL 为高电平时拉低 SDA 线，代表通讯开始。随后主机通过 SCL 线发出 9 个时钟脉冲，前 8 个脉冲用于传输 7 位地址和 1 个读写位。如果从机地址与该 7 位地址一致，那么从机可以通过在第 9 个脉冲拉低 SDA 线来应答。接下来，根据读 / 写标志位，主机和从机之间可以传输更多的数据。根据应答位 (ACK) 的逻辑电平决定是否停止发送数据。在数据传输中，SDA 线仅在 SCL 线为低电平时才发生变化。当主机完成通讯，发送一个停止信号：主机在 SCL 为高电平时，拉高 SDA 线。如果一次通信中主机既有写操作又有读操作，则主机需在读写操作变化前，发送一个重新开始信号、从机地址和读写标志位。重新开始信号不仅用于一次通信中切换方向，也用于切换设备模式（主机或从机模式）。

47.2 主要特性

ESP32-P4 I2C 控制器具有以下几个特点：

- 支持主机模式和从机模式
- 支持多主机和从机通信
- 支持标准模式 (100 Kbit/s)
- 支持快速模式 (400 Kbit/s)
- 支持 7 位以及 10 位地址寻址
- 从机模式下支持拉低 SCL 时钟实现连续数据传输
- 支持可编程数字噪声滤波功能
- 支持从机地址和从机内存或寄存器地址的双寻址模式

47.3 I2C 架构

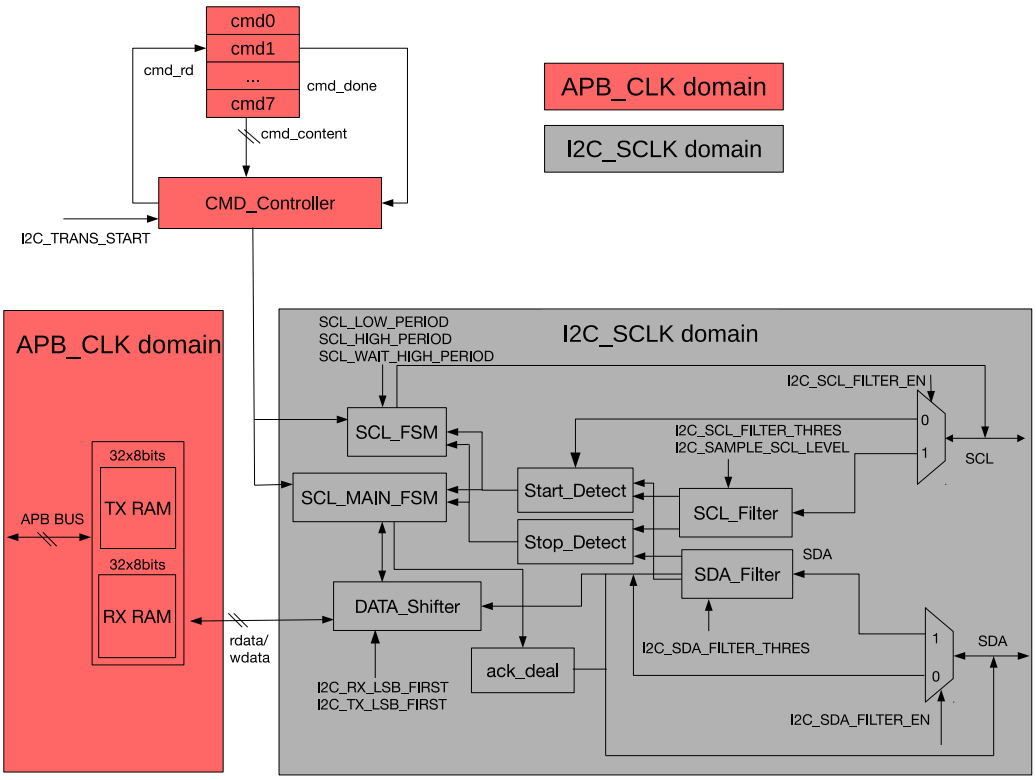


图 47.3-1. I2C 主机基本架构

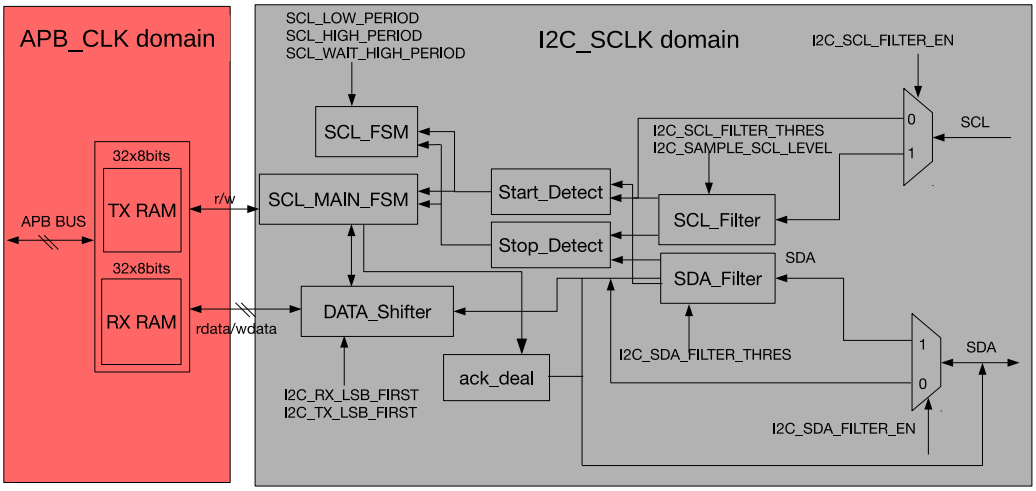


图 47.3-2. I2C 从机基本架构

I2C 控制器可以工作于主机模式或者从机模式，I2C_MS_MODE 寄存器用于模式选择。图 47.3-1 为 I2C 主机基本架构图，图 47.3-2 为 I2C 从机基本架构图。I2C 控制器内部包括的模块主要有：

- 接收和发送存储器 TX/RX RAM：分别用来存储 I2C 要发送和接收到的数据
- 命令控制器 CMD_Controller：产生 (R)START、STOP、WRITE、READ 和 END 指令
- SCL 时钟控制器 SCL_FSM：用来产生满足 I2C 协议的 SCL 时钟。图 47.3-3 和表 47.3-1 是 I2C 协议的时序图和对应的参数表。

- SDA 数据控制器 SCL_MAIN_FSM：用来控制 I2C 指令的执行，和 SDA 线的数据序列，还控制 ack_deal 模块生成 ACK 位并检测 SDA 线上 ACK 位的电平
- 串并转换器 DATA_Shifter：用来完成串行数据和并行数据之间的转换
- SCL 滤波器 SCL_Filter：用于消除 SCL 输入信号上的噪声
- SDA 滤波器 SDA_Filter：用于消除 SDA 输入信号上的噪声
- ACK 位控制器 ack_deal：在 SCL_MAIN_FSM 控制下生成 ACK 位并检测 SDA 线上 ACK 位的电平。

另外，还有产生 I2C 内部时钟的时钟模块，以及在 APB 总线和 I2C 模块之间同步的同步模块。

时钟模块的作用是进行时钟源选择、时钟开关和时钟分频。同步模块用来同步不同时钟域之间信号的传输。

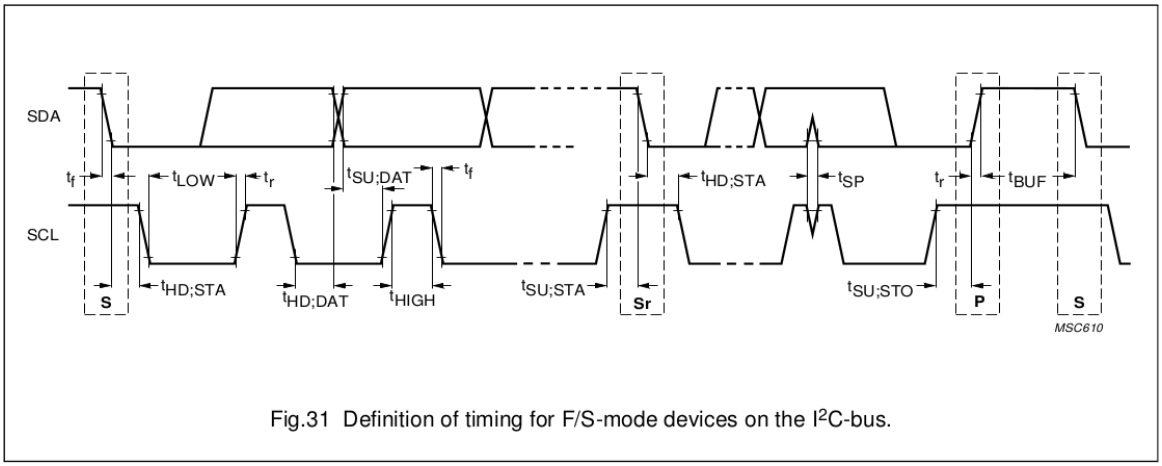


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

图 47.3-3. I2C 协议时序（引自 The I2C-bus specification Version 2.1 Fig.31）

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	20 + 0.1C_b(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	20 + 0.1C_b(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs

表 47.3-1. I2C 时序参数（引自 The I2C-bus specification Version 2.1 Table5）

47.4 功能描述

正如前文介绍，I2C 总线上可以挂载一个或多个主机以及一个或多个从机。下文中介绍的是 ESP32-P4 I2C 控制器的操作方法，请注意 I2C 总线上其他主机或者从机的操作方法可能与 ESP32-P4 I2C 操作方法不同，具体请参考各个 I2C 控制器的技术规格书。

47.4.1 时钟配置

寄存器配置和 TX/RX RAM 部分的时钟域为 APB_CLK。I2C 主要逻辑部分，包括 SCL_FSM、SCL_MAIN_FSM、SCL_FILTER、SDA_FILTER 和 DATA_SHIFTER 都为 I2C_SCLK 时钟域。

用户可以通过 HP_SYS_CLKRST_I2C0_CLK_SRC_SEL，将 HP 系统中 I2C0 的时钟源 I2C_SCLK 配置为 FOSC_20M_CLK 或 XTAL_40M_CLK。HP 系统 I2C1 的时钟源配置寄存器为 HP_SYS_CLKRST_I2C1_CLK_SRC_SEL。对于 LP 系统中 LP_I2C 的时钟源 I2C_SCLK，则可以通过 LPPERI_LP_I2C_CLK_SEL，将其配置为 XTAL_D2_CLK、LP_DYN_FAST_CLK 或 PLL_LP_CLK 8 MHz。

配置 HP I2C0 时钟源的步骤如下：

- HP_SYS_CLKRST_I2C0_CLK_EN 为 1 时，启用 I2C1 I2C_SCLK 时钟源。
- HP_SYS_CLKRST_I2C0_SRC_SEL 为 0 时，配置时钟源为 XTAL_40M_CLK。
- HP_SYS_CLKRST_I2C0_SRC_SEL 为 1 时，配置时钟源为 FOSC_20M_CLK。

配置 HP I2C1 时钟源的步骤如下：

- HP_SYS_CLKRST_I2C1_CLK_EN 为 1 时，启用 I2C1 I2C_SCLK 时钟源。
- HP_SYS_CLKRST_I2C1_SRC_SEL 为 0 时，配置时钟源为 XTAL_40M_CLK。
- HP_SYS_CLKRST_I2C1_SRC_SEL 为 1 时，配置时钟源为 FOSC_20M_CLK。

配置 LP_I2C 时钟源的步骤如下：

- LPPERI_CK_EN_LP_I2C 为 1 时，启用 LP_I2C I2C_SCLK 时钟源。
- LPPERI_LP_I2C_CLK_SEL 为 0 时，配置时钟源为 LP_DYN_FAST_CLK。
- LPPERI_LP_I2C_CLK_SEL 为 1 时，配置时钟源为 XTAL_D2_CLK。
- LPPERI_LP_I2C_CLK_SEL 为 2 时，配置时钟源为 PLL_LP_CLK 8 MHz。

选择后的时钟经过小数分频，得到 I2C 的工作时钟 I2C_SCLK：

$$I2C_SCLK_DIV_NUM + 1 + \frac{I2C_SCLK_DIV_A}{I2C_SCLK_DIV_B}$$

式中 I2C_SCLK_DIV_NUM 代表分频系数的整数部分，I2C_SCLK_DIV_A 代表分频系数小数部分的分子，I2C_SCLK_DIV_B 代表分频系数小数部分的分母。根据时序参数的限制，分频后的 I2C_SCLK 的频率需至少大于 SCL 频率 20 倍。
在 I2C0 中：

- 配置 HP_SYS_CLKRST_I2C0_CLK_DIV_NUM 调整 I2C_SCLK_DIV_NUM
- 配置 HP_SYS_CLKRST_I2C0_CLK_DIV_NUMERATOR 调整 I2C_SCLK_DIV_A
- 配置 HP_SYS_CLKRST_I2C0_CLK_DIV_DENOMINATOR 调整 I2C_SCLK_DIV_B

在 I2C1 中：

- 配置 HP_SYS_CLKRST_I2C1_CLK_DIV_NUM 调整 I2C_SCLK_DIV_NUM
- 配置 HP_SYS_CLKRST_I2C1_CLK_DIV_NUMERATOR 调整 I2C_SCLK_DIV_A
- 配置 HP_SYS_CLKRST_I2C1_CLK_DIV_DENOMINATOR 调整 I2C_SCLK_DIV_B

47.4.2 滤除 SCL 和 SDA 噪声

SCL_Filter 和 SDA_Filter 滤波器模块实现方式相同，用于滤除 SCL 及 SDA 输入信号上的噪声。通过配置 I2C_SCL_FILTER_EN 以及 I2C_SDA_FILTER_EN 寄存器可以开启或关闭滤波器。

以 SCL_Filter 为例，当使能 SCL_Filter 功能时，滤波器会连续采样输入信号 SCL，如果输入信号在连续 I2C_SCL_FILTER_THRES 个 I2C_SCLK 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，SCL_Filter 和 SDA_Filter 滤波器分别会过滤脉冲宽度小于 I2C_SCL_FILTER_THRES 和 I2C_SDA_FILTER_THRES 个 I2C_SCLK 时钟周期的线路毛刺。

47.4.3 SCL 时钟拉伸

从机模式下，可以通过拉低 SCL 线实现时钟拉伸，在此期间数据传输暂停，可以用来给软件足够的时间处理数据。置位 I2C_SLAVE_SCL_STRETCH_EN 位使能 SCL 时钟拉伸功能，配置 I2C_STRETCH_PROTECT_NUM 字段来控制 SCL 拉伸后释放的时长以防出现时序错误。出现以下四种情况时从机可以选择拉低 SCL 线实现时钟拉伸：

1. 地址命中：从机模式下，从机地址与主机发送到 SDA 线上的地址相匹配，且读写标志位为 1。
2. 写满：从机模式下，I2C 控制器的 RX RAM 为满。注意，从机在接收少于 FIFO 深度个字节时（ESP32-P4 I2C 的 FIFO 深度为 32 字节），可以不开启时钟拉伸功能；当要接收大于等于 FIFO 深度个字节时，可以通过 FIFO 阈值中断写 RAM 的乒乓操作，或者开始时钟拉伸功能，给软件提供处理时间。开启时钟拉伸功能时，必须将 I2C_RX_FULL_ACK_LEVEL 置 0，来保证功能正确，否则可能会出现不可预计的后果。
3. 读空：从机模式下，I2C 控制器要发送数据，但 TX RAM 为空。
4. 发送 ACK 时：从机模式下置位 I2C_SLAVE_BYTE_ACK_CTL_EN，从机会在发送 ACK 时拉低 SCL。软件在此阶段进行一些操作，如数据校验，并通过配置 I2C_SLAVE_BYTE_ACK_LVL 控制将要发送的 ACK 的电平高低。要注意的是，当出现从机接收的 RX RAM 满时，要发送的 ACK 电平将由 I2C_RX_FULL_ACK_LEVEL 而不是 I2C_SLAVE_BYTE_ACK_LVL 决定。此时同样需要将 I2C_RX_FULL_ACK_LEVEL 置 0，以保证 SCL 时钟拉伸功能的正常产生。

产生时钟拉伸时，软件可读取 I2C_STRETCH_CAUSE 位获取 SCL 时钟拉伸的原因。置位 I2C_SLAVE_SCL_STRETCH_CLR 位关闭 SCL 时钟拉伸。

47.4.4 SCL 空闲时产生 SCL 脉冲

通常情况下，在 I2C 总线空闲时，SCL 线一直为高。ESP32-P4 I2C 支持在 I2C 主机处于空闲状态时，可编程配置产生 SCL 脉冲的功能。这个功能仅在 I2C 控制器作为主机时有效。置位 I2C_SCL_RST_SLV_EN，硬件会发送 I2C_SCL_RST_SLV_NUM 个 SCL 脉冲，之后 I2C_SCL_RST_SLV_EN 位会自动清零。

47.4.5 同步

I2C 的寄存器配置用 APB 时钟，I2C 主模块用 I2C_SCLK，这之间存在异步处理，需要增加同步的步骤将配置寄存器的值更新进入 I2C 主模块。步骤为先写配置寄存器，再向 I2C_CONF_UPGATE 位写 1。需要通过这种方法更新的配置寄存器详见表 47.4-1。

表 47.4-1. I2C 同步寄存器

配置寄存器	寄存器域	地址
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	I2C_ADDR_10BIT_RW_CHECK_EN	
	I2C_ADDR_BROADCASTING_EN	
	I2C_SDA_FORCE_OUT	
	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	
I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SLAVE_ADDR_REG	I2C_ADDR_10BIT_EN	0x0010
	I2C_SLAVE_ADDR	
I2C_FIFO_CONF_REG	I2C_FIFO_ADDR_CFG_EN	0x0018
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_STRETCH_CONF_REG	I2C_SLAVE_BYTE_ACK_CTL_EN	0x0084
	I2C_SLAVE_BYTE_ACK_LVL	
	I2C_SLAVE_SCL_STRETCH_EN	
	I2C_STRETCH_PROTECT_NUM	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038
	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	

	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

47.4.6 漏级开路输出

SCL 及 SDA 线采用漏级开路的驱动方式。I2C 控制器有两种配置方式实现漏级开路驱动方式：

1. 置位 I2C_SCL_FORCE_OUT、I2C_SDA_FORCE_OUT 并配置相应 SCL 及 SDA PAD 的 GPIO_PIN*n*_PAD_DRIVER 寄存器为漏级开路驱动。
2. 清零 I2C_SCL_FORCE_OUT 以及 I2C_SDA_FORCE_OUT。

SCL 和 SDA 配置成开漏方式时，从低电平转向高电平的时间会较长，这个转变时间由线上的上拉电阻以及电容共同决定。开漏模式下，I2C 输出频率的占空比受限于 SCL 上拉速度，主要受 SCL 的速度限制。

另外，在 I2C_SCL_FORCE_OUT 和 I2C_SCL_PD_EN 置 1 时，可以强制拉低 SCL 线；在 I2C_SDA_FORCE_OUT 和 I2C_SDA_PD_EN 置 1 时，可以强制拉低 SDA 线。

47.4.7 时序参数配置

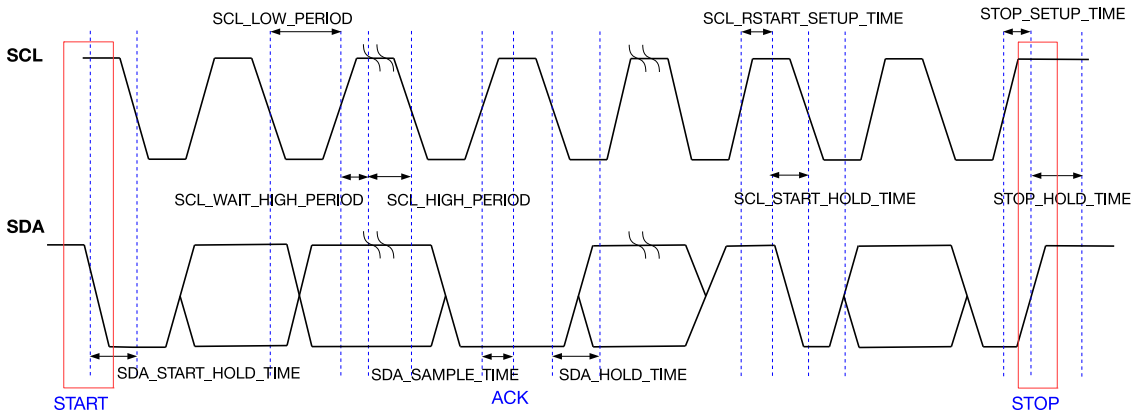


图 47.4-1. I2C 时序图

图 47.4-1 为实现 I2C 协议的 I2C 主机的时序图，图中的寄存器均用来配置时序参数。I2C 控制器的 START 位、STOP 位、数据保持时间、数据采样时间、SCL 上升沿等待时间等时序均可以通过图 47.4-1 中所示的寄存器进行配置。这些寄存器以模块时钟 (I2C_SCLK) 为单位，与各时序参数的对应关系为：

1. $t_{LOW} = (I2C_SCL_LOW_PERIOD + 1) \cdot T_{I2C_SCLK}$
2. $t_{HIGH} = (I2C_SCL_HIGH_PERIOD + 1) \cdot T_{I2C_SCLK}$
3. $t_{SU:STA} = (I2C_SCL_RSTART_SETUP_TIME + 1) \cdot T_{I2C_SCLK}$
4. $t_{HD:STA} = (I2C_SCL_START_HOLD_TIME + 1) \cdot T_{I2C_SCLK}$
5. $t_{r} = (I2C_SCL_WAIT_HIGH_PERIOD + 1) \cdot T_{I2C_SCLK}$
6. $t_{SU:STO} = (I2C_SCL_STOP_SETUP_TIME + 1) \cdot T_{I2C_SCLK}$
7. $t_{BUF} = (I2C_SCL_STOP_HOLD_TIME + 1) \cdot T_{I2C_SCLK}$
8. $t_{HD:DAT} = (I2C_SDA_HOLD_TIME + 1) \cdot T_{I2C_SCLK}$

9. $t_{SU:DAT} = (I2C_SCL_LOW_PERIOD - I2C_SDA_HOLD_TIME) \cdot T_{I2C_SCLK}$

根据在何种模式下有意义，下列时序寄存器可分为两组：

- 主机模式：
  1. I2C_SCL_START_HOLD_TIME：生成 I2C 协议中的 start 位时，SDA 信号拉低到 SCL 信号拉低的时间间隔。该时间间隔为（I2C_SCL_START_HOLD_TIME +1）个模块时钟周期。仅控制器工作在主机模式时有意义。
  2. I2C_SCL_LOW_PERIOD：SCL 低电平持续时间。SCL 低电平时间为（I2C_SCL_LOW_PERIOD + 1）个模块时钟周期。该寄存器仅在控制器工作在主机模式时有意义。

     SCL 低电平时间会在以下情况下延长：
     - 当 I2C 为主机时，外设拉低 SCL
     - I2C 控制器执行 END 命令拉低 SCL
     - I2C 为从机时，发生 SCL 时钟拉伸
  3. I2C_SCL_WAIT_HIGH_PERIOD：等待 SCL 线拉高的模块时钟周期数。请确保在该时间内 SCL 线可以完成上拉。否则会导致 SCL 高电平持续时间不可预测。仅控制器工作在主机模式时有意义。
  4. I2C_SCL_HIGH_PERIOD：SCL 线拉高后维持高电平的模块时钟周期数。仅控制器工作在主机模式时有意义。当 SCL 线在 I2C_SCL_WAIT_HIGH_PERIOD + 1 个模块时钟内完成上拉，则 SCL 线的频率为：$f_{scl} = \frac{f_{I2C_SCLK}}{I2C_SCL_LOW_PERIOD + I2C_SCL_HIGH_PERIOD + I2C_SCL_WAIT_HIGH_PERIOD + 3 + I2C_SCL_FILTER_THRES}$

     其中 3 个时钟周期是对 SCL 进行同步所需的时间，如果打开了 SCL 滤波功能，则还需要加上滤波带来的延迟 I2C_SCL_FILTER_THRES。用 I2C_SCL_WAIT_HIGH_PERIOD + 1 个模块时钟模拟的 SCL 上拉时间受到上拉电阻、IO 驱动能力、SCL 线上电容等等的影响，可能出现测试的实际频率和理论频率有偏差的情况。此时可以通过调整 I2C_SCL_WAIT_HIGH_PERIOD 的值来使实际频率和理论频率接近。
- 主机模式和从机模式：
  1. I2C_SDA_SAMPLE_TIME：SCL 上升沿到采样 SDA 线电平值的时间间隔。推荐设置在 SCL 高电平持续时间的中间值，以保证能够正确采样到 SDA 线上电平。控制器工作在主机模式及从机模式时都有意义。
  2. I2C_SDA_HOLD_TIME：SDA 输出数据变化与 SCL 下降沿的时间间隔。控制器工作在主机模式及从机模式时都有意义。

根据时序参数的限制，对时序寄存器的配置范围也有约束。

1. $\frac{f_{I2C_SCLK}}{f_{SCL}} > 20$
2. $3 \times f_{I2C_SCLK} \leq (I2C_SDA_HOLD_TIME - 4) \times f_{APB_CLK}$
3. I2C_SDA_HOLD_TIME + I2C_SCL_START_HOLD_TIME > SDA_FILTER_THRES + 3
4. I2C_SCL_WAIT_HIGH_PERIOD < I2C_SDA_SAMPLE_TIME < I2C_SCL_HIGH_PERIOD
5. I2C_SDA_SAMPLE_TIME < I2C_SCL_WAIT_HIGH_PERIOD + I2C_SCL_START_HOLD_TIME + I2C_SCL_RSTART_SETUP_TIME
6. I2C_STRETCH_PROTECT_NUM + I2C_SDA_HOLD_TIME > I2C_SCL_LOW_PERIOD

47.4.8 超时控制

I2C 内部有三种超时控制，分别是对 SCL_FSM 状态的超时控制、SCL_MAIN_FSM 状态的超时控制和对 SCL 线状态的超时控制。其中前两种是一直使能的，第三种是可编程配置的。

当 SCL_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C_SCL_ST_TO_I2C+1}-1$ 个时钟周期后，会触发 I2C_SCL_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_ST_TO_I2C 的配置值最大为 23，即最迟在等待超过 $2^{24}-1$ 个 I2C_SCLK 时钟周期后会产生超时中断。

当 SCL_MAIN_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C_SCL_MAIN_ST_TO_I2C+1}-1$ 个 I2C_SCLK 时钟周期后，会触发 I2C_SCL_MAIN_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_MAIN_ST_TO_I2C 的配置值最大为 23，即最迟在等待超过 $2^{24}-1$ 个时钟周期后会产生超时中断。

使能 I2C_TIME_OUT_EN 打开 SCL 线状态的超时控制。当 SCL 线状态长时间维持同一电平不变，且时间超过 $2^{I2C_TIME_OUT_VALUE+1}-1$ 个 I2C_SCLK 时钟周期后，会触发 I2C_TIME_OUT_INT 中断，I2C 总线回到空闲状态。

47.4.9 指令配置

I2C 控制器工作于主机模式时，CMD_Controller 会依次从八个命令寄存器中读出命令并按照命令来控制 SCL_FSM 及 SCL_MAIN_FSM。

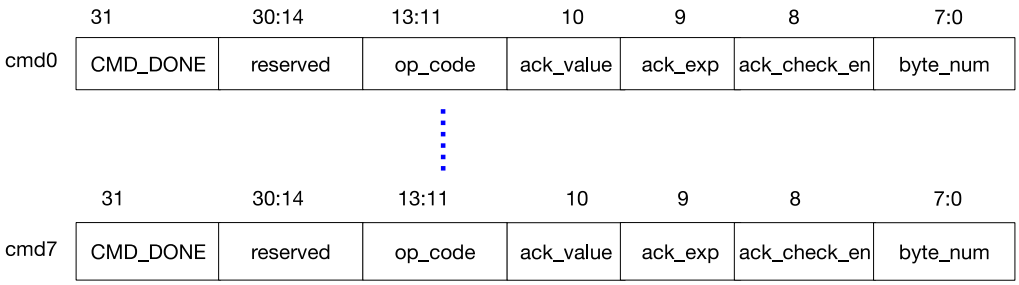


图 47.4-2. I2C 命令寄存器结构

命令寄存器只在 I2C 控制器工作于主机模式时才有效，其内部结构如图 47.4-2 所示。命令寄存器的参数为:

1. CMD_DONE: 命令执行完成标识。每条命令执行完硬件会将对应命令寄存器中的 CMD_DONE 置 1。软件可以通过读取每条命令的 CMD_DONE 位来判断该命令是否执行完毕。每次更新命令时，软件需要将 CMD_DONE 位清零。
2. op_code: 命令编码，共有五种命令:
   - WRITE: op_code 等于 1 时为 WRITE 命令，该命令指示 I2C 控制器向从机发送从机地址、被访问的寄存器地址（仅限双寻址模式）、数据。
   - STOP: op_code 等于 2 时为 STOP 命令，该命令指示 I2C 控制器发送 I2C 协议中的 STOP 位。此条命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件再次启动 CMD_Controller 后，会重新从命令寄存器 0 开始去取指令。
   - READ: op_code 等于 3 时为 READ 命令，该命令指示 I2C 控制器从从机读取数据。
   - END: op_code 等于 4 时为 END 命令，该命令指示 I2C 控制器将 SCL 信号拉低，暂停 I2C 通信。该命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件在更新命令寄存器和

RAM 数据后可重新启动 CMD_Controller，继续进行 I2C 协议传输。再次启动后 CMD_Controller 会重新从命令寄存器 0 开始去取指令。

- RSTART: op_code 等于 6 时为 RSTART 命令，该命令指示 I2C 控制器发送 I2C 协议中的 START 位或 RESTART 位。

3. ack_value: 该位设置读操作时 I2C 控制器在 I2C 协议中的 ACK 位发送的电平值。RSTART、STOP、END、WRITE 命令中该位没有意义。
4. ack_exp: 该位用于设置写操作时 I2C 控制器在 I2C 协议中的 ACK 位期望接收的电平值。RSTART、STOP、END、READ 命令中该位没有意义。
5. ack_check_en: 该位使能写操作中 I2C 控制器检查从机发送的 ACK 位电平与命令中的 ack_exp 是否一致。如果接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致时，I2C 主机会产生 I2C_NACK_INT 中断，停止发送数据并产生 STOP。此位为 1 时，检测从机发送的 ACK 位电平；此位为 0 时，不检测从机发送的 ACK 位电平。RSTART、STOP、END、READ 命令中该位没有意义。
6. byte_num: 读写数据的长度（单位字节），最大为 255，最小为 1。RSTART、STOP、END 命令中 byte_num 无意义。

每次命令序列的执行都是从命令寄存器 0 开始，到 STOP 或 END 命令结束。所以需要保证每个命令序列中必须有 STOP 或 END 命令。

一次完整的 I2C 协议传输应该起始于 START 命令，结束于 STOP 命令。可通过 END 命令将一次 I2C 协议传输分为多个命令序列来完成。每个命令序列可以改变数据传输的方向、时钟频率、从机地址和数据长度等。这样可以弥补 RAM 大小不足的问题，也可以实现更灵活的 I2C 通信。

47.4.10 TX/RX RAM 数据存储

TX/RX RAM 大小均为 32 x 8 位。TX RAM 和 RX RAM 均可以通过 FIFO 和直接地址 (non-FIFO) 两种方式访问。将 I2C_NONFIFO_EN 位设置成 0，为 FIFO 方式；I2C_NONFIFO_EN 位设置成 1 时为直接地址方式。

TX RAM 用于存储 I2C 控制器需要发送的数据。在 I2C 通信的过程中，当 I2C 控制器需要发送数据时（不包括 ACK 位响应），会依次读出 TX RAM 中的数据并串行输出到 SDA 线上。当 I2C 控制器工作于主机模式时，所有需要发送给从机的数据都必须按照发送顺序依次存储在 TX RAM 中。包括被访问的从机地址、读写标志位、被访问的寄存器地址（仅限双地址寻址模式下）、写数据。当 I2C 控制器工作于从机模式时，TX RAM 中只存放写数据。

TX RAM 可被 CPU 读写。CPU 可通过两种方式写 TX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 I2C_DATA_REG 写 TX RAM，硬件自动进行 TX RAM 写地址自增。直接地址访问是通过地址段 (I2C 基地址 + 0x100) ~(I2C 基地址 + 0x17C) 直接访问 TX RAM。TX RAM 的每一个字节占据一个字 (word) 的地址。因此，第一个字节访问地址为 I2C 基地址 + 0x100，第二字节访问地址为 I2C 基地址 + 0x104，第三字节访问地址为 I2C 基地址 + 0x108，以此类推。CPU 只可通过直接地址访问方式读 TX RAM，且 CPU 可以通过直接地址读回写入到 TX RAM 的字节。读 TX RAM 的地址和写 TX RAM 的地址相同。

RX RAM 存储的是 I2C 通信过程中，I2C 控制器接收到的数据。当 I2C 控制器工作于从机模式时，主机发送的从机地址及被访问的寄存器地址（仅限双地址寻址模式下）都不会存储在 RX RAM 中。软件可以在 I2C 通信结束后，读出 RX RAM 的值。

RX RAM 只可被 CPU 读。CPU 可通过两种方式读 RX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 I2C_DATA_REG 读 RX RAM，硬件自动完成 RX RAM 读地址自增。直接地址访问是通过地址段 (I2C 基地址 + 0x180) ~(I2C 基地址 + 0x1FC) 直接访问 RX RAM。RX RAM 的每一个字节占据一个字的地址。因此，第一

个字节访问地址为 I2C 基地址 + 0x180，第二字节访问地址为 I2C 基地址 + 0x184，第三字节访问地址为 I2C 基地址 + 0x188，以此类推。

在 FIFO 模式下可以对 TX RAM 进行乒乓操作，来实现发送大于 FIFO 深度的数据。置位 I2C_FIFO_PRT_EN，当 RAM 中剩下的待发送数据字节数小于 I2C_TXFIFO_WM_THRHD 时，会产生 I2C_TXFIFO_WM_INT 中断。软件收到中断后，继续向 I2C_DATA_REG（主机）中写数。需要保证向 TX RAM 写数或更新数据的操作提前于 I2C 发送数据的动作，否则会产生不可预计的后果。

在 FIFO 模式下也可以对 RX RAM 进行乒乓操作，来实现接收大于 FIFO 深度的数据。置位 I2C_FIFO_PRT_EN，将 I2C_RX_FULL_ACK_LEVEL 置 0。当 RAM 中收到的数据字节数大于等于 I2C_RXFIFO_WM_THRHD（从机）时，会产生 I2C_RXFIFO_WM_INT 中断。软件收到中断后，从 I2C_DATA_REG（从机）中读数。

47.4.11 数据转换

DATA_Shifter 模块用于串并转换，当 I2C 发送数据时，将 TX RAM 中的字节数据转化成比特流；当 I2C 接收数据时，将比特流转化成字节数据并存入 RX RAM。I2C_RX_LSB_FIRST 和 I2C_TX_LSB_FIRST 用于配置最高有效位或最低有效位的优先储存或传输。

47.4.12 寻址模式

ESP32-P4 I2C 支持 7 位寻址模式和 10 位寻址模式，7 位寻址和 10 位寻址可结合使用。除此之外，ESP32-P4 I2C 还支持双地址寻址模式。

假设从机地址为 SLV_ADDR。ESP32-P4 I2C 控制器可以使用 7 位寻址（SLV_ADDR[6:0]），也可以使用 10 位寻址（SLV_ADDR[9:0]）。

对于主机模式而言，7 位寻址下只要发送一个字节地址段 SLV_ADDR[6:0] 和读写标志。7 位寻址模式下，有种特殊情况是广播寻址。在从机中，将 I2C_ADDR_BROADCASTING_EN 置 1，开启广播寻址模式。当接收到主机发送的地址为广播地址 (0x00) 且读写标志位为 0 时，此时无论从机自己的地址是多少，都会响应主机。

10 位寻址需要发送两字节地址段。第一个要发送的数是从机地址的第一个 7 位 slave_addr_first_7bits 和读写标志，slave_addr_first_7bits 的值应该配置为 (0x78 | SLV_ADDR[9:8])。第二个要发送的数是 slave_addr_second_byte，slave_addr_second_byte 的值为 SLV_ADDR[7:0]。

在从机中，可以通过配置 I2C_ADDR_10BIT_EN 寄存器开启 10 位寻址模式。I2C_SLAVE_ADDR 用于配置 I2C 从机地址。I2C_SLAVE_ADDR[14:7] 的值应配置为 SLV_ADDR[7:0]，I2C_SLAVE_ADDR[6:0] 的值应配置为 (0x78 | SLV_ADDR[9:8])。由于 10 位从机地址比 7 位地址多一个字节，所以 WRITE 命令对应的 byte_num 以及 RAM 中数据数量都相应增加 1。详细说明可参考 编程示例。

控制器处于从机模式时，还支持双地址寻址模式。双地址的第一个地址是 I2C 从机地址，第二个地址是 I2C 从机的内存地址。双地址模式下，RAM 必须采用 non-FIFO 方式访问。通过置位 I2C_FIFO_ADDR_CFG_EN 来使能双地址访问功能。当从机接收到的从机地址和内部配置的从机地址不一致时，会产生 I2C_SLAVE_ADDR_UNMATCH 中断。

47.4.13 10 位寻址的读写标志位检查

在 10 位寻址模式下，将 I2C_ADDR_10BIT_RW_CHECK_EN 置 1，会对发送的第一个数 slave_addr_first_7bits 和读写标志做检查。当读写标志不是写，此时与协议不符合，会结束传输。若不打开此功能，当读写标志不是写，还能支持继续传输，但可能出现传输错误。

47.4.14 启动控制器

对于主机，配置成主机模式和命令寄存器等相关配置后，通过向 I2C_TRANS_START 写 1，启动主机解析，执行命令序列。一组命令总是从命令寄存器 0 开始，顺序执行到 STOP 或者 END 命令。当另一组命令需要从命令寄存器 0 开始执行时，需要重新向 I2C_TRANS_START 写 1 来更新命令。

对于从机，有两种启动方式：

- **自动启动传输：** 置位 I2C_SLV_TX_AUTO_START_EN，则从机会在被主机寻址后自动启动传输。此模式适用于主机需要持续访问从机数据的场景。
- **手动启动传输：** 清零 I2C_SLV_TX_AUTO_START_EN，且在每次接收传输前必须显式置位 I2C_TRANS_START。此模式适用于主机需要选择性读取从机数据的场景。

  例如：

  1. 从机准备 5 字节数据；
  2. 主机仅读取前 3 字节；
  3. 主机读取完成后，从机需置位 I2C_TX_FIFO_RST 清除 TXFIFO 中的剩余数据，随后置位 I2C_TRANS_START 重新初始化从机状态。

注意： 在手动启动模式下，必须在下次地址匹配前置位 I2C_TRANS_START。若主机在 I2C_TRANS_START 未置位时向从机发起寻址，从机可能会应答该地址，但状态机无法正常运行，从而导致从机控制器出现错误行为或时序紊乱。

47.5 LP_I2C 和 I2C 的功能差异

LP_I2C 可以作为主机在主系统休眠时和外部设备通信，它包含所有 ESP32-P4 I2C 主机部分的功能，但不包含任何 ESP32-P4 I2C 从机部分的功能。LP_I2C 不包含任何与从机功能相关的寄存器，详细的寄存器列表见 47.8.2 LP_I2C 寄存器列表。

LP_I2C 和 I2C 主机在设计上的区别有下列几点：

- LP_I2C 中 TX/RX RAM 为 16x8 位大小，即 TX/RX FIFO 深度变为 16 字节。
- LP_I2C 的 APB_CLK 的时钟源为 CLK_AON_FAST。通过配置 LPPERI_LP_I2C_CLK_SEL 选择 I2C_SCLK 的时钟源。具体时钟源配置为：
  - 0：CLK_ROOT_FAST
  - 1：CLK_XTALD2
  - 2：PLL_F8M_CLK

  配置 LPPERI_CK_EN_LP_I2C 为高电平来使能 I2C_SCLK 的时钟源。时钟频率发生变化时要注意对时序寄存器做出相应调整。

LP_I2C 的编程示例参见 47.6 中的主机部分。

47.6 编程示例

本节提供一些典型通信场景的编程示例。ESP32-P4 中有两个 I2C 控制器，为了便于描述，下文所有图示中的 I2C 主机和从机都假定为 ESP32-P4 I2C 外设控制器。

47.6.1 I2C 主机写入从机，7 位寻址，单次命令序列

47.6.1.1 场景介绍

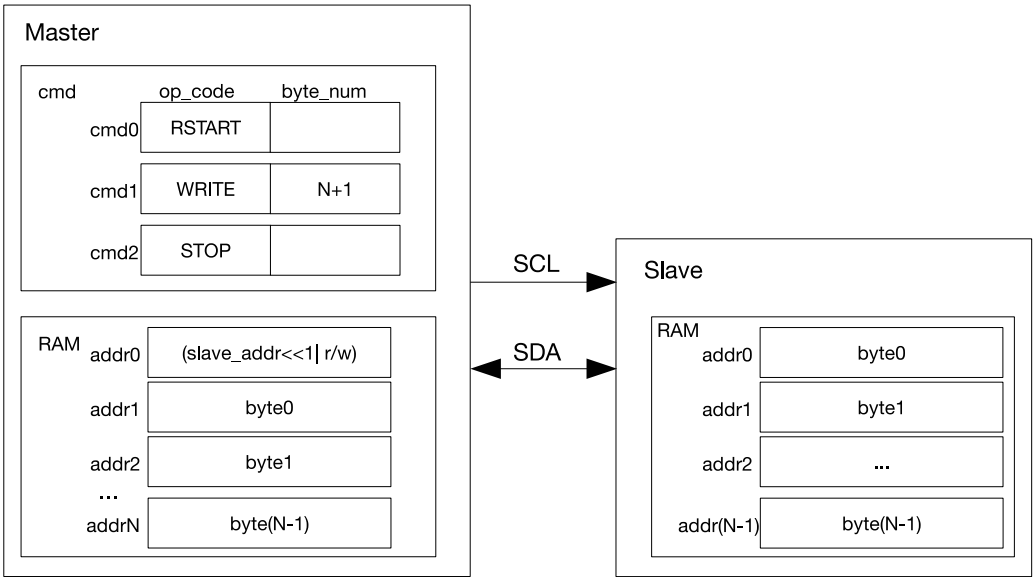


图 47.6-1. I2C 主机写 7 位寻址的从机

图 47.6-1 为 I2C 主机采用 7 位寻址写 N 个字节数据到 I2C 从机的寄存器或 RAM。如图 47.6-1 所示，主机 RAM 中第一个字节数据为 7 位从机地址 + 1 位读写标志位，其中读写标志位为 0 时表示写操作，接下来的连续空间存储待发送的数据。cmd 框中包含了相应的命令序列。

对于主机，在软件配置好命令序列以及 RAM 数据后，置位 I2C_TRANS_START 寄存器启动控制器进行数据传输。控制器的行为可分为四步：

1. 等待 SCL 线为高电平，以避免 SCL 线被其他主机或者从机占用。
2. 通过发送 START 位来执行 RSTART 命令。
3. 执行 WRITE 命令从 RAM 的首地址开始取出 N+1 个字节并依次发送给从机，其中第一个字节为地址。
4. 发送 STOP。当 I2C 主机完成 STOP 位的传输后，会产生 I2C_TRANS_COMPLETE_INT 中断。

47.6.1.2 配置示例

1. 参照章节 47.4.7，配置 I2C 主机和 I2C 从机的时序参数寄存器。
2. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	—	ack_exp	1	N+1
I2C_COMMAND2（主机）	STOP	—	—	—	—

5. 参考章节 47.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。

6. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
10. I2C 主机发送数据，并根据 ack_check_en（主机）的配置决定是否对 ACK 信号进行检测。
11. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 47.4.10。
12. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作，具体做法参照章节 47.4.10。

    若接收数据 N 大于 RX FIFO 深度，另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
13. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.6.2 I2C 主机写入从机，10 位寻址，单次命令序列

47.6.2.1 场景介绍

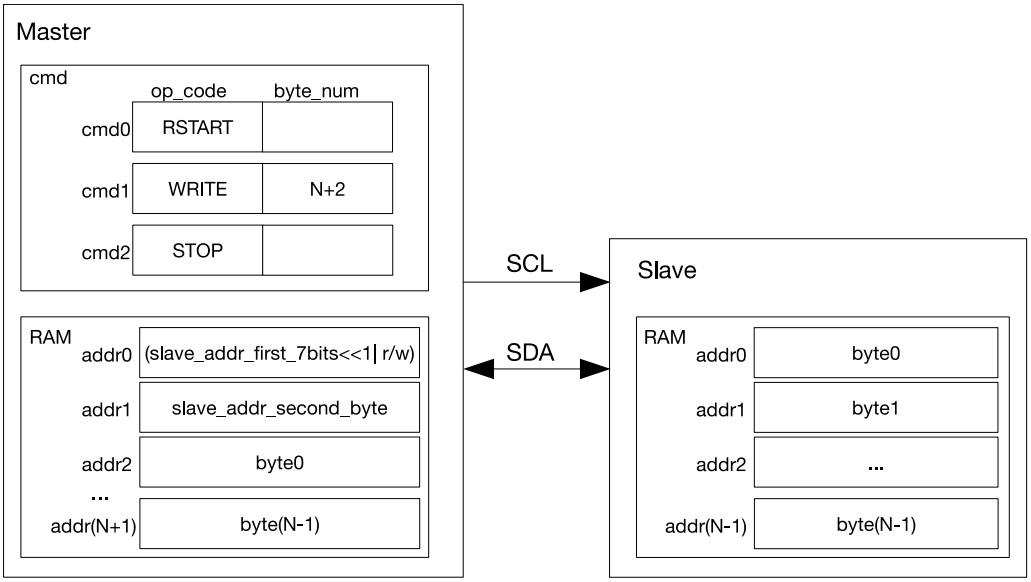


图 47.6-2. I2C 主机写 10 位寻址的从机

图 47.6-2 为 I2C 主机写 N 个字节到 10 位地址 I2C 从机的配置图。整个配置和传输过程都和 47.6.1 中类似，除了在传输的开始主机在 10 位寻址模式下需要发送两字节地址段。由于 10 位从机地址比 7 位地址多一个字节，所以 WRITE 命令对应的 byte_num 以及 TX RAM 中数据数量都相应增加 1。

47.6.2.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	—	ack_exp	1	N+2
I2C_COMMAND2（主机）	STOP	—	—	—	—

4. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的 10 位从机地址，并将 I2C_ADDR_10BIT_EN（从机）置 1 使能 10 位寻址模式。
5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 ((0x78 | I2C_SLAVE_ADDR[9:8])«1)|R/W，第二个地址字节是 I2C_SLAVE_ADDR[7:0]。之后就是要发送的数据，可选 FIFO 方式和直接访问方式。
6. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
7. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
9. I2C 主机发送数据，并根据 ack_check_en（主机）的配置判断是否进行 ACK 检测。
10. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 47.4.10。
11. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作，具体做法参照章节 47.4.10。

    若接收数据 N 大于 RX FIFO 深度，另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
12. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.6.3 I2C 主机写入从机，7 位双地址寻址，单次命令序列

47.6.3.1 场景介绍

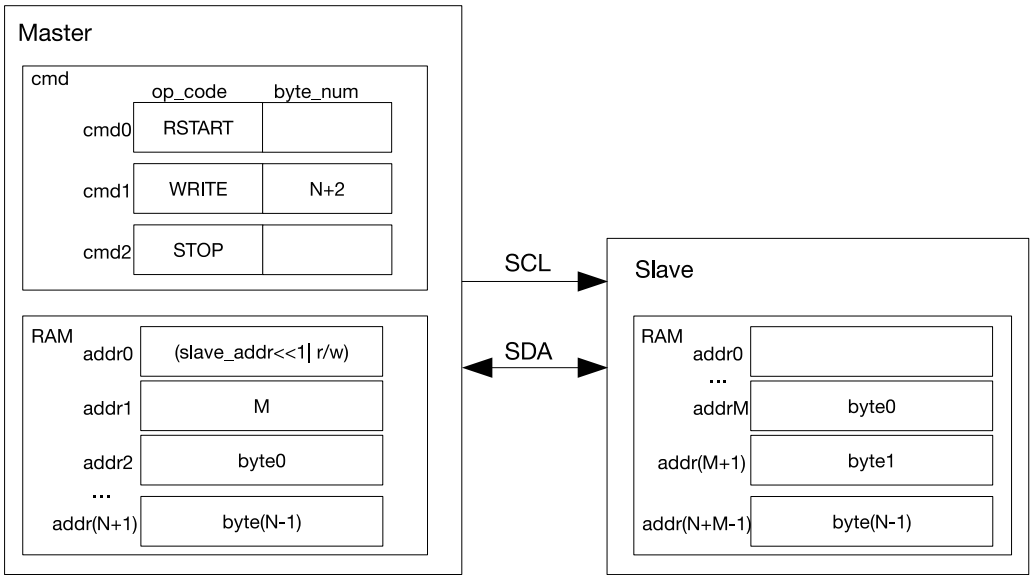


图 47.6-3. I2C 主机写 7 位双地址寻址从机

图 47.6-3 展示了 I2C 主机如何采用 7 位双寻址模式向 I2C 从机的寄存器或 RAM 写入 N 个字节数据。整个配置和传输过程都和章节 47.6.1 中类似，区别是传输的开始主机在 7 位双寻址模式下需要发送两个字节。双地址的第一个地址是 7 位从机地址，第二个地址是 I2C 从机的内存地址（即图 47.6-3 中的 addrM）。双寻址模式下，从机 RX RAM 必须采用 non-FIFO 方式访问，且必须禁用从机的时钟拉伸功能。non-FIFO 模式下，写指针超过 FIFO 深度时会回绕到 FIFO 的起始位置。因此，从机 RX RAM 中的数据会以每 32 字节为周期被循环覆盖。当新数据写入一个非空地址时，新数据会直接覆盖已有数据，且不会给出任何警告。

47.6.3.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 设置I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	—	ack_exp	1	N+2
I2C_COMMAND2（主机）	STOP	—	—	—	—

5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，可以用 FIFO 方式或直接访问方式。
6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。

9. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
10. I2C 从机接收到 I2C 主机发送的内存地址，完成 RX RAM 的地址偏移。
11. I2C 主机发送数据，并根据 ack_check_en（主机）的配置判断是否进行 ACK 检测。
12. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 47.4.10。
13. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.6.4 I2C 主机写入从机，7 位寻址，多次命令序列

47.6.4.1 场景介绍

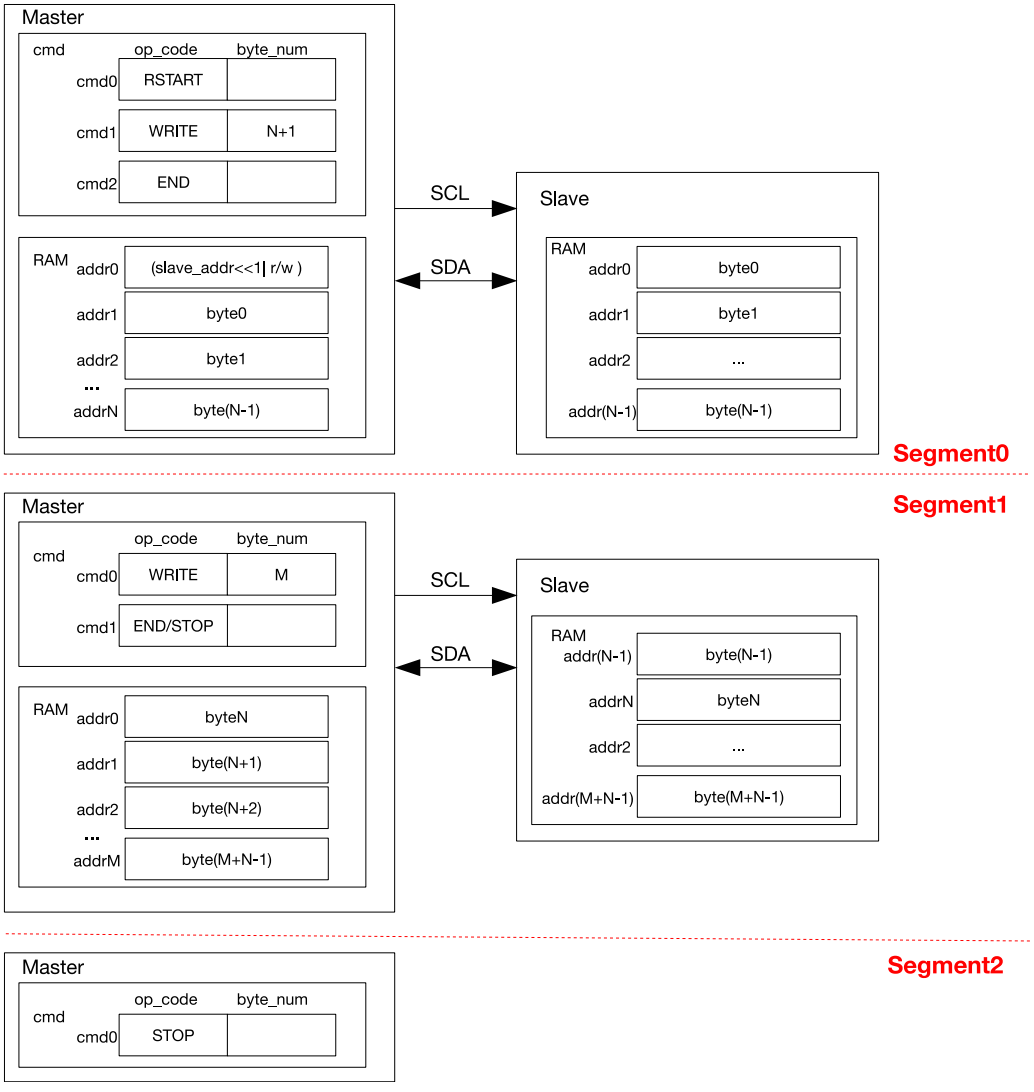


图 47.6-4. I2C 主机分段写 7 位寻址的从机

RAM 的大小只有 TX/RX FIFO 深度，当进行大量的数据传输时，建议使用多次命令序列进行分段传输。每次命令序列以 END 命令结束。当控制器执行 END 命令时，将拉低 SCL 线，软件此时可以更新命令序列寄存器和 RAM 的内容以用于下一次命令序列的传输。

以两段和三段传输为例，如图 47.6-4 所示为 I2C 主机分成三段或者两段写从机。配置 I2C 主机的命令序列如第一段所示，并且在主机的 RAM 中准备好数据，置位 I2C_TRANS_START，I2C 主机即开始数据传输。在执行到 END 命令后，I2C 主机会关闭 SCL 时钟，并将 SCL 线拉低来防止其他设备占用 I2C 总线。此时控制器产生 I2C_END_DETECT_INT 中断。

在检测到 I2C_END_DETECT_INT 中断后，软件可以更新命令序列以及 RAM 中的内容如第二段所示，并清除 I2C_END_DETECT_INT 中断。当第二段中 cmd1 为 STOP 时，不需要第三段，即为两段写从机。置位 I2C_TRANS_START 后，I2C 主机继续发送数据，并在最后发送 STOP 位。

当为三段写从机时，I2C 主机在第二段发送完数据，并检测到 I2C 主机的 I2C_END_DETECT_INT 中断后，即可配置 cmd 如第三段所示。置位 I2C_TRANS_START 后，I2C 主机即产生 STOP 位，从而停止传输。

请注意，在两个分段之间，I2C 总线上的其他主机设备不会占用总线。只有在发送了 STOP 命令后总线才会被释放。任何情况下，置位 I2C_FSM_RST 可复位 I2C 控制器，该寄存器硬件自清 I2C_FSM_RST。

47.6.4.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	—	ack_exp	1	N+1
I2C_COMMAND2（主机）	END	—	—	—	—

4. 参考章节 47.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。
5. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
6. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
7. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
9. I2C 主机发送数据，并根据 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
10. 等到 I2C_END_DETECT_INT（主机）中断产生后，设置I2C_END_DETECT_INT_CLR（主机）为 1 来清除中断。
11. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	WRITE	—	ack_exp	1	M
I2C_COMMAND1（主机）	END/STOP	—	—	—	—

12. 向 I2C 主机的 TX RAM 写入 M 个要发送的数据，可以用 FIFO 方式或直接访问方式。
13. 向 I2C_TRANS_START（主机）位写 1 开始传输，并重复步骤 9 的流程。
14. 若指令为 STOP，I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。
15. 若I2C_COMMAND1（主机）的指令为 END，则重复步骤 10。
16. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1（主机）	STOP	—	—	—	—

17. 向 I2C_TRANS_START（主机）位写 1 开始传输。
18. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.6.5 I2C 主机读取从机，7 位寻址，单次命令序列

47.6.5.1 场景介绍

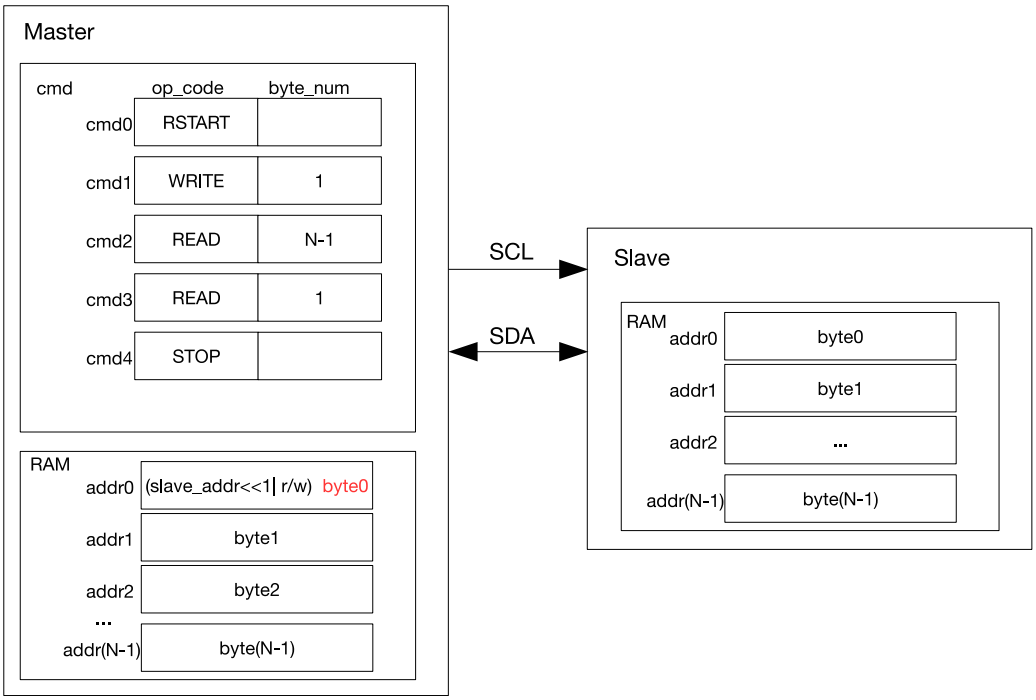


图 47.6-5. I2C 主机读 7 位寻址的从机

图 47.6-5 I2C 主机从 7 位寻址 I2C 从机读取 N 个字节数据的寄存器或 RAM 的值。cmd1 为 WRITE 命令，I2C 主机会将 I2C 从机的地址发送出去。该命令发送的字节是 7 位 I2C 从机地址以及读写标志位。读写标志位为 1 表示读操作。I2C 从机在地址匹配成功之后即开始发送数据给 I2C 主机。I2C 主机根据 READ 命令中设置的 ack_value，在接收完一个字节的数据之后回复 ACK。

图 47.6-5 中 READ 分成两次，I2C 主机对 cmd2 中 N-1 个数据均回复 ACK，对 cmd3 中的数据即传输的最后一个数据回复 NACK，实际使用时可以根据需要进行配置。在存储接收的数据时，I2C 主机从 RX RAM 的首地址开始存储，即为图中红色 byte0 存储的位置。

47.6.5.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	1
I2C_COMMAND2（主机）	READ	0	0	1	N-1
I2C_COMMAND3（主机）	READ	1	0	1	1
I2C_COMMAND4（主机）	STOP	—	—	—	—

5. 参考章节 47.4.10，向 I2C 主机的 TX RAM 写入从机地址。可选 FIFO 方式和直接访问方式。
6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 47.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. 等待I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
12. 参考章节 47.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
13. 将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
14. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
15. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将

I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。

16. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 I2C_NACK_INT 中断，停止发送。
17. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.6.6 I2C 主机读取从机，10 位寻址，单次命令序列

47.6.6.1 场景介绍

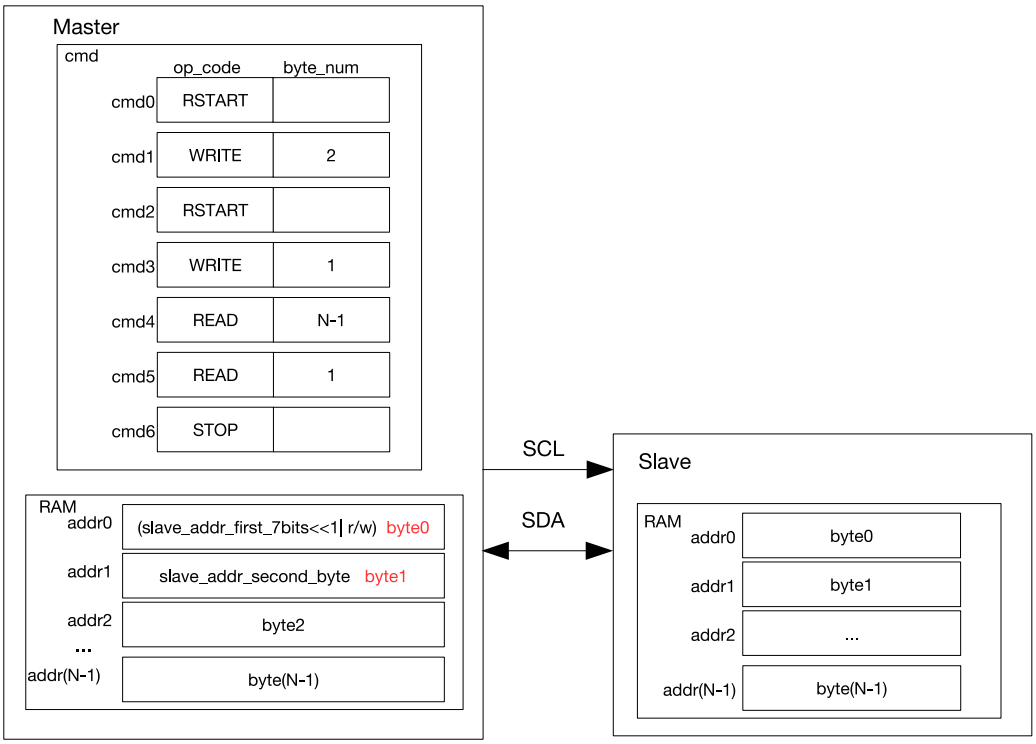


图 47.6-6. I2C 主机读 10 位寻址的从机

图 47.6-6 为 I2C 主机从 10 位寻址的 I2C 从机中读取数据的寄存器或 RAM 的值。相比于 7 位寻址，I2C 主机的第一写命令的字节数为两个字节，相应 TX RAM 中存储两个字节的 I2C 从机 10 位地址，且第一个地址字节的 R/W 位为 W（主机）。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位为 R（从机）。之后主机从从机中读取数据。两个字节地址的配置方式与章节 47.6.2 的相同。

47.6.6.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照 I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1
I2C_COMMAND5（主机）	READ	1	0	1	1
I2C_COMMAND6（主机）	STOP	—	—	—	—

5. 在 I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的 10 位从机地址，并将 I2C_ADDR_10BIT_EN（从机）置 1 使能 10 位寻址模式。
6. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 ((0x78 | I2C_SLAVE_ADDR[9:8])«1)|R/W，R/W 位为 W（主机）；第二个地址字节是 I2C_SLAVE_ADDR[7:0]。第三个字节是重复的第一个地址字节加上 R/W 位，其中 R/W 位为 R（从机）。可选 FIFO 方式和直接访问方式。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 47.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. I2C 主机发送 RSTART 命令，并发送 TX RAM 里的第三个字节，即为重复的地址字节和 R 位。
12. I2C 从机重复执行步骤 10，若地址匹配，继续后面的步骤。
13. 等待I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
14. 参考章节 47.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
15. 将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
16. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
17. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
18. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 I2C_NACK_INT 中断，停止发送。

19. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.6.7 I2C 主机读取从机，7 位双寻址，单次命令序列

47.6.7.1 场景介绍

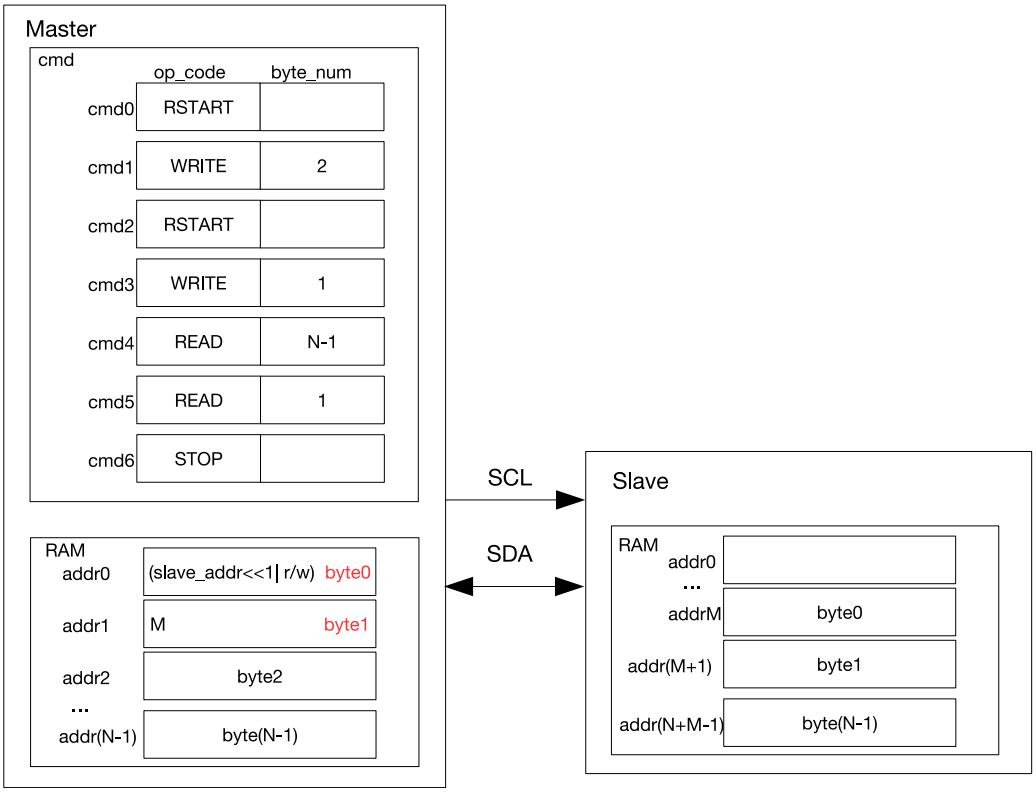


图 47.6-7. I2C 主机从 7 位双寻址从机的 M 地址读取 N 个数据

图 47.6-7 展示了 I2C 主机从 I2C 从机指定地址读取数据的过程。主机在传输开始发送 2 个地址字节，第一个地址字节是从机的 7 位地址加 R/W 位，R/W 位为 W（主机）；第二个地址字节是从机的内存地址 M。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位变为 R（从机）。之后主机从从机的 AddrM 地址开始读取数据。当使用双寻址模式时，从机 TX RAM 必须采用非 FIFO 模式进行访问，且必须禁用从机的时钟拉伸功能。在此配置下，从机的 TX FIFO 以非 FIFO 方式运行。即使在主机尝试读取的数据量超过从机实际可用数据量（下溢情况）时，从机仍会继续返回与当前地址指针对应的数据，且不产生任何警告。当地址指针到达存储器空间末尾时，将自动回绕至起始位置。

47.6.7.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 设置I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
4. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
5. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1
I2C_COMMAND5（主机）	READ	1	0	1	1
I2C_COMMAND6（主机）	STOP	—	—	—	—

6. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的 7 位地址，I2C_ADDR_10BIT_EN（从机）置 0 使能 7 位寻址模式。
7. 参考章节 47.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 (I2C_SLAVE_ADDR[6:0])«1)|R/W，R/W 位为 W（主机）；第二个地址字节是 I2C 从机的内存地址 M。第三个字节是重复的第一个地址字节加上 R/W 位，其中 R/W 位为 R（从机）。可选 FIFO 方式和直接访问方式。
8. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
9. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
10. 参考章节 47.4.14，启动 I2C 从机的传输。
11. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
12. I2C 从机接收到 I2C 主机发送的内存地址，完成 TX RAM 的地址偏移。
13. I2C 主机发送 RSTART 命令，并发送 TX RAM 里的第三个字节，即为重复的地址字节和 R 位。
14. I2C 从机重复执行步骤 11，若地址匹配，继续后面的步骤。
15. 以 non-FIFO 模式将要发送的数据写入 I2C 从机的 TX RAM。
16. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
17. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 NACK 中断，停止发送。
18. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.6.8 I2C 主机读取从机，7 位寻址，多次命令序列

47.6.8.1 场景介绍

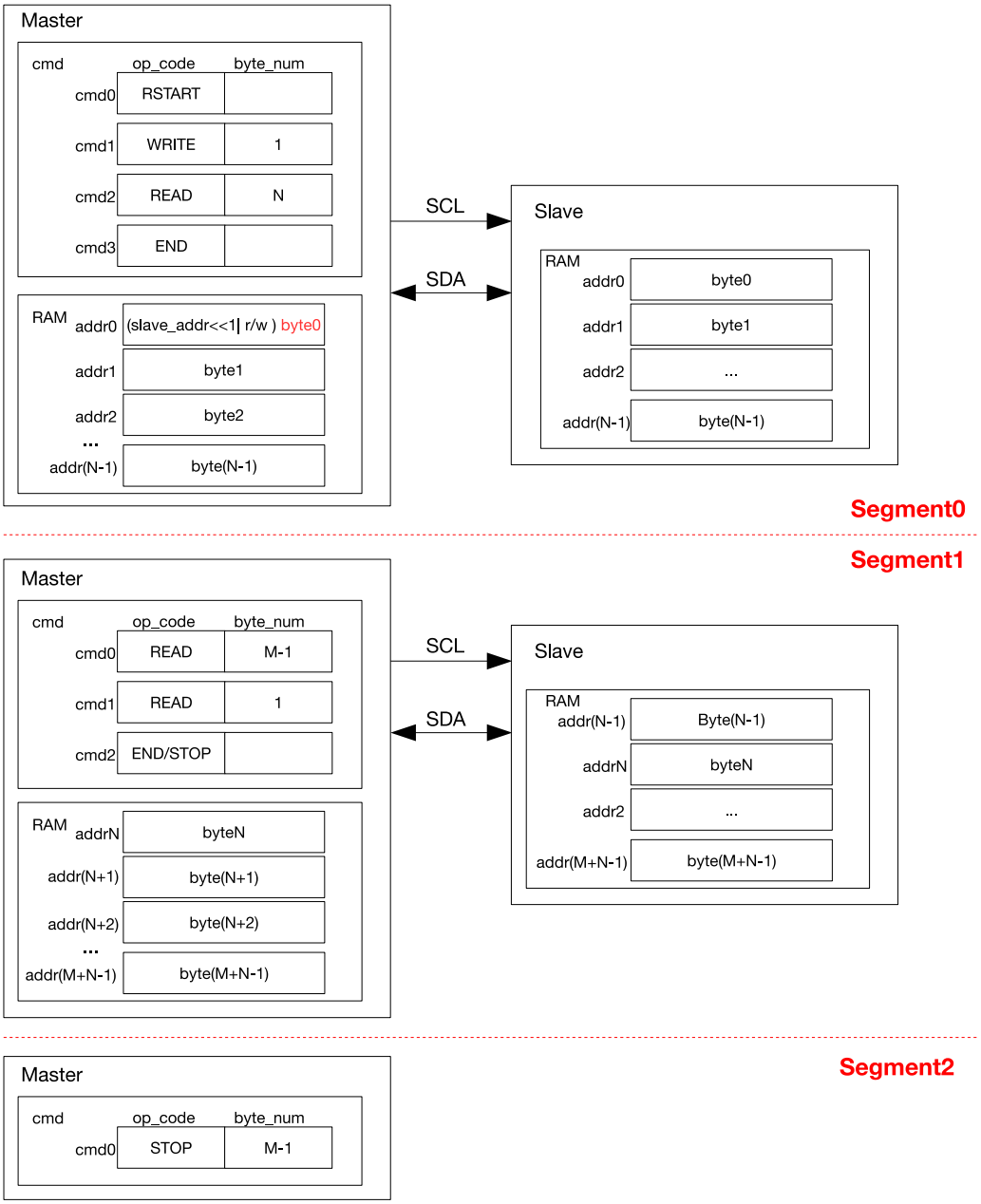


图 47.6-8. I2C 主机分段读 7 位寻址的从机

图 47.6-8 为 I2C 主机通过 END 命令分三段或者分两段，从 I2C 从机读取 N+M 个数据的流程图。

1. 第一段流程和 47.6-5 类似，只是最后一个命令变为 END。
2. 接着在从机的 TX RAM 中准备好数据，置位 I2C_TRANS_START，当执行到 END 命令时，I2C 主机可以更新命令寄存器和 RAM 的内容，如第二段所示，并且清零其对应的 I2C_END_DETECT_INT 中断。当第二段中 cmd2 为 STOP 时，即两段读 I2C 从机，置位 I2C_TRANS_START，I2C 主机继续传输数据，最后发送 STOP 位来停止传输。
3. 当第二段中 cmd2 为 END 时，在 I2C 主机完成第二次数据传输，并检测到 I2C 主机的 I2C_END_DETECT_INT 中断后，配置 cmd 如第三段所示。置位 I2C_TRANS_START，I2C 主机发送 STOP 位停止传输。

47.6.8.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	1
I2C_COMMAND2（主机）	READ	0	0	1	N
I2C_COMMAND3（主机）	END	—	—	—	—

5. 向 I2C 主机的 TX RAM 写入从机地址，可选 FIFO 方式和直接访问方式。
6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 47.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. 等待I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
12. 参考章节 47.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
13. 将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
14. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
15. 若 I2C 主机一个 READ 指令要读取的数 N 或 M 超过 I2C 从机的 TX FIFO 深度个字节，当 I2C 从机的 TX RAM 中发送数据全部发完，为空时产生I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
16. 等到一次 READ 指令完成，I2C 主机执行 END 指令，I2C_END_DETECT_INT（主机）中断产生后，设置 I2C_END_DETECT_INT_CLR（主机）为 1 来清除中断。

17. 更新 I2C 主机的指令寄存器，有两种设置方式：

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	READ	ack_value	ack_exp	1	M
I2C_COMMAND1（主机）	END	—	—	—	—

或者

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	READ	0	0	1	M-1
I2C_COMMAND1（主机）	READ	1	0	1	1
I2C_COMMAND2（主机）	STOP	—	—	—	—

18. 向 I2C 从机的 TX RAM 写入 M 个字节要发送的数据，若 M 大于 TX FIFO 深度，重复步骤 12，可以用 FIFO 方式或直接访问方式。
19. 向 I2C_TRANS_START（主机）位写 1 开始传输，并重复步骤 14 的流程。
20. 若最后一个指令为 STOP，则当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 NACK 中断，停止发送。I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。
21. 若最后一个指令为为 END，则重复步骤 16，并在完成后继续下面的步骤。
22. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1（主机）	STOP	—	—	—	—

23. 向 I2C_TRANS_START（主机）位写 1 开始传输。
24. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

47.7 中断

ESP32-P4 的 I2C*n* 可以生成 I2C*n*_INTR 中断信号并发送给中断矩阵。这一中断信号由以下 I2C*n* 的内部中断源生成：

- I2C_SLAVE_STRETCH_INT: 从机模式下，当从机时钟拉伸时，产生此中断。
- I2C_DET_START_INT：主机或从机检测到 I2C START 信号，触发此中断。
- I2C_SCL_MAIN_ST_TO_INT: 当 I2C 主状态机 SCL_MAIN_FSM 保持某个状态超过 I2C_SCL_MAIN_ST_TO_I2C[23:0] 个模块时钟周期时，触发此中断。
- I2C_SCL_ST_TO_INT: 当 I2C 状态机 SCL_FSM 保持某个状态超过 I2C_SCL_ST_TO_I2C[23:0] 个模块时钟周期时，触发此中断。
- I2C_RXFIFO_UDF_INT: 当 I2C 通过 APB 总线读取 RX FIFO，但 RX FIFO 为空时，触发该中断。
- I2C_TXFIFO_OVF_INT: 当 I2C 通过 APB 总线写 TX FIFO，但 TX FIFO 为满时，触发该中断。

- I2C_NACK_INT: 当 I2C 配置为主机时，接收到的 ACK 与命令中期望的 ACK 值不一致时，即触发该中断；当 I2C 配置为从机时，接收到的 ACK 值为 1 时即触发该中断。
- I2C_TRANS_START_INT: 当 I2C 发送一个 START 位时，即触发该中断。
- I2C_TIME_OUT_INT: 在传输过程中，当 I2C SCL 保持为高或为低电平的时间超过 $2^{I2C_TIME_OUT_VALUE}$ 个模块时钟后，即触发该中断。
- I2C_TRANS_COMPLETE_INT: 当 I2C 检测到 STOP 位时，即触发该中断。
- I2C_MST_TXFIFO_UDF_INT: 当 I2C 主机的 TX FIFO 下溢时，触发此中断。
- I2C_ARBITRATION_LOST_INT: 当 I2C 主机的 SCL 为高电平，SDA 输出值与输入值不相等时，即触发该中断。
- I2C_BYTE_TRANS_DONE_INT: 当 I2C 发送或接收一个字节，即触发该中断。
- I2C_END_DETECT_INT：当 I2C 主机命令的 op_code 为 END，且检测到 I2C END 状态时，触发此中断。
- I2C_RXFIFO_OVF_INT：当 I2C RX FIFO 上溢时，触发此中断。
- I2C_TXFIFO_WM_INT：I2C TX FIFO 水标中断。当I2C_FIFO_PRT_EN 为 1，且 TX FIFO 指针小于 I2C_TXFIFO_WM_THRHD[4:0] 时，触发此中断。
- I2C_RXFIFO_WM_INT：I2C RX FIFO 水标中断。当I2C_FIFO_PRT_EN 为 1，且 RX FIFO 指针大于 I2C_RXFIFO_WM_THRHD[4:0] 时，触发此中断。
- I2C_SLAVE_ADDR_UNMATCH_INT: 从机模式下，接收到的从机地址和内部配置的从机地址不一致时，触发此中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 47.8.1 *I2C 寄存器列表*。

47.8 寄存器列表

47.8.1 I2C 寄存器列表

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时序寄存器			
I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平的保持时间	0x0000	R/W
I2C_SDA_HOLD_REG	配置 SCL 下降沿后的数据保持时间	0x0030	R/W
I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平保持时间	0x0038	R/W
I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟时间	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	配置 STOP 命令生成时 SDA 和 SCL 上升沿之间的间隔时间	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x007C	R/W
配置寄存器			
I2C_CTR_REG	传输配置寄存器	0x0004	varies
I2C_TO_REG	数据接收超时控制寄存器	0x000C	R/W
I2C_SLAVE_ADDR_REG	从机地址配置寄存器	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	R/W
I2C_FILTER_CFG_REG	SCL 和 SDA 滤波配置寄存器	0x0050	R/W
I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
I2C_SCL_STRETCH_CONF_REG	配置 I2C 从机 SCL 时钟拉伸	0x0084	varies
状态寄存器			
I2C_SR_REG	I2C 的工作状态寄存器	0x0008	RO
I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
I2C_DATA_REG	Rx FIFO 读取数据寄存器	0x001C	HRO
中断寄存器			
I2C_INT_RAW_REG	原始中断状态	0x0020	R/ SS/ WTC
I2C_INT_CLR_REG	中断清除位	0x0024	WT
I2C_INT_ENA_REG	中断使能位	0x0028	R/W
I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
I2C_COMD0_REG	I2C 命令寄存器 0	0x0058	varies
I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	varies

名称	描述	地址	访问
I2C_COMD2_REG	I2C 命令寄存器 2	0x0060	varies
I2C_COMD3_REG	I2C 命令寄存器 3	0x0064	varies
I2C_COMD4_REG	I2C 命令寄存器 4	0x0068	varies
I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	varies
I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	varies
I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	varies
版本寄存器			
I2C_DATE_REG	版本寄存器	0x00F8	R/W
地址寄存器			
I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO 基地址寄存器	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO 基地址寄存器	0x0180	HRO

47.8.2 LP_I2C 寄存器列表

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时序寄存器			
LP_I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平的保持时间	0x0000	R/W
LP_I2C_SDA_HOLD_REG	配置 SCL 下降沿后的数据保持时间	0x0030	R/W
LP_I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
LP_I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平保持时间	0x0038	R/W
LP_I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
LP_I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟时间	0x0044	R/W
LP_I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	R/W
LP_I2C_SCL_STOP_SETUP_REG	配置 STOP 命令生成时 SDA 和 SCL 上升沿之间的间隔时间	0x004C	R/W
LP_I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0078	R/W
LP_I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x007C	R/W
配置寄存器			
LP_I2C_CTR_REG	传输配置寄存器	0x0004	varies
LP_I2C_TO_REG	数据接收超时控制寄存器	0x000C	R/W
LP_I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	R/W
LP_I2C_FILTER_CFG_REG	SCL 和 SDA 滤波配置寄存器	0x0050	R/W
LP_I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
状态寄存器			
LP_I2C_SR_REG	I2C 的工作状态寄存器	0x0008	RO
LP_I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
LP_I2C_DATA_REG	RX FIFO 读取数据寄存器	0x001C	RO
中断寄存器			
LP_I2C_INT_RAW_REG	原始中断状态	0x0020	R/SS/WTC

名称	描述	地址	访问
LP_I2C_INT_CLR_REG	中断清除位	0x0024	WT
LP_I2C_INT_ENA_REG	中断使能位	0x0028	R/W
LP_I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
LP_I2C_COMD0_REG	I2C 命令寄存器 0	0x0058	varies
LP_I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	varies
LP_I2C_COMD2_REG	I2C 命令寄存器 2	0x0060	varies
LP_I2C_COMD3_REG	I2C 命令寄存器 3	0x0064	varies
LP_I2C_COMD4_REG	I2C 命令寄存器 4	0x0068	varies
LP_I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	varies
LP_I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	varies
LP_I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	varies
版本寄存器			
LP_I2C_DATE_REG	版本寄存器	0x00F8	R/W
地址寄存器			
LP_I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO 基地址寄存器	0x0100	HRO
LP_I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO 基地址寄存器	0x0180	HRO

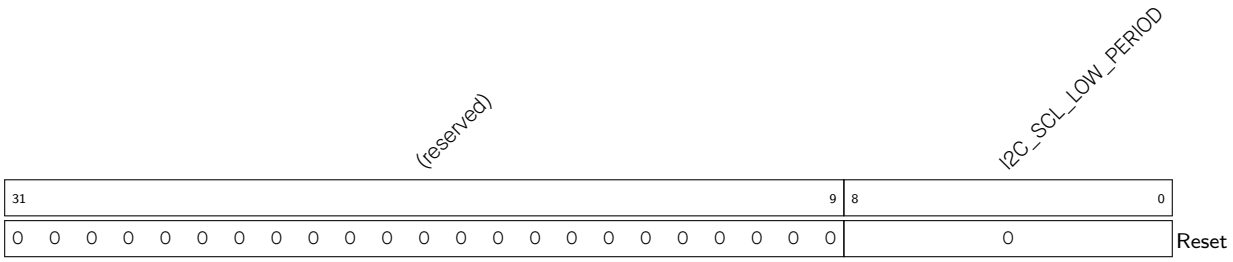
47.9 寄存器

47.9.1 I2C 寄存器

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 47.1. I2C_SCL_LOW_PERIOD_REG (0x0000)

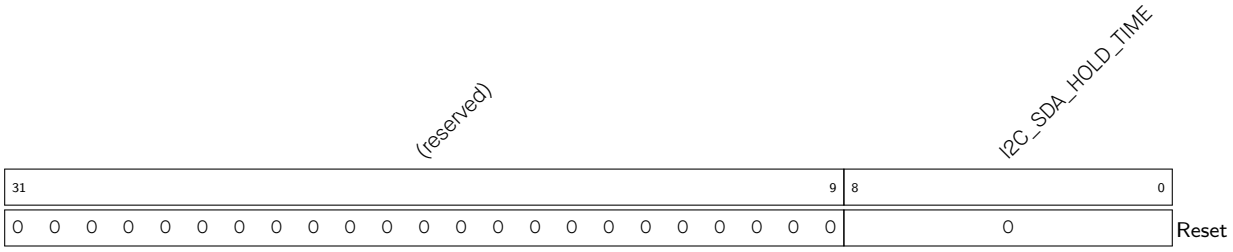


I2C_SCL_LOW_PERIOD 配置主机模式下 SCL 低电平的保持时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.2. I2C_SDA_HOLD_REG (0x0030)

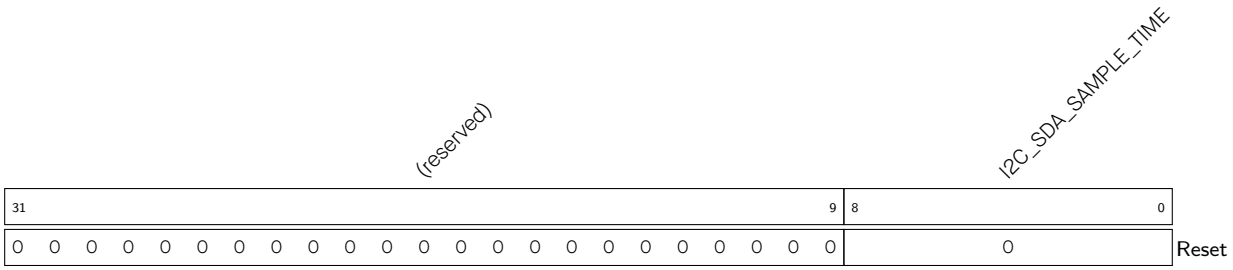


I2C_SDA_HOLD_TIME 配置 SCL 下降沿后的数据保持时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.3. I2C_SDA_SAMPLE_REG (0x0034)

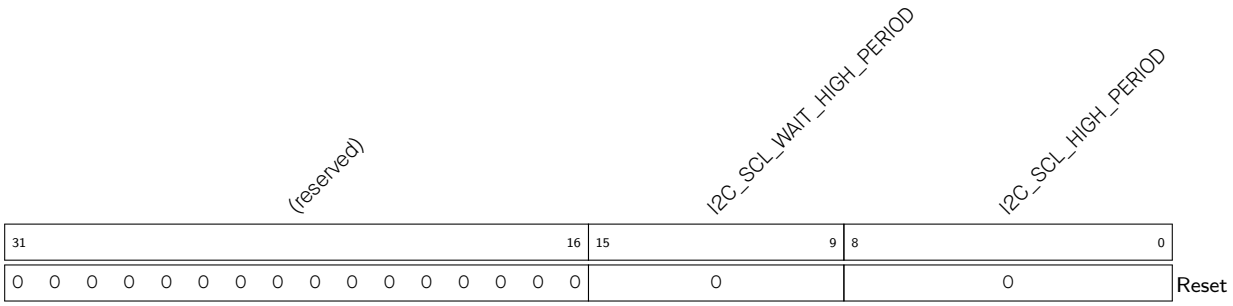


I2C_SDA_SAMPLE_TIME 配置采样 SDA 的时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD 配置 SCL 在主机模式下保持高电平的时间。

单位：I2C_SCLK 时钟周期数

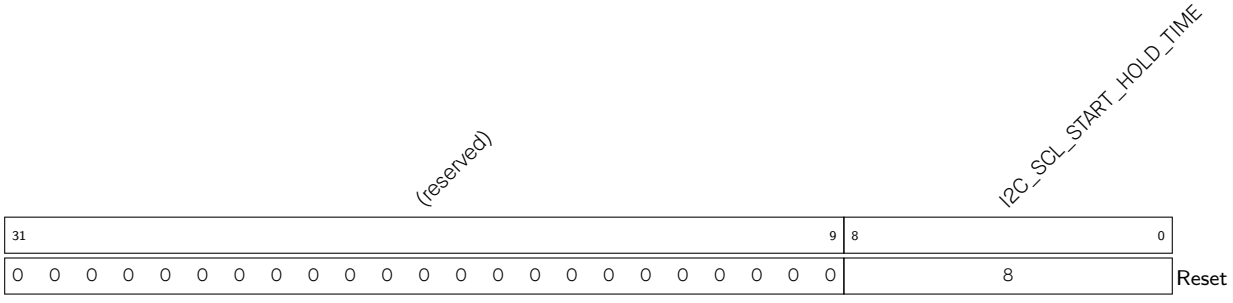
(R/W)

I2C_SCL_WAIT_HIGH_PERIOD 配置 SCL_FSM 等待 SCL 在主机模式下翻转至高电平的时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.5. I2C_SCL_START_HOLD_REG (0x0040)

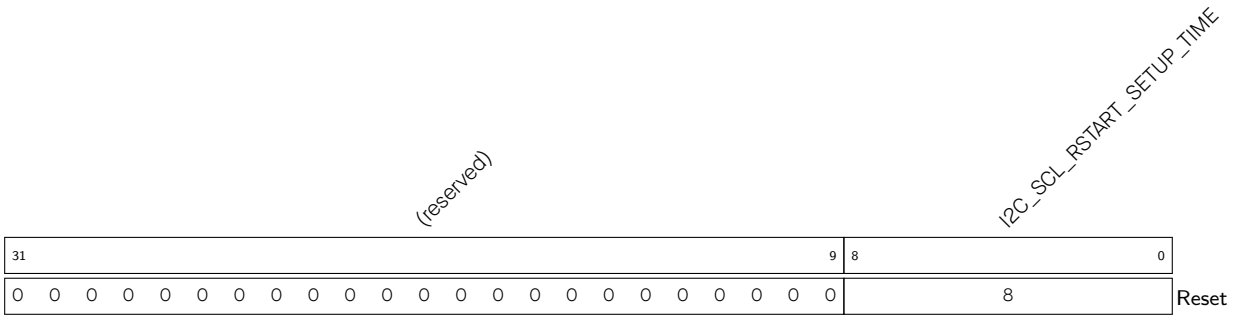


I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.6. I2C_SCL_RSTART_SETUP_REG (0x0044)

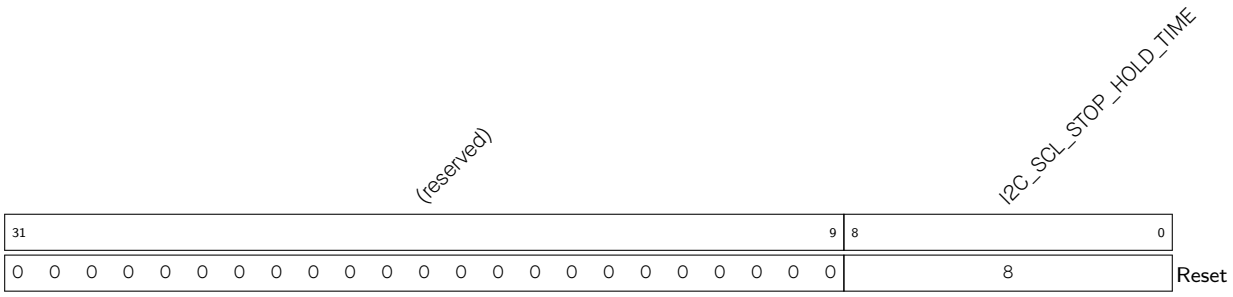


I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.7. I2C_SCL_STOP_HOLD_REG (0x0048)

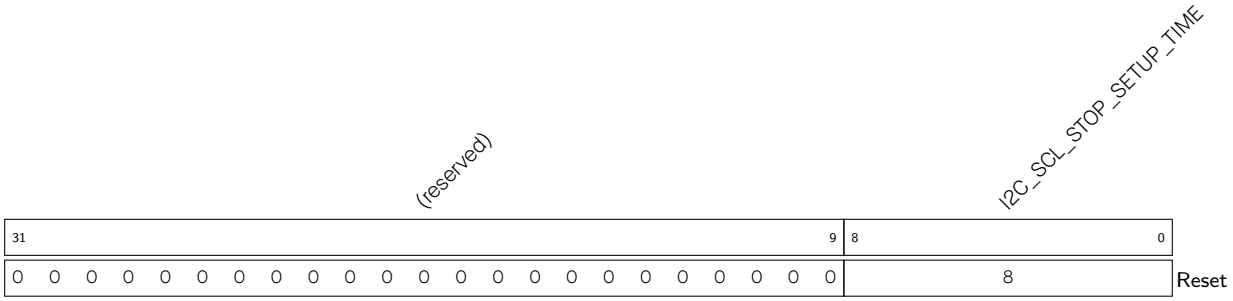


I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.8. I2C_SCL_STOP_SETUP_REG (0x004C)

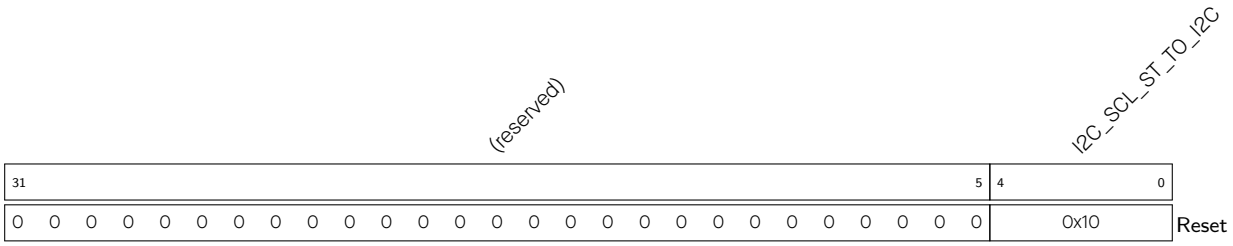


I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



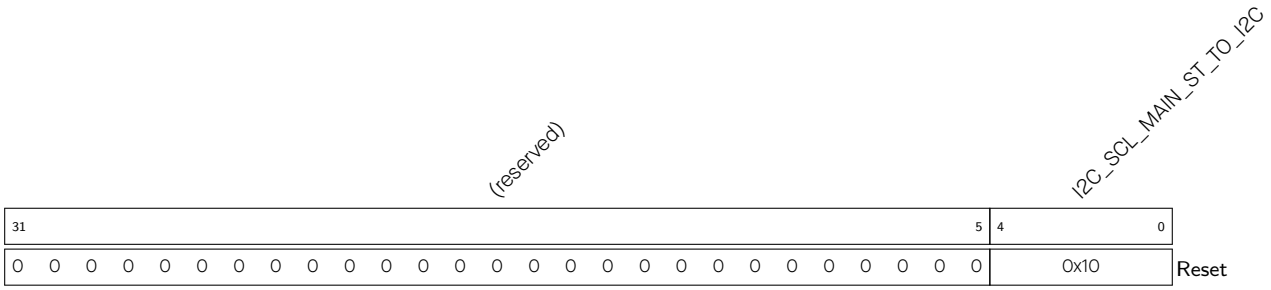
I2C_SCL_ST_TO_I2C 配置 SCL_FSM 状态保持不变的最大时间。该值应该大于 1，小于等于 23。

$$\text{最大时钟周期数} = 2^{\text{I2C_SCL_ST_TO_I2C}+1} - 1$$

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

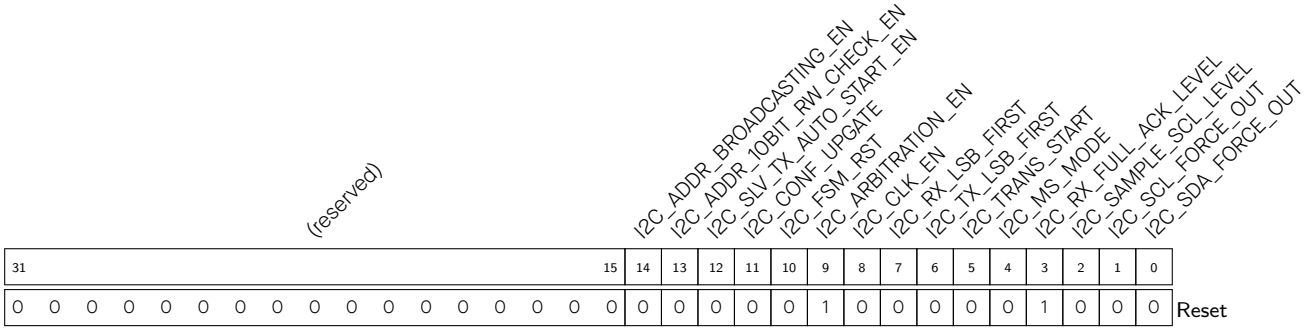


I2C_SCL_MAIN_ST_TO_I2C 配置 SCL_MAIN_FSM 状态保持不变的最大时间，不能大于 23。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 47.11. I2C_CTR_REG (0x0004)



I2C_SDA_FORCE_OUT 配置 SDA 输出模式。

0：开漏输出

1：直接输出

(R/W)

I2C_SCL_FORCE_OUT 配置 SCL 输出模式。

0：开漏输出

1：直接输出

(R/W)

I2C_SAMPLE_SCL_LEVEL 配置 SDA 的采样模式。

0：SCL 为高电平时采样 SDA 数据

1：SCL 为低电平时采样 SDA 数据

(R/W)

I2C_RX_FULL_ACK_LEVEL 用于配置主机在接收的 RX RAM 达到阈值时需发送的 ACK 电平值。(R/W)

I2C_MS_MODE 配置该模块为 I2C 主机或从机。

0：从机

1：主机

(R/W)

I2C_TRANS_START 配置从机是否开始发送 TX FIFO 中的数据。

0：无效

1：开始

(WT)

I2C_TX_LSB_FIRST 控制待发送数据的发送顺序。

0：从最高有效位开始发送数据

1：从最低有效位开始发送数据

(R/W)

I2C_RX_LSB_FIRST 控制接收数据的存储顺序。

0：从最高有效位开始接收数据

1：从最低有效位开始接收数据

(R/W)

见下页...

Register 47.11. I2C_CTR_REG (0x0004)

... 接上页

I2C_CLK_EN 门控寄存器的时钟信号。

0：仅当通过软件读写寄存器时支持时钟

1：强制寄存器时钟开启

(R/W)

I2C_ARBITRATION_EN 配置使能 I2C 总线仲裁检测。

0：无效

1：使能

(R/W)

I2C_FSM_RST 配置复位 SCL_FSM。

0：无效

1：复位

(WT)

I2C_CONF_UPGATE 配置此位实现同步。

0：无效

1：同步

(WT)

I2C_SLV_TX_AUTO_START_EN 配置从机自动发送数据的使能位。

0：禁用

1：使能

(R/W)

I2C_ADDR_10BIT_RW_CHECK_EN 配置使能 10 位寻址模式的读写标志位检查功能，检查读写标志是否符合协议。

0：不检查

1：检查

(R/W)

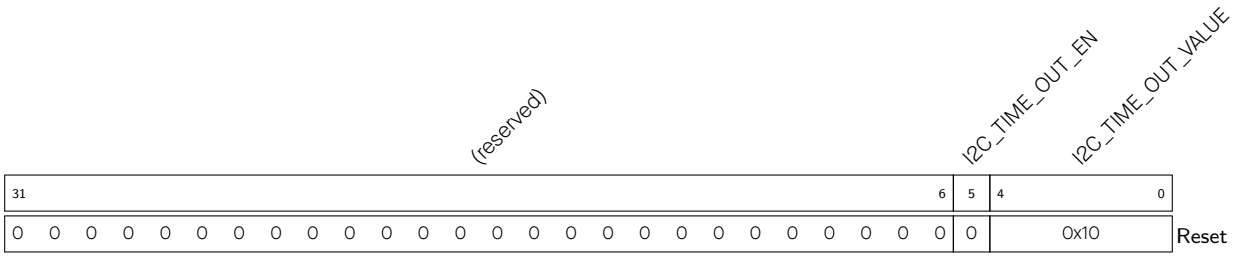
I2C_ADDR_BROADCASTING_EN 配置是否支持 7 位寻址模式的广播功能。

0：不支持

1：支持

(R/W)

Register 47.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE 配置 SCL 在高电平或低电平停滞的超时阈值周期。实际周期是 2^(I2C_TIME_OUT_VALUE+1)–1。

单位：I2C_SCLK 时钟周期数

(R/W)

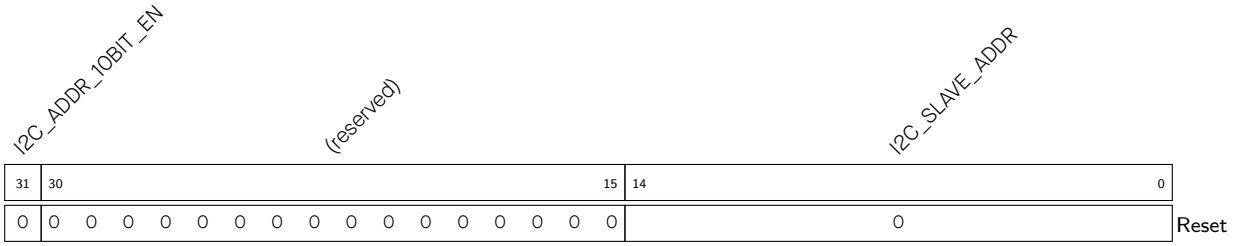
I2C_TIME_OUT_EN 配置使能超时控制。

0：无效

1：使能

(R/W)

Register 47.13. I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR 配置 I2C 从机的从机地址。(R/W)

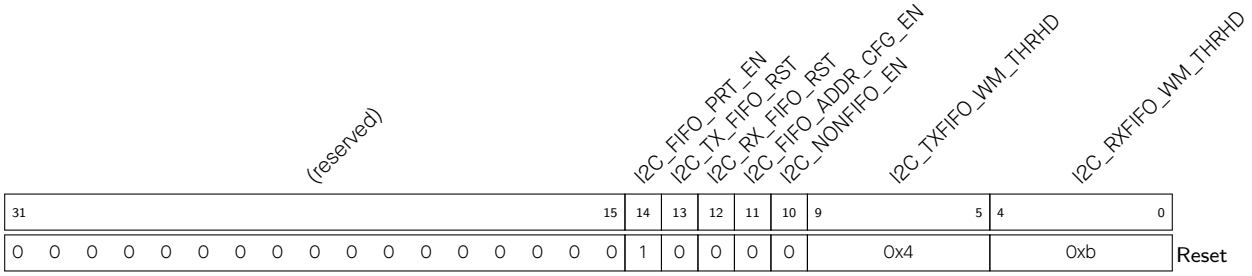
I2C_ADDR_10BIT_EN 配置在主机模式下使能从机的 10 位寻址模式。

0：无效

1：使能

(R/W)

Register 47.14. I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD 配置在直接访问模式下，RX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 I2C_RXFIFO_WM_THRHD[4:0] 时，I2C_RXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_TXFIFO_WM_THRHD 配置在直接访问模式下，TX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 TX FIFO 计数值大于 I2C_TXFIFO_WM_THRHD[4:0] 时，I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_NONFIFO_EN 配置使能 APB 直接访问。(R/W)

I2C_FIFO_ADDR_CFG_EN 配置从机是否使能双寻址模式。当该模式使能时，在 I2C 地址字节之后接收的字节代表 I2C 从机 RAM 中的偏移地址。

0：禁用

1：使能

(R/W)

I2C_RX_FIFO_RST 配置复位 RX FIFO。

0：无效

1：复位

(R/W)

I2C_TX_FIFO_RST 配置复位 TX FIFO。

0：无效

1：复位

(R/W)

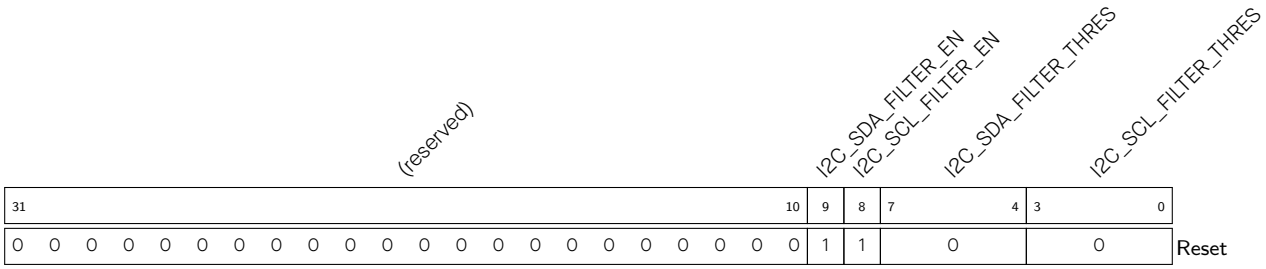
I2C_FIFO_PRT_EN 配置在直接访问模式下是否使能 FIFO 指针。该位控制有效位以及 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的中断。

0：无效

1：使能

(R/W)

Register 47.15. I2C_FILTER_CFG_REG (0x0050)



I2C_SCL_FILTER_THRES 配置 SCL 过滤脉冲宽度的阈值。当 SCL 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SDA_FILTER_THRES 配置 SDA 过滤脉冲宽度的阈值。当 SDA 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SCL_FILTER_EN 配置使能 SCL 的滤波功能。

0：无效

1：复位

(R/W)

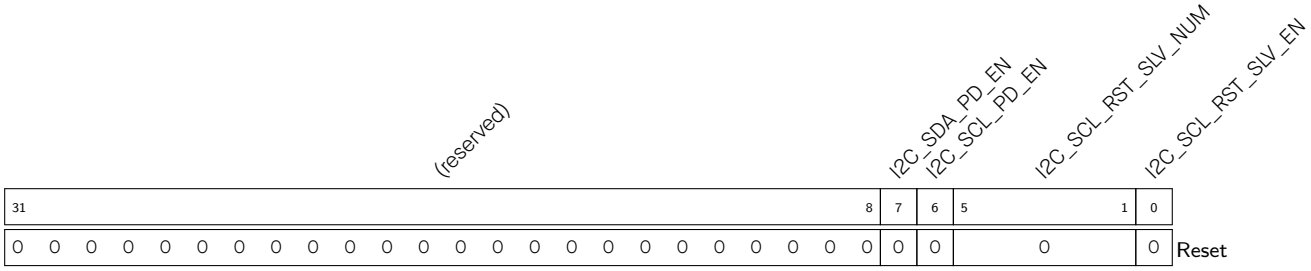
I2C_SDA_FILTER_EN 配置使能 SDA 的滤波功能。

0：无效

1：复位

(R/W)

Register 47.16. I2C_SCL_SP_CONF_REG (0x0080)



I2C_SCL_RST_SLV_EN 在 I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 I2C_SCL_RST_SLV_NUM[4:0]。

0：无效

1：发送 SCL 脉冲

(R/W/SC)

I2C_SCL_RST_SLV_NUM 配置 I2C 主机模式下生成的 SCL 脉冲。

仅在 I2C_SCL_RST_SLV_EN 为 1 时有效。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SCL_PD_EN 配置降低 I2C SCL 输出功耗。

0：正常功耗

1：不工作，降低功耗

仅在 I2C_SCL_FORCE_OUT 为 1 时有效。

(R/W)

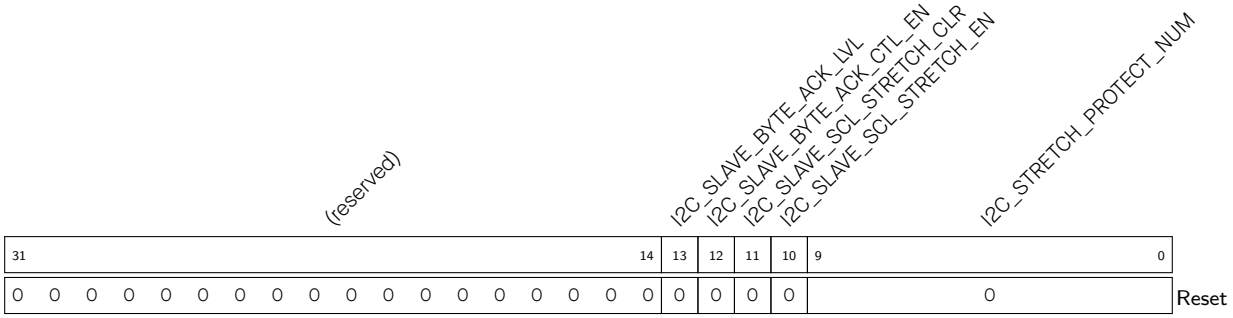
I2C_SDA_PD_EN 配置降低 I2C SDA 输出功耗。

0：正常功耗

1：不工作，降低功耗

仅在 I2C_SDA_FORCE_OUT 为 1 时有效。(R/W)

Register 47.17. I2C_SCL_STRETCH_CONF_REG (0x0084)



I2C_STRETCH_PROTECT_NUM 配置 SCL 时钟拉伸释放后的保护时间，通常设置为大于 SDA 的建立时间。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SLAVE_SCL_STRETCH_EN 配置使能从机的 SCL 时钟拉伸。I2C_SLAVE_SCL_STRETCH_EN 为 1，且出现可触发时钟拉伸的事件时，拉伸 SCL 时钟。SCL 时钟拉伸的原因可见 I2C_STRETCH_CAUSE。

0：禁用

1：使能

(R/W)

I2C_SLAVE_SCL_STRETCH_CLR 配置清除从机 SCL 时钟拉伸。

0：无效

1：清除

(WT)

I2C_SLAVE_BYTE_ACK_CTL_EN 配置使能从机控制 ACK 电平。

0：禁用

1：使能

(R/W)

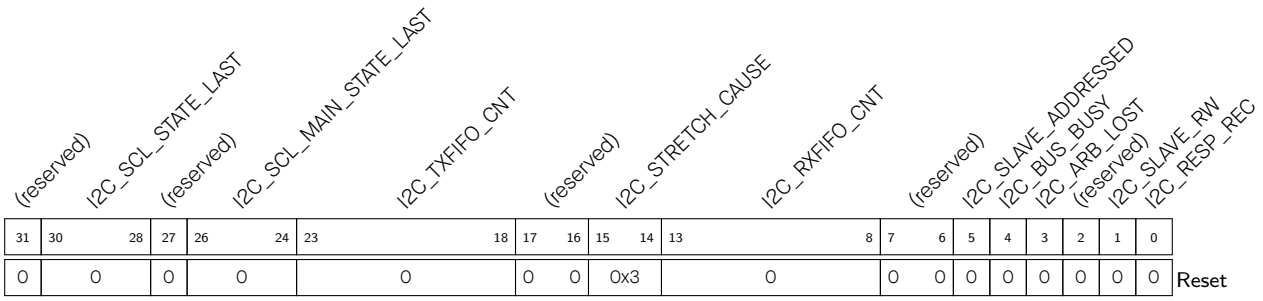
I2C_SLAVE_BYTE_ACK_LVL 从机控制 ACK 电平使能时，通过此位设置 ACK 的电平值。

0：低电平

1：高电平

(R/W)

Register 47.18. I2C_SR_REG (0x0008)



I2C_RESP_REC 表示主机模式或从机模式下接收的 ACK 电平值。

0：ACK

1：NACK

(RO)

I2C_SLAVE_RW 表示在从机模式下数据传输方向。

0：主机向从机写入数据

1：主机读取从机数据

(RO)

I2C_ARB_LOST 表示 I2C 控制器是否控制 SCL 线。

0：仲裁未丢失

1：仲裁丢失

(RO)

I2C_BUS_BUSY 表示 I2C 总线状态。

0：I2C 总线处于空闲状态

1：I2C 总线正在传输数据

(RO)

I2C_SLAVE_ADDRESSED 表示主机发送的地址是否与从机的地址相同。

仅当模块被配置为 I2C 从机时有效。

0：不相同

1：相同

(RO)

I2C_RXFIFO_CNT 表示 RAM 接收数据的字节数。(RO)

I2C_STRETCH_CAUSE 表示从机模式下 SCL 时钟拉伸的原因。

0：主机开始读取数据时拉伸 SCL 时钟

1：从机模式下 I2C TX FIFO 读空时拉伸 SCL 时钟

2：从机模式下 I2C RX FIFO 写满时拉伸 SCL 时钟

3：无效

(RO)

I2C_TXFIFO_CNT 表示需发送数据的字节数。(RO)

见下页...

Register 47.18. I2C_SR_REG (0x0008)

... 接上页

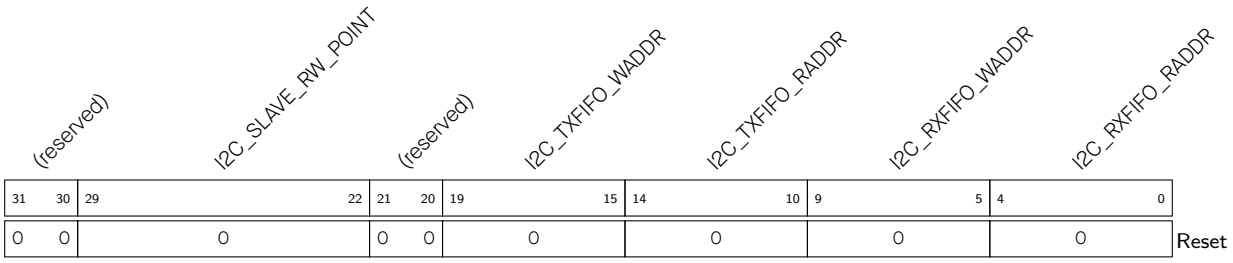
I2C_SCL_MAIN_STATE_LAST 表示 I2C 控制器状态机的状态。

0：空闲
1：地址偏移
2：ACK 地址
3：接收数据
4：发送数据
5：发送 ACK
6：等待 ACK
7：无效
(RO)

I2C_SCL_STATE_LAST 表示用于生成 SCL 的状态机状态。

0：空闲
1：开始
2：下降沿
3：低电平
4：上升沿
5：高电平
6：停止
7：无效
(RO)

Register 47.19. I2C_FIFO_ST_REG (0x0014)



I2C_RXFIFO_RADDR 表示 APB 总线读 RX FIFO 的偏移地址。(RO)

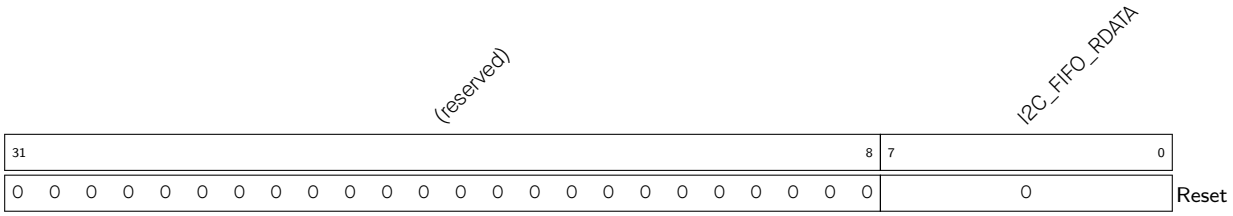
I2C_RXFIFO_WADDR 表示 I2C 控制器接收数据和写 RX FIFO 的偏移地址。(RO)

I2C_TXFIFO_RADDR 表示 I2C 控制器读 TX FIFO 的偏移地址。(RO)

I2C_TXFIFO_WADDR 表示 APB 总线写 TX FIFO 的偏移地址。(RO)

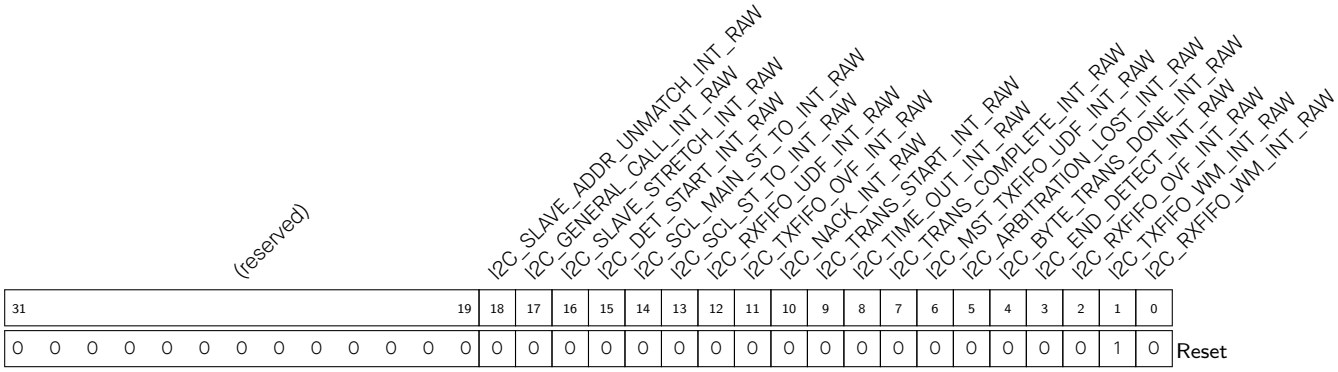
I2C_SLAVE_RW_POINT 表示在 I2C 从机模式下由 I2C 主机寻址的 I2C 从机 RAM 的偏移地址。(RO)

Register 47.20. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA 表示 RX FIFO 读取数据的值。(RO)

Register 47.21. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW I2C_RXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW I2C_TXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW I2C_RXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_END_DETECT_INT_RAW I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW I2C_ARBITRATION_LOST_INT 的原始中断状态。(R/SS/WTC)

见下页...

Register 47.21. I2C_INT_RAW_REG (0x0020)

... 接上页

I2C_MST_TXFIFO_UDF_INT_RAW I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

I2C_TIME_OUT_INT_RAW I2C_TIME_OUT_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_START_INT_RAW I2C_TRANS_START_INT 的原始中断状态。(R/SS/WTC)

I2C_NACK_INT_RAW I2C_SLAVE_STRETCH_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW I2C_TXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW I2C_RXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW I2C_SCL_ST_TO_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW I2C_SCL_MAIN_ST_TO_INT 的原始中断状态。(R/SS/WTC)

I2C_DET_START_INT_RAW I2C_DET_START_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_STRETCH_INT_RAW I2C_SLAVE_STRETCH_INT 的原始中断状态。(R/SS/WTC)

I2C_GENERAL_CALL_INT_RAW I2C_GENARAL_CALL_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_ADDR_UNMATCH_INT_RAW I2C_SLAVE_ADDR_UNMATCH_INT 的原始中断状态。(R/SS/WTC)

Register 47.22. I2C_INT_CLR_REG (0x0024)

Bits	Field	Reset
31:19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	I2C_SLAVE_ADDR_UNMATCH_INT_CLR	0
17	I2C_GENERAL_CALL_INT_CLR	0
16	I2C_SLAVE_STRETCH_INT_CLR	0
15	I2C_DET_START_INT_CLR	0
14	I2C_SCL_MAIN_ST_TO_INT_CLR	0
13	I2C_SCL_ST_TO_INT_CLR	0
12	I2C_RXFIFO_UDF_INT_CLR	0
11	I2C_TXFIFO_OVF_INT_CLR	0
10	I2C_NACK_INT_CLR	0
9	I2C_TRANS_START_INT_CLR	0
8	I2C_TIME_OUT_INT_CLR	0
7	I2C_TRANS_COMPLETE_INT_CLR	0
6	I2C_MST_TXFIFO_UDF_INT_CLR	0
5	I2C_ARBITRATION_LOST_INT_CLR	0
4	I2C_BYTE_TRANS_DONE_INT_CLR	0
3	I2C_END_DETECT_INT_CLR	0
2	I2C_RXFIFO_OVF_INT_CLR	0
1	I2C_TXFIFO_WM_INT_CLR	0
0	I2C_RXFIFO_WM_INT_CLR	0

I2C_RXFIFO_WM_INT_CLR 写 1 清除 I2C_RXFIFO_WM_INT 中断。(WT)

I2C_TXFIFO_WM_INT_CLR 写 1 清除 I2C_TXFIFO_WM_INT 中断。(WT)

I2C_RXFIFO_OVF_INT_CLR 写 1 清除 I2C_RXFIFO_OVF_INT 中断。(WT)

I2C_END_DETECT_INT_CLR 写 1 清除 I2C_END_DETECT_INT 中断。(WT)

I2C_BYTE_TRANS_DONE_INT_CLR 写 1 清除 I2C_BYTE_TRANS_DONE_INT 中断。(WT)

I2C_ARBITRATION_LOST_INT_CLR 写 1 清除 I2C_ARBITRATION_LOST_INT 中断。(WT)

I2C_MST_TXFIFO_UDF_INT_CLR 写 1 清除 I2C_MST_TXFIFO_UDF_INT 中断。(WT)

I2C_TRANS_COMPLETE_INT_CLR 写 1 清除 I2C_TRANS_COMPLETE_INT 中断。(WT)

I2C_TIME_OUT_INT_CLR 写 1 清除 I2C_TIME_OUT_INT 中断。(WT)

I2C_TRANS_START_INT_CLR 写 1 清除 I2C_TRANS_START_INT 中断。(WT)

I2C_NACK_INT_CLR 写 1 清除 I2C_NACK_INT 中断。(WT)

I2C_TXFIFO_OVF_INT_CLR 写 1 清除 I2C_TXFIFO_OVF_INT 中断。(WT)

I2C_RXFIFO_UDF_INT_CLR 写 1 清除 I2C_RXFIFO_UDF_INT 中断。(WT)

I2C_SCL_ST_TO_INT_CLR 写 1 清除 I2C_SCL_ST_TO_INT 中断。(WT)

I2C_SCL_MAIN_ST_TO_INT_CLR 写 1 清除 I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

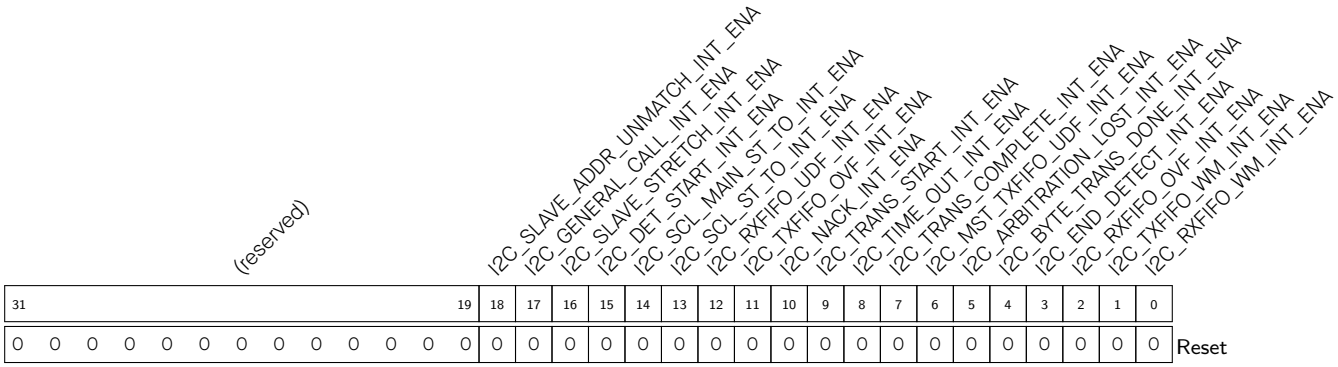
I2C_DET_START_INT_CLR 写 1 清除 I2C_DET_START_INT 中断。(WT)

I2C_SLAVE_STRETCH_INT_CLR 写 1 清除 I2C_SLAVE_STRETCH_INT 中断。(WT)

I2C_GENERAL_CALL_INT_CLR 写 1 清除 I2C_GENARAL_CALL_INT 中断。(WT)

I2C_SLAVE_ADDR_UNMATCH_INT_CLR 写 1 清除 I2C_SLAVE_ADDR_UNMATCH_INT 中断。(WT)

Register 47.23. I2C_INT_ENA_REG (0x0028)



I2C_RXFIFO_WM_INT_ENA 写 1 使能 I2C_RXFIFO_WM_INT 中断。(R/W)

I2C_TXFIFO_WM_INT_ENA 写 1 使能 I2C_TXFIFO_WM_INT 中断。(R/W)

I2C_RXFIFO_OVF_INT_ENA 写 1 使能 I2C_RXFIFO_OVF_INT 中断。(R/W)

I2C_END_DETECT_INT_ENA 写 1 使能 I2C_END_DETECT_INT 中断。(R/W)

I2C_BYTE_TRANS_DONE_INT_ENA 写 1 使能 I2C_END_DETECT_INT 中断。(R/W)

I2C_ARBITRATION_LOST_INT_ENA 写 1 使能 I2C_ARBITRATION_LOST_INT 中断。(R/W)

I2C_MST_TXFIFO_UDF_INT_ENA 写 1 使能 I2C_TRANS_COMPLETE_INT 中断。(R/W)

I2C_TRANS_COMPLETE_INT_ENA 写 1 使能 I2C_TRANS_COMPLETE_INT 中断。(R/W)

I2C_TIME_OUT_INT_ENA 写 1 使能 I2C_TIME_OUT_INT 中断。(R/W)

I2C_TRANS_START_INT_ENA 写 1 使能 I2C_TRANS_START_INT 中断。(R/W)

I2C_NACK_INT_ENA 写 1 使能 I2C_SLAVE_STRETCH_INT 中断。(R/W)

I2C_TXFIFO_OVF_INT_ENA 写 1 使能 I2C_TXFIFO_OVF_INT 中断。(R/W)

I2C_RXFIFO_UDF_INT_ENA 写 1 使能 I2C_RXFIFO_UDF_INT 中断。(R/W)

I2C_SCL_ST_TO_INT_ENA 写 1 使能 I2C_SCL_ST_TO_INT 中断。(R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA 写 1 使能 I2C_SCL_MAIN_ST_TO_INT 中断。(R/W)

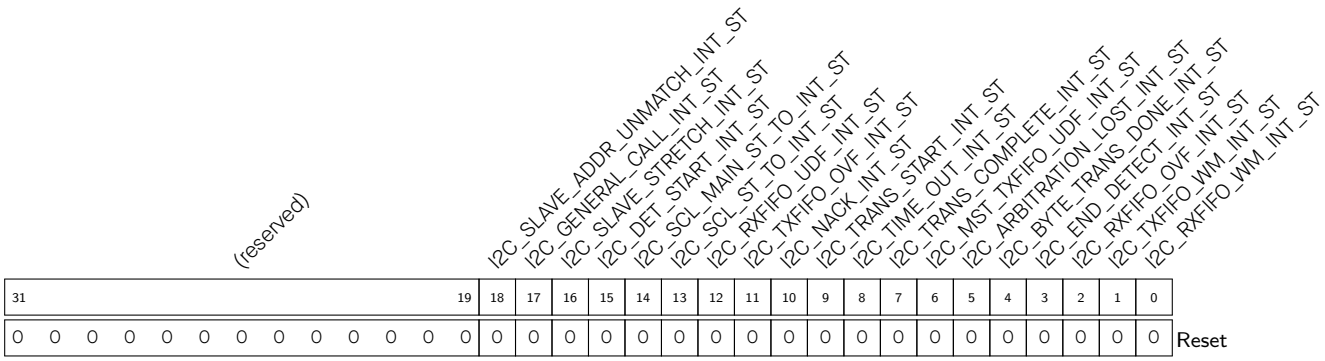
I2C_DET_START_INT_ENA 写 1 使能 I2C_DET_START_INT 中断。(R/W)

I2C_SLAVE_STRETCH_INT_ENA 写 1 使能 I2C_SLAVE_STRETCH_INT 中断。(R/W)

I2C_GENERAL_CALL_INT_ENA 写 1 使能 I2C_GENARAL_CALL_INT 中断。(R/W)

I2C_SLAVE_ADDR_UNMATCH_INT_ENA 写 1 使能 I2C_SLAVE_ADDR_UNMATCH_INT 中断。(R/W)

Register 47.24. I2C_INT_STATUS_REG (0x002C)



I2C_RXFIFO_WM_INT_ST I2C_RXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_WM_INT_ST I2C_TXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_RXFIFO_OVF_INT_ST I2C_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

I2C_END_DETECT_INT_ST I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

I2C_BYTE_TRANS_DONE_INT_ST I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

I2C_ARBITRATION_LOST_INT_ST I2C_ARBITRATION_LOST_INT 的屏蔽中断状态。(RO)

I2C_MST_TXFIFO_UDF_INT_ST I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

I2C_TRANS_COMPLETE_INT_ST I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

I2C_TIME_OUT_INT_ST I2C_TIME_OUT_INT 的屏蔽中断状态。(RO)

I2C_TRANS_START_INT_ST I2C_TRANS_START_INT 的屏蔽中断状态。(RO)

I2C_NACK_INT_ST I2C_SLAVE_STRETCH_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_OVF_INT_ST I2C_TXFIFO_OVF_INT 的屏蔽中断状态。(RO)

见下页...

Register 47.24. I2C_INT_RAW_REG (0x0020)

... 接上页

I2C_RXFIFO_UDF_INT_ST I2C_RXFIFO_UDF_INT 的屏蔽中断状态。(RO)

I2C_SCL_ST_TO_INT_ST I2C_SCL_ST_TO_INT 的屏蔽中断状态。(RO)

I2C_SCL_MAIN_ST_TO_INT_ST I2C_SCL_MAIN_ST_TO_INT 的屏蔽中断状态。(RO)

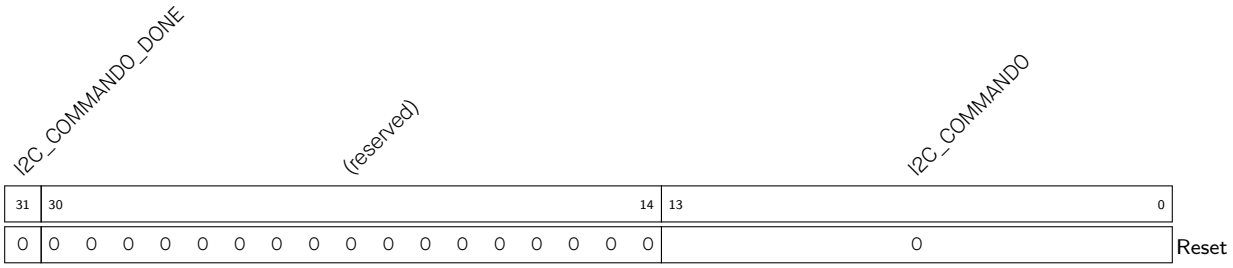
I2C_DET_START_INT_ST I2C_DET_START_INT 的屏蔽中断状态。(RO)

I2C_SLAVE_STRETCH_INT_ST I2C_SLAVE_STRETCH_INT 的屏蔽中断状态。(RO)

I2C_GENERAL_CALL_INT_ST I2C_GENARAL_CALL_INT 的屏蔽中断状态。(RO)

I2C_SLAVE_ADDR_UNMATCH_INT_ST I2C_SLAVE_ADDR_UNMATCH_INT 的屏蔽中断状态。(RO)

Register 47.25. I2C_COMD0_REG (0x0058)



I2C_COMMAND0 配置命令寄存器 0 的内容。

命令寄存器包含三个部分：

op_code 表示命令：

1：WRITE

2：STOP

3：READ

4：END

6：RSTART

Byte_num 表示需要发送或接收数据的字节数。

ack_check_en, ack_exp 和 ack 用于控制 ACK 位。详细信息，请参阅 I2C CMD 结构 47.4-2 。

(R/W)

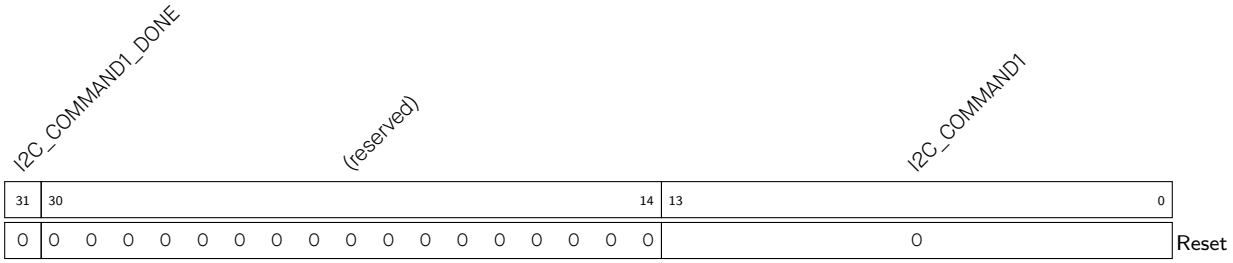
I2C_COMMAND0_DONE 表示在 I2C 主机模式下，命令 0 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.26. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 配置命令寄存器 1 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

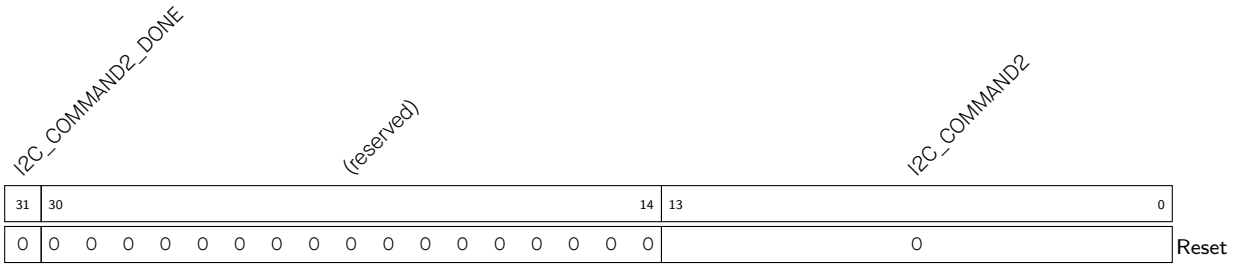
I2C_COMMAND1_DONE 表示在 I2C 主机模式下，命令 1 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.27. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 配置命令寄存器 2 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

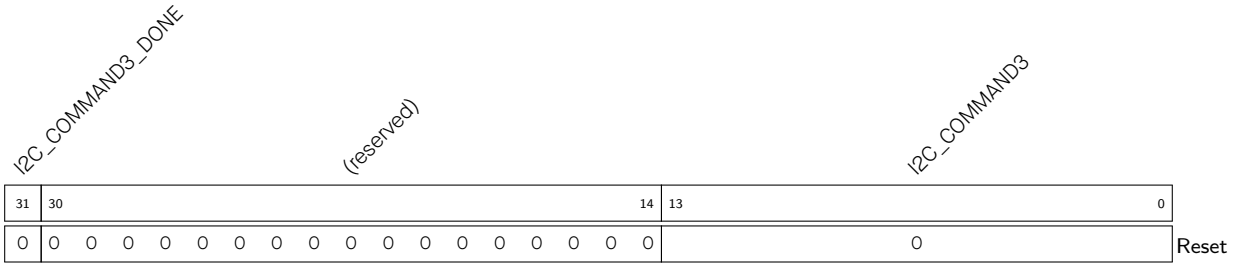
I2C_COMMAND2_DONE 表示在 I2C 主机模式下，命令 2 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.28. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 配置命令寄存器 3 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

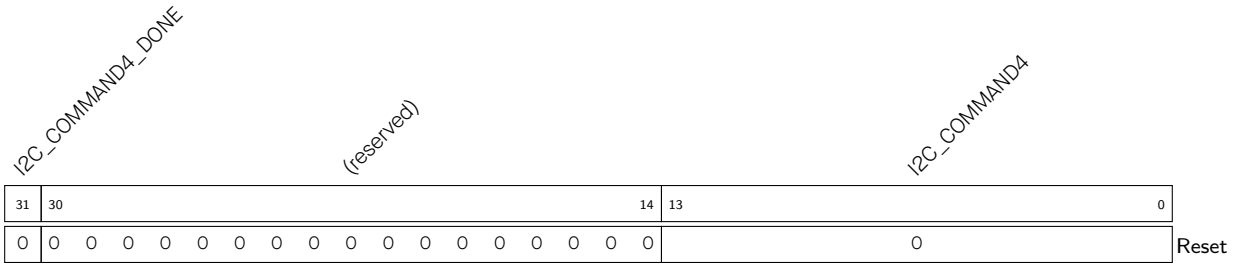
I2C_COMMAND3_DONE 表示在 I2C 主机模式下，命令 3 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.29. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 配置命令寄存器 4 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

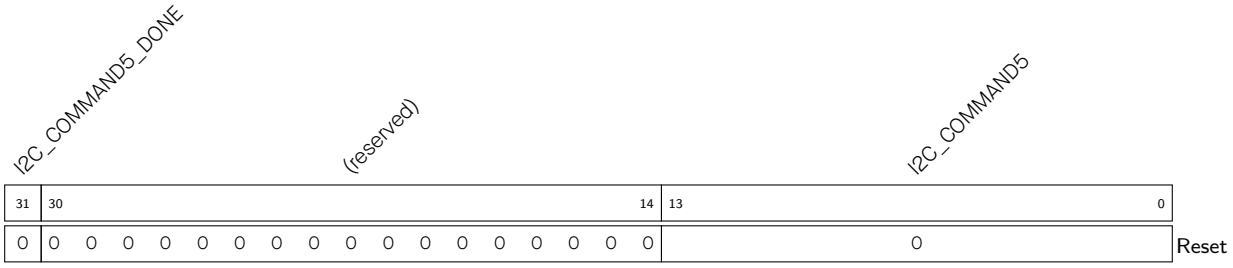
I2C_COMMAND4_DONE 表示在 I2C 主机模式下，命令 4 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.30. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 配置命令寄存器 5 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

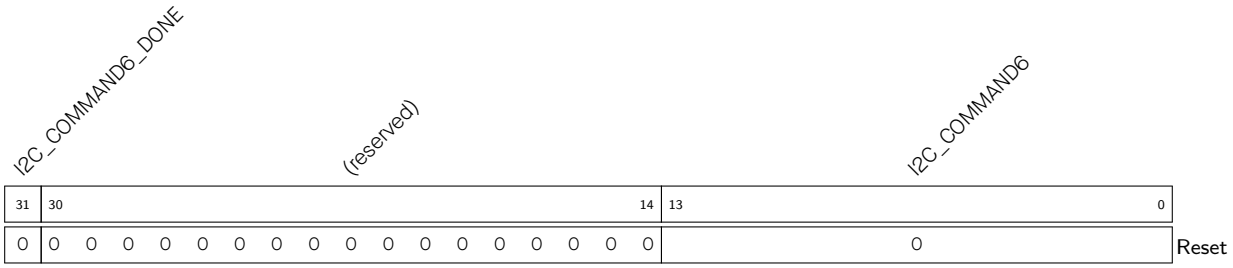
I2C_COMMAND5_DONE 表示在 I2C 主机模式下，命令 5 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.31. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 配置命令寄存器 6 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

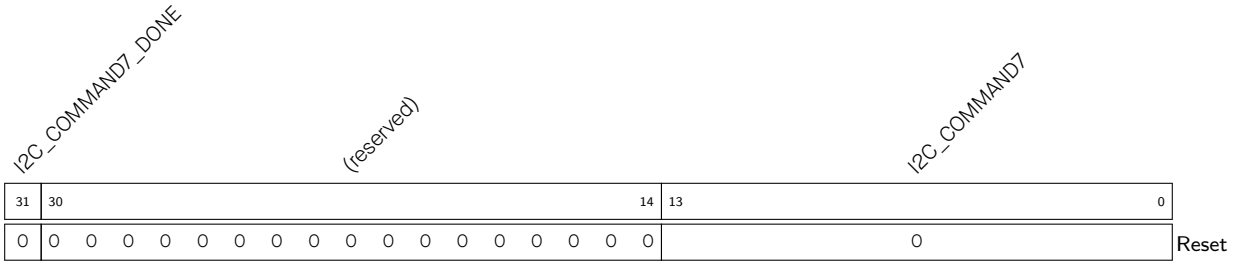
I2C_COMMAND6_DONE 表示在 I2C 主机模式下，命令 6 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.32. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 配置命令寄存器 7 的内容。具体可参考 I2C_CMD0_REG[13:0]。I2C_CMD0_REG[13:0]. (R/W)

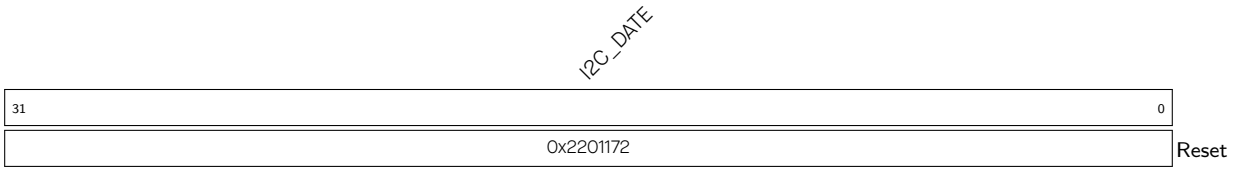
I2C_COMMAND7_DONE 表示在 I2C 主机模式下，命令 7 是否已完成。

0：未完成

1：已完成

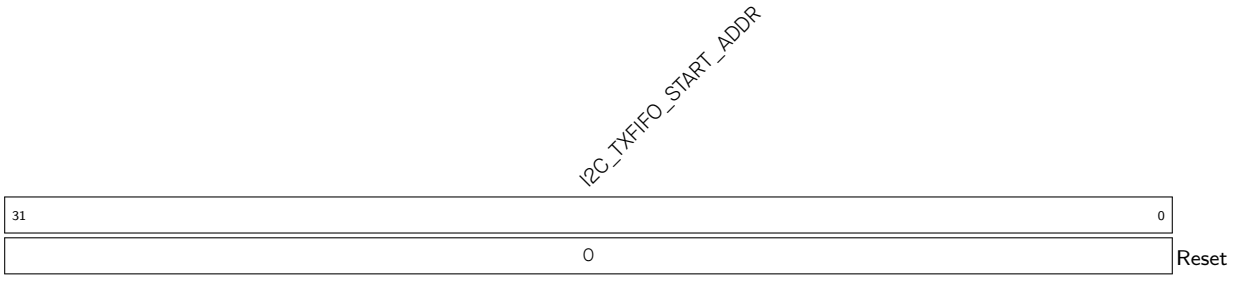
(R/W/SS)

Register 47.33. I2C_DATE_REG (0x00F8)



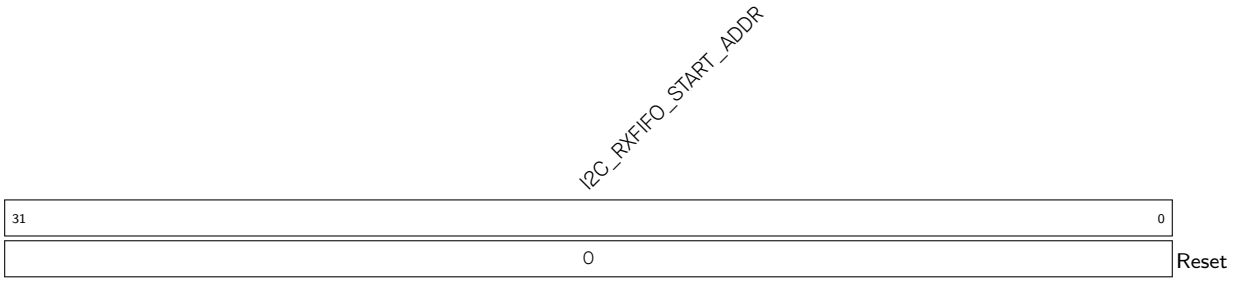
I2C_DATE 版本控制寄存器。(R/W)

Register 47.34. I2C_TXFIFO_START_ADDR_REG (0x0100)



I2C_TXFIFO_START_ADDR 表示 I2C TX FIFO 首地址。(RO)

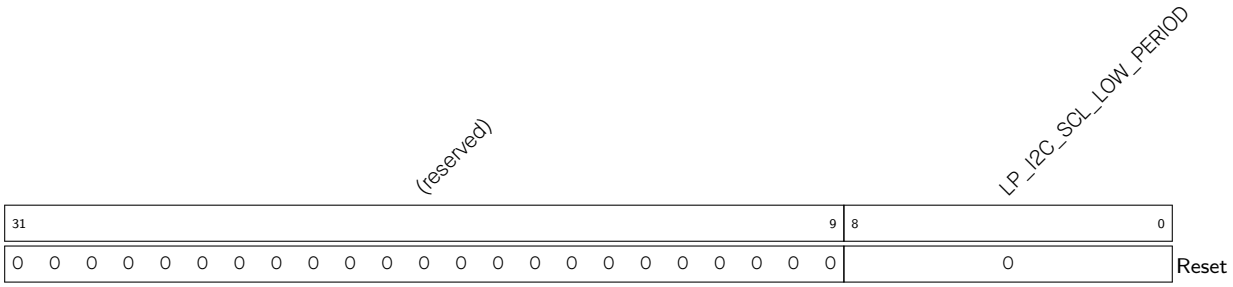
Register 47.35. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR 表示 I2C RX FIFO 首地址。(RO)

47.9.2 LP_I2C 寄存器

Register 47.36. LP_I2C_SCL_LOW_PERIOD_REG (0x0000)

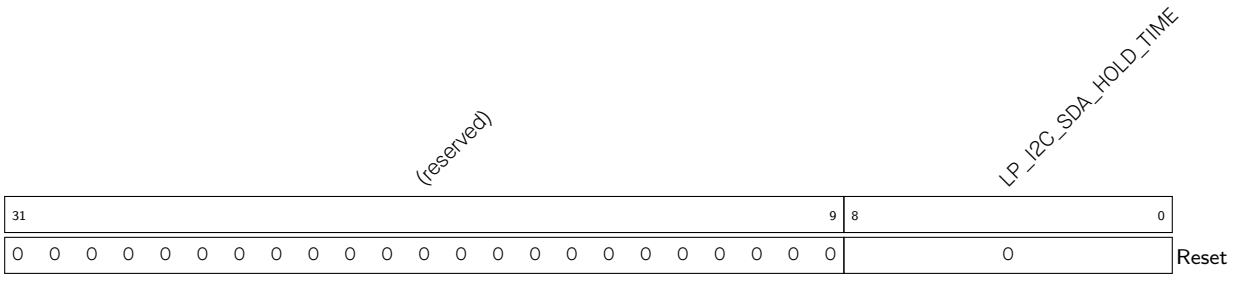


LP_I2C_SCL_LOW_PERIOD 配置主机模式下 SCL 低电平的保持时间。

单位：i2c_sclk

(R/W)

Register 47.37. LP_I2C_SDA_HOLD_REG (0x0030)

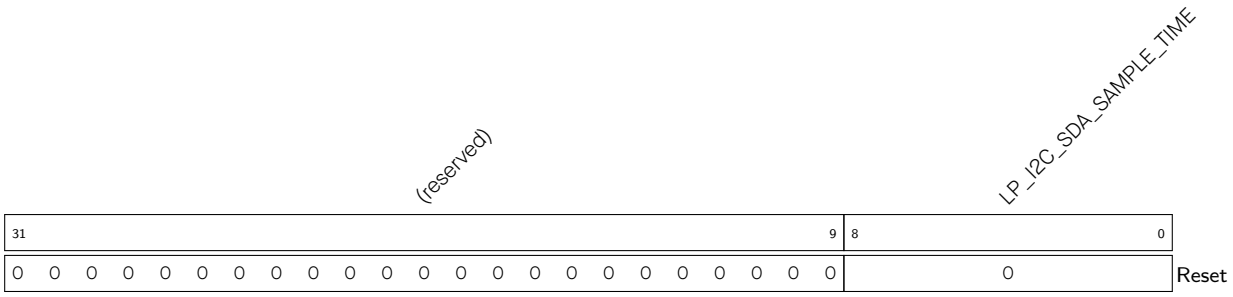


LP_I2C_SDA_HOLD_TIME 配置 SCL 下降沿后的数据保持时间。

单位：i2c_sclk

(R/W)

Register 47.38. LP_I2C_SDA_SAMPLE_REG (0x0034)

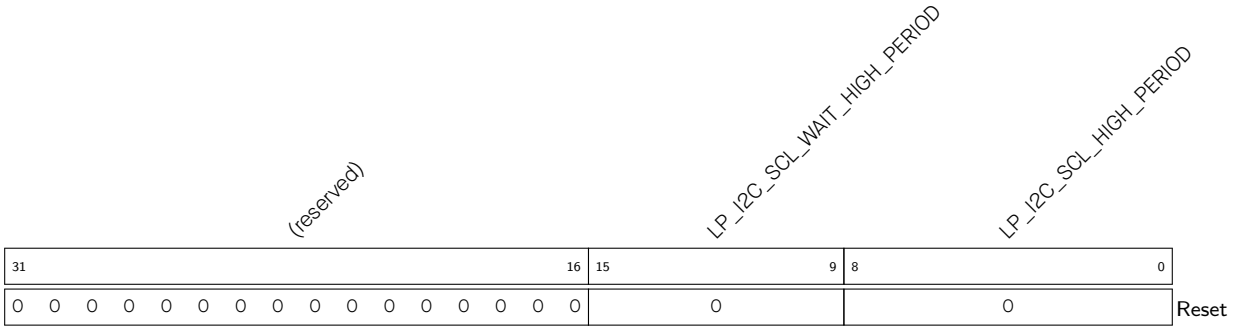


LP_I2C_SDA_SAMPLE_TIME 配置采样 SDA 的时间。

单位：i2c_sclk

(R/W)

Register 47.39. LP_I2C_SCL_HIGH_PERIOD_REG (0x0038)



LP_I2C_SCL_HIGH_PERIOD 配置 SCL 在主机模式下保持高电平的时间。

单位：i2c_sclk

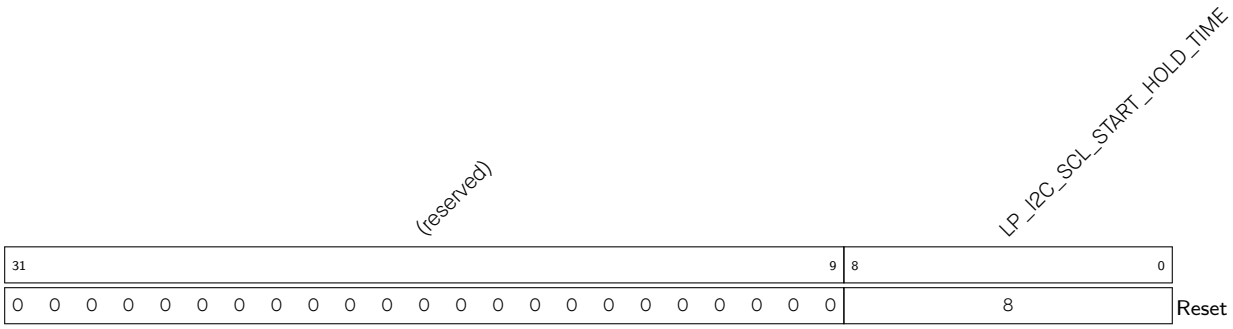
(R/W)

LP_I2C_SCL_WAIT_HIGH_PERIOD 配置 SCL_FSM 等待 SCL 在主机模式下翻转至高电平的时间。

单位：i2c_sclk

(R/W)

Register 47.40. LP_I2C_SCL_START_HOLD_REG (0x0040)

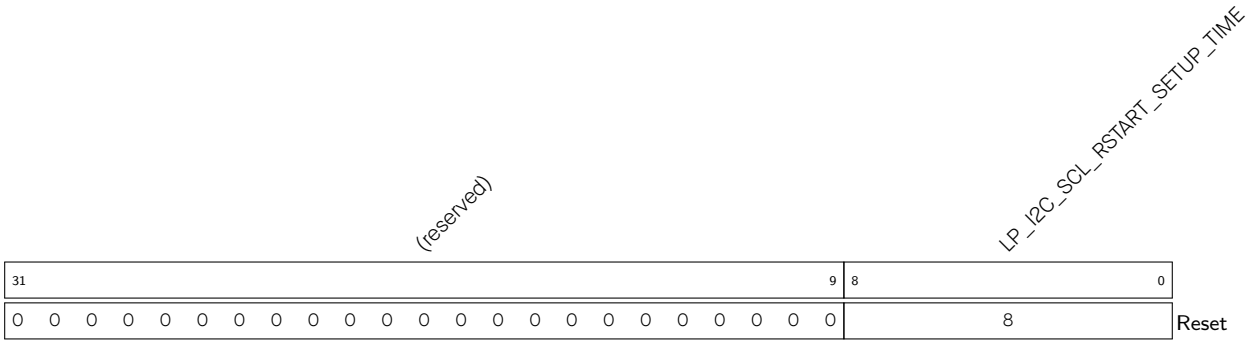


LP_I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间。

单位：i2c_sclk

(R/W)

Register 47.41. LP_I2C_SCL_RSTART_SETUP_REG (0x0044)

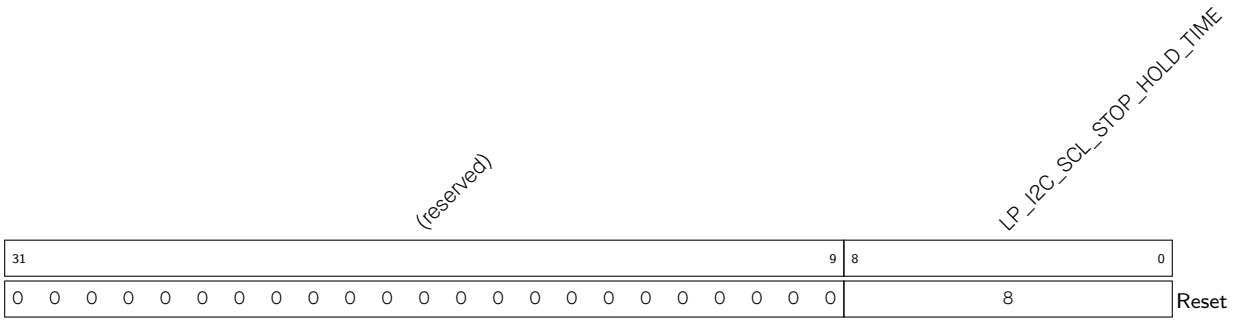


LP_I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间。

单位：i2c_sclk

(R/W)

Register 47.42. LP_I2C_SCL_STOP_HOLD_REG (0x0048)

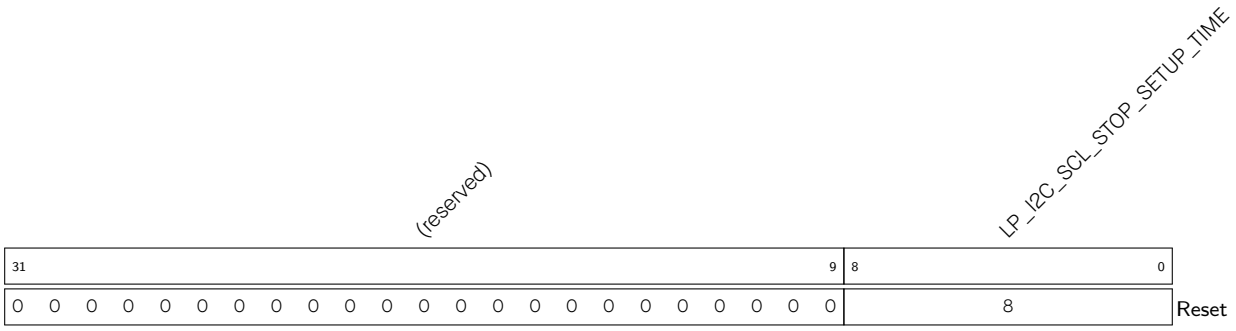


LP_I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟时间。

单位：i2c_sclk

(R/W)

Register 47.43. LP_I2C_SCL_STOP_SETUP_REG (0x004C)

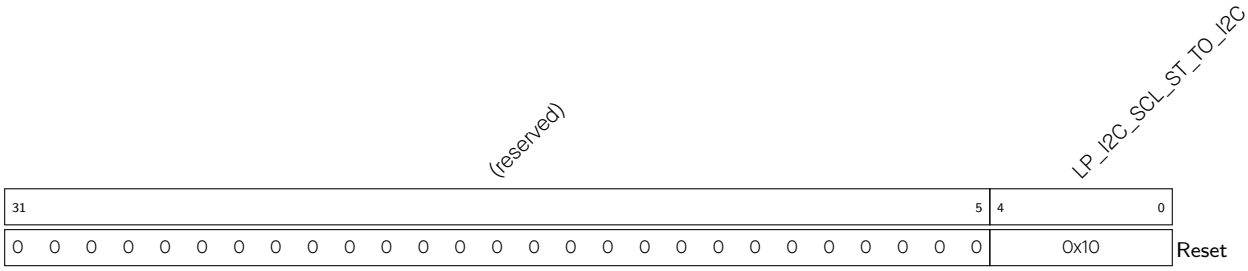


LP_I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间。

单位：i2c_sclk

(R/W)

Register 47.44. LP_I2C_SCL_ST_TIME_OUT_REG (0x0078)

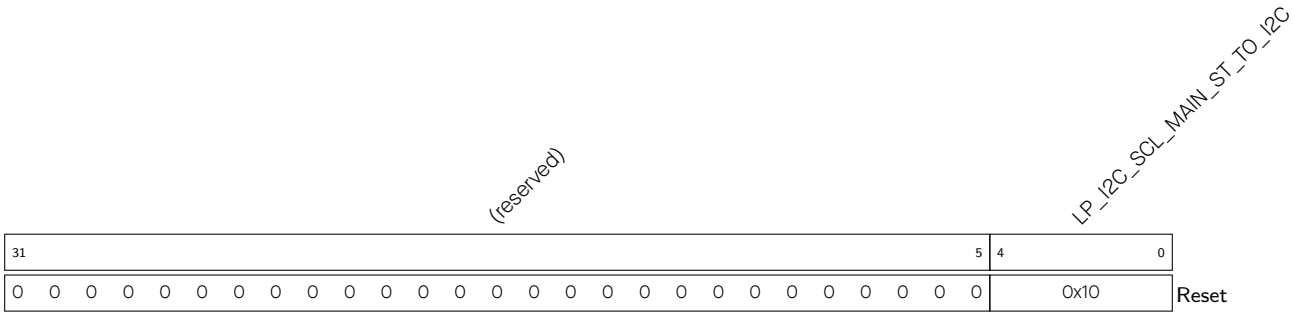


LP_I2C_SCL_ST_TO_I2C 配置 SCL_FSM 状态保持不变的最大时间，不能大于 23。

单位：i2c_sclk

(R/W)

Register 47.45. LP_I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

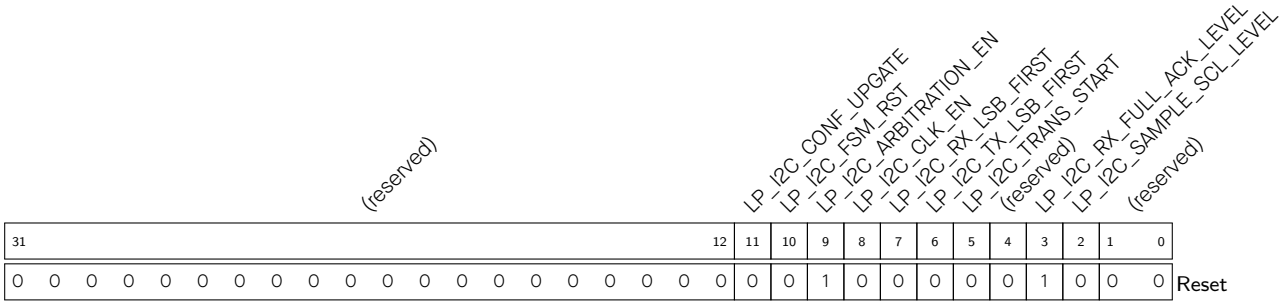


LP_I2C_SCL_MAIN_ST_TO_I2C 配置 SCL_MAIN_FSM 状态保持不变的最大时间，不能大于 23。

单位：i2c_sclk

(R/W)

Register 47.46. LP_I2C_CTR_REG (0x0004)



LP_I2C_SAMPLE_SCL_LEVEL 配置 SDA 的采样模式。

0: SCL 为高电平时采样 SDA 数据

1: SCL 为低电平时采样 SDA 数据

(R/W)

LP_I2C_RX_FULL_ACK_LEVEL 用于配置主机在 rx_fifo_cnt 达到阈值时需发送的 ACK 电平值。(R/W)

LP_I2C_TRANS_START 配置从机是否开始发送 TX FIFO 中的数据。

0：无效

1：开始

(WT)

LP_I2C_TX_LSB_FIRST 控制待发送数据的发送顺序。

0：从最高有效位开始发送数据

1：从最低有效位开始发送数据

(R/W)

LP_I2C_RX_LSB_FIRST 控制接收数据的存储顺序。

0：从最高有效位开始接收数据

1：从最低有效位开始接收数据

(R/W)

LP_I2C_CLK_EN 门控寄存器的时钟信号。

0：仅当通过软件读写寄存器时支持时钟

1：强制寄存器时钟开启。

(R/W)

Continued on the next page...

Register 47.46. I2C_CTR_REG (0x0004)

Continued from the previous page...

LP_I2C_ARBITRATION_EN 配置使能 I2C 总线仲裁检测。

0：无效

1：使能

(R/W)

LP_I2C_FSM_RST 配置复位 SCL_FSM。

0：无效

1：复位

(WT)

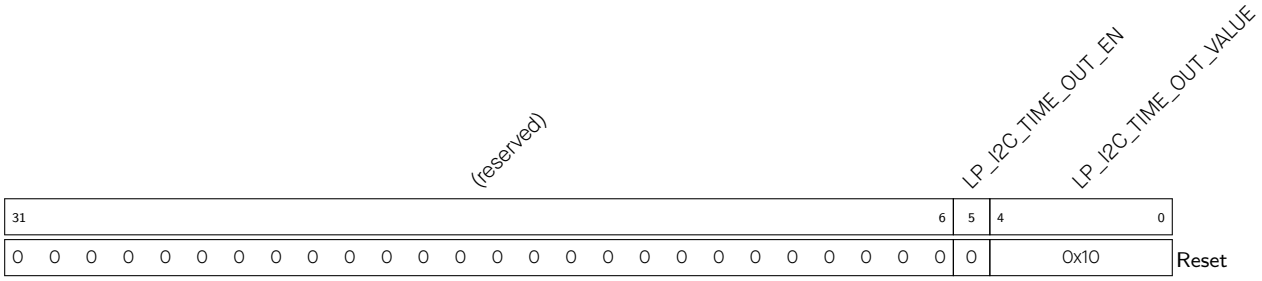
LP_I2C_CONF_UPGATE 配置此位实现同步。

0：无效

1：同步

(WT)

Register 47.47. LP_I2C_TO_REG (0x000C)



LP_I2C_TIME_OUT_VALUE 配置 SCL 在高电平或低电平停滞的超时阈值周期。实际周期是 2^(reg_time_out_value +1)–1。

单位：i2c_sclk

(R/W)

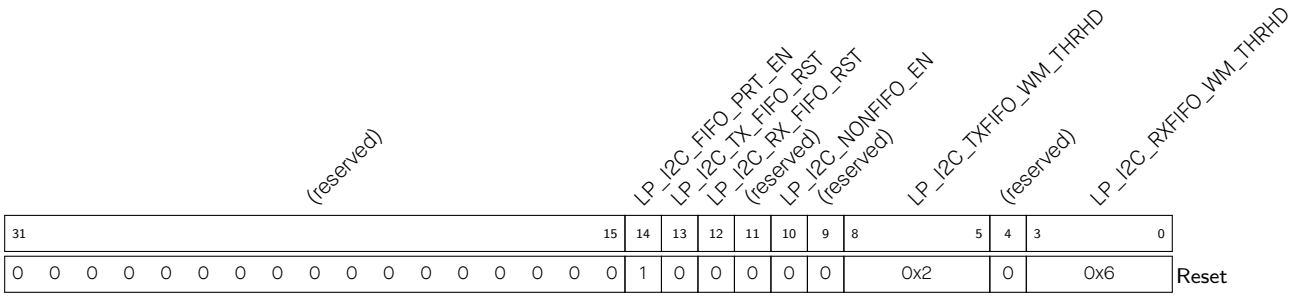
LP_I2C_TIME_OUT_EN 配置使能超时控制。

0：无效

1：使能

(R/W)

Register 47.48. LP_I2C_FIFO_CONF_REG (0x0018)



LP_I2C_RXFIFO_WM_THRHD 配置在直接访问模式下，RX FIFO 的水标阈值。当 LP_I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 LP_I2C_RXFIFO_WM_THRHD[4:0]，LP_I2C_RXFIFO_WM_INT_RAW 位有效。(R/W)

LP_I2C_TXFIFO_WM_THRHD 配置在直接访问模式下，TX FIFO 的水标阈值。当 LP_I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 LP_I2C_TXFIFO_WM_THRHD[4:0] 时，LP_I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

LP_I2C_NONFIFO_EN 配置使能 APB 直接访问。(R/W)

LP_I2C_RX_FIFO_RST 配置复位 RX FIFO。

0：无效

1：复位

(R/W)

LP_I2C_TX_FIFO_RST 配置复位 TX FIFO。

0：无效

1：复位

(R/W)

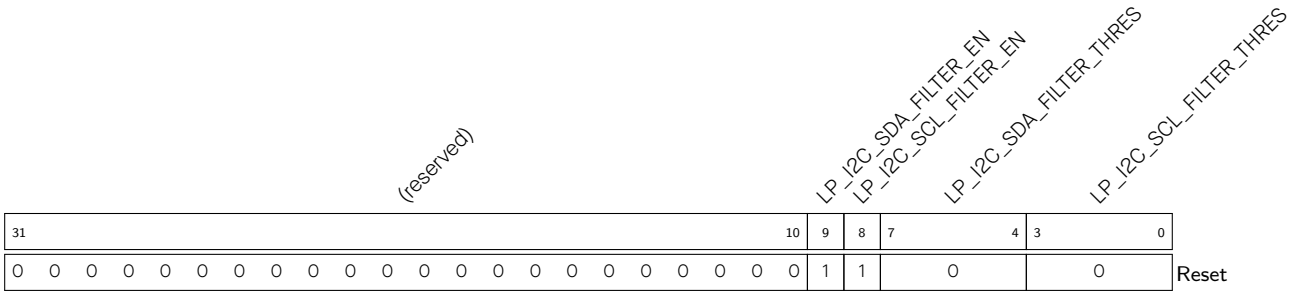
LP_I2C_FIFO_PRT_EN 配置在直接访问模式下是否使能 FIFO 指针。该位控制有效位以及 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的中断。

0：无效

1：使能

(R/W)

Register 47.49. LP_I2C_FILTER_CFG_REG (0x0050)



LP_I2C_SCL_FILTER_THRES 配置 SCL 过滤脉冲宽度的阈值。当 SCL 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：i2c_sclk

(R/W)

LP_I2C_SDA_FILTER_THRES 配置 SDA 过滤脉冲宽度的阈值。当 SDA 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：i2c_sclk

(R/W)

LP_I2C_SCL_FILTER_EN 配置使能 SCL 的滤波功能。

0：无效

1：复位

(R/W)

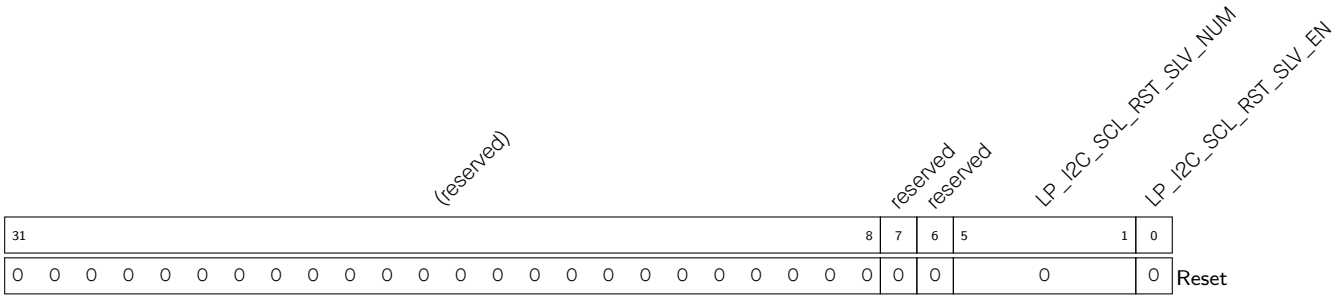
LP_I2C_SDA_FILTER_EN 配置使能 SDA 的滤波功能。

0：无效

1：复位

(R/W)

Register 47.50. LP_I2C_SCL_SP_CONF_REG (0x0080)



LP_I2C_SCL_RST_SLV_EN 在 I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 LP_I2C_SCL_RST_SLV_NUM[4:0]. (R/W/SC)

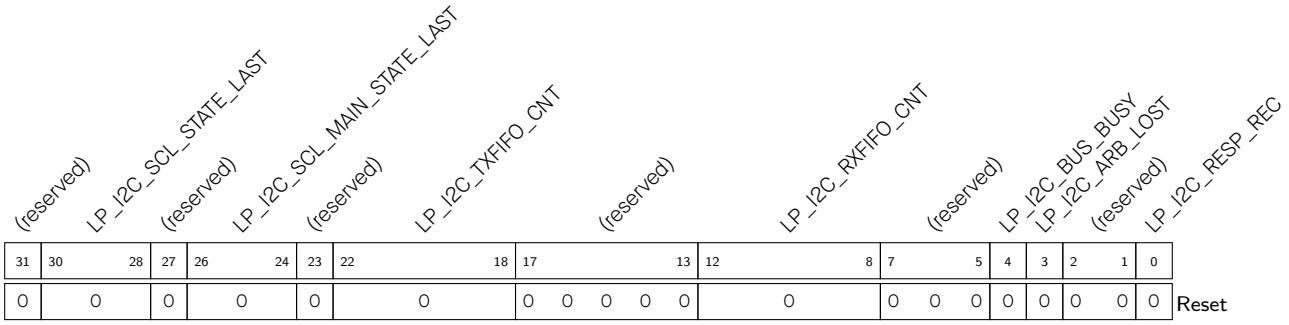
LP_I2C_SCL_RST_SLV_NUM 配置 I2C 主机模式下生成的 SCL 脉冲。

仅在 LP_I2C_SCL_RST_SLV_EN 为 1 时有效。

单位：i2c_sclk

(R/W)

Register 47.51. LP_I2C_SR_REG (0x0008)



LP_I2C_RESP_REC 表示主机模式或从机模式下接收的 ACK 电平值。

0: ACK

1: NACK

(RO)

LP_I2C_ARB_LOST 表示 I2C 控制器是否控制 SCL 线。

0: 仲裁未丢失

1: 仲裁丢失

(RO)

LP_I2C_BUS_BUSY 表示 I2C 总线状态。

0：I2C 总线处于空闲状态

1：I2C 总线正在传输数据

(RO)

LP_I2C_RXFIFO_CNT 表示 RAM 接收数据的字节数。(RO)

LP_I2C_TXFIFO_CNT 表示需发送数据的字节数。(RO)

LP_I2C_SCL_MAIN_STATE_LAST 表示 I2C 控制器状态机的状态。

0：空闲

1：地址偏移

2：ACK 地址

3：接收数据

4：发送数据

5：发送 ACK

6：等待 ACK

(RO)

LP_I2C_SCL_STATE_LAST 表示用于生成 SCL 的状态机状态。

0：空闲

1：开始

2：下降沿

3：低电平

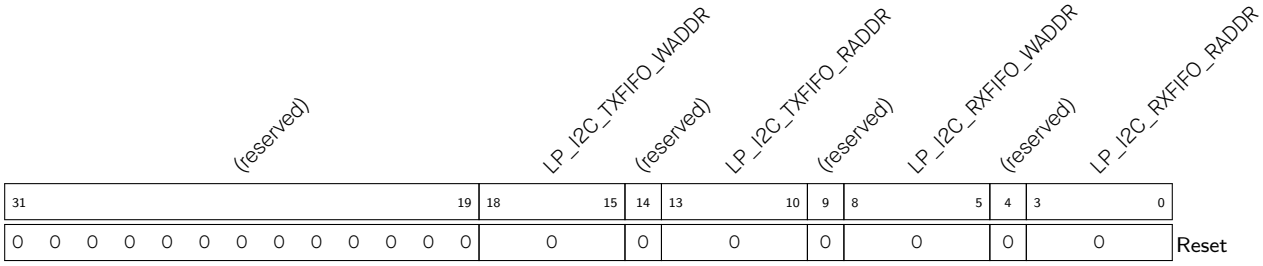
4：上升沿

5：高电平

6：停止

(RO)

Register 47.52. LP_I2C_FIFO_ST_REG (0x0014)



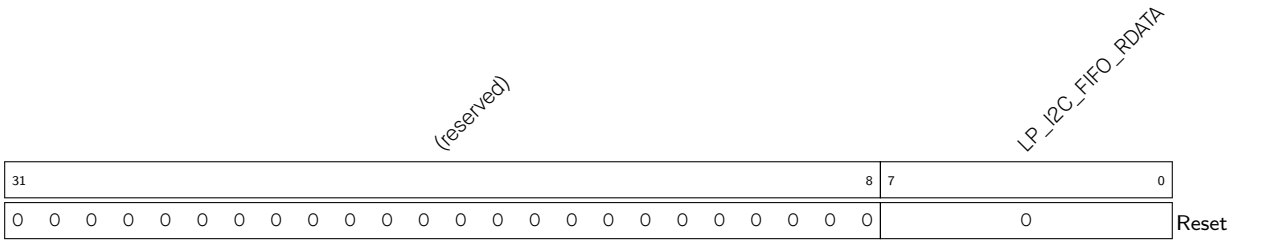
LP_I2C_RXFIFO_RADDR 表示 APB 总线读 RX FIFO 的偏移地址。(RO)

LP_I2C_RXFIFO_WADDR 表示 I2C 控制器接收数据和写 RX FIFO 的偏移地址。(RO)

LP_I2C_TXFIFO_RADDR 表示 I2C 控制器读 TX FIFO 的偏移地址。(RO)

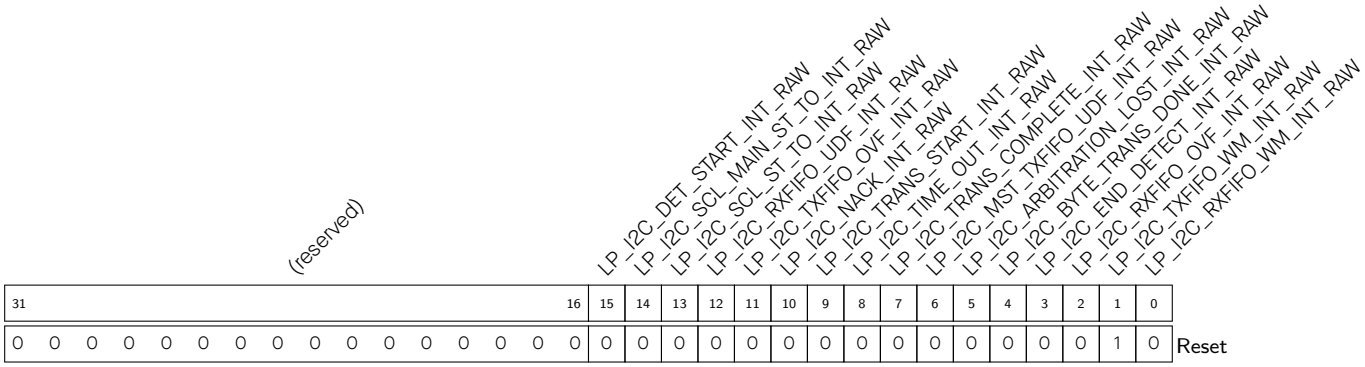
LP_I2C_TXFIFO_WADDR 表示 APB 总线写 TX FIFO 的偏移地址。(RO)

Register 47.53. LP_I2C_DATA_REG (0x001C)



LP_I2C_FIFO_RDATA 表示 RX FIFO 读取数据的值。(RO)

Register 47.54. LP_I2C_INT_RAW_REG (0x0020)



LP_I2C_RXFIFO_WM_INT_RAW LP_I2C_RXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TXFIFO_WM_INT_RAW LP_I2C_TXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_RXFIFO_OVF_INT_RAW LP_I2C_RXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_END_DETECT_INT_RAW LP_I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_BYTE_TRANS_DONE_INT_RAW LP_I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_ARBITRATION_LOST_INT_RAW LP_I2C_ARBITRATION_LOST_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_MST_TXFIFO_UDF_INT_RAW LP_I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TRANS_COMPLETE_INT_RAW LP_I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TIME_OUT_INT_RAW LP_I2C_TIME_OUT_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TRANS_START_INT_RAW LP_I2C_TRANS_START_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_NACK_INT_RAW LP_I2C_SLAVE_STRETCH_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TXFIFO_OVF_INT_RAW LP_I2C_TXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

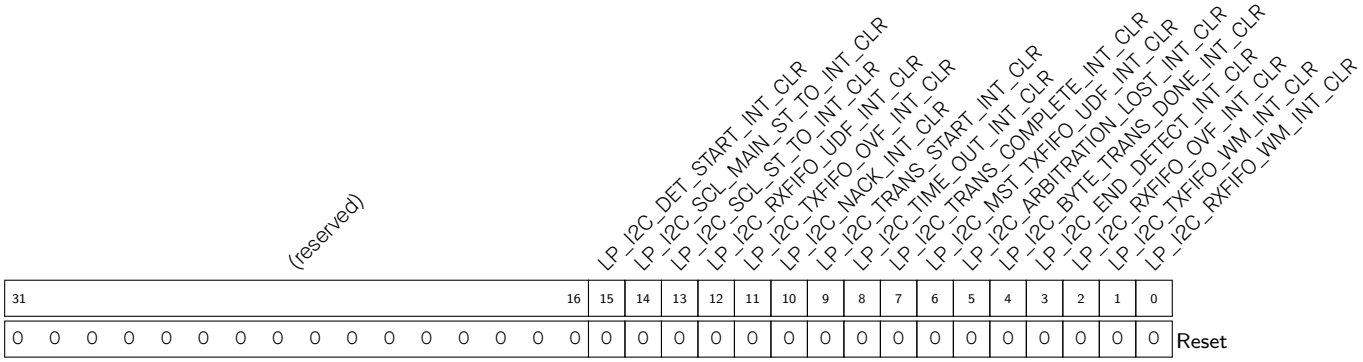
LP_I2C_RXFIFO_UDF_INT_RAW LP_I2C_RXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_SCL_ST_TO_INT_RAW LP_I2C_SCL_ST_TO_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_SCL_MAIN_ST_TO_INT_RAW LP_I2C_SCL_MAIN_ST_TO_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_DET_START_INT_RAW LP_I2C_DET_START_INT 的原始中断状态。(R/SS/WTC)

Register 47.55. LP_I2C_INT_CLR_REG (0x0024)



LP_I2C_RXFIFO_WM_INT_CLR 写 1 清除 LP_I2C_RXFIFO_WM_INT 中断。(WT)

LP_I2C_TXFIFO_WM_INT_CLR 写 1 清除 LP_I2C_TXFIFO_WM_INT 中断。(WT)

LP_I2C_RXFIFO_OVF_INT_CLR 写 1 清除 LP_I2C_RXFIFO_OVF_INT 中断。(WT)

LP_I2C_END_DETECT_INT_CLR 写 1 清除 LP_I2C_END_DETECT_INT 中断。(WT)

LP_I2C_BYTE_TRANS_DONE_INT_CLR 写 1 清除 LP_I2C_END_DETECT_INT 中断。(WT)

LP_I2C_ARBITRATION_LOST_INT_CLR 写 1 清除 LP_I2C_ARBITRATION_LOST_INT 中断。(WT)

LP_I2C_MST_TXFIFO_UDF_INT_CLR 写 1 清除 LP_I2C_TRANS_COMPLETE_INT 中断。(WT)

LP_I2C_TRANS_COMPLETE_INT_CLR 写 1 清除 LP_I2C_TRANS_COMPLETE_INT 中断。(WT)

LP_I2C_TIME_OUT_INT_CLR 写 1 清除 LP_I2C_TIME_OUT_INT 中断。(WT)

LP_I2C_TRANS_START_INT_CLR 写 1 清除 LP_I2C_TRANS_START_INT 中断。(WT)

LP_I2C_NACK_INT_CLR 写 1 清除 LP_I2C_SLAVE_STRETCH_INT 中断。(WT)

LP_I2C_TXFIFO_OVF_INT_CLR 写 1 清除 LP_I2C_TXFIFO_OVF_INT 中断。(WT)

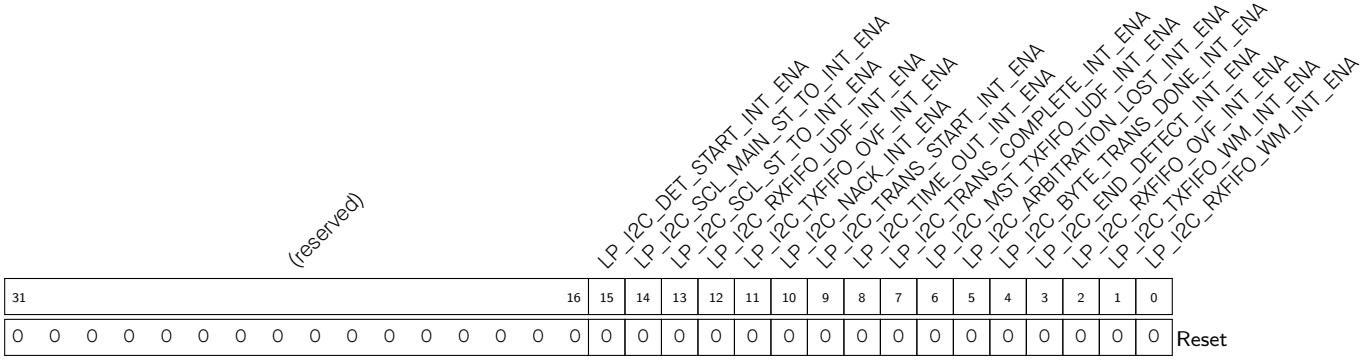
LP_I2C_RXFIFO_UDF_INT_CLR 写 1 清除 LP_I2C_RXFIFO_UDF_INT 中断。(WT)

LP_I2C_SCL_ST_TO_INT_CLR 写 1 清除 LP_I2C_SCL_ST_TO_INT 中断。(WT)

LP_I2C_SCL_MAIN_ST_TO_INT_CLR 写 1 清除 LP_I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

LP_I2C_DET_START_INT_CLR 写 1 清除 LP_I2C_DET_START_INT 中断。(WT)

Register 47.56. LP_I2C_INT_ENA_REG (0x0028)



LP_I2C_RXFIFO_WM_INT_ENA 写 1 使能 LP_I2C_RXFIFO_WM_INT 中断。(R/W)

LP_I2C_TXFIFO_WM_INT_ENA 写 1 使能 LP_I2C_TXFIFO_WM_INT 中断。(R/W)

LP_I2C_RXFIFO_OVF_INT_ENA 写 1 使能 LP_I2C_RXFIFO_OVF_INT 中断。(R/W)

LP_I2C_END_DETECT_INT_ENA 写 1 使能 LP_I2C_END_DETECT_INT 中断。(R/W)

LP_I2C_BYTE_TRANS_DONE_INT_ENA 写 1 使能 LP_I2C_END_DETECT_INT 中断。(R/W)

LP_I2C_ARBITRATION_LOST_INT_ENA 写 1 使能 LP_I2C_ARBITRATION_LOST_INT 中断。(R/W)

LP_I2C_MST_TXFIFO_UDF_INT_ENA 写 1 使能 LP_I2C_TRANS_COMPLETE_INT 中断。(R/W)

LP_I2C_TRANS_COMPLETE_INT_ENA 写 1 使能 LP_I2C_TRANS_COMPLETE_INT 中断。(R/W)

LP_I2C_TIME_OUT_INT_ENA 写 1 使能 LP_I2C_TIME_OUT_INT 中断。(R/W)

LP_I2C_TRANS_START_INT_ENA 写 1 使能 LP_I2C_TRANS_START_INT 中断。(R/W)

LP_I2C_NACK_INT_ENA 写 1 使能 LP_I2C_SLAVE_STRETCH_INT 中断。(R/W)

LP_I2C_TXFIFO_OVF_INT_ENA 写 1 使能 LP_I2C_TXFIFO_OVF_INT 中断。(R/W)

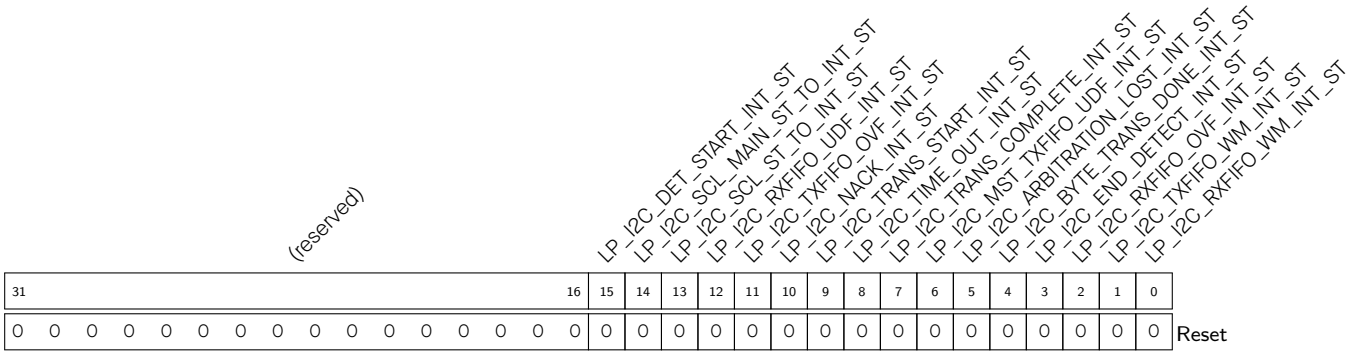
LP_I2C_RXFIFO_UDF_INT_ENA 写 1 使能 LP_I2C_RXFIFO_UDF_INT 中断。(R/W)

LP_I2C_SCL_ST_TO_INT_ENA 写 1 使能 LP_I2C_SCL_ST_TO_INT 中断。(R/W)

LP_I2C_SCL_MAIN_ST_TO_INT_ENA 写 1 使能 LP_I2C_SCL_MAIN_ST_TO_INT 中断。(R/W)

LP_I2C_DET_START_INT_ENA 写 1 使能 LP_I2C_DET_START_INT 中断。(R/W)

Register 47.57. LP_I2C_INT_STATUS_REG (0x002C)



LP_I2C_RXFIFO_WM_INT_ST LP_I2C_RXFIFO_WM_INT 的屏蔽中断状态。(RO)

LP_I2C_TXFIFO_WM_INT_ST LP_I2C_TXFIFO_WM_INT 的屏蔽中断状态。(RO)

LP_I2C_RXFIFO_OVF_INT_ST LP_I2C_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

LP_I2C_END_DETECT_INT_ST LP_I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

LP_I2C_BYTE_TRANS_DONE_INT_ST LP_I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

LP_I2C_ARBITRATION_LOST_INT_ST LP_I2C_ARBITRATION_LOST_INT 的屏蔽中断状态。(RO)

LP_I2C_MST_TXFIFO_UDF_INT_ST LP_I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

LP_I2C_TRANS_COMPLETE_INT_ST LP_I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

LP_I2C_TIME_OUT_INT_ST LP_I2C_TIME_OUT_INT 的屏蔽中断状态。(RO)

LP_I2C_TRANS_START_INT_ST LP_I2C_TRANS_START_INT 的屏蔽中断状态。(RO)

LP_I2C_NACK_INT_ST LP_I2C_SLAVE_STRETCH_INT 的屏蔽中断状态。(RO)

LP_I2C_TXFIFO_OVF_INT_ST LP_I2C_TXFIFO_OVF_INT 的屏蔽中断状态。(RO)

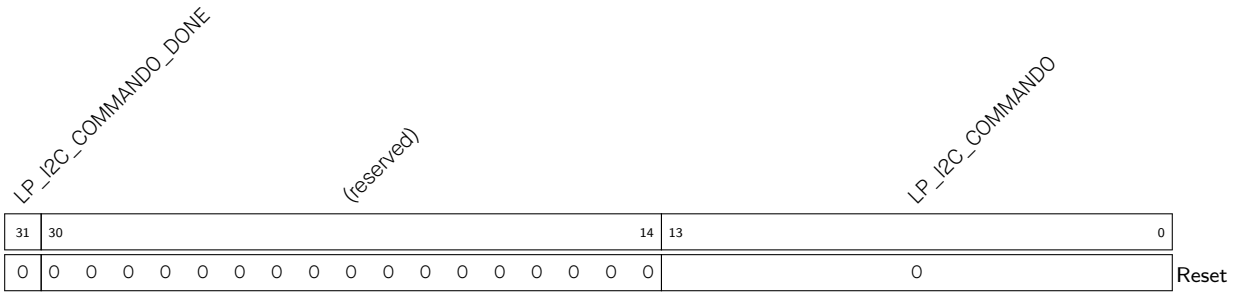
LP_I2C_RXFIFO_UDF_INT_ST LP_I2C_RXFIFO_UDF_INT 的屏蔽中断状态。(RO)

LP_I2C_SCL_ST_TO_INT_ST LP_I2C_SCL_ST_TO_INT 的屏蔽中断状态。(RO)

LP_I2C_SCL_MAIN_ST_TO_INT_ST LP_I2C_SCL_MAIN_ST_TO_INT 的屏蔽中断状态。(RO)

LP_I2C_DET_START_INT_ST LP_I2C_DET_START_INT 的屏蔽中断状态。(RO)

Register 47.58. LP_I2C_COMD0_REG (0x0058)



LP_I2C_COMMAND0 配置命令寄存器 0 的内容。

命令寄存器包含三个部分：

op_code 表示命令：

1：WRITE

2：STOP

3：READ

4：END

6：RSTART

Byte_num 表示需要发送或接收数据的字节数。

ack_check_en, ack_exp 和 ack 用于控制 ACK 位。详细信息，请参阅 I2C CMD 结构47.4-2 。

(R/W)

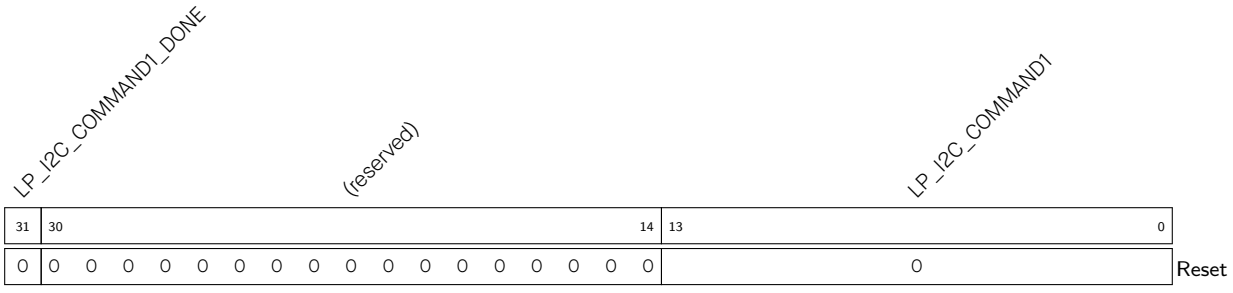
LP_I2C_COMMAND0_DONE 表示在 I2C 主机模式下，命令 0 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 47.59. LP_I2C_COMD1_REG (0x005C)



LP_I2C_COMMAND1 配置命令寄存器 1 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

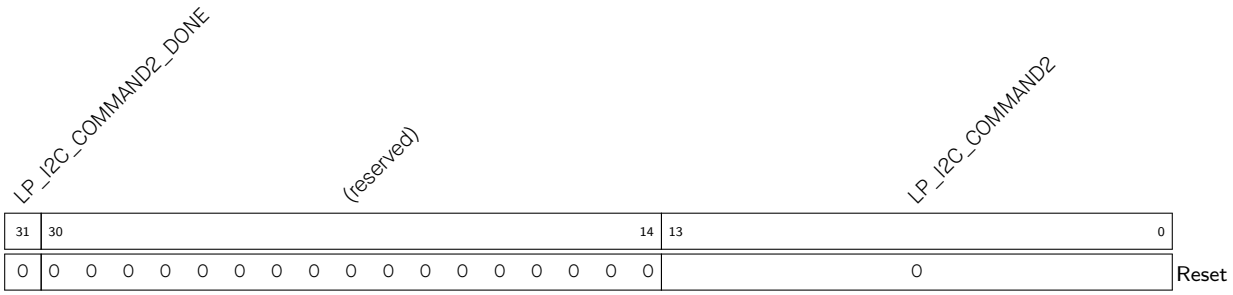
LP_I2C_COMMAND1_DONE 表示在 I2C 主机模式下，命令 1 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 47.60. LP_I2C_COMD2_REG (0x0060)



LP_I2C_COMMAND2 配置命令寄存器 2 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

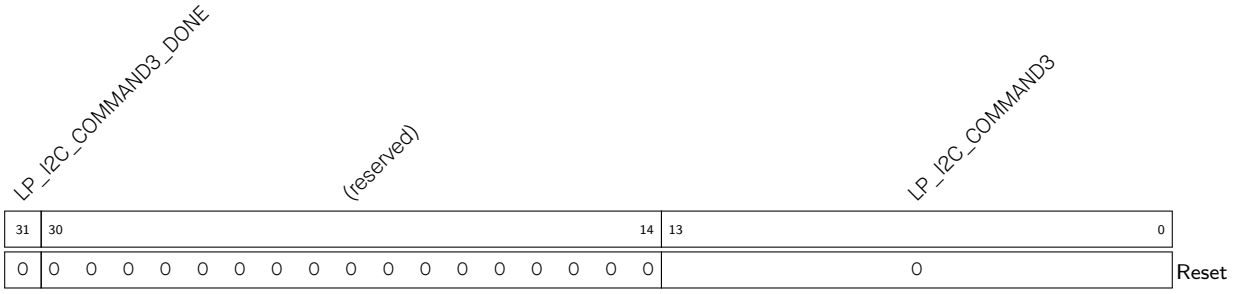
LP_I2C_COMMAND2_DONE 表示在 I2C 主机模式下，命令 2 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 47.61. LP_I2C_COMD3_REG (0x0064)



LP_I2C_COMMAND3 配置命令寄存器 3 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

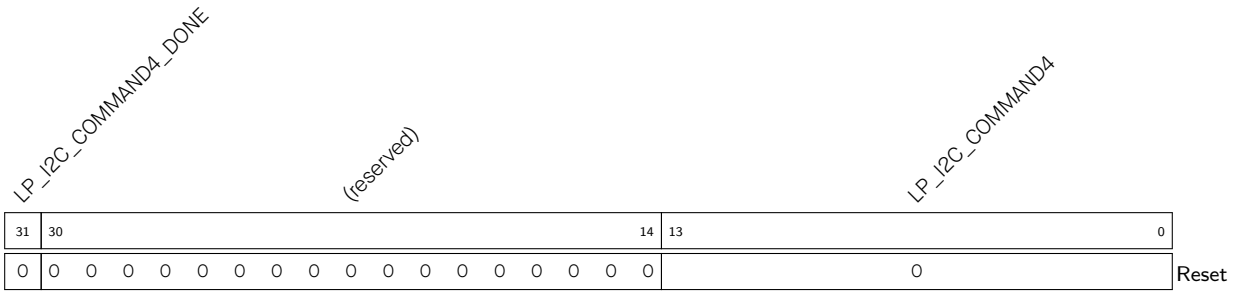
LP_I2C_COMMAND3_DONE 表示在 I2C 主机模式下，命令 3 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 47.62. LP_I2C_COMD4_REG (0x0068)



LP_I2C_COMMAND4 配置命令寄存器 4 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

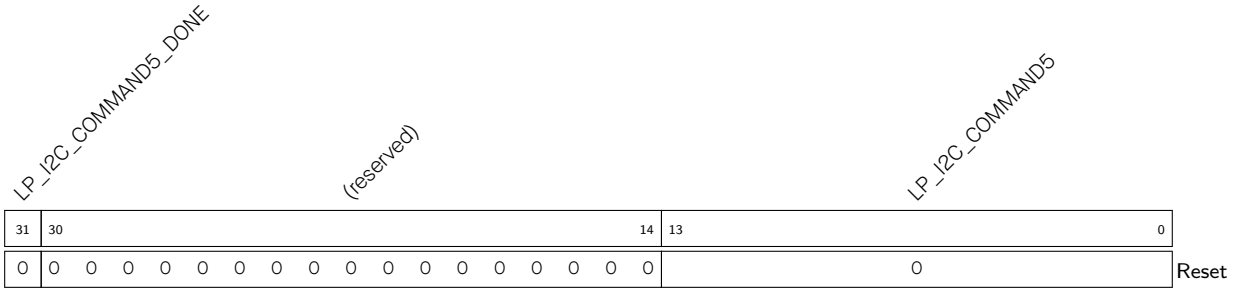
LP_I2C_COMMAND4_DONE 表示在 I2C 主机模式下，命令 4 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 47.63. LP_I2C_COMD5_REG (0x006C)



LP_I2C_COMMAND5 配置命令寄存器 5 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

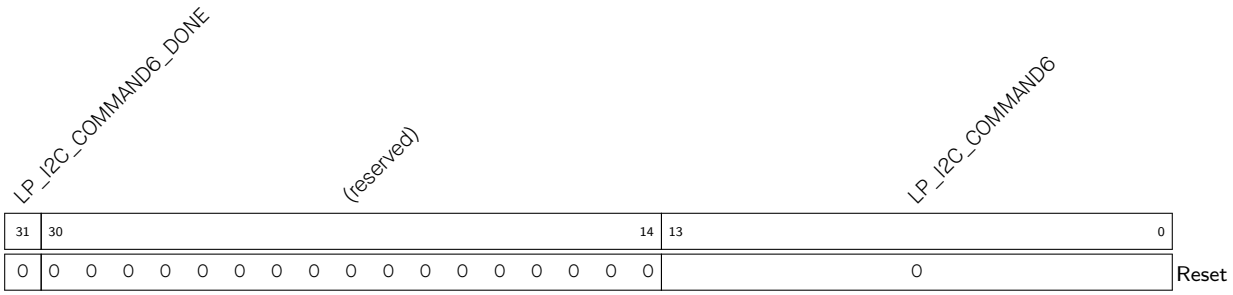
LP_I2C_COMMAND5_DONE 表示在 I2C 主机模式下，命令 5 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 47.64. LP_I2C_COMD6_REG (0x0070)



LP_I2C_COMMAND6 配置命令寄存器 6 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

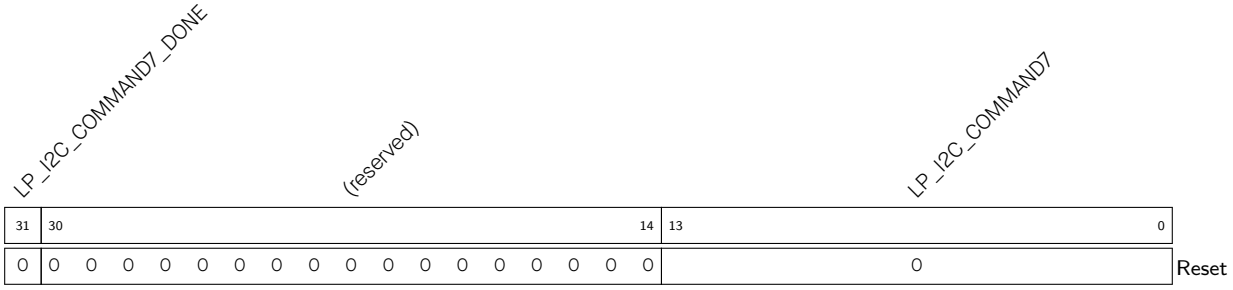
LP_I2C_COMMAND6_DONE 表示在 I2C 主机模式下，命令 6 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 47.65. LP_I2C_COMD7_REG (0x0074)



LP_I2C_COMMAND7 配置命令寄存器 7 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

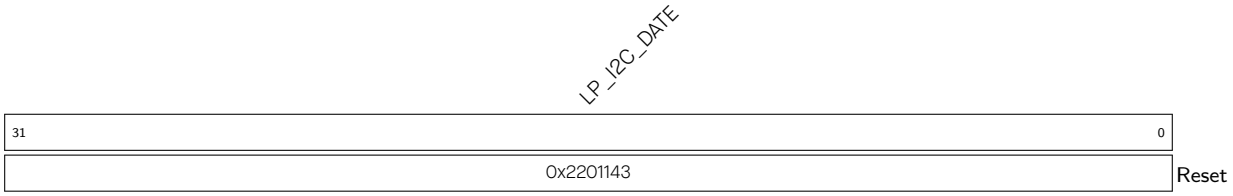
LP_I2C_COMMAND7_DONE 表示在 I2C 主机模式下，命令 7 是否已完成。

0: 未完成

1: 已完成

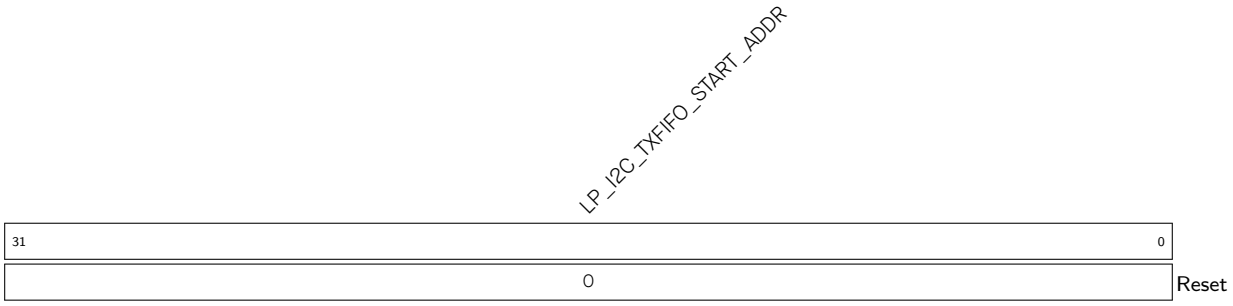
(R/W/SS)

Register 47.66. LP_I2C_DATE_REG (0x00F8)



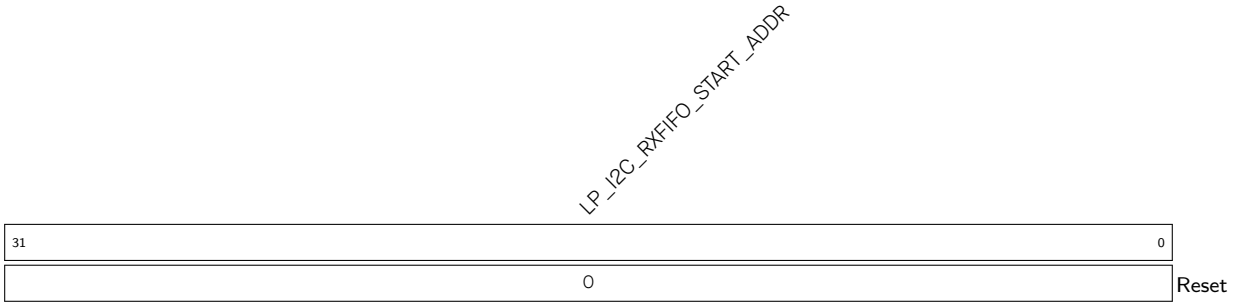
LP_I2C_DATE 版本控制寄存器。(R/W)

Register 47.67. LP_I2C_TXFIFO_START_ADDR_REG (0x0100)



LP_I2C_TXFIFO_START_ADDR 表示 I2C TX FIFO 首地址。(RO)

Register 47.68. LP_I2C_RXFIFO_START_ADDR_REG (0x0180)



LP_I2C_RXFIFO_START_ADDR 表示 I2C RX FIFO 首地址。(RO)

第 48 章

模拟 I2C 控制器

48.1 概述

模拟 I2C 控制器包含两个专用于模拟电路配置的 I2C 主机，用于与一些模拟电路模块的通信，这些模拟模块可通过配置其内部寄存器，使其工作状态更加适用于芯片当前的应用场景。以模拟传感器为例，可以通过模拟 I2C 控制器的配置，给 ADC 采样值统一加一个偏置，这样能扩展 ADC 支持的采样电压范围。每个可配置模拟模块内部都有一个 I2C 从机，并拥有独立的地址，该模拟 I2C 控制器支持如下模拟模块的配置：

- CPU_PLL：主要包含 CPLL_CLK 的配置寄存器，从机地址为 0x67，CPLL_CLK 参见章节 11 复位和时钟
- SYS_PLL：主要包含 SPLL_CLK 的配置寄存器，从机地址为 0x66，SPLL_CLK 参见章节 11 复位和时钟
- MSPI：主要包含 MPLL_CLK 的配置寄存器，从机地址为 0x63，MPLL_CLK 参见章节 11 复位和时钟
- PLLA：主要包含 APLL_CLK 的配置寄存器，从机地址为 0x6f，APLL_CLK 参见章节 11 复位和时钟
- ANA_SENSOR：主要包含模拟传感器的配置寄存器，从机地址为 0x69，主要包括 *温度传感器 (TSENS)* 和 *ADC 控制器 (ADC)*
- BIAS：主要包含电压的检测相关的配置寄存器，从机地址为 0x6a，参见章节 25 欠压检测器

48.2 主要特性

模拟 I2C 控制器支持以下特性：

- 仅支持主机模式
- 7-bit 地址寻址
- 传输速率可调
- 支持睡眠模式下工作（LP CPU 可用的睡眠模式）
- 支持双主机工作模式

48.3 结构概览

下图为模拟 I2C 控制器模块的的架构图。

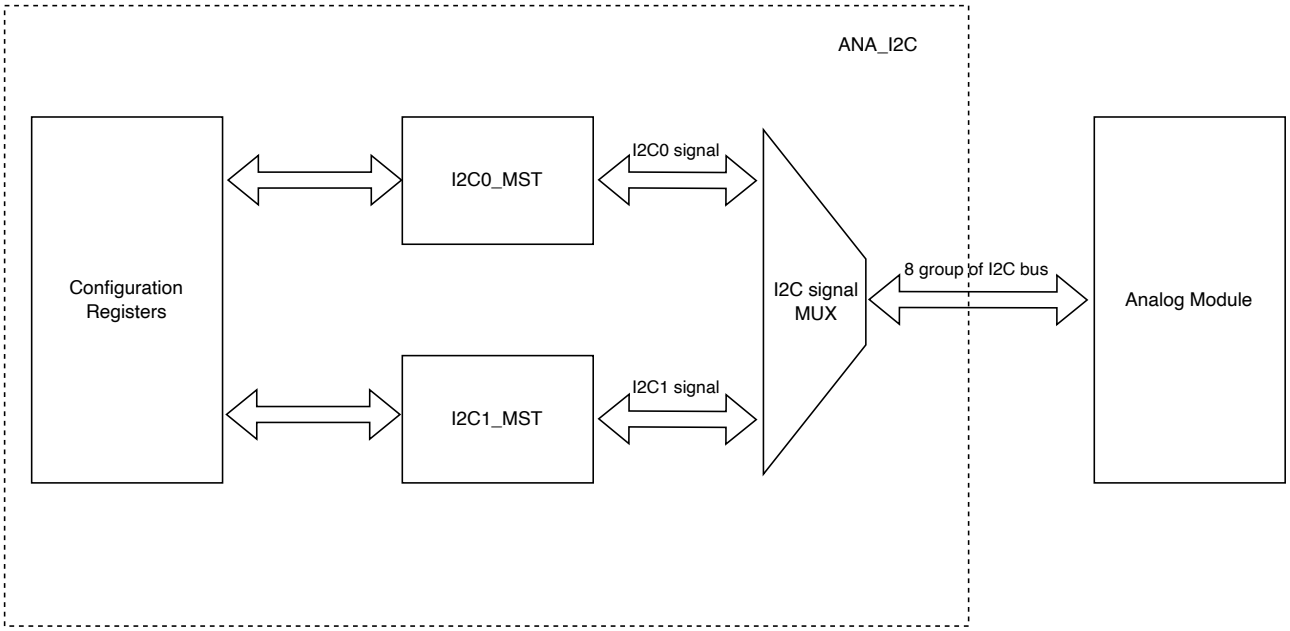


图 48.3-1. 模拟 I2C 主机模块结构图

- 配置寄存器 (Configuration Registers)：寄存器模块，与软件的接口，该模块给出 I2C 主机工作的相关配置，以控制传输的地址、数据等信息，同时通过读取 I2C0/1_MST 的状态寄存器以监控传输过程。
- I2C0/1_MST：两个 I2C 主机，支持独立并行的配置不同模拟模块，它们产生的 I2C 信号将由后级的 MUX 选择一路输出给模拟模块。
- I2C signal MUX：决定某个模拟模块由 I2C0_MST 还是 I2C1_MST 配置。

48.4 功能描述

传输速率调整

可通过配置 ANA_I2C_MST_I2C*x*_SDA_SIDE_GUARD 与 ANA_I2C_MST_I2C*x*_SCL_PULSE_DUR 决定 I2C 信号的传输速率与信号相位，寄存器值与波形关系如下图所示：

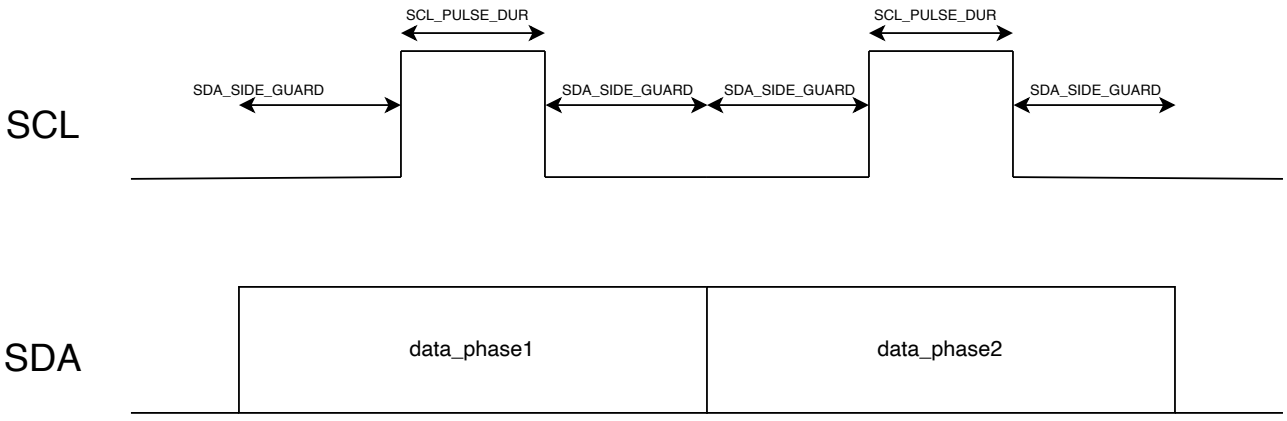


图 48.4-1. 信号相位图

睡眠模式下的使用

芯片睡眠模式下，可使用 LP CPU（章节 5 低功耗处理器）驱动模拟 I2C 控制器的工作。

双主机工作模式

该模块内置两个独立的主机模块，拥有独立的两套配置，两个主机可独立工作，配置不同的模拟模块，以此提高工作效率。

48.5 配置流程

以下为模拟 I2C 控制器配置模拟模块的流程。

- 配置 LPPERI_CK_EN_LP_I2CMST 为 1，使能模块时钟。
- 配置 ANA_I2C_MST_ANA_CONF2 选择要通信的模块。
- 配置 ANA_I2C_MST_I2C*x*_SDA_SIDE_GUARD 与 ANA_I2C_MST_I2C*x*_SCL_PULSE_DUR 决定 I2C 信号的传输速率与信号相位。
- 配置 ANA_I2C_MST_I2C*x*_CTRL 选择本次通讯的相关信息。
- 读取 ANA_I2C_MST_I2C*x*_BUSY，若读到该寄存器拉低，代表 I2C*x* 当次通信结束。

48.6 寄存器列表

本小节的所有地址均为相对于模拟 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

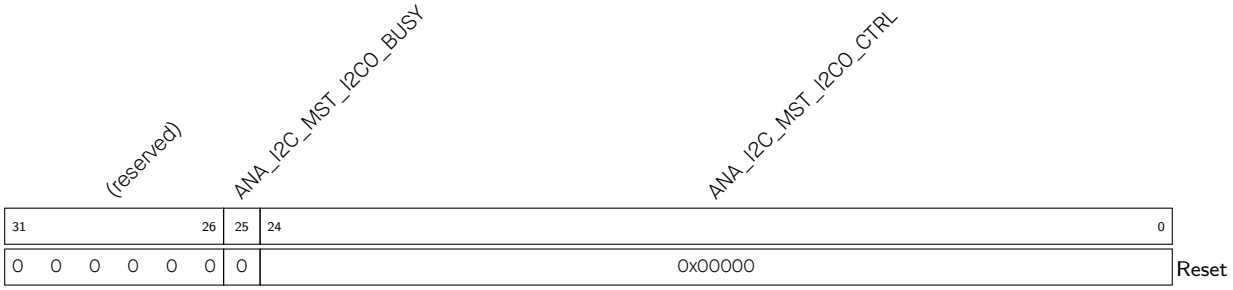
名称	描述	地址	访问
配置寄存器			
ANA_I2C_MST_I2C0_CTRL_REG	I2C0 传输配置寄存器	0x0000	varies
ANA_I2C_MST_I2C1_CTRL_REG	I2C1 传输配置寄存器	0x0004	varies
ANA_I2C_MST_ANA_CONF2_REG	I2C 主机选择寄存器	0x0020	varies
ANA_I2C_MST_I2C0_CTRL1_REG	I2C0 传输速率与信号相位配置寄存器	0x0024	R/W
ANA_I2C_MST_I2C1_CTRL1_REG	I2C1 传输速率与信号相位配置寄存器	0x0028	R/W
ANA_I2C_MST_DATE_REG	版本控制寄存器	0x0038	R/W

48.7 寄存器

本小节的所有地址均为相对于模拟 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 48.1. ANA_I2C_MST_I2C0_CTRL_REG (0x0000)



ANA_I2C_MST_I2C0_CTRL 配置 I2C0 传输信息。

bit[0:7]：配置从机地址

bit[8:15]：配置从机寄存器地址

bit[16:23]：配置传输数据

bit[24]：配置读写操作

0：写

1：读

该寄存器配置完成后 I2C0 主机将产生 I2C 读写信号。

(R/W)

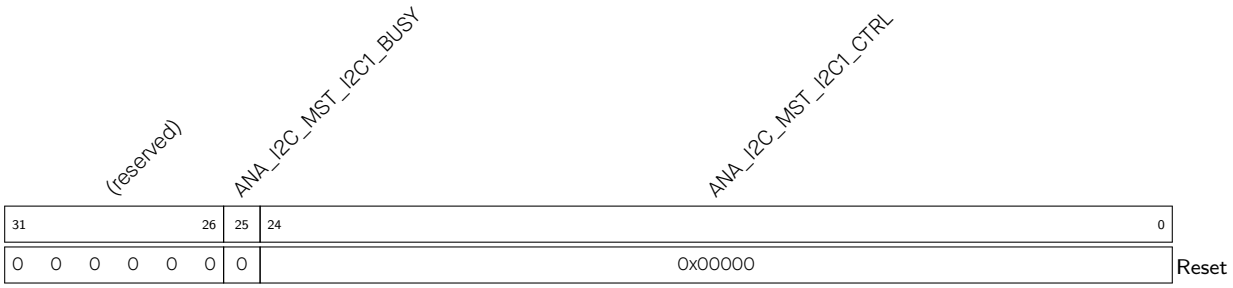
ANA_I2C_MST_I2C0_BUSY 表示 I2C0 是否正在传输数据。

0：I2C0 未进行数据传输

1：I2C0 正在传输数据

(RO)

Register 48.2. ANA_I2C_MST_I2C1_CTRL_REG (0x0004)



ANA_I2C_MST_I2C1_CTRL 配置 I2C1 传输信息。本域中各比特位含义与 ANA_I2C_MST_I2C0_CTRL 相同。(R/W)

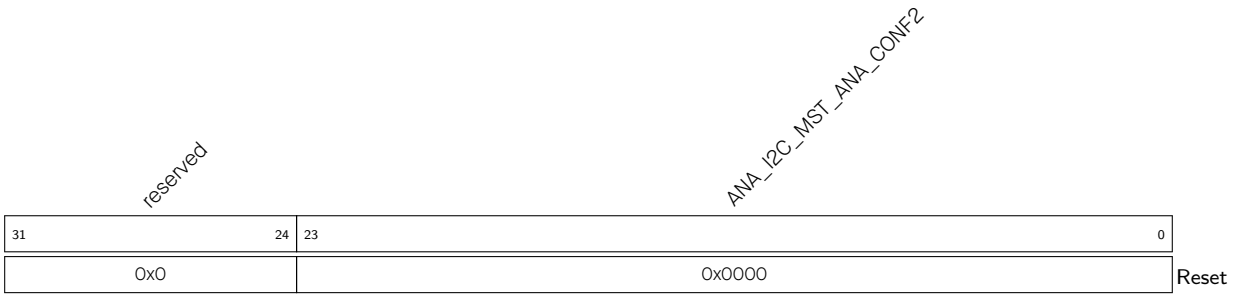
ANA_I2C_MST_I2C1_BUSY 表示 I2C1 是否正在传输数据。

0：I2C1 未进行数据传输。

1：I2C1 正在传输数据。

(RO)

Register 48.3. ANA_I2C_MST_ANA_CONF2_REG (0x0020)



ANA_I2C_MST_ANA_CONF2 配置以下模拟模块选用的 I2C 主机。若对应位配置为 1，表示选用 I2C0 主机进行通信；若配置为 0，表示选用 I2C1 主机。因为从机地址的存在，不会导致错误通信。

bit5：SYS_PLL

bit6：SDIO_PLL

bit7：ANA_SENSOR

bit8：PLLA

bit9：MSPI

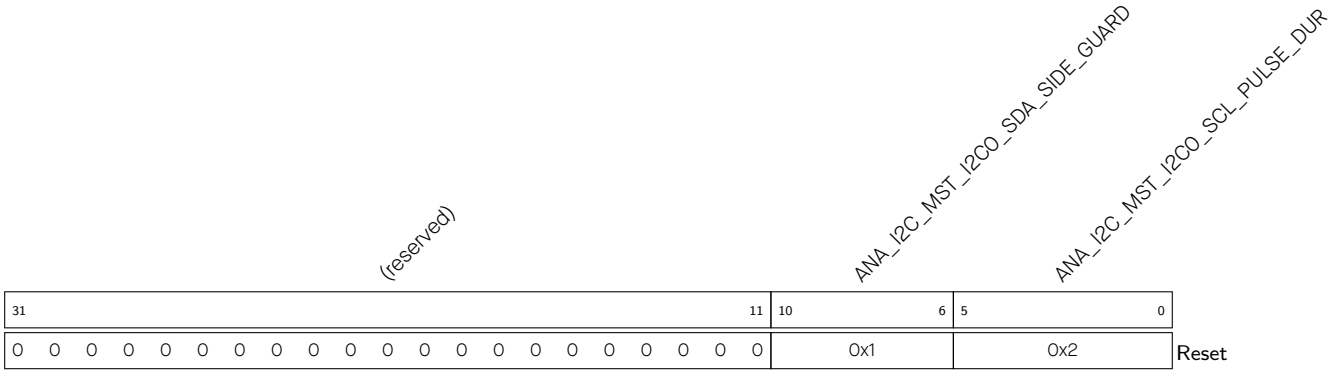
bit10：DIG_REG

bit11：CPU_PLL

bit12：BIAS

(R/W)

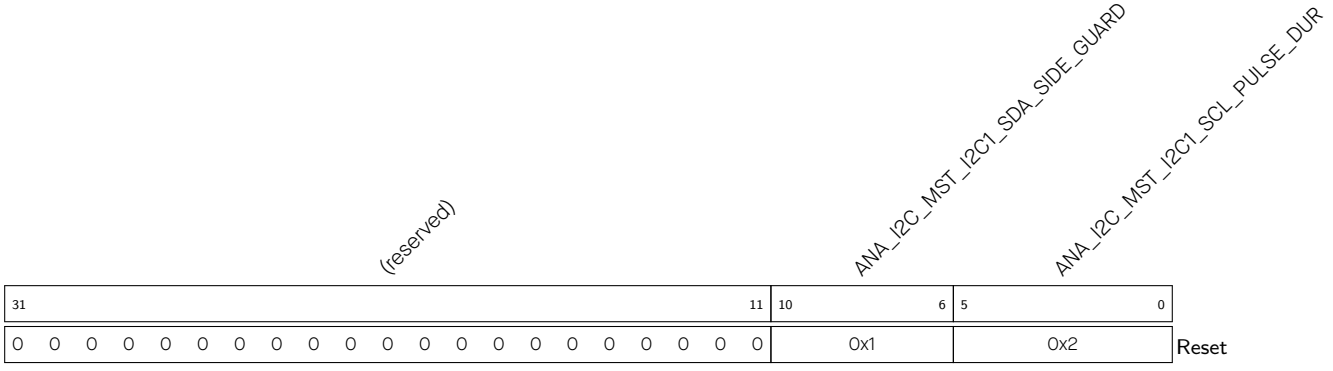
Register 48.4. ANA_I2C_MST_I2C0_CTRL1_REG (0x0024)



ANA_I2C_MST_I2C0_SCL_PULSE_DUR 配置 I2C0 驱动的 SCL 高电平持续周期，持续时间的计数器计数时钟为 LP_FAST_CLK。(R/W)

ANA_I2C_MST_I2C0_SDA_SIDE_GUARD 配置 I2C0 驱动的 SCL 低电平持续周期，持续时间的计数器计数时钟为 LP_FAST_CLK。(R/W)

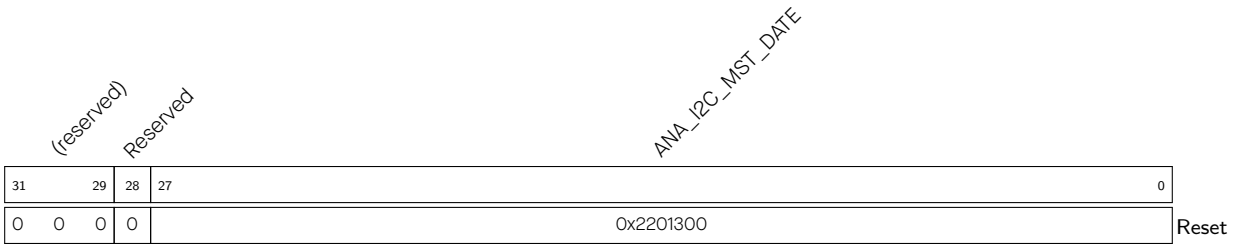
Register 48.5. ANA_I2C_MST_I2C1_CTRL1_REG (0x0028)



ANA_I2C_MST_I2C1_SCL_PULSE_DUR 配置 I2C1 驱动的 SCL 高电平持续周期，持续时间的计数器计数时钟为 LP_FAST_CLK。(R/W)

ANA_I2C_MST_I2C1_SDA_SIDE_GUARD 配置 I2C1 驱动的 SCL 低电平持续周期，持续时间的计数器计数时钟为 LP_FAST_CLK。(R/W)

Register 48.6. ANA_I2C_MST_DATE_REG (0x0038)



ANA_I2C_MST_DATE 版本控制寄存器。(R/W)

第 49 章

I2S 控制器 (I2S)

49.1 概述

ESP32-P4 内置三个 I2S 接口，即 I2S*0*、I2S*1* 和 I2S*2*，为多媒体应用，尤其是为数字音频应用提供了灵活的数据通信接口。

I2S 标准总线定义了三种信号——串行时钟信号 BCK、字选择信号 WS 和串行数据信号 SD。一个基础的 I2S 数据总线包含一个主机和一个从机。主机和从机的角色在通信过程中保持不变。ESP32-P4 的 I2S 模块包含独立的 TX 模块和 RX 模块，能够保证优良的通信性能。

说明：
本章节提供的信息适用于 I2S*0*、I2S*1* 和 I2S*2*。除非另有说明，本章节中 I2S 及 I2S*n* 均指代 I2S*0*、I2S*1* 和 I2S*2*。

49.2 术语

为了更好地说明 I2S 的功能，本章使用了以下术语。

主机模式 I2S*n* 作为主机驱动 BCK/WS，向从机发送或从其接收数据。

从机模式 I2S*n* 作为从机由 BCK/WS 驱动，从主机接收或向其发送数据。

全双工 主机与从机之间的发送线和接收线各自独立，发送数据和接收数据同时进行。

半双工 主机和从机只能有一方先发送数据，另一方接收数据。发送数据和接收数据不能同时进行。

A 律和 μ 律 A 律和 μ 律是数字脉冲编码调制 (PCM) 非均匀量化中的压缩/解压缩算法，可以有效地改善信号的量化信噪比。

TDM RX 模式 即 TDM 输入模式，该模式利用时分复用方式接收脉冲编码调制 (PCM) 数据。信号线包括 BCK、WS 和 DATA。可以接收最多 16 个通道的数据。用户可配置数据格式，支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。

PDM RX 模式 即 PDM 输入模式，该模式接收脉冲密度调制 (PDM) 数据。信号线包括 WS 和 DATA。用户可配置数据格式，支持 PDM 标准格式等。

TDM TX 模式 即 TDM 输出模式，该模式利用时分复用方式发送脉冲编码调制 (PCM) 数据。信号线包括 BCK、WS 和 DATA，可以发送最多 16 个通道的数据。用户可配置数据格式，支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。

PDM TX 模式 即 PDM 输出模式，该模式发送脉冲密度调制 (PDM) 数据。信号线包括 WS 和 DATA。用户可配置数据格式，支持 PDM 标准格式等。

PCM-to-PDM 数据格式转换器（仅适用于 I2S*0*） 将 PCM 数据转换为 PDM 数据的格式转换器（后文简称为 PCM-to-PDM 转换器），可以在 PDM TX 主机模式下启用。启用后，I2S 通过 GDMA 从存储器中取得脉冲编码调制 (PCM) 数据，将其转换为脉冲密度调制 (PDM) 数据并发送。

PDM-to-PCM 数据格式转换器（仅适用于 I2S*0*） 将 PDM 数据转换为 PCM 数据的格式转换器（后文简称为 PDM-to-PCM 转换器），可以在 PDM RX 主机或从机模式下启用。启用后，I2S 接收脉冲密度调制 (PDM) 数据，将其转换为脉冲编码调制 (PCM) 数据，并将其通过 GDMA 存入存储器。

49.3 特性

I2S 模块具有以下特性：

- 支持主机模式和从机模式
- 支持全双工和半双工通信
- 支持 TX 模块和 RX 模块独立工作或同时工作
- 支持多种音频标准：
  - TDM Philips 标准
  - TDM MSB 对齐标准
  - TDM PCM 标准
  - PDM 标准
- 支持多种 TX/RX 模式
  - TDM TX 模式，最多支持 16 通道
  - TDM RX 模式，最多支持 16 通道
  - PDM TX 模式
    - 支持原始 PDM 数据发送
    - 支持将 PCM 数据转换为 PDM 数据发送（仅适用于 I2S*0*），最多支持 2 通道
  - PDM RX 模式
    - 支持原始 PDM 数据接收
    - 支持将 PDM 数据转换为 PCM 数据接收（仅适用于 I2S*0*），最多支持 8 通道
- 可配置 APLL 时钟源，支持最高频率为 125 MHz
- 可配置高精度采样时钟，支持多种采样频率（详见下文注释）
- 支持 8/16/24/32 位的数据位宽
- TX 模式支持同步计数器
- 支持 ETM 功能
- 支持 GDMA（仅 GDMA-AHB）

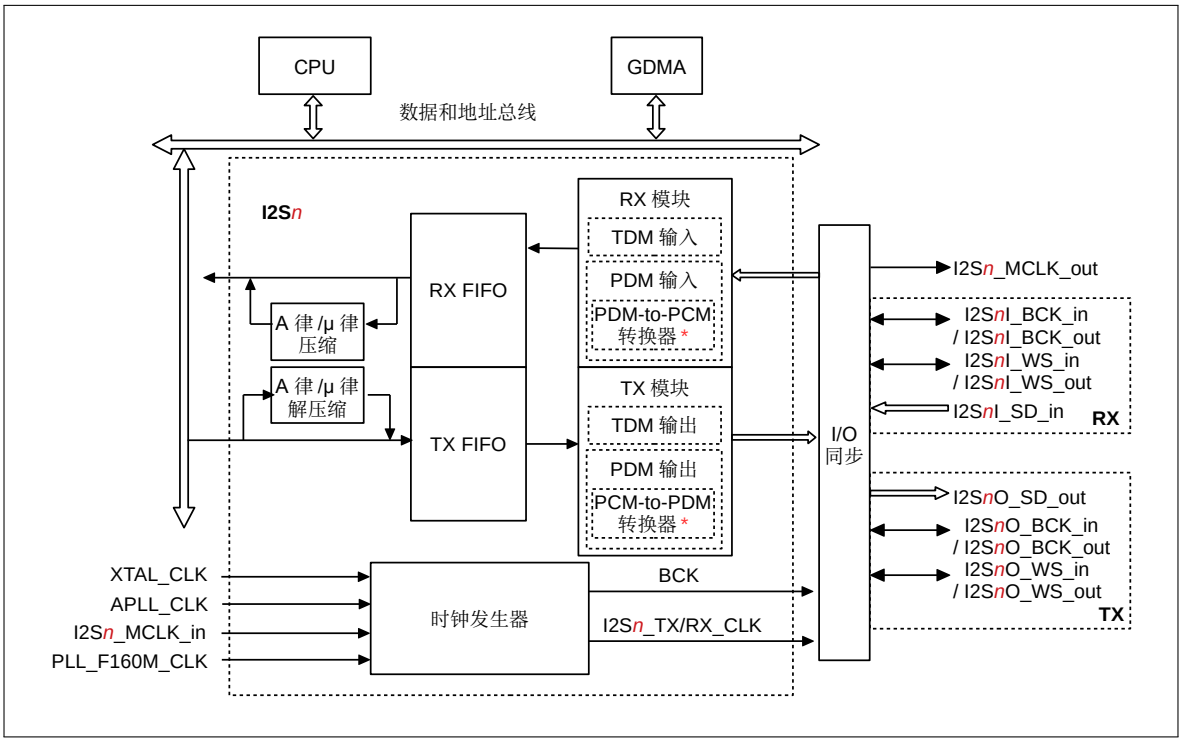
- 支持 I2S 接口中断

说明：
从机模式下，为了保障采样的准确性，模块时钟频率必须大于等于 BCK 时钟频率的 8 倍，因此 I2S*n* 支持的采样频率受到数据位宽和声道数的限制。例如，时钟源频率最高为 125 MHz，即模块时钟最高能配置为 62.5 MHz，能支持的 BCK 时钟频率最高为 7.8125 MHz，因此传输双声道 32 位宽数据时，能够支持 122.070 kHz 以下的采样频率，例如 8 kHz、16 kHz、32 kHz、44.1 kHz、48 kHz、88.2 kHz、96 kHz。具体请参考章节 49.6 *I2S TX/RX 模块时钟*。

49.4 系统架构

图 49.4-1 为 ESP32-P4 I2S*n* 模块的结构框图。ESP32-P4 I2S*n* 模块包含：

- 独立的 TX 模块
- 独立的 RX 模块
- 输入输出时序调节单元（I/O 同步）
- 时钟分频器（时钟发生器）
- 64 x 32 位 TX FIFO
- 64 x 32 位 RX FIFO
- 压缩/解压缩模块



注：图中 PDM-to-PCM 转换器和 PCM-to-PDM 转换器仅适用于 I2S*0*。

图 49.4-1. ESP32-P4 I2S 系统框图

ESP32-P4 I2S*n* 模块支持 GDMA，可访问内部存储器，更多信息见章节 6 *通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*。

TX 模块和 RX 模块各自有一组三线接口，分别为串行时钟线 BCK、字选择线 WS 和串行数据线 SD。其中，TX 模块的 SD 线固定为输出，RX 模块的 SD 线固定为接收。主机模式下，TX 模块和 RX 模块的 BCK 和 WS 信号线为输出，从机模式下，TX 模块和 RX 模块的 BCK 和 WS 信号线为输入。

图 49.4-1 中的右半部分展示了 I2S*n* 模块的信号总线。RX 和 TX 模块的信号命名规则为：I2S*n*A_B_C，例如 I2S*n*I_BCK_in。其中：

- “A” 表示信号属于 I2S*n* TX 或 RX 模块，包括：
  - “I” 表示信号输入至/输出自 RX 模块
  - “O” 表示信号输入至/输出自 TX 模块
- “B” 表示信号功能，包括：
  - BCK
  - WS
  - SD
- “C” 表示该信号的方向，包括：
  - “in” 表示该信号输入 I2S*n* 模块
  - “out” 表示该信号自 I2S*n* 模块输出

I2S 各信号的具体描述见表 49.4-1。

表 49.4-1. 模块信号描述

信号	方向	功能
I2S*n*I_BCK_in	输入	I2S*n* 从机模式下，输入 BCK 信号，用于 RX 模块
I2S*n*I_BCK_out	输出	I2S*n* 主机模式下，输出 BCK 信号，用于 RX 模块
I2S*n*I_WS_in	输入	I2S*n* 从机模式下，输入 WS 信号，用于 RX 模块
I2S*n*I_WS_out	输出	I2S*n* 主机模式下，输出 WS 信号，用于 RX 模块
I2S*n*I_Data_in	输入	I2S*n* RX 模块的串行输入数据线
I2S*n*O_Data_out	输出	I2S*n* TX 模块的串行输出数据线
I2S*n*O_BCK_in	输入	I2S*n* 从机模式下，输入 BCK 信号，用于 TX 模块
I2S*n*O_BCK_out	输出	I2S*n* 主机模式下，输出 BCK 信号，用于 TX 模块
I2S*n*O_WS_in	输入	I2S*n* 从机模式下，输入 WS 信号，用于 TX 模块
I2S*n*O_WS_out	输出	I2S*n* 主机模式下，输出 WS 信号，用于 TX 模块
I2S*n*_MCLK_in	输入	I2S*n* 从机模式下，来自外部芯片的时钟源
I2S*n*_MCLK_out	输出	I2S*n* 主机模式下，作为外部芯片的时钟源
I2S*0*I_Data1_in	输入	RX 模块在启用 PDM-to-PCM 转换器时的串行输入数据线
I2S*0*I_Data2_in	输入	RX 模块在启用 PDM-to-PCM 转换器时的串行输入数据线
I2S*0*I_Data3_in	输入	RX 模块在启用 PDM-to-PCM 转换器时的串行输入数据线
I2S*0*O_Data1_out	输出	TX 模块在启用 PCM-to-PDM 转换器时的串行输出数据线

* I2S*n* 的所有信号均需要经过 HP GPIO 交换矩阵映射到芯片的管脚。更多信息请参考章节 10 *GPIO 交换矩阵和 IO MUX*。

49.5 I2S 模块支持的音频协议

ESP32-P4 I2S*n* 模块支持多种音频标准，包括 TDM Philips 标准、TDM MSB 对齐标准、TDM PCM 标准以及 PDM 标准。

用户可通过配置以下寄存器，选择所需的音频标准：

- I2S_TX/RX_TDM_EN
  - 0：禁用 TDM 模式
  - 1：选择 TDM 模式
- I2S_TX/RX_PDM_EN
  - 0：禁用 PDM 模式
  - 1：选择 PDM 模式
- I2S_TX/RX_MSB_SHIFT
  - 0：配置 WS 信号和 SD 信号同时开始变化，即选择 MSB 对齐标准
  - 1：配置 WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即选择 Philips 标准或 PCM 标准

说明：
不可将 I2S_TX/RX_TDM_EN 和 I2S_TX/RX_PDM_EN 同时配置为 1 或同时配置为 0，否则会导致 ESP32-P4 I2S*n* 以既非 TDM 模式也非 PDM 模式的状态错误发送数据。

49.5.1 TDM Philips 标准模式

在 Philips 标准下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并在当前通道数据发送结束前一个 BCK 时钟周期开始变化。SD 信号线上首先传输音频数据的最高位。

与 Philips 标准相比，TDM Philips 标准支持的通道更多，见图 49.5-1。

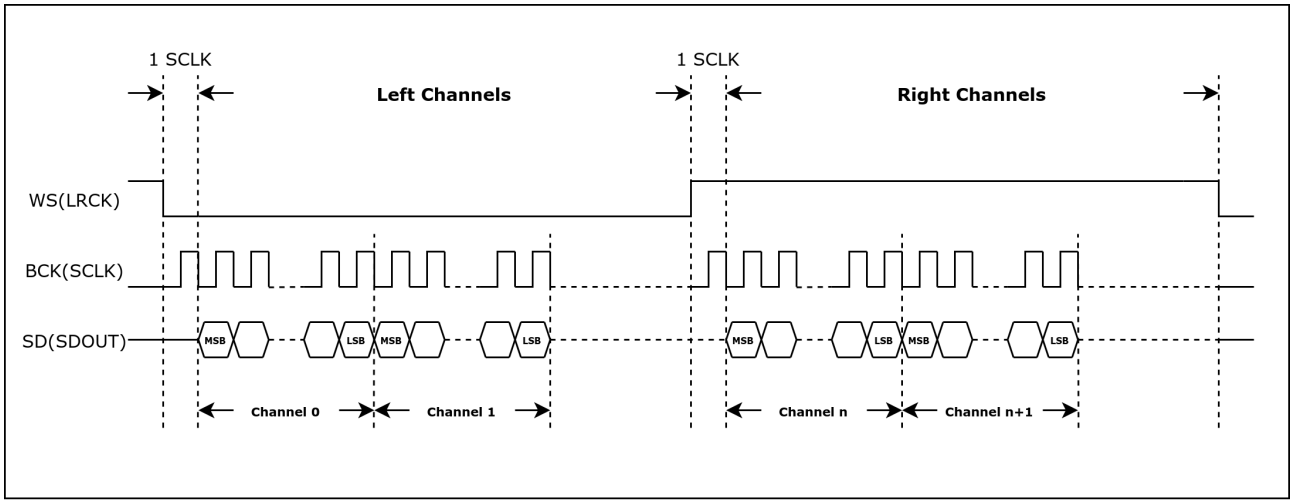


图 49.5-1. 时序图 – TDM Philips 标准

49.5.2 TDM MSB 对齐标准模式

MSB 对齐标准下，在 BCK 下降沿，WS 信号和 SD 信号同时变化。WS 持续到当前通道数据发送结束，SD 信号线上首先传输音频数据的最高位。

与 MSB 对齐标准相比，TDM MSB 对齐标准支持更多通道，见图 49.5-2。

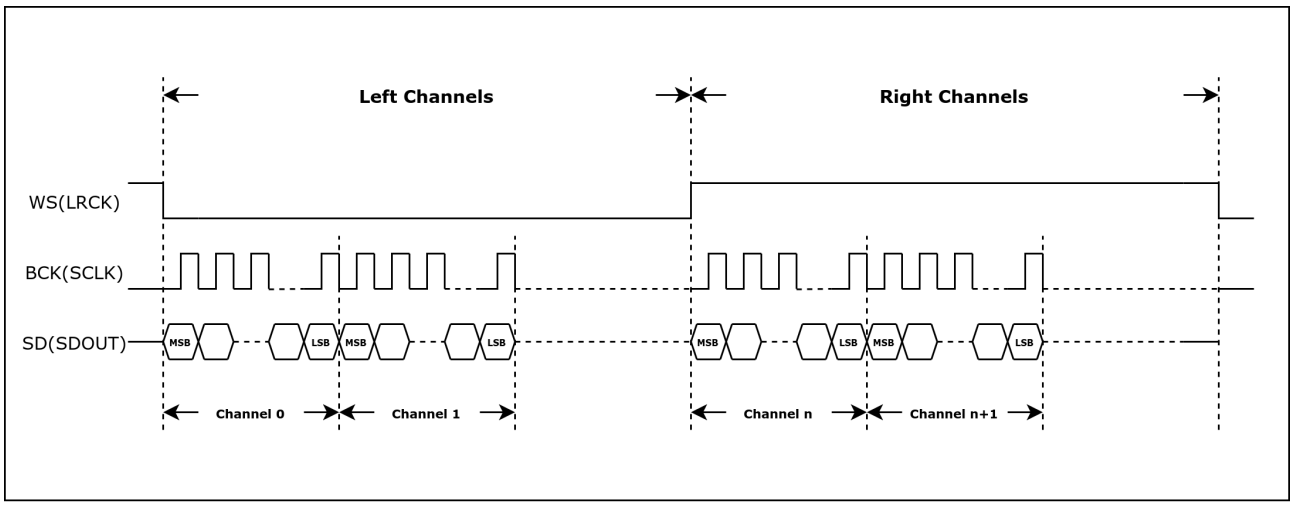


图 49.5-2. 时序图 – TDM MSB 对齐标准

49.5.3 TDM PCM 标准模式

在 PCM 标准的短帧同步模式下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并持续一个 BCK 时钟周期。SD 信号线上首先传输音频数据的最高位。

与 PCM 标准相比，TDM PCM 标准支持更多通道，见图 49.5-3 所示。

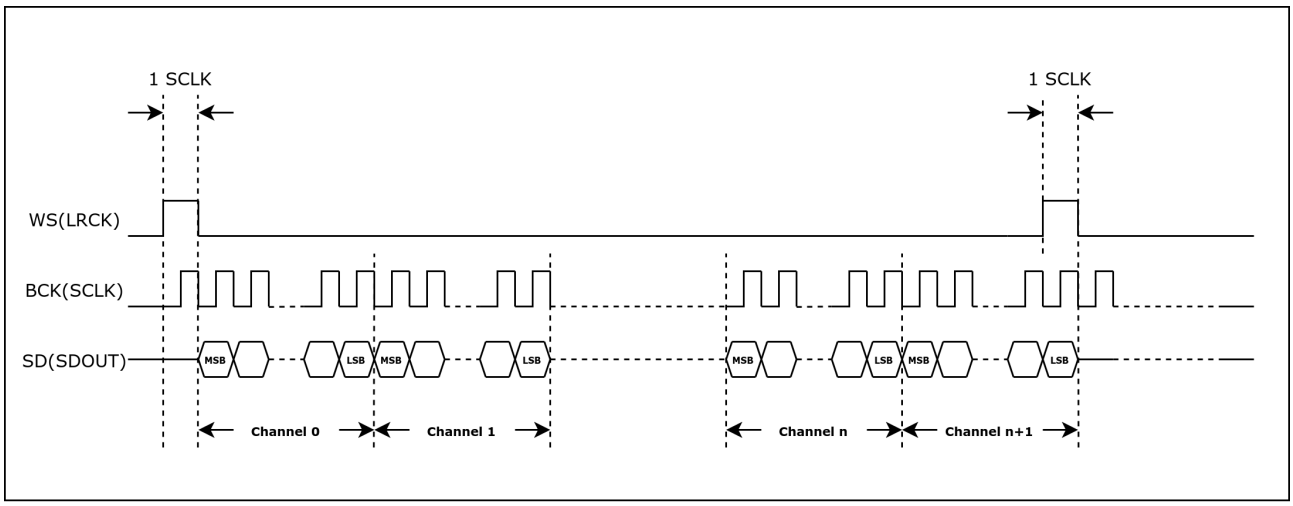


图 49.5-3. 时序图 – TDM PCM 标准

49.5.4 PDM 标准模式

如图 49.5-4 所示，在 PDM 标准下，WS 代表左/右声道，WS 与 SD 同时变化。WS 在数据发送过程中持续变化，WS 的高低对应两个声道。请参考图 49.5-4。

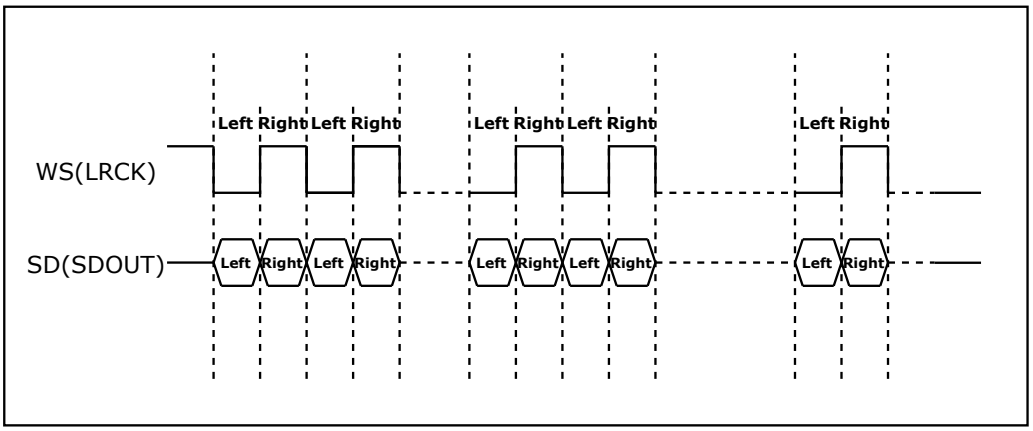


图 49.5-4. 时序图 – PDM 标准

49.6 I2S TX/RX 模块时钟

I2S*n*_TX/RX_CLK 作为 I2S*n* TX/RX 模块的主时钟，可由以下时钟分频所得：

- 40 MHz XTAL_CLK
- 160 MHz PLL_F160M_CLK
- 可调频率的 APLL_CLK
- 外部输入时钟 I2S*n*_MCLK_in

I2S*n* TX/RX 模块的串行时钟 BCK 再由 I2S*n*_TX/RX_CLK 分频获得，如图 49.6-1 所示。HP_SYS_CLKRST_I2S*n*_TX/RX_CLK_SRC_SEL 用于选择 I2S TX/RX 的时钟源，HP_SYS_CLKRST_I2S*n*_TX/RX_CLK_EN 用于使能或者关闭 I2S TX/RX 模块的时钟源。

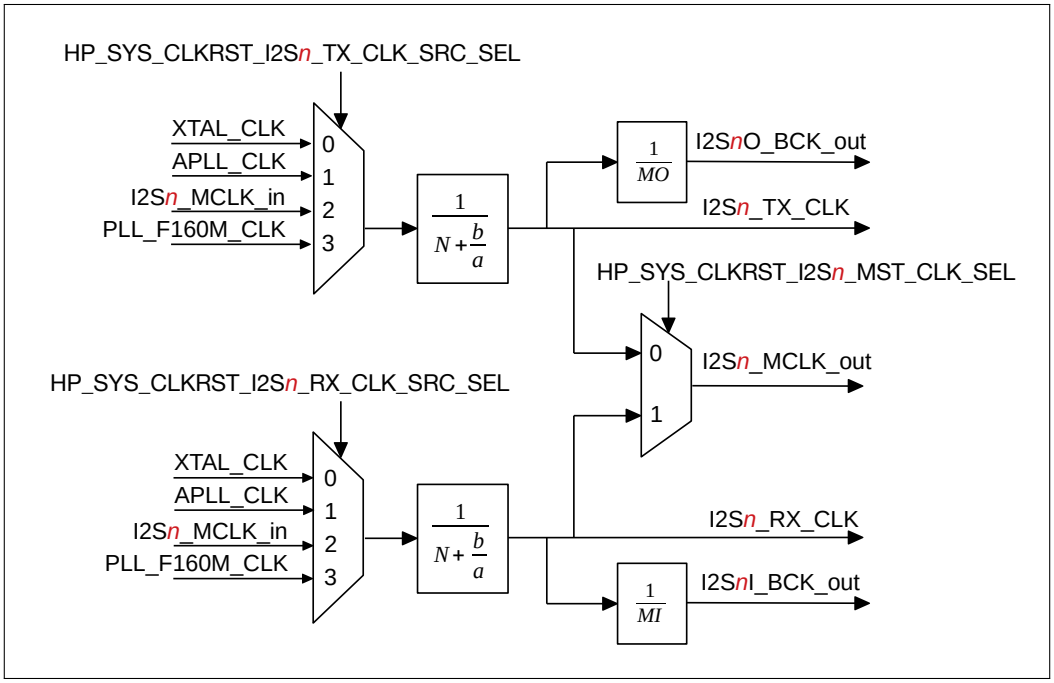


图 49.6-1. I2S*n* 时钟分频器

I2S*n*_TX/RX_CLK 的频率 f_{I2Sn_TX/RX_CLK} 与分频器时钟源频率 $f_{I2Sn_CLK_S}$ 间的关系如下：

$$f_{I2Sn_TX/RX_CLK} = \frac{f_{I2Sn_CLK_S}}{N + \frac{b}{a}}$$

其中，2 <= N <= 256（N 为整数），N 对应寄存器域 HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N 的值，具体为：

- HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N = 0 时，N = 256；
- HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N = 1 时，N = 2；
- HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N 为其他值时，N = HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_N 的值。

分数部分分频系数 a、b 唯一对应系数 x、y、z、yn1。系数对应公式为：

- 当 b <= $\frac{a}{2}$ 时，yn1 = 0，x = $floor([\frac{a}{b}]) - 1$，y = a%b，z = b；
- 当 b > $\frac{a}{2}$ 时，yn1 = 1，x = $floor([\frac{a}{a-b}]) - 1$，y = a%(a - b)，z = a - b。

系数 x、y、z、yn1 在 HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_X、HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Y、HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Z、HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_YN1 中配置。

对于整数分频，HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_X 和 HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Z 清零，HP_SYS_CLKRST_I2S*n*_TX/RX_DIV_Y 置为 1。

说明：
使用小数分频功能可能会产生时钟抖动。

在主机发送模式下，I2S*n* TX 模块的串行时钟 BCK 为 I2S*n*O_BCK_out 信号，由 I2S*n*_TX_CLK 分频获得。即：

$$f_{I2Sn_BCK_out} = \frac{f_{I2Sn_TX_CLK}}{MO}$$

其中，MO 的值为：

$$MO = I2Sn_TX_BCK_DIV_NUM + 1$$

说明：
注意，I2S_TX_BCK_DIV_NUM 不可配置为 1。

在主机接收模式下，I2S*n* RX 模块的串行时钟 BCK 为 I2S*n*I_BCK_out 信号，由 I2S*n*_RX_CLK 分频获得。即：

$$f_{I2Sn_BCK_out} = \frac{f_{I2Sn_RX_CLK}}{MI}$$

其中 MI 的值为：

$$MI = I2S_RX_BCK_DIV_NUM + 1$$

说明：

- I2S_RX_BCK_DIV_NUM 不可配置为 1；
- 当模块处于 I2S 从机模式时，必须保证 f_{I2S_TX/RX_CLK} >= 8 × f_{BCK}。另外模块可以输出 I2S*n*_MCLK_out 作为外部

设备的主时钟。

49.7 I2S 模块复位

I2S*n* 模块中各个单元以及 FIFO 可通过配置相关位进行复位：

- I2S*n* TX/RX 单元：可配置 I2S_TX_RESET 和 I2S_RX_RESET 进行复位；
- I2S*n* TX/RX FIFO：可配置 I2S_TX_FIFO_RESET 和 I2S_RX_FIFO_RESET 进行复位。

说明：
在模块和 FIFO 复位之前，需要先配置 I2S*n* 模块时钟。

49.8 I2S 主/从机模式

ESP32-P4 I2S*n* 模块可作为主机或从机，两种模式均支持半双工通信或全双工通信。用户可配置 I2S_RX_SLAVE_MOD 和 I2S_TX_SLAVE_MOD 选择需要的模式。

- I2S_TX_SLAVE_MOD
  - 0：主机发送模式
  - 1：从机发送模式
- I2S_RX_SLAVE_MOD
  - 0：主机接收模式
  - 1：从机接收模式

49.8.1 主/从机发送模式

- 主机发送模式
  - 置位 I2S_TX_START 启动一次发送操作。置位该位，TX 模块会一直输出时钟信号和串行数据。
  - 置位 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，则主机停止发送数据和时钟信号。
  - 清零 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，发送模块将一直发送最后一帧数据和时钟信号。
  - 当 I2S_TX_START 被清零时，主机停止发送数据。
- 从机发送模式
  - 置位 I2S_TX_START。TX 模块等待主机 BCK 时钟，来启动发送操作。
  - 置位 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，则从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。
  - 清零 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，TX 模块将一直发送最后一帧数据。
  - 当 I2S_TX_START 被清零时，从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。

49.8.2 主/从机接收模式

- 主机接收模式
  - 置位 I2S_RX_START 启动一次接收操作。RX 模块会一直输出时钟信号，并对输入数据进行采样。
  - 配置 I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    * 0：只有 I2S_RX_START 被清除，RX 模块才会暂停接收。
    * 1：RX 模块会在 I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收。当 I2S_RX_START 未被清除时，可通过置位 I2S_RX_RESET 来重新启动接收。
    * 2：RX 模块会在 I2S_RX_START 被清除或 GDMA RX FIFO 满时暂停接收。当 I2S_RX_START 未被清除且 GDMA RX FIFO 不再满时，可通过置位 I2S_RX_RESET 重新启动接收。
  - 清零 I2S_RX_START，RX 模块停止接收数据。
- 从机接收模式
  - 置位 I2S_RX_START。等待主机 BCK 时钟，来启动接收操作。
  - 配置 I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    * 0：只有 I2S_RX_START 被清除，RX 模块才会暂停接收；
    * 1：RX 模块会在 I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收。当 I2S_RX_START 未被清除时，可通过置位 I2S_RX_RESET 来重新开始接收；
    * 2：RX 模块会在 I2S_RX_START 被清除或 GDMA RX FIFO 满时暂停接收。当 I2S_RX_START 未被清除且 GDMA RX FIFO 不再满时，可通过置位 I2S_RX_RESET 恢复接收。
  - 清零 I2S_RX_START，RX 模块停止接收数据。

49.9 发送数据

> **说明：**
> 本小节以及后续小节所述的配置，均需要通过置位 I2S_TX_UPDATE 的方式来进行更新，从而将 I2S*n* TX 寄存器数据从 APB 时钟域同步到 I2S*n* TX 时钟域。详细配置见第 49.13.1 小节。

ESP32-P4 I2S*n* 发送数据时，从 GDMA 读取数据，经过数据格式控制和通道模式控制，从外设输出信号输出对应数据。

49.9.1 数据格式控制

数据格式控制分为三个阶段：

- 第一阶段：从内存中读出有效数据并写入 TX FIFO；
- 第二阶段：将待发送数据从 TX FIFO 中读出，并进行输出数据模式转换；
- 第三阶段：将待发送数据转换为串行数据流输出。

49.9.1.1 通道有效数据位宽

I2S_TX_BITS_MOD 和 I2S_TX_24_FILL_EN 决定了每个通道的有效数据位宽，其可取值和对应的有效数据位宽如下表：

表 49.9-1. 通道有效数据位宽控制

通道有效数据位宽	I2S_TX_BITS_MOD	I2S_TX_24_FILL_EN
32	31	x*
	23	1
24	23	0
16	15	x
8	7	x

* "x" 表示该值被忽略。

49.9.1.2 通道有效数据字节序

I2S*n* 通过 GDMA 读取数据之后，I2S_TX_BIG_ENDIAN 用于控制从 GDMA 读取数据的字节序。表 49.9-2 描述了不同通道有效数据位宽下，该寄存器对读取数据的控制。

表 49.9-2. 通道有效数据字节序控制

通道有效数据位宽	原始数据	控制后数据字节序	I2S_TX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

说明：

B0、B1、B2、B3 各代表一个 8 位数据，符号 {} 表示这些字节被组合到一起。例如，{B3, B2, B1, B0} 代表一个 32 位的数据，其中 B0 代表 0-7 位，B1 代表 8-15 位，B2 代表 16-23 位，B3 代表 24-31 位。

49.9.1.3 A 律/μ 律压缩/解压缩

ESP32-P4 I2S 按照 32 位（缺省高位补 0*）的方式对排列好字节序的有效数据进行 A 律/μ 律压缩/解压缩。

说明：

缺省高位补 0，即如果有效数据位宽小于 32，则待压缩/解压缩数据的 [31: 有效位宽] 位自动填充 0。

配置 I2S_TX_PCM_BYPASS 为：

- 0：进行压缩/解压缩
- 1：不进行压缩/解压缩

配置 I2S_TX_PCM_CONF 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2：μ 律解压缩
- 3：μ 律压缩

至此，数据格式控制的第一阶段完成。

49.9.1.4 通道发送数据位宽

ESP32-P4 I2S*n* 中，I2S_TX_TDM_CHAN_BITS 决定了每个通道发送数据的位宽。

- 当每个通道发送数据的位宽大于有效数据位宽时，会在剩余的位置补充 0。此时，配置 I2S_TX_LEFT_ALIGN 为：
  - 0：有效数据位于发送数据低位，在数据高位补零；
  - 1：有效数据位于发送数据高位，在数据低位补零。
- 当每个通道发送数据的位宽小于有效数据位宽时，每次发送数据仅发送低位有效数据部分，高位部分将被舍弃。

至此，数据格式控制的第二阶段完成。

49.9.1.5 通道数据比特顺序

ESP32-P4 I2S*n* 通道数据比特顺序由 I2S_TX_BIT_ORDER 控制：

- 0：不调换有效数据比特顺序；
- 1：调换有效数据比特顺序。

至此，数据格式控制部分全部完成。经过数据格式控制的数据将从高位向低位依次发送。图 49.9-1 所示即为一次完整的 TX 数据格式控制过程。

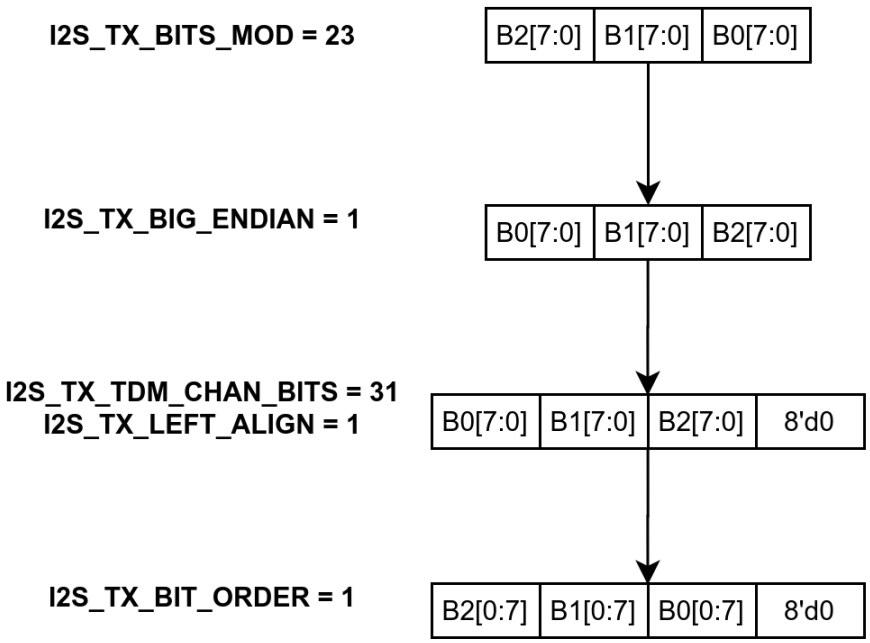


图 49.9-1. TX 数据格式控制

49.9.2 通道模式控制

ESP32-P4 I2S*n* 支持 TDM 和 PDM 两种发送模式。置位 I2S_TX_TDM_EN 使能 TDM 输出模式，置位 I2S_TX_PDM_EN 使能 PDM 输出模式。

> **说明：**
> I2S_TX_TDM_EN 和 I2S_TX_PDM_EN 不能同时置位或同时清零。

49.9.2.1 TDM 输出模式

在 TDM 输出模式下，I2S*n* 最多支持 16 个发送通道。发送通道数由 I2S_TX_TDM_TOT_CHAN_NUM 控制。例如，配置 I2S_TX_TDM_TOT_CHAN_NUM 为 5，则六个通道（通道 0 ~ 5）将用于发送数据，见图 49.9-2。

> **说明：**
> 将 I2S*n* 模块设置为 TDM 双通道模式，可实现控制大多数 I2S*n* 双声道编解码器。

在发送数据的通道中，如果其对应的 I2S_TX_TDM_CHANn_EN 为：

- 1：则该通道发送通道数据；
- 0：则该通道发送数据由 I2S_TX_CHAN_EQUAL 控制，当该值为：
  - 1：则发送前一个通道的数据；
  - 0：则发送 I2S_SINGLE_DATA 的值。

当 I2S*n* 处于主机 TDM 输出模式下，WS 信号由 I2S_TX_WS_IDLE_POL 和 I2S_TX_TDM_WS_WIDTH 控制：

- I2S_TX_WS_IDLE_POL 的值为 WS 信号的默认电平；
- I2S_TX_TDM_WS_WIDTH 的值为在发送所有通道数据的过程中，WS 为默认电平的周期数。

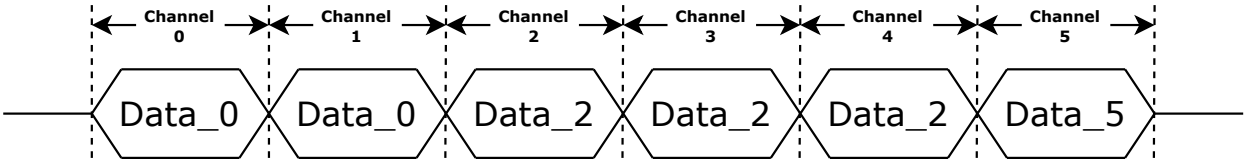
另外，I2S_TX_HALF_SAMPLE_BITS 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

TDM 通道配置示例

在本示例中，寄存器的配置如下：

- 配置 I2S_TX_TDM_CHAN_NUM 为 5，即选择使用通道 0 ~ 5 进行数据发送。
- 配置 I2S_TX_CHAN_EQUAL 为 1，即如果相应通道的 I2S_TX_TDM_CHAN*n*_EN (*n* = 0 ~ 5) 被清除，则该通道将发送前一通道的数据。
- 配置 I2S_TX_TDM_CHAN0/2/5_EN 为 1，即这些通道将用于发送通道数据。
- 配置 I2S_TX_TDM_CHAN1/3/4_EN 为 0，则这些通道将发送前一个通道的数据。

配置完成后，则数据将按下图方式进行发送。



I2S_TX_TDM_CHAN_NUM = 5; I2S_TX_CHAN_EQUAL = 1;
I2S_TX_TDM_CHAN0_EN = 1; I2S_TX_TDM_CHAN1_EN = 0; I2S_TX_TDM_CHAN2_EN = 1;
I2S_TX_TDM_CHAN3_EN = 0; I2S_TX_TDM_CHAN4_EN = 0; I2S_TX_TDM_CHAN5_EN = 1;

图 49.9-2. TDM 通道控制

49.9.2.2 PDM 输出模式

在 PDM 输出模式下，I2S*n* 可以发送原始 PDM 数据，或者将 PCM 数据转换为 PDM 数据后发送（后者仅 I2S*0* 支持）。

在 PDM 输出模式下，从 GDMA 取得数据的过程由 I2S_TX_MONO 和 I2S_TX_MONO_FST_VLD 控制，具体如下表。请根据内存中存储的数据为单/双通道数据来配置该寄存器。

表 49.9-3. PDM 输出模式下 I2S*n* 取数逻辑

取数逻辑	模式	I2S_TX_MONO	I2S_TX_MONO_FST_VLD
每个 WS 沿都向 GDMA 发起取数请求	双声道	0	x
只在 WS 后半周期向 GDMA 发起取数请求	单声道	1	0
只在 WS 前半周期向 GDMA 发起取数请求	单声道	1	1

当 I2S*n* 处于主机 PDM 输出模式时，WS 信号的默认电平由 I2S_TX_WS_IDLE_POL 控制，WS 信号频率为 BCK 信号频率的一半。请参考 49.6 小节配置 BCK 信号的方式来配置 WS 信号，见图 49.9-3。

在**发送原始 PDM 数据**时，I2S*n* 通道数据由 I2S_TX_CHAN_MOD 和 I2S_TX_WS_IDLE_POL 控制，具体如下表。

表 49.9-4. 原始 PDM 数据的 I2S*n* 通道模式控制

模式	左声道	右声道	模式控制字段[1]	声道选择位[2]
双声道	发送左通道数据	发送右通道数据	0	x
单声道	发送左通道数据	发送左通道数据	1	0
	发送右通道数据	发送右通道数据	1	1
	发送右通道数据	发送右通道数据	2	0
	发送左通道数据	发送左通道数据	2	1
	发送 “single” 值[3]	发送右通道数据	3	0
	发送左通道数据	发送 “single” 值	3	1
	发送左通道数据	发送 “single” 值	4	0
	发送 “single” 值	发送右通道数据	4	1

[1] I2S_TX_CHAN_MOD
[2] I2S_TX_WS_IDLE_POL
[3] single 值等于 I2S_SINGLE_DATA 的值。

在 I2S*O* **启用 PCM-to-PDM 转换器**时，可以将 GDMA 中的 PCM 数据转为 PDM 数据，并按照 PDM 信号格式输出，配置 I2S_PCM2PDM_CONV_EN 启动该转换器。PCM-to-PDM 转换器的寄存器配置如下：

- 配置一线 PDM 输出格式或一/二线 DAC 输出格式，具体如下表。

表 49.9-5. I2S*O* PCM-to-PDM 数据输出

通道输出格式	I2S_TX_PDM_DAC_MODE_EN	I2S_TX_PDM_DAC_2OUT_EN
一线 PDM 输出格式[1]	0	x
一线 DAC 输出格式[2]	1	0
二线 DAC 输出格式	1	1

说明：

1. 此处定义的 PDM 输出格式是指一个 WS 周期发送两个通道的 SD 数据。
2. 此处定义的 DAC 输出格式是指一个 WS 周期发送一个通道的 SD 数据。
3. 一线 PDM/DAC 输出格式使用 I2S*n*O_Data_out 作为数据输出线。
4. 二线 DAC 输出格式使用 I2S*n*O_Data_out 和 I2S*n*O_Data1_out 作为数据输出线。
5. 从应用上来说，PDM 输出格式针对的是需要时钟信号、可以解码 PDM 格式编解码器的应用场景，DAC 输出格式针对的是不依赖时钟信号、没有 PDM 格式编解码器、且通过低通滤波可以直接恢复成模拟波形的应用场景。

- 配置采样频率和上采样率。
  I2S*O* 启用 PCM-to-PDM 转换器进行数据输出时，PDM 时钟频率即为 BCK 时钟频率。采样频率（$f_{Sampling}$，即 PCM 数据采样频率）与 BCK 时钟频率（PDM 数据采样频率）的关系如下：

$$f_{Sampling} = \frac{f_{BCK}}{OSR}$$

  其中，上采样率 OSR 和 I2S_TX_PDM_SINC_OSR2 的关系为：

$$OSR = I2S_TX_PDM_SINC_OSR2 \times 64$$

采样频率 $f_{Sampling}$ 和 I2S_TX_PDM_FS 的对应关系为：

$$f_{Sampling} = I2S_TX_PDM_FS \times 100$$

请根据需要的采样频率、上采样率以及 PDM 时钟频率进行寄存器配置。

PDM 通道配置示例

本示例针对 I2S*0*，示例中寄存器的配置如下：

- 配置 I2S_PCM2PDM_CONV_EN 为 0，即选择输出原始 PDM 数据。
- 配置 I2S_TX_MONO 为 0，即 I2S*0* 在 WS 的高低电平均通过 GDMA 从存储器取数。
- 配置 I2S_TX_CHAN_MOD 为 2，即选择单声道模式。
- 配置 I2S_TX_WS_IDLE_POL 为 1，则左声道和右声道均发送左通道数据，右声道数据将被舍弃。

配置完成后，假定存储器中的数据**经过数据格式控制之后**为：

Left	Right	Left	Right	...	Left	Right

说明：

1. 此处的数据并非存储器中的原始数据，而是经过数据格式控制之后的数据。
2. 表中的“Left”和“Right”代表通道数据，其位宽为通道有效数据位宽，请参考章节 49.9.1。

则数据经过通道模式控制后，将按照下图方式进行发送。

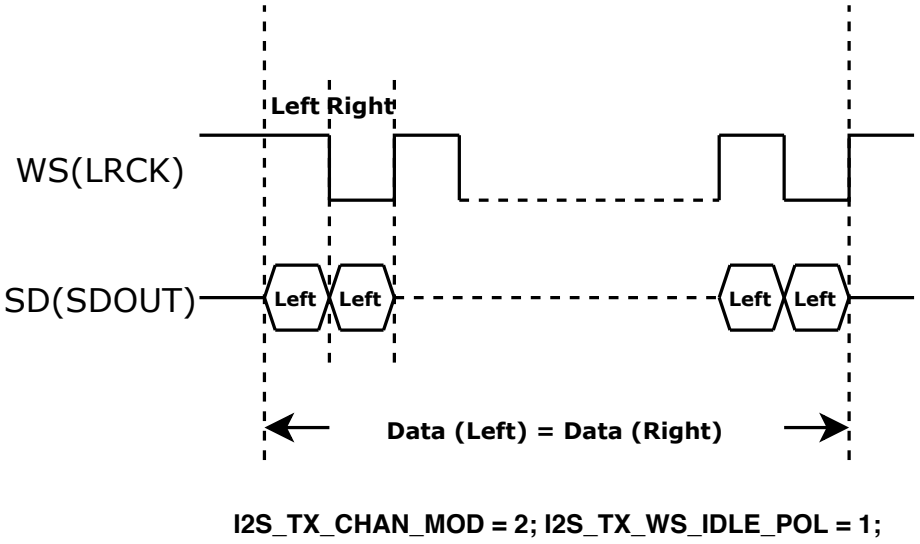


图 49.9-3. PDM 通道控制

49.9.3 同步计数器

ESP32-P4 I2S*n* TX 模块包含同步计数器，用于反映当前 I2S*n* 的数据发送情况。同步计数器可用于检测 I2S*n* 发送数据错误，或在多个设备使用 I2S*n* 发送数据时进行同步。

ESP32-P4 I2S*n* 共包含两种类型的同步计数器：

- I2S_TX_FIFO_CNT
  - 每次 I2S*n* TX FIFO 成功读取一个数据，该计数器加 1。
  - 置位 I2S_TX_FIFO_CNT_RST 可复位该同步计数器。
  - 该计数器位宽为 31 比特，溢出时自动归零。
- I2S_TX_BCK_CNT
  - 每次 I2S*n* TX 发送一个 BCK 周期，该计数器加 1。
  - 可以通过置位 I2S_TX_BCK_CNT_RST 来复位该同步计数器。
  - 该计数器位宽为 31 比特，溢出时自动归零。

说明：

- 每次 I2S*n* TX FIFO 发送一个数据，均意味着发送了一个通道数据，因此，当 I2S*n* 正确发送数据时，I2S_TX_BCK_CNT = I2S_TX_FIFO_CNT × 通道发送数据位宽（请参考小节 49.9.1.4）。
- 由于 I2S*n* TX 模块发送数据时，从 FIFO 取出数据到把数据发送至线上存在一些延迟，因此读到 I2S_TX_BCK_CNT 的值与 I2S_TX_FIFO_CNT 的值可能略微不对应。

应用场景举例

1. 多个设备 I2S TX 同步
   多个设备同时通过 I2S TX 发送数据时，由于时钟源的微小频率差，长时间发送数据后会出现同步问题。此时，可以通过读取每个设备的 I2S_TX_FIFO_CNT 值来判断发送音频数据的位置，从而进行同步操作。
2. I2S TX 稳定性检查
   当设备处于 GDMA 占用较多的应用场景时，可能会由于多路仲裁，导致 I2S TX 出现偶尔缺数。此时，检查 I2S_TX_BCK_CNT 与 I2S_TX_FIFO_CNT 的比例关系，可以确认设备运行 I2S TX 时的稳定性。

49.10 接收数据

I2S*n* 接收数据时，从外设接口读取数据，经过通道模式控制和数据格式控制，通过 GDMA 输入数据至内存。

49.10.1 通道模式控制

ESP32-P4 I2S*n* 支持 TDM 和 PDM 两种输入模式。置位 I2S_RX_TDM_EN 使能 TDM 输入模式，置位 I2S_RX_PDM_EN 使能 PDM 输入模式。

说明：
I2S_RX_TDM_EN 和 I2S_RX_PDM_EN 不能同时置位或同时清零。

49.10.1.1 TDM 输入模式

在 TDM 输入模式下，I2S*n* 最多支持 16 个接收通道。接收通道数由 I2S_RX_TDM_TOT_CHAN_NUM 控制。例如，配置 I2S_RX_TDM_TOT_CHAN_NUM 为 5，则通道 0 ~ 5 接收数据。

在接收数据的通道中，如果其对应的 I2S_RX_TDM_CHANn_EN 为：

- 0：则该通道数据无效，不会存入 RX FIFO；

- 1：则该通道数据有效，会被存入 RX FIFO。

当 I2S*n* 处于主机 TDM 输入模式下，WS 信号由 I2S_RX_WS_IDLE_POL 和 I2S_RX_TDM_WS_WIDTH 控制：

- I2S_RX_WS_IDLE_POL 的值为 WS 信号的默认电平；
- I2S_RX_TDM_WS_WIDTH 的值为在接收所有通道的过程中，WS 为默认电平的周期数。

另外，I2S_RX_HALF_SAMPLE_BITS 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

49.10.1.2 PDM 输入模式

在 PDM 输入模式下，I2S*n* 可以接收原始 PDM 数据，或者将 PDM 数据转换为 PCM 数据后接收（后者仅 I2S*0* 支持）。

在 PDM 输入模式下，I2S*n* 将通道中的串行数据转换为待输入数据。

当 I2S*n* 处于主机 PDM 输入模式下，WS 信号的默认电平由 I2S_RX_WS_IDLE_POL 控制，WS 信号频率为 BCK 信号频率的一半，请参考 49.6 小节配置 BCK 信号的方式配置 WS 信号。注意，在 PDM 输入模式下，请配置 I2S_RX_HALF_SAMPLE_BITS 的值与 I2S_RX_BITS_MOD 的值相同。

在 I2S*0* **启用 PDM-to-PCM 转换器**时，可以将接收的 PDM 数据转为 PCM 数据，再进行数据控制。配置 I2S*0*_RX_PDM2PCM_EN 启用该转换器。PDM-to-PCM 转换器的寄存器配置如下：

- 配置采样频率和下采样率。
  I2S*0* 使用 PDM-to-PCM 转换器接收数据时，PDM 时钟频率与主从机模式的关系如下：
  - 主机模式：PDM 时钟频率即为 BCK 时钟频率。
  - 从机模式：PDM 时钟频率由外部提供。

  采样频率 ($f_{Sampling}$) 与 PDM 时钟频率的关系如下：

$$f_{Sampling} = \frac{f_{PDM}}{DSR}$$

  其中，下采样率 DSR 和 I2S_RX_PDM_SINC_DSR_16_EN 的关系为：

$$DSR = I2S_RX_PDM_SINC_DSR_16_EN \times 64$$

  请根据需要的主从机模式、采样频率以及下采样率进行寄存器配置。
- 配置有效通道。
  启用 PDM-to-PCM 转换器时，最多支持 8 个通道的输入信号，寄存器配置以及对应通道如下表：

表 49.10-1. PDM-to-PCM 数据输入

输入数据信号	通道	使能寄存器
I2S*0*I_Data_in	左声道	I2S_RX_TDM_PDM_CHAN0_EN
	右声道	I2S_RX_TDM_PDM_CHAN1_EN
I2S*0*I1_Data_in	左声道	I2S_RX_TDM_PDM_CHAN2_EN
	右声道	I2S_RX_TDM_PDM_CHAN3_EN
I2S*0*I2_Data_in	左声道	I2S_RX_TDM_PDM_CHAN4_EN
	右声道	I2S_RX_TDM_PDM_CHAN5_EN
I2S*0*I3_Data_in	左声道	I2S_RX_TDM_PDM_CHAN6_EN
	右声道	I2S_RX_TDM_PDM_CHAN7_EN

49.10.2 数据格式控制

数据格式控制分为两个阶段：

- 第一阶段：从串行数据流输入转换为待输入数据，并存入 RX FIFO；
- 第二阶段：将待输入数据从 RX FIFO 中读出，并进行输入数据模式转换。

49.10.2.1 通道数据比特顺序

ESP32-P4 I2S*n* 通道数据将从高位到低位依次存入待输入数据，并由 I2S_RX_BIT_ORDER 控制是否调换比特序，当该值为：

- 0：不调换待输入数据的比特序；
- 1：调换待输入数据的比特序。

至此，数据格式控制的第一阶段完成，经过比特顺序控制的待输入数据被存入 RX FIFO。

49.10.2.2 通道储存数据位宽

I2S_RX_BITS_MOD 和 I2S_RX_24_FILL_EN 决定了每个通道的储存数据位宽，其可取值和对应的储存数据位宽如下表：

表 49.10-2. 通道储存数据位宽控制

通道储存数据位宽	I2S_RX_BITS_MOD	I2S_RX_24_FILL_EN
32	31	x
	23	1
24	23	0
16	15	x
8	7	x

49.10.2.3 通道接收数据位宽

I2S_RX_TDM_CHAN_BITS 决定了每个通道接收数据的位宽。

- 当每个通道储存数据位宽小于接收数据的位宽时，接收时仅会在接收数据中取储存位宽作为储存数据。此时，配置 I2S_RX_LEFT_ALIGN 为：
  - 0：储存数据位于接收数据低位；
  - 1：储存数据位于接收数据高位。
- 当每个通道接收数据的位宽小于储存数据位宽时，会将接收数据高位补 0，作为储存数据。

49.10.2.4 通道储存数据字节序

通道接收数据经过截取/补全后，成为了待储存数据，I2S_RX_BIG_ENDIAN 用于控制待储存数据的字节序。下表描述了不同通道储存数据位宽下，该寄存器对待储存数据的控制。

表 49.10-3. 通道储存数据字节序控制

通道储存数据位宽	原始数据	控制后数据	I2S_RX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

49.10.2.5 A 律/μ 律压缩/解压缩

I2S*n* 对排列好字节序的待储存数据会按照 32 位（缺省高位补 0）的方式进行 A 律/μ 律压缩/解压缩。

配置 I2S_RX_PCM_BYPASS 为：

- 0：进行压缩/解压缩
- 1：不进行压缩/解压缩

配置 I2S_RX_PCM_CONF 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2：μ 律解压缩
- 3：μ 律压缩

至此，数据格式控制部分全部完成，数据通过 GDMA 存入内存。

49.11 事件任务矩阵功能

在 ESP32-P4 中，I2S*n* 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 I2S*n* 的 ETM 任务，或者通过 I2S*n* 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 I2S*n* 相关的 ETM 任务和 ETM 事件。

I2S*n* 可接收的 ETM 任务有：

- I2S*n*_TASK_START_TX：触发时启动 I2S*n* TX 数据传输。
- I2S*n*_TASK_START_RX：触发时启动 I2S*n* RX 数据传输。
- I2S*n*_TASK_STOP_TX：触发时停止 I2S*n* TX 数据传输。
- I2S*n*_TASK_STOP_RX：触发时停止 I2S*n* RX 数据传输。

I2S*n* 可产生的 ETM 事件有：

- I2S*n*_EVT_TX_DONE：表示 I2S*n* TX 发送完成。
- I2S*n*_EVT_RX_DONE：表示 I2S*n* RX 接收完成。
- I2S*n*_EVT_X_WORDS_SENT：表示 I2S*n* TX 发送字数大于等于 I2S_ETM_TX_SEND_WORD_NUM 的值。

- I2S*n*_EVT_X_WORDS_RECEIVED：表示 I2S*n* RX 接收字数大于等于 I2S_ETM_RX_RECEIVE_WORD_NUM 的值。

在具体应用中，可以配置 I2S*n* 的 ETM 事件来触发 I2S*n* 的 ETM 任务，例如配置 I2S*n*_EVT_X_WORDS_SENT 事件触发 I2S*n*_TASK_STOP_TX 任务，从而以 ETM 的方式完成停止 I2S*n* 的操作。

49.12 I2S 中断

ESP32-P4 的 I2S*n* 可以生成 I2S*n*_INTR 中断信号并发送给中断矩阵。这一中断信号由以下 I2S*n* 的内部中断源生成。

- I2S*n*_TX_HUNG_INT：当发送数据超时，触发此中断。例如，I2S*n* 配置为从机发送模式，但主机长时间未提供 BCK 或 WS 信号，则将触发该中断。超时配置见寄存器 I2S_LC_HUNG_CONF_REG。
- I2S*n*_RX_HUNG_INT：当接收数据超时，触发此中断。例如，I2S*n* 配置为从机接收模式，但主机长时间未发送数据，则将触发该中断。超时配置见寄存器 I2S_LC_HUNG_CONF_REG。
- I2S*n*_TX_DONE_INT：当发送数据完成，触发此中断。
- I2S*n*_RX_DONE_INT：当接收数据完成，触发此中断。

> **说明：**
> 中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 49.14 寄存器列表。

49.13 软件配置流程

49.13.1 软件配置 I2S 发送流程

软件配置 I2S*n* 发送的流程如下：

1. 根据 49.6 小节的描述，配置时钟。
2. 根据表 49.4-1 的描述，配置信号管脚。
3. 根据主从机模式配置 I2S_TX_SLAVE_MOD：
   - 0：主机发送模式
   - 1：从机发送模式
4. 根据 49.9 小节的描述，配置正确的发送数据模式和发送通道模式，置位 I2S_TX_UPDATE。
5. 根据 49.7 小节的描述，复位 TX 模块和发送 FIFO。
6. 根据 49.12 小节的描述使能相应的中断。
7. 配置 GDMA 发送链表。
8. 根据需要置位 I2S_TX_STOP_EN，更多信息见章节 49.8.1。
9. 开始发送数据：

- 主机模式下，等待 I2S*n* 从设备配置完成后，置位 I2S_TX_START 开始发送数据；
- 从机模式下，置位 I2S_TX_START。I2S*n* 主设备提供 BCK 和 WS 信号后，开始发送数据。

10. 等待步骤 6 设置的中断信号，或查询 I2S_TX_IDLE 检查传输是否结束：
    - 0：发送设备为工作状态；
    - 1：发送设备为空闲状态。
11. 清零 I2S_TX_START 停止发动数据。

49.13.2 软件配置 I2S 接收流程

软件配置 I2S*n* 接收模式的流程如下：

1. 根据 49.6 小节的描述，配置时钟。
2. 根据表 49.4-1 的描述，配置信号管脚。
3. 配置 I2S_RX_SLAVE_MOD 选择需要的模式：
    - 0：主机接收模式
    - 1：从机接收模式
4. 根据 49.10 小节的描述，配置正确的接收通道模式和接收数据模式，置位 I2S_RX_UPDATE。
5. 根据 49.7 小节的描述，复位 RX 模块和接收 FIFO。
6. 根据 49.12 小节的描述使能相应的中断。
7. 配置 GDMA 接收链表，并在 I2S_RX_EOF_NUM_REG 中配置接收数据长度。
8. 开始接收数据：
    - 在主机模式下，等待从机准备好后，置位 I2S_RX_START 开始接收数据；
    - 在从机模式下，置位 I2S_RX_START，等待主机提供 BCK 和 WS 信号后开始接收数据。
9. 接收的数据由 GDMA 根据配置存到存储器的指定地址。最终产生步骤 6 中设置的中断。

49.14 寄存器列表

本小节的所有地址均为相对于 I2S*n* 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
I2S_INT_RAW_REG	I2S 原始中断寄存器	0x000C	R/SS/WTC
I2S_INT_ST_REG	I2S 中断状态寄存器	0x0010	RO
I2S_INT_ENA_REG	I2S 中断使能寄存器	0x0014	R/W
I2S_INT_CLR_REG	I2S 中断清除寄存器	0x0018	WT
RX 控制和配置寄存器			
I2S_RX_CONF_REG	I2S RX 配置寄存器	0x0020	varies
I2S_RX_CONF1_REG	I2S RX 配置寄存器 1	0x0028	R/W
I2S_RX_PDM2PCM_CONF_REG	I2S RX PDM-to-PCM 配置寄存器	0x0048	R/W
I2S_RX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0050	R/W
I2S_RXEOF_NUM_REG	I2S RX 数据大小控制寄存器	0x0064	R/W
TX 控制和配置寄存器			
I2S_TX_CONF_REG	I2S TX 配置寄存器	0x0024	varies
I2S_TX_CONF1_REG	I2S TX 配置寄存器 1	0x002C	R/W
I2S_TX_PCM2PDM_CONF_REG	I2S TX PCM-to-PDM 配置寄存器（仅适用于 I2S*0*）	0x0040	R/W
I2S_TX_PCM2PDM_CONF1_REG	I2S TX PCM-to-PDM 配置寄存器（仅适用于 I2S*0*）	0x0044	R/W
I2S_TX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0054	R/W
RX 时钟和时序寄存器			
I2S_RX_TIMING_REG	I2S RX 时序控制寄存器（部分字段仅适用于 I2S*0*）	0x0058	R/W
TX 时钟和时序寄存器			
I2S_TX_TIMING_REG	I2S TX 时序控制寄存器（部分字段仅适用于 I2S*0*）	0x005C	R/W
控制和配置寄存器			
I2S_LC_HUNG_CONF_REG	I2S 超时配置寄存器	0x0060	R/W
I2S_CONF_SIGLE_DATA_REG	I2S single 数据寄存器	0x0068	R/W
TX 状态寄存器			
I2S_STATE_REG	I2S TX 状态寄存器	0x006C	RO
ETM 寄存器			
I2S_ETM_CONF_REG	I2S ETM 配置寄存器	0x0070	R/W
同步计数器寄存器			
I2S_FIFO_CNT_REG	I2S 同步计数器寄存器	0x0074	varies
I2S_BCK_CNT_REG	I2S 同步计数器寄存器	0x0078	varies
时钟寄存器			
I2S_CLK_GATE_REG	时钟门控寄存器	0x007C	R/W

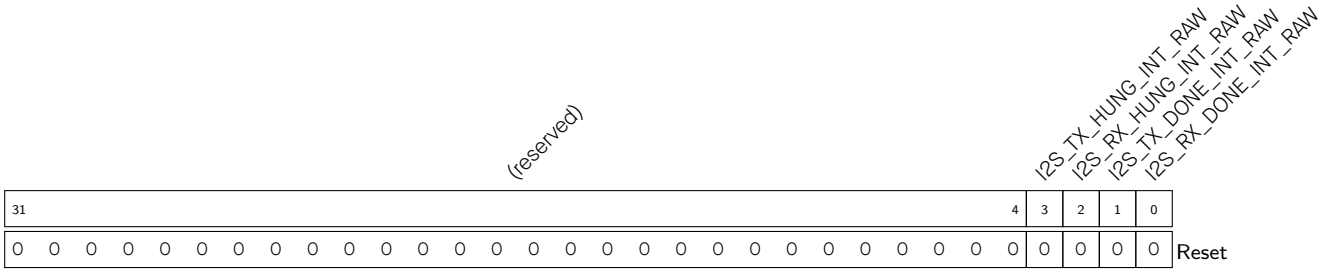
名称	描述	地址	访问
版本寄存器			
I2S_DATE_REG	版本控制寄存器	0x0080	R/W

49.15 寄存器

本小节的所有地址均为相对于 I2S*n* 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 49.1. I2S_INT_RAW_REG (0x000C)



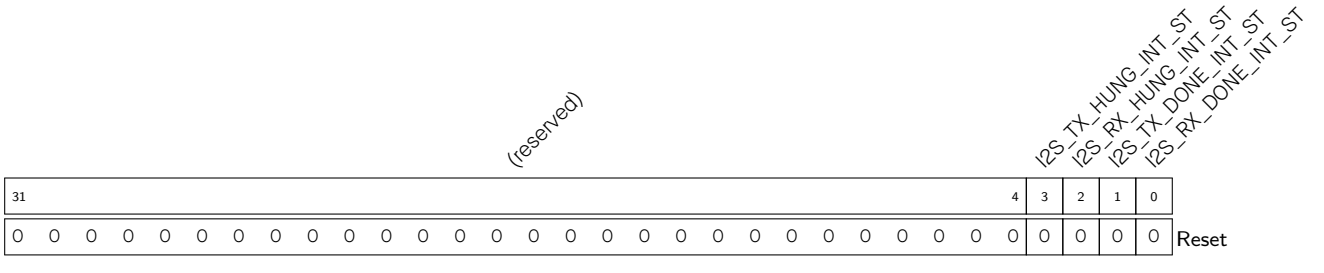
I2S_RX_DONE_INT_RAW I2S_RX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

I2S_TX_DONE_INT_RAW I2S_TX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

I2S_RX_HUNG_INT_RAW I2S_RX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)

I2S_TX_HUNG_INT_RAW I2S_TX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)

Register 49.2. I2S_INT_ST_REG (0x0010)



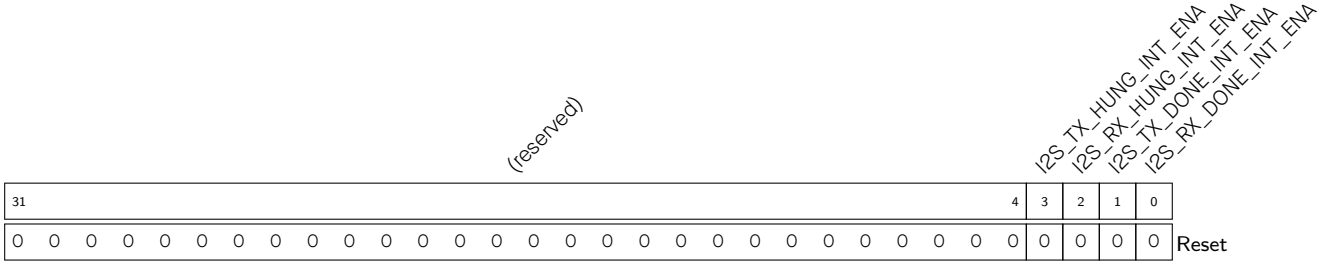
I2S_RX_DONE_INT_ST I2S_RX_DONE_INT 中断的屏蔽中断状态。(RO)

I2S_TX_DONE_INT_ST I2S_TX_DONE_INT 中断的屏蔽中断状态。(RO)

I2S_RX_HUNG_INT_ST I2S_RX_HUNG_INT 中断的屏蔽中断状态。(RO)

I2S_TX_HUNG_INT_ST I2S_TX_HUNG_INT 中断的屏蔽中断状态。(RO)

Register 49.3. I2S_INT_ENA_REG (0x0014)



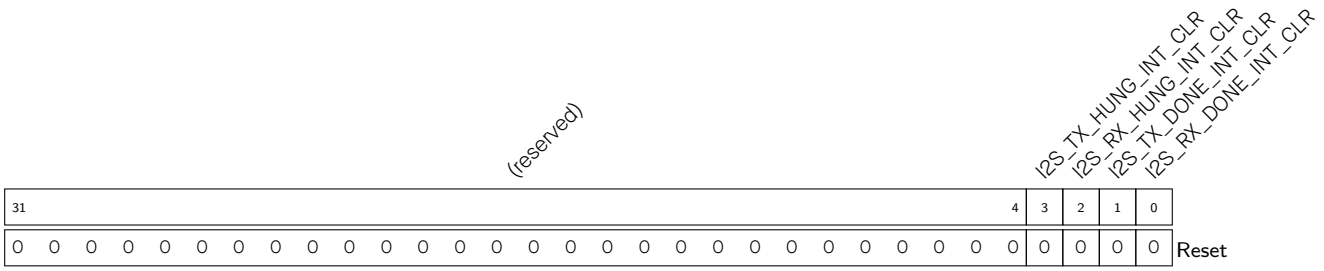
I2S_RX_DONE_INT_ENA 写 1 使能 I2S_RX_DONE_INT。(R/W)

I2S_TX_DONE_INT_ENA 写 1 使能 I2S_TX_DONE_INT。(R/W)

I2S_RX_HUNG_INT_ENA 写 1 使能 I2S_RX_HUNG_INT。(R/W)

I2S_TX_HUNG_INT_ENA 写 1 使能 I2S_TX_HUNG_INT。(R/W)

Register 49.4. I2S_INT_CLR_REG (0x0018)



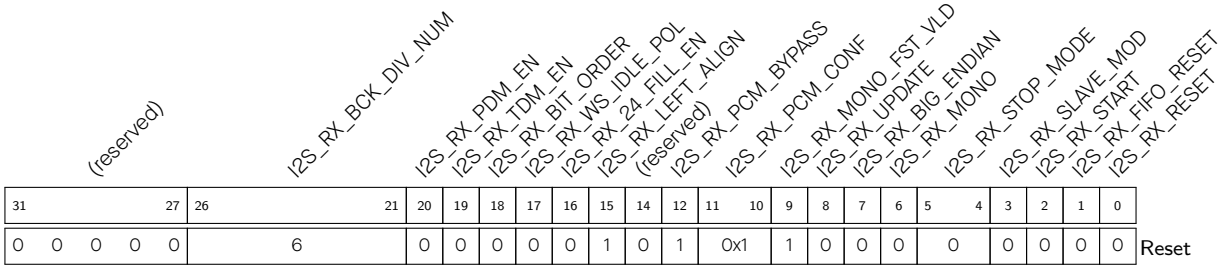
I2S_RX_DONE_INT_CLR 写 1 清除 I2S_RX_DONE_INT。(WT)

I2S_TX_DONE_INT_CLR 写 1 清除 I2S_TX_DONE_INT。(WT)

I2S_RX_HUNG_INT_CLR 写 1 清除 I2S_RX_HUNG_INT。(WT)

I2S_TX_HUNG_INT_CLR 写 1 清除 I2S_TX_HUNG_INT。(WT)

Register 49.5. I2S_RX_CONF_REG (0x0020)



I2S_RX_RESET 配置是否复位接收单元。

0：无效

1：复位

(WT)

I2S_RX_FIFO_RESET 配置是否复位 RX FIFO。

0：无效

1：复位

(WT)

I2S_RX_START 配置是否开始接收数据。

0：无效

1：开始

(R/W/SC)

I2S_RX_SLAVE_MOD 配置是否使能从机接收模式。

0：使能主机模式

1：使能从机模式

(R/W)

I2S_RX_STOP_MODE 配置 I2S RX 何时暂停接收。

0：只有 I2S_RX_START 被清除时暂停接收

1：I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收

2：接收单元会在 I2S_RX_START 被清除，或者 DMA RX FIFO 满时暂停接收

(R/W)

I2S_RX_MONO 配置是否使能接收单元的单声道模式。

0：禁用

1：使能

(R/W)

I2S_RX_BIG_ENDIAN 配置 I2S RX 字节序。

0：低字节数据写入低位地址

1：低字节数据写入高位地址

(R/W)

见下页...

Register 49.5. I2S_RX_CONF_REG (0x0020)

接上页...

I2S_RX_UPDATE 配置是否将 I2S RX 寄存器从 APB 时钟域同步到 I2S RX 时钟域。寄存器完成更新后，该位将由硬件清除。

0：无效

1：同步

(R/W/SC)

I2S_RX_MONO_FST_VLD 配置 I2S RX 单声道模式下的有效数据通道。

0：第二个通道数据有效

1：第一个通道数据有效

(R/W)

I2S_RX_PCM_CONF 配置 I2S RX 压缩/解压缩模式。

0 (atol)：A 律解压缩

1 (ltoa)：A 律压缩

2 (utol)：μ 律解压缩

3 (ltou)：μ 律压缩

(R/W)

I2S_RX_PCM_BYPASS 配置接收数据是否将绕过压缩/解压缩单元。

0：无效

1：绕过压缩/解压缩单元

(R/W)

I2S_RX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0：上升沿对齐

1：相隔一个周期

(R/W)

I2S_RX_LEFT_ALIGN 配置 I2S RX 对齐模式。

0：右对齐模式

1：左对齐模式

(R/W)

I2S_RX_24_FILL_EN 配置 24 位通道数据位的储存数据位数。

0：将 24 位通道数据位保存到 24 位储存数据

1：将 24 位通道数据位保存到 32 位储存数据（多余的位填充为 0）

(R/W)

I2S_RX_WS_IDLE_POL 配置 WS 与接收数据通道的关系。

0：WS 为 0 时，接收左通道数据；WS 为 1 时，接收右通道数据

1：WS 为 1 时，接收左通道数据；WS 为 0 时，接收右通道数据

(R/W)

见下页...

Register 49.5. I2S_RX_CONF_REG (0x0020)

接上页...

I2S_RX_BIT_ORDER 配置是否调换 I2S RX 待接收数据的位顺序。

0：不调换

1：调换

(R/W)

I2S_RX_TDM_EN 配置是否使能 I2S TDM RX 模式。

0：禁用

1：使能

(R/W)

I2S_RX_PDM_EN 配置是否使能 I2S PDM RX 模式。

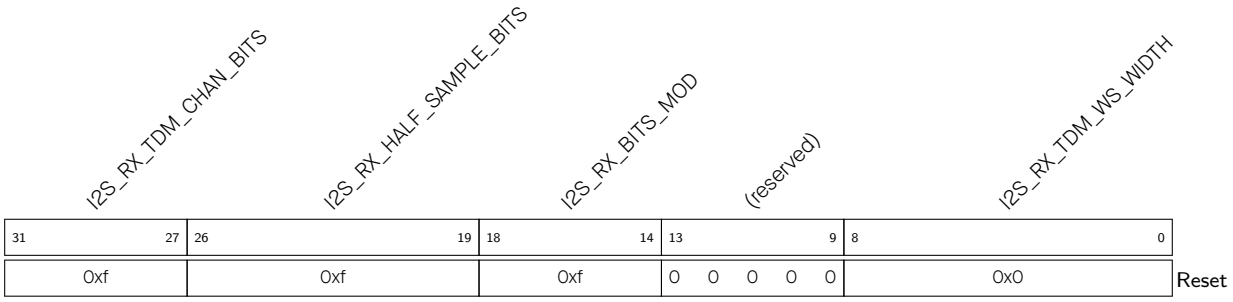
0：禁用

1：使能

(R/W)

I2S_RX_BCK_DIV_NUM 配置在 RX 模式下 BCK 时钟的分频系数。注意，不可配置为 1。(R/W)

Register 49.6. I2S_RX_CONF1_REG (0x0028)



I2S_RX_TDM_WS_WIDTH 配置 I2S*n*I_WS_out（WS 默认电平）空闲时在 TDM 模式下的时长。I2S*n*I_WS_out 空闲时在 TDM 模式下的时长 = (I2S_RX_TDM_WS_WIDTH[8:0] +1) x T_BCK。(R/W)

I2S_RX_BITS_MOD 配置在 RX 模式下接收通道的有效数据位长度。

7：所有有效的通道数据均为 8 位模式

15：所有有效的通道数据均为 16 位模式

23：所有有效的通道数据均为 24 位模式

31：所有有效的通道数据均为 32 位模式

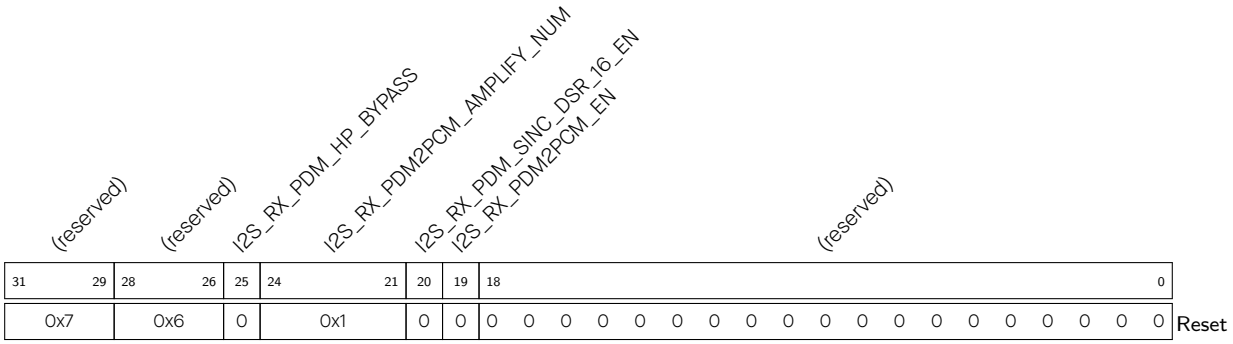
其他值无效。

(R/W)

I2S_RX_HALF_SAMPLE_BITS 配置 I2S RX 单次采样比特数。一个 WS 信号中 BCK 持续的周期 = I2S_RX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_RX_TDM_CHAN_BITS 配置在 TDM RX 模式下每个通道的 RX 数据位长。RX 数据位长 = I2S_RX_TDM_CHAN_BITS + 1。(R/W)

Register 49.7. I2S_RX_PDM2PCM_CONF_REG（仅适用于 I2S0） (0x0048)



I2S_RX_PDM2PCM_EN 配置是否在 RX 模式下使能 PDM-to-PCM 转换器。

0：禁用

1：使能

(R/W)

I2S_RX_PDM_SINC_DSR_16_EN 配置 PDM RX 滤波器组 1 的向下采样率。

0：64

1：128

(R/W)

I2S_RX_PDM2PCM_AMPLIFY_NUM 配置 PDM-to-PCM RX 放大系数。PCM 数据在输出前将乘以该值。

(R/W)

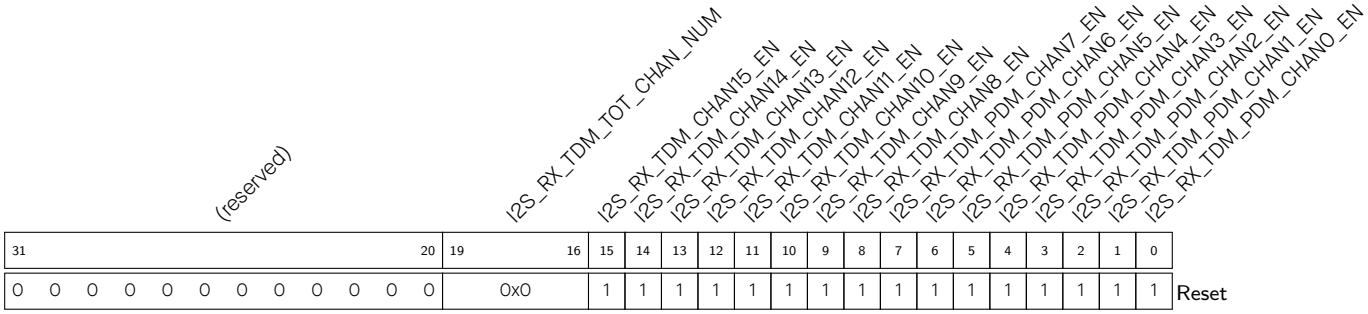
I2S_RX_PDM_HP_BYPASS 配置 PDM-to-PCM RX 是否绕过 HP 滤波器。

0：不绕过

1：绕过

(R/W)

Register 49.8. I2S_RX_TDM_CTRL_REG (0x0050)



I2S_RX_TDM_PDM_CHAN*n*_EN (*n*：0-7) 配置是否使能 I2S TDM 或 PDM RX 通道 *n* 的有效输入数据。

0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

I2S_RX_TDM_CHAN*n*_EN (*n*：8-15) 配置是否使能 I2S TDM RX 通道 *n* 的有效输入数据。

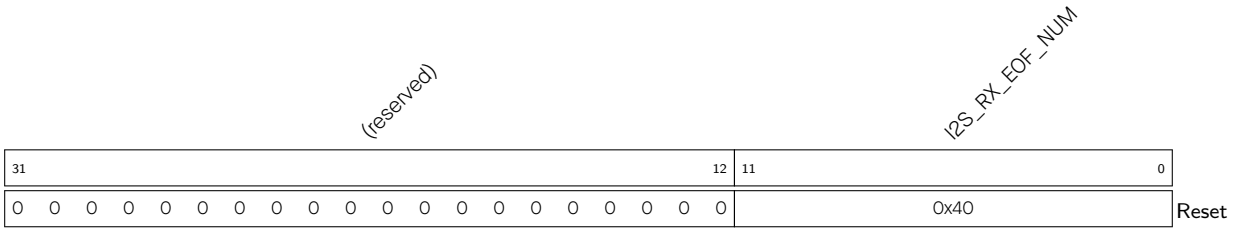
0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

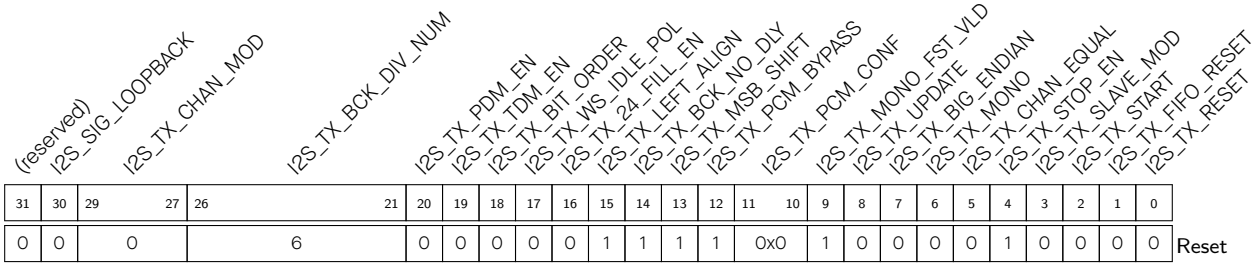
I2S_RX_TDM_TOT_CHAN_NUM 配置在 I2S TDM RX 模式下使用的通道总数。使用的通道总数 = I2S_RX_TDM_TOT_CHAN_NUM + 1。(R/W)

Register 49.9. I2S_RXEOF_NUM_REG (0x0064)



I2S_RX_EOF_NUM 配置接收数据的长度。接收数据长度 = (I2S_RX_BITS_MOD + 1) x (I2S_RX_EOF_NUM + 1)。如果接收数据的长度达到该值，则将在已配置的 GDMA RX 通道中触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。(R/W)

Register 49.10. I2S_TX_CONF_REG (0x0024)



I2S_TX_RESET 配置是否复位发送单元。

0：无效

1：复位

(WT)

I2S_TX_FIFO_RESET 配置是否复位 TX FIFO。

0：无效

1：复位

(WT)

I2S_TX_START 配置是否开始发送数据。

0：无效

1：开始发送

(R/W/SC)

I2S_TX_SLAVE_MOD 配置是否使能从机发送模式。

0：使能主机模式

1：使能从机模式

(R/W)

I2S_TX_STOP_EN 配置当 TX FIFO 为空时，发送单元是否停止输出 BCK 和 WS 信号。

0：无效

1：停止

(R/W)

I2S_TX_CHAN_EQUAL 配置在 I2S TX 单声道模式或 TDM 模式下，是否使能左声道数据等于右声道数据。

0：在 I2S TX 单声道模式或 TDM 模式下，I2S_SINGLE_DATA 为无效的通道数据

1：在 I2S TX 单声道模式或 TDM 模式下，左声道数据等于右声道数据

(R/W)

I2S_TX_MONO 配置是否使能发送单元的单声道模式。

0：禁用

1：使能

(R/W)

I2S_TX_BIG_ENDIAN 配置 I2S TX 字节序。

0：低字节数据写入低位地址

1：低字节数据写入高位地址

(R/W)

见下页...

Register 49.10. I2S_TX_CONF_REG (0x0024)

接上页... (R/W)

I2S_TX_UPDATE 配置是否将 I2S TX 寄存器从 APB 时钟域同步到 I2S TX 时钟域。

0：无效

1：同步

寄存器更新完成后，此位将由硬件清除。

(R/W/SC)

I2S_TX_MONO_FST_VLD 配置在 I2S TX 单声道模式下的有效数据通道。

0：第二个通道数据有效

1：第一个通道数据有效

(R/W)

I2S_TX_PCM_CONF 配置 I2S TX 压缩/解压缩模式。

0 (atol)：A 律解压缩

1 (ltoa)：A 律压缩

2 (utol)：μ 律解压缩

3 (ltou)：μ 律压缩

(R/W)

I2S_TX_PCM_BYPASS 配置是否发送数据将绕过压缩/解压缩单元。

0：无效

1：绕过

(R/W)

I2S_TX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0：上升沿对齐

1：相隔一个周期

(R/W)

I2S_TX_BCK_NO_DLY 配置在主机模式下，BCK 上升沿和下降沿生成来源。

0：用 I2S*n*O_BCK_in 输入的 BCK 构建上升沿和下降沿

1：用 I2S*n* TX 模块时钟分频构建上升沿和下降沿

(R/W)

I2S_TX_LEFT_ALIGN 配置 I2S TX 对齐模式。

0：右对齐模式

1：左对齐模式

(R/W)

I2S_TX_24_FILL_EN 配置 24 位数据的发送格式。

0：将 24 位数据以 24 位的格式发送出去

1：将 24 位数据以 32 位的格式发送出去（不足的位用 0 填充）

(R/W)

见下页...

Register 49.10. I2S_TX_CONF_REG (0x0024)

接上页...

I2S_TX_WS_IDLE_POL 配置 WS 与发送数据通道的关系。

0：WS 为 0 时，发送左通道数据；WS 为 1 时，发送右通道数据

1：WS 为 1 时，发送左通道数据；WS 为 0 时，发送右通道数据

I2S_TX_BIT_ORDER 配置是否调换 I2S TX 待发送有效数据的位顺序。

0：不调换

1：调换

(R/W)

I2S_TX_TDM_EN 配置是否使能 I2S TDM TX 模式。

0：禁用

1：使能

(R/W)

I2S_TX_PDM_EN 配置是否使能 I2S PDM TX 模式。

0：禁用

1：使能

(R/W)

I2S_TX_BCK_DIV_NUM 配置在 TX 模式下 BCK 时钟的分频系数。注意，不可配置为 1。(R/W)

I2S_TX_CHAN_MOD 配置 I2S 发送单元通道，更多信息见表 49.9-4。(R/W)

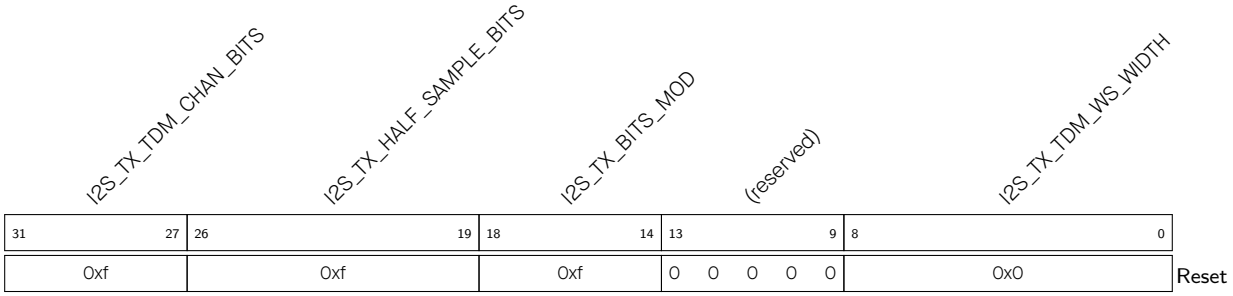
I2S_SIG_LOOPBACK 配置是否使能发送单元和接收单元共享 WS 和 BCK 信号。

0：禁用

1：使能

(R/W)

Register 49.11. I2S_TX_CONF1_REG (0x002C)



I2S_TX_TDM_WS_WIDTH 配置 I2S*n*O_WS_out（WS 默认电平）空闲时在 TDM 模式下的时长。I2S*n*O_WS_out 空闲时在 TDM 模式下的时长 = (I2S_TX_TDM_WS_WIDTH[8:0] +1) x T_BCK。(R/W)

I2S_TX_BITS_MOD 配置 TX 模式下发送通道的有效数据位长度。

7：所有有效的通道数据均为 8 位模式

15：所有有效的通道数据均为 16 位模式

23：所有有效的通道数据均为 24 位模式

31：所有有效的通道数据均为 32 位模式

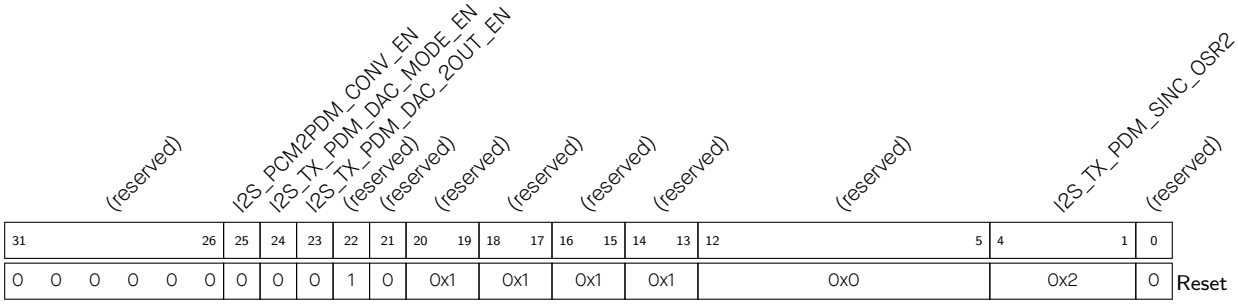
其他值无效。

(R/W)

I2S_TX_HALF_SAMPLE_BITS 配置 TX 单次采样比特数。一个 WS 信号中 BCK 持续的周期长 = I2S_TX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_TX_TDM_CHAN_BITS 配置在 TDM TX 模式下每个通道的 TX 数据位。TX 数据位 = I2S_TX_TDM_CHAN_BITS + 1。(R/W)

Register 49.12. I2S_TX_PCM2PDM_CONF_REG（仅适用于 I2S0） (0x0040)



I2S_TX_PDM_SINC_OSR2 配置 I2S TX PDM 的 OSR 值。(R/W)

I2S_TX_PDM_DAC_2OUT_EN 配置 DAC 输出模式。

0：使能 1-line DAC 输出模式

1：使能 2-line DAC 输出模式

仅在 I2S_TX_PDM_DAC_MODE_EN 为 1 时有效。

(R/W)

I2S_TX_PDM_DAC_MODE_EN 配置使能 1-line PDM 输出模式或 DAC 输出模式。

0：使能 1-line PDM 输出模式

1：使能 DAC 输出模式

(R/W)

I2S_PCM2PDM_CONV_EN 配置是否使能 I2S TX PCM-to-PDM 转换器。

0: 禁用

1: 使能

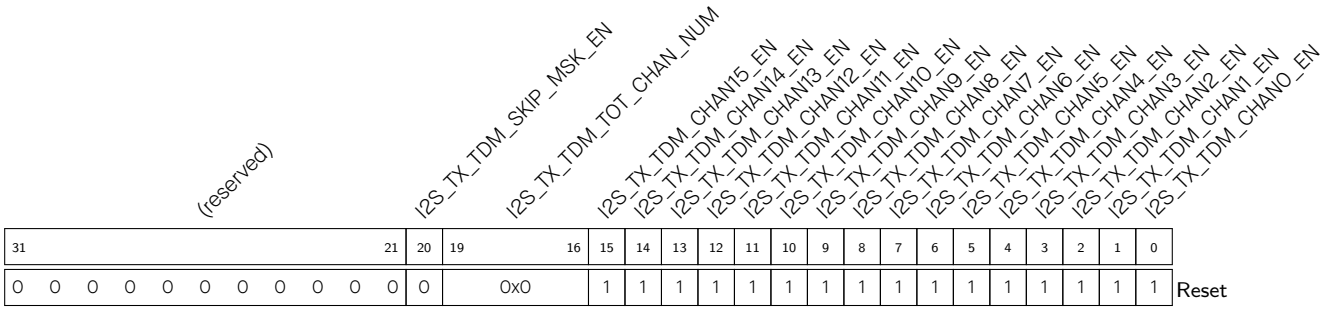
(R/W)

Register 49.13. I2S_TX_PCM2PDM_CONF1_REG (0x0044)



I2S_TX_PDM_FS 配置 I2S TX PDM 上采样率。(R/W)

Register 49.14. I2S_TX_TDM_CTRL_REG (0x0054)



I2S_TX_TDM_CHAN*n*_EN (*n*: 0-15) 配置是否使能 I2S TDM TX 通道 *n* 的有效输出数据。

0：通道发送数据由 I2S_TX_CHAN_EQUAL 和 I2S_SINGLE_DATA 控制，参考章节 49.9.2.1

1：使能

(R/W)

I2S_TX_TDM_TOT_CHAN_NUM 配置在 I2S TDM TX 模式下使用的通道总数。通道总数 = I2S_TX_TDM_TOT_CHAN_NUM + 1。(R/W)

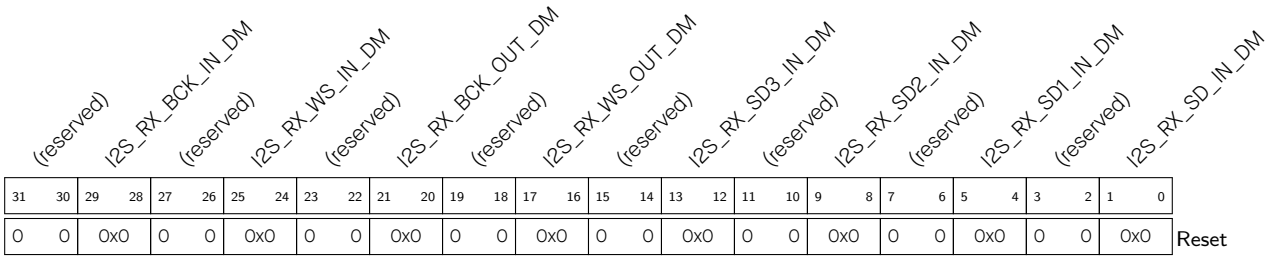
I2S_TX_TDM_SKIP_MSK_EN 配置 GDMA TX buffer 的发送数据。

0：DMA TX buffer 中的数据都用于已启用的通道，未启用的通道发送时不会读取 GDMA TX buffer 中的数据

1：DMA TX buffer 中的数据在所有通道发送时均会被读取，因此在未启用的通道发送时 GDMA TX buffer 中的数据会被跳过

(R/W)

Register 49.15. I2S_RX_TIMING_REG (0x0058)



I2S_RX_SD_IN_DM 配置 I2S RX SD 输入信号的延迟模式。

0：旁路

1：上升沿延迟

2：下降沿延迟

3：保留

(R/W)

I2S_RX_SD1_IN_DM（仅适用于 I2S*0*） 配置 I2S RX SD1 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_SD2_IN_DM（仅适用于 I2S*0*） 配置 I2S RX SD2 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_SD3_IN_DM（仅适用于 I2S*0*） 配置 I2S RX SD3 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

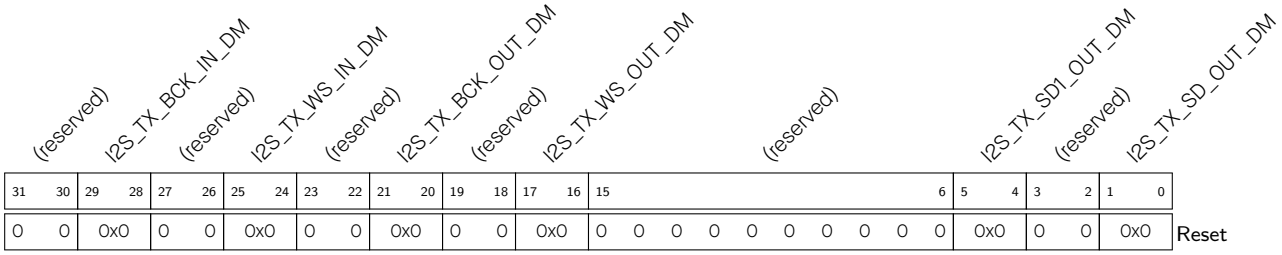
I2S_RX_WS_OUT_DM 配置 I2S RX WS 输出信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_BCK_OUT_DM 配置 I2S RX BCK 输出信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_WS_IN_DM 配置 I2S RX WS 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_BCK_IN_DM 配置 I2S RX BCK 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

Register 49.16. I2S_TX_TIMING_REG (0x005C)



I2S_TX_SD_OUT_DM 配置 I2S TX SD 输出信号的延迟模式。

0：旁路

1：上升沿延迟

2：下降沿延迟

3：保留

(R/W)

I2S_TX_SD1_OUT_DM（仅适用于 I2S0） 配置 I2S TX SD1 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

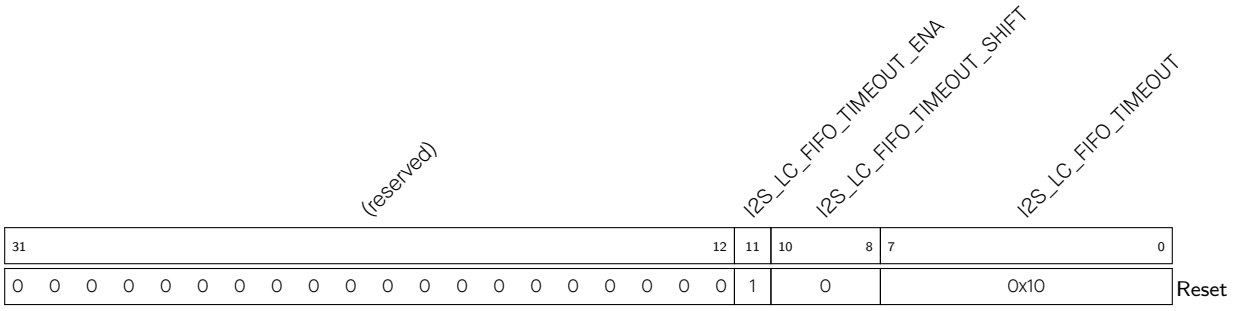
I2S_TX_WS_OUT_DM 配置 I2S TX WS 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_OUT_DM 配置 I2S TX BCK 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_WS_IN_DM 配置 I2S TX WS 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_IN_DM 配置 I2S TX BCK 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

Register 49.17. I2S_LC_HUNG_CONF_REG (0x0060)



I2S_LC_FIFO_TIMEOUT 配置 FIFO 的超时阈值。FIFO Hung 计数器在 I2S 处于错误状态，并且每次分频滴答计数器达到滴答计数器的阈值时会加一。FIFO Hung 计数器等于 FIFO 的超时阈值时，将触发 I2S_TX_HUNG_INT 中断或 I2S_RX_HUNG_INT 中断。(R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT 配置分频滴答计数器的阈值。分频滴答计数器会在 I2S 处于错误状态，并且每个系统 (APB) 时钟上升沿加一。计数器值大于等于 $88000/2^{I2S_LC_FIFO_TIMEOUT_SHIFT}$ 时，复位滴答计数器。(R/W)

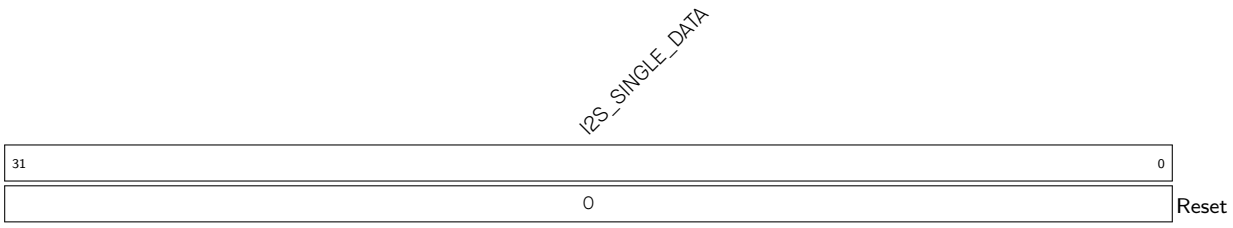
I2S_LC_FIFO_TIMEOUT_ENA 配置是否使能 FIFO 超时。

0：禁用

1：使能

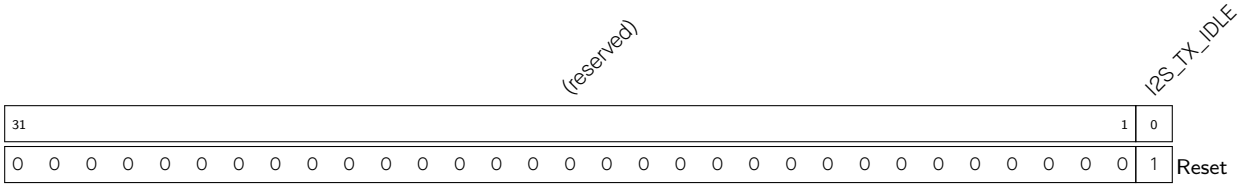
(R/W)

Register 49.18. I2S_CONF_SIGLE_DATA_REG (0x0068)



I2S_SINGLE_DATA 配置用于发送的通道常量数据。(R/W)

Register 49.19. I2S_STATE_REG (0x006C)



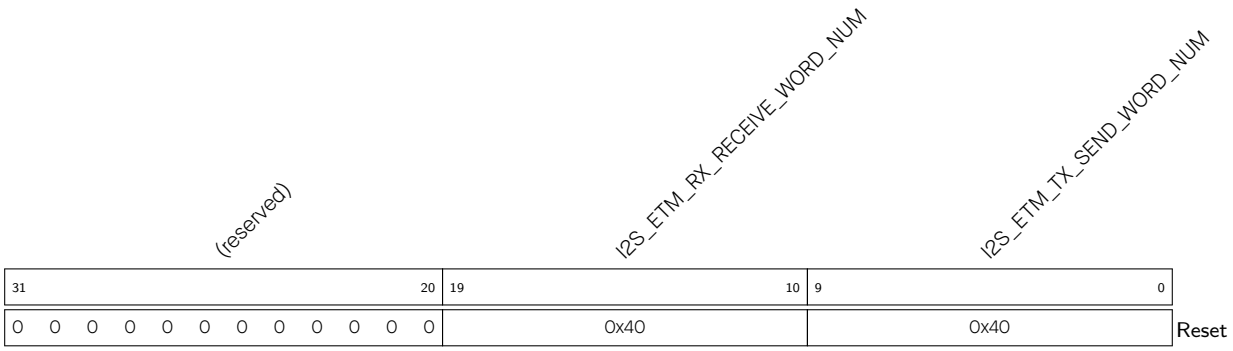
I2S_TX_IDLE 表示 I2S TX 单元的工作状态。

0：I2S TX 单元处于工作状态

1：I2S TX 单元处于空闲状态

(RO)

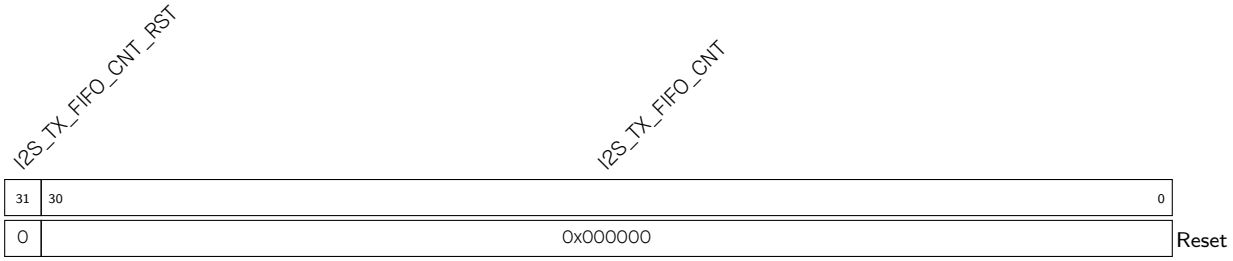
Register 49.20. I2S_ETM_CONF_REG (0x0070)



I2S_ETM_TX_SEND_WORD_NUM 配置触发 I2S_TX_X_WORDS_SENT 事件的阈值。当发送字数为 I2S_ETM_TX_SEND_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

I2S_ETM_RX_RECEIVE_WORD_NUM 配置触发 ETM I2S_RX_X_WORDS_RECEIVED 事件的阈值。当接收字数为 I2S_ETM_RX_RECEIVE_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

Register 49.21. I2S_FIFO_CNT_REG (0x0074)



I2S_TX_FIFO_CNT 配置 TX FIFO 计数值。(RO)

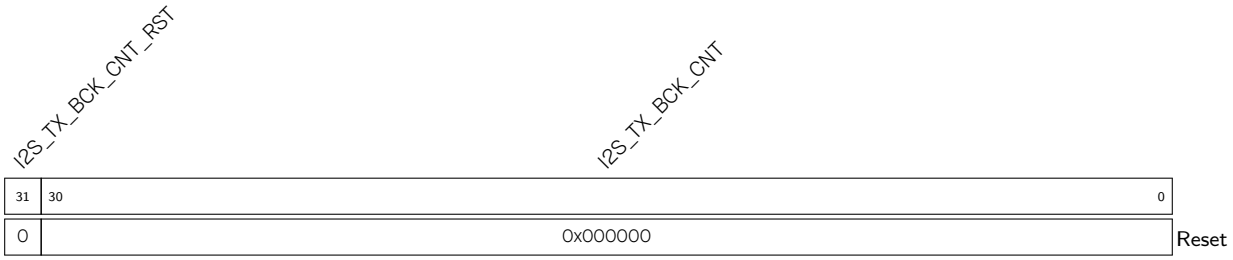
I2S_TX_FIFO_CNT_RST 配置是否复位 TX FIFO 计数器。

0：无效

1：复位

(WT)

Register 49.22. I2S_BCK_CNT_REG (0x0078)



I2S_TX_BCK_CNT 配置 TX BCK 计数值。(RO)

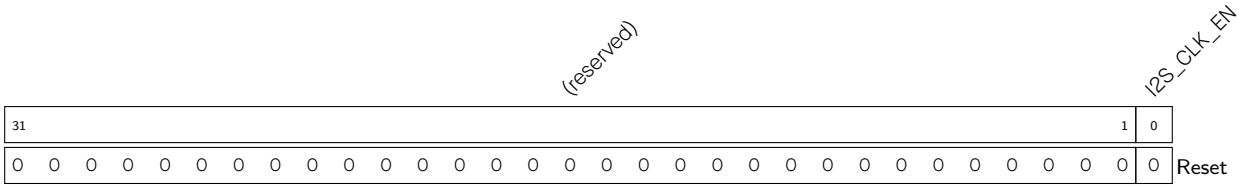
I2S_TX_BCK_CNT_RST 配置是否复位 TX BCK 计数器。

0：无效

1：复位

(WT)

Register 49.23. I2S_CLK_GATE_REG (0x007C)



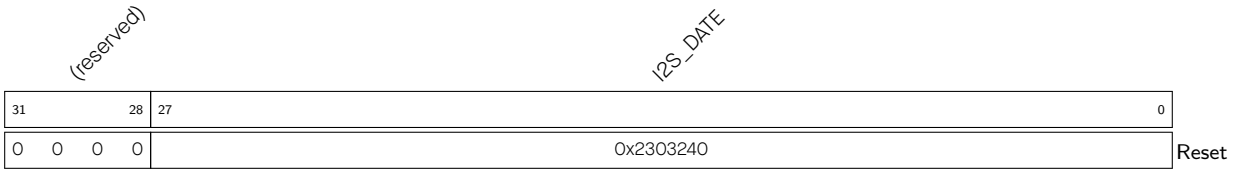
I2S_CLK_EN 配置是否启用时钟门控。

0：禁用

1：启用

(R/W)

Register 49.24. I2S_DATE_REG (0x0080)



I2S_DATE 版本控制寄存器。(R/W)

第 50 章

LP I2S 控制器

50.1 概况

ESP32-P4 内置一个 LP I2S 接口，为人声检测模块 (Voice Activity Detection, VAD) 和一些低功耗模式下的数字音频应用提供了数据接收通信接口。

I2S 标准总线定义了三种信号——串行时钟信号 BCK、字选择信号 WS 和串行数据信号 SD。一个基础的 I2S 数据总线包含一个主机和一个从机。主机和从机的角色在通信过程中保持不变。

ESP32-P4 的 LP I2S 模块包含独立的 RX 模块，能够在芯片以最低功耗运行时，完成数据接收的功能。相对于 HP I2S，即 I2S*0*、I2S*1* 和 I2S*2*，LP I2S 不支持 DMA 访问，而是由一块独立的内部储存器来存储数据。

> **说明：**
> 如需了解 I2S 相关术语，请参考章节 49 *I2S 控制器 (I2S)* 中的 49.2 术语 小节。

50.2 主要特性

LP I2S 模块具有以下特性：

- RX 模块支持主机模式和从机模式
- 支持多种音频标准：
  - TDM Philips 标准
  - TDM MSB 对齐标准
  - TDM PCM 标准
  - PDM 标准
- 支持多种 RX 模式：
  - TDM RX 模式，最多支持 2 通道
  - PDM RX 模式
    * 支持原始 PDM 数据接收
    * 支持将 PDM 数据转换为 PCM 数据接收，最多支持 2 通道
- 可配置采样时钟，支持多种采样频率
- 支持 16 位的数据位宽
- 支持 LP I2S 接口中断

50.3 架构概览

图 50.3-1 为 ESP32-P4 LP I2S 模块的结构框图。LP I2S 模块包含：

- 独立的 RX 模块
- 输入输出时序调节单元（I/O 同步）
- 时钟分频器（时钟发生器）
- 64 x 32 位 RX FIFO
- 256 x 32 位 LP I2S 数据存储空间
- 人声检测模块 (Voice Activity Detection, VAD) 通信接口

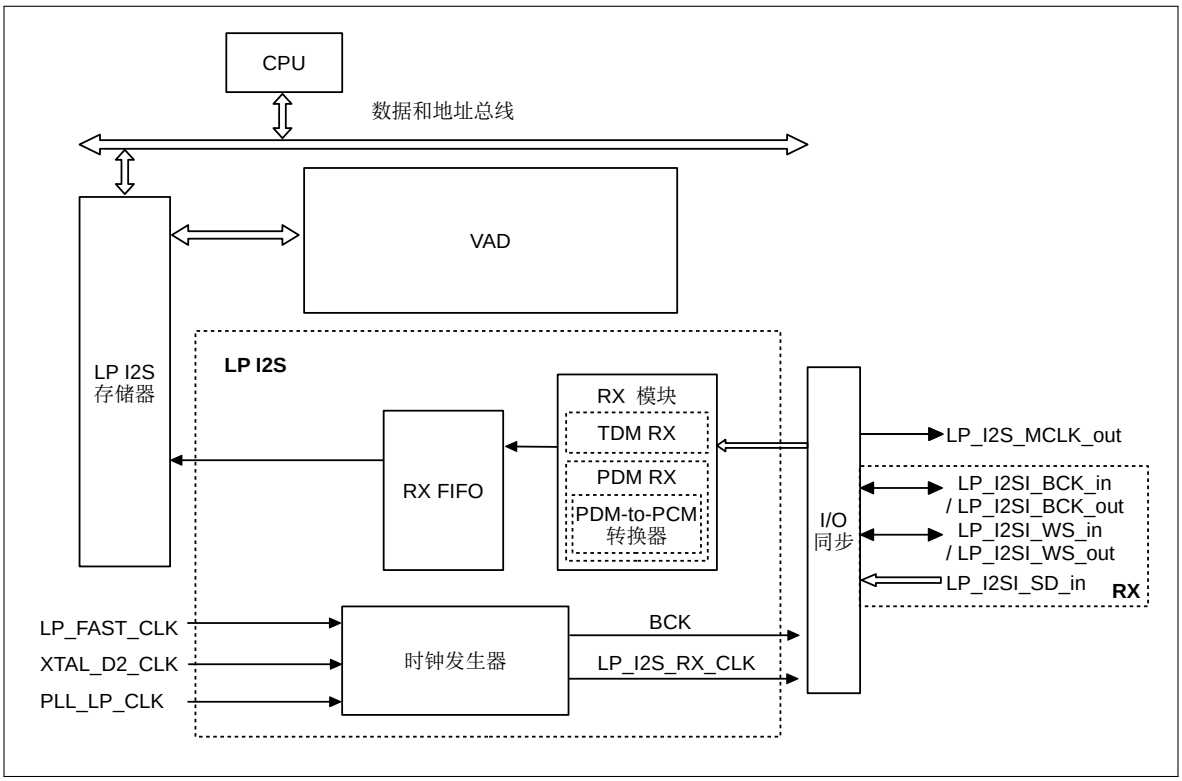


图 50.3-1. ESP32-P4 LP I2S 系统架构

RX 模块有一组三线接口，分别为串行时钟线 BCK、字选择线 WS 和串行数据线 SD。其中，RX 模块的 SD 线固定为输入接收。主机模式下，RX 模块的 BCK 和 WS 信号线为输出，从机模式下，RX 模块的 BCK 和 WS 信号线为输入。

图 50.3-1 中的右半部分展示了 LP I2S 模块的信号总线。RX 模块的信号命名规则为：LP_I2SA_B_C，例如 LP_I2SI_BCK_in。其中：

- “A” 固定为 “I”，表示信号输入至 RX 模块；
- “B” 表示信号功能，包括：
  - BCK
  - WS
  - SD

- “C” 表示该信号的方向，包括：
  - “in” 表示该信号输入 LP I2S 模块
  - “out” 表示该信号自 LP I2S 模块输出

LP I2S 各信号的具体描述见表 50.3-1。

表 50.3-1. 模块信号描述

信号 *	方向	功能
LP_I2SI_BCK_in	输入	LP I2S 从机模式下，输入 BCK 信号，用于 RX 模块
LP_I2SI_BCK_out	输出	LP I2S 主机模式下，输出 BCK 信号，用于 RX 模块
LP_I2SI_WS_in	输入	LP I2S 从机模式下，输入 WS 信号，用于 RX 模块
LP_I2SI_WS_out	输出	LP I2S 主机模式下，输出 WS 信号，用于 RX 模块
LP_I2SI_SD_in	输入	LP I2S RX 模块的串行输入数据线
LP_I2S_MCLK_out	输出	LP I2S 主机时钟输出

* LP I2S 的所有信号均需要经过 LP GPIO 交换矩阵映射到芯片的管脚。更多信息请参考章节 10 *GPIO 交换矩阵和 IO MUX*。

50.4 支持的音频协议

ESP32-P4 LP I2S 模块支持多种音频标准，包括 TDM Philips 标准、TDM MSB 对齐标准、TDM PCM 标准以及 PDM 标准。

用户可通过配置以下寄存器，选择所需的音频标准：

- LP_I2S_RX_TDM_EN
  - 0：禁用 TDM 模式
  - 1：选择 TDM 模式
- LP_I2S_RX_PDM_EN
  - 0：禁用 PDM 模式
  - 1：选择 PDM 模式
- LP_I2S_RX_MSB_SHIFT
  - 0：配置 WS 信号和 SD 信号同时开始变化，即选择 MSB 对齐标准
  - 1：配置 WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即选择 Philips 标准或 PCM 标准

说明：
不可将 LP_I2S_RX_TDM_EN 和 LP_I2S_RX_PDM_EN 同时配置为 1 或同时配置为 0，否则会导致 LP I2S 以既非 TDM 模式也非 PDM 模式的状态错误发送数据。

50.4.1 TDM Philips 标准模式

在 Philips 标准下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并在当前通道数据发送结束前一个 BCK 时钟周期开始变化。SD

信号线上首先传输音频数据的最高位。

与普通 Philips 标准相比，TDM Philips 标准支持的通道更多，见图 50.4-1。

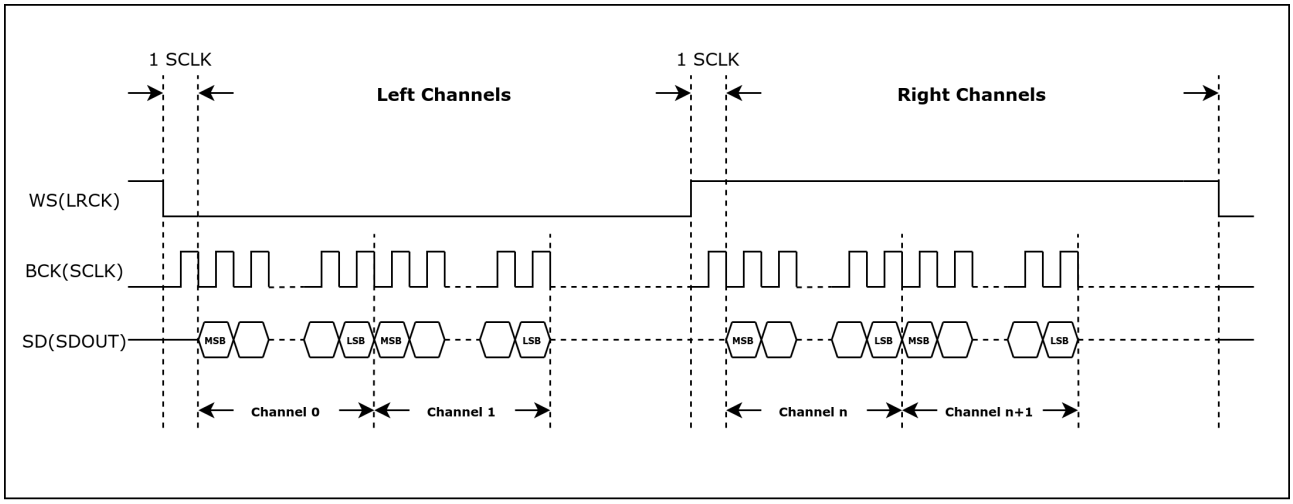


图 50.4-1. 时序图 – TDM Philips 标准

50.4.2 TDM MSB 对齐标准模式

MSB 对齐标准下，在 BCK 下降沿，WS 信号和 SD 信号同时变化。WS 持续到当前通道数据发送结束，SD 信号线上首先传输音频数据的最高位。

与普通 MSB 对齐标准相比，TDM MSB 对齐标准支持更多通道，见图 50.4-2。

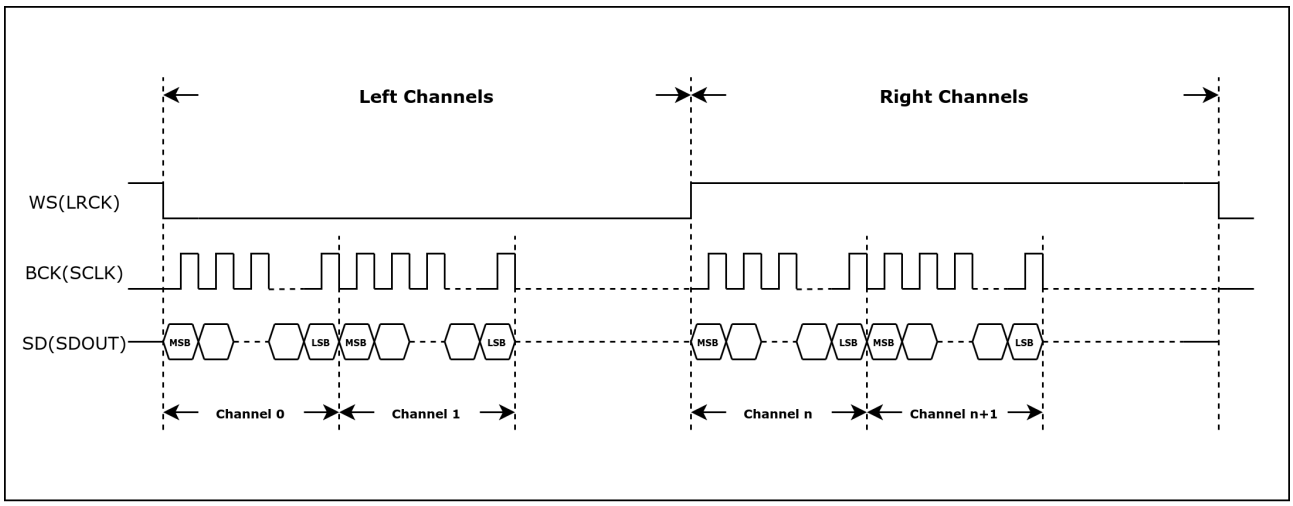


图 50.4-2. 时序图 – TDM MSB 对齐标准

50.4.3 TDM PCM 标准模式

在 PCM 标准的短帧同步模式下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并持续一个 BCK 时钟周期。SD 信号线上首先传输音频数据的最高位。

与普通 PCM 标准相比，TDM PCM 标准支持更多通道，见图 50.4-3 所示。

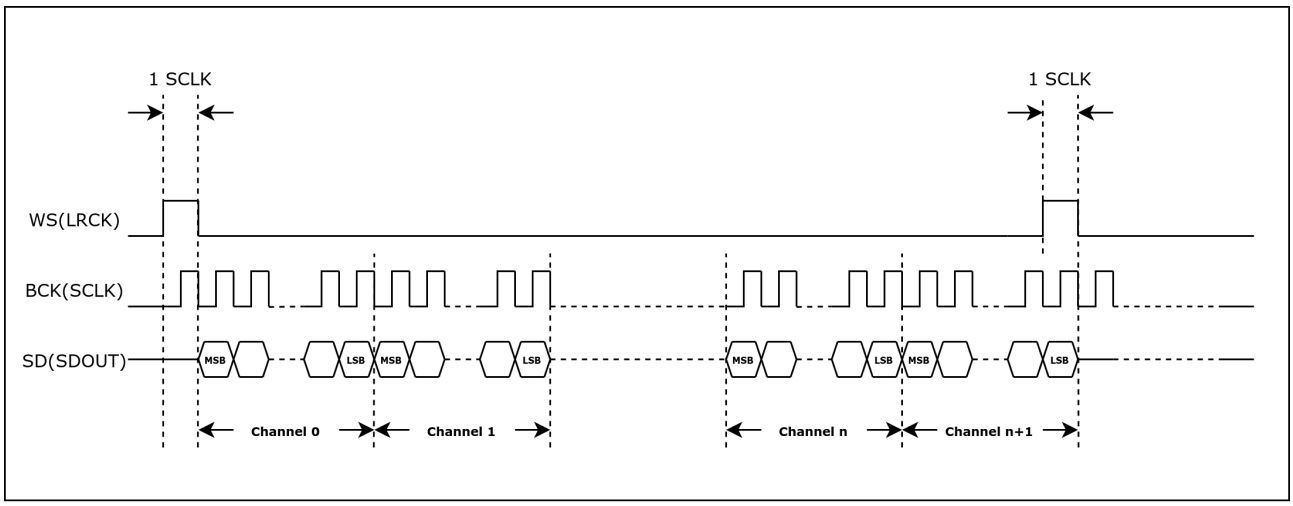


图 50.4-3. 时序图 – TDM PCM 标准

50.4.4 PDM 标准模式

如图 50.4-4 所示，在 PDM 标准下，WS 代表左/右声道，WS 与 SD 同时变化。WS 在数据发送过程中持续变化，WS 的高低对应两个声道。请参考图 50.4-4。

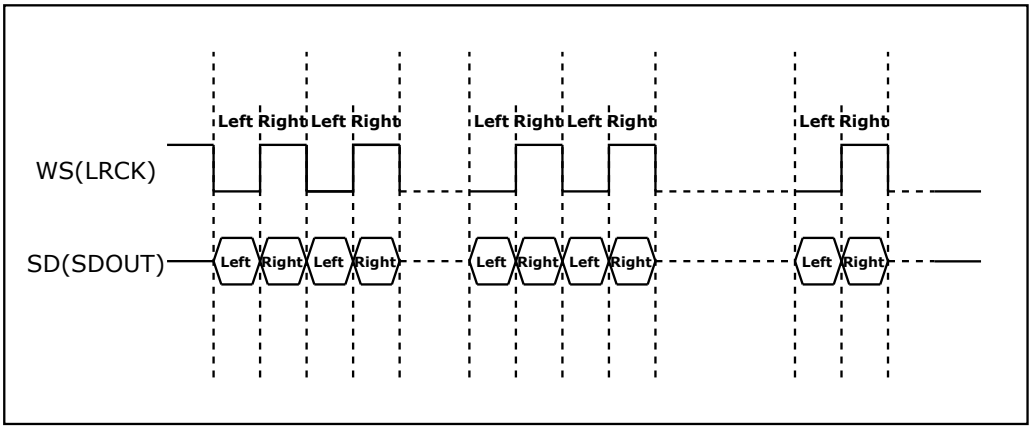


图 50.4-4. 时序图 – PDM 标准

50.5 RX 时钟

作为 LP I2S RX 模块的主时钟，LP_I2S_RX_CLK 可由以下时钟分频所得：

- 最高 40 MHz、可配置时钟源的 LP_FAST_CLK
- 20 MHz XTAL_D2_CLK
- 8 MHz PLL_LP_CLK

LP I2S RX 模块的串行时钟 BCK 再由 LP_I2S_RX_CLK 分频获得，如图 50.5-1 所示。

LPPERI_LP_I2S_RX_CLK_SEL 用于选择 LP I2S RX 的时钟源，LPPERI_CK_EN_LP_I2S_RX 用于使能或者关闭 LP I2S RX 模块的时钟源。

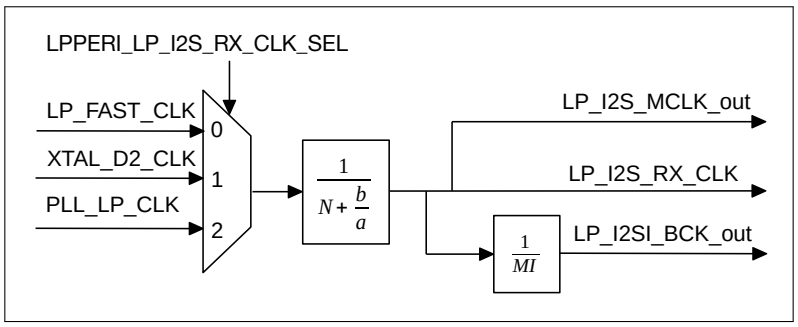


图 50.5-1. LP I2S 时钟分频器

LP_I2S_RX_CLK 的频率 $f_{LP_I2S_RX_CLK}$ 与分频器时钟源频率 $f_{LP_I2S_CLK_S}$ 间的关系如下：

$$f_{LP_I2S_RX_CLK} = \frac{f_{LP_I2S_CLK_S}}{N + \frac{b}{a}}$$

其中，2 <= N <= 256（N 为整数），N 对应寄存器域 LPPERI_LP_I2S_RX_CLKM_DIV_N 的值，具体为：

- LPPERI_LP_I2S_RX_CLKM_DIV_NUM = 0 时，N = 256；
- LPPERI_LP_I2S_RX_CLKM_DIV_NUM = 1 时，N = 2；
- LPPERI_LP_I2S_RX_CLKM_DIV_NUM 为其他值时，N = LPPERI_LP_I2S_RX_CLKM_DIV_NUM 的值。

分数部分分频系数 a、b 唯一对应系数 x、y、z、yn1。系数对应公式为：

- 当 b <= $\frac{a}{2}$ 时，yn1 = 0，x = $floor([\frac{a}{b}]) - 1$，y = a%b，z = b；
- 当 b > $\frac{a}{2}$ 时，yn1 = 1，x = $floor([\frac{a}{a-b}]) - 1$，y = a%(a - b)，z = a - b。

系数 x、y、z、yn1 通过 LPPERI_LP_I2S_RX_CLKM_DIV_X、LPPERI_LP_I2S_RX_CLKM_DIV_Y、LPPERI_LP_I2S_RX_CLKM_DIV_Z、LPPERI_LP_I2S_RX_CLKM_DIV_YN1 配置。

对于整数分频，LPPERI_LP_I2S_RX_CLKM_DIV_X 和 LPPERI_LP_I2S_RX_CLKM_DIV_Z 清零，LPPERI_LP_I2S_RX_CLKM_DIV_Y 置为 1。

说明：
使用小数分频功能可能会产生时钟抖动。

在主机接收模式下，LP I2S RX 模块的串行时钟 BCK 为 LP_I2SI_BCK_out 信号，由 LP_I2S_RX_CLK 分频获得。即：

$$f_{LP_I2SI_BCK_out} = \frac{f_{LP_I2S_RX_CLK}}{MI}$$

其中 MI 的值为：

$$MI = LP_I2S_RX_BCK_DIV_NUM + 1$$

说明：
注意，LP_I2S_RX_BCK_DIV_NUM 不可配置为 1。

当模块处于 LP I2S 从机模式时，为了保障采样的准确性，必须保证 $f_{LP_I2S_RX_CLK}$ >= 8 x f_{BCK}。另外模块可以输出 LP_I2S_MCLK_out 作为外部设备的主时钟。

由于 LP I2S 从机模式下，模块时钟频率必须大于等于 BCK 时钟频率的 8 倍，因此 I2S 支持的采样频率受到数据位宽和声道数的限制。例如，时钟源频率最高为 40 MHz，即模块时钟最高能配置为 20 MHz，能支持的 BCK 时钟频率最高为 5 MHz，因此传输双声道 16 位宽数据时，能够支持 78.125 kHz 以下的采样频率，例如 8 kHz、16 kHz、32 kHz、44.1 kHz、48 kHz。具体请参考章节 50.5 *RX 时钟*。

除了时钟频率限制外，LP I2S 使用 LP I2S 内部存储器而非 DMA 来存储数据。因此，用户需要及时从 LP I2S 内部存储器读取数据。读取速率可能会限制 LP I2S 支持的采样频率。详细信息，请参考小节 50.8.3 *内部存储器*。

50.6 复位

LP I2S 模块中各个单元以及 FIFO 可通过配置相关位进行复位：

- LP I2S RX 模块：可配置 LP_I2S_RX_RESET 进行复位；
- LP I2S RX FIFO：可配置 LP_I2S_RX_FIFO_RESET 进行复位。

说明：
在模块和 FIFO 复位之前，需要先配置 LP I2S 模块时钟。

50.7 主/从机接收模式

在 RX 模式下，LP I2S 可作为主机或从机接收数据。

- 主机接收模式
  - 置位 LP_I2S_RX_START 启动一次接收操作。RX 模块会一直输出时钟信号，并对输入数据进行采样。
  - 配置 LP_I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    * 0：只有 LP_I2S_RX_START 被清零，RX 模块才会暂停接收。
    * 1：RX 模块会在 LP_I2S_RX_START 被清零，或者接收字节数大于等于 LP_I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收。暂停接收后，当 LP_I2S_RX_START 未被清零时，可通过置位 LP_I2S_RX_RESET 来重新启动接收。
  - 清零 LP_I2S_RX_START，RX 模块停止接收数据。
- 从机接收模式
  - 置位 LP_I2S_RX_START。等待主机 BCK 时钟，来启动接收操作。
  - 配置 LP_I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    * 0：只有 LP_I2S_RX_START 被清零，RX 模块才会暂停接收。
    * 1：RX 模块会在 LP_I2S_RX_START 被清零，或者接收字节数大于等于 LP_I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收。暂停接收后，当 LP_I2S_RX_START 未被清零时，可通过置位 LP_I2S_RX_RESET 来重新启动接收。
  - 清零 LP_I2S_RX_START，RX 模块停止接收数据。

50.8 接收数据

LP I2S 接收数据时，从外设接口读取数据，经过通道模式控制和数据格式控制，输入数据至 LP I2S 存储器。

50.8.1 通道模式控制

ESP32-P4 LP I2S 支持 TDM 和 PDM 两种输入模式。置位 LP_I2S_RX_TDM_EN 使能 TDM 输入模式，置位 LP_I2S_RX_PDM_EN 使能 PDM 输入模式。

> **说明：**
> LP_I2S_RX_TDM_EN 和 LP_I2S_RX_PDM_EN 不能同时置位或同时清零。

50.8.1.1 TDM 输入模式

在 TDM 输入模式下，LP I2S 最多支持 2 个接收通道。接收通道数由 LP_I2S_RX_TDM_TOT_CHAN_NUM 控制。例如，配置 LP_I2S_RX_TDM_TOT_CHAN_NUM 为 1，则通道 0 ~ 1 接收数据。

在接收数据的通道中，如果其对应的 LP_I2S_RX_TDM_PDM_CHANn_EN 为：

- 0：则该通道数据无效，不会存入 RX FIFO；
- 1：则该通道数据有效，会被存入 RX FIFO。

当 LP I2S 处于主机 TDM 输入模式下，WS 信号由 LP_I2S_RX_WS_IDLE_POL 和 LP_I2S_RX_TDM_WS_WIDTH 控制：

- LP_I2S_RX_WS_IDLE_POL 的值为 WS 信号的默认电平；
- LP_I2S_RX_TDM_WS_WIDTH 的值为在接收所有通道的过程中，WS 为默认电平的周期数。

另外，LP_I2S_RX_HALF_SAMPLE_BITS 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

50.8.1.2 PDM 输入模式

在 PDM 输入模式下，LP I2S 将通道中的串行数据转换为待输入数据。LP I2S 可以接收原始 PDM 数据，或者将 PDM 数据转换为 PCM 数据后接收。

当 LP I2S 处于主机 PDM 输入模式下，WS 信号的默认电平由 LP_I2S_RX_WS_IDLE_POL 控制，WS 信号频率为 BCK 信号频率的一半，请参考 50.5 小节配置 BCK 信号的方式配置 WS 信号。注意，在 PDM 输入模式下，请配置 LP_I2S_RX_HALF_SAMPLE_BITS 的值与 LP_I2S_RX_BITS_MOD 的值相同。

在 LP I2S 启用 PDM-to-PCM 转换器时，可以将接收的 PDM 数据转为 PCM 数据，再进行数据控制。配置 LP_I2S_RX_PDM2PCM_EN 启用该转换器。PDM-to-PCM 转换器的寄存器配置如下：

- 配置采样频率和下采样率。
  LP I2S 使用 PDM-to-PCM 转换器接收数据时，PDM 时钟频率与主从机模式的关系如下：
  - 主机模式：PDM 时钟频率即为 BCK 时钟频率。
  - 从机模式：PDM 时钟频率由外部提供。

  采样频率 ($f_{Sampling}$) 与 PDM 时钟频率的关系如下：

$$f_{Sampling} = \frac{f_{PDM}}{DSR}$$

其中，下采样率 DSR 和 LP_I2S_RX_PDM_SINC_DSR_16_EN 的关系为：

$$DSR = LP_I2S_RX_PDM_SINC_DSR_16_EN \times 64$$

请根据需要的主从机模式、采样频率以及下采样率进行寄存器配置。

- 配置有效通道。
  启用 PDM-to-PCM 转换器时，最多支持 2 个通道的输入信号，寄存器配置以及对应通道如下表：

表 50.8-1. PDM-to-PCM 数据输入

输入数据信号	通道	使能寄存器
LP_I2SI_SD_in	左声道	LP_I2S_RX_TDM_PDM_CHAN0_EN
	右声道	LP_I2S_RX_TDM_PDM_CHAN1_EN

50.8.2 数据格式控制

数据格式控制分为两个阶段：

- 第一阶段：从串行数据流输入转换为待输入数据，并存入 RX FIFO；
- 第二阶段：将待输入数据从 RX FIFO 中读出，并进行输入数据模式转换。

50.8.2.1 通道数据比特顺序

ESP32-P4 LP I2S 通道数据将从高位到低位依次存入待输入数据，并由 LP_I2S_RX_BIT_ORDER 控制是否调换比特序，当该值为：

- 0：不调换待输入数据的比特序；
- 1：调换待输入数据的比特序。

至此，数据格式控制的第一阶段完成。

50.8.2.2 通道储存数据位宽

LP_I2S_RX_BITS_MOD 决定了每个通道的储存数据位宽。对于 ESP32-P4 LP I2S，LP_I2S_RX_BITS_MOD 的值固定为 15，对应的储存数据位宽为 16。

50.8.2.3 通道接收数据位宽

LP_I2S_RX_TDM_CHAN_BITS 决定了每个通道接收数据的位宽。

- 如果通道储存数据位宽小于接收数据宽度，则只有储存数据位宽内的数据位会被保存到内存中。此时，配置 LP_I2S_RX_LEFT_ALIGN 为：
  - 0：储存数据位于接收数据低位；
  - 1：储存数据位于接收数据高位。
- 当每个通道接收数据的位宽小于储存数据位宽时，会将接收数据高位补 0，作为储存数据。

50.8.2.4 通道储存数据字节序

通道接收数据经过截取/补全后，成为了待储存数据，LP_I2S_RX_BIG_ENDIAN 用于控制待储存数据的字节序。下表描述了该寄存器对待储存数据的控制（通道储存数据位宽固定为 16）。

表 50.8-2. 通道储存数据字节序控制

原始数据	控制后数据	LP_I2S_RX_BIG_ENDIAN
{B1, B0}	{B1, B0}	0
	{B0, B1}	1

至此，数据格式控制部分全部完成，数据将存入 LP I2S 内部存储器。

50.8.3 内部存储器

所有经过数据格式控制的数据，都将存入 LP I2S 内部存储器。LP I2S 内部存储器为 16 位宽的循环 buffer，支持 LP I2S RX 模块写入和系统读取。LP I2S 内部存储器拥有一对由硬件维护的读写指针，写入和读取的操作由读写指针控制。

1. 当 LP I2S 写入数据时：
   - 存储器已满时：覆写最老的数据，写指针加一，读指针也加一（此时读指针的值为写指针加一）
   - 存储器未满时：写入数据，写指针加一，读指针不变
2. 当系统读取 LP I2S 存储器时：
   - 存储器已空时：读写指针均不变（此时读指针的值等于写指针）
   - 存储器非空时：写指针不变，读指针加一

说明：

- 写指针和读指针的加一操作均为循环加法，即当读写指针到达存储器空间边界时，加一操作使其归零。
- 在 LP I2S 内部存储器的结构下，系统永远读取存储器中最老的数据。

50.9 中断

ESP32-P4 的 LP I2S 可以生成中断信号 LP_I2S_INTR 并发送给 中断矩阵。该中断信号由以下 LP I2S 的内部中断源生成。

- LP_I2S_RX_HUNG_INT：当接收数据超时，触发此中断。例如，LP I2S 配置为从机接收模式，但主机长时间未发送数据，则将触发该中断。超时配置见寄存器 LP_I2S_LC_HUNG_CONF_REG。
- LP_I2S_RX_DONE_INT：当接收数据完成，触发此中断。
- LP_I2S_RX_FIFOMEM_UDF_INT：当 LP I2S 存储器读空时，触发此中断。
- LP_I2S_RX_MEM_THRESHOLD_INT：当 LP I2S 存储器中的数据量大于 LP_I2S_RX_MEM_THRESHOLD 中配置的值时，触发此中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 50.10 寄存器列表。

50.10 寄存器列表

本小节的所有地址均为相对于 LP I2S 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

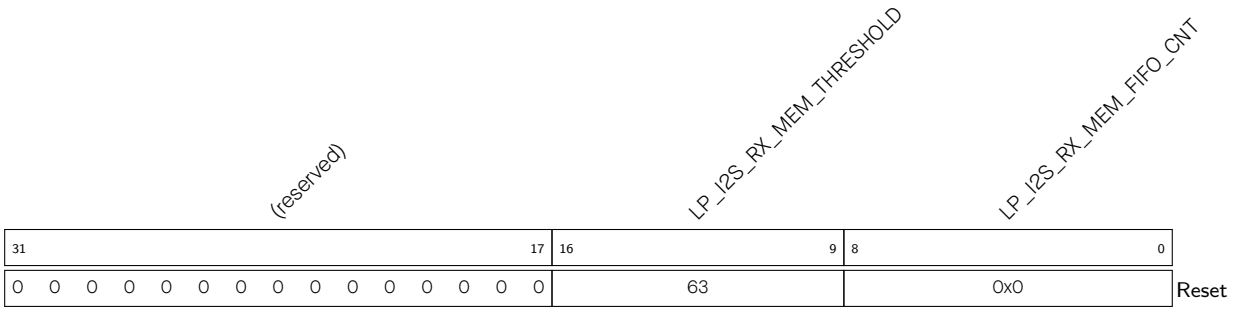
名称	描述	地址	访问
RX 控制和配置寄存器			
LP_I2S_RX_MEM_CONF_REG	LP I2S 存储器配置寄存器	0x0008	varies
LP_I2S_RX_CONF_REG	LP I2S RX 配置寄存器	0x0020	varies
LP_I2S_RX_CONF1_REG	LP I2S RX 配置寄存器 1	0x0028	R/W
LP_I2S_RX_TDM_CTRL_REG	LP I2S RX TDM 模式配置寄存器	0x0050	R/W
LP_I2S_RXEOF_NUM_REG	LP I2S RX 数据大小控制寄存器	0x0064	R/W
LP_I2S_RX_PDM_CONF_REG	LP I2S RX PDM 模式配置寄存器	0x0070	R/W
中断寄存器			
LP_I2S_INT_RAW_REG	LP I2S 原始中断寄存器	0x000C	RO/WTC/SS
LP_I2S_INT_ST_REG	LP I2S 中断状态寄存器	0x0010	RO
LP_I2S_INT_ENA_REG	LP I2S 中断使能寄存器	0x0014	R/W
LP_I2S_INT_CLR_REG	LP I2S 中断清除寄存器	0x0018	WT
RX 时钟和时序寄存器			
LP_I2S_RX_TIMING_REG	LP I2S RX 时序控制寄存器	0x0058	R/W
控制和配置寄存器			
LP_I2S_LC_HUNG_CONF_REG	LP I2S 超时配置寄存器	0x0060	R/W
LP_I2S_CONF_SIGLE_DATA_REG	LP I2S 信号数据寄存器	0x0068	R/W
时钟寄存器			
LP_I2S_CLK_GATE_REG	时钟门控寄存器	0x00F8	R/W
版本控制寄存器			
LP_I2S_DATE_REG	版本控制寄存器	0x00FC	R/W

50.11 寄存器

本小节的所有地址均为相对于 LP I2S 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

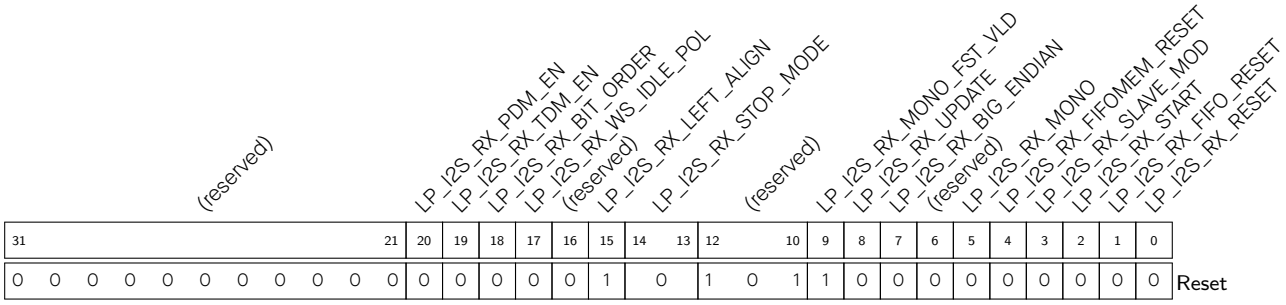
Register 50.1. LP_I2S_RX_MEM_CONF_REG (0x0008)



LP_I2S_RX_MEM_FIFO_CNT 配置 RX 模块存储器中的数据大小。(RO)

LP_I2S_RX_MEM_THRESHOLD 配置 RX 模块存储器阈值。当存储器中的数据超过（不包括等于）该阈值时，LP I2S RX 存储器会触发中断。(R/W)

Register 50.2. LP_I2S_RX_CONF_REG (0x0020)



LP_I2S_RX_RESET 配置是否复位 RX 模块。

0：无效

1：复位

(WT)

LP_I2S_RX_FIFO_RESET 配置是否复位 RX FIFO。

0：无效

1：复位

(WT)

LP_I2S_RX_START 配置是否开始接收数据。

0：无效

1：开始

(R/W)

见下页...

Register 50.2. LP_I2S_RX_CONF_REG (0x0020)

接上页...

LP_I2S_RX_SLAVE_MOD 配置是否使能从机接收模式。

0：使能主机模式

1：使能从机模式

(R/W)

LP_I2S_RX_FIFOMEM_RESET 配置是否复位 RX 同步 FIFO 存储器。

0：无效

1：开始

(WT)

LP_I2S_RX_MONO 配置是否使能接收单元的单声道模式。

0：禁用

1：使能

(R/W)

LP_I2S_RX_BIG_ENDIAN 配置 LP I2S RX 字节序。

0：低字节数据写入低位地址

1：低字节数据写入高位地址

(R/W)

LP_I2S_RX_UPDATE 配置是否将 I2S RX 寄存器从 APB 时钟域同步到 I2S RX 时钟域。寄存器完成更新后，该位将由硬件清除。

0：无效

1：同步

(R/W/SC)

LP_I2S_RX_MONO_FST_VLD 配置 LP I2S RX 单声道模式下的有效数据通道。

0：第二个通道数据有效

1：第一个通道数据有效

(R/W)

LP_I2S_RX_STOP_MODE 配置 LP I2S RX 何时暂停接收。

0：只有 LP_I2S_RX_START 被清除时暂停接收

1：LP_I2S_RX_START 被清除，或者接收字节数大于 LP_I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收

其他值：无效

(R/W)

LP_I2S_RX_LEFT_ALIGN 配置 LP I2S RX 对齐模式。

0：右对齐模式

1：左对齐模式

(R/W)

见下页...

Register 50.2. LP_I2S_RX_CONF_REG (0x0020)

接上页...

LP_I2S_RX_WS_IDLE_POL 配置 WS 与接收数据通道的关系。

0：WS 为 0 时，接收左通道数据；WS 为 1 时，接收右通道数据

1：WS 为 1 时，接收左通道数据；WS 为 0 时，接收右通道数据

(R/W)

LP_I2S_RX_BIT_ORDER 配置是否调换 LP I2S RX 待接收数据的位顺序。

0：不调换

1：调换

(R/W)

LP_I2S_RX_TDM_EN 配置是否使能 LP I2S TDM RX 模式。

0：禁用

1：使能

(R/W)

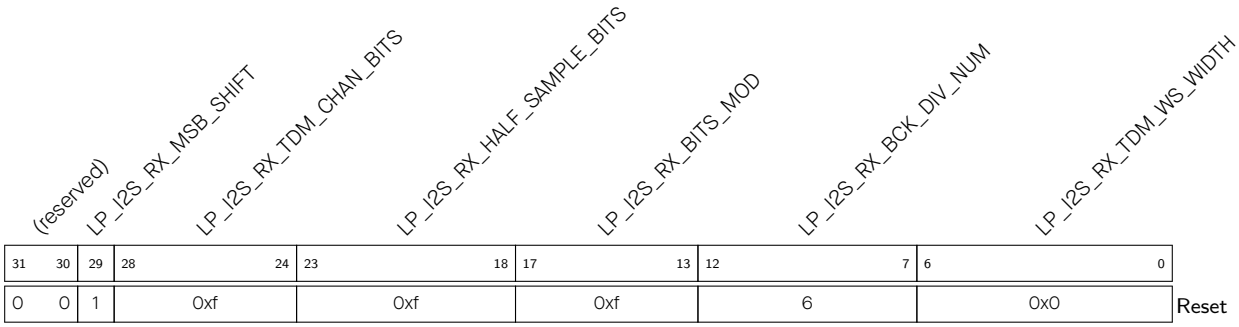
LP_I2S_RX_PDM_EN 配置是否使能 LP I2S PDM RX 模式。

0：禁用

1：使能

(R/W)

Register 50.3. LP_I2S_RX_CONF1_REG (0x0028)



LP_I2S_RX_TDM_WS_WIDTH 配置 LP_I2SI_WS_out（WS 默认电平）空闲时在 TDM 模式下的时长。LP_I2SI_WS_out 空闲时在 TDM 模式下的时长 = (LP_I2S_RX_TDM_WS_WIDTH[6:0] +1) x T_BCK。(R/W)

LP_I2S_RX_BCK_DIV_NUM 配置在 RX 模式下 BCK 时钟的分频系数。注意，该值不可配置为 1。(R/W)

LP_I2S_RX_BITS_MOD 配置在 RX 模式下接收通道的有效数据位长度。由于 LP I2S 仅支持 16 位模式，因此该字段必须配置为 15。配置为其他值会导致 LP I2S 无法正常工作。(R/W)

LP_I2S_RX_HALF_SAMPLE_BITS 配置 LP I2S RX 单次采样比特数。一个 WS 信号中 BCK 持续的周期 = LP_I2S_RX_HALF_SAMPLE_BITS x 2。(R/W)

LP_I2S_RX_TDM_CHAN_BITS 配置在 TDM RX 模式下每个通道的 RX 数据位长。RX 数据位长 = LP_I2S_RX_TDM_CHAN_BITS + 1。(R/W)

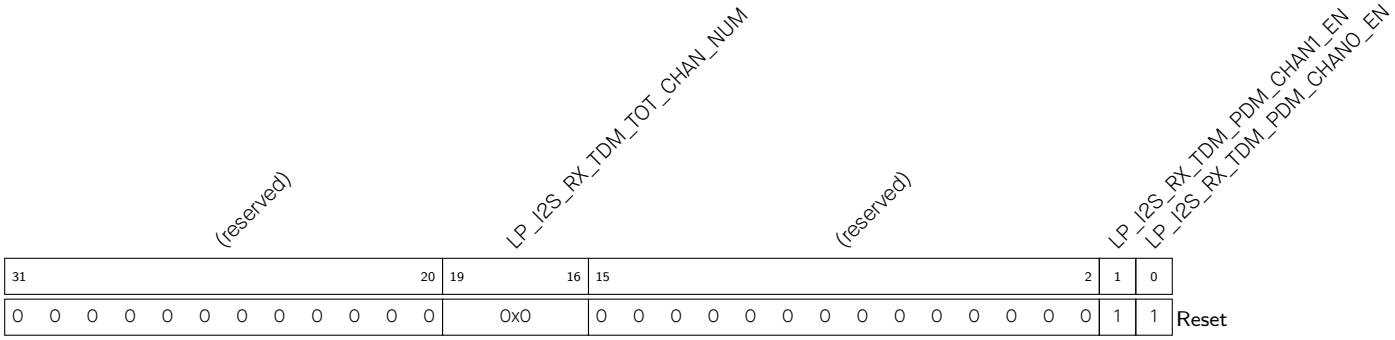
LP_I2S_RX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0：上升沿对齐

1：相隔一个周期

(R/W)

Register 50.4. LP_I2S_RX_TDM_CTRL_REG (0x0050)



LP_I2S_RX_TDM_PDM_CHAN0_EN 配置是否使能 LP I2S TDM 或 PDM RX 通道 0 的有效输入数据。

0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

LP_I2S_RX_TDM_PDM_CHAN1_EN 配置是否使能 LP I2S TDM 或 PDM RX 通道 1 的有效输入数据。

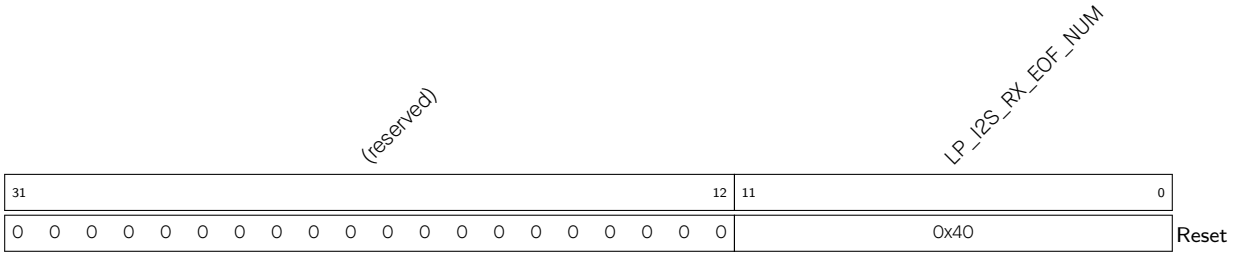
0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

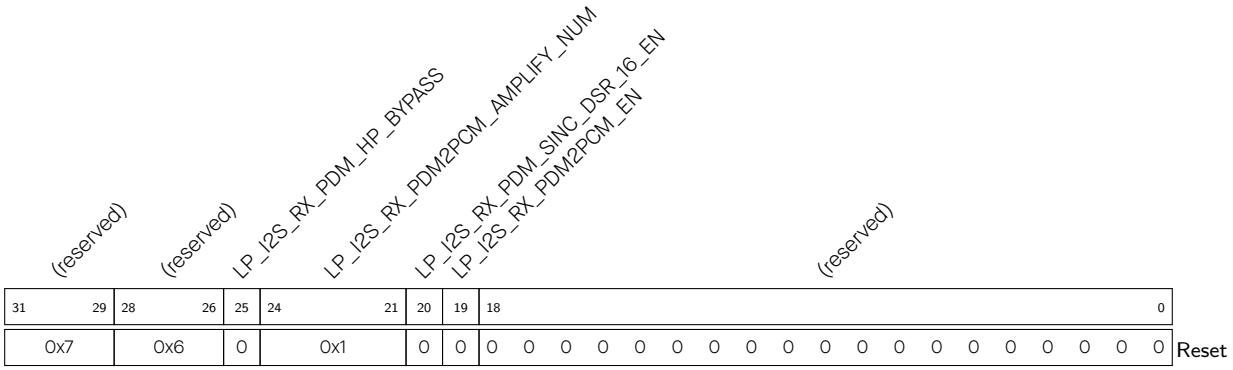
LP_I2S_RX_TDM_TOT_CHAN_NUM 配置在 LP I2S TDM RX 模式下使用的通道总数。使用的通道总数 = LP_I2S_RX_TDM_TOT_CHAN_NUM + 1。(R/W)

Register 50.5. LP_I2S_RXEOF_NUM_REG (0x0064)



LP_I2S_RX_EOF_NUM 配置接收数据的长度。接收数据长度 = (LP_I2S_RX_BITS_MOD[4:0] + 1) x (LP_I2S_RX_EOF_NUM[11:0] + 1)。该值在 LP_I2S_RX_STOP_MODE 为 1 时有效。(R/W)

Register 50.6. LP_I2S_RX_PDM_CONF_REG (0x0070)



LP_I2S_RX_PDM2PCM_EN 配置是否在 RX 模式下使能 PDM-to-PCM 转换器。

0：禁用

1：使能

(R/W)

LP_I2S_RX_PDM_SINC_DSR_16_EN 配置 PDM RX 滤波器组 1 的向下采样率。

0：64

1：128

(R/W)

LP_I2S_RX_PDM2PCM_AMPLIFY_NUM 配置 PDM-to-PCM RX 放大系数。PCM 数据在输出前将乘以该值。(R/W)

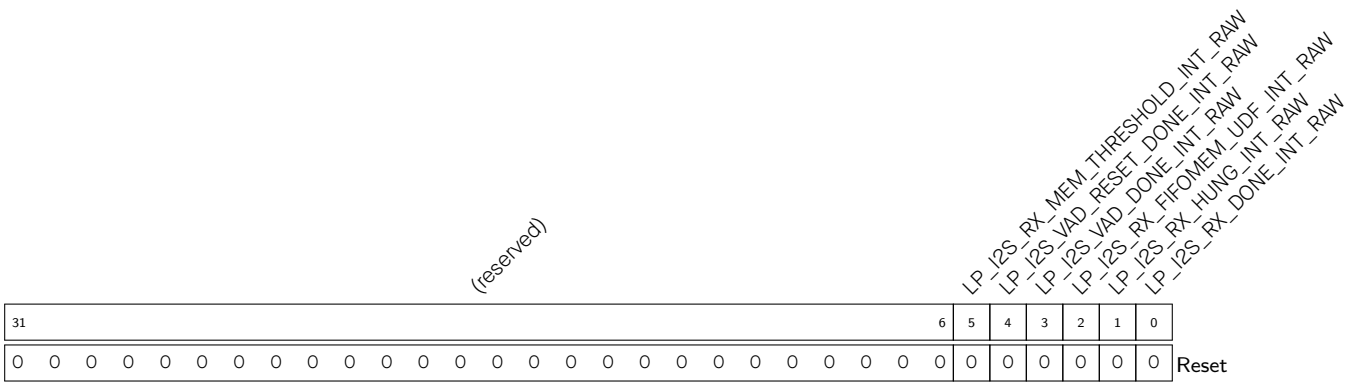
LP_I2S_RX_PDM_HP_BYPASS 配置 PDM-to-PCM RX 是否绕过 HP 滤波器。

0：不绕过

1：绕过

(R/W)

Register 50.7. LP_I2S_INT_RAW_REG (0x000C)



LP_I2S_RX_DONE_INT_RAW LP_I2S_RX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

LP_I2S_RX_HUNG_INT_RAW LP_I2S_RX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

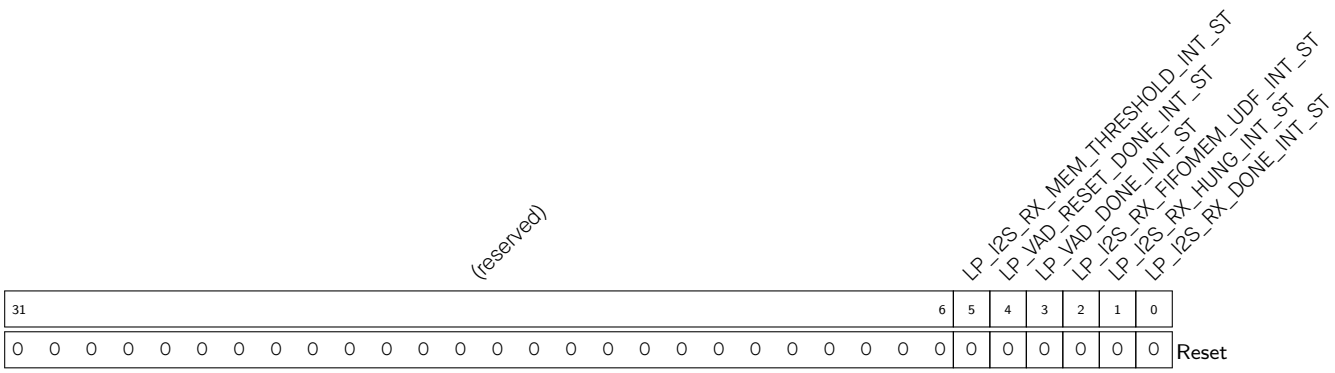
LP_I2S_RX_FIFOMEM_UDF_INT_RAW LP_I2S_RX_FIFOMEM_UDF_INT 中断的原始中断状态。(R/SS/WTC)

LP_I2S_VAD_DONE_INT_RAW LP_I2S_VAD_DONE_INT 中断的原始中断状态。(R/SS/WTC)

LP_I2S_VAD_RESET_DONE_INT_RAW LP_I2S_VAD_RESET_DONE_INT 中断的原始中断状态。(R/SS/WTC)

LP_I2S_RX_MEM_THRESHOLD_INT_RAW LP_I2S_RX_MEM_THRESHOLD_INT 中断的原始中断状态。(R/SS/WTC)

Register 50.8. LP_I2S_INT_ST_REG (0x0010)



LP_I2S_RX_DONE_INT_ST LP_I2S_RX_DONE_INT 中断的屏蔽中断状态。(RO)

LP_I2S_RX_HUNG_INT_ST LP_I2S_RX_DONE_INT 中断的屏蔽中断状态。(RO)

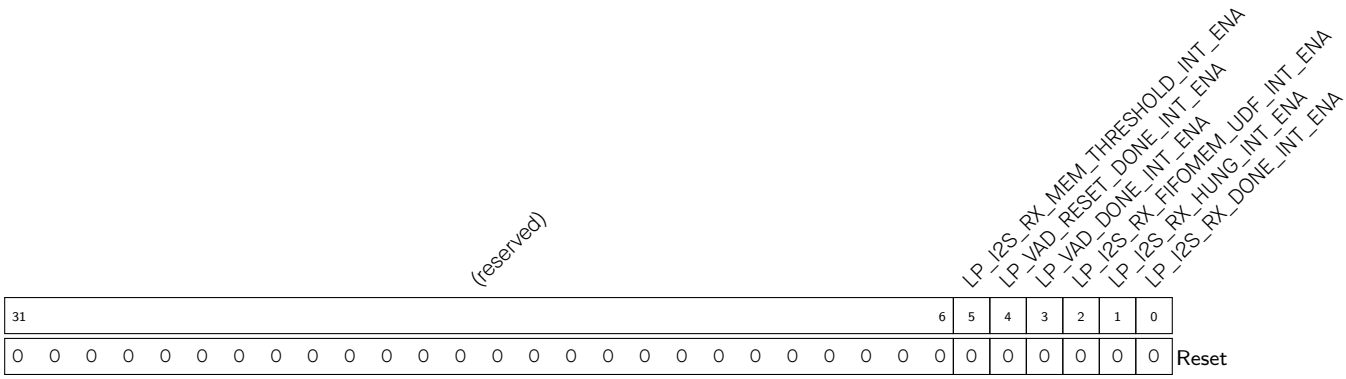
LP_I2S_RX_FIFOMEM_UDF_INT_ST LP_I2S_RX_FIFOMEM_UDF_INT 中断的屏蔽中断状态。(RO)

LP_VAD_DONE_INT_ST LP_I2S_VAD_DONE_INT 中断的屏蔽中断状态。(RO)

LP_VAD_RESET_DONE_INT_ST LP_I2S_VAD_RESET_DONE_INT 中断的屏蔽中断状态。(RO)

LP_I2S_RX_MEM_THRESHOLD_INT_ST LP_I2S_RX_MEM_THRESHOLD_INT 中断的屏蔽中断状态。(RO)

Register 50.9. LP_I2S_INT_ENA_REG (0x0014)



LP_I2S_RX_DONE_INT_ENA 写 1 使能 LP_I2S_RX_DONE_INT。(R/W)

LP_I2S_RX_HUNG_INT_ENA 写 1 使能 LP_I2S_RX_DONE_INT。(R/W)

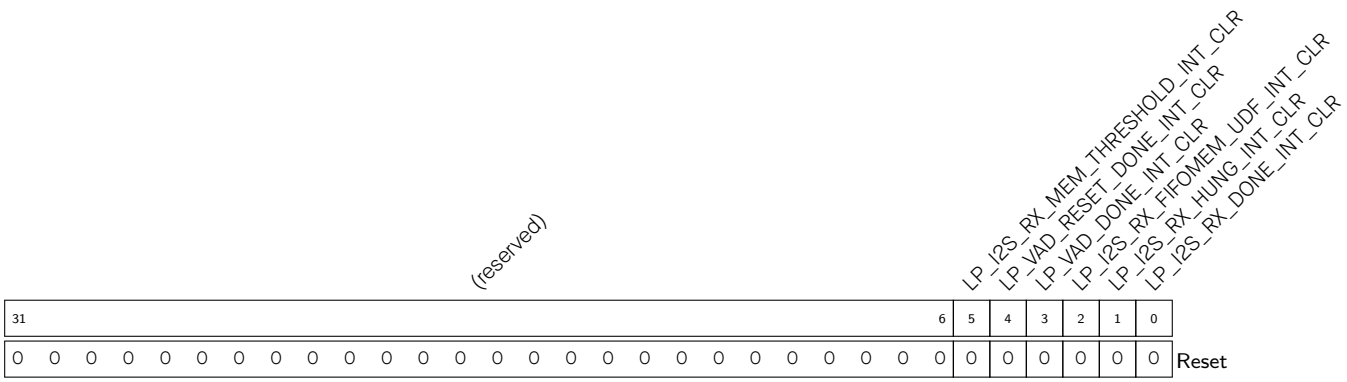
LP_I2S_RX_FIFOMEM_UDF_INT_ENA 写 1 使能 LP_I2S_RX_FIFOMEM_UDF_INT。(R/W)

LP_VAD_DONE_INT_ENA 写 1 使能 LP_I2S_VAD_DONE_INT。(R/W)

LP_VAD_RESET_DONE_INT_ENA 写 1 使能 LP_I2S_VAD_RESET_DONE_INT。(R/W)

LP_I2S_RX_MEM_THRESHOLD_INT_ENA 写 1 使能 LP_I2S_RX_MEM_THRESHOLD_INT。(R/W)

Register 50.10. LP_I2S_INT_CLR_REG (0x0018)



LP_I2S_RX_DONE_INT_CLR 写 1 清除 LP_I2S_RX_DONE_INT。(WT)

LP_I2S_RX_HUNG_INT_CLR 写 1 清除 LP_I2S_RX_DONE_INT。(WT)

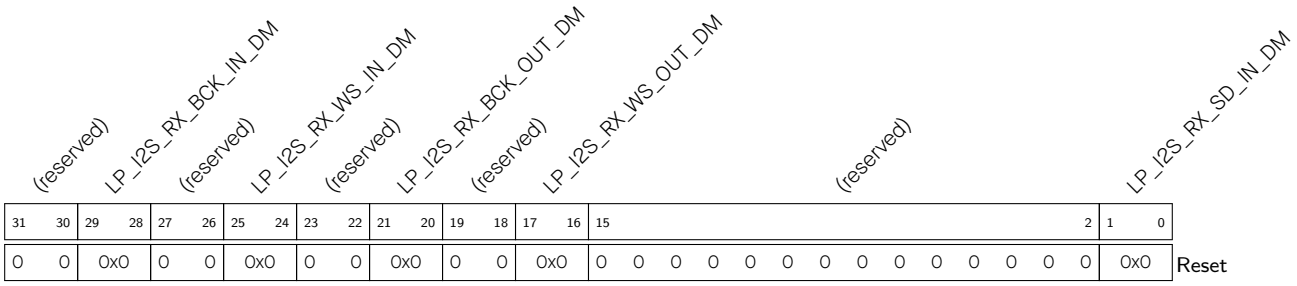
LP_I2S_RX_FIFOMEM_UDF_INT_CLR 写 1 清除 LP_I2S_RX_FIFOMEM_UDF_INT。(WT)

LP_VAD_DONE_INT_CLR 写 1 清除 LP_I2S_VAD_DONE_INT。(WT)

LP_VAD_RESET_DONE_INT_CLR 写 1 清除 LP_I2S_VAD_RESET_DONE_INT。(WT)

LP_I2S_RX_MEM_THRESHOLD_INT_CLR 写 1 清除 LP_I2S_RX_MEM_THRESHOLD_INT。(WT)

Register 50.11. LP_I2S_RX_TIMING_REG (0x0058)



LP_I2S_RX_SD_IN_DM 配置 LP I2S RX SD 输入信号的延迟模式。

0：旁路

1：上升沿延迟

2：下降沿延迟

3：保留

(R/W)

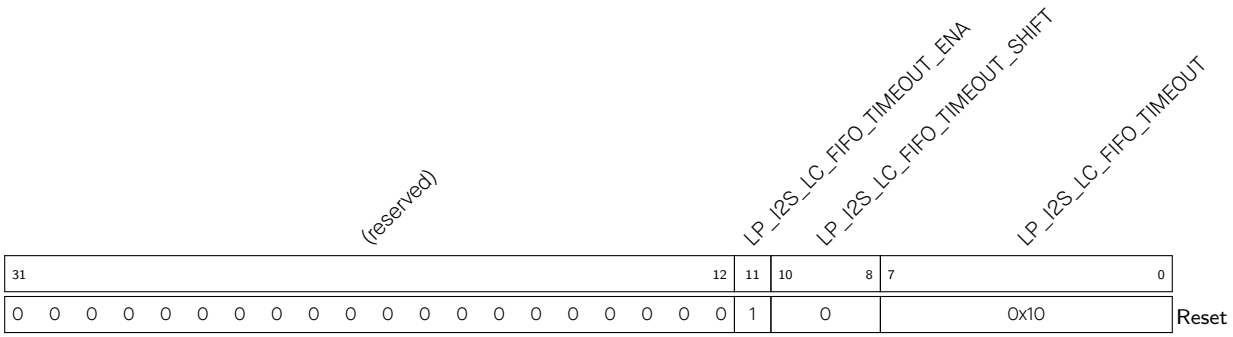
LP_I2S_RX_WS_OUT_DM 配置 LP I2S RX WS 输出信号的延迟模式。具体配置值请参考 LP_I2S_RX_SD_IN_DM。(R/W)

LP_I2S_RX_BCK_OUT_DM 配置 LP I2S RX BCK 输出信号的延迟模式。具体配置值请参考 LP_I2S_RX_SD_IN_DM。(R/W)

LP_I2S_RX_WS_IN_DM 配置 LP I2S RX WS 输入信号的延迟模式。具体配置值请参考 LP_I2S_RX_SD_IN_DM。(R/W)

LP_I2S_RX_BCK_IN_DM 配置 LP I2S RX BCK 输入信号的延迟模式。具体配置值请参考 LP_I2S_RX_SD_IN_DM。(R/W)

Register 50.12. LP_I2S_LC_HUNG_CONF_REG (0x0060)



LP_I2S_LC_FIFO_TIMEOUT 配置 FIFO 的超时阈值。FIFO Hung 计数器在 LP I2S 处于错误状态，并且每次分频滴答计数器达到滴答计数器的阈值时会加一。FIFO Hung 计数器等于 FIFO 的超时阈值时，将触发 LP_I2S_RX_HUNG_INT 中断。(R/W)

LP_I2S_LC_FIFO_TIMEOUT_SHIFT 配置分频滴答计数器的阈值。分频滴答计数器会在 LP I2S 处于错误状态，并且每个系统 (APB) 时钟上升沿加一。计数器值大于等于 $88000/2^{LP_I2S_LC_FIFO_TIMEOUT_SHIFT}$ 时，复位滴答计数器。(R/W)

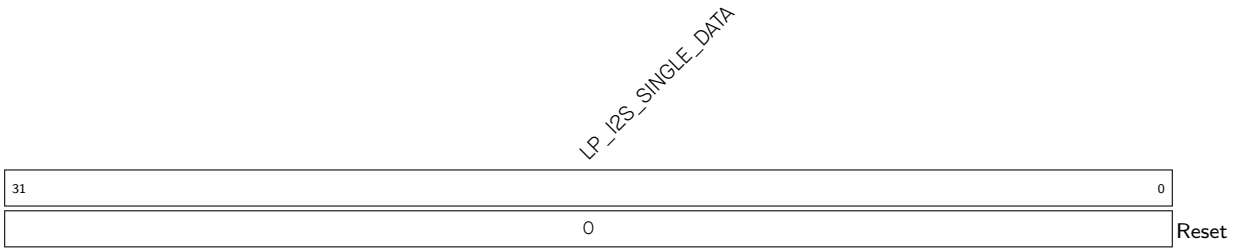
LP_I2S_LC_FIFO_TIMEOUT_ENA 配置是否使能 FIFO 超时。

0：禁用

1：使能

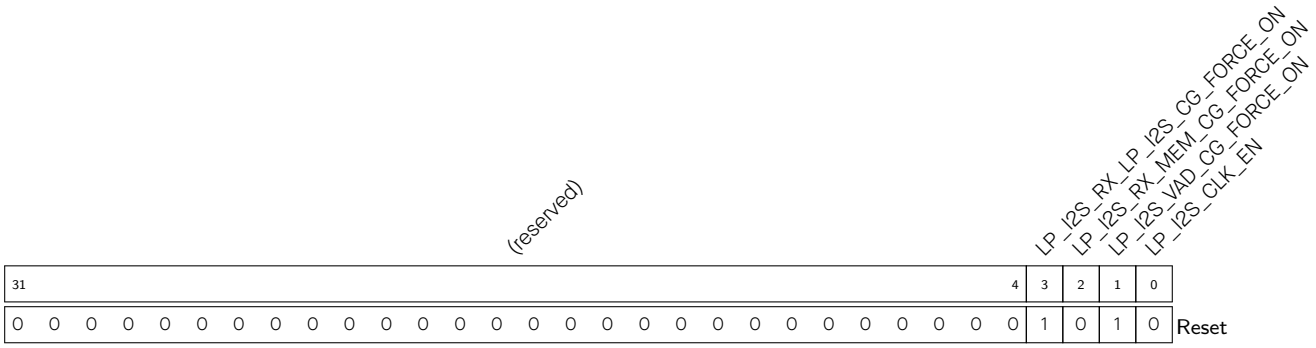
(R/W)

Register 50.13. LP_I2S_CONF_SIGLE_DATA_REG (0x0068)



LP_I2S_SINGLE_DATA 配置用于发送的通道常量数据。(R/W)

Register 50.14. LP_I2S_CLK_GATE_REG (0x00F8)



LP_I2S_CLK_EN 配置是否启用时钟门控。

0：禁用

1：启用

(R/W)

LP_I2S_VAD_CG_FORCE_ON 配置是否强制开启 VAD 时钟门控。

0：无效

1：强制开启

(R/W)

LP_I2S_RX_MEM_CG_FORCE_ON 配置是否强制开启 LP I2S RX 存储器时钟门控。

0：无效

1：强制开启

(R/W)

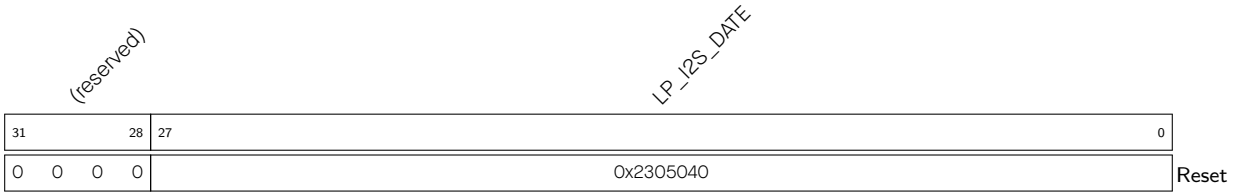
LP_I2S_RX_LP_I2S_CG_FORCE_ON 置是否强制开启 LP I2S RX 寄存器时钟门控。

0：无效

1：强制开启

(R/W)

Register 50.15. LP_I2S_DATE_REG (0x00FC)



LP_I2S_DATE 版本控制寄存器。(R/W)

第 51 章

脉冲计数控制器 (PCNT)

脉冲计数控制器 (Pulse Count Controller, PCNT) 用于记录输入脉冲的个数。

51.1 概述

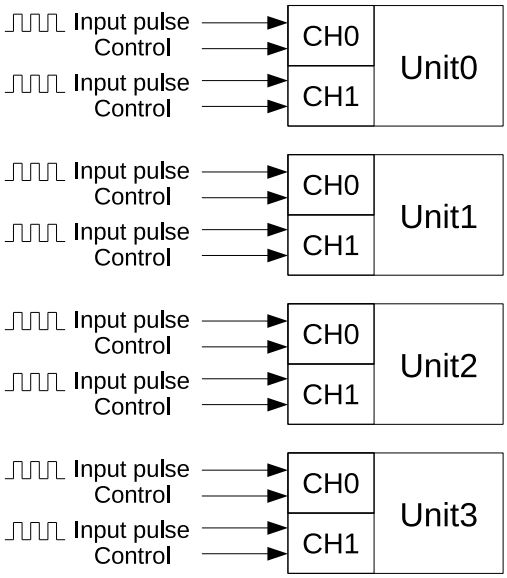


图 51.1-1. PCNT 框图

如图 51.1-1 所示，PCNT 有四个称为“单元”的独立脉冲计数控制器，每个单元都有独立的配置寄存器。每个脉冲计数控制器单元有两个通道和一个 16 位有符号计数器。每个通道对应一路脉冲输入信号以及输入信号滤波模块。16 位有符号计数器可以通过配置实现递增或递减计数功能。PCNT 模块的时钟为 APB_CLK。

51.2 主要特性

PCNT 有如下特性：

- 四个脉冲计数控制器（单元），各自独立工作，计数范围是 1 ~ 65535
- 每个单元有两个独立的通道，共用一个脉冲计数控制器
- 所有通道均有输入脉冲信号（如 sig_ch0_u*n*）和相应的控制信号（如 ctrl_ch0_u*n*）
- 滤波器独立工作，过滤每个单元输入脉冲信号（sig_ch0_u*n* 和 sig_ch1_u*n*）和控制信号（ctrl_ch0_u*n* 和 ctrl_ch1_u*n*）的毛刺
- 每个通道参数如下：

1. 选择在输入脉冲信号的上升沿或下降沿计数
2. 在控制信号为高电平或低电平时可将计数模式配置为递增、递减或停止计数

- 最大脉冲频率：$\frac{f_{APB_CLK}}{2}$

51.3 系统架构

PCNT 每个单元包含两个通道（ch0 和 ch1），两个通道功能相同。下文以通道 0 (ch0) 为例进行介绍，并且描述中 *n* 表示单元编号 0 ~ 3。PCNT 每个通道有两个输入信号：

1. 一个脉冲输入信号（如 sig_ch0_u*n* 为单元 *n* ch0 的脉冲输入信号）
2. 一个控制信号（如 ctrl_ch0_u*n* 为单元 *n* ch0 的控制信号）

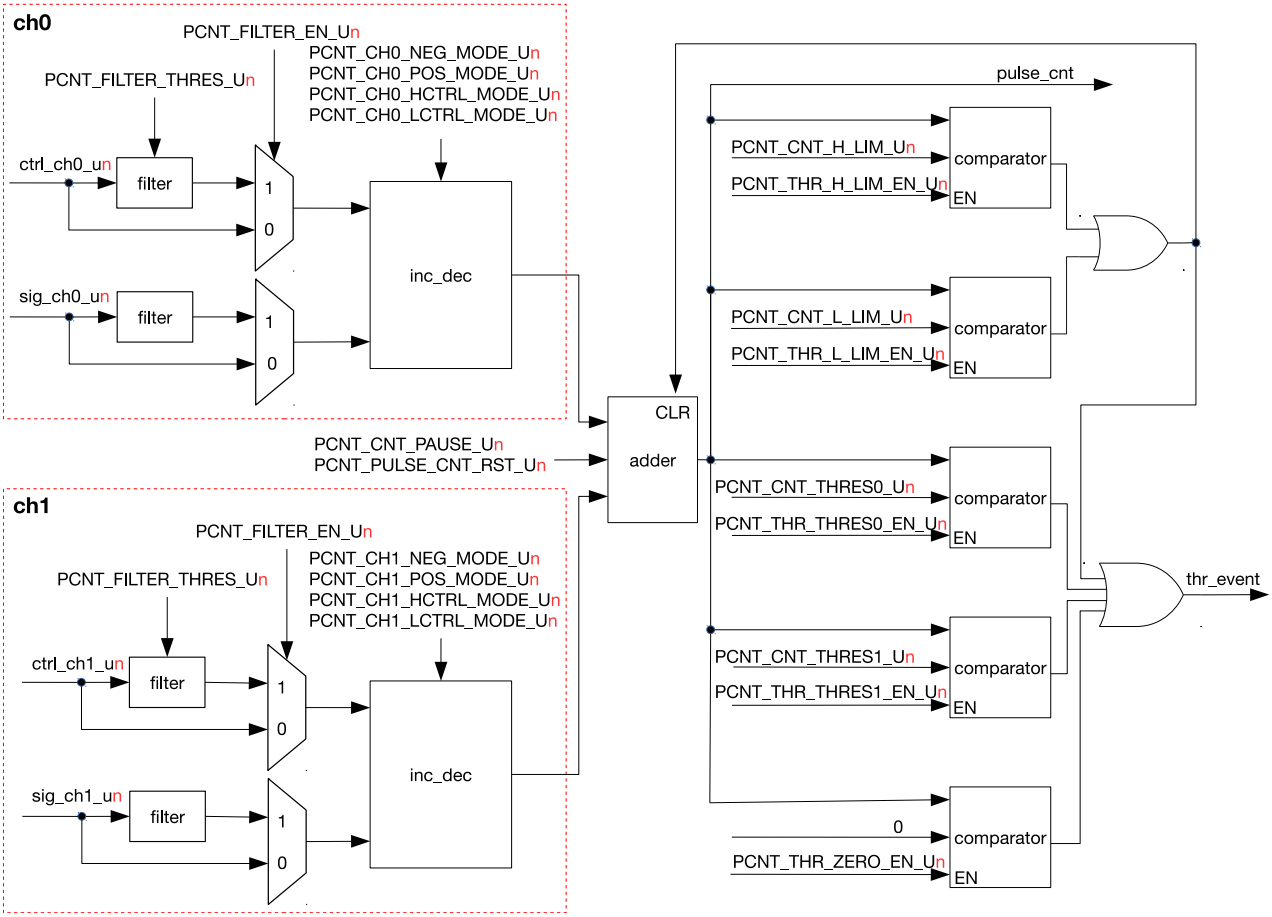


图 51.3-1. PCNT 单元基本架构图

图 51.3-1 为 PCNT 单元的基本架构图。如上所述，ctrl_ch0_u*n* 为单元 *n* ch0 的控制信号，控制信号 ctrl_ch0_u*n* 为高电平或低电平时可配置不同的计数模式，在输入脉冲信号 sig_ch0_u*n* 的上升沿和下降沿计数。

51.4 功能描述

PCNT 的可选计数模式如下：

- 递增模式：通道检测到 sig_ch0_u*n* 的有效边沿（软件可配）时，计数器的值 pulse_cnt 加 1。pulse_cnt 的值达到 PCNT_CNT_H_LIM_U*n* 时被清零。如果 PCNT_CNT_PAUSE_U*n* 置 1，则 pulse_cnt 停止计数。
- 递减模式：通道检测到 sig_ch0_u*n* 的有效边沿（软件可配）时，计数器的值 pulse_cnt 减 1。pulse_cnt 的值达到 PCNT_CNT_L_LIM_U*n* 时被清零。如果 PCNT_CNT_PAUSE_U*n* 置 1，则 pulse_cnt 停止计数。
- 停止计数：计数停止，计数器的值 pulse_cnt 保持不变。

表 51.4-1 至表 51.4-4 说明了如何配置通道 0 的计数模式。

表 51.4-1. 控制信号为低电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 51.4-2. 控制信号为高电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 51.4-3. 控制信号为低电平时输入脉冲信号下降沿的计数模式

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 51.4-4. 控制信号为高电平时输入脉冲信号下降沿的计数模式

PCNT_CHO_NEG_MODE_U*n*	PCNT_CHO_HCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

每个单元均有一个滤波器，作用于该单元的所有控制信号和输入脉冲信号。置位 PCNT_FILTER_EN_U*n* 使能滤波器。滤波器监测信号，滤除脉冲宽度小于 PCNT_FILTER_THRES_U*n* 个 APB 时钟周期的线路毛刺。

如前文所述，每个单元有通道 0 和通道 1 两个通道，处理不同的输入脉冲信号，并通过各自的 inc_dec 模块递增或递减计数值。之后，两个通道将计数值发送给计数器模块，该模块有一个带符号位的 16 位寄存器。软件可以通过置位 PCNT_CNT_PAUSE_U*n* 暂停计数器，也可以通过置位 PCNT_PULSE_CNT_RST_U*n* 清零计数器。

PCNT 可以设置五个观察点，五个观察点共用一个中断，可以通过每个观察点各自的中断使能信号开启或屏蔽中断。

- 最大计数值：当 pulse_cnt 大于等于 PCNT_CNT_H_LIM_U*n* 时，产生上限中断，同时 PCNT_CNT_THR_H_LIM_LAT_U*n* 为高电平。
- 最小计数值：当 pulse_cnt 小于等于 PCNT_CNT_L_LIM_U*n* 时，产生下限中断，同时 PCNT_CNT_THR_L_LIM_LAT_U*n* 为高电平。
- 两个中间阈值：当 pulse_cnt 等于 PCNT_CNT_THRESO_U*n* 或者 PCNT_CNT_THRES1_U*n* 时，产生中断，同时 PCNT_CNT_THR_THRESO_LAT_U*n* 或 PCNT_CNT_THR_THRES1_LAT_U*n* 为高电平。
- 零：当 pulse_cnt 等于 0 时，产生中断，同时 PCNT_CNT_THR_ZERO_LAT_U*n* 有效。

在 PCNT 模块工作过程中，软件如果重新配置 PCNT_CNT_H_LIM_U*n* 和/或 PCNT_CNT_L_LIM_U*n*，则新的配置值会在 pulse_cnt 达到上述五个观察点的任意一个后再生效；软件如果重新配置 PCNT_CNT_THRESO_U*n* 和/或 PCNT_CNT_THRES1_U*n*，则新的配置值会立即生效。

51.5 中断

ESP32-P4 的 PCNT 可以生成以下中断信号并发送给 中断矩阵。

- PCNT_INT

这个中断信号由 PCNT 的内部中断源生成。表 51.5-1 列出了 PCNT 的内部中断源、中断触发条件、以及生成的中断信号。

表 51.5-1. PCNT 的内部中断源

内部中断源	触发条件	中断信号
PCNT_CNT_THR_EVENT_UO_INT	PCNT 单元 0 设置观察点	PCNT_INT

PCNT_CNT_THR_EVENT_U1_INT	PCNT 单元 1 设置观察点	PCNT_INT
PCNT_CNT_THR_EVENT_U2_INT	PCNT 单元 2 设置观察点	PCNT_INT
PCNT_CNT_THR_EVENT_U3_INT	PCNT 单元 3 设置观察点	PCNT_INT

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

51.6 编程指南

每个单元的通道 0 和通道 1 可配置为独立工作或一起工作。下文详细说明了通道 0 独自递增计数、通道 0 独自递减计数和两个通道一起递增计数的应用实例。本节中未详述的通道工作模式（如通道 1 独自递减或递增、双通道一增一减），可参考这三种模式。

51.6.1 通道 0 独自递增计数

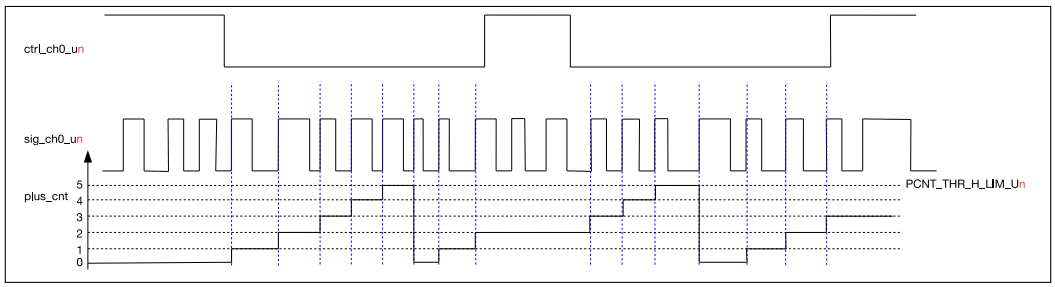


图 51.6-1. 通道 0 递增计数图

图 51.6-1 为通道 0 在 sig_ch0_u*n* 上升沿独立递增计数的示意图，此时通道 1 关闭（请参阅 51.4 一节查看如何关闭通道 1）。通道 0 的配置如下所示。

- PCNT_CH0_LCTRL_MODE_U*n*=0：当 ctrl_ch0_u*n* 为低电平时，递增计数。
- PCNT_CH0_HCTRL_MODE_U*n*=2：当 ctrl_ch0_u*n* 为高电平时，停止计数。
- PCNT_CH0_POS_MODE_U*n*=1：在 sig_ch0_u*n* 的上升沿递增计数。
- PCNT_CH0_NEG_MODE_U*n*=0：在 sig_ch0_u*n* 的下降沿不计数。
- PCNT_CNT_H_LIM_U*n*=5：pulse_cnt 的值递增至 PCNT_CNT_H_LIM_U*n* 时被清零。

51.6.2 通道 0 独自递减计数

图 51.6-2 为通道 0 在 sig_ch0_u*n* 上升沿独立递减计数的示意图，此时通道 1 关闭。此时通道 0 的配置与图 51.6-1 相比有如下区别：

- PCNT_CH0_POS_MODE_U*n*=2：即在 sig_ch0_u*n* 的上升沿递减计数。

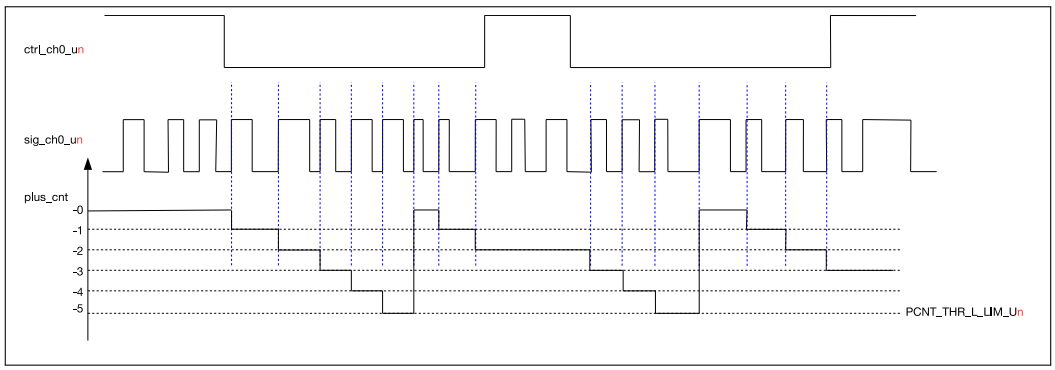


图 51.6-2. 通道 0 递减计数图

- PCNT_CNT_L_LIM_U*n*=-5：pulse_cnt 的值递减到 PCNT_CNT_L_LIM_U*n* 时被清零。

51.6.3 通道 0 和通道 1 同时递增计数

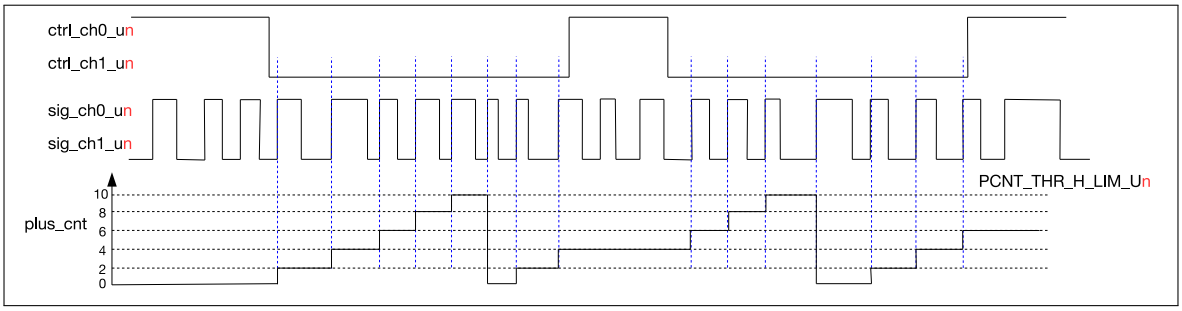


图 51.6-3. 双通道递增计数图

图 51.6-3 为通道 0 和通道 1 分别在 sig_ch0_u*n* 和 sig_ch1_u*n* 上升沿同时递增计数的示意图。如图 51.6-3 所示，控制信号 ctrl_ch0_u*n* 与 ctrl_ch1_u*n* 的波形一致，输入脉冲信号 sig_ch0_u*n* 和 sig_ch1_u*n* 波形一致。具体配置如下：

- 通道 0：
  - PCNT_CH0_LCTRL_MODE_U*n*=0：当 ctrl_ch0_u*n* 为低电平时，递增计数。
  - PCNT_CH0_HCTRL_MODE_U*n*=2：当 ctrl_ch0_u*n* 为高电平时，停止计数。
  - PCNT_CH0_POS_MODE_U*n*=1：在 sig_ch0_u*n* 的上升沿递增计数。
  - PCNT_CH0_NEG_MODE_U*n*=0：在 sig_ch0_u*n* 的下降沿不计数。
- 通道 1：
  - PCNT_CH1_LCTRL_MODE_U*n*=0：当 ctrl_ch1_u*n* 为低电平时，递增计数。
  - PCNT_CH1_HCTRL_MODE_U*n*=2：当 ctrl_ch1_u*n* 为高电平时，停止计数。
  - PCNT_CH1_POS_MODE_U*n*=1：在 sig_ch1_u*n* 的上升沿递增计数。
  - PCNT_CH1_NEG_MODE_U*n*=0：在 sig_ch1_u*n* 的下降沿不计数。
- PCNT_CNT_H_LIM_U*n*=10：pulse_cnt 递增至 PCNT_CNT_H_LIM_U*n* 时被清零。

51.7 寄存器列表

本小节的所有地址均为相对于 脉冲计数控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

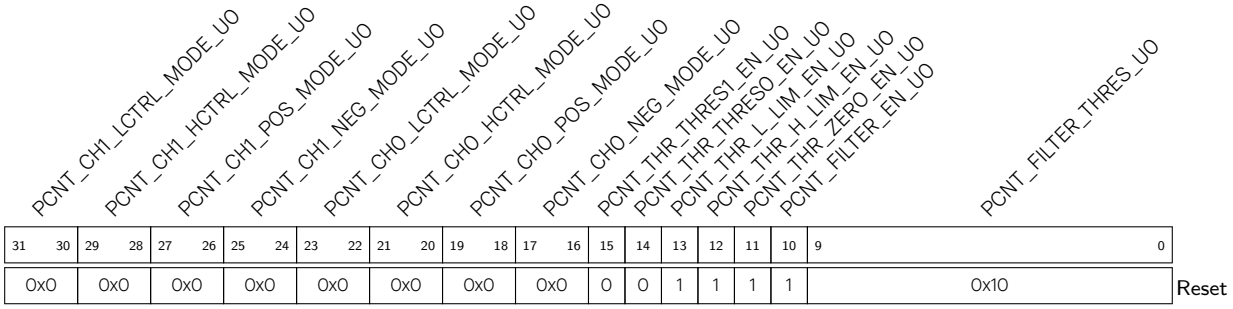
名称	描述	地址	访问
配置寄存器			
PCNT_U0_CONF0_REG	单元 0 的配置寄存器 0	0x0000	R/W
PCNT_U0_CONF1_REG	单元 0 的配置寄存器 1	0x0004	R/W
PCNT_U0_CONF2_REG	单元 0 的配置寄存器 2	0x0008	R/W
PCNT_U1_CONF0_REG	单元 1 的配置寄存器 0	0x000C	R/W
PCNT_U1_CONF1_REG	单元 1 的配置寄存器 1	0x0010	R/W
PCNT_U1_CONF2_REG	单元 1 的配置寄存器 2	0x0014	R/W
PCNT_U2_CONF0_REG	单元 2 的配置寄存器 0	0x0018	R/W
PCNT_U2_CONF1_REG	单元 2 的配置寄存器 1	0x001C	R/W
PCNT_U2_CONF2_REG	单元 2 的配置寄存器 2	0x0020	R/W
PCNT_U3_CONF0_REG	单元 3 的配置寄存器 0	0x0024	R/W
PCNT_U3_CONF1_REG	单元 3 的配置寄存器 1	0x0028	R/W
PCNT_U3_CONF2_REG	单元 3 的配置寄存器 2	0x002C	R/W
PCNT_CTRL_REG	所有计数器的控制寄存器	0x0060	R/W
状态寄存器			
PCNT_U0_CNT_REG	单元 0 的计数器值	0x0030	RO
PCNT_U1_CNT_REG	单元 1 的计数器值	0x0034	RO
PCNT_U2_CNT_REG	单元 2 的计数器值	0x0038	RO
PCNT_U3_CNT_REG	单元 3 的计数器值	0x003C	RO
PCNT_U0_STATUS_REG	脉冲计数器单元 0 的状态寄存器	0x0050	RO
PCNT_U1_STATUS_REG	脉冲计数器单元 1 的状态寄存器	0x0054	RO
PCNT_U2_STATUS_REG	脉冲计数器单元 2 的状态寄存器	0x0058	RO
PCNT_U3_STATUS_REG	脉冲计数器单元 3 的状态寄存器	0x005C	RO
中断寄存器			
PCNT_INT_RAW_REG	原始中断状态寄存器	0x0040	RO
PCNT_INT_ST_REG	中断状态寄存器	0x0044	RO
PCNT_INT_ENA_REG	中断使能寄存器	0x0048	R/W
PCNT_INT_CLR_REG	中断清除寄存器	0x004C	WO
版本寄存器			
PCNT_DATE_REG	脉冲计数器的版本控制寄存器	0x00FC	R/W

51.8 寄存器

本小节的所有地址均为相对于 脉冲计数控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 51.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xC**n*)



PCNT_FILTER_THRES_U*n* 配置滤波器的最大阈值。滤波器启动时，任何小于该值的脉冲都会被过滤。

单位：APB_CLK 时钟周期。

(R/W)

PCNT_FILTER_EN_U*n* 单元 *n* 输入滤波器的使能位。(R/W)

PCNT_THR_ZERO_EN_U*n* 单元 *n* 过零比较器的使能位。(R/W)

PCNT_THR_H_LIM_EN_U*n* 单元 *n* 上限比较器的使能位，控制上限中断事件的产生。(R/W)

PCNT_THR_L_LIM_EN_U*n* 单元 *n* 下限比较器的使能位，控制下限中断事件的产生。(R/W)

PCNT_THR_THRES0_EN_U*n* 单元 *n* 阈值 0 比较器的使能位。(R/W)

PCNT_THR_THRES1_EN_U*n* 单元 *n* 阈值 1 比较器的使能位。(R/W)

PCNT_CH0_NEG_MODE_U*n* 配置通道 0 输入信号检测下降沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH0_POS_MODE_U*n* 配置通道 0 输入信号检测上升沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH0_HCTRL_MODE_U*n* 控制信号为高电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

见下页...

Register 51.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xC**n*)

接上页...

PCNT_CH0_LCTRL_MODE_U*n* 控制信号为低电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

PCNT_CH1_NEG_MODE_U*n* 配置通道 1 输入信号检测下降沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH1_POS_MODE_U*n* 配置通道 1 输入信号检测上升沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH1_HCTRL_MODE_U*n* 控制信号为高电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

PCNT_CH1_LCTRL_MODE_U*n* 控制信号为低电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

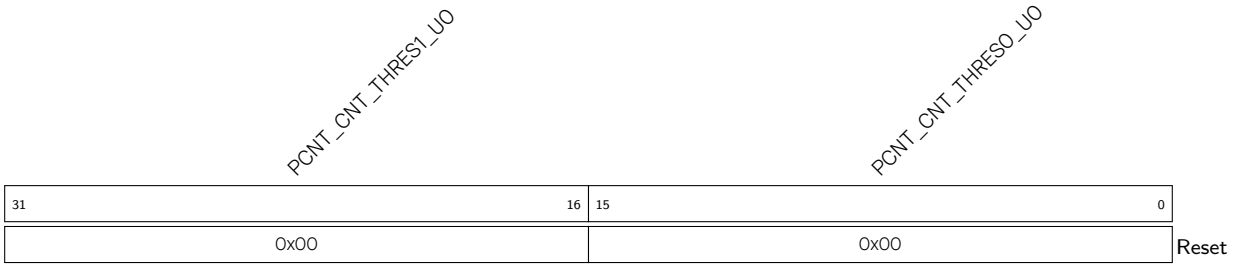
0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

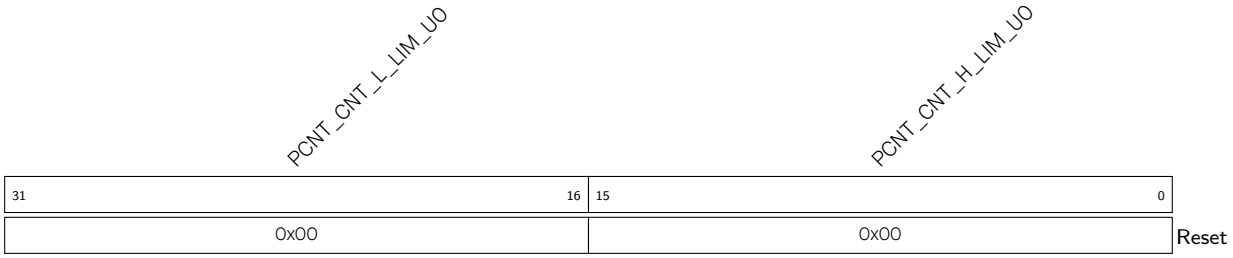
Register 51.2. PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+0xC*n*)



PCNT_CNT_THRES0_U*n* 配置单元 *n* 阈值 0 的值。(R/W)

PCNT_CNT_THRES1_U*n* 配置单元 *n* 阈值 1 的值。(R/W)

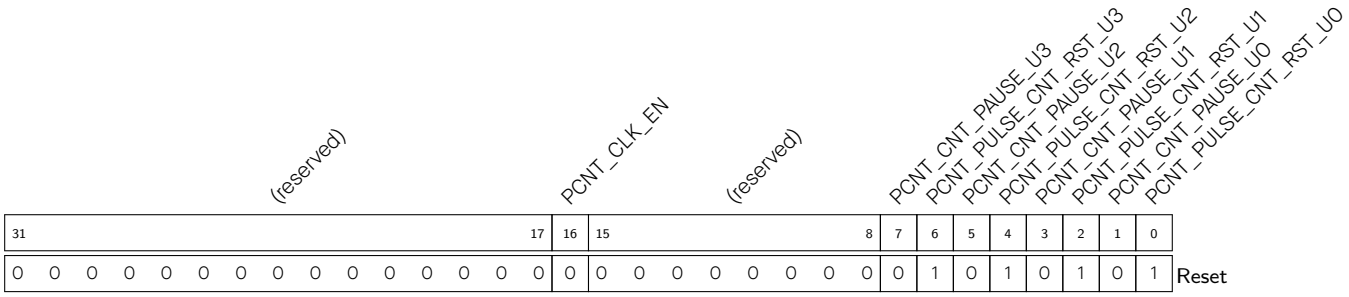
Register 51.3. PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+0xC*n*)



PCNT_CNT_H_LIM_U*n* 配置单元 *n* 的计数上限阈值。当脉冲计数达到上限阈值，计数会被清零。(R/W)

PCNT_CNT_L_LIM_U*n* 配置单元 *n* 的计数下限阈值。当脉冲计数达到下限阈值，计数会被清零。(R/W)

Register 51.4. PCNT_CTRL_REG (0x0060)



PCNT_PULSE_CNT_RST_U*n* 写 1 清零单元 *n* 的计数器。

0：无效

1：清零

(R/W)

PCNT_CNT_PAUSE_U*n* 写 1 暂停单元 *n* 的计数器。

0：无效

1：暂停

(R/W)

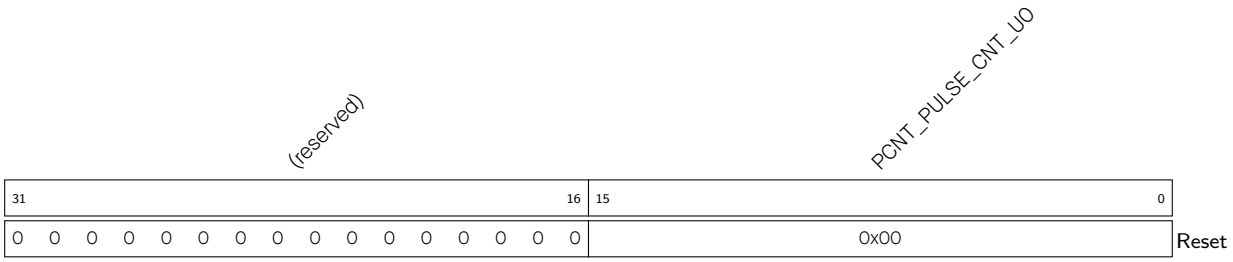
PCNT_CLK_EN 配置是否使能脉冲计数器模块寄存器时钟门控。

0：寄存器时钟仅在有读写操作时开启

1：寄存器时钟一直开启

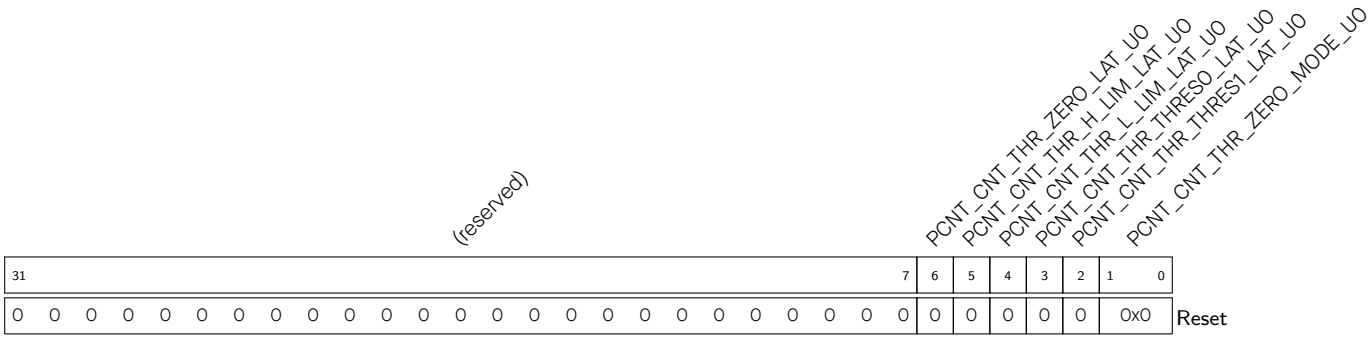
(R/W)

Register 51.5. PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0030+0x4**n*)



PCNT_PULSE_CNT_U*n* 表示单元 *n* 脉冲计数器的当前值。(RO)

Register 51.6. PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+0x4*n*)



PCNT_CNT_THR_ZERO_MODE_U*n* 表示 PCNT_U*n* 为 0 时的脉冲计数器状态。

0：脉冲计数器的值由正数减至 0

1：脉冲计数器的值由负数增至 0

2：脉冲计数器为负

3：脉冲计数器为正

(RO)

PCNT_CNT_THR_THRES1_LAT_U*n* 表示阈值 1 比较器的使能时，PCNT_U*n* 当前值是否与阈值 1 相等。

0：其他

1：脉冲计数器的当前值与阈值 1 相等，阈值 1 有效

(RO)

PCNT_CNT_THR_THRESO_LAT_U*n* 表示阈值 0 比较器的使能时，PCNT_U*n* 当前值是否与阈值 0 相等。

0：其他

1：脉冲计数器的当前值与阈值 0 相等，阈值 0 有效

(RO)

PCNT_CNT_THR_L_LIM_LAT_U*n* 表示下限比较器使能时，PCNT_U*n* 的当前值是否与下限阈值相等。

0：其他

1：脉冲计数器的当前值与下限阈值相等，下限有效

(RO)

PCNT_CNT_THR_H_LIM_LAT_U*n* 表示上限比较器的使能时，PCNT_U*n* 的当前值是否与上限阈值相等。

0：其他

1：脉冲计数器的当前值与上限阈值相等，上限有效

(RO)

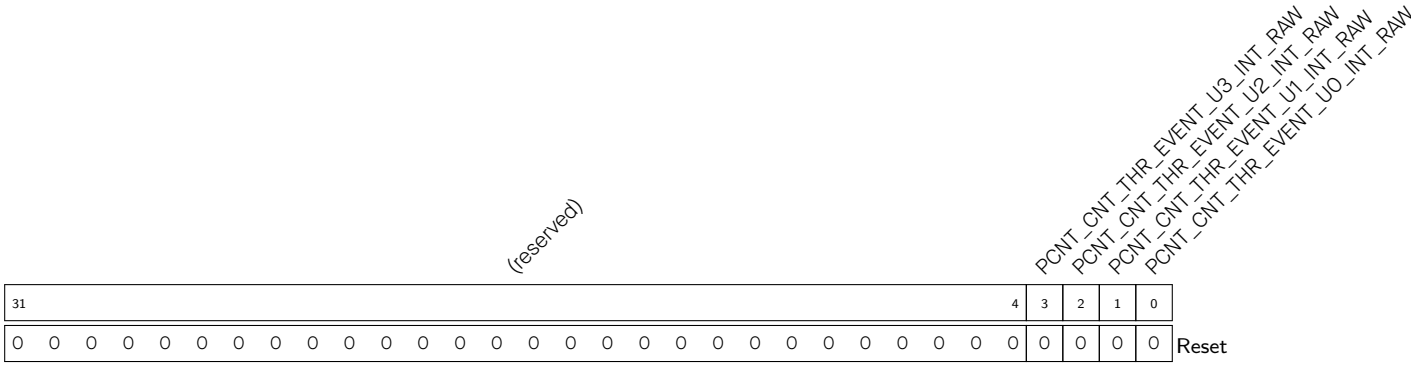
PCNT_CNT_THR_ZERO_LAT_U*n* 表示过零比较器使能时，PCNT_U*n* 的当前值是否为 0。

0：其他

1：脉冲计数器的当前值为 0，阈值 0 有效

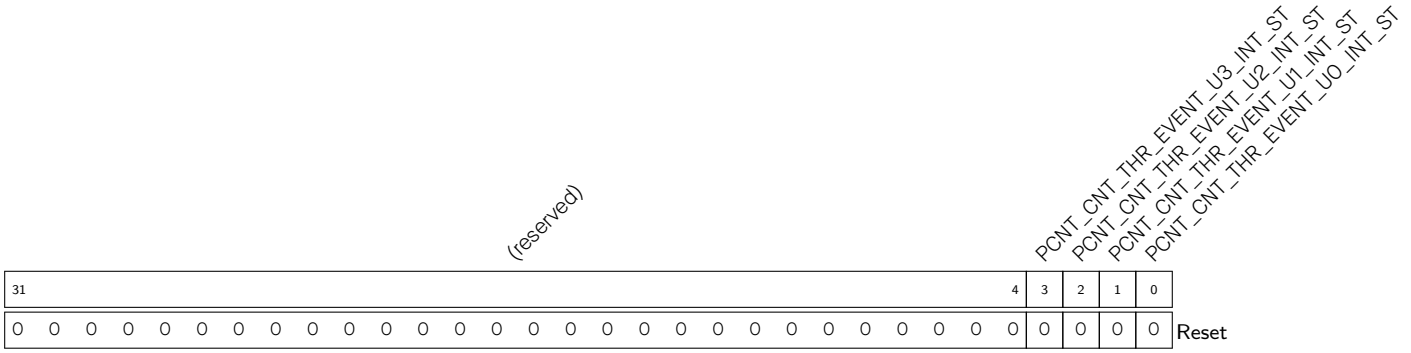
(RO)

Register 51.7. PCNT_INT_RAW_REG (0x0040)



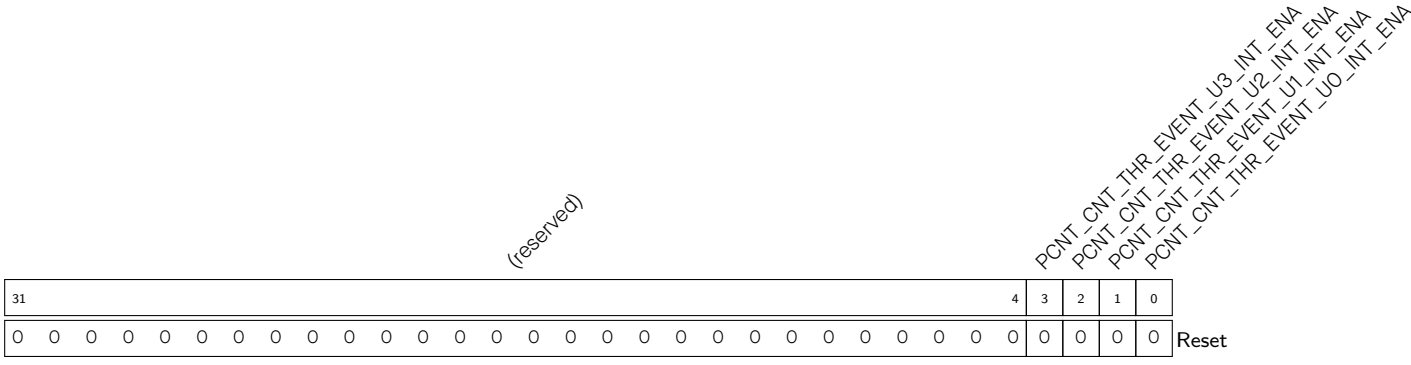
PCNT_CNT_THR_EVENT_U*n*_INT_RAW 单元 *n* 事件中断的原始中断状态。(RO)

Register 51.8. PCNT_INT_ST_REG (0x0044)



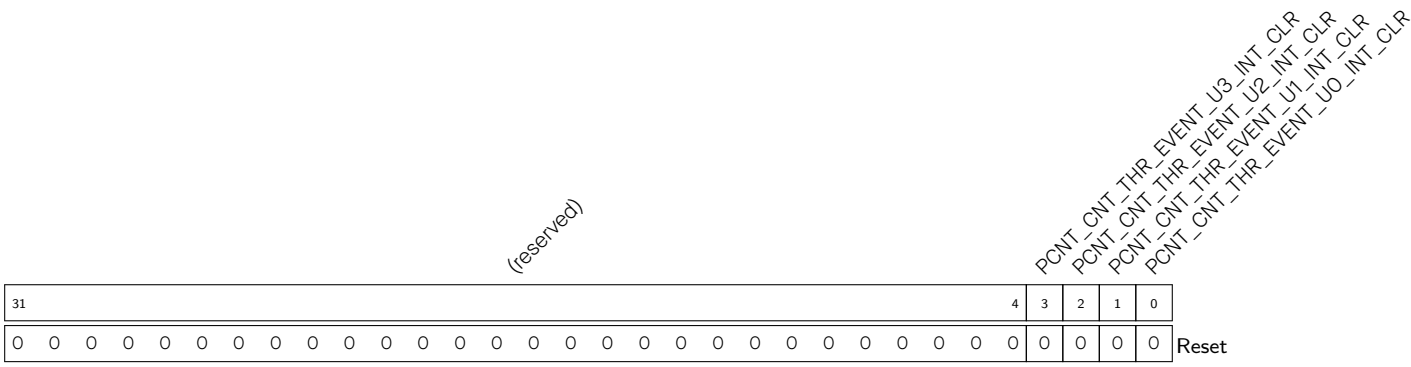
PCNT_CNT_THR_EVENT_U*n*_INT_ST 单元 *n* 事件中断的屏蔽中断状态。(RO)

Register 51.9. PCNT_INT_ENA_REG (0x0048)



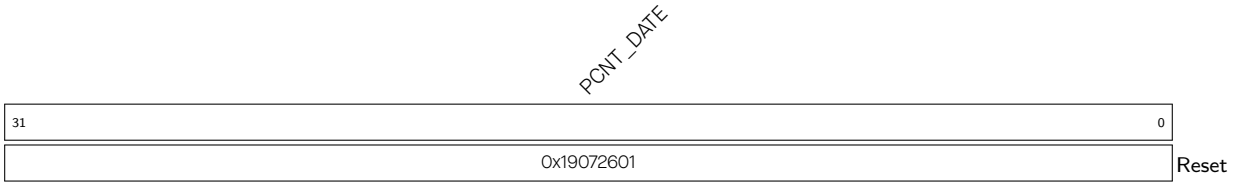
PCNT_CNT_THR_EVENT_U*n*_INT_ENA 写 1 使能单元 *n* 事件中断。(R/W)

Register 51.10. PCNT_INT_CLR_REG (0x004C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR 写 1 清除单元 *n* 事件中断。(WO)

Register 51.11. PCNT_DATE_REG (0x00FC)



PCNT_DATE 版本控制寄存器。(R/W)

第 52 章

USB 2.0 高速 OTG

52.1 概述

ESP32-P4 带有一个集成了收发器的 USB 2.0 高速 OTG 外设，下文将称为 OTG_HS。该 OTG_HS 外设符合 USB 2.0 协议规范，同时兼容 OTG 1.3 协议和 OTG 2.0 协议。OTG_HS 支持 USB 2.0 传输速率为 480 Mbit/s 的高速模式 (High-Speed, HS)、传输速率为 12 Mbit/s 的全速模式 (Full-Speed, FS) 和传输速率为 1.5 Mbit/s 的低速模式 (Low-Speed, LS)。

- 处于高速模式和全速模式的 OTG_HS 可配置成 Host，也可以配置成 Device。
- 处于低速模式的 OTG_HS 只可配置成 Host。

52.2 术语

本章使用到的缩写及术语如下：

主机 (Host)	主机是安装了 USB 主机控制器的计算机系统，包括：主机硬件平台（CPU、总线等）和使用的操作系统。
设备 (Device)	设备是执行某个功能的逻辑或物理实体。具体描述的实体取决于上下文。在最低层次上，设备可能指一个单一的硬件组件，例如存储设备。在较高层次上，设备可能指执行特定功能的一组硬件组件，例如 USB 接口设备。在更高层次上，设备可能指连接到 USB 的实体，例如数据/传真调制解调器设备。非特指时，USB 设备可以是集线器或功能设备。
高速 (High-speed)	USB 运行在 480 Mbit/s 速率下。
全速 (Full-speed)	USB 运行在 12 Mbit/s 速率下。
低速 (Low-speed)	USB 运行在 1.5 Mbit/s 速率下。
端点 (Endpoint)	USB 设备的一个可以唯一寻址的部分，是在主机和设备之间通信流中信息的源或接收点。
Scatter/Gather DMA 模式	通过 DMA 访问不连续的存储区域。
Buffer DMA 模式	通过 DMA 访问连续存储区域。
Slave 模式	通过 CPU 访问存储区域。
RX FIFO	存储接收数据。
TX FIFO	存储发送数据。
周期性 TX FIFO	存储待发送的同步传输和中断传输数据。
非周期性 TX FIFO	存储待发送的控制传输和批量传输数据。

52.3 特性

52.3.1 通用特性

- 兼容 USB 2.0 协议、OTG 1.3 协议、OTG 2.0 协议
- 支持高速速率、全速速率、低速速率
- 在全速和高速模式下既可作为主机，也可以充当设备
- 动态分配 FIFO (DFIFO) 大小，每个设备 EP/主机通道最大可动态分配 4 KB FIFO
- 支持多种存储器访问模式
  - Scatter/Gather DMA 模式
  - Buffer DMA 模式
  - Slave 模式
- 集成 UTMI 高速收发器

52.3.2 设备模式特性

- 端点 0 永远存在，双向控制，由 EP0 IN 和 EP0 OUT 组成
- 15 个附加端点 1 ~ 15，可配置为 IN 或 OUT
- 最多 8 个 IN 端点同时工作，包括 EP0 IN
- 所有 OUT 端点共享一个 RX FIFO
- 每个 IN 端点都有专用的 TX FIFO

52.3.3 主机模式特性

- 16 个主机通道
- 一个 RX FIFO：由所有周期事务和非周期事务共用
- 两个 TX FIFO：
  - 所有非周期事务传输共用一个 TX FIFO
  - 所有周期事务传输共用另一个 TX FIFO
- 上述所有 FIFO 共用 4 KB RAM
- 每个 FIFO 大小可配置，最大 4 KB

52.4 功能描述

52.4.1 控制器内核与接口

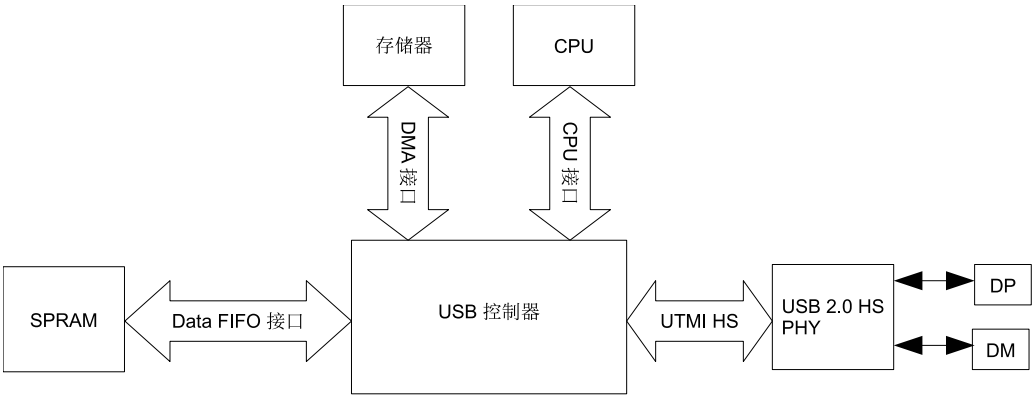


图 52.4-1. OTG_HS 系统架构

OTG_HS 采用与 OTG_FS 相同的控制器内核，称为 USB 控制器内核。如图 52.4-1 所示，控制器内核有以下接口：

- **CPU 接口**
  CPU 可以通过该接口读写控制器内核的多个寄存器和 FIFO。该接口在内部实现为 AHB 从机接口。通过该接口访问 FIFO 的方式称为 Slave 模式。
- **DMA 接口**
  控制器内核的内部 DMA 可以通过该接口读写系统存储器，例如在 DMA 模式下获取和写入有效数据。该接口在内部实现为 AHB 主机接口。
- **USB 2.0 接口 (USB 2.0 HS PHY)**
  控制器内核通过该接口连接 USB 2.0 UTMI 串行收发器。该串行收发器使得 OTG_HS 可以支持高速和全速速率。
- **数据 FIFO 接口**
  控制器内核使用的多个 FIFO 实际上并不位于控制器内核内部，而是位于单端口 RAM (SPRAM) 上。FIFO 的大小可动态配置，运行时在 SPRAM 中进行分配。CPU、DMA 或控制器通过该接口读写 FIFO。

52.4.2 存储器布局

图 52.4-2 展示了用于配置和控制 USB 控制器内核的寄存器布局。

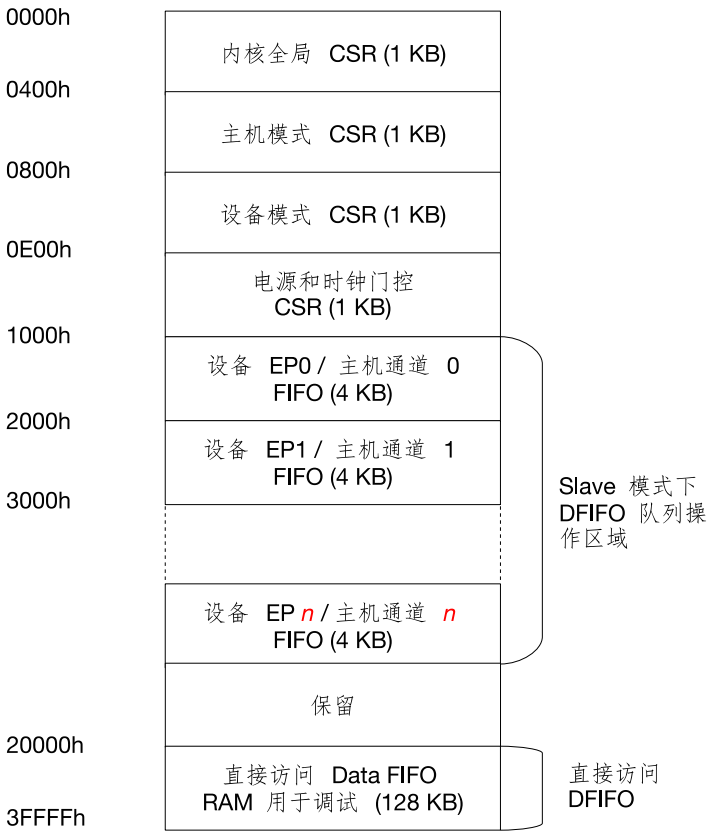


图 52.4-2. 内核地址映射

52.4.2.1 控制 & 状态寄存器 (CSR)

- **内核全局 CSR**
  内核全局寄存器用于配置/控制 OTG_HS 的通用功能，即主机和设备模式共同的功能，并表示其内部状态。通用功能包括 OTG 控制，以及系统级中断。在主机和设备模式下，软件均可以对内核全局 CSR 进行访问。
- **主机模式 CSR**
  主机模式寄存器用于主机模式下的配置/控制/状态表示，只能在主机模式下被访问。每个通道各自有一组主机模式寄存器。
- **设备模式 CSR**
  设备模式寄存器用于设备模式下的配置/控制/状态表示，只能在设备模式下被访问。每个端点各自有一组设备模式寄存器。
- **电源和时钟门控寄存器**
  此单一寄存器用于控制模块电源和门控时钟。

52.4.2.2 FIFO 访问

OTG_HS 利用多个 FIFO 缓冲发送或接收有效数据。FIFO 的数量和类型取决于主机或设备模式，以及所使用的通道或端点的数量，参考章节 52.4.3。FIFO 访问有两种方式：DMA 模式和 Slave 模式。当使用 Slave 模式时，CPU 需要通过读写 DFIFO 的推入/弹出操作 (push/pop) 区域或读写调试区域来访问这些 FIFO。FIFO 访问遵循以下规则：

- 对 4 KB 推入/弹出区域中任何地址的读访问将在共享 RX FIFO 中产生一个弹出 (pop) 操作。

- 对特定 4 KB 推入/弹出区域的写访问将写入相应的端点或通道的 TX FIFO。前提是该端点是 IN 端点，或者该通道是 OUT 通道。
  - 在设备模式下，数据写入相应的 IN 端点的专用 TX FIFO。
  - 在主机模式下，根据通道是非周期性通道还是周期性通道，数据写入非周期性 TX FIFO 或周期性 TX FIFO。
- 访问 128 KB 读写调试区域将直接读/写，而不是进行推入/弹出操作。此种访问通常仅用于调试目的。

请注意，仅在 Slave 模式下，才需要由 CPU 直接向 FIFO 进行数据操作。在 DMA 模式下，内部 DMA 将处理 TX FIFO 和 RX FIFO 的数据推入/弹出操作。

52.4.3 FIFO 和队列组织

OTG_HS 中的 FIFO 主要用于保存有效数据，即 USB 数据包的数据字段。TX FIFO 用于存储将由主机模式下的 OUT 事务或设备模式下的 IN 事务发送的有效数据。RX FIFO 用于存储主机模式下 IN 事务或设备模式下 OUT 事务的已接收的有效数据。除了存储有效数据之外，RX FIFO 还存储每个有效数据的**状态条目**。状态条目中包含关于有效数据的信息，如：通道编号、字节数、是否有效等。在 Slave 模式下，状态条目还用于指示各类通道事件。

可用于 FIFO 分配的 SPRAM 大小为 1024 × 35 位。35 位包括 32 个数据位加 3 个控制位。在主机模式下各个通道或设备模式下端点使用的多个 FIFO 被分配到 SPRAM 中，并且可以动态调整大小。

52.4.3.1 主机模式 FIFO 和队列

如图 52.4-3 所示，主机模式使用以下 FIFO：

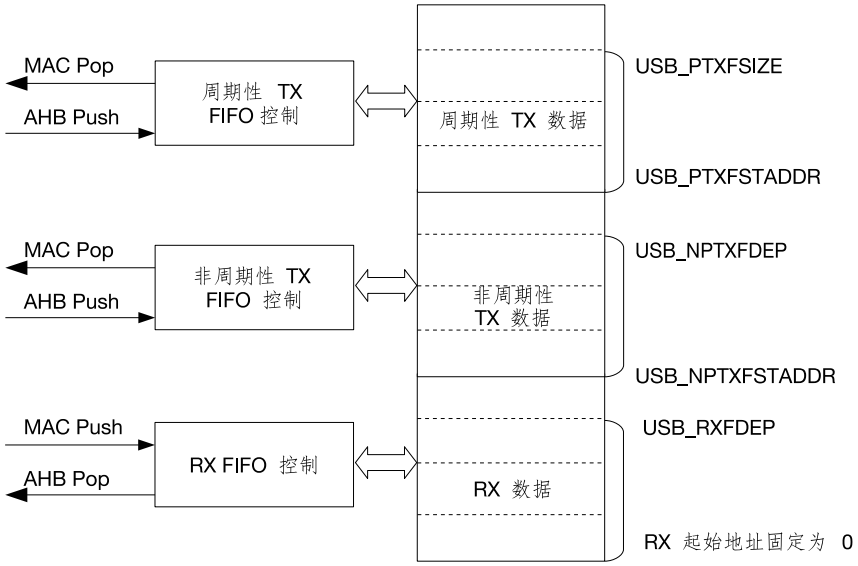


图 52.4-3. 主机模式 FIFO

- **非周期性 TX FIFO**：存储所有通道的批量和控制类型的 OUT 事务的有效数据。
- **周期性 TX FIFO**：存储所有通道的中断或同步类型的 OUT 事务的有效数据。
- **RX FIFO**：存储所有 IN 事务的有效数据，以及用于指示有效数据大小和事务/通道事件的状态条目。例如传输完成或通道暂停。

除 FIFO 外，主机模式还包含两个请求队列，用于存储来自多个通道的事务请求。请求队列中的每个条目都包含 IN/OUT 通道编号以及执行事务的其他信息，例如事务类型。请求队列也用于存储其他请求类型，例如通道暂停请求。

与 FIFO 不同，请求队列的大小是固定的，且不能由软件直接访问。相反，一旦启用了通道，主机内核会自动将请求写入请求队列。请求写入队列的顺序决定了 USB 事务的处理顺序。

主机模式包含以下请求队列：

- **非周期性请求队列**：针对非周期性批量和控制事务的请求队列，最多可以存储 4 个条目。
- **周期性请求队列**：针对周期性中断和同步事务的请求队列，最多可以存储 4 个条目。

调度事务时，硬件将首先执行周期性请求队列上的所有请求，再执行非周期性请求队列上的请求。

52.4.3.2 设备模式 FIFO

如图 52.4-4 所示，设备模式使用以下 FIFO：

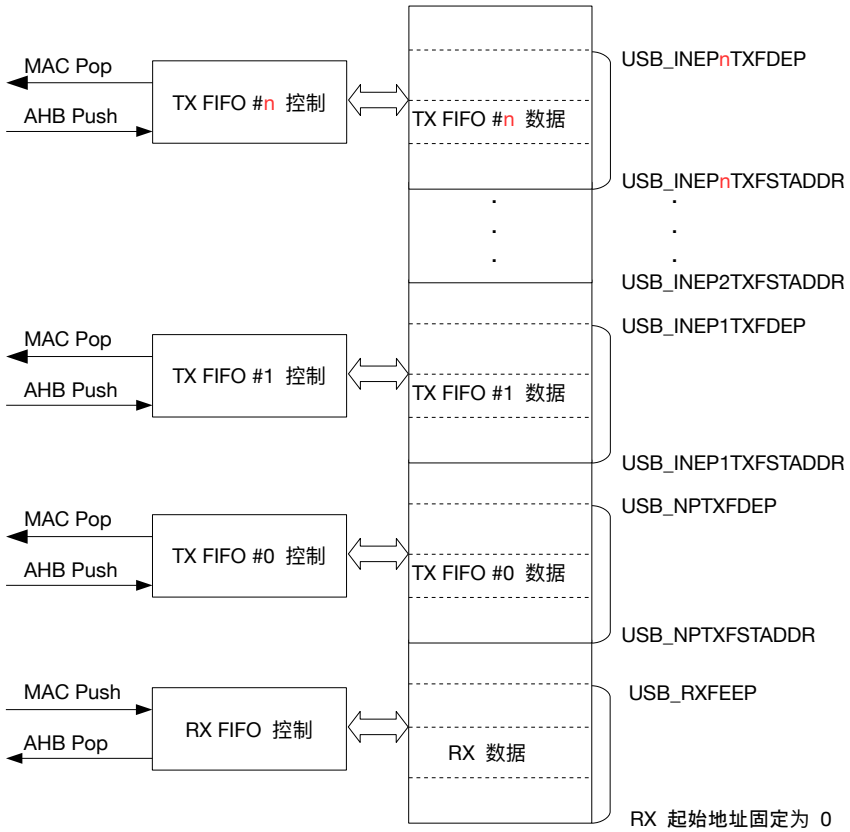


图 52.4-4. 设备模式 FIFO

- **RX FIFO**：存储数据包内接收的有效数据和状态条目，用于指示有效数据的大小。
- **专用 TX FIFO**：每个使能的 IN 端点都有一个专用的 TX FIFO，用于存储该端点的所有 IN 有效数据，而无论事务类型，包括周期性或非周期性 IN 事务。

由于有专用的 FIFO，设备模式不使用任何请求队列。IN 事务的顺序由主机确定。

52.4.4 中断层次结构

OTG_HS 有一条中断线，可以通过中断矩阵连接到一个 CPU。可以通过置位 USB_GLBLINTRMSK 来显示中断信号。OTG_HS 中断是 USB_GINTSTS_REG 寄存器中所有位的逻辑或 (OR)，且置位 USB_GINTMSK_REG 寄存器中的相应位可以使能 USB_GINTSTS_REG 中的位。USB_GINTSTS_REG 包含系统级中断，还包含主机或设备模式专有的中断位以及 OTG 有关中断。OTG_HS 中断源的层次结构如图 52.4-5 所示。

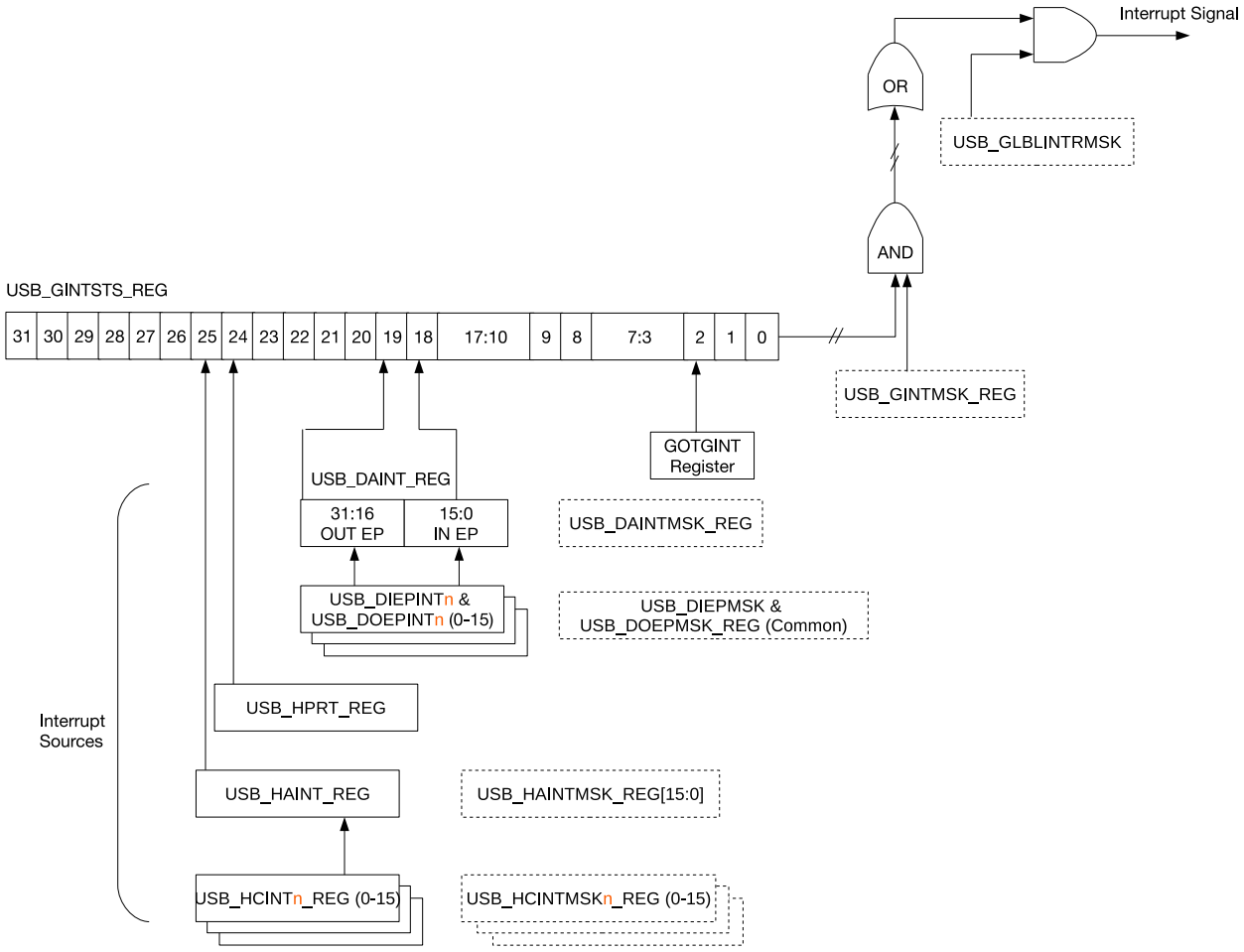


图 52.4-5. OTG_HS 中断层次结构图

USB_GINTSTS_REG 寄存器的以下位指示较低层级的中断源：

- **USB_PRTINT** 表示主机端口有未处理的中断。USB_HPRT_REG 寄存器指示中断源。
- **USB_HCHINT** 表示一个或多个主机通道有未处理的中断。通过读取 USB_HAINT_REG 寄存器可以确定哪些通道有未处理的中断，然后查询该通道的 USB_HCINT*n*_REG 寄存器以确定中断源。
- **USB_OEPINT** 表示一个或多个 OUT 端点有未处理的中断。通过读取 USB_DAINT_REG 寄存器可以确定哪些 OUT 端点有未处理的中断，然后查询 OUT 端点的 USB_DOEPINT*n*_REG 寄存器以确定中断源。
- **USB_IEPINT** 表示一个或多个 IN 端点有未处理的中断。通过读取 USB_DAINT_REG 寄存器可以确定哪些 IN 端点有未处理的中断，然后查询 IN 端点的 USB_DIEPINT*n*_REG 寄存器以确定中断源。
- **USB_OTGINT** 表示 OTG 事件已触发中断。查询 USB_GOTGINT_REG 寄存器以确定哪些 OTG 事件触发了中断。

52.4.5 DMA 模式和 Slave 模式

USB OTG 支持 3 种存储器访问方式：Slave 模式、Buffer DMA 模式、和 Scatter/Gather DMA 模式。

52.4.5.1 Slave 模式

在 Slave 模式下，所有有效数据放入 FIFO 或从 FIFO 中取出都必须通过 CPU 进行。

- 使用 IN 端点或 OUT 通道发送数据包时，必须将有效数据放入相应的端点或通道的 TX FIFO 中。
- 接收到数据包时，必须先通过读取 USB_GRXSTSP_REG 从 RX FIFO 中取出数据包的状态条目，以确定数据包中有效数据的长度，以字节为单位。然后必须由 CPU 通过读取 RX FIFO 中的存储区域，手动从 RX FIFO 中取出相应的字节数。

52.4.5.2 Buffer DMA 模式

Buffer DMA 模式类似于 Slave 模式，不同之处在于该模式为利用内部 DMA 将有效数据放入 FIFO 或从 FIFO 中取出。

- 使用 IN 端点或 OUT 通道传输数据包时，应将有效数据的存储地址写入 USB_HCDMA*n*_REG（主机模式）或 USB_DIEPDMA*n*_REG（设备模式）。启用端点或通道后，内部 DMA 会将有效数据从存储器中推送到通道或端点的 TX FIFO 中。
- 使用 OUT 端点或 IN 通道接收数据包时，应将存储器中空 buffer 的地址写入 USB_HCDMA*n*_REG（主机模式）或 USB_DOEPDMA*n*_REG（设备模式）。启用端点或通道后，内部 DMA 将把有效数据从 RX FIFO 弹出到相应 buffer 中。

52.4.5.3 Scatter/Gather DMA 模式

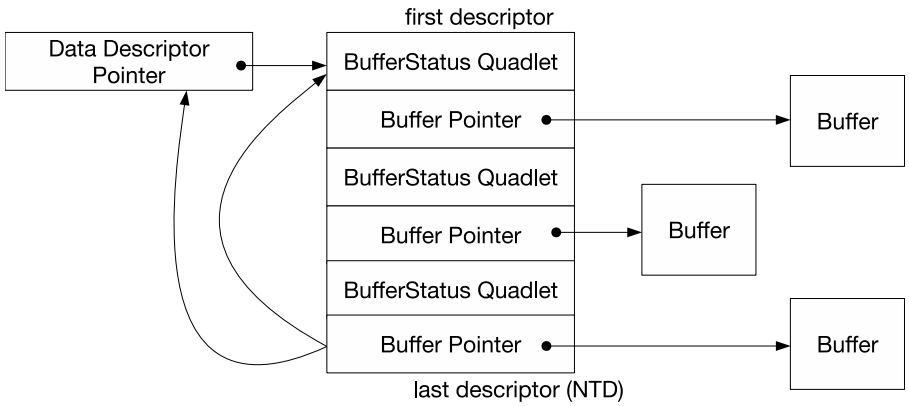


图 52.4-6. Scatter/Gather DMA 链表结构

在 Scatter/Gather DMA 模式下，包含有效数据的接收可能分散在整个存储器各处。每个端点或通道都有一个连续的 DMA 描述符列表。每个描述符包含一个指向有效数据或接收 buffer 的 32 位指针和一个 32 位的 buffer 描述符 (BufferStatus Quadlet)。有效数据和接收 buffer 可以对应单个事务，即 < 1 MPS 字节，MPS：最大数据包大小，或整个传输，即 > 1 MPS 字节。该列表的实现为环形 buffer ，意味着 DMA 在遇到列表中的最后一个条目时将返回至第一个条目。

- 当使用 IN 端点或 OUT 通道发送传输/事务时，DMA 将从多个 buffer 收集有效数据并将其放入 TX FIFO。

- 当使用 OUT 端点或 IN 通道接收传输/事务时，DMA 将取出来自 RX FIFO 的接收有效数据，并将它们分别存放到 DMA 列表条目所指向的多个 buffer 中。

52.4.6 事务和传输级操作

在主机或设备模式下，通信可以在事务级或传输级进行。

52.4.6.1 DMA 模式下的事务和传输级操作

在 DMA 模式下的传输级操作，只有当一个传输通道被终止时，中断才会发生。以下三种情况下发生传输通道终止：传输通道中所有要求传送的数据全部成功传送、接收到 STALL 或出现连续事物级错误，例如，3 个连续的事务级错误。在 DMA 设备模式下操作时，所有错误均由控制器内核进行处理。

在 DMA 模式下的事务级操作，传输大小是一个数据包的大小，即最大数据包大小或短数据包大小。

52.4.6.2 Slave 模式下的事务和传输级操作

在 Slave 模式下的事务级操作，一次只能处理一个事务。每组有效数据应对应一个数据包，软件必须根据 USB 接收到的握手应答，例如 ACK 或 NAK 来确定是否需要重启事务。

下表描述了 Slave 模式下进行 IN 和 OUT 事务级操作的方法。

表 52.4-1. Slave 模式下的 IN 和 OUT 事务级操作

主机模式	设备模式
OUT 事务	
1. 软件配置 USB_HCTSIZ*n*_REG，指定数据包的大小和数量（1 个），使能该通道，然后将数据包的有效数据复制到 TX FIFO 中。 2. 软件在写完每个数据包的最后一个 DWORD 之后，控制器内核将自动把请求条目写入相应的请求队列。 3. 如果该事务成功，将生成 USB_XFERCOMPL 中断。如果该事务失败，则会发生错误中断，例如 USB_H_NACK*n*。	1. 软件配置 USB_DOEPTSIZ*n*_REG，指定数据包的大小 (1 MPS) 和数量（1 个）。端点使能后，将等待主机向其发送数据包。 2. 接收到的数据包将与数据包状态条目一起放入 RX FIFO。 3. 如果该事务失败，例如，由于 RX FIFO 已满，则端点将回传 NAK。
IN 事务	
1. 软件配置 USB_HCTSIZ*n*_REG，指定数据包的大小和数量（1 个），然后使能该通道。 2. 控制器内核自动将请求条目写入相应的请求队列。 3. 如果该事务成功，接收数据以及状态条目将写入 RX FIFO。否则，会产生错误中断，例如 USB_H_NACK*n*。	1. 软件配置 USB_DIEPTSIZ*n*_REG，指定数据包的大小和数量（1 个）。端点使能后，将等待主机从其读取数据包。 2. 数据包传输完成后，将产生 USB_XFERCOMPL 中断。

在 Slave 模式下进行传输级操作时，可以在队列中一次性排入一个或多个事务级操作，达到类似于 DMA 模式下的传输级操作的效果。在同一次触发的传输中，多个事务的数据包均可以从 FIFO 中读写，这样就无需以数据包

为单位触发中断。

Slave 模式下进行传输级操作的方法类似于事务级操作，不同之处在于，需要配置 USB_HCTSIZ*n*_REG 或 USB_DOEPTSIZ*n*_REG 寄存器以指定整次传输的大小和数据包个数。使能通道或端点后，应分别向 TX FIFO 或 RX FIFO 写入或读取对应于多个数据包的有效数据，假设有足够的空间或足够的数据量。

52.4.7 OTG

OTG_HS 在内部将 ID/VBUS 悬空，因此不支持 SRP/HNP 协议，但依旧支持 DRD 双角色设备属性，即既可以作为主机，也可以充当设备。软件可以设置 USB_GUSBCFG 中的 ForceDevMode 或者 ForceHstMode 强制设置控制器工作在主机模式下或是设备模式下而无视 ID 管脚的值。

52.5 寄存器

USB OTG 寄存器列表及详细描述受 IP 供应商保密协议 (NDA) 的约束。若需获取某一寄存器的支持信息，请通过 技术支持 联系乐鑫技术支持团队。

第 53 章

USB 2.0 全速 OTG

53.1 概述

ESP32-P4 带有一个集成了收发器的 USB 2.0 全速 OTG 外设，下文将称为 OTG_FS。该 OTG_FS 外设可配置成主机模式 (Host mode) 或设备模式 (Device mode)，符合 USB 2.0 协议，同时兼容 OTG 1.3 协议和 OTG 2.0 协议。支持传输速率为 12 Mbit/s 的 USB 2.0 全速模式 (Full-Speed, FS) 和传输速率为 1.5 Mbit/s 的 USB 2.0 低速模式 (Low-Speed, LS)，还支持主机协商协议 (Host Negotiation Protocol, HNP) 和会话请求协议 (Session Request Protocol, SRP)。

53.2 术语

本章使用到的缩写及术语如下：

主机 (Host)	主机是安装了 USB 主机控制器的计算机系统，包括：主机硬件平台（CPU、总线等）和使用的操作系统。
设备 (Device)	设备是执行某个功能的逻辑或物理实体。具体描述的实体取决于上下文。在最低层次上，设备可能指一个单一的硬件组件，例如存储设备。在较高层次上，设备可能指执行特定功能的一组硬件组件，例如 USB 接口设备。在更高层次上，设备可能指连接到 USB 的实体，例如数据/传真调制解调器设备。非特指时，USB 设备可以是集线器或功能设备。
全速 (Full-speed)	USB 运行在 12 Mbit/s 速率下。
低速 (Low-speed)	USB 运行在 1.5 Mbit/s 速率下。
主机协商协议 (HNP)	允许两个直接连接的 OTG 设备之间转换主机角色，无需用户切换电路连接即可改变设备之间通信的控制。
会话请求协议 (SRP)	允许 B 设备请求 A 设备打开 USB 接口 (VBUS) 的电源并启动会话。
端点 (Endpoint)	USB 设备的一个可以唯一寻址的部分，是在主机和设备之间通信流中信息的源或接收点。
Scatter/Gather DMA 模式	通过 DMA 访问不连续的存储区域。
Buffer DMA 模式	通过 DMA 访问连续存储区域。
Slave 模式	通过 CPU 访问存储区域。
RX FIFO	存储接收数据。
TX FIFO	存储发送数据。
周期性 TX FIFO	存储待发送的同步类型传输和中断类型传输的数据。
非周期性 TX FIFO	存储待发送的控制类型传输和批量类型传输的数据。

53.3 特性

53.3.1 通用特性

- 兼容 USB 2.0 协议、OTG 1.3 和 OTG 2.0 协议
- 支持主机模式和设备模式
- 支持 USB 2.0 全速和低速速率
- 主机协商协议 (HNP) 和会话请求协议 (SRP)，均可作为 A 或 B 设备
- 动态分配 FIFO (DFIFO) 大小，最大容量 1 KB
- 支持多种存储器访问模式
  - Scatter/Gather DMA 模式
  - Buffer DMA 模式
  - Slave 模式
- 集成 2 个内部收发器

53.3.2 设备模式 (Device mode) 特性

- 端点 0 永远存在，双向控制，由 EP0 IN 和 EP0 OUT 组成
- 6 个附加端点 1 ~ 6，可配置为 IN 或 OUT
- 最多 5 个 IN 端点同时工作，包括 EP0 IN
- 所有 OUT 端点共享一个 RX FIFO
- 每个 IN 端点都有专用的 TX FIFO

53.3.3 主机模式 (Host mode) 特性

- 8 个主机通道
- 一个 RX FIFO：由所有周期事务和非周期事务共用
- 两个 TX FIFO：
  - 所有非周期事务传输共用一个 TX FIFO
  - 所有周期事务传输共用另一个 TX FIFO
- 上述所有 FIFO 共用 1 KB RAM
- 每个 FIFO 大小可配置，最大 1 KB

53.4 功能描述

53.4.1 控制器内核与接口

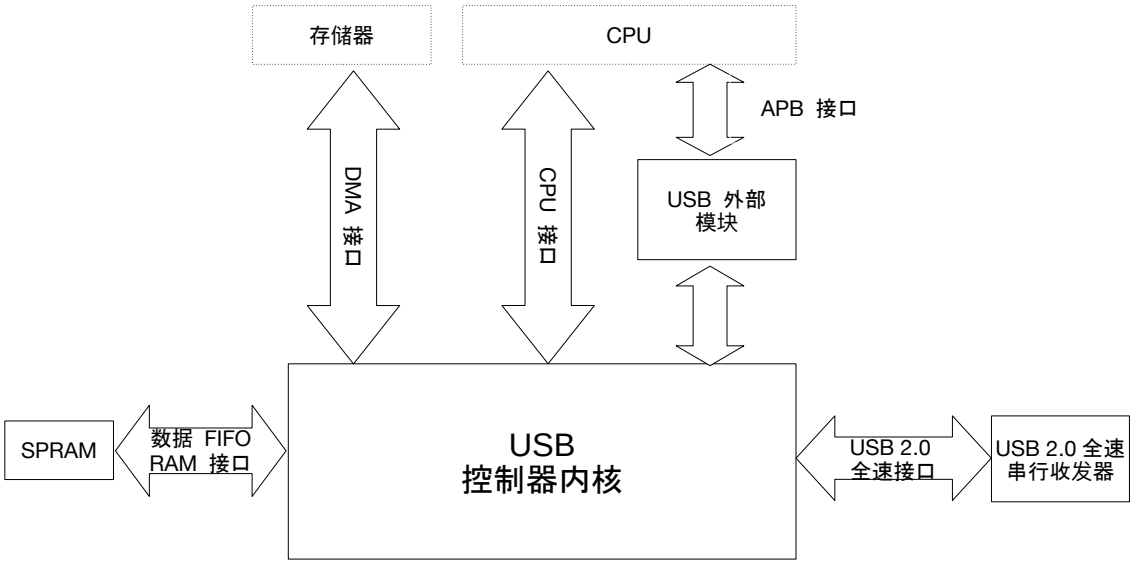


图 53.4-1. OTG_FS 系统架构

OTG_FS 外设的核心称为 USB 控制器内核。如图 53.4-1 所示，控制器内核有以下接口：

- **CPU 接口**
  CPU 可以通过该接口读写控制器内核的多个寄存器和 FIFO。该接口在内部实现为 AHB 从机接口。通过该接口访问 FIFO 的方式称为 Slave 模式。
- **APB 接口**
  CPU 可以通过 USB 外部控制模块来控制 USB 控制器内核。
- **DMA 接口**
  控制器内核的内部 DMA 可以通过该接口读写系统存储器，例如在 DMA 模式下获取和写入有效数据。该接口在内部实现为 AHB 主机接口。
- **USB 2.0 全速接口**
  控制器内核通过该接口连接 USB 2.0 全速串行收发器。ESP32-P4 包含两个全速集成收发器。除 OTG_FS 之外，ESP32-P4 还内置一个 USB 串行/JTAG 控制器，请参阅章节 *54 USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)*。这两个 USB 控制器可通过交叉矩阵选择使用任一内部集成收发器。下文将 GPIO24/25 称为 FS_PHY1，将 GPIO26/27 称为 FS_PHY2。

  默认情况下 FS_PHY1 连接 USB 串行/JTAG 控制器，FS_PHY2 连接 OTG_FS。用户可以通过 EFUSE_USB_PHY_SEL 更改连接关系：

  - 0：FS_PHY1 连接 USB 串行/JTAG 控制器，FS_PHY2 连接 OTG_FS。
  - 1：FS_PHY2 连接 USB 串行/JTAG 控制器，FS_PHY1 连接 OTG_FS。
- **USB 外部控制模块**
  USB 外部控制模块主要用于将 USB 2.0 全速串行接口连接到内部寄存器，从而允许 CPU 进行查询操作等。USB 外部控制模块还能实现省电模式，具体做法是控制器内核时钟，即 AHB 时钟，采用门控时钟，或关闭所连接的 SPRAM 时钟。需要注意的是此节能模式与通过 SRP 实现的节能方式有所不同。
- **数据 FIFO RAM 接口**

控制器内核使用的多个 FIFO 实际上并不位于控制器内核内部，而是位于单端口 RAM (SPRAM) 上。FIFO 的大小可动态配置，运行时在 SPRAM 中进行分配。CPU、DMA 或控制器通过该接口读写 FIFO。

53.4.2 存储器布局

图 53.4-2 显示了用于配置和控制 USB 控制器内核的寄存器布局。请注意，USB 外部控制模块使用另外一组寄存器，称为 wrap 寄存器。

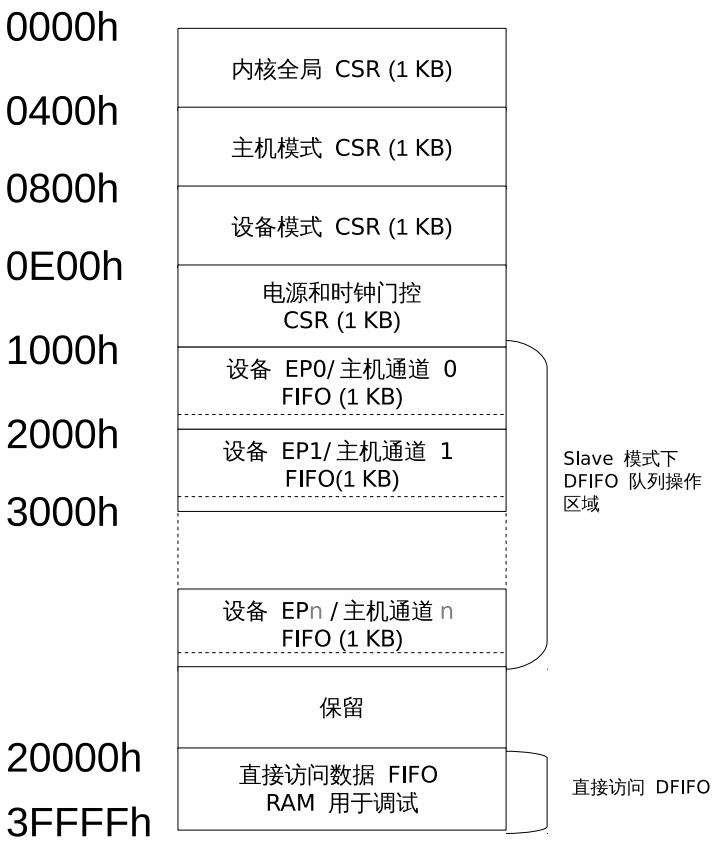


图 53.4-2. 内核地址映射

53.4.2.1 控制 & 状态寄存器 (CSR)

- **内核全局 CSR**
  内核全局寄存器用于配置/控制 OTG_FS 的通用功能，即主机和设备模式共同的功能，并表示其内部状态。通用功能包括：
  - OTG 控制：HNP、SRP 和 A/B 设备检测；
  - USB 配置：选择主机或设备模式，PHY 选择；
  - 系统级中断。

  在主机和设备模式下，软件均可以对内核全局 CSR 进行访问。
- **主机模式 CSR**
  主机模式寄存器用于主机模式下的配置/控制/状态表示，只能在主机模式下被访问。每个通道各自有一组主机模式寄存器。
- **设备模式 CSR**

设备模式寄存器用于设备模式下的配置/控制/状态表示，只能在设备模式下被访问。每个端点各自有一组设备模式寄存器。

- **电源和时钟门控寄存器**
  此单一寄存器用于控制模块电源和门控时钟。

53.4.2.2 FIFO 访问

OTG_FS 利用多个 FIFO 缓冲发送或接收的有效数据。FIFO 的数量和类型取决于主机或设备模式，以及所使用的通道或端点的数量，参考章节 53.4.3。FIFO 访问有两种方式：DMA 模式和 Slave 模式。当使用 Slave 模式时，CPU 需要通过读写 DFIFO 的推入/弹出操作 (push/pop) 区域或读写调试区域来访问这些 FIFO。FIFO 访问遵循以下规则：

- 对 1 KB 推入/弹出区域中任何地址的读访问将在共享 RX FIFO 中产生一个弹出 (pop) 操作。
- 对特定 1 KB 推入/弹出区域的写访问将写入相应的端点或通道的 TX FIFO。前提是该端点是 IN 端点，或者该通道是 OUT 通道。
  - 在设备模式下，数据写入相应的 IN 端点的专用 TX FIFO。
  - 在主机模式下，根据通道是非周期性通道还是周期性通道，数据写入非周期性 TX FIFO 或周期性 TX FIFO。
- 访问 128 KB 读写调试区域将直接读/写，而不是进行推入/弹出操作。此种访问通常仅用于调试目的。

请注意，仅在 Slave 模式下，才需要由 CPU 直接向 FIFO 进行数据操作。在 DMA 模式下，内部 DMA 将处理 TX FIFO 和 RX FIFO 的数据推入/弹出操作。

53.4.3 FIFO 和队列组织

OTG_FS 中的 FIFO 主要用于保存有效数据，即 USB 数据包的数据字段。TX FIFO 用于存储将由主机模式下的 OUT 事务或设备模式下的 IN 事务发送的有效数据。RX FIFO 用于存储主机模式下 IN 事务或设备模式下 OUT 事务的已接收的有效数据。除了存储有效数据之外，RX FIFO 还存储每个有效数据的**状态条目**。状态条目中包含关于有效数据的信息，如：通道编号、字节数、是否有效等。在 Slave 模式下，状态条目还用于指示各类通道事件。

可用于 FIFO 分配的 SPRAM 大小为 256 × 35 位。35 位包括 32 个数据位加 3 个控制位。主机模式下各个通道或设备模式下端点使用的多个 FIFO 被分配到 SPRAM 中，并且可以动态调整大小。

53.4.3.1 主机模式 FIFO 和队列

如图 53.4-3 所示，主机模式使用以下 FIFO：

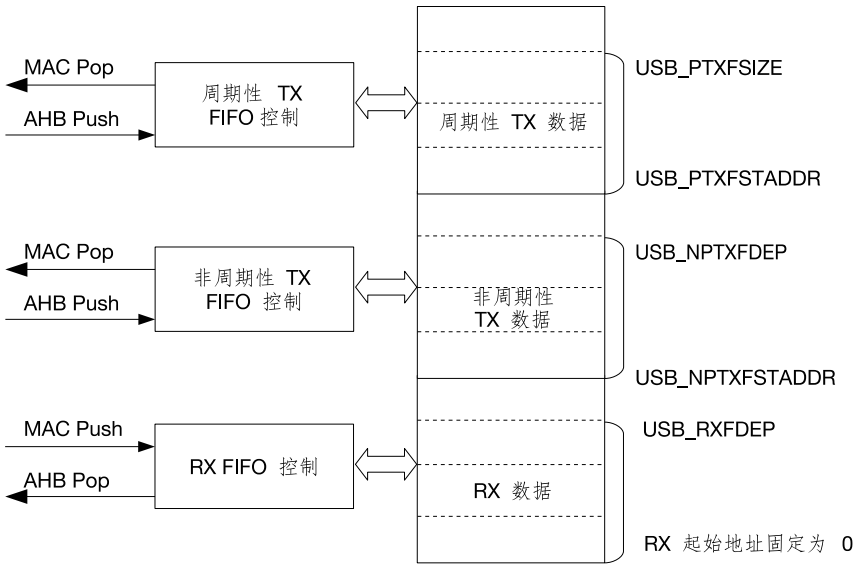


图 53.4-3. 主机模式 FIFO

- **非周期性 TX FIFO**：存储所有通道的批量和控制类型的 OUT 事务的有效数据。
- **周期性 TX FIFO**：存储所有通道的中断或同步类型的 OUT 事务的有效数据。
- **RX FIFO**：存储所有 IN 事务的有效数据，以及用于指示有效数据大小和事务/通道事件的状态条目。例如传输完成或通道暂停。

除 FIFO 外，主机模式还包含两个请求队列，用于存储来自多个通道的事务请求。请求队列中的每个条目都包含 IN/OUT 通道编号以及执行事务的其他信息，例如事务类型。请求队列也用于存储其他请求类型，例如通道暂停请求。

与 FIFO 不同，请求队列的大小是固定的，且不能由软件直接访问。相反，一旦启用了通道，主机内核会自动将请求写入请求队列。请求写入队列的顺序决定了 USB 事务的处理顺序。

主机模式包含以下请求队列：

- **非周期性请求队列**：针对非周期性批量和控制事务的请求队列，最多可以存储 4 个条目。
- **周期性请求队列**：针对周期性中断和同步事务的请求队列，最多可以存储 8 个条目。

调度事务时，硬件将首先执行周期性请求队列上的所有请求，再执行非周期性请求队列上的请求。

53.4.3.2 设备模式 FIFO

如图 53.4-4 所示，设备模式使用以下 FIFO：

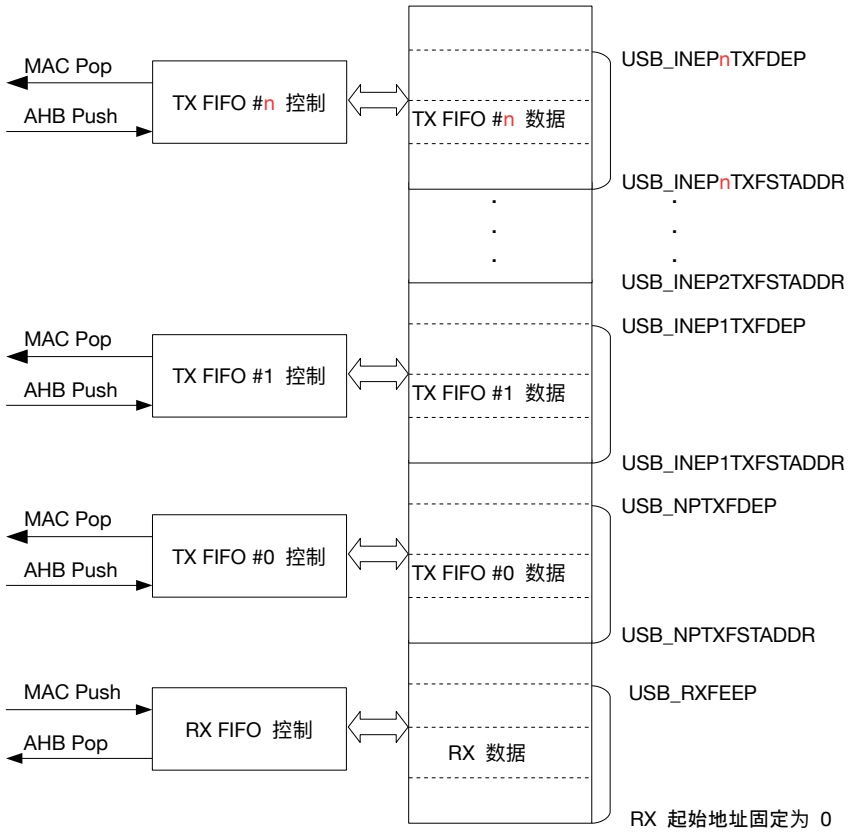


图 53.4-4. 设备模式 FIFO

- **RX FIFO**：存储数据包内接收的有效数据和状态条目，用于指示有效数据的大小。
- **专用 TX FIFO**：每个使能的 IN 端点都有一个专用的 TX FIFO，用于存储该端点的所有 IN 有效数据，而无论事务类型，包括周期性或非周期性 IN 事务。

由于有专用的 TX FIFO，设备模式不使用任何请求队列。IN 事务的顺序由主机确定。

53.4.4 中断层次结构

OTG_FS 有一条中断线，可以通过中断矩阵连接到一个 CPU。可以通过置位 USB_GLBLINTRMSK 来使能中断信号。OTG_FS 中断是 USB_GINTSTS_REG 寄存器中所有位的逻辑或 (OR)，且置位 USB_GINTMSK_REG 寄存器中的相应位可以使能 USB_GINTSTS_REG 中的位。USB_GINTSTS_REG 包含系统级中断，还包含主机或设备模式专有的中断位以及 OTG 有关中断。OTG_FS 中断源的层次结构如图 53.4-5 所示。

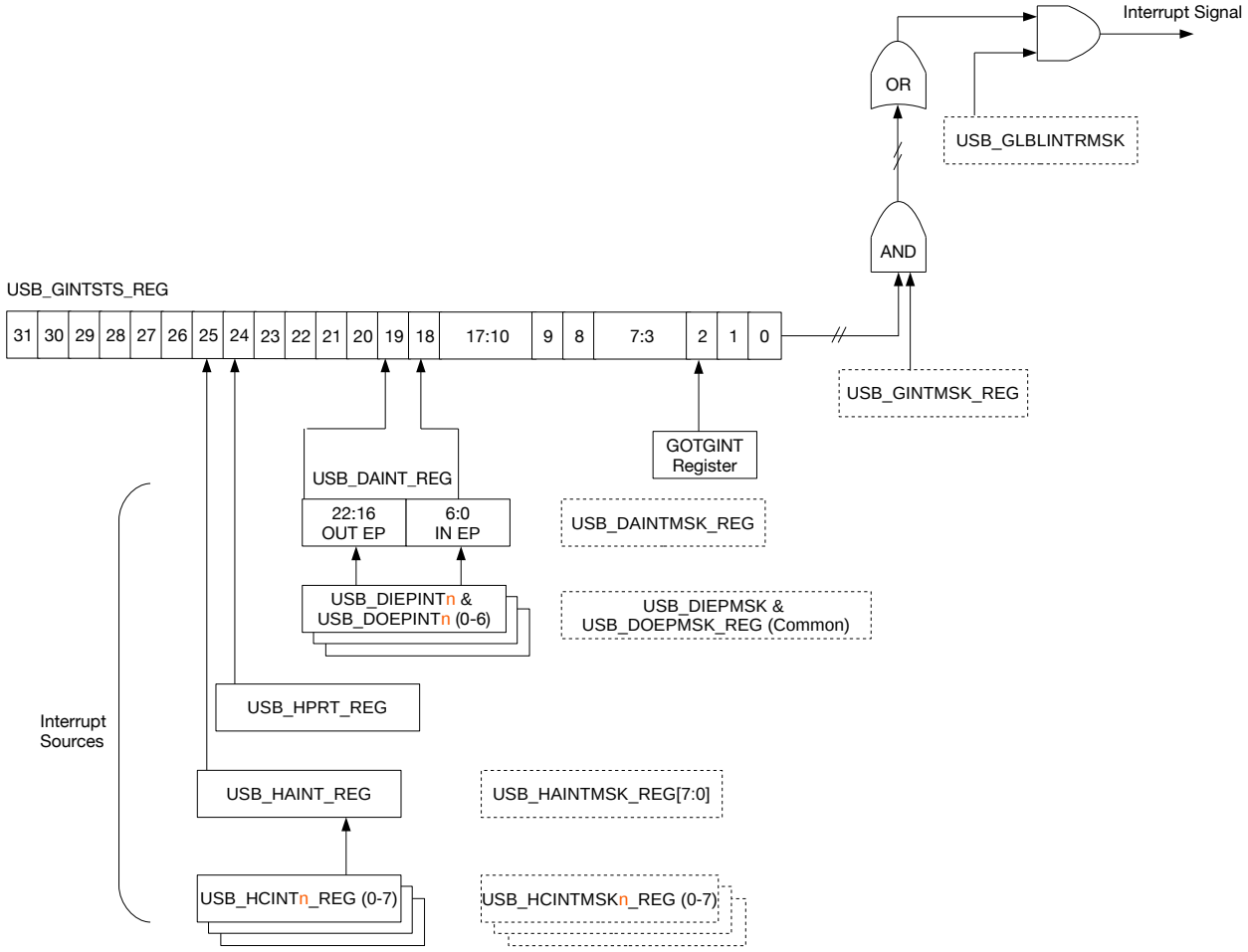


图 53.4-5. OTG_FS 中断层次结构图

USB_GINTSTS_REG 寄存器的以下位指示较低层级的中断源：

- **USB_PRTINT** 表示主机端口有未处理的中断。USB_HPRT_REG 寄存器指示中断源。
- **USB_HCHINT** 表示一个或多个主机通道有未处理的中断。通过读取 USB_HAINT_REG 寄存器可以确定哪些通道有未处理的中断，然后查询该通道的 USB_HCINT*n*_REG 寄存器以确定中断源。
- **USB_OEPINT** 表示一个或多个 OUT 端点有未处理的中断。通过读取 USB_DAINT_REG 寄存器可以确定哪些 OUT 端点有未处理的中断，然后查询 OUT 端点的 USB_DOEPINT*n*_REG 寄存器以确定中断源。
- **USB_IEPINT** 表示一个或多个 IN 端点有未处理的中断。通过读取 USB_DAINT_REG 寄存器可以确定哪些 IN 端点有未处理的中断，然后查询 IN 端点的 USB_DIEPINT*n*_REG 寄存器以确定中断源。
- **USB_OTGINT** 表示 OTG 事件已触发中断。查询 USB_GOTGINT_REG 寄存器以确定哪些 OTG 事件触发了中断。

53.4.5 Slave 模式和 DMA 模式

OTG_FS 支持 3 种存储器访问方式：Slave 模式、Buffer DMA 模式、和 Scatter/Gather DMA 模式。

53.4.5.1 Slave 模式

在 Slave 模式下，所有有效数据放入 FIFO 或从 FIFO 中取出都必须通过 CPU 进行。

- 使用 IN 端点或 OUT 通道传输数据包时，必须将有效数据放入相应的端点或通道的 TX FIFO 中。
- 接收到数据包时，必须先通过读取 USB_GRXSTSP_REG 从 RX FIFO 中取出数据包的状态条目，以确定数据包中有效数据的长度，以字节为单位。然后必须由 CPU 通过读取 RX FIFO 中的存储区域从 RX FIFO 中取出相应的字节数。

53.4.5.2 Buffer DMA 模式

Buffer DMA 模式类似于 Slave 模式，不同之处在于该模式为利用内部 DMA 将有效数据放入 FIFO 或从 FIFO 中取出。

- 使用 IN 端点或 OUT 通道发送数据包时，应将有效数据的存储地址写入 USB_HCDMA*n*_REG（主机模式）或 USB_DIEPDMA*n*_REG（设备模式）。启用端点或通道后，内部 DMA 会将有效数据从存储器中推送到通道或端点的 TX FIFO 中。
- 使用 OUT 端点或 IN 通道接收数据包时，应将存储器中空 buffer 的地址写入 USB_HCDMA*n*_REG（主机模式）或 USB_DOEPDMA*n*_REG（设备模式）。启用端点或通道后，内部 DMA 将把有效数据从 RX FIFO 弹出到相应 buffer 中。

53.4.5.3 Scatter/Gather DMA 模式

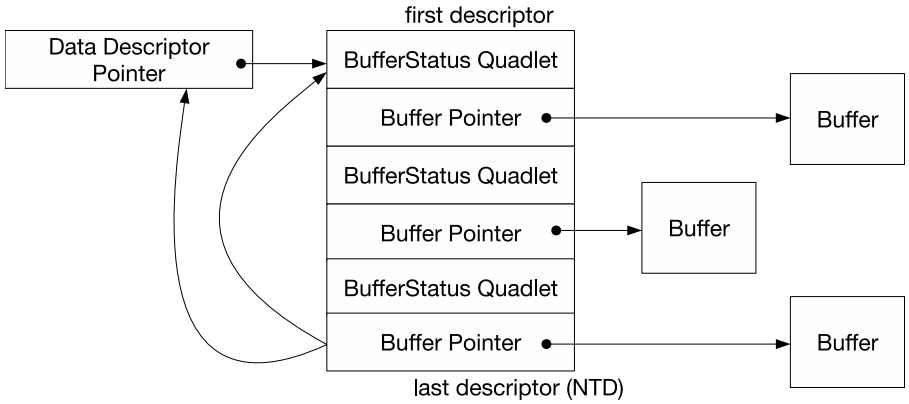


图 53.4-6. Scatter/Gather DMA 链表结构

在 Scatter/Gather DMA 模式下，包含发送或接收数据的 buffer 可能分散在整个存储器各处。每个端点或通道都有一个连续的 DMA 描述符列表。每个描述符包含一个指向发送或接收数据 buffer 的 32 位指针和一个 32 位的 buffer 描述符 (BufferStatus Quadlet)。发送和接收数据 buffer 可以对应单个事务，即 < 1 MPS 字节，MPS：最大数据包大小，或整个传输，即 > 1 MPS 字节。该列表的实现为环形 buffer，意味着 DMA 在遇到列表中的最后一个条目时将返回至第一个条目。

- 当使用 IN 端点或 OUT 通道发送传输/事务时，DMA 将从多个 buffer 收集有效数据并将其放入 TX FIFO。
- 当使用 OUT 端点或 IN 通道接收传输/事务时，DMA 将从 RX FIFO 取出有效数据，并将它们分别存放到 DMA 列表条目所指向的多个 buffer 中。

53.4.6 事务和传输级操作

在主机或设备模式下，通信可以在事务级或传输级进行。

53.4.6.1 DMA 模式下的事务和传输级操作

在 DMA 模式下的事务级操作，传输大小是一个数据包的大小，即最大数据包大小或短数据包大小。

在 DMA 主机模式下的传输级操作，只有当一个传输通道被终止时，中断才会发生。以下三种情况下发生传输通道终止：传输通道中所有要求传送的数据全部成功传送、接收到 STALL 或出现连续事务级错误（如，3 个连续的事务级错误）。在 DMA 设备模式下操作时，所有错误均由控制器内核进行处理。

53.4.6.2 Slave 模式下的事务和传输级操作

在 Slave 模式下的事务级操作，一次只能处理一个事务。每组有效数据应对应一个数据包，软件必须根据 USB 接收到的握手应答，例如 ACK 或 NAK，来确定是否需要重启事务。

下表描述了 Slave 模式下进行 IN 和 OUT 事务级操作的方法。

表 53.4-1. Slave 模式下的 IN 和 OUT 事务级操作

主机模式	设备模式
OUT 事务	
1. 软件配置 USB_HCTSIZ*n*_REG，指定数据包的大小和数量（1 个），使能该通道，然后将数据包的有效数据复制到 TX FIFO 中。 2. 软件在写完每个数据包的最后一个 DWORD 之后，控制器内核将自动把请求条目写入相应的请求队列。 3. 如果该事务成功，将生成 USB_XFERCOMPL 中断。如果该事务失败，则会发生错误中断，例如 USB_H_NACK*n*。	1. 软件配置 USB_DOEPTSIZ*n*_REG，指定数据包的大小 (1 MPS) 和数量（1 个）。端点使能后，将等待主机向其发送数据包。 2. 接收到的数据包将与数据包状态条目一起放入 RX FIFO。 3. 如果该事务失败，例如，由于 RX FIFO 已满，则端点将回传 NAK。
IN 事务	
1. 软件配置 USB_HCTSIZ*n*_REG，指定数据包的大小和数量（1 个），然后使能该通道。 2. 控制器内核自动将请求条目写入相应的请求队列。 3. 如果该事务成功，接收数据以及状态条目将写入 RX FIFO。否则，会产生错误中断，例如 USB_H_NACK*n*。	1. 软件配置 USB_DIEPTSIZ*n*_REG，指定数据包的大小和数量（1 个）。端点使能后，将等待主机从其读取数据包。 2. 数据包传输完成后，将产生 USB_XFERCOMPL 中断。

在 Slave 模式下进行传输级操作时，可以在队列中一次性排入一个或多个事务级操作，达到类似于 DMA 模式下的传输级操作的效果。在同一次触发的传输中，多个事务的数据包均可以从 FIFO 中读写，这样就无需以数据包为单位触发中断。

Slave 模式下进行传输级操作的方法类似于事务级操作，不同之处在于，需要配置 USB_HCTSIZ*n*_REG 或 USB_DOEPTSIZ*n*_REG 寄存器以指定整次传输的大小和数据包个数。使能通道或端点后，应分别向 TX FIFO 或 RX FIFO 写入或读取对应于多个数据包的有效数据，假设有足够的空间或足够的数据量。

53.5 OTG

USB OTG 允许 OTG 设备作为 USB 主机或 USB 设备。因此，OTG 设备上一般都有一个 Mini-AB 或 Micro-AB 接口，可用于连接 A-plug 或 B-plug。当 OTG 设备连接上 A-plug/B-plug 时，其将成为 A 设备/B 设备。

- A 设备默认为主机模式（A 主机），B 设备默认为设备模式（B 外设）。
- 通过使用主机协商协议 (NHP)，A、B 设备可互相交换角色，即变更为 A 外设和 B 主机。
- A 设备可关闭 Vbus 省电。然后，B 设备可通过请求 A 设备启动 Vbus 并发起一个新的会话来唤醒 A 设备。该机制称为会话请求协议 (SRP)。
- Vbus 只能由 A 设备供电，即使 A 设备为外设模式。

OTG 设备可通过接头的 ID 管脚确定其连接的是 A-plug 还是 B-plug。A-plug 中的 ID 管脚为接地，B-plug 中的 ID 管脚则为悬空。

53.5.1 OTG 接口

OTG_FS 支持 OTG 1.3 规范的 SRP 和 HNP 协议。OTG_FS 控制器内核通过 UTMI+ OTG 接口与内部收发器连接。UTMI+ OTG 接口允许控制器内核操作收发器，比如启用/禁用 HNP 中的上拉和下拉，以实现 OTG 的功能，并且还允许收发器指示与 OTG 相关的事件。如果改用外部收发器，那么 UTMI+ OTG 将通过 GPIO 交换矩阵连接到 ESP32-P4 的 GPIO，请参阅章节 10 *GPIO 交换矩阵和 IO MUX*。表 53.5-1 描述了 UTMI+ OTG 接口信号。

表 53.5-1. UTMI+ OTG 接口

接口信号	I/O	描述
usb_otg_iddig_in	I	迷你 A/B 插头指示器。指示所连接的插头是 mini-A 还是 mini-B。仅在 usb_otg_idpullup 被采样断言时有效。 0：连接 mini-A 1：连接 mini-B
usb_otg_avalid_in	I	A 类外设会话有效。指示 Vbus 电压是否在 A 类外设会话的有效电平上。比较器阈值为： 0: Vbus <0.8 V 1: Vbus = 0.2 V ~ 2.0 V
usb_otg_bvalid_in	I	B 类外设会话有效。指示 Vbus 电压是否在 B 类外设会话的有效电平上。比较器阈值为： 0: Vbus <0.8 V 1: Vbus = 0.8 V ~ 4 V
usb_otg_vbusvalid_in	I	Vbus 有效。指示 Vbus 电压是否在 A/B 设备/外设操作的有效电平上。比较器阈值为： 0: Vbus <4.4 V 1: Vbus >4.75 V

接口信号	I/O	描述
usb_srp_sessend_in	I	B 设备会话结束。指示 Vbus 电压是否在 B 设备会话结束的阈值以下。比较器阈值为： 0: Vbus >0.8 V 1: Vbus <0.2 V
usb_otg_idpullup	O	模拟 ID 输入采样使能。使能采样模拟 ID 线。 0：ID 管脚采样禁能 1：ID 管脚采样使能
usb_otg_dppulldown	O	D+ 下拉电阻使能。使能 D+ 线上的 15 kΩ 下拉电阻。
usb_otg_dmpulldown	O	D- 下拉电阻使能。使能 D- 线上的 15 kΩ 下拉电阻。
usb_otg_drvvbus	O	驱动 Vbus。驱动 Vbus 到 5 V。 0：不驱动 Vbus 1：驱动 Vbus
usb_srp_chrgvbus	O	Vbus 输入充电使能。指示 PHY 为 Vbus 充电。 0：不通过电阻为 Vbus 充电 1：通过电阻为 Vbus 充电（需激活至少 30 ms）
usb_srp_dischrgvbus	O	Vbus 输入放电使能。指示 PHY 为 Vbus 放电。 0：不通过电阻为 Vbus 放电 1：通过电阻为 Vbus 放电（需激活至少 50 ms）

53.5.2 ID 管脚检测

寄存器 USB_GOTGCTL_REG 中的 USB_CONIDSTS 位指示 OTG 控制器为 A 设备 (1'b0) 还是 B 设备 (1'b1)。当 USB_CONIDSTS 发生改变，即连接或断开插头时，会产生 USB_CONIDSTSCHNG 中断。

53.5.3 会话请求协议 (SRP)

53.5.3.1 A 设备 SRP

图 53.5-1 说明了 OTG_FS 充当 A 设备，即默认主机并为 Vbus 供电时的 SRP 流程。

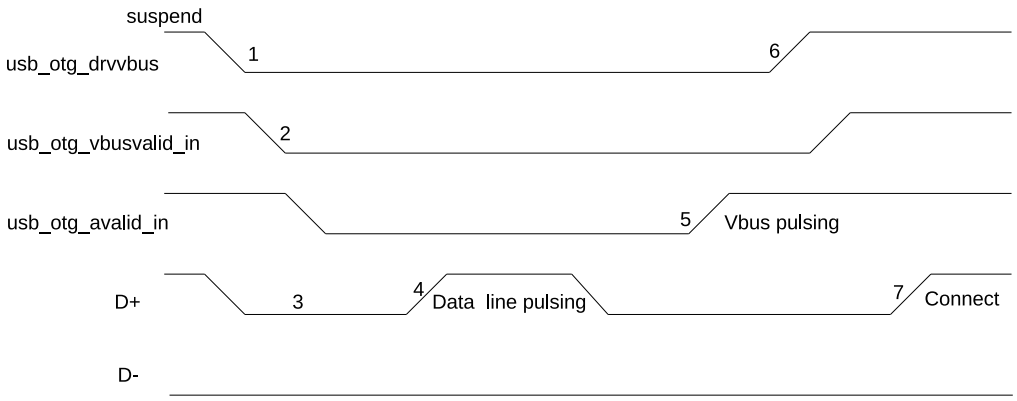


图 53.5-1. A 设备 SRP

1. 为了节省电能，当总线空闲时，应用程序将挂起并关闭端口电源，方法是写入主机端口控制和状态寄存器中的端口挂起位（USB_PRTSUSP 置为 1'b0）和端口电源位（USB_PRTPWR 置为 1'b0）。
2. PHY 通过使 usb_otg_vbusvalid_in 信号无效来指示端口断电。

3. 当 Vbus 电源关闭时，A 设备必须检测到 SEO 至少 2 ms 才能启动 SRP。
4. 要启动 SRP，B 设备会打开其数据线上拉电阻 5 到 10 ms。OTG_FS 内核将检测数据线脉冲。
5. 设备将 Vbus 驱动到 A 设备会话有效阈值之上（至少 2.0 V）以执行 Vbus 脉冲。OTG_FS 内核在检测 SRP 时中断应用程序。全局中断状态寄存器中的会话请求检测位 (USB_SESSREQINT) 将被置位。
6. 应用程序必须处理会话请求检测中断，并通过写入主机端口控制和状态寄存器中的端口电源位来打开端口电源位。PHY 通过确认 usb_otg_vbusvalid_in 信号来指示端口上电。
7. 当 USB 上电时，B 设备连接，完成 SRP 过程。

53.5.3.2 B 设备 SRP

图 53.5-2 说明了 OTG_FS 充当 B 设备，即不为 Vbus 供电时的 SRP 流程。

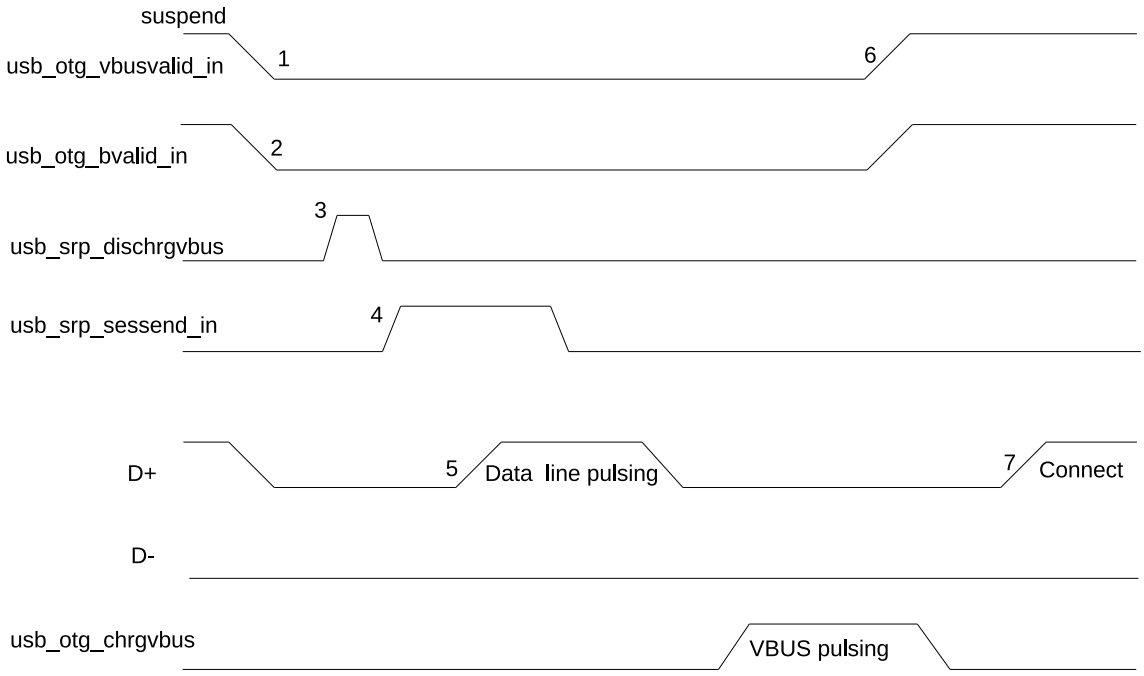


图 53.5-2. B 设备 SRP

1. 为了节省电能，当总线空闲时，主机（A 设备）将挂起并关闭端口电源。PHY 通过使 usb_otg_vbusvalid_in 信号无效来指示端口掉电。在检测到 3 ms 总线空闲后，OTG_FS 内核将内核中断寄存器中的早期挂起位 (USB_ERLYSUSP) 置 1。之后，OTG_FS 内核将内核中断寄存器中的 USB 挂起位 (USB_USBSUSP) 置 1。PHY 通过使 usb_otg_bvalid_in 信号无效来指示 B 设备会话结束。
2. OTG_FS 内核确认 usb_otg_dischrgvbus 信号，指示 PHY 加快 Vbus 放电。
3. PHY 通过确认 usb_otg_sessend_in 信号来指示会话结束。这是 SRP 的初始条件。OTG_FS 内核在启动 SRP 之前需要检测到 SEO 2 ms。对于 USB 2.0 全速串行收发器，在 USB_BSESVLD 无效后，应用程序必须等待 Vbus 放电至 0.2 V。
4. 应用程序等待 1.5 秒（TB_SEO_SRP 时间），然后写入 OTG 控制和状态寄存器中的会话请求位 (USB_SESREQ) 并启动 SRP。OTG_FS 内核执行数据线脉冲，然后执行 Vbus 脉冲。
5. 主机（A 设备）从数据线或 Vbus 脉冲检测到 SRP，然后打开 Vbus。PHY 确认 usb_otg_vbusvalid_in 信号指示 Vbus 上电。

6. OTG_FS 内核确认 usb_srp_chrgvbus 并执行 Vbus 脉冲。主机（A 设备）打开 Vbus，启动新会话，指示 SRP 成功。OTG_FS 内核通过置位 OTG 中断状态寄存器中的会话请求成功状态改变位 (USB_SESREQSC) 来中断应用程序。应用程序读取 OTG 控制和状态寄存器中的会话请求成功位。
7. 当 USB 通电时，OTG_FS 内核连接，从而完成 SRP 过程。

53.5.4 主机协商协议 (HNP)

53.5.4.1 A 设备 HNP

图 53.5-3 说明了 OTG_FS 充当 A 设备时的 HNP 流程。

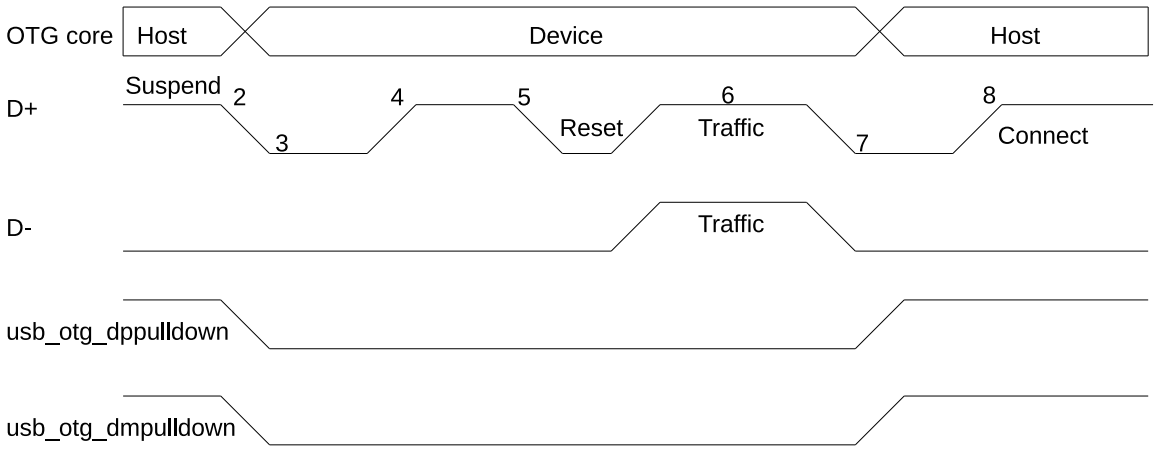


图 53.5-3. A 设备 HNP

1. OTG_FS 内核向 B 设备发送 SetFeature b_hnp_enable 描述符以启用 HNP 支持。B 设备回复 ACK 则表明其支持 HNP。应用程序必须置位 OTG 控制和状态寄存器中的主机设置 HNP 使能位 (USB_HSTSETHNPEN) 向 OTG_FS 内核说明 B 设备支持 HNP。
2. 使用完总线后，应用程序写入主机端口控制和状态寄存器中的端口挂起位 (USB_PRTSUSP) 进入挂起状态。
3. 当 B 设备观察到 USB 挂起时，它将断开连接，表明 HNP 的初始状态。B 设备仅在必须切换到主机角色时才启动 HNP；否则，总线将继续挂起。OTG_FS 内核在 OTG 中断状态寄存器中置位主机协商中断位 (USB_HSTNEGDET)，指示 HNP 的开始。OTG_FS 内核将 usb_otg_dppulldown 和 usb_otg_dmpulldown 信号置为无效，指示设备角色。PHY 启用 D+ 上拉电阻，指示 B 设备的连接。应用程序必须读取 OTG 控制和状态寄存器中的当前模式位 (USB_CURMOD_INT) 以确定设备模式。
4. B 设备检测到连接，发出 USB 复位，对 OTG_FS 内核进行枚举以开始数据通信。
5. B 设备继续充当主机角色，启动数据通信，并在完成后挂起总线。OTG_FS 内核在检测到 3 ms 总线空闲之后，将内核中断寄存器中的早期挂起位 (USB_ERLYSUSP) 置 1。之后，OTG_FS 内核将内核中断寄存器中的 USB 挂起位 (USB_USBSUSP) 置 1。
6. 在协商模式下，OTG_FS 内核检测到挂起，断开连接并切换回主机角色。OTG_FS 内核确认 usb_otg_dppulldown 和 usb_otg_dmpulldown 信号，以表明其承担了主机角色。
7. OTG_FS 内核将 OTG 中断状态寄存器中的连接器 ID 状态改变位 (USB_CONIDSTS) 置位。应用程序必须读取 OTG 控制和状态寄存器中的连接器 ID 状态，以确定 OTG_FS 内核作为 A 设备。这表明该应用程序已完成 HNP。应用程序必须读取 OTG 控制和状态寄存器中的当前模式位 (USB_CURMOD_INT)，以确定主机模式操作。

8. B 设备连接，完成 HNP 过程。

53.5.4.2 B 设备 HNP

图 53.5-4 说明了 OTG_FS 充当 B 设备时的 HNP 流程。

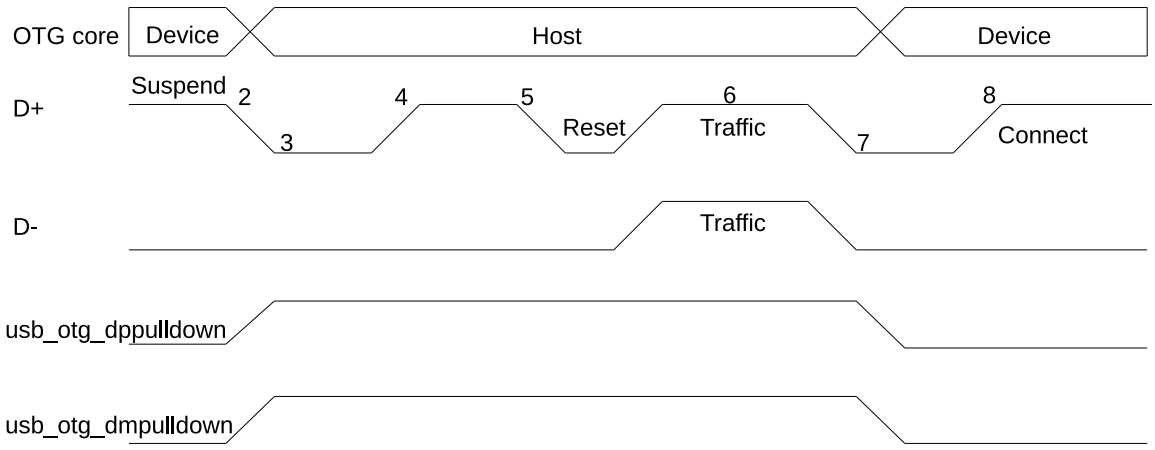


图 53.5-4. B 设备 HNP

1. A 设备发送 SetFeature b_hnp_enable 描述符以启用 HNP 支持。OTG_FS 内核回复 ACK 响应以表明其支持 HNP。应用程序必须将 OTG 控制和状态寄存器中的设备 HNP 使能位 (USB_DEVHNPEN) 置 1，以表明支持 HNP。应用程序将 OTG 控制和状态寄存器中的 HNP 请求位 (USB_DEVHNPEN) 置 1，以指示 OTG_FS 内核启动 HNP。
2. A 设备使用完总线后，将挂起总线。
   (a) OTG_FS 内核在总线空闲 3 ms 之后将内核中断寄存器中的早期挂起位 (USB_ERLYSUSP) 置 1。之后，OTG_FS 内核将内核中断寄存器中的 USB 挂起位 (USB_USBSUSP) 置 1。OTG_FS 内核断开连接，并且 A 设备检测到总线上的 SEO，指示 HNP。
   (b) OTG_FS 内核确认 usb_otg_dppulldown 和 usb_otg_dmpulldown 信号，以表明其承担了主机角色。
   (c) A 设备通过在检测到 SEO 的 3 ms 内激活 D+ 上拉电阻来做出响应。OTG_FS 内核将检测到连接。
   (d) OTG_FS 内核将 OTG 中断状态寄存器中的主机协商成功状态改变位 (USB_CONIDSTS) 置 1，以指示 HNP 状态。应用程序必须读取 OTG 控制和状态寄存器中的主机协商成功位 (USB_HSTNEGSCS)，才能确定主机协商成功。应用程序必须读取内核中断寄存器中的当前模式位 (USB_CURMOD_INT)，才能确定主机模式操作。
3. 将 USB_PRTPWR 位设置为 1'b1，以驱动 USB 上的 Vbus。
4. 等待 USB_PRTCONNDET 中断。这表明设备已连接到端口。
5. 应用程序将重置位 (USB_PRTRST) 置为 1，OTG_FS 内核将发出 USB 重置，对 A 设备进行枚举并开始数据通信。
6. 等待 USB_PRTENCHNG 中断。
7. OTG_FS 内核继续充当启动数据通信的主机角色，完成后，通过写入主机端口控制和状态寄存器中的端口挂起位 (USB_PRTSUSP) 将总线挂起。
8. 在协商模式下，当 A 设备检测到挂起时，它将断开连接并切换回主机角色。OTG_FS 内核将 usb_otg_dppulldown 和 usb_otg_dmpulldown 信号置为无效，以指示承担设备角色。

9. 应用程序必须读取内核中断寄存器中的当前模式位 (USB_CURMOD_INT)，以确定主机模式操作。
10. OTG_FS 内核连接，完成 HNP 过程。

53.6 寄存器

USB OTG 寄存器列表及详细描述受 IP 供应商保密协议 (NDA) 的约束。若需获取某一寄存器的支持信息，请通过 技术支持 联系乐鑫技术支持团队。

第 54 章

USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)

ESP32-P4 中包含一个 USB 串口/JTAG 控制器，可用于烧录芯片的外部 flash、读取程序输出的数据以及将调试器连接到正在运行的程序中。任何带有 USB 主机（下文将简称为“主机”）的计算机都可以实现上述功能，无需其他外部组件辅助。

54.1 概述

使用串口通信和 JTAG 调试端口通信来编程和调试 ESP32-P4 项目都是可行的，但也有其限制。首先，串口通信和 JTAG 调试端口通信都占用 IO 管脚，减少了软件中可用于控制外部信号的管脚数量。此外，串口通信和 JTAG 调试端口通信都需借助外部芯片或适配器与主机连接，因此，如果需要整合这两个功能，则需额外使用外部芯片或调试适配器。

为解决上述问题，ESP32-P4 提供了一个 USB 串口/JTAG 控制器，同时集成 USB-串口转换器和 USB-JTAG 适配器功能。由于该模块仅使用 USB 2.0 所需的两条数据线直接连接外部 USB 主机，因此 ESP32-P4 仅需占用两个管脚用于调试。

54.2 特性

USB 串口/JTAG 控制器具有如下特性：

- 支持 USB 全速标准；集成 CDC-ACM（通信设备类抽象控制模型）和 JTAG 适配器固定功能
- CDC-ACM：
  - 配置虚拟串行功能，在大多数现代操作系统上可实现即插即用
  - 支持主机控制芯片复位和进入下载模式
- JTAG 适配器：
  - 支持使用紧凑的 JTAG 指令实现与 CPU 调试内核的快速通信
- 共配置一个控制端点、一个虚拟中断端点、两个批量输入端点和两个批量输出端点，可实现最大 64 字节的数据载荷
- 集成内部 PHY：基本无需其他外部组件连接主机计算机

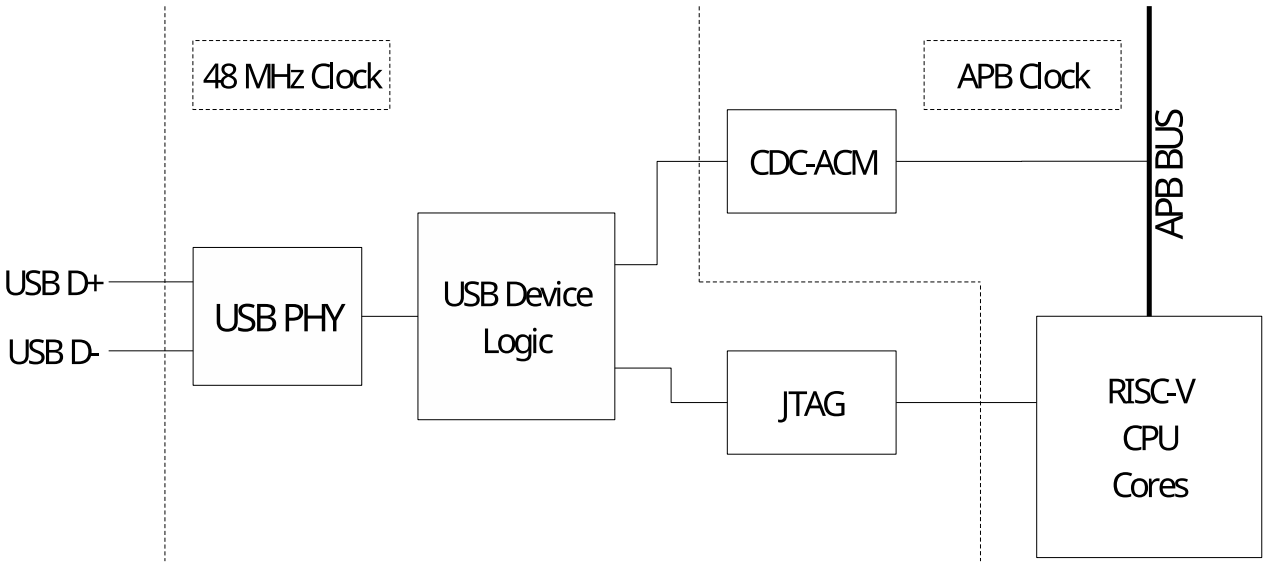


图 54.2-1. USB 串口/JTAG 控制器高层框图

如图 54.2-1 所示，USB 串口/JTAG 控制器包含 USB PHY、USB 设备接口、JTAG 命令处理器、响应捕捉单元以及若干个 CDC-ACM 寄存器。PHY 和设备接口使用主 PLL (BBPLL) 时钟产生的 48 MHz 时钟作为时钟源，CDC-ACM 块中软件可访问的部分则使用 APB_CLK 作为时钟源。JTAG 命令处理器与 ESP32-P4 主处理器中的 JTAG 调试单元相连；CDC-ACM 寄存器则连接至 APB 总线，主 CPU 上运行的软件可对其进行读写访问。

请注意，USB 串口/JTAG 控制器仅支持 USB 2.0 全速标准 (12 Mbps)，不支持 USB 2.0 标准的其他模式（如 480 Mbps 的高速模式）。

图 54.2-2 展示了 USB 串口/JTAG 控制器中 USB 部分的内部详细结构。USB 串口/JTAG 控制器由一个 USB 2.0 全速设备组成，包含一个控制端点、一个虚拟中断端点、两个批量输入端点和两个批量输出端点。这些端点共同组成该复合型 USB 设备，具体可分为 CDC-ACM USB 设备和实现 JTAG 接口功能的供应商特定设备。JTAG 接口直接与芯片的 RISC-V CPU 的调试接口相连，可对运行在该 CPU 上的程序进行调试。同时，CDC-ACM 中包含一组寄存器，CPU 上运行的程序可对其进行读写操作。此外，芯片上的 ROM 启动代码可允许用户通过使用该接口重新烧录 flash。

说明：

如需确定某个功能使用的具体端点，请解析设备返回的相关 USB 描述符。这样可以保证与功能相同但实现方式不同的设备兼容。

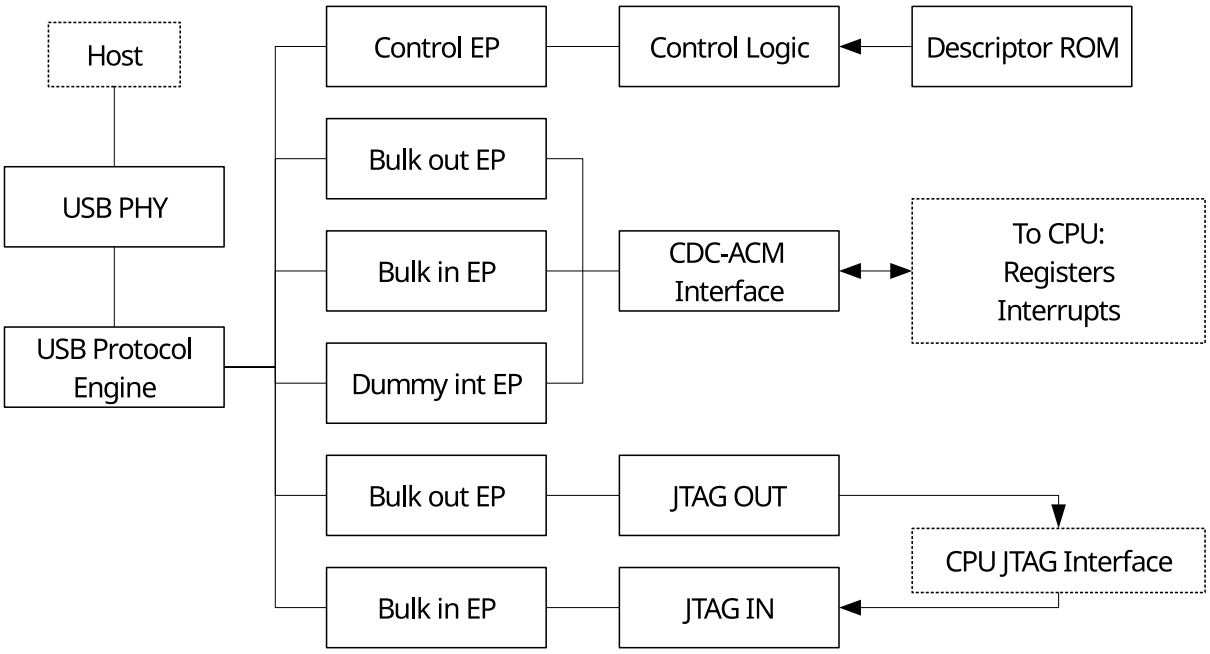


图 54.2-2. USB 串口/JTAG 控制器框图

54.3 功能描述

USB 串口/JTAG 控制器一边与 USB 主机处理器连接，另一边与 CPU 调试硬件以及通过 CDC-ACM 端口通信的软件连接。

54.3.1 CDC-ACM USB 接口描述

CDC-ACM 接口遵循标准 USB CDC-ACM 类别进行虚拟串口通信，包含一个虚拟中断端点（不会发送任何事件，无使用需求）以及一个批量输入端点 (Bulk IN) 和一个批量输出端点 (Bulk OUT) 进行数据接收和发送。这些端点一次可以处理最高 64 字节的数据包，实现高吞吐量。CDC-ACM 为标准的 USB 设备类型，主机一般无需任何特殊安装程序就能正常工作，也就是说，当 USB 调试设备正确连接至主机时，操作系统应能在片刻后显示新的串口信息。

CDC-ACM 接口可以接收以下标准 CDC-ACM 控制请求：

表 54.3-1. 标准 CDC-ACM 控制请求

命令	操作
SEND_BREAK	接收但忽略（虚拟命令）
SET_LINE_CODING	接收，发送值在软件中可读
GET_LINE_CODING	默认返回 9600 baud，无奇偶校验，8 个数据位，1 个停止位（可通过软件更改）
SET_CONTROL_LINE_STATE	设置 RTS/DTR 线的状态，如表 54.3-2 所示

除了通用的通信之外，CDC-ACM 接口还可以复位 ESP32-P4 并选择使其进入下载模式，从而烧录新的固件。这一功能可通过设置虚拟串口的 RTS 和 DTR 线来实现。

表 54.3-2. CDC-ACM 中 RTS 和 DTR 的设置

RTS	DTR	操作
0	0	清除下载模式标志
0	1	置位下载模式标志
1	0	复位 ESP32-P4
1	1	无操作

请注意，当 ESP32-P4 复位时，如果下载模式标志已置位，则 ESP32-P4 重启时将直接进入下载模式；如果下载模式标志已清除，则 ESP32-P4 将从 flash 启动。具体操作流程，请参见章节 54.4。除此之外，也可以通过烧写相应 eFuse 来禁用上述功能，详细信息请参见章节 9.4 *eFuse 控制器 (EFUSE)*。

54.3.2 CDC-ACM 固件接口描述

由于 USB 串口/JTAG 控制器与 ESP32-P4 的内部 APB 总线相连，因此 CPU 可直接与该模块交互，主要用于对连接主机上的虚拟串口进行读写操作。

CPU 向主机发送并从主机接收 0 ~ 64 字节大小的 USB CDC-ACM 串口数据包。主机已积累足够多的 CDC-ACM 数据时，将向 CDC-ACM 的接收端点发送一个数据包。如果 USB 串行/JTAG 控制器中有空闲缓冲区，该缓冲区将接收这一数据包。反之，主机会定期检查 USB 串口/JTAG 控制器内是否有待向主机发送的数据包，如果有，主机将接收这个数据包。

固件可通过以下两种方式之一获知是否有来自主机的新数据：第一，只要缓冲区中还有来自主机的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位将保持为 1；第二，如果有新的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断将被触发。当新数据可用时，固件可通过重复从 USB_SERIAL_JTAG_EP1_REG 读取字节来获取该数据。读取每个字节后，可通过检查 USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位来看是否还有其他可读取的数据，从而确定需要读取的总字节数。读取完所有数据后，USB 设备会自动做好准备，以接收来自主机的新数据包。

当固件需要发送数据时，可将待发送数据置于发送缓冲区并触发刷写，从而使主机以 USB 数据包的形式接收该数据。在此之前，需确保发送缓冲区有可用空间以存储待发送数据。固件可通过读取 USB_REG_SERIAL_IN_EP_DATA_FREE 位检查发送缓冲区是否有可用空间；当该值为 1 时，发送缓冲区有可用空间。此时，固件可通过向 USB_SERIAL_JTAG_EP1_REG 寄存器中写入字节，从而向缓冲区中写入数据。但是，数据写入后并不会立即触发向主机发送数据，需首先对缓冲区执行刷写操作。刷写操作后，整个缓冲区将准备好被 USB 主机立即接收。可通过两种方式触发刷写：1) 将第 64 个字节写入缓冲区后，USB 硬件会自动将缓冲区刷写到主机；2) 固件可通过向 USB_REG_SERIAL_WR_DONE 写入 1 来触发刷写。

不论以何种方式触发刷写操作，在此期间固件都无法向缓冲区写入数据，直到缓冲区中的所有数据都已被主机读取完成。主机读取完成后，将触发 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断，此时可向缓冲区内写入新的 64 字节数据。

在软件中可以处理一些带外串行请求，例如主机设置 DTR 和 RTS，以及改变线路状态。如果 CDC-ACM 接口收到 SET_LINE_CODING 请求，可将外设配置为触发 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断，此时可以从 USB_SERIAL_JTAG_SET_LINE_CODE_WO_REG 寄存器读取线路编码。类似的，如果 SET_CONTROL_LINE_STATE 请求导致线路状态改变，将触发 USB_SERIAL_JTAG_RTS_CHG_INT 和 USB_SERIAL_JTAG_DTR_CHG_INT 中断。此后，软件可以通过 USB_SERIAL_JTAG_RTS 和 USB_SERIAL_JTAG_DTR 位来读取具体状态。注意，如前所述，某些 RTS/DTR 序列会导致 ESP32-P4 硬件复位。软件可以通过设置 USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS 位来禁止硬件识别这些 DTR/RTS 序列，从而允许软件自由解读信号。

最后，主机可以通过 GET_LINE_CODING 读取当前线路状态。此事件将回送 USB_SERIAL_JTAG_GET_LINE_CODE_WO_REG 寄存器中的数据，并触发 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。

54.3.3 USB-JTAG 接口：JTAG 命令处理器

USB-JTAG 接口使用一种供应商特定接口类型实现命令解析的功能。它由两个端点组成：一个用于接收命令，一个用于发送响应。此外，一些对时效性要求不高的命令也可以作为控制请求发出。

JTAG 命令处理器负责解析从主机到 JTAG 接口的命令。JTAG 命令处理器内部包含一个全四线 JTAG 总线，包括发送信号到 RISC-V CPU 的 TCK、TMS 和 TDI 输出线，以及从 CPU 返回信号至 JTAG 响应捕捉单元的 TDO 线。这些信号都符合 IEEE 1149.1 JTAG 标准。此外，还有一条 SRST 线用于复位 ESP32-P4。

另外，软件也可以置位 USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN，将这些信号通过 GPIO 交换矩阵连接至 ESP32-P4 的 IO 焊盘上，此操作也将允许通过 USB 串口/JTAG 控制器对外部设备进行调试。

JTAG 命令处理器会将每个接收到的半字节（4 位）解析为一条命令。由于 USB 以 8 位为一个字节接收数据，这就意味着每个字节中都包含两条命令。USB 命令处理器将先解析高 4 位字节，然后再解析低 4 位字节。这些命令用于控制内部 JTAG 总线的 TCK、TMS、TDI、SRST 线，以及向 JTAG 响应捕捉单元发出信号，说明需要捕捉 TDO 线（由 CPU 调试逻辑驱动）的状态。

JTAG 总线中，TCK、TMS、TDI 和 TDO 线直接与 RISC-V CPU 的 JTAG 调试逻辑相连。上文提到的 SRST 线则与 ESP32-P4 数字电路中的复位逻辑相连，该线电平拉高，芯片将进行芯片复位。请注意，SRST 线并不会对 USB 串口/JTAG 控制器模块产生影响。

1 个半字节中可包含以下命令：

表 54.3-3. 半字节中的命令

位	3	2	1	0
CMD_CLK	0	cap	tms	tdi
CMD_RST	1	0	0	srst
CMD_FLUSH	1	0	1	0
CMD_RSV	1	0	1	1
CMD_REP	1	1	R1	R0

- CMD_CLK：将 TDI 和 TMS 设置为指示值，并在 TCK 上发出一个时钟脉冲。如果 CAP 位为 1，将指示 JTAG 响应捕捉单元捕捉 TDO 线的状态。该指令构成了 JTAG 通信的基础。
- CMD_RST：将 SRST 线的状态设置为指示值。该命令可用于复位 ESP32-P4。
- CMD_FLUSH：指示 JTAG 响应捕捉单元对接收到的所有位的缓冲区进行刷写操作，以便主机读取这些位。请注意在某些情况下，一次 JTAG 通信会结束于第奇数个命令，即结束于第奇数个半字节。此时，可重复执行该命令直到获得偶数个半字节，使其组成整数个字节。
- CMD_RSV：该版本中保留。ESP32-P4 在接收到该命令时会自动忽略。
- CMD_REP：将上一条指令（非 CMD_REP）重复一定次数，以压缩多次重复 CMD_CLK 的命令流。因此，CMD_CLK 命令后可能跟随着多个 CMD_REP。一次 CMD_REP 命令产生的重复次数可表示为 $repetition_count = (R1 \times 2 + R0) \times (4^{cmd_rep_count})$，其中 cmd_rep_count 表示该命令之前的 CMD_REP 数量。请注意，CMD_REP 仅用于重复 CMD_CLK 命令。也就是说，如果在 CMD_FLUSH 后使用该命令，USB 设备将无法响应，需进行 USB 复位后才可恢复正常。

54.3.4 USB-JTAG 接口：CMD_REP 使用示例

下列命令用于演示如何使用 CMD_REP 命令。请注意，该示例中每个命令为半字节，命令流的每个字节为 0x0D 0x5E 0xCF。

1. 0x0 (CMD_CLK: cap=0, tdi=0, tms=0)
2. 0xD (CMD_REP: R1=0, R0=1)
3. 0x5 (CMD_CLK: cap=1, tdi=0, tms=1)
4. 0xE (CMD_REP: R1=1, R0=0)
5. 0xC (CMD_REP: R1=0, R0=0)
6. 0xF (CMD_REP: R1=1, R0=1)

每一步骤的具体操作为：

1. TCK 上发出一个时钟脉冲，TDI 和 TMS 设置为 0。未捕捉到任何数据。
2. TCK 上再次发出 $(0 \times 2 + 1) \times (4^0) = 1$ 个时钟脉冲，其他设置与步骤 1 相同。
3. TCK 上发出一个时钟脉冲，TDI 设置为 0，TMS 设置为 1。捕捉到 TDO 线上的数据。
4. TCK 上再次发出 $(1 \times 2 + 0) \times (4^0) = 2$ 个时钟脉冲，其他设置与步骤 3 相同。
5. 未发生任何动作：$(0 \times 2 + 0) \times (4^1) = 0$。请注意，该操作将增加下一步骤中的 cmd_rep_count 数值。
6. TCK 上再次发出 $(1 \times 2 + 1) \times (4^2) = 48$ 个时钟脉冲，其他设置与步骤 3 相同。

换言之，该命令流示例的操作结果等同于执行两次命令 1，然后执行 51 次命令 3 的效果。

54.3.5 USB-JTAG 接口：响应捕捉单元

响应捕捉单元首先读取内部 JTAG 总线的 TDO 线，并在命令处理器执行 CMD_CLK (cap=1) 命令时捕捉 TDO 线的值，然后将这个值放入内部移位寄存器中，且在接收到 8 位时向 USB 缓冲区写入 1 个字节。这 8 位中的最低有效位即为最先从 TDO 线读取的值。

一旦接收到 64 字节（512 位）数据或执行 CMD_FLUSH 命令后，响应捕捉单元将使缓冲区可被主机接收。请注意，USB 逻辑的接口为双缓冲。因此，只要 USB 的吞吐量充足，响应捕捉单元就可以随时接收更多数据，即当一个缓冲区等待发送给主机时，另一个缓冲区可以继续接收数据。当主机从缓冲区成功接收数据且响应捕捉单元对缓冲区执行刷写操作后，这两个缓冲区便可以交换位置。

同时，这也意味着一个命令流可导致最多 128 字节（若该命令流中有刷写命令，则该数字会减小）的捕捉数据生成，而无需主机主动接收这些数据。如果还是生成了超过该阈值数量的捕捉数据，则命令流将暂停，且设备在这些数据被读取之前不会接收其他命令。

另需注意，一般情况下，响应捕捉单元会尽量不发送 0 字节响应。例如，当发送一系列 CMD_FLUSH 命令后，并不会产生一系列 0 字节 USB 响应。但在当前版本中，一些特殊情况下也可能产生 0 字节响应。建议直接忽略这些响应信息。

54.3.6 USB-JTAG 接口：控制传输请求

除命令处理器和响应捕捉单元之外，USB-JTAG 接口也可接收一些控制请求，具体如下表所示：

表 54.3-4. USB-JTAG 控制请求

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000000b	0 (VEND_JTAG_SETDIV)	[divider]	接口	0	None
01000000b	1 (VEND_JTAG_SETIO)	[iobits]	接口	0	None
11000000b	2 (VEND_JTAG_GETTDO)	0	接口	1	[iostate]
10000000b	6 (GET_DESCRIPTOR)	0x2000	0	256	[jtag cap desc]

- VEND_JTAG_SETDIV：设置使用的分频器。该命令直接影响 TCK 时钟脉冲的持续时间。TCK 时钟脉冲来自于由内部分频器向下分频得到的 48 MHz 基准时钟。该控制请求允许主机设置此内部分频器。请注意，该分频器在启动时的初始值为 2，即 TCK 时钟速率一般为 24 MHz。
- VEND_JTAG_SETIO：跳过 JTAG 命令处理器，直接将内部 TDI、TDO、TMS 和 SRST 线设置为指定值。这些指定值在 wValue 字段中以 11'b0, srst, trst, tck, tms, tdi 的格式编码。
- VEND_JTAG_GETTDO：跳过 JTAG 响应捕捉单元，直接读取内部 TDO 信号。该请求返回 1 个字节，其中的最低有效位代表 TDO 线的状态。
- GET_DESCRIPTOR：为标准 USB 请求，该请求也可使用供应商专用的 0x2000 wValue 获取 JTAG 功能描述符。该请求返回一定字节，具体字节数所代表的 USB-JTAG 适配器功能如表 54.3-5 所示。这一固定结构允许主机软件自动支持未来的硬件新版本，无需再次更新。

ESP32-P4 包含的 JTAG 功能描述符如下表所示。请注意，所有 16 位值都为小端序存储。

表 54.3-5. JTAG 功能描述符

字节	数值	描述
0	1	JTAG 协议功能结构的版本
1	10	JTAG 协议功能长度
2	1	结构类型：1 代表高速功能结构类型
3	8	该高速功能结构长度
4 ~ 5	4800	以 10 kHz 为增量的 JTAG 基准时钟速度，其最大速度为该值的一半
6 ~ 7	1	最小分频系数，可通过 VEND_JTAG_SETDIV 设置
8 ~ 9	255	最大分频系数，可通过 VEND_JTAG_SETDIV 设置

54.4 操作建议

使用 USB 串口/JTAG 控制器之前，几乎不需要多余的配置。除了主机操作系统已经完成的标准 USB 初始化之外，USB-JTAG 硬件本身不需要进行任何配置。此外，CDC-ACM 虚拟串口在主机端也支持即插即用。

固件方面也几乎不需要初始化。USB 硬件是自初始化的，在其启动后，如果固件连接了一台主机并在 CDC-ACM 接口上监听，除非固件设置了中断处理程序，否则无需任何特定设置就可以实现前文所述的数据交换。

需要注意的是，可能会出现主机未连接或 CDC-ACM 虚拟串口未打开的情况。在这种情况下，发送至主机的数据包永远无法被接收，发送缓冲区也永远不会为空。因而，必须对这种情况进行检测并执行超时操作，以便清楚地检测主机侧的端口是否关闭。

其次，需要注意 USB 设备依赖于产生 48 MHz USB PHY 时钟的 BBPLL。如果此 PLL 被禁用，USB 通信将停止。

上述情况可能发生在 Deep-sleep 期间。在 Deep-sleep 模式下，USB 串口/JTAG 控制器（及其连接的 RISC-V CPU）将完全断电。如果有设备需要在这两种模式下进行调试，最好使用外部 JTAG 调试器和串行接口。

CDC-ACM 接口还可用于复位芯片，使其进入或退出下载模式。产生正确的握手信号序列则有些复杂，因为大多数操作系统仅支持分别设置或重置 DTR 和 RTS，无法同时进行。此外，一些驱动程序（如 Windows 系统上的标准 CDC-ACM 驱动程序）须先设置 RTS 后才可设置 DTR，此时用户必须先明确设置 RTS，才能传播 DTR 的值。推荐遵循以下步骤进行设置。

复位芯片使其进入下载模式：

表 54.4-1. 复位芯片进入下载模式

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	初始化以获取数值
清除 RTS	RTS=0, DTR=0	-
设置 DTR	RTS=0, DTR=1	设置下载模式标志
清除 RTS	RTS=0, DTR=1	传播 DTR
设置 RTS	RTS=1, DTR=1	-
清除 DTR	RTS=1, DTR=0	复位芯片
设置 RTS	RTS=1, DTR=0	传播 DTR
清除 RTS	RTS=0, DTR=0	清除下载标志

复位 SoC 使其从 flash 启动：

表 54.4-2. 复位 SoC 从 flash 启动

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	-
清除 RTS	RTS=0, DTR=0	清除下载标志
设置 RTS	RTS=1, DTR=0	复位 SoC
清除 RTS	RTS=0, DTR=0	退出复位

54.5 中断

ESP32-P4 的 USB 串口/JTAG 控制器的下列内部中断源可生成中断信号 USB_SERIAL_JTAG_INTR，并发送给中断矩阵。

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT：JTAG 批量输出端点接收到刷写命令时触发。
- USB_SERIAL_JTAG_SOF_INT：接收到 SOF 帧时触发。
- USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT：串口输出端点接收到 1 个数据包时触发。
- USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT：串口输入端点为空时触发。
- USB_SERIAL_JTAG_PID_ERR_INT：检测到 PID 错误时触发。
- USB_SERIAL_JTAG_CRC5_ERR_INT：检测到 CRC5 错误时触发。
- USB_SERIAL_JTAG_CRC16_ERR_INT：检测到 CRC16 错误时触发。
- USB_SERIAL_JTAG_STUFF_ERR_INT：检测到位填充错误时触发。

- USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT：输入端点 1 接收到一个 IN 令牌时触发。
- USB_SERIAL_JTAG_USB_BUS_RESET_INT：检测到 USB 总线复位时触发。
- USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT：输出端点 1 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT：输出端点 2 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_RTS_CHG_INT：USB 串行通道的 RTS 电平改变时触发。
- USB_SERIAL_JTAG_DTR_CHG_INT：USB 串行通道的 DTR 电平改变时触发。
- USB_SERIAL_JTAG_GET_LINE_CODE_INT：收到 GET LINE CODING 请求的电平时触发。
- USB_SERIAL_JTAG_SET_LINE_CODE_INT：收到 SET LINE CODING 请求的电平时触发。

54.6 寄存器列表

本小节的所有地址均为相对于 USB 串口/JTAG 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

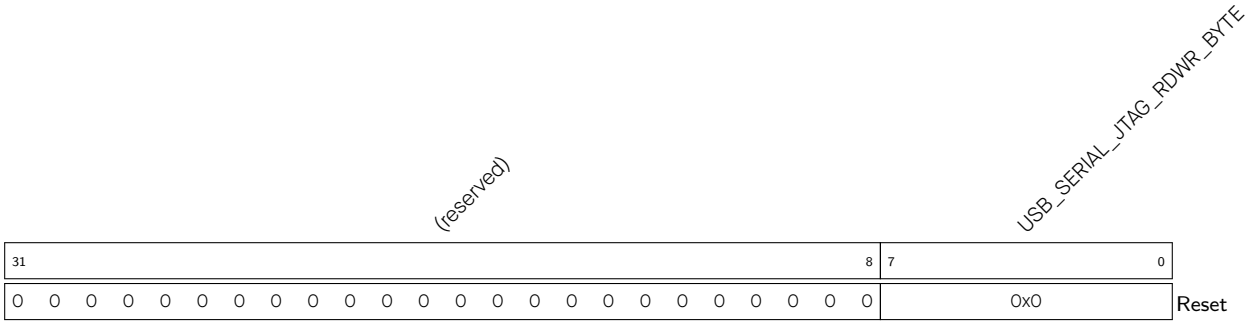
名称	描述	地址	访问
配置寄存器			
USB_SERIAL_JTAG_EP1_REG	CDC-ACM 输入/输出端点 FIFO 访问配置寄存器	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	CDC-ACM FIFO 配置寄存器	0x0004	varies
USB_SERIAL_JTAG_CONF0_REG	PHY 硬件配置寄存器	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	PHY 调试寄存器	0x001C	varies
USB_SERIAL_JTAG_MISC_CONF_REG	时钟使能控制寄存器	0x0044	R/W
USB_SERIAL_JTAG_MEM_CONF_REG	存储器配置寄存器	0x0048	R/W
USB_SERIAL_JTAG_CHIP_RST_REG	CDC-ACM 芯片重置寄存器	0x004C	varies
USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG	GET_LINE_CODING 命令的 W0	0x0058	R/W
USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG	GET_LINE_CODING 命令的 W1	0x005C	R/W
USB_SERIAL_JTAG_CONFIG_UPDATE_REG	配置寄存器更新寄存器	0x0060	WT
USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG	Serial AFIFO 配置寄存器	0x0064	varies
中断寄存器			
USB_SERIAL_JTAG_INT_RAW_REG	中断原始状态寄存器	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	中断状态寄存器	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	中断使能状态寄存器	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	中断清除状态寄存器	0x0014	WT
状态寄存器			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO 状态与控制寄存器	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	接收 SOF 帧索引寄存器	0x0024	RO
USB_SERIAL_JTAG_IN_EP0_ST_REG	输入端点状态信息寄存器	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM 输入端点状态信息寄存器	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM 中断输入端点状态信息寄存器	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG 输入端点状态信息寄存器	0x0034	RO
USB_SERIAL_JTAG_OUT_EP0_ST_REG	输出端点状态信息寄存器	0x0038	RO
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM 输出端点状态信息寄存器	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG 输出端点状态信息寄存器	0x0040	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	SET_LINE_CODING 命令的 W0	0x0050	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG	SET_LINE_CODING 命令的 W1	0x0054	RO
USB_SERIAL_JTAG_BUS_RESET_ST_REG	USB 总线重置状态寄存器	0x0068	RO
版本寄存器			
USB_SERIAL_JTAG_DATE_REG	版本寄存器	0x0088	R/W

54.7 寄存器

本小节的所有地址均为相对于 USB 串口/JTAG 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 54.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



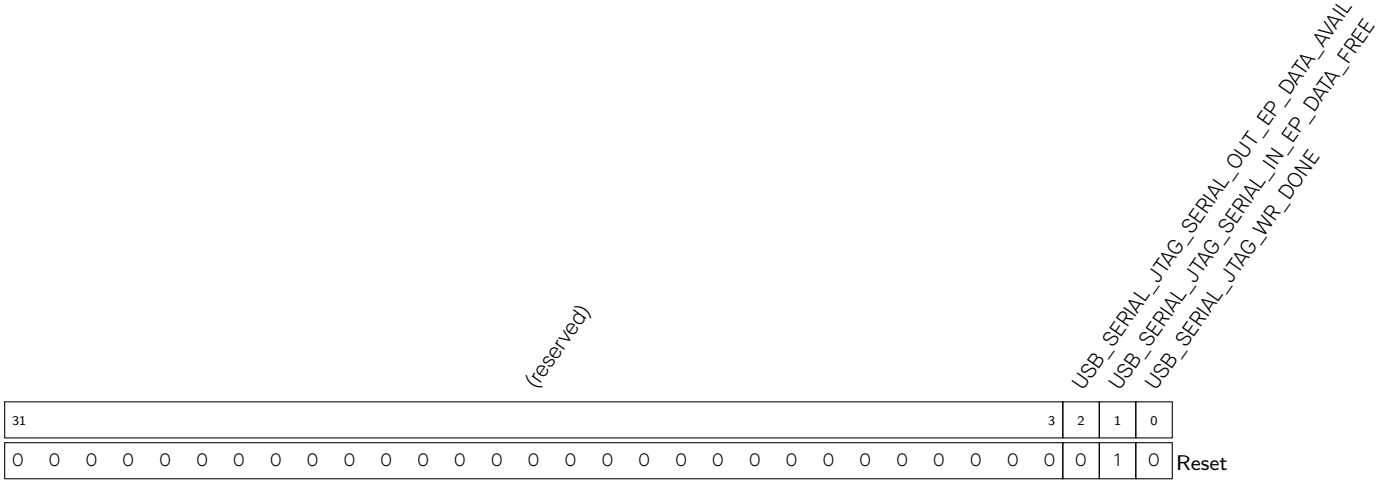
USB_SERIAL_JTAG_RDWR_BYTE 写入该位即为将数据写入 CDC-ACM TX FIFO；读取该位即为从 CDC-ACM RX FIFO 读取一个字节并返回该字节。

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 置位时，用户可向 CDC-ACM TX FIFO 中写入数据（最大 64 字节）。

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 置位时，用户可通过查看 USB_SERIAL_JTAG_OUT_EP1_WR_ADDR 的值获知接收到的数据量，然后从 CDC-ACM RX FIFO 中读取这些数据。

(R/W)

Register 54.2. USB_SERIAL_JTAG_EP1_CONF_REG (0x0004)



USB_SERIAL_JTAG_WR_DONE 配置是否表明已完成向 CDC-ACM TX FIFO 写入字节数据。

0: 无效

1: 表明已完成向 CDC-ACM TX FIFO 写入字节数据

该位将保持为 0，直到 USB 主机读取 CDC-ACM TX FIFO 中的数据。

(WT)

USB_SERIAL_JTAG_SERIAL_IN_EP_DATA_FREE 表示 CDC-ACM TX FIFO 是否可以写入数据。

0：CDC-ACM TX FIFO 已写满，不可写入数据

1：CDC-ACM TX FIFO 未写满且可以写入数据

写入 USB_SERIAL_JTAG_WR_DONE 后，该位将置为 0，直到其中的数据已发送至 USB 主机。

(RO)

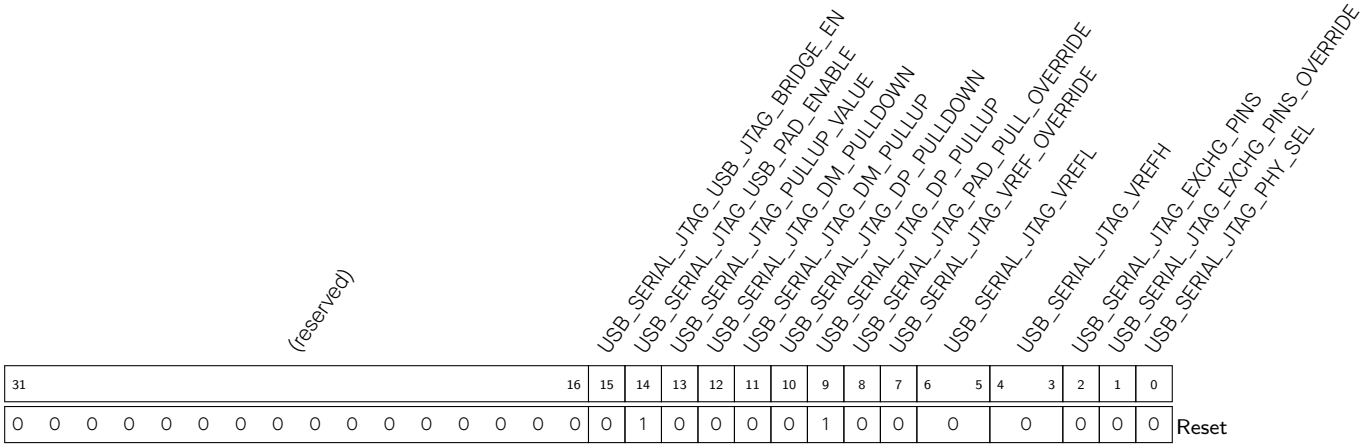
USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 表示 CDC-ACM RX FIFO 中是否有数据。

0: CDC-ACM RX FIFO 中没有数据

1: CDC-ACM RX FIFO 中有数据

(RO)

Register 54.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)



USB_SERIAL_JTAG_PHY_SEL 配置选择使用内部 PHY 或外部 PHY。

0：内部 PHY

1：外部 PHY

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚交换。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS 配置是否交换 USB D+ 和 D- 管脚。

0：无效

1：交换

(R/W)

USB_SERIAL_JTAG_VREFH 配置单端输入高阈值。

0：1.76 V

1：1.84 V

2：1.92 V

3：2.00 V

(R/W)

USB_SERIAL_JTAG_VREFL 配置单端输入低阈值。

0：0.80 V

1：0.88 V

2：0.96 V

3：1.04 V

(R/W)

USB_SERIAL_JTAG_VREF_OVERRIDE 配置是否使能软件控制输入阈值。

0：不使能

1：使能

(R/W)

见下页...

Register 54.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

接上页...

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚的上下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DP_PULLUP 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D+ 管脚的上拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D+ 管脚的下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DM_PULLDOWN 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D- 管脚的下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时的上拉数值。

0：2.2 K

1：1.1 K

(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE 配置是否使能 USB 管脚。

0：不使能

1：使能

(R/W)

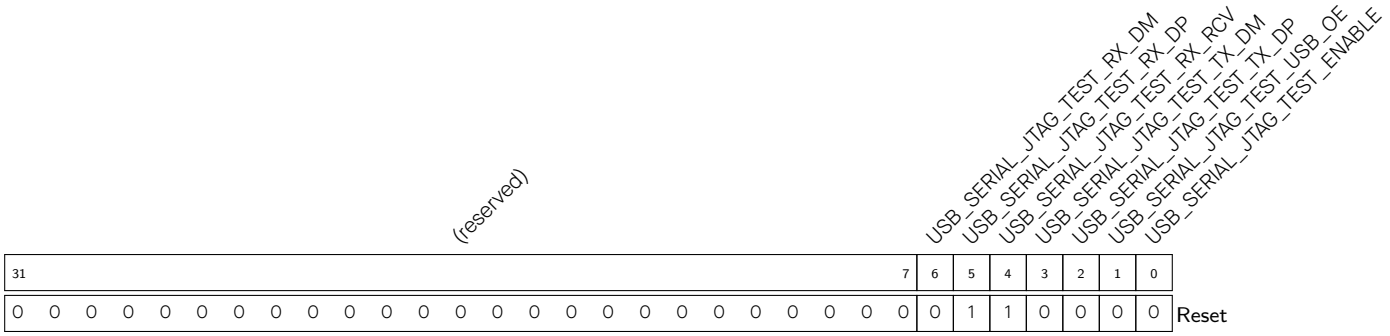
USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN 配置是否断开 USB_JTAG 和内部 JTAG 的连接。

0：USB_JTAG 连接到 CPU 的内部 JTAG 端口

1：USB_JTAG 和内部 JTAG 之间断开连接，MTMS、MTDI 和 MTCK 为通过 GPIO 交换矩阵的输出，MTDO 为通过 GPIO 交换矩阵的输入

(R/W)

Register 54.4. USB_SERIAL_JTAG_TEST_REG (0x001C)



USB_SERIAL_JTAG_TEST_ENABLE 配置是否使能 USB 管脚测试模式。

0：不使能，恢复正常模式

1：使能 USB 管脚测试模式

使能 USB 管脚测试模式将允许使用该寄存器的其他位来控制/读取 USB 管脚。

(R/W)

USB_SERIAL_JTAG_TEST_USB_OE 配置是否使能 USB 管脚输出。

0：将 D+ 和 D- 设为高阻抗

1：在 D+ 和 D- 管脚上输出 USB_SERIAL_JTAG_TEST_TX_DP 和 USB_SERIAL_JTAG_TEST_TX_DM 中设置的值

(R/W)

USB_SERIAL_JTAG_TEST_TX_DP 当 USB_SERIAL_JTAG_TEST_USB_OE 为 1 时，设置测试模式下 USB D+ 管脚的值。(R/W)

USB_SERIAL_JTAG_TEST_TX_DM 当 USB_SERIAL_JTAG_TEST_USB_OE 为 1 时，设置测试模式下 USB D- 管脚的值。(R/W)

USB_SERIAL_JTAG_TEST_RX_RCV 表示在测试模式下 USB D- 和 USB D+ 管脚之间电压差的当前逻辑水平。

0：USB D- 电压高于 USB D+

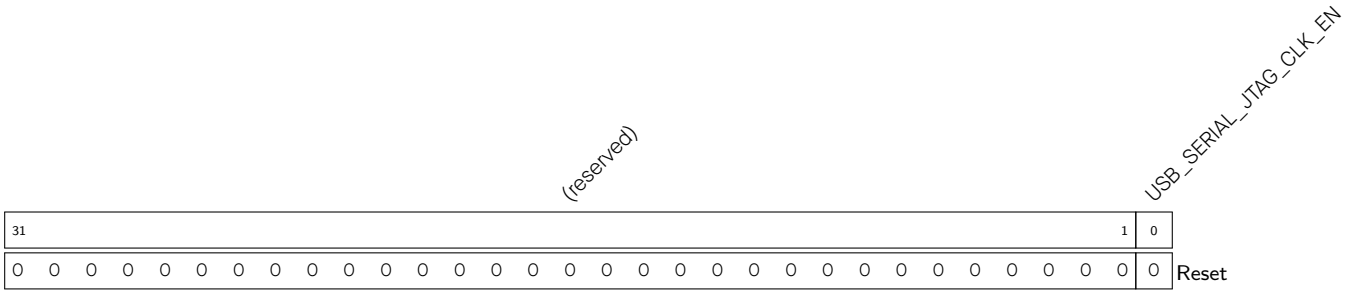
1：USB D- 电压低于 USB D+

(RO)

USB_SERIAL_JTAG_TEST_RX_DP 表示在测试模式下 USB D+ 管脚的逻辑电平。(RO)

USB_SERIAL_JTAG_TEST_RX_DM 表示在测试模式下 USB D- 管脚的逻辑电平。(RO)

Register 54.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



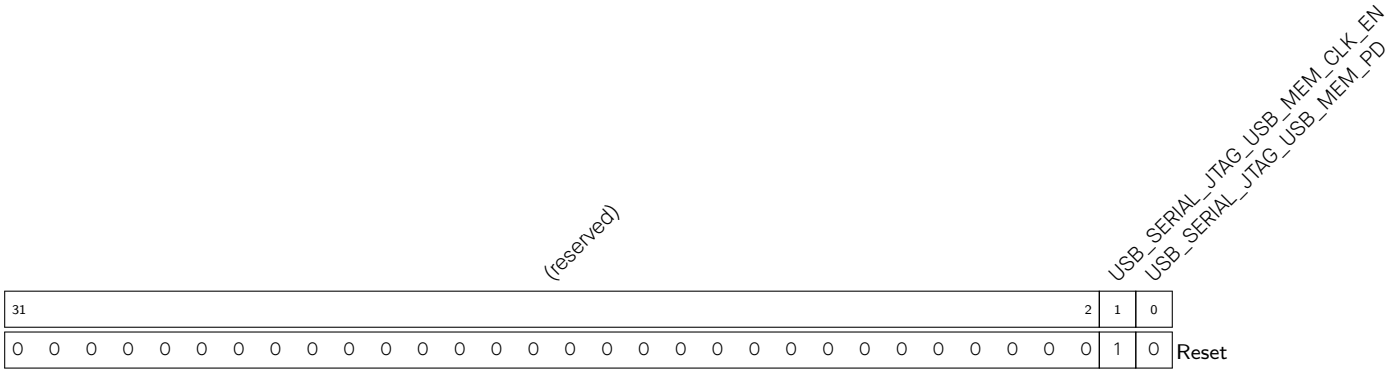
USB_SERIAL_JTAG_CLK_EN 配置是否强制打开寄存器的时钟。

0：支持仅当应用程序向寄存器写入数据时打开时钟

1：强制打开寄存器的时钟

(R/W)

Register 54.6. USB_SERIAL_JTAG_MEM_CONF_REG (0x0048)



USB_SERIAL_JTAG_USB_MEM_PD 配置是否关闭 USB 存储器。

0：无效

1：关闭

(R/W)

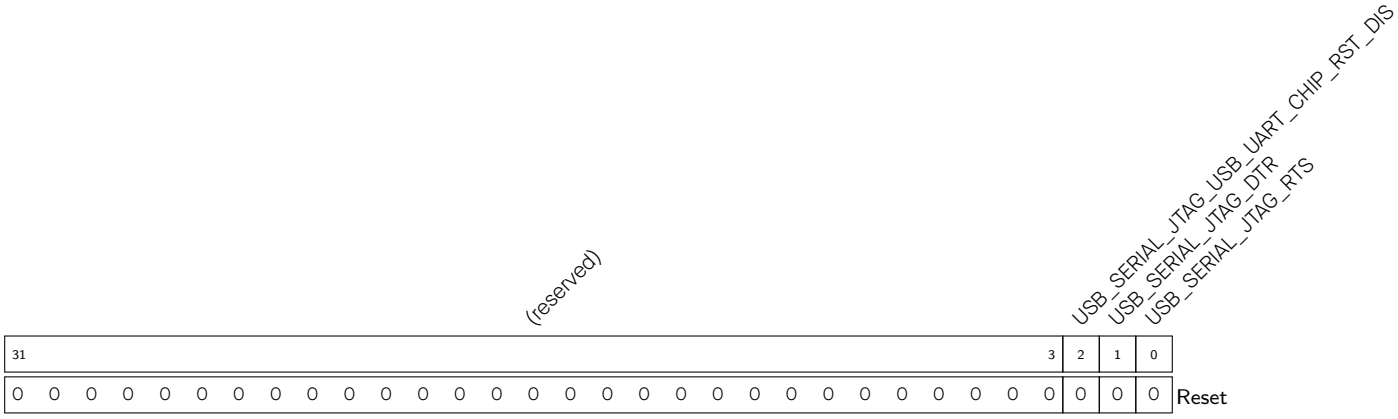
USB_SERIAL_JTAG_USB_MEM_CLK_EN 配置是否强制对 USB 存储器进行时钟分频。

0：无效

1：强制

(R/W)

Register 54.7. USB_SERIAL_JTAG_CHIP_RST_REG (0x004C)



USB_SERIAL_JTAG_RTS 表示最近的 SET_LINE_CODING 命令所设定的 RTS 信号的状态。(RO)

USB_SERIAL_JTAG_DTR 表示最近的 SET_LINE_CODING 命令所设定的 DTR 信号的状态。(RO)

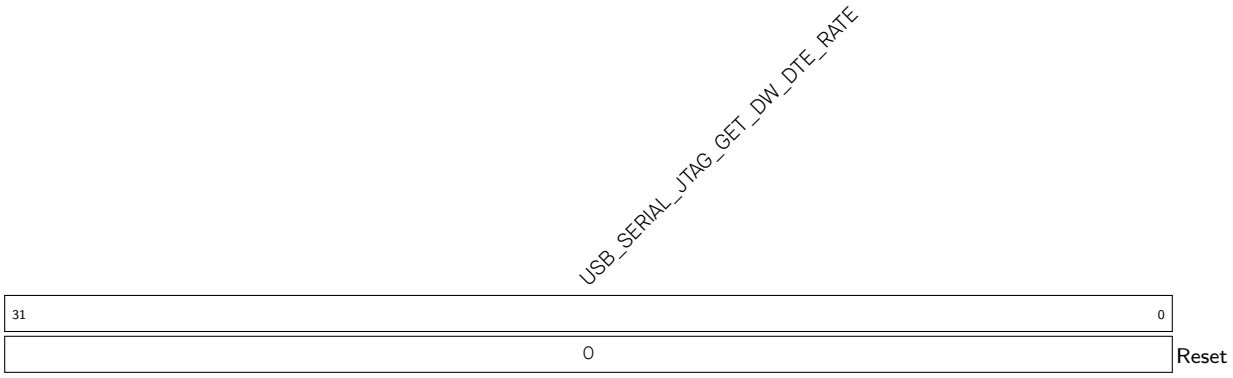
USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS 配置是否禁用来自 USB 串行通道的芯片复位。

0: 无效

1: 禁用

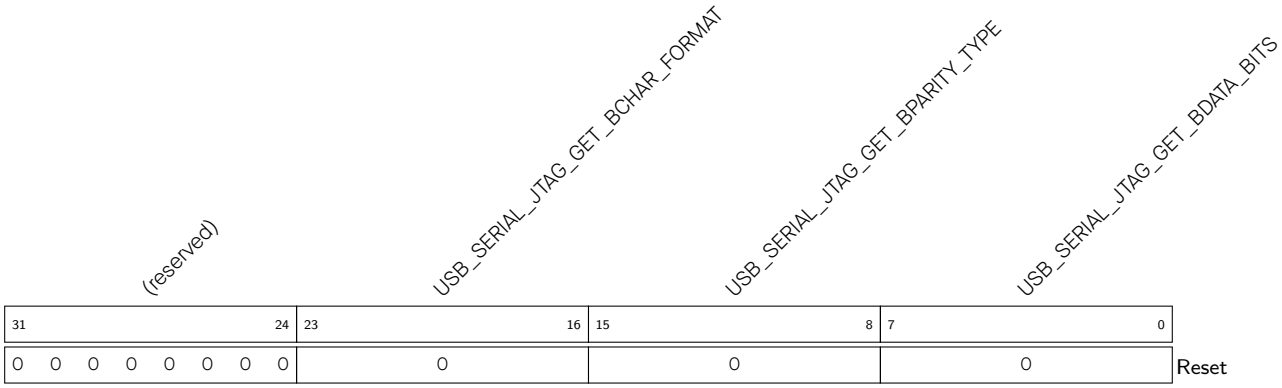
(R/W)

Register 54.8. USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG (0x0058)



USB_SERIAL_JTAG_GET_DW_DTE_RATE 配置软件设置的 dwDTERate 值，该值由 GET_LINE_CODING 命令请求。(R/W)

Register 54.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

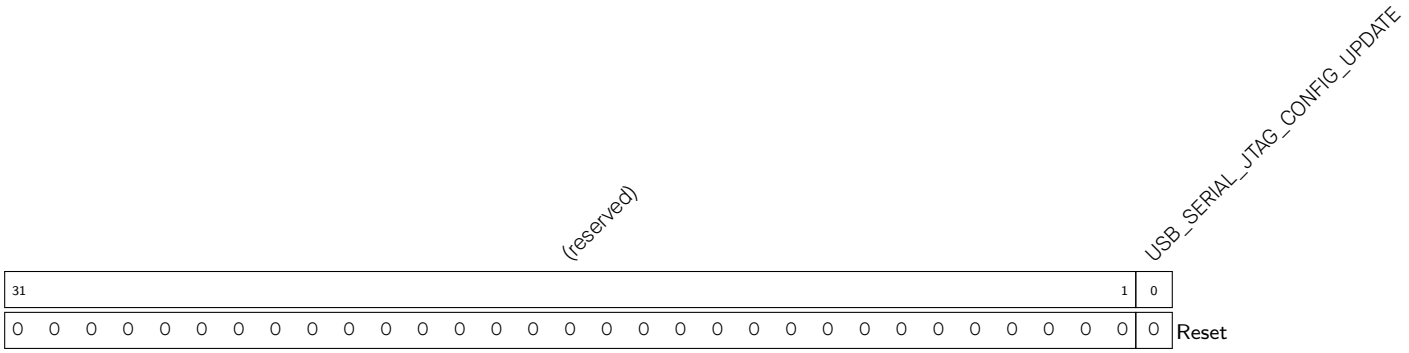


USB_SERIAL_JTAG_GET_BDATA_BITS 配置软件设置的 bDataBits 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BPARITY_TYPE 配置软件设置的 bParityType 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BCHAR_FORMAT 配置软件设置的 bCharFormat 值，该值由 GET_LINE_CODING 命令请求。(R/W)

Register 54.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



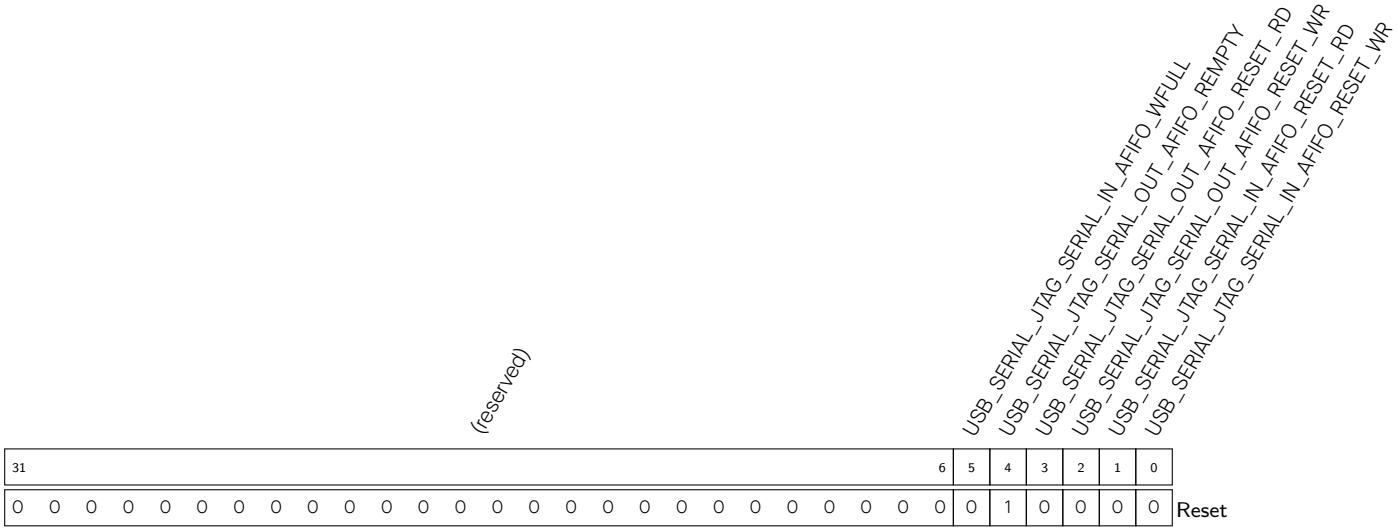
USB_SERIAL_JTAG_CONFIG_UPDATE 配置是否将配置寄存器的值从 APB 时钟域更新到 48 MHz 时钟域。

0: 无效

1: 更新

(WT)

Register 54.11. USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG (0x0064)



USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_WR 配置是否复位 CDC_ACM IN 异步 FIFO 写入时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_RD 配置是否复位 CDC_ACM IN 异步 FIFO 读取时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_WR 配置是否复位 CDC_ACM OUT 异步 FIFO 写入时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_RD 配置是否复位 CDC_ACM OUT 异步 FIFO 读取时钟域。

0: 无效

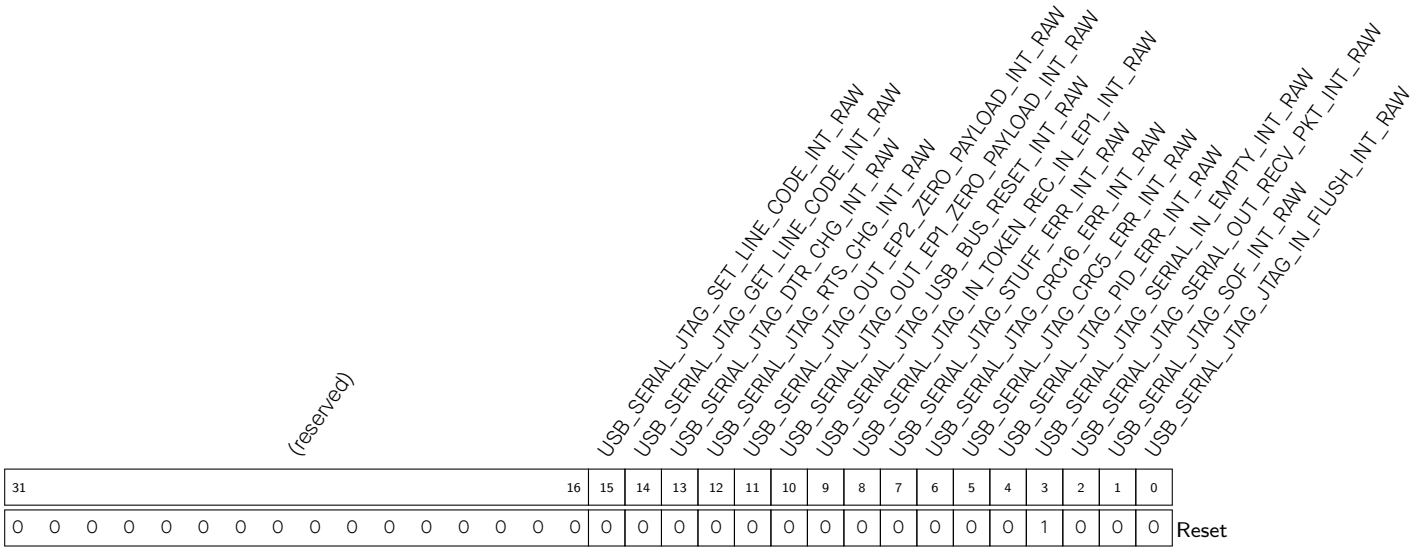
1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_REMPTY 表示 CDC_ACM OUT 读取时钟域的异步 FIFO 空信号。(RO)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_WFULL 表示 CDC_ACM IN 写入时钟域的异步 FIFO 满信号。(RO)

Register 54.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW USB_SERIAL_JTAG_SOF_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW USB_SERIAL_JTAG_PID_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW USB_SERIAL_JTAG_CRC5_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW USB_SERIAL_JTAG_CRC16_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW USB_SERIAL_JTAG_STUFF_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW USB_SERIAL_JTAG_USB_BUS_RESET_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 的原始中断状态。(R/WTC/SS)

见下页...

Register 54.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

接上页...

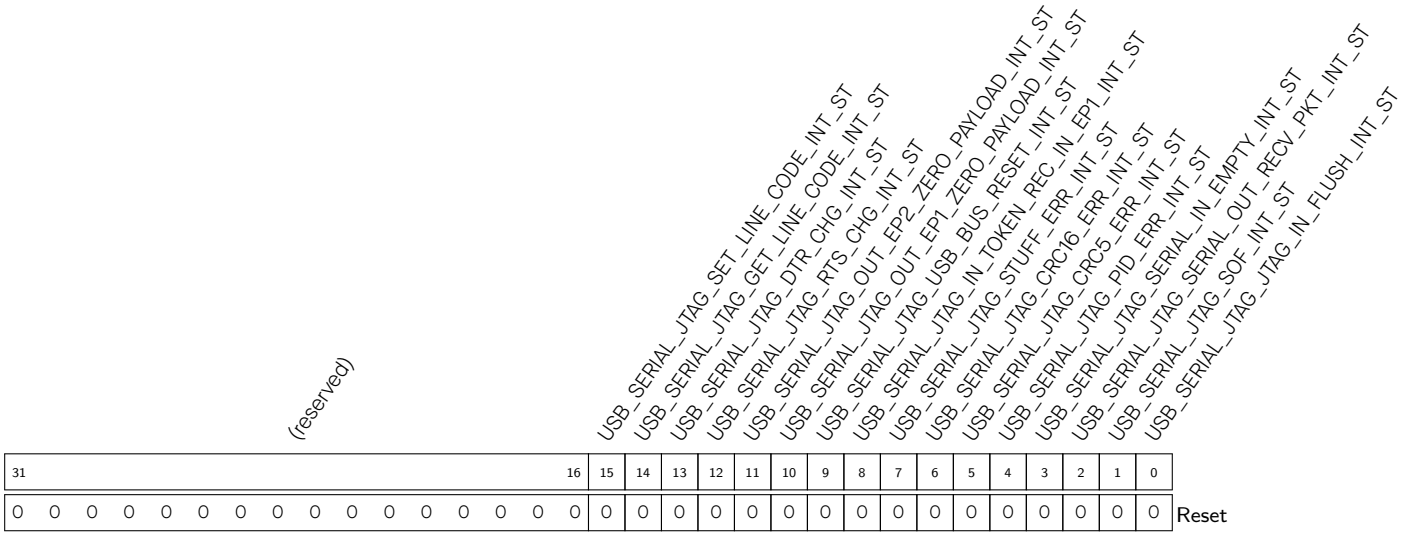
USB_SERIAL_JTAG_RTS_CHG_INT_RAW USB_SERIAL_JTAG_RTS_CHG_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW USB_SERIAL_JTAG_DTR_CHG_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_GET_LINE_CODE_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_SET_LINE_CODE_INT 的原始中断状态。(R/WTC/SS)

Register 54.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SOF_INT_ST USB_SERIAL_JTAG_SOF_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST USB_SERIAL_JTAG_PID_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST USB_SERIAL_JTAG_CRC5_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST USB_SERIAL_JTAG_CRC16_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST USB_SERIAL_JTAG_STUFF_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST USB_SERIAL_JTAG_USB_BUS_RESET_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

见下页...

Register 54.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

接上页...

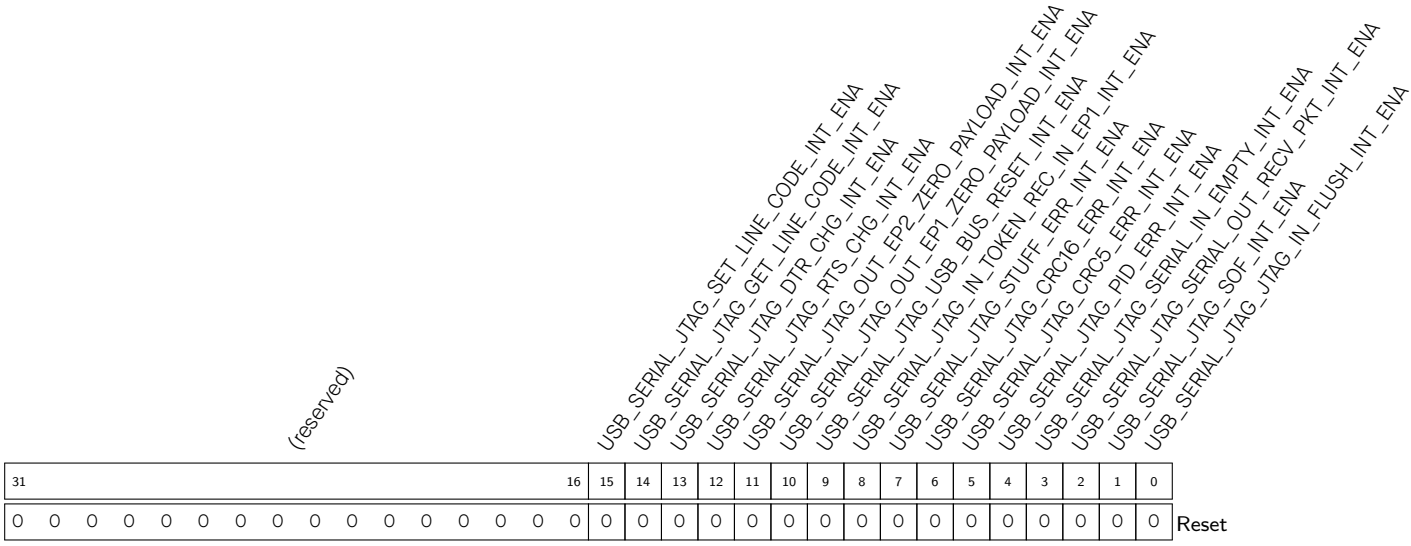
USB_SERIAL_JTAG_RTS_CHG_INT_ST USB_SERIAL_JTAG_RTS_CHG_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST USB_SERIAL_JTAG_DTR_CHG_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST USB_SERIAL_JTAG_GET_LINE_CODE_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST USB_SERIAL_JTAG_SET_LINE_CODE_INT 的屏蔽中断状态。(RO)

Register 54.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA 写 1 使能 USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 中断。(R/W)

USB_SERIAL_JTAG_SOF_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SOF_INT 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断。(R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_PID_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_CRC5_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_CRC16_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_STUFF_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA 写 1 使能 USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 中断。(R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA 写 1 使能 USB_SERIAL_JTAG_USB_BUS_RESET_INT 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA 写 1 使能 USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA 写 1 使能 USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 中断。(R/W)

见下页...

Register 54.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

接上页...

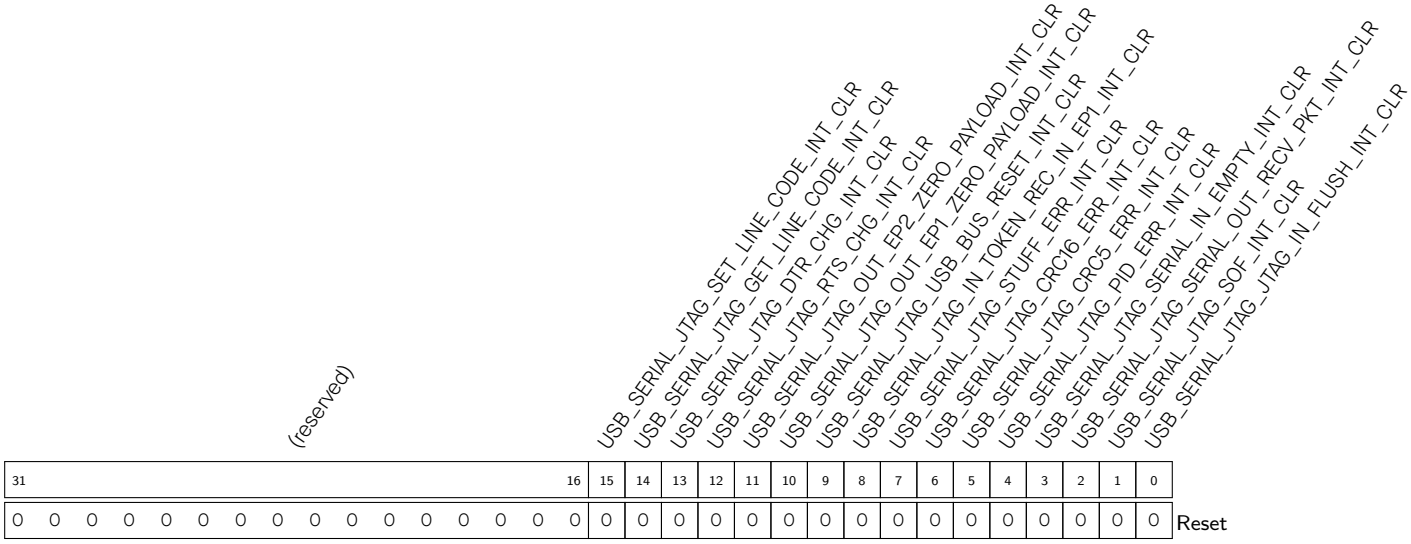
USB_SERIAL_JTAG_RTS_CHG_INT_ENA 写 1 使能 USB_SERIAL_JTAG_RTS_CHG_INT 中断。(R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA 写 1 使能 USB_SERIAL_JTAG_DTR_CHG_INT 中断。(R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA 写 1 使能 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。(R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断。(R/W)

Register 54.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR 写 1 清除 USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 中断。(WT)

USB_SERIAL_JTAG_SOF_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SOF_INT 中断。(WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断。(WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断。(WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_PID_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_CRC5_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_CRC16_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_STUFF_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR 写 1 清除 USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 中断。(WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR 写 1 清除 USB_SERIAL_JTAG_USB_BUS_RESET_INT 中断。(WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR 写 1 清除 USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 中断。(WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR 写 1 清除 USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 中断。(WT)

见下页...

Register 54.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

接上页...

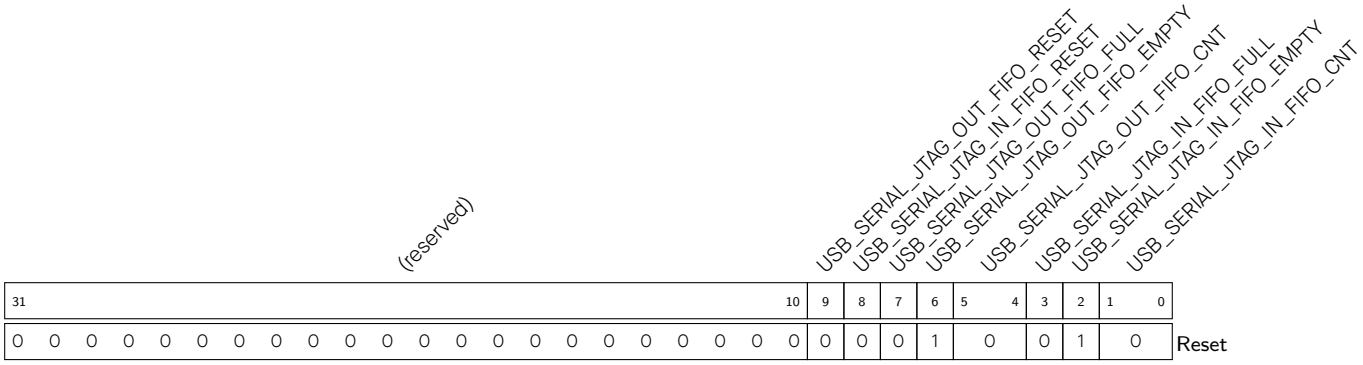
USB_SERIAL_JTAG_RTS_CHG_INT_CLR 写 1 清除 USB_SERIAL_JTAG_RTS_CHG_INT 中断。(WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR 写 1 清除 USB_SERIAL_JTAG_DTR_CHG_INT 中断。(WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR 写 1 清除 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。(WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断。(WT)

Register 54.16. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT 表示 JTAG IN FIFO 计数器。(RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY 表示 JTAG IN FIFO 是否为空。

0：JTAG IN FIFO 不为空

1：JTAG IN FIFO 为空

(RO)

USB_SERIAL_JTAG_IN_FIFO_FULL 表示 JTAG IN FIFO 是否为满。

0：JTAG IN FIFO 不为满

1：JTAG IN FIFO 为满

(RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT 表示 JTAG OUT FIFO 计数器。(RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY 表示 JTAG OUT FIFO 是否为空。

0：JTAG OUT FIFO 不为空

1：JTAG OUT FIFO 为空

(RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL 表示 JTAG OUT FIFO 是否为满。

0：JTAG OUT FIFO 不为满

1：JTAG OUT FIFO 为满

(RO)

USB_SERIAL_JTAG_IN_FIFO_RESET 配置是否复位 JTAG IN FIFO。

0：无效

1：复位

(R/W)

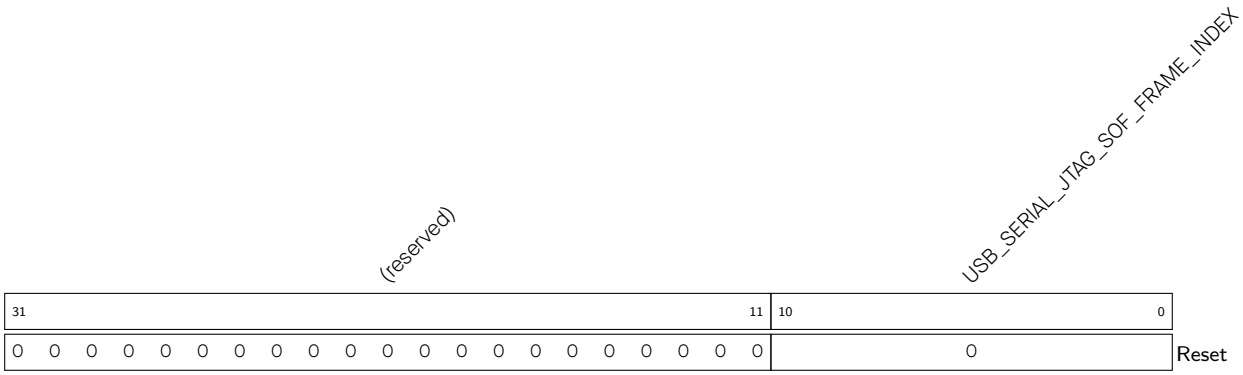
USB_SERIAL_JTAG_OUT_FIFO_RESET 配置是否复位 JTAG OUT FIFO。

0：无效

1：复位

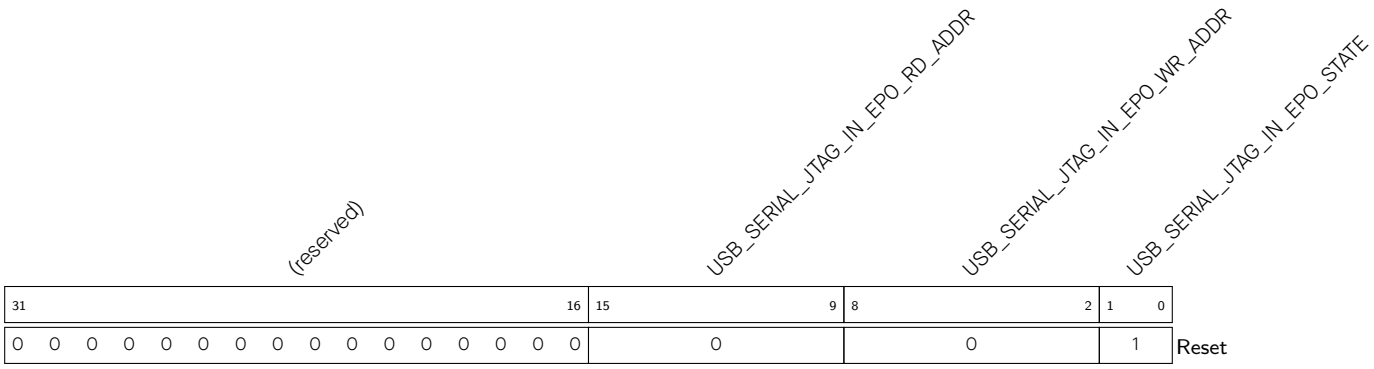
(R/W)

Register 54.17. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)



USB_SERIAL_JTAG_SOF_FRAME_INDEX 接收 SOF 帧的帧索引。(RO)

Register 54.18. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

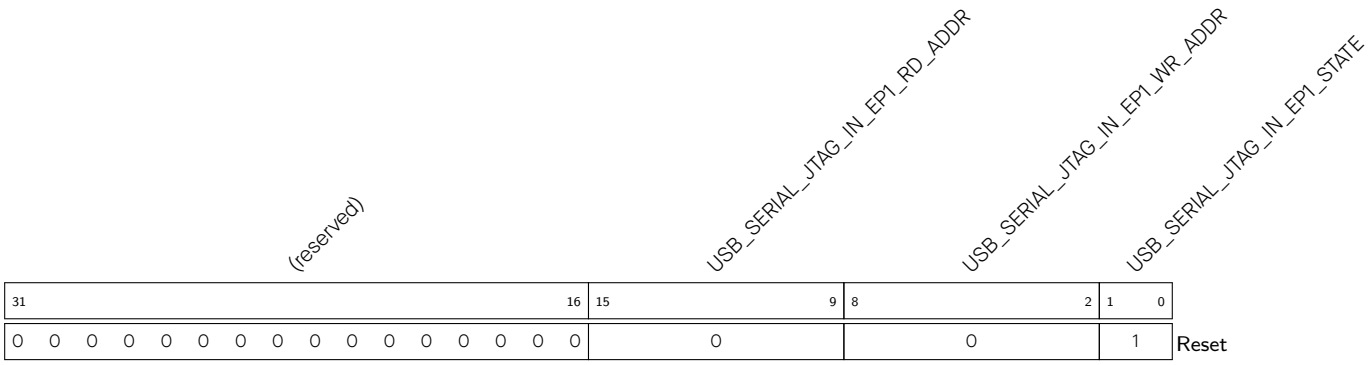


USB_SERIAL_JTAG_IN_EP0_STATE 表示输入端点 0 的状态。(RO)

USB_SERIAL_JTAG_IN_EP0_WR_ADDR 表示输入端点 0 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP0_RD_ADDR 表示输入端点 0 的读取数据地址。(RO)

Register 54.19. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

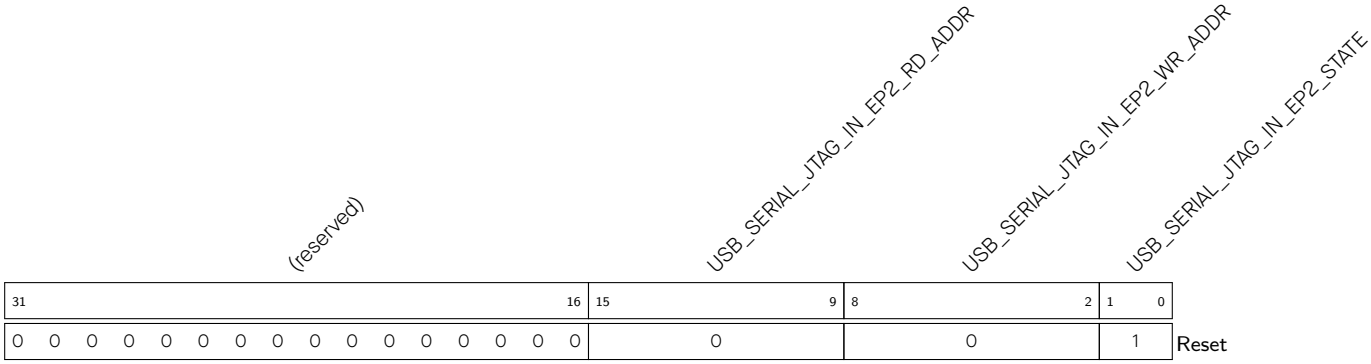


USB_SERIAL_JTAG_IN_EP1_STATE 表示输入端点 1 的状态。(RO)

USB_SERIAL_JTAG_IN_EP1_WR_ADDR 表示输入端点 1 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP1_RD_ADDR 表示输入端点 1 的读取数据地址。(RO)

Register 54.20. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

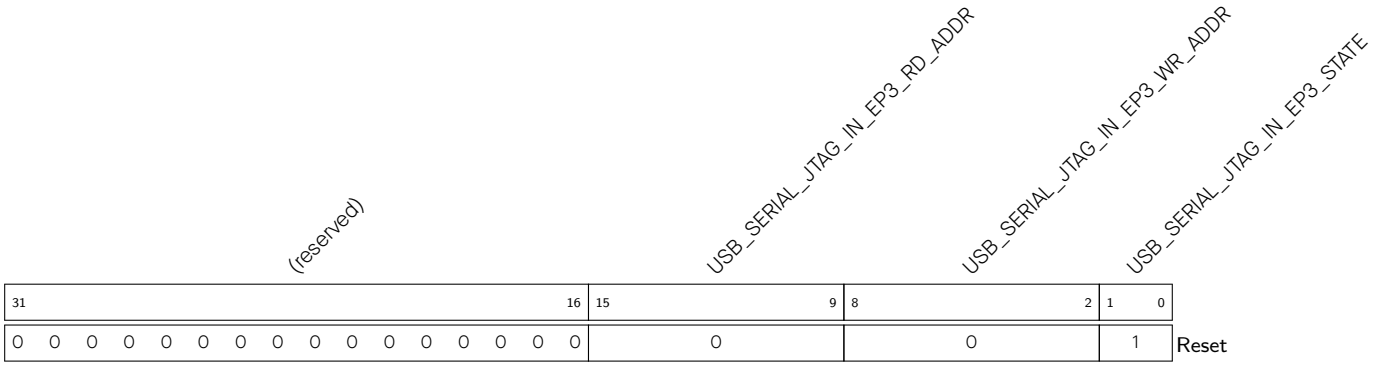


USB_SERIAL_JTAG_IN_EP2_STATE 表示输入端点 2 的状态。(RO)

USB_SERIAL_JTAG_IN_EP2_WR_ADDR 表示输入端点 2 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP2_RD_ADDR 表示输入端点 2 的读取数据地址。(RO)

Register 54.21. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

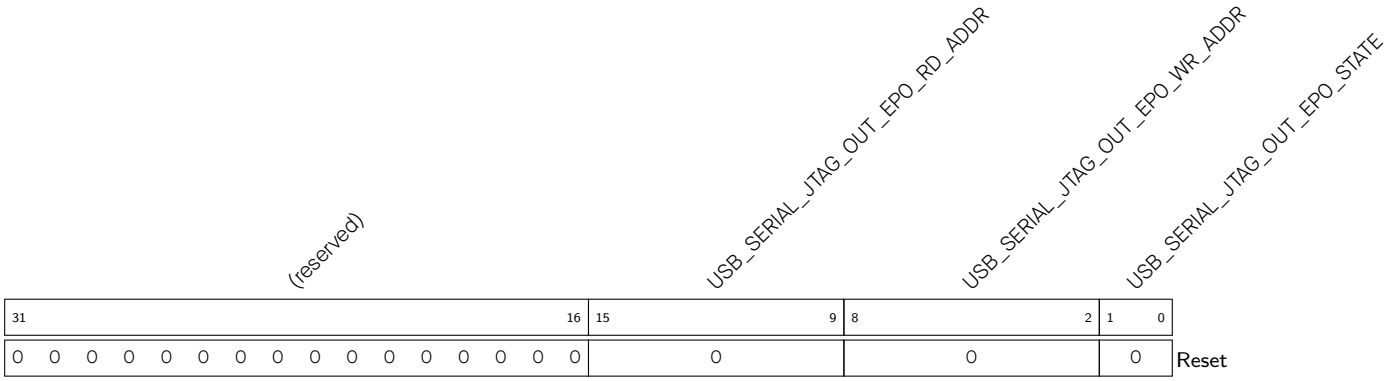


USB_SERIAL_JTAG_IN_EP3_STATE 表示输入端点 3 的状态。(RO)

USB_SERIAL_JTAG_IN_EP3_WR_ADDR 表示输入端点 3 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP3_RD_ADDR 表示输入端点 3 的读取数据地址。(RO)

Register 54.22. USB_SERIAL_JTAG_OUT_EP0_ST_REG (0x0038)

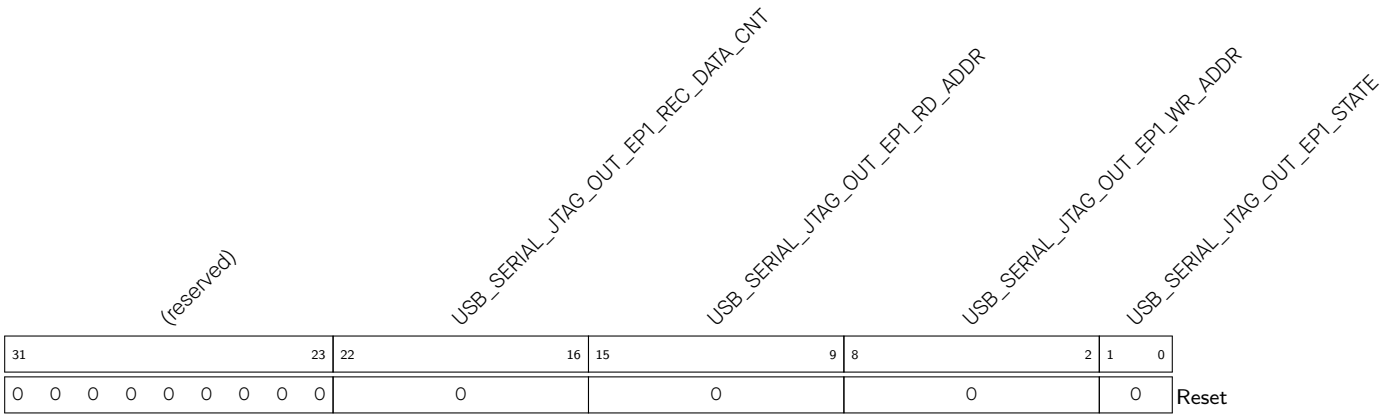


USB_SERIAL_JTAG_OUT_EP0_STATE 表示输出端点 0 的状态。(RO)

USB_SERIAL_JTAG_OUT_EP0_WR_ADDR 表示输出端点 0 的写入数据地址。
当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 0 中有 (USB_SERIAL_JTAG_OUT_EP0_WR_ADDR − 2) 个字节数据。
(RO)

USB_SERIAL_JTAG_OUT_EP0_RD_ADDR 表示输出端点 0 的读取数据地址。(RO)

Register 54.23. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



USB_SERIAL_JTAG_OUT_EP1_STATE 表示输出端点 1 的状态。(RO)

USB_SERIAL_JTAG_OUT_EP1_WR_ADDR 表示输出端点 1 的写入数据地址。

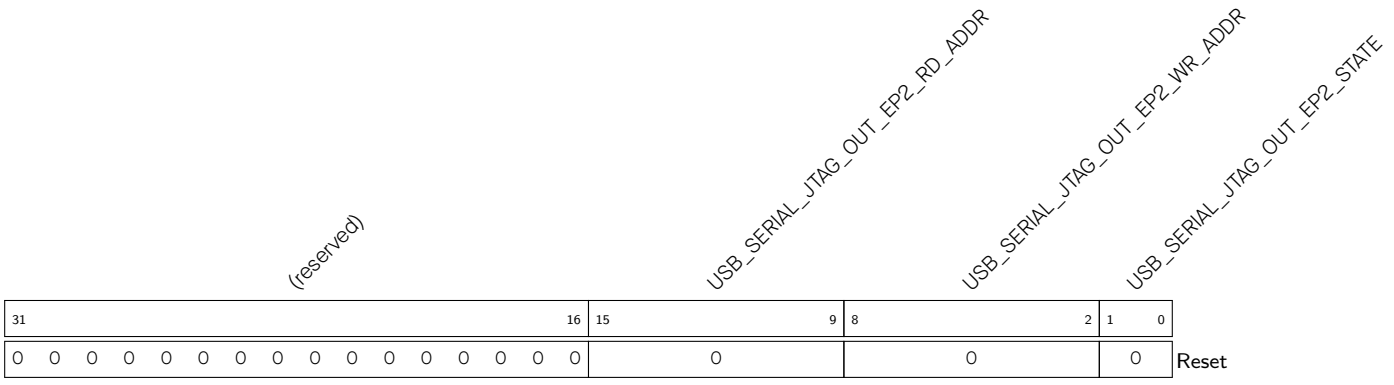
当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 1 中有 (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR – 2) 个字节数据。

(RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR 表示输出端点 1 的读取数据地址。(RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT 当接收到一个数据包时输出端点 1 中的数据计数器。(RO)

Register 54.24. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)



USB_SERIAL_JTAG_OUT_EP2_STATE 表示输出端点 2 的状态。(RO)

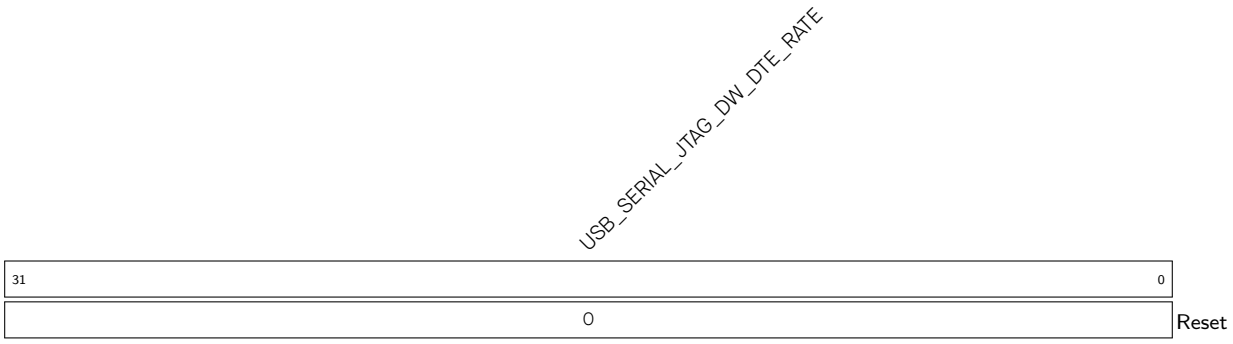
USB_SERIAL_JTAG_OUT_EP2_WR_ADDR 表示输出端点 2 的写入数据地址。

当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 2 中有 (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR – 2) 个字节数据。

(RO)

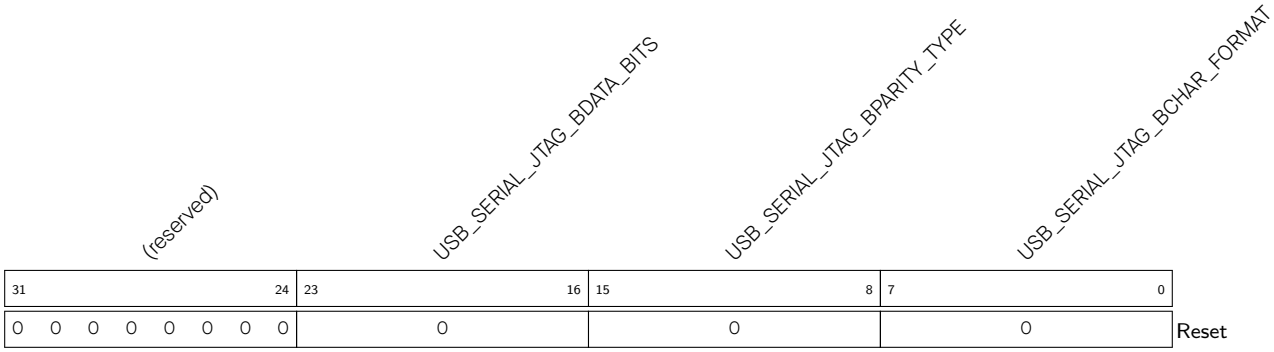
USB_SERIAL_JTAG_OUT_EP2_RD_ADDR 表示输出端点 2 的读取数据地址。(RO)

Register 54.25. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE 表示主机通过 SET_LINE_CODING 命令设置的 dwDTERate 值。(RO)

Register 54.26. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

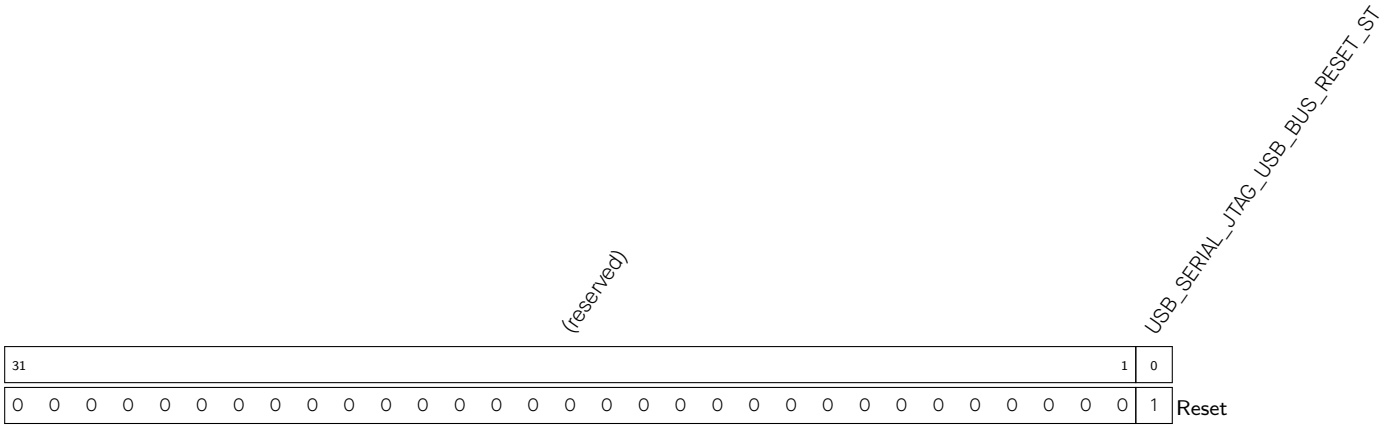


USB_SERIAL_JTAG_BCHAR_FORMAT 表示主机通过 SET_LINE_CODING 命令设置的 bCharFormat 值。(RO)

USB_SERIAL_JTAG_BPARITY_TYPE 表示主机通过 SET_LINE_CODING 命令设置的 bParityTpye 值。(RO)

USB_SERIAL_JTAG_BDATA_BITS 表示主机通过 SET_LINE_CODING 命令设置的 bDataBits 值。(RO)

Register 54.27. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



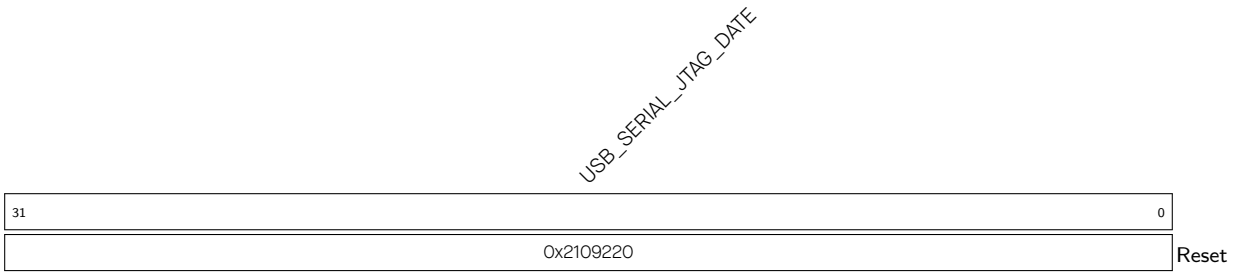
USB_SERIAL_JTAG_USB_BUS_RESET_ST 表示 USB 总线复位是否被释放。

0：USB 串口/JTAG 控制器处于 USB 总线复位状态

1：USB 总线复位被释放

(RO)

Register 54.28. USB_SERIAL_JTAG_DATE_REG (0x0088)



USB_SERIAL_JTAG_DATE 版本控制寄存器。(R/W)

第 55 章

以太网介质访问控制器 (EMAC)

55.1 概述

借助外部以太网物理层 (Ethernet PHY)，ESP32-P4 可以通过以太网介质访问控制 (Ethernet MAC) 按照 IEEE 802.3 标准发送和接收数据。详见图 55.1-1。以太网是当前应用最普遍的局域网 (LAN) 和广域网 (WAN) 进行数据传播的网络协议。

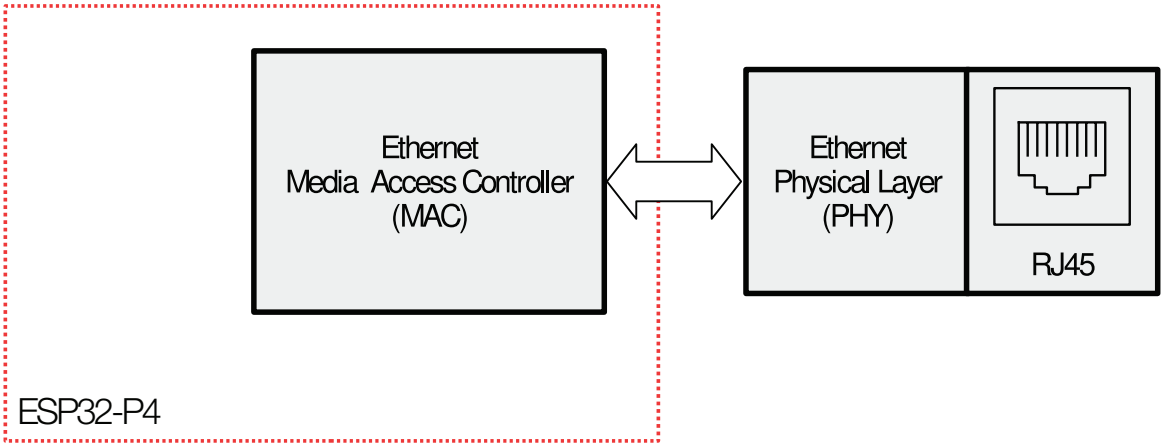


图 55.1-1. Ethernet MAC 功能概述

ESP32-P4 以太网 MAC 符合以下标准：

- 符合 IEEE 802.3-2002，用于以太网 MAC。
- 符合 IEEE 1588-2008 标准，用于规定联网时钟同步的精度。
- 符合 IEEE 802.3 规范工业标准接口：介质独立接口 (MII) 和简化介质独立接口 (RMII)。
- 符合 IEEE 802.3az - 2010 节能以太网标准
- 符合 IEEE 802.1Q 标准，用于支持 VLAN 帧

55.2 主要特性

ESP32-P4 以太网 MAC 具有以下特点：

- 支持外部 PHY 接口实现 10/100 Mbit/s 数据传输速率
- 可通过符合 IEEE802.3 的 MII 接口和 RMII 接口与外部快速以太网 PHY 进行通信
- 支持全双工和半双工模式
  - 支持适用于半双工模式的 CSMA/CD 协议

  - 支持适用于全双工模式的 IEEE 802.3x 流量控制
  - 全双工模式时可以将接收的暂停控制帧转发到用户应用程序
  - 半双工模式时提供背压流量控制
  - 全双工操作中如果流量控制输入信号消失，将自动发送暂停时间为零的暂停帧
- 报头和帧起始数据 (SFD) 在发送路径中插入、在接收路径中删除
- 可逐帧控制 CRC 和 padding（全 0）自动生成
- 如果数据为达到最小帧长度，则自动添加 padding
- 可编程帧长度，支持高达 16 KB 的巨型帧
- 可编程帧间隔 (IFG)（40-96 位时间，以 8 为步长）
- 支持多种灵活的地址过滤模式:
  - 高达 8 个 48 位完美地址过滤器，对每个字节进行掩码操作
  - 高达 8 个 48 位 SA 地址比较检查，对每个字节进行掩码操作
  - 可传送所有多播地址帧
  - 支持混合模式，因此可传送所有帧，无需为网络监视进行过滤
  - 传送所有传入数据包时（每次过滤时）均附有一份状态报告
- 为发送和接收数据包分别返回 32 位状态
- 在接收功能中支持 VLAN 标记帧过滤
- 为应用程序提供单独的发送、接收和控制接口
- 使用 MDIO 接口配置和管理 PHY 设备
- 在接收功能中支持对接收到的由以太网帧封装的 IPv4 和 TCP 数据包进行校验和卸载
- 在接收功能中支持检查 IPv4 头校验和以及在 IPv4/IPv6 数据包中封装的 TCP、UDP 或 ICMP 校验和
- 支持以太网帧时间戳（详细参考 IEEE 1588-2008）。每个帧在发送或接收时带有 64 位时间戳。
- 支持节能以太网（详细参考 IEEE 802.3az-2010）
- 支持传输帧中 CRC 替换、源地址字段插入或替换以及 VLAN 插入、替换或删除
- 两组 FIFO：一个 256 字节发送 FIFO 和一个 256 字节接收 FIFO
- 接收 FIFO 进行多帧存储时，在 EOF 传输后，通过向接收 FIFO 插入接收状态矢量，从而使得接收 FIFO 无需存储这些帧的接收状态
- 可以转发过小的好帧
- 为接收 FIFO 中由于溢出丢失或损坏的帧生成脉冲，借此支持数据统计
- 发送时处理冲突帧的自动重新发送（符合一定条件，详见 55.4.1.1）
- 丢弃延迟冲突、过度冲突、过度延迟和下溢条件下的帧
- 通过软件控制刷新 TX FIFO

55.3 EMAC 架构

EMAC 结构框图如图 55.3-1 所示。

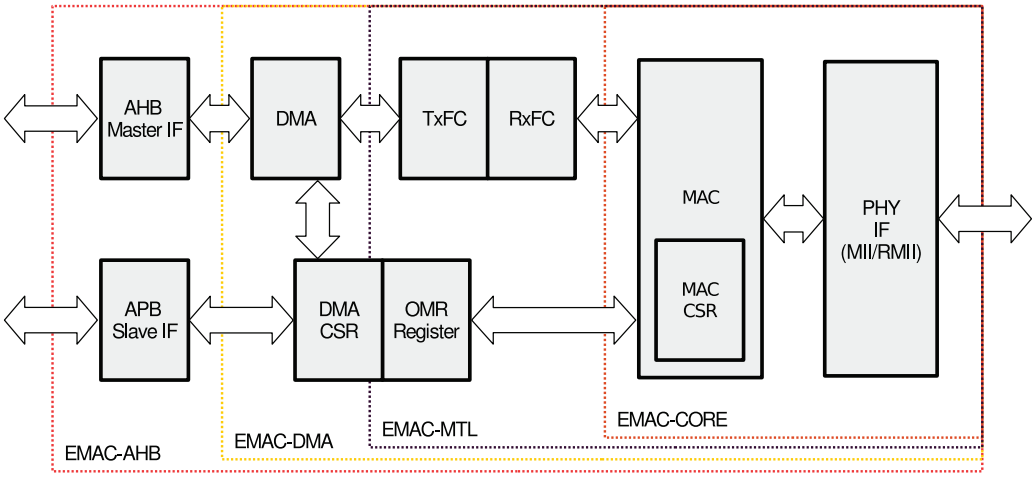


图 55.3-1. EMAC 功能框图

Ethernet MAC 层主要包括 EMAC_CORE、EMAC_MTL (MAC Transition Layer)、EMAC_DMA (Direct Memory Access) 三层，它们分别有 RX 和 TX 两个方向，在芯片内部通过 AHB 和 APB 总线和系统相连接，在芯片外部通过 MII 和 RMII 接口和外部的 PHY 进行通信，最终实现以太网的功能。

55.4 功能描述

55.4.1 EMAC_CORE

MAC 支持两种兼容 PHY 芯片的接口（详见章节 55.4.4）。芯片复位后 PHY 接口只能选一次。MAC 使用 MAC 发送接口 (MTI)，MAC 接收接口 (MRI) 和 MAC 控制接口 (MCI) 与应用侧（DMA 侧）进行通信。

55.4.1.1 传输操作

当 MTL 应用程序推送数据并且 SOF（帧起始）信号拉高时，启动发送操作。当检测到 SOF 信号时，MAC 接收数据并开始发送到 RMII 或 MII。应用程序启动传输之后将帧数据发送到 RMII 或 MII 所需的时间取决于延迟因素，如 IFG 延迟，发送报头或 SFD（起始帧分隔符）的时间以及半双工的任何回退延迟模式。在此之前，通过拉低数据就绪信号，MAC 不接收从 MTL 接收到的数据。

在 EOF（帧结束）发送到 MAC 之后，MAC 完成正常传输，并将传输状态 (Transmit Status) 提供给 MTL。如果在传输过程中（在半双工模式下）发生正常的冲突，则 MAC 使 MTL 的发送状态有效。然后它接受并删除所有剩下的数据，直到接收到下一个 SOF。在传输状态中检测到 MAC 发出的重试请求时，MTL 模块应该从 SOF 开始重传相同的帧。

如果 MTL 不能在传输期间连续提供数据，则 MAC 会发出下溢状态。在从 MTL 正常传输帧的过程中，如果 MAC 接收到一个 SOF 而没有得到前一帧的 EOF，则忽略该 SOF，并将该新帧视为前一帧的延续。

发送流量控制

在全双工模式下，当发送流量控制使能位（TFCE）置 1 时，MAC 将生成暂停帧并根据需要发送暂停帧。暂停帧与计算出的 CRC 附加在一起并发送。

可以通过两种方式启动暂停帧的生成。当应用程序将 FCBBA 位置 1 或接收 FIFO 已满时，将发送暂停帧。

- 如果应用程序已通过将寄存器 FCBBA 置 1 来请求流量控制，则 MAC 将生成并发送单个暂停帧。生成的帧中的暂停时间值为 PAUSE_TIME 字段中编程的暂停时间值。要在先前发送的暂停帧中指定的时间之前延长或结束暂停时间，应用程序必须以适当的值编程暂停时间值（PAUSE_TIME），然后请求另一次暂停帧发送。
- 如果接收 FIFO 填满时应用程序已请求流量控制，则 MAC 将生成并发送暂停帧。生成的帧中的暂停时间值为 PAUSE_TIME 字段中编程的暂停时间值。如果在此暂停时间结束前，接收 FIFO 在可配置的时隙数（PLT）期间保持填满状态，将发送第二个暂停帧。只要接收 FIFO 保持填满状态，该过程将一直重复下去。如果在采样时间之前不再满足此条件，MAC 将发送一个暂停时间为零的暂停帧，向远程端表明接收缓冲区已准备好接收新数据帧。

冲突期间的重新发送

在半双工模式下，向 MAC 传输帧时，可能在 MAC 线接口上发生冲突事件。MAC 甚至会在接收到帧结束之前就给出状态来指示重试。然后将使能重新发送并再次将帧从 FIFO 中弹出。当超过 96 个字节弹向 MAC 内核后，FIFO 控制器将释放该空间，使 DMA 可推入更多数据。这意味着超过阈值后或 MAC 内核指示延迟冲突事件时，无法重新发送。

碰撞、TX FIFO 下溢、载波丢失、jabber 超时、无载波、过度延迟或迟冲突都可能导致 MAC 发送器中止帧传输。当帧传输由于冲突而中止时，MAC 请求重发帧。

发送状态字

以太网帧发送结束时，MAC 通过 EMAC_DMA 向应用输出发送状态。发送状态的详细说明与 TDESO 相同。

55.4.1.2 接收操作

当 MAC 在 RMII 或 MII 上检测到 SFD 时启动接收操作。在继续处理帧之前，MAC 剥去前导码和 SFD。检查用于过滤的报头字段和用于验证数据帧 CRC 的 FCS 字段。接收的帧存储在缓冲器中，直到执行地址过滤。如果帧未通过地址过滤，则帧丢弃在 MAC 中。

MAC 接收的帧将推入 RX FIFO。此 FIFO 的状态一旦超过配置的接收阈值（RX_THRESH_CTRL），并通知给 DMA，这样 DMA 可向 AHB 接口发起预配置的突发传输。

在默认直通模式下，当 FIFO 接收到 64 个字节（RX_THRESH_CTRL）或完整的数据包时，数据将弹出，其可用性将通知给 DMA。DMA 向 AHB 接口发起传输后，数据传输将从 FIFO 持续进行，直到传输完整个数据包。完成帧 EOF 的传输后，状态字将弹出并发送到 DMA 控制器。

接收协议

接收模块接收到包之后，去除接收的帧的报头和 SFD。检测到 SFD 后，MAC 开始向接收 FIFO 发送以太网帧数据，从 SFD 后面的第一个字节（目标地址）开始发送。如果使能 IEEE 1588 时间戳功能，则在 MII 上检测到任何帧的 SFD 时，都将获取系统时间的快照。除非 MAC 滤出并丢弃帧，否则此时间戳将传递给应用程序。

如果接收的帧长度/类型字段小于 0x600 并且为 MAC 使能了自动去除 CRC/padding 选项，则 MAC 将向接收 FIFO 发送帧数据（数据量不超过长度/类型字段中指定的数量），然后开始丢弃字节（包括 FCS 字段）。如果长度/类型字段大于或等于 0x600，则不管配置的自动 CRC 去除选项的值如何，MAC 都会向 RX FIFO 发送所有接收到的以太网帧数据。默认情况下，使能 MAC 看门狗定时器，即，超过 2048 个字节（DA+SA+LT+ 数据

+padding+FCS）帧会被切断。可通过对 MAC 配置寄存器中的看门狗禁止 (EMACWATCHDOG) 位编程来禁止此功能。但是，即使禁止看门狗定时器，仍将切断大于 16 KB 的帧并给出看门狗超时状态。

接收帧控制器

如果复位 MAC 帧过滤寄存器中的 RA (RECEIVE_ALL) 位，则 MAC 将根据目标/源地址执行帧过滤。如果应用程序决定不接收任何不良帧，如矮帧、CRC 错误帧等，则仍需要执行另一等级的过滤。检测到过滤失败时，帧将被丢弃且不会传输到应用程序。当过滤参数动态改变时，如果 (DA-SA) 过滤失败，则剩余的帧将被丢弃并且接收状态字将立即更新（零帧长度位、CRC 错误位和矮帧错误位将置 1），指示过滤失败。

接收流量控制

MAC 将检测接收暂停帧并暂停帧发送，暂停时间为接收的暂停帧内指定的延迟（仅限全双工模式）。可通过 RFCE 字段使能或禁止暂停帧检测功能。使能接收流量控制后，将开始监视接收帧的目标地址是否与控制帧的多播地址 (0x0180 C200 0001) 匹配。如果检测到匹配（接收的帧的目标地址与保留的控制帧的目标地址匹配），MAC 将根据 PCF 字段来决定是否将接收的控制帧传输应用程序。

MAC 还将对接收控制帧的类型、操作码和暂停定时器字段进行解码。如果状态的字节计数指示 64 个字节，并且不存在任何 CRC 错误，则 MAC 发送器将暂停任何数据帧的发送，暂停时间为解码的暂停时间值乘以时隙（对于 10/100 Mbit/s 模式，均为 64 字节）。同时，如果检测到另一个零暂停时间值的暂停帧，MAC 将复位暂停时间并管理新的暂停请求。

如果接收的控制帧与类型字段 (0x8808)、操作码 (0x00001) 以及字节长度（64 字节）均不匹配，或存在 CRC 错误，则 MAC 不会暂停。

如果暂停帧具有多播目标地址，MAC 将根据地址匹配来过滤帧。

对于具有单播目标地址的暂停帧，MAC 将根据 DA 是否与 MAC 地址 0 寄存器的内容匹配以及寄存器 UPFD 是否置 1（检测具有单播目标地址的暂停帧）来进行过滤。寄存器 PCF 可对控制帧的过滤以及地址过滤进行控制。

接收多帧的操作处理

由于接收数据后状态立即可用，因此只要 FIFO 未满，就可以向其中存储帧。

错误处理

如果在从 MAC 接收 EOF 数据之前 RX FIFO 已满，则将产生上溢并丢弃整个帧。由于上溢，状态位 (RDESO[11]) 将指示这是一个部分帧。如果使用寄存器 FLUSH_TX_FIFO 和 FWD_UNDER_GF 使能相应功能，RX FIFO 可过滤错误帧和过小帧。

在直通模式下，如果在从 RX FIFO 读取帧的 SOF 时，该帧的状态和长度可用，则可丢弃整个错误帧。软件可通过将 FWD_ERR_FRAME 置 0 使 DMA 清空正在从 FIFO 读取的错误帧。然后到应用的数据传输将停止，其余帧将从内部读取并丢弃。如果 FIFO 可用，则可以启动下一帧传输。

接收状态字

以太网帧接收结束时，MAC 通过 EMAC_DMA 向应用输出接收状态。接收状态的详细说明与 RDESO 中的位 [31:0] 相同。

55.4.2 EMAC_MTL（MAC 传输层）

MAC 传输层提供 FIFO 存储器来缓冲和调节应用系统存储器和 MAC 之间的帧。它还可以在应用时钟域和 MAC 时钟域之间传输数据。MTL 层具有两条数据路径，即发送路径和接收路径。双向数据路径位宽为 32，采用简单的 FIFO 协议操作。

55.4.3 EMAC_DMA

EMAC 自带 DMA 模块，具有独立的发送和接收引擎以及控制和状态寄存器地址。发送引擎将数据从系统内存传输到设备端口 (MTL)，而接收引擎将数据从设备端口传输到系统内存。控制器使用描述符，以最少的主机 CPU 干预来有效地将数据从源移动到目的地。DMA 用于以数据包为主的数据传输，如以太网中的帧。控制器可以配置为在正常情况下发中断给主 CPU，例如完成帧发送或接收，或发生错误时。

55.4.3.1 发送描述符

发送描述符结构如图 55.4-1 所示。表 55.4-1 至表 55.4-4 为链表描述符说明。

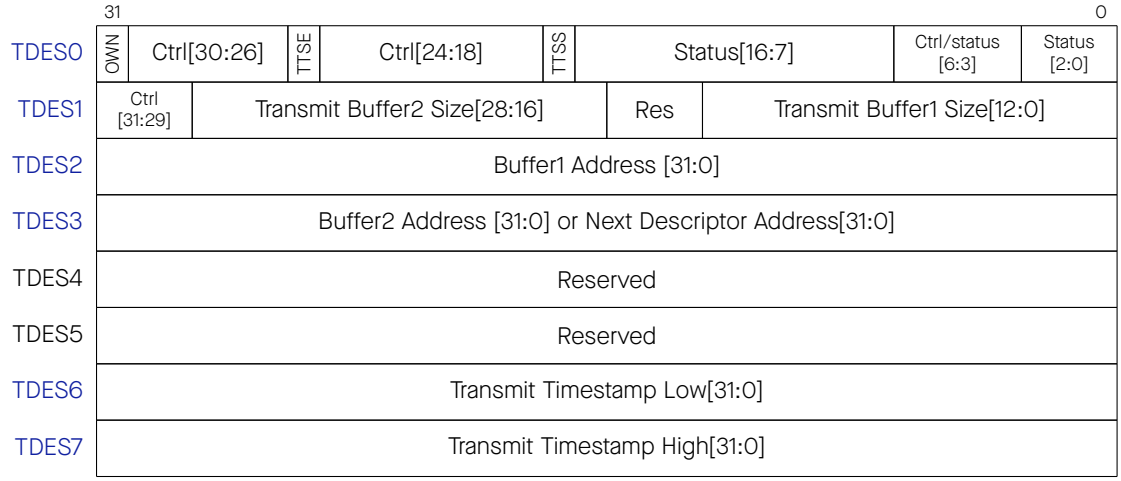


图 55.4-1. 发送描述符

表 55.4-1. 发送描述符 0 (TDES0)

位	命名	描述
[31]	OWN: Own Bit	该位置 1 时，表示描述符由 DMA 拥有。复位时，表示描述符由主机拥有。当 DMA 完成帧传输或分配在描述符中的缓存为空时，DMA 将清除该位。在设置属于同一帧的所有后续描述符之后，应设置帧的第一个描述符的控制位。这避免了在获取描述符和驱动程序设置控制位之间的可能发生的竞争。
[30]	IC: Interrupt on Completion	置 1 后，该位在发送当前帧之后设置发送中断 (TRANS_INT)。该位仅在位 TDES0[29] 置 1 时有效。
[29]	LS: Last Segment	置 1 时，该位表示缓存区中包含该帧的最后一段，TDES1 中的 TBS1 或 TBS2 字段的值不为零。
[28]	FS: First Segment	置 1 时，该位表示缓存区中包含帧的第一段。
[27]	DC: Disable CRC	该位置 1 时，MAC 不会在发送帧的末尾附加循环冗余校验 (CRC)。这仅在位 TDES0[28] 置 1 时有效。

位	命名	描述
[26]	DP: Disable padding	置 1 时，MAC 不会自动在短于 64 字节的帧后面自动添加 padding。当该位复位时，DMA 自动将 padding 和 CRC 添加到短于 64 个字节的帧中，并且无论 DC (TDES0[27]) 位的状态如何，都会添加 CRC 字段。只有当位 TDES0[28] 置 1 时才有效。
[25]	TTSE: Transmit Timestamp Enable	置 1 时，MAC 的传输帧启用 IEEE 1588 硬件时间戳。
[24]	CRCR: CRC Replacement Control	置 1 时，MAC 用重新计算的 CRC 字节替换发送的数据包的最后四个字节。主机应确保 CRC 字节在将要发送的帧的缓存中。当控制位 (TDES0[28]) 置 1 时，该位有效。另外，只有当位 TDES0[27] 置 1 时才进行 CRC 替换。
[23:22]	CIC: Checksum Insertion Control	这些位控制校验和的计算和插入。以下列表描述了位编码： • 00：关闭校验和插入。 • 01：仅使能 IP 报头校验和计算和插入。 • 10：使能 IP 报头校验和以及有效负载校验的计算和插入，伪报头校验不在硬件中计算。 • 11：使能 IP 报头校验和以及有效负载校验的计算和插入，但是伪报头校验在硬件中计算。 当控制位 TDES0[28] 置 1 时，此字段有效。
[21]	TER: Transmit End of Ring	置 1 时，该位表示描述符列表已达到最后一个链表描述符。DMA 返回到链表的基地址，创建一个描述符环。
[20]	TCH: Second Address Chained	置 1 时，该位指示描述符中的第二个地址是下一个描述符地址。当 TDES0[20] 置 1 时，TBS2 (TDES1[28:16]) 是一个“无关”值。TDES0[21] 优先于 TDES0[20]。此位必须置 1。
[19:18]	VLIC: VLAN Insertion Control	置 1 时，这些位请求 MAC 在传输帧之前打上 VLAN 标记或取消标记。如果帧被打上 VLAN 标记，则 MAC 会自动重新计算并替换 CRC 字节。以下为这些位的说明： • 00：不加上 VLAN 标记。 • 01：在传输之前去掉 VLAN 标记，只用于 VLAN 帧。 • 10：插入 VLAN 标记，标记的值在 VLAN 标记插入和替换寄存器中配置。 • 11：用 VLAN 标记插入和替换寄存器中的标记值替换帧当中的 VLAN 标记。该选项只能用于 VLAN 帧。
[17]	TTSS: Transmit Timestamp Status	该字段被用作状态位，以指示捕获了所描述的传输帧的时间戳。当该位置 1 时，TDES6 和 TDES7 具有为传输帧捕获的时间戳值。该字段仅在 LS（TDES0[29]）置 1 时有效。
[16]	IHE: IP Header Error	置 1 时，该位指示 MAC 发送器在 IP 报头中检测到错误。发送器检查 IPv4 数据包中的报头长度与从应用程序接收到的报头字节的数量，如果不匹配，则指示错误状态。对于 IPv6 帧，如果报头长度不是 40 个字节，则指示报头错误。此外，IPv4 或 IPv6 帧的“以太网长度/类型”字段值必须与接收到的 IP 报头版本匹配。对于 IPv4 帧，如果“报头长度”字段的值小于 0x5，则指示错误状态。

位	命名	描述
[15]	ES: Error Summary	此位代表以下 bit 的逻辑或: • TDES0[16]: IP 报头错误 • TDES0[14]: Jabber 超时 • TDES0[13]: 帧刷新 • TDES0[12]: IP 有效负载错误 • TDES0[11]: 载波丢失 • TDES0[10]: 无载波 • TDES0[9]: 延迟冲突 • TDES0[8]: 过度冲突 • TDES0[2]: 过度延迟 • TDES0[1]: 下溢错误
[14]	JT: Jabber Timeout	置 1 时，该位表示 MAC 发送器发生了 jabber 超时。只有在 EMACCONFIG_REG 的 EMACJABBER 位（关闭 Jabber）未置 1 时，该位置 1。
[13]	FF: Frame Flushed	置 1 时，该位表示 DMA 或 MTL 已经按照 CPU 给出的刷新命令刷新帧。
[12]	IPE: IP Payload Error	置 1 时，该位表示 MAC 发送器在 TCP，UDP 或 ICMP IP 数据报有效载荷中检测到错误。 发送器检查 IPv4 或 IPv6 报头中收到的有效负载长度与从应用程序接收到的 TCP，UDP 或 ICMP 数据包字节的实际数量，并在发生不匹配时指示错误状态。
[11]	LOC: Loss of Carrier	置 1 时，该位指示在帧传输期间发生载波丢失（即，在帧传输期间，一个或多个传输时钟周期内，MII_CRS 信号无效）。当 MAC 处于半双工模式时，只对传输中没有冲突的帧有效。
[10]	NC: No Carrier	置 1 时，该位指示在传输过程中来自 PHY 的载波侦听信号未被拉低。
[9]	LC: Late Collision	置 1 时，该位指示在冲突窗口（在 MII 模式下，包括报头在内共 64 字节时间）之后发生冲突，导致帧传输中止。如果下溢错误位置 1，则该位无效。
[8]	EC: Excessive Collision	置 1 时，该位表示在尝试传送当前帧发生 16 次连续冲突之后传输被中止。如果 EMACCONFIG_REG 的 EMACRETRY 位置 1，则该位在第一次冲突后置 1，并且帧传输被中止。
[7]	VF: VLAN Frame	置 1 时，表示传输的帧是 VLAN 类型的帧。
[6:3]	Ctrl/status	这些状态位指示帧发送之前发生的冲突次数。当过度冲突位 (TDES0[8]) 置 1 时，此计数无效。CPU 只在半双工模式下更新这个状态字段。
[2]	ED: Excessive Deferral	置 1 时，如果 MAC 配置寄存器 EMACCONFIG_REG 的 EMACDEFERRALCHECK（延迟检测）位置为高，则该位表示传输已经结束，原因是在使能巨帧的情况下，过度延迟超过 24,288 比特时间。
[1]	UF: Underflow Error	置 1 时，该位表示由于数据从主机存储器延迟到达，MAC 中止了帧发送。下溢错误表示 DMA 在传输帧时遇到空的传输缓存。传输过程进入暂停状态并将传输下溢寄存器（状态寄存器 TRANS_UNDFLOW）和传输中断寄存器（状态寄存器 TRANS_INT）置 1。

位	命名	描述
[0]	DB: Deferred Bit	置 1 时，该位表示 MAC 由于载波的存在而传输延迟。该位仅在半双工模式下有效。

表 55.4-2. 发送描述符 1 (TDES1)

位	名称	描述
[31:29]	SAIC: SA Insertion Control	这些位请求 MAC 使用 EMACADDR0HIGH_REG，EMACADDR0LOW_REG，EMACADDR1HIGH_REG，EMACADDR1LOW_REG 寄存器中给定的值添加或替换以太网帧中的源地址字段。如果“源地址”字段在帧中被修改，则 MAC 会自动重新计算并替换 CRC 字节。位 31 指定用于插入或替换源地址的 MAC 地址寄存器值（1 或 0）。以下列表描述了位 [30:29] 的值： • 00：不加入源地址。 • 01：插入源地址。为确保传输的可靠性，应用程序必须提供没有源地址的帧。 • 10：替换源地址。为确保传输的可靠性，应用程序必须提供带有源地址的帧。 • 11：保留 当控制位 TDES0[28] 置 1 时，这些位有效。
[28:16]	TBS2: Transmit Buffer2 Size	链表第二数据缓存的大小，以字节为单位。如果 TDES0[20] 置 1，则该字段无效。
[15:13]	Reserved	保留
[12:0]	TBS1: Transmit Buffer1 Size	链表第一数据缓存的大小，以字节为单位。如果该字段为 0，则 DMA 将忽略此缓存，并根据 TCH (TDES0[20]) 的值来确定使用第二缓存还是下一个描述符。

表 55.4-3. 发送描述符 2 (TDES2)

位	名称	描述
[31:0]	Buffer1 Address Pointer	第一缓存的物理地址。

表 55.4-4. 发送描述符 3 (TDES3)

位	名称	描述
[31:0]	Buffer2 Address Pointer (Next Descriptor Address)	使用描述符环结构时，表示第二缓冲的物理地址。如果 TCH（TDES0[20]）置 1，则该地址包含下一个描述符所在物理内存的指针。

表 55.4-5. 发送描述符 6 (TDES6)

位	名称	描述
[31:0]	TTSL: Transmit Frame Timestamp Low	该字段由 DMA 更新，为相应传输帧捕获的时间戳的最低有效 32 位。只有当描述符中的最后段 (TDESO[29]) 位被置 1 时且时间戳状态 (TDESO[17]) 位置 1 时，该字段才具有时间戳。

表 55.4-6. 发送描述符 7 (TDES7)

位	名称	描述
[31:0]	TTSH: Transmit Frame Timestamp High	该字段由 DMA 更新，为相应接收帧捕获的时间戳的最高有效 32 位。只有当描述符中的最后段 (TDESO[29]) 位被置 1 时且时间戳状态 (TDESO[17]) 位置 1 时，该字段才具有时间戳。

55.4.3.2 接收描述符

接收链表结构如图 55.4-2 所示。表 55.4-7 至表 55.4-11 为链表描述。

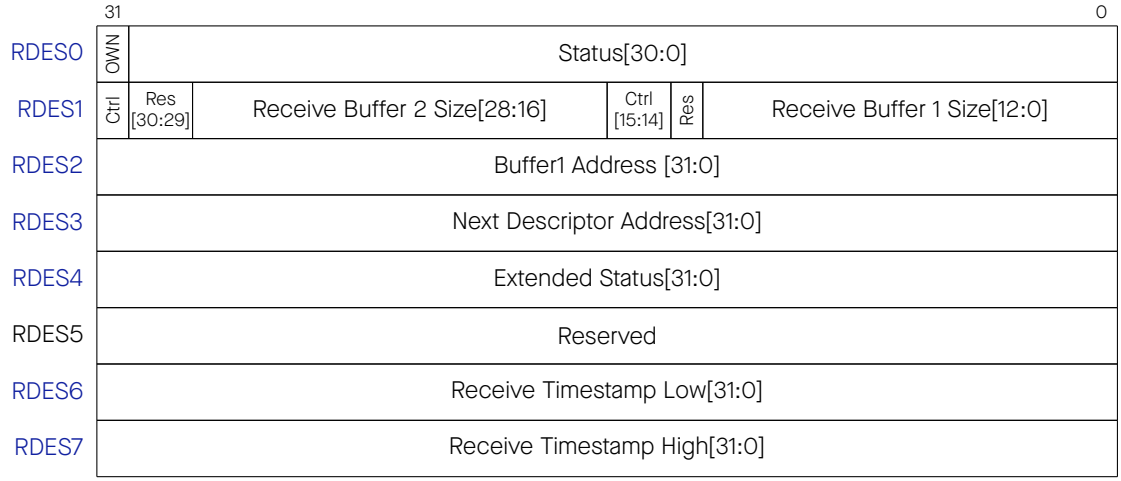


图 55.4-2. 接收链表结构

表 55.4-7. 接收描述符 0 (RDES0)

位	名称	描述
[31]	OWN: Own Bit	置 1 时，该位表示描述符由 DMA 所有。当该位复位时，该位表示描述符由主机拥有。DMA 在完成帧接收或与此链表有关的缓存已满时清除该位。
[30]	AFM: Destination Address Filter Fail	置 1 时，该位表示 MAC 中 DA 滤波器失败的帧。
[29:16]	FL: Frame Length	这些位表示发送到 CPU 内存的接收帧的字节长度。当 RDESO[8] 被置 1 且描述符错误位 RDESO[14] 或溢出错误位 RDESO[11] 被复位时，该字段有效。当使能 IP 校验和计算（类型 1）并且所接收的帧不是 MAC 控制帧时，帧长度还包括附加到以太网帧的两个字节。

位	名称	描述
[15]	ES: Error Summary	该字段表示以下位的逻辑或： • RDES0[1]：CRC 错误 • RDES0[3]：接受错误 • RDES0[4]：看门狗超时 • RDES0[6]：延迟冲突 • RDES0[7]：巨帧 • RDES4[4:3]：IP 报头或负载错误 • RDES0[11]：上溢错误 • RDES0[14]：描述符错误 只有 RDES0[8] 置 1 时，该字段有效。
[14]	DE: Descriptor Error	置 1 时，该位表示由帧大小超过当前描述符缓存的帧被截断，并且 DMA 不拥有下一个描述符。该帧被截断。只有当位 RDES0[8] 置 1 时，该字段才有效。
[13]	SAF: Source Address Filter Fail	置 1 时，该位表示帧的 SA 字段未通过 MAC 中的 SA 过滤。
[12]	LE: Length Error	置 1 时，该位表示接收到的帧的实际长度和长度/类型字段不匹配。该位仅在帧类型 (RDES0[5]) 位复位时有效。
[11]	OE: Overflow Error	置 1 时，该位表示由于 MTL 中的缓存溢出而导致接收到的帧被损坏。
[10]	VLAN: VLAN Tag	置 1 时，该位表示该描述符所指向的帧是由 MAC 标记的 VLAN 帧。VLAN 标记取决于根据 VLAN 标记寄存器设置检查接收到的帧的 VLAN 字段。
[9]	FS: First Descriptor	置 1 时，该位表示该描述符包含帧的第一个缓存区。如果第一个缓存区的大小是 0，则该帧从第二个缓存开始。如果第二个缓存的大小也是 0，则下一个描述符包含该帧的帧头。
[8]	LS: Last Descriptor	置 1 时，该位表示该描述符指向的缓存是该帧的最后一个缓存区。
[7]	IP Checksum Error (Type1), or Giant Frame	高级时间戳功能存在时，置 1 时，该位表示时间戳的快照写入描述符字 6 (RDES6) 和 7 (RDES7)。只有位 RDES0[8] 被置 1 时才有效。当选择 IP 校验和引擎（类型 1）时，该位置 1 表示以下情况之一： • MAC 内核计算的 16 位 IPv4 报头校验和与收到的校验和字节不匹配。 • 非 IPv4 帧绕过报头校验和检查。 若以上两种情况都不符合，则该位置 1 表示巨帧状态。对于普通帧，大于 1,518 字节的帧为巨帧（若该普通帧为 VLAN 帧，则大于 1,522 字节的帧为巨帧；当 ASS2KP 为 1 时，大于 2,000 字节的帧为巨帧）。当使能巨帧处理时，大于 9,018 字节的帧为巨帧（若该巨帧为 VLAN 帧，则大于 9,022 字节的帧为巨帧）。
[6]	LC: Late Collision	置 1 时，该位表示在半双工模式下接收帧时发生延迟冲突。
[5]	FT: Frame Type	置 1 时，该位表示接收帧是以太网类型的帧（LT 字段大于或等于 1,536 字节）。当该位复位时，表示接收到的帧是 IEEE 802.3 帧。此位对于小于 14 个字节的矮帧无效。
[4]	RWT: Receive Watchdog Timeout	置 1 表示接收看门狗定时器在收到当前帧时已经超时，当前帧在看门狗超时后被截断。

位	名称	描述
[3]	RE: Receive Error	置 1 时，该位表示在接收帧期间若 MII_RXDV 被置 1，则发出 MII_RXER。
[2]	DE: Dribble Bit Error	置 1 时，该位表示接收到的帧具有非整数倍的字节（奇数个半字节）。该位仅在 MII 模式下有效。
[1]	CE: CRC Error	置 1 时，该位表示在接收到的帧上发生循环冗余校验 (CRC) 错误。只有位 RDES0[8] 置 1 时，该字段才有效。
[0]	Extended Status Available/ RX MAC Address	当 IP 校验和卸载（类型 2）存在时，该位置 1 时表明扩展状态在描述符字 4 (RDES4) 中可用。只有位 RDES0[8] 置 1 时才有效。位 30 置 1 时该位无效。 当 IP 校验和卸载（类型 2）存在时，即使在 IP 校验和卸载引擎绕过接收帧的处理，该位也被置 1。绕过可能是因为非 IP 帧或非 TCP/UDP/ICMP 有效载荷的 IP 帧。 当 IPC 全部卸载未被选择时，该位表示 RX MAC 地址状态。置 1 时，该位表示 RX MAC 地址寄存器值（1 至 15）与帧的 DA 字段相匹配。复位时，该位表示 RX MAC 地址寄存器 0 的值与 DA 字段匹配。

表 55.4-8. 接收描述符 1 (RDES1)

位	名称	描述
[31]	Ctrl	置 1 时，该位阻止帧的状态寄存器的 RI 位 (RECV_INT) 被置 1，使得收到的帧在当前描述符所指的缓存内结束。从而禁止由于该帧的 RI 向主机的中断触发。
[30:29]	Reserved	保留
[28:16]	RBS2: Receive Buffer 2 Size	表示第二个数据缓存的大小（字节）。即使 RDES3（buffer2 地址指针）的值未对齐，缓存大小也必须是 4 的倍数。当缓存大小不是 4 的倍数时，结果是不确定的。如果 RDES1[14] 置 1，则该字段无效。
[15]	RER: Receive End of Ring	置 1 时，该位表示描述符列表已达到其最后一个描述符。DMA 返回到列表的基地址，创建一个描述符环。
[14]	RCH: Second Address Chained	置 1 时，该位表示描述符中的第二个地址是下一个描述符地址，此位必须置 1。当该位置 1 时，RBS2 (RDES1[28:16]) 是一个“无关”值。RDES1[15] 优先级高于 RDES1[14]。
[13]	Reserved	保留
[12:0]	RBS1: Receive Buffer 1 Size	表示第一个数据缓存的大小（字节）。即使 RDES2（buffer1 地址指针）的值未对齐，缓存大小也必须是 4 的倍数。当缓存大小不是 4 的倍数时，结果是不确定的。如果该字段为 0，则 DMA 将忽略此缓存，并根据 RCH 的值 (bit[14]) 使用缓存 2 或下一个描述符。

表 55.4-9. 接收描述符 2 (RDES2)

位	名称	描述
[31:0]	Buffer 1 Address Pointer	这些位表示第一缓存的物理地址。

表 55.4-10. 接收描述符 3 (RDES3)

位	名称	描述
[31:0]	Buffer 2 Address Pointer (Next Descriptor Address)	使用描述符环结构时，表示第二缓冲的物理地址。如果 RCH (RDES1[14]) 置 1，则该地址包含下一个描述符所在物理内存的指针。

表 55.4-11. 接收描述符 4 (RDES4)

位	名称	描述
[31:28]	Reserved	保留
[27:26]	Reserved	保留
[25]	Reserved	保留
[24]	Reserved	保留
[23:21]	Reserved	保留
[20:18]	Reserved	保留
[17]	Reserved	保留
[16]	Reserved	保留
[15]	Reserved	保留
[14]	Timestamp Dropped	该位置 1 时，表示已捕获该帧的时间戳，但由于溢出而在 MTL RX FIFO 中被丢弃。
[13]	PTP Version	该位置 1 时，指示接收到的 PTP 消息是 IEEE 1588 版本 2 格式。该位清零后，它是版本 1 格式。
[12]	PTP Frame Type	该位置 1 时，指示 PTP 消息直接通过以太网发送。当该位清零且消息类型非零时，表示 PTP 消息通过 UDP-IPv4 或 UDP-IPv6 发送。
[11:8]	Message Type	这些位被编码成所接收消息的类型。 • 0000：未收到 PTP 消息 • 0001：SYNC（所有时钟类型） • 0010：Follow_Up（所有时钟类型） • 0011：Delay_Req（所有时钟类型） • 0100：Delay_Resp（所有时钟类型） • 0101：Pdelay_Req（点对点透明时钟） • 0110：Pdelay_Resp（点对点透明时钟） • 0111：Pdelay_Resp_Follow_Up（点对点透明时钟） • 1000：Announce • 1001：Management • 1010：Signaling • 1011-1110：保留 • 1111：Reserved 消息类型的 PTP 包
[7]	IPv6 Packet Received	置 1 时，该位表示接收到的数据包是一个 IPv6 数据包。该位仅在寄存器（MAC 配置寄存器）的 bit[10] (EMACRXIPCOFFLOAD) 置 1 时更新。
[6]	IPv4 Packet Received	置 1 时，表示接收到的数据包是 IPv4 数据包。该位仅在寄存器（MAC 配置寄存器）的 bit[10] (EMACRXIPCOFFLOAD) 置 1 时更新。

位	名称	描述
[5]	IP Checksum Bypassed	置 1 时，该位表示校验和卸载引擎被旁路。
[4]	IP Payload Error	置 1 时，该位表示 MAC 内核计算的 16 位 IP 有效负载校验和（即 TCP，UDP 或 ICMP 校验和）与接收的段中对应的校验和字段不匹配。当 TCP，UDP 或 ICMP 段的长度与 IP Header 字段中的负载长度值不匹配时，该位置 1。当 bit[7] 或 bit[6] 置 1 时，该位有效。
[3]	IP Header Error	置 1 时，该位表示由 MAC 内核计算的 16 位 IPv4 报头校验和与接收的校验和字节不匹配，或 IP 数据报版本与以太网类型值不一致。当 bit[7] 或 bit[6] 被置 1 时，该位有效。
[2:0]	IP Payload Type	这些位表示封装在由接收校验和卸载引擎 (COE) 处理的 IP 数据报中的有效载荷的类型。如果 COE 由于 IP 报头错误或分段 IP 而不处理 IP 数据报的有效载荷，COE 也将这些位置为 00。 • 000：未知或未处理 IP 负载 • 001：UDP • 010：TCP • 011：ICMP • 1xx：保留 当 Bit[7] 或 Bit[6] 置 1 时，该位有效。

表 55.4-12. 接收描述符 6 (RDES6)

位	名称	描述
[31:0]	RTSH: Receive Frame Timestamp Low	该字段用对应接收帧捕获的时间戳的最低有效 32 位。该字段仅由最后一个描述符状态位 RDESO[8] 指示的接收帧的最后一个链表通过 DMA 更新。

表 55.4-13. 接收描述符 7 (RDES7)

位	名称	描述
[31:0]	RTSH: Receive Frame Timestamp High	该字段由 DMA 更新，用对应接收帧捕获的时间戳的最高有效 32 位更新。该字段仅由最后一个描述符状态位 RDESO[8] 指示的接收帧的最后一个链表通过 DMA 更新。

55.4.4 PHY 接口

MAC 和外部 PHY 接口通信支持以下两个接口：

- MII（介质独立接口）
- RMII（精简介质独立接口）

55.4.4.1 MII（介质独立接口）

介质独立接口 (MII) 定义了 10 Mbit/s 和 100 Mbit/s 的数据传输速率下 MAC 子层与 PHY 之间的互连。

MII 与 PHY 间的接口信号

MII 接口信号如图 55.4-3 所示。

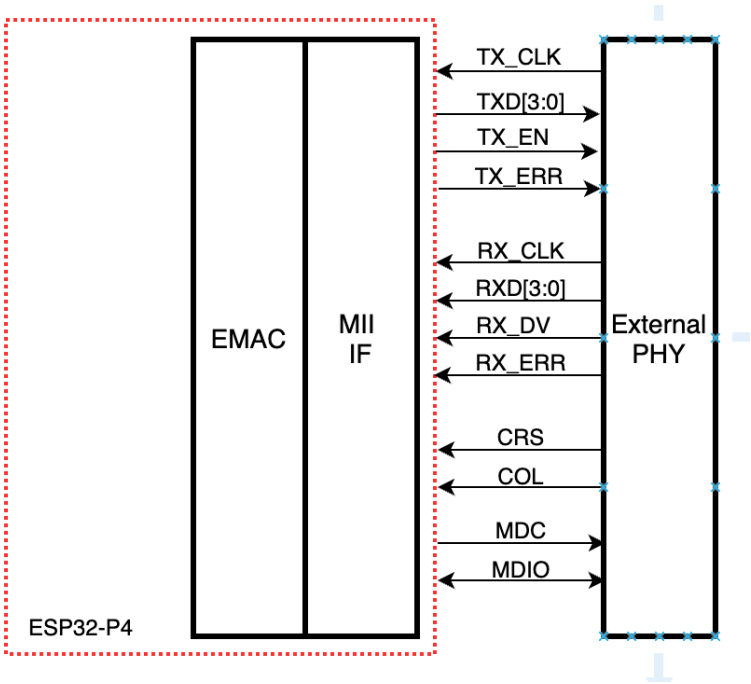


图 55.4-3. MII 接口

MII 接口信号说明：

- MII_TX_CLK：TX 时钟信号。该信号提供进行 TX 数据传输时的参考时序。频率分为两种：速率为 10 Mbit/s 时为 2.5 MHz；速率为 100 Mbit/s 时为 25 MHz。
- MII_TXD[3:0]：发送数据信号。该信号是 4 个一组的数据信号，由 MAC 子层同步驱动，在 MII_TX_EN 信号有效时才为有效信号（有效数据）。MII_TXD[0] 为最低有效位，MII_TXD[3] 为最高有效位。MII_TX_EN 信号为低时，发送数据不会对 PHY 产生任何影响。
- MII_TX_EN：发送数据使能信号。该信号表示 MAC 当前正针对 MII 发送半字节 (4 bits)。该信号必须与报头的前半字节进行同步 (MII_TX_CLK)，并在所有待发送的半字节均发送到 MII 时必须保持同步。
- MII_TX_ERR：发送错误信号。该信号必须保持一个或多个周期 (MII_TX_CLK)，从而向 PHY 层指示在帧的某处有错误。
- MII_RX_CLK：RX 时钟信号。该信号提供进行 RX 数据传输时的参考时序。频率也分为两种：速率为 10 Mbit/s 时为 2.5 MHz；速率为 100 Mbit/s 时为 25 MHz。
- MII_RXD[3:0]：接收数据信号。该信号是 4 个一组的数据信号，由 PHY 同步驱动，在 MII_RX_DV 信号有效时才为有效信号（有效数据）。MII_RXD[0] 为最低有效位，MII_RXD[3] 为最高有效位。当 MII_RX_DV 禁止、MII_RX_ERR 使能时，特定的 MII_RXD[3:0] 值用于代表来自 PHY 的特定信息。
- MII_RX_DV：接收数据有效信号。该信号表示 PHY 当前正针对 MII 接收已恢复并解码的半字节。该信号必须与恢复帧的头半字节进行同步于 MII_RX_CLK，并且一直保持同步到恢复帧的最后半字节。该信号必须在最后半字节随后的第一个时钟周期之前禁止。为了正确地接收帧，MII_RX_DV 信号必须在时间范围上涵盖要接收的帧，其开始时间不得迟于 SFD 字段出现的时间。
- MII_CRS：载波侦听信号。当发送或接收介质处于非空闲状态时，由 PHY 使能该信号。发送和接收介质均

处于空闲状态时，由 PHY 禁止该信号。PHY 必须确保 MII_CS 信号在冲突条件下保持有效状态。该信号无需与 TX 和 RX 时钟保持同步。在全双工模式下，该信号没意义。

- MII_COL：冲突检测信号。检测到介质上存在冲突后，PHY 必须立即使能冲突检测信号，并且只要存在冲突条件，冲突检测信号必须保持有效状态。该信号无需与 TX 和 RX 时钟保持同步。在全双工模式下，该信号没意义。
- MII_RX_ERR：接收错误信号。该信号必须保持一个或多个周期 (MII_RX_CLK)，从而向 MAC 子层指示在帧的某处检测到错误。
- MDIO 和 MDC：管理数据输入输出模块和管理数据时钟。这两个信号构成了符合 IEEE 802.3 标准的以太网串行总线，用于将控制和数据信息传输到 PHY。详见 Station Management Agent (SMA) Interface。

MII 时钟

在 MII 时钟模式下，MII 与 PHY 的接口有 TX 和 RX 两个方向的时钟，clk_tx 用于同步 TX 的数据，clk_rx 用于同步 RX 的数据。clk_tx 时钟和 clk_rx 时钟都由 PHY 提供。

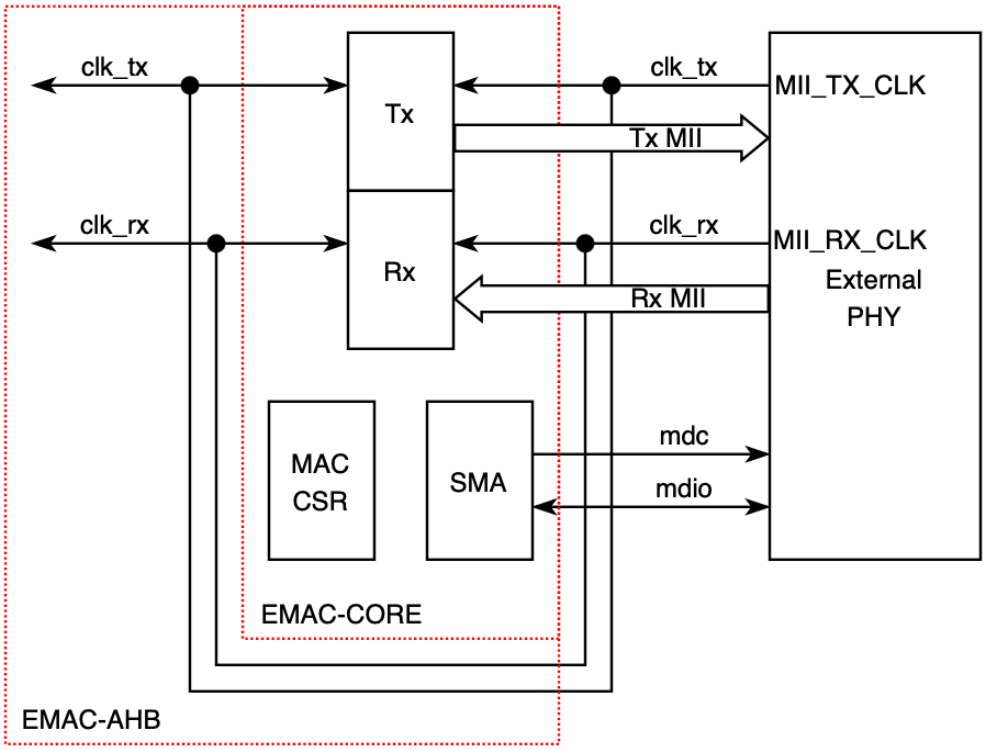


图 55.4-4. MII 时钟

55.4.4.2 RMII（精简介质独立接口）

RMII 接口信号如图 55.4-5 所示。

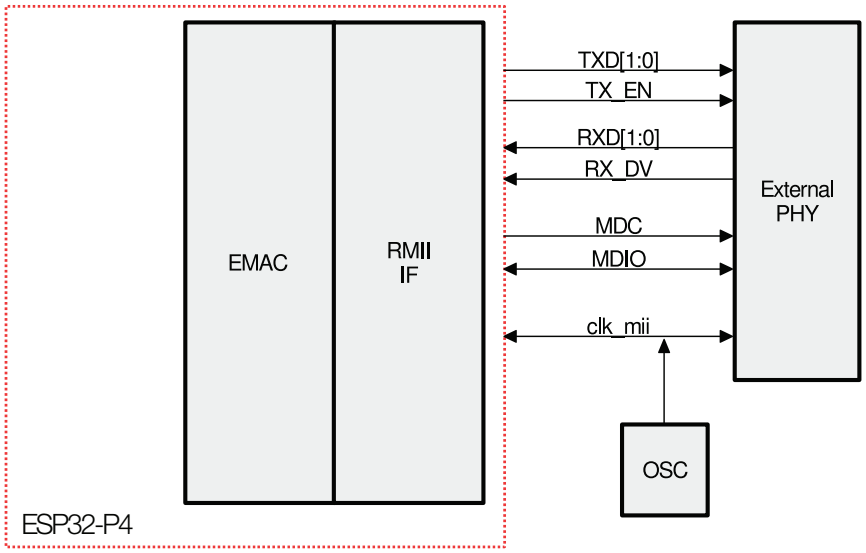


图 55.4-5. RMII 接口

说明：

图 55.4-3 和 55.4-5 统一使用 RX_DV 信号名表示接收有效信号。在 MII 接口中对应 RX_DV，在 RMII 接口中对应 CRS_DV，实际信号语义取决于所选的以太网接口模式。

RMII 接口信号描述

精简介质独立接口 (RMII) 规范降低了 10 Mbit/s 或 100 Mbit/s 下微控制器以太网外设与外部 PHY 间的引脚数。根据 IEEE 802.3u 标准，MII 包括 16 个包含数据和控制信号的引脚。RMII 规范将引脚数减少为 7 个。

RMII 具有以下特性：

- 支持 10 Mbit/s 和 100 Mbit/s 的运行速率
- 参考时钟频率必须是 50 MHz
- 相同的参考时钟必须从外部提供给 MAC 和外部以太网 PHY。
- 提供了独立的 2 位宽的发送和接收数据的路径。

RMII 时钟

RMII 50 MHz 的参考时钟源可由以下方式得到：

- ESP32-P4 内部产生，通过 GPIO 接口环回得到。GPIO 接口配置具体请见章节 *10 GPIO 交换矩阵和 IO MUX*。
- 外部晶振产生，通过 GPIO 接口输入得到。

RMII 时钟如图 55.4-6 所示。

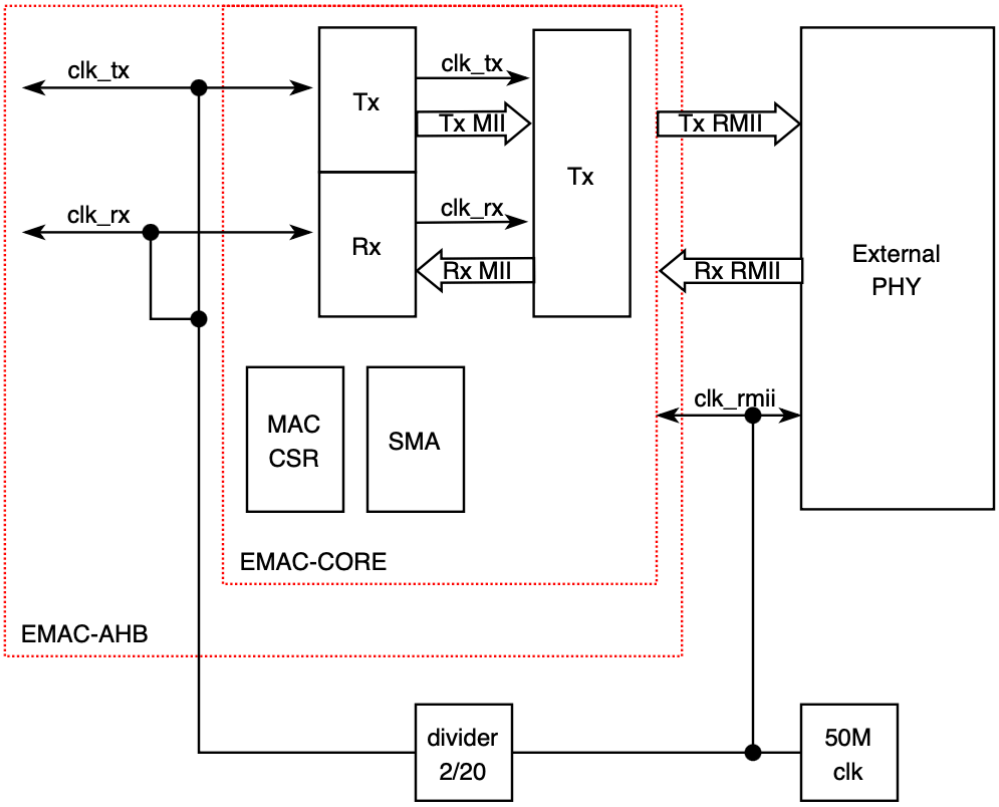


图 55.4-6. RMII 时钟

55.4.4.3 Station Management Agent (SMA) 接口

如图 55.4-3 和图 55.4-5 所示，MAC 通过 MDC 和 MDIO 信号将控制和数据信息传输到 PHY。最大时钟频率为 2.5 MHz。时钟由应用时钟通过时钟分频器分频产生。在通过 MDIO 的写/读操作期间，PHY 发送寄存器数据。该信号与 MDC 时钟同步驱动。

- 读 PHY 寄存器操作：
  - 读取 MIIBUSY，等待其为空闲状态。
  - 配置 MIIDEV，配置访问的 PHY 设备。
  - 配置 MIIREG，配置访问的 PHY 寄存器地址。
  - 配置 MIICSRCLK，配置 APB 时钟频率。
  - 配置 MIIWRITE 为 0，配置为读访问。
  - 读取 MIIBUSY，等待其为空闲状态。
  - 读取 EMACMIIDATA_REG，读取 PHY 寄存器的值。
- 写 PHY 寄存器操作：
  - 读取 MIIBUSY，等待其为空闲状态。
  - 配置 EMACMIIDATA_REG，配置要写入的 PHY 寄存器值。
  - 配置 MIIDEV，配置访问的 PHY 设备。
  - 配置 MIIREG，配置访问的 PHY 寄存器地址。

- 配置 MIICSRCLK，配置 APB 时钟频率。
- 配置 MIIWRITE 为 1，配置为写访问。
- 读取 MIIBUSY，等待其为空闲状态，表示写入完成。

55.4.5 MAC 地址的过滤

地址过滤将检查所有接收的帧的目标地址和源地址并相应报告地址过滤状态。地址检查基于应用程序选择的不同参数（帧过滤寄存器）。还可以识别过滤的帧：多播帧或广播帧。地址过滤使用的物理 (MAC) 地址进行地址检查。

55.4.5.1 单播目标地址过滤

MAC 支持多达 9 个用于单播完美过滤的 MAC 地址。MAC 会将接收的单播地址的所有 48 位与编程的 MAC 地址进行比较来确定是否匹配。默认情况下，始终使能 EMACADDR0，其它地址 EMACADDR0 ~ EMACADDR8 则通过单独的使能位进行选择。将其它地址 (EMACADDR0 ~ EMACADDR8) 的各个字节与接收的相应 DA 字节进行比较时，可以将寄存器中相应的屏蔽字节控制位置 1 来屏蔽该字节。这有助于 DA 的组地址过滤。

55.4.5.2 多播目标地址过滤

可通过将帧过滤寄存器中的 PAM (Pass All Multicast) 位置 1，将 MAC 编程为通过所有多播帧。如果 PAM (Pass All Multicast) 位复位，MAC 将执行对多播地址的过滤。

在完美过滤模式下，将多播地址与编程的 MAC 目标地址寄存器 EMACADDR0 ~ EMACADDR8 进行比较。组地址过滤也受到支持。

55.4.5.3 广播地址过滤

在默认模式下，MAC 不过滤任何广播帧。但是，如果将帧过滤寄存器中的 DBF (Disable Broadcast Frames) 位置 1 来将 MAC 编程为拒绝所有广播帧，则会丢弃任何广播帧。

55.4.5.4 单播源地址过滤

MAC 还可以根据接收的帧的源地址字段来执行完美过滤。默认情况下，AFM 将 SA 字段与 SA 寄存器中编程的值进行过滤。可通过将相应寄存器中的 bit[30] 置 1，来将 MAC 地址寄存器配置为包含 SA 而不是 DA 进行比较。带 SA 的组地址过滤也受到支持。如果帧过滤寄存器中的 SAFE (Source Address Filter Enable) 位置 1，则 MAC 将丢弃未通过 SA 过滤的帧。否则，SA 过滤的结果将通过接收状态字中的状态位给出（详见表 55.4-7）。

SAFE (Source Address Filter Enable) 位置 1 时，对 SA 过滤和 DA 过滤的结果进行与运算，以决定是否需要转发帧。这意味着任何一个过滤未通过都将丢弃帧。两个过滤必须都通过，才能将帧转发到应用。

55.4.5.5 反向过滤操作

对于目标地址和源地址过滤，可在最终输出时选择相反的过滤匹配结果。这分别由帧过滤寄存器中的 DAIF 和 SAIF 位控制。DAIF 位同时适用于单播和多播 DA 帧。在反向过滤操作中，当 DAIF 位置 1 时，将反转单播/多播目标地址过滤的结果。类似地，当 SAIF 位置 1 时，将反转单播 SA 过滤的结果。

下面两个表按照接收帧的类型汇总了目标地址和源地址过滤。

表 55.4-14. 目标地址过滤

帧类型	PM	PF	DAIF	PAM	DB	DA 过滤操作
广播	1	X	X	X	X	通过
	0	X	X	X	0	通过
	0	X	X	X	1	不通过
单播	1	X	X	X	X	通过所有帧
	0	X	0	X	X	完美/组过滤匹配时通过
	0	X	1	X	X	完美/组过滤匹配时不通过
	0	1	0	X	X	完美/组过滤匹配时通过
	0	1	1	X	X	完美/组过滤匹配时不通过
多播	1	X	X	X	X	通过所有帧
	X	X	X	1	X	通过所有帧
	0	X	0	0	X	完美/组过滤匹配时通过，并在 PCF = 0x 时丢弃暂停控制帧
	0	1	0	0	X	完美/组过滤匹配时通过，并在 PCF = 0x 时丢弃暂停控制帧
	0	X	1	0	X	完美/组过滤匹配时不通过，并在 PCF = 0x 时丢弃暂停控制帧
	0	1	1	0	X	完美/组过滤匹配时不通过，并在 PCF = 0x 时丢弃暂停控制帧

表 55.4-14 中，MAC 帧过滤寄存器中的过滤参数如下：

帧类型说明		**参数设置**	
PM:	Pass All Multicast/通过所有组播	1:	置 1
PF:	Perfect Filter/完美过滤	0:	清零
DAIF:	Destination Address Inverse Filtering/目标地址反过滤	X:	无关
PAM:	Pass All Multicast/通过所有组播		
DB:	Disable Broadcast Frames/关闭广播帧		

表 55.4-15. 源地址过滤

帧类型	PM	SAIF	SAF	SA 过滤操作
单播	1	X	X	通过所有帧
	0	0	0	完美/组过滤匹配时通过，但不丢弃未通过的帧
	0	1	0	完美/组过滤匹配时不通过，但不丢弃未通过的帧
	0	0	1	完美/组过滤器匹配时通过并将不通过的帧丢弃
	0	1	1	完美/组过滤器匹配时不通过并将不通过的帧丢弃

表 55.4-15 中，MAC 帧过滤寄存器的过滤参数如下：

帧类型说明		**参数设置**	
PM:	Pass All Multicast/通过所有组播	1:	置 1
SAF:	Source Address Filtering/源地址过滤	0:	清零
SAIF:	Source Address Inverse Filtering/源地址反向过滤	X:	无关

55.4.6 Energy Efficient Ethernet（EEE）

Energy Efficient Ethernet (EEE) 是一种可选的操作模式，它使 MAC 子层以及一系列物理层能够在低功耗空闲 (LPI) 模式下操作。EEE 操作模式支持 100 Mbps 的 MAC 操作。MAC 支持 IEEE Std 802.3az-2010 标准协议。

EEE 功能仅支持工作在全双工模式下的 MII 接口。

当没有数据要发送和接收时，LPI 模式可以通过关闭部分通信设备功能来实现省电。链路两端的系统可以在链路利用率较低期间禁用某些功能并节省电量。MAC 控制系统是否进入或退出 LPI 模式并将其传达给 PHY。

55.4.6.1 传输方向

进入 LPI 模式：软件将 LPIEN 置 1 以指示 MAC 停止传输并启动 LPI 协议。MAC 完成正在进行的传输，生成其传输状态，然后在 IFG 期间开始传输 LPI 序列而不是 IDLE 序列。更新 TLPIEN 并产生中断。

退出 LPI 模式：软件清零 LPIEN 以指示 MAC 退出 LPI 模式。MAC 停止传输 LPI 序列并开始传输 IDLE 序列，等待 LPI_TW_TIMER 到期后，更新 TLPIEX 并产生中断。

55.4.6.2 接收方向

进入 LPI 模式：当 PHY 接收到要进入 LPI 状态的信号时，PHY 开始向 MAC 传输 LPI 序列，MAC 更新 RLPIEN 并产生中断。退出 LPI 模式：当 PHY 接收到要退出 LPI 状态的信号时，PHY 停止向 MAC 传输 LPI 序列，MAC 更新 RLPIEX 并产生中断。

55.4.7 源地址、VLAN 和 CRC 的控制

55.4.7.1 源地址控制

软件可以为所有帧或部分帧启用源地址的插入或替换：

- 通过配置 SAIRC 来开启所有帧源地址的插入或替换。
- 通过配置帧的第一个发送描述符中 TDES1[31:29]，来启用当前帧源地址的插入或替换。当 TDES1[31] 置 1 时，使用 MAC_ADDRESS1_HI 和 EMACADDR1LOW_REG 进行插入或替换。当TDES1[31] 清零时，使用 MAC_ADDRESSO_HI 和 EMACADDROLOW_REG 进行插入或替换。如果 ADDRESS_ENABLE1 未置 1，则无论 TDES1[31] 是否置 1，都使用 MAC_ADDRESSO_HI 和 EMACADDROLOW_REG 进行插入或替换。

55.4.7.2 VLAN 控制

软件可以为所有帧或部分帧启用 VLAN 的插入、替换或删除功能：

- 通过配置 VLC 来开启所有帧 VLAN 的插入、替换或删除。
- 通过配置帧的第一个发送描述符中 TDESO[19:18]，来启用当前帧 VLAN 的插入、替换或删除。

55.4.7.3 CRC 控制

软件可以使用 CRC 替换功能来指示 MAC 用 MAC 计算得到的 CRC 替换传输帧中的 FCS 字段。软件可配置帧的第一个描述符中 TDESO[24]，来启用当前帧 CRC 的替换。

该功能仅在 MAC 没有附加 CRC（TDESO[27] 置 1）时有效，即 FCS 字段由软件在发送帧中提供。如果启用了源地址控制或 VLAN 控制功能，则当 TDESO[27] 置 1 或清零时，MAC 会分别用计算出的 CRC 替换或附加 FCS 字段。

55.4.8 时间戳

IEEE 1588-2008 标准定义了一种协议-精确时间协议 (PTP)，它可以实现通过网络通信、本地计算和分布式对象等技术实现的测量和控制系统中的时钟的精确同步。该协议以最少的网络和本地时钟计算资源支持次微秒范围

内的系统范围同步精度。

55.4.8.1 传输通路

当在 MII 上发送帧的 SFD 时，MAC 捕获时间戳。需要捕获时间戳的帧是可以逐帧控制的。换句话说，每个传输帧可以被标记以指示是否应该为该帧捕获时间戳。MAC 不处理传输帧来识别 PTP 帧。软件需要通过配置 TDES0[25] 以确定要捕获时间戳的帧。MAC 将时间戳写入到对应发送描述符中返回给软件，从而将时间戳自动连接到特定的 PTP 帧。64 位时间戳信息被写入到 TDES6 和 TDES7 字段中。

55.4.8.2 接收通路

MAC 会处理接收到的所有帧以识别有效的 PTP 帧。软件可通过配置 EMACTSTPCTRL_REG 来控制发送到应用程序的快照时间。

MAC 将时间戳写入到对应接收描述符中返回给软件。扩展状态（包含时间戳消息状态和 IPC 状态）被写入到描述符 RDES4 中，时间戳快照被写入到描述符 RDES6 和 RDES7 中。

55.4.9 远端唤醒

MAC 支持远端唤醒功能。当接收端接收到远程唤醒帧或魔术包时，会触发中断通知 PMU 模块唤醒 SoC 系统。

软件可通过配置 RWKPKTEN 来控制是否使用远程唤醒帧来触发中断；软件可通过配置 MGKPKTEN 来控制是否使用魔术包来触发中断；软件可通过配置 PWRDWN 来控制 MAC 在掉电模式下是否丢弃接收到的所有包。

55.4.10 好的传输帧与接收帧

如果帧成功发送，则将发送的帧视为“好帧”。换句话说，如果帧发送过程没有因为以下错误而中止，则认为发送的帧是好帧：

- Jabber 超时 (TDES0[14])
- 无载波/载波丢失 (TDES0[11:10])
- 延迟冲突 (TDES0[9])
- 帧下溢 (TDES0[1])
- 过度延迟 (TDES0[2])
- 过度冲突 (TDES0[8])

如果不存在以下错误，则认为接收帧是“好帧”：

- CRC 错误 (RDES0[1])
- 矮帧（短于 64 字节）
- 对齐错误（仅限 10/100 Mbit/s）(RDES0[2])
- 长度错误（仅限非类型帧）(RDES0[12])
- 超出包最大大小范围（仅限非类型帧，超过最大大小）(RDES0[7])
- MII_RXER 输入错误 (RDES0[3])

最大帧大小取决于帧类型，如下：

- 无标记帧的最大大小 = 1518 字节
- VLAN 帧的最大大小 = 1522 字节

55.5 编程步骤

55.5.1 MAC 系统层配置

1. 配置 IO 接口：MII 接口和 RMII 接口配置具体请见章节 *10 GPIO 交换矩阵和 IO MUX*。
2. 配置 EMAC 系统时钟：HP_SYS_CLKRST_EMAC_SYS_CLK_EN 置 1。
3. 配置 PHY 接口时钟：
   - RMII 接口：配置 HP_SYS_PHY_INTF_SEL 为 4，选择 PHY 接口为 RMII。
     - RMII 参考时钟源选择：
       * 参考时钟由外部晶振产生：
         · 配置 HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN：配置禁止 PAD_EMAC_REF_CLK 输出到 GPIO。
         · 配置 HP_SYS_CLKRST_REF_50M_CLK_EN：配置禁用 PLL_F50M_CLK。
       * 参考时钟由 ESP32-P4 产生：
         · 配置 HP_SYS_CLKRST_PAD_EMAC_REF_CLK_EN：配置使能 PAD_EMAC_REF_CLK 输出到 GPIO。
         · 配置 HP_SYS_CLKRST_REF_50M_CLK_DIV_NUM：配置 PLL_F50M_CLK 分频系数。
         · 配置 HP_SYS_CLKRST_REF_50M_CLK_EN：配置使能 PLL_F50M_CLK。
     - 配置 HP_SYS_CLKRST_EMAC_RMII_CLK_SRC_SEL：配置 EMAC_RMII_CLK 时钟源为 PAD_EMAC_TXRX_CLK。
     - 配置 HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL：配置 EMAC_RX_CLK 时钟源为 PAD_EMAC_TXRX_CLK。
     - 配置 HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM：根据 EMAC 工作在 10 Mbit/s 或 100 Mbit/s 来选择配置 EMAC_RX_CLK 时钟对应分频系数。
     - 配置 HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL：配置 EMAC_TX_CLK 时钟源为 PAD_EMAC_TXRX_CLK。
     - 配置 HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM：根据 EMAC 工作在 10 Mbit/s 或 100 Mbit/s 来选择配置 EMAC_TX_CLK 时钟对应分频系数。
     - 配置 LP_AONCLKRST_HP_PAD_EMAC_TXRX_CLK_EN：配置使能来自 IO 管脚的 PAD_EMAC_TXRX_CLK。
     - 配置 HP_SYS_CLKRST_EMAC_RMII_CLK_EN：配置使能 EMAC_RMII_CLK。
     - 配置 HP_SYS_CLKRST_EMAC_RX_CLK_EN：配置使能 EMAC_RX_CLK。
     - 配置 HP_SYS_CLKRST_EMAC_TX_CLK_EN：配置使能 EMAC_TX_CLK。

- MII 接口：配置 HP_SYS_PHY_INTF_SEL 为 0，选择 PHY 接口为 MII。
  - 配置 HP_SYS_CLKRST_EMAC_RX_CLK_SRC_SEL：配置 EMAC_RX_CLK 时钟源为 PAD_EMAC_RX_CLK。
  - 配置 HP_SYS_CLKRST_EMAC_RX_CLK_DIV_NUM：根据 EMAC 工作在 10 Mbit/s 或 100 Mbit/s 来选择配置 EMAC_RX_CLK 时钟对应分频系数。
  - 配置 HP_SYS_CLKRST_EMAC_TX_CLK_SRC_SEL：配置 EMAC_TX_CLK 时钟源为 PAD_EMAC_TX_CLK。
  - 配置 HP_SYS_CLKRST_EMAC_TX_CLK_DIV_NUM：根据 EMAC 工作在 10 Mbit/s 或 100 Mbit/s 来选择配置 EMAC_TX_CLK 时钟对应分频系数。
  - 配置 LP_AONCLKRST_HP_PAD_EMAC_RX_CLK_EN：配置使能来自 IO 管脚的 PAD_EMAC_RX_CLK。
  - 配置 LP_AONCLKRST_HP_PAD_EMAC_TX_CLK_EN：配置使能来自 IO 管脚的 PAD_EMAC_TX_CLK。
  - 配置 HP_SYS_CLKRST_EMAC_RX_CLK_EN：配置使能 EMAC_RX_CLK。
  - 配置 HP_SYS_CLKRST_EMAC_TX_CLK_EN：配置使能 EMAC_TX_CLK。

55.5.2 EMAC 初始配置

EMAC_DMA 初始化：

- 复位 EMAC：SW_RST 置 1。
- 等待复位操作完成：软件读 SW_RST，值为 0 表示复位操作完成。
- 软件读 AHB_ST，值为 0 表示 AHB 总线事务已全部完成。
- 配置 DMABUSMODE_REG 设置 EMAC_DMA AHB 总线的工作模式。
- 准备好链表，并配置链表基地址 DMARXBASEADDR_REG 和 DMATXBASEADDR_REG。
- 配置 DMAOPERATION_MODE_REG：配置 EMAC_DMA 和 EMAC_MTL 的工作模式。
- 配置 DMAIN_EN_REG 选择是否使能 DMA 中断。

EMAC_CORE 初始化：

- 通过 EMACMIIADDR_REG 和 EMACMIIDATA_REG，读取 PHY 的连接状态、工作频率以及工作模式。
- 配置 EMACADDROHIGH_REG 和 EMACADDROLOW_REG，用于配置 MAC 地址。
- 配置 EMACFF_REG，选择报文过滤模式。
- 配置 EMACFC_REG，用于流控操作。
- 配置 EMACINTMASK_REG 选择是否屏蔽 MAC 中断。
- 配置 EMACCONFIG_REG 设置 MAC 传输和接收的工作模式。

55.5.3 启动传输

- START_STOP_TRANSMISSION_COMMAND 置 1 开启 EMAC_DMA 传输通路。

- EMACTX 置 1 开启 EMAC_CORE 传输通路。
- 若 TRANS_INT 中断使能，可检测 TRANS_INT 中断，等待发送帧结束。

55.5.4 启动接收

- START_STOP_RX 置 1 开启 EMAC_DMA 接收通路。
- EMACRX 置 1 开启 EMAC_CORE 接收通路。
- 若 RECV_INT 中断使能，可检测 RECV_INT，等待接收帧结束。

55.5.5 发送端进退 LPI 状态

- 配置 EMACLPITIMERSCONTROL_REG，设置 LPI 的时间计数器。
- 配置 EMACLPI_CSR_REG，使能发送端进入 LPI 状态。
- 检测 LPI_INT 中断，等待 TLPIEN 为高，表示发送端进入 LPI 状态。
- LPIEN 清零，发送端开始退出 LPI 状态。
- 检测 LPI_INT 中断，等待 TLPIEX 为高，表示发送端退出 LPI 状态。

55.5.6 接收端进退 LPI 状态

- 检测 LPI_INT 中断，等待 RLPIEN 为高，表示接收端进入 LPI 状态。
- 检测 LPI_INT 中断，等待 RLPIEX 为高，表示接收端退出 LPI 状态。

55.6 中断

ESP32-P4 的 EMAC 可以生成以下中断信号并发送给中断矩阵。

- ETH_MAC_INTR
- PMT_INTR
- LPI_INTR

这些中断信号由 EMAC 的内部中断源生成。

以下中断源生成 ETH_MAC_INTR 中断信号：

- ABN_INT_SUMM：DMAIN_AISE 置 1 时，当以下中断触发后，触发此中断。
  - TRANS_PROC_STOP：DMAIN_TSE 置 1 时，当传输进程终止时，触发此中断。
  - TRANS_JABBER_TO：DMAIN_TJTE 置 1 时，当帧大小超过 2048 字节（当 Jumbo 帧被使能时为 10240 字节）时，触发此中断。
  - RECV_OVFLOW：DMAIN_OIE 置 1 时，当接收缓存在帧接收期间存在溢出时，触发此中断。
  - TRANS_UNDFLOW：DMAIN_UIE 置 1 时，当发送缓存在帧传输期间存在下溢时，触发此中断。
  - RECV_BUF_UNAVAIL：DMAIN_RBUE 置 1 时，当主机拥有接收列表中的下一个描述符，但 DMA 无法获取该描述符时，触发此中断。

  - RECV_PROC_STOP：DMAIN_RSE 置 1 时，当接收进程终止时，触发此中断。
  - RECV_WDT_TO：DMAIN_RWTE 置 1 时，接收看门狗定时器在接收当前帧时已经到期，触发此中断。
  - EARLY_TRANS_INT：DMAIN_ETIE 置 1 时，当发送的帧被完全传送到 MTL 发送 FIFO 时，触发此中断。
  - FATAL_BUS_ERR_INT：DMAIN_FBEE 置 1 时，当发生 ERROR_BITS 所述的总线错误时，触发此中断。
- NORM_INT_SUMM：DMAIN_NISE 置 1 时，当以下中断触发后，触发此中断。
  - TRANS_INT：DMAIN_TIE 置 1 时，当传输完成时，触发此中断。
  - TRANS_BUF_UNAVAIL：DMAIN_TBUE 置 1 时，当主机拥有发送列表中的下一个描述符，但 DMA 无法获取该描述符时，触发此中断。
  - RECV_INT：DMAIN_RIE 置 1 时，当帧接收已完成时，触发此中断。
  - EARLY_RECV_INT：DMAIN_ERIE 置 1 时，当 DMA 填充了数据包的第一个缓冲区时，触发此中断。
- EMAC_PMT_INT：PMTINTMASK 清零时，当在掉电模式下收到一个魔术包或远程唤醒帧时，触发此中断。
- TS_TRI_INT：TSINTMASK 清零时，以下任何一个条件成立，都触发此中断：
  - 系统时间值等于或超过目标时间指定的值。
  - 系统时间秒寄存器中有溢出。
- EMAC_LPI_INT：LPIINTMASK 清零时，以下任何一个条件成立，都触发此中断：
  - MAC 传输端进入 LPI 状态。
  - MAC 传输端退出 LPI 状态。
  - MAC 接收端收到 LPI 序列并进入 LPI 状态。
  - MAC 接收端停止接收 LPI 序列并退出 LPI 状态。

以下中断源生成 PMT_INTR 中断信号：

- RWKPRCVD：当收到远程唤醒帧时，会触发此中断。
- MGKPRCVD：当收到 magic 帧时，会触发此中断。

以下中断源生成 LPI_INTR 中断信号：

- RLPIEX：当 MAC 接收端停止接收 LPI 序列并退出 LPI 状态时，会触发此中断。

55.7 寄存器列表

本小节的所有地址均为相对于以太网介质访问控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
DMA 配置和控制寄存器			
DMABUSMODE_REG	配置总线模式	0x1000	R/WS/SC
DMATXPOLLDEMAND_REG	数据传输指令	0x1004	RO/WT
DMARXPOLLDEMAND_REG	数据接收指令	0x1008	RO/WT
DMARXBASEADDR_REG	第一个接收描述符的基地址	0x100C	R/W
DMATXBASEADDR_REG	第一个传输描述符的基地址	0x1010	R/W
DMASTATUS_REG	中断、错误和其他事件的基地址	0x1014	R/SS/WC
DMAOPERATION_MODE_REG	收发操作模式和命令寄存器	0x1018	R/SS/WC
DMAIN_EN_REG	中断关闭/使能	0x101C	R/W
DMAMISSEDFR_REG	丢失帧和缓存溢出计数器	0x1020	R/W
DMARINTWDTIMER_REG	接收看门狗计数器	0x1024	R/W
DMAAHBSTATUS_REG	AHB 主机接口状态	0x102C	RO
DMATXCURRDESC_REG	指向当前传输描述符的指针	0x1048	RO
DMARXCURRDESC_REG	指向当前接收描述符的指针	0x104C	RO
DMATXCURRADDR_BUF_REG	指向当前传输缓存的指针	0x1050	RO
DMARXCURRADDR_BUF_REG	指向当前接受缓存的指针	0x1054	RO
MAC 配置和控制寄存器			
EMACCONFIG_REG	MAC 配置	0x0000	R/W
EMACFF_REG	帧过滤设置	0x0004	R/W
EMACMIIADDR_REG	PHY 配置访问权限	0x0010	R/WS/SC
EMACMIIDATA_REG	PHY 数据读写	0x0014	R/W
EMACFC_REG	帧流控制	0x0018	varies
EMACVLANTAG_REG	802.1Q VLAN 类型报文	0x001C	R/W
EMACDEBUG_REG	状态调试位	0x0024	RO
PMT_RWUFFR_REG	远程唤醒滤波器	0x0028	RO
PMT_CSR_REG	功耗管理控制和状态寄存器	0x002C	RO
EMACLPI_CSR_REG	LPI 控制和状态寄存器	0x0030	RO
EMACLPITIMERSCONTROL_REG	LPI 计时器控制器	0x0034	RO
EMACINTS_REG	中断状态	0x0038	RO
EMACINTMASK_REG	中断屏蔽	0x003C	R/W
EMACADDR0HIGH_REG	第一个 6 字节 MAC 地址的高 16 位	0x0040	R/W
EMACADDR0LOW_REG	第一个 6 字节 MAC 地址的低 32 位	0x0044	R/W
EMACADDR1HIGH_REG	第二个 6 字节 MAC 地址的高 16 位	0x0048	R/W
EMACADDR1LOW_REG	第二个 6 字节 MAC 地址的低 32 位	0x004C	R/W
EMACADDR2HIGH_REG	第三个 6 字节 MAC 地址的高 16 位	0x0050	R/W
EMACADDR2LOW_REG	第三个 6 字节 MAC 地址的低 32 位	0x0054	R/W
EMACADDR3HIGH_REG	第四个 6 字节 MAC 地址的高 16 位	0x0058	R/W

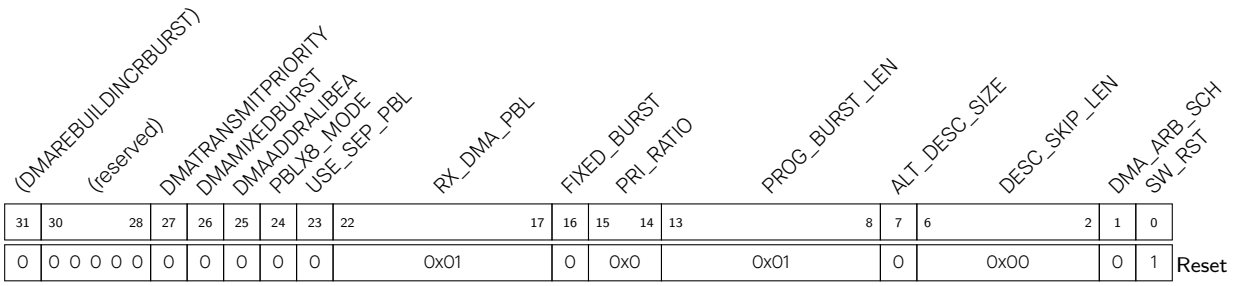
名称	描述	地址	访问
EMACADDR3LOW_REG	第四个 6 字节 MAC 地址的低 32 位	0x005C	R/W
EMACADDR4HIGH_REG	第五个 6 字节 MAC 地址的高 16 位	0x0060	R/W
EMACADDR4LOW_REG	第五个 6 字节 MAC 地址的低 32 位	0x0064	R/W
EMACADDR5HIGH_REG	第六个 6 字节 MAC 地址的高 16 位	0x0068	R/W
EMACADDR5LOW_REG	第六个 6 字节 MAC 地址的低 32 位	0x006C	R/W
EMACADDR6HIGH_REG	第七个 6 字节 MAC 地址的高 16 位	0x0070	R/W
EMACADDR6LOW_REG	第七个 6 字节 MAC 地址的低 32 位	0x0074	R/W
EMACADDR7HIGH_REG	第八个 6 字节 MAC 地址的高 16 位	0x0078	R/W
EMACADDR7LOW_REG	第八个 6 字节 MAC 地址的低 32 位	0x007C	R/W
EMACADDR8HIGH_REG	第九个 6 字节 MAC 地址的高 16 位	0x0080	R/W
EMACADDR8LOW_REG	第九个 6 字节 MAC 地址的低 32 位	0x0084	R/W
EMACCSTATUS_REG	连接通讯状态	0x00D8	RO
EMACWDOGTO_REG	看门狗超时控制	0x00DC	R/W
EMACTXVLTCTRL_REG	传输端 VLAN 标志控制	0x0584	R/W
EMACTSTPCTRL_REG	时间戳控制寄存器	0x0700	R/W
EMACSUB2NDINCR_REG	次秒增量寄存器	0x0704	R/W
EMACSYSTIM2ND_REG	系统时间秒寄存器	0x0708	RO
EMACSYSTIMNS_REG	系统时间纳秒寄存器	0x070C	RO
EMAC2NDUPT_REG	系统时间秒更新寄存器	0x0710	R/W
EMACNSUPT_REG	系统时间纳秒更新寄存器	0x0714	R/W
EMACTSTPADDEND_REG	精细更新模型累加值	0x0718	R/W
EMACTGTIME2ND_REG	目标时间秒寄存器	0x071C	R/W
EMACTGTIMENS_REG	目标时间纳秒寄存器	0x0720	R/W
EMACSYSTIMHIGHWORD2ND_REG	系统时间秒寄存器第 32 到 47 位	0x0724	R/W/SU
EMACTSTPSTATUS_REG	时间戳状态寄存器	0x0728	R/SS/RC

55.8 寄存器

本小节的所有地址均为相对于以太网介质访问控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 55.1. DMABUSMODE_REG (0x1000)



DMAREBUILDINCRBURST 配置是否重建 INCRx 突发传输。

0：不重建

1：如果 AHB 主机收到 EBT（重试、分割或失去总线授权），AHB 主机接口将使用 INCRx 和 SINGLE 重建使用 INCRx 发起的任何突发传输的待处理节拍。

(R/W)

DMATRANSMITPRIORITY 配置发送 DMA 在仲裁期间是否优先级高于接收 DMA。

0：低优先级

1：高优先级

(R/W)

DMAMIXEDBURST 配置是否使用混合突发传输。

0：不使用

1：如果 FIXED_BURST 位为 1，则 AHB 主机接口对突发长度大于 16 的使用 INCR（未定义突发传输长度），对长度为 16 及以下的使用固定传输（INCRx 和 SINGLE）。

(R/W)

DMAADDRALIBEA 配置突发传输是否与地址对齐。

0：不与地址对齐

1：如果 FIXED_BURST 位为 1，则 AHB 接口的所有突发传输与对起始地址 LS 位对齐。如果 FIXED_BURST 位为 0，则第一个突发传输（访问数据缓冲区的起始地址）不对齐，但后续突发传输与地址对齐。

(R/W)

PBLX8_MODE 配置是否将 PBL（可配置的突发长度）值乘以 8。

0：禁用

1：配置的 PBL 值（PROG_BURST_LEN 和 RX_DMA_PBL）乘以 8。因此，DMA 根据 PBL 值以 8、16、32、64、128 和 256 节拍传输数据。

(R/W)

见下页...

Register 55.1. DMABUSMODE_REG (0x1000)

接上页...

USE_SEP_PBL 配置是否使用单独的 PBL。

0：PROG_BURST_LEN 仅适用于 TX DMA，RX DMA 使用 RX_DMA_PBL。

1：PROG_BURST_LEN 适用于 TX 和 RX DMA。

(R/W)

RX_DMA_PBL 配置在一次 RX DMA 传输中要传输的最大节拍数，即在单次读取或写入的最大值。

取值范围：1、2、4、8、16 和 32。其他值会导致未定义行为。

仅在 USE_SEP_PBL 为 1 时有效。

(R/W)

FIXED_BURST 配置 AHB 主机接口是否执行固定长度突发传输。

0：AHB 接口使用 SINGLE 和 INCR 突发传输操作。

1：AHB 接口在正常突发传输开始时仅使用 SINGLE、INCR4、INCR8 或 INCR16。

(R/W)

PRI_RATIO 配置 RX DMA 和 TX DMA 权重轮询仲裁时的优先级比例。

0：1:1

1：2:0

2：3:1

3：4:1

仅当 DMA_ARB_SCH 为 0 时有效。(R/W)

PROG_BURST_LEN 配置一次 DMA 传输中要传输的最大节拍数。

如果要传输的节拍数超过 32，则执行以下步骤：

1. 使用 PBLX8_MODE 设置 PBLx8 模式
2. 使用 PROG_BURST_LEN 设置 PBL

(R/W)

ALT_DESC_SIZE 配置是否将描述符大小增加到 32 字节。

0：保持 16 字节

1：增加到 32 字节

(R/W)

DESC_SKIP_LEN 配置在两个未链接描述符之间跳过的字、双字或长字（取决于 32 位、64 位或 128 位总线）的数量。

地址跳过从当前描述符的末尾到下一个描述符的开始。当此字段为 0 时，DMA 在环模式下将描述符表视为连续的。(R/W)

见下页...

Register 55.1. DMABUSMODE_REG (0x1000)

接上页...

DMA_ARB_SCH 配置传输和接收路径之间的仲裁方案。

0：RX:TX 或 TX:RX 的权重轮询。在这种情况下，路径之间的优先级根据 PRI_RATIO 中指定的优先级确定。

1：固定优先级（传输路径优先于接收路径）。

(R/W)

SW_RST 配置 MAC DMA 控制器是否复位 MAC 的逻辑和所有内部寄存器。

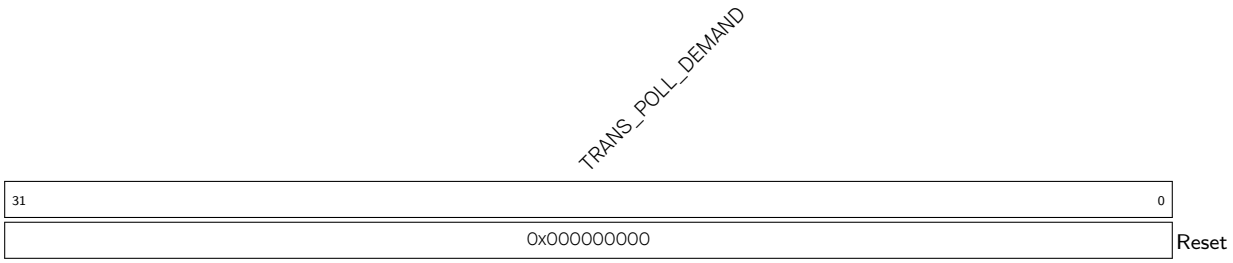
0：释放复位

1：复位

此位在所有 ETH_MAC 时钟域中的复位操作完成后会自动清除。

(R/WS/SC)

Register 55.2. DMATXPOLLDEMAND_REG (0x1004)

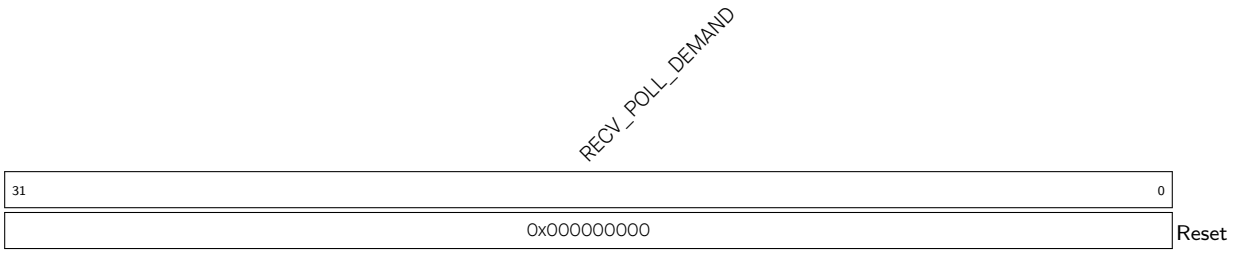


TRANS_POLL_DEMAND 配置是否启用 TX DMA 来检查 DMA 是否拥有当前描述符。

任意值：启用

当写入该字段时，DMA 会读取由DMATXCURRDESC_REG 指向的当前描述符。如果该描述符不可用（主机拥有），传输将返回到挂起状态，并且寄存器 5（状态寄存器）的第 2 位 (TU) 置 1。如果描述符可用，则传输继续。(RO/WT)

Register 55.3. DMARXPOLLDEMAND_REG (0x1008)

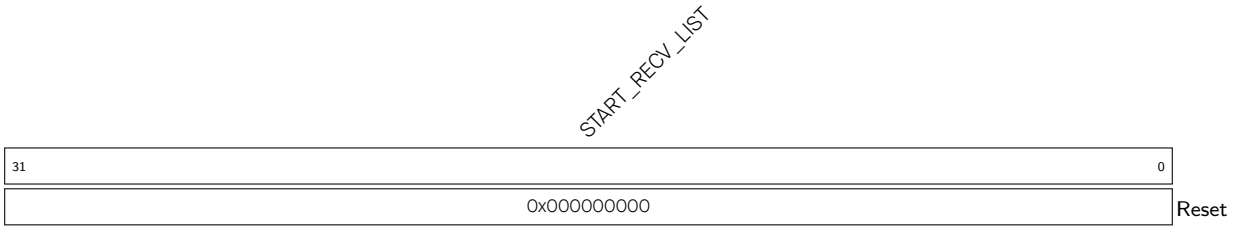


RECV_POLL_DEMAND 配置是否启用 RX DMA 来检查 DMA 是否拥有当前描述符。

任意值：启用

当写入该字段时，DMA 会读取由DMARXCURRDESC_REG 指向的当前描述符。如果该描述符不可用（主机拥有），接收将返回到挂起状态，并且寄存器 5（状态寄存器）的第 7 位 (RU) 不置 1。如果描述符可用，则 RX DMA 返回到活跃状态。(RO/WT)

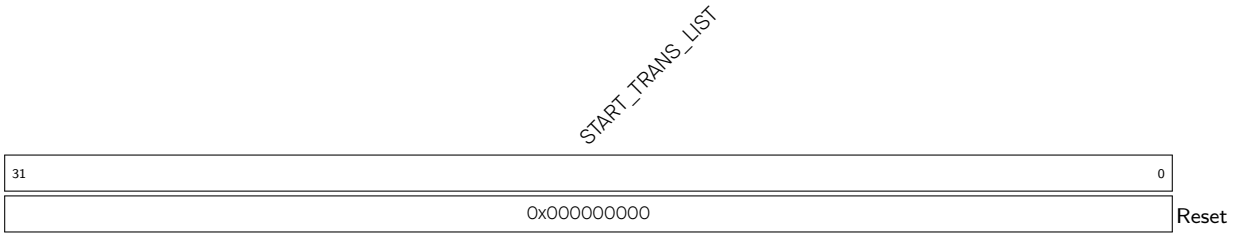
Register 55.4. DMARXBASEADDR_REG (0x100C)



START_RECV_LIST 配置接收描述符链表中第一个描述符的基地址。

LSB 位（1:0）被忽略（32 位宽总线），并且由 DMA 内部视为全零。因此，这些 LSB 位为只读。(R/W)

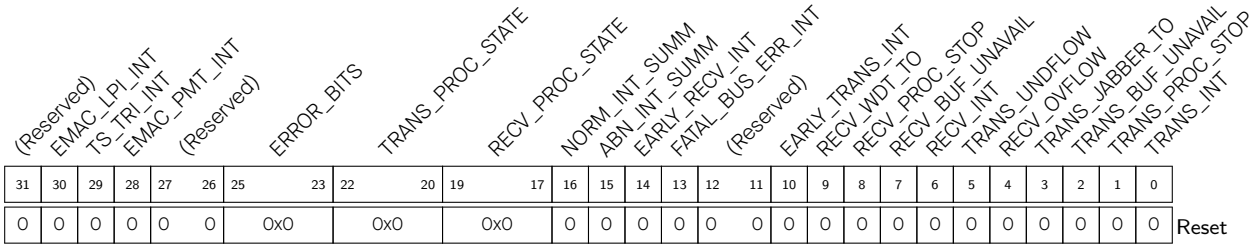
Register 55.5. DMATXBASEADDR_REG (0x1010)



START_TRANS_LIST 配置发送链表中第一个描述符的基地址。

LSB 位（1:0）被忽略（32 位宽总线），并且由 DMA 内部视为全零。因此，这些 LSB 位为只读。(R/W)

Register 55.6. DMASTATUS_REG (0x1014)



EMAC_LPI_INT EMAC_LPI_INT 的原始中断状态。(RO)

TS_TRI_INT TS_TRI_INT 的原始中断状态。(RO)

EMAC_PMT_INT EMAC_PMT_INT 的原始中断状态。(RO)

ERROR_BITS 表示导致总线错误的错误类型，例如 AHB 接口上的错误响应。

0：RX DMA 写数据传输期间出错
1：TX DMA 读数据传输期间出错
2：RX DMA 描述符写访问期间出错
3：TX DMA 描述符写访问期间出错
4：RX DMA 描述符读访问期间出错
5：TX DMA 描述符读访问期间出错
仅当第 13 位 (FBI) 为 1 时有效。此字段不会生成中断。(RO)

TRANS_PROC_STATE 表示传输 DMA 的 FSM 状态。

0：停止；已重置或已发出停止传输命令
1：运行中；正在获取传输传输描述符
2：运行中；等待状态
3：运行中；从主机内存缓冲区读取数据并将其排队到传输缓冲区 (TX FIFO)
4：TIME_STAMP 写状态
5：保留
6：挂起；传输描述符不可用或传输缓冲区下溢
7：运行中；关闭传输描述符
此字段不会生成中断。(RO)

见下页...

Register 55.6. DMASTATUS_REG (0x1014)

接上页...

NORM_INT_SUMM 中断使能寄存器中的相应位为 1 时，该位的值是以下各位的逻辑或。

位 [0]：传输中断

位 [2]：传输缓冲区不可用

位 [6]：接收中断

位 [14]：提前接收中断

只有未屏蔽的中断会影响此位。

这是一个粘滞位，必须在每次引起 NIS 置位的相应位清零（通过向该位写入 1）被清除。(R/SS/WC)

ABN_INT_SUMM 中断使能寄存器中的对应中断使能时，该位的值是以下逻辑或。

位 [1]：传输进程停止

位 [3]：传输 Jabber 超时

位 [4]：接收 FIFO 溢出

位 [5]：传输下溢

位 [7]：接收缓冲区不可用

位 [8]：接收进程停止

位 [9]：接收看门狗超时

位 [10]：提前传输中断

位 [13]：致命总线错误

只有未屏蔽的中断会影响此位。

这是一个粘滞位，必须在每次引起 AIS 置位的相应位清零时（向该位写入 1）清除。(R/SS/WC)

EARLY_RECV_INT EARLY_RECV_INT 的原始中断状态。(R/SS/WC)

FATAL_BUS_ERR_INT FATAL_BUS_ERR_INT 的原始中断状态。(R/SS/WC)

EARLY_TRANS_INT EARLY_TRANS_INT 的原始中断状态。(R/SS/WC)

RECV_WDT_TO 表示在接收当前帧时接收看门狗定时器是否超时，并且当前帧在看门狗超时后被截断。

0：未超时

1：已超时

(R/SS/WC)

RECV_PROC_STOP 表示接收进程是否进入停止状态。

0：未停止

1：已停止

(R/SS/WC)

见下页...

Register 55.6. DMASTATUS_REG (0x1014)

接上页...

RECV_BUF_UNAVAIL 表示接收缓冲区是否可用。

0：可用

1：不可用

只有当前一个接收描述符由 DMA 拥有时，此位才会被置 1。

当此位被置 1 时，主机拥有接收列表中的下一个描述符，并且 DMA 无法获取该描述符。接收进程被暂停。

要恢复处理接收描述符，主机应该更改描述符的所有权并发出接收轮询需求命令。如果没有发出接收轮询需求，接收进程将在接收到下一个被识别的传入帧时恢复。(R/SS/WC)

RECV_INT RECV_INT 的原始中断状态。(R/SS/WC)

TRANS_UNDFLOW 表示在帧传输过程中传输缓冲区是否下溢。

0：无下溢

1：下溢

当此位为 1 时，传输被暂停，并设置下溢错误 TDESO[1]。(R/SS/WC)

RECV_OVFLOW 表示在帧接收过程中接收缓冲区是否发生溢出。

0：无溢出

1：溢出

如果部分帧被传输到应用程序，则在 RDESO[11] 中设置溢出状态。(R/SS/WC)

TRANS_JABBER_TO 表示传输 Jabber 定时器是否超时，帧大小超过 2,048（启用巨帧时为 10,240 字节）时会超时。

0：未超时

1：已超时

当 Jabber 超时发生时，传输过程被中止并进入停止状态。这会导致传输 Jabber 超时 TDESO[14] 标志置 1。(R/SS/WC)

TRANS_BUF_UNAVAIL 表示传输缓冲区是否可用。

0：可用

1：不可用

当此位为 1 时，主机拥有传输列表中的下一个描述符，并且 DMA 无法获取该描述符。传输被暂停。位 [22:20] 表示传输进程的状态转换。

要恢复处理传输描述符，主机应该通过设置 TDESO[31] 更改描述符的所有权，然后发出传输轮询需求命令。(R/SS/WC)

TRANS_PROC_STOP 表示传输是否停止。

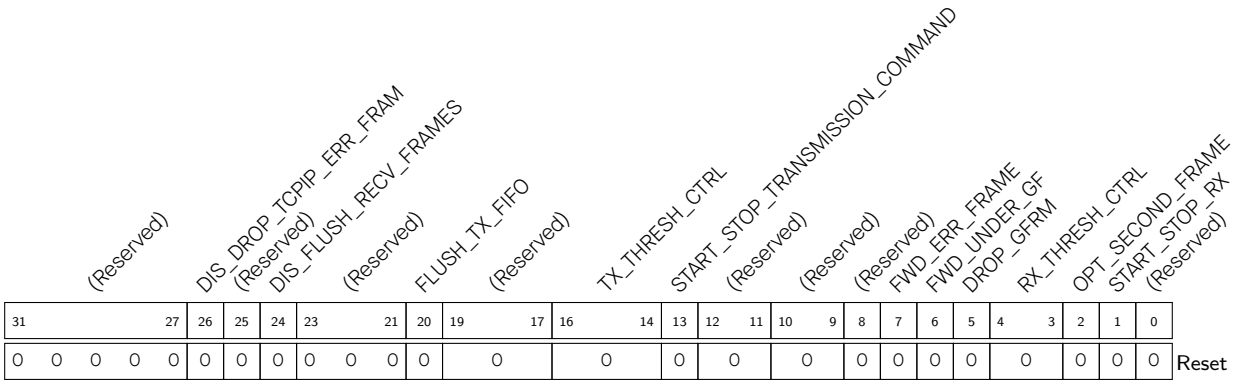
0：未停止

1：已停止

(R/SS/WC)

TRANS_INT TRANS_INT 的原始中断状态。(R/SS/WC)

Register 55.7. DMAOPERATION_MODE_REG (0x1018)



DIS_DROP_TCPIP_ERR_FRAM 配置是否禁用丢弃 TCP/IP 校验和错误帧。

0：启用

1：禁用

当 FWD_ERR_FRAME 为 0 时，此位被忽略，所有错误帧都会被丢弃。(R/W)

DIS_FLUSH_RECV_FRAMES 配置是否因接收描述符或缓冲区不可用而刷新接收帧。

0：启用

1：禁用

(R/W)

FLUSH_TX_FIFO 配置是否将 TX FIFO 刷新为默认值。

0：不刷新

1：刷新

(R/WS/SC)

见下页...

Register 55.7. DMAOPERATION_MODE_REG (0x1018)

接上页...

TX_THRESH_CTRL 配置 TX FIFO 的阈值。当 TX FIFO 中的帧大小大于阈值时开始传输。此外，长度小于阈值的完整帧也会被传输。

0：64
1：128
2：192
3：256
4：40
5：32
6：24
7：16
(R/W)

START_STOP_TRANSMISSION_COMMAND 配置是否启动或停止传输。

0：停止
1：启动
(R/W)

FWD_ERR_FRAME 配置是否将 RX FIFO 中的带有错误状态（CRC 错误、冲突错误、巨帧、看门狗超时或溢出）的帧转发。

0：丢弃
1：转发
(R/W)

FWD_UNDER_GF 配置是否将 RX FIFO 中长度小的好帧（无错误且长度小于 64 字节）转发（包括填充和 CRC）。

0：丢弃
1：转发
(R/W)

DROP_GFRM 配置 ETH_MAC 是否丢弃 RX FIFO 中接收到的巨帧。

0：不丢弃
1：丢弃
(R/W)

见下页...

Register 55.7. DMAOPERATION_MODE_REG (0x1018)

接上页...

RX_THRESH_CTRL 配置 RX FIFO 的阈值。

0：64

1：32

2：96

3：128

当 RX FIFO 中的帧大小大于阈值时，开始传输（请求）给 DMA。(R/W)

OPT_SECOND_FRAME 配置 DMA 是否在获取第一帧的状态之前处理传输数据的第二帧。

0：不处理

1：处理

(R/W)

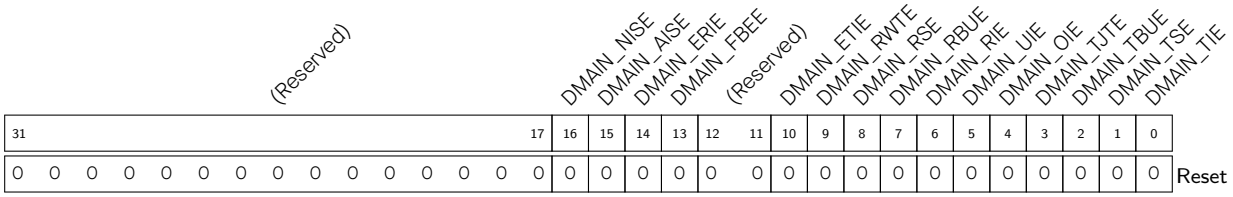
START_STOP_RX 配置是否启动或停止 RX DMA 接收。

0：在当前帧传输完成后停止接收

1：启动接收

(R/W)

Register 55.8. DMAIN_EN_REG (0x101C)



DMAIN_NISE 写 1 使能、写 0 清除以下正常中断使能位的集合位：

位 [0]：传输中断

位 [2]：传输缓冲区不可用

位 [6]：接收中断

位 [14]：提前接收中断

(R/W)

DMAIN_AISE 写 1 使能、写 0 清除以下异常中断使能位的集合位：

位 [1]：传输进程停止

位 [3]：传输 Jabber 超时

位 [4]：接收 FIFO 溢出

位 [5]：传输下溢

位 [7]：接收缓冲区不可用

位 [8]：接收进程停止

位 [9]：接收看门狗超时

位 [10]：提前传输中断

位 [13]：致命总线错误

(R/W)

DMAIN_ERIE 写 1 使能、写 0 清除 EARLY_RECV_INT。(R/W)

DMAIN_FBEE 写 1 使能、写 0 清除 FATAL_BUS_ERR_INT。(R/W)

DMAIN_ETIE 写 1 使能、写 0 清除 EARLY_TRANS_INT。(R/W)

DMAIN_RWTE 写 1 使能、写 0 清除 RECV_WDT_TO_INT。(R/W)

DMAIN_RSE 写 1 使能、写 0 清除 RECV_PROC_STOP_INT。(R/W)

DMAIN_RBUE 写 1 使能、写 0 清除 RECV_BUF_UNAVAIL_INT。(R/W)

DMAIN_RIE 写 1 使能、写 0 清除 RECV_INT。(R/W)

见下页...

Register 55.8. DMAIN_EN_REG (0x101C)

接上页...

DMAIN_UIE 写 1 使能、写 0 清除 TRANS_UNDFLOW_INT。(R/W)

DMAIN_OIE 写 1 使能、写 0 清除 RECV_OVFLOW_INT。(R/W)

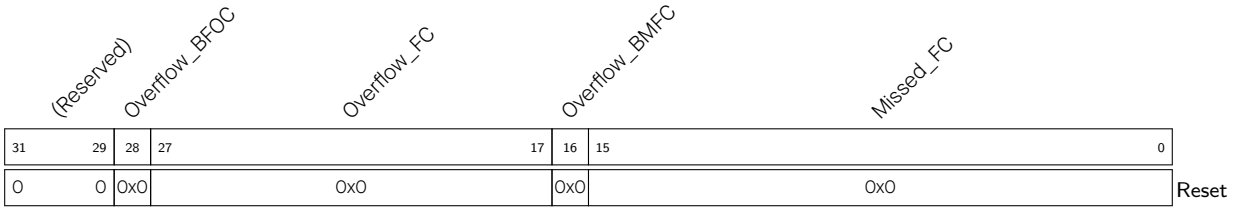
DMAIN_TJTE 写 1 使能、写 0 清除 TRANS_JABBER_TO_INT。(R/W)

DMAIN_TBUE 写 1 使能、写 0 清除 TRANS_BUF_UNAVAIL_INT。(R/W)

DMAIN_TSE 写 1 使能、写 0 清除 TRANS_PROC_STOP_INT。(R/W)

DMAIN_TIE 写 1 使能、写 0 清除 TRANS_INT。(R/W)

Register 55.9. DMAMISSEDFR_REG (0x1020)



Overflow_BFOC 表示溢出帧计数器的状态。

0：无溢出。

1：溢出帧计数器（Overflow_FC）溢出，即 RX FIFO 以最大值的溢出帧计数器溢出。在这种情况下，溢出帧计数器被复位为零。

(R/SS/RC)

Overflow_FC 表示应用程序丢失的帧数。

每当 MTL FIFO 溢出时，该计数器会递增。(R/SS/RC)

Overflow_BMFC 表示丢失帧计数器的状态。

0：无溢出。

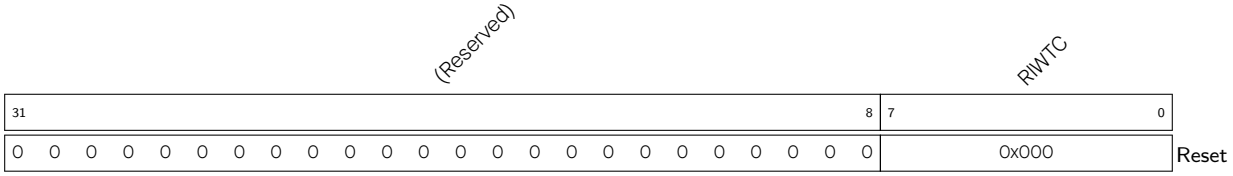
1：丢失帧计数器（Missed_FC）溢出，即主机接收缓冲区不可用，丢失帧计数器以最大值溢出。在这种情况下，丢失帧计数器被复位为零。

(R/SS/RC)

Missed_FC 表示因主机接收缓冲区不可用而被丢失的帧数。

每当 DMA 丢失一个传入帧时，该计数器会递增。(R/SS/RC)

Register 55.10. DMARINTWDTIMER_REG (0x1024)

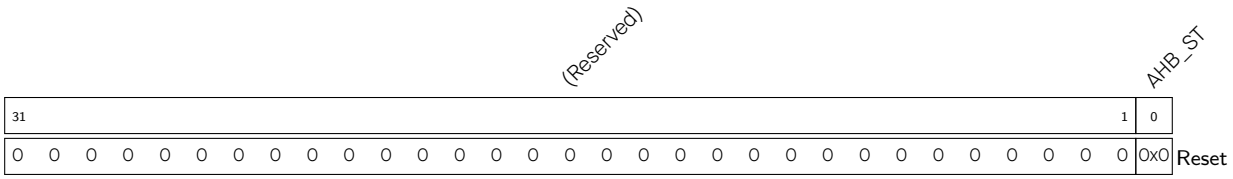


RIWTC 配置系统时钟周期乘以 256 的数量。

在 RX DMA 传输完由于相应描述符 RDES1[31] 中的位置 1 而 RI 状态位不为 1 的帧后，看门狗定时器达到配置值触时发。

看门狗定时器超时时，RI 位置 1 并停止计时器。由于任何接收帧的 RDES1[31] 的自动置 1 而使 RI 位置 1 时，看门狗定时器被复位。(R/W)

Register 55.11. DMAAHBSTATUS_REG (0x102C)



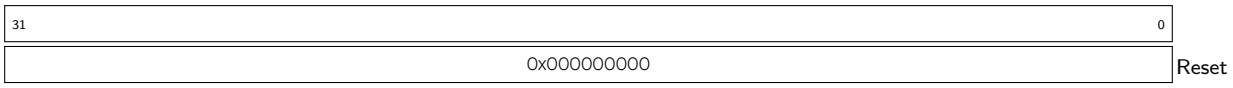
AHB_ST 配置 AHB 主接口的状态。

0：空闲

1：活跃

(R/W)

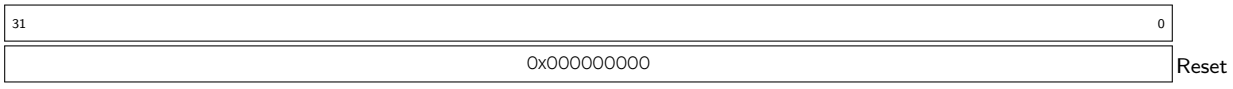
Register 55.12. DMATXCURRDESC_REG (0x1048)



TRANS_DECR_ADDR_PTR 表示 DMA 在传输期间更新的当前发送链表的地址。

此字段在复位时被清除。(RO)

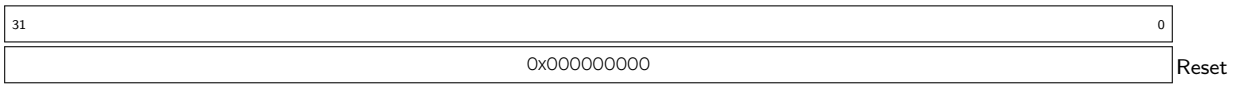
Register 55.13. DMARXCURRDESC_REG (0x104C)



RECV_DECR_ADDR_PTR 表示 DMA 在传输期间更新的当前接收链表的地址。

此字段在复位时被清除。(RO)

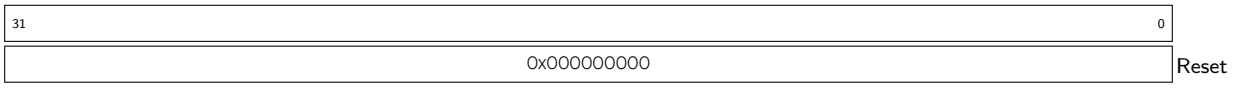
Register 55.14. DMATXCURRADDR_BUF_REG (0x1050)



TRANS_BUFF_ADDR_PTR 表示 DMA 在传输期间更新的传输缓冲区的地址。

此字段在复位时被清除。(RO)

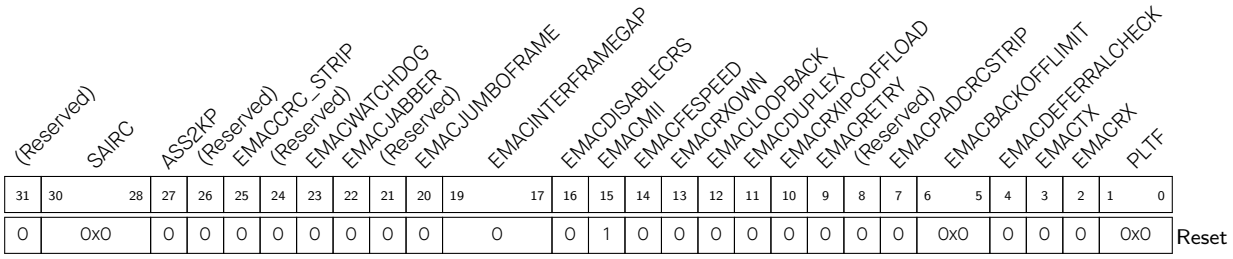
Register 55.15. DMARXCURRADDR_BUF_REG (0x1054)



RECV_BUFF_ADDR_PTR 表示 DMA 在传输期间更新的接收缓冲区的地址。

此字段在复位时被清除。(RO)

Register 55.16. EMACCONFIG_REG (0x0000)



SAIRC 配置所有传输帧的源地址插入或替换。

位 [30] 根据位 [29:28] 的值指定将哪个 MAC 地址寄存器（0 或 1）用于源地址插入或替换。

位 [29:28]：

0, 1：输入信号 mti_sa_ctrl_i 和 ati_sa_ctrl_i 控制 SA 字段生成

2：在所有传输帧的 SA 字段中插入 MAC 地址寄存器（0 或 1）的内容

3：在所有传输帧的 SA 字段中替换 MAC 地址寄存器（0 或 1）的内容

位 [30]：

0：使用 MAC 地址 0 寄存器

1：使用 MAC 地址 1 寄存器

(R/W)

ASS2KP 配置 MAC 是否将长度超过 2000 字节的接收帧视为正常数据包或巨型帧。

0：巨型帧。在这种情况下，如果 Bit[20] (JE) 为 0，则 MAC 将所有大小超过 1518 字节（标记为 1522 字节）的接收帧视为巨型帧。

1：正常数据包。

(R/W)

EMACCRC_STRIP 配置是否在将帧转发到应用程序之前剥离和丢弃所有以太网类型（长度/类型字段大于或等于 1536）帧的最后 4 个字节 (FCS)。

0：不剥离和丢弃

1：剥离和丢弃

(R/W)

EMACWATCHDOG 配置是否禁用看门狗定时器。

0：启用。在这种情况下，MAC 不允许接收超过 2048 字节（如果 JE 设置为高，则为 10240 字节）或看门狗超时寄存器配置值的接收帧。MAC 在看门狗限制字节数之后截断接收到的任何字节。

1：禁用。在这种情况下，MAC 可以接收最多 16384 字节的帧。

(R/W)

EMACJABBER 配置是否禁用发送器上的 jabber 定时器。

0：启用。在这种情况下，如果应用程序在传输过程中发送超过 2048 字节的数据（如果 JE 设置为高，则为 10240 字节），MAC 将切断发送器。

1：禁用。在这种情况下，MAC 可以传输最多 16384 字节的帧。

(R/W)

见下页...

Register 55.16. EMACCONFIG_REG (0x0000)

接上页...

EMACJUMBOFRAME 配置 MAC 是否允许 9,018 字节（带有 VLAN 标记的帧为 9,022 字节）的巨型帧，而不报告接收帧状态中的巨型帧错误。

0：不允许

1：允许

(R/W)

EMACINTERFRAMEGAP 配置传输过程中帧之间的最小帧间隔 (IFG)。

0：96 位时间

1：88 位时间

2：80 位时间

3：72 位时间

4：64 位时间

5：56 位时间

6：48 位时间

7：40 位时间

在半双工模式下，最小 IFG 只能配置为 64 位时间（IFG = 100）。(R/W)

EMACDISABLECRS 配置是否在传输过程中禁用载波侦听。

0：MAC 发送器由于载波感知而生成载波丢失或无载波错误，甚至可以中止传输。

1：MAC 发送器在半双工模式下在帧传输过程中忽略 MII CRS 信号。这个请求不会因为载波丢失或无载波而生成错误。

(R/W)

EMACMII 配置以太网信号线速度。

0：1000 Mbps

1：10 或 100 Mbps

在 10 或 100 Mbps 传输中，具体的信号线速度由此位和 FES（EMACFESPEED）位共同确定。(R/W)

EMACFESPEED 配置 MII 和 RMII 接口的速度。

0：10 Mbps

1：100 Mbps

(R/W)

EMACRXOWN 配置是否禁用接收自己的帧。

0：MAC 在传输时接收 PHY 发出的所有数据包。

1：MAC 在半双工模式下，当 TX_EN 触发中断时，禁止接收帧。

如果 MAC 在全双工模式下运行，则此位无效。(R/W)

EMACLOOPBACK 配置是否启用回环模式。

0：禁用。

1：启用。在这种情况下，MII 接收时钟输入（CLK_RX）对于回环正常工作是必需的，因为传输时钟没有内部回环。

(R/W)

见下页...

Register 55.16. EMACCONFIG_REG (0x0000)

接上页...

EMACDUPLEX 配置是否启用全双工模式，以便 MAC 可以同时发送和接收。

0：禁用

1：启用

在全双工模式下，此位为只读，默认值为 1。(R/W)

EMACRXIPCOFFLOAD 配置是否启用校验和功能。

0：禁用

1：启用

当启用校验和时，MAC 计算所有接收的以太网帧有效载荷的 16 位的补码和，再对相加和取补。它还检查 IPv4 报头校验和（接收到的以太网帧的字节 25/26 或 29/30（VLAN 标记）是否对接收帧是正确的，并在接收状态字中给出状态。MAC 还将 IP 报头数据报有效负载（IPv4 报头后的字节）16 位校验和附加到应用程序的以太网帧当中（当取消选择类型 2 COE 时）。(R/W)

EMACRETRY 配置是否禁用重试，以便 MAC 仅尝试一次传输，并且在 MII 接口上发生冲突时，MAC 忽略当前帧传输并报告传输帧状态中的帧中止和过多冲突错误。

0：启用

1：禁用

仅在半双工模式下有效。(R/W)

EMACPADCRCSTRIP 配置是否启用自动填充或 CRC 剥离。

0：禁用。在这种情况下，MAC 将所有传入的帧原样传递给主机。

1：启用。在这种情况下，MAC 仅在长度字段的值小于 1536 字节时，剥离传入帧的填充或 FCS 字段。所有长度字段大于或等于 1536 字节的接收帧将被传递给应用程序，而不剥离填充或 FCS 字段。

(R/W)

EMACBACKOFFLIMIT 配置后退限制，确定在冲突后重新尝试时，MAC 在重新调度传输尝试之前等待的随机整数个时隙时间延迟（对于 1000 Mbps 为 4096 比特时间，对于 10/100 Mbps 为 512 比特时间）。

00：k = min（n，10）

01：k = min（n，8）

10：k = min（n，4）

11：k = min（n，1）

其中 n = 重传尝试次数。

随机整数 r 的取值范围为 0 <= r < 2 的 k 次方

仅在半双工模式下有效。(R/W)

EMACDEFERRALCHECK 配置是否启用延迟检查功能。

0：禁用

1：启用

(R/W)

见下页...

Register 55.16. EMACCONFIG_REG (0x0000)

接上页...

EMACTX 配置是否启用 MAC 的发送状态机。

0：在当前帧传输完成后禁用，并且不再传输任何帧

1：在 MII 上启用传输

(R/W)

EMACRX 配置是否启用 MAC 的接收状态机。0：在当前帧接收完成后禁用，并且不再从 MII 接收任何帧

1：从 MII 接收帧启用

(R/W)

PLTF 配置在每个传输帧的开头添加的报头字节数。

0：7 字节报头

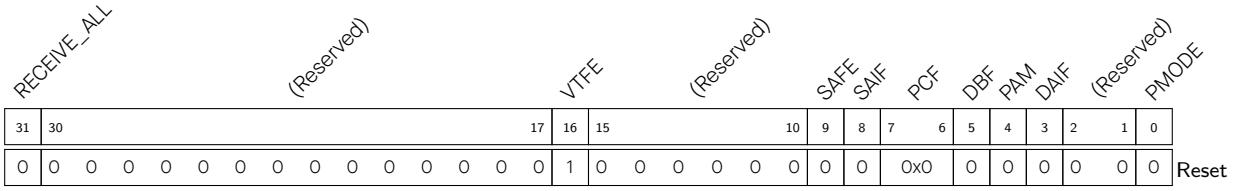
1：5 字节报头

2：3 字节报头

3：保留

只有在 MAC 出于全双工模式下时，才会减少报头字节数量。(R/W)

Register 55.17. EMACFF_REG (0x0004)



RECEIVE_ALL 配置是否传递所有接收到的帧。

0：MAC 接收模块仅将通过 SA 或 DA 地址过滤器的帧传递给应用程序。

1：MAC 接收模块将所有接收到的帧传递给应用程序，无论它们是否通过地址过滤器。SA 或 DA 过滤的结果（通过或失败）将更新到接收状态字中的相应位。

(R/W)

VTFE 配置是否启用 VLAN 标签过滤器。

0：禁用。在这种情况下，MAC 转发所有帧，无论 VLAN 标签的匹配状态如何。

1：启用。在这种情况下，MAC 丢弃不匹配 VLAN 标签比较的带有 VLAN 标签的帧。

(R/W)

SAFE 配置是否启用源地址过滤器。

0：禁用。在这种情况下，MAC 根据 SA 地址比较将接收到的帧传递给应用程序，并根据 SA 地址比较更新 RX 状态的 SAF 位。

1：启用。在这种情况下，MAC 将接收到的帧的 SA 字段与已启用的 SA 寄存器中编程的值进行比较。如果比较失败，MAC 将丢弃该帧。

(R/W)

SAIF 配置是否启用 SA 反向过滤。

0：禁用。在这种情况下，SA 不匹配 SA 寄存器的帧被标记为未通过 SA 地址过滤器。

1：启用。在这种情况下，地址检查块以 SA 地址比较的反向过滤模式运行。与 SA 寄存器匹配的帧被标记为未通过 SA 地址过滤器。

(R/W)

PCF 配置所有控制帧（包括单播和多播暂停帧）的转发。

0：MAC 过滤所有控制帧。

1：MAC 将所有控制帧（除了 PAUSE 控制帧）转发给应用程序，包括未通过地址过滤器的帧。

2：MAC 将所有控制帧转发给应用程序，包括未通过地址过滤器的帧。

3：MAC 转发通过地址过滤器的控制帧。

暂停帧的处理应满足以下条件：

条件 1：MAC 处于全双工模式，并且通过将寄存器 6（流控制寄存器）的位 2 (RFE) 置为 1 启用了流控制。

条件 2：当寄存器 6（流控制寄存器）的位 3 (UP) 置 1 时，接收到的帧的目标地址 (DA) 与特殊多播地址或 EMACADDR0 匹配。

条件 3：接收到的帧的类型字段为 0x8808，OPCODE 字段为 0x0001。

(R/W)

DBF 配置是否禁用广播帧。

0：启用。AFM（地址过滤模块）模块传递所有接收到的广播帧。

1：禁用。AFM 模块过滤所有接收到的广播帧，并且所有其他过滤设置无效。

(R/W)

见下页...

Register 55.17. EMACFF_REG (0x0004)

接上页...

PAM 配置是否传递所有多播帧。

0：多播帧的过滤取决于 HMC 位。

1：传递所有具有多播目标地址（目标地址字段的第一位为'1'）的接收到的帧。

(R/W)

DAIF 配置是否启用单播和多播帧的 DA 反向过滤。

0：禁用

1：启用

(R/W)

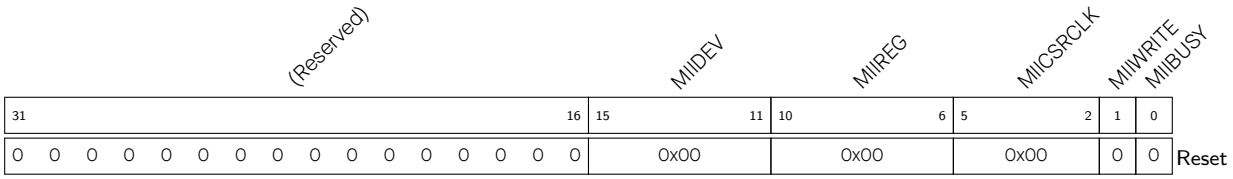
PMODE 配置是否启用混杂模式。

0：禁用。

1：启用。在这种情况下，地址过滤模块传递所有传入的帧，无论其目标或源地址如何。当设置 PR（PRI_RATIO）时，接收状态字的 SA 或 DA 过滤失败状态位始终被清除。

(R/W)

Register 55.18. EMACMIIADDR_REG (0x0010)



MIIDEV 配置正在访问的是 32 个 PHY 设备中的哪一个。(R/W)

MIIREG 配置所选 PHY 设备中所需的地址寄存器。(R/W)

MIICSRCLK 配置 CSR 时钟频率。

0：APB 时钟频率为 60 ~ 100 MHz，MDC 时钟频率为 APB_CLK/42

1：APB 时钟频率为 100 ~ 150 MHz，MDC 时钟频率为 APB_CLK/62

2：APB 时钟频率为 20 ~ 35 MHz，MDC 时钟频率为 APB_CLK/16

3：APB 时钟频率为 35 ~ 60 MHz，MDC 时钟频率为 APB_CLK/26

4：APB 时钟频率为 150 ~ 250 MHz，MDC 时钟频率为 APB_CLK/102

其他值保留

(R/W)

MIIWRITE 配置使用 MII_DATA 的操作的方向。

0：读操作

1：写操作

(R/W)

MIIBUSY 配置 MII 的状态。

0：空闲

1：忙碌

此位与MIIREG 和MII_DATA 结合使用。

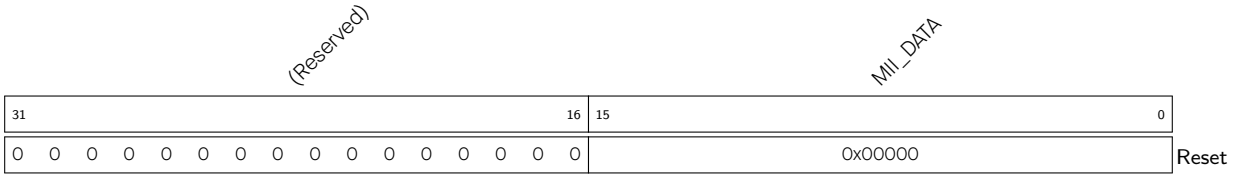
在写入MIIREG 和MII_DATA 之前，此位应读取逻辑 0（默认值）。

要读取或写入MIIREG 和MII_DATA，软件（用户）应将此字段设置为 1。

在此位被硬件 (MAC) 清除之前，访问MII_DATA 时应保持有效（数据保持不变）。

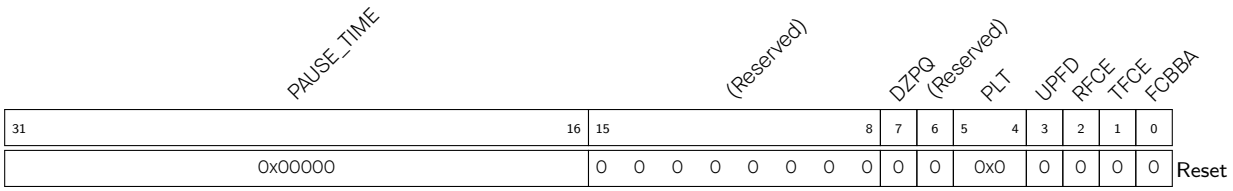
请注意，ESP32-P4 MAC 在对MIIREG 和MII_DATA 进行读取或写入访问时不会从 PHY 接收到 ACK。(读/写/特殊控制)

Register 55.19. EMACMIIDATA_REG (0x0014)



MII_DATA 配置在管理读操作之后从 PHY 读取的 16 位数据值，或在管理写操作之前要写入 PHY 的 16 位数据值。(R/W)

Register 55.20. EMACFC_REG (0x0018)



PAUSE_TIME 配置传输控制帧中暂停时间字段的值。如果暂停时间位配置为与 (G) MII 时钟域双同步，则在目标时钟域中至少经过四个时钟周期后才能对该寄存器连续写入。(R/W)

DZPQ 配置是否禁用自动生成暂停时间字段为 0 的控制帧。

0：启用

1：禁用

(R/W)

PLT 配置暂停计时器的阈值，达到该阈值时自动重新传输暂停帧。

0：阈值为暂停时间减去 4 个时隙时间（PT - 4 个时隙时间）。

1：阈值为暂停时间减去 28 个时隙时间（PT - 28 个时隙时间）。

2：阈值为暂停时间减去 144 个时隙时间（PT - 144 个时隙时间）。

3：阈值为暂停时间减去 256 个时隙时间（PT - 256 个时隙时间）。

阈值应始终小于位 [31:16] 中配置的暂停时间。例如，如果 PT = 100H（256 个时隙时间），PLT = 01，则在第一个暂停帧传输后的 228 (256 - 28) 个时隙时间后自动传输第二个暂停帧。(R/W)

UPFD 配置是否启用单播暂停帧检测。

0：禁用。MAC 仅检测具有唯一多播地址的暂停帧。

1：启用。MAC 还可以检测站点单播地址的暂停帧。此单播地址应与 EMACADDR0 高寄存器和 EMACADDR0 低寄存器中指定的地址相同。(R/W)

RFCE 配置是否启用暂停时间的解码功能。

0：禁用

1：启用。MAC 解码接收到的暂停帧，并在指定的（暂停）时间内禁用其发送器。(R/W)

TFCE 配置是否在全双工模式下启用传输流控制，以及是否在半双工模式下启用背压功能。

0：禁用

1：启用

(R/W)

FCBBA 配置是否在全双工模式下启用暂停控制帧，并在半双工模式下启用背压功能（如果设置了 TFCE 位）。

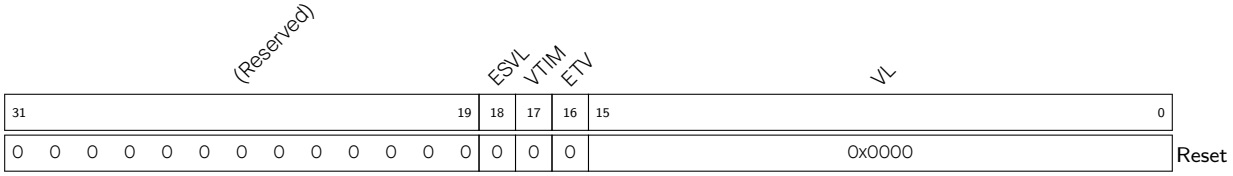
0：禁用

1：启用

在全双工模式下，在写入流控制寄存器之前，应将此位读为 0。要启动暂停控制帧，应用程序必须将此位设置为 1。在传输控制帧传输期间，此位继续设置，表示正在进行帧传输。暂停控制帧传输完成后，MAC 将此位复位为 0。在清除此位之前，不应写入流控制寄存器。(R/W/SC)

在半双工模式下，此位（并且设置了 TFCE 位）为 1 时，MAC 启用背压。在背压期间，当 MAC 接收到新帧时，发送器开始发送 JAM，导致冲突。当 MAC 配置为全双工模式时，BPA 会自动禁用。(R/W)

Register 55.21. EMACVLANTAG_REG (0x001C)



ESVL 配置是否启用 S-VLAN。

0：禁用。

1：启用。MAC 发送器和接收器还将 S-VLAN（类型 =0x88A8）帧视为有效的 VLAN 标记帧。(R/W)

VTIM 配置是否启用 VLAN 标记反向匹配。

0：禁用。具有匹配 VLAN 标记的帧被标记为匹配。

1：启用。没有匹配 VLAN 标记的帧被标记为匹配。(R/W)

ETV 配置是否启用 12 位 VLAN 标记比较。

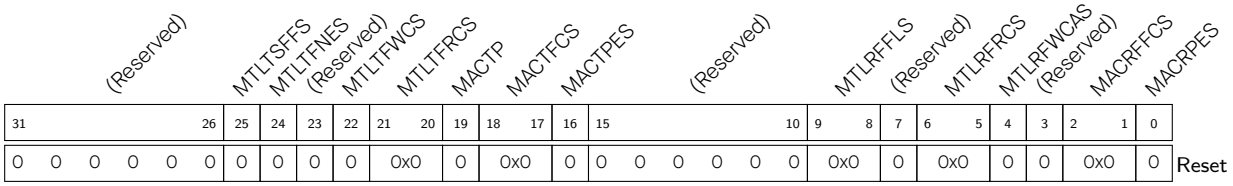
0：禁用。接收到的 VLAN 帧的第 15 个和第 16 个字节的所有 16 位用于比较和 VLAN 哈希过滤。

1：启用。使用 12 位 VLAN 标识符进行比较和过滤，而不是完整的 16 位 VLAN 标记。VLAN 标记的位 [11:0] 与接收到的 VLAN 标记帧中的相应字段进行比较。(R/W)

VL 配置 802.1Q VLAN 标记以识别 VLAN 帧，并与接收到的帧的第 15 个和第 16 个字节进行比较。当 ETV 位置 1 时，仅使用 VID（位 [11:0]）进行比较。

如果 VL（如果设置了 ETV，则为 VL[11:0]）全为零，则 MAC 不会检查第 15 个和第 16 个字节以进行 VLAN 标记比较，并将所有类型字段值为 0x8100 或 0x88a8 的帧视为 VLAN 帧。(R/W)

Register 55.22. EMACDEBUG_REG (0x0024)



MTLTSFFS 表示 MTL TX 状态 FIFO 是否已满并可以接受更多的帧进行传输。

0：未满

1：已满

(RO)

MTLTFNES 表示 MTL TX FIFO 是否为空，是否还有数据待传输。

0：空

1：非空

(RO)

MTLTFWCS 表示 MTL TX FIFO 写控制器是否处于活跃状态，并将数据传输到 TX FIFO。

0：不活跃

1：活跃

(RO)

MTLTFRCS 表示 TX FIFO 读控制器的状态。

0：空闲状态

1：读取状态（将数据传输到 MAC 发送器）

2：等待来自 MAC 发送器的 TXStatus

3：写入接收到的 TXStatus 或清空 TX FIFO

(RO)

MACTP 表示 MAC 发送器是否处于暂停状态（仅全双工模式），因此不安排任何帧进行传输。

0：未暂停

1：已暂停

(RO)

MACTFCS 表示 MAC 发送帧控制器模块的状态。

0：空闲状态

1：等待前一帧的状态或 IFG 或退避期结束

2：生成并传输 PAUSE 控制帧（全双工模式）

3：传输输入帧进行传输

(RO)

MACTPES 表示 MAC MII 传输协议引擎的状态。

0：空闲状态

1：主动传输数据

(RO)

见下页...

Register 55.22. EMACDEBUG_REG (0x0024)

接上页...

MTLRFFLS 表示 RX FIFO 的填充级别状态。

0：RX FIFO 为空

1：RX FIFO 填充级别低于流控停用阈值

2：RX FIFO 填充级别高于流控激活阈值

3：RX FIFO 已满

(RO)

MTLRFRCS 表示 RX FIFO 读控制器的状态。

0：空闲状态

1：读取帧数据

2：保留

3：清空帧数据和状态

(RO)

MTLRFWCAS 表示 MTL RX FIFO 写控制器的状态。

0：不活跃

1：活跃，并将接收到的帧传输到 FIFO。

(RO)

MACRFFCS 表示 MAC 接收帧控制器模块的小 FIFO 读（位 1）和写（位 0）控制器的状态。

0：不活跃

1：活跃

(RO)

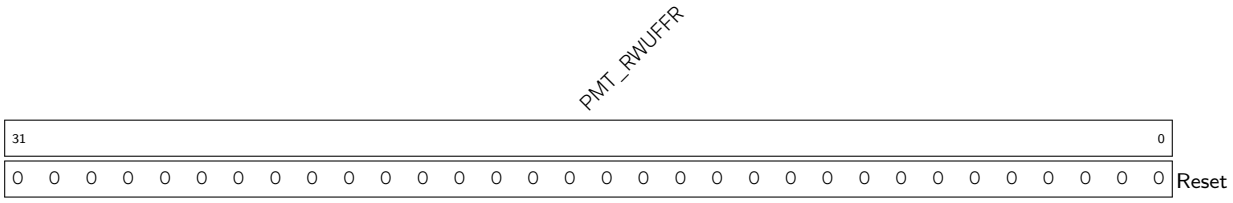
MACRPES 表示 MAC GMII 或 MII 接收协议引擎的状态。

0：空闲

1：主动接收数据

(RO)

Register 55.23. PMT_RWUFFR_REG (0x0028)



WKUPPKTFILTER 当 RWKPTR 为 0 ~ 3 时，配置过滤器 0 ~ 3 的唤醒帧的屏蔽。

Bit[30:0] 对应于偏移量 +j（j = 0 ~ 30）。

Bit[31] 必须为 0。0：不屏蔽

1：屏蔽

当 RWKPTR 为 4 时，bit[3]/bit[11]/bit[19]/bit[27] 配置目标地址类型，bit[0]/bit[8]/bit[16]/bit[24] 配置是否启用过滤器 0 ~ 3。

0：单播/禁用

1：多播/启用

当 RWKPTR 为 5 时，配置过滤器 0 ~ 3 屏蔽字节的偏移量。

Bit[7:0] 配置过滤器 0 屏蔽字节的偏移量，[31:24] 配置过滤器 3 屏蔽字节的偏移量。

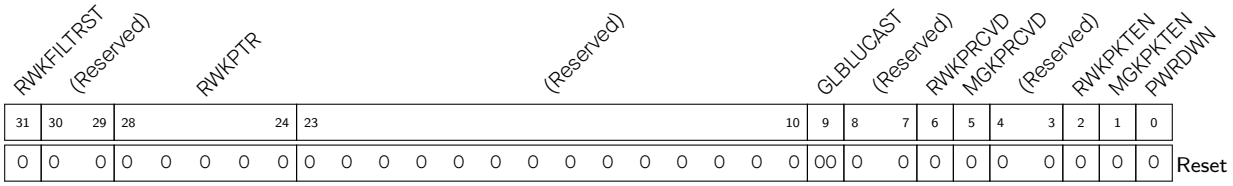
当 RWKPTR 为 6 ~ 7 时，配置基于以下公式的过滤器 0 ~ 3 屏蔽字节的 16 位 CRC 值：

$G(x) = x^{16} + x^{15} + x^2 + 1$

Bit[7:0] 配置 RWKPTR 为 6 时过滤器 0 或过滤器 2 屏蔽字节的 CRC 值，bit[31:16] 配置 RWKPTR 为 6 时过滤器 1 或过滤器 3 屏蔽字节的 CRC 值。

(R/W)

Register 55.24. PMT_CSR_REG (0x002C)



RWKFILTRST 配置是否重置 RWKPTR 寄存器。

0：释放复位

1：复位

(R/WS/SC)

RWKPTR 配置 PMT_RWUFFR 寄存器的用途。详细信息请参见 PMT_RWUFFR 的描述。(RO)

GLBLUCAST 配置是否将目标地址过滤器的单播帧作为唤醒帧启用。

0：禁用

1：启用

(R/W)

RWKPRCVD 表示是否接收到远程唤醒帧。

0：未接收

1：已接收

(R/SS/RC)

MGKPRCVD 表示是否接收到 magic 帧。

0：未接收

1：已接收

(R/SS/RC)

RWKPKTEN 配置是否使用远程唤醒帧启用电源管理。

0：禁用

1：启用

(R/W)

MGKPKTEN 配置是否使用 magic 帧启用电源管理。

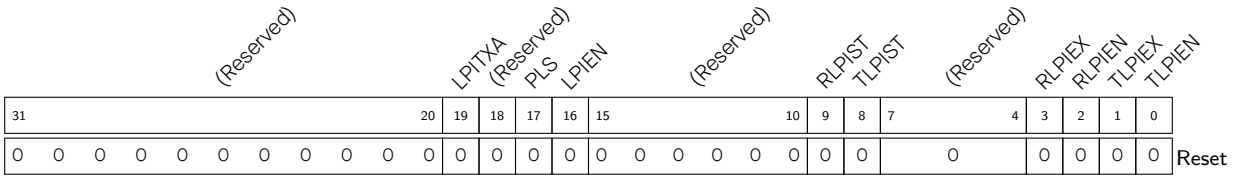
0：禁用

1：启用

(R/W)

PWRDWN 配置 EMAC 接收器是否丢弃所有接收到的帧，直到接收到预期的 magic 帧或远程唤醒帧。仅当 MGKPKTEN、GLBLUCAST 或 RWKPKTEN 为 1 时有效。(R/WS/SC)

Register 55.25. EMACLPI_CSR_REG (0x0030)



LPITXA 配置 MAC 在传输端进入或退出 LPI 模式时的行为。

0：当 MAC 进入或退出 LPI 模式时，此位直接控制 MAC 的行为。

1：当 MAC 与 LPIEN 一起为 1 时，此位与 LPIEN 一起控制 MAC 在进入或退出 LPI 模式时的行为。当核心中的所有未完成帧和应用接口中的所有待处理帧都已传输完毕时，MAC 才进入 LPI 模式。当应用发送任何帧进行传输时，MAC 退出 LPI 模式。

(R/W)

PLS 配置 PHY 的链路状态。

0：链路正常

1：链路断开

(R/W)

LPIEN 配置 EMAC 发送器是否进入 LPI 状态。

0：退出

1：进入

当 LPITXA 位置 1 且 MAC 因为接收到新的待传输数据包而退出 LPI 状态时，此位被清除。

(R/W/SC)

RLPIST 表示接收 LPI 状态。

0：未激活

1：EMAC 正在接收 LPI 模式

(RO)

TLPIST 表示传输 LPI 状态。

0：未激活

1：EMAC 正在传输 LPI 模式

(RO)

RLPIEX 表示接收 LPI 退出状态。

0：未激活

1：EMAC 接收器停止接收 LPI 模式，退出 LPI 状态，并恢复正常接收

(R/SS/RC)

RLPIEN 表示接收 LPI 进入状态。

0：未激活

1：EMAC 接收器接收到 LPI 模式并进入 LPI 状态

(R/SS/RC)

TLPIEX 表示传输 LPI 退出状态。

0：未激活

1：用户清除 LPIEN 位且 LPI_TW_TIMER 已过期后，EMAC 发送器退出 LPI 状态

(R/SS/RC)

见下页...

Register 55.25. EMACLPI_CSR_REG (0x0030)

接上页...

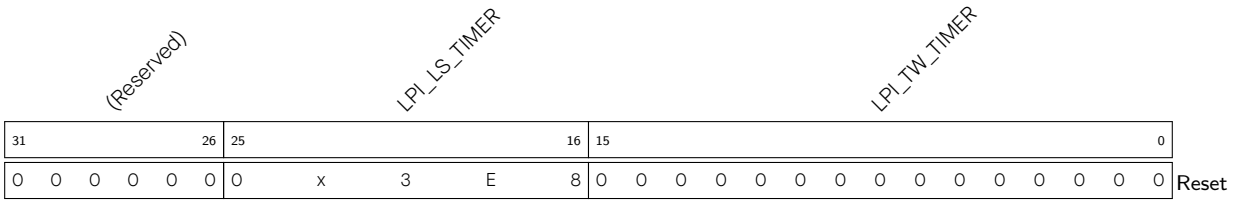
TLPIEN 表示传输 LPI 进入状态。

0：未激活

1：EMAC 发送器因为 LPIEN 位的设置而进入 LPI 状态

(R/SS/RC)

Register 55.26. EMACLPITIMERSCONTROL_REG (0x0034)



LPI_LS_TIMER 配置 PHY 的链路状态应保持的最短时间。

单位：毫秒。

默认值为 IEEE 标准中定义的 1000（1 秒）。

(R/W)

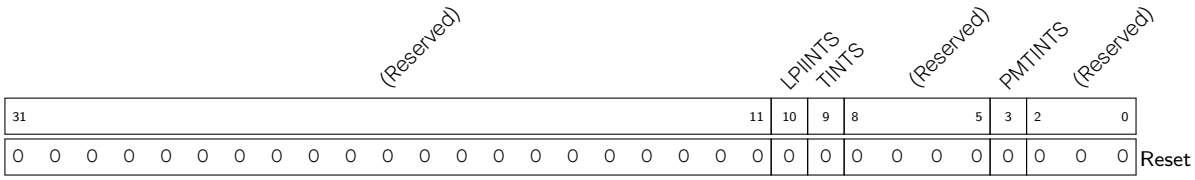
LPI_TW_TIMER 配置 MAC 在停止传输 LPI 模式到 PHY 并恢复正常传输之前等待的最短时间。

单位：毫秒。

此计时器超时后，TLPIEX 状态位被置 1。

(R/W)

Register 55.27. EMACINTS_REG (0x0038)

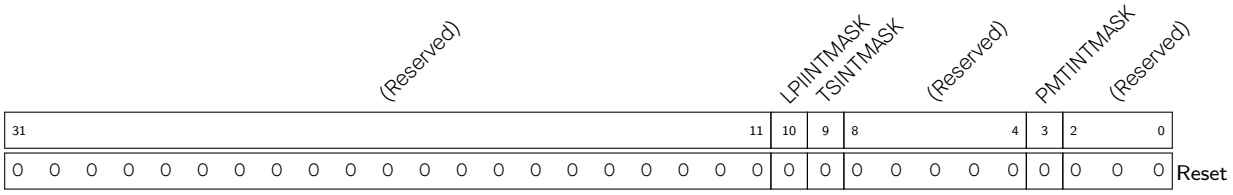


LPIINTS EMAC_LPI_INT 的原始中断状态。(RO)

TINTS TS_TRI_INT 的原始中断状态。(R/SS/RC)

PMTINTS EAMC_PMP_INT 的原始中断状态。(RO)

Register 55.28. EMACINTMASK_REG (0x003C)

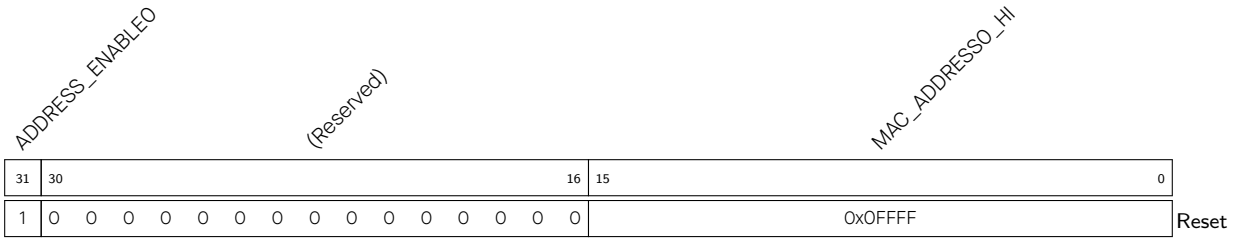


LPIINTMASK 写 1 屏蔽 EMAC_LPI_INT。(R/W)

TSINTMASK 写 1 屏蔽 TS_TRI_INT。(R/W)

PMTINTMASK 写 1 屏蔽 EMAC_PMP_INT。(R/W)

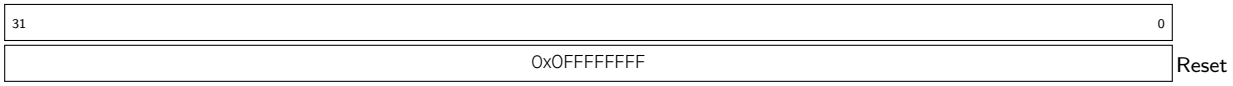
Register 55.29. EMACADDR0HIGH_REG (0x0040)



ADDRESS_ENABLE0 该位始终为 1。(RO)

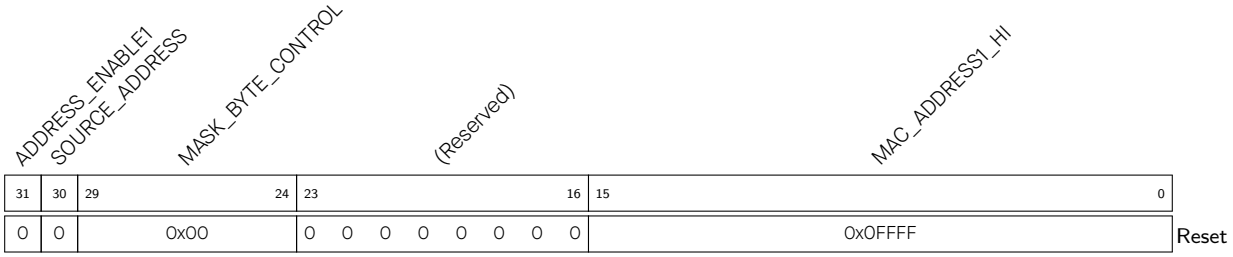
MAC_ADDRESS0_HI 配置第一个 6 字节 MAC 地址 [47:32] 的高 16 位。
MAC 使用此字段来过滤接收的帧并将 MAC 地址插入传输流控制 (PAUSE) 帧中。(R/W)

Register 55.30. EMACADDR0LOW_REG (0x0044)



EMACADDR0LOW_REG 配置第一个 6 字节 MAC 地址的低 32 位。
MAC 使用此字段来过滤接收的帧并将 MAC 地址插入传输流控制 (PAUSE) 帧中。(R/W)

Register 55.31. EMACADDR1HIGH_REG (0x0048)



ADDRESS_ENABLE1 配置是否使用第二个 MAC 地址进行完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS 配置接收帧的 EMACADDR1[47:0] 与哪些字段进行比较。

0：DA 字段

1：SA 字段

(R/W)

MASK_BYTE_CONTROL 配置在将接收到的 DA 或 SA 与 EMACADDR1 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系:

位 [29]：EMACADDR1HIGH_REG[15:8]

位 [28]：EMACADDR1HIGH_REG[7:0]

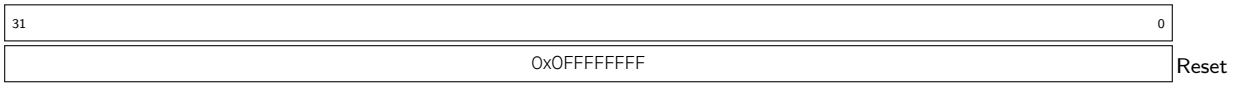
位 [27]：EMACADDR1LOW_REG[31:24]

位 [24]：EMACADDR1LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS1_HI 配置第二个 6 字节 MAC 地址 [47:32] 的高 16 位。(R/W)

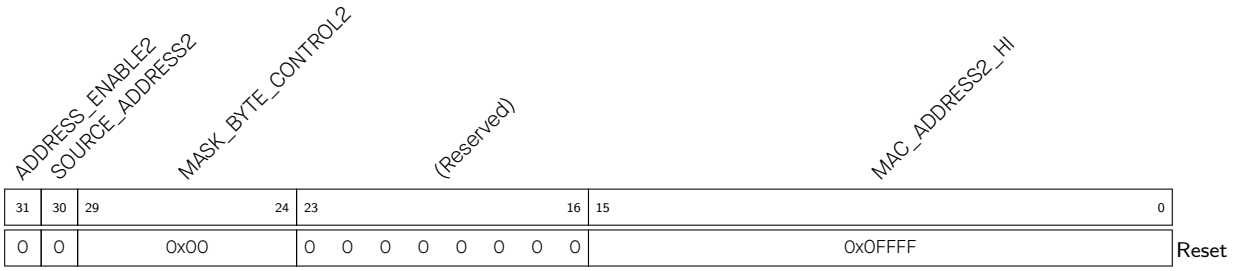
Register 55.32. EMACADDR1LOW_REG (0x004C)



EMACADDR1LOW_REG 配置第二个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.33. EMACADDR2HIGH_REG (0x0050)



ADDRESS_ENABLE2 配置是否启用第三个 MAC 地址的完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS2 配置接收帧的 EMACADDR2[47:0] 与哪些字段进行比较。

0：目的地址字段

1：源地址字段

(R/W)

MASK_BYTE_CONTROL2 配置在将接收到的目的地址或源地址与 EMACADDR2 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系如下:

位 [29]：EMACADDR2HIGH_REG[15:8]

位 [28]：EMACADDR2HIGH_REG[7:0]

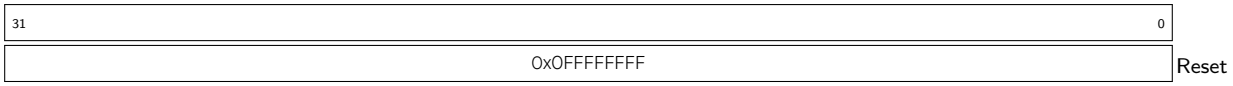
位 [27]：EMACADDR2LOW_REG[31:24]

位 [24]：EMACADDR2LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS2_HI 配置第三个 6 字节 MAC 地址 [47:32] 的高 16 位。(R/W)

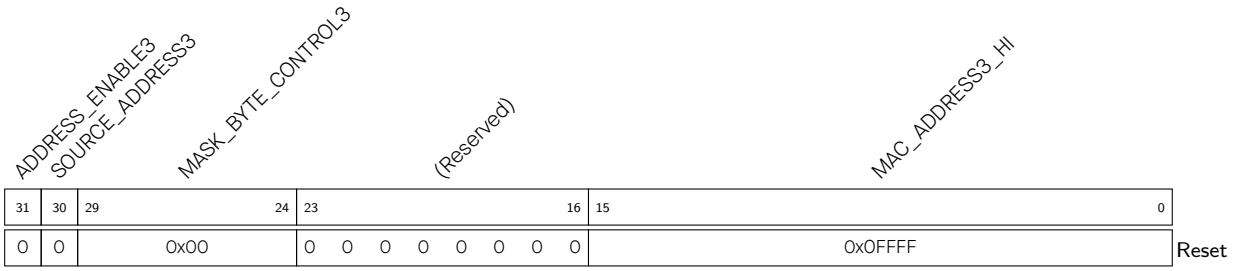
Register 55.34. EMACADDR2LOW_REG (0x0054)



EMACADDR2LOW_REG 配置第三个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.35. EMACADDR3HIGH_REG (0x0058)



ADDRESS_ENABLE3 配置是否启用第四个 MAC 地址的完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS3 配置接收帧的 EMACADDR3[47:0] 与哪些字段进行比较。

0：目的地址字段

1：源地址字段

(R/W)

MASK_BYTE_CONTROL3 配置在将接收到的目的地址或源地址与 EMACADDR3 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系如下:

位 [29]：EMACADDR3HIGH_REG[15:8]

位 [28]：EMACADDR3HIGH_REG[7:0]

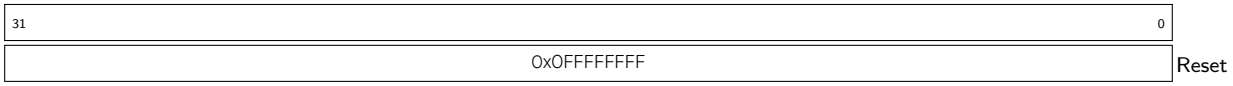
位 [27]：EMACADDR3LOW_REG[31:24]

位 [24]：EMACADDR3LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS3_HI 配置第四个 6 字节 MAC 地址 [47:32] 的高 16 位。(R/W)

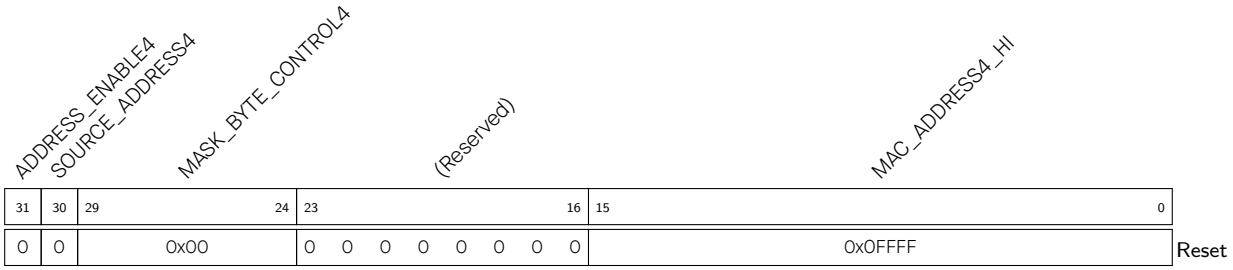
Register 55.36. EMACADDR3LOW_REG (0x005C)



EMACADDR3LOW_REG 配置第四个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.37. EMACADDR4HIGH_REG (0x0060)



ADDRESS_ENABLE4 配置是否启用第五个 MAC 地址的完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS4 配置接收帧的 EMACADDR4[47:0] 与哪些字段进行比较。

0：目的地址字段

1：源地址字段

(R/W)

MASK_BYTE_CONTROL4 配置在将接收到的目的地址或源地址与 EMACADDR4 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系如下:

位 [29]：EMACADDR4HIGH_REG[15:8]

位 [28]：EMACADDR4HIGH_REG[7:0]

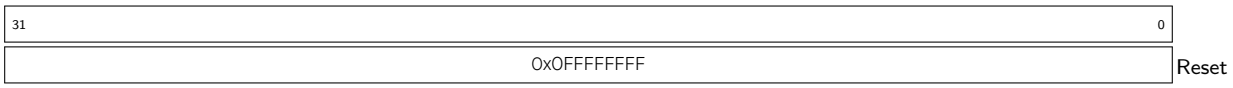
位 [27]：EMACADDR4LOW_REG[31:24]

位 [24]：EMACADDR4LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS4_HI 配置第五个 6 字节 MAC 地址 [47:32] 的高 16 位。(R/W)

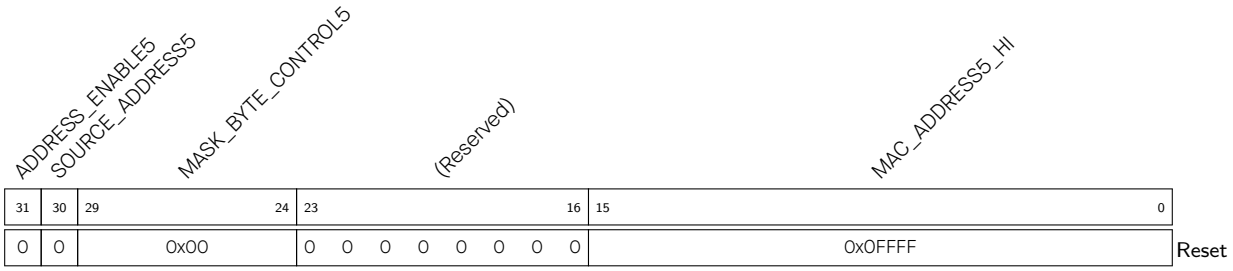
Register 55.38. EMACADDR4LOW_REG (0x0064)



EMACADDR4LOW_REG 配置第五个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.39. EMACADDR5HIGH_REG (0x0068)



ADDRESS_ENABLE5 配置是否使用第六个 MAC 地址进行完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS5 配置接收帧的 EMACADDR5[47:0] 与哪些字段进行比较。

0：目的地址字段

1：源地址字段

(R/W)

MASK_BYTE_CONTROL5 配置在将接收到的目的地址或源地址与 EMACADDR5 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系如下:

位 [29]：EMACADDR5HIGH_REG[15:8]

位 [28]：EMACADDR5HIGH_REG[7:0]

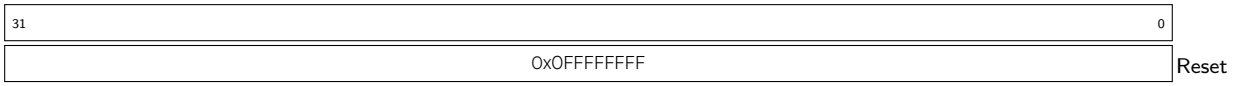
位 [27]：EMACADDR5LOW_REG[31:24]

位 [24]：EMACADDR5LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS5_HI 配置第七个 6 字节 MAC 地址的高 16 位 [47:32]。(R/W)

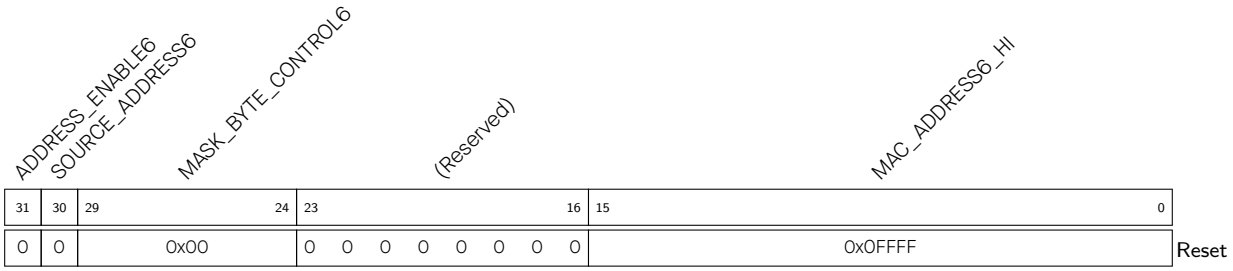
Register 55.40. EMACADDR5LOW_REG (0x006C)



EMACADDR5LOW_REG 配置第七个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.41. EMACADDR6HIGH_REG (0x0070)



ADDRESS_ENABLE6 配置是否使用第七个 MAC 地址进行完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS6 配置接收帧的 EMACADDR6[47:0] 与哪些字段进行比较。

0：目的地址字段

1：源地址字段

(R/W)

MASK_BYTE_CONTROL6 配置在将接收到的目的地址或源地址与 EMACADDR6 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系如下:

位 [29]：EMACADDR6HIGH_REG[15:8]

位 [28]：EMACADDR6HIGH_REG[7:0]

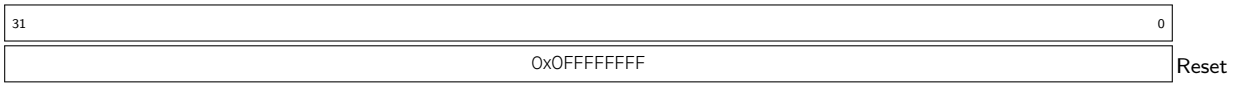
位 [27]：EMACADDR6LOW_REG[31:24]

位 [24]：EMACADDR6LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS6_HI 配置第七个 6 字节 MAC 地址的高 16 位 [47:32]。(R/W)

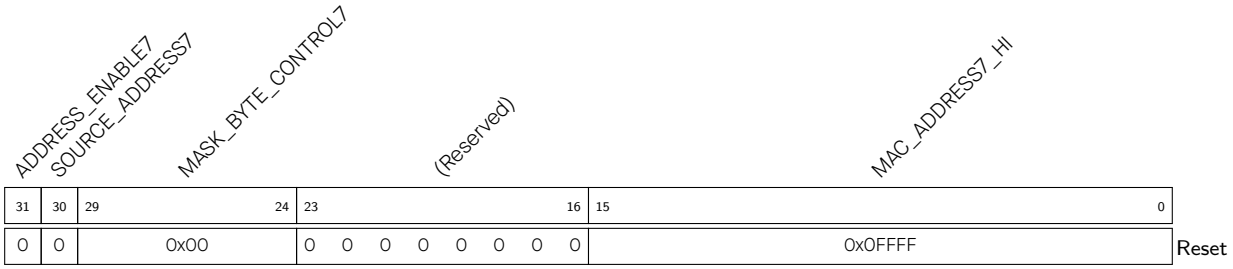
Register 55.42. EMACADDR6LOW_REG (0x0074)



EMACADDR6LOW_REG 配置第七个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.43. EMACADDR7HIGH_REG (0x0078)



ADDRESS_ENABLE7 配置是否启用第八个 MAC 地址的完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS7 配置接收帧的 EMACADDR7[47:0] 与哪些字段进行比较。

0：目的地址字段

1：源地址字段

(R/W)

MASK_BYTE_CONTROL7 配置在将接收到的目的地址或源地址与 EMACADDR7 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系如下：

位 [29]：EMACADDR7HIGH_REG[15:8]

位 [28]：EMACADDR7HIGH_REG[7:0]

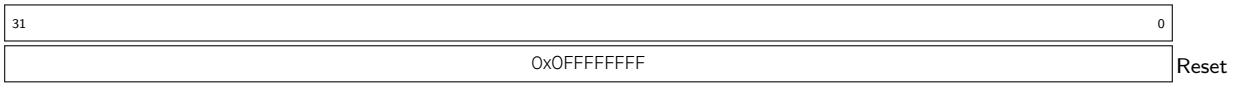
位 [27]：EMACADDR7LOW_REG[31:24]

位 [24]：EMACADDR7LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS7_HI 配置第八个 6 字节 MAC 地址 [47:32] 的高 16 位。(R/W)

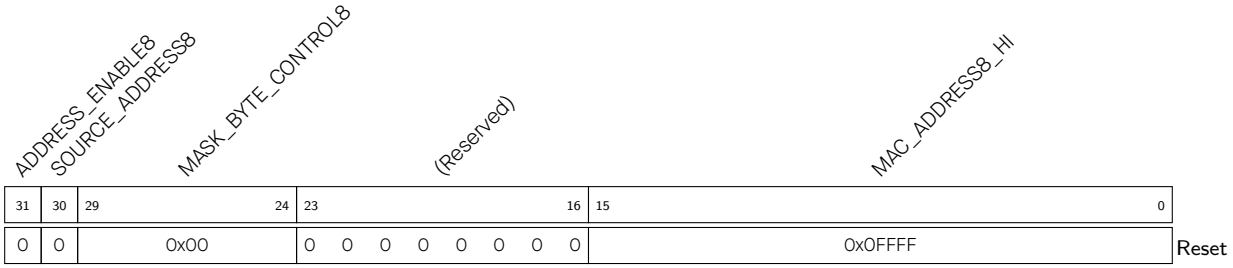
Register 55.44. EMACADDR7LOW_REG (0x007C)



EMACADDR7LOW_REG 配置第八个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.45. EMACADDR8HIGH_REG (0x0080)



ADDRESS_ENABLE8 配置是否启用第九个 MAC 地址的完美过滤。

0：禁用

1：启用

(R/W)

SOURCE_ADDRESS8 配置接收帧的 EMACADDR8[47:0] 与哪些字段进行比较。

0：目的地址字段

1：源地址字段

(R/W)

MASK_BYTE_CONTROL8 配置在将接收到的目的地址或源地址与 EMACADDR8 的内容进行比较时是否屏蔽 MAC 地址字节。

0：不屏蔽

1：屏蔽

屏蔽控制位与字节的对应关系如下：

位 [29]：EMACADDRH8IGH_REG[15:8]

位 [28]：EMACADDR8HIGH_REG[7:0]

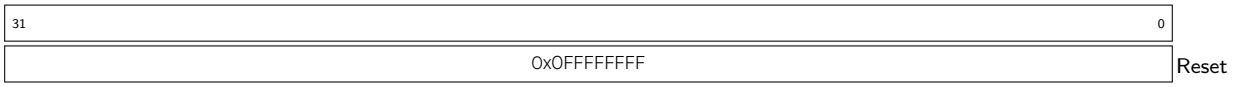
位 [27]：EMACADDR8LOW_REG[31:24]

位 [24]：EMACADDR8LOW_REG[7:0]

通过屏蔽一个或多个字节的地址，可以过滤一组地址（也称为组地址过滤）。(R/W)

MAC_ADDRESS8_HI 配置第九个 6 字节 MAC 地址 [47:32] 的高 16 位。(R/W)

Register 55.46. EMACADDR8LOW_REG (0x0084)



EMACADDR8LOW_REG 配置第九个 6 字节 MAC 地址的低 32 位。

在初始化完成后，应用程序加载该字段之前，配置该字段。(R/W)

Register 55.47. EMACCSTATUS_REG (0x00D8)



JABBER_TIMEOUT 表示是否在接收过程中发生 jabber 错误。

0：未发生

1：发生

(RO)

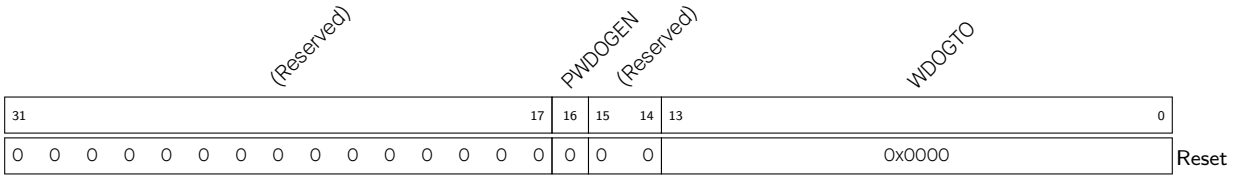
LINK_MODE 表示链路的当前工作模式。

0：半双工

1：全双工

(RO)

Register 55.48. EMACWDOGTO_REG (0x00DC)



PWDOGEN 配置看门狗超时是否可配置。

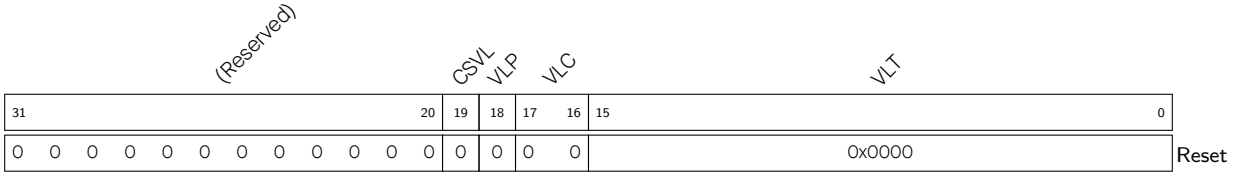
0：不可配置。接收到的帧的看门狗超时由 EMACCONFIG_REG[23] (WD) 和 EMACCONFIG_REG[20] (JE) 的配置控制。

1：如果 EMACCONFIG_REG[23] (WD) 为 0，则可通过 WDOGTO 字段进行配置。

(R/W)

WDOGTO 配置当 PWDOGEN 为 1 且 EMACCONFIG_REG[23] (WD) 为 0 时，接收帧的看门狗超时。如果接收到的帧的长度超过此字段的值，则该帧将终止并视为错误帧。(R/W)

Register 55.49. EMACTXVLTCTRL_REG (0x0584)



CSVL 配置在传输帧的第 13 和第 14 个字节中插入或替换的类型。

0：C-VLAN 类型 (0x8100)

1：S-VLAN 类型 (0x88A8)

(R/W)

VLP 配置 VLAN 删除、插入或替换的控制方式。

0：控制输入信号

1：VLC

(R/W)

VLC 配置传输帧中的 VLAN 标签。

0：不删除、插入或替换 VLAN 标签。

1：删除 VLAN 标签。MAC 会删除所有带有 VLAN 标签的传输帧的 VLAN 类型（字节 13 和 14）和 VLAN 标签（字节 15 和 16）。

2：插入 VLAN 标签。MAC 在插入字节 13 和 14 的 Type 值 (0x8100/0x88a8) 之后，在帧的字节 15 和 16 中插入 VLT。无论传输帧是否已经有 VLAN 标签，此操作都会在所有传输帧上执行。

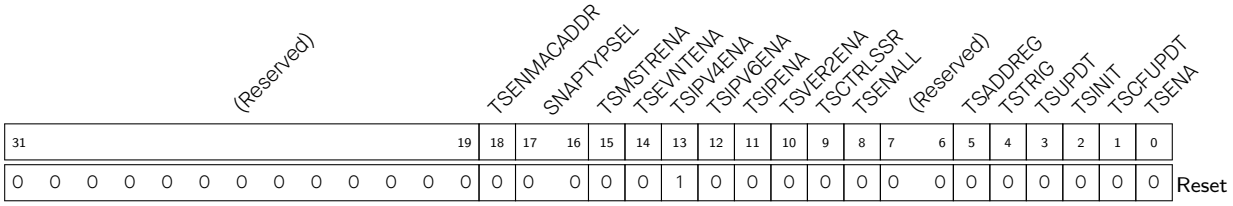
3：替换 VLAN 标签。MAC 会替换所有 VLAN 类型的传输帧的字节 15 和 16 中的 VLT（字节 13 和 14 为 0x8100/0x88a8）。

(R/W)

VLT 配置要插入或替换的 VLAN 标签的值。

只有在传输线处于非活跃状态或初始化阶段时才能更改该值。(R/W)

Register 55.50. EMACTSTPCTRL_REG (0x0700)



TSENMACADDR 配置 PTP 直接通过以太网发送时，是否启用 DA MAC 地址（与任何 MAC 地址寄存器匹配）用于 PTP 帧过滤。

0：禁用

1：启用

(R/W)

SNAPTYPSEL 配置需要对其进行快照的 PTP 数据包类型集合，与 TSMSTRENA 和 TSEVNTENA 结合使用。

SNAPTYPSEL，TSMSTRENA，TSEVNTENA:

00x0：SYNC，Follow_Up，Delay_Req，Delay_Resp

0001：SYNC

0011：Delay_Req

01x0：SYNC，Follow_Up，Delay_Req，Delay_Resp，Pdelay_Req，Pdelay_Resp，Pdelay_Resp_Follow_Up

0101：SYNC，Pdelay_Req，Pdelay_Resp

0111：Delay_Req，Pdelay_Req，Pdelay_Resp

10xx：SYNC，Delay_Req

11xx：Pdelay_Req，Pdelay_Resp

(R/W)

TSMSTRENA 配置是否仅为与主节点相关的消息启用快照。

0：禁用。快照针对与从节点相关的消息

1：启用

(R/W)

TSEVNTENA 配置是否仅为事件消息（SYNC，Delay_Req，Pdelay_Req，Pdelay_Resp）启用时间戳快照。

0：禁用。快照针对除 Announce，Management 和 Signaling 之外的所有消息

1：启用

(R/W)

TSIPV4ENA 配置是否启用处理通过 IPv4-UDP 发送的 PTP 帧。

0：禁用。MAC 忽略通过 UDP-IPv4 数据包传输的 PTP

1：启用。MAC 接收器处理封装在 UDP over IPv4 数据包中的 PTP 数据包

(R/W)

TSIPV6ENA 配置是否启用处理通过 IPv6-UDP 发送的 PTP 帧。

0：禁用。MAC 忽略通过 UDP-IPv6 数据包传输的 PTP

1：启用。MAC 接收器处理封装在 UDP over IPv6 数据包中的 PTP 数据包

(R/W)

见下页...

Register 55.50. EMACTSTPCTRL_REG (0x0700)

接上页...

TSIPENA 配置是否启用处理通过以太网帧发送的 PTP。

0：禁用。MAC 忽略以太网数据帧中的 PTP

1：启用。MAC 接收器直接处理封装在以太网帧中的 PTP 数据包

(R/W)

TSVER2ENA 配置是否启用版本 2 格式的 PTP 数据包处理。

0：禁用。使用版本 1 格式处理 PTP 数据包

1：启用。使用 1588 版本 2 格式处理 PTP 数据包

(R/W)

TSCTRLSSR 配置超时数字或二进制回绕。

0：时间戳低位寄存器在次秒寄存器 0x7FFF_FFFF 的值之后回绕。次秒增量必须根据 PTP 参考时钟频率和此位的值正确编程

1：时间戳低位寄存器在 0x3B9A_C9FF 值（即 1 纳秒精度）之后回绕，并增加时间戳（高位）秒数

(R/W)

TSENALL 配置是否为 MAC 接收到的所有帧启用时间戳。

0：禁用

1：启用

(R/W)

TSADDREG 配置是否更新 PTP 块中的 EMACTSTPADDEND_REG 寄存器的内容以进行精细校正。

0：不更新

1：更新

在设置此位之前，此位应为零。(R/W/SC)

TSTRIG 配置是否启用时间戳中断。

0：禁用

1：启用。当系统时间大于目标时间寄存器中的值时，生成时间戳中断

(R/W/SC)

TSUPDT 配置是否使用 EMAC2NDUPT_REG 和 EMACNSUPT_REG 中指定的值更新（加法或减法）系统时间。

0：不更新

1：更新

在更新之前，此位应读为零。

EMACSYSTIMHIGHWORD2ND_REG 不会被更新。(R/W/SC)

TSINIT 配置是否使用 EMAC2NDUPT_REG 和 EMACNSUPT_REG 中指定的值初始化（覆盖）系统时间。

0：不初始化

1：初始化

在更新之前，此位应为零。

EMACSYSTIMHIGHWORD2ND_REG 将被初始化。(R/W/SC)

见下页...

Register 55.50. EMACTSTPCTRL_REG (0x0700)

接上页...

TSCFUPDT 配置用于更新系统时间的方法。

0：粗略方法

1：精细更新方法

(R/W)

TSENA 配置是否为发送和接收帧添加时间戳。

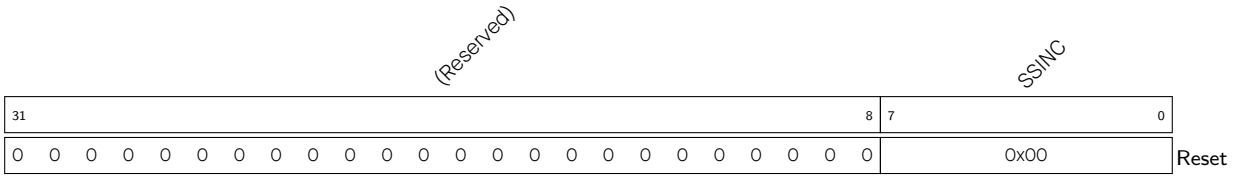
0：不添加时间戳，并且接收帧和时间戳生成器也被挂起

1：添加时间戳

在将此位置为 1 之后，需要初始化时间戳（系统时间）。

在接收端，只有在设置此位时，MAC 才会处理 1588 帧。(R/W)

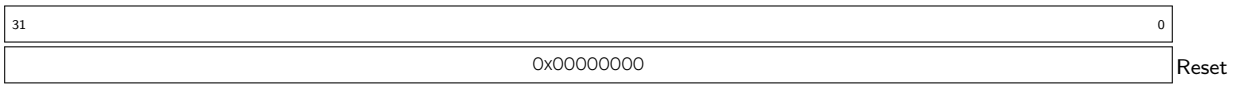
Register 55.51. EMACSUB2NDINCR_REG (0x0704)



SSINC 配置每个时钟周期（PTP 时钟）与次秒寄存器的内容累加的值。

例如，当 PTP 时钟为 50 MHz（周期为 20 ns）时，当系统时间-纳秒寄存器的精度为 1 ns（EMACTSTPCTRL_REG[9]，即 TSCTRLSSR 位置 1）时，应该编程 20 (0x14)。当 TSCTRLSSR 被清除时，纳秒寄存器的分辨率为 0.465 ns。在这种情况下，应该编程一个值为 43 (0x2B)，该值由 20 ns/0.465 得到。(R/W)

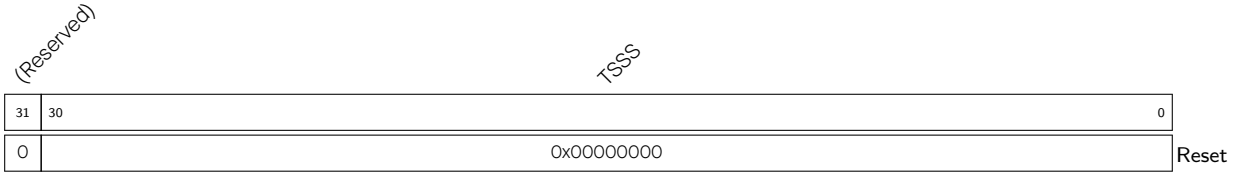
Register 55.52. EMACSYSTIM2ND_REG (0x0708)



TSS 表示 MAC 维护的系统时间的当前值。

单位：秒。(RO)

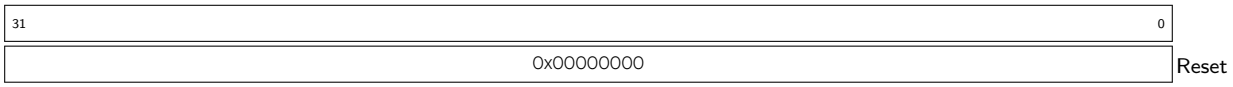
Register 55.53. EMACSYSTIMNS_REG (0x070C)



TSSS 表示次秒时间，精度为 0.46 ns。

当 EMACTSTPCTRL_REG[9] (TSCTRLSSR) 置 1 时，每个位表示 1 ns，最大值为 0x3B9A_C9FF。(RO)

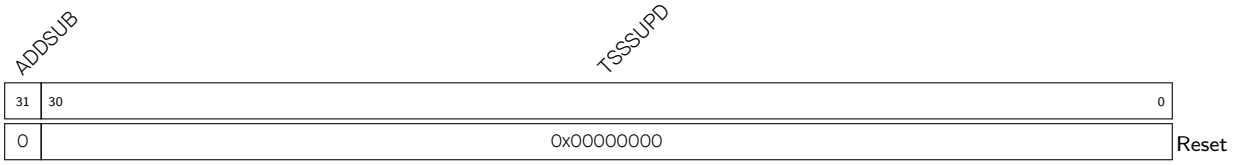
Register 55.54. EMAC2NDUPT_REG (0x0710)



TSSSUPD 配置要初始化或累加到系统时间的时间。

单位：秒。(R/W)

Register 55.55. EMACNSUPT_REG (0x0714)



ADDSUB 配置是否从系统时间值中减去或添加更新寄存器的内容。

0：添加

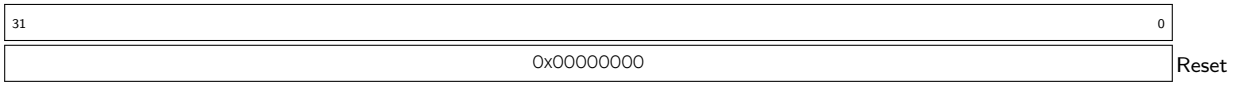
1：减去

(R/W)

TSSSUPD 表示次秒时间，精度为 0.46 ns。

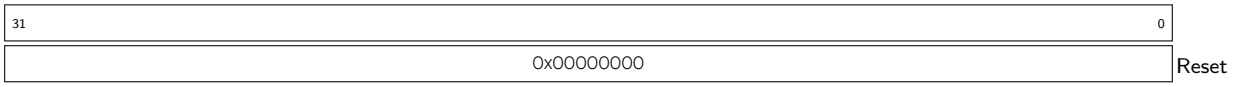
当 EMACTSTPCTRL_REG[9] (TSCTRLSSR) 置 1 时，每个位表示 1 ns，编程值不应超过 0x3B9A_C9FF。(R/W)

Register 55.56. EMACTSTPADDEND_REG (0x0718)



TSAR 配置要添加到累加器寄存器以实现时间同步的 32 位时间值。(R/W)

Register 55.57. EMACTGTIME2ND_REG (0x071C)

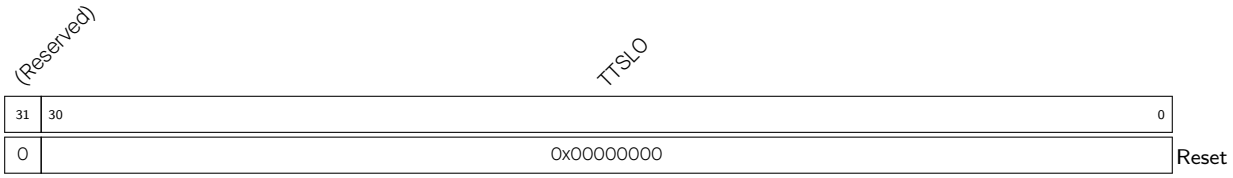


TSTR 配置目标时间。

单位：秒。

当系统时间达到或超过目标时间时，将生成中断（如果已启用）。(R/W)

Register 55.58. EMACTGTIMENS_REG (0x0720)

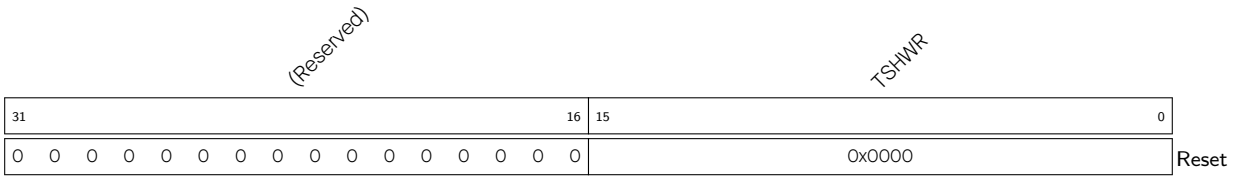


TTSLO 配置有符号目标时间。

单位：纳秒。

当系统时间达到或超过目标时间时，将生成中断（如果已启用）。(R/W)

Register 55.59. EMACSYSTIMHIGHWORD2ND_REG (0x0724)

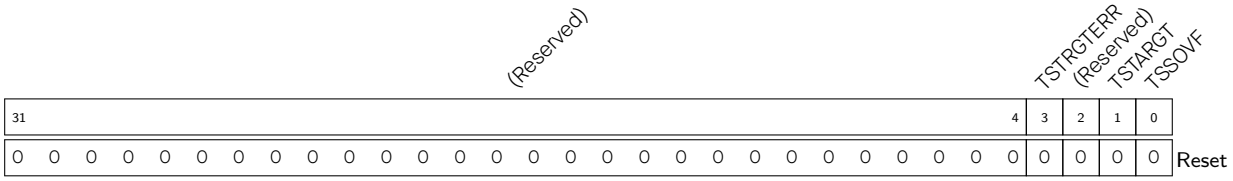


TSHWR 配置时间戳秒值的最高有效 16 位。

直接写入寄存器以初始化该值。

当从系统时间秒寄存器的 32 位发生溢出时，该寄存器会递增。(R/W/SU)

Register 55.60. EMACTSTPSTATUS_REG (0x0728)



TSTRGTERR 表示目标时间是否已经超过了在 EMACTGTIME2ND_REG 和 EMACTGTIMENS_REG 中设定的时间。

0：未超过

1：已超过

(R/SS/RC)

TSTARGT 表示系统时间是否大于或等于在 EMACTGTIME2ND_REG 和 EMACTGTIMENS_REG 中指定的值。

0：小于该值

1：大于或等于该值

(R/SS/RC)

TSSOVF 表示时间戳的第二个值是否已经溢出超过 0xFFFF_FFFF。

0：未溢出

1：溢出

(R/SS/RC)

第 56 章

双线汽车接口 (TWAI)

双线汽车接口 (Two-wire Automotive Interface, TWAI®) 协议是一种多主机、多播的通信协议，具有错误检测、发送错误信号以及内置报文优先仲裁等功能。TWAI 协议适用于汽车和工业应用（参见章节 56.2）。

ESP32-P4 包含三个 TWAI 控制器，即 TWAI 0、TWAI 1 和 TWAI 2，任意控制器都可通过外部收发器连接到 TWAI 总线。TWAI 控制器包含一系列先进的功能，用途广泛，可用于如汽车产品、工业自动化控制、楼宇自动化等。

56.1 主要特性

ESP32-P4 单个 TWAI 控制器具有以下特性：

- 兼容 ISO 11898-1 协议（CAN 规范 2.0）
- 支持标准格式（11 位标识符）和扩展格式（29 位标识符）两种帧格式
- 支持 1 Kbit/s ~ 1 Mbit/s 位速率
- 支持多种操作模式：
  - 正常模式
  - 只听模式（不影响总线）
  - 自测模式（发送数据时无需应答）
- 配置 64 字节接收 FIFO
- 支持特殊发送：
  - 单次发送（发生错误时不会自动重新发送）
  - 自发自收（TWAI 控制器同时发送和接收报文）
- 配置数据接收过滤器（支持单过滤器和双过滤器模式）
- 支持错误检测与处理
  - 配置错误计数器
  - 可配置错误报警阈值
  - 内置错误代码记录
  - 内置仲裁丢失记录
  - 支持收发器自动待机功能

56.2 协议概述

56.2.1 TWAI 特性

TWAI 协议连接网络中的两个或多个节点，并允许各节点以确定性延迟的方式进行报文交互。TWAI 总线具有以下特性：

单通道通信与不归零编码： TWAI 总线只有一根传输线进行单通道通信，为半双工通信。同步调整也在单通道中进行，无需其他通道（如时钟通道和使能通道）。TWAI 上报文的位流采用不归零编码 (NRZ) 方式。

位值： 单通道可处于显性状态或隐性状态，显性状态的逻辑值为 0，隐性状态的逻辑值为 1。发送显性状态数据的节点优先级高于发送隐性状态数据的节点。总线的物理实现（如使用差分对或者单端信号）由其应用决定。

位填充： TWAI 报文的某些域已经过位填充。发送器在发送连续五个位的相同值（如显性数值或隐性数值）后，会自动插入一个互补位。同理，在接收五个相同值的连续位后，接收器应将下一个位视为填充位。位填充应用于以下域：SOF、仲裁域、控制域、数据域和 CRC 序列（参见章节 56.2.2）。

多播： 当各节点连接到同个总线上时，这些节点都将接收到相同的位。如无总线错误（参见章节 56.2.3），各节点上的数据将保持一致。

多主机： 任意节点都可发起数据传输。如果当前已有正在进行的数据传输，则节点将在当前传输结束后再发起新的数据传输。

报文优先级与仲裁： 若两个或多个节点同时发起数据传输，TWAI 协议将确保其中一个节点获得总线的优先仲裁权。优先仲裁权由各节点所发送报文的仲裁域决定。

错误检测： 各节点将积极检测总线上的错误，并向 TWAI 总线发送错误帧来广播检测到的错误。

故障限制： 各节点都配置有错误计数器，计数器数量依据 TWAI 协议规则增加或减少。当错误计数超过一定阈值时，对应节点将自动关闭并退出网络。

可配置位速率： 一个 TWAI 总线的位速率是可配置的。但是，同一总线中的所有节点须以相同位速率工作。

发送器与接收器： 不论何时，任意 TWAI 节点都可作为发送器和接收器。

- 产生报文的节点即为发送器。在总线空闲或该节点失去仲裁优先权之前，该节点将一直作为发送器。请注意，仲裁期间存在多个作为发送器的节点。
- 所有不是发送器的节点都可作为接收器。

56.2.2 TWAI 报文

TWAI 节点使用报文发送数据，并在监测到总线上存在错误时，向其他节点发送错误信号。报文有多种帧类型，不同的帧类型具有不同的帧格式。

TWAI 协议支持以下帧类型：

- 数据帧
- 远程帧
- 错误帧
- 过载帧

- 帧间距

TWAI 协议支持以下帧格式：

- 标准格式 (SFF)，由 11 位标识符组成
- 扩展格式 (EFF)，由 29 位标识符组成

56.2.2.1 数据帧和远程帧

节点使用数据帧向其他节点发送数据，负载可为 0 ~ 8 字节数据。节点使用远程帧向其他节点请求具有相同标识符的数据帧，因此，远程帧中不包含任何数据字节。但是，数据帧和远程帧中包含许多相同域。下图 56.2-1 所示为不同帧类型和不同帧格式中包含的域和子域。

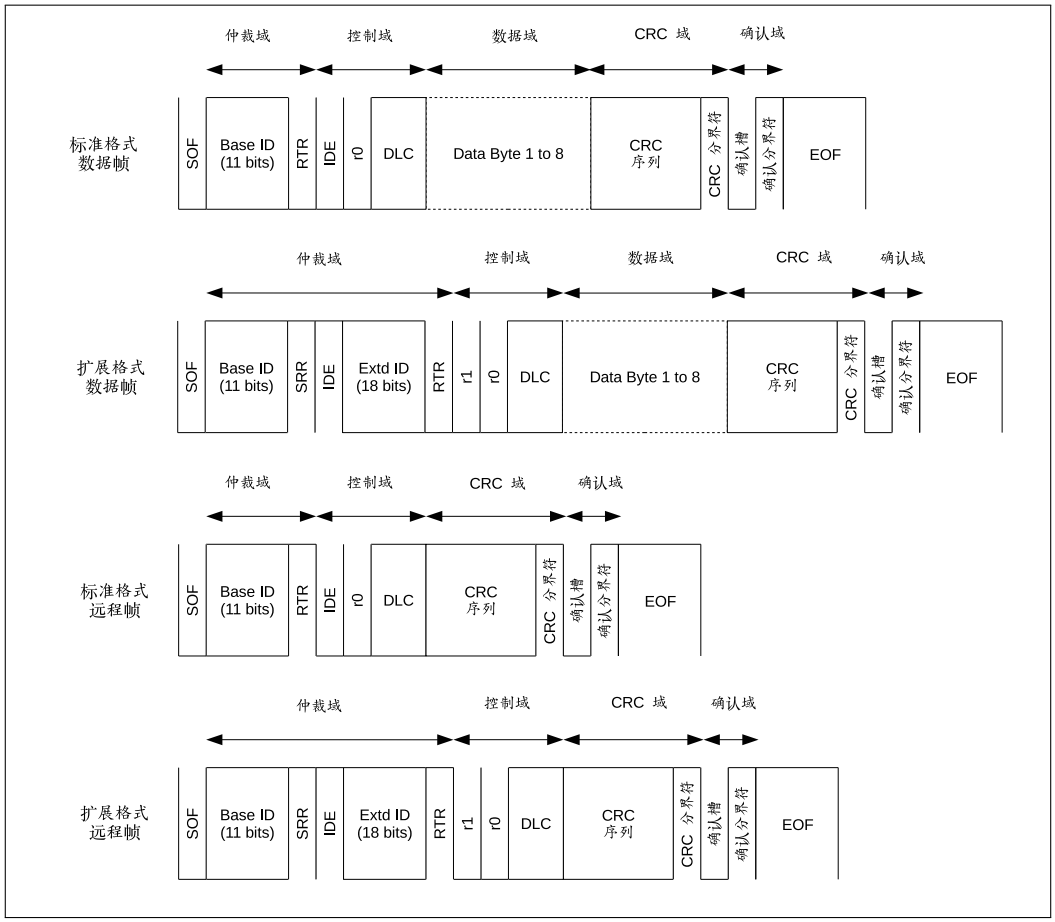


图 56.2-1. 数据帧和远程帧中的位域

仲裁域

当两个或多个节点同时发送数据帧和远程帧时，将根据仲裁域的位信息来决定总线上获得仲裁优先权的节点。在发送仲裁域位信息时，如果一个节点在发送隐性位的同时检测到了一个显性位，表示有其他节点覆盖了这个隐性位。那么，这个发送隐性位的节点将丢失总线仲裁，应立即转为接收器。

仲裁域主要由帧标识符组成，并且由最高位开始发送。根据显性位代表的逻辑值为 0，隐性位代表的逻辑值为 1，有以下规律:

- ID 值最小的帧将总是获得仲裁（逻辑 0 是显性位值）。
- 如果 ID 数值相同，由于数据帧的 RTR 位为显性位，数据帧将优先于远程帧。
- 如果 ID 的前 11 位相同，由于扩展帧的 SRR 位为隐形位，标准格式帧将优先于扩展格式帧。

控制域

控制域主要由数据长度代码 (DLC) 组成。DLC 表示一个数据帧中负载的数据字节长度，或一个远程帧请求的数据字节长度。DLC 从最高位开始发送。

数据域

数据域中包含一个数据帧实际负载的数据字节。远程帧中不包含数据域。

CRC 域

CRC 域主要由 CRC 序列组成。CRC 序列是一个 15 位的循环冗余校验编码，根据数据帧或远程帧中位填充前的内容（从 SOF 到数据域末尾的所有部分）计算而来。

确认域

确认 (ACK) 域由确认槽和确认分界符组成，主要功能在于接收器向发送器报告已正确接收到有效报文。

表 56.2-1. 不同帧类型、帧格式下的域及子域信息

数据/远程帧	描述
SOF	帧起始 (SOF) 是一个用于同步总线上节点的单个显性位。
Base ID	基标识符 (ID.28 ~ ID.18) 是 SFF 的 11 位标识符，或者是 EFF 中 29 位标识符的前 11 位。
RTR	远程发送请求位 (RTR) 显示当前报文是数据帧（显性）还是远程帧（隐性）。这意味着，当某个数据帧和一个远程帧有相同标识符时，数据帧始终优先于远程帧仲裁。
SRR	在 EFF 中发送替代远程请求位 (SRR)，以替代 SFF 中相同位置的 RTR 位。
IDE	标识符扩展位 (IDE) 显示当前报文是 SFF（显性）还是 EFF（隐性）。这意味着，当某 SFF 帧和 EFF 帧有相同基标识符时，SFF 帧将始终优先于 EFF 帧仲裁。
Extd ID	扩展标识符 (ID.17 ~ ID.0) 是 EFF 中 29 位标识符的剩余 18 位。
r1	r1（保留位 1）始终是显性位。
r0	r0（保留位 0）始终是显性位。
DLC	数据长度代码 (DLC) 为 4 位，且为 0 ~ 8 中任一数值。数据帧使用 DLC 表示自身包含的数据字节长度。远程帧使用 DLC 表示从其他节点请求的数据字节长度。
数据字节	表示数据帧的数据负载量。该字节长度应与 DLC 的值匹配。首先发送数据字节的最高有效位 (MSB)。
CRC 序列	CRC 序列是一个 15 位的循环冗余校验编码。
CRC 分界符	CRC 分界符是紧随 CRC 序列的 1 位隐性位。
确认槽	确认槽用于接收器节点表示是否已成功接收数据帧或远程帧。发送器节点将在确认槽中发送一个隐性位，如果接收到的帧没有错误，则接收器节点应发送 1 位显性位覆盖确认槽中的隐性位。
确认分界符	确认分界符是紧随确认槽的 1 位隐性位。
EOF	帧结束 (EOF) 标志着数据帧或远程帧的结束，由 7 个隐性位组成。

56.2.2.2 错误帧和过载帧

错误帧

当某节点检测到总线错误时，将发送一个错误帧。错误帧由特殊的错误标志构成，该标志由某相同值的六个连续位组成，因而违反了位填充的规则。所以，当某节点检测到总线错误并发送错误帧时，其余节点也将相应地

检测到一个填充错误并各自发送错误帧。也就是说，当发生总线错误时，通过上述过程，可将该报文传递至总线上的所有节点。

当某节点检测到总线错误时，该节点将于下一个位发送错误帧。不过，当总线错误类型为 CRC 错误时，错误帧将从确认分界符的下一个位开始（参见章节 56.2.3）。下图 56.2-2 所示为一个错误帧所包含的不同域：

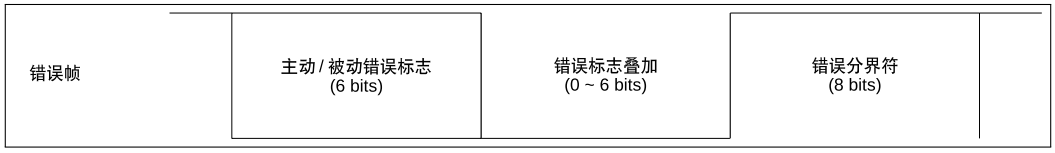


图 56.2-2. 错误帧中的位域

表 56.2-2. 错误帧中的位域信息

错误帧	描述
错误标志	错误标志包括两种形式:主动错误标志和被动错误标志，主动错误标志由 6 个显性位组成，被动错误标志由 6 个隐性位组成（被其他节点的显性位优先仲裁时除外）。处于主动错误状态的节点发送主动错误标志，处于被动错误状态的节点发送被动错误标志。
错误标志叠加	错误标志叠加域的主要目的是允许总线上的其他节点发送各自的主动错误标志。叠加域的范围是 0 ~ 6 位，结束标志是检测到第一个隐性位（如检测到分界符上的第一个位时）。
错误分界符	分界符域标志着错误/过载帧结束，由 8 个隐性位构成。

过载帧

过载帧与包含主动错误标志的错误帧有着相同的位信息。二者区别在于触发发送过载帧的条件。下图 56.2-3 所示为过载帧中包含的位域：

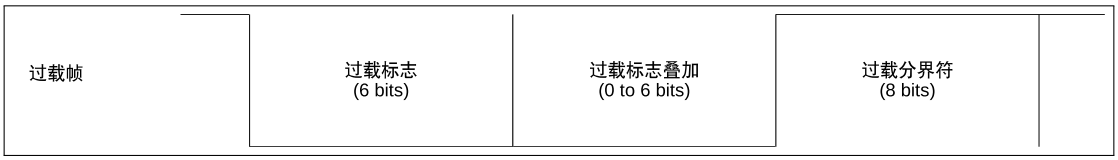


图 56.2-3. 过载帧中的位域

表 56.2-3. 过载帧中的位域信息

过载帧	描述
过载标志	由 6 个显性位构成，与主动错误标志相同。
过载标志叠加	允许其他节点发送过载标志的叠加，与错误标志叠加相似。
过载分界符	由 8 个隐性位构成，与错误分界符相同。

下列情况将触发发送过载帧：

1. 接收器内部要求延迟发送下一个数据帧或远程帧。
2. 在间歇域中的首个和第二个位上检测到显性位。
3. 如果在错误分界符的第八个（最后一个）位上检测到显性位。请注意，在这种情况下，TEC 和 REC 的值不会增加（参见章节 56.2.3）。

上述不同情况触发过载帧时，须满足以下不同的规定：

- 第 1 条情况下，发送的过载帧只能在间歇域的第一个位开始。这种情况最多可生成两个过载帧。
- 第 2、3 条情况下，发送的过载帧须从检测到显性位的后一个位开始。

56.2.2.3 帧间距

帧间距充当各帧之间的分隔符。发送数据帧和远程帧前，必须与前一帧用一个帧间距分隔开，不论前面的帧是何类型（数据帧、远程帧、错误帧、过载帧）。错误帧和过载帧则无需与前一个帧分隔开。

下图 56.2-4 所示为帧间距中包含的域：

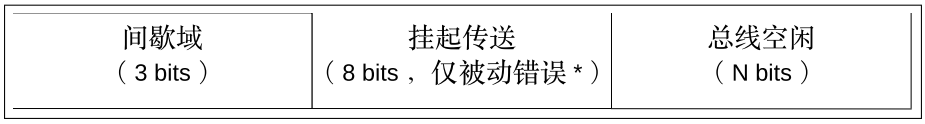


图 56.2-4. 帧间距中的域（* 关于被动错误）

表 56.2-4. 帧间距中的域信息

帧间距	描述
间歇域	间歇域由 3 个隐性位构成。
挂起传送	被动错误节点发送报文后，必须在帧间距中包含一个挂起传送域，由 8 个隐性位构成。主动错误节点中不含这个域。
总线空闲	总线空闲域长度任意。发送 SOF 时，总线空闲结束。若节点中有挂起传送，则 SOF 应在挂起传送域后的第一位发送。

56.2.3 TWAI 错误

56.2.3.1 错误类型

TWAI 中的总线错误包括以下类型：

位错误

当节点发送一个位值（显性位或隐性位）但检测到相反的位时（如发送显性位时检测到了隐性位），就会发生位错误。但是，如果发送的位是隐性位，且位于仲裁域、确认槽或被动错误标志中，那么此时检测到显性位则不会认定为位错误。

填充错误

当检测到相同值的六个连续位时（违反位填充的编码规则），发生填充错误。

CRC 错误

接收器将根据接收到的数据帧和远程帧的有效位（位填充前）计算 CRC 值。当接收器计算的值与接收到的数据帧和远程帧中的 CRC 序列不匹配时，会发生 CRC 错误。

格式错误

当某个报文中的固定格式位中包含非法位时，可检测到格式错误。比如，r1 和 r0 域必须固定为显性。

确认错误

当发送器无法在确认槽中检测到显性位时，会发生确认错误。

56.2.3.2 错误状态

每个节点 TWAI 控制器通过维护两个错误计数器来实现故障界定，计数数值决定错误状态。这两个错误计数器分别为：发送错误计数 (TEC) 和接收错误计数 (REC)。TWAI 包含以下错误状态：

主动错误
处于主动错误状态的节点可参与到总线交互中，且在检测到错误时发送主动错误标志。

被动错误
处于被动错误状态的节点可参与到总线交互中，且在检测到错误时发送被动错误标志。被动错误节点发送数据帧或远程帧后，需在后续的帧间距中增加挂起传送域。

离线
禁止处于离线状态的节点以任意方式干扰总线（如不允许离线节点进行数据传输）。

56.2.3.3 错误状态跳转

1. 当一个节点的 TEC 和/或 REC 数值大于等于 128 时，该节点变为被动错误节点。对于导致节点发生上述状态切换的错误，该节点仍发送主动错误标志。请注意，一旦 REC 数值到达 128，后续任何增加该值的动作都是无效的，直到 REC 数值返回到 128 以下。
2. 当某节点的 TEC 数值大于等于 256 时，该节点进入离线状态。
3. 当某被动错误节点的 TEC 和 REC 数值都小于等于 127 时，该节点变为主动错误节点。
4. 当离线节点在总线上检测到 128 次 11 个连续隐性位后，该节点可变为主动错误节点（TEC 和 REC 数值都重设为 0）。

56.2.3.4 错误计数规则

TEC 和 REC 根据以下规则递增/递减。**请注意，一条报文传输中可应用多个规则**。

TEC 的递增/递减规则：

1. 当发送器发送错误标志时，TEC 数值增加 8。但是，以下情况不适用于该规则：
   - 发送器处于被动错误状态，在检测到应答错误且发送被动错误标志时检测不到显性位时，TEC 数值不应增加。
   - 发送器在仲裁期间因填充错误而发送错误标志，且填充位本该是隐性位但是检测到显性位，TEC 数值不应增加。
2. 若发送器在发送主动错误标志和过载标志时检测到位错误，TEC 数值增加 8。
3. 任意节点在发送主动/被动错误标志或过载标志后，节点仅能承载最多 7 个连续显性位。在发送主动错误标志或过载标志时检测到第 14 个连续显性位，或在被动错误标志后检测到第 8 个连续显性位后，发送器的 TEC 数值增加 8，而接收器的 REC 数值增加 8。每一附加的 8 个连续显性位都会使发送器的 TEC 和接收器的 REC 数值增加 8。
4. 每当发送器成功发送报文后，即接收到 ACK，且直到 EOF 完成都未发生错误时，TEC 数值减小 1，除非 TEC 的数值已经为 0。

REC 的递增/递减规则：

1. 发送错误标志后，当接收器第一个检测到的位是显性位时，REC 数值增加 8。

2. 当接收器检测到错误时，REC 数值增加 1，检测到的错误为发送主动错误标志或过载标志期间的位错误除外。
3. 若接收器在发送主动错误标志和过载标志时检测到位错误，REC 数值增加 8。
4. 当接收器成功接收报文后，即确认槽前未检测到错误，且已成功发送 ACK 时，REC 数值相应减小。
    - 若 REC 数值位于 1 ~ 127 之间，则其值减小 1。
    - 若 REC 数值大于 127，则其值减小到 127。
    - 若 REC 数值为 0，则仍保持为 0。

56.2.4 TWAI 位时序

56.2.4.1 标称位

TWAI 协议允许 TWAI 总线以特定的位速率运行。但是，总线内的所有节点必须以统一位速率运行。

- **标称位速率**为每秒发送比特数量。
- **标称位时间**为 **1/标称位速率**。

每个标称位时间中含多个段，每段由多个时间定额 (Time Quanta) 组成。**时间定额**为最小时间单位，作为一种预分频时钟信号应用于各个节点中。下图 56.2-5 所示为一个标称位时间内所包含的段。

TWAI 控制器将在以一个时间定额的步长内进行操作，每个时间定额中都会分析 TWAI 的总线状态。如果两个连续的时间定额中总线状态不同（隐性-显性，或反之），意味着有边沿产生。PBS1 和 PBS2 的交点将被视为采样点，采样的总线状态即为这个位的数值。

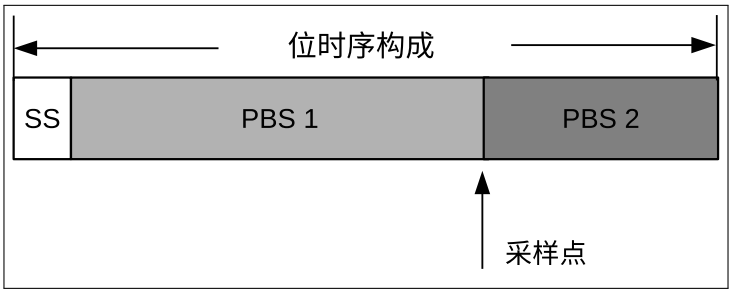


图 56.2-5. 标称位构成

表 56.2-5. 名义位时序中包含的段

段	描述
同步段 (SS)	SS（同步段）的长度为 1 个时间定额。若所有节点都同步，则位边沿发生时应位于该段内。
缓冲时期段 1 (PBS1)	PBS1 的长度可为 1 ~ 16 个时间定额，用于补偿网络中的物理延迟时间。可通过增加 PBS1 的长度来实现同步。
缓冲时期段 2 (PBS2)	PBS2 的长度可为 1 ~ 8 个时间定额，用于补偿节点中的信息处理时间。可通过缩短 PBS2 的长度来实现同步。

56.2.4.2 硬同步与再同步

由于时钟偏移和抖动，同一总线上节点的位时序可能会脱离相位段。因而，位边沿可能会偏移到同步段的前后。针对上述位边沿偏移的问题，TWAI 提供多种同步方式。设发生相位误差时位边沿偏移的时间定额数量为 **"e"**，该值与 SS 相关。

- 主动相位误差 (e > 0)：位边沿位于同步段之后采样点之前，即边沿向后偏移。
- 被动相位误差 (e < 0)：位边沿位于前个位的采样点之后同步段之前，即边沿向前偏移。

为解决相位误差，有两种同步方式，**硬同步**与**再同步**。**硬同步**与**再同步**遵守以下规则：

- 单个位时序中仅可执行一次同步。
- 同步仅可发生在隐性位到显性位的边沿上。

硬同步

总线空闲期间，硬同步发生在隐性位到显性位的变化边沿上（如总线空闲后的第一个 SOF 位）。此时，所有节点都将重启其内部位时序，从而使该变化边沿位于重启位时序的同步段内。

再同步

非总线空闲期间，再同步发生在隐性位到显性位的变化边沿上。如果边沿上有主动相位误差 (e > 0)，则增加 PBS1 段的长度。如果边沿上有被动相位误差 (e < 0)，则减小 PBS2 段的长度。

PBS1/PBS2 具体增加和减小的时间定额取决于相位误差的绝对值，同时也受可配置的同步跳宽 (SJW) 数值限制。

- 当相位误差的绝对值小于等于 SJW 数值时，PBS1/PBS2 将增加/减小 **e** 个时间定额。该过程与硬同步具有相同效果。
- 当相位误差的绝对值大于 SJW 数值时，PBS1/PBS2 将增加/减小与 SJW 相同数值的时间定额。这意味着，在完全解决相位误差之前，可能需要执行多次位的再同步。

56.3 结构概述

TWAI 控制器的主要功能模块如图 56.3-1。

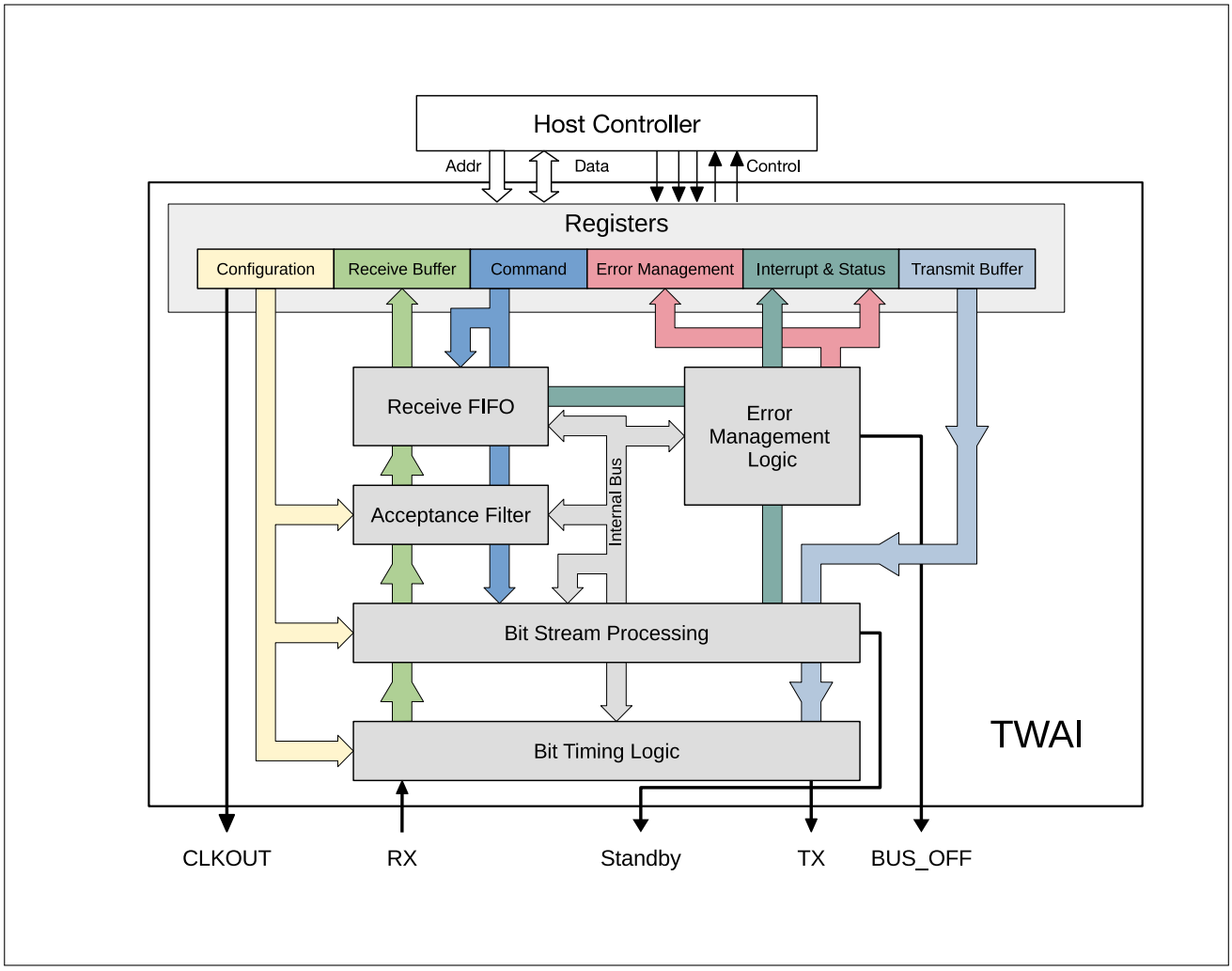


图 56.3-1. TWAI 概略图

56.3.1 位时序逻辑

位时序逻辑 (BTL) 模块负责以预先配置的位速率发送和接受报文。BTL 模块还负责同步位时序，确保数据传输的稳定性。位速率由多个可编程的段组成，且用户可设置每个段的时间定额长度，来调整传播延迟、控制器处理时间等因素。

56.3.2 位流处理器

位流处理 (BSP) 模块负责对发送缓冲器的数据进行帧处理（如位填充和附加 CRC 域），并为位时序逻辑 (BTL) 模块生成位流。同时，BSP 模块还负责处理从 BTL 模块中接收的位流（如去填充和验证 CRC），并将处理后报文写入接收 FIFO。BSP 还负责检测 TWAI 总线上的错误并将此类错误报告给错误管理逻辑 (EML)。

56.3.3 接收过滤器

接收过滤器是一个可编程的报文过滤单元，允许 TWAI 控制器根据报文的标识符域接收或拒绝该报文。通过接收过滤器的报文才能被存储到接收 FIFO 中。用户可配置接收过滤器的模式：单过滤器、双过滤器。

56.3.4 接收 FIFO

接收 FIFO 位于 TWAI 控制器内部，是大小为 64 字节的缓冲器，负责存储通过接收过滤器的接收报文。接收 FIFO 中存储的报文大小可以不同（范围在 3 ~ 13 字节之间）。当接收 FIFO 为满或剩余空间不足以完全存储下当前的接收报文时，将触发溢出中断，后续的接收报文将丢失，直到接收 FIFO 中清除出足够的存储空间。接收 FIFO 中的第一条报文将被映射到 13 字节的接收缓冲器中，直到该报文被清除（通过释放接收缓冲器指令）。清除后，接收 FIFO 将释放上一条已清除报文的空间，同时将窗口映射到接收 FIFO 中的下一条报文。

56.3.5 错误管理逻辑

错误管理逻辑 (EML) 模块负责更新 TEC 和 REC 数值、记录错误信息（如错误类型和错误位置）、更新控制器的错误状态，确保 BSP 模块发送正确的错误标志。此外，该模块还负责记录 TWAI 控制器丢失仲裁时的 bit 位置。

56.3.6 寄存器模块

ESP32-P4 的 CPU 使用 32 位字对齐地址访问外设。但对于 TWAI 控制器中的大部分寄存器，仅最低有效字节 (bits [7:0]) 数据有效。在这些寄存器中，bits [31:8] 数值在写入时被忽略，在读取时返回 0。

配置寄存器

配置寄存器存储 TWAI 控制器的各配置项，如位速率、操作模式、接收过滤器等。只有在 TWAI 控制器处于复位模式时，才可修改配置寄存器（参见章节 56.4.1）。

指令寄存器

CPU 通过指令寄存器驱动 TWAI 控制器执行任务，如发送报文或清除接收缓冲器。只有在 TWAI 控制器处于操作模式时，才可修改指令寄存器（参见章节 56.4.1）。

中断 & 状态寄存器

中断寄存器显示 TWAI 控制器中发生的事件（每个事件由一个单独的位表示）。状态寄存器显示 TWAI 控制器的当前状态。

错误管理寄存器

错误管理寄存器包括错误计数和捕捉寄存器。错误计数寄存器表示 TEC 和 REC 的数值。捕捉寄存器负责记录相关信息，如 TWAI 控制器在何处检测到总线错误或何时丢失仲裁。

发送缓冲寄存器

发送缓冲器大小为 13 字节，用于存储 TWAI 的待发送报文。

接收缓冲寄存器

接收缓冲器大小为 13 字节，用于存储单个报文。接收缓冲器是读取接收 FIFO 的窗口，接收 FIFO 中的第一个报文将被映射到接收缓冲器中。

请注意，发送缓冲寄存器、接收缓冲寄存器和接收过滤配置寄存器共享同一段地址偏移 0x0040 ~ 0x0070。CPU 访问上述地址范围内寄存器遵循以下规则：

- 当 TWAI 控制器处于复位模式时，该地址范围被映射到接收过滤配置寄存器。

- 当 TWAI 控制器处于操作模式时：
  - 对地址范围的所有读取都映射到接收缓冲寄存器中。
  - 对地址范围的所有写入都映射到发送缓冲寄存器中。

56.4 功能描述

56.4.1 模式

ESP32-P4 TWAI 控制器有两种工作模式：复位模式和操作模式。将 TWAI_RESET_MODE 位置 1，进入复位模式；置 0，进入操作模式。

56.4.1.1 复位模式

要修改 TWAI 控制器的各种配置寄存器，需进入复位模式。进入复位模式时，TWAI 控制器彻底与 TWAI 总线断开连接。复位模式下，TWAI 控制器将无法发送任何报文（包括错误信号）。任何正在进行的报文传输将立即被终止。同样的，TWAI 控制器在该模式下也将无法接收任何报文。

56.4.1.2 操作模式

进入操作模式后，TWAI 控制器与总线相连，并且写保护各配置寄存器，以确保控制器的配置在运行期间保持一致。操作模式下，TWAI 控制器可以发送和接收报文（包括错误信号），但具体取决于 TWAI 控制器配置于哪种运行子模式。TWAI 控制器支持以下三种子模式：

- **正常模式：** TWAI 控制器可以发送和接收包含错误信号在内的报文（如，错误帧和过载帧）。
- **自测模式：** 与正常模式相同，但在该模式下，TWAI 控制器发送报文时，即使在 CRC 域之后没有接收到应答信号，也不会产生应答错误。通常在单个 TWAI 控制器自测时使用该模式。
- **只听模式：** TWAI 控制器可以接收报文，但在 TWAI 总线上保持完全被动。因此，TWAI 控制器将无法发送任何报文、应答或错误信号。错误计数寄存器将保持冻结状态。该模式用于 TWAI 总线监控。

请注意，退出复位模式后（如，进入操作模式时），TWAI 控制器需等待 11 个连续隐性位出现，才能完全连接上 TWAI 总线（即，可以发送或接收报文）。

56.4.2 位时序

TWAI 控制器的工作位速率必须在控制器处于复位模式时进行配置，通过寄存器 TWAI_BUS_TIMING_0_REG 和 TWAI_BUS_TIMING_1_REG 配置位速率。

下表 56.4-1 所示为 TWAI_BUS_TIMING_0_REG 包含的位域。TWAI 工作时钟有多个时钟源，用户可以根据需要进行配置，详细配置说明请参见章节 11 复位和时钟。

表 56.4-1. TWAI_BUS_TIMING_0_REG 的位信息 (0x18)

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
保留	SJW.1	SJW.0	BRP.13	BRP.12	.....	BRP.1	BRP.0

说明：

- 预分频值 (BRP)：TWAI 时间定额时钟由 XTAL 时钟（数值可配，默认是 40 MHz）分频得到。可通过以下公式计算分频数值，其中 t_{Tq} 为时间定额的时钟周期，t_{CLK} 为 TWAI 工作时钟周期：

t_{Tq} = 2 × t_{CLK} × (BRP + 1)

- 同步跳宽 (t_{SJW})：t_{SJW} 数值在 SJW[1:0] 中配置，计算公式为：t_{SJW} = (SJW[1:0] + 1)。

下表 56.4-2 所示为 TWAI_BUS_TIMING_1_REG 包含的位域。

表 56.4-2. TWAI_BUS_TIMING_1_REG 的位 (0x1c)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

说明：

- PBS1：根据 PBS1 + 1 计算缓冲时期段 1 中的时间定额数量。
- PBS2：根据 PBS2 + 1 计算缓冲时期段 2 中的时间定额数量。
- SAM：该值置 1 启动三点采样。用于低/中速总线，有利于过滤总线上的尖峰信号。

56.4.3 发送缓冲器与接收缓冲器

56.4.3.1 缓冲器概述

表 56.4-3. SFF 与 EFF 的缓冲器布局

标准格式 (SFF)		扩展格式 (EFF)	
偏移地址	**内容**	**偏移地址**	**内容**
0x40	TX/RX 帧信息	0x40	TX/RX 帧信息
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	保留	0x6c	TX/RX data byte 7
0x70	保留	0x70	TX/RX data byte 8

表 56.4-3 所示为发送缓冲器和接收缓冲器的寄存器布局。发送和接收缓冲寄存器的访问地址范围相同，且只有当 TWAI 控制器处于操作模式时才可访问。CPU 的写入操作将访问发送缓冲寄存器，CPU 的读取操作将访问接收缓冲寄存器。发送缓冲器和接受缓冲器中存储报文（接收报文或待发送报文）的寄存器布局和域完全一致。

发送缓冲寄存器用于配置 TWAI 的待发送报文。CPU 可以在发送缓冲寄存器进行写入操作，指定报文的帧类型、帧格式、帧 ID 和帧数据（有效载荷）。一旦发送缓冲器配置完成后，CPU 可以将 TWAI_CMD_REG 中的 TWAI_TX_REQUEST 位置 1，以开始报文发送。

- 若是自发自收请求，变更为将 TWAI_SELF_RX_REQUEST 置 1。

- 若是单次发送，需要同时将 TWAI_TX_REQUEST 和 TWAI_ABORT_TX 置 1。

接收缓冲寄存器将映射接收 FIFO 中的第一条报文。CPU 可以对接收缓冲寄存器执行读取操作，获取第一条报文的帧类型、帧格式、帧 ID 和帧数据（有效载荷）。读取完接收缓冲寄存器中的报文后，CPU 通过将 TWAI_CMD_REG 中的 TWAI_RELEASE_BUFFER 位置 1 来清除接收缓冲寄存器，若接收 FIFO 中仍有待处理的报文，按照接收报文的先后次序依次将接收到下一个的报文映射到接收缓冲寄存器。

56.4.3.2 帧信息

帧信息的长度为 1-byte，主要用于明确报文的帧类型、帧格式以及数据长度。下表 56.4-4 所示为帧信息域。

表 56.4-4. TX/RX 帧信息 (SFF/EFF)；TWAI 地址 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	FF	RTR	X	X	DLC.3	DLC.2	DLC.1	DLC.0

说明：

1. FF: 报文属于 EFF 还是 SFF。当 FF 位为 1 时，该报文为 EFF，当 FF 位为 0 时，该报文为 SFF。
2. RTR: 报文是数据帧还是远程帧。当 RTR 位为 1 时，该报文为远程帧，当 RTR 位为 0 时，该报文为数据帧。
3. X: 无关位，可以是任意值。
4. DLC: 数据帧中的数据字节数量，或从远程帧中请求的数据字节数量。TWAI 数据帧的最大载荷为 8 个数据字节，因此 DLC 的数值范围应是 0 ~ 8。

56.4.3.3 帧标识符

若报文为 SFF，则对应的帧标识符域占据 2 字节（11 位）；若报文为 EFF，则对应的帧标识符域占据 4 字节（29 位）。

下表 56.4-5 ~ 56.4-6 所示为 SFF（11 位）报文的帧标识符域。

表 56.4-5. TX/RX 标识符 1 (SFF); TWAI 地址 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3

表 56.4-6. TX/RX 标识符 2 (SFF); TWAI 地址 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]	X[2]	X[2]

说明：

1. 无关项。建议设置为与接收缓冲器兼容（设为 RTR），以防需使用自接收功能（或与自接收功能一起使用）。
2. 无关项。建议设置为与接收缓冲器兼容（设为 0），以防需使用自接收功能（或与自接收功能一起使用）。

下表 56.4-7 ~ 56.4-10 所示为 EFF（29 位）报文的帧标识符域。

表 56.4-7. TX/RX 标识符 1 (EFF); TWAI 地址 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

表 56.4-8. TX/RX 标识符 2 (EFF); TWAI 地址 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

表 56.4-9. TX/RX 标识符 3 (EFF); TWAI 地址 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

表 56.4-10. TX/RX 标识符 4 (EFF); TWAI 地址 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

说明：

1. 无关项。建议设置为与接收缓冲器兼容（设为 RTR），以防需使用自接收功能（或与自接收功能一起使用）。
2. 无关项。建议设置为与接收缓冲器兼容（设为 0），以防需使用自接收功能（或与自接收功能一起使用）。

56.4.3.4 帧数据

帧数据域包含发送或接收的数据帧，范围为 0 ~ 8 bytes。其中的有效字节数应与 DLC 相同。但是，如果 DLC 数值大于 8，则帧数据域的有效字节数仍为 8。远程帧中不包含数据载荷，因此不存在帧数据域。

比如，当发送 5 个字节的数据帧时，CPU 应在 DLC 域中写入数值 5，并将数据写入数据域 1 ~ 5 字节对应的寄存器。同样，当接收 DLC 为 5 的数据帧时，只有 1 ~ 5 数据字节中包含 CPU 可以读取的有效载荷数据。

56.4.4 接收 FIFO 和数据溢出

接收 FIFO 是一个 64-byte 的内部缓冲器，用于以先进先出的原则存储接收到的报文。一条接收报文可在接收 FIFO 中占 3 ~ 13 bytes 空间，且其中字节序与接收缓冲器的寄存器地址顺序相同。接收缓冲寄存器将被映射到接收 FIFO 中第一条报文。

当 TWAI 控制器接收到第一条报文时，TWAI_RX_MESSAGE_COUNTER 的值将增至 1，RXI 中断开始有效，第一条报文会自动更新到缓冲器中。后续每当 TWAI 控制器接收到新的报文，TWAI_RX_MESSAGE_COUNTER 的值会增加 1，直至最大值 64，此时如果接收 FIFO 中有足够的剩余空间，报文内容将被写入到接收 FIFO 中。

软件读取接收缓冲器中的消息后，通过将 TWAI_RELEASE_BUFFER 的位置 1，释放接收 FIFO 中此报文所占的空间，TWAI_RX_MESSAGE_COUNTER 的值也将减小 1。然后，接收缓冲器将映射接收 FIFO 中的下一条报文。软件应重复以上操作直至 TWAI_RX_MESSAGE_COUNTER 为 0。此时，RXI 中断停止触发，接收缓冲器中的数据是无效的，TWAI_STATUS_RECEIVE_BUFFER 为 0。

当 TWAI 控制器接收到一条报文，但接收 FIFO 没有足够空间完整地存储这条接收报文时（不论是因为报文内容大小大于接收 FIFO 中的空闲空间，还是因为接收 FIFO 已满），便会发生数据溢出。

数据溢出发生时：

- 接收 FIFO 中剩余的空间将填满溢出报文的内容。如果接收 FIFO 已满，则无法存储溢出报文的任何内容。
- 接收 FIFO 首次发生数据溢出时，将触发数据溢出中断。
- 溢出报文仍将增加 TWAI_RX_MESSAGE_COUNTER 的值，直到 TWAI_RX_MESSAGE_COUNTER 达到最大值 64。
- 接收 FIFO 将在内部将溢出报文标记为无效。可使用 TWAI_STATUS_MISS 位，确认目前接收缓冲器映射的报文是有效报文还是溢出报文。

为了清除接收 FIFO 中的溢出报文，应重复调用 TWAI_RELEASE_BUFFER，直到 TWAI_RX_MESSAGE_COUNTER 为 0。这就要求用户读取接收 FIFO 中的所有有效报文，并清除所有溢出报文。

56.4.5 接收过滤器

接收过滤器允许 TWAI 控制器根据报文 ID 过滤接收报文（甚至可以过滤报文的第一个数据字节和帧类型）。只有通过过滤的报文才能存储到接收 FIFO 中。接收过滤器的使用可以一定程度地减轻 TWAI 控制器的运行负荷（如，可减少使用接收 FIFO 和发生接收中断的次数），因为 TWAI 控制器将只需要操作过滤后的一小部分报文。

接收过滤的配置寄存器和发送/接收缓冲寄存器使用相同的访问地址空间，只有当 TWAI 控制器处于复位模式时，该地址段才被映射到接收过滤器的配置寄存器。

接收过滤器的配置寄存器由 32 位的 Code 值和 32 位的 Mask 值组成。Code 值将指定一种位排列模式，每条过滤报文中的位都必须匹配该模式，才能使该报文通过过滤。Mask 值可屏蔽 Code 值中的某些位（屏蔽位将被设置为“不相关”位）。如图 56.4-1 所示，为了使报文通过过滤，每条过滤报文的 ID 位都必须匹配 Code 值所设模式或者被 Mask 值屏蔽。

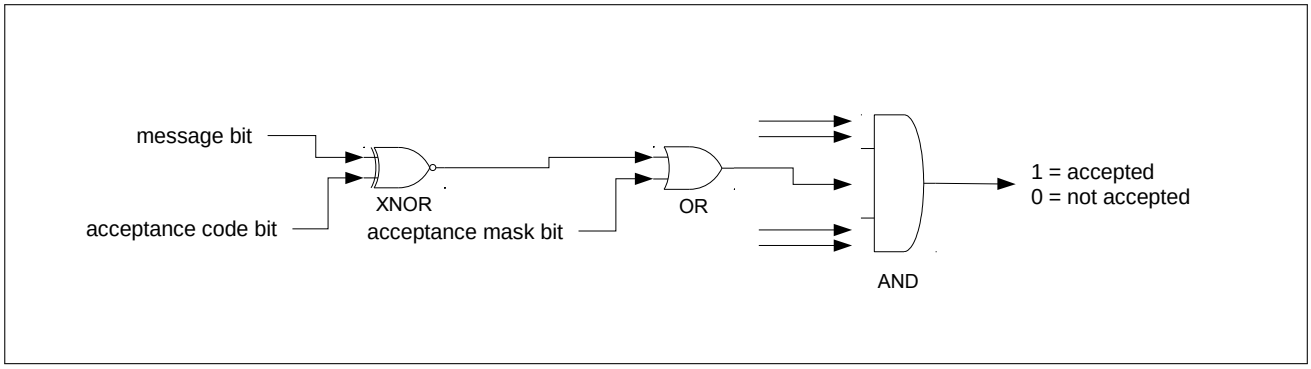


图 56.4-1. 接收过滤器

TWAI 控制器的接收过滤器允许 32 位的 Code 值和 Mask 值定义单个过滤器（单过滤模式），或两个过滤器（双过滤模式）。接收过滤器如何解析 32 位的 Code 值和 Mask 值，取决于过滤模式以及接收报文的格式（如，SFF 还是 EFF）。

56.4.5.1 单过滤模式

将 TWAI_ACCEPTANCE_FILTER_MODE 的位置 1，可启动单过滤模式。此后，32 位 Code/Mask 的值将定义单个过滤器。

单个过滤器可过滤数据帧和远程帧中的以下位：

- SFF
  - 11 位 ID 整体
  - RTR 位
  - 数据字节 1 和数据字节 2
- EFF
  - 29 位 ID 整体
  - RTR 位

下图 56.4-2 所示为单过滤模式下如何解析 32 位 Code/Mask 的值。

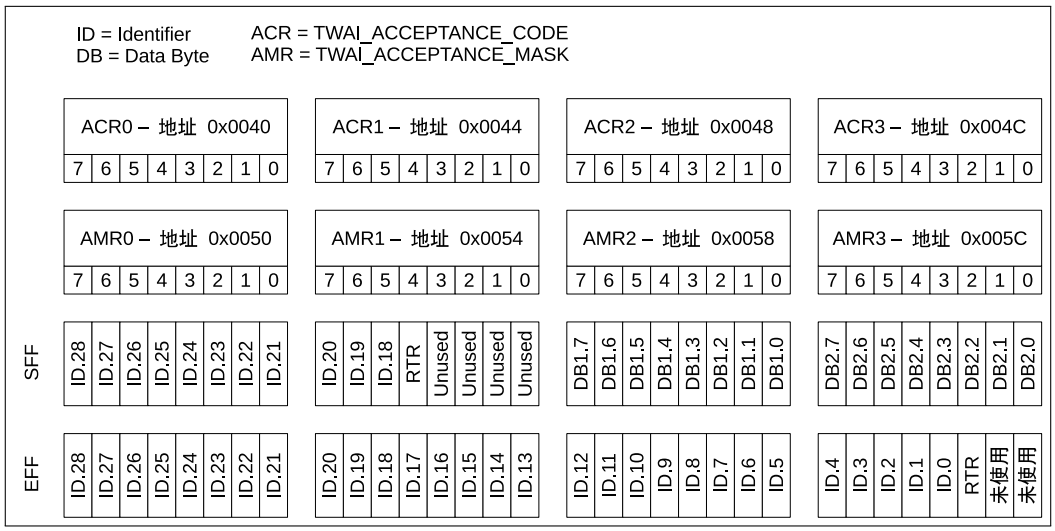


图 56.4-2. 单过滤模式

56.4.5.2 双过滤模式

将 TWAI_ACCEPTANCE_FILTER_MODE 的位置 0，可启动双过滤模式。此后，32 位 Code/Mask 的值将定义两个过滤器之一，即过滤器 1 或过滤器 2。双过滤模式下，如果报文通过这两个过滤器中的至少一个，则表示该报文已成功通过过滤。

这两个过滤器可以过滤数据帧和远程帧中的以下位：

- SFF
  - 11 位 ID 整体
  - RTR 位
  - 数据字节 1（仅适用于过滤器 1）
- EFF
  - 29 位 ID 的前 16 位

下图 56.4-3 所示为双过滤模式下如何解析 32 位 Code/Mask 的值。

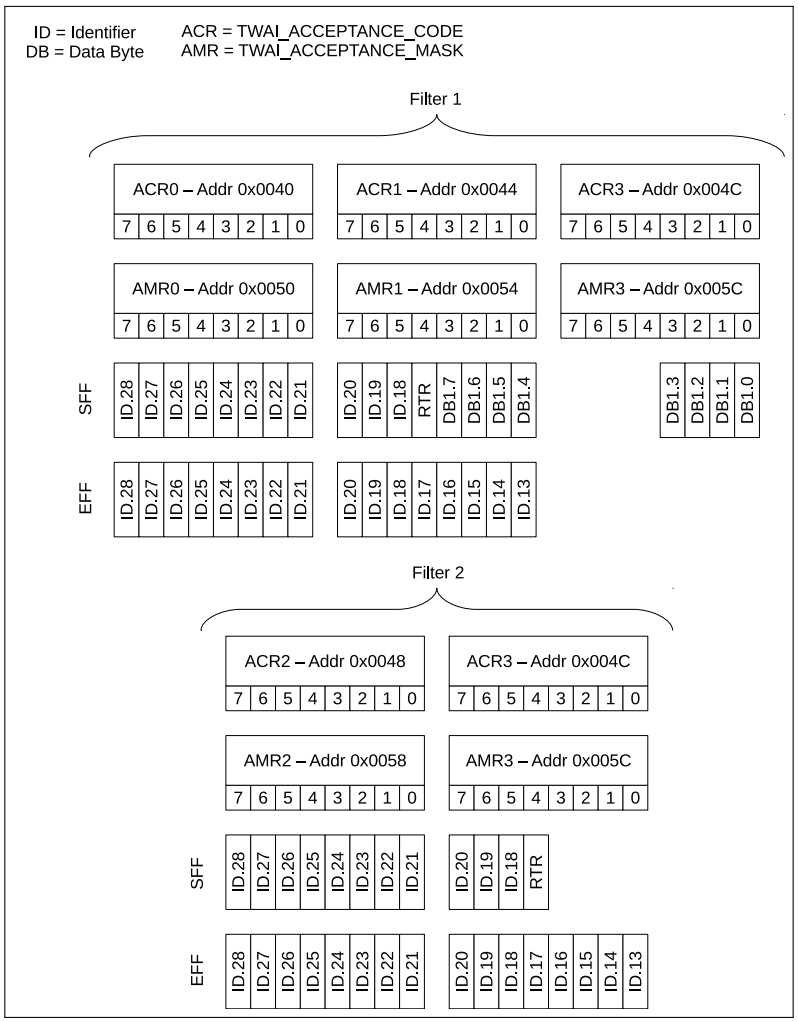


图 56.4-3. 双过滤模式

56.4.6 错误管理

TWAI 协议要求每个 TWAI 节点中都包含发送错误计数器 (TEC) 和接收错误计数器 (REC)。这两个错误计数的数值决定了 TWAI 控制器当前的错误状态（如，主动错误、被动错误、离线）。TWAI 控制器将 TEC 和 REC 的数值分别存储在 TWAI_TX_ERR_CNT_REG 和 TWAI_RX_ERR_CNT_REG 中，CPU 可随时进行读取。除了错误状态之外，TWAI 控制器还提供错误报警限制 (EWL) 的功能，这个功能可在 TWAI 控制器进入被动错误状态之前，提醒用户当前发生的严重总线错误。

TWAI 控制器的当前错误状态通过以下各数值和状态位体现，即：TEC、REC、TWAI_STATUS_ERR 和 TWAI_STATUS_NODE_BUS_OFF。这些数值和状态位的变化也将触发中断，从而提醒用户当前的错误状态变化（可参见第 56.5 章）。下图 56.4-4 所示为错误状态、上述数值和状态位以及错误状态相关中断之间的关系。

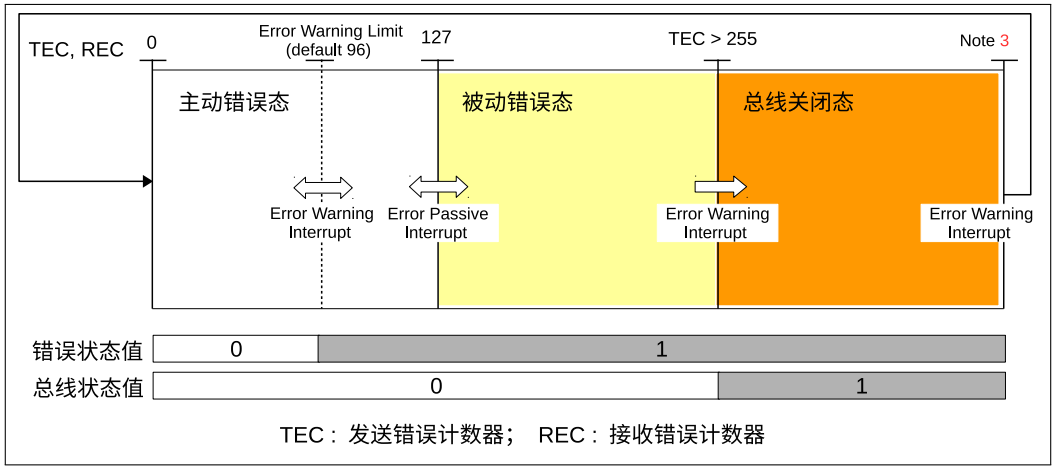


图 56.4-4. 错误状态变化

56.4.6.1 错误报警限制

错误报警限制 (EWL) 为 TEC 和 REC 的可配置阈值，若错误计数数值超过该阈值，将触发 EWI 中断。EWL 将作为一个报警功能提示当前发生的严重 TWAI 总线错误，且在 TWAI 控制器进入被动错误状态之前被触发。EWL 数值应在寄存器 TWAI_ERR_WARNING_LIMIT_REG 中进行配置，配置同时 TWAI 控制器必须处于复位模式下。TWAI_ERR_WARNING_LIMIT_REG 默认数值为 96。

当 TEC 和/或 REC 数值大于等于 EWL 数值时，TWAI_STATUS_ERR 位将立即被置 1。同理，当 TEC 和 REC 数值都小于 EWL 数值时，TWAI_STATUS_ERR 位将立即复位为 0。只要 TWAI_STATUS_ERR（或 TWAI_STATUS_NODE_BUS_OFF）位值发生变化，便会触发错误报警中断。

56.4.6.2 被动错误

当 TEC 或 REC 数值大于 127 时，TWAI 控制器处于被动错误状态。同理，当 TEC 和 REC 数值都小于等于 127 时，TWAI 控制器进入主动错误状态。每当 TWAI 控制器从主动错误状态变为被动错误状态，或反之时，都将触发被动错误中断。

56.4.6.3 离线状态与离线恢复

当 TEC 数值大于 255 时，TWAI 控制器将进入离线状态。进入离线状态后，TWAI 控制器将自动进行以下动作：

- REC 数值置为 0

- TEC 数值置为 127
- TWAI_STATUS_NODE_BUS_OFF 位置 1
- 进入复位模式

每当 TWAI_STATUS_NODE_BUS_OFF 位（或 TWAI_STATUS_ERR 位）数值发生变化时，都将触发错误报警中断。

为了返回主动错误状态，TWAI 控制器必须进行离线恢复。要启动离线恢复，首先需要退出复位模式，进入操作模式。然后要求 TWAI 控制器在总线上检测到 128 次 11 个连续隐性位。

每一次 TWAI 控制器检测到 11 个连续隐性位时，TEC 数值都将减小，以追踪离线恢复进程。当离线恢复完成后（TEC 数值从 127 减小到 0），TWAI_STATUS_NODE_BUS_OFF 位将自动复位为 0，从而触发错误报警中断。

56.4.7 错误捕捉

错误捕捉 (ECC) 功能允许 TWAI 控制器以错误代码的形式记录 TWAI 总线错误的错误类型和 bit 位置。当检测到一个 TWAI 总线错误时，总线错误中断将被触发，相应的错误代码将记录在 TWAI_ERR_CODE_CAP_REG 中。寄存器 TWAI_ERR_CODE_CAP_REG 中存储的当前错误代码被读取之前，后续的总线错误中断触发时，将不会再记录错误代码。

下表 56.4-11 所示为寄存器 TWAI_ERR_CODE_CAP_REG 中的域：

表 56.4-11. TWAI_ERR_CODE_CAP_REG 中的位信息 (0x30)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ERRC.1	ERRC.0	DIR	SEG.4	SEG.3	SEG.2	SEG.1	SEG.0

说明：

- 错误代码 (ERRC)：表示总线错误的类型。00 代表位错误，01 代表格式错误，10 代表填充错误，11 代表其他错误类型。
- 传输方向 (DIR)：表示总线错误发生时，TWAI 控制器处于发送器状态还是接收器状态。0 代表发送器，1 代表接收器。
- 错误段 (SEG)：表示总线错误发生在 TWAI 报文的哪个段。

下表 56.4-12 所示为 SEG.0 ~ SEG.4 的位信息。

表 56.4-12. SEG.4 - SEG.0 的位信息

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	描述
0	0	0	1	1	帧起始
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5
0	1	1	1	0	ID.4 ~ ID.0

见下页

表 56.4-12 – 接上页

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	描述
0	1	1	0	0	bit RTR
0	1	1	0	1	保留位 1
0	1	0	0	1	保留位 0
0	1	0	1	1	数据长度代码
0	1	0	1	0	数据域
0	1	0	0	0	CRC 序列
1	1	0	0	0	CRC 分界符
1	1	0	0	1	确认槽
1	1	0	1	1	确认分界符
1	1	0	1	0	帧结束
1	0	0	1	0	间歇域
1	0	0	0	1	主动错误标志
1	0	1	1	0	被动错误标志
1	0	0	1	1	兼容显性位
1	0	1	1	1	错误分界符
1	1	1	0	0	过载标志

说明：

- Bit SRTR: 标准格式 RTR bit。
- Bit IDE: 标识符扩展位。0 表示标准格式。

56.4.8 仲裁丢失捕捉

仲裁丢失捕捉 (ALC) 功能允许 TWAI 控制器记录丢失仲裁的 bit 位置。当 TWAI 控制器丢失仲裁时，bit 位置将被记录在寄存器 TWAI_ARB_LOST_CAP_REG 中，同时触发仲裁丢失中断。

后续的仲裁丢失中断触发时，bit 位置将不会被记录在 TWAI_ARB_LOST_CAP_REG 中，直到 TWAI_ERR_CODE_CAP_REG 中的当前仲裁丢失捕捉被读取。

下表 56.4-13 所示为 TWAI_ERR_CODE_CAP_REG 中的位域；下图 56.4-5 所示为一条 TWAI 报文的 bit 位置。

表 56.4-13. TWAI_ARB LOST CAP_REG 中的位信息 (0x2c)

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	BITNO.4	BITNO.3	BITNO.2	BITNO.1	BITNO.0

说明：

- 位号 (BITNO)：表示丢失仲裁的 TWAI 报文的第 n 个位。

56.4.9 收发器自动待机

为降低功耗，TWAI 收发器通常支持自动待机模式。TWAI 控制器可以在适当的时候（例如，当总线将长时间空闲时）产生待机信号，将收发器置于待机状态。如果控制器停止发出待机信号，或在收发器检测到总线活动（也称为唤醒功能）时，收发器将退出待机模式。

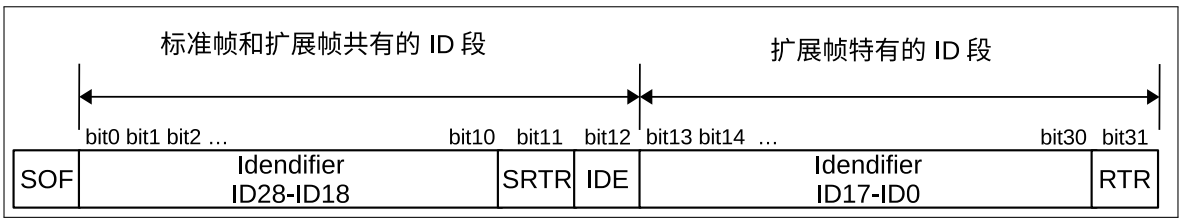


图 56.4-5. 丢失仲裁的 bit 位置

ESP32-P4 的 TWAI 控制器支持通过硬件（自动）和软件（手动）两种方式输出待机信号，来控制芯片外部连接的 TWAI 收发器的开关。在硬件控制下，当总线保持空闲的时间超过可配置的时间时，TWAI 控制器将自动输出待机信号。在软件控制下，待机信号可由软件直接手动产生/关闭。

- 硬件方式：
  1. 通过将 TWAI_HW_CFG_REG 寄存器中的 TWAI_HW_STANDBY_EN 域置位，使能硬件待机功能。
  2. 配置 TWAI_HW_STANDB_CNT_REG 寄存器。该寄存器表明 TWAI 控制器进入空闲状态后多久会产生待机信号，数值为 TWAI 控制器工作时钟（默认为 40 MHz）的周期数。
- 软件方式：
  1. 直接置位 TWAI_SW_STANDBY_CFG_REG 寄存器中的 TWAI_SW_STANDBY_EN 域，便可使 TWAI 控制器产生待机信号。

上述两种方式产生的待机信号，在满足以下任一条件后，会被拉低（清零）。

1. TWAI 控制器接收端退出空闲状态时（例如接收端收到新报文），待机信号将被自动清零。
2. 用户通过置位 TWAI_SW_STANDBY_CFG_REG 寄存器中的 TWAI_SW_STANDBY_CLR 域，也可以将待机信号拉低。

56.5 中断

ESP32-P4 的 TWAI 可以生成以下中断信号并发送给 中断矩阵。

- TWAI_INT

以上中断信号由 TWAI 的内部中断源生成。表 56.5-1 列出了 TWAI 的内部中断源、中断触发条件、以及生成的中断信号。

表 56.5-1. TWAI 的内部中断源

内部中断源	触发条件	中断信号
TWAI_RECEIVE_INT (RXI)	当 TWAI 接收 FIFO 中有待读取报文时（TWAI_RX_MESSAGE_COUNTER_REG > 0），都会触发接收中断 (RXI)。TWAI_RX_MESSAGE_COUNTER_REG 中记录的报文数量包括接收 FIFO 中的有效报文和溢出报文。直到通过 TWAI_RELEASE_BUFFER 指令位清除所有挂起接收报文后，RXI 才会失效。	TWAI_INT

见下页

表 56.5-1 – 接上页

内部中断源	触发条件	中断信号
TWAI_TRANSMIT_INT (TXI)	当发送缓冲器由锁定状态变为为空闲状态，将会触发发送中断 (TXI)。此时软件可以将其他报文加载到发送缓冲器中等待发送。以下几种情况都会触发该中断： • 报文发送已成功完成（如应答未发现错误）。任何发送失败将自动重发。 • 单次发送已完成（TWAI_STATUS_TRANSMISSION_COMPLETE 位指示发送成功与否）。 • 使用 TWAI_ABORT_TX 指令位终止报文发送。	TWAI_INT
TWAI_ERR_WARNING_INT (EWI)	每当寄存器 TWAI_STATUS_REG 中 TWAI_STATUS_ERR 和 TWAI_STATUS_NODE_BUS_OFF 的位值改变时（如，从 0 变为 1 或反之），都会触发错误报警中断 (EWI)。根据 EWI 触发时 TWAI_STATUS_ERR 和 TWAI_STATUS_NODE_BUS_OFF 的值分成以下几种情况： • 如果 TWAI_STATUS_ERR = 0 且 TWAI_STATUS_NODE_BUS_OFF = 0： – 如果 TWAI 控制器处于主动错误状态，则表示 TEC 和 REC 的值都返回到了 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值之下。 – 如果 TWAI 控制器此前正处于总线恢复状态，则表示此时总线恢复已成功完成。 • 如果 TWAI_STATUS_ERR = 1 且 TWAI_STATUS_NODE_BUS_OFF = 0：表示 TEC 或 REC 数值已超过 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值。 • 如果 TWAI_STATUS_ERR = 1 且 TWAI_STATUS_NODE_BUS_OFF = 1：表示 TWAI 控制器已进入 BUS_OFF 状态（因 TEC >= 256）。 • 如果 TWAI_STATUS_ERR = 0 且 TWAI_STATUS_NODE_BUS_OFF = 1：表示 BUS_OFF 恢复期间，TWAI 控制器的 TEC 数值已低于 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值。	TWAI_INT

见下页

表 56.5-1 – 接上页

内部中断源	触发条件	中断信号
TWAI_DATA_OVERRUN_INT (DOI)	每当接收 FIFO 中有溢出发生时，都会触发数据溢出中断 (DOI)。DOI 表示接收 FIFO 已满且应立即进行读取，以防出现更多溢出报文。只有导致接收 FIFO 溢出的第一条报文可触发 DOI（如，当接收 FIFO 从未满变为开始溢出时）。任意后续的溢出报文将不会再次重复触发 DOI。只有当所有接收的（有效报文或溢出）报文都被读取后，才能再次触发 DOI。	TWAI_INT
TWAI_TS_COUNTER_OVFL_INT	每当 TWAI 控制器时间戳 32-bit 计数器溢出时，触发时间戳溢出中断。	TWAI_INT
TWAI_ERR_PASSIVE_INT (EPI)	每当 TWAI 控制器从主动错误变为被动错误，或反之时，都会触发被动错误中断 (EPI)。	TWAI_INT
TWAI_ARBITRATION_LOST_INT (ALI)	每当 TWAI 控制器尝试发送报文且丢失仲裁时，都会触发仲裁丢失中断 (ALI)。TWAI 控制器丢失仲裁的 bit 位置将自动记录在仲裁丢失捕捉寄存器 (TWAI_ARB_LOST_CAP_REG) 中。仲裁丢失捕捉寄存器被清除（通过 CPU 读取该寄存器）之前，将不会再记录新发生的仲裁失败时的 bit 位置。	TWAI_INT
TWAI_BUS_ERR_INT (BEI)	每当 TWAI 控制器在 TWAI 总线上检测到错误时，都会触发总线错误中断 (BEI)。发生总线错误时，总线错误的类型和发生错误时的 bit 位置都将自动记录在错误捕捉寄存器 (TWAI_ERR_CODE_CAP_REG) 中。错误捕捉寄存器被清除（通过 CPU 的读取）之前，将不会再记录新的总线错误信息。	TWAI_INT
TWAI_IDLE_INT (BISI)	当 TWAI 控制器进入空闲状态的时钟周期数超过预先设配置的 TWAI_IDLE_INTR_CNT_REG 寄存器内的数值时，将产生总线空闲状态中断 (BISI)。用户可以通过配置该中断来获知 TWAI 控制器空闲状态，进一步决定是否关闭外部 TWAI 接收器来降低整体功耗（参见56.4.9）。	TWAI_INT

只要在 TWAI_INTERRUPT_REG 一个或多个中断位为 1，TWAI 控制器中的中断信号即为有效，当 TWAI_INTERRUPT_REG 中的所有位都被清除时，TWAI 控制器中的中断信号则失效。读操作访问寄存器 TWAI_INTERRUPT_REG 后，其中的大多数中断位将自动清除。但是，只有通过 TWAI_RELEASE_BUFFER 指令位清除所有接收报文后，接收中断位才能被清除。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 56.6 寄存器列表。

56.6 寄存器列表

请注意，“访问权限”一栏中，“|”用于区分第 56.4.1 中描述的工作模式，其中左侧为操作模式下的访问权限，右侧标红字体为复位模式下的访问权限。本小节的所有地址均为相对于双线汽车接口基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
TWAI_MODE_REG	模式寄存器	0x0000	R/W
TWAI_CMD_REG	指令寄存器	0x0004	WO
TWAI_BUS_TIMING_0_REG	时序配置寄存器 0	0x0018	R/W
TWAI_BUS_TIMING_1_REG	时序配置寄存器 1	0x001C	R/W
TWAI_ERR_WARNING_LIMIT_REG	错误阈值配置寄存器	0x0034	R/W
TWAI_CLOCK_DIVIDER_REG	时钟配置寄存器	0x007C	R/W
TWAI_SW_STANDBY_CFG_REG	软件 standby 寄存器	0x0080	R/W
TWAI_HW_CFG_REG	硬件 standby 寄存器	0x0084	R/W
TWAI_HW_STANDBY_CNT_REG	standby 计数长度寄存器	0x0088	R/W
TWAI_IDLE_INTR_CNT_REG	空闲状态时长寄存器	0x008C	R/W
状态寄存器			
TWAI_STATUS_REG	状态寄存器	0x0008	RO
TWAI_ARB_LOST_CAP_REG	仲裁丢失寄存器	0x002C	RO
TWAI_ERR_CODE_CAP_REG	错误捕获寄存器	0x0030	RO
TWAI_RX_ERR_CNT_REG	接收错误寄存器	0x0038	R/W
TWAI_TX_ERR_CNT_REG	发送错误寄存器	0x003C	R/W
TWAI_RX_MESSAGE_COUNTER_REG	接收计数寄存器	0x0074	RO
中断寄存器			
TWAI_INTERRUPT_REG	中断屏蔽寄存器	0x000C	RO
TWAI_INTERRUPT_ENABLE_REG	中断使能寄存器	0x0010	varies
数据寄存器			
TWAI_DATA_0_REG	数据寄存器 0	0x0040	R/W
TWAI_DATA_1_REG	数据寄存器 1	0x0044	R/W
TWAI_DATA_2_REG	数据寄存器 2	0x0048	R/W
TWAI_DATA_3_REG	数据寄存器 3	0x004C	R/W
TWAI_DATA_4_REG	数据寄存器 4	0x0050	R/W
TWAI_DATA_5_REG	数据寄存器 5	0x0054	R/W
TWAI_DATA_6_REG	数据寄存器 6	0x0058	R/W
TWAI_DATA_7_REG	数据寄存器 7	0x005C	R/W
TWAI_DATA_8_REG	数据寄存器 8	0x0060	R/W | R/O
TWAI_DATA_9_REG	数据寄存器 9	0x0064	R/W | R/O
TWAI_DATA_10_REG	数据寄存器 10	0x0068	R/W | R/O
TWAI_DATA_11_REG	数据寄存器 11	0x006C	R/W | R/O
TWAI_DATA_12_REG	数据寄存器 12	0x0070	R/W | R/O
时间戳寄存器			

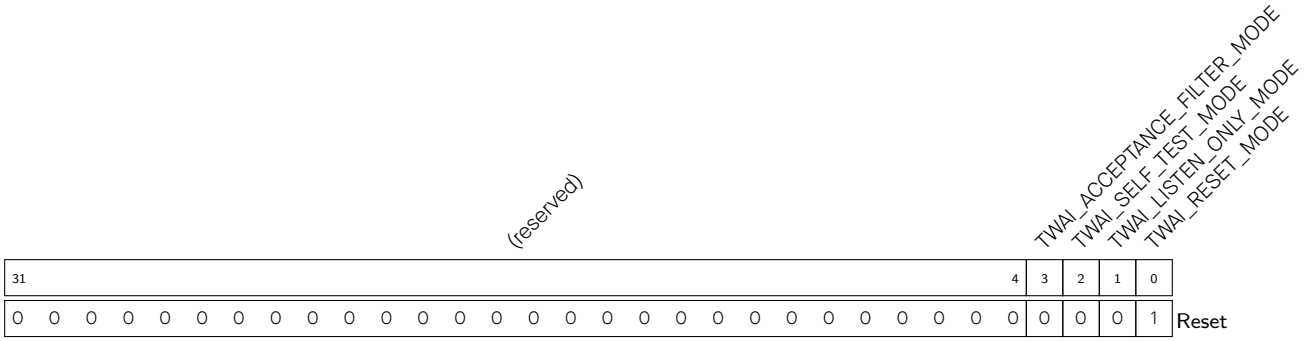
名称	描述	地址	访问
TWAI_TIMESTAMP_DATA_REG	时间戳数据寄存器	0x0094	RO
TWAI_TIMESTAMP_PRESCALER_REG	时间戳配置寄存器 0	0x0098	R/W
TWAI_TIMESTAMP_CFG_REG	时间戳配置寄存器 1	0x009C	R/W

56.7 寄存器

请注意“访问权限”一栏中，“|”左侧为操作模式下的访问权限，右侧标红字体为复位模式下的访问权限。本小节的所有地址均为相对于双线汽车接口基地址的地址偏移量（相对地址）。其中，TWAI 0 和 TWAI 1 都具有各自的基地址。具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 56.1. TWAI_MODE_REG (0x0000)



TWAI_RESET_MODE 配置 TWAI 控制器操作模式。

0：操作模式

1：复位模式

(R/W)

TWAI_LISTEN_ONLY_MODE 配置只听模式。

0：无效

1：只听模式。在该模式下，节点只接收总线上数据，不产生应答信号，也不更新接收错误计数。

(R/W)

TWAI_SELF_TEST_MODE 配置自测模式。

0：无效

1：自测模式。在该模式下，发送节点发送完数据后无需应答信号反馈。该模式常配合自接自收指令测试某个节点。

(R/W)

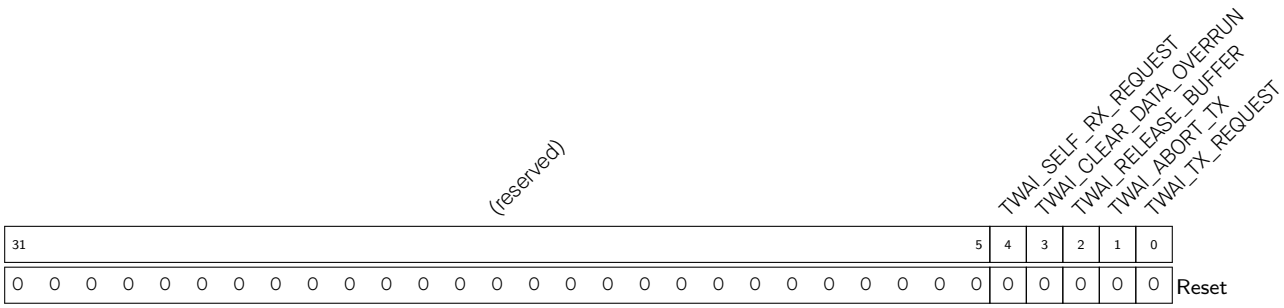
TWAI_ACCEPTANCE_FILTER_MODE 配置过滤模式。

0：双过滤模式

1：单过滤模式

(R/W)

Register 56.2. TWAI_CMD_REG (0x0004)



TWAI_TX_REQUEST 配置是否驱动节点开始发送数据任务。

0：无效

1：驱动节点开始发送数据任务

(WO)

TWAI_ABORT_TX 配置是否取消当前还未开始的发送任务。

0：无效

1：取消当前还未开始的发送任务

(WO)

TWAI_RELEASE_BUFFER 配置是否释放接收缓冲器。

0：无效

1：释放接收缓冲器

(WO)

TWAI_CLEAR_DATA_OVERRUN 配置是否清除数据溢出状态。

0：无效

1：清除数据溢出状态

(WO)

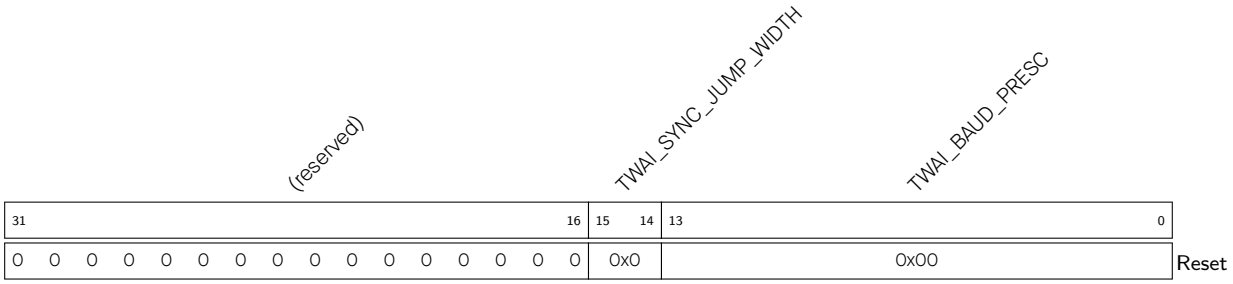
TWAI_SELF_RX_REQUEST 配置是否驱动节点发送数据的同时接收总线上的数据。

0：无效

1：驱动节点开始发送数据的同时接收总线上的数据（需要同时配置 TWAI_TX_REQUEST 为 0，否则只进行发送数据任务）

(WO)

Register 56.3. TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC 配置预分频值，决定分频比例。

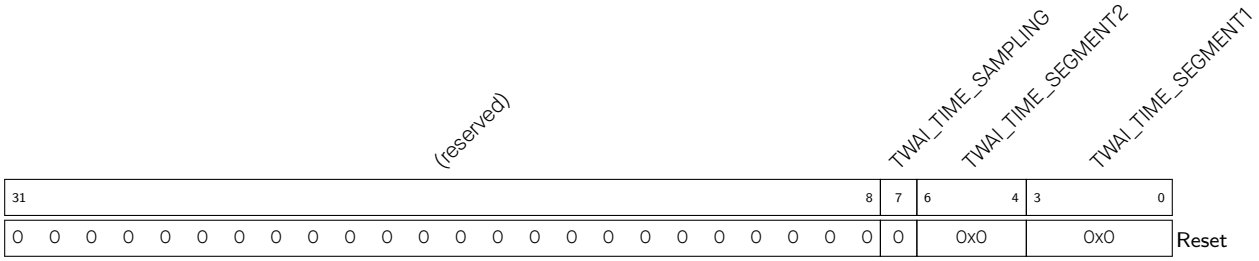
0：低比例

1：高比例

(R/W)

TWAI_SYNC_JUMP_WIDTH 配置同步跳宽，范围为 1 ~ 4 个时间定额。(R/W)

Register 56.4. TWAI_BUS_TIMING_1_REG (0x001C)



TWAI_TIME_SEGMENT1 配置缓冲时期段 1 的宽度。(R/W)

TWAI_TIME_SEGMENT2 配置缓冲时期段 2 的宽度。(R/W)

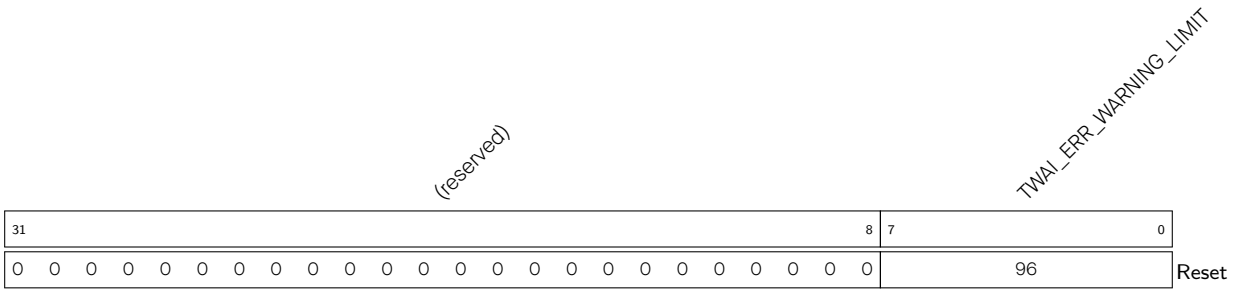
TWAI_TIME_SAMPLING 配置采样点数目。

0：采样一次

1：采样三次

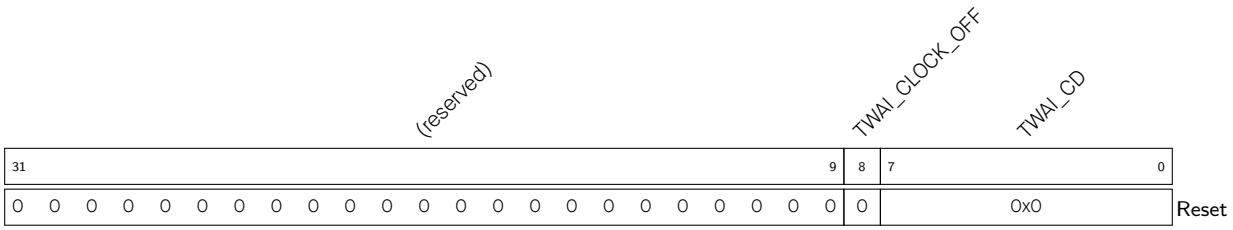
(R/W)

Register 56.5. TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT 配置错误报警阈值。当任一错误计数数值（TWAI_RX_ERR_CNT 或 TWAI_TX_ERR_CNT）超过该阈值时，TWAI 控制器将进入错误状态；当所有错误计数数值都小于该阈值时，TWAI 控制器将解除错误状态。TWAI 控制器进入或解除错误状态将触发一次错误报警中断。(R/W)

Register 56.6. TWAI_CLOCK_DIVIDER_REG (0x007C)



TWAI_CD 配置输出时钟 CLKOUT 的分频系数。(R/W)

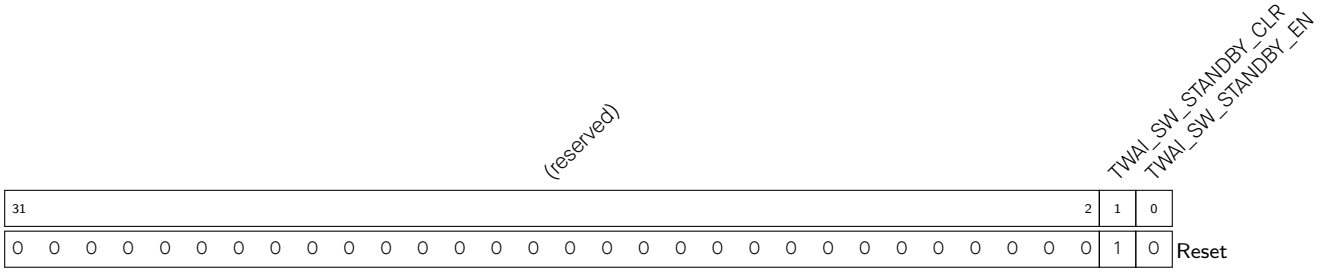
TWAI_CLOCK_OFF 复位模式下，配置是否打开 CLKOUT 时钟。

0：打开 CLKOUT 时钟

1：关闭 CLKOUT 时钟

(R/W)

Register 56.7. TWAI_SW_STANDBY_CFG_REG (0x0080)



TWAI_SW_STANDBY_EN 配置是否软件置位待机信号。

0：无效

1：置位待机信号

(R/W)

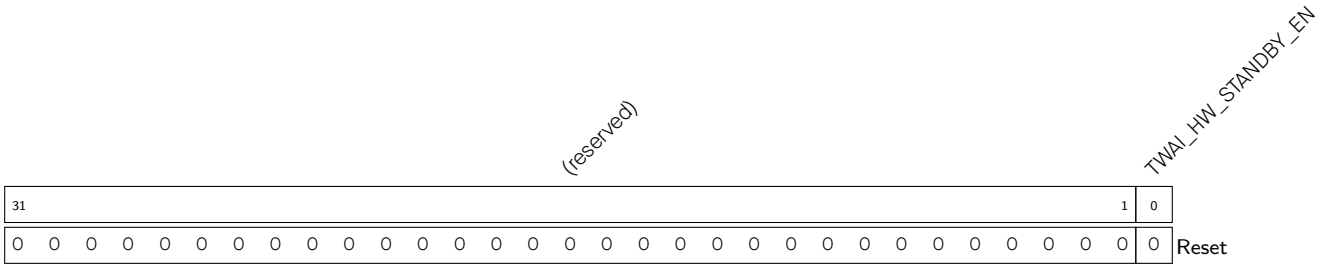
TWAI_SW_STANDBY_CLR 配置是否软件清除待机信号。

0：无效

1：清除待机信号

(R/W)

Register 56.8. TWAI_HW_CFG_REG (0x0084)



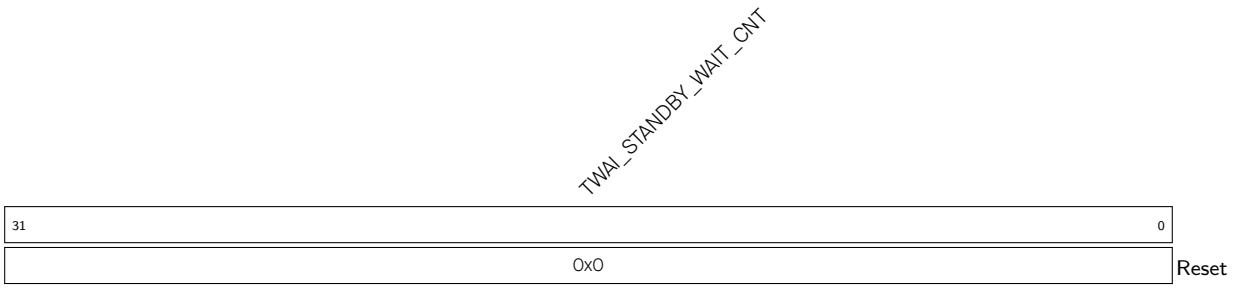
TWAI_HW_STANDBY_EN 配置是否使能硬件自动待机功能。

0：无效

1：使能硬件自动待机功能

(R/W)

Register 56.9. TWAI_HW_STANDBY_CNT_REG (0x0070)

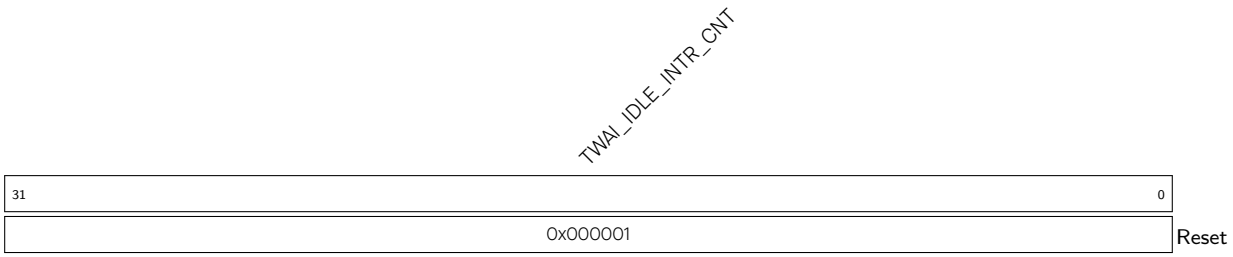


TWAI_STANDBY_WAIT_CNT 配置空闲状态下硬件触发待机信号的时间。

单位：TWAI 控制器时钟周期数。

(R/W)

Register 56.10. TWAI_IDLE_INTR_CNT_REG (0x008C)

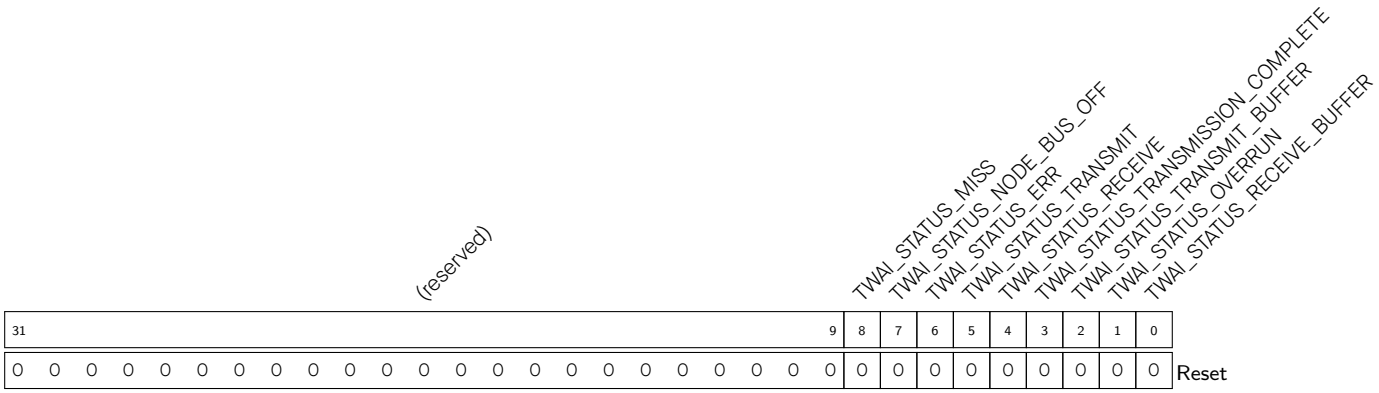


TWAI_IDLE_INTR_CNT 配置空闲状态下硬件产生总线空闲中断信号的时间。

单位：TWAI 控制器时钟周期数。

(R/W)

Register 56.11. TWAI_STATUS_REG (0x0008)



TWAI_STATUS_RECEIVE_BUFFER 表示接收缓冲器中数据是否为空。

0：为空

1：不为空，至少有一个已经接收到的数据包。

(RO)

TWAI_STATUS_OVERRUN 表示接收 FIFO 中存储的数据是否已满。

0：未满

1：已满，产生了溢出。

(RO)

TWAI_STATUS_TRANSMIT_BUFFER 表示发送缓冲器是否为空。

0：不为空

1：为空，允许写入待发送数据。

(RO)

TWAI_STATUS_TRANSMISSION_COMPLETE 表示是否已经往总线上发送一个完整的数据包。

0：未发送

1：已发送

(RO)

TWAI_STATUS_RECEIVE 表示节点是否正从总线上接收数据。

0：未接收

1：正在接收

(RO)

TWAI_STATUS_TRANSMIT 表示节点是否正在往总线上发送数据。

0：未接收

1：正在接收

(RO)

TWAI_STATUS_ERR 表示接收错误计数和发送错误计数中至少有一个数值大于等于寄存器 TWAI_ERR_WARNING_LIMIT_REG 中配置的数值。(RO)

TWAI_STATUS_NODE_BUS_OFF 表示节点是否处于离线状态。

0：非离线状态

1：离线状态，不再响应总线上的数据传输。

(RO)

见下页...

Register 56.11. LP_UART_CONF0_SYNC_REG (0x0020)

接上页...

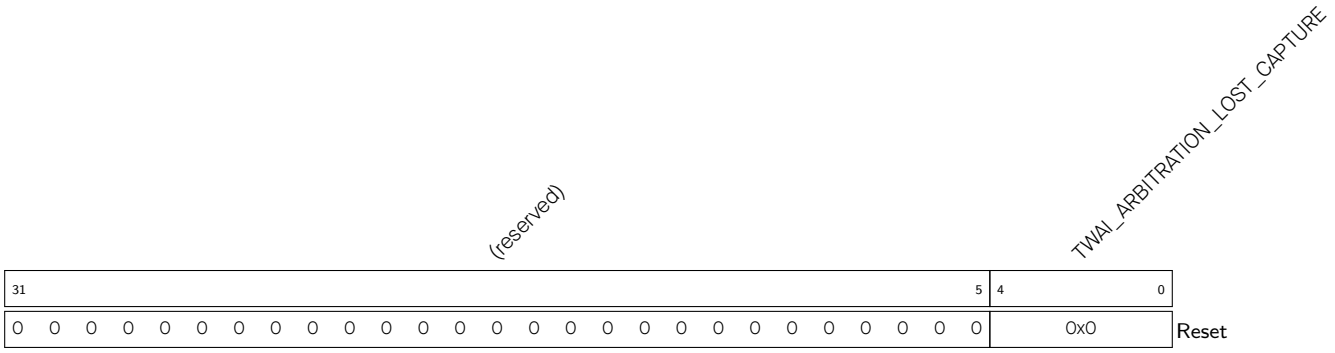
TWAI_STATUS_MISS 表示从接收 FIFO 中取出数据包的完整状态。

0：当前数据包是完整的。

1：当前数据包是缺失的。

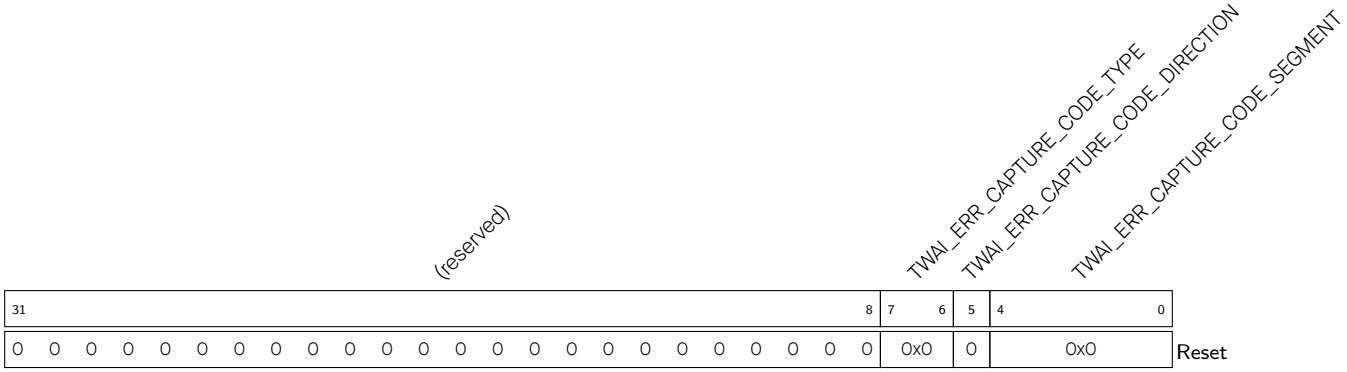
(RO)

Register 56.12. TWAI_ARB_LOST_CAP_REG (0x002C)



TWAI_ARBITRATION_LOST_CAPTURE 表示发送节点仲裁丢失的 bit 位置。(RO)

Register 56.13. TWAI_ERR_CODE_CAP_REG (0x0030)



TWAI_ERR_CAPTURE_CODE_SEGMENT 表示错误发生的位置，详见表 56.4-11。(RO)

TWAI_ERR_CAPTURE_CODE_DIRECTION 表示错误时节点的数据传输方向。

0：发送数据时发生错误。

1：接收数据时发生错误。

(RO)

TWAI_ERR_CAPTURE_CODE_TYPE 表示错误类别。

0：位错误

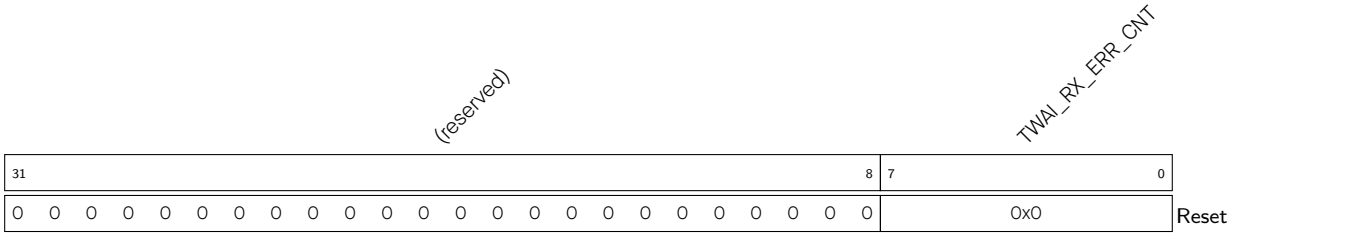
1：格式错误

2：填充错误

3：其他错误

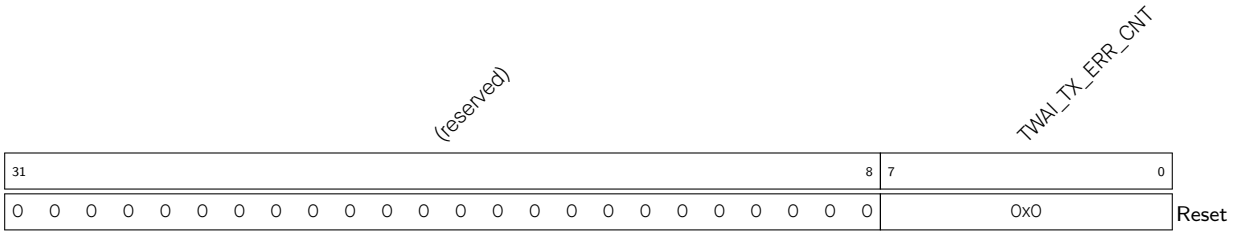
(RO)

Register 56.14. TWAI_RX_ERR_CNT_REG (0x0038)



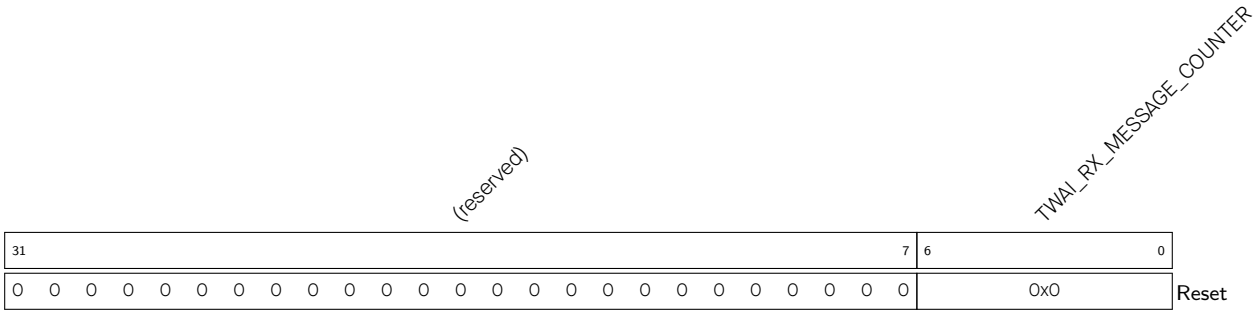
TWAI_RX_ERR_CNT 接收错误计数，数值变化发生在接收状态下。(R/W)

Register 56.15. TWAI_TX_ERR_CNT_REG (0x003C)



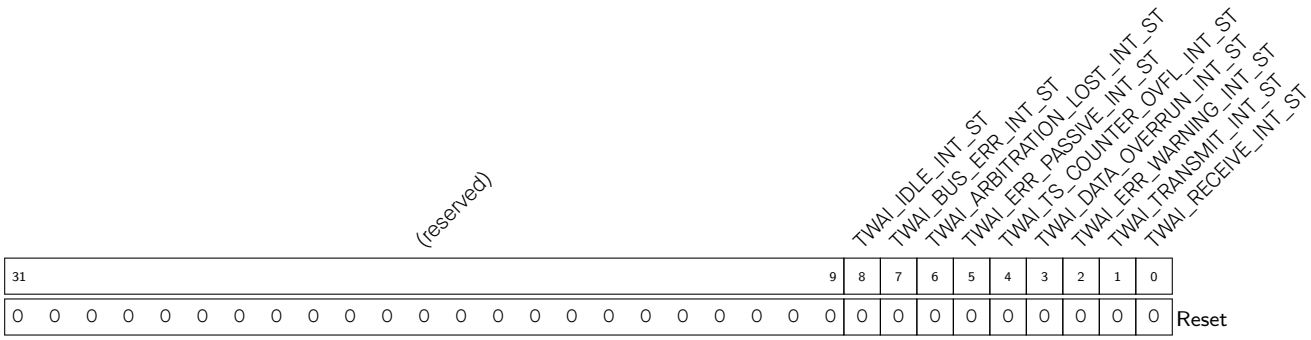
TWAI_TX_ERR_CNT 发送错误计数，数值变化发生在发送状态下。(R/W)

Register 56.16. TWAI_RX_MESSAGE_COUNTER_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER 表示接收 FIFO 中数据包的个数。(RO)

Register 56.17. TWAI_INTERRUPT_REG (0x000C)



TWAI_RECEIVE_INT_ST TWAI_RX_INT 的屏蔽中断状态。(RO)

TWAI_TRANSMIT_INT_ST TWAI_TX_INT 的屏蔽中断状态。(RO)

TWAI_ERR_WARNING_INT_ST TWAI_ERR_WARN_INT 的屏蔽中断状态。(RO)

TWAI_DATA_OVERRUN_INT_ST TWAI_OVERRUN_INT 的屏蔽中断状态。(RO)

TWAI_TS_COUNTER_OVFL_INT_ST TWAI_TS_COUNTER_OVFL_INT 的屏蔽中断状态。(RO)

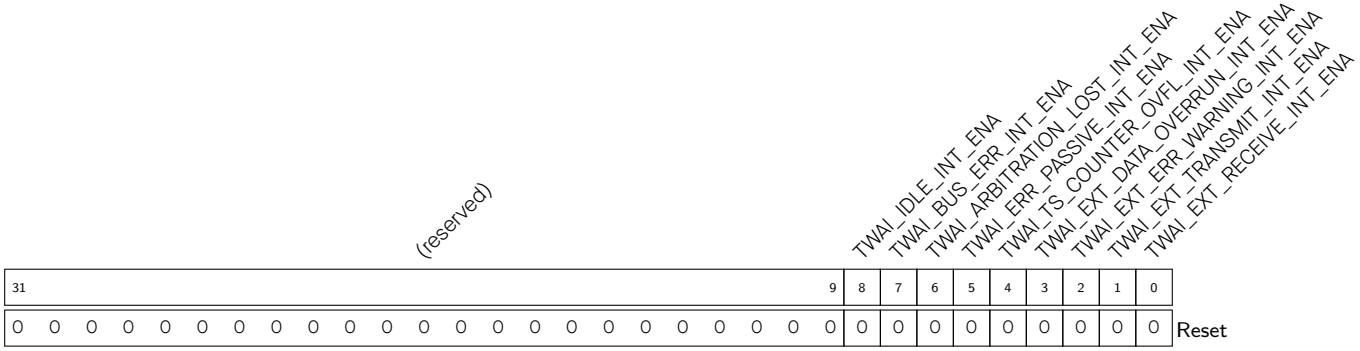
TWAI_ERR_PASSIVE_INT_ST TWAI_ERR_PASSIVE_INT 的屏蔽中断状态。(RO)

TWAI_ARBITRATION_LOST_INT_ST TWAI_ARB_LOST_INT 的屏蔽中断状态。(RO)

TWAI_BUS_ERR_INT_ST TWAI_BUS_ERR_INT 的屏蔽中断状态。(RO)

TWAI_IDLE_INT_ST TWAI_IDLE_INT 的屏蔽中断状态。(RO)

Register 56.18. TWAI_INTERRUPT_ENABLE_REG (0x0010)



TWAI_EXT_RECEIVE_INT_ENA 置 1 使能 TWAI_RX_INT 中断。(R/W)

TWAI_EXT_TRANSMIT_INT_ENA 置 1 使能 TWAI_TX_INT 中断。(R/W)

TWAI_EXT_ERR_WARNING_INT_ENA 置 1 使能 TWAI_ERR_WARN_INT 中断。(R/W)

TWAI_EXT_DATA_OVERRUN_INT_ENA 置 1 使能 TWAI_OVERRUN_INT 中断。(R/W)

TWAI_TS_COUNTER_OVFL_INT_ENA 置 1 使能 TWAI_TS_COUNTER_OVFL_INT 中断。(R/W)

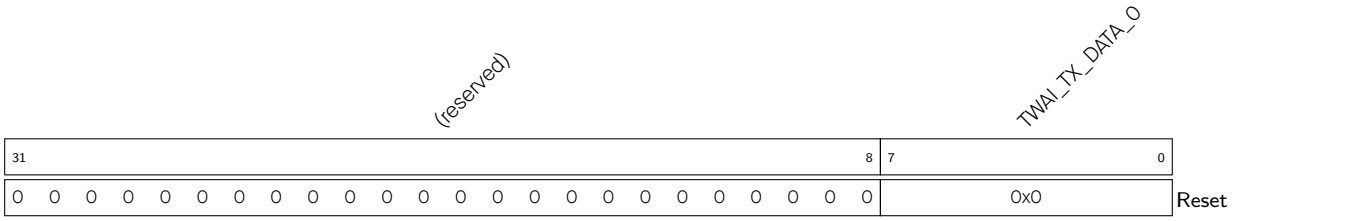
TWAI_ERR_PASSIVE_INT_ENA 置 1 使能 TWAI_ERR_PASSIVE_INT 中断。(R/W)

TWAI_ARBITRATION_LOST_INT_ENA 置 1 使能 TWAI_ARB_LOST_INT 中断。(R/W)

TWAI_BUS_ERR_INT_ENA 置 1 使能 TWAI_BUS_ERR_INT 中断。(R/W)

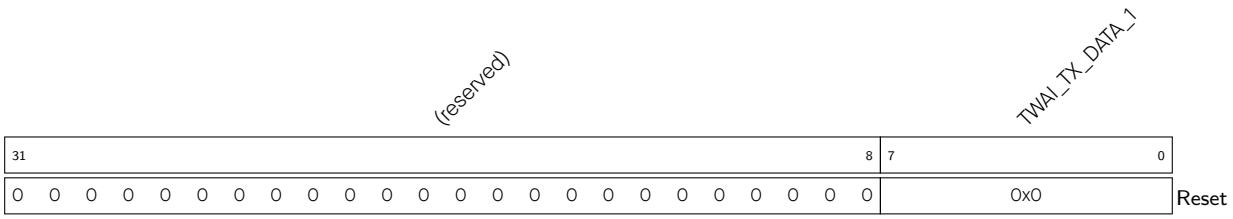
TWAI_IDLE_INT_ENA 置 1 使能 TWAI_IDLE_INT 中断。(R/W)

Register 56.19. TWAI_DATA_0_REG (0x0040)



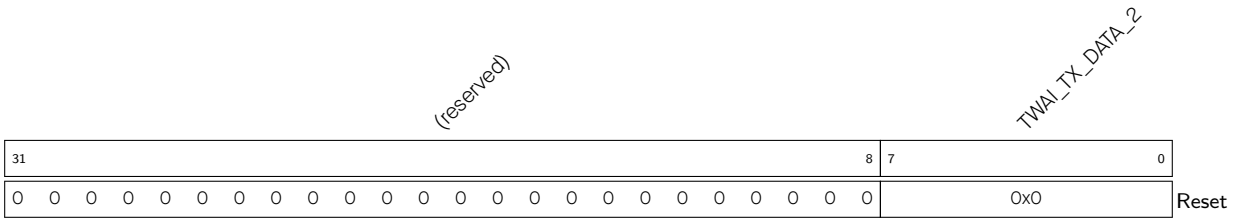
TWAI_TX_DATA_0 操作模式下，配置待发送数据的第 0 个字节内容，或者读取已接收数据的第 0 个字节内容。复位模式下，配置过滤编码的第 1 个字节。(R/W)

Register 56.20. TWAI_DATA_1_REG (0x0044)



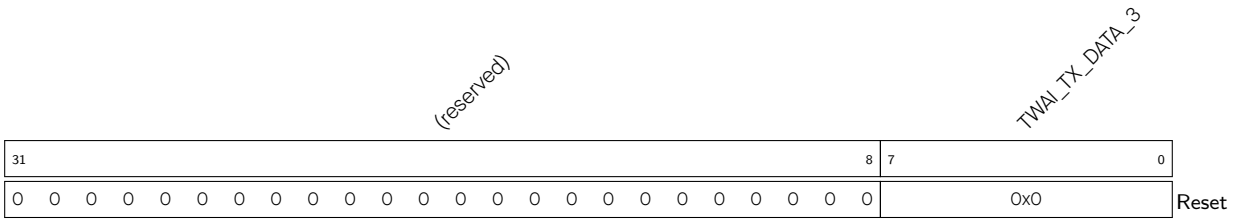
TWAI_TX_DATA_1 操作模式下，配置待发送数据的第 1 个字节内容，或者读取已接收数据的第 1 个字节内容。复位模式下，配置过滤编码的第 2 个字节。(R/W)

Register 56.21. TWAI_DATA_2_REG (0x0048)



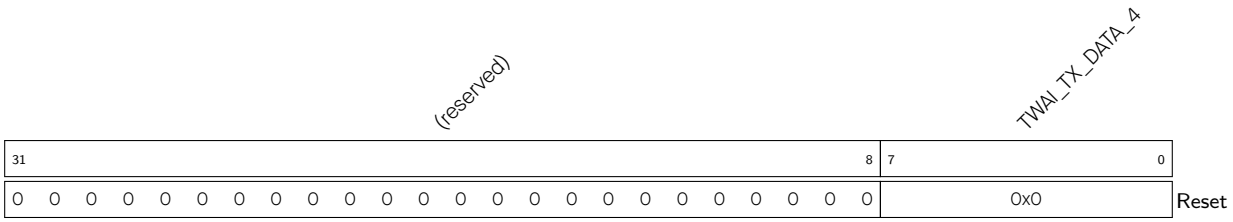
TWAI_TX_DATA_2 操作模式下，配置待发送数据的第 2 个字节内容，或者读取已接收数据的第 2 个字节内容。复位模式下，配置过滤编码的第 3 个字节。(R/W)

Register 56.22. TWAI_DATA_3_REG (0x004C)



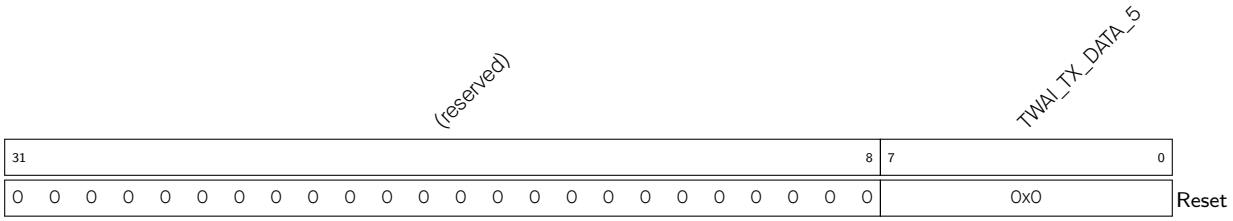
TWAI_TX_DATA_3 操作模式下，配置待发送数据的第 3 个字节内容，或者读取已接收数据的第 3 个字节内容。复位模式下，配置过滤编码的第 4 个字节。(R/W)

Register 56.23. TWAI_DATA_4_REG (0x0050)



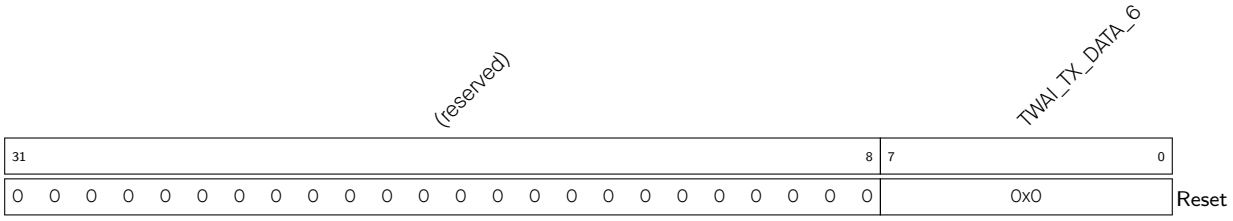
TWAI_TX_DATA_4 操作模式下，配置待发送数据的第 4 个字节内容，或者读取已接收数据的第 4 个字节内容。复位模式下，配置屏蔽过滤编码的第 1 个字节。(R/W)

Register 56.24. TWAI_DATA_5_REG (0x0054)



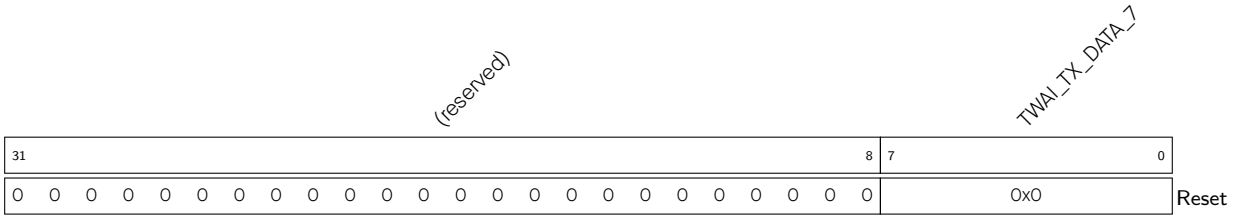
TWAI_TX_DATA_5 操作模式下，配置待发送数据的第 5 个字节内容，或者读取已接收数据的第 5 个字节内容。复位模式下，配置屏蔽过滤编码的第 2 个字节。(R/W)

Register 56.25. TWAI_DATA_6_REG (0x0058)



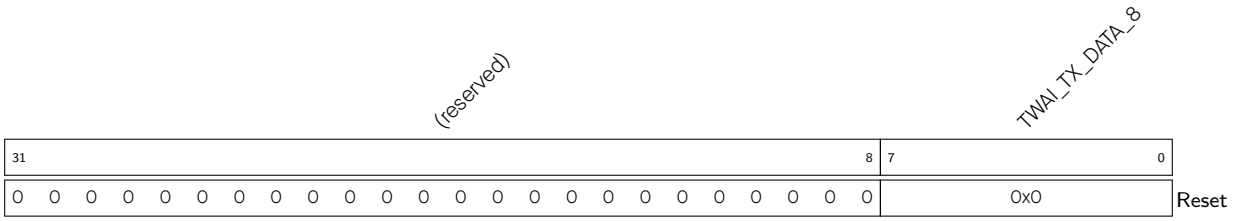
TWAI_TX_DATA_6 操作模式下，配置待发送数据的第 6 个字节内容，或者读取已接收数据的第 6 个字节内容。复位模式下，配置屏蔽过滤编码的第 3 个字节。(R/W)

Register 56.26. TWAI_DATA_7_REG (0x005C)



TWAI_TX_DATA_7 操作模式下，配置待发送数据的第 7 个字节内容，或者读取已接收数据的第 7 个字节内容。复位模式下，配置屏蔽过滤编码的第 4 个字节。(R/W)

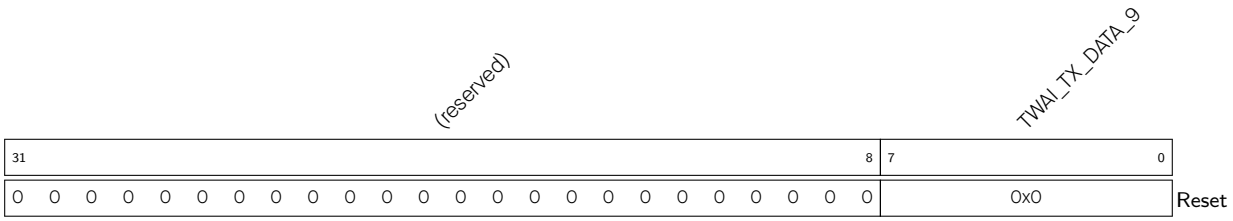
Register 56.27. TWAI_DATA_8_REG (0x0060)



TWAI_TX_DATA_8 操作模式下，配置待发送数据的第 8 个字节内容，或者读取已接收数据的第 8 个字节内容。(R/W)

复位模式下，保留。(RO)

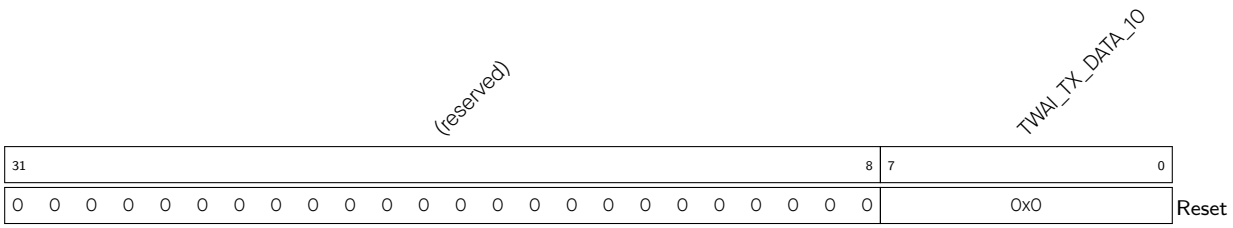
Register 56.28. TWAI_DATA_9_REG (0x0064)



TWAI_TX_DATA_9 操作模式下，配置待发送数据的第 9 个字节内容，或者读取已接收数据的第 9 个字节内容。(R/W)

复位模式下，保留。(RO)

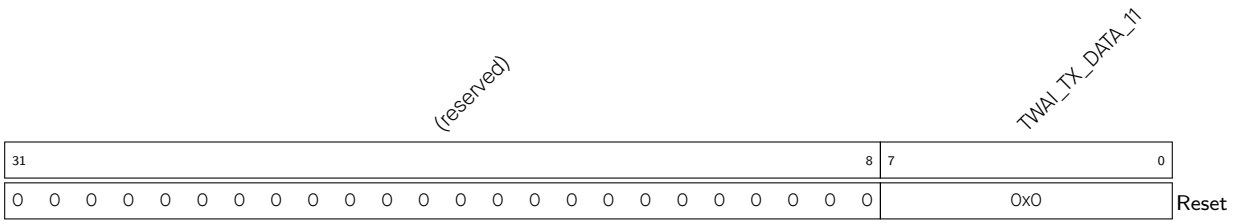
Register 56.29. TWAI_DATA_10_REG (0x0068)



TWAI_TX_DATA_10 操作模式下，配置待发送数据的第 10 个字节内容，或者读取已接收数据的第 10 个字节内容。(R/W)

复位模式下，保留。(RO)

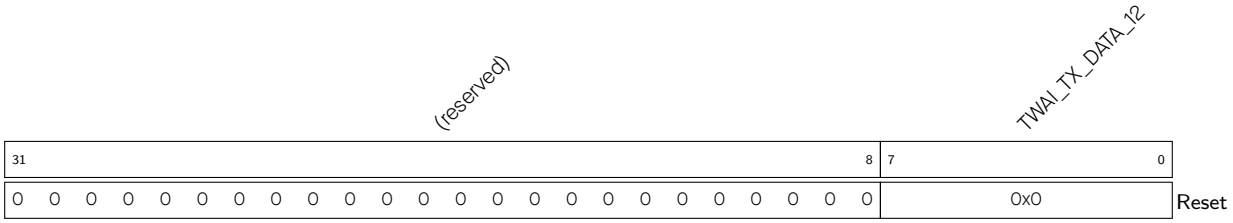
Register 56.30. TWAI_DATA_11_REG (0x006C)



TWAI_TX_DATA_11 操作模式下，配置待发送数据的第 11 个字节内容，或者读取已接收数据的第 11 个字节内容。(R/W)

复位模式下，保留。(RO)

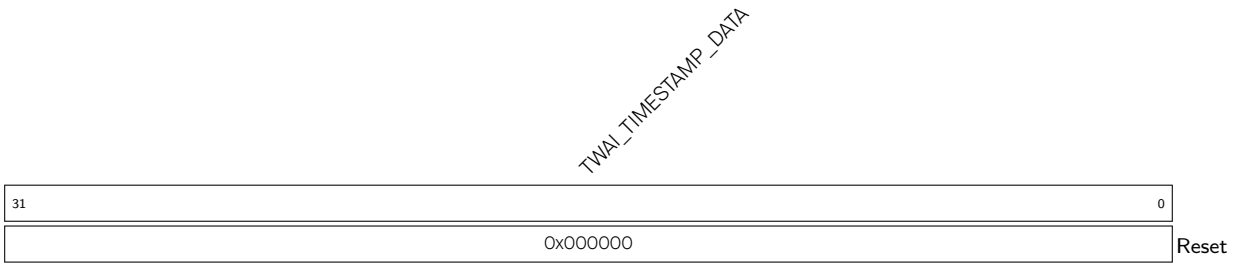
Register 56.31. TWAI_DATA_12_REG (0x0070)



TWAI_TX_DATA_12 操作模式下，当软件启动写操作时，配置待发送数据的第 12 个字节内容，或者读取已接收数据的第 12 个字节内容。(R/W)

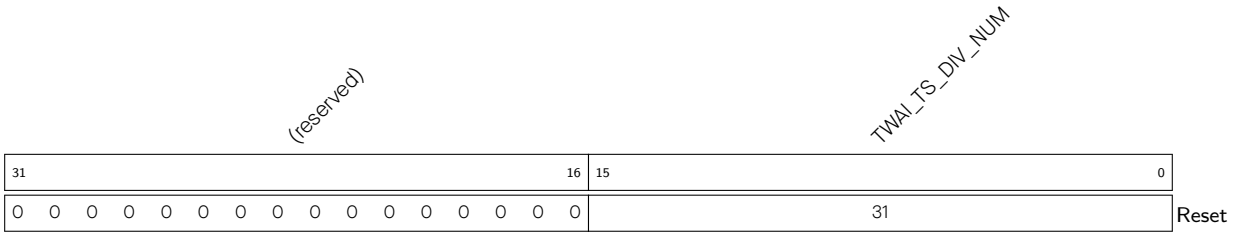
复位模式下，保留。(RO)

Register 56.32. TWAI_TIMESTAMP_DATA_REG (0x0094)



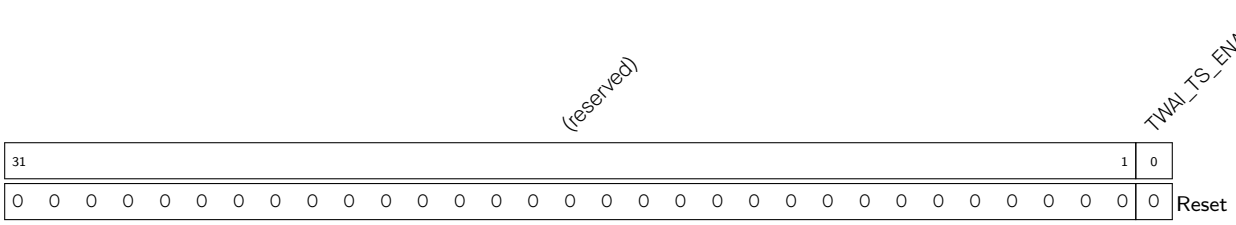
TWAI_TIMESTAMP_DATA 表示 RX FIFO 读指针当前所指向的 TWAI 帧的时间戳，对应接收完该 TWAI 帧的时间点。仅当 TWAI_TS_ENABLE 使能时有效。(RO)

Register 56.33. TWAI_TIMESTAMP_PRESCALER_REG (0x0098)



TWAI_TS_DIV_NUM 配置时间戳计数器的时钟分频系数。(R/W)

Register 56.34. TWAI_TIMESTAMP_CFG_REG (0x009C)



TWAI_TS_ENABLE 配置是否使能时间戳收集功能。

0：禁用

1：使能

(R/W)

第 57 章

SD/MMC 主机控制器 (SDHOST)

57.1 概述

ESP32-P4 存储卡接口控制器提供了一个访问安全数字输入输出卡 (SDIO)、MMC 卡和 CE-ATA 设备的硬件接口，用于连接高级外围设备总线 (APB) 和外部存储设备。该控制器支持两个外部卡（卡 0 和卡 1）。所有 SD/MMC 模块接口信号都须通过 GPIO 矩阵传输至 GPIO 管脚。

57.2 主要特性

该模块支持以下特性：

- 支持两个外部卡
- 支持 3.0、3.01 版本 SD 存储卡标准，最高支持传输速率为 DDR50
- 支持 4.41、4.5、4.51 版本 MMC，最高支持传输速率为 DDR50
- 支持 1.1 版本 CE-ATA
- 支持 1-bit、4-bit 和 8-bit 位宽模式

SD/MMC 控制器连接的拓扑结构如图 57.2-1 所示。该控制器支持两组外设工作，但不支持同时工作。

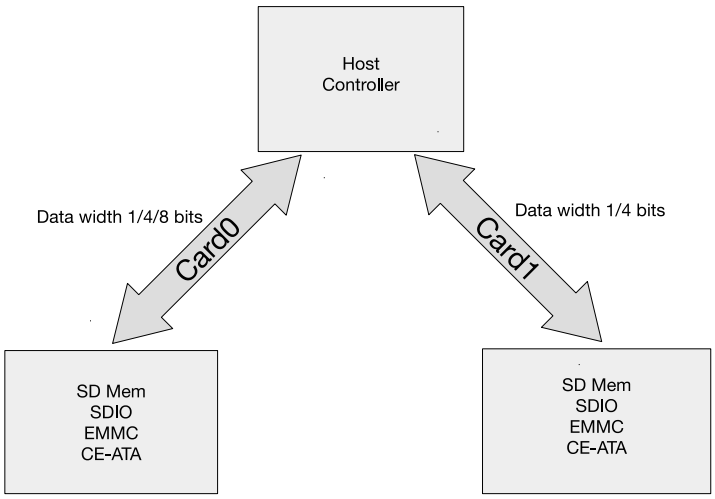


图 57.2-1. SD/MMC 控制器连接的拓扑结构

57.3 SD/MMC 外部接口信号

SD/MMC 的外部接口信号主要为时钟信号 (sdhost_cclk_out_1.eg:card1)、命令信号 (sdhost_ccmd_out_1)、数据信号 (sdhost_cdata_in_1[7:0]/sdhost_cdata_out_1[7:0])，SD/MMC 控制器可通过这些外部接口信号与外部设备通信。其它信号还包括卡中断信号、卡检测信号和写保护信号等。各个信号的方向如图 57.3-1 所示。每个管脚的方向和描述如表 57.3-1 所示。

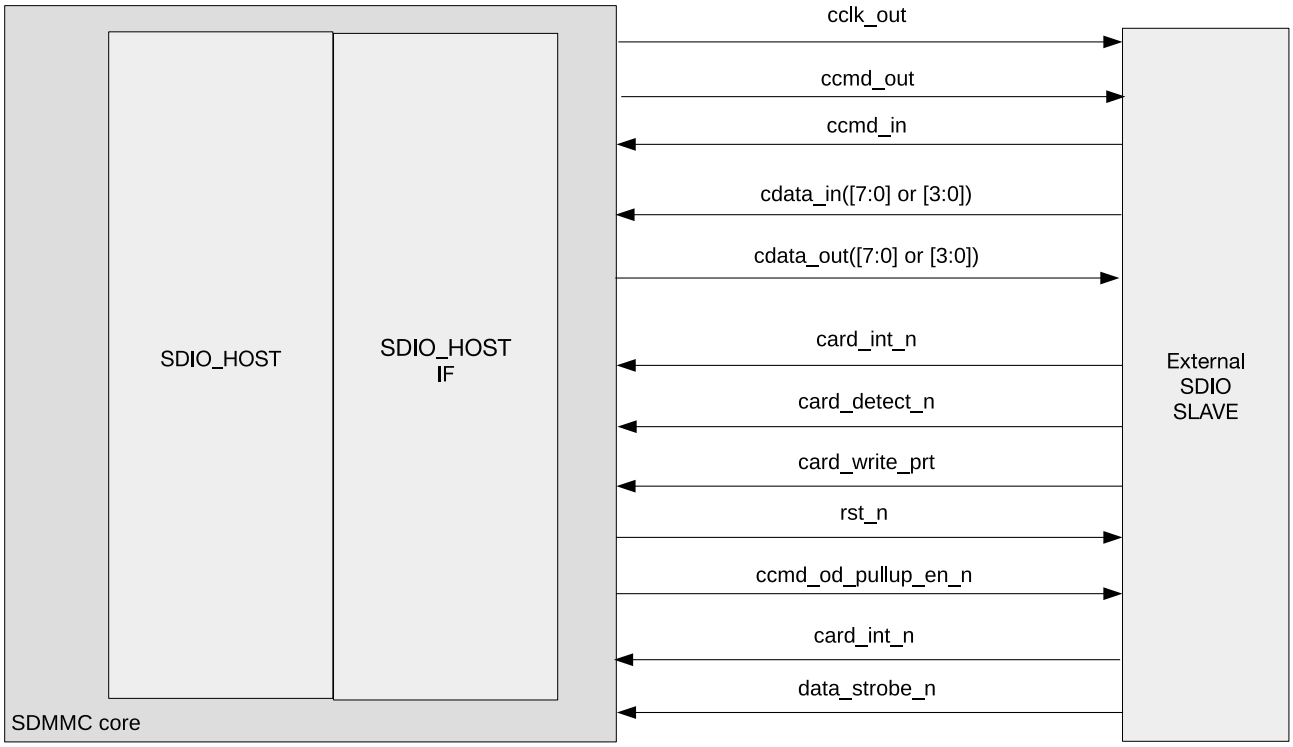


图 57.3-1. SD/MMC 控制器外部接口信号

表 57.3-1. SD/MMC 信号描述

信号	方向	描述
sdhost_cclk_out	输出	向从机发送时钟信号
sdhost_ccmd	双向	双向命令/响应线
sdhost_cdata	双向	双向数据读/写线
sdhost_card_detect_n	输入	卡检测输入线
sdhost_card_write_prt	输入	卡写保护状态输入

57.4 功能描述

57.4.1 SD/MMC 主机控制器结构

SD/MMC 主机控制器主要由两大功能块组成，如图 57.4-1 所示，分别为：

- 总线接口单元 (BIU)：提供 APB 总线访问寄存器的接口、RAM 数据访问方式和 DMA 数据读写操作
- 卡接口单元 (CIU)：处理外部存储卡的接口协议，控制时钟。

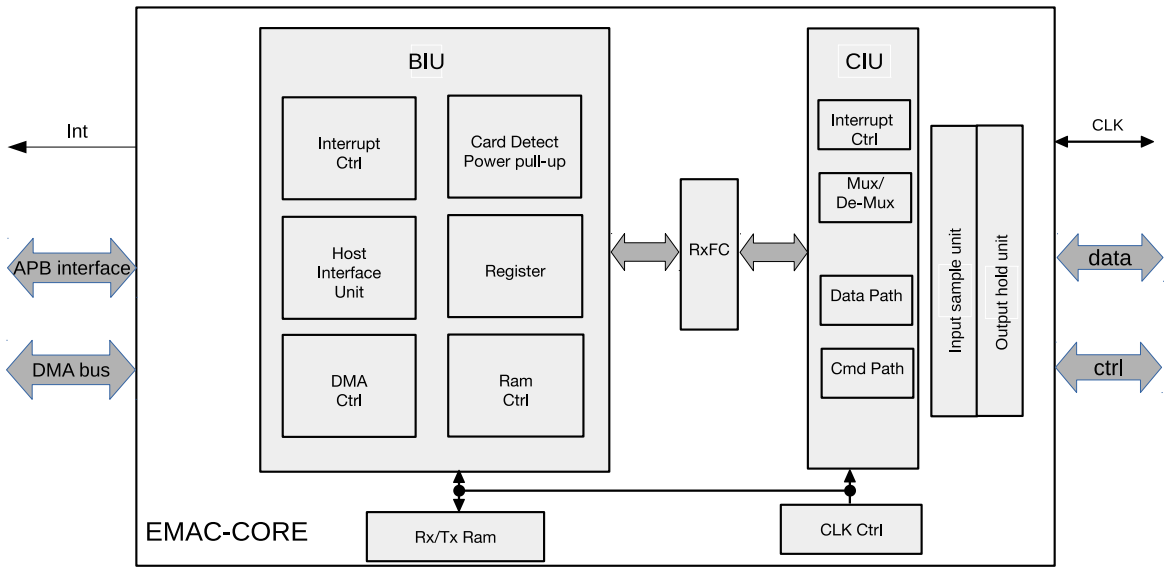


图 57.4-1. SDIO 主机结构框图

57.4.1.1 总线接口单元 (BIU)

该模块提供了通过主机接口单元 (HIU) 访问寄存器和 RAM 数据的方式。除此之外，它通过 DMA 接口提供独立的 RAM 数据访问。它通过 DMA 接口访问存储器中的数据。图 57.4-1 描述了该模块的内部结构。

BIU 支持以下功能：

- 主机接口
- DMA 接口
- 中断控制
- 寄存器访问
- FIFO 访问
- 上电/上拉控制和卡检测

57.4.1.2 卡接口单元 (CIU)

该模块实现卡专用接口协议。在 CIU 中，命令通路 (Cmd Path) 控制单元和数据通路 (Data Path) 控制单元将控制器分别连接到 SD/MMC/CE-ATA 卡的命令接口和数据接口。CIU 还提供时钟控制。图 57.4-1 描述了 CIU 的内部结构。

CIU 由以下主要功能模块组成：

- 命令通路
- 数据通路
- SDIO 中断控制
- 时钟控制
- Mux/De-Mux 单元

57.4.2 命令通路

命令通路具有以下功能：

- 配置时钟参数
- 配置卡命令参数
- 向卡总线发送命令 (sdhost_ccmd_out)
- 从卡总线接收响应 (sdhost_ccmd_in)
- 向 BIU 发送响应
- 在命令线上发送 P-bit

命令通路状态机如图 57.4-2 所示。

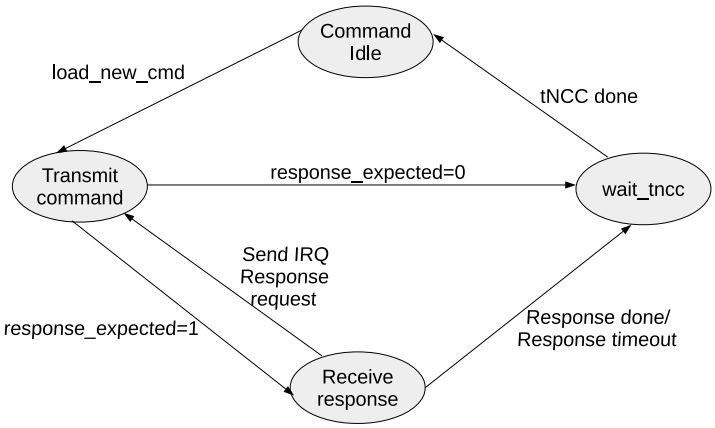


图 57.4-2. 命令通路状态机

57.4.3 数据通路

发送时，数据通路模块会取出 RAM 中的数据，并将其发送至 sdhost_cdata_out；接收时，数据通路会接收 sdhost_cdata_in 上的数据，并将其导入 RAM 中。数据发送命令未运行时，数据通路将加载新的数据参数，即希望发送的数据、读/写数据发送、流/块发送、块大小、字节数、卡类型、超时寄存器等。

如果在命令寄存器 (SDHOST_CMD_REG) 中置了 SDHOST_DATA_EXPECTED 位，则新命令为数据传输命令，数据通路将执行以下操作之一：

- 若 SDHOST_READ_WRITE 位为 1，发送数据
- 若 SDHOST_READ_WRITE 位为 0，接收数据

57.4.3.1 数据发送

接收到数据写入命令的响应后，该模块将在两个时钟周期开始发送数据。即使命令通路在响应信息中检测到响应错误或循环冗余检查 (CRC) 错误，数据发送也会正常进行。但是，如果直到响应超时仍没有从卡接收到响应，则不发送数据。根据 SDHOST_CMD_REG 寄存器中 SDHOST_TRANSFER_MODE 位的值，数据发送状态机将数据以流或块的形式加至卡数据总线上。数据发送状态机如图 57.4-3 所示。

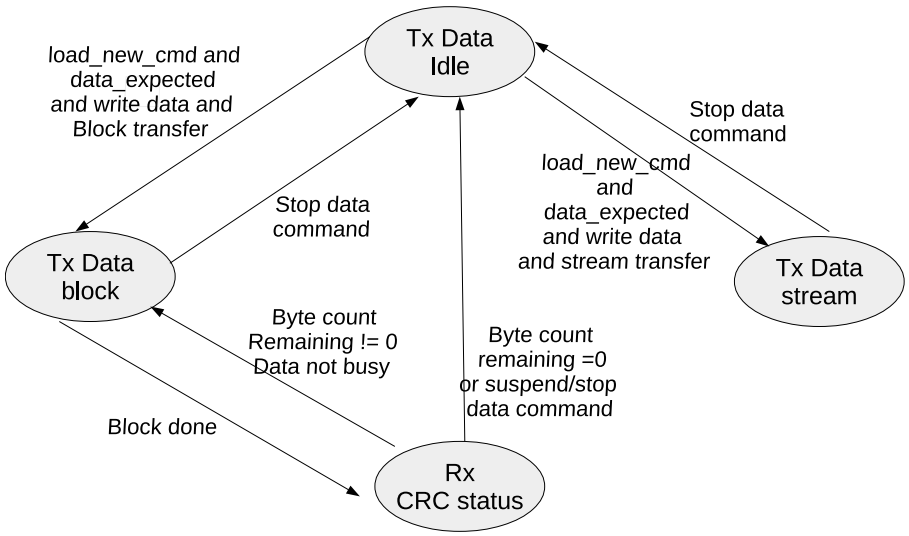


图 57.4-3. 数据发送状态机

57.4.3.2 数据接收

数据读取命令完成两个时钟周期后，该模块将开始接收数据。即使命令通路检测到响应错误或 CRC 错误，数据接收也会正常进行。如果一直未从卡接收到响应而发生了响应超时，则 BIU 无法接收到 CIU 数据发送完成的信号。如果 CIU 发送的命令是卡所禁止的非法操作，那么将无法开始读取从卡发送的数，且 BIU 也不会接收到 CIU 数据发送完成的信号。

若在数据超时前未接收到数据，数据通路将向 BIU 发出数据超时信号并结束数据传输。根据 SDHOST_CMD_REG 寄存器中的 SDHOST_TRANSFER_MODE 位的值，数据接收状态机以流或块的形式从卡数据总线获取数据。数据接收状态机如图 57.4-4 所示。

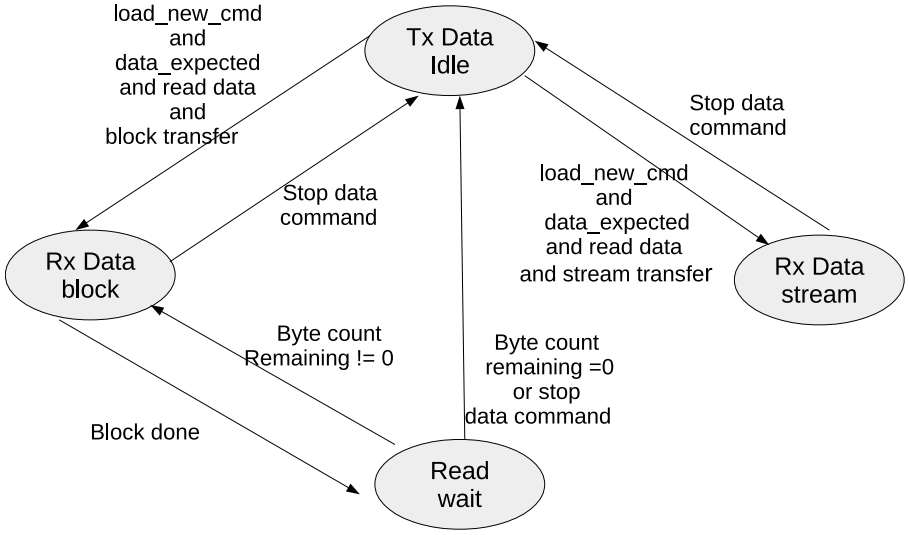


图 57.4-4. 数据接收状态机

57.5 CIU 操作的软件限制

- 一次只能选择一个卡执行命令或发送数据。例如，当向某张卡发送数据或从这张卡接收数据时，不可以给另一张卡发送其它新的命令。但是，可以将新的命令发送到同一张卡上，使其读取设备状态或停止数据传

输。

- 一次只能发出一个数据传输命令。
- 在开放式写卡操作期间，如果由于 RAM 为空导致卡时钟停止，那么软件必须先将数据填充到 RAM 中并启动卡时钟，然后才可以向卡发出一个停止/中止命令。
- 在 SDIO/Combo 卡数据传输期间，如果卡功能暂停，并且软件想要恢复所暂停的数据传输，则其必须先重置 RAM（置位并释放 SDHOST_FIFO_RESET）并启动恢复命令，这和启动一个新的数据传输命令相似。
- 要在进行卡数据传输中发出卡复位命令（CMD0、CMD15 或 CMD52_reset），软件必须在 SDHOST_CMD_REG 寄存器上置位 SDHOST_STOP_ABORT_CMD，保证 CIU 可以在发出卡复位命令后停止数据传输。
- 当在 SDHOST_RINTSTS_REG 寄存器中设置数据结束位错误时，CIU 无法控制 SDIO 中断。因此在该情况下，软件应忽略 SDIO 中断，并向卡发出停止/中止命令使其停止发送数据。
- 在一次读卡过程中，如果由于 RAM 已满而导致卡时钟停止，那么软件将至少读取两个 RAM 地址来重启卡时钟。
- 在一次命令/数据传输中只能选取一个 CE-ATA 设备。例如，当一个 CE-ATA 设备已经在传输数据，则其它 CE-ATA 设备不可传输新命令。
- 使能 CE-ATA 设备的中断 (nIEN=0) 后，如果该设备已经有正在执行的 SDHOST_RW_BLK 命令，则不可以再向这一设备发送新的 SDHOST_RW_BLK 命令。只有在等待 CCS (Command Completion Signal) 时可以发送 CCSD (Command Completion Signal Disable)。
- 但是，如果一个 CE-ATA (nIEN=1) 设备未使能，则可以向同一设备发送新的命令来读取状态信息。
- CE-ATA 设备不支持开放式数据传输。
- CE-ATA 数据传输不支持 sdhost_send_auto_stop 信号（禁止软件置位 SDHOST_SEND_AUTO_STOP）。

置位命令起始位后，在发送出命令之前以下寄存器的值不可改变：

- CMD — 命令
- CMDARG — 命令参数
- BYTCNT — 字节计数
- BLKSIZ — 块大小
- CLKDIV — 时钟分频器
- CKLENA — 时钟使能
- CLKSRC — 时钟源
- TMOUT — 超时
- CTYPE — 卡类型

57.6 收发数据 RAM

RAM 子模块是一个收发数据缓存，分为接收和发送两个单元。可通过 CPU 和 DMA 实现数据的收发，从而进行读写操作，可参阅章节 57.8。

57.6.1 TX RAM 模块

可通过两种方式使能写入操作：CPU 和 DMA 写入。

使用 CPU 将数据写入 TX RAM 时，可将数据通过 APB 接口直接写入到 SDHOST_BUFFIFO_REG 寄存器中，该寄存器对应的 FIFO 位宽为 32 位，深度为 512。

使用 DMA 将数据写入 TX RAM 时，可通过构造 DMA 链表描述符实现，参阅章节 57.8。

57.6.2 RX RAM 模块

可通过两种方式使能读取操作：CPU 和 DMA 读取。

使用 CPU 从 RX RAM 读取数据时，可通过 APB 接口读取 SDHOST_BUFFIFO_REG 寄存器获取数据，该寄存器对应的 FIFO 位宽为 32 位，深度为 512。

使用 DMA 从 RX RAM 读取数据时，可通过构造 DMA 链表描述符实现，参阅章节 57.8。

57.7 DMA 链表

链表中的每个单元都由两部分组成：描述符和一个数据缓存。也就是说，每一个描述符都指向一个独特的数据缓存，同时还连接至下一个描述符。链表结构如图 57.7-1 所示。

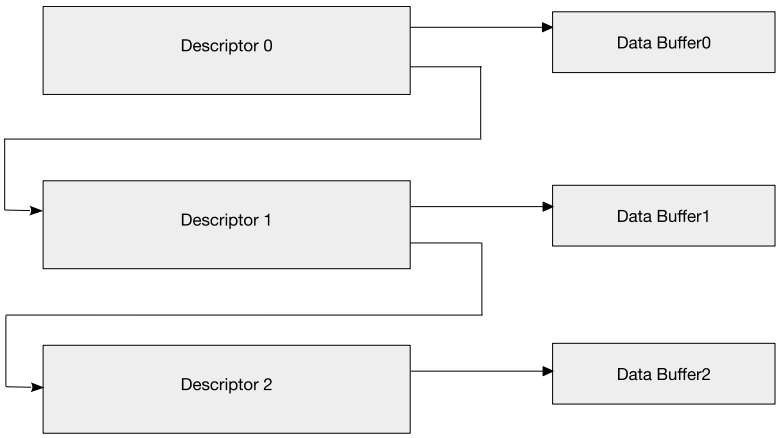


图 57.7-1. 链表结构

57.8 DMA 描述符结构

每个描述符由 4 个字组成，如图 57.8-1 所示。表 57.8-1、表 57.8-2、表 57.8-3、表 57.8-4 为这 4 个字的描述。

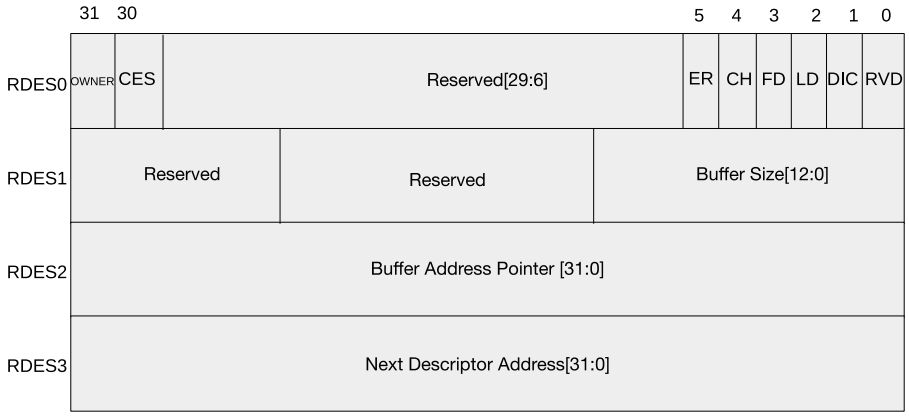


图 57.8-1. 描述符结构

DESO 单元包含控制和状态信息。

表 57.8-1. DESO 单元描述

位	名称	描述
31	OWNER	置位时，表明该描述符允许的操作者为 DMA 控制器。该位被重置时，表明该描述符允许的操作者为主机。DMA 控制器在完成数据传输后清除该位。
30	CES (Card Error Summary)	该位显示卡的读/写状态。 该位为 SDHOST_RINTSTS_REG 寄存器中下列字段的或运算： • EBE：结束位错误 • RTO：响应超时 • RCRC：响应 CRC • SBE：起始位错误 • DRTO：读取数据超时 • DCRC：对接收数据进行循环冗余校验 • RE：响应错误
29:6	保留	保留
5	ER (End of Ring)	置位时，表示当前描述符为数据包的最后一个描述符。此时，DMA 控制器返回到链表的基地址，创建一个链表环。
4	CH (Second Address Chained)	置位时，表示该描述符中的第二个地址为下一描述符的地址。此时，BS2 (DES1[25:13]) 应全部归零。
3	FD (First Descriptor)	置位时，表示该描述符内包含了数据的第一个缓存。若该缓存大小为 0，则下一个描述符内为数据的开始。
2	LD (Last Descriptor)	该位与 DMA 传输的最后一个数据块相关。置位时，表示该描述符指向的缓存是数据的最后一个缓存。在该描述符完成之后，剩余字节数为 0。也就是说，带有被置位的 LD 位的描述符完成之后，剩余字节数应为 0。

位	名称	描述
1	DIC (Disable Interrupt on Completion)	置位时，为了保留在该描述符指向的缓存中结束的数据，该位将阻止置位 DMA 状态寄存器 (SDHOST_IDSTS_REG) 上 TI/RI 位。
0	保留	保留

DES1 单元包含缓存大小。

表 57.8-2. DES1 单元描述

位	名称	描述
31:26	保留	保留
25:13	保留	保留
12:0	BS（缓存大小）	表示数据缓存的字节大小。该数值必须为 4 的倍数。否则，其大小将无法被定义成功。该字段不可设置为 0。

DES2 单元包含指向数据缓存的地址指针。

表 57.8-3. DES2 单元描述

位	名称	描述
31:0	Buffer Address Pointer（数据缓存地址指针）	表示数据缓存的地址，须按字对齐。

如果当前的描述符不是链表中的最后一个，则 DES3 单元包含指向下一个描述符的地址指针。

表 57.8-4. DES3 单元描述

位	名称	描述
31:0	Next Descriptor Address（下一个描述符地址）	如果置位 CH (DESO[4])，则该位中有指向包含下一个描述符位置的指针。 如果该描述符不是链表结构中的最后一个描述符，则下一个描述符的地址指针必须满足 DES3[1:0] = 0。

57.9 操作流程

57.9.1 寄存器初始化流程

寄存器初始化过程如下：

1. 写控制寄存器 (SDHOST_CTRL_REG)，原始中断寄存器 (SDHOST_RINTSTS_REG) 和 SDIO 中断屏蔽寄存器 (SDHOST_INTMASK_REG)，清除残留的中断信息，设置控制和中断使能参数。
2. 写时钟分频器配置寄存器 (SDHOST_CLKDIV_REG)，时钟源选择寄存器 (SDHOST_CLKSRC_REG) 和时钟使能寄存器 (SDHOST_CLKENA_REG)，设置输出给卡的时钟。
3. 根据卡的具体参数写其他配置寄存器。例如，可以设置

- 卡位宽 (SDHOST_CTYPE_REG)
- 用户身份标识 (SDHOST_USRID_REG)
- UHS 模式电压和 DDR (SDHOST_UHS_REG)
- EMMC 起始位模式 (SDHOST_EMMCDDR_REG)
- SDIO 模式 (SDHOST_CLK_EDGE_SEL_REG)
- 调整超时阈值（SDHOST_TMOUT_REG）
- 防抖滤波参数（SDHOST_DEBNCE_REG）
- 内部时序（SDHOST_ENSHIFT_REG）

配置 SDHOST_RST_N_REG 可进行卡复位。

57.9.2 命令发送流程

命令发送过程如下：

1. 写命令参数数据寄存器 (SDHOST_CMDARG_REG)，设置命令参数数据。
2. 如果需要传输数据，写数据传输长度配置寄存器 (SDHOST_BYTCNT_REG)，卡数据块大小配置寄存器 (SDHOST_BLKSIZ_REG)，卡数据阈值控制寄存器 (SDHOST_CARDTHRCTL_REG)，设置数据传输参数。
3. 如果需要传输数据，可以通过 DMA 或者 FIFO 实现，建议使用 DMA 进行数据传输。
   - 如果选择通过 DMA 传输数据，参考章节 57.9.3、章节 57.9.4、及章节 57.9.5。
   - 如果选择通过 FIFO 传输数据，需写 FIFO 配置寄存器 (SDHOST_FIFOTH_REG) 设置所需的数据收发阈值，若为向卡写数据还需将数据写入 FIFO （SDHOST_BUFFIFO_REG）并在命令执行过程中持续将数据补充写入 FIFO 。
4. 写命令和启动配置寄存器 (SDHOST_CMD_REG)，设置命令信息并启动命令。
5. 硬件接收命令后会将寄存器 SDHOST_CMD_REG 中的 SDHOST_START_CMD 字段清 0，硬件无法接收命令时会将寄存器 SDHOST_RINTSTS_REG 中的第 12 位 HLE 置 1。
6. 读寄存器 SDHOST_RINTSTS_REG 或使能中断来获知命令执行情况。如有需要，可以读寄存器 SDHOST_RESP*n*_REG（*n* 为 0 ~ 3）获知卡返回的命令响应信息。
7. 如果选择通过 FIFO 从卡读取数据，读 FIFO (SDHOST_BUFFIFO_REG) 可以获取卡返回的数据。

具体命令格式请参考卡规范协议，并根据命令格式确定各寄存器字段的配置值。

57.9.3 DMA 控制器初始化流程

DMA 控制器初始化过程如下：

1. 写 DMA 控制器的总线模式寄存器 (SDHOST_BMOD_REG)，设置主机总线访问参数。
2. 写 DMA 控制器的中断使能寄存器 (SDHOST_IDINTEN_REG)，屏蔽不必要的中断类型。
3. 软件驱动器创建发送链表或接收链表。然后向 DMA 控制器链表基地址寄存器 (SDHOST_DBADDR_REG) 中写入链表的起始地址。
4. DMA 控制器引擎尝试从链表中获取描述符。

57.9.4 DMA 控制器数据发送流程

DMA 控制器发送数据的过程如下：

1. 主机设置发送数据的描述符单元 (DES0 ~ DES3)，置位 OWNER 位 (DES0[31])，同时准备数据缓存。
2. 主机在命令寄存器 (SDHOST_CMD_REG) 中写入数据命令。
3. 主机设置所需的数据发送阈值（在 SDHOST_FIFOTH_REG 寄存器中的 SDHOST_TX_WMARK 字段进行）。
4. DMA 控制器引擎读取描述符并检查 OWNER 位。如果 OWNER 位未置位，表明该描述符所允许的操作者为主机。此时，DMA 控制器进入挂起状态，并在 SDHOST_IDSTS_REG 寄存器中产生禁能描述符中断。然后，主机需在 SDHOST_PLDMND_REG 中写入任意值来释放 DMA 控制器资源。
5. DMA 控制器等待 SDHOST_RINTSTS_REG 寄存器中的命令完成 (CMDD) 位变成 1，如果 BIU 无报错，则表明数据发送已完成。
6. 然后，DMA 控制器引擎等待 BIU 发送的 DMA 总线请求，该请求将基于配置的数据发送阈值生成。对于使用突发传输不能访问的最后一个字节，则在 AHB 总线上执行单次发送。
7. DMA 控制器从主机存储器的数据缓存获取发送数据，并通过 RAM 将其发送至卡中。
8. 当数据跨越多个描述符时，DMA 控制器将获取下一个描述符，并继续使用后面的描述符进行后续操作。最后一个描述符位将显示该数据是否跨越多个描述符。
9. 当数据发送完成且 SDHOST_IDSTS_TI 位已使能，将通过置位 SDHOST_IDSTS_TI 将状态信息更新至寄存器 SDHOST_IDSTS_REG。同时，DMA 控制器将通过对 DES0 执行写操作来清除 OWNER 位。

57.9.5 DMA 控制器数据接收流程

DMA 控制器接收数据的过程如下：

1. 主机设置接收数据的描述符单元 (DES0 ~ DES3)，置位 OWNER 位 (DES0[31])。
2. 主机在命令寄存器 (SDHOST_CMD_REG) 中写入读数据命令。
3. 主机设置所需的数据接收阈值（在 SDHOST_FIFOTH_REG 寄存器中的 SDHOST_RX_WMARK 字段进行）。
4. DMA 控制器引擎读取描述符并检查 OWNER 位。如果 OWNER 位未置位，表明该描述符所允许的操作者为主机。此时，DMA 控制器进入挂起状态，并在 SDHOST_IDSTS_REG 寄存器中产生禁能描述符中断。然后，软件需在 SDHOST_PLDMND_REG 中写入任意值释放 DMA 控制器资源。
5. DMA 控制器等待 SDHOST_RINTSTS_REG 寄存器中的命令完成位 (CMDD) 变成 1，如果 BIU 无报错，则表明数据接收已完成。
6. DMA 控制器引擎等待 BIU 发送的 DMA 总线请求。该请求将基于配置的数据接收阈值生成。对于使用突发传输不能访问的最后一个字节，则在 AHB 总线上执行单次传输。
7. DMA 控制器从 RAM 获取数据，并将其发送至主机存储器。
8. 当数据跨越多个描述符时，DMA 控制器将获取下一个描述符，并继续使用后面的描述符进行后续操作。最后一个描述符位将显示该数据是否跨越多个描述符。
9. 当数据接收完成且 SDHOST_IDSTS_RI 位已使能，将通过置位 SDHOST_IDSTS_RI 将状态信息更新至寄存器 SDHOST_IDSTS_REG。同时，DMA 控制器将通过对 DES0 执行写操作来清除 OWNER 位。

57.10 时钟相位选择

SD/MMC 主机控制器的源时钟相位可以选择，用来调整输入数据信号或输出数据信号的建立时间的时序。

SD/MMC 主机控制器的源时钟可以选择高频高性能时钟或者低频低功耗时钟。选择高频高性能时钟时，可通过设置寄存器 SDHOST_DLL_CLK_CONF_REG，来调整内部信号时钟相位、输出信号驱动时钟相位和输入信号采样时钟相位，调整单位为时钟源周期的 1/64。选择低频低功耗时钟时，可通过设置系统时钟相关寄存器，来调整内部信号时钟相位、输出信号驱动时钟相位和输入信号采样时钟相位，可选择的相位有 0 度、90 度、180 度和 270 度，即调整单位为时钟源周期的 1/4。

请查看章节请查看章节 11 复位和时钟，了解相关内容。

57.11 中断

SD/MMC 主机控制器的下列内部中断源可以生成中断信号 SDIO_HOST_INTR，并发送给中断矩阵。

- SDIO_INT：来自卡的中断。
- EBE：结束位错误/无 CRC 错误时触发。
- ACD：自动命令结束时触发。
- SBE/BCI：接收起始位错误时触发。
- HLE：硬件锁定写入错误时触发。
- FRUN：FIFO 空/满错误时触发。
- HTO：主机填充数据超时时触发。
- DRTO：数据读取超时时触发。
- RTO：卡响应超时时触发。
- DCRC：数据 CRC 错误时触发。
- RCRC：卡响应 CRC 错误时触发。
- RXDR：从卡读取数据，FIFO 内数据量超过阈值时触发。
- TXDR：向卡发送数据，FIFO 内数据量少于阈值时触发。
- DTO：数据传输结束时触发。
- CMDD：命令执行完毕时触发。
- RE：卡响应错误时触发。
- CD：检测到卡时触发。
- IDSTS_RI：DMA 链表的数据接收完成时触发。
- IDSTS_TI：DMA 链表的数据发送完成时触发。
- IDSTS_FBE：致命总线错误发生时触发，发送/接收期间主机中止。此时，DMA 禁用所有的总线访问。
- IDSTS_DU：DMA 链表由于 OWNER 位 = 0 (DESO [31] = 0) 而不可用时触发。

寄存器 SDHOST_RINTSTS_REG 和 SDHOST_IDSTS_REG 中包含所有可能导致中断的位。寄存器 SDHOST_INTMASK_REG 和 SDHOST_IDINTEN_REG 中包含所有可导致中断的事件的使能位。

寄存器 SDHOST_IDSTS_REG 中有三组中断汇总：卡错误汇总 (第 5 位：SDHOST_IDSTS_CES)，正常中断汇总 (第 8 位：SDHOST_IDSTS_NIS) 和异常中断汇总 (第 9 位：SDHOST_IDSTS_AIS)。对相应的位写 1，可以清除中断。当某组中的所有使能中断都被清除，相应的汇总位将被清零。

中断不排序，如果中断在驱动程序响应之前发生，则不会产生其他中断。例如，IDSTS_RI 表示一个或多个数据已发送至主机缓存。

并发事件只会产生一个中断。驱动程序须扫描寄存器 SDHOST_RINTSTS_REG 和 SDHOST_IDSTS_REG 来查找导致中断的原因。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

57.12 寄存器列表

本小节的所有地址均为相对于 SD/MMC 主机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
SDHOST_CTRL_REG	控制寄存器	0x0000	R/W
SDHOST_CLKDIV_REG	时钟分频器配置寄存器	0x0008	R/W
SDHOST_CLKSRC_REG	时钟源选择寄存器	0x000C	R/W
SDHOST_CLKENA_REG	时钟使能寄存器	0x0010	R/W
SDHOST_TMOUT_REG	数据和响应超时寄存器	0x0014	R/W
SDHOST_CTYPE_REG	卡总线宽度配置寄存器	0x0018	R/W
SDHOST_BLKSIZ_REG	卡数据块大小配置寄存器	0x001C	R/W
SDHOST_BYTCNT_REG	数据传输长度配置寄存器	0x0020	R/W
SDHOST_CMDARG_REG	命令参数数据寄存器	0x0028	R/W
SDHOST_CMD_REG	命令和启动配置寄存器	0x002C	R/W
SDHOST_FIFOTH_REG	FIFO 配置寄存器	0x004C	R/W
SDHOST_DEBNCE_REG	防抖滤波器时间配置寄存器	0x0064	R/W
SDHOST_USRID_REG	用户 ID（暂存）寄存器	0x0068	R/W
SDHOST_VERID_REG	版本 ID（暂存）寄存器	0x006C	RO
SDHOST_HCON_REG	硬件特性寄存器	0x0070	RO
SDHOST_UHS_REG	UHS-1 寄存器	0x0074	R/W
SDHOST_RST_N_REG	卡复位寄存器	0x0078	R/W
SDHOST_BMOD_REG	突发传输配置寄存器	0x0080	R/W
SDHOST_PLDMND_REG	轮询需求配置寄存器	0x0084	WO
SDHOST_DBADDR_REG	链表基地址寄存器	0x0088	R/W
SDHOST_CARDTHRCTL_REG	卡阈值控制寄存器	0x0100	R/W
SDHOST_EMMCDDR_REG	eMMC 双倍数据速率寄存器	0x010C	R/W
SDHOST_ENSHIFT_REG	相位偏移使能寄存器	0x0110	R/W
SDHOST_CLK_EDGE_SEL_REG	SDIO 控制寄存器	0x0800	R/W
SDHOST_DLL_CLK_CONF_REG	SDIO DLL 时钟控制寄存器	0x0808	R/W
中断寄存器			
SDHOST_INTMASK_REG	SDIO 中断屏蔽寄存器	0x0024	R/W
SDHOST_MINTSTS_REG	屏蔽中断状态寄存器	0x0040	RO
SDHOST_RINTSTS_REG	原始中断寄存器	0x0044	R/W1C
SDHOST_IDSTS_REG	DMA 中断寄存器	0x008C	R/W1C
SDHOST_IDINTEN_REG	DMA 中断使能寄存器	0x0090	R/W
状态寄存器			
SDHOST_RESP0_REG	响应数据寄存器	0x0030	RO
SDHOST_RESP1_REG	超长响应数据寄存器	0x0034	RO
SDHOST_RESP2_REG	超长响应数据寄存器	0x0038	RO
SDHOST_RESP3_REG	超长响应数据寄存器	0x003C	RO

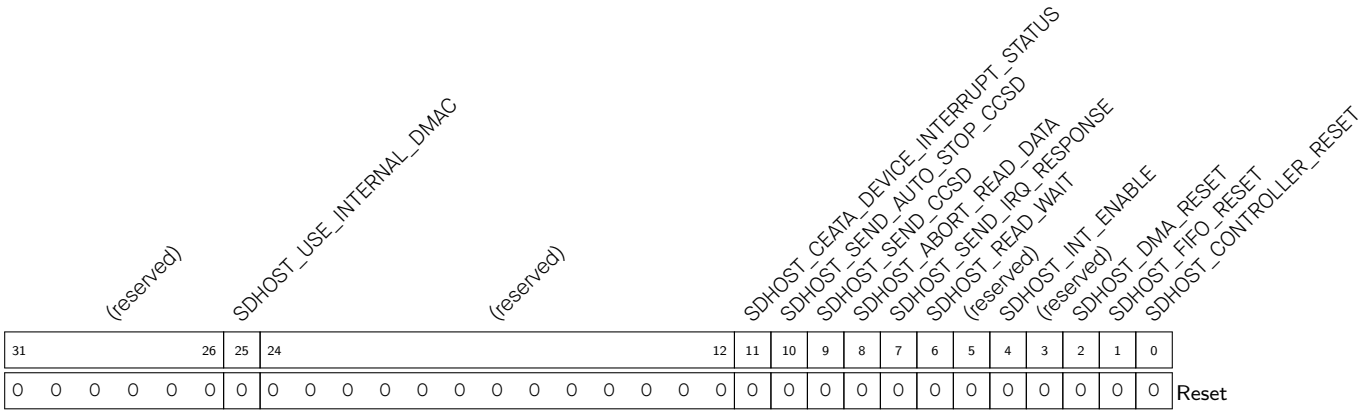
名称	描述	地址	访问
SDHOST_STATUS_REG	SD/MMC 状态寄存器	0x0048	RO
SDHOST_CDETECT_REG	卡检测寄存器	0x0050	RO
SDHOST_WRTPRT_REG	卡写保护状态寄存器	0x0054	RO
SDHOST_TCBCNT_REG	传输字节计数寄存器	0x005C	RO
SDHOST_TBBCNT_REG	传输字节计数寄存器	0x0060	RO
SDHOST_DSCADDR_REG	主机描述符地址指针寄存器	0x0094	RO
SDHOST_BUFADDR_REG	主机缓存地址指针寄存器	0x0098	RO
发送和接收数据寄存器			
SDHOST_BUFFIFO_REG	软件读写 FIFO 传输数据寄存器	0x0200	R/W

57.13 寄存器

本小节的所有地址均为相对于 SD/MMC 主机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 57.1. SDHOST_CTRL_REG (0x0000)



SDHOST_CONTROLLER_RESET 配置是否复位控制器。此位在控制器复位后自动清除。

0：无效

1：复位

(R/W)

SDHOST_FIFO_RESET 配置是否复位 FIFO。此位在 FIFO 复位后自动清除。

0：无效

1：复位

(R/W)

SDHOST_DMA_RESET 配置是否复位 DMA 接口。此位在 DMA 复位后自动清除。

0：无效

1：复位

(R/W)

SDHOST_INT_ENABLE 写入 1 使能全局中断端口。(R/W)

SDHOST_READ_WAIT 配置是否向 SDIO 卡发送读取等待。

0：清除读取等待

1：生成读取等待

(R/W)

见下页...

Register 57.1. SDHOST_CTRL_REG (0x0000)

接上页...

SDHOST_SEND_IRQ_RESPONSE 配置是否发送自动中断请求 (IRQ) 响应。此位在响应发送后自动清除。

主机等待 MMC 卡中断时，会发出 CMD40 命令，并等待来自 MMC 卡的中断响应。如果主机希望 SD/MMC 退出等待中断状态，可以将此位置 1，此时 SD/MMC 命令状态机在总线上发送 CMD40 响应并返回到空闲状态。

0：无效

1：发送自动 IRQ 响应

(R/W)

SDHOST_ABORT_READ_DATA 配置是否中止读取数据。此位在数据状态机重置为空闲后自动清除。在读取操作期间发出暂停命令后，软件轮询卡获取暂停事件发生的时间。暂停事件发生后，软件将设置该位，该位将重置正在等待下一个数据块的数据状态机。

0：无效

1：中止读取数据

(R/W)

SDHOST_SEND_CCSD 配置是否向 CE-ATA 设备发送命令完成信号禁用 (CCSD)。CCSD 发送给设备后，SD/MMC 会自动清除 SDHOST_SEND_CCSD 位，并将 SDHOST_RINTSTS_REG 寄存器中的命令完成 (CD) 位置 1。

只有在当前命令期待 CCS（即，RW_BLK），并且为 CE-ATA 设备使能了中断时，软件才会设置此位。

0: 无效

1：向 CE-ATA 设备发送 CCSD

(R/W)

SDHOST_SEND_AUTO_STOP_CCSD 配置是否向 CE-ATA 设备发送内部生成的 STOP 命令 (CMD12)。发送 CCSD 后，SD/MMC 会自动清除 SDHOST_SEND_AUTO_STOP_CCSD 位。发送此内部生成的 STOP 命令后，SDHOST_RINTSTS_REG 中的自动命令完成 (ACD) 位会被置 1。

SDHOST_SEND_AUTO_STOP_CCSD 和 SDHOST_SEND_CCSD 位需一起配置。

0: 无效

1：向 CE-ATA 设备发送内部生成的 STOP 命令 (CMD12)

(R/W)

SDHOST_CEATA_DEVICE_INTERRUPT_STATUS 写入 1 以在 CE-ATA 设备中使能中断（ATE 控制寄存器中的 nIEN = 0）。

在电源复位或 CE-ATA 设备的任何其他复位后，软件应适当地写入此位。复位后，CE-ATA 设备的中断通常被禁用（nIEN = 1）。如果主机使能了 CE-ATA 设备的中断，那么软件应设置此位。(R/W)

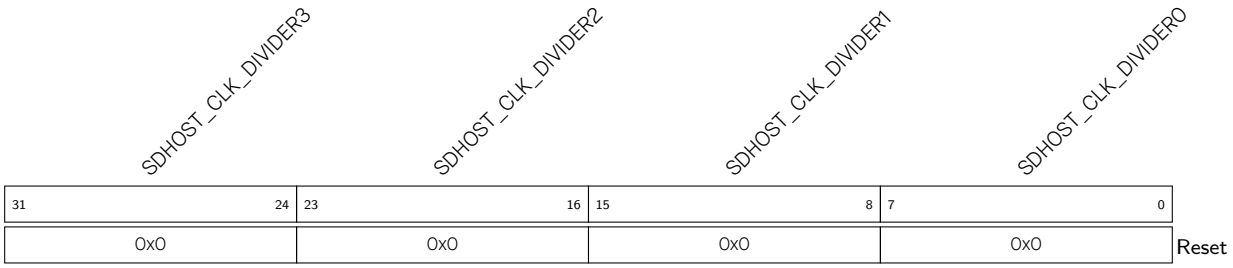
SDHOST_USE_INTERNAL_DMAC 配置是否使用内部 DMA 进行数据传输。

0：不使用

1：使用

(R/W)

Register 57.2. SDHOST_CLKDIV_REG (0x0008)



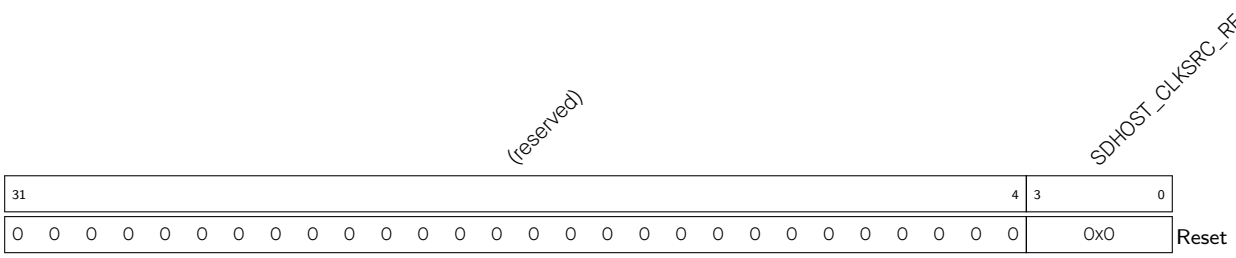
SDHOST_CLK_DIVIDER0 配置时钟分频器 0 分频系数的二分之一。该字段为 0 时分频系数为 1。(R/W)

SDHOST_CLK_DIVIDER1 配置时钟分频器 1 分频系数的二分之一。该字段为 0 时分频系数为 1。(R/W)

SDHOST_CLK_DIVIDER2 配置时钟分频器 2 分频系数的二分之一。该字段为 0 时分频系数为 1。(R/W)

SDHOST_CLK_DIVIDER3 配置时钟分频器 3 分频系数的二分之一。该字段为 0 时分频系数为 1。(R/W)

Register 57.3. SDHOST_CLKSRC_REG (0x000C)



SDHOST_CLKSRC_REG 配置两个 SD 卡输出到 sdhost_cclk_out[1:0] 信号的时钟分频器源。每张卡两位，位 [1:0] 对应卡 0，位 [3:2] 对应卡 1。:

0：时钟分频器 0

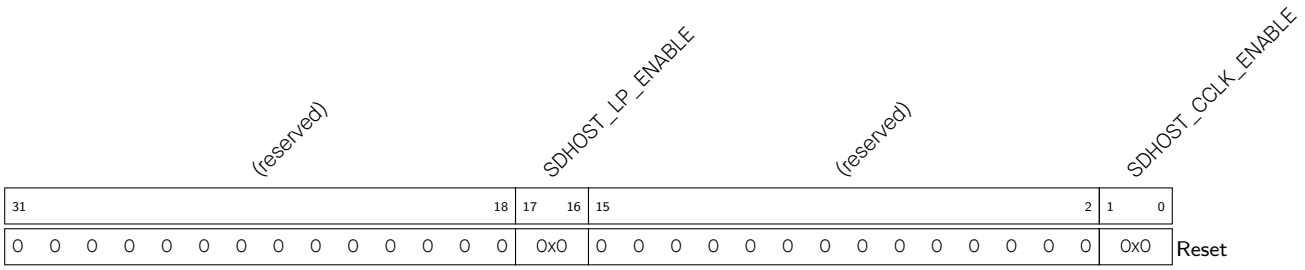
1：时钟分频器 1

2：时钟分频器 2

3：时钟分频器 3

(R/W)

Register 57.4. SDHOST_CLKENA_REG (0x0010)



SDHOST_CCLK_ENABLE 配置是否使能两个 SD 卡时钟和一个 MMC 卡时钟。每张卡一位，位 0 对应卡 0。

0：时钟禁用

1：时钟使能

(R/W)

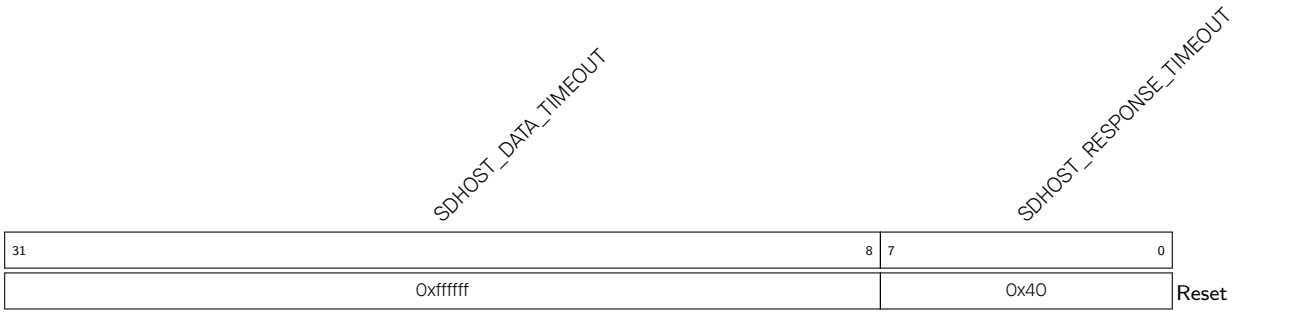
SDHOST_LP_ENABLE 配置卡处于空闲状态时是否禁用时钟。每张卡一位，位 0 对应卡 0。

0：时钟禁用

1：时钟使能

(R/W)

Register 57.5. SDHOST_TMOUT_REG (0x0014)



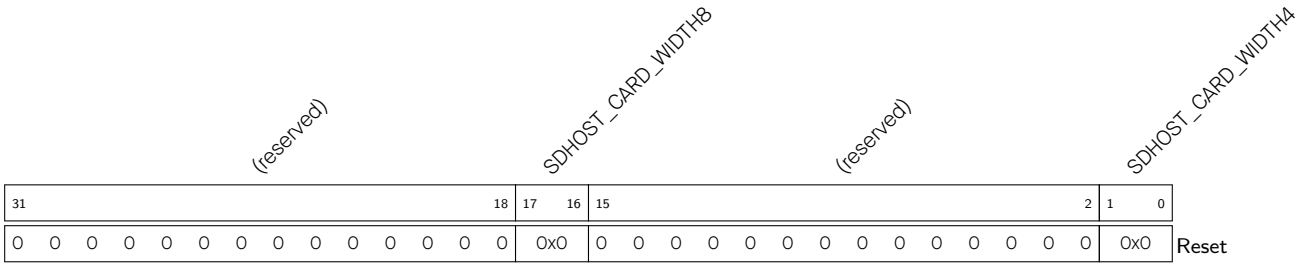
SDHOST_RESPONSE_TIMEOUT 配置响应超时时间。单位：卡输出时钟周期。(R/W)

SDHOST_DATA_TIMEOUT 配置卡数据读取超时时间和主机填充数据超时时间。超时计数器在卡时钟停止后才开始计数。单位：卡输出时钟周期。

注意：如果超时时间约为 100 毫秒，需使用软件定时器。此时需要禁用读取数据超时中断 (DRTO)。

(R/W)

Register 57.6. SDHOST_CTYPE_REG (0x0018)



SDHOST_CARD_WIDTH4 配置卡是否为 1 位模式或 4 位模式。每张卡一位，位 0 对应卡 0。

0：1 位模式

1：4 位模式

(R/W)

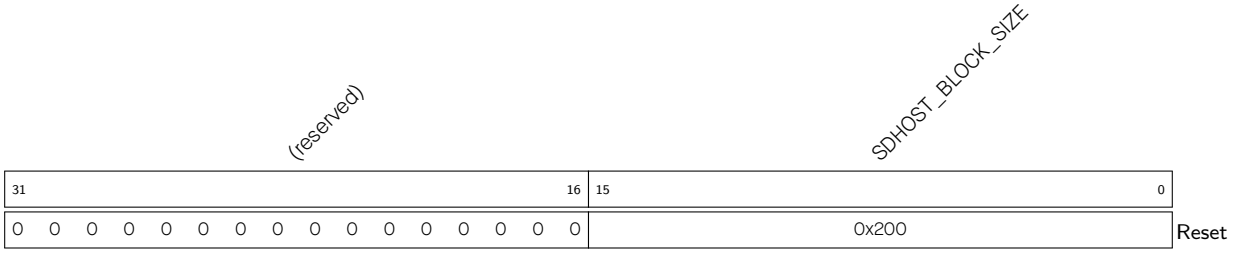
SDHOST_CARD_WIDTH8 配置卡是否为 8 位模式。每张卡一位，位 0 对应卡 0。

0：非 8 位模式

1：8 位模式

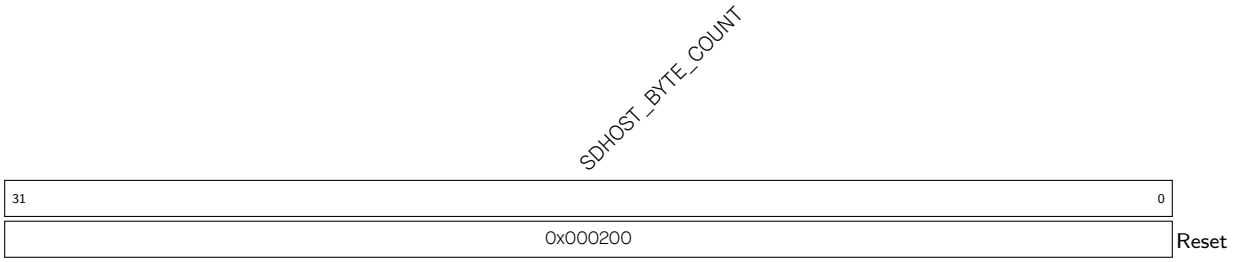
(R/W)

Register 57.7. SDHOST_BLKSIZ_REG (0x001C)



SDHOST_BLOCK_SIZE 配置块大小。(R/W)

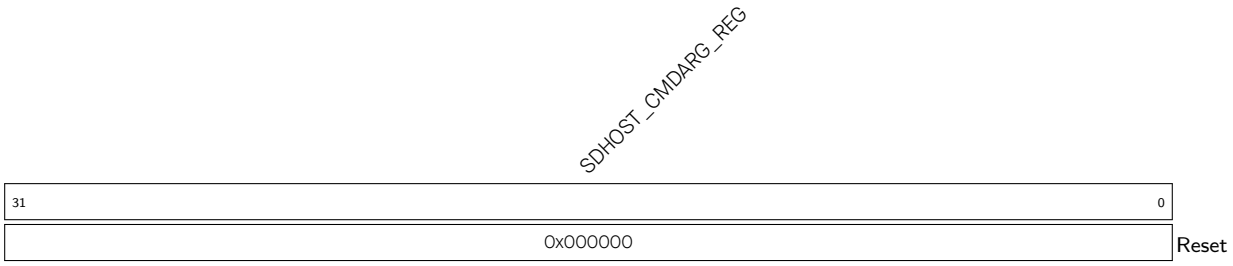
Register 57.8. SDHOST_BYTCNT_REG (0x0020)



SDHOST_BYTE_COUNT 配置待传输的字节数，应为块传输的块大小的整数倍。

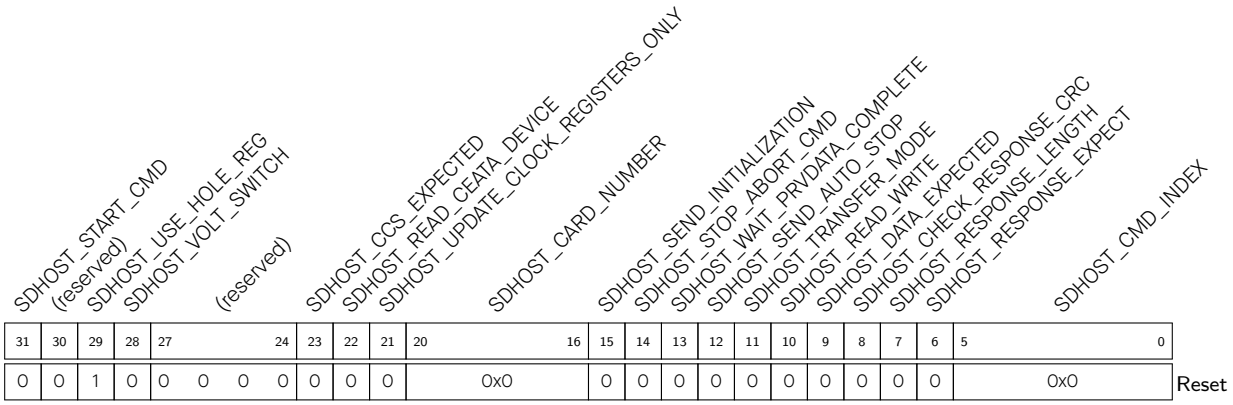
对于未定义字节长度的数据传输，该字段应设置为 0。此时，主机需发送停止/中止命令终止数据传输。(R/W)

Register 57.9. SDHOST_CMDARG_REG (0x0028)



SDHOST_CMDARG_REG 配置要传递给卡的命令参数。(R/W)

Register 57.10. SDHOST_CMD_REG (0x002C)



SDHOST_CMD_INDEX 配置发送到卡的命令索引编号。(R/W)

SDHOST_RESPONSE_EXPECT 配置是否需要卡响应。

0：不需要卡响应

1：需要卡响应

(R/W)

SDHOST_RESPONSE_LENGTH 配置卡的响应长度。

0：短响应

1：长响应

(R/W)

SDHOST_CHECK_RESPONSE_CRC 配置是否检查响应 CRC。

0：不检查

1：检查响应 CRC

(R/W)

SDHOST_DATA_EXPECTED 配置是否需要传输数据。

0：不需要数据传输

1：需要数据传输

(R/W)

SDHOST_READ_WRITE 配置数据传输方向。

0：从卡读取

1：写入卡

(R/W)

SDHOST_TRANSFER_MODE 配置数据传输模式。

0：传输数据块

1：传输数据流

(R/W)

见下页...

Register 57.10. SDHOST_CMD_REG (0x002C)

接上页...

SDHOST_SEND_AUTO_STOP 配置数据传输结束时是否自动发送停止命令。

0：不发送

1：发送

(R/W)

SDHOST_WAIT_PRVDATA_COMPLETE 配置是否立即发送命令。

0：即使前一个数据传输尚未完成，也立即发送命令

1：在发送命令之前，等待前一个数据传输完成

(R/W)

SDHOST_STOP_ABORT_CMD 配置停止/中止命令是否可以停止数据传输。

0：停止或中止命令都不能停止当前的数据传输。如果当前选择的卡设备，之前已向它发送过中止命令，或它不在数据传输模式中，则此位应设置为 0

1：停止或中止命令旨在停止当前正在进行的数据传输

(R/W)

SDHOST_SEND_INITIALIZATION 配置是否在发送此命令之前发送初始化序列（80 个时钟周期的 1），以便控制器在向卡发送命令之前初始化时钟。上电后，必须先向卡发送初始化序列，然后才能发送命令。所以向卡发送第一个命令时需要置位该字段。

0：不发送初始化序列

1：发送初始化序列

(R/W)

SDHOST_CARD_NUMBER 配置正在使用的卡号和正在访问的卡的物理插槽号。在 SD-only 模式下，支持最多两张卡。(R/W)

SDHOST_UPDATE_CLOCK_REGISTERS_ONLY 配置是否仅将 SDHOST_CLKDIV_REG、SDHOST_CLRSRC_REG 和 SDHOST_CLKENA_REG 时钟寄存器的值更新到卡时钟域。

0：正常发送命令序列

1：不发送命令，只更新时钟寄存器值到卡时钟域

该位可以确保更改时钟频率或停止时钟时无需向卡发送命令，也不会触发命令完成中断。(R/W)

SDHOST_READ_CEATA_DEVICE 配置主机是否在读取 CE-ATA 设备，用于开启或禁用读取超时指示。

0：主机未读取（RW_REG 或 RW_BLK）CE-ATA 设备，开启数据读取超时指示

1：主机正在读取（RW_REG 或 RW_BLK）向 CE-ATA 设备，禁用数据读取超时指示

(R/W)

见下页...

Register 57.10. SDHOST_CMD_REG (0x002C)

接上页...

SDHOST_CCS_EXPECTED 配置是否需要命令完成信号 (CCS)。

0：CE-ATA 设备中未使能中断（ATA 控制寄存器中的 nIEN = 1），或命令不需要 CE-ATA 设备发送 CCS

1：CE-ATA 设备中使能了中断（nIEN = 0），并且 RW_BLK 命令需要 CE-ATA 设备发送 CCS

(R/W)

SDHOST_VOLT_SWITCH 配置是否使能电压切换。

0：没有电压切换

1：使能电压切换。

CMD11 中此位必须为 1。(R/W)

SDHOST_USE_HOLE_REG 配置是否使用保持寄存器 (hold register)。

0：CMD 和 DATA 绕过保持寄存器发送到卡

1：CMD 和 DATA 通过保持寄存器发送到卡

(R/W)

SDHOST_START_CMD 配置是否开始命令。

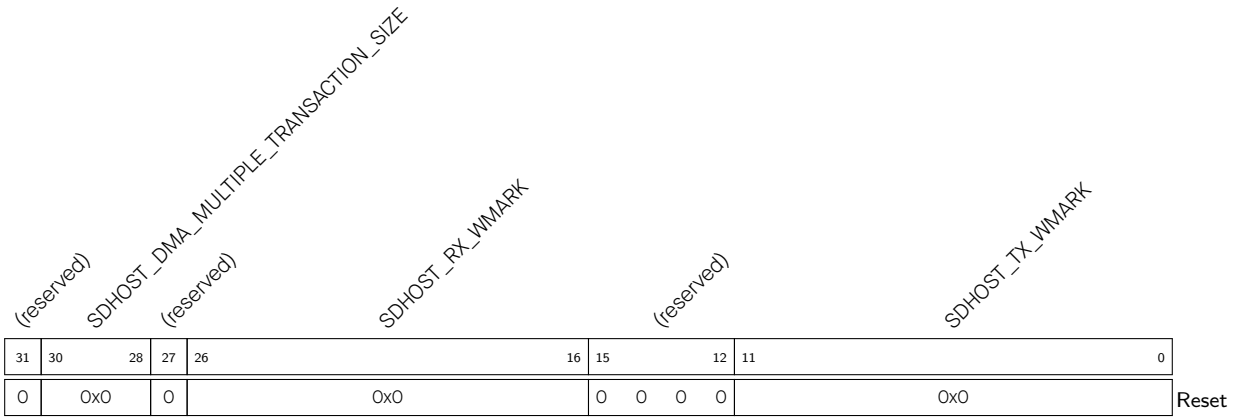
0：不开始

1：开始

一旦命令由 CIU 服务，此位将自动清除。

此位为 1 时，主机不能尝试写入任何命令寄存器，否则会触发硬件锁定错误。(R/W/SC)

Register 57.11. SDHOST_FIFOTH_REG (0x004C)



SDHOST_TX_WMARK 配置向卡传输数据时的 FIFO 阈值。当 FIFO 数据计数小于或等于此数字时，会产生 DMA/FIFO 请求。在数据包结束时，无论阈值编程如何，都会生成请求或中断。

在非 DMA 模式下，当使能发送 FIFO 阈值 (TXDR) 中断时，会生成中断而不是 DMA 请求。在数据包结束时，最后一次中断时，主机负责只填充所需的剩余字节（不在 FIFO 满或 CIU 完成数据传输后，因为 FIFO 可能不为空）。

在 DMA 模式下，如果最后一次传输小于突发大小，DMA 控制器会进行单周期传输，直到传输所需的字节。(R/W)

SDHOST_RX_WMARK 配置从卡接收数据时的 FIFO 阈值。当 FIFO 数据计数达到大于此数字时，会产生 DMA/FIFO 请求。在数据包结束时，无论阈值编程如何，都会生成请求，以完成任何剩余的数据。

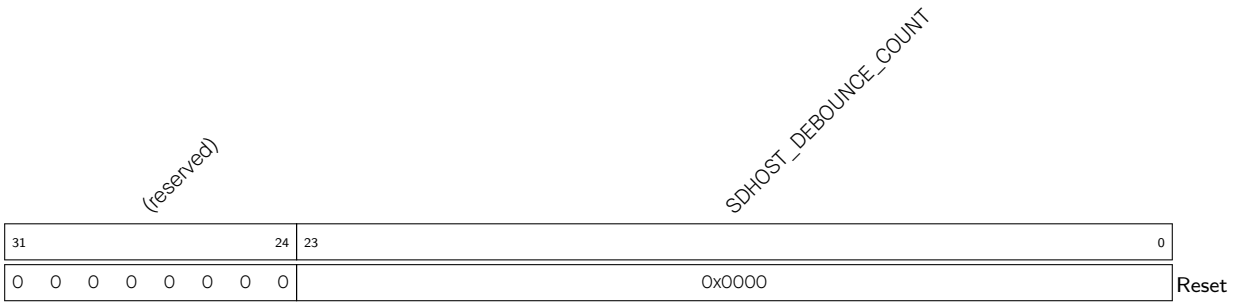
在非 DMA 模式下，当使能接收器 FIFO 阈值 (RXDR) 中断时，会生成中断而不是 DMA 请求。在数据包结束时，如果阈值编程大于任何剩余的数据，中断不会生成。主机有责任在看到数据传输完成中断时读取剩余的字节。

在 DMA 模式下，即使剩余的字节少于阈值，DMA 请求也会进行单次传输，以在设置数据传输完成中断之前清除任何剩余的字节。(R/W)

SDHOST_DMA_MULTIPLE_TRANSACTION_SIZE 配置多次突发传输的大小，应与 DMA 控制器的多次传输大小 SDHOST_SRC/DEST_MSIZE 编程相同。

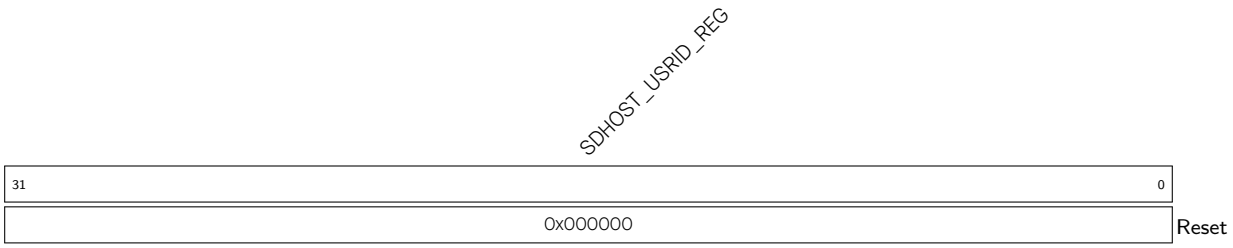
0：1 字节传输
1：4 字节传输
2：8 字节传输
3：16 字节传输
4：32 字节传输
5：64 字节传输
6：128 字节传输
7：256 字节传输
(R/W)

Register 57.12. SDHOST_DEBNCE_REG (0x0064)



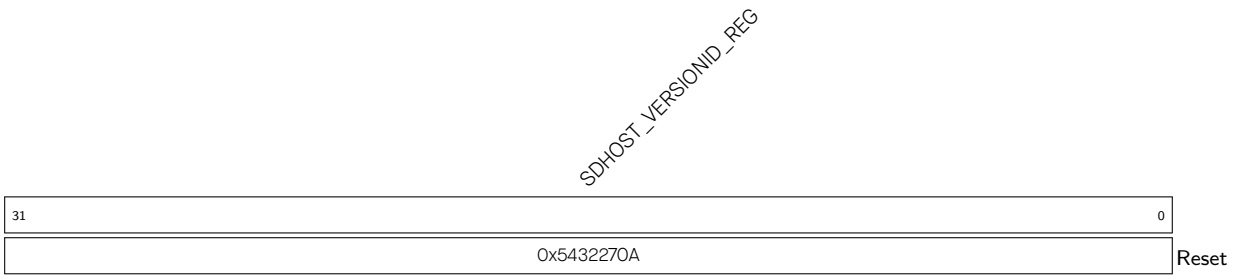
SDHOST_DEBOUNCE_COUNT 配置防抖滤波器逻辑使用的主机时钟数量。典型的防抖时间是 5 ~ 25 毫秒，防止卡插入或移除时不稳定。(R/W)

Register 57.13. SDHOST_USRID_REG (0x0068)



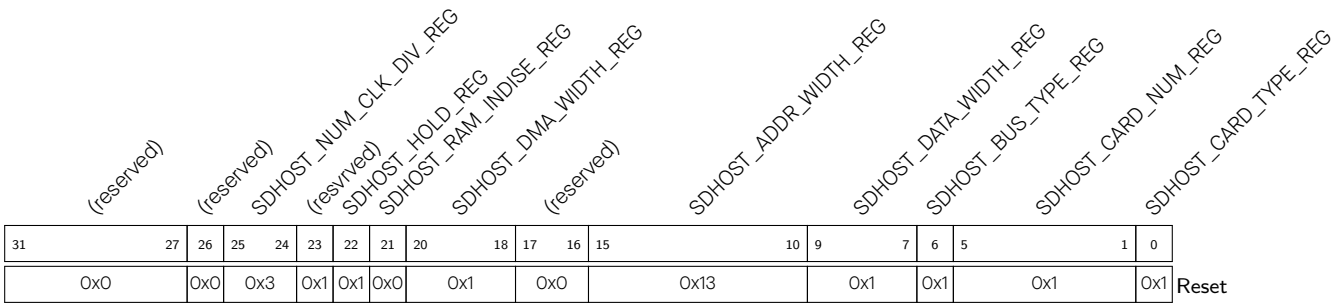
SDHOST_USRID_REG 配置用户身份标识，值由用户配置。用户也可将该寄存器用作暂存寄存器。(R/W)

Register 57.14. SDHOST_VERID_REG (0x006C)



SDHOST_VERSIONID_REG 表示硬件版本。固件也可以读取该寄存器。(RO)

Register 57.15. SDHOST_HCON_REG (0x0070)



SDHOST_NUM_CLK_DIV_REG 表示设计中有 4 个时钟分频器。(RO)

SDHOST_HOLD_REG 表示数据路径中有一个保持寄存器。(RO)

SDHOST_RAM_INDISE_REG 表示 SDMMC 模块内部包含 RAM。(RO)

SDHOST_DMA_WIDTH_REG 表示 DMA 数据宽度为 32。(RO)

SDHOST_ADDR_WIDTH_REG 表示寄存器地址宽度为 32。(RO)

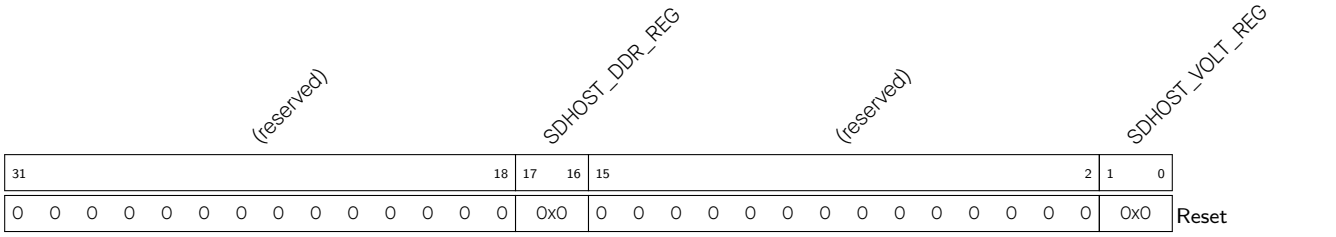
SDHOST_DATA_WIDTH_REG 表示寄存器数据宽度为 32。(RO)

SDHOST_BUS_TYPE_REG 表示寄存器配置为 APB 总线。(RO)

SDHOST_CARD_NUM_REG 表示支持的卡数量为 2。(RO)

SDHOST_CARD_TYPE_REG 表示硬件支持 SDIO 和 MMC。(RO)

Register 57.16. SDHOST_UHS_REG (0x0074)



SDHOST_VOLT_REG 配置卡是否使用高电压模式。每张卡一位，位 0 对应卡 0。

0：电压 3.3 V

1：电压 1.8 V

(R/W)

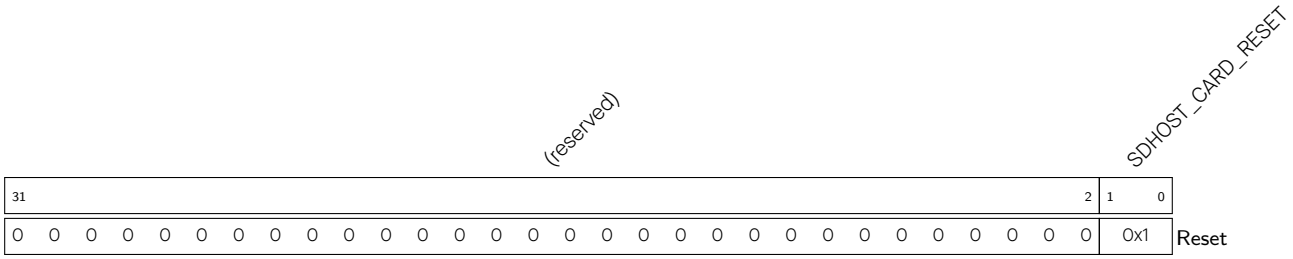
SDHOST_DDR_REG 配置卡是否使用 DDR（双倍数据速率）模式。每张卡一位，位 0 对应卡 0。

0：非 DDR 模式

1：DDR 模式

(R/W)

Register 57.17. SDHOST_RST_N_REG (0x0078)



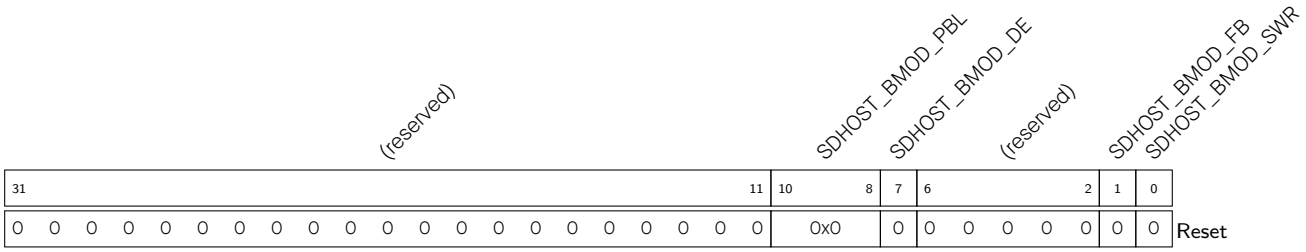
SDHOST_CARD_RESET 配置卡的硬件复位。每张卡一位，位 0 对应卡 0。卡复位后进入预空闲状态，需要重新初始化。

0：复位

1：活跃模式

(R/W)

Register 57.18. SDHOST_BMOD_REG (0x0080)



SDHOST_BMOD_SWR 配置是否复位所有 DMA 内部寄存器。该位在一个时钟周期后自动清除。

0：无效

1：重置

(R/W)

SDHOST_BMOD_FB 配置 AHB 主接口是否用于固定长度突发传输。

0：使用单次和 INCR（未指定长度）突发传输操作

1：仅使用单次、INCR4、INCR8 或 INCR16 突发传输

(R/W)

SDHOST_BMOD_DE 配置是否使能 DMA。

0：不使能

1：使能

(R/W)

SDHOST_BMOD_PBL 配置一次 DMA 传输的最大传输长度。

0：1 拍传输

1：4 拍传输

2：8 拍传输

3：16 拍传输

4：32 拍传输

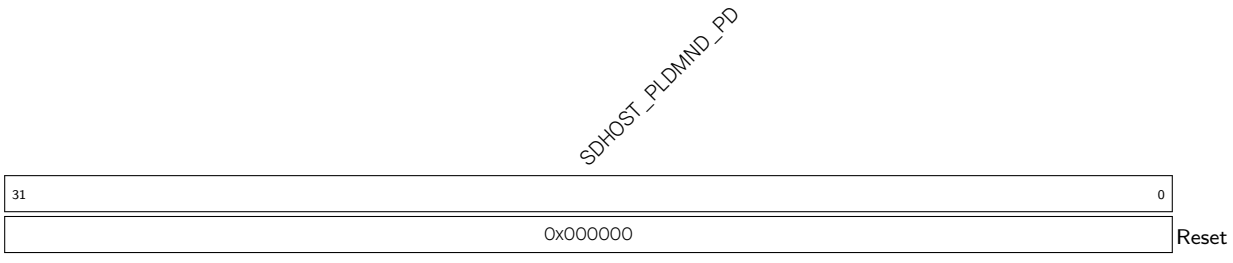
5：64 拍传输

6：128 拍传输

7：256 拍传输

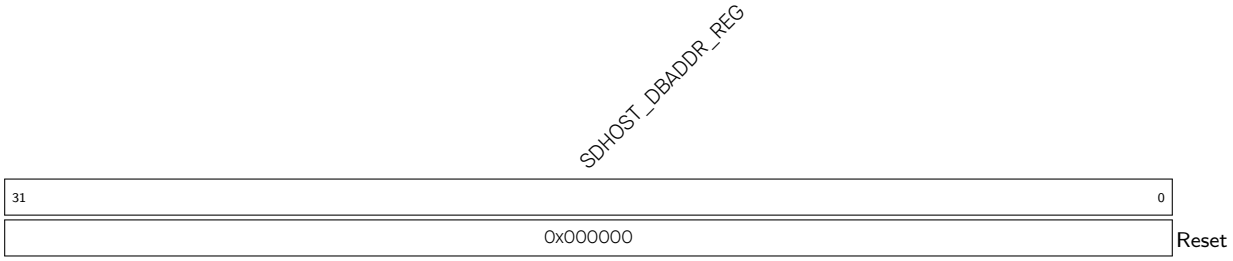
(R/W)

Register 57.19. SDHOST_PLDMND_REG (0x0084)



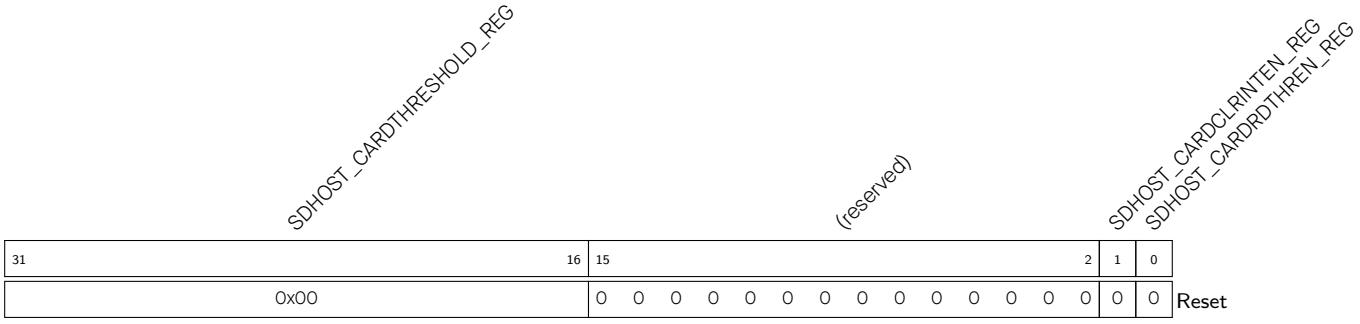
SDHOST_PLDMND_PD 配置 DMA FSM 是否恢复获取描述符。如果描述符的 OWNER 位未置 1，FSM 将进入挂起状态，软件需要向此字段写入任何值以使 FSM 恢复获取描述符。(WO)

Register 57.20. SDHOST_DBADDR_REG (0x0088)



SDHOST_DBADDR_REG 配置链表第一个描述符的基地址。位 [1:0] 被内部 DMA 内部忽略并视为全零。(R/W)

Register 57.21. SDHOST_CARDTHRCTL_REG (0x0100)



SDHOST_CARDRDTHREN_REG 配置是否使能卡读取阈值。

0：不使能

1：使能

(R/W)

SDHOST_CARDCLRINTEN_REG 配置是否使能忙碌清除中断生成。

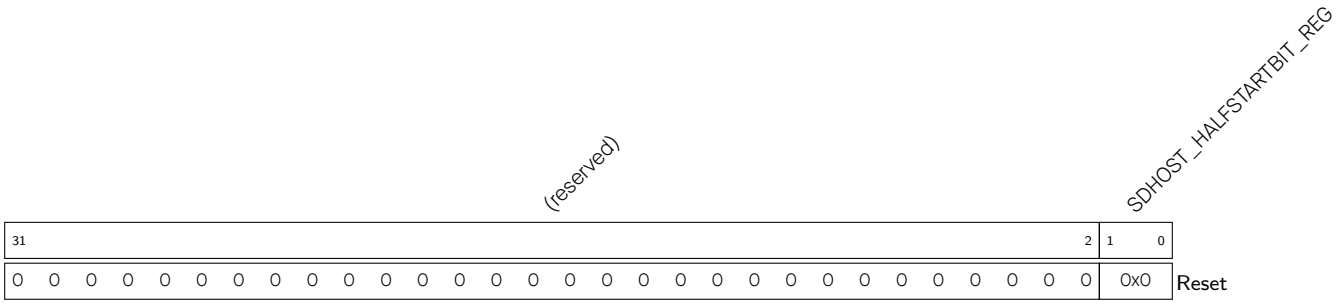
0：不使能

1：使能

(R/W)

SDHOST_CARDTHRESHOLD_REG 配置卡读取阈值。单位：字节。仅在 SDHOST_CARDRDTHREN_REG 为 1 时有效，需小于内部 FIFO 大小 512。(R/W)

Register 57.22. SDHOST_EMMCDDR_REG (0x010C)



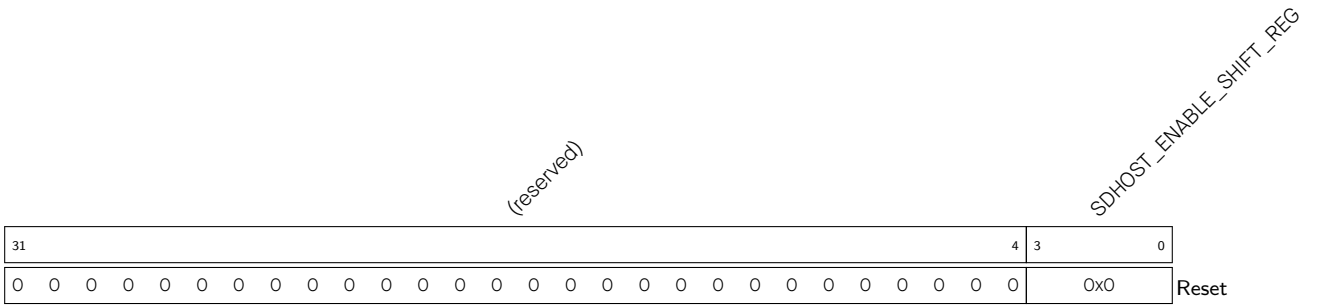
SDHOST_HALFSTARTBIT_REG 配置开始位检测机制的开始位检测长度。每张卡一位，位 0 对应卡 0。eMMC4.5 及以上版本此位需为 1，SD 应用此位需为 0。对于 eMMC4.5，起始位检测模式可配置为：

0：完整周期

1：少于一个完整周期

(R/W)

Register 57.23. SDHOST_ENSHIFT_REG (0x0110)



SDHOST_ENABLE_SHIFT_REG 配置设计中默认的相位偏移量。每张卡 2 位，位 1:0 对应卡 0。

0x0：默认相位偏移

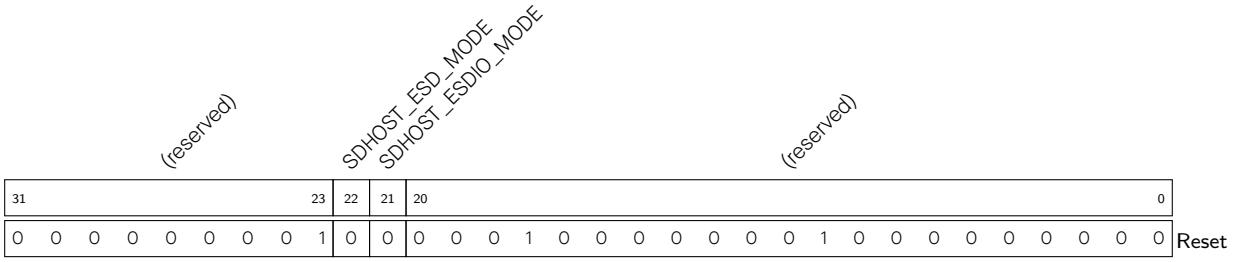
0x1：偏移到下个上升沿

0x2：偏移到下个下降沿

0x3：保留

(R/W)

Register 57.24. SDHOST_CLK_EDGE_SEL_REG (0x0800)



SDHOST_ESDIO_MODE 配置是否使能 eSDIO 模式。

0：不使能

1：使能

(R/W)

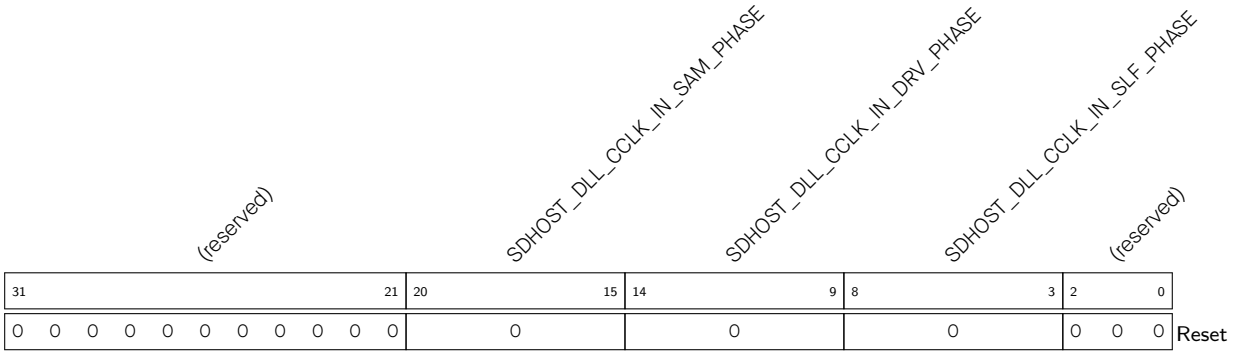
SDHOST_ESD_MODE 配置是否使能 eSD 模式。

0：不使能

1：使能

(R/W)

Register 57.25. SDHOST_DLL_CLK_CONF_REG (0x0808)

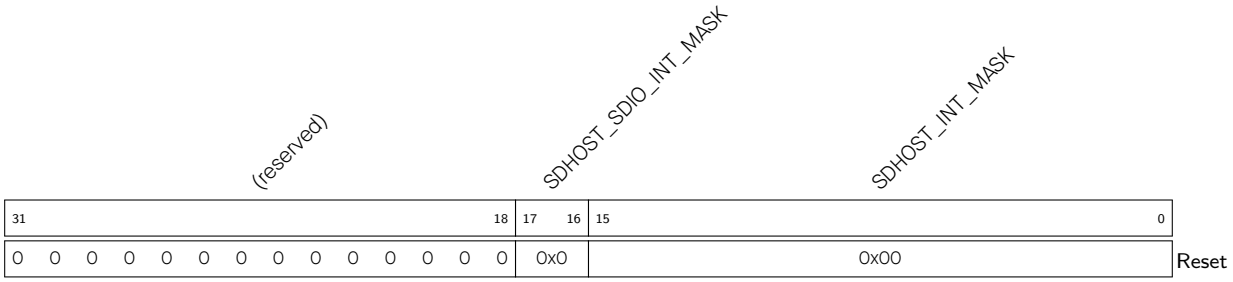


SDHOST_DLL_CCLK_IN_SLF_PHASE 配置选择高频时钟时内部信号的时钟相位。单位是时钟周期的 1/64。(R/W)

SDHOST_DLL_CCLK_IN_DRV_PHASE 配置选择高频时钟时输出信号的时钟相位。单位是时钟周期的 1/64。(R/W)

SDHOST_DLL_CCLK_IN_SAM_PHASE 配置选择高频时钟时输入信号的时钟相位。单位是时钟周期的 1/64。(R/W)

Register 57.26. SDHOST_INTMASK_REG (0x0024)

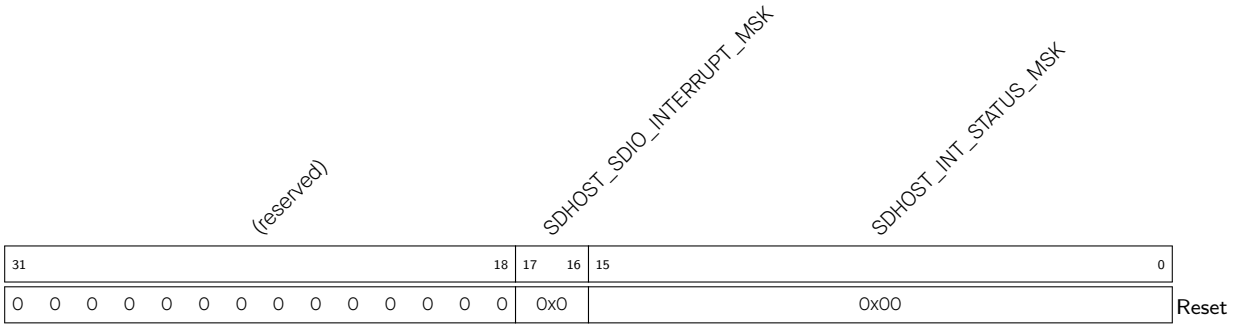


SDHOST_INT_MASK 写 1 到对应位使能对应中断。

位 15 (EBE)：结束位错误/无 CRC 错误中断
位 14 (ACD)：自动命令结束中断
位 13 (SBE/BCI)：接收起始位错误中断
位 12 (HLE)：硬件锁定写入错误中断
位 11 (FRUN)：FIFO 空/满错误中断
位 10 (HTO)：主机填充数据超时中断
位 9 (DRTO)：数据读取超时中断
位 8 (RTO)：卡响应超时中断
位 7 (DCRC)：数据 CRC 错误中断
位 6 (RCRC)：卡响应 CRC 错误中断
位 5 (RXDR)：接收 FIFO 数据请求中断
位 4 (TXDR)：发送 FIFO 数据请求中断
位 3 (DTO)：数据传输结束中断
位 2 (CMDD)：命令执行完毕中断
位 1 (RE)：卡响应错误中断
位 0 (CD)：检测到卡中断
(R/W)

SDHOST_SDIO_INT_MASK 写 1 到对应位使能来自对应卡的中断。每张卡一位，位 0 对应卡 0。(R/W)

Register 57.27. SDHOST_MINTSTS_REG (0x0040)

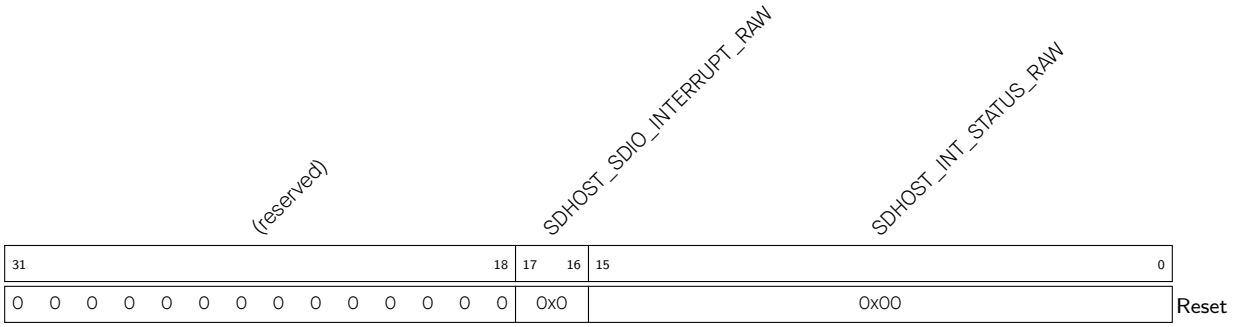


SDHOST_INT_STATUS_MSK 中断的屏蔽状态。每一位对应一个中断：

位 15 (EBE)：结束位错误/无 CRC 错误中断
位 14 (ACD)：自动命令结束中断
位 13 (SBE/BCI)：接收起始位错误中断
位 12 (HLE)：硬件锁定写入错误中断
位 11 (FRUN)：FIFO 空/满错误中断
位 10 (HTO)：主机填充数据超时中断
位 9 (DRTO)：数据读取超时中断
位 8 (RTO)：卡响应超时中断
位 7 (DCRC)：数据 CRC 错误中断
位 6 (RCRC)：卡响应 CRC 错误中断
位 5 (RXDR)：接收 FIFO 数据请求中断
位 4 (TXDR)：发送 FIFO 数据请求中断
位 3 (DTO)：数据传输结束中断
位 2 (CMDD)：命令执行完毕中断
位 1 (RE)：卡响应错误中断
位 0 (CD)：检测到卡中断
(RO)

SDHOST_SDIO_INTERRUPT_MSK 来自 SDIO 卡的中断屏蔽状态。每张卡一位，位 0 对应卡 0。(RO)

Register 57.28. SDHOST_RINTSTS_REG (0x0044)

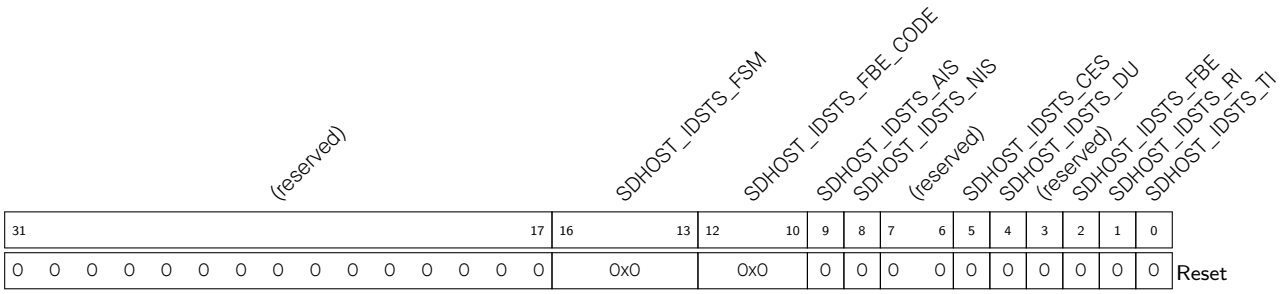


SDHOST_INT_STATUS_RAW 中断的原始状态。每一位对应一个中断：

位 15 (EBE)：结束位错误/无 CRC 错误中断
位 14 (ACD)：自动命令结束中断
位 13 (SBE/BCI)：接收起始位错误中断
位 12 (HLE)：硬件锁定写入错误中断
位 11 (FRUN)：FIFO 空/满错误中断
位 10 (HTO)：主机填充数据超时中断
位 9 (DRTO)：数据读取超时中断
位 8 (RTO)：卡响应超时中断
位 7 (DCRC)：数据 CRC 错误中断
位 6 (RCRC)：卡响应 CRC 错误中断
位 5 (RXDR)：接收 FIFO 数据请求中断
位 4 (TXDR)：发送 FIFO 数据请求中断
位 3 (DTO)：数据传输结束中断
位 2 (CMDD)：命令执行完毕中断
位 1 (RE)：卡响应错误中断
位 0 (CD)：检测到卡中断
(R/W1C)

SDHOST_SDIO_INTERRUPT_RAW 来自 SDIO 卡的中断原始状态。每张卡一位，位 0 对应卡 0。(R/W1C)

Register 57.29. SDHOST_IDSTS_REG (0x008C)



SDHOST_IDSTS_TI IDSTS_TI 中断的原始中断状态。(R/W1C)

SDHOST_IDSTS_RI IDSTS_RI 中断的原始中断状态。(R/W1C)

SDHOST_IDSTS_FBE IDSTS_FBE 中断的原始中断状态。(R/W1C)

SDHOST_IDSTS_DU IDSTS_DU 中断的原始中断状态。(R/W1C)

SDHOST_IDSTS_CES 卡错误汇总中断的原始中断状态，为 SDHOST_RINTSTS_REG 中以下中断原始状态的逻辑或运算结果：
EBE：结束位错误/无 CRC 错误中断
SBE/BCI：接收起始位错误中断
DRTO：数据读取超时中断
RTO：卡响应超时中断
DCRC：数据 CRC 错误中断
RCRC：卡响应 CRC 错误中断
RE：卡响应错误中断
此该位为 1 时，DMA 会在响应错误时中止。(R/W1C)

SDHOST_IDSTS_NIS 正常中断汇总的原始中断状态，为 SDHOST_IDSTS_REG[1:0] 的逻辑或运算结果。这是一个粘性位，软件每次清除导致此字段被设置的相应位时，必须也清除此位。(R/W1C)

SDHOST_IDSTS_AIS 异常中断汇总的原始中断状态，为 SDHOST_IDSTS_REG[2] 和 SDHOST_IDSTS_REG[4] 的逻辑或运算结果。这是一个粘性位，软件每次清除导致此字段被设置的相应位时，必须也清除此位。(R/W1C)

见下页...

Register 57.29. SDHOST_IDSTS_REG (0x008C)

接上页...

SDHOST_IDSTS_FBE_CODE 表示导致总线错误的错误类型，仅在 SDHOST_IDSTS_REG[2] 为 1 时有效。

1：在传输过程中接收到主机中止

2：在接收过程中接收到主机中止

其他值：保留

(RO)

SDHOST_IDSTS_FSM 表示 DMA FSM 当前状态。

0：DMA_IDLE（空闲状态）

1：DMA_SUSPEND（暂停状态）

2：DESC_RD（描述符读取状态）

3：DESC_CHK（描述符检查状态）

4：DMA_RD_REQ_WAIT（等待读取数据请求状态）

5：DMA_WR_REQ_WAIT（等待写入数据请求状态）

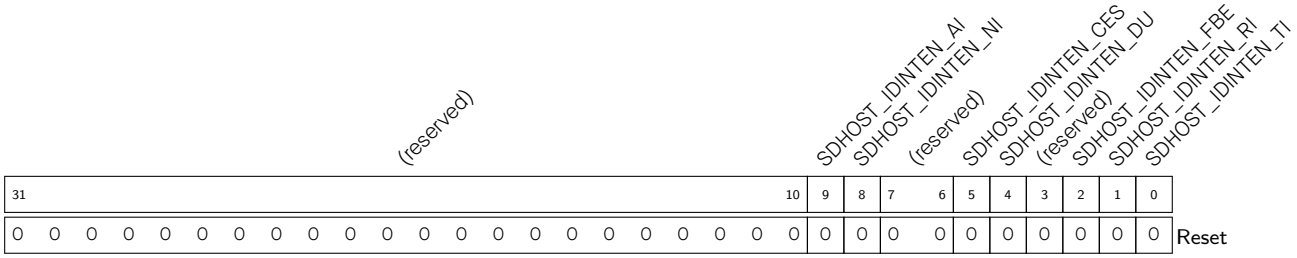
6：DMA_RD（数据读取状态）

7：DMA_WR（数据写入状态）

8：DESC_CLOSE（描述符关闭状态）

(RO)

Register 57.30. SDHOST_IDINTEN_REG (0x0090)



SDHOST_IDINTEN_TI 写入 1 使能 IDSTS_TI 中断。(R/W)

SDHOST_IDINTEN_RI 写入 1 使能 IDSTS_RI 中断。(R/W)

SDHOST_IDINTEN_FBE 写入 1 使能 IDSTS_FBE 中断。(R/W)

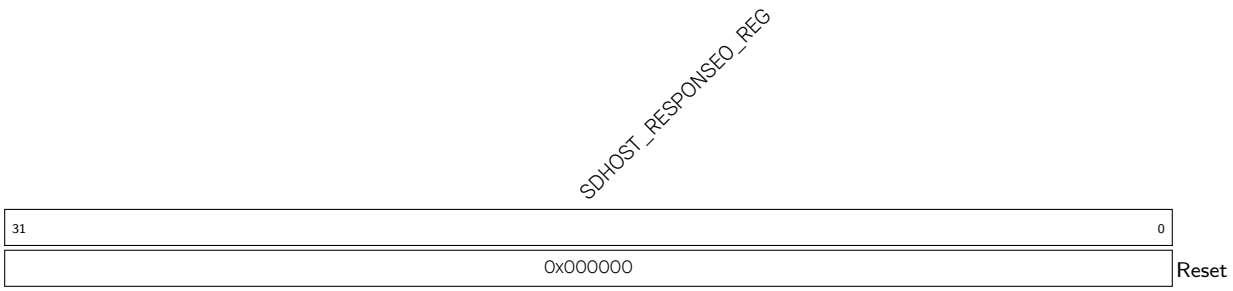
SDHOST_IDINTEN_DU 写入 1 使能 IDSTS_DU 中断。(R/W)

SDHOST_IDINTEN_CES 写入 1 使能卡错误汇总中断。(R/W)

SDHOST_IDINTEN_NI 写入 1 使能正常中断汇总。(R/W)

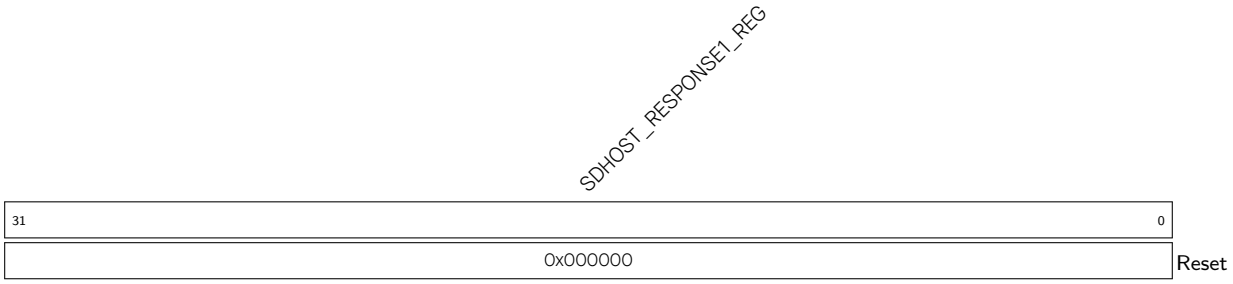
SDHOST_IDINTEN_AI 写入 1 使能异常中断汇总。(R/W)

Register 57.31. SDHOST_RESP0_REG (0x0030)



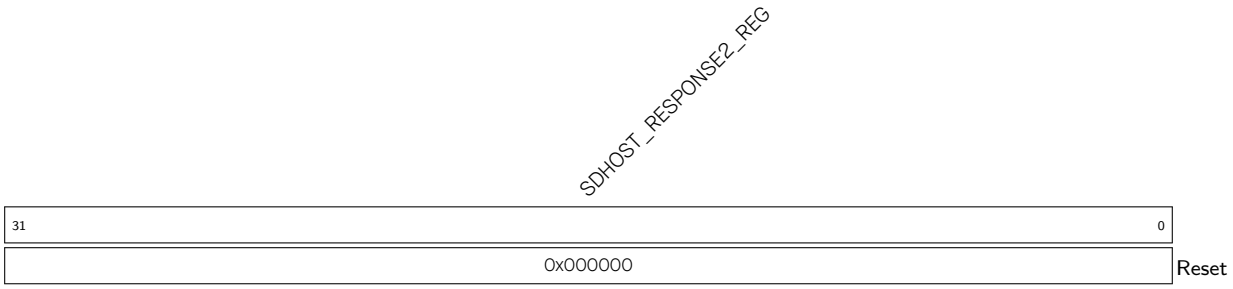
SDHOST_RESPONSE0_REG 表示响应的位 [31:0]。(RO)

Register 57.32. SDHOST_RESP1_REG (0x0034)



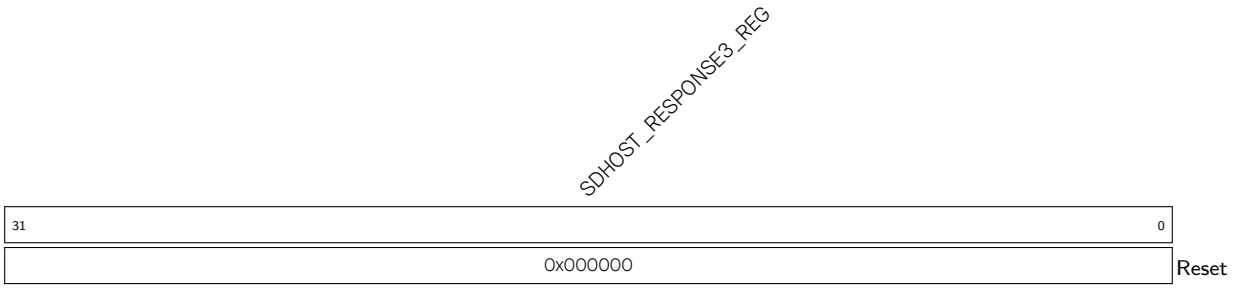
SDHOST_RESPONSE1_REG 表示长响应的位 [63:32]。(RO)

Register 57.33. SDHOST_RESP2_REG (0x0038)



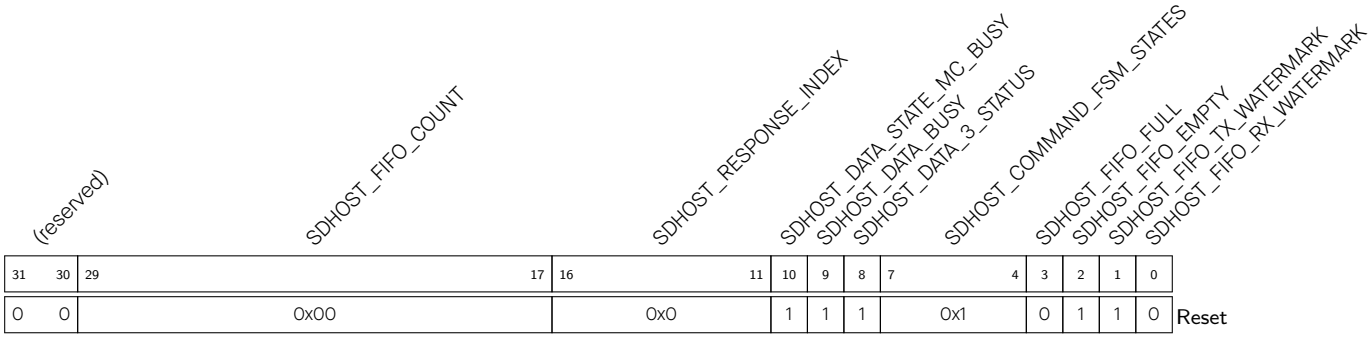
SDHOST_RESPONSE2_REG 表示长响应的位 [95:64]。(RO)

Register 57.34. SDHOST_RESP3_REG (0x003C)



SDHOST_RESPONSE3_REG 表示长响应的位 [127:96]。(RO)

Register 57.35. SDHOST_STATUS_REG (0x0048)



SDHOST_FIFO_RX_WATERMARK 表示 FIFO 是否达到接收阈值，不关注数据是否传输。

0：未达到

1：已达到

(RO)

SDHOST_FIFO_TX_WATERMARK 表示 FIFO 是否达到发送阈值，不关注数据是否传输。

0：未达到

1：已达到

(RO)

SDHOST_FIFO_EMPTY 表示 FIFO 是否已空。

0：未空

1：已空

(RO)

SDHOST_FIFO_FULL 表示 FIFO 是否已满。

0：未满

1：已满

(RO)

见下页...

Register 57.35. SDHOST_STATUS_REG (0x0048)

接上页...

SDHOST_COMMAND_FSM_STATES 表示命令 FSM 状态。

0x0：空闲
0x1：发送初始化序列
0x2：发送命令开始位
0x3：发送命令接收位
0x4：发送命令索引编号和参数
0x5：发送命令 CRC-7
0x6：发送命令结束位
0x7：接收响应开始位
0x8：接收响应 IRQ 响应
0x9：接收响应发送位
0xA：接收响应命令索引编号
0xB：接收响应数据
0xC：接收响应 CRC-7
0xD：接收响应结束位
0xE：命令通路等待 NCC
0xF：等待，命令到响应转换
(RO)

SDHOST_DATA_3_STATUS 表示选择的 sdhost_card_data[3] 输入信号值，检查是否存在卡。

0：卡不存在
1：卡存在
(RO)

SDHOST_DATA_BUSY 表示选择的 sdhost_card_data[0] 输入信号反向值，检查卡数据是否忙碌。

0：卡数据不忙碌
1：卡数据忙碌
(RO)

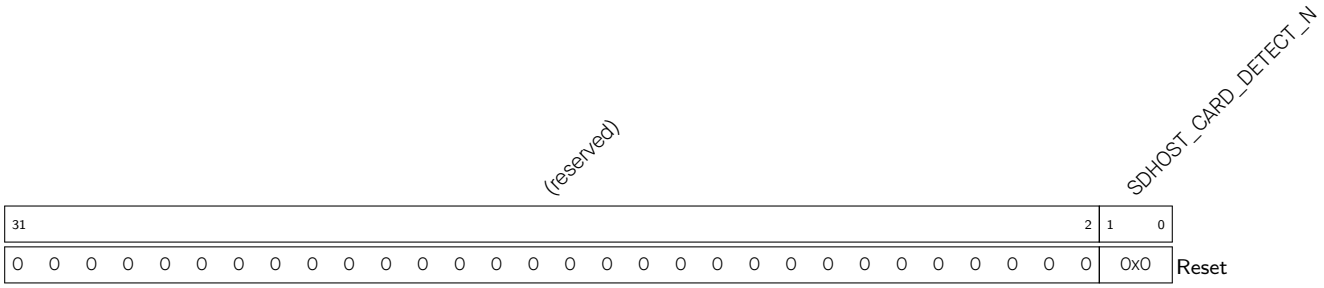
SDHOST_DATA_STATE_MC_BUSY 表示数据传输或接收状态机是否忙碌。

0：不忙碌
1：忙碌
(RO)

SDHOST_RESPONSE_INDEX 表示前一个响应的索引，包括由核心发送的任何自动停止。(RO)

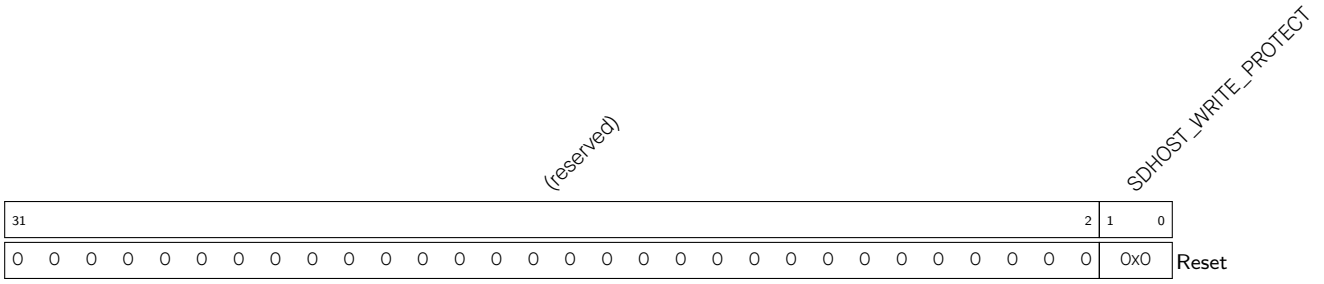
SDHOST_FIFO_COUNT 表示 FIFO 中已填充位置的数量。(RO)

Register 57.36. SDHOST_CDETECT_REG (0x0050)



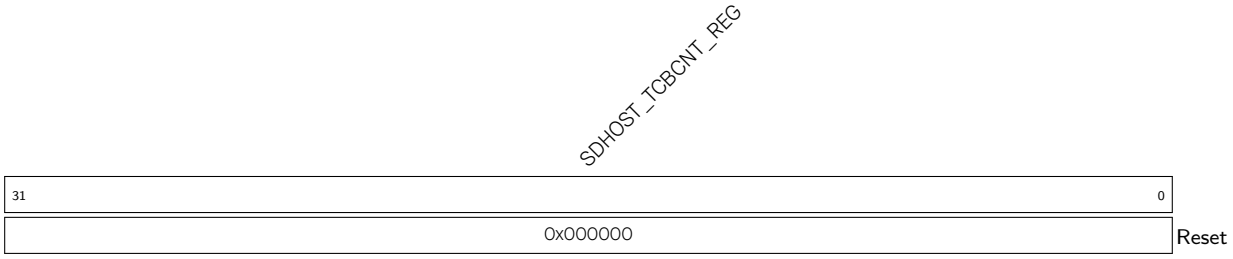
SDHOST_CARD_DETECT_N 表示 sdhost_card_detect_n 输入信号的值。每张卡一位，位 0 对应卡 0。(RO)

Register 57.37. SDHOST_WRTPRT_REG (0x0054)



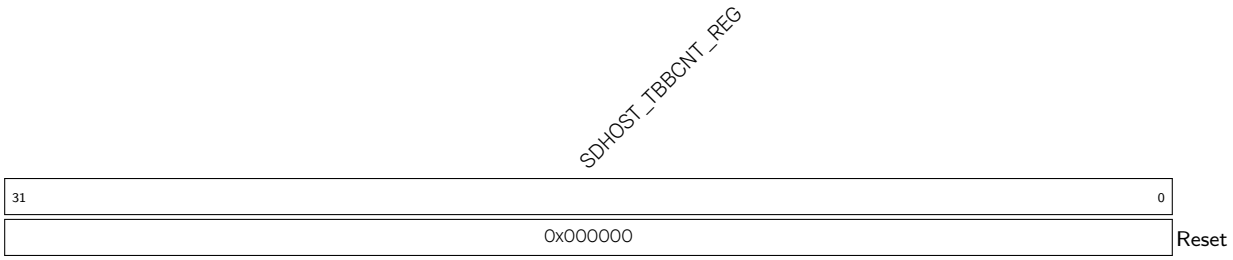
SDHOST_WRITE_PROTECT 表示 sdhost_card_write_prt 输入信号上的值，每张卡一位，位 0 对应卡 0。(RO)

Register 57.38. SDHOST_TCBCNT_REG (0x005C)



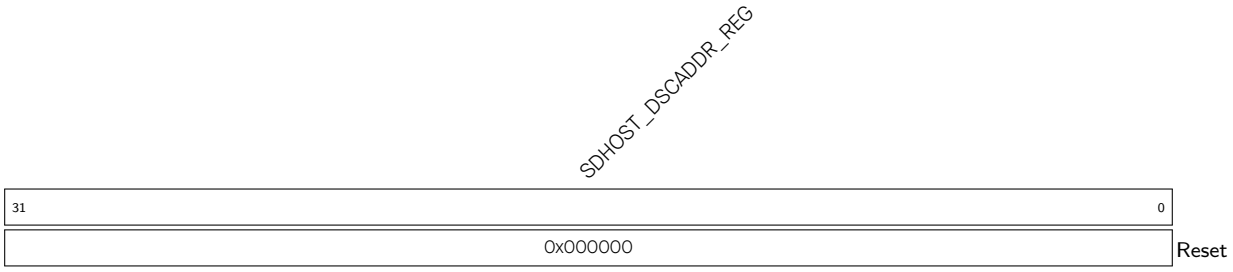
SDHOST_TCBCNT_REG 表示 CIU 传输到卡的字节数。(RO)

Register 57.39. SDHOST_TBBCNT_REG (0x0060)



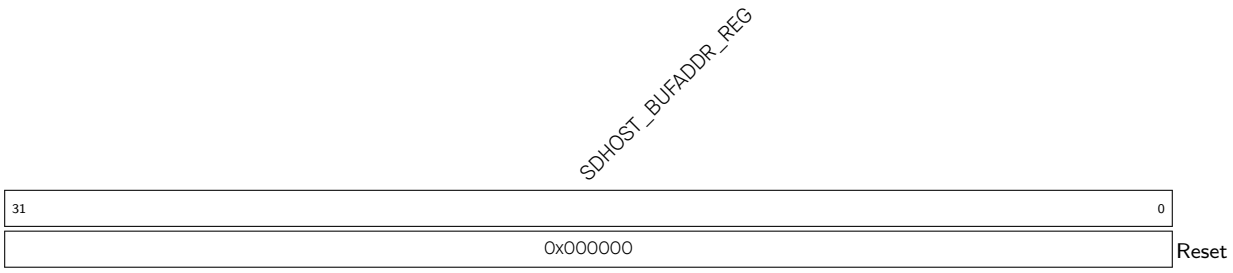
SDHOST_TBBCNT_REG 表示主机或 DMA 内存与 BIU FIFO 之间传输的字节数。(RO)

Register 57.40. SDHOST_DSCADDR_REG (0x0094)



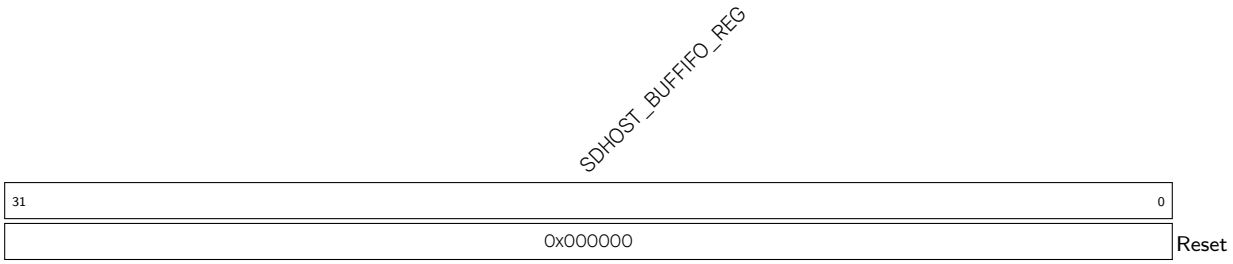
SDHOST_DSCADDR_REG 表示 DMA 当前读取的描述符起始地址。(RO)

Register 57.41. SDHOST_BUFADDR_REG (0x0098)



SDHOST_BUFADDR_REG 表示 DMA 当前访问的数据缓存地址。(RO)

Register 57.42. SDHOST_BUFFIFO_REG (0x0200)



SDHOST_BUFFIFO_REG FIFO 的接受数据或待发送数据。(R/W)

第 58 章

LED PWM 控制器 (LEDC)

58.1 概述

LED PWM 控制器用于生成控制 LED 的脉冲宽度调制信号 (PWM)，具有占空比自动渐变等功能。该外设也可生成 PWM 信号用作其他用途。

58.2 特性

LED PWM 控制器具有如下特性:

- 八个独立的 PWM 生成器（即八个通道）
- PWM 占空比最大精度为 20 位
- 四个独立定时器，可实现小数分频
- PWM 输出信号相位可调节
- PWM 占空比微调
- 占空比自动渐变—即 PWM 信号占空比可逐渐增加或减小，无须处理器干预，渐变完成时产生中断
- 每个 PWM 生成器包含 16 个占空比渐变区间，用于生成占空比伽马曲线渐变的信号。每个区间都可以独立配置占空比变化方向（增加或减少）、变化步长、变化次数以及变化频率
- 低功耗模式 (Light-sleep mode) 下可输出 PWM 信号
- 可以生成 ETM（事件任务矩阵）外设相关的事件，可以接收 ETM 外设相关的任务

由于四个定时器具有相同的功能和运行方式，下文将四个定时器统称为定时器*x*（*x* 的范围是 0 到 3）。同样的，八个 PWM 生成器的功能和运行方式也相同，因此下文将统称为 PWM*n*（*n* 的范围是 0 到 7）。

58.3 架构概览

图 58.3-1 为 LED PWM 控制器的架构。

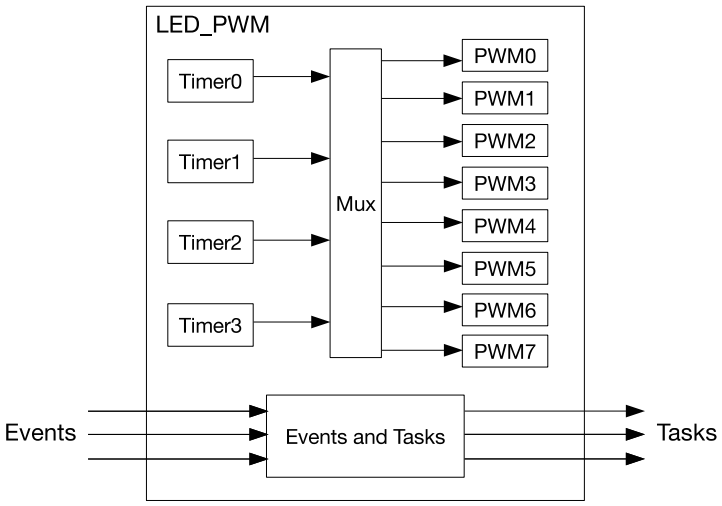


图 58.3-1. LED PWM 控制器架构

四个定时器每个内部都有一个时基计数器（即基于基准时钟周期计数的计数器），因此每个定时器都可以独立配置（可配置时钟分频器和计数器最大值）。每个 PWM 生成器通过配置 LEDC_TIMER_SEL_CH*n* 可以在四个定时器中择一，并以该定时器的计数值 timer*x*_cnt 为基准生成 PWM 信号。

图 58.3-2 为定时器和 PWM 生成器的主要功能块。

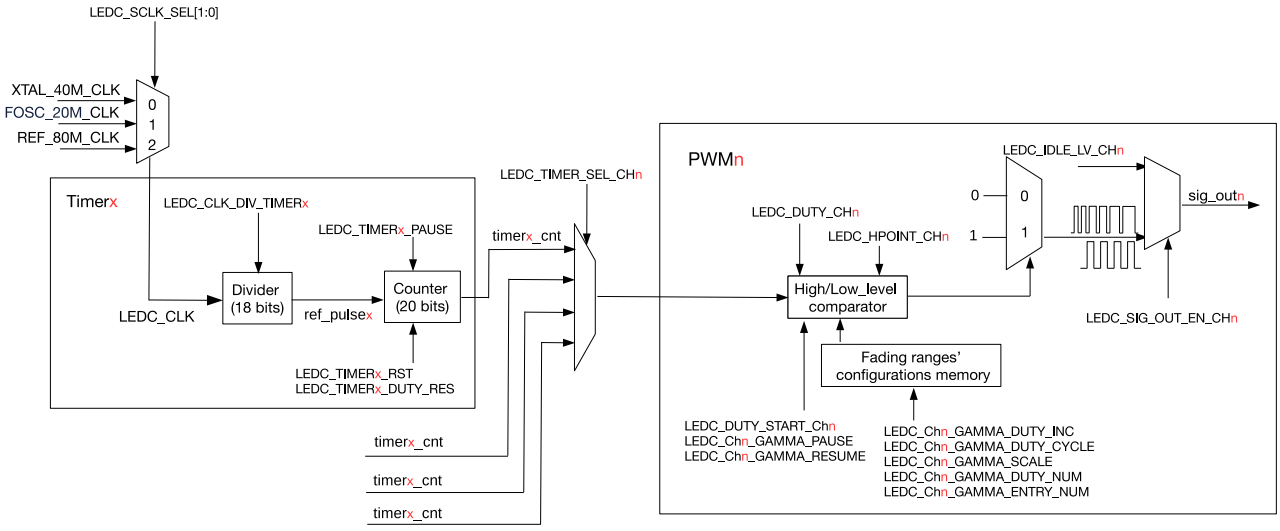


图 58.3-2. 定时器和 PWM 生成器功能框图

58.4 功能描述

58.4.1 定时器

LED PWM 控制器的每个定时器内部都有一个时基计数器。图 58.3-2 中时基计数器使用的时钟信号称为 ref_pulse*x*。所有定时器使用同一个时钟源信号 LEDC_CLK，该时钟源信号经分频器分频后产生 ref_pulse*x* 供计数器使用。

58.4.1.1 时钟源

软件可配置的 LED PWM 寄存器由 APB_CLK 时钟驱动。要使用 LED PWM 控制器，需使能 LED PWM 的 APB_CLK 时钟信号。APB_CLK 时钟信号可通过置位 HP_SYS_CLKRST_SOC_CLK_CTRL3_REG 寄存器的 HP_SYS_CLKRST_LEDC_APB_CLK_EN 字段使能。LEDC_CLK 时钟信号可通过置位 HP_SYS_CLKRST_PERI_CLK_CTRL22_REG 寄存器的 HP_SYS_CLKRST_LEDC_CLK_EN 字段使能。LED PWM 可通过软件置位 HP_SYS_CLKRST_HP_RST_EN1_REG 寄存器的 HP_SYS_CLKRST_RST_EN_LEDC 字段复位。

LED PWM 控制器的定时器从下述时钟信号中选择一个作为其时钟源信号：XTAL_40M_CLK，FOSC_20M_CLK 和 REF_80M_CLK。为 LEDC_CLK 选择时钟源信号的配置如下：

- XTAL_40M_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 0
- FOSC_20M_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 1
- REF_80M_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 2

之后，LEDC_CLK 信号会进入时钟分频器。

更多信息，请参阅章节 11 复位和时钟。

58.4.1.2 时钟分频器配置

LEDC_CLK 信号传输到时钟分频器，产生 ref_pulse*x* 信号供计数器使用。ref_pulse*x* 的频率等于 LEDC_CLK 的频率经分频系数 LEDC_CLK_DIV 分频后的结果（见图 58.3-2）。

由于时钟分频器为小数分频，因此分频系数 LEDC_CLK_DIV 可为非整数，其可根据下列等式配置：

$$\text{LEDC_CLK_DIV} = A + \frac{B}{256}$$

- 整数部分 *A* 为 LEDC_CLK_DIV_TIMER*x* 字段的高 10 位（即 LEDC_TIMER*x*_CONF_REG[22:13]）
- 小数部分 *B* 为 LEDC_CLK_DIV_TIMER*x* 字段的低 8 位（即 LEDC_TIMER*x*_CONF_REG[12:5]）

小数部分 *B* 为 0 时，LEDC_CLK_DIV 的值为整数（整数分频）。也就是说，每 *A* 个 LEDC_CLK 时钟周期产生一个 ref_pulse*x* 时钟脉冲。

小数部分 *B* 不为 0 时，LEDC_CLK_DIV 的值为非整数。时钟分频器按照 *A* 个 LEDC_CLK 时钟周期和 (*A*+1) 个 LEDC_CLK 时钟周期轮流产生 ref_pulse*x* 时钟脉冲，实现非整数分频。这样一来，ref_pulse*x* 时钟脉冲的平均频率便会是理想值（非整数分频的频率）。每 256 个 ref_pulse*x* 时钟脉冲中：

- 有 *B* 个 ref_pulse*x* 时钟脉冲以 (*A*+1) 个 LEDC_CLK 时钟周期分频
- 有 (256-*B*) 个 ref_pulse*x* 时钟脉冲以 *A* 个 LEDC_CLK 时钟周期分频
- 以 (*A*+1) 个 LEDC_CLK 时钟周期分频的 ref_pulse*x* 时钟脉冲均匀分布在以 *A* 个 LEDC_CLK 时钟周期分频的 ref_pulse*x* 时钟脉冲中

图 58.4-1 展示了分频系数 LEDC_CLK_DIV 非整数时，LEDC_CLK 时钟脉冲和 ref_pulse*x* 时钟脉冲的关系。

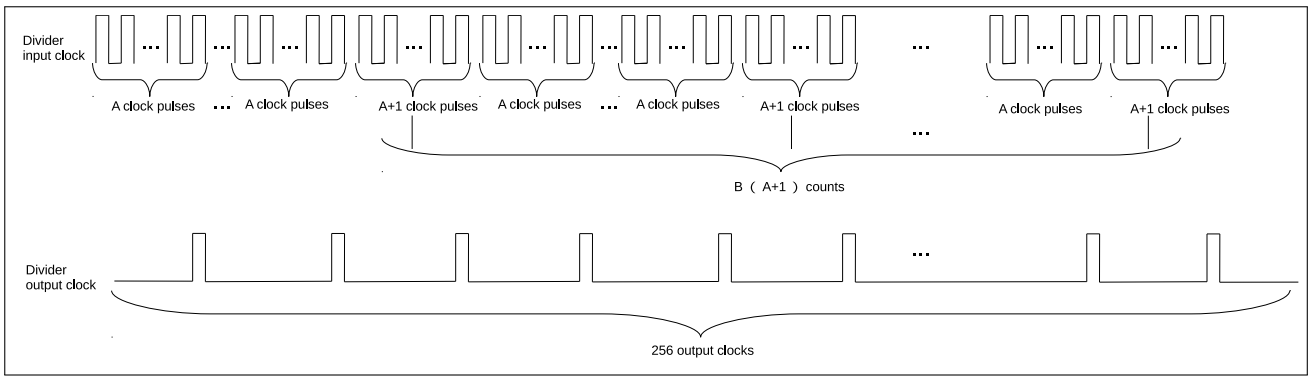


图 58.4-1. LEDC_CLK_DIV 为非整数时的分频

在运行时改变定时器时钟的分频系数，需先配置 LEDC_CLK_DIV_TIMER*x* 字段，然后置位 LEDC_TIMER*x*_PARA_UP 字段应用新配置。新配置会在计数器下次溢出时生效。LEDC_TIMER*x*_PARA_UP 字段由硬件自动清除。

58.4.1.3 20 位计数器

每个定时器有一个以 ref_pulse*x* 为基准时钟的 20 位时基计数器（见图 58.3-2）。LEDC_TIMER*x*_DUTY_RES 字段用于配置 20 位计数器的实际使用位宽。因此，PWM 信号的最大精度为 20 位。计数器最大可计数至 $2^{\text{LEDC_TIMER}x\text{_DUTY_RES}} - 1$，然后溢出并重新从 0 开始计数。软件可以读取、复位、暂停计数器。计数器和 PWM 信号精度的关系如图 58.4-2 所示。

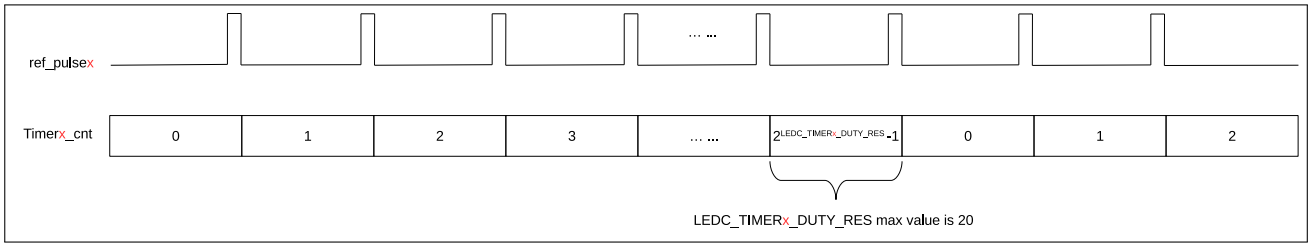


图 58.4-2. 计数器和 PWM 信号精度关系

计数器可在每次溢出时触发 LEDC_TIMER*x*_OVF_INT 中断（这个中断为硬件自动产生，不需要配置）。计数器也可配置为在溢出 LEDC_OVF_NUM_CH*n* + 1 次时触发 LEDC_OVF_CNT_CH*n*_INT 中断，该中断配置步骤如下：

1. 配置 LEDC_TIMER_SEL_CH*n* 为 PWM 生成器选择该定时器
2. 置位 LEDC_OVF_CNT_EN_CH*n* 使能溢出次数计数器
3. 把 LEDC_OVF_NUM_CH*n* 的值设为触发中断的计数器溢出次数减 1
4. 置位 LEDC_OVF_CNT_CH*n*_INT_ENA 使能溢出中断
5. 配置 LEDC_TIMER*x*_DUTY_RES 为所选的定时器 Timer*x* 指定计数器的位宽，等待 LEDC_OVF_CNT_CH*n*_INT 中断产生

在运行时改变计数器的最大值，需先配置 LEDC_TIMER*x*_DUTY_RES 字段，然后置位 LEDC_TIMER*x*_PARA_UP 字段。新的配置在计数器下一次溢出时生效。如果重新配置 LEDC_OVF_CNT_EN_CH*n* 字段，需置位 LEDC_PARA_UP_CH*n* 应用新配置。总之，更改配置时需置位 LEDC_TIMER*x*_PARA_UP 或 LEDC_PARA_UP_CH*n* 应用新配置。

LEDC_TIMERx_PARA_UP 和 LEDC_PARA_UP_CHn 字段由硬件自动清除。

如图 58.3-2 所示，PWM 生成器输出信号 sig_outn 的频率取决于定时器时钟源 LEDC_CLK 的频率、时钟分频系数 LEDC_CLK_DIV 以及占空比精度（计数器位宽）LEDC_TIMERx_DUTY_RES：

$$f_{\text{PWM}} = \frac{f_{\text{LEDC_CLK}}}{\text{LEDC_CLK_DIV} \cdot 2^{\text{LEDC_TIMERx_DUTY_RES}}}$$

上述公式变形后，可得到以下公式计算预期的占空比精度：

$$\text{LEDC_TIMERx_DUTY_RES} = \log_2\left(\frac{f_{\text{LEDC_CLK}}}{f_{\text{PWM}} \cdot \text{LEDC_CLK_DIV}}\right)$$

表 58.4-1 列出了常用配置频率及其对应精度。

表 58.4-1. 常用配置频率及精度

LEDC_CLK	PWM 频率	最高精度（位）[1]	最低精度（位）[2]
REF_80M_CLK (80 MHz)	1 kHz	16	6
REF_80M_CLK (80 MHz)	5 kHz	13	3
REF_80M_CLK (80 MHz)	10 kHz	12	2
XTAL_40M_CLK (40 MHz)	1 kHz	15	5
XTAL_40M_CLK (40 MHz)	4 kHz	13	3
FOSC_20M_CLK (20 MHz)	1 kHz	14	4
FOSC_20M_CLK (20 MHz)	2 kHz	13	3

[1] 最高精度指时钟分频系数 LEDC_CLK_DIV 为 1 时的精度。如果经公式计算出的最高精度超过了计数器位宽 20 位，则最高精度为 20。

[2] 最低精度指时钟分频系数 LEDC_CLK_DIV 为 $1023 + \frac{255}{256}$ 时的精度。如果经公式计算出的最低精度小于 0，则最低精度为 1。

58.4.2 PWM 生成器

要生成 PWM 信号，PWM 生成器 (PWMn) 需选择一个定时器 (Timerx)。每个 PWM 生成器均可通过置位 LEDC_TIMER_SEL_CHn 单独配置，在四个定时器中选择一个用于生成 PWM 信号。

如图 58.3-2 所示，每个 PWM 生成器主要包括一个高低电平比较器和两个选择器。PWM 生成器将定时器的 20 位计数值 (Timerx_cnt) 与高低电平比较器的值 Hpointn 和 Lpointn 比较。如果定时器的计数值等于 Hpointn 或 Lpointn，PWM 信号可以输出高低电平：

- 如果 Timerx_cnt == Hpointn，则 sig_outn 为 1。
- 如果 Timerx_cnt == Lpointn，则 sig_outn 为 0。

图 58.4-3 展示了如何使用 Hpointn 和 Lpointn 生成占空比固定的 PWM 信号。

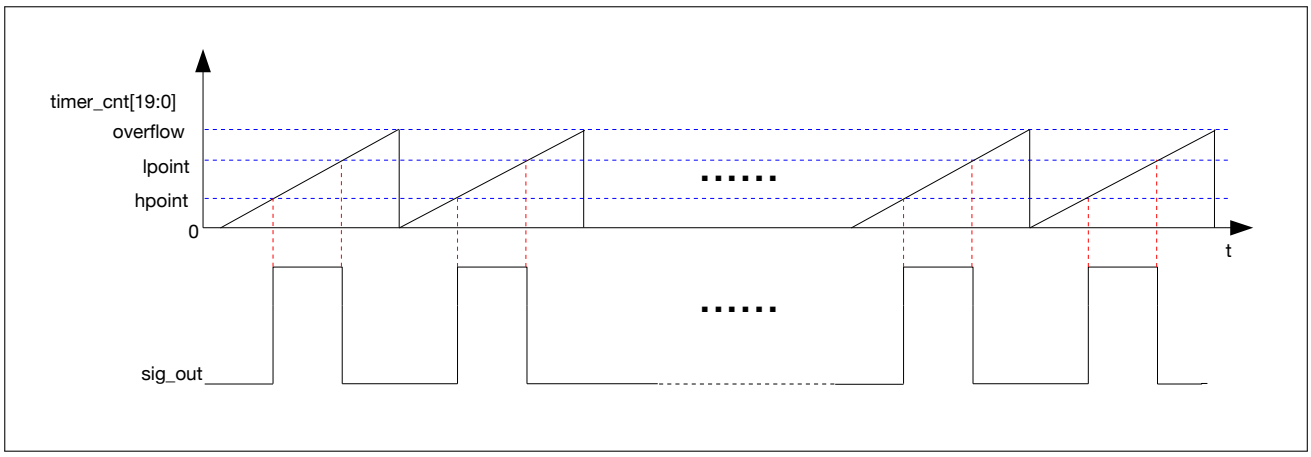


图 58.4-3. LED PWM 输出信号图

每当所选定时器的计数器溢出时，PWM 生成器 (PWM*n*) 的 Hpoint*n* 值更新为 LEDC_HPOINT_CH*n*。Lpoint*n* 的值同样在计数器每次溢出时更新，为 LEDC_DUTY_CH*n*[24:4] 和 LEDC_HPOINT_CH*n* 的和。通过配置 LEDC_DUTY_CH*n*[24:4] 和 LEDC_HPOINT_CH*n* 两个字段可对 Hpoint*n* 和 Lpoint*n* 进行设置，进而可设置 PWM 输出信号的相对相位和占空比。

置位 LEDC_SIG_OUT_EN_CH*n*，开启 PWM 信号 (sig_out*n*) 输出；清除 LEDC_SIG_OUT_EN_CH*n*，关闭 PWM 信号输出，输出信号 (sig_out*n*) 输出恒定电平，电平值为 LEDC_IDLE_LV_CH*n*。

LEDC_DUTY_CH*n*[3:0] 通过周期性改变 PWM 输出信号 sig_out*n* 的占空比实现微调。如 LEDC_DUTY_CH*n*[3:0] 不为 0，那么 sig_out*n* 每 16 个周期中，有 LEDC_DUTY_CH*n*[3:0] 个周期的 PWM 脉冲占空比要比 (16 - LEDC_DUTY_CH*n*[3:0]) 个周期的脉冲占空比多一个定时器的计数周期。比如，如果 LEDC_DUTY_CH*n*[24:4] 设为 10，LEDC_DUTY_CH*n*[3:0] 设为 5，则 16 个周期中，有 5 个周期的 PWM 脉冲占空比为 11，剩余 11 个周期的 PWM 脉冲占空比为 10。16 个周期的平均占空比为 10.3125。

如果重新配置 LEDC_TIMER_SEL_CH*n*、LEDC_HPOINT_CH*n*、LEDC_DUTY_CH*n*[24:4] 和 LEDC_SIG_OUT_EN_CH*n* 字段，需置位 LEDC_PARA_UP_CH*n* 应用新配置。新配置在计数器下次溢出时生效。LEDC_PARA_UP_CH*n* 字段由硬件自动清除。

58.4.3 占空比渐变

PWM 生成器可以渐变 PWM 输出信号的占空比（即由一种占空比逐渐变为为另一种占空比）。每个 PWM 生成器最多可以包含 16 个占空比渐变区间，每个区间可以独立配置占空比渐变方向（增加或减少）、渐变步长、渐变次数和渐变频率。如果开启占空比渐变功能，每个区间的 Lpoint*n* 的值会根据该区间的渐变配置进行变化。

58.4.3.1 线性占空比渐变

通过配置第一个占空比渐变区间的渐变方向、渐变步长、渐变次数和渐变频率，可以产生线性占空比渐变 PWM 信号。

配置步骤请见章节 58.6。

当上述配置步骤完成后，每当计数器溢出 LEDC_CH*n*_GAMMA_DUTY_CYCLE 次后，PWM 生成器将 PWM 信号的占空比渐变一次。PWM 信号每次渐变时，其 Lpoint*n* 会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）LEDC_CH*n*_GAMMA_SCALE，相应地占空比会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）。

$$\frac{\text{LEDC_CH}n\text{_GAMMA_SCALE}}{\text{LEDC_TIMER}x\text{_DUTY_RES}}$$

PWM 信号的占空比会渐变 LEDC_CH*n*_GAMMA_DUTY_NUM 次。完成该次数的渐变后，PWM 信号的占空比停止变化并保持以该占空比输出。由于每次渐变时占空比增加/减少的值都相同，所以产生的 PWM 信号为线性占空比渐变信号。

图 58.4-4 展示了一个 PWM 信号占空比线性渐变的例子。

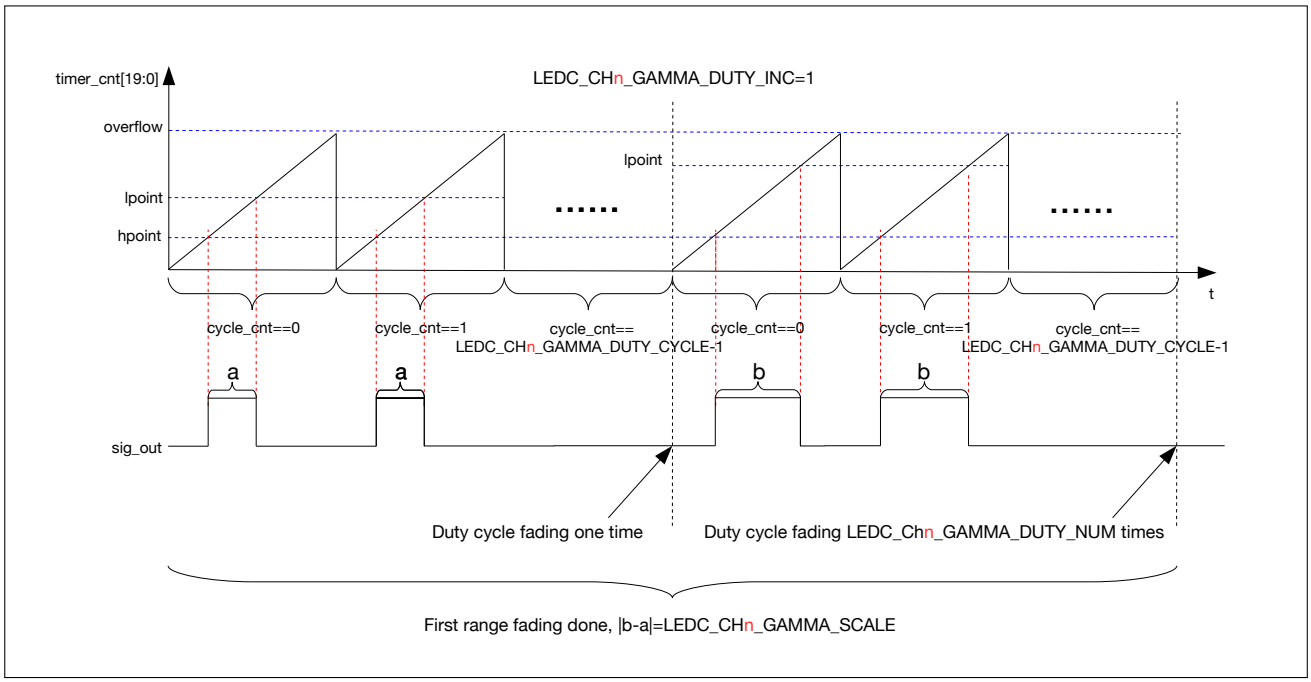


图 58.4-4. 输出信号占空比线性渐变图

58.4.3.2 伽马曲线渐变

通过配置多个占空比渐变区间的占空比渐变方向、渐变步长、渐变次数和渐变频率，PWM 生成器可以产生伽马曲线渐变的 PWM 信号。

配置步骤请见章节 58.6。

经过上述所有的配置步骤，PWM 生成器会产生包含 LEDC_CH*n*_GAMMA_ENTRY_NUM 个占空比渐变区间的 PWM 信号。首先，PWM 信号会根据序号为 0 的占空比渐变区间的配置进行占空比渐变，渐变完成后，按照序号为 1 的占空比渐变区间的配置进行占空比渐变，直到序号为 (LEDC_CH*n*_GAMMA_ENTRY_NUM − 1) 的占空比渐变区间（即最后一个区间）完成渐变。在每个区间中，PWM 信号都是独立变化的。对于每个区间，PWM 信号会在计数器每溢出 LEDC_CH*n*_GAMMA_DUTY_CYCLE 次后，进行一次占空比渐变，对于每次渐变，其 Lpoint*n* 会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）LEDC_CH*n*_GAMMA_SCALE，相对的，占空比会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）

$$\frac{\text{LEDC_CH}n\text{_GAMMA_SCALE}}{\text{LEDC_TIMER}x\text{_DUTY_RES}}$$

在该区间内，占空比会渐变 LEDC_CH*n*_GAMMA_DUTY_NUM 次，当指定次数的占空比渐变完成后，视为当前区间的占空比渐变完成。

当 LEDC_CHn_GAMMA_ENTRY_NUM 个区间的占空比渐变都完成后，PWM 信号会停止占空比渐变，其占空比会保持最后一次渐变的占空比。由于每个占空比渐变区间的渐变过程都是独立不同的线性渐变，因此多个区间的线性占空比渐变就可以拟合成一个特定的伽马曲线占空比渐变。

图 58.4-5 展示了一个伽马曲线渐变 PWM 信号。

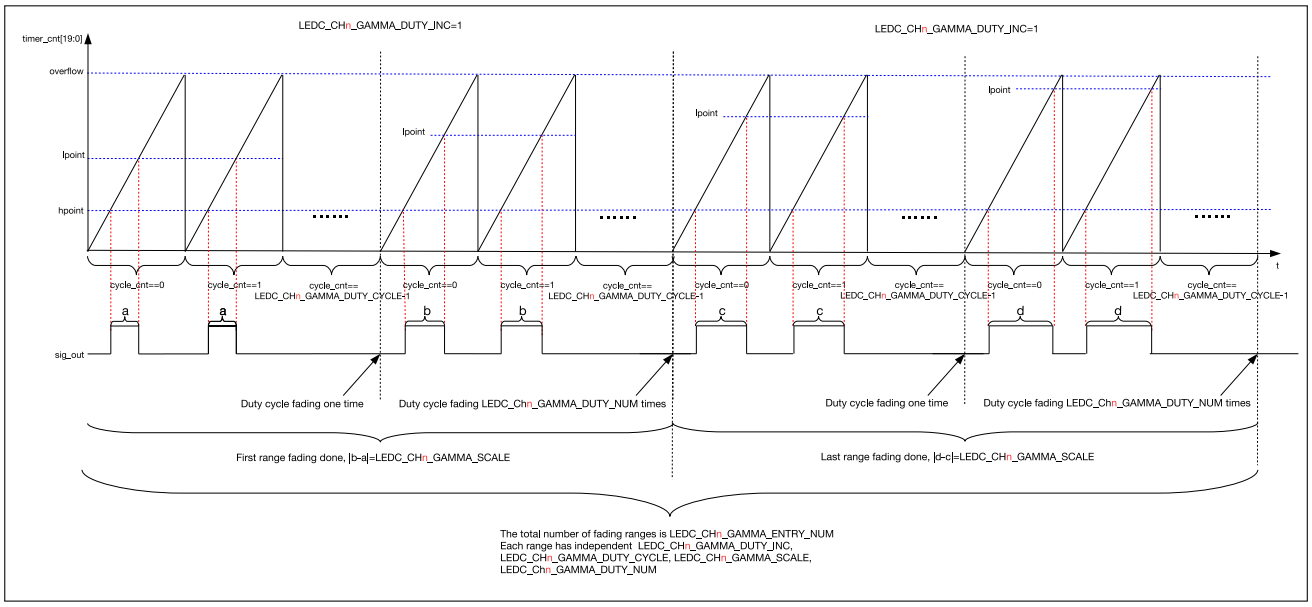


图 58.4-5. 输出信号占空比伽马曲线渐变图

58.4.3.3 占空比渐变暂停和恢复

向 LEDC_CHn_GAMMA_CONF_REG 寄存器的 LEDC_CHn_GAMMA_PAUSE 字段写 1 可以暂停已经开始的占空比渐变过程。一旦将 LEDC_CHn_GAMMA_PAUSE 字段置 1，PWM 信号将会保持最近一次渐变后的占空比。

向 LEDC_CHn_GAMMA_CONF_REG 寄存器的 LEDC_CHn_GAMMA_RESUME 字段写 1 可以恢复被暂停的占空比渐变过程。一旦将 LEDC_CHn_GAMMA_RESUME 字段置 1，PWM 信号将会按照暂停时所在的占空比渐变区间的配置，从暂停时的占空比继续进行占空比渐变，直到所有有效的区间都完成指定的占空比渐变（LEDC_CHn_GAMMA_RESUME 字段置 1 后，硬件会自动清除 LEDC_CHn_GAMMA_PAUSE 字段）。

58.4.4 事件任务矩阵功能

在 ESP32-P4 中，LEDC 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 LEDC 的 ETM 任务，或者通过 LEDC 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 LEDC 相关的 ETM 任务和 ETM 事件。

ETM 外设相关的事件和任务的使能由寄存器 LEDC_EVT_TASK_EN0_REG、LEDC_EVT_TASK_EN1_REG 和 LEDC_EVT_TASK_EN2_REG 的相关字段所配置。事件和任务与寄存器 LEDC_EVT_TASK_EN0_REG、LEDC_EVT_TASK_EN1_REG、LEDC_EVT_TASK_EN2_REG 的每个字段的对应关系参照章节 58.8。

LEDC 可接收的 ETM 任务有：

- LEDC_TASK_DUTY_SCALE_UPDATE_CHn：如果 LEDC_TASK_DUTY_SCALE_UPDATE_CHn_EN 字段使能，当接收到 LEDC_TASK_DUTY_SCALE_UPDATE_CHn 任务后，相应的 LEDC 通道（PWMn）将会根据最新配置的 LEDC_CHn_GAMMA_SCALE 的值，重新开始生成占空比渐变 PWM 信号。

- LEDC_TASK_TIMER*x*_RES_UPDATE：如果 LEDC_TASK_TIMER*x*_RES_UPDATE_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RES_UPDATE 任务后，相应的定时器（Timer*x*）将会根据最新配置的 LEDC_TIMER*x*_DUTY_RES 字段的值，在下一次计数器溢出时，更新计数器的最大计数值为该值。
- LEDC_TASK_TIMER*x*_CAP：如果 LEDC_TASK_TIMER*x*_CAP_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_CAP 任务时，相应的定时器（Timer*x*）将会抓取当前计数器的计数值。该计数值将会存储在寄存器 LEDC_TIMER*x*_CNT_CAP_REG 的 LEDC_TIMER*x*_CNT_CAP 字段。
- LEDC_TASK_SIG_OUT_DIS_CH*n*：如果 LEDC_TASK_SIG_OUT_DIS_CH*n*_EN 字段使能，当接收到 LEDC_TASK_SIG_OUT_DIS_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）的 PWM 信号将停止输出，该通道的输出信号 (sig_out*n*) 将会变为 LEDC_IDLE_LV_CH*n* 字段所配置电平的固定电平信号，如图 58.3-2 所示。
- LEDC_TASK_OVF_CNT_RST_CH*n*：如果 LEDC_TASK_OVF_CNT_RST_CH*n*_EN 字段使能，当接收到 LEDC_TASK_OVF_CNT_RST_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）定时器的计数器溢出次数的计数器值将会被复位为 0。
- LEDC_TASK_TIMER*x*_RST：如果 LEDC_TASK_TIMER*x*_RST_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RST 任务时，相应的定时器（Timer*x*）的计数器值将会被复位为 0。
- LEDC_TASK_TIMER*x*_RESUME 和 LEDC_TASK_TIMER*x*_PAUSE：如果 LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RESUME 和 LEDC_TASK_TIMER*x*_PAUSE 任务时，相应的定时器（Timer*x*）将会被交替暂停或恢复。即，当该任务被接收一次时，定时器（Timer*x*）将会暂停，当该任务再被接收一次时，定时器（Timer*x*）将会恢复。
- LEDC_TASK_GAMMA_RESTART_CH*n*：如果 LEDC_TASK_GAMMA_RESTART_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_RESTART_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会重新开始产生占空比渐变 PWM 信号。
- LEDC_TASK_GAMMA_PAUSE_CH*n*：如果 LEDC_TASK_GAMMA_PAUSE_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_PAUSE_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会在计数器下一次溢出时停止占空比的渐变。即，当该任务被接收后，相应的 LEDC 通道（PWM*n*）的 PWM 信号的占空比将不会再进行渐变。
- LEDC_TASK_GAMMA_RESUME_CH*n*：如果 LEDC_TASK_GAMMA_RESUME_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_RESUME_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会在计数器下一次溢出时恢复占空比的渐变。即，当该任务被接收后，相应的 LEDC 通道（PWM*n*）的 PWM 信号将会按照暂停时的占空比渐变配置继续进行占空比的渐变。

LEDC 可产生的 ETM 事件有：

- LEDC_EVT_DUTY_CHNG_END_CH*n*：如果 LEDC_EVT_DUTY_CHNG_END_CH*n*_EN 字段使能，当相应的 LEDC 通道（PWM*n*）占空比渐变完成后，将会产生 LEDC_EVT_DUTY_CHNG_END_CH*n* 事件。
- LEDC_EVT_OVF_CNT_PLS_CH*n*：如果 LEDC_EVT_OVF_CNT_PLS_CH*n*_EN 字段使能，当相应的 LEDC 通道（PWM*n*）的定时器的计数器溢出 LEDC_OVF_NUM_CH*n* + 1 次后，将会产生 LEDC_EVT_OVF_CNT_PLS_CH*n* 事件。
- LEDC_EVT_TIME_OVF_TIMER*x*：如果 LEDC_EVT_TIME_OVF_TIMER*x*_EN 字段使能，当相应的定时器（Timer*x*）的计数器溢出时，将会产生 LEDC_EVT_TIME_OVF_TIMER*x* 事件。
- LEDC_EVT_TIMER*x*_CMP：如果 LEDC_EVT_TIME*x*_CMP_EN 字段使能，当相应的定时器（Timer*x*）的计数器值等于寄存器 LEDC_TIMER*x*_CMP_REG 的 LEDC_TIMER*x*_CMP 字段的值时，将会产生 LEDC_EVT_TIMER*x*_CMP 事件。

在具体应用中，LEDC 的 ETM 事件可以用来触发 LEDC 的 ETM 任务，例如，LEDC_EVT_DUTY_CHNG_END_CH*n* 事件可以触发 LEDC_TASK_GAMMA_RESTART_CH*n* 任务，从而在一次渐变完成后直接开始下一次渐变。

58.5 中断

ESP32-P4 的 LEDC 模块会产生 LEDC_INT 中断信号，该信号会被发送到 中断矩阵。

下列内部中断源可以产生 LEDC_INT 中断信号：

- LEDC_OVF_CNT_CH*n*_INT：定时器计数器溢出 LEDC_OVF_NUM_CH*n* + 1 次且寄存器 LEDC_OVF_CNT_EN_CH*n* 置 1 时触发中断。
- LEDC_DUTY_CHNG_END_CH*n*_INT：PWM*n* 渐变完成后触发中断。
- LEDC_TIMER*x*_OVF_INT：定时器达到最大计数值时触发中断。

只有当中断使能寄存器 *XXX*_ENA 置位为 1 时，才能触发上述中断源。置位 *XXX*_CLR 可清除中断（*XXX* 为中断源）。

58.6 配置流程

LED PWM 控制器生成 PWM 信号的配置流程为：

1. 参照 58.4.1 对定时器 Timer*x* 进行配置。配置完成后，时钟源、分频器与计数器就配置完成了。
2. 参照 58.4.2 对 PWM*n* 生成器进行配置。配置完成后，PWM*n* 从四个定时器中选择其中一个，并设置 PWM*n* 输出的相位和占空比。
3. 根据不同的占空比渐变类型选择相应的配置过程：
   (a) 线性占空比渐变：
      i. 配置寄存器字段 LEDC_DUTY_CH*n*。该字段用来设置 Lpoint*n* 的初始值。
      ii. 置位寄存器字段 LEDC_DUTY_START_CH*n*。当该字段置位/清除，占空比渐变会使能/关闭。
      iii. 将线性占空比渐变配置写入寄存器（LEDC_CH*n*_MEM 起始地址 + 占空比区间值 * 4），LEDC_CH*n*_MEM 起始地址包含在表 58.7-1 中。

      请注意以下的字段名称只是为了方便叙述，寄存器中并不存在这些字段。

      A. 第 0 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_INC）用于配置方向。当该字段置位或清零时，当前所配置的占空比渐变区间的 Lpoint*n* 会增加/减少。

      B. 第 1 ~ 10 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_CYCLE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每增加/减少一次，计数器溢出的次数。即当计数器溢出次数为配置次数，Lpoint*n* 将会增加/减少一次。

      C. 第 11 ~ 20 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_SCALE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每次增加/减少的量。

      D. 第 21 ~ 30 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_NUM）用来设置当前所配置的占空比渐变区间的渐变次数。

E. 占空比范围值（从 0 到 15）用于指定上述配置的属性属于哪个占空比渐变区间。由于线性占空比渐变只需要配置第一个占空比渐变区间即可，所以占空比范围值应配置为 0。

iv. 将寄存器 LEDC_CH*n*_GAMMA_CONF_REG 的 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为 1。该字段用来指定一次渐变过程中有效的占空比渐变区间的个数（当指定数目的区间都渐变完成后，PWM 信号会停止占空比渐变，PWM 生成器会触发 LEDC_DUTY_CHNG_END_CH*n*_INT 中断）。由于线性占空比渐变只包含一个渐变区间（第一个占空比渐变区间），所以 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为 1。

v. 置位寄存器字段 LEDC_PARA_UP_CH*n* 应用上述步骤的配置。当该字段被置位后，上述的占空比渐变配置会在下一次计数器溢出时生效，PWM 生成器会输出指定配置的线性占空比渐变 PWM 信号（LEDC_PARA_UP_CH*n* 字段会被硬件自动清除）。

(b) 伽马曲线渐变：

i. 与章节 58.4.3.1 的步骤 1 相同。

ii. 与章节 58.4.3.1 的步骤 2 相同。

iii. 配置多个占空比渐变区间：将伽马曲线渐变配置写入寄存器（LEDC_CH*n*_MEM 起始地址 + 占空比区间值 * 4），LEDC_CH*n*_MEM 起始地址包含在表 58.7-1 中。

A. 第 0 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_INC）用于配置方向。当该字段置位或清零时，当前所配置的占空比渐变区间的 Lpoint*n* 会增加/减少。

B. 第 1 ~ 10 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_CYCLE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每增加/减少一次，计数器溢出的次数。即当计数器溢出次数为配置次数，Lpoint*n* 将会增加/减少一次。

C. 第 11 ~ 20 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_SCALE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每次增加/减少的量。

D. 第 21 ~ 30 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_NUM）用来设置当前所配置的占空比渐变区间的渐变次数。

E. 占空比范围值 (0 ~ 15) 用于指定上述配置的属性属于哪个占空比渐变区间。对于伽马曲线渐变，所配置的占空比渐变区间的序号必须从 0 开始，每配置完一个占空比渐变区间，下一个要配置的占空比渐变区间的序号需要加 1。

F. 上述步骤配置完毕后，一个占空比渐变区间的配置视为完成，然后可以重复上述的步骤配置其他的占空比渐变区间的参数。每个占空比渐变区间的配置都是独立的，拥有有效配置的区间个数 (0 ~ 15) 可以是任意的。

iv. 当所需的占空比渐变区间都配置完毕后，将 LEDC_CH*n*_GAMMA_CONF_REG 寄存器的 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为步骤 3 中配置过的占空比渐变区间的总个数。

v. 置位寄存器的 LEDC_PARA_UP_CH*n* 字段，应用以上步骤的配置。当该字段被置位后，上述步骤的占空比渐变配置会在下一次计数器溢出时生效，PWM 生成器会按照配置输出伽马曲线渐变的 PWM 信号（LEDC_PARA_UP_CH*n* 字段会自动被硬件清除）。

按照上述步骤配置完成后，LED PWM 控制器会生成所需的 PWM 信号。

占空比渐变随时都可以暂停或恢复，详情请参考 58.4.3.3。如果需要在 PWM 信号停止渐变之前（即 LEDC_DUTY_CHNG_END_CH*n*_INT 尚未触发）更改占空比渐变配置，则需要在更改配置之前暂停占空比渐变。但是，如果需要在 PWM 信号停止渐变后（即 LEDC_DUTY_CHNG_END_CH*n*_INT 已被触发）更改占空比渐变配置，则可以直接进行更改。

软件随时都可以启用或禁用 PWM 信号输出，也可对输出信号的电平进行更改。详情请参考 58.4.2。

58.7 存储空间

每个 LEDC 通道都有一块存储空间用于存放占空比渐变的配置数据。每个存储空间最多可储存 16 个区间的配置数据。本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

表 58.7-1. LEDC 存储空间

名称	描述	大小	起始地址	终止地址	访问权限
LEDC_CH*n*_MEM	PWM*n* 内存	64 B	0x400 + PWM*n* * 64	0x400 + (PWM*n* + 1) * 64	WR

58.8 寄存器列表

本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LEDC_CH0_CONF0_REG	通道 0 的配置寄存器 0	0x0000	varies
LEDC_CH0_HPOINT_REG	通道 0 的 Hpoint 寄存器	0x0004	R/W
LEDC_CH0_DUTY_REG	通道 0 的初始占空比	0x0008	R/W
LEDC_CH0_CONF1_REG	通道 0 的配置寄存器 1	0x000C	R/W/SC
LEDC_CH1_CONF0_REG	通道 1 的配置寄存器 0	0x0014	varies
LEDC_CH1_HPOINT_REG	通道 1 的 Hpoint 寄存器	0x0018	R/W
LEDC_CH1_DUTY_REG	通道 1 的初始占空比	0x001C	R/W
LEDC_CH1_CONF1_REG	通道 1 的配置寄存器 1	0x0020	R/W/SC
LEDC_CH2_CONF0_REG	通道 2 的配置寄存器 0	0x0028	varies
LEDC_CH2_HPOINT_REG	通道 2 的 Hpoint 寄存器	0x002C	R/W
LEDC_CH2_DUTY_REG	通道 2 的初始占空比	0x0030	R/W
LEDC_CH2_CONF1_REG	通道 2 的配置寄存器 1	0x0034	R/W/SC
LEDC_CH3_CONF0_REG	通道 3 的配置寄存器 0	0x003C	varies
LEDC_CH3_HPOINT_REG	通道 3 的 Hpoint 寄存器	0x0040	R/W
LEDC_CH3_DUTY_REG	通道 3 的初始占空比	0x0044	R/W
LEDC_CH3_CONF1_REG	通道 3 的配置寄存器 1	0x0048	R/W/SC
LEDC_CH4_CONF0_REG	通道 4 的配置寄存器 0	0x0050	varies
LEDC_CH4_HPOINT_REG	通道 4 的 Hpoint 寄存器	0x0054	R/W
LEDC_CH4_DUTY_REG	通道 4 的初始占空比	0x0058	R/W
LEDC_CH4_CONF1_REG	通道 4 的配置寄存器 1	0x005C	R/W/SC
LEDC_CH5_CONF0_REG	通道 5 的配置寄存器 0	0x0064	varies
LEDC_CH5_HPOINT_REG	通道 5 的 Hpoint 寄存器	0x0068	R/W
LEDC_CH5_DUTY_REG	通道 5 的初始占空比	0x006C	R/W
LEDC_CH5_CONF1_REG	通道 5 的配置寄存器 1	0x0070	R/W/SC
LEDC_CH6_CONF0_REG	通道 6 的配置寄存器 0	0x0078	varies
LEDC_CH6_HPOINT_REG	通道 6 的 Hpoint 寄存器	0x007C	R/W
LEDC_CH6_DUTY_REG	通道 6 的初始占空比	0x0080	R/W
LEDC_CH6_CONF1_REG	通道 6 的配置寄存器 1	0x0084	R/W/SC
LEDC_CH7_CONF0_REG	通道 7 的配置寄存器 0	0x008C	varies
LEDC_CH7_HPOINT_REG	通道 7 的 Hpoint 寄存器	0x0090	R/W
LEDC_CH7_DUTY_REG	通道 7 的初始占空比	0x0094	R/W
LEDC_CH7_CONF1_REG	通道 7 的配置寄存器 1	0x0098	R/W/SC
LEDC_TIMER0_CONF_REG	配置定时器 0	0x00A0	varies
LEDC_TIMER1_CONF_REG	配置定时器 1	0x00A8	varies
LEDC_TIMER2_CONF_REG	配置定时器 2	0x00B0	varies
LEDC_TIMER3_CONF_REG	配置定时器 3	0x00B8	varies

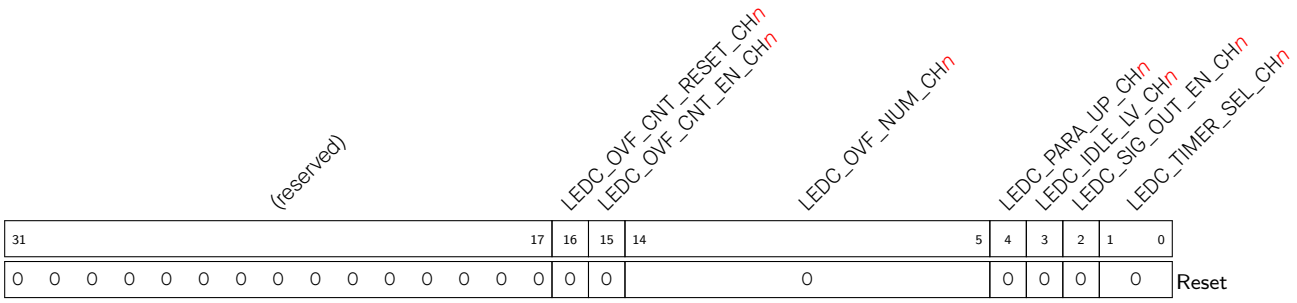
名称	描述	地址	访问
LEDC_CH0_GAMMA_CONF_REG	通道 0 伽马配置寄存器	0x0100	varies
LEDC_CH1_GAMMA_CONF_REG	通道 1 伽马配置寄存器	0x0104	varies
LEDC_CH2_GAMMA_CONF_REG	通道 2 伽马配置寄存器	0x0108	varies
LEDC_CH3_GAMMA_CONF_REG	通道 3 伽马配置寄存器	0x010C	varies
LEDC_CH4_GAMMA_CONF_REG	通道 4 伽马配置寄存器	0x0110	varies
LEDC_CH5_GAMMA_CONF_REG	通道 5 伽马配置寄存器	0x0114	varies
LEDC_CH6_GAMMA_CONF_REG	通道 6 伽马配置寄存器	0x0118	varies
LEDC_CH7_GAMMA_CONF_REG	通道 7 伽马配置寄存器	0x011C	varies
LEDC_EVT_TASK_EN0_REG	LEDC 事件任务使能寄存器 0	0x0120	R/W
LEDC_EVT_TASK_EN1_REG	LEDC 事件任务使能寄存器 1	0x0124	R/W
LEDC_EVT_TASK_EN2_REG	LEDC 事件任务使能寄存器 2	0x0128	R/W
LEDC_TIMER0_CMP_REG	定时器 0 的比较值	0x0140	R/W
LEDC_TIMER1_CMP_REG	定时器 1 的比较值	0x0144	R/W
LEDC_TIMER2_CMP_REG	定时器 2 的比较值	0x0148	R/W
LEDC_TIMER3_CMP_REG	定时器 3 的比较值	0x014C	R/W
LEDC_CONF_REG	LEDC 全局配置寄存器	0x0170	R/W
状态寄存器			
LEDC_CH0_DUTY_R_REG	通道 0 的当前占空比	0x0010	RO
LEDC_CH1_DUTY_R_REG	通道 1 的当前占空比	0x0024	RO
LEDC_CH2_DUTY_R_REG	通道 2 的当前占空比	0x0038	RO
LEDC_CH3_DUTY_R_REG	通道 3 的当前占空比	0x004C	RO
LEDC_CH4_DUTY_R_REG	通道 4 的当前占空比	0x0060	RO
LEDC_CH5_DUTY_R_REG	通道 5 的当前占空比	0x0074	RO
LEDC_CH6_DUTY_R_REG	通道 6 的当前占空比	0x0088	RO
LEDC_CH7_DUTY_R_REG	通道 7 的当前占空比	0x009C	RO
LEDC_TIMER0_VALUE_REG	配置定时器 0 的当前计数值	0x00A4	RO
LEDC_TIMER1_VALUE_REG	配置定时器 1 的当前计数值	0x00AC	RO
LEDC_TIMER2_VALUE_REG	配置定时器 2 的当前计数值	0x00B4	RO
LEDC_TIMER3_VALUE_REG	配置定时器 3 的当前计数值	0x00BC	RO
LEDC_TIMER0_CNT_CAP_REG	抓取定时器 0 的计数值	0x0150	RO
LEDC_TIMER1_CNT_CAP_REG	抓取定时器 1 的计数值	0x0154	RO
LEDC_TIMER2_CNT_CAP_REG	抓取定时器 2 的计数值	0x0158	RO
LEDC_TIMER3_CNT_CAP_REG	抓取定时器 3 的计数值	0x015C	RO
中断寄存器			
LEDC_INT_RAW_REG	原始中断寄存器	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	屏蔽中断寄存器	0x00C4	RO
LEDC_INT_ENA_REG	中断使能寄存器	0x00C8	R/W
LEDC_INT_CLR_REG	中断清除寄存器	0x00CC	WT
版本寄存器			
LEDC_DATE_REG	版本控制寄存器	0x0174	R/W

58.9 寄存器

本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 58.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14**n*)



LEDC_TIMER_SEL_CH*n* 配置通道 *n* 选取的定时器。

0：定时器 0

1：定时器 1

2：定时器 2

3：定时器 3

(R/W)

LEDC_SIG_OUT_EN_CH*n* 配置是否开启通道 *n* 的信号输出。

0：关闭

1：开启

(R/W)

LEDC_IDLE_LV_CH*n* 配置通道 *n* 关闭（即 LEDC_SIG_OUT_EN_CH*n* 为 0）时的输出电平值。

0：低电平

1：高电平

(R/W)

LEDC_PARA_UP_CH*n* 配置是否更新以下字段：LEDC_HPOINT_CH*n*、LEDC_DUTY_START_CH*n*、LEDC_SIG_OUT_EN_CH*n*、LEDC_TIMER_SEL_CH*n*、LEDC_OVF_CNT_EN_CH*n* 和占空比区间配置

0：无效值，没有作用

1：更新

该字段由硬件自清

(WT)

LEDC_OVF_NUM_CH*n* 配置定时器的最大溢出次数减 1。

通道 *n* 的定时器溢出次数达到 (LEDC_OVF_NUM_CH*n* + 1) 次时，触发 LEDC_OVF_CNT_CH*n*_INT 中断。(R/W)

LEDC_OVF_CNT_EN_CH*n* 配置是否开启通道 *n* 的溢出次数计数器。

0：关闭

1：开启

(R/W)

见下页...

Register 58.1. LEDC_CH*n*_CONF0_REG (*n*: 0-7) (0x0000+0x14**n*)

接上页...

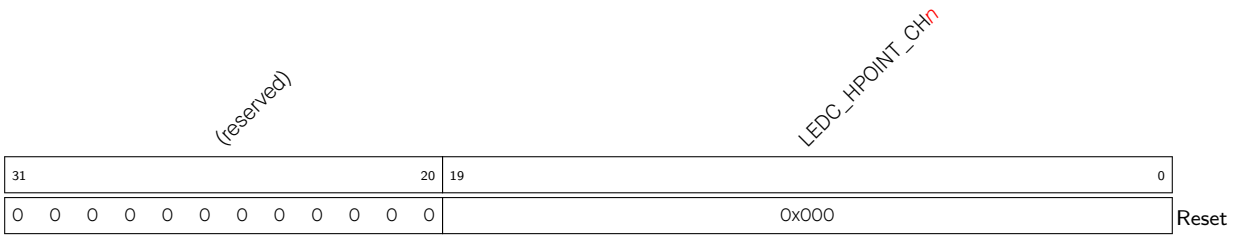
LEDC_OVF_CNT_RESET_CH*n* 配置是否复位通道 *n* 的溢出次数计数器。

0：无效值，没有作用

1：复位

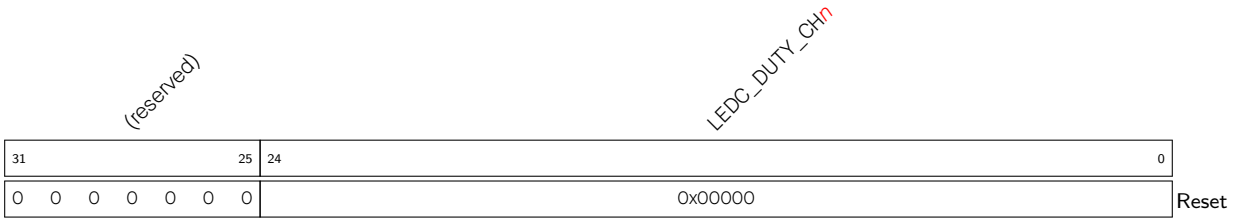
(WT)

Register 58.2. LEDC_CH*n*_HPOINT_REG (*n*: 0-7) (0x0004+0x14**n*)



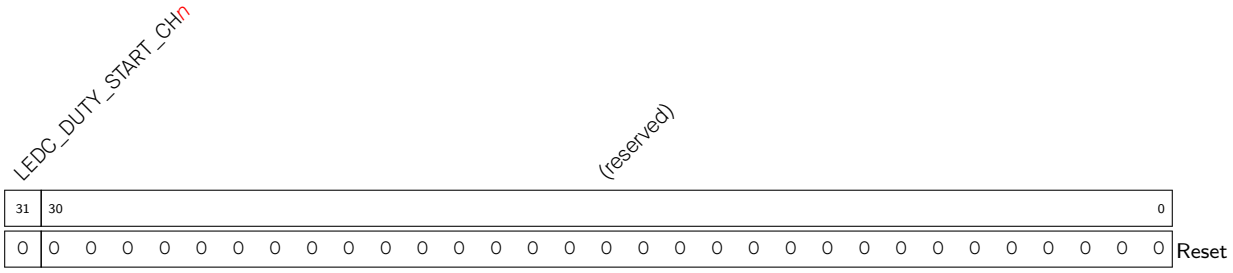
LEDC_HPOINT_CH*n* 配置通道 *n* 输出的 Hpoint 值。当定时器达到该寄存器的规定值时，输出值变为高电平。(R/W)

Register 58.3. LEDC_CH*n*_DUTY_REG (*n*: 0-7) (0x0008+0x14**n*)



LEDC_DUTY_CH*n* 配置通道 *n* 输出信号的占空比。(R/W)

Register 58.4. LEDC_CH*n*_CONF1_REG (*n*: 0-7) (0x000C+0x14**n*)



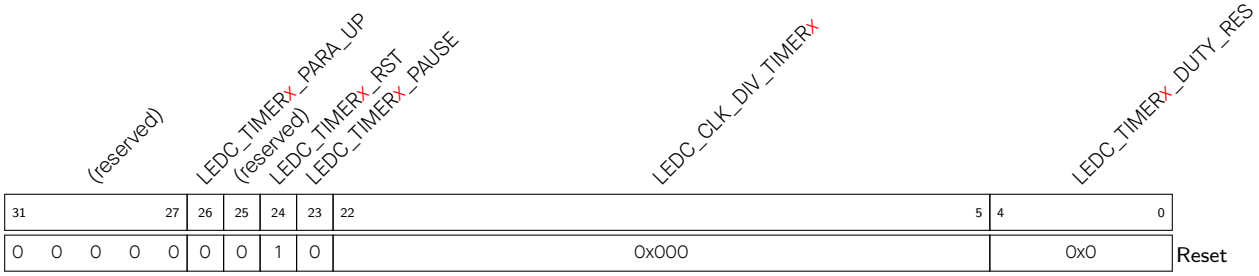
LEDC_DUTY_START_CH*n* 配置占空比渐变配置是否生效。

0：不生效

1：生效

(R/W/SC)

Register 58.5. LEDC_TIMER*x*_CONF_REG (*x*: 0-3) (0x00A0+0x8**x*)



LEDC_TIMER*x*_DUTY_RES 配置定时器 *x* 计数器的位宽，有效值为 1 到 20。(R/W)

LEDC_CLK_DIV_TIMER*x* 配置定时器 *x* 中分频器的分频系数。

低 8 位为小数部分，高 10 位为整数部分。(R/W)

LEDC_TIMER*x*_PAUSE 配置是否暂停定时器 *x* 的计数器。

0：不暂停

1：暂停

(R/W)

LEDC_TIMER*x*_RST 配置是否复位定时器 *x* 的计数器（复位后，计数器的值为 0。）。

0：不复位

1：复位

(R/W)

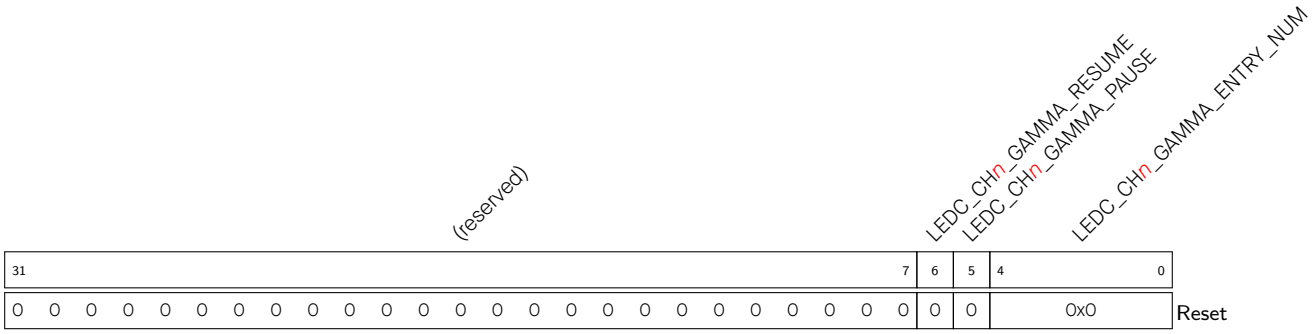
LEDC_TIMER*x*_PARA_UP 配置是否更新 LEDC_CLK_DIV_TIMER*x* 和 LEDC_TIMER*x*_DUTY_RES 字段。

0：无效值，没有作用

1：更新

(WT)

Register 58.6. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-7) (0x0100+0x4**n*)



LEDC_CH*n*_GAMMA_ENTRY_NUM 配置通道 *n* 的占空比渐变区间数量。(R/W)

LEDC_CH*n*_GAMMA_PAUSE 配置是否暂停占空比渐变。

0：无效值，没有作用

1：暂停

(WT)

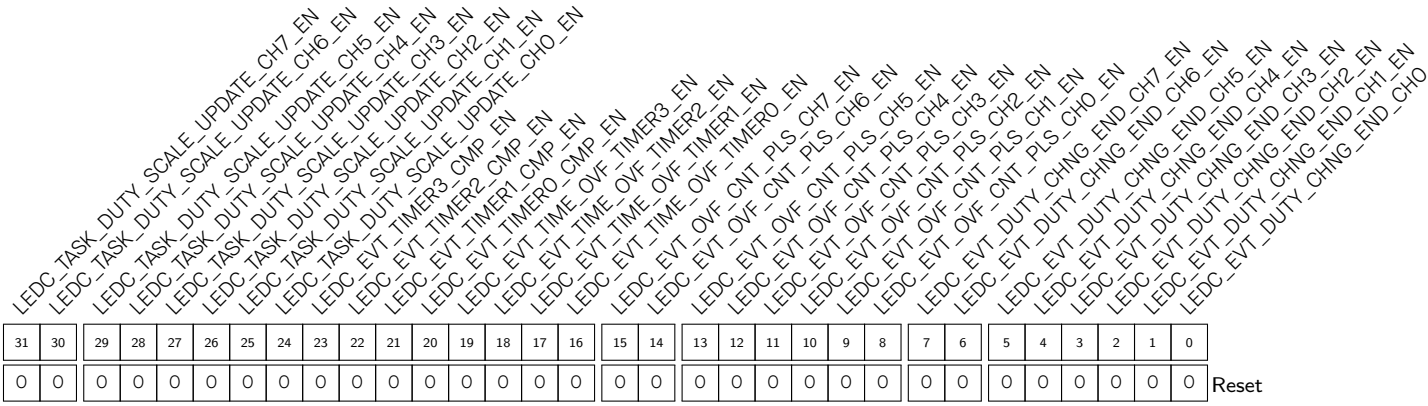
LEDC_CH*n*_GAMMA_RESUME 配置是否恢复已暂停的占空比渐变。

0：无效值，没有作用

1：恢复

(WT)

Register 58.7. LEDC_EVT_TASK_EN0_REG (0x0120)



LEDC_EVT_DUTY_CHNG_END_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_EVT_DUTY_CHNG_END_CH*n* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_OVF_CNT_PLS_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_EVT_OVF_CNT_PLS_CH*n* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_TIME_OVF_TIMER*x*_EN (*x*: 0-3) 配置是否使能 LEDC_EVT_TIME_OVF_TIMER*x* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_TIME*x*_CMP_EN (*x*: 0-3) 配置是否使能 LEDC_EVT_TIMER*x*_CMP 事件。

0：关闭

1：使能

(R/W)

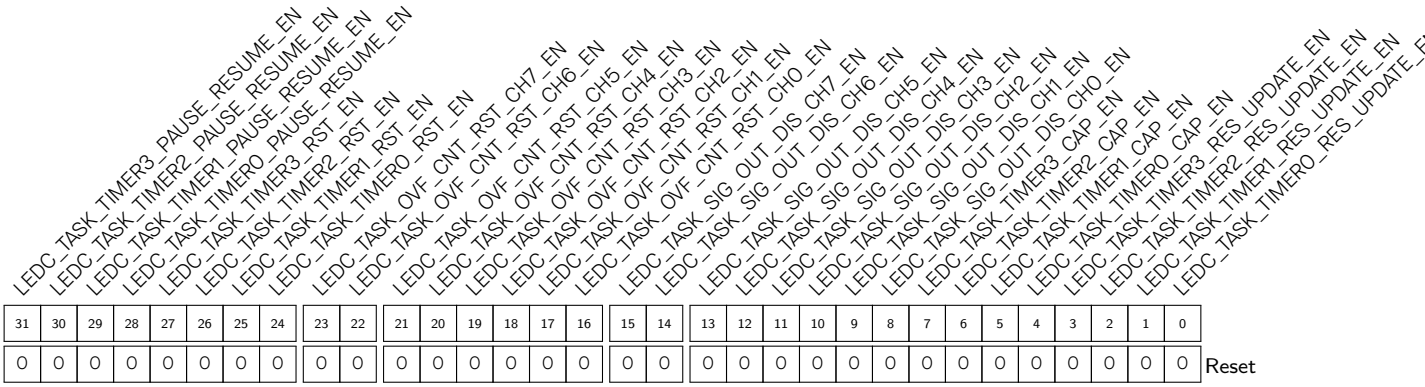
LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* 事件。

0：关闭

1：使能

(R/W)

Register 58.8. LEDC_EVT_TASK_EN1_REG (0x0124)



LEDC_TASK_TIMER*x*_RES_UPDATE_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_RES_UPDATE 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_TIMER*x*_CAP_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_CAP 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_SIG_OUT_DIS_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_TASK_SIG_OUT_DIS_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_OVF_CNT_RST_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_TASK_OVF_CNT_RST_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_TIMER*x*_RST_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_RST 任务。

0：关闭

1：使能

(R/W)

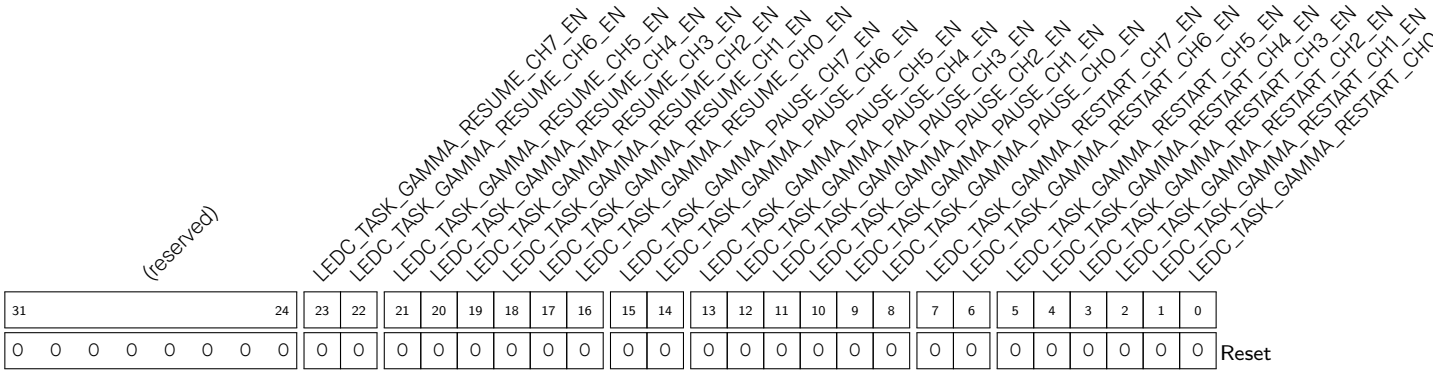
LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_PAUSE 和 LEDC_TASK_TIMER*x*_RESUME 任务。

0：关闭

1：使能

(R/W)

Register 58.9. LEDC_EVT_TASK_EN2_REG (0x0128)



LEDC_TASK_GAMMA_RESTART_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_TASK_GAMMA_RESTART_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_GAMMA_PAUSE_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_TASK_GAMMA_PAUSE_CH*n* 任务。

0：关闭

1：使能

(R/W)

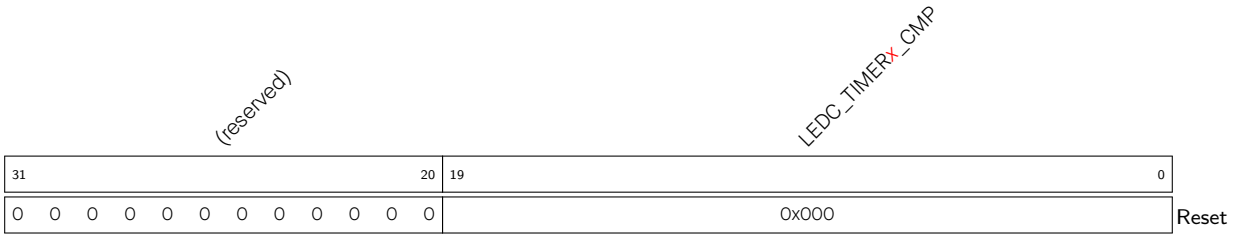
LEDC_TASK_GAMMA_RESUME_CH*n*_EN (*n*: 0-7) 配置是否使能 LEDC_TASK_GAMMA_RESUME_CH*n* 任务。

0：关闭

1：使能

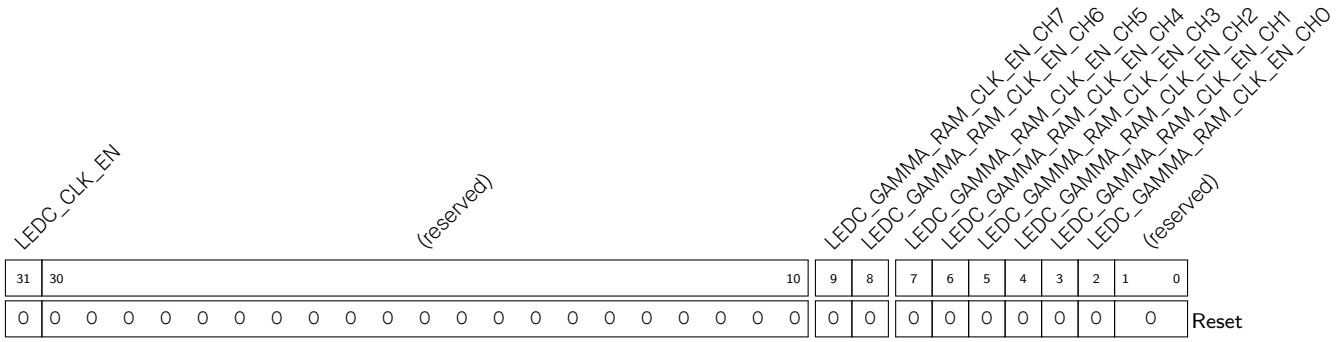
(R/W)

Register 58.10. LEDC_TIMER*x*_CMP_REG (*x*: 0-3) (0x0140+0x4**x*)



LEDC_TIMER*x*_CMP 配置 LEDC 定时器 *x* 的比较值。(R/W)

Register 58.11. LEDC_CONF_REG (0x0170)



LEDC_GAMMA_RAM_CLK_EN_CH*n* (*n*: 0-7) 配置是否开启通道 *n* 伽马 RAM 时钟门控。

0：仅在应用读或写通道 *n* 的伽马 RAM 时开启时钟门控

1：强制为通道 *n* 伽马 RAM 开启时钟门控

(R/W)

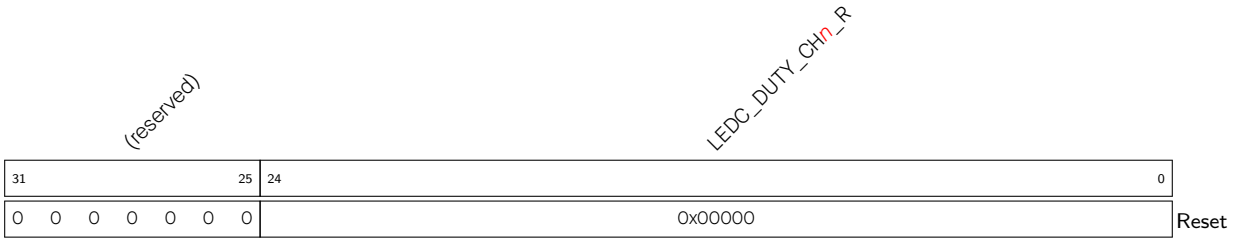
LEDC_CLK_EN 配置是否开启寄存器门控。

0：仅在应用写寄存器时开启时钟门控

1：强制为寄存器开启时钟门控

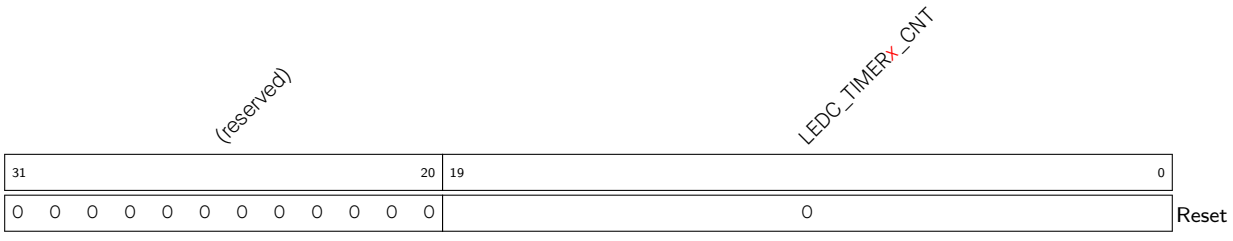
(R/W)

Register 58.12. LEDC_CH*n*_DUTY_R_REG (*n*: 0-7) (0x0010+0x14*n)



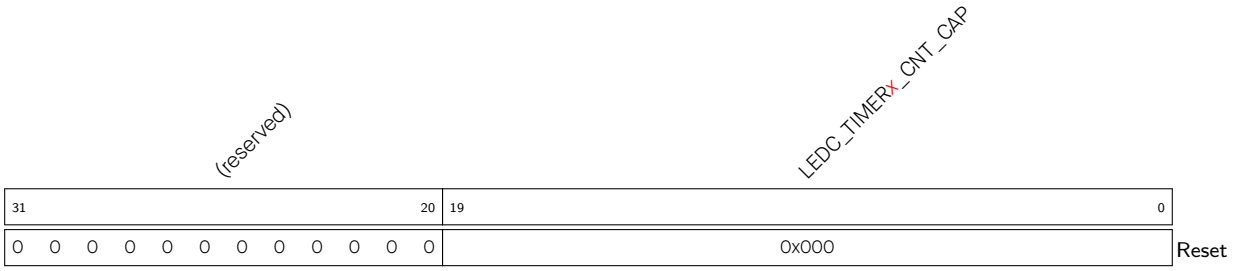
LEDC_DUTY_CH*n*_R 通道 *n* 输出信号的当前占空比。(RO)

Register 58.13. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8*x)



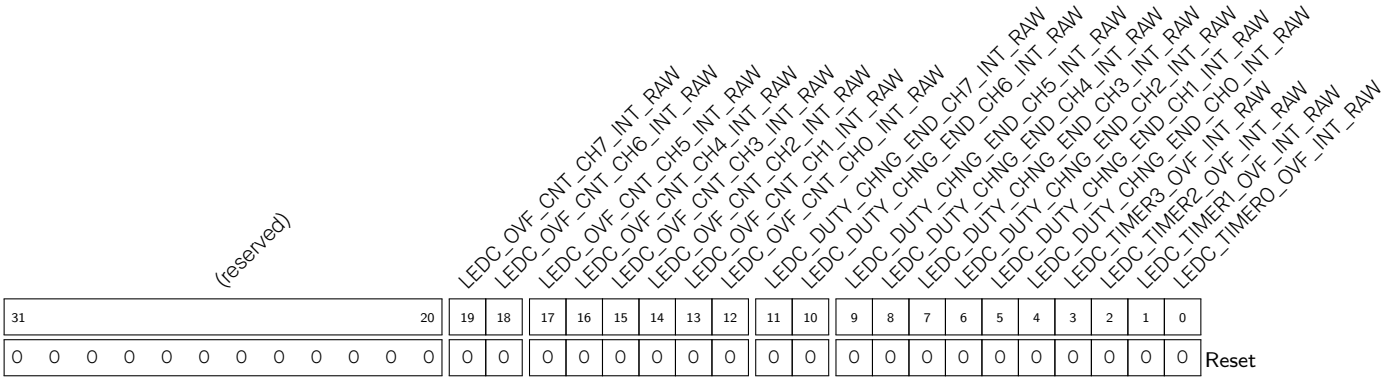
LEDC_TIMER*x*_CNT 定时器 *x* 计数器的当前数值。(RO)

Register 58.14. LEDC_TIMERx_CNT_CAP_REG (x: 0-3) (0x0150+0x4*x)



LEDC_TIMERx_CNT_CAP 抓取的定时器 x 的计数器数值。(RO)

Register 58.15. LEDC_INT_RAW_REG (0x00C0)

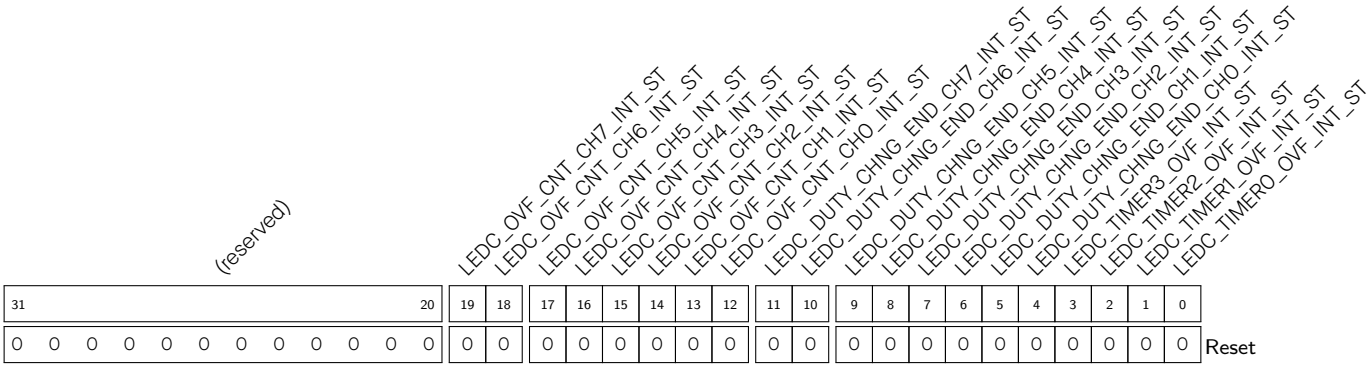


LEDC_TIMERx_OVF_INT_RAW (x: 0-3) 原始状态位：LEDC_TIMERx_OVF_INT 的原始中断状态。定时器 x 的计数器到达最大值时，触发该中断。(R/WTC/SS)

LEDC_DUTY_CHNG_END_CHn_INT_RAW (n: 0-7) 原始状态位：LEDC_DUTY_CHNG_END_CHn_INT 的原始中断状态。占空比渐变结束时会触发该中断。(R/WTC/SS)

LEDC_OVF_CNT_CHn_INT_RAW (n: 0-7) 原始状态位：LEDC_OVF_CNT_CHn_INT 的原始中断状态。当 ovf_cnt 达到 LEDC_OVF_NUM_CHn 的设定值时，触发该中断。(R/WTC/SS)

Register 58.16. LEDC_INT_ST_REG (0x00C4)



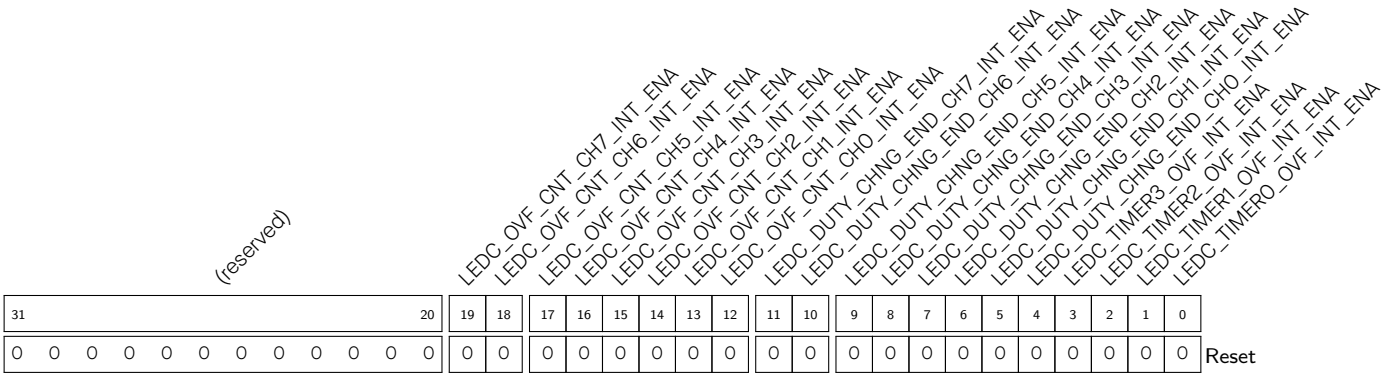
LEDC_TIMER*x*_OVF_INT_ST (*x*: 0-3) 屏蔽状态位：LEDC_TIMER*x*_OVF_INT 的屏蔽中断状态。仅在 LEDC_TIMER*x*_OVF_INT_ENA 为 1 时有效。(RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST (*n*: 0-7) 屏蔽状态位：LEDC_DUTY_CHNG_END_CH*n*_INT 的屏蔽中断状态。

仅在 LEDC_DUTY_CHNG_END_CH*n*_INT_ENA 为 1 时有效。(RO)

LEDC_OVF_CNT_CH*n*_INT_ST (*n*: 0-7) 屏蔽状态位：LEDC_OVF_CNT_CH*n*_INT 的屏蔽中断状态。仅在 LEDC_OVF_CNT_CH*n*_INT_ENA 为 1 时有效。(RO)

Register 58.17. LEDC_INT_ENA_REG (0x00C8)

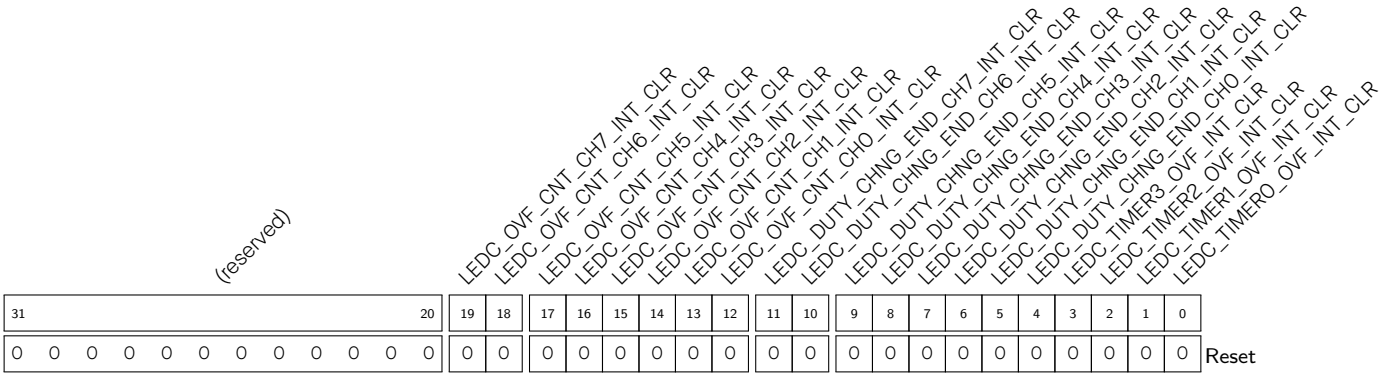


LEDC_TIMER*x*_OVF_INT_ENA (*x*: 0-3) 使能位：写 1 使能 LEDC_TIMER*x*_OVF_INT。(R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA (*n*: 0-7) 使能位：写 1 使能 LEDC_DUTY_CHNG_END_CH*n*_INT。(R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA (*n*: 0-7) 使能位：写 1 使能 LEDC_OVF_CNT_CH*n*_INT。(R/W)

Register 58.18. LEDC_INT_CLR_REG (0x00CC)

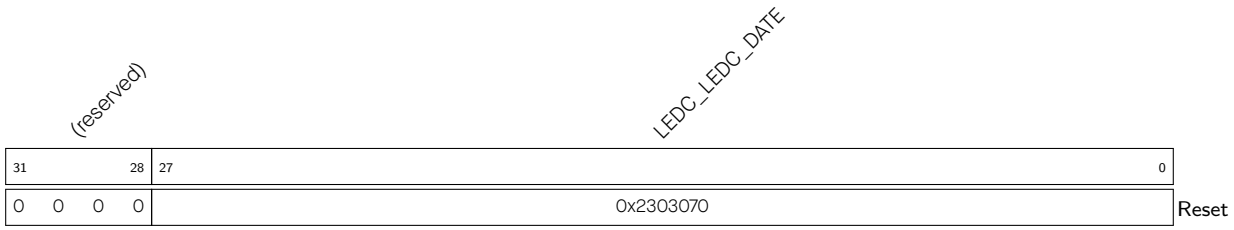


LEDC_TIMER*x*_OVF_INT_CLR (*x*: 0-3) 清零位：写 1 清零 LEDC_TIMER*x*_OVF_INT。(WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR (*n*: 0-7) 清零位：写 1 清零 LEDC_DUTY_CHNG_END_CH*n*_INT。(WT)

LEDC_OVF_CNT_CH*n*_INT_CLR (*n*: 0-7) 清零位：写 1 清零 LEDC_OVF_CNT_CH*n*_INT。(WT)

Register 58.19. LEDC_DATE_REG (0x0174)



LEDC_LEDC_DATE 版本控制寄存器。(R/W)

第 59 章

电机控制脉宽调制器 (MCPWM)

59.1 概述

电机控制脉宽调制器 (MCPWM) 外设用于电机和电源控制。该外设提供了 6 个 PWM 输出，可在几种拓扑结构中运行。常见的拓扑结构之一是用一对 PWM 输出来驱动 H 桥以控制电机旋转速度和旋转方向。

MCPWM 外设主要分为五个子模块：PWM 定时器、PWM 操作器、捕获模块、事件矩阵 (Event Task Matrix, ETM) 模块、故障检测模块。PWM 定时器提供参考时序，可以自由运行，或同步到其他定时器或外部源。PWM 操作器包含为一个 PWM 通道生成波形对的所有控制资源。捕获模块用于需要精确定时外部事件的系统。ETM 模块对输入到 MCPWM 的任务做出相应的响应动作，根据不同的运动状态产生相应的事件。故障检测模块用于捕获外部故障，使得系统可以选择做出响应。

ESP32-P4 有两个 MCPWM 外设，即 MCPWM0 和 MCPWM1。

59.2 主要特性

MCPWM 外设有一个时钟分频器（预分频器）、三个 PWM 定时器、三个 PWM 操作器、一个捕获模块、一个 ETM 模块和一个故障检测模块。MCPWM 的主时钟可以从三个时钟源中任选其一，分别为：PLL_F160M_CLK, XTAL_CLK 和 RC_FAST_CLK（通过 HP_SYS_CLKRST_PERI_CLK_CTRL20_REG 寄存器的 HP_SYS_CLKRST_MCPWM*n*_CLK_SRC_SEL 字段配置）。图 59.2-1 描述了 MCPWM 内的模块和接口上的信号。PWM 定时器用于生成定时参考，PWM 操作器将根据生成的定时参考生成所需的波形。通过配置，任一 PWM 操作器可以使用任一 PWM 定时器的定时参考。不同的 PWM 操作器可以使用相同的 PWM 定时器的定时参考产生 PWM 信号，也可以使用不同的 PWM 定时器的值来生成单独的 PWM 信号。不同的 PWM 定时器可以进行同步。

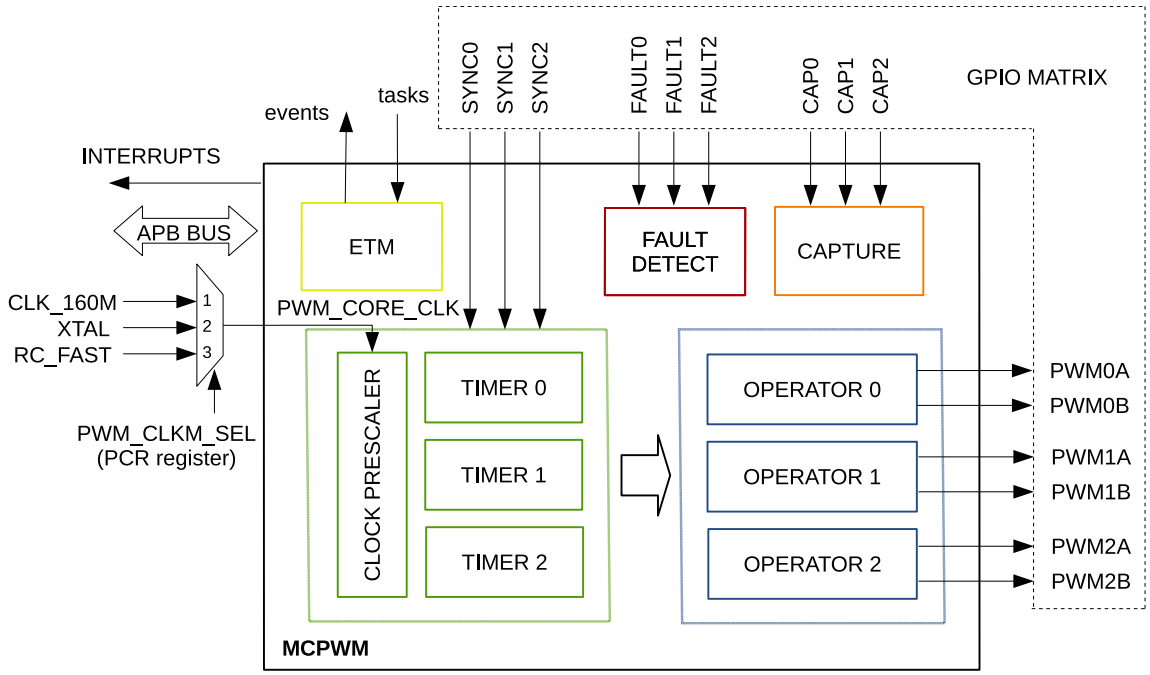


图 59.2-1. MCPWM 外设概览

以下是图 59.2-1 中模块的功能概述：

- PWM 定时器 0、1 和 2：
  - 每个 PWM 定时器都有一个专用的 8 位时钟预分频器。
  - PWM 定时器中的 16 位计数器的工作模式包括：递增计数模式，递减计数模式，递增递减循环计数模式。
  - 硬件/软件同步可以触发 PWM 定时器重载，重载值位于相位寄存器中；同时触发预分频的重启，从而同步定时器的时钟。硬件同步源可以来自任何 GPIO 或任何其他 PWM 定时器的 sync_out 信号。软件同步源通过向 MCPWM_TIMER*n*_SYNC_SW 位写入取反值获取。
- PWM 操作器 0、1 和 2：
  - 每个 PWM 操作器有两个 PWM 输出（PWM*x*A 和 PWM*x*B），可以在对称和非对称配置中独立工作。
  - 可以通过异步方式更新对 PWM 信号的控制。
  - 死区时间在上升沿和下降沿可配置，并可分别设置。
  - 所有事件都可触发 CPU 中断。
  - 通过高频载波信号调制 PWM 输出，在使用变压器隔离栅极驱动器时可发挥巨大作用。
  - 周期、时间戳寄存器和其他主要的控制寄存器有影子寄存器，具有灵活的更新方式。
- 故障检测模块：
  - 出现故障时，可选择在逐周期模式或一次性模式下处理。
  - 故障条件可强制 PWM 输出高或低电平。
- 捕获模块：
  - 捕获模块的时钟为 MCPWM 的主时钟。

  - 旋转电机的速度测量。
  - 位置传感器脉冲之间的间隔时间测量。
  - 脉冲序列信号的周期和占空比测量。
  - 从电流/电压传感器的占空比编码信号导出的解码电流或电压振幅。
  - 3 个独立的捕获通道，各具备一个 32 位的时间戳寄存器。
  - 输入捕获信号可以预分频，边沿极性可选。
  - 捕获定时器可以与 PWM 定时器或外部信号同步。
  - 3 个捕获通道上都可以产生中断。
- ETM 模块：
  - 根据每个定时器和操作器不同的运行状态，产生不同的事件。
  - 每个定时器和操作器响应其对应的任务，自动进行相应的操作。
  - 每个事件和任务都可以独立使能。某个事件不使能时，不产生对应的事件；某个任务不使能时，不响应对应的任务。

59.3 模块

59.3.1 概述

每个 MCPWM 包含的关键模块为定时器模块，操作器模块，故障检测模块，捕获模块和 ETM 模块，关键模块的功能详见下文。

59.3.1.1 定时器模块

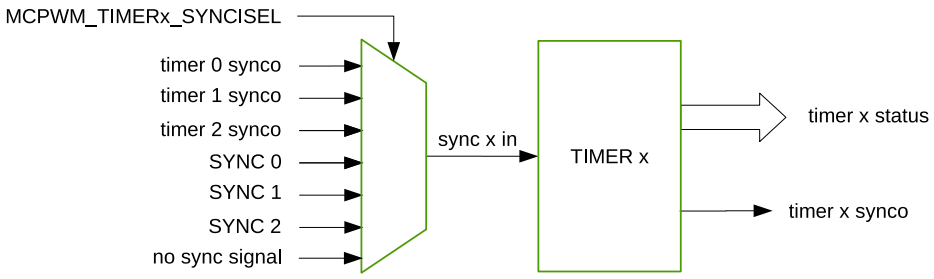


图 59.3-1. 定时器模块

该模块产生一个定时器，它可以按照指定的周期进行计数，它可以工作在递增计数模式，递减计数模式或递增递减循环计数模式下，它可以选择不同的同步输入源（共 7 个可选输入源）来进行计数值和方向的重载，从而使多个定时器彼此同步，它还可以输出一个同步输出源（共 4 个可选输出源）供其他定时器使用。

图中的 timer*x* status 指定时器模块输出的定时器状态，如计数模式，计数值等于 0 或周期等。

59.3.1.2 操作器模块

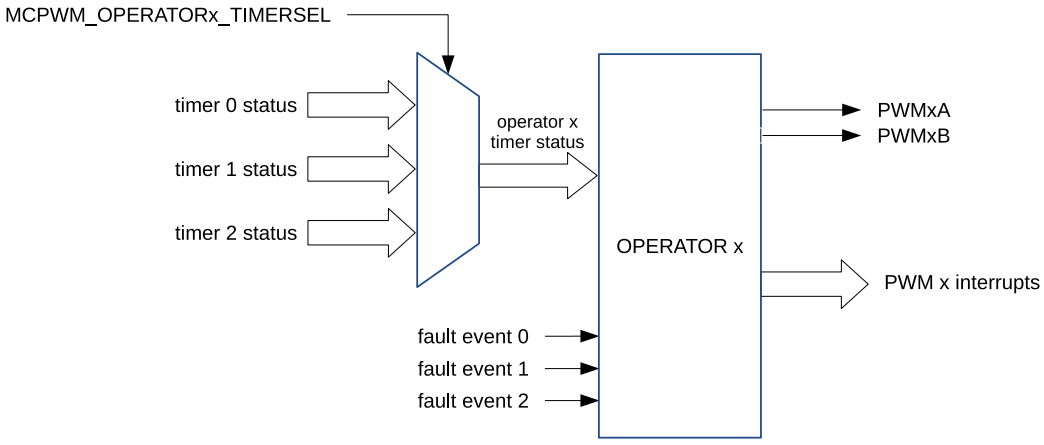


图 59.3-2. 操作器模块

一个 PWM 操作器包含一个 PWM 生成器，一个死区生成器，一个故障检测模块以及一个 PWM 载波模块，它们的功能如表 59.3-1 所示：

表 59.3-1. 操作器模块包含的各个子模块功能

子模块	功能
PWM 生成器	按照指定的占空比和波形，生成 PWM 信号
死区生成器	可以选择对 PWM 生成器产生的 PWM 信号的上升沿和下降沿添加死区（即对上升沿和下降沿添加延迟）以及进行反相等操作
PWM 载波	可以选择对死区生成器产生的 PWM 信号添加载波，载波的频率可以配置，并且添加载波后的 PWM 波形的第一个脉冲持续时间可以配置
故障检测	检测故障并生成对应的故障事件，按照指定的间隔模式（一次性模式或逐周期模式）响应故障事件以及发生故障事件时，按照指定的方式对 PWM 载波模块产生的 PWM 信号采取操作

59.3.1.3 捕获模块

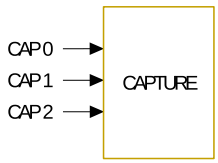


图 59.3-3. 捕获模块

该模块包含一个可以进行同步的定时器，当输入的捕获信号有效时，将捕获定时器的计数值。

59.3.1.4 ETM 模块

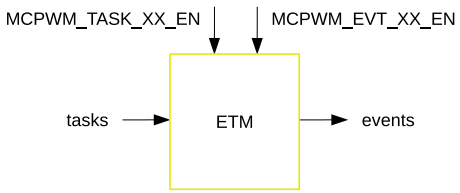


图 59.3-4. ETM 模块

该模块对接收到的任务进行响应，或产生事件并输出，可以独立配置每个事件和任务的使能。

59.3.2 PWM 定时器模块

MCPWM 外设有三个 PWM 定时器模块。它们中的任何一个都可以决定三个 PWM 操作器模块中任意一个的必要事件时序。通过使用 GPIO 交换矩阵的同步信号，内置同步逻辑允许一个或多个 MCPWM 外设中的多个 PWM 定时器模块作为一个系统协同工作。

59.3.2.1 PWM 定时器模块的配置

用户可配置 PWM 定时器模块的以下功能：

- 通过指定 PWM 定时器频率或周期来控制事件发生的频率

- 配置某个 PWM 定时器与其他 PWM 定时器或模块进行相位同步
- 配置定时器计数模式：递增，递减，或递增递减循环计数模式
- 使用预分频器更改 PWM 定时器时钟 (PT_clk) 的速率。每个定时器都有自己的预分频器，通过寄存器 MCPWM_TIMER0_CFG0_REG 的 MCPWM_TIMER*n*_PRESCALE 配置。PWM 定时器根据该寄存器的设置以较慢的速度递增或递减。当定时器停止后并再次开始计数时，新的MCPWM_TIMER*n*_PRESCALE 配置值才会生效。

59.3.2.2 PWM 定时器工作模式和定时事件生成

PWM 定时器有三种工作模式，由 PWM*x* 定时器模式字段配置：

- 递增计数模式：
  定时器从零增加到周期字段中配置的值。达到周期值后，PWM 定时器清零，并再次开始递增。PWM 周期 = 周期寄存器中的周期值 + 1。
  说明：周期寄存器为 MCPWM_TIMER*n*_PERIOD (*x* = 0, 1, 2)，对应 MCPWM_TIMER0_PERIOD、MCPWM_TIMER1_PERIOD、MCPWM_TIMER2_PERIOD。
- 递减计数模式：
  PWM 定时器从周期寄存器中的值开始递减到零。达到零后，将恢复为周期值，并再次开始递减。在这种情况下，PWM 周期 = 周期寄存器中的周期值 + 1。
- 递增-递减循环模式：
  此模式结合了上述两种模式。PWM 定时器从零开始递增，直到达到周期值，再次递减为零。PWM 定时器按照此模式循环递增递减。PWM 周期 = 周期寄存器的周期值 × 2。

图 59.3-5 至 59.3-8 显示不同模式下的 PWM 定时器波形，包括同步事件期间的定时器行为。递增计数模式中，同步后永远保持递增计数；递减计数模式中，同步后永远保持递减计数；递增-递减循环模式中，同步后由
MCPWM_TIMER*n*_PHASE_DIRECTION 配置计数方向。

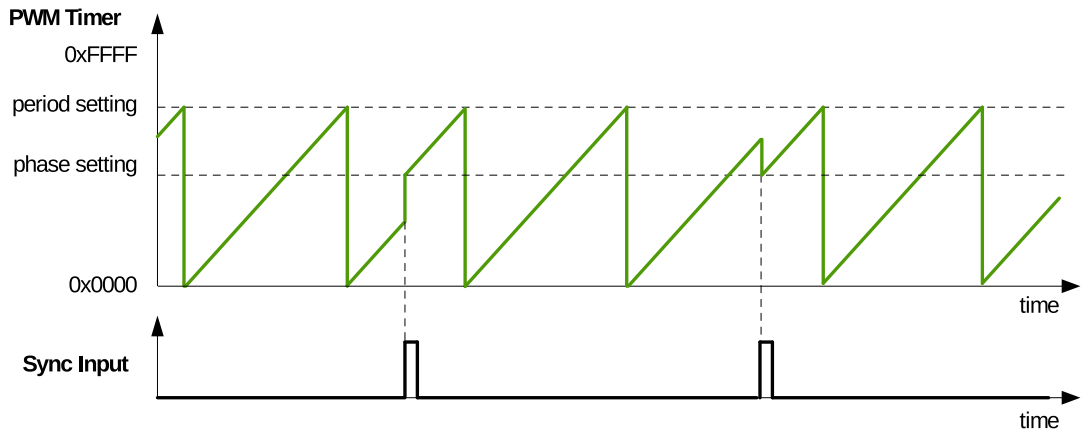


图 59.3-5. 递增计数模式波形

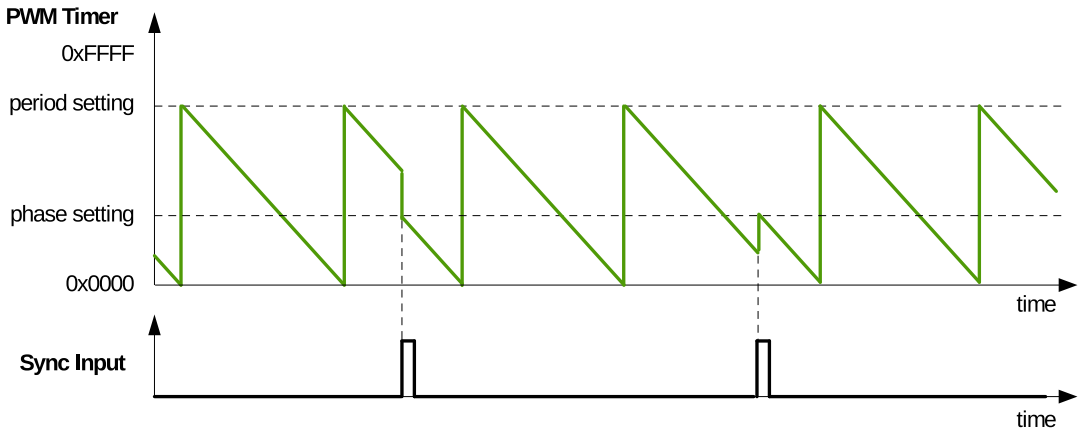


图 59.3-6. 递减计数模式波形

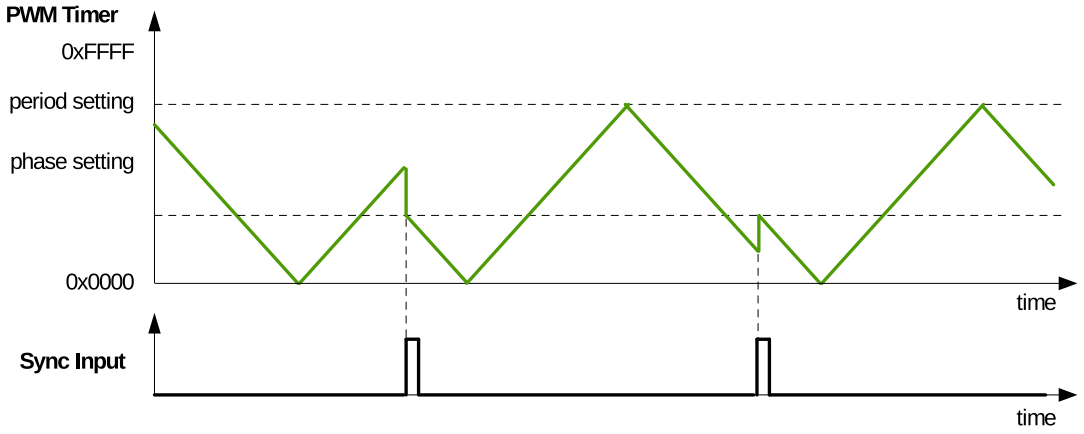


图 59.3-7. 递增递减循环模式波形，同步事件后递减

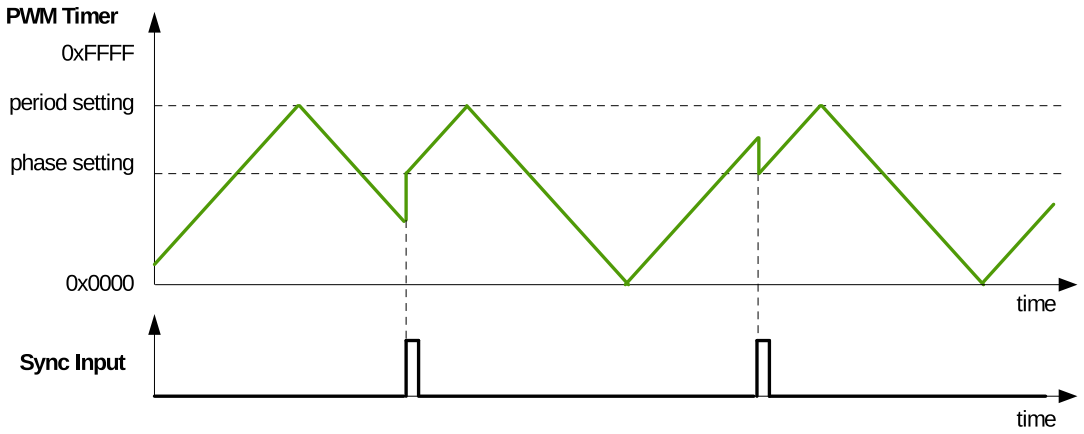


图 59.3-8. 递增递减循环模式波形，同步事件后递增

PWM 定时器运行时，定期自动生成以下定时事件：

- UTEP：当 PWM 定时器等于周期字段的值 (MCPWM_TIMER*n*_PERIOD) 且 PWM 定时器递增计数时，生成的定时事件。
- UTEZ：当 PWM 定时器等于零且 PWM 定时器递增计数时，生成的定时事件。
- DTEP：当 PWM 定时器等于周期字段的值 (MCPWM_TIMER*n*_PERIOD) 且 PWM 定时器递减时，生成的定时事件。
- DTEZ：当 PWM 定时器等于零且 PWM 定时器递减时，生成的定时事件。

图 59.3-9 至 59.3-11 为 U/DTEP 和 U/DTEZ 定时事件的时序波形。

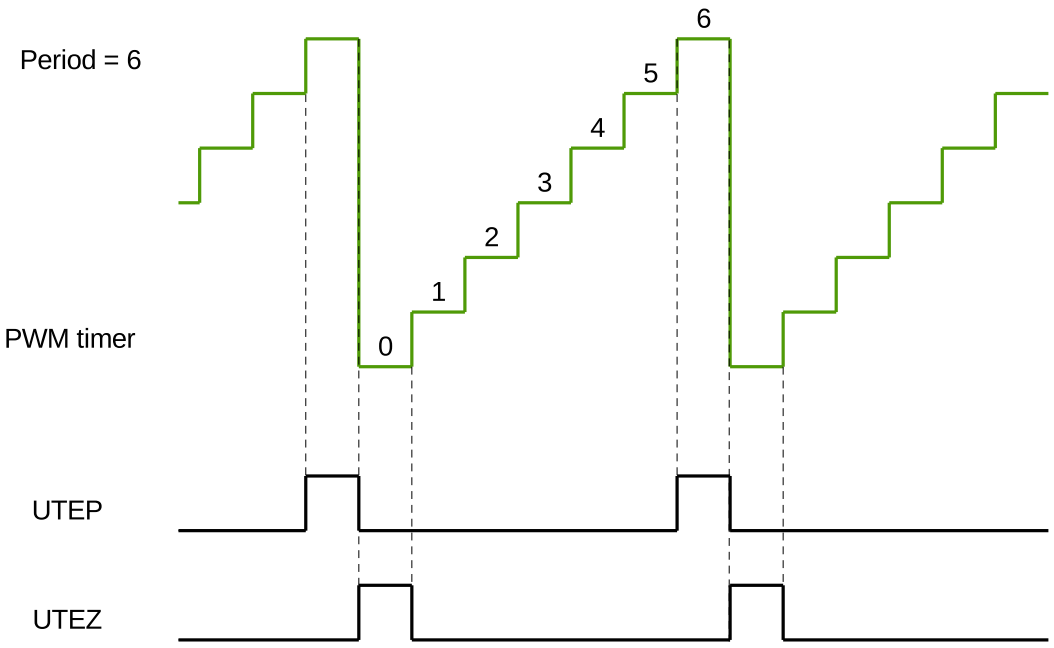


图 59.3-9. 递增模式中生成的 UTEP 和 UTEZ

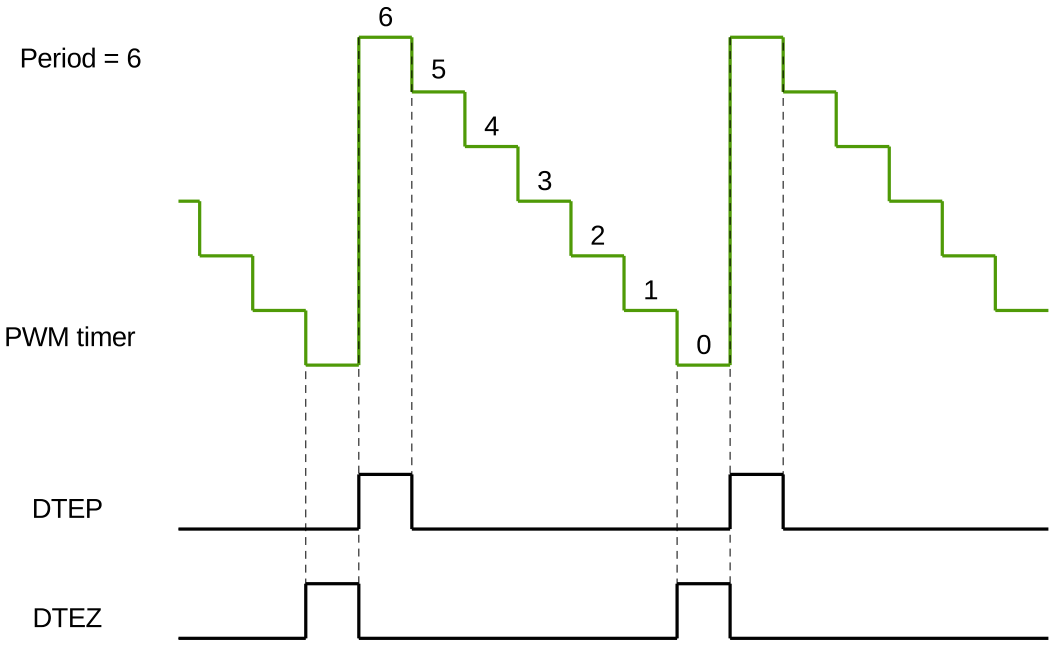


图 59.3-10. 递减模式中生成的 DTEP 和 DTEZ

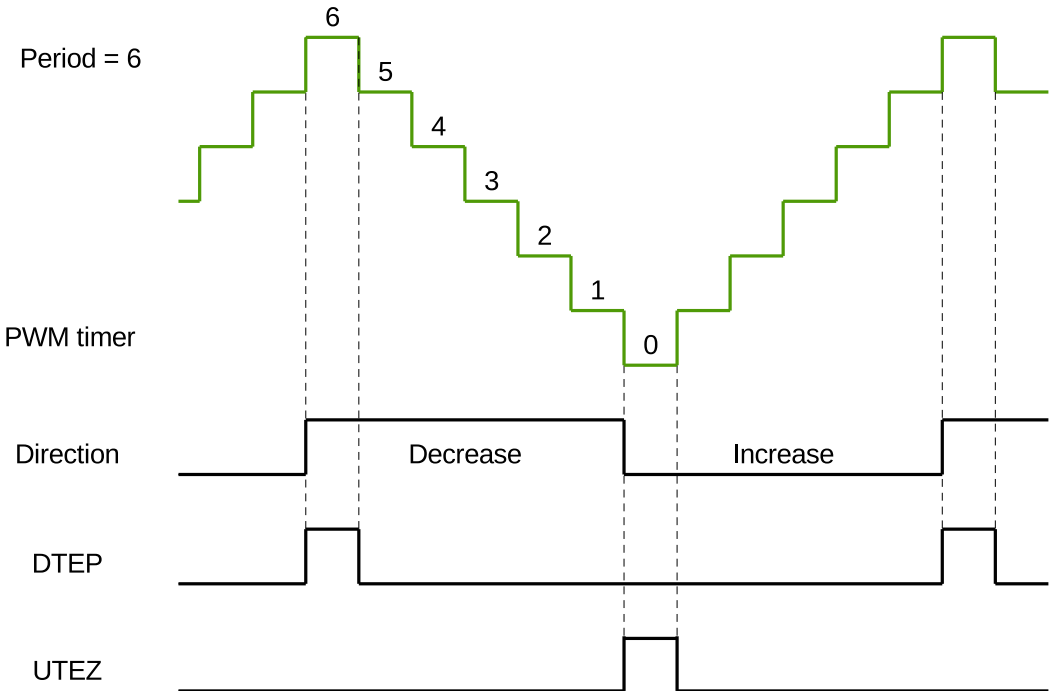


图 59.3-11. 递增递减模式中生成的 DTEP 和 UTEZ

需注意，在递增递减模式下，当定时器递增计数时，定时器范围为 [0，周期值 - 1]；递减计数时，定时器范围为 [周期值，1]。也就是说，在该模式下，将定时器同步为 0 时，则不支持递减计数，即 MCPWM_TIMER*n*_PHASE_DIRECTION 不能为 1；定时器同步为周期值时，也不支持递增计数，即 MCPWM_TIMER*n*_PHASE_DIRECTION 不能为 0。所以，当定时器同步为 0 时，计数方向只能为递增，即将 MCPWM_TIMER*n*_PHASE_DIRECTION 配置为 0；定时器同步为周期值时，计数方向只能为递减，即将 MCPWM_TIMER*n*_PHASE_DIRECTION 配置为 1。

59.3.2.3 PWM 定时器影子寄存器

PWM 定时器周期寄存器和 PWM 定时器时钟预分频器寄存器配有影子寄存器。影子寄存器能够备份即将写入有效寄存器的值，并在硬件同步的特定时刻写入有效寄存器。有效寄存器和影子寄存器的功能如下：

- 有效寄存器：直接控制硬件执行的所有操作。
- 影子寄存器：作为临时缓冲区，存储即将写入有效寄存器的值。在用户配置的某个时间点，影子寄存器中的值被写入有效寄存器。在此之前，影子寄存器的内容对受控硬件没有任何直接影响，防止寄存器由软件异步修改时可能发生的错误硬件操作。影子寄存器和有效寄存器具有相同的存储器地址。软件写入或读取影子寄存器。

  当定时器开始工作时，时钟预分频器的有效寄存器更新。当 MCPWM_GLOBAL_UP_EN 置 1 时，可通过以下方式选择更新有效寄存器的时间点：

  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 0，立即更新。
  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 1，当 PWM 定时器值为 0 时，开始更新。
  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 2，当 PWM 定时器同步时，开始更新。
  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 3，当 PWM 定时器值为 0 时或同步时，开始更新。
  - 软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。

59.3.2.4 PWM 定时器同步和锁相

PWM 模块采用灵活的同步方法。每个 PWM 定时器都有一个同步输入和一个同步输出。同步输入可以从 GPIO 矩阵的三个同步输出和三个 PWM 定时器的三个同步输出信号中选择。同步输出可以在 PWM 定时器等于周期、PWM 定时器等于零、同步输入有效或软件同步时生成。因此，每个 PWM 定时器都可以灵活的选择同步源，从而实现多种相位同步方式。在同步时，PWM 定时器的预分频器会复位其计数器，并重新开始对时钟的分频，以保证定时器的正确性。

59.3.3 PWM 操作器模块

PWM 操作器模块具备以下功能：

- 根据相应 PWM 定时器的定时参考生成 PWM 信号对。
- PWM 信号对的每个信号都可以独立设置特定的死区（即对 PWM 信号的上升沿和下降沿添加延迟）。
- 可通过配置将载波叠加到 PWM 信号上。
- 故障条件下处理响应。

图 59.3-12 为 PWM 操作器的框图。

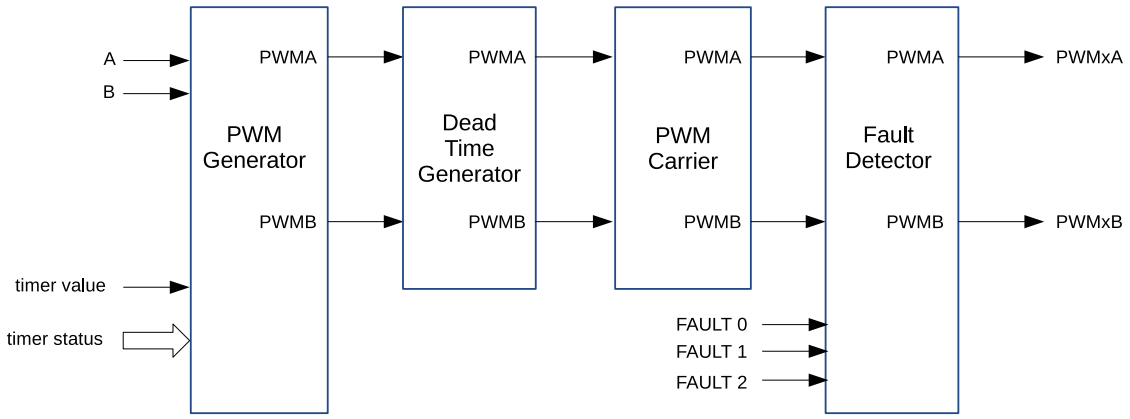


图 59.3-12. PWM 操作器的子模块

59.3.3.1 PWM 生成器模块

PWM 生成器模块的作用

此模块可以生成或导入重要的时序事件，转化为特定操作后，在 PWM*x*A 和 PWM*x*B 输出处生成所需的波形。

PWM 生成器模块执行以下操作：

- 基于使用寄存器 A 和 B 配置的时间戳生成定时事件。满足以下条件时发生定时事件（寄存器 A 和 B 的配置方法在章节 59.3.3.1 描述）：
  - UTEA：PWM 定时器递增计数并且其值等于寄存器 A。
  - UTEB：PWM 定时器递增计数并且其值等于寄存器 B。
  - DTEA：PWM 定时器递减计数并且其值等于寄存器 A。
  - DTEB：PWM 定时器递减计数并且其值等于寄存器 B。
- 基于故障或同步事件生成 U/DTO、U/DT1 定时事件。
  - UTO：定时器递增计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_TO_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_TO_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_TO_SEL 置 2）或同步（字段 MCPWM_GEN*n*_TO_SEL 置 3）。
  - UT1：定时器递增计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_T1_SEL 置 2）或同步（字段 MCPWM_GEN*n*_T1_SEL 置 3）。
  - DTO：定时器递减计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_TO_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_TO_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_TO_SEL 置 2）或同步（字段 MCPWM_GEN*n*_TO_SEL 置 3）。
  - DT1：定时器递减计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_T1_SEL 置 2）或同步（字段 MCPWM_GEN*n*_T1_SEL 置 3）。
- 当这些定时事件同时发生时管理优先级。
- 基于定时事件产生置 1、置 0 和取反操作。

- 根据 PWM 生成器模块的配置来控制 PWM 占空比。
- 使用影子寄存器处理新的时间戳值，以防止 PWM 波形中的毛刺干扰。

PWM 生成器影子寄存器

硬件使用的时间戳寄存器 A 和 B 都配有影子寄存器，为 MCPWM_GEN*n*_A_REG 和 MCPWM_GEN*n*_B_REG。影子寄存器提供了一种与硬件同步更新寄存器的方法。

当 MCPWM_GLOBAL_UP_EN 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。MCPWM_GEN*n*_A_REG 和 MCPWM_GEN*n*_B_REG 的更新方式字段为 MCPWM_GEN*n*_CFG_UPMETHOD。软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。更多关于影子寄存器的描述，请参考章节 59.3.2.3。

定时事件

表 59.3-2 概括了所有定时信号和事件。

表 59.3-2. PWM 生成器中的所有定时事件

信号	事件描述	PWM 定时器操作
DTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递减计数
DTEZ	PWM 定时器的值等于 0	
DTEA	PWM 定时器的值等于寄存器 A	
DTEB	PWM 定时器的值等于寄存器 B	
DTO 事件	基于故障或同步事件	
DT1 事件	基于故障或同步事件	
UTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递增计数
UTEZ	定时器的值等于 0	
UTEA	PWM 定时器的值等于寄存器 A	
UTEB	PWM 定时器的值等于寄存器 B	
UTO 事件	基于故障或同步事件	
UT1 事件	基于故障或同步事件	
软件强制事件	软件触发的异步事件	-

软件强制事件用于在 PWM*x*A 和 PWM*x*B 的输出上施加非连续或连续的强制电平。此更改是异步完成的。软件强制由寄存器 MCPWM_GEN*n*_FORCE_REG 控制。

PWM 生成器模块中 TO/T1 的选择和配置独立于故障处理模块中的故障事件的配置。跳闸事件可以不被配置为在故障处理器模块中引起跳闸动作，但相同的事件可以由 PWM 生成器用于触发 TO/T1 以控制 PWM 波形。

需要注意的是，当 PWM 定时器处于递增递减循环计数模式时，它将在 TEP 事件后递减，在 TEZ 事件后递增。因此，在此模式下，将出现 DTEP 和 UTEZ，但不会出现 UTEP 和 DTEZ。

PWM 生成器可以同时处理多个事件。事件优先级由硬件决定，详见表 59.3-3 和表 59.3-4。优先级从 1（最高）到 7（最低）排列。需要注意的是，TEP 和 TEZ 事件的优先级取决于 PWM 定时器的计数模式。

如果 A 或 B 的值设置为大于周期，则 U/DTEA 和 U/DTEB 将永远不会发生。

表 59.3-3. PWM 定时器递增计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	UTEP
3	UTO
4	UT1
5	UTEB
6	UTEA
7（最低）	UTEZ

表 59.3-4. PWM 定时器递减计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	DTEZ
3	DTO
4	DT1
5	DTEB
6	DTEA
7（最低）	DTEP

说明：

1. UTEP 和 UTEZ 不同时发生。当 PWM 定时器处于递增计数模式，UTEP 将始终比 UTEZ 提前一个周期发生，如图 59.3-9 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，UTEP 不会发生。
2. DTEP 和 DTEZ 不同时发生。当 PWM 定时器处于递减计数模式时，DTEZ 始终比 DTEP 早一个周期发生，如图 59.3-10 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，DTEZ 不会发生。

PWM 信号生成

当某个定时事件发生时，PWM 生成器控制输出 PWM*x*A 和 PWM*x*B 的电平。定时事件通过 PWM 定时器计数模式（递增或递减）进一步限定。根据定时器计数方向，模块可以对 PWM 定时器递增或递减计数的阶段执行不同的操作。

可以在 PWM*x*A 和 PWM*x*B 输出上配置以下操作：

- 置为高电平：将 PWM*x*A 或 PWM*x*B 的输出配置为高电平。
- 置为低电平：通过将 PWM*x*A 或 PWM*x*B 的输出配置为低电平，来清除 PWM*x*A 或 PWM*x*B 的输出。
- 取反：将 PWM*x*A 或 PWM*x*B 的当前输出电平配置为相反的值。如果它当前被拉高，则拉低，或反之。
- 不进行操作：保持 PWM*x*A 和 PWM*x*B 输出电平不变。在这种状态下，仍然可以触发中断。

输出上的操作通过寄存器 MCPWM_GEN*n*_A_REG 和 MCPWM_GEN*n*_B_REG 配置。每一次输出的操作都独立配置。此外，根据事件在某个输出上，灵活地执行不同的操作。表 59.3-2 中列举的任何事件都可以作用于 PWM*x*A 或 PWM*x*B 输出上。关于生成器 0，1 或 2 的寄存器信息，请参考第 59.5 章。

常见配置的波形

在下面各种波形的配置中，period 指 PWM 操作器所选 PWM 定时器的周期寄存器的配置值（即 MCPWM_TIMER*n*_PERIOD (*x* = 0, 1, 2)），配置方法参考章节 59.3.2。A 和 B 指硬件使用的时间戳寄存器 A 和 B，配置方法参考章节 59.3.3.1。UTEA/B，UTEZ/P，DTEA/B，DTEZ/P，DTO/1，UTO/1 指 PWM 生成器的各种定时事件，其定义参考章节 59.3.3.1。向上的箭头、向下的箭头、T 标识分别代表定时事件发生时将 PWM 波形置为高电平、置为低电平、取反，配置方法参考章节 59.3.3.1。PWM*x*A 和 PWM*x*B 是 PWM 生成器 *x* 的输出信号。

图 59.3-13 为 PWM 定时器在递增递减循环计数时生成的对称 PWM 波形。该模式下的直流 0%-100% 调制可由以下公式获得：

$$Duty = (Period - A) \div Period$$

如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递增，则 PWM 输出被拉高。如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递减，则 PWM 输出被拉低。

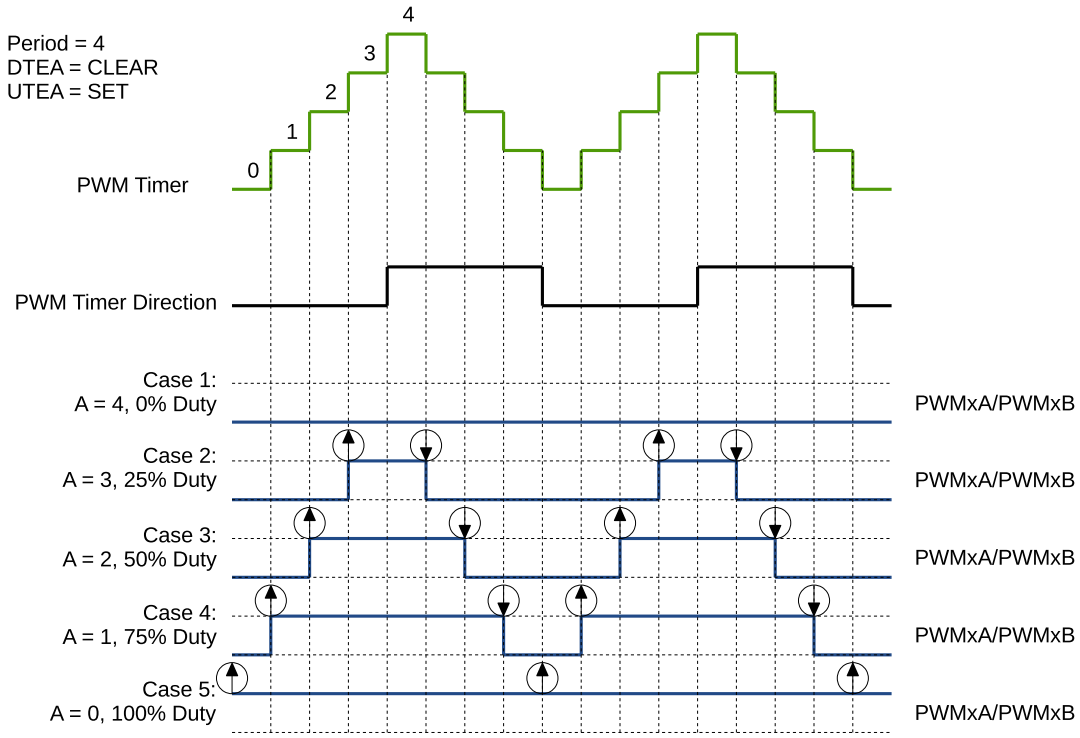


图 59.3-13. 递增递减模式下的对称波形

图 59.3-14 至图 59.3-17 的 PWM 波形描述了常见的 PWM 生成器配置。

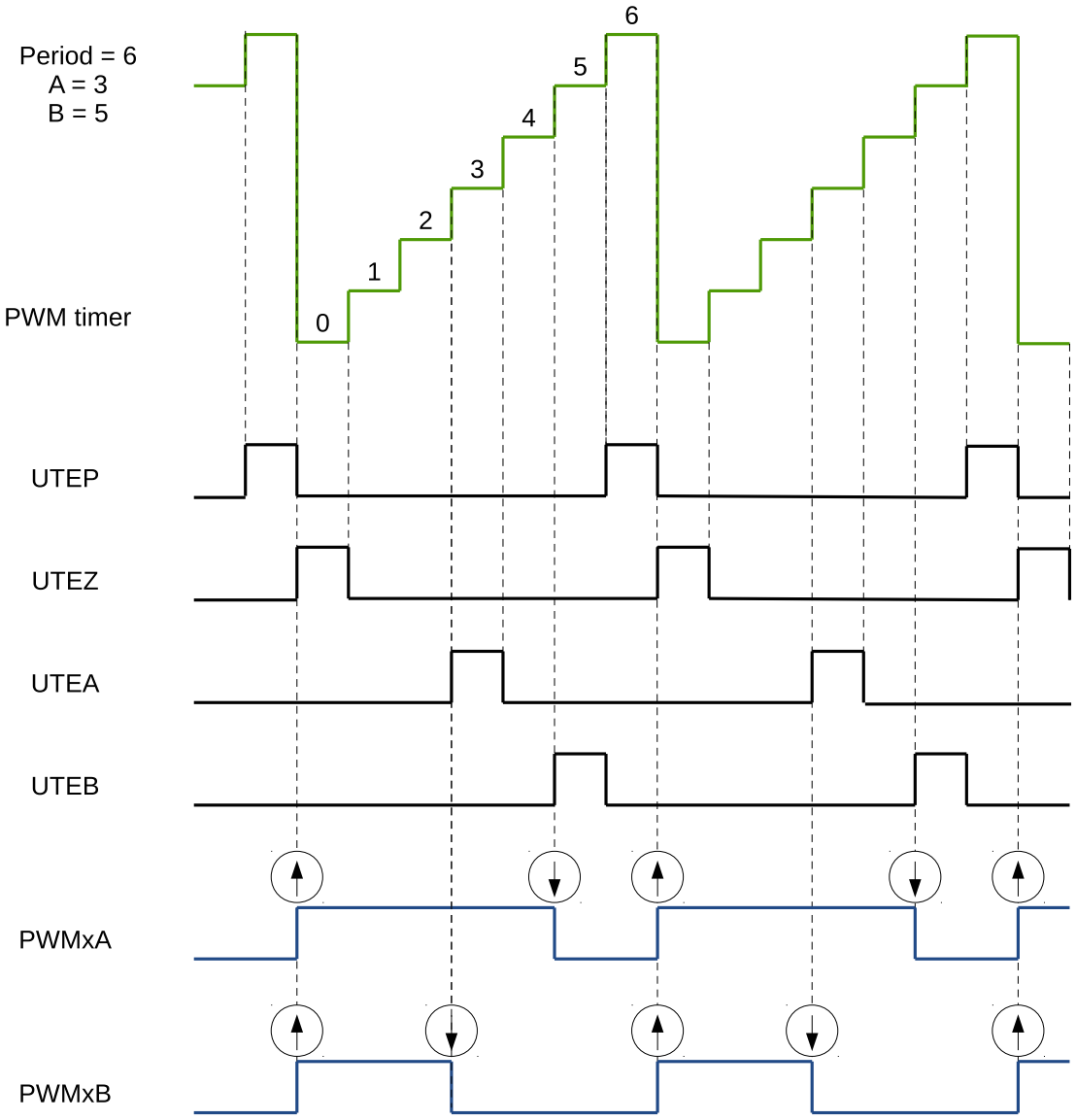


图 59.3-14. 递增计数模式，单边不对称波形，PWM*x*A 和 PWM*x*B 独立调制–高电平

PWM*x*A 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWM*x*B 的占空比调制由 A 设置，高电平有效，与 A 成正比。

$$Period = (MCPWM_TIMERn_PERIOD + 1) \times T_{PT_clk}$$

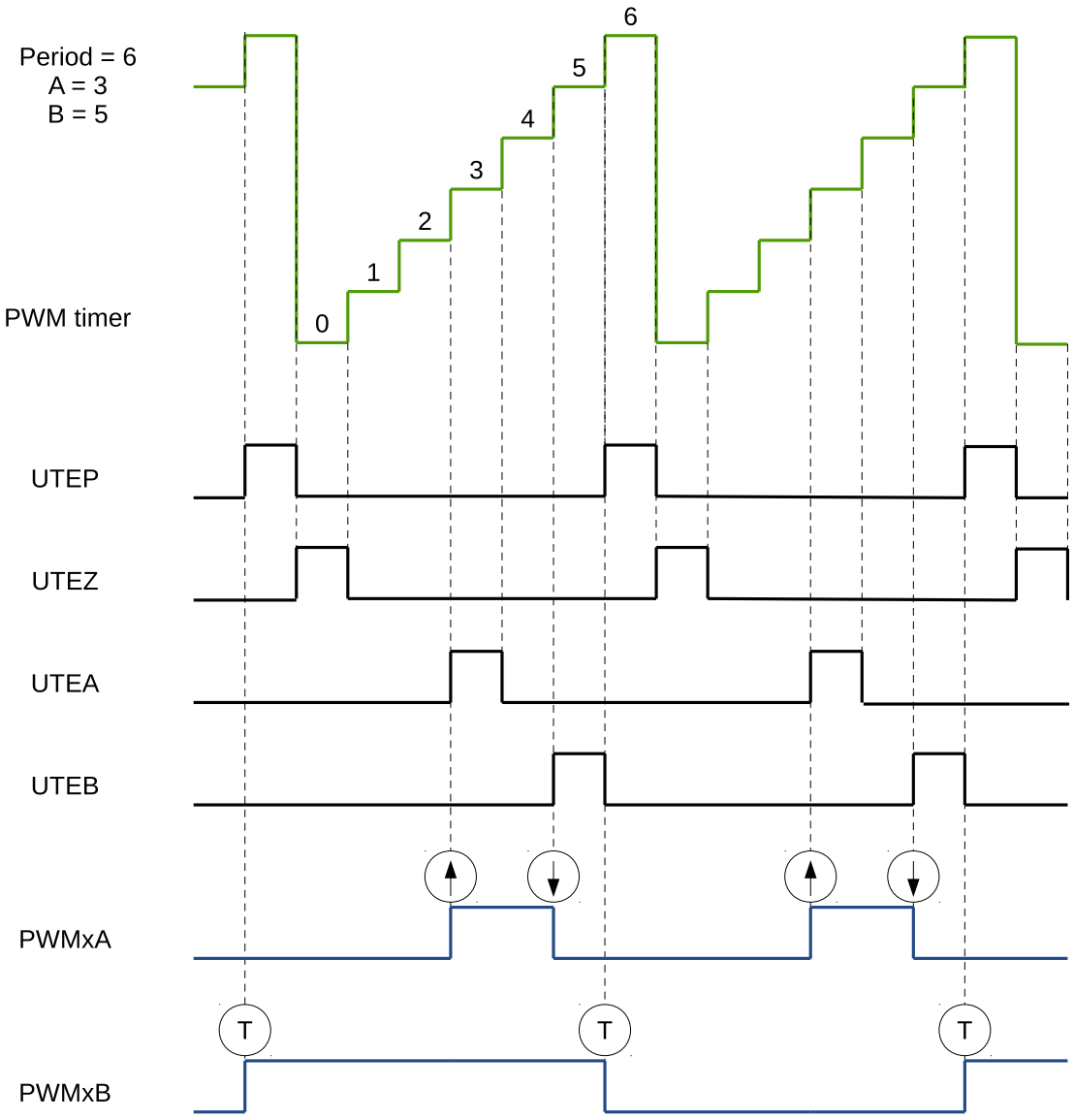


图 59.3-15. 递增计数模式，脉冲位置不对称波形，PWM*x*A 独立调制

脉冲可以在 PWM 波形内（零至周期值之间）的任何地方生成。

PWM*x*A 占空比与 (B – A) 成正比。

$$Period = (MCPWM_TIMERn_PERIOD + 1) \times T_{PT_clk}$$

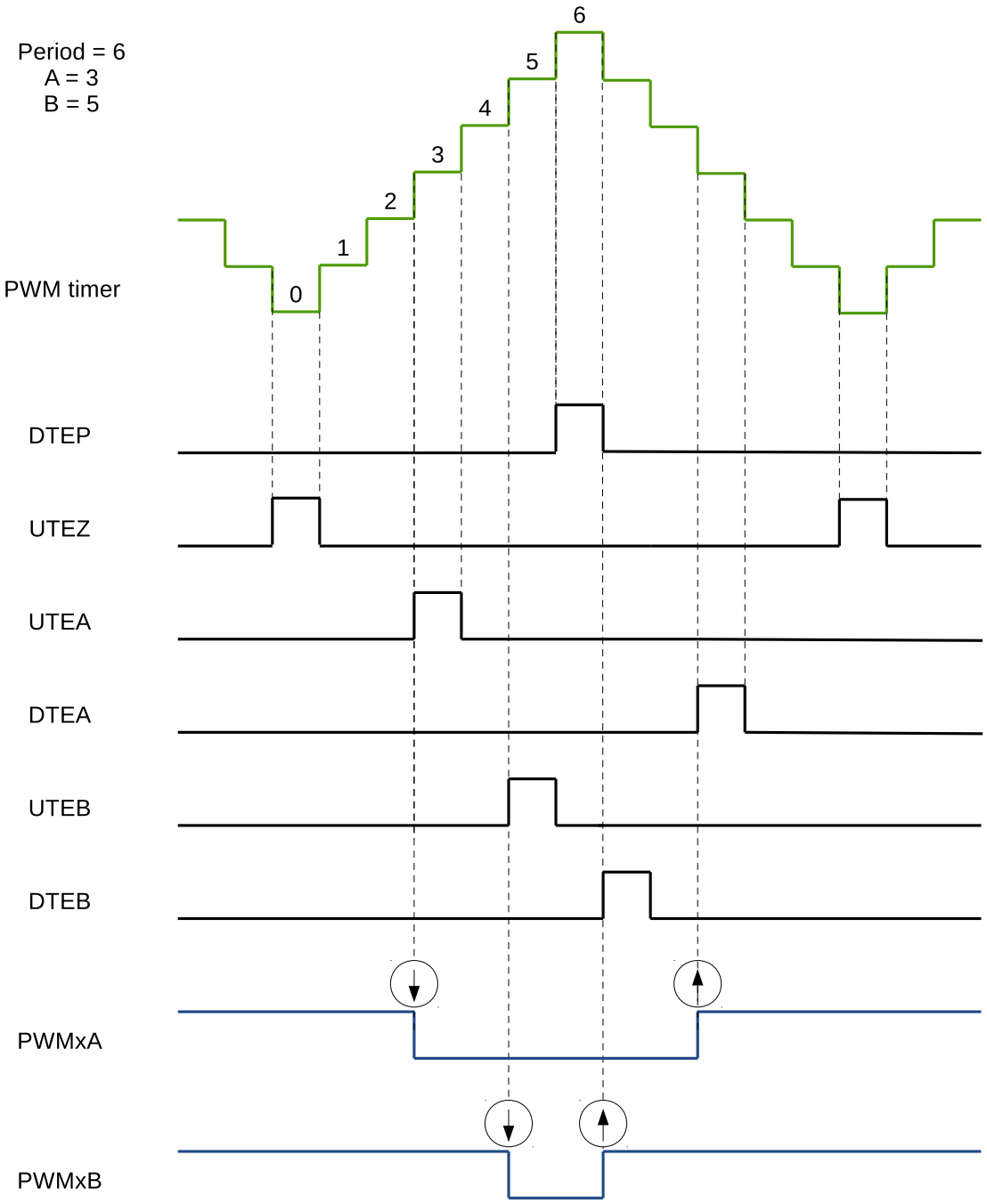


图 59.3-16. 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–高电平有效

PWM*x*A 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWM*x*B 的占空比调制由 B 设置，高电平有效，与 B 成正比。
输出 PWM*x*A 和 PWM*x*B 可驱动不同开关。

$$Period = (2 \times MCPWM_TIMERn_PERIOD) \times T_{PT_clk}$$

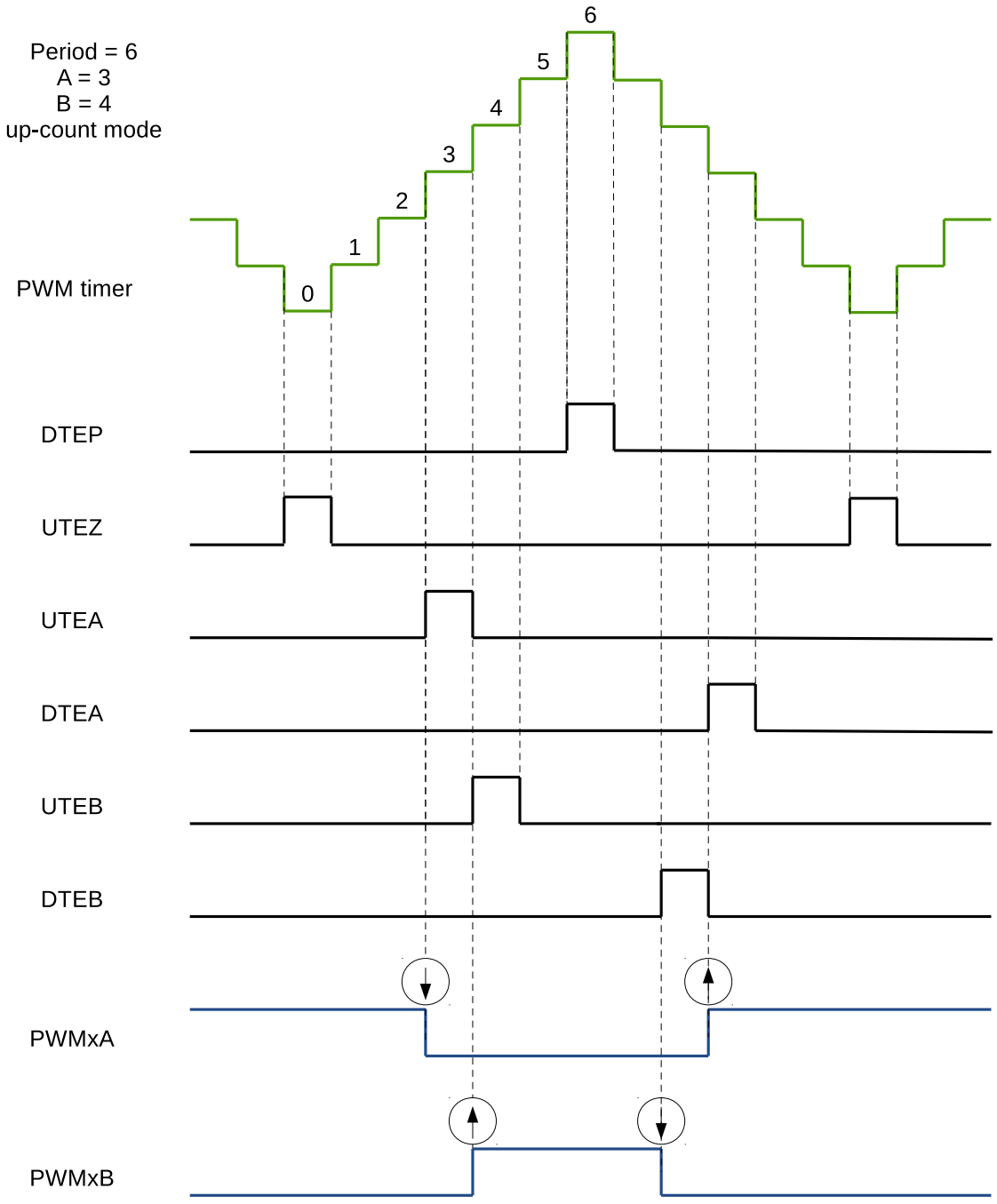


图 59.3-17. 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–互补

PWM*x*A 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWM*x*B 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWM*x*A/B 输出可驱动上/下（互补）开关。
死区 = B – A，边沿位置完全可由软件配置。必要时，可使用死区生成器模块设置其他边沿延迟方式。

$$Period = (2 \times MCPWM_TIMERn_PERIOD) \times T_{PT_clk}$$

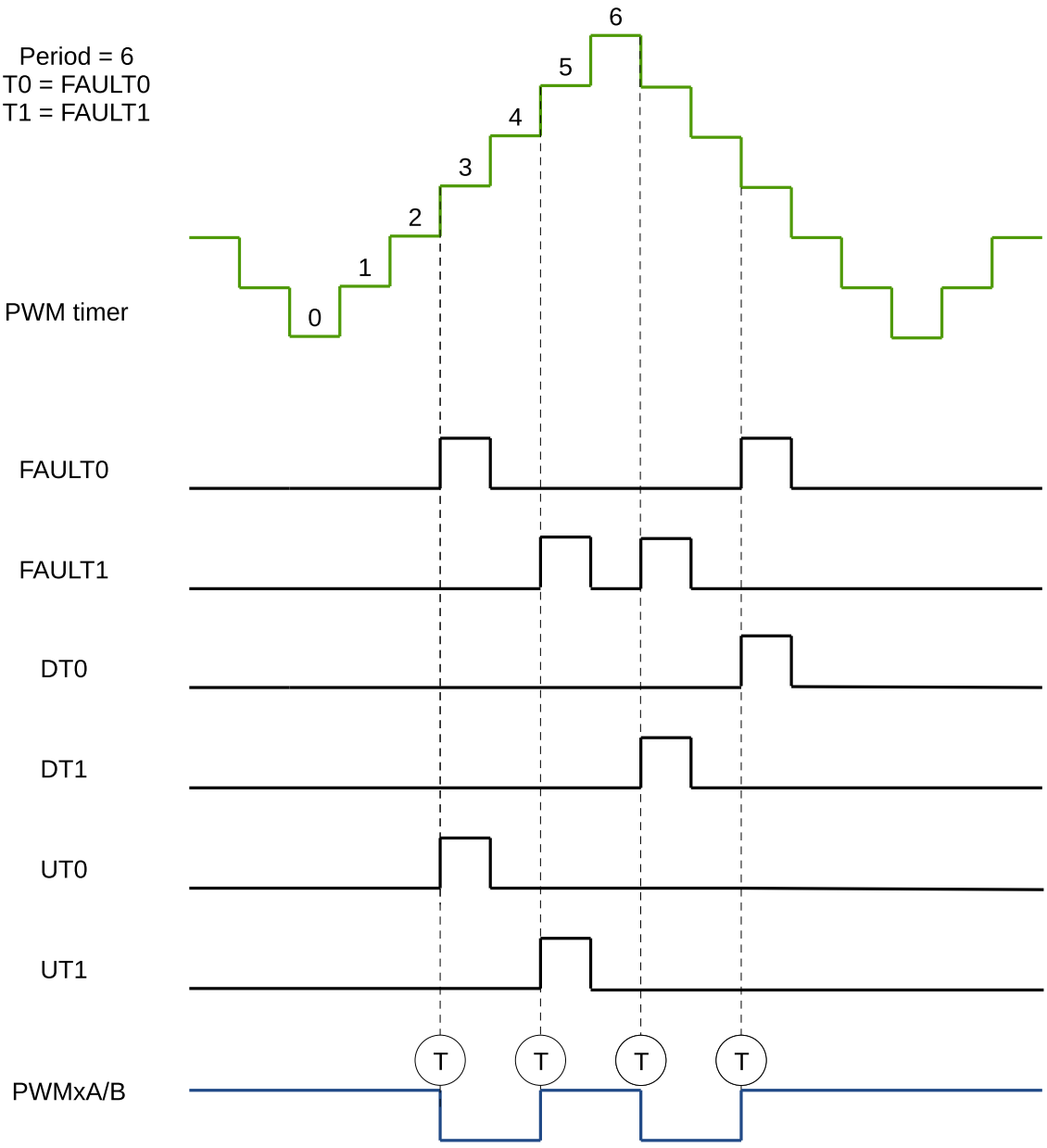


图 59.3-18. 递增递减循环计数模式，错误或同步事件，在 PWM*x*A 和 PWM*x*B 上相同调制

图 59.3-18 为当 UTO/1 和 DTO/1 事件产生时的波形。在本例中，TO 选择 FAULTO ，T1 选择 FAULT1。TO 和 T1 所选择的事件可以独立配置，这些事件可以是 FAULTO，FAULT1，FAULT2 或同步事件。具体配置方法见59.3.3.1。

软件强制事件

在 PWM 生成器内有 2 种软件强制事件：

- 非连续即时 (NCI) 软件强制事件：当由软件触发时，这类事件在 PWM 输出上立即生效。强制是不连续的，意味着下一个激活的定时事件能够改变 PWM 输出。
- 连续 (CNTU) 软件强制事件：这类事件是连续的。强制 PWM 将持续输出，直到通过软件释放。事件触发器可配置。这类事件可配置为定时或即时发生。

对于 NCI 软件强制事件，向寄存器 MCPWM_GEN*n*_A_NCIFORCE 或 MCPWM_GEN*n*_B_NCIFORCE 写 1 再写

0，触发 PWM*x*A 或 PWM*x*B 的 NCI 软件强制事件。配置寄存器 MCPWM_GEN*n*_A_NCIFORCE_MODE 或 MCPWM_GEN*n*_B_NCIFORCE_MODE 指定当 NCI 软件强制事件发生时，对 PWM*x*A 或 PWM*x*B 波形的操作，配置为 0 或 3 表示无操作，配置为 1 表示置为低电平，配置为 2 表示置为高电平。

对于 CNTU 软件强制事件，配置寄存器 MCPWM_GEN*n*_CNTUFORCE_UPMETHOD，指定 CNTU 软件强制事件的触发方式，所有 bit 配置为 0，表示立即触发 CNTU 软件强制事件，bit 0 配置为 1，表示 TEZ 事件触发 CNTU 软件强制事件，bit 1 配置为 1，表示 TEP 事件触发 CNTU 软件强制事件，bit 2 配置为 1，表示 TEA 事件触发 CNTU 软件强制事件，bit 3 配置为 1，表示 TEB 事件触发 CNTU 软件强制事件，bit 4 配置为 1，表示 PWM 生成器所选的 PWM 定时器同步时触发 CNTU 软件强制事件，bit 5 配置为 1，表示不触发 CNTU 软件强制事件。配置寄存器 MCPWM_GEN*n*_A_CNTUFORCE_MODE 或 MCPWM_GEN*n*_B_CNTUFORCE_MODE 指定当 CNTU 软件强制事件发生时，对 PWM*x*A 或 PWM*x*B 波形的操作，配置为 0 或 3 表示无操作，配置为 1 表示置为低电平，配置为 2 表示置为高电平。

图 59.3-19 为 NCI 软件强制事件的一种波形。NCI 用于单独强制 PWM*x*A 输出为低电平，PWM*x*B 不受强制。

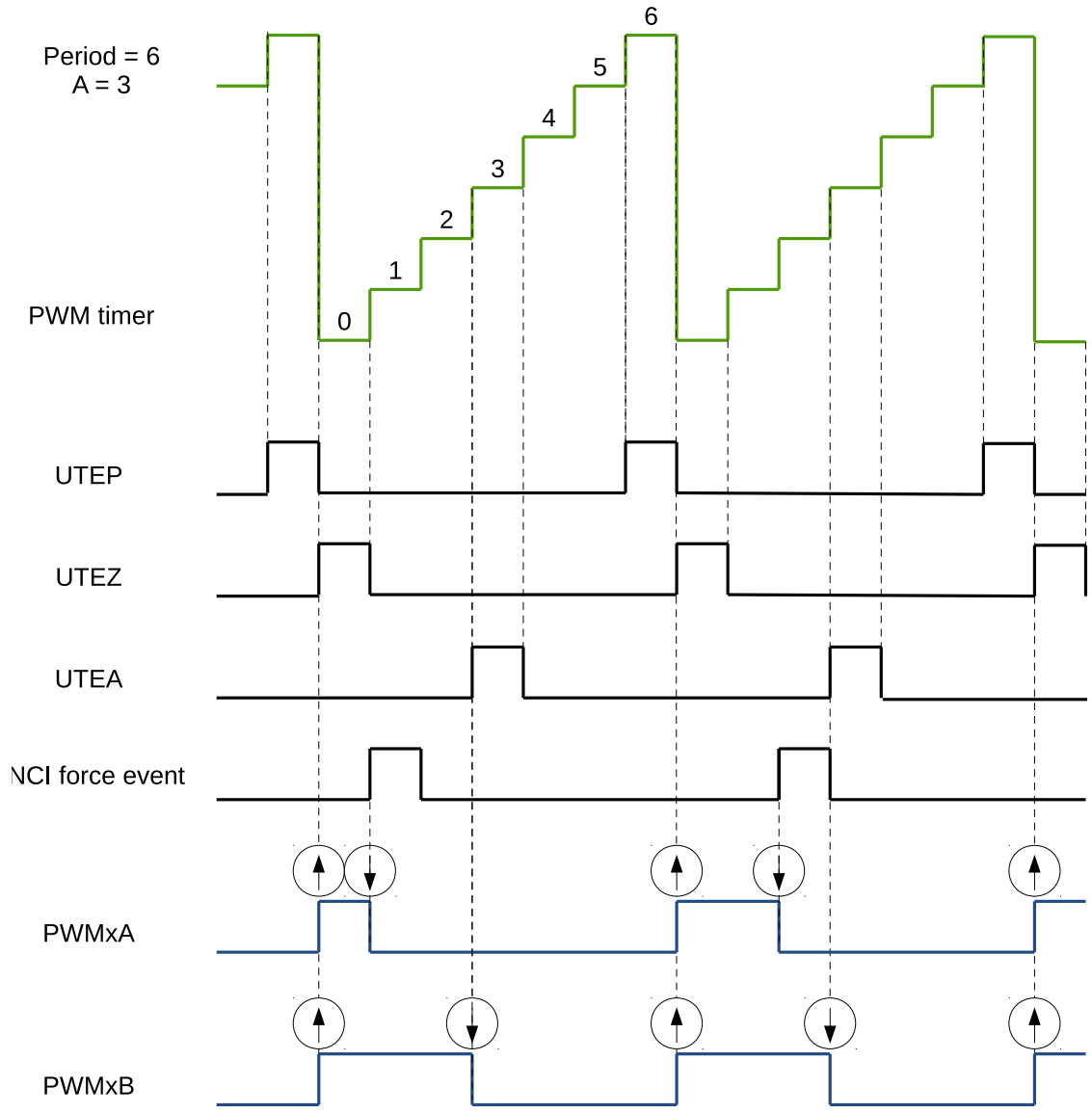


图 59.3-19. NCI 在 PWM*x*A 输出上软件强制事件示例

图 59.3-20 为 CNTU 软件强制事件的波形。TEZ 事件被选为 CNTU 软件强制事件的触发器。CNTU 用于单独强制 PWM*x*B 输出为低电平，但 PWM*x*A 不受强制。

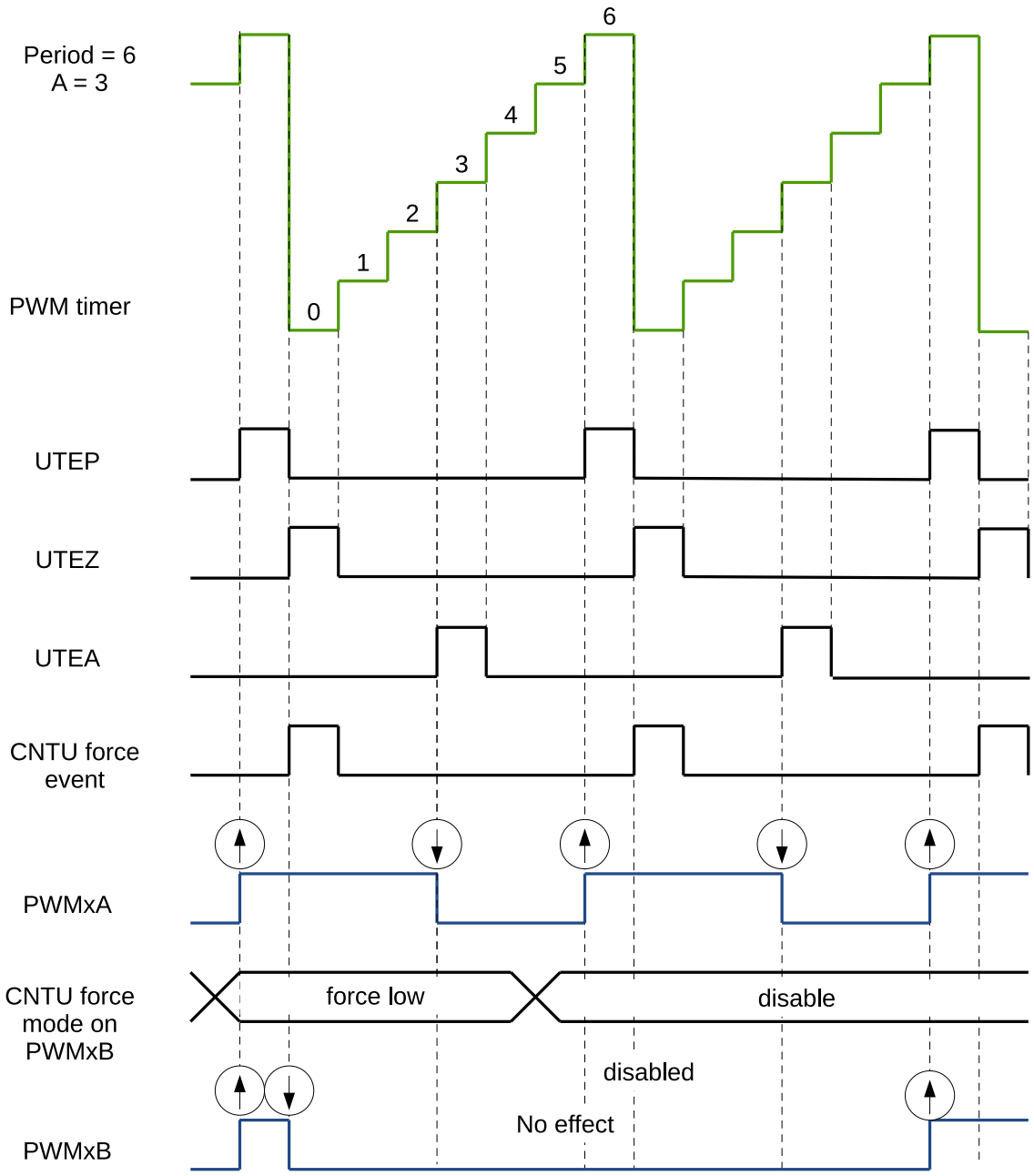


图 59.3-20. CNTU 在 PWM*x*B 输出上软件强制事件示例

59.3.3.2 死区生成器模块

死区生成器模块作用

章节 59.3.3.1 介绍了在 PWM*x*A 和 PWM*x*B 输出上生成信号的几种方式/事件，包括信号边沿的特定位置。通过改变信号之间的边沿位置以及设置信号的占空比，可获得所需的死区。此外，也可以使用专门的死区生成器模块控制死区。

死区生成器模块的主要功能如下：

- 可以将输入的 PWM*x*A 或 PWM*x*B 信号作为公共源来生成具有死区的 PWM*x*A 和 PWM*x*B 信号对输出
- 通过在信号边沿增加延迟来生成死区：
  - 上升沿延迟 (RED)
  - 下降沿延迟 (FED)
- 可以生成多种 PWM 波形，经典的死区配置为：
  - 高电平有效互补 (AHC)
  - 低电平有效互补 (ALC)
  - 高电平有效 (AH)
  - 低电平有效 (AL)
- 如果死区已经在生成器中配置，死区发生器也可以不进行配置。

死区模块生成器影子寄存器

延迟寄存器 RED 和 FED 的影子寄存器为 MCPWM_DT*n*_RED_CFG_REG 和 MCPWM_DT*n*_FED_CFG_REG。
MCPWM_GLOBAL_UP_EN 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。
MCPWM_DT*n*_RED_CFG_REG 的更新方式寄存器为 MCPWM_DB*n*_RED_UPMETHOD；
MCPWM_DT*n*_FED_CFG_REG 的更新方式寄存器为 MCPWM_DB*n*_FED_UPMETHOD。软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。寄存器描述详见 59.3.2.3。

死区生成器模块的操作要点

图 59.3-21 描述了创建死区模块的开关拓扑。

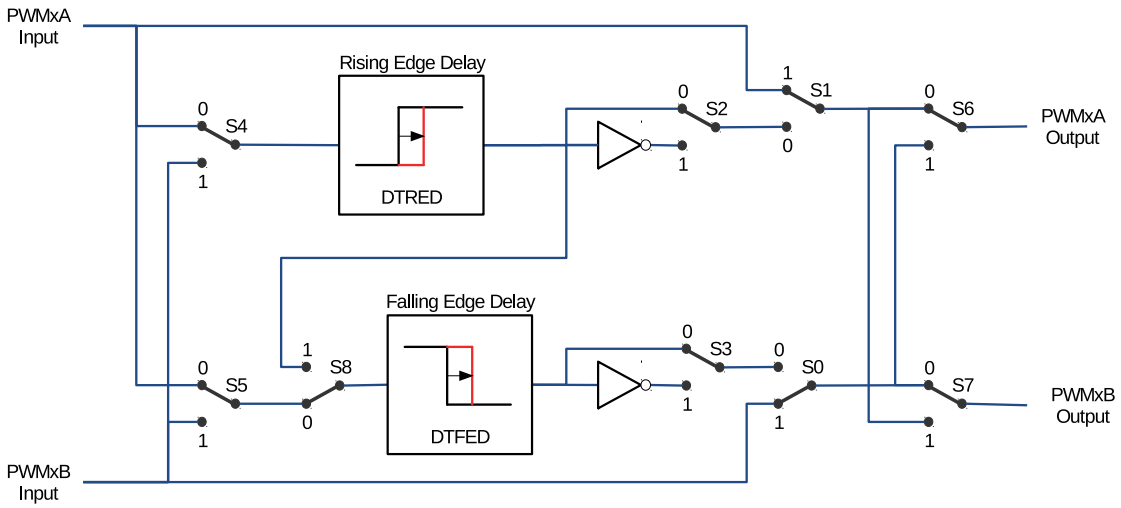


图 59.3-21. 死区模块的开关拓扑

上图中的 S0 - S8 开关由 MCPWM_DT*n*_CFG_REG 寄存器中的字段控制，详细信息见表 59.3-5。

表 59.3-5. 控制死区时间生成器开关的字段

开关	字段
S0	MCPWM_DB*n*_B_OUTBYPASS
S1	MCPWM_DB*n*_A_OUTBYPASS
S2	MCPWM_DB*n*_RED_OUTINVERT
S3	MCPWM_DB*n*_FED_OUTINVERT
S4	MCPWM_DB*n*_RED_INSEL
S5	MCPWM_DB*n*_FED_INSEL
S6	MCPWM_DB*n*_A_OUTSWAP
S7	MCPWM_DB*n*_B_OUTSWAP
S8	MCPWM_DB*n*_DEB_MODE

所有开关组合皆受支持，但不是所有的开关模式都是典型的使用模式。表 59.3-6 列举了一些典型的死区配置。在这些配置中，S4 和 S5 的开关位置将 PWM*x*A 设置为下降沿和上升沿延迟的公共源。表 59.3-6 中的模式可分为以下几类：

- **模式 1：绕过下降沿 (FED) 和上升沿 (RED) 的延迟**
  该模式下，死区模块被关闭。PWM*x*A 和 PWM*x*B 信号的波形无变化。
- **模式 2-5：经典死区极性设置**
  这些模式为典型极性配置，涵盖工业电源栅极驱动器中的高 / 低电平有效模式。图 59.3-22 至 59.3-25 为典型波形。
- **模式 6 和 7：绕过下降沿 (FED) 或上升沿 (RED) 的延迟**
  这两种模式下，绕过上升沿延迟 (RED) 或下降沿延迟 (FED)。因此，无需使用对应延迟。

表 59.3-6. 死区生成器的典型操作模式

模式	描述	S0	S1	S2	S3
1	PWM*x*A 和 PWM*x*B 波形无变化	1	1	X	X
2	高电平有效互补 (AHC)，参见图 59.3-22	0	0	0	1
3	低电平有效互补 (ALC)，参见图 59.3-23	0	0	1	0
4	高电平有效 (AH)，参见图 59.3-24	0	0	0	0
5	低电平有效 (AL)，参见图 59.3-25	0	0	1	1
6	PWM*x*A 输出 = PWM*x*A 输入（无延迟） PWM*x*B 输出 = PWM*x*A 输入，下降沿延迟	0	1	0 或 1	0 或 1
7	PWM*x*A 输出 = PWM*x*A 输入，上升沿延迟 PWM*x*B 输出 = PWM*x*B 输入（无延迟）	1	0	0 或 1	0 或 1

说明：

以上所有模式中，S4 - S8 的开关位置都置 0。

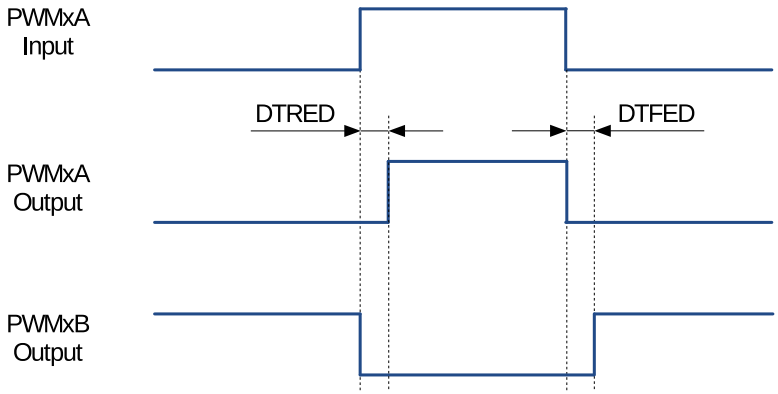


图 59.3-22. 高电平有效互补 (AHC) 死区波形

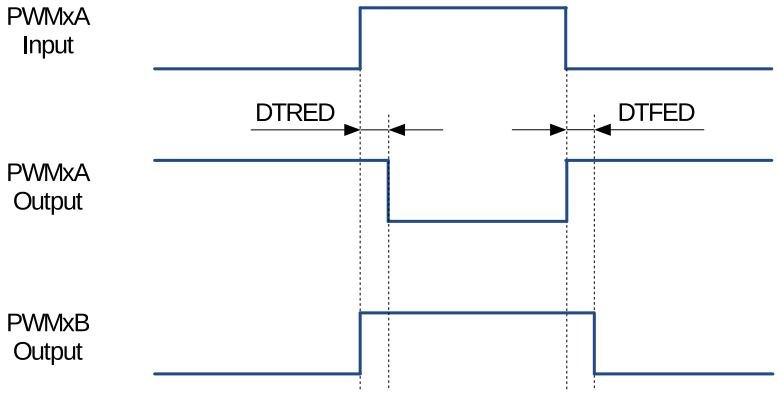


图 59.3-23. 低电平有效互补 (ALC) 死区波形

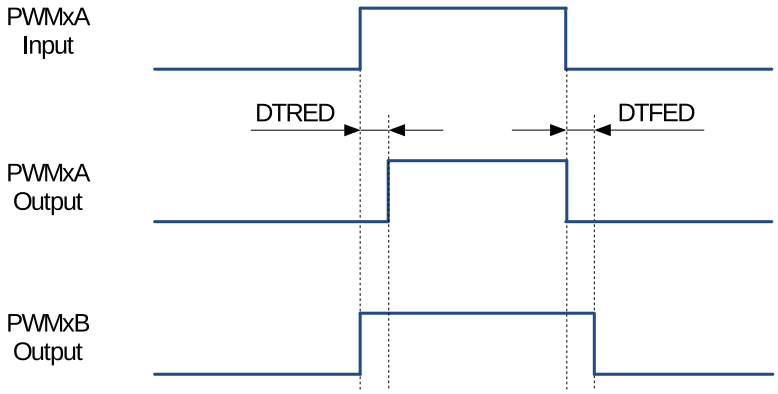


图 59.3-24. 高电平有效 (AH) 死区波形

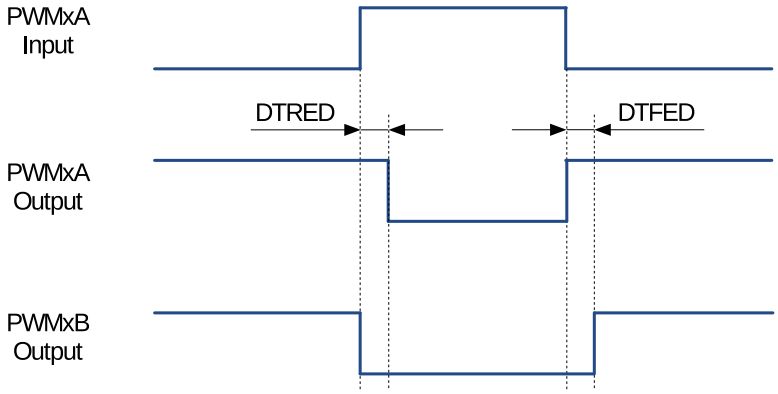


图 59.3-25. 低电平有效 (AL) 死区波形

上升沿延迟 (RED) 和下降沿延迟 (FED) 可分别设置。延迟的值通过 16 位寄存器 MCPWM_DT*x*_*RED* 和 MCPWM_DT*x*_*FED* 配置。寄存器值表示一个信号边沿可以延迟的 DT_clk 时钟周期值。DT_clk 可通过寄存器 MCPWM_DB*n*_CLK_SEL 从 PWM_clk 或 PT_clk 中选择。

通过以下公式计算下降沿延迟 (FED) 和上升沿延迟 (RED) 的值:

$$FED = MCPWM_DTn_FED \times T_{DT_clk}$$

$$RED = MCPWM_DTn_RED \times T_{DT_clk}$$

59.3.3.3 PWM 载波模块

将 PWM 输出耦合到电机驱动器可能需要使用变压器隔离。变压器只提供交流信号，而 PWM 信号的占空比可能在 0% 到 100% 之间变化。PWM 载波模块可以通过使用高频载波对其进行调制，将该信号传递给变压器。

功能概述

此模块的以下关键功能可配置:

- 载波频率
- 第一个脉冲的脉宽
- 第二个以及之后的脉冲的占空比
- 开启/关闭载波

操作要点

PWM 载波时钟 (PC_clk) 来自于 PWM_CLK。通过寄存器 MCPWM_CARRIER*n*_CFG_REG 的 MCPWM_CHOPPER*n*_PRESCALE 和 MCPWM_CHOPPER*n*_DUTY 位配置频率和占空比。一次性脉冲的功能在于提供高能量脉冲以接通电源开关。随后的脉冲用于保持上电的状态。一次性脉冲宽度可通过 MCPWM_CHOPPER*n*_OSHTWTH 位进行配置。通过 MCPWM_CHOPPER*n*_EN 位来使能/禁用载波模块。

载波示例

图 59.3-26 描述了载波叠加在原始 PWM 脉冲上的示例波形。图中未显示第一个脉冲和占空比控制，相关详细信息将在后两节中介绍。

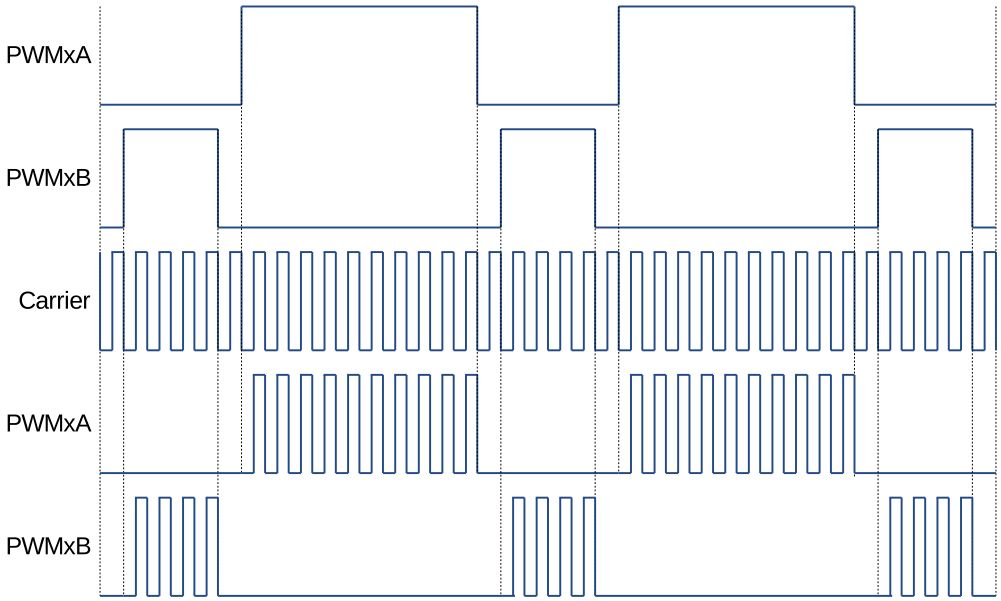


图 59.3-26. PWM 载波操作的波形示例

第一个脉冲

第一个脉冲的宽度可配置，其值有 16 种可能，可通过下公式计算：

$$T_{1stpulse} = T_{PWM_clk} \times 8 \times (MCPWM_CARRIERn_PRESCALE + 1) \times (MCPWM_CARRIERn_OSHTWTH + 1)$$

其中：

- T_{PMW_clk} 为 PWM 时钟周期 (PWM_clk)
- $(MCPWM_CARRIERn_OSHTWTH + 1)$ 为一次性脉冲宽度值（取值范围：1-16）
- $(MCPWM_CARRIERn_PRESCALE + 1)$ PWM 载波时钟 (PC_clk) 预分频值

图 59.3-27 展示了第一个脉冲和之后持续的脉冲。

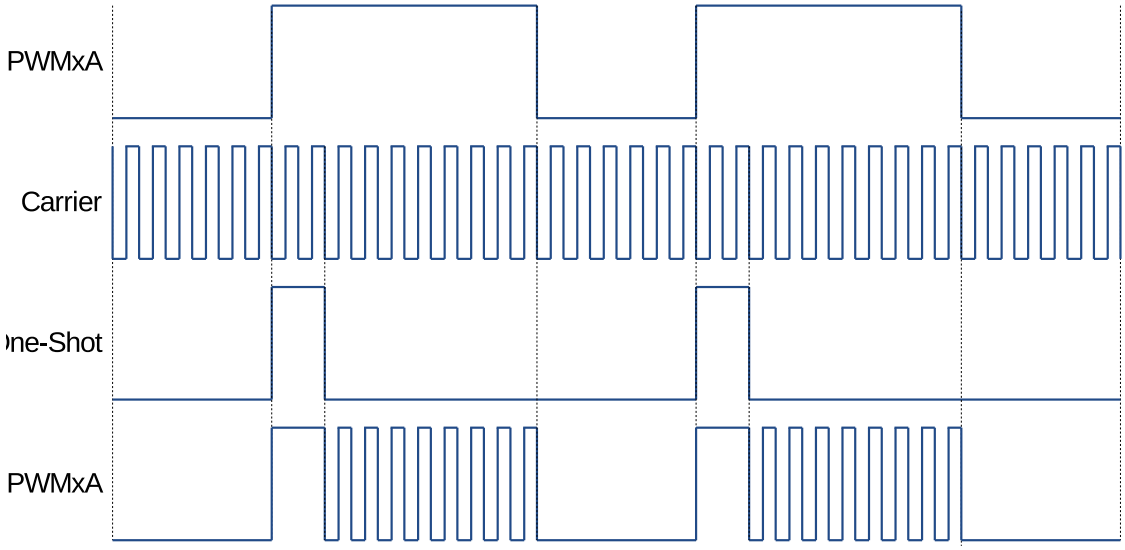


图 59.3-27. 载波模块的第一个脉冲和之后持续的脉冲示例

占空比控制

在发出第一个一次性脉冲之后，根据载波频率调制剩余的 PWM 信号。用户可配置该信号的占空比。在一定情况下，调整占空比可使信号通过隔离变压器后仍然可以开启或关闭电动机驱动器，改变电机旋转速度和方向。

占空比通过 MCPWM_CHOPPER*n*_DUTY 或寄存器 MCPWM_CARRIER*n*_CFG_REG 的 [7:5] 位设置，其值有 7 种可能性。

占空比的值可通过以下方式计算：

$$Duty = MCPWM_CARRIERn_DUTY \div 8$$

图 59.3-28 为所有 7 种占空比设置。

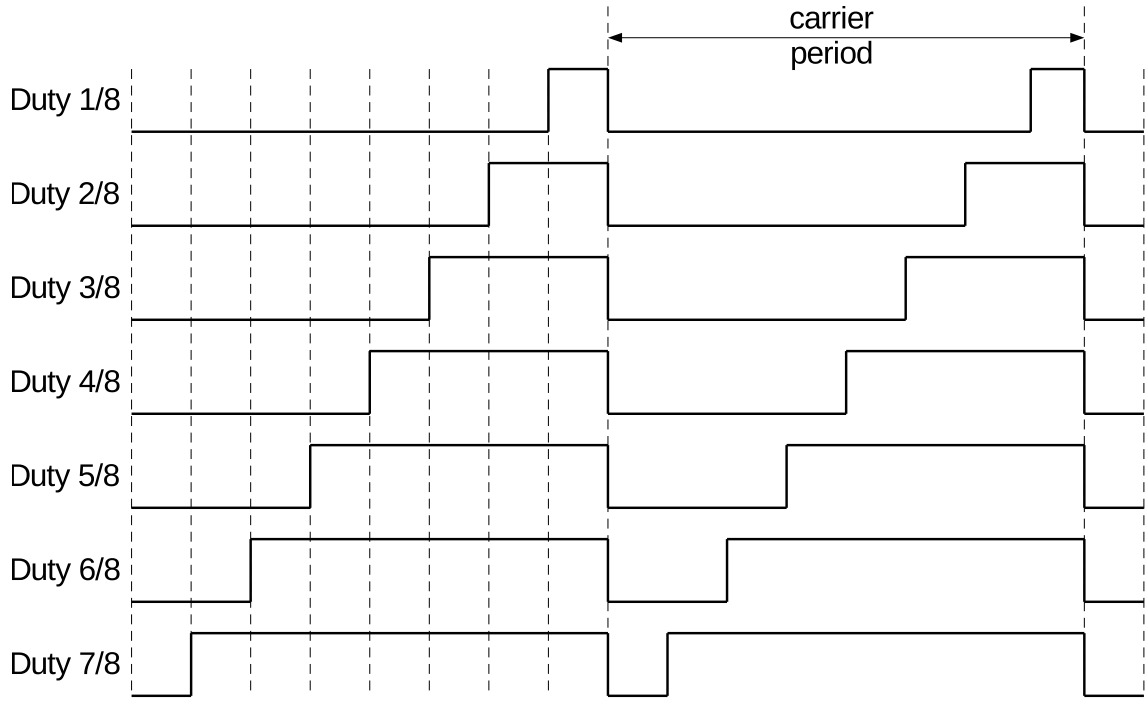


图 59.3-28. PWM 载波模块中持续脉冲的 7 种占空比设置

59.3.3.4 故障检测模块

每个 MCPWM 外设都连接来自 GPIO 矩阵的 3 个故障信号（FAULT0、FAULT1 和 FAULT2）。这些信号用于指示外部故障状况，并且可由故障检测模块预处理后生成故障事件。故障事件通过执行用户代码，针对特定故障调整 MCPWM 输出。

故障检测模块功能

故障检测模块的主要功能为：

- 在检测到故障时强制 PWM*x*A 和 PWM*x*B 输出信号进入以下状态之一：
  - 高
  - 低
  - 取反
  - 无
- 在检测到过电流/短路时执行一次性跳闸 (OST)。
- 逐周期跳闸 (CBC) 以提供限流操作。
- 每个故障信号单独分配一次性或逐周期操作。
- 每个故障输入都生成中断。
- 支持软件强制跳闸。
- 根据需要开启或关闭模块功能。

操作与配置要点

本节介绍故障处理模块的操作要点和配置选项。

来自管脚的故障信号在 GPIO 矩阵中采样和同步。为了保证故障脉冲成功采样，每个脉冲持续时间必须至少为 2 个 APB 时钟周期。故障检测模块使用 PWM_clk 对故障信号进行采样，因此，来自 GPIO 矩阵的故障脉冲持续时间必须至少为 1 个 PWM_clk 周期。换句话说，无论 APB 时钟周期和 PWM_clk 周期的大小关系如何，管脚上的故障信号脉冲的宽度必须至少等于两个 APB 时钟周期与一个 PWM_clk 周期的和。

故障处理器模块可以使用故障信号 FAULT0 至 FAULT2 中的高电平或低电平来生成故障事件 fault_event0 至 fault_event2。每个故障事件可以单独配置为进行 CBC 操作、OST 操作或无操作。

- **逐周期 (CBC) 操作：**
  触发 CBC 操作时，PWM*x*A 和 PWM*x*B 的状态立即根据字段 MCPWM_TZ*n*_A_CBC_U/D 和 MCPWM_TZ*n*_B_CBC_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 CBC 操作中断。通过状态字段 MCPWM_TZ*n*_CBC_ON 开启或关闭 CBC 操作。在没有故障事件时，将在指定时间点，即发生 D/UTEP 或 D/UTEZ 事件时，清除 PWM*x*A/B 上的 CBC 操作。由字段 MCPWM_TZ*n*_CBCPULSE 决定 PWM*x*A 和 PWM*x*B 恢复正常的事件。因此，在此模式下，每个 PWM 循环后，CBC 操作都将清除或刷新。
- **一次性 (OST) 操作：**
  触发 OST 操作时，PWM*x*A 和 PWM*x*B 的状态立即根据字段 MCPWM_TZ*n*_A_OST_U/D 和 MCPWM_TZ*n*_B_OST_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 OST 操作中断。通过状态字段 MCPWM_TZ*n*_OST_ON 开启或关闭 OST 操作。在没有故障事件时，PWM*x*A/B 上的 OST 操作无法自动清除。须通过将 MCPWM_TZ*n*_CLR_OST 位置 1 来清除一次性操作。

59.3.4 捕获模块

59.3.4.1 介绍

捕获模块包含 3 个完整的捕获通道。通道输入信号 CAP0、CAP1 和 CAP2 来自于 GPIO 矩阵。由于 GPIO 矩阵的灵活性，CAP0、CAP1 和 CAP2 可以通过任一管脚输入配置。多个捕获通道可以来自同一管脚输入，每个通道的预分频可以分别设置。同时，每个捕获通道也可以来自不同的管脚输入。因此，可以通过后台硬件用多种方式处理捕获信号，而不直接由 CPU 处理。

捕捉模块有以下独立资源：

- 一个 32 位定时器（计数器），可与 PWM 定时器、另一个模块或软件同步。
- 三个捕获通道，每个通道配有一个 32 位时间戳和一个捕获预分频器。
- 任何捕获通道的边沿极性（上升/下降沿）可独立选择。
- 输入捕获信号预分频（分频取值范围：1 ~ 256）。
- 三个捕获事件都有中断功能。

59.3.4.2 捕获定时器

捕获定时器是一个 32 位计数器，使能时不断递增计数。将 MCPWM_CAP_TIMER_EN 置 1 使能捕获寄存器。其操作时钟源为 MCPWM 模块的主时钟。配置 MCPWM_CAP_SYNCI_EN 后，发生同步事件时，存储在

MCPWM_CAP_TIMER_PHASE_REG 寄存器中的相位将被加载至计数器中。同步事件可来自 PWM 定时器同步输出或 PWM 模块同步输入，通过 MCPWM_CAP_SYNCI_SEL 进行配置。也可将 MCPWM_CAP_SYNC_SW 置 1 生成同步事件。捕获定时器为所有三个捕获通道提供定时参考。

59.3.4.3 捕获通道

必要时，到达捕获通道的捕获信号可先被反相，然后预分频。每个捕获通道都有一个预分频寄存器 MCPWM_CAP*n*_PRESCALE。最后，预处理后的捕获信号的指定边沿将触发捕获事件。将 MCPWM_CAP*n*_EN 置 1 使能捕获通道。捕获事件将在 MCPWM_CAP*n*_MODE 配置的时间发生。在捕获事件发生时，捕获定时器的值存储在时间戳寄存器 MCPWM_CAP_CH*n*_REG 中。捕获事件中的不同捕获通道可生成不同的中断。触发捕获事件的边沿储存在寄存器 MCPWM_CAP*n*_EDGE 中。捕获事件可通过将 MCPWM_CAP*n*_SW 置 1 由软件强制发生。

59.3.5 ETM 模块

59.3.5.1 介绍

在 ESP32-P4 中，MCPWM 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 MCPWM 的 ETM 任务，或者通过 MCPWM 的 ETM 事件触发任意外设的 ETM 任务。MCPWM 的捕获模块、故障检测模块、三个定时器以及三个操作器都可以独立产生事件或响应任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 MCPWM 相关的 ETM 任务和 ETM 事件。

59.3.5.2 MCPWM 可产生的 ETM 事件

将使能字段置 1 后，满足对应生成条件时，将产生对应 ETM 事件，详见表 59.3-7：

表 59.3-7. MCPWM 可产生的 ETM 事件

使能字段	生成条件	产生事件
MCPWM_EVT_CAP*n*_EN	发生 CAP*n* 捕获事件	MCPWM*m*_EVT_CAP*n*
MCPWM_EVT_TZ*n*_OST_EN	PWM 操作器 *n* 执行一次性跳闸 (OST) 操作	MCPWM*m*_EVT_TZ*n*_OST
MCPWM_EVT_TZ*n*_CBC_EN	PWM 操作器 *n* 执行逐周期跳闸 (CBC) 操作	MCPWM*m*_EVT_TZ*n*_CBC
MCPWM_EVT_F*n*_CLR_EN	清除故障事件 fault_event*n*	MCPWM*m*_EVT_F*n*_CLR
MCPWM_EVT_F*n*_EN	产生故障事件 fault_event*n*	MCPWM*m*_EVT_F*n*
MCPWM_EVT_OP*n*_TEB_EN	PWM 操作器 *n* 选择的定时器计数值等于时间戳 B 的值	MCPWM*m*_EVT_OP*n*_TEB
MCPWM_EVT_OP*n*_TEA_EN	PWM 操作器 *n* 选择的定时器计数值等于时间戳 A 的值[1]	MCPWM*m*_EVT_OP*n*_TEA
MCPWM_EVT_OP*n*_TEE1_EN	PWM 操作器 *n* 选择的定时器计数值等于寄存器 MCPWM_OP*n*_TSTMP_E1_REG 的值	MCPWM*m*_EVT_OP*n*_TEE1
MCPWM_EVT_OP*n*_TEE2_EN	PWM 操作器 *n* 选择的定时器计数值等于寄存器 MCPWM_OP*n*_TSTMP_E2_REG 的值	MCPWM*m*_EVT_OP*n*_TEE2
MCPWM_EVT_TIMER*n*_TEP_EN	定时器 *n* 的计数值等于周期值 MCPWM_TIMER*n*_PERIOD	MCPWM*m*_EVT_TIMER*n*_TEP
MCPWM_EVT_TIMER*n*_TEZ_EN	定时器 *n* 的计数值等于 0	MCPWM*m*_EVT_TIMER*n*_TEZ
MCPWM_EVT_TIMER*n*_STOP_EN	定时器 *n* 停止计数	MCPWM*m*_EVT_TIMER*n*_STOP

[1] 有关时间戳 A 和时间戳 B 的描述，详见章节 59.3.3.1。

59.3.5.3 MCPWM 可接收的 ETM 任务

将使能字段置 1 后，接收对应有效任务后，将产生对应的响应操作，详见表 59.3-8：

表 59.3-8. MCPWM 可接收的 ETM 任务

使能字段	接收的有效任务	响应操作
MCPWM_TASK_CAP*n*_EN	MCPWM*m*_TASK_CAP*n*	CAP*n* 通道进行捕获操作
MCPWM_TASK_CLR*n*_OST_EN	MCPWM*m*_TASK_CLR*n*_OST	清除 PWM 操作器 *n* 的一次性跳闸 (OST) 操作
MCPWM_TASK_TZ*n*_OST_EN	MCPWM*m*_TASK_TZ*n*_OST	PWM 操作器 *n* 执行一次性跳闸 (OST) 操作
MCPWM_TASK_TIMER*n*_PERIOD_UP_EN	MCPWM*m*_TASK_TIMER*n*_PERIOD_UP	定时器 *n* 的周期更新为周期寄存器 MCPWM_TIMER*n*_PERIOD 配置的值
MCPWM_TASK_TIMER*n*_SYNC_EN	MCPWM*m*_TASK_TIMER*n*_SYN	定时器 *n* 执行一次同步操作
MCPWM_TASK_GEN_STOP_EN	MCPWM*m*_TASK_GEN_STOP	所有定时器停止计数，所有的 PWM 操作器输出的 PWM 信号保持不变
MCPWM_TASK_CMPR*n*_B_UP_EN	MCPWM*m*_TASK_CMPR*n*_B_UP	将 PWM 操作器 *n* 的时间戳 B 更新为时间戳 B 影子寄存器 MCPWM_GEN*n*_B 的值
MCPWM_TASK_CMPR*n*_A_UP_EN	MCPWM*m*_TASK_CMPR*n*_A_UP	将 PWM 操作器 *n* 的时间戳 A 更新为时间戳 A 影子寄存器 MCPWM_GEN*n*_A 的值

59.4 中断

ESP32-P4 的 MCPWM0 和 MCPWM1 会分别生成 PWM0_INTR 和 PWM1_INTR 中断信号，并发送给中断矩阵。

PWM*n*_INTR 中断信号由以下内部中断源生成：

- MCPWM_TIMER*x*_STOP_INT：定时器*x* 停止后触发。
- MCPWM_TIMER*x*_TEZ_INT：由 PWM 定时器*x* TEZ 事件触发。
- MCPWM_TIMER*x*_TEP_INT：由 PWM 定时器*x* TEP 事件触发。
- MCPWM_FAULT*x*_INT：fault_event*x* 开始后触发。
- MCPWM_FAULT*x*_CLR_INT：fault_event*x* 结束后触发。
- MCPWM_CMPR*x*_TEA_INT：由 PWM 操作器*x* TEA 事件触发。
- MCPWM_CMPR*x*_TEB_INT：由 PWM 操作器*x* TEB 事件触发。
- MCPWM_TZ*x*_CBC_INT：由 PWM*x* 逐周期操作触发。
- MCPWM_TZ*x*_OST_INT：由 PWM*x* 一次性操作触发。
- MCPWM_CAP*x*_INT：由信道*x* 上捕获事件触发。

上述中断源仅在对应的中断寄存器使能字段 *interrupt_source_name*_ENA 置 1 时才能触发，将 *interrupt_source_name*_CLR 字段置 1 即清除对应中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 59.5 寄存器列表。

59.5 寄存器列表

本小节的所有地址均为相对于电机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
预分频器配置			
MCPWM_CLK_CFG_REG	配置预分频器	0x0000	R/W
MCPWM 定时器 0 配置与状态			
MCPWM_TIMER0_CFG0_REG	定时器周期与更新方法	0x0004	R/W
MCPWM_TIMER0_CFG1_REG	工作模式与开始/停止控制	0x0008	varies
MCPWM_TIMER0_SYNC_REG	PWM 定时器 0 同步功能配置寄存器	0x000C	R/W
MCPWM_TIMER0_STATUS_REG	PWM 定时器 0 状态	0x0010	RO
MCPWM 定时器 1 配置与状态			
MCPWM_TIMER1_CFG0_REG	定时器更新方式与周期	0x0014	R/W
MCPWM_TIMER1_CFG1_REG	工作模式与开始/停止控制	0x0018	varies
MCPWM_TIMER1_SYNC_REG	同步设置	0x001C	R/W
MCPWM_TIMER1_STATUS_REG	PWM 定时器 1 状态	0x0020	RO
MCPWM 定时器 2 配置与状态			
MCPWM_TIMER2_CFG0_REG	定时器方式与周期	0x0024	R/W
MCPWM_TIMER2_CFG1_REG	工作模式与开始/停止控制	0x0028	varies
MCPWM_TIMER2_SYNC_REG	同步设置	0x002C	R/W
MCPWM_TIMER2_STATUS_REG	PWM 定时器 2 状态	0x0030	RO
MCPWM 定时器常见配置			
MCPWM_TIMER_SYNCI_CFG_REG	定时器同步输入选择	0x0034	R/W
MCPWM_OPERATOR_TIMERSEL_REG	为 PWM 操作器选择特定的计时器	0x0038	R/W
MCPWM 操作器 0 配置与状态			
MCPWM_GEN0_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x003C	varies
MCPWM_GEN0_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0040	R/W
MCPWM_GEN0_TSTMP_B_REG	寄存器 B 的影子寄存器	0x0044	R/W
MCPWM_GEN0_CFG0_REG	故障事件 T0 和 T1 处理	0x0048	R/W
MCPWM_GEN0_FORCE_REG	软件强制 PWM0A 和 PWM0B 输出	0x004C	R/W
MCPWM_GEN0_A_REG	PWM0A 输出上事件触发的操作	0x0050	R/W
MCPWM_GEN0_B_REG	PWM0B 输出上事件触发的操作	0x0054	R/W
MCPWM_DT0_CFG_REG	死区配置	0x0058	R/W
MCPWM_DT0_FED_CFG_REG	FED 的影子寄存器	0x005C	R/W
MCPWM_DT0_RED_CFG_REG	RED 的影子寄存器	0x0060	R/W
MCPWM_CARRIER0_CFG_REG	载波使能与配置	0x0064	R/W
MCPWM_FH0_CFG0_REG	故障事件中 PWM0A 和 PWM0B 上的操作	0x0068	R/W
MCPWM_FH0_CFG1_REG	故障处理的软件触发	0x006C	R/W
MCPWM_FH0_STATUS_REG	故障事件状态	0x0070	RO
MCPWM 操作器 1 配置与状态			
MCPWM_GEN1_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x0074	varies

名称	描述	地址	访问
MCPWM_GEN1_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0078	R/W
MCPWM_GEN1_TSTMP_B_REG	寄存器 B 的影子寄存器	0x007C	R/W
MCPWM_GEN1_CFG0_REG	故障事件 T0 和 T1 处理	0x0080	R/W
MCPWM_GEN1_FORCE_REG	软件强制 PWM1A 和 PWM1B 输出	0x0084	R/W
MCPWM_GEN1_A_REG	PWM1A 输出上的事件触发的操作	0x0088	R/W
MCPWM_GEN1_B_REG	PWM1B 输出上的事件触发的操作	0x008C	R/W
MCPWM_DT1_CFG_REG	死区配置	0x0090	R/W
MCPWM_DT1_FED_CFG_REG	FED 的影子寄存器	0x0094	R/W
MCPWM_DT1_RED_CFG_REG	RED 的影子寄存器	0x0098	R/W
MCPWM_CARRIER1_CFG_REG	使能与配置载波	0x009C	R/W
MCPWM_FH1_CFG0_REG	故障事件中 PWM1A 和 PWM1B 输出上的操作	0x00A0	R/W
MCPWM_FH1_CFG1_REG	故障处理的软件触发	0x00A4	R/W
MCPWM_FH1_STATUS_REG	故障事件状态	0x00A8	RO
MCPWM 操作器 2 的配置与状态			
MCPWM_GEN2_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x00AC	varies
MCPWM_GEN2_TSTMP_A_REG	寄存器 A 的影子寄存器	0x00B0	R/W
MCPWM_GEN2_TSTMP_B_REG	寄存器 B 的影子寄存器	0x00B4	R/W
MCPWM_GEN2_CFG0_REG	故障事件 T0 和 T1 处理	0x00B8	R/W
MCPWM_GEN2_FORCE_REG	软件强制 PWM2A 和 PWM2B 输出	0x00BC	R/W
MCPWM_GEN2_A_REG	PWM2A 输出上的事件触发的操作	0x00C0	R/W
MCPWM_GEN2_B_REG	PWM2B 输出上的事件触发的操作	0x00C4	R/W
MCPWM_DT2_CFG_REG	死区类型的选择与配置	0x00C8	R/W
MCPWM_DT2_FED_CFG_REG	FED 影子寄存器	0x00CC	R/W
MCPWM_DT2_RED_CFG_REG	RED 影子寄存器	0x00D0	R/W
MCPWM_CARRIER2_CFG_REG	使能与配置载波	0x00D4	R/W
MCPWM_FH2_CFG0_REG	故障事件中 PWM2A 和 PWM2B 输出上的操作	0x00D8	R/W
MCPWM_FH2_CFG1_REG	故障处理的软件触发	0x00DC	R/W
MCPWM_FH2_STATUS_REG	故障事件状态	0x00E0	RO
故障检测与配置			
MCPWM_FAULT_DETECT_REG	故障检测与配置	0x00E4	varies
捕获配置与状态			
MCPWM_CAP_TIMER_CFG_REG	配置捕获定时器	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	捕获定时器同步相位	0x00EC	R/W
MCPWM_CAP_CH0_CFG_REG	捕获通道 0 的配置与使能	0x00F0	varies
MCPWM_CAP_CH1_CFG_REG	捕获通道 1 的配置与使能	0x00F4	varies
MCPWM_CAP_CH2_CFG_REG	捕获通道 2 的配置与使能	0x00F8	varies
MCPWM_CAP_CH0_REG	捕获通道 0 值的状态	0x00FC	RO
MCPWM_CAP_CH1_REG	捕获通道 1 值的状态	0x0100	RO
MCPWM_CAP_CH2_REG	捕获通道 2 值的状态	0x0104	RO
MCPWM_CAP_STATUS_REG	上一次捕获触发器的边沿	0x0108	RO
使能有效寄存器的更新			
MCPWM_UPDATE_CFG_REG	使能更新	0x010C	R/W
管理中断			

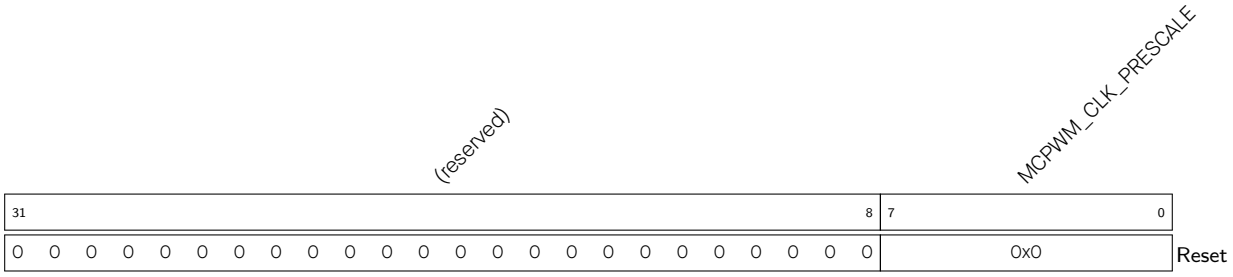
名称	描述	地址	访问
MCPWM_INT_ENA_REG	中断使能位	0x0110	R/W
MCPWM_INT_RAW_REG	原始中断状态	0x0114	R/WTC/SS
MCPWM_INT_ST_REG	屏蔽中断状态	0x0118	RO
MCPWM_INT_CLR_REG	中断清除位	0x011C	WT
使能 MCPWM 事件			
MCPWM_EVT_EN_REG	使能 MCPWM 事件	0x0120	R/W
MCPWM_EVT_EN2_REG	使能 MCPWM 事件	0x0128	R/W
使能 MCPWM 任务			
MCPWM_TASK_EN_REG	使能 MCPWM 任务	0x0124	R/W
MCPWM 生成器配置			
MCPWM_OP0_TSTMP_E1_REG	配置生成器 0 时间戳 E1 值	0x012C	R/W
MCPWM_OP0_TSTMP_E2_REG	配置生成器 0 时间戳 E2 值	0x0130	R/W
MCPWM_OP1_TSTMP_E1_REG	配置生成器 1 时间戳 E1 值	0x0134	R/W
MCPWM_OP1_TSTMP_E2_REG	配置生成器 1 时间戳 E2 值	0x0138	R/W
MCPWM_OP2_TSTMP_E1_REG	配置生成器 2 时间戳 E1 值	0x013C	R/W
MCPWM_OP2_TSTMP_E2_REG	配置生成器 2 时间戳 E2 值	0x0140	R/W
MCPWM APB 配置			
MCPWM_CLK_REG	MCPWM APB 配置	0x0144	R/W
版本寄存器			
MCPWM_VERSION_REG	版本控制寄存器	0x0148	R/W

59.6 寄存器

本小节的所有地址均为相对于电机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

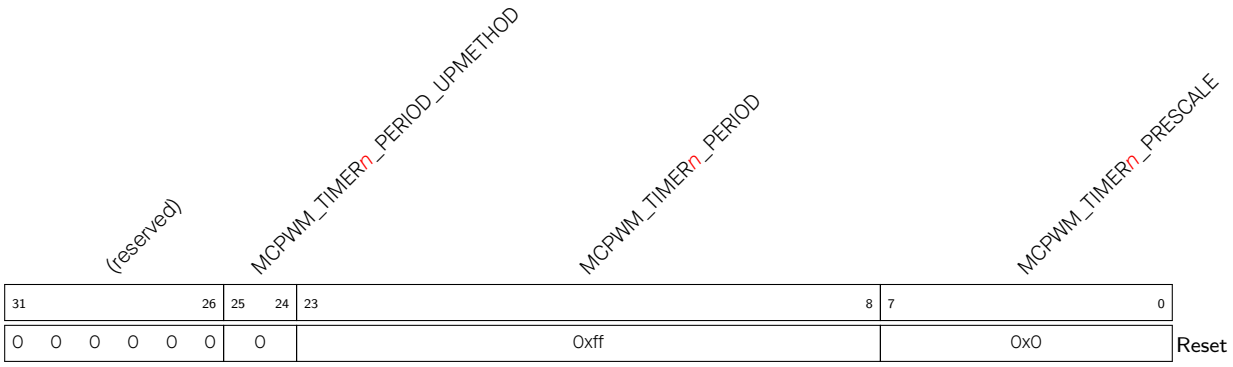
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 59.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE 配置时钟预分频系数，使得 PWM_CLK 的周期为 6.25ns * (PWM_CLK_PRESCALE + 1)。(R/W)

Register 59.2. MCPWM_TIMER*n*_CFG0_REG (*n*: 0-2) (0x0004+0x10**n*)

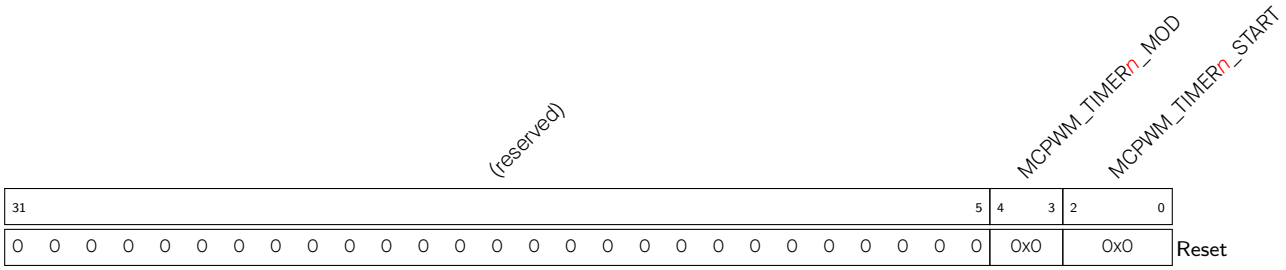


MCPWM_TIMER*n*_PRESCALE 配置定时器*n* 的预分频系数，使得 PTO_CLK 的周期 = PWM_CLK 的周期 * (PWM_TIMER*n*_PRESCALE + 1)。
(R/W)

MCPWM_TIMER*n*_PERIOD 配置定时器*n* 的影子周期。
(R/W)

MCPWM_TIMER*n*_PERIOD_UPMETHOD 配置 PWM 定时器*n* 周期有效寄存器的更新方式。
0：立即更新
1：发生 TEZ 事件时更新
2：同步时更新
3：发生 TEZ 事件或同步时更新
本文档中，TEZ 指定时器为 0 时的事件。
(R/W)

Register 59.3. MCPWM_TIMER*n*_CFG1_REG(*n*: 0-2) (0x0008+0x10**n*)



MCPWM_TIMER*n*_START 控制 PWM 定时器的开启与关闭。

0：如果开启，在 TEZ 事件发生时停止

1：如果开启，在 TEP 事件发生时停止

2：开启

3：开启，并在下一个 TEZ 事件发生时停止

4：开启，并在下一个 TEP 事件发生时停止

其他值：无效值，不产生影响

本文档中，TEP 指定时器为周期值时发生的事件。

(R/W/SC)

MCPWM_TIMER*n*_MOD 配置 PWM 定时器 0 的工作模式。

0：暂停

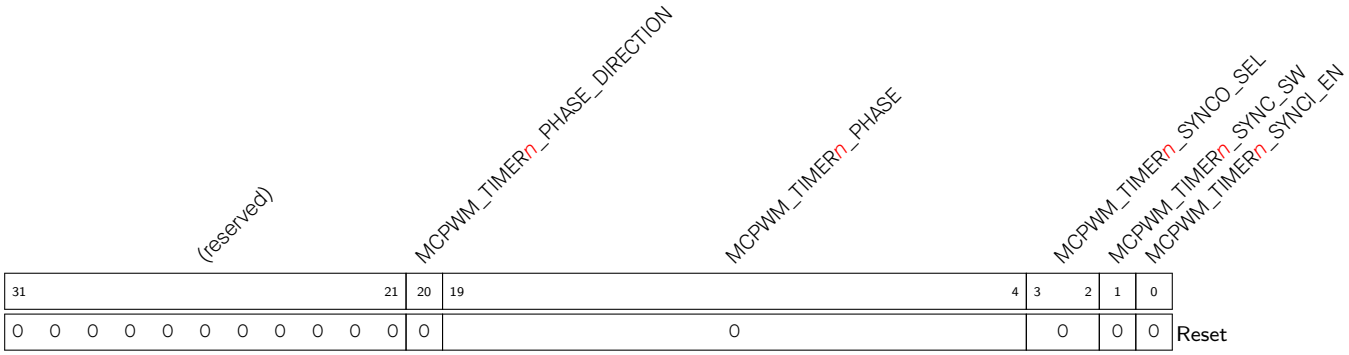
1：递增模式

2：递减模式

3：递增递减循环模式

(R/W)

Register 59.4. MCPWM_TIMER*n*_SYNC_REG (*n*: 0-2) (0x000C+0x10**n*)



MCPWM_TIMER*n*_SYNCI_EN 配置是否开启同步输入事件发生时的定时器*n* 相位重载。

0：关闭

1：开启

(R/W)

MCPWM_TIMER*n*_SYNC_SW 配置是否触发软件同步。

0：无效

1：触发软件同步

(R/W)

MCPWM_TIMER*n*_SYNCO_SEL 选择 PWM 定时器*n* 的同步输出来源。

0：同步。取反 MCPWM_TIMER0_SYNC_SW 位时始终生成同步输出。

1：TEZ

2：TEP

3：无效

(R/W)

MCPWM_TIMER*n*_PHASE 同步事件中定时器*n* 重载的相位。

(R/W)

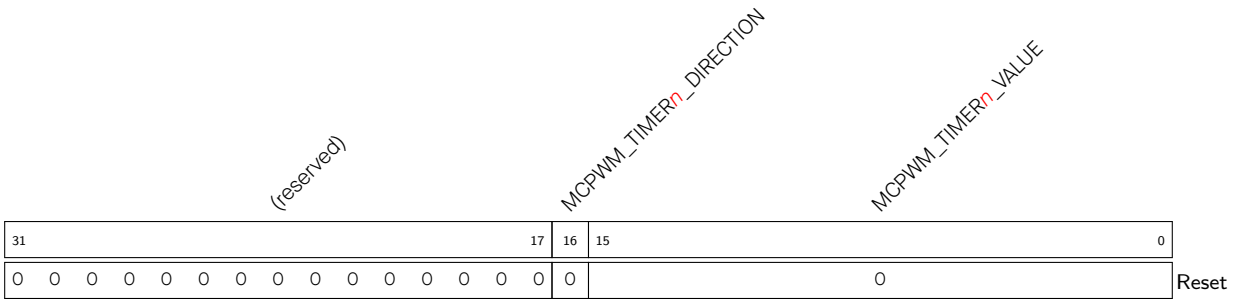
MCPWM_TIMER*n*_PHASE_DIRECTION 当定时器*n* 为递增-递减循环模式时，配置该定时器方向。

0：递增

1：递减

(R/W)

Register 59.5. MCPWM_TIMER*n*_STATUS_REG(*n*: 0-2) (0x0010+0x10**n*)



MCPWM_TIMER*n*_VALUE 表示当前 PWM 定时器*n* 计数器的值。(RO)

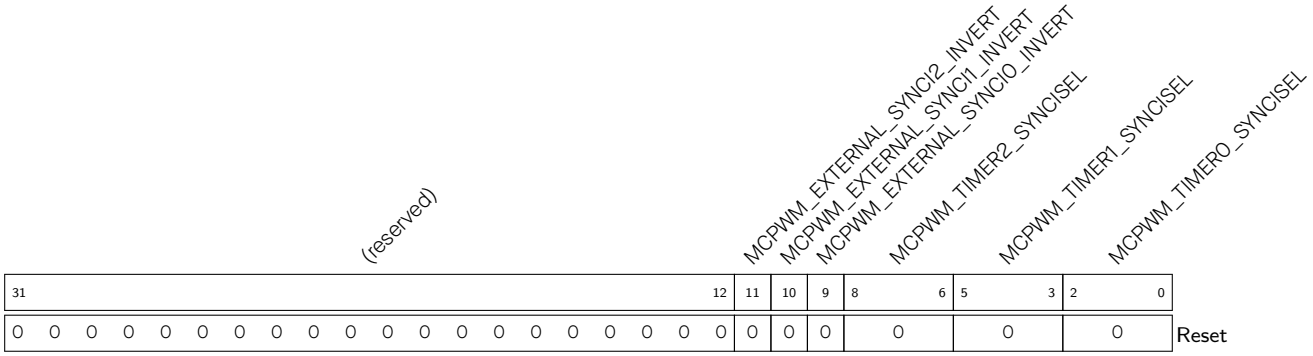
MCPWM_TIMER*n*_DIRECTION 配置当前 PWM 定时器*n* 的计数器模式。

0：递增模式

1：递减模式

(RO)

Register 59.6. MCPWM_TIMER_SYNCI_CFG_REG (0x0034)



MCPWM_TIMER*n*_SYNCISEL 选择 PWM 定时器*n* 的同步输入来源。

1：PWM 定时器 0 同步输出

2：PWM 定时器 1 同步输出

3：PWM 定时器 2 同步输出

4：来自 GPIO 矩阵的 SYNC0

5：来自 GPIO 矩阵的 SYNC1

6：来自 GPIO 矩阵的 SYNC2

其他值：未选择任何同步输入

(R/W)

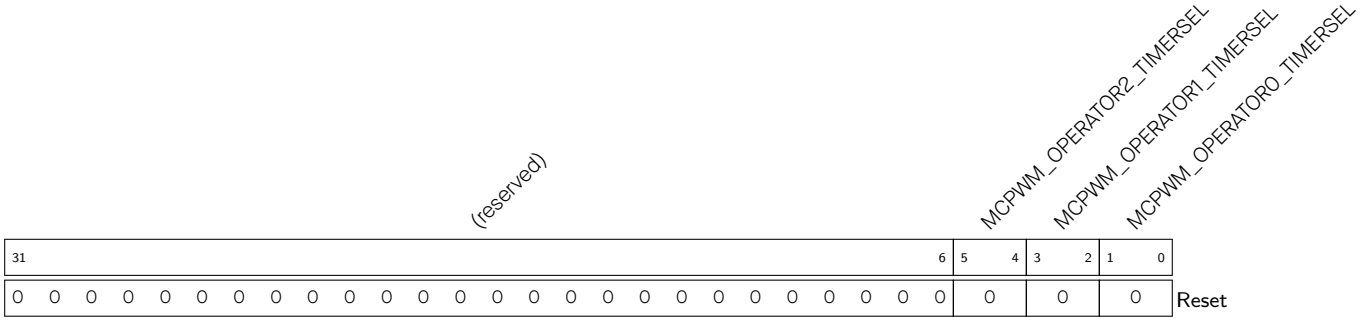
MCPWM_EXTERNAL_SYNCI*n*_INVERT 配置是否将来自 GPIO 矩阵的 SYNC*n* 反相。

0：不反相

1：反相

(R/W)

Register 59.7. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR*n*_TIMERSEL 选择 PWM 操作器*n* 的定时参考来源。

0：定时器 0

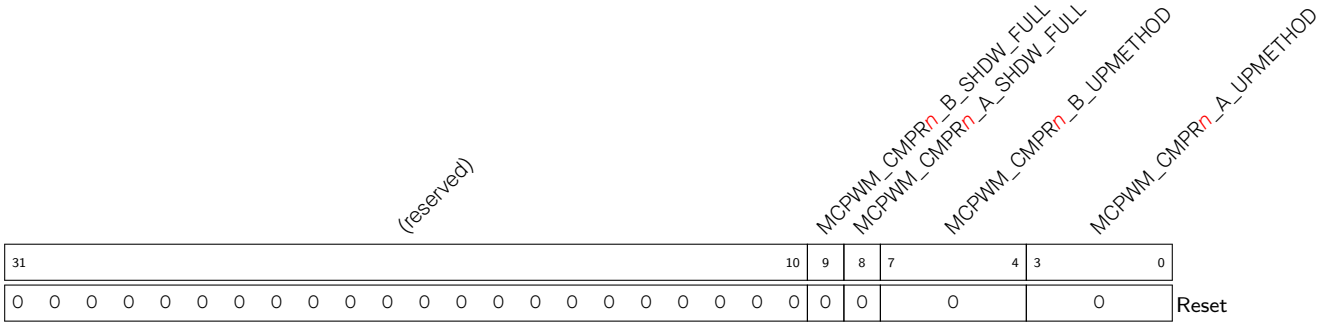
1：定时器 1

2：定时器 2

3：无效，将选择定时器 2

(R/W)

Register 59.8. MCPWM_GEN*n*_STMP_CFG_REG (*n*: 0-2) (0x003C+0x38**n*)



MCPWM_CMPR*n*_A_UPMETHOD 配置 PWM 生成器*n* 时间戳寄存器 A 的有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_CMPR*n*_B_UPMETHOD 配置 PWM 生成器*n* 时间戳寄存器 B 的有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_CMPR*n*_A_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 A 中

1：将值写入 PWM 生成器*n* 时间戳寄存器 A 的影子寄存器中，等待传输给有效寄存器 A。

(R/SC/WTC)

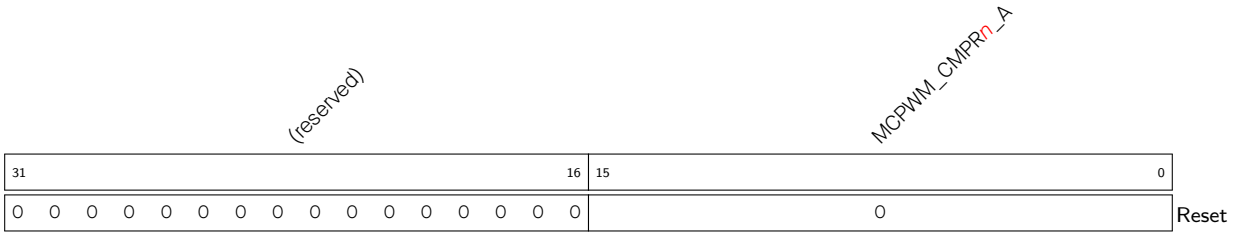
MCPWM_CMPR*n*_B_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 B 中

1：将值写入 PWM 生成器*n* 时间戳寄存器 B 的影子寄存器中，等待传输给有效寄存器 B。

(R/SC/WTC)

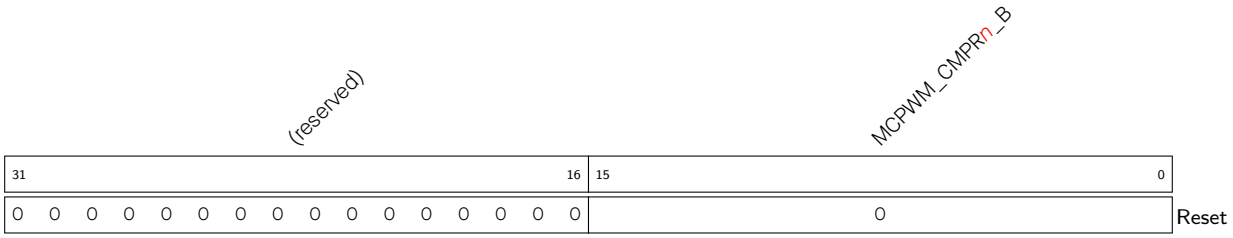
Register 59.9. MCPWM_GEN*n*_TSTMP_A_REG(*n*: 0-2) (0x0040+0x38**n*)



MCPWM_CMPR*n*_A 配置 PWM 生成器*n* 时间戳寄存器 A 的影子寄存器。

(R/W)

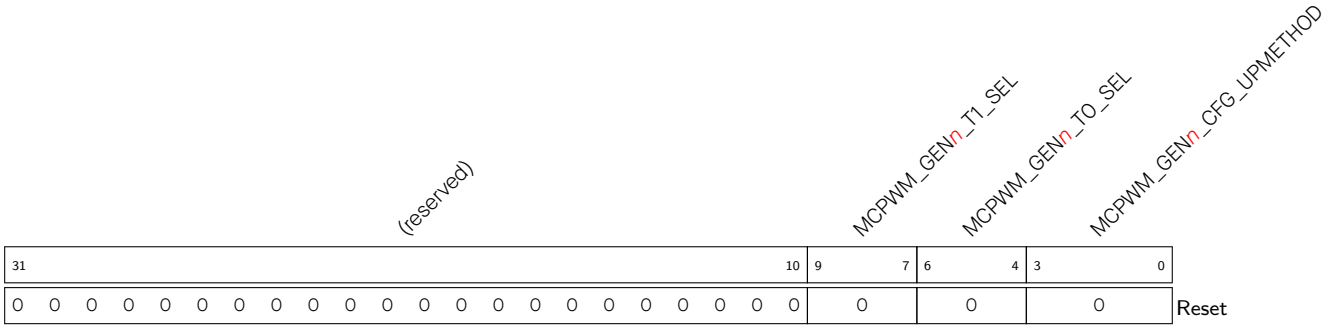
Register 59.10. MCPWM_GEN*n*_TSTMP_B_REG(*n*: 0-2) (0x0044+0x38**n*)



MCPWM_CMPR*n*_B 配置 PWM 生成器*n* 时间戳寄存器 B 的影子寄存器。

(R/W)

Register 59.11. MCPWM_GEN*n*_CFG0_REG(*n*: 0-2) (0x0048+0x38**n*)



MCPWM_GEN*n*_CFG_UPMETHOD 配置 PWM 生成器*n* 有效配置寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN*n*_TO_SEL 选择 PWM 生成器*n* event_t0 的信号源，立即生效。

0：fault_event0

1：fault_event1

2：fault_event2

3：sync_taken

4：无

(R/W)

MCPWM_GEN*n*_T1_SEL 选择 PWM 生成器*n* event_t1 的信号源，立即生效。

0：fault_event0

1：fault_event1

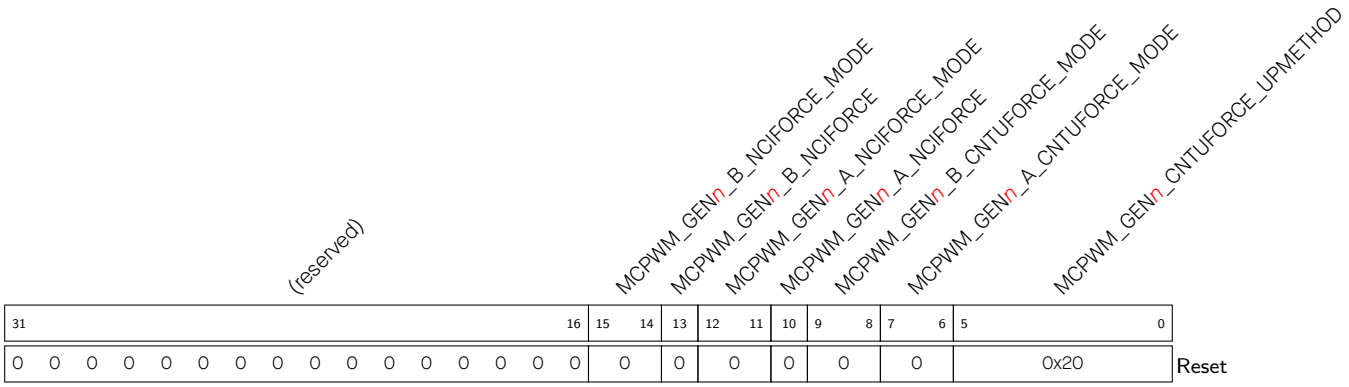
2：fault_event2

3：sync_taken

4：无

(R/W)

Register 59.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)



MCPWM_GEN*n*_CNTUFORCE_UPMETHOD 配置生成器*n* 的连续软件强制事件更新方式。

所有 bit 为 0 时：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生 TEA 事件时更新

bit3 为 1：发生 TEB 事件时更新

bit4：发生同步事件时更新

bit5 为 1：关闭更新

本文档中，TEA/B 指定时器值为寄存器 A/B 的值时生成的事件。

(R/W)

MCPWM_GEN*n*_A_CNTUFORCE_MODE 配置 PWM*n* A 的连续软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN*n*_B_CNTUFORCE_MODE 配置 PWM*n* B 的连续软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN*n*_A_NCIFORCE 配置是否触发 PWM*n* A 上的非连续即时软件强制事件。

0：无效

1：触发 PWM*n* A 上的非连续即时软件强制事件

(R/W)

见下页...

Register 59.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)

... 接上页

MCPWM_GEN*n*_A_NCIFORCE_MODE 配置用于 PWM*n* A 的非连续即时软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN*n*_B_NCIFORCE 配置是否触发 PWM*n* B 上的非连续即时软件强制事件。

0：无效

1：触发 PWM*n* B 上的非连续即时软件强制事件

(R/W)

MCPWM_GEN*n*_B_NCIFORCE_MODE 配置用于 PWM*n* B 的非连续即时软件强制模式。

0：关闭

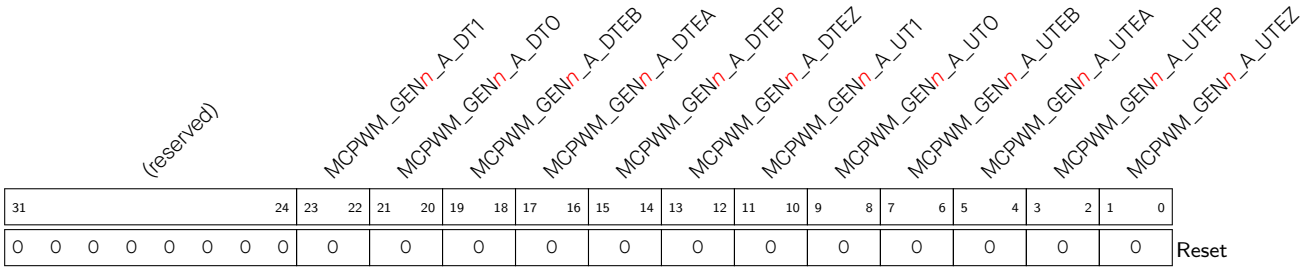
1：低电平

2：高电平

3：关闭

(R/W)

Register 59.13. MCPWM_GEN*n*_A_REG (*n*: 0-2) (0x0050+0x38**n*)



MCPWM_GEN*n*_A_UTEZ 配置定时器递增时，TEZ 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_UTEP 配置定时器递增时，TEP 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_UTEA 配置定时器递增时，TEA 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_UTEB 配置定时器递增时，TEB 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

见下页...

Register 59.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

... 接上页

MCPWM_GEN*n*_A_UT0 配置定时器递增时，event_t0 在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_UT1 配置定时器递增时，event_t1 在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_DTEZ 配置定时器递减时，TEZ 事件在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_DTEP 配置定时器递减时，TEP 事件在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_DTEA 配置定时器递减时，TEA 事件在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 59.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

... 接上页

MCPWM_GEN*n*_A_DTEB 配置定时器递减时，TEB 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_DTO 配置定时器递减时，event_t0 在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_DT1 配置定时器递减时，event_t1 在 PWM*n*A 上触发的操作。

0：波形无改变

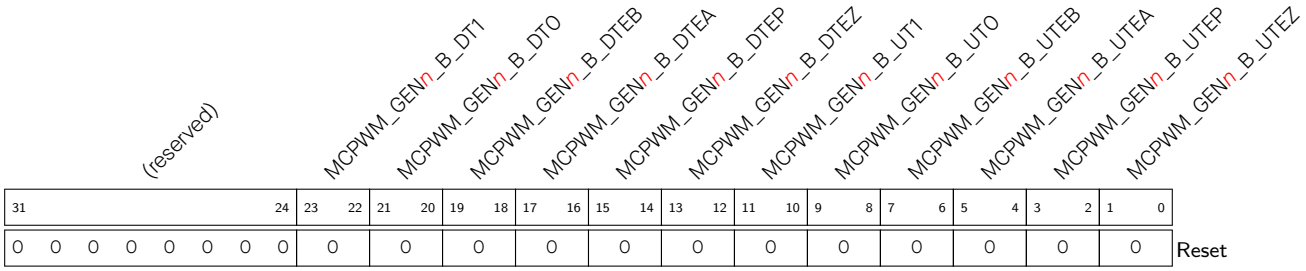
1：拉低

2：拉高

3：取反

(R/W)

Register 59.14. MCPWM_GEN*n*_B_REG (*n*: 0-2) (0x0054+0x38**n*)



MCPWM_GEN*n*_B_UTEZ 定时器递增时，TEZ 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_UTEP 定时器递增时，TEP 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_UTEA 定时器递增时，TEA 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_UTEB 定时器递增时，TEB 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

见下页...

Register 59.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

... 接上页

MCPWM_GEN*n*_B_UT0 定时器递增时，event_t0 在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_UT1 定时器递增时，event_t1 在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_DTEZ 定时器递减时，TEZ 事件在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_DTEP 定时器递减时，TEP 事件在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_DTEA 定时器递减时，TEA 事件在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 59.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

... 接上页

MCPWM_GEN*n*_B_DTEB 定时器递减时，TEB 事件在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_DT0 定时器递减时，event_t0 事件在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_DT1 定时器递减时，event_t1 在 PWM*n*B 上触发的操作。

0：波形无改变

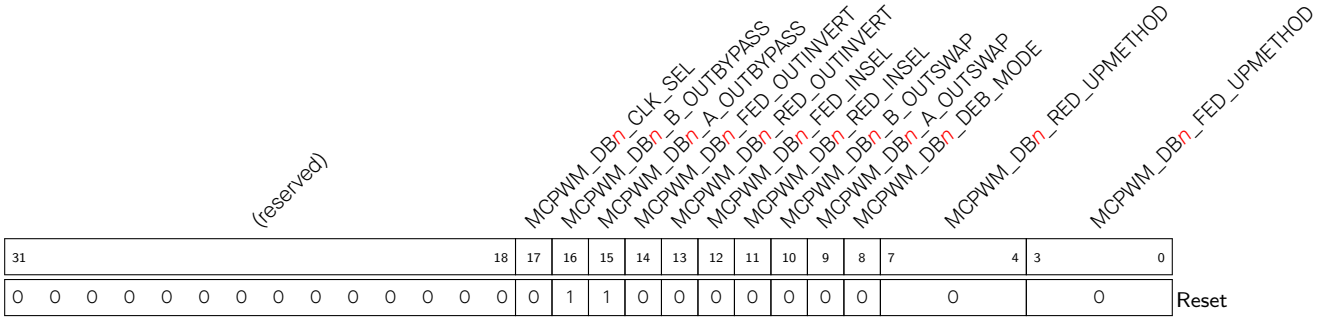
1：拉低

2：拉高

3：取反

(R/W)

Register 59.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)



MCPWM_DB*n*_FED_UPMETHOD 配置下降沿延迟有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DB*n*_RED_UPMETHOD 配置上升沿延迟有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DB*n*_DEB_MODE 配置表 59.3-5 中的 S8 开关。详细配置信息请参考表 59.3-6。

0：FED/RED 分别在不同路径上生效

1：FED/RED 在 B 路径上生效，A 输出为旁路或 dulpB 模式

(R/W)

MCPWM_DB*n*_A_OUTSWAP 配置表 59.3-5 中的 S6 开关。详细配置信息请参考表 59.3-6。(R/W)

MCPWM_DB*n*_B_OUTSWAP 配置表 59.3-5 中的 S7 开关。详细配置信息请参考表 59.3-6。(R/W)

MCPWM_DB*n*_RED_INSEL 配置表 59.3-5 中的 S4 开关。详细配置信息请参考表 59.3-6。(R/W)

MCPWM_DB*n*_FED_INSEL 配置表 59.3-5 中的 S5 开关。详细配置信息请参考表 59.3-6。(R/W)

MCPWM_DB*n*_RED_OUTINVERT 配置表 59.3-5 中的 S2 开关。详细配置信息请参考表 59.3-6。(R/W)

MCPWM_DB*n*_FED_OUTINVERT 配置表 59.3-5 中的 S3 开关。详细配置信息请参考表 59.3-6。(R/W)

MCPWM_DB*n*_A_OUTBYPASS 配置表 59.3-5 中的 S1 开关。详细配置信息请参考表 59.3-6。(R/W)

MCPWM_DB*n*_B_OUTBYPASS 配置表 59.3-5 中的 S0 开关。详细配置信息请参考表 59.3-6。(R/W)

见下页...

Register 59.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)

... 接上页

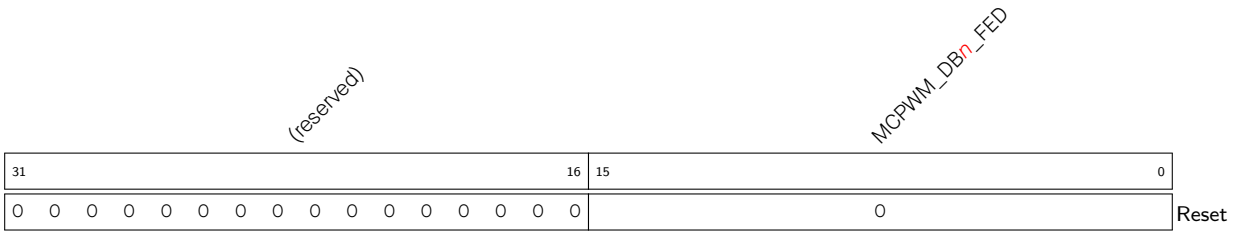
MCPWM_DB*n*_CLK_SEL 选择死区时间生成器 0 的时钟。

0：PWM_CLK

1：PT_CLK

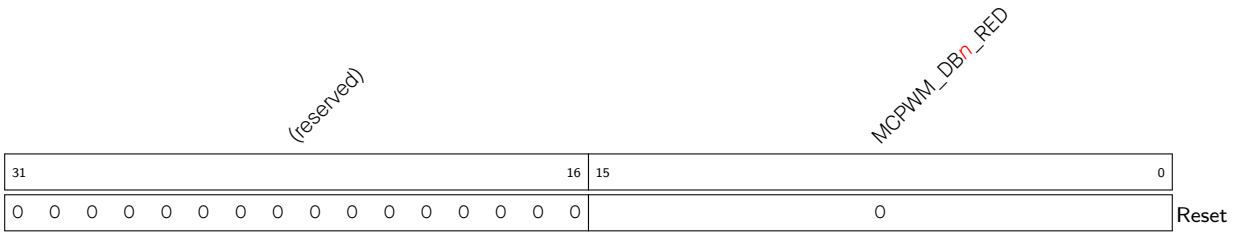
(R/W)

Register 59.16. MCPWM_DT*n*_FED_CFG_REG(*n*: 0-2) (0x005C+0x38**n*)



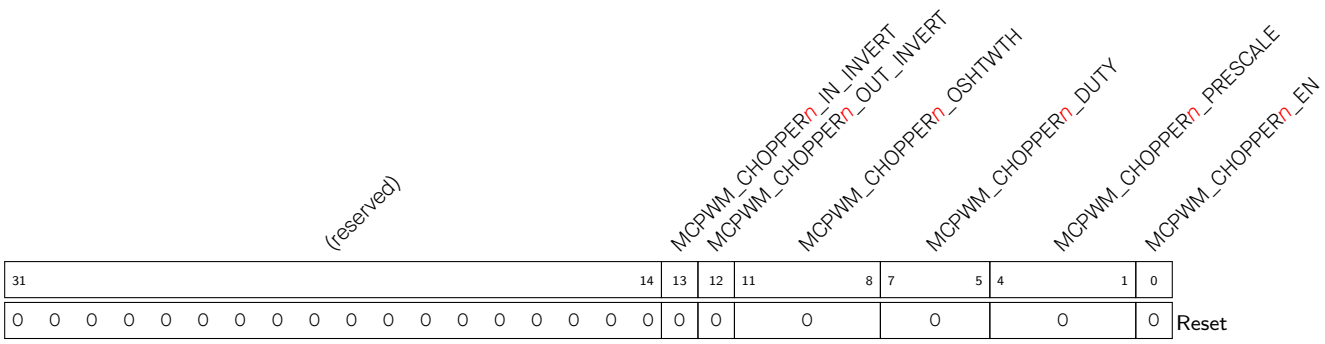
MCPWM_DB*n*_FED 配置 FED 的影子寄存器。(R/W)

Register 59.17. MCPWM_DT*n*_RED_CFG_REG(*n*: 0-2) (0x0060+0x38**n*)



MCPWM_DB*n*_RED 配置 RED 的影子寄存器。(R/W)

Register 59.18. MCPWM_CARRIER*n*_CFG_REG(*n*: 0-2) (0x0064+0x38**n*)



MCPWM_CHOPPER*n*_EN 配置是否使能载波*n*。

0：绕过

1：使能

(R/W)

MCPWM_CHOPPER*n*_PRESCALE 配置 PWM 载波*n* 时钟 (PC_CLK) 的预分频值。PC_CLK 周期 = PWM_CLK 周期 * (PWM_CARRIER*n*_PRESCALE + 1)。(R/W)

MCPWM_CHOPPER*n*_DUTY 配置载波占空比。占空比 = PWM_CARRIER*n*_DUTY/8。(R/W)

MCPWM_CHOPPER*n*_OSHTWTH 配置载波第一个脉冲的宽度，单位为载波周期。(R/W)

MCPWM_CHOPPER*n*_OUT_INVERT 配置是否将此模块的 PWM*n*A 和 PWM*n*B 输出反相。

0：正常输出

1：反相

(R/W)

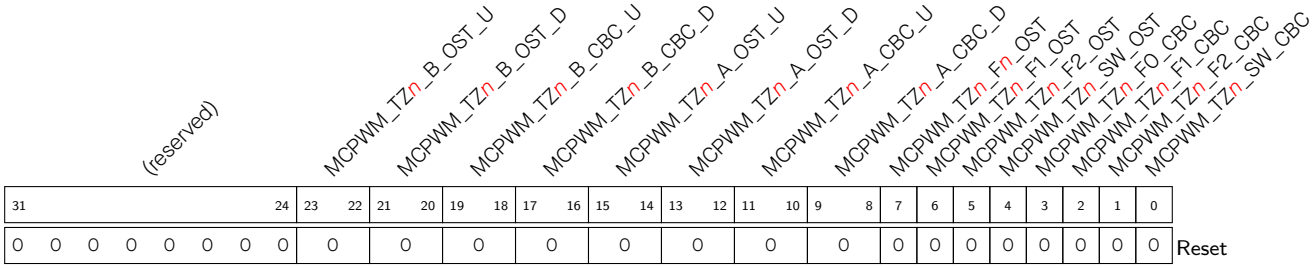
MCPWM_CHOPPER*n*_IN_INVERT 配置是否将此模块的 PWM*n*A 和 PWM*n*B 输入反相。

0：正常输入

1：反相

(R/W)

Register 59.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)



MCPWM_TZ*n*_SW_CBC 配置是否使能软件强制逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F2_CBC 配置 fault_event2 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F1_CBC 配置 fault_event1 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F0_CBC 配置 fault_event0 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_SW_OST 配置是否使能软件强制一次性模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F2_OST 配置 fault_event2 是否触发一次性模式操作。

0：关闭

1：使能

(R/W)

见下页...

Register 59.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

... 接上页

MCPWM_TZ*n*_F1_OST 配置 fault_event1 是否触发一次性模式操作。

0：关闭
1：使能
(R/W)

MCPWM_TZ*n*_F*n*_OST 配置 fault_event0 是否触发一次性模式操作。

0：关闭
1：使能
(R/W)

MCPWM_TZ*n*_A_CBC_D 配置定时器递减计数并且发生故障事件时，PWM*n*A 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_A_CBC_U 配置定时器递增计数并且发生故障事件时，PWM*n*A 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_A_OST_D 配置定时器递减计数并且发生故障事件时，PWM*n*A 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

见下页...

Register 59.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

... 接上页

MCPWM_TZ*n*_A_OST_U 配置定时器递增计数并且发生故障事件时，PWM*n*A 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_B_CBC_D 配置定时器递减计数并且发生故障事件时，PWM*n*B 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_B_CBC_U 配置定时器递增计数并且发生故障事件时，PWM*n*B 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

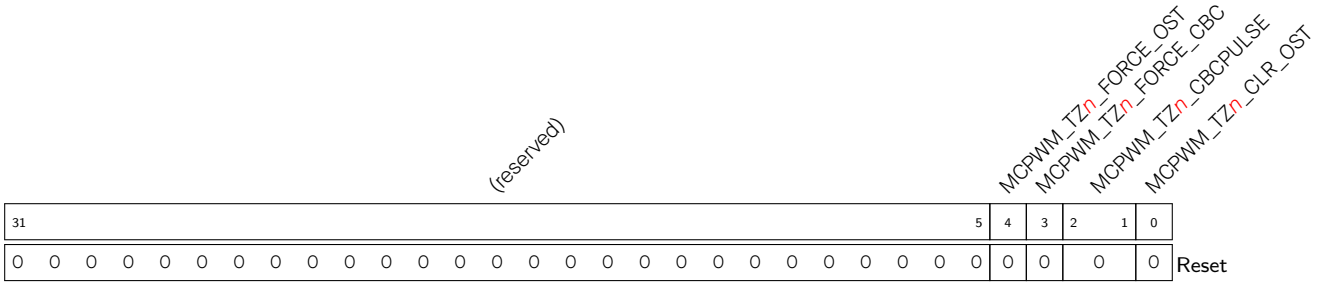
MCPWM_TZ*n*_B_OST_D 配置定时器递减计数并且发生故障事件时，PWM*n*B 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_B_OST_U 配置定时器递增计数并且发生故障事件时，PWM*n*B 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

Register 59.20. MCPWM_FH*n*_CFG1_REG(*n*: 0-2) (0x006C+0x38**n*)



MCPWM_TZ*n*_CLR_OST 配置是否通过软件清除持续一次性模式操作。

0：无效

1：通过软件清除持续一次性模式操作

(R/W)

MCPWM_TZ*n*_CBCPULSE 配置逐周期模式操作更新的时间点。

bit0 为 1：发生 TEZ 事件时

bit1 为 1：发生 TEP 事件时

(R/W)

MCPWM_TZ*n*_FORCE_CBC 配置是否触发逐周期模式的操作。

0：无效

1：触发

(R/W)

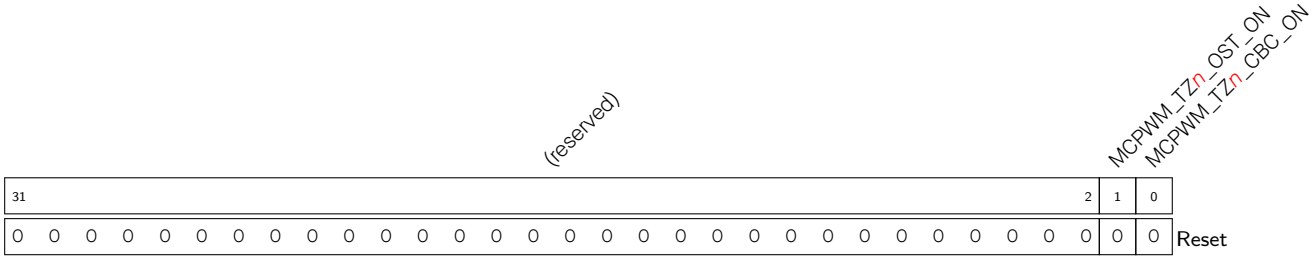
MCPWM_TZ*n*_FORCE_OST 配置是否触发一次性模式的操作。

0：无效

1：触发

(R/W)

Register 59.21. MCPWM_FH*n*_STATUS_REG(*n*: 0-2) (0x0070+0x38**n*)



MCPWM_TZ*n*_CBC_ON 表示是否正在进行逐周期模式的操作。

0：不存在正在进行的逐周期模式操作

1：逐周期模式的操作正在进行

(RO)

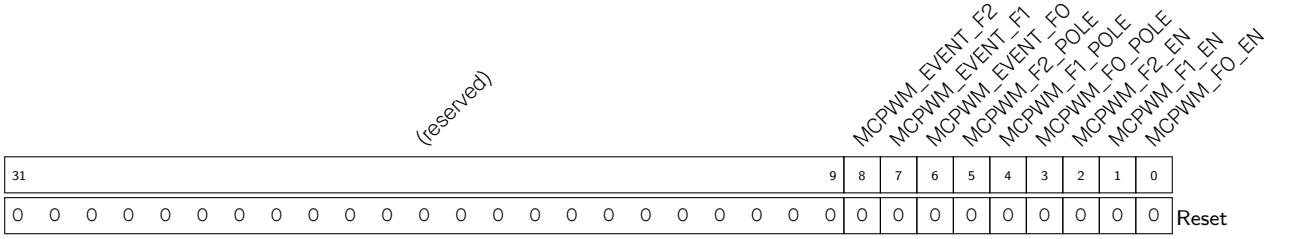
MCPWM_TZ*n*_OST_ON 表示是否正在进行一次性模式的操作。

0：不存在正在进行的一次性模式操作

1：一次性模式的操作正在进行

(RO)

Register 59.22. MCPWM_FAULT_DETECT_REG (0x00E4)



MCPWM_F*n*_EN 配置是否使能生成 fault_event*n*。

0：无效

1：使能

(R/W)

MCPWM_F*n*_POLE 配置来自 GPIO 矩阵的 FAULT*n* 信号源触发 fault_event*n* 时极性。

0：低电平触发

1：高电平触发

(R/W)

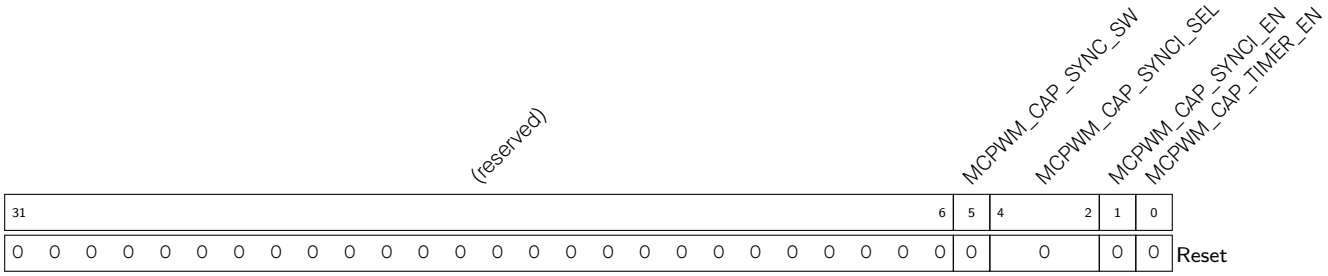
MCPWM_EVENT_F*n* 表示 fault_event*n* 事件的状态。此字段由硬件置 1 和清零。

0：fault_event*n* 事件停止

1：fault_event*n* 事件持续

(RO)

Register 59.23. MCPWM_CAP_TIMER_CFG_REG (0x00E8)



MCPWM_CAP_TIMER_EN 配置是否使能捕获定时器在 APB_CLK 下的递增。

0：无效

1：使能捕获定时器在 APB_CLK 下的递增

(R/W)

MCPWM_CAP_SYNCI_EN 配置是否使能捕获定时器同步。

0：无效

1：使能捕获定时器同步

(R/W)

MCPWM_CAP_SYNCI_SEL 配置捕获模块的同步输入。

0：无

1：定时器 0 的同步输出

2：定时器 1 的同步输出

3：定时器 2 的同步输出

4：来自 GPIO 矩阵的 SYNC0

5：来自 GPIO 矩阵的 SYNC1

6：来自 GPIO 矩阵的 SYNC2

7：无

(R/W)

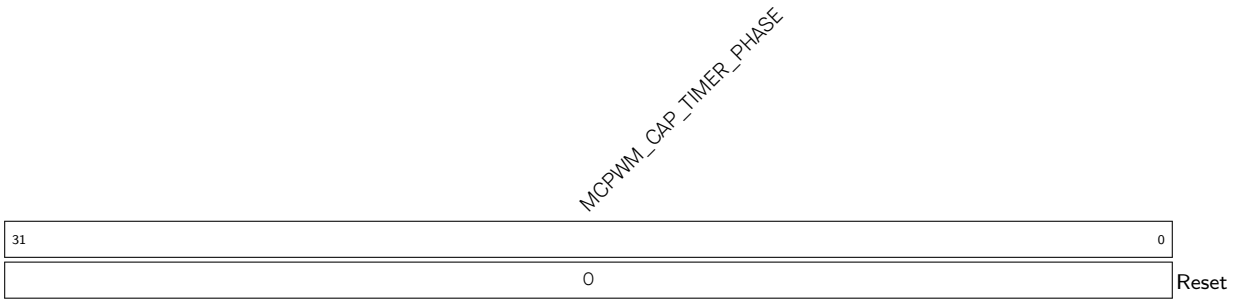
MCPWM_CAP_SYNC_SW 当 MCPWM_CAP_SYNCI_EN 置位时，配置是否同步捕获定时器，在捕获定时器中写入相位寄存器的值。

0：无效

1：同步捕获定时器

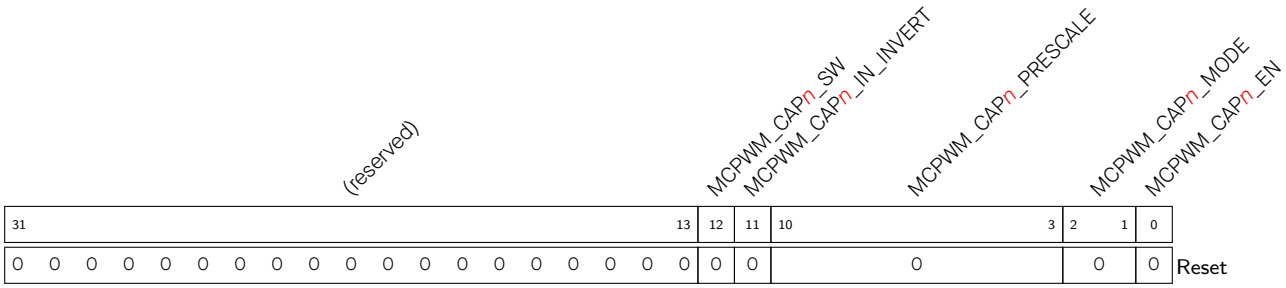
(WT)

Register 59.24. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM_CAP_TIMER_PHASE 配置捕获定时器同步操作的相位值。(R/W)

Register 59.25. MCPWM_CAP_CH*n*_CFG_REG (*n*: 0-2) (0x00F0+0x4**n*)



MCPWM_CAP*n*_EN 配置是否使能信道 *n* 上的捕获事件。

0：不使能

1：使能

(R/W)

MCPWM_CAP*n*_MODE 配置预分频后信道 *n* 上的捕获沿。

0：无

bit0 为 1：使能下降沿捕获。

bit1 为 1：使能上升沿捕获。

(R/W)

MCPWM_CAP*n*_PRESCALE 配置 CAP*n* 上升沿的预分频值。预分频值 = PWM_CAP*n*_PRESCALE + 1。(R/W)

MCPWM_CAP*n*_IN_INVERT 配置是否在预分频前反相来自 GPIO 矩阵的 CAP*n*。

0：无效

1：在预分频前反相来自 GPIO 矩阵的 CAP*n*

(R/W)

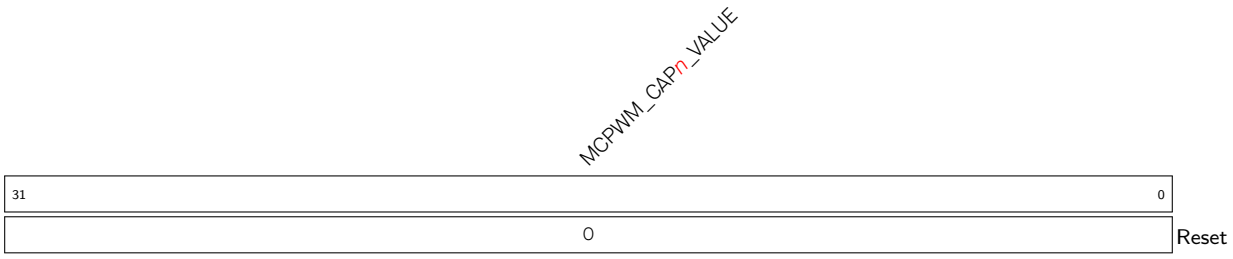
MCPWM_CAP*n*_SW 配置是否触发信道 0 上的软件强制捕获事件。

0：不触发

1：触发

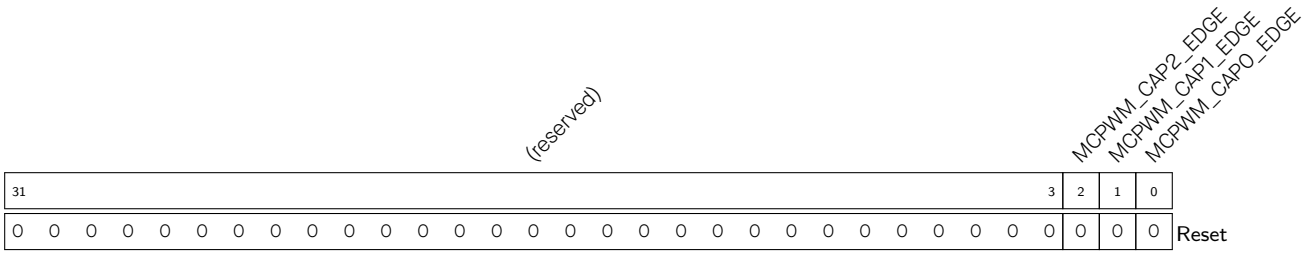
(WT)

Register 59.26. MCPWM_CAP_CH*n*_REG (*n*: 0-2) (0x00FC+0x4**n*)



MCPWM_CAP*n*_VALUE 表示信道 *n* 上一次捕获的值。(RO)

Register 59.27. MCPWM_CAP_STATUS_REG (0x0108)



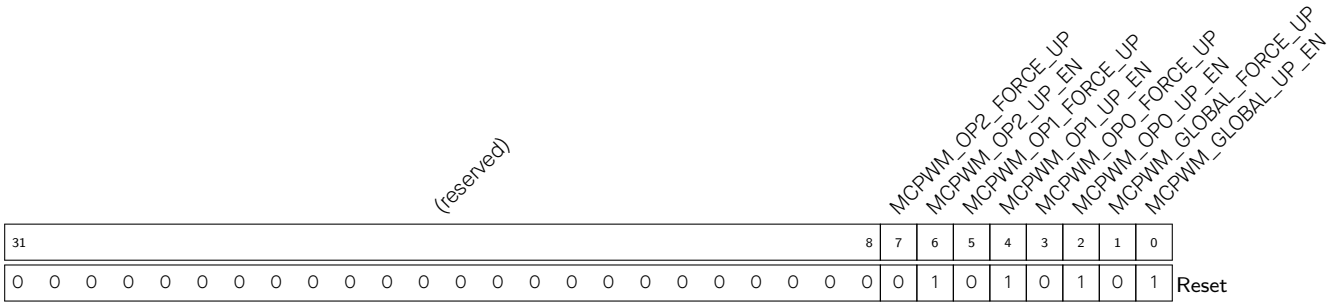
MCPWM_CAP*n*_EDGE 信道*n* 上一次捕获触发事件的边沿。

0：上升沿

1：下降沿

(RO)

Register 59.28. MCPWM_UPDATE_CFG_REG (0x010C)



MCPWM_GLOBAL_UP_EN 配置是否更新 MCPWM 模块的所有有效寄存器。

0：无效

1：更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_GLOBAL_FORCE_UP 配置是否强制更新 MCPWM 模块的所有有效寄存器。

0：无效

1：强制更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_OP0_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 0 中的有效寄存器。

0：无效

1：更新 PWM 操作器 0 中的有效寄存器

(R/W)

MCPWM_OP0_FORCE_UP 配置是否强制更新 PWM 操作器 0 中的有效寄存器。

0：无效

1：强制更新

(R/W)

MCPWM_OP1_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 1 中的有效寄存器。

0：无效

1：更新 PWM 操作器 1 中的有效寄存器

(R/W)

MCPWM_OP1_FORCE_UP 配置是否强制更新 PWM 操作器 1 中的有效寄存器。

0：无效

1：强制更新

(R/W)

MCPWM_OP2_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 2 中的有效寄存器。

0：无效

1：更新 PWM 操作器 2 中的有效寄存器

(R/W)

见下页...

Register 59.28. MCPWM_UPDATE_CFG_REG (0x010C)

... 接上页

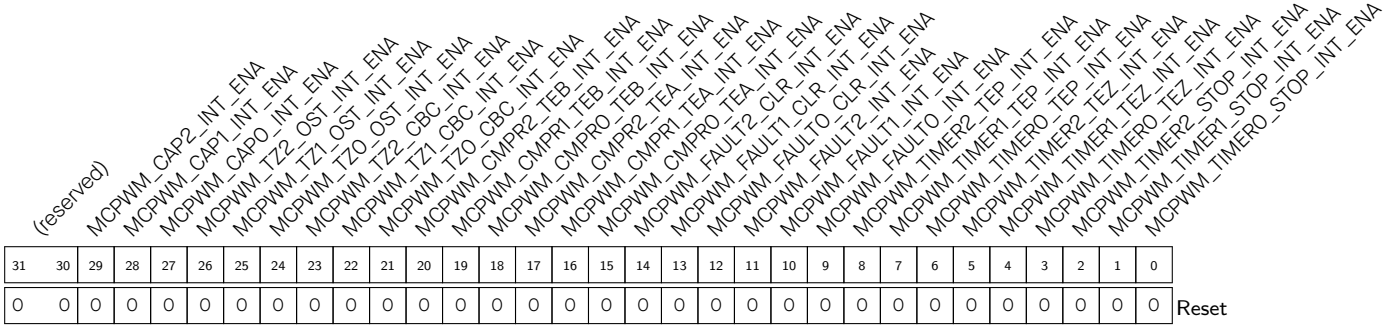
MCPWM_OP2_FORCE_UP 配置是否强制更新 PWM 操作器 2 中的有效寄存器。

0：无效

1：强制更新

(R/W)

Register 59.29. MCPWM_INT_ENA_REG (0x0110)



31 30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_ENA 写 1 使能 MCPWM_TIMER0_STOP_INT。(R/W)

MCPWM_TIMER1_STOP_INT_ENA 写 1 使能 MCPWM_TIMER1_STOP_INT。(R/W)

MCPWM_TIMER2_STOP_INT_ENA 写 1 使能 MCPWM_TIMER2_STOP_INT。(R/W)

MCPWM_TIMER0_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER0_TEZ_INT。(R/W)

MCPWM_TIMER1_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER1_TEZ_INT。(R/W)

MCPWM_TIMER2_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER2_TEZ_INT。(R/W)

MCPWM_TIMER0_TEP_INT_ENA 写 1 使能 MCPWM_TIMER0_TEP_INT。(R/W)

MCPWM_TIMER1_TEP_INT_ENA 写 1 使能 MCPWM_TIMER1_TEP_INT。(R/W)

MCPWM_TIMER2_TEP_INT_ENA 写 1 使能 MCPWM_TIMER2_TEP_INT。(R/W)

MCPWM_FAULT0_INT_ENA 写 1 使能 MCPWM_FAULT0_INT。(R/W)

MCPWM_FAULT1_INT_ENA 写 1 使能 MCPWM_FAULT1_INT。(R/W)

MCPWM_FAULT2_INT_ENA 写 1 使能 MCPWM_FAULT2_INT。(R/W)

MCPWM_FAULT0_CLR_INT_ENA 写 1 使能 MCPWM_FAULT0_CLR_INT。(R/W)

MCPWM_FAULT1_CLR_INT_ENA 写 1 使能 MCPWM_FAULT1_CLR_INT。(R/W)

MCPWM_FAULT2_CLR_INT_ENA 写 1 使能 MCPWM_FAULT2_CLR_INT。(R/W)

MCPWM_CMPR0_TEA_INT_ENA 写 1 使能 MCPWM_CMPR0_TEA_INT。(R/W)

MCPWM_CMPR1_TEA_INT_ENA 写 1 使能 MCPWM_CMPR1_TEA_INT。(R/W)

MCPWM_CMPR2_TEA_INT_ENA 写 1 使能 MCPWM_CMPR2_TEA_INT。(R/W)

见下页...

Register 59.29. MCPWM_INT_ENA_REG (0x0110)

... 接上页

MCPWM_CMPR0_TEB_INT_ENA 写 1 使能 MCPWM_CMPR0_TEB_INT。(R/W)

MCPWM_CMPR1_TEB_INT_ENA 写 1 使能 MCPWM_CMPR1_TEB_INT。(R/W)

MCPWM_CMPR2_TEB_INT_ENA 写 1 使能 MCPWM_CMPR2_TEB_INT。(R/W)

MCPWM_TZ0_CBC_INT_ENA 写 1 使能 MCPWM_TZ0_CBC_INT。(R/W)

MCPWM_TZ1_CBC_INT_ENA 写 1 使能 MCPWM_TZ1_CBC_INT。(R/W)

MCPWM_TZ2_CBC_INT_ENA 写 1 使能 MCPWM_TZ2_CBC_INT。(R/W)

MCPWM_TZ0_OST_INT_ENA 写 1 使能 MCPWM_TZ0_OST_INT。(R/W)

MCPWM_TZ1_OST_INT_ENA 写 1 使能 MCPWM_TZ1_OST_INT。(R/W)

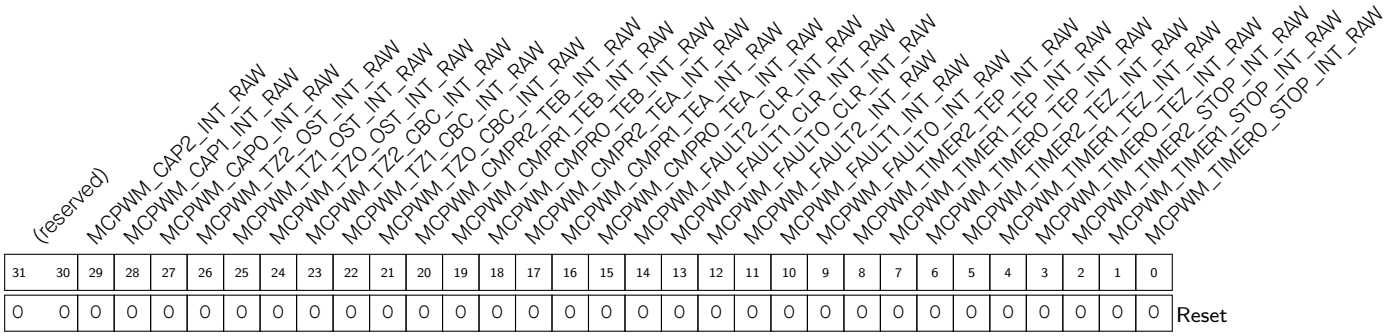
MCPWM_TZ2_OST_INT_ENA 写 1 使能 MCPWM_TZ2_OST_INT。(R/W)

MCPWM_CAP0_INT_ENA 写 1 使能 MCPWM_CAP0_INT。(R/W)

MCPWM_CAP1_INT_ENA 写 1 使能 MCPWM_CAP1_INT。(R/W)

MCPWM_CAP2_INT_ENA 写 1 使能 MCPWM_CAP2_INT。(R/W)

Register 59.30. MCPWM_INT_RAW_REG (0x0114)



MCPWM_TIMER0_STOP_INT_RAW MCPWM_TIMER0_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_STOP_INT_RAW MCPWM_TIMER1_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_STOP_INT_RAW MCPWM_TIMER2_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEZ_INT_RAW MCPWM_TIMER0_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEZ_INT_RAW MCPWM_TIMER1_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEZ_INT_RAW MCPWM_TIMER2_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEP_INT_RAW MCPWM_TIMER0_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEP_INT_RAW MCPWM_TIMER1_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEP_INT_RAW MCPWM_TIMER2_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_INT_RAW MCPWM_FAULT0_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_INT_RAW MCPWM_FAULT1_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_INT_RAW MCPWM_FAULT2_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_CLR_INT_RAW MCPWM_FAULT0_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_CLR_INT_RAW MCPWM_FAULT1_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_CLR_INT_RAW MCPWM_FAULT2_CLR_INT 的原始状态位。(R/WTC/SS)

见下页...

Register 59.30. MCPWM_INT_RAW_REG (0x0114)

... 接上页

MCPWM_CMPR0_TEA_INT_RAW MCPWM_CMPR0_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEA_INT_RAW MCPWM_CMPR1_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEA_INT_RAW MCPWM_CMPR2_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR0_TEB_INT_RAW MCPWM_CMPR0_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEB_INT_RAW MCPWM_CMPR1_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEB_INT_RAW MCPWM_CMPR2_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ0_CBC_INT_RAW MCPWM_TZ0_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_CBC_INT_RAW MCPWM_TZ1_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ2_CBC_INT_RAW MCPWM_TZ2_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ0_OST_INT_RAW MCPWM_TZ0_OST_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_OST_INT_RAW MCPWM_TZ1_OST_INT 的原始状态位。(R/WTC/SS)

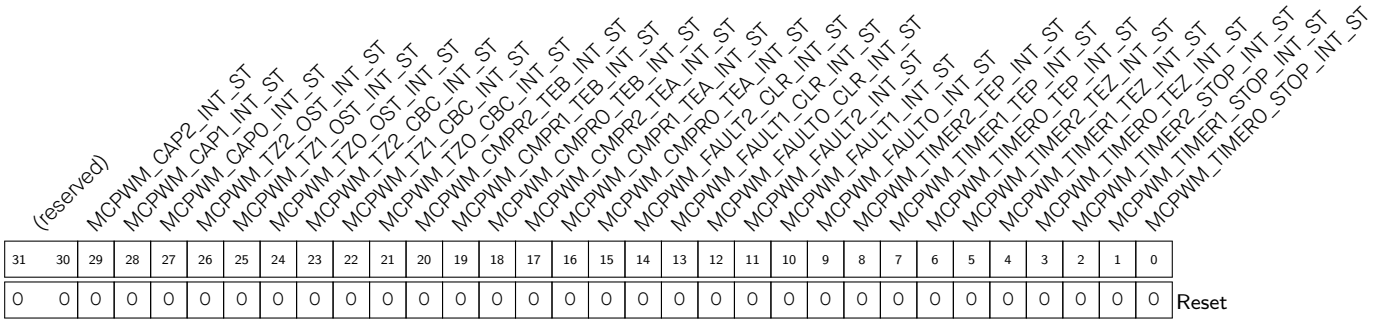
MCPWM_TZ2_OST_INT_RAW MCPWM_TZ2_OST_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP0_INT_RAW MCPWM_CAP0_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP1_INT_RAW MCPWM_CAP1_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP2_INT_RAW MCPWM_CAP2_INT 的原始状态位。(R/WTC/SS)

Register 59.31. MCPWM_INT_ST_REG (0x0118)



MCPWM_TIMER0_STOP_INT_ST MCPWM_TIMER0_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_STOP_INT_ST MCPWM_TIMER1_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_STOP_INT_ST MCPWM_TIMER2_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEZ_INT_ST MCPWM_TIMER0_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEZ_INT_ST MCPWM_TIMER1_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEZ_INT_ST MCPWM_TIMER2_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEP_INT_ST MCPWM_TIMER0_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEP_INT_ST MCPWM_TIMER1_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEP_INT_ST MCPWM_TIMER2_TEP_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_INT_ST MCPWM_FAULT0_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_INT_ST MCPWM_FAULT1_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_INT_ST MCPWM_FAULT2_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_CLR_INT_ST MCPWM_FAULT0_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_CLR_INT_ST MCPWM_FAULT1_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_CLR_INT_ST MCPWM_FAULT2_CLR_INT 的屏蔽状态位。(RO)

MCPWM_CMPR0_TEA_INT_ST MCPWM_CMPR0_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEA_INT_ST MCPWM_CMPR1_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEA_INT_ST MCPWM_CMPR2_TEA_INT 的屏蔽状态位。(RO)

见下页...

Register 59.31. MCPWM_INT_ST_REG (0x0118)

... 接上页

MCPWM_CMPR0_TEB_INT_ST MCPWM_CMPR0_TEB_INT 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEB_INT_ST MCPWM_CMPR1_TEB_INT 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEB_INT_ST MCPWM_CMPR2_TEB_INT 的屏蔽状态位。(RO)

MCPWM_TZ0_CBC_INT_ST MCPWM_TZ0_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ1_CBC_INT_ST MCPWM_TZ1_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ2_CBC_INT_ST MCPWM_TZ2_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ0_OST_INT_ST MCPWM_TZ0_OST_INT 的屏蔽状态位。(RO)

MCPWM_TZ1_OST_INT_ST MCPWM_TZ1_OST_INT 的屏蔽状态位。(RO)

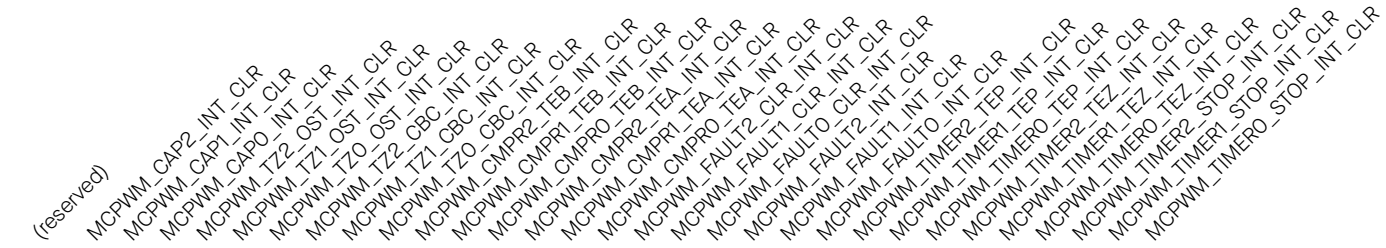
MCPWM_TZ2_OST_INT_ST MCPWM_TZ2_OST_INT 的屏蔽状态位。(RO)

MCPWM_CAP0_INT_ST MCPWM_CAP0_INT 的屏蔽状态位。(RO)

MCPWM_CAP1_INT_ST MCPWM_CAP1_INT 的屏蔽状态位。(RO)

MCPWM_CAP2_INT_ST MCPWM_CAP2_INT 的屏蔽状态位。(RO)

Register 59.32. MCPWM_INT_CLR_REG (0x011C)



MCPWM_TIMER0_STOP_INT_CLR 写 1 清除 MCPWM_TIMER0_STOP_INT。(WT)

MCPWM_TIMER1_STOP_INT_CLR 写 1 清除 MCPWM_TIMER1_STOP_INT。(WT)

MCPWM_TIMER2_STOP_INT_CLR 写 1 清除 MCPWM_TIMER2_STOP_INT。(WT)

MCPWM_TIMER0_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER0_TEZ_INT。(WT)

MCPWM_TIMER1_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER1_TEZ_INT。(WT)

MCPWM_TIMER2_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER2_TEZ_INT。(WT)

MCPWM_TIMER0_TEP_INT_CLR 写 1 清除 MCPWM_TIMER0_TEP_INT。(WT)

MCPWM_TIMER1_TEP_INT_CLR 写 1 清除 MCPWM_TIMER1_TEP_INT。(WT)

MCPWM_TIMER2_TEP_INT_CLR 写 1 清除 MCPWM_TIMER2_TEP_INT。(WT)

MCPWM_FAULT0_INT_CLR 写 1 清除 MCPWM_FAULT0_INT。(WT)

MCPWM_FAULT1_INT_CLR 写 1 清除 MCPWM_FAULT1_INT。(WT)

MCPWM_FAULT2_INT_CLR 写 1 清除 MCPWM_FAULT2_INT。(WT)

MCPWM_FAULT0_CLR_INT_CLR 写 1 清除 MCPWM_FAULT0_CLR_INT。(WT)

MCPWM_FAULT1_CLR_INT_CLR 写 1 清除 MCPWM_FAULT1_CLR_INT。(WT)

MCPWM_FAULT2_CLR_INT_CLR 写 1 清除 MCPWM_FAULT2_CLR_INT。(WT)

见下页...

Register 59.32. MCPWM_INT_CLR_REG (0x011C)

... 接上页

MCPWM_CMPR0_TEA_INT_CLR 写 1 清除 MCPWM_CMPR0_TEA_INT。(WT)

MCPWM_CMPR1_TEA_INT_CLR 写 1 清除 MCPWM_CMPR1_TEA_INT。(WT)

MCPWM_CMPR2_TEA_INT_CLR 写 1 清除 MCPWM_CMPR2_TEA_INT。(WT)

MCPWM_CMPR0_TEB_INT_CLR 写 1 清除 MCPWM_CMPR0_TEB_INT。(WT)

MCPWM_CMPR1_TEB_INT_CLR 写 1 清除 MCPWM_CMPR1_TEB_INT。(WT)

MCPWM_CMPR2_TEB_INT_CLR 写 1 清除 MCPWM_CMPR2_TEB_INT。(WT)

MCPWM_TZ0_CBC_INT_CLR 写 1 清除 MCPWM_TZ0_CBC_INT。(WT)

MCPWM_TZ1_CBC_INT_CLR 写 1 清除 MCPWM_TZ1_CBC_INT。(WT)

MCPWM_TZ2_CBC_INT_CLR 写 1 清除 MCPWM_TZ2_CBC_INT。(WT)

MCPWM_TZ0_OST_INT_CLR 写 1 清除 MCPWM_TZ0_OST_INT。(WT)

MCPWM_TZ1_OST_INT_CLR 写 1 清除 MCPWM_TZ1_OST_INT。(WT)

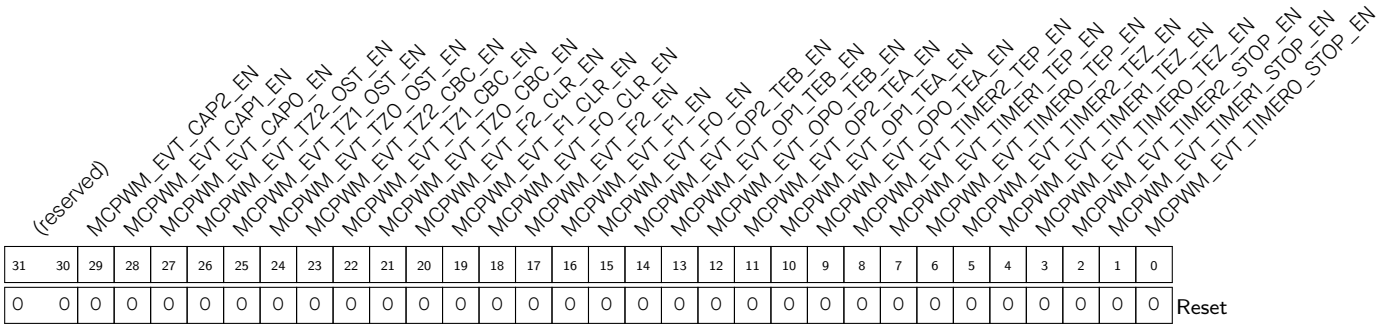
MCPWM_TZ2_OST_INT_CLR 写 1 清除 MCPWM_TZ2_OST_INT。(WT)

MCPWM_CAP0_INT_CLR 写 1 清除 MCPWM_CAP0_INT。(WT)

MCPWM_CAP1_INT_CLR 写 1 清除 MCPWM_CAP1_INT。(WT)

MCPWM_CAP2_INT_CLR 写 1 清除 MCPWM_CAP2_INT。(WT)

Register 59.33. MCPWM_EVT_EN_REG (0x0120)



MCPWM_EVT_TIMER0_STOP_EN 配置是否使能当定时器 0 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_STOP_EN 配置是否使能当定时器 1 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER2_STOP_EN 配置是否使能当定时器 2 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER0_TEZ_EN 配置是否使能当定时器 0 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_TEZ_EN 配置是否使能当定时器 1 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER2_TEZ_EN 配置是否使能当定时器 2 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER0_TEP_EN 配置是否使能当定时器 0 等于周期时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_TEP_EN 配置是否使能当定时器 1 等于周期时的事件生成。

0：关闭

1：使能

(R/W)

见下页...

Register 59.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TIMER2_TEP_EN 配置是否使能当定时器 2 等于周期时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP0_TEA_EN 配置是否使能当生成器 0 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP1_TEA_EN 配置是否使能当生成器 1 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP2_TEA_EN 配置是否使能当生成器 2 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP0_TEB_EN 配置是否使能当生成器 0 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP1_TEB_EN 配置是否使能当生成器 1 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP2_TEB_EN 配置是否使能当生成器 2 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F0_EN 配置是否使能 FAULT0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F1_EN 配置是否使能 FAULT1 事件生成。

0：关闭
1：使能
(R/W)

见下页...

Register 59.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_F2_EN 配置是否使能 FAULT2 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F0_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F1_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F2_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ0_CBC_EN 配置是否使能 CBC0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ1_CBC_EN 配置是否使能 CBC1 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ2_CBC_EN 配置是否使能 CBC2 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ0_OST_EN 配置是否使能 OST0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ1_OST_EN 配置是否使能 OST1 事件生成。

0：关闭
1：使能
(R/W)

见下页...

Register 59.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TZ2_OST_EN 配置是否使能 OST2 事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_CAP0_EN 配置是否使能 capture0 事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_CAP1_EN 配置是否使能 capture1 事件生成。

0：关闭

1：使能

(R/W)

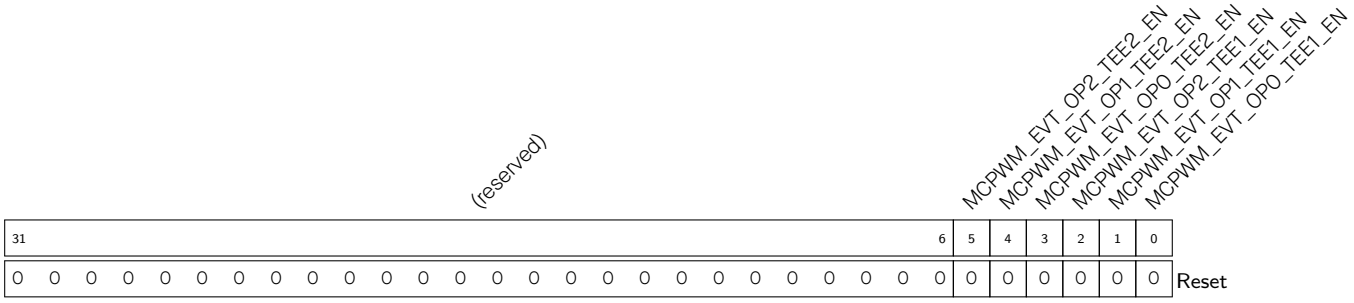
MCPWM_EVT_CAP2_EN 配置是否使能 capture2 事件生成。

0：关闭

1：使能

(R/W)

Register 59.34. MCPWM_EVT_EN2_REG (0x0128)



MCPWM_EVT_OP*n*_TEE1_EN 配置是否在 PWM 生成器计数器的值等于 MCPWM_OP*n*_TSTMP_E1_REG 寄存器的值时，生成 MCPWM_EVT_OP*n*_TEE1 事件。

0：不生成

1：生成

(R/W)

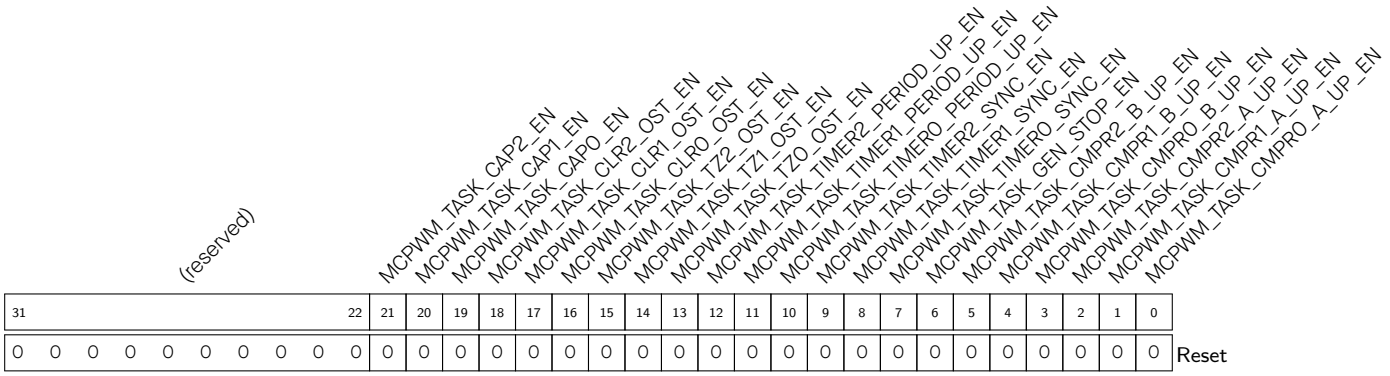
MCPWM_EVT_OP*n*_TEE2_EN 配置是否在 PWM 生成器计数器的值等于 MCPWM_OP*n*_TSTMP_E2_REG 寄存器的值时，生成 MCPWM_EVT_OP*n*_TEE2 事件。

0：不生成

1：生成

(R/W)

Register 59.35. MCPWM_TASK_EN_REG (0x0124)



MCPWM_TASK_CMPR0_A_UP_EN 配置是否接收 PWM 生成器 0 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR1_A_UP_EN 配置是否接收 PWM 生成器 1 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR2_A_UP_EN 配置是否接收 PWM 生成器 2 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR0_B_UP_EN 配置是否接收 PWM 生成器 0 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR1_B_UP_EN 配置是否接收 PWM 生成器 1 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR2_B_UP_EN 配置是否接收 PWM 生成器 2 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

见下页...

Register 59.35. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_GEN_STOP_EN 配置是否接收所有 PWM 生成器的停止任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER0_SYNC_EN 配置是否接收定时器 0 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER1_SYNC_EN 配置是否接收定时器 1 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER2_SYNC_EN 配置是否接收定时器 2 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER0_PERIOD_UP_EN 配置是否接收定时器 0 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER1_PERIOD_UP_EN 配置是否接收定时器 1 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER2_PERIOD_UP_EN 配置是否接收定时器 2 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TZ0_OST_EN 配置是否接收 OST0 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TZ1_OST_EN 配置是否接收 OST1 任务。

0：无效

1：接收

(R/W)

见下页...

Register 59.35. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_TZ2_OST_EN 配置是否接收 OST2 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR0_OST_EN 配置是否接收 OST0 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR1_OST_EN 配置是否接收 OST1 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR2_OST_EN 配置是否接收 OST2 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CAP0_EN 配置是否接收 capture0 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CAP1_EN 配置是否接收 capture1 任务。

0：无效

1：接收

(R/W)

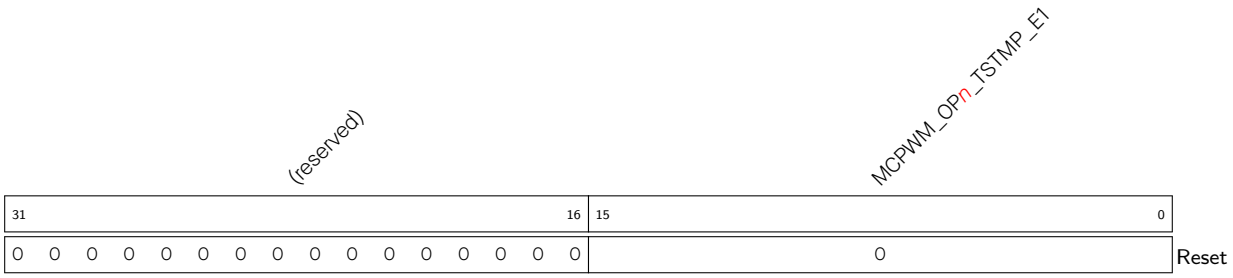
MCPWM_TASK_CAP2_EN 配置是否接收 capture2 任务。

0：无效

1：接收

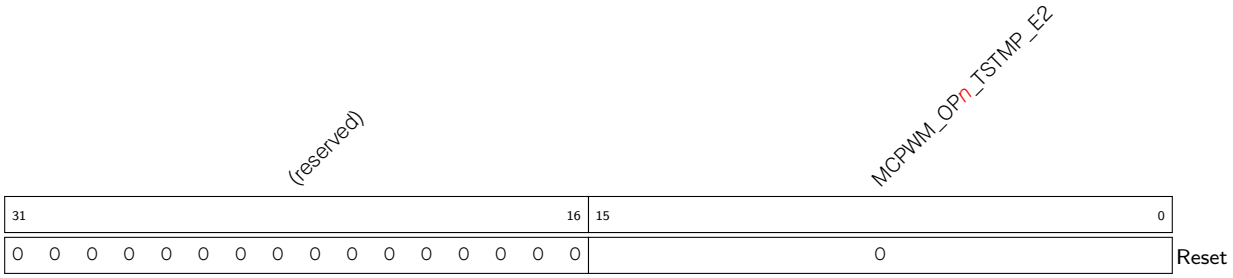
(R/W)

Register 59.36. MCPWM_OP*n*_TSTMP_E1_REG (*n*: 0-2) (0x012C+0x8**n*)



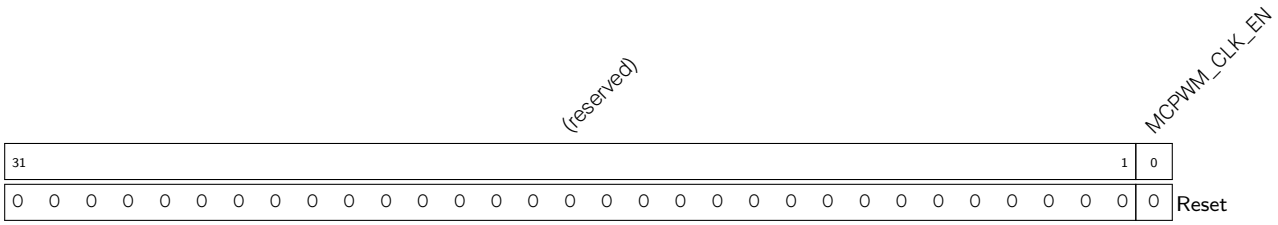
MCPWM_OP*n*_TSTMP_E1 配置生成器*n* 时间戳 E1 值。(R/W)

Register 59.37. MCPWM_OP*n*_TSTMP_E2_REG(*n*: 0-2) (0x0130+0x8**n*)



MCPWM_OP*n*_TSTMP_E2 配置生成器*n* 时间戳 E2 值。(R/W)

Register 59.38. MCPWM_CLK_REG (0x0144)



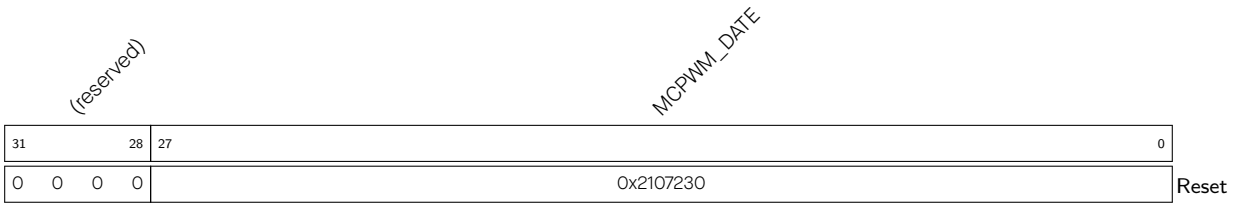
MCPWM_CLK_EN 配置是否强制使能时钟。

0：无效

1：强制使能时钟

(R/W)

Register 59.39. MCPWM_VERSION_REG (0x0148)



MCPWM_DATE 版本控制寄存器。(R/W)

第 60 章

红外遥控 (RMT)

60.1 概述

红外遥控 (RMT) 是一个红外发送和接收控制器，可通过软件编解码多种红外协议。RMT 模块可以实现将模块内置 RAM 中的脉冲编码转换为信号输出，或将模块的输入信号转换为脉冲编码存入 RAM 中。此外，RMT 模块也支持对输出信号进行载波调制或对输入信号进行解调和滤波处理。

RMT 共有八个通道，编码为 0 ~ 7，各通道可独立用于发送或接收信号：

- 0 ~ 3 通道专门用于发送信号；
- 4 ~ 7 通道专门用于接收信号。

每个发送通道和接收通道分别有一组功能相同的寄存器。另外，发送通道 3 和接收通道 7 对应的 RAM 支持 GDMA 访问，因此还有 GDMA 相关的控制和状态寄存器。为了方便叙述，以 *n* 表示各个发送通道，以 *m* 表示各个接收通道。

60.2 主要特性

RMT 控制器具有如下特性：

- 共配置八个通道：
  - 0 ~ 3 通道支持发送
  - 4 ~ 7 通道支持接收
  - 八个通道共享 384 x 32 位的 RAM
- 发射器支持以下模式：
  - 普通发送模式
  - 乒乓发送模式
  - 持续发送模式
  - 载波调制
  - 多通道同时发送
  - 发送通道 3 支持 GDMA 访问
- 接收器支持以下模式：
  - 普通接收模式
  - 乒乓接收模式

- 接收滤波
- 载波解调
- 接收通道 7 支持 GDMA 访问

60.3 功能描述

60.3.1 RMT 架构

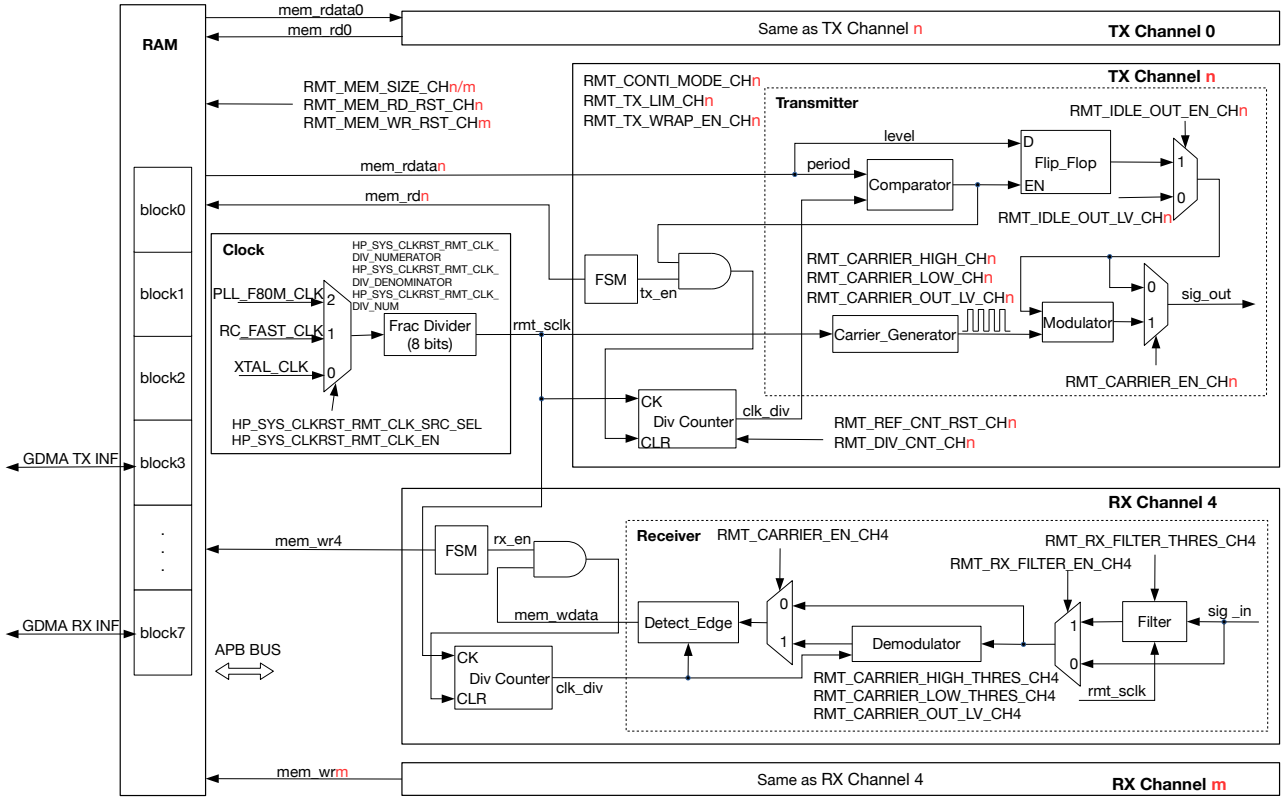


图 60.3-1. RMT 结构框图

如图 60.3-1 所示，每个发送通道内部各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个发射器 (Transmitter)

每个接收通道内部也各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个接收器 (Receiver)

八个通道共享一块 384 x 32 位的 RAM。

60.3.2 RAM

60.3.2.1 RAM 结构

RAM 中脉冲编码结构如图 60.3-2 所示。每个脉冲编码为 16 位，由 level 与 period 两部分组成。其中 level 表示输入或输出信号的逻辑电平值（0 或 1），period 表示该电平信号持续的时钟（见图 60.3-1 中的 clk_div）周期数。

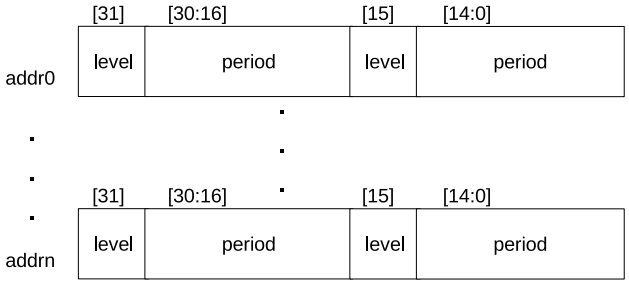


图 60.3-2. RAM 中脉冲编码结构

period 的最小值为 0，period 为 0 是一次传输的结束标志。对于非 0 的 period，即非结束标志的 period，其值需要满足与 APB 时钟和 RMT 时钟相关的限制条件，具体请参考以下公式：

$$5 \times T_{apb_clk} + 6 \times T_{rmt_sclk} < period \times T_{clk_div} \tag{60.1}$$

说明：

RMT 时钟与 APB 时钟需满足以下关系：

$$1.5 \times T_{apb_clk} < 9 \times T_{rmt_sclk} \tag{60.2}$$

RMT 可捕获的脉冲宽度，即 $period \times T_{clk_div}$ 受上述公式以及 rmt_sclk 时钟频率的限制。具体如下：

- 最小脉冲长度应大于 $(5 \times T_{apb_clk} + 6 \times T_{rmt_sclk})$；
- 最大脉冲长度应小于或等于（最大 period × 最大 T_{clk_div}），即 $((2^{15} - 1) \times$ 最大 $T_{rmt_sclk} \times 256)$。

更多有关 rmt_sclk 时钟频率，或 APB_CLK 时钟频率，请参考小节 60.3.3，或章节 11 复位和时钟。

60.3.2.2 RAM 使用说明

RAM 按照 48 x 32 位分成八个 block。默认情况下每个通道只能使用一个 block（固定为通道 0 使用 block 0，通道 1 使用 block 1，以此类推）。

当发送通道 *n* 或接收通道 *m* 单次传输的脉冲编码数大于一个 block 时，可以：

- 置位 RMT_MEM_TX/RX_WRAP_EN_CH*n*/*m* 使能乒乓操作；
- 或通过配置 RMT_MEM_SIZE_CH*n*/*m* 寄存器，允许该通道占用多个 block。

当设置 RMT_MEM_SIZE_CH*n*/*m* > 1 时，通道 *n*/*m* 将占用 block (*n*/*m*) ~ block (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* -1) 的存储空间。通道 *n*/*m* + 1 ~ *n*/*m* + RMT_MEM_SIZE_CH*n*/*m* - 1 因为对应的 RAM block 被占用而无法使用。例如，如果通道 0 配置使用了 block 0 和 block 1，则通道 1 无法使用，通道 2 和通道 3 不受影响。

注意，每个通道使用 RAM 的空间是根据地址从低到高进行映射的，因此通道 0 可以通过配置 RMT_MEM_SIZE_CH0 寄存器来使用通道 1、2、3、...、7 的 RAM 空间，但是通道 7 不能使用通道 0、1、2、...、6 的 RAM 空间。因此 RMT_MEM_SIZE_CH*n* 不应超过 (8 - *n*)，RMT_MEM_SIZE_CH*m* 不应超过 (8 - *m*)。

RAM 可被 APB 总线及通道的发射器或接收器访问，为了防止接收通道访问 RAM 和 APB 访问时发生冲突，例如 APB 读 RAM 时，通道向 RAM 发起写操作，用户可以通过配置 RMT_MEM_OWNER_CH*m* 来决定当前 RAM 的使用权。当接收通道发生越权访问时会产生 RMT_MEM_OWNER_ERR_CH*m* 标志信号。

60.3.2.3 RAM 访问方式

APB 总线访问 RAM 有 FIFO 和 NONFIFO（直接地址）两种模式：

- RMT_APB_FIFO_MASK 置 0 时，选择 FIFO 模式；
- RMT_APB_FIFO_MASK 置 1 时，选择 NONFIFO 模式。

另外，通道 3 和通道 7 还分别支持 GDMA 访问方式。

FIFO 模式

在 FIFO 模式下，APB 通过固定地址 RMT_CH*n*/*m*DATA_REG 向 RAM 写数据或从 RAM 读数据。

NONFIFO 模式

在 NONFIFO 模式下，APB 向连续地址段写入数据或从连续地址段读出数据：

- 发送通道 *n* 对应的写地址段的首地址是：RMT 基地址 + 0x800 + *n*x48，第 2 个数据的访问地址是 RMT 基地址 + 0x800 + *n*x48 + 0x4，以此类推，后面的地址依次加上 0x4。
- 接收通道 *m* 对应的读地址段的首地址是：RMT 基地址 + 0x8C0 + *m*x48，第 2 个数据的访问地址是 RMT 基地址 + 0x8C0 + *m*x48 + 0x4，以此类推，后面的地址依次加上 0x4。

GDMA 模式

不同于其它几个发送通道，通道 3 还支持 GDMA 访问方式。如果将 RMT_DMA_ACCESS_EN_CH3 置 1，则通道 3 对应的 RAM 仅支持从 GDMA 取数。APB 对通道 3 的 FIFO 方式访问将被忽略，同时也不允许 APB 通过 NONFIFO 方式访问通道 3 的 RAM 区域，**否则会出现不可预计的后果**。

为保证传输数据的正确性，需要先启动 GDMA，在 GDMA 通道已经接收到数据后，再启动 RMT 发送。普通发送模式下，GDMA 写满通道 3 的 RAM 后会触发 RMT_APB_MEM_WR_ERR_CH3。使能 RMT_MEM_TX_WRAP_EN_CH3 后，不需要软件进行乒乓操作就可以连续发送大于一个 block 的数据，但要保证通道准备发送数据前 GDMA 已经将待发送数据准备好，否则可能会发出与期望不符的数据。

不同于其它几个接收通道，通道 7 还支持 GDMA 访问方式。如果将 RMT_DMA_ACCESS_EN_CH7 置 1，通道 7 对应的 RAM 支持向 GDMA 发送数据，同时也支持 APB 的 NONFIFO 方式访问。

普通接收模式下，GDMA 读到通道 7 RAM 大小的数据后就会触发 RMT_APB_MEM_RD_ERR_CH7，之后收到的数会被丢弃。使能 RMT_MEM_TX_WRAP_EN_CH7 后，不需要软件进行乒乓操作就可以连续接收大于一个 block 的数据。如果通道 7 的 RAM 接收满后 GDMA 还没有接收，新收到的数据会替换上一个数据。

> **说明：**
> 当通道 7 接收到结束标志时，会产生 GDMA 的 in_suc_eof 中断。如果是 period[14:0] 为 0，则会向 GDMA 写两个字节；如果是 period[30:16] 为 0，则会向 GDMA 写四个字节。

60.3.3 时钟

用户可以通过配置 HP_SYS_CLKRST_RMT_CLK_SRC_SEL 选择 RMT 的时钟源：PLL_F80M_CLK、RC_FAST_CLK 或 XTAL_CLK，配置 HP_SYS_CLKRST_RMT_CLK_EN 为高电平来打开 RMT 的时钟。选择后的时钟经过小数分频得到 RMT 的工作时钟（图 60.3-1 rmt_sclk），分频系数为 HP_SYS_CLKRST_RMT_CLK_DIV_NUM + 1 + HP_SYS_CLKRST_RMT_CLK_DIV_NUMERATOR 或 HP_SYS_CLKRST_RMT_CLK_DIV_DENOMINATOR。

更多信息，请参考章节 11 复位和时钟。RMT_DIV_CNT_CH*n*/*m* 用于配置 RMT 通道内部的时钟分频器的分频系数，除 0 表示 256 分频外，其他分频数等同于 RMT_DIV_CNT_CH*n*/*m* 的值。时钟分频器可以通过配置 RMT_REF_CNT_RST_CH*n*/*m* 进行复位。时钟分频器的分频时钟可供计数器使用，见图 60.3-1。

60.3.4 发射器

说明：
本小节以及后续小节所述的配置，均需要通过置位 RMT_CONF_UPDATE_CH*n*/*m* 的方式来进行更新，详情见第 60.3.6 小节。

60.3.4.1 普通发送模式

当 RMT_TX_START_CH*n* 置为 1 时，通道 *n* 的发射器开始从通道对应 RAM block 的起始地址，按照地址从低到高依次读取脉冲编码进行发送。当遇到结束标志（period 等于 0）时，发射器将结束发送返回空闲状态，并产生 RMT_CH*n*_TX_END_INT 中断。配置 RMT_TX_STOP_CH*n* 可以使发射器立刻停止发送并进入空闲状态。发射器空闲状态发送的电平由结束标志中的 level 段或者是 RMT_IDLE_OUT_LV_CH*n* 决定。用户可以配置 RMT_IDLE_OUT_EN_CH*n* 来选择这两种方式：

- 清除 RMT_IDLE_OUT_EN_CH*n*，发射器空闲状态发送的电平由结束标志中的 level 段决定；
- 置位 RMT_IDLE_OUT_EN_CH*n*，发射器空闲状态发送的电平由 RMT_IDLE_OUT_LV_CH*n* 决定。

60.3.4.2 乒乓发送模式

当发送的脉冲编码较多时，可通过置位 RMT_MEM_TX_WRAP_EN_CH*n* 使能通道 *n* 的乒乓模式。在乒乓操作模式下，发射器会循环从通道对应的 RAM 区域取出脉冲编码进行发送，直到遇到结束标识为止。例如，当 RMT_MEM_SIZE_CH*n* = 1 时，发射器将从 48 * *n* 地址开始发送，然后对应 RAM 的地址递增。发完 (48 * (*n* + 1) - 1) 地址的数据后，下次继续从 48 * *n* 地址开始递增发送数据，依此类推，遇到结束标识时停止发送。RMT_MEM_SIZE_CH*n* > 1 的情形下，乒乓操作同样适用。

每当发射器发送的脉冲编码数大于等于 RMT_TX_LIM_CH*n* 时，会产生 RMT_CH*n*_TX_THR_EVENT_INT 中断。在乒乓模式下，可以设置 RMT_TX_LIM_CH*n* 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 RMT_CH*n*_TX_THR_EVENT_INT 中断之后，可以更新已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

说明：
当 RAM 采用 GDMA 方式访问时，不需要额外操作就能支持多于一个 RAM 的脉冲编码发送。而采用 APB 方式访问时，需要软件进行乒乓操作。

60.3.4.3 发送加载波

此外，发射器还可以对输出信号进行载波调制，置位 RMT_CARRIER_EN_CH*n* 可以使能该功能。载波的波形可配置。一个载波周期中高电平持续时间为 (RMT_CARRIER_HIGH_CH*n* + 1) 个 rmt_sclk 时钟周期，低电平持续的时间为 (RMT_CARRIER_LOW_CH*n* + 1) 个 rmt_sclk 时钟周期。置位 RMT_CARRIER_OUT_LV_CH*n* 时在输出信号高电平上加载波信号，清零 RMT_CARRIER_OUT_LV_CH*n* 时在输出信号低电平上加载波信号。同时，在进行载波调制时，载波可以一直加载在输出信号上，也可以仅加载在有效的脉冲编码（RAM 中的数据）上。通过配置 RMT_CARRIER_EFF_EN_CH*n*，可以选择这两种模式。RMT_CARRIER_EFF_EN_CH*n* 设置为 0 时在所有信号上加载波，设置为 1 时在有效信号上加载波。

60.3.4.4 持续发送模式

置位 RMT_TX_CONTI_MODE_CH*n* 可以使能发射器的持续发送功能。置位后，发射器会循环发送 RAM 中的脉冲编码。

持续发送模式下，

- 如果遇到结束标志，则重新从该通道 RAM 中的第一个数据开始发送；
- 如果没有结束标志，会在发送到最后一个数据处回卷，重新开始发送第一个数据。

置位 RMT_TX_LOOP_CNT_EN_CH*n* 后，发射器每遇到一次结束标志，循环发送的次数会加 1。当该次数达到 RMT_TX _LOOP_NUM_CH*n* 设定的值时，会产生 RMT_CH*n*_TX_LOOP_INT 中断。如果置位 RMT_LOOP_STOP_EN_CH*n*，则发送会在产生 RMT_CH*n*_TX_LOOP_INT 中断后立即停止，否则会继续发送。持续发送模式下，如果遇到的结束标志类型是 period[14:0] 为 0，那么这个结束标志前一个数据的 period 需要满足：

$$10 \times T_{apb_clk} + 19 \times T_{rmt_sclk} < period \times T_{clk_div} \tag{60.3}$$

而其它数据的 period，满足非 0 period 关系式 即可。

60.3.4.5 多通道同时发送

RMT 模块支持多通道同时发送，具体配置步骤如下所示：

- 首先配置 RMT_TX_SIM_CH*n* 用于选择同步发送的通道；
- 然后置位 RMT_TX_SIM_EN 使能发射器多个通道同步发送的功能；
- 最后将所选同步发送通道的 RMT_TX_START_CH*n* 置为 1。

当最后一个通道完成配置时，此时多个通道会同时启动发送。另外，RMT 模块还支持通道 0 ~ 2 的 RAM 采用 APB 访问和通道 3 的 RAM 采用 GDMA 方式访问下的多通道同时发送。

60.3.5 接收器

60.3.5.1 普通接收模式

RMT_RX_EN_CH*m* 置为 1 时接收器开始工作，置为 0 时会停止接收。接收器会从信号的第一个跳变沿开始计数，并检测信号电平及其持续的时钟周期数，将其按照脉冲编码的格式存入 RAM 中。当信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时，接收器结束接收过程，返回空闲状态，并产生 RMT_CH*m*_RX_END_INT 中断。RMT_IDLE_THRES_CH*m* 的值需要大于输入信号的高电平或低电平的最大时钟

周期，否则接收器会将该信号误判为空闲信号而进入空闲状态。当接收数据存满了接收通道设置的 RAM 空间时，会停止接收，并产生 RMT_CH*m*_ERR_INT 中断（由 RAM 满事件触发）。

60.3.5.2 乒乓接收模式

当接收的脉冲编码较多时，可通过置位 RMT_MEM_RX_WRAP_EN_CH*m* 使能通道 *m* 的乒乓模式。当 RAM 采用 GDMA 方式访问时，不需要额外操作就能支持多于一个 RAM 的脉冲编码接收。而采用 APB 方式访问时，需要软件进行乒乓操作。在乒乓操作模式下，接收器会将接收到的脉冲编码循环存入通道对应的 RAM 区域，当信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时，接收器结束接收过程，返回空闲状态，并产生 RMT_CH*m*_RX_END_INT 中断。例如，当 RMT_MEM_SIZE_CH*m* = 1 时，接收器将从 48 * *m* 地址开始接收，然后对应 RAM 的地址递增。收完 (48 * (*m* +1) - 1) 地址的数据后，下次继续从 48 * *m* 地址开始递增接收数据，以此类推，遇到信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时停止接收。RMT_MEM_SIZE_CH*m* > 1 的情形下，乒乓操作同样适用。

每当接收器接收的脉冲编码数大于或等于 RMT_CH*m*_RX_LIM_REG 时，会产生 RMT_CH*m*_RX_THR_EVENT_INT 中断。在乒乓模式下，可以设置 RMT_CH*m*_RX_LIM_REG 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 RMT_CH*m*_RX_THR_EVENT_INT 中断之后，可以回收已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

60.3.5.3 接收滤波

置位 RMT_RX_FILTER_EN_CH*m* 可使能通道 *m* 的接收器对输入信号进行滤波。滤波器的功能为连续采样输入信号，如果输入信号在连续 RMT_RX_FILTER _THRES_CH*m* 个 rmt_sclk 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，滤波器会滤除脉冲宽度小于 RMT_RX_FILTER_THRES_CH*m* 个 rmt_sclk 时钟周期的线路毛刺。

60.3.5.4 接收去载波

此外，接收器还可以对输入信号或滤波后的信号进行去载波调制，置位 RMT_CARRIER_EN_CH*m* 可以使能该功能。去载波可以对高电平调制或低电平调制的载波进行滤除：

- 清零 RMT_CARRIER_OUT_LV_CH*m*，配置滤除低电平载波；
- 置位 RMT_CARRIER_OUT_LV_CH*m*，配置滤除高电平载波。

配置 RMT_CARRIER_HIGH_THRES_CH*m* 和 RMT_CARRIER_LOW_THRES_CH*m* 可设置去载波的高电平阈值和低电平阈值。当信号的高电平持续时间小于 RMT_CARRIER_HIGH_THRES_CH*m* 个 clk_div 分频时钟周期，或者信号的低电平持续时间小于 RMT_CARRIER _LOW_THRES_CH*m* 个 clk_div 分频时钟周期，则信号会被认为是载波而被滤除。

60.3.6 配置参数更新

RMT 的配置寄存器均需要配置各自通道的 RMT_CONF_UPDATE_CH*n*/*m* 位来更新进入各自通道，更新方法是向 RMT_CONF_UPDATE_CH*n*/*m* 写入 1。发送通道和接收通道需要通过这种方法更新的配置参数见表 60.3-1。

表 60.3-1. 更新配置参数

配置寄存器	配置参数
发送通道	

见下页

表 60.3-1 – 接上页

配置寄存器	配置参数
RMT_CH*n*CONFO_REG	RMT_CARRIER_OUT_LV_CH*n*
	RMT_CARRIER_EN_CH*n*
	RMT_CARRIER_EFF_EN_CH*n*
	RMT_DIV_CNT_CH*n*
	RMT_TX_STOP_CH*n*
	RMT_IDLE_OUT_EN_CH*n*
	RMT_IDLE_OUT_LV_CH*n*
	RMT_TX_CONTI_MODE_CH*n*
RMT_CH*n*CARRIER_DUTY_REG	RMT_CARRIER_HIGH_CH*n*
	RMT_CARRIER_LOW_CH*n*
RMT_CH*n*_TX_LIM_REG	RMT_TX_LOOP_CNT_EN_CH*n*
	RMT_TX_LOOP_NUM_CH*n*
	RMT_TX_LIM_CH*n*
RMT_TX_SIM_REG	RMT_TX_SIM_EN
接收通道	
RMT_CH*m*CONFO_REG	RMT_CARRIER_OUT_LV_CH*m*
	RMT_CARRIER_EN_CH*m*
	RMT_IDLE_THRES_CH*m*
	RMT_DIV_CNT_CH*m*
RMT_CH*m*CONF1_REG	RMT_RX_FILTER_THRES_CH*m*
	RMT_RX_EN_CH*m*
RMT_CH*m*_RX_CARRIER_RM_REG	RMT_CARRIER_HIGH_THRES_CH*m*
	RMT_CARRIER_LOW_THRES_CH*m*
RMT_CH*m*_RX_LIM_REG	RMT_RX_LIM_CH*m*
RMT_REF_CNT_RST_REG	RMT_REF_CNT_RST_CH*m*

60.4 中断

ESP32-P4 的 RMT 控制器可以生成 RMT_INTR 中断信号并发送给 中断矩阵，该中断信号由 RMT 的内部中断源生成。

以下中断源生成 RMT_INTR 中断信号：

- RMT_CH*n*_ERR_INT：当通道 *n* 发生读写数据不正确，或内存空满错误时，即触发此中断。
- RMT_CH*m*_ERR_INT：当通道 *m* 发生读写数据不正确，或内存空满错误时，即触发此中断。
- RMT_CH*n*_TX_THR_EVENT_INT：发射器每发送 RMT_CH*n*_TX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH*m*_RX_THR_EVENT_INT：接收器每接收 RMT_CH*m*_RX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH*n*_TX_END_INT：当发射器停止发送信号时，即触发此中断。
- RMT_CH*m*_RX_END_INT：当接收器停止接收信号时，即触发此中断。
- RMT_CH*n*_TX_LOOP_INT：发射器处于循环发送模式时，当循环次数达到 RMT_TX_LOOP_NUM_CH*n* 的值后，会产生此中断。

- RMT_CH3_DMA_ACCESS_FAIL_INT：当 GDMA 向通道 3 RAM 区域写入数据个数减去通道 3 实际发送个数超过通道 3 RAM 大小，且 GDMA 又写入数据，会产生此中断。
- RMT_CH7_DMA_ACCESS_FAIL_INT：当通道 7 RAM 接收数据个数减去 GDMA 取走的数据超过通道 7 RAM 大小，且通道 7 又接收到数据，会产生此中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 60.5 寄存器列表。

60.5 寄存器列表

本小节的所有地址均为相对于红外遥控基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 读/写寄存器			
RMT_CH0DATA_REG	通道 0 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0000	HRO
RMT_CH1DATA_REG	通道 1 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0004	HRO
RMT_CH2DATA_REG	通道 2 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0008	HRO
RMT_CH3DATA_REG	通道 3 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x000C	HRO
RMT_CH4DATA_REG	通道 4 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0010	HRO
RMT_CH5DATA_REG	通道 5 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0014	HRO
RMT_CH6DATA_REG	通道 6 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0018	HRO
RMT_CH7DATA_REG	通道 7 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x001C	HRO
配置寄存器			
RMT_CH0CONF0_REG	通道 0 的配置寄存器 0	0x0020	varies
RMT_CH1CONF0_REG	通道 1 的配置寄存器 0	0x0024	varies
RMT_CH2CONF0_REG	通道 2 的配置寄存器 0	0x0028	varies
RMT_CH3CONF0_REG	通道 3 的配置寄存器 0	0x002C	varies
RMT_CH4CONF0_REG	通道 4 的配置寄存器 0	0x0030	R/W
RMT_CH4CONF1_REG	通道 4 的配置寄存器 1	0x0034	varies
RMT_CH5CONF0_REG	通道 5 的配置寄存器 0	0x0038	R/W
RMT_CH5CONF1_REG	通道 5 的配置寄存器 1	0x003C	varies
RMT_CH6CONF0_REG	通道 6 的配置寄存器 0	0x0040	R/W
RMT_CH6CONF1_REG	通道 6 的配置寄存器 1	0x0044	varies
RMT_CH7CONF0_REG	通道 7 的配置寄存器 0	0x0048	R/W
RMT_CH7CONF1_REG	通道 7 的配置寄存器 1	0x004C	varies
RMT_CH4_RX_CARRIER_RM_REG	通道 4 的载波滤除寄存器	0x0090	R/W
RMT_CH5_RX_CARRIER_RM_REG	通道 5 的载波滤除寄存器	0x0094	R/W
RMT_CH6_RX_CARRIER_RM_REG	通道 6 的载波滤除寄存器	0x0098	R/W
RMT_CH7_RX_CARRIER_RM_REG	通道 7 的载波滤除寄存器	0x009C	R/W
RMT_SYS_CONF_REG	RMT APB 配置寄存器	0x00C0	R/W
RMT_REF_CNT_RST_REG	RMT 时钟分频器复位寄存器	0x00C8	WT
状态寄存器			

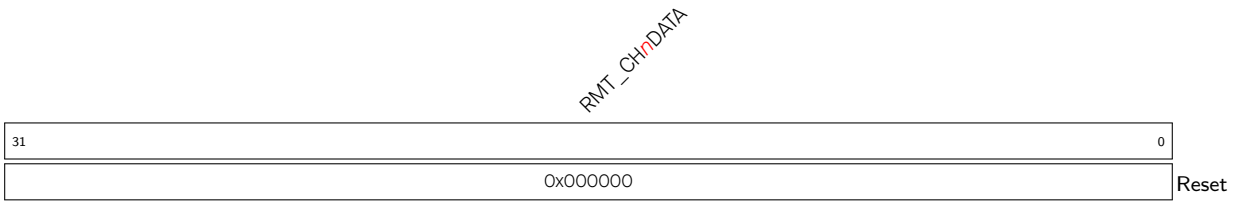
名称	描述	地址	访问
RMT_CH0STATUS_REG	通道 0 的状态寄存器	0x0050	RO
RMT_CH1STATUS_REG	通道 1 的状态寄存器	0x0054	RO
RMT_CH2STATUS_REG	通道 2 的状态寄存器	0x0058	RO
RMT_CH3STATUS_REG	通道 3 的状态寄存器	0x005C	RO
RMT_CH4STATUS_REG	通道 4 的状态寄存器	0x0060	RO
RMT_CH5STATUS_REG	通道 5 的状态寄存器	0x0064	RO
RMT_CH6STATUS_REG	通道 6 的状态寄存器	0x0068	RO
RMT_CH7STATUS_REG	通道 7 的状态寄存器	0x006C	RO
中断寄存器			
RMT_INT_RAW_REG	原始中断状态寄存器	0x0070	R/WTC/SS
RMT_INT_ST_REG	屏蔽中断状态寄存器	0x0074	RO
RMT_INT_ENA_REG	中断使能寄存器	0x0078	R/W
RMT_INT_CLR_REG	中断清零寄存器	0x007C	WT
载波占空比寄存器			
RMT_CH0CARRIER_DUTY_REG	通道 0 的占空比配置寄存器	0x0080	R/W
RMT_CH1CARRIER_DUTY_REG	通道 1 的占空比配置寄存器	0x0084	R/W
RMT_CH2CARRIER_DUTY_REG	通道 2 的占空比配置寄存器	0x0088	R/W
RMT_CH3CARRIER_DUTY_REG	通道 3 的占空比配置寄存器	0x008C	R/W
TX 事件配置寄存器			
RMT_CH0_TX_LIM_REG	通道 0 的 TX 事件配置寄存器	0x00A0	varies
RMT_CH1_TX_LIM_REG	通道 1 的 TX 事件配置寄存器	0x00A4	varies
RMT_CH2_TX_LIM_REG	通道 2 的 TX 事件配置寄存器	0x00A8	varies
RMT_CH3_TX_LIM_REG	通道 3 的 TX 事件配置寄存器	0x00AC	varies
RMT_TX_SIM_REG	RMT TX 同步发送寄存器	0x00C4	R/W
RX 事件配置寄存器			
RMT_CH4_RX_LIM_REG	通道 4 RX 事件配置寄存器	0x00B0	R/W
RMT_CH5_RX_LIM_REG	通道 5 RX 事件配置寄存器	0x00B4	R/W
RMT_CH6_RX_LIM_REG	通道 6 RX 事件配置寄存器	0x00B8	R/W
RMT_CH7_RX_LIM_REG	通道 7 RX 事件配置寄存器	0x00BC	R/W
版本寄存器			
RMT_DATE_REG	版本控制寄存器	0x00CC	R/W

60.6 寄存器

本小节的所有地址均为相对于红外遥控基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 60.1. RMT_CH*n*DATA_REG (*n*: 0-3) (0x0000+0x4**n*)



RMT_CH*n*DATA 通道 *n* 通过 APB FIFO 进行读写操作时用到的数据寄存器。(HRO)

Register 60.2. RMT_CH*m*DATA_REG (*m*: 4-7) (0x0000+0x4**m*)



RMT_CH*m*DATA 通道 *m* 通过 APB FIFO 进行读写操作时用到的数据寄存器。(HRO)

Register 60.3. RMT_CH*n*CONF0_REG (*n*: 0-3) (0x0020+0x4*n*)



RMT_TX_START_CH*n* 配置是否使能通道 *n* 开始发送数据。

0：无效

1：使能

(WT)

RMT_MEM_RD_RST_CH*n* 配置是否复位通道 *n* 通过发射器访问的 RAM 读地址。

0：无效

1：复位

(WT)

RMT_APB_MEM_RST_CH*n* 配置是否复位通道 *n* 通过 APB FIFO 访问的读/写 RAM 地址。

0：无效

1：复位

(WT)

RMT_TX_CONTI_MODE_CH*n* 配置是否使能通道 *n* 的持续发送模式。

0：无效

1：使能

在这种模式下，发射器从第一个数据开始发送，如果遇到结束标志，则再次发送第一个数据；如果没有遇到结束标志，发送完最后一个数据后，回卷到第一个数据再次继续发送。

(R/W)

RMT_MEM_TX_WRAP_EN_CH*n* 配置是否使能通道 *n* 的乒乓发送模式。

0：无效

1：使能

在这种模式下，如果待发送的数据长度大于该通道的 RAM Block 长度，则发射器将继续从第一个数据开始循环发送。

(R/W)

RMT_IDLE_OUT_LV_CH*n* 配置通道 *n* 在空闲模式下的输出信号电平。(R/W)

RMT_IDLE_OUT_EN_CH*n* 配置是否使能通道 *n* 在空闲状态下的输出模式。

0：无效

1：使能

(R/W)

见下页...

Register 60.3. RMT_CH*n*CONF0_REG (*n*: 0-3) (0x0020+0x4**n*)

接上页...

RMT_TX_STOP_CH*n* 配置通道 *n* 的发射器是否停止发送数据。

0：无效

1：停止

(R/W/SC)

RMT_DIV_CNT_CH*n* 配置通道 *n* 的时钟分频系数。

测量单位：rmt_sclk

(R/W)

RMT_MEM_SIZE_CH*n* 配置通道 *n* 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EFF_EN_CH*n* 配置是否仅在发送数据状态下对输出信号载波调制。

0：配置通道 *n* 对发送数据状态和空闲状态均加载载波

1：配置通道 *n* 仅在发送数据状态下对输出信号载波调制

仅在 RMT_CARRIER_EN_CH*n* 为 1 时有效。

(R/W)

RMT_CARRIER_EN_CH*n* 配置是否使能通道 *n* 对输出信号进行载波调制。

0：不使能

1：使能

(R/W)

RMT_CARRIER_OUT_LV_CH*n* 配置通道 *n* 的载波调制方式。

0：载波加载在低电平上

1：载波加载在高电平上

(R/W)

RMT_CONF_UPDATE_CH*n* 通道 *n* 的同步位。(WT)

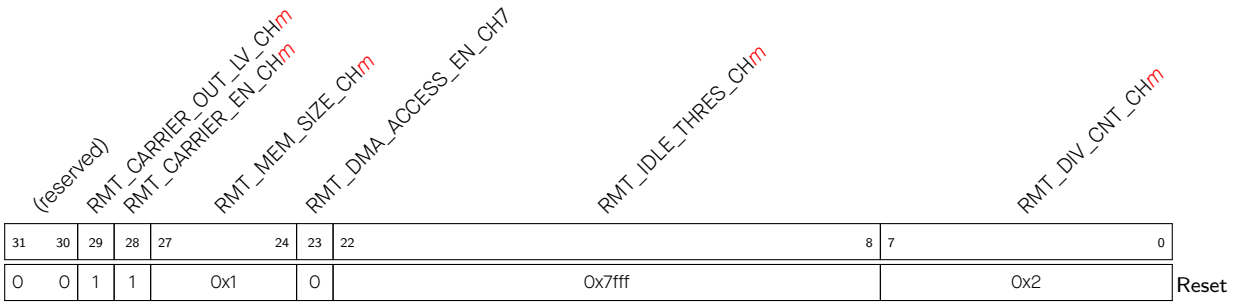
RMT_DMA_ACCESS_EN_CH3 配置是否启用通道 3 的 DMA 访问功能。通道 0 ~ 2 中该位保留。

0: 不启用

1: 启用

(R/W)

Register 60.4. RMT_CH*m*CONF0_REG (*m*: 4-7) (0x0030, 0x0038, 0x0040, 0x0048)



RMT_DIV_CNT_CH*m* 配置通道 *m* 的时钟分频时钟。

测量单位：rmt_sclk

(R/W)

RMT_IDLE_THRES_CH*m* 配置接收阈值。接收器长时间检测不到信号变化，且持续的时间大于 RMT_IDLE_THRES_CH*m* 的值，则接收器停止接收过程。

测量单位：clk_div

(R/W)

RMT_DMA_ACCESS_EN_CH7 配置是否启用通道 7 的 DMA 访问功能。通道 4 ~ 6 中该位保留。

0：不启用

1：启用

(R/W)

RMT_MEM_SIZE_CH*m* 配置通道 *m* 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EN_CH*m* 配置是否使能通道 *m* 对输入信号进行载波解调。

0：不使能

1：使能

(R/W)

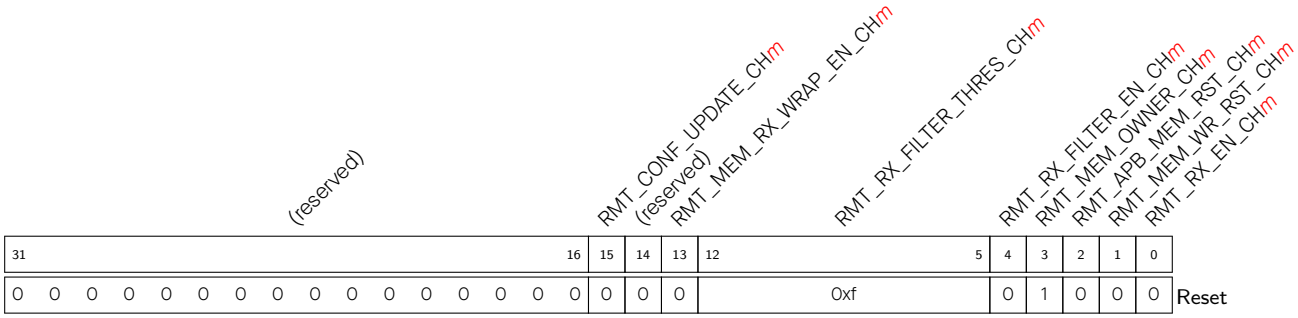
RMT_CARRIER_OUT_LV_CH*m* 配置通道 *m* 的载波解调方式。

0：在低电平上进行载波解调

1：在高电平上进行载波解调

(R/W)

Register 60.5. RMT_CH*m*CONF1_REG (*m*: 4-7) (0x0034, 0x003C, 0x0044, 0x004C)



RMT_RX_EN_CH*m* 配置是否使能通道 *m* 的接收器，使接收器开始接收数据。

0：不使能

1：使能

(R/W)

RMT_MEM_WR_RST_CH*m* 配置是否复位通道 *m* 通过接收器访问的 RAM 写地址。

0：无效

1：复位

(WT)

RMT_APB_MEM_RST_CH*m* 配置是否复位通道 *m* 通过 APB FIFO 访问的读/写 RAM 地址。

0：无效

1：复位

(WT)

RMT_MEM_OWNER_CH*m* 配置通道 *m* 的 RAM 使用权。

0：APB 总线有权使用该通道的 RAM

1：接收器有权使用该通道的 RAM

(R/W/SC)

RMT_RX_FILTER_EN_CH*m* 配置是否使能通道 *m* 的接收器滤波功能。

0：不使能

1：使能

(R/W)

RMT_RX_FILTER_THRES_CH*m* 配置是否在接收模式下，忽略宽度小于 RMT_RX_FILTER_THRES_CH*m* 个 rmt_sclk 周期的输入脉冲。

0：无效

1：忽略

(R/W)

RMT_MEM_RX_WRAP_EN_CH*m* 配置是否使能通道 *m* 的乒乓接收模式。

0：不使能

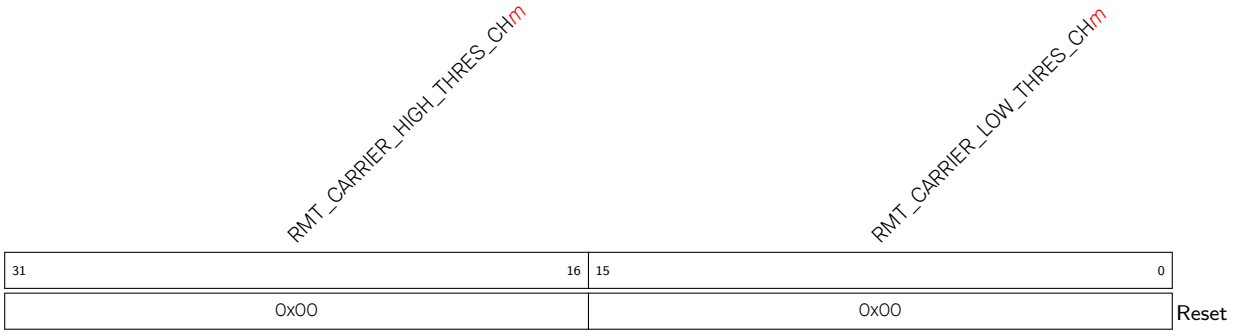
1：使能

在这种模式下，如果待接收的数据长度大于该通道的 RAM Block 长度，则接收器将继续把接收数据存入 RAM 的第一个地址，依次循环。

(R/W)

RMT_CONF_UPDATE_CH*m* 通道 *m* 的同步位。(WT)

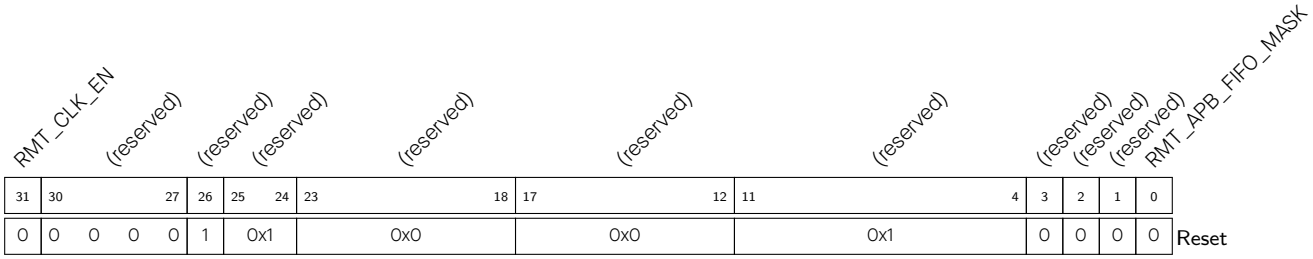
Register 60.6. RMT_CH*m*_RX_CARRIER_RM_REG (*m*: 4-7) (0x0090, 0x0094, 0x0098, 0x009C)



RMT_CARRIER_LOW_THRES_CH*m* 配置通道 *m* 载波调制模式下的低电平周期，该周期为 (RMT_CARRIER_LOW_THRES_CH*m* + 1)。(R/W)

RMT_CARRIER_HIGH_THRES_CH*m* 配置通道 *m* 载波调制模式下的高电平周期，该周期为 (RMT_CARRIER_HIGH_THRES_CH*m* + 1)。(R/W)

Register 60.7. RMT_SYS_CONF_REG (0x00C0)



RMT_APB_FIFO_MASK 配置 RAM 的访问模式。

0：通过 FIFO 访问 RAM

1：直接访问 RAM

(R/W)

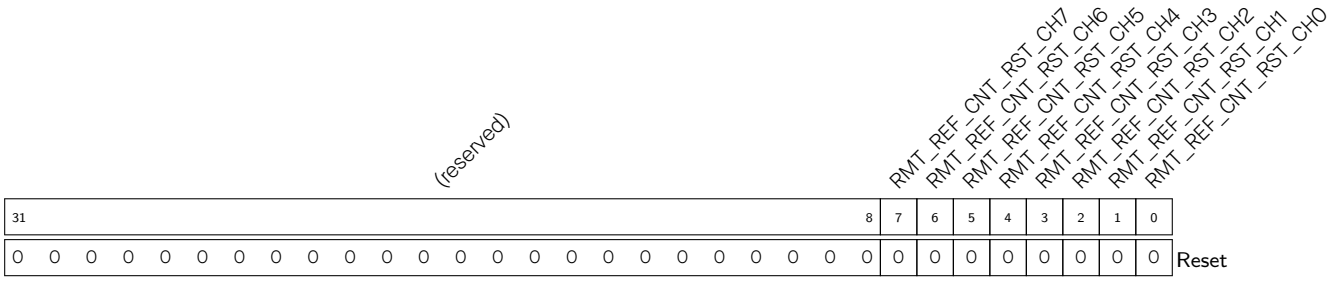
RMT_CLK_EN 配置是否使能 RMT 寄存器的时钟门控。

0：关闭寄存器的驱动时钟

1：打开寄存器的驱动时钟

(R/W)

Register 60.8. RMT_REF_CNT_RST_REG (0x00C8)



RMT_REF_CNT_RST_CH*n* (*n*: 0-3) 配置是否复位通道 *n* 的时钟分频器。

0：无效

1：复位

(WT)

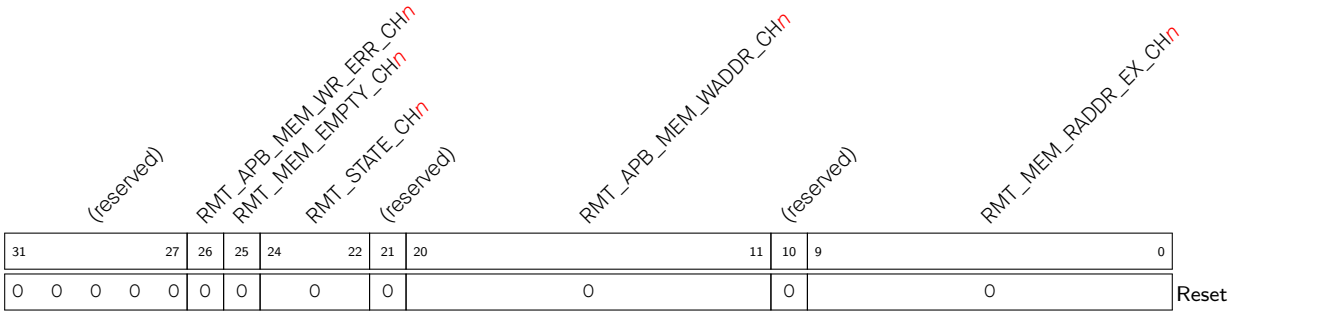
RMT_REF_CNT_RST_CH*m* (*m*: 4-7) 配置是否复位通道 *m* 的时钟分频器。

0：无效

1：复位

(WT)

Register 60.9. RMT_CH*n*STATUS_REG (*n*: 0-3) (0x0050+0x4**n*)



RMT_MEM_RADDR_EX_CH*n* 表示通道 *n* 发射器使用 RAM 时的地址偏移量。(RO)

RMT_APB_MEM_WADDR_CH*n* 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)

RMT_STATE_CH*n* 表示通道 *n* 的 FSM 状态。(RO)

RMT_MEM_EMPTY_CH*n* 表示发送的数据长度是否溢出 RAM Block，且乒乓模式未启用。

0：未溢出

1：溢出

(RO)

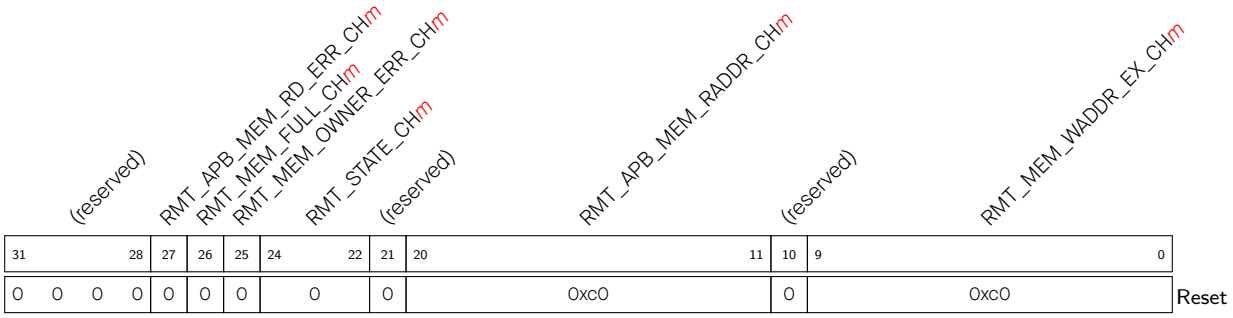
RMT_APB_MEM_WR_ERR_CH*n* 表示 RMT 使用 APB 总线进行写 RAM 操作时，偏移地址是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

Register 60.10. RMT_CH*m*STATUS_REG (*m*: 4-7) (0x0050+0x4**m*)



RMT_MEM_WADDR_EX_CH*m* 表示通道 *m* 接收器使用 RAM 时的地址偏移量。(RO)

RMT_APB_MEM_RADDR_CH*m* 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)

RMT_STATE_CH*m* 表示通道 *m* 的 FSM 状态。(RO)

RMT_MEM_OWNER_ERR_CH*m* 表示 RAM Block 使用权是否发生错误。

0：无错误

1：发生错误

(RO)

RMT_MEM_FULL_CH*m* 表示接收器接收的数据长度是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

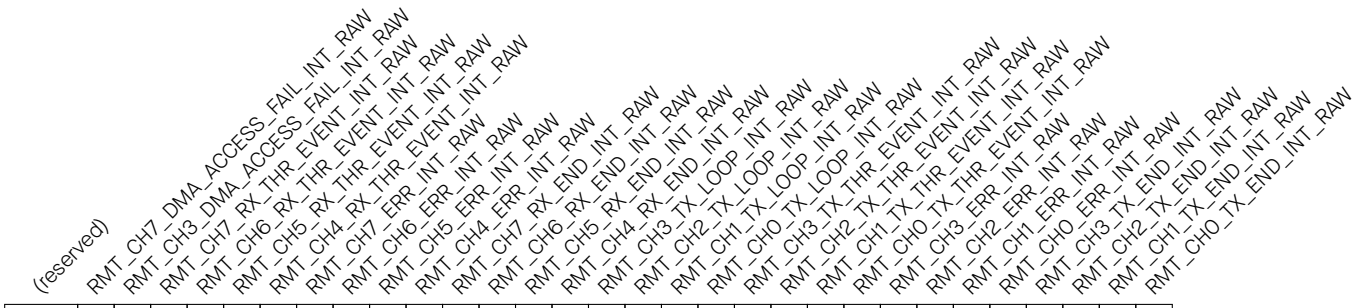
RMT_APB_MEM_RD_ERR_CH*m* 表示 RMT 使用 APB 总线执行 RAM 读操作时，偏移地址是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

Register 60.11. RMT_INT_RAW_REG (0x0070)



RMT_CH*n*_TX_END_INT_RAW (*n*: 0-3) RMT_CH*n*_TX_END_INT 的原始中断状态。(R/WTC/SS)

RMT_CH*n*_ERR_INT_RAW (*n*: 0-3) RMT_CH*n*_ERR_INT 的原始中断状态。(R/WTC/SS)

RMT_CH*n*_TX_THR_EVENT_INT_RAW (*n*: 0-3) RMT_CH*n*_TX_THR_EVENT_INT 的原始中断状态。(R/WTC/SS)

RMT_CH*n*_TX_LOOP_INT_RAW (*n*: 0-3) RMT_CH*n*_TX_LOOP_INT 的原始中断状态。(R/WTC/SS)

RMT_CH*m*_RX_END_INT_RAW (*m*: 4-7) RMT_CH*m*_RX_END_INT 的原始中断状态。(R/WTC/SS)

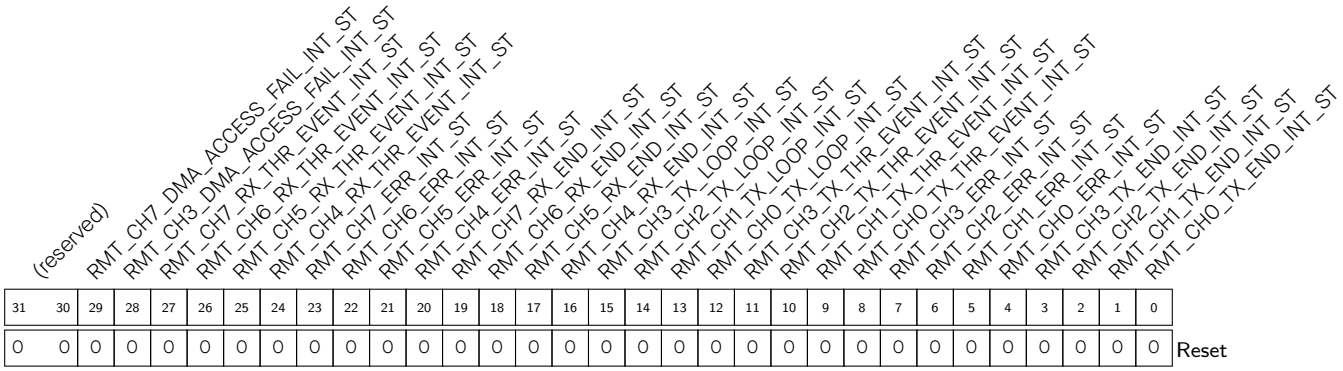
RMT_CH*m*_ERR_INT_RAW (*m*: 4-7) RMT_CH*m*_ERR_INT 的原始中断状态。(R/WTC/SS)

RMT_CH*m*_RX_THR_EVENT_INT_RAW (*m*: 4-7) RMT_CH*m*_RX_THR_EVENT_INT 的原始中断状态。(R/WTC/SS)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_RAW (*n*: 3) RMT_CH*n*_DMA_ACCESS_FAIL_INT 的原始中断状态。(R/WTC/SS)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_RAW (*m*: 7) RMT_CH*m*_DMA_ACCESS_FAIL_INT 的原始中断状态。(R/WTC/SS)

Register 60.12. RMT_INT_ST_REG (0x0074)



RMT_CH*n*_TX_END_INT_ST (*n*: 0-3) RMT_CH*n*_TX_END_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_ERR_INT_ST (*n*: 0-3) RMT_CH*n*_ERR_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST (*n*: 0-3) RMT_CH*n*_TX_THR_EVENT_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_TX_LOOP_INT_ST (*n*: 0-3) RMT_CH*n*_TX_LOOP_INT 的屏蔽中断状态。(RO)

RMT_CH*m*_RX_END_INT_ST (*m*: 4-7) RMT_CH*m*_RX_END_INT 的屏蔽中断状态。(RO)

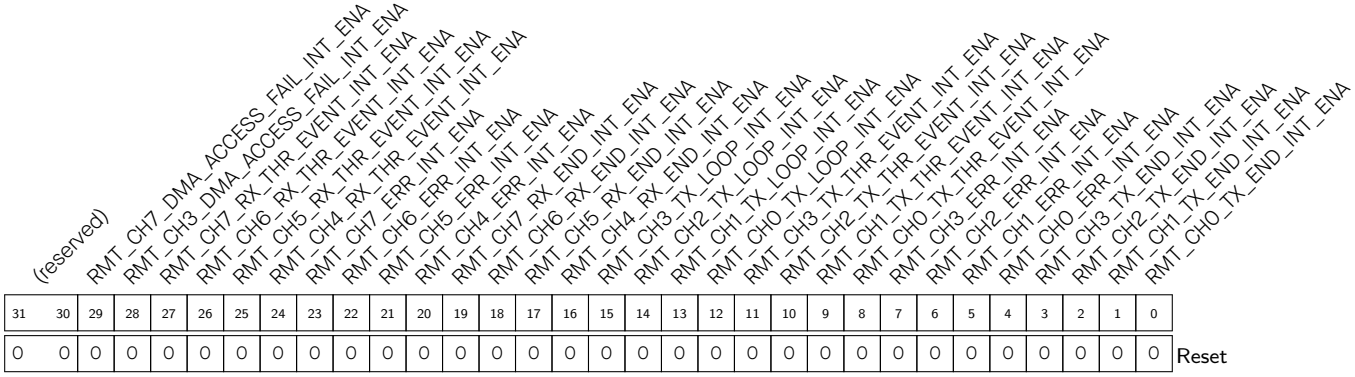
RMT_CH*m*_ERR_INT_ST (*m*: 4-7) RMT_CH*m*_ERR_INT 的屏蔽中断状态。(RO)

RMT_CH*m*_RX_THR_EVENT_INT_ST (*m*: 4-7) RMT_CH*m*_RX_THR_EVENT_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_ST (*n*: 3) RMT_CH*n*_DMA_ACCESS_FAIL_INT 的屏蔽中断状态。(RO)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_ST (*m*: 7) RMT_CH*m*_DMA_ACCESS_FAIL_INT 的屏蔽中断状态。(RO)

Register 60.13. RMT_INT_ENA_REG (0x0078)



RMT_CH*n*_TX_END_INT_ENA (*n*: 0-3) 写 1 使能 RMT_CH*n*_TX_END_INT。(R/W)

RMT_CH*n*_ERR_INT_ENA (*n*: 0-3) 写 1 使能 RMT_CH*n*_ERR_INT。(R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA (*n*: 0-3) 写 1 使能 RMT_CH*n*_TX_THR_EVENT_INT。(R/W)

RMT_CH*n*_TX_LOOP_INT_ENA (*n*: 0-3) 写 1 使能 RMT_CH*n*_TX_LOOP_INT。(R/W)

RMT_CH*m*_RX_END_INT_ENA (*m*: 4-7) 写 1 使能 RMT_CH*m*_RX_END_INT。(R/W)

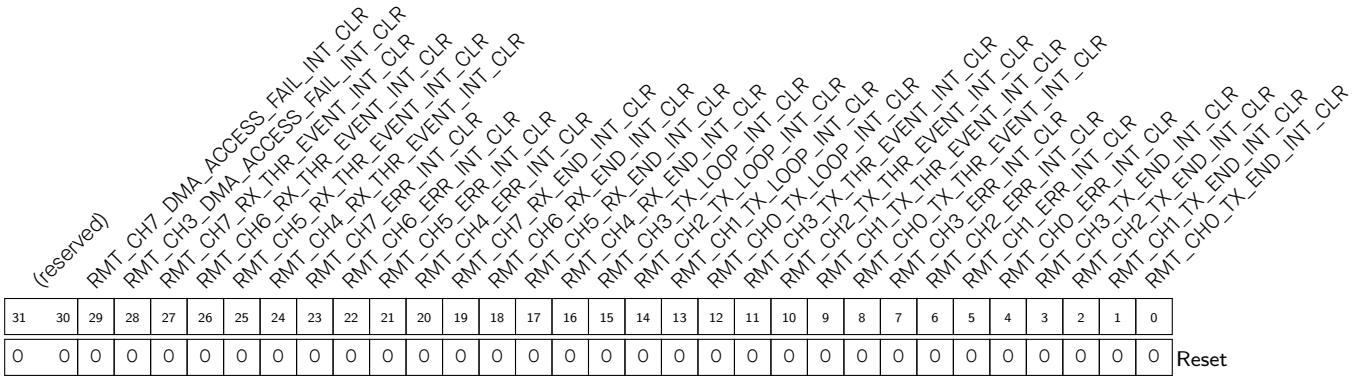
RMT_CH*m*_ERR_INT_ENA (*m*: 4-7) 写 1 使能 RMT_CH*m*_ERR_INT。(R/W)

RMT_CH*m*_RX_THR_EVENT_INT_ENA (*m*: 4-7) 写 1 使能 RMT_CH*m*_RX_THR_EVENT_INT。(R/W)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_ENA (*n*: 3) 写 1 使能 RMT_CH*n*_DMA_ACCESS_FAIL_INT。(R/W)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_ENA (*m*: 7) 写 1 使能 RMT_CH*m*_DMA_ACCESS_FAIL_INT。(R/W)

Register 60.14. RMT_INT_CLR_REG (0x007C)



RMT_CH*n*_TX_END_INT_CLR (*n*: 0-3) 写 1 清除 RMT_CH*n*_TX_END_INT。(WT)

RMT_CH*n*_ERR_INT_CLR (*n*: 0-3) 写 1 清除 RMT_CH*n*_ERR_INT。(WT)

RMT_CH*n*_TX_THR_EVENT_INT_CLR (*n*: 0-3) 写 1 清除 RMT_CH*n*_TX_THR_EVENT_INT。(WT)

RMT_CH*n*_TX_LOOP_INT_CLR (*n*: 0-3) 写 1 清除 RMT_CH*n*_TX_LOOP_INT。(WT)

RMT_CH*m*_RX_END_INT_CLR (*m*: 4-7) 写 1 清除 RMT_CH*m*_RX_END_INT。(WT)

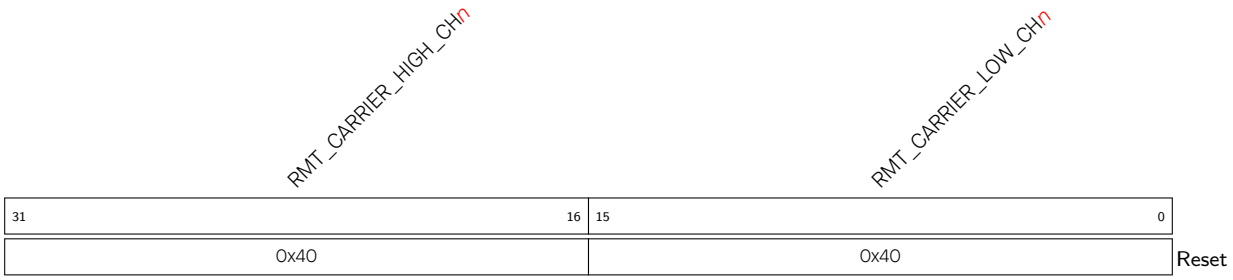
RMT_CH*m*_ERR_INT_CLR (*m*: 4-7) 写 1 清除 RMT_CH*m*_ERR_INT。(WT)

RMT_CH*m*_RX_THR_EVENT_INT_CLR (*m*: 4-7) 写 1 清除 RMT_CH*m*_RX_THR_EVENT_INT。(WT)

RMT_CH*n*_DMA_ACCESS_FAIL_INT_CLR (*n*: 3) 写 1 清除 RMT_CH*n*_DMA_ACCESS_FAIL_INT。(WT)

RMT_CH*m*_DMA_ACCESS_FAIL_INT_CLR (*m*: 7) 写 1 清除 RMT_CH*m*_DMA_ACCESS_FAIL_INT。(WT)

Register 60.15. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-3) (0x0080+0x4**n*)



RMT_CARRIER_LOW_CH*n* 配置通道 *n* 载波的低电平时钟周期。

测量单位：rmt_sclk

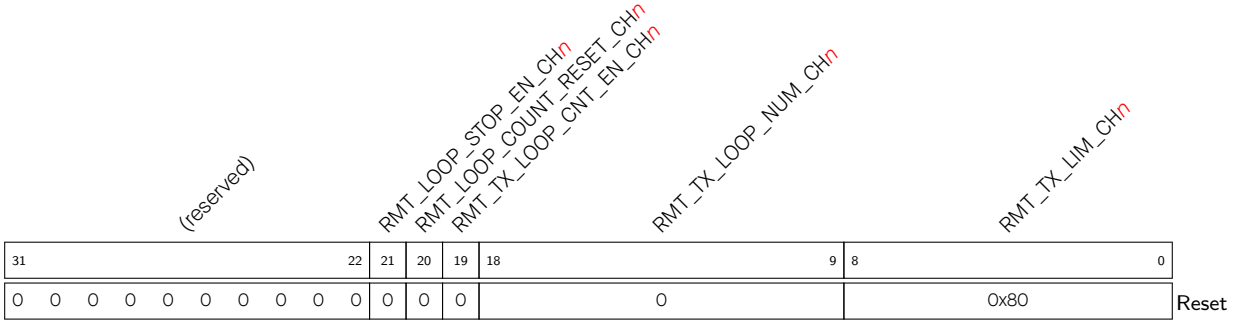
(R/W)

RMT_CARRIER_HIGH_CH*n* 配置通道 *n* 载波的高电平时钟周期。

测量单位：rmt_sclk

(R/W)

Register 60.16. RMT_CH*n*_TX_LIM_REG (*n*: 0-3) (0x00A0+0x4*n*)



RMT_TX_LIM_CH*n* 配置通道 *n* 发送脉冲编码数量的上限值。(R/W)

RMT_TX_LOOP_NUM_CH*n* 配置持续发送模式下最大循环发送次数。(R/W)

RMT_TX_LOOP_CNT_EN_CH*n* 配置是否使能循环次数计数。

0：无效

1：使能

(R/W)

RMT_LOOP_COUNT_RESET_CH*n* 配置是否重置持续发送模式下的循环计数器。

0：无效

1：重置

(WT)

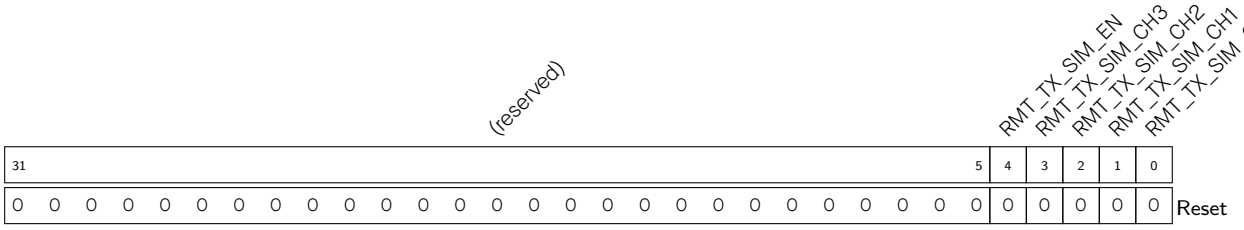
RMT_LOOP_STOP_EN_CH*n* 配置是否在通道 *n* 的循环计数器记到循环数后使能循环发送停止功能。

0：无效

1：使能

(R/W)

Register 60.17. RMT_TX_SIM_REG (0x00C4)



RMT_TX_SIM_CH*n* (*n*: 0-3) 配置是否使能通道 *n* 与其它启用的通道同步开始发送数据。

0: 不使能

1: 使能

(R/W)

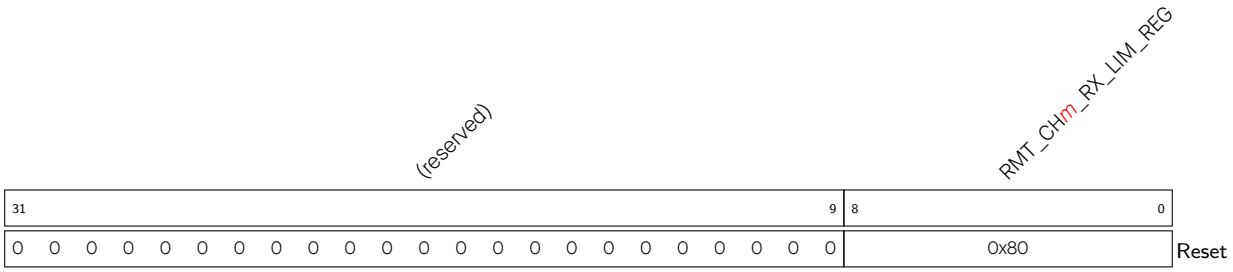
RMT_TX_SIM_EN 配置是否使能多个通道开始同步发送数据。

0: 不使能

1: 使能

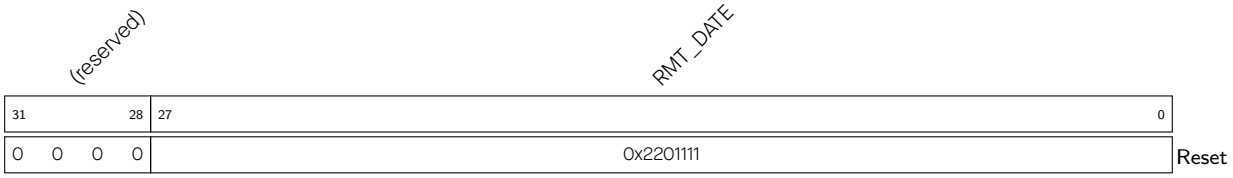
(R/W)

Register 60.18. RMT_CH*m*_RX_LIM_REG (*m*: 4-7) (0x00B0, 0x00B4, 0x00B8, 0x00BC)



RMT_CH*m*_RX_LIM_REG 配置通道 *m* 最大可接收的脉冲编码数量。(R/W)

Register 60.19. RMT_DATE_REG (0x00CC)



RMT_DATE 版本控制寄存器。(R/W)

第 61 章

并行 IO 控制器 (PARLIO)

61.1 概况

ESP32-P4 包含一个并行 IO 控制器 (PARLIO)，支持通过通用直接存储访问 (GDMA) 在并行总线上实现外部设备和内部存储器之间的数据通信。它由 TX 和 RX 两个子模块组成，分别作为发送器和接收器。PARLIO 支持全双工通信。

鉴于 IO 数据的灵活性，PARLIO 可用作通用接口，连接各种外围设备。例如可以把 SPI 作为主机，PARLIO 作为从机，实现点对点传输。详细的应用实例，请参考小节 61.8。

61.2 术语

为了更好地说明 PARLIO 的功能，本章使用了以下术语。

RX 模块 PARLIO 子模块，从外部并行总线接收数据并将其存储在内部存储器中。

TX 模块 PARLIO 子模块，将数据从内部存储器传输到外部并行总线上。

RXD 从 RX 模块的 IO 接口接收的并行数据。

TXD 从 TX 模块的 IO 接口发送的并行数据。

帧 传输数据的单位，从设置 START 信号时起，到收到 EOF 信号止。

自由运行时钟 持续翻转的时钟。若非自由运行时钟，则时钟只在收到有效数据的期间翻转，其余时间停止翻转。

GDMA SUC EOF GDMA 帧传输成功结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧正确，且数据接收结束。

GDMA ERR EOF GDMA 帧传输错误结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧错误，且数据接收结束。

CDC 跨时钟域处理 (Clock Domain Crossing)。

61.3 特性

PARLIO 模块具备以下特性：

- 支持多种时钟源可选：
  - 包括外部 IO 时钟 PAD_CLK_TX/RX、内部系统时钟 XTAL_CLK、PLL_F160M_CLK 和 RC_FAST_CLK
  - 最大支持 40 MHz 的 IO 时钟频率
  - 时钟支持整数和小数分频
- 支持将传输数据总线位宽配置为 1/2/4/8/16 位

- 支持 16 位全双工传输
- 总线位宽为 1/2/4 位时，支持在一个字节范围内对比特数据顺序进行翻转
- 包含用于接收 IO 并行数据的 RX 子模块：
  - 支持对输出时钟进行门控
  - 支持 RX 子模块输入时钟和输出时钟分别取反
  - 支持多种接收模式
  - 支持配置 GDMA SUC EOF 信号生成模式
  - 支持配置外部使能信号的 IO 管脚
- 包含用于发送 IO 并行数据的 TX 子模块：
  - 支持片选功能
  - 支持对输出时钟进行门控
  - 支持 TX 子模块输入时钟和输出时钟分别取反
  - 支持配置 TX EOF 信号生成模式
  - 支持配置总线空闲值

61.4 系统架构

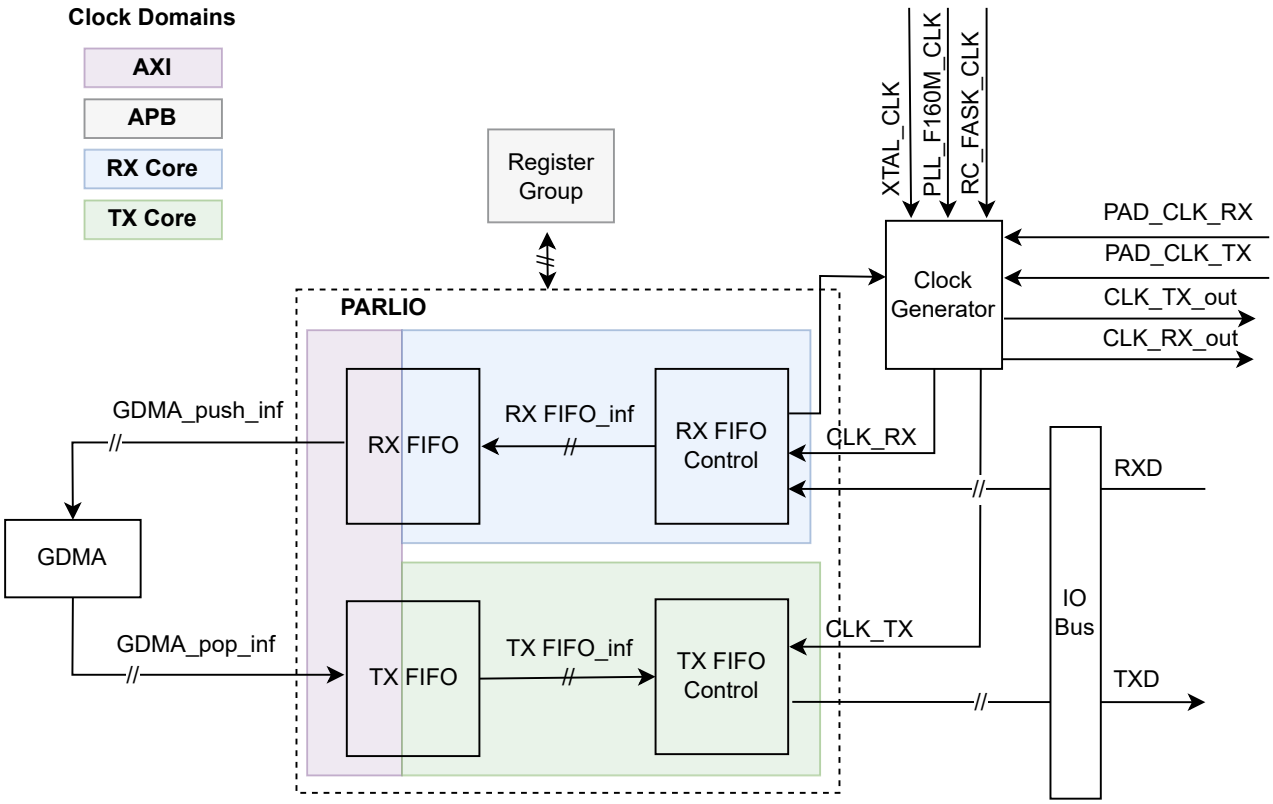


图 61.4-1. PARLIO 系统框图

图 61.4-1 展示了 PARLIO 的系统内部架构。除了 RX 模块和 TX 模块外，PARLIO 还包括一组状态配置寄存器。RX 模块接收 RXD 信号，并在完成串并转换后将其存入异步 FIFO，然后将这些数据通过 GDMA 存到内部存储器。

TX 模块通过 GDMA 从内部存储器获取数据并存入异步 FIFO，将这些数据进行并串转换后，通过 TXD 信号从 IO 总线输出。

61.5 功能描述

61.5.1 时钟生成器

PARLIO 模块有四个输入时钟域，即 RX Core、TX Core、AXI 和 APB，如图 61.4-1 所示。

状态配置寄存器组位于 APB 时钟域内。

GDMA 接口逻辑位于 AXI 时钟域内。

RX Core 有四个时钟源可供选择，分别为内部系统时钟源 XTAL_CLK、RC_FAST_CLK、PLL_F160M_CLK 和外部时钟源 (PAD_CLK_RX)，如图 61.5-1 所示。配置 HP_SYS_CLKRST_PARLIO_RX_CLK_SRC_SEL 选择时钟源，配置 HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUM、HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_DENOMINATOR 和 HP_SYS_CLKRST_PARLIO_RX_CLK_DIV_NUMERATOR 对时钟进行分频，这三个寄存器最大值均为 $(2^8 - 1)$。

TX Core 有四个时钟源可供选择，分别为内部系统时钟源 XTAL_CLK、RC_FAST_CLK、PLL_F160M_CLK 和外部时钟源 (PAD_CLK_TX)，如图 61.5-1 所示。配置 HP_SYS_CLKRST_PARLIO_TX_CLK_SRC_SEL 选择时钟源，配置 HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUM、HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_DENOMINATOR 和 HP_SYS_CLKRST_PARLIO_TX_CLK_DIV_NUMERATOR 对时钟进行分频，这三个寄存器最大值均为 $(2^8 - 1)$。

RX 模块和 TX 模块支持对输入时钟进行取反，同时其工作时钟可以输出到 IO，并在输出前支持对输出时钟进行取反。此外 RX 模块和 TX 模块还支持对输出时钟进行门控。

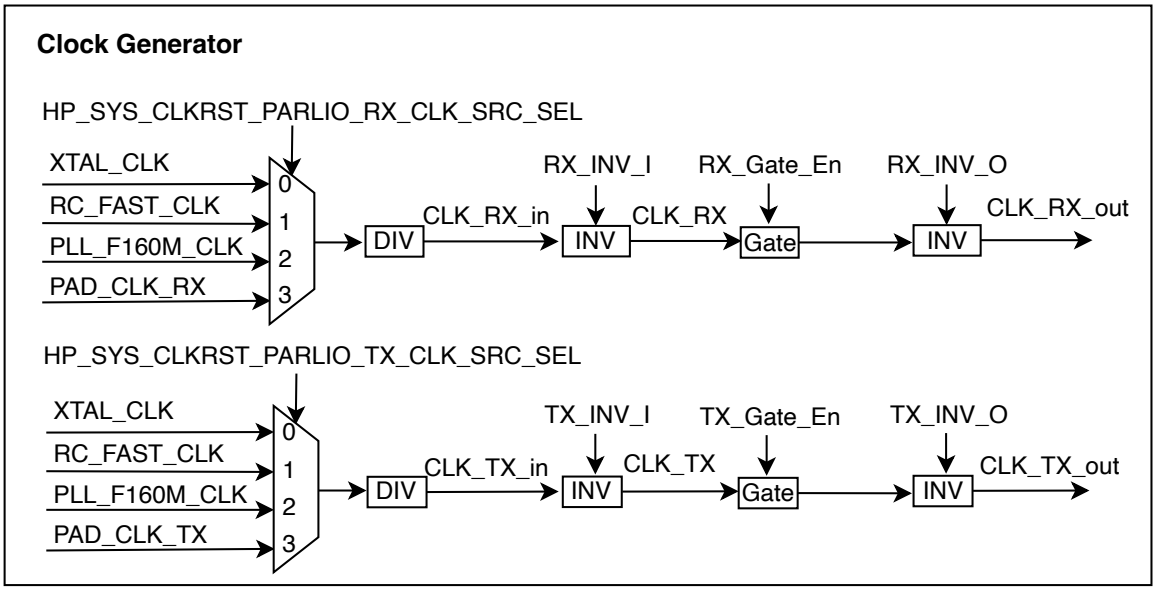


图 61.5-1. PARLIO 时钟生成

61.5.2 时钟复位使用限制

由于 PARLIO 模块的通用性，其 PAD 时钟 (PAD_CLK_TX/RX) 可能来自不同的主机（外部设备或内部时钟源），这些时钟可能是自由运行时钟，也可能不是。如果外部运行时钟不是自由运行时钟，PARLIO 内部的一些控制信号将无法进行 CDC，因此对操作过程会有一定限制。

1. 异步 FIFO 复位时，需要两个时钟周期才能从 AXI 时钟域同步到 Core 时钟域。因此如果需要在使用非自由运行时钟时复位 AXI 时钟域，需保证提前两个时钟周期进行复位同步。具体的操作步骤见下表。

表 61.5-1. 时钟限制下复位 AXI 时钟域操作步骤

时钟限制		具体操作
当前帧传输时钟	下一帧传输时钟	
自由运行时钟	非自由运行时钟	在切换到非自由运行时钟之前复位下一帧传输，并在复位完成后配置时钟切换。
非自由运行时钟	自由运行时钟	只需确保在复位和正式开始传输期间有两个时钟周期的间隔。
非自由运行时钟	非自由运行时钟	如需复位下一帧传输，需要先切换到内部的自由运行时钟，复位完成后再切换回实际时钟。

2. 由于非自由运行时钟造成的限制，PARL_IO_RX_START 和 PARL_IO_TX_START 无法进行 CDC，所以在操作时必须等到 PARL_IO_RX_START 和 PARL_IO_TX_START 稳定后再开始数据传输，否则可能会发生亚稳态的问题。

   RX 模块中的具体操作步骤为：

   - 清除 HP_SYS_CLKRST_PARLIO_RX_CLK_EN 关闭 RX Core 时钟域；
   - 向 PARL_IO_RX_START 写 1；
   - 置位 HP_SYS_CLKRST_PARLIO_RX_CLK_EN 打开 RX Core 时钟域；
   - 操作外部设备，使其开始发送数据；
   - 清除 HP_SYS_CLKRST_PARLIO_RX_CLK_EN 关闭 RX Core 时钟域；
   - 向 PARL_IO_TX_START 写 0。

   TX 模块中的具体操作步骤为：

   - 清除 HP_SYS_CLKRST_PARLIO_TX_CLK_EN 关闭 TX Core 时钟域；
   - 向 PARL_IO_TX_START 写 1；
   - 置位 HP_SYS_CLKRST_PARLIO_TX_CLK_EN 打开 TX Core 时钟域；
   - 操作外部设备，使其开始接收数据；
   - 清除 HP_SYS_CLKRST_PARLIO_TX_CLK_EN 关闭 TX Core 时钟域；
   - 向 PARL_IO_TX_START 写 0。

3. 复位需遵循以下要求：

   - 模块启动时，时钟复位应遵循以下顺序：

     (a) 复位 APB 时钟域；

     (b) 复位 GDMA 时钟域；

(c) 复位 Core 时钟域。

- 帧间传输需要复位 Core 时钟域和异步 FIFO。

61.5.3 主从机模式

RX 模块和 TX 模块均支持主从机模式。

当 TX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。TX 模块在时钟的上升沿驱动 TXD。

当 TX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 61.5-2. 时钟限制下 TX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求。
非自由运行时钟	正向波形（图 61.5-2）	主机需要在下降沿时采集。
非自由运行时钟	负向波形（图 61.5-3）	主机在输出前需要对原时钟进行取反，将其转换成正向波形。

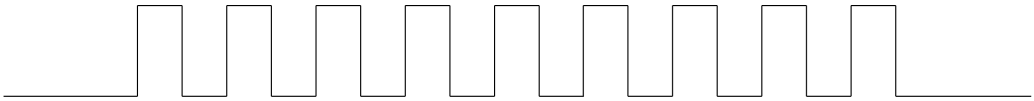


图 61.5-2. 主机时钟正向波形

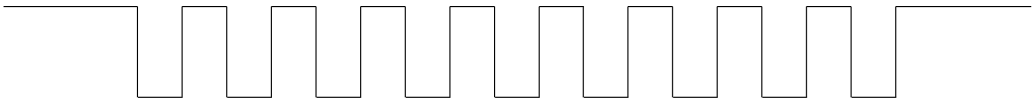


图 61.5-3. 主机时钟负向波形

当 RX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。RX 模块在时钟的上升沿驱动 RXD。

当 RX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 61.5-3. 时钟限制下 RX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求，采集的有效数据以外部使能信号为准。
非自由运行时钟	正向波形（图 61.5-2）	主机需在上升沿驱动数据，RX 模块需在下降沿采集数据（即对主时钟进行翻转）。
非自由运行时钟	负向波形（图 61.5-3）	主机需在下降沿驱动数据，RX 模块需在上升沿采集数据（即使用原始主时钟）

61.5.4 RX 模块接收模式

PARLIO 的 RX 模块有八种接收模式，可根据使能信号规格分为三大类：

- 电平使能模式：接收的数据由外部信号电平使能；
- 脉冲使能模式：接收的数据由外部信号脉冲使能；
- 软件使能模式：数据接收的使能信号可由用户直接配置。

RX 模块还支持对外部使能信号进行翻转。如果外部使能信号为低电平或负脉冲，可通过置位 PARL_IO_RX_EXT_EN_INV 使能该功能，转换为以下相应的接收模式。

61.5.4.1 电平使能模式

电平使能模式如图 61.5-4 所示。在这种模式下，外部使能信号的有效电平必须与有效数据对齐。由于外部电平使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 15 个。

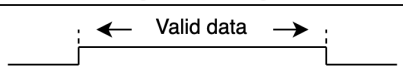
Mode	Sub-mode	Description
LEVEL_ENABLE	\	*signal level high* 

图 61.5-4. RX 模块电平使能模式

61.5.4.2 脉冲使能模式

根据脉冲有效电平及其与有效数据的对应性，脉冲使能模式可分为六种子模式，详细分类可参见图 61.5-5。

子模式 1 ~ 4 都包含起始脉冲和结束脉冲，不同之处在于起始脉冲和结束脉冲是否与有效数据对齐。

子模式 5 ~ 6 只包含起始脉冲，通过配置 PARL_IO_RX_BITLEN 可表示有效数据传输结束。

由于外部脉冲使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 15 个。但在子模式 4 和子模式 6 下，数据在脉冲的第一个边沿后和最后一个边沿前被认为有效，可以实现使能信号的 IO 管脚与数据 IO 管脚的复用，此时 RXD 可使用 16 个 IO 管脚。

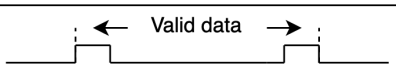
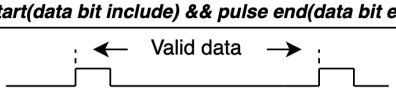
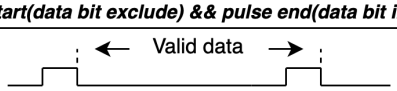
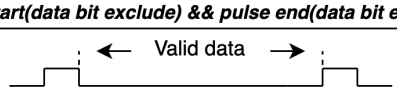
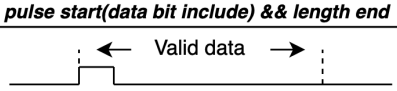
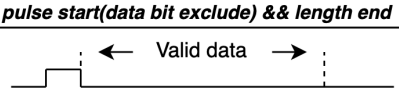
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	pulse start(data bit include) && pulse end(data bit include) 
	sub-mode2	pulse start(data bit include) && pulse end(data bit exclude) 
	sub-mode3	pulse start(data bit exclude) && pulse end(data bit include) 
	sub-mode4	pulse start(data bit exclude) && pulse end(data bit exclude) 
	sub-mode5	pulse start(data bit include) && length end 
	sub-mode6	pulse start(data bit exclude) && length end 

图 61.5-5. RX 模块脉冲使能子模式

61.5.4.3 软件使能模式

软件使能模式下的使能信号由内部配置寄存器决定。用户切换到此模式时，只有同时写 1 到 PARL_IO_RX_SW_EN 和 PARL_IO_RX_START，才能开始接收数据。

由于使能信号不占用接口 IO 管脚，RXD 在接口上的最大可用 IO 管脚数量为 16。由于时钟域不同，使能信号无法与有效数据对齐，因此只能通过有效时钟沿来判定有效数据。此时，RX Core 时钟需要与有效数据一一对应。

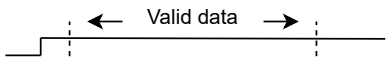
Mode	Sub-mode	Description
SW_ENABLE	/	/ 

图 61.5-6. RX 模块软件使能模式

61.5.5 RX 模块 GDMA SUC EOF 信号生成

RX 模块产生一个 GDMA SUC EOF 信号，表示当前帧传输结束，并将其发送到 GDMA 接口。GDMA SUC EOF 信号可由外部使能信号生成，也可基于内部配置的比特长度触发。

- 当 GDMA SUC EOF 信号由内部配置的比特长度生成时，接收模式的选择不受限制。但在这种情况下必须配置 PARL_IO_RX_BITLEN。如果 PARL_IO_RX_BITLEN 的配置值小于实际接收的数据，将提前触发 GDMA SUC EOF，此时 RX 模块停止从 FIFO 读取数据，但 FIFO 会继续接收外部数据，直到 RX_FIFO_WOVF_INT 中断被触发。
- 当 GDMA SUC EOF 信号由外部使能信号产生时，只可选脉冲使能模式的子模式 1、3。在这种模式下，传输不受 PARL_IO_RX_BITLEN 值的影响，并且传输帧的数据长度不受限制。

61.5.6 TX 模块 EOF 信号生成

TX 模块产生一个 EOF 信号，并生成 TX_EOF_INT 中断表示当前帧传输结束。TX 模块 EOF 信号可由 GDMA 配置的 EOF 信号生成，也可基于内部配置的比特长度触发。

- 当 TX EOF 信号由内部配置的比特长度生成时，但在这种情况下必须配置 PARL_IO_TX_BITLEN。如果 PARL_IO_TX_BITLEN 的配置值小于实际发送的数据，将提前触发 TX EOF。
- 当 TX EOF 信号由 GDMA 配置的 EOF 信号产生时，传输不受 PARL_IO_TX_BITLEN 值的影响，并且传输帧的数据长度不受限制，直到接收到 GDMA 配置的 EOF 信号时，传输才会结束。

61.5.7 RX 模块超时

RX 模块支持接收超时功能。当 RX 模块 FIFO 长时间未接收到有效数据，超时将被触发，产生一个 ERR EOF 信号，并发送至 GDMA 接口以表示结束数据接收。配置 PARL_IO_RX_TIMEOUT_THRES 可设置超时阈值。

默认情况下启用超时功能，用户也可以选择禁用该功能。超时阈值的配置上限为 GDMA 时钟域的 $(2^{16}-1)$ 周期，下限取决于 GDMA 时钟域和 RX Core 时钟域之间的相对频率关系。建议将 PARL_IO_RX_TIMEOUT_THRES 设置为较大的数值，以避免生成非期望的 GDMA ERR EOF 信号。

61.5.8 TX 模块无限制长度数据传输

默认情况下，TX 模块的传输结束标志由 PARL_IO_TX_BITLEN 的配置值决定。当发送的数据达到配置值时，系统会生成一个 TX EOF 中断信号，标志着该数据帧的传输结束。此时，单帧传输的最大值受到 PARL_IO_TX_BITLEN 的位宽限制，使其无法实现无限长度的数据传输。

为了实现更大长度的单帧传输，可以将 PARL_IO_TX_EOF_GEN_SEL 设置为 1。在这种配置下，硬件将使用 GDMA 发送链表配置的 EOF 信号作为单帧传输结束的标志。此时，数据传输不再受限于 PARL_IO_TX_BITLEN 的位宽，从而能够支持更大长度的数据传输。

61.5.9 TX 片选功能

为模拟 LCD 的 I8080 协议，TX 模块配备了一个可输出的片选信号，在拉低时有效。用户可通过配置 PARL_IO_TX_CS_START_DELAY 和 PARL_IO_TX_CS_STOP_DELAY 来控制 DMA 输出数据和片选信号的相位关系。需要注意的是，在开启时钟门控、且使用片选信号作为时钟门控控制源的情况下，对上述两个寄存器的操作在实现基础功能的同时，还可以控制输出时钟和片选信号的相位关系。

61.5.10 TX 模块输出时钟门控

TX 模块支持对输出时钟进行门控。时钟门控功能默认关闭，可通过软件置位 PARL_IO_TX_GATING_EN 使能该功能。

目前 TXD 最高位有两个控制源可配置，分别为 DMA 输出数据和片选信号 (CS)。将 PARL_IO_TX_VALID_OUTPUT_EN 配置为 0 即选择 DMA 输出数据为控制源，此时必须将 TX 模块配置为最大位宽，因为当控制源为 GDMA 输出数据时，门控信号固定为 TXD 最高位，该位为高电平时，时钟信号可以翻转。需要注意的是，在选择 GDMA 输出数据作为控制源时，如果外设处于空闲状态，则此时时钟门控将被用户配置的总线空闲值的最高位控制，应参照小节 61.5.11 *TX 模块总线空闲值* 中的注意事项进行配置；若选择 DMA 输出数据作为门控功能控制源时启用时钟门控功能，门控信号将占用一个 IO，此时 TXD 可以使用的 IO 管脚最多为 15 个。

将 PARL_IO_TX_VALID_OUTPUT_EN 配置为 1 即选择片选信号为控制源，此时对位宽无限制。

61.5.11 TX 模块总线空闲值

TX 模块不传输数据时被认为保持在空闲状态。TX 模块支持总线空闲值可配置。

总线空闲值默认为 0x0，最大可配置值为 0xFF。注意，应避免配置的空闲值与其他已开启的功能产生冲突，例如当使用 TXD 的最高有效位作为有效信号时，应避免将空闲值的最高有效位配置为 1。

61.5.12 单帧数据传输

RX/TX 模块以比特为单位进行传输，即单帧至少传输 1 个比特的数据。

当 RX 模块通过比特长度生成 GDMA EOF 信号时，单帧传输的最大长度是 $(2^{19}-1)$ 比特。当 RX 模块通过外部设备的使能信号产生 GDMA EOF 信号时，单帧传输的比特数没有限制。

当 TX 模块通过比特长度生成 EOF 信号时，单帧传输的最大长度为 $(2^{19}-1)$ 比特。当 TX 模块通过 GDMA 配置的 EOF 生成 TX EOF 信号，单帧传输的比特数不受限制。

由于 PARLIO 在 GDMA 一侧以字节为单位进行传输，因此当 IO 一侧的数据不与字节对齐时，PARLIO 会对数据进行特殊处理。

- 在接收状态下，通过 GDMA 存入内存的数据将自动在高位补 0，从而与字节对齐。
- 在发送状态下，从内存中取出的数据将根据配置的 PARL_IO_TX_BITLEN 值被截断，超过该配置值的数据不会被发送。

当配置的数据总线宽度为 2/4/8/16 位时，比特长度必须配置为对应总线宽度的倍数。

61.5.13 字节范围内数据顺序翻转

总线位宽为 1/2/4 位时，可以对一个字节范围内的数据顺序进行翻转。以 RX 模块为例，当配置总线位宽为 2 比特时，数据在被写入 RX FIFO 之前需要被打包成一个字节。

假设原始的比特数据排列顺序为：

$$\{\{b_0,b_1\},\{b_2,b_3\},\{b_4,b_5\},\{b_6,b_7\}\}$$

则启用该功能后，数据顺序将更改为：

$$\{\{b_6,b_7\},\{b_4,b_5\},\{b_2,b_3\},\{b_0,b_1\}\}$$

61.6 中断

ESP32-P4 的 PARLIO 模块可以生成以下中断信号并发送给中断矩阵。

- PARL_IO_TX_INTR
- PARL_IO_RX_INTR

这些中断信号由 PARLIO 模块的内部中断源生成。表 61.6-1 列出了 PARLIO 的内部中断源、中断触发条件、以及生成的中断信号。

表 61.6-1. PARLIO 的内部中断源

内部中断源	触发条件	中断信号
TX_FIFO_REMPTY_INT	TX 模块 FIFO 空（此时 TX 发送的数据可能出现错误）	PARL_IO_TX_INTR

RX_FIFO_WOVF_INT	RX 模块 FIFO 满（此时 RX 接收的数据可能出现错误）	PARL_IO_RX_INTR
TX_EOF_INT	TX 模块发送完成完整的一帧数据	PARL_IO_TX_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 61.9 寄存器列表。

61.7 配置流程

61.7.1 数据接收操作流程

本节介绍 RX 模块接收数据的配置流程。参照以下流程，从连接到外部设备的 IO 管脚接收并行数据，并存储在内部存储器中。关于 RX 模块时钟和复位的详细操作限制，请参考小节 61.5.2。

1. 复位 RX 模块。具体复位方法与顺序，请参考小节 61.5.2。
2. 置位 PARL_IO_RX_FIFO_WOVF_INT_CLR 和 PARL_IO_RX_FIFO_WOVF_INT_ENA。
3. 选择 RXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置时钟寄存器，选择时钟源并对时钟进行分频。
5. 关闭 RX 模块 Core 时钟域的时钟。
6. 按照小节 61.3 和 61.5 中的描述，选择所需接收模式并启用相关功能。
7. 配置 GDMA 接收链表。
8. 置位 PARL_IO_RX_REG_UPDATE，同步寄存器信号。
9. 置位 PARL_IO_RX_START。
10. 打开 RX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其发送数据。
12. 轮询 GDMA SUC EOF 中断信号。
13. 清除 GDMA SUC EOF 中断信号。
14. 关闭 RX 模块 Core 时钟域的时钟。
15. 清除 PARL_IO_RX_START。

61.7.2 数据发送操作流程

本节介绍 TX 模块发送数据的编程流程。参照以下流程，将内部存储器中的并行数据传输到与外部设备相连的 IO 管脚。关于 TX 模块时钟和复位的详细操作限制，请参考小节 61.5.2。

1. 复位 TX 模块。具体复位方法与顺序，请参考小节 61.5.2。

2. 依次置位 PARL_IO_TX_FIFO_REMPTY_INT_CLR、PARL_IO_TX_EOF_INT_CLR、PARL_IO_TX_FIFO_REMPTY_INT_ENA 和 PARL_IO_TX_EOF_INT_ENA。
3. 选择 TXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置时钟寄存器，选择时钟源并对时钟进行分频。
5. 关闭 TX 模块 Core 时钟域的时钟。
6. 按照小节 61.5 中的描述，选择开启所需功能。
7. 配置 GDMA 发送链表。
8. 轮询 PARL_IO_TX_READY。
9. 置位 PARL_IO_TX_START。
10. 打开 TX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其接收数据。
12. 轮询 PARL_IO_TX_EOF_INT_ST 中断信号。
13. 置位 PARL_IO_TX_EOF_INT_CLR 中断信号。
14. 关闭 TX 模块 Core 时钟域的时钟。
15. 清除 PARL_IO_TX_START。

61.8 应用示例

本节介绍 PARLIO 模块的应用示例及其具体操作过程。示例中使用的外围设备都来自 ESP 系列芯片，这些外设均可以和 PARLIO 模块组成完整的数据通路。

> **说明：**
> 示例中所构建的数据通路可能不是最佳方案，例如，在实际应用中，用户可使用两颗相同 ESP 芯片上的 SPI 外设来完成点对点传输，而无需使用 PARLIO。但本小节中的示例在一定程度上证明了 PARLIO 接口的灵活性。

61.8.1 与 SPI 进行数据传输

在本示例中，外部 SPI 作为主机发送数据，PARLIO RX 模块作为从机接收数据，与此同时，PARLIO TX 模块作为主机发送数据，外部 SPI 作为从机接收数据，从而实现点对点串行数据传输。

- 实现 SPI 发送、PARLIO 接收，请参考以下操作过程：
  1. 配置 SPI 时钟。
  2. 将 SPI 配置为主机。
  3. 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_RX，FSPICS0 管脚连接至 RXD[15]，FSPID 管脚连接至 RXD[0]。
  4. 将发送的数据写入 SPI 缓冲器中，并配置发送数据的位长。
  5. 置位 SPI_UPDATE，更新配置的寄存器值。
  6. 复位 PARLIO RX 模块。

7. 配置 PARLIO RX 模块时钟。
8. 关闭 PARLIO RX 模块 Core 时钟域。
9. 将 PARLIO 接收模式配置为电平使能模式的子模式 1，配置 RX 模块数据总线宽度为 1 位，然后根据 SPI 的发送数据长度配置 PARL_IO_RX_BITLEN，最后置位 PARL_IO_RX_REG_UPDATE。
10. 配置 PARLIO GDMA 接收链表。
11. 置位 PARL_IO_RX_START。
12. 打开 PARLIO RX 模块 Core 时钟域。
13. 置位 SPI_USR，开始传输 SPI 的数据。
14. 轮询 GDMA SUC EOF 中断信号。
15. 清除 PARL_IO_RX_START。

- 实现 PARLIO 发送、SPI 接收，请参考以下操作过程：
  1. 配置 SPI 时钟。
  2. 将 SPI 配置为从机。
  3. 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_TX，FSPICSO 管脚连接至 TXD[15]，FSPID 管脚连接至 TXD[0]。
  4. 置位 SPI_RD_BIT_ORDER，翻转比特数据顺序。
  5. 置位 SPI_UPDATE，更新配置的寄存器值。
  6. 复位 PARLIO TX 模块。
  7. 置位 PARL_IO_TX_EOF_INT_CLR 和 PARL_IO_TX_EOF_INT_ENA。
  8. 配置 PARLIO TX 模块时钟。
  9. 关闭 PARLIO TX 模块 Core 时钟域。
  10. 配置数据总线宽度为 1 位，写 1 至 PARL_IO_TX_VALID_OUTPUT_EN，然后配置 PARL_IO_TX_BITLEN。
  11. 配置 PARLIO GDMA 发送链表。
  12. 轮询 PARL_IO_TX_READY。
  13. 写 1 至 PARL_IO_TX_START。
  14. 打开 PARLIO TX 模块 Core 时钟域。
  15. 开始数据传输。
  16. 轮询 PARL_IO_TX_EOF_INT_ST。
  17. 置位 PARL_IO_TX_EOF_INT_CLR。
  18. 关闭 PARLIO TX 模块 Core 时钟域。
  19. 清除 PARL_IO_TX_START。

61.8.2 与 I2S 进行数据传输

在本示例中，外部 I2S 作为主机发送数据，PARLIO RX 模块作为从机接收数据。PARLIO 支持 I2S TDM MSB 对齐标准模式和 TDM PCM 标准模式下的传输。当 I2S 传输协议为 TDM MSB 对齐标准模式时，需将 PARLIO 的接收模式配置为电平使能模式。当 I2S 传输协议为 TDM PCM 标准模式时，需将 PARLIO 的接收模式配置为脉冲使能模式的子模式 4，并置位 PARL_IO_RX_EXT_EN_INV。

本节以 TDM PCM 标准模式为例，具体操作过程如下：

1. 配置 I2S 时钟。
2. 配置信号管脚，将 I2SO_BCK_out 连接至 PAD_CLK_RX，I2SO_WS_out 连接至 RXD[15]，I2SO_Data_out 连接至 RXD[0]。
3. 将 I2S 配置为主机。
4. 配置所需 I2S TX 数据和通道模式，并置位 I2S_TX_UPDATE。
5. 复位 I2S TX 模块和 TX FIFO。
6. 使能 I2S_TX_DONE_INT。
7. 配置 I2S GDMA 发送链表。
8. 置位 I2S_TX_STOP_EN。
9. 复位 PARLIO RX 模块。
10. 配置 PARLIO RX 模块时钟。
11. 关闭 PARLIO RX 模块 Core 时钟域。
12. 将 PARLIO 接收模式配置为脉冲使能模式的子模式 4，并置位 PARL_IO_RX_EXT_EN_INV，配置 RX 模块数据总线宽度为 1 位，然后根据 I2S 的发送数据长度配置 PARL_IO_RX_BITLEN，最后置位 PARL_IO_RX_REG_UPDATE。
13. 配置 PARLIO GDMA 接收链表。
14. 置位 PARL_IO_RX_START。
15. 打开 PARLIO RX 模块 Core 时钟域。
16. 置位 I2S_TX_START，开始发送数据。
17. 轮询 I2S_TX_DONE_INT。
18. 轮询 GDMA SUC EOF 中断信号。
19. 清除 I2S_TX_START。
20. 清除 PARL_IO_RX_START。

61.8.3 与 LCD 进行数据传输

在本示例中，PARLIO TX 模块作为主机向外部 LCD 发送数据（使用 I8080/MOTO6800 格式），设备间传输 8 位并行数据。具体操作过程如下：

1. 配置信号管脚。将 CLK_TX_out 连接至 LCD 的像素时钟 PCLK，TXD[7:0] 连接至 LCD 数据输入脚，TXD[8] 连接至 LCD 的 CS 脚，TXD[9] 连接至 LCD 的 CD 脚。

2. 复位 PARLIO TX 模块。
3. 依次置位 PARL_IO_TX_FIFO_REMPTY_INT_CLR、PARL_IO_TX_EOF_INT_CLR、PARL_IO_TX_FIFO_REMPTY_INT_ENA 和 PARL_IO_TX_EOF_INT_ENA。
4. 配置 PARLIO TX 模块时钟。
5. 关闭 PARLIO TX 模块 Core 时钟域。
6. 配置数据总线宽度为 16 位，然后配置 PARL_IO_TX_BITLEN。
7. 配置 GDMA 发送链表。注意，在链表中发送的数据应符合 I8080/MOTO6800 格式。低 8 位是有效的并行数据，第 9、10 位分别是 CS 和 CD，最高有效位是常数 1，其余位为任意值。
8. 轮询 PARL_IO_TX_READY。
9. 写 1 至 PARL_IO_TX_START。
10. 打开 PARLIO TX 模块 Core 时钟域。
11. 开始数据传输。
12. 轮询 PARL_IO_TX_EOF_INT_ST。
13. 置位 PARL_IO_TX_EOF_INT_CLR。
14. 关闭 PARLIO TX 模块 Core 时钟域。
15. 清除 PARL_IO_TX_START。

61.9 寄存器列表

本小节的所有地址均为相对于并行 IO 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

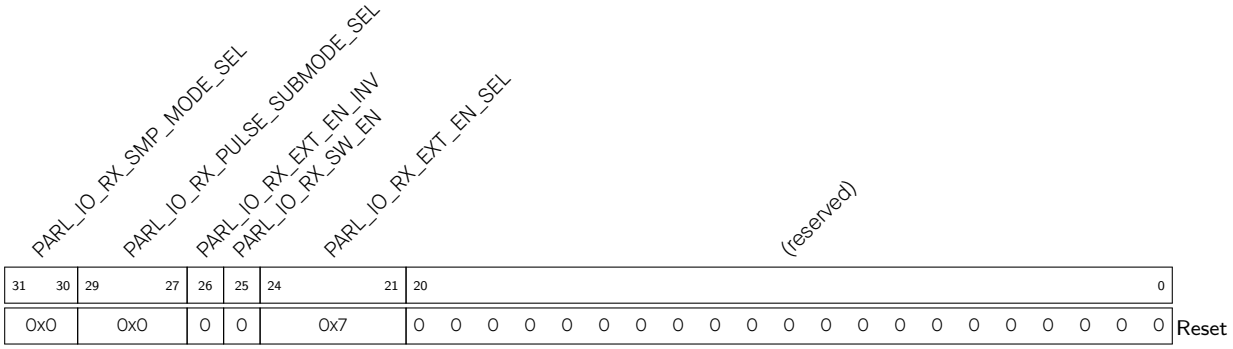
名称	描述	地址	访问
PARLIO RX 配置寄存器			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX 采样模式配置寄存器	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX 数据配置寄存器	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX 通用配置寄存器	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX 开始配置寄存器	0x000C	R/W
PARLIO TX 配置寄存器			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX 数据配置寄存器	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX 开始配置寄存器	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX 通用配置寄存器	0x0018	R/W
PARLIO 配置与状态寄存器			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO 配置寄存器	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO 寄存器更新配置寄存器	0x0020	WT
PARL_IO_ST_REG	PARLIO 状态寄存器	0x0024	RO
PARLIO 中断配置与状态寄存器			
PARL_IO_INT_ENA_REG	PARLIO 中断使能寄存器	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO 原始中断寄存器	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO 中断状态寄存器	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO 中断清除寄存器	0x0034	WT
PARLIO RX/TX 状态寄存器			
PARL_IO_RX_ST0_REG	PARLIO RX 状态寄存器 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX 状态寄存器 1	0x003C	RO
PARL_IO_TX_ST0_REG	PARLIO TX 状态寄存器 0	0x0040	RO
PARLIO 时钟配置寄存器			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX 时钟配置寄存器	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX 时钟配置寄存器	0x0048	R/W
PARL_IO_CLK_REG	PARLIO 时钟配置寄存器	0x0120	R/W
PARLIO TX 片选配置寄存器			
PARL_IO_TX_CS_CFG_REG	PARLIO TX 片选配置寄存器	0x004C	R/W
PARLIO 版本寄存器			
PARL_IO_VERSION_REG	版本控制寄存器	0x03FC	R/W

61.10 寄存器

本小节的所有地址均为相对于并行 IO 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 61.1. PARL_IO_RX_MODE_CFG_REG (0x0000)



PARL_IO_RX_EXT_EN_SEL 配置 RX 外部使能信号的 IO 管脚。(R/W)

PARL_IO_RX_SW_EN 配置是否使能软件数据采集。

0：不使能

1：使能

(R/W)

PARL_IO_RX_EXT_EN_INV 配置是否翻转外部使能信号。

0：无效

1：翻转

(R/W)

PARL_IO_RX_PULSE_SUBMODE_SEL 配置 RXD 脉冲使能子模式。

0：正脉冲起始（包括数据位）& 正脉冲结束（包括数据位）

1：正脉冲起始（包括数据位）& 正脉冲结束（不包括数据位）

2：正脉冲起始（不包括数据位）& 正脉冲结束（包括数据位）

3：正脉冲起始（不包括数据位）& 正脉冲结束（不包括数据位）

4：正脉冲起始（包括数据位）& 长度结束

5：正脉冲起始（不包括数据位）& 长度结束

(R/W)

PARL_IO_RX_SMP_MODE_SEL 配置 RXD 采集模式。

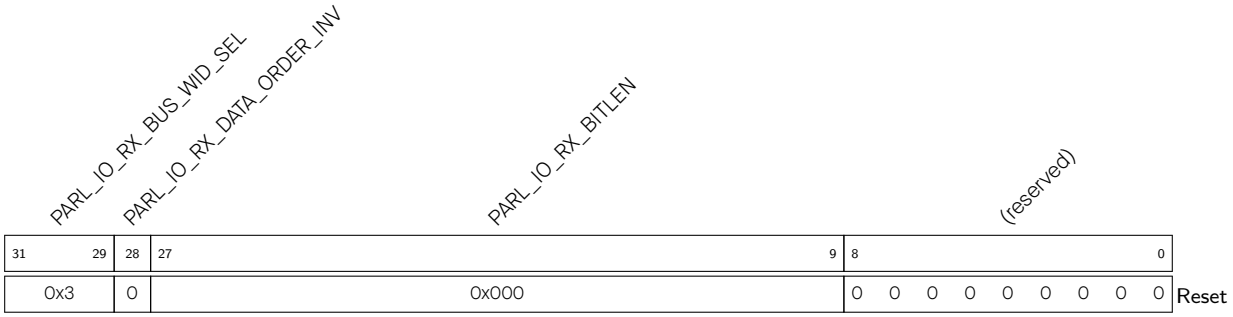
0：外部电平使能模式

1：外部脉冲使能模式

2：内部软件使能模式

(R/W)

Register 61.2. PARL_IO_RX_DATA_CFG_REG (0x0004)



PARL_IO_RX_BITLEN 配置 RXD 的比特长度。(R/W)

PARL_IO_RX_DATA_ORDER_INV 配置是否翻转从 RX_FIFO 发送到 DMA 的数据的字节范围内比特顺序。

0：无效

1：翻转数据顺序

(R/W)

PARL_IO_RX_BUS_WID_SEL 配置 RXD 总线位宽。

0：1 位

1：2 位

2：4 位

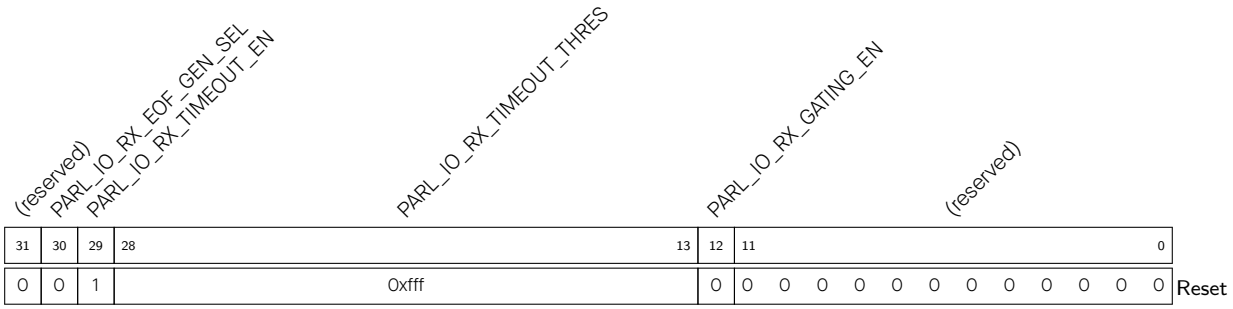
3：8 位

4：16 位

5 ~ 15：无效值，会引发错误

(R/W)

Register 61.3. PARL_IO_RX_GENRL_CFG_REG (0x0008)



PARL_IO_RX_GATING_EN 配置是否使能 RX 输出时钟的时钟门控。

0：不使能

1：使能

(R/W)

PARL_IO_RX_TIMEOUT_THRES 配置 RX 超时计数器的阈值。(R/W)

PARL_IO_RX_TIMEOUT_EN 配置是否使能超时功能来生成 DMA ERR EOF。

0：不使能

1：使能

(R/W)

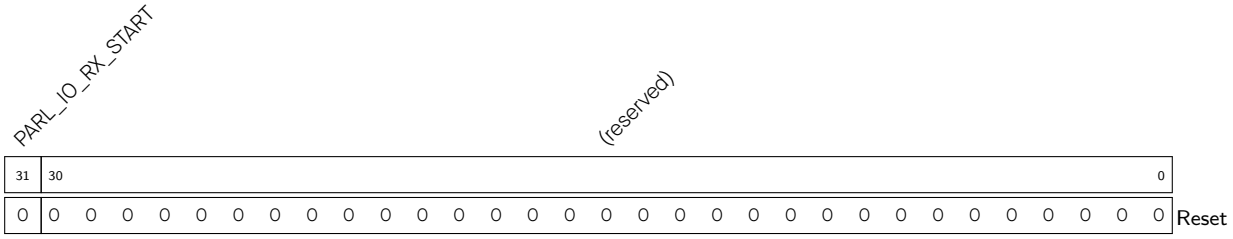
PARL_IO_RX_EOF_GEN_SEL 配置 DMA SUC EOF 的生成模式。

0：通过配置的数据比特长度生成 DMA SUC EOF

1：通过外部使能信号生成 DMA SUC EOF

(R/W)

Register 61.4. PARL_IO_RX_START_CFG_REG (0x000C)



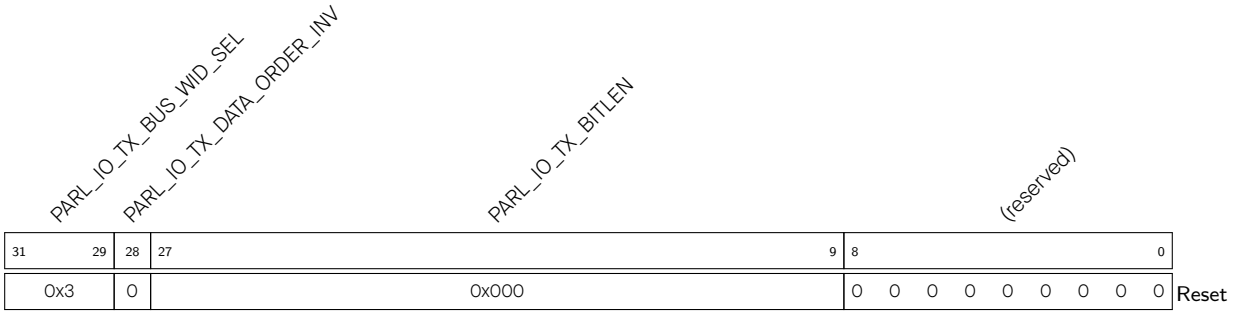
PARL_IO_RX_START 配置是否开始 RX 数据采集。

0：无效

1：开始

(R/W)

Register 61.5. PARL_IO_TX_DATA_CFG_REG (0x0010)



PARL_IO_TX_BITLEN 配置 TXD 的比特长度。(R/W)

PARL_IO_TX_DATA_ORDER_INV 配置是否翻转从 TX_FIFO 发送到 IO 的数据的字节范围内比特顺序。

0：无效

1：翻转数据顺序

(R/W)

PARL_IO_TX_BUS_WID_SEL 配置 TXD 总线位宽。

0：1 位

1：2 位

2：4 位

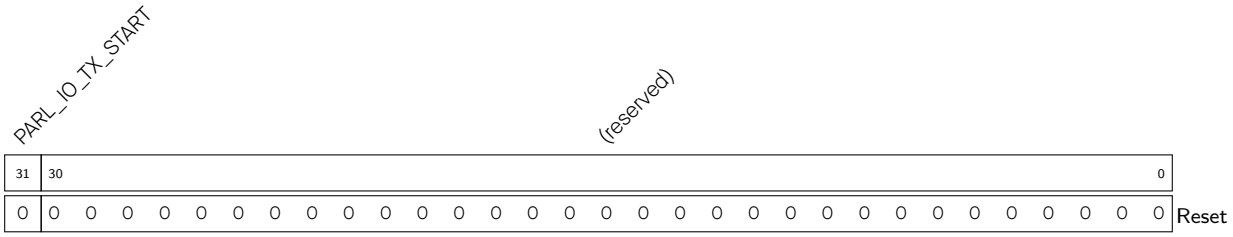
3：8 位

4：16 位

5 ~ 15：无效值，会引发错误

(R/W)

Register 61.6. PARL_IO_TX_START_CFG_REG (0x0014)



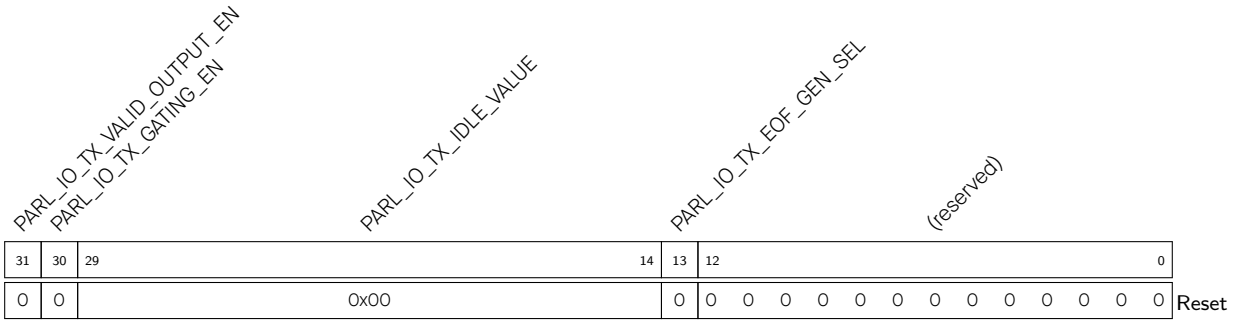
PARL_IO_TX_START 配置是否开始 TX 数据输出。

0：无效

1：开始

(R/W)

Register 61.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)



PARL_IO_TX_EOF_GEN_SEL 配置 TX EOF 的生成模式。

0：通过配置的数据比特长度生成 TX EOF

1：通过 DMA EOF 信号生成 TX EOF

(R/W)

PARL_IO_TX_IDLE_VALUE 配置 TX 总线上在未发送状态下的数据值。(R/W)

PARL_IO_TX_GATING_EN 配置是否使能 TX 输出时钟的时钟门控。

0：不使能

1：使能

(R/W)

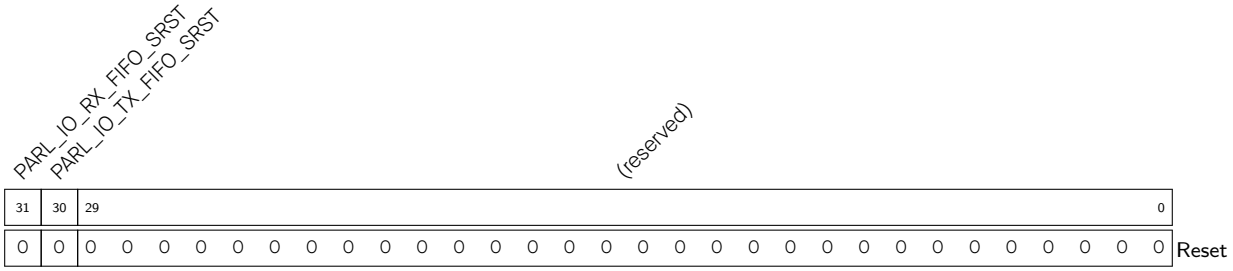
PARL_IO_TX_VALID_OUTPUT_EN 配置使能 DMA 输出数据或者片选信号作为 TXD 最高位控制源。

0：选择 DMA 输出数据为控制源

1：选择片选信号为控制源

(R/W)

Register 61.8. PARL_IO_FIFO_CFG_REG (0x001C)



PARL_IO_TX_FIFO_SRST 配置是否复位 TX 异步 FIFO。

0：无效

1：复位

(R/W)

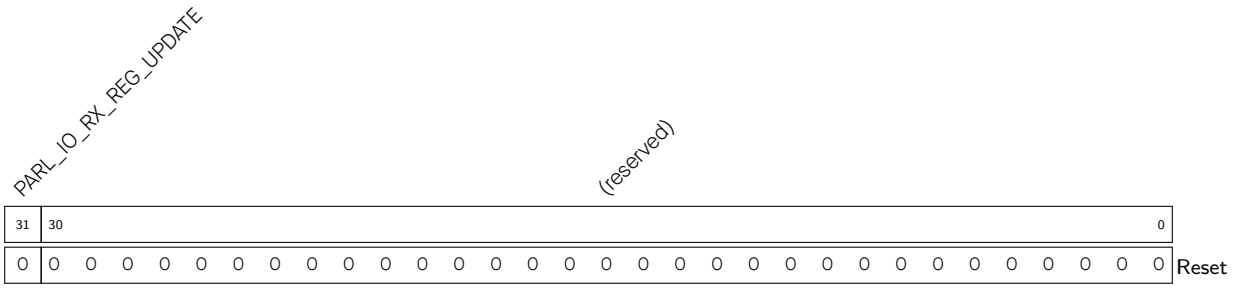
PARL_IO_RX_FIFO_SRST 配置是否复位 RX 异步 FIFO。

0：无效

1：复位

(R/W)

Register 61.9. PARL_IO_REG_UPDATE_REG (0x0020)



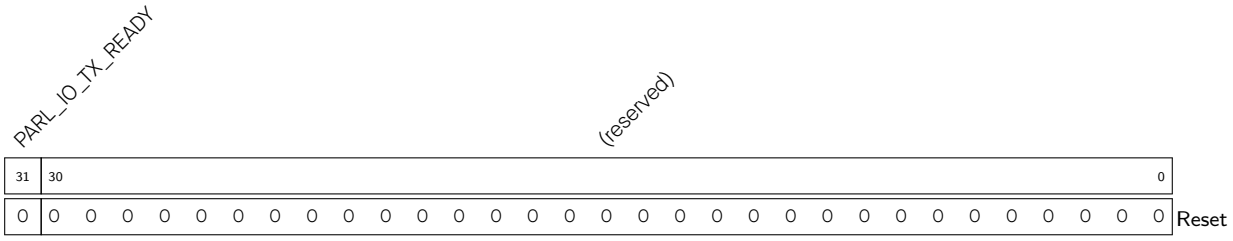
PARL_IO_RX_REG_UPDATE 配置是否更新 RX 寄存器配置。

0：无效

1：更新

(WT)

Register 61.10. PARL_IO_ST_REG (0x0024)



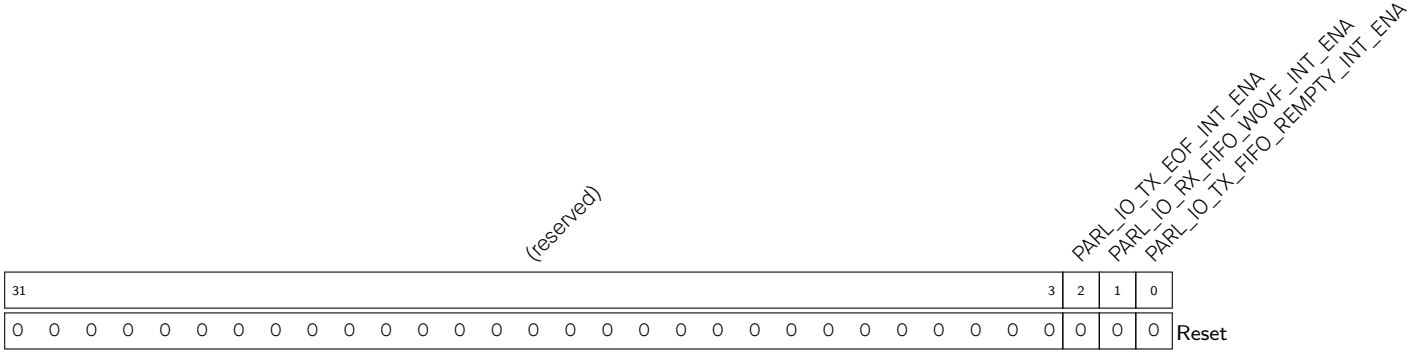
PARL_IO_TX_READY 表示 TX 模块是否已准备好传输数据。

0：未准备好

1：已准备好

(RO)

Register 61.11. PARL_IO_INT_ENA_REG (0x0028)

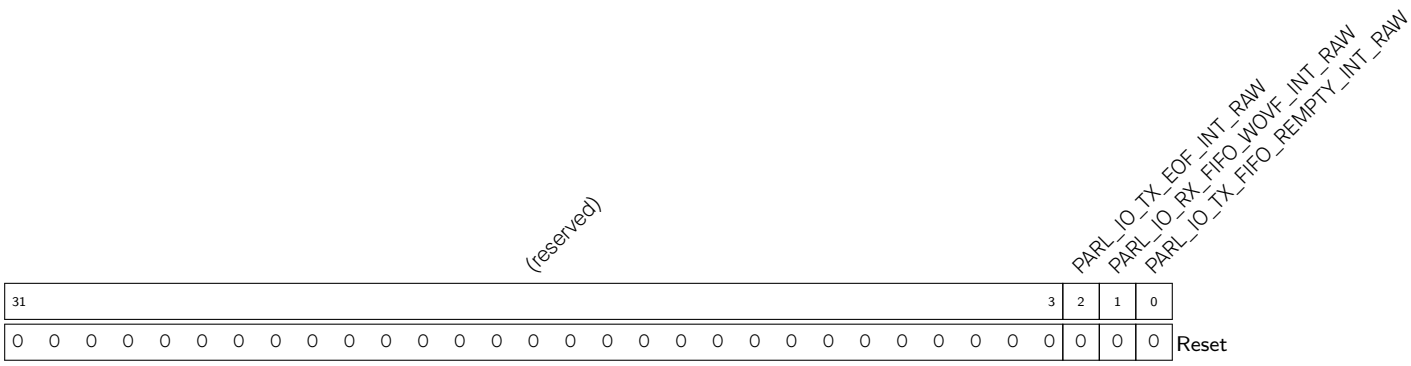


PARL_IO_TX_FIFO_REMPTY_INT_ENA 写 1 使能 TX_FIFO_REMPTY_INT 的屏蔽中断状态。(R/W)

PARL_IO_RX_FIFO_WOVF_INT_ENA 写 1 使能 RX_FIFO_WOVF_INT 的屏蔽中断状态。(R/W)

PARL_IO_TX_EOF_INT_ENA 写 1 使能 TX_EOF_INT 的屏蔽中断状态。(R/W)

Register 61.12. PARL_IO_INT_RAW_REG (0x002C)

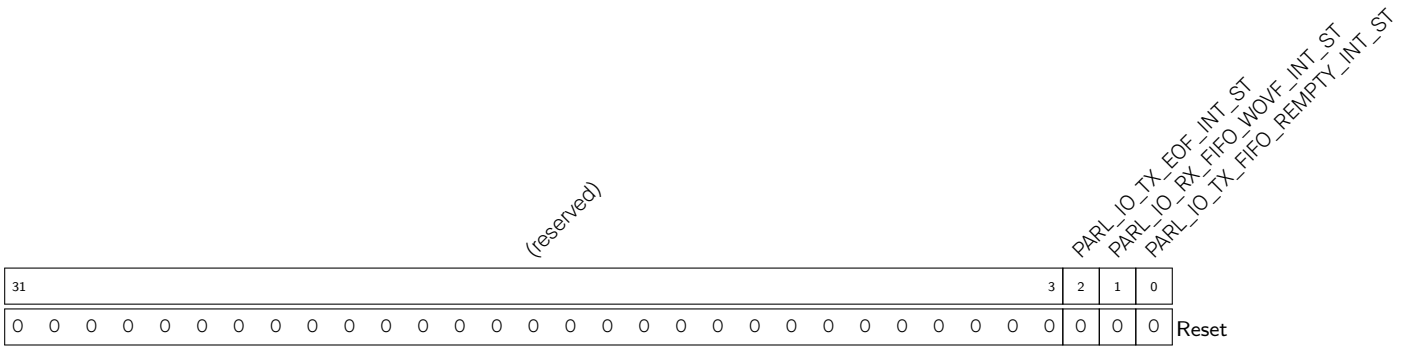


PARL_IO_TX_FIFO_REMPTY_INT_RAW TX_FIFO_REMPTY_INT 的原始中断状态。(R/WTC/SS)

PARL_IO_RX_FIFO_WOVF_INT_RAW RX_FIFO_WOVF_INT 的原始中断状态。(R/WTC/SS)

PARL_IO_TX_EOF_INT_RAW TX_EOF_INT 的原始中断状态。(R/WTC/SS)

Register 61.13. PARL_IO_INT_ST_REG (0x0030)

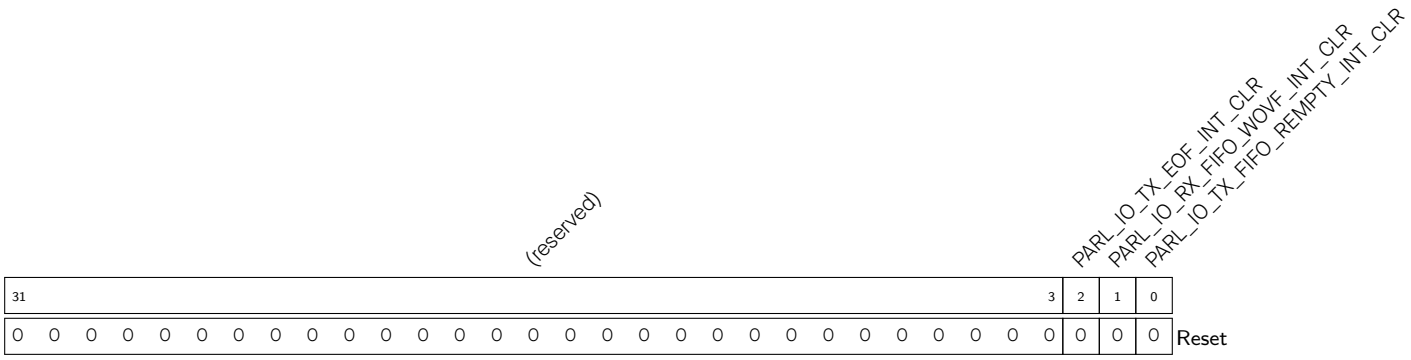


PARL_IO_TX_FIFO_REMPTY_INT_ST TX_FIFO_REMPTY_INT 的屏蔽中断状态。(RO)

PARL_IO_RX_FIFO_WOVF_INT_ST RX_FIFO_WOVF_INT 的屏蔽中断状态。(RO)

PARL_IO_TX_EOF_INT_ST TX_EOF_INT 的屏蔽中断状态。(RO)

Register 61.14. PARL_IO_INT_CLR_REG (0x0034)

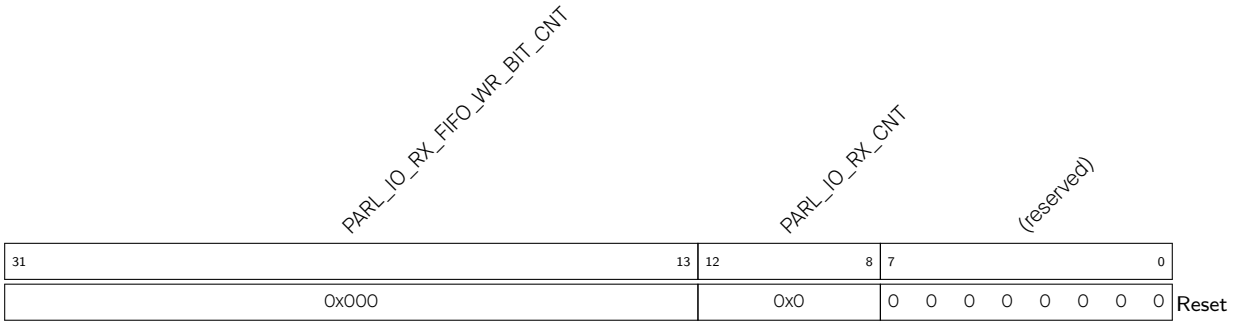


PARL_IO_TX_FIFO_REMPTY_INT_CLR 写 1 清除 TX_FIFO_REMPTY_INT。(WT)

PARL_IO_RX_FIFO_WOVF_INT_CLR 写 1 清除 RX_FIFO_WOVF_INT。(WT)

PARL_IO_TX_EOF_INT_CLR 写 1 清除 TX_EOF_INT。(WT)

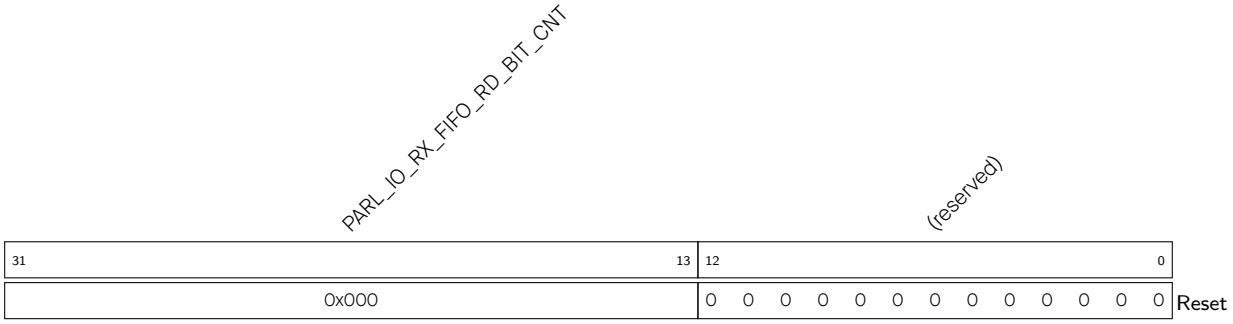
Register 61.15. PARL_IO_RX_ST0_REG (0x0038)



PARL_IO_RX_CNT 表示读取 RX FIFO 的时钟周期数。(RO)

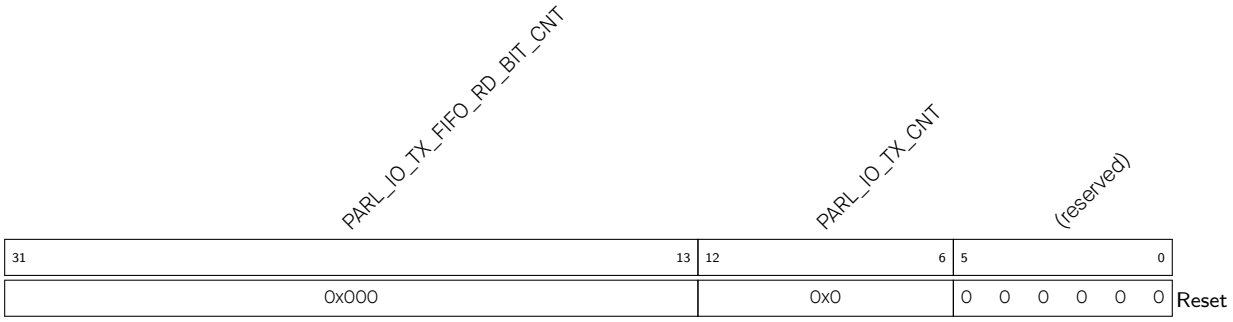
PARL_IO_RX_FIFO_WR_BIT_CNT 表示当前写入 RX FIFO 的位数。(RO)

Register 61.16. PARL_IO_RX_ST1_REG (0x003C)



PARL_IO_RX_FIFO_RD_BIT_CNT 表示当前从 RX FIFO 中读取的位数。(RO)

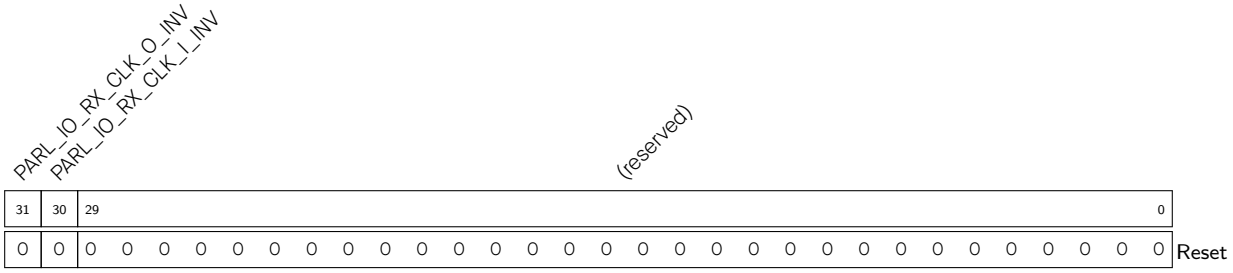
Register 61.17. PARL_IO_TX_ST0_REG (0x0040)



PARL_IO_TX_CNT 表示读取 TX FIFO 的周期数。(RO)

PARL_IO_TX_FIFO_RD_BIT_CNT 表示当前从 TX FIFO 中读取的位数。(RO)

Register 61.18. PARL_IO_RX_CLK_CFG_REG (0x0044)



PARL_IO_RX_CLK_I_INV 配置是否翻转 RX 的输入 Core 时钟。

0：无效

1：翻转

(R/W)

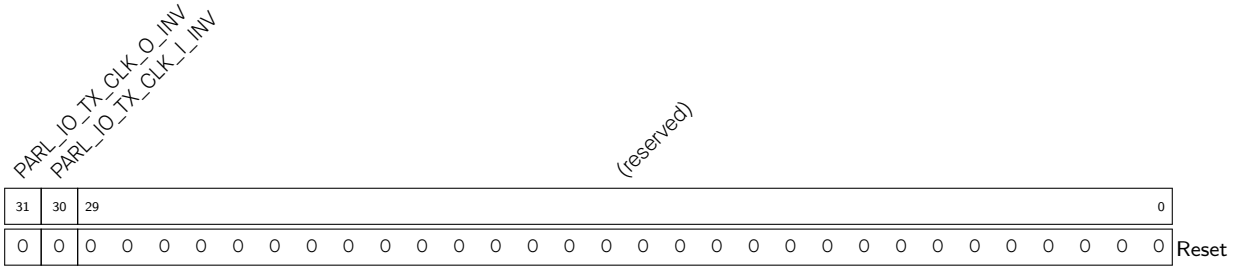
PARL_IO_RX_CLK_O_INV 配置是否翻转 RX 的输出 Core 时钟。

0：无效

1：翻转

(R/W)

Register 61.19. PARL_IO_TX_CLK_CFG_REG (0x0048)



PARL_IO_TX_CLK_I_INV 配置是否翻转 TX 的输入 Core 时钟。

0：无效

1：翻转

(R/W)

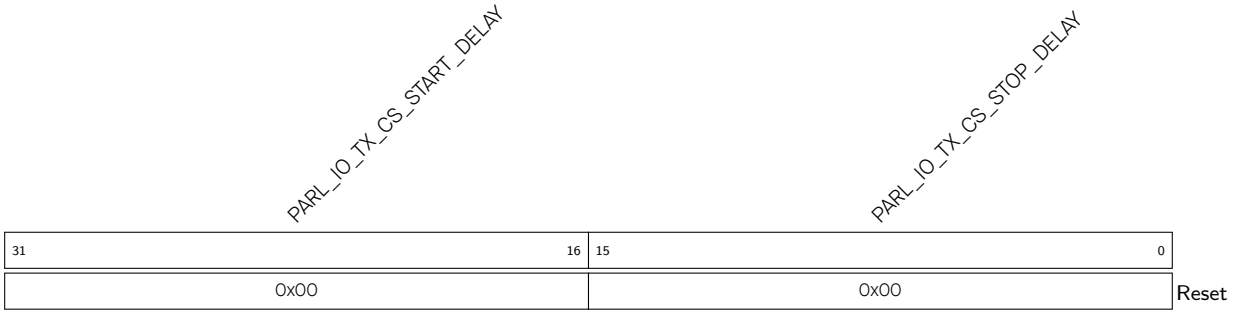
PARL_IO_TX_CLK_O_INV 配置是否翻转 TX 的输出 Core 时钟。

0：无效

1：翻转

(R/W)

Register 61.20. PARL_IO_TX_CS_CFG_REG (0x004C)



PARL_IO_TX_CS_STOP_DELAY 配置 TX 数据发送结束与 TX_CS_O 上升沿之间的延迟。

单位：TX Core 时钟周期

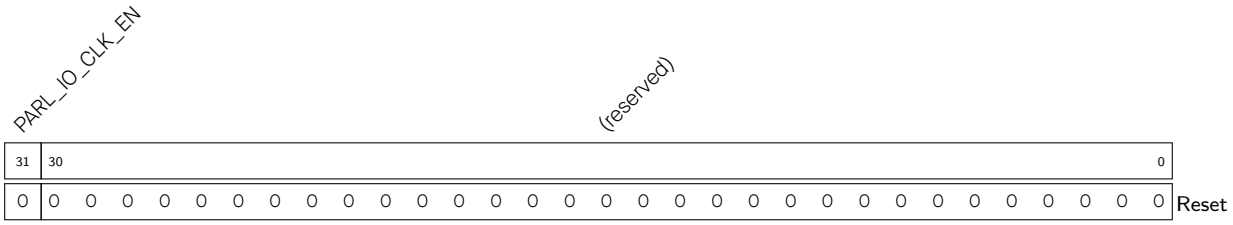
(R/W)

PARL_IO_TX_CS_START_DELAY 配置 TX_CS_O 下降沿与 TX 数据开始传输之间的延迟。

单位：TX Core 时钟周期

(R/W)

Register 61.21. PARL_IO_CLK_REG (0x0120)



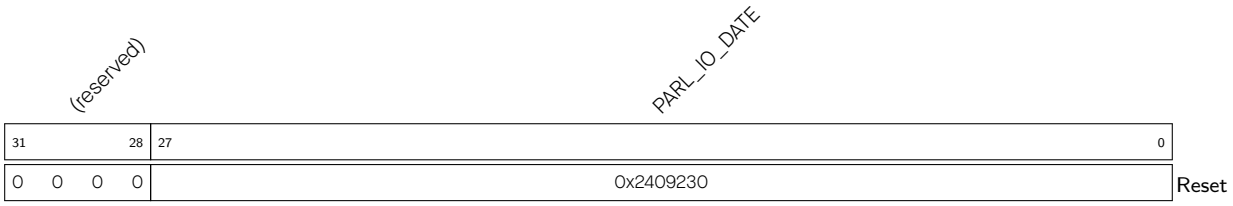
PARL_IO_CLK_EN 寄存器时钟门控。

0：只在读写寄存器时开启寄存器时钟

1：始终开启寄存器时钟

(R/W)

Register 61.22. PARL_IO_VERSION_REG (0x03FC)



PARL_IO_DATE 版本控制寄存器。(R/W)

第 62 章

比特调节器

ESP32-P4 中有大量支持 DMA（直接存储器访问）的外设，它们可以在 CPU 不参与的情况下将数据从存储器传输到外设或从外设传输到存储器，但这需要外设传输的数据格式与软件支持的数据格式相同，如果格式不同，则需要 CPU 重写数据格式，如交换字节、反转字节和左右移位数据。

由于位操作通常相当耗费 CPU 资源，而设计 DMA 的初衷是在传输过程中避免使用 CPU，因此 ESP32-P4 集成了两个比特调节器 (BitScrambler)，专门用于修改存储器和外设之间传输数据的格式，一个传输控制器用于存储器到外设（或存储器到存储器）方向的传输，另一个传输控制器用于外设到存储器方向的传输。除此之外，比特调节器还是一个灵活的可编程状态机，能够执行更高级的操作。

62.1 概述

在 ESP32-P4 中，支持 DMA 的外设是指能够通过 DMA 通道直接从存储器读取数据和/或直接向存储器写入数据的外设。这些数据的读取/写入格式取决于外设，灵活性较低。对于 ESP32-P4 中大多数支持 DMA 的外设来说，可以在这些路径中挂载比特调节器来提高灵活性。

当某个外设挂载比特调节器后，比特传输器就可以修改与该外设连接的 DMA 通道上的任何数据。写入 DMA 通道的数据会出现在比特调节器的输入端，而比特调节器输出端的数据将沿着 DMA 通道继续传输。比特调节器内置指令存储器，存储的指令用于控制输入和输出数据格式的映射关系。因此，通过编写合适的比特调节器程序，可以让比特调节器改写源数据。

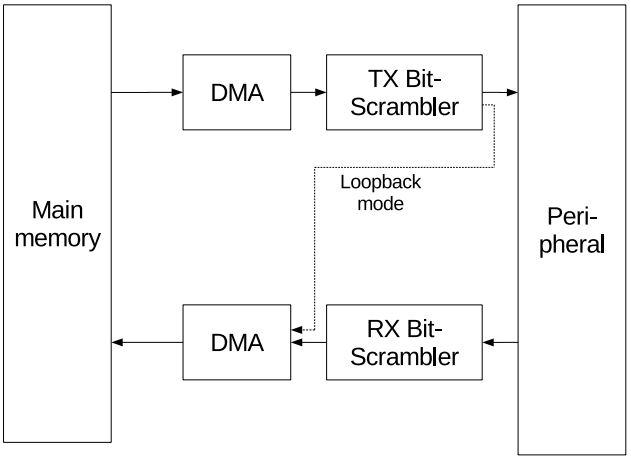


图 62.1-1. 比特调节器系统架构图

62.2 特性

比特调节器有以下特性：

- 两个比特调节器，一个用于 RX（外设到存储器），一个用于 TX（存储器到外设）
- 支持存储器到存储器的传输
- 每个 DMA 时钟周期最多可处理 32 位数据
- 数据路径由存储在指令存储器中的比特调节器程序控制
- 输入寄存器每个时钟周期可读取 0、8、16 或 32 位
- 输出寄存器：
  - 每个时钟周期可写入 0、8、16 或 32 位
  - 输出寄存器位的数据源：64 位输入数据、两个计数器、LUT RAM 数据、上个周期的数据输出、比较器
  - 32 位输出寄存器位中的每一位可以来自数据源的任意位
- 8 x 257 位指令存储器，用于存储八条指令，配置控制流和数据路径
- 2048 字节查找表（LUT）存储器，可配置为不同的字宽

62.3 架构

比特调节器可以分为数据路径、控制路径以及寄存器，CPU 可通过这些寄存器配置数据路径和控制路径。数据路径在 DMA 输入流、内部寄存器和 DMA 输出流之间传输比特。控制路径解析指令存储器中的指令，以控制数据路径，并处理程序流程。

62.3.1 数据路径

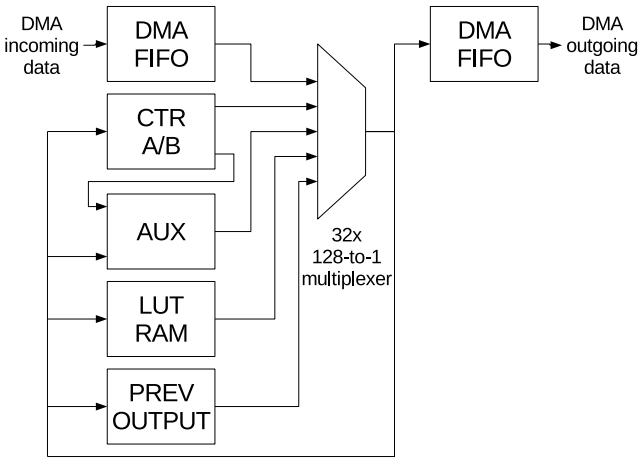


图 62.3-1. 比特调节器数据路径图

比特调节器的数据路径从 DMA 输入流中获取数据，根据存储在指令存储器中的程序代码处理数据，然后将其写入 DMA 输出流中。数据从 DMA 输入流保存到一个 64 位寄存器中。在每个指令周期结束时，根据指令的配置从 DMA 输入流中读取 N 位（N 可以是 0、8、16 或 32）。然后，寄存器中的数据向寄存器的最低有效位（LSB）方向移动，最低有效位的 N 位消失。最后，将最新读取的数据保存在最高的 N 位。在比特调节器启动时，可以选择预先自动读取前 64 位到寄存器中。

在比特调节器的另一端，数据存入一个 32 位输出寄存器。在每个指令周期结束时，根据指令的配置将最低的 0、8、16 或 32 位写入 DMA 输出流。

比特调节器的名称来源于其可以将输入比特放置在输出中的任意位置，这是通过 32 个 128 选 1 多路复用器 (128-to-1 multiplexer) 来实现的。对于输出寄存器中的 32 位的每一位，都有一个多路复用器根据比特调节器控制路径的输出，从相应的数据源中选择输入信号。这些数据源包括：

- 从输入 FIFO 移入的 64 位数据（即图 62.3-1 中左侧的 DMA FIFO 模块）。如前所述，来自 DMA 输入流的数据被存储在此处。为了便于使用循环来遍历输入的 64 位数据，指令可以启用相对寻址。在这种寻址模式下，任何从输入 FIFO 寄存器获取比特的多路复用器实际上会获取偏移计数器 A 寄存器值的比特（即图 62.3-1 中的 CTR A）。
- 从上一个周期发送到输出的 32 位数据（即图 62.3-1 中的 PREV OUTPUT 模块）。发送到输出寄存器的数据将在一个时钟周期后出现在这个寄存器中。这使得比特调节器可以在多个时钟周期中“记住”数据：即使某个比特没有发送到输出 FIFO，只要是输出到了输出寄存器，下一个时钟周期仍然可以通过选择这个寄存器来访问该比特。通过再次从这个寄存器中选择比特进行输出，可以实现比特的长期记忆，使它们在随后的周期中出现在此寄存器中。
- 由 LUT RAM 输出的 8、16 或 32 位数据。当 LUT 宽度为 32 位、16 位、8 位时，则数据地址分别是上一周期输出数据的第 16 到 24、25、26 位。LUT RAM 是比特调节器的一部分，它首先接收到某个地址，然后在 LUT RAM 中查找该地址对应的数据，最后输出该数据。比特调节器初始化时可填充 LUT RAM，但比特调节器不能修改 LUT RAM 。可根据需求将 LUT RAM 配置成以下模式进行初始化：
  - 512 x 32 位字
  - 1024 x 16 位字
  - 2048 x 8 位字

  LUT RAM 可以用于数据的转换，例如用于某传感器的校准曲线，也可用于实现数学函数：将地址输入拆分成两个或两个以上的输入变量，数据输出作为函数的输出；存储器应加载每个输入值对应的函数输出。
- 2 x 16 位计数器提供的 32 位数据（即图 62.3-1 中的 CTR A/B 模块）。该寄存器包含两个 16 位计数器，分别命名为计数器 A 和 B，可通过控制路径指令对它们进行加载、递增和递减等操作。
- 计数器 B 和之前输出数据（对应图 62.3-1 中的 PREV OUT）之间的比较器中包含的 30 位数据，属于 AUX 模块的一部分比较结果可用于输出或程序流程，例如在解码游程编码流 (run-length encoded stream) 或生成 PWM 信号时非常有用。
- 2 位数据（属于图 62.3-1 中的 AUX 模块），一位固定为高电平，一位固定为低电平。使用场景：某协议需要输出一个始终为高电平或低电平的位，不受输入数据影响。

请注意，某些数据源无法同时访问。具体而言，指令无法同时从输入 FIFO 源寄存器的高 32 位以及计数器寄存器中获取数据。当从 LUT 读取数据时，对于宽度为 N 的 LUT，输入 FIFO 寄存器的最高 N 位和计数器寄存器的最高 N 位不可用。

62.3.2 控制路径

比特调节器的行为由存储在程序 RAM 中的程序控制。程序 RAM 最多可以存储 8 条指令，每条指令为 257 位。这些指令可配置该指令运行时钟周期内的数据路径，以及控制程序流程的操作码。

比特调节器的程序和微处理器的程序类似，每个周期都会执行指令存储器中的下一条指令，除非操作码明确指示需要跳转到另一个位置。具体来说，这种行为是使用指令指针（IP，参见图 62.3-2）来实现的，指向当前正在执行的指令。除此之外，操作码还可以影响计数器寄存器。

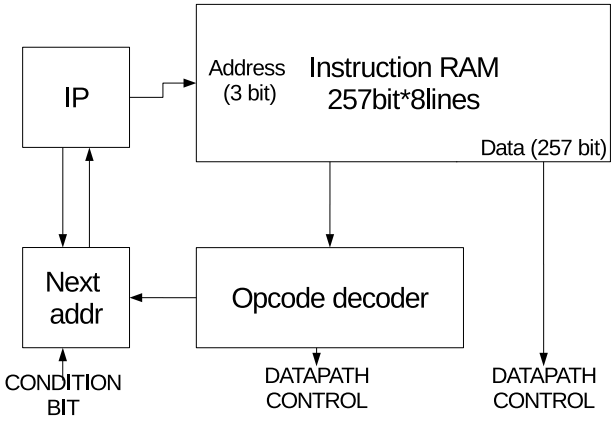


图 62.3-2. 比特调节器控制路径图

在配置数据路径的指令位中，一些位用于配置 32 个多路复用器，另一些位用于配置计数器位或 LUT 输出是否可用。指令还可以将多路复用器设置为相对模式，在这种模式下，从 DMA FIFO 获取的比特的位置将被偏移计数器 B 位。

62.4 功能描述

比特调节器是一种灵活的设备，其行为可通过加载到控制器内部的程序来控制。此外，一些配置寄存器也可以全局配置控制器的一些行为。因此，比特调节器程序包括 257 位 x 8 的指令 RAM 以及各种配置寄存器的设置，下文将对二者进行介绍。

62.4.1 指令

表 62.4-1 为 257 位指令的结构。

表 62.4-1. 指令结构

位	域	位	域
0 - 6	CTL_MUX_SEL_0	133 - 139	CTL_MUX_SEL_19
7 - 13	CTL_MUX_SEL_1	140 - 146	CTL_MUX_SEL_20
14 - 20	CTL_MUX_SEL_2	147 - 153	CTL_MUX_SEL_21
21 - 27	CTL_MUX_SEL_3	154 - 160	CTL_MUX_SEL_22
28 - 34	CTL_MUX_SEL_4	161 - 167	CTL_MUX_SEL_23
35 - 41	CTL_MUX_SEL_5	168 - 174	CTL_MUX_SEL_24
42 - 48	CTL_MUX_SEL_6	175 - 181	CTL_MUX_SEL_25
49 - 55	CTL_MUX_SEL_7	182 - 188	CTL_MUX_SEL_26
56 - 62	CTL_MUX_SEL_8	189 - 195	CTL_MUX_SEL_27
63 - 69	CTL_MUX_SEL_9	196 - 202	CTL_MUX_SEL_28
70 - 76	CTL_MUX_SEL_10	203 - 209	CTL_MUX_SEL_29
77 - 83	CTL_MUX_SEL_11	210 - 216	CTL_MUX_SEL_30
84 - 90	CTL_MUX_SEL_12	217 - 223	CTL_MUX_SEL_31
91 - 97	CTL_MUX_SEL_13	224 - 249	OPCODE
98 - 104	CTL_MUX_SEL_14	250 - 251	CTL_READ_IN
105 - 111	CTL_MUX_SEL_15	252 - 253	CTL_WR_OUT
112 - 118	CTL_MUX_SEL_16	254	CTL_MUX_REL
119 - 125	CTL_MUX_SEL_17	255	CTL_CTR_SRC_SEL
126 - 132	CTL_MUX_SEL_18	256	CTL_LUT_SRC_SEL

下文介绍指令各域的含义。

- CTL_MUX_SEL_n：包含 7 位，选择输出寄存器中第 n 位的来源。详情参见表 62.4-3。
- OPCODE：操作码，该字段的位格式请参见表 62.4-4。
- CTL_READ_IN：在指令结束时选择从输入 FIFO 读取的位数。
  - 0：不读取任何数据
  - 1：读取 8 位数据
  - 2：读取 16 位数据
  - 3：读取 32 位数据
- CTL_WR_OUT：选择从输出寄存器写入到输出 FIFO 的位数：
  - 0：不写入任何数据
  - 1：写入 8 位数据
  - 2：写入 16 位数据
  - 3：写入 32 位数据
- CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL：配置哪些位可以通过 CTL_MUX_SEL 域来选择。
  - CTRL_CTR_SRC_SEL（控制计数器源选择）配置计数器 A 和 B 的位是否可用。

- CTL_LUT_SRC_SEL（控制 LUT 源选择）配置 LUT RAM 的输出值是否可用。详见表 62.4-3。
- 当 CTL_MUX_REL（控制多路复用器相对寻址）为 1 时，且 CTL_MUX_SEL_n 小于等于 63 时，比特调节器将使用计数器 A 重新计算有效的 CTL_MUX_SEL_n。具体行为取决于 CTR_SRC_SEL，详见表 62.4-2。

表 62.4-2. 当 CTL_MUX_REL 为 1 时 CTL_MUX_SEL_n 生效的值

条件	CTL_MUX_SEL 生效的值
CTL_MUX_SEL_n >= 64	CTL_MUX_SEL（不变）
CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 0	(CTL_MUX_SEL_n + CTRA) & 63
CTL_MUX_SEL_n < 31 and CTL_CTR_SRC_SEL = 1	(CTL_MUX_SEL_n + CTRA) & 31
32 <= CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 1	((CTL_MUX_SEL_n + CTRA) & 31) + 32

这些域中，大部分是数据路径配置项，其中 OPCODE 是影响控制路径的配置项，但 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 可通过调整 IF/IFN 操作码的来源来影响某些操作码的工作方式。

CTL_MUX_SEL_n: 每个 CTL_MUX_SEL_n 域都可用于选择输出寄存器中与之相对应的位的来源。具体选择哪个位在一定程度上取决于 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置。假设 CTL_MUX_SRC 的值为 N，选择该位的方式如表 62.4-3 所示。请注意，IF/IFN 操作码的数据源选择也采用相同的方式。

表 62.4-3. CTL_MUX_SEL 的值

位	描述
0-31	该位来自输入 FIFO 寄存器中的第 N 位。
32-63	该位的来源取决于 CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置 如果 CTL_CTR_SRC_SEL = 0，该位来源于输入 FIFO 寄存器中的第 N 位。 如果 CTL_CTR_SRC_SEL = 1，该位来源于计数器寄存器中的第 N-32 位。具体来说，如果 N 是 32 到 47，则该位来源于计数器 A 的第 N-32 位；如果 N 是 48 到 63，则该位来源于计数器 B 的第 N-48 位。 如果 CTL_LUT_SRC_SEL = 1，根据 LUT 的宽度（8、16 或 32 位），读取最高 8、16 或 32 位将返回来自 LUT 的数据，而不是 CTL_CTR_SRC_SEL 选择的数据。具体来说，假定 LUT 宽度为 N，63 将返回 LUT 输出的第 N-1 位，62 将返回 LUT 输出的第 N-2 位，依此类推。
64	CounterB[7:0] =< last[7:0]
65	CounterB[7:0] > last[7:0]
66	CounterB[7:0] = last[7:0]
67	CounterB[7:0] =< last[15:8]
68	CounterB[7:0] > last[15:8]
69	CounterB[7:0] = last[15:8]
70	CounterB[7:0] =< last[23:16]
71	CounterB[7:0] > last[23:16]
72	CounterB[7:0] = last[23:16]
73	CounterB[7:0] =< last[31:24]
74	CounterB[7:0] > last[31:24]
75	CounterB[7:0] = last[31:24]
76	CounterB[15:8] =< last[7:0]
77	CounterB[15:8] > last[7:0]
78	CounterB[15:8] = last[7:0]
79	CounterB[15:8] =< last[15:8]
80	CounterB[15:8] > last[15:8]
81	CounterB[15:8] = last[15:8]
82	CounterB[15:8] =< last[23:16]
83	CounterB[15:8] > last[23:16]
84	CounterB[15:8] = last[23:16]
85	CounterB[15:8] =< last[31:24]
86	CounterB[15:8] > last[31:24]
87	CounterB[15:8] = last[31:24]
88	CounterB[15:0] =< last[15:0]
89	CounterB[15:0] > last[15:0]
90	CounterB[15:0] = last[15:0]
91	CounterB[15:0] =< last[31:16]
92	CounterB[15:0] > last[31:16]
93	CounterB[15:0] = last[31:16]
94	始终为 0
95	始终为 1
96-127	这些位与上一周期结束时输出寄存器上的位相同

比特调节器可识别六种操作码，编码格式如表 62.4-4 所示。

表 62.4-4. 指令操作码

Bit	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOOP	1	c	tgt			end_val																ctr_add				
ADD	0	c	h	l	0	0	0	0	0	0	ctr_add															
IF	0	0	tgt			0	0	0	0	1	0	0	0	0	0	0	0	0	0	src						
IFN	0	0	tgt			0	0	0	1	0	0	0	0	0	0	0	0	0	0	src						
LDCTD	0	c	h	l	0	0	0	0	1	1	ctr_set															
LDCTI	0	c	h	l	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

操作码字段的定义如下：

- c：配置使用哪个计数器。
  - 0：计数器 A
  - 1：计数器 B
- end_val, ctr_add, ctr_set：16 位（或 5 位）值。这些字段的含义请参阅下面的指令描述。
- tgt：指令的位置。范围：0 到 7。
- h：如果需要将高 8 位写回计数器，则为 1。
- l：如果需要将低 8 位写回计数器，则为 1。

请注意，如果操作码没有指定 H 或 L，则默认需要将 16 位全部写回，此时 h 和 l 位都会为 1。

下方表格主要介绍这六个操作码如何影响比特调节器的状态。

表 62.4-5. LOOP 操作码

`LOOPc end_val, ctr_add, tgt`	
使用 c 参数设置的计数器进行循环，将 ctr_add 加到计数器中，并循环回到 tgt 位置，直到计数器达到 end_val。当计数器达到 end_val 时，重置计数器并继续执行。	
参数	c：计数器 A 或 B，表示要使用的计数器 tgt：3 位值，表示目标指令地址 end_val：16 位值，用于与计数器进行比较 ctr_add：5 位有符号值，用于加到计数器上

伪代码	`if ((unsigned)ctrC < end_val) begin`
	`  IP <= tgt`
	`  ctr_c <= ctr_c + sign_extend(ctr_add)`
	`end else begin`
	`  IP <= IP + 1`
	`  ctr_c <= 0`
	`end`

表 62.4-6. ADD 操作码

ADDc[H|L] ctr_add	
将立即数加到计数器上，可以选择只写回高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 ctr_add：16 位值，加到计数器上的值 如果操作码是 ADDcH，则只写回最高 8 位 如果操作码是 ADDcL，则只写回最低 8 位 如果操作码是 ADDc，则 16 位全部都将被写回（h=1，l=1）

伪代码	`tmp <= ctr_c + ctr_add`
	`if (h) begin`
	`  ctr_c[15:8] <= tmp[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= tmp[7:0]`
	`end`
	`IP <= IP + 1`

说明	NOP（无操作）伪操作码为 “ADDA 0”。

表 62.4-7. IF 操作码

IF ctl_cond_src, tgt	
如果 src 字段对应的多路复用器值为 1，则跳转到目标。	
参数	ctl_cond_src：多路复用器来源，可参考表 CTR_MUX_SRC sources tgt：3 位值，表示目标指令地址

伪代码	`if (ctl_cond_src) begin`
	`  IP <= tgt`
	`end else begin`
	`  IP <= IP + 1`
	`end`

说明	JMP（始终跳转）伪操作码为 “IF 95,tgt”，使用 AUX 位中始终为 1 的位。

表 62.4-8. IFN 操作码

IFN ctl_cond_src, tgt	
如果指定的多路复用器源位未设置，则跳转到目标（注：与 IF 操作相同，但条件取反）。	
参数	ctl_cond_src：多路复用器来源，可参考表 CTR_MUX_SRC sources tgt：3 位值，表示目标指令地址

伪代码	`if (ctl_cond_src) begin`
	`  IP <= IP + 1`
	`end else begin`
	`  IP <= tgt`
	`end`

表 62.4-9. LDCTD 操作码

LDCTDc[H|L] ctr_set	
将立即数直接加载到计数器，可以选择只加载计数器的高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 ctr_set：16 位值，要设置的计数器的值 h：如果操作码是 LDCTDcH，则只设置最高 8 位 l：如果操作码是 LDCTDcL，则只设置最低 8 位 如果操作码是 LDCTDc，则 16 位全部都将被设置（h=1，l=1）

伪代码	`if (h) begin`
	`  ctr_c[15:8] <= ctr_set[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= ctr_set[7:0]`
	`end`
	`IP <= IP + 1`

表 62.4-10. LDCTI 操作码

LDCTIc[H|L]	
将多路复用器输出的高 16 位间接加载到计数器，可以选择只加载到计数器的高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 h：如果操作码是 LDCTIcH，则只设置最高 8 位 l：如果操作码是 LDCTIcL，则只设置最低 8 位 如果操作码是 LDCTIc，则 16 位全部都将被设置（h=1，l=1）

伪代码	`if (h) begin`
	`  ctr_c[15:8] <= mux_out[31:24]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= mux_out[23:16]`
	`end`
	`IP <= IP + 1`

62.4.2 配置寄存器

比特调节器的配置是通过一组寄存器实现的。这些寄存器可向比特调节器写入程序、写入 LUT RAM、配置控制器行为、以及启动和停止控制器的操作，还可将控制器挂载到外设的 DMA 路径。

ESP32-P4 有两个比特调节器，一个可以放置在 TX 路径中（数据从存储器发送到外设），另一个可以放置在 RX 路径中（数据从外设发送到存储器）。两个比特调节器的配置寄存器是相同的，唯一的区别在于 TX 路径的比特调节器寄存器前缀为 BITSCRAMBLER_TX_，而 RX 路径的为 BITSCRAMBLER_RX_。

说明：
为方便阐述，本节仅介绍 TX 比特调节器路径，若想了解 RX 路径，只需替换描述中的前缀即可。

当比特调节器修改某外设的输入或输出时，需要挂载到该外设的 DMA 路径。为此，需将该外设的对应值写入

HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG 寄存器的 HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 域或 HP_SYSTEM_BITSCRAMBLER_PERI_RX_SEL 域。

比特调节器也可以在存储器到存储器模式下运行，但需要将 BITSCRAMBLER_SYS_REG 寄存器中的 BITSCRAMBLER_LOOP_MODE 域置为 1。该模式下，TX 比特调节器进入环回模式，而 RX 比特调节器则未使用且无法使用。并且比特调节器仍然需要连接到外设，但该外设无需配置，而且对应的外设 DMA 路径将无法使用。

主处理器无法直接访问比特调节器的程序存储器和 LUT RAM，但可以间接访问，即在一个寄存器中设置地址，然后在另一个寄存器中写入数据。

具体而言，对于 LUT RAM，地址写入 BITSCRAMBLER_TX_LUT_CFG0_REG 的 BITSCRAMBLER_TX_LUT_IDX 域，数据写入或从 BITSCRAMBLER_TX_LUT_CFG1_REG 寄存器读取数据。LUT 的位宽对比特调节器可见的，可以使用 BITSCRAMBLER_TX_LUT_MODE 域进行配置。需要注意的是，当主处理器以上述方式写入 LUT 时，地址和数据大小需要符合配置的 LUT 位宽。

同样，对于指令存储器，地址写入 BITSCRAMBLER_TX_INST_CFG0_REG，数据写入或从 BITSCRAMBLER_TX_INST_CFG1_REG 中读取。由于比特调节器指令为 257 位，因此一个指令需要写入 9 个 32 位字，第 257 位位于第 9 个字的最低有效位。为此，BITSCRAMBLER_TX_INST_CFG0_REG 寄存器被分为两个域，指令的位置写入 BITSCRAMBLER_TX_INST_IDX 域，而指令内的字偏移写入 BITSCRAMBLER_TX_INST_POS 域。

由于比特调节器生成的数据可以比输入多也可以比输入少，所以它控制的 DMA 流可以以一种或两种方式结束。第一种情况是接收器（在 RX 比特调节器的情况下是存储器，而在 TX 比特调节器的情况下是外设）停止接收数据，因为它的配置是只接收一定数量的字节。这种情况比较简单，比特调节器因为无法再写入数据而停止。另一种方式是发射器（在 RX 比特调节器的情况下是外设，在 TX 比特调节器的情况下是存储器）停止发送数据。这种情况更加复杂，因为比特调节器可能正在处理需要写入输出的一些数据，也可能不在处理。

ESP32-P4 特调节器设计了两种方法来处理输入数据流结束后的一定数量的数据，目的是为了完整地处理这些数据。这两种方法依赖于 BITSCRAMBLER_RX_TAILING_BITS_REG 寄存器设置的具体值 N：

- EOF-on-N-reads（读取 N 字节后结束）：在输入数据流结束后，比特调节器从输入读取 N 个虚拟字节。这些字节的值为零。在读取完第 N 个字节后，比特调节器停止，DMA 流结束。
- EOF-on-N-writes（写入 N 字节后结束）：在输入数据流结束后，比特调节器允许向输出写入 N 个字节。在此期间，任何输入流上的读取都会返回值为零的虚拟字节。在写入完第 N 个字节后，比特调节器停止，DMA 流结束。

请按照下表内容配置上述两种模式。

表 62.4-11. EOF 模式设置

寄存器	EOF-on-N-reads	EOF-on-N-writes
BITSCRAMBLER_TX_RD_DUMMY	1	1
BITSCRAMBLER_TX_FETCH_MODE	0	0
BITSCRAMBLER_TX_EOF_MODE	1	0
BITSCRAMBLER_TX_HALT_MODE	1	1

62.5 配置流程

请注意，下文描述的是 TX 比特调节器的配置流程，TX 比特调节器位于存储器到外设方向的路径中，或者在环回模式下位于存储器到存储器方向的路径。对于外设到存储器方向的路径，则使用 RX 比特调节器。除非另作说明，否则将下文描述中的 TX 替换为 RX 即为 RX 比特调节器的配置流程。

1. 配置 HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG 寄存器中的 HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 域将比特调节器挂载到外设。请注意，即使在环回模式下，此步骤也是必须的。
2. 配置 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_ENA 域使能比特调节器。
3. 根据需要决定是否配置环回模式。如果需要，将 BITSCRAMBLER_SYS_REG 寄存器的 BITSCRAMBLER_LOOP_MODE 域置 1，否则清零。
4. 将程序加载到比特调节器中。对于每条指令和指令内的每个 32 位字，设置 BITSCRAMBLER_TX_INST_CFG0_REG 寄存器中的 BITSCRAMBLER_TX_INST_IDX 和 BITSCRAMBLER_TX_INST_POS 域，然后将字写入 BITSCRAMBLER_TX_INST_CFG1_REG 寄存器。注意，第 257 位写入到 BITSCRAMBLER_TX_INST_CFG1_REG 的第 0 位。
5. 如果需要，将 LUT 挂载到比特调节器中：
   (a) 配置 BITSCRAMBLER_TX_LUT_CFG0_REG 寄存器中 BITSCRAMBLER_TX_LUT_MODE 域来设置 LUT 宽度
   (b) 将地址写入 BITSCRAMBLER_TX_LUT_CFG0_REG
   (c) 将数据字或字节写入 BITSCRAMBLER_TX_LUT_CFG1_REG

   请注意，在写入数据之后和启动比特调节器之前更改 LUT 宽度都是可以的。
6. 配置比特调节器。根据比特调节器程序的需求，设置或清除 BITSCRAMBLER_TX_CTRL_REG 寄存器中的相应位，也可将 BITSCRAMBLER_TX_TAILING_BITS_REG 设置为适当的值。
7. 清除 BITSCRAMBLER_TX_CTRL_REG 寄存器的BITSCRAMBLER_TX_HALT 域，以启动比特调节器。
8. 启动 DMA 传输。有关 DMA 子系统的更多信息，请参阅章节 *6 通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*，如果未使用环回模式，请参阅各外设章节。
9. 等待 DMA 传输完成。
10. 设置 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_HALT 域停止比特调节器。
11. 等待比特调节器处于停止状态。当 BITSCRAMBLER_TX_STATE_REG 寄存器中的 BITSCRAMBLER_TX_IN_IDLE 位被置位时，比特调节器停止。
12. 在 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_FIFO_RST 位先写入 1 然后再写入 0 来复位 FIFO。
13. 比特调节器可以使用当前的程序和 LUT 配置开始进行新的传输，从步骤 7 开始即可。

62.6 寄存器列表

本小节的所有地址均为相对于比特调节器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

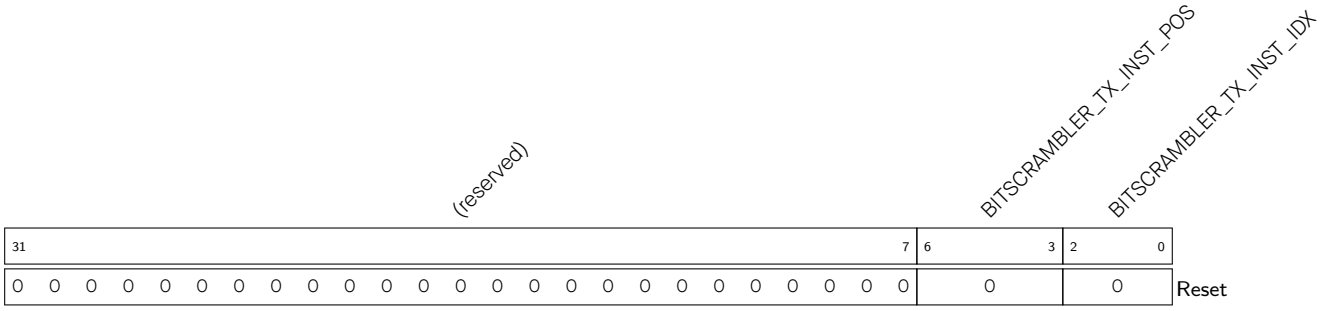
名称	描述	地址	访问
控制和配置寄存器			
BITSCRAMBLER_TX_INST_CFG0_REG	比特调节器 TX 内核指令存储器地址选择寄存器	0x0000	R/W
BITSCRAMBLER_TX_INST_CFG1_REG	比特调节器 TX 内核指令存储器访问寄存器	0x0004	R/W
BITSCRAMBLER_RX_INST_CFG0_REG	比特调节器 RX 内核指令存储器地址选择寄存器	0x0008	R/W
BITSCRAMBLER_RX_INST_CFG1_REG	比特调节器 RX 内核指令存储器访问寄存器	0x000C	R/W
BITSCRAMBLER_TX_LUT_CFG0_REG	比特调节器 TX 内核 LUT 存储器地址选择寄存器	0x0010	R/W
BITSCRAMBLER_TX_LUT_CFG1_REG	比特调节器 TX 内核 LUT 存储器访问寄存器	0x0014	R/W
BITSCRAMBLER_RX_LUT_CFG0_REG	比特调节器 RX 内核 LUT 存储器地址选择寄存器	0x0018	R/W
BITSCRAMBLER_RX_LUT_CFG1_REG	比特调节器 RX 内核 LUT 存储器访问寄存器	0x001C	R/W
配置寄存器			
BITSCRAMBLER_TX_TAILING_BITS_REG	比特调节器 TX 内核额外数据长度寄存器	0x0020	R/W
BITSCRAMBLER_RX_TAILING_BITS_REG	比特调节器 RX 内核额外数据长度寄存器	0x0024	R/W
BITSCRAMBLER_TX_CTRL_REG	比特调节器 TX 内核控制寄存器	0x0028	varies
BITSCRAMBLER_RX_CTRL_REG	比特调节器 RX 内核控制寄存器	0x002C	varies
BITSCRAMBLER_SYS_REG	环回控制寄存器	0x00F8	R/W
状态寄存器			
BITSCRAMBLER_TX_STATE_REG	比特调节器 TX 内核状态寄存器	0x0030	varies
BITSCRAMBLER_RX_STATE_REG	比特调节器 RX 内核状态寄存器	0x0034	varies
版本寄存器			
BITSCRAMBLER_VERSION_REG	版本控制寄存器	0x00FC	R/W

62.7 寄存器

本小节的所有地址均为相对于比特调节器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

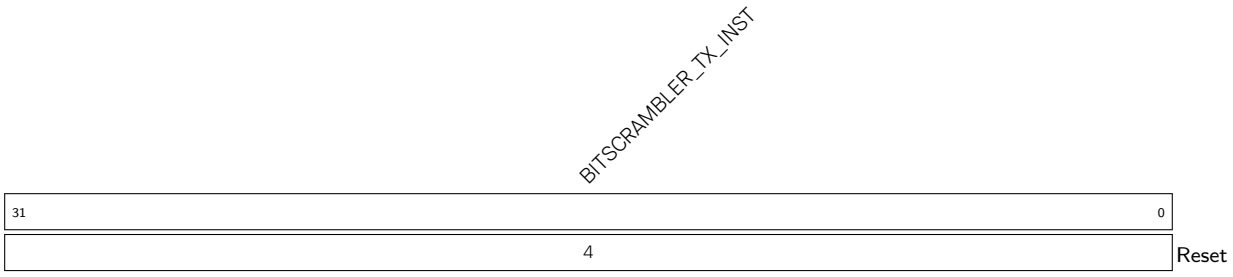
Register 62.1. BITSCRAMBLER_TX_INST_CFG0_REG (0x0000)



BITSCRAMBLER_TX_INST_IDX 配置比特调节器指令索引（通过 BITSCRAMBLER_TX_INST_CFG1_REG 访问指令）。(R/W)

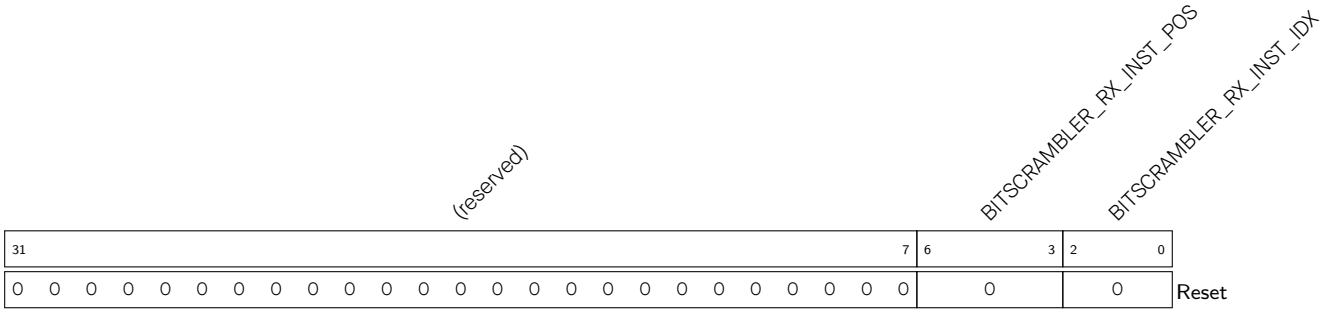
BITSCRAMBLER_TX_INST_POS 配置比特调节器 257 位指令的偏移量（以 32 位增量，通过 BITSCRAMBLER_TX_INST_CFG1_REG 访问指令）。(R/W)

Register 62.2. BITSCRAMBLER_TX_INST_CFG1_REG (0x0004)



BITSCRAMBLER_TX_INST 配置在 BITSCRAMBLER_TX_INST_CFG0_REG 指定的地址处访问的指令。(R/W)

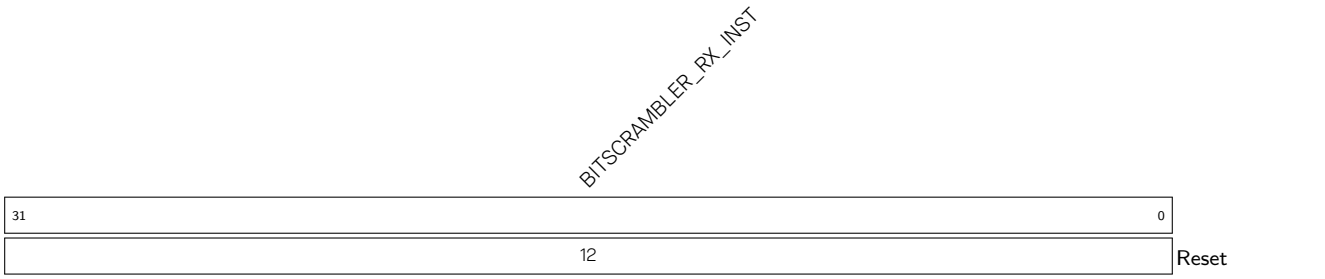
Register 62.3. BITSCRAMBLER_RX_INST_CFG0_REG (0x0008)



BITSCRAMBLER_RX_INST_IDX 配置比特调节器指令索引（通过 BITSCRAMBLER_RX_INST_CFG1_REG 访问指令）。(R/W)

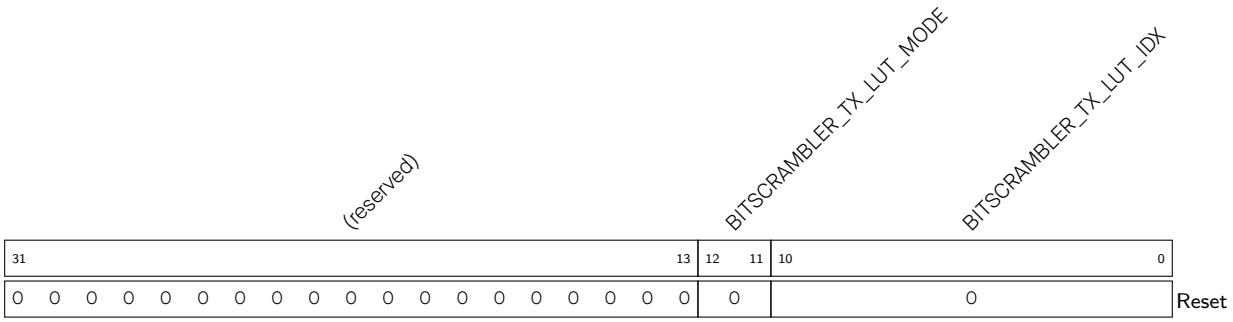
BITSCRAMBLER_RX_INST_POS 配置比特调节器 257 位指令的偏移量（以 32 位为增量，通过 BITSCRAMBLER_RX_INST_CFG1_REG 访问指令）。(R/W)

Register 62.4. BITSCRAMBLER_RX_INST_CFG1_REG (0x000C)



BITSCRAMBLER_RX_INST 配置在 BITSCRAMBLER_RX_INST_CFG0_REG 指定的地址处访问的指令。(R/W)

Register 62.5. BITSCRAMBLER_TX_LUT_CFG0_REG (0x0010)



BITSCRAMBLER_TX_LUT_IDX 配置访问 LUT RAM 的位置。单位：BITSCRAMBLER_TX_LUT_MODE 配置的字大小。(R/W)

BITSCRAMBLER_TX_LUT_MODE 为比特调节器程序配置 LUT RAM 的字大小。

0：1 字节

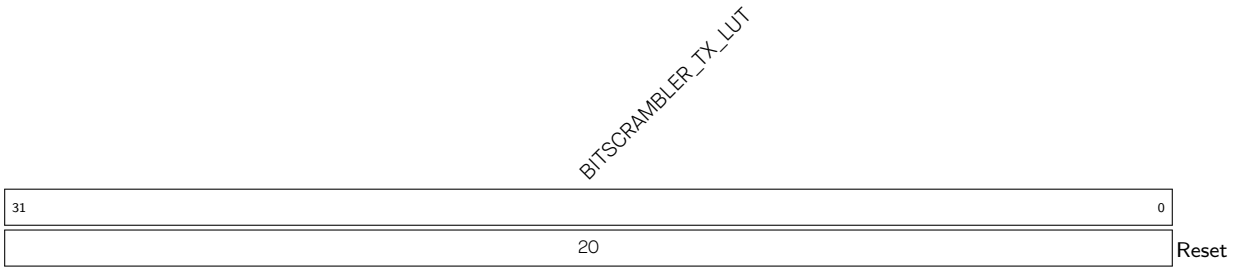
1：2 字节

2：4 字节

3：保留

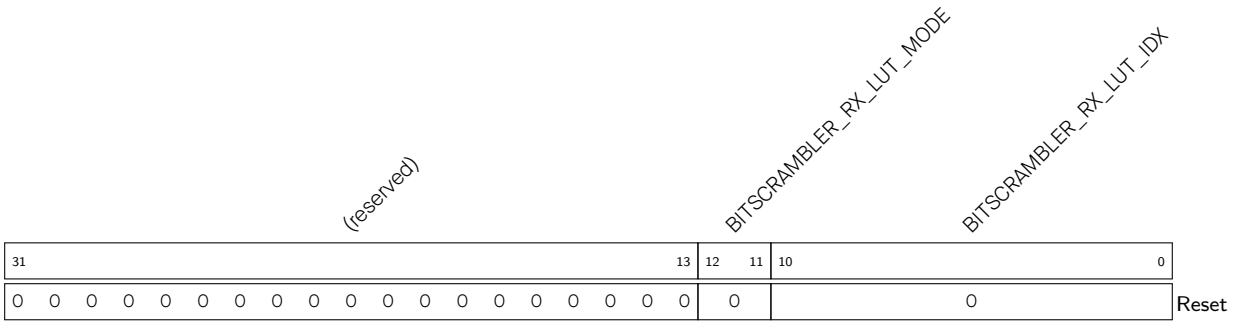
(R/W)

Register 62.6. BITSCRAMBLER_TX_LUT_CFG1_REG (0x0014)



BITSCRAMBLER_TX_LUT 配置在 BITSCRAMBLER_TX_LUT_CFG0_REG 指定的位置访问的 LUT 表项。(R/W)

Register 62.7. BITSCRAMBLER_RX_LUT_CFG0_REG (0x0018)



BITSCRAMBLER_RX_LUT_IDX 配置访问 LUT RAM 的位置。单位：BITSCRAMBLER_RX_LUT_MODE 配置的字大小。(R/W)

BITSCRAMBLER_RX_LUT_MODE 为比特调节器程序配置 LUT RAM 的字大小。

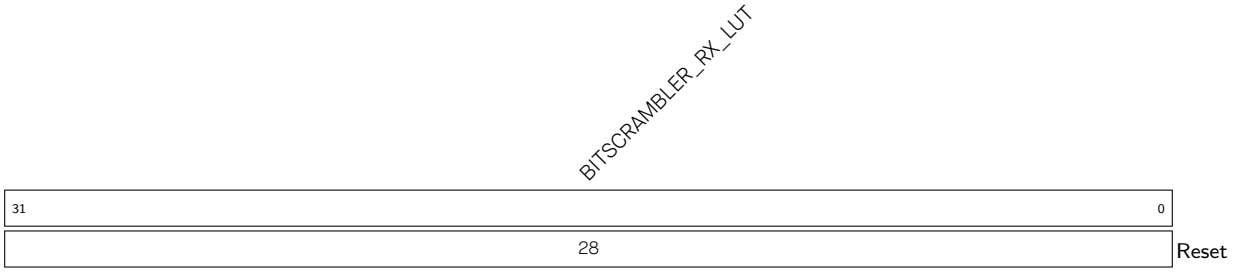
0：1 字节

1：2 字节

2：4 字节

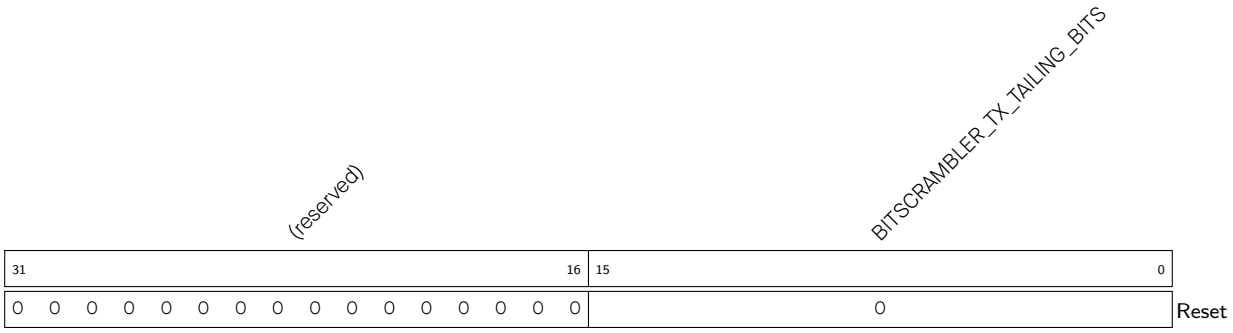
3：保留

Register 62.8. BITSCRAMBLER_RX_LUT_CFG1_REG (0x001C)



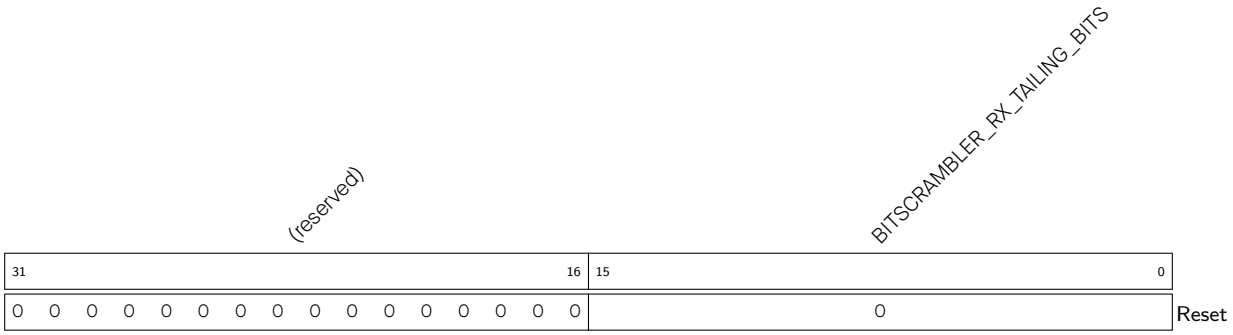
BITSCRAMBLER_RX_LUT 配置在 BITSCRAMBLER_RX_LUT_CFG0_REG 指定的位置访问的 LUT 表项。(R/W)

Register 62.9. BITSCRAMBLER_TX_TAILING_BITS_REG (0x0020)



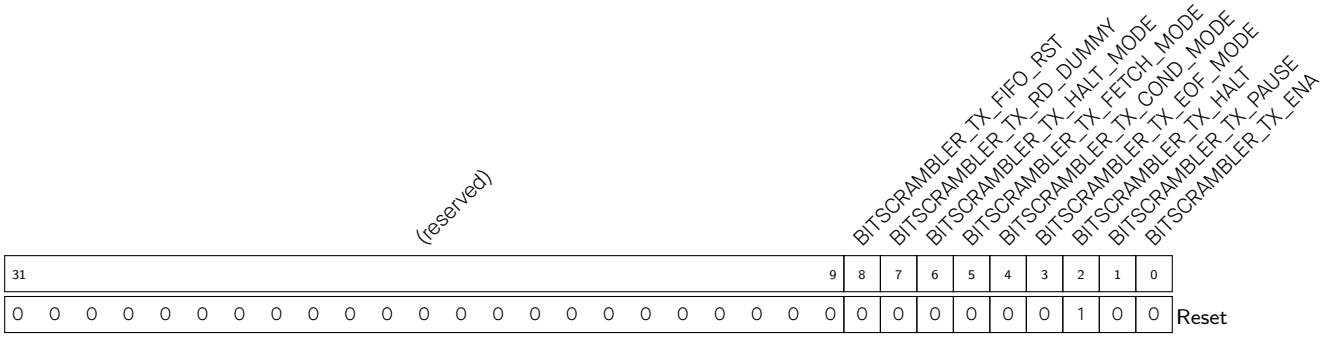
BITSCRAMBLER_TX_TAILING_BITS 配置比特调节器 TX 内核收到 EOF 后仍可处理额外数据的长度。单位：比特。(R/W)

Register 62.10. BITSCRAMBLER_RX_TAILING_BITS_REG (0x0024)



BITSCRAMBLER_RX_TAILING_BITS 配置比特调节器 RX 内核收到 EOF 后仍可处理额外数据的长度。单位：比特。(R/W)

Register 62.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)



BITSCRAMBLER_TX_ENA 配置是否启用比特调节器 TX 内核。

0：禁用

1：启用

(R/W)

BITSCRAMBLER_TX_PAUSE 配置是否暂停比特调节器 TX 内核。暂停的内核可取消暂停并恢复执行。

0：不暂停

1：暂停

(R/W)

BITSCRAMBLER_TX_HALT 配置是否停止比特调节器 TX 内核。停止的 TX 内核将刷新 FIFO 并且无法恢复，需要重置 FIFO 并重新启动。

0：不停止

1：停止

(R/W)

BITSCRAMBLER_TX_EOF_MODE 配置比特调节器 TX 内核 EOF 信号生成模式，与 BITSCRAMBLER_TX_TAILING_BITS 结合使用。

0：对写入输出 FIFO 的数据进行计数，生成延迟的 EOF

1：对从输入 FIFO 读取的数据进行计数，生成延迟的 EOF

(R/W)

BITSCRAMBLER_TX_COND_MODE 配置比特调节器 TX LOOP 指令条件模式。

0：使用小于运算符获取条件

1：使用不等于运算符获取条件

(R/W)

BITSCRAMBLER_TX_FETCH_MODE 配置比特调节器 TX 内核获取指令模式。

0：通过复位预取

1：通过指令获取

(R/W)

BITSCRAMBLER_TX_HALT_MODE 当 BITSCRAMBLER_TX_HALT 置 1 时，配置比特调节器 TX 内核暂停模式。

0：等待写回数据完成

1：忽略回写数据

(R/W)

见下页……

Register 62.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)

接上页……

BITSCRAMBLER_TX_RD_DUMMY 当接收到 EOF 时，配置比特调节器 TX 内核读取数据模式。

0：等待读取数据

1：忽略读取数据

(R/W)

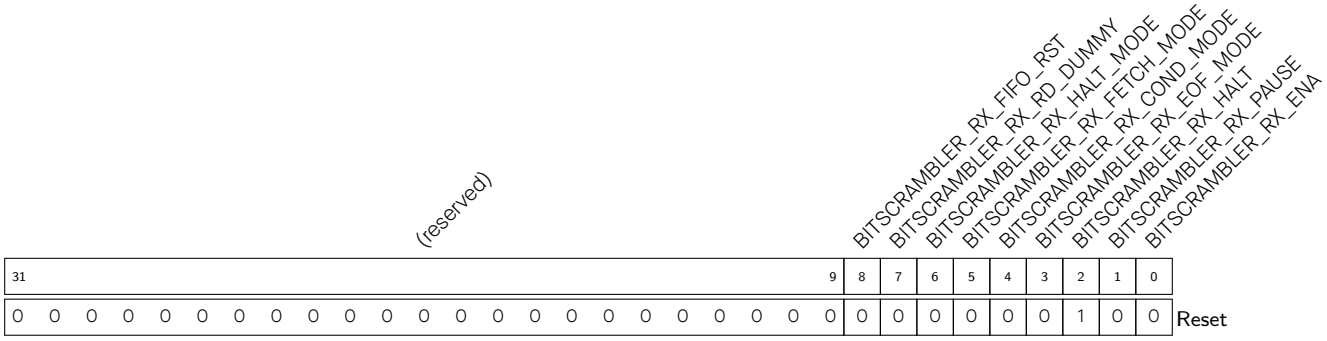
BITSCRAMBLER_TX_FIFO_RST 配置是否重置比特调节器 TX FIFO。

0：不复位

1：复位

(WT)

Register 62.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)



BITSCRAMBLER_RX_ENA 配置是否启用比特调节器 RX 内核。

0：禁用

1：启用

(R/W)

BITSCRAMBLER_RX_PAUSE 配置是否暂停比特调节器 RX 内核。暂停的内核可取消暂停并恢复执行。

0：不暂停

1：暂停

(R/W)

BITSCRAMBLER_RX_HALT 配置是否停止比特调节器 RX 内核。停止的 RX 内核将刷新 FIFO 并且无法恢复，需要重置 FIFO 并重新启动。

0：不停止

1：停止

(R/W)

BITSCRAMBLER_RX_EOF_MODE 配置比特调节器 RX 内核 EOF 信号生成模式，与 BITSCRAMBLER_RX_TAILING_BITS for use 结合使用。

0：对写入输出 FIFO 的数据进行计数，生成延迟的 EOF

1：对从输入 FIFO 读取的数据进行计数，生成延迟的 EOF

(R/W)

BITSCRAMBLER_RX_COND_MODE 配置比特调节器 RX LOOP 指令条件模式。

0：使用小于运算符获取条件

1：使用不等于运算符获取条件

(R/W)

BITSCRAMBLER_RX_FETCH_MODE 配置比特调节器 RX 内核获取指令模式。

0：通过复位预取

1：通过指令获取

(R/W)

BITSCRAMBLER_RX_HALT_MODE 当 BITSCRAMBLER_RX_HALT 置 1 时，配置比特调节器 RX 内核暂停模式。

0：等待写回数据完成

1：忽略回写数据

(R/W)

见下页……

Register 62.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)

接上页……

BITSCRAMBLER_RX_RD_DUMMY 当接收到 EOF 时，配置比特调节器 RX 内核读取数据模式。

0：等待读取数据

1：忽略读取数据

(R/W)

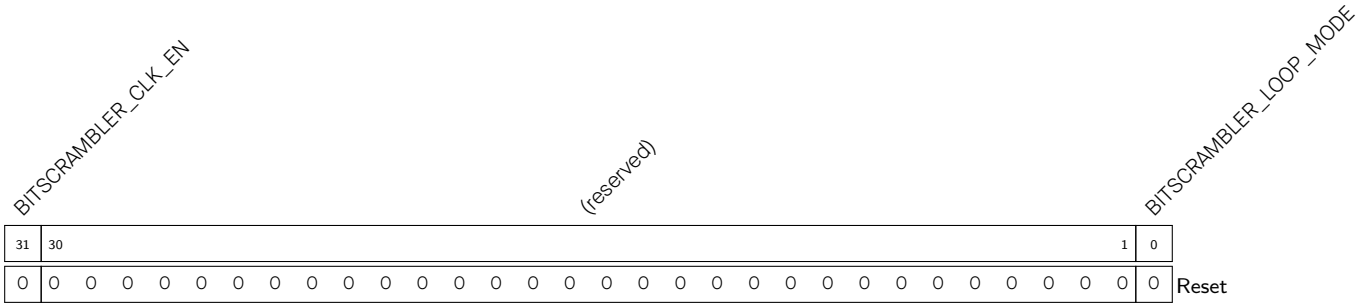
BITSCRAMBLER_RX_FIFO_RST 配置是否重置比特调节器 RX FIFO。

0：不复位

1：复位

(WT)

Register 62.13. BITSCRAMBLER_SYS_REG (0x00F8)



BITSCRAMBLER_LOOP_MODE 配置是否启用比特调节器 TX 到 DMA RX 环回模式

0：禁用

1：启用

(R/W)

BITSCRAMBLER_CLK_EN 保留。(R/W)

Register 62.14. BITSCRAMBLER_TX_STATE_REG (0x0030)



BITSCRAMBLER_TX_IN_IDLE 表示比特调节器 TX 内核是否停止。

0：未停止

1：停止

(RO)

BITSCRAMBLER_TX_IN_RUN 表示比特调节器 TX 内核是否正在运行。

0：未运行

1：运行

(RO)

BITSCRAMBLER_TX_IN_WAIT 表示比特调节器 TX 内核是否正在等待写回完成。

0：不等待

1：等待

(RO)

BITSCRAMBLER_TX_IN_PAUSE 表示比特调节器 TX 内核是否暂停。

0：未暂停

1：暂停

(RO)

BITSCRAMBLER_TX_FIFO_EMPTY 表示比特调节器 TX FIFO 是否为空。

0：不为空

1：空

(RO)

BITSCRAMBLER_TX_EOF_GET_CNT 表示收到 EOF 后比特调节器 TX 内核的字节计数。(RO)

BITSCRAMBLER_TX_EOF_OVERLOAD 表示比特调节器 TX 内核是否尝试处理多个 EOF。

0：不尝试处理多个 EOF

1：尝试处理多个 EOF

(RO)

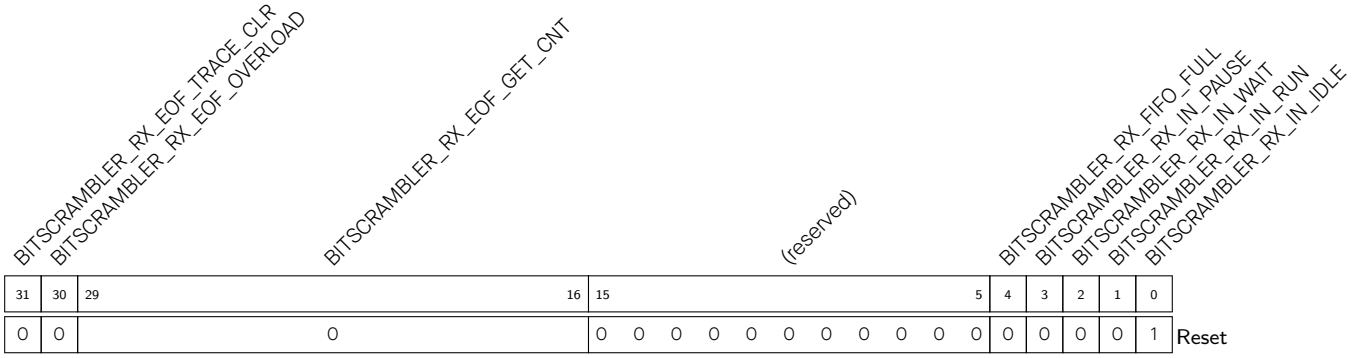
BITSCRAMBLER_TX_EOF_TRACE_CLR 配置是否清除 BITSCRAMBLER_TX_EOF_OVERLOAD 和 BITSCRAMBLER_TX_EOF_GET_CNT。

0：不清楚

1：清除

(WT)

Register 62.15. BITSCRAMBLER_RX_STATE_REG (0x0034)



BITSCRAMBLER_RX_IN_IDLE 表示比特调节器 RX 内核是否停止。

0：未停止

1：停止

(RO)

BITSCRAMBLER_RX_IN_RUN 表示比特调节器 RX 内核是否正在运行。

0：未运行

1：运行

(RO)

BITSCRAMBLER_RX_IN_WAIT 表示比特调节器 RX 内核是否正在等待写回完成。

0：不等待

1：等待

(RO)

BITSCRAMBLER_RX_IN_PAUSE 表示比特调节器 RX 内核是否暂停。

0：未暂停

1：暂停

(RO)

BITSCRAMBLER_RX_FIFO_FULL 表示比特调节器 RX FIFO 是否已满。

0：未满

1：已满

(RO)

BITSCRAMBLER_RX_EOF_GET_CNT 表示收到 EOF 后比特调节器 RX 内核的字节计数。(RO)

BITSCRAMBLER_RX_EOF_OVERLOAD 表示比特调节器 RX 内核是否尝试处理多个 EOF。

0：不尝试处理多个 EOF

1：尝试处理多个 EOF

(RO)

BITSCRAMBLER_RX_EOF_TRACE_CLR 配置是否清除 BITSCRAMBLER_RX_EOF_OVERLOAD 和 BITSCRAMBLER_RX_EOF_GET_CNT。

0：不清楚

1：清除

(WT)

Register 62.16. BITSCRAMBLER_VERSION_REG (0x00FC)



BITSCRAMBLER_BITSCRAMBLER_VER 版本控制寄存器。(R/W)

第 VIII 卷

模拟信号处理

该部分描述了与模数转换、电压比较、片上传感器以及温度和触摸感应等功能相关的模块，展示了系统在处理模拟信号方面的性能。

第 63 章

触摸传感器 (TOUCH)

ESP32-P4 搭载了 14 个电容式触摸传感器单元，每个单元可通过一个触摸管脚外接触摸板，构成触摸传感器系统。ESP32-P4 触摸传感器系统主要应用于人机交互场景中，检测手指触碰或接近，此外还支持防潮功能和遇水保护功能。

63.1 术语

为了更好地对电容式触摸传感器进行说明，本章及后续小节使用了以下术语。

触摸传感器 (Touch Sensor)	实现触摸传感功能的内部电路。
触摸管脚 (Touch Pin)	具有触摸传感功能的芯片管脚。
触摸板 (Touch Panel)	芯片外部连接的用于接收触摸动作的设备。
触摸传感器系统 (Touch Sensor System)	触摸管脚、触摸传感器、触摸板、以及连接各部分的电路共同构成触摸传感器系统。

说明：
为方便描述，对触摸板进行采样、扫描、测量、充放电等操作，即表示对触摸管脚进行采样、扫描、测量、充放电等操作。

63.2 特性

ESP32-P4 的触摸传感器模块具有以下特性：

- 支持 14 个电容触摸管脚的检测
- 可由软件或专用硬件定时器触发采样操作
- 支持两种采样方式：
  - 将来自触摸管脚的脉冲序列信号作为时钟信号处理，利用该时钟来计数采样周期
  - 将来自触摸管脚的脉冲序列信号作为数字信号处理，利用系统时钟采样该数字信号的上升沿来计数采样周期
- 支持扫描模式，可配置 Touch FSM 按照固定顺序对多个触摸管脚进行采样
- 支持超时机制，监测通道异常
- 支持跳频采样，增加检测的抗干扰性
- 支持接近感应模式，最多可配置三个通道
- 支持配置单个触摸传感器在休眠模式时正常工作
- 支持触摸传感器用作唤醒源

- 支持防潮功能
- 支持遇水保护功能

63.3 系统架构

63.3.1 触摸板

ESP32-P4 触摸传感器的触摸板主要由三部分组成：

- 电极 (Electrode)：手指触碰时，电极电容发生变化。
- 基片 (Substrate)：基础材料，承载平面保护层、电极及接入相应通道需要的连接部件。
- 平面保护层 (Protective cover)：将其余部件与外界环境隔开。

用户需自行设计触摸板，确定触摸板用材及排列方式等，具体信息可参考 触摸传感器应用方案简介。触摸板经触摸管脚（通道）连接至 ESP32-P4 提供的触摸传感器，如下图 63.3-1 所示。

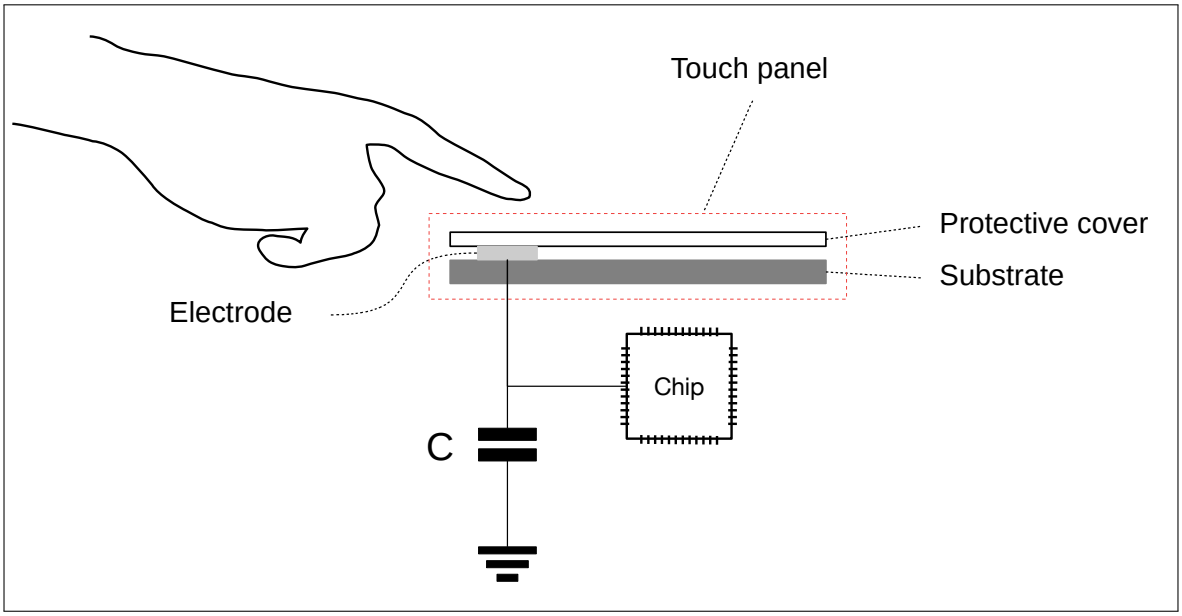


图 63.3-1. 触摸传感器系统

用户触碰保护层时，电极电容增加。如果该电极通过通道已连接至 ESP32-P4 触摸传感器，则传感器将检测到电极的电容变化。电容变化超过阈值（可配置），传感器将触发中断。

63.3.2 电容触摸管脚

ESP32-P4 搭载 14 个电容触摸管脚 (T1 ~ T14)，通过芯片电容触摸管脚与外部触摸板连接，用于检测外部的触摸情况。对应的连接关系见表 63.3-1。

表 63.3-1. ESP32-P4 电容式触摸传感器的管脚

电容触摸管脚	对应 GPIO 管脚
T1	GPIO2
T2	GPIO3

电容触摸管脚	对应 GPIO 管脚
T3	GPIO4
T4	GPIO5
T5	GPIO6
T6	GPIO7
T7	GPIO8
T8	GPIO9
T9	GPIO10
T10	GPIO11
T11	GPIO12
T12	GPIO13
T13	GPIO14
T14	GPIO15

63.3.3 触摸传感器

触摸传感器使用固定电源对触摸管脚进行充放电，每次充放电会生成一个脉冲信号，即 TOUCH_OUT 信号。触摸传感器可检测到触摸管脚上的电容变化，当手指触碰或靠近触摸管脚时，管脚电容增大，充放电变慢，因而产生脉冲信号的频率变慢。通过测量产生固定次数个脉冲信号所需的时间变化，即可判断管脚是否被触碰。

图 63.3-2 展示了触摸传感器的内部结构。每个触摸传感器均有一组输入输出信号，其中一些信号连接至 Touch FSM，见小节 63.4.1。表 63.3-2 展示了触摸传感器的各个输出信号及其功能。

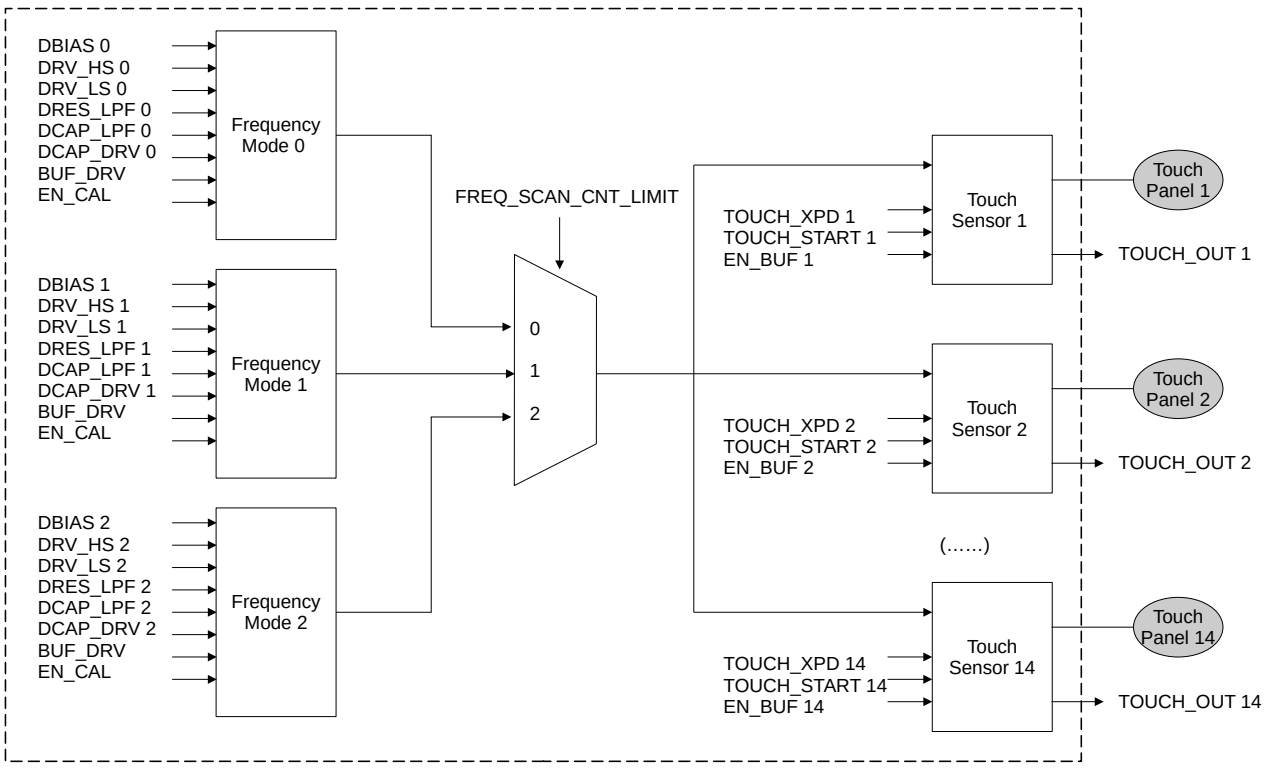


图 63.3-2. 触摸传感器内部结构

表 63.3-2. 电容式触摸传感器输出信号

信号	信号源	功能
TOUCH_XPD	Touch FSM	控制触摸传感器的上下电
TOUCH_START	Touch FSM	控制触摸传感器发起一次测量。Touch FSM 拉低 TOUCH_START 信号，测量结束
DBIAS	Touch FSM	FSM 会根据不同频率模式选择不同的 DBIAS 参数，用于调节触摸传感器内部 LDO 的电压，从而影响触摸传感器的充放电速度
DRV_HS	Touch FSM	FSM 会根据不同频率模式选择不同的 DRV_HS 参数，用于调节触摸传感器的高速驱动能力
DRV_LS	Touch FSM	FSM 会根据不同频率模式选择不同的 DRV_LS 参数，用于调节触摸传感器的低速驱动能力
DRES_LPF	Touch FSM	FSM 会根据不同频率模式选择不同的 DRES_LPF 参数，用于调节 RC 低通滤波器，调节档位分为 0 kΩ、3 kΩ、4.5 kΩ 和 7.5 kΩ。
DCAP_LPF	Touch FSM	FSM 会根据不同频率模式选择不同的 DCAP_LPF 参数，用于调节 RC 低通滤波器，调节范围为 0 ~ 2.54 pF，调节精度为 20 fF
DCAP_DRV	Touch FSM	用于在测量模式下调节连接触摸通道的内部电容值，通常设置为 0
EN_CAL	Touch FSM	用于断开和触摸管脚的连接，可以测试触摸传感器内部的电容（需要关闭触摸板）
BUF_DRV	Touch FSM	调节 TOUCH BUFF 的驱动能力
EN_BUF	Touch FSM	使能 TOUCH BUFF（软件可配置的一块触摸板）用于防潮模式，缓和小水滴对传感器的影响
TOUCH_OUT	触摸传感器	Touch FSM 对 TOUCH_OUT 信号中的脉冲进行计数，测量产生 N 个脉冲需要的时间

63.4 功能描述

63.4.1 Touch FSM

触摸传感器的 Touch FSM 负责执行测量操作，包括选择要测量的触摸传感器、为触摸传感器生成需要的信号、以及处理触摸传感器输出的信号等。Touch FSM 的内部结构及工作原理见图 63.4-1，时钟源为 LP_DYN_FAST_CLK。关于 LP_DYN_FAST_CLK 更多信息，请参考章节 11 复位和时钟。

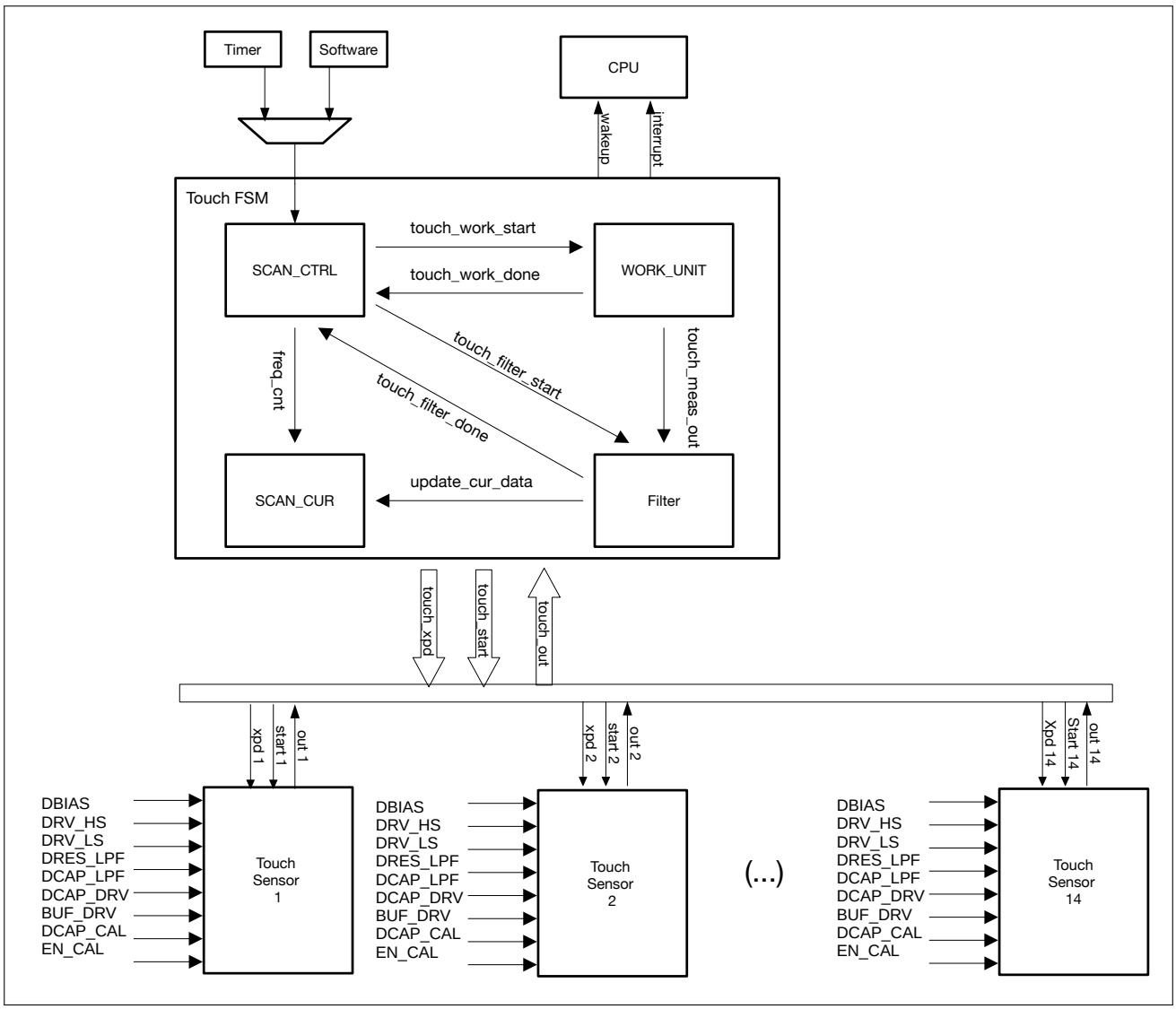


图 63.4-1. Touch FSM 内部结构和工作原理

Touch FSM 各个模块的功能如下：

- SCAN_CTRL 模块：扫描模式下，选择要测量的触摸传感器以及跳频等参数，控制每个触摸传感器的测量启动和结束。
- WORK_UNIT 模块：测量过程中，负责采样已选择的触摸传感器。
- Filter 模块：使能此模块，则触摸传感器在测量时将使用 IIR 滤波器，返回的滤波值即为采样值。详细信息见小节 63.4.3.1。
- SCAN_CUR 模块：测量过程中，用于缓存不同触摸传感器在不同频率模式下的参数，用于下一次测量时的计算。

63.4.2 采样信号预处理

触摸传感器的核心是通过检测触摸板上的电容变化，输出对应的脉冲序列（TOUCH_OUT 信号），通过计算触摸和非触摸状态下 TOUCH_OUT 信号的变化，来反映触摸板是否被触摸，因此，对 TOUCH_OUT 信号的采样十分关键。图 63.4-2 展示了对 TOUCH_OUT 信号预处理的过程。

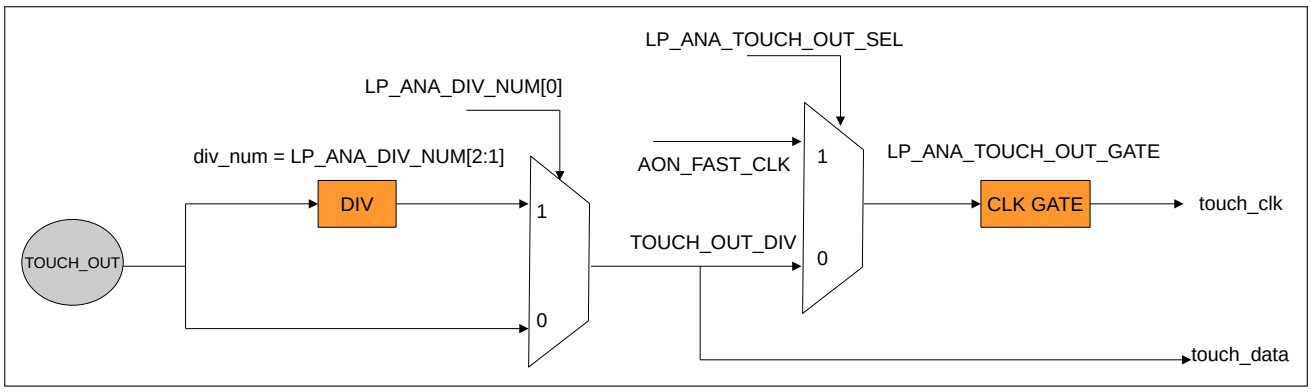


图 63.4-2. TOUCH_OUT 信号预处理

当 TOUCH_OUT 信号的频率远高于采样频率时，为了提高信号采样的准确性，可以通过配置 LP_ANA_DIV_NUM*n* 对 TOUCH_OUT 信号进行分频处理。*n* 代表在跳频模式（参考小节 63.4.2.4）下的不同频率模式，取值为 0 ~ 2。该寄存器的最低位为是否开启分频的标志位，若最低位配置为 0，则表示不对 TOUCH_OUT 信号进行分频处理，若最低位配置为 1，则表示需要对 TOUCH_OUT 信号进行分频处理。该寄存器的高两位表示分频系数，取值范围为 0、1、2、3，对应的分频范围为 0、2、4、6 分频。

若分频处理后，频率依然高于采样频率的两倍，可以配置 LP_ANA_TOUCH_OUT_SEL 为 1，将分频处理后的 TOUCH_OUT 信号直接当作时钟使用。配置 LP_ANA_TOUCH_OUT_GATE 可控制开启或关闭该时钟。

63.4.2.1 测量过程

根据是否开启跳频，触摸传感器的一次测量操作可以分为以下两种流程：

- 非跳频采样：
  1. Touch FSM 选择需要测量的触摸传感器，将相关配置信号发送到该传感器。
  2. Touch FSM 发送 START 信号到触摸传感器，初始化测量操作。Touch FSM 启动内部触摸计数器 (touch counter) 记录测量持续时长。
  3. Touch FSM 内部的脉冲计数器 (pulse counter) 开始对 TOUCH_OUT 信号的脉冲数量进行计数。
  4. 脉冲计数器达到 LP_ANA_TOUCH_MEAS_NUM0 中设定的计数阈值，则测量结束。START 信号拉低，触摸计数器停止计数。触摸计数器的值 x LP_DYN_SLOW_CLK 周期即为对触摸管脚进行 LP_ANA_TOUCH_MEAS_NUM0 次充放电所需的时间。
- 跳频采样：
  1. Touch FSM 选择需要测量的触摸传感器，将相关配置信号发送到该传感器。
  2. 置位 LP_ANA_FREQ_SCAN_EN 开启跳频模式，配置 LP_ANA_FREQ_SCAN_CNT_LIMIT 选择所支持跳频的频率模式个数，最大支持三种采样频率模式，并配置各频率模式所需的触摸传感器参数以及采样 TOUCH_OUT 信号的脉冲次数。
  3. Touch FSM 发送 START 信号到触摸传感器，初始化测量操作。Touch FSM 启动内部触摸计数器 (touch counter) 记录测量持续时长。
  4. Touch FSM 内部的脉冲计数器 (pulse counter) 开始对 TOUCH_OUT 信号的脉冲数量进行计数。
  5. 脉冲计数器达到对应频率模式 LP_ANA_TOUCH_MEAS_NUM*n* 中设定的计数阈值，则当前频率测量结束，触摸计数器的值即为当前频率模式下对触摸管脚进行 LP_ANA_TOUCH_MEAS_NUM*n* 次充放电所需时间。

6. 采样频率计数器加 1，触摸计数器清零，等待一小段延时后开始下一频率模式的采样。当所有频率模式采样结束后，START 信号拉低，触摸计数器清零。

采样值（测量值）为触摸计数器的值，表示为原始触摸数据 (touch_raw_data)。用户可从 RTC_TOUCH_PAD*n*_DATA 中读取原始触摸数据的值。注意，RTC_TOUCH_PAD*n*_DATA 返回的值也可能是滤波后的原始触摸数据，即平滑触摸数据 (touch_smooth_data) 和基准数据 (benchmark)。用户可配置 LP_ANA_TOUCH_DATA_SEL 和 LP_ANA_TOUCH_FREQ_SEL 选择具体的返回值类型。有关采样值返回类型的详细信息，以及如何使用这些数据来判断触摸动作，请参考小节 63.4.3.1。

说明：
注意，如果脉冲计数器长时间未达到设置的计数阈值，而触摸计数器已达到 LP_ANA_TOUCH_TIMEOUT_NUM 中设定的超时阈值，且触摸计数器超时中断使能寄存器 LP_ANA_TOUCH_TIMEOUT_EN 置位，则会触发 RTC_TOUCH_TIMEOUT_INT_RAW 超时中断，提示电路异常。

63.4.2.2 测量操作的触发源

Touch FSM 通过发送 START 信号发起一次测量操作。START 信号可由软件触发，或由触摸定时器 (touch timer) 触发。使用触摸定时器可发起周期性测量，而无需软件介入。

触摸定时器的时钟源为 LP_DYN_SLOW_CLK，时长可设置为若干个 LP_DYN_SLOW_CLK 周期。定时器到期则生成 START 信号。测量操作执行完毕后，定时器计数复位，开始重新计数，直至下次过期。

- 配置 START 信号由软件触发：
  - 置位 LP_ANA_TOUCH_START_FORCE，选择 START 信号由软件触发。
  - 软件置位 LP_ANA_TOUCH_START_EN，产生 START 信号，初始化一次测量操作。
- 配置 START 信号由触摸定时器触发：
  - 清零 LP_ANA_TOUCH_START_FORCE。
  - 通过 PMU_TOUCH_WAIT_CYCLES 配置定时器发送 START 信号前的等待时间（单位：LP_DYN_SLOW_CLK 周期）。
  - 通过 PMU_TOUCH_SLEEP_CYCLES 配置定时器接收到触摸传感器采样计算完成的 DONE 信号后的等待时间（单位：LP_DYN_SLOW_CLK 周期）。
  - 置位 PMU_TOUCH_SLEEP_TIMER_EN 使能定时器。
- 软件配置强制退出触摸定时器当前测量周期，可以采用两种配置方式：
  1. 采用软件产生触摸传感器的测量结束信号，进而退出触摸定时器当前测量周期。
     - 置位 LP_ANA_TOUCH_DONE_FORCE，选择 DONE 信号由软件触发。
     - 软件置位 LP_ANA_TOUCH_DONE_EN，产生 DONE 信号，结束一次测量操作。
  2. 采用软件直接强制退出触摸定时器当前测量周期。
     - 置位 PMU_TOUCH_FORCE_DONE，产生 DONE 强制退出触摸定时器当前测量周期。

说明：
触摸定时器在产生 START 信号之前和接收到 DONE 信号之后都会等待一段时间，因此连续测量两个触摸传感器之间的间隔时间为 (PMU_TOUCH_WAIT_CYCLES + PMU_TOUCH_SLEEP_CYCLES)，计时时钟为

LP_DYN_SLOW_CLK。

63.4.2.3 扫描模式

在扫描模式下，Touch FSM 对多个触摸传感器按照一定顺序进行测量。每生成一个 START 信号，就选择一个触摸传感器进行测量。因此，利用多个 START 信号即可测量多个触摸传感器。具体的扫描过程见图 63.4-3。

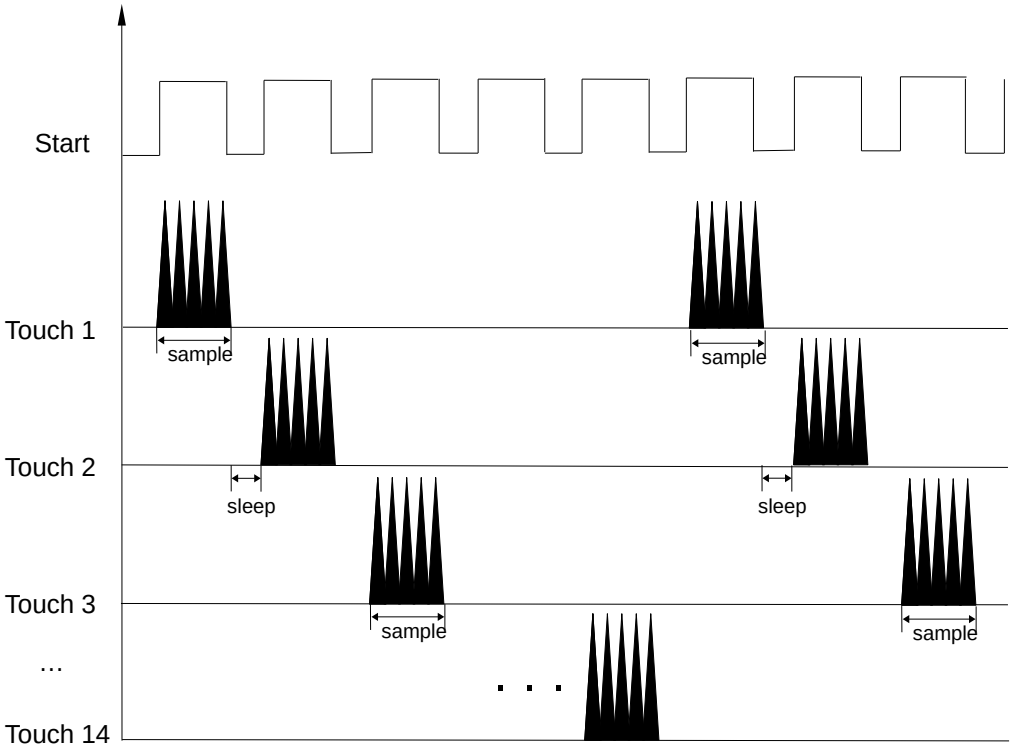


图 63.4-3. 扫描模式时序图

如需使能扫描模式，请在 LP_ANA_TOUCH_SCAN_PAD_MAP 中指定已使能触摸传感器的位图。对于每个 START 信号，Touch FSM 将从位图中依次选择一个触摸传感器进行测量。一个 START 信号仅执行一次测量操作。经过多个 START 信号，Touch FSM 按照顺序依次对使能的触摸传感器进行测量。

对不同触摸传感器进行测量时，两次测量之间的休眠时间间隔为 PMU_TOUCH_WAIT_CYCLES + PMU_TOUCH_SLEEP_CYCLES，时钟周期为 LP_DYN_SLOW_CLK。

说明：

- 触摸传感器在接收到来自触摸定时器的 START 信号后，为了保证采样信号的稳定，还会再等待一小段时间才开始测试，等待时间可通过 LP_ANA_TOUCH_XPD_WAIT 配置，计时时钟为 AON_FAST_CLK。
- 如果使用触摸定时器生成 START 信号，整个扫描过程无需软件介入。

63.4.2.4 跳频模式

在跳频模式下，Touch FSM 可以在不同频率模式下对触摸传感器进行测量。跳频模式最多支持三种频率模式，每种频率模式都分别对应一套参数配置。每生成一个 START 信号，即会按照频率模式依次进行测量，当所有频

率模式都测量结束时，才生成 DONE 信号。跳频模式和扫描模式可以一起使用，因此利用多个 START 信号即可监测多个触摸传感器在不同频率模式下的触摸情况。具体的扫描过程见图 63.4-4。

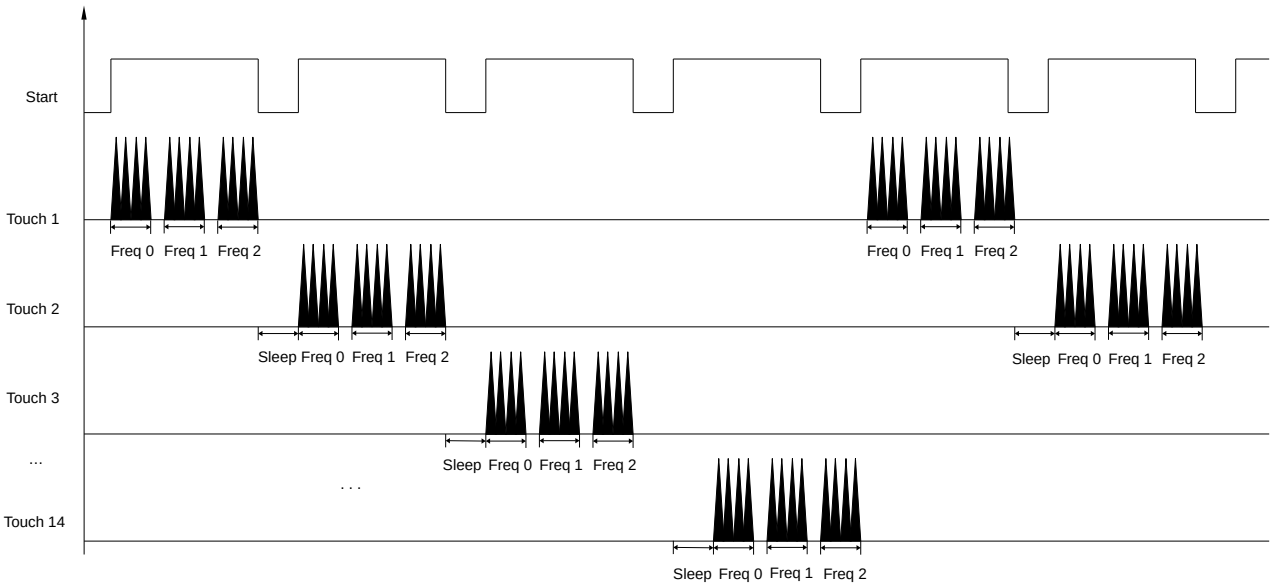


图 63.4-4. 触摸传感器跳频时序图

置位 LP_ANA_FREQ_SCAN_EN 可开启跳频模式，配置 LP_ANA_FREQ_SCAN_CNT_LIMIT 选择所支持的最大跳频数量，另外还需配置不同频率模式下调节触摸传感器采样性能的模拟参数寄存器组，具体如下：

- LP_ANA_TOUCH_FREQ*n*_DBIAS (*n* = 0 ~ 2)
- LP_ANA_TOUCH_FREQ*n*_DRV_HS (*n* = 0 ~ 2)
- LP_ANA_TOUCH_FREQ*n*_DRV_LS (*n* = 0 ~ 2)
- LP_ANA_TOUCH_FREQ*n*_DRES_LPF (*n* = 0 ~ 2)
- LP_ANA_TOUCH_FREQ*n*_DCAP_LPF (*n* = 0 ~ 2)

对于每个 START 信号，Touch FSM 将按照频率模式 0、1、2 的顺序依次进行测量。一个 START 信号最多触发三次测量操作。当所有频率模式都测量结束，START 信号拉低，等待下一个触摸传感器的测量。经过多个 START 信号，Touch FSM 依次对使能的触摸传感器进行测量。同一触摸传感器只有在大于等于 LP_ANA_FREQ_SCAN_CNT_RISE 个频点均被触发时，才能判定为触摸。

说明：

- 同一个触摸传感器在不同频率模式下测量时，两次测量之间的等待时间可通过 LP_ANA_TOUCH_XPD_WAIT 配置，计数时钟是 AON_FAST_CLK。
- 触摸传感器在接收到来自触摸定时器的 START 信号后，为了保证采样信号的稳定，会等待一小段时间才开始测量，等待时间可通过 LP_ANA_TOUCH_XPD_WAIT 配置，计数时钟为 AON_FAST_CLK。
- 如果使用触摸定时器生成 START 信号，整个扫描过程无需软件介入。

63.4.3 触摸检测

63.4.3.1 采样值

如小节 63.4.2.1 所述，用户可从 RTC_TOUCH_PAD*n*_DATA 中读取测量所得采样值。但 RTC_TOUCH_PAD*n*_DATA 中所存储的具体采样值类型由 LP_ANA_TOUCH_DATA_SEL 和 LP_ANA_TOUCH_FREQ_SEL 共同决定。支持以下三种类型的采样值：

1. **原始触摸数据** (touch_raw_data)：测量结束时，触摸计数器的计数值表示对触摸传感器进行固定次数的充放电所需的时间，单位为 LP_DYN_SLOW_CLK 周期。开启 IIR 滤波器后，该数据为滤波后的数据。
2. **平滑触摸数据** (touch_smooth_data)：一系列来自某个触摸传感器的原始触摸数据经 IIR 滤波生成的移动平均线。

   平滑触摸数据不易被噪声或采样异常值干扰，因此常用于硬件触摸检测。若开启跳频模式，每个触摸管脚在不同频率模式下分别对应一套平滑触摸数据，各自维护，互不影响。生成平滑触摸数据的 IIR 滤波设置可在 LP_ANA_TOUCH_SMOOTH_LVL 中配置。具体信息见表 63.4-1。

表 63.4-1. 平滑触摸数据计算

LP_ANA_TOUCH_SMOOTH_LVL	类型	公式
0	—	touch_raw_data
1	IIR 1/2	1/2 touch_raw_data + 1/2 touch_smooth_data
2	IIR 1/4	1/4 touch_raw_data + 3/4 touch_smooth_data
3	IIR 1/8	1/8 touch_raw_data + 7/8 touch_smooth_data

3. **基准数据** (benchmark)：同样由原始触摸数据经 IIR 滤波处理而生成，但其移动平均线窗口更广。

   基准数据常用于表示触摸传感器的稳定读数，不受触摸动作的影响。触摸检测的判断基于基准数据进行。每个触摸传感器在不同频率模式下都有一套独立的基准数据，其初始默认值为 0xFFFF。生成基准数据的 IIR 滤波可通过 LP_ANA_TOUCH_FILTER_MODE 配置，具体信息见表 63.4-2。

   除了通过 IIR 滤波的方式更新基准数据以外，还可以通过软件的方式直接更新。软件更新基准数据的步骤如下：

   - 通过 LP_ANA_TOUCH_UPDATE_BENCHMARK_FREQ_SEL 配置需要更新基准数据的目标频点。
   - 通过 LP_ANA_TOUCH_UPDATE_BENCHMARK_PAD_SEL 配置需要更新基准数据的目标通道。
   - 通过 LP_ANA_TOUCH_BENCHMARK_SW 配置基准数据值。
   - 配置 LP_ANA_TOUCH_UPDATE_BENCHMARK_SW，即可通过软件触发基准数据更新。

   该方式一般用于软件调试。当目标频点和通道的基准数据完成更新后，会产生中断 TOUCH_BENCHMARK_UPDATE_INT。

表 63.4-2. 基准数据算法

LP_ANA_TOUCH_FILTER_MODE	类型	公式
0	IIR 1/4	1/4 touch_raw_data + 3/4 benchmark
1	IIR 1/8	1/8 touch_raw_data + 7/8 benchmark
2	IIR 1/16	1/16 touch_raw_data + 15/16 benchmark
3	IIR 1/32	1/32 touch_raw_data + 31/32 benchmark
4	IIR 1/64	1/64 touch_raw_data + 63/64 benchmark
5	IIR 1/128	1/128 touch_raw_data + 127/128 benchmark

表 63.4-2. 基准数据算法

LP_ANA_TOUCH_FILTER_MODE	类型	公式
6	IIR 1/256	1/256 touch_raw_data + 255/256 benchmark
7	JITTER	touch_raw_data +/– LP_ANA_TOUCH_JITTER_STEP

63.4.3.2 硬件触摸检测

硬件触摸检测可检测到触摸动作或触摸释放动作，然后触发中断。使用硬件触摸检测，需定义如下几个关键参数：

- **触碰阈值** (finger_threshold)：用于判定触摸或触摸中断的比较值。触碰阈值由寄存器 LP_ANA_TOUCH_PAD*x*_TH*y* 配置，*x* 代表触摸管脚编号，取值为 1 ~ 14，*y* 代表跳频频率模式，取值为 0 ~ 2。
- **噪声阈值** (noise_threshold)：用于表示平滑触摸数据上下波动的最大值。噪声阈值由 LP_ANA_TOUCH_NOISE_THRES 配置。
- **迟滞** (hysteresis)：用于避免平滑触摸数据在触碰阈值附近来回波动导致的触摸误判。迟滞由 LP_ANA_TOUCH_HYSTERESIS 配置。

触碰阈值和噪声阈值均为基准数据的偏移，而不是绝对阈值，迟滞为触碰阈值的偏移。采用这种方法可以避免将原始触摸数据的渐变（缓慢变化）误测为触摸。多种环境因素，比如温度、电源、噪声，均有可能造成原始触摸数据的渐变。

触摸传感器基于上述参数进行一系列计算和比较之后，最终会将触摸管脚上检测到的触碰结果通过中断的方式输出。每个触摸传感器都对应一个控制寄存器 LP_ANA_TOUCH_OUTEN 来配置是否输出检测结果。

63.4.4 接近模式

物体靠近，但未接触触摸管脚时，触摸管脚电容将发生轻微变动，远小于物理触摸引起的电容变化。接近模式可用于检测电容轻微变动。

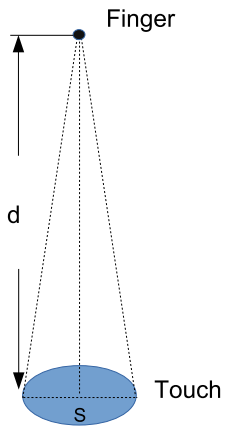


图 63.4-5. 感测面积示意图

- 检测距离 “d” 最大为 16 cm，感应面积 “S” 为 20 cm^2，“d” 与 “S” 正相关，如图 63.4-5 所示。
- 最多可以同时将三个触摸管脚配置为接近模式。
- 接近模式同样支持跳频检测。

在接近模式下，触摸传感器进行固定次数的采样并累计采样值。如果最终累计采样值超出设定的阈值，则认为检测到了物体靠近，并触发中断。注意，接近模式会对采样值进行累计，所以触摸传感器不会生成小节 63.4.3.1 所述的数值，即原始触摸数据、平滑触摸数据和基准数据。

接近模式操作流程如下：

- 通过配置 LP_ANA_TOUCH_APPROACH_PAD0、LP_ANA_TOUCH_APPROACH_PAD1 或 LP_ANA_TOUCH_APPROACH_PAD2，配置触摸传感器为接近模式。
- 通过配置 LP_ANA_TOUCH_APPROACH_LIMIT 设置采样次数，用于生成累计值。触摸传感器内置采样计数器，用于记录采样次数。
- 配置 LP_ANA_TOUCH_PAD*x*_TH*y* 设置阈值，*x* 代表触摸管脚编号，取值为 1 ~ 14，*y* 代表跳频频率模式，取值为 0 ~ 2。
- 采样计数器达到 LP_ANA_TOUCH_APPROACH_MEAS_NUM*n* 设置的采样次数：
  - 如果累计值大于设定的阈值，则触发中断。
  - 采样计数器和累计值复位归零。触摸传感器重新开始累计采样值。

63.4.5 休眠模式

为了降低功耗，触摸传感器支持休眠模式。在休眠状态下，可以指定一个触摸传感器处于活跃状态（以下简称为休眠触摸传感器），实时监测是否有触摸发生，其余 13 个触摸传感器全部掉电并处于休眠状态。休眠触摸传感器一旦检测到触摸或触摸释放，会立即产生对应中断，唤醒其余 13 个触摸传感器。休眠触摸传感器和正常模式下的触摸传感器一样，支持跳频模式和接近感应模式。休眠触摸传感器对应一套独立的阈值参数，其配置流程如下：

- 配置 LP_ANA_TOUCH_SLP_PAD 选择某一个触摸传感器作为休眠触摸传感器。
- 配置 LP_ANA_TOUCH_SLP_TH0、LP_ANA_TOUCH_SLP_TH1、LP_ANA_TOUCH_SLP_TH2，设置休眠触摸传感器在不同频率模式下的触碰阈值。
- 配置 LP_ANA_TOUCH_SLP_APPROACH_EN，设置休眠触摸传感器开启接近模式。
- 配置 LP_ANA_TOUCH_SLP_CHANNEL_CLR，清零休眠触摸传感器的基准数据、平滑触摸数据以及采样次数等中间数据。

说明：
休眠模式下，休眠触摸传感器的噪声阈值、迟滞、平滑触摸数据、基准数据与正常模式下的配置相同。

63.4.6 防潮功能

如果触摸板上有小水珠，且面积大到足以连通两个或更多触摸板，则可能导致临近触摸板出现电耦合现象。耦合现象会导致电容变化，此时触摸板可能会误测触摸动作。当小水珠跨接在触摸板和接地之间时也会引入较大的寄生电容，导致误触发。防潮设计可缓和小水珠带来的影响。

用户可配置使用防潮功能。具体步骤如下：

- 配置 LP_ANA_TOUCH_SHIELD_PAD_EN 开启防潮功能；
- 配置 LP_ANA_TOUCH_BUFSEL 选择用于防潮功能的触摸管脚。

说明：
防潮功能仅在频率低于 10 MHz 时有效。因此，当频率高于 10 MHz 时，请禁用防潮功能。

63.4.7 遇水保护功能

如果传感器触摸板阵列变潮，即大部分触摸板已浸水，全部或大部分触摸管脚因存在误测触摸动作的可能，触摸传感器将无法继续检测触摸动作。因此，ESP32-P4 触摸传感器支持遇水保护功能，该设计可检测到阵列已浸水，可关停传感器阵列。

用户可配置 LP_ANA_TOUCH_OUT_RING 选择用于遇水保护功能的触摸管脚。

63.5 中断

ESP32-P4 的触摸传感器可以生成 LP_TOUCH_INTR 中断信号并发送给 中断矩阵。

LP_TOUCH_INTR 中断信号由触摸传感器的内部中断源生成。表 63.5-1 列出了触摸传感器的内部中断源、中断触发条件、以及生成的中断信号。

表 63.5-1. 触摸传感器的内部中断源

内部中断源	触发条件	中断信号
TOUCH_APPROACH_LOOP_DONE_INT	接近模式累计采样结束	LP_TOUCH_INTR
TOUCH_TIMEOUT_INT	对 TOUCH_OUT 信号采样超时	
TOUCH_INACTIVE_INT	检测到触摸释放	
TOUCH_ACTIVE_INT	检测到触摸	
TOUCH_DONE_INT	对单个触摸管脚在某个频率模式下的采样比较完成	
TOUCH_SCAN_DONE_INT	对所有选中的触摸管脚在所有频率模式下都采样比较完成	
TOUCH_BENCHMARK_UPDATE_INT	软件更新基准数据完成	

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4 中断术语*。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 63.6 寄存器列表。

63.6 寄存器列表

63.6.1 中断与状态寄存器列表

本小节的所有地址均为相对于触摸传感器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
RTC_TOUCH_INT_RAW_REG	触摸传感器原始中断寄存器	0x0000	R/SS/WTC
RTC_TOUCH_INT_ST_REG	触摸传感器中断状态寄存器	0x0004	RO
RTC_TOUCH_INT_ENA_REG	触摸传感器中断使能寄存器	0x0008	R/W
RTC_TOUCH_INT_CLR_REG	触摸传感器中断清零寄存器	0x000C	WT
状态寄存器			
RTC_TOUCH_CHN_STATUS_REG	扫描模式状态寄存器	0x0010	RO
RTC_TOUCH_STATUS_1_REG	触摸管脚 1 状态寄存器	0x0018	RO
RTC_TOUCH_STATUS_2_REG	触摸管脚 2 状态寄存器	0x001C	RO
RTC_TOUCH_STATUS_3_REG	触摸管脚 3 状态寄存器	0x0020	RO
RTC_TOUCH_STATUS_4_REG	触摸管脚 4 状态寄存器	0x0024	RO
RTC_TOUCH_STATUS_5_REG	触摸管脚 5 状态寄存器	0x0028	RO
RTC_TOUCH_STATUS_6_REG	触摸管脚 6 状态寄存器	0x002C	RO
RTC_TOUCH_STATUS_7_REG	触摸管脚 7 状态寄存器	0x0030	RO
RTC_TOUCH_STATUS_8_REG	触摸管脚 8 状态寄存器	0x0034	RO
RTC_TOUCH_STATUS_9_REG	触摸管脚 9 状态寄存器	0x0038	RO
RTC_TOUCH_STATUS_10_REG	触摸管脚 10 状态寄存器	0x003C	RO
RTC_TOUCH_STATUS_11_REG	触摸管脚 11 状态寄存器	0x0040	RO
RTC_TOUCH_STATUS_12_REG	触摸管脚 12 状态寄存器	0x0044	RO
RTC_TOUCH_STATUS_13_REG	触摸管脚 13 状态寄存器	0x0048	RO
RTC_TOUCH_STATUS_14_REG	触摸管脚 14 状态寄存器	0x004C	RO
RTC_TOUCH_STATUS_15_REG	休眠模式的触摸管脚状态寄存器	0x0050	RO

GoBack

名称	描述	地址	访问
RTC_TOUCH_STATUS_16_REG	接近模式状态寄存器	0x0054	RO
RTC_TOUCH_STATUS_17_REG	跳频模式状态寄存器	0x0058	RO
RTC_TOUCH_CHN_TMP_STATUS_REG	触摸状态寄存器	0x005C	RO
版本寄存器			
RTC_TOUCH_DATE_REG	版本控制寄存器	0x0100	R/W

63.6.2 配置寄存器列表

本小节的所有地址均为相对于 LP 模拟外设基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_ANA_TOUCH_APPROACH_WORK_MEAS_NUM_REG	接近模式下测量次数配置寄存器	0x00FC	R/W
LP_ANA_TOUCH_SCAN_CTRL1_REG	扫描模式配置寄存器 1	0x0100	R/W
LP_ANA_TOUCH_SCAN_CTRL2_REG	扫描模式配置寄存器 2	0x0104	R/W
LP_ANA_TOUCH_WORK_REG	采样预处理寄存器	0x0108	varies
LP_ANA_TOUCH_WORK_MEAS_NUM_REG	不同频率模式下测量次数配置寄存器	0x010C	R/W
LP_ANA_TOUCH_FILTER1_REG	触摸检测配置寄存器 1	0x0110	R/W
LP_ANA_TOUCH_FILTER2_REG	触摸检测配置寄存器 2	0x0114	R/W
LP_ANA_TOUCH_FILTER3_REG	触摸检测配置寄存器 3	0x0118	varies
LP_ANA_TOUCH_SLP0_REG	休眠模式配置寄存器 1	0x011C	varies
LP_ANA_TOUCH_SLP1_REG	休眠模式配置寄存器 2	0x0120	R/W
LP_ANA_TOUCH_CLR_REG	状态清零寄存器	0x0124	WT
LP_ANA_TOUCH_APPROACH_REG	接近模式配置寄存器	0x0128	R/W
LP_ANA_TOUCH_FREQ0_SCAN_PARA_REG	触摸传感器实现采样频率模式 0 的模拟参数配置寄存器	0x012C	R/W
LP_ANA_TOUCH_FREQ1_SCAN_PARA_REG	触摸传感器实现采样频率模式 1 的模拟参数配置寄存器	0x0130	R/W
LP_ANA_TOUCH_FREQ2_SCAN_PARA_REG	触摸传感器实现采样频率模式 2 的模拟参数配置寄存器	0x0134	R/W

GoBack

GoBack

名称	描述	地址	访问
LP_ANA_TOUCH_ANA_PARA_REG	触摸传感器模拟参数配置寄存器	0x0138	R/W
LP_ANA_TOUCH_MUX0_REG	DEBUG 模式选择配置寄存器	0x013C	R/W
LP_ANA_TOUCH_MUX1_REG	DEBUG 模式功能配置寄存器	0x0140	R/W
LP_ANA_TOUCH_PAD1_TH0_REG	触摸管脚 1 在采样频率模式 0 下的触碰阈值配置寄存器	0x0150	R/W
LP_ANA_TOUCH_PAD1_TH1_REG	触摸管脚 1 在采样频率模式 1 下的触碰阈值配置寄存器	0x0154	R/W
LP_ANA_TOUCH_PAD1_TH2_REG	触摸管脚 1 在采样频率模式 2 下的触碰阈值配置寄存器	0x0158	R/W
LP_ANA_TOUCH_PAD2_TH0_REG	触摸管脚 2 在采样频率模式 0 下的触碰阈值配置寄存器	0x015C	R/W
LP_ANA_TOUCH_PAD2_TH1_REG	触摸管脚 2 在采样频率模式 1 下的触摸阈值配置寄存器	0x0160	R/W
LP_ANA_TOUCH_PAD2_TH2_REG	触摸管脚 2 在采样频率模式 2 下的触摸阈值配置寄存器	0x0164	R/W
LP_ANA_TOUCH_PAD3_TH0_REG	触摸管脚 3 在采样频率模式 0 下的触摸阈值配置寄存器	0x0168	R/W
LP_ANA_TOUCH_PAD3_TH1_REG	触摸管脚 3 在采样频率模式 1 下的触摸阈值配置寄存器	0x016C	R/W
LP_ANA_TOUCH_PAD3_TH2_REG	触摸管脚 3 在采样频率模式 2 下的触摸阈值配置寄存器	0x0170	R/W
LP_ANA_TOUCH_PAD4_TH0_REG	触摸管脚 4 在采样频率模式 0 下的触摸阈值配置寄存器	0x0174	R/W
LP_ANA_TOUCH_PAD4_TH1_REG	触摸管脚 4 在采样频率模式 1 下的触摸阈值配置寄存器	0x0178	R/W
LP_ANA_TOUCH_PAD4_TH2_REG	触摸管脚 4 在采样频率模式 2 下的触摸阈值配置寄存器	0x017C	R/W
LP_ANA_TOUCH_PAD5_TH0_REG	触摸管脚 5 在采样频率模式 0 下的触摸阈值配置寄存器	0x0180	R/W
LP_ANA_TOUCH_PAD5_TH1_REG	触摸管脚 5 在采样频率模式 1 下的触摸阈值配置寄存器	0x0184	R/W
LP_ANA_TOUCH_PAD5_TH2_REG	触摸管脚 5 在采样频率模式 2 下的触摸阈值配置寄存器	0x0188	R/W
LP_ANA_TOUCH_PAD6_TH0_REG	触摸管脚 6 在采样频率模式 0 下的触摸阈值配置寄存器	0x018C	R/W
LP_ANA_TOUCH_PAD6_TH1_REG	触摸管脚 6 在采样频率模式 1 下的触摸阈值配置寄存器	0x0190	R/W
LP_ANA_TOUCH_PAD6_TH2_REG	触摸管脚 6 在采样频率模式 2 下的触摸阈值配置寄存器	0x0194	R/W
LP_ANA_TOUCH_PAD7_TH0_REG	触摸管脚 7 在采样频率模式 0 下的触摸阈值配置寄存器	0x0198	R/W
LP_ANA_TOUCH_PAD7_TH1_REG	触摸管脚 7 在采样频率模式 1 下的触摸阈值配置寄存器	0x019C	R/W
LP_ANA_TOUCH_PAD7_TH2_REG	触摸管脚 7 在采样频率模式 2 下的触摸阈值配置寄存器	0x01A0	R/W
LP_ANA_TOUCH_PAD8_TH0_REG	触摸管脚 8 在采样频率模式 0 下的触摸阈值配置寄存器	0x01A4	R/W
LP_ANA_TOUCH_PAD8_TH1_REG	触摸管脚 8 在采样频率模式 1 下的触摸阈值配置寄存器	0x01A8	R/W
LP_ANA_TOUCH_PAD8_TH2_REG	触摸管脚 8 在采样频率模式 2 下的触摸阈值配置寄存器	0x01AC	R/W
LP_ANA_TOUCH_PAD9_TH0_REG	触摸管脚 9 在采样频率模式 0 下的触摸阈值配置寄存器	0x01B0	R/W
LP_ANA_TOUCH_PAD9_TH1_REG	触摸管脚 9 在采样频率模式 1 下的触摸阈值配置寄存器	0x01B4	R/W

名称	描述	地址	访问
LP_ANA_TOUCH_PAD9_TH2_REG	触摸管脚 9 在采样频率模式 2 下的触摸阈值配置寄存器	0x01B8	R/W
LP_ANA_TOUCH_PAD10_TH0_REG	触摸管脚 10 在采样频率模式 0 下的触摸阈值配置寄存器	0x01BC	R/W
LP_ANA_TOUCH_PAD10_TH1_REG	触摸管脚 10 在采样频率模式 1 下的触摸阈值配置寄存器	0x01C0	R/W
LP_ANA_TOUCH_PAD10_TH2_REG	触摸管脚 10 在采样频率模式 2 下的触摸阈值配置寄存器	0x01C4	R/W
LP_ANA_TOUCH_PAD11_TH0_REG	触摸管脚 11 在采样频率模式 0 下的触摸阈值配置寄存器	0x01C8	R/W
LP_ANA_TOUCH_PAD11_TH1_REG	触摸管脚 11 在采样频率模式 1 下的触摸阈值配置寄存器	0x01CC	R/W
LP_ANA_TOUCH_PAD11_TH2_REG	触摸管脚 11 在采样频率模式 2 下的触摸阈值配置寄存器	0x01D0	R/W
LP_ANA_TOUCH_PAD12_TH0_REG	触摸管脚 12 在采样频率模式 0 下的触摸阈值配置寄存器	0x01D4	R/W
LP_ANA_TOUCH_PAD12_TH1_REG	触摸管脚 12 在采样频率模式 1 下的触摸阈值配置寄存器	0x01D8	R/W
LP_ANA_TOUCH_PAD12_TH2_REG	触摸管脚 12 在采样频率模式 2 下的触摸阈值配置寄存器	0x01DC	R/W
LP_ANA_TOUCH_PAD13_TH0_REG	触摸管脚 13 在采样频率模式 0 下的触摸阈值配置寄存器	0x01E0	R/W
LP_ANA_TOUCH_PAD13_TH1_REG	触摸管脚 13 在采样频率模式 1 下的触摸阈值配置寄存器	0x01E4	R/W
LP_ANA_TOUCH_PAD13_TH2_REG	触摸管脚 13 在采样频率模式 2 下的触摸阈值配置寄存器	0x01E8	R/W
LP_ANA_TOUCH_PAD14_TH0_REG	触摸管脚 14 在采样频率模式 0 下的触摸阈值配置寄存器	0x01EC	R/W
LP_ANA_TOUCH_PAD14_TH1_REG	触摸管脚 14 在采样频率模式 1 下的触摸阈值配置寄存器	0x01F0	R/W
LP_ANA_TOUCH_PAD14_TH2_REG	触摸管脚 14 在采样频率模式 2 下的触摸阈值配置寄存器	0x01F4	R/W
LP_ANA_DATE_REG	版本控制寄存器	0x03FC	R/W

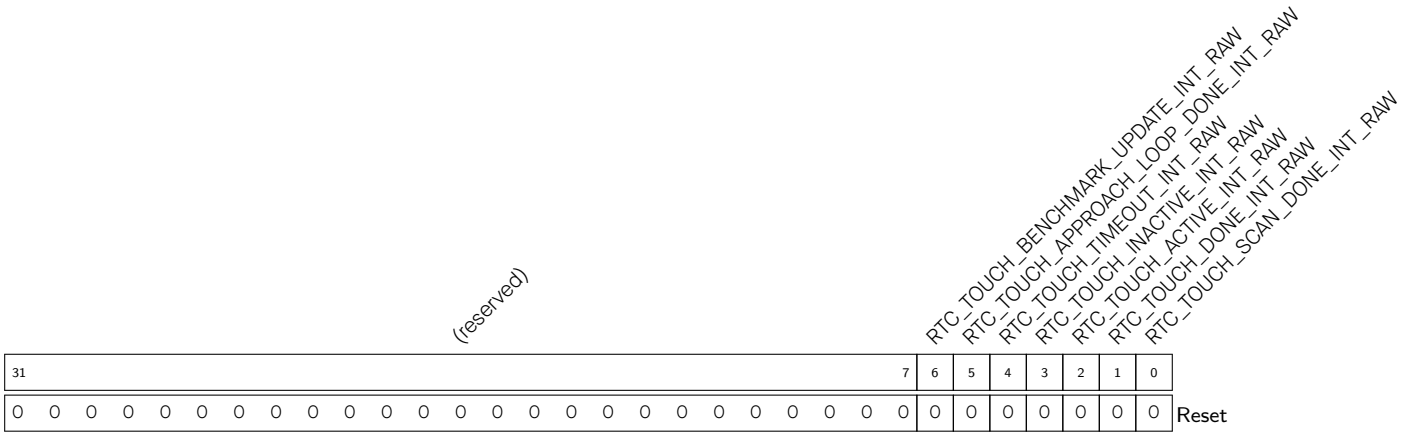
63.7 寄存器

63.7.1 中断与状态寄存器

本小节的所有地址均为相对于触摸传感器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 63.1. RTC_TOUCH_INT_RAW_REG (0x0000)



RTC_TOUCH_SCAN_DONE_INT_RAW TOUCH_SCAN_DONE_INT 中断的原始中断状态。(R/ SS/WTC)

RTC_TOUCH_DONE_INT_RAW TOUCH_DONE_INT 中断的原始中断状态。(R/SS/WTC)

RTC_TOUCH_ACTIVE_INT_RAW TOUCH_ACTIVE_INT 中断的原始中断状态。(R/SS/WTC)

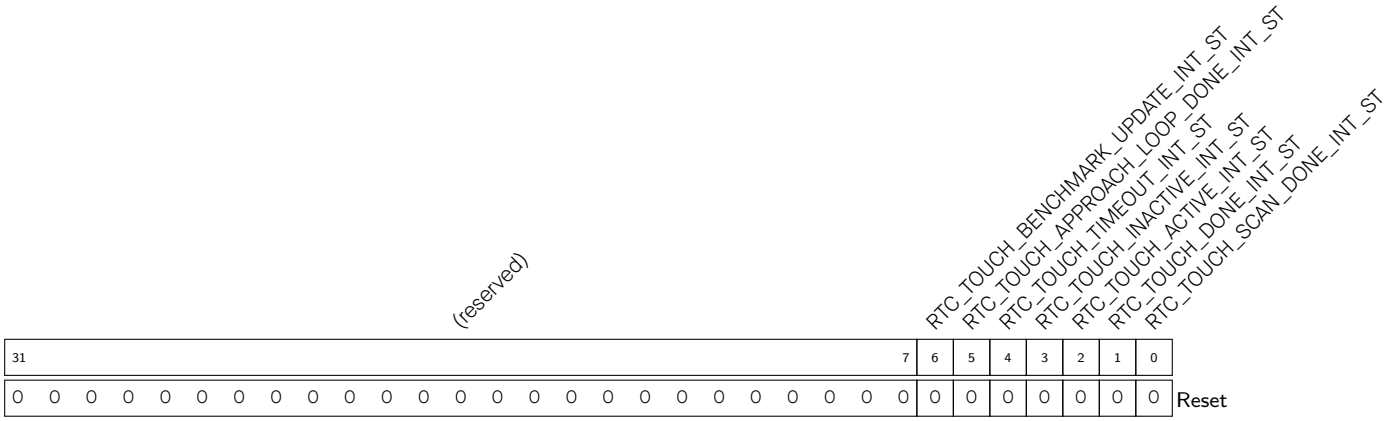
RTC_TOUCH_INACTIVE_INT_RAW TOUCH_INACTIVE_INT 中断的原始中断状态。(R/SS/WTC)

RTC_TOUCH_TIMEOUT_INT_RAW TOUCH_TIMEOUT_INT 中断的原始中断状态。(R/SS/WTC)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_RAW TOUCH_APPROACH_LOOP_DONE_INT 中断的原始中断状态。(R/SS/WTC)

RTC_TOUCH_BENCHMARK_UPDATE_INT_RAW TOUCH_BENCHMARK_UPDATE_INT 中断的原始中断状态。(R/SS/WTC)

Register 63.2. RTC_TOUCH_INT_ST_REG (0x0004)



RTC_TOUCH_SCAN_DONE_INT_ST TOUCH_DONE_INT 中断的屏蔽中断状态。(RO)

RTC_TOUCH_DONE_INT_ST TOUCH_DONE_INT 中断的屏蔽中断状态。(RO)

RTC_TOUCH_ACTIVE_INT_ST TOUCH_ACTIVE_INT 中断的屏蔽中断状态。(RO)

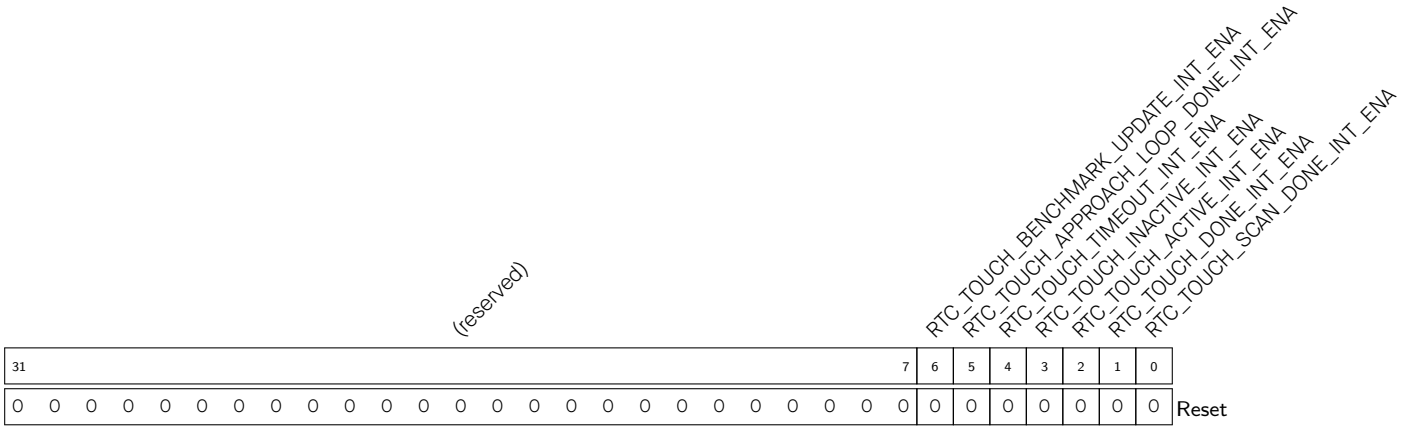
RTC_TOUCH_INACTIVE_INT_ST TOUCH_INACTIVE_INT 中断的屏蔽中断状态。(RO)

RTC_TOUCH_TIMEOUT_INT_ST TOUCH_TIMEOUT_INT 中断的屏蔽中断状态。(RO)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_ST TOUCH_APPROACH_LOOP_DONE_INT 中断的屏蔽中断状态。(RO)

RTC_TOUCH_BENCHMARK_UPDATE_INT_ST TOUCH_BENCHMARK_UPDATE_INT 中断的屏蔽中断状态。(RO)

Register 63.3. RTC_TOUCH_INT_ENA_REG (0x0008)



RTC_TOUCH_SCAN_DONE_INT_ENA 写 1 使能 TOUCH_DONE_INT 中断。(R/W)

RTC_TOUCH_DONE_INT_ENA 写 1 使能 TOUCH_DONE_INT 中断。(R/W)

RTC_TOUCH_ACTIVE_INT_ENA 写 1 使能 TOUCH_ACTIVE_INT 中断。(R/W)

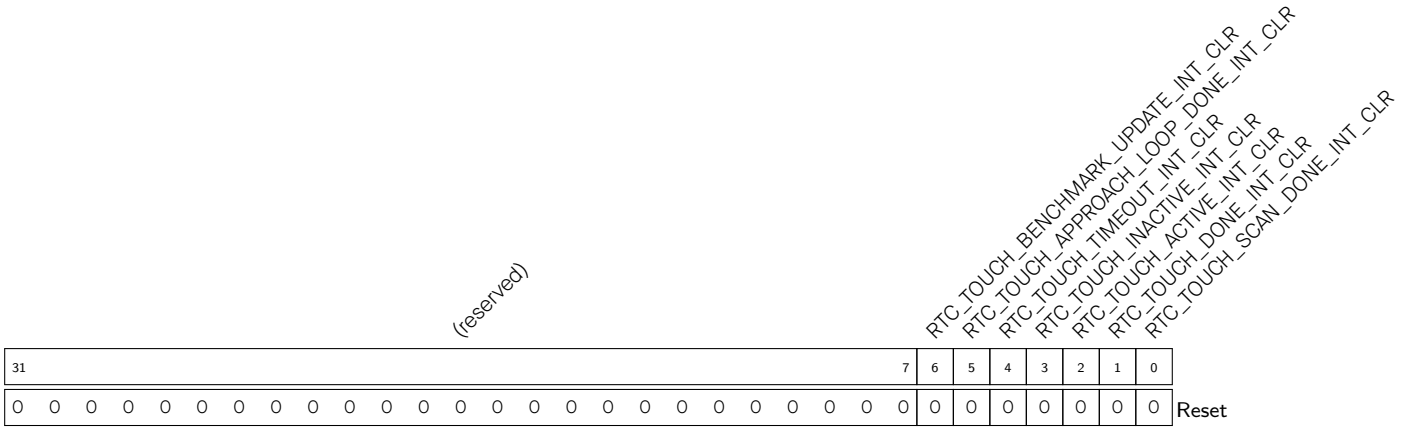
RTC_TOUCH_INACTIVE_INT_ENA 写 1 使能 TOUCH_INACTIVE_INT 中断。(R/W)

RTC_TOUCH_TIMEOUT_INT_ENA 写 1 使能 TOUCH_TIMEOUT_INT 中断。(R/W)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_ENA 写 1 使能 TOUCH_APPROACH_LOOP_DONE_INT 中断。(R/W)

RTC_TOUCH_BENCHMARK_UPDATE_INT_ENA 写 1 使能 TOUCH_BENCHMARK_UPDATE_INT 中断。(R/W)

Register 63.4. RTC_TOUCH_INT_CLR_REG (0x000C)



RTC_TOUCH_SCAN_DONE_INT_CLR 写 1 清除 TOUCH_DONE_INT 中断。(WT)

RTC_TOUCH_DONE_INT_CLR 写 1 清除 TOUCH_DONE_INT 中断。(WT)

RTC_TOUCH_ACTIVE_INT_CLR 写 1 清除 TOUCH_ACTIVE_INT 中断。(WT)

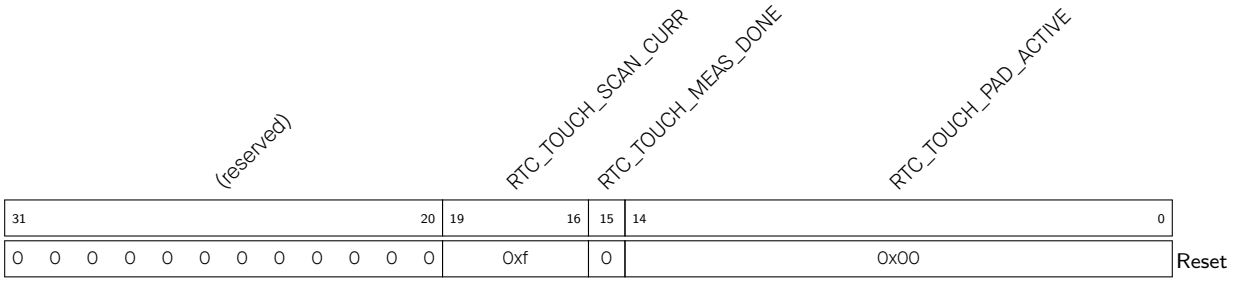
RTC_TOUCH_INACTIVE_INT_CLR 写 1 清除 TOUCH_INACTIVE_INT 中断。(WT)

RTC_TOUCH_TIMEOUT_INT_CLR 写 1 清除 TOUCH_TIMEOUT_INT 中断。(WT)

RTC_TOUCH_APPROACH_LOOP_DONE_INT_CLR 写 1 清除 TOUCH_APPROACH_LOOP_DONE_INT 中断。(WT)

RTC_TOUCH_BENCHMARK_UPDATE_INT_CLR 写 1 清除 TOUCH_BENCHMARK_UPDATE_INT 中断。(WT)

Register 63.5. RTC_TOUCH_CHN_STATUS_REG (0x0010)



RTC_TOUCH_PAD_ACTIVE 表示触摸管脚是否检测到触摸状态，每一位均代表一个触摸管脚的触摸状态，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

0：未检测到触摸状态

1：检测到触摸状态

(RO)

RTC_TOUCH_MEAS_DONE 表示触摸管脚是否在所有频率模式下都测量完成。

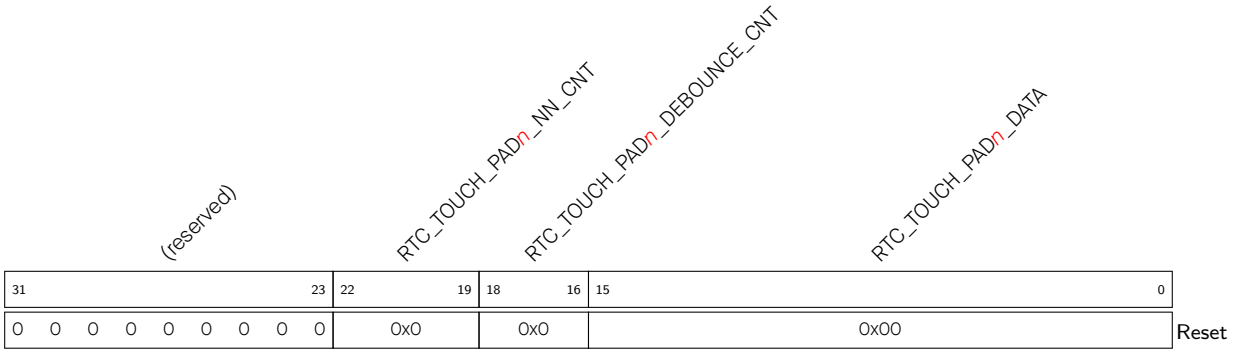
0：未完成

1：完成

(RO)

RTC_TOUCH_SCAN_CURR 表示当前被扫描的触摸管脚的序号。(RO)

Register 63.6. RTC_TOUCH_STATUS_*n*_REG (*n*: 1-14) (0x0018+0x4*(*n*-1))

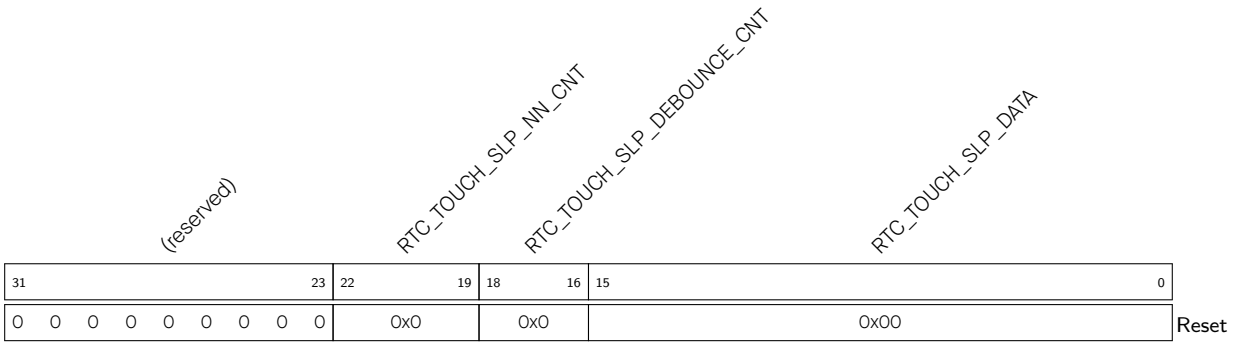


RTC_TOUCH_PAD*n*_DATA 表示触摸管脚 *n* 采样的原始数据或平滑触摸数据或基准数据。(RO)

RTC_TOUCH_PAD*n*_DEBOUNCE_CNT 表示触摸管脚 *n* 连续检测到触摸或触摸释放的累计次数。(RO)

RTC_TOUCH_PAD*n*_NN_CNT 表示触摸管脚 *n* 连续检测到平滑触摸数据小于参考阈值的累计次数。(RO)

Register 63.7. RTC_TOUCH_STATUS_15_REG (0x0050)

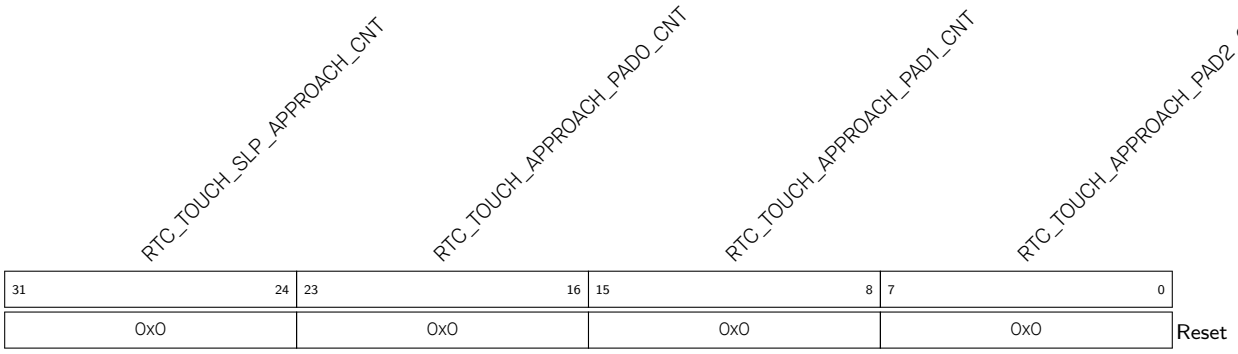


RTC_TOUCH_SLP_DATA 表示休眠模式下触摸管脚采样的平滑触摸数据或基准数据。(RO)

RTC_TOUCH_SLP_DEBOUNCE_CNT 表示休眠模式下触摸管脚连续检测到触摸或触摸释放的累计次数。(RO)

RTC_TOUCH_SLP_NN_CNT 表示休眠模式下触摸管脚连续检测到平滑触摸数据小于参考阈值的累计次数。(RO)

Register 63.8. RTC_TOUCH_STATUS_16_REG (0x0054)



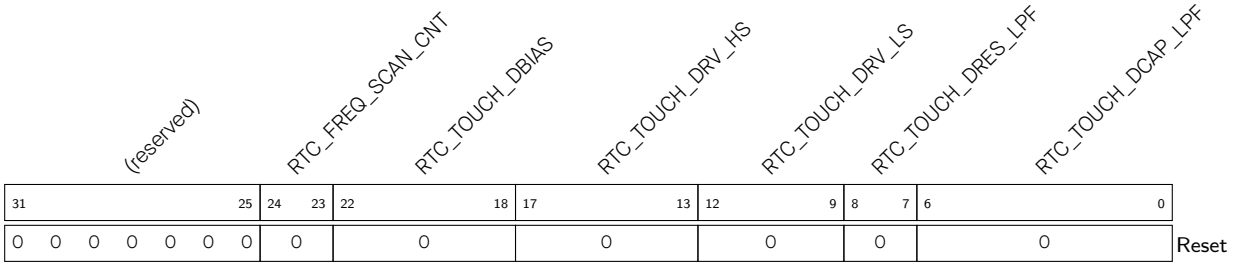
RTC_TOUCH_APPROACH_PAD2_CNT 表示接近模式下触摸管脚 2 采样的累计次数。(RO)

RTC_TOUCH_APPROACH_PAD1_CNT 表示接近模式下触摸管脚 1 采样的累计次数。(RO)

RTC_TOUCH_APPROACH_PAD0_CNT 表示接近模式下触摸管脚 0 采样的累计次数。(RO)

RTC_TOUCH_SLP_APPROACH_CNT 表示接近模式下休眠管脚采样的累计次数。(RO)

Register 63.9. RTC_TOUCH_STATUS_17_REG (0x0058)



RTC_TOUCH_DCAP_LPF 表示当前触摸管脚的 DCAP_LPF 值。(RO)

RTC_TOUCH_DRES_LPF 表示当前触摸管脚的 DRES_LPF 值。(RO)

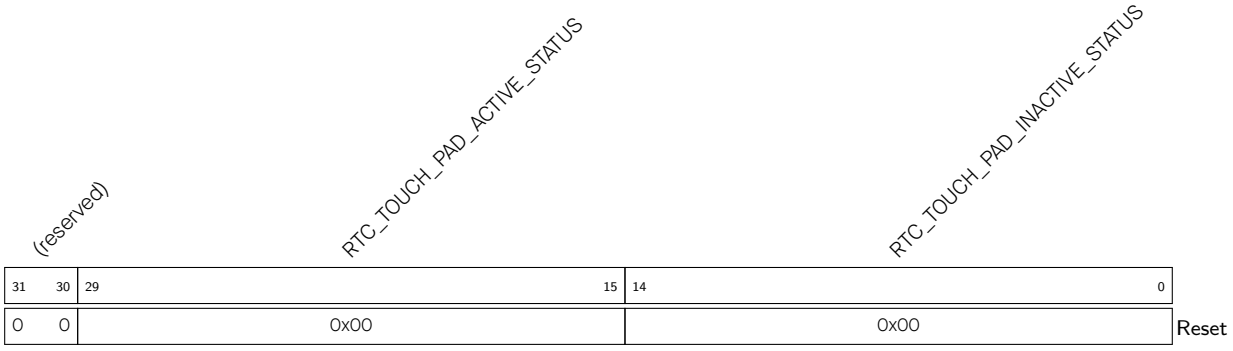
RTC_TOUCH_DRV_LS 表示当前触摸管脚的 DRV_LS 值。(RO)

RTC_TOUCH_DRV_HS 表示当前触摸管脚的 DRV_HS 值。(RO)

RTC_TOUCH_DBIAS 表示当前触摸管脚的 DBIAS 值。(RO)

RTC_FREQ_SCAN_CNT 表示当前触摸管脚的 SCAN_CNT 值。(RO)

Register 63.10. RTC_TOUCH_CHN_TMP_STATUS_REG (0x005C)



RTC_TOUCH_PAD_INACTIVE_STATUS 表示触摸管脚是否检测到触摸板从触摸状态到触摸释放状态的过程，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

0：未检测到触摸释放

1：检测到触摸释放状态

(RO)

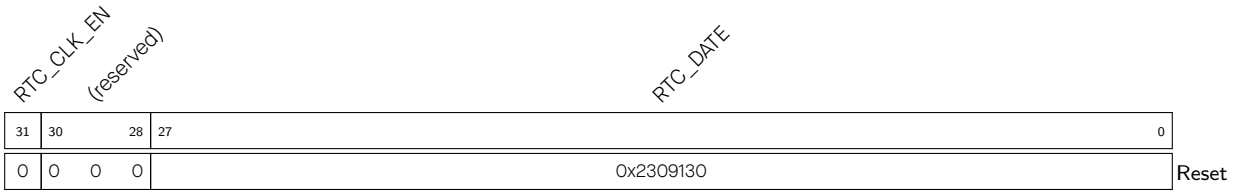
RTC_TOUCH_PAD_ACTIVE_STATUS 表示触摸管脚是否检测到被触摸的过程，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

0：未检测到被触摸

1：检测到触摸管脚被触摸

(RO)

Register 63.11. RTC_TOUCH_DATE_REG (0x0100)



RTC_CLK_EN 配置是否使能寄存器读写时钟。

0：不使能

1：使能

(R/W)

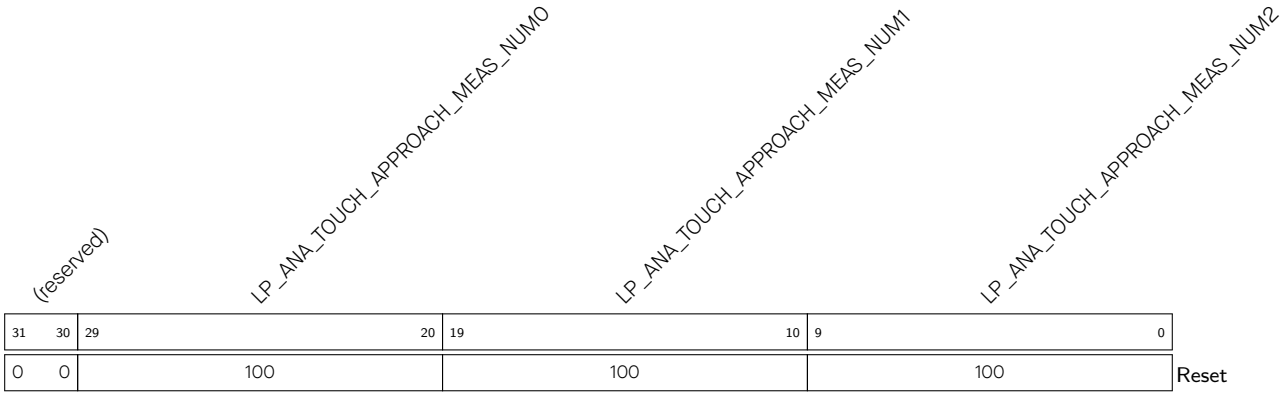
RTC_DATE 版本控制寄存器。(R/W)

63.7.2 配置寄存器

本小节的所有地址均为相对于 LP 模拟外设器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 63.12. LP_ANA_TOUCH_APPROACH_WORK_MEAS_NUM_REG (0x00FC)

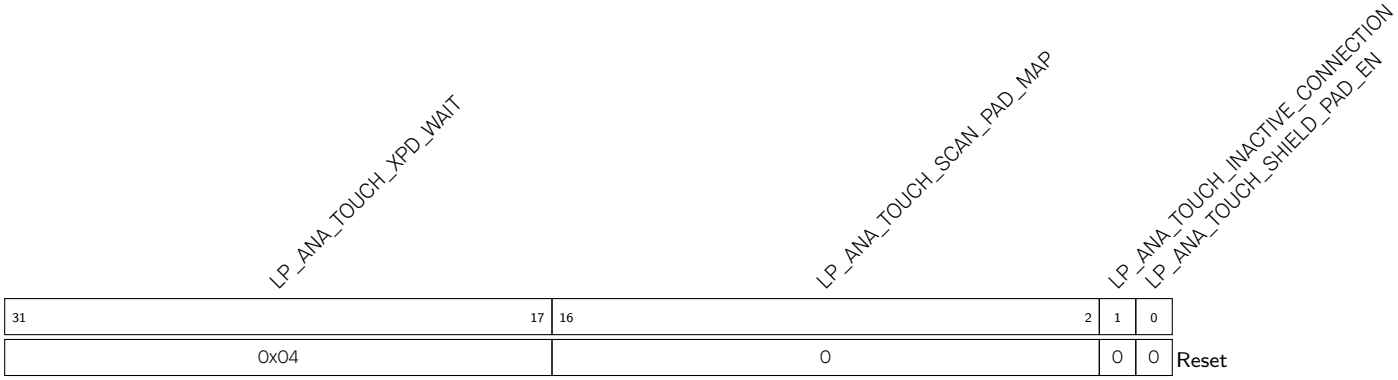


LP_ANA_TOUCH_APPROACH_MEAS_NUM2 配置接近模式在采样频率模式为 2 时的测量次数。(R/W)

LP_ANA_TOUCH_APPROACH_MEAS_NUM1 配置接近模式在采样频率模式为 1 时的测量次数。(R/W)

LP_ANA_TOUCH_APPROACH_MEAS_NUM0 配置接近模式在采样频率模式为 0 时的测量次数。(R/W)

Register 63.13. LP_ANA_TOUCH_SCAN_CTRL1_REG (0x0100)



LP_ANA_TOUCH_SHIELD_PAD_EN 配置是否开启防潮功能。

0：不开启

1：开启

(R/W)

LP_ANA_TOUCH_INACTIVE_CONNECTION 配置是否上电在扫描模式下参与测量的触摸管脚。

0：不上电

1：上电

(R/W)

LP_ANA_TOUCH_SCAN_PAD_MAP 配置扫描模式下参与测量的触摸管脚，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

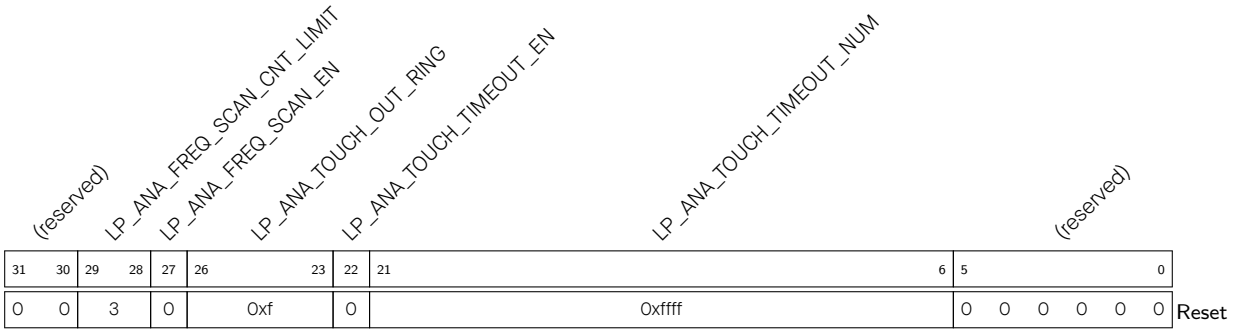
0：无效

1：使能该管脚

(R/W)

LP_ANA_TOUCH_XPD_WAIT 配置启动测量后的上电等待时间。(R/W)

Register 63.14. LP_ANA_TOUCH_SCAN_CTRL2_REG (0x0104)



LP_ANA_TOUCH_TIMEOUT_NUM 配置测量超时次数。(R/W)

LP_ANA_TOUCH_TIMEOUT_EN 配置是否使能测量超时。

0：不使能

1：使能

(R/W)

LP_ANA_TOUCH_OUT_RING 配置选择用于遇水保护功能的触摸管脚。可以配置为 1 ~ 14，其他值无效。(R/W)

LP_ANA_FREQ_SCAN_EN 配置是否使能跳频模式。

0：不使能

1：使能

(R/W)

LP_ANA_FREQ_SCAN_CNT_LIMIT 配置跳频次数。

0：1

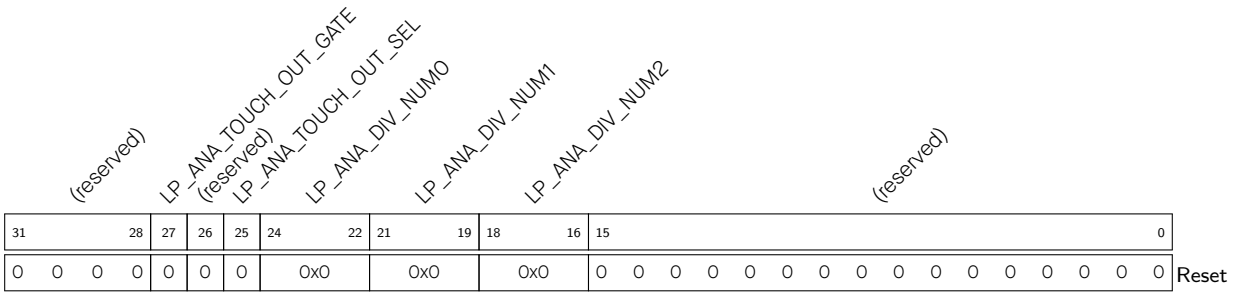
1：2

2：3

3：无效

(R/W)

Register 63.15. LP_ANA_TOUCH_WORK_REG (0x0108)



LP_ANA_DIV_NUM2 配置采样频率模式为 2 时是否分频及对应的 TOUCH_OUT 信号分频系数。

bit[0]：配置是否分频。

0：不分频

1：分频

bit[1-2]：配置 TOUCH_OUT 信号分频系数。

0：0

1：2

2：4

3：6

(R/W)

LP_ANA_DIV_NUM1 配置采样频率模式为 1 时是否分频及对应的 TOUCH_OUT 信号分频系数。具体配置参考 LP_ANA_DIV_NUM2。(R/W)

LP_ANA_DIV_NUM0 配置采样频率模式为 0 时是否分频及对应的 TOUCH_OUT 信号分频系数。具体配置参考 LP_ANA_DIV_NUM2。(R/W)

LP_ANA_TOUCH_OUT_SEL 配置 TOUCH_OUT 信号作为时钟或数据使用。

0：TOUCH_OUT 信号作为数据使用

1：TOUCH_OUT 信号作为时钟使用

(R/W)

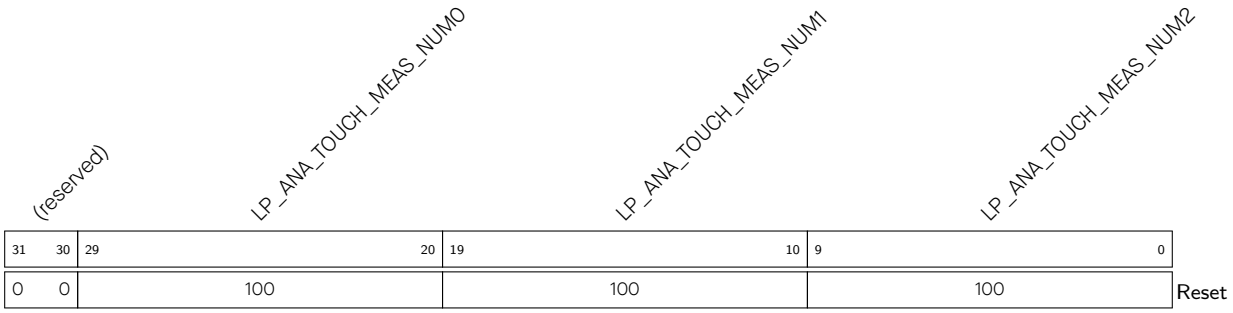
LP_ANA_TOUCH_OUT_GATE 配置是否开启 TOUCH_OUT 信号分频后的门控。

0：不开启

1：开启

(R/W)

Register 63.16. LP_ANA_TOUCH_WORK_MEAS_NUM_REG (0x010C)

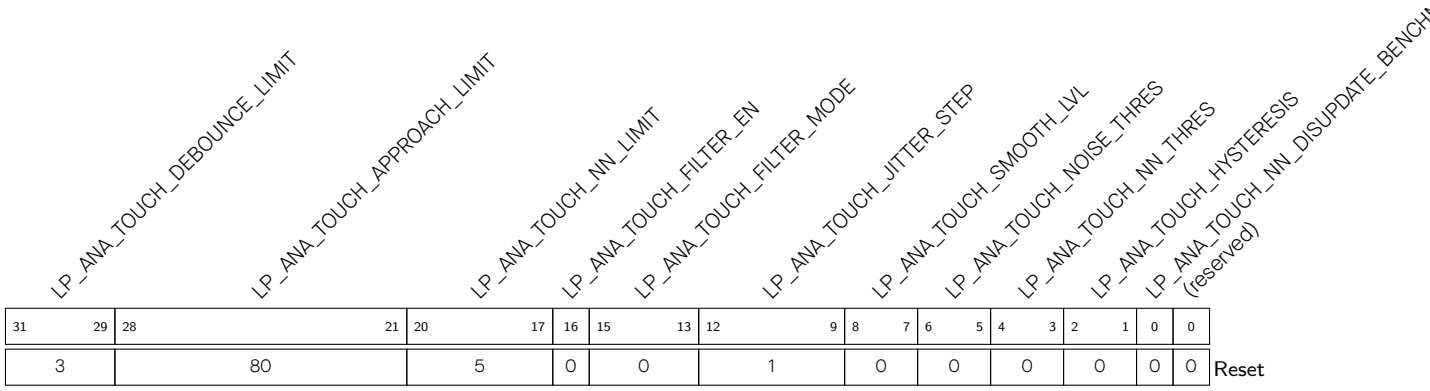


LP_ANA_TOUCH_MEAS_NUM2 配置正常模式下采样频率模式为 2 时的测量次数。(R/W)

LP_ANA_TOUCH_MEAS_NUM1 配置正常模式下采样频率模式为 1 时的测量次数。(R/W)

LP_ANA_TOUCH_MEAS_NUM0 配置正常模式下采样频率模式为 0 时的测量次数。(R/W)

Register 63.17. LP_ANA_TOUCH_FILTER1_REG (0x0110)



LP_ANA_TOUCH_NN_DISUPDATE_BENCHMARK_EN 配置是否关闭基准数据修正功能。

0：不关闭

1：关闭

(R/W)

LP_ANA_TOUCH_HYSTERESIS 配置采样迟滞。(R/W)

LP_ANA_TOUCH_NN_THRES 配置反相噪声阈值。(R/W)

LP_ANA_TOUCH_NOISE_THRES 配置噪声阈值。(R/W)

LP_ANA_TOUCH_SMOOTH_LVL 配置平滑触摸数据的 IIR 滤波模式。(R/W)

LP_ANA_TOUCH_JITTER_STEP 配置基准数据滤波模式下的抖动参数。(R/W)

LP_ANA_TOUCH_FILTER_MODE 配置基准数据滤波模式。(R/W)

LP_ANA_TOUCH_FILTER_EN 配置是否使能基准数据滤波模式。

0：不使能

1：使能

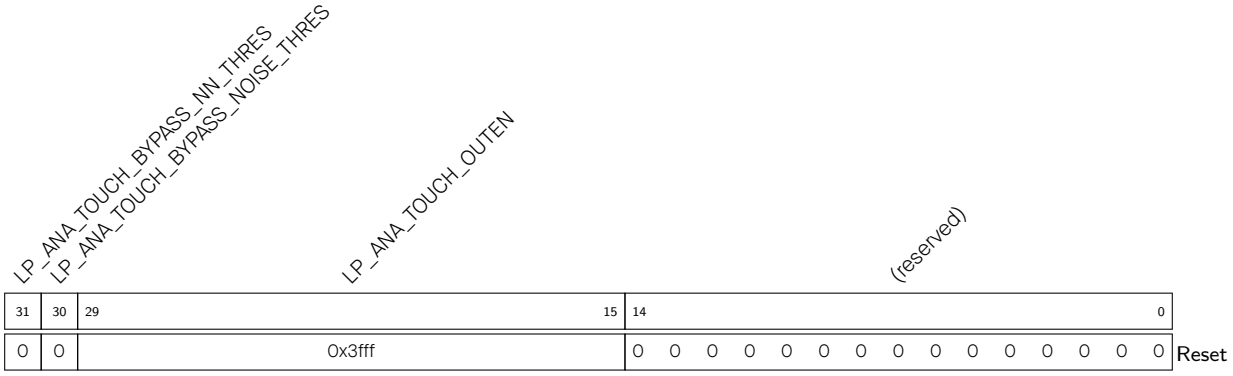
(R/W)

LP_ANA_TOUCH_NN_LIMIT 配置超过反相噪声阈值的连续采样次数。(R/W)

LP_ANA_TOUCH_APPROACH_LIMIT 配置接近模式采样次数。(R/W)

LP_ANA_TOUCH_DEBOUNCE_LIMIT 配置触摸状态改变连续采样次数。(R/W)

Register 63.18. LP_ANA_TOUCH_FILTER2_REG (0x0114)



LP_ANA_TOUCH_OUTEN 配置是否使能输出 14 个触摸管脚检测结果，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

0：不使能

1：使能

(R/W)

LP_ANA_TOUCH_BYPASS_NOISE_THRES 配置是否使能平滑数据 bypass 模式。

0：不使能

1：使能

(R/W)

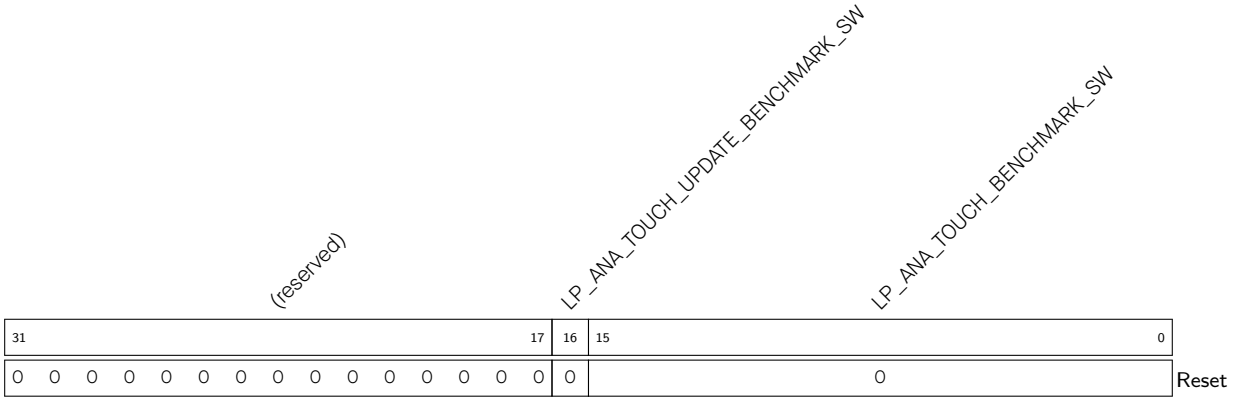
LP_ANA_TOUCH_BYPASS_NN_THRES 配置是否使能反相噪声的 bypass 模式。

0：不使能

1：使能

(R/W)

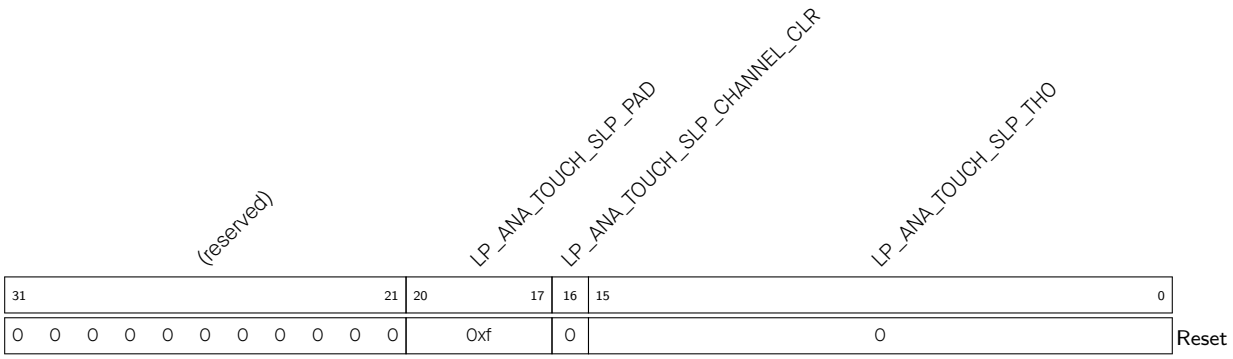
Register 63.19. LP_ANA_TOUCH_FILTER3_REG (0x0118)



LP_ANA_TOUCH_BENCHMARK_SW 配置软件写入基准数据。(R/W)

LP_ANA_TOUCH_UPDATE_BENCHMARK_SW 写 1 使能软件写入基准数据。(WT)

Register 63.20. LP_ANA_TOUCH_SLP0_REG (0x011C)



LP_ANA_TOUCH_SLP_TH0 配置休眠模式下在采样频率模式为 0 时的触碰阈值。(R/W)

LP_ANA_TOUCH_SLP_CHANNEL_CLR 写 1 清理休眠模式下基准数据。(WT)

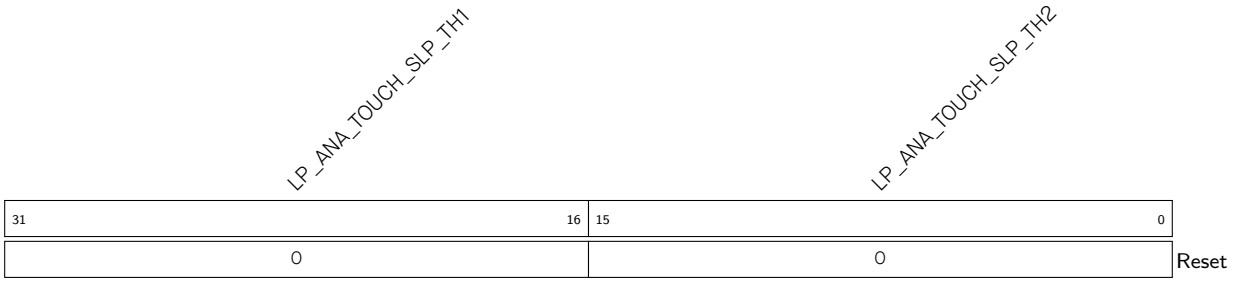
LP_ANA_TOUCH_SLP_PAD 配置选择触摸传感器为休眠模式。位 1 ~ 14 分别对应触摸传感器 1 ~ 14，其他位无效。

0：无效

1：选择该触摸传感器进入休眠模式

(R/W)

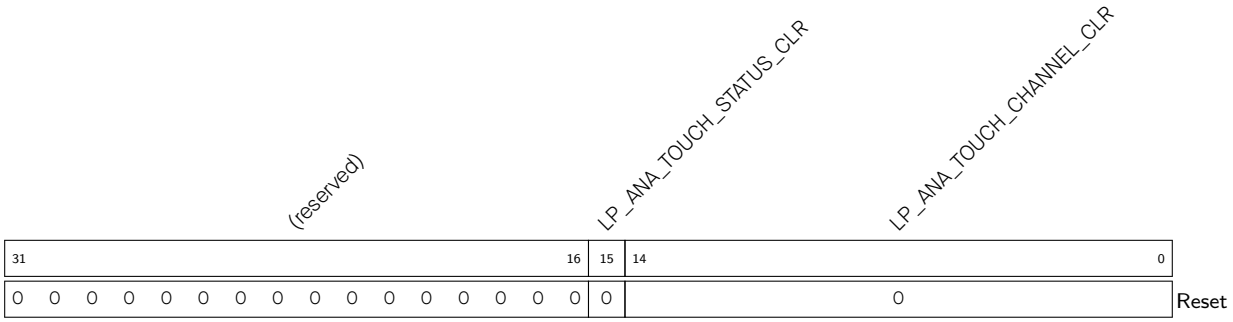
Register 63.21. LP_ANA_TOUCH_SLP1_REG (0x0120)



LP_ANA_TOUCH_SLP_TH2 配置休眠模式下在采样频率模式为 2 时的触碰阈值。(R/W)

LP_ANA_TOUCH_SLP_TH1 配置休眠模式下在采样频率模式为 1 时的触碰阈值。(R/W)

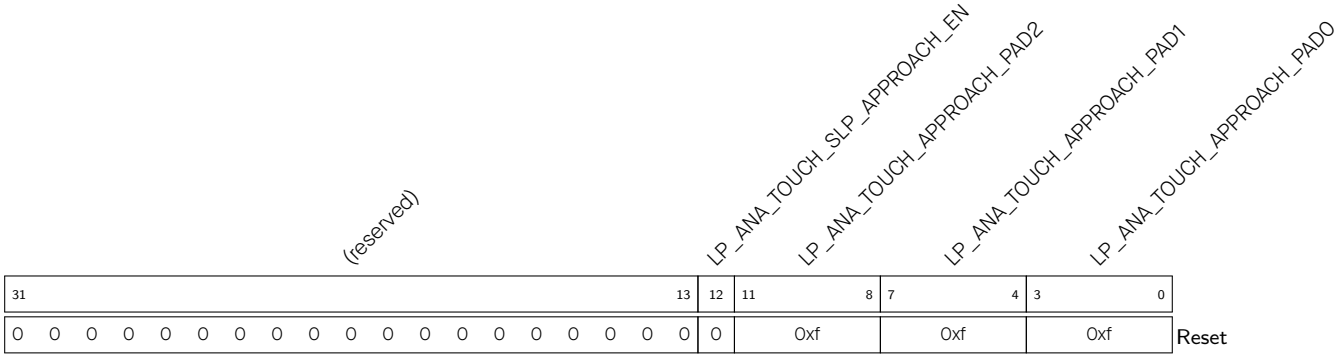
Register 63.22. LP_ANA_TOUCH_CLR_REG (0x0124)



LP_ANA_TOUCH_CHANNEL_CLR 写 1 独立清零 14 个触摸传感器的触摸状态，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。(WT)

LP_ANA_TOUCH_STATUS_CLR 写 1 清零触摸状态。(WT)

Register 63.23. LP_ANA_TOUCH_APPROACH_REG (0x0128)



LP_ANA_TOUCH_APPROACH_PAD0 配置接近模式使用的第一个传感器触摸管脚，可以配置为 1 ~ 14，其他值无效。(R/W)

LP_ANA_TOUCH_APPROACH_PAD1 配置接近模式使用的第二个传感器触摸管脚，可以配置为 1 ~ 14，其他值无效。(R/W)

LP_ANA_TOUCH_APPROACH_PAD2 配置接近模式使用的第三个传感器触摸管脚，可以配置为 1 ~ 14，其他值无效。(R/W)

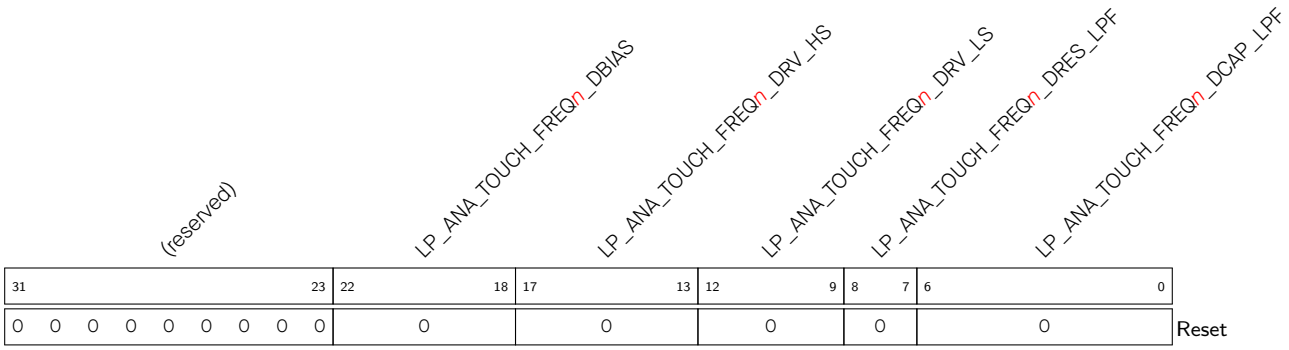
LP_ANA_TOUCH_SLP_APPROACH_EN 配置是否使能接近模式。

0：不使能

1：使能

(R/W)

Register 63.24. LP_ANA_TOUCH_FREQ*n*_SCAN_PARA_REG (*n*: 0-2) (0x012C+0x4**n*)



LP_ANA_TOUCH_FREQ*n*_DCAP_LPF 配置采样频率模式为 *n* 时触摸传感器低通滤波器参数，调节范围为 0 ~ 2.54 pF，调节精度为 20 fF。(R/W)

LP_ANA_TOUCH_FREQ*n*_DRES_LPF 配置采样频率模式为 *n* 时触摸传感器低通滤波器参数。

0: 0 kΩ

1: 3 kΩ

2: 4.5 kΩ

3: 7.5 kΩ

(R/W)

LP_ANA_TOUCH_FREQ*n*_DRV_LS 配置采样频率模式为 *n* 时触摸传感器的 DRV_LS 参数。(R/W)

LP_ANA_TOUCH_FREQ*n*_DRV_HS 配置采样频率模式为 *n* 时触摸传感器的 DRV_HS 参数。(R/W)

LP_ANA_TOUCH_FREQ*n*_DBIAS 配置采样频率模式为 *n* 时是否开启防潮功能及配置触摸传感器内部电压。

bit[0]：配置在采样频率模式为 *n* 时是否开启防潮功能。

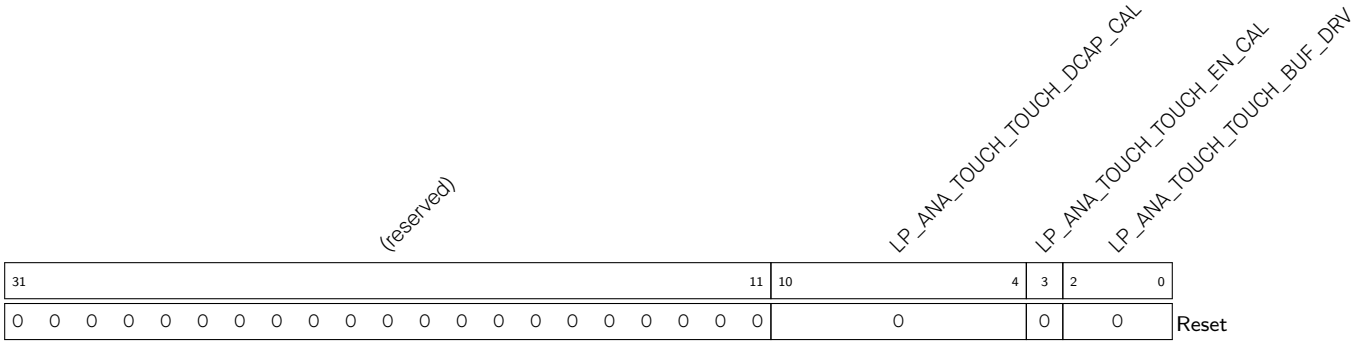
0：不开启

1：开启

bit[1-4]：配置采样频率模式为 *n* 时触摸传感器的内部电压。

(R/W)

Register 63.25. LP_ANA_TOUCH_ANA_PARA_REG (0x0138)



LP_ANA_TOUCH_TOUCH_BUF_DRV 配置 BUF_DRV 的值。(R/W)

LP_ANA_TOUCH_TOUCH_EN_CAL 配置是否使能触摸传感器内部电容测试模式。

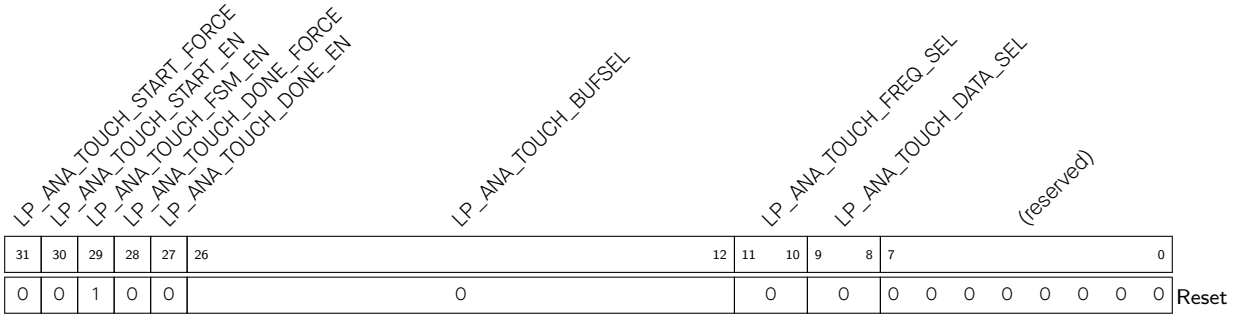
0：不使能

1：使能

(R/W)

LP_ANA_TOUCH_TOUCH_DCAP_CAL 配置连接到触摸管脚上的内部电容，默认值为 0，调节范围为 0 ~ 127 pF，调节步长为 1。(R/W)

Register 63.26. LP_ANA_TOUCH_MUX0_REG (0x013C)



LP_ANA_TOUCH_DATA_SEL 配置返回值类型。

0/1：原始触摸数据

2：基准数据

3：平滑触摸数据

(R/W)

LP_ANA_TOUCH_FREQ_SEL 配置返回值所对应的采样频率模式。

0：0

1：1

2：2

3：无效

(R/W)

LP_ANA_TOUCH_BUFSEL 配置是否使能某一触摸管脚用于防潮功能，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

0：不使能

1：使能

(R/W)

LP_ANA_TOUCH_DONE_EN 配置是否使能测量的 DONE 信号由软件触发。

0：不使能

1：使能

(R/W)

LP_ANA_TOUCH_DONE_FORCE 配置是否产生 DONE 信号，结束当前测量操作。

0：不产生

1：产生

(R/W)

LP_ANA_TOUCH_FSM_EN 配置触摸管脚上下电和启动的控制源。

0：由软件控制

1：由硬件控制

(R/W)

LP_ANA_TOUCH_START_EN 配置是否选择测量的 START 信号由软件触发。

0：不使能

1：使能

(R/W)

见下页...

Register 63.26. LP_ANA_TOUCH_MUX0_REG (0x013C)

接上页...

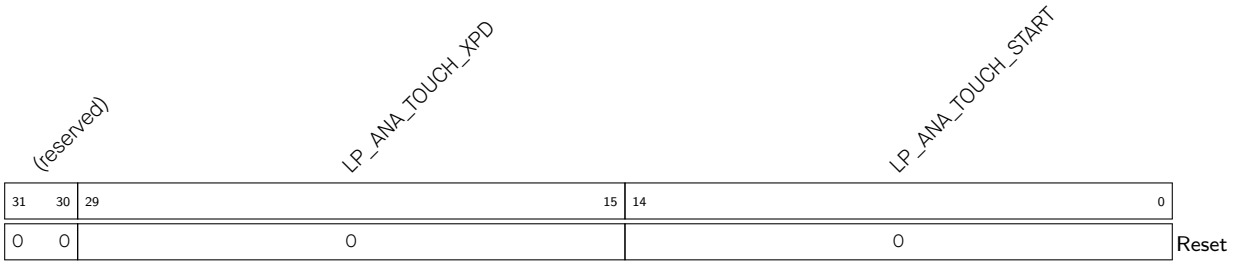
LP_ANA_TOUCH_START_FORCE 配置是否产生 START 信号，开启一次测量操作。

0：不产生

1：产生

(R/W)

Register 63.27. LP_ANA_TOUCH_MUX1_REG (0x0140)



LP_ANA_TOUCH_START 配置是否启动 14 个触摸管脚，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

0：无效

1：启动

(R/W)

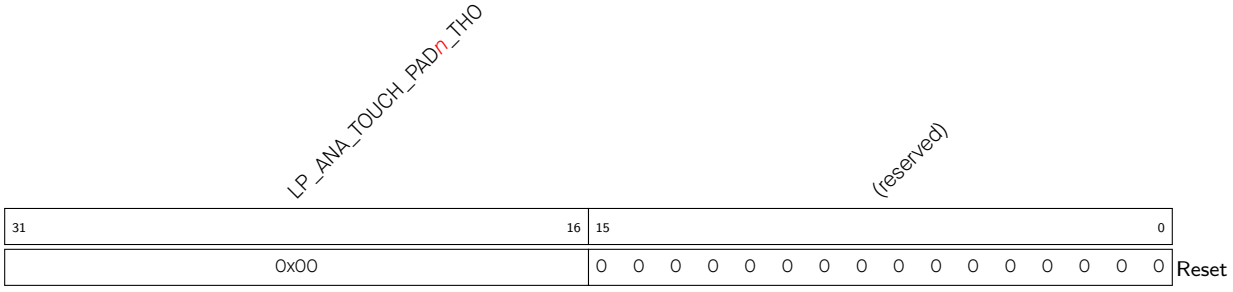
LP_ANA_TOUCH_XPD 配置 14 个触摸管脚的上下电，位 1 ~ 14 分别对应触摸管脚 1 ~ 14，其他位无效。

0：不上电

1：上电

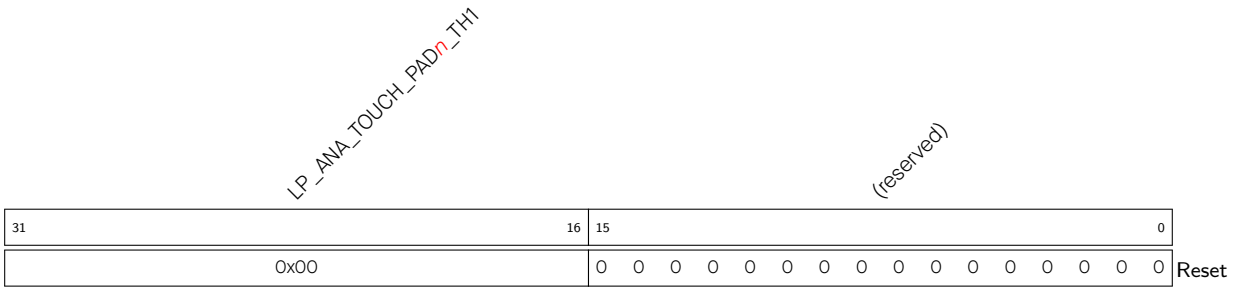
(R/W)

Register 63.28. LP_ANA_TOUCH_PAD*n*_TH0_REG (*n*: 1-14) (0x0150+0xC*(*n*-1))



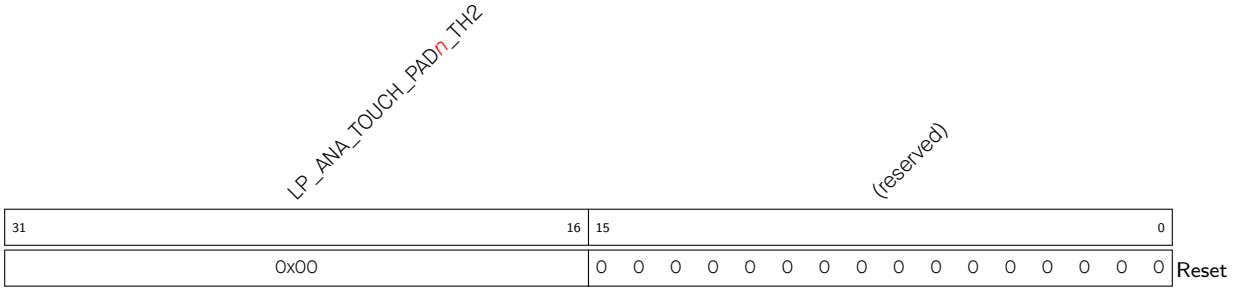
LP_ANA_TOUCH_PAD*n*_TH0 配置触摸传感器 *n* 在采样频率模式为 0 时的触碰阈值。(R/W)

Register 63.29. LP_ANA_TOUCH_PADn_TH1_REG (n: 1-14) (0x0154+0xC*(n-1))



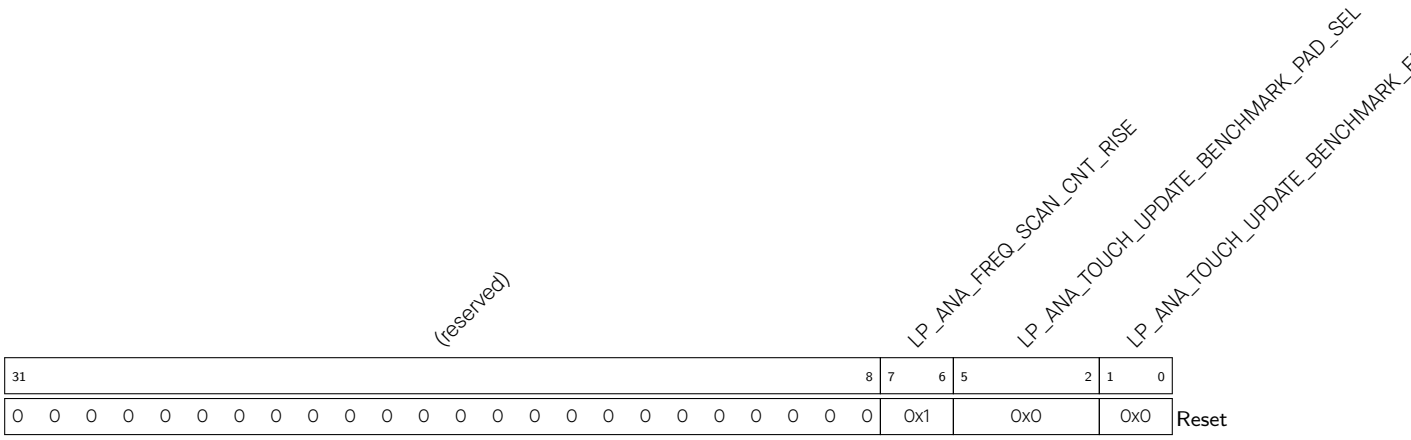
LP_ANA_TOUCH_PADn_TH1 配置触摸传感器 n 在采样频率模式为 1 时的触碰阈值。(R/W)

Register 63.30. LP_ANA_TOUCH_PADn_TH2_REG (n: 1-14) (0x0158+0xC*(n-1))



LP_ANA_TOUCH_PADn_TH2 配置触摸传感器 n 在采样频率模式为 2 时的触碰阈值。(R/W)

Register 63.31. LP_ANA_TOUCH_CTRL_REG (0x01FC)

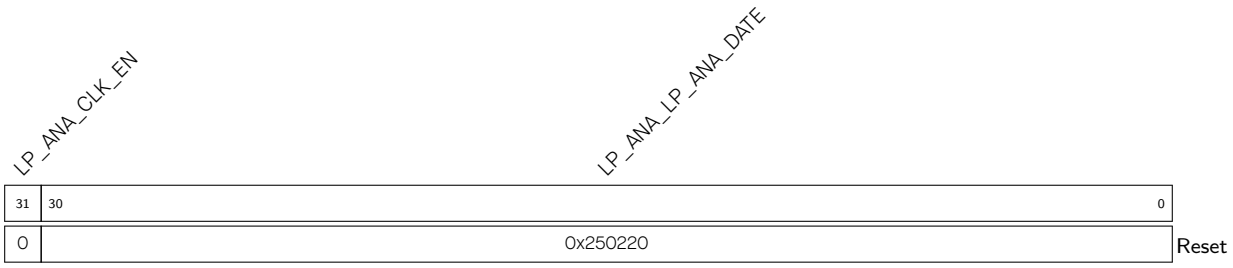


LP_ANA_TOUCH_UPDATE_BENCHMARK_FREQ_SEL 配置软件用于更新基准数据的频率点。(R/W)

LP_ANA_TOUCH_UPDATE_BENCHMARK_PAD_SEL 配置软件用于更新基准数据的通道。(R/W)

LP_ANA_FREQ_SCAN_CNT_RISE 配置在跳频模式下用于触摸检测的触发频率点数量。(R/W)

Register 63.32. LP_ANA_DATE_REG (0x03FC)



LP_ANA_LP_ANA_DATE 版本控制寄存器。(R/W)

LP_ANA_CLK_EN 配置是否使能访问寄存器的时钟。

0：不使能

1：使能

(R/W)

第 64 章

温度传感器 (TSENS)

64.1 介绍

ESP32-P4 配备了一个温度传感器，用于实时测量芯片内部温度。温度传感器能将输出电压转换成数字值，并具有补偿温度偏移的功能。

64.2 特性

温度传感器的主要特性包括：

- 支持软件触发测量温度，且一旦触发后，传感器可持续测量温度，软件可实时读取数据
- 支持硬件触发自动监测温度，支持两种自动监测唤醒模式
- 支持根据使用环境配置温度偏移，提高测试精度
- 温度测量范围可配置
- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

64.3 结构概览

温度传感器的内部结构如图 64.3-1 所示。

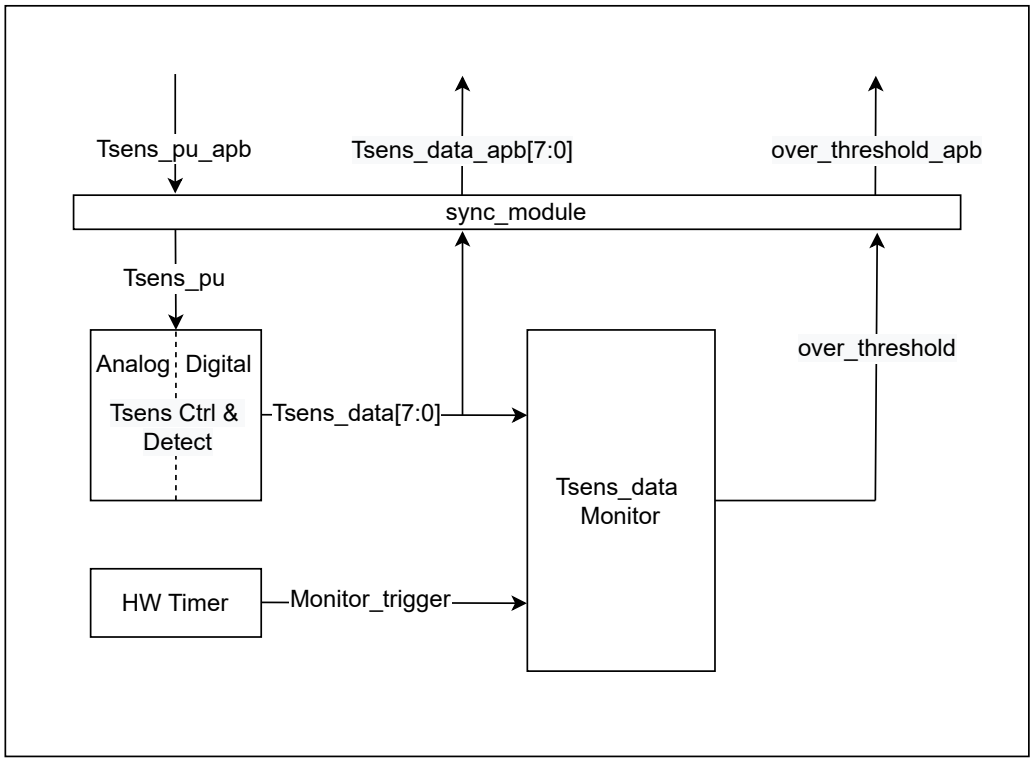


图 64.3-1. 温度传感器结构框图

根据图 64.3-1，温度传感器模块主要包括以下功能模块：

- Tsens Ctrl & Detect：温度传感器
- HW Timer（硬件定时器）：用于触发温度监测
- Tsens_data 监测器：用于监测温度是否超出阈值
- sync_module：APB 时钟域与温度传感器时钟域间的同步模块

64.4 功能描述

64.4.1 温度传感器上电

通过设置 TSENS_POWER_UP 给温度传感器上电。

64.4.2 时钟管理

温度传感器只有一个时钟源：LP_PERI_CLK，并且没有时钟分频寄存器。

64.4.3 自动监测唤醒模式

温度传感器的自动温度监测功能有两种唤醒模式：绝对值模式和变化量模式，通过寄存器 TSENS_WAKEUP_MODE 进行选择。

- 绝对值模式：

  - 用于监测当前温度的绝对值。配置 TSENS_WAKEUP_TH_LOW 和 TSENS_WAKEUP_TH_HIGH 来决定温度阈值，温度超过高阈值或小于低阈值时将触发唤醒。

- 变化量模式：

  - 用于监测温度的变化量。若连续两次采样的温度值增量超过 TSENS_WAKEUP_TH_HIGH 的配置值或减少量超过 TSENS_WAKEUP_TH_LOW 的配置值，将触发唤醒。例如当 TSENS_WAKEUP_TH_LOW 的值为 8 时，若连续两次温度采样分别为 28 和 19，即温度减少量为 9，将触发唤醒。

64.4.4 温度测量范围与温度偏移

为了提高测试精度，温度传感器的测量范围被设计为 5 个范围，如表 64.4-1 所示。每个测量范围都有一定的测温误差。用户无需关注测温误差，只需选择相应的温度偏移，即可获得校准后的数据。

表 64.4-1. 温度测量范围与温度偏移

测量范围 (°C)	温度偏移
50 ~ 125	–2
20 ~ 100	–1
–10 ~ 80	0
–30 ~ 50	1
–40 ~ 20	2

64.4.5 数据转换

温度传感器的输出值 $VALUE$ 存储在寄存器 TSENS_OUT 中。用户可使用以下公式将输出值 $VALUE$ 转换成实际的温度值 T (°C)。转换公式如下：

$$T = 0.4386 \times VALUE - 27.88 \times offset - 20.52$$

其中，$offset$ 即表 64.4-1 所示温度偏移。

64.5 事件任务矩阵功能

在 ESP32-P4 中，温度传感器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发温度传感器的 ETM 任务，或者通过温度传感器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与温度传感器相关的 ETM 任务和 ETM 事件。

温度传感器可接收的 ETM 任务有：

- TMPSNSR_TASK_START_SAMPLE：该任务被触发时，温度传感器开始采样。
- TMPSNSR_TASK_STOP_SAMPLE：该任务被触发时，温度传感器停止采样。

温度传感器可产生的 ETM 事件有：

- TMPSNSR_EVT_OVER_LIMIT：温度传感器发现温度超过阈值时生成。

在具体应用中，温度传感器的 ETM 事件可以用来触发温度传感器的 ETM 任务，例如，TMPSNSR_EVT_OVER_LIMIT 事件可以触发 TMPSNSR_TASK_STOP_SAMPLE 任务。

64.6 中断

ESP32-P4 的温度传感器可以生成以下中断信号并发送给中断矩阵。

- LP_TSENS_INTR

这个中断信号由以下内部中断源生成：

- TSENS_COCPU_TSENS_WAKE_INT：当温度样本值超过阈值时触发。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 64.8 寄存器列表。

64.7 配置流程

温度传感器可由软件启动，具体配置如下：

1. 置位 TSENS_POWER_UP 使温度传感器上电；
2. 置位 LPPERI_CK_EN_LP_TSENS 使能温度传感器时钟；
3. 等待 TSENS_XPD_WAIT 个时钟周期后，温度传感器的复位释放，开始测量环境温度；
4. 等待 100 μs 后（输出值会随着测量时间的增加而逐渐逼近真实的温度值），直接从 TSENS_OUT 中读取温度值。

如要使用硬件自动温度监测功能，则可在给温度传感器上电之前增加如下步骤：

1. 配置 TSENS_SAMPLE_RATE 设置采样率；
2. 配置 TSENS_WAKEUP_MODE 选择温度监测唤醒模式；
3. 配置 TSENS_WAKEUP_TH_HIGH/LOW 设置温度高低阈值；
4. 置位 TSENS_WAKEUP_EN 启用温度监测；
5. 置位 TSENS_SAMPLE_EN 自动触发持续温度监测。

自动温度监测工作模式下，温度传感器的输出值不会存储，但用户可以随时读取寄存器 TSENS_OUT 获取当前输出值。

64.8 寄存器列表

本小节的所有地址均为相对于温度传感器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

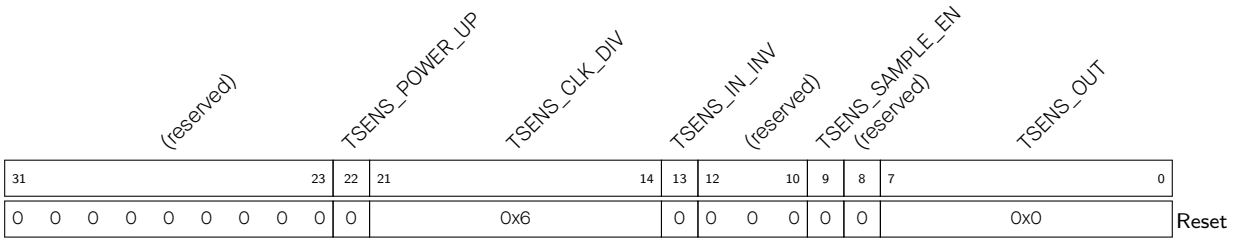
名称	描述	地址	访问
控制寄存器			
TSENS_CTRL_REG	温度传感器控制寄存器 1	0x0000	varies
TSENS_CTRL2_REG	温度传感器控制寄存器 2	0x0004	R/W
中断寄存器			
TSENS_INT_RAW_REG	温度传感器原始中断寄存器	0x0008	R/WTC/SS
TSENS_INT_ST_REG	温度传感器中断状态寄存器	0x000C	RO
TSENS_INT_ENA_REG	温度传感器中断使能寄存器	0x0010	R/WTC
TSENS_INT_CLR_REG	温度传感器中断清除寄存器	0x0014	WT
TSENS_INT_ENA_W1TS_REG	TSENS_INT_ENA_REG 的配置寄存器	0x001C	WT
TSENS_INT_ENA_W1TC_REG	TSENS_INT_ENA_REG 的配置寄存器	0x0020	WT
唤醒控制寄存器			
TSENS_WAKEUP_CTRL_REG	温度传感器唤醒控制寄存器	0x0024	varies
TSENS_SAMPLE_RATE_REG	温度传感器采样配置寄存器	0x0028	R/W

64.9 寄存器

本小节的所有地址均为相对于温度传感器基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 64.1. TSENS_CTRL_REG (0x0000)



TSENS_OUT 表示温度传感器的输出值。(RO)

TSENS_SAMPLE_EN 配置是否使能自动温度监测。

0：不使能

1：使能

(R/W)

TSENS_IN_INV 配置是否反转温度传感器的输入值。

0：无效

1：反转

(R/W)

TSENS_CLK_DIV 配置温度传感器的时钟分频系数，用于控制模拟电路产生温度值的频率。(R/W)

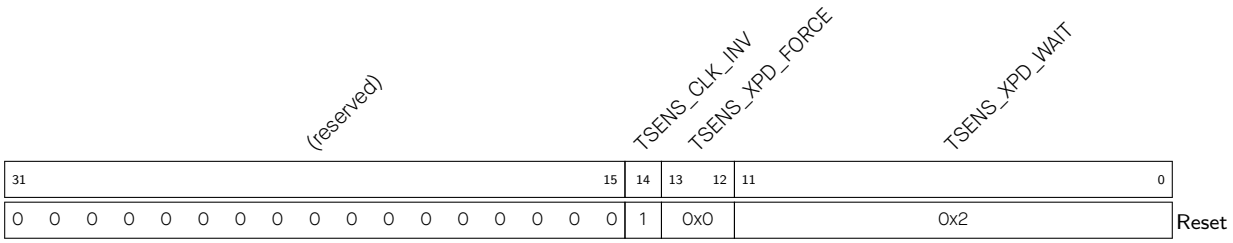
TSENS_POWER_UP 配置是否使能温度传感器上电。

0：无效

1：上电

(R/W)

Register 64.2. TSENS_CTRL2_REG (0x0004)



TSENS_XPD_WAIT 配置温度传感器的复位释放前需要等待的时钟周期。(R/W)

TSENS_XPD_FORCE 配置是否使能强制上电/断电温度传感器。

0：禁用强制上电功能

1：禁用强制断电功能

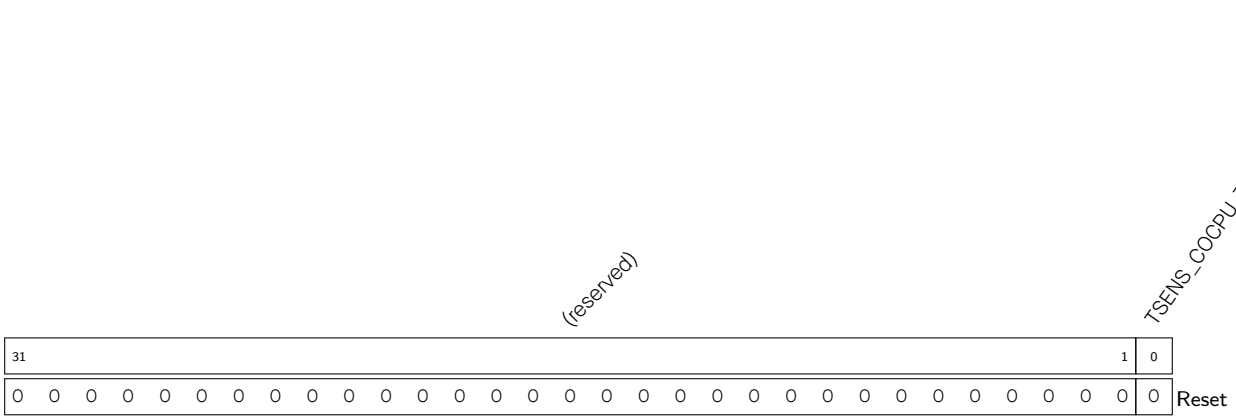
2：强制上电温度传感器

3：强制断电温度传感器

(R/W)

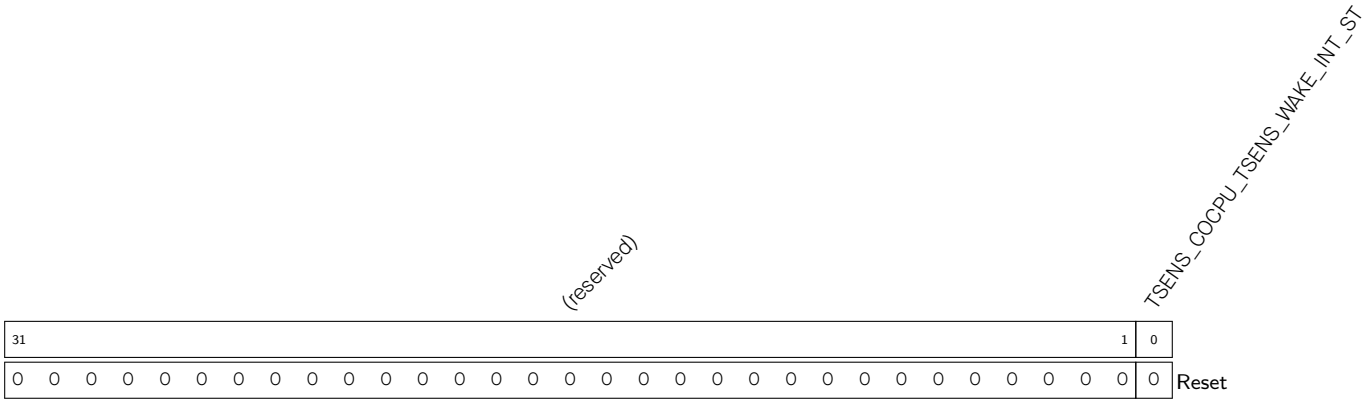
TSENS_CLK_INV 配置温度传感器采样时钟的相位。(R/W)

Register 64.3. TSENS_INT_RAW_REG (0x0008)



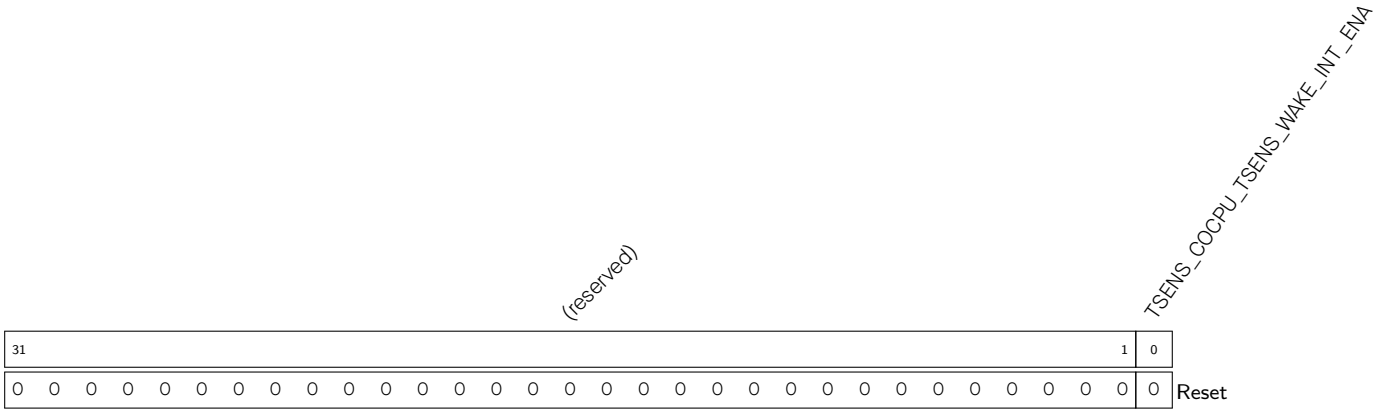
TSENS_COCPU_TSENS_WAKE_INT_RAW TSENS_COCPU_TSENS_WAKE_INT 的原始中断状态。(R/WTC/SS)

Register 64.4. TSENS_INT_ST_REG (0x000C)



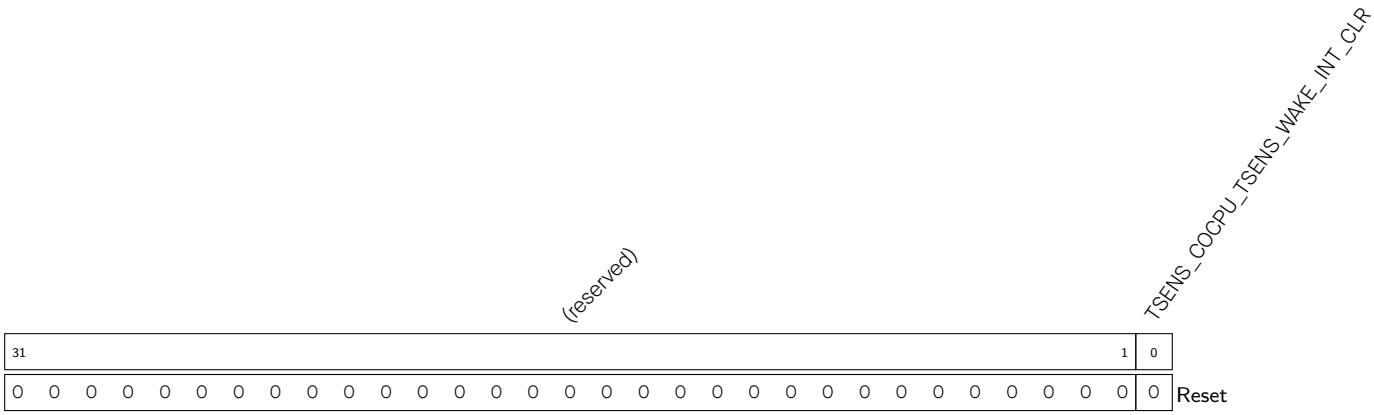
TSENS_COCPU_TSENS_WAKE_INT_ST TSENS_COCPU_TSENS_WAKE_INT 的屏蔽中断状态。(RO)

Register 64.5. TSENS_INT_ENA_REG (0x0010)



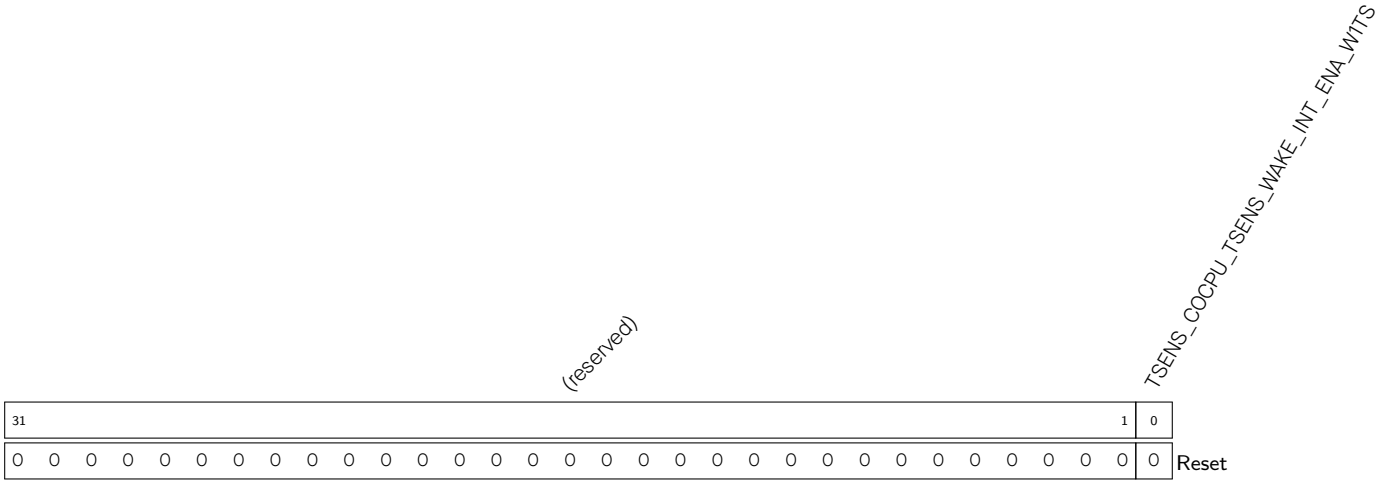
TSENS_COCPU_TSENS_WAKE_INT_ENA 写 1 使能 TSENS_COCPU_TSENS_WAKE_INT。(R/WTC)

Register 64.6. TSENS_INT_CLR_REG (0x0014)



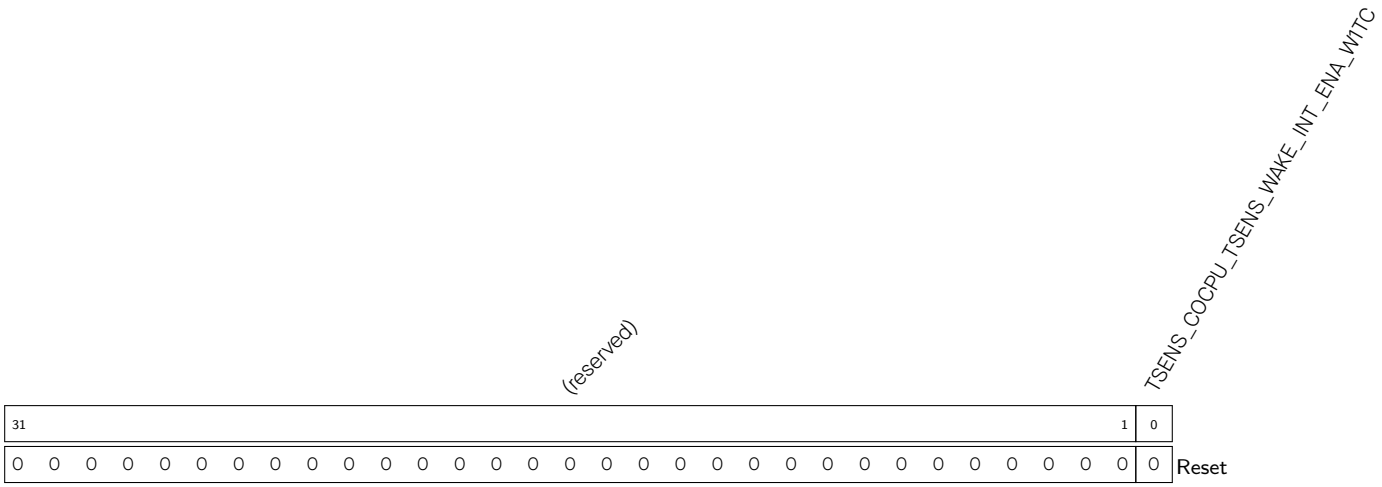
TSENS_COCPU_TSENS_WAKE_INT_CLR 写 1 清除 TSENS_COCPU_TSENS_WAKE_INT。(WT)

Register 64.7. TSENS_INT_ENA_W1TS_REG (0x001C)



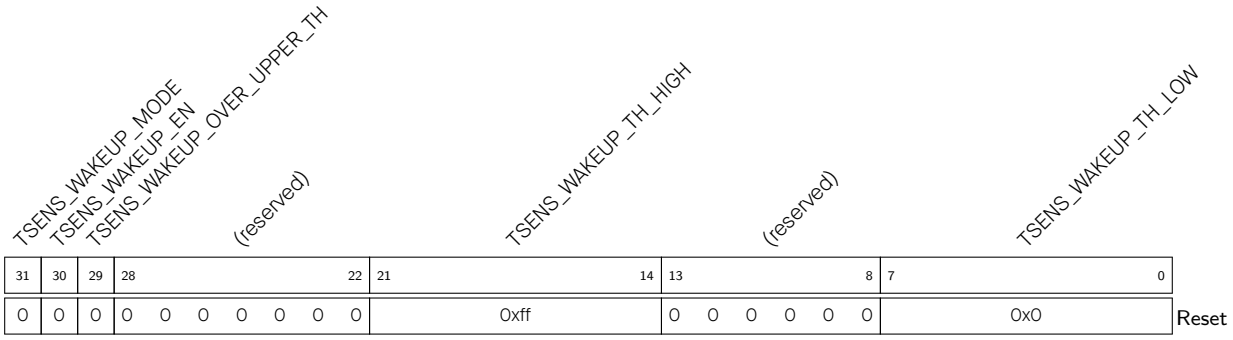
TSENS_COCPU_TSENS_WAKE_INT_ENA_W1TS 写 1 置位 TSENS_INT_ENA_REG。(WT)

Register 64.8. TSENS_INT_ENA_W1TC_REG (0x0020)



TSENS_COCPU_TSENS_WAKE_INT_ENA_W1TC 写 1 清除 TSENS_INT_ENA_REG。(WT)

Register 64.9. TSENS_WAKEUP_CTRL_REG (0x0024)



TSENS_WAKEUP_TH_LOW 配置温度传感器唤醒功能的低阈值。(R/W)

TSENS_WAKEUP_TH_HIGH 配置温度传感器唤醒功能的高阈值。(R/W)

TSENS_WAKEUP_OVER_UPPER_TH 表示输出值是否超过阈值。

0：输出值低于低阈值

1：输出值高于高阈值

(RO)

TSENS_WAKEUP_EN 配置是否启用唤醒功能。

0：不启用

1：启用

(R/W)

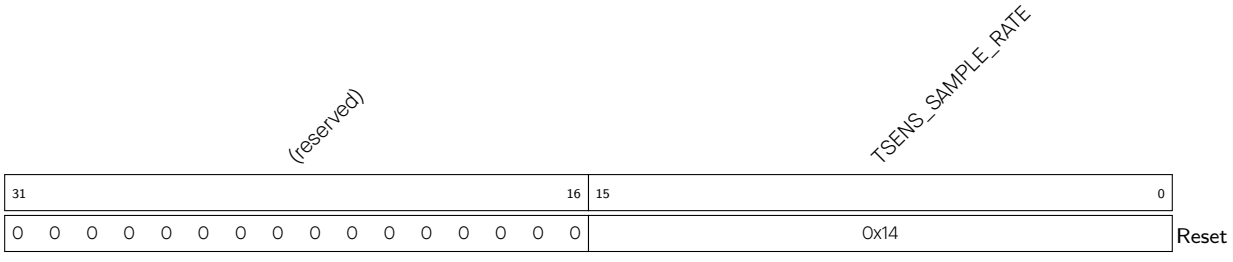
TSENS_WAKEUP_MODE 配置温度监测唤醒模式。仅当 TSENS_WAKEUP_EN=1 时有效。

0：绝对值模式

1：变化量模式

(R/W)

Register 64.10. TSENS_SAMPLE_RATE_REG (0x0028)



TSENS_SAMPLE_RATE 配置硬件自动温度监测的采样率。采样周期 = 配置值 × 传感器工作时钟周期。(R/W)

第 65 章

ADC 控制器 (ADC)

65.1 概述

ESP32-P4 内置了两个 12 位的逐次逼近型模拟数字转换器 (SAR ADC)，可测量最多来自 14 个管脚的模拟信号。图 65.1-1 所示为 SAR ADC 概图。

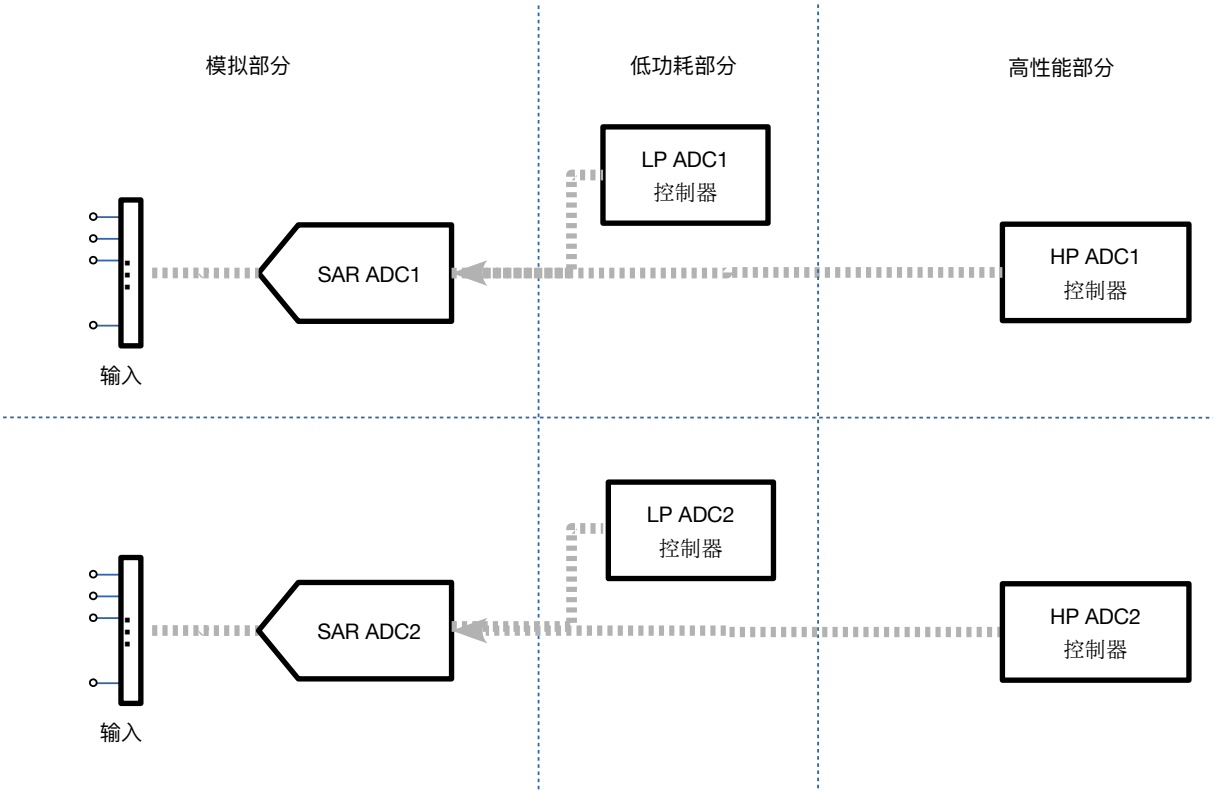


图 65.1-1. SAR ADC 结构概图

如图 65.1-1 所示，SAR ADC 由四个专用控制器控制，包括：

- 两个 HP（高性能）控制器，包括 HP ADC1 控制器和 HP ADC2 控制器，支持多通道采样模式，支持 GDMA 连续数据搬运。
- 两个 LP（低功耗）控制器，包括 LP ADC1 控制器和 LP ADC2 控制器，支持在低功耗模式下工作和单次采样模式。

65.2 术语表

下列有关 SAR ADC 的术语是在《ESP32-P4 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

SAR ADC： 包含模拟 ADC 电路与数字 ADC 控制器。下文的 HP ADC 与 LP ADC 共用一套模拟 ADC 电路，但拥有各自的控制器，即 HP ADC 控制器与 LP ADC 控制器。文中出现的 SAR ADC 既指 HP ADC 也指 LP ADC。

HP ADC： 高性能 ADC，支持多通道采样，GDMA 数据搬运等，睡眠模式下不可用。

LP ADC： 低功耗 ADC，支持单次采样模式，睡眠模式下可用。

单次采样模式： 在此模式下，LP ADC 一次对一个通道进行采样。

多通道采样模式： 在此模式下，HP ADC 依次对一组通道进行采样，或者连续对单个通道进行采样。

双 HP ADC 采样： 指的是两个 HP ADC 可以同时或轮流进行采样。

转换结果： 转换结果是模拟到数字转换后得到的二进制数字值。有时也被称为转换数据。

滤波数据： 滤波数据是转换数据经过滤波器处理得到的结果。

采样： 该术语通常指的是对模拟输入进行采样、转换采样数据，并将转换结果传输到内存的整个过程。在特定语境和流程中，“采样”也可能指 SAR ADC 从模拟信号中采集数据。

65.3 主要特性

SAR ADC 具有以下特性：

- 支持 HP ADC*x* 控制器和 LP ADC*x* 控制器通过软件选择的方式获取 SAR ADC*x* 的控制权，*x* = 1, 2
- 支持 12 位采样分辨率
- 支持采集最多 14 个管脚上的模拟信号
- HP ADC*x* 控制器：
  - 配有多通道采样控制模块，支持多通道采样模式，采样通道顺序可配
  - 提供模式控制模块，支持双 HP ADC 采样
  - 提供两个滤波器，滤波系数可配
  - 提供两个阈值监控器，滤波数据大于设置的高阈值或小于设置的低阈值将产生中断
  - GDMA 连续数据搬运
- LP ADC*x* 控制器：
  - 支持单次采样模式
  - 支持在低功耗模式（如 Deep-sleep）下工作
- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

65.4 结构概览

SAR ADC 的主要元件与连接情况见图 65.4-1。

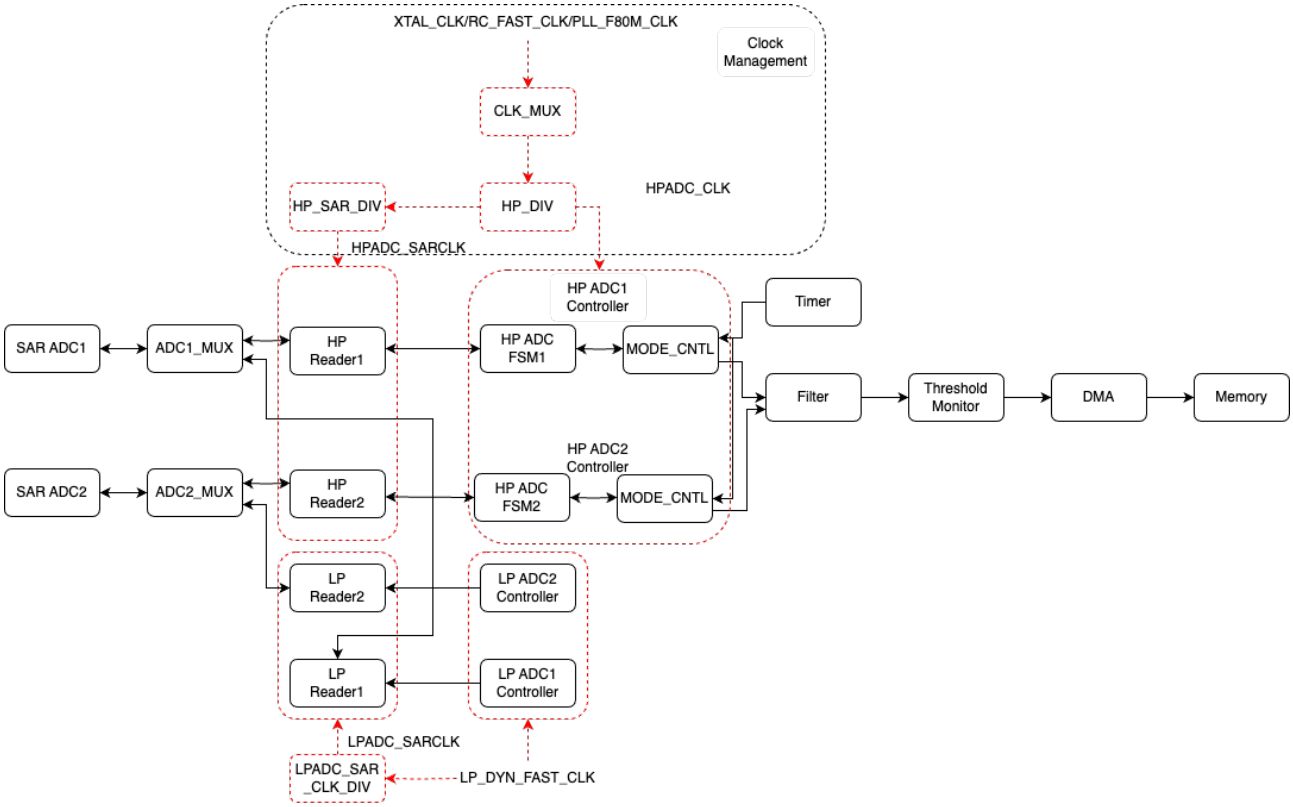


图 65.4-1. SAR ADC 结构图

- —>：时钟信号
- ：时钟分频器、多路器、以及时钟作用的模块

如图 65.4-1 所示，SAR ADC 模块主要包括以下部分：

- SAR ADC1：可对 8 个通道进行电压检测；
- SAR ADC2：可对 6 个通道进行电压检测；
- 时钟管理 (Clock Management)：对时钟源进行选择和分频；
- 定时器 (Timer)：HP ADC*x* 控制器的专用定时器，可发起采样使能信号。
- HP ADC FSM*x*：有限状态机 1 和有限状态机 2，用于
  - 接收定时器提供的采样使能信号
  - 根据样式表生成 HP ADC*x* 的采样配置
  - 驱动 HP Reader*x* 模块读取转换数据
  - 将转换数据传输到滤波器
- 滤波器 (Filter)：滤波器 0 和滤波器 1，用于在多通道采样模式下对目标通道的转换数据进行滤波
- 阈值监控器 (Threshold Monitor)：阈值监控器 0 和阈值监控器 1，可在滤波数据大于设定的高阈值，或小于设定的低阈值时触发中断。

- 模式控制模块 (MODE_CNTL)：用于过滤定时器触发的采样信号，支持双 HP ADC 采样。
- HP Reader*x*：HP 读值器 1 和 HP 读值器 2，由 HP ADC FSM*x* 驱动，读取 HP ADC*x* 的数值。
- LP Reader*x*：LP 读值器 1 和 LP 读值器 2，由 LP ADC*x* 控制器驱动，读取 LP ADC*x* 的数值。
- LP ADC*x* Controller：LP ADC1 控制器和 LP ADC2 控制器，提供采样使能信号，根据配置驱动 LP Reader*x* 模块读取 LP ADC*x* 的转换数据，并存储转换数据。

65.5 功能描述

65.5.1 SAR ADC 上电

置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 寄存器给 SAR ADC 上电。上电后即可进行采样，硬件设计上已考虑等待时间，因此用户无需特意等待。

65.5.2 SAR ADC 通道

SAR ADC 共有 14 个通道，连接芯片的 14 个管脚。SAR ADC 需首先通过内部多路器选择待测量的模拟管脚，然后才能采样模拟信号。

表 65.5-1 列出了用作 SAR ADC 通道的芯片管脚。

表 65.5-1. SAR ADC 通道及芯片管脚

芯片管脚	SAR ADC 通道编号	SAR ADC 选择
GPIO16	0	SAR ADC1
GPIO17	1	
GPIO18	2	
GPIO19	3	
GPIO20	4	
GPIO21	5	
GPIO22	6	
GPIO23	7	
GPIO49	0	SAR ADC2
GPIO50	1	
GPIO51	2	
GPIO52	3	
GPIO53	4	
GPIO54	5	

65.5.3 SAR ADC 时钟

用户可配置 HP_SYS_CLKRST_ADC_CLK_SRC_SEL 选择 HP ADC*x* 控制器的时钟源：

- XTAL_CLK
- RC_FAST_CLK
- PLL_F80M_CLK

HP ADC*x* 控制器的分频时钟有：

- HPADC_SARCLK：HP ADC*x*、HP Reader*x* 的工作时钟。
- HPADC_CLK：HP ADC FSM*x* 的工作时钟。

注意，HPADC_SARCLK 频率会影响采样精度。当 HPADC_SARCLK 的频率高于 5 MHz 时，采样精度会变低。HPADC_SARCLK 由 HPADC_CLK 经过专用分频器分频所得，分频系数通过 ADC_SAR_CLK_DIV 配置。

HP ADC 每采样一个数据需要 25 个 HPADC_SARCLK 时钟周期数，所以最大采样速率受到 HPADC_SARCLK 的频率限制。

LP ADC*x* 控制器的时钟源为：LP_DYN_FAST_CLK。

LP ADC*x* 控制器的分频时钟有：

- LPADC_SARCLK：LP ADC*x*、LP Reader*x* 的工作时钟；其中控制 LPADC_SARCLK 分频的 LPADC_SAR1_CLK_DIV 和 LPADC_SAR2_CLK_DIV 的分频系数至少是 2 分频，且 LPADC_SARCLK 的频率不能超过 5 MHz。
- LPADC_CLK：LP ADC FSM*x* 的工作时钟。

更多时钟信息，见章节 11 复位和时钟。

65.5.4 SAR ADC 转换和衰减

SAR ADC 转换模拟信号时，转换电压范围为 0 mV ~ V_{ref}。V_{ref} 为 SAR ADC 内部参考电压。SAR ADC 的转换结果是个 12 位的数字值，是原始值。用户可使用以下公式将原始值 $data$ 转换成模拟电压 V_{data}：

$$V_{data} = \frac{V_{ref}}{k} \times \frac{data}{4095}$$

其中 k 为衰减对应的系数。

如需转换大于 V_{ref} 的电压，信号输入 SAR ADC 前可进行衰减。支持的衰减等级包括：

- 0 dB (k≈100%)
- 2.5 dB (k≈75%)
- 6 dB (k≈50%)
- 12 dB (k≈25%)

65.5.5 HP ADC FSM

HP ADC FSM（下文简称 FSM）在多通道采样模式下工作，用于生成整个采样过程中所需的各种信号。图 65.5-1 展示了 FSM 的工作原理。

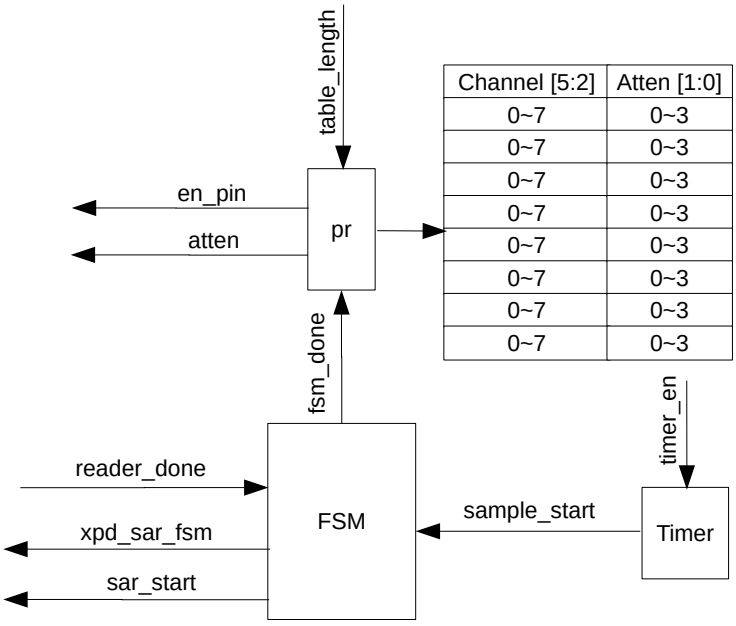


图 65.5-1. HP ADC FSM 概况

其中，

- Timer（定时器）：表示 HP ADC 控制器的专用定时器，用于触发多通道采样模式。定时器可产生 `sample_start` 信号，表示开始采样。
- pr：样式表指针，FSM 将根据该指针指向的样式配置，发送相应信号。

用户通过置位 ADC_TIMER_EN 使能定时器。定时器超时将触发 `sample_start` 信号驱动 FSM 模块开始采样。FSM 模块收到 `sample_start` 信号后，执行以下操作：

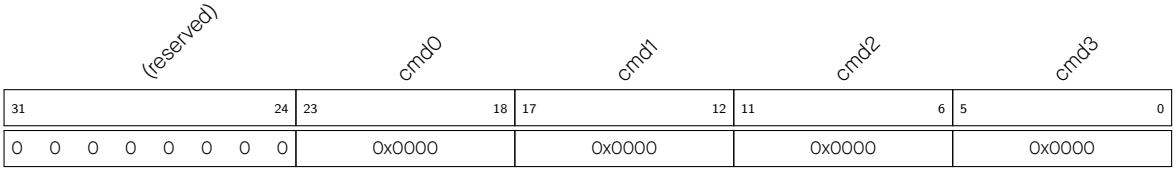
- 开启 HP ADC 电源；
- 根据当前 pr 指向的样式，决定该次采样通道以及衰减；
- 根据配置信息，输出相应的 `en_pin`（使能管脚）以及 `atten`（衰减）信号到模拟端；
- 发起 `sar_start` 信号，开启采样。

FSM 收到 HP Reader 返回的 `reader_done` 信号后，执行以下操作：

- 结束采样；
- 将数据传输给滤波器，然后阈值监控器通过 GDMA 将数据传输给内存（见图 65.4-1）；
- 更新样式表指针 pr，等待下一次采样。pr 在 0 - ADC_SAR1/2_PATT_LEN (table_length) 之间循环计数。

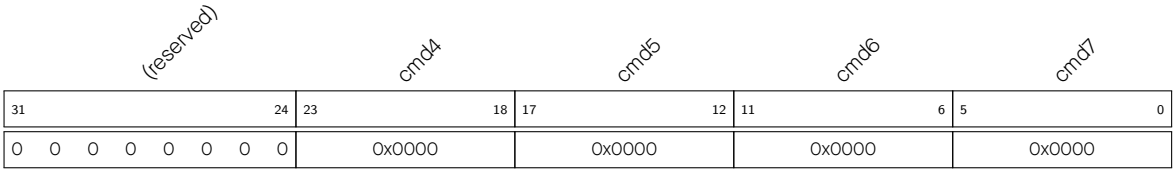
65.5.6 HP ADC 样式表

HP ADC FSM1/2 各自包含一个独立样式表，由 ADC_SAR*x*_PATT_TAB*m*_REG 配置，其中 *m* 表示样式表的寄存器序号 1 ~ 4。每个寄存器包含四个样式，每个样式长度为六位，如下图所示：



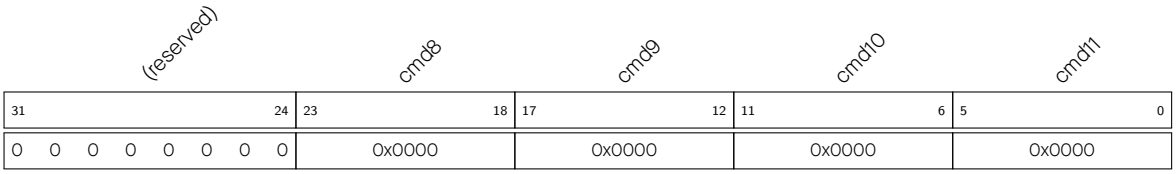
cmd *n* (*n* = 0 - 3) 表示样式表中的样式，即样式 0 ~ 样式 3。

图 65.5-2. ADC_SAR*x*_PATT_TAB1_REG 包含样式 0 - 3



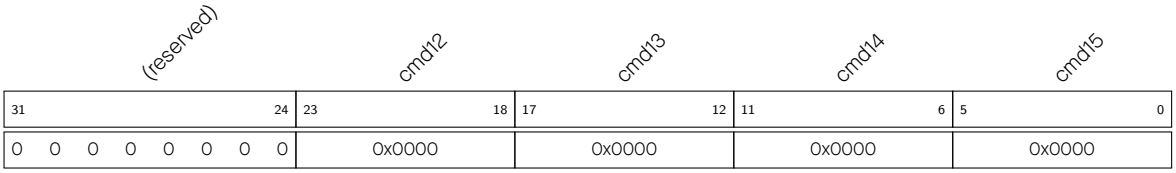
cmd *n* (*n* = 4 - 7) 表示样式表中的样式，即样式 4 ~ 样式 7。

图 65.5-3. ADC_SAR*x*_PATT_TAB2_REG 包含样式 4 - 7



cmd *n* (*n* = 8 - 11) 表示样式表中的样式，即样式 8 ~ 样式 11。

图 65.5-4. ADC_SAR*x*_PATT_TAB3_REG 包含样式 8 - 11



cmd *n* (*n* = 12 - 15) 表示样式表中的样式，即样式 12 ~ 样式 15。

图 65.5-5. ADC_SAR*x*_PATT_TAB4_REG 包含样式 12 - 15

每个样式长度为六位，共包括 2 个字段，分别存储通道和衰减信息，具体见图 65.5-6。

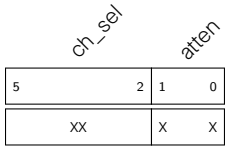


图 65.5-6. 样式结构

atten 衰减配置信息：

0: 0 dB

1: 2.5 dB

2: 6 dB

3: 12 dB

ch_sel 采样通道选择信息，对于 HP ADC1，0-7 分别对应 0-7 通道。对于 HP ADC2，0-5 分别对应 0-5 通道。

65.5.7 HP ADC 多通道采样配置示例

假设，希望实现如下顺序的 HP ADC1 多通道采样方式：

- 采样 HP ADC1 的通道 2，且衰减配置为 12 dB；
- 采样 HP ADC1 的通道 0，且衰减配置为 2.5 dB。

则具体的配置如下：

- 配置 HP ADC1 第一个样式 cmd0（即 ADC_SAR1_PATT_TAB1_REG[5:0]），如下图所示：

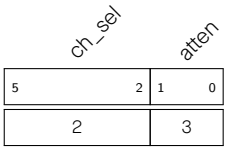


图 65.5-7. HP ADC1 cmd0 配置示例

atten 配置该字段的值为 3，即衰减配置为 12 dB。

ch_sel 配置该字段的值为 2，即选择通道 2。

- 配置 HP ADC1 第二个样式 cmd1（即 ADC_SAR1_PATT_TAB1_REG[11:6]），如下图所示：

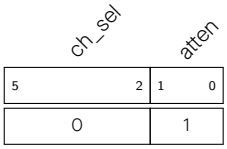


图 65.5-8. HP ADC1 cmd1 配置示例

atten 配置该字段的值为 1，即衰减配置为 2.5 dB。

ch_sel 配置该字段的值为 0，即选择通道 0（见表 65.5-1）。

- 配置 ADC_SAR1_PATT_LEN 为 1，即样式表大小为 (配置值 +1)= 2，也即选择使用上述配置好的样式 0 和样式 1；
- 使能定时器，则 HP ADC1 控制器将根据上述样式配置，周期性采样 HP ADC1 的通道 2 和通道 0。

65.5.8 双 HP ADC 采样控制

ADC_WORK_MODE 控制是否让两个 HP ADC 同时工作：

- 0：单 HP ADC 采样，即只有一个 HP ADC 采样。当定时器触发采样请求时，由 ADC_SAR_SEL 控制某个 FSM 收到采样请求：
  - 0：HP ADC FSM1 收到采样请求，即使用 HP ADC1 采样。
  - 1：HP ADC FSM2 收到采样请求，即使用 HP ADC2 采样。
- 1：双 HP ADC 同时采样。当定时器触发采样请求时，HP ADC FSM1 和 HP ADC FSM2 将同时收到采样请求。
- 2：双 HP ADC 轮流采样。当有奇数次定时器触发的采样请求时，HP ADC FSM1 收到采样请求；当有偶数次采样请求时，HP ADC FSM2 收到采样请求。

65.5.9 HP ADC 滤波器

HP ADC*x* 控制器提供两个滤波器，用于在多通道采样模式下对目标通道的转换数据进行滤波。两个滤波器均可配置给 HP ADC 的任一通道，但不能配置给同一通道。如果配置给同一通道，则第一次配置的滤波器生效。

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} + 0.5$$

- $data_{cur}$：滤波后数据
- $data_{in}$：转换数据
- $data_{prev}$：上次滤波数据
- k：滤波系数

配置滤波器如下：

- 配置 ADC_FILTER_CHANNEL*x* 设置滤波器 *x* 作用的 HP ADC 通道，*x*=0, 1。
- 配置 ADC_FILTER_FACTOR*x* 设置滤波器 *x* 的滤波系数，*x*=0, 1。

65.5.10 HP ADC 阈值监控器

HP ADC 控制器包含两个阈值监控器，用于在多通道采样模式下对滤波之后的数据进行监控。当数据大于设定的高阈值，则触发高阈值中断；当数据小于设定的低阈值，则触发低阈值中断。两个监控器可配置给 HP ADC 的任一通道，但不能配置给同一通道。

阈值监控配置如下：

- 置位 ADC_THRES*x*_EN 使能阈值监控 *x* (*x*=0, 1)；
- 配置 ADC_THRES*x*_LOW 设置低阈值；
- 配置 ADC_THRES*x*_HIGH 设置高阈值；
- 配置 ADC_THRES*x*_CHANNEL 设置监控的 HP ADC 以及通道。

65.5.11 HP ADC 的 GDMA 支持

当转换多个通道时需要使用 GDMA，通过 GDMA 接口将转换数据连续存储至内存。

由 HP ADC*x* 专用定时器产生触发信号，用户可通过软件配置 ADC_APB_ADC_TRANS 将 GDMA 的数据通路切换到 HP ADC。关于 GDMA 的具体配置，请参考章节 6 *通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)*。

HP ADC 最终向 GDMA 传递 32 位数据，如下图所示：

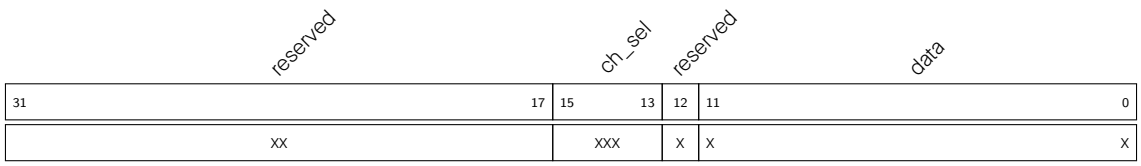


图 65.5-9. GDMA 数据格式

data HP ADC 转换结果，12 位

ch_sel 通道信息，3 位

65.5.12 LP ADC 自动监测唤醒模式

LP ADC 控制器支持自动监测转换数据，一个专用的定时器会每根据 LPADC_ADC*x*_HW_READ_RATE_I 配置值周期性产生 LP ADC 采样信号，并将得到的转换数据送入一个阈值监控器（未复用 HP ADC 的阈值监控器）中，若转换数据超过阈值，将产生中断。

自动监测有两种唤醒模式：绝对值模式和变化量模式，通过寄存器 LPADC_SAR*x*_WAKEUP_MODE 进行选择：

- 0：绝对值模式—
  用于监测当前转换数据的绝对值。配置 LPADC_SAR*x*_WAKEUP_TH_LOW 和 LPADC_SAR*x*_WAKEUP_TH_HIGH 来决定转换数据阈值，转换数据超过高阈值或小于低阈值时将触发唤醒。
- 1：变化量模式—
  用于监测转换数据的变化量。若连续两次采样的转换数据增量超过 LPADC_SAR*x*_WAKEUP_TH_HIGH 的配置值或减少量超过 LPADC_SAR*x*_WAKEUP_TH_LOW 的配置值，将触发唤醒。例如当 LPADC_SAR*x*_WAKEUP_TH_LOW 的值为 8 时，若连续两次采样后的转换数据分别为 28 和 19，即转换数据减少量为 9，将触发唤醒。

65.6 事件任务矩阵功能

在 ESP32-P4 中，SAR ADC 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 SAR ADC 的 ETM 任务，或者通过 SAR ADC 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与 SAR ADC 相关的 ETM 任务和 ETM 事件。

SAR ADC 可接收的 ETM 任务有：

- ADC_TASK_SAMPLE0：该任务被触发时，LP ADC 开始进行单次采样。
- ADC_TASK_START0：该任务被触发时，HP ADC 开始进行多通道采样。
- ADC_TASK_STOP0：该任务被触发时，SAR ADC 停止采样。

SAR ADC 可产生的 ETM 事件有：

- ADC_EVT_CONV_CMPLT0：在单次采样模式和多通道采样模式下，SAR ADC 每完成一次采样时生成。
- ADC_EVT_EQ_ABOVE_THRESH*x*：HP ADC 过滤后数据高于阈值时生成。*x*=0, 1，代表阈值监控器 0, 1。
- ADC_EVT_EQ_BELOW_THRESH*x*: HP ADC 过滤后数据低于阈值时生成。*x*=0, 1，代表阈值监控器 0, 1。
- ADC_EVT_STARTED0：SAR ADC 开始采样时生成，单次采样不会触发该事件。
- ADC_EVT_STOPPED0：SAR ADC 停止采样时生成，单次采样不会触发该事件。

在具体应用中，SAR ADC 的 ETM 事件可以用来触发 SAR ADC 的 ETM 任务，例如，ADC_EVT_EQ_ABOVE_THRESH*x* 事件可以触发 ADC_TASK_STOP0 任务。

65.7 中断

ESP32-P4 的 SAR ADC 可以生成以下中断信号并发送给中断矩阵。

- ADC_INTR

- LP_ADC_INTR

ADC_INTR 由以下 HP ADC 的内部中断源生成。

- ADC_SAR*x*_DONE_INT：当 HP ADC*x* 完成一次采样后触发。
- ADC_THRES*x*_HIGH_INT：当滤波后的数据超过高阈值时触发。
- ADC_THRES*x*_LOW_INT：当滤波后的数据低于低阈值时触发。

LP_ADC_INTR 由以下 LP ADC 的内部中断源生成。

- LPADC_COCPU_SARADC*x*_DONE_INT：当 LP ADC*x* 完成一次采样时触发。
- LPADC_COCPU_SARADC*x*_ERROR_INT：当 LP ADC*x* 发生采样错误时触发。
- LPADC_COCPU_SARADC*x*_WAKE_INT：当 LP ADC*x* 产生一个唤醒事件（如转换数据超阈值）时触发。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 65.9 寄存器列表。

65.8 配置流程

65.8.1 HP ADC 多通道采样配置

专用定时器驱动的多通道采样的具体配置如下：

1. 置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 给 HP ADC 上电；
2. 配置 HP_SYS_CLKRST_ADC_CLK_SRC_SEL 选择时钟源；
3. 配置时钟分频寄存器 HP_SYS_CLKRST_ADC_CLK_DIV_NUM 和 HP_SYS_CLKRST_ADC_SAR*x*_CLK_DIV_NUM；
4. 置位 HP_SYS_CLKRST_ADC_CLK_EN 使能 HP ADC 时钟；
5. 置位 LPADC_SAR*x*_DIG_FORCE 打开 HP ADC 采样通道；
6. 按照章节 65.5.6 *HP ADC 样式表* 所述步骤配置样式表；
7. 按照章节 65.5.9 *HP ADC 滤波器* 所述步骤配置滤波器作用的通道和滤波系数；
8. 根据需要，按照章节 65.5.10 *HP ADC 阈值监控器* 所述步骤设置阈值监控；
9. 置位 ADC_APB_ADC_TRANS 选择使用 GDMA；
10. 配置 ADC_TIMER_TARGET 设置 HP ADC 定时器的触发周期；
11. 置位 ADC_TIMER_EN 使能定时器。

定时器超时则将驱动 HP ADC FSM 根据样式表进行采样。数据通过 GDMA 自动存储到内存空间中。当采集到 ADC_APB_ADC_EOF_NUM 寄存器设置的次数时，采样停止。

采样完成将产生 ADC_SAR*x*_DONE_INT_RAW 中断。

65.8.2 LP ADC 单次采样配置

软件驱动的单次采样具体配置过程如下：

1. 置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RSTB 给 LP ADC 上电；

2. 配置时钟分频寄存器 LPPERI_LPADC_FUNC_DIV_NUM 和 LPPERI_LPADC_SAR*x*_DIV_NUM，分别决定 LPADC_CLK 与 LPADC*x*_SARCLK 的频率；
3. 配置 LPADC_SAR*x*_EN_PAD 和 LPADC_SAR*x*_ATTEN 选择本次采样的通道及衰减值；
4. 配置 LPADC_SAR_MEAS*x*_START_SAR 触发 LP ADC*x* 控制器采样。

采样结束即触发 LPADC_COCPU_SARADC*x*_INT_RAW。软件检测到该中断后，可从 LPADC_SAR_MEAS*n*_DATA_SAR 读取转换数据。

65.8.3 LP ADC 自动监测唤醒配置

LP ADC 控制器的自动监测唤醒模式配置过程如下：

1. 配置 LPADC_ADC*x*_HW_READ_RATE_I 设置采样率；
2. 配置 LPADC_SAR*x*_WAKEUP_MODE 选择唤醒模式；
3. 配置 LPADC_SAR*x*_WAKEUP_TH_HIGH 与 LPADC_SAR*x*_WAKEUP_TH_LOW 设置高低阈值；
4. 置位 LPADC_SAR*x*_WAKEUP_EN 启用自动监测；
5. 置位 LPADC_ADC*x*_HW_READ_EN_I 自动触发持续监测。

自动监测工作模式下，转换数据不会存储，但用户可以随时读取寄存器 LPADC_SAR_MEAS*n*_DATA_SAR 获取最后一次转换数据。

65.9 寄存器列表

65.9.1 HP ADC 寄存器列表

下表的所有地址均为相对于 HP ADC（写作 ADC 控制器）基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
HP ADC 配置寄存器			
ADC_CTRL_REG	HP ADC 控制器配置寄存器	0x0000	R/W
ADC_CTRL2_REG	HP ADC 控制器配置寄存器	0x0004	R/W
ADC_FILTER_CTRL1_REG	HP ADC 滤波器配置寄存器 1	0x0008	R/W
ADC_SAR1_PATT_TAB1_REG	HP ADC1 样式表寄存器 1	0x0018	R/W
ADC_SAR1_PATT_TAB2_REG	HP ADC1 样式表寄存器 2	0x001C	R/W
ADC_SAR1_PATT_TAB3_REG	HP ADC1 样式表寄存器 3	0x0020	R/W
ADC_SAR1_PATT_TAB4_REG	HP ADC1 样式表寄存器 4	0x0024	R/W
ADC_SAR2_PATT_TAB1_REG	HP ADC2 样式表寄存器 1	0x0028	R/W
ADC_SAR2_PATT_TAB2_REG	HP ADC2 样式表寄存器 2	0x002C	R/W
ADC_SAR2_PATT_TAB3_REG	HP ADC2 样式表寄存器 3	0x0030	R/W
ADC_SAR2_PATT_TAB4_REG	HP ADC2 样式表寄存器 4	0x0034	R/W
ADC_FILTER_CTRL0_REG	HP ADC 滤波器配置寄存器 0	0x003C	R/W
ADC_THRES0_CTRL_REG	滤波数据阈值控制寄存器 0	0x0044	R/W
ADC_THRES1_CTRL_REG	滤波数据阈值控制寄存器 1	0x0048	R/W
ADC_THRES_CTRL_REG	滤波器配置寄存器	0x004C	R/W
ADC_INT_ENA_REG	HP ADC 中断使能寄存器	0x0050	R/W
ADC_INT_RAW_REG	HP ADC 原始中断寄存器	0x0054	R/WTC/SS
ADC_INT_ST_REG	HP ADC 中断状态寄存器	0x0058	RO
ADC_INT_CLR_REG	HP ADC 中断清除寄存器	0x005C	WT
ADC_DMA_CONF_REG	HP ADC GDMA 配置寄存器	0x0060	R/W
HP ADC 版本控制寄存器			
ADC_CTRL_DATE_REG	HP ADC 版本控制寄存器	0x03FC	RO

65.9.2 LP ADC 寄存器列表

下表的所有地址均为相对于 LP ADC 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
LP ADC 配置寄存器			
LPADC_READER1_CTRL_REG	LP ADC1 Reader 控制寄存器	0x0000	R/W
LPADC_MEAS1_CTRL2_REG	LP ADC1 配置寄存器	0x000C	varies

名称	描述	地址	访问
LPADC_MEAS1_MUX_REG	为 SAR ADC1 选择 LP 或 HP ADC 控制器	0x0010	R/W
LPADC_ATTEN1_REG	LP ADC1 衰减配置寄存器	0x0014	R/W
LPADC_READER2_CTRL_REG	LP ADC2 Reader 控制寄存器	0x0024	R/W
LPADC_MEAS2_CTRL2_REG	LP ADC2 配置寄存器	0x0030	varies
LPADC_MEAS2_MUX_REG	为 SAR ADC2 选择 LP 或 HP ADC 控制器	0x0034	R/W
LPADC_ATTEN2_REG	LP ADC2 衰减配置寄存器	0x0038	R/W
LP ADC 上电控制寄存器			
LPADC_FORCE_WPD_SAR_REG	LP ADC 上电控制寄存器	0x003C	R/W
LP ADC 中断寄存器			
LPADC_COCPU_INT_RAW_REG	LP ADC 原始中断寄存器	0x0048	R/WTC/SS
LPADC_INT_ENA_REG	LP ADC 中断使能寄存器	0x004C	R/WTC
LPADC_INT_ST_REG	LP ADC 中断状态寄存器	0x0050	RO
LPADC_INT_CLR_REG	LP ADC 中断清除寄存器	0x0054	WT
LPADC_INT_ENA_W1TS_REG	LPADC_INT_ENA_REG 配置寄存器	0x0058	WT
LPADC_INT_ENA_W1TC_REG	LPADC_INT_ENA_REG 配置寄存器	0x005C	WT
LP ADC 唤醒控制寄存器			
LPADC_WAKEUP1_REG	LP ADC1 唤醒配置寄存器	0x0060	varies
LPADC_WAKEUP2_REG	LP ADC2 唤醒配置寄存器	0x0064	varies
LPADC_WAKEUP_SEL_REG	唤醒源选择寄存器	0x0068	R/W
LPADC_SAR1_HW_WAKEUP_REG	LP ADC1 自动监测配置寄存器	0x006C	R/W
LPADC_SAR2_HW_WAKEUP_REG	LP ADC2 自动监测配置寄存器	0x0070	R/W

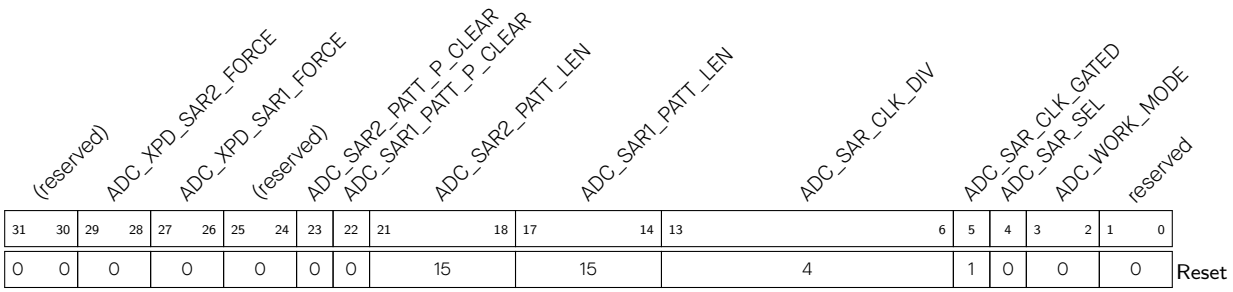
65.10 寄存器

65.10.1 HP ADC 寄存器

以下所有地址均为相对于 HP ADC（写作 ADC 控制器）基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 65.1. ADC_CTRL_REG (0x0000)



ADC_WORK_MODE 配置双 HP ADC 采样。

0：单 HP ADC 采样

1：双 HP ADC 同时采样

2：双 HP ADC 轮流采样

(R/W)

ADC_SAR_SEL 配置使用哪一个 HP ADC，仅用于单 HP ADC 采样。

0：使用 HP ADC1

1：使用 HP ADC2

(R/W)

ADC_SAR_CLK_GATED 当 HP ADC 处于空闲状态时配置是否使能时钟门控。

0：不使能

1：使能

(R/W)

ADC_SAR_CLK_DIV 配置 HP ADC 的时钟分频。分频系数为配置值 +1。(R/W)

ADC_SAR1_PATT_LEN 配置 HP ADC1 需要使用的样式数量。

0：仅使用 cmd0

1：使用 cmd0 和 cmd1

n：依次使用 cmd0 至 cmdn, n 最大取 7

(R/W)

ADC_SAR2_PATT_LEN 配置 HP ADC2 需要使用的样式数量。

0：仅使用 cmd0

1：使用 cmd0 和 cmd1

n：依次使用 cmd0 至 cmdn, n 最大取 7

(R/W)

见下页...

Register 65.1. ADC_CTRL_REG (0x0000)

接上页...

ADC_SAR1_PATT_P_CLEAR 配置是否清除 HP ADC1 控制器样式表指针。

0：无效

1：清除

(R/W)

ADC_SAR2_PATT_P_CLEAR 配置是否清除 HP ADC2 控制器样式表指针。

0：无效

1：清除

(R/W)

ADC_XPD_SAR1_FORCE 配置是否强制上电 HP ADC1。

0：无效

1：强制上电

(R/W)

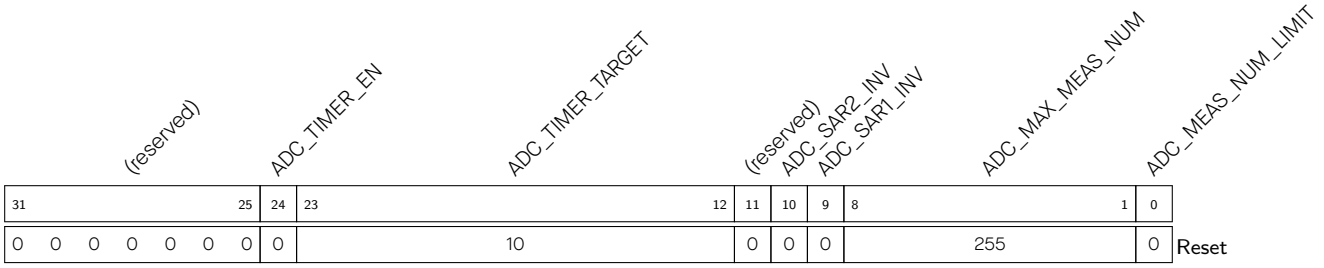
ADC_XPD_SAR2_FORCE 配置是否强制上电 HP ADC2。

0：无效

1：强制上电

(R/W)

Register 65.2. ADC_CTRL2_REG (0x0004)



ADC_MEAS_NUM_LIMIT 配置是否使能 HP ADC 最大转换次数限制。

0：不使能

1：使能

(R/W)

ADC_MAX_MEAS_NUM 配置 HP ADC 的最大转换次数。(R/W)

ADC_SAR1_INV 配置是否反转 HP ADC1 数据。

0：无效

1：反转

(R/W)

ADC_SAR2_INV 配置是否反转 HP ADC2 数据。

0：无效

1：反转

(R/W)

ADC_TIMER_TARGET 设置 HP ADC 定时器目标，即定时器的触发周期。(R/W)

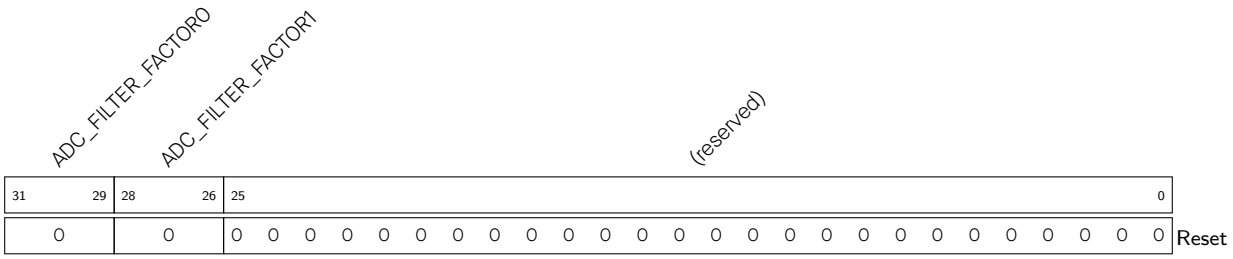
ADC_TIMER_EN 配置是否使能 HP ADC 定时器触发。

0：不使能

1：使能

(R/W)

Register 65.3. ADC_FILTER_CTRL1_REG (0x0008)



ADC_FILTER_FACTOR1 设置 HP ADC 滤波器 1 的滤波系数 k。

0: k=0（即滤波器关闭）

1: k=2

2: k=4

3: k=8

4: k=16

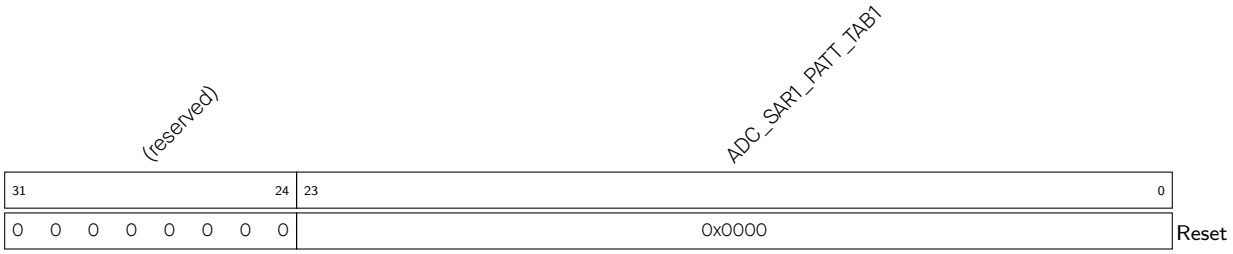
5: k=32

6: k=64

(R/W)

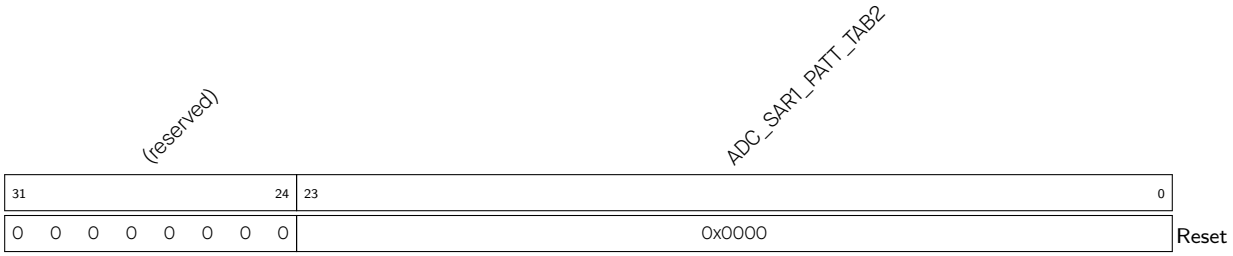
ADC_FILTER_FACTOR0 设置 HP ADC 滤波器 0 的滤波系数 k。配置同上。(R/W)

Register 65.4. ADC_SAR1_PATT_TAB1_REG (0x0018)



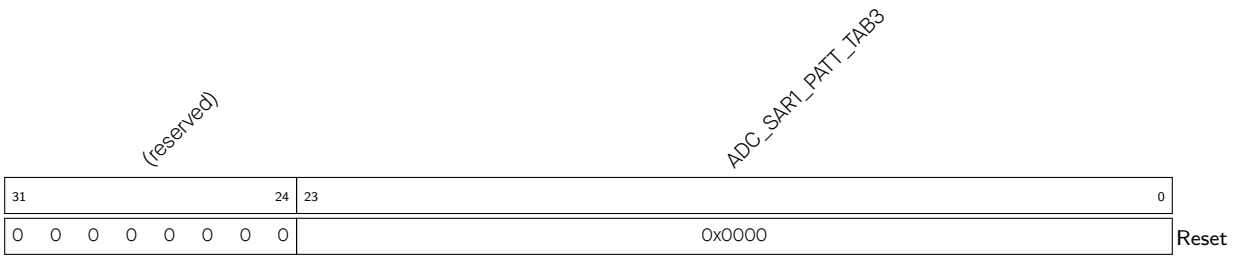
ADC_SAR1_PATT_TAB1 配置 HP ADC FSM1 样式表的第 0 ~ 3 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.5. ADC_SAR1_PATT_TAB2_REG (0x001C)



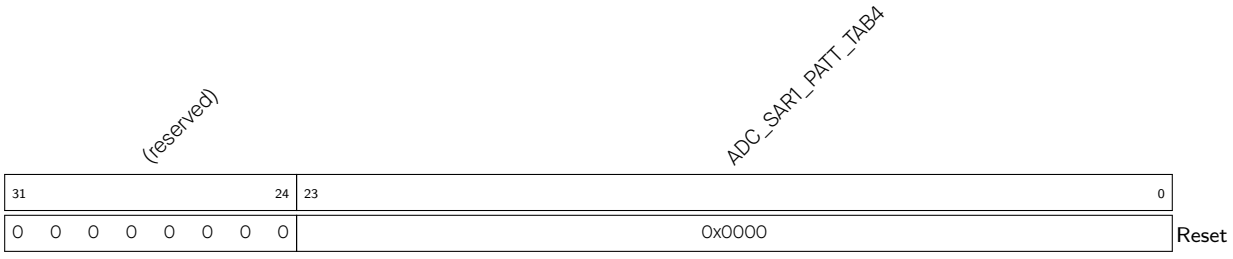
ADC_SAR1_PATT_TAB2 配置 HP ADC FSM1 样式表的第 4 ~ 7 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.6. ADC_SAR1_PATT_TAB3_REG (0x0020)



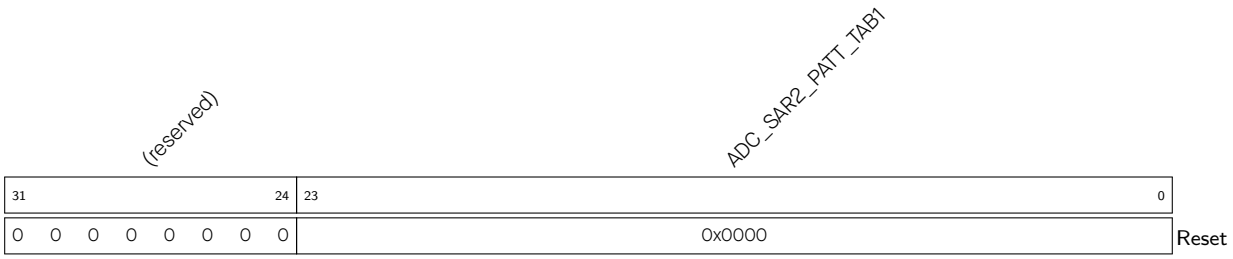
ADC_SAR1_PATT_TAB3 配置 HP ADC FSM1 样式表的第 8 ~ 11 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.7. ADC_SAR1_PATT_TAB4_REG (0x0024)



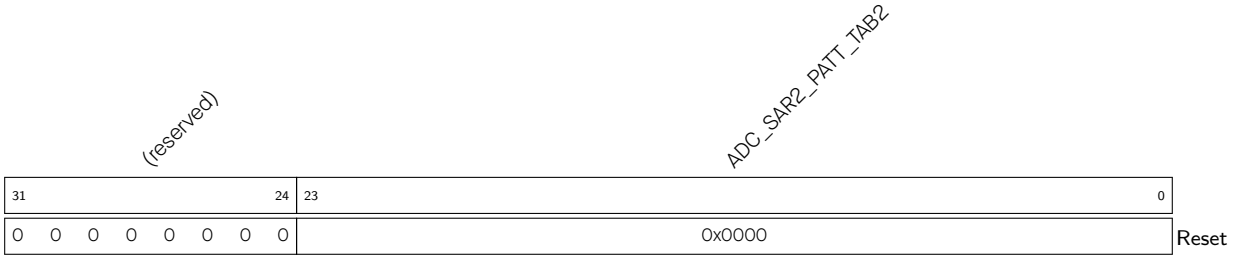
ADC_SAR1_PATT_TAB4 配置 HP ADC FSM1 样式表的第 12 ~ 15 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.8. ADC_SAR2_PATT_TAB1_REG (0x0028)



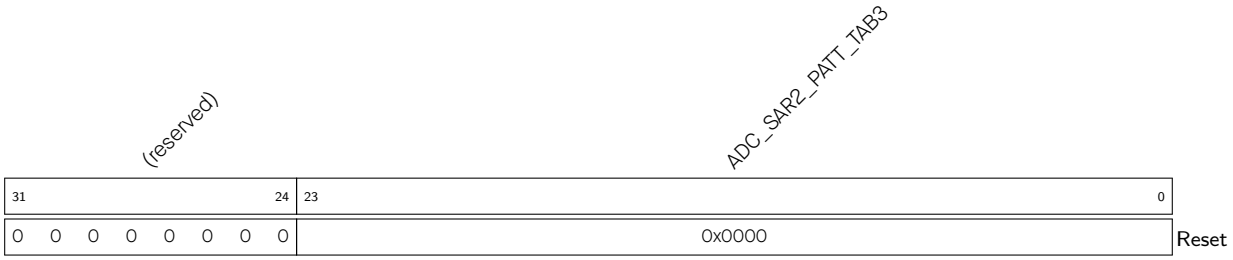
ADC_SAR2_PATT_TAB1 配置 HP ADC FSM2 样式表的第 0 ~ 3 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.9. ADC_SAR2_PATT_TAB2_REG (0x002C)



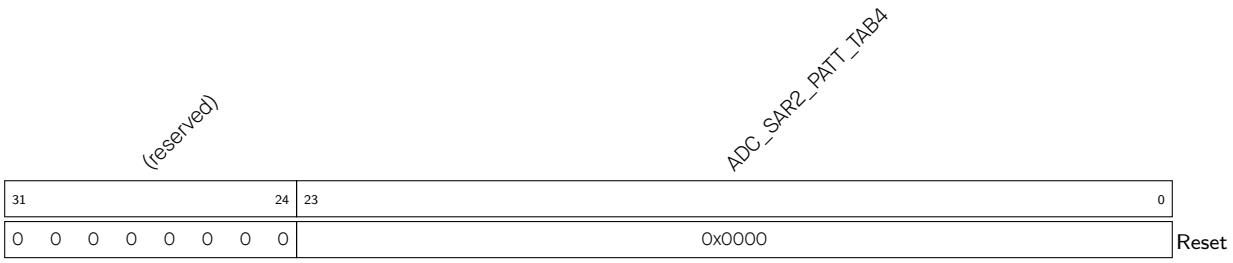
ADC_SAR2_PATT_TAB2 配置 HP ADC FSM2 样式表的第 4 ~ 7 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.10. ADC_SAR2_PATT_TAB3_REG (0x0030)



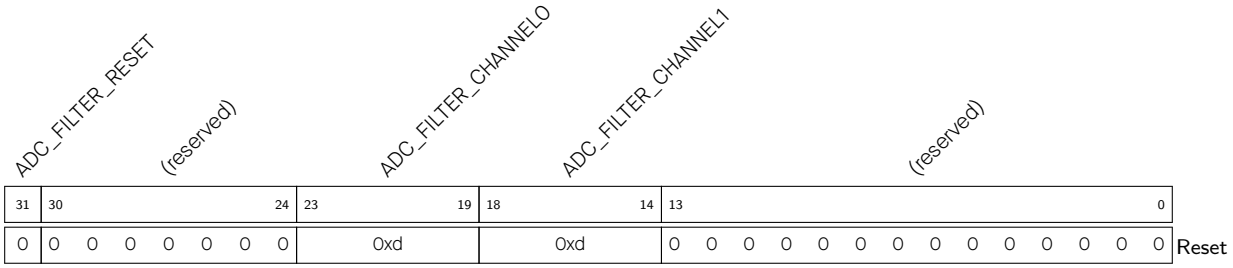
ADC_SAR2_PATT_TAB3 配置 HP ADC FSM2 样式表的第 8 ~ 11 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.11. ADC_SAR2_PATT_TAB4_REG (0x0034)



ADC_SAR2_PATT_TAB4 配置 HP ADC FSM2 样式表的第 12 ~ 15 个样式。每个样式占六位。具体见小节 65.5.6 *HP ADC 样式表*。(R/W)

Register 65.12. ADC_FILTER_CTRL0_REG (0x003C)



ADC_FILTER_CHANNEL0 配置滤波器 0 作用的 HP ADC 通道，0 ~ 7 对应 HP ADC1 的 0 ~ 7 通道，10 ~ 15 对应 HP ADC2 的 0 ~ 5 通道。(R/W)

ADC_FILTER_CHANNEL1 配置滤波器 1 作用的 HP ADC 通道，配置同上。(R/W)

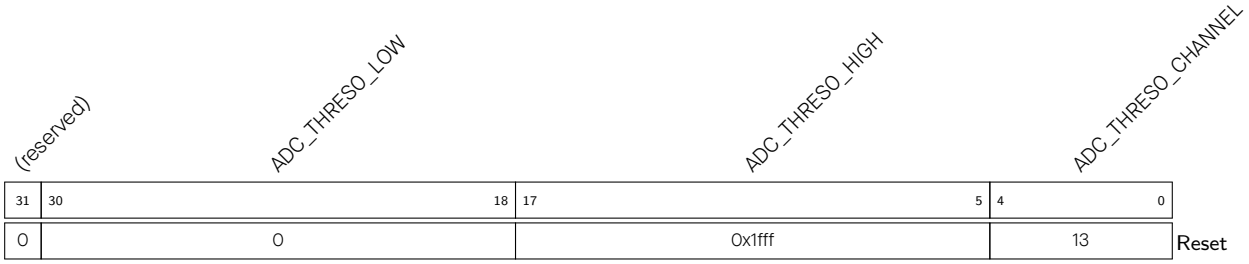
ADC_FILTER_RESET 配置是否复位滤波器。

0：无效

1：复位

(R/W)

Register 65.13. ADC_THRES0_CTRL_REG (0x0044)

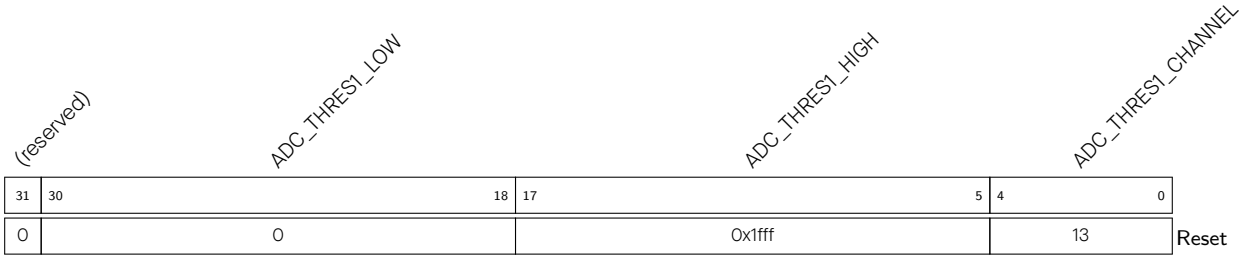


ADC_THRES0_CHANNEL 配置阈值监控器 0 作用的 HP ADC 通道，0 ~ 7 对应 HP ADC1 的 0 ~ 7 通道，10 ~ 15 对应 HP ADC2 的 0 ~ 5 通道。(R/W)

ADC_THRES0_HIGH 配置 HP ADC 阈值监控器 0 的高阈值。(R/W)

ADC_THRES0_LOW 配置 HP ADC 阈值监控器 0 的低阈值。(R/W)

Register 65.14. ADC_THRES1_CTRL_REG (0x0048)

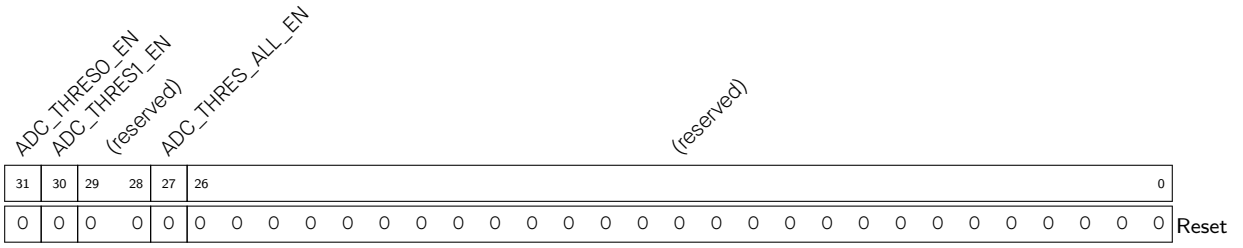


ADC_THRES1_CHANNEL 配置阈值监控器 1 作用的 HP ADC 通道，0 ~ 7 对应 HP ADC1 的 0 ~ 7 通道，10 ~ 15 对应 HP ADC2 的 0 ~ 5 通道。(R/W)

ADC_THRES1_HIGH 配置 HP ADC 阈值监控器 1 的高阈值。(R/W)

ADC_THRES1_LOW 配置 HP ADC 阈值监控器 1 的低阈值。(R/W)

Register 65.15. ADC_THRES_CTRL_REG (0x004C)



ADC_THRES_ALL_EN 配置是否使能所有已配置通道的阈值监控。

0：不使能

1：使能

(R/W)

ADC_THRES1_EN 配置是否使能阈值监控器 1。

0：不使能

1：使能

(R/W)

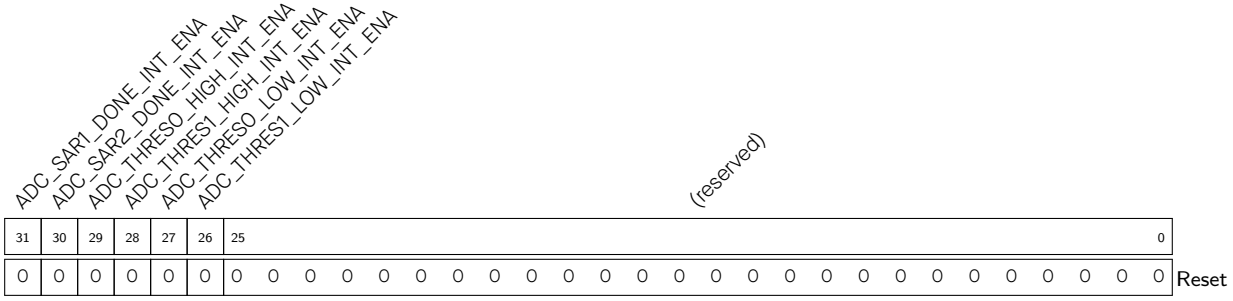
ADC_THRES0_EN 配置是否使能阈值监控器 0。

0：不使能

1：使能

(R/W)

Register 65.16. ADC_INT_ENA_REG (0x0050)



ADC_THRES1_LOW_INT_ENA 写 1 使能 ADC_THRES1_LOW_INT。(R/W)

ADC_THRES0_LOW_INT_ENA 写 1 使能 ADC_THRES0_LOW_INT。(R/W)

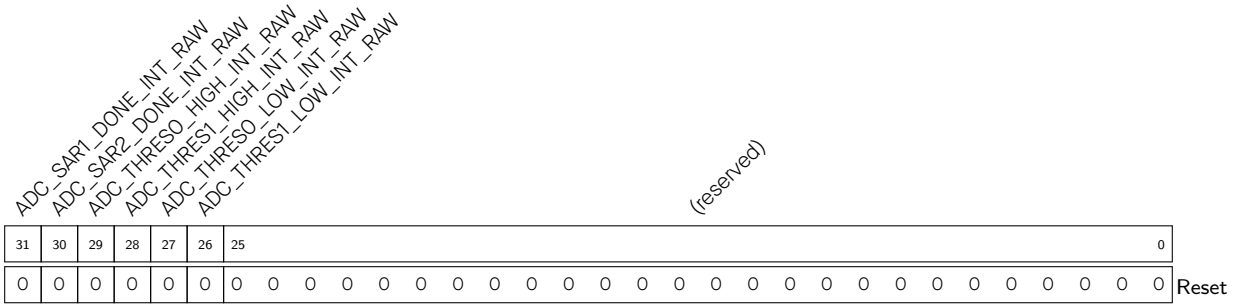
ADC_THRES1_HIGH_INT_ENA 写 1 使能 ADC_THRES1_HIGH_INT。(R/W)

ADC_THRES0_HIGH_INT_ENA 写 1 使能 ADC_THRES0_HIGH_INT。(R/W)

ADC_SAR2_DONE_INT_ENA 写 1 使能 ADC_SAR2_DONE_INT。(R/W)

ADC_SAR1_DONE_INT_ENA 写 1 使能 ADC_SAR1_DONE_INT。(R/W)

Register 65.17. ADC_INT_RAW_REG (0x0054)



ADC_THRES1_LOW_INT_RAW ADC_THRES1_LOW_INT 的原始中断状态。(R/WTC/SS)

ADC_THRES0_LOW_INT_RAW ADC_THRES0_LOW_INT 的原始中断状态。(R/WTC/SS)

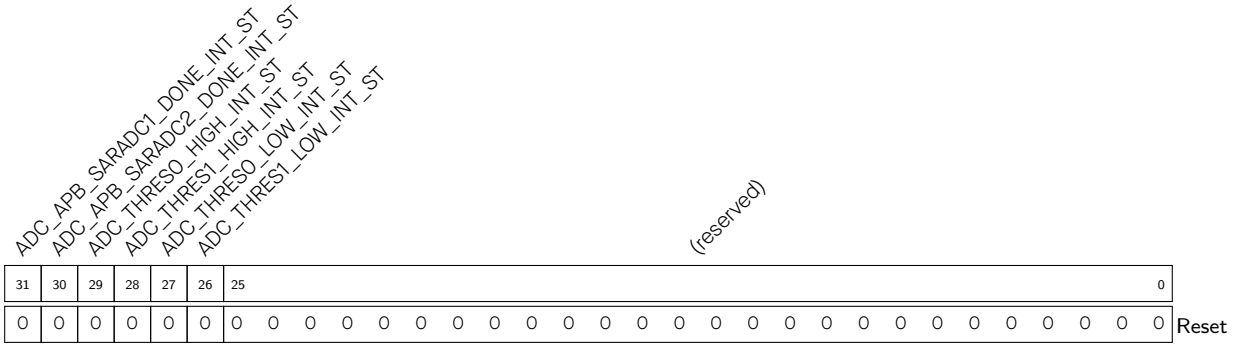
ADC_THRES1_HIGH_INT_RAW ADC_THRES1_HIGH_INT 的原始中断状态。(R/WTC/SS)

ADC_THRES0_HIGH_INT_RAW ADC_THRES0_HIGH_INT 的原始中断状态。(R/WTC/SS)

ADC_SAR2_DONE_INT_RAW ADC_SAR2_DONE_INT 的原始中断状态。(R/WTC/SS)

ADC_SAR1_DONE_INT_RAW ADC_SAR1_DONE_INT 的原始中断状态。(R/WTC/SS)

Register 65.18. ADC_INT_ST_REG (0x0058)



ADC_THRES1_LOW_INT_ST ADC_THRES1_LOW_INT 的屏蔽中断状态。(RO)

ADC_THRES0_LOW_INT_ST ADC_THRES0_LOW_INT 的屏蔽中断状态。(RO)

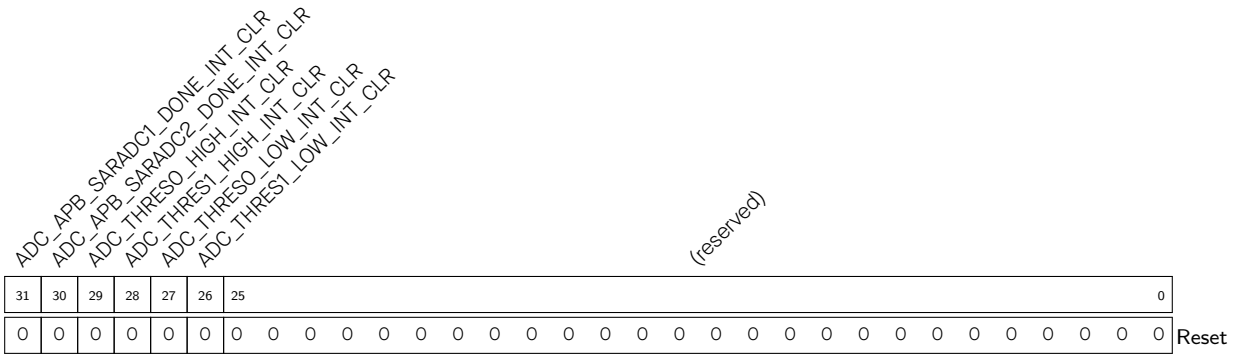
ADC_THRES1_HIGH_INT_ST ADC_THRES1_HIGH_INT 的屏蔽中断状态。(RO)

ADC_THRES0_HIGH_INT_ST ADC_THRES0_HIGH_INT 的屏蔽中断状态。(RO)

ADC_APB_SARADC2_DONE_INT_ST ADC_SAR2_DONE_INT 的屏蔽中断状态。(RO)

ADC_APB_SARADC1_DONE_INT_ST ADC_SAR1_DONE_INT 的屏蔽中断状态。(RO)

Register 65.19. ADC_INT_CLR_REG (0x005C)



ADC_THRES1_LOW_INT_CLR 写 1 清除 ADC_THRES1_LOW_INT。(WT)

ADC_THRES0_LOW_INT_CLR 写 1 清除 ADC_THRES0_LOW_INT。(WT)

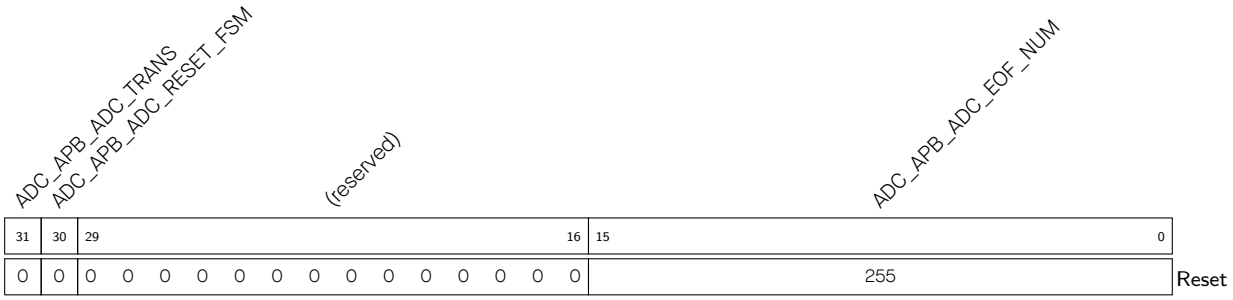
ADC_THRES1_HIGH_INT_CLR 写 1 清除 ADC_THRES1_HIGH_INT。(WT)

ADC_THRES0_HIGH_INT_CLR 写 1 清除 ADC_THRES0_HIGH_INT。(WT)

ADC_APB_SARADC2_DONE_INT_CLR 写 1 清除 ADC_SAR2_DONE_INT。(WT)

ADC_APB_SARADC1_DONE_INT_CLR 写 1 清除 ADC_SAR1_DONE_INT。(WT)

Register 65.20. ADC_DMA_CONF_REG (0x0060)



ADC_APB_ADC_EOF_NUM 当 HP ADC 采样得到的数据个数达到该寄存器配置值时，发送给 GDMA 的 EOF 标志位将被拉高。(R/W)

ADC_APB_ADC_RESET_FSM 配置是否复位 HP ADC 控制器状态。

0：无效

1：复位

(R/W)

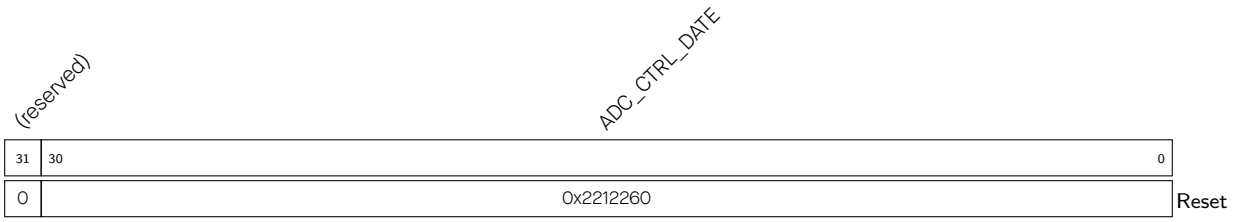
ADC_APB_ADC_TRANS 配置 HP ADC 控制器是否使用 GDMA。

0：无效

1：使用 GDMA

(R/W)

Register 65.21. ADC_CTRL_DATE_REG (0x03FC)



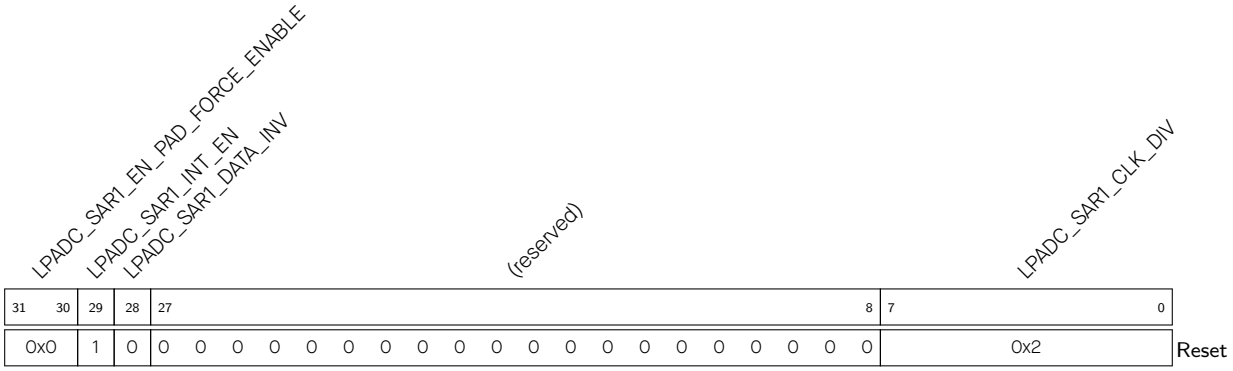
ADC_CTRL_DATE HP ADC 版本寄存器。(RO)

65.10.2 LP ADC 寄存器

以下所有地址均为相对于 LP ADC 基地址的地址偏移量（相对地址），具体基地址请见章节 9 系统和存储器 中的表 9.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 65.22. LPADC_READER1_CTRL_REG (0x0000)



LPADC_SAR1_CLK_DIV 配置 LP ADC1 的时钟 LPADC_SARCLK 的分频系数。(R/W)

LPADC_SAR1_DATA_INV 配置是否反转 LP ADC1 数据。

0：无效

1：反转

(R/W)

LPADC_SAR1_INT_EN 置 1 使能 LP ADC1 发送中断。(R/W)

LPADC_SAR1_EN_PAD_FORCE_ENABLE 配置是否使用软件强制使能 LPADC_SAR1_EN_PAD。

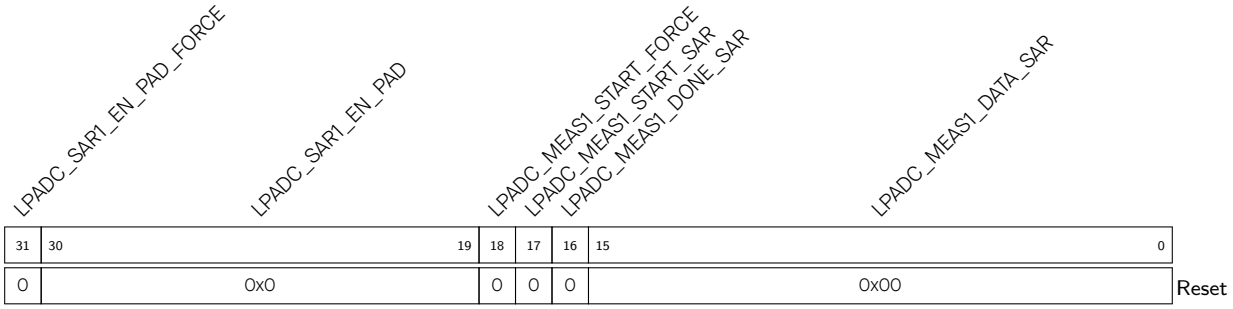
2：强制关闭

3：强制使能

其他值：硬件控制（采样时使能）

(R/W)

Register 65.23. LPADC_MEAS1_CTRL2_REG (0x000C)



LPADC_MEAS1_DATA_SAR 存储 LP ADC1 单次采样的转换数据。(RO)

LPADC_MEAS1_DONE_SAR 为 1 代表 LP ADC1 数据转换完成。(RO)

LPADC_MEAS1_START_SAR 配置是否使用软件启动 LP ADC1。

0：无效

1：使用软件启动

仅当 LPADC_MEAS1_START_FORCE = 1 时有效。

(R/W)

LPADC_MEAS1_START_FORCE 配置是否使用软件启动 LP ADC1 采样。

0：选择使用 FSM 启动 LP ADC1 采样

1：选择使用软件启动 LP ADC1 采样

(R/W)

LPADC_SAR1_EN_PAD 选择 LP ADC1 单次采样的通道，0 ~ 7 对应通道 0 ~ 7。(R/W)

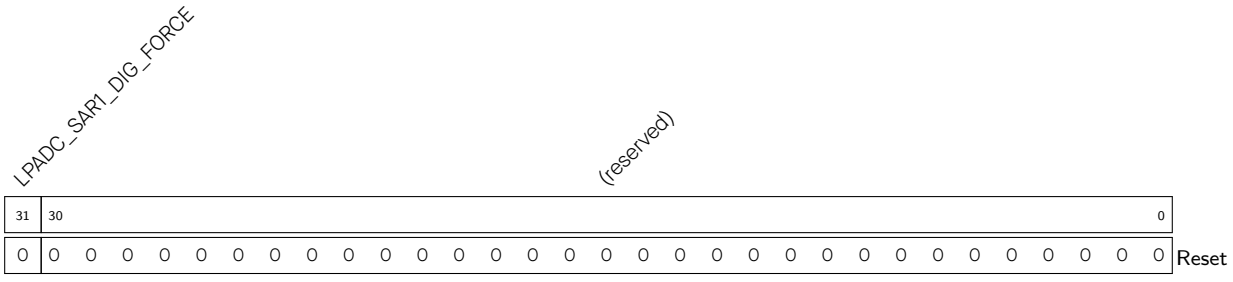
LPADC_SAR1_EN_PAD_FORCE 配置是否使用软件控制选择 LP ADC1 单次采样的通道。

0：无效

1：使用软件控制

(R/W)

Register 65.24. LPADC_MEAS1_MUX_REG (0x0010)



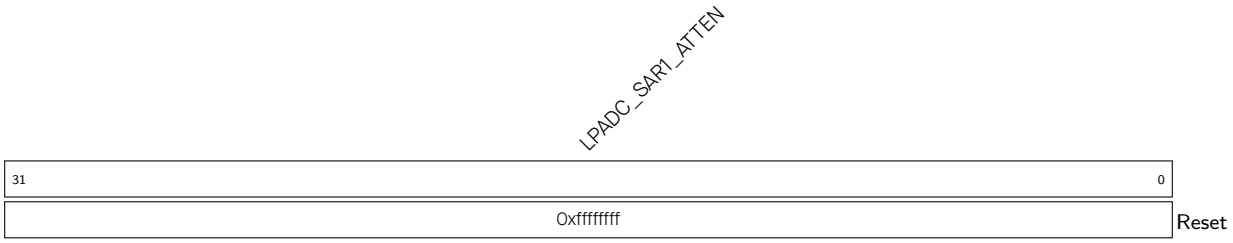
LPADC_SAR1_DIG_FORCE 配置是否由 HP ADC1 控制器控制 SAR ADC1。

0：由 LP ADC1 控制器控制。

1：由 HP ADC1 控制器控制。

(R/W)

Register 65.25. LPADC_ATTEN1_REG (0x0014)



LPADC_SAR1_ATTEN 配置 LP ADC1 单次采样的衰减值。

0：0 dB

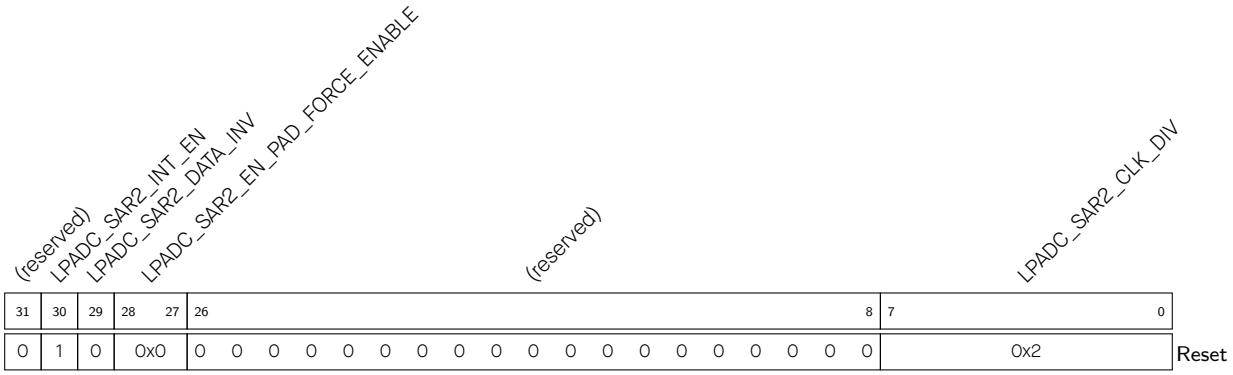
1：2.5 dB

2：6 dB

3：12 dB

(R/W)

Register 65.26. LPADC_READER2_CTRL_REG (0x0024)



LPADC_SAR2_CLK_DIV 配置 LP ADC2 的时钟 LPADC_SARCLK 的分频系数。(R/W)

LPADC_SAR2_EN_PAD_FORCE_ENABLE 配置是否使用软件强制使能 LPADC_SAR2_EN_PAD。

2：强制关闭

3：强制使能

其他值：硬件控制（采样时使能）

(R/W)

LPADC_SAR2_DATA_INV 配置是否反转 LP ADC2 数据。

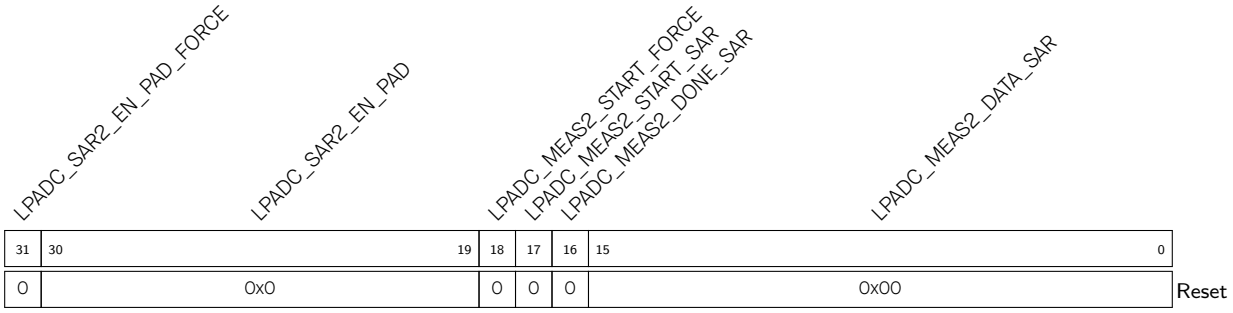
0：无效

1：反转 SAR ADC 数据

(R/W)

LPADC_SAR2_INT_EN 置 1 使能 LP ADC2 发送中断。(R/W)

Register 65.27. LPADC_MEAS2_CTRL2_REG (0x0030)



LPADC_MEAS2_DATA_SAR 存储 LP ADC2 单次采样的转换数据。(RO)

LPADC_MEAS2_DONE_SAR 为 1 代表 LP ADC2 数据转换完成。(RO)

LPADC_MEAS2_START_SAR 配置是否使用软件启动 LP ADC2。

0：无效

1：使用软件启动

仅当 LPADC_MEAS2_START_FORCE = 1 时有效。

(R/W)

LPADC_MEAS2_START_FORCE 配置是否使用软件启动 LP ADC2 采样。

0：选择使用 FSM 启动 LP ADC2 采样

1：选择使用软件启动 LP ADC2 采样

(R/W)

LPADC_SAR2_EN_PAD 选择 LP ADC2 单次采样的通道，2 ~ 7 对应通道 0 ~ 5。(R/W)

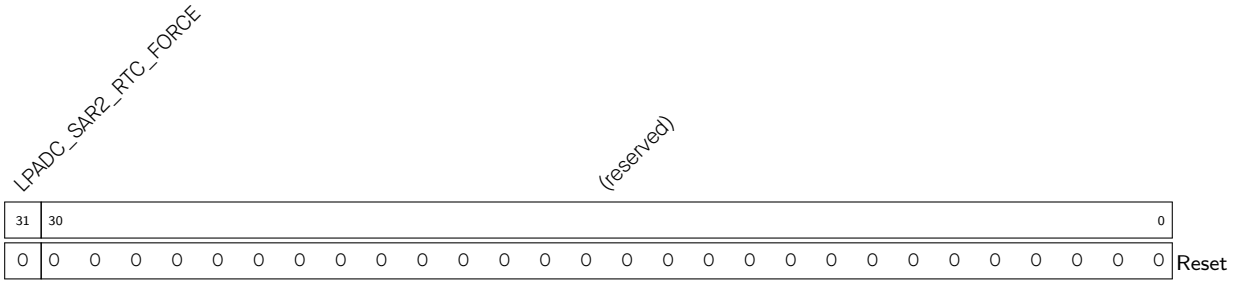
LPADC_SAR2_EN_PAD_FORCE 配置是否使用软件控制选择 LP ADC2 单次采样的通道。

0：无效

1：使用软件控制

(R/W)

Register 65.28. LPADC_MEAS2_MUX_REG (0x0034)



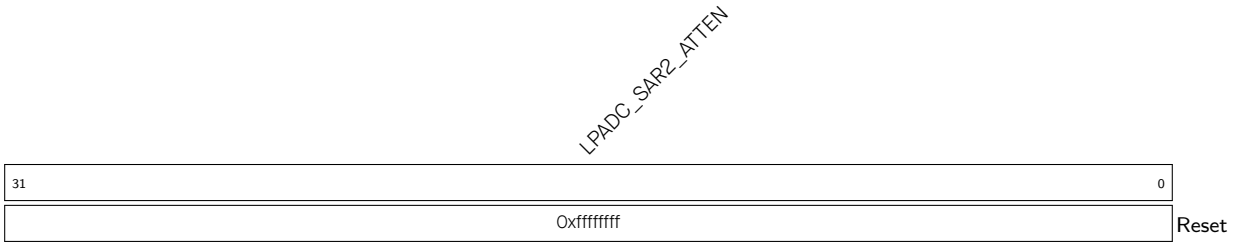
LPADC_SAR2_RTC_FORCE 配置是否禁止 HP ADC2 控制 SAR ADC2。

0：不禁止 HP ADC2 控制器控制。

1：禁止 HP ADC2 控制器控制。

(R/W)

Register 65.29. LPADC_ATTEN2_REG (0x0038)



LPADC_SAR2_ATTEN 配置 LP ADC2 单次采样的衰减值。

0：0 dB

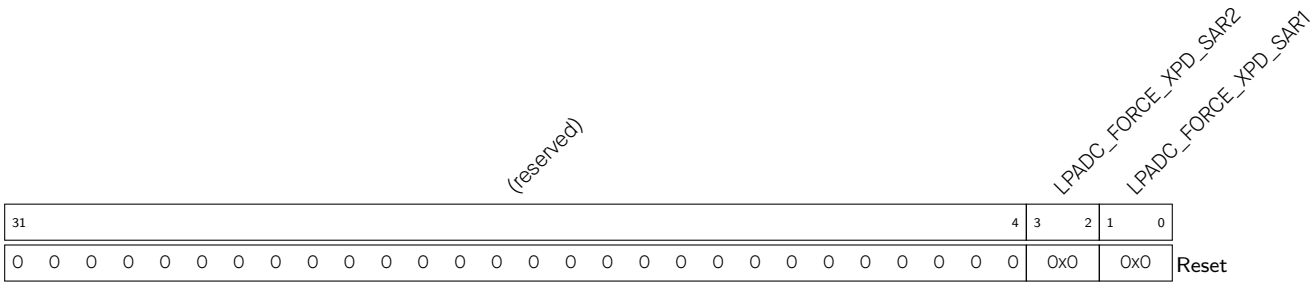
1：2.5 dB

2：6 dB

3：12 dB

(R/W)

Register 65.30. LPADC_FORCE_WPD_SAR_REG (0x003C)



LPADC_FORCE_XPD_SAR1 配置 LP ADC1 的上电控制。

3：软件控制，强制上电

2：软件控制，强制断电

1/0：硬件控制

(R/W)

LPADC_FORCE_XPD_SAR2 配置 LP ADC2 的上电控制。

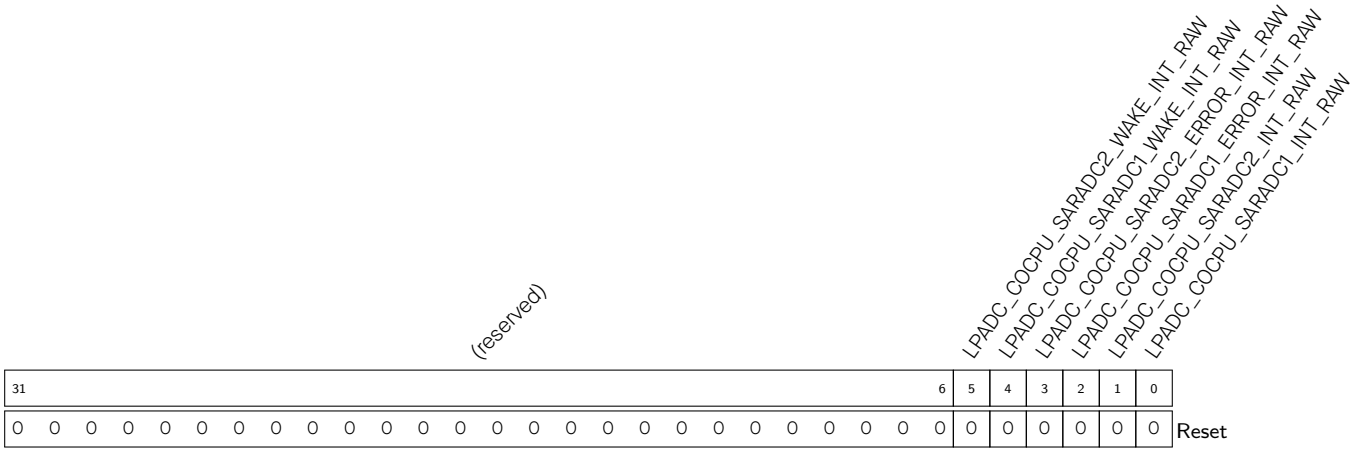
3：软件控制，强制上电

2：软件控制，强制断电

1/0：硬件控制

(R/W)

Register 65.31. LPADC_COCPU_INT_RAW_REG (0x0048)



LPADC_COCPU_SARADC1_INT_RAW LPADC_COCPU_SARADC1_DONE_INT 的原始中断状态。(R/WTC/SS)

LPADC_COCPU_SARADC2_INT_RAW LPADC_COCPU_SARADC2_DONE_INT 的原始中断状态。(R/WTC/SS)

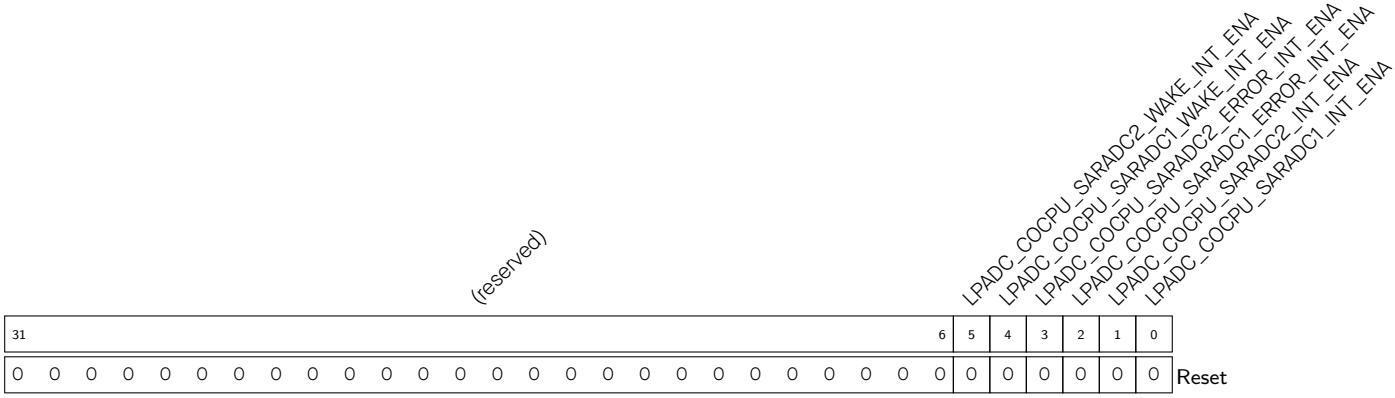
LPADC_COCPU_SARADC1_ERROR_INT_RAW LPADC_COCPU_SARADC1_ERROR_INT 的原始中断状态。(R/WTC/SS)

LPADC_COCPU_SARADC2_ERROR_INT_RAW LPADC_COCPU_SARADC2_ERROR_INT 的原始中断状态。(R/WTC/SS)

LPADC_COCPU_SARADC1_WAKE_INT_RAW LPADC_COCPU_SARADC1_WAKE_INT 的原始中断状态。(R/WTC/SS)

LPADC_COCPU_SARADC2_WAKE_INT_RAW LPADC_COCPU_SARADC2_WAKE_INT 的原始中断状态。(R/WTC/SS)

Register 65.32. LPADC_INT_ENA_REG (0x004C)



LPADC_COCPU_SARADC1_INT_ENA 置 1 使能 LPADC_COCPU_SARADC1_DONE_INT。(R/WTC)

LPADC_COCPU_SARADC2_INT_ENA 置 1 使能 LPADC_COCPU_SARADC2_DONE_INT。(R/WTC)

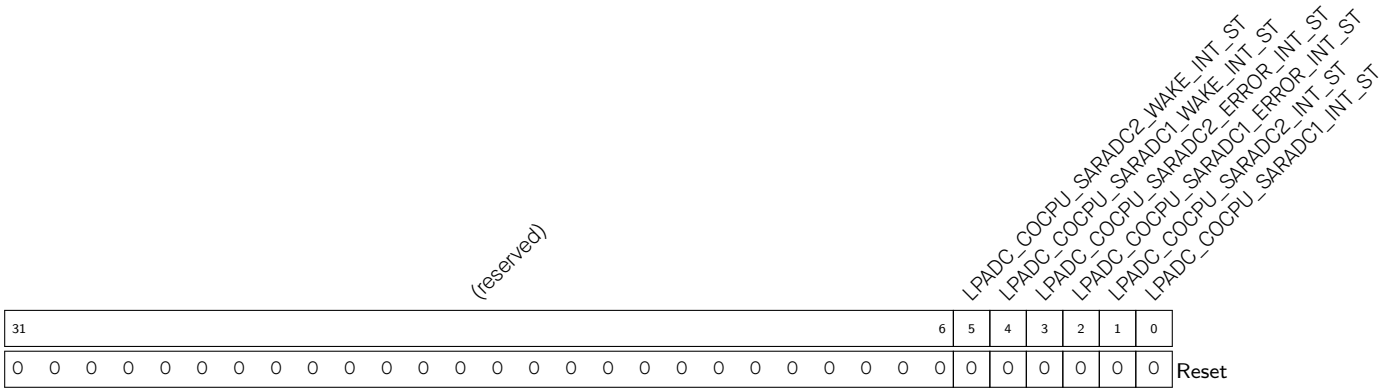
LPADC_COCPU_SARADC1_ERROR_INT_ENA 置 1 使能 LPADC_COCPU_SARADC1_ERROR_INT。(R/WTC)

LPADC_COCPU_SARADC2_ERROR_INT_ENA 置 1 使能 LPADC_COCPU_SARADC2_ERROR_INT。(R/WTC)

LPADC_COCPU_SARADC1_WAKE_INT_ENA 置 1 使能 LPADC_COCPU_SARADC1_WAKE_INT。(R/WTC)

LPADC_COCPU_SARADC2_WAKE_INT_ENA 置 1 使能 LPADC_COCPU_SARADC2_WAKE_INT。(R/WTC)

Register 65.33. LPADC_INT_ST_REG (0x0050)



LPADC_COCPU_SARADC1_INT_ST LPADC_COCPU_SARADC1_DONE_INT 的屏蔽中断状态。(RO)

LPADC_COCPU_SARADC2_INT_ST LPADC_COCPU_SARADC2_DONE_INT 的屏蔽中断状态。(RO)

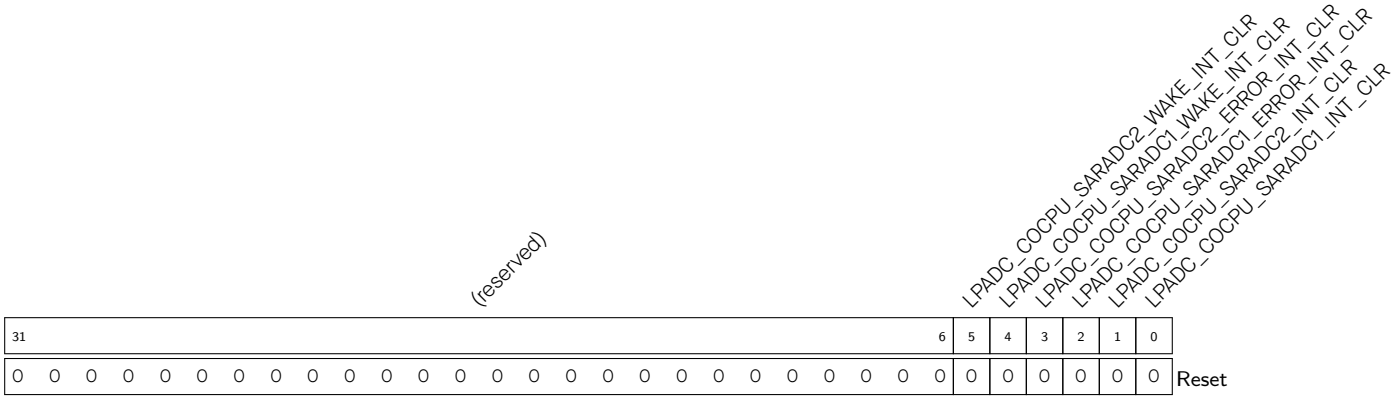
LPADC_COCPU_SARADC1_ERROR_INT_ST LPADC_COCPU_SARADC1_ERROR_INT 的屏蔽中断状态。(RO)

LPADC_COCPU_SARADC2_ERROR_INT_ST LPADC_COCPU_SARADC2_ERROR_INT 的屏蔽中断状态。(RO)

LPADC_COCPU_SARADC1_WAKE_INT_ST LPADC_COCPU_SARADC1_WAKE_INT 的屏蔽中断状态。(RO)

LPADC_COCPU_SARADC2_WAKE_INT_ST LPADC_COCPU_SARADC2_WAKE_INT 的屏蔽中断状态。(RO)

Register 65.34. LPADC_INT_CLR_REG (0x0054)



LPADC_COCPU_SARADC1_INT_CLR 写 1 清除 LPADC_COCPU_SARADC1_DONE_INT。(WT)

LPADC_COCPU_SARADC2_INT_CLR 写 1 清除 LPADC_COCPU_SARADC2_DONE_INT。(WT)

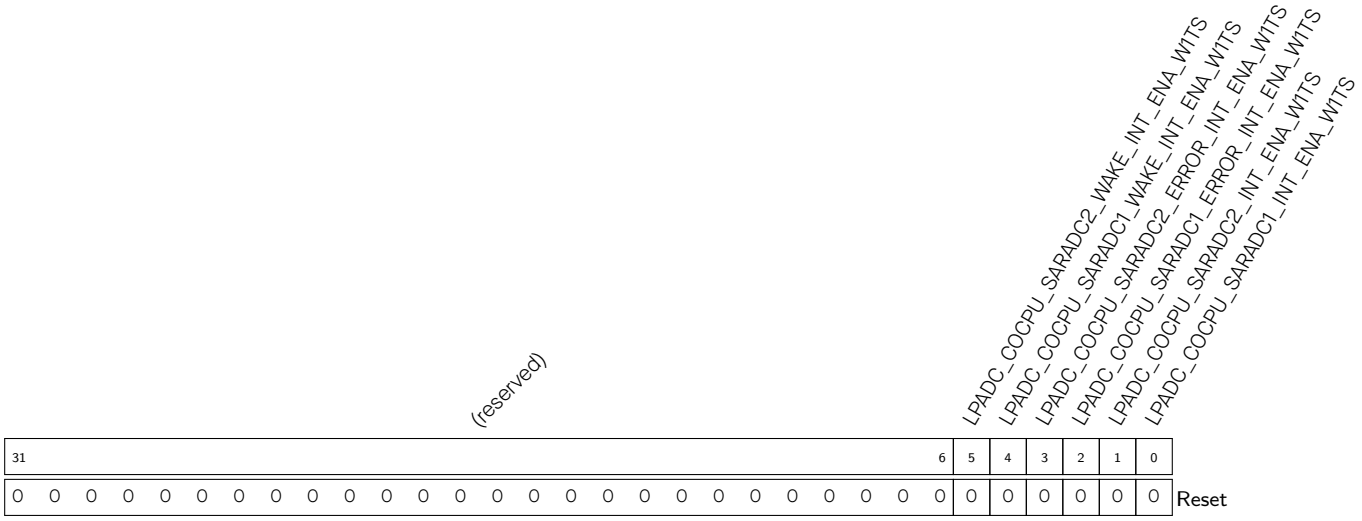
LPADC_COCPU_SARADC1_ERROR_INT_CLR 写 1 清除 LPADC_COCPU_SARADC1_ERROR_INT。(WT)

LPADC_COCPU_SARADC2_ERROR_INT_CLR 写 1 清除 LPADC_COCPU_SARADC2_ERROR_INT。(WT)

LPADC_COCPU_SARADC1_WAKE_INT_CLR 写 1 清除 LPADC_COCPU_SARADC1_WAKE_INT。(WT)

LPADC_COCPU_SARADC2_WAKE_INT_CLR 写 1 清除 LPADC_COCPU_SARADC2_WAKE_INT。(WT)

Register 65.35. LPADC_INT_ENA_W1TS_REG (0x0058)



LPADC_COCPU_SARADC1_INT_ENA_W1TS 写 1 使能 LPADC_COCPU_SARADC1_INT_ENA。(WT)

LPADC_COCPU_SARADC2_INT_ENA_W1TS 写 1 使能 LPADC_COCPU_SARADC2_INT_ENA。(WT)

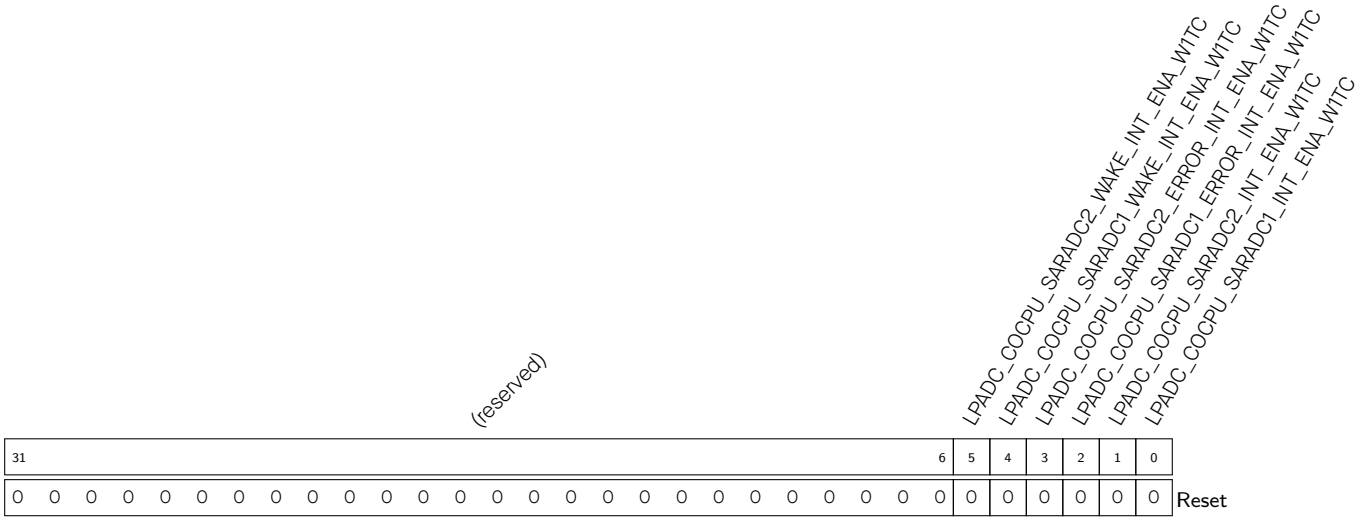
LPADC_COCPU_SARADC1_ERROR_INT_ENA_W1TS 写 1 使能 LPADC_COCPU_SARADC1_ERROR_INT_ENA。(WT)

LPADC_COCPU_SARADC2_ERROR_INT_ENA_W1TS 写 1 使能 LPADC_COCPU_SARADC2_ERROR_INT_ENA。(WT)

LPADC_COCPU_SARADC1_WAKE_INT_ENA_W1TS 写 1 使能 LPADC_COCPU_SARADC1_WAKE_INT_ENA。(WT)

LPADC_COCPU_SARADC2_WAKE_INT_ENA_W1TS 写 1 使能 LPADC_COCPU_SARADC2_WAKE_INT_ENA。(WT)

Register 65.36. LPADC_INT_ENA_W1TC_REG (0x005C)



LPADC_COCPU_SARADC1_INT_ENA_W1TC 写 1 清除 LPADC_COCPU_SARADC1_INT_ENA。(WT)

LPADC_COCPU_SARADC2_INT_ENA_W1TC 写 1 清除 LPADC_COCPU_SARADC2_INT_ENA。(WT)

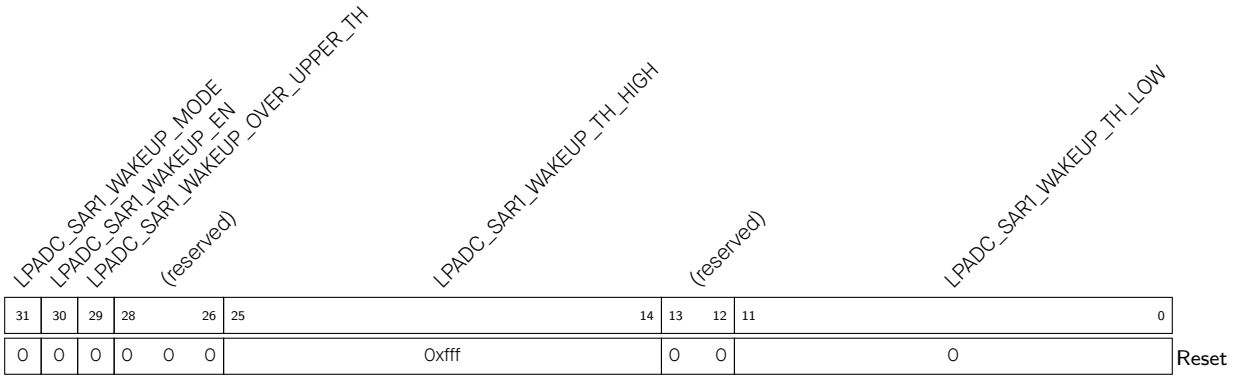
LPADC_COCPU_SARADC1_ERROR_INT_ENA_W1TC 写 1 清除 LPADC_COCPU_SARADC1_ERROR_INT_ENA。(WT)

LPADC_COCPU_SARADC2_ERROR_INT_ENA_W1TC 写 1 清除 LPADC_COCPU_SARADC2_ERROR_INT_ENA。(WT)

LPADC_COCPU_SARADC1_WAKE_INT_ENA_W1TC 写 1 清除 LPADC_COCPU_SARADC1_WAKE_INT_ENA。(WT)

LPADC_COCPU_SARADC2_WAKE_INT_ENA_W1TC 写 1 清除 LPADC_COCPU_SARADC2_WAKE_INT_ENA。(WT)

Register 65.37. LPADC_WAKEUP1_REG (0x0060)



LPADC_SAR1_WAKEUP_TH_LOW 配置 LP ADC1 唤醒功能的低阈值。(R/W)

LPADC_SAR1_WAKEUP_TH_HIGH 配置 LP ADC1 唤醒功能的高阈值。(R/W)

LPADC_SAR1_WAKEUP_OVER_UPPER_TH 表示是否超出高阈值导致唤醒事件。(RO)

LPADC_SAR1_WAKEUP_EN 配置是否为 LP ADC1 启用唤醒功能。

0：不启用

1：启用

(R/W)

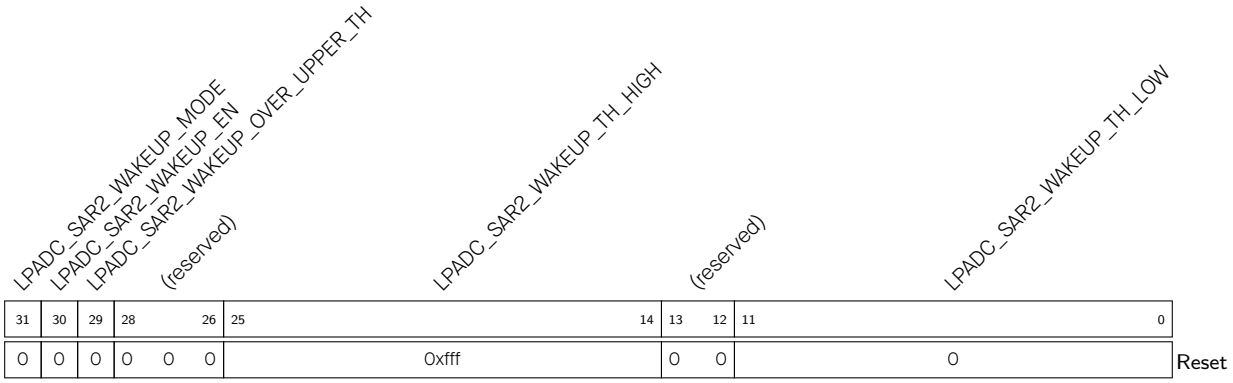
LPADC_SAR1_WAKEUP_MODE 配置自动监测唤醒模式。仅当 LPADC_SAR1_WAKEUP_EN=1 时有效。

0：绝对值模式

1：变化量模式

(R/W)

Register 65.38. LPADC_WAKEUP2_REG (0x0064)



LPADC_SAR2_WAKEUP_TH_LOW 配置 LP ADC2 唤醒功能的低阈值。(R/W)

LPADC_SAR2_WAKEUP_TH_HIGH 配置 LP ADC2 唤醒功能的高阈值。(R/W)

LPADC_SAR2_WAKEUP_OVER_UPPER_TH 表示是否超出高阈值导致唤醒事件。(RO)

LPADC_SAR2_WAKEUP_EN 配置是否为 LP ADC2 启用唤醒功能。

0：不启用

1：启用

(R/W)

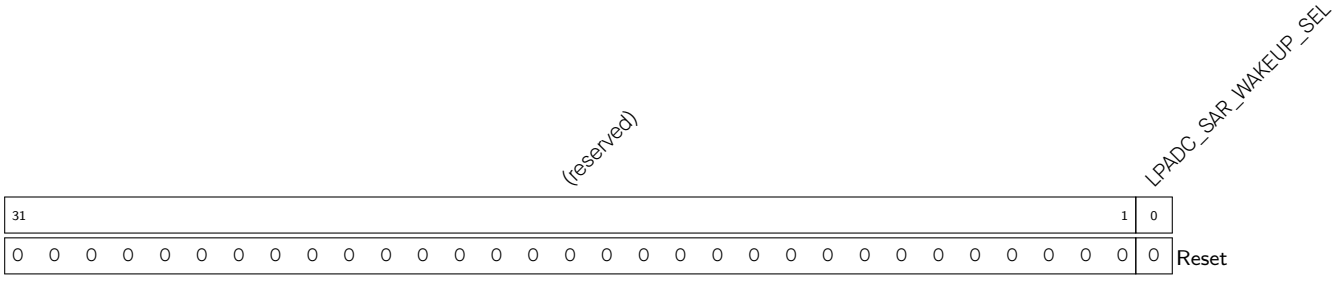
LPADC_SAR2_WAKEUP_MODE 配置自动监测唤醒模式。仅当 LPADC_SAR2_WAKEUP_EN=1 时有效。

0：绝对值模式

1：变化量模式

(R/W)

Register 65.39. LPADC_WAKEUP_SEL_REG (0x0068)



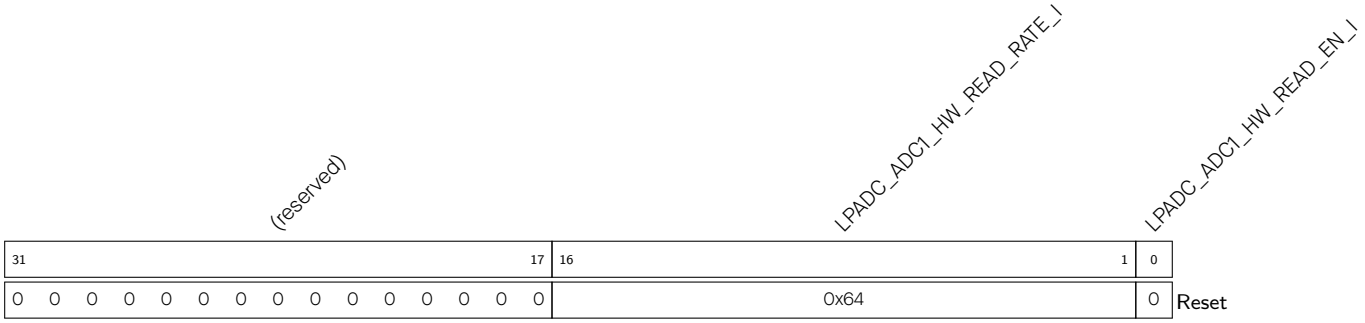
LPADC_SAR_WAKEUP_SEL 配置 LP ADC1 或 LP ADC2 的唤醒功能。

0：LP ADC1

1：LP ADC2

(R/W)

Register 65.40. LPADC_SAR1_HW_WAKEUP_REG (0x006C)



LPADC_ADC1_HW_READ_EN_I 配置是否使能 LP ADC1 自动监测。

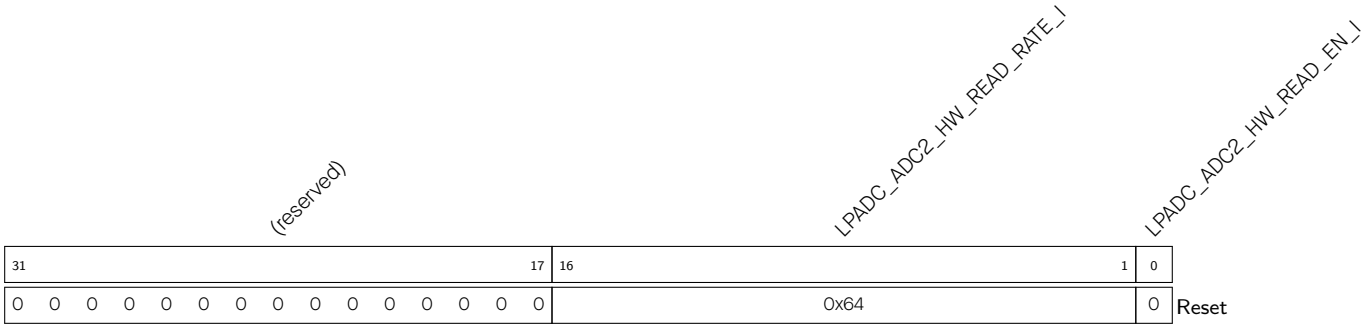
0：不使能

1：使能

(R/W)

LPADC_ADC1_HW_READ_RATE_I 配置 LP ADC1 硬件自动监测的采样率。采样周期 = 配置值 × LP ADC1 的工作时钟周期。(R/W)

Register 65.41. LPADC_SAR2_HW_WAKEUP_REG (0x0070)



LPADC_ADC2_HW_READ_EN_I 配置是否使能 LP ADC2 自动监测。

0：不使能

1：使能

(R/W)

LPADC_ADC2_HW_READ_RATE_I 配置 LP ADC2 硬件自动监测的采样率。采样周期 = 配置值 × LP ADC2 的工作时钟周期。(R/W)

第 66 章

模拟电压比较器

66.1 概述

ESP32-P4 芯片集成了两个模拟电压比较器，即模拟电压比较器 *0* 和模拟电压比较器 *1*。模拟电压比较器依靠支持电压比较功能的特殊芯片焊盘 (PAD) 实现，用于监测 PAD 上的电压变化。每个模拟电压比较器都有外部两个电压接入点，分为主电压和参考电压。模拟电压比较器输出的电压比较结果可以作为 ETM（事件任务矩阵）事件来驱动其他外设的 ETM 任务，也可以触发中断。

说明：
本章节提供的信息适用于模拟电压比较器 *0* 和模拟电压比较器 *1*。除非另有说明，本章节中的模拟电压比较器同时指代模拟电压比较器 *0* 和模拟电压比较器 *1*。

66.2 主要特性

模拟电压比较器具有以下主要特性：

- 支持电压比较功能
  - 电压比较模式可配置
  - 内部参考电压值可配置
- 支持电压比较中断
- 支持 ETM 事件

66.3 系统架构

图 66.3-1 展示了模拟电压比较器的系统架构。

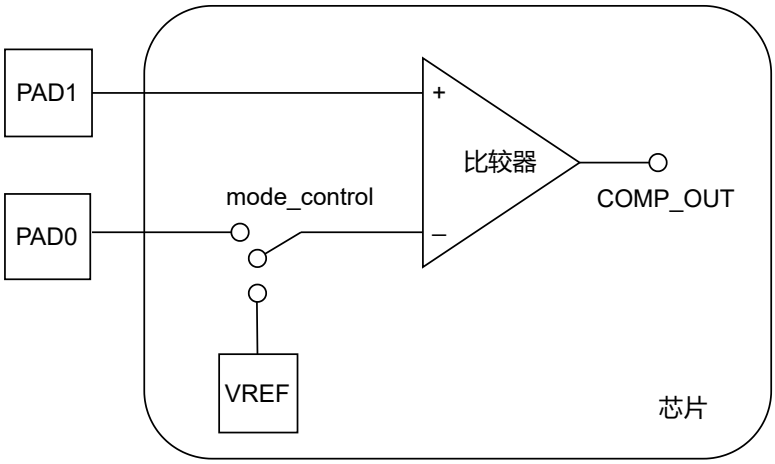


图 66.3-1. 模拟电压比较器系统框图

如上图所示，模拟电压比较器的结构如下：

- 两个 PAD 接口：
  - PAD1：主电压接口
  - PAD0：外部参考电压接口
- VREF：内部参考电压接口
- mode_control：寄存器控制的选择开关，用于选择参考电压的输入源
- COMP_OUT：比较器输出的电压比较结果，为片上内部信号，外部无法直接获取
  - 若主电压高于参考电压，COMP_OUT 信号输出为高电平
  - 若主电压低于参考电压，COMP_OUT 信号输出为低电平

66.4 功能描述

模拟电压比较器具有以下功能：

- **电压比较**

  模拟电压比较器采用的是支持电压比较功能的特殊 PAD，这种特殊 PAD 由两个通用 PAD 组成。下表列出了模拟电压比较器的 PAD 与 GPIO 管脚之间的映射关系。

表 66.4-1. 模拟电压比较器 PAD 和 GPIO 之间的映射关系

模拟电压比较器	PAD0	PAD1
模拟电压比较器 *0*	GPIO51	GPIO52
模拟电压比较器 *1*	GPIO53	GPIO54

  电压比较 PAD 的比较模式和参考电压可配置，具体配置流程见小节 66.7 配置流程。

  电压比较结果输出为 COMP_OUT 信号。

- **电压比较模式控制**

模拟电压比较器将主电压与参考电压进行比较，参考电压可选内部参考电压和外部参考电压。根据参考电压的选择不同，电压比较模式分为两种：

- 主电压与外部参考电压比较；
- 主电压与内部参考电压比较；

电压比较模式可通过 LPSYSTEM_MODE_COMP*n* (*n* = 0 ~ 1) 配置。

- **参考电压配置**

  内部参考电压范围为 0 ~ (0.7 × VDD_IO_6) V，步长为 0.1 × VDD_IO_6 V，其中 VDD_IO_6 为模拟电压比较器的供电电压，等于芯片的电源电压（通常为 3.3 V）。内部参考电压值可通过寄存器 LPSYSTEM_DREF_COMP*n* (*n* = 0 ~ 1) 配置。

  外部参考电压直接从 PAD 接入，无需配置，输入范围与内部参考电压一致，为 0 ~ (0.7 x VDD_IO_6) V。

- **电压比较中断处理**

  电压比较的输出值 COMP_OUT 信号为高电平或低电平，只要输出值发生变化，就会产生对应的中断信号。

66.5 事件任务矩阵

在 ESP32-P4 中，模拟电压比较器支持 ETM 功能，即可以通过模拟电压比较器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 14 事件任务矩阵 (ETM)。这里仅介绍与模拟电压比较器相关的 ETM 事件。

模拟电压比较器可产生的 ETM 事件有：

- GPIO_EVT_ZERO_DET_POS*n*：表示 COMP_OUT 信号由低电平变为高电平，即主电压从低于参考电压变为高于参考电压。
- GPIO_EVT_ZERO_DET_NEG*n*：表示 COMP_OUT 信号由高电平变为低电平，即主电压从高于参考电压变为低于参考电压。

模拟电压比较器不支持任何 ETM 任务。

66.6 中断

ESP32-P4 的模拟电压比较器可以生成 GPIO_PAD_COMP_INT 中断信号并发送给中断矩阵。

这一中断信号由模拟电压比较器的内部中断源生成。表 66.6-1 列出了模拟电压比较器的内部中断源、中断触发条件、以及生成的中断信号。

表 66.6-1. 模拟电压比较器的内部中断源

内部中断源	触发条件	中断信号
GPIO_COMP*n*_ALL_INT (*n* = 0 ~ 1)	COMP_OUT 信号发生跳变	GPIO_PAD_COMP_INT
GPIO_COMP*n*_NEG_INT (*n* = 0 ~ 1)	COMP_OUT 信号由高电平变为低电平	GPIO_PAD_COMP_INT
GPIO_COMP*n*_POS_INT (*n* = 0 ~ 1)	COMP_OUT 信号由低电平变为高电平	GPIO_PAD_COMP_INT

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 13 中断矩阵 > 13.2 *ESP32-P4* 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 10.19.1 *HP GPIO* 交换矩阵寄存器列表。

66.7 配置流程

软件配置电压比较器功能的流程如下：

说明：
有关本节提及的寄存器字段的更多信息，请参阅章节 21 系统寄存器 (SYSREG)。

1. 将 LPSYSTEM_XPD_COMP*n* (*n* = 0 ~ 1) 设置为 1，启用电压比较器功能。
2. 禁用 PAD1 的正常数字 PAD 功能，包括 IE、OE、WPU 和 WPD，如果使用外部参考电压，则同时仅用 PAD2 的正常数字 PAD 功能。具体的配置方式请参考章节 10 GPIO 交换矩阵和 IO MUX。
3. 配置比较模式，将 LPSYSTEM_MODE_COMP*n* (*n* = 0 ~ 1) 设置为：
   - 0：选择主电压和内部参考电压进行比较；
   - 1：选择主电压和外部参考电压进行比较。
4. 通过 LPSYSTEM_DREF_COMP*n* (*n* = 0 ~ 1) 配置内部参考电压，可调电压范围为 0 ~ (0.7 × VDD_IO_6) V，步长为 0.1 × VDD_IO_6 V。

第 IX 卷

附录

附录部分包括以下内容：

- 相关文档和资源
- 词汇列表
- 如何配置寄存器的保留域
- 中断配置寄存器
- 修订历史

相关文档和资源

相关文档

- 《ESP32-P4 系列芯片技术规格书》- 提供 ESP32-P4 芯片的硬件技术规格。
- 《ESP32-P4 硬件设计指南》 – 提供基于 ESP32-P4 芯片的产品设计规范。
- 《ESP32-P4 系列芯片勘误表》 – 描述 ESP32-P4 系列芯片的已知错误。
- 技术参考手册 LaTeX 源码 – 乐鑫芯片技术参考手册 LaTeX 源码的公开仓库。
  https://github.com/espressif/esp-technical-reference-manual-latex
- 证书
  https://documentation.espressif.com/zh/documentList?t= 证书 &eol=false
- ESP32-P4 产品/工艺变更通知 (PCN)
  https://documentation.espressif.com/zh/documentList?t=PCN&s=ESP32-P4
- ESP32-P4 公告 – 提供有关安全、bug、兼容性、器件可靠性的信息
  https://documentation.espressif.com/zh/documentList?t= 公告 &s=ESP32-P4
- 文档更新和订阅通知
  https://documentation.espressif.com/zh/documentList?eol=false

开发者社区

- 《ESP32-P4 ESP-IDF 编程指南》 – ESP-IDF 开发框架的文档中心。
- ESP-IDF 及 GitHub 上的其它开发框架
  https://github.com/espressif
- ESP32 论坛 – 工程师对工程师 (E2E) 的社区，您可以在这里提出问题、解决问题、分享知识、探索观点。
  https://esp32.com/
- *ESP-FAQ* – 由乐鑫官方推出的针对常见问题的总结。
  https://espressif.com/projects/esp-faq/zh_CN/latest/index.html
- *The ESP Journal* – 分享乐鑫工程师的最佳实践、技术文章和工作随笔。
  https://blog.espressif.com/
- SDK 和演示、App、工具、AT 等下载资源
  https://espressif.com/zh-hans/support/download/sdks-demos

产品

- ESP32-P4 系列芯片 – ESP32-P4 全系列芯片。
  https://espressif.com/zh-hans/products/socs?id=ESP32-P4
- ESP32-P4 系列开发板 – ESP32-P4 全系列开发板。
  https://espressif.com/zh-hans/products/devkits?id=ESP32-P4
- ESP Product Selector（乐鑫产品选型工具）– 通过筛选性能参数、进行产品对比快速定位您所需要的产品。
  https://products.espressif.com/#/product-selector?language=zh

联系我们

- 商务问题、技术支持、电路原理图 & PCB 设计审阅、购买样品（线上商店）、成为供应商、意见与建议
  https://espressif.com/zh-hans/contact-us/sales-questions

词汇列表

外设相关词汇

2D-DMA	2D-DMA 控制器
ADC	ADC 控制器
AES	AES 加速器
DMA	DMA 控制器
ECC	ECC 加速器
ECDSA_DS	ECDSA 数字签名外设
eFuse	eFuse 控制器
EMAC	以太网介质访问控制器
ETM	事件任务矩阵
HMAC	HMAC 加速器
I2C	I2C 控制器
I2S	I2S 控制器
I3C	I3C 控制器
ISP	图像信号处理器
LCD_CAM	LCD 与 Camera 控制器
LEDC	LED PWM 控制器
MCPWM	电机控制脉宽调制器
PARLIO	并行 IO 控制器
PCNT	脉冲计数器控制器
PIE	处理器指令拓展
PMS	权限控制
PPA	像素处理加速器
RMT	红外遥控
RNG	随机数发生器
RSA	RSA 加速器
RSA_DS	RSA 数字签名外设
SDHOST	SD/MMC 主机控制器
SHA	SHA 加速器
SPI	SPI 控制器
TIMG	定时器组
TOUCH	触摸传感器
TRACE	RISC-V Trace 编码器
TSENS	温度传感器
TWAI	双线汽车接口
UART	UART 控制器
ULP 协处理器	超低功耗协处理器
USB OTG	USB On-The-Go
USB_SERIAL_JTAG	USB 串口/JTAG 控制器
VAD	语音活动检测
VDMA	VDMA 控制器
WDT	看门狗定时器

XTS_AES	片外存储器加密与解密

寄存器相关缩写

REG	**寄存器**。
SYSREG	**系统寄存器**是一组控制系统复位、存储器、时钟、软件中断、电源管理、时钟门控等的寄存器。
ISO	**隔离**。如果外设或其他芯片组件断电，其输出信号的管脚（若有）将会浮空。ISO 寄存器会隔离上述引脚并令其保持在某个确定值，以使连接到这些引脚的其他非断电外设/设备免受影响。
NMI	**非屏蔽中断**是一种 CPU 指令无法禁用或忽略的硬件中断。出现此类中断说明发生严重错误。
W1TS	添加到寄存器/字段名称中的缩写，表示此类寄存器/字段用于置位名称相似寄存器中的相应字段。例如，寄存器 GPIO_ENABLE_W1TS_REG 用于置位寄存器 GPIO_ENABLE_REG 中的相应字段。
W1TC	与 *W1TS* 相同，但用于清除相应寄存器中的字段。

寄存器的访问类型

TRM 章节 寄存器列表 和 寄存器 详述了寄存器及其字段的访问类型。

常用访问类型及组合如下：

- RO
- WO
- WT
- R/W
- R/W1
- WL
- R/W/SC
- R/W/SS
- R/W/SS/SC
- R/WC/SS
- R/WC/SC
- R/WC/SS/SC
- R/WS/SC
- R/WS/SS
- R/WS/SS/SC
- R/SS/WTC
- R/SC/WTC
- R/SS/SC/WTC
- RF/WF
- R/SS/RC
- varies

下文提供了所有访问类型的具体描述。

R **软件可读**。用户软件可以读取此寄存器/字段；通常与其他访问类型结合使用。

RO **软件只读**。用户软件只可读取此寄存器/字段。

HRO **硬件只读**。仅硬件可以读取此寄存器/字段；用于存储变量参数的默认设置。

W **软件可写**。用户软件可以写入此寄存器/字段；通常与其他访问类型结合使用。

WO **软件只写**。用户软件只可写入此寄存器/字段。

W1 **软件只写一次**。用户软件只可写入一次此寄存器/字段；只允许写入 1，写入 0 无效。

SS **硬件置位**。在指定事件中，硬件自动将 1 写入此寄存器/字段；与一位字段一同使用。

SC **硬件清零**。在指定事件中，硬件自动将 0 写入此寄存器/字段；与一位和多位字段一同使用。

SM **硬件修改**。在指定事件中，硬件自动将指定值写入此寄存器/字段；与多位字段一同使用。

SU **硬件更新**。在指定事件中，硬件自动更新此寄存器/字段；与多位字段一同使用。

RS **软件读置位**。如果用户软件读取此寄存器/字段，硬件会自动写 1。

RC **软件读清零**。如果用户软件读取此寄存器/字段，硬件会自动写 0。

RF **软件读 FIFO**。如果用户软件将新数据写入 FIFO，寄存器/字段会自动读取。

WF **软件写 FIFO**。如果用户软件将新数据写入此寄存器/字段，寄存器/字段会自动通过 APB 总线将数据传递到 FIFO。

WS **软件写置位**。如果用户软件写入此寄存器/字段，硬件会自动置位此寄存器/字段。

W1S **软件写 1 置位**。如果用户软件将 1 写入此寄存器/字段，硬件会自动置位此寄存器/字段。

WOS **软件写 0 置位**。如果用户软件将 0 写入此寄存器/字段，硬件会自动置位此寄存器/字段。

WC **软件写清零**。如果用户软件写入此寄存器/字段，硬件会自动清零此寄存器/字段。

W1C **软件写 1 清零**。如果用户软件将 1 写入此寄存器/字段，硬件会自动清零此寄存器/字段。

WOC **软件写 0 清零**。如果用户软件将 0 写入此寄存器/字段，硬件会自动清零此寄存器/字段。

WT **软件写产生边沿触发信号**。如果用户软件将 1 写入此字段，将会产生边沿触发信号（APB 总线中的脉冲）或清除相应的 WTC 字段（详见 WTC）。

WTC **软件写其他寄存器位清零本寄存器位**。如果用户软件将 1 写入相应的 WT 字段，硬件会自动清除此字段（详见 WT）。

W1T **软件写 1 取反**。如果用户软件将 1 写入此字段，硬件会自动取反相应字段，否则不会取反。

W0T **软件写 0 取反**。如果用户软件将 0 写入此字段，硬件会自动取反相应字段，否则不会取反。

WL **软件仅在锁禁用时写**。如果锁被禁用，用户软件可以写入此寄存器/字段。

varies **访问类型不定**。此寄存器中的不同字段访问类型可能不同。

如何配置寄存器的保留域

概述

寄存器的保留域指的是寄存器中不对用户开放、或者配置为非默认值时会导致不可预测的结果的域。

如何配置保留域

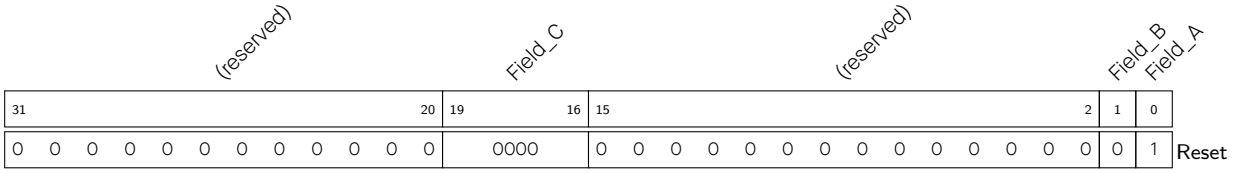
保留域的值不能修改。由于写寄存器时必须整体写，不能只写部分域，因此，在写带有保留域的寄存器时需要特别注意，只能采用以下两种方式：

1. 读取寄存器的值，仅修改需要配置的域，然后将修改后的值以及其他未修改的值一起写回寄存器，这样保留域的值就会保持不变。

   或者

2. 仅修改需要配置的域，然后将保留域的默认值写回寄存器。默认值即寄存器图表中的“Reset”值。例如，寄存器 X 中 Field_A 的默认值为 1。

Register 66.1. 寄存器 X (地址)



假设您要将 寄存器 X 的 Field_A、Field_B 和 Field_C 设置为 0x0、0x1 和 0x2，您可以：

- 使用第 1 种方式，修改这三个域的值，然后把读到的值写回寄存器。假设寄存器读取的值为 0x0000_0003，修改三个域的值后，将值 0x0002_0002 写入寄存器。
- 使用第 2 种方式，修改这三个域的值，然后把保留域的默认值写回寄存器，即把 0x0002_0002 写入寄存器。

中断配置寄存器

大部分外设的内部中断源都有以下配置寄存器：

- **RAW**（原始状态）寄存器：该寄存器指示原始中断状态，每个位对应一个内部中断源。当中断源触发时，其 RAW 位为 1。
- **ENA**（使能）寄存器：该寄存器用于启用或禁用内部中断源，每个位对应一个内部中断源。

  通过操作 ENA 寄存器，可以根据需要屏蔽或取消屏蔽某个内部中断源。当中断源被屏蔽（禁用）时，它不会生成中断信号，但仍可以从 RAW 寄存器中读取其值。
- **ST**（状态）寄存器：该寄存器指示中断源的屏蔽状态，每个位对应一个内部中断源。ST 位为 1 代表 RAW 位和 ENA 位都为 1，即中断源已生成且未被屏蔽。RAW 位和 ENA 位的值为其他组合时，ST 位为 0。

  ENA/RAW/ST 寄存器的配置见表 66.7-4。
- **CLR**（清除）寄存器：CLR 寄存器负责清除内部中断源。写 1 将清除该位对应的中断源。

表 66.7-4. ENA/RAW/ST 寄存器的配置

ENA 位的值	RAW 位的值	ST 位的值
0	忽略	0
1	0	0
	1	1

修订历史

日期	版本	发布说明
2026-08-20	v0.7	新增以下章节： • 章节 4 处理器指令扩展 (PIE) 更新以下章节： • 章节 21 系统寄存器 (SYSREG)：针对最新芯片版本进行全面更新 • 章节 41 H264 编码器：针对最新芯片版本进行全面更新
2026-07-02	v0.6	新增以下章节： • 章节 1 总线架构 • 章节 23 AXI 性能监测器 • 章节 43 MIPI DSI 更新以下章节： • 章节 6 通用 DMA 控制器 (GDMA-AHB, GDMA-AXI)：更新 GDMA-AHB 存储到存储数据传输的描述 • 章节 38 图像信号处理器 (ISP)：更新镜头阴影矫正、gamma 矫正、自动白平衡统计模块的功能描述 • 章节 49 I2S 控制器 (I2S)：更新 APLL 时钟源的最高频率
2025-03-11	v0.5	首次发布



免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，乐鑫不对信息的准确性、真实性做任何保证。

乐鑫不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他乐鑫提案、规格书或样品在他处提到的任何保证。

乐鑫不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

版权归 © 2026 乐鑫信息科技（上海）股份有限公司。保留所有权利。

www.espressif.com