ESP32-C5

技术参考手册 版本 1.1

ESPRESSIF

关于本文档

ESP32-C5 技术参考手册面向使用 ESP32-C5 系列产品进行底层软件开发的人员，介绍了 ESP32-C5 系列产品中内置的硬件模块，包括概述、功能列表、硬件架构、编程指南、寄存器列表等信息。

本文档中的跳转

在本文档中实现跳转，请参考以下建议：

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

发布进度速览

No.	章节	最新进度
	第 I 卷：微处理器和主机	
1	总线架构	已发布
2	高性能处理器	已发布
3	RISC-V 追踪编码器 (TRACE)	已发布
4	低功耗处理器	已发布
5	通用 DMA 控制器 (GDMA)	已发布
	第 II 卷：存储器组织结构	
6	系统和存储器	已发布
7	eFuse 控制器 (EFUSE)	已发布
	第 III 卷：系统组件	
8	GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)	已发布
9	复位和时钟	已发布
10	芯片 Boot 控制	已发布
11	中断矩阵	已发布
12	事件任务矩阵 (ETM)	已发布
13	低功耗管理	已发布
14	系统定时器	已发布
15	定时器组 (TIMG)	已发布
16	看门狗定时器 (WDT)	已发布
17	实时时钟定时器	已发布
18	权限控制 (PMS)	已发布
19	系统寄存器	已发布
20	辅助调试	已发布
21	电源检测器	已发布
	第 IV 卷：加密和安全组件	
22	AES 加速器 (AES)	已发布
23	ECC 加速器 (ECC)	已发布
24	HMAC 加速器 (HMAC)	已发布
25	RSA 加速器 (RSA)	已发布
26	SHA 加速器 (SHA)	已发布
27	RSA 数字签名外设 (RSA_DS)	已发布
28	ECDSA 数字签名外设 (ECDSA_DS)	已发布
29	片外存储器加密与解密 (XTS_AES)	已发布
30	随机数发生器 (RNG)	已发布
31	密钥管理器	已发布
	第 V 卷：通讯接口	
32	UART 控制器 (UART)	已发布
33	SPI 控制器 (SPI)	已发布
34	I2C 控制器 (I2C)	已发布
35	I2S 控制器 (I2S)	已发布
36	脉冲计数控制器 (PCNT)	已发布

No.	章节	最新进度
37	USB 串口/JTAG 控制器	已发布
38	CAN FD	已发布
39	SDIO 从机控制器 (SDIO)	已发布
40	LED PWM 控制器 (LEDC)	已发布
41	电机控制脉宽调制器 (MCPWM)	已发布
42	红外遥控 (RMT)	已发布
43	并行 IO 控制器 (PARLIO)	已发布
44	比特调节器	已发布
第 VI 卷：模拟信号处理		
45	温度传感器	已发布
46	ADC 控制器	已发布
47	模拟电压比较器	已发布

说明：

点击链接或扫描二维码确保您使用的是最新版本的文档：
https://www.espressif.com/documentation/esp32-c5_technical_reference_manual_cn.pdf

目录

I 微处理器和主机 44

1 总线架构 45

2 高性能处理器 48
2.1 概述 48
2.2 特性 48
2.3 术语 49
2.4 地址分布 50
2.5 配置和状态寄存器 (CSRs) 50
2.5.1 寄存器列表 50
2.5.2 寄存器描述 53
2.6 RISC-V 标准 ISA 扩展支持 76
2.6.1 乘法/除法 (M) 扩展 76
2.6.1.1 概述 76
2.6.1.2 功能描述 76
2.6.2 压缩 (C) 扩展 76
2.6.2.1 概述 76
2.6.2.2 功能描述 76
2.6.3 代码体积优化 (Zc) 扩展 77
2.6.3.1 概述 77
2.6.3.2 功能描述 77
2.6.3.3 Zc 指令 77
2.6.3.4 限制 77
2.6.4 原子 (A) 扩展 78
2.6.4.1 概述 78
2.6.4.2 功能描述 78
2.6.4.3 加载保留字 (LR.W) 指令 78
2.6.4.4 条件存入字 (SC.W) 指令 78
2.6.4.5 AMO 指令 79
2.7 存储器映射寄存器 80
2.7.1 概述 80
2.7.2 特性 80
2.7.3 功能描述 80
2.8 中断控制器 81
2.8.1 概述 81
2.8.2 CLIC 81
2.8.2.1 概述 81
2.8.2.2 CLIC CSRs 81
2.8.2.3 中断映射 81
2.8.2.4 CLIC 参数 82
2.8.2.5 中断级别和优先级编码 82

2.8.2.6 CLIC 存储器映射寄存器列表 83
2.8.2.7 CLIC 存储器映射寄存器描述 84
2.8.3 核心本地中断 (CLINT) 87
2.8.3.1 概述 87
2.8.3.2 特性 87
2.8.3.3 软件中断 87
2.8.3.4 定时器计数器和中断 87
2.8.3.5 寄存器列表 87
2.8.3.6 寄存器描述 88
2.9 存储器保护单元 92
2.9.1 标准物理存储器保护 92
2.9.1.1 概述 92
2.9.1.2 特性 92
2.9.1.3 功能描述 92
2.9.1.4 寄存器列表 93
2.9.1.5 寄存器描述 93
2.9.2 自定义物理存储器属性 (PMA) 检查器 96
2.9.2.1 概述 96
2.9.2.2 特性 96
2.9.2.3 功能描述 96
2.9.2.4 寄存器列表 97
2.9.2.5 寄存器 98
2.10 调试和追踪 100
2.10.1 调试 100
2.10.1.1 概述 100
2.10.1.2 功能 101
2.10.1.3 功能描述 101
2.10.1.4 JTAG 控制 101
2.10.1.5 寄存器 102
2.10.1.6 寄存器描述 102
2.10.2 硬件触发器 106
2.10.2.1 概述 106
2.10.2.2 特性 106
2.10.2.3 功能描述 106
2.10.2.4 触发执行流程 107
2.10.2.5 寄存器列表 107
2.10.2.6 寄存器描述 108
2.10.3 追踪 113
2.10.3.1 概述 113
2.10.3.2 特性 113
2.10.3.3 功能描述 113
2.11 性能 114
2.11.1 分支预测 114
2.11.1.1 概述 114
2.11.1.2 特性 114
2.11.1.3 功能描述 114

2.11.2 RAS 115
2.11.2.1 概述 115
2.11.2.2 特性 115
2.11.2.3 功能描述 115
2.11.3 性能配置控制状态寄存器 116
2.12 自定义功能 117
2.12.1 总线错误响应 117
2.12.1.1 概述 117
2.12.1.2 功能描述 117
2.12.1.3 控制状态寄存器 117
2.12.2 专用 IO 119
2.12.2.1 概述 119
2.12.2.2 特性 119
2.12.2.3 功能描述 119
2.12.2.4 寄存器列表 120
2.12.2.5 寄存器 120
2.12.3 运行暂停 (RunStall) 122
2.12.3.1 概述 122
2.12.3.2 功能描述 122
2.12.3.3 寄存器列表 122
2.12.4 调试辅助信息 123
2.12.5 核心锁定 123
2.12.5.1 概述 123
2.12.5.2 功能描述 123
2.12.5.3 寄存器列表 123
2.12.5.4 寄存器描述 123

3 RISC-V 追踪编码器 (TRACE) 124
3.1 术语 124
3.2 介绍 124
3.3 特性 125
3.4 架构概览 127
3.5 功能描述 127
3.5.1 同步 127
3.5.2 地址模式 127
3.5.3 可选边带信号 128
3.5.4 过滤 128
3.5.5 锚定 129
3.5.6 写存储器模式 129
3.5.7 自动重启 129
3.6 编码器输出数据包 129
3.6.1 头部 129
3.6.2 索引 130
3.6.3 有效载荷 130
3.6.3.1 格式 3 130
3.6.3.2 格式 2 132

3.6.3.3 格式 1 132
3.7 中断 134
3.8 编程流程 134
3.8.1 配置编码器选项 134
3.8.2 配置过滤器 135
3.8.3 使能编码器 136
3.8.4 关闭编码器 137
3.8.5 通知 137
3.8.6 解码数据包 138
3.8.7 AHB 配置 138
3.8.8 软件保留 138
3.9 寄存器列表 139
3.10 寄存器 140

4 低功耗处理器 151
4.1 特性 151
4.2 配置与状态寄存器 (CSR) 152
4.2.1 寄存器列表 152
4.2.2 寄存器 153
4.3 中断和异常 159
4.3.1 中断 159
4.3.2 中断处理 159
4.3.3 异常 160
4.4 调试 160
4.4.1 特性 160
4.4.2 功能描述 161
4.4.3 寄存器列表 161
4.4.4 寄存器 161
4.5 硬件触发器 163
4.5.1 特性 163
4.5.2 功能描述 163
4.5.3 触发执行流程 164
4.5.4 寄存器列表 164
4.5.5 寄存器 164
4.6 性能计数器 167
4.7 系统访问 168
4.7.1 存储器 168
4.7.2 外设 168
4.8 事件任务矩阵功能 168
4.9 睡眠和唤醒流程 168
4.9.1 特性 168
4.9.2 流程 169
4.9.3 唤醒源 170
4.10 寄存器列表 170
4.11 寄存器 171

5 通用 DMA 控制器 (GDMA) 172
5.1 概述 172
5.2 特性 172
5.3 架构 173
5.4 功能描述 173
5.4.1 链表 174
5.4.2 外设到存储器及存储器到外设的数据传输 175
5.4.3 存储器到存储器数据传输 175
5.4.4 启动 GDMA 176
5.4.5 读链表 176
5.4.6 数据传输结束标志 177
5.4.7 访问存储器 177
5.4.8 仲裁 177
5.4.9 总线错误响应信息记录 178
5.4.10 自动时钟门控 178
5.5 事件任务矩阵功能 179
5.6 中断 180
5.7 编程流程 181
5.7.1 GDMA TX 通道配置流程 181
5.7.2 GDMA RX 通道配置流程 181
5.7.3 GDMA 存储器到存储器配置流程 181
5.7.4 通道优先级和权重配置流程 182
5.8 寄存器列表 183
5.9 寄存器 189

II 存储器组织结构 221

6 系统和存储器 222
6.1 概述 222
6.2 主要特性 222
6.3 功能描述 223
6.3.1 地址映射 223
6.3.2 内部存储器 225
6.3.2.1 ROM 225
6.3.2.2 HP SRAM 225
6.3.2.3 LP SRAM 226
6.3.3 外部存储器 226
6.3.3.1 外部存储器地址映射 226
6.3.3.2 Cache 226
6.3.3.3 Cache 操作 227
6.3.4 GDMA 地址空间 228
6.3.5 模块/外设地址空间映射 229

7 eFuse 控制器 (EFUSE) 232
7.1 概述 232
7.2 主要特性 232

7.3 功能描述 232
7.3.1 结构 232
7.3.1.1 EFUSE_WR_DIS 238
7.3.1.2 EFUSE_RD_DIS 238
7.3.1.3 数据存储方式 238
7.3.2 烧写参数 240
7.3.3 用户读取参数 241
7.3.4 eFuse VDDQ 时序 242
7.3.5 硬件模块使用参数 243
7.4 中断 243
7.5 寄存器列表 244
7.6 寄存器 248

III 系统组件 277

8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX) 278
8.1 概述 278
8.2 主要特性 278
8.2.1 HP GPIO 交换矩阵和 HP IO MUX 278
8.2.2 LP GPIO 交换矩阵和 LP IO MUX 279
8.3 结构概览 279
8.4 通过 GPIO 交换矩阵的外设输入 281
8.4.1 概述 281
8.4.2 信号同步 282
8.4.3 GPIO 滤波器 282
8.4.4 毛刺滤波器 283
8.4.5 简单 GPIO 输入 283
8.4.6 GPIO 唤醒 283
8.4.6.1 HP GPIO 唤醒 283
8.4.6.2 LP GPIO 唤醒 283
8.4.7 编程流程 284
8.4.7.1 HP GPIO 交换矩阵 284
8.4.7.2 LP GPIO 交换矩阵 285
8.5 通过 GPIO 交换矩阵的外设输出 286
8.5.1 概述 286
8.5.2 简单 GPIO 输出 286
8.5.3 Sigma Delta 调制输出 (SDM) 286
8.5.3.1 功能描述 286
8.5.3.2 配置方法 287
8.5.4 编程流程 287
8.5.4.1 HP GPIO 交换矩阵 287
8.5.4.2 LP GPIO 交换矩阵 288
8.6 IO MUX 的直接输入输出功能 288
8.6.1 概述 288
8.6.2 功能描述 289
8.6.2.1 HP IO MUX 289

8.6.2.2 LP IO MUX 289
8.7 模拟功能描述 289
8.7.1 概述 289
8.7.2 模拟功能描述 289
8.8 Light-sleep 模式管脚功能 290
8.9 GPIO 管脚的 Hold 特性 290
8.10 GPIO 管脚的迟滞特性 291
8.11 GPIO 管脚供电和电源管理 292
8.11.1 GPIO 管脚供电 292
8.11.2 电源管理 292
8.12 HP 外设信号列表 292
8.13 HP IO MUX 管脚功能列表 298
8.14 LP IO MUX 管脚功能列表 300
8.15 GPIO 管脚模拟功能列表 301
8.16 事件任务矩阵 (ETM) 功能 301
8.17 中断 302
8.18 寄存器列表 303
8.18.1 HP GPIO 交换矩阵寄存器列表 303
8.18.2 HP IO MUX 寄存器列表 307
8.18.3 GPIO EXT 寄存器列表 307
8.18.4 LP GPIO 交换矩阵寄存器列表 309
8.18.5 LP IO MUX 寄存器列表 309
8.19 寄存器 310
8.19.1 HP GPIO 交换矩阵寄存器 310
8.19.2 HP IO MUX 寄存器 326
8.19.3 GPIO EXT 寄存器 329
8.19.4 LP GPIO 矩阵寄存器 349
8.19.5 LP IO MUX 寄存器 357

9 复位和时钟 361
9.1 复位 361
9.1.1 概述 361
9.1.2 结构图 361
9.1.3 特性 361
9.1.4 功能描述 362
9.1.5 外设复位 363
9.2 时钟 363
9.2.1 概述 363
9.2.2 结构图 364
9.2.3 特性 364
9.2.4 功能描述 365
9.2.4.1 高性能系统时钟 365
9.2.4.2 低功耗系统时钟 365
9.2.4.3 外设时钟 366
9.2.4.4 PMU 控制高性能系统时钟门控 370
9.3 配置流程 372

9.3.1 高性能系统时钟配置 372
9.3.2 低功耗系统时钟配置 373
9.3.3 外设时钟复位配置 373
9.4 寄存器列表 375
9.4.1 PCR 模块寄存器列表 375
9.4.2 低功耗系统时钟寄存器列表 377
9.5 寄存器 378
9.5.1 PCR 模块寄存器 378
9.5.2 低功耗系统时钟复位寄存器 431

10 芯片 Boot 控制 442
10.1 概述 442
10.2 功能描述 442
10.2.1 默认配置 442
10.2.2 芯片启动模式控制 443
10.2.3 SDIO 输入采样沿和输出驱动沿控制 445
10.2.4 ROM 代码日志打印控制 445
10.2.5 JTAG 信号源控制 446

11 中断矩阵 447
11.1 概述 447
11.2 中断术语 447
11.2.1 中断 447
11.2.2 中断信号/中断源 447
11.2.3 ESP32-C5 中断流 447
11.3 特性 448
11.4 结构概览 448
11.5 功能描述 448
11.5.1 外部中断源 448
11.5.2 HP CPU 中断 453
11.5.3 分配外部中断源至 HP CPU 外部中断 453
11.5.3.1 分配一个外部中断源 SOURCE 至 HP CPU 外部中断 453
11.5.3.2 分配多个外部中断源 SOURCE 至 HP CPU 外部中断 453
11.5.3.3 关闭 HP CPU 外部中断源 SOURCE 453
11.5.4 委托中断 453
11.5.5 查询外部中断源 SOURCE 当前的中断状态 454
11.6 寄存器列表 455
11.6.1 中断矩阵寄存器列表 455
11.6.2 软件中断寄存器列表 458
11.7 寄存器 459
11.7.1 中断矩阵寄存器 459
11.7.2 软件中断寄存器 465

12 事件任务矩阵 (ETM) 467
12.1 概述 467
12.2 特性 467

12.3 功能描述 467
12.3.1 架构 467
12.3.2 事件 468
12.3.3 任务 471
12.3.4 事件任务触发状态 475
12.3.5 时序考虑因素 475
12.3.6 通道控制 476
12.4 寄存器列表 477
12.5 寄存器 481

13 低功耗管理 571
13.1 概述 571
13.2 术语 571
13.3 主要特性 571
13.4 功能描述 572
13.4.1 电源结构 572
13.4.1.1 调压器 573
13.4.1.2 数字电源域 573
13.4.1.3 模拟电源域 574
13.4.2 PMU 574
13.4.2.1 PMU 主状态机 575
13.4.2.2 睡眠与唤醒控制器 576
13.4.2.3 模拟电源控制器 577
13.4.2.4 数字电源控制器 578
13.4.2.5 时钟控制器 579
13.4.2.6 数据备份控制器 580
13.4.2.7 系统控制器 580
13.5 功耗模式 581
13.6 RTC 启动 581
13.7 事件任务矩阵功能 582
13.8 中断 583
13.9 寄存器列表 585
13.9.1 PMU 寄存器列表 585
13.9.2 Always-on 寄存器列表 587
13.10 寄存器 587
13.10.1 PMU 寄存器 587
13.10.2 Always-on 寄存器 627

14 系统定时器 629
14.1 概述 629
14.2 主要特性 629
14.3 系统定时器架构 630
14.4 时钟源选择 630
14.5 功能描述 631
14.5.1 计数器 631
14.5.2 比较器和报警 632

14.5.3 事件任务矩阵 633
14.5.4 同步操作 633
14.6 中断 633
14.7 编程示例 634
14.7.1 读取当前计数器的值 634
14.7.2 在单次报警模式下配置一次性报警 634
14.7.3 在周期报警模式下配置周期性报警 635
14.7.4 唤醒后时间补偿 635
14.8 寄存器列表 636
14.9 寄存器 638

15 定时器组 (TIMG) 653
15.1 概述 653
15.2 主要特性 653
15.3 结构概览 654
15.4 功能描述 654
15.4.1 16 位预分频器与时钟选择器 654
15.4.2 54 位时基计数器 655
15.4.3 警报生成 655
15.4.4 定时器重新加载 656
15.4.5 TIMG0 定时器组慢速时钟频率计算 656
15.5 事件任务矩阵功能 656
15.6 中断 658
15.7 配置流程 658
15.7.1 定时器用作简单时钟 658
15.7.2 定时器用于单次报警 658
15.7.3 通过 APB 设置定时器用于周期性报警 659
15.7.4 通过 ETM 设置定时器用于周期性报警 659
15.7.5 慢速时钟频率计算 660
15.8 寄存器列表 662
15.9 寄存器 663

16 看门狗定时器 (WDT) 675
16.1 概述 675
16.2 数字看门狗定时器 676
16.2.1 主要特性 676
16.2.2 功能描述 677
16.2.2.1 时钟源与 32 位计数器 677
16.2.2.2 阶段与超时动作 678
16.2.2.3 写保护 678
16.2.2.4 Flash 引导保护 679
16.3 超级看门狗定时器 679
16.3.1 主要特性 679
16.3.2 SWD 控制器 679
16.3.2.1 结构 679
16.3.2.2 工作流程 680

16.4 中断 680
16.5 寄存器列表 680
16.6 寄存器 681

17 实时时钟定时器 689
17.1 概述 689
17.2 主要特性 689
17.3 功能描述 689
17.4 事件任务矩阵功能 691
17.5 中断 691
17.6 寄存器列表 692
17.7 寄存器 693

18 权限控制 (PMS) 701
18.1 概述 701
18.2 APM 简介 702
18.3 主要特性 703
18.4 TEE 与 REE 术语 704
18.5 功能描述 704
18.5.1 TEE 控制器功能描述 704
18.5.2 APM 控制器功能描述 705
18.5.2.1 结构概览 705
18.5.2.2 SYS_APM 控制器 706
18.5.2.2.1 地址范围 707
18.5.2.2.2 地址范围的访问权限 707
18.5.2.3 PERI_APM 控制器 708
18.6 非法访问与中断 709
18.6.1 SYS_APM 控制器 709
18.6.2 PERI_APM 控制器 710
18.7 配置流程 710
18.8 寄存器列表 711
18.8.1 HP_APM_REG 711
18.8.2 LP_APM_REG 713
18.8.3 LP_APM0_REG 714
18.8.4 CPU_APM_REG 715
18.8.5 TEE_REG 716
18.8.6 LP_TEE_REG 718
18.9 寄存器 720
18.9.1 HP_APM_REG 720
18.9.2 LP_APM_REG 731
18.9.3 LP_APM0_REG 738
18.9.4 CPU_APM_REG 742
18.9.5 TEE_REG 750
18.9.6 LP_TEE_REG 752

19 系统寄存器 756

19.1 概述 756
19.2 主要特性 756
19.3 功能描述 756
19.3.1 外部存储器加密/解密配置 756
19.3.2 防 DPA 攻击安全控制 756
19.3.3 HP/LP CPU 调试控制 757
19.3.4 总线超时保护 757
19.3.4.1 CPU 外设超时保护寄存器 757
19.3.4.2 HP 外设超时保护寄存器 757
19.3.4.3 LP 外设超时保护寄存器 758
19.4 寄存器列表 759
19.5 寄存器 760

20 辅助调试 769
20.1 概述 769
20.2 主要特性 769
20.3 功能描述 769
20.3.1 区域读写监测 769
20.3.2 栈指针监测 770
20.3.3 PC 记录 770
20.3.4 CPU/DMA 总线访问记录 770
20.4 中断 770
20.5 配置流程 771
20.5.1 区域监测和栈指针监测配置 771
20.5.2 PC 记录配置 772
20.5.3 CPU/DMA 总线访问记录配置 772
20.5.3.1 总线访问记录 1 配置 772
20.5.3.2 总线访问记录 2 配置 776
20.6 寄存器列表 779
20.6.1 总线记录配置寄存器列表 779
20.6.2 其他寄存器列表 780
20.7 寄存器 782
20.7.1 总线记录配置寄存器 782
20.7.2 其他寄存器 791

21 电源检测器 805
21.1 概述 805
21.2 主要特性 805
21.3 功能描述 805
21.3.1 架构图 805
21.3.2 欠压检测器 806
21.3.3 电压毛刺检测器 807
21.4 中断 808
21.5 寄存器列表 810
21.6 寄存器 811

IV 加密和安全组件 818

22 AES 加速器 (AES) 819
22.1 概述 819
22.2 主要特性 819
22.3 时钟和复位 819
22.4 工作模式简介 819
22.5 Typical AES 工作模式 821
22.5.1 密钥、明文、密文 821
22.5.2 字节序 821
22.5.3 Typical AES 工作模式的流程 824
22.6 DMA-AES 工作模式 824
22.6.1 密钥、明文、密文 825
22.6.2 字节序 825
22.6.3 标准增量函数 826
22.6.4 块个数 826
22.6.5 初始向量 826
22.6.6 DMA-AES 工作模式的流程 826
22.7 抗攻击伪轮函数 827
22.8 存储器列表 828
22.9 寄存器列表 829
22.10 寄存器 830

23 ECC 加速器 (ECC) 836
23.1 概述 836
23.2 主要特性 836
23.3 ECC 背景知识 836
23.3.1 椭圆曲线与曲线上的点 836
23.3.2 仿射坐标系与 Jacobian 坐标系 836
23.3.3 内存块 837
23.3.4 数据与数据块 837
23.3.5 数据存储 838
23.3.6 数据读取 838
23.3.7 标准运算与 Jacobian 运算 838
23.4 功能描述 838
23.4.1 曲线模式 838
23.4.2 工作模式 839
23.4.2.1 标准点乘模式 839
23.4.2.2 标准点验证模式 840
23.4.2.3 标准点验证 + 标准点乘模式 840
23.4.2.4 Jacobian 点乘模式 840
23.4.2.5 点加模式 840
23.4.2.6 Jacobian 点验证模式 841
23.4.2.7 标准点验证 + Jacobian 点乘模式 841
23.4.2.8 模加模式 841
23.4.2.9 模减模式 842

23.4.2.10 模乘模式 842
23.4.2.11 模除模式 842
23.4.3 提升抗攻击性能 843
23.4.4 时钟 843
23.4.5 复位 844
23.5 中断 844
23.6 配置流程 845
23.7 寄存器列表 846
23.8 寄存器 847

24 HMAC 加速器 (HMAC) 852
24.1 概述 852
24.2 主要特性 852
24.3 功能描述 852
24.3.1 上行模式 852
24.3.2 下行模式 - JTAG 启动功能 853
24.3.3 下行模式 - RSA 数字签名外设密钥派生功能 853
24.4 HMAC eFuse 配置 853
24.5 配置流程 855
24.5.1 启动 HMAC 模块 855
24.5.2 配置 HMAC 密钥和密钥功能 855
24.5.3 下行模式流程 855
24.5.4 上行模式流程 855
24.6 HMAC 算法细节 856
24.6.1 附加填充位 856
24.6.2 HMAC 算法结构 857
24.7 寄存器列表 859
24.8 寄存器 860

25 RSA 加速器 (RSA) 866
25.1 概述 866
25.2 主要特性 866
25.3 功能描述 866
25.3.1 大数模幂运算 867
25.3.2 大数模乘运算 868
25.3.3 大数乘法运算 868
25.3.4 控制加速 869
25.4 中断 870
25.5 存储器列表 872
25.6 寄存器列表 872
25.7 寄存器 873

26 SHA 加速器 (SHA) 878
26.1 概述 878
26.2 主要特性 878
26.3 工作模式简介 878

26.4 功能描述 879
26.4.1 信息预处理 879
26.4.1.1 附加填充比特 879
26.4.1.2 信息解析 879
26.4.1.3 哈希初始值 (Initial Hash Value) 880
26.4.2 哈希运算流程 880
26.4.2.1 Typical SHA 模式下的运算流程 880
26.4.2.2 DMA-SHA 模式下的运算流程 881
26.4.3 信息摘要存储 882
26.4.4 中断 882
26.5 寄存器列表 883
26.6 寄存器 884

27 RSA 数字签名外设 (RSA_DS) 888
27.1 概述 888
27.2 主要特性 888
27.3 功能描述 888
27.3.1 概述 888
27.3.2 私钥运算子 889
27.3.3 软件准备工作 889
27.3.4 硬件工作流程 890
27.3.5 软件工作流程 891
27.4 存储器列表 893
27.5 寄存器列表 894
27.6 寄存器 895

28 ECDSA 数字签名外设 (ECDSA_DS) 898
28.1 概述 898
28.2 主要特性 898
28.3 ECDSA 背景知识 898
28.3.1 域参数 898
28.3.2 密钥生成 899
28.3.3 签名生成 899
28.3.4 签名验证 899
28.4 功能描述 900
28.4.1 ECDSA_DS 工作模式 900
28.4.2 数据和数据块 901
28.4.2.1 数据写入 901
28.4.2.2 数据读取 902
28.4.2.3 消息填充 902
28.4.2.4 解析消息 902
28.4.3 安全功能 902
28.4.3.1 抗攻击性能 903
28.4.3.2 基于状态的寄存器访问控制 903
28.4.3.3 硬件占用 903
28.5 配置流程 903

28.5.1 ECDSA_DS 流程 903
28.5.1.1 IDLE 阶段 904
28.5.1.2 PREP 阶段 905
28.5.1.3 LOAD 阶段 905
28.5.1.4 PROC 阶段 905
28.5.1.5 GAIN 阶段 905
28.5.1.6 POST 阶段 906
28.5.1.7 ECDSA_DS SHA 接口 906
28.5.2 时钟 906
28.5.3 复位 907
28.6 中断 908
28.7 存储器块 909
28.8 寄存器列表 910
28.9 寄存器 911

29 片外存储器加密与解密 (XTS_AES) 918
29.1 概述 918
29.2 主要特性 918
29.3 模块结构 918
29.4 功能描述 919
29.4.1 XTS 算法 919
29.4.2 密钥 920
29.4.3 目标空间 920
29.4.4 数据写入 921
29.4.5 手动加密模块 921
29.4.6 自动加密模块 922
29.4.7 自动解密模块 922
29.5 软件流程 923
29.6 抗 DPA 攻击 923
29.6.1 时钟抗 DPA 攻击 923
29.6.2 伪轮次抗 DPA 功能 924
29.7 寄存器列表 926
29.8 寄存器 927

30 随机数发生器 (RNG) 933
30.1 概述 933
30.2 主要特性 933
30.3 功能描述 933
30.4 编程指南 934
30.5 寄存器列表 936
30.6 寄存器 937

31 密钥管理器 938
31.1 概述 938
31.2 背景知识 938
31.3 术语 938

31.4 主要特性 940
31.5 结构概览 940
31.6 功能描述 941
31.6.1 HUK 生成器 941
31.6.1.1 HUK 生成过程 941
31.6.1.2 HUK 状态检查 942
31.6.2 密钥管理器 943
31.6.2.1 随机部署模式 943
31.6.2.2 AES 部署方式 943
31.6.2.3 ECDH0 部署模式 945
31.6.2.4 ECDH1 部署模式 946
31.6.2.5 私钥恢复模式 948
31.6.2.6 key_info 导出模式 948
31.7 编程流程 948
31.7.1 HUK 生成器 948
31.7.1.1 IDLE 阶段 949
31.7.1.2 PREP 阶段 949
31.7.1.3 LOAD 阶段 949
31.7.1.4 PROC 阶段 950
31.7.1.5 GAIN 阶段 950
31.7.1.6 POST 阶段 950
31.7.2 密钥管理器 950
31.7.2.1 IDLE 阶段 951
31.7.2.2 PREP 阶段 953
31.7.2.3 LOAD 阶段 954
31.7.2.4 PROC 阶段 954
31.7.2.5 GAIN 阶段 954
31.7.2.6 POST 阶段 955
31.8 编程示例 955
31.8.1 芯片私钥部署阶段的软件流程示例 955
31.8.2 芯片正常启动阶段的软件流程示例 956
31.9 中断 956
31.10 内存块 957
31.11 寄存器列表 957
31.11.1 HUK 寄存器列表 957
31.11.2 密钥管理器寄存器列表 958
31.12 寄存器 959
31.12.1 HUK 寄存器 959
31.12.2 密钥管理器寄存器 963

V 通讯接口 972

32 UART 控制器 (UART) 973
32.1 概述 973
32.2 主要特性 973
32.3 UART 架构 974

32.4 功能描述 975
32.4.1 时钟与复位 975
32.4.2 UART FIFO 975
32.4.3 波特率产生与检测 976
32.4.3.1 波特率产生 976
32.4.3.2 波特率检测 976
32.4.4 UART 数据帧 977
32.4.5 AT_CMD 字符格式 978
32.4.6 RS485 978
32.4.6.1 驱动控制 979
32.4.6.2 转换延时 979
32.4.6.3 总线侦听 979
32.4.7 IrDA 979
32.4.8 唤醒 980
32.4.9 流控 981
32.4.9.1 硬件流控 981
32.4.9.2 软件流控 982
32.4.10 GDMA 模式 983
32.5 中断 984
32.6 编程流程 985
32.6.1 寄存器类型 985
32.6.2 具体步骤 986
32.6.2.1 UART*n* 模块初始化 986
32.6.2.2 UART*n* 通信配置 986
32.6.2.3 启动 UART*n* 987
32.7 寄存器列表 988
32.7.1 UART 寄存器列表 988
32.7.2 LP UART 寄存器列表 989
32.7.3 UHCI 寄存器列表 990
32.8 寄存器 992
32.8.1 UART 寄存器 992
32.8.2 LP UART 寄存器 1014
32.8.3 UHCI 寄存器 1033

33 SPI 控制器 (SPI) 1052
33.1 概述 1052
33.2 术语 1052
33.3 特性 1053
33.4 架构概览 1054
33.5 功能描述 1054
33.5.1 数据模式 1054
33.5.2 FSPI 总线信号描述 1055
33.5.3 数据位读/写顺序控制 1057
33.5.4 非字节对齐传输 1059
33.5.5 传输类型 1059
33.5.6 CPU 控制的数据传输 1059

33.5.6.1 CPU 控制的主机传输 1060
33.5.6.2 CPU 控制的从机传输 1062
33.5.7 DMA 控制的数据传输 1062
33.5.7.1 GDMA 配置 1063
33.5.7.2 GDMA TX/RX Buffer 长度控制 1064
33.5.8 GP-SPI2 用作主机和从机时的数据流控制 1064
33.5.8.1 GP-SPI2 功能块图 1064
33.5.8.2 GP-SPI2 用作主机时的数据流控制 1065
33.5.8.3 GP-SPI2 用作从机时的数据流控制 1066
33.5.9 GP-SPI2 用作主机 1067
33.5.9.1 主机状态机 1067
33.5.9.2 状态控制和位模式控制寄存器 1069
33.5.9.3 主机全双工通信（仅支持 1-bit 模式） 1072
33.5.9.4 主机半双工通信（支持 1/2/4-bit 模式） 1073
33.5.9.5 DMA 控制的分段配置传输 1074
33.5.10 GP-SPI2 用作从机 1077
33.5.10.1 可配置的通信格式 1078
33.5.10.2 半双工通信支持的 CMD 值 1079
33.5.10.3 从机单次传输和从机连读传输 1081
33.5.10.4 配置从机单次传输 1081
33.5.10.5 配置半双工通信下从机连续传输 1082
33.5.10.6 配置全双工通信下从机连续传输 1082
33.6 CS 建立时间和保持时间控制 1083
33.7 GP-SPI2 时钟控制 1084
33.7.1 时钟相位和极性 1085
33.7.2 主机时钟控制 1086
33.7.3 从机时钟控制 1087
33.8 GP-SPI2 时序补偿 1087
33.9 中断 1089
33.9.1 中断描述 1089
33.9.2 GP-SPI2 用作主机和从机时分别用到的中断 1090
33.10 寄存器列表 1092
33.11 寄存器 1093

34 I2C 控制器 (I2C) 1132
34.1 概述 1132
34.2 主要特性 1132
34.3 结构概览 1133
34.4 功能描述 1135
34.4.1 时钟配置 1135
34.4.2 滤除 SCL 和 SDA 噪声 1135
34.4.3 SCL 时钟拉伸 1136
34.4.4 SCL 空闲时产生 SCL 脉冲 1136
34.4.5 同步 1136
34.4.6 漏级开路输出 1137
34.4.7 时序参数配置 1138

34.4.8 超时控制 1139
34.4.9 指令配置 1140
34.4.10 TX/RX RAM 数据存储 1141
34.4.11 数据转换 1142
34.4.12 寻址模式 1142
34.4.13 10 位寻址的读写标志位检查 1142
34.4.14 启动控制器 1142
34.5 LP_I2C 和 I2C 的功能差异 1143
34.6 中断 1143
34.7 配置流程 1144
34.7.1 I2C 主机写入从机，7 位寻址，单次命令序列 1144
34.7.1.1 场景介绍 1145
34.7.1.2 配置示例 1145
34.7.2 I2C 主机写入从机，10 位寻址，单次命令序列 1146
34.7.2.1 场景介绍 1146
34.7.2.2 配置示例 1147
34.7.3 I2C 主机写入从机，7 位双地址寻址，单次命令序列 1147
34.7.3.1 场景介绍 1148
34.7.3.2 配置示例 1148
34.7.4 I2C 主机写入从机，7 位寻址，多次命令序列 1149
34.7.4.1 场景介绍 1150
34.7.4.2 配置示例 1151
34.7.5 I2C 主机读取从机，7 位寻址，单次命令序列 1152
34.7.5.1 场景介绍 1152
34.7.5.2 配置示例 1152
34.7.6 I2C 主机读取从机，10 位寻址，单次命令序列 1153
34.7.6.1 场景介绍 1154
34.7.6.2 配置示例 1154
34.7.7 I2C 主机读取从机，7 位双寻址，单次命令序列 1155
34.7.7.1 场景介绍 1156
34.7.7.2 配置示例 1156
34.7.8 I2C 主机读取从机，7 位寻址，多次命令序列 1157
34.7.8.1 场景介绍 1158
34.7.8.2 配置示例 1159
34.8 寄存器列表 1161
34.8.1 I2C 寄存器列表 1161
34.8.2 LP_I2C 寄存器列表 1162
34.9 寄存器 1164
34.9.1 I2C 寄存器 1164
34.9.2 LP_I2C 寄存器 1186

35 I2S 控制器 (I2S) 1207
35.1 概述 1207
35.2 术语 1207
35.3 特性 1208
35.4 系统架构 1209

35.5 I2S 模块支持的音频协议 1210
35.5.1 TDM Philips 标准模式 1211
35.5.2 TDM MSB 对齐标准模式 1211
35.5.3 TDM PCM 标准模式 1212
35.5.4 PDM 标准模式 1212
35.6 I2S TX/RX 模块时钟 1213
35.7 I2S 模块复位 1215
35.8 I2S 主/从机模式 1215
35.8.1 主/从机发送模式 1215
35.8.2 主/从机接收模式 1216
35.9 发送数据 1216
35.9.1 数据格式控制 1216
35.9.1.1 通道有效数据位宽 1217
35.9.1.2 通道有效数据字节序 1217
35.9.1.3 A 律/μ 律压缩/解压缩 1217
35.9.1.4 通道发送数据位宽 1218
35.9.1.5 通道数据比特顺序 1218
35.9.2 通道模式控制 1219
35.9.2.1 TDM 输出模式 1219
35.9.2.2 PDM 输出模式 1220
35.9.3 同步计数器 1222
35.10 接收数据 1223
35.10.1 通道模式控制 1223
35.10.1.1 TDM 输入模式 1223
35.10.1.2 PDM 输入模式 1224
35.10.2 数据格式控制 1224
35.10.2.1 通道数据比特顺序 1224
35.10.2.2 通道储存数据位宽 1224
35.10.2.3 通道接收数据位宽 1225
35.10.2.4 通道储存数据字节序 1225
35.10.2.5 A 律/μ 律压缩/解压缩 1225
35.11 事件任务矩阵功能 1225
35.12 I2S 中断 1226
35.13 软件配置流程 1227
35.13.1 软件配置 I2S 发送流程 1227
35.13.2 软件配置 I2S 接收流程 1227
35.14 寄存器列表 1229
35.15 寄存器 1230

36 脉冲计数控制器 (PCNT) 1248
36.1 主要特性 1248
36.2 功能描述 1250
36.3 应用实例 1253
36.3.1 通道 0 独自递增计数 1253
36.3.2 通道 0 独自递减计数 1253
36.3.3 通道 0 和通道 1 同时递增计数 1254

36.4 寄存器列表 1254
36.5 寄存器 1256

37 USB 串口/JTAG 控制器 1264
37.1 概述 1264
37.2 主要特性 1264
37.3 功能描述 1266
37.3.1 CDC-ACM USB 接口描述 1266
37.3.2 CDC-ACM 固件接口描述 1267
37.3.3 USB-JTAG 接口：JTAG 命令处理器 1268
37.3.4 USB-JTAG 接口：CMD_REP 使用示例 1269
37.3.5 USB-JTAG 接口：响应捕捉单元 1269
37.3.6 USB-JTAG 接口：控制传输请求 1270
37.4 中断 1270
37.5 配置流程 1271
37.6 寄存器列表 1273
37.7 寄存器 1275

38 CAN FD 1301
38.1 概述 1301
38.2 主要特性 1301
38.3 功能描述 1301
38.3.1 时钟 1301
38.3.2 复位 1302
38.3.3 时间基准 1302
38.3.4 工作模式 1303
38.3.5 初始化顺序 1304
38.3.6 反初始化顺序 1304
38.3.7 CAN 总线配置 1304
38.3.7.1 位速率 1304
38.3.7.2 发送器延迟 1305
38.3.7.3 二次采样点 1305
38.3.7.4 CAN FD 支持 1306
38.3.7.5 协议异常处理 1306
38.3.7.6 实现类型 1307
38.3.7.7 最小位时间/最大位速率 1307
38.3.8 CAN 帧发送 1308
38.3.8.1 TX buffer 选择 1309
38.3.8.2 定时发送模式 1309
38.3.8.3 发送 CAN 帧的类型 1310
38.3.8.4 重发限制 1310
38.3.8.5 中止 1311
38.3.8.6 TX buffer 离线行为 1311
38.3.9 CAN 帧接收 1311
38.3.9.1 帧数量 1312
38.3.9.2 RX buffer 内存 1312

38.3.9.3 RX buffer 状态 1312
38.3.9.4 溢出 1313
38.3.9.5 刷新 1313
38.3.9.6 不一致保护 1313
38.3.9.7 时间戳 1313
38.3.9.8 帧过滤 1313
38.3.9.9 屏蔽过滤器 1314
38.3.9.10 范围过滤器 1314
38.3.10 故障限制 1314
38.3.11 容错 1315
38.3.11.1 RX buffer RAM 的奇偶校验保护 1315
38.3.11.2 TX buffer RAM 的奇偶校验保护 1315
38.3.11.3 TX buffer 备份模式 1316
38.3.12 特殊模式 1317
38.3.12.1 回环模式 1317
38.3.12.2 自检模式 1318
38.3.12.3 禁止确认模式 1318
38.3.12.4 总线监控模式 1318
38.3.12.5 受限操作模式 1318
38.3.13 其他功能 1318
38.3.13.1 错误代码捕获 1318
38.3.13.2 仲裁丢失捕获 1318
38.3.13.3 流量计数器 1319
38.3.13.4 调试寄存器 1319
38.4 中断 1319
38.5 配置示例 1320
38.5.1 500 kbit/4 Mbit 示例 1320
38.5.1.1 CAN 帧发送 1320
38.5.1.1.1 示例代码 1320
38.5.1.2 CAN 帧接收 1321
38.5.1.2.1 示例代码 - 自动模式下的帧接收（32 位访问方式） 1321
38.6 寄存器列表 1322
38.7 寄存器 1324

39 SDIO 从机控制器 (SDIO) 1362
39.1 概述 1362
39.2 主要特性 1362
39.3 结构概览 1362
39.4 协议标准 1363
39.5 功能描述 1363
39.5.1 物理总线 1363
39.5.2 支持命令 1364
39.5.3 I/O function 0 地址空间 1364
39.5.4 I/O function 1/2 地址空间 1366
39.5.4.1 访问 SLC HOST 寄存器空间 1367
39.5.4.2 使用递增地址传输包 1367

39.5.4.3 使用固定地址传输包 1367
39.5.5 DMA 1367
39.5.5.1 链表 1368
39.5.5.2 链表回写 1369
39.5.5.3 数据填充与丢弃 1370
39.5.6 SDIO 总线时序 1371
39.6 中断 1372
39.6.1 主机侧中断 1372
39.6.2 从机侧中断 1372
39.7 包的发送和接收流程 1372
39.7.1 从机向主机发送包 1373
39.7.2 从机从主机接收包 1374
39.8 寄存器列表 1377
39.8.1 HINF 寄存器列表 1377
39.8.2 SLC 寄存器列表 1377
39.8.3 SLC Host 寄存器列表 1378
39.9 寄存器 1379
39.9.1 HINF 寄存器 1379
39.9.2 SLC 寄存器 1384
39.9.3 SLC Host 寄存器 1406

40 LED PWM 控制器 (LEDC) 1416
40.1 概述 1416
40.2 主要特性 1416
40.3 结构概览 1416
40.4 功能描述 1417
40.4.1 定时器 1417
40.4.1.1 时钟源 1418
40.4.1.2 时钟分频器配置 1418
40.4.1.3 20 位计数器 1419
40.4.2 PWM 生成器 1420
40.4.3 占空比渐变 1421
40.4.3.1 线性占空比渐变 1421
40.4.3.2 伽马曲线渐变 1422
40.4.3.3 占空比渐变暂停和恢复 1423
40.5 事件任务矩阵功能 1423
40.6 中断 1425
40.7 配置流程 1425
40.8 存储空间 1427
40.9 寄存器列表 1428
40.10 寄存器 1430

41 电机控制脉宽调制器 (MCPWM) 1442
41.1 概述 1442
41.2 主要特性 1442
41.3 模块 1445

41.3.1 概述 1445
41.3.1.1 定时器模块 1445
41.3.1.2 操作器模块 1445
41.3.1.3 捕获模块 1446
41.3.1.4 ETM 模块 1446
41.3.2 PWM 定时器模块 1446
41.3.2.1 PWM 定时器模块的配置 1446
41.3.2.2 PWM 定时器工作模式和定时事件生成 1447
41.3.2.3 PWM 定时器影子寄存器 1451
41.3.2.4 PWM 定时器同步和锁相 1451
41.3.3 PWM 操作器模块 1451
41.3.3.1 PWM 生成器模块 1452
41.3.3.2 死区生成器模块 1463
41.3.3.3 PWM 载波模块 1467
41.3.3.4 故障检测模块 1470
41.3.4 捕获模块 1471
41.3.4.1 介绍 1471
41.3.4.2 捕获定时器 1471
41.3.4.3 捕获通道 1472
41.3.5 ETM 模块 1472
41.3.5.1 介绍 1472
41.3.5.2 MCPWM 可产生的 ETM 事件 1472
41.3.5.3 MCPWM 可接收的 ETM 任务 1473
41.4 中断 1473
41.5 寄存器列表 1475
41.6 寄存器 1478

42 红外遥控 (RMT) 1524
42.1 概述 1524
42.2 主要特性 1524
42.3 功能描述 1525
42.3.1 RMT 架构 1525
42.3.2 RAM 1526
42.3.2.1 RAM 结构 1526
42.3.2.2 RAM 使用说明 1526
42.3.2.3 RAM 访问方式 1527
42.3.3 时钟 1527
42.3.4 发射器 1527
42.3.4.1 普通发送模式 1527
42.3.4.2 乒乓发送模式 1528
42.3.4.3 发送加载波 1528
42.3.4.4 持续发送模式 1528
42.3.4.5 多通道同时发送 1529
42.3.5 接收器 1529
42.3.5.1 普通接收模式 1529
42.3.5.2 乒乓接收模式 1529

42.3.5.3 接收滤波 1529
42.3.5.4 接收去载波 1529
42.4 配置参数更新 1530
42.5 中断 1531
42.6 寄存器列表 1532
42.7 寄存器 1534

43 并行 IO 控制器 (PARLIO) 1547
43.1 概况 1547
43.2 术语 1547
43.3 特性 1547
43.4 系统架构 1548
43.5 功能描述 1549
43.5.1 时钟生成器 1549
43.5.2 时钟复位使用限制 1549
43.5.3 主从机模式 1550
43.5.4 RX 模块接收模式 1551
43.5.4.1 电平使能模式 1552
43.5.4.2 脉冲使能模式 1552
43.5.4.3 软件使能模式 1553
43.5.5 RX 模块 GDMA SUC EOF 信号生成 1553
43.5.6 RX 模块超时 1553
43.5.7 TX 模块无限制长度数据传输 1553
43.5.8 TX 片选功能 1554
43.5.9 TX 模块输出时钟门控 1554
43.5.10 TX 模块总线空闲值 1554
43.5.11 单帧数据传输 1554
43.5.12 字节范围内数据顺序翻转 1554
43.6 中断 1555
43.7 配置流程 1555
43.7.1 数据接收操作流程 1555
43.7.2 数据发送操作流程 1556
43.8 应用示例 1556
43.8.1 与 SPI 进行数据传输 1557
43.8.2 与 I2S 进行数据传输 1558
43.9 寄存器列表 1560
43.10 寄存器 1561

44 比特调节器 1572
44.1 概述 1572
44.2 特性 1573
44.3 应用示例 1573
44.4 架构 1573
44.4.1 数据路径 1573
44.4.2 控制路径 1575
44.5 功能描述 1575

44.5.1 指令 1575
44.5.2 配置寄存器 1582
44.6 配置流程 1583
44.7 寄存器列表 1585
44.8 寄存器 1586

VI 模拟信号处理 1598

45 温度传感器 1599
45.1 介绍 1599
45.2 特性 1599
45.3 结构概览 1599
45.4 功能描述 1600
45.4.1 温度传感器上电 1600
45.4.2 时钟管理 1600
45.4.3 自动监测模式 1600
45.4.4 温度测量范围与温度偏移 1601
45.4.5 数据转换 1601
45.5 配置流程 1601
45.6 中断 1602
45.7 事件任务矩阵功能 1602
45.8 寄存器列表 1603
45.9 寄存器 1604

46 ADC 控制器 1608
46.1 概述 1608
46.2 术语表 1608
46.3 特性 1608
46.4 结构概览 1609
46.5 功能描述 1610
46.5.1 ADC 上电 1610
46.5.2 ADC 通道 1610
46.5.3 ADC 时钟 1610
46.5.4 单次采样模式 1611
46.5.5 多通道采样模式 1611
46.5.6 ADC 转换和衰减 1611
46.5.7 DIG ADC FSM 1611
46.5.8 样式表 1612
46.5.9 ADC 滤波器 1614
46.5.10 阈值监控器 1615
46.5.11 GDMA 支持 1615
46.6 事件任务矩阵功能 1615
46.7 中断 1616
46.8 配置流程 1616
46.8.1 单次采样配置 1616
46.8.2 多通道采样配置 1617

46.9 寄存器列表 1618
46.10 寄存器 1619

47 模拟电压比较器 1630
47.1 概述 1630
47.2 主要特性 1630
47.3 系统架构 1630
47.4 功能描述 1631
47.5 事件任务矩阵 1632
47.6 中断 1632
47.7 配置流程 1632

VII 附录 1634

相关文档和资源 1635

词汇列表 1636
外设相关词汇 1636
寄存器相关缩写 1636
寄存器的访问类型 1638

如何配置寄存器的保留域 1640
概述 1640
如何配置保留域 1640

中断配置寄存器 1641

修订历史 1642

表格

2.4-1 CPU 地址分布 50
2.7-1 存储器映射寄存器访问的地址解码 80
2.7-2 内部存储器地址映射 80
2.8-1 CLIC 中断映射 82
2.8-2 CLIC 级别和优先级编码 (CLICINTCTLBITS=3) 82
2.8-4 核心本地中断源 87
2.10-3 进入调试模式时 DPC 中的虚拟地址 104
2.10-4 NAPOT 编码的 maddress 106

3.3-1 追踪编码器参数 126
3.5-1 可选边带信号 128
3.6-1 头部格式 129
3.6-2 索引格式 130
3.6-3 格式 3 子格式 0 130
3.6-4 格式 3 子格式 1 131
3.6-5 格式 3 子格式 3 131
3.6-6 格式 2 132
3.6-7 格式 1，有地址 133
3.6-8 格式 1，无地址 134
3.7-1 TRACE 的内部中断源 134
3.8-1 调试模块的触发支持（mcontrol 寄存器的 action 字段） 135
3.8-2 追踪状态 137

4.3-1 LP CPU 异常原因 160
4.6-1 性能计数器 167
4.9-1 唤醒源 170

5.4-1 GDMA 配置寄存器与外设选择关系表 175

6.3-1 地址映射 224
6.3-2 模块/外设地址空间映射表 229

7.3-1 BLOCK0 参数 234
7.3-2 密钥用途数值对应的含义 237
7.3-3 BLOCK1-10 参数 237
7.3-4 用户读取寄存器信息 241
7.3-5 VDDQ 默认时序参数配置 243
7.4-1 eFuse 控制器的内部中断源 243

8.4-1 HP GPIO 唤醒信号触发条件和清除条件 283
8.4-2 LP GPIO 唤醒信号触发条件和清除条件 284
8.8-1 IO MUX Light-sleep 管脚功能控制寄存器 290
8.12-1 GPIO 交换矩阵外设信号 294
8.13-1 HP IO MUX 管脚功能 299
8.14-1 LP IO MUX 管脚功能 300

8.15-1 GPIO 管脚模拟功能 301
8.17-1 HP IO MUX 和 HP GPIO 交换矩阵内部中断源 302
8.17-2 LP IO MUX 和 LP GPIO 交换矩阵内部中断源 303

9.1-1 复位源 362
9.2-1 CPU_CLK 时钟源选择 365
9.2-2 CPU_CLK、AHB_CLK 和 HP_ROOT_CLK 的时钟频率 365
9.2-3 衍生高性能时钟源 367
9.2-4 高性能系统外设时钟 368
9.2-5 衍生低功耗时钟源 368
9.2-6 低功耗系统外设时钟 369
9.2-7 控制模块的读写寄存器时钟门控与 PMU 相关寄存器映射关系 371
9.2-8 控制模块的工作时钟门控与 PMU 相关寄存器映射关系 372

10.2-1 Strapping 管脚默认配置 443
10.2-2 芯片启动模式控制 443
10.2-3 SDIO 输入采样沿/输出驱动沿控制 445
10.2-4 UART0 ROM 日志打印控制 445
10.2-5 USB 串口/JTAG ROM 日志打印控制 446
10.2-6 JTAG 信号源控制 446

11.5-1 CPU 外部中断源映射寄存器、外部中断状态寄存器、外部中断源 450

12.3-1 事件任务矩阵通道*n* 可选事件 468
12.3-2 事件任务矩阵通道*n*可映射任务 471

13.4-1 唤醒源 577
13.5-1 预设的功耗模式 581

14.5-1 UNIT*n* 配置控制位 631
14.5-2 报警触发条件 632
14.5-3 同步操作 633
14.6-1 系统定时器的内部中断源 634

15.4-1 可逆计数器向上计数时的报警场景 655
15.4-2 可逆计数器向下计数时的报警场景 656
15.6-1 TIMG*n* 的内部中断源 658

16.2-1 超时动作 678

17.3-1 配置 RTC 定时器记录特定事件的发生时刻 690
17.3-2 RTC 定时器目标时刻配置 690
17.5-1 RTC 定时器的内部中断源 691

18.1-1 PMP 和 APM 管控的区域分布 701
18.4-2 TEE 与 REE 之间的比较 704
18.5-1 HP 系统主机访问来源 704
18.5-2 APM 控制器配置信息 706

19.3-1 安全等级 757

20.5-1 HP CPU 包格式 774
20.5-2 LP CPU 包格式 774
20.5-3 DMA 包格式 774
20.5-4 LOST 包格式 774
20.5-5 DMA 访问来源 775
20.5-6 DMA_0 包格式 777
20.5-7 DMA LOST 包格式 777

21.3-1 电源管脚检测的配置 808
21.4-1 电源检测器的内部中断源 808

22.4-1 工作模式 820
22.4-2 密钥长度和加解密方向 820
22.5-1 Typical AES 模式下状态返回值 821
22.5-2 Typical AES 文本字节序 821
22.5-3 AES-128 密钥字节序 822
22.5-4 AES-256 密钥字节序 823
22.6-1 块模式选择 824
22.6-2 DMA-AES 模式下状态返回值 825
22.6-3 TEXT-PADDING 825
22.6-4 DMA-AES 存储字节序 826

23.3-1 ECC 硬件加速器内存块 837
23.4-1 ECC 加速器曲线模式控制 839
23.4-2 ECC 硬件加速器工作模式控制 839

24.4-1 HMAC 功能及配置数值 854

25.3-1 加速效果 870
25.4-1 RSA 的内部中断源 870

26.3-1 工作模式选择 879
26.3-2 运算标准选择 879
26.4-1 不同运算标准信息摘要的寄存器占用情况 882
26.4-2 SHA 加速器的内部中断源 882

28.4-1 ECDSA_DS 工作模式 900
28.4-2 ECDSA_DS 椭圆曲线选择 900
28.4-3 ECDSA_DS SHA 算法 901
28.4-4 ECDSA_DS 工作状态 901
28.7-1 ECDSA_DS 存储器块 909

29.4-1 根据 Key_A 生成的 Key 值 920
29.4-2 目标空间与寄存器堆的映射关系 921
29.6-1 XTS_AES 伪轮次功能配置 925

31.6-1 HUK 生成器工作模式 942
31.9-1 密钥管理器和 HUK 生成器内部中断源 956
31.10-1 密钥管理器内存块 957

31.10-2 HUK 生成器内存块 957

32.2-1 UART 和 LP UART 特性区分 973
32.4-1 UART_CHAR_WAKEUP 模式配置 981

33.5-1 GP-SPI2 支持的数据模式 1054
33.5-2 FSPI 总线信号功能描述 1055
33.5-3 各种 SPI 模式下使用到的信号 1056
33.5-4 GP-SPI2 用作主机和从机时的数据位控制 1058
33.5-5 GP-SPI2 用作主机或从机时所支持的传输类型 1059
33.5-6 GP-SPI2 用作从机时数据传输的中断触发条件 1063
33.5-7 1/2/4-bit 模式下状态控制寄存器 1069
33.5-8 命令值的发送顺序 1071
33.5-9 地址值的发送顺序 1071
33.5-10 CONF 阶段 BM 位图 1076
33.5-11 传输事务 *i* 中 CONF buffer*i* 配置示例 1077
33.5-12 BM 位图与待更新的寄存器 1077
33.5-13 GP-SPI2 从机 SPI 模式支持的 CMD 值 1080
33.5-14 QPI 模式支持的 CMD 值 1081
33.7-1 主机时钟相位和极性配置 1086
33.7-2 从机时钟相位和极性配置 1087
33.9-1 GP-SPI2 的内部中断源 1089
33.9-2 GP-SPI2 用作主机时用到的中断 1090
33.9-3 GP-SPI2 用作从机时用到的中断 1091

34.3-1 I2C 时序参数（引自 The I2C-bus specification Version 2.1 Table 5） 1134
34.4-1 I2C 同步寄存器 1137

35.4-1 模块信号描述 1210
35.9-1 通道有效数据位宽控制 1217
35.9-2 通道有效数据字节序控制 1217
35.9-3 PDM 输出模式下 I2S 取数逻辑 1220
35.9-4 原始 PDM 数据的 I2S 通道模式控制 1221
35.9-5 I2S PCM-to-PDM 数据输出 1221
35.10-1 通道储存数据位宽控制 1224
35.10-2 通道储存数据字节序控制 1225

36.2-1 控制信号为低电平时输入脉冲信号上升沿的计数模式 1251
36.2-2 控制信号为高电平时输入脉冲信号上升沿的计数模式 1251
36.2-3 控制信号为低电平时输入脉冲信号下降沿的计数模式 1251
36.2-4 控制信号为高电平时输入脉冲信号下降沿的计数模式 1252

37.3-1 标准 CDC-ACM 控制请求 1266
37.3-2 CDC-ACM 中 RTS 和 DTR 的设置 1267
37.3-3 半字节中的命令 1268
37.3-4 USB-JTAG 控制请求 1270
37.3-5 JTAG 功能描述符 1270
37.5-1 复位芯片进入下载模式 1272

37.5-2 复位 SoC 从 flash 启动 1272

38.3-1 CAN 实现类型 1307
38.3-3 重发限制配置 1311

39.5-1 SDIO 从机的 CCCR 设置 1365
39.5-2 SDIO 从机的 FBR 设置 1366

40.4-1 常用配置频率及精度 1420
40.8-1 LEDC 存储空间 1427

41.3-1 操作器模块包含的各个子模块功能 1446
41.3-2 PWM 生成器中的所有定时事件 1453
41.3-3 PWM 定时器递增计数时，定时事件的优先级 1454
41.3-4 PWM 定时器递减计数时，定时事件的优先级 1454
41.3-5 控制死区时间生成器开关的字段 1464
41.3-6 死区生成器的典型操作模式 1465
41.3-7 MCPWM 可产生的 ETM 事件 1472
41.3-8 MCPWM 可接收的 ETM 任务 1473

42.4-1 更新配置参数 1530

43.5-1 时钟限制下复位 AHB 时钟域操作步骤 1550
43.5-2 时钟限制下 TX 模块从机模式操作要求 1551
43.5-3 时钟限制下 RX 模块从机模式操作要求 1551
43.6-1 PARLIO 的内部中断源 1555

44.5-1 指令结构 1576
44.5-2 当 CTL_MUX_REL 为 1 时 CTL_MUX_SEL_n 生效的值 1577
44.5-3 CTL_MUX_SEL 的值 1578
44.5-4 指令操作码 1579
44.5-5 LOOP 操作码 1579
44.5-6 ADD 操作码 1580
44.5-7 IF 操作码 1580
44.5-8 IFN 操作码 1580
44.5-9 LDCTD 操作码 1581
44.5-10 LDCTI 操作码 1581
44.5-11 ADDCTI Opcode 1582
44.5-12 EOF 模式设置 1583

45.4-1 温度测量范围与温度偏移 1601

46.5-1 SAR ADC 通道 1610

47.4-1 模拟电压比较器 PAD 和 GPIO 之间的映射关系 1631
47.6-1 模拟电压比较器的内部中断源 1632

47.7-4 ENA/RAW/ST 寄存器的配置 1641

插图

1.0-1 ESP32-C5 总线架构 46

2.1-1 CPU 核心系统框图 48
2.10-1 调试系统概述 100
2.11-1 2 位饱和计数器 115
2.12-1 使用 RunStall 时 CPU 核心状态的切换 122

3.0-1 追踪编码器概述图 124
3.2-1 指令追踪流程概览 125

4.0-1 LP CPU 概览 151
4.9-1 LP CPU 唤醒和睡眠流程 169

5.1-1 共用 GDMA 通道的模块 172
5.3-1 GDMA 控制器的架构 173
5.4-1 链表结构图 174
5.4-2 链表关系图 176

6.3-1 系统结构与地址映射结构 224
6.3-2 ROM-Cache 系统结构 225
6.3-3 Cache 系统结构 227
6.3-4 具有 GDMA 功能的外设/模块 229

7.3-1 eFuse 系统数据通路 233
7.3-2 移位寄存器电路图（前 32 字节） 239
7.3-3 移位寄存器电路图（后 12 字节） 239

8.3-1 HP GPIO 交换矩阵、HP IO MUX、LP GPIO 交换矩阵和 LP IO MUX 框图 280
8.3-2 焊盘内部结构 281
8.4-1 GPIO 输入经 HP IO MUX 运行时钟上升沿或下降沿同步 282
8.4-2 GPIO 输入信号滤波时序图 282
8.4-3 毛刺滤波器时序示例 285
8.10-1 未使能迟滞功能时 PAD 上的电平翻转示例 291
8.10-2 使能迟滞功能时 PAD 上的电平翻转示例 292

9.1-1 四种复位等级 361
9.2-1 系统时钟 364
9.3-1 时钟配置示例 373

10.2-1 芯片启动流程 444

11.2-1 ESP32-C5 的中断流 448
11.4-1 中断矩阵结构图 448

12.3-1 事件任务矩阵通道*n*架构 468
12.3-2 事件任务矩阵时钟结构 475

13.4-1 ESP32-C5 电源结构 573
13.4-2 PMU 的主要工作流程 575
13.6-1 ESP32-C5 唤醒后的启动流程 582

14.3-1 系统定时器结构图 630
14.5-1 系统定时器生成报警 631

15.1-1 定时器组概览 653
15.3-1 定时器组架构 654

16.1-1 看门狗定时器概览 675
16.2-1 ESP32-C5 的数字看门狗定时器 677
16.3-1 SWD 控制器结构 679

18.1-1 PMP 和 APM 的管控关系 702
18.5-1 APM 控制器结构图 705

21.3-1 电源检测器架构图 806
21.3-2 欠压检测器处理方式 807
21.3-3 电压毛刺检测器 808

23.4-1 clk_ecc 时钟图 843
23.4-2 clk_apb_ecc 时钟图 844

24.6-1 HMAC 附加填充位示意图 857
24.6-2 HMAC 结构示意图 857

27.3-1 软件准备工作与硬件工作流程 889

28.5-1 ECDSA_DS 流程 904
28.5-2 clk_ecdsa 时钟图 907
28.5-3 clk_apb_ecdsa 时钟图 907

29.3-1 片外存储器加解密结构 919

30.3-1 噪声源 934

31.5-1 密钥管理器 & HUK 生成器架构 941
31.6-1 AES 部署模式 944
31.6-2 ECDH0 部署模式 945
31.6-3 ECDH1 部署模式 947

32.3-1 UART 基本架构图 974
32.4-1 UART 控制器分频 976
32.4-2 UART 信号下降沿较差时序图 977
32.4-3 UART 数据帧结构 977
32.4-4 AT_CMD 字符格式 978
32.4-5 RS485 模式驱动控制结构图 979
32.4-6 SIR 模式编解码时序图 980
32.4-7 IrDA 编解码结构图 980

32.4-8 硬件流控图 981
32.4-9 硬件流控信号连接图 982
32.4-10 GDMA 模式数据传输 983
32.6-1 UART 编程流程 986

33.4-1 SPI 模块概览 1054
33.5-1 CPU 控制的传输中使用的数据 Buffer 1060
33.5-2 GP-SPI2 功能块图 1064
33.5-3 GP-SPI2 用作主机时的数据流控制 1065
33.5-4 GP-SPI2 用作从机时的数据流控制 1066
33.5-5 GP-SPI2 主机状态机 1068
33.5-6 GP-SPI2 主机使用全双工模式与 SPI 从机通信框图 1072
33.5-7 4-bit 模式下 GP-SPI2 与 Flash 以及外部 RAM 的连接方式 1074
33.5-8 GP-SPI2 发送到 Flash 的 SPI Quad I/O 命令序列 1074
33.5-9 DMA 控制的分段配置传输 1075
33.6-1 GP-SPI2 访问外部 RAM 时推荐的 CS 时序配置 1084
33.6-2 GP-SPI2 访问 Flash 时推荐的 CS 时序配置 1084
33.7-1 SPI 时钟模式 0 和时钟模式 2 1085
33.7-2 SPI 时钟模式 1 和时钟模式 3 1086
33.8-1 GP-SPI2 主机时序补偿控制图 1088
33.8-2 GP-SPI2 主机时序补偿示例 1089

34.3-1 I2C 主机基本架构 1133
34.3-2 I2C 从机基本架构 1133
34.3-3 I2C 协议时序（引自 The I2C-bus specification Version 2.1 Fig.31） 1134
34.4-1 I2C 时序图 1138
34.4-2 I2C 命令寄存器结构 1140
34.7-1 I2C 主机写 7 位寻址的从机 1145
34.7-2 I2C 主机写 10 位寻址的从机 1146
34.7-3 I2C 主机写 7 位双地址寻址从机 1148
34.7-4 I2C 主机分段写 7 位寻址的从机 1150
34.7-5 I2C 主机读 7 位寻址的从机 1152
34.7-6 I2C 主机读 10 位寻址的从机 1154
34.7-7 I2C 主机从 7 位寻址从机的 M 地址读取 N 个数据 1156
34.7-8 I2C 主机分段读 7 位寻址的从机 1158

35.4-1 ESP32-C5 I2S 系统框图 1209
35.5-1 时序图 – TDM Philips 标准 1211
35.5-2 时序图 – TDM MSB 对齐标准 1212
35.5-3 时序图 – TDM PCM 标准 1212
35.5-4 时序图 – PDM 标准 1213
35.6-1 I2S 时钟分频器 1213
35.9-1 TX 数据格式控制 1219
35.9-2 TDM 通道控制 1220
35.9-3 PDM 通道控制 1222

36.0-1 PCNT 框图 1248

36.2-1 PCNT 单元基本架构图 1250
36.3-1 通道 0 递增计数图 1253
36.3-2 通道 0 递减计数图 1253
36.3-3 双通道递增计数图 1254

37.2-1 USB 串口/JTAG 控制器高层框图 1265
37.2-2 USB 串口/JTAG 控制器框图 1266

38.3-1 TWAI_CLK Diagram 1302
38.3-2 工作模式 1303
38.3-3 位时间 1304
38.3-4 发送器延迟 1305
38.3-5 发送器延迟测量 1305
38.3-6 二次采样点 1306
38.3-7 二次采样点 2 1306
38.3-8 TX buffer 状态 1308
38.3-9 TX buffer 选择 1309
38.3-10 定时发送 1310
38.3-11 TX 帧类型 1310
38.3-12 RX buffer 1312
38.3-13 帧过滤器工作流程（* 以 A/B/C/R 类型过滤器为例） 1314
38.3-14 故障限制 1315
38.3-15 TX buffer 对 1316
38.3-16 TX buffer 备份模式工作流程 1317

39.3-1 SDIO 从机功能结构图 1363
39.5-1 CMD52 内容 1364
39.5-2 CMD53 内容 1364
39.5-3 function 0 地址空间 1364
39.5-4 function 1/2 地址空间 1367
39.5-5 SDIO 从机 DMA 链表描述符结构 1368
39.5-6 SDIO 从机 DMA 链表 1369
39.5-7 主机向从机发送递增地址包的数据流 1370
39.5-8 采样时序图 1371
39.5-9 输出时序图 1371
39.7-1 从机向主机发送包的流程 1373
39.7-2 从机从主机接收包的流程 1375
39.7-3 从机 CPU 挂载 buffer 的流程 1376

40.3-1 LED PWM 控制器架构 1417
40.3-2 定时器和 PWM 生成器功能框图 1417
40.4-1 LEDC_CLK_DIV 为非整数时的分频 1419
40.4-2 计数器和 PWM 信号精度关系 1419
40.4-3 LED PWM 输出信号图 1421
40.4-4 输出信号占空比线性渐变图 1422
40.4-5 输出信号占空比伽马曲线渐变图 1423

41.2-1 MCPWM 外设概览 1443

41.3-1 定时器模块 1445
41.3-2 操作器模块 1445
41.3-3 捕获模块 1446
41.3-4 ETM 模块 1446
41.3-5 递增计数模式波形 1447
41.3-6 递减计数模式波形 1448
41.3-7 递增递减循环模式波形，同步事件后递减 1448
41.3-8 递增递减循环模式波形，同步事件后递增 1448
41.3-9 递增模式中生成的 UTEP 和 UTEZ 1449
41.3-10 递减模式中生成的 DTEP 和 DTEZ 1450
41.3-11 递增递减模式中生成的 DTEP 和 UTEZ 1450
41.3-12 PWM 操作器的子模块 1452
41.3-13 递增递减模式下的对称波形 1455
41.3-14 递增计数模式，单边不对称波形，PWM*x*A 和 PWM*x*B 独立调制–高电平 1456
41.3-15 递增计数模式，脉冲位置不对称波形，PWM*x*A 独立调制 1457
41.3-16 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–高电平有效 1458
41.3-17 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–互补 1459
41.3-18 递增递减循环计数模式，错误或同步事件，在 PWM*x*A 和 PWM*x*B 上相同调制 1460
41.3-19 NCI 在 PWM*x*A 输出上软件强制事件示例 1461
41.3-20 CNTU 在 PWM*x*B 输出上软件强制事件示例 1462
41.3-21 死区模块的开关拓扑 1464
41.3-22 高电平有效互补 (AHC) 死区波形 1465
41.3-23 低电平有效互补 (ALC) 死区波形 1466
41.3-24 高电平有效 (AH) 死区波形 1466
41.3-25 低电平有效 (AL) 死区波形 1467
41.3-26 PWM 载波操作的波形示例 1468
41.3-27 载波模块的第一个脉冲和之后持续的脉冲示例 1469
41.3-28 PWM 载波模块中持续脉冲的 7 种占空比设置 1470

42.3-1 RMT 结构框图 1525
42.3-2 RAM 中脉冲编码结构 1526

43.4-1 PARLIO 系统框图 1548
43.5-1 PARLIO 时钟生成 1549
43.5-2 主机时钟正向波形 1551
43.5-3 主机时钟负向波形 1551
43.5-4 RX 模块电平使能子模式 1552
43.5-5 RX 模块脉冲使能子模式 1552
43.5-6 RX 模块软件使能子模式 1553

44.1-1 比特调节器系统架构图 1572
44.4-1 比特调节器数据路径图 1574
44.4-2 比特调节器控制路径图 1575

45.3-1 温度传感器结构框图 1600

46.4-1 SAR ADC 结构图 1609
46.5-1 SAR ADC 时钟结构图 1610

46.5-2 DIG ADC FSM 概况 1612
46.5-3 APB_SARADC_SAR_PATT_TAB1_REG 包含样式 0 - 3 1613
46.5-4 APB_SARADC_SAR_PATT_TAB2_REG 包含样式 4 - 7 1613
46.5-5 样式结构 1613
46.5-6 cmd0 配置示例 1614
46.5-7 cmd1 配置示例 1614
46.5-8 GDMA 数据格式 1615

47.3-1 模拟电压比较器系统框图 1630

第 I 卷

微处理器和主机

该部分介绍构成芯片系统的基本要素，包括高性能 CPU 和低功耗 CPU 的微处理器架构、用于调试的跟踪编码、直接内存访问 (DMA) 控制器等。

第 1 章

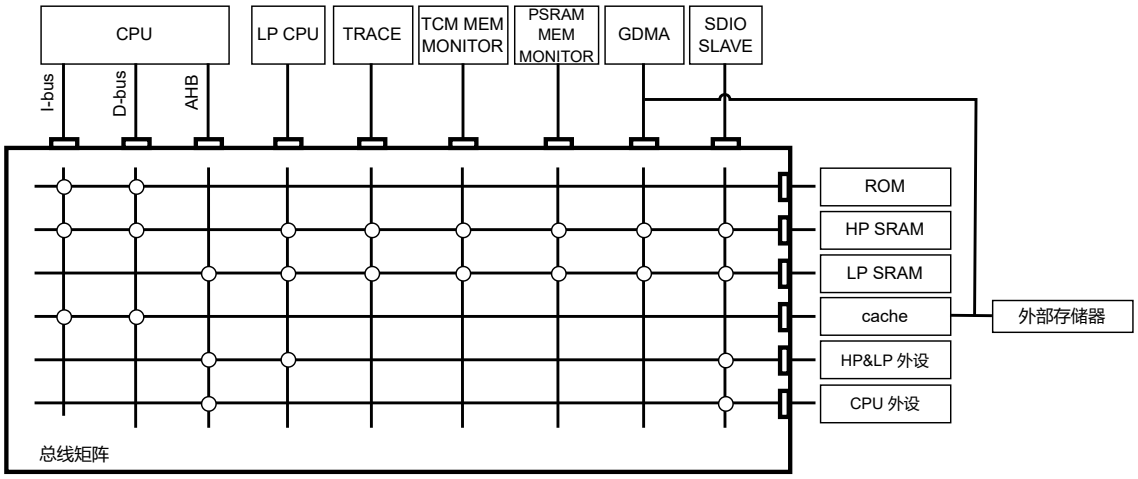
总线架构

ESP32-C5 SoC 采用多层总线矩阵作为通信架构的核心，通过高效互连的结构连接多个主设备和从设备。该架构支持并发访问，有效缓解访问冲突，使高性能和低功耗的 RISC-V 核心、DMA 控制器、调试接口及各类外设能够在竞争最小的情况下同时运行。通过对数据流与系统性能进行优化，即使多个高速外设同时工作，总线矩阵也能够确保系统高效运转。

ESP32-C5 SoC 的主系统由以下部分组成：

- 九个主设备 (master)：
  - 高性能 (HP) 32 位 RISC-V 处理器的 I-bus、D-bus 和 AHB 总线
  - 低功耗 (LP) 32 位 RISC-V 处理器
  - TRACE
  - TCM MEM MONITOR
  - PSRAM MEM MONITOR
  - GDMA
  - SDIO SLAVE
- 六个从设备 (slave)：
  - ROM
  - HP SRAM
  - LP SRAM
  - Cache
  - HP&LP 外设
  - CPU 外设

ESP32-C5 的总线架构如下图所示。



注：

1. 图中交叉点处的圆圈表示对应列的总线可以访问对应行的从设备。
2. GDMA 上存在一条不经过总线矩阵、直接访问外部存储器的数据通路。
3. CPU 访问外部存储器时必须通过 cache；通常情况下，cache 对用户而言是透明的。

图 1.0-1. ESP32-C5 总线架构

图 1.0-1 *ESP32-C5 总线架构* 中的总线说明如下：

- **HP CPU I-bus 总线**：连接高性能 (HP) 32 位 RISC-V 处理器的指令总线和总线矩阵，用于处理器取指。其可访问对象为包含代码的存储器（如 ROM、HP SRAM 或通过 MSPI 访问的外部存储器）。
- **HP CPU D-bus 总线**：连接高性能 (HP) 32 位 RISC-V 处理器的数据总线和总线矩阵，用于字面量加载和调试访问。其可访问对象为存储代码或数据的存储器（如 ROM、HP SRAM 或通过 MSPI 访问的外部存储器）。
- **HP CPU AHB 总线**：连接高性能 (HP) 32 位 RISC-V 处理器的 AHB 总线和总线矩阵，用于访问外设或 LP SRAM 中的数据。可访问对象包括 LP SRAM、HP&LP 外设和 CPU 外设。
- **LP CPU 总线**：连接低功耗 (LP) 32 位 RISC-V 处理器的 AHB 总线和总线矩阵，供 LP CPU 访问内存和外设。可访问对象包括 HP SRAM、LP SRAM 和 HP&LP 外设。
- **TRACE 总线**：连接 TRACE 模块和总线矩阵，用于执行调试操作。可访问对象包括 HP SRAM 和 LP SRAM。
- **TCM MEM MONITOR 总线**：连接 TCM MEM MONITOR 和总线矩阵，用于将 TCM MEM MONITOR 监控数据写入内存。可访问对象包括 HP SRAM 和 LP SRAM。
- **PSRAM MEM MONITOR 总线**：连接 PSRAM MEM MONITOR 和总线矩阵，用于将 PSRAM MEM MONITOR 记录的监控信息写入内存。可访问对象包括 HP SRAM 和 LP SRAM。
- **GDMA 总线**：提供一条连接至总线矩阵的线路和一条连接外部存储器的线路。GDMA 通过该总线进行高速数据传输，包括外设到内存、内存到外设、外设到外设以及内存到内存的传输。可访问对象包括 HP SRAM、LP SRAM 和外部存储器。针对 GDMA 访问外部存储器的场景，为确保数据吞吐率，建议尽可能设置较长的 burst 长度。
- **SDIO SLAVE 总线**：连接 SDIO SLAVE 和总线矩阵，用于访问内存或外设中的数据。可访问对象包括 HP SRAM、LP SRAM、HP&LP 外设和 CPU 外设。

说明：
通过 TRACE、TCM MEM MONITOR 或 PSRAM MEM MONITOR 访问 LP SRAM 的速度较慢，且 LP SRAM 空间较小，因此不建议使用这些路径访问 LP SRAM。

第 2 章

高性能处理器

2.1 概述

ESP32-C5 采用了乐鑫第二代 32 位 RISC-V 指令集架构 (ISA) 的 CPU 核心系统。核心系统包含单核 RISC-V CPU、专用的核心本地中断控制器 (CLIC)、调试模块以及核心本地中断 (CLINT) 定时器。图 2.1 为 CPU 核心系统的框图。

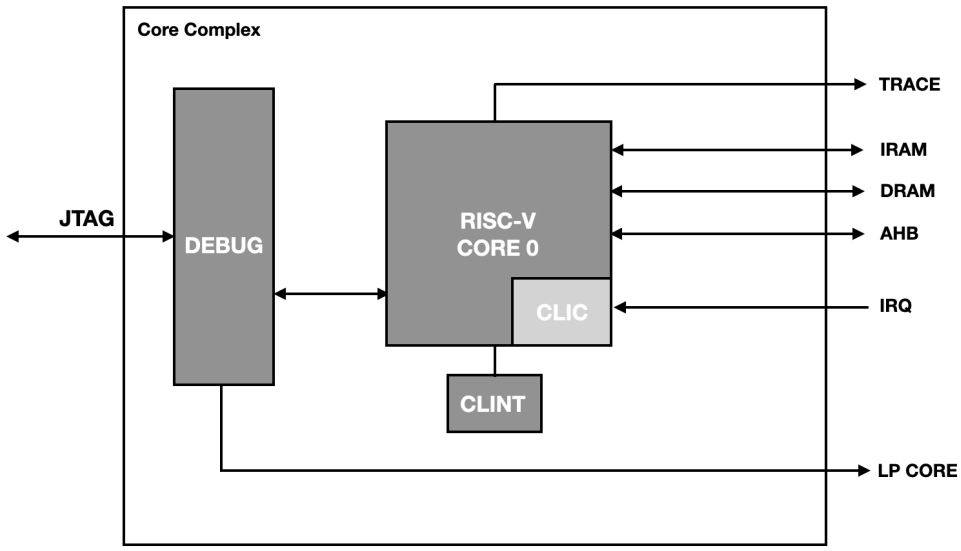


图 2.1-1. CPU 核心系统框图

在 CPU 核心系统中，RISC-V 核心使用 5 级顺序标量流水线架构，针对面积、功耗和性能进行了优化，内置专用的核心本地中断控制器 (CLIC) 和系统总线 (SYS BUS) 接口，用于访问存储器和外设。调试模块将 JTAG 接口分别连接至 RISC-V 核心和 LP 核心调试。RISC-V 核心还提供用于离线调试的追踪接口。

2.2 特性

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

- 必须实现的基本整数 (I)
- 乘法/除法 (M)
- 原子 (A)

- 压缩 (C)
- 代码体积优化 (Zc)

除了上述 RISC-V 指令扩展，CPU 核心还支持以下特性：

- 具有五级流水线，支持高达 240 MHz 的工作时钟频率
- 兼容《RISC-V 指令集手册 v2.2 第一卷 “RISC-V 用户级 ISA”》(RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA, Version 2.2)
- 兼容《RISC-V 指令集手册 v1.10 第二卷 “特权架构”》(RISC-V Instruction Set Manual, Volume II: RISC-V Privileged Architecture, Version 1.10)
- 兼容《核心本地中断控制器 (CLIC) v0.9》(RISC-V Core-Local Interrupt Controller (CLIC) Version 0.9)
- 兼容《RISC-V 外部调试支持规范 v0.13.2》(RISC-V External Debug Support Version 0.13.2)
- 通过 IRAM/DRAM 接口零等待周期访问片上 SRAM 和缓存中的程序和数据
- 32 位 AHB 系统总线用于访问外设
- 核心本地中断 (CLINT)
- 支持用户 (U) 特权模式
- 总线错误响应
- 标准物理存储器保护 (PMP) 和自定义物理存储器属性 (PMA)，最多可配置 16 个区域以确保安全
- 分支目标缓冲器 (BTB) 具有 BHT 和 RAS 以提高性能
- 调试模块 (DM) 符合《RISC-V 外部调试支持规范 v0.13.2》，支持通过行业标准的 JTAG/USB 端口连接外部调试器
- 调试器通过核心数据总线以及系统总线 (SBA) 直接访问存储器和外设
- 支持指令追踪以进行离线调试
- 硬件触发器符合《RISC-V 调试规范 v0.13.2》(RISC-V External Debug Support Version 0.13.2)，具有多达 3 个断点/观察点
- 支持多达 8 个专用 IO
- 可配置的核心性能指标事件

2.3 术语

为了更好地说明 ESP-RISC-V CPU 的功能，本章使用了以下术语。

ABI	应用二进制接口 (application binary interface)
分支 (branch)	在一定条件下改变执行流程的指令
CLIC	核心本地中断控制器 (core-local interrupt controller)
CLINT	核心本地中断 (core-local interrupt)
CSR	控制和状态寄存器 (control and status register)
delta	指令在存储器中的存储地址不连续时，程序计数器的地址偏移
DTM	调试传输模块 (debug transport module)

FPR 浮点寄存器 (floating-point register)
GPIO 通用输入/输出 (general-purpose input/output)
GPR 通用寄存器 (general-purpose register)
HWLP 硬件循环 (hardware loop)
hart RISC-V 硬件线程
LSB 最低有效位 (least significant bit)
LR 加载保留 (load reserved)
MSB 最高有效位 (most significant bit)
PMP 物理存储器保护 (physical memory protection)
退役 (retire) 机器状态更新时，执行指令的最后一个阶段
RO 只读域 (read-only)
R/W 可读写域 (readable and writable)
SC 条件存储 (store conditional)
SoC 片上系统 (system on chip)
WO 只写域 (write-only)
WARL 写入任何读取合法 (write any read legal)
W1 写入 1 (write 1)

2.4 地址分布

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

表 2.4-1. CPU 地址分布

名称	描述	起始地址	结束地址	访问类型
CPU	CLIC/CLINT 寄存器	0x2000_0000	0x2FFF_FFFF	R/W
IRAM/DRAM	指令/数据区域	0x4000_0000	0x4FFF_FFFF	R/W
CPU 外设	AHB（强顺序访问）	0x6000_0000	0x6FFF_FFFF	R/W
AHB	AHB（弱顺序访问）	* 默认	* 默认	R/W

*** 默认：**上述地址范围以外的地址空间通过 AHB 总线访问。

2.5 配置和状态寄存器 (CSRs)

2.5.1 寄存器列表

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

名称	描述	地址	访问
机器信息 CSR			
mvendorid	机器供应商 ID	0xF11	RO
marchid	机器架构 ID	0xF12	RO
mimpid	机器实现 ID	0xF13	RO

名称	描述	地址	访问
mhartid	机器 hart ID	0xF14	RO
机器陷阱设置 CSR			
mstatus	机器模式状态	0x300	R/W
misa [1]	机器 ISA	0x301	R/W
mideleg	机器中断委托寄存器（在 CLIC 模式下无效）	0x303	RO
mie	机器中断使能寄存器（在 CLIC 模式下无效）	0x304	RO
mtvec [2]	机器陷阱向量	0x305	R/W
mcounteren	机器计数器使能	0x306	R/W
mtvt	机器向量中断基地址（参见 CLIC 规范）	0x307	R/W
机器陷阱处理 CSR			
mscratch	机器暂存	0x340	R/W
mepc	机器陷阱程序计数器	0x341	R/W
mcause [3]	机器陷阱原因	0x342	R/W
mtval	机器陷阱值	0x343	R/W
mip	机器中断挂起（在 CLIC 模式下无效）	0x344	RO
mnxti	中断处理程序地址和使能（参见 CLIC 规范）	0x345	R/W
mintthresh	中断阈值 CSR （详情请参考 CLIC 规范）	0x347	RO
mintstatus	当前中断级别（参见 CLIC 规范）	0xFB1	RO
mscratchcsw	特权模式改变时的条件暂存交换（参见 CLIC 规范）	0x348	R/W
mscratchcswl	级别改变时的条件暂存交换（参见 CLIC 规范）	0x349	R/W
mclicbase	机器模式中断控制器基地址寄存器（自定义）（参见 CLIC 规范）	0x350	RO
物理存储器保护 (PMP) CSR			
pmpcfg0	物理存储器保护配置	0x3A0	R/W
pmpcfg1	物理存储器保护配置	0x3A1	R/W
pmpcfg2	物理存储器保护配置	0x3A2	R/W
pmpcfg3	物理存储器保护配置	0x3A3	R/W
pmpaddr*n* (*n*: 0-15)	物理存储器保护地址寄存器	0x3B0+0x1**n*	R/W
触发器模块 CSR（与调试模式共享）			
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
tdata3	触发器抽象数据寄存器 3	0x7A3	R/W
tinfo	触发器信息寄存器	0x7A4	R/W
tcontrol	全局触发器控制	0x7A5	R/W
mcontext	机器模式上下文寄存器	0x7A8	R/W
调试模式 CSR（仅在调试模式下可访问）			
dcsr	调试控制和状态	0x7B0	R/W
dpc	调试 PC	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试暂存寄存器 1	0x7B3	R/W

[1]尽管 misa 具有读/写属性，但由于它的域是硬连线的，所以写操作无效。在 RISC-V 术语中称为 WARL（写入任意数值读取合法数值）。

[2]mtvec 保存了陷阱向量配置，包括向量基地址和向量模式。

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

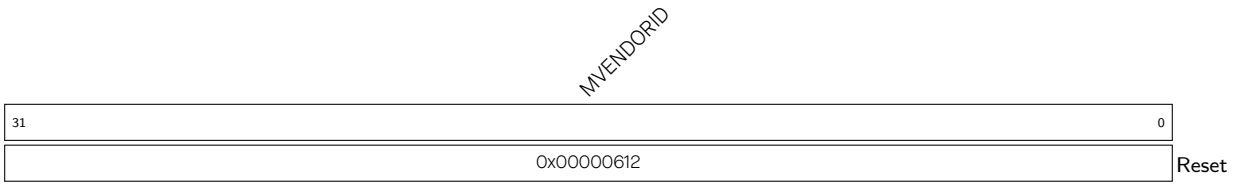
名称	描述	地址	访问
机器计数器设置			
mcountinhibit	机器计数器控制寄存器	0x320	R/W
mhpmevent8	机器性能监控事件选择器	0x308	R/W
mhpmevent9	机器性能监控事件选择器	0x329	R/W
mhpmevent13	机器性能监控事件选择器	0x32D	R/W
机器计数器/定时器			
mcycle	机器周期计数器	0xB00	R/W
minstret	机器指令退役计数器	0xB02	R/W
mhpmcounter8	机器性能监控计数器	0xB08	R/W
mhpmcounter9	机器性能监控计数器	0xB09	R/W
mhpmcounter13	机器性能监控计数器	0xB0D	R/W
mcycleh	mcycle 的高 32 位	0xB80	R/W
minstreth	minstret 的高 32 位	0xB82	R/W
mhpmcounter8h	mhpmcounter8 的高 32 位	0xB88	R/W
mhpmcounter9h	mhpmcounter9 的高 32 位	0xB89	R/W
mhpmcounter13h	mhpmcounter13 的高 32 位	0xB8D	R/W
非特权/用户模式计数器/定时器			
cycle	RDCYCLE 指令的周期计数器	0xC00	RO
time	RDTIME 指令的定时器	0xC01	RO
instret	RDINSTRET 指令的退役计数器	0xC02	RO
hpmcounter8	性能监控计数器	0xC08	RO
hpmcounter9	性能监控计数器	0xC09	RO
hpmcounter13	性能监控计数器	0xC0D	RO
cycleh	cycle 的高 32 位	0xC80	RO
instreth	instret 的高 32 位	0xC82	RO
hpmcounter8h	hpmcounter8 的高 32 位	0xC88	RO
hpmcounter9h	hpmcounter9 的高 32 位	0xC89	RO
hpmcounter13h	hpmcounter13 的高 32 位	0xC8D	RO
自定义系统 CSR（自定义）			
GPIO 访问 CSR（自定义）			
cpu_gpio_oen	GPIO 输出使能	0x803	R/W
cpu_gpio_in	GPIO 输入值	0x804	RO
cpu_gpio_out	GPIO 输出值	0x805	R/W
Zc 扩展 CSR			
jvt	机器向量跳转表基地址和控制寄存器	0x017	R/W
机器扩展状态 CSR（自定义）			
mhcr	性能配置控制状态寄存器	0x7C1	R/W
mexstatus	机器扩展控制和状态寄存器	0x7E1	R/W
mhint	机器隐式操作寄存器	0x7C5	R/W
物理存储器属性检查器 (PMAC) CSR（自定义）			
pma_cfg*n* (*n*: 0-15)	物理存储器属性配置寄存器	0xBC0+0x1**n*	R/W
pma_addr*n* (*n*: 0-15)	物理存储器属性地址寄存器	0xBD0+0x1**n*	R/W
总线错误异常上下文 CSR（自定义）			

名称	描述	地址	访问
ldpc0	加载总线错误 PC 寄存器 0	0xBE0	R/W
ldpc1	加载总线错误 PC 寄存器 1	0xBE1	R/W
ldtval0	加载总线错误访问地址寄存器 0	0xBE8	R/W
ldtval1	加载总线错误访问地址寄存器 1	0xBE9	R/W
stpc0	存储总线错误 PC 寄存器 0	0xBF0	R/W
stpc1	存储总线错误 PC 寄存器 1	0xBF1	R/W
stpc2	存储总线错误 PC 寄存器 2	0xBF2	R/W
sttval0	存储总线错误访问地址寄存器 0	0xBF8	R/W
sttval1	存储总线错误访问地址寄存器 1	0xBF9	R/W
sttval2	存储总线错误访问地址寄存器 2	0xBFA	R/W

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

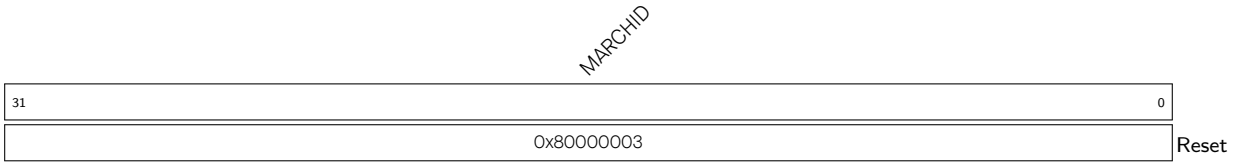
2.5.2 寄存器描述

Register 2.1. mvendorid (0xF11)



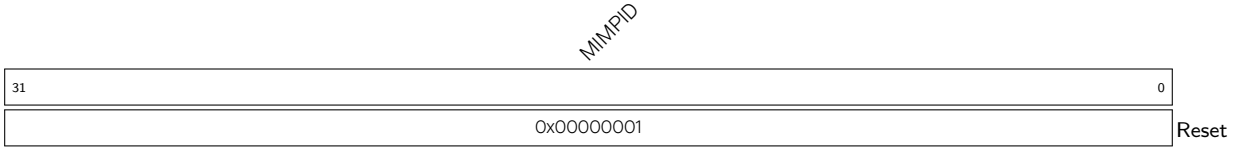
MVENDORID 表示供应商 ID。(RO)

Register 2.2. marchid (0xF12)



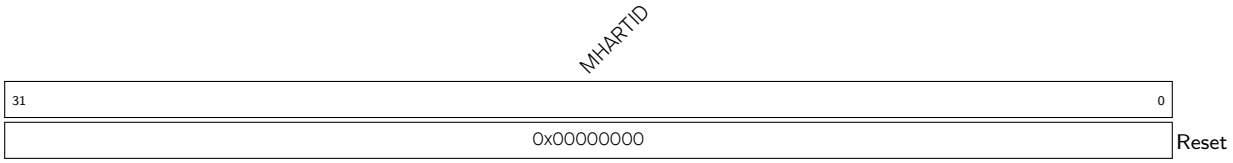
MARCHID 表示架构 ID。(RO)

Register 2.3. mimpid (0xF13)



MIMPID 表示实现 ID。(RO)

Register 2.4. mhartid (0xF14)



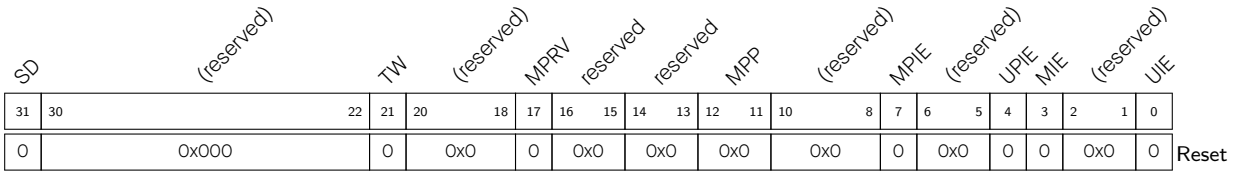
MHARTID 表示 hart ID。

0：hart 0

1: hart 1

(RO)

Register 2.5. mstatus (0x300)



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

TW 配置 WFI（等待中断）指令是否可以在较低特权级模式下执行。

0：WFI 可在较低特权级模式下执行。

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

(R/W)

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

0：不作为

1：作为

注意，此位不影响指令保护。

(R/W)

MPP 配置机器之前的特权模式（陷阱前）。

0x0：用户模式

0x3：机器模式

注意：只有低位可写。对高位的任何写入都将被忽略，因为它直接与低位相连。

(R/W)

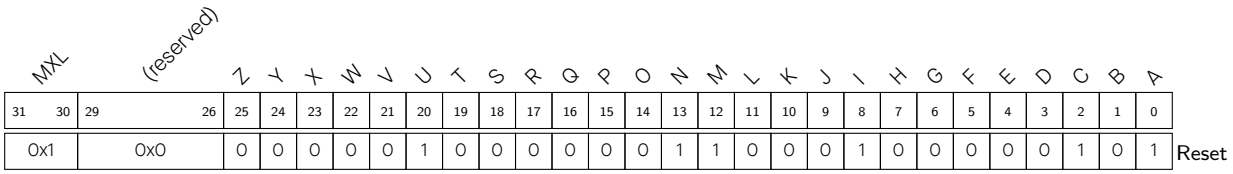
MPIE 写 1 使能（陷阱）之前的机器中断。(R/W)

UPIE 由于不支持用户模式中断，因此此位硬连线为 0。(RO)

MIE 写 1 使能全局机器模式中断。(R/W)

UIE 由于不支持用户模式中断，因此此位硬连线为 0。(RO)

Register 2.6. misa (0x301)



MXL 机器 XLEN = 1（32 位）。(RO)

Z 保留 = 0。(RO)

Y 保留 = 0。(RO)

X 非标准扩展 = 0。(RO)

W 保留 = 0。(RO)

V 保留 = 0。(RO)

U 实现用户模式 = 1。(RO)

T 保留 = 0。(RO)

S 实现监督模式 = 0。(RO)

R 保留 = 0。(RO)

Q 四精度浮点扩展 = 0。(RO)

P 保留 = 0。(RO)

O 保留 = 0。(RO)

N 支持用户级别中断 = 1。(RO)

M 整数乘除法标准扩展 = 1。(RO)

L 保留 = 0。(RO)

K 保留 = 0。(RO)

J 保留 = 0。(RO)

I RV32I 基本 ISA = 1。(RO)

H 虚拟机管理程序扩展 = 0。(RO)

G 其他标准扩展 = 0。(RO)

F 单精度浮点扩展 = 0。(RO)

E RV32E 基本 ISA = 0。(RO)

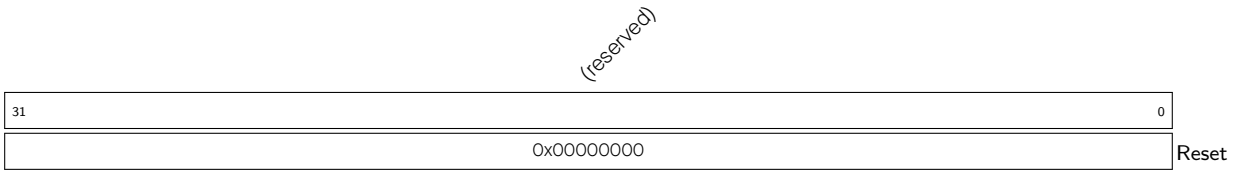
D 双精度浮点扩展 = 0。(RO)

C 压缩标准扩展 = 1。(RO)

B 保留 = 0。(RO)

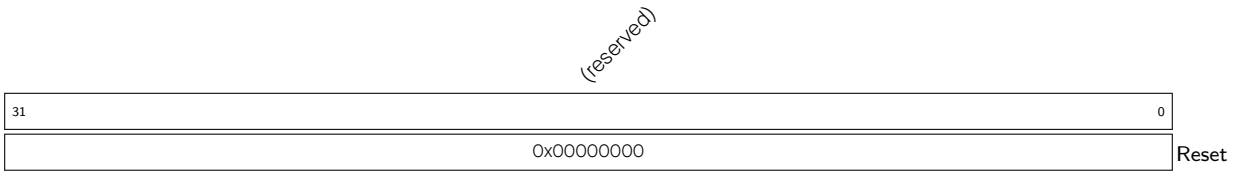
A 原子标准扩展 = 1。(RO)

Register 2.7. mideleg (0x303)



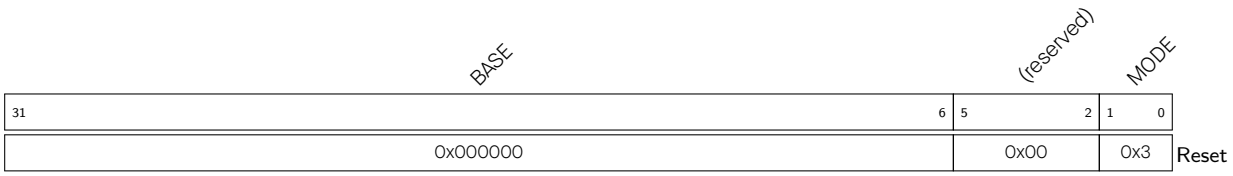
mideleg 所有位都硬连线为 0，因为仅支持 CLIC 模式。(RO)

Register 2.8. mie (0x304)



mie 所有位都硬连线为 0，因为仅支持 CLIC 模式。(RO)

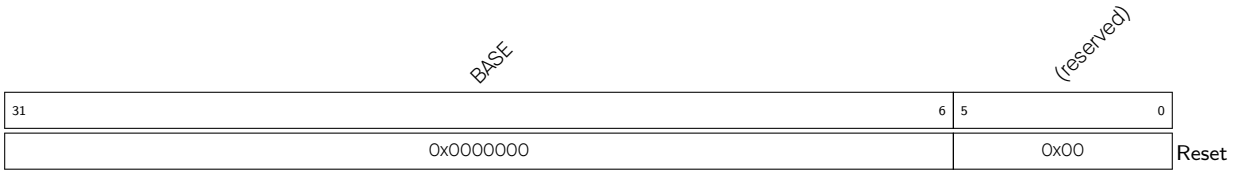
Register 2.9. mtvec (0x305)



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

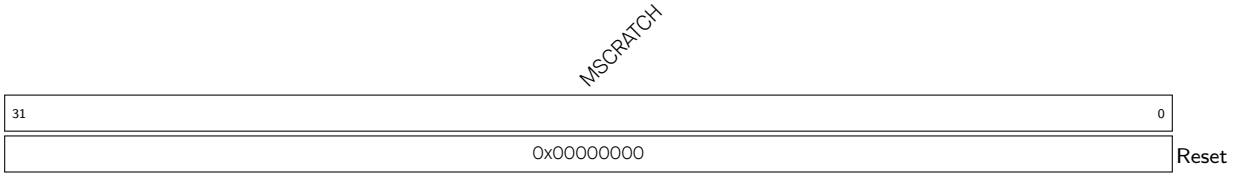
MODE 表示机器模式中断在 CLIC 模式还是向量/非向量 CLINT 模式运行。仅支持 CLIC 模式 **0x3**。(RO)

Register 2.10. mtvt (0x307)



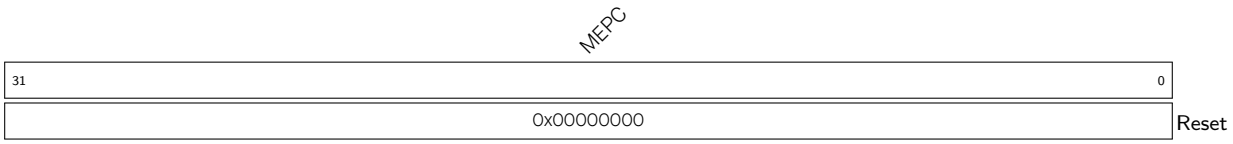
BASE 配置 CLIC 机器模式中断向量基地址的高 26 位，地址 64 字节对齐。(R/W)

Register 2.11. mscratch (0x340)



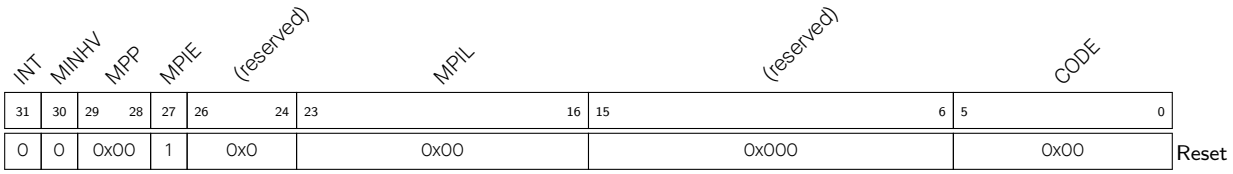
MSCRATCH 配置机器自定义的暂存信息。(R/W)

Register 2.12. mepc (0x341)



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

Register 2.13. mcause (0x342)



INT 表示 CPU 是否因中断进入陷阱。

0：因异常进入陷阱

1：因中断进入陷阱

(R/W)

MINHV 表示 mepc 是指令地址还是向量表表项地址。

1：向量表表项地址

0：指令地址

(R/W)

MPP 表示之前的特权模式，与 mstatus.MPP 相同。(R/W)

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

MPIL 表示之前的 8 位中断级别。(R/W)

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

0x01：指令访问错误

0x02：非法指令

0x03：硬件断点/观察点或 EBREAK

0x05：PMP 加载访问错误

0x06：存储地址或 AMO 地址未对齐

0x07：存储访问或 AMO 访问错误

0x08：用户模式环境调用 (ECALL)

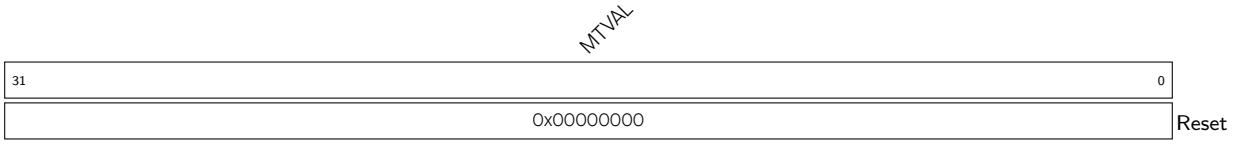
0x0b：机器模式环境调用

0x1f：PIE 指令非法其他值：保留

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

(R/W)

Register 2.14. mtval (0x343)



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

根据异常编号有以下解读：

0x1：指令虚拟地址错误

0x2：指令操作码错误

0x5：存储器读操作的数据地址错误

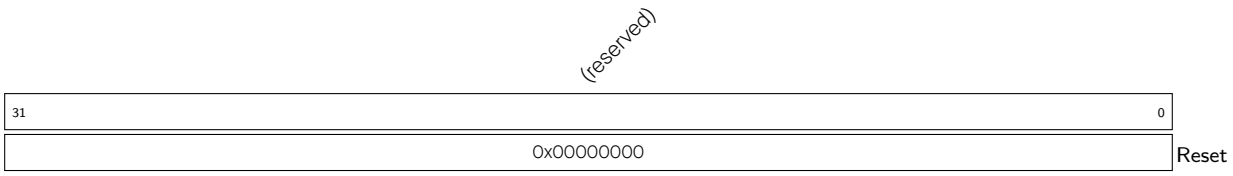
0x7：存储器写操作的数据地址错误

0x1F：PIE 非法指令操作码错误

注意：该寄存器不支持其他异常 ID 和中断。

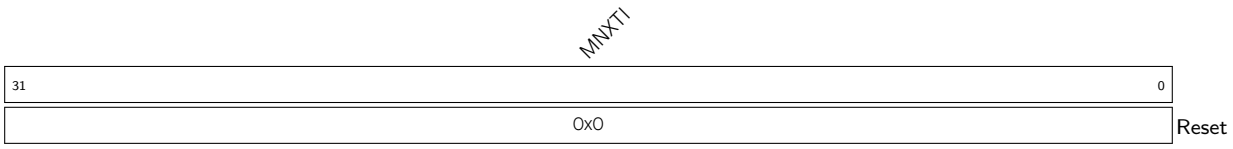
(R/W)

Register 2.15. mip (0x344)



MIP 所有位都硬连线为 0，因为仅支持 CLIC 模式。(RO)

Register 2.16. mnxti (0x345)



MNXTI 表示下一个机器模式中断入口地址，并配置中断使能状态。

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

此 CSR 使用 CSRRSI/CSRRCI 指令访问，读取的值是陷阱处理程序表中表项的指针，写回的是中断使能状态。此外，写入 mnxti 会产生副作用：更新中断上下文状态。这与普通的 CSR 指令不同，因为返回给软件的值与在读 □ 改 □ 写操作中实际使用的值并不相同。

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

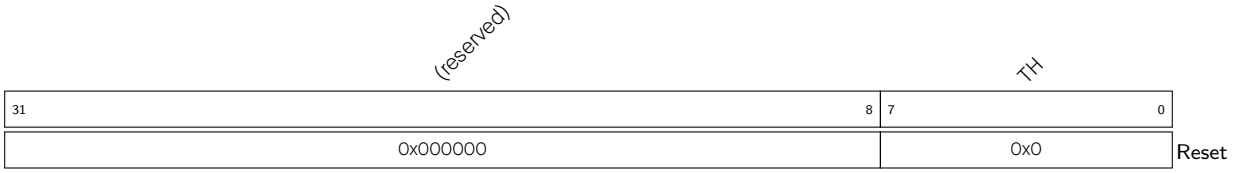
如果访问 mnxti 的 CSR 指令包含写操作，则用于读-改-写操作的 CSR 是 mstatus CSR，而 mcause 寄存器的 exccode 域和 mintstatus.MIL 域也可以更新为新的中断 ID 和级别。如果中断是边沿触发，则挂起位也会被清零。

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

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

(R/W)

Register 2.17. mintthresh (0x347)

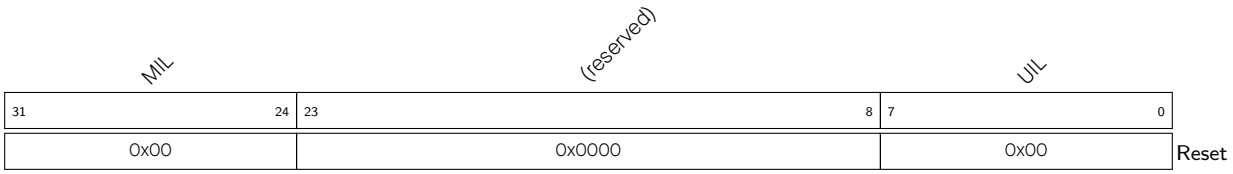


TH 配置机器模式中断的 8 位级别阈值。

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

(R/W)

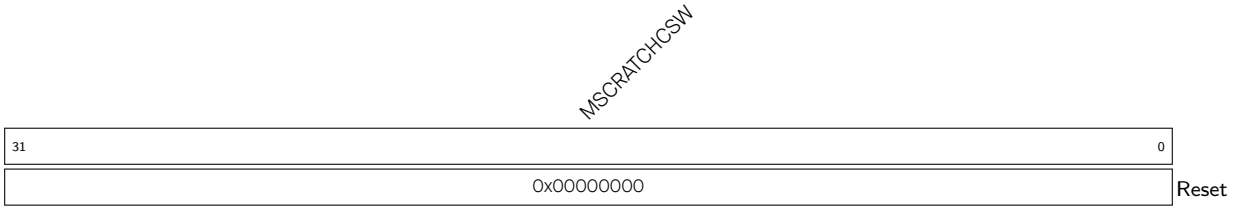
Register 2.18. mintstatus (0xFB1)



MIL 表示当前机器模式中断有效的 8 位中断级别。(RO)

UIL 硬连线为 0x0，因为不支持用户模式中断。(RO)

Register 2.19. mscratchcsw (0x348)



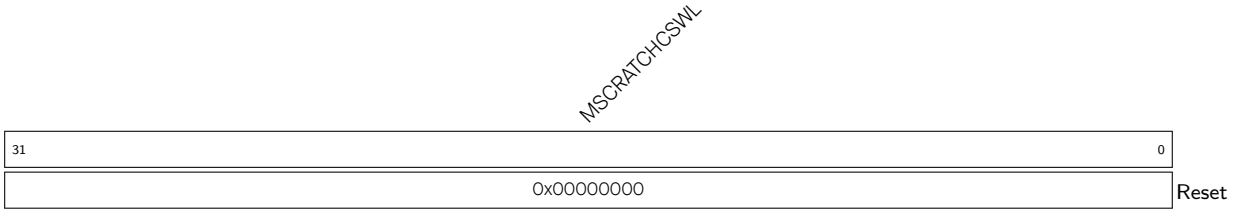
MSCRATCHCSW 配置 MSCRATCH 的值，MSCRATCH 的值取决于当前特权模式和 MPP 中之前的特权模式是否不同，如果不同，交换其 MSCRATCH 和 RS1 的值。

MSCRATCHCSW 是条件暂存交换 CSR，可以在单条指令中根据特权模式是否变化来决定是否交换暂存值。

使用 CSRRW 指令访问此 MSCRATCHCSW 时，如果 MPP 与当前特权模式不同，将 MSCRATCH 的值写入 RD，将 RS1 原来的值写入 MSCRATCH；如果相同，将 RS1 的值写入 RD。

(R/W)

Register 2.20. mscratchcswl (0x349)



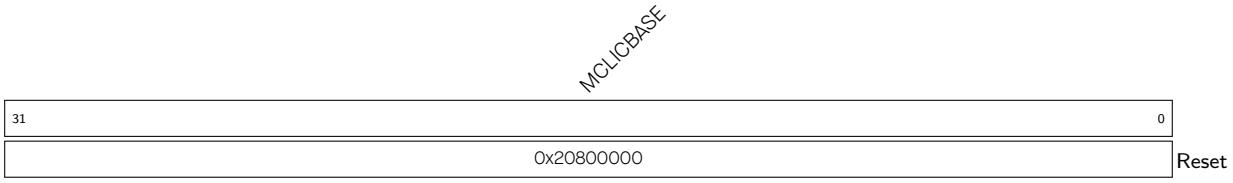
MSCRATCHCSWL 配置 MSCRATCH 的值，MSCRATCH 的值取决于 mintstatus.MIL 中的当前中断级别和 mcause.MPIL 中的之前中断级别是否不同，如果不同，交换 MSCRATCH 与 RS1 的值。

MSCRATCHCSWL 是条件暂存交换 CSR，可以在单条指令中根据中断级别是否变化来决定是否交换暂存值。

使用 CSRRW 指令访问 MSCRATCHCSWL 时，如果 mcause.MPIL 和 mintstatus.MIL 中只有一个等于 0，将 MSCRATCH 的值写入 RD，将 RS1 原来的值写入 MSCRATCH；如果是其他情况，将 RS1 写入 RD。

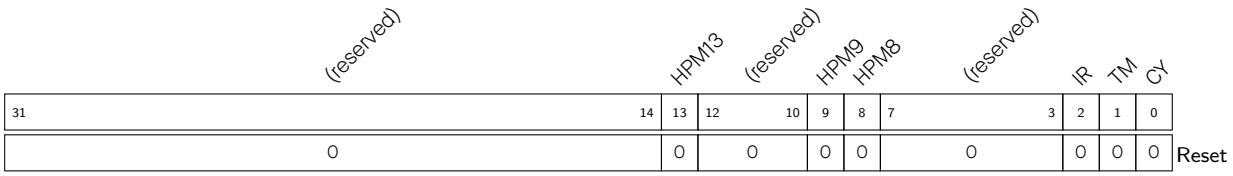
(R/W)

Register 2.21. mclicbase (0x350)



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

Register 2.22. mcounteren (0x306)



HPM13 配置是否从用户模式访问 hpmcounter13。

0：从用户模式访问 hpmcounter13 会导致非法指令异常

1：从用户模式访问有效

(R/W)

HPM9 配置是否从用户模式访问 hpmcounter9。

0：从用户模式访问 hpmcounter9 会导致非法指令异常

1：从用户模式访问有效

(R/W)

HPM8 配置是否从用户模式访问 hpmcounter8。

0：从用户模式访问 hpmcounter8 会导致非法指令异常

1：从用户模式访问有效

(R/W)

CY 配置从用户模式访问 cycle 计数器。

0：在用户模式下访问 cycle CSR 将生成非法指令异常

1：在用户模式下访问 cycle CSR 是有效的

(R/W)

TM 配置从用户模式访问 time 计数器。

0：在用户模式下访问 time CSR 将生成非法指令异常

1：在用户模式下访问 time CSR 是有效的

(R/W)

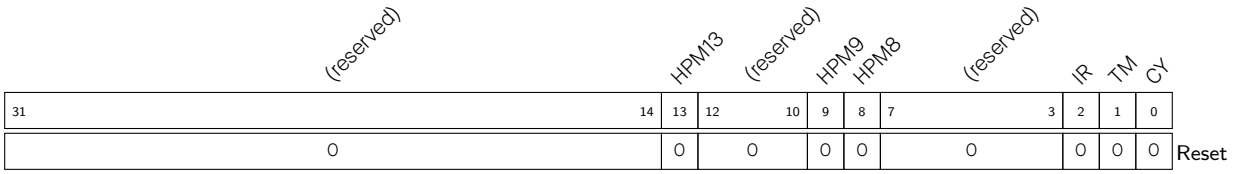
IR 配置从用户模式访问 instret 计数器。

0：在用户模式下访问 instret CSR 将生成非法指令异常

1：在用户模式下访问 instret CSR 是有效的

(R/W)

Register 2.23. mcountinhibit (0x320)



HPM13 配置 mhpmcounter13 的递增。

0：继续递增 mhpmcounter13

1：停止递增 mhpmcounter13

(R/W)

HPM9 配置 mhpmcounter9 的递增。

0：继续递增 mhpmcounter9

1：停止递增 mhpmcounter9

(R/W)

HPM8 配置 mhpmcounter8 的递增。

0：继续递增 mhpmcounter8

1：停止递增 mhpmcounter8

(R/W)

IR 配置 instret 计数器的递增。

0：继续递增 instret 计数器

1：停止递增 instret 计数器

(R/W)

TM 配置 time 计数器的递增。

0：继续递增 time 计数器

1：停止递增 time 计数器

(R/W)

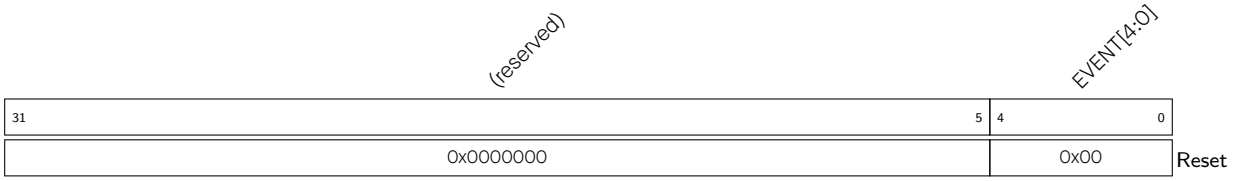
CY 配置 cycle 计数器的递增。

0：继续递增 cycle 计数器

1：停止递增 cycle 计数器

(R/W)

Register 2.24. mhpmevent8 (0x328)

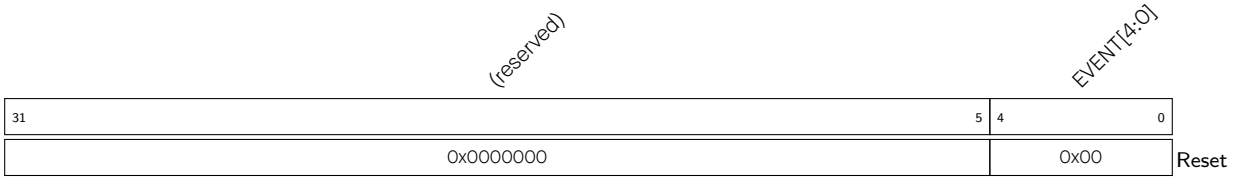


EVENT[4:0] mhpmcounter8 的事件选择器。

此计数器的唯一有效事件值为 0x6，用于条件分支预测错误。

(R/W)

Register 2.25. mhpmevent9 (0x329)

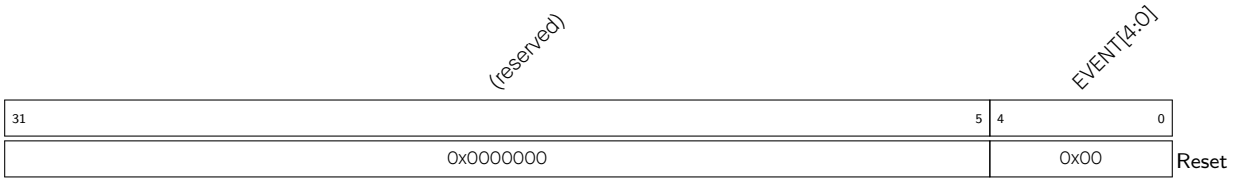


EVENT[4:0] mhpmcounter9 的事件选择器。

此计数器的唯一有效事件值为 0x7，用于条件分支指令。

(R/W)

Register 2.26. mhpmevent13 (0x32D)

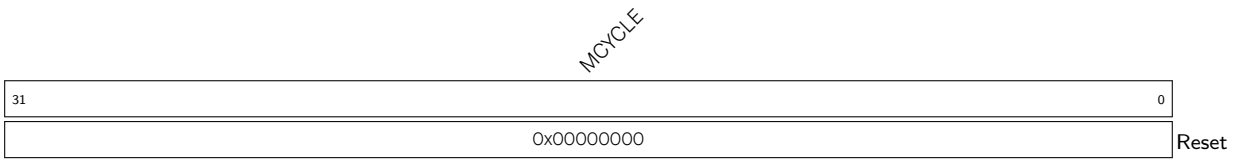


EVENT[4:0] mhpmcounter13 的事件选择器。

此计数器的唯一有效事件值为 0xB，用于存储指令。

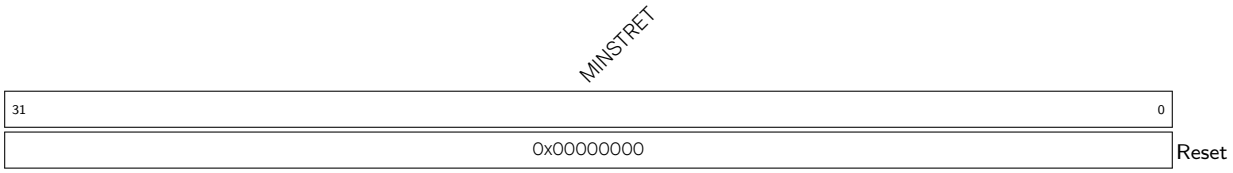
(R/W)

Register 2.27. mcycle (0xB00)



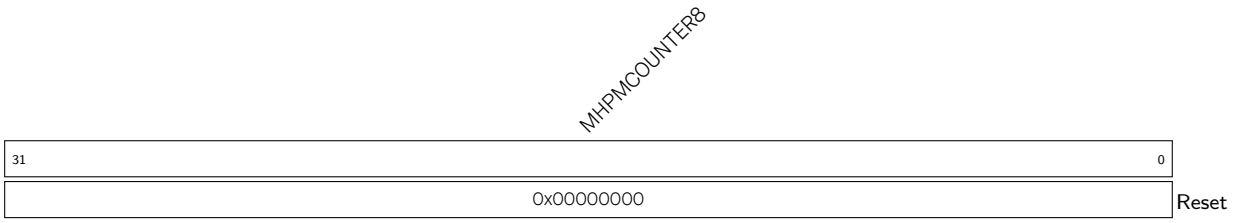
MCYCLE 表示机器模式周期计数器的低 32 位。(R/W)

Register 2.28. minstret (0xB02)



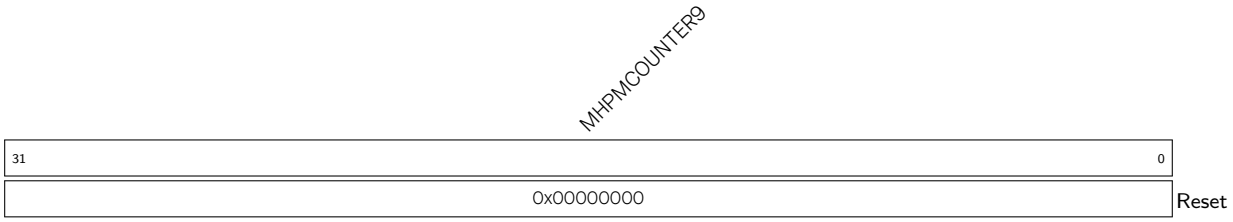
MINSTRET 表示退役机器指令计数器的低 32 位。(R/W)

Register 2.29. mhpmcounter8 (0xB08)



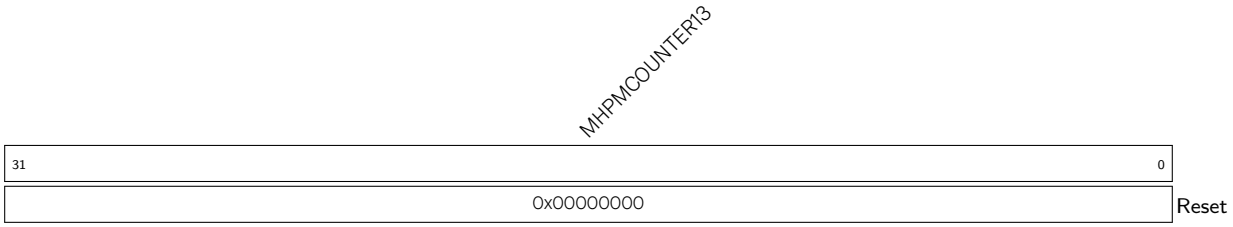
MHPMCOUNTER8 表示机器性能监控计数器 8 的低 32 位。(R/W)

Register 2.30. mhpmcounter9 (0xB09)



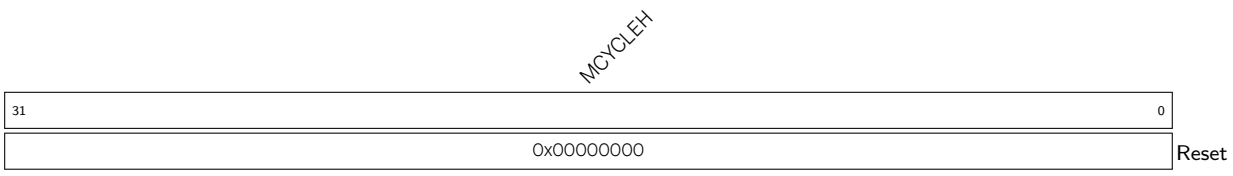
MHPMCOUNTER9 表示机器性能监控计数器 9 的低 32 位。(R/W)

Register 2.31. mhpmcounter13 (0xB0D)



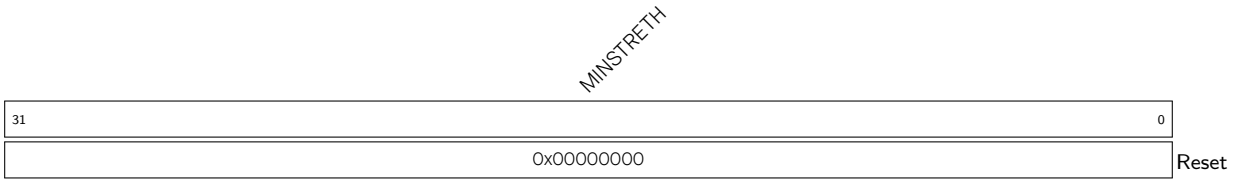
MHPMCOUNTER13 表示机器性能监控计数器 13 的低 32 位。(R/W)

Register 2.32. mcycleh (0xB80)



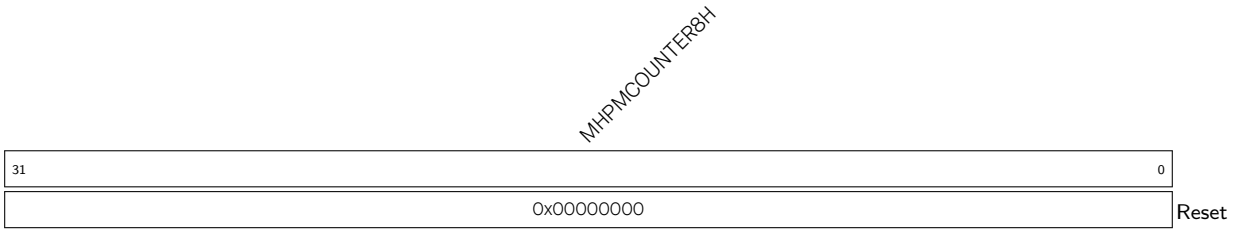
MCYCLEH 表示 mcycle 计数器的高 32 位。(R/W)

Register 2.33. minstreth (0xB82)



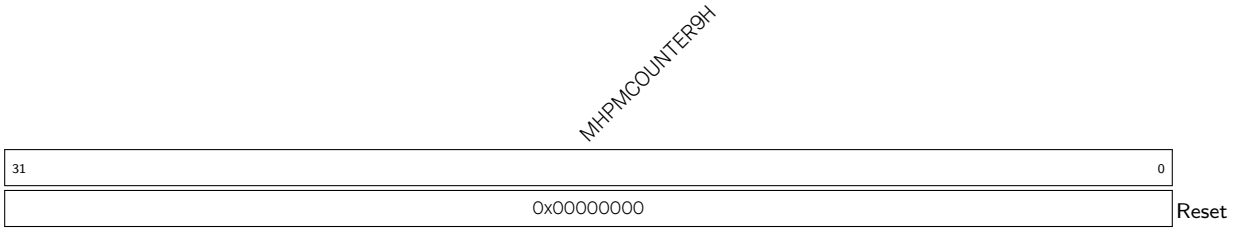
MINSTRETH 表示 minstret 计数器的高 32 位。(R/W)

Register 2.34. mhpmcounter8h (0xB88)



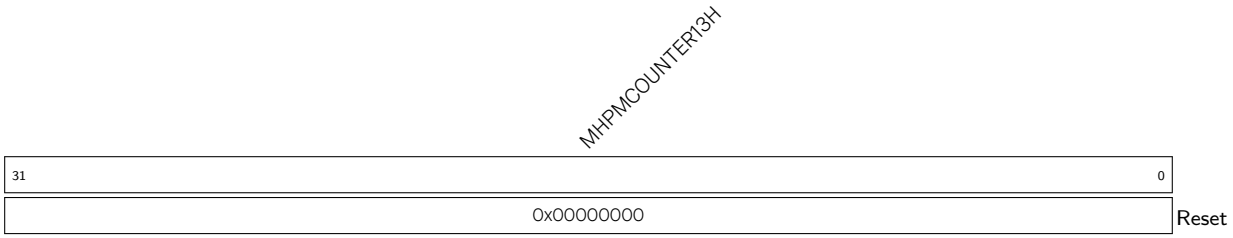
MHPMCOUNTER8H 表示 mhpmcounter8 计数器的高 32 位。(R/W)

Register 2.35. mhpmcounter9h (0xB89)



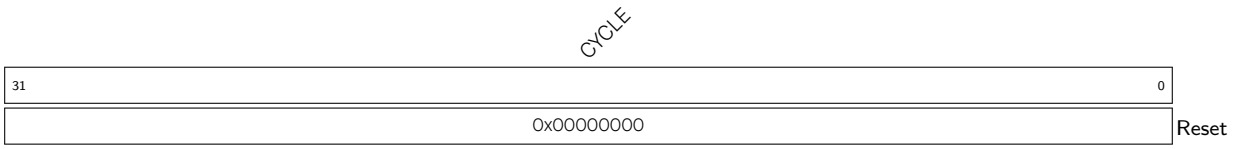
MHPMCOUNTER9H 表示 mhpmcounter9 计数器的高 32 位。(R/W)

Register 2.36. mhpmcounter13h (0xB8D)



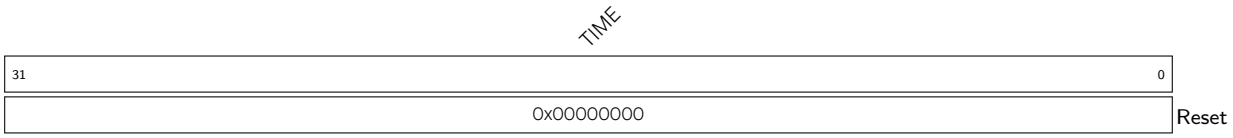
MHPMCOUNTER13H 表示 mhpmcounter13 计数器的高 32 位。(R/W)

Register 2.37. cycle (0xC00)



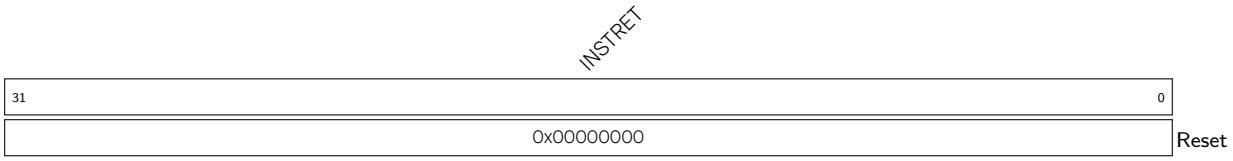
CYCLE mcycle 的只读镜像。(RO)

Register 2.38. time (0xC01)



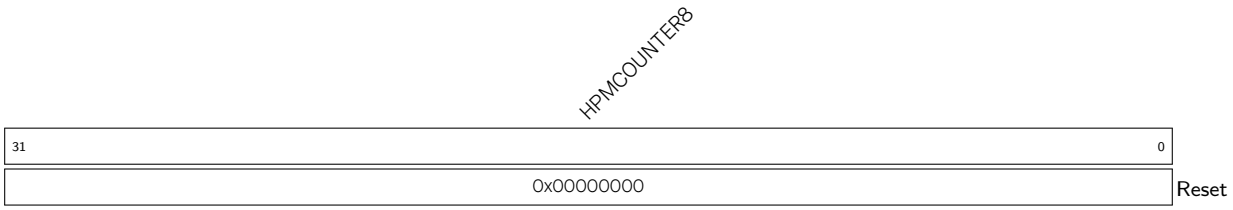
TIME 表示 RDTIME 指令的定时器值。(RO)

Register 2.39. instret (0xC02)



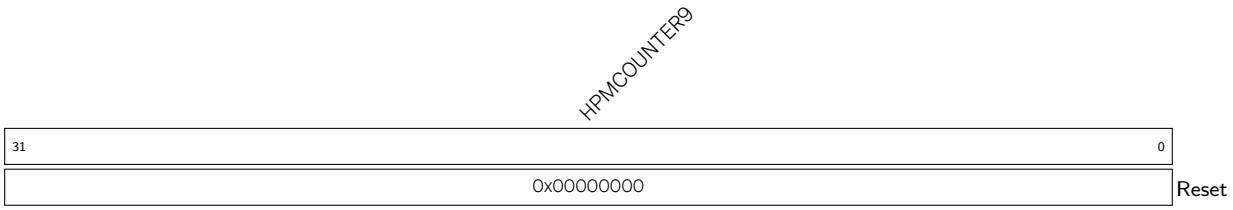
INSTRET minstret 的只读镜像。(RO)

Register 2.40. hpmcounter8 (0xC08)



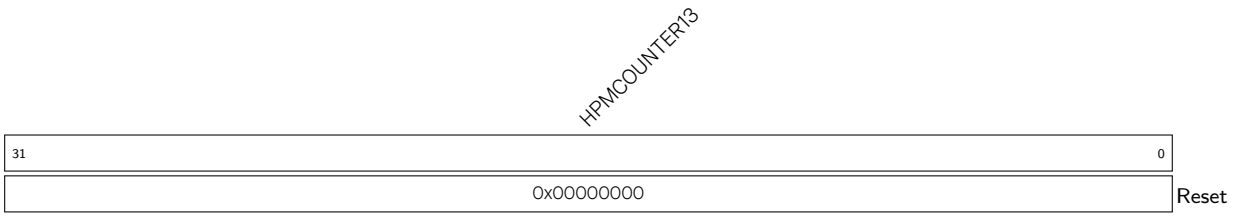
HPMCOUNTER8 mhpmcounter8 的只读镜像。(RO)

Register 2.41. hpmcounter9 (0xC09)



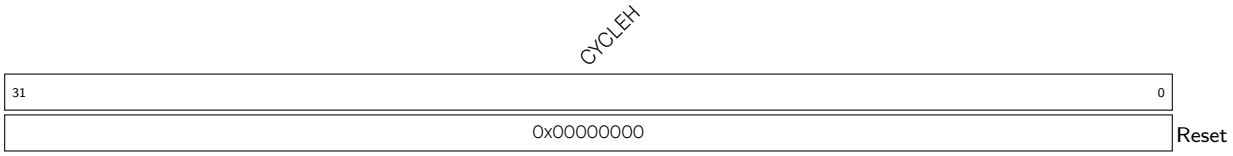
HPMCOUNTER9 mhpmcounter9 的只读镜像。(RO)

Register 2.42. hpmcounter13 (0xC0D)



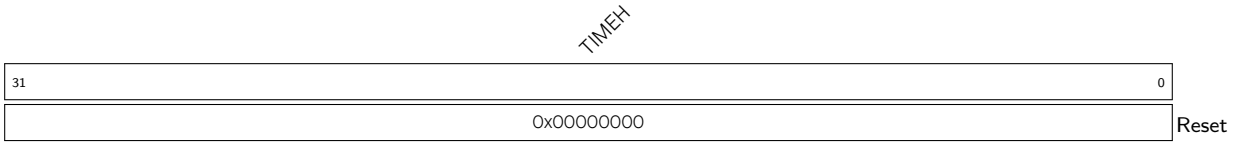
HPMCOUNTER13 mhpmcounter13 的只读镜像。(RO)

Register 2.43. cycleh (0xC80)



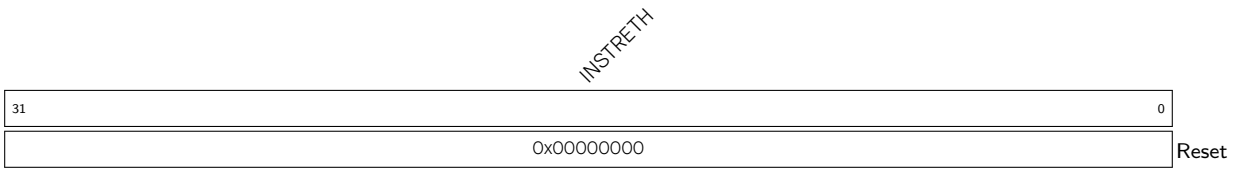
CYCLEH mcycleh 的只读镜像。(RO)

Register 2.44. timeh (0xC81)



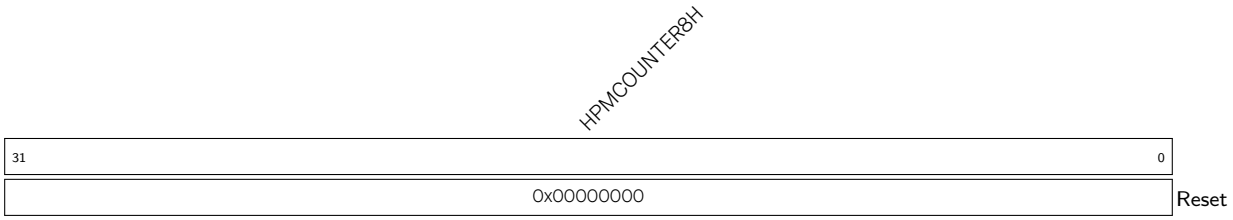
TIMEH 表示 RDTIME 指令定时器的高 32 位。(RO)

Register 2.45. instreth (0xC82)



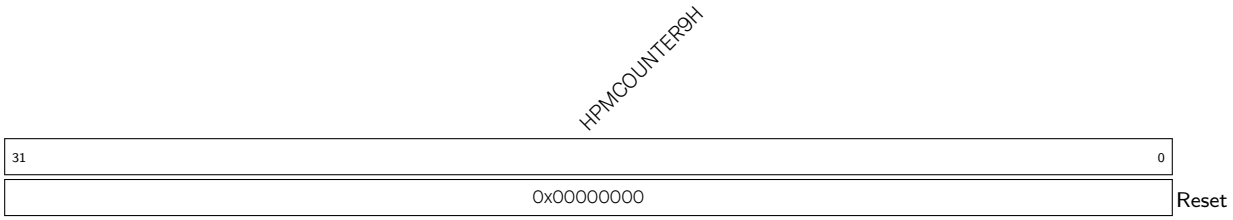
INSTRETH minstreth 的只读镜像。(RO)

Register 2.46. hpmcounter8h (0xC88)



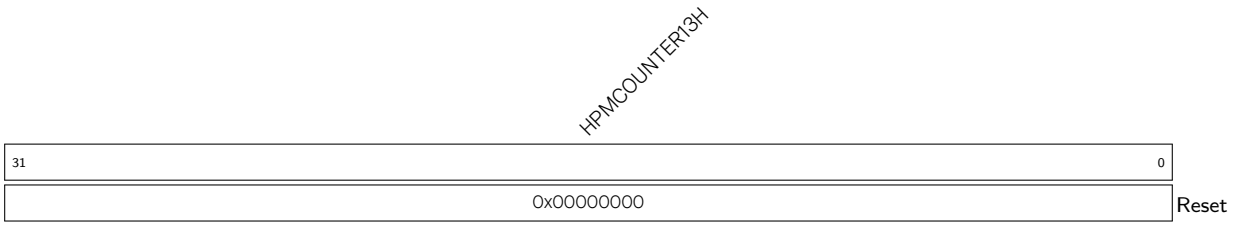
HPMCOUNTER8H mhpmcounter8h 的只读镜像。(RO)

Register 2.47. hpmcounter9h (0xC89)



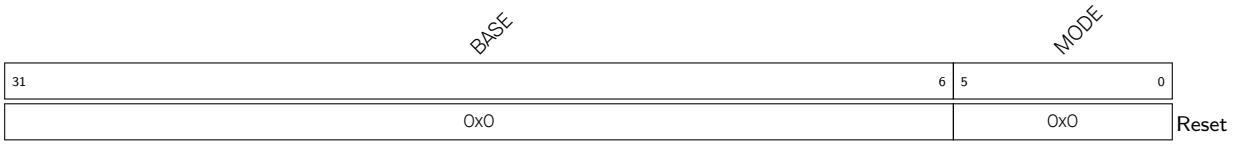
HPMCOUNTER9H mhpmcounter9h 的只读镜像。(RO)

Register 2.48. hpmcounter13h (0xC8D)



HPMCOUNTER13H mhpmcounter13h 的只读镜像。(RO)

Register 2.49. jvt (0x017)



BASE 表示跳转表基地址。(R/W)

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

0x0：JVT_BASE 用作跳转表。

其他：保留，供将来使用

(R/W)

Register 2.50. mexstatus (0x7E1)



EXPT_VLD 进入陷阱处理程序时，自动设置此位。

此位仅可从调试模式写入。

(RO)

LOCKUP 当两个异常发生时，自动设置此位。

一旦锁定，核心将暂停运行，并等待调试器处理锁定。

此位仅可从调试模式写入。

(RO)

BUS_ERR 由于总线错误异常进入加载/存储错误陷阱时，自动设置此位。

一旦进入陷阱处理程序，必须读取此位，确定加载/存储错误是否由总线错误造成。

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

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

(R/W)

PBEXE 配置是否启用挂起的总线错误异常。

0：禁用

1：启用

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

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

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

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

(R/W)

PPBEXE 配置是否启用之前挂起的总线错误异常。

0：禁用

1：启用

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

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

(R/W)

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

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

MRET 时此位自动清除。

(R/W)

见下页……

Register 2.50. mexstatus (0x7E1)

接上页……

FETCH_ERR 由于总线错误异常进入取指错误陷阱时，自动设置此位。

一旦进入陷阱处理程序，必须读取此位，确定取指错误是否由总线错误造成。

通过 MRET 退出陷阱时，此位自动清除。

(R/W)

NMFT 配置在进入陷阱后是否禁用存储器屏障。

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

0：CPU 等待挂起的存储器加载/存储操作完成后，再进入陷阱处理程序。

(R/W)

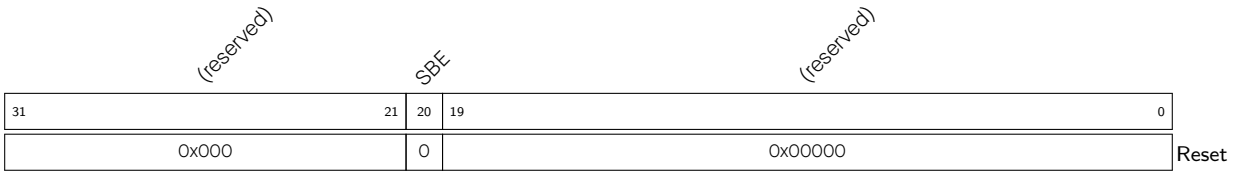
CLIC_INHV 配置是否启用恢复错误向量表取指。

0：禁用

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

(R/W)

Register 2.51. mhint (0x7C5)



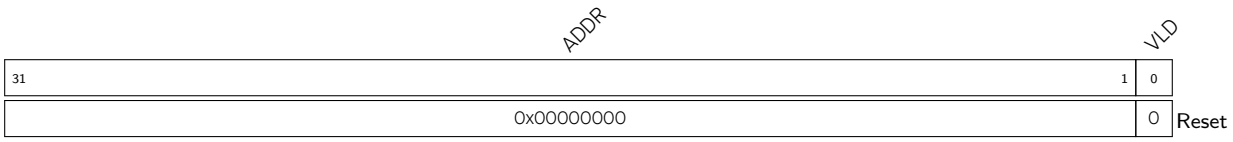
SBE 配置是否启用同步加载/存储总线错误异常。

0：禁用

1：启用

(R/W)

Register 2.52. ldpc0 (0xBE0)



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

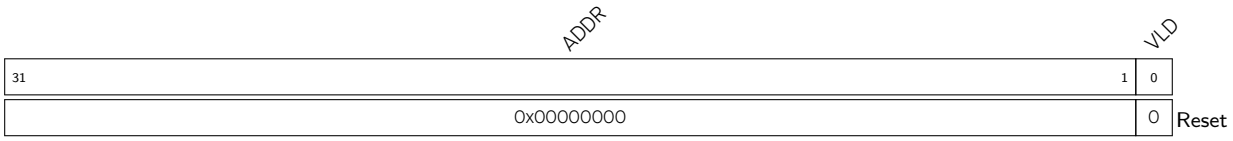
0：无效

1：有效

(R/W)

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

Register 2.53. ldpc1 (0xBE1)



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

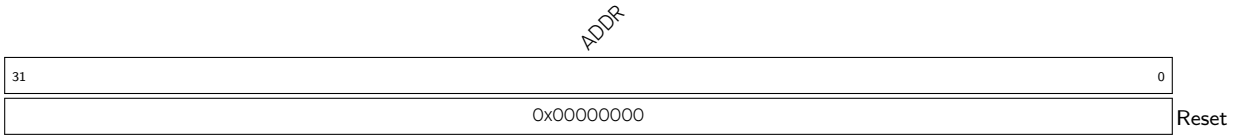
0：无效

1：有效

(R/W)

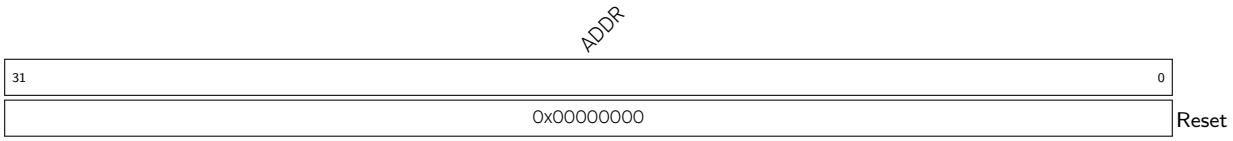
ADDR 配置与加载总线错误异常对应的程序计数器的高 31 位，数值应半字对齐。(R/W)

Register 2.54. ldtval0 (0xBE8)



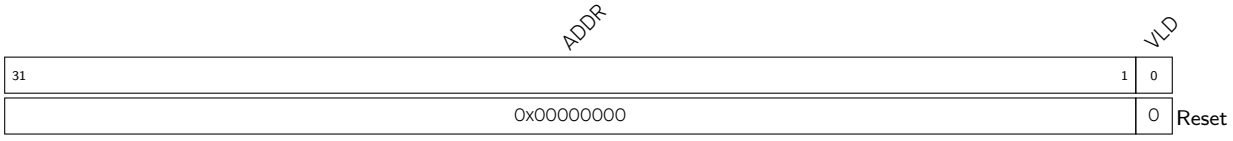
ADDR 配置与 ldpc0 中记录的加载总线错误异常对应的访问地址。(R/W)

Register 2.55. ldtval1 (0xBE9)



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

Register 2.56. stpc0 (0xBF0)



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

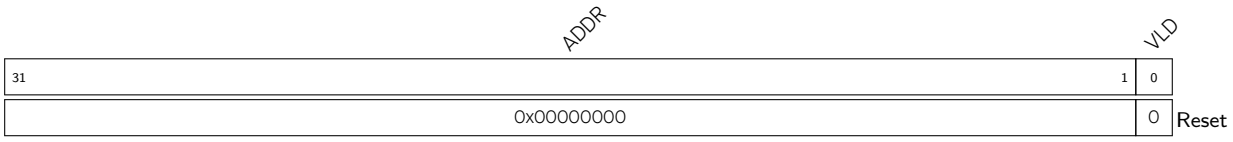
0：无效

1：有效

(R/W)

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

Register 2.57. stpc1 (0xBF1)



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

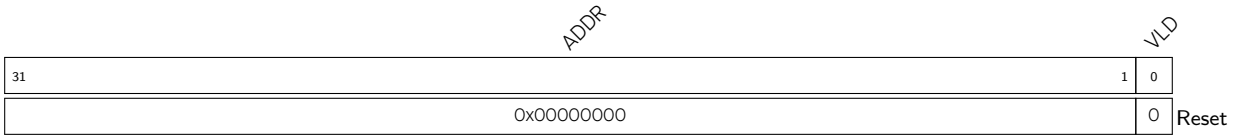
0：无效

1：有效

(R/W)

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

Register 2.58. stpc2 (0xBF2)



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

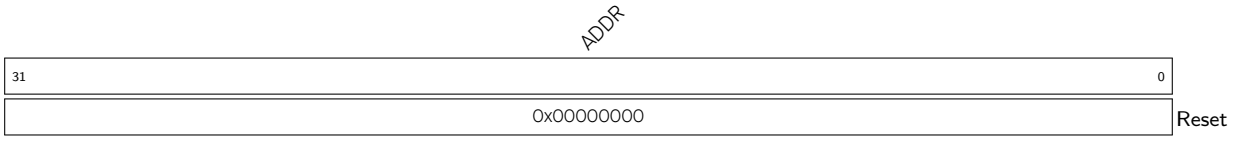
0：无效

1：有效

(R/W)

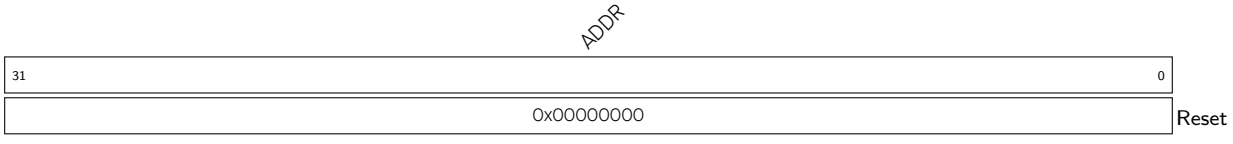
ADDR 配置与存储总线错误异常对应的程序计数器的高 31 位，数值应半字对齐。(R/W)

Register 2.59. sttval0 (0xBF8)



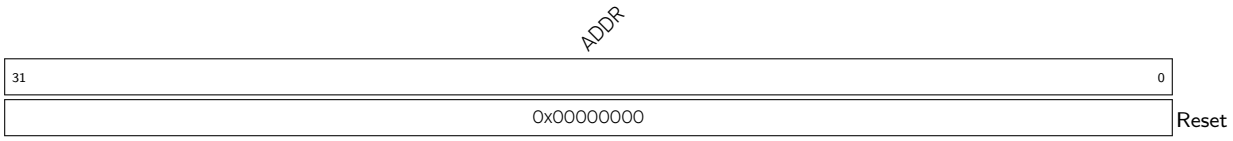
ADDR 配置与 stpc0 中记录的存储总线错误异常对应的访问地址。(R/W)

Register 2.60. sttval1 (0xBF9)



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

Register 2.61. sttval2 (0xBFA)



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

2.6 RISC-V 标准 ISA 扩展支持

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

2.6.1 乘法/除法 (M) 扩展

2.6.1.1 概述

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

2.6.1.2 功能描述

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

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

2.6.2 压缩 (C) 扩展

2.6.2.1 概述

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

2.6.2.2 功能描述

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

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

2.6.3 代码体积优化 (Zc) 扩展

2.6.3.1 概述

Zc 扩展有助于减少代码量，包含多个子集扩展。ESP32-C5 HP 核心实现的 Zc 扩展与 《Zc 规范 v1.0.4-3》 兼容。Zc 规范包含多个子集扩展，ESP32-C5 HP CPU 核心实现的是 Zcmt 扩展。

2.6.3.2 功能描述

Zcmt 扩展

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

表跳转使用指令存储器中的 256 项 32 位宽的表来存储函数地址。表必须至少 64 字节对齐。作为字典压缩的一种形式，表跳可减少 jal、auipc+jalr、jr、auipc+jr 指令的代码大小。表跳转允许链接器用 cm.jalt 编码和表中的一个条目替换以下指令序列：

- 32 位 j 调用
- 32 位 jal ra 调用

2.6.3.3 Zc 指令

指令	助记符	描述
Zcmt 扩展		
cm.jt	cm.jt index	从存储器中的跳转向量表中读取一个条目并跳转到读取的地址
cm.jalt	cm.jalt index	从存储器中的跳转向量表中读取一个条目并跳转到读取的地址，链接到 ra

有关指令的更多详细信息，请参阅 《Zc 规范 v1.0.4-3》 。

2.6.3.4 限制

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

2.6.4 原子 (A) 扩展

2.6.4.1 概述

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

原子指令目前忽略 aq（获取）和 rl（释放）位，因为这些位对于目前这种可以保证存储器排序的架构无关。

2.6.4.2 功能描述

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

2.6.4.3 加载保留字 (LR.W) 指令

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

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

如果发生上述任何一种情况，存储器锁将立刻被释放。如果遇到 SC.W 指令，则存储器锁不会立即被释放，但最终会被释放，有关具体如何释放的信息，请参考章节 2.6.4.4。

如果地址未对齐，则会导致异常 mcause = 6。

2.6.4.4 条件存入字 (SC.W) 指令

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

如果锁已失效（失效原因见章节 2.6.4.3），SC.W 指令将在目标寄存器 rd 中设置失败代码，目前只能设置为 1。

如果地址未对齐，则会导致异常 mcause = 6。

2.6.4.5 AMO 指令

AMO (原子性存储器操作) 指令分三步执行：

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

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

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

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

2.7 存储器映射寄存器

2.7.1 概述

核心系统中存在以下紧密耦合的存储器映射寄存器：

- CLIC 寄存器
- CLINT 寄存器

2.7.2 特性

HP CPU 核心支持：

- 可访问 CLIC 寄存器
- 访问无效地址时生成总线错误异常
- 访问 CLINT 寄存器

2.7.3 功能描述

下表为 HP 核心用于访问 CLIC 和 CLINT 地址范围的地址解码。

表 2.7-1. 存储器映射寄存器访问的地址解码

位	描述
31:28	固定为 0x2，即基地址
27:20	标识子模块： 0x0：CLINT 0x8：CLIC 0xB：CLIC 用户 其他：保留
19:16	控制访问类型 0x0：访问本核心的寄存器 0x1：保留
15:0	寄存器偏移

表 2.7-2. 内部存储器地址映射

块	起始地址	结束地址
CLINT	0x20000000	0x2000FFFF
CLIC	0x20800000	0x2080FFFF

2.8 中断控制器

2.8.1 概述

HP 核心使用 CLIC + CLINT 方案来实现中断支持。

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

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

2.8.2 CLIC

2.8.2.1 概述

HP 核心中实现的 CLIC 遵循 RISC-V 国际基金会的提案：核心本地中断控制器 (CLIC) RISC-V 特权架构扩展。在撰写本节时，官方提案尚未获得批准，因此，以下规范可能与批准的提案有所不同。

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

仅支持机器模式中断。

2.8.2.2 CLIC CSRs

以下机器模式 CSR 与 CLIC 方案中的中断处理相关：

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

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

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

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

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

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

2.8.2.3 中断映射

HP 核心中的 CLIC 支持 32 个外部中断和 2 个核心本地中断。

ID 16-47 映射到外部中断，而 ID 15 及以下用于核心本地中断。

表 2.8-1. CLIC 中断映射

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

有关核心本地中断源的详细描述，请参阅章节 2.8.3。

32 个外部中断到 SoC 中断源的映射由中断矩阵管理。有关如何配置和映射到 HP 核心的 CLIC 中断的详细信息，请参阅章节 11 中断矩阵。

2.8.2.4 CLIC 参数

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

- mclicbase 为只读 CSR，用于保存机器模式下 CLIC 存储器映射寄存器的基地址。
- clicinfo 为存储器映射只读寄存器，包含以下域：
  - NUM_INTERRUPTS：表示当前 CLIC 实现支持的中断数量
  - CLICINTCTLBITS：表示 clicintctl[i] 和 mintthresh.TH 中用于编码每个中断的级别和优先级的可编程高位数量。
- 假设 mclicbase 已知，与 clicinfo 对应的地址偏移量 0x0004 相加（参考章节 2.8.2.6），即可计算出 clicinfo 的完整地址。

2.8.2.5 中断级别和优先级编码

表 2.8-2 显示了配置 CLIC 中断级别和优先级的各种方式，具体取决于 mcliccfg.MNLBITS 和 clicintctl[i] 寄存器的配置的值，以及 CLICINTCTLBITS 参数的固定值 (0x3)。

表 2.8-2. CLIC 级别和优先级编码 (CLICINTCTLBITS=3)

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

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

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

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

2.8.2.6 CLIC 存储器映射寄存器列表

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

请注意，clicintip[i]、clicintie[i]、clicintattr[i] 和 clicintctl[i] 为单字节寄存器，对这些寄存器进行字和半字对齐访问是原子的。请注意，mintthresh 是机器模式 CSR，不是存储器映射寄存器。

机器模式 CLIC 寄存器

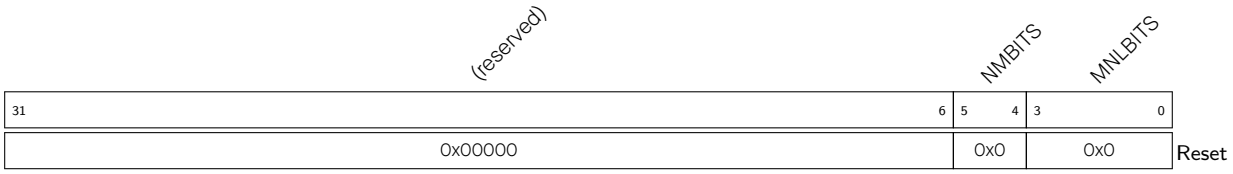
名称	描述	MCLICBASE 偏移	绝对地址	访问
mcliccfg	CLIC 机器模式全局配置寄存器	0x0000	0x20800000	R/W
clicinfo	CLIC 信息寄存器	0x0004	0x20800004	RO
clicintip[3]	CLINT 软件中断挂起寄存器	0x100C	0x2080100C	R/W
clicintie[3]	CLINT 软件中断使能寄存器	0x100D	0x2080100D	R/W
clicintattr[3]	CLINT 软件中断属性寄存器	0x100E	0x2080100E	R/W
clicintctl[3]	CLINT 软件中断级别控制寄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

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

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

2.8.2.7 CLIC 存储器映射寄存器描述

Register 2.62. mcliccfg (0x20800000)



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

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

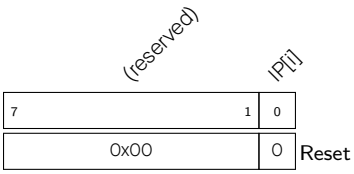
Register 2.63. clicinfo (0x20800004)



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

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

Register 2.64. clicintip[i] (0x20801000 + 4*i)



IP[i] 表示第 i 个 CLIC 机器模式中断的挂起状态。

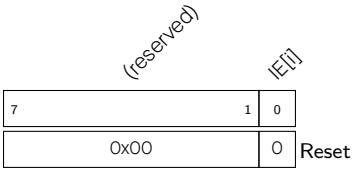
0：第 i 个中断未挂起

1：第 i 个中断挂起

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

(R/W)

Register 2.65. clicintie[i] (0x20801001 + 4*i)



IE[i] 配置是否使能第 i 个 CLIC 机器模式中断。

0：禁用

1：启用

(R/W)

Register 2.66. clicintattr[i] (0x20801002 + 4*i)



SHV 配置第 i 个 CLIC 机器模式中断的硬件向量化。

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

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

(R/W)

TRIG 配置第 i 个 CLIC 机器模式中断的触发类型和极性。

0x0：中断为正电平触发

0x1：中断为正边缘触发

0x2：中断为负电平触发

0x3：中断为负边缘触发

(R/W)

MODE 配置第 i 个 CLIC 中断的特权模式。

此值硬连线为 0x3，因为不支持用户模式中断。

(RO)

Register 2.67. clicintctl[i] (0x20801003 + 4*i)



CTL[i] 配置第 i 个 CLIC 机器模式中断的级别和优先级。

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

(R/W)

2.8.3 核心本地中断 (CLINT)

2.8.3.1 概述

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

表 2.8-4. 核心本地中断源

ID	描述
3	机器模式软件中断
7	机器模式定时器中断

2.8.3.2 特性

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

2.8.3.3 软件中断

机器模式软件中断通过设置或清除存储器映射寄存器 MSIP 来控制。

设置寄存器位 clicintie[3] 来在核心级别启用软件中断。

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

2.8.3.4 定时器计数器和中断

设置 mtimectl 寄存器的 mtime_EN 位启用定时器计数器。

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

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

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

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

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

2.8.3.5 寄存器列表

本小节的所有地址均为相对于 CPU 子系统基地址的地址偏移量（相对地址），CPU 子系统基地址为 0x20000000。

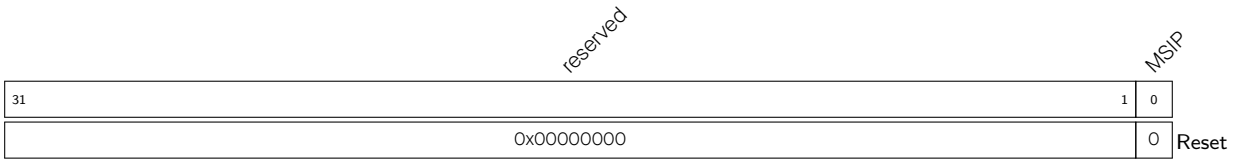
名称	描述	地址	访问
msip	核心本地机器软件中断挂起寄存器	0x0000	R/W
mtimecmplo	核心本地机器定时器比较值的低 32 位	0x4000	R/W

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

2.8.3.6 寄存器描述

本小节的所有地址均为相对于 CPU 子系统基地址的地址偏移量（相对地址），CPU 子系统基地址为 0x20000000。

Register 2.68. msip (0x0000)



MSIP 配置机器软件中断的挂起状态。

0：未挂起

1：挂起

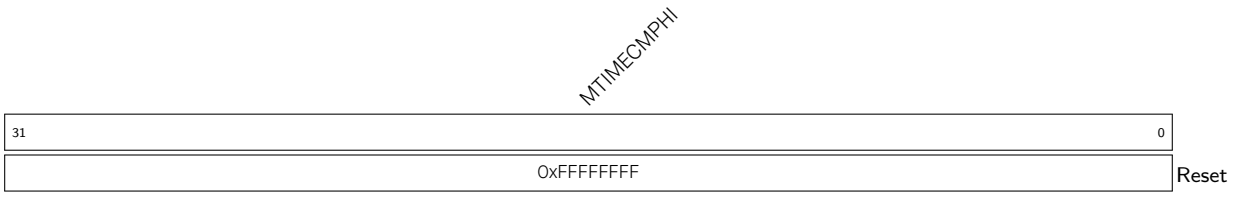
(R/W)

Register 2.69. mtimecmplo (0x4000)



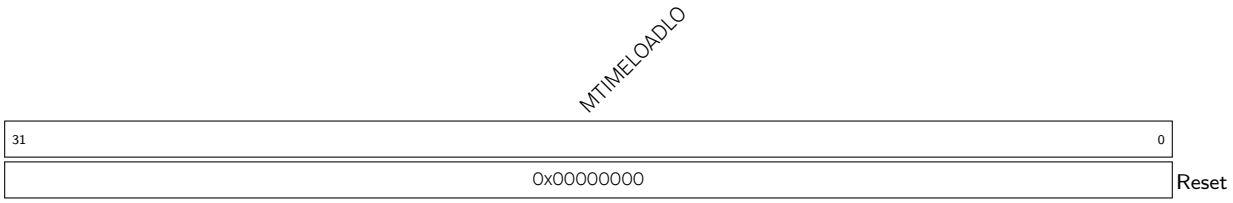
MTIMECMPLO 表示与系统计数器低 32 位进行比较的值。(R/W)

Register 2.70. mtimecmphi (0x4004)



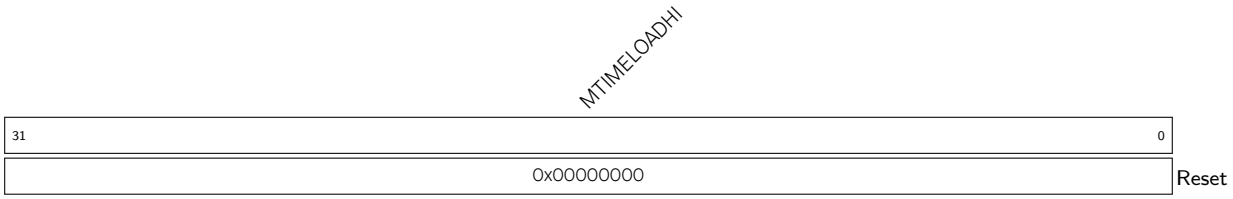
MTIMECMPHI 表示与系统计数器高 32 位进行比较的值。(R/W)

Register 2.71. mtimeloadlo (0x4008)



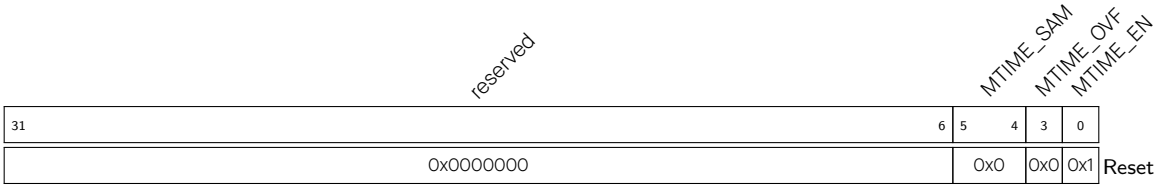
MTIME_LOADLO 表示要加载到系统计数器的低 32 位。(R/W)

Register 2.72. mtimeloadhi (0x400C)



MTIME_LOADHI 表示要加载到系统计数器的高 32 位。(R/W)

Register 2.73. mtimectl (0x4010)



MTIME_EN 配置是否启用系统计数器。

1：启用

0：禁用

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

(R/W)

MTIME_OVF 配置系统计数器的溢出位。

1：表示系统计数器值为最大值，即 0xFFFFFFFFFFFFFFFF

0：其他情况

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

(R/W)

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

0x0：无采样（默认）

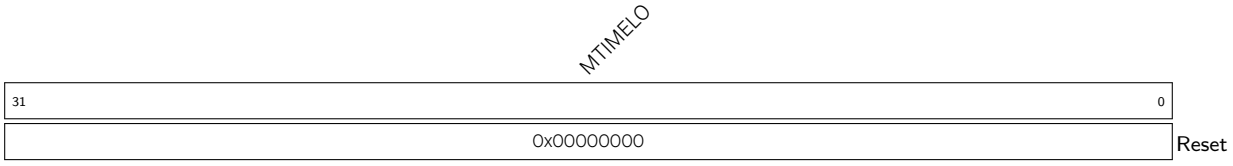
0x1：读取 MTIMELO 时采样系统计数的高 32 位

0x2：读取 MTIMEHI 时采样系统计数的低 32 位

0x3：读取 MTIMELO 或 MTIMEHI 时，采样系统计数的另一半

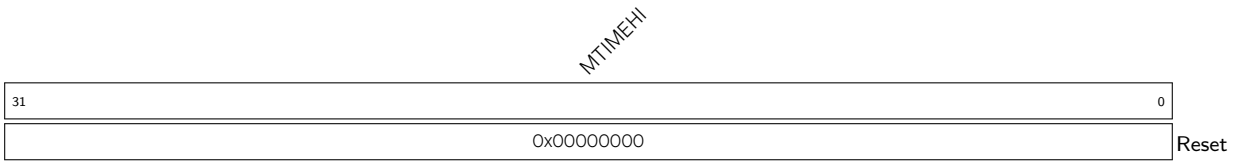
(R/W)

Register 2.74. mtimelo (0xBFF8)



MTIMELO 表示系统计数器当前低 32 位值。(RO)

Register 2.75. mtimehi (0xBFFC)



MTIMEHI 表示系统计数器当前高 32 位值。(RO)

2.9 存储器保护单元

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

2.9.1 标准物理存储器保护

2.9.1.1 概述

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

2.9.1.2 特性

PMP 单元用于控制对物理存储器的访问，主要特性包括：

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

2.9.1.3 功能描述

软件可以设置 PMP 单元的配置和地址寄存器，以保存错误并确保安全执行。配置 PMP CSR 可使能 PMP 单元，但只能在机器模式下进行配置，一旦 PMP 单元被使能，则将根据 16 个 pmpcfgX 和 pmpaddrX 寄存器（见 2.9.1.4）中的配置值对用户模式下的所有存储器访问进行写入、读取、访问权限检查。

默认情况下，PMP 允许机器模式下的所有存储器访问，而撤销用户模式下的所有访问。这意味着必须通过 pmpcfg 和 pmpaddr 寄存器（见 2.9.1.4）设置用户模式可以访问的地址范围和有效权限，以确保访问成功。但是在机器模式下没有此要求，因为机器模式下默认 PMP 允许所有访问。如果在机器模式下也需要 PMP 检查，则软件可以将所需 PMP 表项的锁定位置位来使能权限检查。锁定位一旦置位，就只能通过 CPU 复位被清零。

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

2.9.1.4 寄存器列表

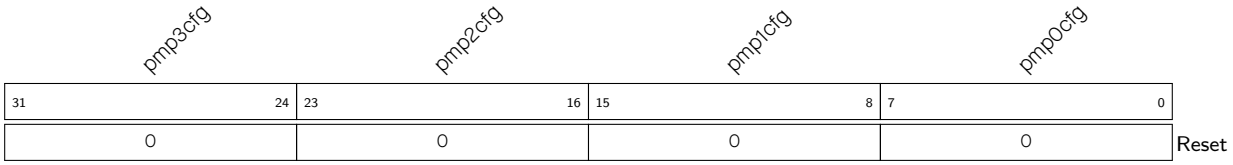
下表列出了 CPU 可访问的的 PMP CSR，只有在机器模式下才可以对它们进行读写。

名称	描述	地址	访问
pmpcfg0	物理存储器保护配置寄存器	0x3A0	R/W
pmpcfg1	物理存储器保护配置寄存器	0x3A1	R/W
pmpcfg2	物理存储器保护配置寄存器	0x3A2	R/W
pmpcfg3	物理存储器保护配置寄存器	0x3A3	R/W
pmpaddr*n* (*n*: 0-15)	物理存储器保护地址寄存器	0x3B0+0x1**n*	R/W

2.9.1.5 寄存器描述

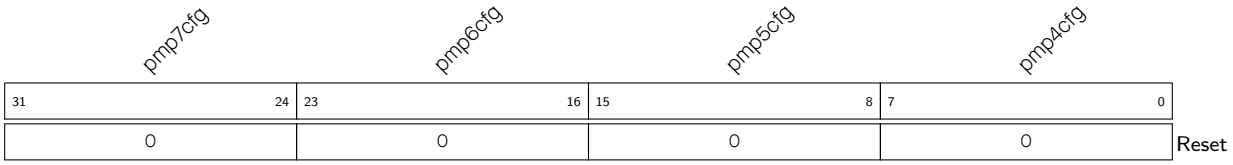
PMP 单元实现了《RISC-V 指令集手册 v1.10 第二卷 “特权架构”》中定义的所有 pmpcfg0-3 和 pmpaddr0-15 CSR。

Register 2.76. pmpcfg0 (0x3A0)



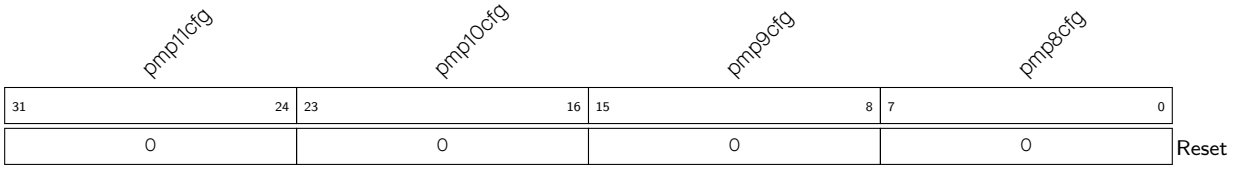
pmp3cfg PMP 表项 3 的配置寄存器。(R/W)
pmp2cfg PMP 表项 2 的配置寄存器。(R/W)
pmp1cfg PMP 表项 1 的配置寄存器。(R/W)
pmp0cfg PMP 表项 0 的配置寄存器。(R/W)

Register 2.77. pmpcfg1 (0x3A1)



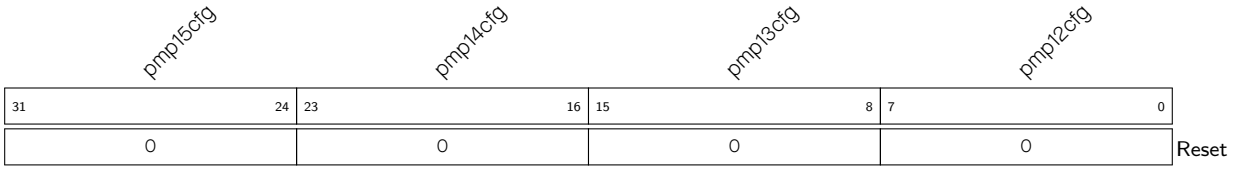
pmp7cfg PMP 表项 7 的配置寄存器。(R/W)
pmp6cfg PMP 表项 6 的配置寄存器。(R/W)
pmp5cfg PMP 表项 5 的配置寄存器。(R/W)
pmp4cfg PMP 表项 4 的配置寄存器。(R/W)

Register 2.78. pmpcfg2 (0x3A1)



pmp11cfg PMP 表项 11 的配置寄存器。(R/W)
pmp10cfg PMP 表项 10 的配置寄存器。(R/W)
pmp9cfg PMP 表项 9 的配置寄存器。(R/W)
pmp8cfg PMP 表项 8 的配置寄存器。(R/W)

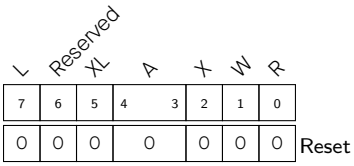
Register 2.79. pmpcfg3 (0x3A2)



pmp15cfg PMP 表项 15 的配置寄存器。(R/W)
pmp14cfg PMP 表项 14 的配置寄存器。(R/W)
pmp13cfg PMP 表项 13 的配置寄存器。(R/W)
pmp12cfg PMP 表项 12 的配置寄存器。(R/W)

以下为 PMP 表项配置寄存器的格式。

Register 2.80. pmpXcfg



L 配置锁定位。一旦设置，只能通过核心复位来清除。
(R/W)

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

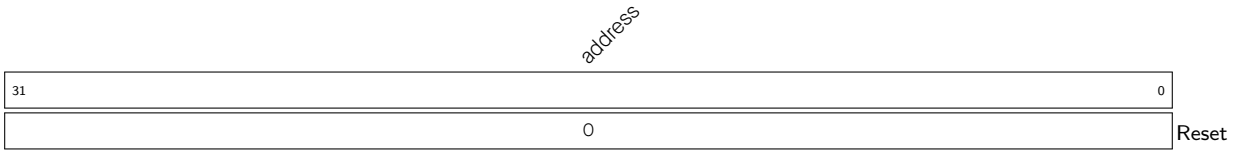
A 配置地址匹配模式。
0：关闭
1：TOR（范围顶端）
2：不支持
3：NAPOT（自然对齐的 2 的幂次方区域，大小 >= 128 字节）
(R/W)

X 配置执行权限。设置后，允许从地址范围执行。(R/W)

W 配置写入权限。设置后，允许对地址范围进行存储器写入。(R/W)

R 配置读取权限。设置后，允许从地址范围进行存储器读取。(R/W)

Register 2.81. pmpaddr*n* (*n*: 0-15) (0x3B0+0x1**n*)



address 配置 pmpaddr*n* 的编码地址。(R/W)

2.9.2 自定义物理存储器属性 (PMA) 检查器

2.9.2.1 概述

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

2.9.2.2 特性

PMAC 支持以下特性：

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

2.9.2.3 功能描述

软件可以设置 PMAC 单元的配置和地址寄存器，避免因访问无效存储器区域造成的错误。PMAC CSR 只能在机器模式下进行配置。一旦 PMAC 被使能，则将根据 16 个 pma_cfgX 和 pma_addrX 寄存器（见 2.9.2.4）中的配置值对各个模式下的所有存储器访问进行写入、读取、访问权限检查。访问被设置为无效的存储器的表项将导致获取错误或读写错误异常。

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

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

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

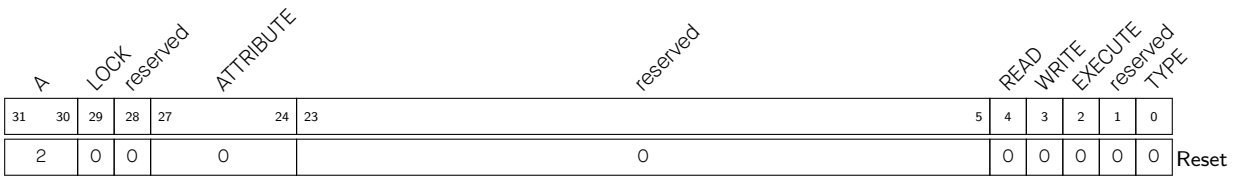
2.9.2.4 寄存器列表

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

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

2.9.2.5 寄存器

Register 2.82. pma_cfg*n* (*n*: 0-15) (0xBC0+0x1**n*)



A 配置地址类型。功能与 pmpcfg 寄存器的 A 字段相同。

0x0：OFF

0x1：TOR

0x2：NA4

0x3：NAPOT

(R/W)

LOCK 配置是否锁定对应的 pma_cfgX 和 pma_addrX。

0：不锁定

1：锁定，即撤销写入 pma_cfgX 和 pma_addrX 的权限

只有 CPU 复位才能解锁。

(R/W)

ATTRIBUTE 配置要在 DRAM 属性端口上驱动的值。

bit 27：

0：可缓存 (cacheable)

1：不可缓存 (non-cacheable)

bit 26：

0：写回 (write-back)

1：写直达 (write-through)

bit 25：

0：写失效分配 (write miss allocate)

1：写失效不分配 (write miss no allocate)

bit 24：0：读失效分配 (read miss allocate)

1：读失效不分配 (read miss no allocate)

(R/W)

READ 配置相应区域的读取权限。

0：不允许读取

1：允许读取

(R/W)

WRITE 配置相应区域的写权限。

0：不允许写入

1：允许写入

(R/W)

见下页……

Register 2.82. pma_cfg*n* (*n*: 0-15) (0xBC0+0x1**n*)

接上页……

EXECUTE 配置相应区域的执行权限。

0：不允许执行

1：允许执行

(R/W)

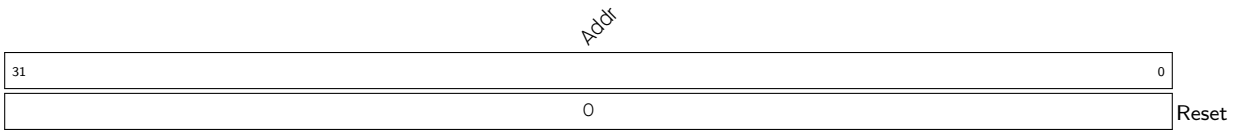
TYPE 配置区域类型。

0x0：存储器区域无效（RWX 访问将被视为 0，即使配置为 1）

0x1：存储器区域有效（RWX 访问可用）

(R/W)

Register 2.83. pma_addr*n* (*n*: 0-15) (0xBD0+0x1**n*)



ADDR 配置地址。功能与 pmpaddr 寄存器相同。(R/W)

2.10 调试和追踪

2.10.1 调试

2.10.1.1 概述

本节介绍如何调试在 HP 和 LP CPU 核心上运行的软件。调试功能由标准 JTAG 管脚提供，并符合 《RISC-V 外部调试支持规范 v0.13.2》。HP 核心和 LP 核心各自配备了独立的 JTAG DTM（调试传输模块），用于连接其对应的调试模块（Debug Module）。这两个 JTAG DTM 通过菊花链（daisy chain）串联至一个 JTAG 接口，调试器可通过该接口调试所需核心，并且在调试过程中，可以根据需求将未使用的 DTM 模块保持在旁路状态。HP 核心可通过调试模块 (DM) 进行调试，该调试模块连接至 JTAG DTM (HP Core) 模块。LP 核心使用单独的调试模块，该模块连接至 JTAG DTM (LP Core)。

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

图 2.10-1 显示了外部调试支持的主要组件。

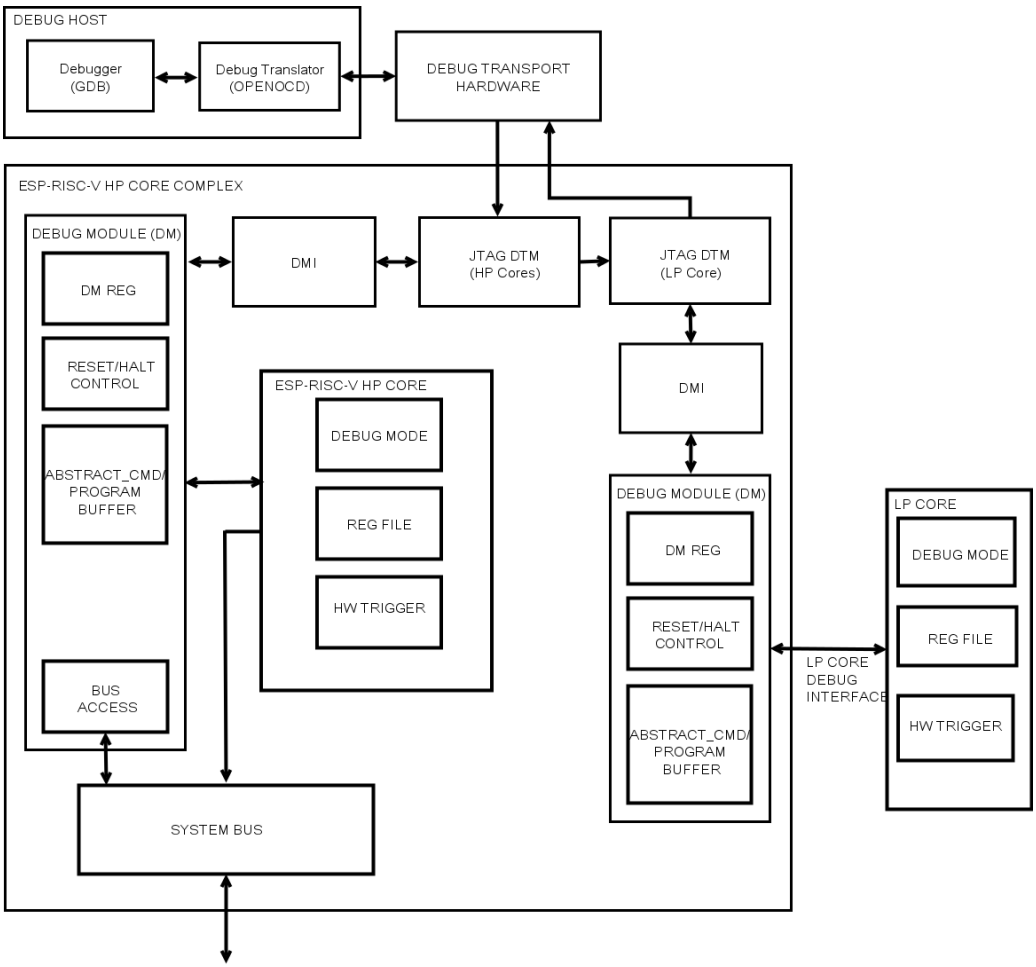


图 2.10-1. 调试系统概述

本节重点介绍 HP 核心的调试功能，有关 LP 核心的调试功能，请参阅章节 4 低功耗处理器。

用户与运行调试器（Debugger，例如 GDB）的调试主机（DEBUG HOST，例如笔记本电脑）进行交互。调试器通过调试转换器（Debug Translator，可能包含硬件驱动，例如 OPENOCD）与调试传输硬件（DEBUG TRANSPORT，例如 ESP-Prog 适配器）进行通信。调试传输硬件通过标准 JTAG 接口将调试主机连接到 HP 核

心的调试传输模块 (JTAG DTM)，绕过 LP 核心 DTM。JTAG DTM 使用调试模块接口 (DMI) 提供对调试模块 (DM) 的访问。

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

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

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

2.10.1.2 功能

HP CPU 核心支持以下基本调试功能：

- 向调试器提供有关实现的必要信息
- 支持暂停和恢复核心
- 核心寄存器（包括 CSR）可以由调试器读取/写入
- 可以从复位后执行的第一条指令开始调试核心
- 可以通过调试器复位核心
- 可以通过软件断点指令暂停核心
- 硬件单步调试
- 通过程序缓冲区在暂停的核心中执行任意指令。支持 2 个字的程序缓冲区
- 支持系统总线的字对齐地址访问
- HP 核心支持 3 个硬件触发器（可用作断点/观察点），详情见章节 2.10.2
- LP 核心可调试，通过菊花链串联 LP 核心的 JTAG DTM 来实现

2.10.1.3 功能描述

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

2.10.1.4 JTAG 控制

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

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

使用哪种 JTAG 方法取决于许多因素，具体配置方法如下表所示。

暂时关闭 JTAG 3,4	EFUSE_DIS_USB_JTAG 4	EFUSE_DIS_USB_SERIAL_JTAG 4	EFUSE_DIS_PAD_JTAG 4	EFUSE_JTAG_SEL_ENABLE 4	Strapping 管脚 GPIO7 5	USB JTAG 状态	PAD JTAG 状态
0	0	0	0	0	x 2	可用 1	不可用 1
0	0	0	0	1	1	可用	不可用

暂时关闭 JTAG 3	EFUSE_DIS_USB_JTAG 4	EFUSE_DIS_USB_SERIAL_JTAG 4	EFUSE_DIS_PAD_JTAG 4	EFUSE_JTAG_SEL_ENABLE 4	Strapping 管脚 GPIO7 5	USB JTAG 状态	PAD JTAG 状态
0	0	0	0	1	0	不可用	可用
0	0	1	0	x	x	不可用	可用
0	1	0	0	x	x	不可用	可用
0	1	1	0	x	x	不可用	可用
0	0	0	1	x	x	可用	不可用
0	0	1	1	x	x	不可用	不可用
0	1	0	1	x	x	不可用	不可用
0	1	1	1	x	x	不可用	不可用
1	x	x	x	x	x	不可用	不可用

说明：

1. 可用：相应的 JTAG 功能可用
   不可用：相应的 JTAG 功能不可用
2. x：无关项
3. “暂时关闭 JTAG” 表示如果 EFUSE_SOFT_DIS_JTAG[2:0] 中有偶数位 “1”，则开启 JTAG 功能（表中对应值为 1），否则，JTAG 功能被关闭（表中对应的值为 0）。但是，在 ESP32-C5 HMAC 加速器的某些特殊条件下，即使 EFUSE_SOFT_DIS_JTAG[2:0] 中有奇数位 “1”，JTAG 功能也可能被开启。有关 HMAC 如何影响 JTAG 功能的信息，请参阅 HMAC Accelerator 章节。
4. 有关 eFuse 的更多信息，请参考 eFuse Controller 章节。
5. 有关 strapping 管脚 GPIO7 的更多信息，请参考 Chip Boot Control 章节。

2.10.1.5 寄存器

以下是 HP 核心支持的调试 CSR 列表：

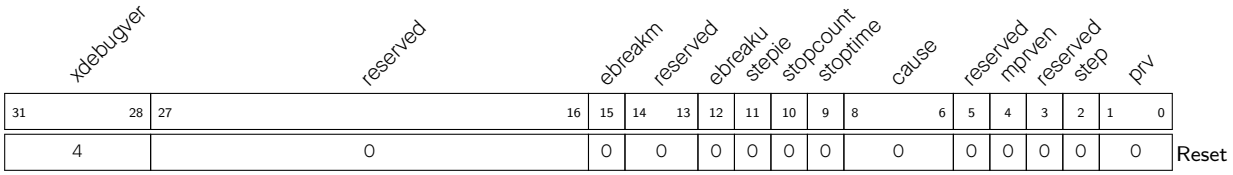
名称	描述	地址	访问
dcsr	调试控制和状态寄存器	0x7B0	R/W
dpc	调试 PC 寄存器	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试暂存寄存器 1	0x7B3	R/W

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

2.10.1.6 寄存器描述

以下是 HP 核心支持的调试 CSR 的详细信息：

Register 2.84. dcsr (0x7B0)



xdebugver 表示调试版本。

4：存在外部调试支持

(RO)

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

(R/W)

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

(R/W)

stepie 配置调试模式下的中断控制。

0：单步执行时禁用中断

1：单步执行时启用中断

调试器在 hart 运行时不得更改此位的值。

(R/W)

stopcount 配置调试模式下的 mcycle 计数器。

0：累加计数器

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

(R/W)

stoptime 配置调试模式下的计时器。

0：累加计时器

1：在调试模式下不累加任何 hart 本地计时器

(R/W)

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

1：执行了 ebreak 指令（优先级 3）

2：触发模块导致暂停（优先级 4，最高）

3：haltreq 被设置（优先级 1）

4：由于 step 被设置，CPU 核心单步执行（优先级 0，最低）

5：由于 resethaltreq，hart 在复位后暂停（优先级 2）

其他值：保留，供未来使用

(RO)

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

0：禁用

1：使能

(R/W)

见下页……

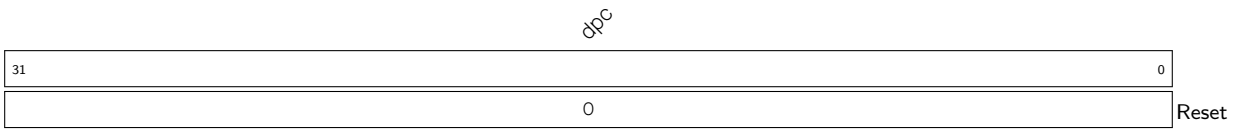
Register 2.84. dcsr (0x7B0)

接上页……

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

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

Register 2.85. dpc (0x7B1)

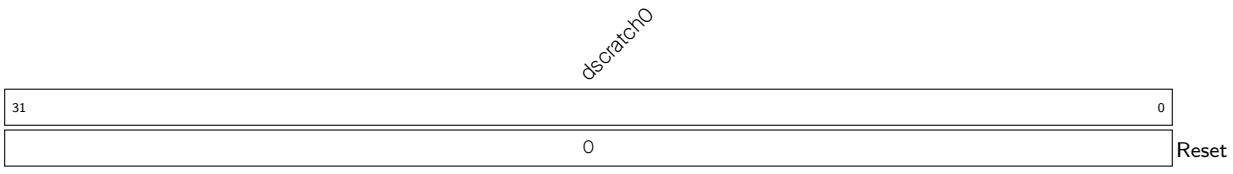


dpc 进入调试模式后，dpc 将写入下一条指令的虚拟地址。恢复执行时，CPU 核心的 PC 将更新为 dpc 保存的虚拟地址。调试器可以写入 dpc 来配置 CPU 恢复执行的位置。(R/W)

表 2.10-3. 进入调试模式时 DPC 中的虚拟地址

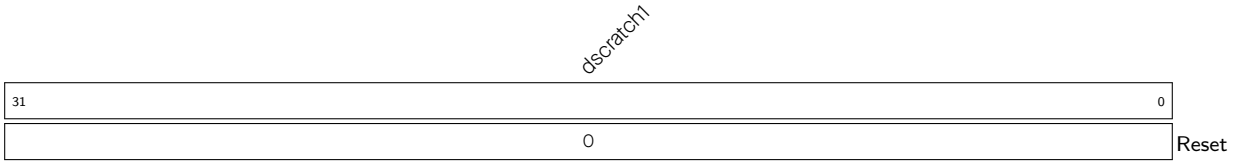
原因	DPC 中的虚拟地址
ebreak	ebreak 指令的地址
单步执行	如果没有调试进行，将要执行的下一条指令的地址，即对于不改变程序流的 32 位指令为 pc+4，跳转/分支为目标 PC 等。
触发模块	如果 timing 为 0，则为导致触发的指令的地址。如果 timing 为 1，则为进入调试模式时要执行的下一条指令的地址。
暂停请求	进入调试模式时要执行的下一条指令的地址。

Register 2.86. dscratch0 (0x7B2)



dscratch0 由调试模块内部使用。(R/W)

Register 2.87. dscratch1 (0x7B3)



dscratch1 由调试模块内部使用。(R/W)

2.10.2 硬件触发器

2.10.2.1 概述

HP 核心实现一个硬件触发器模块，提供调试的断点和监视点功能。该模块遵循 《RISC-V 外部调试支持规范 v0.13.2》 。有关具体功能和操作方式，请参考该规范文档。

2.10.2.2 特性

HP 核心硬件触发器模块支持以下特性：

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

2.10.2.3 功能描述

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

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

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

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

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

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

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

每个触发单元仅支持地址匹配，该地址可以是存储器访问地址，也可以是指令的虚拟地址。通过写入所选触发单元的 maddress (tdata2) CSR，可以指定区域的地址和大小。大于 1 个字节的区域大小通过 NAPOT 编码（见表 2.10-4）指定，并通过置位 mcontrol 中 match 域来使能。注意，根据定义，NAPOT 编码地址的起始地址需与区域大小对齐（即，是区域大小的整数倍），最大支持 128 字节。

表 2.10-4. NAPOT 编码的 maddress

maddress(31-0)	起始地址	大小（字节）

aaa...aaaaaaaaa0	aaa...aaaaaaaaa0	2
aaa...aaaaaaaa01	aaa...aaaaaaaa00	4
aaa...aaaaaaa011	aaa...aaaaaaa000	8
aaa...aaaaaa0111	aaa...aaaaaa0000	16
	....	
aaa...0111111111	aaa...aaa0000000	2^8

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

2.10.2.4 触发执行流程

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

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

当触发器触发引起 hart 进入陷阱时 (action = 0)：

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

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

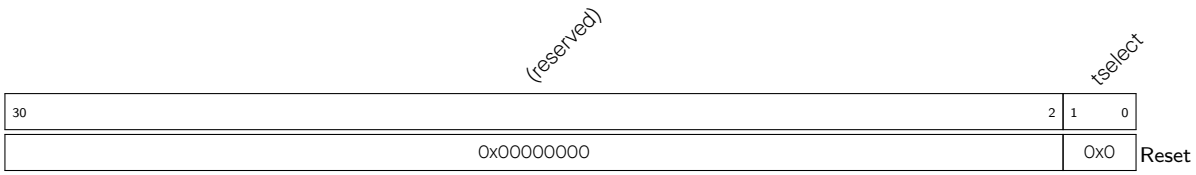
2.10.2.5 寄存器列表

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

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

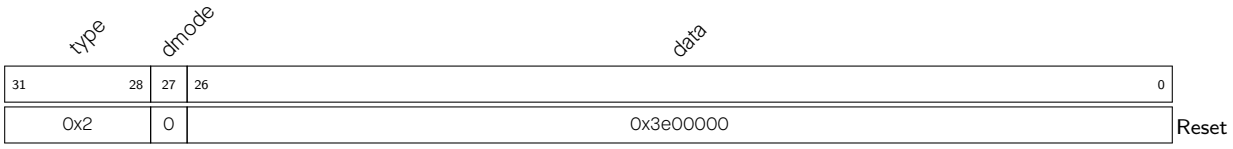
2.10.2.6 寄存器描述

Register 2.88. tselect (0x7A0)



tselect 配置选中触发器的单元编号 (0-3)。(R/W)

Register 2.89. tdata1 (0x7A1)



type 表示触发器类型。仅支持匹配类型 (0x2)，此域保留。(RO)

dmode 如果某触发器正在被调试器使用，则此域置为 1。

0：在调试模式和机器模式下都能写入 tdata1 和 tdata2

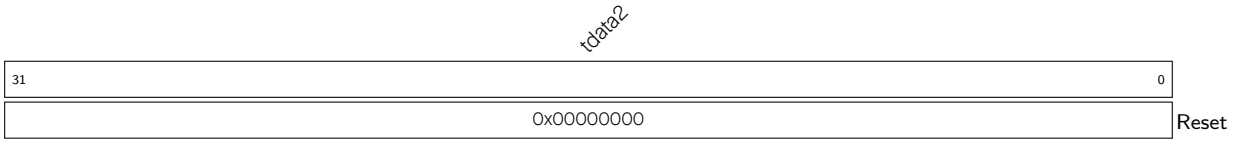
1：只有在调试模式下才能写入 tdata1 和 tdata2，其他模式下的写操作将被忽略

注意：仅支持调试模式下的写操作

(R/W)

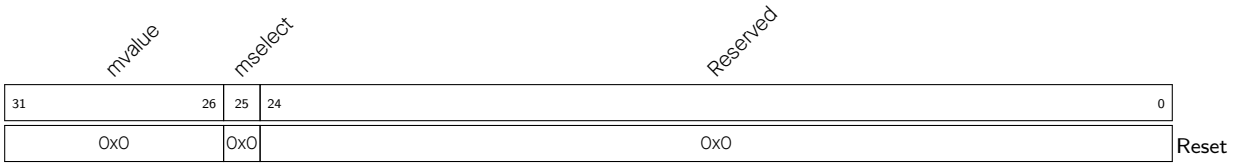
data 配置抽象 tdata1 的内容。由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 mcontrol 的域。(R/W)

Register 2.90. tdata2 (0x7A2)



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

Register 2.91. tdata3 (0x7A3)

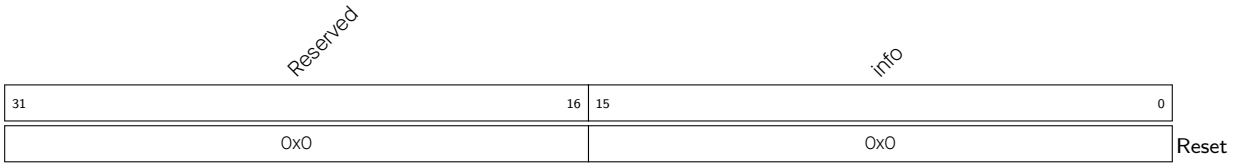


mvalue 与 mselect 一起使用的数据。(R/W)

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

Reserved 读取为 0。(RO)

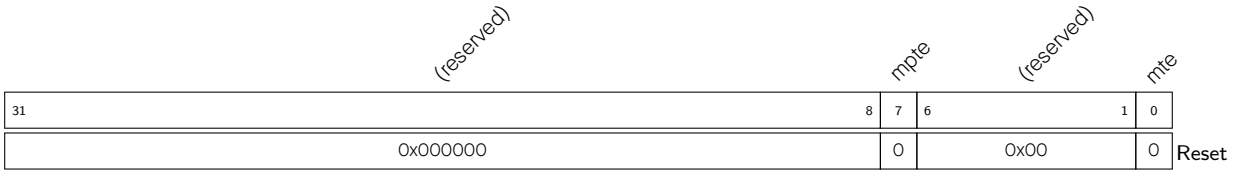
Register 2.92. tinfo (0x7A4)



Reserved 读取为 0。(RO)

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

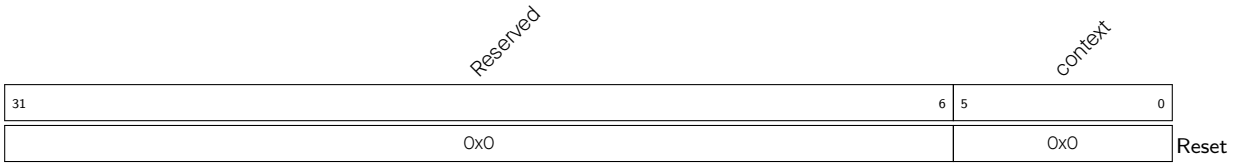
Register 2.93. tcontrol (0x7A5)



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

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

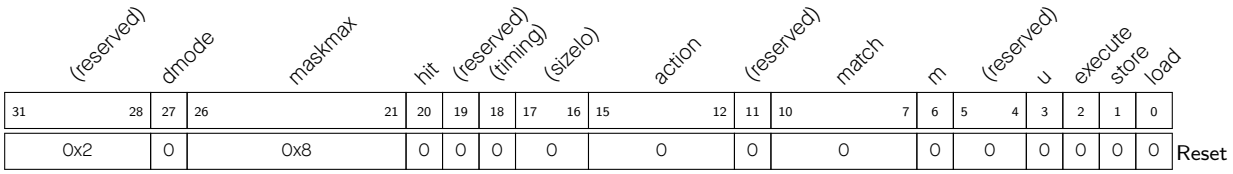
Register 2.94. MCONTEXT (0x7A8)



Reserved 读取为 0。(RO)

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

Register 2.95. mcontrol (0x7A1)



dmode 与 tdata1 中的 dmode 相同。(RW*)

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

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

timing 设置此域以启用加载/存储触发器动作的时序。

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

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

(R/W)

sizelo 此域包含访问地址的最低两位。

0x0：触发器将尝试匹配任何大小地址的访问

0x1：触发器将尝试匹配 8 位存储器访问

0x2：触发器将尝试匹配 16 位存储器访问

0x3：触发器将尝试匹配 32 位存储器访问

(R/W)

action 配置所选触发器在触发时执行动作。有效选项为：

0x0：引发断点异常

0x1：进入调试模式（仅在 dmode=1 时有效）

0x2：在触发器 0 命中时启用追踪

0x3：在触发器 1 命中时启用追踪

0x4：在触发器 2 命中时启用追踪

注意：写入无效值将设置为默认值 0x0。

(R/W)

match 配置触发器进行数据/指令地址的匹配操作。有效选项为：

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

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

注意：写入超过最大值的数值会被裁剪为最大值 0x1。

(R/W)

见下页……

Register 2.95. mcontrol (0x7A1)

接上页……

m 置位使选定的触发器在机器模式下操作。(R/W)

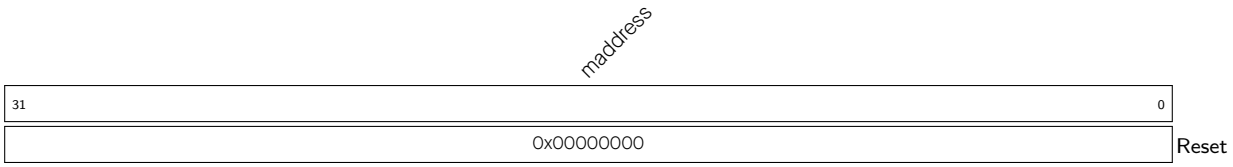
u 置位使选定的触发器在用户模式下操作。(R/W)

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

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

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

Register 2.96. maddress (0x7A2)



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

2.10.3 追踪

2.10.3.1 概述

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

2.10.3.2 特性

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

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

2.10.3.3 功能描述

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

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

2.11 性能

2.11.1 分支预测

2.11.1.1 概述

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

2.11.1.2 特性

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

2.11.1.3 功能描述

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

BHT 是一块 512×16 位的 SRAM 存储器，由 HP CPU 用于分支预测。在预测时，使用虚拟分支历史寄存器 (VBHR) 作为索引。在更新时，分支指令执行结果出来之后，由全局分支历史寄存器 (GBHR) 作为索引。GBHR 包含前 13 条分支指令执行结果，VBHR 包含前 12 条结果和当前分支指令的预测结果。因此，这两个寄存器都保存了先前执行的分支轨迹。基于 VBHR/GBHR 的 13 位，BHT 中存储的 2 位饱和计数器值分别被读取或写入。

如图 2.11-1 所示，2 位饱和计数器表示分支是否为：

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

如果预测计数器处于任意“未跳转”状态，则该分支会被预测为未跳转；同样地，如果预测计数器处于任意“跳转”状态，则会被预测为跳转。当分支的实际执行结果确定后，会根据下面的 2 位饱和计数器状态转换规则来更新对应的 BHT 表项。

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

图 2.11-1 详细介绍了 HP 核心中使用的 2 位饱和方案。

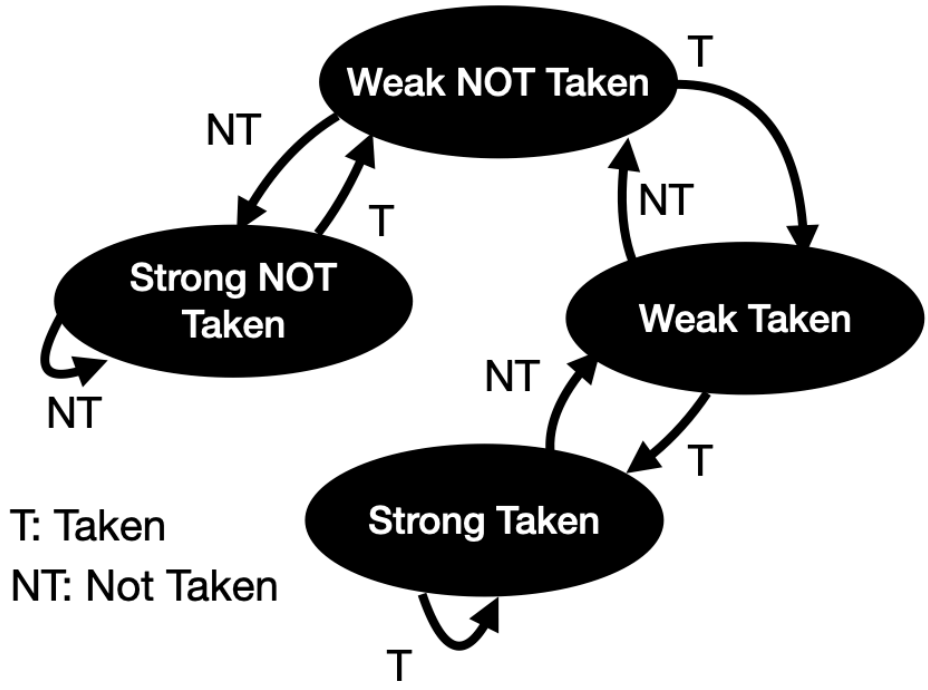


图 2.11-1. 2 位饱和计数器

2.11.2 RAS

2.11.2.1 概述

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

2.11.2.2 特性

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

2.11.2.3 功能描述

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

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

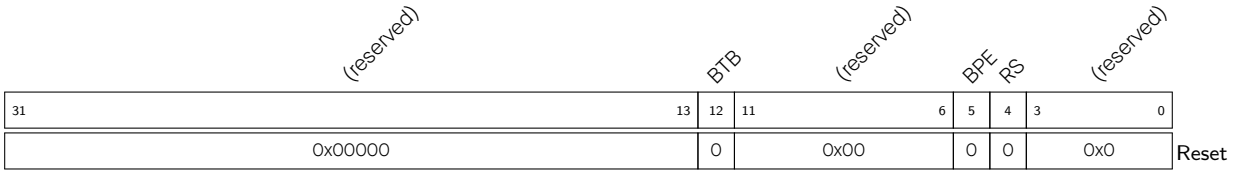
RAS 预测取决于以下条件：

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

rd != x1	rs1 == x1	弹出
rd == x1	rs1 != x1	推入
rd == x1	rs1 == x1	推入并弹出

2.11.3 性能配置控制状态寄存器

Register 2.97. mhcr (0x7C1)



RS 配置是否启用返回地址栈逻辑进行预测。

0：禁用

1：启用

(R/W)

BPE 配置是否启用条件分支预测。

0：禁用

1：启用

(R/W)

BTB 配置是否启用分支目标缓冲区。

0：禁用

1：启用

(R/W)

2.12 自定义功能

2.12.1 总线错误响应

2.12.1.1 概述

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

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

下一小节将介绍这些总线错误 CSR 以及获取总线错误异常上下文的推荐方法。

2.12.1.2 功能描述

有两种方法来处理加载/存储总线错误：

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

2.12.1.3 控制状态寄存器

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

2.12.2 专用 IO

2.12.2.1 概述

GPIO 通常为 APB 外设，这意味着对输出的更改和对输入的读取，可能会卡在写缓冲区或其他传输之后，并且速度通常较慢，因为 APB 总线的运行速度通常低于 CPU。作为替代方案，CPU 核心实现了 I/O 处理器特定的 CPU 寄存器 (CSR)，直接连接到 GPIO 矩阵或 IO 管脚。这些寄存器可以通过单条指令访问，从而能够快速控制 IO 管脚。

2.12.2.2 特性

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

2.12.2.3 功能描述

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

CPU 自定义了 3 个 CSR：

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

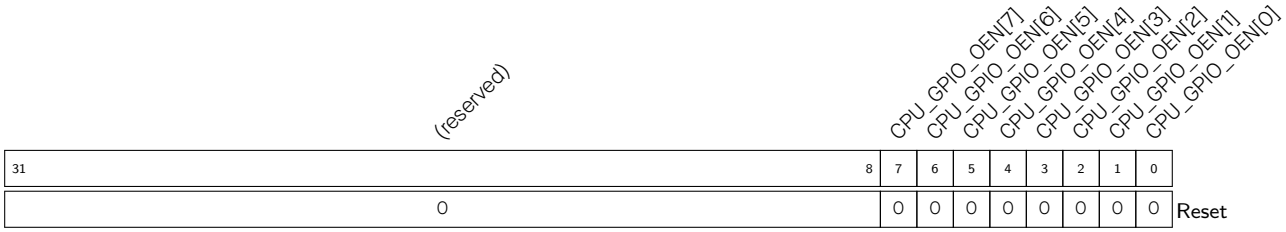
2.12.2.4 寄存器列表

下面是在核心中自定义的专用 IO CSR 的列表。

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

2.12.2.5 寄存器

Register 2.98. cpu_gpio_oen (0x803)



CPU_GPIO_OEN 配置是否使能 GPIOn (n=0 ~ 21) 输出。CPU_GPIO_OEN[7:0] 分别对应章节 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* 中表 8.12-1 里的 cpu_gpio_out_oen[7:0] 输出使能信号。

CPU_GPIO_OEN 的值与 cpu_gpio_out_oen 的值对应。

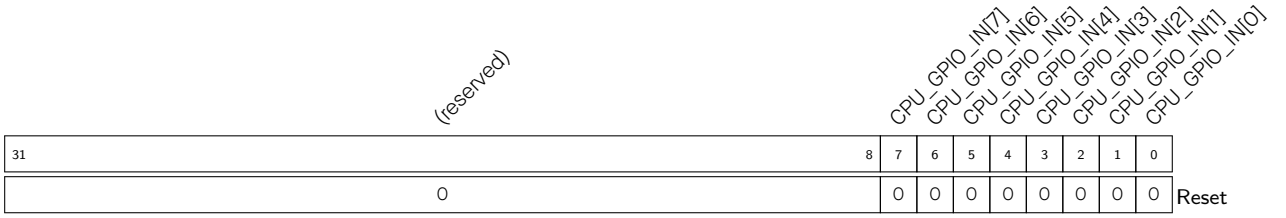
此寄存器是 CPU_GPIO_OUT 的使能寄存器。

0：关闭 GPIO 输出

1：使能 GPIO 输出

(R/W)

Register 2.99. cpu_gpio_in (0x804)



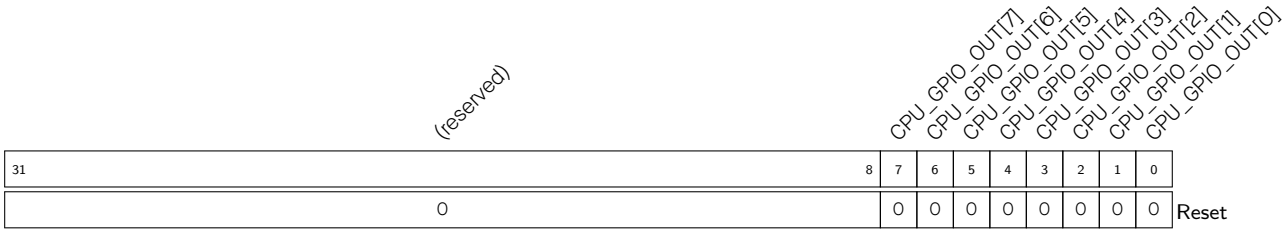
CPU_GPIO_IN 表示 SoC GPIOn (n=0 ~ 21) 的输入值（1 为高电平，0 为低电平）。

CPU_GPIO_IN[7:0] 分别对应章节 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* 中表 8.12-1 里的 cpu_gpio_in[7:0] 输入信号。

CPU_GPIO_IN[7:0] 只能通过 GPIO 交换矩阵映射到 GPIO。详细描述请参考章节 8.4。

(RO)

Register 2.100. cpu_gpio_out (0x805)



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

CPU_GPIO_OUT[7:0] 分别对应章节 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* 中表 8.12-1 里的 cpu_gpio_out[7:0] 输出信号。

CPU_GPIO_OUT[7:0] 只能通过 GPIO 交换矩阵映射到 GPIO。详细描述请参考章节 8.5。

(R/W)

2.12.3 运行暂停 (RunStall)

2.12.3.1 概述

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

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

2.12.3.2 功能描述

RunStall 输入信号可用于以下两种场景：

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

图 2.12-1 显示了 RunStall 和 CoreStalled 信号的时序图以及 CPU 核心状态的切换。

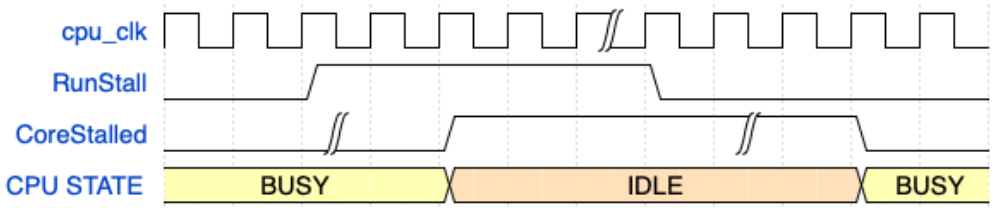


图 2.12-1. 使用 RunStall 时 CPU 核心状态的切换

2.12.3.3 寄存器列表

章节 3 *RISC-V 追踪编码器 (TRACE)* 中的 TRACE_STALL_ENA 寄存器域可用于激活 HP 核心的 RunStall。

2.12.4 调试辅助信息

HP CPU 核心系统向 ESP32-C5 中的调试辅助模块提供了额外的调试信息。详细信息请参阅章节 20 辅助调试。

2.12.5 核心锁定

2.12.5.1 概述

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

2.12.5.2 功能描述

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

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

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

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

2.12.5.3 寄存器列表

mexstatus.LOCKUP 位表示单个核心的锁定状态。

2.12.5.4 寄存器描述

请参阅 mexstatus CSR。

第 3 章

RISC-V 追踪编码器 (TRACE)

ESP32-C5 的高性能 CPU (HP CPU) 通过追踪编码器 (Trace Encoder) 实现了对指令追踪接口的支持。追踪编码器连接 CPU 的指令追踪接口，将信息压缩成更小的数据包，存入内部 SRAM。

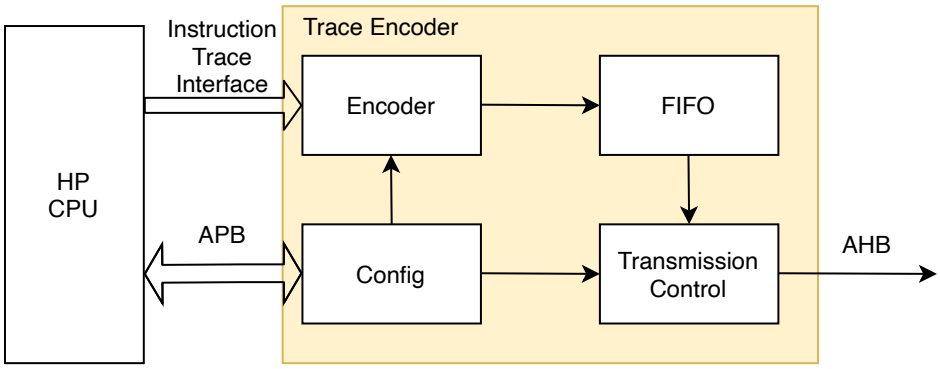


图 3.0-1. 追踪编码器概述图

3.1 术语

为了更好地说明 RISC-V 追踪编码器的功能，本章使用了以下术语。

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

3.2 介绍

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

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

此时，可使用指令追踪来追踪程序执行。指令追踪的工作原理是从一个已知的起始地址追踪执行，并发送通常由跳转、调用、返回和分支指令产生的地址增量，中断和异常也会产生地址增量。

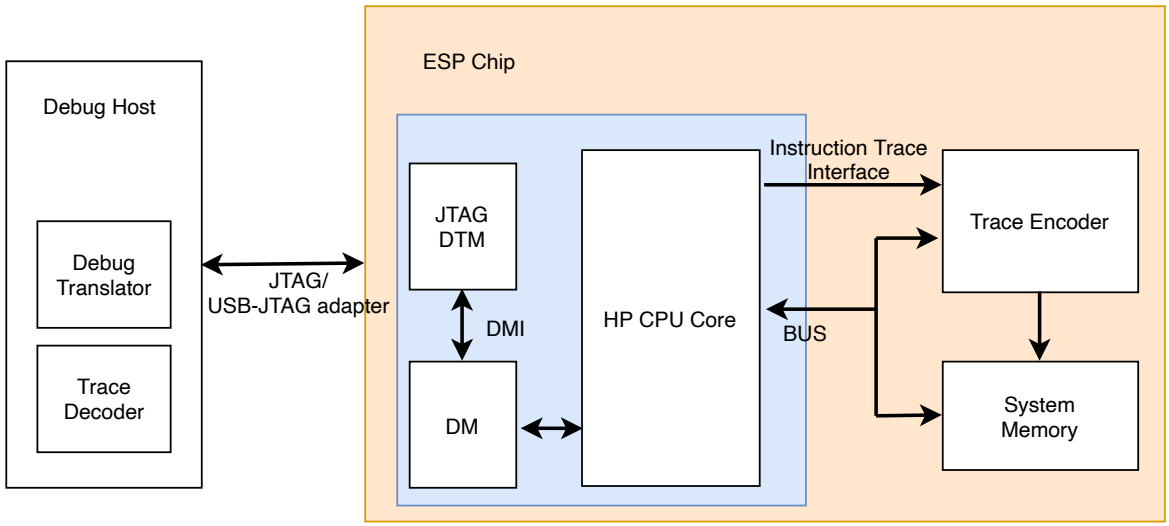


图 3.2-1. 指令追踪流程概览

图 3.2-1 为指令增量追踪的流程。

- HP CPU 核有一个指令追踪接口，可以输出 HP CPU 执行指令的信息，包括指令地址、指令类型等。更多关于 ESP32-C5 HP CPU 指令追踪接口的信息，可参考章节 2 高性能处理器。
- 追踪编码器通过 HP CPU 的指令追踪接口获取相关指令追踪信息，将信息压缩成带宽更小的数据包，存入系统存储器中。
- 调试器（即图中的 Debug Host）可以将系统存储器中的追踪数据包通过 JTAG 或 USB Serial/JTAG 导出，然后用解码器解压缩、重建程序的执行流。追踪解码器 (Trace Decoder) 通常是外部 PC 上的软件，在获取追踪数据包后，根据在 hart 上运行的程序二进制数据，重建程序的指令流。解码可以离线进行，也可以在 hart 执行时实时进行。

3.3 特性

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

  - hart 复位后，编码器在报告最后一个数据包后停止工作
  - 追踪 FIFO 即将变满时暂停 hart
- 支持任意地址范围用作追踪存储器
- 可配置的同步模式：
  - 同步计数器按包计数
  - 同步计数器按周期计数
  - 关闭同步计数器
- 支持丢包状态标识
- 支持丢包后自动重启
- 写追踪存储器时支持循环和非循环模式
- 具有两个中断：
  - 包的大小超过配置的存储器空间时触发中断
  - 丢包时触发中断
- 具有 128 × 8 位 FIFO，用于缓存数据包
- 支持 AHB 突发传输，突发长度可配置

表 3.3-1. 追踪编码器参数

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

参数名称	值	描述
retires_p	1	每个块退役指令的最大数量
return_stack_size_p	0	不支持隐式返回模式
sijump_p	0	不支持根据顺序推测跳转
taken_branches_p	1	每个时钟周期仅一条指令退役
impdef_width_p	0	未实现

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

3.4 架构概览

本章节主要介绍 ESP32-C5 追踪编码器的具体实现。

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

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

发送控制模块通过 AHB 总线将 FIFO 中的数据写入内部 SRAM。

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

3.5 功能描述

3.5.1 同步

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

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

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

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

3.5.2 地址模式

ESP32-C5 支持两种地址模式：增量地址模式和完整地址模式。

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

3.5.3 可选边带信号

Efficient Trace for RISC-V Version 2.0 > 章节 Optional Sideband Signals 介绍了用于控制编码器的可选边带信号。ESP32-C5 实现了下列信号：

表 3.5-1. 可选边带信号

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

3.5.4 过滤

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

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

每个过滤器单元包含一个过滤器和一个比较器单元。

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

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

每个过滤器可以指定要过滤的 hart 指令，并具有以下功能：

- 选择比较器单元进行匹配
- 可以选择特定的特权等级 (priv) 和异常原因 (ecause) 作为输入进行匹配
- 选择匹配中断

3.5.5 锚定

由于数据包的长度不定，为在写入存储器时分清数据包之间的边界，ESP32-C5 编码器在数据包之间插入 0：

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

3.5.6 写存储器模式

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

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

3.5.7 自动重启

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

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

3.6 编码器输出数据包

本节主要介绍 ESP32-C5 追踪编码器输出的数据包格式。

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

3.6.1 头部

头部为 1 个字节，格式如表 3.6-1 所示。

表 3.6-1. 头部格式

字段	位宽	描述	值
length	5	数据包的总长度	4~13

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

3.6.2 索引

索引为 2 个字节，格式如表 3.6-2 所示。

表 3.6-2. 索引格式

字段	位宽	描述	值
index	16	每个数据包的索引	0~65536

3.6.3 有效载荷

有效载荷的宽度为 1 至 10 字节。

3.6.3.1 格式 3

格式 3 的数据包用于同步和发送支持信息。规范中定义了四种子格式，ESP32-C5 仅支持 3 种。

格式 3 子格式 0 - 同步

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

表 3.6-3. 格式 3 子格式 0

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

格式 3 子格式 1 - 异常

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

表 3.6-4. 格式 3 子格式 1

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

格式 3 子格式 3 - 支持

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

表 3.6-5. 格式 3 子格式 3

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

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

3.6.3.2 格式 2

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

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

表 3.6-6. 格式 2

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

3.6.3.3 格式 1

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

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

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

格式 1，有地址时 branch_map 字段

有效载荷长度不定。

表 3.6-7. 格式 1，有地址

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

格式 1，无地址时 branch_map 字段

该格式数据包有效载荷的长度为 5 个字节。

表 3.6-8. 格式 1，无地址

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

3.7 中断

ESP32-C5 的 TRACE（即 HP CPU 的追踪编码器）可以生成 TRACE_INTR 中断信号并发送给中断矩阵。

TRACE_INTR 中断信号由 TRACE 的内部中断源生成。表 3.7-1 列出了 TRACE 的内部中断源、中断触发条件、以及生成的中断信号。

表 3.7-1. TRACE 的内部中断源

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

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

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

3.8 编程流程

3.8.1 配置编码器选项

默认情况下，追踪数据包和增量地址会发送到存储器中，直到追踪 FIFO 溢出或追踪存储器变满为止。可以通过一些可选的配置设置来改变追踪行为。编码器具有一些可选的配置选项，可以改变追踪行为。

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

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

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

  HP CPU 的调试模块有一个触发单元，其匹配控制寄存器 (mcontrol) 有一个 4 位的 action 字段，该字段值为 2 ~ 4 时用于追踪。ESP32-C5 已实现此功能。action 字段的值详见表 3.8-1。

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

值	描述
2	开始追踪
3	结束追踪
4	通知编码器报告地址

有关调试模块触发单元的详细信息，请参考章节 2 高性能处理器。

3.8.2 配置过滤器

章节 3.5.4 中描述的过滤单元可以按以下方式配置：

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

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

  如果置位 TRACE_P_NOTIFY 或 TRACE_S_NOTIFY，比较器会以通知的形式报告匹配地址，指令不会过滤。例如：
  - 如果 TRACE_MATCH_MODE 为 0，并且 TRACE_P_NOTIFY 为 1，则所有指令都会追踪，指令与主比较器匹配时会发出通知。关于通知的介绍，请查看章节 3.8.5。
  - 如果 TRACE_MATCH_MODE 为 0，并且 TRACE_S_NOTIFY 为 1，则指令将通过主比较器过滤，指令与次比较器匹配时会发出通知。

2. 置位 TRACE_FILTER_EN 以启用过滤单元。

3.8.3 使能编码器

- 通过 TRACE_MEM_START_ADDR_REG 和 TRACE_MEM_END_ADDR_REG 配置追踪存储器的地址空间。编码器可以访问的地址范围为 0x40800000 ~ 0x4085FFFF
- 置位 TRACE_MEM_CURRENT_ADDR_UPDATE，将 TRACE_MEM_CURRENT_ADDR_REG 的值更新为 TRACE_MEM_START_ADDR_REG 的值
- （可选）通过 TRACE_TRIGGER_REG 寄存器的 TRACE_MEM_LOOP 位配置写存储器的模式
  - 0：非循环模式
  - 1：循环模式（默认）
- 通过 TRACE_RESYNC_PROLONGED_REG 寄存器的 TRACE_RESYNC_MODE 位配置同步模式
  - 0：关闭同步计数器（默认）
  - 1：无效值，没有作用
  - 2：同步计数器按数据包计数
  - 3：同步计数器按周期计数

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

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

3.8.4 关闭编码器

停止输出追踪数据包，有以下两种方式：

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

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

表 3.8-2. 追踪状态

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

3.8.5 通知

如果一个目标地址很特殊，即使不是无法推测的不连续地址，用户也可能想要报告。

通知和报告特殊地址有以下两种方式：

1. 调试模块的触发器地址匹配，并且 action 为 4

2. 过滤器比较器匹配

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

3.8.6 解码数据包

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

3.8.7 AHB 配置

ESP32-C5 编码器支持 AHB 总线，可选配置项可以控制编码器的传输带宽。

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

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

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

该配置用于调整带宽。带宽足够时，不需要配置突发传输，可使用默认配置。

3.8.8 软件保留

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

3.9 寄存器列表

本小节的所有地址均为相对于 RISC-V 追踪编码器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

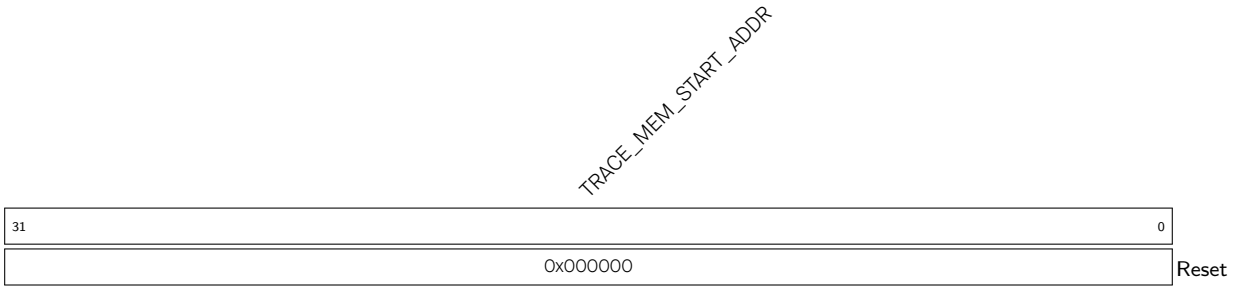
名称	描述	地址	访问
存储器配置寄存器			
TRACE_MEM_START_ADDR_REG	存储器起始地址	0x0000	R/W
TRACE_MEM_END_ADDR_REG	存储器结束地址	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	存储器当前地址	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	存储器地址更新	0x000C	WT
追踪编码器 FIFO 状态寄存器			
TRACE_FIFO_STATUS_REG	FIFO 装填寄存器	0x0010	RO
中断寄存器			
TRACE_INTR_ENA_REG	中断使能寄存器	0x0014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

3.10 寄存器

本小节的所有地址均为相对于 RISC-V 追踪编码器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

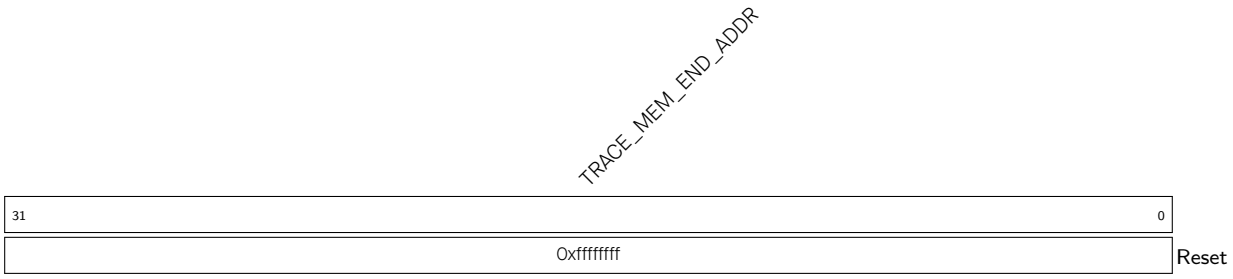
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 3.1. TRACE_MEM_START_ADDR_REG (0x0000)



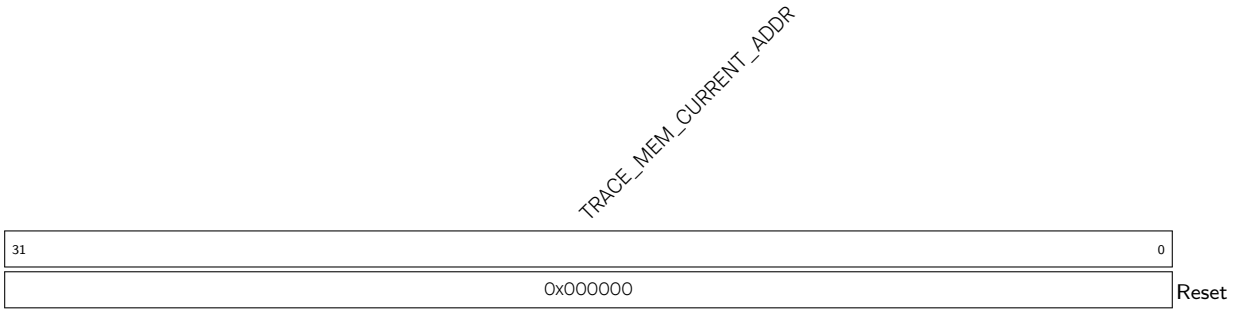
TRACE_MEM_START_ADDR 配置追踪存储器的起始地址。(R/W)

Register 3.2. TRACE_MEM_END_ADDR_REG (0x0004)



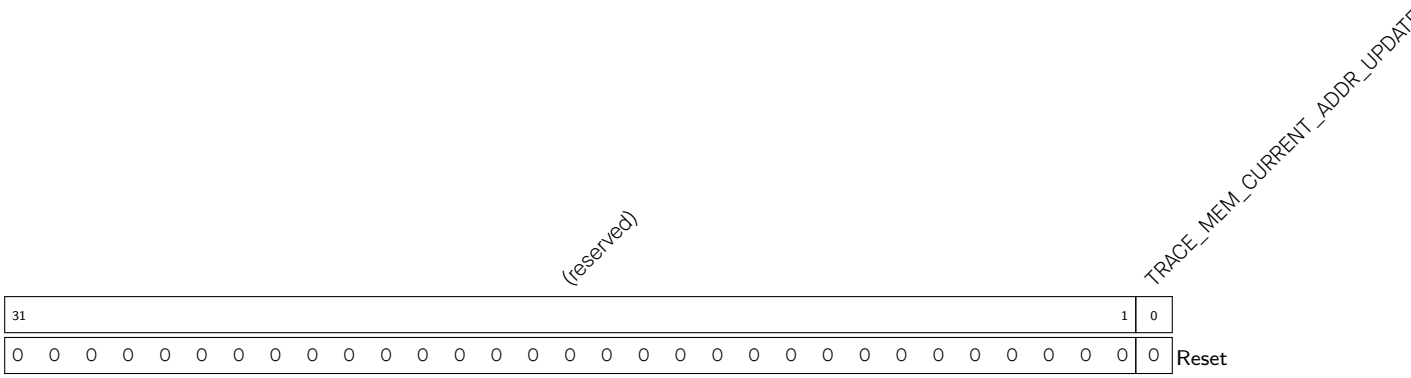
TRACE_MEM_END_ADDR 配置追踪存储器的结束地址。(R/W)

Register 3.3. TRACE_MEM_CURRENT_ADDR_REG (0x0008)



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

Register 3.4. TRACE_MEM_ADDR_UPDATE_REG (0x000C)



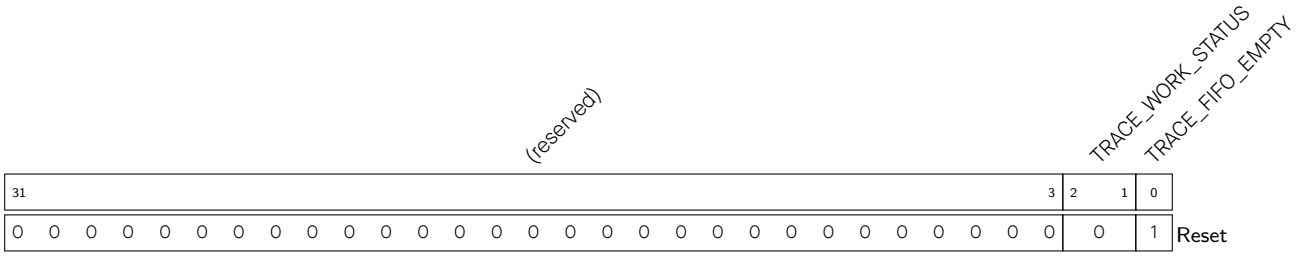
TRACE_MEM_CURRENT_ADDR_UPDATE 配置是否将 TRACE_MEM_CURRENT_ADDR 的值更新为 TRACE_MEM_START_ADDR 的值。

0：不更新

1：更新

(WT)

Register 3.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY 表示 FIFO 的状态。

0：不为空

1：为空

(RO)

TRACE_WORK_STATUS 表示编码器的状态。

0：空闲状态

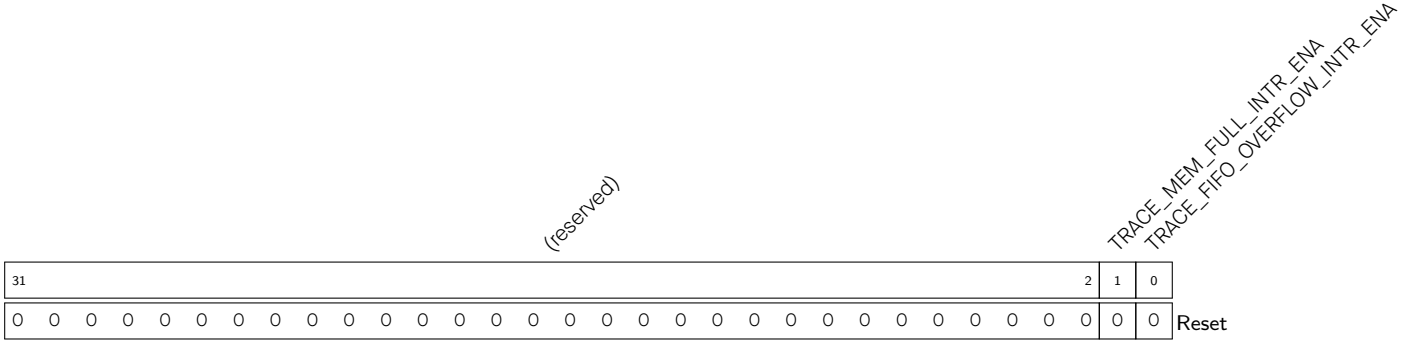
1：工作状态

2：因 hart 暂停或复位处于待机状态

3：丢失状态

(RO)

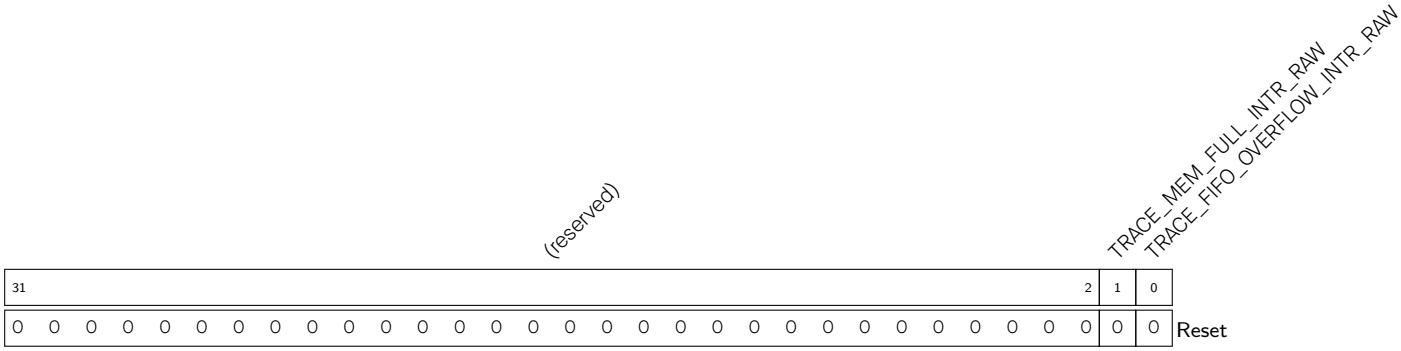
Register 3.6. TRACE_INTR_ENA_REG (0x0014)



TRACE_FIFO_OVERFLOW_INTR_ENA 写 1 使能 TRACE_FIFO_OVERFLOW_INTR。(R/W)

TRACE_MEM_FULL_INTR_ENA 写 1 使能 TRACE_MEM_FULL_INTR。(R/W)

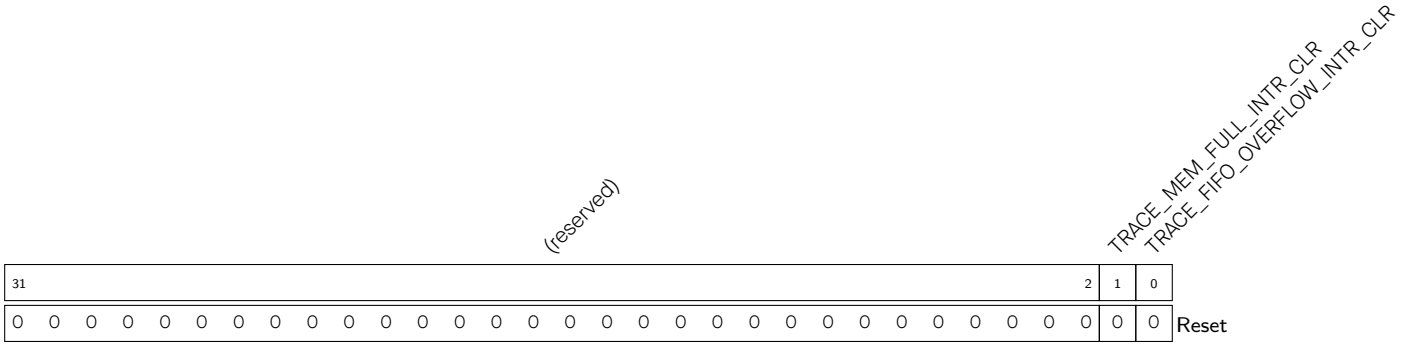
Register 3.7. TRACE_INTR_RAW_REG (0x0018)



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

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

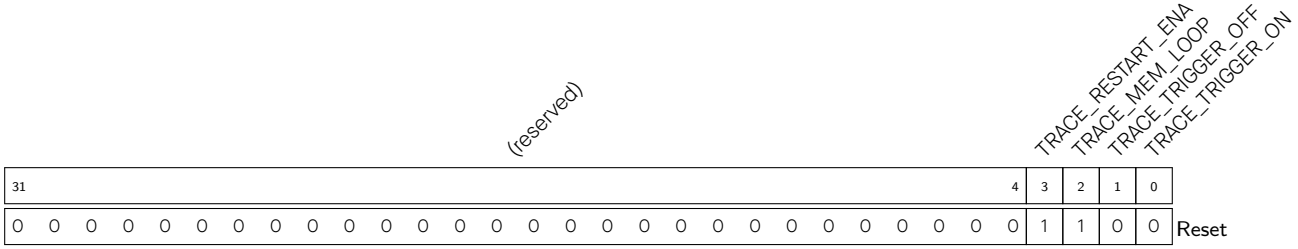
Register 3.8. TRACE_INTR_CLR_REG (0x001C)



TRACE_FIFO_OVERFLOW_INTR_CLR 写 1 清除 TRACE_FIFO_OVERFLOW_INTR。(WT)

TRACE_MEM_FULL_INTR_CLR 写 1 清除 TRACE_MEM_FULL_INTR。(WT)

Register 3.9. TRACE_TRIGGER_REG (0x0020)



TRACE_TRIGGER_ON 配置开启追踪编码器。

0：无效值，没有作用

1：开启

(WT)

TRACE_TRIGGER_OFF 配置关闭追踪编码器。

0：无效值，没有作用

1：关闭

(WT)

TRACE_MEM_LOOP 配置存储器写入模式。

0：非循环模式

1：循环模式

(R/W)

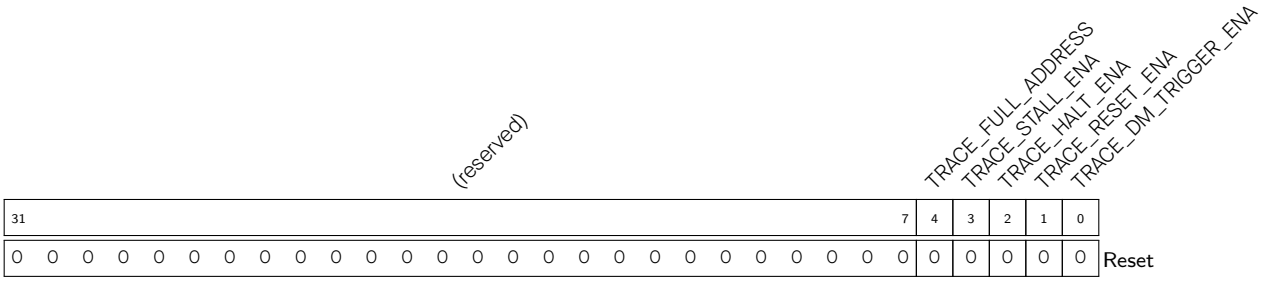
TRACE_RESTART_ENA 配置是否开启编码器自动重启功能。

0：关闭

1：开启

(R/W)

Register 3.10. TRACE_CONFIG_REG (0x0024)



TRACE_DM_TRIGGER_ENA 配置是否使能 trigger 信号。

0：禁用

1：使能

(R/W)

TRACE_RESET_ENA 配置是否使能 reset 信号。

0：禁用

1：使能

(R/W)

TRACE_HALT_ENA 配置是否使能 halted 信号。

0：禁用

1：使能

(R/W)

TRACE_STALL_ENA 配置是否使能 stall 信号。

0：禁用

1：使能

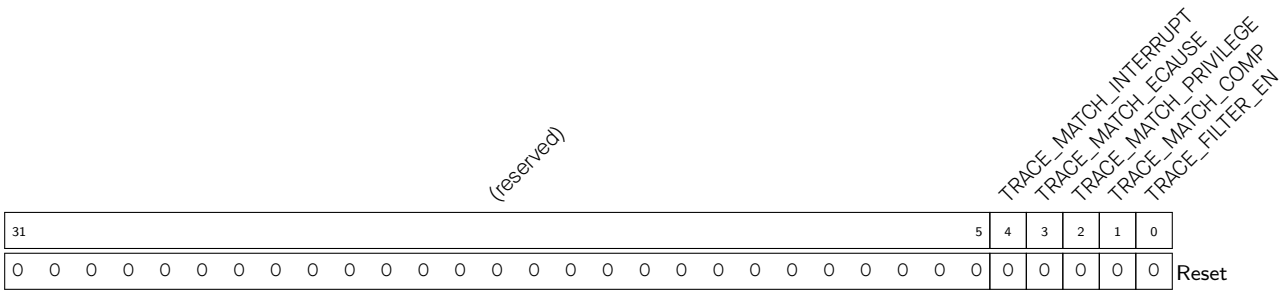
(R/W)

TRACE_FULL_ADDRESS 配置地址模式。

0：增量地址模式

1：完整地址模式

Register 3.11. TRACE_FILTER_CONTROL_REG (0x0028)



TRACE_FILTER_EN 配置是否使能过滤功能。

0：禁用

1：使能

(R/W)

TRACE_MATCH_COMP 配置是否使能比较器匹配模式。

0：禁用

1：使能

(R/W)

TRACE_MATCH_PRIVILEGE 配置是否使能特权匹配模式。

0：禁用

1：使能

(R/W)

TRACE_MATCH_ECAUSE 配置是否使能 ecause 匹配模式。

0：禁用

1：使能 (R/W)

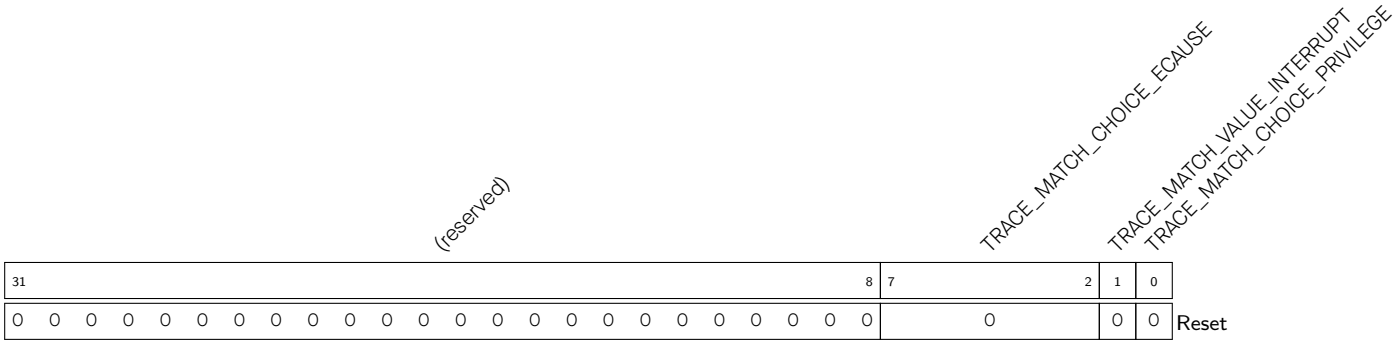
TRACE_MATCH_INTERRUPT 配置是否使能中断匹配模式。

0：禁用

1：使能

(R/W)

Register 3.12. TRACE_FILTER_MATCH_CONTROL_REG (0x002C)



TRACE_MATCH_CHOICE_PRIVILEGE 配置匹配的特权等级。

0：用户模式

1：机器模式

(R/W)

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

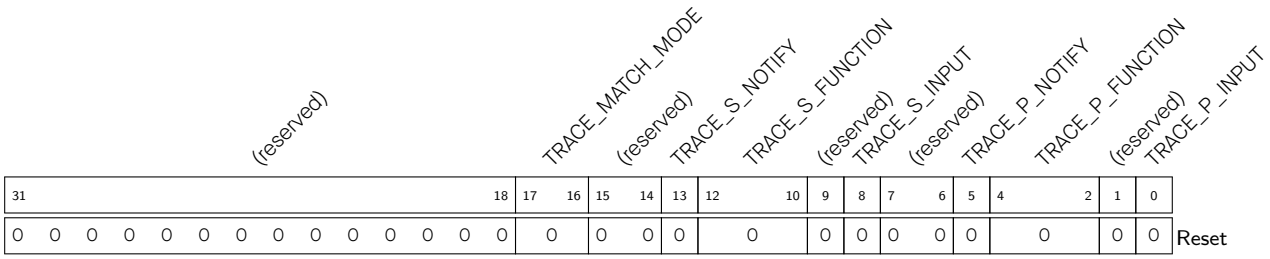
0：itype=1

1：itype=2

(R/W)

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

Register 3.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)



TRACE_P_INPUT 配置主比较器匹配的输入。

0：iaddr

1：tval

(R/W)

TRACE_P_FUNCTION 配置主比较器的运算符。

0：等于

1：不等于

2：小于

3：小于或等于

4: 大于

5：大于或等于

其他值：始终匹配

(R/W)

TRACE_P_NOTIFY 配置是否明确报告主比较器匹配的指令地址。

0：不报告

1：报告

(R/W)

TRACE_S_INPUT 配置次比较器匹配的输入。

0：iaddr

1：tval

(R/W)

TRACE_S_FUNCTION 配置次比较器的运算符。

0：等于

1：不等于

2：小于

3：小于或等于

4: 大于

5：大于或等于

6：次匹配值是主比较器的屏蔽值

其他值：始终匹配

(R/W)

见下页...

Register 3.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)

接上页...

TRACE_S_NOTIFY 配置是否明确报告次比较器匹配的指令地址。

0：不报告

1：报告

(R/W)

TRACE_MATCH_MODE 配置比较器的匹配条件。

0：仅主比较器匹配

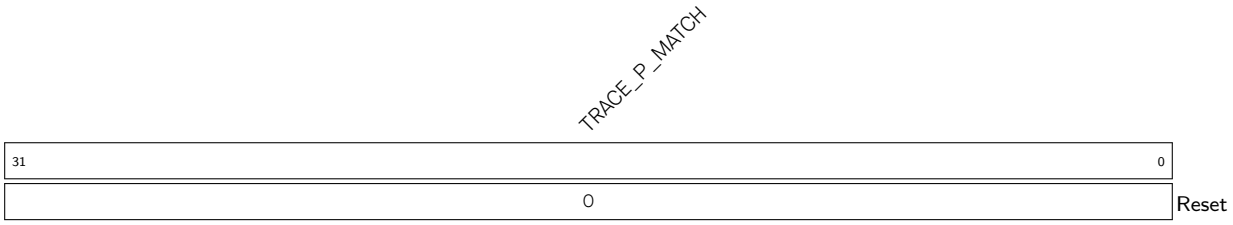
1：主比较器和次比较器都用于匹配 (P&&S)

2：主比较器和次比较器都不用于匹配 (!P&&S)

3：主比较器匹配时开始过滤，次比较器匹配时结束过滤

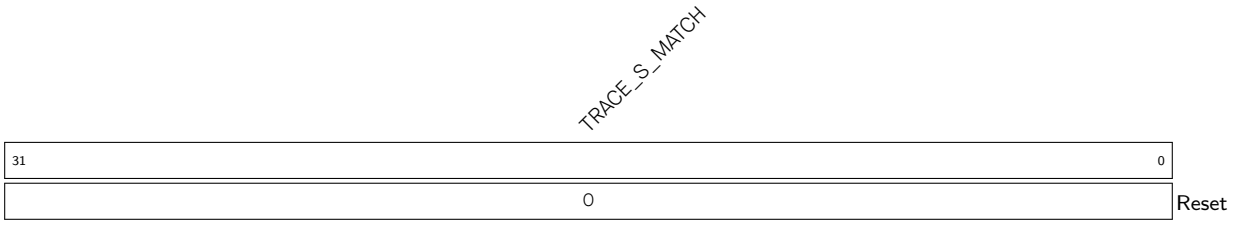
(R/W)

Register 3.14. TRACE_FILTER_P_COMPARATOR_MATCH_REG (0x0034)



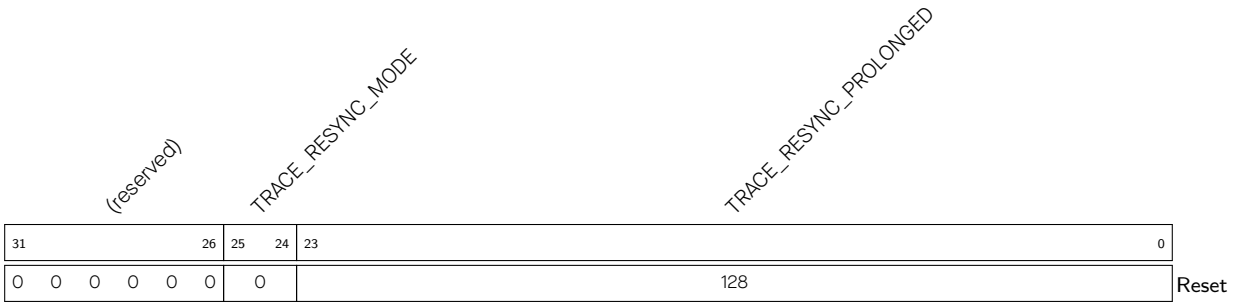
TRACE_P_MATCH 配置主比较器的匹配值。(R/W)

Register 3.15. TRACE_FILTER_S_COMPARATOR_MATCH_REG (0x0038)



TRACE_S_MATCH 配置次比较器的匹配值。(R/W)

Register 3.16. TRACE_RESYNC_PROLONGED_REG (0x003C)

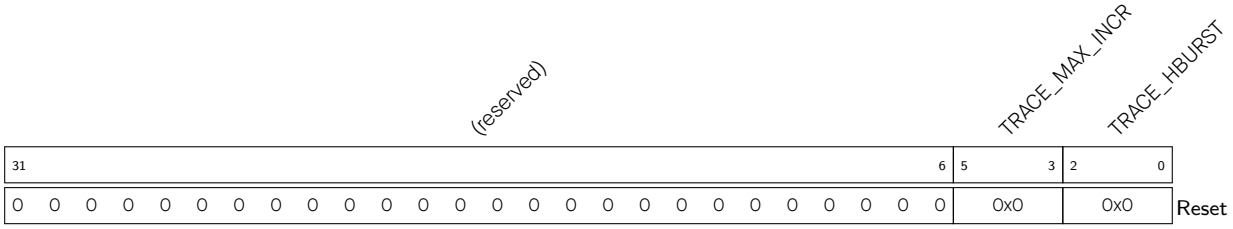


TRACE_RESYNC_PROLONGED 配置同步计数器的阈值。(R/W)

TRACE_RESYNC_MODE 配置同步模式。

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

Register 3.17. TRACE_AHB_CONFIG_REG (0x0040)

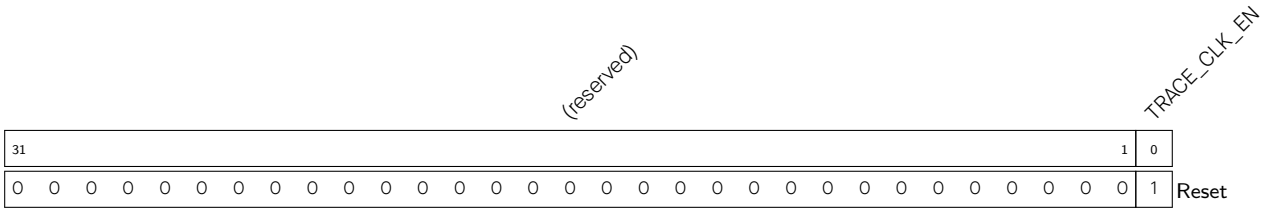


TRACE_HBURST 配置 AHB 突发传输模式。

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

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

Register 3.18. TRACE_CLOCK_GATE_REG (0x0044)



TRACE_CLK_EN 配置寄存器时钟门控。

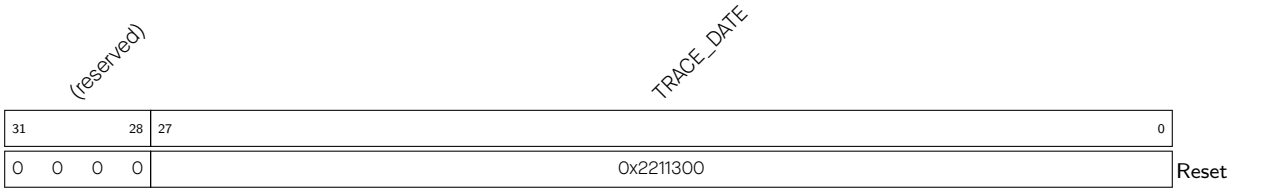
0：仅在应用写寄存器时开启时钟，从而降低功耗

1：一直强制为寄存器开启时钟

该位不影响寄存器访问。

(R/W)

Register 3.19. TRACE_DATE_REG (0x03FC)



TRACE_DATE 版本控制寄存器。(R/W)

第 4 章

低功耗处理器

ESP32-C5 低功耗处理器 (Low-Power CPU, LP CPU) 是基于 RISC-V ISA 的 32 位处理器，包括整数 (I)、乘法/除法 (M)、原子 (A) 和压缩 (C) 标准扩展。LP CPU 内核具有 2 级有序标量流水线，具有极低的功耗。CPU 内核架构包含中断控制器 (INTC)、调试模块 (DM) 和用于访问存储器和外设的系统总线 (SYS BUS) 接口。

LP CPU 默认处于睡眠状态（详见章节 4.9），在芯片进入 Deep-sleep 时可保持上电状态（详见章节 13 低功耗管理），能够访问绝大部分外设和存储器，详情请参考章节 6 系统和存储器。LP CPU 有两种应用场景：

- 功耗不敏感场景：高性能处理器 (HP CPU) 处于工作状态，LP CPU 可协助 HP CPU 完成一些对速度和效率不敏感的控制或计算
- 功耗敏感场景：HP CPU 处于掉电状态来节省功耗时，可以唤醒 LP CPU 处理一些外部唤醒事件

HP CPU 和 LP CPU 可访问的资源如图 4.0-1 所示：

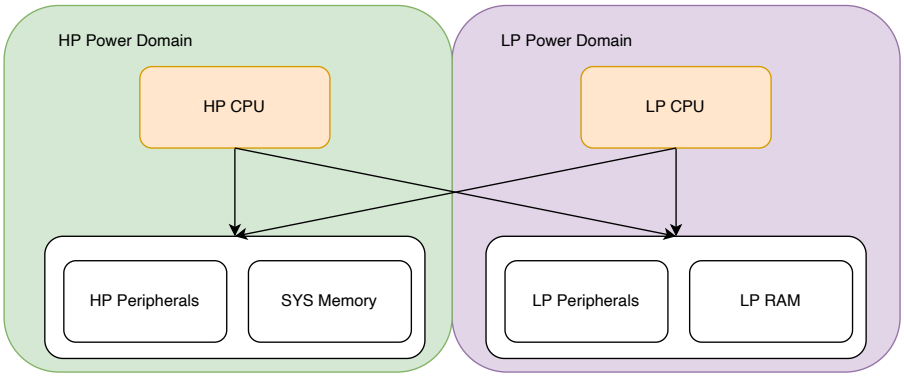


图 4.0-1. LP CPU 概览

在上图中，HP 电源域 (HP Power Domain) 和 LP 电源域 (LP Power Domain) 可分别上下电，当其中一个电源域处于掉电状态时，另一个电源域的 CPU 无法访问该域的资源，只有当该域处于上电状态时才可访问其资源，更多关于电源的信息，请参考章节 13 低功耗管理。

4.1 特性

LP CPU 具有以下特性：

- 时钟工作频率高达 20 MHz
- 支持 1 个向量中断入口
- 调试模块 (DM) 符合 RISC-V 调试规范 v0.13，支持通过行业标准的 JTAG/USB 端口连接外部调试器
- 硬件触发器符合 RISC-V 调试规范 v0.13，具有 2 个断点/观察点
- 32 位 AHB 系统总线，用于访问存储器和外设

- 支持核心性能指标事件
- 可唤醒 HP CPU 或向 HP CPU 发送中断
- 可访问 HP 存储器和 LP 存储器
- 可访问所有外设空间

4.2 配置与状态寄存器 (CSR)

4.2.1 寄存器列表

下表为 CPU 可访问的 CSR 列表。除了自定义的性能计数器 CSR 外，所有已实现的 CSR 都遵循 RISC-V 指令集手册 V1.10 第二卷“特权架构”(RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10) 中所述的位域标准映射。必须注意的是，受 CPU 中实现的功能子集的限制，即使在标准 CSR 中也并非实现了所有位域。有关详细的 CSR 寄存器描述，请参阅下一小节。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
机器模式信息 CSR			
mhartid	机器模式 hart ID 寄存器	0xF14	RO
机器模式陷阱设置 CSR			
mstatus	机器模式状态寄存器	0x300	R/W
misa [1]	机器模式 ISA 寄存器	0x301	R/W
mie	机器模中断使能寄存器	0x304	R/W
mtvec [2]	机器模式陷阱向量寄存器	0x305	R/W
机器模式陷阱处理 CSR			
mscratch	机器模式暂存寄存器	0x340	R/W
mepc	机器模式陷阱程序计数器	0x341	R/W
mcause [3]	机器模式陷阱原因寄存器	0x342	R/W
mtval	机器模式陷阱值寄存器	0x343	R/W
mip	机器模中断等待寄存器	0x344	R/W
触发器模块 CSR（与调试模块共用）			
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
调试模式 CSR			
dcsr	调试模式控制与状态寄存器	0x7B0	R/W
dpc	调试模式 PC 寄存器	0x7B1	R/W
dscratch0	调试模式暂存寄存器 0	0x7B2	R/W
dscratch1	调试模式暂存寄存器 1	0x7B3	R/W
机器模式计数器/定时器 CSR			
mcycle	机器模式时钟周期计数器寄存器	0xB00	R/W

[1]尽管 misa 具有 R/W 属性，但由于它的域是硬连线的，所以写操作无效。在 RISC-V 术语中称为 WARL（写入任意数值读取合法数值）。
[2]mtvec 仅支持在向量模式下对陷阱处理进行配置，基地址为 256 字节对齐。
[3]mcause 中反映的外部中断 ID 也包括 RISC-V 标准为处理器内部中断源预留的 ID。

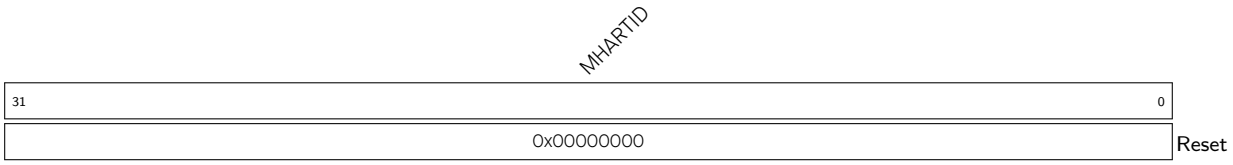
名称	描述	地址	访问
minstret	机器模式退役指令计数器寄存器	0xB02	R/W
mhpmcounter*n*(*n*:3-12)	机器模式性能监测计数器寄存器	0xB00+*n*	R/W
mcycleh	mcycle 的高 32 位	0xB80	R/W
minstreth	minstret 的高 32 位	0xB82	R/W
mhpmcounter*n*h(*n*:3-12)	mhpmcounter*n*(*n*:3-12) 的高 32 位	0xB80+*n*	R/W
机器模式计数器设置 CSR			
mcountinhibit	机器模式计数器控制寄存器	0x320	R/W

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

4.2.2 寄存器

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

Register 4.1. mhartid (0xF14)



MHARTID 表示硬件线程编号。LP CPU 硬件线程编号为 0。(RO)

Register 4.2. mstatus (0x300)



MIE 写 1 使能全局机器模式中断。(R/W)

MPIE 写 1 使能（陷阱）之前的机器模式中断。(R/W)

MPP 配置（陷阱）之前的机器特权模式。

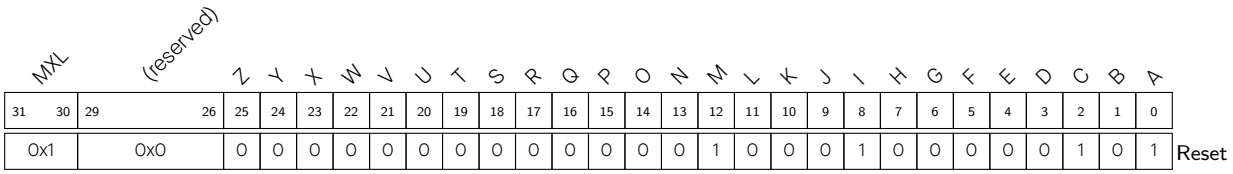
0x3：机器模式

其他值：无效

注意：仅低位可写。由于高位直接绑定低位，写入高位将被忽略。

(R/W)

Register 4.3. misa (0x301)



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

Reset

MXL 机器 XLEN = 1（32 位）。(RO)

Z 保留 = 0。(RO)

Y 保留 = 0。(RO)

X 非标准扩展 = 0。(RO)

W 保留 = 0。(RO)

V 保留 = 0。(RO)

U 实现用户模式 = 0。(RO)

T 保留 = 0。(RO)

S 实现监督模式 = 0。(RO)

R 保留 = 0。(RO)

Q 四精度浮点扩展 = 0。(RO)

P 保留 = 0。(RO)

O 保留 = 0。(RO)

N 支持用户级别中断 = 0。(RO)

M 整数乘除法标准扩展 = 1。(RO)

L 保留 = 0。(RO)

K 保留 = 0。(RO)

J 保留 = 0。(RO)

I RV32I 基本 ISA = 1。(RO)

H 虚拟机管理程序扩展 = 0。(RO)

G 其他标准扩展 = 0。(RO)

F 单精度浮点扩展 = 0。(RO)

E RV32E 基本 ISA = 0。(RO)

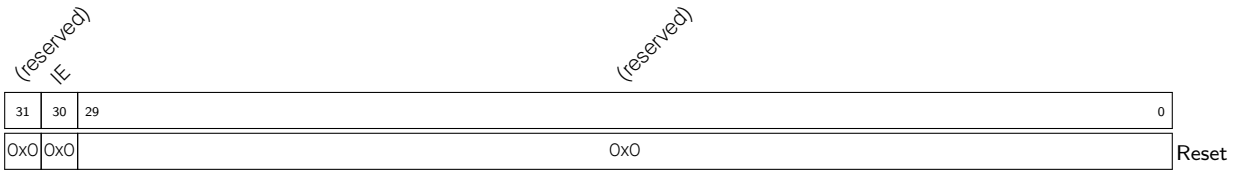
D 双精度浮点扩展 = 0。(RO)

C 压缩标准扩展 = 1。(RO)

B 保留 = 0。(RO)

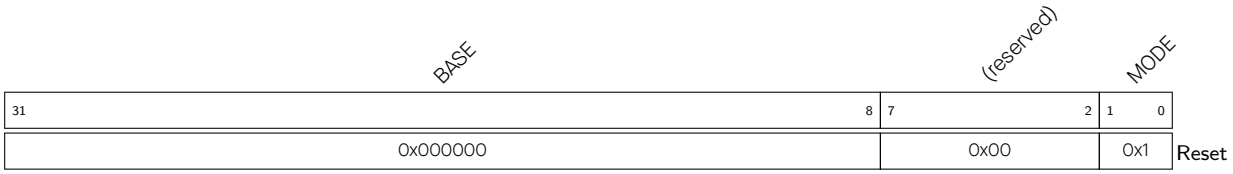
A 原子标准扩展 = 1。(RO)

Register 4.4. mie (0x304)



IE 写 1 使能中断。(R/W)

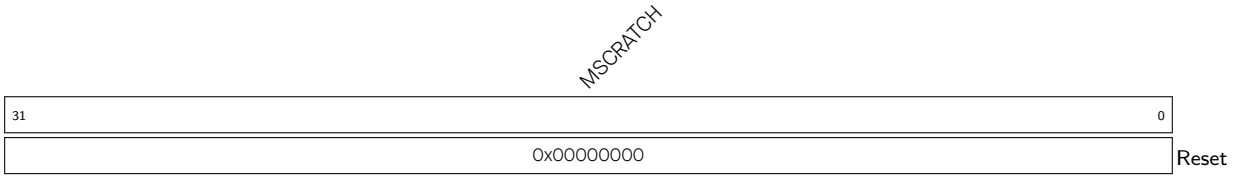
Register 4.5. mtvec (0x305)



MODE 表示机器模式中断是否为向量模式。仅支持向量模式 **0x1**。(RO)

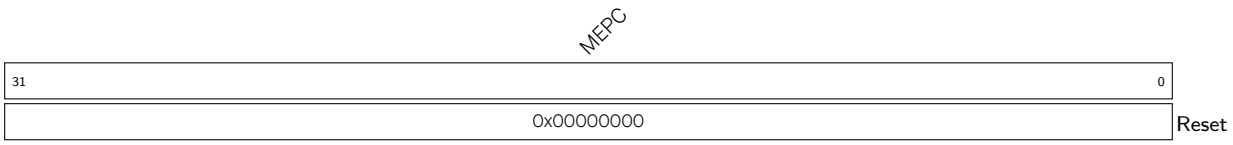
BASE 配置陷阱向量基址的高 24 位，地址为 256 字节对齐。(R/W)

Register 4.6. mscratch (0x340)



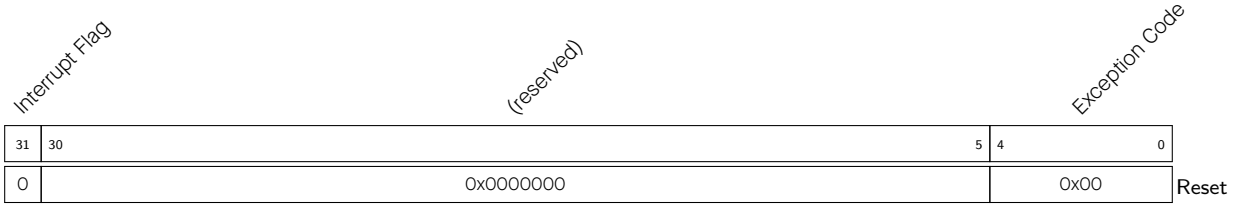
MSCRATCH 保存用户自定义的机器暂存信息。(R/W)

Register 4.7. mepc (0x341)



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

Register 4.8. mcause (0x342)



Exception Code CPU 进入异常时，此域将自动更新为最近的陷阱或中断的唯一 ID。可能的异常 ID：

0x2：非法指令

0x3：硬件断点/观察点或 EBREAK

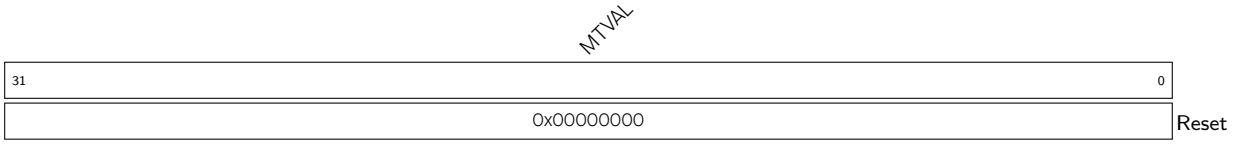
0x6：原子指令非对齐

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

(R/W)

Interrupt Flag CPU 进入陷阱时，此标志位将自动更新。如果被置位，则表示最近的陷阱是由中断引起。在异常情况下保持为 0。(R/W)

Register 4.9. mtval (0x343)



MTVAL 配置机器模式陷阱值。将自动更新为与异常有关的数据，该数据可能有助于处理该异常。根据异常编号有以下解读：

0x1：指令地址错误

0x2：指令 opcode 错误

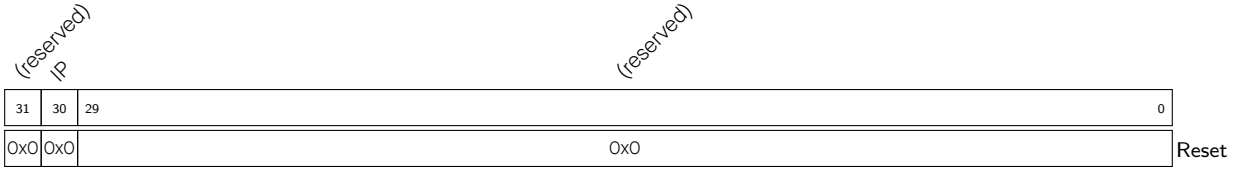
0x5：存储器读操作的数据地址错误

0x7：存储器写操作的数据地址错误

注意：该寄存器不支持其他异常 ID 和中断

(R/W)

Register 4.10. mip (0x344)



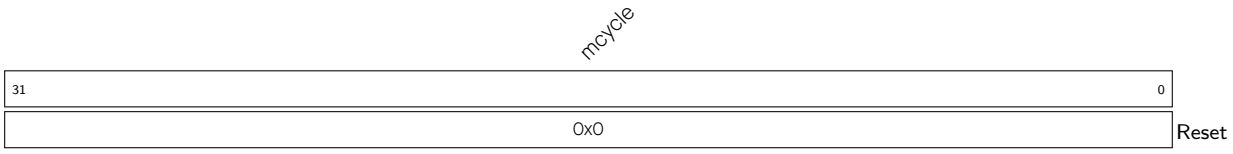
IP 配置中断的等待状态。

0：不等待

1：等待

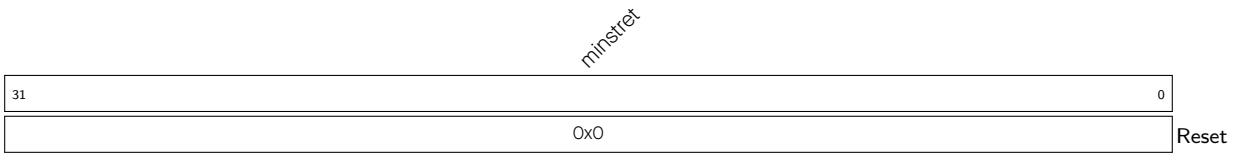
(R/W)

Register 4.11. mcycle (0xB00)



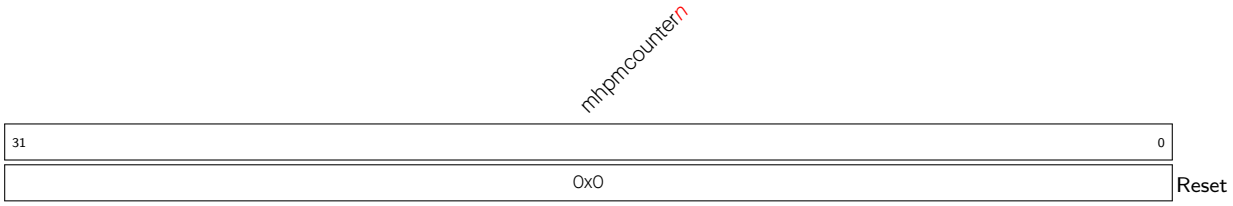
MCYCLE 配置时钟周期计数器的低 32 位。(R/W)

Register 4.12. minstret (0xB02)



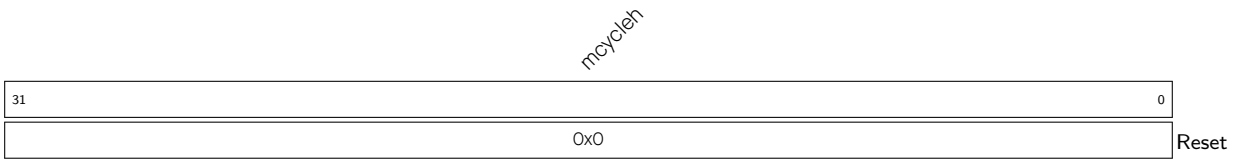
MINSTRET 配置指令计数器的低 32 位。(R/W)

Register 4.13. mhpmcounter*n*(*n*: 3-12) (0xB00+*n*)



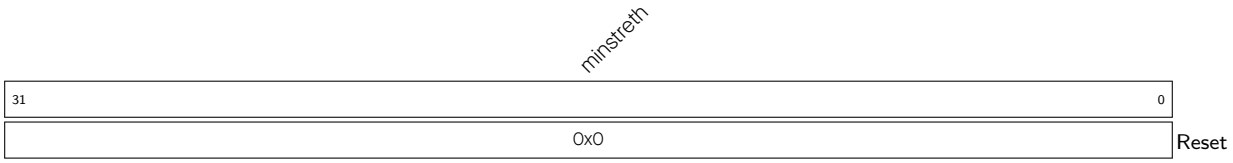
MHPMCOUNTER*n* 配置性能计数器 *n* 的低 32 位。(R/W)

Register 4.14. mcycleh (0xB80)



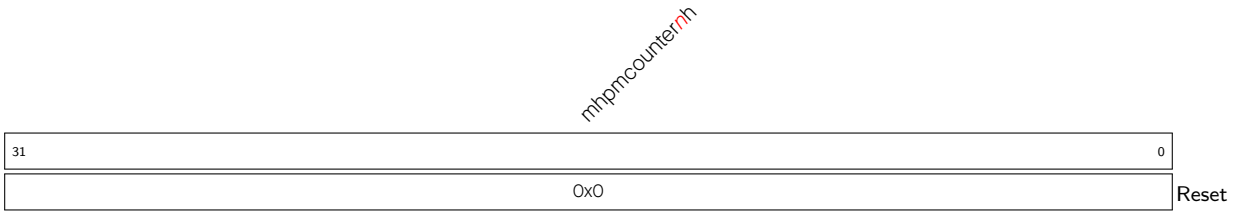
MCYCLEH 配置时钟周期计数器的高 32 位。(R/W)

Register 4.15. minstreth (0xB82)



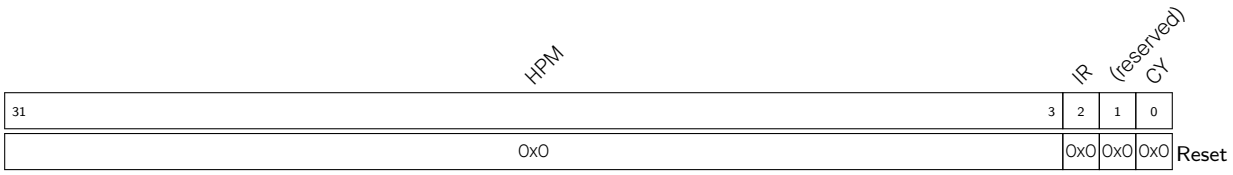
MINSTRETH 配置指令计数器的高 32 位。(R/W)

Register 4.16. mhpmcounter*n*(*n*: 3-12)h (0xB80+*n*)



MHPMCOUNTER*n*h 配置性能计数器 *n* 的高 32 位。(R/W)

Register 4.17. mcouninhibit (0x320)



CY 配置时钟周期计数器是否递增计数。

0：不计数

1：递增计数

(R/W)

IR 配置指令计数器是否递增计数。

0：不计数

1：递增计数

(R/W)

HPM 配置性能计数器 *n*(*n*:3-12) 是否递增计数。

0：不计数

1：递增计数

(R/W)

4.3 中断和异常

低功耗 CPU 根据 RISC-V 指令集手册 V1.10 第二卷 “特权架构” 实现了中断和异常的处理。当 CPU 进入中断/异常处理程序时，内核将执行以下操作：

- 将当前 PC 保存到 mepc CSR 中
- 将 mstatus 的 MIE 保存到 mstatus 的 MPIE
- 将当前特权模式保存到 mstatus 的 MPP
- 将 mstatus 的 MIE 清 0
- 特权模式切换到机器模式
- 跳转到处理程序地址
  - 对于所有异常原因，跳转到 mtvec CSR 中向量表的基地址
  - 对于中断，跳转到 $mtvec + 4 * 30$
- 执行 mret 指令后，内核跳转到之前保存在 mepc CSR 中的程序计数器，并将 mstatus 的 MIE 恢复为 mstatus 的 MPIE，特权模式切换到 MPP 对应的特权模式

当内核启动时，向量表的基地址被初始化为启动地址 0x50000000。启动后可通过写入 mtvec CSR 更改基地址。有关配置和状态寄存器的详细信息，请参阅章节 4.2.1。

内核在复位后从 0x50000080 的地址开始取指。

4.3.1 中断

ESP32-C5 低功耗 CPU 仅支持 1 个中断入口，所有的中断事件都会跳转到该入口。ESP32-C5 低功耗 CPU 支持中断的外设源有：

- 电源管理单元 (PMU)
- 低功耗定时器 (RTC_TIMER)
- 低功耗 UART (LP_UART)
- 低功耗 I2C (LP_I2C)
- 低功耗 IO MUX (LP IO MUX)

各外设的中断请参考各外设章节。

4.3.2 中断处理

默认情况下，全局禁用中断，因为 mstatus 的 MIE 的复位值为 0。软件必须设置该位以启用全局中断。

1. 中断使能
   - 置位 mstatus 的 MIE 域，启用全局中断使能
   - 置位 mie CSR 的第 30 位，使能中断 30
2. 中断使能后，LP CPU 即可响应中断，同时去配置相应外设的中断，使其能够触发中断信号给 LP CPU。
3. 触发中断后 LP CPU 会跳转到 $mtvec + 4 * 30$。

4. 进入中断服务程序 (ISR) 后，用户需要查询 LPPERI_INTERRUPT_SOURCE_REG 获取触发中断的外设，并进行处理，注意如果有多个外设触发，依次处理每一个中断，直到所有的外设都处理完成，如果未处理完所有中断，会再次进入中断服务程序。
5. 中断清除，只需要清除各外设的中断信号即可。

4.3.3 异常

低功耗 CPU 支持 RISC-V 标准异常，能够触发以下异常：

表 4.3-1. LP CPU 异常原因

异常编号	描述
2	非法指令
3	断点 (EBREAK)
6	原子指令非对齐

4.4 调试

本节介绍如何调试和测试低功耗 CPU。调试功能由标准 JTAG 管脚提供，并符合 RISC-V 外部调试支持规范 v0.13 (RISC-V External Debug Support, Version 0.13)。

ESP32-C5 系统调试结构请参考章节 2 高性能处理器 中的图 2.10-1。

用户与运行调试器（Debugger，例如 GDB）的调试主机（DEBUG HOST，例如笔记本电脑）进行交互。调试器通过调试转换器（Debug Translator，可能包含硬件驱动，例如 OpenOCD）与调试传输硬件（DEBUG TRANSPORT，例如 ESP-Prog 适配器）进行通信。调试传输硬件通过标准 JTAG 接口将调试主机连接到处理器内核的调试传输模块 (JTAG DTM)。JTAG DTM 使用调试模块接口 (DMI) 提供对调试模块 (DM) 的访问。

ESP32-C5 有两个 DTM 通过菊花链结构互联，其中 TAP0 用于连接 HP CPU，TAP1 用于连接 LP CPU。

HP 调试模块 (DM) 调试单核 HP CPU。LP 调试模块 (DM) 调试单核 LP CPU。

低功耗 CPU 实现了四个内核调试寄存器：dcsr、dpc、dscratch0 和 dscratch1。所有这些寄存器只能从调试模式访问，如果软件试图在内核未处于调试模式的情况下进行访问，则会触发非法指令异常。

4.4.1 特性

低功耗 CPU 支持以下调试特性：

- 向调试器提供有关实现的必要信息
- 支持暂停和恢复 CPU 内核
- CPU 内核寄存器（包括 CSR）可以由调试器读取/写入
- 可以通过调试器复位 CPU 内核
- 可以在软件断点（植入的断点指令）上暂停 CPU
- 硬件单步调试
- 支持两个硬件触发器（可用作断点/观察点），具体见章节 4.5

4.4.2 功能描述

调试机制遵守 RISC-V 外部调试支持规范 v0.13。有关调试功能的详细介绍，请参考 RISC-V 外部调试支持规范 v0.13 (RISC-V External Debug Support, Version 0.13)。

根据 RISC-V 外部调试支持规范，hart 存在 nonexistent、unavail、running、halted 等状态，低功耗 CPU 默认处于 unavail 状态，需要用户配置 LPPERI_CPU_REG 寄存器清除 unavail 状态，从而能够连接低功耗 CPU 进行调试。

4.4.3 寄存器列表

下表列出了低功耗 CPU 支持的调试 CSR。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

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

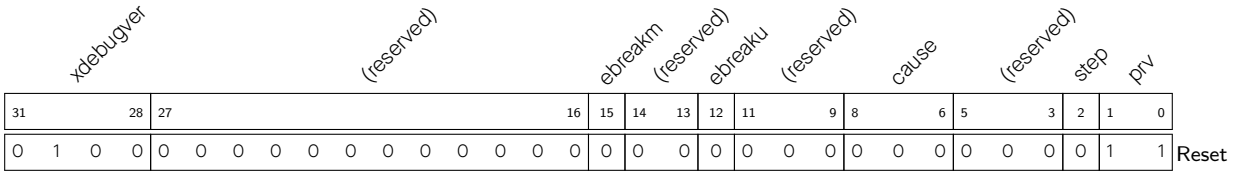
所有调试模块寄存器的实现均符合 RISC-V 外部调试支持规范 v0.13。请参考 RISC-V 外部调试支持规范 v0.13 获取详细信息。

4.4.4 寄存器

以下是低功耗 CPU 支持的调试 CSR 的详细描述。

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

Register 4.18. dcsr (0x7B0)



xdebugver 表示调试版本。

4：存在外部调试支持

(RO)

ebreakm 配置机器模式下执行 EBREAK 指令的行为。

0：触发异常，mcause 是 3

1：进入调试模式

(R/W)

ebreaku 配置用户模式下执行 ebreak 指令的行为。

0：如特权模式所述，触发异常，mcause 是 3

1: 进入调试模式

(R/W)

cause 表示进入调试模式的原因。当单个周期中有多个原因导致进入调试模式，会反映出具有最高优先级数值的那个原因。

1：执行了一条 ebreak 指令（优先级 3）

2：触发模块引起暂停（优先级 4）

3：haltreq 被置位（优先级 2）

4：step 被置位导致 CPU 单步执行（优先级 1）

其他值：保留，供日后使用

(RO)

step 当被置位且不处于调试模式时，内核将仅执行单个指令，然后进入调试模式。

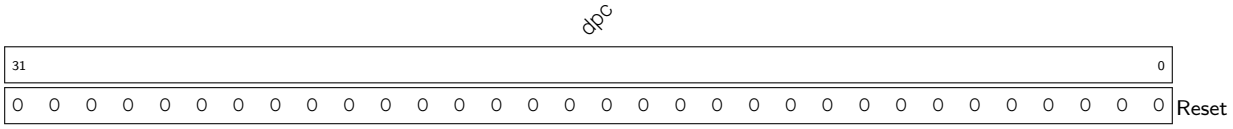
如果指令由于异常而未能完成，则内核将在执行异常处理程序之前立即进入调试模式，并置位相应的异常寄存器。

设置该位不会屏蔽中断，这与 RISC-V 外部调试支持规范 v0.13 中的规定不同。

(R/W)

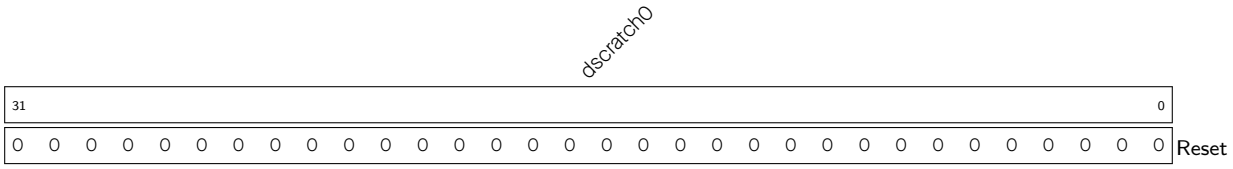
prv 保存 CPU 进入调试模式时候的特权级别。退出调试模式时，调试器可以更改此值以改变内核的特权级别。仅支持 **0x3**（机器模式）和 **0x0**（用户模式）。(RO)

Register 4.19. dpc (0x7B1)



dpc 进入调试模式时，dpc 会更新为将要执行的下一条指令的地址。恢复时，PC 设置为存储在 dpc 中的地址。调试模式可能会修改 dpc。可在调试模式下访问。(R/W)

Register 4.20. dscratch0 (0x7B2)



dscratch0 供调试模块内部使用。(R/W)

Register 4.21. dscratch1 (0x7B3)



dscratch1 供调试模块内部使用。(R/W)

4.5 硬件触发器

4.5.1 特性

硬件触发器模块提供了断点和观察点功能，供调试使用。硬件触发器具有以下特性：

- 2 个独立触发单元
- 每个单元都可以配置为匹配程序计数器的地址
- 可以暂停执行并将控制权转移给调试器

4.5.2 功能描述

硬件触发器模块提供了 3 个 CSR，详情见章节 4.5.5。其中，tdata1 和 tdata2 是抽象 CSR，也就是说它们是用于访问某个触发单元中的内部寄存器的影子寄存器，一次访问一个触发单元。

要选择特定的触发单元，需要将相应的编号 (0-1) 写入 tselect CSR。当写入有效数值时，抽象 CSR tdata1 和 tdata2 将自动匹配该触发单元的内部寄存器。每个触发单元都有两个内部寄存器，即 mcontrol 和 maddress，它们分别与 tdata1 和 tdata2 匹配。

向 tselect 写入超过 1 的数值时会导致 tselect 写入 1。

由于软件或调试器可能需要知道所选触发器的类型以便正确解读 tdata1 和 tdata2，因此 tdata1 的 4 个位 (31-28) 对所选触发器的类型进行了编码。此域为只读访问属性，并且值始终为 0x2，代表触发器仅支持地址/数据匹配。因此，可以推断 tdata1 和 tdata2 会通过 mcontrol 和 maddress 被解读。RISC-V 外部调试支持规范 v0.13 提供了其他可能值的信息，但是该触发模块仅支持 0x2 类型。

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

4.5.3 触发执行流程

当触发器触发引起硬件线程暂停并进入调试模式时 (action = 1)：

- dpc 被设置为当前 PC（在解码阶段）
- dcsr 的 cause 域被设置为 2，表示暂停是由于触发器触发引起

4.5.4 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
mcontrol	tdata1 影子寄存器	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W

4.5.5 寄存器

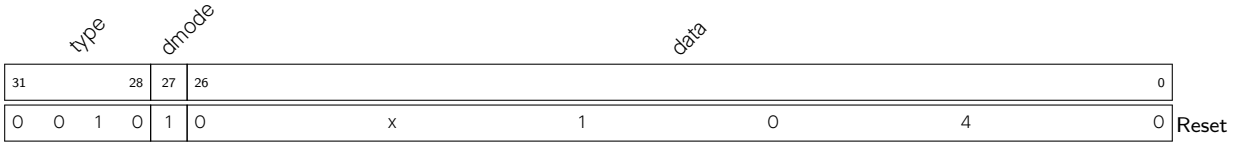
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 4.22. tselect (0x7A0)



tselect 配置选择触发器单元。(R/W)

Register 4.23. tdata1 (0x7A1)



type 表示触发器类型。仅支持匹配类型 (0x2)，此域保留。(RO)

dmode 如果某触发器正在被调试器使用，则此域置为 1。仅支持调试模式使用，此域保留。(RO)

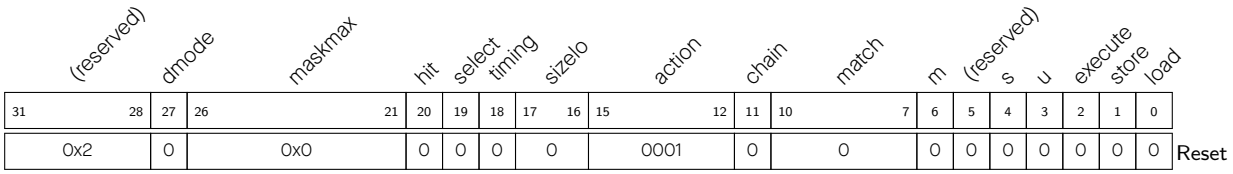
data 配置抽象 tdata1 的内容。由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 mcontrol 的域。(R/W)

Register 4.24. tdata2 (0x7A2)



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

Register 4.25. mcontrol (0x7A1)



31–28	27	26–21	20	19	18	17–16	15–12	11	10–7	6	5	4	3	2	1	0	
(reserved)	dmode	maskmax	hit	select	timing	sizelo	action	chain	match	m	(reserved)	s	u	execute	store	load	
0x2	0	0x0	0	0	0	0	0001	0	0	0	0	0	0	0	0	0	Reset

dmode 与 tdata1 的 dmode 一致。(RO)

maskmax 表示硬件支持的最大自然对齐的二次幂范围。

0：一个字节的范围，仅支持精确匹配

其他值：不支持

(RO)

hit 硬件未实现，始终为 0。(RO)

select 选择地址匹配或数据匹配。

0：匹配虚拟地址

1：匹配读取或写入的数据值或执行的指令

注意：只实现了地址匹配，此域总为 0。

(RO)

timing 表示触发器何时被触发。

0：当匹配后，触发器会在指令执行前触发

1：当匹配后，触发器会在指令执行后触发

注意：此域总为 0。

(RO)

sizelo 仅支持任意大小匹配，此域总为 0。(RO)

action 配置选定的触发器在触发时进行以下操作。

0x0：引起断点异常

0x1：进入调试模式（仅当 dmode = 1 时有效）

注意：仅支持进入调试模式，此域总为 1。

(RO)

CHAIN 硬件未实现，始终为 0。(RO)

match 配置触发器进行数据/指令地址的匹配操作。

0x0：精确字节匹配，即与访问中某个字节对应的地址必须精确匹配 maddress 的值。0x1：NAPOT 匹配，即访问中至少有一个字节处于 maddress 中规定的 NAPOT 区域。

注意：仅支持精确字节匹配，此域总为 0。

(R/W)

m 置位使选定的触发器在机器模式下操作。(RO)

s 置位使选定的触发器在监督模式下操作。不支持在监督模式下操作，此域总为 0。(RO)

u 置位使选定的触发器在用户模式下操作。不支持在用户模式下操作，此域总为 0。(RO)

见下页……

Register 4.25. mcontrol (0x7A1)

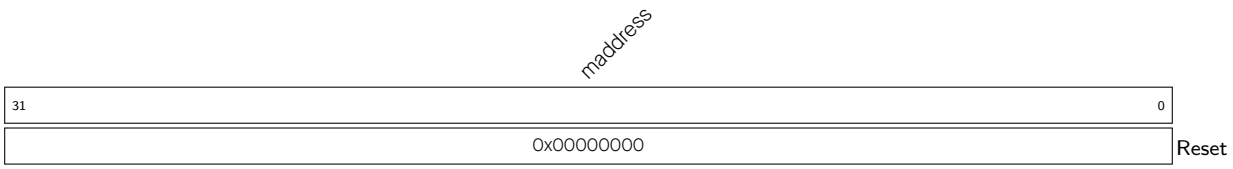
接上页……

execute 置位使选定的触发器匹配指令的虚拟地址。(R/W)

store 置位使选定的触发器匹配存储器写操作的虚拟地址。硬件不支持，此域总为 0。(RO)

load 置位使选定的触发器匹配存储器读操作的虚拟地址。硬件不支持，此域总为 0。(RO)

Register 4.26. maddress (0x7A2)



maddress 配置选定的触发器执行匹配操作时使用的地址。(R/W)

4.6 性能计数器

低功耗 CPU 实现了时钟周期计数器 mcycle(h)、指令计数器 minstret(h) 以及 10 个事件计数器 mhpmcounter*n*(*n*:3-12) 。时钟周期计数器和指令计数器始终可用且为 64 位，其余性能计数器为 40 位。

默认情况下，所有计数器在复位后启用。通过 mcountinhibit CSR 中的相应位来单独启用/禁用对应的计数器。

每个计数器的计数事件都是固定的，参考表 4.6-1。

表 4.6-1. 性能计数器

计数器	计数事件
mcycle	统计时钟周期数
minstret	统计指令数量
mhpmcounter3	统计内存访问的等待周期
mhpmcounter4	统计取指等待周期
mhpmcounter5	统计内存读操作的数量，非对齐读视作两次。
mhpmcounter6	统计内存写操作的数量，非对齐写视作两次。
mhpmcounter7	统计无条件跳转指令的数量 (jal, jr, jalr)
mhpmcounter8	统计分支指令的数量
mhpmcounter9	统计分支指令跳转的数量
mhpmcounter10	统计压缩指令的数量
mhpmcounter11	统计乘法指令的等待周期
mhpmcounter12	统计除法指令的等待周期

4.7 系统访问

4.7.1 存储器

ESP32-C5 低功耗 CPU 可以访问的存储器资源有 LP SRAM 和 HP SRAM，详情参考章节 6 系统和存储器。其中：

- LP SRAM：16 KB, 地址范围为 0x5000_0000 ~ 0x5000_3FFF，可进行取指、读数据、写数据等操作
- HP SRAM：384 KB, 地址范围为 0x4080_0000 ~ 0x4085_FFFF，可进行取指、读数据、写数据等操作

说明：
LP CPU 访问 HP SRAM 的延迟很大，但可以无延迟访问 LP SRAM。

LP CPU 支持原子指令集，LP CPU 和 HP CPU 可通过原子指令访问存储器，从而实现内存访问的原子性。关于原子指令集的细节请参考 RISC-V 指令集手册 v2.2 第一卷“非特权架构”(RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Version 2.2)。

注意：只有 HP SRAM 可以支持 HP CPU 和 LP CPU 的原子访问。

4.7.2 外设

章节 6 系统和存储器 表6.3-2 详细列出了低功耗 CPU 可访问的外设及基地址。

4.8 事件任务矩阵功能

在 ESP32-C5 中，LP CPU 支持事件任务矩阵 (ETM) 功能，即可以通过任意外设的 ETM 事件触发 LP CPU 的 ETM 任务，或者通过 LP CPU 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与 LP CPU 相关的 ETM 任务和 ETM 事件。

LP CPU 可接收的 ETM 任务有：

- ULP_TASK_WAKEUP_CPU：唤醒 LP CPU

LP CPU 可产生的 ETM 事件有：

- ULP_EVT_ERR_INTR：表示 LP CPU 发生异常
- ULP_EVT_START_INTR：表示 LP CPU 时钟打开

4.9 睡眠和唤醒流程

4.9.1 特性

- 低功耗 CPU 工作在低功耗系统中，可以在高性能系统工作或进入睡眠的时候，独立睡眠、唤醒及工作
- 低功耗 CPU 可以根据软件运行状态，主动配置寄存器进入睡眠状态
- 低功耗 CPU 可以被如下事件唤醒：
  - 主核置位寄存器 PMU_HP_TRIGGER_LP

- LP IO 的中断状态
- ETM 事件
- RTC 定时器超时
- LP UART 接收一定数量的 RX 脉冲，需要使能 LPPERI_LP_UART_WAKEUP_EN

4.9.2 流程

低功耗 CPU 默认处于睡眠状态，低功耗 CPU 的睡眠唤醒模块将按照如下流程唤醒开始工作以及进入睡眠。

LP CPU 的唤醒源配置如表 4.9-1 所示。

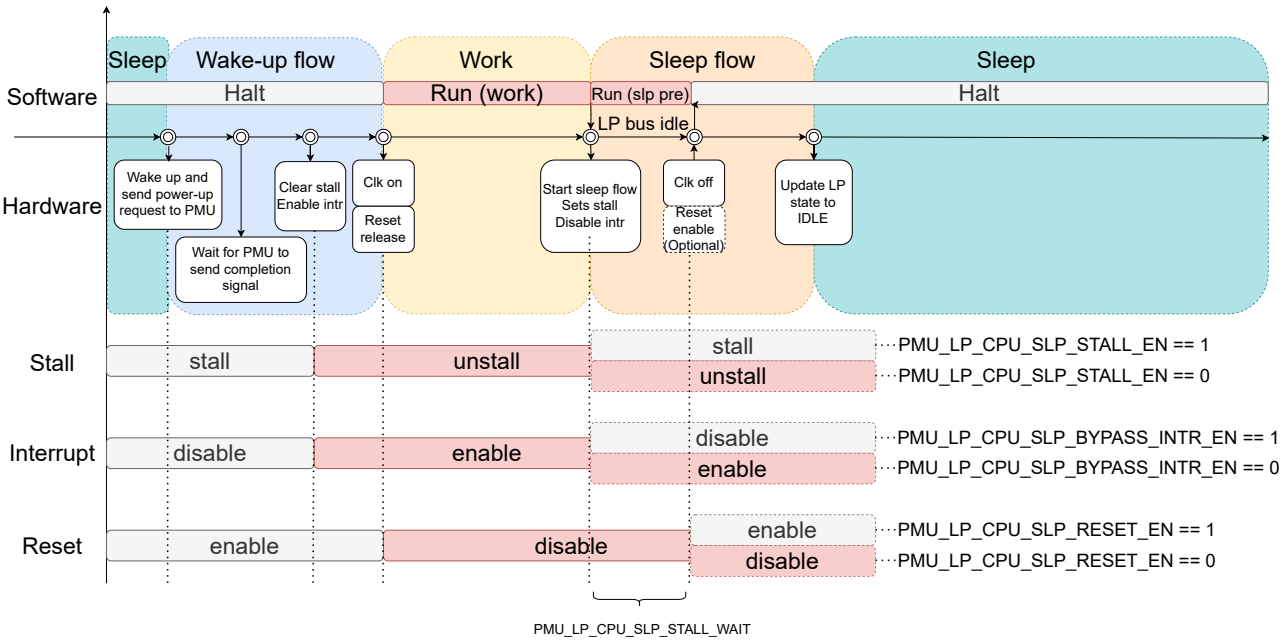


图 4.9-1. LP CPU 唤醒和睡眠流程

LP CPU 上电后的首次启动取决于 HP CPU 对能唤醒使能信号和唤醒源的配置。

- LP CPU 的初始化
  - 初始化 LP 存储器。
  - 启动 LP CPU。由于 LP CPU 的启动依赖于唤醒过程，所以推荐使用 PMU_HP_TRIGGER_LP 寄存器来开始 LP CPU 的初始化，方法如下：
    * 将 PMU_LP_CPU_WAKEUP_EN 设置为 0x1
    * 将 PMU_HP_TRIGGER_LP 设置为 0x1
    * LP CPU 被唤醒后开始运行
- 唤醒启动流程：
  - 睡眠唤醒模块接收到唤醒信号向 PMU 发送上电请求。
  - 如果当前功耗状态（时钟，电源等）已经满足低功耗 CPU 需求，则 PMU 立刻回复完成信号，否则将调整功耗状态再回复完成信号。

- 睡眠唤醒模块关闭低功耗 CPU 的 STALL 状态，并打开中断接收使能。
- 睡眠唤醒模块开启时钟并释放复位（如果睡眠没有使能复位则忽略该步骤），开始工作。

- 睡眠流程：
  - 低功耗 CPU 配置寄存器 PMU_LP_CPU_SLEEP_REQ，睡眠唤醒模块启动睡眠流程。
  - 如果 PMU_LP_CPU_SLP_STALL_EN 为 1，睡眠唤醒模块则使能低功耗 CPU 的 STALL 状态，如果为 0，则不使能低功耗 CPU 的 STALL 状态。
  - 睡眠唤醒模块等待 PMU_LP_CPU_SLP_STALL_WAIT 个低功耗 CPU 时钟周期后，关闭低功耗 CPU 的时钟，如果 PMU_LP_CPU_SLP_RESET_EN 为 1，则使能低功耗 CPU 的复位。

4.9.3 唤醒源

表 4.9-1. 唤醒源

寄存器值[1]	唤醒源	描述
0x1	寄存器 PMU_HP_TRIGGER_LP	通过主核置位寄存器 PMU_HP_TRIGGER_LP 可唤醒低功耗 CPU，置位 PMU_HP_SW_TRIGGER_INT_CLR 可清除唤醒源
0x2	LP UART	LP UART 接收一定数量的 RX 脉冲，需要使能 LPPERI_LP_UART_WAKEUP_EN，详情可见章节 *32 UART 控制器 (UART)*
0x4	LP IO	该唤醒源使用 LP IO 的中断状态寄存器信号，详情可见章节 *8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*
0x8	ETM	可接收来自 ETM 的唤醒源，详情可见章节 *12 事件任务矩阵 (ETM)*
0x10	RTC 定时器	使用 RTC 定时器的 target 1 超时中断控制，详情可见章节 *13 低功耗管理*

[1] 寄存器 PMU_LP_CPU_WAKEUP_EN 的值

4.10 寄存器列表

本小节的所有地址均为相对于低功耗外设的基地址的地址偏移量（相对地址），具体基地址请见章节 *6 系统和存储器* 中的表 6.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

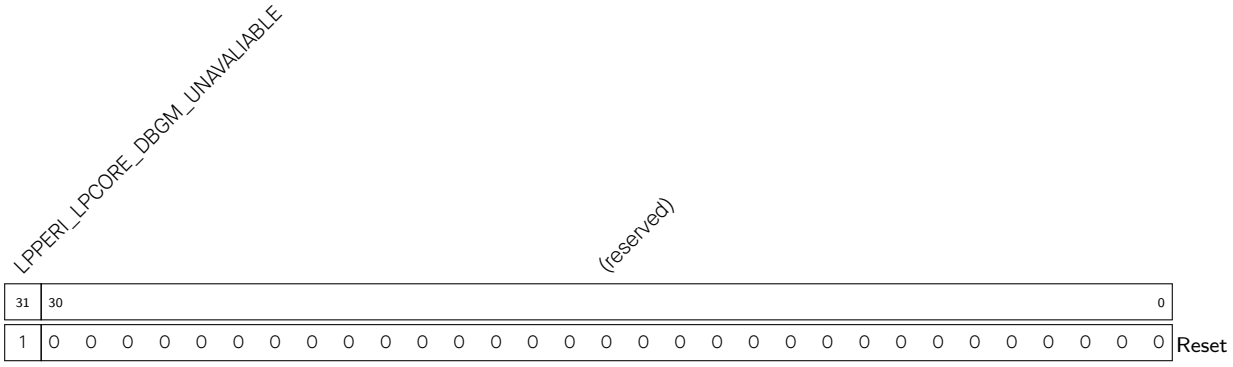
名称	描述	地址	访问
LPPERI_CPU_REG	LP CPU 控制寄存器	0x000C	R/W
LPPERI_INTERRUPT_SOURCE_REG	LP CPU 中断状态寄存器	0x0020	RO

4.11 寄存器

本小节的所有地址均为相对于低功耗外设的基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 4.27. LPPERI_CPU_REG (0x000C)



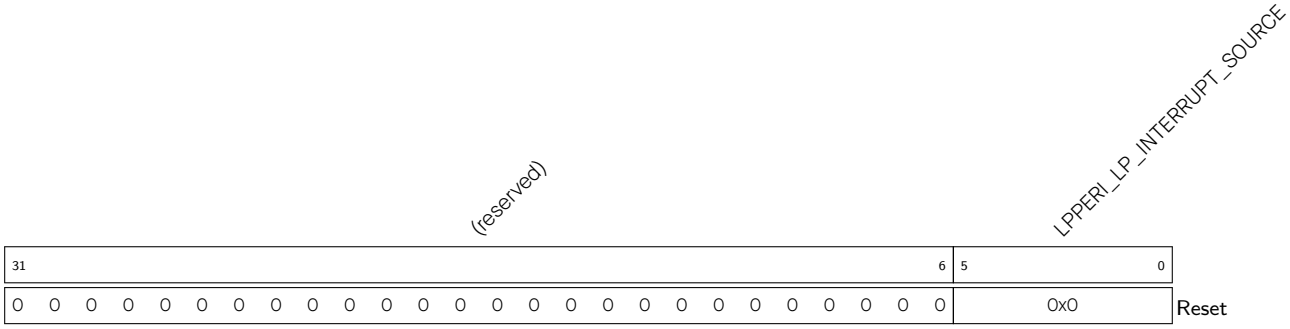
LPPERI_LPCORE_DBGM_UNAVALIABLE 配置 LP CPU 状态。

0：LP CPU 可以连接 JTAG

1：LP CPU 不可用，不能连接 JTAG

(R/W)

Register 4.28. LPPERI_INTERRUPT_SOURCE_REG (0x0020)



LPPERI_LP_INTERRUPT_SOURCE 表示 LP 中断源。

Bit 5：PMU_LP_INT

Bit 4：保留

Bit 3：RTC_TIMER_LP_INT

Bit 2：LP_UART_INT

Bit 1：LP_I2C_INT

Bit 0：LP_IO_INT

(RO)

第 5 章

通用 DMA 控制器 (GDMA)

5.1 概述

通用直接存储访问 (General Direct Memory Access, GDMA) 用于在外设与存储器之间以及存储器与存储器之间提供高速数据传输。软件可以在无需 CPU 干预的情况下通过 GDMA 快速搬移数据，从而降低了 CPU 的工作负载，提高了效率。

GDMA 有 6 个独立的通道，其中包括 3 个发送通道和 3 个接收通道。这 6 个通道被支持 GDMA 功能的外设所共享，用户可以将通道分配给任何支持 GDMA 功能的外设。GDMA 支持的外设包括 PARLIO、ADC、UHCI、AES、SHA、I2S、通用 SPI (GP-SPI)。ESP32-C5 的 GDMA 可以访问片内存储器和片外存储器。

GDMA 控制器支持通道间可配置优先级及权重仲裁以管理外设不同的带宽需求。

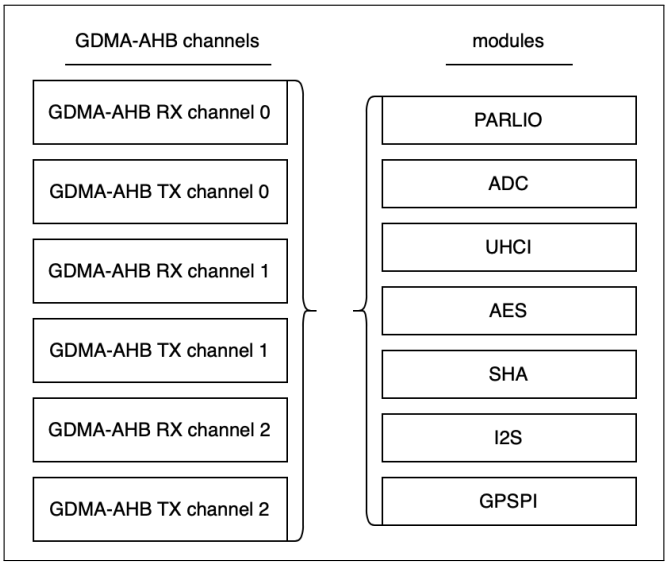


图 5.1-1. 共用 GDMA 通道的模块

5.2 特性

GDMA 具有以下特性：

- AHB 总线架构
- 数据传输以字节为单位，传输数据量可软件编程
- 支持任意地址和大小 (size) 访问
- 对齐要求：
  - 描述符存储地址：字对齐

  - 数据地址和长度：无要求
- 支持链表
- 访问存储器时，支持 INCR 突发传输
- 三个传输通道和三个接收通道
- 可配置支持 GDMA 的外设连接任一通道
- 支持通道间优先级和权重仲裁配置
- 支持存储搬运功能

5.3 架构

ESP32-C5 中所有需要进行高速数据传输的模块都具有 GDMA 功能。GDMA 控制器与 CPU 的数据总线使用相同的地址空间访问存储器。图 5.3-1 为 GDMA 控制器基本架构图。

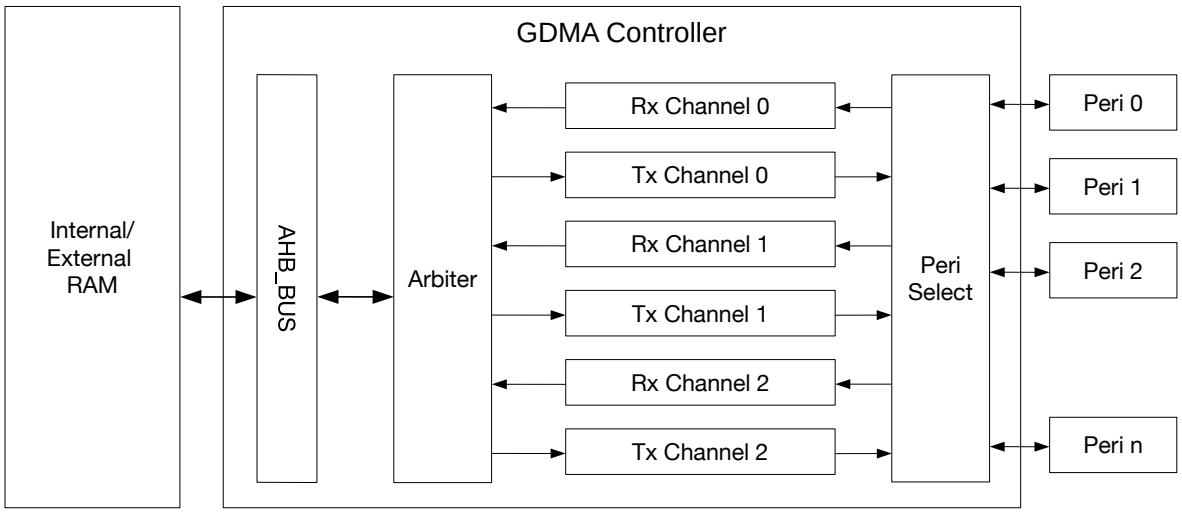


图 5.3-1. GDMA 控制器的架构

GDMA 控制器有 6 个独立的通道，其中包括 3 个发送通道和 3 个接收通道。每个通道可选择与不同的外设相连，从而实现通道资源由外设共享。

GDMA 通过 AHB_BUS 将数据存入存储器，或者将数据从存储器取出。在通过总线传输数据之前，GDMA 采用可配置的仲裁机制对每个通道的读写请求进行仲裁。存储器的具体使用范围详见章节 6 *系统和存储器*。

软件可以通过挂载链表的方式来使用 GDMA 控制器。GDMA 链表存储在存储器中。这些链表包括 outlink*n* 与 inlink*n*，本文以 *n* 来表示通道号，*n* 为 0 ~ 2。GDMA 从存储器中取得链表，然后根据 outlink*n* 中的内容将相应存储器中的数据发送出去，也可根据 inlink*n* 中的内容将接收的数据存入指定存储器地址空间。

5.4 功能描述

5.4.1 链表

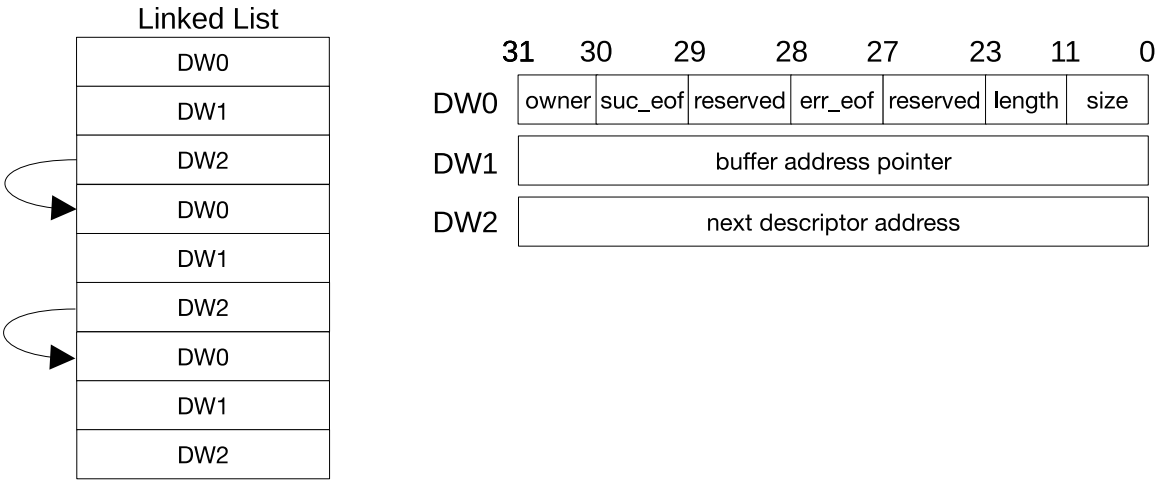


图 5.4-1. 链表结构图

图 5.4-1 所示为链表的结构图。发送链表与接收链表结构相同。每个链表由一个或者若干个描述符构成，一个描述符由 3 个字组成。链表应存放在存储器中，供 GDMA 控制器使用。描述符每一字段的意义如下：

- owner (DW0) [31]：表示当前描述符对应的 buffer 允许的操作者。
  0：允许的操作者为 CPU。
  1：允许的操作者为 GDMA 控制器。
  在 GDMA 使用完该描述符对应的 buffer 后，对于接收描述符，硬件默认会自动将该位清零；对于发送描述符，需要将 AHB_DMA_OUT_AUTO_WRBACK_CH*n* 置 1，硬件才会自动将该位清零。软件也可通过置位 AHB_DMA_OUT_LOOP_TEST_CH*n* 或 AHB_DMA_IN_LOOP_TEST_CH*n* 来关闭硬件自动清零的功能。软件在挂载链表时需要将该位置 1。

  注意：本文以 AHB_DMA_OUT 开头的寄存器对应 TX 通道寄存器，以 AHB_DMA_IN 开头的寄存器对应 RX 通道寄存器。

- suc_eof (DW0) [30]：表示一个描述符对应的数据成功传输后是否触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 或 AHB_DMA_OUT_EOF_CH*n*_INT 中断。
  0：当前描述符成功传输后不会触发中断；
  1：当前描述符成功传输后触发中断。
  对于接收描述符，需要软件将该位写 0，硬件会在接收到包含 EOF 标志的数据后将该位置 1。
  对于发送描述符，需要软件按需要将描述符中的该位置 1。如果软件在某个描述符中将该位配置为 1，则 GDMA 在处理完该描述符时，会在发送给外设的数据中加入 EOF 标志，告知外设该段数据是一个阶段性结束。

- Reserved (DW0) [29]：保留。此位为无关项。

- err_eof (DW0) [28]：表示接收结束错误标志。
  0：接收数据没有错误。
  1：接收数据有错误。
  该位只用于 UHCI 或 PARLIO 利用 GDMA 接收数据。对于接收描述符，硬件在收完描述符对应的数据段并检测到接收数据错误会将该位置 1。

- Reserved (DW0) [27:24]：保留。

- length (DW0) [23:12]：表示当前描述符对应的 buffer 中的有效字节数。对于发送描述符，该段由软件填写，表示从 buffer 中读取数据时需要读取的字节数；对于接收描述符，该段由硬件使用完该 buffer 后或者接收到最后一个数据时自动填写，表示 buffer 中存储的有效字节数。
- size (DW0) [11:0]：表示当前描述符对应的 buffer 容量的字节数。
- buffer address pointer (DW1)：buffer 的地址。
- next descriptor address (DW2)：下一个描述符的地址。如果当前描述符为链表中最后一个描述符时，该值可以为 0。该地址必须指向存储器的地址空间。

> **说明：**
> 需要注意，GDMA 的链表描述符本身存储地址要求 4 字节对齐。

用 GDMA 接收数据时，如果接收数据的长度小于当前描述符指定的 buffer size，那么下一个描述符对应的接收数据不会占用该 buffer 的剩余空间。如果接收数据的长度大于当前描述符指定的 buffer size，那么 GDMA 完成当前描述符后继续获取下一个描述符来完成数据传输。

5.4.2 外设到存储器及存储器到外设的数据传输

GDMA 支持存储器到外设及外设到存储器的数据传输，分别对应 TX 及 RX 功能。TX 通道通过 outlink*n* 实现将指定存储器中的数据搬运到外设的发送端；RX 通道通过 inlink*n* 实现将外设接收到的数据搬运到存储器中。

每个 RX/TX 通道均可以被配置连接到任意一个支持 GDMA 功能的外设，表 5.4-1 所示为通过寄存器配置通道与其对应外设的关系。其中 “Dummy-*n*” 是 GDMA 在实现存储器到存储器数据搬运功能时可选配的寄存器值。当其中一个通道已经与某一个外设连接时，其他通道将不能配置为与该外设连接。

表 5.4-1. GDMA 配置寄存器与外设选择关系表

AHB_DMA_PERI_IN_SEL_CH*n* AHB_DMA_PERI_OUT_SEL_CH*n*	外设
0	Dummy-0
1	GP-SPI
2	UHCI
3	I2S
4 ~ 5	Dummy-4 ~ 5
6	AES
7	SHA
8	ADC
9	PARLIO
10 ~ 15	Dummy-10 ~ 15
16 ~ 63	无效值

5.4.3 存储器到存储器数据传输

GDMA 支持存储器到存储器的数据传输。置位 AHB_DMA_MEM_TRANS_EN_CH*n*，TX 通道 *n* 的输出将与 RX 通道 *n* 的输入相连，从而使能存储器到存储器的数据传输功能。需要注意的是，一个 TX 通道只与其编号对应的 RX 通道相连而实现存储器到存储器的数据传输，并且需要将 AHB_DMA_PERI_IN_SEL_CH*n* 和 AHB_DMA_PERI_OUT_SEL_CH*n* 配置成相同且对应 Dummy 的值。

5.4.4 启动 GDMA

软件通过挂载链表的方式来使用 GDMA。对于接收数据，软件挂载好接收链表并准备好接收数据，配置 AHB_DMA_INLINK_ADDR_CH*n* 字段指向第一个接收链表描述符，置位 AHB_DMA_INLINK_START_CH*n* 位启动 GDMA。对于发送数据，软件挂载好发送链表并准备好发送数据，配置 AHB_DMA_OUTLINK_ADDR_CH*n* 字段指向第一个发送链表描述符，置位 AHB_DMA_OUTLINK_START_CH*n* 位启动 GDMA。AHB_DMA_INLINK_START_CH*n* 与 AHB_DMA_OUTLINK_START_CH*n* 位由硬件自动清零。

有时您可能想要在 DMA 数据传输已经开始后追加更多描述符。要挂载更多描述符，原本看似只需清空已挂载链表最后一个描述符的 EOF 位，并将该描述符的 next descriptor address (DW2) 字段配置为新链表第一个描述符的地址。但如果 DMA 数据传输已经或即将结束，这个方法便无法奏效。GDMA 控制器有专门的逻辑来确保数据传输继续或重启：如果数据传输仍在进行，GDMA 控制器会确保顾及到新追加的描述符；如果数据传输已经结束，GDMA 控制器会重启数据传输，传输新追加的描述符。这个逻辑由 Restart 功能实现。

软件使用 Restart 功能时，需要重写已挂载链表的最后一个描述符，使其第三个字中的内容（即 DW2）指向新链表的首地址；然后置位 AHB_DMA_INLINK_RESTART_CH*n* 或者 AHB_DMA_OUTLINK_RESTART_CH*n*（这两个位由硬件自动清零），如图 5.4-2 所示，硬件会在读取已挂载链表的最后一个描述符时，获取新挂载链表的地址，从而继续处理新挂载的链表。

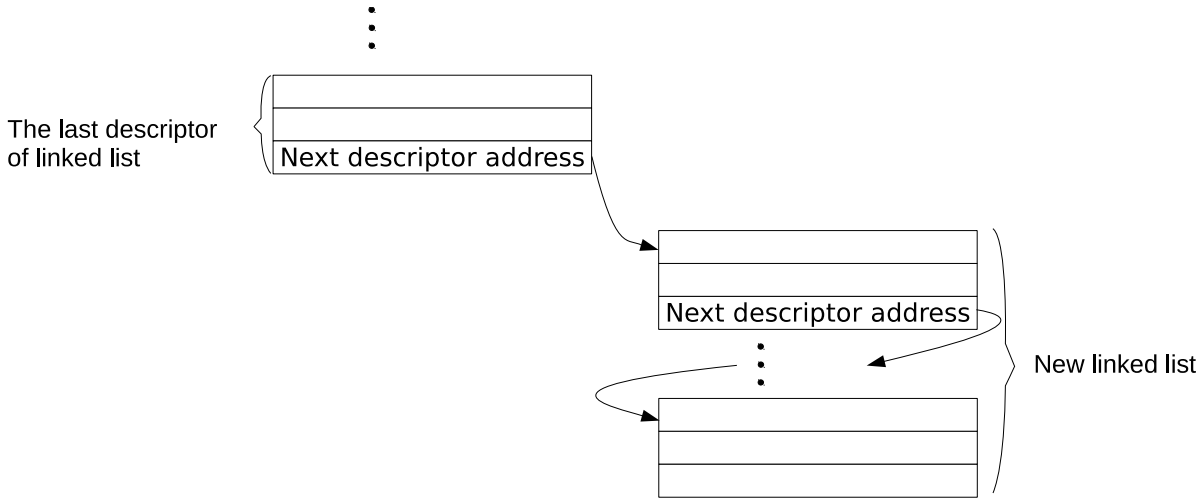


图 5.4-2. 链表关系图

5.4.5 读链表

软件在配置并启动 GDMA 后，GDMA 会从存储器中读取链表。GDMA 会检查读入的链表描述符是否正确。只有当链表描述符通过检查时，GDMA 对应的通道才会开始搬运数据。当链表描述符没有通过检查，硬件将触发描述符错误中断（AHB_DMA_IN_DSCR_ERR_CH*n*_INT 或者 AHB_OUT_DSCR_ERR_CH*n*_INT），同时该通道将会处于阻塞状态，停止工作。

描述符检查项包括：

- AHB_DMA_IN_CHECK_OWNER_CH*n* 或者 AHB_DMA_OUT_CHECK_OWNER_CH*n* 置 1 时，检查描述符的 owner 位。如果该位为 0，表示当前操作者为 CPU，则不通过检查。AHB_DMA_IN_CHECK_OWNER_CH*n* 或者 AHB_DMA_OUT_CHECK_OWNER_CH*n* 置 0 时，不检查描述符的 owner 位。
- 检查描述符本身的存储地址是否在配置的存储器地址空间内。如果 GDMA 的描述符地址在 AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR 值的范围内，则通过检查。请参见章节 5.4.7）。

软件在检查到通道描述符错误中断后，需要准备新描述符，并置位 AHB_DMA_OUTLINK_START_CH*n* 或者 AHB_DMA_INLINK_START_CH*n* 位启动 GDMA。

5.4.6 数据传输结束标志

注意：本章中提到发送链表描述符的 EOF 为描述符中的第一个字中的 bit 30 指示的 suc_eof，接收链表描述符的 EOF 可以为 bit 30 指示的 suc_eof 和 bit 28 指示的 err_eof。

GDMA 通过 EOF 来指示对应描述符所需传输的数据段传输结束。

发送数据时，产生 EOF 和 TOTAL EOF 两种中断：

- AHB_DMA_OUT_EOF_CH*n*_INT：链表中的任意一个描述符有配置 suc_eof 标志位，则当该描述符对应的数据传输完成后，GDMA 都会产生 EOF 中断。置位 AHB_DMA_OUT_EOF_CH*n*_INT_ENA 位使能该中断。
- AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT：如果该链表中的最后一个描述符配置了 suc_eof 标志位，则当该链表的最后一个描述符对应的数据传输完成后，GDMA 会产生 TOTAL EOF 中断。置位 AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA 位使能该中断。

接收数据时，可以产生以下中断：

- AHB_DMA_IN_SUC_EOF_CH*n*_INT：表示带有 EOF 标志的数据段接收完成。置位 AHB_DMA_IN_SUC_EOF_CH*n*_INT_ENA 位使能该中断。
- （仅 UHCI 和 PARLIO）AHB_DMA_IN_ERR_CH*n*_EOF_INT：表示对应描述符所需传输的数据段接收完成但接收数据有错误。置位 AHB_DMA_IN_ERR_EOF_CH*n*_INT_ENA 使能该中断。需要注意的是，只有当通道连接的外设为 UHCI 或 PARLIO 时，才支持该中断。

软件在检测到 AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT 或 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断时，可以通过获取 AHB_DMA_OUT_EOF_DES_ADDR_CH*n* 或 AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n* 字段中的的值，获取当前已经完成数据传输的描述符的存储地址。这样，软件可以知道哪些描述符已经被使用并根据需要回收描述符。RX 方向还支持记录 AHB_DMA_IN_ERR_EOF_CH*n*_INT 中断触发时描述符的存储地址，可通过 AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n* 字段获取。

5.4.7 访问存储器

GDMA 任意 RX/TX 通道均可以访问存储器，其可访问的地址空间为寄存器 AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR 的配置值区间。

为加速数据传输速率，支持突发传输模式。GDMA 默认不使能突发传输模式，用户可以通过配置 AHB_DMA_IN_DATA_BURST_MODE_SEL_CH*n* 和 AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH*n* 为 0、1、2 将 AHB 总线突发传输长度分别配置为 SINGLE、INCR4 或 INCR8。

GDMA 访问配置的存储器地址空间时，链表描述符参数无对齐要求。

5.4.8 仲裁

为了确保及时响应通用 SPI 等高速低延迟外设的请求，GDMA 引入通道优先级仲裁和通道权重仲裁两种仲裁机制。

- 优先级仲裁方式：GDMA 支持 6（0 ~ 5）个等级的优先级，数字越大优先级越高。若多个通道在同一时刻存在数据传输需求，GDMA 会按照通道优先级从高到低顺序响应每个通道的数据传输请求；其数值越大，

对应的优先级越高，请求响应越及时

- 权重仲裁方式：
  - 每个通道的的权重（即令牌数）可以配置为 0 ~ 15 之间的任意值。GDMA 将时间按照一定的总线 (AHB) 时钟周期长度划分成片，在每个时间片内，各个通道的数据传输请求次数应该将按照通道所配置的令牌数进行分配，在任意一个时间片内，某通道每请求一次数据传输消耗一次令牌，若某个通道在某个时间片内消耗尽令牌数，则该通道请求将不再被响应，直到仲裁时间达到时间片规定长度或者所有通道的令牌数均已经消耗尽，此时将刷新令牌数，进入下一个时间片周期内。
  - 若通道令牌数配置为非零值，即使无数据传输请求，也会在每个时间片内等待所有通道的令牌数耗尽，不会提前退出当前仲裁时间片。通道无数据传输请求且令牌值非 0 时，会导致一些带宽浪费，基于此，GDMA 提供额外的权重仲裁优化开关。权重仲裁优化打开时，如果通道无数据传输需求，则该通道令牌数会被认为是 0，不再参与仲裁，当所有通道均无数据传输请求时，仲裁器提前退出当前时间片，以此提高仲裁响应时间，提高传输速率。

说明：

- 若同时配置优先级仲裁和通道权重仲裁，且优先级不同，权重相同，则在各个通道传输相同的数据量，且数据量较少的前提下，优先级高的通道优先传输完成。
- 若优先级不同且权重也不同，各个通道传输数据量相同且较大，则权重较大的通道是能优先分配到大带宽的，会优先完成传输。

5.4.9 总线错误响应信息记录

在 GDMA 搬运数据期间，如果总线上从机侧回复了错误响应 (Error Response)，数据传输不会中断，GDMA 依旧会按照已有的链表搬运数据搬运。这样可能会导致完成数据搬运后，实际得到的数据与预期不符。为了便于定位问题，GDMA 传输数据过程中，如果发生了错误相应，可以通过以下内部寄存器存储相关传输信息：

- AHB_DMA_AHBINF_RESP_ERR_ADDR：记录本次总线传输对应的 32 位地址。
- AHB_DMA_AHBINF_RESP_ERR_CH_ID：记录本次总线传输使用的 GDMA 通道。
- AHB_DMA_AHBINF_RESP_ERR_ID：记录本次总线上外设与存储器传输或存储器与存储器传输的传输 ID。
- AHB_DMA_AHBINF_RESP_ERR_WR：记录本次总线传输对应的传输类型。

GDMA 作为主机发起 AHB 总线传输且收到错误响应时，还会产生错误相应中断来提醒 CPU。中断产生后，GDMA 仍不会中断数据传输。此外，GDMA 不会锁存第一笔发生错误响应的 AHB 总线传输相关寄存器信息。如果连续发生多笔错误响应传输，上述错误响应记录寄存器的内容会实时更新，最终寄存器中保留的信息是查询寄存器之前最后一笔发生错误响应的 AHB 总线传输。

5.4.10 自动时钟门控

GDMA 在工作时，某些场景只需要用到部分通道，此时可关闭未使用通道的时钟来降低功耗。同理，内部仲裁和总线接口模块仅在有数据传输时才需要工作，在空闲时也可以通过关闭时钟来降低功耗。目前 GDMA 内部有以下模块：

- 总线接口处理模块
- 仲裁模块
- RX 数据处理模块

- RX 描述符处理模块
- TX 数据处理模块
- TX 描述符处理模块
- 寄存器配置同步模块

这些模块可以在空闲时自动关闭模块时钟，来更大限度的减小功耗，也支持通过寄存器配置强制保持时钟开启。例如，置位 AHB_DMA_AHBINF_CLK_EN 可使总线接口处理模块时钟保持常开，其余模块时钟常开配置与其类似。

5.5 事件任务矩阵功能

在 ESP32-C5 中，GDMA 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GDMA 的 ETM 任务，或者通过 GDMA 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与 GDMA 相关的 ETM 任务和 ETM 事件。

GDMA 可接收的 ETM 任务有：

- GDMA_TASK_IN_START_CH*n*：触发时开启 GDMA 的接收通道 *n* 进行数据传输。
- GDMA_TASK_OUT_START_CH*n*：触发时开启 GDMA 的发送通道 *n* 进行数据传输。

说明：
以上两个 ETM 任务与 CPU 配置 AHB_DMA_INLINK_START_CH*n*、AHB_DMA_OUTLINK_START_CH*n* 实现的功能相同。当 AHB_DMA_IN_ETM_EN_CH*n* 或 AHB_DMA_OUT_ETM_EN_CH*n* 为 1 时，只有 ETM 可以配置对应方向及对应通道 GDMA 的启动，当 AHB_DMA_IN_ETM_EN_CH*n* 或 AHB_DMA_OUT_ETM_EN_CH*n* 为 0 时，仅可通过 CPU 使能对应的 GDMA。

GDMA 可产生的 ETM 事件有：

- GDMA_EVT_IN_DONE_CH*n*：表示通道 *n* 接收方向链表描述符指向的数据接收完成。
- GDMA_EVT_IN_SUC_EOF_CH*n*：通道 *n* 接收方向 EOF 位为 1 的链表描述符指向的数据接收完成指示。
- GDMA_EVT_IN_FIFO_EMPTY_CH*n*：通道 *n* 接收方向 FIFO 非空到空状态改变指示。
- GDMA_EVT_IN_FIFO_FULL_CH*n*：通道 *n* 接收方向 FIFO 非满到满状态改变指示。
- GDMA_EVT_OUT_DONE_CH*n*：通道 *n* 表示通道 *n* 发送方向链表描述符指向的数据发送完成。
- GDMA_EVT_OUT_EOF_CH*n*：通道 *n* 发送方向 EOF 位为 1 的链表描述符指向的数据发送完成指示。
- GDMA_EVT_OUT_TOTAL_EOF_CH*n*：通道 *n* 发送方向最后一个并且 EOF 位为 1 的链表描述指向的数据发送符完成指示。
- GDMA_EVT_OUT_FIFO_EMPTY_CH*n*：通道 *n* 发送方向 FIFO 非空到空状态改变指示。
- GDMA_EVT_OUT_FIFO_FULL_CH*n*：通道 *n* 发送方向 FIFO 非满到满状态改变指示。

在具体应用中，GDMA 的 ETM 事件可以用来触发 GDMA 的 ETM 任务，例如，GDMA_EVT_OUT_TOTAL_EOF_CH0 事件可以触发 GDMA_TASK_IN_START_CH1 任务，从而触发新一轮的 GDMA 操作。

5.6 中断

ESP32-C5 的 GDMA 可以生成以下中断信号并发送给中断矩阵。

- GDMA_IN_CH0_INTR
- GDMA_IN_CH1_INTR
- GDMA_IN_CH2_INTR
- GDMA_OUT_CH0_INTR
- GDMA_OUT_CH1_INTR
- GDMA_OUT_CH2_INTR

这些中断信号由 GDMA 的内部中断源生成。

以下中断源生成 GDMA_IN_CH*n*_INTR (*n* 为 0 ~ 2) 中断信号：

- AHB_DMA_IN_DONE_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成时触发此中断。
- AHB_DMA_IN_SUC_EOF_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成，并且描述符的 suc_eof 为 1 时触发此中断。
- AHB_DMA_IN_ERR_EOF_CH*n*_INT：对于接收通道*n*，当接收的描述符对应数据段中有错误发生时触发此中断。(该中断只用于外设选择 UHCI (UART0/UART1) 或 PARLIO 时)。
- AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT：对于接收通道*n*，当接收链表描述符指向的 buffer size 小于待接收数据长度时触发此中断。
- AHB_DMA_IN_DSCR_ERR_CH*n*_INT：对于接收通道*n*，当接收链表描述符未通过任一描述符检查时触发此中断。
- AHB_DMA_INFIFO_OVF_CH*n*_INT：当 GDMA 接收数据方向缓存溢出时触发此中断。
- AHB_DMA_INFIFO_UDF_CH*n*_INT：当 GDMA 接收数据方向缓存无数据时触发此中断。
- AHB_DMA_IN_AHBINF_RESP_ERR_CH*n*_INT：GDMA 接收通道*n* 进行数据传输时，AHB 总线上如果有错误相应，触发此中断。

以下中断源生成 GDMA_OUT_CH*n*_INTR (*n* 为 0 ~ 2) 中断信号：

- AHB_DMA_OUT_DONE_CH*n*_INT：对于发送通道*n*，当一个发送链表描述符对应的数据发送完成时触发此中断。
- AHB_DMA_OUT_EOF_CH*n*_INT：对于发送通道*n*，当发送描述符的 EOF 位为 1，且 AHB_DMA_OUT_EOF_MODE_CH*n* 为 0 时，该描述符对应的最后一个数据由总线传入 GDMA TX 通道时，该中断触发；当发送描述符的 EOF 位为 1，且 AHB_DMA_OUT_EOF_MODE_CH*n* 为 1 时，该描述符对应的最后一个数据从 GDMA TX 通道发出时，该中断触发。
- AHB_DMA_OUT_DSCR_ERR_CH*n*_INT：对于发送通道*n*，当发送链表描述符未通过任一描述符检查时会触发此中断。
- AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT：对于发送通道*n*，当一个链表（可包含多个链表描述符）对应的所有数据都已发送完成时触发此中断。
- AHB_DMA_OUTFIFO_OVF_CH*n*_INT：当 GDMA 发送数据方向缓存溢出时触发此中断。

- AHB_DMA_OUTFIFO_UDF_CH*n*_INT：当 GDMA 发送数据方向缓存无数据时触发此中断。
- AHB_DMA_OUT_AHBINF_RESP_ERR_CH*n*_INT：GDMA 发送通道*n* 进行数据传输时，AHB 总线上如果有错误相应，触发此中断。

5.7 编程流程

可通过配置 PCR_GDMA_CLK_EN 寄存器管理 GDMA 时钟门控，默认时钟门控打开。可通过配置 PCR_GDMA_RST_EN 对 GDMA 模块全局复位。

5.7.1 GDMA TX 通道配置流程

利用 GDMA 发送数据时，GDMA TX 通道的软件配置流程如下：

1. 对寄存器 AHB_DMA_OUT_RST_CH*n* 置 1 然后置 0，复位 GDMA TX 通道状态机和 FIFO 指针；
2. 挂载好发送链表，配置寄存器 AHB_DMA_OUTLINK_ADDR_CH*n* 指向第一个发送链表描述符；
3. 配置 AHB_DMA_PERI_OUT_SEL_CH*n* 为对应的外设号，见表 5.4-1；
4. 置位 AHB_DMA_OUTLINK_START_CH*n* 启动 GDMA TX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；
6. 等待 AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT 中断，即数据传输完成。

5.7.2 GDMA RX 通道配置流程

利用 GDMA 接收数据时，GDMA RX 通道的软件配置流程如下：

1. 对寄存器 AHB_DMA_IN_RST_CH*n* 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针；
2. 挂载好接收链表，配置寄存器 AHB_DMA_INLINK_ADDR_CH*n* 指向第一个接收链表描述符；
3. 配置 AHB_DMA_PERI_IN_SEL_CH*n* 为对应的外设号，见表 5.4-1；
4. 置位 AHB_DMA_INLINK_START_CH*n* 启动 GDMA RX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；

5.7.3 GDMA 存储器到存储器配置流程

利用 GDMA 从存储器到存储器搬运数据时配置流程如下：

1. 对寄存器 AHB_DMA_OUT_RST_CH*n* 置 1 然后置 0，复位 GDMA TX 通道状态机和 FIFO 指针；
2. 对寄存器 AHB_DMA_IN_RST_CH*n* 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针；
3. 挂载好发送链表，配置寄存器 AHB_DMA_OUTLINK_ADDR_CH*n* 指向第一个发送链表描述符；
4. 挂载好接收链表，配置寄存器 AHB_DMA_INLINK_ADDR_CH*n* 指向第一个接收链表描述符；
5. 置位 AHB_DMA_MEM_TRANS_EN_CH*n* 使能存储器到存储器传输功能；
6. 置位 AHB_DMA_OUTLINK_START_CH*n* 启动 GDMA TX 通道发送数据；
7. 置位 AHB_DMA_INLINK_START_CH*n* 启动 GDMA RX 通道发送数据；

8. 如果某个发送链表描述符的 suc_eof 位配置为 1，则该描述符对应的数据段传输结束后会触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。

5.7.4 通道优先级和权重配置流程

优先级仲裁配置方法如下：

1. 配置 AHB_DMA_TX_PRI_CH*n* 和 AHB_DMA_RX_PRI_CH*n* 为 TX 和 RX 通道的优先级。

权重仲裁配置方法如下：

1. 配置 AHB_DMA_ARB_TIMEOUT 为权重仲裁的时间片长度；
2. 配置 AHB_DMA_TX_CH_ARB_WEIGH_CH*n* 和 AHB_DMA_RX_CH_ARB_WEIGH_CH*n* 为各个通道的令牌值；
3. 清除 AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*，使能 TX 和 RX 方向的权重仲裁优化；
4. 置位 AHB_DMA_WEIGHT_EN，使能权重仲裁。

5.8 寄存器列表

本小节的所有地址均为相对于 GDMA 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
AHB_DMA_IN_INT_RAW_CH0_REG	RX 通道 0 原始中断状态寄存器	0x0000	R/ WTC/ SS
AHB_DMA_IN_INT_ST_CH0_REG	RX 通道 0 屏蔽中断状态寄存器	0x0004	RO
AHB_DMA_IN_INT_ENA_CH0_REG	RX 通道 0 中断使能寄存器	0x0008	R/W
AHB_DMA_IN_INT_CLR_CH0_REG	RX 通道 0 中断清除寄存器	0x000C	WT
AHB_DMA_IN_INT_RAW_CH1_REG	RX 通道 1 原始中断状态寄存器	0x0010	R/ WTC/ SS
AHB_DMA_IN_INT_ST_CH1_REG	RX 通道 1 屏蔽中断状态寄存器	0x0014	RO
AHB_DMA_IN_INT_ENA_CH1_REG	RX 通道 1 中断使能寄存器	0x0018	R/W
AHB_DMA_IN_INT_CLR_CH1_REG	RX 通道 1 中断清除寄存器	0x001C	WT
AHB_DMA_IN_INT_RAW_CH2_REG	RX 通道 2 原始中断状态寄存器	0x0020	R/ WTC/ SS
AHB_DMA_IN_INT_ST_CH2_REG	RX 通道 2 屏蔽中断状态寄存器	0x0024	RO
AHB_DMA_IN_INT_ENA_CH2_REG	RX 通道 2 中断使能寄存器	0x0028	R/W
AHB_DMA_IN_INT_CLR_CH2_REG	RX 通道 2 中断清除寄存器	0x002C	WT
AHB_DMA_OUT_INT_RAW_CH0_REG	TX 通道 0 原始中断状态寄存器	0x0030	R/ WTC/ SS
AHB_DMA_OUT_INT_ST_CH0_REG	TX 通道 0 屏蔽中断状态寄存器	0x0034	RO
AHB_DMA_OUT_INT_ENA_CH0_REG	TX 通道 0 中断使能寄存器	0x0038	R/W
AHB_DMA_OUT_INT_CLR_CH0_REG	TX 通道 0 中断清除寄存器	0x003C	WT
AHB_DMA_OUT_INT_RAW_CH1_REG	TX 通道 1 原始中断状态寄存器	0x0040	R/ WTC/ SS
AHB_DMA_OUT_INT_ST_CH1_REG	TX 通道 1 屏蔽中断状态寄存器	0x0044	RO

GoBack

GoBack

名称	描述	地址	访问
AHB_DMA_OUT_INT_ENA_CH1_REG	TX 通道 1 中断使能寄存器	0x0048	R/W
AHB_DMA_OUT_INT_CLR_CH1_REG	TX 通道 1 中断清除寄存器	0x004C	WT
AHB_DMA_OUT_INT_RAW_CH2_REG	TX 通道 2 原始中断状态寄存器	0x0050	R/ WTC/ SS
AHB_DMA_OUT_INT_ST_CH2_REG	TX 通道 2 屏蔽中断状态寄存器	0x0054	RO
AHB_DMA_OUT_INT_ENA_CH2_REG	TX 通道 2 中断使能寄存器	0x0058	R/W
AHB_DMA_OUT_INT_CLR_CH2_REG	TX 通道 2 中断清除寄存器	0x005C	WT
调试寄存器			
AHB_DMA_AHB_TEST_REG	保留	0x0060	R/W
配置寄存器			
AHB_DMA_MISC_CONF_REG	杂项寄存器	0x0064	R/W
AHB_DMA_IN_CONF0_CH0_REG	RX 通道 0 的配置寄存器 0	0x0070	R/W
AHB_DMA_IN_CONF1_CH0_REG	RX 通道 0 的配置寄存器 1	0x0074	R/W
AHB_DMA_IN_POP_CH0_REG	RX 通道 0 的弹出控制寄存器	0x007C	varies
AHB_DMA_IN_LINK_CH0_REG	RX 通道 0 的链表描述符配置和控制寄存器	0x0080	varies
AHB_DMA_OUT_CONF0_CH0_REG	TX 通道 0 的配置寄存器 0	0x00D0	R/W
AHB_DMA_OUT_CONF1_CH0_REG	TX 通道 0 的配置寄存器 1	0x00D4	R/W
AHB_DMA_OUT_PUSH_CH0_REG	TX 通道 0 的推入控制寄存器	0x00DC	varies
AHB_DMA_OUT_LINK_CH0_REG	TX 通道 0 的链表描述符配置和控制寄存器	0x00E0	varies
AHB_DMA_IN_CONF0_CH1_REG	RX 通道 1 的配置寄存器 0	0x0130	R/W
AHB_DMA_IN_CONF1_CH1_REG	RX 通道 1 的配置寄存器 1	0x0134	R/W
AHB_DMA_IN_POP_CH1_REG	RX 通道 1 的弹出控制寄存器	0x013C	varies
AHB_DMA_IN_LINK_CH1_REG	RX 通道 1 的链表描述符配置和控制寄存器	0x0140	varies
AHB_DMA_OUT_CONF0_CH1_REG	TX 通道 1 的配置寄存器 0	0x0190	R/W
AHB_DMA_OUT_CONF1_CH1_REG	TX 通道 1 的配置寄存器 1	0x0194	R/W
AHB_DMA_OUT_PUSH_CH1_REG	TX 通道 1 的推入控制寄存器	0x019C	varies
AHB_DMA_OUT_LINK_CH1_REG	TX 通道 1 的链表描述符配置和控制寄存器	0x01A0	varies
AHB_DMA_IN_CONF0_CH2_REG	RX 通道 2 的配置寄存器 0	0x01F0	R/W
AHB_DMA_IN_CONF1_CH2_REG	RX 通道 2 的配置寄存器 1	0x01F4	R/W

GoBack

名称	描述	地址	访问
AHB_DMA_IN_POP_CH2_REG	RX 通道 2 的弹出控制寄存器	0x01FC	varies
AHB_DMA_IN_LINK_CH2_REG	RX 通道 2 的链表描述符配置和控制寄存器	0x0200	varies
AHB_DMA_OUT_CONF0_CH2_REG	TX 通道 2 的配置寄存器 0	0x0250	R/W
AHB_DMA_OUT_CONF1_CH2_REG	TX 通道 2 的配置寄存器 1	0x0254	R/W
AHB_DMA_OUT_PUSH_CH2_REG	TX 通道 2 的推入控制寄存器	0x025C	varies
AHB_DMA_OUT_LINK_CH2_REG	TX 通道 2 的链表描述符配置和控制寄存器	0x0260	varies
AHB_DMA_TX_CH_ARB_WEIGH_CH0_REG	TX 通道 0 仲裁权重配置寄存器	0x02DC	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH0_REG	TX 通道 0 权重仲裁优化使能寄存器	0x02E0	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH1_REG	TX 通道 1 仲裁权重配置寄存器	0x0304	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH1_REG	TX 通道 1 权重仲裁优化使能寄存器	0x0308	R/W
AHB_DMA_TX_CH_ARB_WEIGH_CH2_REG	TX 通道 2 仲裁权重配置寄存器	0x032C	R/W
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH2_REG	TX 通道 2 权重仲裁优化使能寄存器	0x0330	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH0_REG	RX 通道 0 仲裁权重配置寄存器	0x0354	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH0_REG	RX 通道 0 权重仲裁优化使能寄存器	0x0358	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH1_REG	RX 通道 1 仲裁权重配置寄存器	0x037C	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH1_REG	RX 通道 1 权重仲裁优化使能寄存器	0x0380	R/W
AHB_DMA_RX_CH_ARB_WEIGH_CH2_REG	RX 通道 2 仲裁权重配置寄存器	0x03A4	R/W
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH2_REG	RX 通道 2 权重仲裁优化使能寄存器	0x03A8	R/W
AHB_DMA_IN_LINK_ADDR_CH0_REG	RX 通道 0 链表描述符配置寄存器	0x03AC	R/W
AHB_DMA_IN_LINK_ADDR_CH1_REG	RX 通道 1 链表描述符配置寄存器	0x03B0	R/W
AHB_DMA_IN_LINK_ADDR_CH2_REG	RX 通道 2 链表描述符配置寄存器	0x03B4	R/W
AHB_DMA_OUT_LINK_ADDR_CH0_REG	TX 通道 0 链表描述符配置寄存器	0x03B8	R/W
AHB_DMA_OUT_LINK_ADDR_CH1_REG	TX 通道 1 链表描述符配置寄存器	0x03BC	R/W
AHB_DMA_OUT_LINK_ADDR_CH2_REG	TX 通道 2 链表描述符配置寄存器	0x03C0	R/W
AHB_DMA_INTR_MEM_START_ADDR_REG	可访问地址空间起始地址配置寄存器	0x03C4	R/W
AHB_DMA_INTR_MEM_END_ADDR_REG	可访问地址空间结束地址配置寄存器	0x03C8	R/W
AHB_DMA_ARB_TIMEOUT_REG	权重仲裁超时配置寄存器	0x03DC	R/W
AHB_DMA_WEIGHT_EN_REG	权重仲裁使能寄存器	0x0400	R/W
AHB_DMA_MODULE_CLK_EN_REG	GDMA 内部各模块强制时钟使能寄存器	0x0404	R/W

名称	描述	地址	访问
版本寄存器			
AHB_DMA_DATE_REG	版本控制寄存器	0x0068	R/W
状态寄存器			
AHB_DMA_INFIFO_STATUS_CH0_REG	RX 通道 0 FIFO 状态	0x0078	RO
AHB_DMA_IN_STATE_CH0_REG	RX 通道 0 状态	0x0084	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH0_REG	RX 通道 0 传输结束 (EOF) 时的接收链表描述符地址	0x0088	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH0_REG	RX 通道 0 上发生错误时的接收链表描述符地址	0x008C	RO
AHB_DMA_IN_DSCR_CH0_REG	RX 通道 0 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0090	RO
AHB_DMA_IN_DSCR_BF0_CH0_REG	RX 通道 0 当前已预读取的接收链表描述符地址	0x0094	RO
AHB_DMA_IN_DSCR_BF1_CH0_REG	RX 通道 0 上一个已预读取的接收链表描述符地址	0x0098	RO
AHB_DMA_OUTFIFO_STATUS_CH0_REG	TX 通道 0 FIFO 状态	0x00D8	RO
AHB_DMA_OUT_STATE_CH0_REG	TX 通道 0 状态	0x00E4	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH0_REG	TX 通道 0 传输结束 (EOF) 时的发送链表描述符地址	0x00E8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	TX 通道 0 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x00EC	RO
AHB_DMA_OUT_DSCR_CH0_REG	TX 通道 0 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x00F0	RO
AHB_DMA_OUT_DSCR_BF0_CH0_REG	TX 通道 0 当前已预读取的发送链表描述符地址	0x00F4	RO
AHB_DMA_OUT_DSCR_BF1_CH0_REG	TX 通道 0 上一个已预读取的发送链表描述符地址	0x00F8	RO
AHB_DMA_INFIFO_STATUS_CH1_REG	RX 通道 1 FIFO 状态	0x0138	RO
AHB_DMA_IN_STATE_CH1_REG	RX 通道 1 状态	0x0144	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX 通道 1 传输结束 (EOF) 时的接收链表描述符地址	0x0148	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX 通道 1 上发生错误时的接收链表描述符地址	0x014C	RO
AHB_DMA_IN_DSCR_CH1_REG	RX 通道 1 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0150	RO
AHB_DMA_IN_DSCR_BF0_CH1_REG	RX 通道 1 当前已预读取的接收链表描述符地址	0x0154	RO
AHB_DMA_IN_DSCR_BF1_CH1_REG	RX 通道 1 上一个已预读取的接收链表描述符地址	0x0158	RO
AHB_DMA_OUTFIFO_STATUS_CH1_REG	TX 通道 1 FIFO 状态	0x0198	RO

GoBack

名称	描述	地址	访问
AHB_DMA_OUT_STATE_CH1_REG	TX 通道 1 状态	0x01A4	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的发送链表描述符地址	0x01A8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x01AC	RO
AHB_DMA_OUT_DSCR_CH1_REG	TX 通道 1 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x01B0	RO
AHB_DMA_OUT_DSCR_BF0_CH1_REG	TX 通道 1 当前已预读取的发送链表描述符地址	0x01B4	RO
AHB_DMA_OUT_DSCR_BF1_CH1_REG	TX 通道 1 上一个已预读取的发送链表描述符地址	0x01B8	RO
AHB_DMA_INFIFO_STATUS_CH2_REG	RX 通道 2 FIFO 状态	0x01F8	RO
AHB_DMA_IN_STATE_CH2_REG	RX 通道 2 状态	0x0204	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX 通道 2 传输结束 (EOF) 时接收链表描述符地址	0x0208	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	当 RX 通道 2 上发生错误时接收链表描述符地址	0x020C	RO
AHB_DMA_IN_DSCR_CH2_REG	当前预读取接收链表描述符指向的下一个接收链表描述符的地址在 RX 通道 2 上	0x0210	RO
AHB_DMA_IN_DSCR_BF0_CH2_REG	当前预读取接收链表描述符的地址在 RX 通道 2 上	0x0214	RO
AHB_DMA_IN_DSCR_BF1_CH2_REG	上一个已预读取的取接收链表描述符的地址在 RX 通道 2 上	0x0218	RO
AHB_DMA_OUTFIFO_STATUS_CH2_REG	TX 通道 2 FIFO 状态	0x0258	RO
AHB_DMA_OUT_STATE_CH2_REG	TX 通道 2 的状态	0x0264	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH2_REG	当 TX 通道 2 传输结束 (EOF) 时发送链表描述符地址	0x0268	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	TX 通道 2 传输结束 (EOF) 时最后一个发送链表描述符地址	0x026C	RO
AHB_DMA_OUT_DSCR_CH2_REG	TX 通道 2 当前预读取发送链表描述符指向的下一个发送链表描述符的地址	0x0270	RO
AHB_DMA_OUT_DSCR_BF0_CH2_REG	TX 通道 2 当前预读取发送链表描述符的地址	0x0274	RO
AHB_DMA_OUT_DSCR_BF1_CH2_REG	TX 通道 2 上一个已预读取的取发送链表描述符的地址在	0x0278	RO
AHB_DMA_AHBINF_RESP_ERR_STATUSO_REG	GDMA 收到 AHB 总线错误响应对应传输的地址	0x0408	RO

GoBack

名称	描述	地址	访问
AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG	GDMA 收到 AHB 总线错误响应对应传输的传输类型、ID 和通道 ID	0x040C	RO
AHB_DMA_IN_DONE_DES_ADDR_CH0_REG	记录 RX 通道 0 当前已完成描述符的地址	0x0410	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH0_REG	记录 TX 通道 0 当前已完成描述符的地址	0x0414	RO
AHB_DMA_IN_DONE_DES_ADDR_CH1_REG	记录 RX 通道 1 当前已完成描述符的地址	0x0418	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH1_REG	记录 TX 通道 1 当前已完成描述符的地址	0x041C	RO
AHB_DMA_IN_DONE_DES_ADDR_CH2_REG	记录 RX 通道 2 当前已完成描述符的地址	0x0420	RO
AHB_DMA_OUT_DONE_DES_ADDR_CH2_REG	记录 TX 通道 2 当前已完成描述符的地址	0x0424	RO
优先级寄存器			
AHB_DMA_IN_PRI_CH0_REG	RX 通道 0 的优先级寄存器	0x009C	R/W
AHB_DMA_OUT_PRI_CH0_REG	TX 通道 0 的优先级寄存器	0x00FC	R/W
AHB_DMA_IN_PRI_CH1_REG	RX 通道 1 的优先级寄存器	0x015C	R/W
AHB_DMA_OUT_PRI_CH1_REG	TX 通道 1 的优先级寄存器	0x01BC	R/W
AHB_DMA_IN_PRI_CH2_REG	RX 通道 2 的优先级寄存器	0x021C	R/W
AHB_DMA_OUT_PRI_CH2_REG	TX 通道 2 的优先级寄存器	0x027C	R/W
外设选择寄存器			
AHB_DMA_IN_PERI_SEL_CH0_REG	RX 通道 0 的外设选择寄存器	0x00A0	R/W
AHB_DMA_OUT_PERI_SEL_CH0_REG	TX 通道 0 的外设选择寄存器	0x0100	R/W
AHB_DMA_IN_PERI_SEL_CH1_REG	RX 通道 1 的外设选择寄存器	0x0160	R/W
AHB_DMA_OUT_PERI_SEL_CH1_REG	TX 通道 1 的外设选择寄存器	0x01C0	R/W
AHB_DMA_IN_PERI_SEL_CH2_REG	RX 通道 2 的外设选择寄存器	0x0220	R/W
AHB_DMA_OUT_PERI_SEL_CH2_REG	TX 通道 2 的外设选择寄存器	0x0280	R/W

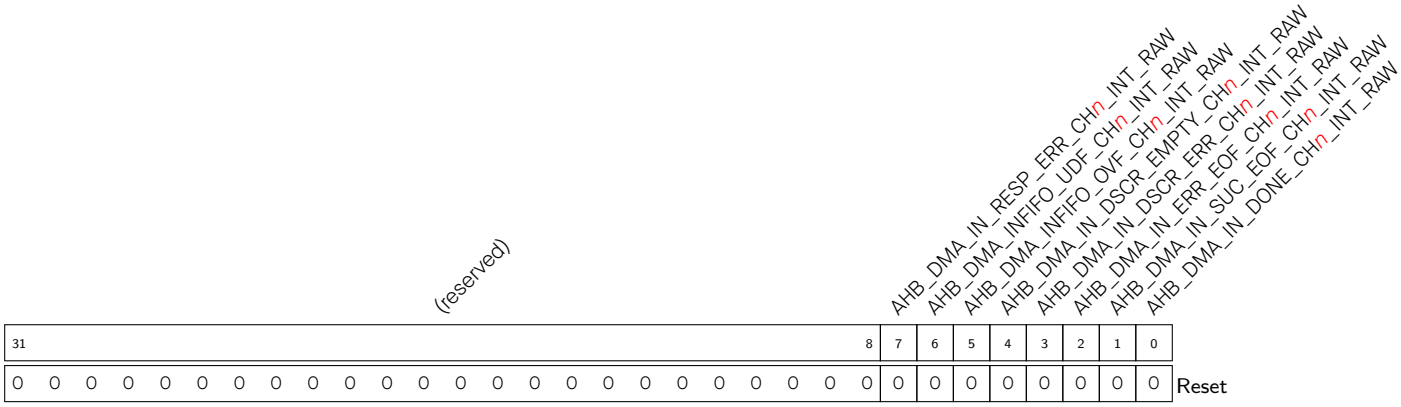
GoBack

5.9 寄存器

本小节的所有地址均为相对于 GDMA 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 5.1. AHB_DMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x10n*)**



AHB_DMA_IN_DONE_CH*n*_INT_RAW AHB_DMA_IN_DONE_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_RAW AHB_DMA_IN_SUC_EOF_CH*n*_INT 的原始中断状态。对于 UHCI，接收链表描述符指向的最后一个字节数据接收成功、且接收通道 *n* 没有检测到数据错误时，该位变为 1。(R/WTC/SS)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_RAW AHB_DMA_IN_ERR_EOF_CH*n*_INT 的原始中断状态。仅对 UHCI 有效。(R/WTC/SS)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_RAW AHB_DMA_IN_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

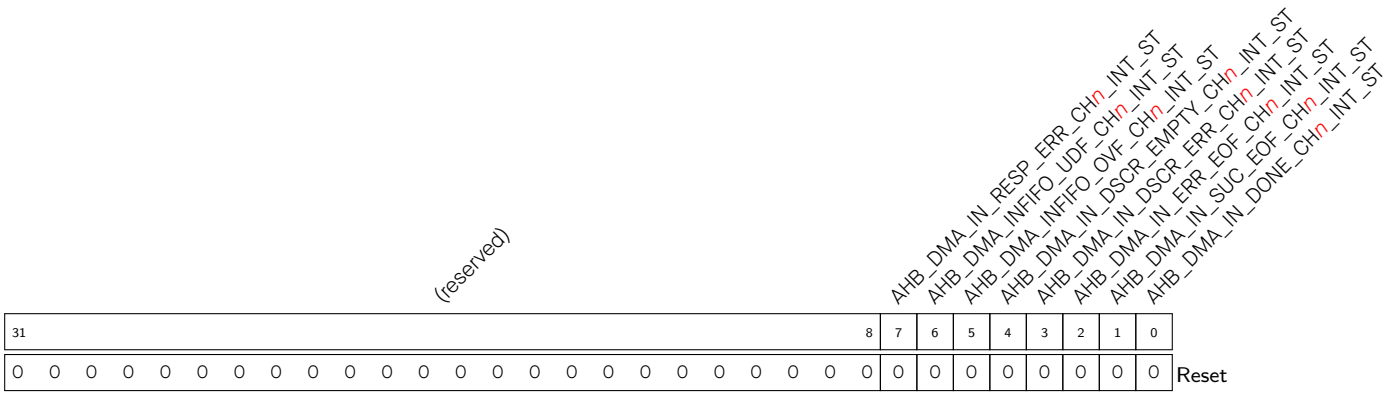
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_RAW AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_INFIFO_OVF_CH*n*_INT_RAW AHB_DMA_INFIFO_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_INFIFO_UDF_CH*n*_INT_RAW AHB_DMA_INFIFO_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_RAW AHB_DMA_IN_RESP_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 5.2. AHB_DMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x10***n*)



AHB_DMA_IN_DONE_CH*n*_INT_ST AHB_DMA_IN_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_ST AHB_DMA_IN_SUC_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_ST AHB_DMA_IN_ERR_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ST AHB_DMA_IN_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

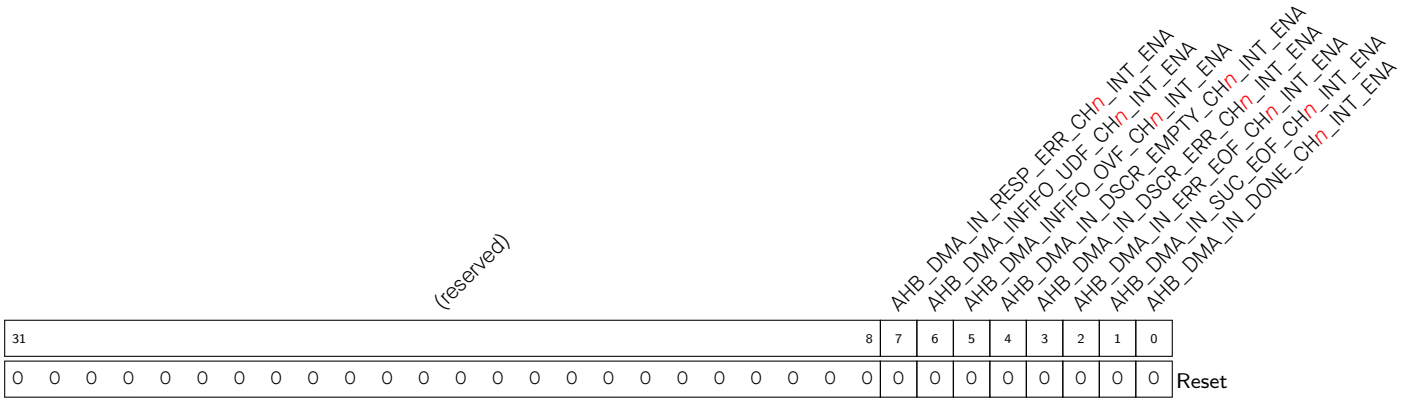
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ST AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_INFIFO_OVF_CH*n*_INT_ST AHB_DMA_INFIFO_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_INFIFO_UDF_CH*n*_INT_ST AHB_DMA_INFIFO_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_ST AHB_DMA_IN_RESP_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

Register 5.3. AHB_DMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x10**n*)



AHB_DMA_IN_DONE_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_DONE_CH*n*_INT。(R/W)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_SUC_EOF_CH*n*_INT。(R/W)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_ERR_EOF_CH*n*_INT。(R/W)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_DSCR_ERR_CH*n*_INT。(R/W)

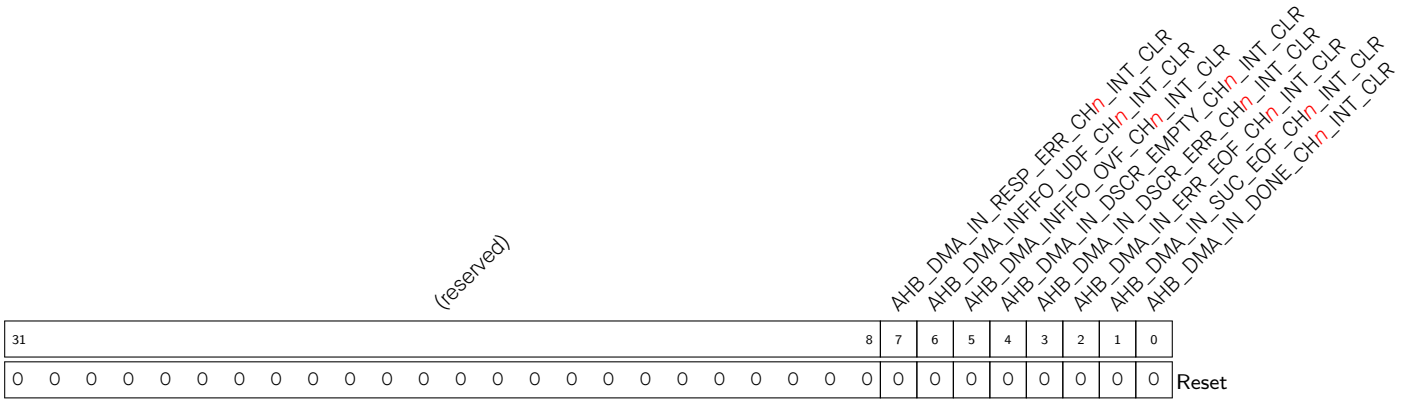
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT。(R/W)

AHB_DMA_INFIFO_OVF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_INFIFO_OVF_CH*n*_INT。(R/W)

AHB_DMA_INFIFO_UDF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_INFIFO_UDF_CH*n*_INT。(R/W)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_IN_RESP_ERR_CH*n*_INT。(R/W)

Register 5.4. AHB_DMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x10**n*)



AHB_DMA_IN_DONE_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_DONE_CH*n*_INT。(WT)

AHB_DMA_IN_SUC_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_SUC_EOF_CH*n*_INT。(WT)

AHB_DMA_IN_ERR_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_ERR_EOF_CH*n*_INT。(WT)

AHB_DMA_IN_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_DSCR_ERR_CH*n*_INT。(WT)

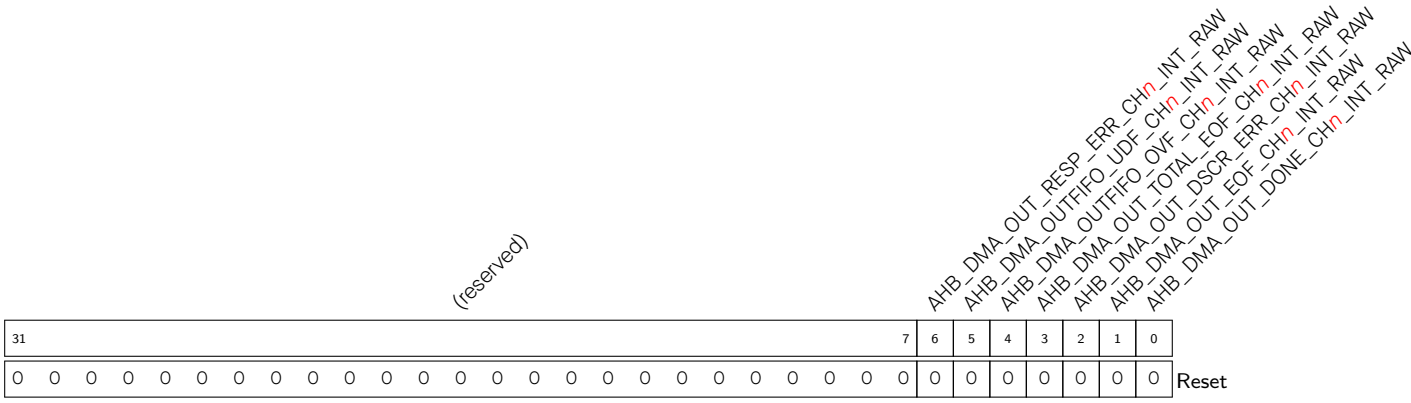
AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_DSCR_EMPTY_CH*n*_INT。(WT)

AHB_DMA_INFIFO_OVF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_INFIFO_OVF_CH*n*_INT。(WT)

AHB_DMA_INFIFO_UDF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_INFIFO_UDF_CH*n*_INT。(WT)

AHB_DMA_IN_RESP_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_IN_RESP_ERR_CH*n*_INT。(WT)

Register 5.5. AHB_DMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0030+0x10***n*)



AHB_DMA_OUT_DONE_CH*n*_INT_RAW AHB_DMA_OUT_DONE_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_EOF_CH*n*_INT_RAW AHB_DMA_OUT_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_RAW AHB_DMA_OUT_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

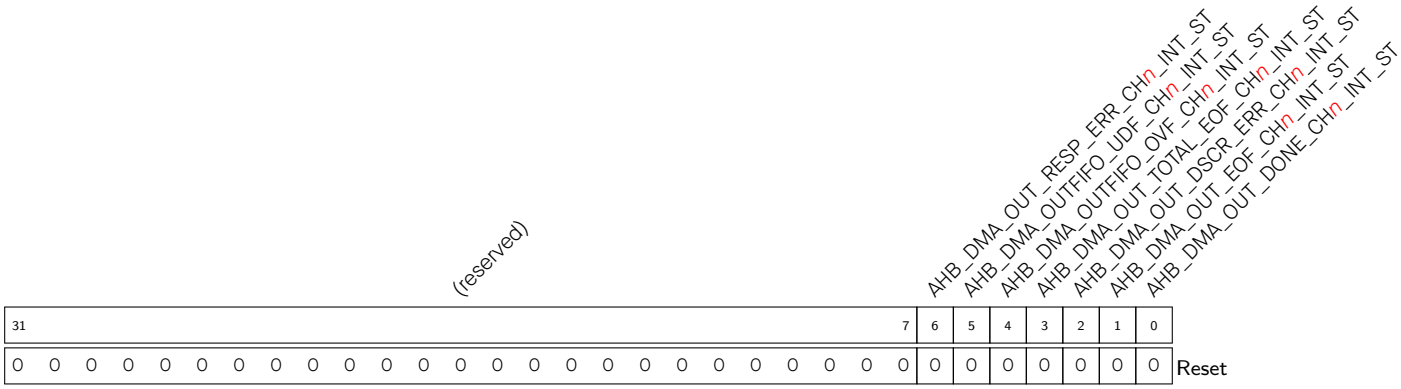
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_RAW AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_RAW AHB_DMA_OUTFIFO_OVF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_RAW AHB_DMA_OUTFIFO_UDF_CH*n*_INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_RAW AHB_DMA_OUT_RESP_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

Register 5.6. AHB_DMA_OUT_INT_ST_CH*n*_REG (*n*: 0-2) (0x0034+0x10**n*)



AHB_DMA_OUT_DONE_CH*n*_INT_ST AHB_DMA_OUT_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_EOF_CH*n*_INT_ST AHB_DMA_OUT_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ST AHB_DMA_OUT_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

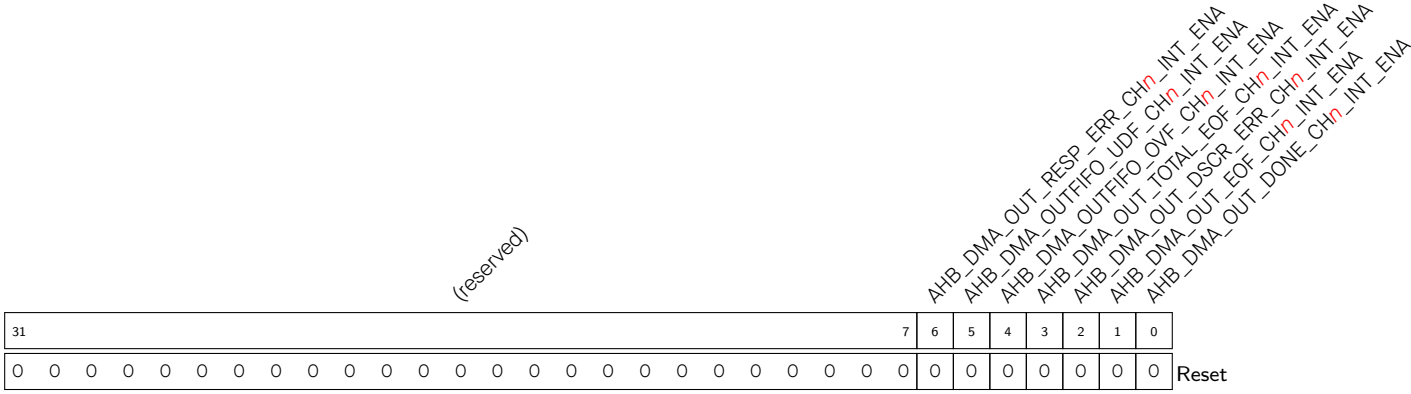
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ST AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_ST AHB_DMA_OUTFIFO_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_ST AHB_DMA_OUTFIFO_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_ST AHB_DMA_OUT_RESP_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

Register 5.7. AHB_DMA_OUT_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0038+0x10**n*)



AHB_DMA_OUT_DONE_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_DONE_CH*n*_INT。(R/W)

AHB_DMA_OUT_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_EOF_CH*n*_INT。(R/W)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_DSCR_ERR_CH*n*_INT。(R/W)

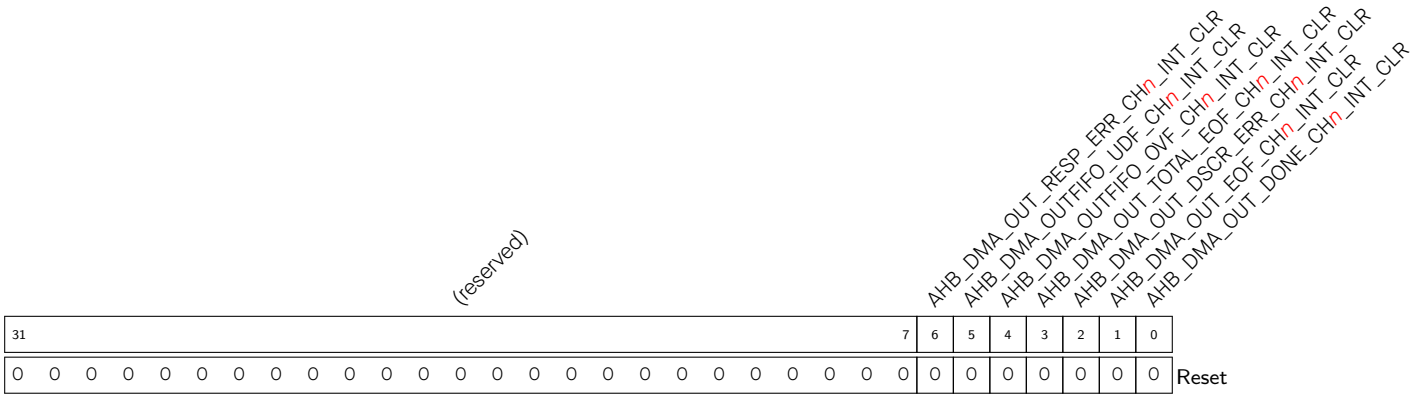
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT。(R/W)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUTFIFO_OVF_CH*n*_INT。(R/W)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUTFIFO_UDF_CH*n*_INT。(R/W)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_ENA 写 1 使能 AHB_DMA_OUT_RESP_ERR_CH*n*_INT。(R/W)

Register 5.8. AHB_DMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x003C+0x10**n*)



AHB_DMA_OUT_DONE_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_DONE_CH*n*_INT。(WT)

AHB_DMA_OUT_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_EOF_CH*n*_INT。(WT)

AHB_DMA_OUT_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_DSCR_ERR_CH*n*_INT。(WT)

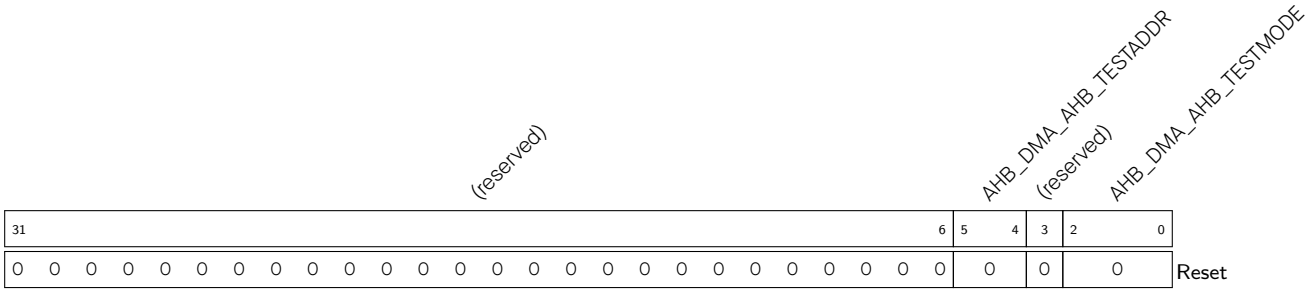
AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_TOTAL_EOF_CH*n*_INT。(WT)

AHB_DMA_OUTFIFO_OVF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUTFIFO_OVF_CH*n*_INT。(WT)

AHB_DMA_OUTFIFO_UDF_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUTFIFO_UDF_CH*n*_INT。(WT)

AHB_DMA_OUT_RESP_ERR_CH*n*_INT_CLR 写 1 清除 AHB_DMA_OUT_RESP_ERR_CH*n*_INT。(WT)

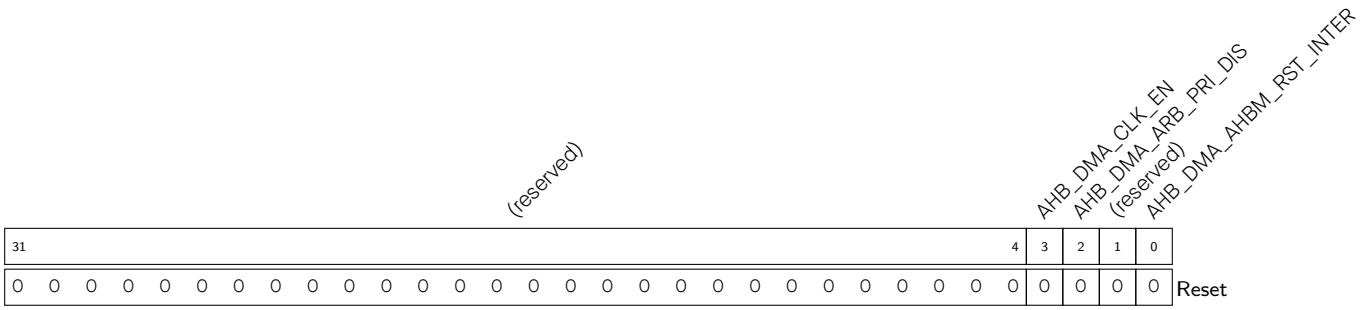
Register 5.9. AHB_DMA_AHB_TEST_REG (0x0060)



AHB_DMA_AHB_TESTMODE 保留。(R/W)

AHB_DMA_AHB_TESTADDR 保留。(R/W)

Register 5.10. AHB_DMA_MISC_CONF_REG (0x0064)



AHB_DMA_AHBM_RST_INTER 先写 1 再写 0 复位内部 AHB FSM。(R/W)

AHB_DMA_ARB_PRI_DIS 配置关闭优先级仲裁。

0：开启

1：关闭

(R/W)

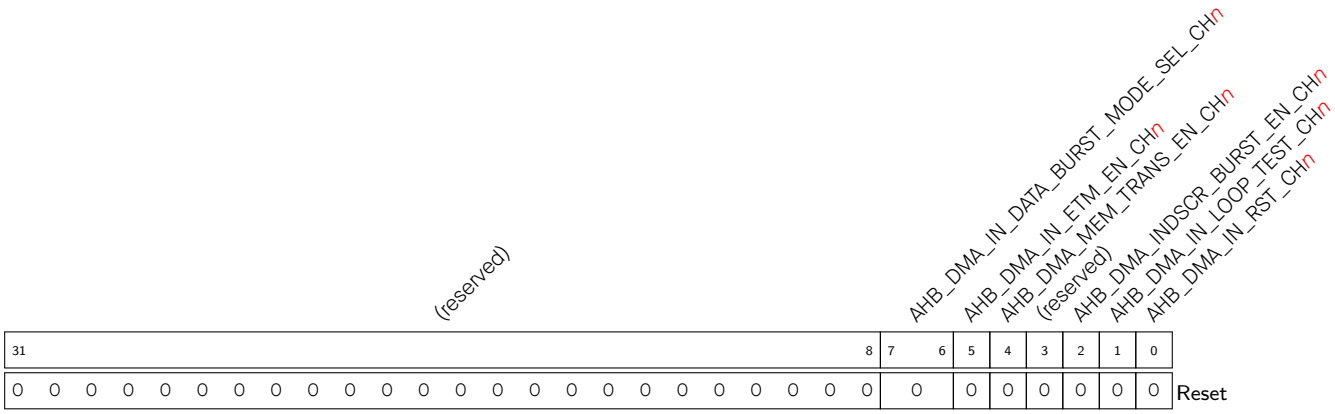
AHB_DMA_CLK_EN 配置 AHB DMA 时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 5.11. AHB_DMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0070+0xC0*n*)



AHB_DMA_IN_RST_CH*n* 先写 1 再写 0 复位接收通道 *n* 的 FSM 和 RX FIFO 指针。(R/W)

AHB_DMA_IN_LOOP_TEST_CH*n* 配置接收链表回写时 owner 位的值。(R/W)

AHB_DMA_INDSCR_BURST_EN_CH*n* 配置开启 INCR 突发传输让接收通道 *n* 读取描述符。

0：关闭

1：开启

(R/W)

AHB_DMA_MEM_TRANS_EN_CH*n* 配置开启存储器到存储器数据传输。

0：关闭

1：开启

(R/W)

AHB_DMA_IN_ETM_EN_CH*n* 配置开启接收通道 *n* 的 ETM 控制。

0：关闭

1：开启

(R/W)

AHB_DMA_IN_DATA_BURST_MODE_SEL_CH*n* 配置 RX 通道 *n* 最大突发长度。

0：SINGLE

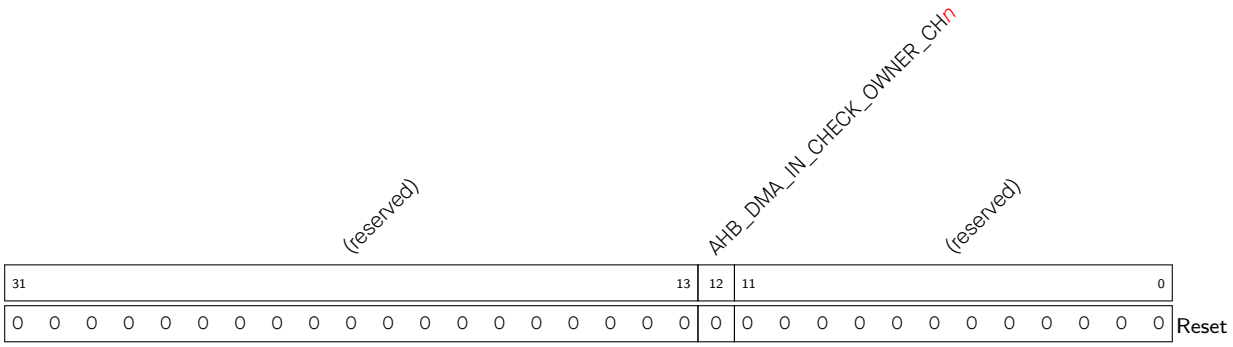
1：INCR4

2：INCR8

3：保留

(R/W)

Register 5.12. AHB_DMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0074+0xC0*n*)



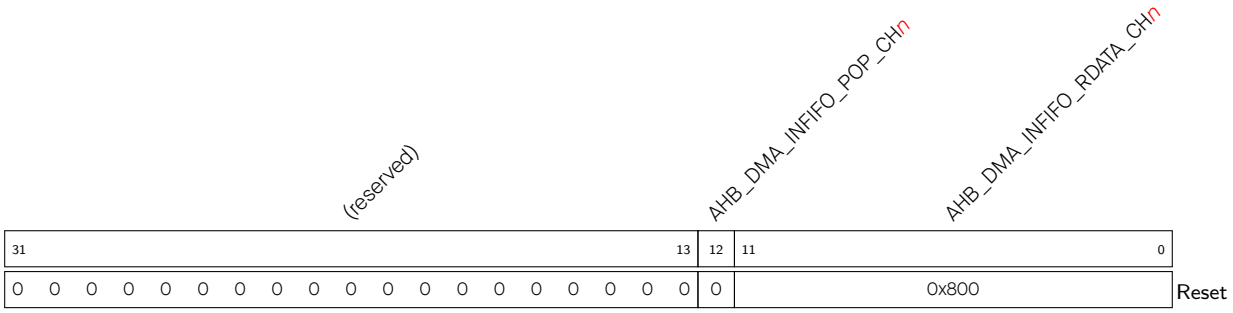
AHB_DMA_IN_CHECK_OWNER_CH*n* 配置开启接收通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

Register 5.13. AHB_DMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x007C+0xC0*n*)



AHB_DMA_INFIFO_RDATA_CH*n* 表示从 AHB DMA RX FIFO 弹出的数据。(RO)

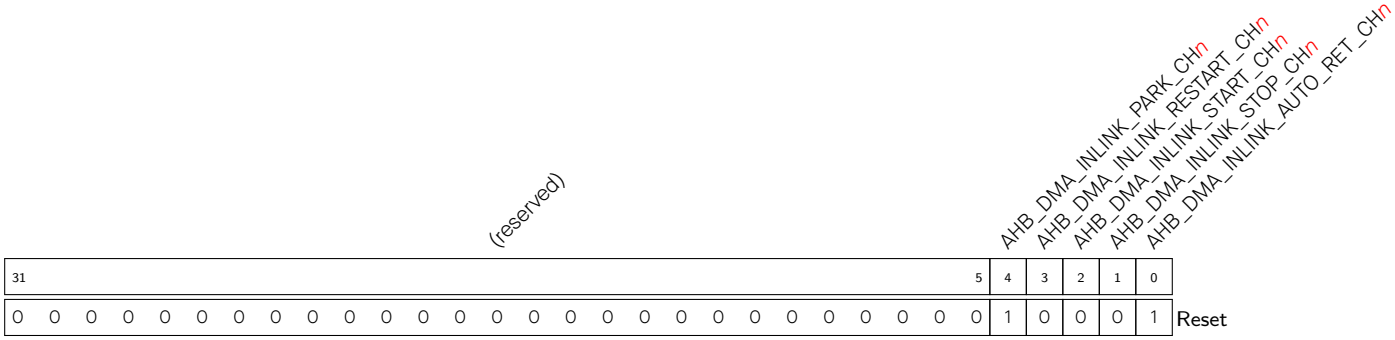
AHB_DMA_INFIFO_POP_CH*n* 配置从 AHB DMA RX FIFO 中弹出数据。

0：无效值。没有作用

1：弹出

(WT)

Register 5.14. AHB_DMA_IN_LINK_CH*n*_REG (*n*: 0-2) (0x0080+0xC0*n*)



AHB_DMA_INLINK_AUTO_RET_CH*n* 配置在接收数据有错时返回到当前接收链表描述符的地址。

0：不返回

1：返回

(R/W)

AHB_DMA_INLINK_STOP_CH*n* 配置接收通道 *n* 停止接收数据。

0：无效值。没有作用

1：停止

(WT)

AHB_DMA_INLINK_START_CH*n* 配置开启接收通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

AHB_DMA_INLINK_RESTART_CH*n* 配置重启接收通道 *n* 进行 AHB DMA 传输。

0：无效值。没有作用

1：重启

(WT)

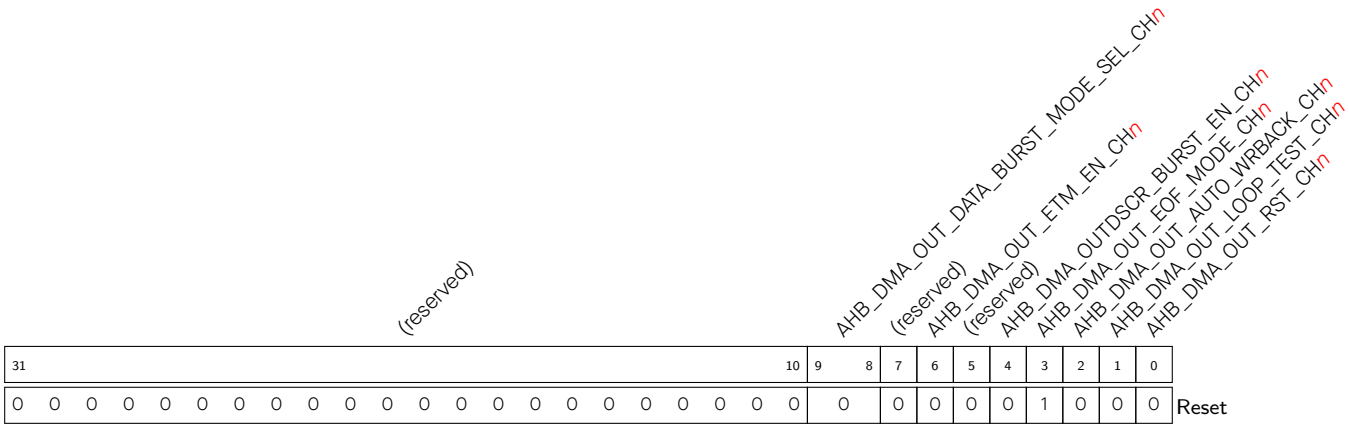
AHB_DMA_INLINK_PARK_CH*n* 表示接收链表描述符 FSM 的状态。

0：运行

1：空闲

(RO)

Register 5.15. AHB_DMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x00D0+0xC0*(*n*-1))



AHB_DMA_OUT_RST_CH*n* 配置发送通道 *n* FSM 和 TX FIFO 指针的复位状态。

0：复位释放

1：复位

(R/W)

AHB_DMA_OUT_LOOP_TEST_CH*n* 配置发送链表回写时 owner 位的值。(R/W)

AHB_DMA_OUT_AUTO_WRBACK_CH*n* 配置在发送完 TX FIFO 中的所有数据时是否开启发送链表自动回写。

0：关闭

1：开启

(R/W)

AHB_DMA_OUT_EOF_MODE_CH*n* 配置生成 EOF 标志的时间。

0：待发送数据推入 AHB DMA FIFO 时生成发送通道 *n* 的 EOF 标志。

1：待发送数据从 AHB DMA FIFO 中弹出时生成发送通道 *n* 的 EOF 标志。

(R/W)

AHB_DMA_OUTDSCR_BURST_EN_CH*n* 配置开启 INCR 突发传输，让发送通道 *n* 读取描述符。

0：关闭

1：开启

(R/W)

AHB_DMA_OUT_ETM_EN_CH*n* 配置开启发送通道 *n* 的 ETM 控制。

0：关闭

1：开启

(R/W)

AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH*n* 配置 TX 通道 *n* 最大突发长度。

0：SINGLE

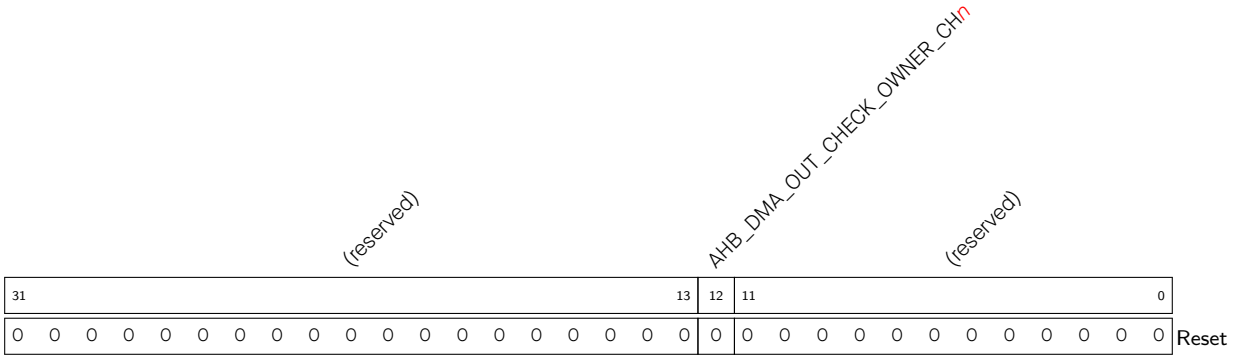
1：INCR4

2：INCR8

3：保留

(R/W)

Register 5.16. AHB_DMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x00D4+0xC0*n*)



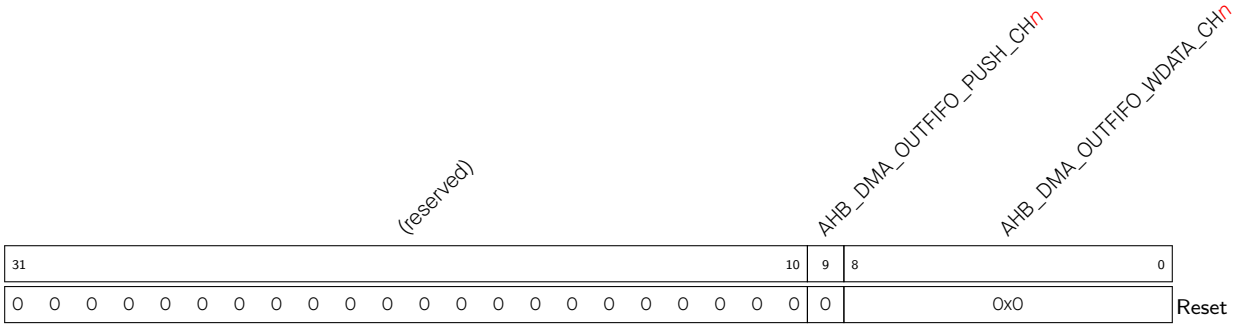
AHB_DMA_OUT_CHECK_OWNER_CH*n* 配置开启发送通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

Register 5.17. AHB_DMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x00DC+0xC0*n*)



AHB_DMA_OUTFIFO_WDATA_CH*n* 表示需要推入 AHB DMA TX FIFO 中的数据。(R/W)

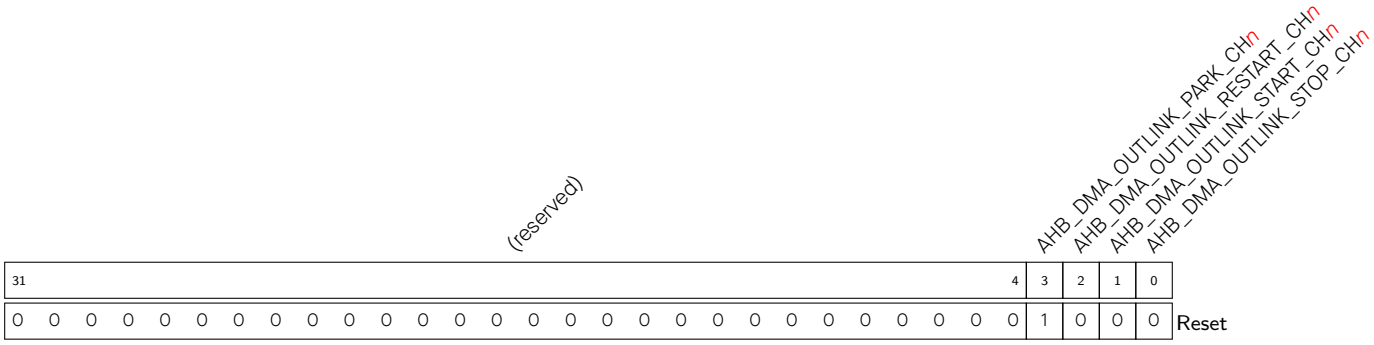
AHB_DMA_OUTFIFO_PUSH_CH*n* 配置将数据推入 AHB DMA TX FIFO。

0：无效值。没有作用

1：推入

(WT)

Register 5.18. AHB_DMA_OUT_LINK_CH*n*_REG (*n*: 0-2) (0x00E0+0xC0**n*)



AHB_DMA_OUTLINK_STOP_CH*n* 配置发送通道 *n* 停止发送数据。

0：无效值。没有作用

1：停止

(WT)

AHB_DMA_OUTLINK_START_CH*n* 配置开启发送通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

AHB_DMA_OUTLINK_RESTART_CH*n* 配置重启发送通道 *n* 进行 AHB DMA 传输。

0：无效值。没有作用

1：重启

(WT)

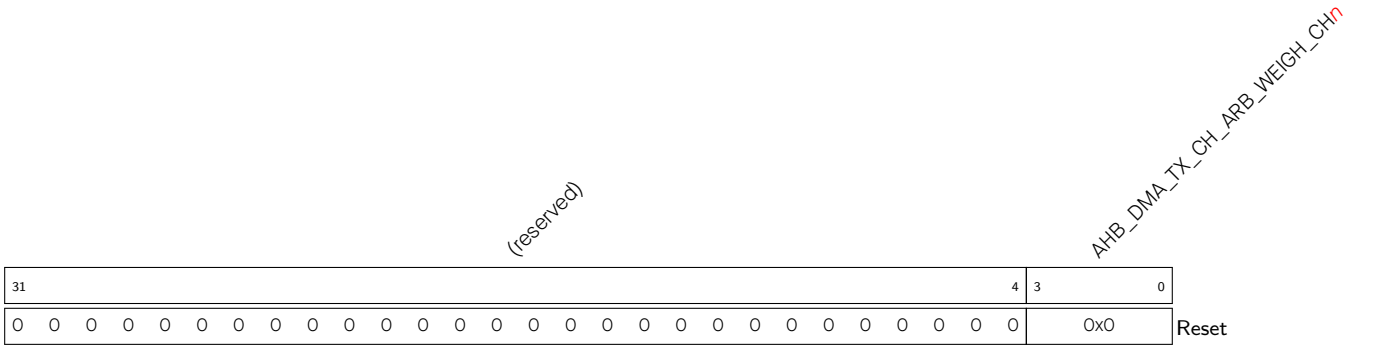
AHB_DMA_OUTLINK_PARK_CH*n* 表示发送链表描述符 FSM 的状态。

0：运行

1：空闲

(RO)

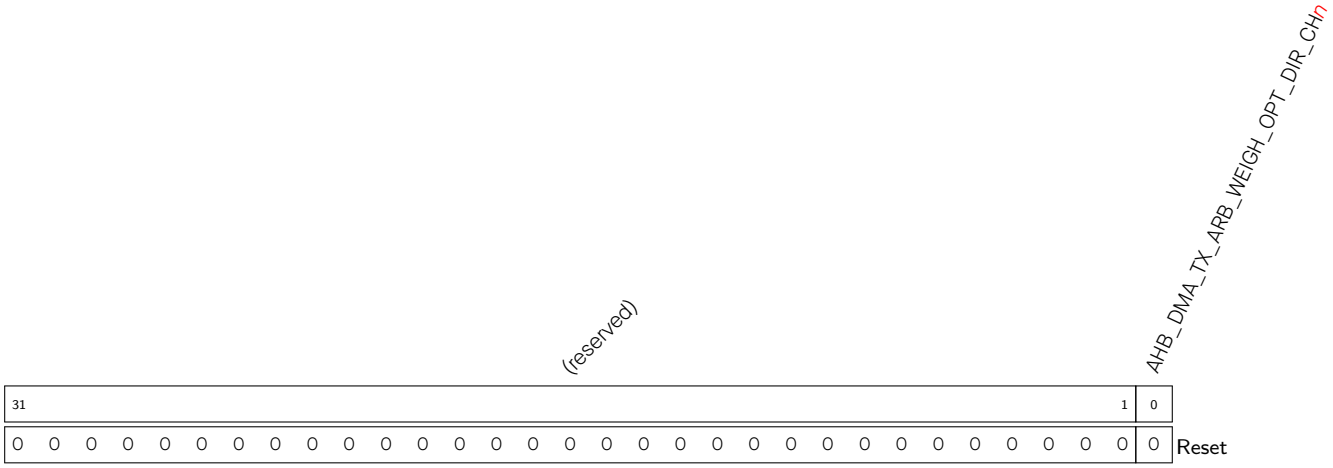
Register 5.19. AHB_DMA_TX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x02DC+0x28**n*)



AHB_DMA_TX_CH_ARB_WEIGH_CH*n* 配置 TX 通道 *n* 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

Register 5.20. AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x02E0+0x28**n*)



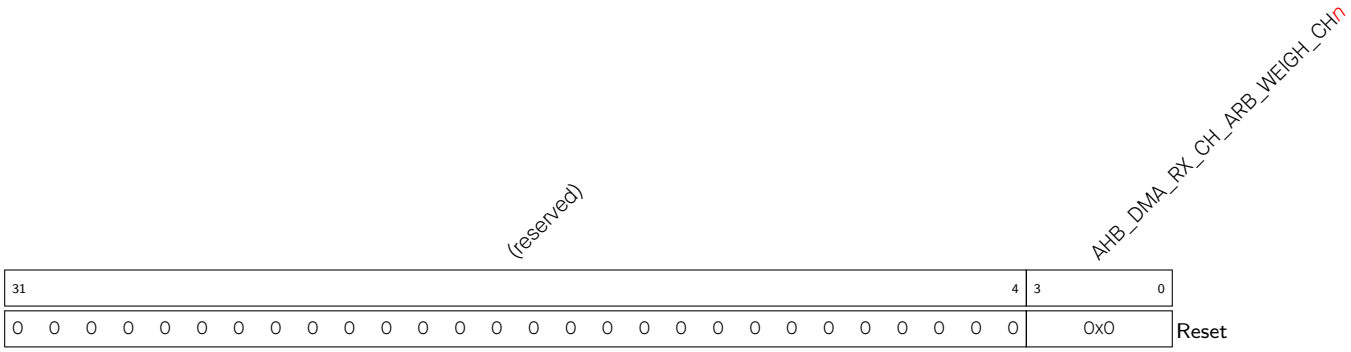
AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CH*n* 配置发送通道 *n* 是否开启权重仲裁优化。

0：关闭

1：开启

(R/W)

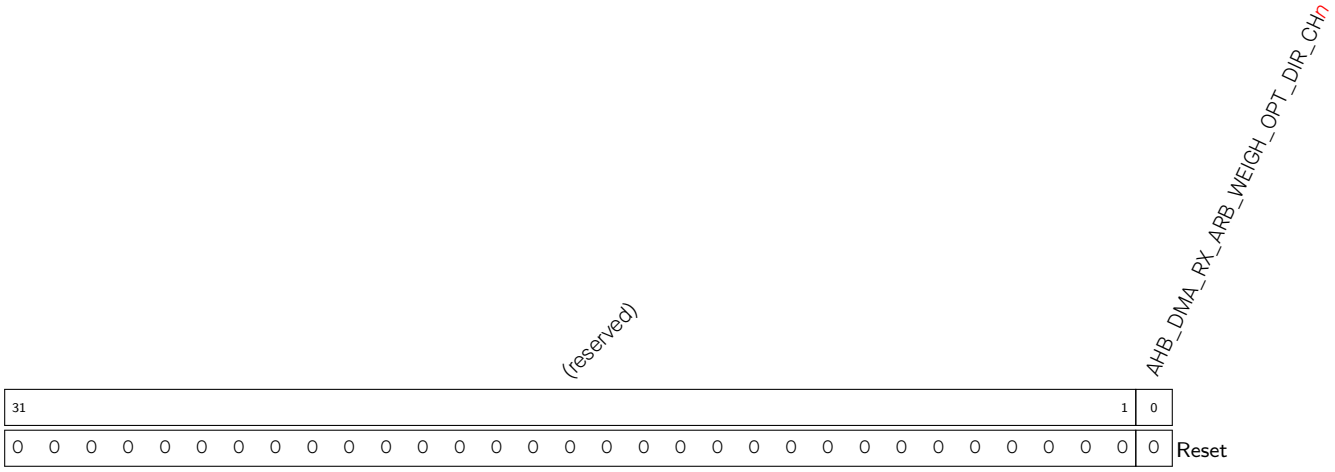
Register 5.21. AHB_DMA_RX_CH_ARB_WEIGH_CH*n*_REG (*n*: 0-2) (0x0354+0x28**n*)



AHB_DMA_RX_CH_ARB_WEIGH_CH*n* 配置 RX 通道 *n* 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

Register 5.22. AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n*_REG (*n*: 0-2) (0x0358+0x28**n*)



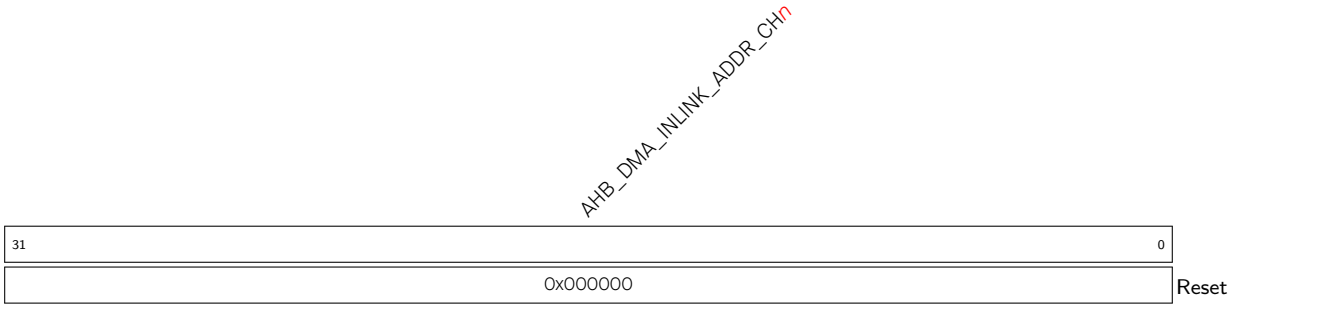
AHB_DMA_RX_ARB_WEIGH_OPT_DIR_CH*n* 配置接收通道 *n* 是否开启权重仲裁优化。

0：关闭

1：开启

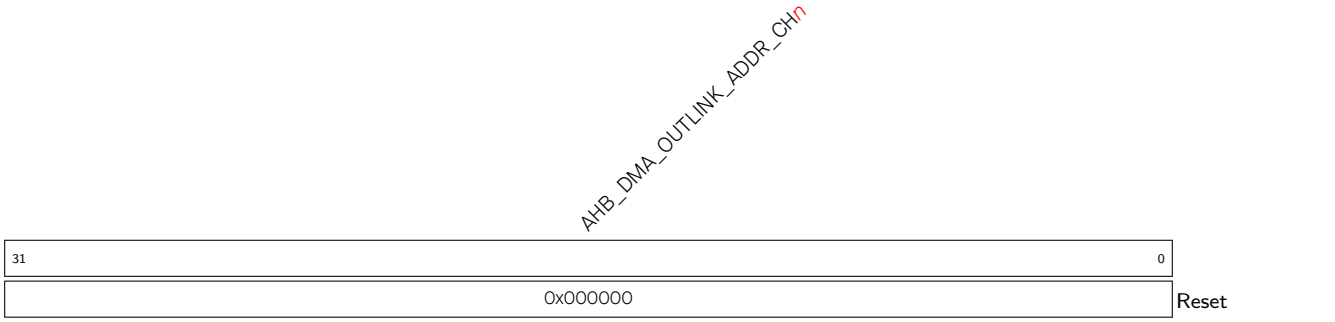
(R/W)

Register 5.23. AHB_DMA_IN_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03AC+0x4**n*)



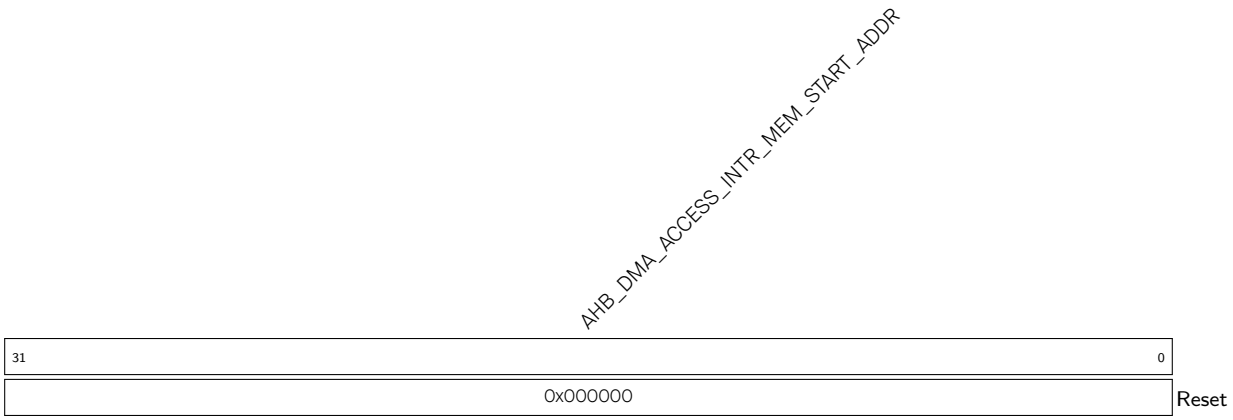
AHB_DMA_INLINK_ADDR_CH*n* 表示第一个接收链表描述符的地址。(R/W)

Register 5.24. AHB_DMA_OUT_LINK_ADDR_CH*n*_REG (*n*: 0-2) (0x03B8+0x4**n*)



AHB_DMA_OUTLINK_ADDR_CH*n* 表示第一个发送链表描述符的地址。(R/W)

Register 5.25. AHB_DMA_INTR_MEM_START_ADDR_REG (0x03C4)



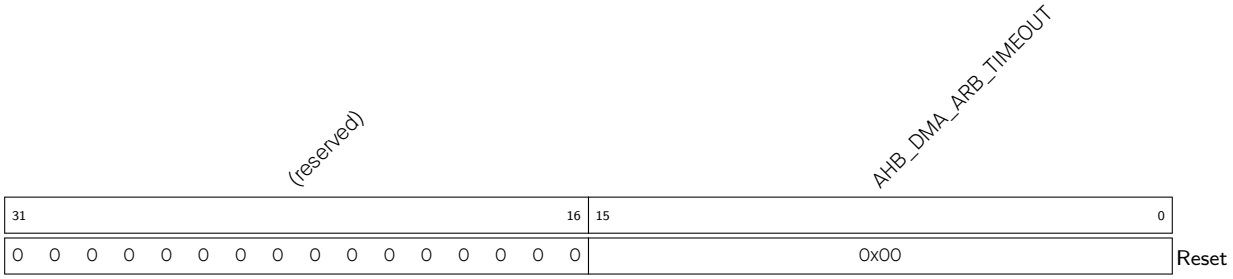
AHB_DMA_ACCESS_INTR_MEM_START_ADDR 配置可访问地址空间的起始地址。(注意：只可配置内部存储地址空间，不可配置片外地址范围，可能会发生意外问题)。(R/W)

Register 5.26. AHB_DMA_INTR_MEM_END_ADDR_REG (0x03C8)



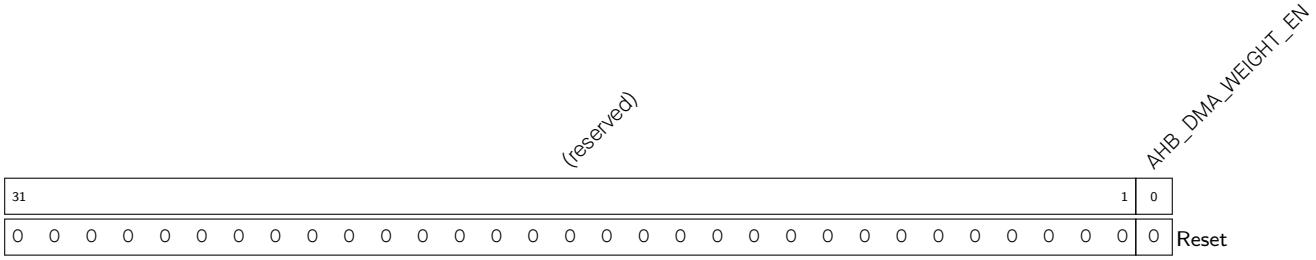
AHB_DMA_ACCESS_INTR_MEM_END_ADDR 配置可访问地址空间的结束地址。(R/W)

Register 5.27. AHB_DMA_ARB_TIMEOUT_REG (0x03DC)



AHB_DMA_ARB_TIMEOUT 配置权重仲裁时间片。单位：AHB 总线时钟周期。(R/W)

Register 5.28. AHB_DMA_WEIGHT_EN_REG (0x0400)



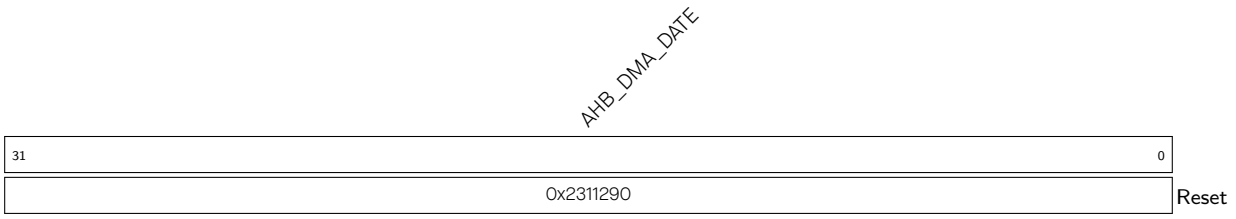
AHB_DMA_WEIGHT_EN 配置开启权重仲裁。

0：关闭

1：开启

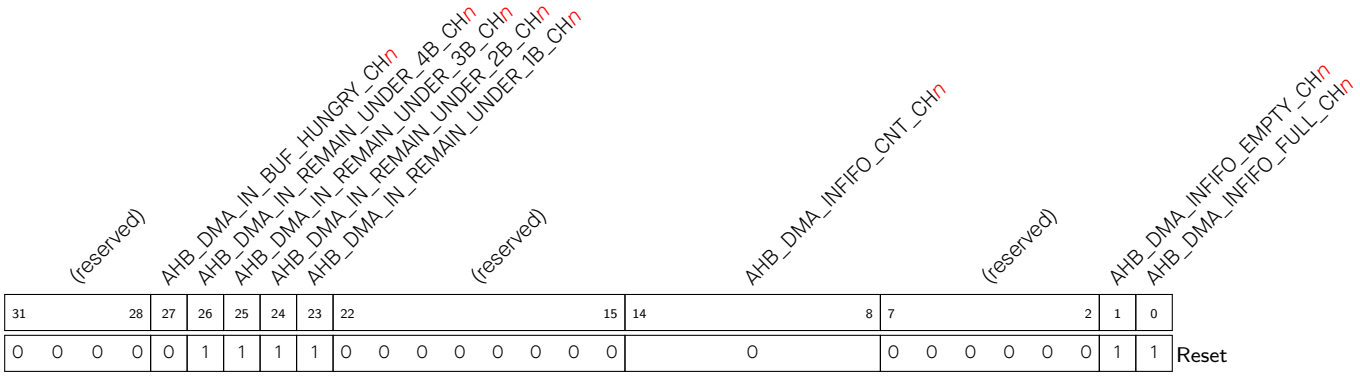
(R/W)

Register 5.29. AHB_DMA_DATE_REG (0x0068)



AHB_DMA_DATE 版本控制寄存器。(R/W)

Register 5.30. AHB_DMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0078+0xC0*n*)



AHB_DMA_INFIFO_FULL_CH*n* 表示 L1 RX FIFO 是否为满。

0：未满

1：已满

(RO)

AHB_DMA_INFIFO_EMPTY_CH*n* 表示 L1 RX FIFO 是否为空。

0：未空

1：已空

(RO)

AHB_DMA_INFIFO_CNT_CH*n* 表示接收通道 *n* L1 RX FIFO 中的字节数。(RO)

AHB_DMA_IN_REMAIN_UNDER_1B_CH*n* 保留。(RO)

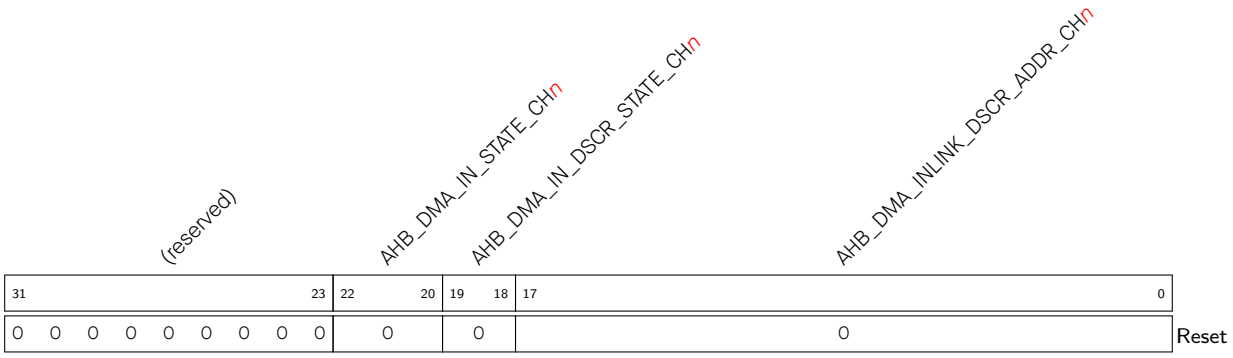
AHB_DMA_IN_REMAIN_UNDER_2B_CH*n* 保留。(RO)

AHB_DMA_IN_REMAIN_UNDER_3B_CH*n* 保留。(RO)

AHB_DMA_IN_REMAIN_UNDER_4B_CH*n* 保留。(RO)

AHB_DMA_IN_BUF_HUNGRY_CH*n* 保留。(RO)

Register 5.31. AHB_DMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0084+0xC0*n*)

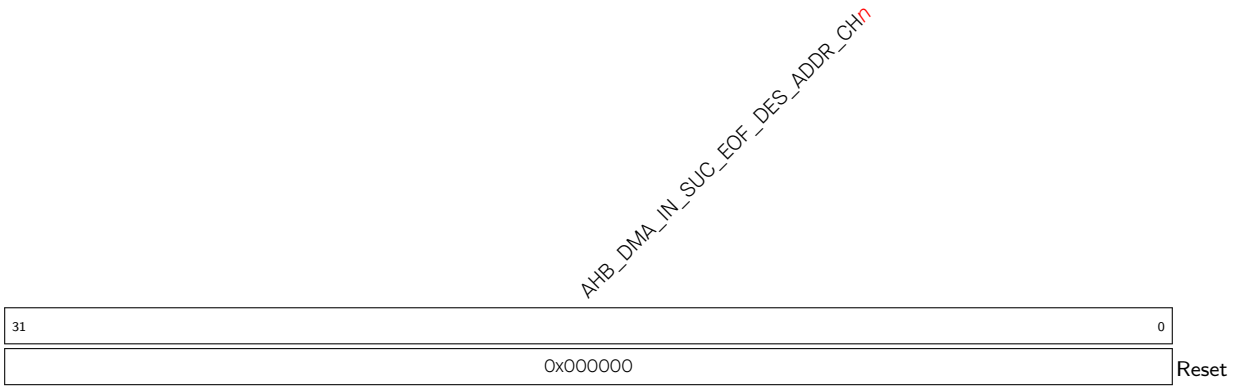


AHB_DMA_INLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的接收链表描述符所在地址的低 18 位。如果当前处理的接收链表描述符是最后一个描述符，则该字段表示当前处理的接收链表描述符地址。(RO)

AHB_DMA_IN_DSCR_STATE_CH*n* 保留。(RO)

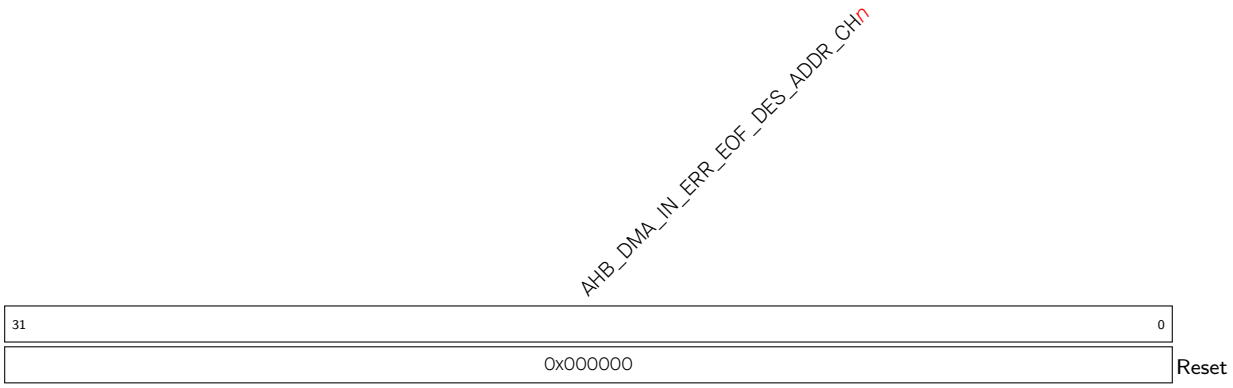
AHB_DMA_IN_STATE_CH*n* 保留。(RO)

Register 5.32. AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0088+0xC0*n*)



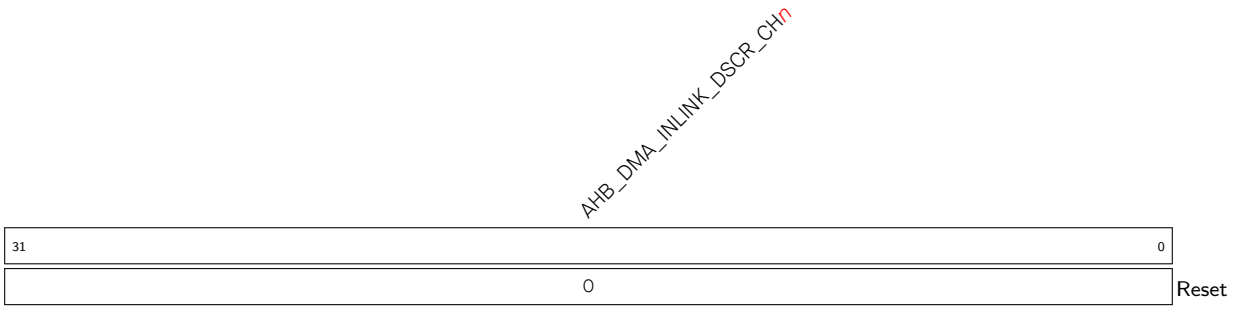
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的接收链表描述符的地址。(RO)

Register 5.33. AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x008C+0xC0*n*)



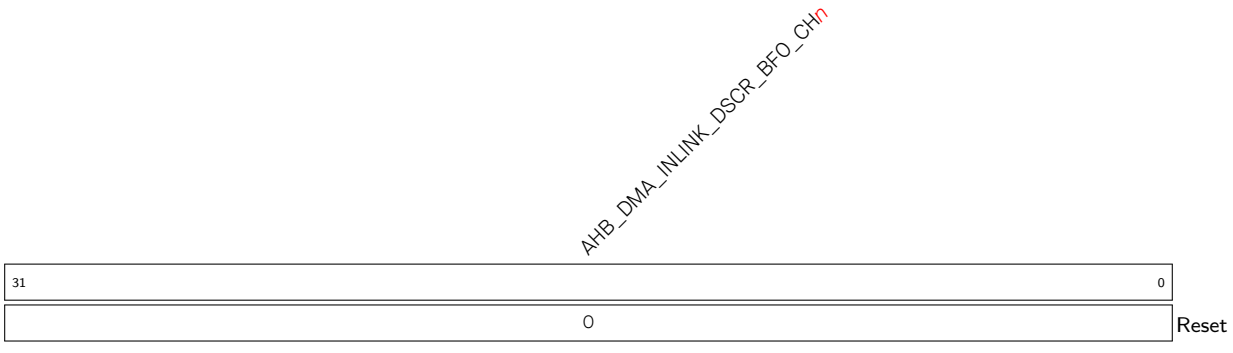
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH*n* 表示当前接收数据有错时，相应接收链表描述符的位置。仅对 UHCI 有效。(RO)

Register 5.34. AHB_DMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0090+0xC0*n*)



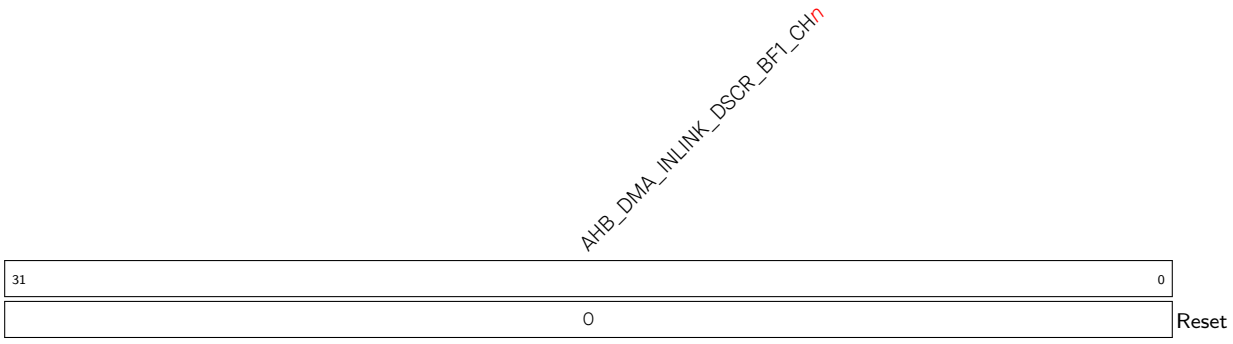
AHB_DMA_INLINK_DSCR_CH*n* 表示当前已预读取的接收链表描述符指向的下一个接收链表描述符地址 x+1。(RO)

Register 5.35. AHB_DMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0094+0xC0*n*)



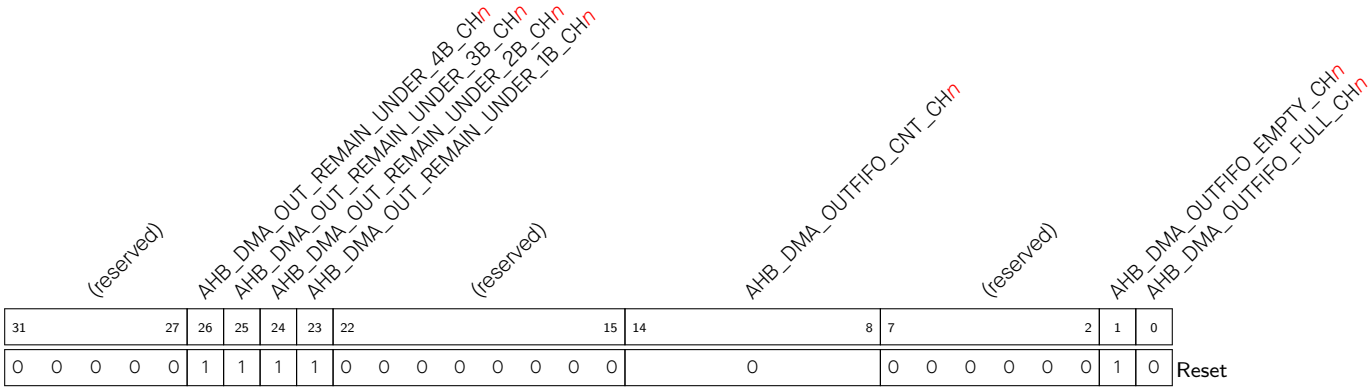
AHB_DMA_INLINK_DSCR_BF0_CH*n* 表示当前已预读取的接收链表描述符所在地址 x。(RO)

Register 5.36. AHB_DMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0098+0xC0*n*)



AHB_DMA_INLINK_DSCR_BF1_CH*n* 表示前一个已预读取的接收链表描述符所在地址 x-1。(RO)

Register 5.37. AHB_DMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x00D8+0xC0*n*)



AHB_DMA_OUTFIFO_FULL_CH*n* 表示 L1 TX FIFO 是否为满。

0：未满

1：已满

(RO)

AHB_DMA_OUTFIFO_EMPTY_CH*n* 表示 L1 TX FIFO 是否为空。

0：未空

1：已空

(RO)

AHB_DMA_OUTFIFO_CNT_CH*n* 表示发送通道 *n* L1 TX FIFO 中的字节数。(RO)

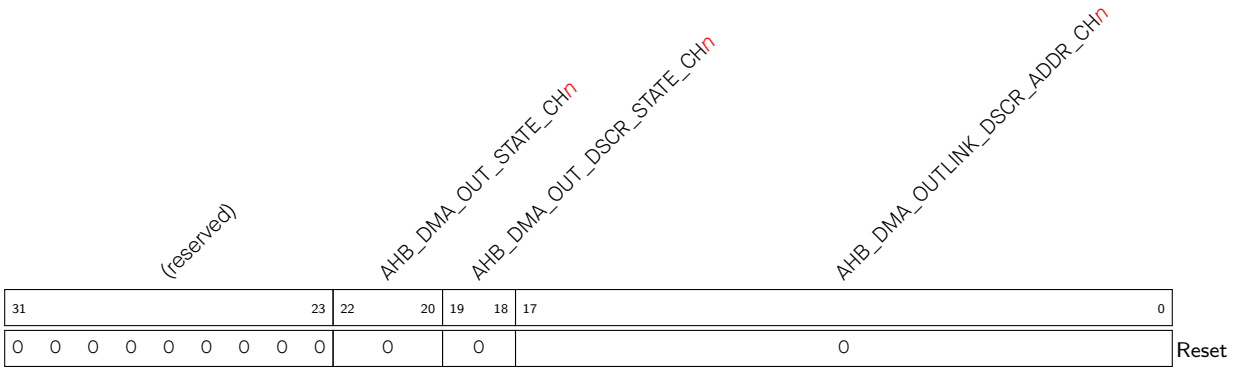
AHB_DMA_OUT_REMAIN_UNDER_1B_CH*n* 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_2B_CH*n* 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_3B_CH*n* 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_4B_CH*n* 保留。(RO)

Register 5.38. AHB_DMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x00E4+0xC0*n*)

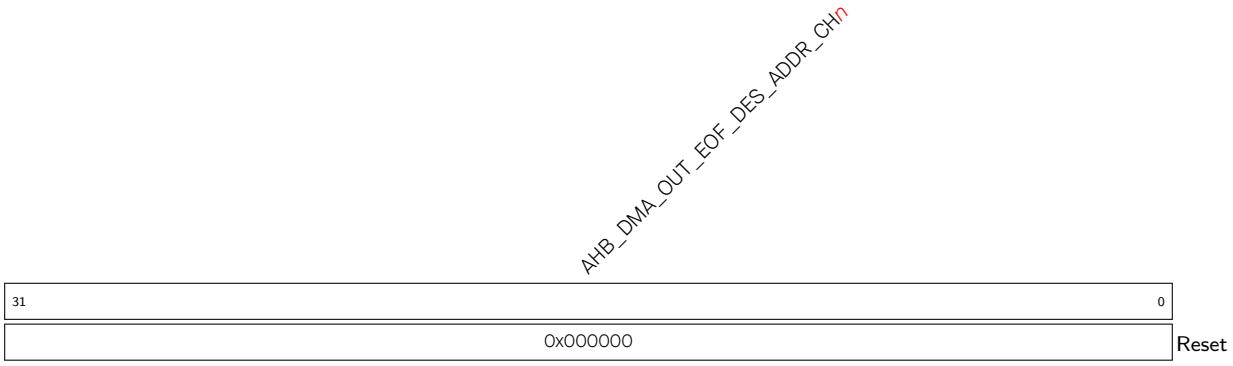


AHB_DMA_OUTLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的发送接收链表描述符所在地址的低 18 位。如果当前处理的发送链表描述符是最后一个描述符，则该字段表示当前处理的发送链表描述符地址。(RO)

AHB_DMA_OUT_DSCR_STATE_CH*n* 保留。(RO)

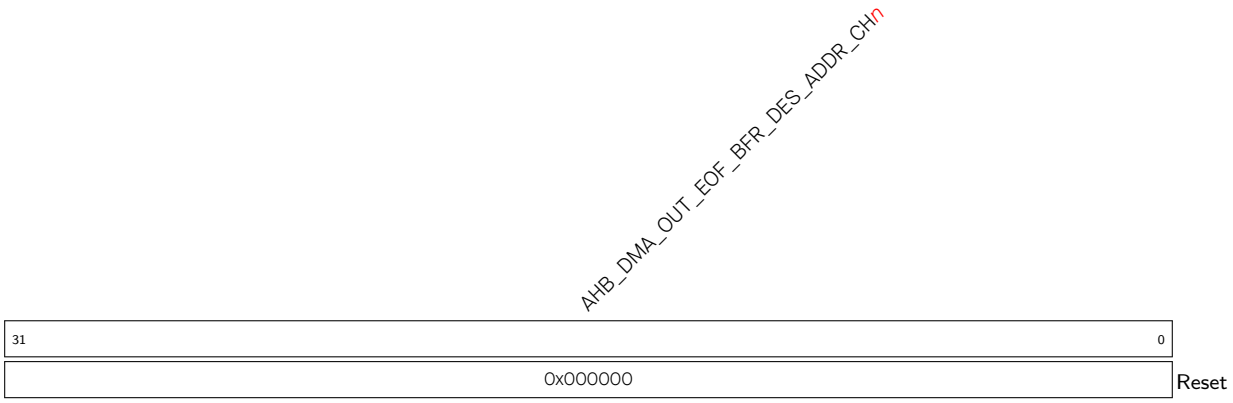
AHB_DMA_OUT_STATE_CH*n* 保留。(RO)

Register 5.39. AHB_DMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00E8+0xC0*n*)



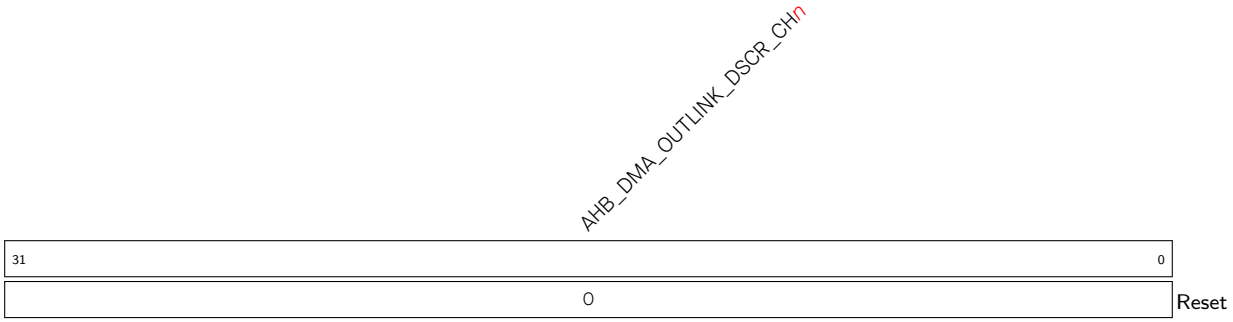
AHB_DMA_OUT_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的发送链表描述符的地址。(RO)

Register 5.40. AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00EC+0xC0*n*)



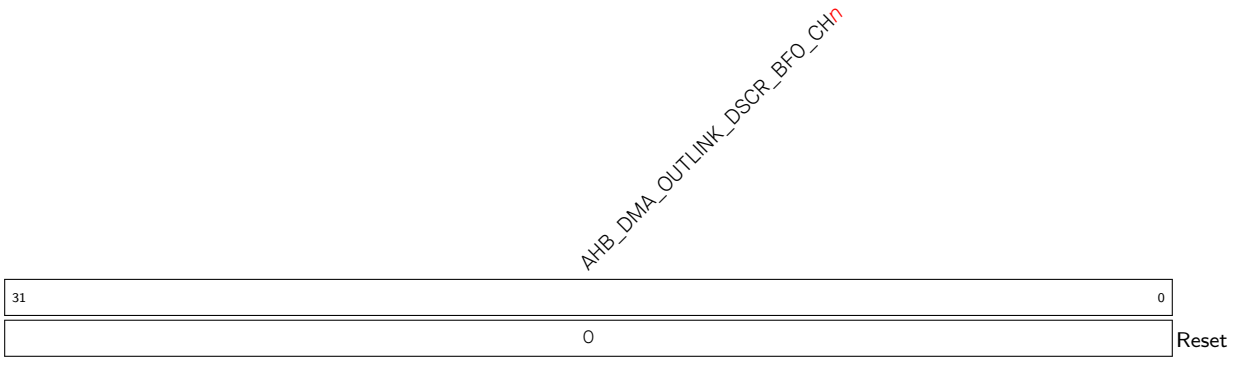
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH*n* 表示倒数第二个发送链表描述符的地址。(RO)

Register 5.41. AHB_DMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x00F0+0xC0*n*)



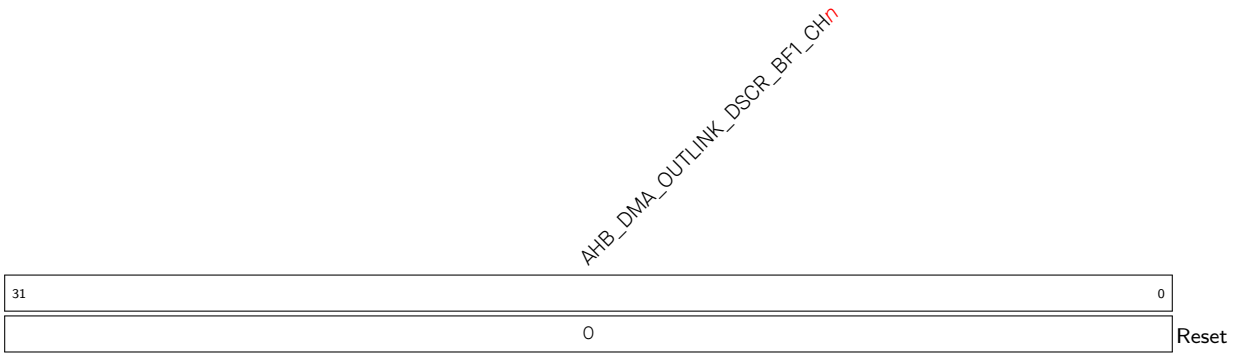
AHB_DMA_OUTLINK_DSCR_CH*n* 表示当前已预读取的发送链表描述符指向的下一个发送链表描述符地址 y+1。(RO)

Register 5.42. AHB_DMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x00F4+0xC0*n*)



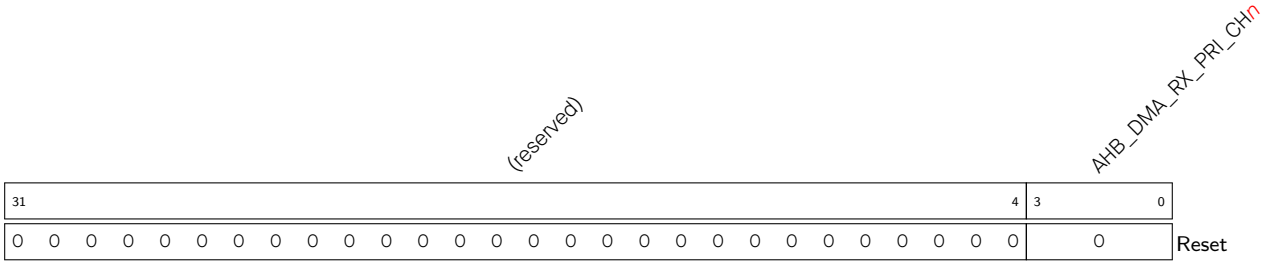
AHB_DMA_OUTLINK_DSCR_BF0_CH*n* 表示当前已预读取的发送链表描述符所在地址 y。(RO)

Register 5.43. AHB_DMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x00F8+0xC0*n*)



AHB_DMA_OUTLINK_DSCR_BF1_CH*n* 表示前一个已预读取的发送链表描述符所在地址 y-1。(RO)

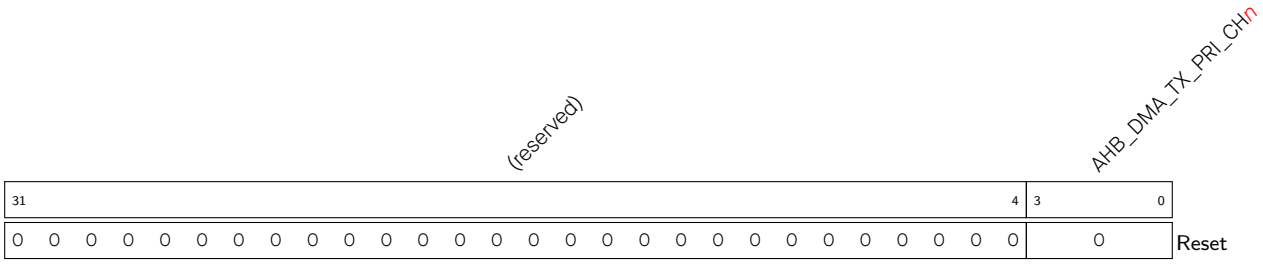
Register 5.44. AHB_DMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x009C+0xC0*n*)



AHB_DMA_RX_PRI_CH*n* 配置接收通道 *n* 的优先级。值越大，优先级越高。

取值范围：0 ~ 5 (R/W)

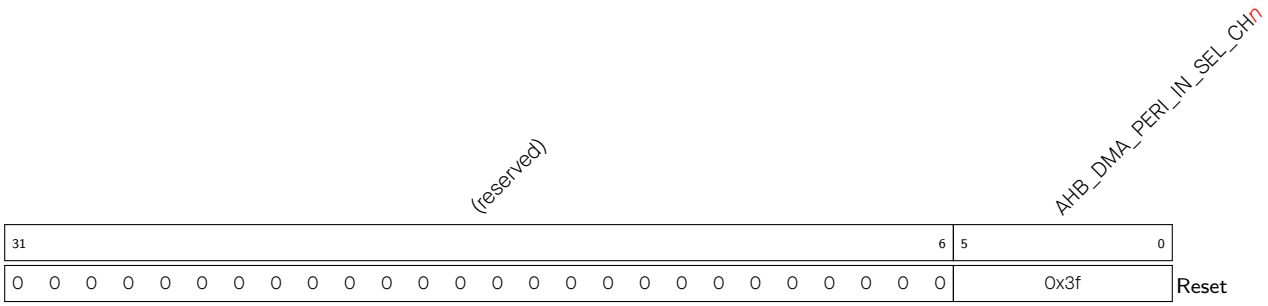
Register 5.45. AHB_DMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x00FC+0xC0*n*)



AHB_DMA_TX_PRI_CH*n* 配置发送通道 *n* 的优先级。值越大，优先级越高。

取值范围：0 ~ 5 (R/W)

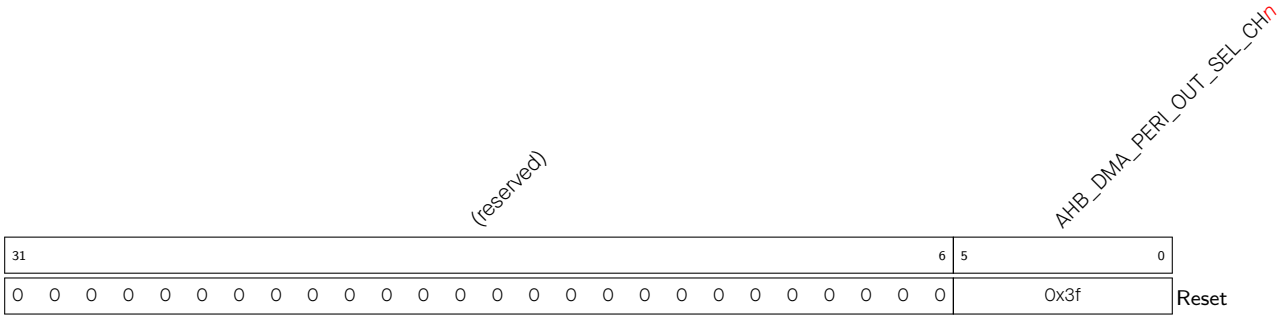
Register 5.46. AHB_DMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x00A0+0xC0**n*)



AHB_DMA_PERI_IN_SEL_CH*n* 配置连接接收通道 *n* 的外设。

0：Dummy
1：GP-SPI
2：UHCI
3：I2S
4：Dummy
5：Dummy
6：AES
7：SHA
8：ADC
9：PARLIO
10：Dummy
11 ~ 15：Dummy
16 ~ 63：无效值
(R/W)

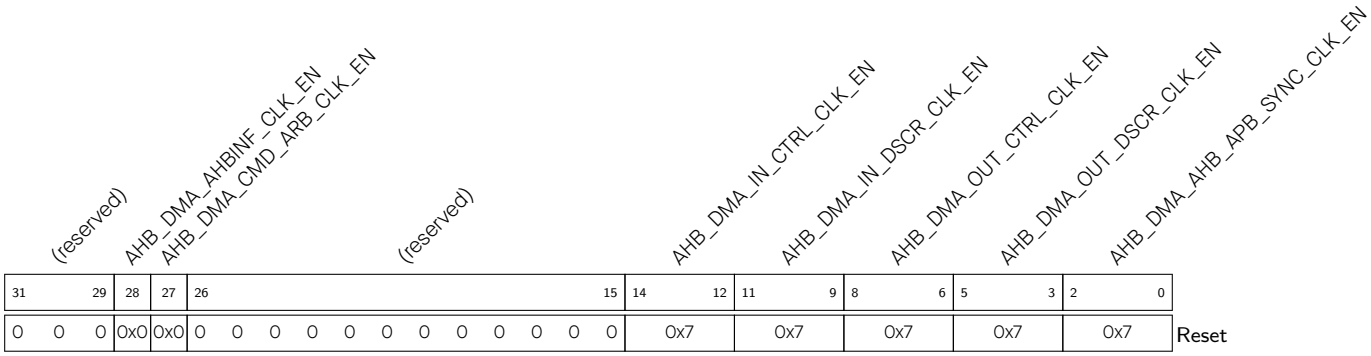
Register 5.47. AHB_DMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0100+0xC0**n*)



AHB_DMA_PERI_OUT_SEL_CH*n* 配置连接发送通道 *n* 的外设。

0：Dummy
1：GP-SPI
2：UHCI
3：I2S
4：Dummy
5：Dummy
6：AES
7：SHA
8：ADC
9：PARLIO
10：Dummy
11 ~ 15：Dummy
16 ~ 63：无效值
(R/W)

Register 5.48. AHB_DMA_MODULE_CLK_EN_REG (0x0404)



AHB_DMA_AHB_APB_SYNC_CLK_EN 配置强制开启 GDMA 寄存器配置同步模块时钟。

位 0 对应 TX、RX 通道 0，位 1 对应 TX、RX 通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_OUT_DSCR_CLK_EN 配置强制开启 GDMA TX 链表描述符处理模块时钟。

位 0 对应发送通道 0，位 1 对应发送通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_OUT_CTRL_CLK_EN 配置强制开启 GDMA TX 数据处理模块时钟。

位 0 对应发送通道 0，位 1 对应发送通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_IN_DSCR_CLK_EN 配置强制开启 GDMA RX 链表描述符处理模块时钟。

位 0 对应接收通道 0，位 1 对应接收通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_IN_CTRL_CLK_EN 配置强制开启 GDMA RX 数据处理模块时钟。

位 0 对应接收通道 0，位 1 对应 TR 通道 1，以此类推。

0：不强制开启

1：强制开启

(R/W)

AHB_DMA_CMD_ARB_CLK_EN 配置强制开启 GDMA 仲裁模块时钟。

0：不强制开启

1：强制开启

(R/W)

见下页...

Register 5.48. AHB_DMA_MODULE_CLK_EN_REG (0x0404)

接上页...

AHB_DMA_AHBINF_CLK_EN 配置强制开启 GDMA 总线接口处理模块时钟。

0：不强制开启

1：强制开启

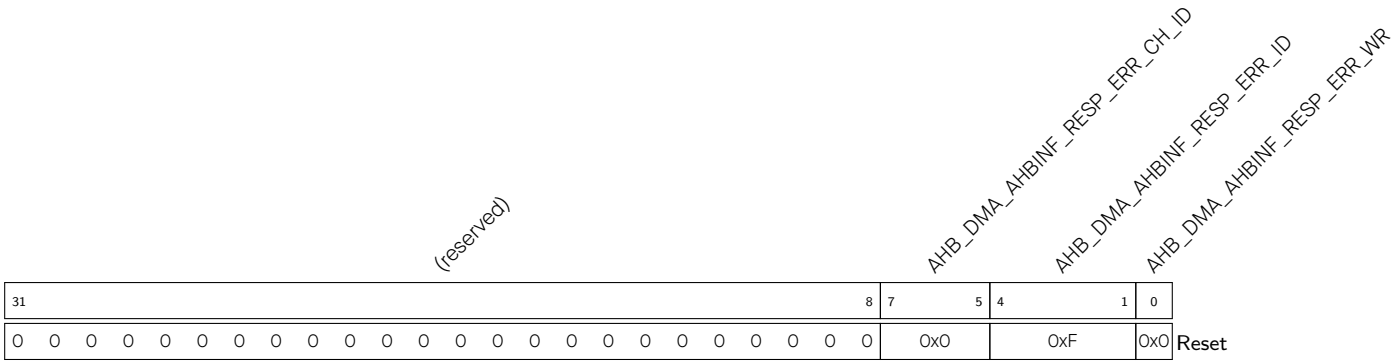
(R/W)

Register 5.49. AHB_DMA_AHBINF_RESP_ERR_STATUS0_REG (0x0408)



AHB_DMA_AHBINF_RESP_ERR_ADDR 表示 GDMA 收到 AHB 总线错误响应对应传输的地址。(RO)

Register 5.50. AHB_DMA_AHBINF_RESP_ERR_STATUS1_REG (0x040C)



AHB_DMA_AHBINF_RESP_ERR_CH_ID 表示 GDMA 收到 AHB 总线错误响应对应传输的通道 ID。

位 [0:1]：通道号。范围：0 ~ 2

位 [2]：通道方向。0：RX 方向

1：TX 方向

(RO)

AHB_DMA_AHBINF_RESP_ERR_ID 表示 GDMA 收到 AHB 总线错误响应对应传输的传输 ID。

1：GP-SPI 和存储器之间的传输

2：UHCI 和存储器之间的传输

3：I2S 和存储器之间的传输

6：AES 和存储器之间的传输

7：SHA 和存储器之间的传输

8：ADC 和存储器之间的传输

9：PARLIO 和存储器之间的传输

10：通道 0 存储器到存储器的传输

11：通道 1 存储器到存储器的传输

12：通道 2 存储器到存储器的传输

其余值为无效值

(RO)

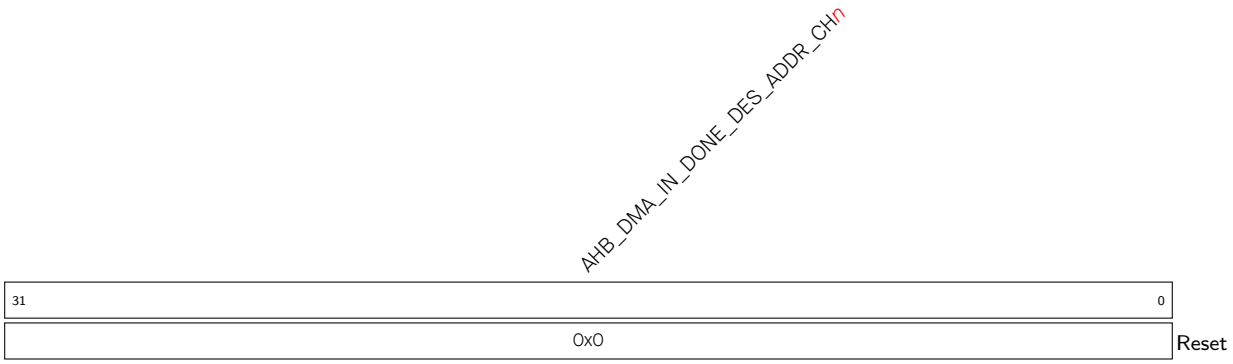
AHB_DMA_AHBINF_RESP_ERR_WR 表示 GDMA 收到 AHB 总线错误响应对应传输的类型。

0：读传输

1：写传输

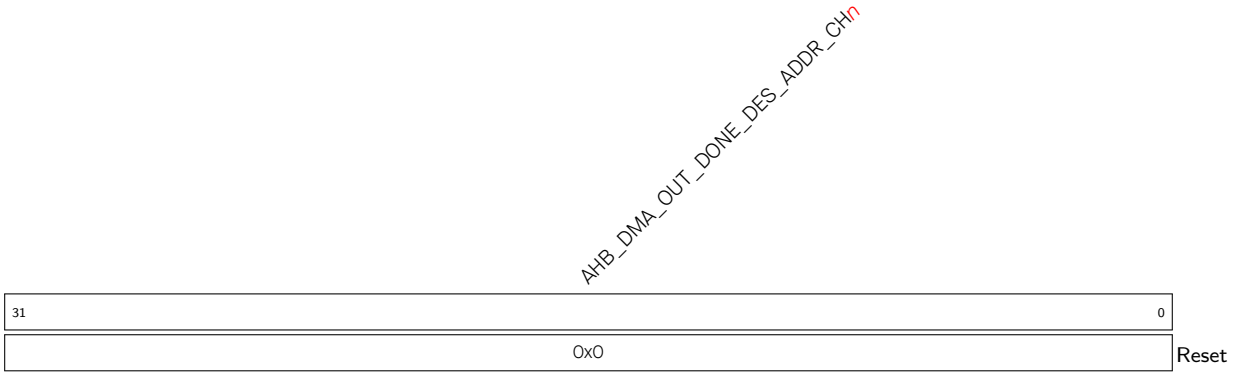
(RO)

Register 5.51. AHB_DMA_IN_DONE_DES_ADDR_CH*n*_REG(*n*: 0-2) (0x0410+0x08**n*)



AHB_DMA_IN_DONE_DES_ADDR_CH*n* 表示 RX 方向传输完成时对应描述符的地址。(RO)

Register 5.52. AHB_DMA_OUT_DONE_DES_ADDR_CH*n*_REG(*n*: 0-2) (0x0414+0x08**n*)



AHB_DMA_OUT_DONE_DES_ADDR_CH*n* 表示 TX 方向传输完成时对应描述符的地址。(RO)

第 II 卷

存储器组织结构

该部分详细描述系统的存储器组织结构，例如 RAM、ROM、eFuse 和外部存储器的组织和映射，为理解与存储器相关的子系统提供了框架。

第 6 章

系统和存储器

6.1 概述

ESP32-C5 是一个超低功耗和高度集成的系统，集成了：

- 一颗高性能 RISC-V 32 位处理器 (HP CPU)，五级流水线架构，主频高达 240 MHz
- 一颗低功耗 RISC-V 32 位处理器 (LP CPU)，二级流水线架构，主频高达 48 MHz

所有的内部存储器、外部存储器以及外设都分布在 HP CPU 和 LP CPU 的总线上。

6.2 主要特性

- **地址空间**
  - 720 KB 内部存储器空间，可被指令总线或数据总线访问
  - 832 KB 外设地址空间
  - 32 MB 外部存储器虚拟地址空间，可被指令总线或数据总线访问
  - 384 KB 内部 DMA 地址空间
  - 32 MB 外部 DMA 地址空间
- **内部存储器**
  - 320 KB ROM
  - 384 KB HP SRAM
  - 16 KB LP SRAM
- **外部存储器**
  - 最大支持 32 MB 片外 flash
  - 最大支持 32 MB 片外 RAM
- **外设空间**
  - 总计 61 个模块/外设
- **GDMA**
  - 7 个模块/外设支持 GDMA

6.3 功能描述

6.3.1 地址映射

图 6.3-1 展示了系统结构与地址映射结构。所有非保留地址均可通过指令总线和数据总线进行访问，即指令总线和数据总线访问的地址空间一样。HP CPU 和 LP CPU 的数据总线与指令总线均为小端序。所有总线均为 32 位数据宽度。HP CPU 和 LP CPU 均可以通过数据总线进行单字节、双字节、四字节的数据访问。

HP CPU 支持：

- 直接通过数据总线和指令总线访问内部存储器。
  - 384 KB HP SRAM (0x4080_0000 ~ 0x4085_FFFF)
  - 16 KB LP SRAM (0x5000_0000 ~ 0x5000_3FFF)
- 通过 ROM-Cache 访问内部存储器。
  - 320 KB ROM (0x4000_0000 ~ 0x4004_FFFF)

> **说明：**
> 软件可以以 cache-able 或 noncache-able 的方式访问 ROM 的地址。其中 cache-able 访问方式是指通过 cache 进行访问；noncache-able 访问方式是不通过 cache 进行直接访问。cache-able 访问还是 noncache-able 访问则是由 CPU PMA (Physical Memory Attribution) 控制。

- 通过 cache 访问外部存储器，虚地址经过 MMU 映射到外部存储器的实地址空间。
  - 多达 32 MB 外部 flash (0x4200_0000 ~ 0x43FF_FFFF)
  - 多达 32 MB 外部 RAM (0x4200_0000 ~ 0x43FF_FFFF)
- 直接通过数据总线访问模块/外设。
  - HP CPU 外设
  - HP 外设
  - LP 外设

LP CPU 支持：

- 直接通过数据总线和指令总线访问 HP SRAM 和 LP SRAM。
  - 384 KB HP SRAM (0x4080_0000 ~ 0x4085_FFFF)
  - 16 KB LP SRAM (0x5000_0000 ~ 0x5000_3FFF)
- 直接通过数据总线访问模块/外设。
  - HP 外设
  - LP 外设

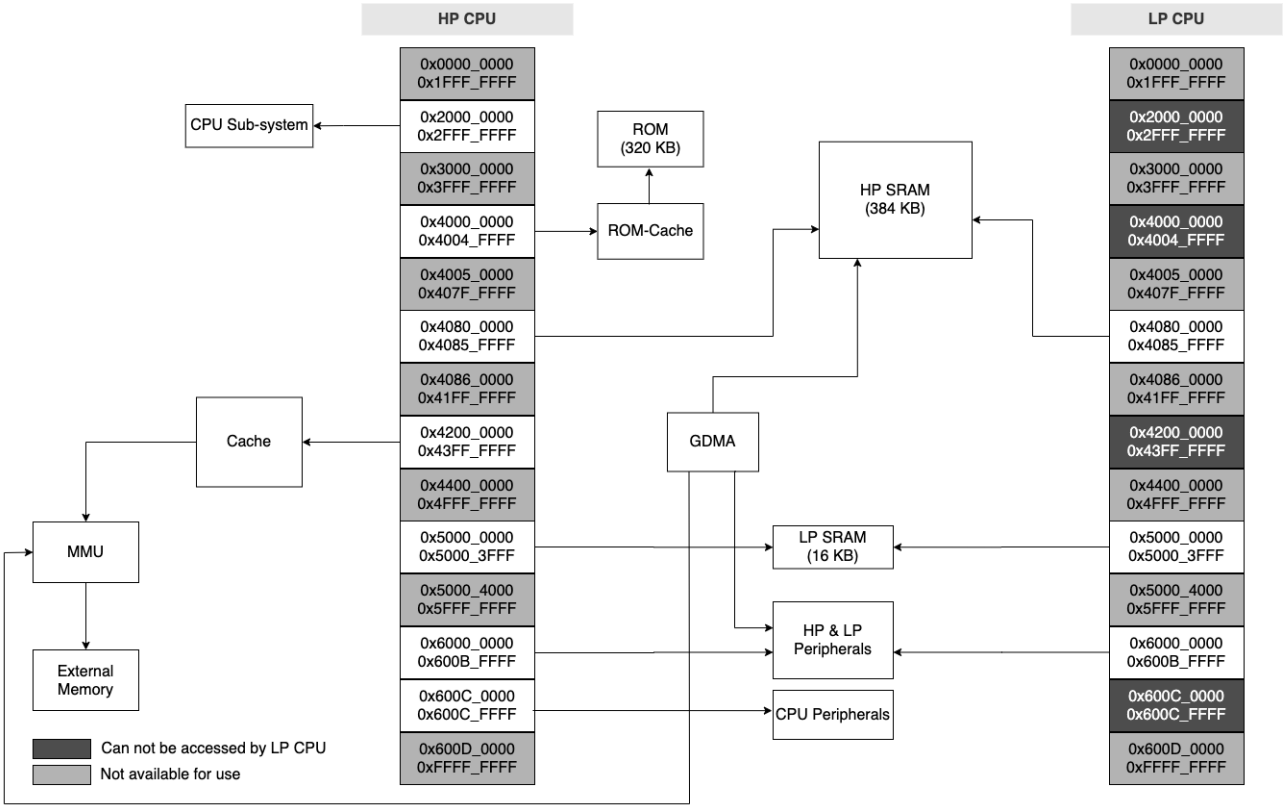


图 6.3-1. 系统结构与地址映射结构

说明：

- 地址空间中可用的地址范围可能大于实际可用的内存。
- 关于 CPU Sub-system，详情请参考章节 2 高性能处理器。
- 部分地址空间只可被 HP CPU 访问，不能被 LP CPU 访问，已在图 6.3-1 中注明。后文描述中将不再对这部分地址空间进行区分。

表 6.3-1 描述了数据总线与指令总线中的各段地址所能访问的目标。

表 6.3-1. 地址映射

总线类型	边界地址		容量	目标
	低位地址	高位地址		
	0x0000_0000	0x3FFF_FFFF		保留
数据总线/指令总线	0x4000_0000	0x4004_FFFF	320 KB	ROM*
	0x4005_0000	0x407F_FFFF		保留
数据总线/指令总线	0x4080_0000	0x4085_FFFF	384 KB	HP SRAM*
	0x4086_0000	0x41FF_FFFF		保留
数据总线/指令总线	0x4200_0000	0x43FF_FFFF	32 MB	外部存储器
	0x4400_0000	0x4FFF_FFFF		保留
数据总线/指令总线	0x5000_0000	0x5000_3FFF	16 KB	LP SRAM*
	0x5000_4000	0x5FFF_FFFF		保留

见下页

表 6.3-1 – 接上页

总线类型	边界地址		容量	目标
	低位地址	高位地址		
数据总线/指令总线	0x6000_0000	0x600C_FFFF	832 KB	外设
	0x600D_0000	0xFFFF_FFFF		保留

* 所有内部存储器均接受权限管理。只有获取到访问内部存储器的访问权限，才可以执行正常的访问操作，HP CPU 和 LP CPU 访问内部存储器时才可以被响应。关于权限管理的更多信息，请参考章节 18 权限控制 (PMS)。

6.3.2 内部存储器

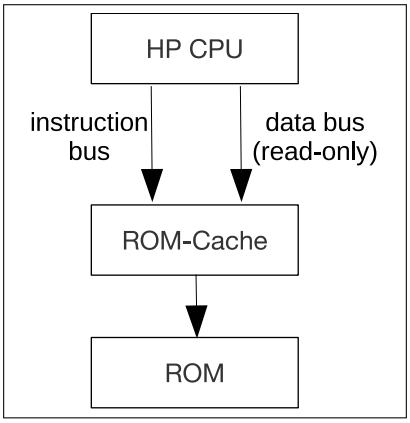
ESP32-C5 的内部存储器包含如下三种类型：

- ROM (320 KB)：只读存储器，不可编程，仅可由 HP CPU 访问。用于存放系统底层软件的 ROM 代码和只读数据。
- HP SRAM (384 KB)：易失性存储器，可快速响应 HP CPU 和 LP CPU 的访问请求。HP CPU 的访问延迟通常是一个 HP CPU 时钟周期。
- LP SRAM (16 KB)：以静态 RAM (SRAM) 方式实现，因此也是易失性存储器。但在 Deep-sleep 模式下，存放在 LP SRAM 中的数据不会丢失。LP SRAM 可被 HP CPU 和 LP CPU 访问，通常用来存放一些在睡眠模式下仍需保持的程序指令和数据。

6.3.2.1 ROM

ROM 的容量为 320 KB，只读。如表 6.3-1 所示，HP CPU 可以通过指令总线或数据总线地址段 0x4000_0000 ~ 0x4004_FFFF 直接或借助 ROM-Cache 访问这部分存储器（见图 6.3-2）。

ESP32-C5 ROM-Cache 为 2 路组相联，容量为 4 KB，块大小为 64 字节。指令总线和数据总线可以同时访问该 ROM-Cache，ROM-Cache 经过仲裁对其中一个做出响应。当 ROM-Cache 缺失时，ROM-Cache 控制器会向 ROM 发起请求。



注：图中 → 表示控制流方向

图 6.3-2. ROM-Cache 系统结构

6.3.2.2 HP SRAM

HP SRAM 的容量为 384 KB，可读可写。如表 6.3-1 所示，HP CPU 和 LP CPU 可以通过数据总线或指令总线访问这部分存储器。

6.3.2.3 LP SRAM

LP SRAM 容量为 16 KB，为可读可写的 SRAM。如表 6.3-1 所示，HP CPU 和 LP CPU 可以通过数据总线和指令总线的共用地址段 0x5000_0000 ~ 0x5000_3FFF 访问这部分存储器。

LP SRAM 存在两种访问模式：

- 高速访问模式：以 HP CPU 时钟频率进行访问，此时 HP CPU 能够无延迟访问 LP SRAM，但是 LP CPU 访问 LP SRAM 的延迟在十几个到几十个 LP CPU 周期不等。
- 低速访问模式：以 LP CPU 时钟频率进行访问，此时 LP CPU 能够无延迟访问 LP SRAM，但是 HP CPU 访问 LP SRAM 的延迟在十几个到几十个 HP CPU 周期不等。

用户可根据应用场景进行模式的切换，比如：

- 如果 LP CPU 未启动，可以切换到高速访问模式，从而提高 HP CPU 访问速度。
- 如果 LP CPU 正在 LP SRAM 执行代码，可以切换到低速访问模式。
- 如果 HP CPU 处于休眠时，必须切换到低速模式。

模式切换配置流程

- 配置 LP_AON_FAST_MEM_MUX_SEL 选择模式：
  - 0：低速访问模式
  - 1：高速访问模式
- 置位 LP_AON_FAST_MEM_MUX_SEL_UPDATE 开始切换模式
- 查询 LP_AON_FAST_MEM_MUX_SEL_STATUS 确认是否切换完成：
  - 0：表示切换完成
  - 1：表示切换未完成

6.3.3 外部存储器

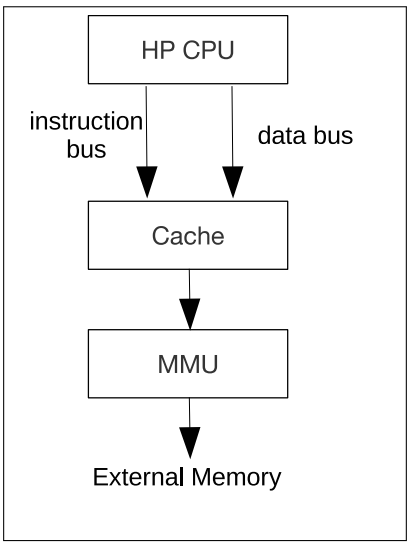
ESP32-C5 支持以 SPI、Dual SPI、Quad SPI、QPI 等接口形式连接片外 flash 和片外 RAM。ESP32-C5 还支持基于 XTS-AES 算法的硬件手动加密和自动解密功能，从而保护开发者片外 flash 和片外 RAM 中的程序和数据。

6.3.3.1 外部存储器地址映射

HP CPU 借助高速缓存 (cache) 来访问外部存储器，GDMA 也可以访问外部存储器。内存管理单元 (Memory Management Unit, MMU) 把 HP CPU 和 GDMA 的地址 (0x4200_0000 ~ 0x43FF_FFFF) 映射为访问片外存储器的实地址。经过地址映射，ESP32-C5 最大支持 32 MB 的片外 flash 和 32 MB 的片外 RAM。
请注意，指令总线地址空间 (32 MB) 和数据总线地址空间 (32 MB) 是共用的。

6.3.3.2 Cache

如图 6.3-3 所示，ESP32-C5 采用一个共享 cache，为 4 路组相联，容量为 32 KB，块大小为 32 字节。指令总线和数据总线可以同时访问该 cache，cache 经过仲裁对其中一个做出响应。当 cache 缺失时，cache 控制器会向外部存储器发起请求。



注：图中 → 表示控制流方向

图 6.3-3. Cache 系统结构

6.3.3.3 Cache 操作

ESP32-C5 cache 支持如下几种操作：

1. **回写 (Write-back)**
   - 清除标记存储器中的脏位，并将新数据更新到外部存储器。
   - 回写操作完成后，新数据将被更新到外部存储器中。
   - 可以选择是否对 cache 中的数据作失效处理。
   - 如果没有选择失效处理，那么 HP CPU 可以直接从 cache 读取/写入数据，而不会发生 cache 缺失。
2. **清除 (Clean)**
   - 清除标记存储器中的脏位，但不把新数据更新到外部存储器。
   - 清理操作完成后，外部存储器中存储的仍然是旧数据，而 cache 会保留新的数据，但 cache 并不知道其存储的为新数据。
   - 然后 HP CPU 可以直接从 cache 读取/写入数据，而不会发生 cache 缺失。
3. **失效 (Invalidate)**
   - 清除标记存储器中的有效位，用于删除 cache 中的有效数据。
   - 该操作完成后，删除的数据将仅存于外部存储器中。
   - 如果 HP CPU 接着去访问该数据，那么需要访问外部存储器。
   - 该操作包括两种类型：
     - 手动失效 (Manual-Invalidate)：仅对 cache 中落入指定区域的地址对应的数据做失效处理。
     - 整体失效 (Invalidate-All)：会对 cache 中的所有数据做失效处理。
4. **预取 (Preload)**
   - 将指令和数据提前加载到 cache 中。

- 预取操作的最小单位为 1 个 cache 块。
- 该操作包括两种类型：
  - 手动预取 (Manual-Preload)：硬件按软件指定的虚地址预取一段连续的数据。
  - 自动预取 (Auto-Preload)：硬件根据当前命中/缺失（取决于配置）的地址，自动预取一段连续的数据。

5. **锁定/解锁 (Lock/Unlock)**
   - 保护 cache 中的数据不被替换掉。
   - 该锁定操作包括两种类型：
     - 预锁定：在填充缺失数据到 cache 时，如果该数据落在指定区域，cache 则将该数据锁定，未落入指定区域的数据不会被锁定。
     - 手动锁定：检查 cache 中的数据，并将落在指定区域的数据锁定，未落入指定区域的数据不会被锁定。
   - 当缺失发生时，cache 会优先替换掉未被锁定的那一路的数据，因此锁定区域的数据会一直保存在 cache 中。
   - 但当所有路都被锁定时，cache 将进行正常替换，就像所有路都没有被锁定一样。
   - 解锁是锁定的逆操作，但解锁只有手动解锁。
   - 请注意，手动失效操作只对未被锁定的数据起作用。如需对已锁定的数据执行手动失效操作，请先解锁这些数据。

6.3.4 GDMA 地址空间

ESP32-C5 中的 GDMA 外设包含三个 TX 通道和三个 RX 通道，可提供 DMA 服务，包括：

- 内部存储器中不同位置的数据搬运
- 模块/外设和内部存储器之间的数据搬运
- 外部存储器中不同位置的数据搬运
- 模块/外设和外部存储器之间的数据搬运
- 内部存储器和外部存储器之间的数据搬运

GDMA 可以通过与数据总线完全相同的地址读写 HP SRAM 和片外 RAM，即 GDMA 通过地址 0x4080_0000 ~ 0x4085_FFFF 访问 HP SRAM，通过地址 0x4200_0000 ~ 0x43FF_FFFF 访问片外 RAM。ESP32-C5 中共有七个外设/模块可以和 GDMA 联合工作。如图 6.3-4 所示，其中的七根竖线依次对应这七个具有 GDMA 功能的外设/模块，横线表示 GDMA 的某一通道（可为任意通道），竖线与横线的交点表示对应外设/模块可以访问 GDMA 的某一通道。

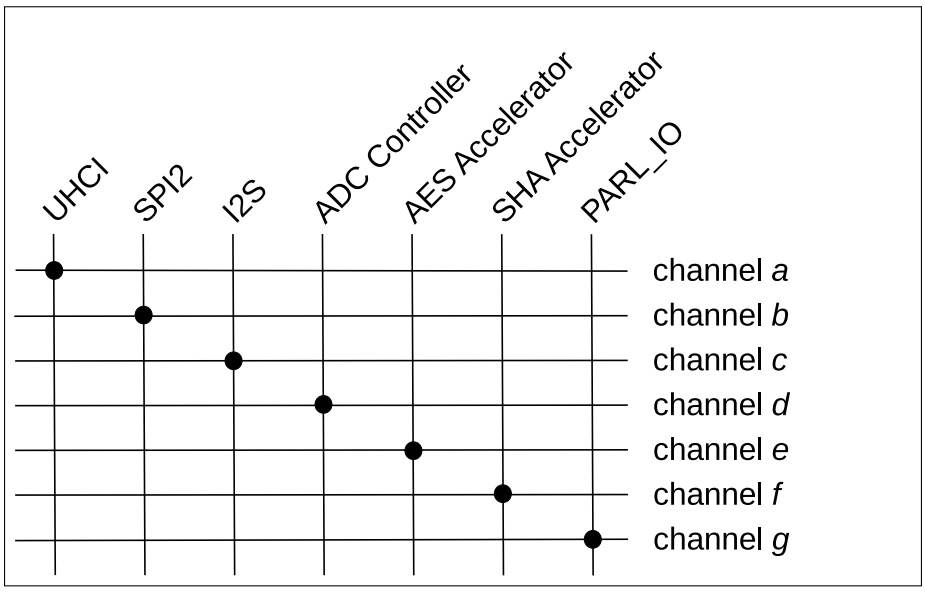


图 6.3-4. 具有 GDMA 功能的外设/模块

具有 GDMA 功能的模块/外设通过 GDMA 可以访问任何 GDMA 可以访问到的存储器。更多关于 GDMA 的信息，请参考章节 5 通用 *DMA 控制器 (GDMA)*。

说明：
当使用 GDMA 访问任何存储器时，都需要获取对应的访问权限，否则访问将会失败。关于权限管理的更多信息，请参考章节 18 权限控制 *(PMS)*。

6.3.5 模块/外设地址空间映射

表 6.3-2 详细列出了模块/外设地址空间的各段地址与其能访问到的模块/外设的映射关系。其中，“边界地址”（包括低位地址和高位地址）栏中的两列数值共同决定了对应模块/外设的地址空间。

表 6.3-2. 模块/外设地址空间映射表

目标	边界地址		容量 (KB)
	低位地址	高位地址	
UART 控制器 0 (UART0)	0x6000_0000	0x6000_0FFF	4
UART 控制器 1 (UART1)	0x6000_1000	0x6000_1FFF	4
片外存储器加密与解密 (XTS_AES) [1]	0x6000_2000	0x6000_2FFF	4
SPI 控制器 0 (SPI0) [1]	0x6000_2000	0x6000_2FFF	4
SPI 控制器 1 (SPI1)	0x6000_3000	0x6000_3FFF	4
I2C 控制器 (I2C)	0x6000_4000	0x6000_4FFF	4
UHCI 控制器 (UHCI)	0x6000_5000	0x6000_5FFF	4
红外遥控 (RMT)	0x6000_6000	0x6000_6FFF	4
LED PWM 控制器 (LEDC)	0x6000_7000	0x6000_7FFF	4
定时器组 0 (TIMG0)	0x6000_8000	0x6000_8FFF	4
定时器组 1 (TIMG1)	0x6000_9000	0x6000_9FFF	4

见下页

表 6.3-2 – 接上页

目标	边界地址		容量 (KB)
	低位地址	高位地址	
系统定时器 (SYSTIMER)	0x6000_A000	0x6000_AFFF	4
双线汽车接口 0 (TWAI0)	0x6000_B000	0x6000_BFFF	4
I2S 控制器 (I2S)	0x6000_C000	0x6000_CFFF	4
双线汽车接口 1 (TWAI1)	0x6000_D000	0x6000_DFFF	4
逐次逼近型模数转换器 (SAR ADC)	0x6000_E000	0x6000_EFFF	4
温度传感器	0x6000_E000	0x6000_EFFF	4
USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)	0x6000_F000	0x6000_FFFF	4
中断矩阵 (INTMTX)	0x6001_0000	0x6001_0FFF	4
保留	0x6001_1000	0x6001_1FFF	
脉冲计数控制器 (PCNT)	0x6001_2000	0x6001_2FFF	4
事件任务矩阵 (SOC_ETM)	0x6001_3000	0x6001_3FFF	4
电机控制器 (MCPWM)	0x6001_4000	0x6001_4FFF	4
并行 IO 控制器 (PARL_IO)	0x6001_5000	0x6001_5FFF	4
SDIO HINF[2]	0x6001_6000	0x6001_6FFF	4
SDIO SLC[2]	0x6001_7000	0x6001_7FFF	4
SDIO SLCHOST[2]	0x6001_8000	0x6001_8FFF	4
保留	0x6001_6000	0x6001_9FFF	
存储器访问监控 2 (PSARM_MEM_MONITOR) [2]	0x6001_A000	0x6001_AFFF	4
保留	0x6001_B000	0x6007_FFFF	
通用 DMA 控制器 (GDMA)	0x6008_0000	0x6008_0FFF	4
通用 SPI2 控制器 (GP-SPI2)	0x6008_1000	0x6008_1FFF	4
比特传输控制器 (BITSCRAMBLER)	0x6008_2000	0x6008_2FFF	4
保留	0x6008_3000	0x6008_6FFF	
密钥管理器	0x6008_7000	0x6008_7FFF	4
AES 加速器 (AES)	0x6008_8000	0x6008_8FFF	4
SHA 加速器 (SHA)	0x6008_9000	0x6008_9FFF	4
RSA 加速器 (RSA)	0x6008_A000	0x6008_AFFF	4
ECC 加速器 (ECC)	0x6008_B000	0x6008_BFFF	4
RSA 数字签名外设 (RSA_DS)	0x6008_C000	0x6008_CFFF	4
HMAC 加速器 (HMAC)	0x6008_D000	0x6008_DFFF	4
ECDSA 数字签名外设 (ECDSA_DS)	0x6008_E000	0x6008_EFFF	4
保留	0x6008_F000	0x6008_FFFF	
IO MUX	0x6009_0000	0x6009_0FFF	4
GPIO 交换矩阵	0x6009_1000	0x6009_1FFF	4
存储器访问监控 1 (TCM_MEM_MONITOR) [2]	0x6009_2000	0x6009_2FFF	4
保留	0x6009_3000	0x6009_4FFF	
高性能系统寄存器 (HP_SYSREG)	0x6009_5000	0x6009_5FFF	4
电源/时钟/复位寄存器 (PCR)	0x6009_6000	0x6009_6FFF	4
保留	0x6009_7000	0x6009_7FFF	
可信执行环境寄存器 (TEE) [2]	0x6009_8000	0x6009_8FFF	4

见下页

表 6.3-2 – 接上页

目标	边界地址		容量 (KB)
	低位地址	高位地址	
高性能访问权限管理 (HP_APM)	0x6009_9000	0x6009_97FF	2
低功耗访问权限管理 (LP_APMO)	0x6009_9800	0x6009_9FFF	2
CPU 访问权限管理 (CPU_APM)	0x6009_A000	0x6009_AFFF	4
保留	0x6009_B000	0x600A_FFFF	
电源管理单元 (PMU)	0x600B_0000	0x600B_03FF	1
低功耗时钟复位寄存器 (LP_CLKRST)	0x600B_0400	0x600B_07FF	1
保留	0x600B_0800	0x600B_0BFF	
低功耗定时器 (LP_TIMER)	0x600B_0C00	0x600B_0FFF	1
低功耗常开寄存器 (LP_AON)	0x600B_1000	0x600B_13FF	1
低功耗 UART (LP_UART)	0x600B_1400	0x600B_17FF	1
低功耗 I2C (LP_I2C)	0x600B_1800	0x600B_1BFF	1
低功耗看门狗定时器 (LP_WDT)	0x600B_1C00	0x600B_1FFF	1
保留	0x600B_2000	0x600B_23FF	
I2C 模拟主机 (I2C_ANA_MST)	0x600B_2400	0x600B_27FF	1
低功耗外设 (LPPERI)	0x600B_2800	0x600B_2BFF	1
低功耗模拟外设 (LP_ANA_PERI)	0x600B_2C00	0x600B_2FFF	1
HUK 生成器	0x600B_3000	0x600B_33FF	1
低功耗可信执行环境寄存器 (LP_TEE) [2]	0x600B_3400	0x600B_37FF	1
低功耗访问权限管理 (LP_APM) [2]	0x600B_3800	0x600B_3BFF	1
保留	0x600B_3C00	0x600B_3FFF	
低功耗 IO MUX (LP_IO_MUX)	0x600B_4000	0x600B_43FF	1
低功耗 GPIO 交换矩阵 (LP_GPIO)	0x600B_4400	0x600B_47FF	1
eFuse 控制器 (eFuse)	0x600B_4800	0x600B_4FFF	2
保留	0x600B_5000	0x600B_FFFF	
RISC-V Trace 编码器 (TRACE)	0x600C_0000	0x600C_0FFF	4
保留	0x600C_1000	0x600C_1FFF	
总线访问监控 (BUS_MONITOR) [2]	0x600C_2000	0x600C_2FFF	4
保留	0x600C_3000	0x600C_4FFF	
中断优先级寄存器 (INTPRI)	0x600C_5000	0x600C_5FFF	4
保留	0x600C_6000	0x600C_7FFF	
Cache	0x600C_8000	0x600C_8FFF	4
保留	0x600C_9000	0x600C_FFFF	

[1] 片外存储器加密与解密 (XTS_AES) 模块的寄存器空间与 SPI 控制器 0 (SPI0) 模块的寄存器空间共用地址 0x6000_2000 ~ 0x6000_2FFF。

[2] 该模块/外设的地址空间不连续。

说明：

如图 6.3-1 所示：

- HP CPU 可访问表 6.3-2 列出的所有外设。
- LP CPU 不能访问表 6.3-2 中的 RISC-V Trace 编码器 (TRACE)、辅助调试 (ASSIST_DEBUG)、中断优先级寄存器 (INTPRI) 和 cache，其余外设均可访问。

第 7 章

eFuse 控制器 (EFUSE)

7.1 概述

ESP32-C5 系统中有一块 4096 位的 eFuse 存储器用于存储参数内容和用户数据，参数内容包括一些硬件模块的控制参数、系统数据参数以及加解密模块使用的密钥等。eFuse 存储器的各个位一旦被烧写为 1，则不能再恢复为 0。eFuse 控制器按照用户配置完成对 eFuse 存储器中各个位的烧写。从芯片外部，eFuse 数据只能通过 eFuse 控制器读取。对于某些数据，例如存储在 eFuse 中供硬件加密模块（例如数字签名、HMAC 等）在内部使用的某些密钥，如果未启用读保护，则可以从芯片外部读取该数据；如果启用了读保护，则无法从芯片外部读取该数据。

7.2 主要特性

- 4096 位一次性可编程存储（最多有 1792 个保留位供用户使用，具体取决于 EFUSE_KEY_PURPOSE_*n* 是否都配置成 0，见表 7.3-2）
- 烧写保护可配置
- 读取保护可配置
- 使用多种硬件编码方式保护参数内容

7.3 功能描述

7.3.1 结构

eFuse 控制器和 eFuse 存储器共同组成了 eFuse 系统。其内部的数据通路如图 7.3-1 所示。

用户可以通过将待烧写的数据填入烧写寄存器中并执行烧写指令，通过 eFuse 控制器完成对 eFuse 存储器内部各个位的烧写。详细烧写步骤可以参考章节 7.3.2。

用户不能直接读取 eFuse 存储器中烧写的信息内容，因此需要通过 eFuse 控制器将烧写的数据信息读取到对应的地址段的读寄存器内。如果数据在读取过程中出现了和 eFuse 存储器不一致的错误，eFuse 控制器还可以通过硬件编码机制对数据进行自动校正（详见章节 7.3.1.3 ），并将错误信息自动更新至错误报告寄存器中。详细的读取参数步骤可以参考章节 7.3.3。

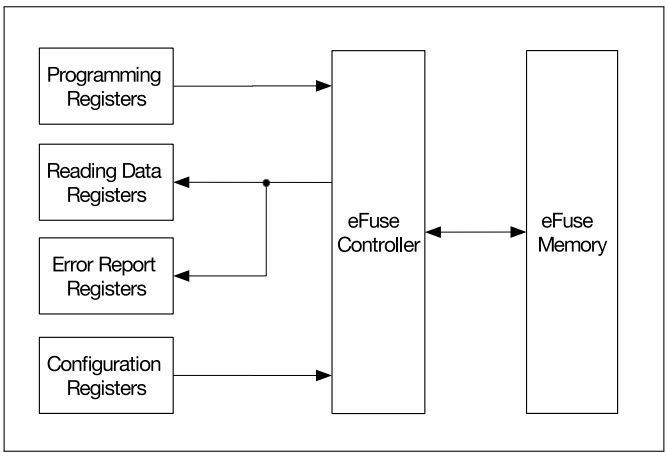


图 7.3-1. eFuse 系统数据通路

eFuse 存储器从结构上分成 11 个块 (BLOCK0 ~ BLOCK10)。

BLOCK0 存储了大部分供软硬件使用的控制参数。

表 7.3-1 列出了用户可访问（可读并可用）的所有 BLOCK0 中的参数名称、位宽、是否可供硬件使用、烧写保护位、以及简要的功能描述，若需了解更多有关该参数的描述，请点击表格中对应参数的链接。

在这些参数中，EFUSE_WR_DIS 用于控制 BLOCK0 ~ BLOCK10 中其他参数的烧写保护状态，EFUSE_RD_DIS 用于控制 BLOCK4 ~ BLOCK10 的读取保护状态。更多关于这两个参数的信息请见章节 7.3.1.1、7.3.1.2。

表 7.3-1. BLOCK0 参数

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_WR_DIS	32	Y	N/A	表示是否禁用 eFuse 存储器各个位的烧写。
EFUSE_RD_DIS	7	Y	0	表示是否禁用 eFuse 各个 block (BLOCK4 ~ BLOCK10) 的读取。
EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_-VERSION_HI	1	N	N/A	表示一级 (ROM) 引导程序使用的二级引导程序的防回滚安全版本（该字段的高位部分）。
EFUSE_DIS_ICACHE	1	Y	2	表示 cache 是否禁用。
EFUSE_DIS_USB_JTAG	1	Y	2	表示是否禁用 USB Serial/JTAG 模块中 USB 转 JTAG 功能。
EFUSE_BOOTLOADER_ANTI_ROLLBACK_EN	1	N	N/A	表示是否启用了对二级引导程序的防回滚检查。
EFUSE_DIS_FORCE_DOWNLOAD	1	Y	2	表示是否禁用强制芯片进入 Download 模式的功能。
EFUSE_SPI_DOWNLOAD_MSPI_DIS	1	Y	2	表示在 boot_mode_download 过程中 MSPI 控制器是否禁用。
EFUSE_DIS_TWAI	1	Y	2	表示是否禁用 TWAI® 功能。
EFUSE_JTAG_SEL_ENABLE	1	Y	2	表示EFUSE_DIS_PAD_JTAG 和 EFUSE_DIS_USB_JTAG 均配置为 0 时，是否使能通过 strapping 管脚值选择 JTAG 信号源。详情请参考章节 10 芯片 *Boot* 控制。
EFUSE_SOFT_DIS_JTAG	3	Y	31	表示是否软禁用 PAD JTAG。可通过 HMAC 重新启动。
EFUSE_DIS_PAD_JTAG	1	Y	2	表示是否硬禁用 PAD JTAG（永久）。
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	1	Y	2	表示是否使能 flash 加密（SPI 启动模式除外）。
EFUSE_USB_EXCHG_PINS	1	Y	30	表示是否交换 USB D+ 和 D- 管脚。
EFUSE_VDD_SPI_AS_GPIO	1	Y	30	表示 VDD SPI 管脚是否用作 GPIO。
EFUSE_WDT_DELAY_SEL	2	Y	3	表示 RTC 看门狗 STGO 超时阈值的档位。
EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_-VERSION_LO	3	N	N/A	表示一级 (ROM) 引导加载程序使用的二级引导加载程序的防回滚安全版本（字段低位）。
EFUSE_KM_DISABLE_DEPLOY_MODE	4	Y	1	表示是否禁用密钥管理器的新密钥部署。
EFUSE_KM_RND_SWITCH_CYCLE	2	Y	1	表示密钥管理器切换随机数的周期。
EFUSE_KM_DEPLOY_ONLY_ONCE	4	Y	1	表示对应的密钥是否只能部署一次。

见下页

GoBack

表 7.3-1 – 接上页

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_FORCE_USE_KEY_MANAGER_KEY	4	Y	1	表示相应的密钥是否必须来自密钥管理器。
EFUSE_FORCE_DISABLE_SW_INIT_KEY	1	Y	1	表示是否禁用由软件编写的初始化密钥，而强制使用 efuse_init_key。
EFUSE_BOOTLOADER_ANTI_ROLLBACK_UPDATE_IN_ROM	1	N	N/A	表示是否从一级（ROM）引导加载程序更新防回滚 SECURE_VERSION。
EFUSE_SPI_BOOT_CRYPT_CNT	3	Y	4	表示是否使能 SPI boot 加解密。
EFUSE_SECURE_BOOT_KEY_REVOKE0	1	N	5	表示是否撤销安全启动秘钥摘要 0。
EFUSE_SECURE_BOOT_KEY_REVOKE1	1	N	6	表示是否撤销安全启动秘钥摘要 1。
EFUSE_SECURE_BOOT_KEY_REVOKE2	1	N	7	表示是否撤销第安全启动秘钥摘要 2。
EFUSE_KEY_PURPOSE_0	4	Y	8	表示 Key0 用途，更多信息请见表 7.3-2。
EFUSE_KEY_PURPOSE_1	4	Y	9	表示 Key1 用途，更多信息请见表 7.3-2。
EFUSE_KEY_PURPOSE_2	4	Y	10	表示 Key2 用途，更多信息请见表 7.3-2。
EFUSE_KEY_PURPOSE_3	4	Y	11	表示 Key3 用途，更多信息请见表 7.3-2。
EFUSE_KEY_PURPOSE_4	4	Y	12	表示 Key4 用途，更多信息请见表 7.3-2。
EFUSE_KEY_PURPOSE_5	4	Y	13	表示 Key5 用途，更多信息请见表 7.3-2。
EFUSE_SEC_DPA_LEVEL	2	Y	14	表示防差分功耗分析 (differential power analysis, DPA) 攻击的安全级别
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_HI	3	N	N/A	表示主引导程序启动失败时，一级（ROM）引导加载程序使用的恢复引导程序的起始 flash 扇区（flash 扇区大小为 0x1000）。值为 0 或 0xFFF 表示该功能禁用（字段高位）
EFUSE_SECURE_BOOT_EN	1	N	15	表示是否使能安全启动。
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	1	N	16	表示是否使能安全启动的激进撤销。
EFUSE_KM_XTS_KEY_LENGTH_256	1	N	1	表示 flash 加密使用的密钥。
EFUSE_FLASH_TPUW	4	N	18	表示 flash 上电等待时间。单位：ms。当值小于 15 时，等待时间为配置值。当值大于等于 15 时，等待时间固定为 30 ms。

见下页

GoBack

表 7.3-1 – 接上页

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_DIS_DOWNLOAD_MODE	1	N	18	表示是否禁用所有 Download 模式。
EFUSE_DIS_DIRECT_BOOT	1	N	18	表示是否禁用 direct boot 模式。
EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT	1	N	18	表示是否禁用 ROM boot 过程中的 USB-Serial-JTAG 打印。
EFUSE_LOCK_KM_KEY	1	N	1	表示密钥管理器中的密钥在部署后是否被锁定。
EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE	1	N	18	表示是否禁用 USB-Serial-JTAG 下载功能。
EFUSE_ENABLE_SECURITY_DOWNLOAD	1	N	18	表示是否使能安全下载。仅支持下载到 flash，不支持读写 RAM 或寄存器，即不支持 stub 下载。
EFUSE_UART_PRINT_CONTROL	2	N	18	表示 UART 打印类型。
EFUSE_FORCE_SEND_RESUME	1	N	18	表示是否强制 ROM 代码在 SPI 启动过程中发送恢复指令。
EFUSE_SECURE_VERSION	16	N	18	表示安全版本，用于 ESP-IDF 防回滚功能。
EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE	1	N	18	表示安全启动使能时是否禁用 FAST VERIFY ON WAKE。
EFUSE_HYS_EN_PAD	1	Y	2	表示是否使能 PAD0 ~ 27 的迟滞功能。
EFUSE_XTS_DPA_PSEUDO_LEVEL	2	Y	14	表示 XTS-AES 抗 DPA 攻击的伪轮级别。
EFUSE_XTS_DPA_CLK_ENABLE	1	Y	14	表示 XTS-AES 抗 DPA 攻击时钟是否使能。
EFUSE_ECDSA_P384_ENABLE	1	N	14	表示芯片是否支持 ECDSA_DS P384。
EFUSE_HUK_GEN_STATE	9	Y	19	表示 HUK 生成模式是否有效。
EFUSE_ECC_FORCE_CONST_TIME	1	Y	14	表示是否强制 ECC 在点乘运算中使用常数时间模式。
EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_LO	9	N	N/A	表示当主引导程序失败时，ROM 引导程序使用的恢复引导程序的起始 Flash 扇区（Flash 扇区大小为 0x1000）。设置为 0 或 0xFFF 表示该功能被禁用（字段低位）。

GoBack

表 7.3-2 为密钥用途各个数值对应的含义。通过配置参数 EFUSE_KEY_PURPOSE_*n* 来声明 KEY*n* 用途 (*n*: 0 ~ 5)。

表 7.3-2. 密钥用途数值对应的含义

密钥用途数值	含义
0	指定为用户使用
1	指定为 ECDSA_KEY 使用
2	指定为 XTS_AES_256_KEY_1 使用
3	指定为 XTS_AES_256_KEY_2 使用
4	指定为 XTS_AES_128_KEY 使用（用于 flash/SRAM 加解密）
5	指定为 HMAC Downstream（下行）模式（JTAG 和数字签名）使用
6	指定为 HMAC Downstream 模式下的 JTAG 使用
7	指定为 HMAC Downstream 模式下的数字签名使用
8	指定为 HMAC Upstream（上行）模式使用
9	指定为 SECURE_BOOT_DIGEST0 使用（secure boot 密钥摘要）
10	指定为 SECURE_BOOT_DIGEST1 使用（secure boot 密钥摘要）
11	指定为 SECURE_BOOT_DIGEST2 使用（secure boot 密钥摘要）

表 7.3-3 列出了 BLOCK1 ~ BLOCK10 中存储的参数的信息。

表 7.3-3. BLOCK1-10 参数

块	参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	EFUSE_RD_DIS 读取保护位	描述
BLOCK1	EFUSE_MAC	48	N	20	N/A	MAC 地址
	EFUSE_MAC_EXT	16	N	20	N/A	MAC 拓展地址
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	系统数据
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	系统数据
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	用户数据
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	KEY0 或用户数据
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	KEY1 或用户数据
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	KEY2 或用户数据
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	KEY3 或用户数据
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	KEY4 或用户数据
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	KEY5 或用户数据
BLOCK10	EFUSE_SYS_DATA_PART2	256	N	29	6	系统数据

其中，BLOCK4 ~ 9 可用于存储 KEY0 ~ 5，表示 eFuse 中至多可以烧写 6 个 256 位的密钥。每烧写一个密钥，还需要烧写该密钥用途的数值（见表 7.3-2）。例如，用户将用于 HMAC Downstream 模式下的 JTAG 功能的密钥烧写到 KEY3（即 BLOCK7），还需要将密钥用途的数值 6 烧写到 EFUSE_KEY_PURPOSE_3。

BLOCK1 ~ BLOCK10 均采用 RS(里德-所罗门码) 编码方式，因此参数烧写受到一定的限制，具体请参考章节 7.3.1.3 和章节 7.3.2。

7.3.1.1 EFUSE_WR_DIS

参数 EFUSE_WR_DIS 决定了 eFuse 存储器中所有的参数是否处于烧写保护状态。烧写完 EFUSE_WR_DIS 参数后，需要更新 eFuse 控制器的读寄存器以保证烧写保护状态生效。

表 7.3-1 以及表 7.3-3 中的“EFUSE_WR_DIS 烧写保护位”列描述了各参数的烧写保护状态具体由 EFUSE_WR_DIS 的哪个位决定。

当某个参数对应的烧写保护位为 0 时，表示此参数未处于烧写保护状态，可以烧写该参数，但已经被烧写的参数不能被重复烧写。

当某个参数对应的烧写保护位为 1 时，表示此参数处于烧写保护状态，此参数的每一个位都无法被更改，未被烧写的位永远为 0，已经被烧写的位永远为 1。因此如果某个参数已经处于烧写保护状态了，则会一直处在该状态，无法再更改。

7.3.1.2 EFUSE_RD_DIS

所有参数中，只有 BLOCK4 ~ BLOCK10 的参数可以被配置为用户读取保护状态，即表 7.3-3 中“EFUSE_RD_DIS 读取保护”列非“N/A”的参数。烧写完 EFUSE_RD_DIS 参数后，需要更新 eFuse 控制器的读寄存器以保证读取保护状态生效。

参数 EFUSE_RD_DIS 中的某个位为 0，表示此位管理的参数未处于用户读取保护状态；某个位为 1，表示此位管理的参数处于用户读取保护状态。

除 BLOCK4 ~ BLOCK10 之外，其他参数不受读取保护状态的约束，均可被用户读取。

BLOCK4 ~ BLOCK10 即使被配置处于读取保护状态，仍然可以通过设置 EFUSE_KEY_PURPOSE_*n* 供硬件加密模块在内部使用。

7.3.1.3 数据存储方式

eFuse 使用硬件编码机制保护数据，对用户不可见。

BLOCK0 使用 4 备份方式存储参数，即 BLOCK0 中的所有参数（除了 EFUSE_WR_DIS ）均在 eFuse 存储器中存储了 4 份。4 备份机制对用户不可见。

BLOCK0 中 EFUSE_WR_DIS 为 32 比特，其余参数为 152 比特，因此 BLOCK0 在 eFuse 存储器中共占据了 32 + 152 * 4 = 640 比特的存储空间。

BLOCK1 ~ BLOCK10 使用 RS (44, 32) 编码方式，最多支持自动校正 6 个字节。本文 RS (44, 32) 使用的本源多项式为 $p(x) = x^8 + x^4 + x^3 + x^2 + 1$。

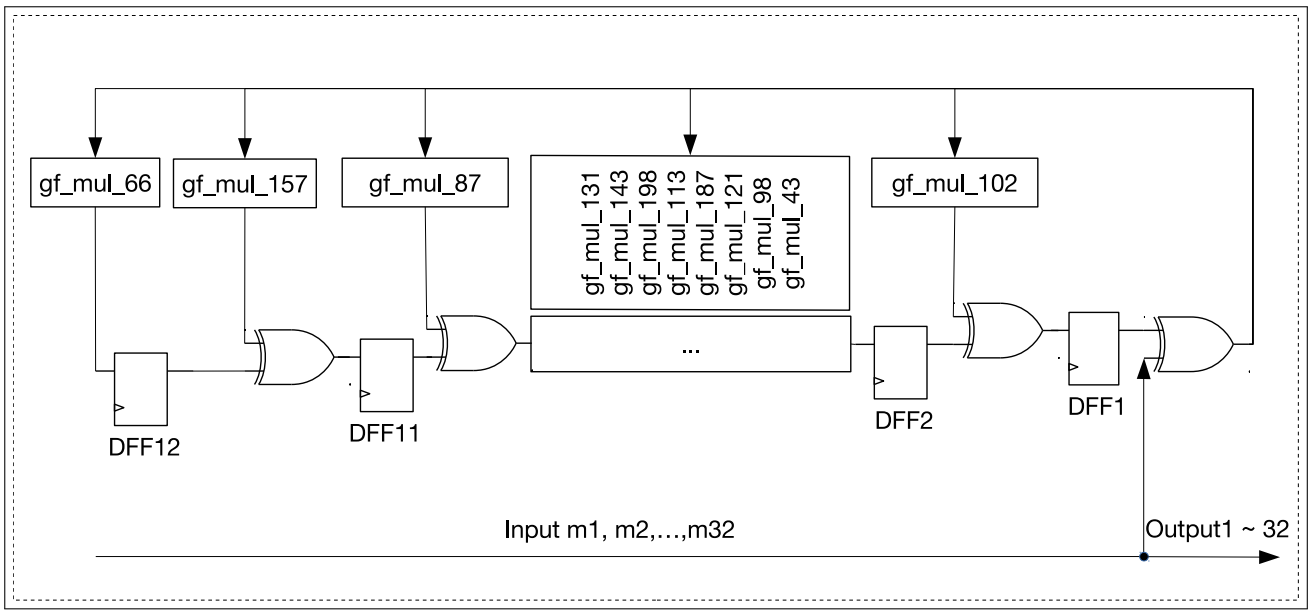


图 7.3-2. 移位寄存器电路图（前 32 字节）

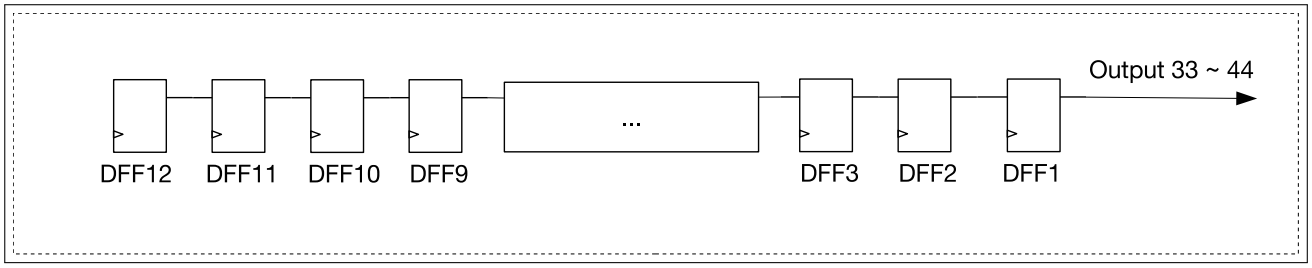


图 7.3-3. 移位寄存器电路图（后 12 字节）

如图 7.3-2 和 7.3-3 所示，移位寄存器电路对 32 字节参数进行 RS (44, 32) 编码处理，将 32 字节数据处理为 44 字节，其中：

- 字节 [0:31] 为数据本身
- 字节 [32:43] 为储存在 8 位触发器 DFF1, DFF2, ..., DFF12 中的奇偶校验字节（gf_mul_n 为 $GF(2^8)$ 域中某一字节数据与元素 α^n 相乘的结果，n 为整数）

然后，硬件将这 44 字节数据一起烧入 eFuse。eFuse 控制器会在读 eFuse 的过程中自动完成解码和自动校正。

由于 RS 校验码是在整个 32 字节的 eFuse 块上生成的，因此每个块只能写入一次。

BLOCK1 由于数据不足 32 字节，因此不足 32 字节的部分在 RS (44，32) 编码时硬件会将其视为 0，不会影响最终的编码结果。

使用 RS (44，32) 编码方式的块中，BLOCK1 数据参数为 24 字节，RS 校验码为 12 字节，因此 BLOCK1 在 eFuse 存储器中共占据了 24 + 12 = 36 字节的存储空间。

其余块 (BLOCK2 ~ 10) 的数据参数均为 32 字节，RS 校验码为 12 字节，因此在 eFuse 存储器中共占据了 (32 + 12) * 9 = 396 字节的存储空间。

7.3.2 烧写参数

烧写 eFuse 参数时，需要按块烧写。BLOCK0 ~ BLOCK10 共用同一段地址来存储即将烧写的参数。通过配置 EFUSE_BLK_NUM 参数表明当前需要烧写的是哪一个块。

由于读取数据的寄存器和烧写数据的寄存器在位置上存在一一对应的关系（详见表 7.3-4），因此用户可以通过查找读取寄存器的参数描述和位置确定待烧写数据在烧写寄存器中的具体位置。

例如，用户想将 BLOCK0 中的 EFUSE_USB_EXCHG_PINS 烧写成 1:

1. 查找 BLOCK0 读取数据的寄存器 EFUSE_RD_REPEAT_DATA0 ~ 4_REG，定位到 EFUSE_USB_EXCHG_PINS 参数位于 EFUSE_RD_REPEAT_DATA0_REG 寄存器的第 25 个比特。
2. 将 EFUSE_PGM_DATA1_REG 待烧写数据的第 25 个比特写成 1。
3. 执行下方的烧写步骤。

烧写准备

- 确认时钟频率为 48 MHz
- 烧写 BLOCK0
  1. 将寄存器域 EFUSE_BLK_NUM 配置为 0。
  2. EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG 配置 BLOCK0 即将烧写的参数。EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG 以及 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中的数据不影响 BLOCK0 的烧写。
- BLOCK1 中数据信息在出厂时已经烧写完毕，不允许用户再次烧写。
- 烧写 BLOCK2 ~ 10
  1. 将寄存器域 EFUSE_BLK_NUM 配置为烧写的块数值。
  2. EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 配置即将烧写的参数，EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中配置对应的 RS 校验码。

烧写流程

烧写参数的流程如下:

1. 配置 EFUSE_BLK_NUM 参数，决定烧写哪一个块。
2. 将需要烧写的参数填写到寄存器 EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 和 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中。
3. 确保 eFuse 烧写电压 VDDQ 的配置正确，具体请参考章节 7.3.4。
4. 配置寄存器 EFUSE_CONF_REG 的 EFUSE_OP_CODE 位域为 0x5A5A。
5. 配置寄存器 EFUSE_CMD_REG 的 EFUSE_PGM_CMD 位域为 1。
6. 轮询寄存器 EFUSE_CMD_REG 直到其为 0x0，或者等待 PGM_DONE（烧写完成）中断产生。识别 PGM_DONE 或 READ_DONE （读取完成）中断产生的方法详见章节 7.3.3 最后的说明。
7. 将 EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 和 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中写入的参数清零。
8. 执行更新 eFuse 控制器读寄存器操作使写入的新值生效，具体请参考章节 7.3.3。

9. 检查错误寄存器内容。若读取错误寄存器内数值不为 0，需要再次执行上述步骤 1 ~ 7 重新烧写一次。

   通过该方式可以解决由于烧写不充分导致错误寄存器内数值不为 0 的问题。

   对于不同的 eFuse 块，需要检查的错误寄存器如下。

   - BLOCK0：EFUSE_RD_REPEAT_DATA_ERRO_REG ~ EFUSE_RD_REPEAT_DATA_ERR4_REG
   - BLOCK1：EFUSE_RD_MAC_SYS_ERR_NUM、EFUSE_RD_MAC_SYS_FAIL
   - BLOCK2：EFUSE_RD_SYS_PART1_DATA_ERR_NUM、EFUSE_RD_SYS_PART1_DATA_FAIL
   - BLOCK3：EFUSE_RD_USR_DATA_ERR_NUM、EFUSE_RD_USR_DATA_FAIL
   - BLOCK4：EFUSE_RD_KEY0_DATA_ERR_NUM、EFUSE_RD_KEY0_DATA_FAIL
   - BLOCK5：EFUSE_RD_KEY1_DATA_ERR_NUM、EFUSE_RD_KEY1_DATA_FAIL
   - BLOCK6：EFUSE_RD_KEY2_DATA_ERR_NUM、EFUSE_RD_KEY2_DATA_FAIL
   - BLOCK7：EFUSE_RD_KEY3_DATA_ERR_NUM、EFUSE_RD_KEY3_DATA_FAIL
   - BLOCK8：EFUSE_RD_KEY4_DATA_ERR_NUM、EFUSE_RD_KEY4_DATA_FAIL
   - BLOCK9：EFUSE_RD_KEY5_DATA_ERR_NUM、EFUSE_RD_KEY5_DATA_FAIL
   - BLOCK10：EFUSE_RD_SYS_PART2_DATA_ERR_NUM、EFUSE_RD_SYS_PART2_DATA_FAIL

限制

BLOCK0 中不同的参数，甚至对于同一个参数中的不同位可以在多次烧写中分别完成。但是并不推荐这样做，而是建议尽量减少烧写次数。我们建议对于某个参数中的所有需要烧写的位都在一次烧写中完成。并且当 EFUSE_WR_DIS 的某个位管理的所有参数都烧写之后，就立即烧写 EFUSE_WR_DIS 的这个位。甚至可以在同一次烧写中既烧写 EFUSE_WR_DIS 的某个位管理的所有参数，同时也烧写 EFUSE_WR_DIS 的这个位。

BLOCK1 中数据信息在出厂时已经烧写完毕，不允许再次烧写。

BLOCK2 ~ 10 中每一个 BLOCK 都只能烧写一次，不允许重复烧写。

7.3.3 用户读取参数

用户不能直接读取 eFuse 存储器中烧写的信息内容。eFuse 控制器能够将烧写的数据信息读取到对应的地址段的寄存器内，用户再通过读取以 EFUSE_RD_ 开始的寄存器来获取 eFuse 存储器信息。下表 7.3-4 列出了读取数据的寄存器名称以及对应烧写时的烧写寄存器名称。

表 7.3-4. 用户读取寄存器信息

BLOCK	读寄存器	烧写寄存器
0	EFUSE_RD_WR_DISO_REG	EFUSE_PGM_DATAO_REG
0	EFUSE_RD_REPEAT_DATA*n*_REG (*n*: 0 ~ 4)	EFUSE_PGM_DATA*n*_REG (*n*: 1 ~ 5)
1	EFUSE_RD_MAC_SYS_*n*_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA*n*_REG (*n*: 0 ~ 5)
2	EFUSE_RD_SYS_PART1_DATA*n*_REG (*n*: 0 ~ 7)	EFUSE_PGM_DATA*n*_REG (*n*: 0 ~ 7)
3	EFUSE_RD_USR_DATA*n*_REG (*n*: 0 ~ 7)	EFUSE_PGM_DATA*n*_REG (*n*: 0 ~ 7)
4-9	EFUSE_RD_KEY*n*_DATA*m*_REG (*n*: 0 ~ 5) (*m*: 0 ~ 7)	EFUSE_PGM_DATA*n*_REG (*n*: 0 ~ 7)
10	EFUSE_RD_SYS_PART2_DATA*n*_REG (*n*: 0 ~ 7)	EFUSE_PGM_DATA*n*_REG (*n*: 0 ~ 7)

更新 eFuse 控制器的读寄存器

eFuse 控制器通过读取 eFuse 存储器来更新相应寄存器的数据。读取操作在系统复位时进行，也可以根据需要由用户手动触发（例如在需要读取新烧写 eFuse 存储器中的数据内容时）。用户触发 eFuse 存储器读取操作的流程如下：

1. 配置寄存器 EFUSE_CONF_REG 的 EFUSE_OP_CODE 位域为 0x5AA5。
2. 配置寄存器 EFUSE_CMD_REG 的 EFUSE_READ_CMD 位域为 1。
3. 轮询寄存器 EFUSE_CMD_REG 直到其为 0x0，或者等待 READ_DONE 中断产生，识别 PGM_DONE 或 READ_DONE 中断产生的方法详见下方说明。
4. 用户从 eFuse 存储器中读取参数的值。

eFuse 控制器的读寄存器中的数值将一直保持到下一次执行更新 eFuse 存储器读取操作。

烧写错误检测

烧写错误记录寄存器允许用户检测 eFuse 存储器中参数的完整性。例如，四备份参数部分是否出现四备份不一致的情况，以及受 RS 编码保护的参数部分是否出现 RS 解码失败的情况。

EFUSE_RD_REPEAT_DATA_ERR*n*_REG (*n*: 0 ~ 3) 寄存器用于指示 BLOCK0 中除了 EFUSE_WR_DIS 外的其他参数的烧写是否出错。对应位为 1 代表烧写出错，为 0 代表烧写正确。

EFUSE_RD_RS_DATA_ERR*n*_REG (*n*: 0 ~ 1) 寄存器记录 eFuse 控制器读 BLOCK1 ~ BLOCK10 过程中，纠错的字节数目以及 RS 解码是否失败的信息。

每次更新 eFuse 控制器读寄存器操作完成之后，上述寄存器内的数值都会被更新。

识别烧写/读取操作完成

识别烧写/读取操作完成的方法如下。位 1 对应烧写操作，位 0 对应读取操作。

- 方法 1：轮询寄存器 EFUSE_INT_RAW_REG 的位 1/0，当值为 1 时，则表示烧写/读取操作完成。
- 方法 2：
    1. 将寄存器 EFUSE_INT_ENA_REG 的位 1/0 置 1，使 eFuse 控制器能够产生 PGM_DONE 或 READ_DONE 中断。
    2. 配置中断矩阵使 CPU 能够响应 eFuse 的中断信号，可参见 11 中断矩阵。
    3. 等待 PGM_DONE 或 READ_DONE 中断产生。
    4. 对寄存器 EFUSE_INT_CLR_REG 的位 1/0 置 1 以清除 PGM_DONE 或 READ_DONE 中断。

注意事项

在 eFuse 控制器执行寄存器更新操作过程中，会复用 EFUSE_PGM_DATA*n*_REG (n=0, 1, ..., 7) 寄存器的存储空间，所以在启动 eFuse 控制器更新寄存器之前，不要将有意义的数据写入上述寄存器中。
芯片启动过程中，eFuse 控制器会自动更新 eFuse 存储器数据到用户可访问的寄存器。用户可以通过读取相应的寄存器获取 eFuse 存储器内烧写的数据。因此，用户无需再驱动 eFuse 控制器执行读更新操作。

7.3.4 eFuse VDDQ 时序

eFuse 控制器烧写电压 VDDQ 的时序配置参数需要满足以下条件：

- EFUSE_DAC_NUM（烧写电压上升周期数），默认烧写电压为 2.5 V，每个上升周期增加 0.01 V，该参数对应的默认值为 255；
- EFUSE_DAC_CLK_DIV（烧写电压时钟分频系数），要求烧写电压时钟周期大于 1 μs；
- EFUSE_PWR_ON_NUM （eFuse 烧写电压上电等待时间），要求该等待时间结束后烧写电压已稳定，即要求配置数值大于 EFUSE_DAC_CLK_DIV * EFUSE_DAC_NUM；
- EFUSE_PWR_OFF_NUM（烧写电压掉电等待时间），要求该时间大于 10 μs；

VDDQ 的默认时序参数配置如表 7.3-5 所示。

表 7.3-5. VDDQ 默认时序参数配置

EFUSE_DAC_NUM	EFUSE_DAC_CLK_DIV	EFUSE_PWR_ON_NUM	EFUSE_PWR_OFF_NUM
0xFF	0x28	0x3000	0x190

7.3.5 硬件模块使用参数

硬件模块使用参数是通过电路连接实现的，用户无法干预这个过程。硬件使用的参数为表 7.3-1 和 7.3-3 “硬件使用”一栏中标记为“Y”的参数。

7.4 中断

ESP32-C5 的 eFuse 控制器可以生成以下中断信号并发送给中断矩阵。

- EFUSE_INT

以上中断信号由 eFuse 控制器的内部中断源生成。表 7.4-1 列出了 eFuse 控制器的内部中断源、中断触发条件、以及生成的中断信号。

表 7.4-1. eFuse 控制器的内部中断源

内部中断源	触发条件	中断信号
EFUSE_PGM_DONE_INT	eFuse 烧写完成	EFUSE_INT
EFUSE_READ_DONE_INT	eFuse 读取完成	

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 7.5 寄存器列表。

7.5 寄存器列表

本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
烧写数据寄存器			
EFUSE_PGM_DATA0_REG	存放待烧写数据的第 0 个寄存器内容	0x0000	R/W
EFUSE_PGM_DATA1_REG	存放待烧写数据的第 1 个寄存器内容	0x0004	R/W
EFUSE_PGM_DATA2_REG	存放待烧写数据的第 2 个寄存器内容	0x0008	R/W
EFUSE_PGM_DATA3_REG	存放待烧写数据的第 3 个寄存器内容	0x000C	R/W
EFUSE_PGM_DATA4_REG	存放待烧写数据的第 4 个寄存器内容	0x0010	R/W
EFUSE_PGM_DATA5_REG	存放待烧写数据的第 5 个寄存器内容	0x0014	R/W
EFUSE_PGM_DATA6_REG	存放待烧写数据的第 6 个寄存器内容	0x0018	R/W
EFUSE_PGM_DATA7_REG	存放待烧写数据的第 7 个寄存器内容	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	存放待烧写 RS 代码的第 0 个寄存器数据	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	存放待烧写 RS 代码的第 1 个寄存器数据	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	存放待烧写 RS 代码的第 2 个寄存器数据	0x0028	R/W
BLOCK0 读取数据寄存器			
EFUSE_RD_WR_DIS_REG	BLOCK0 的第 0 个寄存器内容，表示是否禁用 eFuse 存储器各个位的烧写。	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	BLOCK0 的第 1 个寄存器内容	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	BLOCK0 的第 2 个寄存器内容	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	BLOCK0 的第 3 个寄存器内容	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	BLOCK0 的第 4 个寄存器内容	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	BLOCK0 的第 5 个寄存器内容	0x0040	RO
BLOCK1 读取数据寄存器			
EFUSE_RD_MAC_SYS0_REG	BLOCK1 的第 0 个寄存器内容	0x0044	RO
EFUSE_RD_MAC_SYS1_REG	BLOCK1 的第 1 个寄存器内容	0x0048	RO
EFUSE_RD_MAC_SYS2_REG	BLOCK1 的第 2 个寄存器内容	0x004C	RO
EFUSE_RD_MAC_SYS3_REG	BLOCK1 的第 3 个寄存器内容	0x0050	RO
EFUSE_RD_MAC_SYS4_REG	BLOCK1 的第 4 个寄存器内容	0x0054	RO
EFUSE_RD_MAC_SYS5_REG	BLOCK1 的第 5 个寄存器内容	0x0058	RO
BLOCK2 读取数据寄存器			
EFUSE_RD_SYS_PART1_DATA0_REG	BLOCK2 (system) 的第 0 个寄存器内容	0x005C	RO
EFUSE_RD_SYS_PART1_DATA1_REG	BLOCK2 (system) 的第 1 个寄存器内容	0x0060	RO
EFUSE_RD_SYS_PART1_DATA2_REG	BLOCK2 (system) 的第 2 个寄存器内容	0x0064	RO
EFUSE_RD_SYS_PART1_DATA3_REG	BLOCK2 (system) 的第 3 个寄存器内容	0x0068	RO
EFUSE_RD_SYS_PART1_DATA4_REG	BLOCK2 (system) 的第 4 个寄存器内容	0x006C	RO
EFUSE_RD_SYS_PART1_DATA5_REG	BLOCK2 (system) 的第 5 个寄存器内容	0x0070	RO
EFUSE_RD_SYS_PART1_DATA6_REG	BLOCK2 (system) 的第 6 个寄存器内容	0x0074	RO
EFUSE_RD_SYS_PART1_DATA7_REG	BLOCK2 (system) 的第 7 个寄存器内容	0x0078	RO
BLOCK3 读取数据寄存器			

名称	描述	地址	访问
EFUSE_RD_USR_DATA0_REG	BLOCK3 (user) 的第 0 个寄存器内容	0x007C	RO
EFUSE_RD_USR_DATA1_REG	BLOCK3 (user) 的第 1 个寄存器内容	0x0080	RO
EFUSE_RD_USR_DATA2_REG	BLOCK3 (user) 的第 2 个寄存器内容	0x0084	RO
EFUSE_RD_USR_DATA3_REG	BLOCK3 (user) 的第 3 个寄存器内容	0x0088	RO
EFUSE_RD_USR_DATA4_REG	BLOCK3 (user) 的第 4 个寄存器内容	0x008C	RO
EFUSE_RD_USR_DATA5_REG	BLOCK3 (user) 的第 5 个寄存器内容	0x0090	RO
EFUSE_RD_USR_DATA6_REG	BLOCK3 (user) 的第 6 个寄存器内容	0x0094	RO
EFUSE_RD_USR_DATA7_REG	BLOCK3 (user) 的第 7 个寄存器内容	0x0098	RO
BLOCK4 读取数据寄存器			
EFUSE_RD_KEY0_DATA0_REG	BLOCK4 (KEY0) 的第 0 个寄存器内容	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	BLOCK4 (KEY0) 的第 1 个寄存器内容	0x00A0	RO
EFUSE_RD_KEY0_DATA2_REG	BLOCK4 (KEY0) 的第 2 个寄存器内容	0x00A4	RO
EFUSE_RD_KEY0_DATA3_REG	BLOCK4 (KEY0) 的第 3 个寄存器内容	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	BLOCK4 (KEY0) 的第 4 个寄存器内容	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	BLOCK4 (KEY0) 的第 5 个寄存器内容	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	BLOCK4 (KEY0) 的第 6 个寄存器内容	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	BLOCK4 (KEY0) 的第 7 个寄存器内容	0x00B8	RO
BLOCK5 读取数据寄存器			
EFUSE_RD_KEY1_DATA0_REG	BLOCK5 (KEY1) 的第 0 个寄存器内容	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	BLOCK5 (KEY1) 的第 1 个寄存器内容	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	BLOCK5 (KEY1) 的第 2 个寄存器内容	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	BLOCK5 (KEY1) 的第 3 个寄存器内容	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	BLOCK5 (KEY1) 的第 4 个寄存器内容	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	BLOCK5 (KEY1) 的第 5 个寄存器内容	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	BLOCK5 (KEY1) 的第 6 个寄存器内容	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	BLOCK5 (KEY1) 的第 7 个寄存器内容	0x00D8	RO
BLOCK6 读取数据寄存器			
EFUSE_RD_KEY2_DATA0_REG	BLOCK6 (KEY2) 的第 0 个寄存器内容	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	BLOCK6 (KEY2) 的第 1 个寄存器内容	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	BLOCK6 (KEY2) 的第 2 个寄存器内容	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	BLOCK6 (KEY2) 的第 3 个寄存器内容	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	BLOCK6 (KEY2) 的第 4 个寄存器内容	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	BLOCK6 (KEY2) 的第 5 个寄存器内容	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	BLOCK6 (KEY2) 的第 6 个寄存器内容	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	BLOCK6 (KEY2) 的第 7 个寄存器内容	0x00F8	RO
BLOCK7 读取数据寄存器			
EFUSE_RD_KEY3_DATA0_REG	BLOCK7 (KEY3) 的第 0 个寄存器内容	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	BLOCK7 (KEY3) 的第 1 个寄存器内容	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	BLOCK7 (KEY3) 的第 2 个寄存器内容	0x0104	RO
EFUSE_RD_KEY3_DATA3_REG	BLOCK7 (KEY3) 的第 3 个寄存器内容	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	BLOCK7 (KEY3) 的第 4 个寄存器内容	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	BLOCK7 (KEY3) 的第 5 个寄存器内容	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	BLOCK7 (KEY3) 的第 6 个寄存器内容	0x0114	RO

名称	描述	地址	访问
EFUSE_RD_KEY3_DATA7_REG	BLOCK7 (KEY3) 的第 7 个寄存器内容	0x0118	RO
BLOCK8 读取数据寄存器			
EFUSE_RD_KEY4_DATA0_REG	BLOCK8 (KEY4) 的第 0 个寄存器内容	0x011C	RO
EFUSE_RD_KEY4_DATA1_REG	BLOCK8 (KEY4) 的第 1 个寄存器内容	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	BLOCK8 (KEY4) 的第 2 个寄存器内容	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	BLOCK8 (KEY4) 的第 3 个寄存器内容	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	BLOCK8 (KEY4) 的第 4 个寄存器内容	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	BLOCK8 (KEY4) 的第 5 个寄存器内容	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	BLOCK8 (KEY4) 的第 6 个寄存器内容	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	BLOCK8 (KEY4) 的第 7 个寄存器内容	0x0138	RO
BLOCK9 读取数据寄存器			
EFUSE_RD_KEY5_DATA0_REG	BLOCK9 (KEY5) 的第 0 个寄存器内容	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	BLOCK9 (KEY5) 的第 1 个寄存器内容	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	BLOCK9 (KEY5) 的第 2 个寄存器内容	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	BLOCK9 (KEY5) 的第 3 个寄存器内容	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	BLOCK9 (KEY5) 的第 4 个寄存器内容	0x014C	RO
EFUSE_RD_KEY5_DATA5_REG	BLOCK9 (KEY5) 的第 5 个寄存器内容	0x0150	RO
EFUSE_RD_KEY5_DATA6_REG	BLOCK9 (KEY5) 的第 6 个寄存器内容	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	BLOCK9 (KEY5) 的第 7 个寄存器内容	0x0158	RO
BLOCK10 读取数据寄存器			
EFUSE_RD_SYS_PART2_DATA0_REG	BLOCK10 (system) 的第 0 个寄存器内容	0x015C	RO
EFUSE_RD_SYS_PART2_DATA1_REG	BLOCK10 (system) 的第 1 个寄存器内容	0x0160	RO
EFUSE_RD_SYS_PART2_DATA2_REG	BLOCK10 (system) 的第 2 个寄存器内容	0x0164	RO
EFUSE_RD_SYS_PART2_DATA3_REG	BLOCK10 (system) 的第 3 个寄存器内容	0x0168	RO
EFUSE_RD_SYS_PART2_DATA4_REG	BLOCK10 (system) 的第 4 个寄存器内容	0x016C	RO
EFUSE_RD_SYS_PART2_DATA5_REG	BLOCK10 (system) 的第 5 个寄存器内容	0x0170	RO
EFUSE_RD_SYS_PART2_DATA6_REG	BLOCK10 (system) 的第 6 个寄存器内容	0x0174	RO
EFUSE_RD_SYS_PART2_DATA7_REG	BLOCK10 (system) 的第 7 个寄存器内容	0x0178	RO
BLOCK0 错误报告寄存器			
EFUSE_RD_REPEAT_DATA_ERR0_REG	BLOCK0 参数烧写错误记录第 0 个寄存器	0x017C	RO
EFUSE_RD_REPEAT_DATA_ERR1_REG	BLOCK0 参数烧写错误记录第 1 个寄存器	0x0180	RO
EFUSE_RD_REPEAT_DATA_ERR2_REG	BLOCK0 参数烧写错误记录第 2 个寄存器	0x0184	RO
EFUSE_RD_REPEAT_DATA_ERR3_REG	BLOCK0 参数烧写错误记录第 3 个寄存器	0x0188	RO
EFUSE_RD_REPEAT_DATA_ERR4_REG	BLOCK0 参数烧写错误记录第 4 个寄存器	RO	
RS Block 错误报告寄存器			
EFUSE_RD_RS_DATA_ERR0_REG	BLOCK1 ~ 10 参数烧写错误记录第 0 个寄存器	0x0190	RO
EFUSE_RD_RS_DATA_ERR1_REG	BLOCK1 ~ 10 参数烧写错误记录第 1 个寄存器	0x0194	RO
eFuse 版本寄存器			
EFUSE_DATE_REG	eFuse 版本控制寄存器	0x0198	R/W
eFuse 时钟寄存器			
EFUSE_CLK_REG	eFuse 时钟配置寄存器	0x01C8	R/W
eFuse 配置寄存器			
EFUSE_CONF_REG	配置 eFuse 运行模式	0x01CC	R/W

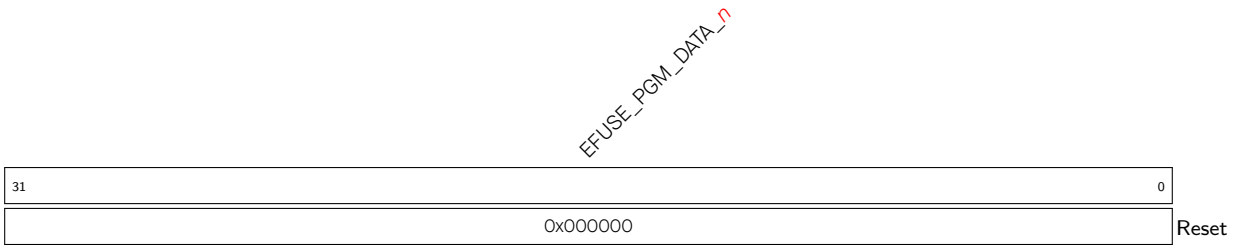
名称	描述	地址	访问
EFUSE_DAC_CONF_REG	配置 eFuse 烧写电压	0x01EC	R/W
EFUSE_RD_TIM_CONF_REG	配置 eFuse 读取时序参数	0x01F0	R/W
EFUSE_WR_TIM_CONF1_REG	配置 eFuse 烧写时序参数	0x01F4	R/W
EFUSE_WR_TIM_CONF2_REG	配置 eFuse 烧写时序参数	0x01F8	R/W
EFUSE_WR_TIM_CONF0_RS_BYPASS_REG	配置 eFuse 烧写时序参数和 RS bypass 操作	0x01FC	varies
eFuse 状态寄存器			
EFUSE_STATUS_REG	eFuse 状态寄存器	0x01D4	RO
eFuse 命令寄存器			
EFUSE_CMD_REG	eFuse 命令寄存器	0x01D8	varies
eFuse 中断寄存器			
EFUSE_INT_RAW_REG	eFuse 原始中断寄存器	0x01DC	R/ SS/ WTC
EFUSE_INT_ST_REG	eFuse 中断状态寄存器	0x01E0	RO
EFUSE_INT_ENA_REG	eFuse 中断使能寄存器	0x01E4	R/W
EFUSE_INT_CLR_REG	eFuse 中断清除寄存器	0x01E8	WT

7.6 寄存器

本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

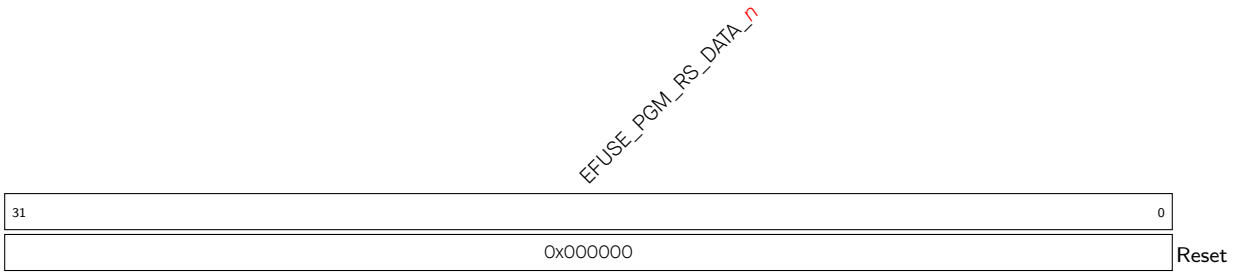
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 7.1. EFUSE_PGM_DATA*n*_REG (*n*: 0-7) (0x0000+0x4**n*)



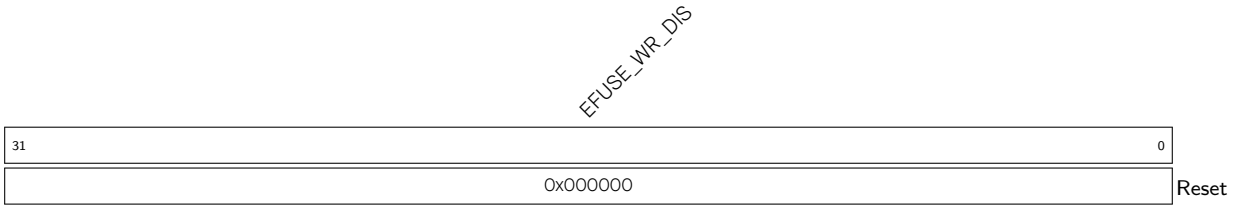
EFUSE_PGM_DATA_*n* 配置待烧写数据的第 *n* 个 32 位数据。(R/W)

Register 7.2. EFUSE_PGM_CHECK_VALUE*n*_REG (*n*: 0-2) (0x0020+0x4**n*)



EFUSE_PGM_RS_DATA_*n* 配置待烧写的第 *n* 个 32 位 RS 码。(R/W)

Register 7.3. EFUSE_RD_WR_DIS_REG (0x002C)



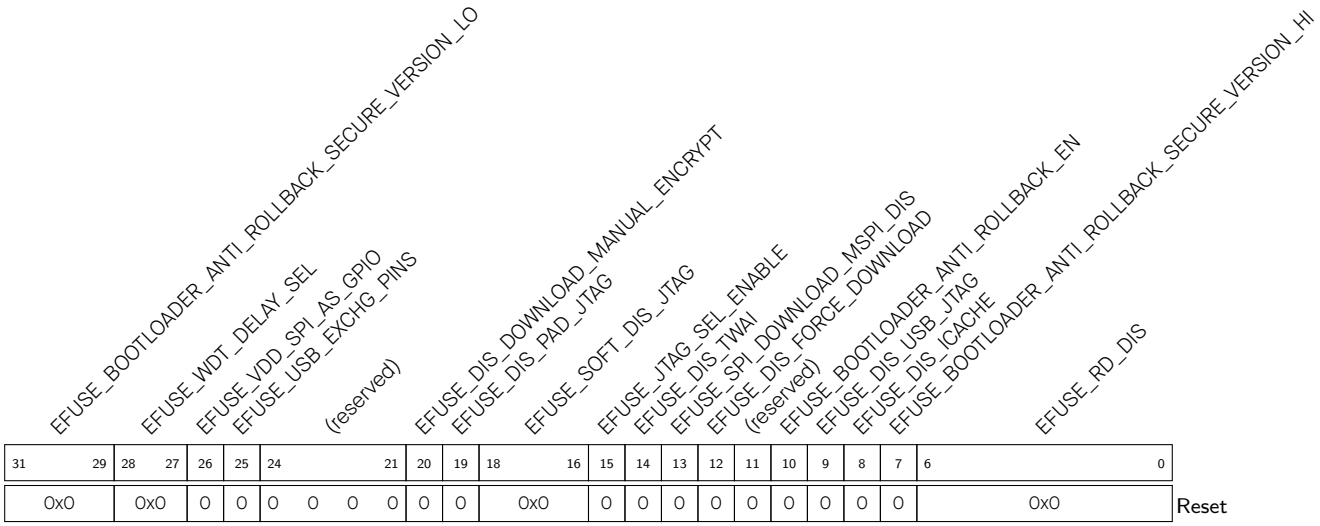
EFUSE_WR_DIS 表示是否禁用 eFuse 存储器各个位的烧写。本寄存器域中各个位与 eFuse 存储器各个位的对应关系请参考表 7.3-1 和表 7.3-3。

1：禁用

0：使能

(RO)

Register 7.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS 表示是否禁用 eFuse 各个 block (BLOCK4 ~ BLOCK10) 的读取。本寄存器域中各个位与 eFuse block 的对应关系请参考表 7.3-3。

1：禁用

0：使能

(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_VERSION_HI 表示一级 (ROM) 引导程序所使用的二级引导程序的防回滚安全版本（该字段的高位部分）。(RO)

EFUSE_DIS_ICACHE 表示 cache 是否禁用。

1：禁用

0：使能

(RO)

EFUSE_DIS_USB_JTAG 表示是否禁用 USB Serial/JTAG 模块中 USB 转 JTAG 功能。注意，只有当 EFUSE_DIS_USB_JTAG 被配置为 0 时，EFUSE_DIS_USB_JTAG 才可用。有关详细信息，请参见章节 10 *芯片 Boot 控制*。

1：禁用

0：使能

(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_EN 表示是否启用了第二阶段引导加载程序的回滚保护检查。

1：启用

0：禁用

(RO)

EFUSE_DIS_FORCE_DOWNLOAD 表示是否禁用强制芯片进入 Download 模式的功能。

1：禁用

0：使能

(RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS 表示在 boot_mode_download 过程中 SPI0 控制器是否禁用。

1：禁用

0：使能

(RO)

见下页……

Register 7.4. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

接上页……

EFUSE_DIS_TWAI 表示是否禁用 TWAI® 功能。

1：禁用

0：使能

(RO)

EFUSE_JTAG_SEL_ENABLE 表示EFUSE_DIS_PAD_JTAG 和EFUSE_DIS_USB_JTAG 均配置为 0 时，是否使能通过 strapping 管脚值选择 JTAG 信号源。详情请参考章节 10 芯片 Boot 控制。

1：使能

0：禁用

(RO)

EFUSE_SOFT_DIS_JTAG 表示是否软禁用 PAD JTAG。可通过 HMAC 重新启动。

奇数个比特为 1：禁用

偶数个比特为 1：使能

(RO)

EFUSE_DIS_PAD_JTAG 表示是否硬禁用 PAD JTAG（永久）。

1：禁用

0：使能

(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 表示是否使能 flash 加密（SPI 启动模式除外）。

1：禁用

0：使能

(RO)

EFUSE_USB_EXCHG_PINS 表示是否交换 USB D+ 和 D- 管脚。

1：交换

0：不交换

(RO)

EFUSE_VDD_SPI_AS_GPIO 表示 VDD SPI 管脚是否作为 GPIO。

1：作为

0：不作为

(RO)

EFUSE_WDT_DELAY_SEL 表示 RTC 看门狗的超时阈值。

0：原始配置的 STG0 阈值 × 2

1：原始配置的 STG0 阈值 × 4

2：原始配置的 STG0 阈值 × 8

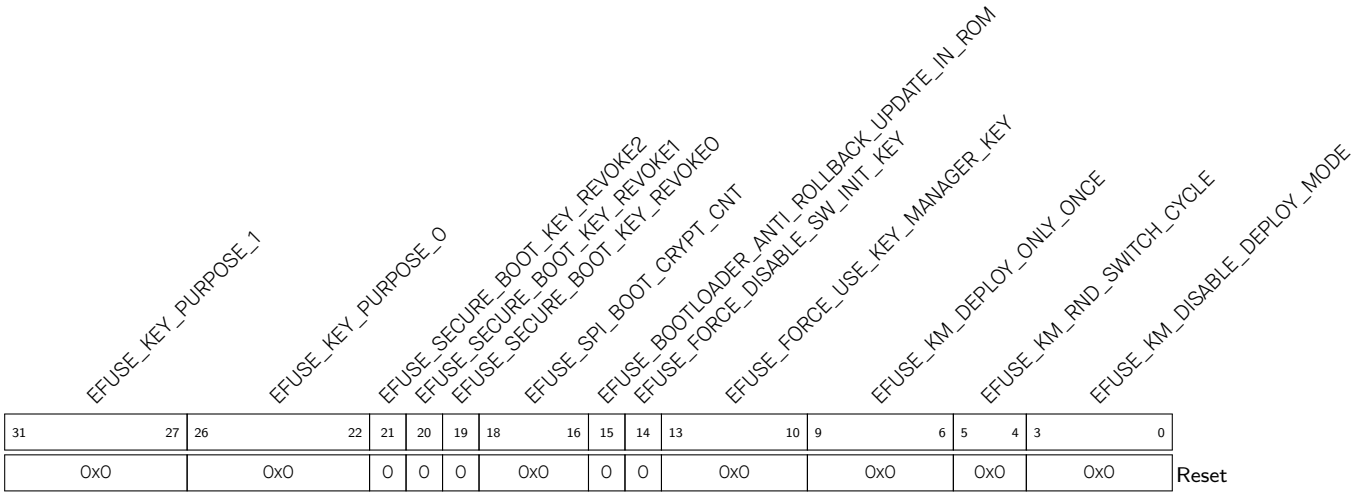
3：原始配置的 STG0 阈值 × 16

(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_VERSION_LO 表示 ROM 引导加载程序使用的第二阶段引导加载程序的回滚保护安全版本（字段的低位部分）。

(RO)

Register 7.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)



EFUSE_KM_DISABLE_DEPLOY_MODE 表示是否禁用了密钥管理器的新密钥部署。

Bit0：表示是否禁用了新的 ECDSA_DS 密钥部署

0：启用

1：禁用

Bit1：表示是否禁用了新的 XTS-AES（flash 和 PSRAM）密钥部署

0：启用

1：禁用

Bit2：表示是否禁用了新的 HMAC 密钥部署

0：启用

1：禁用

Bit3：表示是否禁用了新的 RSA_DS_KEY 部署

0：启用

1：禁用

(RO)

EFUSE_KM_RND_SWITCH_CYCLE 表示密钥管理器切换随机数的周期。

0：由 KEYMNG_RND_SWITCH_CYCLE 寄存器控制。

1：8 个密钥管理器时钟周期

2：16 个密钥管理器时钟周期

3：32 个密钥管理器时钟周期

(RO)

见下页……

Register 7.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)

接上页……

EFUSE_KM_DEPLOY_ONLY_ONCE 表示对应密钥是否只能部署一次。

Bit0：表示 ECDSA_DS 密钥是否只能部署一次

0：该密钥可多次部署

1：该密钥只能部署一次

Bit1：表示 XTS-AES（flash 和 PSRAM）密钥是否只能部署一次

0：该密钥可多次部署

1：该密钥只能部署一次

Bit2：表示 HMAC 密钥是否只能部署一次

0：该密钥可多次部署

1：该密钥只能部署一次

Bit3：表示 RSA_DS_KEY 是否只能部署一次

0：该密钥可多次部署

1：该密钥只能部署一次

(RO)

EFUSE_FORCE_USE_KEY_MANAGER_KEY 表示对应密钥是否必须来自密钥管理器。

Bit0：表示 ECDSA_DS 密钥是否必须来自密钥管理器

0：该密钥不必来自密钥管理器

1：该密钥必须来自密钥管理器

Bit1：表示 XTS-AES（flash 和 PSRAM）密钥是否必须来自密钥管理器

0：该密钥不必来自密钥管理器

1：该密钥必须来自密钥管理器

Bit2：表示 HMAC 密钥是否必须来自密钥管理器

0：该密钥不必来自密钥管理器

1：该密钥必须来自密钥管理器

Bit3：表示 RSA_DS_KEY 是否必须来自密钥管理器

0：该密钥不必来自密钥管理器

1：该密钥必须来自密钥管理器

(RO)

EFUSE_FORCE_DISABLE_SW_INIT_KEY 表示是否禁用由软件写入的初始化密钥，并强制使用 efuse_init_key。

0：启用

1：禁用

(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_UPDATE_IN_ROM 表示回滚保护 SECURE_VERSION 是否由 ROM 引导加载程序更新。

1：启用

0：禁用

(RO)

见下页……

Register 7.5. EFUSE_RD_REPEAT_DATA1_REG (0x0034)

接上页……

EFUSE_SPI_BOOT_CRYPT_CNT 表示是否使能 SPI boot 加解密。

奇数个比特为 1：使能

偶数个比特为 1：不使能

(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0 表示是否撤销安全启动秘钥摘要 0。

1：撤销

0：不撤销

(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE1 表示是否撤销安全启动秘钥摘要 1。

1：撤销

0：不撤销

(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 表示是否撤销第安全启动秘钥摘要 2。

1：撤销

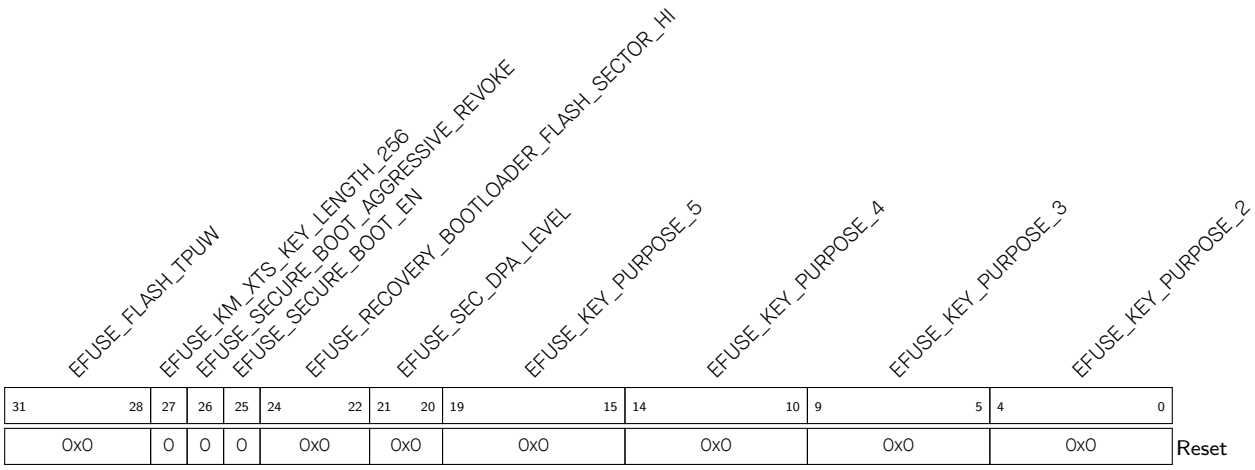
0：不撤销

(RO)

EFUSE_KEY_PURPOSE_0 表示 Key0 用途，更多信息见表 7.3-2。(RO)

EFUSE_KEY_PURPOSE_1 表示 Key1 用途，更多信息见表 7.3-2。(RO)

Register 7.6. EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 表示 Key2 用途，更多信息请见表 7.3-2。(RO)

EFUSE_KEY_PURPOSE_3 表示 Key3 用途，更多信息请见表 7.3-2。(RO)

EFUSE_KEY_PURPOSE_4 表示 Key4 用途，更多信息请见表 7.3-2。(RO)

EFUSE_KEY_PURPOSE_5 表示 Key5 用途，更多信息请见表 7.3-2。(RO)

EFUSE_SEC_DPA_LEVEL 表示抗差分功耗分析 (differential power analysis, DPA) 攻击的安全级别。该寄存器域实际上通过配置时钟随机分频模式来调整抗 DPA 攻击的安全等级。

0：安全级别为 SEC_DPA_OFF

1：安全级别为 SEC_DPA_LOW

2：安全级别为 SEC_DPA_MIDDLE

3：安全级别为 SEC_DPA_HIGH

有关详细信息，请参阅章节 19 *系统寄存器* > 章节 19.3.2 *防 DPA 攻击安全控制*。

(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_HI 表示当主引导加载程序启动失败时，ROM 引导加载程序使用的恢复引导加载程序的起始 flash 扇区（flash 扇区大小为 0x1000）。值为 0 或 0xFFF 时表示该功能被禁用。(字段的高位部分)。(RO)

EFUSE_SECURE_BOOT_EN 表示是否使能安全启动。

1：使能

0：禁用

(RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE 表示是否使能安全启动的激进撤销。

1：使能

0：禁用

(RO)

EFUSE_KM_XTS_KEY_LENGTH_256 表示 flash 加密使用的密钥类型。

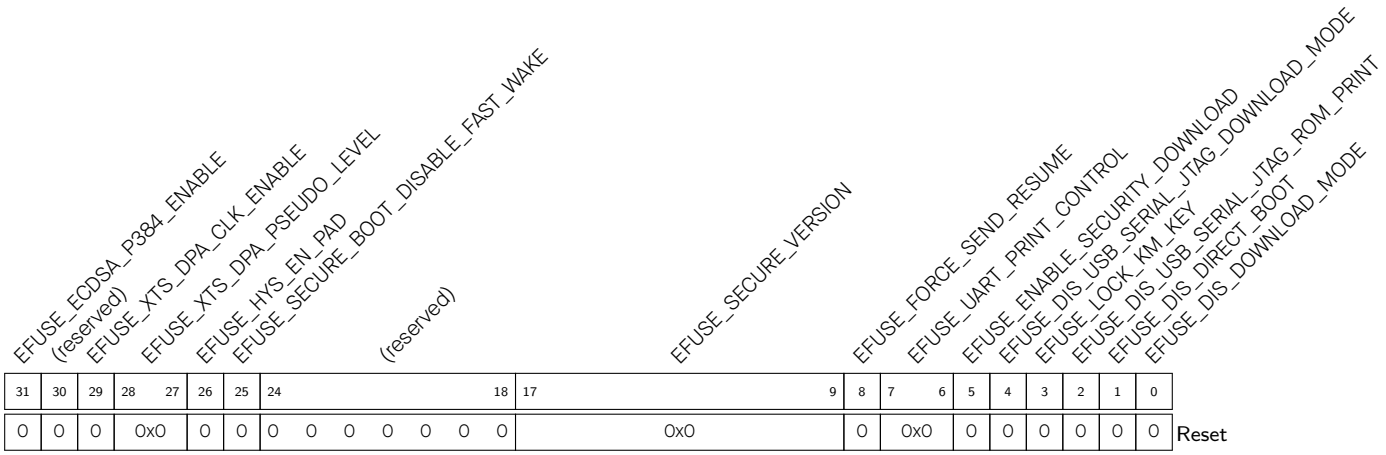
0：XTS-AES-256 密钥

1：XTS-AES-128 密钥

(RO)

EFUSE_FLASH_TPUW 表示 flash 上电等待时间。单位：ms。当值小于 15 时，等待时间为配置值。当值大于等于 15 时，等待时间固定为 30 ms。(RO)

Register 7.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



EFUSE_DIS_DOWNLOAD_MODE 表示是否禁用所有 Download 模式。

1：禁用

0：使能

(RO)

EFUSE_DIS_DIRECT_BOOT 表示是否禁用 direct boot 模式。

1：禁用

0：使能

(RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 表示是否禁用 ROM boot 过程中的 USB-Serial-JTAG 打印。

1：禁用

0：使能

(RO)

EFUSE_LOCK_KM_KEY 表示部署后密钥管理器中的密钥是否已锁定。

0：未锁定

1：已锁定

(RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE 表示是否禁用 USB-Serial-JTAG 下载功能。

1：禁用

0：使能

(RO)

见下页……

Register 7.7. EFUSE_RD_REPEAT_DATA3_REG (0x003C)

接上页……

EFUSE_ENABLE_SECURITY_DOWNLOAD 表示是否使能安全下载。仅支持下载到 flash，不支持读写 RAM 或寄存器，即不支持 stub 下载。

1：使能

0：禁用

(RO)

EFUSE_UART_PRINT_CONTROL 表示 UART 打印类型。

0：强制打印

1：GPIO27 低电平复位时使能打印

2：GPIO27 高电平复位时使能打印

3：强制关闭打印

(RO)

EFUSE_FORCE_SEND_RESUME 表示是否强制 ROM 代码在 SPI 启动过程中发送恢复指令。

1：强制

0：不强制

(RO)

EFUSE_SECURE_VERSION 表示安全版本，用于 ESP-IDF 防回滚功能。(RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE 表示安全启动使能时是否禁用 FAST VERIFY ON WAKE。

1：禁用

0：使能

(RO)

EFUSE_HYS_EN_PAD 表示是否使能 PAD0 ~ 27 的迟滞功能。

1：使能

0：禁用

(RO)

EFUSE_XTS_DPA_PSEUDO_LEVEL 表示 XTS-AES 抗 DPA 攻击的伪轮级别。

0：禁用

1：低

2：中

3：高

(RO)

EFUSE_XTS_DPA_CLK_ENABLE 表示 XTS-AES 抗 DPA 攻击时钟是否使能。

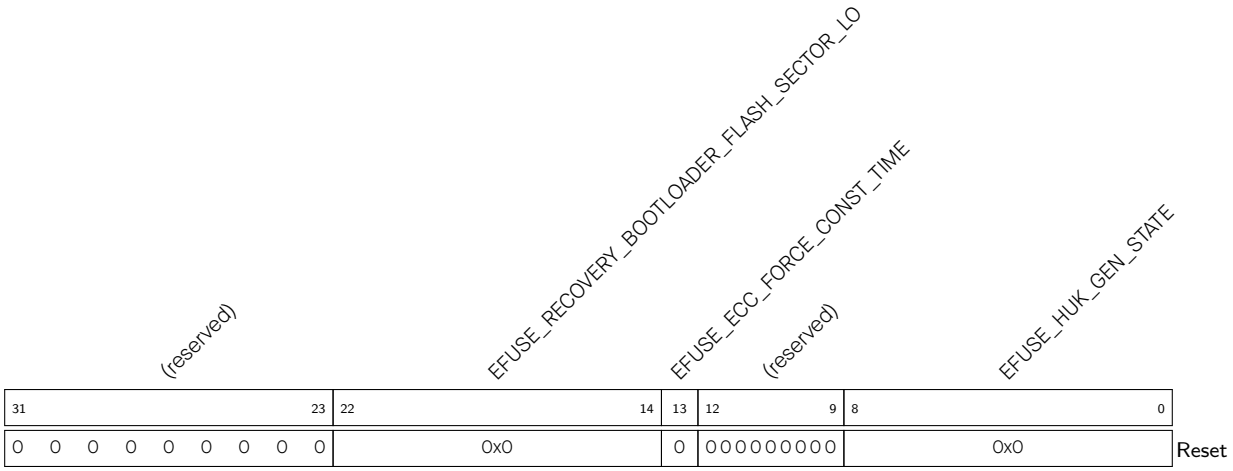
0：禁用

1：使能

(RO)

EFUSE_ECDSA_P384_ENABLE 表示芯片是否支持 ECDSA_DS P384。(RO)

Register 7.8. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



EFUSE_HUK_GEN_STATE 表示 HUK 生成模式是否有效。

1 的位数为奇数：无效

1 的位数为偶数：有效

(RO)

EFUSE_ECC_FORCE_CONST_TIME 表示是否强制 ECC 在点乘计算中使用常时间模式。

0：不强制

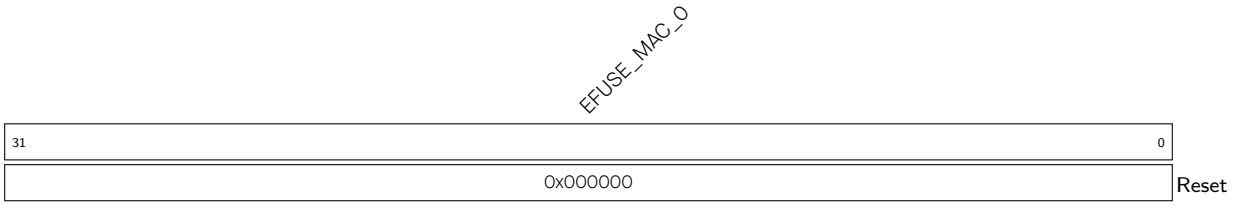
1：强制

(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_LO 表示当主引导加载程序启动失败时，ROM 引导加载程序使用的恢复引导加载程序的起始 flash 扇区（flash 扇区大小为 0x1000）。值为 0 或 0xFFF 时表示该功能被禁用。（字段的低位部分）。

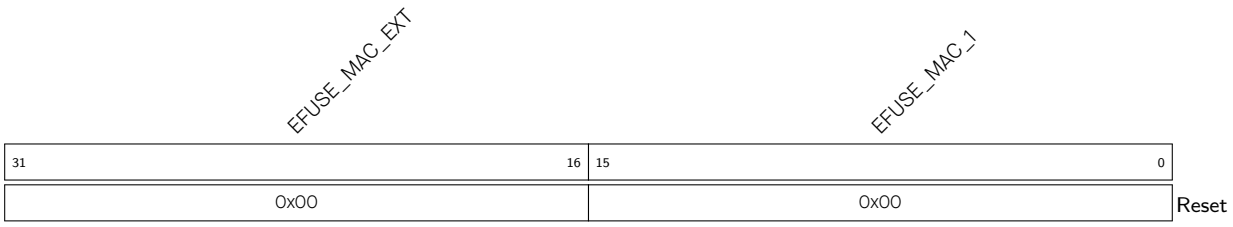
(RO)

Register 7.9. EFUSE_RD_MAC_SYS0_REG (0x0044)



EFUSE_MAC_0 表示 MAC 地址低 32 位。(RO)

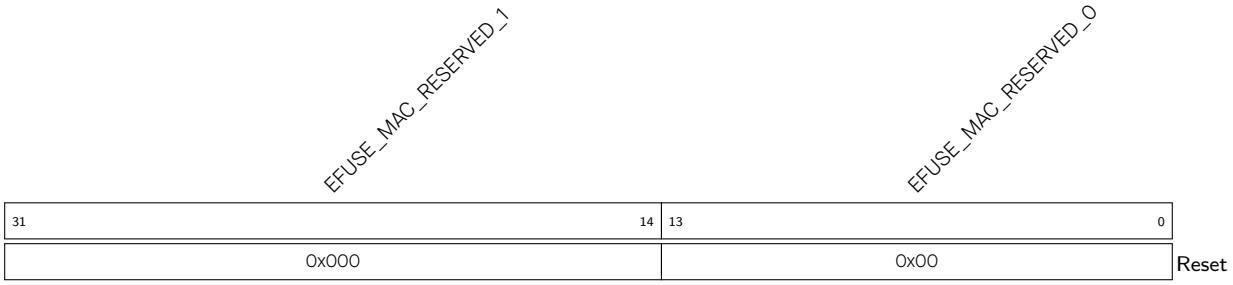
Register 7.10. EFUSE_RD_MAC_SYS1_REG (0x0048)



EFUSE_MAC_1 表示 MAC 地址高 16 位。(RO)

EFUSE_MAC_EXT 表示 MAC 地址扩展位。(RO)

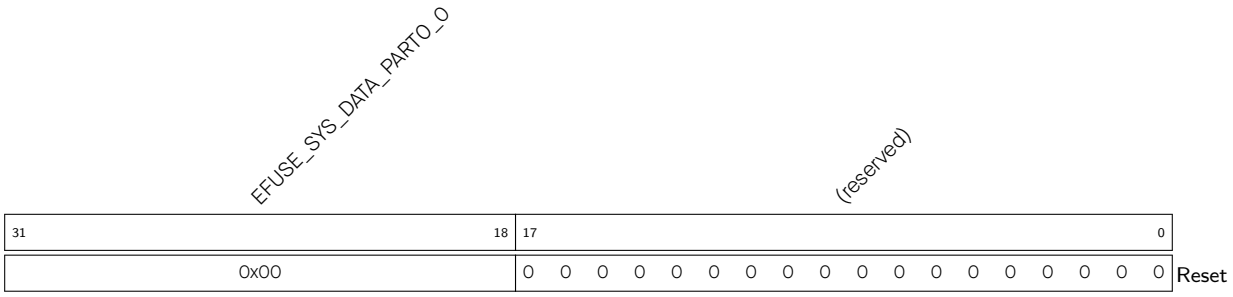
Register 7.11. EFUSE_RD_MAC_SYS2_REG (0x004C)



EFUSE_MAC_RESERVED_0 包含芯片版本信息。EFUSE_RD_MAC_SYS2_REG[3:0] 表示次版本号，EFUSE_RD_MAC_SYS2_REG[5:4] 表示主版本号。其他为保留位。(RO)

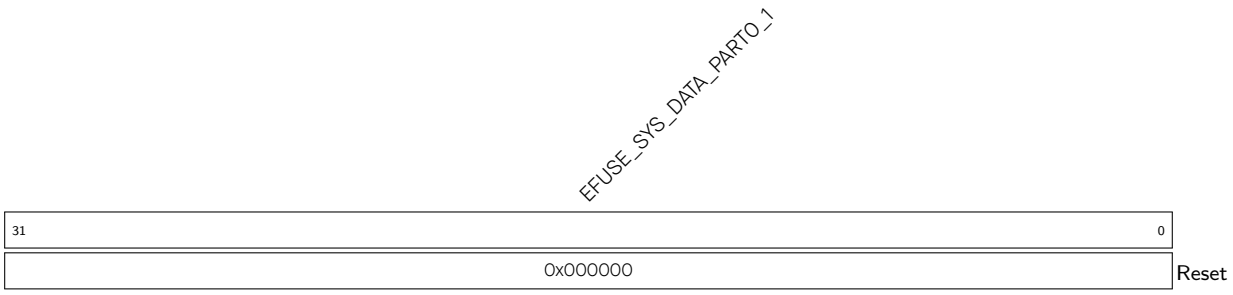
EFUSE_MAC_RESERVED_1 保留。(RO)

Register 7.12. EFUSE_RD_MAC_SYS3_REG (0x0050)



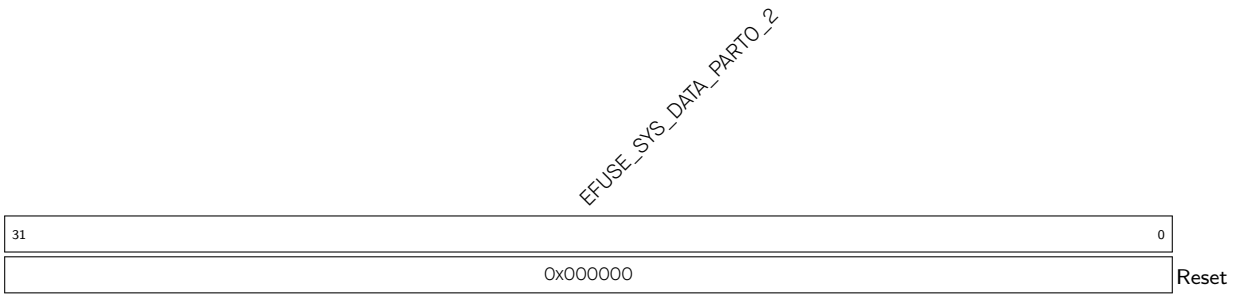
EFUSE_SYS_DATA_PART0_0 表示系统数据第 0 部分的第 1 个 14 位内容。(RO)

Register 7.13. EFUSE_RD_MAC_SYS4_REG (0x0054)



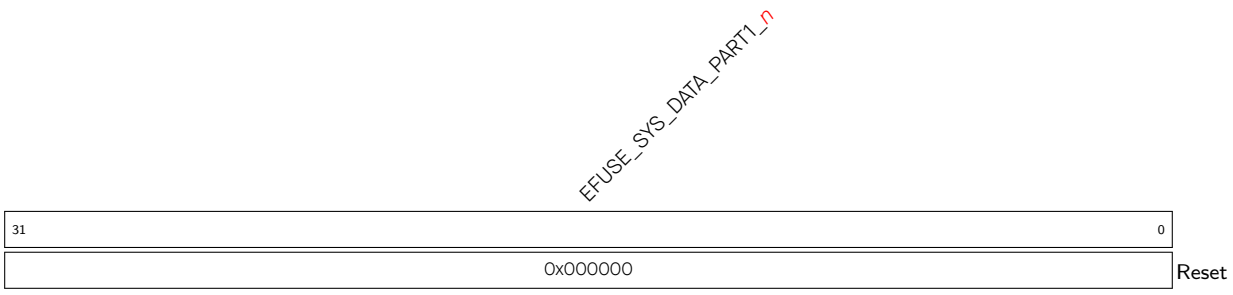
EFUSE_SYS_DATA_PART0_1 表示系统数据第 0 部分的第 1 个 32 位内容。(RO)

Register 7.14. EFUSE_RD_MAC_SYS5_REG (0x0058)



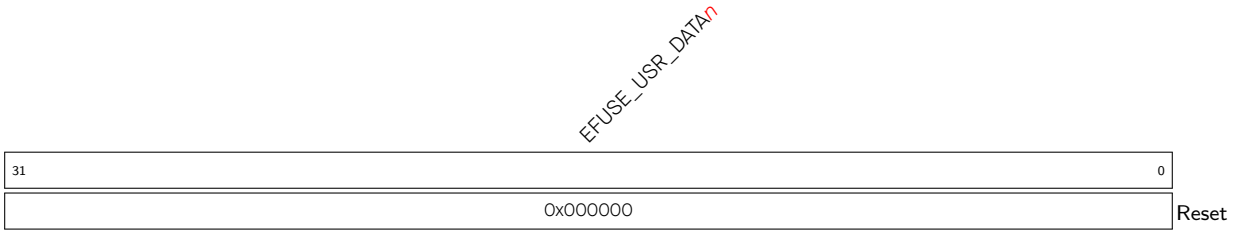
EFUSE_SYS_DATA_PART0_2 表示系统数据第 0 部分的第 2 个 32 位内容。(RO)

Register 7.15. EFUSE_RD_SYS_PART1_DATA*n*_REG (*n*: 0-7) (0x005C+0x4**n*)



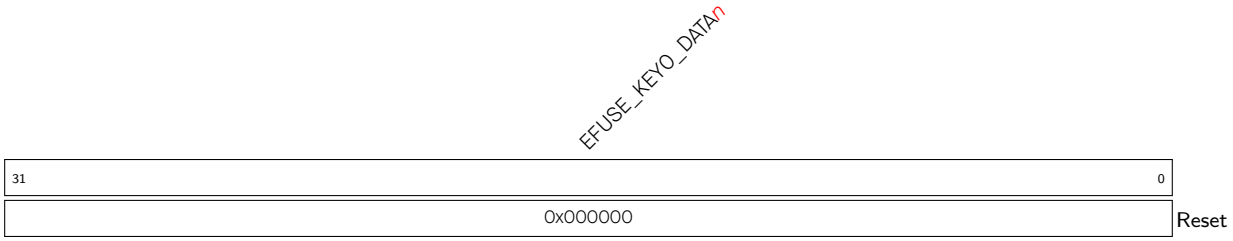
EFUSE_SYS_DATA_PART1_*n* 表示系统数据第 1 部分的第 *n* 个 32 位内容。(RO)

Register 7.16. EFUSE_RD_USR_DATA*n*_REG (*n*: 0-7) (0x007C+0x4*n*)



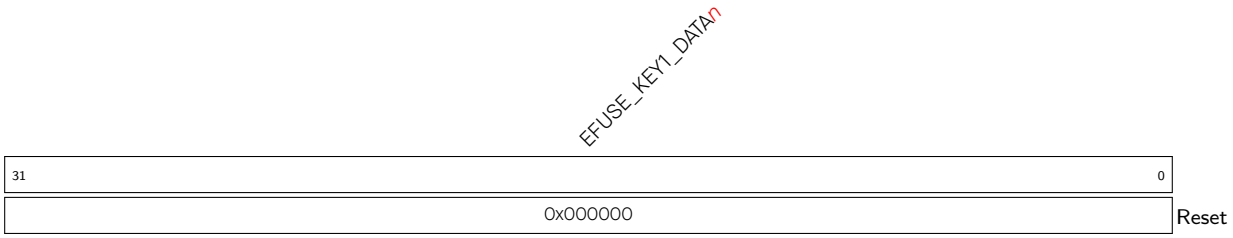
EFUSE_USR_DATA*n* 表示 BLOCK3 (user) 第 *n* 个 32 位内容。(RO)

Register 7.17. EFUSE_RD_KEY0_DATA*n*_REG (*n*: 0-7) (0x009C+0x4*n*)



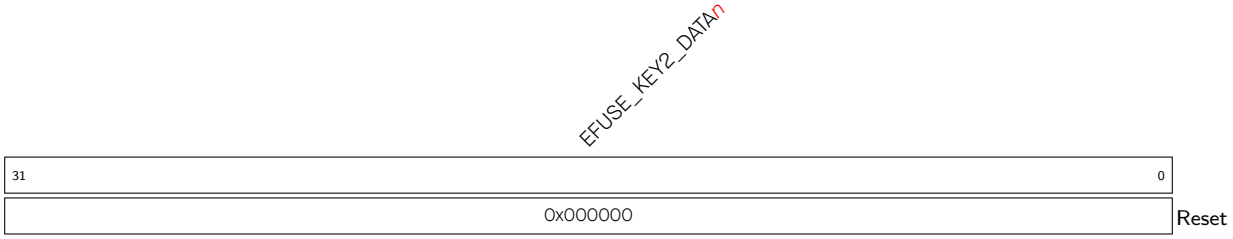
EFUSE_KEY0_DATA*n* 表示 key0 第 *n* 个 32 位内容。(RO)

Register 7.18. EFUSE_RD_KEY1_DATA*n*_REG (*n*: 0-7) (0x00BC+0x4*n*)



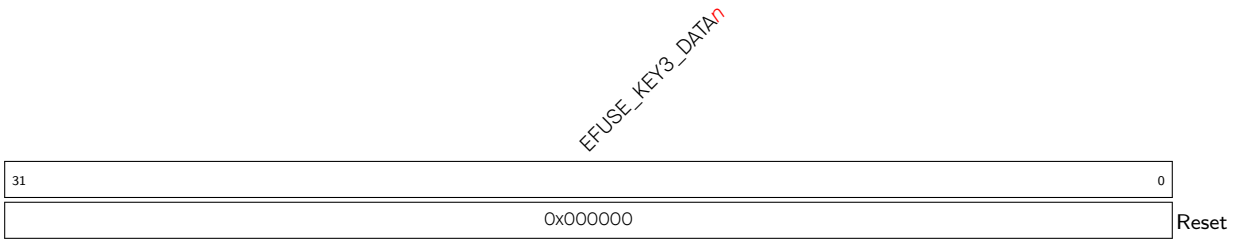
EFUSE_KEY1_DATA*n* 表示 key1 第 *n* 个 32 位内容。(RO)

Register 7.19. EFUSE_RD_KEY2_DATA*n*_REG (*n*: 0-7) (0x00DC+0x4*n*)



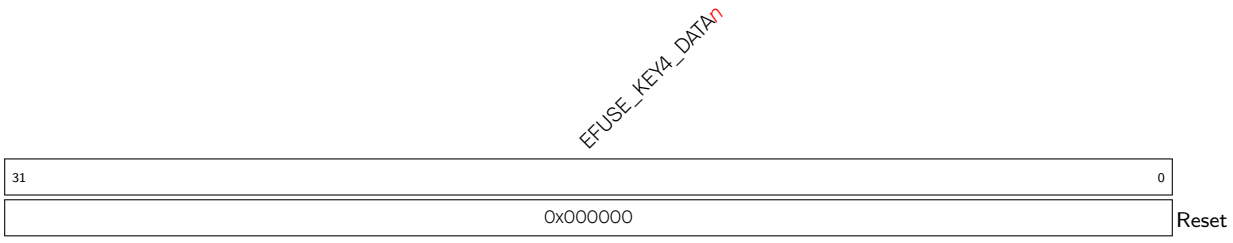
EFUSE_KEY2_DATA*n* 表示 key2 第 *n* 个 32 位内容。(RO)

Register 7.20. EFUSE_RD_KEY3_DATA*n*_REG (*n*: 0-7) (0x00FC+0x4**n*)



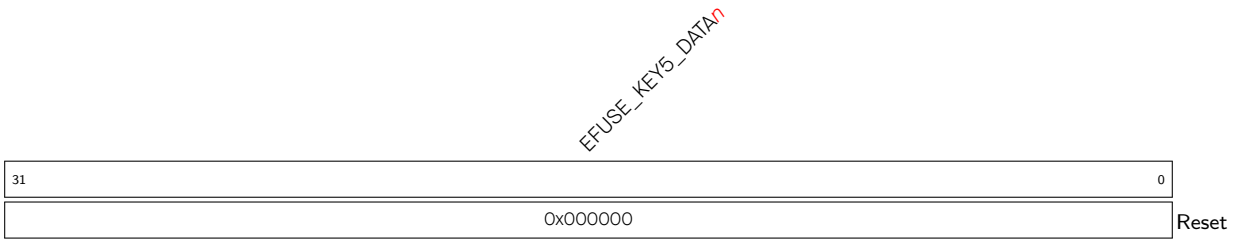
EFUSE_KEY3_DATA*n* 表示 key3 第 *n* 个 32 位内容。(RO)

Register 7.21. EFUSE_RD_KEY4_DATA*n*_REG (*n*: 0-7) (0x011C+0x4**n*)



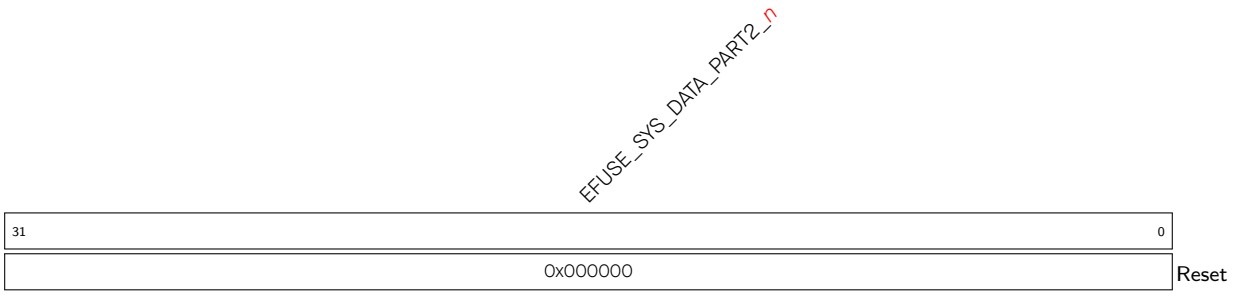
EFUSE_KEY4_DATA*n* 表示 key4 第 *n* 个 32 位内容。(RO)

Register 7.22. EFUSE_RD_KEY5_DATA*n*_REG (*n*: 0-7) (0x013C+0x4**n*)



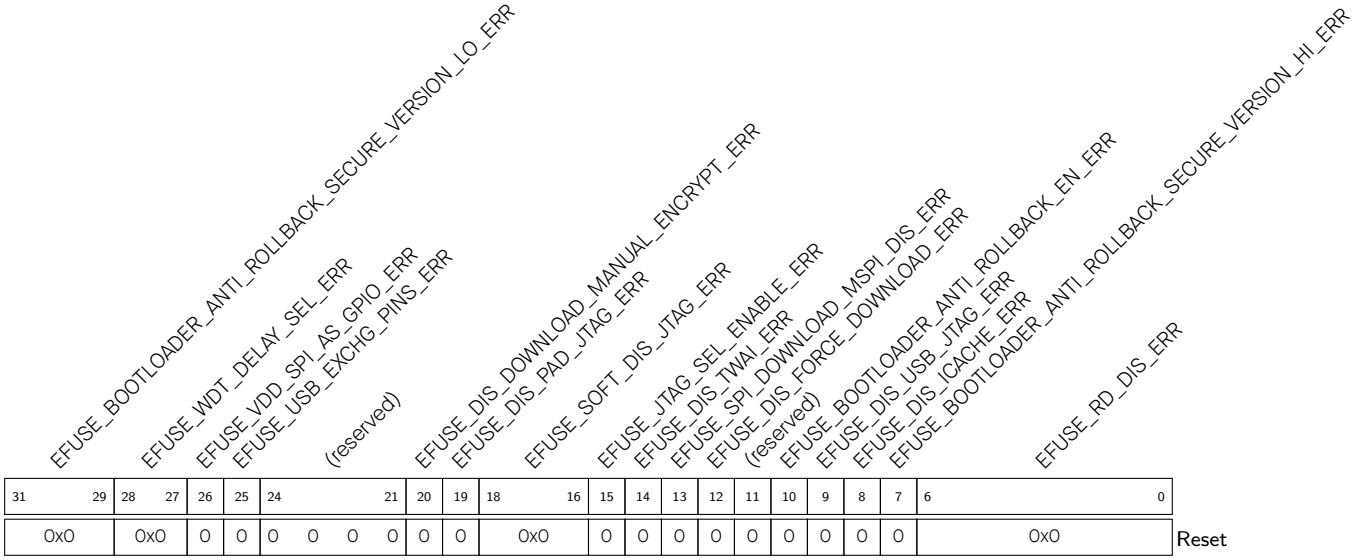
EFUSE_KEY5_DATA*n* 表示 key5 第 *n* 个 32 位内容。(RO)

Register 7.23. EFUSE_RD_SYS_PART2_DATA*n*_REG (*n*: 0-7) (0x015C+0x4**n*)



EFUSE_SYS_DATA_PART2_*n* 表示系统数据第 2 部分的第 *n* 个 32 位内容。(RO)

Register 7.24. EFUSE_RD_REPEAT_DATA_ERR0_REG (0x017C)



EFUSE_RD_DIS_ERR 若该域中的任何位为 1，则表示 RD_DIS 烧写错误。(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_VERSION_HI_ERR 表示 EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_VERSION_HI 烧写错误。(RO)

EFUSE_DIS_CACHE_ERR 若该位为 1，则表示 DIS_CACHE 烧写错误。(RO)

EFUSE_DIS_USB_JTAG_ERR 若该位为 1，则表示 DIS_USB_JTAG 烧写错误。(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_EN_ERR 表示 EFUSE_BOOTLOADER_ANTI_ROLLBACK_EN 烧写错误。(RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR 若该位为 1，则表示 DIS_FORCE_DOWNLOAD 烧写错误。(RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS_ERR 若该位为 1，则表示 SPI_DOWNLOAD_MSPI_DIS 烧写错误。(RO)

EFUSE_DIS_TWAI_ERR 若该位为 1，则表示 DIS_TWAI 烧写错误。(RO)

EFUSE_JTAG_SEL_ENABLE_ERR 若该位为 1，则表示 JTAG_SEL_ENABLE 烧写错误。(RO)

EFUSE_SOFT_DIS_JTAG_ERR 若该域中的任何位为 1，则表示 SOFT_DIS_JTAG 烧写错误。(RO)

EFUSE_DIS_PAD_JTAG_ERR 若该位为 1，则表示 DIS_PAD_JTAG 烧写错误。(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR 若该位为 1，则表示 DIS_DOWNLOAD_MANUAL_ENCRYPT 烧写错误。(RO)

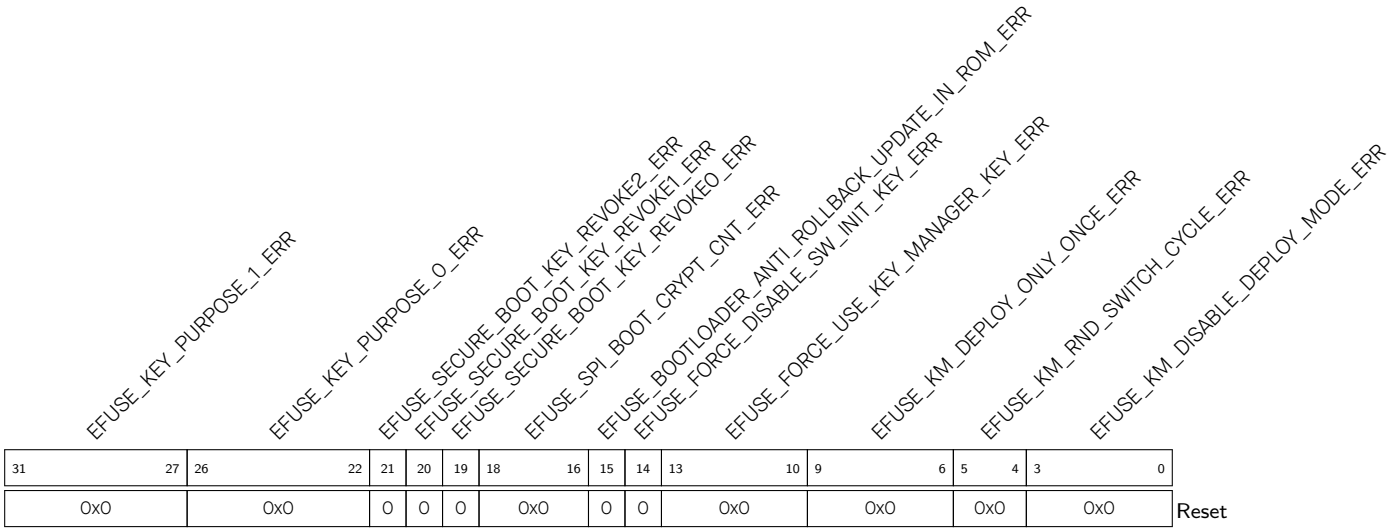
EFUSE_USB_EXCHG_PINS_ERR 若该位为 1，则表示 USB_EXCHG_PINS 烧写错误。(RO)

EFUSE_VDD_SPI_AS_GPIO_ERR 若该位为 1，则表示 VDD_SPI_AS_GPIO 烧写错误。(RO)

EFUSE_WDT_DELAY_SEL_ERR 表示 EFUSE_WDT_DELAY_SEL 的烧写错误。(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_VERSION_LO_ERR 表示 EFUSE_BOOTLOADER_ANTI_ROLLBACK_SECURE_VERSION_LO 的烧写错误。(RO)

Register 7.25. EFUSE_RD_REPEAT_DATA_ERR1_REG (0x0180)



EFUSE_KM_DISABLE_DEPLOY_MODE_ERR 表示 EFUSE_KM_DISABLE_DEPLOY_MODE 的烧写错误。(RO)

EFUSE_KM_RND_SWITCH_CYCLE_ERR 表示 EFUSE_KM_RND_SWITCH_CYCLE 的烧写错误。(RO)

EFUSE_KM_DEPLOY_ONLY_ONCE_ERR 表示 EFUSE_KM_DEPLOY_ONLY_ONCE 的烧写错误。(RO)

EFUSE_FORCE_USE_KEY_MANAGER_KEY_ERR 表示 EFUSE_FORCE_USE_KEY_MANAGER_KEY 的烧写错误。(RO)

EFUSE_FORCE_DISABLE_SW_INIT_KEY_ERR 表示 EFUSE_FORCE_DISABLE_SW_INIT_KEY 的烧写错误。(RO)

EFUSE_BOOTLOADER_ANTI_ROLLBACK_UPDATE_IN_ROM_ERR 表示 EFUSE_BOOTLOADER_ANTI_ROLLBACK_UPDATE_IN_ROM 的烧写错误。(RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR 若该域中的任何位为 1，则表示 SPI_BOOT_CRYPT_CNT 烧写错误。(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR 若该位为 1，则表示 SECURE_BOOT_KEY_REVOKE0 烧写错误。(RO)

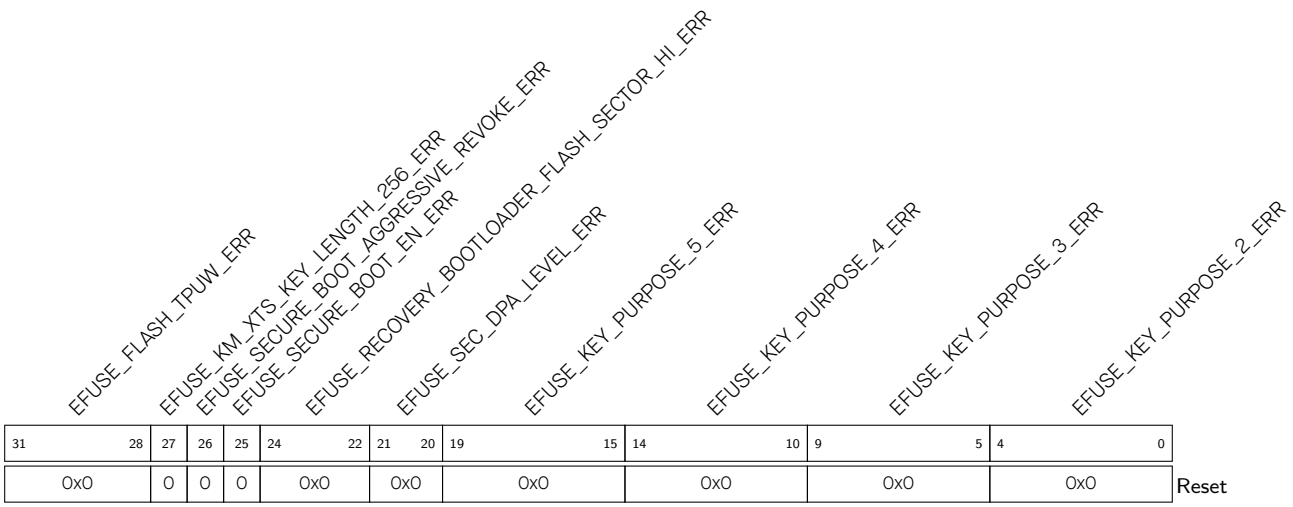
EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR 若该位为 1，则表示 SECURE_BOOT_KEY_REVOKE1 烧写错误。(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR 若该位为 1，则表示 SECURE_BOOT_KEY_REVOKE2 烧写错误。(RO)

EFUSE_KEY_PURPOSE_0_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_0 烧写错误。(RO)

EFUSE_KEY_PURPOSE_1_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_1 烧写错误。(RO)

Register 7.26. EFUSE_RD_REPEAT_DATA_ERR2_REG (0x0184)



EFUSE_KEY_PURPOSE_2_ERR 若该域中的任何位为 1，则表示 EFUSE_KEY_PURPOSE_2 烧写错误。(RO)

EFUSE_KEY_PURPOSE_3_ERR 若该域中的任何位为 1，则表示 EFUSE_KEY_PURPOSE_3 烧写错误。(RO)

EFUSE_KEY_PURPOSE_4_ERR 若该域中的任何位为 1，则表示 EFUSE_KEY_PURPOSE_4 烧写错误。(RO)

EFUSE_KEY_PURPOSE_5_ERR 若该域中的任何位为 1，则表示 EFUSE_KEY_PURPOSE_5 烧写错误。(RO)

EFUSE_SEC_DPA_LEVEL_ERR 若该域中的任何位为 1，则表示 EFUSE_SEC_DPA_LEVEL 烧写错误。(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_HI_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_HI 的烧写错误。(RO)

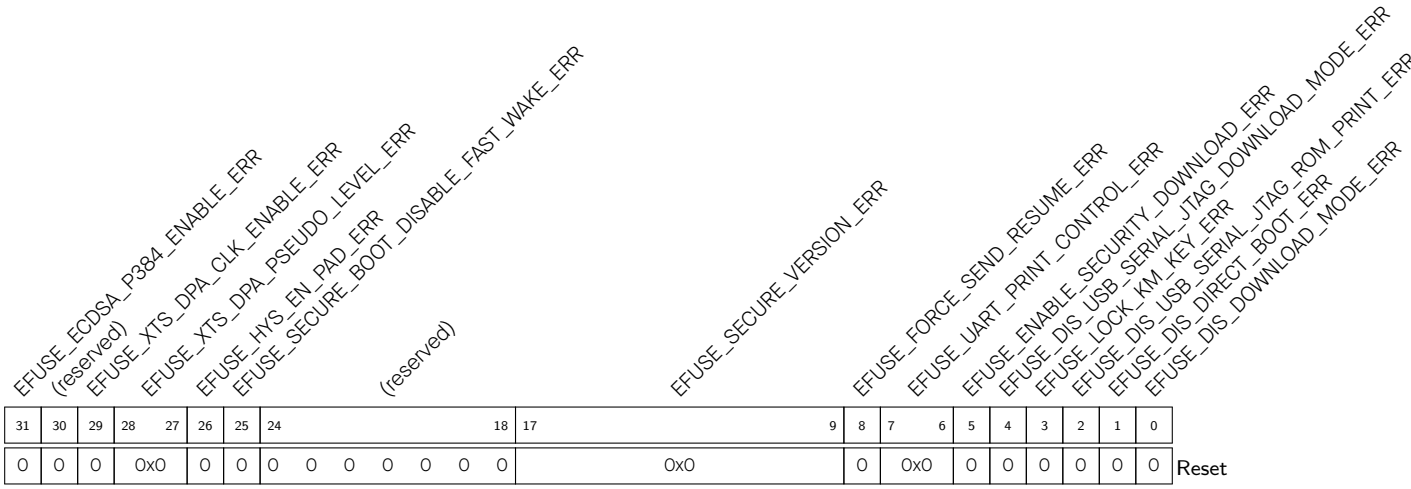
EFUSE_SECURE_BOOT_EN_ERR 该位为 1 表示 EFUSE_SECURE_BOOT_EN 的烧写错误。(RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR 若该位为 1，则表示 EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE 烧写错误。(RO)

EFUSE_KM_XTS_KEY_LENGTH_256_ERR 表示 EFUSE_KM_XTS_KEY_LENGTH_256 的烧写错误。(RO)

EFUSE_FLASH_TPUW_ERR 若该位为 1，则表示 EFUSE_FLASH_TPUW 烧写错误。(RO)

Register 7.27. EFUSE_RD_REPEAT_DATA_ERR3_REG (0x0188)



EFUSE_DIS_DOWNLOAD_MODE_ERR 若该位为 1，则表示 EFUSE_DIS_DOWNLOAD_MODE 烧写错误。(RO)

EFUSE_DIS_DIRECT_BOOT_ERR 若该位为 1，则表示 EFUSE_DIS_DIRECT_BOOT 烧写错误。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT_ERR 若该位为 1，则表示 EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT_ERR 烧写错误。(RO)

EFUSE_LOCK_KM_KEY_ERR 表示 EFUSE_LOCK_KM_KEY 的烧写错误。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE_ERR 若该位为 1，则表示 EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE 烧写错误。(RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR 若该位为 1，则表示 EFUSE_ENABLE_SECURITY_DOWNLOAD 烧写错误。(RO)

EFUSE_UART_PRINT_CONTROL_ERR 若该域中的任何位为 1，则表示 EFUSE_UART_PRINT_CONTROL 烧写错误。(RO)

EFUSE_FORCE_SEND_RESUME_ERR 若该位为 1，则表示 EFUSE_FORCE_SEND_RESUME 烧写错误。(RO)

EFUSE_SECURE_VERSION_ERR 若该域中的任何位为 1，则表示 EFUSE_SECURE_VERSION 烧写错误。(RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE_ERR 若该位为 1，则表示 EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE 烧写错误。(RO)

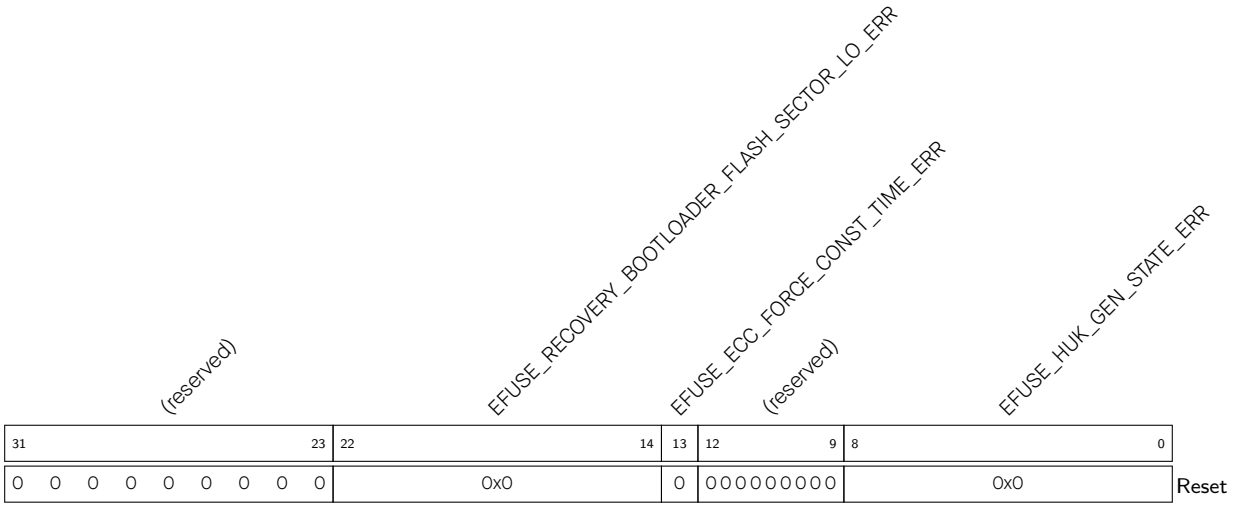
EFUSE_HYS_EN_PAD_ERR 若该位为 1，则表示 EFUSE_HYS_EN_PAD 烧写错误。(RO)

EFUSE_XTS_DPA_PSEUDO_LEVEL_ERR 若该域中的任何位为 1，则表示 EFUSE_XTS_DPA_PSEUDO_LEVEL 烧写错误。(RO)

EFUSE_XTS_DPA_CLK_ENABLE_ERR 若该位为 1，则表示 EFUSE_XTS_DPA_CLK_ENABLE 烧写错误。(RO)

EFUSE_ECDSA_P384_ENABLE_ERR 表示 EFUSE_ECDSA_P384_ENABLE 的烧写错误。(RO)

Register 7.28. EFUSE_RD_REPEAT_DATA_ERR4_REG (0x018C)

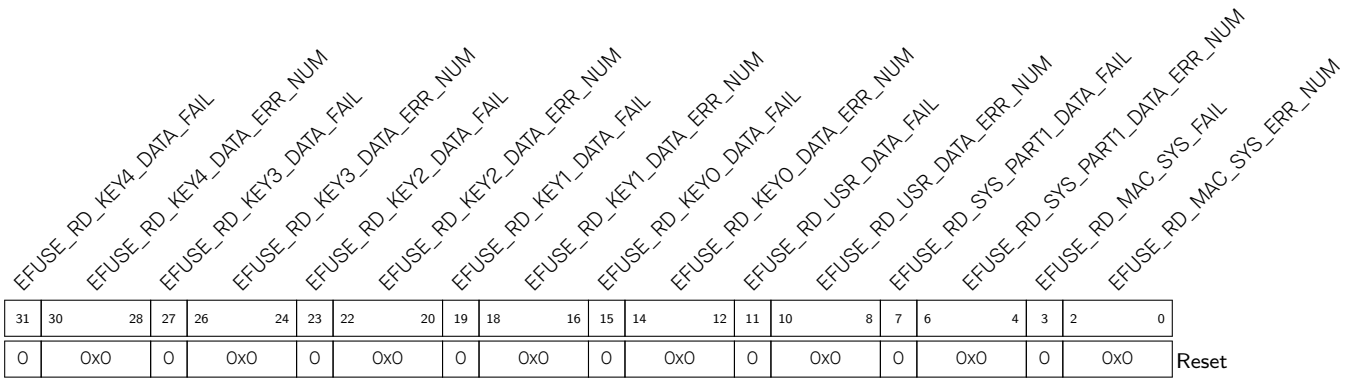


EFUSE_HUK_GEN_STATE_ERR 表示 EFUSE_HUK_GEN_STATE 的烧写错误。(RO)

EFUSE_ECC_FORCE_CONST_TIME_ERR 表示 EFUSE_ECC_FORCE_CONST_TIME 的烧写错误。(RO)

EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_LO_ERR 表示 EFUSE_RECOVERY_BOOTLOADER_FLASH_SECTOR_LO 的烧写错误。(RO)

Register 7.29. EFUSE_RD_RS_DATA_ERR0_REG (0x0190)



EFUSE_RD_MAC_SYS_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_MAC_SYS_FAIL 表示 MAC_SYS 的数据是否烧写失败。

0：无烧写错误，MAC_SYS 的数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_SYS_PART1_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_SYS_PART1_DATA_FAIL 表示第 1 部分的系统数据是否烧写失败。

0：无烧写错误，第 1 部分的系统数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_USR_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_USR_DATA_FAIL 表示用户数据烧写是否失败。

0：无烧写错误，用户数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_KEY0_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_KEY0_DATA_FAIL 表示 key0 数据是否烧写失败。

0：无烧写错误，key0 数据是可靠的

1：key0 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_KEY1_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_KEY1_DATA_FAIL 表示 key1 数据是否烧写失败。

0：无烧写错误，key1 数据是可靠的

1：key1 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_KEY2_DATA_ERR_NUM 表示错误字节数。(RO)

见下页……

Register 7.29. EFUSE_RD_RS_DATA_ERR0_REG (0x0190)

接上页……

EFUSE_RD_KEY2_DATA_FAIL 表示 key2 数据是否烧写失败。

0：无烧写错误，key2 数据是可靠的

1：key2 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_KEY3_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_KEY3_DATA_FAIL 表示 key3 数据是否烧写失败。

0：无烧写错误，key3 数据是可靠的

1：key3 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_KEY4_DATA_ERR_NUM 表示错误字节数。(RO)

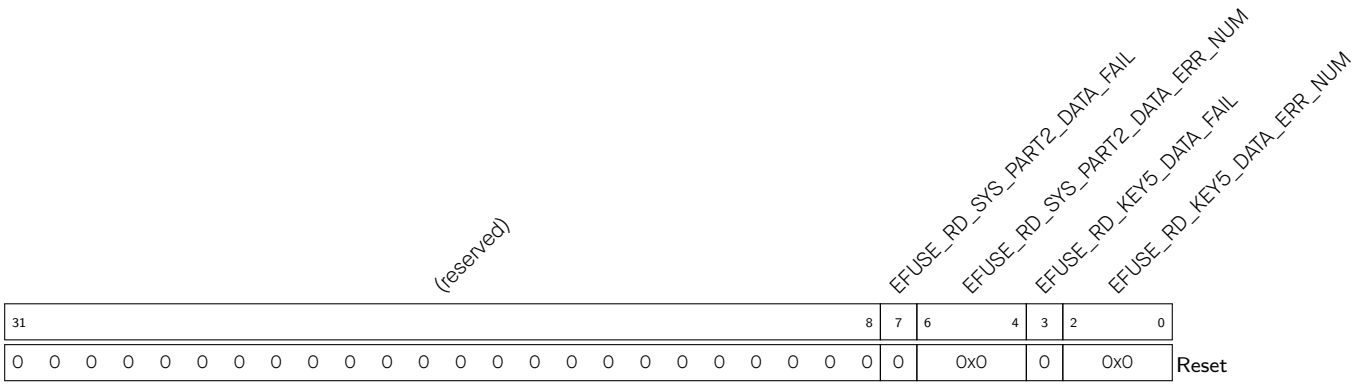
EFUSE_RD_KEY4_DATA_FAIL 表示 key4 数据是否烧写失败。

0：无烧写错误，key4 数据是可靠的

1：key4 数据烧写失败，错误字节数超过 6

(RO)

Register 7.30. EFUSE_RD_RS_DATA_ERR1_REG (0x0194)



EFUSE_RD_KEY5_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_RD_KEY5_DATA_FAIL 表示 key5 数据是否烧写失败。

0：无烧写错误，key5 数据是可靠的

1：key5 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_RD_SYS_PART2_DATA_ERR_NUM 表示错误字节数。(RO)

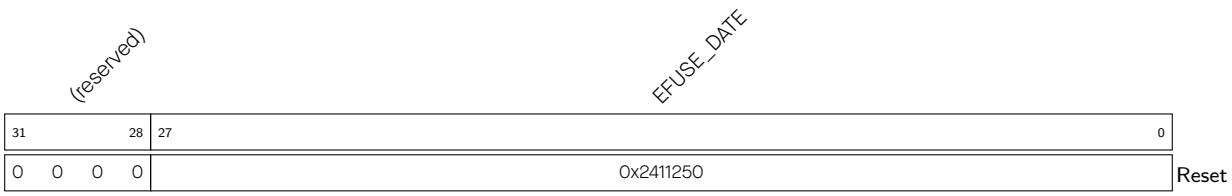
EFUSE_RD_SYS_PART2_DATA_FAIL 表示第 2 部分的系统数据是否烧写失败。

0：无烧写错误，第 2 部分的系统数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

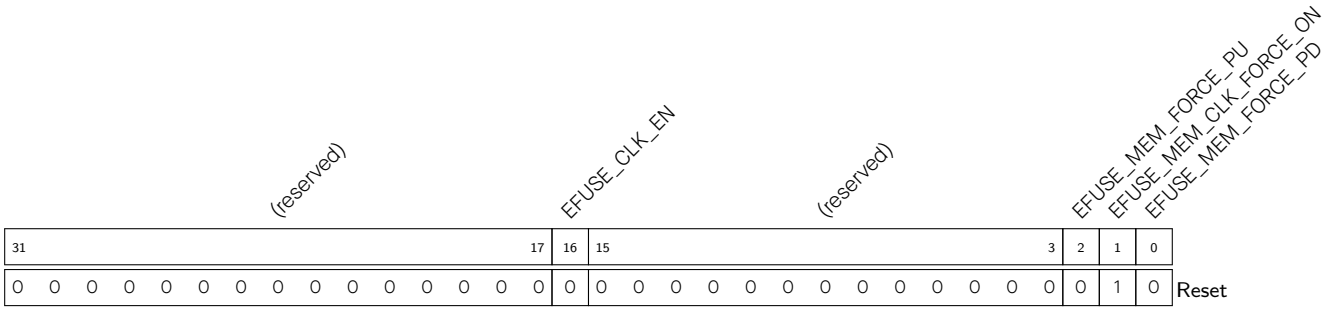
(RO)

Register 7.31. EFUSE_DATE_REG (0x0198)



EFUSE_DATE 版本控制寄存器。(R/W)

Register 7.32. EFUSE_CLK_REG (0x01C8)



EFUSE_MEM_FORCE_PD 配置是否强制 eFuse SRAM 下电。

1：强制

0：没有作用

(R/W)

EFUSE_MEM_CLK_FORCE_ON 配置是否强制激活 eFuse SRAM 时钟信号。

1：强制激活

0：没有作用

(R/W)

EFUSE_MEM_FORCE_PU 配置是否强制 eFuse SRAM 上电。

1：强制

0：没有作用

(R/W)

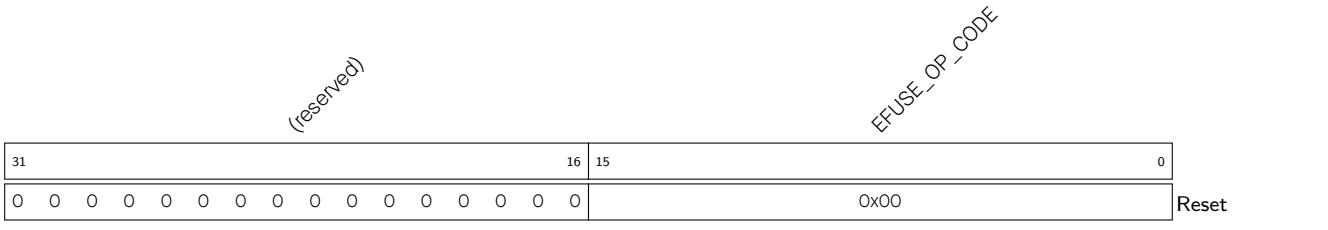
EFUSE_CLK_EN 配置是否强制使能 eFuse 寄存器配置时钟信号。

1：强制使能

0：时钟仅在读写寄存器时开启

(R/W)

Register 7.33. EFUSE_CONF_REG (0x01CC)



EFUSE_OP_CODE 配置操作指令类型。

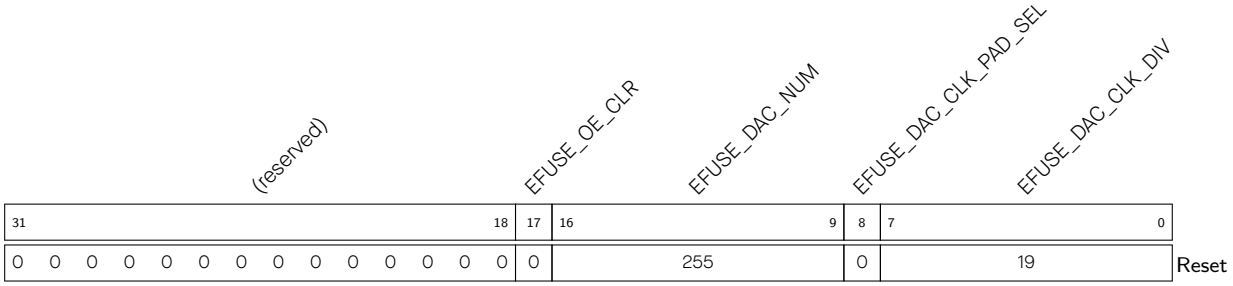
0x5A5A：烧写操作指令

0x5AA5：读操作指令

其他值：没有作用

(R/W)

Register 7.34. EFUSE_DAC_CONF_REG (0x01EC)



EFUSE_DAC_CLK_DIV 配置烧写电压的爬升时钟分频系数。(R/W)

EFUSE_DAC_NUM 配置烧写电压上升所需的时钟周期数。单位：一个时钟周期除以 EFUSE_DAC_CLK_DIV。(R/W)

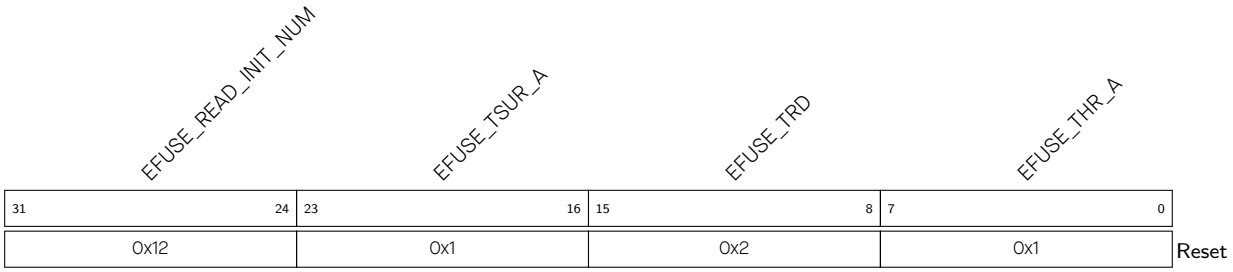
EFUSE_OE_CLR 配置是否降低烧写电压的供电能力。

0：不降低

1：降低

(R/W)

Register 7.35. EFUSE_RD_TIM_CONF_REG (0x01F0)



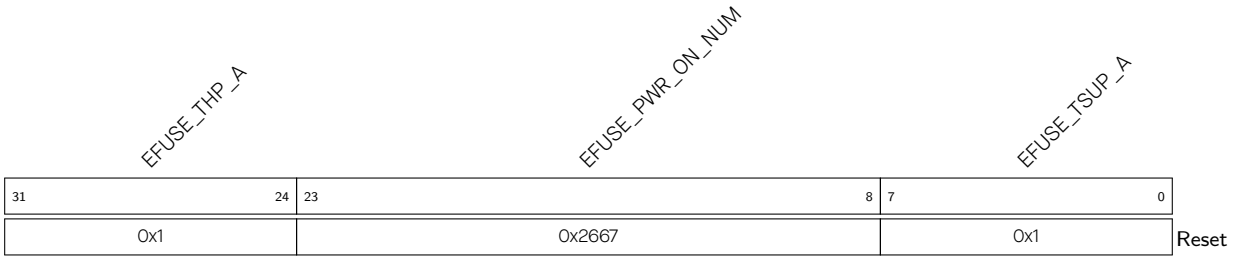
EFUSE_THR_A 配置读操作的保持时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_TRD 配置读操作的时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_TSUR_A 配置读操作的设置时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_READ_INIT_NUM 配置读取 eFuse 存储器的等待时间，单位：一个 eFuse 核心时钟周期。(R/W)

Register 7.36. EFUSE_WR_TIM_CONF1_REG (0x01F4)

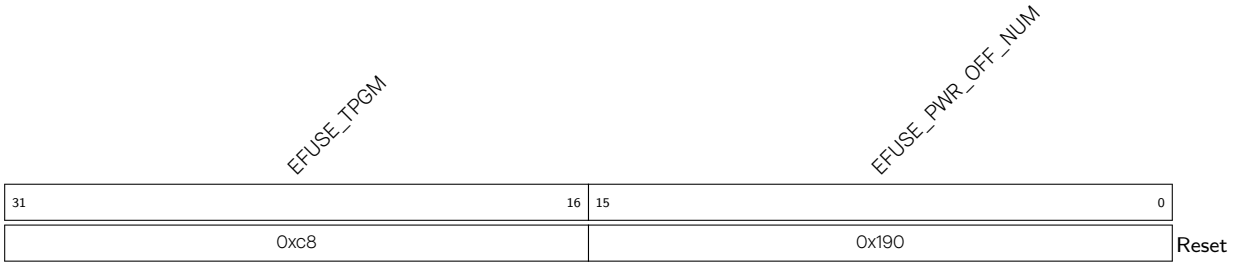


EFUSE_TSUP_A 配置烧写的设置时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_PWR_ON_NUM 配置 VDDQ 的上电时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_THP_A 配置烧写的保持时间，单位：一个 eFuse 核心时钟周期。(R/W)

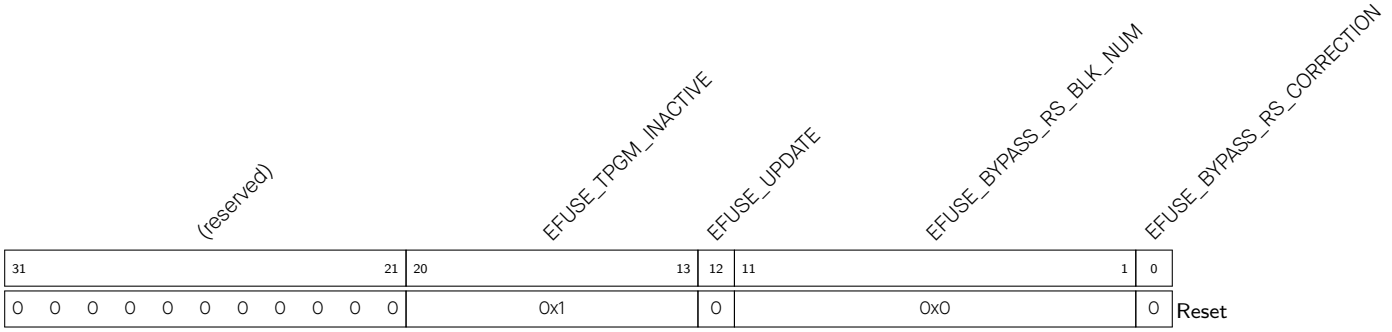
Register 7.37. EFUSE_WR_TIM_CONF2_REG (0x01F4)



EFUSE_PWR_OFF_NUM 配置 VDDQ 的掉电时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_TPGM 配置活跃状态的烧写时间，单位：一个 eFuse 核心时钟周期。(R/W)

Register 7.38. EFUSE_WR_TIM_CONF0_RS_BYPASS_REG (0x01FC)



EFUSE_BYPASS_RS_CORRECTION 配置是否跳过 Reed-Solomon (RS) 修正步骤。

0：不跳过

1：跳过

(R/W)

EFUSE_BYPASS_RS_BLK_NUM 配置需跳过 RS 自动修正步骤的 block 编号。(R/W)

EFUSE_UPDATE 配置是否更新多位寄存器信号。

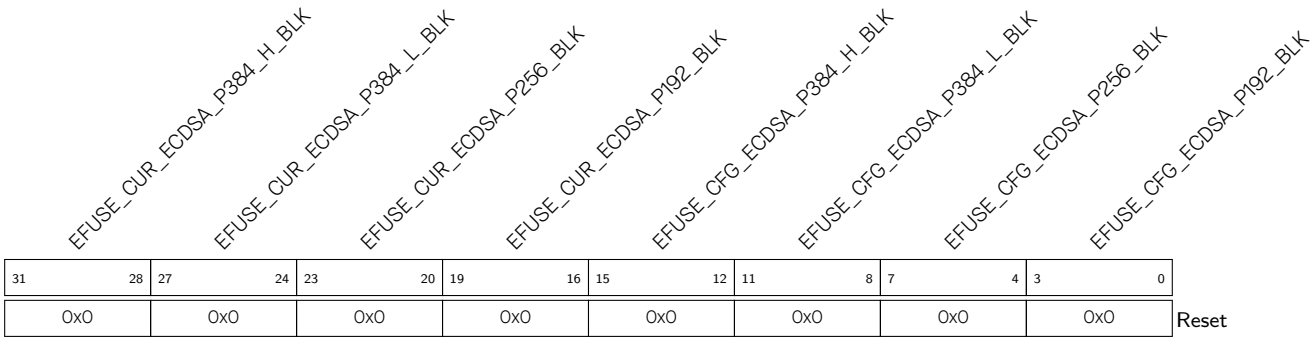
1：更新

0：没有作用

(WT)

EFUSE_TPGM_INACTIVE 配置非活跃状态的烧写时间，单位：一个 eFuse 核心时钟周期。(R/W)

Register 7.39. EFUSE_ECDSA_REG (0x01D0)



EFUSE_CFG_ECDSA_P192_BLK 配置用于 ECDSA_DS P192 密钥输出的块。(R/W)

EFUSE_CFG_ECDSA_P256_BLK 配置用于 ECDSA_DS P256 密钥输出的块。(R/W)

EFUSE_CFG_ECDSA_P384_L_BLK 配置用于 ECDSA_DS P384 密钥低部分输出的块。(R/W)

EFUSE_CFG_ECDSA_P384_H_BLK 配置用于 ECDSA_DS P384 密钥高部分输出的块。(R/W)

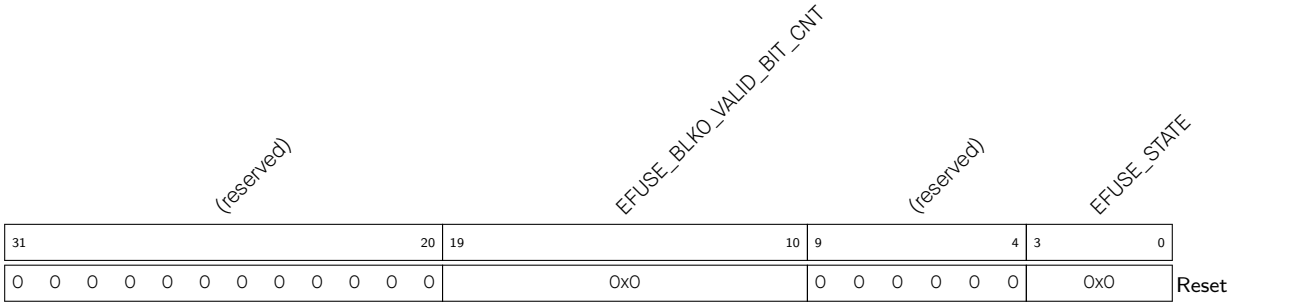
EFUSE_CUR_ECDSA_P192_BLK 表示当前用于 ECDSA_DS P192 密钥输出的块。(RO)

EFUSE_CUR_ECDSA_P256_BLK 表示当前用于 ECDSA_DS P256 密钥输出的块。(RO)

EFUSE_CUR_ECDSA_P384_L_BLK 表示当前用于 ECDSA_DS P384 密钥低部分输出的块。(RO)

EFUSE_CUR_ECDSA_P384_H_BLK 表示当前用于 ECDSA_DS P384 密钥高部分输出的块。(RO)

Register 7.40. EFUSE_STATUS_REG (0x01D4)



EFUSE_STATE 表示 eFuse 状态机的当前状态。

0：复位状态，上电后的初始状态

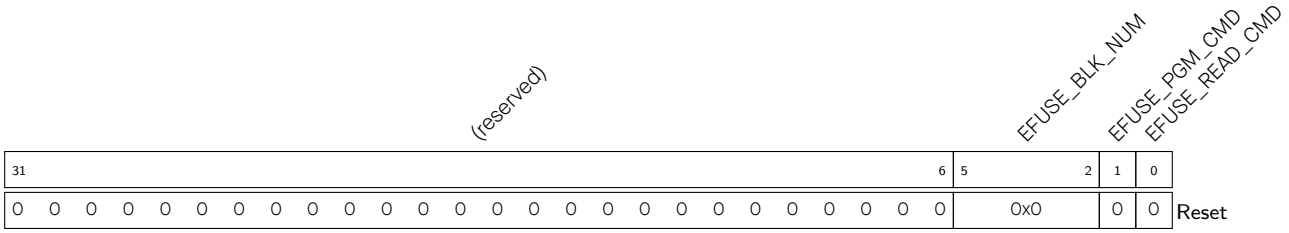
1：空闲状态

其他值：非空闲状态

(RO)

EFUSE_BLK0_VALID_BIT_CNT 表示块 0 中有效位的数量。(RO)

Register 7.41. EFUSE_CMD_REG (0x01D8)



EFUSE_READ_CMD 配置是否发送读指令。

1：发送

0：没有作用

(R/W/SC)

EFUSE_PGM_CMD 配置是否发送烧写指令。

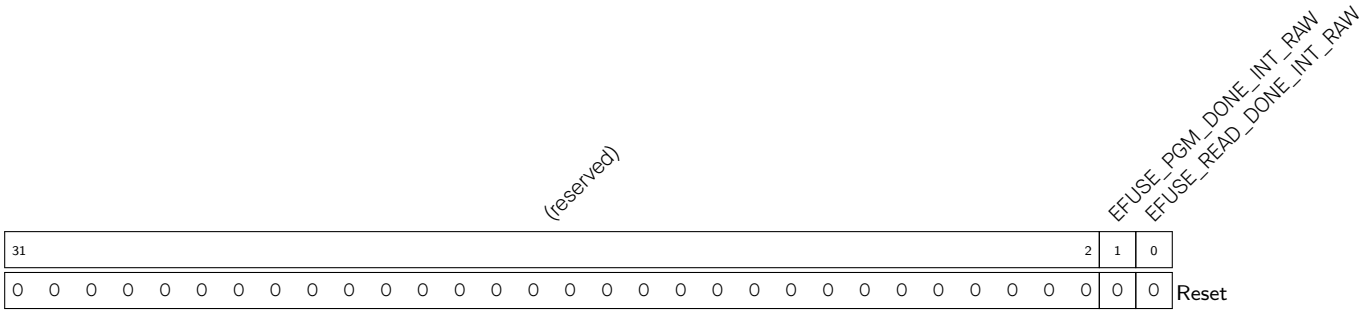
1：发送

0：没有作用

(R/W/SC)

EFUSE_BLK_NUM 配置待烧写的 block 编号。值 0 ~ 10 分别对应 BLOCK0 ~ BLOCK10。(R/W)

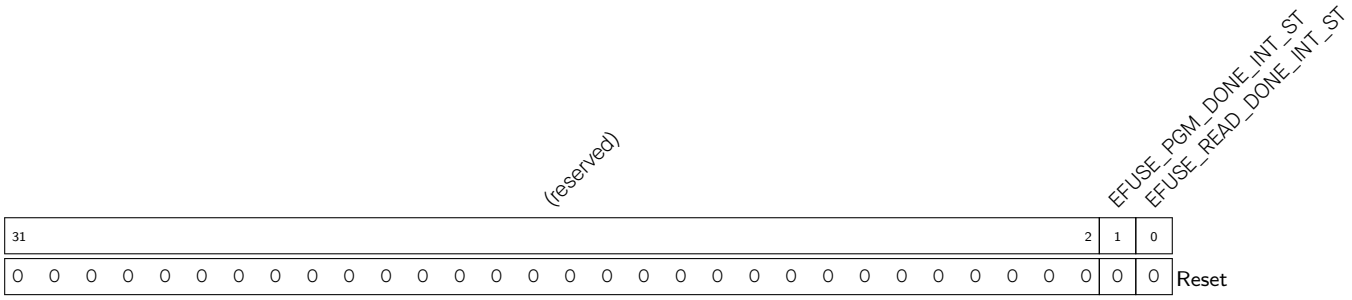
Register 7.42. EFUSE_INT_RAW_REG (0x01DC)



EFUSE_READ_DONE_INT_RAW EFUSE_READ_DONE_INT 的原始中断状态。(R/SS/WTC)

EFUSE_PGM_DONE_INT_RAW EFUSE_PGM_DONE_INT 的原始中断状态。(R/SS/WTC)

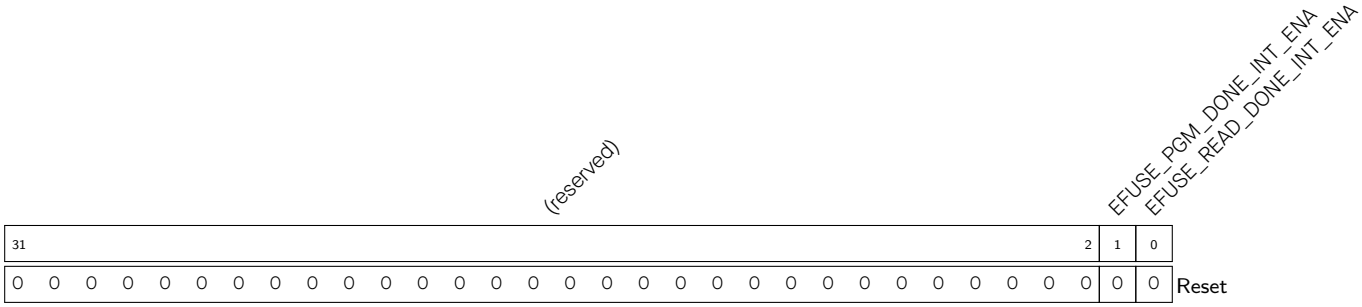
Register 7.43. EFUSE_INT_ST_REG (0x01E0)



EFUSE_READ_DONE_INT_ST EFUSE_READ_DONE_INT 的屏蔽中断状态。(RO)

EFUSE_PGM_DONE_INT_ST EFUSE_PGM_DONE_INT 的屏蔽中断状态。(RO)

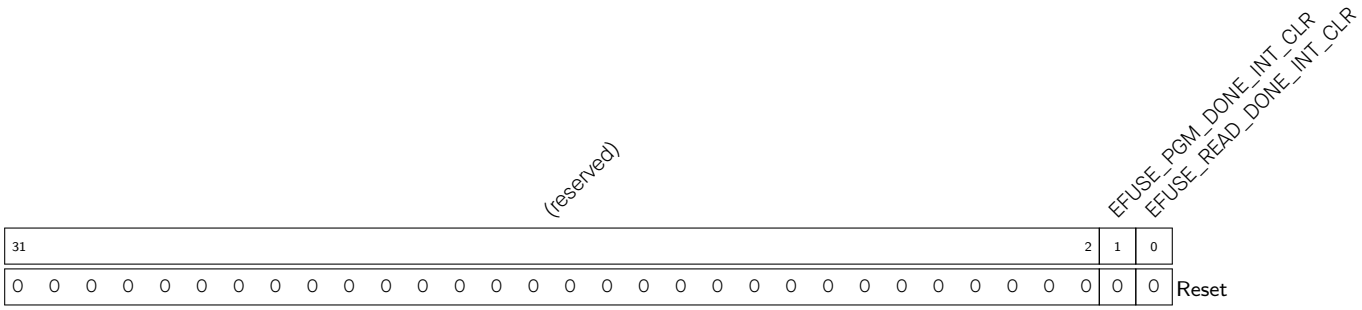
Register 7.44. EFUSE_INT_ENA_REG (0x01E4)



EFUSE_READ_DONE_INT_ENA 写 1 使能 EFUSE_READ_DONE_INT。(R/W)

EFUSE_PGM_DONE_INT_ENA 写 1 使能 EFUSE_PGM_DONE_INT。(R/W)

Register 7.45. EFUSE_INT_CLR_REG (0x01E8)



EFUSE_READ_DONE_INT_CLR 写 1 清除 EFUSE_READ_DONE_INT。(WT)

EFUSE_PGM_DONE_INT_CLR 写 1 清除 EFUSE_PGM_DONE_INT。(WT)

第 III 卷

系统组件

该部分介绍芯片的系统级功能，包括系统启动、时钟、GPIO、定时器、看门狗、调试辅助、事件和中断处理、低功耗管理以及各种系统寄存器。

第 8 章

GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)

8.1 概述

GPIO 交换矩阵是一种硬件结构，允许芯片上的 GPIO 管脚与外设输入/输出信号之间进行动态的重新映射和交换，以提供更大的灵活性。IO MUX（输入/输出多路复用）是一种技术，允许同一管脚在不同时间承担不同的功能，通过配置实现对管脚功能的动态切换。

ESP32-C5 提供两组 GPIO 交换矩阵和 IO MUX：

- HP GPIO 交换矩阵和 HP IO MUX
- LP GPIO 交换矩阵和 LP IO MUX

ESP32-C5 共有 29 个 GPIO 管脚，编号为 GPIO0~GPIO28，其中包括：

- 7 个 LP GPIO 管脚 (GPIO0~GPIO6)，可由 HP 或 LP 外设使用。
- 14 个 HP GPIO 管脚（GPIO7~GPIO14 和 GPIO23~GPIO28），只能由 HP 外设使用。
- 8 个外接 flash 或 PSRAM 专用的 GPIO 管脚 (GPIO15~GPIO22)。**这些专用管脚不可用于其他用途，因此不在后续章节进行描述**。如需了解专用管脚的更多信息，请参考《*ESP32-C5 技术规格书*》> 章节 *芯片与 flash 的管脚对应关系*。

每个管脚都可用作一个通用 IO，或连接一个内部的外设信号。这些模块共同组成了芯片的 GPIO 控制。

- 利用 HP GPIO 交换矩阵和 HP IO MUX，可配置 HP 外设模块的输入信号来源于**任何的 GPIO 管脚**，并且 HP 外设模块的输出信号也可连接到**任意 GPIO 管脚**。
- 利用 LP GPIO 交换矩阵和 LP IO MUX，可配置 LP 外设模块的输入信号来源于**任何的 LP GPIO 管脚**，并且 LP 外设模块的输出信号也可连接到**任意 LP GPIO 管脚**。

除非另有说明，本章节所述的 GPIO 交换矩阵同时包括 HP GPIO 交换矩阵和 LP GPIO 交换矩阵；IO MUX 同时包括 HP IO MUX 和 LP IO MUX。

8.2 主要特性

8.2.1 HP GPIO 交换矩阵和 HP IO MUX

HP GPIO 交换矩阵具有以下特性：

- HP 外设输入输出信号和 GPIO 管脚之间的全交换矩阵
- 76 个 HP 外设输入信号可以选择任意一个 GPIO 管脚的输入信号
- 每个 GPIO 管脚的输出信号可以来自 77 个 HP 外设输出信号的任意一个
- HP 外设输入信号经 GPIO SYNC 模块同步至 HP IO MUX 运行时钟

- 支持 GPIO 滤波器对输入信号进行滤波
- 支持毛刺滤波器对输入信号进行二次滤波
- 支持 Sigma Delta 调制输出 (SDM)
- 支持 GPIO 简单输入输出
- 支持 HP GPIO 唤醒

HP IO MUX 具有以下特性：

- 控制 21 个 GPIO (GPIO0~GPIO14、GPIO23~GPIO28) 供 HP 外设使用
- 为每个 GPIO 管脚提供一个寄存器 IO_MUX_GPIO*n*_REG，用于控制管脚的输入/输出、上拉/下拉、驱动强度、功能选择等配置
- 支持高频信号如 SPI 直接通过 HP IO MUX 输入和输出外设，实现更好的高频数字特性

8.2.2 LP GPIO 交换矩阵和 LP IO MUX

LP GPIO 交换矩阵具有如下特性：

- 支持 GPIO 简单输入输出
- 支持 LP GPIO 唤醒

LP IO MUX 具有如下特性：

- 7 个 LP GPIO 管脚 (GPIO0~GPIO6) 供 LP 外设使用
- 为每个 LP GPIO 管脚提供一个 LP_IO_MUX_GPIO*n*_REG 寄存器，用于控制管脚的输入/输出、上拉/下拉、驱动强度、功能选择等配置
- 支持使用 LP_AON_GPIO_MUX_SEL 来选择管脚控制来自 HP IO MUX 还是 LP IO MUX

8.3 结构概览

图 8.3-1 所示为 HP GPIO 交换矩阵、HP IO MUX、LP GPIO 交换矩阵和 LP IO MUX 将信号引入外设和引出至管脚的具体过程。

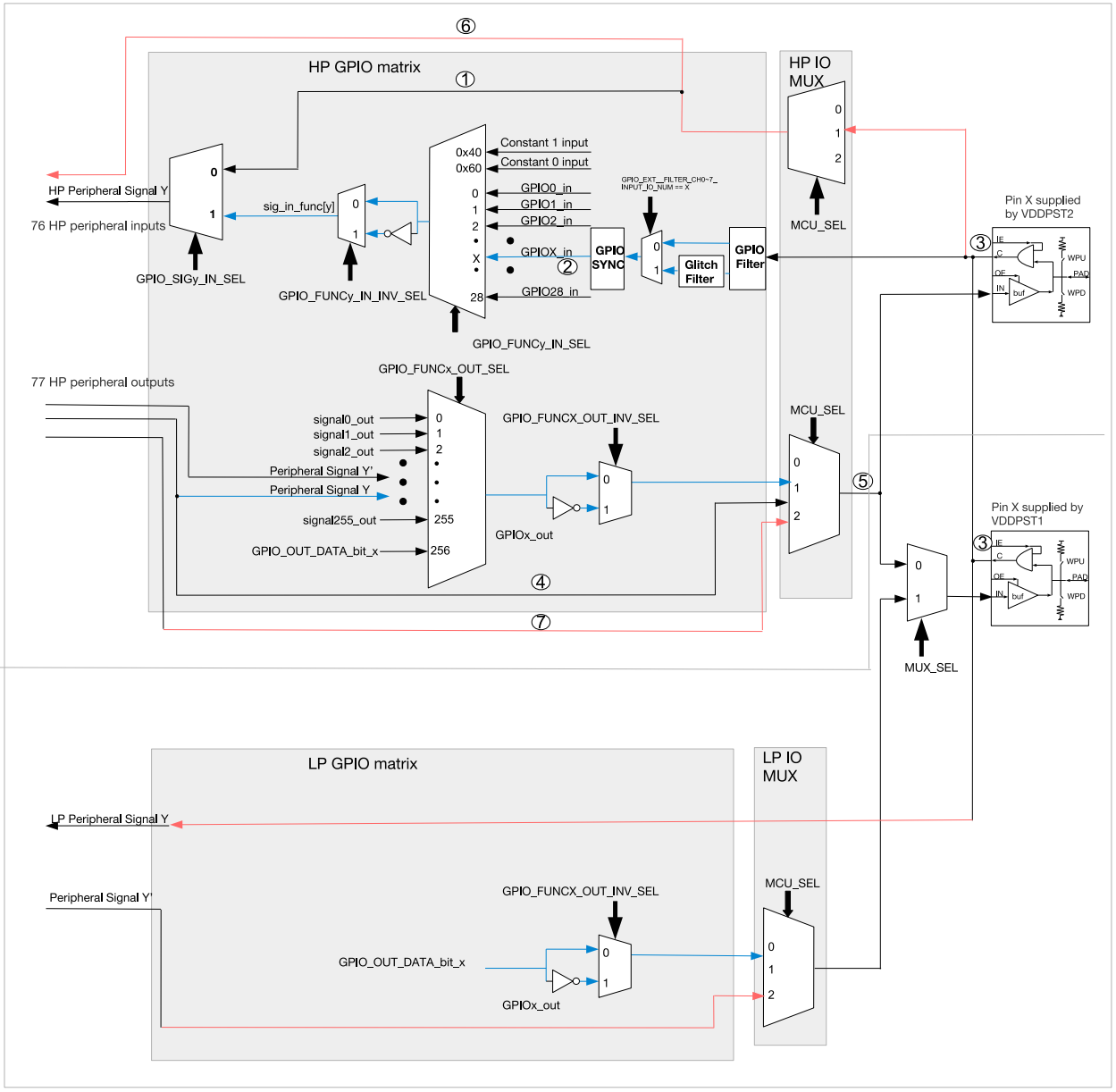


图 8.3-1. HP GPIO 交换矩阵、HP IO MUX、LP GPIO 交换矩阵和 LP IO MUX 框图

以下各点内容是对上图数字标注区域的解释说明，以 HP 系统为例：

① 部分输入信号既可以通过 HP GPIO 交换矩阵连接外设又可以直接通过 HP IO MUX 直连外设，这些输入信号在表 8.12-1 “信号可经由 IO MUX 直接输入”一栏中被标为“yes”。剩余其他信号只能通过 HP GPIO 交换矩阵连接至外设。

② ESP32-C5 共有 21 个用户可用的 GPIO 管脚，因此从 GPIO SYNC 进入到 HP GPIO 交换矩阵的输入共有 21 个。

③ 管脚由 IE、OE、WPU 和 WPD 信号控制。

④ 部分输出信号既可以通过 HP GPIO 交换矩阵连接外设又可通过 HP IO MUX 直连管脚，这些输出信号在表 8.12-1 “信号可经由 IO MUX 直接输出”一栏中被标为“yes”。剩余其他信号只能通过 HP GPIO 交换矩阵连接至管脚。

⑤ 从 HP GPIO 交换矩阵到 HP IO MUX 的输出共有 21 个，对应 GPIO *X*：0~14、23~28。

⑥ 仅通过 HP IO MUX 的外设输入。

⑦ 仅通过 HP IO MUX 的外设输出。

图 8.3-2 展示了芯片焊盘 (PAD) 的内部结构，即芯片逻辑与 GPIO 管脚之间的电气接口。ESP32-C5 所有的 GPIO 管脚均采用这一结构，且由 IE、OE、WPU 和 WPD 信号控制。信号配置请参考 IO_MUX_GPIO*x*_REG 寄存器或 LP_IO_MUX_GPIO*x*_REG 寄存器。

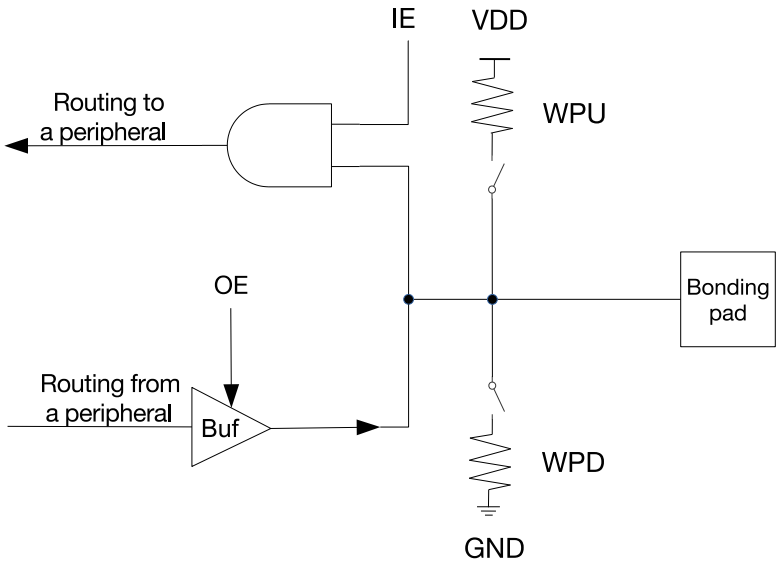


图 8.3-2. 焊盘内部结构

- IE：输入使能
- OE：输出使能
- WPU：内部弱上拉电阻
- WPD：内部弱下拉电阻
- Bonding pad：接合焊盘，芯片逻辑的结点，实现芯片封装内晶片与 GPIO 管脚之间的物理连接。

8.4 通过 GPIO 交换矩阵的外设输入

8.4.1 概述

通过 HP GPIO 交换矩阵接收外部输入信号，一般步骤如下：

- 配置 HP GPIO 交换矩阵从 21 个 GPIO (0~14、23~28) 中获取外部输入信号，见表 8.12-1。
- 配置外设输入选择通过 HP GPIO 交换矩阵接收输入信号。
- 配置 GPIO 管脚由 HP IO MUX 控制。

具体配置流程，可参考图 8.3-1 和章节 8.4.7。

如图 8.3-1 所示，使用 GPIO 交换矩阵将信号从管脚输入时，所有外部输入信号均来源于 GPIO 管脚，然后经 GPIO 滤波器进行滤波，见章节 8.4.7 中的步骤 2。

毛刺滤波器仅在 HP GPIO 交换矩阵中可用。毛刺滤波器硬件可从 GPIO 滤波器输出的信号中选择八个信号进行滤波，其他没被选中的信号则直接进入 GPIO SYNC 硬件，见章节 8.4.7 中的步骤 3。

所有信号经 GPIO 滤波器硬件或毛刺滤波器硬件滤波后，均将被 GPIO SYNC 硬件同步至 IO MUX 运行时钟，然后进入 GPIO 交换矩阵，见章节 8.4.2。外部输入信号也可经 IO MUX 直接输入至外设，但信号无法进行滤波及时钟同步操作。

8.4.2 信号同步

HP GPIO 交换矩阵支持信号同步功能。GPIO SYNC 模块的功能如图 8.4-1 所示。其中，negative sync 为 GPIO 输入经过 HP IO MUX 运行时钟的下降沿同步，positive sync 为 GPIO 输入经过 HP IO MUX 运行时钟的上升沿同步。

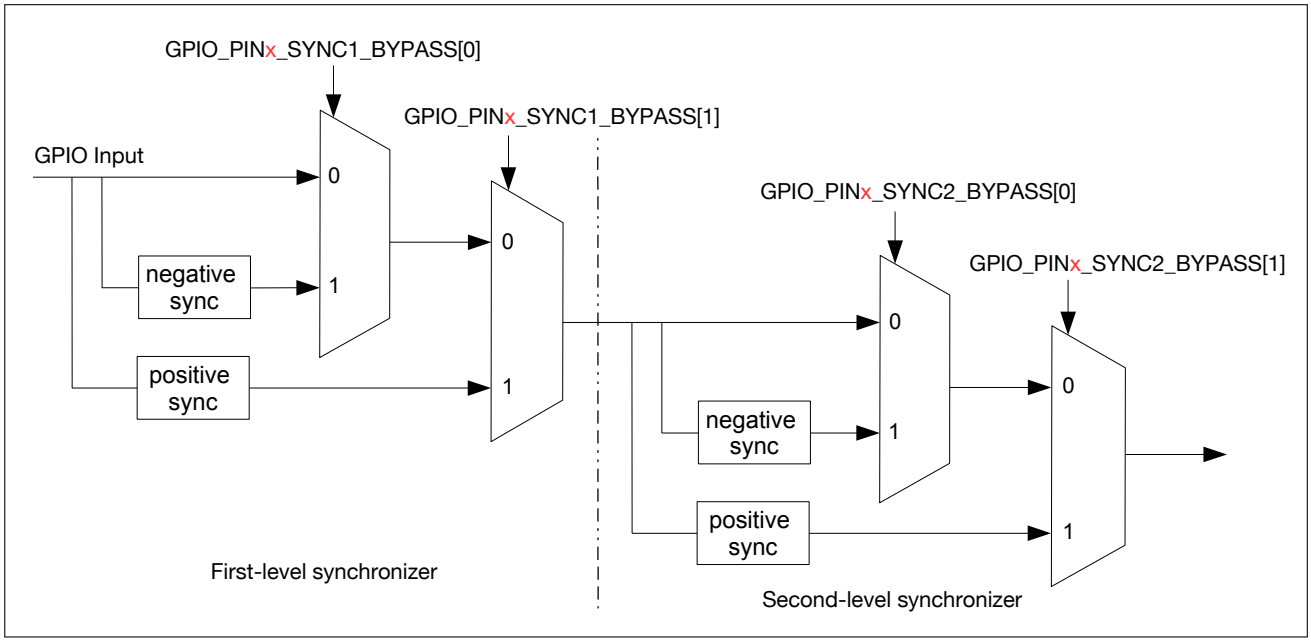


图 8.4-1. GPIO 输入经 HP IO MUX 运行时钟上升沿或下降沿同步

同步器 (synchronizer) 默认关闭同步功能。但如果一个异步外设信号连接到管脚时，该信号应通过两级同步器（即图 8.4-1 中的 first-level synchronizer 和 second-level synchronizer）进行同步，以减小亚稳态产生的概率。

8.4.3 GPIO 滤波器

GPIO 滤波功能仅在 HP GPIO 交换矩阵中可用。使能 GPIO 滤波器功能后，只有当输入信号的有效宽度大于两个时钟周期时，输入信号才会被采样。否则，输入信号将会被滤掉。见图 8.4-2。

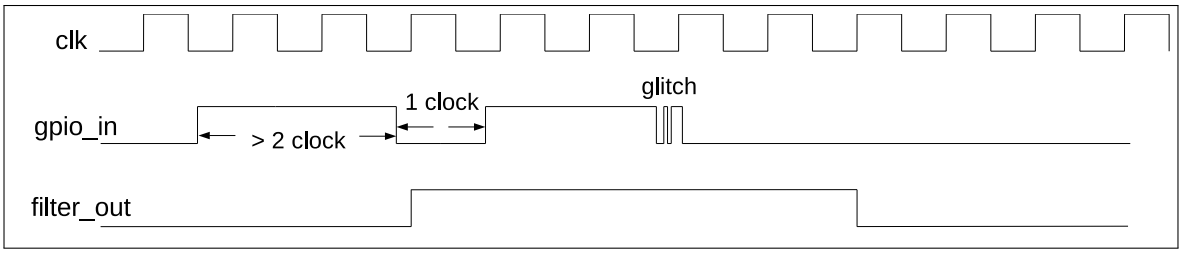


图 8.4-2. GPIO 输入信号滤波时序图

8.4.4 毛刺滤波器

毛刺滤波功能仅在 HP GPIO 交换矩阵中可用。毛刺滤波器硬件支持八个通道，每个通道可从 GPIO 滤波器硬件输出的 21 个信号 (0~14、23~28) 中选择一个信号，进行二次滤波。该毛刺滤波器硬件可用于对慢速信号进行滤波。见章节 8.4.7.1 中的步骤 3。

8.4.5 简单 GPIO 输入

HP GPIO 交换矩阵和 LP GPIO 交换矩阵均支持简单 GPIO 输入。使用简单 GPIO 输入模式时，任意 GPIO 管脚的值均可随时读取，而无需将 GPIO 管脚输入绑定到某个外设信号。

配置 HP GPIO 交换矩阵实现简单 GPIO 输入，具体过程如下：

- 置位 IO_MUX_GPIO*x*_REG 的 IO_MUX_GPIO*x*_MCU_IE 位，使能管脚输入。
- 读取 GPIO_IN_REG[*x*]，即可实现简单 GPIO 输入。

配置 LP GPIO 交换矩阵实现简单 GPIO 输入，具体过程如下：

- 置位 LP_IO_MUX_GPIO*x*_REG 的 LP_IO_MUX_GPIO*x*_FUN_IE 位，使能管脚输入。
- 读取 LP_GPIO_IN_REG[*x*]，即可实现简单 GPIO 输入。

8.4.6 GPIO 唤醒

8.4.6.1 HP GPIO 唤醒

当 HP 系统处于 Light-sleep 模式时，HP GPIO 唤醒功能可用于将系统从 Light-sleep 模式唤醒。当 HP 系统掉电时，HP GPIO 唤醒功能不可用。可通过配置 GPIO_PIN*x*_WAKEUP_ENABLE (*x*: 0~14, 23~28) 使能 HP GPIO 唤醒功能。

HP GPIO 支持高电压敏感和低电压敏感两种唤醒方式，具体由 GPIO_PIN*x*_INT_TYPE 配置决定，见下表。

表 8.4-1. HP GPIO 唤醒信号触发条件和清除条件

GPIO_PIN*x*_INT_TYPE[1,2]	触发条件	清除条件
5	GPIO 输入为高电平	GPIO 输入为低电平
4	GPIO 输入为低电平	GPIO 输入为高电平

[1] *x* = 0~14、23~28。

[2] 配置为其他值时，HP GPIO 唤醒功能不可用。

8.4.6.2 LP GPIO 唤醒

LP GPIO 唤醒功能可用于将系统从 Deep-sleep 模式唤醒。当 LP 外设掉电时，LP GPIO 唤醒功能不可用。GPIO0~GPIO6 可通过配置 LP_GPIO_PIN*x*_WAKEUP_ENABLE (*x* = 0~6) 使能 LP GPIO 唤醒功能。

LP GPIO 唤醒支持上升沿敏感、下降沿敏感、上升沿和下降沿敏感、高电压敏感以及低电压敏感唤醒。具体情况由寄存器配置决定，见下表。

表 8.4-2. LP GPIO 唤醒信号触发条件和清除条件

LP_GPIO_PINx_INT_TYPE[1,2]	触发条件	清除条件
1	GPIO 输入从低电平跳变为高电平	置位 LP_GPIO_PINx_EDGE_WAKEUP_CLR
2	GPIO 输入从高电平跳变为低电平	置位 LP_GPIO_PINx_EDGE_WAKEUP_CLR
3	GPIO 输入从高电平跳变为低电平，或从低电平跳变为高电平	置位 LP_GPIO_PINx_EDGE_WAKEUP_CLR
4	GPIO 输入为低电平	GPIO 输入为高电平
5	GPIO 输入为高电平	GPIO 输入为低电平

[1] *x* = 0~6。
[2] 配置为其他值时，LP GPIO 唤醒功能不可用。

8.4.7 编程流程

8.4.7.1 HP GPIO 交换矩阵

把某个 HP 外设输入信号 *Y* 绑定到某个 GPIO 管脚 *X* 的配置过程如下：

1. 在 HP GPIO 交换矩阵中配置外设信号 *Y* 的 GPIO_FUNC*y*_IN_SEL_CFG_REG 寄存器：
   - 置位 GPIO_SIG*y*_IN_SEL 选择通过 HP GPIO 交换矩阵接收外部输入信号。
   - 设置 GPIO_FUNC*y*_IN_SEL 为需要的 GPIO 管脚编号，此处应为 *X*。

   注意： 并不是所有的 HP 外设信号都有有效的 GPIO_SIG*y*_IN_SEL 位，即有些外设信号只能通过 HP GPIO 交换矩阵接收外部输入信号。
2. 可选：置位 IO_MUX_GPIO*n*_FILTER_EN 使能 GPIO 管脚的输入信号滤波功能。
3. 启用毛刺滤波器功能，步骤如下：
   - 配置 GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM 为 *m*。此处 *n* 为通道编号，取值范围为：0~7。*m* 为 GPIO 管脚编号，取值范围为：0~14、23~28。
   - 配置 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 为 **VALUE1**，配置 GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 为 **VALUE2**。在 **VALUE1** + 1 个周期内，如果存在 **VALUE2** + 1 个输入信号与当前输出信号值不匹配，毛刺滤波器硬件将反转输出信号。GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 和 GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 可以配置为相同的值 **VALUE3**，那么只有宽度大于 **VALUE3** + 1 个时钟周期的信号才会被采样。
   - 置位 GPIO_EXT_FILTER_CH*n*_EN 使能通道 *n*。

   相关示例见图 8.4-3，其中 GPIO_EXT_FILTER_CH*x*_WINDOW_WIDTH 配置为 3 GPIO_EXT_FILTER_CH*x*_WINDOW_THRES 配置为 2。在 T1 之前的四个时钟周期内，输出信号值 (signal_out) 一直为 “0”。输入信号值 (signal_in) 有三个时钟周期为 “1”，则输出信号在 T1 之后反转为 “1”。

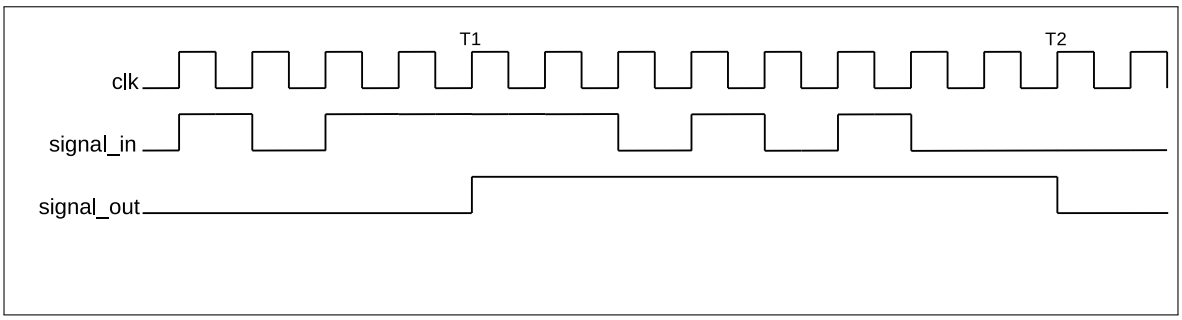


图 8.4-3. 毛刺滤波器时序示例

4. 同步 GPIO 输入信号。配置 GPIO 管脚 *X* 的 GPIO_PIN*x*_REG 来同步 GPIO 输入信号，过程如下：
   - 如图 8.4-1 所示，配置 GPIO_PIN*x*_SYNC1_BYPASS 使能输入信号在第一级同步中为上升沿或下降沿同步。
   - 如图 8.4-1 所示，配置 GPIO_PIN*x*_SYNC2_BYPASS 使能输入信号在第二级同步中为上升沿或下降沿同步。
5. 配置 HP IO MUX 寄存器使能 GPIO 管脚的输入功能。配置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_REG，过程如下：
   - 置位 IO_MUX_GPIO*x*_FUN_IE 使能输入[2,3]。
   - 置位或清零 IO_MUX_GPIO*x*_FUN_WPU 和 IO_MUX_GPIO*x*_FUN_WPD，使能或关闭内部上拉/下拉电阻。

说明：

1. 同一个输入管脚可以同时绑定多个输入信号。
2. 置位 GPIO_FUNC*y*_IN_INV_SEL 可以把输入信号取反。
3. 无需将输入信号绑定到一个 GPIO 管脚也可以使 HP 外设读取恒低或恒高电平的输入值。实现方式为选择特定的 GPIO_FUNC*y*_IN_SEL 输入值而不是一个 GPIO 序号：
   - 设置 GPIO_FUNC*y*_IN_SEL 为 0x40，则输入信号恒为 1。
   - 设置 GPIO_FUNC*y*_IN_SEL 为 0x60，则输入信号恒为 0。

编程示例

要把 UART0 RXD 输入信号[3]（U0RXD_in，信号索引号 6）绑定到 GPIO7，请按照以下步骤操作：

1. 置位 GPIO_FUNC6_IN_SEL_CFG_REG 中的 GPIO_SIG6_IN_SEL 位，使能通过 HP GPIO 交换矩阵接收外部输入信号。
2. 配置 GPIO_FUNC6_IN_SEL_CFG_REG 中的 GPIO_FUNC6_IN_SEL 为 7，即选择管脚 GPIO7。
3. 置位 IO_MUX_GPIO7_REG 中 IO_MUX_GPIO7_FUN_IE 位使能管脚输入。

8.4.7.2 LP GPIO 交换矩阵

LP GPIO 交换矩阵输入仅支持 GPIO 唤醒，配置 LP GPIO 输入唤醒可参考章节 8.4.6.2。

8.5 通过 GPIO 交换矩阵的外设输出

8.5.1 概述

通过 GPIO 交换矩阵输出外设信号，一般步骤如下：

- 配置 HP GPIO 交换矩阵将 HP 外设信号（即在表 8.12-1 中“输出信号”一栏所列出的信号）输出到 21 个 GPIO (0~14、23~28) 管脚
- 输出信号从外设输出到 GPIO 交换矩阵，然后到达 IO MUX。IO MUX 必须设置相应管脚为 GPIO 功能，这样输出 GPIO 信号就能连接到相应管脚。

具体的编程步骤见章节 8.5.4.1。

说明：
表 8.12-1 中输出索引号为 97~100 的外设信号，没有连接至外设，可配置为从一个 GPIO 管脚输出后，直接由另一个 GPIO 管脚输入（索引号：97~100）。

8.5.2 简单 GPIO 输出

GPIO 交换矩阵也可用于简单 GPIO 输出，即 GPIO 管脚可直接输出所期望的输出值，而无需将某个外设信号绑定到该 GPIO 管脚。

对于 HP GPIO 交换矩阵，具体配置如下：

- 设置 GPIO_FUNC*x*_OUT_SEL 为特定的外设索引值 256 (0x100)。
- 设置 GPIO_OUT_REG 中相应位的值为期望 GPIO 输出的值。

说明：

- GPIO_OUT_REG[0~14 和 23~28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。
- 推荐使用相应的 W1TS 和 W1TC 寄存器，例如 GPIO_OUT_W1TS 来置位/清除 GPIO_OUT_REG。

对于 LP GPIO 交换矩阵，仅支持简单 GPIO 输出，因此无需配置功能选择寄存器，仅需设置 LP_GPIO_OUT_REG 中相应位的值为期望 GPIO 输出的值。

说明：
推荐使用相应的 W1TS 和 W1TC 寄存器，例如 LP_GPIO_OUT_W1TS 或 LP_GPIO_OUT_W1TC 来置位/清除 LP_GPIO_OUT_REG。

8.5.3 Sigma Delta 调制输出 (SDM)

8.5.3.1 功能描述

HP 外设输出信号中有四个信号（在表 8.12-1 中索引为：76~79）支持 1-bit 二阶 SDM 调制输出。上述四个信号通道默认输出使能。Sigma Delta 调制器可实现输出可配占空比的 PDM（脉冲密度调制）信号。二阶 Sigma Delta 调制器函数为：

$$H(z) = X(z)z^{-1} + E(z)(1-z^{-1})^2$$

E(z) 为量化误差，X(z) 为输入。
Sigma Delta 调制器内部支持对 IO MUX 运行时钟的 1~256 倍分频：

- 置位 GPIO_EXT_SIGMADELTA_CLK_EN 使能调制器时钟。
- 配置 GPIO_EXT_SD*n*_PRESCALE 实现分频。*n* 取值范围为 0~3，对应四个信号通道。

分频后的时钟周期为调制器输出单位脉冲的周期。

GPIO_EXT_SD*n*_IN 为有符号数，范围为 [-128, 127]，配置此寄存器控制输出 PDM 信号的占空比 [1]。

- GPIO_EXT_SD*n*_IN = -128，调制器输出信号占空比为 0%。
- GPIO_EXT_SD*n*_IN = 0，调制器输出信号占空比接近 50%。
- GPIO_EXT_SD*n*_IN = 127，调制器输出信号占空比接近 100%。

PDM 信号占空比计算公式为：

$$Duty\_Cycle = \frac{GPIO\_EXT\_SDn\_IN + 128}{256}$$

说明：
对 PDM 信号来说，占空比是指在若干脉冲周期内（比如 256 个脉冲周期），高电平占整个统计周期的比值。

8.5.3.2 配置方法

SDM 的配置方法如下：

- 将 SDM 输出经 HP GPIO 交换矩阵连接至相应管脚，见章节 8.5.4。
- 置位 GPIO_EXT_SIGMADELTA_CLK_EN，使能 SDM 时钟
- 配置 GPIO_EXT_SD*n*_PRESCALE 设置时钟分频系数。
- 配置 GPIO_EXT_SD*n*_IN 设置 SDM 输出信号的占空比。

8.5.4 编程流程

8.5.4.1 HP GPIO 交换矩阵

如图 8.3-1 所示，表 8.12-1 中“输出信号”列的某一个信号通过 HP GPIO 交换矩阵到达 HP IO MUX，然后连接到某个 GPIO 管脚。

输出外设信号 *Y* 到某一 GPIO 管脚 *X* [1] 的步骤如下：

1. 在 HP GPIO 交换矩阵中配置 GPIO 管脚 *X* 的 GPIO_FUNC*x*_OUT_SEL_CFG_REG 和 GPIO_ENABLE_REG[*x*]。推荐使用相应 W1TS（写 1 置位）和 W1TC（写 1 清零）来更新 GPIO_ENABLE_REG 中的值。
    - 设置 GPIO_FUNC*x*_OUT_SEL_CFG_REG 中的 GPIO_FUNC*x*_OUT_SEL 字段为外设输出信号 Y 的索引号 (*Y*)。

- 要将信号强制使能为输出模式，需要将 GPIO 管脚 *X* 对应的 GPIO_FUNC*x*_OUT_SEL_CFG_REG 寄存器中 GPIO_FUNC*x*_OE_SEL 字段置位；同时需要将 GPIO_ENABLE_W1TS_REG 中的相应位置位。或者，将 GPIO_FUNC*x*_OE_SEL 清零，即选择采用外设的输出使能信号，此时输出使能信号由内部逻辑功能决定。比如，表 8.12-1 中 “GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号” 一栏的 SPIQ_oe 信号。
- 置位 GPIO_ENABLE_W1TC_REG 中相应位可以关闭 GPIO 管脚的输出。

2. 要选择以开漏方式输出，可以设置 GPIO 管脚 *X* 的 GPIO_PIN*x*_REG 寄存器中 GPIO_PIN*x*_PAD_DRIVER 位。
3. 配置 HP IO MUX 寄存器来选择经由 HP GPIO 交换矩阵输出信号。配置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_REG 的过程如下：
   - 设置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_MCU_SEL 为所需的管脚功能。此处选择数值 1，即功能 1（GPIO 功能），适用于所有管脚。
   - 设置 IO_MUX_GPIO*x*_FUN_DRV 字段为特定的输出强度值 (0~3)，值越大，输出驱动能力越强。
   - 在开漏模式下，通过置位/清零 IO_MUX_GPIO*x*_FUN_WPU 和 IO_MUX_GPIO*x*_FUN_WPD 使能或关闭上拉/下拉电阻。
4. 使能迟滞功能：
   - 请参考章节 8.10。

说明：

- 某一个外设的输出信号可以同时从多个管脚输出。
- 置位 GPIO_FUNC*x*_OUT_INV_SEL 可以把输出的信号取反。

8.5.4.2 LP GPIO 交换矩阵

LP GPIO 交换矩阵输出仅支持简单 GPIO 输出，配置 LP GPIO 简单输出可参考章节 8.5.2。

8.6 IO MUX 的直接输入输出功能

8.6.1 概述

SPI、JTAG 等数字信号可以旁路 GPIO 交换矩阵以实现更好的高频数字特性。所以此类信号可以直接通过 IO MUX 输入和输出。这样比使用 GPIO 交换矩阵的灵活度要低，即每个 GPIO 管脚的 IO MUX 寄存器只有较少的功能选择，但可以实现更好的高频数字特性。

ESP32-C5 中有 7 个 GPIO 管脚 (GPIO0~GPIO6) 具有低功耗 (LP) 性能。这些管脚可由 HP IO MUX 或 LP IO MUX 控制。如果由 LP IO MUX 控制，这些管脚将旁路 HP IO MUX 和 HP GPIO 交换矩阵，直接供 LP 系统中的外设使用。

当这些管脚被配置为 LP GPIO 管脚，仍然能够在芯片处于 Deep-sleep 模式下由 LP 系统的外设控制，也可以作为唤醒源将芯片从 Deep-sleep 中唤醒。

8.6.2 功能描述

LP 管脚 (GPIO0~GPIO6) 由 LP_AON_GPIO_MUX_REG 中的 LP_AON_GPIO_MUX_SEL 的 bit*[n]* (*n* = 0~6) 位控制。此位默认置为 0，通过 HP IO MUX 输入输出信号。

如果置位 LP_AON_GPIO_MUX_SEL 的 bit*[n]*，则控制权交由 LP IO MUX 接管。在这种模式下，LP_IO_MUX_GPIO*n*_REG 寄存器用于控制 LP GPIO 管脚。表 8.14-1 列出了 LP GPIO 管脚的 LP 功能。LP_IO_MUX_GPIO*n*_REG 寄存器使用的是 LP GPIO 管脚的序号，ESP32-C5 芯片中 LP GPIO 管脚的序号与 HP GPIO 管脚的序号相同。

8.6.2.1 HP IO MUX

对于 HP 外设输入信号，旁路 HP GPIO 交换矩阵必须配置以下两个字段：

1. GPIO 管脚的 IO_MUX_GPIO*n*_MCU_SEL 必须配置为相应的管脚功能。表 8.13-1 列出了管脚功能。
2. 清除 GPIO_SIG*n*_IN_SEL，直接将输入信号连接到外设。

对于 HP 外设输出信号，旁路 HP GPIO 交换矩阵只需将 GPIO 管脚的 IO_MUX_GPIO*n*_MCU_SEL 配置为相应的管脚功能即可。

8.6.2.2 LP IO MUX

对于 LP 外设输入信号，旁路 LP GPIO 交换矩阵必须配置 LP_IO_MUX_GPIO*n*_MCU_SEL 字段：GPIO 管脚的 LP_IO_MUX_GPIO*n*_MCU_SEL 必须配置为相应的管脚功能。表 8.14-1 列出了管脚功能。

对于 LP 外设输出信号，旁路 LP GPIO 交换矩阵只需将 LP GPIO 管脚的 LP_IO_MUX_GPIO*n*_MCU_SEL 配置为相应的管脚功能即可。

> **说明：**
> 并非所有外设输入/输出信号均可直接通过 IO MUX 连接到外设，某些输入/输出信号只能通过 HP GPIO 交换矩阵连接到外设，详见表 8.12-1。

8.7 模拟功能描述

8.7.1 概述

ESP32-C5 共有 11 个 GPIO 管脚 (GPIO0~GPIO6, GPIO8~GPIO9, GPIO13~GPIO14) 具有模拟功能，其中包括 6 个 LP GPIO 管脚和 5 个 HP GPIO 管脚。见表 8.15-1。

8.7.2 模拟功能描述

当使用管脚的模拟功能时，需要确保该管脚处于浮空状态，此时外部模拟信号通过 GPIO 管脚直接与内部的模拟信号相连。相关配置如下：

- 清除 IO_MUX_GPIO*n*_FUN_IE、IO_MUX_GPIO*n*_FUN_WPU 和 IO_MUX_GPIO*n*_FUN_WPD 将管脚设置为浮空状态。
- 向 GPIO_ENABLE_W1TC 中对应位写 1，清除输出使能。

对于表 8.15-1 中列出的不同模拟功能的使用方法，请参考相关章节：

- XTAL_32K 相关功能见章节 13 低功耗管理。
- ADC 相关功能见章节 46 *ADC* 控制器。
- 模拟电压比较器功能见章节 47 模拟电压比较器。

说明：
表 8.15-1 中 GPIO1 有两种模拟功能，可以同时使用。

8.8 Light-sleep 模式管脚功能

当 ESP32-C5 处于 Light-sleep 模式时管脚可以有不同的功能。如果某一 GPIO 管脚的 IO_MUX_GPIO*n*_REG 寄存器中 IO_MUX_GPIO*n*_SLP_SEL 置位时，芯片处于 Light-sleep 模式下将由另一组不同的寄存器控制管脚。

表 8.8-1. IO MUX Light-sleep 管脚功能控制寄存器

IO MUX 功能	正常工作模式 OR IO_MUX_GPIO*n*_SLP_SEL = 0	Light-sleep 模式 AND IO_MUX_GPIO*n*_SLP_SEL = 1
输出驱动强度	IO_MUX_GPIO*n*_FUN_DRV	IO_MUX_GPIO*n*_MCU_DRV
上拉电阻	IO_MUX_GPIO*n*_FUN_WPU	IO_MUX_GPIO*n*_MCU_WPU
下拉电阻	IO_MUX_GPIO*n*_FUN_WPD	IO_MUX_GPIO*n*_MCU_WPD
输入使能	IO_MUX_GPIO*n*_FUN_IE	IO_MUX_GPIO*n*_MCU_IE
输出使能	由 GPIO 交换矩阵的 OE_SEL*	IO_MUX_GPIO*n*_MCU_OE

说明：
如果 IO_MUX_GPIO*n*_SLP_SEL 置为 0，则芯片在正常工作和 Light-sleep 模式下，管脚的功能一样。此时，具体的输出使能配置请参考 8.5.4 章节。

8.9 GPIO 管脚的 Hold 特性

每个 GPIO 管脚都有单独的 Hold 功能，由电源管理单元 (PMU) 或寄存器控制。管脚的 Hold 功能被置上后，管脚在置上 Hold 那一刻的状态被强制保持，无论内部信号如何变化，修改 IO MUX 配置或者 GPIO 配置，都不会改变管脚的状态。应用如果希望在看门狗超时触发内核复位时或者 Deep-sleep 触发内核复位时管脚的状态不被改变，就需要提前把 Hold 置上。

每个 GPIO 管脚的 Hold 状态由各管脚的 Hold 使能信号和全局的 Hold 使能信号经过**或运算**后共同控制。

- 各个 GPIO 管脚的 Hold 信号
  - LP_AON_GPIO_HOLD0_REG[*n*] (*n* = 0~14, 23~28) 控制各个 GPIO 管脚的 Hold 信号。
- 全局 Hold 信号
  - HP GPIO 的全局 Hold 信号
    * PMU_TIE_HIGH_HP_PAD_HOLD_ALL：用于使能所有 HP GPIO 管脚的全局 Hold 信号。

    * PMU_TIE_LOW_HP_PAD_HOLD_ALL：用于关闭所有 HP GPIO 管脚的全局 Hold 信号。
  - LP GPIO 的全局 Hold 信号
    * PMU_TIE_HIGH_LP_PAD_HOLD_ALL：用于使能 LP GPIO 管脚的全局 Hold 信号。
    * PMU_TIE_LOW_LP_PAD_HOLD_ALL：用于关闭所有 LP GPIO 管脚的全局 Hold 信号。

可通过以下任一方式使能或关闭管脚的 Hold 功能。

- **方式 1：使能或关闭某一个管脚的 Hold 功能**
  如要在 Deep-sleep 中保持该管脚的输入输出的状态值，需要置位 LP_AON_GPIO_HOLD0_REG[*n*] 中的相应位，其中 *n* = 0~14, 23~28，分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。在芯片被唤醒后，若要关闭 Hold 功能，可将 LP_AON_GPIO_HOLD0_REG 中的相应位 bit[*n*] 清除。
- **方式 2：使能或关闭所有 HP 管脚和 LP 管脚的 Hold 功能**
  - HP 管脚
    置位 PMU_TIE_HIGH_HP_PAD_HOLD_ALL 可保持所有 HP 管脚的输入输出状态；置位 PMU_TIE_LOW_HP_PAD_HOLD_ALL 可关闭所有 HP 管脚的 Hold 功能。
  - LP 管脚
    置位 PMU_TIE_HIGH_LP_PAD_HOLD_ALL 可保持所有 LP 管脚的输入输出状态；置位 PMU_TIE_LOW_LP_PAD_HOLD_ALL 可关闭所有 LP 管脚的 Hold 功能。
- **方式 3：使用 PMU 任务**
  在芯片进入 Deep-sleep 前，可配置 PMU 任务，用以保持所有 HP 管脚的状态。进入 Deep-sleep 后，PMU 可自动保持这些 HP 管脚的值。

8.10 GPIO 管脚的迟滞特性

每个 GPIO 管脚都具有迟滞功能。当未使能迟滞功能时，如图 8.10-1 所示，芯片焊盘 (PAD) 上输入给芯片内部的信号 (C) 的电平翻转只有一个阈值（Vt，约 1.7 V）。当 PAD 的电压高于 Vt 时，C 的电平为高，否则为低。但如果 PAD 上的信号有噪声，可能会影响 C 上的信号。

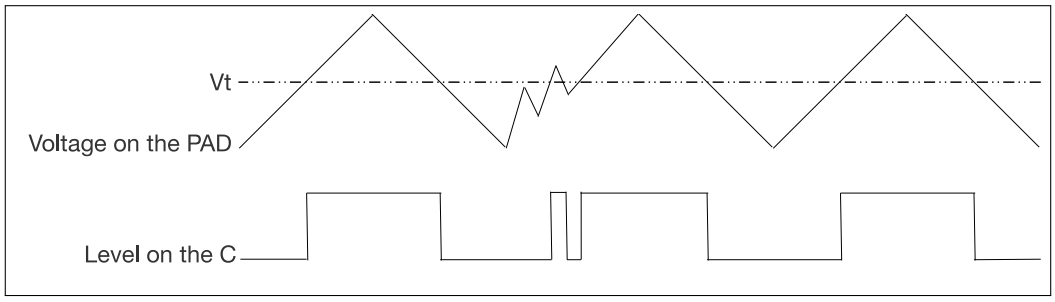


图 8.10-1. 未使能迟滞功能时 PAD 上的电平翻转示例

当使能迟滞功能时，如图 8.10-2 所示，C 的电平翻转有两个阈值，分别为高电平阈值（Vth，约 1.7 V）和低电平阈值（Vtl，约 1.4 V）。当 PAD 的电压由低变高时，如果电压高于 Vth，则 C 的电平为高。当 PAD 的电压由高变低时，若电压低于 Vtl，则 C 的电平为低。当 PAD 的电压在 Vth 和 Vtl 之间时，C 的电平不变。迟滞功能可以减少噪声的影响，起到抗干扰作用，减少 C 的电平翻转次数。

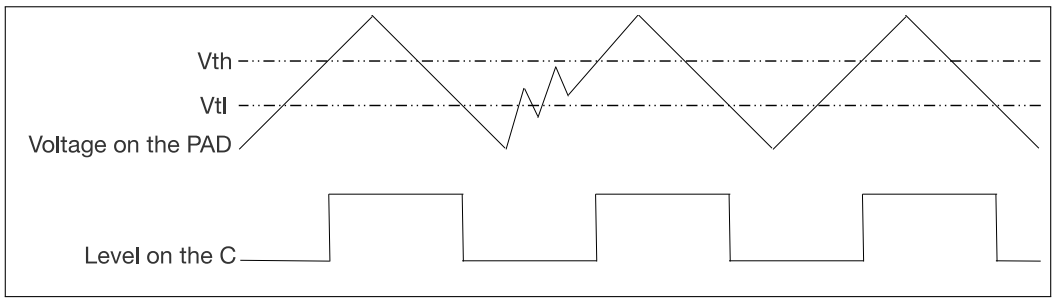


图 8.10-2. 使能迟滞功能时 PAD 上的电平翻转示例

要使能迟滞功能，请按照以下步骤操作：

- 当 IO_MUX_GPIO*n*_HYS_SEL 为 0 时（*n* 的取值范围是 0~14、23~28，分别对应 GPIO0~GPIO14、GPIO23~GPIO28）：
  - 当 EFUSE_HYS_EN_PAD 为 1 时，开启 GPIO0 ~GPIO14、GPIO23~GPIO28 的迟滞功能。
  - 当 EFUSE_HYS_EN_PAD 为 0 时，关闭 GPIO0 ~GPIO14、GPIO23~GPIO28 的迟滞功能。
- 当 IO_MUX_GPIO*n*_HYS_SEL 为 1 时（*n* 的取值范围是 0~14、23~28，分别对应 GPIO0~GPIO14、GPIO23~GPIO28）：
  - 设置 IO_MUX_GPIO*n*_HYS_EN 为 1，可开启 GPIO*n* 的迟滞功能
  - 设置 IO_MUX_GPIO*n*_HYS_EN 为 0，可关闭 GPIO*n* 的迟滞功能

建议设置 IO_MUX_GPIO*n*_HYS_SEL 为 1，使用 IO_MUX_GPIO*n*_HYS_EN 来开启或关闭 GPIO*n* 的迟滞功能。

8.11 GPIO 管脚供电和电源管理

8.11.1 GPIO 管脚供电

GPIO 管脚供电请参考《*ESP32-C5 技术规格书*》中管脚定义章节。所有管脚均可用于将芯片从 Light-sleep 中唤醒，但仅有 VDDPST1 域中的管脚 (GPIO0~GPIO6) 可用于将芯片从 Deep-sleep 唤醒。

8.11.2 电源管理

ESP32-C5 的管脚可由如下不同的电源域供电。

- VDDPST1：LP GPIO 管脚的输入电源
- VDDPST2：HP GPIO 管脚的输入电源

8.12 HP 外设信号列表

表 8.12-1 列出了所有经由 HP GPIO 交换矩阵的外设输入输出信号。

请注意 GPIO_FUNC*n*_OE_SEL 位的配置：

- GPIO_FUNC*n*_OE_SEL = 1，则寄存器 GPIO_ENABLE_REG 中的相应位 *n* 将用于控制信号输出使能。

  - GPIO_ENABLE_REG[*n*] = 0：输出关闭。
  - GPIO_ENABLE_REG[*n*] = 1：输出使能。

- GPIO_FUNC*n*_OE_SEL = 0，则输出信号的使能由外设控制，例如表 8.12-1 中“GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号”一栏的 SPIQ_oe。注意，使能信号 SPIQ_oe 可设置为 1 (1'd1) 或 0 (1'd0)，具体由外设的配置决定。如果“GPIO_FUNC*n*_OE_SEL = 0 时输出信号的输出使能信号”一栏中为 1'd1，则表示输出信号默认始终使能。

说明：

信号连续编号，但并非所有信号均有效。

- 表 8.12-1“输入信号”一栏中有名字的信号均为有效输入信号。
- 表 8.12-1“输出信号”一栏中有名字的信号均为有效输出信号。

表 8.12-1. GPIO 交换矩阵外设信号

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
0	—	-	-	ledc_ls_sig_out0	1'd1	no
1	—	—	—	ledc_ls_sig_out1	1'd1	no
2	—	—	—	ledc_ls_sig_out2	1'd1	no
3	—	—	—	ledc_ls_sig_out3	1'd1	no
4	—	—	—	ledc_ls_sig_out4	1'd1	no
5	—	—	—	ledc_ls_sig_out5	1'd1	no
6	U0RXD_in	0	yes	U0TXD_out	1'd1	yes
7	U0CTS_in	0	no	U0RTS_out	1'd1	no
8	U0DSR_in	0	no	U0DTR_out	1'd1	no
9	U1RXD_in	1	no	U1TXD_out	1'd1	no
10	U1CTS_in	0	no	U1RTS_out	1'd1	no
11	U1DSR_in	0	no	U1DTR_out	1'd1	no
12	I2S_MCLK_in	0	no	I2S_MCLK_out	1'd1	no
13	I2SO_BCK_in	0	no	I2SO_BCK_out	1'd1	no
14	I2SO_WS_in	0	no	I2SO_WS_out	1'd1	no
15	I2SI_SD_in	0	no	I2SO_SD_out	1'd1	no
16	I2SI_BCK_in	0	no	I2SI_BCK_out	1'd1	no
17	I2SI_WS_in	0	no	I2SI_WS_out	1'd1	no
18	—	—	—	I2SO_SD1_out	1'd1	no
19	—	—	—	—	—	—
20	—	—	—	—	—	—
21	—	—	—	—	—	—
22	—	—	—	—	—	—
23	—	—	—	—	—	—
24	—	—	—	—	—	—
25	—	—	—	—	—	—
26	—	—	—	—	—	—
27	cpu_gpio_in0	0	no	cpu_gpio_out0	cpu_gpio_out_oen0	no
28	cpu_gpio_in1	0	no	cpu_gpio_out1	cpu_gpio_out_oen1	no

GoBack

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
29	cpu_gpio_in2	0	no	cpu_gpio_out2	cpu_gpio_out_oen2	no
30	cpu_gpio_in3	0	no	cpu_gpio_out3	cpu_gpio_out_oen3	no
31	cpu_gpio_in4	0	no	cpu_gpio_out4	cpu_gpio_out_oen4	no
32	cpu_gpio_in5	0	no	cpu_gpio_out5	cpu_gpio_out_oen5	no
33	cpu_gpio_in6	0	no	cpu_gpio_out6	cpu_gpio_out_oen6	no
34	cpu_gpio_in7	0	no	cpu_gpio_out7	cpu_gpio_out_oen7	no
35	usb_jtag_tdo_bridge	0	no	—	—	—
36	—	—	—	—	—	—
37	—	—	—	—	—	—
38	—	—	—	—	—	—
39	—	—	—	—	—	—
40	—	—	—	—	—	—
41	—	—	—	—	—	—
42	—	—	—	—	—	—
43	—	—	—	—	—	—
44	—	—	—	—	—	—
45	—	—	—	—	—	—
46	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe	no
47	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe	no
48	parl_rx_data0	0	no	parl_tx_data0	1'd1	no
49	parl_rx_data1	0	no	parl_tx_data1	1'd1	no
50	parl_rx_data2	0	no	parl_tx_data2	1'd1	no
51	parl_rx_data3	0	no	parl_tx_data3	1'd1	no
52	parl_rx_data4	0	no	parl_tx_data4	1'd1	no
53	parl_rx_data5	0	no	parl_tx_data5	1'd1	no
54	parl_rx_data6	0	no	parl_tx_data6	1'd1	no
55	parl_rx_data7	0	no	parl_tx_data7	1'd1	no
56	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe	yes
57	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe	yes
58	FSPID_in	0	yes	FSPID_out	FSPID_oe	yes

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
59	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe	yes
60	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe	yes
61	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe	yes
62	parl_rx_clk_in	0	no	parl_rx_clk_out	1'd1	no
63	parl_tx_clk_in	0	no	parl_tx_clk_out	1'd1	no
64	rmt_sig_in0	0	no	rmt_sig_out0	1'd1	no
65	rmt_sig_in1	0	no	rmt_sig_out1	1'd1	no
66	twai0_rx	1	no	twai0_tx	1'd1	no
67	—	—	—	twai0_bus_off_on	1'd1	no
68	—	—	—	twai0_clkout	1'd1	no
69	—	—	—	twai0_standby	1'd1	no
70	twai1_rx	1	no	twai1_tx	1'd1	no
71	—	—	—	twai1_bus_off_on	1'd1	no
72	—	—	—	twai1_clkout	1'd1	no
73	—	—	—	twai1_standby	1'd1	no
74	—	—	—	—	—	—
75	—	—	—	—	—	—
76	pcnt_rst_in0	0	no	gpio_sd0_out	1'd1	no
77	pcnt_rst_in1	0	no	gpio_sd1_out	1'd1	no
78	pcnt_rst_in2	0	no	gpio_sd2_out	1'd1	no
79	pcnt_rst_in3	0	no	gpio_sd3_out	1'd1	no
80	pwm0_sync0_in	0	no	pwm0_out0a	1'd1	no
81	pwm0_sync1_in	0	no	pwm0_out0b	1'd1	no
82	pwm0_sync2_in	0	no	pwm0_out1a	1'd1	no
83	pwm0_f0_in	0	no	pwm0_out1b	1'd1	no
84	pwm0_f1_in	0	no	pwm0_out2a	1'd1	no
85	pwm0_f2_in	0	no	pwm0_out2b	1'd1	no
86	pwm0_cap0_in	0	no	parl_tx_cs_o	1'd1	no
87	pwm0_cap1_in	0	no	—	—	—
88	pwm0_cap2_in	0	no	—	—	—

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
89	—	—	—	—	—	—
90	—	—	—	—	—	—
91	—	—	—	—	—	—
92	—	—	—	—	—	—
93	—	—	—	—	—	—
94	—	—	—	—	—	—
95	—	—	—	—	—	—
96	—	—	—	—	—	—
97	sig_in_func_97	0	no	sig_in_func97	1'd1	no
98	sig_in_func_98	0	no	sig_in_func98	1'd1	no
99	sig_in_func_99	0	no	sig_in_func99	1'd1	no
100	sig_in_func_100	0	no	sig_in_func100	1'd1	no
101	pcnt_sig_ch0_in0	0	no	FSPICS1_out	FSPICS1_oe	yes
102	pcnt_sig_ch1_in0	0	no	FSPICS2_out	FSPICS2_oe	yes
103	pcnt_ctrl_ch0_in0	0	no	FSPICS3_out	FSPICS3_oe	yes
104	pcnt_ctrl_ch1_in0	0	no	FSPICS4_out	FSPICS4_oe	yes
105	pcnt_sig_ch0_in1	0	no	FSPICS5_out	FSPICS5_oe	yes
106	pcnt_sig_ch1_in1	0	no	—	—	—
107	pcnt_ctrl_ch0_in1	0	no	—	—	—
108	pcnt_ctrl_ch1_in1	0	no	—	—	—
109	pcnt_sig_ch0_in2	0	no	—	—	—
110	pcnt_sig_ch1_in2	0	no	—	—	—
111	pcnt_ctrl_ch0_in2	0	no	—	—	—
112	pcnt_ctrl_ch1_in2	0	no	—	—	—
113	pcnt_sig_ch0_in3	0	no	—	—	—
114	pcnt_sig_ch1_in3	0	no	—	—	—
115	pcnt_ctrl_ch0_in3	0	no	—	—	—
116	pcnt_ctrl_ch1_in3	0	no	—	—	—
117 ~ 255	—	—	—	—	—	—

8.13 HP IO MUX 管脚功能列表

表 8.13-1 列出了每个 GPIO 管脚的 HP IO MUX 功能及其默认状态。

表 8.13-1. HP IO MUX 管脚功能

GPIO No.	名称	功能 0	功能 1	功能 2	功能 3	驱动强度	复位	说明
0	XTAL_32K_P	GPIO0	GPIO0	—	—	2	0	R
1	XTAL_32K_N	GPIO1	GPIO1	—	—	2	0	R
2	MTMS	MTMS	GPIO2	FSPIQ	—	2	1	R
3	MTDI	MTDI	GPIO3	—	—	2	1	R
4	MTCK	MTCK	GPIO4	FSPIHD	—	2	1*	R
5	MTDO	MTDO	GPIO5	FSPIWP	—	2	1	R
6	GPIO6	GPIO6	GPIO6	FSPICLK	—	2	0	R
7	GPIO7	SDIO_DATA1	GPIO7	FSPID	—	2	1	
8	GPIO8	SDIO_DATA0	GPIO8	—	—	2	0	—
9	GPIO9	SDIO_CLK	GPIO9	—	—	2	0	—
10	GPIO10	SDIO_CMD	GPIO10	FSPICS0	—	2	0	—
11	U0TXD	U0TXD	GPIO11	—	—	2	4	—
12	U0RXD	U0RXD	GPIO12	—	—	2	1	—
13	GPIO13	SDIO_DATA3	GPIO13	—	—	3	1	USB
14	GPIO14	SDIO_DATA2	GPIO14	—	—	3	3*	USB
23	GPIO23	GPIO23	GPIO23	—	—	2	0	—
24	GPIO24	GPIO24	GPIO24	—	—	2	0	—
25	GPIO25	GPIO25	GPIO25	—	—	2	1	—
26	GPIO26	GPIO26	GPIO26	—	—	2	1	—
27	GPIO27	GPIO27	GPIO27	—	—	2	3	—
28	GPIO28	GPIO28	GPIO28	—	—	2	3	—

注意：

HP IO MUX 中，没有用到的管脚需要配置成 GPIO 功能。

驱动强度

“驱动强度”一栏所示为每个管脚复位后的默认驱动强度。

- **0** - 驱动电流 = ~5 mA
- **1** - 驱动电流 = ~10 mA
- **2** - 驱动电流 = ~20 mA
- **3** - 驱动电流 = ~40 mA

复位

“复位”一栏所示为每个管脚复位后的默认配置。

- **0** - IE = 0（输入关闭）
- **1** - IE = 1（输入使能）
- **3** - IE = 1，WPU = 1（输入使能，上拉电阻使能）
- **4** - IE = 1，OE = 1（输入使能，输出使能）
- **1*** - 如果 EFUSE_DIS_PAD_JTAG = 1，则 MTCK 管脚复位后浮空，即 IE = 1。如果 EFUSE_DIS_PAD_JTAG = 0，则 MTCK 管脚连接内部上拉电阻，即 IE = 1，WPU = 1。
- **3*** - IE = 1，WPU = 0，GPIO14 的 USB 上拉默认值为 1，因此，其上拉电阻使能，具体见以下**说明**。

说明

- **R** - LP 管脚，部分具有模拟功能，见表 8.15-1。
- **USB** - USB 上拉电阻使能
  - USB 管脚（GPIO13 和 GPIO14）默认开启 USB 功能，此时管脚是否上拉由 USB 是否上拉决定。USB 上拉由 USB_SERIAL_JTAG_DP/DM_PULLUP 控制，USB 上拉电阻的具体阻值可通过 USB_SERIAL_JTAG_PULLUP_VALUE 位控制，详见《ESP32-C5 技术参考手册》> 章节 *USB 串口/JTAG 控制器*。
  - USB 管脚关闭 USB 功能时，用作普通 GPIO，默认禁用管脚内部弱上/下拉电阻，可通过 IO_MUX_GPIO*n*_MCU_WPU/WPD 配置。

8.14 LP IO MUX 管脚功能列表

表 8.14-1 列出了每个 GPIO 管脚的 LP IO MUX 功能。GPIO 管脚默认由 HP IO MUX 控制，因此表 8.14-1 仅列出了管脚的 LP IO MUX 功能。

表 8.14-1. LP IO MUX 管脚功能

LP IO 名称	功能 0	功能 1	功能 2	功能 3
GPIO0	LP_UART_DTRN	LP_GPIO0	—	—
GPIO1	LP_UART_DSRN	LP_GPIO1	—	—
GPIO2	LP_UART_RTSN	LP_GPIO2	—	LP_I2C_SDA
GPIO3	LP_UART_CTSN	LP_GPIO3	—	LP_I2C_SCL
GPIO4	LP_UART_RXD_PAD	LP_GPIO4	—	—
GPIO5	LP_UART_TXD_PAD	LP_GPIO5	—	—

LP IO 名称	功能 0	功能 1	功能 2	功能 3
GPIO6	—	LP_GPIO6	—	—

注意：

- LP_UART 的硬件流控功能无法和 LP_I2C 同时使用。
- LP IO MUX 中，没有用到的管脚需要配置成 LP_GPIO 功能。
- 由于 LP IO MUX 功能通过使用 LP GPIO 编号的 LP GPIO 寄存器配置，上表列出的是 LP GPIO 名称。

8.15 GPIO 管脚模拟功能列表

表 8.15-1 列出了 GPIO 管脚及其相应的模拟功能。

表 8.15-1. GPIO 管脚模拟功能

名称	模拟功能 0	模拟功能 1
XTAL_32K_P	XTAL_32K_P	
XTAL_32K_N	XTAL_32K_N	ADC1_CH0
MTMS	—	ADC1_CH1
MTDI	—	ADC1_CH2
MTCK	—	ADC1_CH3
MTDO	—	ADC1_CH4
GPIO6	—	ADC1_CH5
GPIO8	ZCD0*	—
GPIO9	ZCD1*	—
GPIO13	USB_D-	—
GPIO14	USB_D+	—

* ZCD0 和 ZCD1 是模拟 PAD 电压比较功能，见章节 47 模拟电压比较器。

8.16 事件任务矩阵 (ETM) 功能

在 ESP32-C5 中，GPIO 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GPIO 的 ETM 任务，或者通过 GPIO 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与 GPIO 相关的 ETM 任务和 ETM 事件。

GPIO ETM 有八个任务通道 *x* (0~7)。每个任务通道可接收的 ETM 任务有：

- GPIO_TASK_CH*x*_SET：触发时 GPIO 变为高电平。
- GPIO_TASK_CH*x*_CLEAR：触发时 GPIO 变为低电平。
- GPIO_TASK_CH*x*_TOGGLE：触发时 GPIO 电平翻转。

将任务通道 *x* 配置到 GPIO*y* 的示例如下：

- 配置 IO_MUX_GPIO*y*_MCU_SEL 的值为 1，使得 GPIO*y* 选择表 8.13-1 中的功能 1。

- 配置 GPIO_ENABLE_REG[*y*] 的值为 1。
- 配置 GPIO_EXT_ETM_TASK_GPIO*y*_SEL 为 *x*。
- 置位 GPIO_EXT_ETM_TASK_GPIO*y*_EN，使能 GPIO*y* 受到任务通道 *x* 的控制。

说明：

- 一个任务通道可被一个或多个 GPIO 选择。
- 如果 GPIO*y* 选择的任务通道 *x* 其信号 (GPIO_TASK_CH*x*_SET、GPIO_TASK_CH*x*_CLEAR 和 GPIO_TASK_CH*x*_TOGGLE) 中有两个或三个同时有效，则 GPIO_TASK_CH*x*_SET 优先级最高，GPIO_TASK_CH*x*_CLEAR 优先级次之，GPIO_TASK_CH*x*_TOGGLE 优先级最低。
- GPIO*y* 受到 ETM 任务通道控制时，GPIO_OUT_REG、GPIO_FUNC*n*_OUT_INV_SEL 和 GPIO_FUNC*n*_OUT_SEL 的值可能会被硬件修改。在 GPIO 退出 ETM 任务通道控制后，建议对上述寄存器重新进行配置。

GPIO 有八个事件通道，每个事件通道可以生成的 ETM 事件有：

- GPIO_EVT_CH*x*_RISE_EDGE：表示对应 GPIO 滤波器（见图 8.3-1）输出信号出现上升沿。
- GPIO_EVT_CH*x*_FALL_EDGE：表示对应 GPIO 滤波器（见图 8.3-1）输出信号出现下降沿。
- GPIO_EVT_CH*x*_ANY_EDGE：表示对应 GPIO 滤波器（见图 8.3-1）输出信号发生翻转。

事件通道具体配置如下：

- 置位 GPIO_EXT_ETM_CH*x*_EVENT_EN 使能事件通道 *x* (0~7)。
- 配置 GPIO_EXT_ETM_CH*x*_EVENT_SEL 为 *y* (0~14、23~28)，即选择 21 个 GPIO 中的一个 GPIO。

说明：
一个 GPIO 可被一个或多个事件通道选择。

GPIO 的 ETM 事件可以用来触发 GPIO 的 ETM 任务。例如，事件通道 0 选择 GPIO0，GPIO1 选择任务通道 0，并且 GPIO_EVT_CH*0*_RISE_EDGE 事件用于触发 GPIO_TASK_CH*0*_TOGGLE 任务。当通过 GPIO0 向芯片输入一个方波信号时，芯片通过 GPIO1 输出一个二分频的方波信号。

8.17 中断

ESP32-C5 的 HP IO MUX 和 HP GPIO 交换矩阵可以生成以下中断信号并发送给中断矩阵。

- GPIO_EXT_REG_INT
- GPIO_PROCPU_INT

HP IO MUX 和 HP GPIO 交换矩阵有多个中断源可生成以上中断信号。表 8.17-1 列出了 HP IO MUX 和 HP GPIO 交换矩阵内部的中断源及其触发条件。

表 8.17-1. HP IO MUX 和 HP GPIO 交换矩阵内部中断源

内部中断源	触发条件	中断信号
GPIO_EXT_COMP_0_ALL_INT	参考章节 47 模拟电压比较器	GPIO_EXT_REG_INT
GPIO_EXT_COMP_0_NEG_INT	参考章节 47 模拟电压比较器	GPIO_EXT_REG_INT

GPIO_EXT_COMP_0_POS_INT	参考章节 47 模拟电压比较器	GPIO_EXT_REG_INT
GPIO_REG_INTR	GPIO_STATUS_INTERRUPT[*n*] & GPIO_PIN*n*_INT_ENA[0] (*n*: 0~28)	GPIO_PROCPU_INT

ESP32-C5 的 LP IO MUX 和 LP GPIO 交换矩阵可以生成以下中断信号，生成的中断信号只会发送给 LP_CORE。

- LP_GPIO_INTR

表 8.17-2 列出了 LP IO MUX 和 LP GPIO 交换矩阵内部的中断源及其触发条件。

表 8.17-2. LP IO MUX 和 LP GPIO 交换矩阵内部中断源

内部中断源	触发条件	中断信号
LP_GPIO_REG_INTR	LP_GPIO_STATUS_INTERRUPT[*n*] (*n*: 0~6)	LP_GPIO_INTR

8.18 寄存器列表

8.18.1 HP GPIO 交换矩阵寄存器列表

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。请查看章节寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
配置寄存器			
GPIO_STRAP_REG	Strapping 管脚寄存器	0x0000	RO
GPIO_OUT_REG	GPIO 输出寄存器	0x0004	R/ W/ SC/ WTC
GPIO_OUT_W1TS_REG	GPIO 输出置位寄存器	0x0008	WT
GPIO_OUT_W1TC_REG	GPIO 输出清除寄存器	0x000C	WT
GPIO_ENABLE_REG	GPIO 输出使能寄存器	0x0034	R/ W/ WTC
GPIO_ENABLE_W1TS_REG	GPIO 输出使能置位寄存器	0x0038	WT
GPIO_ENABLE_W1TC_REG	GPIO 输出使能清除寄存器	0x003C	WT
GPIO_IN_REG	GPIO 输入寄存器	0x0064	RO
中断状态寄存器			
GPIO_PROCPU_INT_REG	GPIO 的 CPU 中断状态寄存器	0x00A4	RO
GPIO_SDIO_INT_REG	GPIO_SDIO_INT 中断状态寄存器	0x00A8	RO
GPIO_STATUS_REG	GPIO 中断状态寄存器	0x0074	R/ W/ WTC
GPIO_STATUS_W1TS_REG	GPIO 中断状态置位寄存器	0x0078	WT
GPIO_STATUS_W1TC_REG	GPIO 中断状态清除寄存器	0x007C	WT
GPIO_STATUS_NEXT_REG	GPIO 中断源寄存器	0x00C4	RO
管脚配置寄存器			

名称	描述	地址	访问
GPIO_PIN0_REG	GPIO0 管脚配置寄存器	0x00D4	R/W
GPIO_PIN1_REG	GPIO1 管脚配置寄存器	0x00D8	R/W
GPIO_PIN2_REG	GPIO2 管脚配置寄存器	0x00DC	R/W
GPIO_PIN3_REG	GPIO3 管脚配置寄存器	0x00E0	R/W
GPIO_PIN4_REG	GPIO4 管脚配置寄存器	0x00E4	R/W
GPIO_PIN5_REG	GPIO5 管脚配置寄存器	0x00E8	R/W
GPIO_PIN6_REG	GPIO6 管脚配置寄存器	0x00EC	R/W
GPIO_PIN7_REG	GPIO7 管脚配置寄存器	0x00F0	R/W
GPIO_PIN8_REG	GPIO8 管脚配置寄存器	0x00F4	R/W
GPIO_PIN9_REG	GPIO9 管脚配置寄存器	0x00F8	R/W
GPIO_PIN10_REG	GPIO10 管脚配置寄存器	0x00FC	R/W
GPIO_PIN11_REG	GPIO11 管脚配置寄存器	0x0100	R/W
GPIO_PIN12_REG	GPIO12 管脚配置寄存器	0x0104	R/W
GPIO_PIN13_REG	GPIO13 管脚配置寄存器	0x0108	R/W
GPIO_PIN14_REG	GPIO14 管脚配置寄存器	0x010C	R/W
GPIO_PIN15_REG	GPIO15 管脚配置寄存器	0x0110	R/W
GPIO_PIN16_REG	GPIO16 管脚配置寄存器	0x0114	R/W
GPIO_PIN17_REG	GPIO17 管脚配置寄存器	0x0118	R/W
GPIO_PIN18_REG	GPIO18 管脚配置寄存器	0x011C	R/W
GPIO_PIN19_REG	GPIO19 管脚配置寄存器	0x0120	R/W
GPIO_PIN20_REG	GPIO20 管脚配置寄存器	0x0124	R/W
GPIO_PIN21_REG	GPIO21 管脚配置寄存器	0x0128	R/W
GPIO_PIN22_REG	GPIO22 管脚配置寄存器	0x012C	R/W
GPIO_PIN23_REG	GPIO23 管脚配置寄存器	0x0130	R/W
GPIO_PIN24_REG	GPIO24 管脚配置寄存器	0x0134	R/W
GPIO_PIN25_REG	GPIO25 管脚配置寄存器	0x0138	R/W
GPIO_PIN26_REG	GPIO26 管脚配置寄存器	0x013C	R/W
GPIO_PIN27_REG	GPIO27 管脚配置寄存器	0x0140	R/W
GPIO_PIN28_REG	GPIO28 管脚配置寄存器	0x0144	R/W
输入配置寄存器			
GPIO_FUNC6_IN_SEL_CFG_REG	输入信号 6 配置寄存器	0x02EC	R/W
GPIO_FUNC7_IN_SEL_CFG_REG	输入信号 7 配置寄存器	0x02F0	R/W
GPIO_FUNC8_IN_SEL_CFG_REG	输入信号 8 配置寄存器	0x02F4	R/W
GPIO_FUNC9_IN_SEL_CFG_REG	输入信号 9 配置寄存器	0x02F8	R/W
GPIO_FUNC10_IN_SEL_CFG_REG	输入信号 10 配置寄存器	0x02FC	R/W
GPIO_FUNC11_IN_SEL_CFG_REG	输入信号 11 配置寄存器	0x0300	R/W
GPIO_FUNC12_IN_SEL_CFG_REG	输入信号 12 配置寄存器	0x0304	R/W
GPIO_FUNC13_IN_SEL_CFG_REG	输入信号 13 配置寄存器	0x0308	R/W
GPIO_FUNC14_IN_SEL_CFG_REG	输入信号 14 配置寄存器	0x030C	R/W
GPIO_FUNC15_IN_SEL_CFG_REG	输入信号 15 配置寄存器	0x0310	R/W
GPIO_FUNC16_IN_SEL_CFG_REG	输入信号 16 配置寄存器	0x0314	R/W
GPIO_FUNC17_IN_SEL_CFG_REG	输入信号 17 配置寄存器	0x0318	R/W
GPIO_FUNC27_IN_SEL_CFG_REG	输入信号 27 配置寄存器	0x0340	R/W

名称	描述	地址	访问
GPIO_FUNC28_IN_SEL_CFG_REG	输入信号 28 配置寄存器	0x0344	R/W
GPIO_FUNC29_IN_SEL_CFG_REG	输入信号 29 配置寄存器	0x0348	R/W
GPIO_FUNC30_IN_SEL_CFG_REG	输入信号 30 配置寄存器	0x034C	R/W
GPIO_FUNC31_IN_SEL_CFG_REG	输入信号 31 配置寄存器	0x0350	R/W
GPIO_FUNC32_IN_SEL_CFG_REG	输入信号 32 配置寄存器	0x0354	R/W
GPIO_FUNC33_IN_SEL_CFG_REG	输入信号 33 配置寄存器	0x0358	R/W
GPIO_FUNC34_IN_SEL_CFG_REG	输入信号 34 配置寄存器	0x035C	R/W
GPIO_FUNC35_IN_SEL_CFG_REG	输入信号 35 配置寄存器	0x0360	R/W
GPIO_FUNC46_IN_SEL_CFG_REG	输入信号 46 配置寄存器	0x038C	R/W
GPIO_FUNC47_IN_SEL_CFG_REG	输入信号 47 配置寄存器	0x0390	R/W
GPIO_FUNC48_IN_SEL_CFG_REG	输入信号 48 配置寄存器	0x0394	R/W
GPIO_FUNC49_IN_SEL_CFG_REG	输入信号 49 配置寄存器	0x0398	R/W
GPIO_FUNC50_IN_SEL_CFG_REG	输入信号 50 配置寄存器	0x039C	R/W
GPIO_FUNC51_IN_SEL_CFG_REG	输入信号 51 配置寄存器	0x03A0	R/W
GPIO_FUNC52_IN_SEL_CFG_REG	输入信号 52 配置寄存器	0x03A4	R/W
GPIO_FUNC53_IN_SEL_CFG_REG	输入信号 53 配置寄存器	0x03A8	R/W
GPIO_FUNC54_IN_SEL_CFG_REG	输入信号 54 配置寄存器	0x03AC	R/W
GPIO_FUNC55_IN_SEL_CFG_REG	输入信号 55 配置寄存器	0x03B0	R/W
GPIO_FUNC56_IN_SEL_CFG_REG	输入信号 56 配置寄存器	0x03B4	R/W
GPIO_FUNC57_IN_SEL_CFG_REG	输入信号 57 配置寄存器	0x03B8	R/W
GPIO_FUNC58_IN_SEL_CFG_REG	输入信号 58 配置寄存器	0x03BC	R/W
GPIO_FUNC59_IN_SEL_CFG_REG	输入信号 59 配置寄存器	0x03C0	R/W
GPIO_FUNC60_IN_SEL_CFG_REG	输入信号 60 配置寄存器	0x03C4	R/W
GPIO_FUNC61_IN_SEL_CFG_REG	输入信号 61 配置寄存器	0x03C8	R/W
GPIO_FUNC62_IN_SEL_CFG_REG	输入信号 62 配置寄存器	0x03CC	R/W
GPIO_FUNC63_IN_SEL_CFG_REG	输入信号 63 配置寄存器	0x03D0	R/W
GPIO_FUNC64_IN_SEL_CFG_REG	输入信号 64 配置寄存器	0x03D4	R/W
GPIO_FUNC65_IN_SEL_CFG_REG	输入信号 65 配置寄存器	0x03D8	R/W
GPIO_FUNC66_IN_SEL_CFG_REG	输入信号 66 配置寄存器	0x03DC	R/W
GPIO_FUNC70_IN_SEL_CFG_REG	输入信号 70 配置寄存器	0x03EC	R/W
GPIO_FUNC75_IN_SEL_CFG_REG	输入信号 75 配置寄存器	0x0400	R/W
GPIO_FUNC76_IN_SEL_CFG_REG	输入信号 76 配置寄存器	0x0404	R/W
GPIO_FUNC77_IN_SEL_CFG_REG	输入信号 77 配置寄存器	0x0408	R/W
GPIO_FUNC78_IN_SEL_CFG_REG	输入信号 78 配置寄存器	0x040C	R/W
GPIO_FUNC79_IN_SEL_CFG_REG	输入信号 79 配置寄存器	0x0410	R/W
GPIO_FUNC80_IN_SEL_CFG_REG	输入信号 80 配置寄存器	0x0414	R/W
GPIO_FUNC81_IN_SEL_CFG_REG	输入信号 81 配置寄存器	0x0418	R/W
GPIO_FUNC82_IN_SEL_CFG_REG	输入信号 82 配置寄存器	0x041C	R/W
GPIO_FUNC83_IN_SEL_CFG_REG	输入信号 83 配置寄存器	0x0420	R/W
GPIO_FUNC84_IN_SEL_CFG_REG	输入信号 84 配置寄存器	0x0424	R/W
GPIO_FUNC85_IN_SEL_CFG_REG	输入信号 85 配置寄存器	0x0428	R/W
GPIO_FUNC86_IN_SEL_CFG_REG	输入信号 86 配置寄存器	0x042C	R/W
GPIO_FUNC87_IN_SEL_CFG_REG	输入信号 87 配置寄存器	0x0430	R/W

名称	描述	地址	访问
GPIO_FUNC88_IN_SEL_CFG_REG	输入信号 88 配置寄存器	0x0434	R/W
GPIO_FUNC97_IN_SEL_CFG_REG	输入信号 97 配置寄存器	0x0458	R/W
GPIO_FUNC98_IN_SEL_CFG_REG	输入信号 98 配置寄存器	0x045C	R/W
GPIO_FUNC99_IN_SEL_CFG_REG	输入信号 99 配置寄存器	0x0460	R/W
GPIO_FUNC100_IN_SEL_CFG_REG	输入信号 100 配置寄存器	0x0464	R/W
GPIO_FUNC101_IN_SEL_CFG_REG	输入信号 101 配置寄存器	0x0468	R/W
GPIO_FUNC102_IN_SEL_CFG_REG	输入信号 102 配置寄存器	0x046C	R/W
GPIO_FUNC103_IN_SEL_CFG_REG	输入信号 103 配置寄存器	0x0470	R/W
GPIO_FUNC104_IN_SEL_CFG_REG	输入信号 104 配置寄存器	0x0474	R/W
GPIO_FUNC105_IN_SEL_CFG_REG	输入信号 105 配置寄存器	0x0478	R/W
GPIO_FUNC106_IN_SEL_CFG_REG	输入信号 106 配置寄存器	0x047C	R/W
GPIO_FUNC107_IN_SEL_CFG_REG	输入信号 107 配置寄存器	0x0480	R/W
GPIO_FUNC108_IN_SEL_CFG_REG	输入信号 108 配置寄存器	0x0484	R/W
GPIO_FUNC109_IN_SEL_CFG_REG	输入信号 109 配置寄存器	0x0488	R/W
GPIO_FUNC110_IN_SEL_CFG_REG	输入信号 110 配置寄存器	0x048C	R/W
GPIO_FUNC111_IN_SEL_CFG_REG	输入信号 111 配置寄存器	0x0490	R/W
GPIO_FUNC112_IN_SEL_CFG_REG	输入信号 112 配置寄存器	0x0494	R/W
GPIO_FUNC113_IN_SEL_CFG_REG	输入信号 113 配置寄存器	0x0498	R/W
GPIO_FUNC114_IN_SEL_CFG_REG	输入信号 114 配置寄存器	0x049C	R/W
GPIO_FUNC115_IN_SEL_CFG_REG	输入信号 115 配置寄存器	0x04A0	R/W
GPIO_FUNC116_IN_SEL_CFG_REG	输入信号 116 配置寄存器	0x04A4	R/W
输出配置寄存器			
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 输出配置寄存器	0x0AD4	varies
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 输出配置寄存器	0x0AD8	varies
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 输出配置寄存器	0x0ADC	varies
GPIO_FUNC3_OUT_SEL_CFG_REG	GPIO3 输出配置寄存器	0x0AE0	varies
GPIO_FUNC4_OUT_SEL_CFG_REG	GPIO4 输出配置寄存器	0x0AE4	varies
GPIO_FUNC5_OUT_SEL_CFG_REG	GPIO5 输出配置寄存器	0x0AE8	varies
GPIO_FUNC6_OUT_SEL_CFG_REG	GPIO6 输出配置寄存器	0x0AEC	varies
GPIO_FUNC7_OUT_SEL_CFG_REG	GPIO7 输出配置寄存器	0x0AF0	varies
GPIO_FUNC8_OUT_SEL_CFG_REG	GPIO8 输出配置寄存器	0x0AF4	varies
GPIO_FUNC9_OUT_SEL_CFG_REG	GPIO9 输出配置寄存器	0x0AF8	varies
GPIO_FUNC10_OUT_SEL_CFG_REG	GPIO10 输出配置寄存器	0x0AFC	varies
GPIO_FUNC11_OUT_SEL_CFG_REG	GPIO11 输出配置寄存器	0x0B00	varies
GPIO_FUNC12_OUT_SEL_CFG_REG	GPIO12 输出配置寄存器	0x0B04	varies
GPIO_FUNC13_OUT_SEL_CFG_REG	GPIO13 输出配置寄存器	0x0B08	varies
GPIO_FUNC14_OUT_SEL_CFG_REG	GPIO14 输出配置寄存器	0x0B0C	varies
GPIO_FUNC23_OUT_SEL_CFG_REG	GPIO23 输出配置寄存器	0x0B30	varies
GPIO_FUNC24_OUT_SEL_CFG_REG	GPIO24 输出配置寄存器	0x0B34	varies
GPIO_FUNC25_OUT_SEL_CFG_REG	GPIO25 输出配置寄存器	0x0B38	varies
GPIO_FUNC26_OUT_SEL_CFG_REG	GPIO26 输出配置寄存器	0x0B3C	varies
GPIO_FUNC27_OUT_SEL_CFG_REG	GPIO27 输出配置寄存器	0x0B40	varies
GPIO_FUNC28_OUT_SEL_CFG_REG	GPIO28 输出配置寄存器	0x0B44	varies

名称	描述	地址	访问
时钟门控寄存器			
GPIO_CLOCK_GATE_REG	GPIO 时钟门控寄存器	0x0DF8	R/W
版本寄存器			
GPIO_DATE_REG	版本控制寄存器	0x0DFC	R/W

8.18.2 HP IO MUX 寄存器列表

本小节的所有地址均为相对于 HP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
配置寄存器			
IO_MUX_GPIO0_REG	GPIO0 的 HP IO MUX 配置寄存器	0x0000	R/W
IO_MUX_GPIO1_REG	GPIO1 的 HP IO MUX 配置寄存器	0x0004	R/W
IO_MUX_GPIO2_REG	GPIO2 的 HP IO MUX 配置寄存器	0x0008	R/W
IO_MUX_GPIO3_REG	GPIO3 的 HP IO MUX 配置寄存器	0x000C	R/W
IO_MUX_GPIO4_REG	GPIO4 的 HP IO MUX 配置寄存器	0x0010	R/W
IO_MUX_GPIO5_REG	GPIO5 的 HP IO MUX 配置寄存器	0x0014	R/W
IO_MUX_GPIO6_REG	GPIO6 的 HP IO MUX 配置寄存器	0x0018	R/W
IO_MUX_GPIO7_REG	GPIO7 的 HP IO MUX 配置寄存器	0x001C	R/W
IO_MUX_GPIO8_REG	GPIO8 的 HP IO MUX 配置寄存器	0x0020	R/W
IO_MUX_GPIO9_REG	GPIO9 的 HP IO MUX 配置寄存器	0x0024	R/W
IO_MUX_GPIO10_REG	GPIO10 的 HP IO MUX 配置寄存器	0x0028	R/W
IO_MUX_GPIO11_REG	GPIO11 的 HP IO MUX 配置寄存器	0x002C	R/W
IO_MUX_GPIO12_REG	GPIO12 的 HP IO MUX 配置寄存器	0x0030	R/W
IO_MUX_GPIO13_REG	GPIO13 的 HP IO MUX 配置寄存器	0x0034	R/W
IO_MUX_GPIO14_REG	GPIO14 的 HP IO MUX 配置寄存器	0x0038	R/W
IO_MUX_GPIO23_REG	GPIO23 的 HP IO MUX 配置寄存器	0x005C	R/W
IO_MUX_GPIO24_REG	GPIO24 的 HP IO MUX 配置寄存器	0x0060	R/W
IO_MUX_GPIO25_REG	GPIO25 的 HP IO MUX 配置寄存器	0x0064	R/W
IO_MUX_GPIO26_REG	GPIO26 的 HP IO MUX 配置寄存器	0x0068	R/W
IO_MUX_GPIO27_REG	GPIO27 的 HP IO MUX 配置寄存器	0x006C	R/W
IO_MUX_GPIO28_REG	GPIO28 的 HP IO MUX 配置寄存器	0x0070	R/W
版本寄存器			
IO_MUX_DATE_REG	版本控制寄存器	0x01FC	R/W

8.18.3 GPIO EXT 寄存器列表

GPIO EXT 寄存器包括 SDM 寄存器、毛刺滤波器寄存器和 ETM 寄存器。

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址 + 0x0F00 的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
SDM 配置寄存器			
GPIO_EXT_SIGMADELTA_MISC_REG	MISC 寄存器	0x0004	R/W
GPIO_EXT_SIGMADELTA0_REG	SDM 通道 0 的占空比配置寄存器	0x0008	R/W
GPIO_EXT_SIGMADELTA1_REG	SDM 通道 1 的占空比配置寄存器	0x000C	R/W
GPIO_EXT_SIGMADELTA2_REG	SDM 通道 2 的占空比配置寄存器	0x0010	R/W
GPIO_EXT_SIGMADELTA3_REG	SDM 通道 3 的占空比配置寄存器	0x0014	R/W
配置寄存器			
GPIO_EXT_PAD_COMP_CONFIG_0_REG	跨零检测配置寄存器	0x0058	R/W
GPIO_EXT_PAD_COMP_FILTER_0_REG	配置跨零检测的中断源屏蔽周期	0x005C	R/W
毛刺滤波器配置寄存器			
GPIO_EXT_GLITCH_FILTER_CH0_REG	毛刺滤波器通道 0 的配置寄存器	0x00D8	R/W
GPIO_EXT_GLITCH_FILTER_CH1_REG	毛刺滤波器通道 1 的配置寄存器	0x00DC	R/W
GPIO_EXT_GLITCH_FILTER_CH2_REG	毛刺滤波器通道 2 的配置寄存器	0x00E0	R/W
GPIO_EXT_GLITCH_FILTER_CH3_REG	毛刺滤波器通道 3 的配置寄存器	0x00E4	R/W
GPIO_EXT_GLITCH_FILTER_CH4_REG	毛刺滤波器通道 4 的配置寄存器	0x00E8	R/W
GPIO_EXT_GLITCH_FILTER_CH5_REG	毛刺滤波器通道 5 的配置寄存器	0x00EC	R/W
GPIO_EXT_GLITCH_FILTER_CH6_REG	毛刺滤波器通道 6 的配置寄存器	0x00F0	R/W
GPIO_EXT_GLITCH_FILTER_CH7_REG	毛刺滤波器通道 7 的配置寄存器	0x00F4	R/W
ETM 配置寄存器			
GPIO_EXT_ETM_EVENT_CH0_CFG_REG	ETM 通道 0 的配置寄存器	0x0118	R/W
GPIO_EXT_ETM_EVENT_CH1_CFG_REG	ETM 通道 1 的配置寄存器	0x011C	R/W
GPIO_EXT_ETM_EVENT_CH2_CFG_REG	ETM 通道 2 的配置寄存器	0x0120	R/W
GPIO_EXT_ETM_EVENT_CH3_CFG_REG	ETM 通道 3 的配置寄存器	0x0124	R/W
GPIO_EXT_ETM_EVENT_CH4_CFG_REG	ETM 通道 4 的配置寄存器	0x0128	R/W
GPIO_EXT_ETM_EVENT_CH5_CFG_REG	ETM 通道 5 的配置寄存器	0x012C	R/W
GPIO_EXT_ETM_EVENT_CH6_CFG_REG	ETM 通道 6 的配置寄存器	0x0130	R/W
GPIO_EXT_ETM_EVENT_CH7_CFG_REG	ETM 通道 7 的配置寄存器	0x0134	R/W
GPIO_EXT_ETM_TASK_P0_CFG_REG	ETM 的 GPIO 选择配置寄存器 0	0x0158	R/W
GPIO_EXT_ETM_TASK_P1_CFG_REG	ETM 的 GPIO 选择配置寄存器 1	0x015C	R/W
GPIO_EXT_ETM_TASK_P2_CFG_REG	ETM 的 GPIO 选择配置寄存器 2	0x0160	R/W
GPIO_EXT_ETM_TASK_P3_CFG_REG	ETM 的 GPIO 选择配置寄存器 3	0x0164	R/W
GPIO_EXT_ETM_TASK_P4_CFG_REG	ETM 的 GPIO 选择配置寄存器 4	0x0168	R/W
GPIO_EXT_ETM_TASK_P5_CFG_REG	ETM 的 GPIO 选择配置寄存器 5	0x016C	R/W
中断寄存器			
GPIO_EXT_INT_RAW_REG	GPIO_EXT 的原始中断寄存器	0x01D0	RO/ WTC/ SS
GPIO_EXT_INT_ST_REG	GPIO_EXT 的屏蔽中断寄存器	0x01D4	RO
GPIO_EXT_INT_ENA_REG	GPIO_EXT 的中断使能寄存器	0x01D8	R/W
GPIO_EXT_INT_CLR_REG	GPIO_EXT 的中断清除寄存器	0x01DC	WT
版本寄存器			
GPIO_EXT_VERSION_REG	版本控制寄存器	0x01FC	R/W

8.18.4 LP GPIO 交换矩阵寄存器列表

本小节的所有地址均为相对于 LP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。请查看章节寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_GPIO_OUT_REG	LP GPIO 输出寄存器	0x0004	R/ W/ WTC
LP_GPIO_OUT_W1TS_REG	LP GPIO 输出置位寄存器	0x0008	WT
LP_GPIO_OUT_W1TC_REG	LP GPIO 输出清除寄存器	0x000C	WT
LP_GPIO_ENABLE_REG	LP GPIO 输出使能寄存器	0x0010	R/ W/ WTC
LP_GPIO_ENABLE_W1TS_REG	LP GPIO 输出使能置位寄存器	0x0014	WT
LP_GPIO_ENABLE_W1TC_REG	LP GPIO 输出使能清除寄存器	0x0018	WT
LP_GPIO_IN_REG	LP GPIO 输入寄存器	0x001C	RO
LP_GPIO_STATUS_REG	LP GPIO 中断状态寄存器	0x0020	R/ W/ WTC
LP_GPIO_STATUS_W1TS_REG	LP GPIO 中断状态置位寄存器	0x0024	WT
LP_GPIO_STATUS_W1TC_REG	LP GPIO 中断状态清除寄存器	0x0028	WT
LP_GPIO_STATUS_NEXT_REG	LP GPIO 中断源寄存器	0x002C	RO
LP_GPIO_PIN0_REG	LP GPIO0 配置寄存器	0x0030	varies
LP_GPIO_PIN1_REG	LP GPIO1 配置寄存器	0x0034	varies
LP_GPIO_PIN2_REG	LP GPIO2 配置寄存器	0x0038	varies
LP_GPIO_PIN3_REG	LP GPIO3 配置寄存器	0x003C	varies
LP_GPIO_PIN4_REG	LP GPIO4 配置寄存器	0x0040	varies
LP_GPIO_PIN5_REG	LP GPIO5 配置寄存器	0x0044	varies
LP_GPIO_PIN6_REG	LP GPIO6 配置寄存器	0x0048	varies
LP_GPIO_PIN7_REG	LP GPIO7 配置寄存器	0x004C	varies
LP_GPIO_FUNC0_OUT_SEL_CFG_REG	LP GPIO0 的输出配置寄存器	0x02B0	R/W
LP_GPIO_FUNC1_OUT_SEL_CFG_REG	LP GPIO1 的输出配置寄存器	0x02B4	R/W
LP_GPIO_FUNC2_OUT_SEL_CFG_REG	LP GPIO2 的输出配置寄存器	0x02B8	R/W
LP_GPIO_FUNC3_OUT_SEL_CFG_REG	LP GPIO3 的输出配置寄存器	0x02BC	R/W
LP_GPIO_FUNC4_OUT_SEL_CFG_REG	LP GPIO4 的输出配置寄存器	0x02C0	R/W
LP_GPIO_FUNC5_OUT_SEL_CFG_REG	LP GPIO5 的输出配置寄存器	0x02C4	R/W
LP_GPIO_FUNC6_OUT_SEL_CFG_REG	LP GPIO6 的输出配置寄存器	0x02C8	R/W
LP_GPIO_FUNC7_OUT_SEL_CFG_REG	LP GPIO7 的输出配置寄存器	0x02CC	R/W
时钟门控寄存器			
LP_GPIO_CLOCK_GATE_REG	LP GPIO 时钟门控寄存器	0x03F8	R/W
版本寄存器			
LP_GPIO_DATE_REG	版本控制寄存器	0x03FC	R/W

8.18.5 LP IO MUX 寄存器列表

本小节的所有地址均为相对于 LP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_IO_MUX_GPIO0_REG	GPIO0 的 LP IO MUX 配置寄存器	0x0000	R/W
LP_IO_MUX_GPIO1_REG	GPIO1 的 LP IO MUX 配置寄存器	0x0004	R/W
LP_IO_MUX_GPIO2_REG	GPIO2 的 LP IO MUX 配置寄存器	0x0008	R/W
LP_IO_MUX_GPIO3_REG	GPIO3 的 LP IO MUX 配置寄存器	0x000C	R/W
LP_IO_MUX_GPIO4_REG	GPIO4 的 LP IO MUX 配置寄存器	0x0010	R/W
LP_IO_MUX_GPIO5_REG	GPIO5 的 LP IO MUX 配置寄存器	0x0014	R/W
LP_IO_MUX_GPIO6_REG	GPIO6 的 LP IO MUX 配置寄存器	0x0018	R/W
版本寄存器			
LP_IO_MUX_DATE_REG	版本控制寄存器	0x01FC	R/W

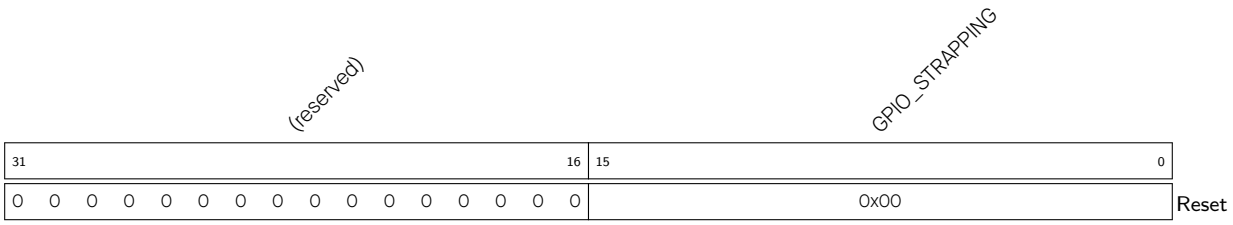
8.19 寄存器

8.19.1 HP GPIO 交换矩阵寄存器

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 8.1. GPIO_STRAP_REG (0x0000)



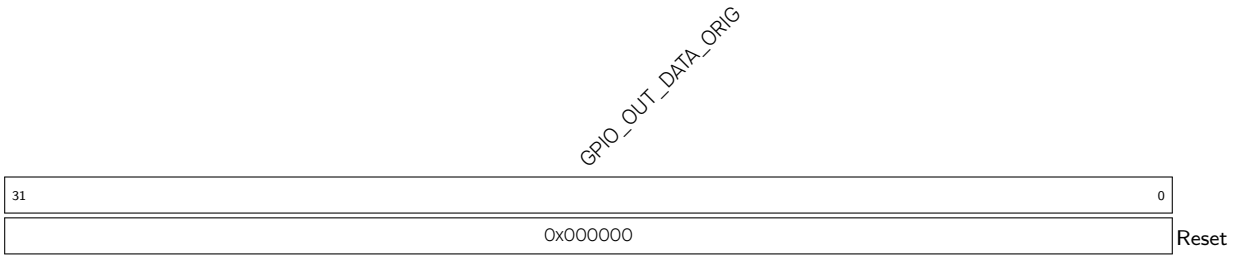
GPIO_STRAPPING 表示 strapping 管脚的值。

- Bit[0]: GPIO26
- Bit[1]: MTMS
- Bit[2]: MTDI
- Bit[3]: GPIO27
- Bit[4]: GPIO28
- Bit[5]: GPIO7
- Bit[6]: GPIO25
- Bit[7]~bit[15]：无效

更多有关 strapping 管脚控制的功能，见章节 10 芯片 Boot 控制。

(RO)

Register 8.2. GPIO_OUT_REG (0x0004)



GPIO_OUT_DATA_ORIG 配置简单输出模式下 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出值。

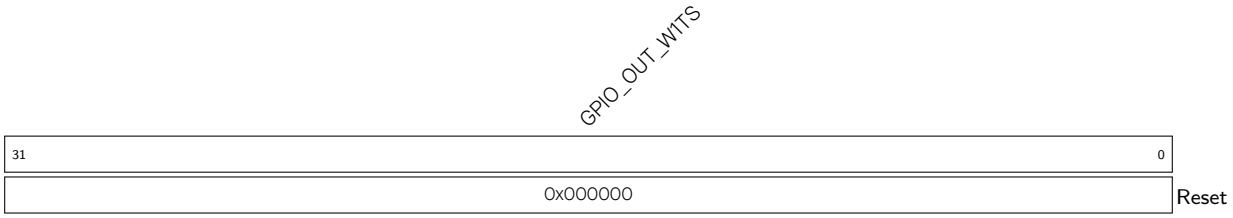
0：低电平

1：高电平

bit[0]~bit[14] 和 bit[23]~bit[28] 的值分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出值。

(R/W/SC/WTC)

Register 8.3. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS 配置是否置位 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出寄存器 GPIO_OUT_REG。

0：不置位

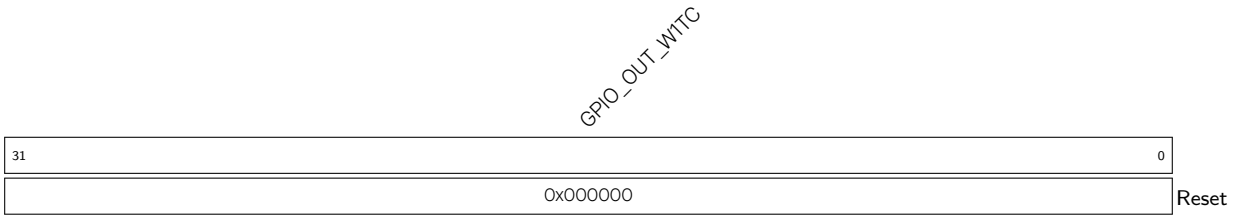
1：GPIO_OUT_REG 中相应位也将置 1

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来置位 GPIO_OUT_REG。

(WT)

Register 8.4. GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC 配置是否清零 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出寄存器 GPIO_OUT_REG。

0：不清零

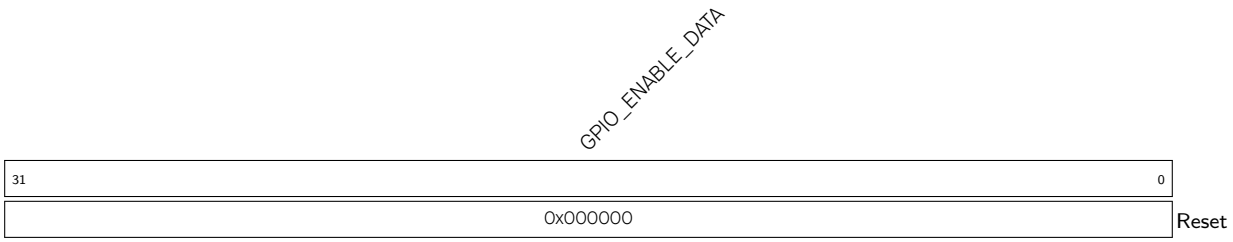
1：GPIO_OUT_REG 中相应位将被清零

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来清零 GPIO_OUT_REG。

(WT)

Register 8.5. GPIO_ENABLE_REG (0x0034)



GPIO_ENABLE_DATA 配置是否使能 GPIO0~GPIO14 和 GPIO23~GPIO28 输出。

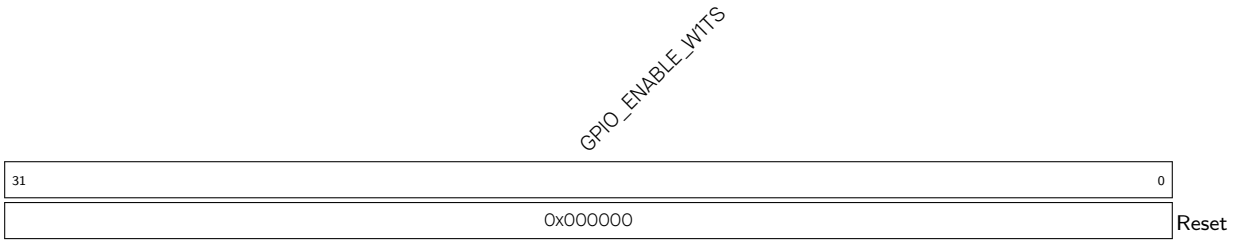
0：不使能

1：使能

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

(R/W/WTC)

Register 8.6. GPIO_ENABLE_W1TS_REG (0x0038)



GPIO_ENABLE_W1TS 配置是否置位 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出使能寄存器 GPIO_ENABLE_REG。

0：不置位

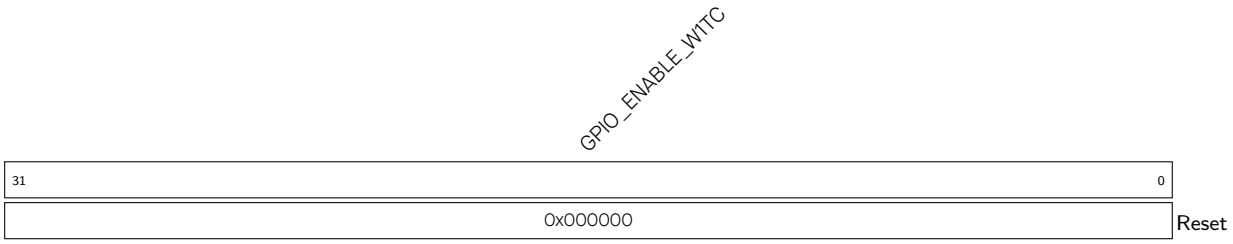
1：GPIO_ENABLE_REG 中相应位也将置 1

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用该寄存器来置位 GPIO_ENABLE_REG。

(WT)

Register 8.7. GPIO_ENABLE_W1TC_REG (0x003C)



GPIO_ENABLE_W1TC 配置是否清零 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出使能寄存器 GPIO_ENABLE_REG。

0：不清零

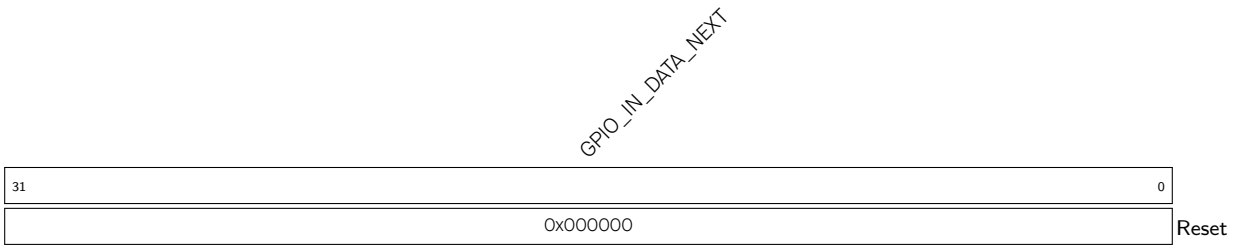
1：GPIO_ENABLE_REG 中相应位将被清零

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用该寄存器来清零 GPIO_ENABLE_REG。

(WT)

Register 8.8. GPIO_IN_REG (0x0064)



GPIO_IN_DATA_NEXT 表示 GPIO0~GPIO14 和 GPIO23~GPIO28 的输入值。

每一位均代表一个管脚的输入值:

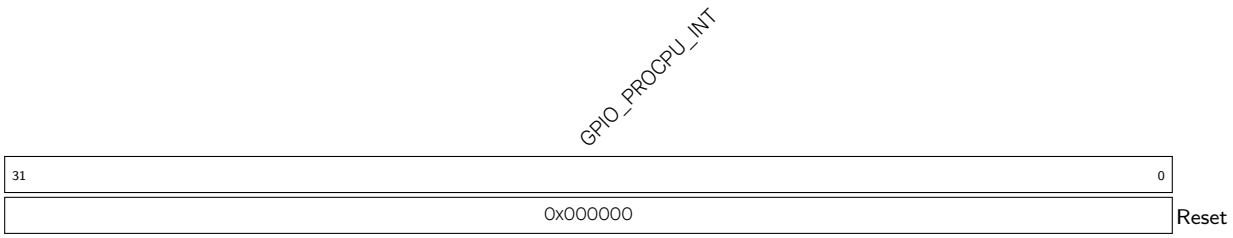
0：低电平

1：高电平

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

(RO)

Register 8.9. GPIO_PROCPU_INT_REG (0x00A4)



GPIO_PROCPU_INT 表示 GPIO0~GPIO14 和 GPIO23~GPIO28 的 CPU 中断状态。

每位代表:

0：表示 CPU 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

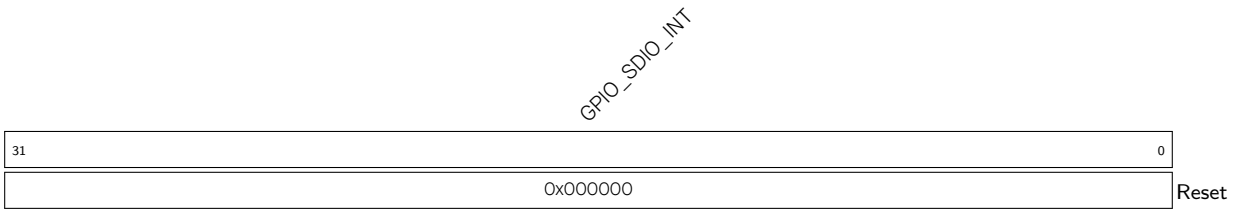
1：表示 CPU 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

如果 GPIO_PIN*n*_REG 中 bit[13] 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS_REG 中相应位的中断状态一致。

(RO)

Register 8.10. GPIO_SDIO_INT_REG (0x00A8)



GPIO_SDIO_INT 表示 GPIO0~GPIO14 和 GPIO23~GPIO28 的 GPIO_SDIO_INT 中断状态。

每位代表：

0：表示 GPIO_SDIO_INT 中断未使能，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断。

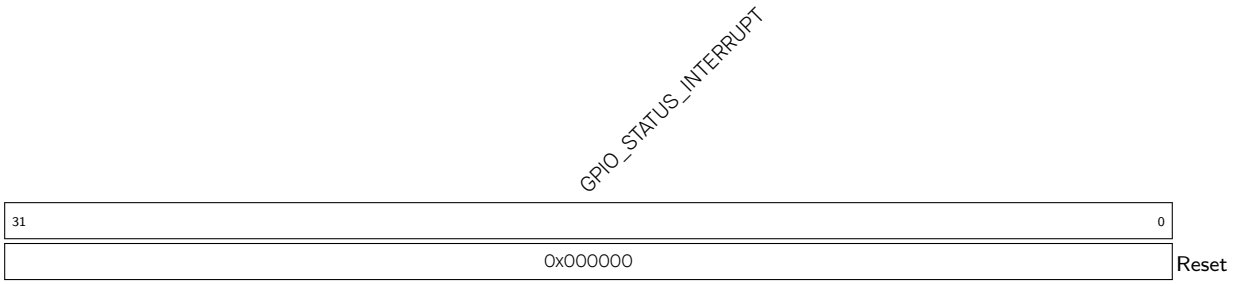
1：表示 GPIO_SDIO_INT 中断使能后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断。

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

如果 GPIO_PIN*n*_REG 中 bit[15] 高电平有效，即使能 GPIO_SDIO_INT 中断，则此寄存器所示的中断状态应与 GPIO_STATUS_REG 中相应位的中断状态一致。

(RO)

Register 8.11. GPIO_STATUS_REG (0x0074)



GPIO_STATUS_INTERRUPT GPIO0~GPIO14 和 GPIO23~GPIO28 的中断状态寄存器，软件可配置该寄存器的值。

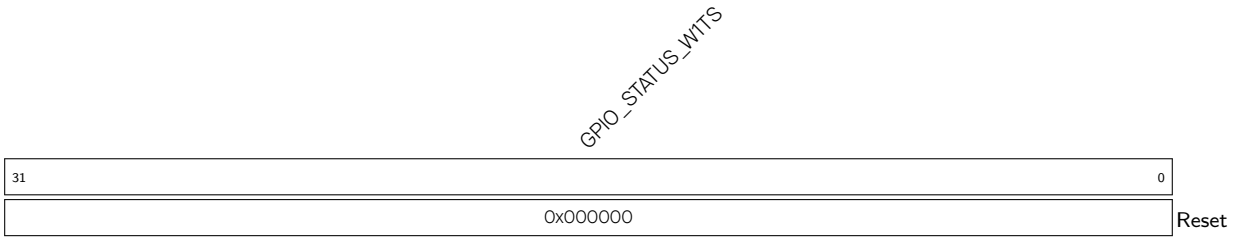
bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

每一位均代表对应 GPIO 的中断状态：

- 0：表示对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断，或软件配置该位寄存器的值为 0。
- 1：表示对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断，或软件配置该位寄存器的值为 1。

(R/W/WTC)

Register 8.12. GPIO_STATUS_W1TS_REG (0x0078)



GPIO_STATUS_W1TS 配置是否置位 GPIO0~GPIO14 和 GPIO23~GPIO28 的中断状态寄存器 GPIO_STATUS_INTERRUPT。

每一位的值可以为：

0：不置位

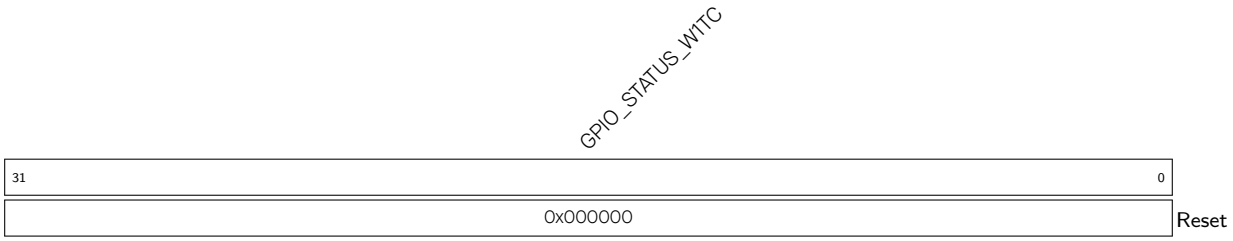
1：GPIO_STATUS_INTERRUPT 中的相应位也将置 1

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来置位 GPIO_STATUS_INTERRUPT。

(WT)

Register 8.13. GPIO_STATUS_W1TC_REG (0x007C)



GPIO_STATUS_W1TC 配置是否清零 GPIO0~GPIO14 和 GPIO23~GPIO28 的中断状态寄存器 GPIO_STATUS_INTERRUPT。

每一位的值可以为：

0：不清零

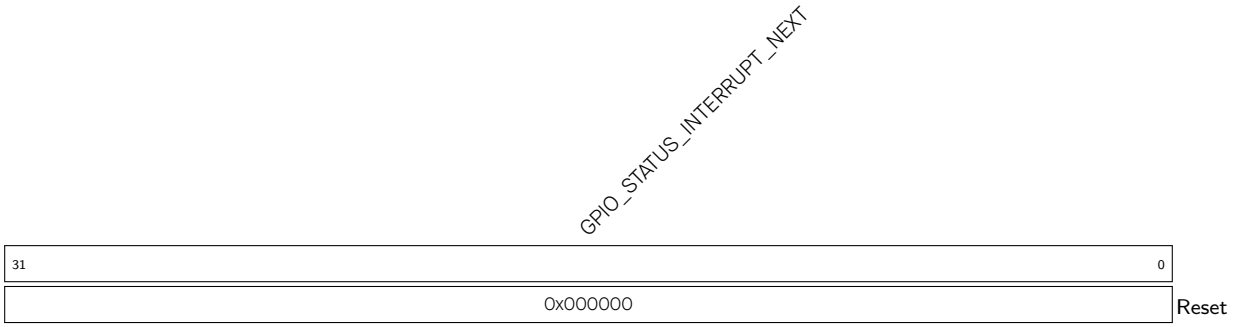
1：GPIO_STATUS_INTERRUPT 中的相应位将被清零

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来清零 GPIO_STATUS_INTERRUPT。

(WT)

Register 8.14. GPIO_STATUS_NEXT_REG (0x00C4)



GPIO_STATUS_INTERRUPT_NEXT 表示 GPIO0~GPIO14 和 GPIO23~GPIO28 的中断源信号。

每位代表：

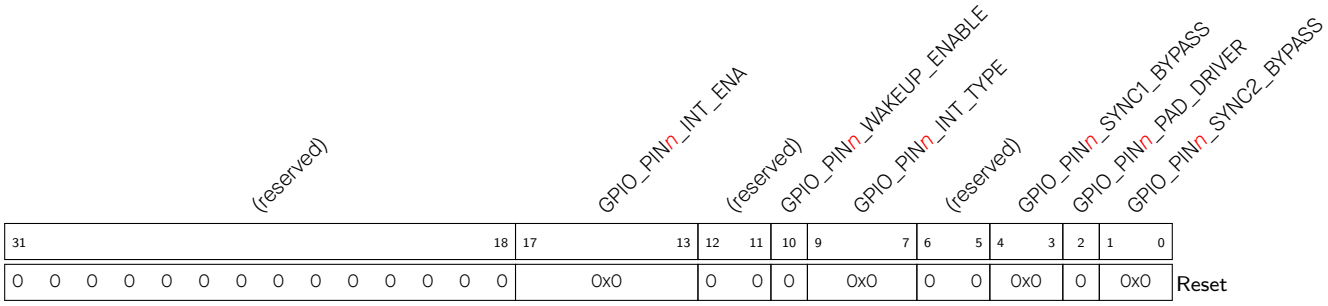
0：表示 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所配置的中断

1：GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所配置的中断

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。

(RO)

Register 8.15. GPIO_PIN*n*_REG (*n*: 0-14, 23-28) (0x00D4+0x4**n*)



GPIO_PIN*n*_SYNC2_BYPASS 配置是否使能 GPIO 输入数据在第二级同步中 HP IO MUX 运行时钟上升沿同步或下降沿同步。

0：不同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

GPIO_PIN*n*_PAD_DRIVER 配置选择管脚的输出驱动模式。

0：正常输出

1：开漏输出

(R/W)

GPIO_PIN*n*_SYNC1_BYPASS 配置是否使能 GPIO 输入数据在第一级同步中 HP IO MUX 运行时钟上升沿同步或下降沿同步。

0：不同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

GPIO_PIN*n*_INT_TYPE 配置 GPIO 中断类型。

0：关闭 GPIO 中断

1：上升沿触发

2：下降沿触发

3：任一沿触发

4：低电平触发

5：高电平触发

(R/W)

GPIO_PIN*n*_WAKEUP_ENABLE 配置是否使能 GPIO 唤醒功能。

0：不使能

1：使能

该功能只能将 CPU 从 Light-sleep 中唤醒。

(R/W)

见下页

Register 8.15. GPIO_PIN*n*_REG (*n*: 0-14, 23-28) (0x00D4+0x4**n*)

接上页

GPIO_PIN*n*_INT_ENA 配置是否使能 CPU 中断或 CPU 非屏蔽中断。

- bit[13]：配置是否使能 CPU 中断：

0：不使能

1：使能

- bit[14]：无效

- bit[15]：配置是否使能 GPIO_SDIO_INT 中断：

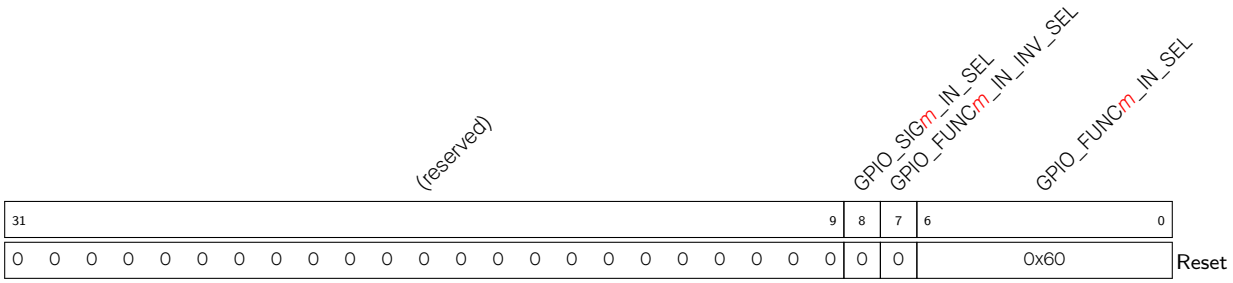
0：不使能

1：使能

- bit[16]~bit[17]：无效

(R/W)

Register 8.16. GPIO_FUNC*m*_IN_SEL_CFG_REG (*m*: 6-17) (0x02EC+0x4*(*m*-6))



GPIO_FUNC*m*_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 *m*。

0x0：选择 GPIO0

0x1：选择 GPIO1

.....

0x1B：选择 GPIO27

0x1C：选择 GPIO28

Or

0x40：选择恒高电平输入

0x60：选择恒低电平输入

(R/W)

GPIO_FUNC*m*_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

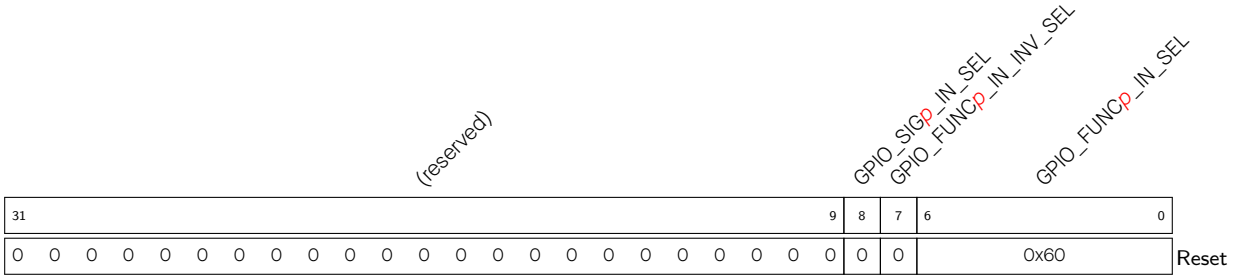
GPIO_SIG*m*_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0：旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设

1：通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 8.17. GPIO_FUNC*p*_IN_SEL_CFG_REG (*p*: 27-35) (0x0340+0x4*(*p*-27))



GPIO_FUNC*p*_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 *p*。

0x0：选择 GPIO0

0x1：选择 GPIO1

.....

0x1B：选择 GPIO27

0x1C：选择 GPIO28

Or

0x40：选择恒高电平输入

0x60：选择恒低电平输入

(R/W)

GPIO_FUNC*p*_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

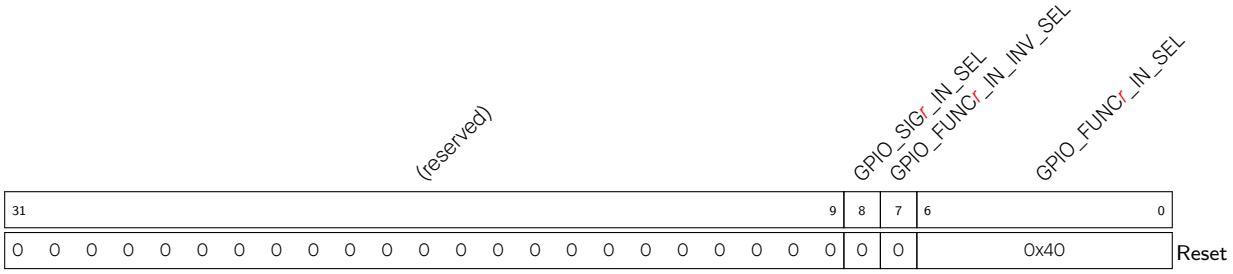
GPIO_SIG*p*_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0：旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设

1：通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 8.18. GPIO_FUNCr_IN_SEL_CFG_REG (r: 46-66) (0x038C+0x4*(r-46))



GPIO_FUNCr_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 r。

0x0：选择 GPIO0

0x1：选择 GPIO1

.....

0x1B：选择 GPIO27

0x1C：选择 GPIO28

Or

0x40：选择恒高电平输入

0x60：选择恒低电平输入

(R/W)

GPIO_FUNCr_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

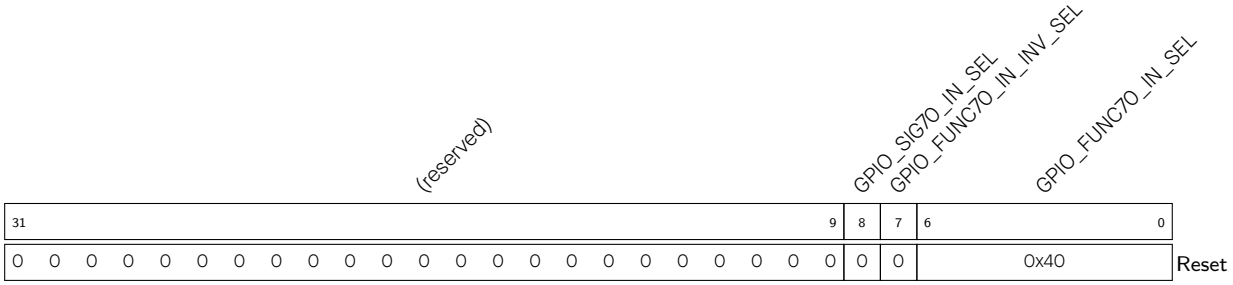
GPIO_SIGr_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0：旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设

1：通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 8.19. GPIO_FUNC70_IN_SEL_CFG_REG (0x03EC)



GPIO_FUNC70_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 70。

0x0：选择 GPIO0

0x1：选择 GPIO1

.....

0x1B：选择 GPIO27

0x1C：选择 GPIO28

Or

0x40：选择恒高电平输入

0x60：选择恒低电平输入

(R/W)

GPIO_FUNC70_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

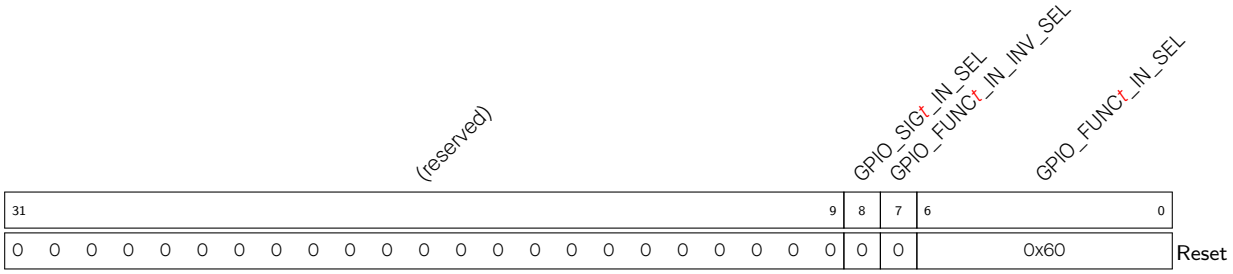
GPIO_SIG70_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0：旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设

1：通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 8.20. GPIO_FUNC*t*_IN_SEL_CFG_REG (*t*: 76-88) (0x0404+0x4*(*t*-76))



GPIO_FUNC*t*_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 *t*。

0x0：选择 GPIO0

0x1：选择 GPIO1

.....

0x1B：选择 GPIO27

0x1C：选择 GPIO28

Or

0x40：选择恒高电平输入

0x60：选择恒低电平输入

(R/W)

GPIO_FUNC*t*_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

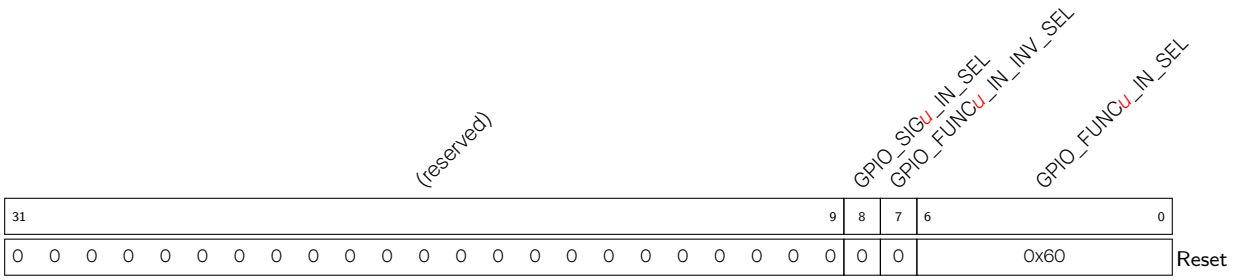
GPIO_SIG*t*_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0：旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设

1：通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 8.21. GPIO_FUNC*u*_IN_SEL_CFG_REG (*u*: 97-116) (0x0458+0x4*(*u*-97))



GPIO_FUNC*u*_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 *u*。

0x0：选择 GPIO0

0x1：选择 GPIO1

.....

0x1B：选择 GPIO27

0x1C：选择 GPIO28

Or

0x40：选择恒高电平输入

0x60：选择恒低电平输入

(R/W)

GPIO_FUNC*u*_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

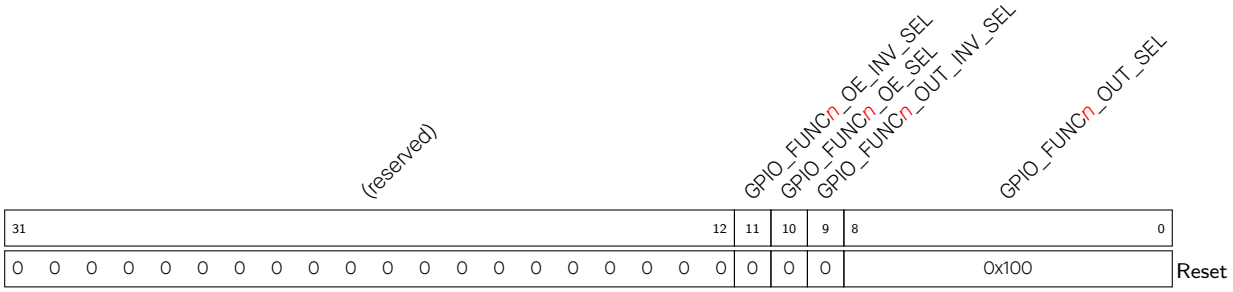
GPIO_SIG*u*_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0：旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设

1：通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 8.22. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-14, 23-28) (0x0AD4+0x4**n*)



GPIO_FUNC*n*_OUT_SEL 配置从外设信号中选择一个信号 Y (0 <= Y < 256) 输出至 GPIO*n*。

0：选择信号 0

1：选择信号 1

.....

254：选择信号 254

255：选择信号 255

256：GPIO_OUT_REG 和 GPIO_ENABLE_REG 中的 bit[*n*] 将用作输出值和输出使能信号。

信号列表见表 8.12-1。

(R/W/SC)

GPIO_FUNC*n*_OUT_INV_SEL 配置是否反转输出值。

0：不反转

1：反转

(R/W/SC)

GPIO_FUNC*n*_OE_SEL 配置选择输出使能信号。

0：使用外设的输出使能信号

1：强制使用 GPIO_ENABLE_REG 中的 bit[*n*] 用作输出使能信号。

(R/W)

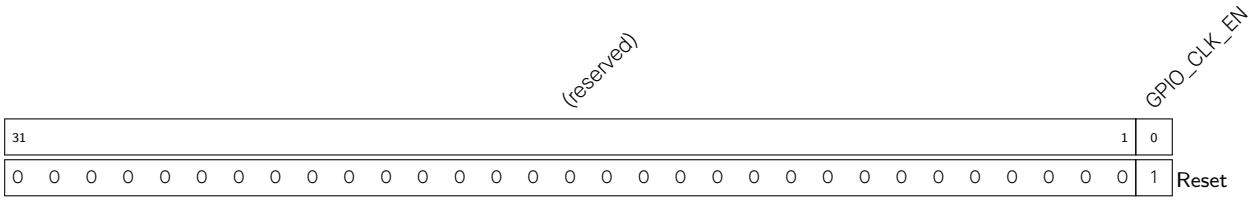
GPIO_FUNC*n*_OE_INV_SEL 配置是否反转输出使能信号。

0：不反转

1：反转

(R/W)

Register 8.23. GPIO_CLOCK_GATE_REG (0x0DF8)



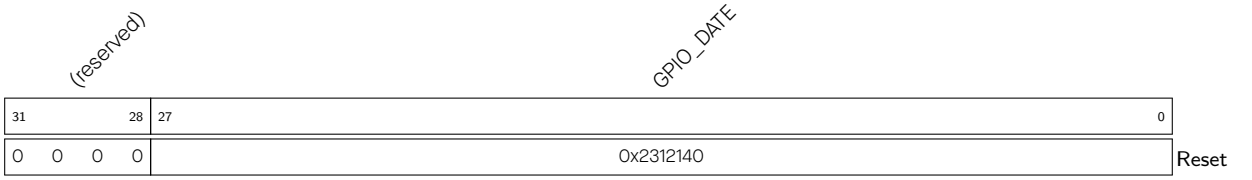
GPIO_CLK_EN 配置是否使能时钟门控。

0：不使能

1：此位置 1，则时钟自由运转。

(R/W)

Register 8.24. GPIO_DATE_REG (0x0DFC)



GPIO_DATE 版本控制寄存器。

(R/W)

8.19.2 HP IO MUX 寄存器

本小节的所有地址均为相对于 HP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 8.25. IO_MUX_GPIO*n*_REG (*n*: 0-14, 23-28) (0x0000+0x4**n*)



IO_MUX_GPIO*n*_MCU_OE 配置是否使能睡眠模式下 GPIO*n* 的输出。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_SLP_SEL 配置是否使能 GPIO*n* 进入睡眠模式。

0：不进入

1：进入

(R/W)

IO_MUX_GPIO*n*_MCU_WPD 配置是否使能睡眠模式下 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_WPU 配置是否使能睡眠模式下 GPIO*n* 的上拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_IE 配置是否使能睡眠模式下 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_DRV 配置睡眠模式下 GPIO*n* 的驱动强度。

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

IO_MUX_GPIO*n*_FUN_WPD 配置是否使能 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

见下页

Register 8.25. IO_MUX_GPIO*n*_REG (*n*: 0-14, 23-28) (0x0000+0x4**n*)

接上页

IO_MUX_GPIO*n*_FUN_WPU 配置是否使能 GPIO*n* 的上拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_FUN_IE 配置是否使能 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_FUN_DRV 配置 GPIO*n* 的驱动强度。

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

IO_MUX_GPIO*n*_MCU_SEL 配置选择管脚的 HP IO MUX 功能。

0：选择功能 0

1：选择功能 1

.....

(R/W)

IO_MUX_GPIO*n*_FILTER_EN 配置是否使能管脚输入信号的滤波功能。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_HYS_EN 当 IO_MUX_GPIO*n*_HYS_SEL 设置为 1 时，配置是否使能管脚的迟滞功能。

0：不使能

1：使能

(R/W)

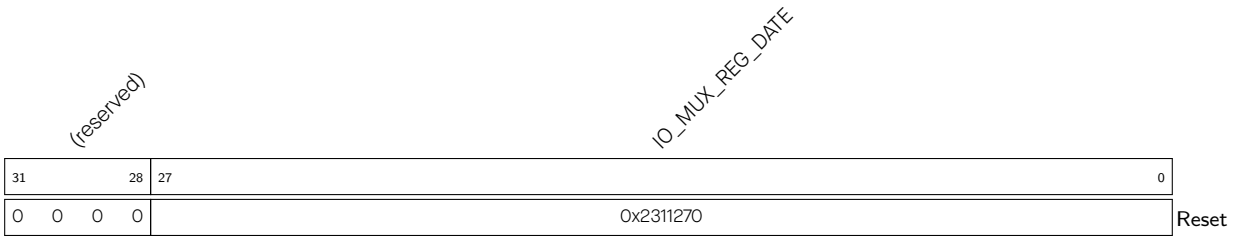
IO_MUX_GPIO*n*_HYS_SEL 配置选择 GPIO*n* 迟滞功能的使能信号。

0：使用 eFuse 的输出使能信号

1：使用 IO_MUX_GPIO*n*_HYS_EN 用作输出使能信号

(R/W)

Register 8.26. IO_MUX_DATE_REG (0x01FC)



IO_MUX_REG_DATE 版本控制寄存器。

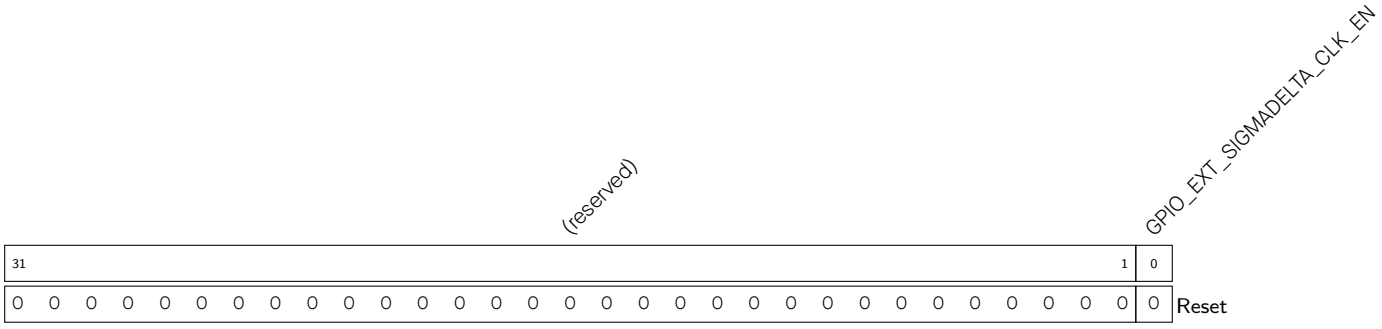
(R/W)

8.19.3 GPIO EXT 寄存器

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址 + 0x0F00 的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 8.27. GPIO_EXT_SIGMADELTA_MISC_REG (0x0004)



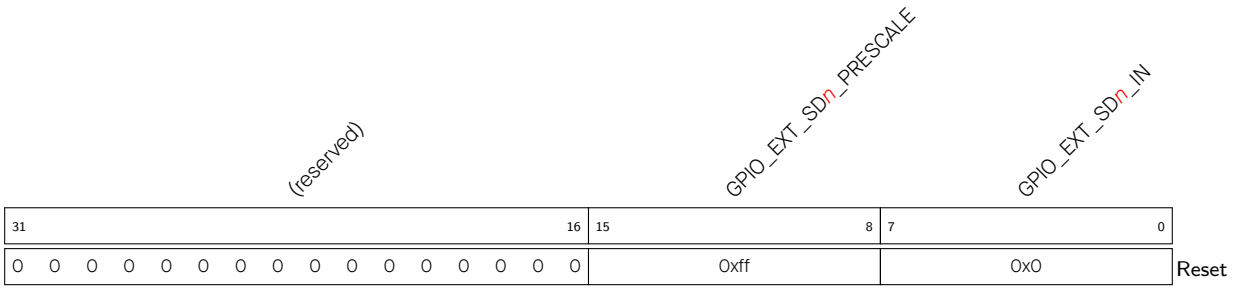
GPIO_EXT_SIGMADELTA_CLK_EN 配置是否使能 SDM 时钟。

0：不使能

1：使能

(R/W)

Register 8.28. GPIO_EXT_SIGMADELTA*n*_REG (*n*: 0-3) (0x0008+0x4**n*)



GPIO_EXT_SD*n*_IN 配置 SDM 输出信号的占空比。

(R/W)

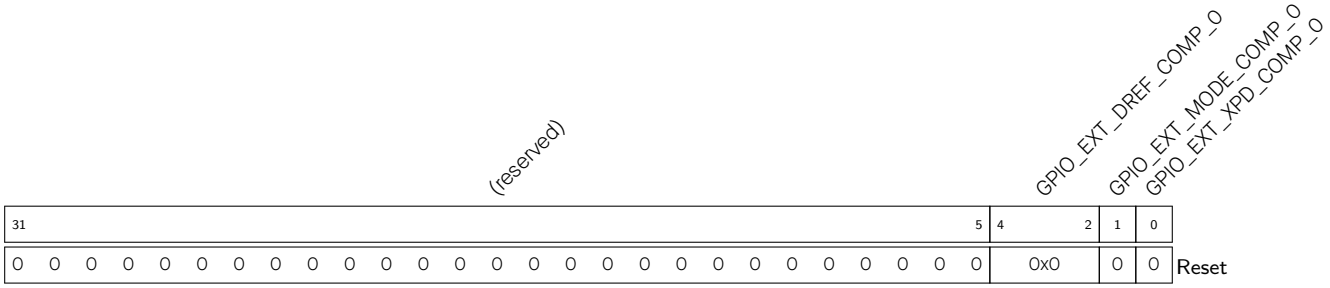
GPIO_EXT_SD*n*_PRESCALE 配置 HP IO MUX 运行时钟的分频系数，分频系数有效值为 0~255。

0：不分频

1~255：2~256 分频

(R/W)

Register 8.29. GPIO_EXT_PAD_COMP_CONFIG_0_REG (0x0058)



GPIO_EXT_XPD_COMP_0 配置是否使能模拟 PAD 电压比较器。

0：不使能

1：使能

(R/W)

GPIO_EXT_MODE_COMP_0 配置选择模拟 PAD 电压比较器的参考电压。

0：参考电压为内部参考电压，同时 GPIO8 可用作常规 GPIO

1：参考电压为 GPIO8 PAD 上的电压

(R/W)

GPIO_EXT_DREF_COMP_0 配置选择模拟 PAD 电压比较器的内部参考电压。

0：内部参考电压为 0 * VDDPST1

1：内部参考电压为 0.1 * VDDPST1

2：内部参考电压为 0.2 * VDDPST1

3：内部参考电压为 0.3 * VDDPST1

4：内部参考电压为 0.4 * VDDPST1

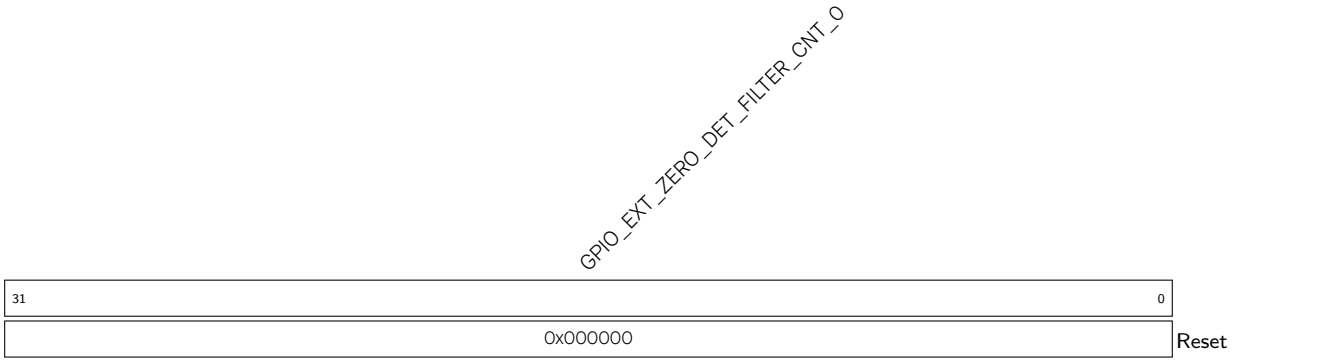
5：内部参考电压为 0.5 * VDDPST1

6：内部参考电压为 0.6 * VDDPST1

7：内部参考电压为 0.7 * VDDPST1

(R/W)

Register 8.30. GPIO_EXT_PAD_COMP_FILTER_0_REG (0x005C)

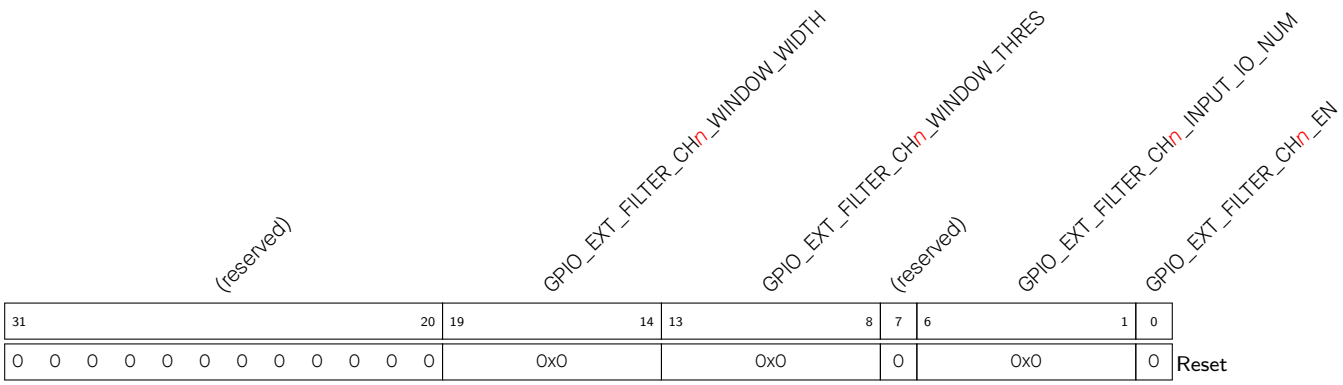


GPIO_EXT_ZERO_DET_FILTER_CNT_0 配置模拟 PAD 电压比较器的新中断源屏蔽周期。

单位：HP IO MUX 运行时钟周期

(R/W)

Register 8.31. GPIO_EXT_GLITCH_FILTER_CH*n*_REG (*n*: 0-7) (0x00D8+0x4**n*)



GPIO_EXT_FILTER_CH*n*_EN 配置是否使能毛刺滤波器的通道 *n*。

0：不使能

1：使能

(R/W)

GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM 配置选择毛刺滤波器的 GPIO 输入。

0：选择 GPIO0

1：选择 GPIO1

.....

27：选择 GPIO27

28：选择 GPIO28

29～63：保留

(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 配置毛刺滤波器的窗口阈值。窗口阈值应小于或等于 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH。

单位：HP IO MUX 运行时钟的一个周期

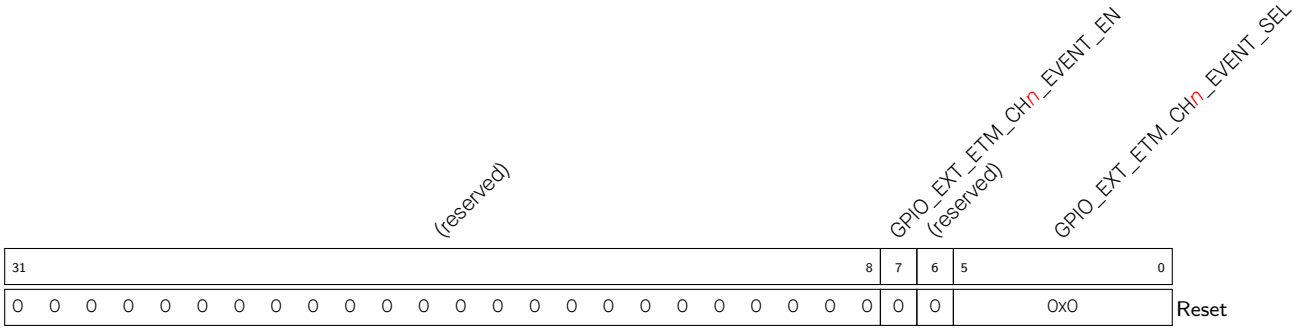
(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 配置毛刺滤波器的窗口宽度。窗口宽度的有效值为 0～63。

单位：HP IO MUX 运行时钟的一个周期

(R/W)

Register 8.32. GPIO_EXT_ETM_EVENT_CH*n*_CFG_REG (*n*: 0-7) (0x0118+0x4**n*)



GPIO_EXT_ETM_CH*n*_EVENT_SEL 配置选择 ETM 事件通道使用的 GPIO。

0：选择 GPIO0

1：选择 GPIO1

.....

27：选择 GPIO27

28：选择 GPIO28

29~63：保留

(R/W)

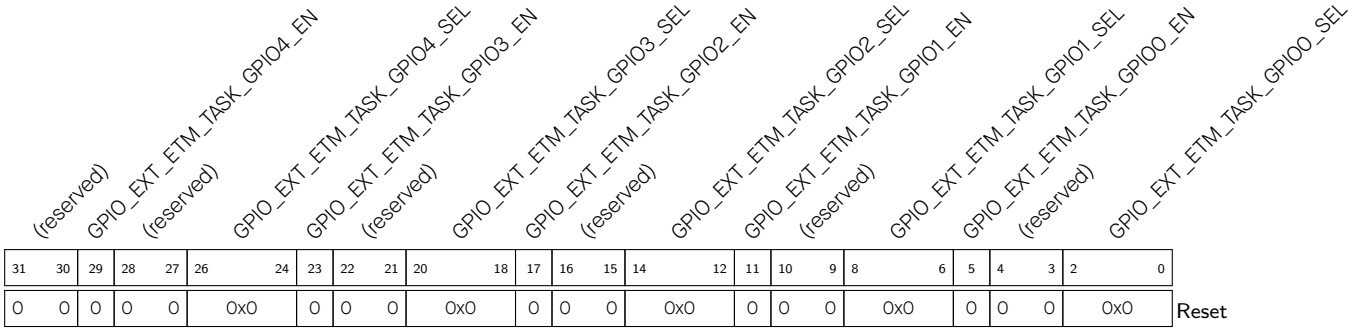
GPIO_EXT_ETM_CH*n*_EVENT_EN 配置是否使能 ETM 事件发送。

0：不使能

1：使能

(R/W)

Register 8.33. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x0158)



GPIO_EXT_ETM_TASK_GPIO0_SEL 配置选择 GPIO0 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO0_EN 配置是否使能 GPIO0 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO1_SEL 配置选择 GPIO1 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO1_EN 配置是否使能 GPIO1 响应 ETM 任务。

0：不使能
1：使能
(R/W)

见下页

Register 8.33. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x0158)

接上页

GPIO_EXT_ETM_TASK_GPIO2_SEL 配置选择 GPIO2 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO2_EN 配置是否使能 GPIO2 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO3_SEL 配置选择 GPIO3 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO3_EN 配置是否使能 GPIO3 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO4_SEL 配置选择 GPIO4 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

见下页

Register 8.33. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x0158)

接上页

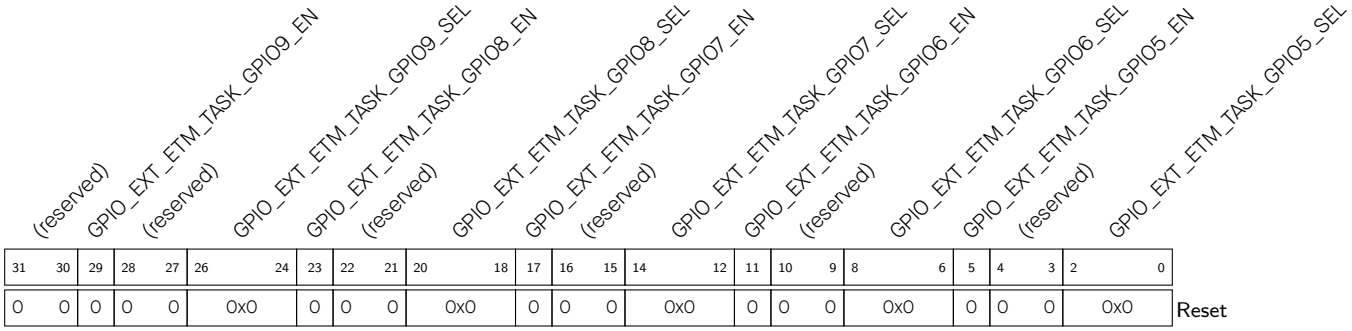
GPIO_EXT_ETM_TASK_GPIO4_EN 配置是否使能 GPIO4 响应 ETM 任务。

0：不使能

1：使能

(R/W)

Register 8.34. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)



GPIO_EXT_ETM_TASK_GPIO5_SEL 配置选择 GPIO5 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO5_EN 配置是否使能 GPIO5 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO6_SEL 配置选择 GPIO6 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO6_EN 配置是否使能 GPIO6 响应 ETM 任务。

0：不使能
1：使能
(R/W)

见下页

Register 8.34. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)

接上页

GPIO_EXT_ETM_TASK_GPIO7_SEL 配置选择 GPIO7 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO7_EN 配置是否使能 GPIO7 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO8_SEL 配置选择 GPIO8 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO8_EN 配置是否使能 GPIO8 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO9_SEL 配置选择 GPIO9 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

见下页

Register 8.34. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)

接上页

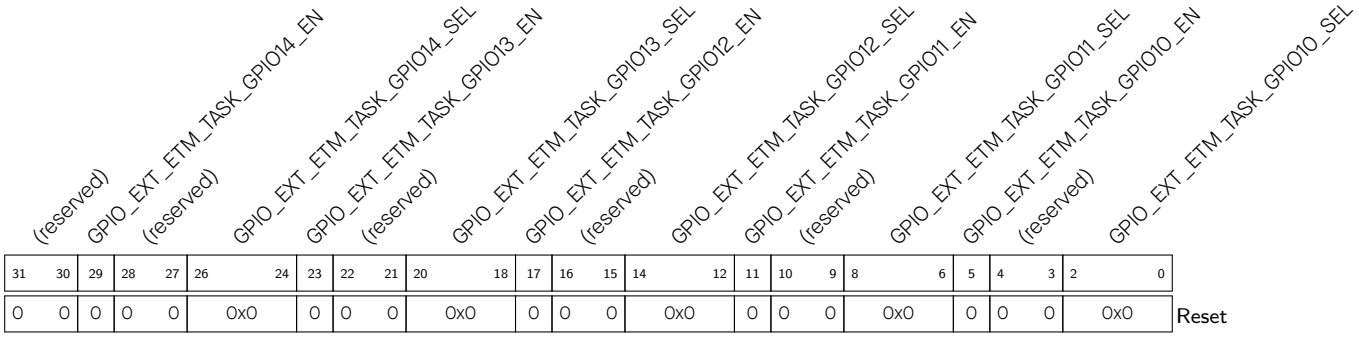
GPIO_EXT_ETM_TASK_GPIO9_EN 配置是否使能 GPIO9 响应 ETM 任务。

0：不使能

1：使能

(R/W)

Register 8.35. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)



GPIO_EXT_ETM_TASK_GPIO10_SEL 配置选择 GPIO10 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO10_EN 配置是否使能 GPIO10 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO11_SEL 配置选择 GPIO11 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO11_EN 配置是否使能 GPIO11 响应 ETM 任务。

0：不使能
1：使能
(R/W)

见下页

Register 8.35. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)

接上页

GPIO_EXT_ETM_TASK_GPIO12_SEL 配置选择 GPIO12 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO12_EN 配置是否使能 GPIO12 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO13_SEL 配置选择 GPIO13 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO13_EN 配置是否使能 GPIO13 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO14_SEL 配置选择 GPIO14 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

见下页

Register 8.35. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)

接上页

GPIO_EXT_ETM_TASK_GPIO14_EN 配置是否使能 GPIO14 响应 ETM 任务。

0：不使能

1：使能

(R/W)

Register 8.36. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x0168)



GPIO_EXT_ETM_TASK_GPIO23_SEL 配置选择 GPIO23 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

2：选择通道 2

3：选择通道 3

4：选择通道 4

5：选择通道 5

6：选择通道 6

7：选择通道 7

(R/W)

GPIO_EXT_ETM_TASK_GPIO23_EN 配置是否使能 GPIO23 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO24_SEL 配置选择 GPIO24 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

2：选择通道 2

3：选择通道 3

4：选择通道 4

5：选择通道 5

6：选择通道 6

7：选择通道 7

(R/W)

见下页

Register 8.36. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x0168)

接上页

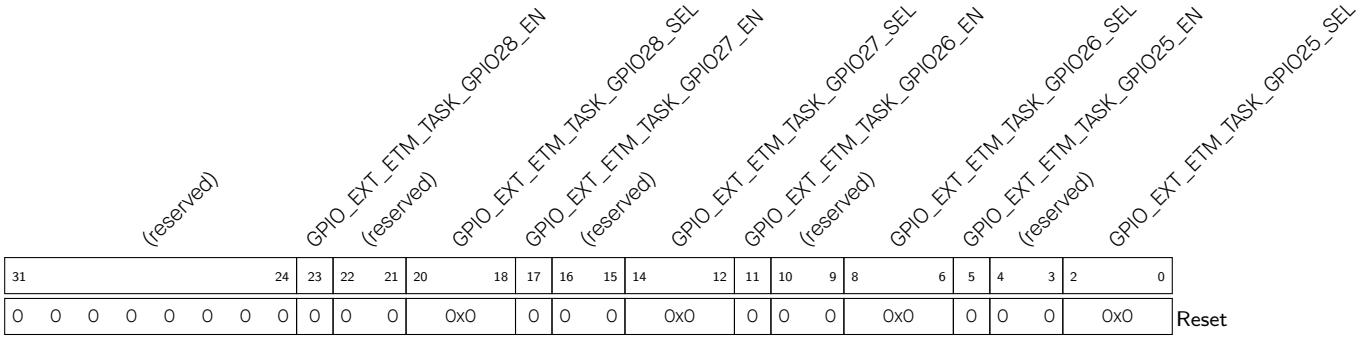
GPIO_EXT_ETM_TASK_GPIO24_EN 配置是否使能 GPIO24 响应 ETM 任务。

0：不使能

1：使能

(R/W)

Register 8.37. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x016C)



GPIO_EXT_ETM_TASK_GPIO25_SEL 配置选择 GPIO25 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO25_EN 配置是否使能 GPIO25 响应 ETM 任务。

0：不使能
1：使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO26_SEL 配置选择 GPIO26 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO26_EN 配置是否使能 GPIO26 响应 ETM 任务。

0：不使能
1：使能
(R/W)

见下页

Register 8.37. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x016C)

接上页

GPIO_EXT_ETM_TASK_GPIO27_SEL 配置选择 GPIO27 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO27_EN 配置是否使能 GPIO27 响应 ETM 任务。

0：不使能
1：使能
(R/W)

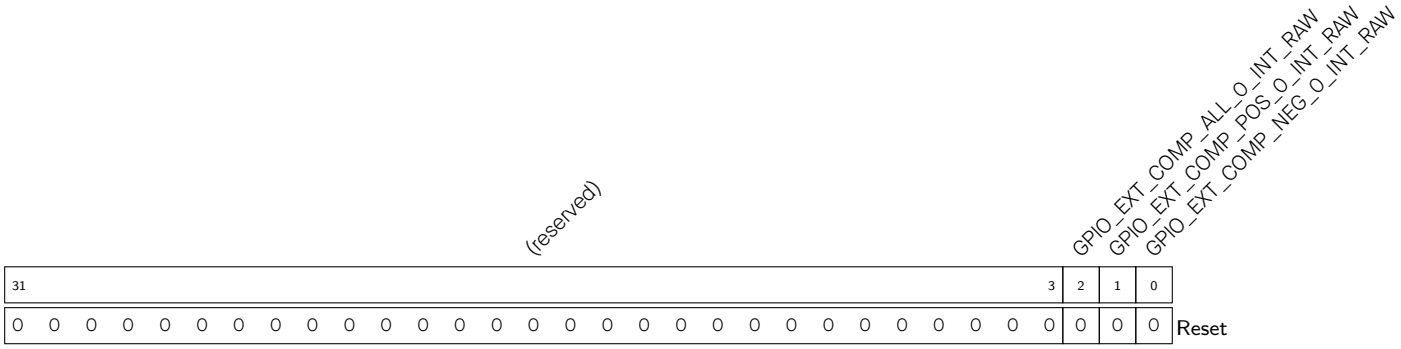
GPIO_EXT_ETM_TASK_GPIO28_SEL 配置选择 GPIO28 的 ETM 任务通道。

0：选择通道 0
1：选择通道 1
2：选择通道 2
3：选择通道 3
4：选择通道 4
5：选择通道 5
6：选择通道 6
7：选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO28_EN 配置是否使能 GPIO28 响应 ETM 任务。

0：不使能
1：使能
(R/W)

Register 8.38. GPIO_EXT_INT_RAW_REG (0x01D0)

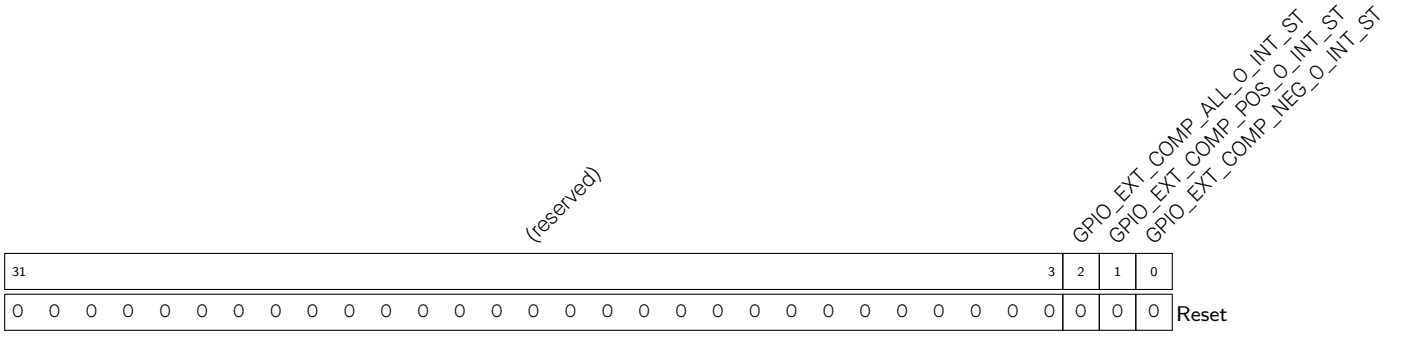


GPIO_EXT_COMP_NEG_0_INT_RAW GPIO_EXT_COMP_NEG_0_INT 的原始中断状态。(RO/WTC/SS)

GPIO_EXT_COMP_POS_0_INT_RAW GPIO_EXT_COMP_POS_0_INT 的原始中断状态。(RO/WTC/SS)

GPIO_EXT_COMP_ALL_0_INT_RAW GPIO_EXT_COMP_ALL_0_INT 的原始中断状态。(RO/WTC/SS)

Register 8.39. GPIO_EXT_INT_ST_REG (0x01D4)

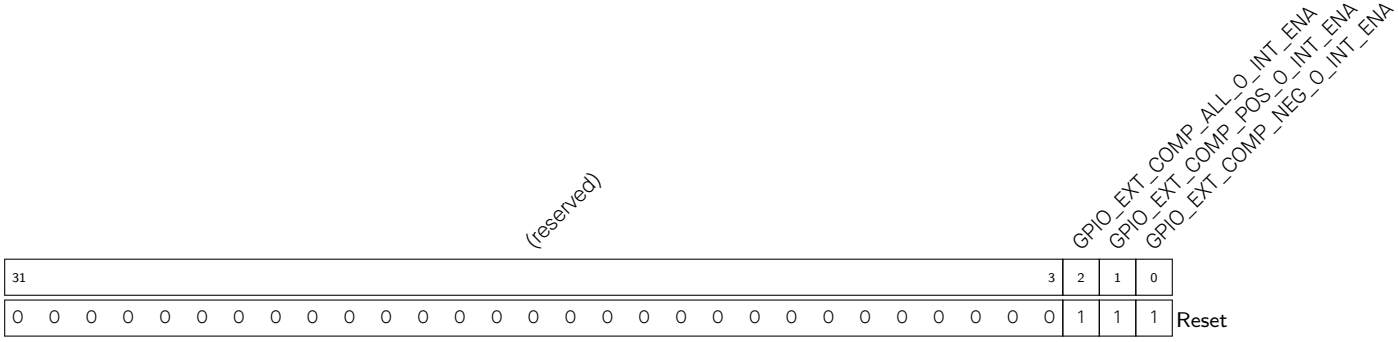


GPIO_EXT_COMP_NEG_0_INT_ST GPIO_EXT_COMP_NEG_0_INT 的屏蔽中断状态。(RO)

GPIO_EXT_COMP_POS_0_INT_ST GPIO_EXT_COMP_POS_0_INT 的屏蔽中断状态。(RO)

GPIO_EXT_COMP_ALL_0_INT_ST GPIO_EXT_COMP_ALL_0_INT 的屏蔽中断状态。(RO)

Register 8.40. GPIO_EXT_INT_ENA_REG (0x01D8)

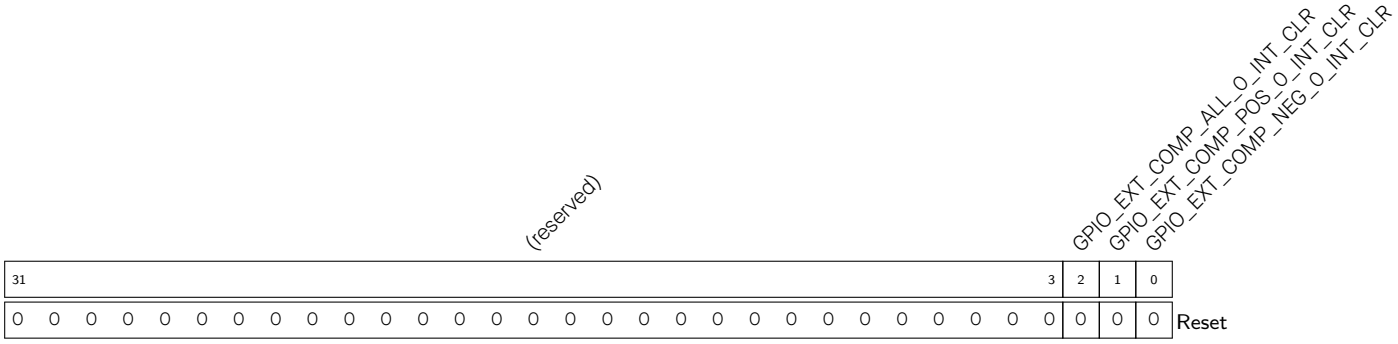


GPIO_EXT_COMP_NEG_0_INT_ENA 写 1 使能 GPIO_EXT_COMP_NEG_0_INT 中断。(R/W)

GPIO_EXT_COMP_POS_0_INT_ENA 写 1 使能 GPIO_EXT_COMP_POS_0_INT 中断。(R/W)

GPIO_EXT_COMP_ALL_0_INT_ENA 写 1 使能 GPIO_EXT_COMP_ALL_0_INT 中断。(R/W)

Register 8.41. GPIO_EXT_INT_CLR_REG (0x01DC)

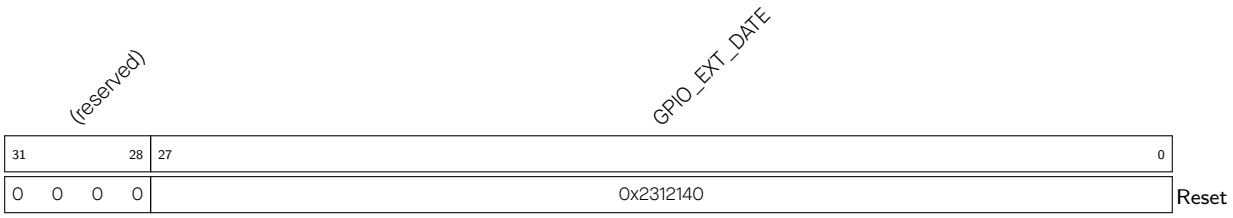


GPIO_EXT_COMP_NEG_0_INT_CLR 写 1 清除 GPIO_EXT_COMP_NEG_0_INT 中断。(WT)

GPIO_EXT_COMP_POS_0_INT_CLR 写 1 清除 GPIO_EXT_COMP_POS_0_INT 中断。(WT)

GPIO_EXT_COMP_ALL_0_INT_CLR 写 1 清除 GPIO_EXT_COMP_ALL_0_INT 中断。(WT)

Register 8.42. GPIO_EXT_VERSION_REG (0x01FC)



GPIO_EXT_DATE 版本控制寄存器。

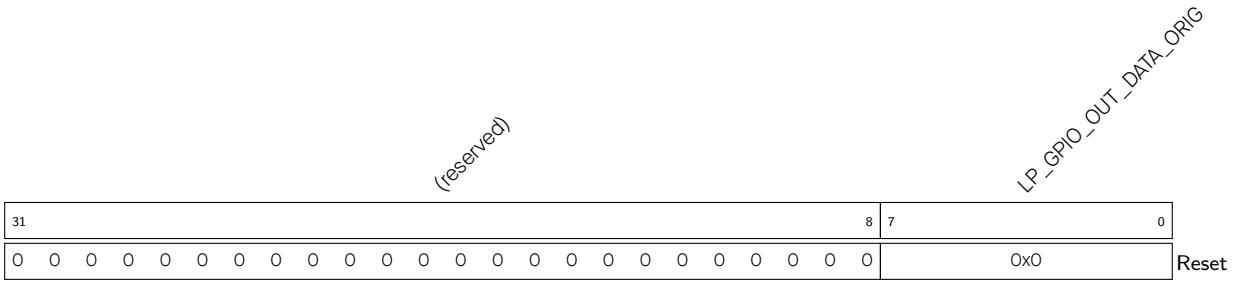
(R/W)

8.19.4 LP GPIO 矩阵寄存器

本小节的所有地址均为相对于 LP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 8.43. LP_GPIO_OUT_REG (0x0004)



LP_GPIO_OUT_DATA_ORIG 配置 GPIO0~GPIO6 的输出。

每一位的取值可以为：

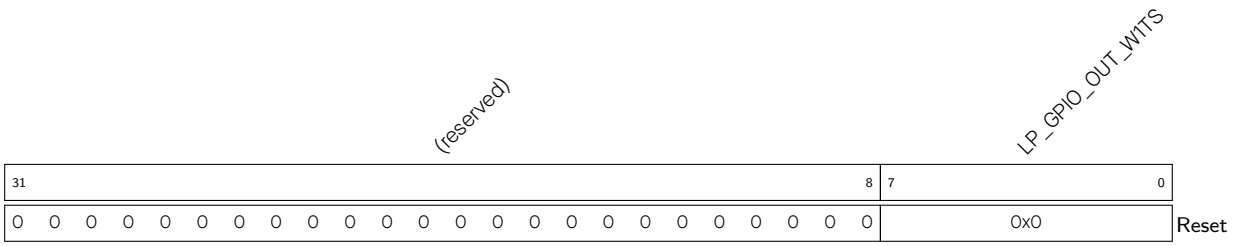
0：低电平

1：高电平

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

(R/W/WTC)

Register 8.44. LP_GPIO_OUT_W1TS_REG (0x0008)



LP_GPIO_OUT_W1TS 配置是否置位 GPIO0~GPIO6 的输出寄存器 LP_GPIO_OUT_REG。

每一位的取值可以为：

0：不置位

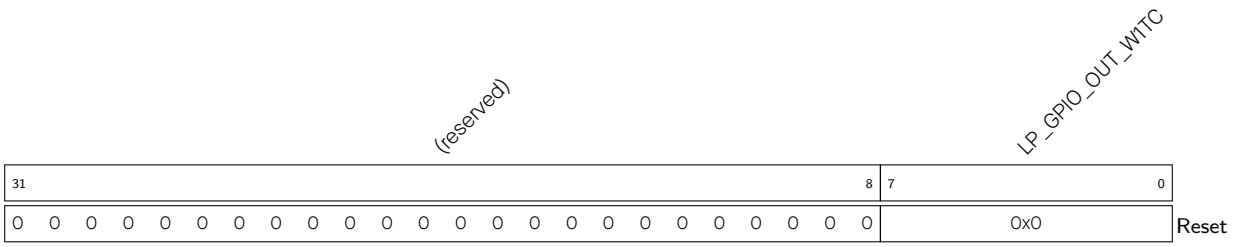
1：LP_GPIO_OUT_REG 中相应位也将置 1

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来置位 LP_GPIO_OUT_REG。

(WT)

Register 8.45. LP_GPIO_OUT_W1TC_REG (0x000C)



LP_GPIO_OUT_W1TC 配置是否清零 GPIO0~GPIO6 的输出寄存器 LP_GPIO_OUT_REG。

每一位的取值可以为：

0：不清零

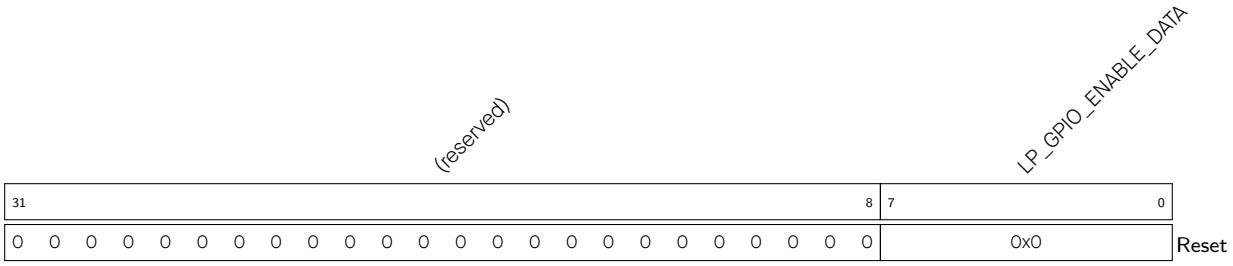
1：LP_GPIO_OUT_REG 中相应位将被清零。

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来清零 LP_GPIO_OUT_REG。

(WT)

Register 8.46. LP_GPIO_ENABLE_REG (0x0010)



LP_GPIO_ENABLE_DATA 配置是否使能 GPIO0~GPIO6 的输出。

每一位的取值可以为:

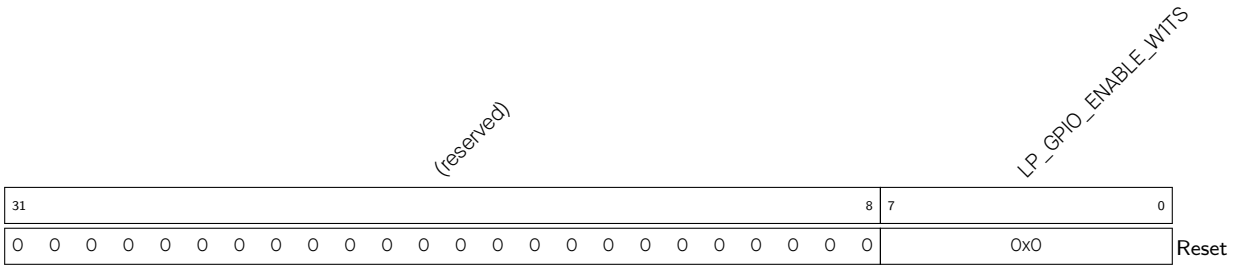
0: 不使能

1: 使能

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

(R/W/WTC)

Register 8.47. LP_GPIO_ENABLE_W1TS_REG (0x0014)



LP_GPIO_ENABLE_W1TS 配置是否置位 GPIO0~GPIO6 的输出使能寄存器 LP_GPIO_ENABLE_REG。

每一位的取值可以为:

0: 不置位

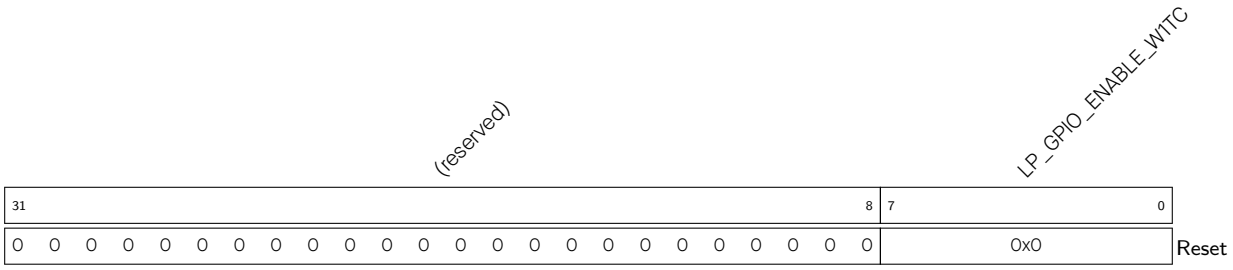
1: LP_GPIO_ENABLE_REG 中相应位也将置 1

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来置位 LP_GPIO_ENABLE_REG。

(WT)

Register 8.48. LP_GPIO_ENABLE_W1TC_REG (0x0018)



LP_GPIO_ENABLE_W1TC 配置是否清零 GPIO0~GPIO6 的输出使能寄存器 LP_GPIO_ENABLE_REG。

每一位的取值可以为：

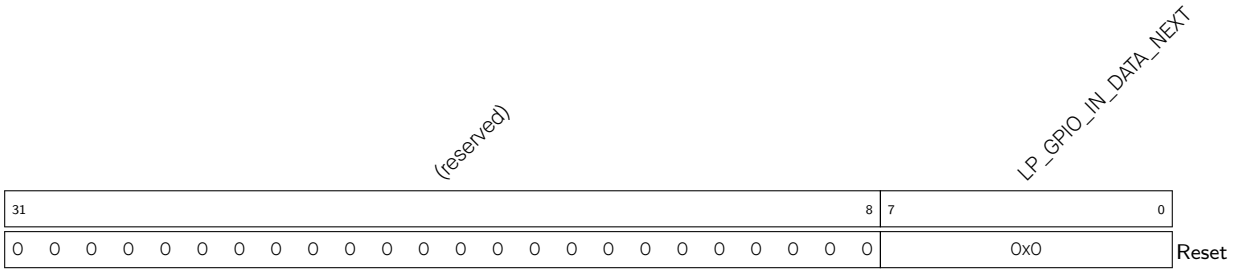
0：不清零

1：LP_GPIO_ENABLE_REG 中相应位将被清零

推荐使用此寄存器来清零 LP_GPIO_ENABLE_REG。

(WT)

Register 8.49. LP_GPIO_IN_REG (0x001C)



LP_GPIO_IN_DATA_NEXT 表示 GPIO0~GPIO6 的输入值。

每一位均代表一个管脚的输入值：

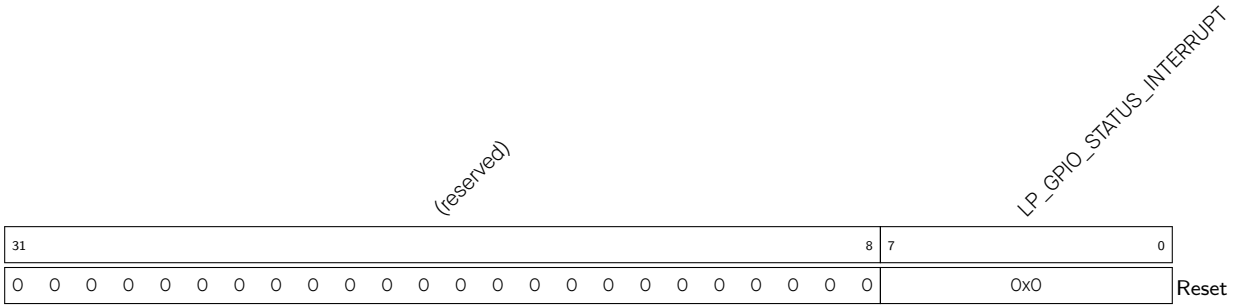
0：低电平

1：高电平

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

(RO)

Register 8.50. LP_GPIO_STATUS_REG (0x0020)



LP_GPIO_STATUS_INTERRUPT 配置 GPIO0~GPIO6 的中断状态。

每一位的取值可以为：

0：没有中断

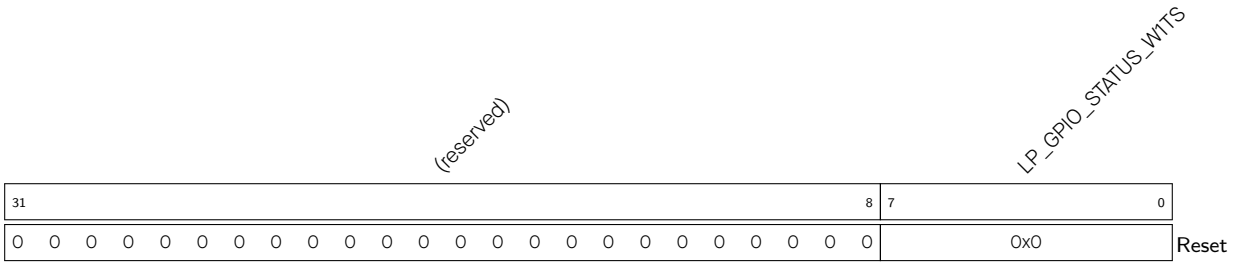
1：已触发中断

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

该字段与 LP_GPIO_PIN*n*_REG 中的 LP_GPIO_PIN*n*_INT_TYPE 配合使用。

(R/W/WTC)

Register 8.51. LP_GPIO_STATUS_W1TS_REG (0x0024)



LP_GPIO_STATUS_W1TS 配置是否置位 GPIO0~GPIO6 的中断状态寄存器 LP_GPIO_STATUS_INT。

每一位的取值可以为：

0：不置位

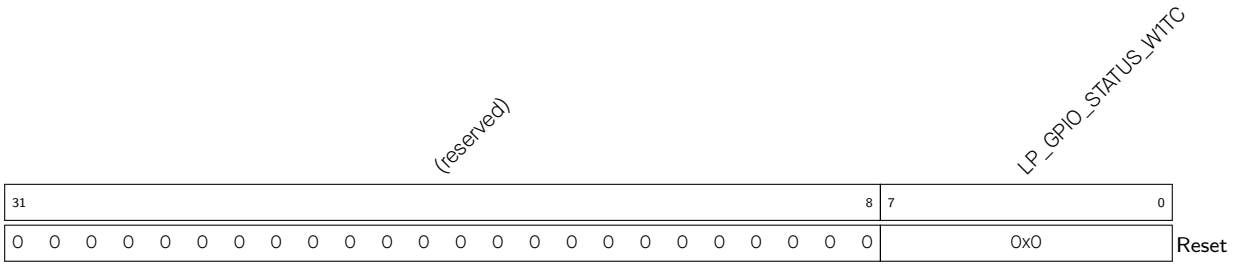
1：LP_GPIO_STATUS_INT 中的相应位也将置 1

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来置位 LP_GPIO_STATUS_INT。

(WT)

Register 8.52. LP_GPIO_STATUS_W1TC_REG (0x0028)



LP_GPIO_STATUS_W1TC 配置是否清零 GPIO0~GPIO6 的中断状态寄存器 LP_GPIO_STATUS_INT。

每一位的取值可以为：

0：不清零

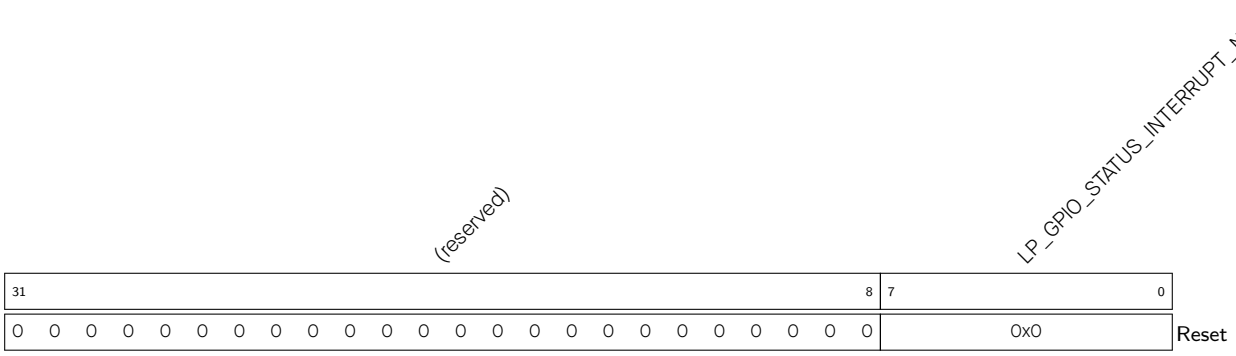
1：LP_GPIO_STATUS_INT 中相应位将被清除

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来清零 LP_GPIO_STATUS_INT。

(WT)

Register 8.53. LP_GPIO_STATUS_NEXT_REG (0x002C)



LP_GPIO_STATUS_INTERRUPT_NEXT 表示 GPIO0~GPIO6 的中断源状态。

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

每一位代表：

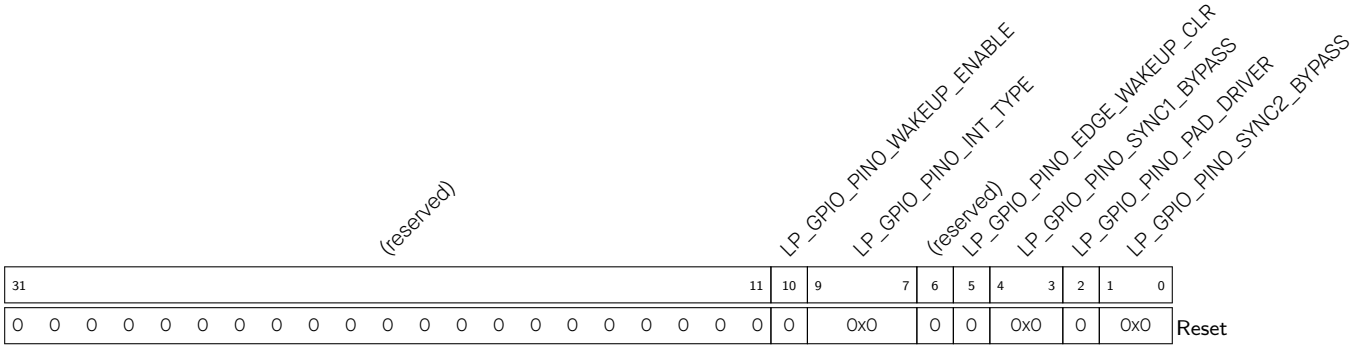
0：中断源状态无效

1：中断源状态有效

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。

(RO)

Register 8.54. LP_GPIO_PIN*n*_REG (*n*: 0-6) (0x0030+0x4**n*)



LP_GPIO_PIN*n*_SYNC2_BYPASS 配置是否使能 GPIO 输入数据在第二级同步中 LP IO MUX 运行时钟上升沿同步或下降沿同步。

0：不同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

LP_GPIO_PIN*n*_PAD_DRIVER 配置选择管脚的输出驱动模式。

0：正常输出

1：开漏输出

(R/W)

LP_GPIO_PIN*n*_SYNC1_BYPASS 配置是否使能 GPIO 输入数据在第一级同步中 LP IO MUX 运行时钟上升沿同步或下降沿同步。

0：不同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

LP_GPIO_PIN*n*_EDGE_WAKEUP_CLR 配置是否清除 GPIO0~GPIO6 的跳变沿唤醒状态。

0：无效

1：清除

(WT)

LP_GPIO_PIN*n*_INT_TYPE 配置 GPIO*n* 的中断类型。

0：关闭 GPIO 中断

1：上升沿触发

2：下降沿触发

3：任一沿触发

4：低电平触发

5：高电平触发

(R/W)

见下页

Register 8.54. LP_GPIO_PIN*n*_REG (*n*: 0-6) (0x0030+0x4*n*)

接上页

LP_GPIO_PIN*n*_WAKEUP_ENABLE 配置是否使能 GPIO*n* 唤醒功能。

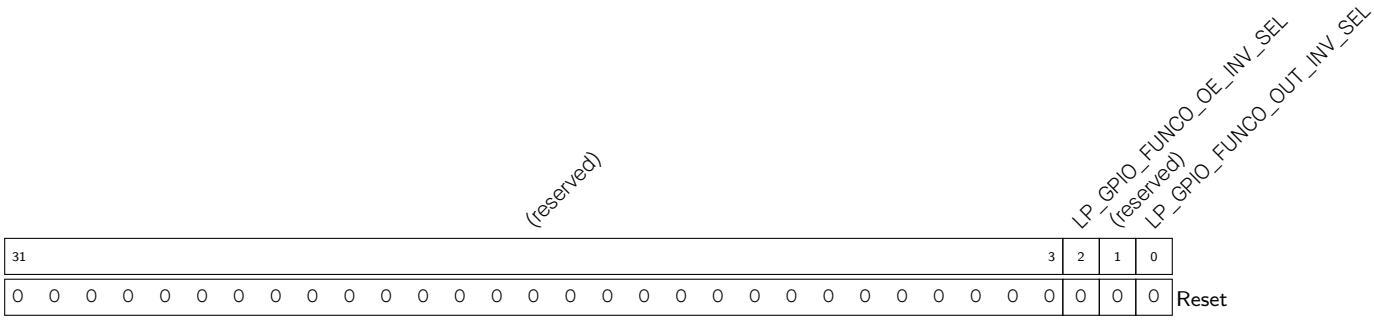
0：不使能

1：使能

PD_LP_PERI 域掉电后，该唤醒功能将被禁用。

(R/W)

Register 8.55. LP_GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-6) (0x02B0+0x4*n*)



LP_GPIO_FUNC*n*_OUT_INV_SEL 配置是否反转输出值。

0：不反转

1：反转

(R/W)

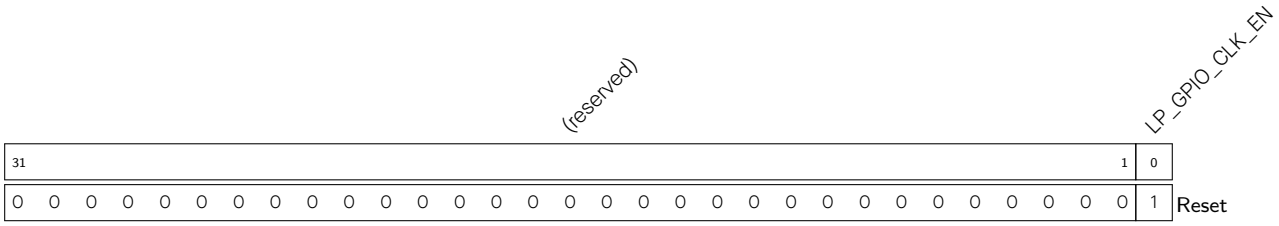
LP_GPIO_FUNC*n*_OE_INV_SEL 配置是否反转输出使能信号。

0：不反转

1：反转

(R/W)

Register 8.56. LP_GPIO_CLOCK_GATE_REG (0x03F8)



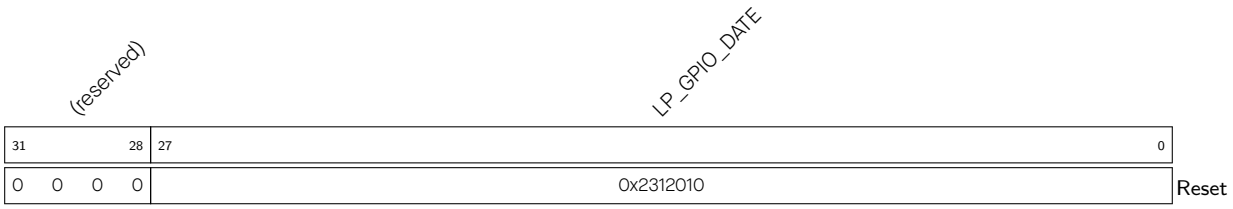
LP_GPIO_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能

(R/W)

Register 8.57. LP_GPIO_DATE_REG (0x03FC)



LP_GPIO_DATE 版本控制寄存器。

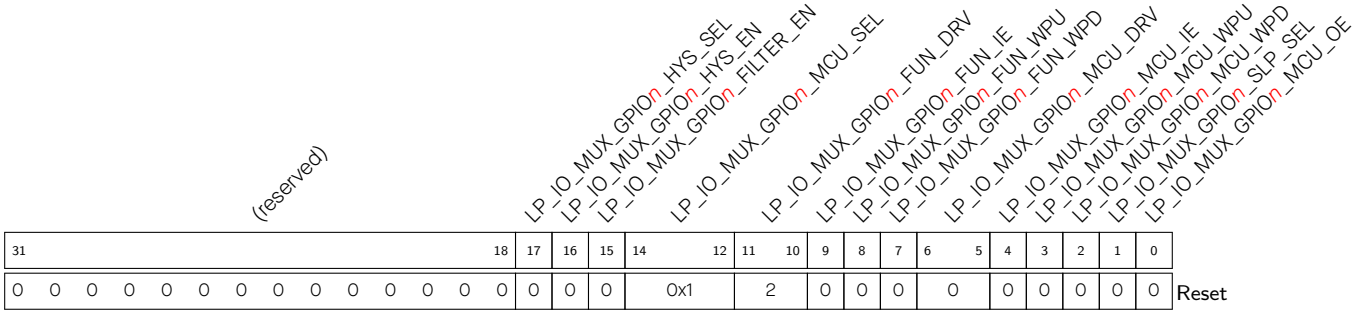
(R/W)

8.19.5 LP IO MUX 寄存器

本小节的所有地址均为相对于 LP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 8.58. LP_IO_MUX_GPIO*n*_REG (*n*: 0-6) (0x0000+0x4**n*)



LP_IO_MUX_GPIO*n*_MCU_OE 配置是否使能睡眠模式下 GPIO*n* 的输出。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_SLP_SEL 配置是否使能 GPIO*n* 进入睡眠模式。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_MCU_WPD 配置是否使能睡眠模式下 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_MCU_WPU 配置是否使能睡眠模式下 GPIO*n* 的上拉电阻。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_MCU_IE 配置是否使能睡眠模式下 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_MCU_DRV 配置睡眠模式下 GPIO*n* 的驱动强度。

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

见下页

Register 8.58. LP_IO_MUX_GPIO*n*_REG (*n*: 0-6) (0x0000+0x4**n*)

接上页

LP_IO_MUX_GPIO*n*_FUN_WPD 配置是否使能 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_FUN_WPU 配置是否使能 GPIO*n* 的上拉电阻。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_FUN_IE 配置是否使能 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_FUN_DRV 配置 GPIO*n* 的驱动强度。

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

LP_IO_MUX_GPIO*n*_MCU_SEL 配置选择 GPIO*n* 的 LP IO MUX 功能。

0：选择功能 0

1：选择功能 1

.....

(R/W)

LP_IO_MUX_GPIO*n*_FILTER_EN 配置是否使能管脚输入信号的滤波功能。

0：不使能

1：使能

(R/W)

LP_IO_MUX_GPIO*n*_HYS_EN 当 LP_IO_MUX_GPIO*n*_HYS_SEL 设置为 1 时，配置是否使能管脚的迟滞功能。

0：不使能

1：使能

(R/W)

见下页

Register 8.58. LP_IO_MUX_GPIO*n*_REG (*n*: 0-6) (0x0000+0x4**n*)

接上页

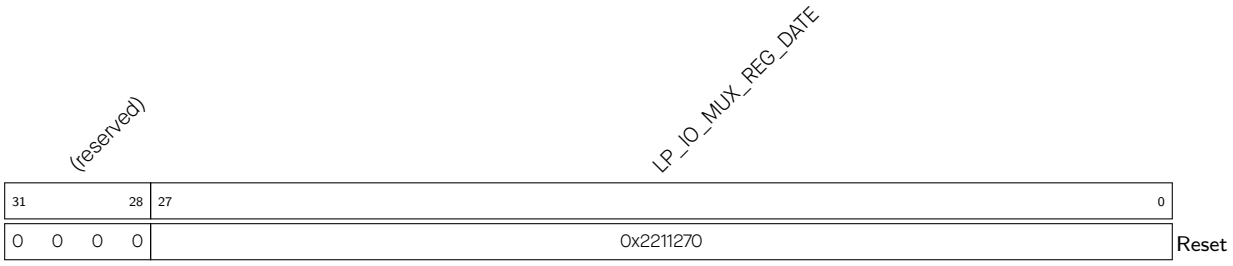
LP_IO_MUX_GPIO*n*_HYS_SEL 配置选择 GPIO*n* 迟滞功能的使能信号。

0：使用 eFuse 的输出使能信号

1：使用 LP_IO_MUX_GPIO*n*_HYS_EN 用作输出使能信号

(R/W)

Register 8.59. LP_IO_MUX_DATE_REG (0x01FC)



LP_IO_MUX_REG_DATE 版本控制寄存器。

(R/W)

第 9 章

复位和时钟

9.1 复位

9.1.1 概述

ESP32-C5 提供四种级别的复位方式，分别是 CPU 复位 (CPU reset)、内核复位 (Core reset)、系统复位 (System reset) 和芯片复位 (Chip reset)。除芯片复位外，其他复位方式不影响片上内存存储的数据。图 9.1-1 展示了整个芯片系统的结构以及四种复位等级。

9.1.2 结构图

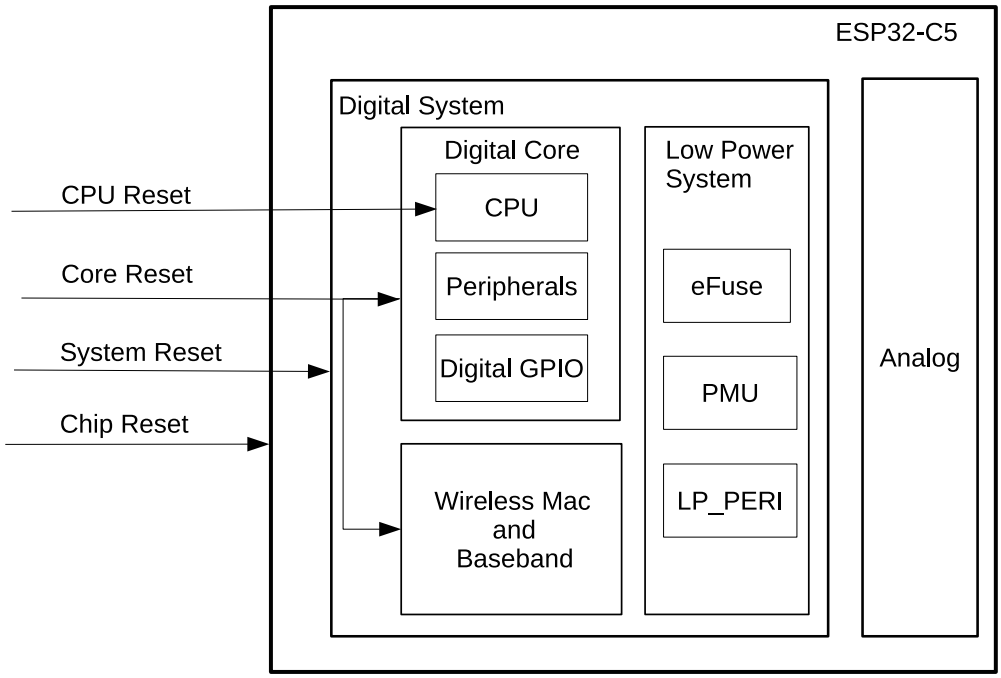


图 9.1-1. 四种复位等级

ESP32-C5 的数字系统分为两部分：高性能系统 (High Performance System, HP system) 和 低功耗系统 (Low Power System, LP system)。其中高性能系统包含数字内核 (Digital Core) 、无线 MAC 和基带 (Wireless Mac and Baseband) 和 HP SRAM，低功耗系统中仅包含一些低功耗外设和 LP SRAM。详细信息可参考图 9.1-1（HP SRAM 和 LP SRAM 不会复位，因此未在图中标注）。

9.1.3 特性

- 支持四种复位等级：

  - CPU 复位：复位 CPU 核。复位释放后，程序将从 CPU Reset Vector (0x40000000) 开始执行；
  - 内核复位：复位除低功耗系统以外的其他数字系统，包括 CPU、外设、数字 GPIO、无线 MAC 和基带；
  - 系统复位：复位包括低功耗系统在内的整个数字系统；
  - 芯片复位：复位包括数字系统和模拟系统在内的整个芯片。
- 支持软件复位和硬件复位：
  - 软件复位：CPU 配置相关寄存器可触发软件复位，见章节 13 低功耗管理；
  - 硬件复位：硬件复位直接由硬件电路触发。

9.1.4 功能描述

上述任一复位发生时，CPU 将立刻复位。复位释放后，CPU 可通过读取 LP_CLKRST_RESET_CAUSE 获取复位源。可通过配置 LP_CLKRST_CORE0_RESET_CAUSE_CLR 清除当前 LP_CLKRST_RESET_CAUSE 寄存器记录的复位源。

表 9.1-1 列出了从上述寄存器中可能读出的复位源及其触发的复位等级。当有多个复位源同时生效时，LP_CLKRST_RESET_CAUSE 中的复位源为表 9.1-1 中最靠上的一个生效复位源。当有多个复位源生效，但不在同一时间生效时，LP_CLKRST_RESET_CAUSE 记录的为最近发生的一次复位的复位源。

表 9.1-1. 复位源

编码	复位源	复位等级	说明
0x1A	CPU 锁死复位	CPU 复位	—
0x01	芯片复位[1]	芯片复位	—
0x12	超级看门狗复位	系统复位	详见章节 16 看门狗定时器 (WDT)
0x19	电源毛刺复位	系统复位	详见章节 21 电源检测器
0x0F	欠压系统复位	芯片复位或系统复位	欠压检测器触发的复位[2]
0x10	RWDT 系统复位	系统复位	详见章节 16 看门狗定时器 (WDT)
0x09	RWDT 内核复位	内核复位	详见章节 16 看门狗定时器 (WDT)
0x14	eFuse 复位	内核复位	eFuse CRC 校验错误触发复位
0x07	MWDT0 内核复位	内核复位	详见章节 16 看门狗定时器 (WDT)
0x08	MWDT1 内核复位	内核复位	详见章节 16 看门狗定时器 (WDT)
0x15	USB (UART) 复位	内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 Serial 接口将触发此复位，详见章节 37 USB 串口/JTAG 控制器
0x16	USB (JTAG) 复位	内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 JTAG 接口将触发此复位，详见章节 37 USB 串口/JTAG 控制器
0x03	软件系统复位	内核复位	配置 LP_AON_HPSYS_SW_RESET 触发
0x0D	RWDT CPU 复位	CPU 复位	详见章节 16 看门狗定时器 (WDT)
0x0C	软件 CPU 复位	CPU 复位	配置 LP_AON_CPU_CORE0_SW_RESET 触发
0x0B	MWDT0 CPU 复位	CPU 复位	详见章节 16 看门狗定时器 (WDT)
0x11	MWDT1 CPU 复位	CPU 复位	详见章节 16 看门狗定时器 (WDT)
0x18	JTAG CPU 复位	CPU 复位	接收 “JDB 复位 CPU” 指令时触发

见下页

表 9.1-1 – 接上页

编码	复位源	复位等级	说明
0x05	Deep-sleep 复位	内核复位	详见章节 13 低功耗管理

[1] 芯片复位的触发源包括以下两项：

- 芯片上电触发芯片复位
- 欠压检测器触发芯片复位

[2] 欠压检测器在检测到欠压状态时，将根据寄存器配置，选择触发系统复位或者芯片复位，详见章节 13 低功耗管理。

9.1.5 外设复位

外设可单独复位，也可通过配置内核复位、系统复位、芯片复位连带复位。在解复位后，需要读取各外设的解复位完成指示寄存器，此类寄存器为 1 表示对应外设解复位成功，否则外设可能无法正常工作。

ESP32-C5 高性能系统的外设复位寄存器已整合到 PCR (POWER/CLOCK/RESET) 模块，低功耗系统的外设复位寄存器在低功耗系统时钟复位寄存器中。可通过章节 9.4寄存器列表 查看各外设对应的复位寄存器以及解复位成功指示寄存器。

9.2 时钟

9.2.1 概述

ESP32-C5 的时钟主要来源于振荡器 (oscillator, OSC)、RC 振荡电路和 PLL 时钟生成电路。上述时钟源产生的时钟经时钟分频器或时钟选择器等时钟模块的处理，使得大部分功能模块可以根据不同功耗和性能需求来获取及选择对应频率的工作时钟。图 9.2-1 为系统时钟结构。

9.2.2 结构图

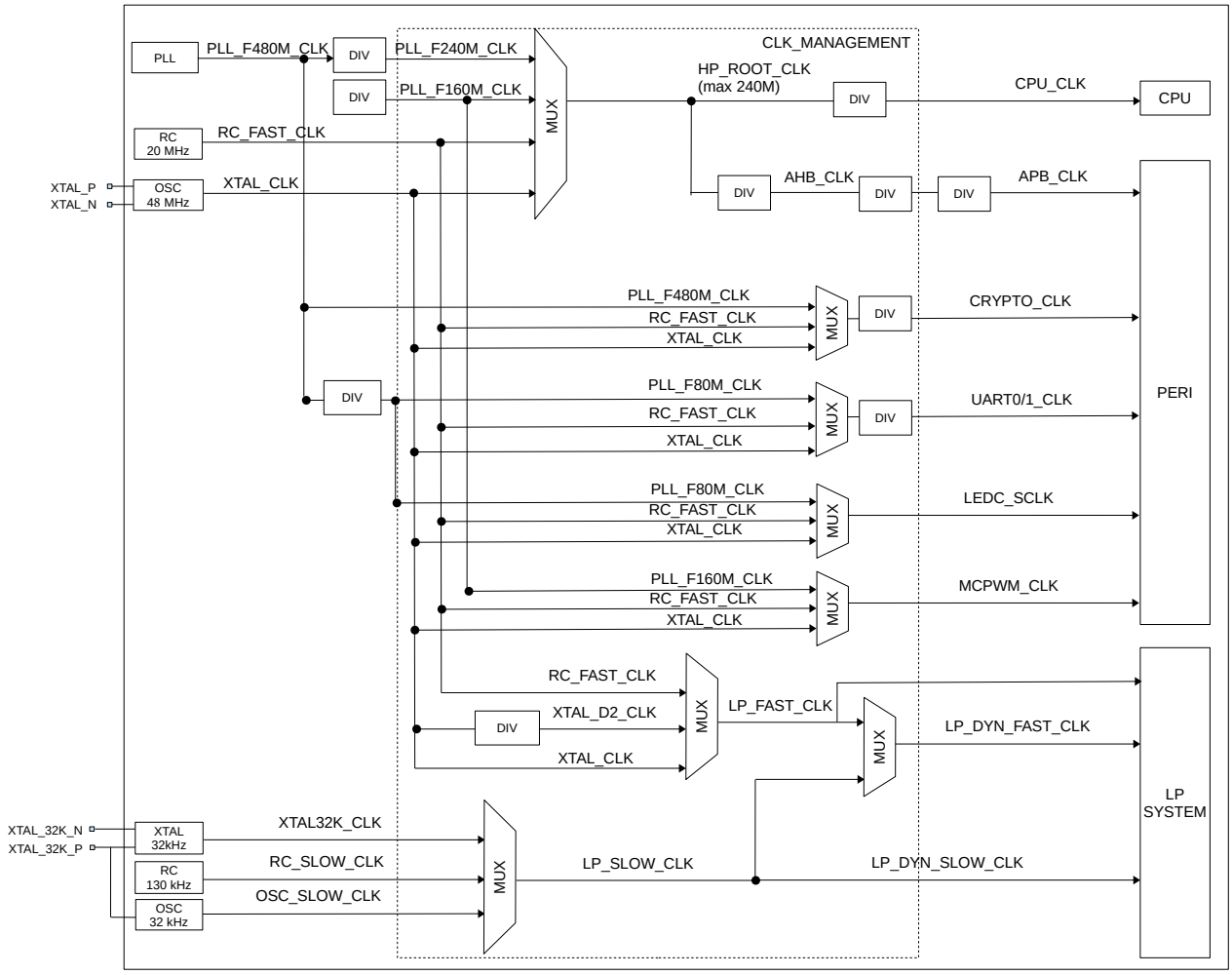


图 9.2-1. 系统时钟

9.2.3 特性

ESP32-C5 的时钟根据频率不同，可分为：

- 高性能时钟，主要为 CPU 和数字外设提供工作时钟
  - PLL_F480M_CLK：480 MHz 内部 PLL 时钟（参考时钟是 XTAL_CLK）
  - XTAL_CLK：48 MHz 外部晶振时钟
- 低功耗时钟，主要为低功耗系统以及部分处于低功耗模式的外设提供工作时钟
  - XTAL32K_CLK：32 kHz 外部晶振时钟
  - RC_FAST_CLK：内置快速 RC 振荡器时钟，频率可调节（通常为 20 MHz）
  - RC_SLOW_CLK：内置慢速 RC 振荡器，通常频率为 130 kHz
  - OSC_SLOW_CLK：外置低速时钟，通常频率为 32 kHz，通过 XTAL_32K_P 输入，在配置好这个 GPIO 的状态后需要配置 Hold 功能，具体请参考章节 *8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* > *8.9 GPIO 管脚的 Hold 特性*

9.2.4 功能描述

9.2.4.1 高性能系统时钟

如图 9.2-1 所示，CPU_CLK 为 CPU 主时钟。CPU 主频最高可以达到 240 MHz，并且能够在超低频下工作（通常为 2 MHz），以减少功耗。CPU_CLK 与 AHB_CLK、APB_CLK 共用一个时钟源。用户可配置 PCR_SOC_CLK_SEL 选择 XTAL_CLK、PLL_240M_CLK、PLL_160M_CLK 或 RC_FAST_CLK 作为 CPU_CLK 的时钟源，具体请参考表 9.2-1 和表 9.2-2。默认状态下，CPU 的时钟源为 XTAL_CLK，且分频系数为 1 分频。

表 9.2-1. CPU_CLK 时钟源选择

PCR_SOC_CLK_SEL	时钟源
0	XTAL_CLK
1	RC_FAST_CLK
2	PLL_F160M_CLK
3	PLL_F240M_CLK

表 9.2-2. CPU_CLK、AHB_CLK 和 HP_ROOT_CLK 的时钟频率

时钟名称	时钟源	时钟频率
HP_ROOT_CLK	PLL_F240M_CLK	240 MHz
	PLL_F160M_CLK	160 MHz
	XTAL_CLK	48 MHz
	RC_FAST_CLK	20 MHz
CPU_CLK[1]	HP_ROOT_CLK	$f_{HP\_ROOT\_CLK}$ / (PCR_CPU_DIV_NUM + 1)
AHB_CLK[2]	HP_ROOT_CLK	$f_{HP\_ROOT\_CLK}$ / (PCR_AHB_DIV_NUM + 1)

[1] CPU_CLK 频率必须大于等于 AHB_CLK 频率，并且必须为 AHB_CLK 频率的整数倍

[2] AHB_CLK 时钟频率不能超过 XTAL_CLK 频率

说明：

配置 HP_ROOT_CLK 的时钟源选择寄存器或者 CPU_CLK、AHB_CLK 的分频寄存器后，需要额外配置 PCR_BUS_CLOCK_UPDATE 寄存器将上述寄存器配置生效，可通过读取 PCR_BUS_CLOCK_UPDATE 寄存器来判断上述配置是否成功生效。

如图 9.2-1 所示，APB_CLK 是在 AHB_CLK 基础上经过两级分频得到的。其中第一级分频为恒定分频，即始终按照分频系数（PCR_APB_DIV_NUM + 1）的值进行分频。第二级分频为自动降频，如果芯片中的主机没有访问外设寄存器的请求时，会以分频系数（APB_DECREASE_DIV_NUM + 1）进行二次分频产生 APB_CLK，从而达到节约功耗的目的。但当芯片中的主机发起了访问外设寄存器的请求时，APB_CLK 将恢复至一级分频后的频率。

用户可将 APB_DECREASE_DIV_NUM 设置为 0 以禁用自动降频功能。默认情况下，该功能禁用。

9.2.4.2 低功耗系统时钟

低功耗系统能够在大多数时钟源关闭的状态下工作。低功耗系统时钟包括 LP_SLOW_CLK 时钟和 LP_FAST_CLK 时钟。

LP_SLOW_CLK 和 LP_FAST_CLK 的时钟源为低频时钟，其中：

- LP_SLOW_CLK 时钟有三种可能的时钟源：
  - RC_SLOW_CLK
  - XTAL32K_CLK
  - OSC_SLOW_CLK
- LP_FAST_CLK 时钟有三种可能的时钟源：
  - XTAL_CLK 的 2 分频时钟 XTAL_D2_CLK (24 MHz)
  - RC_FAST_CLK
  - XTAL_CLK

LP_DYN_SLOW_CLK 时钟源为 LP_SLOW_CLK，频率同时钟源频率。

LP_DYN_FAST_CLK 根据芯片电源的模式（详见章节 13 低功耗管理）选择时钟源：

- 芯片电源为 Active、Modem-sleep 模式时始终选择 LP_FAST_CLK 作为时钟源
- 芯片电源为 Light-sleep、Deep-sleep 模式时选择 LP_SLOW_CLK 作为时钟源

9.2.4.3 外设时钟

表 9.2-3、表 9.2-4、表 9.2-5、和表 9.2-6 分别列出了接入各个外设时钟的衍生高性能时钟源、接入高性能系统各个外设的时钟、衍生低功耗时钟源、以及接入低功耗系统各个外设的时钟。

GoBack

表 9.2-3. 衍生高性能时钟源

衍生时钟	源时钟		衍生时钟		源时钟				衍生时钟									源时钟
	XTAL_CLK 48 MHz	PLL_F480M_CLK 480 MHz	PLL_CLK PLL_F240M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	RC_FAST_CLK 20 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	HP_ROOT_CLK 240 MHz/160 MHz/48 MHz/20 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 24 MHz	LP_FAST_CLK	Clock from IO
PLL_F240M_CLK		Y																
PLL_F160M_CLK		Y																
PLL_F120M_CLK		Y																
PLL_F80M_CLK		Y																
PLL_F48M_CLK		Y																
PLL_F40M_CLK		Y																
HP_ROOT_CLK	Y		Y	Y	Y													
CRYPTO_CLK	Y	Y			Y													
APB_CLK									Y									
AHB_CLK									Y									
CPU_CLK									Y									

表 9.2-4. 高性能系统外设时钟

外设	源时钟	衍生时钟 PLL_CLK					源时钟	衍生时钟									源时钟
	XTAL_CLK 48 MHz	PLL_F240M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	PLL_F120M_CLK 120 MHz	PLL_F80M_CLK 80 MHz	PLL_F48M_CLK 48 MHz	RC_FAST_CLK 20 MHz	HP_ROOT_CLK 240 MHz/160 MHz/48 MHz/20 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_FAST_CLK	LP_DYN_SLOW_CLK	XTAL_D2_CLK 24 MHz	LP_FAST_CLK	Clock from IO
定时器组 (TIMG)	Y				Y		Y										
主系统看门狗定时器 (MWDT)	Y				Y		Y										
I2S 控制器 (I2S)	Y	Y	Y														I2S_MCLK_in
UART 控制器 (UART)	Y				Y		Y										
红外遥控 (RMT)	Y					Y											
电机控制脉宽调制器 (MCPWM)	Y		Y				Y										
I2C 控制器 (I2C)	Y						Y										
SPI2	Y		Y	Y			Y										
SAR ADC	Y				Y		Y										
USB 串口/JTAG 控制器						Y											
CAN FD	Y						Y										
LED PWM 控制器 (LEDC)	Y	Y				Y											
系统定时器	Y						Y										
并行 IO 控制器 (PARLIO)	Y	Y					Y										parl_rx_clk_in parl_tx_clk_in
IO MUX	Y				Y		Y										
AES SHA RSA ECC RSA_DS HMAC 密钥管理器 ECDSA_DS									Y								
中断矩阵											Y						
SDIO 从机控制器 (SDIO)											Y						
脉冲计数控制器 (PCNT)										Y							
事件任务矩阵 (ETM)											Y						
通用 DMA 控制器 (GDMA)											Y						
UHCI											Y						
辅助调试												Y					
RISC-V 追踪编码器 (TRACE)											Y	Y					
中断优先级寄存器 (INTPRI)												Y					

表 9.2-5. 衍生低功耗时钟源

衍生时钟	源时钟		衍生时钟		源时钟				衍生时钟									源时钟
		PLL_CLK							HP_ROOT_CLK 240 MHz/160 MHz/48 MHz/20 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_FAST_CLK	LP_DYN_SLOW_CLK	XTAL_D2_CLK 24 MHz	LP_FAST_CLK	Clock from IO
	XTAL_CLK 48 MHz	PLL_F480M_CLK 480 MHz	PLL_F240M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	RC_FAST_CLK 20 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz										
LP_DYN_FAST_CLK						Y	Y	Y									Y	
LP_DYN_SLOW_CLK						Y	Y	Y										
XTAL_D2_CLK	Y																	
LP_FAST_CLK	Y				Y											Y		

GoBack

GoBack

表 9.2-6. 低功耗系统外设时钟

衍生时钟	源时钟		衍生时钟		源时钟				衍生时钟									源时钟
	XTAL_CLK 48 MHz	PLL_F480M_CLK 480 MHz	PLL_CLK PLL_F120M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	RC_FAST_CLK 20 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	HP_ROOT_CLK 240 MHz/160 MHz/48 MHz/20 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_FAST_CLK	LP_DYN_SLOW_CLK	XTAL_D2_CLK 24 MHz	LP_FAST_CLK	Clock from IO
eFuse 控制器 (eFuse)																Y		
RTC 看门狗定时器 (RWDT)														Y	Y			
RTC 定时器 (RTC Timer)														Y	Y			
欠压检测器 (Brownout Detector)														Y				
电源管理单元 (PMU)														Y	Y		Y	

PLL_F480M_CLK 时钟

PLL_F480M_CLK 的频率是 480 MHz。PLL_F240M_CLK、PLL_F160M_CLK、PLL_F80M_CLK、PLL_F48M_CLK、PLL_F40M_CLK 均为 PLL_F480M_CLK 分频所得。

CRYPTO_CLK 时钟

如表 9.2-3 所示，可选择 XTAL_CLK、PLL_F480M_CLK 或 RC_FAST_CLK 作为 CRYPTO_CLK 的时钟源，CRYPTO_CLK 时钟频率最大为 160 MHz。

为了防止 DPA (Differential Power Analysis) 攻击加解密外设，加解密功能时钟采用随机分频策略。根据随机分频范围划分为四个安全等级。可通过配置 HP_SYSTEM_SEC_DPA_CONF_REG 选择安全等级。当 HP_SYSTEM_SEC_DPA_CFG_SEL 为 1 时，安全等级由 eFuse 中的配置信息 (EFUSE_SEC_DPA_LEVEL) 决定，否则由 HP_SYSTEM_SEC_DPA_LEVEL 的值决定。

LED_PWM 时钟

LEDC 模块能将 PLL_F80M_CLK、RC_FAST_CLK 以及 XTAL_CLK 作为时钟源使用，即在 APB_CLK 关闭的时候，LEDC 也可工作。换而言之，当系统处于低功耗模式时，其它外设都将停止工作（APB_CLK 关闭），但是 LEDC 仍然可以通过 RC_FAST_CLK 来正常工作。

TIMG0 定时器组慢速时钟

TIMG0 定时器组可以以 XTAL_CLK 为基准时钟，计算 ESP32-C5 提供的慢速时钟的实际频率。慢速时钟源可以通过配置 PCR_32K_SEL 进行选择，对应关系如下：

0：选择 XTAL32K_CLK 时钟
1：选择 OSC_SLOW_CLK 时钟
2：选择 RC_SLOW_CLK 时钟
3：选择 RC_FAST_CLK 时钟

低功耗系统时钟

低功耗系统外设的时钟相关寄存器在低功耗系统时钟复位寄存器中。可通过章节 9.4.2 *低功耗系统时钟寄存器列表* 查看各外设对应的时钟相关寄存器。

9.2.4.4 PMU 控制高性能系统时钟门控

在 ESP32-C5 的各个工作模式下，可提前配置以下寄存器字段，实现 电源管理单元 (PMU) 对高性能系统外设的时钟门控：

- PMU_HP_*x*_DIG_ICG_APB_EN (x = ACTIVE/MODEM/SLEEP)：控制高性能系统外设读写寄存器的时钟门控
- PMU_HP_*x*_DIG_ICG_FUNC_EN (x = ACTIVE/MODEM/SLEEP)：控制高性能系统外设的功能时钟门控。

具体配置流程请参考章节 低功耗管理。

表 9.2-7 和表 9.2-8 列出了 PMU 预配置寄存器位与高性能系统时钟门控的对应关系。

表 9.2-7. 控制模块的读写寄存器时钟门控与 PMU 相关寄存器映射关系

PMU_HP_*x*_DIG_ICG_APB_EN bit 位	控制模块的读写寄存器时钟门控
0	AES 加速器 (AES) SHA 加速器 (SHA) RSA 加速器 (RSA) ECC 加速器 (ECC) RSA 数字签名外设 (RSA_DS) HMAC 加速器 (HMAC) ECDSA 数字签名外设 (ECDSA_DS) 密钥管理器
1	通用 DMA 控制器 (GDMA)
2	SPI2
3	中断矩阵
4	I2S 控制器 (I2S)
6	UART0
7	UART1
8	UHCI
11	定时器组 0
12	定时器组 1
13	I2C 控制器 (I2C)
14	LED PWM 控制器 (LEDC)
15	红外遥控 (RMT)
16	系统定时器
17	USB 串口/JTAG 控制器
18	双线汽车接口 0
19	双线汽车接口 1
20	脉冲计数控制器 (PCNT)
21	电机控制脉宽调制器 (MCPWM)
22	事件任务矩阵 (ETM)
23	并行 IO 控制器 (PARLIO)
25	辅助调试
26	GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)
28	比特调节器

表 9.2-8. 控制模块的工作时钟门控与 PMU 相关寄存器映射关系

PMU_HP_x_DIG_ICG_FUNC_EN bit 位	控制模块的功能时钟门控
0	通用 DMA 控制器 (GDMA)
1	SPI2
2	I2S 接收侧
3	UART0
4	UART1
5	UHCI
6	USB 串口/JTAG 控制器
7	I2S 发送
10	MEM_MONITOR
11	SDIO 从机控制器 (SDIO)
12	温度传感器
13	定时器组 0
14	定时器组 1
16	事件任务矩阵 (ETM)
17	高性能处理器 ASSIST_DEBUG
18	系统定时器
19	AES 加速器 (AES) SHA 加速器 (SHA) RSA 加速器 (RSA) ECC 加速器 (ECC) RSA 数字签名外设 (RSA_DS) HMAC 加速器 (HMAC) ECDSA 数字签名外设 (ECDSA_DS) 密钥管理器
21	红外遥控 (RMT)
22	电机控制脉宽调制器 (MCPWM)
24	比特调节器
25	并行 IO 控制器 (PARLIO)
27	LED PWM 控制器 (LEDC)
28	GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)
29	I2C 控制器 (I2C)
30	双线汽车接口 0
31	双线汽车接口 1

9.3 配置流程

9.3.1 高性能系统时钟配置

当配置 HP_ROOT_CLK 的时钟选源 PCR_SOC_CLK_SEL，或 CPU_CLK 时钟分频 PCR_CPU_DIV_NUM，或 AHB_CLK 时钟分频 PCR_AHB_DIV_NUM 时，在配置完上述寄存器后，需要将 PCR_BUS_CLOCK_UPDATE 寄存器写 1 来生效上述配置。可通过读 PCR_BUS_CLOCK_UPDATE 寄存器判断上述配置是否更新完成，当为 0

时表示配置更新完成，为 1 表示未更新完成。

9.3.2 低功耗系统时钟配置

可配置 LP_CLKRST_SLOW_CLK_SEL 选择 LP_SLOW_CLK 的时钟源。

可配置 LP_CLKRST_FAST_CLK_SEL 选择 LP_FAST_CLK 的时钟源。

9.3.3 外设时钟复位配置

注意：

ESP32-C5 具有低功耗特性，因此有些外设时钟默认为关闭状态。在启用这些外设之前，必须将外设的时钟打开，即把相应的 CLK_EN 位置为 1，并且解除外设的复位状态，即把相应的 RST_EN 位置为 0。

大部分外设模块的时钟都可以分为总线时钟和功能时钟。

- 外设模块的总线时钟主要是用于配置外设模块的寄存器。
- 功能时钟主要是模块工作所使用的时钟，例如 UART 的参考时钟。

大部分模块的功能时钟可以选择多个时钟源。在门控寄存器的描述中会说明该寄存器属于总线（AHB_CLK、APB_CLK）时钟门控寄存器还是功能时钟门控寄存器。

ESP32-C5 的总线时钟开关、功能时钟的开关以及时钟源选择和时钟分频的配置寄存器均放在了 PCR 模块中。更多信息，详见 9.4 寄存器列表。

功能模块不工作时，可以配置 PCR 模块中对应寄存器关闭功能时钟，关闭模块功能时钟不会影响系统其他部分。

下文以 I2C 时钟配置为例：

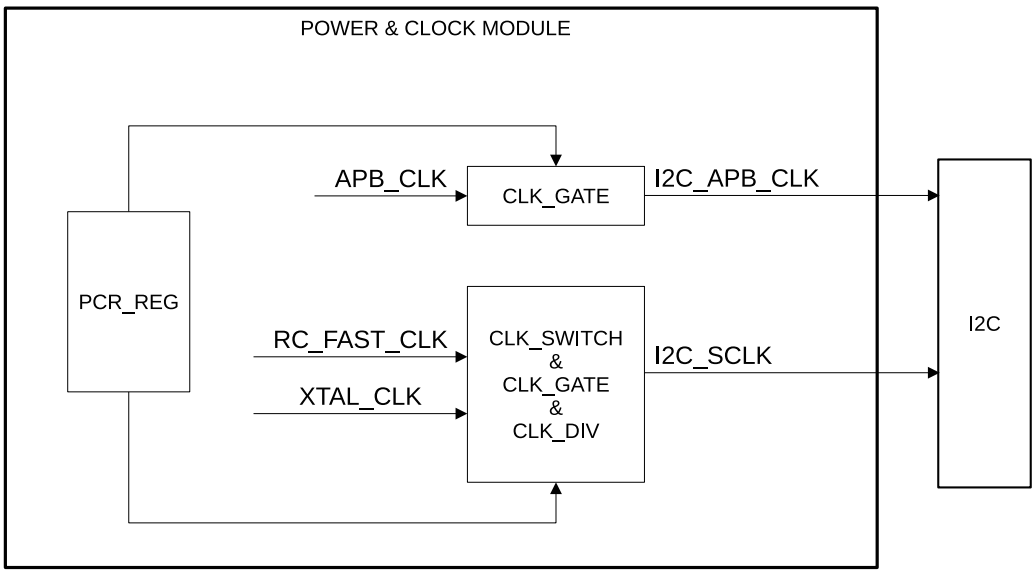


图 9.3-1. 时钟配置示例

图 9.3-1 是 I2C 的时钟结构图，其他模块的时钟结构类似。其中 CLK_SWITCH 的功能是选择一个时钟输出，CLK_GATE 的功能是打开/关闭时钟。

对于要求低功耗的场景，当模块不工作时，除了关闭功能时钟，还可以关闭模块的总线时钟来进一步降低功耗。

注意，如果先关闭模块总线时钟，那么模块的功能时钟还是有可能继续工作，所以推荐先关闭功能时钟再关闭总线时钟，打开时钟时推荐先打开总线时钟再打开功能时钟。

说明：

本章节中所有分频寄存器配置的值为实际分频值减去 1。

9.4 寄存器列表

9.4.1 PCR 模块寄存器列表

本小节的所有地址均为相对于电源/时钟/复位寄存器（PCR）基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
PCR_UART0_CONF_REG	UART0 配置寄存器	0x0000	varies
PCR_UART0_SCLK_CONF_REG	UART0_SCLK 配置寄存器	0x0004	R/W
PCR_UART0_PD_CTRL_REG	UART0 电源控制寄存器	0x0008	R/W
PCR_UART1_CONF_REG	UART1 配置寄存器	0x000C	varies
PCR_UART1_SCLK_CONF_REG	UART1_SCLK 配置寄存器	0x0010	R/W
PCR_UART1_PD_CTRL_REG	UART1 电源控制寄存器	0x0014	R/W
PCR_I2C_CONF_REG	I2C 配置寄存器	0x0020	R/W
PCR_I2C_SCLK_CONF_REG	I2C_SCLK 配置寄存器	0x0024	R/W
PCR_TWAI0_CONF_REG	TWAI0 配置寄存器	0x0028	varies
PCR_TWAI0_FUNC_CLK_CONF_REG	TWAI0_FUNC_CLK 配置寄存器	0x002C	R/W
PCR_TWAI1_CONF_REG	TWAI1 配置寄存器	0x0030	varies
PCR_TWAI1_FUNC_CLK_CONF_REG	TWAI1_FUNC_CLK 配置寄存器	0x0034	R/W
PCR_UHCI_CONF_REG	UHCI 配置寄存器	0x0038	varies
PCR_RMT_CONF_REG	RMT 配置寄存器	0x003C	R/W
PCR_RMT_SCLK_CONF_REG	RMT_SCLK 配置寄存器	0x0040	R/W
PCR_RMT_PD_CTRL_REG	RMT 电源控制寄存器	0x0044	R/W
PCR_LEDC_CONF_REG	LEDC 配置寄存器	0x0048	varies
PCR_LEDC_SCLK_CONF_REG	LEDC_SCLK 配置寄存器	0x004C	R/W
PCR_LEDC_PD_CTRL_REG	LEDC 电源控制寄存器	0x0050	R/W
PCR_TIMERGROUP0_CONF_REG	TIMERGROUP0 配置寄存器	0x0054	varies
PCR_TIMERGROUP0_TIMER_CLK_CONF_REG	TIMERGROUP0_TIMER_CLK 配置寄存器	0x0058	R/W
PCR_TIMERGROUP0_WDT_CLK_CONF_REG	TIMERGROUP0_WDT_CLK 配置寄存器	0x005C	R/W
PCR_TIMERGROUP1_CONF_REG	TIMERGROUP1 配置寄存器	0x0060	varies
PCR_TIMERGROUP1_TIMER_CLK_CONF_REG	TIMERGROUP1_TIMER_CLK 配置寄存器	0x0064	R/W
PCR_TIMERGROUP1_WDT_CLK_CONF_REG	TIMERGROUP1_WDT_CLK 配置寄存器	0x0068	R/W
PCR_SYSTIMER_CONF_REG	SYSTIMER 配置寄存器	0x006C	varies
PCR_SYSTIMER_FUNC_CLK_CONF_REG	SYSTIMER_FUNC_CLK 配置寄存器	0x0070	R/W
PCR_I2S_CONF_REG	I2S 配置寄存器	0x0074	varies
PCR_I2S_TX_CLKM_CONF_REG	I2S_TX_CLKM 配置寄存器	0x0078	R/W
PCR_I2S_TX_CLKM_DIV_CONF_REG	I2S_TX_CLKM_DIV 配置寄存器	0x007C	R/W
PCR_I2S_RX_CLKM_CONF_REG	I2S_RX_CLKM 配置寄存器	0x0080	R/W
PCR_I2S_RX_CLKM_DIV_CONF_REG	I2S_RX_CLKM_DIV 配置寄存器	0x0084	R/W

名称	描述	地址	访问
PCR_SARADC_CONF_REG	SARADC 配置寄存器	0x0088	R/W
PCR_SARADC_CLKM_CONF_REG	SARADC_CLKM 配置寄存器	0x008C	R/W
PCR_TSENS_CLK_CONF_REG	TSENS_CLK 配置寄存器	0x0090	R/W
PCR_USB_DEVICE_CONF_REG	USB_DEVICE 配置寄存器	0x0094	varies
PCR_INTMTX_CONF_REG	INTMTX 配置寄存器	0x0098	varies
PCR_PCNT_CONF_REG	PCNT 配置寄存器	0x009C	varies
PCR_ETM_CONF_REG	ETM 配置寄存器	0x00A0	varies
PCR_PWM_CONF_REG	PWM 配置寄存器	0x00A4	varies
PCR_PWM_CLK_CONF_REG	PWM_CLK 配置寄存器	0x00A8	R/W
PCR_PARL_IO_CONF_REG	PARL_IO 配置寄存器	0x00AC	varies
PCR_PARL_CLK_RX_CONF_REG	PARL_CLK_RX 配置寄存器	0x00B0	R/W
PCR_PARL_CLK_TX_CONF_REG	PARL_CLK_TX 配置寄存器	0x00B4	R/W
PCR_GDMA_CONF_REG	GDMA 配置寄存器	0x00C0	R/W
PCR_SPI2_CONF_REG	SPI2 配置寄存器	0x00C4	varies
PCR_SPI2_CLKM_CONF_REG	SPI2_CLKM 配置寄存器	0x00C8	R/W
PCR_AES_CONF_REG	AES 配置寄存器	0x00CC	varies
PCR_SHA_CONF_REG	SHA 配置寄存器	0x00D0	varies
PCR_RSA_CONF_REG	RSA 配置寄存器	0x00D4	varies
PCR_RSA_PD_CTRL_REG	RSA 电源控制寄存器	0x00D8	R/W
PCR_ECC_CONF_REG	ECC 配置寄存器	0x00DC	varies
PCR_ECC_PD_CTRL_REG	ECC 电源控制寄存器	0x00E0	R/W
PCR_DS_CONF_REG	DS 配置寄存器	0x00E4	varies
PCR_HMAC_CONF_REG	HMAC 配置寄存器	0x00E8	varies
PCR_ECDSA_CONF_REG	ECDSA_DS 配置寄存器	0x00EC	varies
PCR_IOMUX_CONF_REG	IOMUX 配置寄存器	0x00F0	R/W
PCR_IOMUX_CLK_CONF_REG	IOMUX_CLK 配置寄存器	0x00F4	R/W
PCR_TRACE_CONF_REG	TRACE 配置寄存器	0x00FC	R/W
PCR_ASSIST_CONF_REG	ASSIST 配置寄存器	0x0100	R/W
PCR_CACHE_CONF_REG	CACHE 配置寄存器	0x0104	R/W
PCR_TIMEOUT_CONF_REG	TIMEOUT 配置寄存器	0x010C	R/W
PCR_SYSCLK_CONF_REG	SYSCLK 配置寄存器	0x0110	varies
PCR_CPU_WAITI_CONF_REG	CPU_WAITI 配置寄存器	0x0114	varies
PCR_CPU_FREQ_CONF_REG	CPU_FREQ 配置寄存器	0x0118	R/W
PCR_AHB_FREQ_CONF_REG	AHB_FREQ 配置寄存器	0x011C	R/W
PCR_APB_FREQ_CONF_REG	APB_FREQ 配置寄存器	0x0120	R/W
PCR_PLL_DIV_CLK_EN_REG	SPLL DIV clock-gating 配置寄存器	0x0128	R/W
PCR_CTRL_32K_CONF_REG	32KHz clock 配置寄存器	0x0130	R/W
PCR_SEC_CONF_REG	片外存储器加密与解密模块时钟源配置寄存器	0x013C	R/W
PCR_BUS_CLK_UPDATE_REG	高性能系统时钟源更新使能寄存器	0x0144	R/ W/ WTC
PCR_SAR_CLK_DIV_REG	SAR ADC 时钟分频系数配置寄存器	0x0148	R/W
PCR_BS_CONF_REG	BS 配置寄存器	0x0150	R/W

名称	描述	地址	访问
PCR_BS_FUNC_CONF_REG	BS_FUNC_CLK 配置寄存器	0x0154	R/W
PCR_BS_PD_CTRL_REG	BS 电源控制寄存器	0x0158	R/W
PCR_TIMERGROUP_WDT_CONF_REG	TIMERGROUP_WDT 配置寄存器	0x015C	R/W
PCR_TIMERGROUP_XTAL_CONF_REG	TIMERGROUP1 配置寄存器	0x0160	R/W
PCR_KM_CONF_REG	密钥管理器配置寄存器	0x0164	R/W
PCR_KM_PD_CTRL_REG	密钥管理器电源控制寄存器	0x0168	R/W
PCR_TCM_MEM_MONITOR_CONF_REG	TCM_MEM_MONITOR 配置寄存器	0x016C	varies
PCR_PSRAM_MEM_MONITOR_CONF_REG	PSRAM_MEM_MONITOR 配置寄存器	0x0170	varies
PCR_HPCORE_0_PD_CTRL_REG	HP COREO 电源控制寄存器	0x0178	R/W
PCR_SDIO_SLAVE_CONF_REG	SDIO_SLAVE 配置寄存器	0x017C	R/W
频率数据寄存器			
PCR_SYSCLK_FREQ_QUERY_0_REG	SYSCLK frequency query 0 register	0x0124	HRO
版本寄存器			
PCR_DATE_REG	版本控制寄存器	0x0FFC	R/W

9.4.2 低功耗系统时钟寄存器列表

本小节前缀为 LPPERI 的寄存器（最后两个）地址为相对于低功耗外设寄存器（LPPERI）基地址的地址偏移量（相对地址），其余所有地址均为相对于低功耗时钟复位寄存器（LP_CLKRST）基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

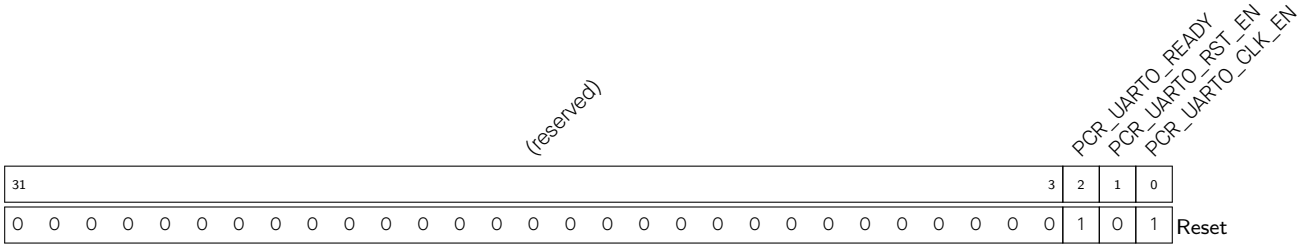
名称	描述	地址	访问
配置寄存器			
LP_CLKRST_LP_CLK_CONF_REG	LP 系统时钟源配置寄存器	0x0000	R/W
LP_CLKRST_LP_CLK_PO_EN_REG	管脚时钟信号门控配置寄存器	0x0004	R/W
LP_CLKRST_LP_CLK_EN_REG	LP 时钟源门控配置寄存器	0x0008	R/W
LP_CLKRST_LP_RST_EN_REG	LP 外设软件复位配置寄存器	0x000C	R/W
LP_CLKRST_RESET_CAUSE_REG	复位原因状态寄存器	0x0010	varies
LP_CLKRST_CPU_RESET_REG	CPU 复位配置寄存器	0x0014	R/W
LP_CLKRST_FOSC_CNTL_REG	RC_FAST_CLK 时钟频率配置寄存器	0x0018	R/W
LP_CLKRST_CLK_TO_HP_REG	HP 系统时钟信号门控配置寄存器	0x0020	R/W
LP_CLKRST_LPMEM_FORCE_REG	LP 存储器强制开启时钟门控配置寄存器	0x0024	R/W
LP_CLKRST_LPPERI_REG	LP 外设时钟配置寄存器	0x0028	R/W
LP_CLKRST_XTAL32K_REG	XTAL32K 配置寄存器	0x002C	R/W
LP_CLKRST_DATE_REG	版本控制寄存器	0x03FC	R/W
LPPERI_CLK_EN_REG	LP 外设的时钟使能寄存器	0x0000	R/W
LPPERI_RESET_EN_REG	LP 外设的复位使能寄存器	0x0004	R/W

9.5 寄存器

9.5.1 PCR 模块寄存器

本小节的所有地址均为相对于电源/时钟/复位寄存器（PCR）基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 9.1. PCR_UART0_CONF_REG (0x0000)



PCR_UART0_CLK_EN 配置是否使能 UART0 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_UART0_RST_EN 配置是否复位 UART0 模块。

0：不复位

1：复位

(R/W)

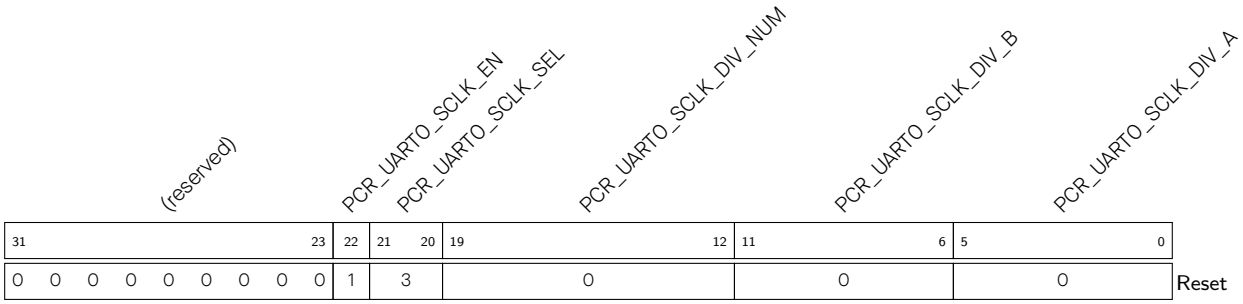
PCR_UART0_READY 表示 UART0 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.2. PCR_UART0_SCLK_CONF_REG (0x0004)



PCR_UART0_SCLK_DIV_A 配置 UART0 功能时钟分频系数小数部分的分母。(R/W)

PCR_UART0_SCLK_DIV_B 配置 UART0 功能时钟分频系数小数部分的分子。(R/W)

PCR_UART0_SCLK_DIV_NUM 配置 UART0 功能时钟分频系数的整数部分。(R/W)

PCR_UART0_SCLK_SEL 配置 UART0 时钟源。

0：不选择任何时钟

1：选择 PLL_F80M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3（默认）：选择 XTAL_CLK 时钟

(R/W)

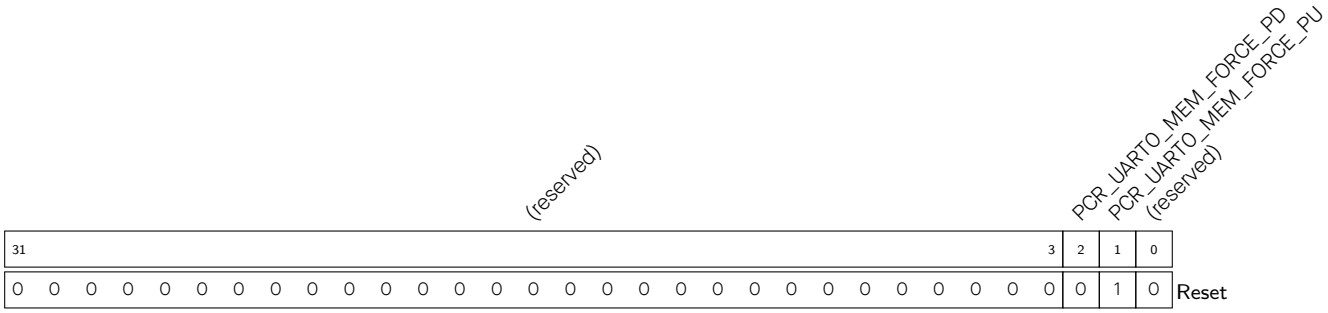
PCR_UART0_SCLK_EN 配置是否使能 UART0 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.3. PCR_UART0_PD_CTRL_REG (0x0008)



PCR_UART0_MEM_FORCE_PU 配置是否强制 UART0 存储器上电。

0：不强制 UART0 存储器上电

1：强制 UART0 存储器上电

(R/W)

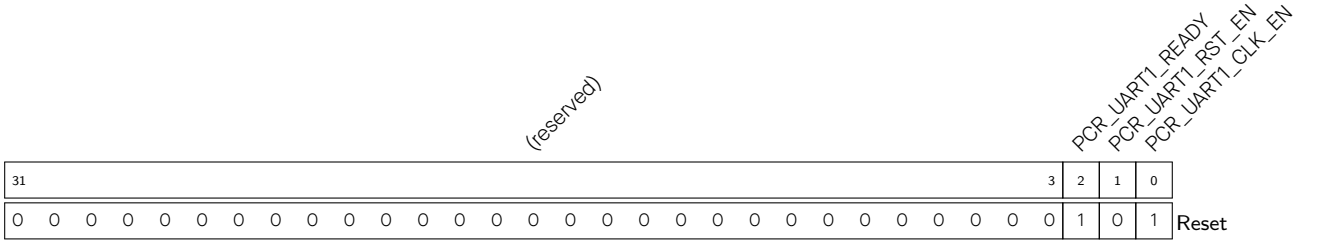
PCR_UART0_MEM_FORCE_PD 配置是否强制 UART0 存储器掉电。

0：不强制 UART0 存储器掉电

1：强制 UART0 存储器掉电

(R/W)

Register 9.4. PCR_UART1_CONF_REG (0x000C)



PCR_UART1_CLK_EN 配置是否使能 UART1 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_UART1_RST_EN 配置是否复位 UART1 模块。

0：不复位

1：复位

(R/W)

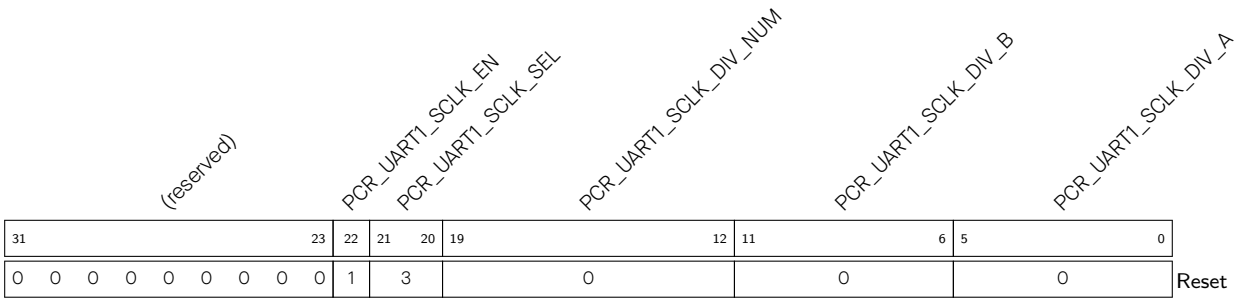
PCR_UART1_READY 表示 UART1 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.5. PCR_UART1_SCLK_CONF_REG (0x0010)



PCR_UART1_SCLK_DIV_A 配置 UART1 功能时钟分频系数小数部分的分母。(R/W)

PCR_UART1_SCLK_DIV_B 配置 UART1 功能时钟分频系数小数部分的分子。(R/W)

PCR_UART1_SCLK_DIV_NUM 配置 UART1 功能时钟分频系数的整数部分。(R/W)

PCR_UART1_SCLK_SEL 配置选择 UART1 时钟源。

0：不选择任何时钟

1：选择 PLL_F80M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3（默认）：选择 XTAL_CLK 时钟

(R/W)

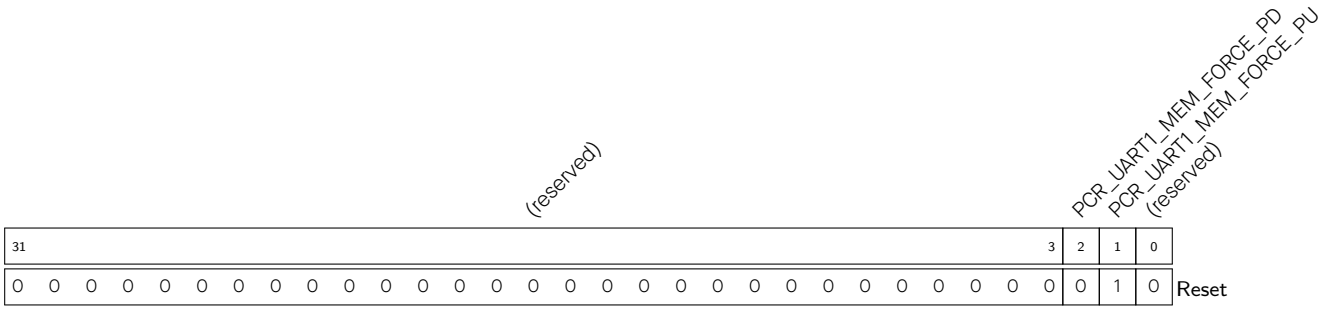
PCR_UART1_SCLK_EN 配置是否使能 UART1 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.6. PCR_UART1_PD_CTRL_REG (0x0014)



PCR_UART1_MEM_FORCE_PU 配置是否强制 UART1 存储器上电。

0：不强制 UART1 存储器上电

1：强制 UART1 存储器上电

(R/W)

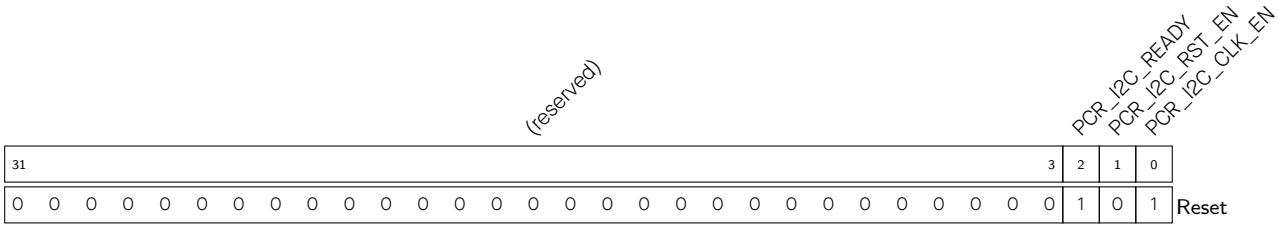
PCR_UART1_MEM_FORCE_PD 配置是否强制 UART1 存储器掉电。

0：不强制 UART1 存储器掉电

1：强制 UART1 存储器掉电

(R/W)

Register 9.7. PCR_I2C_CONF_REG (0x0020)



PCR_I2C_CLK_EN 配置是否使能 I2C APB_CLK。

0：不使能

1：使能

(R/W)

PCR_I2C_RST_EN 配置是否复位 I2C 模块。

0：不复位

1：复位

(R/W)

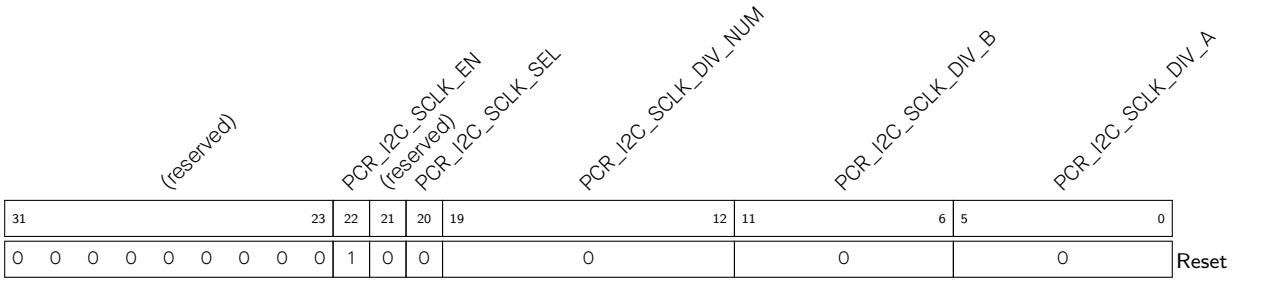
PCR_I2C_READY 表示 I2C 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.8. PCR_I2C_SCLK_CONF_REG (0x0024)



PCR_I2C_SCLK_DIV_A 配置 I2C 功能时钟分频系数小数部分的分母。(R/W)

PCR_I2C_SCLK_DIV_B 配置 I2C 功能时钟分频系数小数部分的分子。(R/W)

PCR_I2C_SCLK_DIV_NUM 配置 I2C 功能时钟分频系数的整数部分。(R/W)

PCR_I2C_SCLK_SEL 配置选择 I2C 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

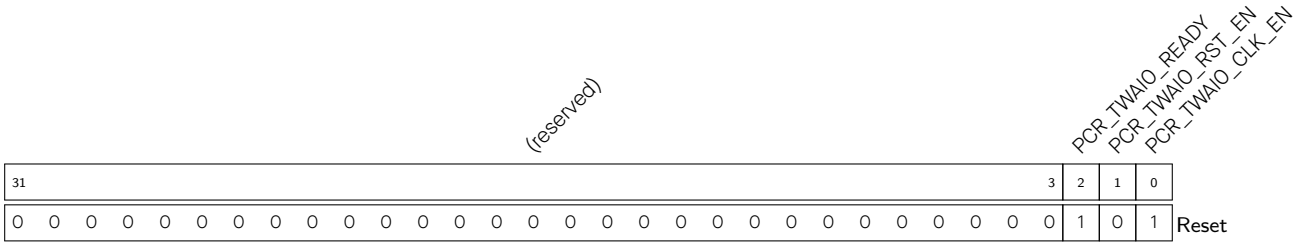
PCR_I2C_SCLK_EN 配置是否使能 I2C 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.9. PCR_TWAIO_CONF_REG (0x0028)



PCR_TWAIO_CLK_EN 配置是否使能 TWAIO APB_CLK。

0：不使能

1：使能

(R/W)

PCR_TWAIO_RST_EN 配置是否复位 TWAIO 模块。

0：不复位

1：复位

(R/W)

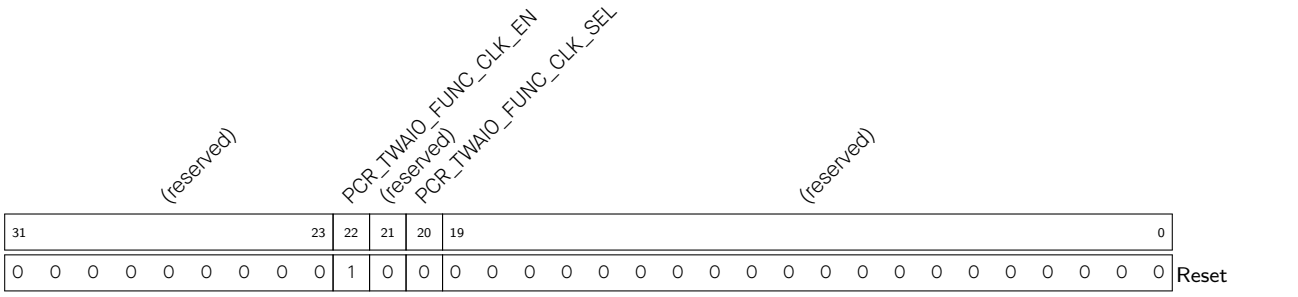
PCR_TWAIO_READY 表示 TWAIO 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.10. PCR_TWAIO_FUNC_CLK_CONF_REG (0x002C)



PCR_TWAIO_FUNC_CLK_SEL 配置选择 TWAIO 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

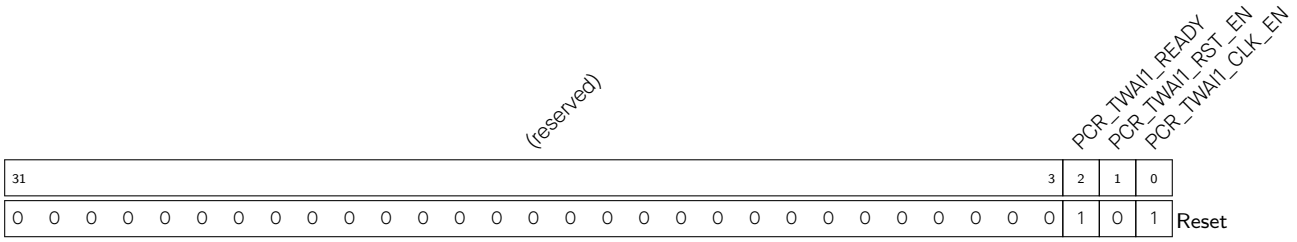
PCR_TWAIO_FUNC_CLK_EN 配置是否使能 TWAIO 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.11. PCR_TWAI1_CONF_REG (0x0030)



PCR_TWAI1_CLK_EN 配置是否使能 TWAI1 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_TWAI1_RST_EN 配置是否复位 TWAI1 模块。

0：不复位

1：复位

(R/W)

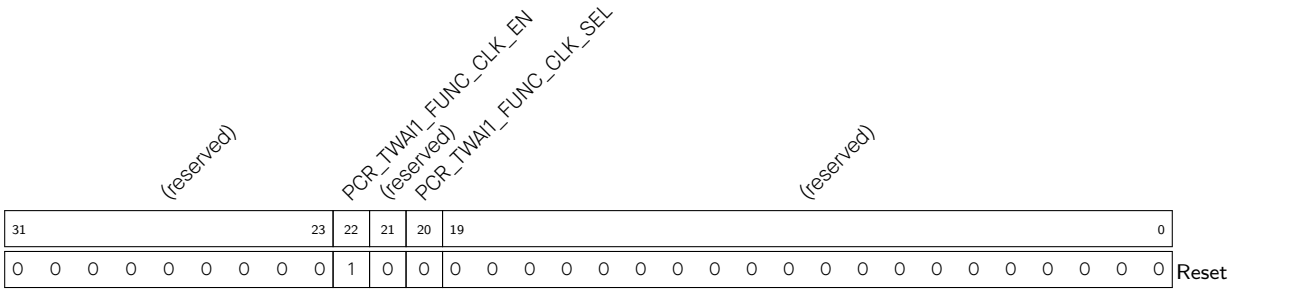
PCR_TWAI1_READY 表示 TWAI1 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.12. PCR_TWAI1_FUNC_CLK_CONF_REG (0x0034)



PCR_TWAI1_FUNC_CLK_SEL 配置选择 TWAI1 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

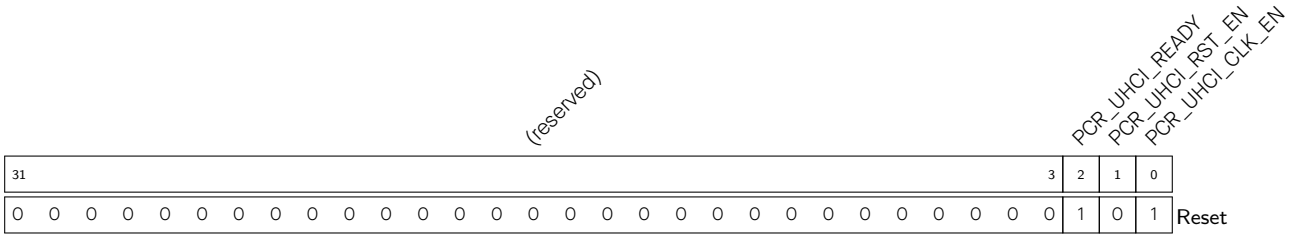
PCR_TWAI1_FUNC_CLK_EN 配置是否使能 TWAI1 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.13. PCR_UHCI_CONF_REG (0x0038)



PCR_UHCI_CLK_EN 配置是否使能 UHCI 时钟。

0：不使能

1：使能

(R/W)

PCR_UHCI_RST_EN 配置是否复位 UHCI 模块。

0：不复位

1：复位

(R/W)

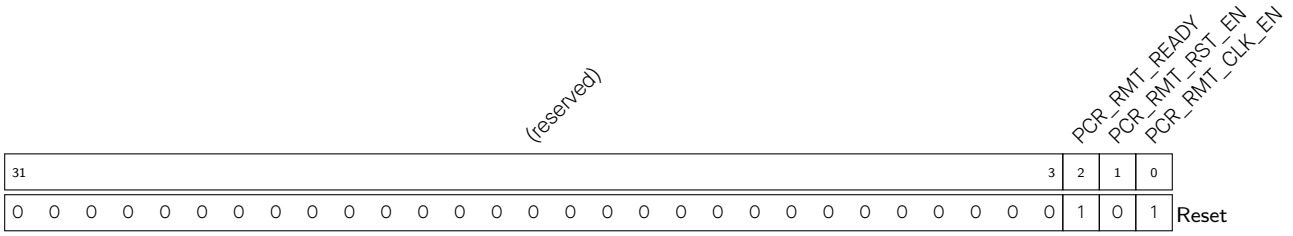
PCR_UHCI_READY 表示 UHCI 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.14. PCR_RMT_CONF_REG (0x003C)



PCR_RMT_CLK_EN 配置是否使能 RMT APB_CLK。

0：不使能

1：使能

(R/W)

PCR_RMT_RST_EN 配置是否复位 RMT 模块。

0：不复位

1：复位

(R/W)

PCR_RMT_READY 表示 RMT 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.15. PCR_RMT_SCLK_CONF_REG (0x0040)



PCR_RMT_SCLK_DIV_A 配置 RMT 功能时钟分频系数小数部分的分母。(R/W)

PCR_RMT_SCLK_DIV_B 配置 RMT 功能时钟分频系数小数部分的分子。(R/W)

PCR_RMT_SCLK_DIV_NUM 配置 RMT 功能时钟分频系数的整数部分。(R/W)

PCR_RMT_SCLK_SEL 配置选择 RMT 时钟源。

0：选择 XTAL_CLK 时钟

1（默认）：选择 RC_FAST_CLK 时钟

(R/W) 2：选择 PLL_F80M_CLK 时钟

(R/W)

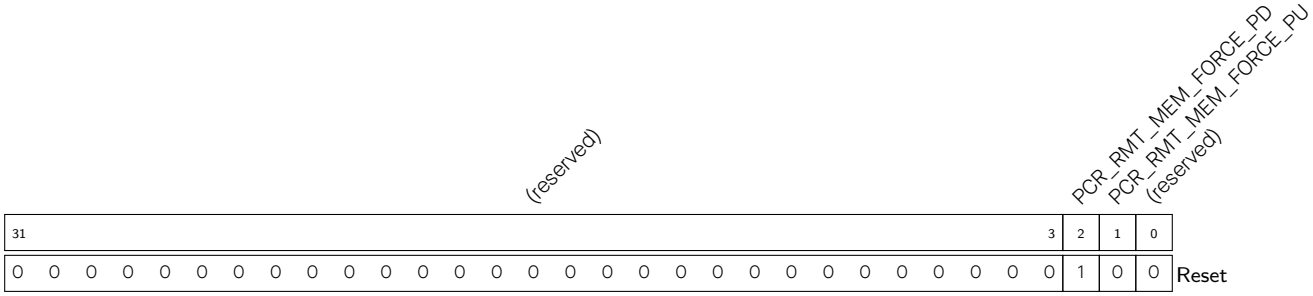
PCR_RMT_SCLK_EN 配置是否使能 RMT 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.16. PCR_RMT_PD_CTRL_REG (0x0044)



PCR_RMT_MEM_FORCE_PU 配置是否强制 RMT 存储器上电。

0：不强制 RMT 存储器上电

1：强制 RMT 存储器上电

(R/W)

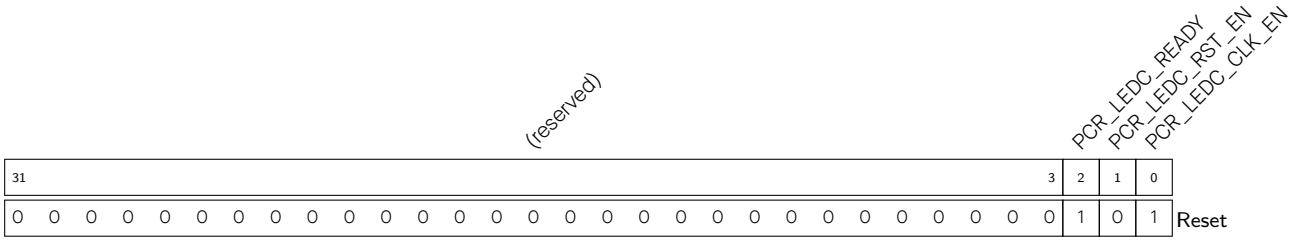
PCR_RMT_MEM_FORCE_PD 配置是否强制 RMT 存储器掉电。

0：不强制 RMT 存储器掉电

1：强制 RMT 存储器掉电

(R/W)

Register 9.17. PCR_LEDC_CONF_REG (0x0048)



PCR_LEDC_CLK_EN 配置是否使能 LEDC APB_CLK。

0：不使能

1：使能

(R/W)

PCR_LEDC_RST_EN 配置是否复位 LEDC 模块。

0：不复位

1：复位

(R/W)

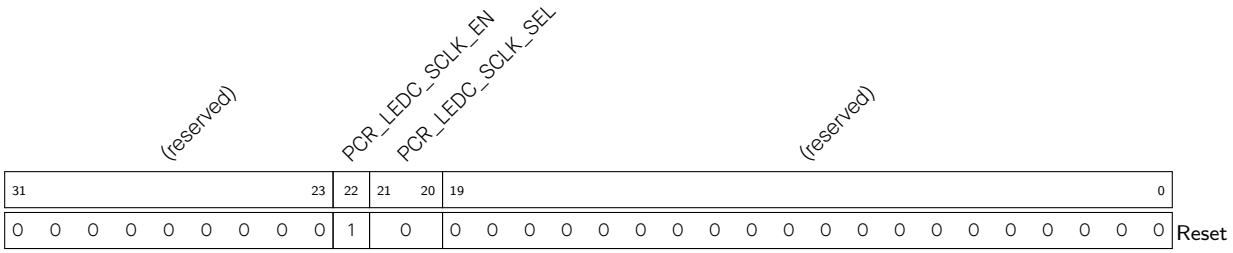
PCR_LEDC_READY 表示 LEDC 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.18. PCR_LEDC_SCLK_CONF_REG (0x004C)



PCR_LEDC_SCLK_SEL 配置选择 LEDC 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F80M_CLK 时钟

3：不选择任何时钟

(R/W)

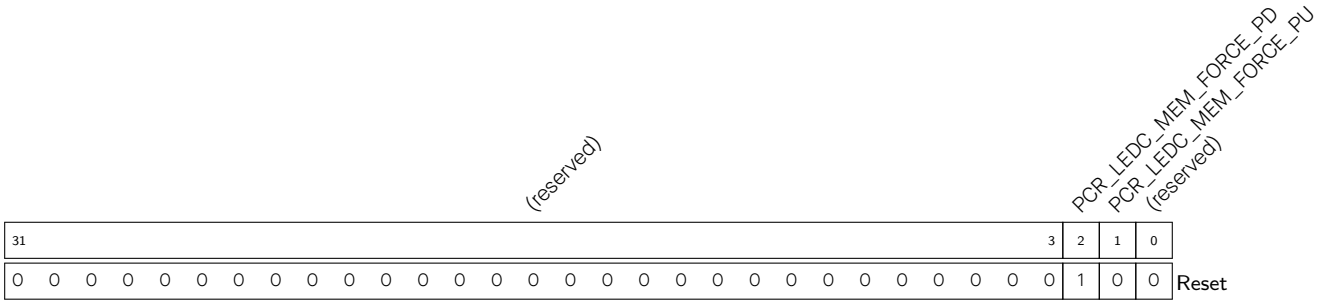
PCR_LEDC_SCLK_EN 配置是否使能 LEDC 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.19. PCR_LEDC_PD_CTRL_REG (0x0050)



PCR_LEDC_MEM_FORCE_PU 配置是否强制 LEDC 存储器上电。

0：不强制 LEDC 存储器上电

1：强制 LEDC 存储器上电

(R/W)

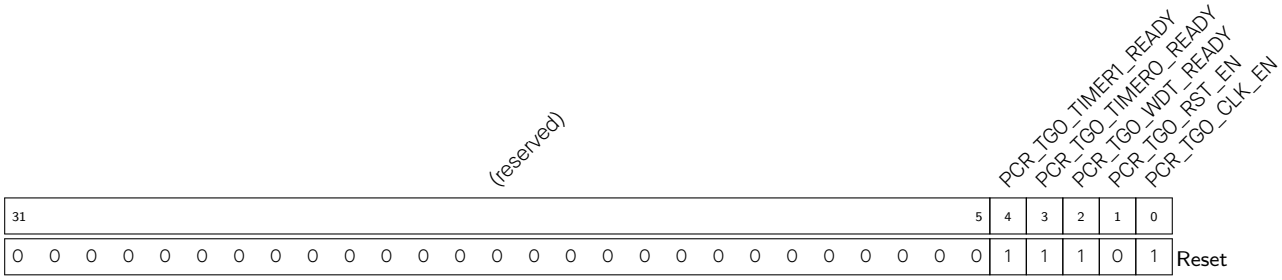
PCR_LEDC_MEM_FORCE_PD 配置是否强制 LEDC 存储器掉电。

0：不强制 LEDC 存储器掉电

1：强制 LEDC 存储器掉电

(R/W)

Register 9.20. PCR_TIMERGROUP0_CONF_REG (0x0054)



PCR_TG0_CLK_EN 配置是否使能定时器组 0 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_TG0_RST_EN 配置是否复位定时器组 0 模块。

0：不复位

1：复位

(R/W)

PCR_TG0_WDT_READY 表示定时器组 0 中的 WDT 是否解复位完成。

0：未完成

1：完成

(RO)

PCR_TG0_TIMER0_READY 表示定时器组 0 中的定时器 0 是否解复位完成。

0：未完成

1：完成

(RO)

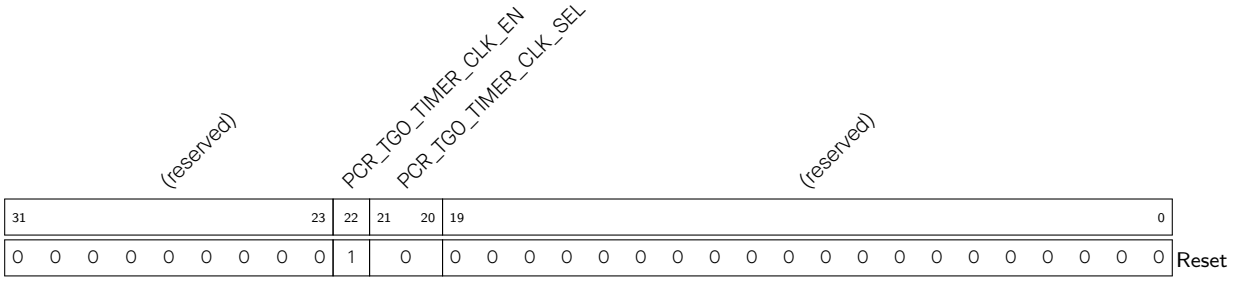
PCR_TG0_TIMER1_READY 表示定时器组 0 中的定时器 1 是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.21. PCR_TIMERGROUP0_TIMER_CLK_CONF_REG (0x0058)



PCR_TG0_TIMER_CLK_SEL 配置选择定时器组 0 通用定时器的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F80M_CLK 时钟

3：不选择任何时钟

(R/W)

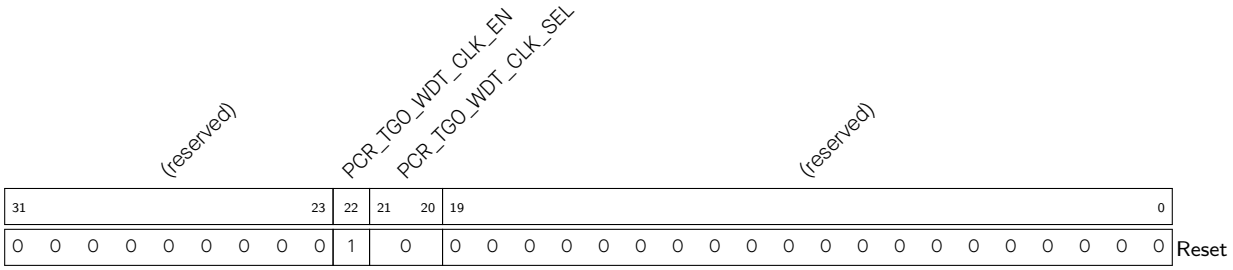
PCR_TG0_TIMER_CLK_EN 配置是否使能定时器组 0 通用定时器时钟。

0：不使能

1：使能

(R/W)

Register 9.22. PCR_TIMERGROUP0_WDT_CLK_CONF_REG (0x005C)



PCR_TG0_WDT_CLK_SEL 配置选择定时器组 0 中 WDT 的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F80M_CLK 时钟

3：不选择任何时钟

(R/W)

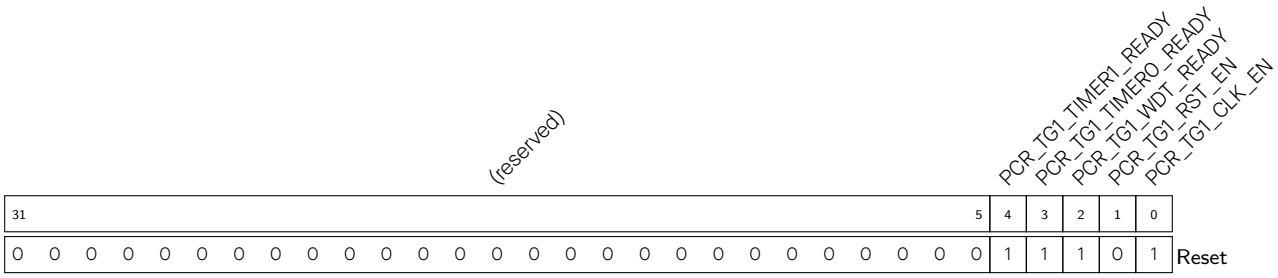
PCR_TG0_WDT_CLK_EN 配置是否使能定时器组 0 中 WDT 的时钟。

0：不使能

1：使能

(R/W)

Register 9.23. PCR_TIMERGROUP1_CONF_REG (0x0060)



PCR_TG1_CLK_EN 配置是否使能定时器组 1 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_TG1_RST_EN 配置是否复位定时器组 1 模块。

0：不复位

1：复位

(R/W)

PCR_TG1_WDT_READY 表示定时器组 1 中的看门狗定时器是否解复位完成。

0：未完成

1：完成

(RO)

PCR_TG1_TIMER0_READY 表示定时器组 1 中的定时器 0 是否解复位完成。

0：未完成

1：完成

(RO)

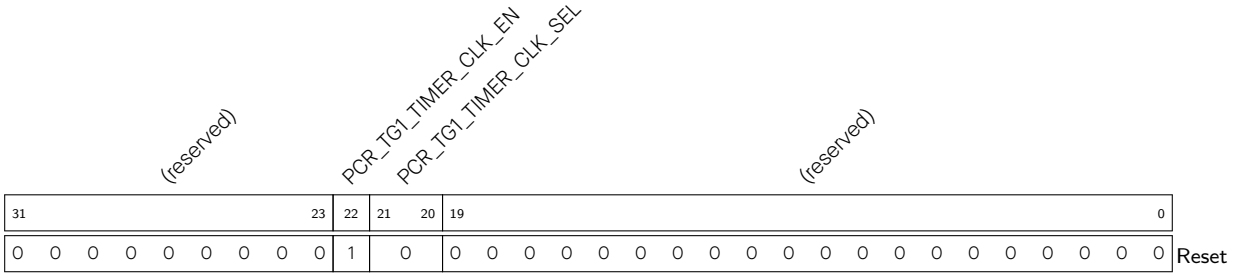
PCR_TG1_TIMER1_READY 表示定时器组 1 中的定时器 1 是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.24. PCR_TIMERGROUP1_TIMER_CLK_CONF_REG (0x0064)



PCR_TG1_TIMER_CLK_SEL 配置选择定时器组 1 通用定时器的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F80M_CLK 时钟

3：不选择任何时钟

(R/W)

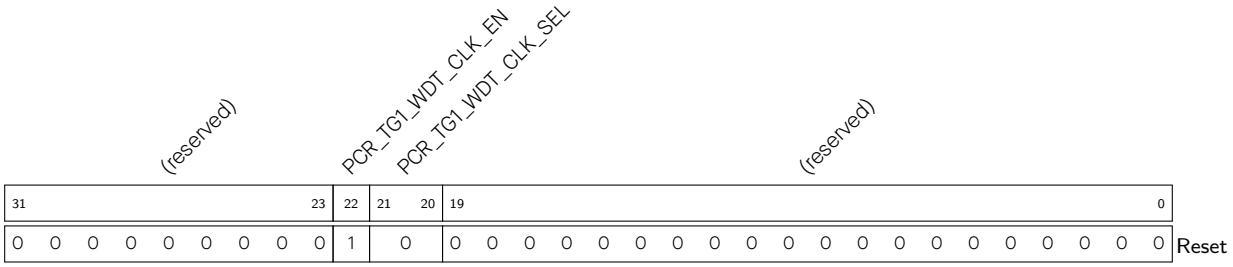
PCR_TG1_TIMER_CLK_EN 配置是否使能定时器组 1 通用定时器的时钟。

0：不使能

1：使能

(R/W)

Register 9.25. PCR_TIMERGROUP1_WDT_CLK_CONF_REG (0x0068)



PCR_TG1_WDT_CLK_SEL 配置选择定时器组 1 WDT 的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F80M_CLK 时钟

3：不选择任何时钟

(R/W)

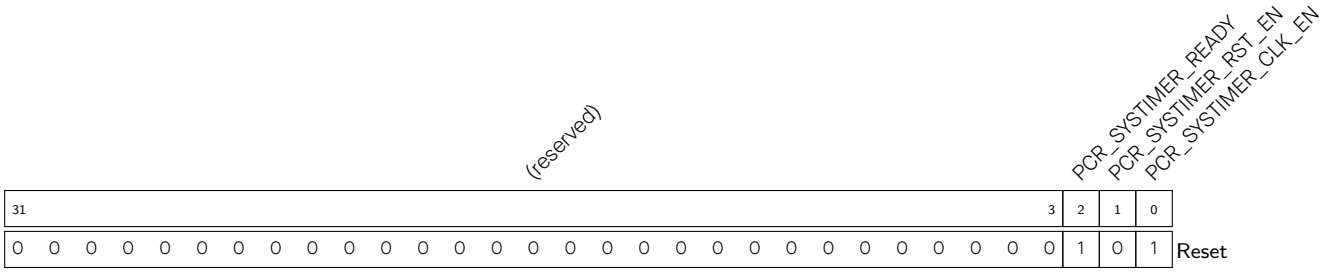
PCR_TG1_WDT_CLK_EN 配置是否使能定时器组 1 WDT 的时钟。

0：不使能

1：使能

(R/W)

Register 9.26. PCR_SYSTIMER_CONF_REG (0x006C)



PCR_SYSTIMER_CLK_EN 配置是否使能系统定时器 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_SYSTIMER_RST_EN 配置是否复位系统定时器模块。

0：不复位

1：复位

(R/W)

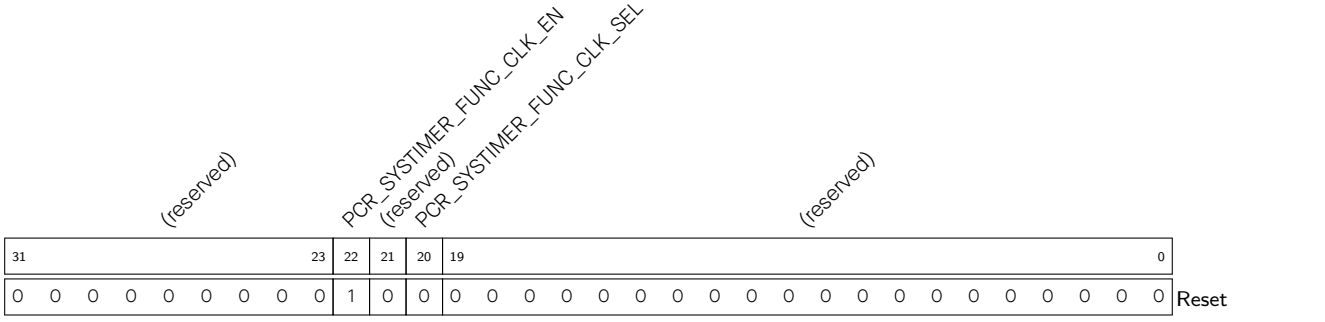
PCR_SYSTIMER_READY 表示系统定时器模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.27. PCR_SYSTIMER_FUNC_CLK_CONF_REG (0x0070)



PCR_SYSTIMER_FUNC_CLK_SEL 配置选择系统定时器时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

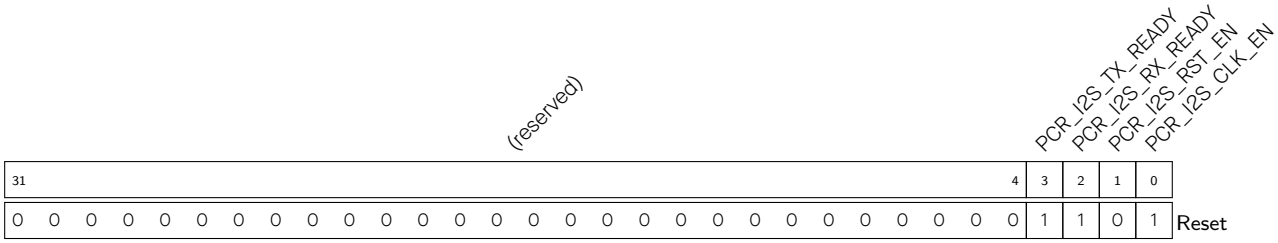
PCR_SYSTIMER_FUNC_CLK_EN 配置是否使能系统定时器功能时钟。

0：不使能

1：使能

(R/W)

Register 9.28. PCR_I2S_CONF_REG (0x0074)



PCR_I2S_CLK_EN 配置是否使能 I2S APB_CLK。

0：不使能

1：使能

(R/W)

PCR_I2S_RST_EN 配置是否复位 I2S 模块。

0：不复位

1：复位

(R/W)

PCR_I2S_RX_READY 表示 I2S RX 模块是否解复位完成。

0：未完成

1：完成

(RO)

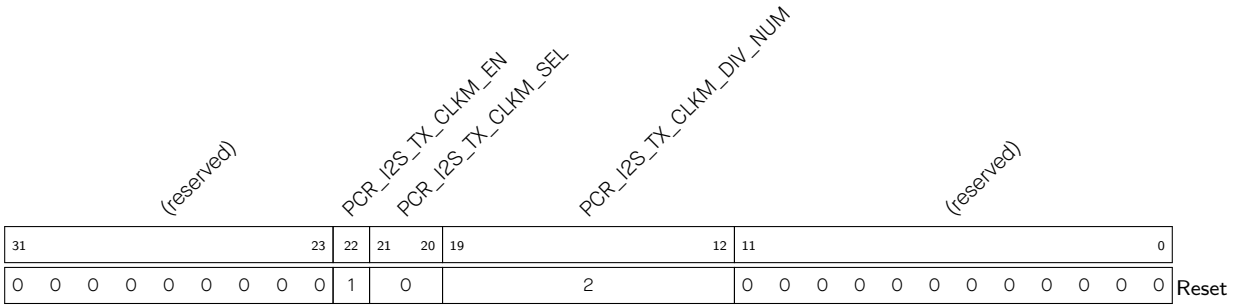
PCR_I2S_TX_READY 表示 I2S TX 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.29. PCR_I2S_TX_CLKM_CONF_REG (0x0078)



PCR_I2S_TX_CLKM_DIV_NUM 配置 I2S TX 时钟分频系数的整数部分。

(R/W)

PCR_I2S_TX_CLKM_SEL 配置选择 I2S TX 模块的时钟源。

0：选择 XTAL_CLK 时钟

1：选择 PLL_F240M_CLK 时钟

2：选择 PLL_F160M_CLK 时钟

3：选择 I2S_MCLK_in 时钟

(R/W)

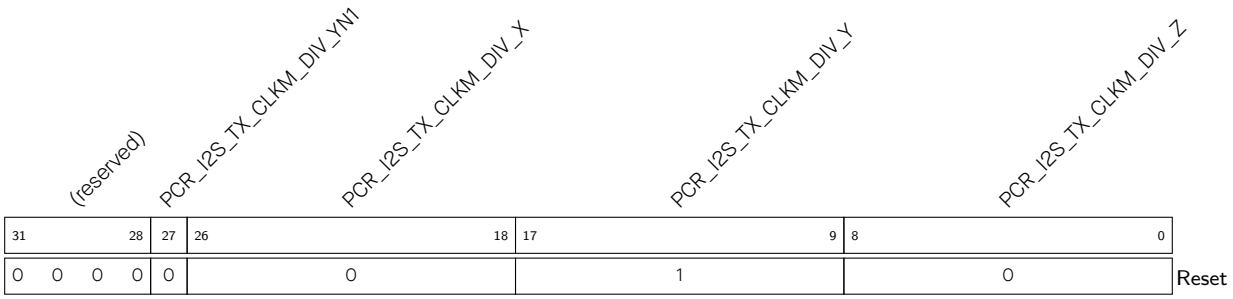
PCR_I2S_TX_CLKM_EN 配置是否使能 I2S TX 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.30. PCR_I2S_TX_CLKM_DIV_CONF_REG (0x007C)



PCR_I2S_TX_CLKM_DIV_Z b <= a/2 时，I2S_TX_CLKM_DIV_Z 的值为 b。b > a/2 时，I2S_TX_CLKM_DIV_Z 的值为 a - b。(R/W)

PCR_I2S_TX_CLKM_DIV_Y b <= a/2 时，I2S_TX_CLKM_DIV_Y 的值为 a%b。b > a/2 时，I2S_TX_CLKM_DIV_Y 的值为 a%(a - b)。(R/W)

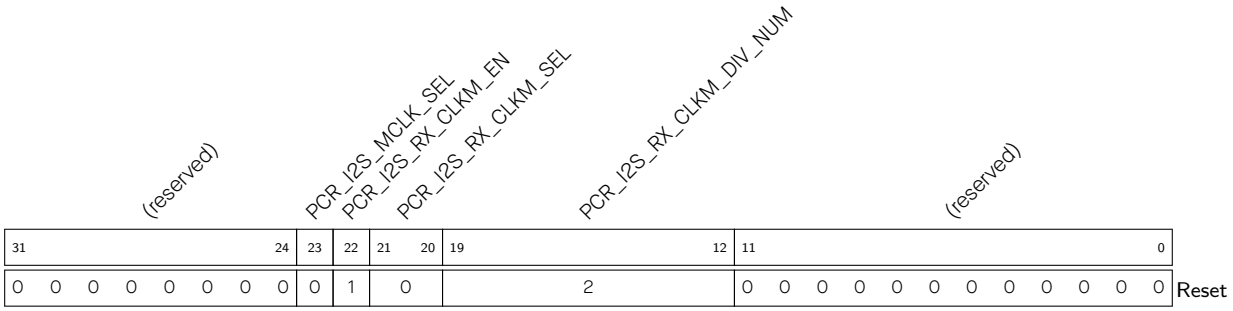
PCR_I2S_TX_CLKM_DIV_X b <= a/2 时，I2S_TX_CLKM_DIV_X 的值为 floor(a/b) - 1。b > a/2，I2S_TX_CLKM_DIV_X 的值为 floor(a/(a - b)) - 1。(R/W)

PCR_I2S_TX_CLKM_DIV_YN1 b <= a/2 时，I2S_TX_CLKM_DIV_YN1 的值为 0。b > a/2 时，I2S_TX_CLKM_DIV_YN1 的值为 1。(R/W)

说明：

上文所述的 “a” 和 “b” 分别为小数分频的分母部分和分子部分。更多信息，见章节 *I2S 控制器 (I2S)* > 章节 35.6。

Register 9.31. PCR_I2S_RX_CLKM_CONF_REG (0x0080)



PCR_I2S_RX_CLKM_DIV_NUM 配置 I2S 时钟的整数分频系数。(R/W)

PCR_I2S_RX_CLKM_SEL 配置选择 I2S RX 模块的时钟源。

0：选择 XTAL_CLK 时钟

1：选择 PLL_F240M_CLK 时钟

2：选择 PLL_F160M_CLK 时钟

3：选择 I2S_MCLK_in 时钟

(R/W)

PCR_I2S_RX_CLKM_EN 配置是否使能 I2S RX 功能时钟。

0：不使能

1：使能

(R/W)

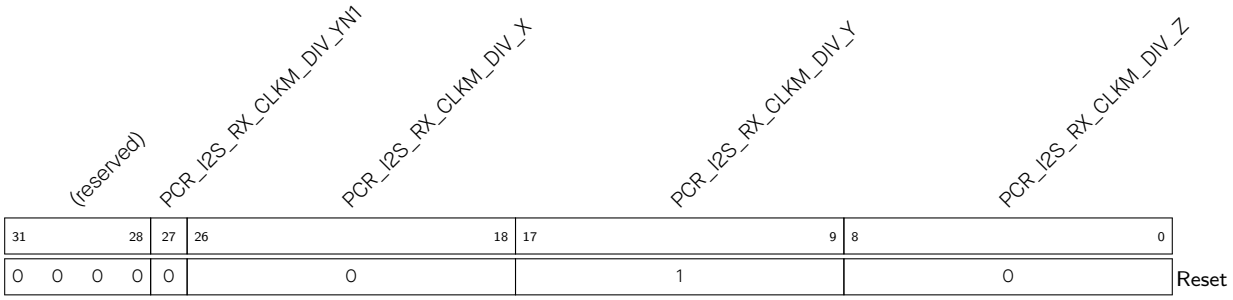
PCR_I2S_MCLK_SEL 配置选择主时钟。

0（默认）：选择 I2S_RX_CLK

1：选择 I2S_TX_CLK

(R/W)

Register 9.32. PCR_I2S_RX_CLKM_DIV_CONF_REG (0x0084)



PCR_I2S_RX_CLKM_DIV_Z b <= a/2 时，I2S_RX_CLKM_DIV_Z 的值为 b。b > a/2 时，I2S_RX_CLKM_DIV_Z 的值为 a - b。

(R/W)

PCR_I2S_RX_CLKM_DIV_Y b <= a/2 时，I2S_RX_CLKM_DIV_Y 的值为 a%b。b > a/2 时，I2S_RX_CLKM_DIV_Y 的值为 a%(a-b)。

(R/W)

PCR_I2S_RX_CLKM_DIV_X b <= a/2 时，I2S_RX_CLKM_DIV_X 的值为 floor(a/b) - 1。b > a/2，I2S_RX_CLKM_DIV_X 的值为 floor(a/(a - b)) - 1。

(R/W)

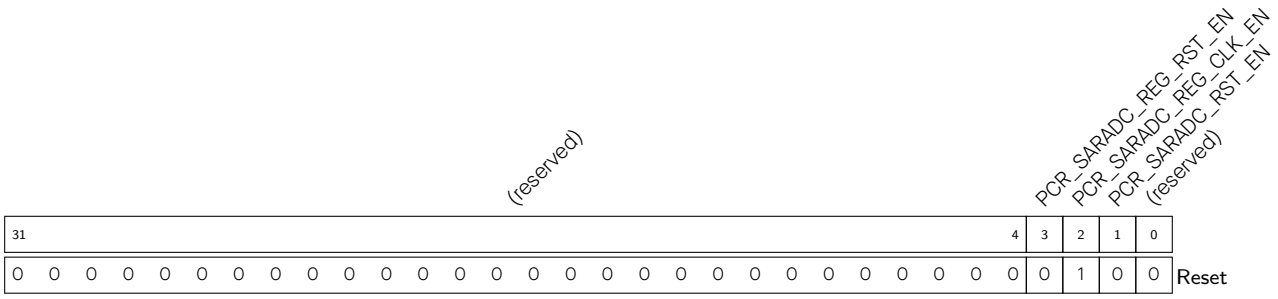
PCR_I2S_RX_CLKM_DIV_YN1 b <= a/2 时，I2S_RX_CLKM_DIV_YN1 的值为 0。b > a/2 时，I2S_RX_CLKM_DIV_YN1 的值为 1。

(R/W)

说明：

上文所述的 “a” 和 “b” 分别为小数分频的分母部分和分子部分。更多信息，见章节 35.6 。

Register 9.33. PCR_SARADC_CONF_REG (0x0088)



PCR_SARADC_RST_EN 配置是否复位 SAR ADC 功能寄存器。

0：不复位

1：复位

(R/W)

PCR_SARADC_REG_CLK_EN 配置是否使能 SAR ADC APB_CLK。

0：不使能

1：使能

(R/W)

PCR_SARADC_REG_RST_EN 配置是否复位 SAR ADC 模块的 APB 寄存器。

0：不复位

1：复位

(R/W)

Register 9.34. PCR_SARADC_CLKM_CONF_REG (0x008C)



PCR_SARADC_CLKM_DIV_A 配置 SAR ADC 功能时钟分频系数小数部分的分母。(R/W)

PCR_SARADC_CLKM_DIV_B 配置 SAR ADC 功能时钟分频系数小数部分的分子。(R/W)

PCR_SARADC_CLKM_DIV_NUM 配置 SAR ADC 功能时钟分频系数的整数部分。(R/W)

PCR_SARADC_CLKM_SEL 配置选择 SAR ADC 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F80M_CLK 时钟

3：不选择任何时钟

(R/W)

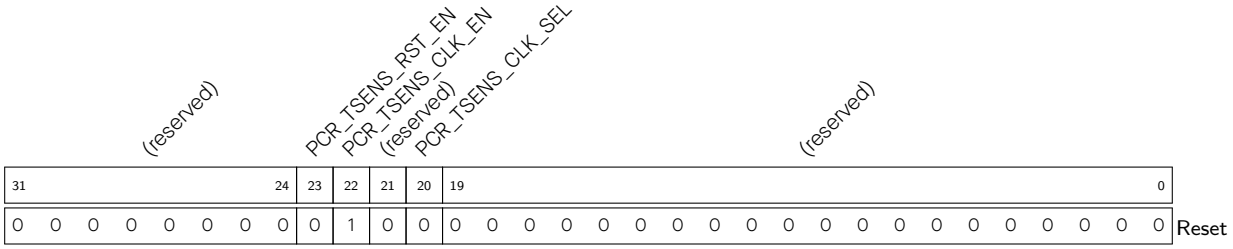
PCR_SARADC_CLKM_EN 配置是否使能 SAR ADC 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.35. PCR_TSENS_CLK_CONF_REG (0x0090)



PCR_TSENS_CLK_SEL 配置选择温度传感器时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

PCR_TSENS_CLK_EN 配置是否使能温度传感器时钟。

0：不使能

1：使能

(R/W)

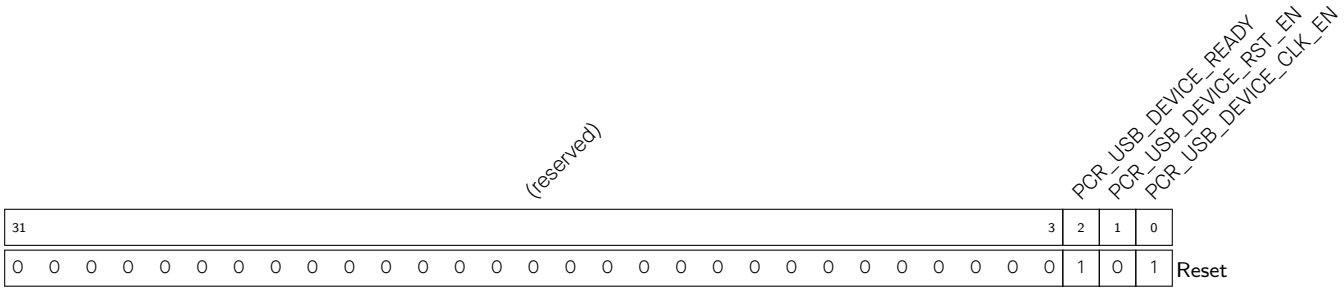
PCR_TSENS_RST_EN 配置是否复位温度传感器模块。

0：不复位

1：复位

(R/W)

Register 9.36. PCR_USB_DEVICE_CONF_REG (0x0094)



PCR_USB_DEVICE_CLK_EN 配置是否使能 USB Serial/JTAG 时钟。

0：不使能

1：使能

(R/W)

PCR_USB_DEVICE_RST_EN 配置是否复位 USB Serial/JTAG 模块。

0：不复位

1：复位

(R/W)

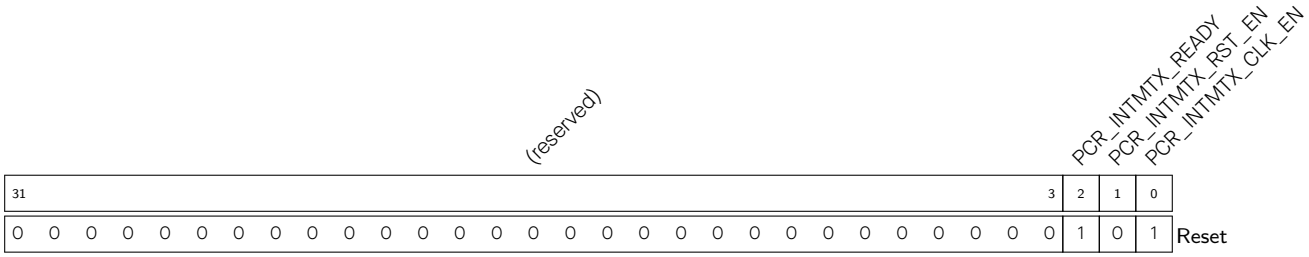
PCR_USB_DEVICE_READY 表示 USB Serial/JTAG 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.37. PCR_INTMTX_CONF_REG (0x0098)



PCR_INTMTX_CLK_EN 配置是否使能中断矩阵时钟。

0：不使能

1：使能

(R/W)

PCR_INTMTX_RST_EN 配置是否复位中断矩阵模块。

0：不复位

1：复位

(R/W)

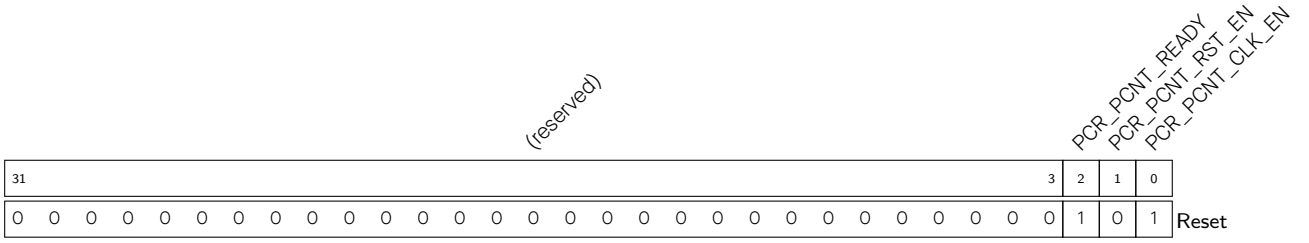
PCR_INTMTX_READY 表示中断矩阵模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.38. PCR_PCNT_CONF_REG (0x009C)



PCR_PCNT_CLK_EN 配置是否使能 PCNT 时钟。

0：不使能

1：使能

(R/W)

PCR_PCNT_RST_EN 配置是否复位 PCNT 模块。

0：不复位

1：复位

(R/W)

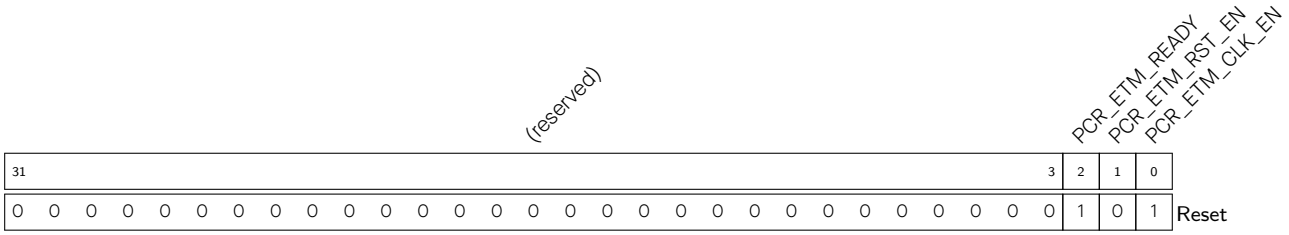
PCR_PCNT_READY 表示 PCNT 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.39. PCR_ETM_CONF_REG (0x00A0)



PCR_ETM_CLK_EN 配置是否使能 ETM 时钟。

0：不使能

1：使能

(R/W)

PCR_ETM_RST_EN 配置是否复位 ETM 模块。

0：不复位

1：复位

(R/W)

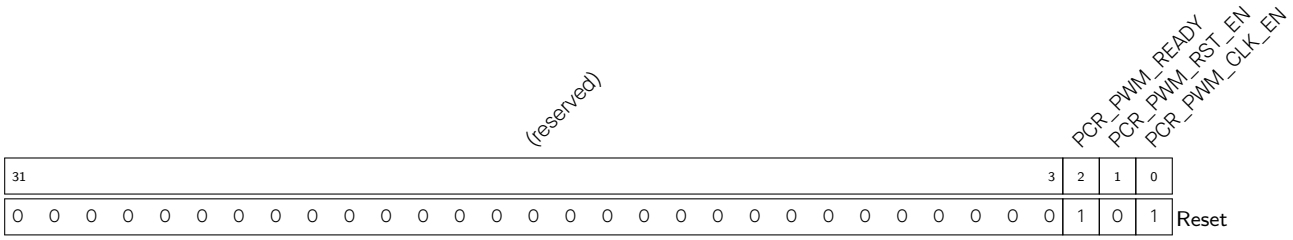
PCR_ETM_READY 表示 ETM 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.40. PCR_PWM_CONF_REG (0x00A4)



PCR_PWM_CLK_EN 配置是否使能 MCPWM 总线时钟。

0：不使能

1：使能

(R/W)

PCR_PWM_RST_EN 配置是否复位 MCPWM 模块。

0：不复位

1：复位

(R/W)

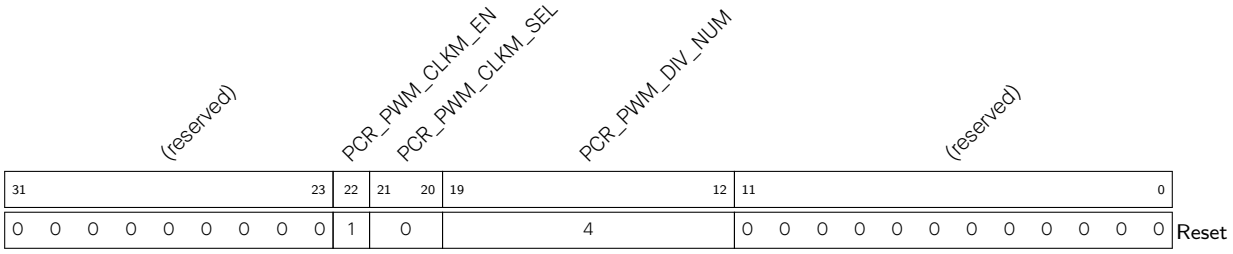
PCR_PWM_READY 表示 MCPWM 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.41. PCR_PWM_CLK_CONF_REG (0x00A8)



PCR_PWM_DIV_NUM 配置 MCPWM 功能时钟分频系数的整数部分。(R/W)

PCR_PWM_CLKM_SEL 配置选择 MCPWM 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F160M_CLK 时钟

3：不选择任何时钟

(R/W)

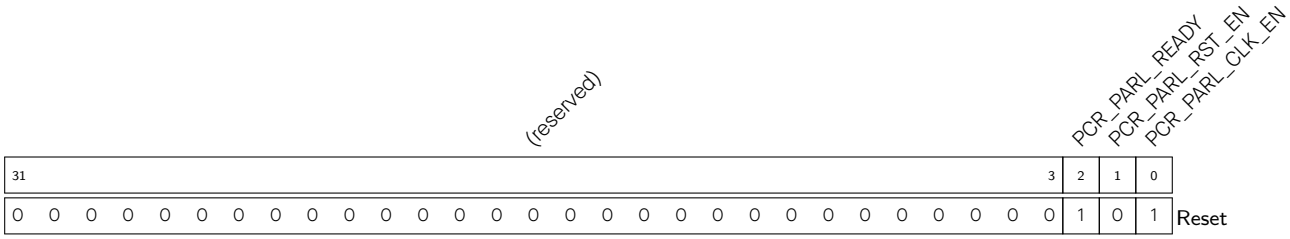
PCR_PWM_CLKM_EN 配置是否使能 MCPWM 的功能时钟。

0：不使能

1：使能

(R/W)

Register 9.42. PCR_PARL_IO_CONF_REG (0x00AC)



PCR_PARL_CLK_EN 配置是否使能并行 IO 控制器 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_PARL_RST_EN 配置是否复位并行 IO 控制器 APB 寄存器。

0：不复位

1：复位

(R/W)

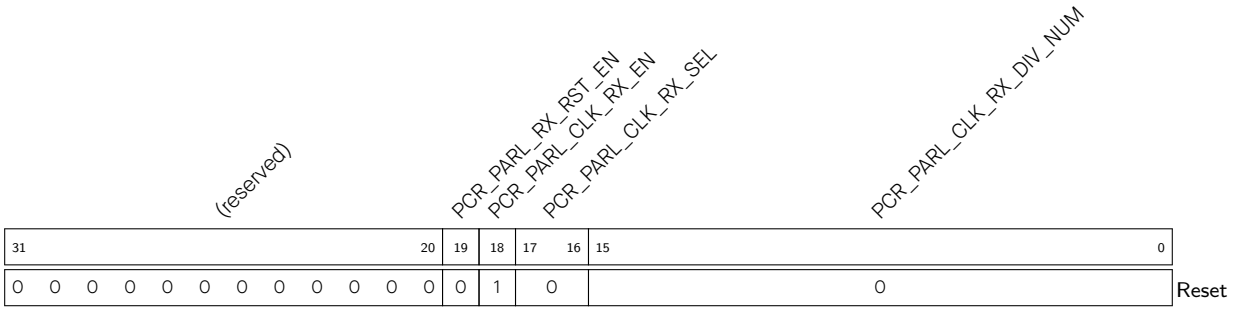
PCR_PARL_READY 表示并行 IO 控制器模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.43. PCR_PARL_CLK_RX_CONF_REG (0x00B0)



PCR_PARL_CLK_RX_DIV_NUM 配置并行 IO 控制器 RX 时钟的整数分频系数。(R/W)

PCR_PARL_CLK_RX_SEL 配置并行 IO 控制器 RX 选择时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F240M_CLK 时钟

3：使用来自管脚的时钟

(R/W)

PCR_PARL_CLK_RX_EN 配置是否使能并行 IO 控制器 RX 时钟。

0：不使能

1：使能

(R/W)

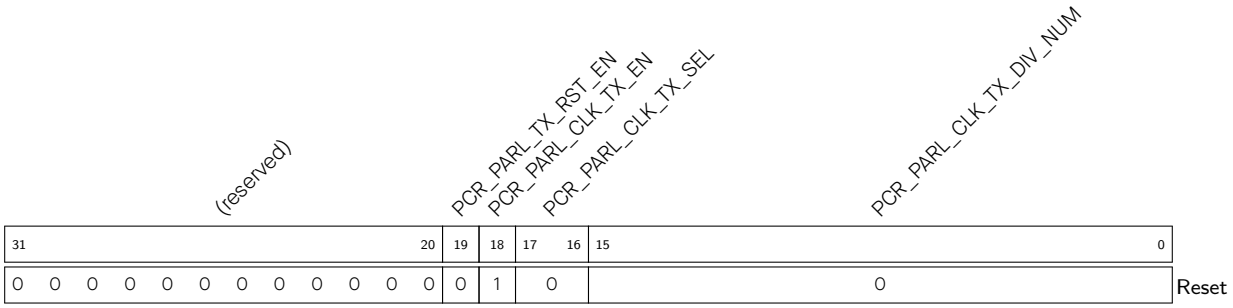
PCR_PARL_RX_RST_EN 配置是否复位并行 IO 控制器 RX 模块。

0：不复位

1：复位

(R/W)

Register 9.44. PCR_PARL_CLK_TX_CONF_REG (0x00B4)



PCR_PARL_CLK_TX_DIV_NUM 配置并行 IO 控制器 TX 时钟的整数分频系数。(R/W)

PCR_PARL_CLK_TX_SEL 配置选择并行 IO 控制器 TX 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F240M_CLK 时钟

3：使用来自管脚的时钟

(R/W)

PCR_PARL_CLK_TX_EN 配置是否使能并行 IO 控制器 TX 时钟。

0：不使能

1：使能

(R/W)

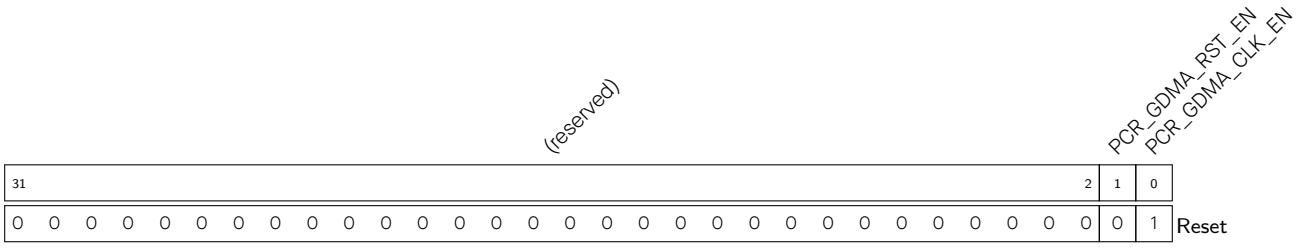
PCR_PARL_TX_RST_EN 配置是否复位并行 IO 控制器 TX 模块。

0：不复位

1：复位

(R/W)

Register 9.45. PCR_GDMA_CONF_REG (0x00C0)



PCR_GDMA_CLK_EN 配置是否使能 GDMA 时钟。

0：不使能

1：使能

(R/W)

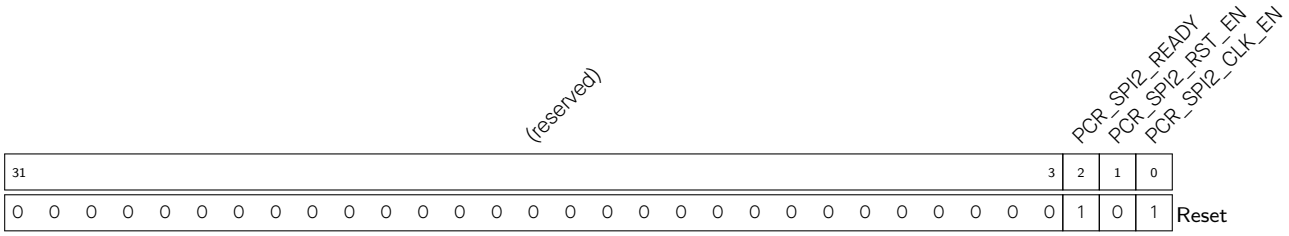
PCR_GDMA_RST_EN 配置是否复位 GDMA 模块。

0：不复位

1：复位

(R/W)

Register 9.46. PCR_SPI2_CONF_REG (0x00C4)



PCR_SPI2_CLK_EN 配置是否使能 SPI2 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_SPI2_RST_EN 配置是否复位 SPI2 模块。

0：不复位

1：复位

(R/W)

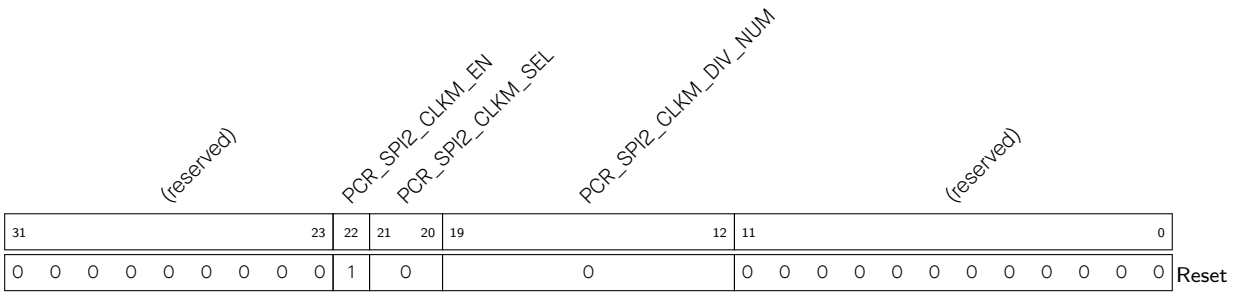
PCR_SPI2_READY 表示 SPI2 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.47. PCR_SPI2_CLKM_CONF_REG (0x00C8)



PCR_SPI2_CLKM_DIV_NUM 配置 clk_spi_mst 时钟分频系数的整数部分。

(R/W)

PCR_SPI2_CLKM_SEL 配置选择 SPI2 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 PLL_F160M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3：选择 PLL_F120M_CLK 时钟

(R/W)

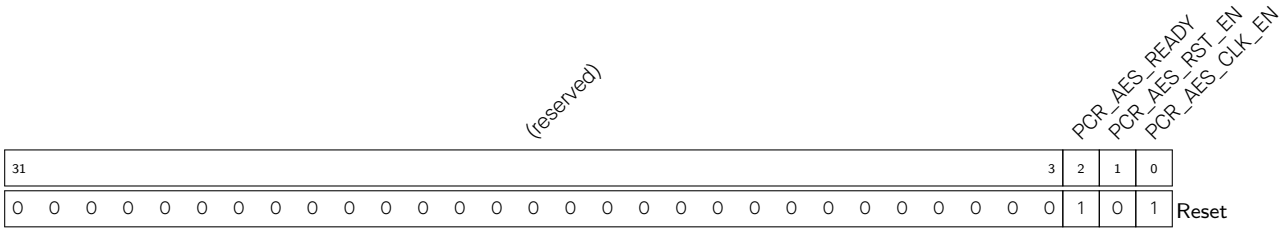
PCR_SPI2_CLKM_EN 配置是否使能 SPI2 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.48. PCR_AES_CONF_REG (0x00CC)



PCR_AES_CLK_EN 配置是否使能 AES 时钟。

0：不使能

1：使能

(R/W)

PCR_AES_RST_EN 配置是否复位 AES 模块。

0：不复位

1：复位

(R/W)

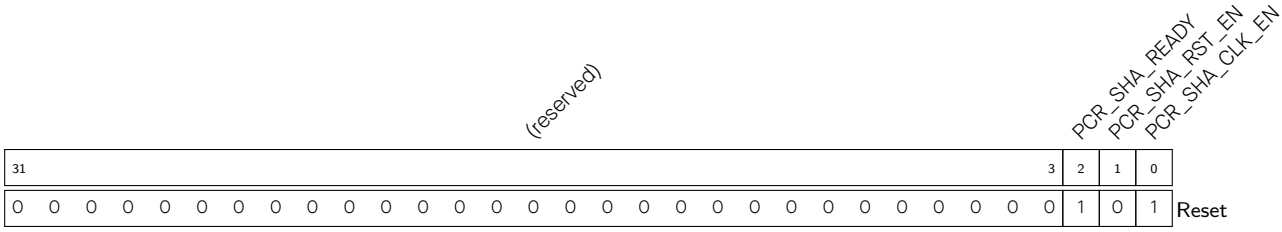
PCR_AES_READY 表示 AES 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.49. PCR_SHA_CONF_REG (0x00D0)



PCR_SHA_CLK_EN 配置是否使能 SHA 时钟。

0：不使能

1：使能

(R/W)

PCR_SHA_RST_EN 配置是否复位 SHA 模块。

0：不复位

1：复位

(R/W)

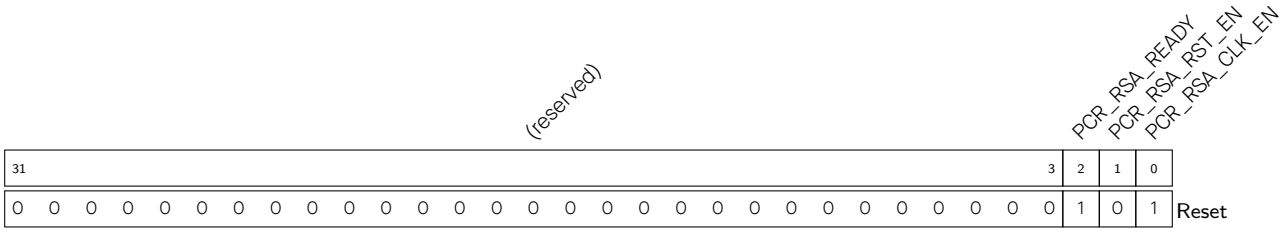
PCR_SHA_READY 表示 SHA 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.50. PCR_RSA_CONF_REG (0x00D4)



PCR_RSA_CLK_EN 配置是否使能 RSA 时钟。

0：不使能

1：使能

(R/W)

PCR_RSA_RST_EN 配置是否复位 RSA 模块。

0：不复位

1：复位

(R/W)

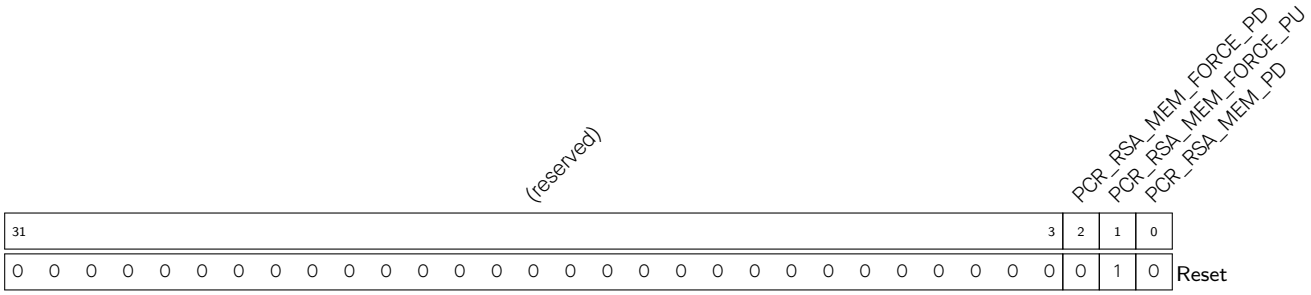
PCR_RSA_READY 表示 RSA 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.51. PCR_RSA_PD_CTRL_REG (0x00D8)



PCR_RSA_MEM_PD 配置是否使 RSA 内部存储器掉电。

0：不掉电

1：强制掉电

(R/W)

PCR_RSA_MEM_FORCE_PU 配置是否强制 RSA 内部存储器上电。

0：不强制上电

1：强制上电

(R/W)

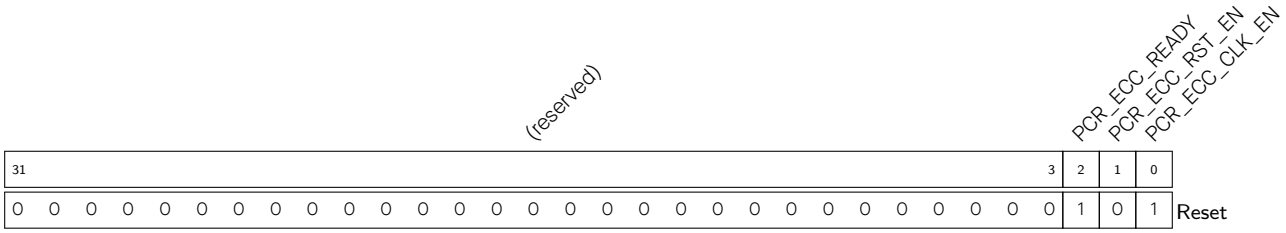
PCR_RSA_MEM_FORCE_PD 配置是否强制 RSA 内部存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

Register 9.52. PCR_ECC_CONF_REG (0x00DC)



PCR_ECC_CLK_EN 配置是否使能 ECC 时钟。

0：不使能

1：使能

(R/W)

PCR_ECC_RST_EN 配置是否复位 ECC 模块。

0：不复位

1：复位

(R/W)

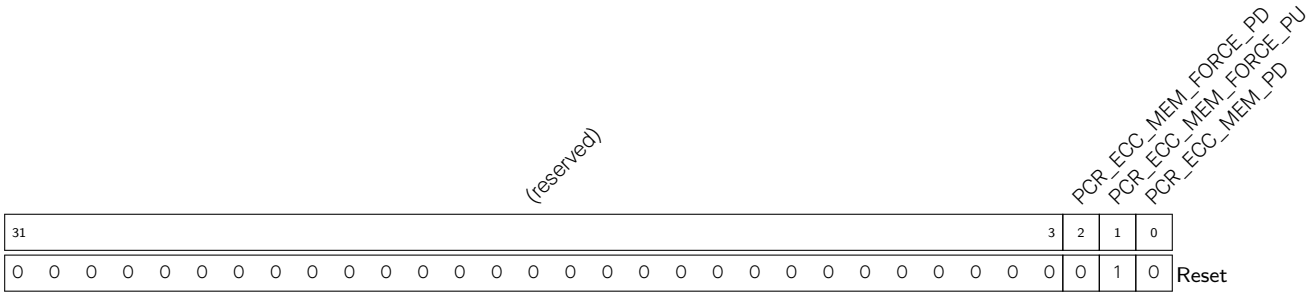
PCR_ECC_READY 表示 ECC 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.53. PCR_ECC_PD_CTRL_REG (0x00E0)



PCR_ECC_MEM_PD 配置是否使 ECC 内部存储器掉电。

0：不掉电

1：掉电

(R/W)

PCR_ECC_MEM_FORCE_PU 配置是否强制 ECC 内部存储器上电。

0：不强制上电

1：强制上电

(R/W)

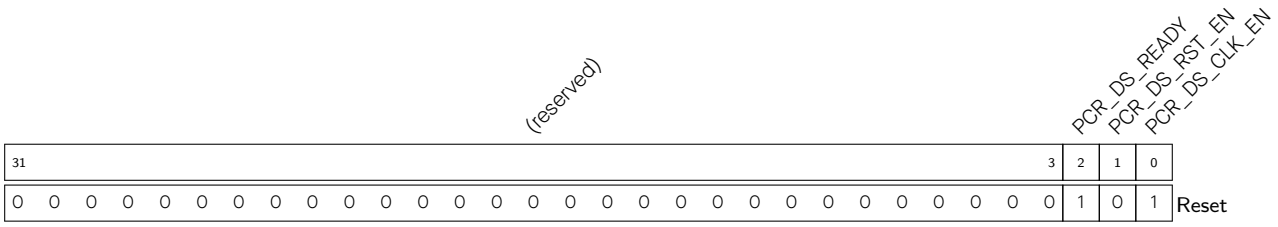
PCR_ECC_MEM_FORCE_PD 配置是否强制 ECC 内部存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

Register 9.54. PCR_DS_CONF_REG (0x00E4)



PCR_DS_CLK_EN 配置是否使能 DS 时钟。

0：不使能

1：使能

(R/W)

PCR_DS_RST_EN 配置是否复位 DS 模块。

0：不复位

1：复位

(R/W)

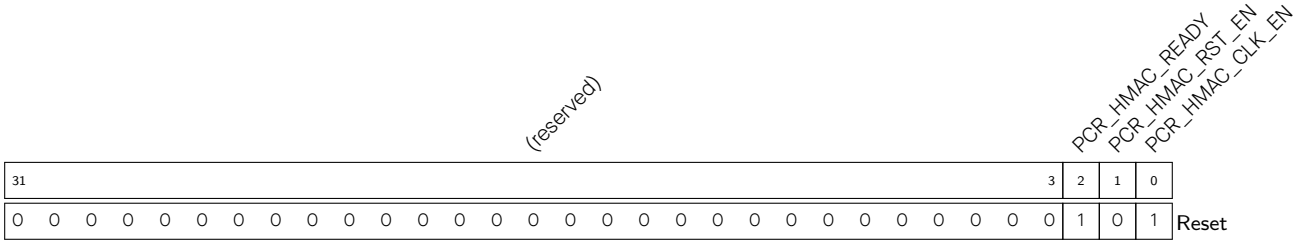
PCR_DS_READY 表示 DS 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.55. PCR_HMAC_CONF_REG (0x00E8)



PCR_HMAC_CLK_EN 配置是否使能 HMAC 时钟。

0：不使能

1：使能

(R/W)

PCR_HMAC_RST_EN 配置是否复位 HMAC 模块。

0：不复位

1：复位

(R/W)

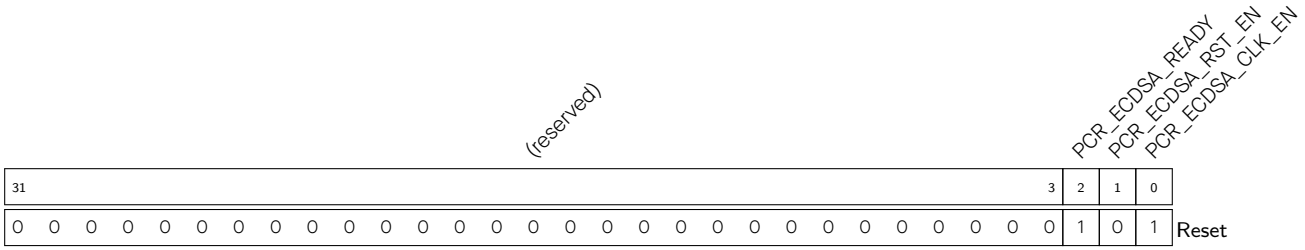
PCR_HMAC_READY 表示 HMAC 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.56. PCR_ECDSA_CONF_REG (0x00EC)



PCR_ECDSA_CLK_EN 配置是否使能 ECDSA_DS 模块时钟。

0：不使能

1：使能

(R/W)

PCR_ECDSA_RST_EN 配置是否复位 ECDSA_DS 模块。

0：不复位

1：复位

(R/W)

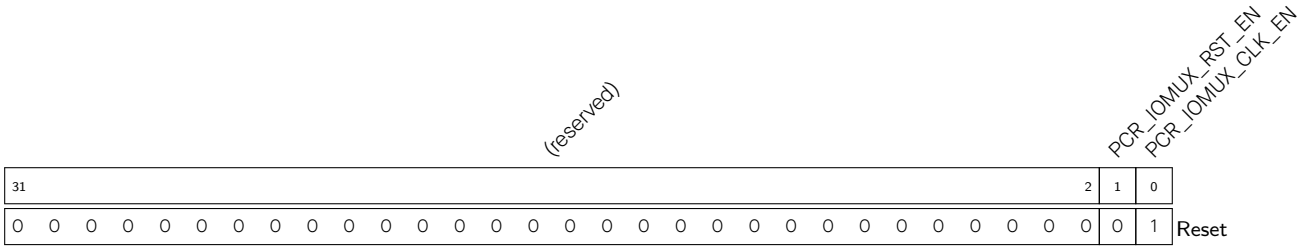
PCR_ECDSA_READY 表示 ECDSA_DS 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.57. PCR_IOMUX_CONF_REG (0x00F0)



PCR_IOMUX_CLK_EN 配置是否使能 IO MUX APB_CLK。

0：不使能

1：使能

(R/W)

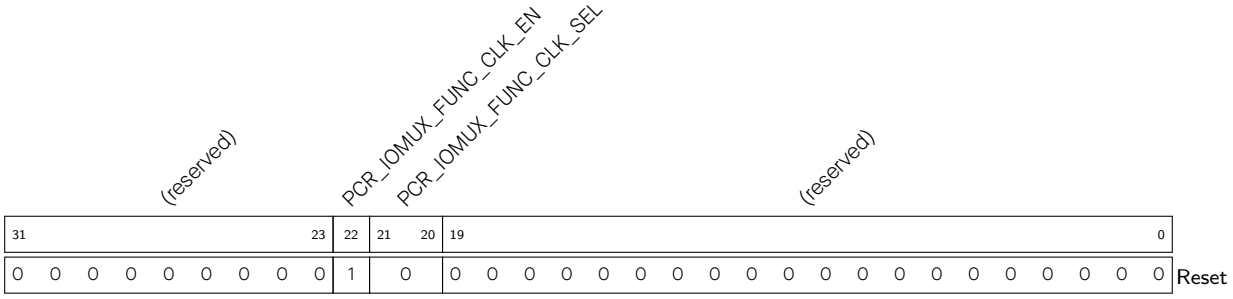
PCR_IOMUX_RST_EN 配置是否复位 IO MUX 模块。

0：不复位

1：复位

(R/W)

Register 9.58. PCR_IOMUX_CLK_CONF_REG (0x00F4)



PCR_IOMUX_FUNC_CLK_SEL 配置选择 IO MUX 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F48M_CLK 时钟

3：不选择任何时钟

(R/W)

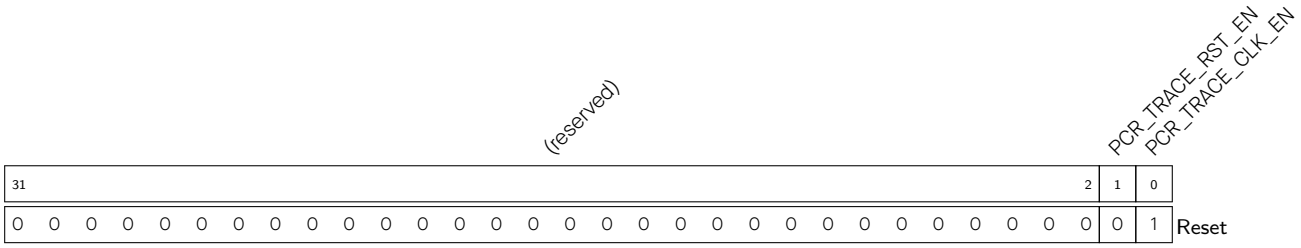
PCR_IOMUX_FUNC_CLK_EN 配置是否使能 IO MUX 功能时钟。

0：不使能

1：使能

(R/W)

Register 9.59. PCR_TRACE_CONF_REG (0x00FC)



PCR_TRACE_CLK_EN 配置是否使能 RISC-V 追踪编码器时钟。

0：不使能

1：使能

(R/W)

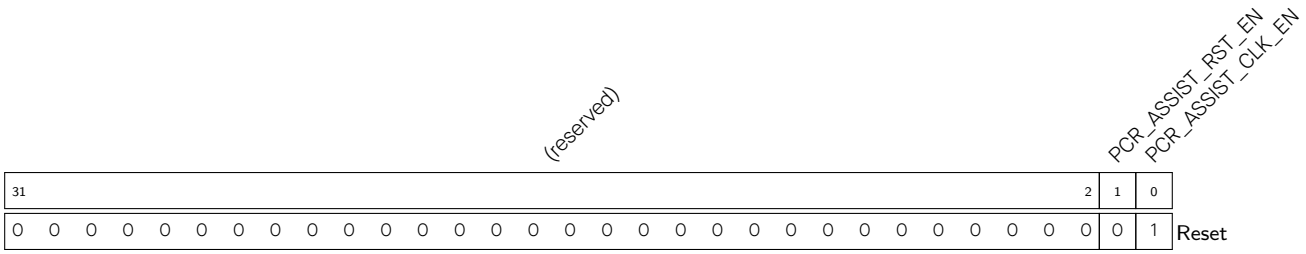
PCR_TRACE_RST_EN 配置是否复位 RISC-V 追踪编码器模块。

0：不复位

1：复位

(R/W)

Register 9.60. PCR_ASSIST_CONF_REG (0x0100)



PCR_ASSIST_CLK_EN 配置是否使能辅助调试模块时钟。

0：不使能

1：使能

(R/W)

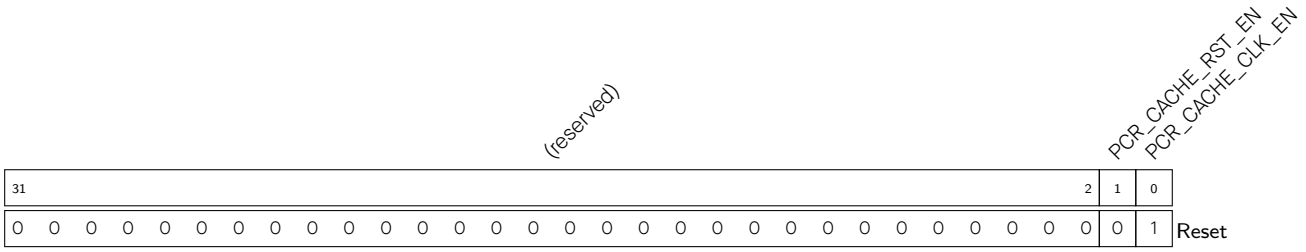
PCR_ASSIST_RST_EN 配置是否复位辅助调试模块。

0：不复位

1：复位

(R/W)

Register 9.61. PCR_CACHE_CONF_REG (0x0104)



PCR_CACHE_CLK_EN 配置是否使能 Cache 时钟。

0：不使能

1：使能

(R/W)

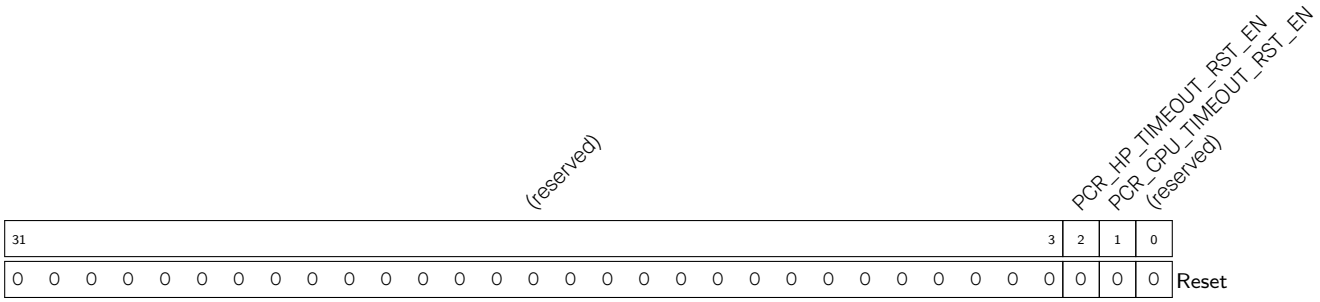
PCR_CACHE_RST_EN 配置是否复位 Cache 模块。

0：不复位

1：复位

(R/W)

Register 9.62. PCR_TIMEOUT_CONF_REG (0x010C)



PCR_CPU_TIMEOUT_RST_EN 配置是否复位 CPU 外设超时保护模块。

0：不复位

1：复位

(R/W)

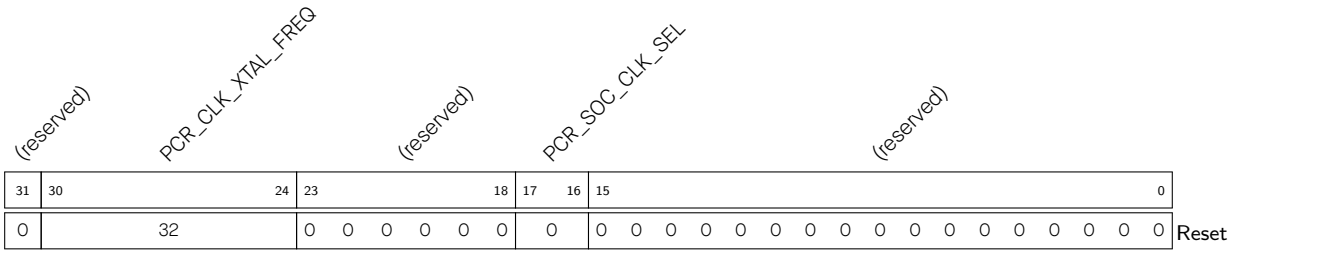
PCR_HP_TIMEOUT_RST_EN 配置是否复位 HP 外设超时保护模块。

0：不复位

1：复位

(R/W)

Register 9.63. PCR_SYSCLK_CONF_REG (0x0110)



PCR_SOC_CLK_SEL 配置选择 HP_ROOT_CLK 时钟源。

0：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F160M_CLK 时钟

3：选择 PLL_F240M_CLK 时钟

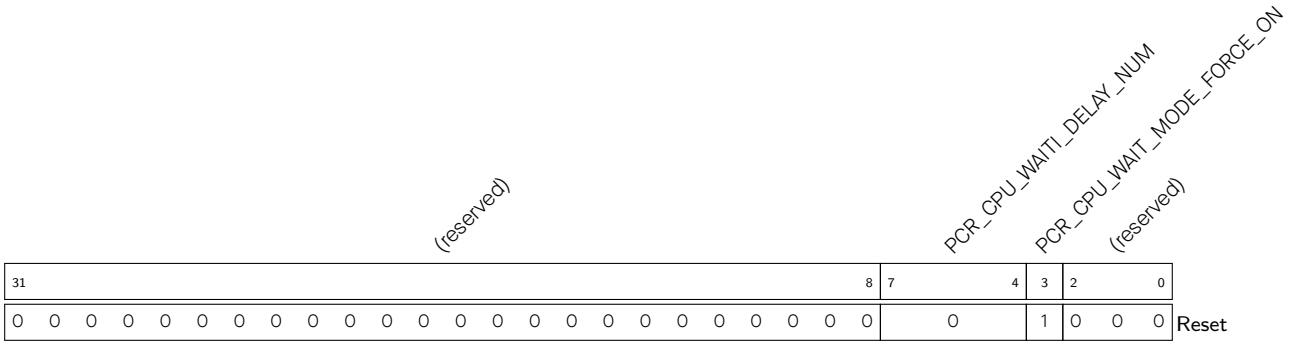
(R/W)

PCR_CLK_XTAL_FREQ 表示 XTAL_CLK 时钟频率。

单位：MHz。

(RO)

Register 9.64. PCR_CPU_WAITI_CONF_REG (0x0114)



PCR_CPU_WAIT_MODE_FORCE_ON 配置是否强制打开 CPU 等待中断模式下的门控时钟。

0：不强制使能

1：强制使能

通常情况下，CPU 执行 WFI (wait for interrupt) 指令后会进入等待中断模式。在此模式下 CPU 的时钟门控一直处于关闭状态，直到中断产生，因此可降低功耗。若此位置 1，CPU 的门控时钟会被强制打开，不受 WFI 指令的影响。

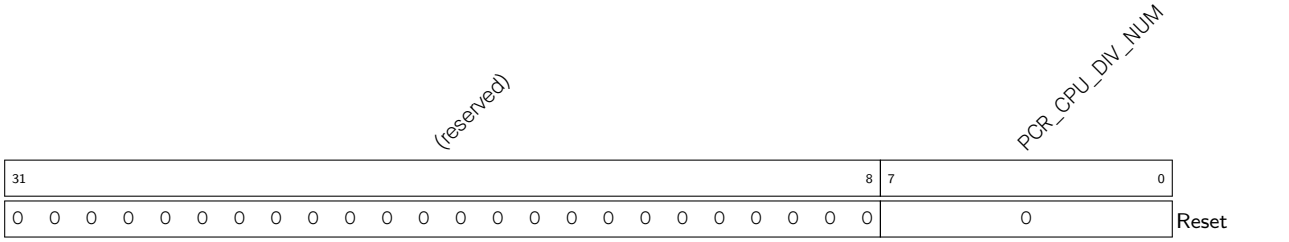
(R/W)

PCR_CPU_WAITI_DELAY_NUM 配置 CPU 在收到 WFI 指令后进入 CPU 等待中断模式后，关闭 CPU 的门控时钟需要的等待周期。

单位：CPU_CLK 周期。

(R/W)

Register 9.65. PCR_CPU_FREQ_CONF_REG (0x0118)

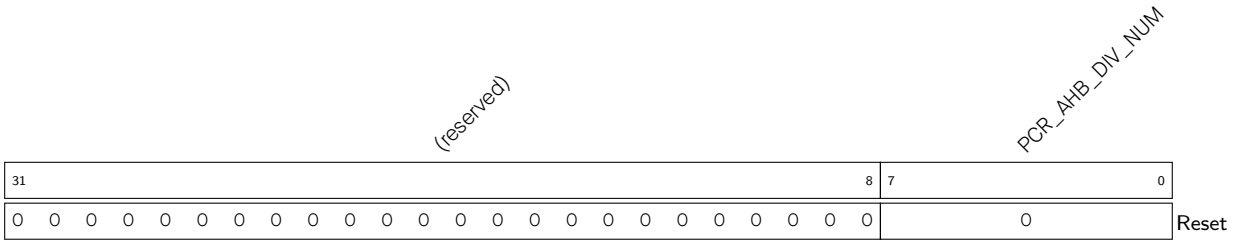


PCR_CPU_DIV_NUM 配置 HP_ROOT_CLK 生成 CPU_CLK 时使用的分频系数。

该字段需要与 PCR_AHB_DIV_NUM 搭配使用。

(R/W)

Register 9.66. PCR_AHB_FREQ_CONF_REG (0x011C)

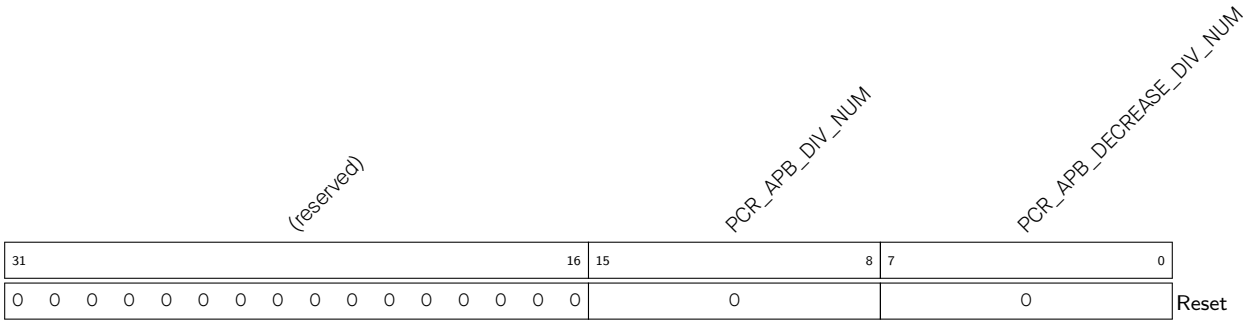


PCR_AHB_DIV_NUM 配置 HP_ROOT_CLK 生成 AHB_CLK 时使用的分频系数。

该字段需要与 PCR_CPU_DIV_NUM 搭配使用。

(R/W)

Register 9.67. PCR_APB_FREQ_CONF_REG (0x0120)



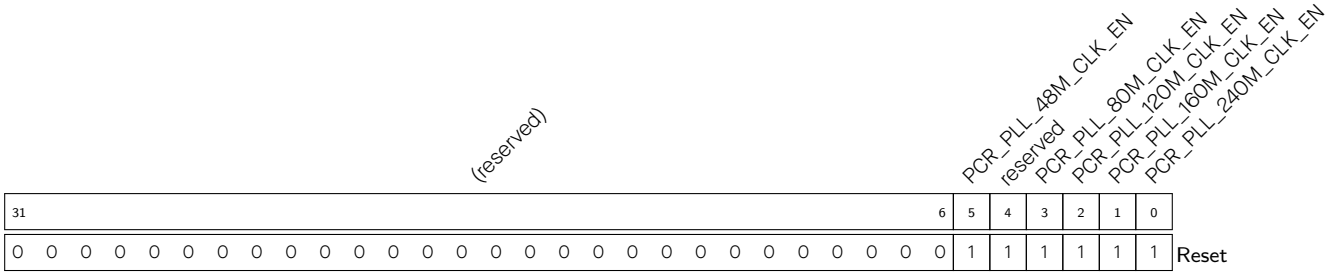
PCR_APB_DECREASE_DIV_NUM 配置 AHB_CLK 生成 APB_CLK 时使用的第二级分频系数。

(R/W)

PCR_APB_DIV_NUM 配置 AHB_CLK 生成 APB_CLK 时使用的第一级分频系数。

(R/W)

Register 9.68. PCR_PLL_DIV_CLK_EN_REG (0x0128)



PCR_PLL_240M_CLK_EN 配置是否使能 PLL_F240M_CLK 时钟。

0：不使能

1（默认）：使能

(R/W)

PCR_PLL_160M_CLK_EN 配置是否使能 PLL_F160M_CLK 时钟。

0：不使能

1（默认）：使能

(R/W)

PCR_PLL_120M_CLK_EN 配置是否使能 PLL_F120M_CLK 时钟。

0：不使能

1（默认）：使能

(R/W)

PCR_PLL_80M_CLK_EN 配置是否使能 PLL_F80M_CLK 时钟。

0：不使能

1（默认）：使能

(R/W)

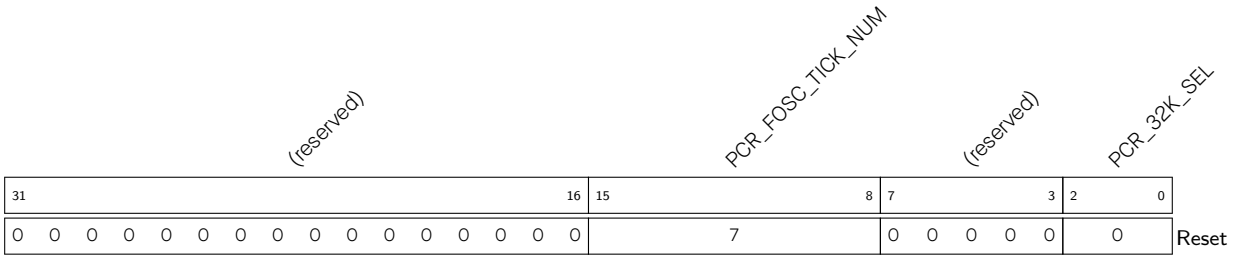
PCR_PLL_48M_CLK_EN 配置是否使能 PLL_F48M_CLK 时钟。

0：不使能

1（默认）：使能

(R/W)

Register 9.69. PCR_CTRL_32K_CONF_REG (0x0130)



PCR_32K_SEL 为 TIMER_GROUP 配置选择一个 32 kHz 时钟。

1：选择 XTAL32K_CLK

2：选择 OSC_SLOW_CLK

3：选择 RC_SLOW_CLK

4：选择 RC_FAST_CLK 经过 PCR_FOSC_TICK_NUM 分频后的时钟

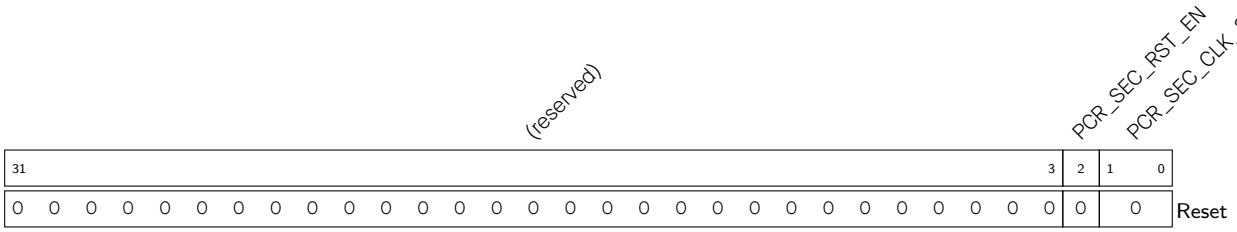
其他值：无效

(R/W)

PCR_FOSC_TICK_NUM 配置待进入校准模块 RC_FAST_CLK 的分频系数。该字段仅当 PCR_32K_SEL 为 4 时有效。

(R/W)

Register 9.70. PCR_SEC_CONF_REG (0x013C)



PCR_SEC_CLK_SEL 配置选择片外存储器加密与解密模块的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F480M_CLK 时钟

3：不选择任何时钟

(R/W)

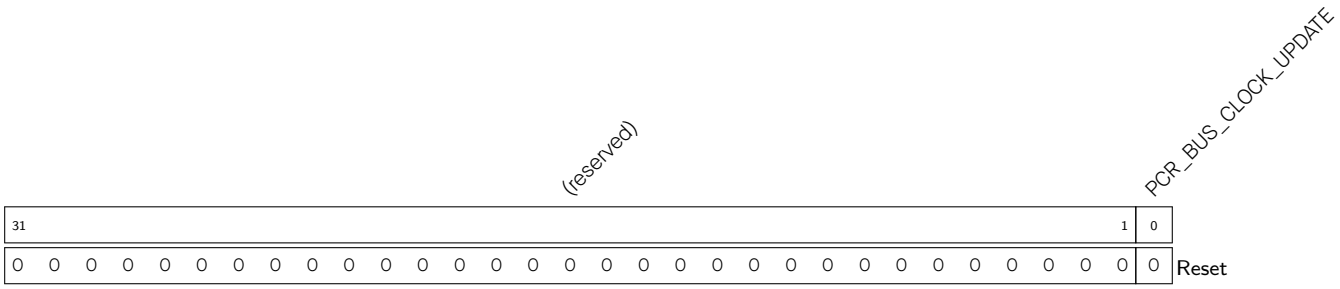
PCR_SEC_RST_EN 配置是否复位 SEC 模块。

0：不复位

1：复位

(R/W)

Register 9.71. PCR_BUS_CLK_UPDATE_REG (0x0144)



PCR_BUS_CLOCK_UPDATE 配置是否更新 CPU_CLK 分频、AHB_CLK 分频、HP_ROOT_CLK 时钟源选择的配置。

0：不更新配置

1：更新配置

此位在更新完成后自动清零。(R/W/WTC)

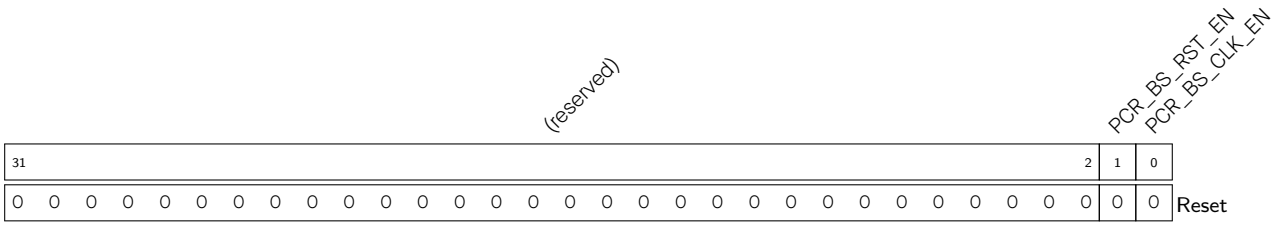
Register 9.72. PCR_SAR_CLK_DIV_REG (0x0148)



PCR_SAR2_CLK_DIV_NUM SAR ADC2 采样逻辑时钟的分频系数，用来产生 ADC 模拟电路控制信号。(R/W)

PCR_SAR1_CLK_DIV_NUM SAR ADC 采样逻辑时钟的分频系数，用来产生 ADC 模拟电路控制信号。(R/W)

Register 9.73. PCR_BS_CONF_REG (0x0150)



PCR_BS_CLK_EN 配置是否使能比特传输控制器模块时钟。

0：不使能

1：使能

(R/W)

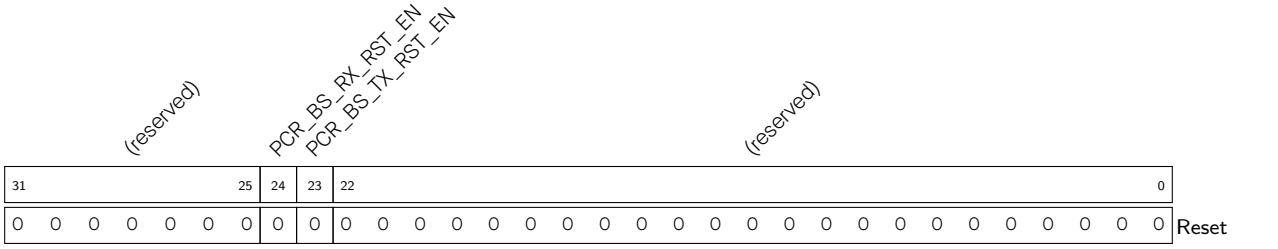
PCR_BS_RST_EN 配置是否复位比特传输控制器模块。

0：不复位

1：复位

(R/W)

Register 9.74. PCR_BS_FUNC_CONF_REG (0x0154)



PCR_BS_TX_RST_EN 配置是否复位比特传输控制器模块的发送侧。

0：不复位

1：复位

(R/W)

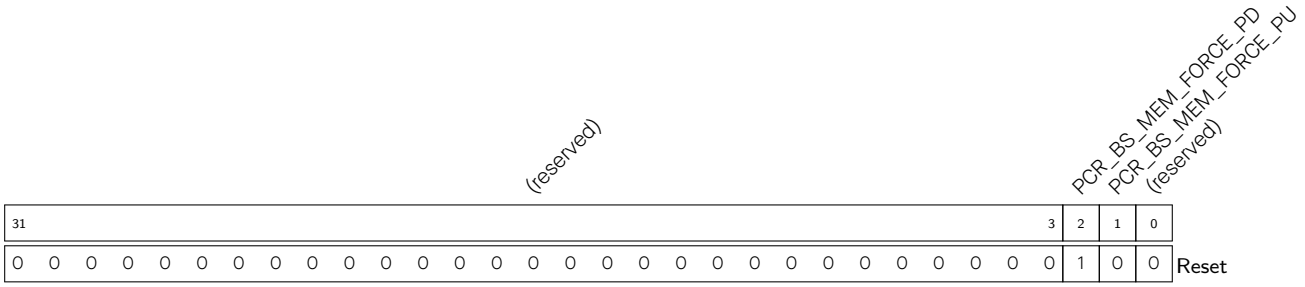
PCR_BS_RX_RST_EN 配置是否复位比特传输控制器模块的接收侧。

0：不复位

1：复位

(R/W)

Register 9.75. PCR_BS_PD_CTRL_REG (0x0158)



PCR_BS_MEM_FORCE_PU 配置是否强制 BS 模块存储器上电。

0：不强制 BS 模块存储器上电

1：强制 BS 模块存储器上电

(R/W)

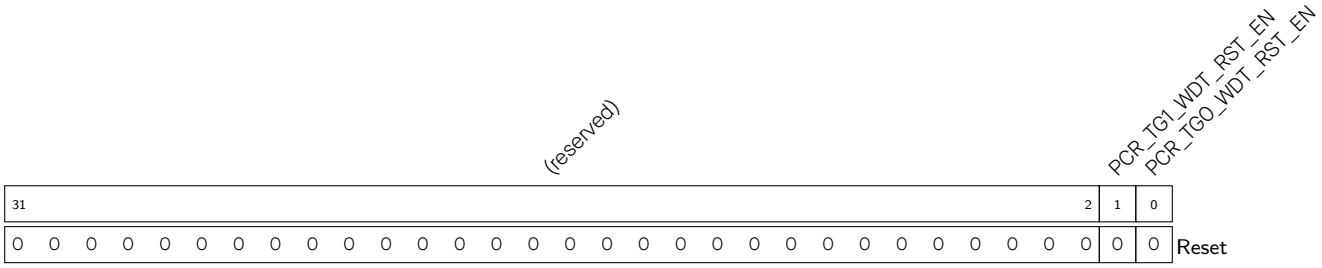
PCR_BS_MEM_FORCE_PD 配置是否强制 BS 模块存储器掉电。

0：不强制 BS 模块存储器掉电

1：强制 BS 模块存储器掉电

(R/W)

Register 9.76. PCR_TIMERGROUP_WDT_CONF_REG (0x015C)



PCR_TG0_WDT_RST_EN 配置是否复位 timergroup0 模块看门狗功能。

0：不复位

1：复位

(R/W)

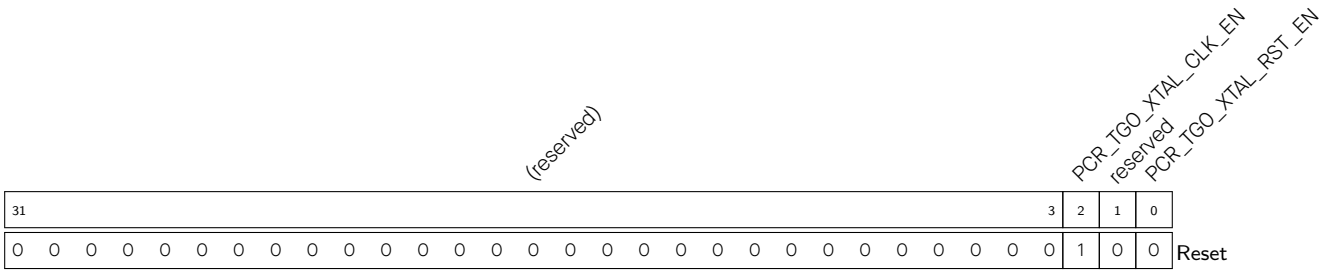
PCR_TG1_WDT_RST_EN 配置是否复位 timergroup1 模块看门狗功能。

0：不复位

1：复位

(R/W)

Register 9.77. PCR_TIMERGROUP_XTAL_CONF_REG (0x0160)



PCR_TG0_XTAL_RST_EN 配置是否复位 timergroup0 模块时钟校准功能。

0：不复位

1：复位

(R/W)

PCR_TG0_XTAL_CLK_EN 配置是否使能 timergroup0 模块时钟校准功能。

0：不使能

1：使能

(R/W)

Register 9.78. PCR_KM_CONF_REG (0x0164)



PCR_KM_CLK_EN 配置是否使能密钥管理器的时钟。

0：不使能

1：使能

(R/W)

PCR_KM_RST_EN 配置是否复位密钥管理器。

0：不复位

1：复位

(R/W)

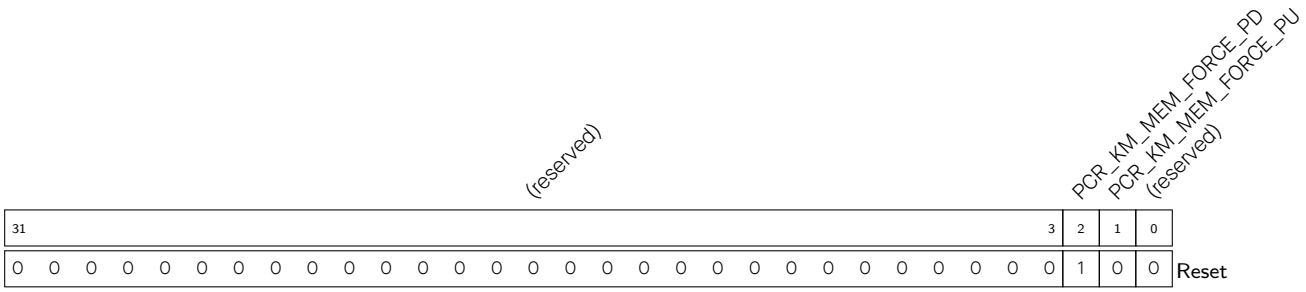
PCR_KM_READY 表示密钥管理器是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.79. PCR_KM_PD_CTRL_REG (0x0168)



PCR_KM_MEM_FORCE_PU 配置是否强制密钥管理器存储器上电。

0：不强制上电

1：强制上电

(R/W)

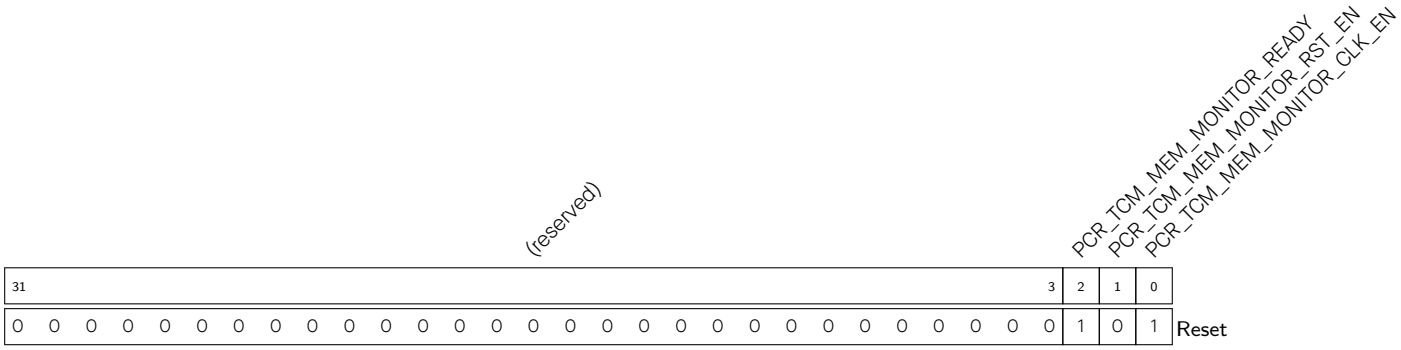
PCR_KM_MEM_FORCE_PD 配置是否强制密钥管理器存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

Register 9.80. PCR_TCM_MEM_MONITOR_CONF_REG (0x016C)



PCR_TCM_MEM_MONITOR_CLK_EN 配置是否使能内存访问监控模块时钟。

0：不使能

1：使能

(R/W)

PCR_TCM_MEM_MONITOR_RST_EN 配置是否复位内存访问监控模块。

0：不复位

1：复位

(R/W)

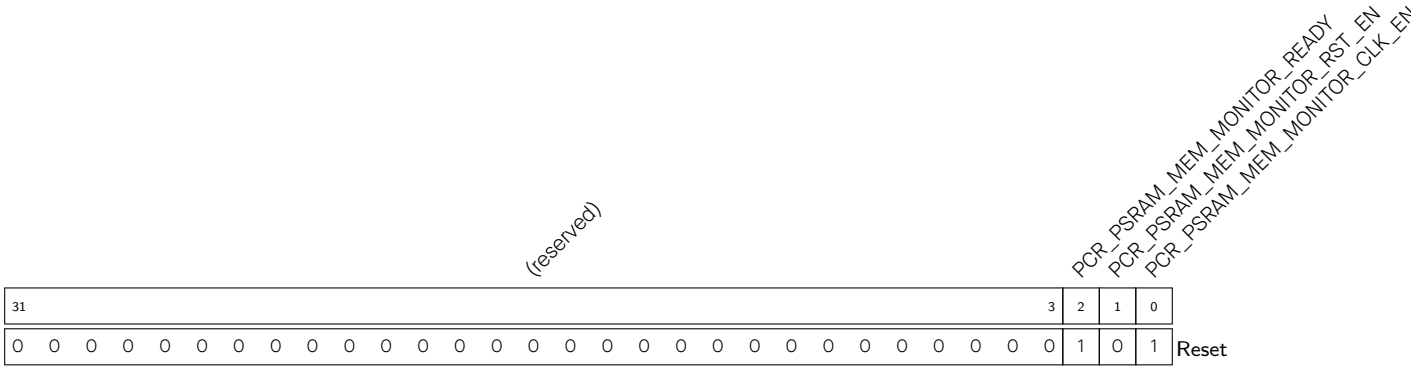
PCR_TCM_MEM_MONITOR_READY 表示内存访问监控模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.81. PCR_PSRAM_MEM_MONITOR_CONF_REG (0x0170)



PCR_PSRAM_MEM_MONITOR_CLK_EN 配置是否使能外存访问监控模块时钟。

0：不使能

1：使能

(R/W)

PCR_PSRAM_MEM_MONITOR_RST_EN 配置是否复位外存访问监控模块。

0：不复位

1：复位

(R/W)

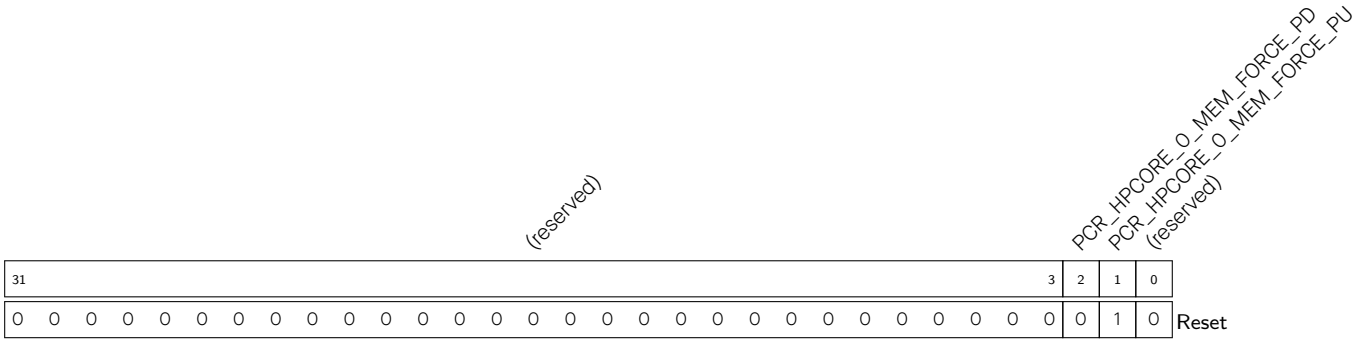
PCR_PSRAM_MEM_MONITOR_READY 表示外存访问监控模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 9.82. PCR_HPCORE_0_PD_CTRL_REG (0x0178)



PCR_HPCORE_0_MEM_FORCE_PU 配置是否强制 HP COREO 存储器上电。

0：不强制上电

1：强制上电

(R/W)

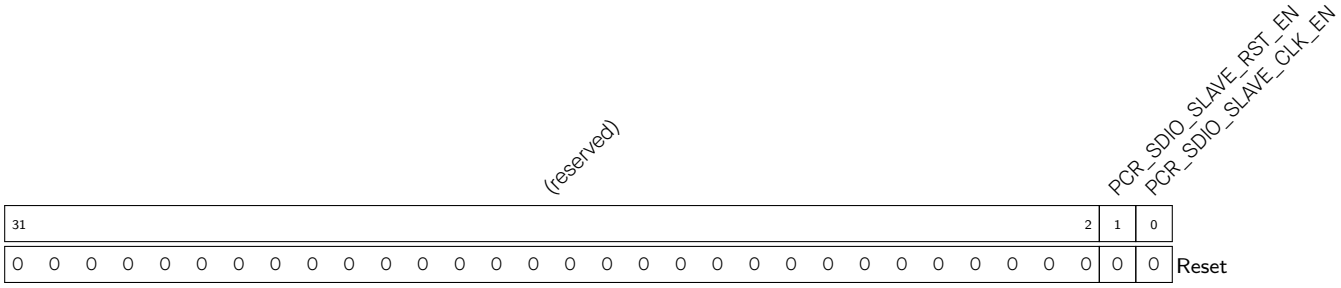
PCR_HPCORE_0_MEM_FORCE_PD 配置是否强制 HP COREO 存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

Register 9.83. PCR_SDIO_SLAVE_CONF_REG (0x017C)



PCR_SDIO_SLAVE_RST_EN 配置是否复位 SDIO slave 模块。

0：不复位

1：复位

(R/W)

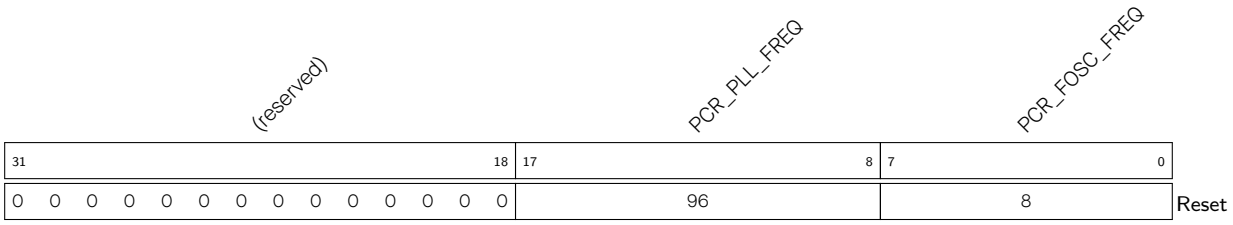
PCR_SDIO_SLAVE_CLK_EN 配置是否使能 SDIO slave 模块时钟。

0：不使能

1：使能

(R/W)

Register 9.84. PCR_SYSCLK_FREQ_QUERY_0_REG (0x0120)



PCR_FOSC_FREQ 表示 RC_FAST_CLK 时钟频率。

单位：MHz

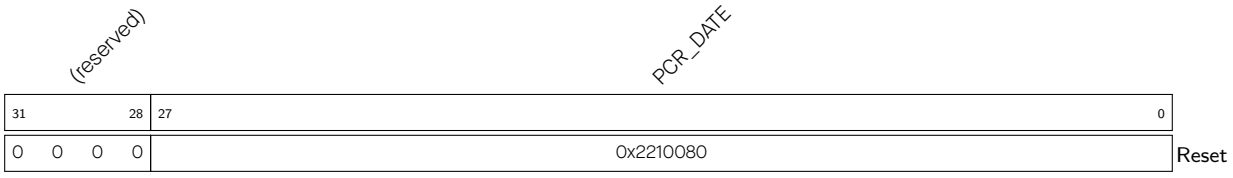
(HRO)

PCR_PLL_FREQ 表示 PLL_F480M_CLK 时钟频率。

单位：MHz

(HRO)

Register 9.85. PCR_DATE_REG (0x0FFC)

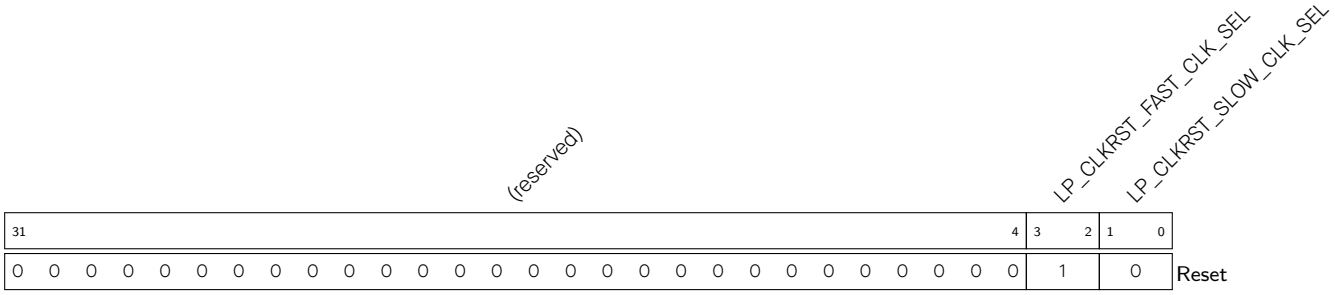


PCR_DATE 版本控制寄存器。(R/W)

9.5.2 低功耗系统时钟复位寄存器

本小节前缀为 LPPERI 的寄存器（最后两个）地址为相对于低功耗外设寄存器（LPPERI）基地址的地址偏移量（相对地址），其余所有地址均为相对于低功耗时钟复位寄存器（LP_CLKRST）基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 9.86. LP_CLKRST_LP_CLK_CONF_REG (0x0000)



LP_CLKRST_SLOW_CLK_SEL 配置 LP_SLOW_CLK 的时钟源。

0：RC_SLOW_CLK

1：XTAL32K_CLK

2：OSC_SLOW_CLK

3：无效值。没有作用

(R/W)

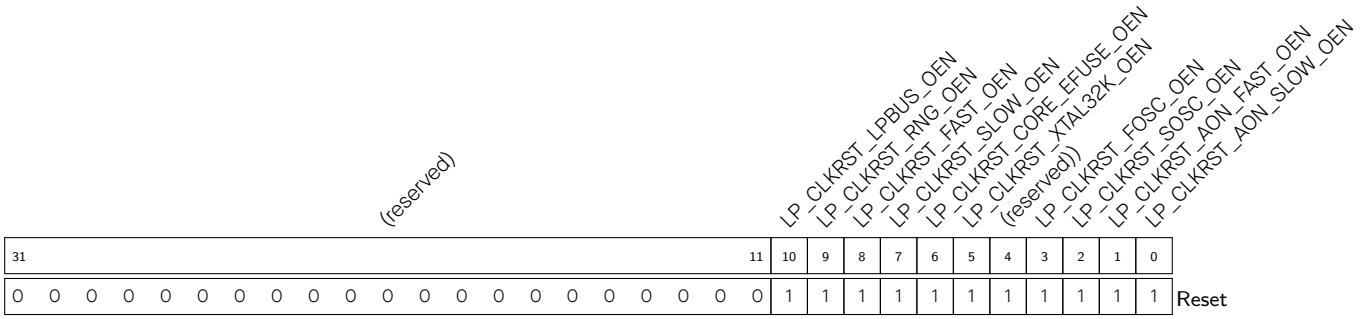
LP_CLKRST_FAST_CLK_SEL 配置 LP_FAST_CLK 的时钟源。

0：RC_FAST_CLK

1：XTAL_D2_CLK

(R/W)

Register 9.87. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)



LP_CLKRST_AON_SLOW_OEN 配置是否使能 LP_DYN_SLOW_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_AON_FAST_OEN 配置是否使能 LP_DYN_FAST_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_SOSC_OEN 配置是否使能 OSC_SLOW_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_FOSC_OEN 配置是否使能 RC_FAST_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_XTAL32K_OEN 配置是否使能 XTAL32K_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

见下页...

Register 9.87. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)

接上页...

LP_CLKRST_CORE_EFUSE_OEN 配置是否使能 EFUSE_CTRL 时钟输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_SLOW_OEN 配置是否使能 LP_SLOW_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_FAST_OEN 配置是否使能 LP_FAST_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_RNG_OEN 配置是否使能 RNG 时钟输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

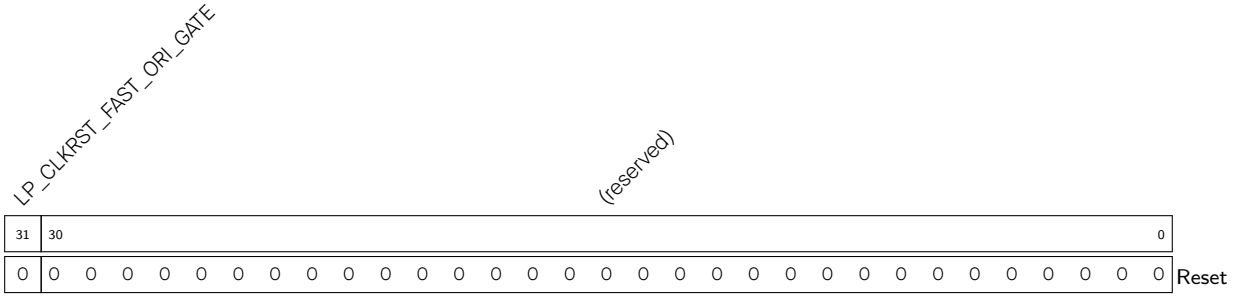
LP_CLKRST_LPBUS_OEN 配置是否使能 LP 总线时钟输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

Register 9.88. LP_CLKRST_LP_CLK_EN_REG (0x0008)



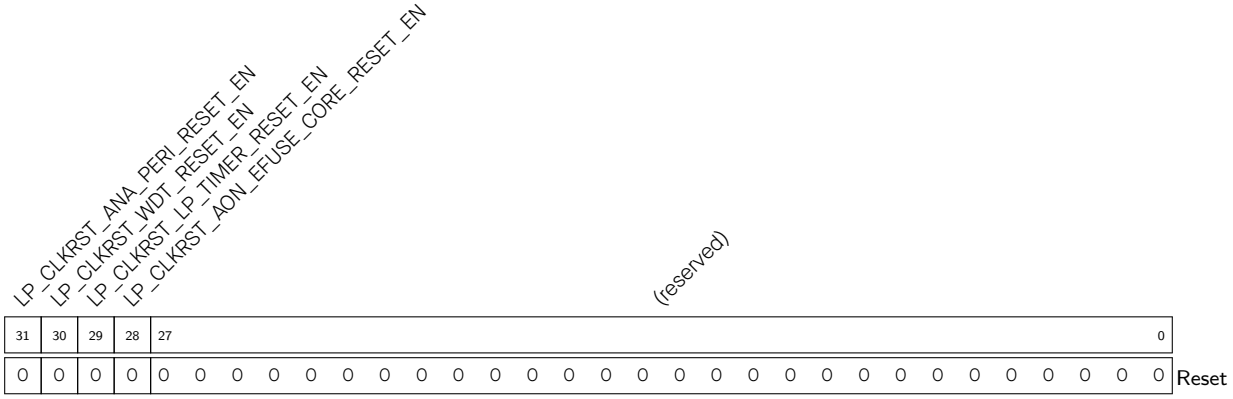
LP_CLKRST_FAST_ORI_GATE 配置 LP_FAST_CLK 的时钟门控。

0：无效。时钟门控由硬件 FSM 控制

1：强制时钟通过门控

(R/W)

Register 9.89. LP_CLKRST_LP_RST_EN_REG (0x000C)



LP_CLKRST_AON_EFUSE_CORE_RESET_EN 配置复位 EFUSE_CTRL 常开部分。

0：无效

1：复位

(R/W)

LP_CLKRST_LP_TIMER_RESET_EN 配置复位 RTC 定时器。

0：无效

1：复位

(R/W)

LP_CLKRST_WDT_RESET_EN 配置复位 RWDT 和超级看门狗定时器。

0：无效

1：复位

(R/W)

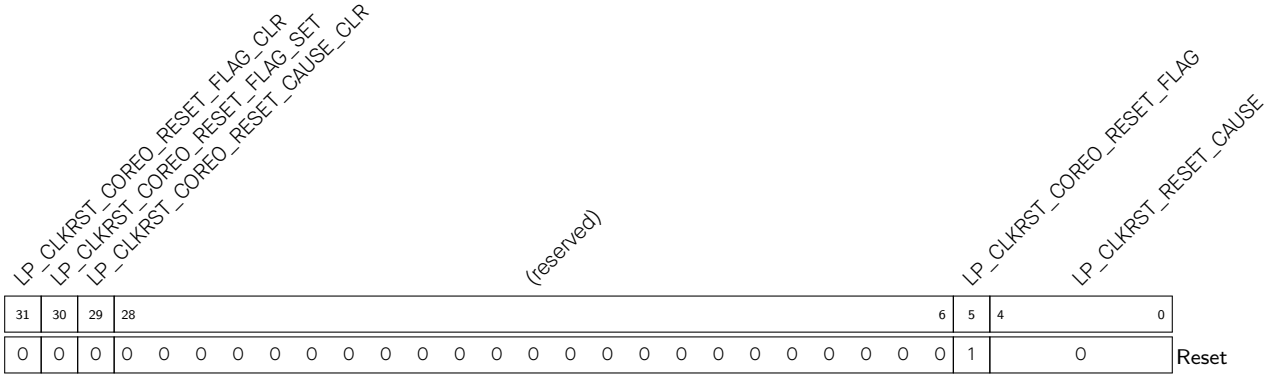
LP_CLKRST_ANA_PERI_RESET_EN 配置复位模拟外设，包括欠压检测器。

0：无效

1：复位

(R/W)

Register 9.90. LP_CLKRST_RESET_CAUSE_REG (0x0010)



LP_CLKRST_RESET_CAUSE 表示复位原因。(RO)

LP_CLKRST_COREO_RESET_FLAG 表示发生的复位是否为 LP_CLKRST_RESET_CAUSE 中记录的复位。

0：非法复位

1：已记录的复位

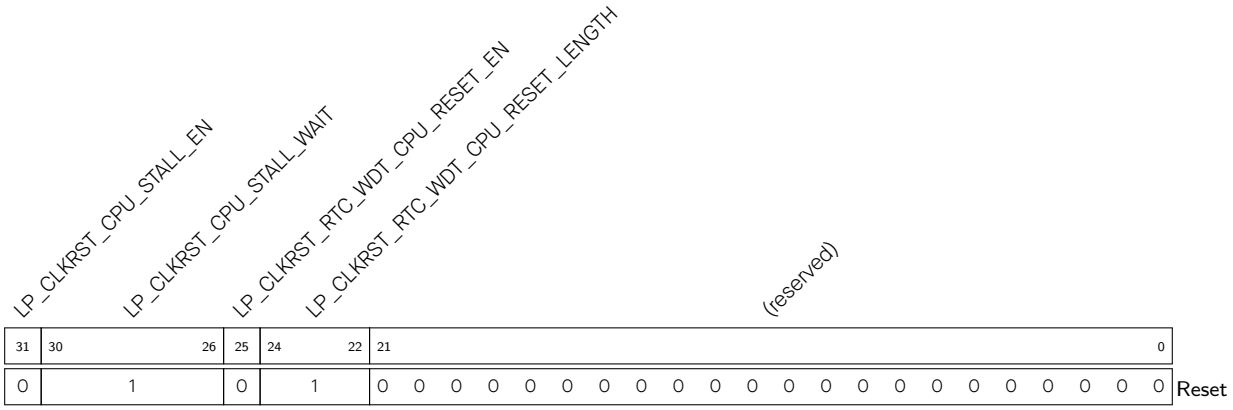
(RO)

LP_CLKRST_COREO_RESET_CAUSE_CLR 写 1 清除 LP_CLKRST_RESET_CAUSE。(WT)

LP_CLKRST_COREO_RESET_FLAG_SET 写 1 置位 LP_CLKRST_COREO_RESET_FLAG。(WT)

LP_CLKRST_COREO_RESET_FLAG_CLR 写 1 清除 LP_CLKRST_COREO_RESET_FLAG。(WT)

Register 9.91. LP_CLKRST_CPU_RESET_REG (0x0014)



LP_CLKRST_RTC_WDT_CPU_RESET_LENGTH 配置 RWDT 复位 CPU 的时间长度。

单位：LP_DYN_FAST_CLK 周期。

(R/W)

LP_CLKRST_RTC_WDT_CPU_RESET_EN 配置 RWDT 是否能复位 CPU。

0：RWDT 超时时无法复位 CPU

1：RWDT 超时时复位 CPU

(R/W)

LP_CLKRST_CPU_STALL_WAIT 配置 CPU 停顿状态 (CPU Stall) 到复位之间的时间间隔。

单位：LP_DYN_FAST_CLK 周期。

(R/W)

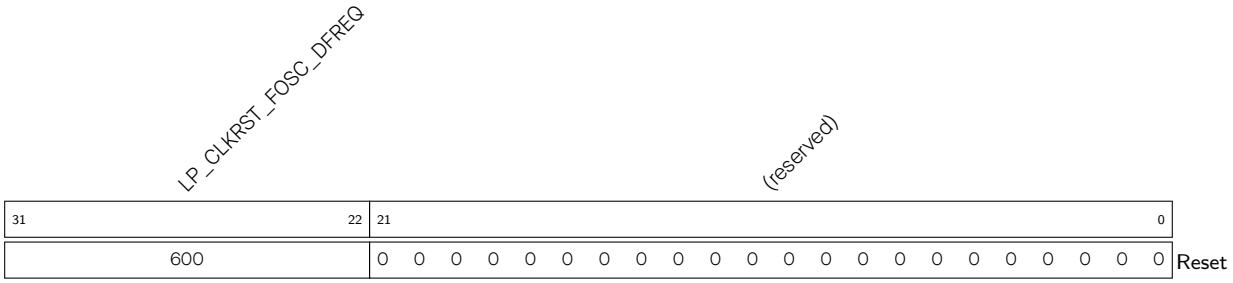
LP_CLKRST_CPU_STALL_EN 配置在 RWDT 和软件复位 CPU 之前是否进入 CPU 停顿状态。

0：否

1：是

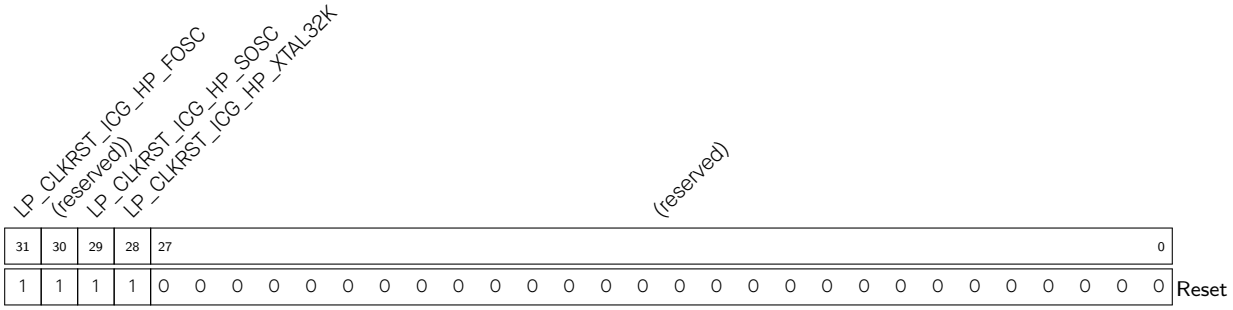
(R/W)

Register 9.92. LP_CLKRST_FOSC_CNTL_REG (0x0018)



LP_CLKRST_FOSC_DFREQ 配置 RC_FAST_CLK 频率，值越大，时钟周期越大。(R/W)

Register 9.93. LP_CLKRST_CLK_TO_HP_REG (0x0020)



LP_CLKRST_ICG_HP_XTAL32K 配置 XTAL32K_CLK 到 HP System 的时钟门控。

0：时钟无法到达 HP System

1：时钟能够到达 HP System

(R/W)

LP_CLKRST_ICG_HP_SOSC 配置 RC_SLOW_CLK 到 HP System 的时钟门控。

0：时钟无法到达 HP System

1：时钟能够到达 HP System

(R/W)

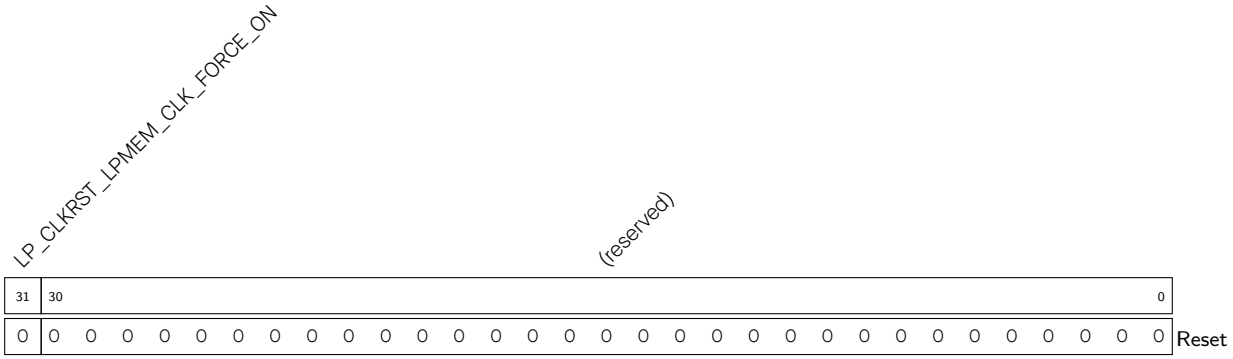
LP_CLKRST_ICG_HP_FOSC 配置 RC_FAST_CLK 到 HP System 的时钟门控。

0：时钟无法到达 HP System

1：时钟能够到达 HP System

(R/W)

Register 9.94. LP_CLKRST_LPMEM_FORCE_REG (0x0024)



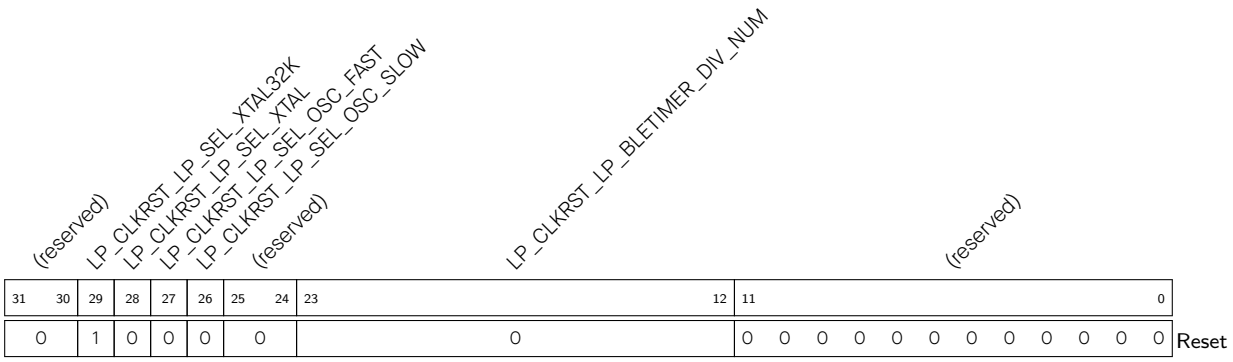
LP_CLKRST_LPMEM_CLK_FORCE_ON 配置强制打开 LP 存储器的时钟门控。

0：无效。时钟门控由硬件 FSM 控制

1：强制打开时钟门控

(R/W)

Register 9.95. LP_CLKRST_LPPERI_REG (0x0028)



LP_CLKRST_LP_BLETIMER_DIV_NUM 配置 RTC BLE 定时器的时钟分频器的分频系数，RTC BLE 定时器的工作时钟频率 = 源时钟频率 / （(LP_CLKRST_LP_BLETIMER_DIV_NUM - 1） / 2）(R/W)

LP_CLKRST_LP_SEL_OSC_SLOW 配置 RTC BLE 定时器的时钟源

1: 选择 RC_SLOW_CLK 为 RTC BLE 定时器的源时钟

0 : 禁止 RC_SLOW_CLK 为 RTC BLE 定时器的源时钟

(R/W)

LP_CLKRST_LP_SEL_OSC_FAST 配置 RTC BLE 定时器的时钟源

1: 选择 RC_FAST_CLK 为 RTC BLE 定时器的源时钟

0: 禁止 RC_FAST_CLK 为 RTC BLE 定时器的源时钟 (R/W)

LP_CLKRST_LP_SEL_XTAL 配置 RTC BLE 定时器的时钟源

1: 选择 XTAL_CLK 为 RTC BLE 定时器的源时钟

0: 禁止 XTAL_CLK 为 RTC BLE 定时器的源时钟

(R/W)

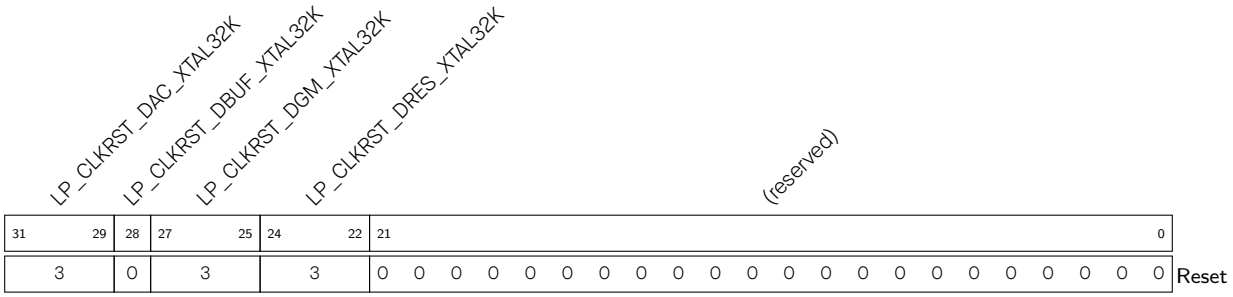
LP_CLKRST_LP_SEL_XTAL32K 配置 RTC BLE 定时器的时钟源

1: 选择 XTAL32K_CLK 为 RTC BLE 定时器的源时钟

0: 禁止 XTAL32K_CLK 为 RTC BLE 定时器的源时钟

(R/W)

Register 9.96. LP_CLKRST_XTAL32K_REG (0x002C)



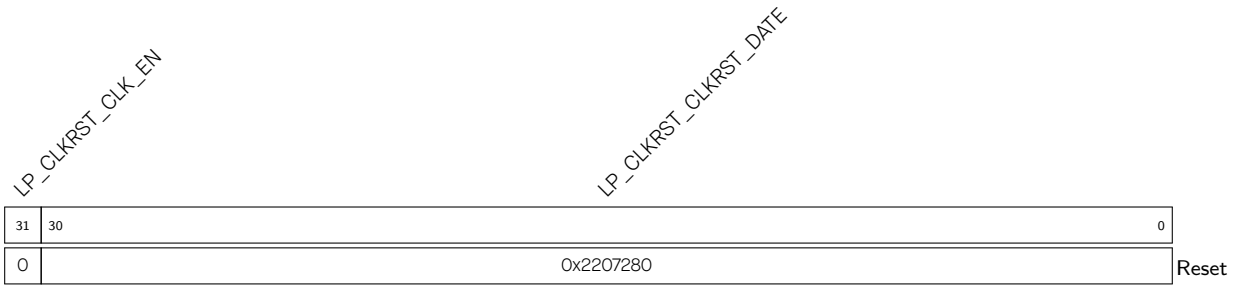
LP_CLKRST_DRES_XTAL32K 配置 DRES。(R/W)

LP_CLKRST_DGM_XTAL32K 配置 DGM。(R/W)

LP_CLKRST_DBUF_XTAL32K 配置 DBUF。(R/W)

LP_CLKRST_DAC_XTAL32K 配置 DAC。(R/W)

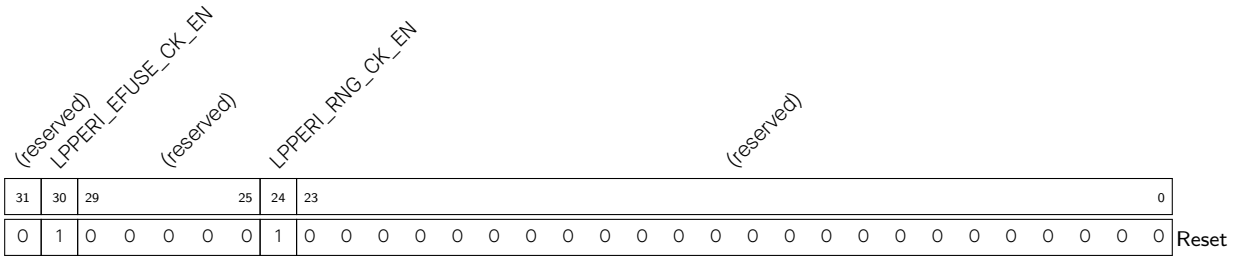
Register 9.97. LP_CLKRST_DATE_REG (0x03FC)



LP_CLKRST_CLKRST_DATE 版本控制寄存器。(R/W)

LP_CLKRST_CLK_EN 配置强制开启寄存器的配置的时钟。0：无效值
1：强制开启
(R/W)

Register 9.98. LPPERI_CLK_EN_REG (0x0000)



LPPERI_RNG_CK_EN 配置是否使能 RNG 模块的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

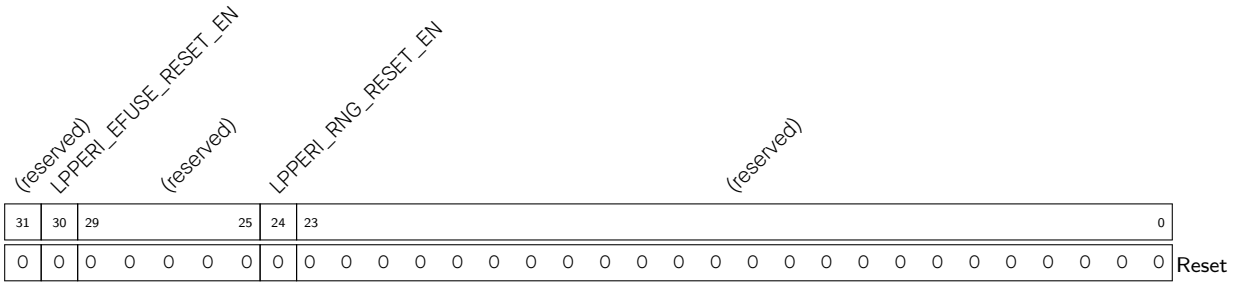
LPPERI_EFUSE_CK_EN 配置是否使能 eFuse 控制器模块的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

Register 9.99. LPPERI_RESET_EN_REG (0x0004)



LPPERI_RNG_RESET_EN 配置 RNG 控制器模块的软件复位。

0：不复位

1：复位

(R/W)

LPPERI_EFUSE_RESET_EN 配置 eFuse 控制器模块的软件复位。

0：不复位

1：复位

(R/W)

第 10 章

芯片 Boot 控制

10.1 概述

芯片在上电或硬件复位时，可以通过 strapping 管脚和 eFuse 参数配置如下启动参数，无需微处理器的参与：

- **芯片启动模式**
  - Strapping 管脚：GPIO26、GPIO27 和 GPIO28
- **SDIO 输入采样沿和输出驱动沿控制**
  - Strapping 管脚：GPIO25 和 MTDI
- **ROM 代码日志打印**
  - Strapping 管脚：GPIO27
  - eFuse 参数：EFUSE_UART_PRINT_CONTROL 和 EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT
- **JTAG 信号源**
  - Strapping 管脚：GPIO7
  - eFuse 参数：EFUSE_DIS_PAD_JTAG、EFUSE_DIS_USB_JTAG 和 EFUSE_JTAG_SEL_ENABLE

上述 eFuse 位的默认值均为 0，也就是说没有烧写过。eFuse 只能烧写一次，一旦烧写为 1，便不能恢复为 0。有关烧写 eFuse 的信息，请参考章节 *7 eFuse 控制器 (EFUSE)*。

在芯片复位过程中（请参考章节 9 复位和时钟），硬件将采样 strapping 管脚电平存储到锁存器中，并一直保持到芯片掉电或下一次芯片复位。strapping 管脚的锁存值可以通过软件从寄存器 GPIO_STRAPPING 中读取。

10.2 功能描述

本小节主要介绍芯片复位时的功能以及控制该功能使用到的 strapping 管脚和 eFuse 组合模式。

> **注意：** 请使用本章节所介绍的组合，其它组合可能会导致不可控结果。

10.2.1 默认配置

上述 strapping 管脚如果没有连接任何电路或连接的电路处于高阻抗状态，则其默认值（即逻辑电平值）取决于管脚内部弱上拉/下拉电阻在复位时的状态。

表 10.2-1. Strapping 管脚默认配置

Strapping 管脚	默认配置	值
GPIO25	浮空	–
GPIO26	浮空	–
GPIO27	上拉	1
GPIO28	上拉	1
GPIO7	浮空	–
MTMS	浮空	–
MTDI	浮空	–

如需改变 strapping 管脚的默认值，用户可以应用外部下拉/上拉电阻，或者应用主机 MCU 的 GPIO 来控制 ESP32-C5 上电复位时的 strapping 管脚电平。复位释放后，strapping 管脚和普通管脚功能相同。

10.2.2 芯片启动模式控制

复位释放后，GPIO26、GPIO28 和 GPIO27 共同决定启动模式，详见表 10.2-2 芯片启动模式控制。

表 10.2-2. 芯片启动模式控制

启动模式	GPIO26	GPIO27	GPIO28
SPI Boot[1]	任意值	任意值	1[1]
Joint Download Boot 0[2]	任意值	1	0
Joint Download Boot 1[3]	0	0	0

[1] **加粗**表示默认值和默认配置。

[2] Joint Download Boot 0 模式下支持以下下载方式：

- USB-Serial-JTAG Download Boot
- UART Download Boot

[3] Joint Download Boot 1 模式下支持以下下载方式：

- UART Download Boot
- SDIO Download Boot

在 SPI Boot 模式下，ROM 引导加载程序通过从 SPI flash 中读取程序来启动系统。

在 Joint Download Boot 0 模式下，用户可通过 UART0、USB 或 SPI Slave 接口将二进制文件下载至 flash，或将二进制文件下载至 SRAM 并运行 SRAM 中的程序。

在 Joint Download Boot 1 模式下，用户可通过 UART0、SDIO 接口将二进制文件下载至 flash，或将二进制文件下载至 SRAM 并运行 SRAM 中的程序。

芯片启动的具体流程见图 10.2-1 芯片启动流程。

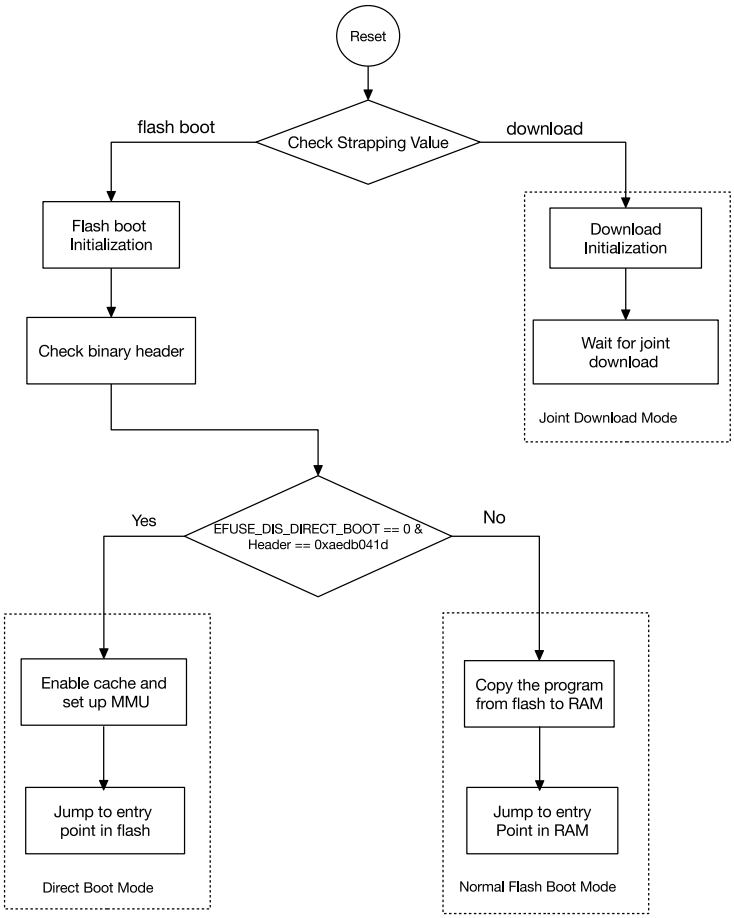


图 10.2-1. 芯片启动流程

下面几个 eFuse 可用于控制启动模式的具体行为：

- EFUSE_DIS_FORCE_DOWNLOAD
  - 如果此 eFuse 设置为 0（默认），软件可通过设置 LP_AON_FORCE_DOWNLOAD_BOOT，触发 CPU 复位，将芯片启动模式强制从 SPI Boot 模式切换至 Joint Download Boot 模式。这种情况下，硬件会将寄存器 GPIO_STRAPPING[4:3] 的值从"1x" 覆盖为"01"；
  - 如果此 eFuse 设置为 1，则禁用 LP_AON_FORCE_DOWNLOAD_BOOT，寄存器 GPIO_STRAPPING 的值不受影响。
- EFUSE_DIS_DOWNLOAD_MODE

  如果此 eFuse 设置为 1，则禁用 Joint Download Boot 模式，寄存器 GPIO_STRAPPING 的值则不受 LP_AON_FORCE_DOWNLOAD_BOOT 的影响。
- EFUSE_ENABLE_SECURITY_DOWNLOAD

  如果此 eFuse 设置为 1，则在 Joint Download Boot 模式下，只允许读取、写入和擦除明文 flash，不支持 SRAM 或寄存器操作。如已禁用 Joint Download Boot 模式，请忽略此 eFuse。
- EFUSE_DIS_DIRECT_BOOT

  如果此 eFuse 设置为 1，则禁用 Direct Boot 模式。

USB 串口/JTAG 控制器可将芯片从 SPI Boot 模式强制切换到 Joint Download Boot 模式，或从 Joint Download Boot 模式强制切换到 SPI Boot 模式。更多信息，请参考章节 *37 USB 串口/JTAG 控制器*。

10.2.3 SDIO 输入采样沿和输出驱动沿控制

GPIO25 和 MTDI 管脚可用于调节 SDIO 输入采样沿和输出驱动沿。详见表 10.2-3 *SDIO 输入采样沿/输出驱动沿控制*。

表 10.2-3. SDIO 输入采样沿/输出驱动沿控制

沿控制	GPIO25	MTDI
下降沿采样下降沿输出	0	0
下降沿采样上升沿输出	0	1
上升沿采样下降沿输出	1	0
上升沿采样上升沿输出	1	1

[1] GPIO25 和 MTDI 默认浮空，以上均非默认配置。

10.2.4 ROM 代码日志打印控制

系统启动过程中，ROM 代码日志可打印至：

- **（默认）UART0 和 USB 串口/JTAG 控制器**
- UART0
- USB 串口/JTAG 控制器

要将 ROM 日志打印至 **UART0** 或 **USB 串口/JTAG 控制器**，详见下文。

EFUSE_UART_PRINT_CONTROL 和 GPIO27 控制 ROM 日志打印至 **UART0**，如表 10.2-4 *UART0 ROM 日志打印控制* 所示。

表 10.2-4. UART0 ROM 日志打印控制

UART0 ROM 日志打印	Register[2]	eFuse[3]	GPIO27
启动过程中，ROM 代码日志始终打印至 UART0	**0**	**0(0b00)**	x[4]
启动过程中使能打印		1(0b01)	0
启动过程中关闭打印			1
启动过程中关闭打印		2(0b10)	0
启动过程中使能打印			1
启动过程中关闭打印		3(0b11)	x
启动过程中关闭打印	1	x	x

[1] **加粗**表示默认值和默认配置。
[2] 寄存器：LP_AON_STORE4_REG[0]
[3] eFuse：EFUSE_UART_PRINT_CONTROL
[4] x：x 表示该值被忽略，任何取值不影响该状态。

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 控制 **USB 串口/JTAG 控制器** ROM 日志打印，如表 10.2-5 *USB 串口/JTAG ROM 日志打印控制* 所示。

表 10.2-5. USB 串口/JTAG ROM 日志打印控制

USB 串口/JTAG ROM 日志打印控制	EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT
使能	0
关闭	1
	忽略

[1] **加粗**表示默认值和默认配置。

10.2.5 JTAG 信号源控制

在系统启动早期阶段，GPIO7 可用于控制 JTAG 信号源。该管脚没有内部上下拉电阻，strapping 的值必须由不处于高阻抗状态的外部电路控制。

GPIO7 与 EFUSE_DIS_PAD_JTAG、EFUSE_DIS_USB_JTAG 和 EFUSE_JTAG_SEL_ENABLE 共同控制 JTAG 信号源，见表 10.2-6 *JTAG 信号源控制*。

表 10.2-6. JTAG 信号源控制

JTAG 信号源	eFuse 1[2]	eFuse 2[3]	eFuse 3[4]	GPIO7
USB 串口/JTAG 控制器[6]	**0**	**0**	0	x[5]
			1	1
JTAG 管脚 MTDI、MTCK、MTMS 和 MTDO				0
		x	x	x
		1		
USB 串口/JTAG 控制器[6]	1	0		
JTAG 关闭		x		
		1		

[1] **加粗**表示默认值和默认配置。

[2] eFuse 1：EFUSE_DIS_PAD_JTAG

[3] eFuse 2：EFUSE_DIS_USB_JTAG

[4] eFuse 3：EFUSE_JTAG_SEL_ENABLE

[5] x：x 表示该值被忽略，任何取值不影响该状态。

[6] 在 Joint Download Boot 1 模式下，USB 串口/JTAG 控制器被强制关闭，JTAG 信号源只能来自 JTAG 管脚。如果 PAD_JTAG 也被禁用，则 JTAG 功能关闭。

第 11 章

中断矩阵

11.1 概述

ESP32-C5 中断矩阵将任一或多个外部中断源映射到 ESP-RISC-V CPU 的任一外部中断上。

ESP32-C5 有 84 个外部中断源，但 CPU 只支持 32 个外部中断。因此，将这些外部中断源映射至 CPU 中断必须使用中断矩阵。

说明：
本章节只涉及将外部中断源映射到 CPU 中断，关于中断配置、中断向量表、中断服务程序推荐处理机制请参考章节 2 高性能处理器 > 2.8中断控制器。

11.2 中断术语

下列有关中断的术语是在《ESP32-C5 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

11.2.1 中断

中断指的是在特定事件或条件下，CPU 中止当前执行去处理优先级更高的任务。中断机制使 CPU 能够迅速响应特定的事件。

《ESP32-C5 技术参考手册》里涉及的“中断”是个广义的概念，可以泛指中断信号或中断源。

11.2.2 中断信号/中断源

中断信号和中断源只是定义的角度不同，实际意思相同。

从外设的角度来看，中断信号是由外设的内部中断源产生的，并发送到中断矩阵。

从中断矩阵的角度来看，它接收来自外设发送的中断信号，并将其视为中断源。然后，中断矩阵将 CPU 外部中断信号发送到 CPU。

从 CPU 的角度来看，来自中断矩阵的中断信号又变成中断源，与核心本地中断源一起发送给 CPU 中断控制器。

11.2.3 ESP32-C5 中断流

图 11.2-1 所示为 ESP32-C5 的中断流。

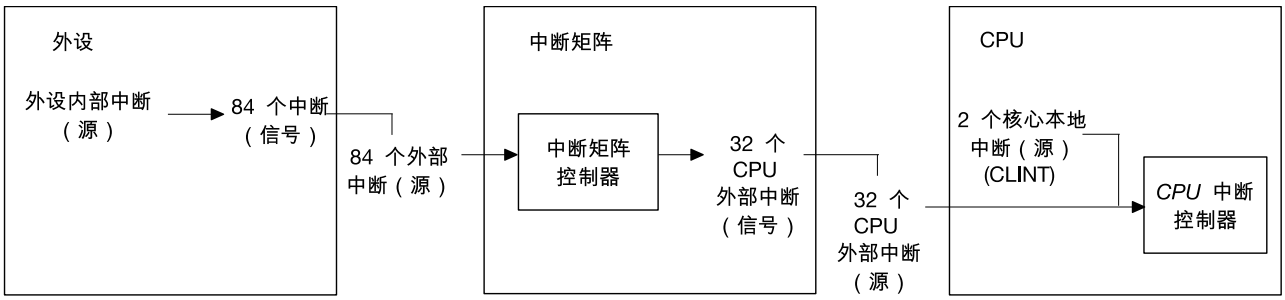


图 11.2-1. ESP32-C5 的中断流

11.3 特性

ESP32-C5 的中断矩阵模块具有如下特性：

- 接收 84 个外部中断源作为输入
- 生成 32 个 HP CPU 的外部中断作为输出
- 支持查询外部中断源当前的中断状态
- 支持将多个中断源映射到单个 HP CPU 中断（即共享中断）
- 支持 CPU 用户模式中断委托到机器模式中断

11.4 结构概览

中断矩阵的结构如图 11.4-1 所示。

用户通过 中断矩阵寄存器 配置外部中断源和 HP CPU 中断的映射关系。图 11.4-1 中的中断矩阵控制器负责映射，并将每个外部中断源的中断状态给到中断矩阵寄存器里的中断状态寄存器。

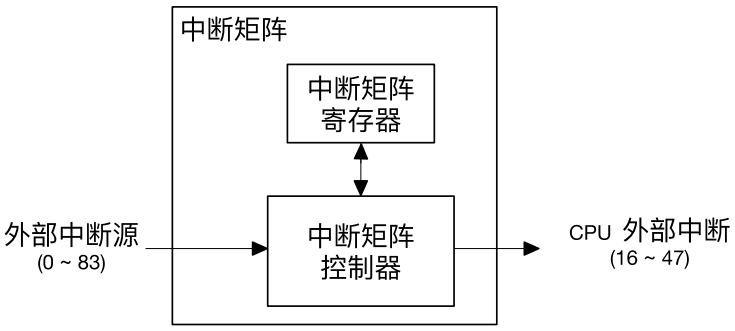


图 11.4-1. 中断矩阵结构图

11.5 功能描述

11.5.1 外部中断源

ESP32-C5 共有 84 个外部中断源。表 11.5-1 列出了所有外部中断源及其对应的中断源映射寄存器与中断状态寄存器。

- “No.”：表示外部中断源序号，范围：0 ~ 83
- “章节”：详细描述外部中断源的章节
- “中断源”：外部中断源名称
- “中断源映射寄存器”：用于将外部中断源分配至 HP CPU 外部中断
- “中断状态寄存器”：用于读取中断源的中断状态
  - “中断状态寄存器 - 位”：表示在中断状态寄存器中的比特位置，用于记录相应中断源的状态
  - “中断状态寄存器 - 名称”：表示中断状态寄存器的名称

表 11.5-1. CPU 外部中断源映射寄存器、外部中断状态寄存器、外部中断源

No.	章节	中断源	中断源映射寄存器	中断状态寄存器 位	中断状态寄存器 名称
0	n/a	保留	保留	0	
1	n/a	保留	保留	1	
2	n/a	保留	保留	2	
3	n/a	保留	保留	3	
4	n/a	保留	保留	4	
5	n/a	保留	保留	5	
6	n/a	保留	保留	6	
7	n/a	保留	保留	7	
8	n/a	保留	保留	9	
9	n/a	保留	保留	9	
10	n/a	保留	保留	10	
11	n/a	保留	保留	11	
12	n/a	保留	保留	12	
13	低功耗管理	PMU_INTR	INTMTX_COREO_PMU_INTR_MAP_REG	13	
14	*eFuse 控制器 (EFUSE)*	EFUSE_INTR	INTMTX_COREO_EFUSE_INTR_MAP_REG	14	
15	低功耗管理	LP_RTC_TIMER_INTR	INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG	15	
16	*UART 控制器 (UART)*	LP_UART_INTR	INTMTX_COREO_LP_UART_INTR_MAP_REG	16	INTMTX_COREO_INT_STATUS_0_REG
17	*I2C 控制器 (I2C)*	LP_I2C_INTR	INTMTX_COREO_LP_I2C_INTR_MAP_REG	17	
18	低功耗管理	LP_WDT_INTR	INTMTX_COREO_LP_WDT_INTR_MAP_REG	18	
19	系统寄存器	LP_PERI_TIMEOUT_INTR	INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG	19	
20	权限控制 *(PMS)*	LP_APM_M0_INTR	INTMTX_COREO_LP_APM_M0_INTR_MAP_REG	20	
21	权限控制 *(PMS)*	LP_APM_M1_INTR	INTMTX_COREO_LP_APM_M1_INTR_MAP_REG	21	
22	密钥管理器	HUK_INTR	INTMTX_COREO_HUK_INTR_MAP_REG	22	
23	软件中断寄存器	CPU_INTR_FROM_CPU_0	INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG	23	
24	软件中断寄存器	CPU_INTR_FROM_CPU_1	INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG	24	
25	软件中断寄存器	CPU_INTR_FROM_CPU_2	INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG	25	
26	高性能处理器	CPU_INTR_FROM_CPU_3	INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG	26	
27	辅助调试	BUS_MONITOR_INTR	INTMTX_COREO_BUS_MONITOR_INTR_MAP_REG	27	
28	高性能处理器	TRACE_INTR	INTMTX_COREO_TRACE_INTR_MAP_REG	28	
29	n/a	保留	保留	29	
30	系统寄存器	CPU_PERI_TIMEOUT_INTR	INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG	30	
31	*GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*	GPIO_INTR_PRO	INTMTX_COREO_GPIO_INTR_PRO_MAP_REG	31	
32	*GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*	GPIO_INTR_EXT	INTMTX_COREO_GPIO_INTR_EXT_MAP_REG	0	INTMTX_COREO_INT_STATUS_1_REG
33	低功耗管理	PAU_INTR	INTMTX_COREO_PAU_INTR_MAP_REG	1	
34	系统寄存器	HP_PERI_TIMEOUT_INTR	INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG	2	
35	n/a	保留	保留	3	
36	权限控制 *(PMS)*	HP_APM_M0_INTR	INTMTX_COREO_HP_APM_M0_INTR_MAP_REG	4	

No.	章节	中断源	中断源映射寄存器	中断状态寄存器 位	中断状态寄存器 名称
37	权限控制 (PMS)	HP_APM_M1_INTR	INTMTX_COREO_HP_APM_M1_INTR_MAP_REG	5	
38	权限控制 (PMS)	HP_APM_M2_INTR	INTMTX_COREO_HP_APM_M2_INTR_MAP_REG	6	
39	权限控制 (PMS)	HP_APM_M3_INTR	INTMTX_COREO_HP_APM_M3_INTR_MAP_REG	7	
40	权限控制 (PMS)	HP_APM_M4_INTR	INTMTX_COREO_HP_APM_M4_INTR_MAP_REG	8	
41	权限控制 (PMS)	LP_APM0_INTR	INTMTX_COREO_LP_APM0_INTR_MAP_REG	9	
42	权限控制 (PMS)	CPU_APM_M0_INTR	INTMTX_COREO_CPU_APM_M0_INTR_MAP_REG	10	
43	权限控制 (PMS)	CPU_APM_M1_INTR	INTMTX_COREO_CPU_APM_M1_INTR_MAP_REG	11	
44	权限控制 (PMS)	MSPI_INTR	INTMTX_COREO_MSPI_INTR_MAP_REG	12	
45	I2S 控制器 (I2S)	I2S_INTR	INTMTX_COREO_I2S_INTR_MAP_REG	13	
46	UART 控制器 (UART)	UHCI0_INTR	INTMTX_COREO_UHCI0_INTR_MAP_REG	14	
47	UART 控制器 (UART)	UART0_INTR	INTMTX_COREO_UART0_INTR_MAP_REG	15	
48	UART 控制器 (UART)	UART1_INTR	INTMTX_COREO_UART1_INTR_MAP_REG	16	
49	LED PWM 控制器 (LEDC)	LEDC_INTR	INTMTX_COREO_LEDC_INTR_MAP_REG	17	
50	CAN FD	TWAI0_TIMER_INTR	INTMTX_COREO_TWAI0_TIMER_INTR_MAP_REG	18	
51	CAN FD	TWAI0_INTR	INTMTX_COREO_TWAI0_INTR_MAP_REG	19	
52	CAN FD	TWAI1_TIMER_INTR	INTMTX_COREO_TWAI1_TIMER_INTR_MAP_REG	20	
53	CAN FD	TWAI1_INTR	INTMTX_COREO_TWAI1_INTR_MAP_REG	21	
54	USB 串口/JTAG 控制器	USB_SERIAL_JTAG_INTR	INTMTX_COREO_USB_INTR_MAP_REG	22	
55	红外遥控 (RMT)	RMT_INTR	INTMTX_COREO_RMT_INTR_MAP_REG	23	
56	I2C 控制器 (I2C)	I2C_EXT0_INTR	INTMTX_COREO_I2C_EXT0_INTR_MAP_REG	24	
57	定时器组 (TIMG)	TG0_T0_INTR	INTMTX_COREO_TG0_T0_INTR_MAP_REG	25	
58	定时器组 (TIMG)	TG0_WDT_INTR	INTMTX_COREO_TG0_WDT_INTR_MAP_REG	26	
59	定时器组 (TIMG)	TG1_T0_INTR	INTMTX_COREO_TG1_T0_INTR_MAP_REG	27	
60	定时器组 (TIMG)	TG1_WDT_INTR	INTMTX_COREO_TG1_WDT_INTR_MAP_REG	28	
61	系统定时器	SYSTIMER_TARGET0_INTR	INTMTX_COREO_SYSTIMER_TARGET0_INTR_MAP_REG	29	
62	系统定时器	SYSTIMER_TARGET1_INTR	INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG	30	
63	系统定时器	SYSTIMER_TARGET2_INTR	INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG	31	
64	ADC 控制器	APB_ADC_INTR	INTMTX_COREO_APB_ADC_INTR_MAP_REG	0	INTMTX_COREO_INT_STATUS_2_REG
65	电机控制脉宽调制器 (MCPWM)	PWM_INTR	INTMTX_COREO_PWM_INTR_MAP_REG	1	
66	脉冲计数控制器 (PCNT)	PCNT_INTR	INTMTX_COREO_PCNT_INTR_MAP_REG	2	
67	并行 IO 控制器 (PARLIO)	PARL_IO_TX_INTR	INTMTX_COREO_PARL_IO_TX_INTR_MAP_REG	3	
68	并行 IO 控制器 (PARLIO)	PARL_IO_RX_INTR	INTMTX_COREO_PARL_IO_RX_INTR_MAP_REG	4	
69	SDIO 从机控制器 (SDIO)	SLC0_INTR	INTMTX_COREO_SLC0_INTR_MAP_REG	5	
70	SDIO 从机控制器 (SDIO)	SLC1_INTR	INTMTX_COREO_SLC1_INTR_MAP_REG	6	
71	通用 DMA 控制器 (GDMA)	GDMA_IN_CH0_INTR	INTMTX_COREO_DMA_IN_CH0_INTR_MAP_REG	7	
72	通用 DMA 控制器 (GDMA)	GDMA_IN_CH1_INTR	INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG	8	
73	通用 DMA 控制器 (GDMA)	GDMA_IN_CH2_INTR	INTMTX_COREO_DMA_IN_CH2_INTR_MAP_REG	9	
74	通用 DMA 控制器 (GDMA)	GDMA_OUT_CH0_INTR	INTMTX_COREO_DMA_OUT_CH0_INTR_MAP_REG	10	
75	通用 DMA 控制器 (GDMA)	GDMA_OUT_CH1_INTR	INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG	11	

GoBack

No.	章节	中断源	中断源映射寄存器	中断状态寄存器	
				位	名称
76	通用 DMA 控制器 (GDMA)	GDMA_OUT_CH2_INTR	INTMTX_COREO_DMA_OUT_CH2_INTR_MAP_REG	12	
77	SPI 控制器 (SPI)	GPSPI2_INTR	INTMTX_COREO_GPSPI2_INTR_MAP_REG	13	
78	AES 加速器 (AES)	AES_INTR	INTMTX_COREO_AES_INTR_MAP_REG	14	
79	SHA 加速器 (SHA)	SHA_INTR	INTMTX_COREO_SHA_INTR_MAP_REG	15	
80	RSA 加速器 (RSA)	RSA_INTR	INTMTX_COREO_RSA_INTR_MAP_REG	16	
81	ECC 加速器 (ECC)	ECC_INTR	INTMTX_COREO_ECC_INTR_MAP_REG	17	
82	ECDSA 数字签名外设 (ECDSA_DS)	ECDSA_INTR	INTMTX_COREO_ECDSA_INTR_MAP_REG	18	
83	密钥管理器	KM_INTR	INTMTX_COREO_KM_INTR_MAP_REG	19	

GoBack

11.5.2 HP CPU 中断

ESP32-C5 HP CPU 支持 RISC-V Core-Local Interrupt Controller (CLIC) 标准 (v0.9) 中断。包含 2 个内部中断和 32 个外部中断 (中断号 16 ~ 47)，通过中断矩阵将外部中断源映射到 32 个 HP CPU 外部中断。每个中断：

- 优先级可配置，具体编码方式见章节 2 高性能处理器 > 2.8.2.5 中断级别和优先级编码；
- 可配置为电平触发或者边沿触发；
- 可通过设置中断阈值屏蔽低优先级的中断。

说明：
CPU 中断的具体功能及配置流程见章节 2 高性能处理器 > 2.8中断控制器。CPU 中断的配置寄存器请见本章节的 11.6.2 软件中断寄存器列表。

11.5.3 分配外部中断源至 HP CPU 外部中断

在本小节中，我们将使用以下术语描述中断矩阵相关操作：

- *SOURCE*：代表某个外部中断中断源，详见表 11.5-1。
- INTMTX_COREO_*SOURCE*_INTR_MAP_REG：*SOURCE* 的中断源映射寄存器。
- Num_P：HP CPU 外部中断编号，范围：16 ~ 47。
- Interrupt_P：表示中断号为 Num_P 的 HP CPU 中断。

11.5.3.1 分配一个外部中断源 SOURCE 至 HP CPU 外部中断

将 *SOURCE* 对应的寄存器 INTMTX_COREO_*SOURCE*_INTR_MAP_REG 配成 Num_P，即可将该中断源分配至序号为 Num_P 的 CPU 中断 (Interrupt_P)。

11.5.3.2 分配多个外部中断源 SOURCE 至 HP CPU 外部中断

将各个中断源对应的寄存器 INTMTX_COREO_*SOURCE*_INTR_MAP_REG 均配置成相同的 Num_P，即可将多个 *SOURCE* 分配至同一 CPU 外部中断 Interrupt_P。上述任一外部中断均会触发 CPU 外部中断 Interrupt_P。待中断触发后，CPU 需查询中断状态寄存器，判断产生中断的外设。更多信息，见章节 2 高性能处理器 > 2.8中断控制器。

11.5.3.3 关闭 HP CPU 外部中断源 SOURCE

将中断源对应的寄存器 INTMTX_COREO_*SOURCE*_INTR_MAP_REG 配置成 0，即可关闭外部中断源。

11.5.4 委托中断

ESP32-C5 RISC-V CPU 支持机器模式和用户模式。32 个 HP CPU 外部中断可配制为机器模式中断或用户模式中断。当 HP CPU 处于机器模式时，只会响应机器模式中断，但不会相应用户模式中断。当 HP CPU 处于用户模式时，可以响应所有中断。

委托中断功能支持 CPU 在机器模式下响应用户模式中断。实现方式为通过寄存器配置，将用户模式中断委托到机器模式中断。

- 每个中断矩阵的中断源都可以独立开启中断委托功能。

- 可同时有多个中断矩阵的中断源开启中断委托功能。
- 只可以委托到一个中断矩阵的中断信号。

软件可以通过配置 INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC_REG 为 n 选择将用户模式中断委托到中断号为 n 的机器模式中断。

软件可配置 INTMTX_COREO_*SOURCE*_INTR_PASS_IN_SEC_REG 为 1 使能中断矩阵中断源 *SOURCE* 的委托中断功能。

11.5.5 查询外部中断源 SOURCE 当前的中断状态

配置开启 *SOURCE* 后，读取寄存器 INTMTX_COREO_INT_STATUS_*n*_REG（只读）中特定位的值可以获取 *SOURCE* 当前的中断状态。寄存器 INTMTX_COREO_INT_STATUS_*n*_REG 与 *SOURCE* 的对应关系如表 11.5-1 所示。

11.6 寄存器列表

11.6.1 中断矩阵寄存器列表

本小节的所有地址均为相对于中断矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
INTMTX_COREO_PMU_INTR_MAP_REG	PMU_INTR 映射寄存器	0x034	R/W
INTMTX_COREO_EFUSE_INTR_MAP_REG	EFUSE_INTR 映射寄存器	0x038	R/W
INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG	LP_RTC_TIMER_INTR 映射寄存器	0x03C	R/W
INTMTX_COREO_LP_UART_INTR_MAP_REG	LP_UART_INTR 映射寄存器	0x040	R/W
INTMTX_COREO_LP_I2C_INTR_MAP_REG	LP_I2C_INTR 映射寄存器	0x044	R/W
INTMTX_COREO_LP_WDT_INTR_MAP_REG	LP_WDT_INTR 映射寄存器	0x048	R/W
INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG	LP_PERI_TIMEOUT_INTR 映射寄存器	0x04C	R/W
INTMTX_COREO_LP_APM_M0_INTR_MAP_REG	LP_APM_M0_INTR 映射寄存器	0x050	R/W
INTMTX_COREO_LP_APM_M1_INTR_MAP_REG	LP_APM_M1_INTR 映射寄存器	0x054	R/W
INTMTX_COREO_HUK_INTR_MAP_REG	HUK_INTR 映射寄存器	0x058	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 映射寄存器	0x05C	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 映射寄存器	0x060	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 映射寄存器	0x064	R/W
INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 映射寄存器	0x068	R/W
INTMTX_COREO_BUS_MONITOR_INTR_MAP_REG	BUS_MONITOR_INTR 映射寄存器	0x06C	R/W
INTMTX_COREO_TRACE_INTR_MAP_REG	TRACE_INTR 映射寄存器	0x070	R/W
INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG	CPU_PERI_TIMEOUT_INTR 映射寄存器	0x078	R/W
INTMTX_COREO_GPIO_INTR_PRO_MAP_REG	GPIO_INTR_PRO 映射寄存器	0x07C	R/W
INTMTX_COREO_GPIO_INTR_EXT_MAP_REG	GPIO_INTR_EXT 映射寄存器	0x080	R/W
INTMTX_COREO_PAU_INTR_MAP_REG	PAU_INTR 映射寄存器	0x084	R/W
INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG	HP_PERI_TIMEOUT_INTR 映射寄存器	0x088	R/W

GoBack

名称	描述	地址	访问
INTMTX_COREO_HP_APM_MO_INTR_MAP_REG	HP_APM_MO_INTR 映射寄存器	0x090	R/W
INTMTX_COREO_HP_APM_M1_INTR_MAP_REG	HP_APM_M1_INTR 映射寄存器	0x094	R/W
INTMTX_COREO_HP_APM_M2_INTR_MAP_REG	HP_APM_M2_INTR 映射寄存器	0x098	R/W
INTMTX_COREO_HP_APM_M3_INTR_MAP_REG	HP_APM_M3_INTR 映射寄存器	0x09C	R/W
INTMTX_COREO_HP_APM_M4_INTR_MAP_REG	HP_APM_M4_INTR 映射寄存器	0x0A0	R/W
INTMTX_COREO_LP_APMO_INTR_MAP_REG	LP_APMO_INTR 映射寄存器	0x0A4	R/W
INTMTX_COREO_CPU_APM_MO_INTR_MAP_REG	CPU_APM_MO_INTR 映射寄存器	0x0A8	R/W
INTMTX_COREO_CPU_APM_M1_INTR_MAP_REG	CPU_APM_M1_INTR 映射寄存器	0x0AC	R/W
INTMTX_COREO_MSPI_INTR_MAP_REG	MSPI_INTR 映射寄存器	0x0B0	R/W
INTMTX_COREO_I2S_INTR_MAP_REG	I2S_INTR 映射寄存器	0x0B4	R/W
INTMTX_COREO_UHCIO_INTR_MAP_REG	UHCIO_INTR 映射寄存器	0x0B8	R/W
INTMTX_COREO_UARTO_INTR_MAP_REG	UARTO_INTR 映射寄存器	0x0BC	R/W
INTMTX_COREO_UART1_INTR_MAP_REG	UART1_INTR 映射寄存器	0x0C0	R/W
INTMTX_COREO_LEDC_INTR_MAP_REG	LEDC_INTR 映射寄存器	0x0C4	R/W
INTMTX_COREO_TWAIO_INTR_MAP_REG	TWAIO_INTR 映射寄存器	0x0C8	R/W
INTMTX_COREO_TWAIO_TIMER_INTR_MAP_REG	TWAIO_TIMER_INTR 映射寄存器	0x0CC	R/W
INTMTX_COREO_TWAI1_INTR_MAP_REG	TWAI1_INTR 映射寄存器	0x0D0	R/W
INTMTX_COREO_TWAI1_TIMER_INTR_MAP_REG	TWAI1_TIMER_INTR 映射寄存器	0x0D4	R/W
INTMTX_COREO_USB_SERIAL_JTAG_INTR_MAP_REG	USB_SERIAL_JTAG_INTR 映射寄存器	0x0D8	R/W
INTMTX_COREO_RMT_INTR_MAP_REG	RMT_INTR 映射寄存器	0x0DC	R/W
INTMTX_COREO_I2C_EXTO_INTR_MAP_REG	I2C_EXTO_INTR 映射寄存器	0x0E0	R/W
INTMTX_COREO_TGO_TO_INTR_MAP_REG	TGO_TO_INTR 映射寄存器	0x0E4	R/W
INTMTX_COREO_TGO_WDT_INTR_MAP_REG	TGO_WDT_INTR 映射寄存器	0x0E8	R/W
INTMTX_COREO_TG1_TO_INTR_MAP_REG	TG1_TO_INTR 映射寄存器	0x0EC	R/W
INTMTX_COREO_TG1_WDT_INTR_MAP_REG	TG1_WDT_INTR 映射寄存器	0x0F0	R/W
INTMTX_COREO_SYSTIMER_TARGETO_INTR_MAP_REG	SYSTIMER_TARGETO_INTR 映射寄存器	0x0F4	R/W
INTMTX_COREO_SYSTIMER_TARGET1_INTR_MAP_REG	SYSTIMER_TARGET1_INTR 映射寄存器	0x0F8	R/W
INTMTX_COREO_SYSTIMER_TARGET2_INTR_MAP_REG	SYSTIMER_TARGET2_INTR 映射寄存器	0x0FC	R/W
INTMTX_COREO_APB_ADC_INTR_MAP_REG	APB_ADC_INTR 映射寄存器	0x100	R/W

GoBack

名称	描述	地址	访问
INTMTX_COREO_PWM_INTR_MAP_REG	PWM_INTR 映射寄存器	0x104	R/W
INTMTX_COREO_PCNT_INTR_MAP_REG	PCNT_INTR 映射寄存器	0x108	R/W
INTMTX_COREO_PARL_IO_TX_INTR_MAP_REG	PARL_IO_TX_INTR 映射寄存器	0x10C	R/W
INTMTX_COREO_PARL_IO_RX_INTR_MAP_REG	PARL_IO_RX_INTR 映射寄存器	0x110	R/W
INTMTX_COREO_SLCO_INTR_MAP_REG	SLCO_INTR 映射寄存器	0x114	R/W
INTMTX_COREO_SLC1_INTR_MAP_REG	SLC1_INTR 映射寄存器	0x118	R/W
INTMTX_COREO_DMA_IN_CHO_INTR_MAP_REG	DMA_IN_CHO_INTR 映射寄存器	0x11C	R/W
INTMTX_COREO_DMA_IN_CH1_INTR_MAP_REG	DMA_IN_CH1_INTR 映射寄存器	0x120	R/W
INTMTX_COREO_DMA_IN_CH2_INTR_MAP_REG	DMA_IN_CH2_INTR 映射寄存器	0x124	R/W
INTMTX_COREO_DMA_OUT_CHO_INTR_MAP_REG	DMA_OUT_CHO_INTR 映射寄存器	0x128	R/W
INTMTX_COREO_DMA_OUT_CH1_INTR_MAP_REG	DMA_OUT_CH1_INTR 映射寄存器	0x12C	R/W
INTMTX_COREO_DMA_OUT_CH2_INTR_MAP_REG	DMA_OUT_CH2_INTR 映射寄存器	0x130	R/W
INTMTX_COREO_GPSPI2_INTR_MAP_REG	GPSPI2_INTR 映射寄存器	0x134	R/W
INTMTX_COREO_AES_INTR_MAP_REG	AES_INTR 映射寄存器	0x138	R/W
INTMTX_COREO_SHA_INTR_MAP_REG	SHA_INTR 映射寄存器	0x13C	R/W
INTMTX_COREO_RSA_INTR_MAP_REG	RSA_INTR 映射寄存器	0x140	R/W
INTMTX_COREO_ECC_INTR_MAP_REG	ECC_INTR 映射寄存器	0x144	R/W
INTMTX_COREO_ECDSA_INTR_MAP_REG	ECDSA_INTR 映射寄存器	0x148	R/W
INTMTX_COREO_KM_INTR_MAP_REG	KM_INTR 映射寄存器	0x14C	R/W
INTMTX_COREO_INT_STATUS_0_REG	中断源 0 ~ 31 的状态寄存器	0x0150	RO
INTMTX_COREO_INT_STATUS_1_REG	中断源 32 ~ 63 的状态寄存器	0x0154	RO
INTMTX_COREO_INT_STATUS_2_REG	中断源 64 ~ 83 的状态寄存器	0x0158	RO
INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_0_REG	中断源 0 ~ 31 的中断委托状态寄存器	0x015C	RO
INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_1_REG	中断源 32 ~ 63 的中断委托状态寄存器	0x0160	RO
INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_2_REG	中断源 64 ~ 83 的中断委托状态寄存器	0x0164	RO
INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC_REG	中断委托配置寄存器	0x0168	R/W
INTMTX_COREO_SECURE_STATUS_REG	中断委托状态寄存器	0x016C	RO
INTMTX_COREO_CLOCK_GATE_REG	时钟门控寄存器	0x0170	R/W
版本寄存器			

GoBack

名称	描述	地址	访问
INTMTX_COREO_INTMTX_DATE_REG	版本控制寄存器	0x07FC	R/W

11.6.2 软件中断寄存器列表

本小节的所有地址均为相对于软件中断基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
INTPRI_CPU_INTR_FROM_CPU_0_REG	CPU_INTR_FROM_CPU_0 映射寄存器	0x0090	R/W
INTPRI_CPU_INTR_FROM_CPU_1_REG	CPU_INTR_FROM_CPU_1 映射寄存器	0x0094	R/W
INTPRI_CPU_INTR_FROM_CPU_2_REG	CPU_INTR_FROM_CPU_2 映射寄存器	0x0098	R/W
INTPRI_CPU_INTR_FROM_CPU_3_REG	CPU_INTR_FROM_CPU_3 映射寄存器	0x009C	R/W
版本寄存器			
INTPRI_DATE_REG	版本控制寄存器	0x00A0	R/W

GoBack

11.7 寄存器

11.7.1 中断矩阵寄存器

本小节的所有地址均为相对于中断矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 11.1. INTMTX_COREO_PMU_INTR_MAP_REG (0x034)

Register 11.2. INTMTX_COREO_EFUSE_INTR_MAP_REG (0x038)

Register 11.3. INTMTX_COREO_LP_RTC_TIMER_INTR_MAP_REG (0x03C)

Register 11.4. INTMTX_COREO_LP_UART_INTR_MAP_REG (0x040)

Register 11.5. INTMTX_COREO_LP_I2C_INTR_MAP_REG (0x044)

Register 11.6. INTMTX_COREO_LP_WDT_INTR_MAP_REG (0x048)

Register 11.7. INTMTX_COREO_LP_PERI_TIMEOUT_INTR_MAP_REG (0x04C)

Register 11.8. INTMTX_COREO_LP_APM_M0_INTR_MAP_REG (0x050)

Register 11.9. INTMTX_COREO_LP_APM_M1_INTR_MAP_REG (0x054)

Register 11.10. INTMTX_COREO_HUK_INTR_MAP_REG (0x058)

Register 11.11. INTMTX_COREO_CPU_INTR_FROM_CPU_0_MAP_REG (0x05C)

Register 11.12. INTMTX_COREO_CPU_INTR_FROM_CPU_1_MAP_REG (0x060)

Register 11.13. INTMTX_COREO_CPU_INTR_FROM_CPU_2_MAP_REG (0x064)

Register 11.14. INTMTX_COREO_CPU_INTR_FROM_CPU_3_MAP_REG (0x068)

Register 11.15. INTMTX_COREO_BUS_MONITOR_INTR_MAP_REG (0x06C)

Register 11.16. INTMTX_COREO_TRACE_INTR_MAP_REG (0x070)

Register 11.17. INTMTX_COREO_CPU_PERI_TIMEOUT_INTR_MAP_REG (0x078)

Register 11.18. INTMTX_COREO_GPIO_INTR_PRO_MAP_REG (0x07C)

Register 11.19. INTMTX_COREO_GPIO_INTR_EXT_MAP_REG (0x080)

Register 11.20. INTMTX_COREO_PAU_INTR_MAP_REG (0x084)

Register 11.21. INTMTX_COREO_HP_PERI_TIMEOUT_INTR_MAP_REG (0x088)

Register 11.22. INTMTX_COREO_HP_APM_M0_INTR_MAP_REG (0x090)

Register 11.23. INTMTX_COREO_HP_APM_M1_INTR_MAP_REG (0x094)

Register 11.24. INTMTX_COREO_HP_APM_M2_INTR_MAP_REG (0x098)

Register 11.25. INTMTX_COREO_HP_APM_M3_INTR_MAP_REG (0x09C)

Register 11.26. INTMTX_COREO_HP_APM_M4_INTR_MAP_REG (0x0A0)

Register 11.27. INTMTX_COREO_LP_APM0_INTR_MAP_REG (0x0A4)

Register 11.28. INTMTX_COREO_CPU_APM_M0_INTR_MAP_REG (0x0A8)

Register 11.29. INTMTX_COREO_CPU_APM_M1_INTR_MAP_REG (0x0AC)

Register 11.30. INTMTX_COREO_MSPI_INTR_MAP_REG (0x0B0)

Register 11.31. INTMTX_CORE0_I2S_INTR_MAP_REG (0x0B4)
Register 11.32. INTMTX_CORE0_UHCI0_INTR_MAP_REG (0x0B8)
Register 11.33. INTMTX_CORE0_UART0_INTR_MAP_REG (0x0BC)
Register 11.34. INTMTX_CORE0_UART1_INTR_MAP_REG (0x0C0)
Register 11.35. INTMTX_CORE0_LEDC_INTR_MAP_REG (0x0C4)
Register 11.36. INTMTX_CORE0_TWAI0_INTR_MAP_REG (0x0C8)
Register 11.37. INTMTX_CORE0_TWAI0_TIMER_INTR_MAP_REG (0x0CC)
Register 11.38. INTMTX_CORE0_TWAI1_INTR_MAP_REG (0x0D0)
Register 11.39. INTMTX_CORE0_TWAI1_TIMER_INTR_MAP_REG (0x0D4)
Register 11.40. INTMTX_CORE0_USB_SERIAL_JTAG_INTR_MAP_REG (0x0D8)
Register 11.41. INTMTX_CORE0_RMT_INTR_MAP_REG (0x0DC)
Register 11.42. INTMTX_CORE0_I2C_EXT0_INTR_MAP_REG (0x0E0)
Register 11.43. INTMTX_CORE0_TG0_T0_INTR_MAP_REG (0x0E4)
Register 11.44. INTMTX_CORE0_TG0_WDT_INTR_MAP_REG (0x0E8)
Register 11.45. INTMTX_CORE0_TG1_T0_INTR_MAP_REG (0x0EC)
Register 11.46. INTMTX_CORE0_TG1_WDT_INTR_MAP_REG (0x0F0)
Register 11.47. INTMTX_CORE0_SYSTIMER_TARGET0_INTR_MAP_REG (0x0F4)
Register 11.48. INTMTX_CORE0_SYSTIMER_TARGET1_INTR_MAP_REG (0x0F8)
Register 11.49. INTMTX_CORE0_SYSTIMER_TARGET2_INTR_MAP_REG (0x0FC)
Register 11.50. INTMTX_CORE0_APB_ADC_INTR_MAP_REG (0x100)
Register 11.51. INTMTX_CORE0_PWM_INTR_MAP_REG (0x104)
Register 11.52. INTMTX_CORE0_PCNT_INTR_MAP_REG (0x108)
Register 11.53. INTMTX_CORE0_PARL_IO_TX_INTR_MAP_REG (0x10C)
Register 11.54. INTMTX_CORE0_PARL_IO_RX_INTR_MAP_REG (0x110)
Register 11.55. INTMTX_CORE0_SLC0_INTR_MAP_REG (0x114)
Register 11.56. INTMTX_CORE0_SLC1_INTR_MAP_REG (0x118)
Register 11.57. INTMTX_CORE0_DMA_IN_CH0_INTR_MAP_REG (0x11C)
Register 11.58. INTMTX_CORE0_DMA_IN_CH1_INTR_MAP_REG (0x120)
Register 11.59. INTMTX_CORE0_DMA_IN_CH2_INTR_MAP_REG (0x124)
Register 11.60. INTMTX_CORE0_DMA_OUT_CH0_INTR_MAP_REG (0x128)
Register 11.61. INTMTX_CORE0_DMA_OUT_CH1_INTR_MAP_REG (0x12C)
Register 11.62. INTMTX_CORE0_DMA_OUT_CH2_INTR_MAP_REG (0x130)
Register 11.63. INTMTX_CORE0_GPSPI2_INTR_MAP_REG (0x134)
Register 11.64. INTMTX_CORE0_AES_INTR_MAP_REG (0x138)
Register 11.65. INTMTX_CORE0_SHA_INTR_MAP_REG (0x13C)

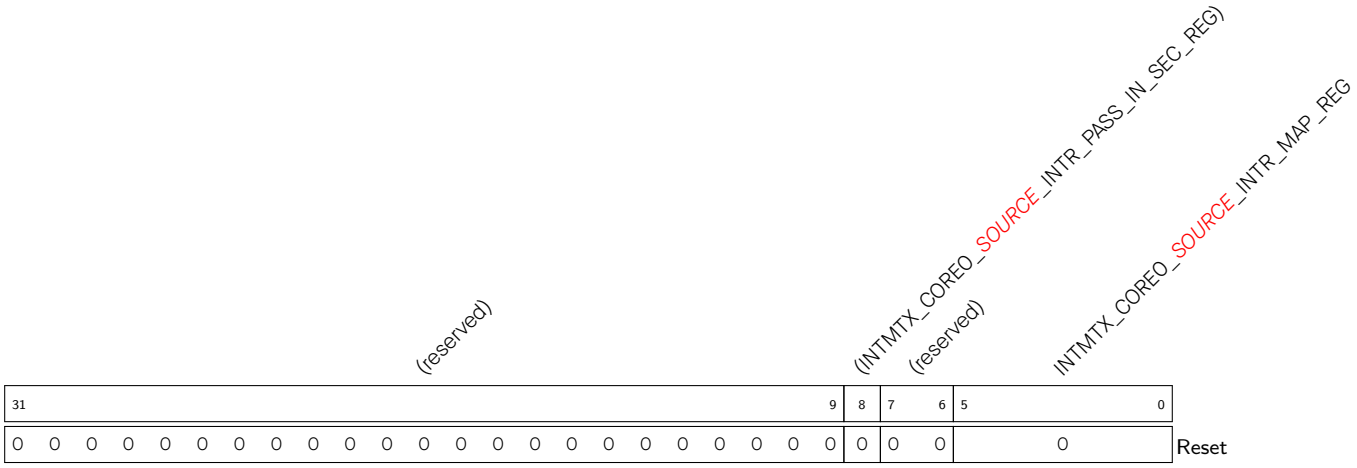
Register 11.66. INTMTX_COREO_RSA_INTR_MAP_REG (0x140)

Register 11.67. INTMTX_COREO_ECC_INTR_MAP_REG (0x144)

Register 11.68. INTMTX_COREO_ECDSA_INTR_MAP_REG (0x148)

Register 11.69. INTMTX_COREO_KM_INTR_MAP_REG (0x14C)

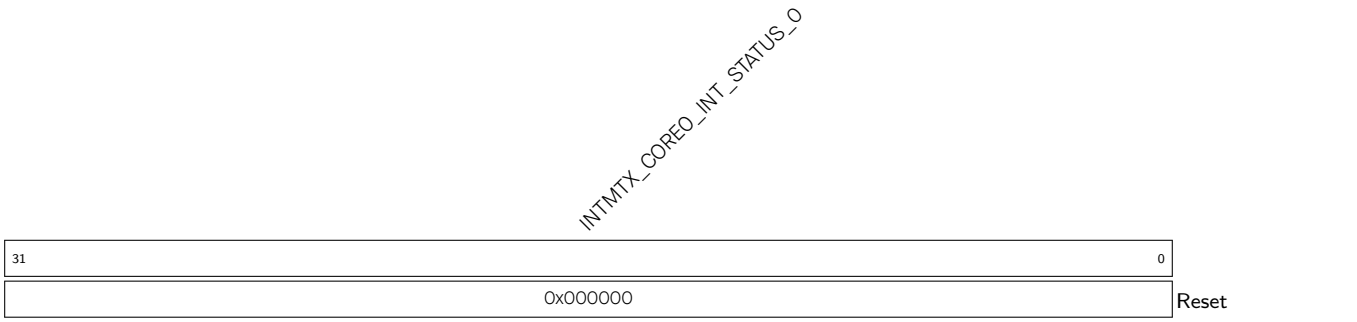
Register 11.70. INTMTX_COREO_*SOURCE*_INTR_MAP_REG (0x0000 - 0x0100)



INTMTX_COREO_*SOURCE*_INTR_MAP 将中断源 (*SOURCE*) 的中断信号映射至 CPU 外部中断。关于 *SOURCE* 的详细信息，请参考表格 11.5-1。(R/W)

INTMTX_COREO_*SOURCE*_INTR_PASS_IN_SEC 将中断源 (*SOURCE*) 的中断信号委托至 CPU 外部机器模式中断。(R/W)

Register 11.71. INTMTX_COREO_INT_STATUS_0_REG (0x0150)



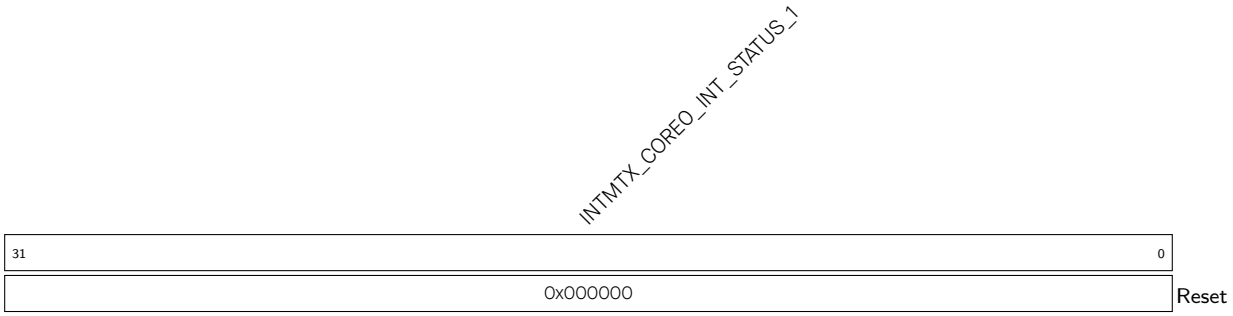
INTMTX_COREO_INT_STATUS_0 中断源 0 ~ 31 的状态寄存器，每一位均代表一个中断源的状态。

0：对应的中断源未触发中断

1：对应的中断源触发了中断

(RO)

Register 11.72. INTMTX_COREO_INT_STATUS_1_REG (0x0154)



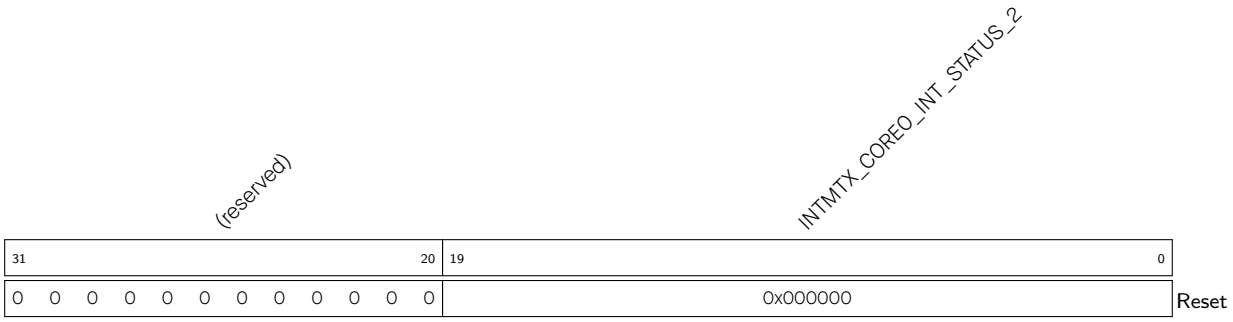
INTMTX_COREO_INT_STATUS_1 中断源 32 ~ 63 的状态寄存器，每一位均代表一个中断源的状态。

0：对应的中断源未触发中断

1：对应的中断源触发了中断

(RO)

Register 11.73. INTMTX_COREO_INT_STATUS_2_REG (0x0158)



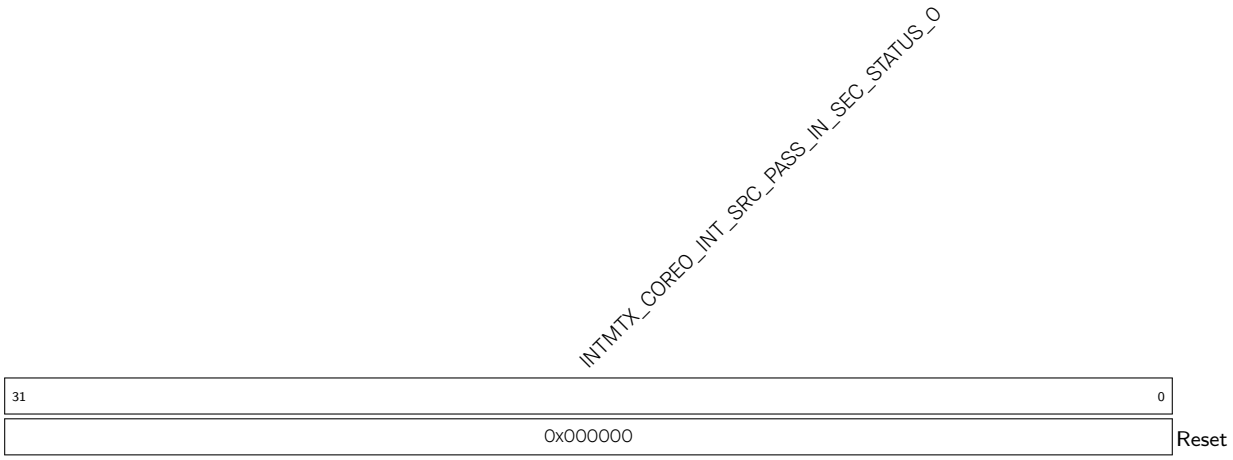
INTMTX_COREO_INT_STATUS_2 中断源 64 ~ 83 的状态寄存器，每一位均代表一个中断源的状态。

0：对应的中断源未触发中断

1：对应的中断源触发了中断

(RO)

Register 11.74. INTMTX_COREO_SRC_PASS_IN_SEC_STATUS_0_REG (0x015C)



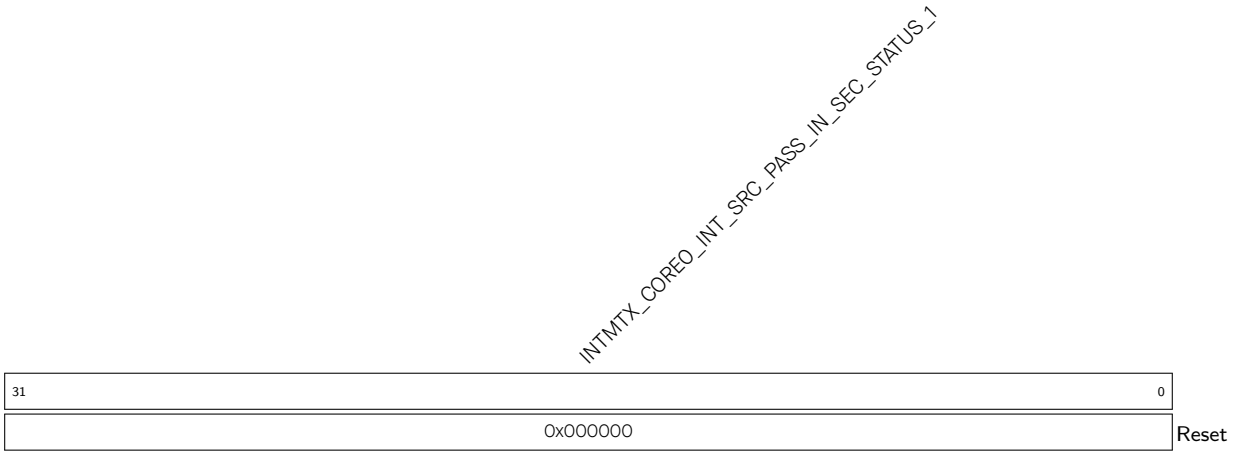
INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_0 表示中断源 0 ~ 31 是否被配置了委托中断，每一位均代表一个中断源配置信息。

0：对应的中断源未配置委托中断

1：对应的中断源已配置委托中断

(RO)

Register 11.75. INTMTX_COREO_SRC_PASS_IN_SEC_STATUS_1_REG (0x0160)



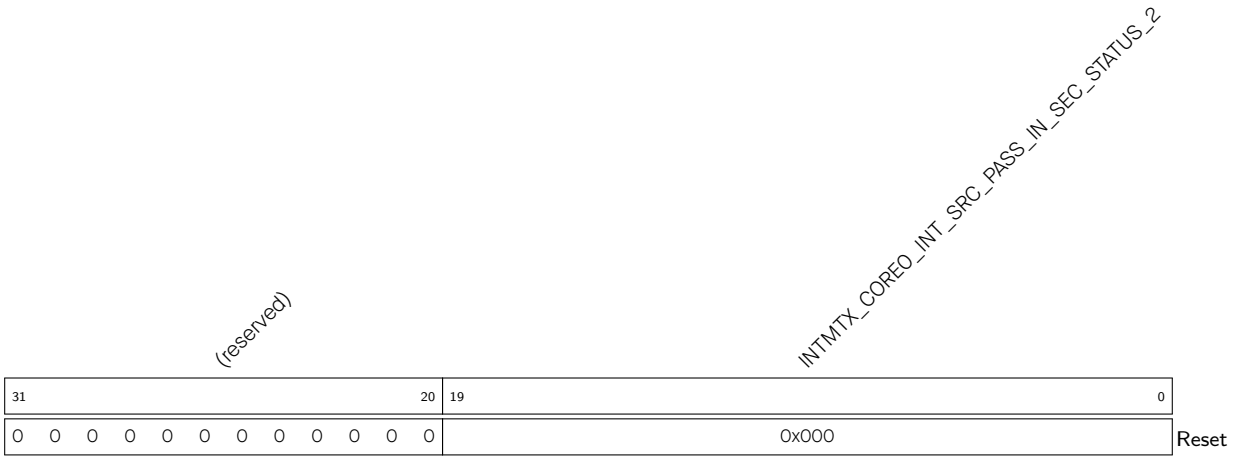
INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_1 表示中断源 32 ~ 63 是否被配置了委托中断，每一位均代表一个中断源配置信息。

0：对应的中断源未配置委托中断

1：对应的中断源已配置委托中断

(RO)

Register 11.76. INTMTX_COREO_SRC_PASS_IN_SEC_STATUS_2_REG (0x0164)



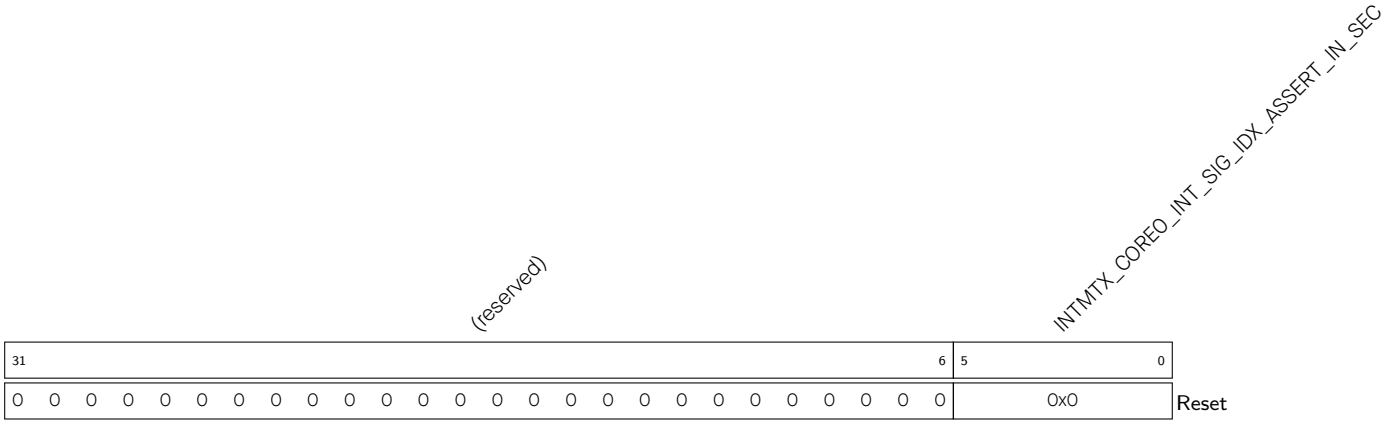
INTMTX_COREO_INT_SRC_PASS_IN_SEC_STATUS_2 表示中断源 64 ~ 83 是否被配置了委托中断，每一位均代表一个中断源配置信息。

0：对应的中断源未配置委托中断

1：对应的中断源已配置委托中断

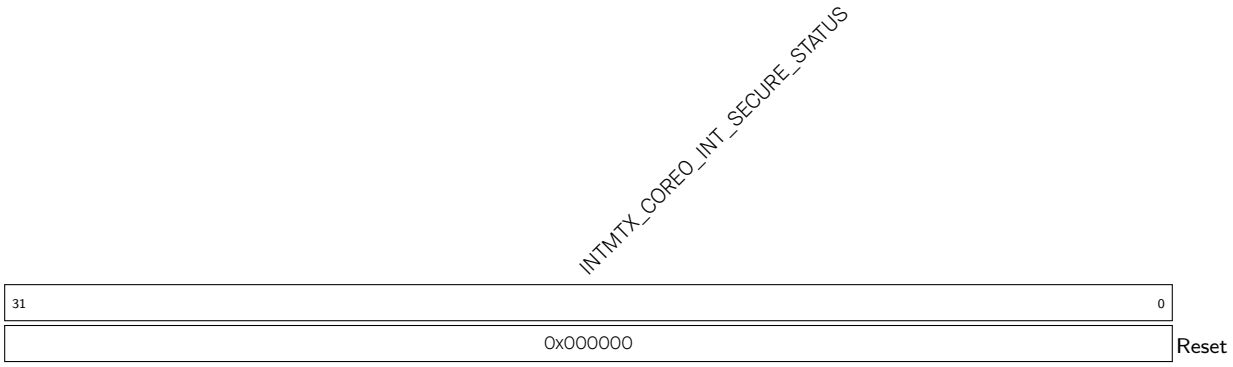
(RO)

Register 11.77. INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC_REG (0x0168)



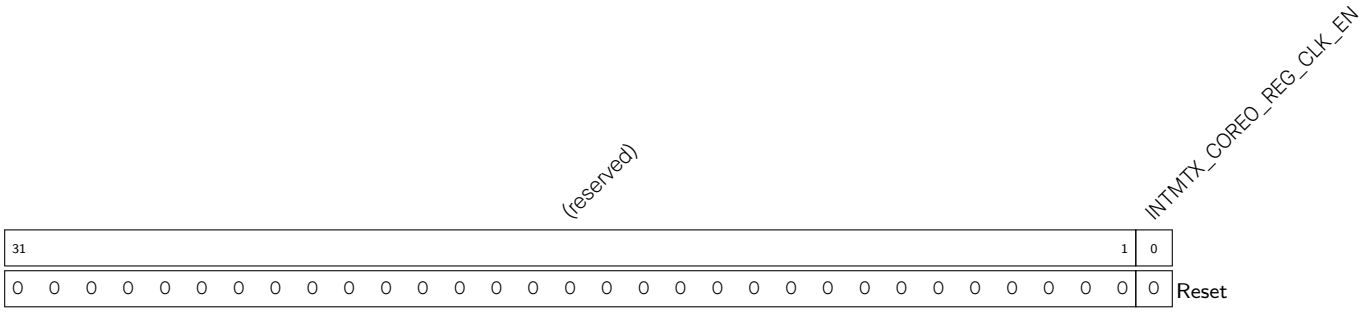
INTMTX_COREO_INT_SIG_IDX_ASSERT_IN_SEC 配置中断应被委托至哪个 CPU 外部机器模式中断源。(R/W)

Register 11.78. INTMTX_COREO_SECURE_STATUS_REG (0x016C)



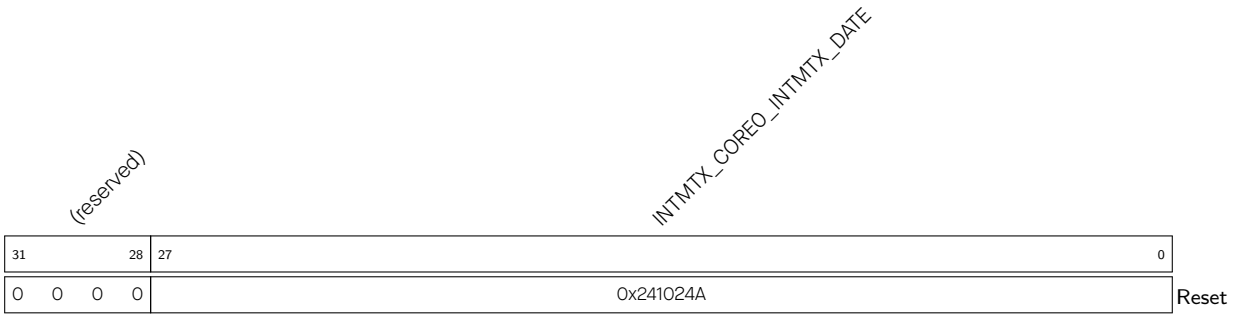
INTMTX_COREO_INT_SECURE_STATUS 表示被委托到 CPU 机器模式的中断，原本应该映射到哪个 CPU 用户模式中断。(RO)

Register 11.79. INTMTX_COREO_CLOCK_GATE_REG (0x0170)



INTMTX_COREO_REG_CLK_EN 配置是否强制开启中断寄存器时钟门控。(R/W)

Register 11.80. INTMTX_COREO_INTMTX_DATE_REG (0x07FC)

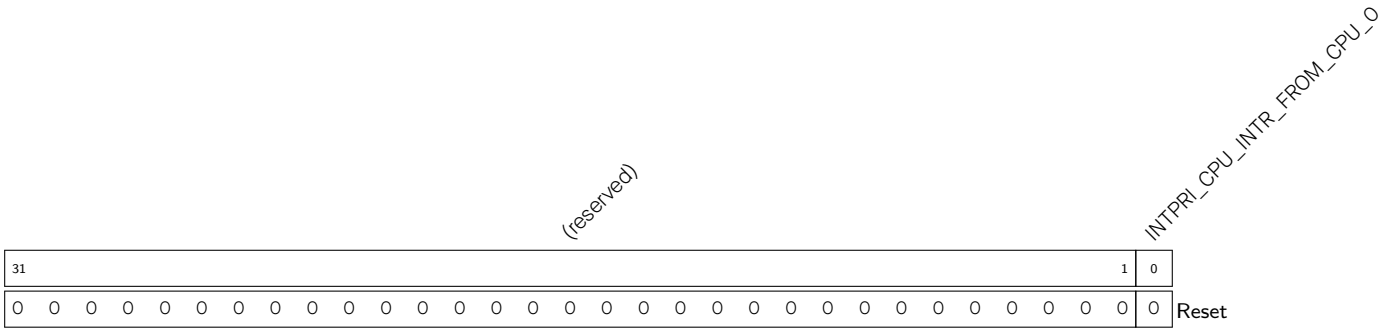


INTMTX_COREO_INTMTX_DATE 版本控制寄存器。(R/W)

11.7.2 软件中断寄存器

本小节的所有地址均为相对于软件中断基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 11.81. INTPRI_CPU_INTR_FROM_CPU_*n*_REG (*n*: 0-3) (0x0090+0x4*n*)



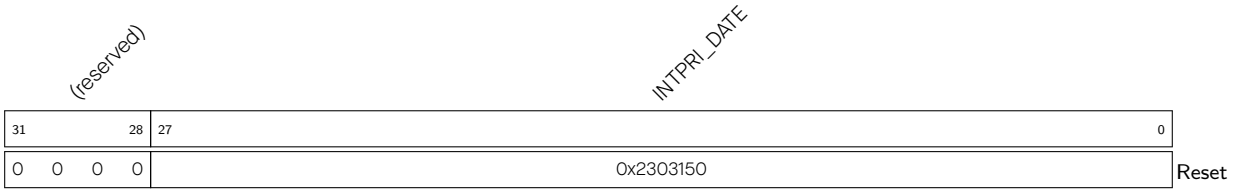
INTPRI_CPU_INTR_FROM_CPU_*n* CPU_INTR_FROM_CPU_*n* 映射寄存器，通过软件配置寄存器产生中断。

0：停止产生中断

1：产生中断

(R/W)

Register 11.82. INTPRI_DATE_REG (0x00A0)



INTPRI_DATE 版本控制寄存器。(R/W)

第 12 章

事件任务矩阵 (ETM)

12.1 概述

事件任务矩阵 (ETM) 外设包含 50 个可配置通道。每个通道可以将任意指定外设的事件映射到任意指定外设的任务，从而触发外设执行指定任务，无需 CPU 干预。

12.2 特性

事件任务矩阵具有以下特性：

- 支持从多个外设接收多种事件
- 支持为多个外设生成多种任务
- 拥有 50 个可独立配置的 ETM 通道
- 每个通道接收到的事件可以是所有事件中的任意一个，每个通道接收到的事件可以映射到任意的任务上输出
- 每个 ETM 通道都可以独立使能或关闭。通道关闭时，不会响应所配置的事件，也不会生成要映射到的任务
- 支持查看每个事件和任务的触发状态
- 能够产生事件、接收任务的外设有：GPIO、LED PWM、通用定时器、RTC 定时器、系统定时器、MCPWM、温度传感器、ADC、I2S、LP CPU、GDMA 和 PMU

由于 50 个 ETM 通道的功能和操作都是相同的。以下部分将 ETM 通道统称为通道*n*（其中 *n* 范围从 0 到 49）。

12.3 功能描述

12.3.1 架构

事件任务矩阵有 50 个独立通道，每个通道可以选择所有事件中的任意一个作为该通道的输入（参考章节 12.3.2），并且可以选择映射事件到任意任务（参考章节 12.3.3）。每个通道都有独立的使能配置位（参考章节 12.3.6）。

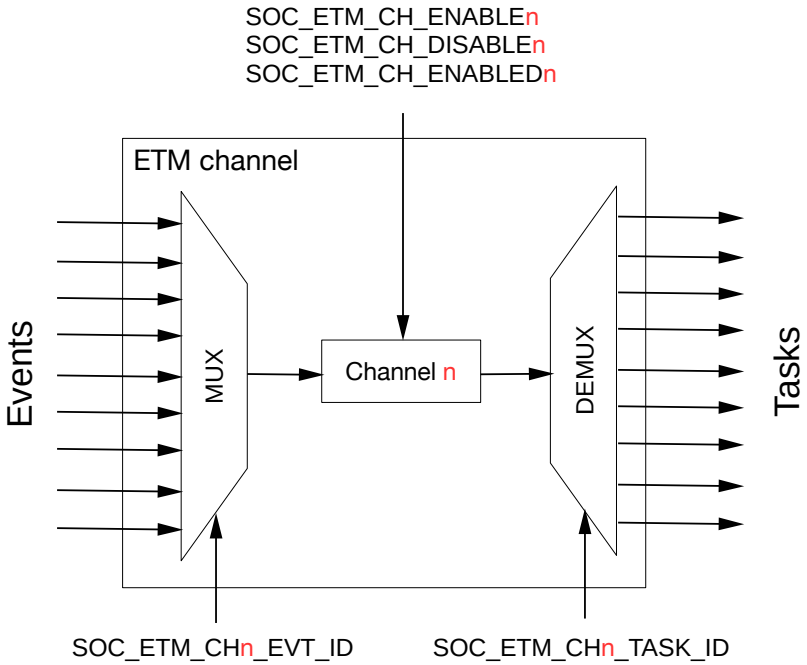


图 12.3-1. 事件任务矩阵通道*n*架构

图 12.3-1 展示了单个事件任务矩阵通道的结构，其中 SOC_ETM_CH*n*_EVT_ID 字段配置 MUX 选择任意一个事件作为通道*n* 的输入，SOC_ETM_CH*n*_TASK_ID 字段配置 DEMUX 以映射通道*n* 所选择的事件到任意一个任务。字段 SOC_ETM_CH_ENABLE*n* 和 SOC_ETM_CH_DISABLE*n* 用于使能或关闭通道*n*，字段 SOC_ETM_CH_ENABLED*n* 用于指示通道*n* 的开启或关闭状态。

12.3.2 事件

每个事件任务矩阵通道可以通过字段 SOC_ETM_CH*n*_EVT_ID 指定选择接收的事件。字段 SOC_ETM_CH*n*_EVT_ID 的配置值与所选择的事件的关系如表 12.3-1 所示。

表 12.3-1. 事件任务矩阵通道*n* 可选事件

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
1	GPIO_EVT_CH0_RISE_EDGE	GPIO
2	GPIO_EVT_CH1_RISE_EDGE	
3	GPIO_EVT_CH2_RISE_EDGE	
4	GPIO_EVT_CH3_RISE_EDGE	
5	GPIO_EVT_CH4_RISE_EDGE	
6	GPIO_EVT_CH5_RISE_EDGE	
7	GPIO_EVT_CH6_RISE_EDGE	
8	GPIO_EVT_CH7_RISE_EDGE	
9	GPIO_EVT_CH0_FALL_EDGE	
10	GPIO_EVT_CH1_FALL_EDGE	
11	GPIO_EVT_CH2_FALL_EDGE	
12	GPIO_EVT_CH3_FALL_EDGE	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
13	GPIO_EVT_CH4_FALL_EDGE	
14	GPIO_EVT_CH5_FALL_EDGE	
15	GPIO_EVT_CH6_FALL_EDGE	
16	GPIO_EVT_CH7_FALL_EDGE	
17	GPIO_EVT_CH0_ANY_EDGE	
18	GPIO_EVT_CH1_ANY_EDGE	
19	GPIO_EVT_CH2_ANY_EDGE	
20	GPIO_EVT_CH3_ANY_EDGE	
21	GPIO_EVT_CH4_ANY_EDGE	
22	GPIO_EVT_CH5_ANY_EDGE	
23	GPIO_EVT_CH6_ANY_EDGE	
24	GPIO_EVT_CH7_ANY_EDGE	
25	GPIO_EVT_ZERO_DET_POS0	
26	GPIO_EVT_ZERO_DET_NEG0	
27	LEDC_EVT_DUTY_CHNG_END_CH0	LED PWM 控制器 (LEDC)
28	LEDC_EVT_DUTY_CHNG_END_CH1	
29	LEDC_EVT_DUTY_CHNG_END_CH2	
30	LEDC_EVT_DUTY_CHNG_END_CH3	
31	LEDC_EVT_DUTY_CHNG_END_CH4	
32	LEDC_EVT_DUTY_CHNG_END_CH5	
33	LEDC_EVT_OVF_CNT_PLS_CH0	
34	LEDC_EVT_OVF_CNT_PLS_CH1	
35	LEDC_EVT_OVF_CNT_PLS_CH2	
36	LEDC_EVT_OVF_CNT_PLS_CH3	
37	LEDC_EVT_OVF_CNT_PLS_CH4	
38	LEDC_EVT_OVF_CNT_PLS_CH5	
39	LEDC_EVT_TIME_OVF_TIMER0	
40	LEDC_EVT_TIME_OVF_TIMER1	
41	LEDC_EVT_TIME_OVF_TIMER2	
42	LEDC_EVT_TIME_OVF_TIMER3	
43	LEDC_EVT_TIMER0_CMP	
44	LEDC_EVT_TIMER1_CMP	
45	LEDC_EVT_TIMER2_CMP	
46	LEDC_EVT_TIMER3_CMP	
47	TG0_EVT_CNT_CMP_TIMER0	通用定时器组 0
49	TG1_EVT_CNT_CMP_TIMER0	通用定时器组 1
51	SYSTIMER_EVT_CNT_CMP0	系统定时器
52	SYSTIMER_EVT_CNT_CMP1	
53	SYSTIMER_EVT_CNT_CMP2	
54	MCPWM0_EVT_TIMER0_STOP	MCPWM0
55	MCPWM0_EVT_TIMER1_STOP	
56	MCPWM0_EVT_TIMER2_STOP	
57	MCPWM0_EVT_TIMER0_TEZ	
58	MCPWM0_EVT_TIMER1_TEZ	
59	MCPWM0_EVT_TIMER2_TEZ	

SOC_ETM_CHn_EVT_ID	所选事件	产生该事件的外设
60	MCPWM0_EVT_TIMER0_TEP	
61	MCPWM0_EVT_TIMER1_TEP	
62	MCPWM0_EVT_TIMER2_TEP	
63	MCPWM0_EVT_OP0_TEA	
64	MCPWM0_EVT_OP1_TEA	
65	MCPWM0_EVT_OP2_TEA	
66	MCPWM0_EVT_OP0_TEB	
67	MCPWM0_EVT_OP1_TEB	
68	MCPWM0_EVT_OP2_TEB	
69	MCPWM0_EVT_F0	
70	MCPWM0_EVT_F1	
71	MCPWM0_EVT_F2	
72	MCPWM0_EVT_F0_CLR	
73	MCPWM0_EVT_F1_CLR	
74	MCPWM0_EVT_F2_CLR	
75	MCPWM0_EVT_TZ0_CBC	
76	MCPWM0_EVT_TZ1_CBC	
77	MCPWM0_EVT_TZ2_CBC	
78	MCPWM0_EVT_TZ0_OST	
79	MCPWM0_EVT_TZ1_OST	
80	MCPWM0_EVT_TZ2_OST	
81	MCPWM0_EVT_CAP0	
82	MCPWM0_EVT_CAP1	
83	MCPWM0_EVT_CAP2	
84	MCPWM0_EVT_OP0_TEE1	
85	MCPWM0_EVT_OP1_TEE1	
86	MCPWM0_EVT_OP2_TEE1	
87	MCPWM0_EVT_OP0_TEE2	
88	MCPWM0_EVT_OP1_TEE2	
89	MCPWM0_EVT_OP2_TEE2	
90	ADC_EVT_CONV_CMPLT0	ADC 控制器
91	ADC_EVT_EQ_ABOVE_THRESH0	
92	ADC_EVT_EQ_ABOVE_THRESH1	
93	ADC_EVT_EQ_BELOW_THRESH0	
94	ADC_EVT_EQ_BELOW_THRESH1	
96	ADC_EVT_STOPPED0	
97	ADC_EVT_STARTED0	
106	TMPSNSR_EVT_OVER_LIMIT	温度传感器
107	I2S0_EVT_RX_DONE	I2S
108	I2S0_EVT_TX_DONE	
109	I2S0_EVT_X_WORDS_RECEIVED	
110	I2S0_EVT_X_WORDS_SENT	
111	ULP_EVT_ERR_INTR	低功耗处理器
113	ULP_EVT_START_INTR	
114	RTC_EVT_TICK	实时时钟定时器

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
115	RTC_EVT_OVF	
116	RTC_EVT_CMP	
117	GDMA_EVT_IN_DONE_CH0	通用 DMA 控制器 (GDMA)
118	GDMA_EVT_IN_DONE_CH1	
119	GDMA_EVT_IN_DONE_CH2	
120	GDMA_EVT_IN_SUC_EOF_CH0	
121	GDMA_EVT_IN_SUC_EOF_CH1	
122	GDMA_EVT_IN_SUC_EOF_CH2	
123	GDMA_EVT_IN_FIFO_EMPTY_CH0	
124	GDMA_EVT_IN_FIFO_EMPTY_CH1	
125	GDMA_EVT_IN_FIFO_EMPTY_CH2	
126	GDMA_EVT_IN_FIFO_FULL_CH0	
127	GDMA_EVT_IN_FIFO_FULL_CH1	
128	GDMA_EVT_IN_FIFO_FULL_CH2	
129	GDMA_EVT_OUT_DONE_CH0	
130	GDMA_EVT_OUT_DONE_CH1	
131	GDMA_EVT_OUT_DONE_CH2	
132	GDMA_EVT_OUT_EOF_CH0	
133	GDMA_EVT_OUT_EOF_CH1	
134	GDMA_EVT_OUT_EOF_CH2	
135	GDMA_EVT_OUT_TOTAL_EOF_CH0	
136	GDMA_EVT_OUT_TOTAL_EOF_CH1	
137	GDMA_EVT_OUT_TOTAL_EOF_CH2	
138	GDMA_EVT_OUT_FIFO_EMPTY_CH0	
139	GDMA_EVT_OUT_FIFO_EMPTY_CH1	
140	GDMA_EVT_OUT_FIFO_EMPTY_CH2	
141	GDMA_EVT_OUT_FIFO_FULL_CH0	
142	GDMA_EVT_OUT_FIFO_FULL_CH1	
143	GDMA_EVT_OUT_FIFO_FULL_CH2	
144	PMU_EVT_SLEEP_WEEKUP	PMU

以上任意事件产生时，对应外设会生成一个脉冲信号。脉冲信号为高电平时，视为接收到相应的事件。

对于事件更详细的描述，请参考产生该事件外设的章节。

12.3.3 任务

每个事件任务矩阵通道可以通过字段 SOC_ETM_CH*n*_TASK_ID 选择要映射到的任务。字段 SOC_ETM_CH*n*_TASK_ID 的配置值与要映射到的任务的关系如表 12.3-2 所示。

表 12.3-2. 事件任务矩阵通道*n*可映射任务

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
1	GPIO_TASK_CH0_SET	GPIO
2	GPIO_TASK_CH1_SET	
3	GPIO_TASK_CH2_SET	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
4	GPIO_TASK_CH3_SET	
5	GPIO_TASK_CH4_SET	
6	GPIO_TASK_CH5_SET	
7	GPIO_TASK_CH6_SET	
8	GPIO_TASK_CH7_SET	
9	GPIO_TASK_CH0_CLEAR	
10	GPIO_TASK_CH1_CLEAR	
11	GPIO_TASK_CH2_CLEAR	
12	GPIO_TASK_CH3_CLEAR	
13	GPIO_TASK_CH4_CLEAR	
14	GPIO_TASK_CH5_CLEAR	
15	GPIO_TASK_CH6_CLEAR	
16	GPIO_TASK_CH7_CLEAR	
17	GPIO_TASK_CH0_TOGGLE	
18	GPIO_TASK_CH1_TOGGLE	
19	GPIO_TASK_CH2_TOGGLE	
20	GPIO_TASK_CH3_TOGGLE	
21	GPIO_TASK_CH4_TOGGLE	
22	GPIO_TASK_CH5_TOGGLE	
23	GPIO_TASK_CH6_TOGGLE	
24	GPIO_TASK_CH7_TOGGLE	
25	LEDC_TASK_TIMER0_RES_UPDATE	LED PWM 控制器 (LEDC)
26	LEDC_TASK_TIMER1_RES_UPDATE	
27	LEDC_TASK_TIMER2_RES_UPDATE	
28	LEDC_TASK_TIMER3_RES_UPDATE	
29	LEDC_TASK_DUTY_SCALE_UPDATE_CH0	
30	LEDC_TASK_DUTY_SCALE_UPDATE_CH1	
31	LEDC_TASK_DUTY_SCALE_UPDATE_CH2	
32	LEDC_TASK_DUTY_SCALE_UPDATE_CH3	
33	LEDC_TASK_DUTY_SCALE_UPDATE_CH4	
34	LEDC_TASK_DUTY_SCALE_UPDATE_CH5	
35	LEDC_TASK_TIMER0_CAP	
36	LEDC_TASK_TIMER1_CAP	
37	LEDC_TASK_TIMER2_CAP	
38	LEDC_TASK_TIMER3_CAP	
39	LEDC_TASK_SIG_OUT_DIS_CH0	
40	LEDC_TASK_SIG_OUT_DIS_CH1	
41	LEDC_TASK_SIG_OUT_DIS_CH2	
42	LEDC_TASK_SIG_OUT_DIS_CH3	
43	LEDC_TASK_SIG_OUT_DIS_CH4	
44	LEDC_TASK_SIG_OUT_DIS_CH5	
45	LEDC_TASK_OVF_CNT_RST_CH0	
46	LEDC_TASK_OVF_CNT_RST_CH1	
47	LEDC_TASK_OVF_CNT_RST_CH2	
48	LEDC_TASK_OVF_CNT_RST_CH3	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
49	LEDC_TASK_OVF_CNT_RST_CH4	
50	LEDC_TASK_OVF_CNT_RST_CH5	
51	LEDC_TASK_TIMER0_RST	
52	LEDC_TASK_TIMER1_RST	
53	LEDC_TASK_TIMER2_RST	
54	LEDC_TASK_TIMER3_RST	
55	LEDC_TASK_TIMER0_RESUME	
56	LEDC_TASK_TIMER1_RESUME	
57	LEDC_TASK_TIMER2_RESUME	
58	LEDC_TASK_TIMER3_RESUME	
59	LEDC_TASK_TIMER0_PAUSE	
60	LEDC_TASK_TIMER1_PAUSE	
61	LEDC_TASK_TIMER2_PAUSE	
62	LEDC_TASK_TIMER3_PAUSE	
63	LEDC_TASK_GAMMA_RESTART_CH0	
64	LEDC_TASK_GAMMA_RESTART_CH1	
65	LEDC_TASK_GAMMA_RESTART_CH2	
66	LEDC_TASK_GAMMA_RESTART_CH3	
67	LEDC_TASK_GAMMA_RESTART_CH4	
68	LEDC_TASK_GAMMA_RESTART_CH5	
69	LEDC_TASK_GAMMA_PAUSE_CH0	
70	LEDC_TASK_GAMMA_PAUSE_CH1	
71	LEDC_TASK_GAMMA_PAUSE_CH2	
72	LEDC_TASK_GAMMA_PAUSE_CH3	
73	LEDC_TASK_GAMMA_PAUSE_CH4	
74	LEDC_TASK_GAMMA_PAUSE_CH5	
75	LEDC_TASK_GAMMA_RESUME_CH0	
76	LEDC_TASK_GAMMA_RESUME_CH1	
77	LEDC_TASK_GAMMA_RESUME_CH2	
78	LEDC_TASK_GAMMA_RESUME_CH3	
79	LEDC_TASK_GAMMA_RESUME_CH4	
80	LEDC_TASK_GAMMA_RESUME_CH5	
81	TG0_TASK_CNT_START_TIMER0	通用定时器组 0
82	TG0_TASK_ALARM_START_TIMER0	
83	TG0_TASK_CNT_STOP_TIMER0	
84	TG0_TASK_CNT_RELOAD_TIMER0	
85	TG0_TASK_CNT_CAP_TIMER0	
91	TG1_TASK_CNT_START_TIMER0	通用定时器组 1
92	TG1_TASK_ALARM_START_TIMER0	
93	TG1_TASK_CNT_STOP_TIMER0	
94	TG1_TASK_CNT_RELOAD_TIMER0	
95	TG1_TASK_CNT_CAP_TIMER0	
101	MCPWM0_TASK_CMPR0_A_UP	MCPWM0
102	MCPWM0_TASK_CMPR1_A_UP	
103	MCPWM0_TASK_CMPR2_A_UP	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
104	MCPWM0_TASK_CMPR0_B_UP	
105	MCPWM0_TASK_CMPR1_B_UP	
106	MCPWM0_TASK_CMPR2_B_UP	
107	MCPWM0_TASK_GEN_STOP	
108	MCPWM0_TASK_TIMER0_SYN	
109	MCPWM0_TASK_TIMER1_SYN	
110	MCPWM0_TASK_TIMER2_SYN	
111	MCPWM0_TASK_TIMER0_PERIOD_UP	
112	MCPWM0_TASK_TIMER1_PERIOD_UP	
113	MCPWM0_TASK_TIMER2_PERIOD_UP	
114	MCPWM0_TASK_TZ0_OST	
115	MCPWM0_TASK_TZ1_OST	
116	MCPWM0_TASK_TZ2_OST	
117	MCPWM0_TASK_CLR0_OST	
118	MCPWM0_TASK_CLR1_OST	
119	MCPWM0_TASK_CLR2_OST	
120	MCPWM0_TASK_CAP0	
121	MCPWM0_TASK_CAP1	
122	MCPWM0_TASK_CAP2	
123	ADC_TASK_SAMPLE0	ADC 控制器
125	ADC_TASK_START0	
126	ADC_TASK_STOP0	
131	TMPSNSR_TASK_START_SAMPLE	温度传感器
132	TMPSNSR_TASK_STOP_SAMPLE	
133	I2S0_TASK_START_RX	I2S
134	I2S0_TASK_START_TX	
135	I2S0_TASK_STOP_RX	
136	I2S0_TASK_STOP_TX	
137	ULP_TASK_WEAKUP_CPU	低功耗处理器
143	GDMA_TASK_IN_START_CH0	通用 DMA 控制器 (GDMA)
144	GDMA_TASK_IN_START_CH1	
145	GDMA_TASK_IN_START_CH2	
146	GDMA_TASK_OUT_START_CH0	
147	GDMA_TASK_OUT_START_CH1	
148	GDMA_TASK_OUT_START_CH2	
149	PMU_TASK_SLEEP_REQ	PMU

一个通道接收到一个有效事件脉冲信号时，会产生映射任务脉冲信号。

对于任务更详细的描述，请参考接收该任务外设的章节。

来自多个不同通道的事件可以选择映射到同一个任务。例如，可以将多个通道的字段 SOC_ETM_CH*n*_TASK_ID 都配置为相同的值，而字段 SOC_ETM_CH*n*_EVT_ID 可以配置为相同或不同的值。在这种情况下，当任意一个通道接收到的事件有效时，指定的任务就会被生成。当多个通道接收到的事件同时有效时，指定的任务只会产生一次。

12.3.4 事件任务触发状态

ETM 外设支持通过读取寄存器来查看事件和任务的触发状态。

- 事件触发状态：读取寄存器 SOC_ETM_EVT_STn_REG 中所要查看事件对应的字段，如果该字段的值为 1，表示对应的事件被 ETM 接收过，如果该字段的值为 0，表示对应的事件未被 ETM 接收到。向寄存器 SOC_ETM_EVT_STn_CLR_REG 中所要查看事件对应的字段写 1，可以将寄存器 SOC_ETM_EVT_STn_REG 中所要查看事件对应的字段清 0。
- 任务触发状态：读取寄存器 SOC_ETM_TASK_STn_REG 中所要查看任务对应的字段，如果该字段的值为 1，表示 ETM 产生过对应的任务，如果该字段的值为 0，表示 ETM 未产生过对应的任务。向寄存器 SOC_ETM_TASK_STn_CLR_REG 中所要查看任务对应的字段写 1，可以将寄存器 SOC_ETM_TASK_STn_REG 中所要查看任务对应的字段清 0。

12.3.5 时序考虑因素

ETM 外设接收的事件、发送的任务和 ETM 通道之间驱动时钟的结构如图 12.3-2 所示。

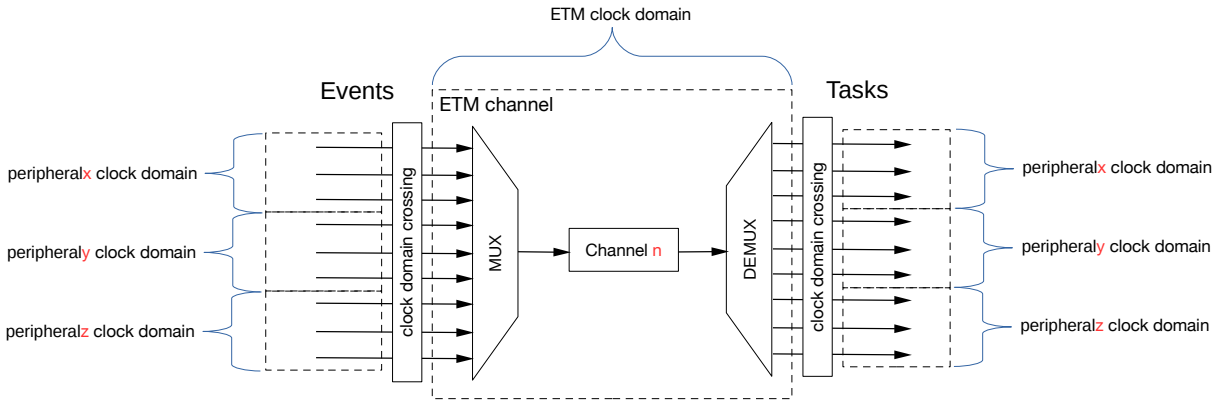


图 12.3-2. 事件任务矩阵时钟结构

ETM 运行在 SYS_CLK 时钟域下（详见章节 9 复位和时钟），每个事件对应一个由相应外设在其时钟域中产生的脉冲信号，每个任务被 ETM 映射为对应外设时钟域下的脉冲信号。由于产生事件的外设、事件任务矩阵和接收任务的外设不一定由同个时钟驱动，需要同步处理，两个连续时间之间需要根据生成事件外设的时钟、ETM 的时钟、执行任务外设的时钟设置延迟，否则会丢失事件。

为了让事件任务矩阵成功接收到每个事件，对于每个可以产生事件的外设，其产生的两个连续事件脉冲之间的时间间隔必须大于一个事件任务矩阵的时钟周期，即 $ceil(\frac{peripheral\_clock\_frequenty}{ETM\_clock\_frequenty})$ 个外设时钟周期。

例如，假设外设 A 所在时钟域（PLL_F80M_CLK）为 80 MHz，ETM 所在时钟域（SYS_CLK）为 40 MHz，外设 A 生成事件 1，ETM 接收事件 1。要成功接收每个事件 1，两个连续事件 1 之间的时间间隔必须大于外设 A 的两个时钟周期（即一个 ETM 时钟周期）。

同样的，为了使事件任务矩阵成功地将接收到的事件（即同步到事件任务矩阵时钟域的事件）映射为外设的任务，事件任务矩阵时钟域中的两个连续事件脉冲之间的时间间隔必须大于一个外设的时钟周期，即 $ceil(\frac{ETM\_clock\_frequenty}{peripheral\_clock\_frequenty})$ 个事件任务矩阵时钟周期。

例如，假设外设 B 所在时钟域（RC_FAST_CLK）为 20 MHz，ETM 所在时钟域（SYS_CLK）为 40 MHz，ETM 将事件映射到任务 1，外设 B 接收任务 1。要成功将每个接收到的事件映射到任务 1，两个连续事件之间的时间

间隔必须大于两个 ETM 时钟周期（即一个外设 B 时钟周期）。

因此，要将外设 A 产生的两个连续事件完整映射到外设 B 的两个任务中，这两个事件之间的时间间隔必须大于 $ceil(\frac{peripheral\_A\_clock\_frequenty}{ETM\_clock\_frequenty}) * ceil(\frac{ETM\_clock\_frequenty}{peripheral\_B\_clock\_frequenty})$ 个外设 A 时钟周期。

例如，假设外设 A 所在时钟域（PLL_F80M_CLK）为 80 MHz，外设 B 所在时钟域（RC_FAST_CLK）为 20 MHz，ETM 所在时钟域（SYS_CLK）为 40 MHz，外设 A 生成事件 1，ETM 将事件映射到任务 1，外设 B 接收任务 1。要成功将每个事件 1（外设 A 生成）映射到任务 1（外设 B 接收），两个连续事件 1 之间的时间间隔必须大于 $2 * 2 = 4$ 个外设 A 时钟周期。

12.3.6 通道控制

事件任务矩阵的每个通道都可以独立配置为使能或关闭。当通道*n* 配置为使能时，如果该通道接收到所配置的事件（字段 SOC_ETM_CH*n*_EVT_ID），该事件将映射到所配置的任务上（字段 SOC_ETM_CH*n*_TASK_ID）。当通道*n* 配置为不使能时，即使该通道接收到所配置的事件（字段 SOC_ETM_CH*n*_EVT_ID），也不会产生任何任务。

使能事件任务矩阵通道*n*，需向字段 SOC_ETM_CH_ENABLE*n* 写 1。关闭事件任务矩阵通道*n*，需向字段 SOC_ETM_CH_DISABLE*n* 写 1。

通道*n* 的状态可通过读取字段 SOC_ETM_CH_ENABLED*n* 的值获取。1 表示通道*n* 已使能，0 表示通道*n* 未使能。

如果将字段 SOC_ETM_CH*n*_EVT_ID 的值配置为 0 或将字段 SOC_ETM_CH*n*_TASK_ID 的值配置为 0，事件任务矩阵通道*n* 也将被关闭。

事件任务矩阵通道*n* 的完整配置过程如下：

1. 向字段 PCR_ETM_CLK_EN 写 1，开启 ETM 的时钟
2. 配置字段 SOC_ETM_CH*n*_EVT_ID 的值，选择通道*n* 所要接收的事件
3. 配置字段 SOC_ETM_CH*n*_TASK_ID 的值，选择通道*n* 所接收到的事件要映射到的任务
4. 向字段 SOC_ETM_CH_ENABLE*n* 写 1 使能通道
5. 当通道*n* 不再需要将配置的事件映射到配置的任务时，向字段 SOC_ETM_CH_DISABLE*n* 写 1 来关闭通道*n*。之后，如果要为通道*n* 配置新的事件和任务的映射关系，则重复步骤 2-4，如果不再配置，则通道*n* 保持停止运行状态
6. 若需复位整个 ETM，即复位所有通道的配置，可通过向 PCR_ETM_RST_EN 字段先写 1 再写 0 来实现

12.4 寄存器列表

本小节的所有地址均为相对于事件任务矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
状态寄存器			
SOC_ETM_CH_ENA_AD0_REG	通道使能状态寄存器	0x0000	R/WTC/WTS
SOC_ETM_CH_ENA_AD1_REG	通道使能状态寄存器	0x000C	R/WTC/WTS
SOC_ETM_EVT_ST0_REG	事件触发状态寄存器	0x01A8	R/WTC/SS
SOC_ETM_EVT_ST1_REG	事件触发状态寄存器	0x01B0	R/WTC/SS
SOC_ETM_EVT_ST2_REG	事件触发状态寄存器	0x01B8	R/WTC/SS
SOC_ETM_EVT_ST3_REG	事件触发状态寄存器	0x01C0	R/WTC/SS
SOC_ETM_EVT_ST4_REG	事件触发状态寄存器	0x01C8	R/WTC/SS
SOC_ETM_TASK_ST0_REG	任务触发状态寄存器	0x01D0	R/WTC/SS
SOC_ETM_TASK_ST1_REG	任务触发状态寄存器	0x01D8	R/WTC/SS
SOC_ETM_TASK_ST2_REG	任务触发状态寄存器	0x01E0	R/WTC/SS
SOC_ETM_TASK_ST3_REG	任务触发状态寄存器	0x01E8	R/WTC/SS
SOC_ETM_TASK_ST4_REG	任务触发状态寄存器	0x01F0	R/WTC/SS
配置寄存器			
SOC_ETM_CH_ENA_AD0_SET_REG	通道使能寄存器	0x0004	WT
SOC_ETM_CH_ENA_AD0_CLR_REG	通道关闭寄存器	0x0008	WT
SOC_ETM_CH_ENA_AD1_SET_REG	通道使能寄存器	0x0010	WT
SOC_ETM_CH_ENA_AD1_CLR_REG	通道关闭寄存器	0x0014	WT
SOC_ETM_CH0_EVT_ID_REG	通道 0 事件 ID 寄存器	0x0018	R/W
SOC_ETM_CH0_TASK_ID_REG	通道 0 任务 ID 寄存器	0x001C	R/W
SOC_ETM_CH1_EVT_ID_REG	通道 1 事件 ID 寄存器	0x0020	R/W
SOC_ETM_CH1_TASK_ID_REG	通道 1 任务 ID 寄存器	0x0024	R/W
SOC_ETM_CH2_EVT_ID_REG	通道 2 事件 ID 寄存器	0x0028	R/W
SOC_ETM_CH2_TASK_ID_REG	通道 2 任务 ID 寄存器	0x002C	R/W
SOC_ETM_CH3_EVT_ID_REG	通道 3 事件 ID 寄存器	0x0030	R/W
SOC_ETM_CH3_TASK_ID_REG	通道 3 任务 ID 寄存器	0x0034	R/W
SOC_ETM_CH4_EVT_ID_REG	通道 4 事件 ID 寄存器	0x0038	R/W
SOC_ETM_CH4_TASK_ID_REG	通道 4 任务 ID 寄存器	0x003C	R/W
SOC_ETM_CH5_EVT_ID_REG	通道 5 事件 ID 寄存器	0x0040	R/W
SOC_ETM_CH5_TASK_ID_REG	通道 5 任务 ID 寄存器	0x0044	R/W
SOC_ETM_CH6_EVT_ID_REG	通道 6 事件 ID 寄存器	0x0048	R/W
SOC_ETM_CH6_TASK_ID_REG	通道 6 任务 ID 寄存器	0x004C	R/W
SOC_ETM_CH7_EVT_ID_REG	通道 7 事件 ID 寄存器	0x0050	R/W
SOC_ETM_CH7_TASK_ID_REG	通道 7 任务 ID 寄存器	0x0054	R/W
SOC_ETM_CH8_EVT_ID_REG	通道 8 事件 ID 寄存器	0x0058	R/W
SOC_ETM_CH8_TASK_ID_REG	通道 8 任务 ID 寄存器	0x005C	R/W
SOC_ETM_CH9_EVT_ID_REG	通道 9 事件 ID 寄存器	0x0060	R/W

名称	描述	地址	访问
SOC_ETM_CH9_TASK_ID_REG	通道 9 任务 ID 寄存器	0x0064	R/W
SOC_ETM_CH10_EVT_ID_REG	通道 10 事件 ID 寄存器	0x0068	R/W
SOC_ETM_CH10_TASK_ID_REG	通道 10 任务 ID 寄存器	0x006C	R/W
SOC_ETM_CH11_EVT_ID_REG	通道 11 事件 ID 寄存器	0x0070	R/W
SOC_ETM_CH11_TASK_ID_REG	通道 11 任务 ID 寄存器	0x0074	R/W
SOC_ETM_CH12_EVT_ID_REG	通道 12 事件 ID 寄存器	0x0078	R/W
SOC_ETM_CH12_TASK_ID_REG	通道 12 任务 ID 寄存器	0x007C	R/W
SOC_ETM_CH13_EVT_ID_REG	通道 13 事件 ID 寄存器	0x0080	R/W
SOC_ETM_CH13_TASK_ID_REG	通道 13 任务 ID 寄存器	0x0084	R/W
SOC_ETM_CH14_EVT_ID_REG	通道 14 事件 ID 寄存器	0x0088	R/W
SOC_ETM_CH14_TASK_ID_REG	通道 14 任务 ID 寄存器	0x008C	R/W
SOC_ETM_CH15_EVT_ID_REG	通道 15 事件 ID 寄存器	0x0090	R/W
SOC_ETM_CH15_TASK_ID_REG	通道 15 任务 ID 寄存器	0x0094	R/W
SOC_ETM_CH16_EVT_ID_REG	通道 16 事件 ID 寄存器	0x0098	R/W
SOC_ETM_CH16_TASK_ID_REG	通道 16 任务 ID 寄存器	0x009C	R/W
SOC_ETM_CH17_EVT_ID_REG	通道 17 事件 ID 寄存器	0x00A0	R/W
SOC_ETM_CH17_TASK_ID_REG	通道 17 任务 ID 寄存器	0x00A4	R/W
SOC_ETM_CH18_EVT_ID_REG	通道 18 事件 ID 寄存器	0x00A8	R/W
SOC_ETM_CH18_TASK_ID_REG	通道 18 任务 ID 寄存器	0x00AC	R/W
SOC_ETM_CH19_EVT_ID_REG	通道 19 事件 ID 寄存器	0x00B0	R/W
SOC_ETM_CH19_TASK_ID_REG	通道 19 任务 ID 寄存器	0x00B4	R/W
SOC_ETM_CH20_EVT_ID_REG	通道 20 事件 ID 寄存器	0x00B8	R/W
SOC_ETM_CH20_TASK_ID_REG	通道 20 任务 ID 寄存器	0x00BC	R/W
SOC_ETM_CH21_EVT_ID_REG	通道 21 事件 ID 寄存器	0x00C0	R/W
SOC_ETM_CH21_TASK_ID_REG	通道 21 任务 ID 寄存器	0x00C4	R/W
SOC_ETM_CH22_EVT_ID_REG	通道 22 事件 ID 寄存器	0x00C8	R/W
SOC_ETM_CH22_TASK_ID_REG	通道 22 任务 ID 寄存器	0x00CC	R/W
SOC_ETM_CH23_EVT_ID_REG	通道 23 事件 ID 寄存器	0x00D0	R/W
SOC_ETM_CH23_TASK_ID_REG	通道 23 任务 ID 寄存器	0x00D4	R/W
SOC_ETM_CH24_EVT_ID_REG	通道 24 事件 ID 寄存器	0x00D8	R/W
SOC_ETM_CH24_TASK_ID_REG	通道 24 任务 ID 寄存器	0x00DC	R/W
SOC_ETM_CH25_EVT_ID_REG	通道 25 事件 ID 寄存器	0x00E0	R/W
SOC_ETM_CH25_TASK_ID_REG	通道 25 任务 ID 寄存器	0x00E4	R/W
SOC_ETM_CH26_EVT_ID_REG	通道 26 事件 ID 寄存器	0x00E8	R/W
SOC_ETM_CH26_TASK_ID_REG	通道 26 任务 ID 寄存器	0x00EC	R/W
SOC_ETM_CH27_EVT_ID_REG	通道 27 事件 ID 寄存器	0x00F0	R/W
SOC_ETM_CH27_TASK_ID_REG	通道 27 任务 ID 寄存器	0x00F4	R/W
SOC_ETM_CH28_EVT_ID_REG	通道 28 事件 ID 寄存器	0x00F8	R/W
SOC_ETM_CH28_TASK_ID_REG	通道 28 任务 ID 寄存器	0x00FC	R/W
SOC_ETM_CH29_EVT_ID_REG	通道 29 事件 ID 寄存器	0x0100	R/W
SOC_ETM_CH29_TASK_ID_REG	通道 29 任务 ID 寄存器	0x0104	R/W
SOC_ETM_CH30_EVT_ID_REG	通道 30 事件 ID 寄存器	0x0108	R/W
SOC_ETM_CH30_TASK_ID_REG	通道 30 任务 ID 寄存器	0x010C	R/W

名称	描述	地址	访问
SOC_ETM_CH31_EVT_ID_REG	通道 31 事件 ID 寄存器	0x0110	R/W
SOC_ETM_CH31_TASK_ID_REG	通道 31 任务 ID 寄存器	0x0114	R/W
SOC_ETM_CH32_EVT_ID_REG	通道 32 事件 ID 寄存器	0x0118	R/W
SOC_ETM_CH32_TASK_ID_REG	通道 32 任务 ID 寄存器	0x011C	R/W
SOC_ETM_CH33_EVT_ID_REG	通道 33 事件 ID 寄存器	0x0120	R/W
SOC_ETM_CH33_TASK_ID_REG	通道 33 任务 ID 寄存器	0x0124	R/W
SOC_ETM_CH34_EVT_ID_REG	通道 34 事件 ID 寄存器	0x0128	R/W
SOC_ETM_CH34_TASK_ID_REG	通道 34 任务 ID 寄存器	0x012C	R/W
SOC_ETM_CH35_EVT_ID_REG	通道 35 事件 ID 寄存器	0x0130	R/W
SOC_ETM_CH35_TASK_ID_REG	通道 35 任务 ID 寄存器	0x0134	R/W
SOC_ETM_CH36_EVT_ID_REG	通道 36 事件 ID 寄存器	0x0138	R/W
SOC_ETM_CH36_TASK_ID_REG	通道 36 任务 ID 寄存器	0x013C	R/W
SOC_ETM_CH37_EVT_ID_REG	通道 37 事件 ID 寄存器	0x0140	R/W
SOC_ETM_CH37_TASK_ID_REG	通道 37 任务 ID 寄存器	0x0144	R/W
SOC_ETM_CH38_EVT_ID_REG	通道 38 事件 ID 寄存器	0x0148	R/W
SOC_ETM_CH38_TASK_ID_REG	通道 38 任务 ID 寄存器	0x014C	R/W
SOC_ETM_CH39_EVT_ID_REG	通道 39 事件 ID 寄存器	0x0150	R/W
SOC_ETM_CH39_TASK_ID_REG	通道 39 任务 ID 寄存器	0x0154	R/W
SOC_ETM_CH40_EVT_ID_REG	通道 40 事件 ID 寄存器	0x0158	R/W
SOC_ETM_CH40_TASK_ID_REG	通道 40 任务 ID 寄存器	0x015C	R/W
SOC_ETM_CH41_EVT_ID_REG	通道 41 事件 ID 寄存器	0x0160	R/W
SOC_ETM_CH41_TASK_ID_REG	通道 41 任务 ID 寄存器	0x0164	R/W
SOC_ETM_CH42_EVT_ID_REG	通道 42 事件 ID 寄存器	0x0168	R/W
SOC_ETM_CH42_TASK_ID_REG	通道 42 任务 ID 寄存器	0x016C	R/W
SOC_ETM_CH43_EVT_ID_REG	通道 43 事件 ID 寄存器	0x0170	R/W
SOC_ETM_CH43_TASK_ID_REG	通道 43 任务 ID 寄存器	0x0174	R/W
SOC_ETM_CH44_EVT_ID_REG	通道 44 事件 ID 寄存器	0x0178	R/W
SOC_ETM_CH44_TASK_ID_REG	通道 44 任务 ID 寄存器	0x017C	R/W
SOC_ETM_CH45_EVT_ID_REG	通道 45 事件 ID 寄存器	0x0180	R/W
SOC_ETM_CH45_TASK_ID_REG	通道 45 任务 ID 寄存器	0x0184	R/W
SOC_ETM_CH46_EVT_ID_REG	通道 46 事件 ID 寄存器	0x0188	R/W
SOC_ETM_CH46_TASK_ID_REG	通道 46 任务 ID 寄存器	0x018C	R/W
SOC_ETM_CH47_EVT_ID_REG	通道 47 事件 ID 寄存器	0x0190	R/W
SOC_ETM_CH47_TASK_ID_REG	通道 47 任务 ID 寄存器	0x0194	R/W
SOC_ETM_CH48_EVT_ID_REG	通道 48 事件 ID 寄存器	0x0198	R/W
SOC_ETM_CH48_TASK_ID_REG	通道 48 任务 ID 寄存器	0x019C	R/W
SOC_ETM_CH49_EVT_ID_REG	通道 49 事件 ID 寄存器	0x01A0	R/W
SOC_ETM_CH49_TASK_ID_REG	通道 49 任务 ID 寄存器	0x01A4	R/W
SOC_ETM_EVT_ST0_CLR_REG	事件触发状态清除寄存器	0x01AC	WT
SOC_ETM_EVT_ST1_CLR_REG	事件触发状态清除寄存器	0x01B4	WT
SOC_ETM_EVT_ST2_CLR_REG	事件触发状态清除寄存器	0x01BC	WT
SOC_ETM_EVT_ST3_CLR_REG	事件触发状态清除寄存器	0x01C4	WT
SOC_ETM_EVT_ST4_CLR_REG	事件触发状态清除寄存器	0x01CC	WT

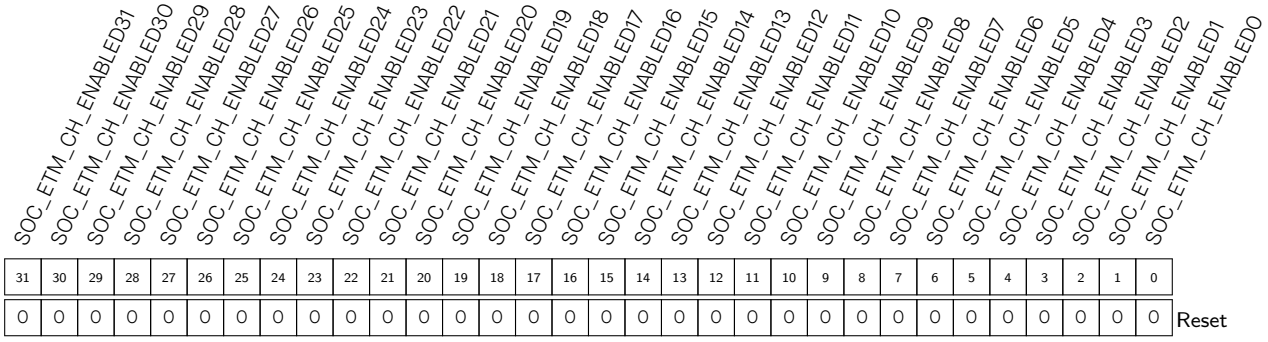
名称	描述	地址	访问
SOC_ETM_TASK_ST0_CLR_REG	任务触发状态清除寄存器	0x01D4	WT
SOC_ETM_TASK_ST1_CLR_REG	任务触发状态清除寄存器	0x01DC	WT
SOC_ETM_TASK_ST2_CLR_REG	任务触发状态清除寄存器	0x01E4	WT
SOC_ETM_TASK_ST3_CLR_REG	任务触发状态清除寄存器	0x01EC	WT
SOC_ETM_TASK_ST4_CLR_REG	任务触发状态清除寄存器	0x01F4	WT
SOC_ETM_CLK_EN_REG	ETM 时钟使能寄存器	0x01F8	R/W
版本寄存器			
SOC_ETM_DATE_REG	版本控制寄存器	0x01FC	R/W

12.5 寄存器

本小节的所有地址均为相对于事件任务矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 12.1. SOC_ETM_CH_ENA_AD0_REG (0x0000)



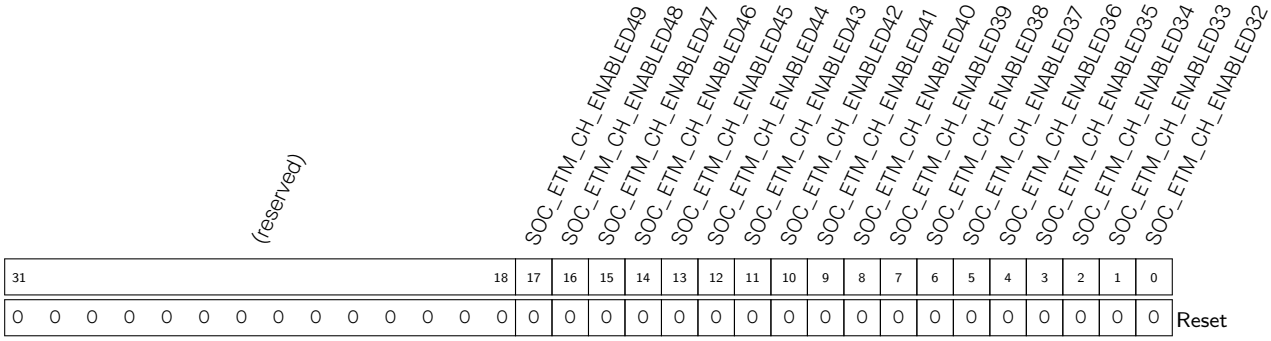
SOC_ETM_CH_ENABLED*(n*: 0-31) 表示通道 *n* 的使能状态。

0：关闭

1：使能

(R/WTC/WTS)

Register 12.2. SOC_ETM_CH_ENA_AD1_REG (0x000C)



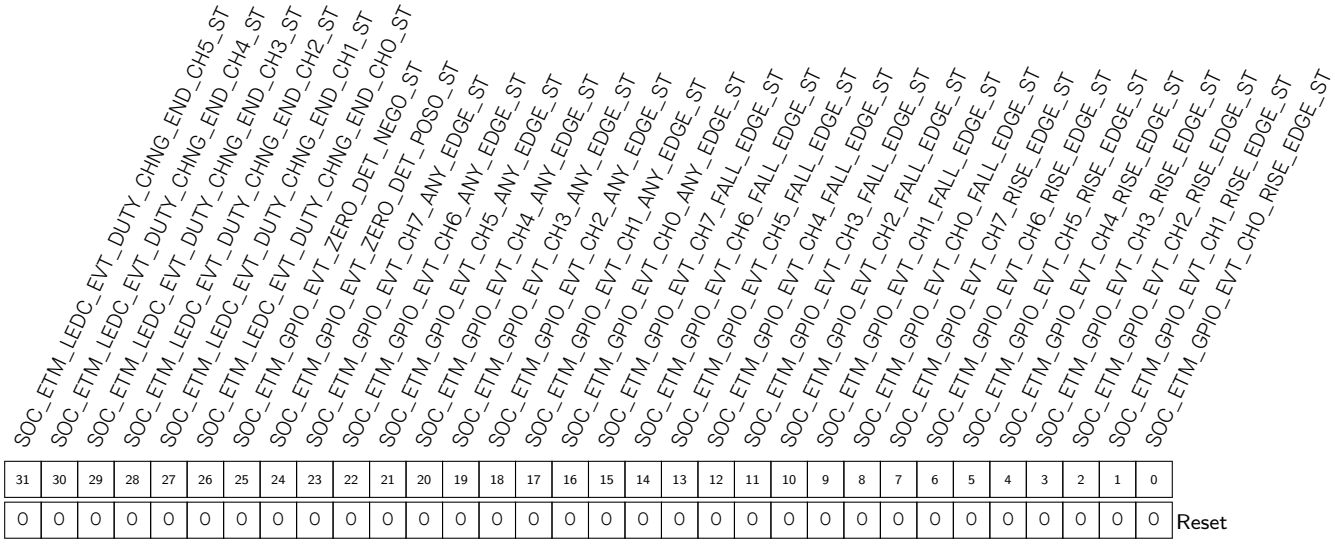
SOC_ETM_CH_ENABLED*n* (*n*: 32-49) 表示通道 *n* 的使能状态。

0：关闭

1：使能

(R/WTC/WTS)

Register 12.3. SOC_ETM_EVT_ST0_REG (0x01A8)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST 表示 GPIO_EVT_CH0_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST 表示 GPIO_EVT_CH1_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST 表示 GPIO_EVT_CH2_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST 表示 GPIO_EVT_CH3_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST 表示 GPIO_EVT_CH4_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST 表示 GPIO_EVT_CH5_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.3. SOC_ETM_EVT_ST0_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST 表示 GPIO_EVT_CH6_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST 表示 GPIO_EVT_CH7_RISE_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST 表示 GPIO_EVT_CH0_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST 表示 GPIO_EVT_CH1_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST 表示 GPIO_EVT_CH2_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST 表示 GPIO_EVT_CH3_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST 表示 GPIO_EVT_CH4_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST 表示 GPIO_EVT_CH5_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST 表示 GPIO_EVT_CH6_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.3. SOC_ETM_EVT_ST0_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST 表示 GPIO_EVT_CH7_FALL_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST 表示 GPIO_EVT_CH0_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST 表示 GPIO_EVT_CH1_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST 表示 GPIO_EVT_CH2_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST 表示 GPIO_EVT_CH3_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST 表示 GPIO_EVT_CH4_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST 表示 GPIO_EVT_CH5_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST 表示 GPIO_EVT_CH6_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.3. SOC_ETM_EVT_ST0_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST 表示 GPIO_EVT_CH7_ANY_EDGE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST 表示 GPIO_EVT_ZERO_DET_POS0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST 表示 GPIO_EVT_ZERO_DET_NEG0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH3 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH4 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.3. SOC_ETM_EVT_ST0_REG (0x01A8)

接上页...

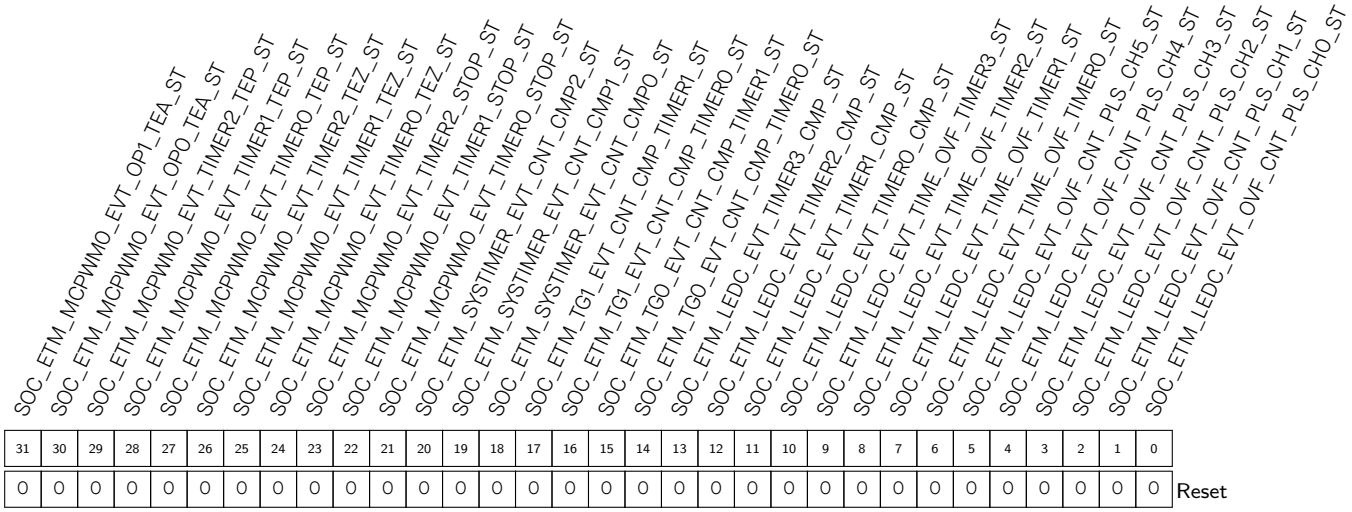
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH5 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.4. SOC_ETM_EVT_ST1_REG (0x01B0)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH3 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH4 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH5 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST 表示 LEDC_EVT_TIME_OVF_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST 表示 LEDC_EVT_TIME_OVF_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST 表示 LEDC_EVT_TIME_OVF_TIMER2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST 表示 LEDC_EVT_TIME_OVF_TIMER3 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST 表示 LEDC_EVT_TIMER0_CMP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST 表示 LEDC_EVT_TIMER1_CMP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST 表示 LEDC_EVT_TIMER2_CMP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST 表示 LEDC_EVT_TIMER3_CMP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TGO_EVT_CNT_CMP_TIMER0_ST 表示 TGO_EVT_CNT_CMP_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST 表示 TG0_EVT_CNT_CMP_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST 表示 TG1_EVT_CNT_CMP_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST 表示 TG1_EVT_CNT_CMP_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST 表示 SYSTIMER_EVT_CNT_CMP0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST 表示 SYSTIMER_EVT_CNT_CMP1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST 表示 SYSTIMER_EVT_CNT_CMP2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST 表示 MCPWM0_EVT_TIMER0_STOP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST 表示 MCPWM0_EVT_TIMER1_STOP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST 表示 MCPWM0_EVT_TIMER2_STOP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST 表示 MCPWM0_EVT_TIMER0_TEZ 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST 表示 MCPWM0_EVT_TIMER1_TEZ 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST 表示 MCPWM0_EVT_TIMER2_TEZ 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST 表示 MCPWM0_EVT_TIMER0_TEP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST 表示 MCPWM0_EVT_TIMER1_TEP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST 表示 MCPWM0_EVT_TIMER2_TEP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST 表示 MCPWM0_EVT_OP0_TEA 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

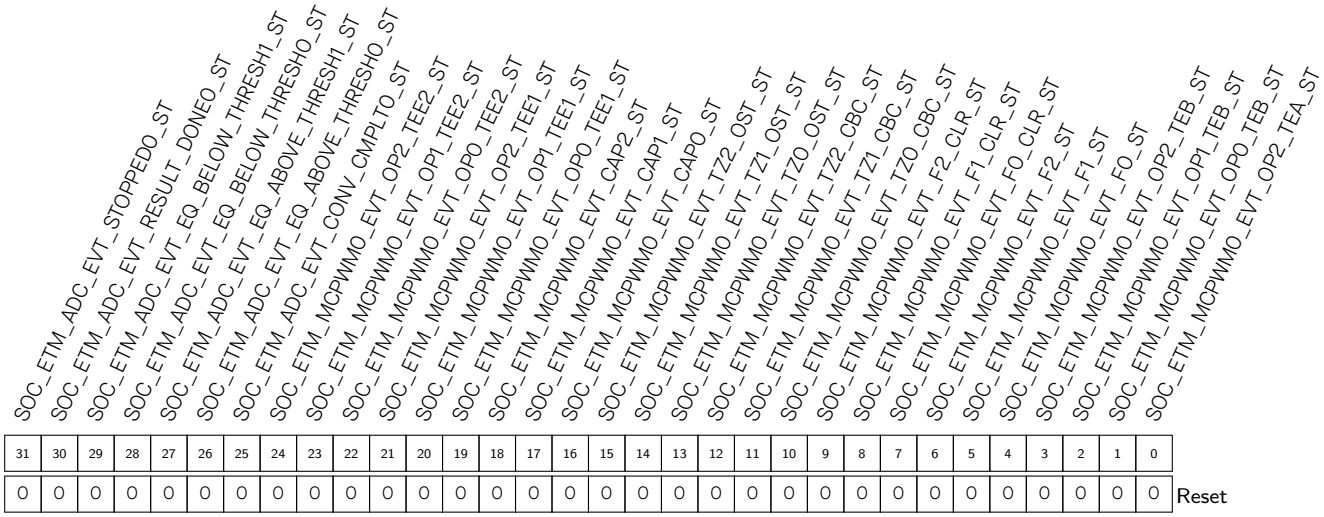
SOC_ETM_MCPWM0_EVT_OP1_TEA_ST 表示 MCPWM0_EVT_OP1_TEA 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.5. SOC_ETM_EVT_ST2_REG (0x01B8)



SOC_ETM_MCPWM0_EVT_OP2_TEA_ST 表示 MCPWM0_EVT_OP2_TEA 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST 表示 MCPWM0_EVT_OP0_TEB 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST 表示 MCPWM0_EVT_OP1_TEB 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST 表示 MCPWM0_EVT_OP2_TEB 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F0_ST 表示 MCPWM0_EVT_F0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_ST 表示 MCPWM0_EVT_F1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F2_ST 表示 MCPWM0_EVT_F2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_MCPWM0_EVT_F0_CLR_ST 表示 MCPWM0_EVT_F0_CLR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F1_CLR_ST 表示 MCPWM0_EVT_F1_CLR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_F2_CLR_ST 表示 MCPWM0_EVT_F2_CLR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST 表示 MCPWM0_EVT_TZ0_CBC 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST 表示 MCPWM0_EVT_TZ1_CBC 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST 表示 MCPWM0_EVT_TZ2_CBC 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST 表示 MCPWM0_EVT_TZ0_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST 表示 MCPWM0_EVT_TZ1_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST 表示 MCPWM0_EVT_TZ2_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_MCPWM0_EVT_CAP0_ST 表示 MCPWM0_EVT_CAP0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP1_ST 表示 MCPWM0_EVT_CAP1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_CAP2_ST 表示 MCPWM0_EVT_CAP2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST 表示 MCPWM0_EVT_OP0_TEE1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST 表示 MCPWM0_EVT_OP1_TEE1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST 表示 MCPWM0_EVT_OP2_TEE1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST 表示 MCPWM0_EVT_OP0_TEE2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST 表示 MCPWM0_EVT_OP1_TEE2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST 表示 MCPWM0_EVT_OP2_TEE2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_ADC_EVT_CONV_CMPLTO_ST 表示 ADC_EVT_CONV_CMPLTO 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESHO_ST 表示 ADC_EVT_EQ_ABOVE_THRESHO 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST 表示 ADC_EVT_EQ_ABOVE_THRESH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESHO_ST 表示 ADC_EVT_EQ_BELOW_THRESHO 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST 表示 ADC_EVT_EQ_BELOW_THRESH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ADC_EVT_RESULT_DONEO_ST 表示 ADC_EVT_RESULT_DONEO 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

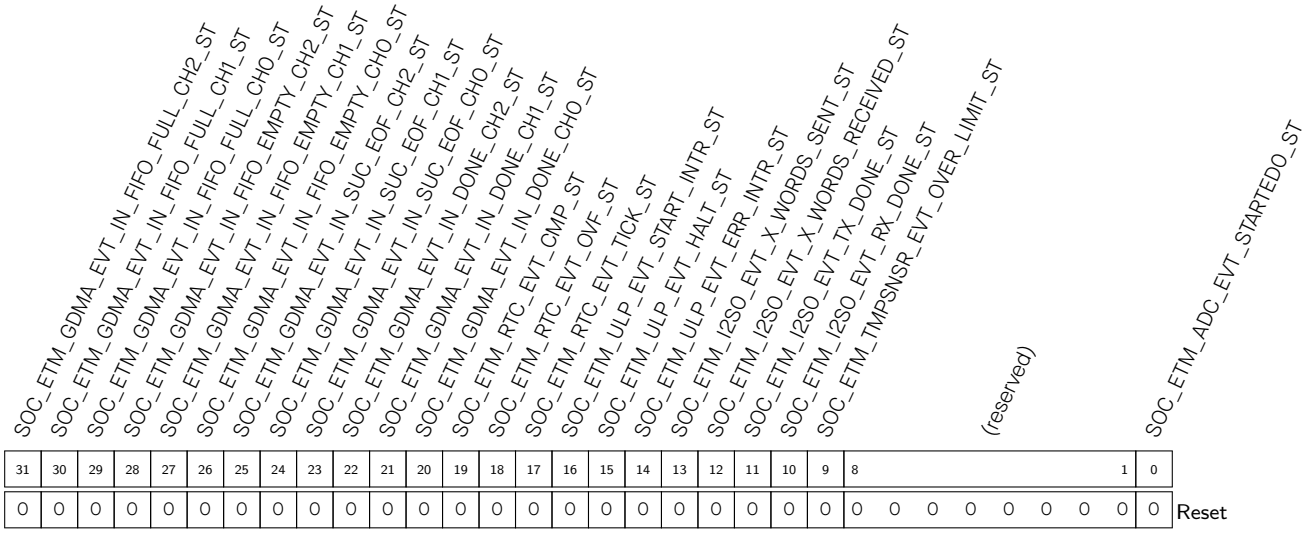
SOC_ETM_ADC_EVT_STOPPEDO_ST 表示 ADC_EVT_STOPPEDO 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.6. SOC_ETM_EVT_ST3_REG (0x01C0)



SOC_ETM_ADC_EVT_STARTEDO_ST 表示 ADC_EVT_STARTEDO 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST 表示 TMPSNSR_EVT_OVER_LIMIT 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_I2SO_EVT_RX_DONE_ST 表示 I2SO_EVT_RX_DONE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_I2SO_EVT_TX_DONE_ST 表示 I2SO_EVT_TX_DONE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_I2SO_EVT_X_WORDS_RECEIVED_ST 表示 I2SO_EVT_X_WORDS_RECEIVED 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_I2SO_EVT_X_WORDS_SENT_ST 表示 I2SO_EVT_X_WORDS_SENT 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ULP_EVT_ERR_INTR_ST 表示 ULP_EVT_ERR_INTR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.6. SOC_ETM_EVT_ST3_REG (0x01C0)

接上页...

SOC_ETM_ULP_EVT_HALT_ST 表示 ULP_EVT_HALT 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ULP_EVT_START_INTR_ST 表示 ULP_EVT_START_INTR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_RTC_EVT_TICK_ST 表示 RTC_EVT_TICK 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_RTC_EVT_OVF_ST 表示 RTC_EVT_OVF 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_RTC_EVT_CMP_ST 表示 RTC_EVT_CMP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_DONE_CH0_ST 表示 GDMA_EVT_IN_DONE_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_DONE_CH1_ST 表示 GDMA_EVT_IN_DONE_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_DONE_CH2_ST 表示 GDMA_EVT_IN_DONE_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH0_ST 表示 GDMA_EVT_IN_SUC_EOF_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.6. SOC_ETM_EVT_ST3_REG (0x01C0)

接上页...

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH1_ST 表示 GDMA_EVT_IN_SUC_EOF_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH2_ST 表示 GDMA_EVT_IN_SUC_EOF_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH0_ST 表示 GDMA_EVT_IN_FIFO_EMPTY_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH1_ST 表示 GDMA_EVT_IN_FIFO_EMPTY_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH2_ST 表示 GDMA_EVT_IN_FIFO_EMPTY_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH0_ST 表示 GDMA_EVT_IN_FIFO_FULL_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH1_ST 表示 GDMA_EVT_IN_FIFO_FULL_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

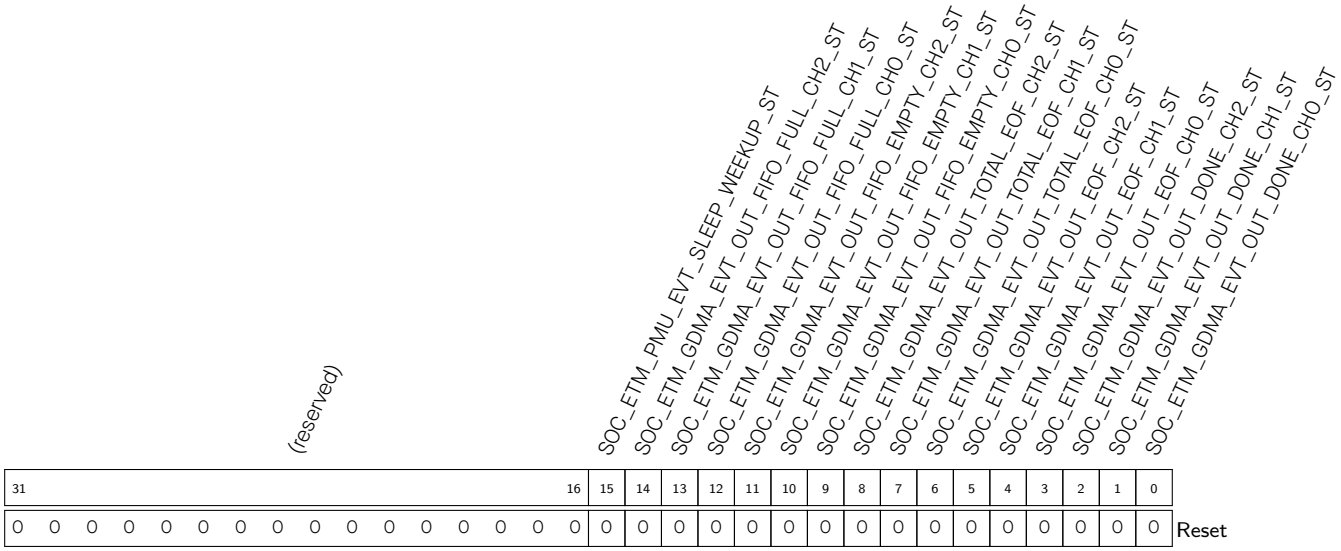
SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH2_ST 表示 GDMA_EVT_IN_FIFO_FULL_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.7. SOC_ETM_EVT_ST4_REG (0x01C8)



SOC_ETM_GDMA_EVT_OUT_DONE_CH0_ST 表示 GDMA_EVT_OUT_DONE_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_DONE_CH1_ST 表示 GDMA_EVT_OUT_DONE_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_DONE_CH2_ST 表示 GDMA_EVT_OUT_DONE_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_EOF_CH0_ST 表示 GDMA_EVT_OUT_EOF_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_EOF_CH1_ST 表示 GDMA_EVT_OUT_EOF_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_EOF_CH2_ST 表示 GDMA_EVT_OUT_EOF_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH0_ST 表示 GDMA_EVT_OUT_TOTAL_EOF_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.7. SOC_ETM_EVT_ST4_REG (0x01C8)

接上页...

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH1_ST 表示 GDMA_EVT_OUT_TOTAL_EOF_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH2_ST 表示 GDMA_EVT_OUT_TOTAL_EOF_CH2 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH0_ST 表示 GDMA_EVT_OUT_FIFO_EMPTY_CH0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH1_ST 表示 GDMA_EVT_OUT_FIFO_EMPTY_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH2_ST 表示 GDMA_EVT_OUT_FIFO_EMPTY_CH2 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH0_ST 表示 GDMA_EVT_OUT_FIFO_FULL_CH0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH1_ST 表示 GDMA_EVT_OUT_FIFO_FULL_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

见下页...

Register 12.7. SOC_ETM_EVT_ST4_REG (0x01C8)

接上页...

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH2_ST 表示 GDMA_EVT_OUT_FIFO_FULL_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

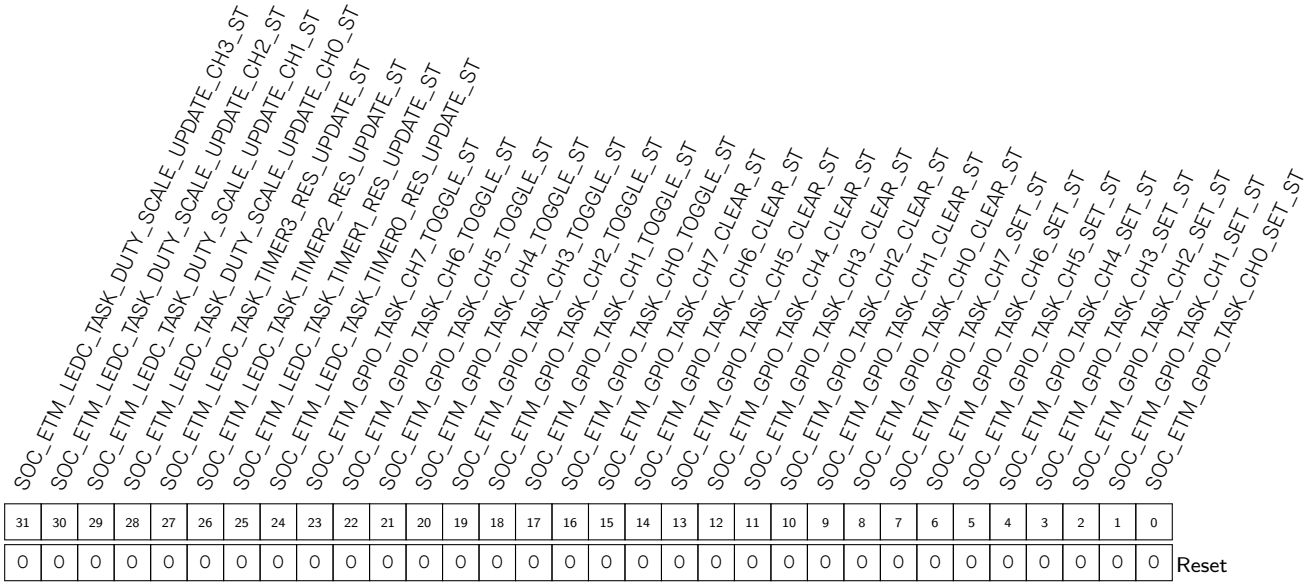
SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST 表示 PMU_EVT_SLEEP_WEEKUP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.8. SOC_ETM_TASK_ST0_REG (0x01D0)



SOC_ETM_GPIO_TASK_CH0_SET_ST 表示 GPIO_TASK_CH0_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_SET_ST 表示 GPIO_TASK_CH1_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_SET_ST 表示 GPIO_TASK_CH2_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_SET_ST 表示 GPIO_TASK_CH3_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_SET_ST 表示 GPIO_TASK_CH4_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_SET_ST 表示 GPIO_TASK_CH5_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_SET_ST 表示 GPIO_TASK_CH6_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.8. SOC_ETM_TASK_ST0_REG (0x01D0)

接上页...

SOC_ETM_GPIO_TASK_CH7_SET_ST 表示 GPIO_TASK_CH7_SET 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST 表示 GPIO_TASK_CH0_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST 表示 GPIO_TASK_CH1_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST 表示 GPIO_TASK_CH2_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST 表示 GPIO_TASK_CH3_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST 表示 GPIO_TASK_CH4_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST 表示 GPIO_TASK_CH5_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST 表示 GPIO_TASK_CH6_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST 表示 GPIO_TASK_CH7_CLEAR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.8. SOC_ETM_TASK_ST0_REG (0x01D0)

接上页...

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST 表示 GPIO_TASK_CH0_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST 表示 GPIO_TASK_CH1_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST 表示 GPIO_TASK_CH2_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST 表示 GPIO_TASK_CH3_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST 表示 GPIO_TASK_CH4_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST 表示 GPIO_TASK_CH5_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST 表示 GPIO_TASK_CH6_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST 表示 GPIO_TASK_CH7_TOGGLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.8. SOC_ETM_TASK_ST0_REG (0x01D0)

接上页...

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST 表示 LEDC_TASK_TIMER0_RES_UPDATE 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST 表示 LEDC_TASK_TIMER1_RES_UPDATE 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST 表示 LEDC_TASK_TIMER2_RES_UPDATE 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST 表示 LEDC_TASK_TIMER3_RES_UPDATE 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

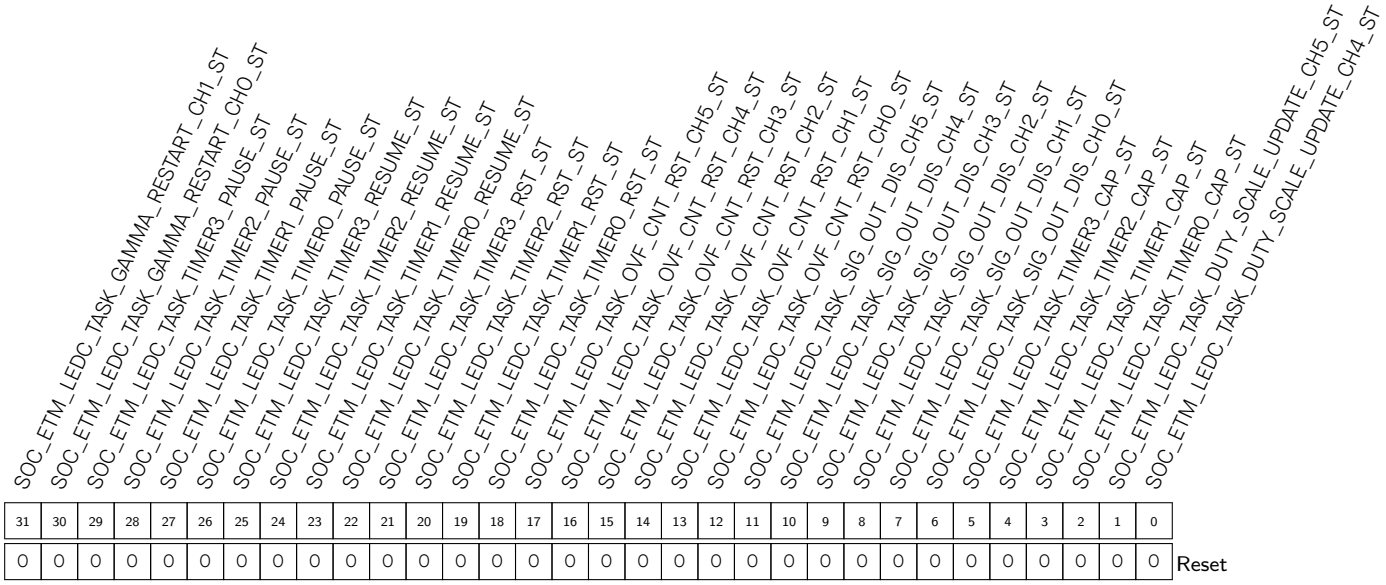
SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH2 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH3 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

Register 12.9. SOC_ETM_TASK_ST1_REG (0x01D8)



SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH4 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH5 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST 表示 LEDC_TASK_TIMER0_CAP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST 表示 LEDC_TASK_TIMER1_CAP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST 表示 LEDC_TASK_TIMER2_CAP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST 表示 LEDC_TASK_TIMER3_CAP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.9. SOC_ETM_TASK_ST1_REG (0x01D8)

接上页...

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH2 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH3 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH4 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH5 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST 表示 LEDC_TASK_OVF_CNT_RST_CH0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

见下页...

Register 12.9. SOC_ETM_TASK_ST1_REG (0x01D8)

接上页...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST 表示 LEDC_TASK_OVF_CNT_RST_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST 表示 LEDC_TASK_OVF_CNT_RST_CH2 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST 表示 LEDC_TASK_OVF_CNT_RST_CH3 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST 表示 LEDC_TASK_OVF_CNT_RST_CH4 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST 表示 LEDC_TASK_OVF_CNT_RST_CH5 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST 表示 LEDC_TASK_TIMER0_RST 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST 表示 LEDC_TASK_TIMER1_RST 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

见下页...

Register 12.9. SOC_ETM_TASK_ST1_REG (0x01D8)

接上页...

SOC_ETM_LEDC_TASK_TIMER2_RST_ST 表示 LEDC_TASK_TIMER2_RST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST 表示 LEDC_TASK_TIMER3_RST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST 表示 LEDC_TASK_TIMER0_RESUME 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST 表示 LEDC_TASK_TIMER1_RESUME 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST 表示 LEDC_TASK_TIMER2_RESUME 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST 表示 LEDC_TASK_TIMER3_RESUME 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.9. SOC_ETM_TASK_ST1_REG (0x01D8)

接上页...

SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST 表示 LEDC_TASK_TIMER0_PAUSE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST 表示 LEDC_TASK_TIMER1_PAUSE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST 表示 LEDC_TASK_TIMER2_PAUSE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST 表示 LEDC_TASK_TIMER3_PAUSE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST 表示 LEDC_TASK_GAMMA_RESTART_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

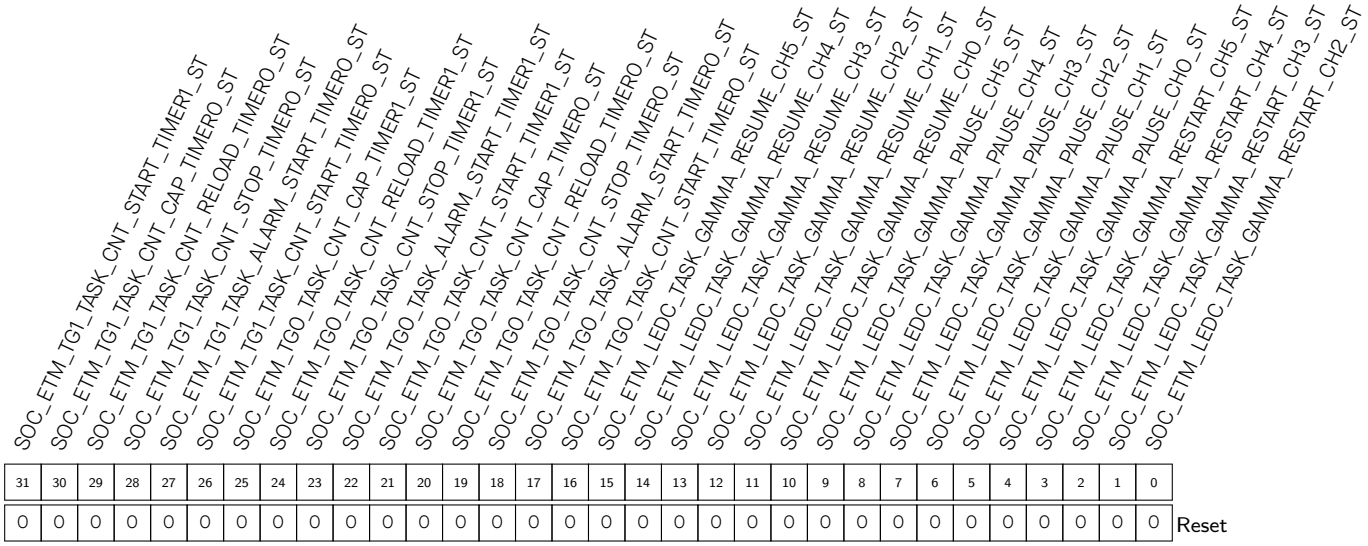
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST 表示 LEDC_TASK_GAMMA_RESTART_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.10. SOC_ETM_TASK_ST2_REG (0x01E0)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST 表示 LEDC_TASK_GAMMA_RESTART_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST 表示 LEDC_TASK_GAMMA_RESTART_CH3 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST 表示 LEDC_TASK_GAMMA_RESTART_CH4 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST 表示 LEDC_TASK_GAMMA_RESTART_CH5 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.10. SOC_ETM_TASK_ST2_REG (0x01E0)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH2 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH3 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH4 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH5 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST 表示 LEDC_TASK_GAMMA_RESUME_CH0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST 表示 LEDC_TASK_GAMMA_RESUME_CH1 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

见下页...

Register 12.10. SOC_ETM_TASK_ST2_REG (0x01E0)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST 表示 LEDC_TASK_GAMMA_RESUME_CH2 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST 表示 LEDC_TASK_GAMMA_RESUME_CH3 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST 表示 LEDC_TASK_GAMMA_RESUME_CH4 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST 表示 LEDC_TASK_GAMMA_RESUME_CH5 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST 表示 TGO_TASK_CNT_START_TIMER0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST 表示 TGO_TASK_ALARM_START_TIMER0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST 表示 TGO_TASK_CNT_STOP_TIMER0 的触发状态。
0：未触发
1：已触发
(R/WTC/SS)

见下页...

Register 12.10. SOC_ETM_TASK_ST2_REG (0x01E0)

接上页...

SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER0_ST 表示 TG0_TASK_CNT_RELOAD_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST 表示 TG0_TASK_CNT_CAP_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST 表示 TG0_TASK_CNT_START_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST 表示 TG0_TASK_ALARM_START_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_STOP_TIMER1_ST 表示 TG0_TASK_CNT_STOP_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST 表示 TG0_TASK_CNT_RELOAD_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST 表示 TG0_TASK_CNT_CAP_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST 表示 TG1_TASK_CNT_START_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.10. SOC_ETM_TASK_ST2_REG (0x01E0)

接上页...

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST 表示 TG1_TASK_ALARM_START_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST 表示 TG1_TASK_CNT_STOP_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST 表示 TG1_TASK_CNT_RELOAD_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST 表示 TG1_TASK_CNT_CAP_TIMER0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

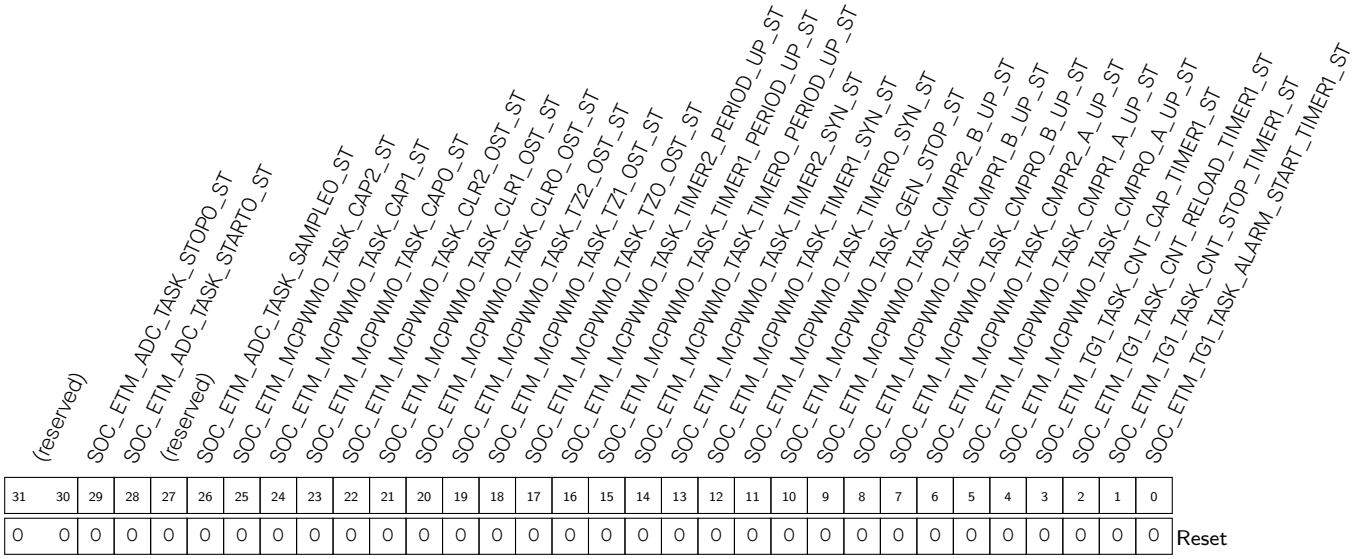
SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST 表示 TG1_TASK_CNT_START_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.11. SOC_ETM_TASK_ST3_REG (0x01E8)



SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST 表示 TG1_TASK_ALARM_START_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST 表示 TG1_TASK_CNT_STOP_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST 表示 TG1_TASK_CNT_RELOAD_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST 表示 TG1_TASK_CNT_CAP_TIMER1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST 表示 MCPWM0_TASK_CMPR0_A_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST 表示 MCPWM0_TASK_CMPR1_A_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.11. SOC_ETM_TASK_ST3_REG (0x01E8)

接上页...

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST 表示 MCPWM0_TASK_CMPR2_A_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST 表示 MCPWM0_TASK_CMPR0_B_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST 表示 MCPWM0_TASK_CMPR1_B_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST 表示 MCPWM0_TASK_CMPR2_B_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST 表示 MCPWM0_TASK_GEN_STOP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST 表示 MCPWM0_TASK_TIMER0_SYN 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST 表示 MCPWM0_TASK_TIMER1_SYN 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST 表示 MCPWM0_TASK_TIMER2_SYN 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.11. SOC_ETM_TASK_ST3_REG (0x01E8)

接上页...

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST 表示 MCPWM0_TASK_TIMER0_PERIOD_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST 表示 MCPWM0_TASK_TIMER1_PERIOD_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST 表示 MCPWM0_TASK_TIMER2_PERIOD_UP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST 表示 MCPWM0_TASK_TZ0_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST 表示 MCPWM0_TASK_TZ1_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_TZ2_OST_ST 表示 MCPWM0_TASK_TZ2_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR0_OST_ST 表示 MCPWM0_TASK_CLR0_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST 表示 MCPWM0_TASK_CLR1_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.11. SOC_ETM_TASK_ST3_REG (0x01E8)

接上页...

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST 表示 MCPWM0_TASK_CLR2_OST 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP0_ST 表示 MCPWM0_TASK_CAP0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP1_ST 表示 MCPWM0_TASK_CAP1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CAP2_ST 表示 MCPWM0_TASK_CAP2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ADC_TASK_SAMPLE0_ST 表示 ADC_TASK_SAMPLE0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ADC_TASK_START0_ST 表示 ADC_TASK_START0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

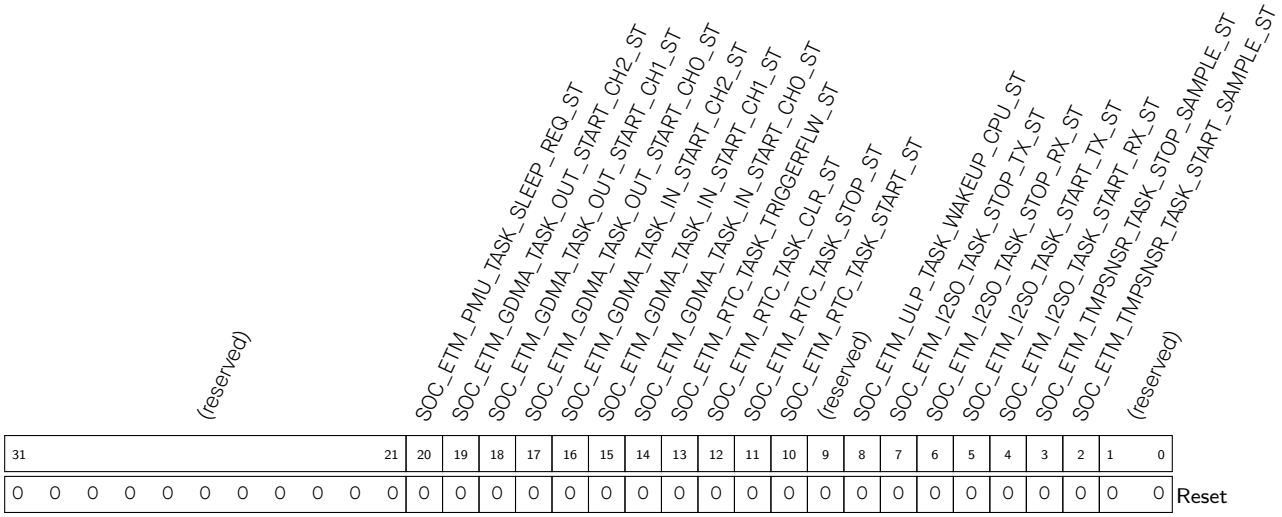
SOC_ETM_ADC_TASK_STOP0_ST 表示 ADC_TASK_STOP0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.12. SOC_ETM_TASK_ST4_REG (0x01F0)



SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST 表示 TMPSNSR_TASK_START_SAMPLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST 表示 TMPSNSR_TASK_STOP_SAMPLE 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_I2S0_TASK_START_RX_ST 表示 I2S0_TASK_START_RX 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_I2S0_TASK_START_TX_ST 表示 I2S0_TASK_START_TX 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.12. SOC_ETM_TASK_ST4_REG (0x01F0)

接上页...

SOC_ETM_I2S0_TASK_STOP_RX_ST 表示 I2S0_TASK_STOP_RX 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_I2S0_TASK_STOP_TX_ST 表示 I2S0_TASK_STOP_TX 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST 表示 ULP_TASK_WAKEUP_CPU 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_RTC_TASK_START_ST 表示 RTC_TASK_START 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_RTC_TASK_STOP_ST 表示 RTC_TASK_STOP 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_RTC_TASK_CLR_ST 表示 RTC_TASK_CLR 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST 表示 RTC_TASK_TRIGGERFLW 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

见下页...

Register 12.12. SOC_ETM_TASK_ST4_REG (0x01F0)

接上页...

SOC_ETM_GDMA_TASK_IN_START_CH0_ST 表示 GDMA_TASK_IN_START_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_TASK_IN_START_CH1_ST 表示 GDMA_TASK_IN_START_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_TASK_IN_START_CH2_ST 表示 GDMA_TASK_IN_START_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_TASK_OUT_START_CH0_ST 表示 GDMA_TASK_OUT_START_CH0 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_TASK_OUT_START_CH1_ST 表示 GDMA_TASK_OUT_START_CH1 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

SOC_ETM_GDMA_TASK_OUT_START_CH2_ST 表示 GDMA_TASK_OUT_START_CH2 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

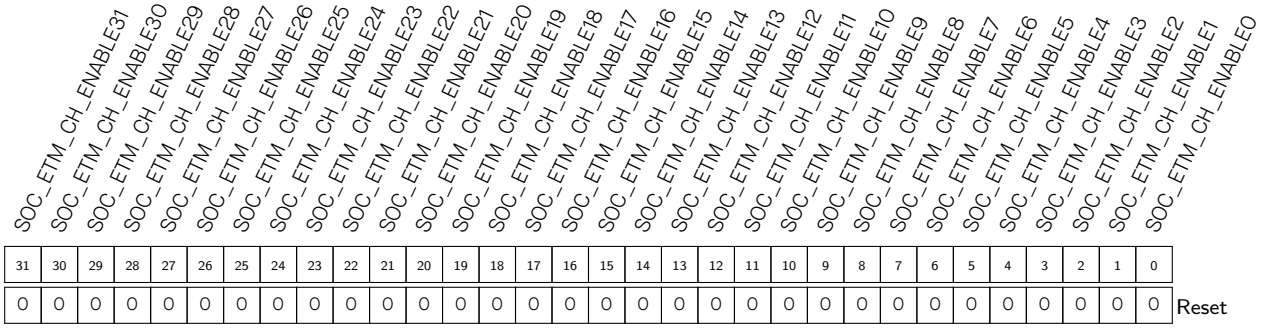
SOC_ETM_PMU_TASK_SLEEP_REQ_ST 表示 PMU_TASK_SLEEP_REQ 的触发状态。

0：未触发

1：已触发

(R/WTC/SS)

Register 12.13. SOC_ETM_CH_ENA_AD0_SET_REG (0x0004)



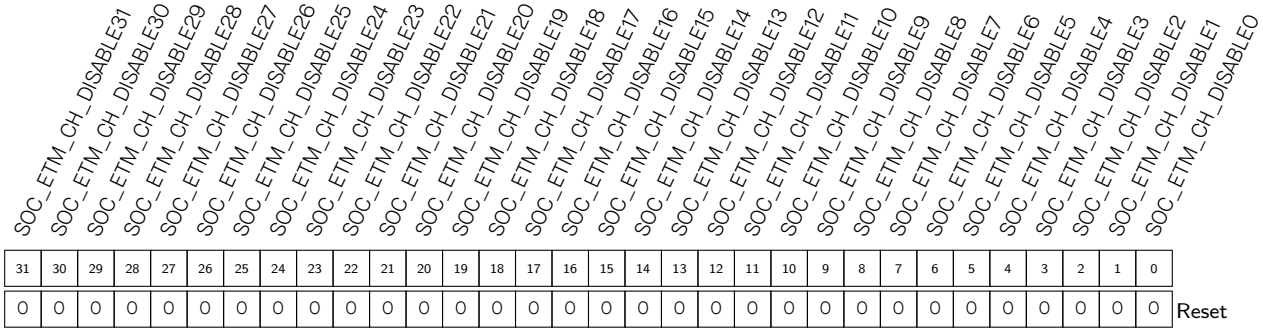
SOC_ETM_CH_ENABLE*n* (*n*: 0-31) 配置是否使能通道 *n*。

0：无效值，没有作用

1：使能

(WT)

Register 12.14. SOC_ETM_CH_ENA_AD0_CLR_REG (0x0008)



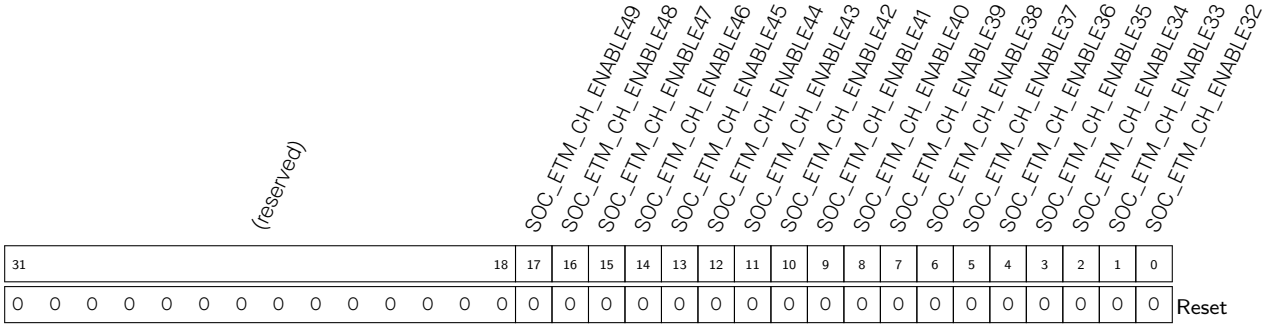
SOC_ETM_CH_DISABLE*n* (*n*: 0-31) 配置是否关闭通道 *n*。

0：无效值，没有作用

1：关闭

(WT)

Register 12.15. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)



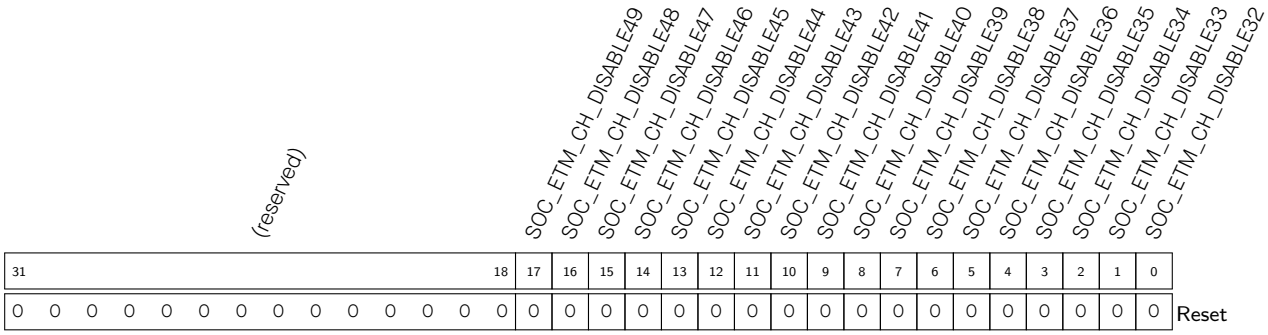
SOC_ETM_CH_ENABLE*n* (*n*: 32-49) 配置是否使能通道 *n*。

0：无效值，没有作用

1：使能

(WT)

Register 12.16. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)



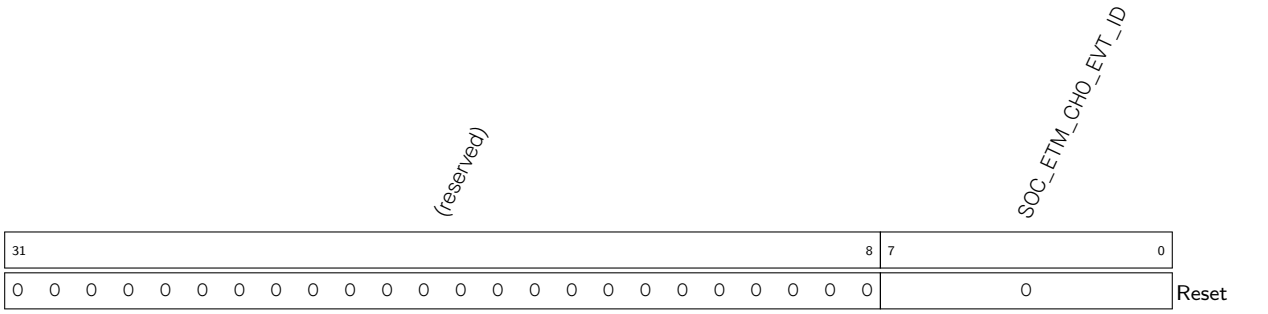
SOC_ETM_CH_DISABLE*n* (*n*: 32-49) 配置是否关闭通道 *n*。

0：无效值，没有作用

1：清除

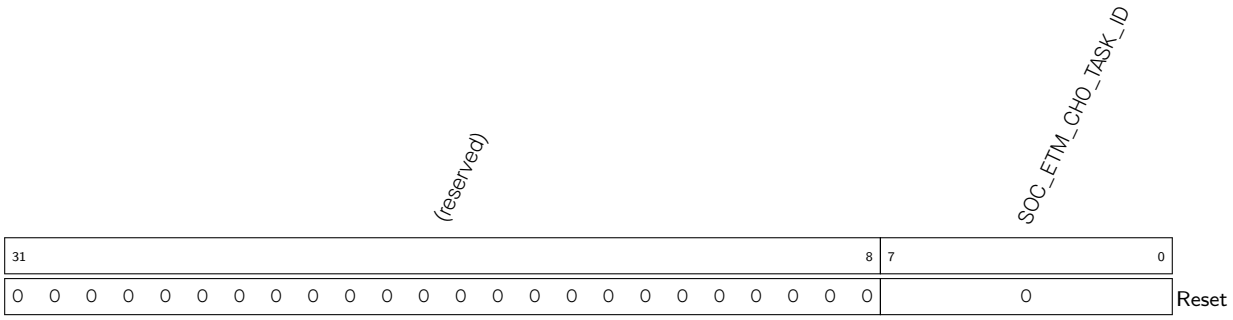
(WT)

Register 12.17. SOC_ETM_CH*n*_EVT_ID_REG (*n*: 0-49) (0x0018+0x8**n*)



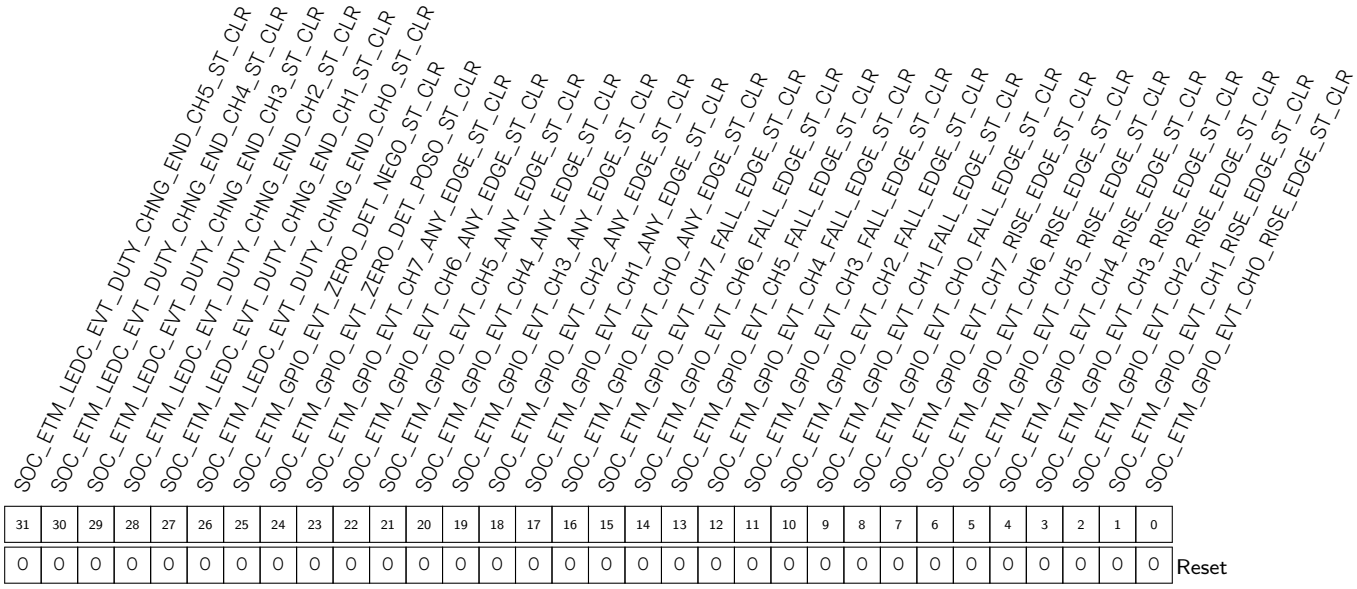
SOC_ETM_CH*n*_EVT_ID 配置通道 *n* 的事件 ID。(R/W)

Register 12.18. SOC_ETM_CH*n*_TASK_ID_REG (*n*: 0-49) (0x001C+0x8**n*)



SOC_ETM_CH*n*_TASK_ID 配置通道 *n* 的任务 ID。(R/W)

Register 12.19. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)



SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH0_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH1_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH2_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH3_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH4_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.19. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH5_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH6_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH7_RISE_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH0_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH1_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH2_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH3_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.19. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH4_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH5_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH6_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH7_FALL_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH0_ANY_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH1_ANY_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH2_ANY_EDGE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.19. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH3_ANY_EDGE 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH4_ANY_EDGE 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH5_ANY_EDGE 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH6_ANY_EDGE 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH7_ANY_EDGE 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST_CLR 配置是否清除 GPIO_EVT_ZERO_DET_POS0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST_CLR 配置是否清除 GPIO_EVT_ZERO_DET_NEG0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 12.19. SOC_ETM_EVT_ST0_CLR_REG (0x01AC)

接上页...

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST_CLR 配 置 是 否 清 除
LEDC_EVT_DUTY_CHNG_END_CH0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST_CLR 配 置 是 否 清 除
LEDC_EVT_DUTY_CHNG_END_CH1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

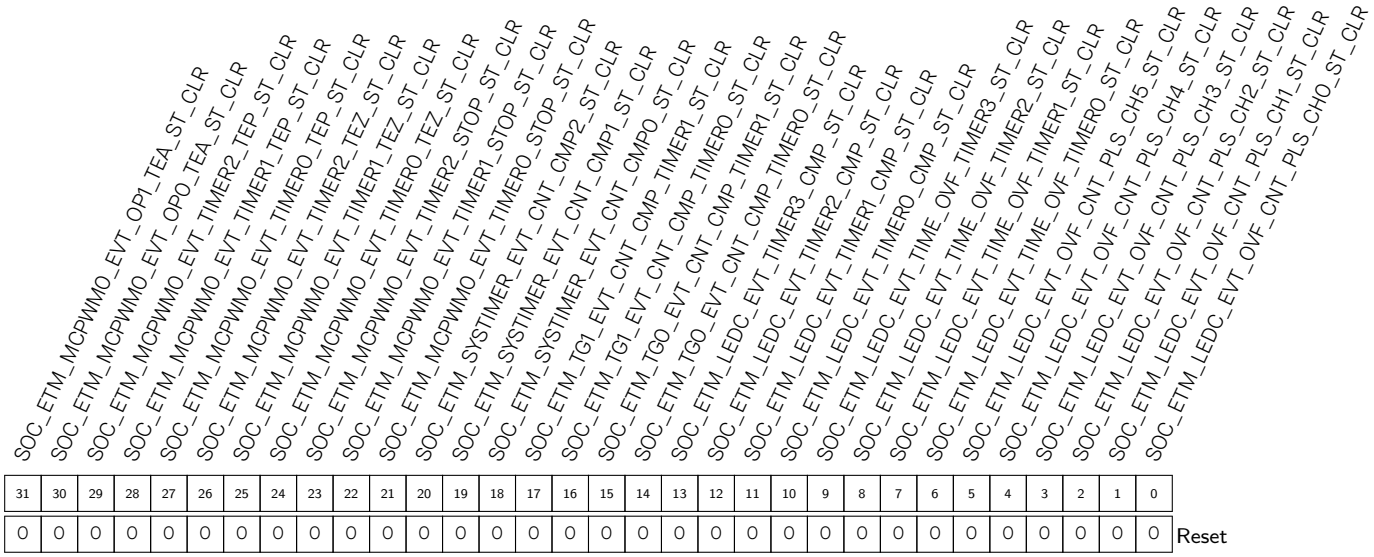
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST_CLR 配 置 是 否 清 除
LEDC_EVT_DUTY_CHNG_END_CH2 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST_CLR 配 置 是 否 清 除
LEDC_EVT_DUTY_CHNG_END_CH3 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST_CLR 配 置 是 否 清 除
LEDC_EVT_DUTY_CHNG_END_CH4 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST_CLR 配 置 是 否 清 除
LEDC_EVT_DUTY_CHNG_END_CH5 的触发状态。
0：无效值，没有作用
1：清除
(WT)

Register 12.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)



SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH4 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST_CLR 配置是否清除 LEDC_EVT_OVF_CNT_PLS_CH5 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER0_CMP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER1_CMP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER2_CMP 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER3_CMP 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST_CLR 配置是否清除 TG0_EVT_CNT_CMP_TIMER0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST_CLR 配置是否清除 TG0_EVT_CNT_CMP_TIMER1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST_CLR 配置是否清除 TG1_EVT_CNT_CMP_TIMER0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST_CLR 配置是否清除 TG1_EVT_CNT_CMP_TIMER1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP0_ST_CLR 配置是否清除 SYSTIMER_EVT_CNT_CMP0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 12.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST_CLR 配置是否清除 SYSTIMER_EVT_CNT_CMP1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST_CLR 配置是否清除 SYSTIMER_EVT_CNT_CMP2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_STOP_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER0_STOP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_STOP_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER1_STOP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_STOP_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER2_STOP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEZ_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER0_TEZ 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_TEZ_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER1_TEZ 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_MCPWM0_EVT_TIMER2_TEZ_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER2_TEZ 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER0_TEP_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER0_TEP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER1_TEP_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER1_TEP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TIMER2_TEP_ST_CLR 配置是否清除 MCPWM0_EVT_TIMER2_TEP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEA_ST_CLR 配置是否清除 MCPWM0_EVT_OP0_TEA 的触发状态。

0：无效值，没有作用

1：清除

(WT)

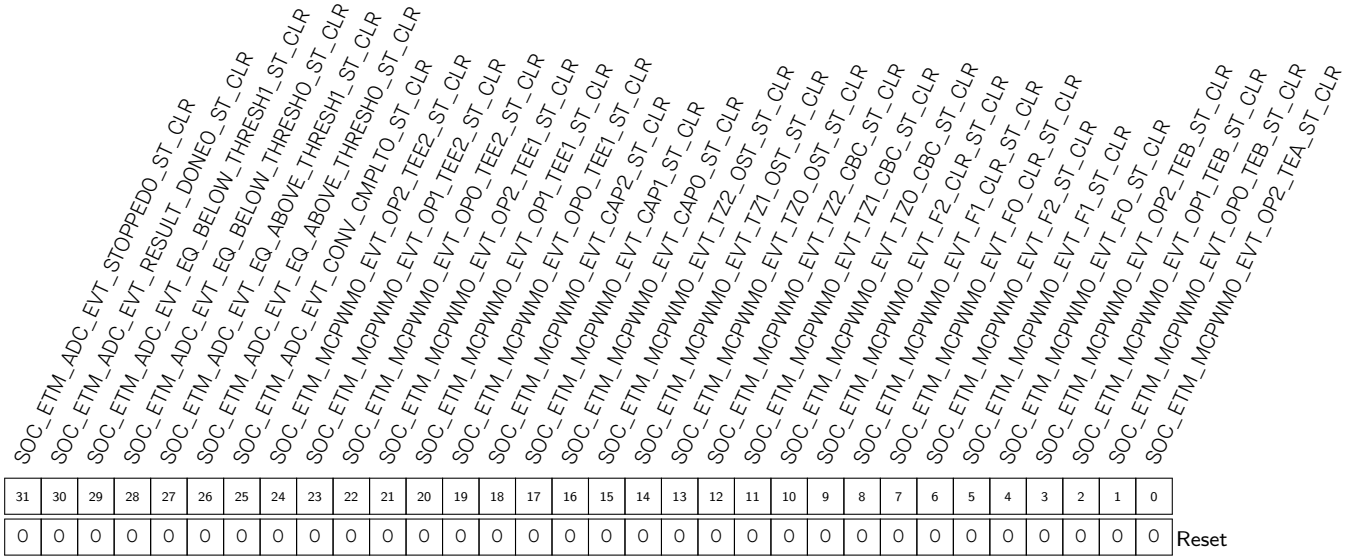
SOC_ETM_MCPWM0_EVT_OP1_TEA_ST_CLR 配置是否清除 MCPWM0_EVT_OP1_TEA 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_MCPWM0_EVT_OP2_TEA_ST_CLR 配置是否清除 MCPWM0_EVT_OP2_TEA 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEB_ST_CLR 配置是否清除 MCPWM0_EVT_OP0_TEB 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEB_ST_CLR 配置是否清除 MCPWM0_EVT_OP1_TEB 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEB_ST_CLR 配置是否清除 MCPWM0_EVT_OP2_TEB 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F0_ST_CLR 配置是否清除 MCPWM0_EVT_F0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWM0_EVT_F1_ST_CLR 配置是否清除 MCPWM0_EVT_F1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F2_ST_CLR 配置是否清除 MCPWM0_EVT_F2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F0_CLR_ST_CLR 配置是否清除 MCPWM0_EVT_F0_CLR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F1_CLR_ST_CLR 配置是否清除 MCPWM0_EVT_F1_CLR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_F2_CLR_ST_CLR 配置是否清除 MCPWM0_EVT_F2_CLR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ0_CBC_ST_CLR 配置是否清除 MCPWM0_EVT_TZ0_CBC 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ1_CBC_ST_CLR 配置是否清除 MCPWM0_EVT_TZ1_CBC 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_TZ2_CBC_ST_CLR 配置是否清除 MCPWM0_EVT_TZ2_CBC 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWM0_EVT_TZ0_OST_ST_CLR 配置是否清除 MCPWM0_EVT_TZ0_OST 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_EVT_TZ1_OST_ST_CLR 配置是否清除 MCPWM0_EVT_TZ1_OST 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_EVT_TZ2_OST_ST_CLR 配置是否清除 MCPWM0_EVT_TZ2_OST 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_EVT_CAP0_ST_CLR 配置是否清除 MCPWM0_EVT_CAP0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_EVT_CAP1_ST_CLR 配置是否清除 MCPWM0_EVT_CAP1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_EVT_CAP2_ST_CLR 配置是否清除 MCPWM0_EVT_CAP2 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE1_ST_CLR 配置是否清除 MCPWM0_EVT_OP0_TEE1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE1_ST_CLR 配置是否清除 MCPWM0_EVT_OP1_TEE1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 12.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWM0_EVT_OP2_TEE1_ST_CLR 配置是否清除 MCPWM0_EVT_OP2_TEE1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP0_TEE2_ST_CLR 配置是否清除 MCPWM0_EVT_OP0_TEE2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP1_TEE2_ST_CLR 配置是否清除 MCPWM0_EVT_OP1_TEE2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_EVT_OP2_TEE2_ST_CLR 配置是否清除 MCPWM0_EVT_OP2_TEE2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_CONV_CMPLTO_ST_CLR 配置是否清除 ADC_EVT_CONV_CMPLTO 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST_CLR 配置是否清除 ADC_EVT_EQ_ABOVE_THRESH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST_CLR 配置是否清除 ADC_EVT_EQ_ABOVE_THRESH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_ADC_EVT_EQ_BELOW_THRESHO_ST_CLR 配置是否清除 ADC_EVT_EQ_BELOW_THRESHO 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST_CLR 配置是否清除 ADC_EVT_EQ_BELOW_THRESH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_EVT_RESULT_DONEO_ST_CLR 配置是否清除 ADC_EVT_RESULT_DONEO 的触发状态。

0：无效值，没有作用

1：清除

(WT)

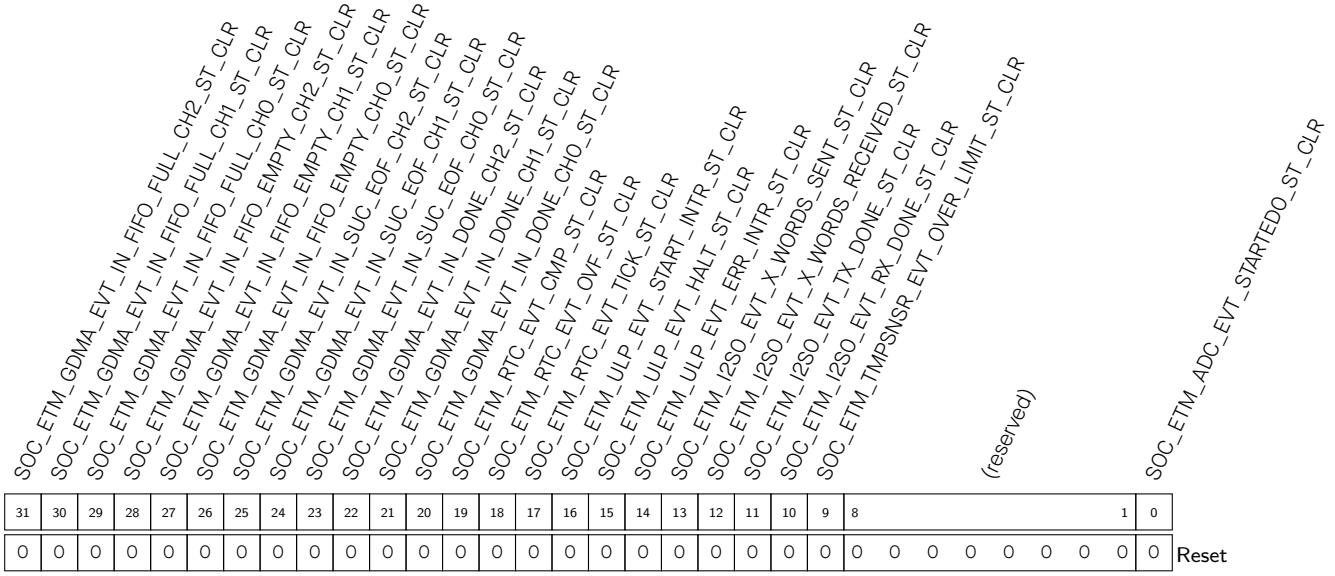
SOC_ETM_ADC_EVT_STOPPEDO_ST_CLR 配置是否清除 ADC_EVT_STOPPEDO 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)



SOC_ETM_ADC_EVT_STARTEDO_ST_CLR 配置是否清除 ADC_EVT_STARTEDO 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST_CLR 配置是否清除 TMPSNSR_EVT_OVER_LIMIT 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2SO_EVT_RX_DONE_ST_CLR 配置是否清除 I2SO_EVT_RX_DONE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2SO_EVT_TX_DONE_ST_CLR 配置是否清除 I2SO_EVT_TX_DONE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2SO_EVT_X_WORDS_RECEIVED_ST_CLR 配置是否清除 I2SO_EVT_X_WORDS_RECEIVED 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2SO_EVT_X_WORDS_SENT_ST_CLR 配置是否清除 I2SO_EVT_X_WORDS_SENT 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_ULP_EVT_ERR_INTR_ST_CLR 配置是否清除 ULP_EVT_ERR_INTR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ULP_EVT_HALT_ST_CLR 配置是否清除 ULP_EVT_HALT 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ULP_EVT_START_INTR_ST_CLR 配置是否清除 ULP_EVT_START_INTR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_EVT_TICK_ST_CLR 配置是否清除 RTC_EVT_TICK 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_EVT_OVF_ST_CLR 配置是否清除 RTC_EVT_OVF 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_EVT_CMP_ST_CLR 配置是否清除 RTC_EVT_CMP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_DONE_CH0_ST_CLR 配置是否清除 GDMA_EVT_IN_DONE_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_DONE_CH1_ST_CLR 配置是否清除 GDMA_EVT_IN_DONE_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_DONE_CH2_ST_CLR 配置是否清除 GDMA_EVT_IN_DONE_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH0_ST_CLR 配置是否清除 GDMA_EVT_IN_SUC_EOF_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH1_ST_CLR 配置是否清除 GDMA_EVT_IN_SUC_EOF_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH2_ST_CLR 配置是否清除 GDMA_EVT_IN_SUC_EOF_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH0_ST_CLR 配置是否清除 GDMA_EVT_IN_FIFO_EMPTY_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH1_ST_CLR 配置是否清除 GDMA_EVT_IN_FIFO_EMPTY_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH2_ST_CLR 配置是否清除 GDMA_EVT_IN_FIFO_EMPTY_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH0_ST_CLR 配置是否清除 GDMA_EVT_IN_FIFO_FULL_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH1_ST_CLR 配置是否清除 GDMA_EVT_IN_FIFO_FULL_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

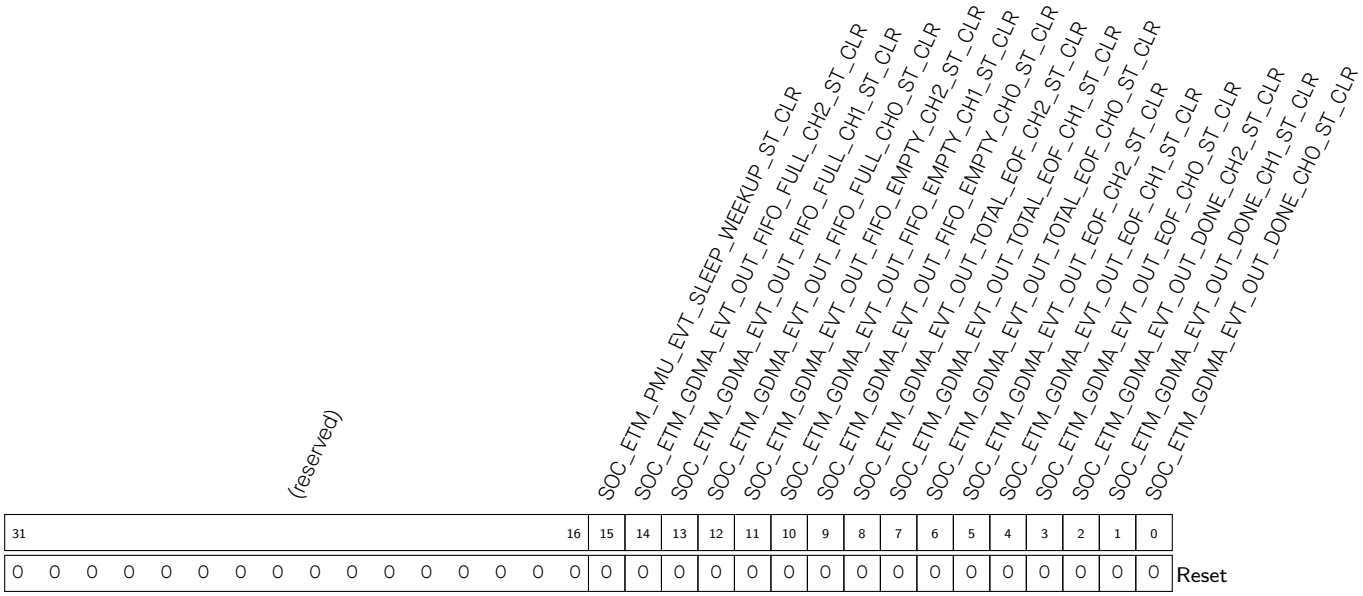
SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH2_ST_CLR 配置是否清除 GDMA_EVT_IN_FIFO_FULL_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.23. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)



SOC_ETM_GDMA_EVT_OUT_DONE_CH0_ST_CLR 配置是否清除 GDMA_EVT_OUT_DONE_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_OUT_DONE_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_DONE_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_OUT_DONE_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_DONE_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_OUT_EOF_CH0_ST_CLR 配置是否清除 GDMA_EVT_OUT_EOF_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_OUT_EOF_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_EOF_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.23. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

接上页...

SOC_ETM_GDMA_EVT_OUT_EOF_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_EOF_CH2 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH0_ST_CLR 配置是否清除 GDMA_EVT_OUT_TOTAL_EOF_CH0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_TOTAL_EOF_CH1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_TOTAL_EOF_CH2 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH0_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_EMPTY_CH0 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_EMPTY_CH1 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_EMPTY_CH2 的触发状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 12.23. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

接上页...

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH0_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_FULL_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_FULL_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_FULL_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

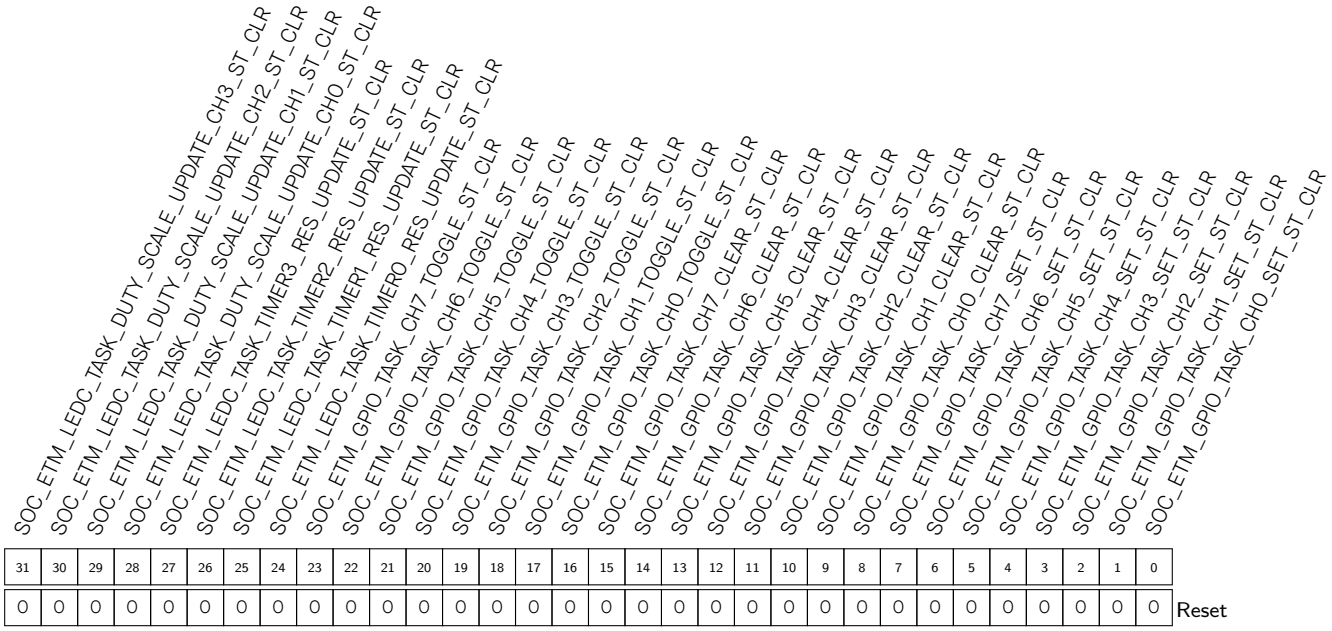
SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST_CLR 配置是否清除 PMU_EVT_SLEEP_WEEKUP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.24. SOC_ETM_TASK_ST0_CLR_REG (0x01D4)



SOC_ETM_GPIO_TASK_CH0_SET_ST_CLR 配置是否清除 GPIO_TASK_CH0_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH1_SET_ST_CLR 配置是否清除 GPIO_TASK_CH1_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH2_SET_ST_CLR 配置是否清除 GPIO_TASK_CH2_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH3_SET_ST_CLR 配置是否清除 GPIO_TASK_CH3_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH4_SET_ST_CLR 配置是否清除 GPIO_TASK_CH4_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH5_SET_ST_CLR 配置是否清除 GPIO_TASK_CH5_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.24. SOC_ETM_TASK_ST0_CLR_REG (0x01D4)

接上页...

SOC_ETM_GPIO_TASK_CH6_SET_ST_CLR 配置是否清除 GPIO_TASK_CH6_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH7_SET_ST_CLR 配置是否清除 GPIO_TASK_CH7_SET 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH0_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH1_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH2_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH3_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH4_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH5_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.24. SOC_ETM_TASK_ST0_CLR_REG (0x01D4)

接上页...

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH6_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH7_CLEAR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH0_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH1_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH2_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH3_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH4_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH5_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.24. SOC_ETM_TASK_ST0_CLR_REG (0x01D4)

接上页...

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH6_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH7_TOGGLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_RES_UPDATE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_RES_UPDATE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_RES_UPDATE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_RES_UPDATE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH0_ST_CLR 配置是否清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.24. SOC_ETM_TASK_ST0_CLR_REG (0x01D4)

接上页...

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST_CLR 配 置 是 否 清 除 LEDC_TASK_DUTY_SCALE_UPDATE_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST_CLR 配 置 是 否 清 除 LEDC_TASK_DUTY_SCALE_UPDATE_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

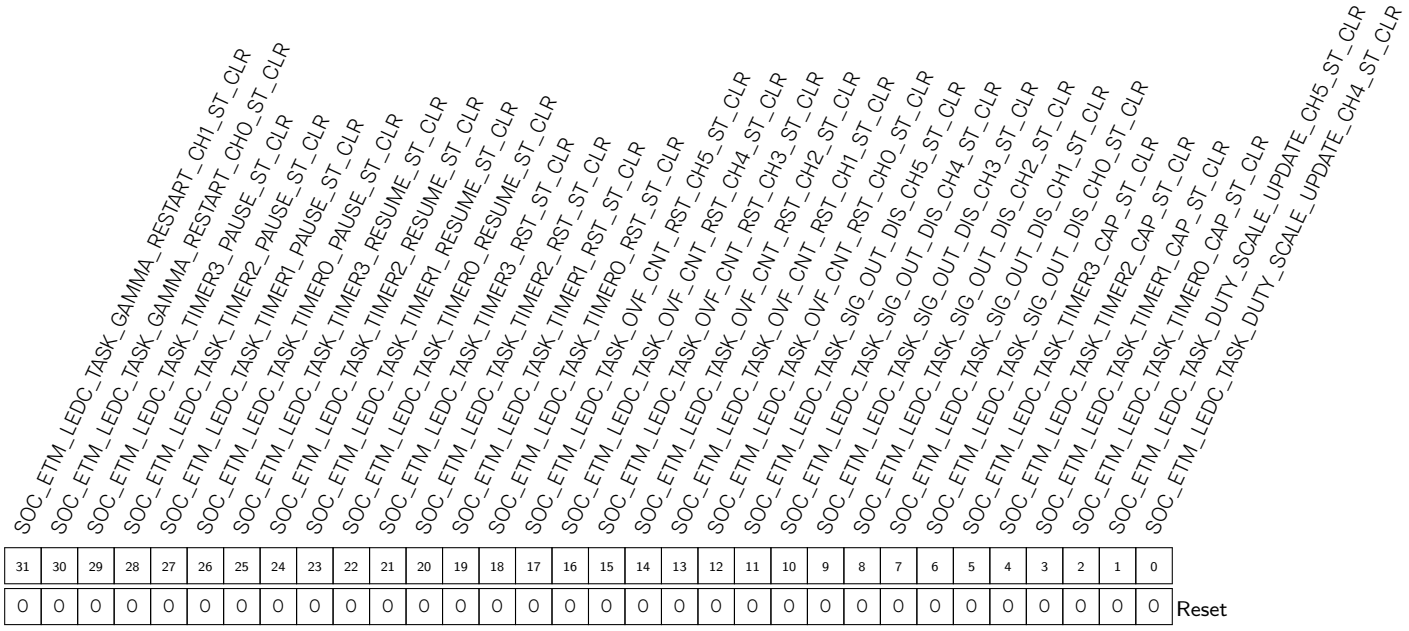
SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST_CLR 配 置 是 否 清 除 LEDC_TASK_DUTY_SCALE_UPDATE_CH3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)



SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST_CLR 配置是否清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH4 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST_CLR 配置是否清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH5 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_CAP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_CAP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_CAP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_CAP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH0_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH4 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH5 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH0_ST_CLR 配置是否清除 LEDC_TASK_OVF_CNT_RST_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST_CLR 配置是否清除 LEDC_TASK_OVF_CNT_RST_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST_CLR 配置是否清除 LEDC_TASK_OVF_CNT_RST_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST_CLR 配置是否清除 LEDC_TASK_OVF_CNT_RST_CH3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST_CLR 配置是否清除 LEDC_TASK_OVF_CNT_RST_CH4 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST_CLR 配置是否清除 LEDC_TASK_OVF_CNT_RST_CH5 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_RST 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_TIMER1_RST_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_RST 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER2_RST_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_RST 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_RST 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_RESUME 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_RESUME 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_RESUME 的触发状态。
0：无效值，没有作用
1：清除
(WT)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_RESUME 的触发状态。
0：无效值，没有作用
1：清除
(WT)

见下页...

Register 12.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_PAUSE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_PAUSE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_PAUSE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_PAUSE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

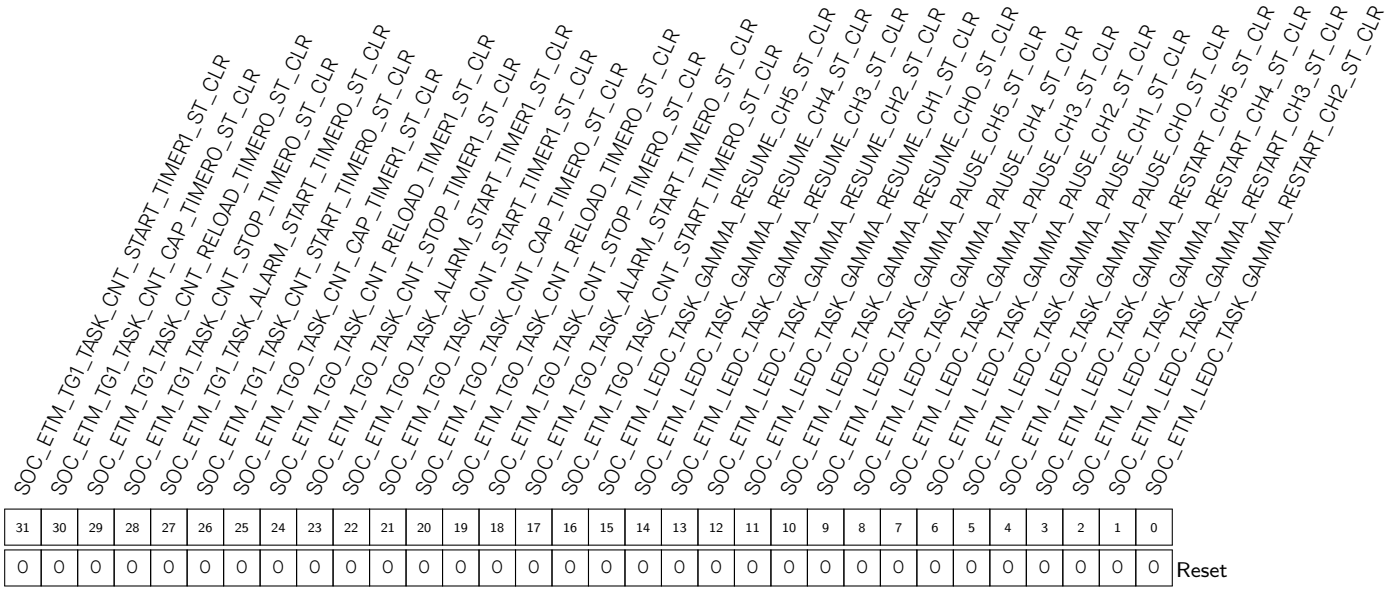
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)



SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CH3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CH4 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CH5 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_PAUSE_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_PAUSE_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_PAUSE_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_PAUSE_CH3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_PAUSE_CH4 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_PAUSE_CH5 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESUME_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESUME_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESUME_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESUME_CH3 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESUME_CH4 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESUME_CH5 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST_CLR 配置是否清除 TGO_TASK_CNT_START_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST_CLR 配置是否清除 TGO_TASK_ALARM_START_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST_CLR 配置是否清除 TGO_TASK_CNT_STOP_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

接上页...

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST_CLR 配置是否清除 TGO_TASK_CNT_RELOAD_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_CAP_TIMER0_ST_CLR 配置是否清除 TGO_TASK_CNT_CAP_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_START_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_START_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_ALARM_START_TIMER1_ST_CLR 配置是否清除 TGO_TASK_ALARM_START_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_STOP_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_RELOAD_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TGO_TASK_CNT_CAP_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_CAP_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

接上页...

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST_CLR 配置是否清除 TG1_TASK_CNT_START_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST_CLR 配置是否清除 TG1_TASK_ALARM_START_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST_CLR 配置是否清除 TG1_TASK_CNT_STOP_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST_CLR 配置是否清除 TG1_TASK_CNT_RELOAD_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST_CLR 配置是否清除 TG1_TASK_CNT_CAP_TIMER0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

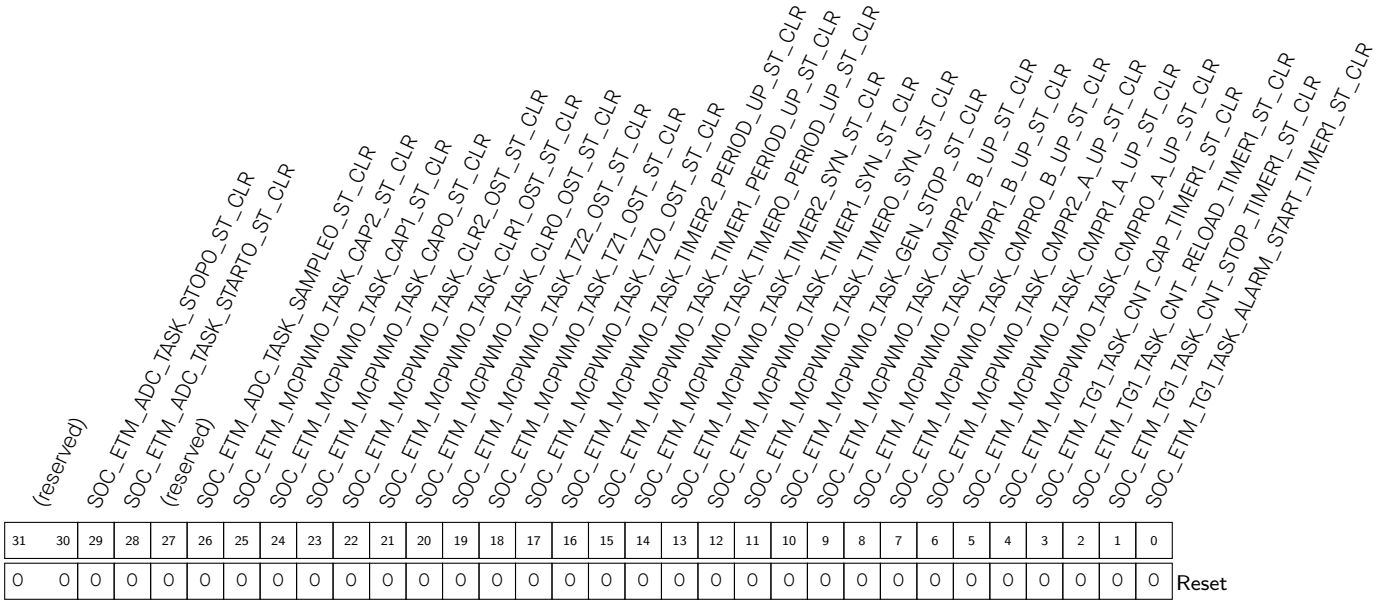
SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_START_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)



SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST_CLR 配置是否清除 TG1_TASK_ALARM_START_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_STOP_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_RELOAD_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_CAP_TIMER1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CMPR0_A_UP_ST_CLR 配置是否清除 MCPWM0_TASK_CMPR0_A_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

接上页...

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST_CLR 配置是否清除 MCPWM0_TASK_CMPR1_A_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CMPR2_A_UP_ST_CLR 配置是否清除 MCPWM0_TASK_CMPR2_A_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CMPR0_B_UP_ST_CLR 配置是否清除 MCPWM0_TASK_CMPR0_B_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CMPR1_B_UP_ST_CLR 配置是否清除 MCPWM0_TASK_CMPR1_B_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CMPR2_B_UP_ST_CLR 配置是否清除 MCPWM0_TASK_CMPR2_B_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_GEN_STOP_ST_CLR 配置是否清除 MCPWM0_TASK_GEN_STOP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER0_SYN_ST_CLR 配置是否清除 MCPWM0_TASK_TIMER0_SYN 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

接上页...

SOC_ETM_MCPWM0_TASK_TIMER1_SYN_ST_CLR 配置是否清除 MCPWM0_TASK_TIMER1_SYN 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER2_SYN_ST_CLR 配置是否清除 MCPWM0_TASK_TIMER2_SYN 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER0_PERIOD_UP_ST_CLR 配置是否清除 MCPWM0_TASK_TIMER0_PERIOD_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER1_PERIOD_UP_ST_CLR 配置是否清除 MCPWM0_TASK_TIMER1_PERIOD_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TIMER2_PERIOD_UP_ST_CLR 配置是否清除 MCPWM0_TASK_TIMER2_PERIOD_UP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TZ0_OST_ST_CLR 配置是否清除 MCPWM0_TASK_TZ0_OST 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_TZ1_OST_ST_CLR 配置是否清除 MCPWM0_TASK_TZ1_OST 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

接上页...

SOC_ETM_MCPWM0_TASK_TZ2_OST_ST_CLR 配置是否清除 MCPWM0_TASK_TZ2_OST 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CLR0_OST_ST_CLR 配置是否清除 MCPWM0_TASK_CLR0_OST 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CLR1_OST_ST_CLR 配置是否清除 MCPWM0_TASK_CLR1_OST 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CLR2_OST_ST_CLR 配置是否清除 MCPWM0_TASK_CLR2_OST 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CAP0_ST_CLR 配置是否清除 MCPWM0_TASK_CAP0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CAP1_ST_CLR 配置是否清除 MCPWM0_TASK_CAP1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_MCPWM0_TASK_CAP2_ST_CLR 配置是否清除 MCPWM0_TASK_CAP2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

接上页...

SOC_ETM_ADC_TASK_SAMPLE0_ST_CLR 配置是否清除 ADC_TASK_SAMPLE0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ADC_TASK_START0_ST_CLR 配置是否清除 ADC_TASK_START0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

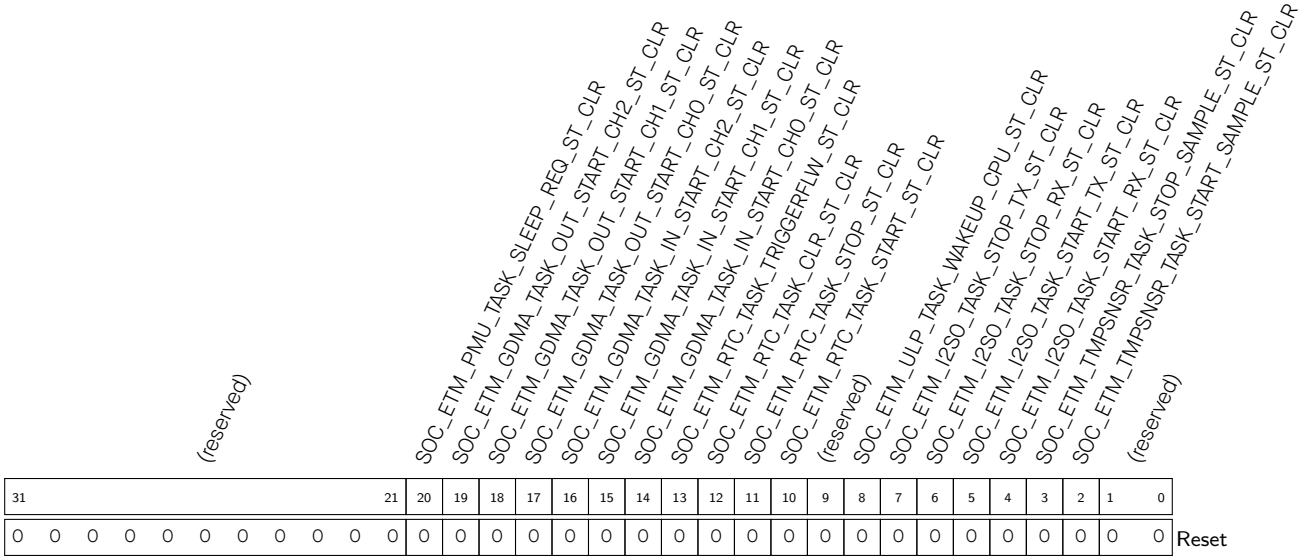
SOC_ETM_ADC_TASK_STOP0_ST_CLR 配置是否清除 ADC_TASK_STOP0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.28. SOC_ETM_TASK_ST4_CLR_REG (0x01F4)



SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST_CLR 配置是否清除 TMPSNSR_TASK_START_SAMPLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST_CLR 配置是否清除 TMPSNSR_TASK_STOP_SAMPLE 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_TASK_START_RX_ST_CLR 配置是否清除 I2S0_TASK_START_RX 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_TASK_START_TX_ST_CLR 配置是否清除 I2S0_TASK_START_TX 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.28. SOC_ETM_TASK_ST4_CLR_REG (0x01F4)

接上页...

SOC_ETM_I2S0_TASK_STOP_RX_ST_CLR 配置是否清除 I2S0_TASK_STOP_RX 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_I2S0_TASK_STOP_TX_ST_CLR 配置是否清除 I2S0_TASK_STOP_TX 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST_CLR 配置是否清除 ULP_TASK_WAKEUP_CPU 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_TASK_START_ST_CLR 配置是否清除 RTC_TASK_START 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_TASK_STOP_ST_CLR 配置是否清除 RTC_TASK_STOP 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_RTC_TASK_CLR_ST_CLR 配置是否清除 RTC_TASK_CLR 的触发状态。

0：无效值，没有作用

1：清除

(WT)

见下页...

Register 12.28. SOC_ETM_TASK_ST4_CLR_REG (0x01F4)

接上页...

SOC_ETM_RTC_TASK_TRIGGERFLW_ST_CLR 配置是否清除 RTC_TASK_TRIGGERFLW 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_TASK_IN_START_CH0_ST_CLR 配置是否清除 GDMA_TASK_IN_START_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_TASK_IN_START_CH1_ST_CLR 配置是否清除 GDMA_TASK_IN_START_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_TASK_IN_START_CH2_ST_CLR 配置是否清除 GDMA_TASK_IN_START_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_TASK_OUT_START_CH0_ST_CLR 配置是否清除 GDMA_TASK_OUT_START_CH0 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_TASK_OUT_START_CH1_ST_CLR 配置是否清除 GDMA_TASK_OUT_START_CH1 的触发状态。

0：无效值，没有作用

1：清除

(WT)

SOC_ETM_GDMA_TASK_OUT_START_CH2_ST_CLR 配置是否清除 GDMA_TASK_OUT_START_CH2 的触发状态。

0：无效值，没有作用

1：清除

(WT)

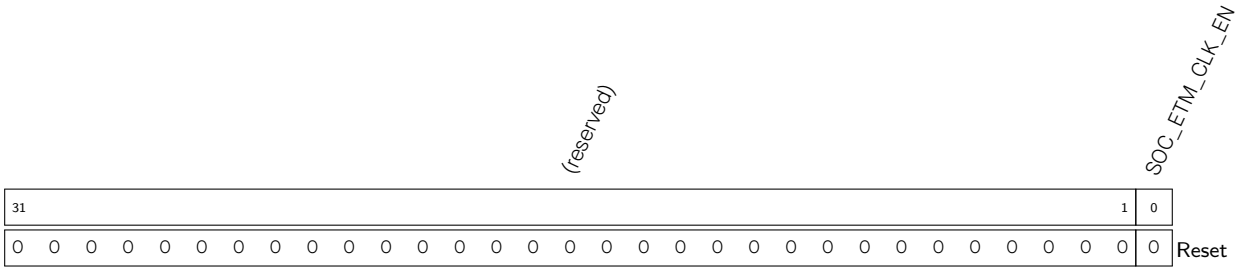
SOC_ETM_PMU_TASK_SLEEP_REQ_ST_CLR 配置是否清除 PMU_TASK_SLEEP_REQ 的触发状态。

0：无效值，没有作用

1：清除

(WT)

Register 12.29. SOC_ETM_CLK_EN_REG (0x01F8)



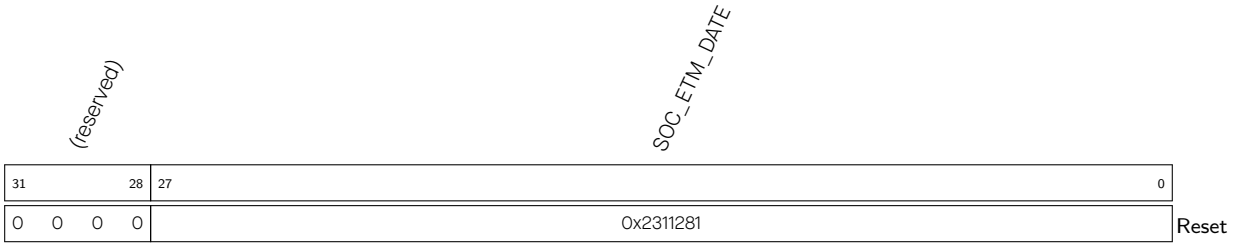
SOC_ETM_CLK_EN 配置寄存器时钟门控。

0：仅在应用写寄存器时开启时钟

1：强制为寄存器开启时钟门控

(R/W)

Register 12.30. SOC_ETM_DATE_REG (0x01FC)



SOC_ETM_DATE 版本控制寄存器。(R/W)

第 13 章

低功耗管理

13.1 概述

ESP32-C5 拥有先进的低功耗管理系统，可以优化设备的功耗，并同时保持高性能。

该系统通过采用多种节能技术，例如睡眠模式、动态电压和频率调节以及外设电源门控等，可以有效地将设备功耗降至最低。

在这一管理系统中，电源管理单元 (PMU) 扮演着核心角色，它可以灵活地控制各个电源域的上下电，以此实现芯片性能、功耗和唤醒时间之间的最佳平衡。

13.2 术语

下列有关低功耗管理的术语是在《ESP32-C5 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

低功耗管理系统	指的是管理芯片功耗的整个系统。
电源管理单元 (PMU)	指的是负责芯片功耗管理的核心控制电路，它包括对芯片电源、时钟以及功耗相关逻辑的控制。
电源域	指的是可独立控制上下电的最小单元。一个电源域可能包含一个或多个模块。
PMU 状态	指的是 PMU 状态机的状态。用户可以配置不同电源域在每一种状态下的时钟开关和电源开关。
功耗模式	指的是预设的典型功耗模式，可满足一些常见场景的功耗需求。

13.3 主要特性

PMU 具有以下特性：

- 支持 4 种可配置 PMU 状态，软件可根据需求灵活配置
  - HP_ACTIVE
  - HP_MODEM
  - HP_SLEEP
  - LP_SLEEP
- 支持 5 种预设功耗模式，可满足应用场景中的多种典型功耗需求
  - Active
  - Modem-sleep

  - Light-sleep0
  - Light-sleep1
  - Deep-sleep
- 16 KB SRAM (LP SRAM)
- 10 个 “always-on” (AON) 寄存器
- RTC 快速启动
- 可灵活配置的备份 DMA，可在芯片切换 PMU 状态时，备份与恢复 CPU 及外设的关键数据
- 功耗控制器，可根据芯片所处的功耗模式切换状态，以控制电源和时钟等

13.4 功能描述

ESP32-C5 的低功耗管理系统包括如下组成部分：

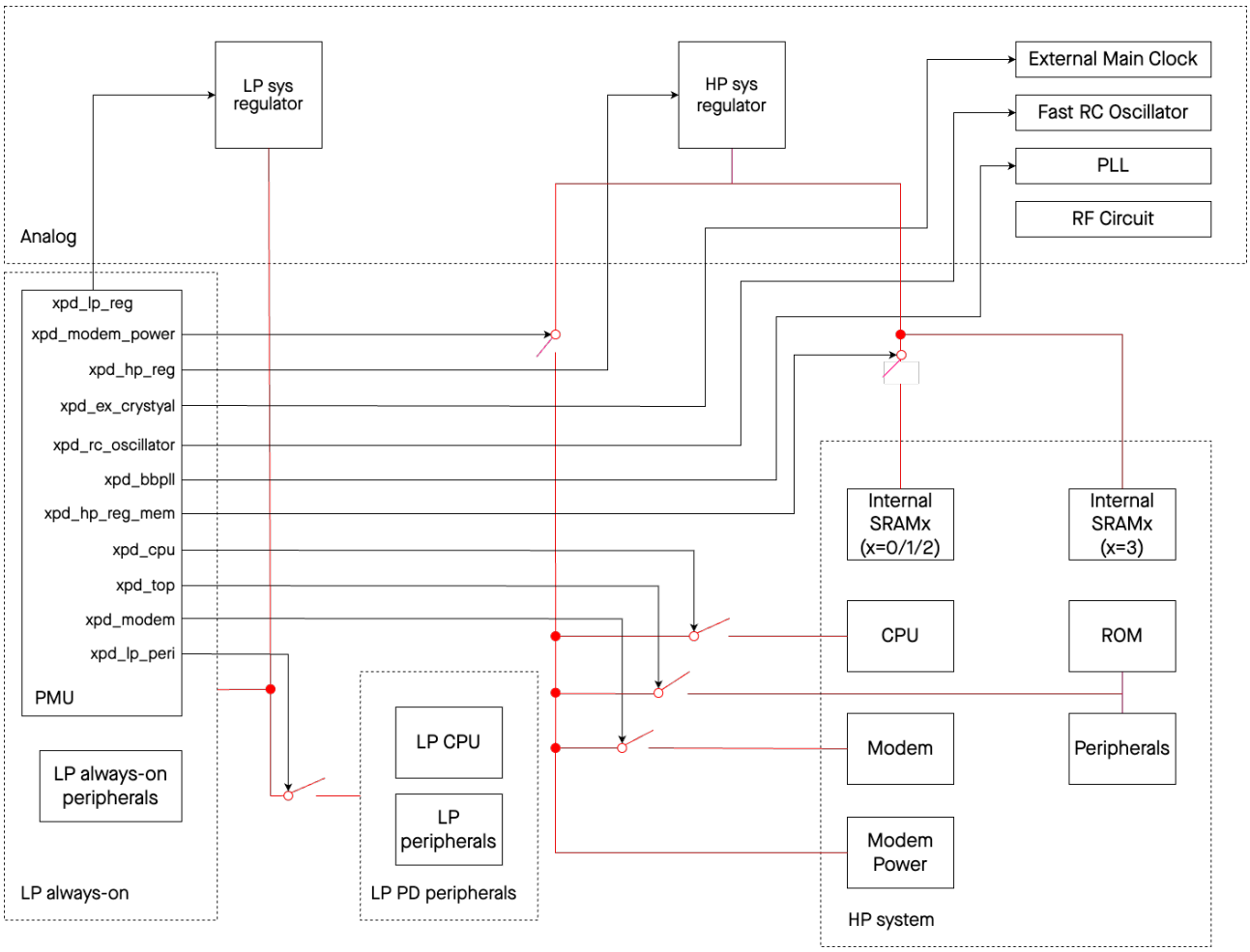
- **电源结构**：即供电管理网络，包括调压器、数字电源域、模拟电源域等部分。
- **PMU 控制器**：PMU 的核心部分，用于实现模拟电源、数字电源、时钟等系统控制。
- **1 个 RTC 定时器**：更多信息，请参考章节 17 实时时钟定时器。
- **10 个 “always-on” 寄存器 (LP_AON_STORE0_REG ~ LP_AON_STORE9_REG)**：这 10 个寄存器永远处于工作状态，不受功耗模式的影响，可用于存储一些关键数据。
- **7 个 LP GPIO 管脚 (GPIO0 ~ GPIO6)**：这 7 个管脚永远处于工作状态，不受功耗模式的影响，可用作睡眠模式下的唤醒源，也可以作为标准的 GPIO 管脚使用。更多细节，请见章节 8 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*。
- **16 KB LP SRAM**：HP CPU 与 LP CPU 均可访问。HP CPU 访问时，该存储器使用 HP CPU 时钟，而在 LP CPU 访问时，使用 LP CPU 时钟。

以下详细介绍这几个组成部分。

13.4.1 电源结构

ESP32-C5 的电源结构如图 13.4-1 所示，主要包含了：

- 2 个调压器
- 模拟电源域
- 数字电源域



—>：控制信号 —：电源线

图 13.4-1. ESP32-C5 电源结构

13.4.1.1 调压器

如图 13.4-1 所示，ESP32-C5 的模拟部分包含 2 个调压器，负责调节不同电源域的供电：

- 1 个 HP（高性能）系统调压器，负责调节高性能模块的供电，驱动能力较强，功耗较大，输出电压可调。
- 1 个 LP（低功耗）系统调压器，负责调节低功耗模块的供电，输出电压同样可调。

13.4.1.2 数字电源域

ESP32-C5 的数字部分主要分为如下几个数字电源域，其中 HP 系统由 HP 系统调压器供电，调压器与每个电源域之间均有一个独立的电源开关，可实现数字电源域的上下电控制；LP 系统由 LP 系统调压器供电。

- HP 系统的电源域包括：
  - **CPU**：主要包括 CPU 及支持其工作的外设（追踪模块等）。
  - **Modem**：包括无线 MAC 和基带
  - **Peripherals + ROM**：主要包括总线、HP 外设等功能模块。
  - **Internal SRAMx**：细分了 4 个子电源域，如图 13.4-1 所示，SRAM0/1/2 每个都有独立的开关控制供电，SRAM3 与调压器直接连接，无开关控制上下电。

  - **Modem Power**：主要包括控制无线部分工作状态的各个模块。
- LP 系统的电源域包括：
  - **LP PD peripherals**：主要包括 LP CPU、LP 外设。
  - **LP always-on**：主要包括 LP always on 外设（如 RTC 定时器）和 PMU 控制模块，所有模式下均不掉电。

13.4.1.3 模拟电源域

HP 系统包括以下模拟电源域：

- PLL

LP 系统包括以下模拟电源域：

- External Main Clock（外部主时钟）
- Fast RC Oscillator（快速 RC 振荡器）
- RF circuit（射频电路）

13.4.2 PMU

ESP32-C5 的 PMU 控制着每个电源域功耗相关的组件，如电源、时钟，它主要由如下几个部分构成：

- **PMU 主状态机**：记录并切换 PMU 状态
- **睡眠与唤醒控制器**：用于向 PMU 主状态机发起睡眠与唤醒请求
- **功耗控制器**：可根据芯片所处的功耗模式切换状态，以控制电源和时钟等。功耗控制器包括：
  - **数字电源控制器**：用于控制数字电源域的上下电
  - **模拟电源控制器**：用于控制模拟模块的使能，比如调压器、模拟时钟等
  - **时钟控制器**：用于管理外设时钟的门控，以及选择数字时钟的模拟时钟源
  - **数据备份控制器**：用于控制 PMU 状态切换时的数据备份与恢复过程
  - **系统控制器**：用于控制部分系统级模块，如睡眠模式下（此时 CPU 不可用）看门狗功能的挂起等

PMU 的工作流程为：

- 睡眠与唤醒控制器向 PMU 主状态机发送睡眠或唤醒请求。
- PMU 主状态机接着产生电源门控、时钟门控和复位信号。
- 电源控制器和时钟控制器会根据 PMU 主状态机产生的信号，打开或关闭不同的电源域和时钟信号，从而让芯片进入或退出各种低功耗模式。

PMU 的工作流程如图 13.4-2 所示。其中，HP_SWITCH 和 LP_SWITCH 的定义如下：

- HP_SWITCH：HP 系统在 HP_ACTIVE 和 HP_SLEEP 之间切换的中间状态。硬件将在这个状态完成各个控制器的控制切换。
- LP_SWITCH：LP 系统在 HP_SLEEP 和 LP_SLEEP 之间切换的中间状态。硬件将在这个状态完成各个控制器的控制切换。

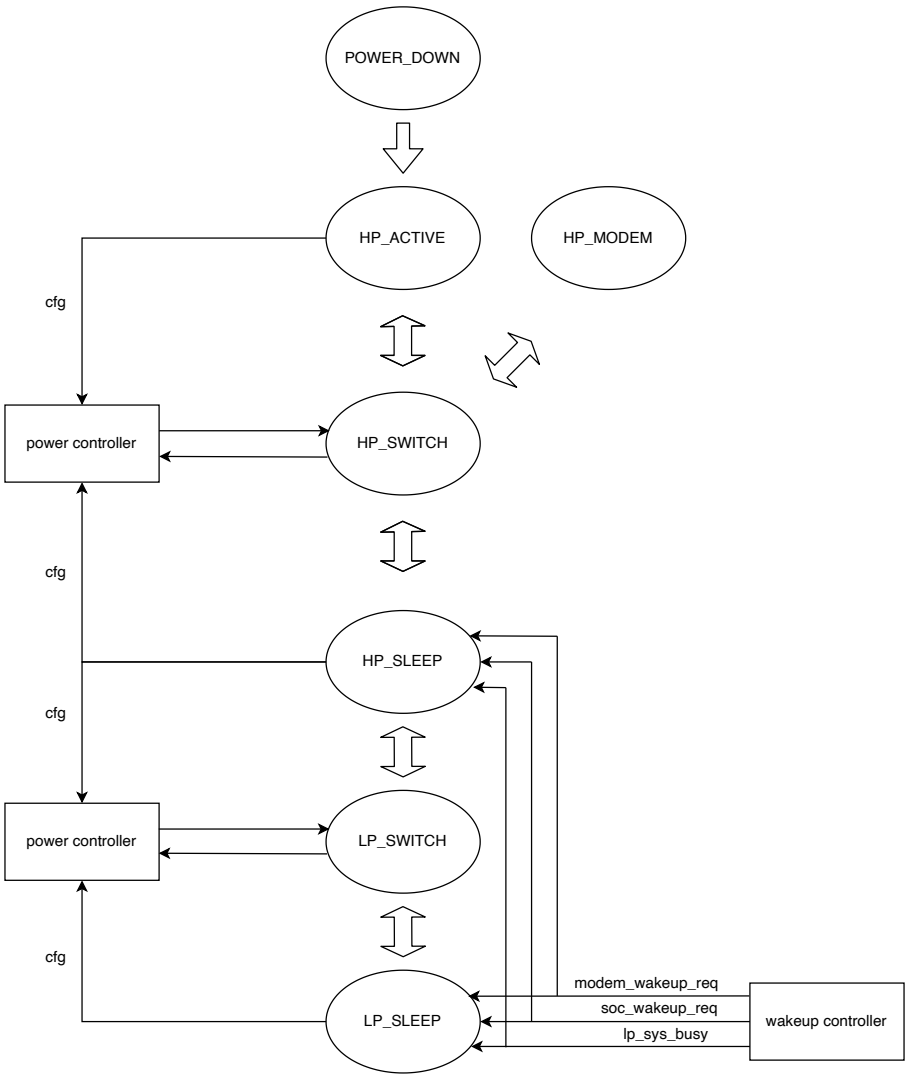


图 13.4-2. PMU 的主要工作流程

以下章节分别介绍 PMU 的各个主要组成部分。

13.4.2.1 PMU 主状态机

PMU 主状态机可以接收睡眠与唤醒信号，并通过功耗控制器完成电源上下电和时钟开关，从而实现 PMU 状态的切换，使芯片在目标 PMU 状态下达到性能与功耗的平衡。

PMU 主状态机包括 4 个 PMU 状态，每个状态由不同的睡眠与唤醒信号控制切换，支持软件对每个状态进行个性化的配置，如配置电源、时钟等。软件可基于这 4 个 PMU 状态，拓展出不同的应用层面的功耗模式。4 个 PMU 状态为：

- HP_ACTIVE：所有电路最大程度开启的 PMU 状态，该状态支持 HP 系统和 LP 系统工作。
- HP_MODEM：该状态支持 Modem 系统（无线 MAC 和基带）独立于 CPU 工作。
- HP_SLEEP：HP 系统睡眠状态，支持 LP 外设工作。
- LP_SLEEP：HP 系统与 LP 外设均处于睡眠状态，always-on 电路保持工作。

说明：
HP 和 LP 系统的划分为：

- LP 系统（外设）：LP RISC-V 32-bit Microprocessor、LP GPIO、LP UART、LP I2C
- LP 系统（always-on 电路）：LP Memory (LP SRAM)、PMU、RTC 看门狗定时器、RTC 超级看门狗、RTC 定时器、eFuse 控制器、电源毛刺监测器
- HP 系统：除 LP 系统以外的所有外设，包括 Modem

更多内容可参考 《ESP32-C5 技术规格书》 > 功能框图。

HP_ACTIVE，HP_MODEM，HP_SLEEP 为 HP 系统的状态，LP_SLEEP 为 LP 系统的状态。

- 某个模块如果属于 HP 系统，则只能配置在 HP_ACTIVE/HP_MODEM/HP_SLEEP 三种状态下的上下电，在 LP_SLEEP 的状态下会复用 HP_SLEEP 的状态。
- 某个模块如果属于 LP 系统，则只能配置 LP_SLEEP 状态下的上下电，HP_ACTIVE/HP_MODEM/HP_SLEEP 三种状态复用 HP_SLEEP 的状态。

例如，HP CPU 属于 HP 系统，则可以通过以下寄存器配置 HP CPU 在 HP_ACTIVE/HP_MODEM/HP_SLEEP 三种状态下的上下电，LP_SLEEP 复用 HP_SLEEP 的配置。

- HP_ACTIVE：PMU_HP_ACTIVE_PD_HP_CPU_PD_EN
- HP_MODEM：PMU_HP_MODEM_PD_HP_CPU_PD_EN
- HP_SLEEP：PMU_HP_SLEEP_PD_HP_CPU_PD_EN

类似地，用户可以用同样的方法，定义其他电源域在不同 PMU 状态下的上下电，具体寄存器请见第 13.9 小节。

说明：
为方便描述，后文中此类寄存器全部统称为 PMU_*PMUSTATE*_PD_*POWERDOMAIN*_PD_EN，*PMUSTATE* 代表 PMU 的 4 种状态。

配置完成后，PMU 将借助各控制器使这些配置生效，详见下方各章节的描述。

13.4.2.2 睡眠与唤醒控制器

睡眠与唤醒控制器负责向 PMU 主状态机发起睡眠与唤醒请求。ESP32-C5 支持多种唤醒源将 CPU 从不同功耗模式下唤醒，用户可通过 PMU_WAKEUP_ENA 按照下表使能对应的唤醒源。

表 13.4-1. 唤醒源

PMU_WAKEUP_ENA	唤醒源	Light-sleep	Deep-sleep
0x2	EXT_IO[1]	Y	Y
0x4	GPIO[1]	Y	Y
0x8	Wi-Fi beacon	Y	-
0x10	RTC Timer	Y	Y
0x20	Wi-Fi[2]	Y	–
0x40	UART0[3]	Y	–
0x80	UART1[3]	Y	–
0x100	SDIO 从机控制器[4]	Y	–
0x400	Bluetooth	Y	–
0x800	LP CPU	Y	Y

[1] Deep-sleep 模式下，只有 LP GPIO 可作为唤醒源。

[2] 为了通过 Wi-Fi 唤醒芯片，芯片将在 Active、Modem-sleep 和 Light-sleep 之间进行切换，RF 模块将在预设间隔中唤醒，保证 Wi-Fi 的正常连接和数据通信。

[3] 当接收到的 RX 脉冲数量超过阈值寄存器 UART_ACTIVE_THRESHOLD 的设置时，即触发唤醒。详情请见章节 *32 UART 控制器 (UART)*。

[4] 当 SDIO 从机控制器接收到 CMD52/CMD53 的时候，触发唤醒。该唤醒模式需要休眠的电压高于 0.9 V。电压配置存在芯片体质差异。

ESP32-C5 提供了硬件拒绝睡眠的机制，即当某些外设处于不可中断的工作状态而 CPU 试图进入睡眠状态时，外设将会发出唤醒信号，以此来拒绝 CPU 的睡眠命令，从而保证外设正常工作。

表 13.4-1 中的唤醒源均可配置成作为拒绝睡眠的事件，用户可配置如下寄存器实现拒绝睡眠。PMU_SLEEP_REJECT_ENA 和 PMU_SLP_REJECT_CAUSE_REG 的配置值与唤醒源的对应关系均与表中一致。

- 使能拒绝睡眠功能
  - 置位 PMU_SLP_REJECT_EN 使能拒绝睡眠功能
  - 配置 PMU_SLEEP_REJECT_ENA 使能拒绝睡眠信号源
- 读取寄存器 PMU_SLP_REJECT_CAUSE_REG 确认拒绝睡眠的事件源

13.4.2.3 模拟电源控制器

模拟电源控制器用于控制模拟电路部分（包括调压器、高速时钟、低速时钟）在某种 PMU 状态下的上下电情况。

调压器的具体配置如下：

- 配置寄存器 PMU_*PMUSTATE*_HP_REGULATOR_XPD 或 PMU_*PMUSTATE*_LP_REGULATOR_XPD 可使能或关闭目标 PMU 状态下 HP 系统或 LP 系统调压器的输出电压。LP 系统调压器不建议关闭，否则会导致 PMU 本身掉电，从而导致芯片异常。

高性能时钟（XTAL_CLK 和 PLL_CLK）的具体配置如下：

- XTAL_CLK：配置 PMU_HP_SLEEP_XPD_XTAL 为 1，在芯片 PMU 状态切换至 HP_SLEEP 的过程中，开启 XTAL_CLK。

  注意：为避免 XTAL_CLK 起振阶段的不稳定时钟影响电路，可配置 PMU_WAIT_XTAL_STABLE 实现 XTAL_CLK 全局门控的延迟打开，即该门控将在 XTAL_CLK 上电后的 PMU_WAIT_XTAL_STABLE 个 CLK_DYN_FAST_CLK 周期时间后打开。

- PLL_CLK：PMU 可在芯片处于不同 PMU 状态下打开 PLL_CLK，仅需配置不同 PMU 状态下的 PMU_HP_ACTIVE_XPD_BBPLL 寄存器为 1。例如，配置 PMU_HP_ACTIVE_XPD_BBPLL 为 1，则在芯片处于 HP_ACTIVE 状态下，打开 PLL_CLK 时钟。

  注意：

  - 在打开 PLL_CLK 前还需确保 XTAL_CLK 已经稳定开启。
  - 为避免 PLL_CLK 起振阶段的不稳定时钟影响电路，可配置 PMU_WAIT_PLL_STABLE 实现 PLL_CLK 全局门控的延迟打开，即该门控将在 PLL_CLK 上电后的 PMU_WAIT_PLL_STABLE 个 CLK_DYN_FAST_CLK 周期时间后打开。

低速时钟功耗较低（RC_FAST_CLK 和 XTAL32K_CLK），在 HP_ACTIVE、HP_MODEM 和 HP_SLEEP 状态下的开关情况均由 PMU_HP_SLEEP_LP_CK_POWER_REG 决定。在 LP_SLEEP 状态下由 PMU_LP_SLEEP_LP_CK_POWER_REG 配置。

13.4.2.4 数字电源控制器

数字电源控制器用于控制数字电源域在某种 PMU 状态下的上下电情况。与模拟电源控制器不同的是，数字电源控制器不直接控制调压器，而是通过控制与调压器相连的电源开关，实现各个数字电源域的上下电。

数字电源域中 LP PD Peripherals 只支持在 HP_SLEEP 与 LP_SLEEP 两个 PMU 状态之间切换时上下电，其余电源域只支持在 HP_SLEEP、HP_ACTIVE，HP_MODEM 三个 PMU 状态间切换时上下电。

当芯片在不同 PMU 状态之间切换时，如果某电源域在当下 PMU 状态下的上下电配置与即将切换至的上下电配置不符，则将激活上下电流程。以 CPU 电源域从上电到下电的流程为例。配置寄存器 PMU_HP_MODEM_PD_HP_CPU_PD_EN 为 1，表示在 HP_MODEM 状态下关闭 CPU 电源域的供电；或配置为 0，表示在 HP_MODEM 状态下对 CPU 电源域供电。PMU 将做出如下控制：

- 使能数字隔离单元，确保下电模块不往上电模块输出不稳定的电平值。当有电源域掉电时，该模块的输出将被钳位至固定值。
- 使能复位，CPU 电源域掉电后，该电源域的全局复位信号处于复位状态，在重新上电后的一段时间内该电源域仍然保持复位，这将确保该电源域在上电的过程中存在一个复位到释放的过程，消除上下电引入的寄存器不定态。

下面说明各个数字电源域的开关方式：

- Internal SRAMx

  Internal SRAMx 电源域的上下电状态并无专门寄存器控制，它与 Peripherals 共用一套 PMU_*PMUSTATE*_PD_TOP_PD_EN 寄存器。若未配置 PMU_PD_HP_MEM*n*_PD_MASK (*n*=0,1,2) 寄存器，则 Internal SRAMx 与 Peripherals 电源域将同时上下电；若配置 PMU_PD_HP_MEMn_PD_MASK 寄存器为 1，则 Internal SRAMx 在 Peripherals 掉电时可保持上电。

- Modem Power

  Modem Power 与 HP 系统调压器通过一个开关连接，可通过配置寄存器 PMU_HP_ACTIVE_PD_HP_AON_PD_EN 决定 HP_ACTIVE 状态下 Modem Power 的开关。由图 13.4-1 可

知，CPU、Modem、Peripherals + ROM 中若要一个电源域上电，则需同时保证 Modem Power 电源域上电，这种设计也与功能需求保持一致。

- Peripherals + ROM/Modem/CPU

  可通过寄存器 PMU_*PMUSTATE*_PD_*n*_PD_EN (*n*=TOP/HP_WIFI/HP_CPU) 配置各电源域在对应 PMU 状态下的上下电（寄存器中的 WIFI 即为 Modem 电源域）。

  Peripherals 掉电时还存在以下几个可配功能：

  - Peripherals 掉电后会导致 GPIO 管脚状态不稳定，可通过 PMU 控制 GPIO 管脚（8 个 LP GPIO 除外）进入保持状态。例如配置 PMU_HP_SLEEP_HP_PAD_HOLD_ALL 使 GPIO 管脚在 HP_SLEEP 下保持 Peripherals 掉电前的相关配置。
  - 在 HP_ACTIVE 和 HP_MODEM 状态下，除了支持 Internal SRAMx 电源域掉电外，还支持标准供电下的 Memory Deep-sleep 模式，该模式下存储器将无法读写，但可以保持数据。配置寄存器 PMU_*PMUSTATE*_HP_MEM_DSLP 可实现 Memory Deep-sleep 模式。

- LP PD Peripherals

  LP PD Peripherals 电源域在芯片处于 HP_ACTIVE 和 HP_MODEM 状态下始终保持上电，无法配置，仅允许配置 HP_SLEEP 与 LP_SLEEP 状态下的上下电。此电源域拥有独立的数字电源开关，且它的上下电控制不依赖于其他数字电源域的上下电。

13.4.2.5 时钟控制器

时钟控制器主要用于芯片 PMU 状态切换时，对时钟源的选择及部分时钟的使能，以此实现对高性能系统时钟与低功耗系统时钟的控制。

高性能系统时钟包括 HP_ROOT_CLK 和高性能系统外设时钟。当芯片的 PMU 状态在 HP_ACTIVE、HP_MODEM 与 HP_SLEEP 之间切换时，PMU 可实现对 HP_ROOT_CLK 时钟源的切换、开关、分频，以及高性能系统外设时钟的开关。

- HP_ROOT_CLK 控制的相关寄存器及功能介绍如下：
  - 配置不同 PMU 状态对应的 PMU_*PMUSTATE*_SYS_CLK_SLP_SEL 寄存器为 1，表示芯片进入该 PMU 状态后，时钟源受 PMU 控制。
  - 配置不同 PMU 状态对应的 PMU_*PMUSTATE*_ICG_SYS_CLOCK_EN 寄存器为 0，表示芯片进入该 PMU 状态后，关闭 HP_ROOT_CLK 的时钟门控，即该时钟将不可用。
  - 配置不同 PMU 状态对应的 PMU_*PMUSTATE*_DIG_SYS_CLK_SEL，选择芯片进入对应 PMU 状态后使用的时钟源，具体请见章节 9 复位和时钟 > 表 9.2-1。
- 高性能系统外设时钟控制的相关寄存器及功能介绍如下：
  - 配置 PMU_*PMUSTATE*_ICG_SLP_SEL 寄存器为 1，目标状态下的各时钟门控的开关将受 PMU 的控制，该寄存器为 0 时，门控开关由 PCR 寄存器决定。
  - 配置寄存器 PMU_*PMUSTATE*_DIG_ICG_FUNC_EN 决定目标 PMU 状态的功能时钟门控开关。具体配置请见章节 9 复位和时钟 > 表 9.2-8。
  - 配置寄存器 PMU_*PMUSTATE*_DIG_ICG_APB_EN 决定目标 PMU 状态的 APB 时钟门控开关。具体配置请见章节 9 复位和时钟 > 表 9.2-7。

低功耗系统时钟主要用于低功耗系统，包括如下四种时钟：

- LP_SLOW_CLK
- LP_FAST_CLK
- LP_DYN_SLOW_CLK
- LP_DYN_FAST_CLK

其中 LP_DYN_FAST_CLK 的时钟频率根据 PMU 状态由硬件完成如下固定切换（用户无法更改）：

- LP_SLEEP：频率与 LP_SLOW_CLK 一致。
- HP_ACTIVE、HP_MODEM、HP_SLEEP：频率与 LP_FAST_CLK 一致。

13.4.2.6 数据备份控制器

ESP32-C5 支持在切换 PMU 状态时，驱动一个 Retention DMA 模块，进行存储器与外设间的数据搬移，从而在电源域掉电时备份数据并在再次上电时恢复数据。

数据搬移操作由 Peripherals 电源域中的模块实现，PMU 只产生相关控制信号。需要注意的是，决定数据搬移的控制寄存器存在方向性，因为与 PMU 的其他控制寄存器不同是，这些控制行为由原来的 PMU 状态和目标 PMU 状态共同决定。

以目标 PMU 状态为 HP_SLEEP 为例，从 HP_ACTIVE 睡眠至 HP_SLEEP 与从 HP_MODEM 睡眠至 HP_SLEEP 所需要备份的寄存器显然是不同的，因此下文的寄存器名中的 *PMUSTATE_TRANS* 存在如下可能：

- HP_SLEEP2ACTIVE
- HP_SLEEP2MODEM
- HP_MODEM2ACTIVE
- HP_MODEM2SLEEP
- HP_ACTIVE2SLEEP

下面将介绍 PMU 支持的 Retention DMA 控制：

- 使能数据搬移：配置寄存器 PMU_*PMUSTATE_TRANS*_BACKUP_EN 为 1，表示在进行对应的 PMU 状态切换时使能数据搬移。
- 打开对应时钟：在数据搬移开始前，配置 PMU_*PMUSTATE_TRANS*_BACKUP_CLK_SEL 选择 Retention DMA 工作时钟源，配置 PMU_*PMUSTATE*_BACKUP_ICG_FUNC_EN 为 1 打开时钟。

  数据搬移结束后，门控时钟寄存器 PMU_*PMUSTATE*_BACKUP_ICG_FUNC_EN 的值将由目标状态下的 PMU_*PMUSTATE*_DIG_ICG_FUNC_EN 决定。
- 配置数据搬移方向：配置 PMU_*PMUSTATE_TRANS*_BACKUP_MODE 寄存器的最高位：
  - 1：外设到存储器
  - 0：存储器到外设
- 选择链表指针：配置 PMU_*PMUSTATE_TRANS*_BACKUP_MODE 寄存器的低四位，选择链表指针，指针在链表中设置。

13.4.2.7 系统控制器

系统控制器用于在芯片切换 PMU 状态时控制部分功能模块，以实现芯片在各 PMU 状态下的稳定、节能工作，主要可控制的功能模块包括：

- **暂停看门狗功能**：配置 PMU_*PMUSTATE*_DIG_PAUSE_WDT 寄存器为 1，可在 PMU 状态切换至对应 PMU 状态时禁用 RTC 看门狗定时器 (RWDT) 功能。注意，任一睡眠模式下若配置该寄存器为 0，表示不禁用该功能，则 RWDT 将因为没有 CPU 参与喂狗而产生复位。
- **切换 GPIO 至管脚睡眠模式**，该模式下将固定管脚的配置。例如将某个 GPIO 管脚固定为输入，低驱动模式以降低其功耗，并作为一个唤醒管脚使用，配置 PMU_*PMUSTATE*_HP_PAD_HOLD_ALL 寄存器为 1，可在 PMU 状态切换至对应 PMU 状态后锁存当前的配置作为 GPIO 的睡眠配置。GPIO 管脚保持功能的详细介绍请参考章节 *8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* > 小节 *8.9 GPIO 管脚的 Hold 特性*。
- **禁用 UART 唤醒功能**：配置 PMU_*PMUSTATE*_UART_WAKEUP_EN 寄存器为 0，禁用对应 PMU 状态下的 4 种 UART 唤醒模式。有关 UART 唤醒模式的具体介绍，请见章节 *32 UART 控制器 (UART)*。
- **暂停 CPU 运行**：配置 PMU_*PMUSTATE*_DIG_CPU_STALL 寄存器为 1，在对应 PMU 状态下将 CPU 挂起。

13.5 功耗模式

ESP32-C5 有 4 个可配置的 PMU 状态，为了方便用户的使用，我们基于这 4 个可配置的 PMU 状态，提供了 5 种预设功耗模式，可满足最常见的场景需求。

表 13.5-1. 预设的功耗模式

功耗模式	电源域								
	LP always-on	LP PD peripherals	Peripherals + ROM	Modem	CPU	RC_FAST_CLK	XTAL_CLK	PLL	RF circuit
Active	ON	ON	ON	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	ON	ON	ON	ON	ON	ON/OFF	OFF
Light-sleep0	ON	ON	ON	ON/OFF	OFF	ON	ON/OFF	ON/OFF	ON/OFF
Light-sleep1	ON	ON/OFF	ON/OFF	ON/OFF	OFF	ON/OFF	ON/OFF	ON/OFF	ON/OFF
Deep-sleep	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

说明：

1. 具体功耗数据可参考《*ESP32-C5 技术规格书*》。
2. 具体可支持的唤醒源，请见表 13.4-1。

13.6 RTC 启动

在 Deep-sleep 下，芯片的 ROM 和 RAM 均断电，因此在唤醒时 SPI Boot（从 flash 复制数据）所需时间更长。因此，相较于 Light-sleep 和 Modem-sleep 模式，Deep-sleep 模式的唤醒时间要长的多。不过，值得注意的是，在 Deep-sleep 模式下，16 KB SRAM 可以处于上电状态。因此，用户可以将一些代码规模不大（如 “deep sleep wake stub”）写入低功耗管理系统内的 16 KB SRAM，避免 SPI 启动带来的延迟，从而加速芯片唤醒过程。

使用过程如下所示：

1. 配置 LP_AON_CORE0_STAT_VECTOR_SEL 为 0，使 CPU 从 LP SRAM 启动。
2. 计算 LP SRAM 的 CRC 码，并将结果保存在寄存器 LP_AON_STORE7_REG 中。
3. 配置寄存器 LP_AON_STORE6_REG 为“deep sleep wake stub”在 LP SRAM 的入口地址。
4. 配置芯片进入睡眠模式。
5. 当 CPU 开启时，开始进行 ROM 解包和部分初始化工作。此后，再次计算 LP SRAM 的 CRC 码。如果与寄存器 LP_AON_STORE7_REG 中保存的结果一致，则 CPU 跳转至“deep sleep wake stub”；如果 CRC 不一致，则直接进入 SPI boot 流程。

ESP32-C5 唤醒后的启动流程如图 13.6-1 所示：

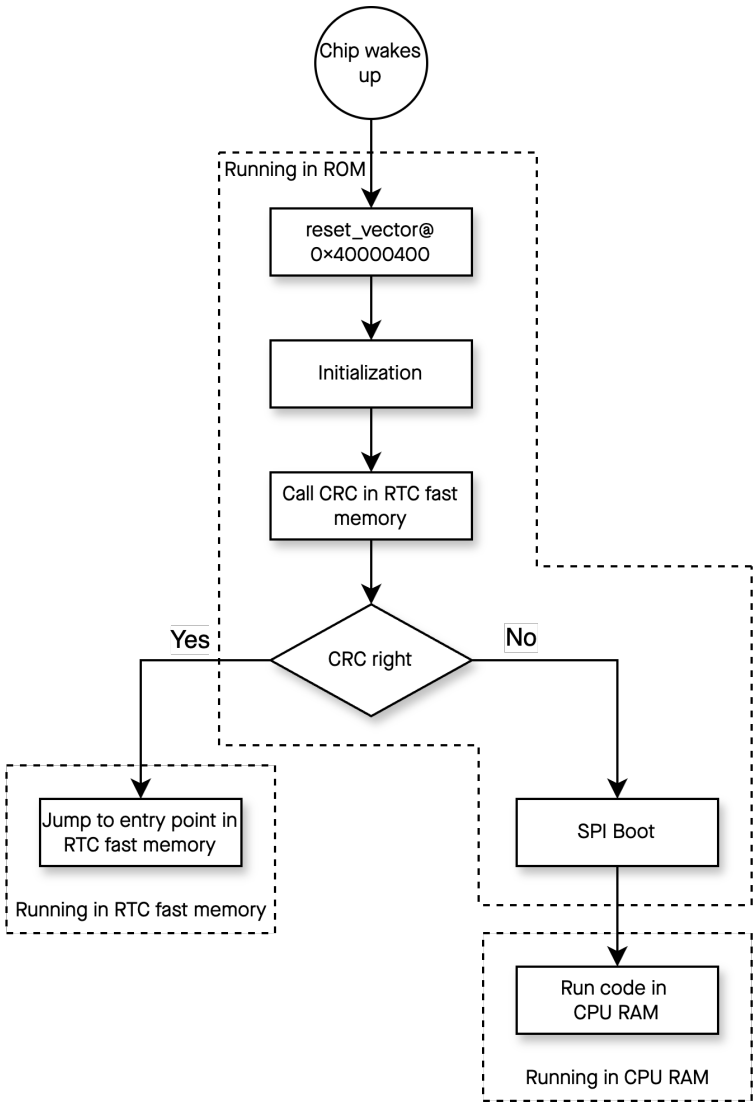


图 13.6-1. ESP32-C5 唤醒后的启动流程

13.7 事件任务矩阵功能

在 ESP32-C5 中，低功耗管理模块支持 ETM 功能，即可以通过任意外设的 ETM 事件触发低功耗管理模块的 ETM 任务，或者通过低功耗管理模块的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与低功耗管理模块相关的 ETM 任务和 ETM 事件。

低功耗管理模块可接收的 ETM 任务有：

- PMU_TASK_SLEEP_REQ：该任务将触发 PMU 开始睡眠流程。

低功耗管理模块可产生的 ETM 事件有：

- PMU_EVT_SLEEP_WAKEUP：该事件表示 PMU 被唤醒至 HP_ACTIVE。

13.8 中断

ESP32-C5 的低功耗管理模块可以生成以下中断信号：

- PMU_INTR
- PMU_LP_INT

PMU_INTR 发送给 中断矩阵，PMU_LP_INT 输出到 LP CPU。

这些中断信号由各自模块的内部中断源生成。具体为：

PMU_INTR：

- PMU_SOC_WAKEUP_INT：当芯片被唤醒至 HP_ACTIVE 时触发。
- PMU_SOC_SLEEP_REJECT_INT：当拒绝睡眠源拒绝了一个睡眠请求时触发。
- PMU_SW_INT：当用 LP CPU 作为唤醒源将芯片唤醒至 HP_ACTIVE 时触发。
- PMU_SDIO_IDLE_INT：当 SDIO 处于 IDLE 状态时触发。
- PMU_LP_CPU_EXC_INT：当 LP CPU 异常时触发。

PMU_LP_INT：

- PMU_HP_SW_TRIGGER_INT：当用 HP CPU 作为唤醒源唤醒 LP CPU 时触发。
- PMU_ACTIVE_SWITCH_SLEEP_START_INT：当 PMU 状态开始从 HP_ACTIVE 切换至 HP_SLEEP 时触发。
- PMU_MODEM_SWITCH_SLEEP_START_INT：当 PMU 状态开始从 HP_MODEM 切换至 HP_SLEEP 时触发。
- PMU_SLEEP_SWITCH_MODEM_START_INT：当 PMU 状态开始从 HP_SLEEP 切换至 HP_MODEM 时触发。
- PMU_SLEEP_SWITCH_ACTIVE_START_INT：当 PMU 状态开始从 HP_SLEEP 切换至 HP_ACTIVE 时触发。
- PMU_MODEM_SWITCH_ACTIVE_START_INT：当 PMU 状态开始从 HP_MODEM 切换至 HP_ACTIVE 时触发。
- PMU_ACTIVE_SWITCH_SLEEP_END_INT：当 PMU 状态完成从 HP_ACTIVE 切换至 HP_SLEEP 时触发。
- PMU_MODEM_SWITCH_SLEEP_END_INT：当 PMU 状态完成从 HP_MODEM 切换至 HP_SLEEP 时触发。
- PMU_SLEEP_SWITCH_MODEM_END_INT：当 PMU 状态完成从 HP_SLEEP 切换至 HP_MODEM 时触发。
- PMU_SLEEP_SWITCH_ACTIVE_END_INT：当 PMU 状态完成从 HP_SLEEP 切换至 HP_ACTIVE 时触发。
- PMU_MODEM_SWITCH_ACTIVE_END_INT：当 PMU 状态完成从 HP_MODEM 切换至 HP_ACTIVE 时触发。
- PMU_LP_CPU_WAKEUP_INT：当 LP CPU 被唤醒时触发。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 13.9 寄存器列表。

13.9 寄存器列表

13.9.1 PMU 寄存器列表

本小节的所有地址均为相对于 PMU 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
PMU_HP_ACTIVE_DIG_POWER_REG	HP_ACTIVE 状态下数字电源域配置寄存器	0x0000	R/W
PMU_HP_ACTIVE_ICG_HP_FUNC_REG	HP_ACTIVE 状态下高性能系统外设功能时钟控制寄存器	0x0004	R/W
PMU_HP_ACTIVE_ICG_HP_APB_REG	HP_ACTIVE 状态下高性能系统外设 APB 时钟门控控制寄存器	0x0008	R/W
PMU_HP_ACTIVE_HP_SYS_CNTL_REG	HP_ACTIVE 状态下系统控制寄存器	0x0010	R/W
PMU_HP_ACTIVE_HP_CK_POWER_REG	HP_ACTIVE 状态下时钟源电源控制寄存器	0x0014	R/W
PMU_HP_ACTIVE_BACKUP_REG	HP_ACTIVE 状态下数据备份控制寄存器	0x001C	R/W
PMU_HP_ACTIVE_BACKUP_CLK_REG	HP_ACTIVE 状态下备份模块工作时钟控制寄存器	0x0020	R/W
PMU_HP_ACTIVE_SYSCLK_REG	HP_ACTIVE 状态下系统时钟控制寄存器	0x0024	R/W
PMU_HP_ACTIVE_HP_REGULATORO_REG	HP_ACTIVE 状态下 HP sys 调压器电源控制寄存器	0x0028	R/W
PMU_HP_ACTIVE_XTAL_REG	HP_ACTIVE 状态下 XTAL_CLK 电源控制寄存器	0x0030	R/W
PMU_HP_MODEM_DIG_POWER_REG	HP_MODEM 状态下数字电源域控制寄存器	0x0034	R/W
PMU_HP_MODEM_ICG_HP_FUNC_REG	HP_MODEM 状态下高性能系统外设功能时钟控制寄存器	0x0038	R/W
PMU_HP_MODEM_ICG_HP_APB_REG	HP_MODEM 状态下高性能系统外设 APB 时钟门控控制寄存器	0x003C	R/W
PMU_HP_MODEM_HP_SYS_CNTL_REG	HP_MODEM 状态下系统控制寄存器	0x0044	R/W
PMU_HP_MODEM_HP_CK_POWER_REG	HP_MODEM 状态下时钟源电源控制寄存器	0x0048	R/W
PMU_HP_MODEM_BACKUP_REG	HP_MODEM 状态下数据备份控制寄存器	0x0050	R/W
PMU_HP_MODEM_BACKUP_CLK_REG	HP_MODEM 状态下数据备份模块工作时钟控制寄存器	0x0054	R/W
PMU_HP_MODEM_SYSCLK_REG	HP_MODEM 状态下系统时钟控制寄存器	0x0058	R/W

名称	描述	地址	访问
PMU_HP_MODEM_HP_REGULATOR0_REG	HP_MODEM 状态下 HP sys 调压器控制寄存器	0x005C	R/W
PMU_HP_MODEM_XTAL_REG	HP_MODEM 状态下 XTAL_CLK 电源控制寄存器	0x0064	R/W
PMU_HP_SLEEP_DIG_POWER_REG	HP_SLEEP 状态下数字电源域控制寄存器	0x0068	R/W
PMU_HP_SLEEP_ICG_HP_FUNC_REG	HP_SLEEP 状态下高性能系统外设功能时钟控制寄存器	0x006C	R/W
PMU_HP_SLEEP_ICG_HP_APB_REG	HP_SLEEP 状态下高性能系统外设 APB 时钟门控控制寄存器	0x0070	R/W
PMU_HP_SLEEP_HP_SYS_CNTL_REG	HP_SLEEP 状态下系统控制寄存器	0x0078	R/W
PMU_HP_SLEEP_HP_CK_POWER_REG	HP_SLEEP 状态下时钟源电源控制寄存器	0x007C	R/W
PMU_HP_SLEEP_BACKUP_REG	HP_SLEEP 状态下数据备份控制寄存器	0x0084	R/W
PMU_HP_SLEEP_BACKUP_CLK_REG	目前状态为 HP_SLEEP 状态时，数据备份时钟控制寄存器	0x0088	R/W
PMU_HP_SLEEP_SYSCLK_REG	HP_SLEEP 状态下系统时钟控制寄存器	0x008C	R/W
PMU_HP_SLEEP_HP_REGULATOR0_REG	HP_SLEEP 状态下 HP sys 调压器控制寄存器	0x0090	R/W
PMU_HP_SLEEP_XTAL_REG	HP_SLEEP 状态下 XTAL_CLK 电源控制寄存器	0x0098	R/W
PMU_HP_SLEEP_LP_REGULATOR0_REG	HP_SLEEP 状态下 LP sys 调压器电源控制寄存器	0x009C	R/W
PMU_HP_SLEEP_LP_DIG_POWER_REG	HP_SLEEP 状态下 LP 系统数字电源域控制寄存器	0x00A8	R/W
PMU_HP_SLEEP_LP_CK_POWER_REG	HP_ACTIVE/HP_MODEM/HP_SLEEP 状态下低速时钟电源控制寄存器	0x00AC	R/W
PMU_LP_SLEEP_LP_REGULATOR0_REG	LP_SLEEP 状态下 LP sys 调压器电源控制寄存器	0x00B4	R/W
PMU_LP_SLEEP_XTAL_REG	LP_SLEEP 状态下 XTAL_CLK 电源控制寄存器	0x00BC	R/W
PMU_LP_SLEEP_LP_DIG_POWER_REG	LP_SLEEP 状态下数字电源域控制寄存器	0x00C0	R/W
PMU_LP_SLEEP_LP_CK_POWER_REG	LP_SLEEP 状态下低速时钟电源控制寄存器	0x00C4	R/W
PMU_POWER_PD_MEM_MASK_REG	Internal SRAMx 电源域强制上电寄存器	0x0114	R/W
PMU_POWER_CK_WAIT_CNTL_REG	XTAL_CLK 和 PLL_CLK 等待稳定周期配置寄存器	0x0120	R/W
PMU_SLP_WAKEUP_CNTL0_REG	睡眠请求寄存器	0x0124	WT
PMU_SLP_WAKEUP_CNTL1_REG	睡眠拒绝寄存器	0x0128	R/W
PMU_SLP_WAKEUP_CNTL2_REG	唤醒源使能寄存器	0x012C	R/W
PMU_SLP_WAKEUP_CNTL4_REG	睡眠拒绝事件清除寄存器	0x0134	WT
PMU_SLP_WAKEUP_STATUS0_REG	唤醒事件寄存器	0x0144	RO
PMU_SLP_WAKEUP_STATUS1_REG	复位拒绝事件寄存器	0x0148	RO

名称	描述	地址	访问
PMU_INT_RAW_REG	PMU 睡眠/唤醒原始中断	0x0160	R/WTC/SS
PMU_HP_INT_ST_REG	PMU 睡眠/唤醒状态中断	0x0164	RO
PMU_HP_INT_ENA_REG	PMU 睡眠/唤醒中断使能	0x0168	R/W
PMU_HP_INT_CLR_REG	PMU 睡眠/唤醒中断清除	0x016C	WT
PMU_LP_INT_RAW_REG	低功耗系统原始中断	0x0170	R/WTC/SS
PMU_LP_INT_ST_REG	PMU 状态转换中断状态	0x0174	RO
PMU_LP_INT_ENA_REG	PMU 状态转换中断使能	0x0178	R/W
PMU_LP_INT_CLR_REG	PMU 状态转换中断清除	0x017C	WT
PMU_LP_CPU_PWR0_REG	LP CPU 控制寄存器	0x0180	R/W
PMU_LP_CPU_PWR1_REG	LP CPU 睡眠请求寄存器	0x0184	varies
PMU_HP_LP_CPU_COMM_REG	软件中断寄存器	0x0188	WT
PMU_DATE_REG	版本控制寄存器	0x01A8	R/W

13.9.2 Always-on 寄存器列表

本小节的所有地址均为相对于 Always-on 寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_AON_STORE0_REG	Always-on 寄存器 0	0x0000	R/W
LP_AON_STORE1_REG	Always-on 寄存器 1	0x0004	R/W
LP_AON_STORE2_REG	Always-on 寄存器 2	0x0008	R/W
LP_AON_STORE3_REG	Always-on 寄存器 3	0x000C	R/W
LP_AON_STORE4_REG	Always-on 寄存器 4	0x0010	R/W
LP_AON_STORE5_REG	Always-on 寄存器 5	0x0014	R/W
LP_AON_STORE6_REG	Always-on 寄存器 6	0x0018	R/W
LP_AON_STORE7_REG	Always-on 寄存器 7	0x001C	R/W
LP_AON_STORE8_REG	Always-on 寄存器 8	0x0020	R/W
LP_AON_STORE9_REG	Always-on 寄存器 9	0x0024	R/W
LP_AON_SYS_CFG_REG	配置系统复位	0x0034	WT
LP_AON_CPUCORE0_CFG_REG	CPU 启动地址配置寄存器	0x0038	varies

13.10 寄存器

13.10.1 PMU 寄存器

本小节的所有地址均为相对于 PMU 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 13.1. PMU_HP_ACTIVE_DIG_POWER_REG (0x0000)

Bits	Field	Reset
31	PMU_HP_ACTIVE_PD_TOP_PD_EN	0
30	PMU_HP_ACTIVE_PD_HP_AON_PD_EN	0
29	PMU_HP_ACTIVE_PD_HP_CPU_PD_EN	0
28	(reserved)	0
27	PMU_HP_ACTIVE_PD_HP_WIFI_PD_EN	0
26–23	(reserved)	0 0 0 0
22	PMU_HP_ACTIVE_HP_MEM_DSLP	0
21	PMU_HP_ACTIVE_VDD_SPI_PD_EN	0
20–0	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

PMU_HP_ACTIVE_VDD_SPI_PD_EN 配置是否在 HP_ACTIVE 状态下关闭外部 flash。

0：打开

1：关闭

(R/W)

PMU_HP_ACTIVE_HP_MEM_DSLP 配置是否在 HP_ACTIVE 状态下让 Internal SRAMx 进入 Deep-sleep。

0：不进入 Deep-sleep

1：进入 Deep-sleep

(R/W)

PMU_HP_ACTIVE_PD_HP_WIFI_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 Modem 电源域。

0：打开

1：关闭

(R/W)

PMU_HP_ACTIVE_PD_HP_CPU_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 CPU 电源域。

0：打开

1：关闭

(R/W)

PMU_HP_ACTIVE_PD_HP_AON_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 Modem Power 电源域。

0：打开

1：关闭

(R/W)

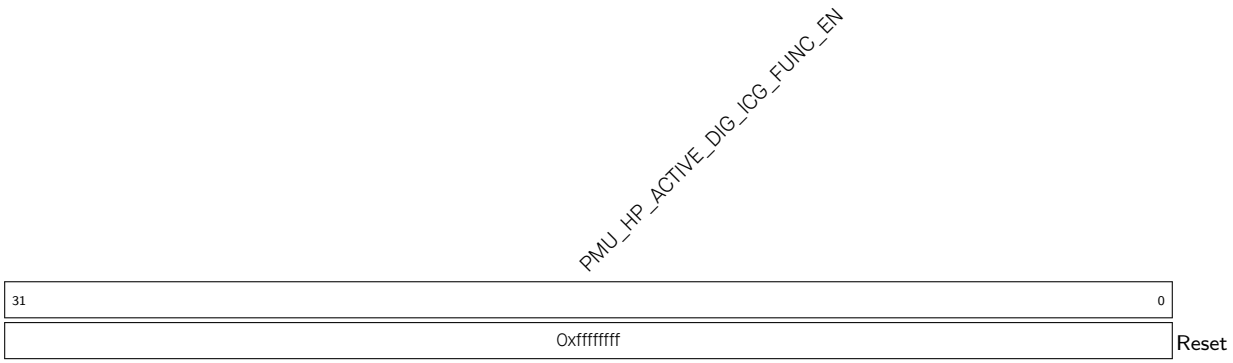
PMU_HP_ACTIVE_PD_TOP_PD_EN 配置是否在 HP_ACTIVE 状态下关闭 Peripherals 电源域。

0：打开

1：关闭

(R/W)

Register 13.2. PMU_HP_ACTIVE_ICG_HP_FUNC_REG (0x0004)



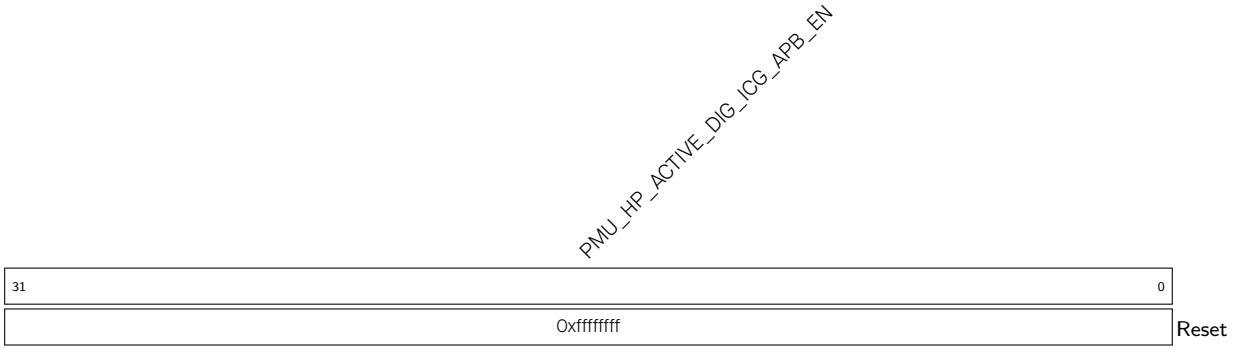
PMU_HP_ACTIVE_DIG_ICG_FUNC_EN 配置 HP_ACTIVE 状态下高性能系统外设功能时钟的开关。具体配置请见章节 9 复位和时钟 > 表 9.2-8。

0：关闭

1：打开

(R/W)

Register 13.3. PMU_HP_ACTIVE_ICG_HP_APB_REG (0x0008)



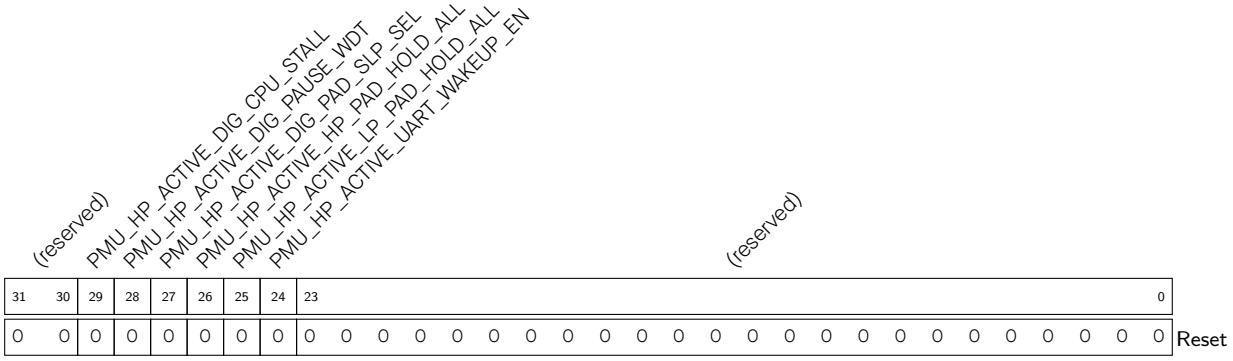
PMU_HP_ACTIVE_DIG_ICG_APB_EN 配置 HP_ACTIVE 状态下高性能系统外设 APB 时钟的开关。具体配置请见章节 9 复位和时钟 > Table 9.2-7。

0：关闭

1：打开

(R/W)

Register 13.4. PMU_HP_ACTIVE_HP_SYS_CNTL_REG (0x0010)



PMU_HP_ACTIVE_UART_WAKEUP_EN 配置是否在 HP_ACTIVE 状态下使能 UART 唤醒功能。

0：关闭唤醒功能

1：使能唤醒功能

(R/W)

PMU_HP_ACTIVE_LP_PAD_HOLD_ALL 配置是否在 HP_ACTIVE 状态下保持 LP GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_ACTIVE_HP_PAD_HOLD_ALL 配置是否在 HP_ACTIVE 状态下保持 GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_ACTIVE_DIG_PAD_SLP_SEL 配置是否在 HP_ACTIVE 状态下使用 GPIO 的 Light-sleep 模式配置。

0：正常工作模式的配置

1：Light-sleep 模式配置，详见章节 *8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* > 小节 *8.8 Light-sleep 模式管脚功能*。

(R/W)

PMU_HP_ACTIVE_DIG_PAUSE_WDT 配置是否在 HP_ACTIVE 状态下暂停看门狗。

0：不暂停

1：暂停

(R/W)

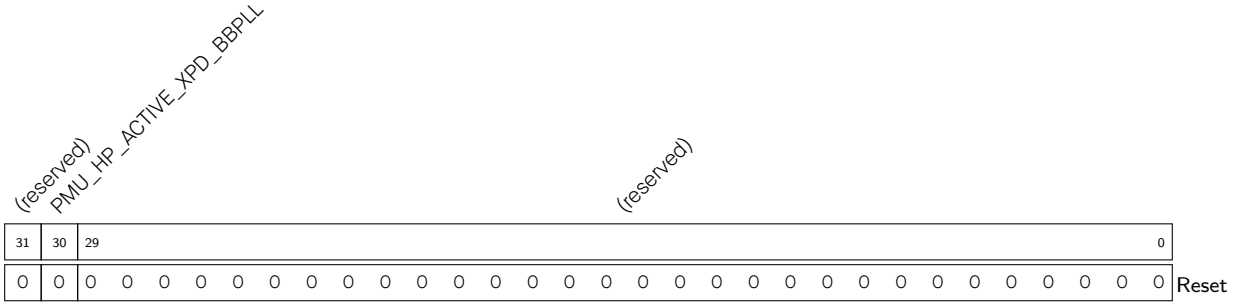
PMU_HP_ACTIVE_DIG_CPU_STALL 配置是否在 HP_ACTIVE 状态下暂停 CPU。

0：不暂停

1：暂停

(R/W)

Register 13.5. PMU_HP_ACTIVE_HP_CK_POWER_REG (0x0014)



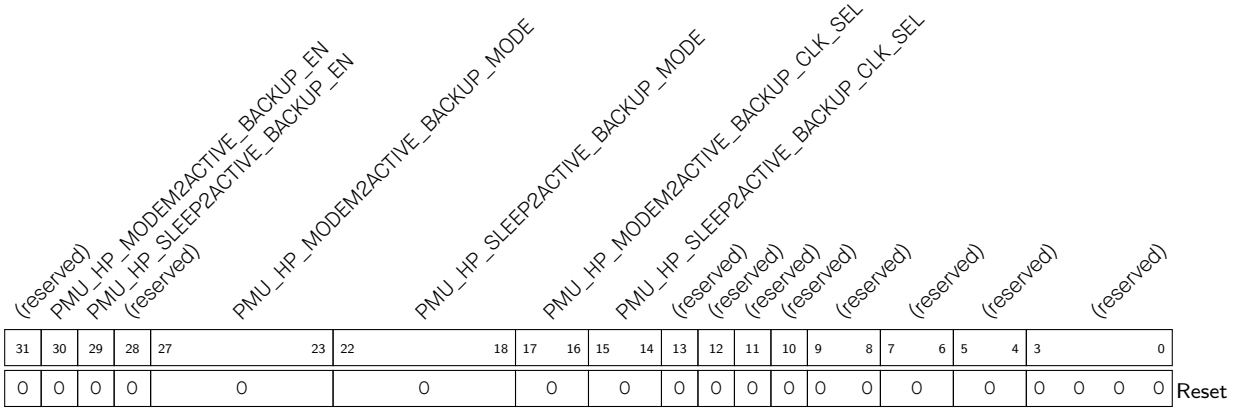
PMU_HP_ACTIVE_XPD_BBPLL 配置是否在 HP_ACTIVE 状态下打开 PLL_CLK 时钟。

0：关闭

1：打开

(R/W)

Register 13.6. PMU_HP_ACTIVE_BACKUP_REG (0x001C)



PMU_HP_SLEEP2ACTIVE_BACKUP_CLK_SEL 配置备份模块从 HP_SLEEP 切换为 HP_ACTIVE 时的功能时钟源。

0：时钟源为 XTAL

1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

PMU_HP_MODEM2ACTIVE_BACKUP_CLK_SEL 配置备份模块从 HP_MODEM 切换为 HP_ACTIVE 时的功能时钟源。配置同上。(R/W)

PMU_HP_SLEEP2ACTIVE_BACKUP_MODE 配置从 HP_SLEEP 切换为 HP_ACTIVE 时数据搬移的方向和链表指针。

最高位：

0：存储器到外设

1：外设到存储器

低四位：选择链表指针，指针在链表中设置。

(R/W)

PMU_HP_MODEM2ACTIVE_BACKUP_MODE 配置从 HP_MODEM 切换为 HP_ACTIVE 时数据搬移的方向和链表指针。配置同上。

PMU_HP_SLEEP2ACTIVE_BACKUP_EN 配置是否在从 HP_SLEEP 切换为 HP_ACTIVE 时使能数据搬移。

0：禁能

1：使能

(R/W)

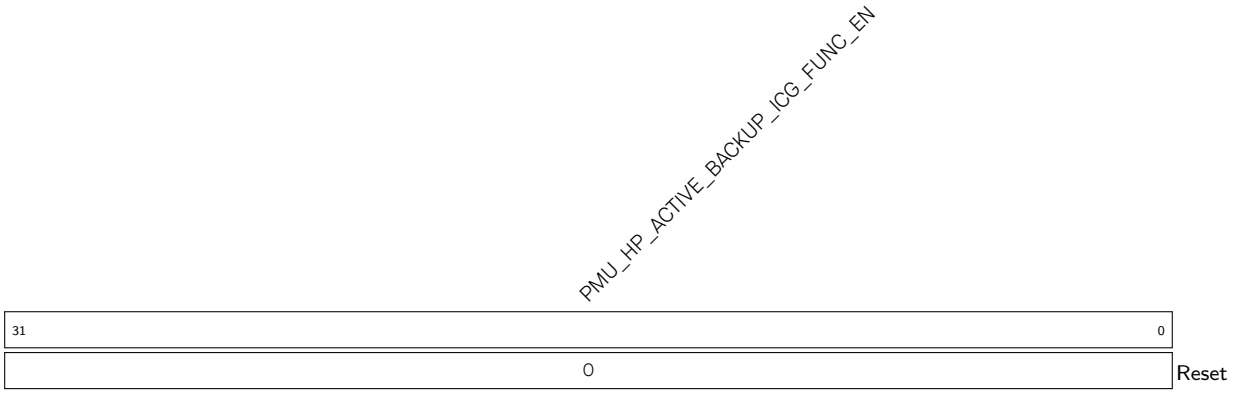
PMU_HP_MODEM2ACTIVE_BACKUP_EN 配置是否在从 HP_MODEM 切换为 HP_ACTIVE 时使能数据搬移。

0：禁能

1：使能

(R/W)

Register 13.7. PMU_HP_ACTIVE_BACKUP_CLK_REG (0x0020)



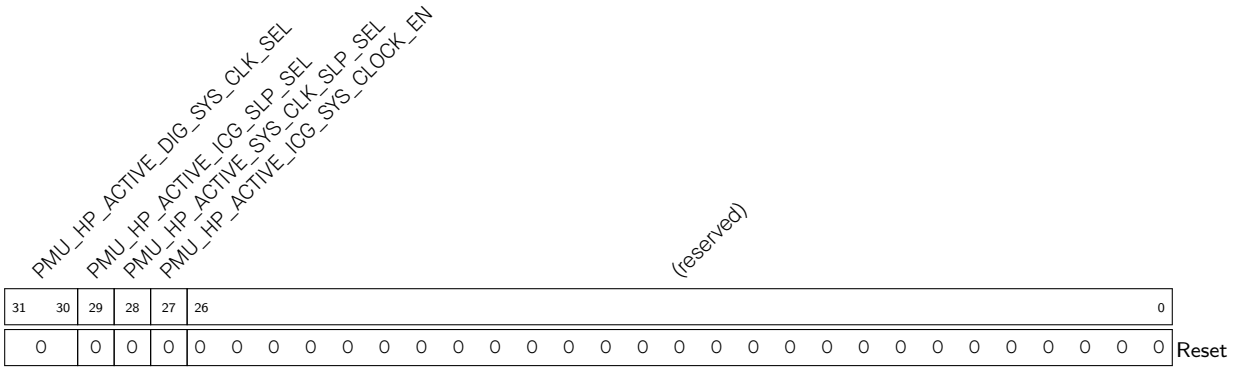
PMU_HP_ACTIVE_BACKUP_ICG_FUNC_EN 配置目标状态为 HP_ACTIVE 时不同外设功能时钟的开关情况，不同位代表不同外设，详见章节 9 复位和时钟 > 表 9.2-8。

0：关闭

1：打开

(R/W)

Register 13.8. PMU_HP_ACTIVE_SYSCLK_REG (0x0024)



PMU_HP_ACTIVE_ICG_SYS_CLOCK_EN 配置是否在 HP_ACTIVE 状态下打开 HP_ROOT_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_ACTIVE_SYS_CLK_SLP_SEL 配置是否在 HP_ACTIVE 状态下由 PMU 控制时钟源。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_ACTIVE_ICG_SLP_SEL 配置是否在 HP_ACTIVE 状态下是否由 PMU 控制各时钟门控的开关。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_ACTIVE_DIG_SYS_CLK_SEL 配置 HP_ACTIVE 状态下 HP_ROOT_CLK 的时钟源。

0：时钟源为 XTAL

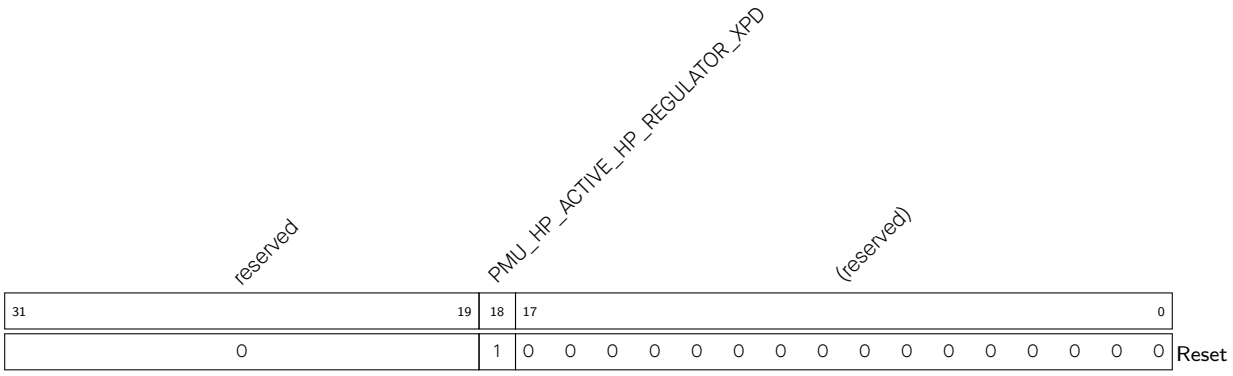
1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

Register 13.9. PMU_HP_ACTIVE_HP_REGULATOR0_REG (0x0028)



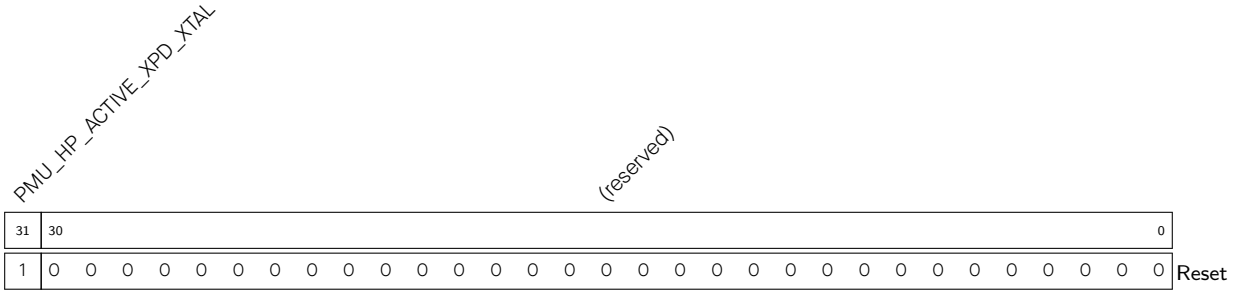
PMU_HP_ACTIVE_HP_REGULATOR_XPD 配置是否在 HP_ACTIVE 状态下打开 HP 系统调压器。

0：关闭 HP 系统调压器

1：打开 HP 系统调压器

(R/W)

Register 13.10. PMU_HP_ACTIVE_XTAL_REG (0x0030)



PMU_HP_ACTIVE_XPD_XTAL 配置是否在 HP_ACTIVE 状态下打开 XTAL_CLK 的模拟源。

0：关闭

1：打开

(R/W)

Register 13.11. PMU_HP_MODEM_DIG_POWER_REG (0x0034)

Bits	31	30	29	28	27	26–23	22	21	20–0
Field	PMU_HP_MODEM_PD_TOP_PD_EN	PMU_HP_MODEM_PD_HP_AON_PD_EN	PMU_HP_MODEM_PD_HP_CPU_PD_EN	(reserved)	PMU_HP_MODEM_PD_HP_WIFI_PD_EN	(reserved)	PMU_HP_MODEM_HP_MEM_DSLP	PMU_HP_MODEM_VDD_SPI_PD_EN	(reserved)
Reset	0	0	0	0	0	0 0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

PMU_HP_MODEM_VDD_SPI_PD_EN 配置是否在 HP_MODEM 状态下关闭外部 flash。
0：打开
1：关闭
(R/W)

PMU_HP_MODEM_HP_MEM_DSLP 配置是否在 HP_MODEM 状态下让 Internal SRAMx 进入 Deep-sleep。
0：不进入 Deep-sleep
1：进入 Deep-sleep
(R/W)

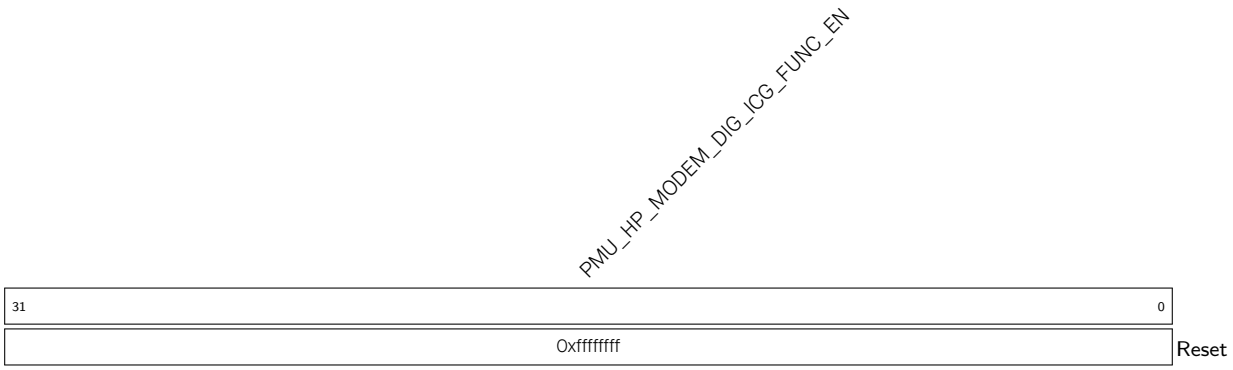
PMU_HP_MODEM_PD_HP_WIFI_PD_EN 配置是否在 HP_MODEM 状态下关闭 Modem 电源域。
0：打开
1：关闭
(R/W)

PMU_HP_MODEM_PD_HP_CPU_PD_EN 配置是否在 HP_MODEM 状态下关闭 CPU 电源域。
0：打开
1：关闭
(R/W)

PMU_HP_MODEM_PD_HP_AON_PD_EN 配置是否在 HP_MODEM 状态下关闭 Modem Power 电源域。
0：打开
1：关闭
(R/W)

PMU_HP_MODEM_PD_TOP_PD_EN 配置是否在 HP_MODEM 状态下关闭 Peripherals 电源域。
0：打开
1：关闭
(R/W)

Register 13.12. PMU_HP_MODEM_ICG_HP_FUNC_REG (0x0038)



PMU_HP_MODEM_DIG_ICG_FUNC_EN 配置 HP_MODEM 状态下高性能系统外设功能时钟的开关。

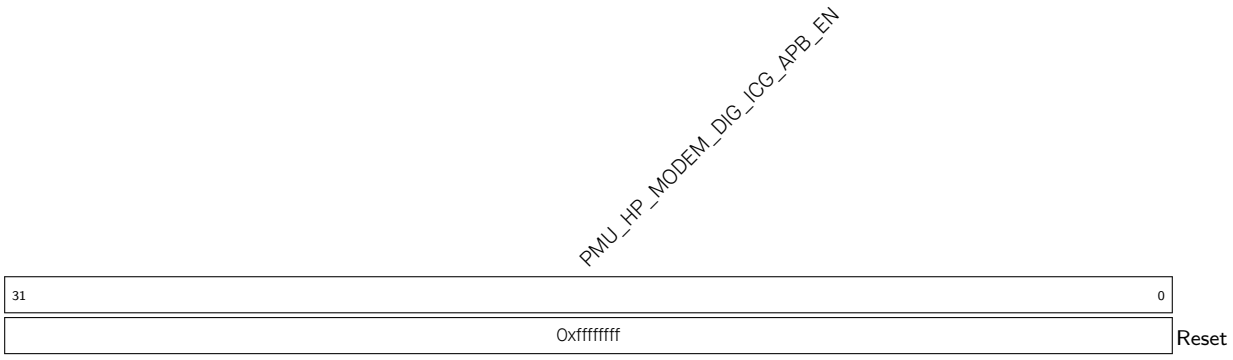
具体配置请见章节 9 复位和时钟 > 表 9.2-8。

0：关闭

1：打开

(R/W)

Register 13.13. PMU_HP_MODEM_ICG_HP_APB_REG (0x003C)



PMU_HP_MODEM_DIG_ICG_APB_EN 配置 HP_MODEM 状态下高性能系统外设 APB 时钟的开关。

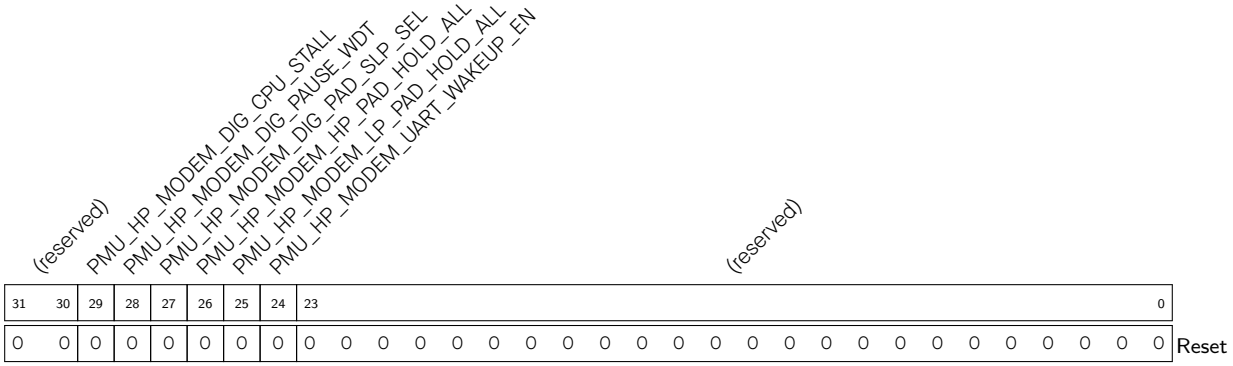
具体配置请见章节 9 复位和时钟 > Table 9.2-7。

0：关闭

1：打开

(R/W)

Register 13.14. PMU_HP_MODEM_HP_SYS_CNTL_REG (0x0044)



PMU_HP_MODEM_UART_WAKEUP_EN 配置是否在 HP_MODEM 状态下使能 UART 唤醒功能。

0：关闭唤醒功能

1：使能唤醒功能

(R/W)

PMU_HP_MODEM_LP_PAD_HOLD_ALL 配置是否在 HP_MODEM 状态下保持 LP GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_MODEM_HP_PAD_HOLD_ALL 配置是否在 HP_MODEM 状态下保持 GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_MODEM_DIG_PAD_SLP_SEL 配置是否在 HP_MODEM 状态下使用 GPIO 的 Light-sleep 模式配置。

0：正常工作模式的配置

1：Light-sleep 模式配置，详见章节 *8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* > 小节 *8.8 Light-sleep 模式管脚功能*。

(R/W)

PMU_HP_MODEM_DIG_PAUSE_WDT 配置是否在 HP_MODEM 状态下暂停看门狗。

0：不暂停

1：暂停

(R/W)

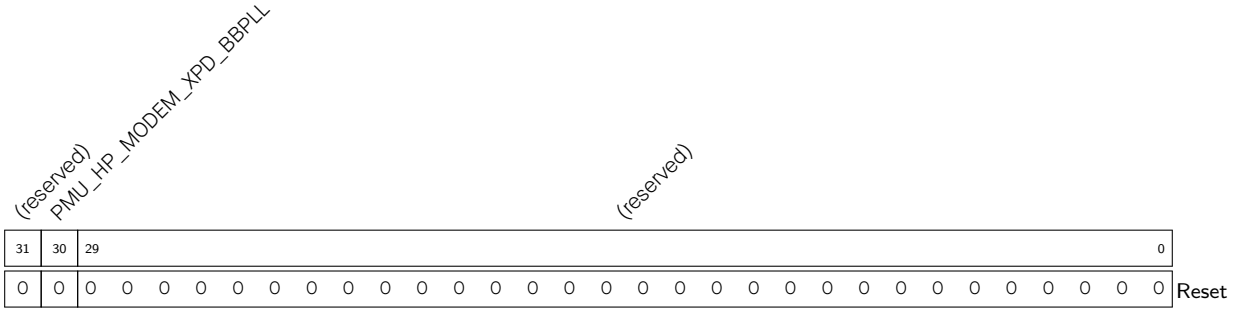
PMU_HP_MODEM_DIG_CPU_STALL 配置是否 HP_MODEM 状态下暂停 CPU。

0：不暂停

1：暂停

(R/W)

Register 13.15. PMU_HP_MODEM_HP_CK_POWER_REG (0x0048)



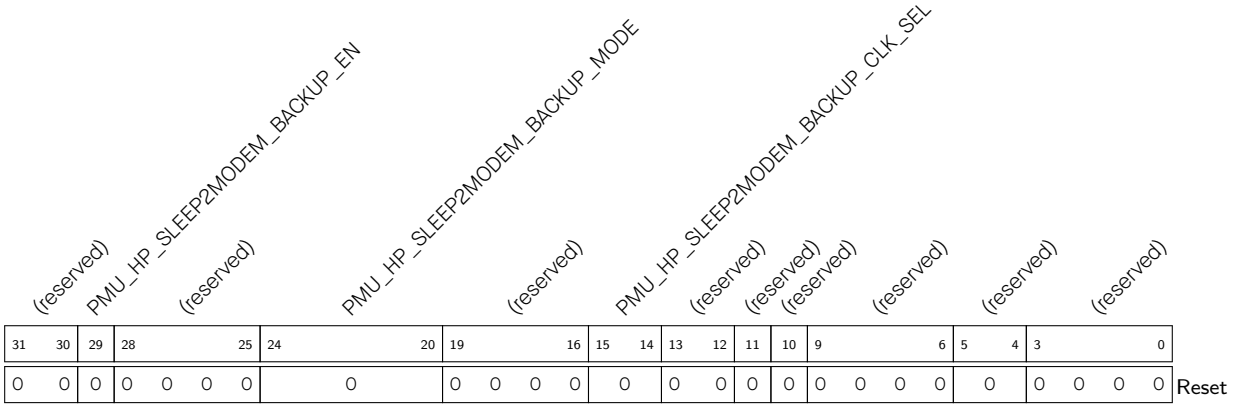
PMU_HP_MODEM_XPD_BBPLL 配置是否在 HP_MODEM 状态下打开 PLL_CLK 时钟。

0：关闭

1：打开

(R/W)

Register 13.16. PMU_HP_MODEM_BACKUP_REG (0x0050)



PMU_HP_SLEEP2MODEM_BACKUP_CLK_SEL 配置备份模块从 HP_SLEEP 切换为 HP_MODEM 时的功能时钟源。

0：时钟源为 XTAL

1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

PMU_HP_SLEEP2MODEM_BACKUP_MODE 配置从 HP_SLEEP 切换为 HP_MODEM 时数据搬移的方向和链表指针。

最高位：

0：存储器到外设

1：外设到存储器

低四位：选择链表指针，指针在链表中设置。

(R/W)

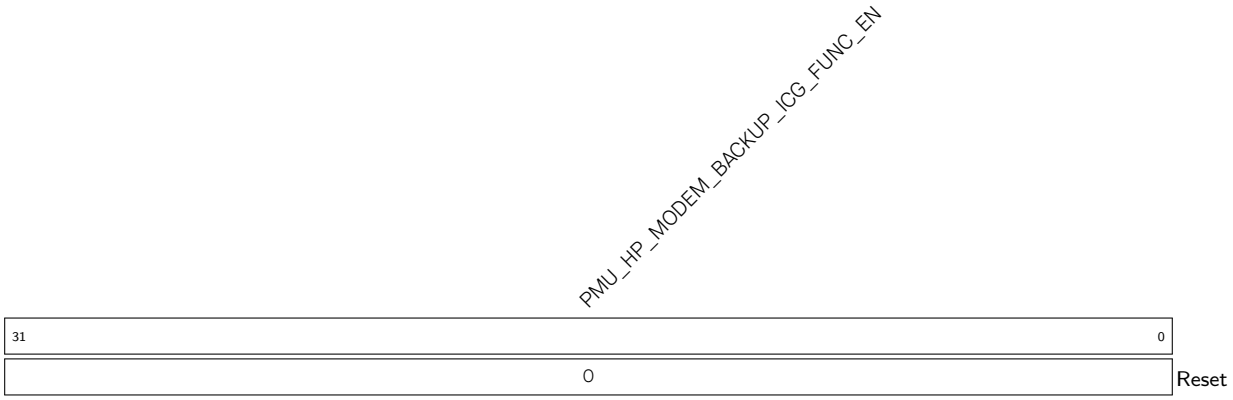
PMU_HP_SLEEP2MODEM_BACKUP_EN 配置是否在从 HP_SLEEP 切换为 HP_MODEM 时使能数据搬移。

0：禁能

1：使能

(R/W)

Register 13.17. PMU_HP_MODEM_BACKUP_CLK_REG (0x0054)



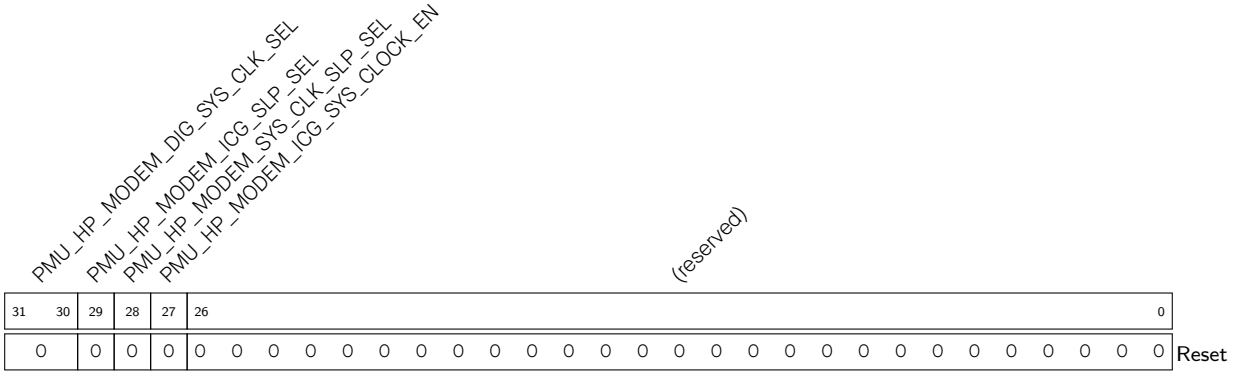
PMU_HP_MODEM_BACKUP_ICG_FUNC_EN 配置目标状态为 HP_MODEM 时不同外设功能时钟的开关情况，不同位代表不同外设，详见章节 9 复位和时钟 > 表 9.2-8。

0：关闭

1：打开

(R/W)

Register 13.18. PMU_HP_MODEM_SYSCLK_REG (0x0058)



PMU_HP_MODEM_ICG_SYS_CLOCK_EN 配置是否在 HP_MODEM 状态下打开 HP_ROOT_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_MODEM_SYS_CLK_SLP_SEL 配置是否在 HP_MODEM 状态下由 PMU 控制时钟源。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_MODEM_ICG_SLP_SEL 配置是否在 HP_MODEM 状态下是否由 PMU 控制各时钟门控的开关。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_MODEM_DIG_SYS_CLK_SEL 配置 HP_MODEM 状态下 HP_ROOT_CLK 的时钟源。

0：时钟源为 XTAL

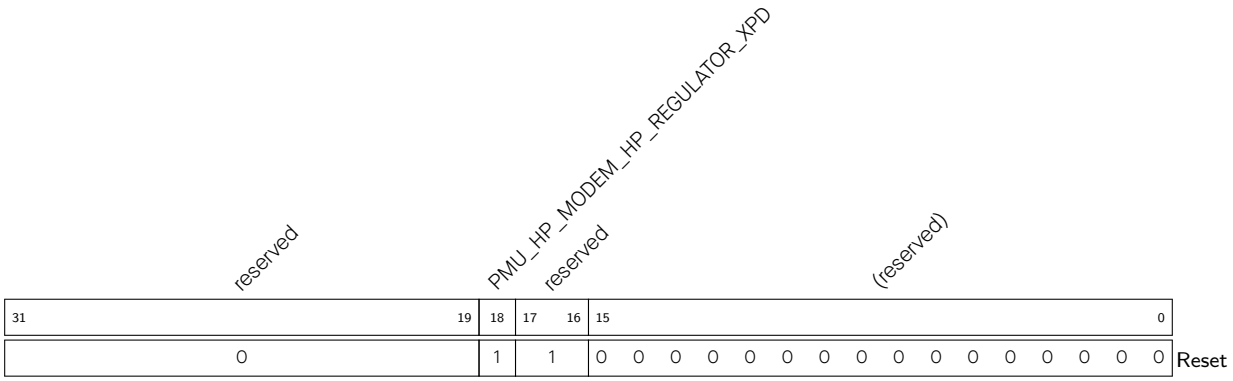
1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

Register 13.19. PMU_HP_MODEM_HP_REGULATOR0_REG (0x005C)



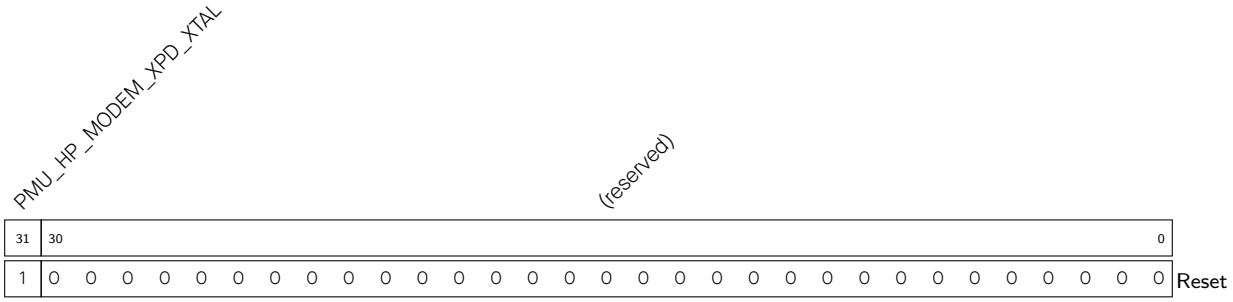
PMU_HP_MODEM_HP_REGULATOR_XPD 配置是否在 HP_MODEM 状态下打开 HP 系统调压器。

0：关闭 HP 系统调压器

1：打开 HP 系统调压器。

(R/W)

Register 13.20. PMU_HP_MODEM_XTAL_REG (0x0064)



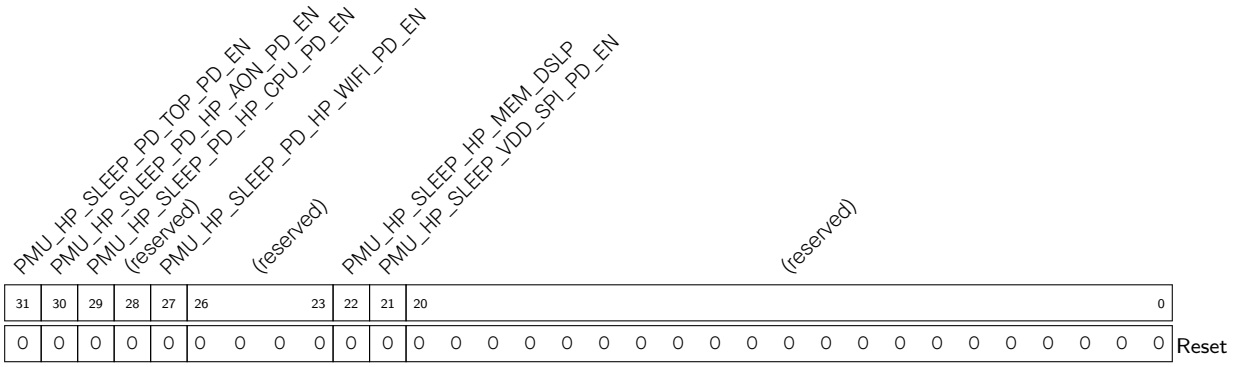
PMU_HP_MODEM_XPD_XTAL 配置是否在 HP_MODEM 状态下打开 XTAL_CLK 的模拟源。

0：关闭

1：打开

(R/W)

Register 13.21. PMU_HP_SLEEP_DIG_POWER_REG (0x0068)



PMU_HP_SLEEP_VDD_SPI_PD_EN 配置是否在 HP_SLEEP 状态下关闭外部 flash。

0：打开

1：关闭

(R/W)

PMU_HP_SLEEP_HP_MEM_DSLP 配置是否在 HP_SLEEP 状态下让 Internal SRAMx 进入 Deep-sleep。

0：不进入 Deep-sleep

1：进入 Deep-sleep

(R/W)

PMU_HP_SLEEP_PD_HP_WIFI_PD_EN 配置是否在 HP_SLEEP 状态下关闭 Modem 电源域。

0：打开

1：关闭

(R/W)

PMU_HP_SLEEP_PD_HP_CPU_PD_EN 配置是否在 HP_SLEEP 状态下关闭 CPU 电源域。

0：打开

1：关闭

(R/W)

PMU_HP_SLEEP_PD_HP_AON_PD_EN 配置是否在 HP_SLEEP 状态下关闭 Modem Power 电源域。

0：打开

1：关闭

(R/W)

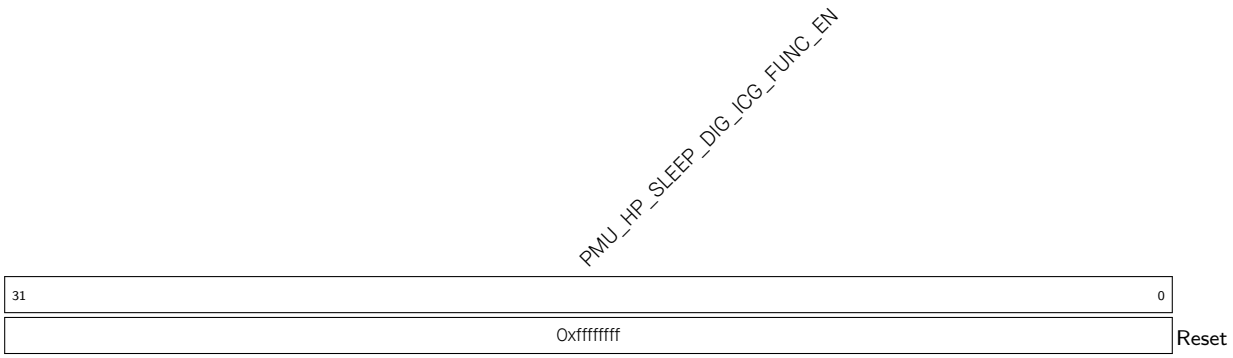
PMU_HP_SLEEP_PD_TOP_PD_EN 配置是否在 HP_SLEEP 状态下关闭 Peripherals 电源域。

0：打开

1：关闭

(R/W)

Register 13.22. PMU_HP_SLEEP_ICG_HP_FUNC_REG (0x006C)

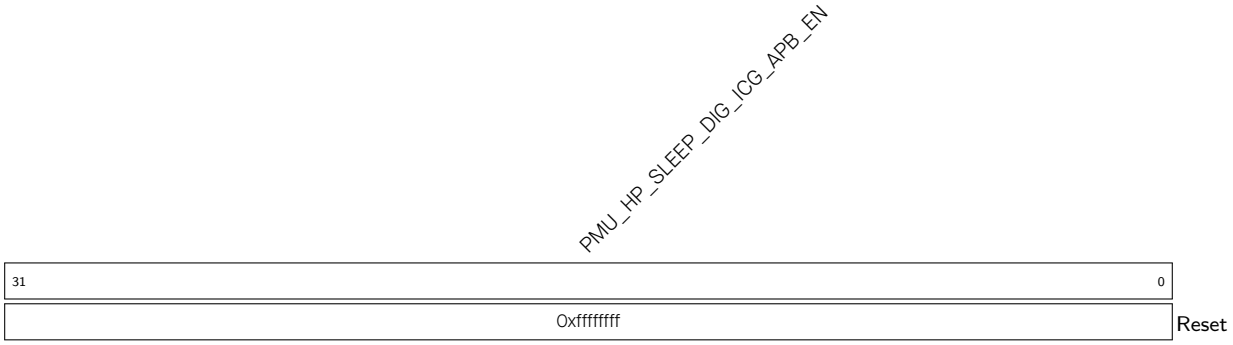


PMU_HP_SLEEP_DIG_ICG_FUNC_EN 配置 HP_SLEEP 状态下高性能系统外设功能时钟的开关。具体配置请见章节 9 复位和时钟 > 表 9.2-8。

0：关闭

1：打开

Register 13.23. PMU_HP_SLEEP_ICG_HP_APB_REG (0x0070)



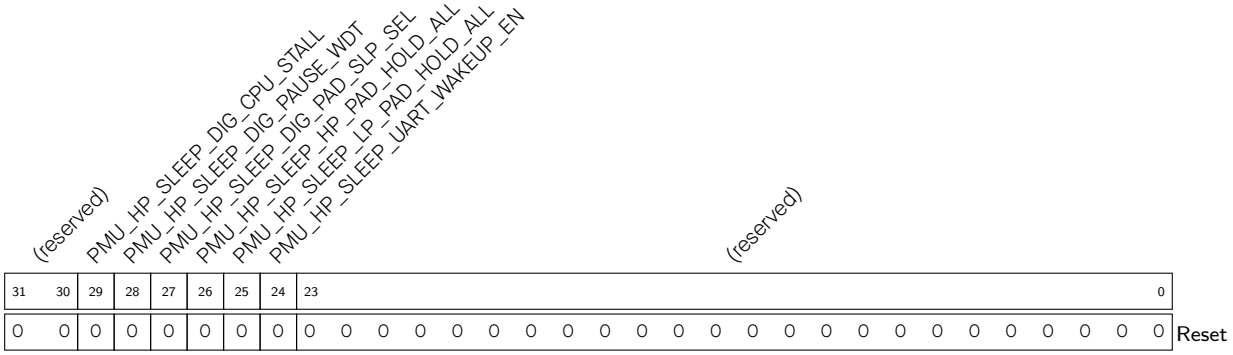
PMU_HP_SLEEP_DIG_ICG_APB_EN 配置 HP_SLEEP 状态下高性能系统外设 APB 时钟的开关。具体配置请见章节 9 复位和时钟 > Table 9.2-7。

0：关闭

1：打开

(R/W)

Register 13.24. PMU_HP_SLEEP_HP_SYS_CNTL_REG (0x0078)



PMU_HP_SLEEP_UART_WAKEUP_EN 配置是否在 HP_SLEEP 状态下使能 UART 唤醒功能。

0：关闭唤醒功能

1：使能唤醒功能

(R/W)

PMU_HP_SLEEP_LP_PAD_HOLD_ALL 配置是否在 HP_SLEEP 状态下保持 LP GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_SLEEP_HP_PAD_HOLD_ALL 配置是否在 HP_SLEEP 状态下保持 GPIO 的配置。

0：不保持

1：保持

(R/W)

PMU_HP_SLEEP_DIG_PAD_SLP_SEL 配置是否在 HP_SLEEP 状态下使用 GPIO 的 Light-sleep 模式配置。

0：正常工作模式的配置

1：Light-sleep 模式配置，详见章节 *8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)* > 小节 *8.8 Light-sleep 模式管脚功能*。

(R/W)

PMU_HP_SLEEP_DIG_PAUSE_WDT 配置是否在 HP_SLEEP 状态下暂停看门狗。

0：不暂停

1：暂停

(R/W)

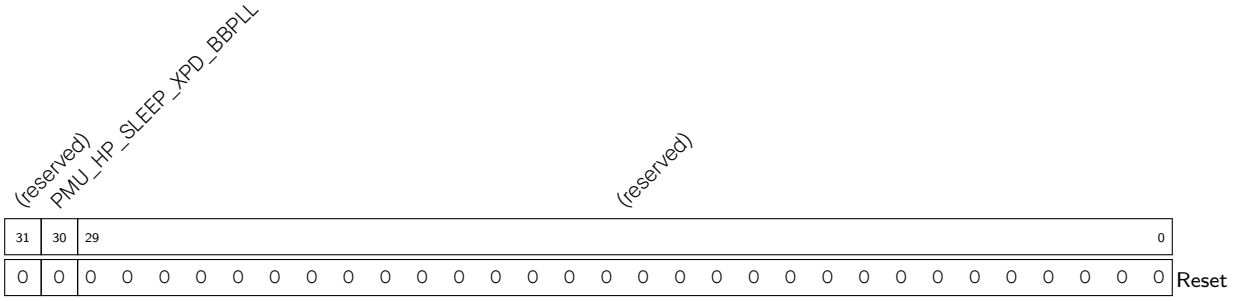
PMU_HP_SLEEP_DIG_CPU_STALL 配置是否在 HP_SLEEP 状态下暂停 CPU。

0：不暂停

1：暂停

(R/W)

Register 13.25. PMU_HP_SLEEP_HP_CK_POWER_REG (0x007C)



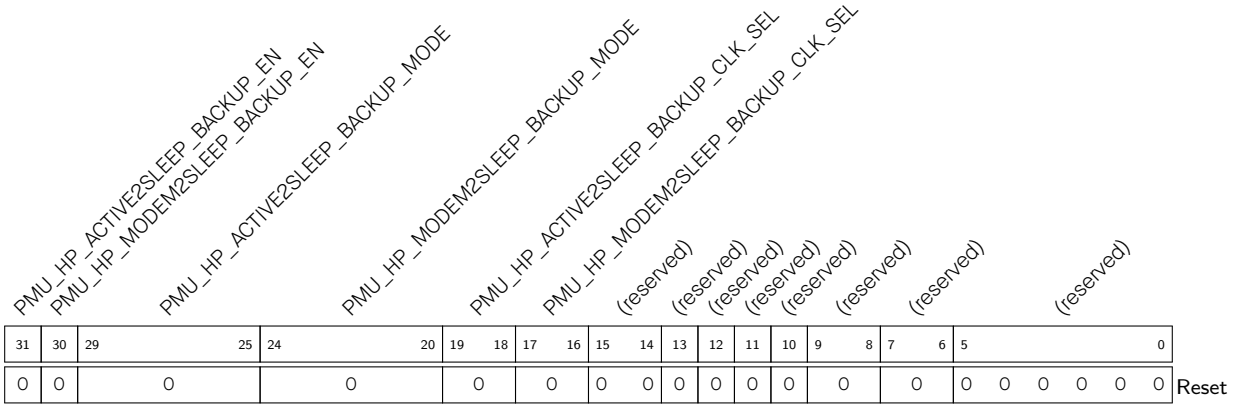
PMU_HP_SLEEP_XPD_BBPLL 配置是否在 HP_SLEEP 状态下打开 PLL_CLK 时钟。

0：关闭

1：打开

(R/W)

Register 13.26. PMU_HP_SLEEP_BACKUP_REG (0x0084)



PMU_HP_MODEM2SLEEP_BACKUP_CLK_SEL 配置备份模块从 HP_MODEM 切换为 HP_SLEEP 时的功能时钟源。

0：时钟源为 XTAL

1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

PMU_HP_ACTIVE2SLEEP_BACKUP_CLK_SEL 配置备份模块从 HP_ACTIVE 切换为 HP_SLEEP 时的功能时钟源。配置同上。(R/W)

PMU_HP_MODEM2SLEEP_BACKUP_MODE 配置从 HP_MODEM 切换为 HP_SLEEP 时数据搬移的方向和链表指针。

最高位：

0：存储器到外设

1：外设到存储器

低四位：选择链表指针，指针在链表中设置。

(R/W)

PMU_HP_ACTIVE2SLEEP_BACKUP_MODE 配置从 HP_ACTIVE 切换为 HP_SLEEP 时数据搬移的方向和链表指针。配置同上。(R/W)

PMU_HP_MODEM2SLEEP_BACKUP_EN 配置是否在从 HP_MODEM 切换为 HP_SLEEP 时使能数据搬移。

0：禁能

1：使能

(R/W)

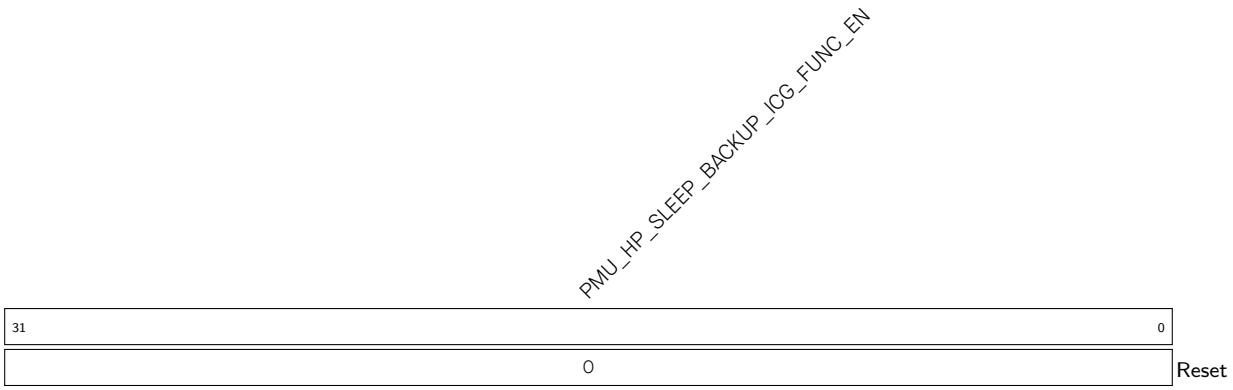
PMU_HP_ACTIVE2SLEEP_BACKUP_EN 配置是否在从 HP_ACTIVE 切换为 HP_SLEEP 时使能数据搬移。

0：禁能

1：使能

(R/W)

Register 13.27. PMU_HP_SLEEP_BACKUP_CLK_REG (0x0088)



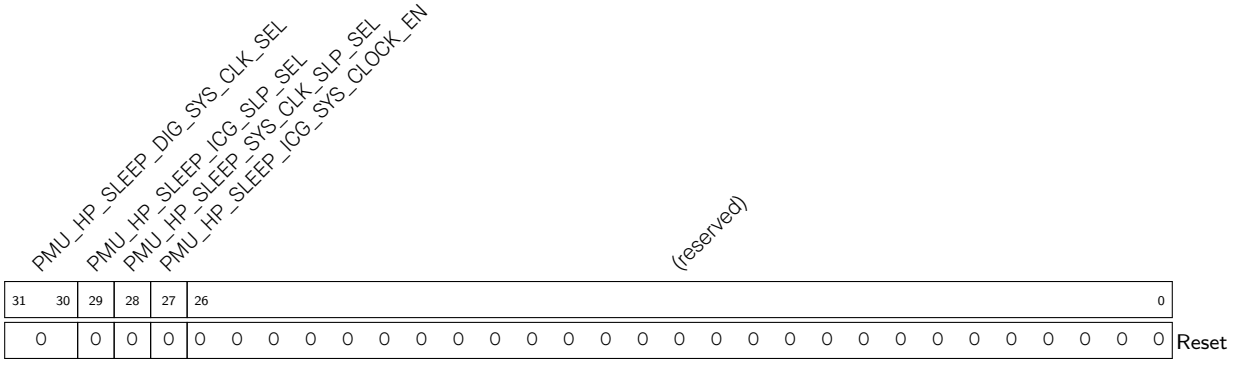
PMU_HP_SLEEP_BACKUP_ICG_FUNC_EN 配置目标状态为 HP_SLEEP 时不同外设功能时钟的开关情况，不同位代表不同外设，详见章节 9 复位和时钟 > 表 9.2-8。

0：关闭

1：打开

(R/W)

Register 13.28. PMU_HP_SLEEP_SYSCLK_REG (0x008C)



PMU_HP_SLEEP_ICG_SYS_CLOCK_EN 配置是否在 HP_SLEEP 状态下打开 HP_ROOT_CLK。

0：关闭

1：打开

(R/W)

PMU_HP_SLEEP_SYS_CLK_SLP_SEL 配置是否在 HP_SLEEP 状态下由 PMU 控制时钟源。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_SLEEP_ICG_SLP_SEL 配置是否在 HP_SLEEP 状态下是否由 PMU 控制各时钟门控的开关。

0：由 PCR 寄存器控制

1：由 PMU 控制

(R/W)

PMU_HP_SLEEP_DIG_SYS_CLK_SEL 配置 HP_SLEEP 状态下 HP_ROOT_CLK 的时钟源。

0：时钟源为 XTAL

1：时钟源为 PLL_CLK

2：时钟源为 RC_FAST_CLK

3：无效配置

(R/W)

Register 13.29. PMU_HP_SLEEP_HP_REGULATOR0_REG (0x0090)



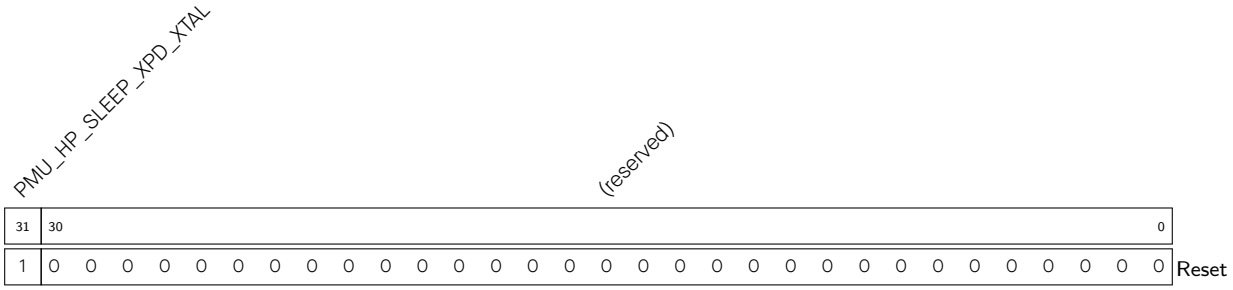
PMU_HP_SLEEP_HP_REGULATOR_XPD 配置是否在 HP_SLEEP 状态下打开 HP 系统调压器。

0：关闭 HP 系统调压器

1：打开 HP 系统调压器。

(R/W)

Register 13.30. PMU_HP_SLEEP_XTAL_REG (0x0098)



PMU_HP_SLEEP_XPD_XTAL 配置是否在 HP_SLEEP 状态下打开 XTAL_CLK 的模拟源。

0：关闭

1：打开

(R/W)

Register 13.31. PMU_HP_SLEEP_LP_REGULATOR0_REG (0x009C)

(reserved)	(reserved)	PMU_HP_SLEEP_LP_REGULATOR_XPD	(reserved)	(reserved)	
31 27	26 23	22	21	20 0	
24	12	1	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

PMU_HP_SLEEP_LP_REGULATOR_XPD 配置是否在 HP_SLEEP 状态下打开 LP 系统调压器。

0：关闭 LP 系统调压器

1：打开 LP 系统调压器

(R/W)

Register 13.32. PMU_HP_SLEEP_LP_DIG_POWER_REG (0x00A8)

PMU_HP_SLEEP_PD_LP_PERI_PD_EN	PMU_HP_SLEEP_LP_MEM_DSLP	(reserved)	
31	30	29 0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

PMU_HP_SLEEP_LP_MEM_DSLP 配置是否在 HP_ACTIVE/HP_MODEM/HP_SLEEP 状态下使低功耗管理系统的 16 KB SRAM 进入 Memory Deep-sleep 模式。

0：不使能

1：使能

(R/W)

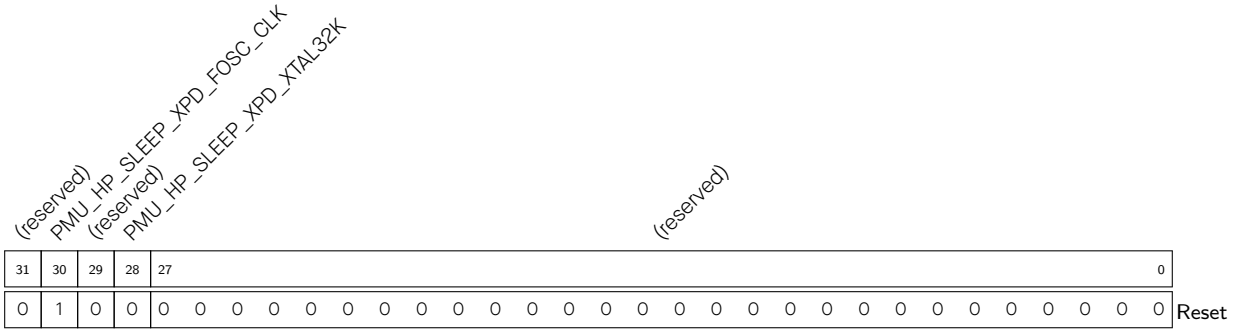
PMU_HP_SLEEP_PD_LP_PERI_PD_EN 配置是否在 HP_ACTIVE/HP_MODEM/HP_SLEEP 状态下关闭 LP PD Peripherals 电源域。

0：打开

1：关闭

(R/W)

Register 13.33. PMU_HP_SLEEP_LP_CK_POWER_REG (0x00AC)



PMU_HP_SLEEP_XPD_XTAL32K 配置是否在 HP_ACTIVE/ HP_MODEM/ HP_SLEEP 状态下打开 XTAL32K_CLK。

0：关闭

1：打开

(R/W)

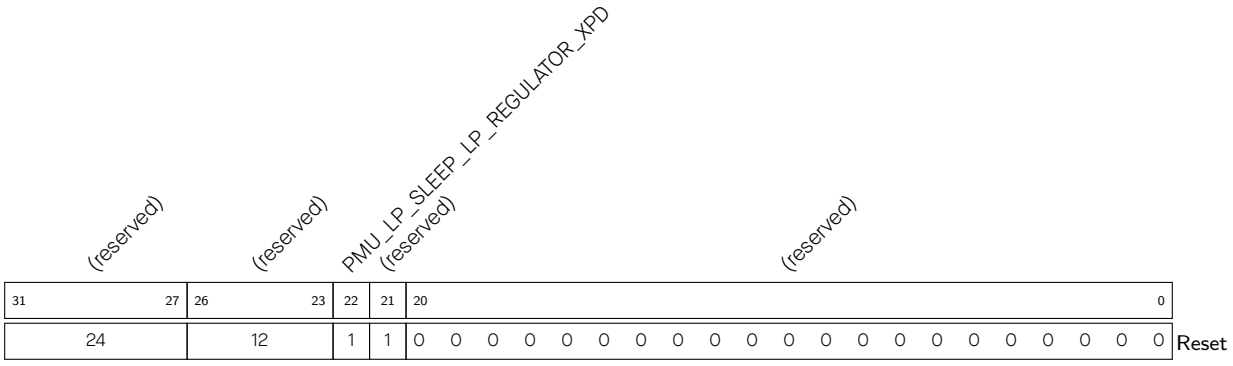
PMU_HP_SLEEP_XPD_FOSC_CLK 配置是否在 HP_ACTIVE/HP_MODEM/HP_SLEEP 状态下打开 RC_FAST_CLK。

0：关闭

1：打开

(R/W)

Register 13.34. PMU_LP_SLEEP_LP_REGULATOR0_REG (0x00B4)



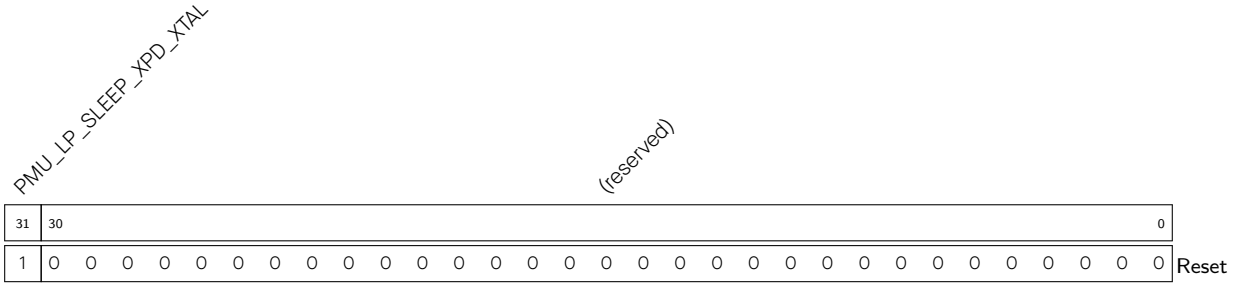
PMU_LP_SLEEP_LP_REGULATOR_XPD 配置是否在 LP_SLEEP 状态下打开 LP 系统调压器。

0：关闭 LP 系统调压器

1：打开 LP 系统调压器

(R/W)

Register 13.35. PMU_LP_SLEEP_XTAL_REG (0x00BC)



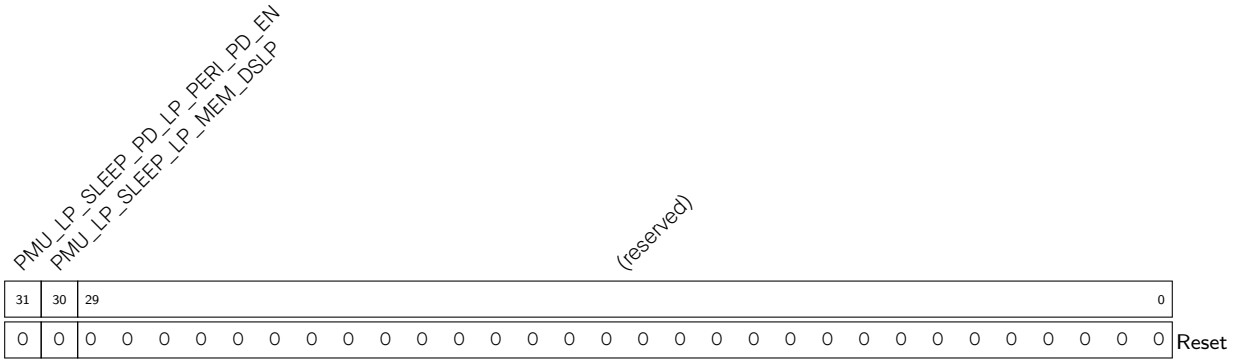
PMU_LP_SLEEP_XPD_XTAL 配置是否在 LP_SLEEP 状态下打开 XTAL_CLK 的模拟源。

0：关闭

1：打开

(R/W)

Register 13.36. PMU_LP_SLEEP_LP_DIG_POWER_REG (0x00C0)



PMU_LP_SLEEP_LP_MEM_DSLP 配置是否在 LP_SLEEP 状态下使低功耗管理系统的 16 KB SRAM 进入 Memory Deep-sleep 模式。

0：不使能

1：使能

(R/W)

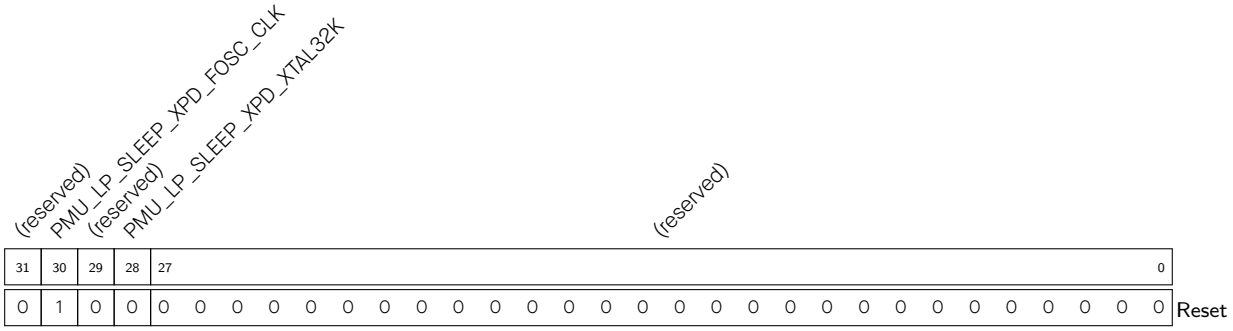
PMU_LP_SLEEP_PD_LP_PERI_PD_EN 配置是否在 LP_SLEEP 状态下关闭 LP PD Peripherals 电源域。

0：打开

1：关闭

(R/W)

Register 13.37. PMU_LP_SLEEP_LP_CK_POWER_REG (0x00C4)



PMU_LP_SLEEP_XPD_XTAL32K 配置是否在 LP_SLEEP 状态下打开 XTAL32K_CLK。

0：关闭

1：打开

(R/W)

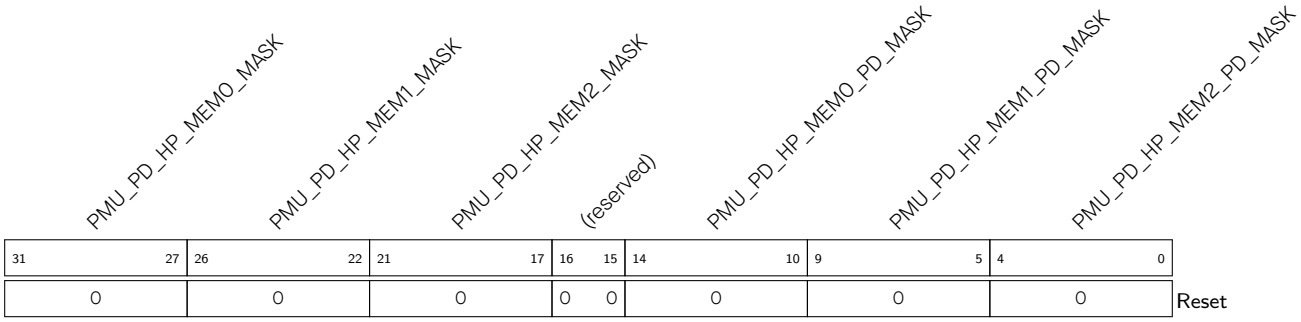
PMU_LP_SLEEP_XPD_FOSC_CLK 配置是否在 LP_SLEEP 状态下打开 RC_FAST_CLK。

0：关闭

1：打开

(R/W)

Register 13.38. PMU_POWER_PD_MEM_MASK_REG (0x0114)



PMU_PD_HP_MEM2_PD_MASK 配置 Internal SRAM2 与 Peripherals 电源域是否同时上下电。

0：同时上下电

1：Internal SRAM2 在 Peripherals 掉电时可保持上电

(R/W)

PMU_PD_HP_MEM1_PD_MASK 配置 Internal SRAM1 与 Peripherals 电源域是否同时上下电。

0：同时上下电

1：Internal SRAM1 在 Peripherals 掉电时可保持上电

(R/W)

PMU_PD_HP_MEM0_PD_MASK 配置 Internal SRAM0 与 Peripherals 电源域是否同时上下电。

0：同时上下电

1：Internal SRAM0 在 Peripherals 掉电时可保持上电

(R/W)

PMU_PD_HP_MEM2_MASK 配置是否强制打开 Internal SRAM2 电源域。

0：不强制打开

1：强制打开

(R/W)

PMU_PD_HP_MEM1_MASK 配置是否强制打开 Internal SRAM1 电源域。

0：不强制打开

1：强制打开

(R/W)

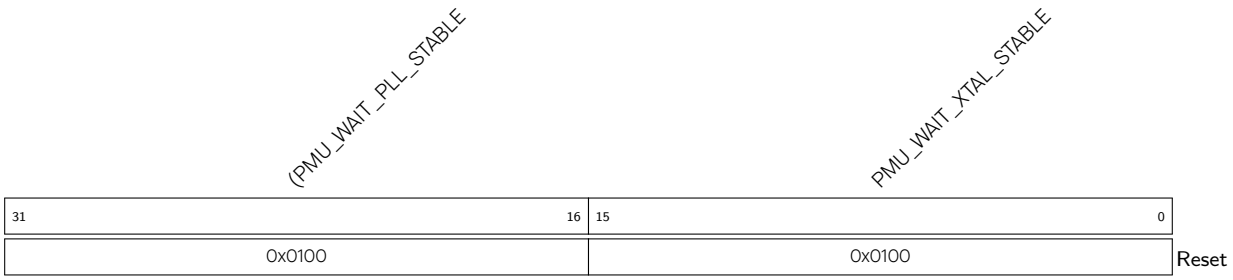
PMU_PD_HP_MEM0_MASK 配置是否强制打开 Internal SRAM0 电源域。

0：不强制打开

1：强制打开

(R/W)

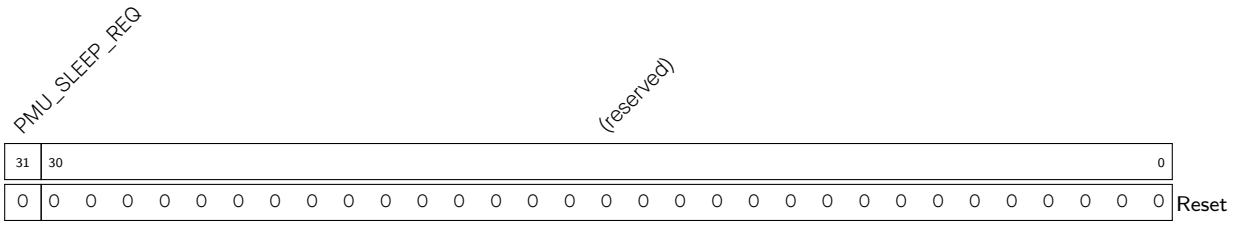
Register 13.39. PMU_POWER_CK_WAIT_CNTL_REG (0x0120)



PMU_WAIT_PLL_STABLE 配置 PLL_CLK 全局门控在多少个 CLK_DYN_FAST_CLK 周期时间后打开。(R/W)

PMU_WAIT_XTAL_STABLE 配置 XTAL_CLK 全局门控在多少个 CLK_DYN_FAST_CLK 周期时间后打开。(R/W)(R/W)

Register 13.40. PMU_SLP_WAKEUP_CNTL0_REG (0x0124)



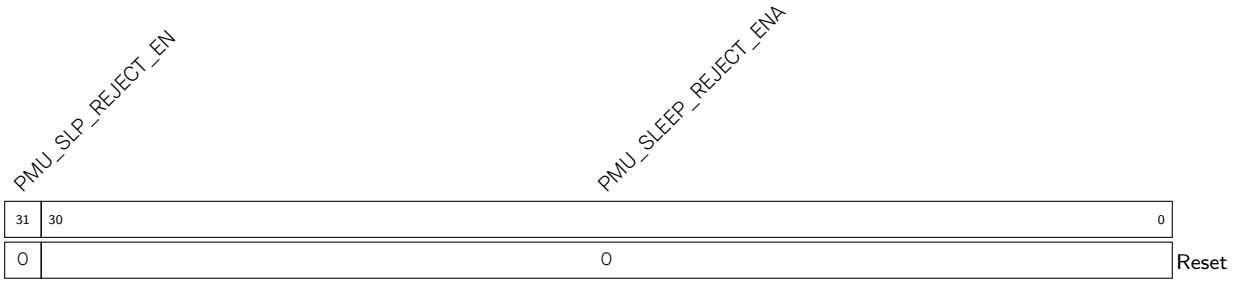
PMU_SLEEP_REQ 配置是否将芯片的 PMU 状态切换至 HP_SLEEP 或 LP_SLEEP。

0：不切换

1：切换，目标状态取决于 LP CPU 的工作状态。

(WT)

Register 13.41. PMU_SLP_WAKEUP_CNTL1_REG (0x0128)



PMU_SLEEP_REJECT_ENA 配置拒绝睡眠信号源。配置值与拒绝睡眠信号源的对应关系见表 13.4-1。(R/W)

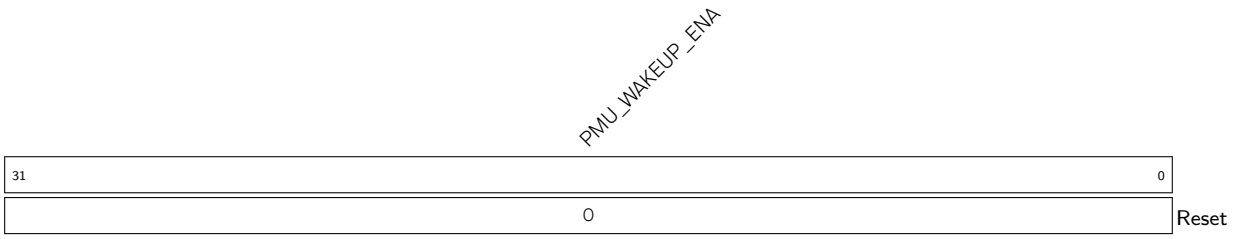
PMU_SLP_REJECT_EN 配置是否使能拒绝睡眠功能。

0：禁能

1：使能

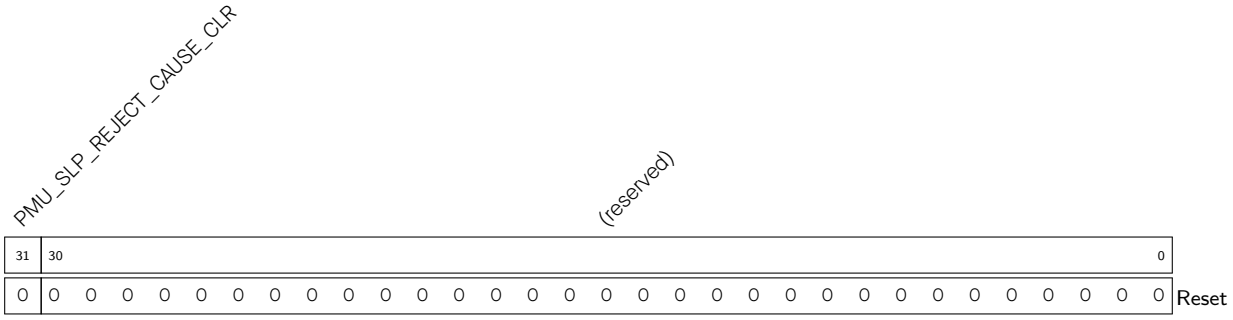
(R/W)

Register 13.42. PMU_SLP_WAKEUP_CNTL2_REG (0x012C)



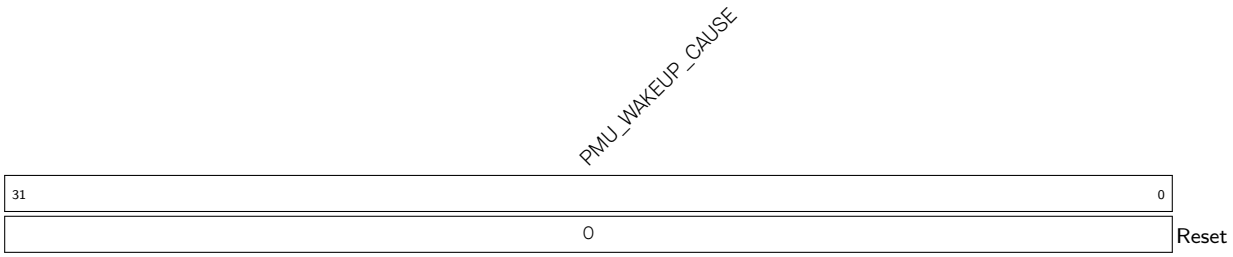
PMU_WAKEUP_ENA 配置唤醒源。配置值与唤醒源的对应关系见表 13.4-1。(R/W)

Register 13.43. PMU_SLP_WAKEUP_CNTL4_REG (0x0134)



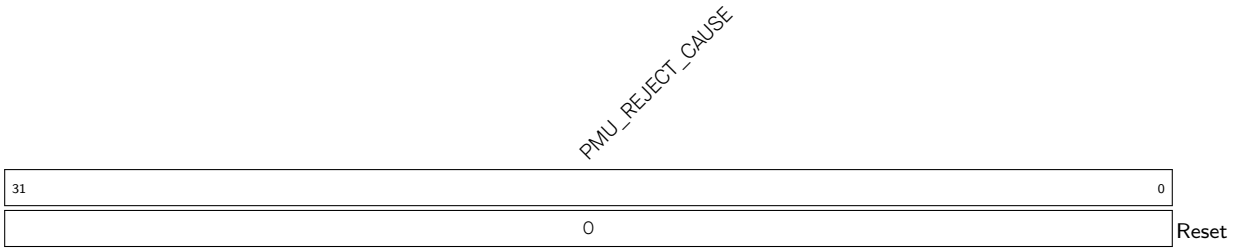
PMU_SLP_REJECT_CAUSE_CLR 写 1 清除 PMU_REJECT_CAUSE。(WT)

Register 13.44. PMU_SLP_WAKEUP_STATUS0_REG (0x0144)



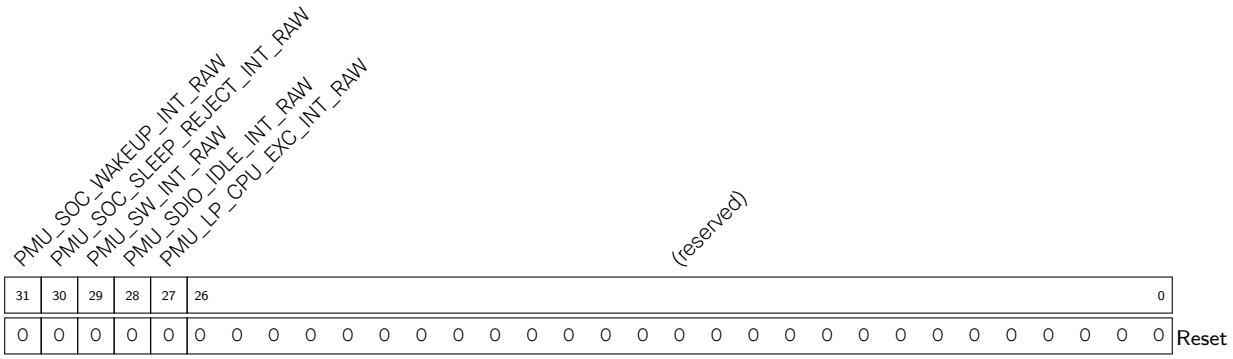
PMU_WAKEUP_CAUSE 指示唤醒源。配置值与唤醒源的对应关系请见表 13.4-1。(RO)

Register 13.45. PMU_SLP_WAKEUP_STATUS1_REG (0x0148)



PMU_REJECT_CAUSE 指示拒绝唤醒的事件源。配置值与唤醒源的对应关系请见表 13.4-1。(RO)

Register 13.46. PMU_INT_RAW_REG (0x0160)



PMU_LP_CPU_EXC_INT_RAW PMU_LP_CPU_EXC_INT 的原始中断状态。(R/WTC/SS)

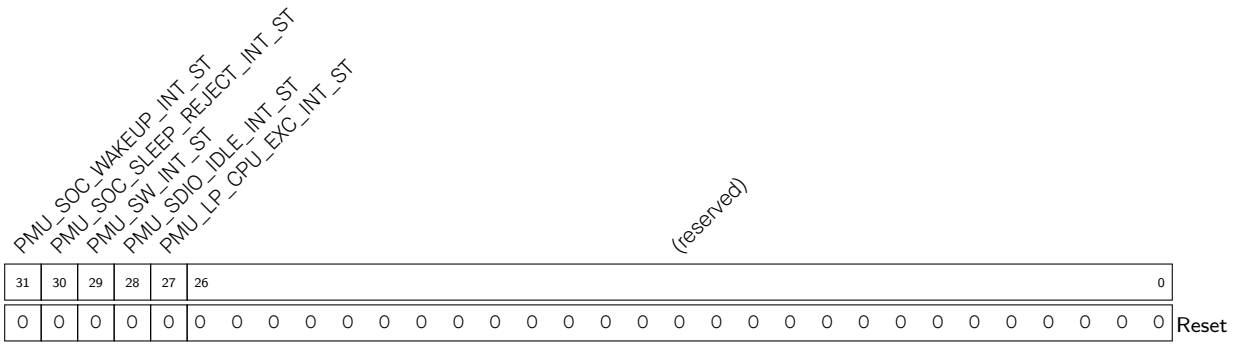
PMU_SDIO_IDLE_INT_RAW PMU_SDIO_IDLE_INT 的原始中断状态。(R/WTC/SS)

PMU_SW_INT_RAW PMU_SW_INT 的原始中断状态。(R/WTC/SS)

PMU_SOC_SLEEP_REJECT_INT_RAW PMU_SOC_SLEEP_REJECT_INT 的原始中断状态。(R/WTC/SS)

PMU_SOC_WAKEUP_INT_RAW PMU_SOC_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 13.47. PMU_HP_INT_ST_REG (0x0164)



PMU_LP_CPU_EXC_INT_ST PMU_LP_CPU_EXC_INT 的屏蔽中断状态。(RO)

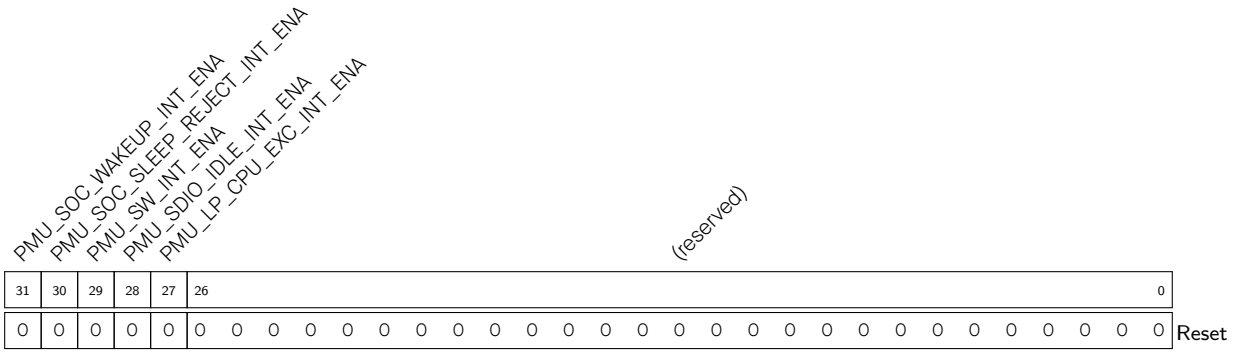
PMU_SDIO_IDLE_INT_ST PMU_SDIO_IDLE_INT 的屏蔽中断状态。(RO)

PMU_SW_INT_ST PMU_SW_INT 的屏蔽中断状态。(RO)

PMU_SOC_SLEEP_REJECT_INT_ST PMU_SOC_SLEEP_REJECT_INT 的屏蔽中断状态。(RO)

PMU_SOC_WAKEUP_INT_ST PMU_SOC_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 13.48. PMU_HP_INT_ENA_REG (0x0168)



PMU_LP_CPU_EXC_INT_ENA 写 1 使能 PMU_LP_CPU_EXC_INT。(R/W)

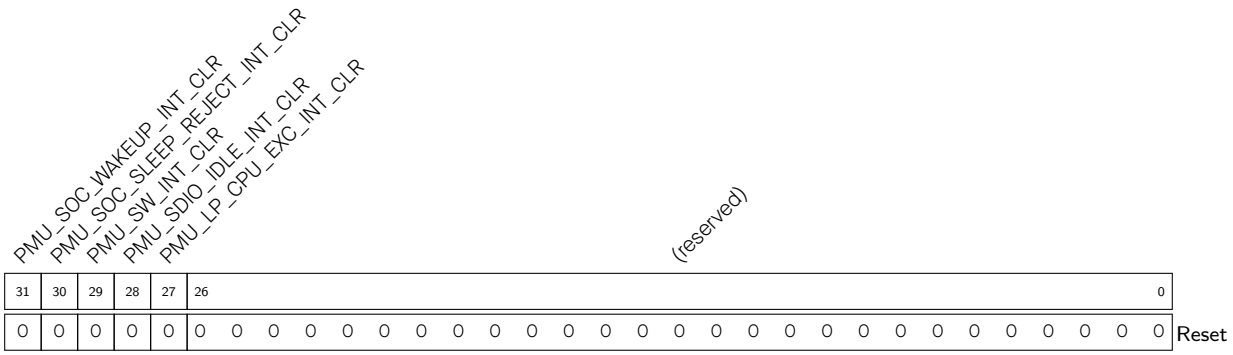
PMU_SDIO_IDLE_INT_ENA 写 1 使能 PMU_SDIO_IDLE_INT。(R/W)

PMU_SW_INT_ENA 写 1 使能 PMU_SW_INT。(R/W)

PMU_SOC_SLEEP_REJECT_INT_ENA 写 1 使能 PMU_SOC_SLEEP_REJECT_INT。(R/W)

PMU_SOC_WAKEUP_INT_ENA 写 1 使能 PMU_SOC_WAKEUP_INT。(R/W)

Register 13.49. PMU_HP_INT_CLR_REG (0x016C)



PMU_LP_CPU_EXC_INT_CLR 写 1 清除 PMU_LP_CPU_EXC_INT。(WT)

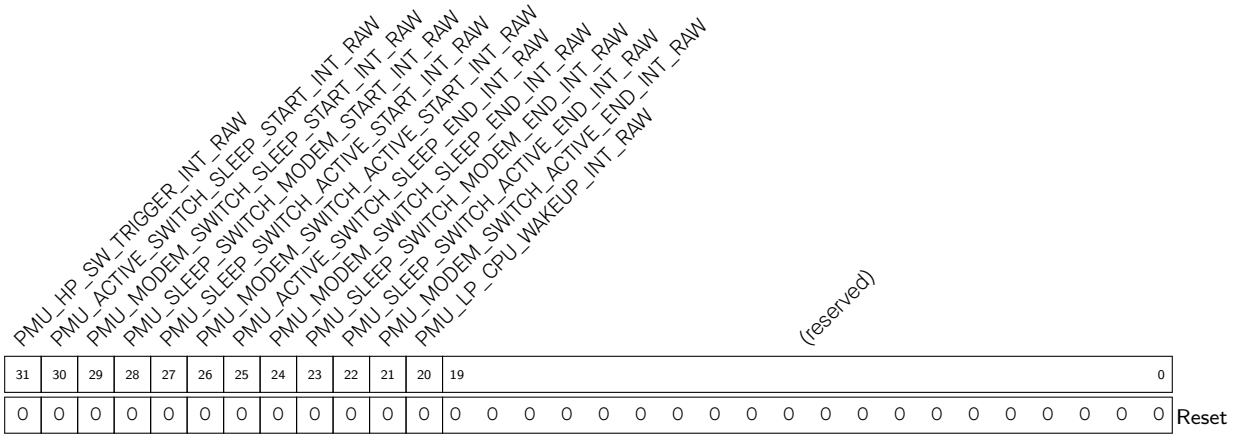
PMU_SDIO_IDLE_INT_CLR 写 1 清除 PMU_SDIO_IDLE_INT。(WT)

PMU_SW_INT_CLR 写 1 清除 PMU_SW_INT。(WT)

PMU_SOC_SLEEP_REJECT_INT_CLR 写 1 清除 PMU_SOC_SLEEP_REJECT_INT。(WT)

PMU_SOC_WAKEUP_INT_CLR 写 1 清除 PMU_SOC_WAKEUP_INT。(WT)

Register 13.50. PMU_LP_INT_RAW_REG (0x0170)



PMU_LP_CPU_WAKEUP_INT_RAW PMU_LP_CPU_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

PMU_MODEM_SWITCH_ACTIVE_END_INT_RAW PMU_MODEM_SWITCH_ACTIVE_END_INT 的原始中断状态。(R/WTC/SS)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_RAW PMU_SLEEP_SWITCH_ACTIVE_END_INT 的原始中断状态。(R/WTC/SS)

PMU_SLEEP_SWITCH_MODEM_END_INT_RAW PMU_SLEEP_SWITCH_MODEM_END_INT 的原始中断状态。(R/WTC/SS)

PMU_MODEM_SWITCH_SLEEP_END_INT_RAW PMU_MODEM_SWITCH_SLEEP_END_INT 的原始中断状态。(R/WTC/SS)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_RAW PMU_ACTIVE_SWITCH_SLEEP_END_INT 的原始中断状态。(R/WTC/SS)

PMU_MODEM_SWITCH_ACTIVE_START_INT_RAW PMU_MODEM_SWITCH_ACTIVE_START_INT 的原始中断状态。(R/WTC/SS)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_RAW PMU_SLEEP_SWITCH_ACTIVE_START_INT 的原始中断状态。(R/WTC/SS)

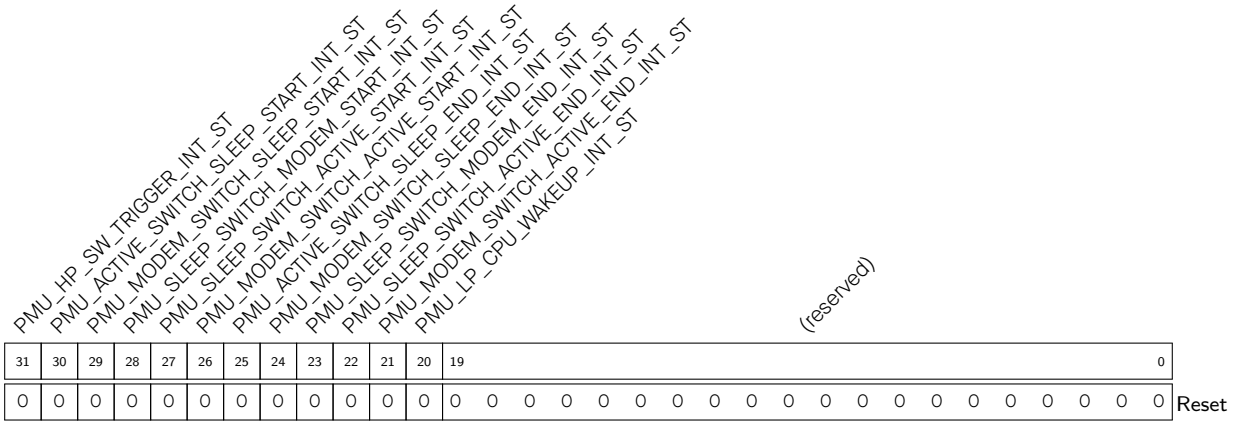
PMU_SLEEP_SWITCH_MODEM_START_INT_RAW PMU_SLEEP_SWITCH_MODEM_START_INT 的原始中断状态。(R/WTC/SS)

PMU_MODEM_SWITCH_SLEEP_START_INT_RAW PMU_MODEM_SWITCH_SLEEP_START_INT 的原始中断状态。(R/WTC/SS)

PMU_ACTIVE_SWITCH_SLEEP_START_INT_RAW PMU_ACTIVE_SWITCH_SLEEP_START_INT 的原始中断状态。(R/WTC/SS)

PMU_HP_SW_TRIGGER_INT_RAW PMU_HP_SW_TRIGGER_INT 的原始中断状态。(R/WTC/SS)

Register 13.51. PMU_LP_INT_ST_REG (0x0174)



PMU_LP_CPU_WAKEUP_INT_ST PMU_LP_CPU_WAKEUP_INT 的屏蔽中断状态。(RO)

PMU_MODEM_SWITCH_ACTIVE_END_INT_ST PMU_MODEM_SWITCH_ACTIVE_END_INT 的屏蔽中断状态。(RO)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_ST PMU_SLEEP_SWITCH_ACTIVE_END_INT 的屏蔽中断状态。(RO)

PMU_SLEEP_SWITCH_MODEM_END_INT_ST PMU_SLEEP_SWITCH_MODEM_END_INT 的屏蔽中断状态。(RO)

PMU_MODEM_SWITCH_SLEEP_END_INT_ST PMU_MODEM_SWITCH_SLEEP_END_INT 的屏蔽中断状态。(RO)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_ST PMU_ACTIVE_SWITCH_SLEEP_END_INT 的屏蔽中断状态。(RO)

PMU_MODEM_SWITCH_ACTIVE_START_INT_ST PMU_MODEM_SWITCH_ACTIVE_START_INT 的屏蔽中断状态。(RO)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_ST PMU_SLEEP_SWITCH_ACTIVE_START_INT 的屏蔽中断状态。(RO)

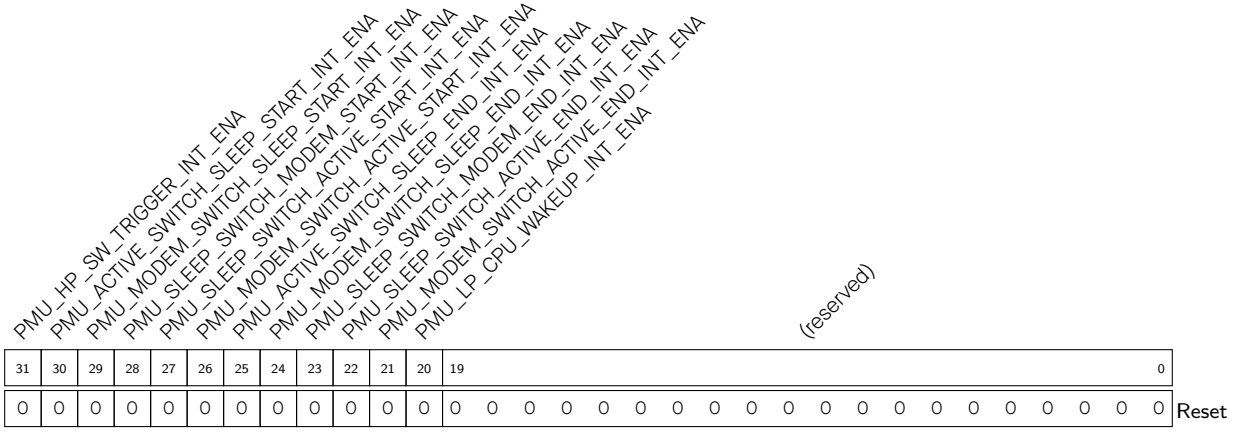
PMU_SLEEP_SWITCH_MODEM_START_INT_ST PMU_SLEEP_SWITCH_MODEM_START_INT 的屏蔽中断状态。(RO)

PMU_MODEM_SWITCH_SLEEP_START_INT_ST PMU_MODEM_SWITCH_SLEEP_START_INT 的屏蔽中断状态。(RO)

PMU_ACTIVE_SWITCH_SLEEP_START_INT_ST PMU_ACTIVE_SWITCH_SLEEP_START_INT 的屏蔽中断状态。(RO)

PMU_HP_SW_TRIGGER_INT_ST PMU_HP_SW_TRIGGER_INT 的屏蔽中断状态。(RO)

Register 13.52. PMU_LP_INT_ENA_REG (0x0178)



PMU_LP_CPU_WAKEUP_INT_ENA 写 1 使能 PMU_LP_CPU_WAKEUP_INT。(R/W)

PMU_MODEM_SWITCH_ACTIVE_END_INT_ENA 写 1 使能 PMU_MODEM_SWITCH_ACTIVE_END_INT。(R/W)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_ENA 写 1 使能 PMU_SLEEP_SWITCH_ACTIVE_END_INT。(R/W)

PMU_SLEEP_SWITCH_MODEM_END_INT_ENA 写 1 使能 PMU_SLEEP_SWITCH_MODEM_END_INT。(R/W)

PMU_MODEM_SWITCH_SLEEP_END_INT_ENA 写 1 使能 PMU_MODEM_SWITCH_SLEEP_END_INT。(R/W)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_ENA 写 1 使能 PMU_ACTIVE_SWITCH_SLEEP_END_INT。(R/W)

PMU_MODEM_SWITCH_ACTIVE_START_INT_ENA 写 1 使能 PMU_MODEM_SWITCH_ACTIVE_START_INT。(R/W)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_ENA 写 1 使能 PMU_SLEEP_SWITCH_ACTIVE_START_INT。(R/W)

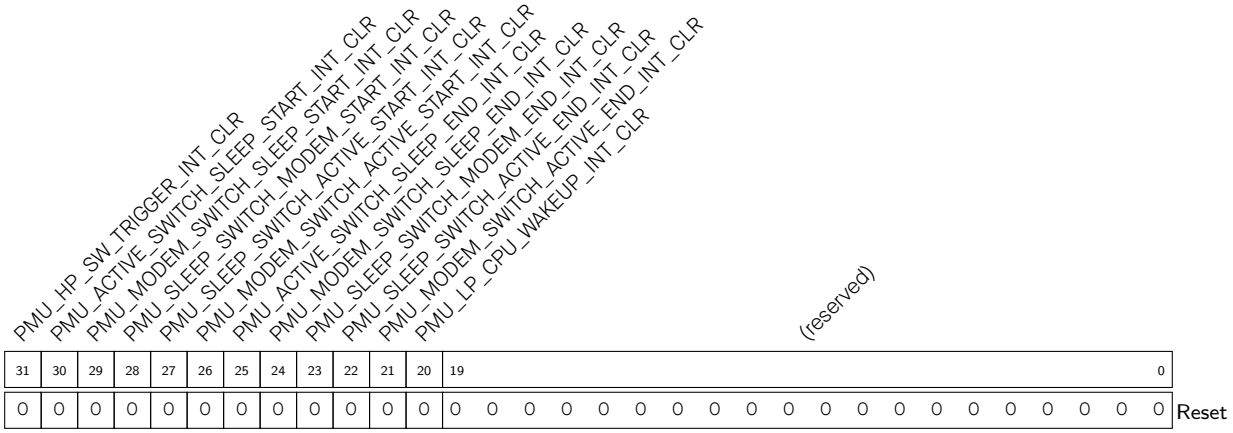
PMU_SLEEP_SWITCH_MODEM_START_INT_ENA 写 1 使能 PMU_SLEEP_SWITCH_MODEM_START_INT。(R/W)

PMU_MODEM_SWITCH_SLEEP_START_INT_ENA 写 1 使能 PMU_MODEM_SWITCH_SLEEP_START_INT。(R/W)

PMU_ACTIVE_SWITCH_SLEEP_START_INT_ENA 写 1 使能 PMU_ACTIVE_SWITCH_SLEEP_START_INT。(R/W)

PMU_HP_SW_TRIGGER_INT_ENA 写 1 使能 PMU_HP_SW_TRIGGER_INT。(R/W)

Register 13.53. PMU_LP_INT_CLR_REG (0x017C)



PMU_LP_CPU_WAKEUP_INT_CLR 写 1 清除 PMU_LP_CPU_WAKEUP_INT。(WT)

PMU_MODEM_SWITCH_ACTIVE_END_INT_CLR 写 1 清除 PMU_MODEM_SWITCH_ACTIVE_END_INT。(WT)

PMU_SLEEP_SWITCH_ACTIVE_END_INT_CLR 写 1 清除 PMU_SLEEP_SWITCH_ACTIVE_END_INT。(WT)

PMU_SLEEP_SWITCH_MODEM_END_INT_CLR 写 1 清除 PMU_SLEEP_SWITCH_MODEM_END_INT。(WT)

PMU_MODEM_SWITCH_SLEEP_END_INT_CLR 写 1 清除 PMU_MODEM_SWITCH_SLEEP_END_INT。(WT)

PMU_ACTIVE_SWITCH_SLEEP_END_INT_CLR 写 1 清除 PMU_ACTIVE_SWITCH_SLEEP_END_INT。(WT)

PMU_MODEM_SWITCH_ACTIVE_START_INT_CLR 写 1 清除 PMU_MODEM_SWITCH_ACTIVE_START_INT。(WT)

PMU_SLEEP_SWITCH_ACTIVE_START_INT_CLR 写 1 清除 PMU_SLEEP_SWITCH_ACTIVE_START_INT。(WT)

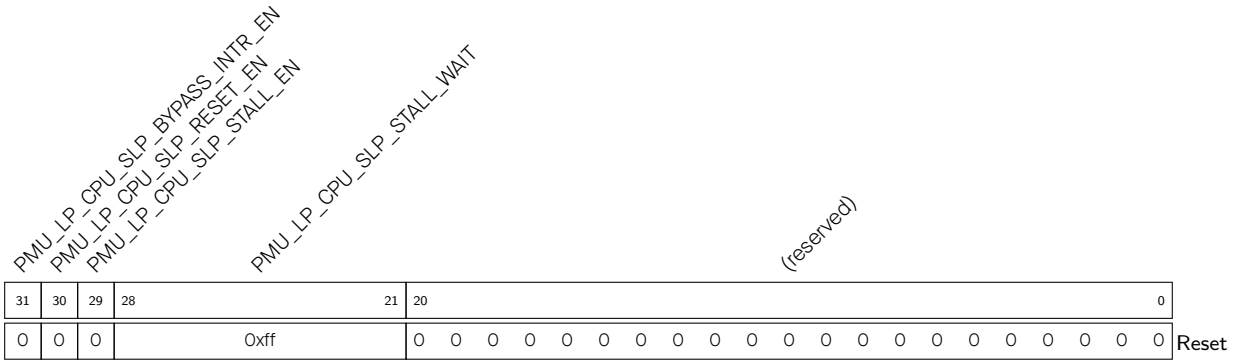
PMU_SLEEP_SWITCH_MODEM_START_INT_CLR 写 1 清除 PMU_SLEEP_SWITCH_MODEM_START_INT。(WT)

PMU_MODEM_SWITCH_SLEEP_START_INT_CLR 写 1 清除 PMU_MODEM_SWITCH_SLEEP_START_INT。(WT)

PMU_ACTIVE_SWITCH_SLEEP_START_INT_CLR 写 1 清除 PMU_ACTIVE_SWITCH_SLEEP_START_INT。(WT)

PMU_HP_SW_TRIGGER_INT_CLR 写 1 清除 PMU_HP_SW_TRIGGER_INT。(WT)

Register 13.54. PMU_LP_CPU_PWR0_REG (0x0180)



PMU_LP_CPU_SLP_STALL_WAIT 配置 LP CPU 在进入睡眠状态使能 stall 状态之后，等待 stall 生效的时间。单位是 LP_DYN_FAST_CLK。(R/W)

PMU_LP_CPU_SLP_STALL_EN 配置是否暂停进入睡眠模式的 LP CPU。

0：不暂停

1：暂停

(R/W)

PMU_LP_CPU_SLP_RESET_EN 配置是否复位进入睡眠模式的 LP CPU。

0：不复位

1：复位

(R/W)

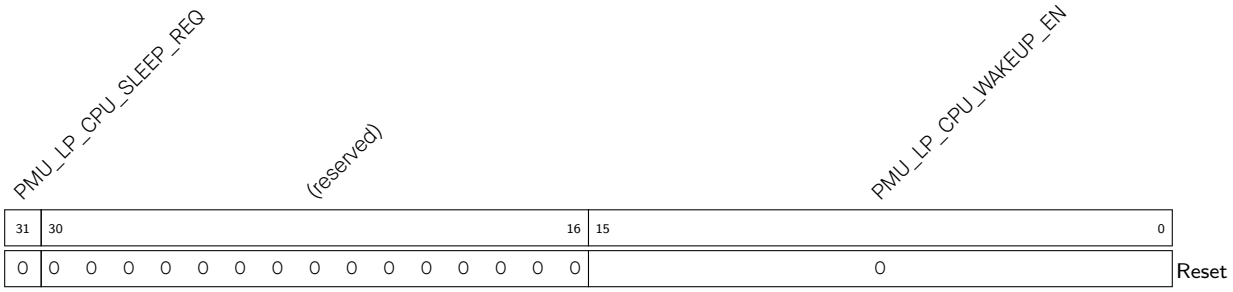
PMU_LP_CPU_SLP_BYPASS_INTR_EN 配置是否为进入睡眠模式的 LP CPU 打开中断信号。

0：关闭中断信号

1：打开中断信号

(R/W)

Register 13.55. PMU_LP_CPU_PWR1_REG (0x0184)



PMU_LP_CPU_WAKEUP_EN 配置 LP CPU 的唤醒源。具体请参考章节 4 低功耗处理器 > 表 4.9-1 唤醒源。(R/W)

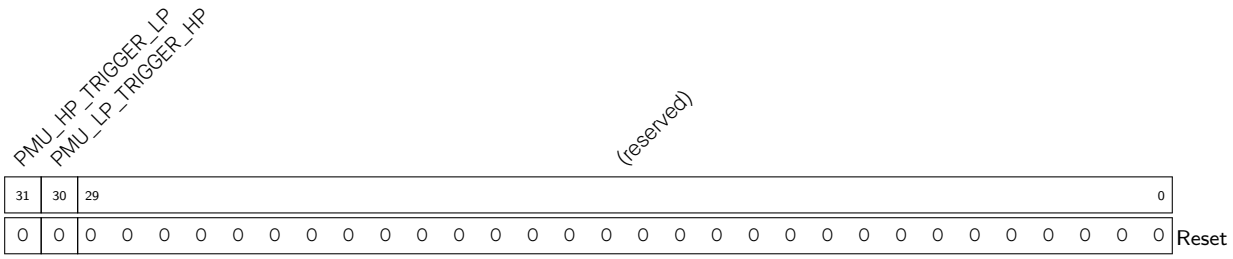
PMU_LP_CPU_SLEEP_REQ LP CPU 睡眠请求寄存器。

0：不睡眠

1：进入睡眠

(WT)

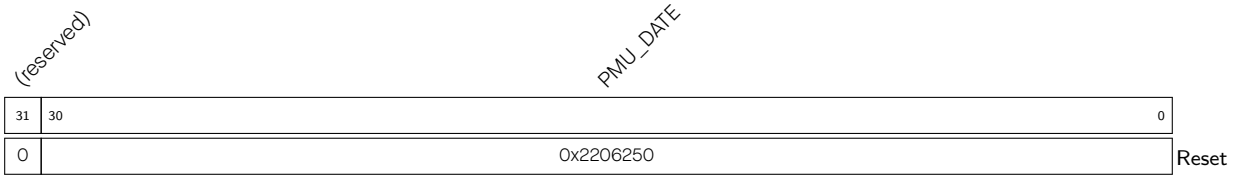
Register 13.56. PMU_HP_LP_CPU_COMM_REG (0x0188)



PMU_LP_TRIGGER_HP LP CPU 配置该寄存器为 1 时，芯片被唤醒。(WT)

PMU_HP_TRIGGER_LP HP CPU 配置该寄存器为 1 时，LP CPU 被唤醒。(WT)

Register 13.57. PMU_DATE_REG (0x01A8)



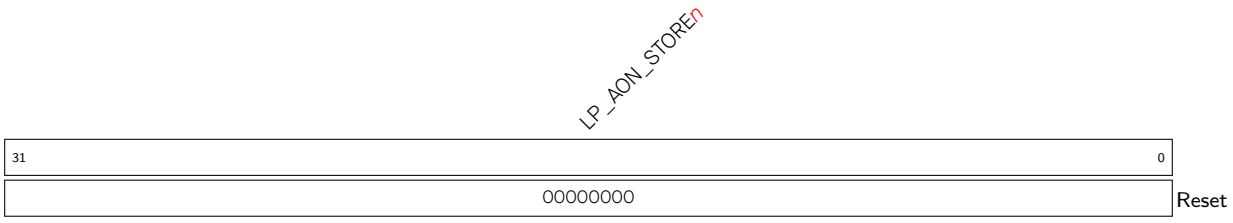
PMU_DATE 版本控制寄存器。(R/W)

13.10.2 Always-on 寄存器

本小节的所有地址均为相对于 Always-on 寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

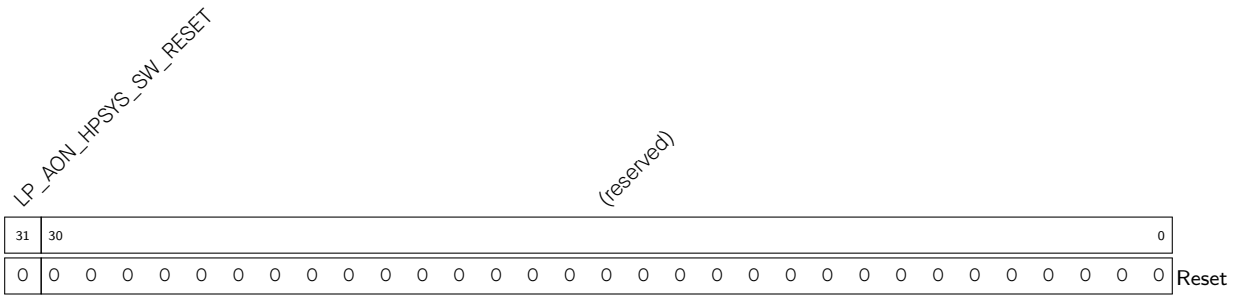
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 13.58. LP_AON_STORE*n*_REG (*n*: 0-9) (0x0000+0x4**n*)



LP_AON_STORE*n* Always-on 寄存器 *n*。
(R/W)

Register 13.59. LP_AON_SYS_CFG_REG (0x0034)



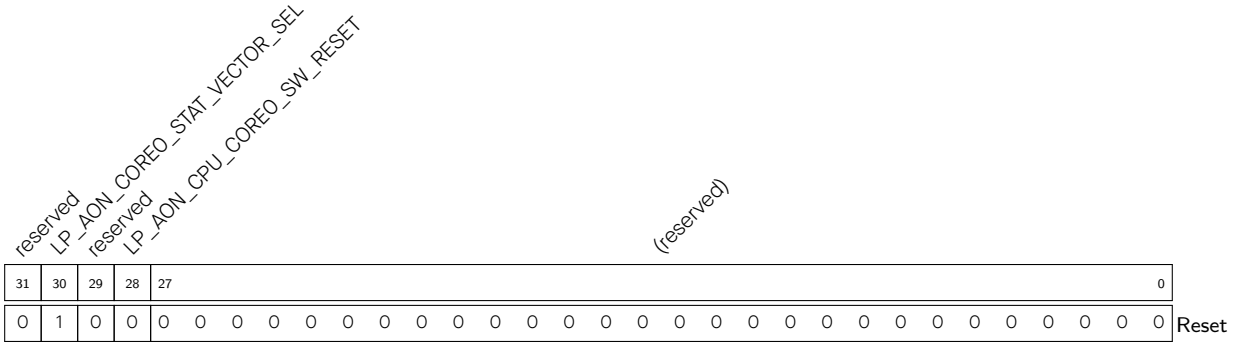
LP_AON_HPSYS_SW_RESET 配置系统复位。

0：无影响

1：使能复位

(WT)

Register 13.60. LP_AON_CPUCORE0_CFG_REG (0x0038)



LP_AON_CPU_CORE0_SW_RESET 配置 CPU 复位。

0：无影响

1：使能复位

(WT)

LP_AON_CORE0_STAT_VECTOR_SEL 配置是否使 CPU 从 LP SRAM 启动。

0：从 LP SRAM 启动

1：不从 LP SRAM 启动

(R/W)

第 14 章

系统定时器

14.1 概述

ESP32-C5 芯片内置 52 位系统定时器。该定时器可用于生成操作系统所需的滴答定时中断，也可以用作普通的定时器生成周期或单次延时中断。

14.2 主要特性

系统定时器具有如下特性：

- 集成两个 52 位计数器和三个 52 位比较器
- 支持软件访问由 APB_CLK 时钟驱动的寄存器
- 支持 CNT_CLK 时钟计数，两次计数周期的平均频率为 16 MHz
- 配置 XTAL_CLK (40 MHz/48 MHz) 作为 CNT_CLK 时钟源
- 支持 52 位报警值 (t) 和 26 位报警周期 (δt)
- 支持两种报警模式：
  - 单次报警模式：根据设定的目标报警值 (t)，生成一次性报警
  - 周期报警模式：根据设定的报警周期 (δt)，生成周期性报警
- 支持根据设置的报警值 (t) 或报警周期 (δt)，通过三个比较器生成三个独立中断
- 支持软件配置基准计数值。例如，支持从 Light-sleep 唤醒之后，系统定时器通过软件加载 RTC 定时器记录的睡眠时间，并进行补偿
- CPU 处于停止状态或处于在线调试状态时，系统定时器可选择停止运行或继续运行
- 支持事件任务矩阵 (ETM) 事件报警

14.3 系统定时器架构

系统定时器内置两个计数器（UNIT0 和 UNIT1）以及三个比较器（COMP0、COMP1 和 COMP2）。比较器用于监控计数器的计数值是否达到报警值。定时器的功能块图见图 14.3-1。

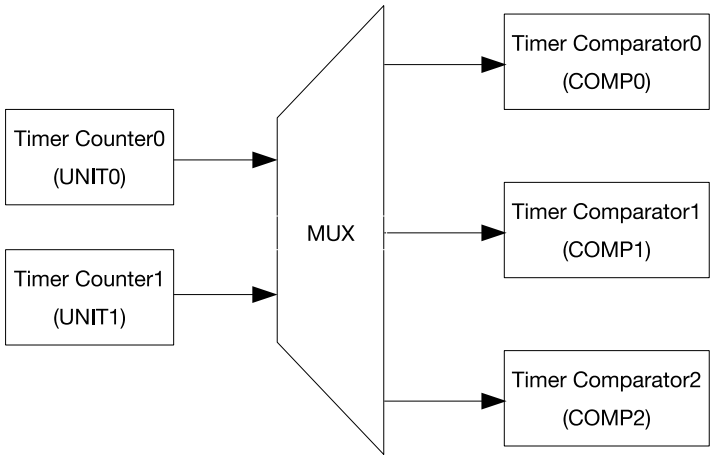


图 14.3-1. 系统定时器结构图

14.4 时钟源选择

计数器和比较器使用 XTAL_CLK 或者 RC_FAST_CLK 作为时钟源。可通过配置寄存器 PCR_SYSTIMER_FUNC_CLK_CONF_REG 的 PCR_SYSTIMER_FUNC_CLK_SEL 字段来选择时钟源。XTAL_CLK 经分频后，生成平均频率为 16 MHz 的计数器时钟信号 CNT_CLK，如图 14.5-1 所示。每个 CNT_CLK 时钟周期，计数递增 1/16 μs，即 16 个周期递增 1 μs。

APB_CLK 为配置寄存器等软件操作提供时钟信号。更多有关 APB_CLK 的信息，见章节 9 复位和时钟。

用户可使用系统寄存器的两个相关位来控制系统定时器，具体如下：

- 置位寄存器 PCR_SYSTIMER_CONF_REG 中 PCR_SYSTIMER_CLK_EN 位，使能系统定时器的 APB_CLK 信号。
- 置位寄存器 PCR_SYSTIMER_CONF_REG 中 PCR_SYSTIMER_RST_EN 位，复位系统定时器。

注意，复位后，系统定时器的寄存器将恢复到默认值。更多信息可参考章节 9 复位和时钟。

14.5 功能描述

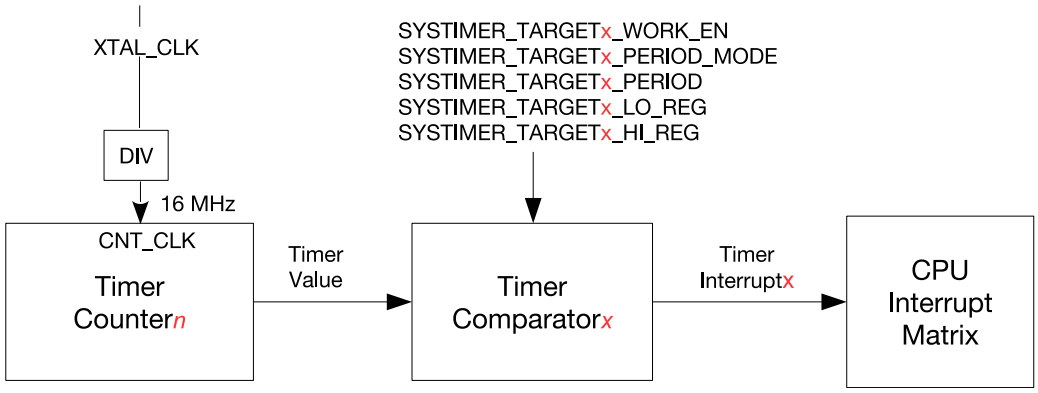


图 14.5-1. 系统定时器生成报警

图 14.5-1 展示了系统定时器生成报警的过程。过程中需要用到一个计数器和一个比较器，比较器将根据比较结果，生成报警中断。

14.5.1 计数器

系统定时器集成两个 52 位计数器，用 UNIT*n* 表示，*n* 可以取 0 或 1。计数器使用 16 MHz CNT_CLK 作为计数时钟。用户可通过配置寄存器 SYSTIMER_CONF_REG 中的下面三个位来控制计数器 UNIT*n*，具体如下：

- SYSTIMER_TIMER_UNIT*n*_WORK_EN：置位此位，使能计数器 UNIT*n*。
- SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN：置位此位，CPU0 停止运行后，计数器 UNIT*n* 也将停止工作。CPU 恢复运行后，计数器重新开始工作。
- SYSTIMER_TIMER_UNIT*n*_CORE1_STALL_EN：置位此位，CPU1 停止运行后，计数器 UNIT*n* 也将停止工作。CPU 恢复运行后，计数器重新开始工作。

假设 CPU 当前状态为停止工作，UNIT*n* 的具体配置见下表：

表 14.5-1. UNIT*n* 配置控制位

SYSTIMER_TIMER_ UNIT*n*_WORK_EN	SYSTIMER_TIMER_ UNIT*n*_CORE0_STALL_EN	SYSTIMER_TIMER_ UNIT*n*_CORE1_STALL_EN	计数器 UNIT*n*
0	x*	x*	未处于工作状态。
1	x	1	暂停计数，但在 CPU core1 恢复运行后继续计数。
1	1	x	暂停计数，但在 CPU core0 恢复运行后继续计数。
1	0	0	不受影响，照常计数。

* x：无关项

计数器 UNIT*n* 处于工作状态时，计数值按计数周期递增。UNIT*n* 停止工作时，计数值将保持不变，不再递增。

计数起始值的低 32 位和高 20 位分别从 SYSTIMER_TIMER_UNIT*n*_LOAD_LO 和 SYSTIMER_TIMER_

UNIT*n*_LOAD_HI 中装载。置位 SYSTIMER_TIMER_UNIT*n*_LOAD 将触发重装载事件，当前计数起始值立即更新。如果计数器 UNIT*n* 处于工作状态，则从新装载的计数值开始计数。

置位 SYSTIMER_TIMER_UNIT*n*_UPDATE 将触发更新事件。当前计数值的低 32 位和高 20 位被锁存至寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_LO 和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 后，SYSTIMER_TIMER_UNIT*n*_VALUE_VALID 会被置起。在下次更新事件发生前，SYSTIMER_TIMER_UNIT*n*_VALUE_LO 和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 寄存器中的值保持不变。

14.5.2 比较器和报警

系统定时器有三个 52 位比较器，用 COMP*x* 表示，其中 *x* 可以取 0、1、2。比较器可根据设置的不同报警值 (t) 或报警周期 (δt)，触发不同的中断。

用户可配置寄存器 SYSTIMER_TARGET*x*_PERIOD_MODE，选择比较器 COMP*x* 生成报警的模式：

- 1：周期报警模式
- 0：单次报警模式

选择周期报警模式时，寄存器 SYSTIMER_TARGET*x*_PERIOD 中的值为报警周期 (δt)。假设当前计数值为 t1，经过一段时间，当计数值达到 t1 + δt 时，将触发一次报警中断。再经过一段时间，当计数值达到 t1 + 2×δt 时，将再一次触发报警中断，以此类推。通过上述方式即可实现周期性报警。

选择单次报警模式时，SYSTIMER_TIMER_TARGET*x*_LO 和 SYSTIMER_TIMER_TARGET*x*_HI 分别提供报警值 (t) 的低 32 位和高 20 位。假设当前计数值为 t2 (t2 <= t)，经过一段时间，当计数到报警值 (t) 时，触发一次报警。与周期报警模式不同，单次报警模式仅生成一次报警中断。

用户可通过寄存器 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择用于与 COMP*x* 进行比较的计数器值，然后生成报警：

- 1：与计数器 UNIT*1* 的计数值进行比较
- 0：与计数器 UNIT*0* 的计数值进行比较

置位 SYSTIMER_TARGET*x*_WORK_EN，COMP*x* 开始进行比较：

- 在单次报警模式下，COMP*x* 将比较计数器中的实际计数值与寄存器中设置的报警值 (t)；
- 在周期报警模式下，COMP*x* 将比较计数器中的实际计数值与 t1 + n×δt (n = 1,2,3...)。

在单次报警模式下，当实际计数值等于报警值 (t)，或在周期报警模式下，实际计数值等于 t1 + n×δt (n = 1,2,3...) 时，将触发一次报警中断。但如果设定的报警值 (t) 小于当前计数值，即报警值 (t) 已成为过去，在当前计数值超过设定的报警值 (t) 一定范围 (0 ~ 2^{51} –1) 时，也将立即触发中断。无论是在单次报警模式还是在周期报警模式下，都可以从 SYSTIMER_TARGET*x*_LO_RO 和 SYSTIMER_TARGET*x*_HI_RO 中读取实际报警值的低 32 位和高 20 位。当前计数值 t_c、报警值 t_t 和触发报警的关系如下表所示：

表 14.5-2. 报警触发条件

t_c 与 t_t 的关系	触发条件
t_c- t_t <= 0	当 t_c = t_t 时，触发报警
0 <= t_c - t_t < 2^{51} - 1 (t_c < 2^{51} 且 t_t < 2^{51}； 或 t_c >= 2^{51} 且 t_t >= 2^{51})	立即触发报警

t_c - t_t >= 2^{51} - 1	t_c 达到最大值 52'hfffffffffffff 后溢出，然后从 0 开始计数，计数达到 t_t 时触发报警

14.5.3 事件任务矩阵

在 ESP32-C5 中，系统定时器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发系统定时器的 ETM 任务，或者通过系统定时器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与系统定时器相关的 ETM 任务和 ETM 事件。

系统定时器可产生的 ETM 事件有：

- SYSTIMER_EVT_CNT_CMP*x*：表示 COMP*x* 产生的报警事件。

当 SYSTIMER_ETM_EN 置 1 时，对应的报警事件会传到 ETM 模块。

14.5.4 同步操作

计数器与比较器的工作时钟和软件的配置时钟频率并不相同，因此在配置部分寄存器时，必须进行同步。完整的同步过程包括下面两个步骤：

1. 通过软件向配置寄存器字段写入合适的值，见表 14.5-3 第一列。
2. 通过软件置位相应的同步使能位，开始同步操作，见表 14.5-3 第二列。

表 14.5-3. 同步操作

需要同步的字段	同步使能位
SYSTIMER_TIMER_UNIT*n*_LOAD_LO SYSTIMER_TIMER_UNIT*n*_LOAD_HI	SYSTIMER_TIMER_UNIT*n*_LOAD
SYSTIMER_TARGET*x*_PERIOD SYSTIMER_TIMER_TARGET*x*_HI SYSTIMER_TIMER_TARGET*x*_LO	SYSTIMER_TIMER_COMP*x*_LOAD

计数器与比较器的工作时钟和软件的读取时钟频率并不相同，因此在读取部分寄存器时，也需要进行同步。完整的同步步骤如下：

1. 软件将 1 写入更新寄存器 SYSTIMER_TIMER_UNIT*n*_UPDATE 中。
2. 软件读取相应位 SYSTIMER_TIMER_UNIT*n*_VALUE_VALID 为有效，即表示同步完成。
3. 软件读取相应的状态寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 和 SYSTIMER_TIMER_UNIT*n*_VALUE_LO。

14.6 中断

ESP32-C5 的系统定时器可以生成以下中断信号并发送给 中断矩阵。

- SYSTIMER_TARGET0_INT
- SYSTIMER_TARGET1_INT

- SYSTIMER_TARGET2_INT

以上中断信号由系统定时器的内部中断源生成。表 14.6-1 列出了系统定时器的内部中断源、中断触发条件以及生成的中断信号。

表 14.6-1. 系统定时器的内部中断源

内部中断源	触发条件	中断信号
SYSTIMER_INT0	比较器 COMP0 报警时	SYSTIMER_TARGET0_INT
SYSTIMER_INT1	比较器 COMP1 报警时	SYSTIMER_TARGET1_INT
SYSTIMER_INT2	比较器 COMP2 报警时	SYSTIMER_TARGET2_INT

以上中断为电平类型中断。比较器开始报警时，中断信号拉高。中断信号将一直保持高电平，直至软件清除中断。用户可置位 SYSTIMER_TARGETx_INT_ENA 使能中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 14.8 寄存器列表。

14.7 编程示例

注意，在配置 COMPx 和 UNITn 过程中，需保证对应的 COMP 和 UNIT 处于工作状态。

14.7.1 读取当前计数器的值

1. 置位 SYSTIMER_TIMER_UNITn_UPDATE，将计数器 UNITn 的值更新至寄存器 SYSTIMER_TIMER_UNITn_VALUE_HI 和 SYSTIMER_TIMER_UNITn_VALUE_LO；
2. 轮询 SYSTIMER_TIMER_UNITn_VALUE_VALID，直至其值为 1。之后，用户可从寄存器 SYSTIMER_TIMER_UNITn_VALUE_HI 和 SYSTIMER_TIMER_UNITn_VALUE_LO 中读取计数器的值；
3. 读取寄存器 SYSTIMER_TIMER_UNITn_VALUE_LO（低 32 位）和 SYSTIMER_TIMER_UNITn_VALUE_HI（高 20 位）。

14.7.2 在单次报警模式下配置一次性报警

1. 设置 SYSTIMER_TARGETx_TIMER_UNIT_SEL 选择与 COMPx 进行比较的计数器（UNIT0 或 UNIT1）。
2. 读取当前计数器的值，步骤见章节 14.7.1。读取的当前值可用于计算步骤 4 中的报警值 (t)。
3. 清除 SYSTIMER_TARGETx_PERIOD_MODE，使能单次报警模式。
4. 设置报警值 (t)，并将报警值 (t) 的低 32 位和高 20 位分别写入 SYSTIMER_TIMER_TARGETx_LO 和 SYSTIMER_TIMER_TARGETx_HI。
5. 置位 SYSTIMER_TIMER_COMPx_LOAD，同步报警值 (t)，即将报警值 (t) 装载至比较器 COMPx。
6. 置位 SYSTIMER_TARGETx_WORK_EN 使能选择的比较器 COMPx；比较器 COMPx 开始比较计数值与报警值 (t)。

7. 置位 SYSTIMER_TARGET*x*_INT_ENA，使能中断。UNIT*n* 达到报警值 (t) 时，触发一次报警中断 SYSTIMER_
   TARGET*x*_INT。

14.7.3 在周期报警模式下配置周期性报警

1. 设置 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择与 COMP*x* 进行比较的计数器。
2. 将报警周期 (δt) 写入 SYSTIMER_TARGET*x*_PERIOD。
3. 置位 SYSTIMER_TIMER_COMP*x*_LOAD 同步报警周期值，即将 (δt) 装载至比较器 COMP*x*。
4. 先清除再置位 SYSTIMER_TARGET*x*_PERIOD_MODE 将 COMP*x* 配置为周期报警模式。
5. 置位 SYSTIMER_TARGET*x*_WORK_EN 使能选择的比较器 COMP*x*；比较器 COMP*x* 开始将计数值与计数初始值 + n×δt (n = 1, 2, 3...) 进行比较。
6. 置位 SYSTIMER_TARGET*x*_INT_ENA，使能中断。UNIT*n* 计数达到计数初始值 + n×δt (n = 1, 2, 3...) 时，触发一次 SYSTIMER_TARGET*x*_INT 中断。

14.7.4 唤醒后时间补偿

1. 在芯片进入 Light-sleep 模式之前，用户需配置 RTC 定时器，精确记录睡眠时间，见章节 13 低功耗管理。
2. 从 Light-sleep 模式唤醒后，读取 RTC 定时器记录的睡眠时间。
3. 读取当前系统定时器的计数值，见章节 14.7.1。
4. 将 RTC 记录的以 RTC_SLOW_CLK 周期为单位的睡眠时间，转换成以 CNT_CLK (16 MHz) 周期为单位的睡眠时间。例如，如果 RTC_SLOW_CLK 频率为 32 kHz，则 RTC 定时器记录的时间乘以 500。
5. 将 RTC 定时器转换后的值加到系统定时器的当前计数值：
   - 将计算所得值的低 32 位写入 SYSTIMER_TIMER_UNIT*n*_LOAD_LO，高 20 位写入 SYSTIMER_TIMER_UNIT*n*_LOAD_HI。
   - 置位 SYSTIMER_TIMER_UNIT*n*_LOAD，将新的定时器值装载到系统定时器。至此，系统定时器更新完成。

14.8 寄存器列表

本小节的所有地址均为相对于系统定时器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时钟控制寄存器			
SYSTIMER_CONF_REG	配置系统定时器时钟	0x0000	R/W
UNIT*0* 控制和配置寄存器			
SYSTIMER_UNIT0_OP_REG	读取 UNT0 的值到相应寄存器	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	待装载至 UNIT0 的值，高 20 位	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	待装载至 UNIT0 的值，低 32 位	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 的读值，高 20 位	0x0040	RO
SYSTIMER_UNIT0_VALUE_LO_REG	UNIT0 的读值，低 32 位	0x0044	RO
SYSTIMER_UNIT0_LOAD_REG	计数器 UNIT0 的装载同步寄存器	0x005C	WT
UNIT*1* 控制和配置寄存器			
SYSTIMER_UNIT1_OP_REG	读取计数器 UNIT1 的值	0x0008	varies
SYSTIMER_UNIT1_LOAD_HI_REG	待装载至计数器 UNIT1 的值，高 20 位	0x0014	R/W
SYSTIMER_UNIT1_LOAD_LO_REG	待装载至计数器 UNIT1 的值，低 32 位	0x0018	R/W
SYSTIMER_UNIT1_VALUE_HI_REG	计数器 UNIT1 的读值，高 20 位	0x0048	RO
SYSTIMER_UNIT1_VALUE_LO_REG	计数器 UNIT1 的读值，低 32 位	0x004C	RO
SYSTIMER_UNIT1_LOAD_REG	计数器 UNIT1 的装载同步寄存器	0x0060	WT
比较器 COMP*0* 的控制和配置寄存器			
SYSTIMER_TARGET0_HI_REG	待装载至比较器 COMP0 的报警值，高 20 位	0x001C	R/W
SYSTIMER_TARGET0_LO_REG	待装载至比较器 COMP0 的报警值，低 32 位	0x0020	R/W
SYSTIMER_TARGET0_CONF_REG	配置比较器 COMP0 的报警模式	0x0034	R/W
SYSTIMER_COMP0_LOAD_REG	比较器 COMP0 的装载同步寄存器	0x0050	WT
比较器 COMP*1* 的控制和配置寄存器			
SYSTIMER_TARGET1_HI_REG	待装载至比较器 COMP1 的报警值，高 20 位	0x0024	R/W
SYSTIMER_TARGET1_LO_REG	待装载至比较器 COMP1 的报警值，低 32 位	0x0028	R/W
SYSTIMER_TARGET1_CONF_REG	配置比较器 COMP1 的报警模式	0x0038	R/W
SYSTIMER_COMP1_LOAD_REG	比较器 COMP1 的装载同步寄存器	0x0054	WT
比较器 COMP*2* 的控制和配置寄存器			
SYSTIMER_TARGET2_HI_REG	待装载至比较器 COMP2 的报警值，高 20 位	0x002C	R/W
SYSTIMER_TARGET2_LO_REG	待装载至比较器 COMP2 的报警值，低 32 位	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	配置比较器 COMP2 的报警模式	0x003C	R/W
SYSTIMER_COMP2_LOAD_REG	比较器 COMP2 的装载同步寄存器	0x0058	WT
中断寄存器			
SYSTIMER_INT_ENA_REG	系统定时器的中断使能寄存器	0x0064	R/W
SYSTIMER_INT_RAW_REG	系统定时器的原始中断寄存器	0x0068	R/WTC/SS
SYSTIMER_INT_CLR_REG	系统定时器的中断清除寄存器	0x006C	WT

名称	描述	地址	访问
SYSTIMER_INT_ST_REG	系统定时器的中断状态寄存器	0x0070	RO
COMP*0* 状态寄存器			
SYSTIMER_REAL_TARGET0_LO_REG	COMP0 的实际报警值，低 32 位	0x0074	RO
SYSTIMER_REAL_TARGET0_HI_REG	COMP0 的实际报警值，高 20 位	0x0078	RO
COMP*1* 状态寄存器			
SYSTIMER_REAL_TARGET1_LO_REG	COMP1 的实际报警值，低 32 位	0x007C	RO
SYSTIMER_REAL_TARGET1_HI_REG	COMP1 的实际报警值，高 20 位	0x0080	RO
COMP*2* 状态寄存器			
SYSTIMER_REAL_TARGET2_LO_REG	COMP2 的实际报警值，低 32 位	0x0084	RO
SYSTIMER_REAL_TARGET2_HI_REG	COMP2 的实际报警值，高 20 位	0x0088	RO
版本寄存器			
SYSTIMER_DATE_REG	版本控制寄存器	0x00FC	R/W

14.9 寄存器

本小节的所有地址均为相对于系统定时器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 14.1. SYSTIMER_CONF_REG (0x0000)

Bits	31	30	29	28	27	26	25	24	23	22	21 ... 2	1	0
Field	SYSTIMER_CLK_EN	SYSTIMER_TIMER_UNIT0_WORK_EN	SYSTIMER_TIMER_UNIT1_WORK_EN	SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN	SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN	SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN	SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN	SYSTIMER_TARGET0_WORK_EN	SYSTIMER_TARGET1_WORK_EN	SYSTIMER_TARGET2_WORK_EN	(reserved)	SYSTIMER_ETM_EN	(reserved)
Reset	0	1	0	0	0	1	1	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0

SYSTIMER_ETM_EN 配置是否使能生成 ETM 事件。
- 0：无效
- 1：使能生成 ETM 事件

(R/W)

SYSTIMER_TARGET2_WORK_EN 配置是否使能 COMP2。
- 0：不使能
- 1：使能 COMP2

(R/W)

SYSTIMER_TARGET1_WORK_EN 配置是否使能 COMP1。具体配置值请见 SYSTIMER_TARGET2_WORK_EN。(R/W)

SYSTIMER_TARGET0_WORK_EN 配置是否使能 COMP0。具体配置值请见 SYSTIMER_TARGET2_WORK_EN。(R/W)

SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT1 是否也将停止工作。
- 0：无效
- 1：计数器 UNIT1 停止工作

(R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT1 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

见下页...

Register 14.1. SYSTIMER_CONF_REG (0x0000)

... 接上页

SYSTIMER_TIMER_UNIT1_WORK_EN 配置是否使能计数器 UNIT1。

0：无效

1：使能计数器 UNIT1

(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN 配置是否使能计数器 UNIT0。

0：无效

1：使能计数器 UNIT0

(R/W)

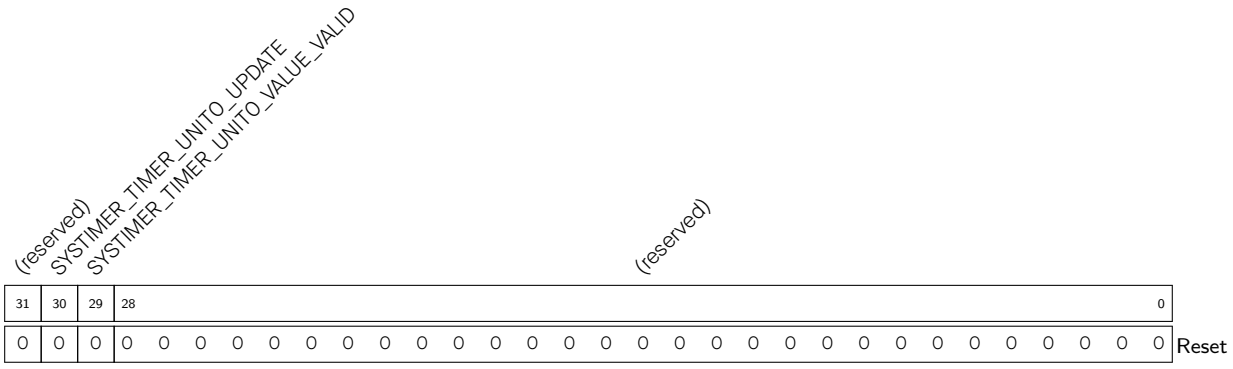
SYSTIMER_CLK_EN 配置寄存器时钟门控。

1：持续开启读写寄存器的时钟。

0：只在读写寄存器时打开所需时钟。

(R/W)

Register 14.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID 表示计数器 UNIT0 的值是否已同步读取至寄存器，且读值有效。

0：UNIT0 的值未同步读取至寄存器，读值无效

1：UNIT0 的值已同步读取至寄存器，读值有效

(R/SS/WTC)

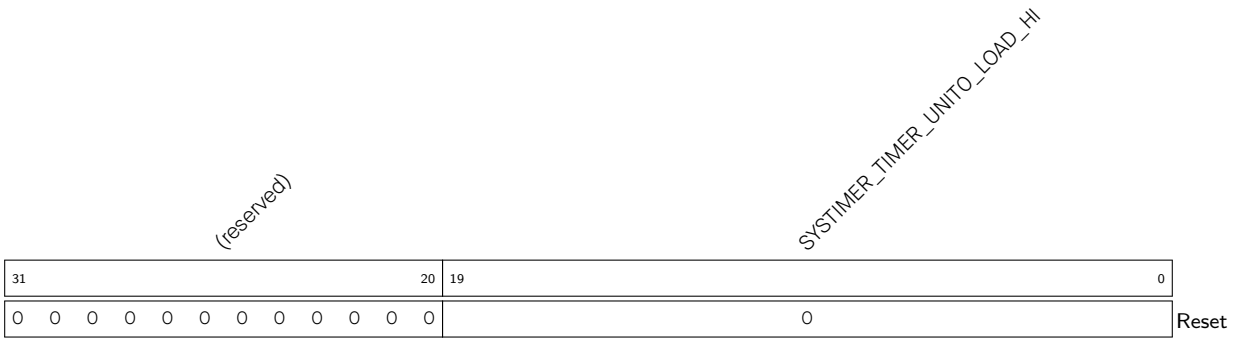
SYSTIMER_TIMER_UNIT0_UPDATE 配置是否更新计数器 UNIT0，即读取计数器 UNIT0 的值到寄存器 SYSTIMER_TIMER_UNIT0_VALUE_HI 和 SYSTIMER_TIMER_UNIT0_VALUE_LO。

0：无效

1：更新计数器 UNIT0

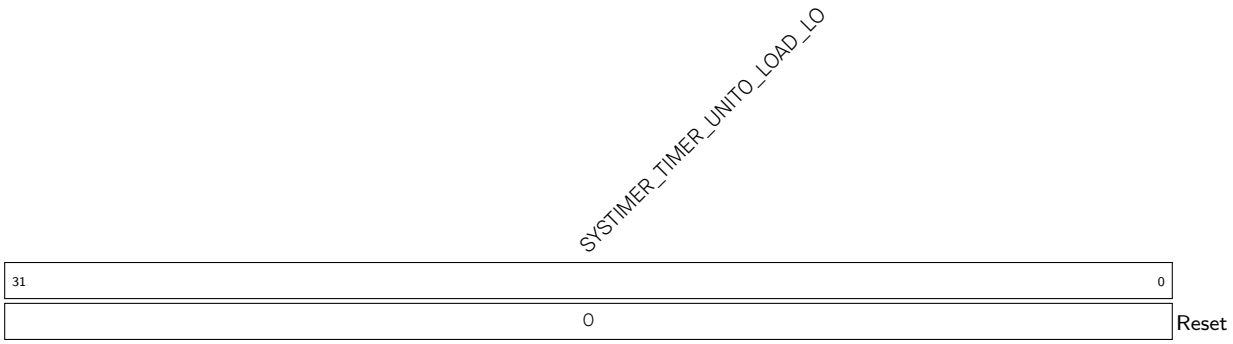
(WT)

Register 14.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



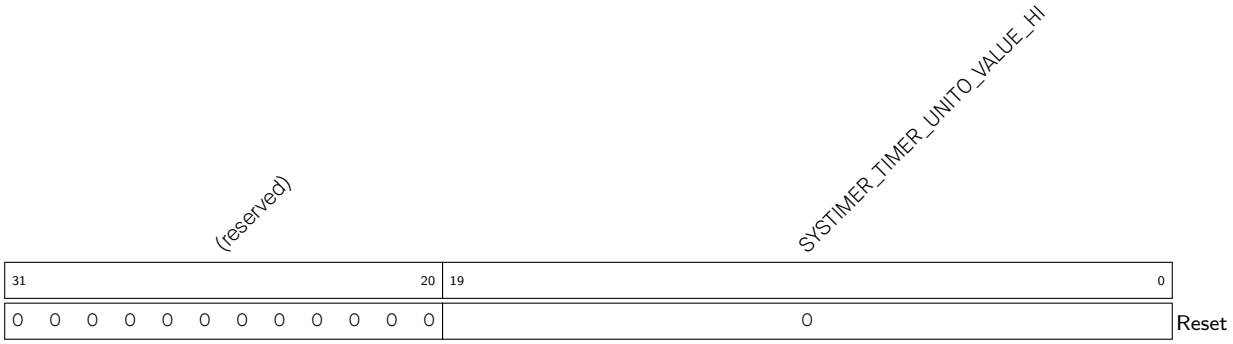
SYSTIMER_TIMER_UNIT0_LOAD_HI 配置待装载至计数器 UNIT0 的值，高 20 位。(R/W)

Register 14.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



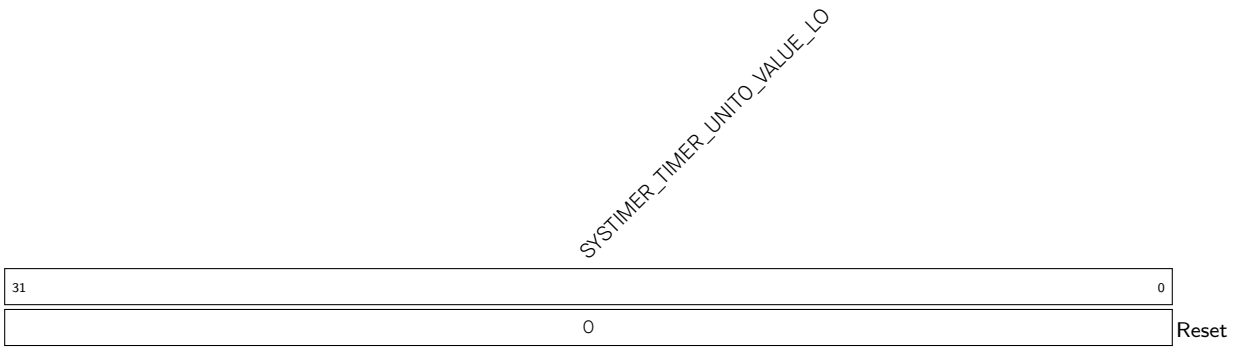
SYSTIMER_TIMER_UNIT0_LOAD_LO 配置待装载至计数器 UNIT0 的值，低 32 位。(R/W)

Register 14.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



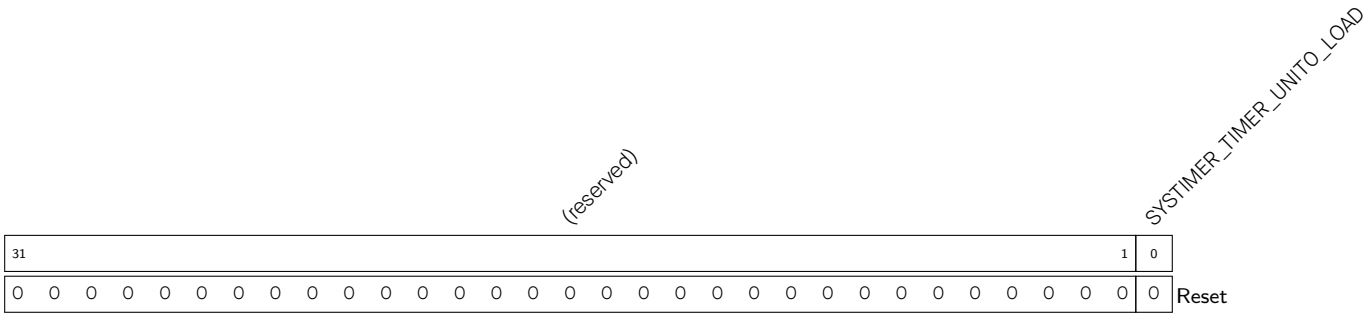
SYSTIMER_TIMER_UNIT0_VALUE_HI 表示计数器 UNIT0 的读数，高 20 位。(RO)

Register 14.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



SYSTIMER_TIMER_UNIT0_VALUE_LO 表示计数器 UNIT0 的读数，低 32 位。(RO)

Register 14.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



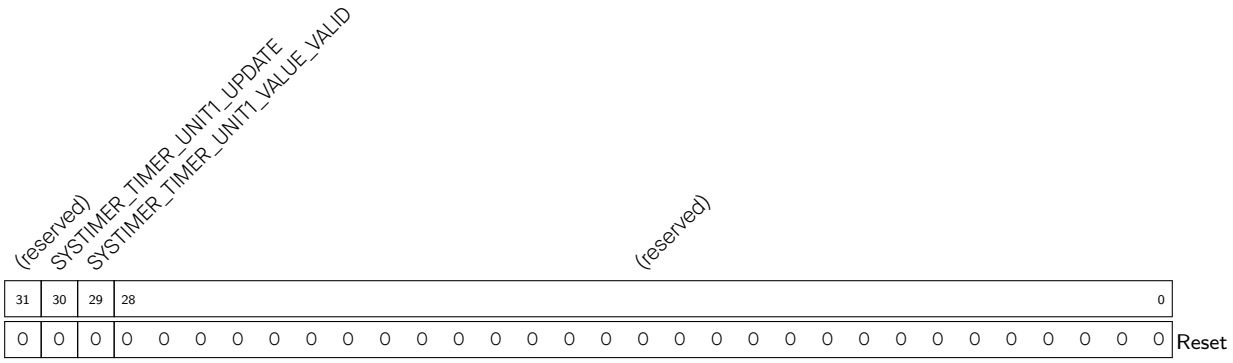
SYSTIMER_TIMER_UNIT0_LOAD 配置是否重新装载计数器 UNIT0，即是否重新装载寄存器 SYSTIMER_TIMER_UNIT0_LOAD_HI 和 SYSTIMER_TIMER_UNIT0_LOAD_LO 的值到计数器 UNIT0。

0：无效

1：重新装载计数器 UNIT0

(WT)

Register 14.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID 表示计数器 UNIT1 的值已同步读取至寄存器，且读值有效。(R/SS/WTC)

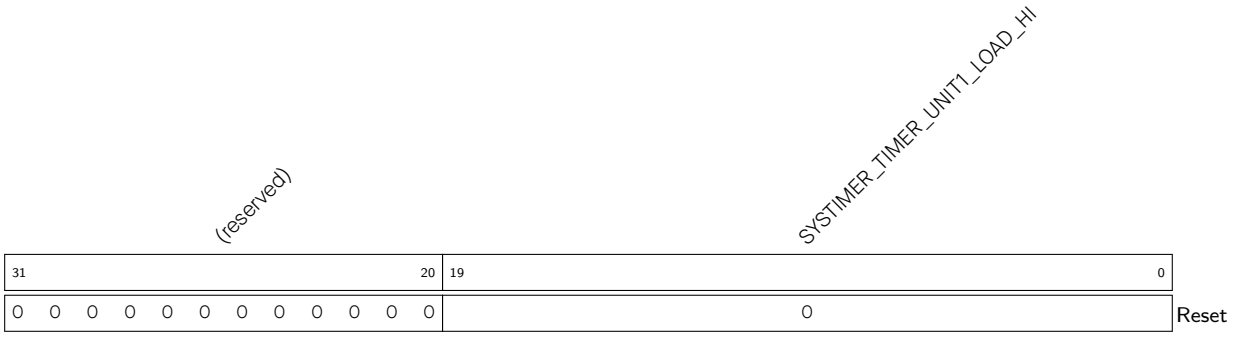
SYSTIMER_TIMER_UNIT1_UPDATE 配置是否更新计数器 UNIT1，即将计数器 UNIT1 的值读取到寄存器 SYSTIMER_TIMER_UNIT1_VALUE_HI 和 SYSTIMER_TIMER_UNIT1_VALUE_LO。

0：无效

1：更新计数器 UNIT1

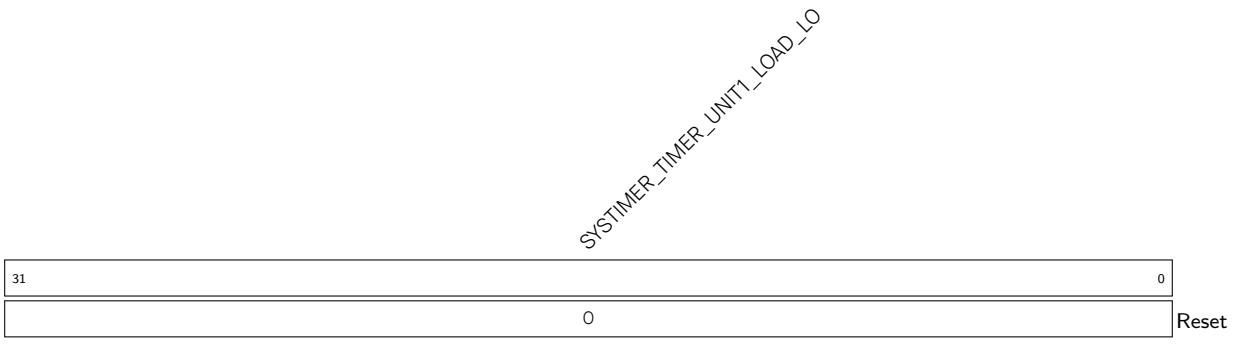
(WT)

Register 14.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



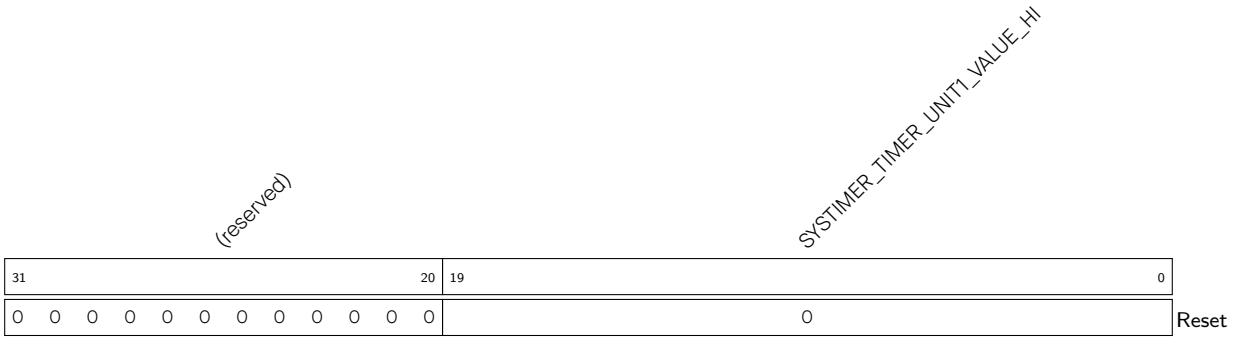
SYSTIMER_TIMER_UNIT1_LOAD_HI 配置待装载至计数器 UNIT1 的值，高 20 位。(R/W)

Register 14.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



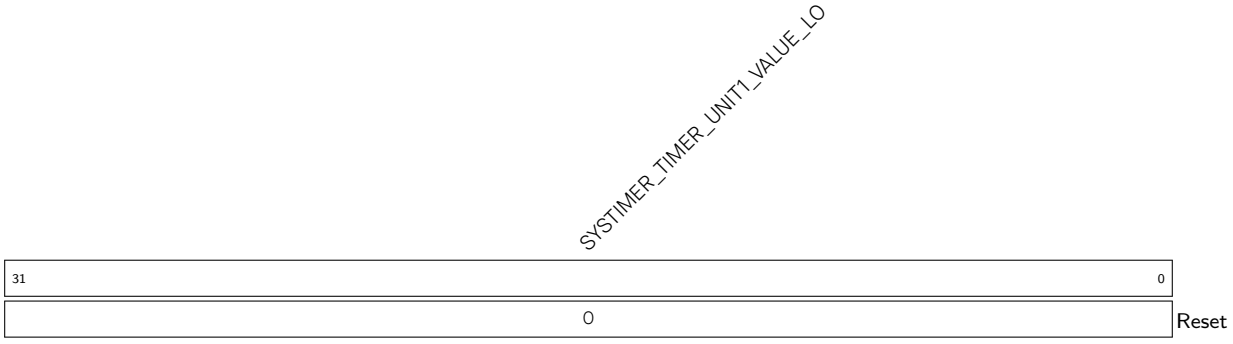
SYSTIMER_TIMER_UNIT1_LOAD_LO 配置待装载至计数器 UNIT1 的值，低 32 位。(R/W)

Register 14.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



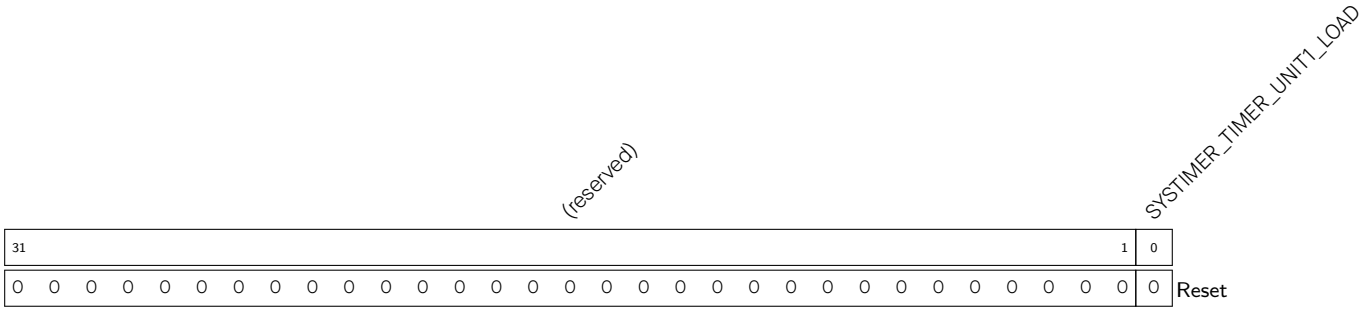
SYSTIMER_TIMER_UNIT1_VALUE_HI 表示计数器 UNIT1 的读数，高 20 位。(RO)

Register 14.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



SYSTIMER_TIMER_UNIT1_VALUE_LO 表示计数器 UNIT1 的读数，低 32 位。(RO)

Register 14.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



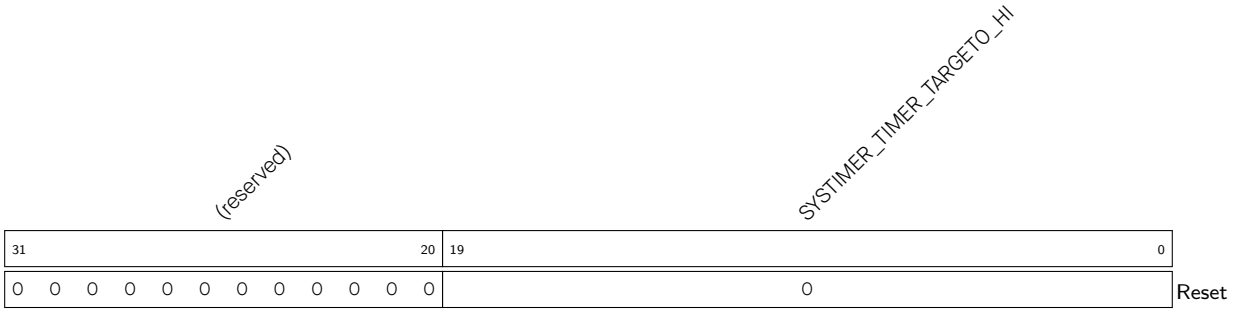
SYSTIMER_TIMER_UNIT1_LOAD 配置是否重新装载计数器 UNIT1，即是否重新装载寄存器 SYSTIMER_TIMER_UNIT1_LOAD_HI 和 SYSTIMER_TIMER_UNIT1_LOAD_LO 的值到计数器 UNIT1。

0：无效

1：重新装载计数器 UNIT1

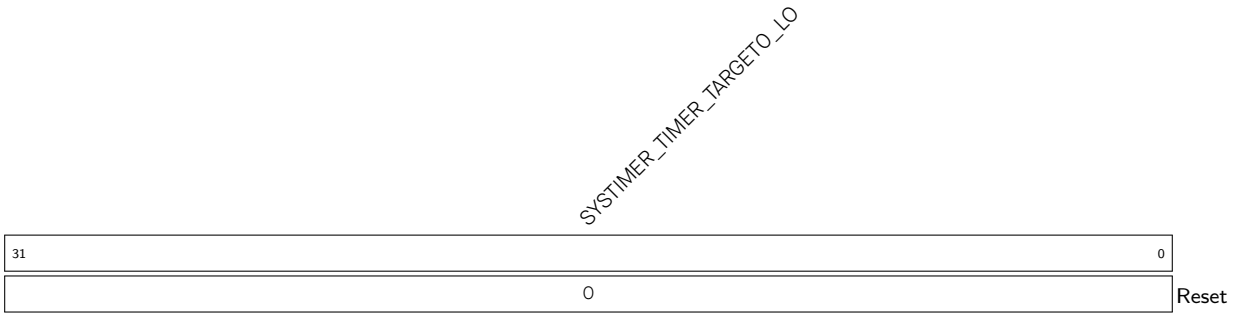
(WT)

Register 14.14. SYSTIMER_TARGET0_HI_REG (0x001C)



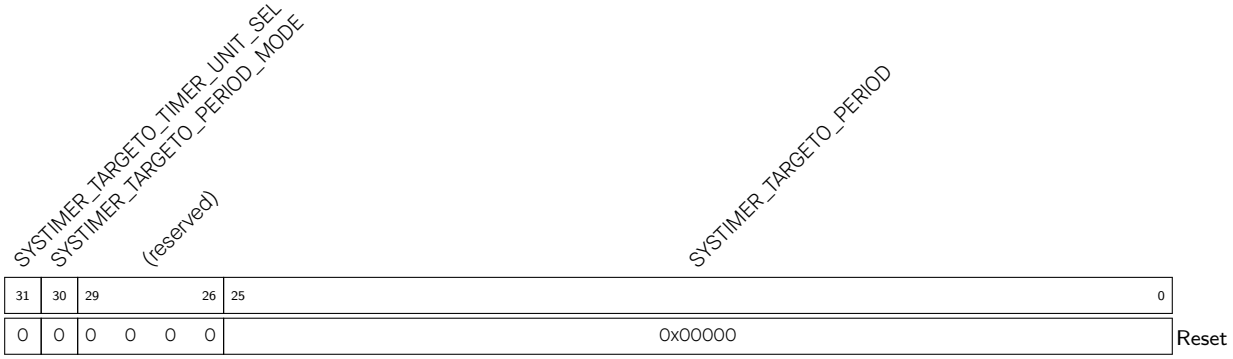
SYSTIMER_TIMER_TARGET0_HI 配置待装载至 COMP0 的报警值，高 20 位。(R/W)

Register 14.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO 配置待装载至 COMP0 的报警值，低 32 位。(R/W)

Register 14.16. SYSTIMER_TARGET0_CONF_REG (0x0034)



SYSTIMER_TARGET0_PERIOD 配置待装载至 COMP0 的报警周期。(R/W)

SYSTIMER_TARGET0_PERIOD_MODE 选择比较器 COMP0 生成报警的模式。

0：单次报警模式

1：周期报警模式

(R/W)

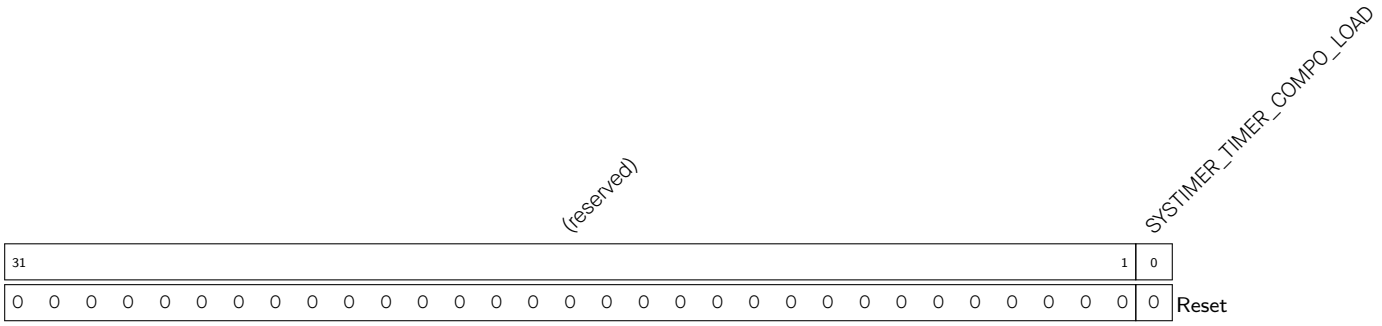
SYSTIMER_TARGET0_TIMER_UNIT_SEL 选择要与 COMP0 比较的计数器。

0：与计数器 UNIT*0* 的计数值进行比较

1：与计数器 UNIT*1* 的计数值进行比较

(R/W)

Register 14.17. SYSTIMER_COMP0_LOAD_REG (0x0050)



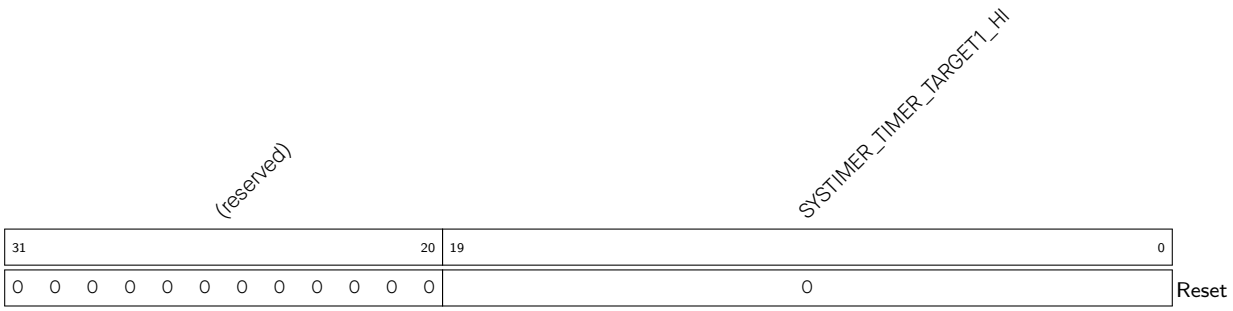
SYSTIMER_TIMER_COMP0_LOAD 配置是否使能比较器 COMP0 同步，即是否重新装载报警值或报警周期到 COMP0。

0：无效

1：使能比较器 COMP0 同步

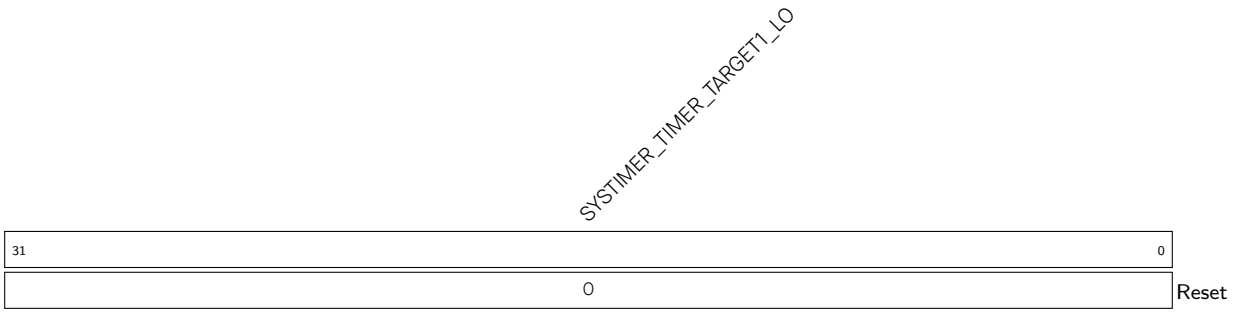
(WT)

Register 14.18. SYSTIMER_TARGET1_HI_REG (0x0024)



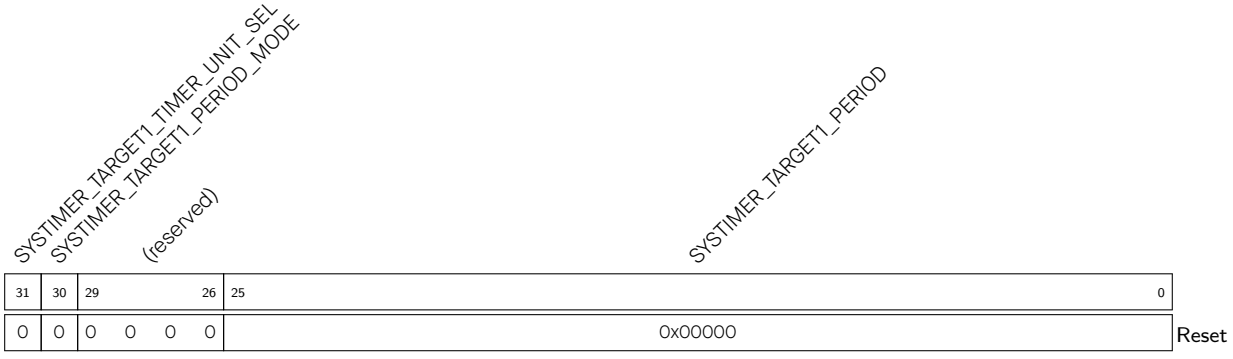
SYSTIMER_TIMER_TARGET1_HI 配置待装载至 COMP1 的报警值，高 20 位。(R/W)

Register 14.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO 配置待装载至 COMP1 的报警值，低 32 位。(R/W)

Register 14.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

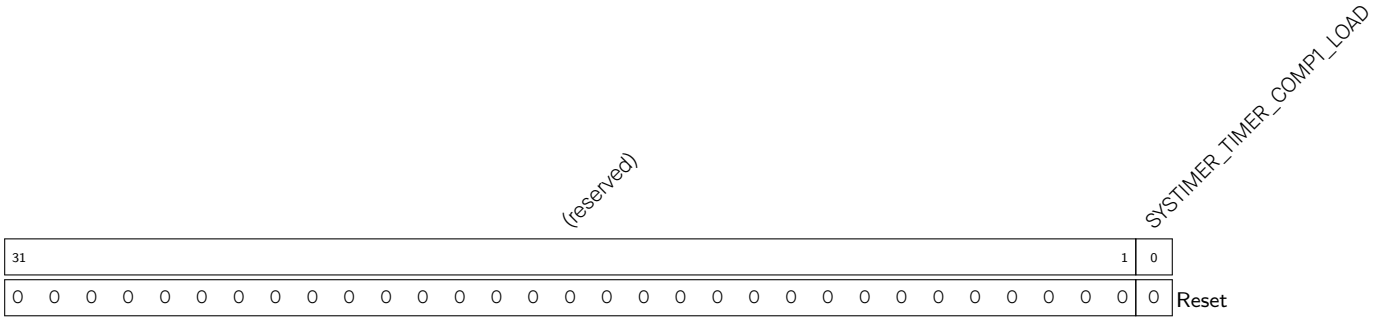


SYSTIMER_TARGET1_PERIOD 配置待装载至 COMP1 的报警周期。(R/W)

SYSTIMER_TARGET1_PERIOD_MODE 选择比较器 COMP1 生成报警的模式。具体配置值请见 SYSTIMER_TARGET0_PERIOD_MODE。(R/W)

SYSTIMER_TARGET1_TIMER_UNIT_SEL 选择要与 COMP1 比较的计数器。具体配置值请见 SYSTIMER_TARGET0_TIMER_UNIT_SEL。(R/W)

Register 14.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



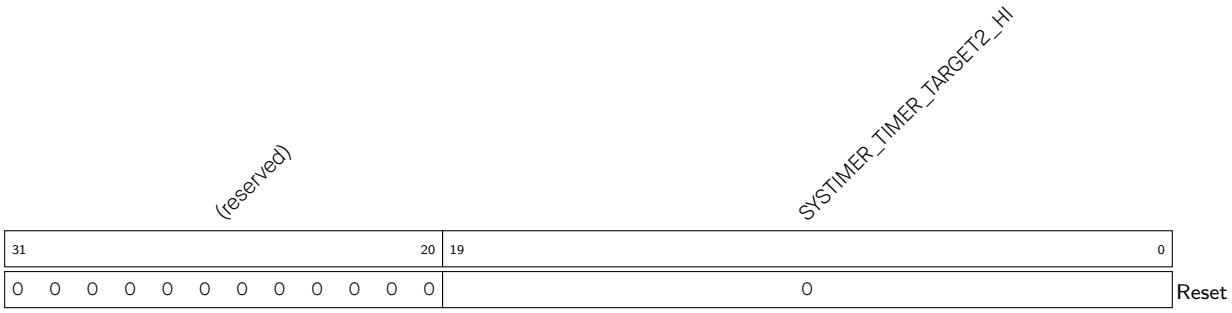
SYSTIMER_TIMER_COMP1_LOAD 配置是否使能比较器 COMP1 同步，即是否重新装载报警值或报警周期到 COMP1。

0：无效

1：使能比较器 COMP1 同步

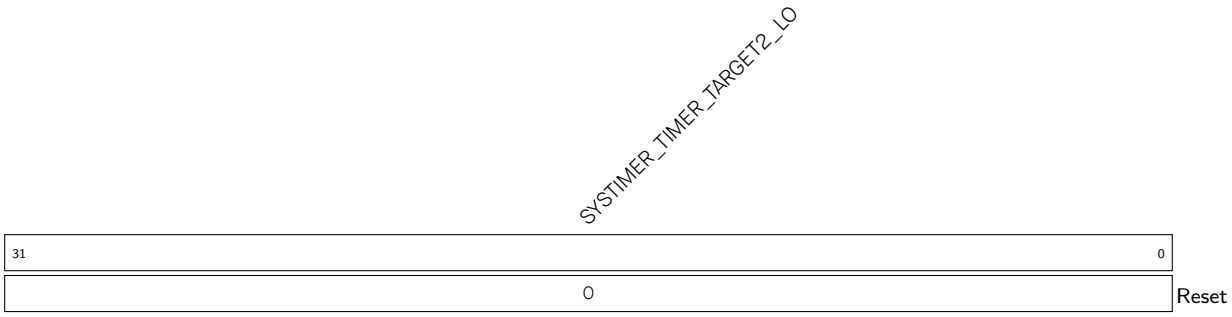
(WT)

Register 14.22. SYSTIMER_TARGET2_HI_REG (0x002C)



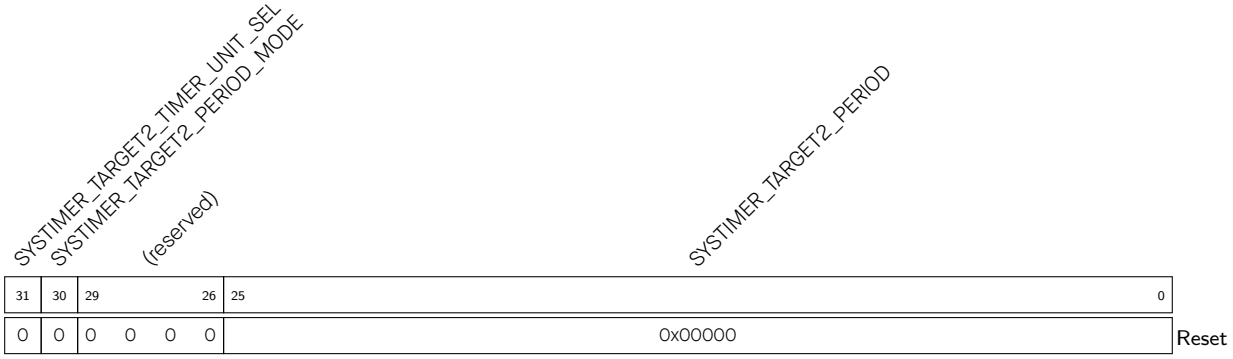
SYSTIMER_TIMER_TARGET2_HI 配置待装载至比较器 COMP2 的报警值，高 20 位。(R/W)

Register 14.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO 配置待装载至比较器 COMP2 的报警值，低 32 位。(R/W)

Register 14.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

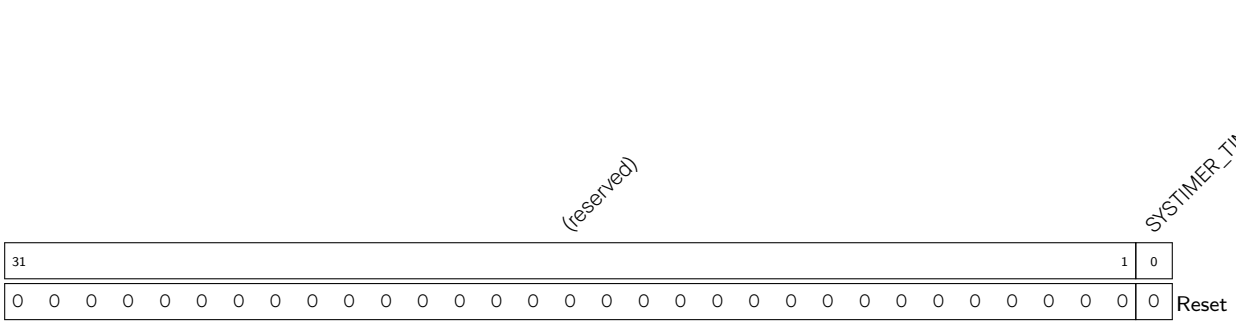


SYSTIMER_TARGET2_PERIOD 配置待装载至 COMP2 的报警周期。(R/W)

SYSTIMER_TARGET2_PERIOD_MODE 选择比较器 COMP2 生成报警的模式。具体配置值请见 SYSTIMER_TARGETO_PERIOD_MODE。(R/W)

SYSTIMER_TARGET2_TIMER_UNIT_SEL 选择要与 COMP2 比较的计数器。具体配置值请见 SYSTIMER_TARGETO_TIMER_UNIT_SEL。(R/W)

Register 14.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



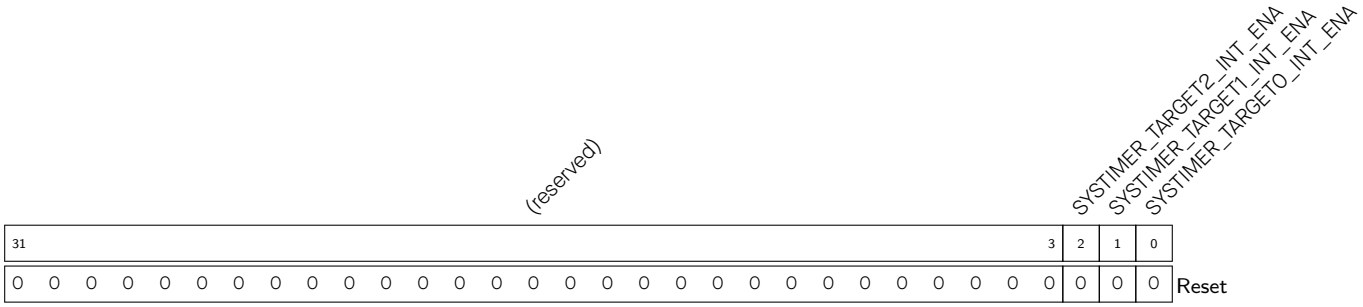
SYSTIMER_TIMER_COMP2_LOAD 配置是否使能比较器 COMP2 同步，即是否重新装载报警值或报警周期到 COMP2。

0：无效

1：使能比较器 COMP2 同步

(WT)

Register 14.26. SYSTIMER_INT_ENA_REG (0x0064)

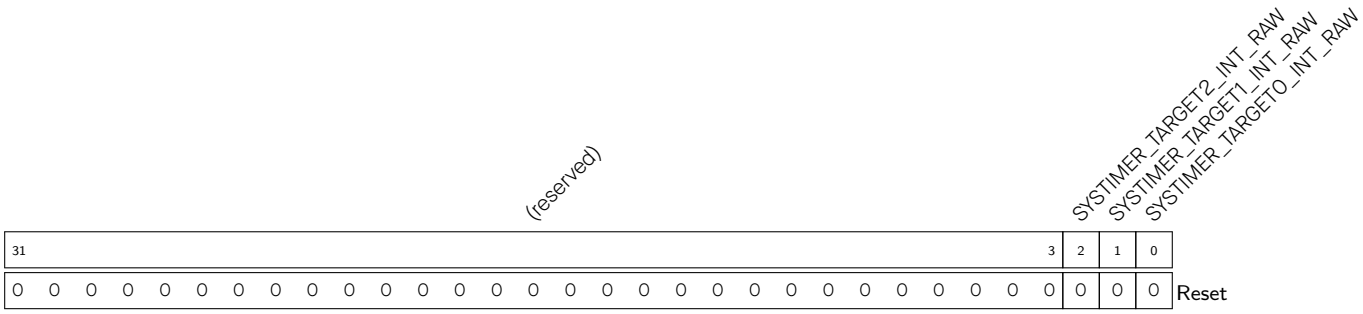


SYSTIMER_TARGET0_INT_ENA 写 1 使能 SYSTIMER_TARGET0_INT。(R/W)

SYSTIMER_TARGET1_INT_ENA 写 1 使能 SYSTIMER_TARGET1_INT。(R/W)

SYSTIMER_TARGET2_INT_ENA 写 1 使能 SYSTIMER_TARGET2_INT。(R/W)

Register 14.27. SYSTIMER_INT_RAW_REG (0x0068)

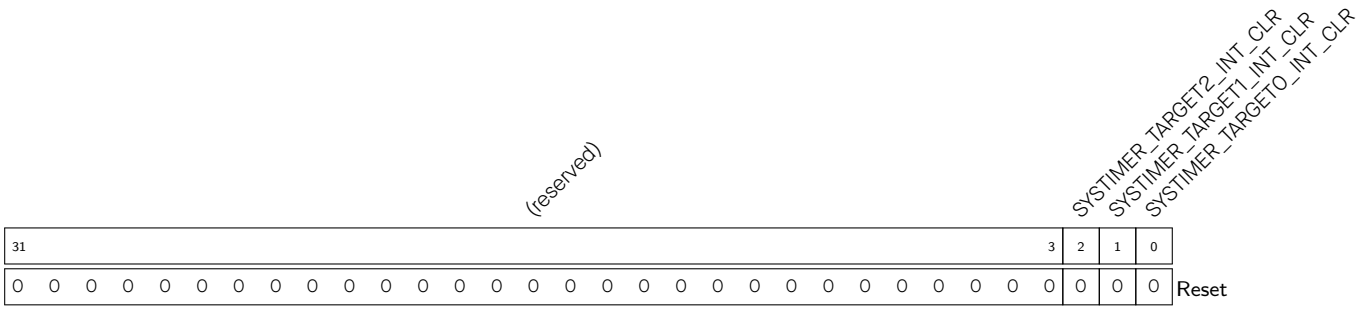


SYSTIMER_TARGET0_INT_RAW SYSTIMER_TARGET0_INT 的原始中断状态。(R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW SYSTIMER_TARGET1_INT 的原始中断状态。(R/WTC/SS)

SYSTIMER_TARGET2_INT_RAW SYSTIMER_TARGET2_INT 的原始中断状态。(R/WTC/SS)

Register 14.28. SYSTIMER_INT_CLR_REG (0x006C)

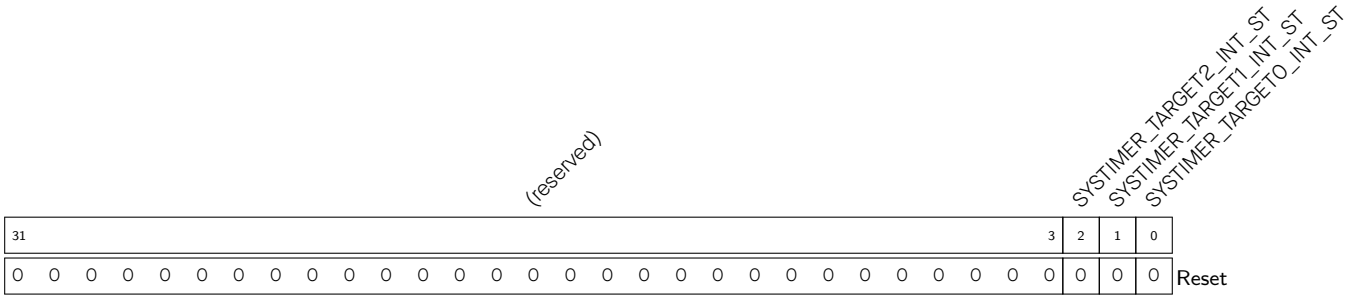


SYSTIMER_TARGET0_INT_CLR 写 1 清除 SYSTIMER_TARGET0_INT。(WT)

SYSTIMER_TARGET1_INT_CLR 写 1 清除 SYSTIMER_TARGET1_INT。(WT)

SYSTIMER_TARGET2_INT_CLR 写 1 清除 SYSTIMER_TARGET2_INT。(WT)

Register 14.29. SYSTIMER_INT_ST_REG (0x0070)

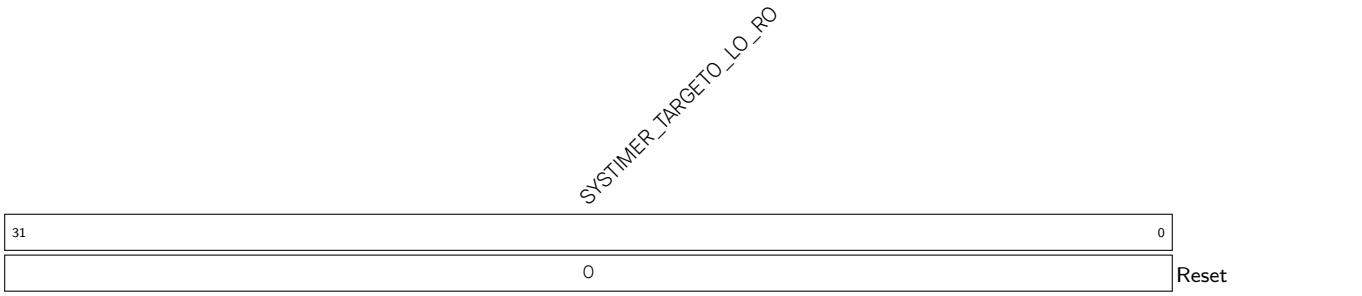


SYSTIMER_TARGET0_INT_ST SYSTIMER_TARGET0_INT 的屏蔽中断状态。(RO)

SYSTIMER_TARGET1_INT_ST SYSTIMER_TARGET1_INT 的屏蔽中断状态。(RO)

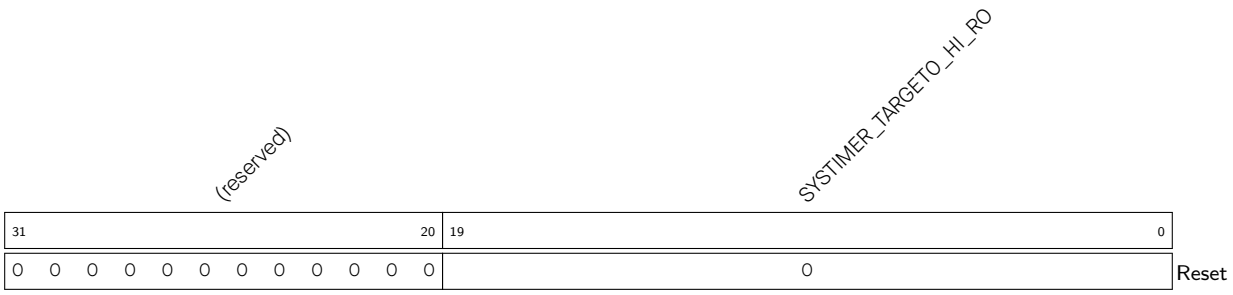
SYSTIMER_TARGET2_INT_ST SYSTIMER_TARGET2_INT 的屏蔽中断状态。(RO)

Register 14.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)



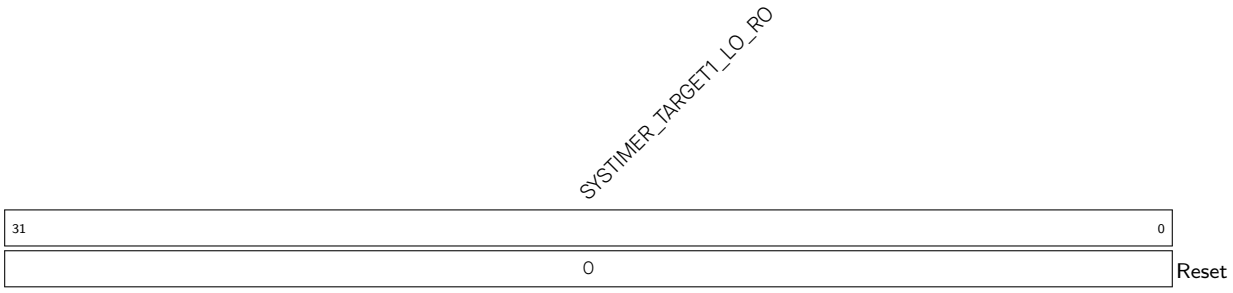
SYSTIMER_TARGET0_LO_RO 表示 COMP0 的实际报警值，低 32 位。(RO)

Register 14.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)



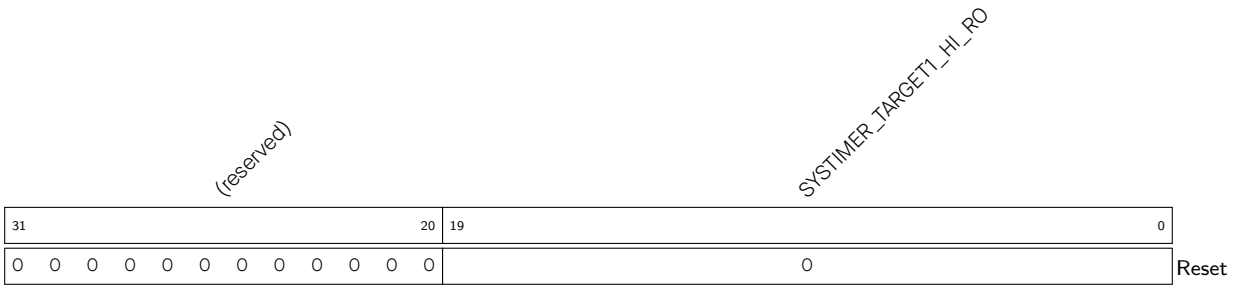
SYSTIMER_TARGET0_HI_RO 表示 COMP0 的实际报警值，高 20 位。(RO)

Register 14.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)



SYSTIMER_TARGET1_LO_RO 表示 COMP1 的实际报警值，低 32 位。(RO)

Register 14.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)



SYSTIMER_TARGET1_HI_RO 表示 COMP1 的实际报警值，高 20 位。(RO)

Register 14.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)



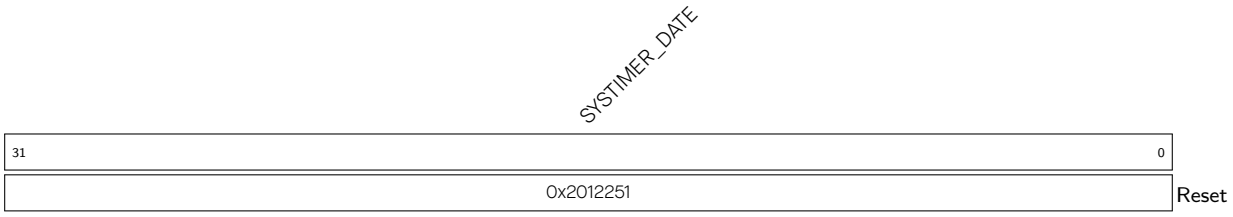
SYSTIMER_TARGET2_LO_RO 表示 COMP2 的实际报警值，低 32 位。(RO)

Register 14.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)



SYSTIMER_TARGET2_HI_RO 表示 COMP2 的实际报警值，高 20 位。(RO)

Register 14.36. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE 版本控制寄存器。(R/W)

第 15 章

定时器组 (TIMG)

15.1 概述

通用定时器可用于准确设定时间间隔、在一定间隔后触发（周期或非周期的）中断或充当硬件时钟。如图 15.1-1 所示，ESP32-C5 包含两个定时器组，即定时器组 0 和定时器组 1（下文用 TIMG*n* 表示，*n* 为 0 或 1）。每个定时器组有一个通用定时器（下文用 T0 表示）和一个主系统看门狗定时器。通用定时器基于 16 位预分频器和 54 位可自动重新加载的可逆计数器。

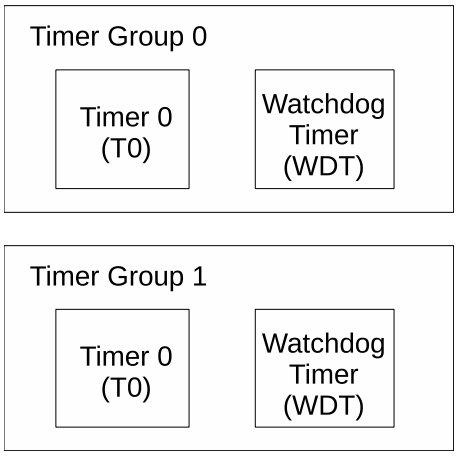


图 15.1-1. 定时器组概览

本章虽然包含主系统看门狗定时器的寄存器描述，但其功能描述请参阅章节 16 *看门狗定时器 (WDT)*。本章中“定时器”指代通用定时器。

15.2 主要特性

定时器具有如下功能：

- 54 位时基计数器，可配置成递增或递减
- 三个时钟源：PLL_F80M_CLK 或 XTAL_CLK 或 RC_FAST_CLK 时钟
- 16 位时钟预分频器，分频系数为 2 到 65536
- 可读取时基计数器的实时值
- 暂停和恢复时基计数器
- 可配置的警报生成机制
- 计数器值重新加载——报警时自动重新加载或软件控制的即时重新加载
- TIMG0 定时器组慢速时钟频率计算

- 电平触发中断
- 支持多个 ETM 任务和事件

15.3 结构概览

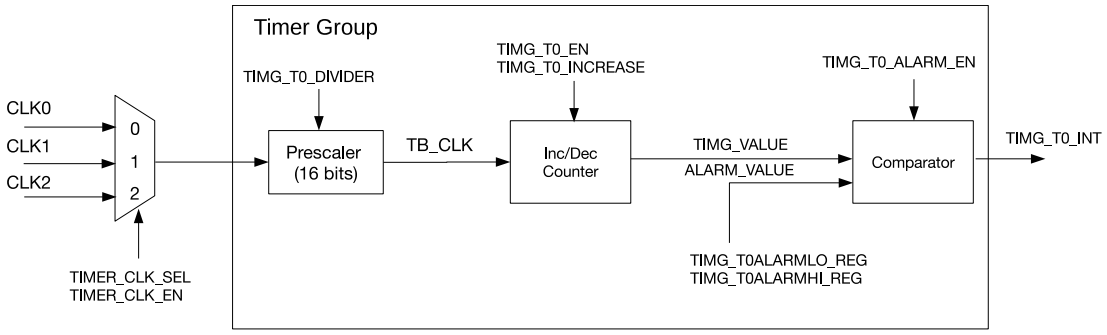


图 15.3-1. 定时器组架构

图 15.3-1 为定时器组的 T0。T0 包含一个 16 位整数预分频器、一个时基计数器和一个用于产生警报的比较器。

15.4 功能描述

15.4.1 16 位预分频器与时钟选择器

以定时器组 0 中的 T0 定时器为例：

- 该定时器可通过配置 TIMER_CLK_SEL 来选择时钟源，TIMER_CLK_SEL 是寄存器 PCR_TIMERGROUP0_TIMER_CLK_CONF_REG 的 PCR_TG0_TIMER_CLK_SEL 字段。该字段为 0 时，选择 CLK0 时钟源进行输入，CLK0 是 XTAL_CLK；为 1 时，选择 CLK1 时钟源进行输入，CLK1 是 RC_FAST_CLK；为 2 时，选择 CLK2 时钟源进行输入，CLK2 是 PLL_F80M_CLK 为时钟源。
- 要开启所选时钟需配置 TIMER_CLK_EN，TIMER_CLK_EN 是 PCR_TIMERGROUP0_TIMER_CLK_CONF_REG 寄存器的 PCR_TG0_TIMER_CLK_EN 字段。要开启时钟需置位该字段，要关闭时钟需清零该字段。所选时钟经 16 位预分频器分频，产生时基计数器使用的时基计数器时钟 (TB_CLK)。16 位预分频器的分频系数可通过 TIMG_T0_DIVIDER 字段配置。

TIMG_T0_DIVIDER 字段可配置为 0 ~ 65535 之间的任意值，分频系数的范围为 2 ~ 65536。具体来说：

- 当 TIMG_T0_DIVIDER 字段置 0 时：分频系数为 65536。
- 当 TIMG_T0_DIVIDER 字段置 1 时：分频系数为 2。
- 当 TIMG_T0_DIVIDER 字段置 2 时：分频系数仍为 2。
- 当 TIMG_T0_DIVIDER 字段置 3 ~ 65535 时：分频系数为 3 ~ 65535。

要更改 16 位预分频器，需先重新配置 TIMG_T0_DIVIDER 字段，再将 TIMG_T0_DIVCNT_RST 置 1，同时需要关闭定时器（即 TIMG_T0_EN 必须清零），否则会造成不可预知的结果。

15.4.2 54 位时基计数器

54 位时基计数器基于 TB_CLK，可通过 TIMG_TO_INCREASE 字段配置为递增或递减。时基计数器可通过置位或清零 TIMG_TO_EN 字段使能或关闭。使能时，时基计数器的值会在每个 TB_CLK 周期递增或递减。关闭时，时基计数器暂停计数。注意，无论 TIMG_TO_EN 是否置位，TIMG_TO_INCREASE 字段都可以更改，时基计数器可立即改变计数方向。

时基计数器 54 位定时器的当前值必须被锁入两个寄存器，才能被 CPU 读取（因为 CPU 为 32 位）。向 TIMG_TOUPDATE_REG 写入任意值时，54 位定时器的值开始被锁入寄存器 TIMG_TOLO_REG 和 TIMG_TOHI_REG，两个寄存器分别锁存低 32 位和高 22 位。当 TIMG_TOUPDATE_REG 被硬件清零，表明锁存操作已经完成，可以从这两个寄存器中读取当前计数值。在 TIMG_TOUPDATE_REG 被写入新值之前，保持寄存器 TIMG_TOLO_REG 和 TIMG_TOHI_REG 的值不变，以供 32 位的 CPU 读值。

15.4.3 警报生成

定时器可配置为在当前值与报警值相同时触发报警。报警会产生中断，同时（可选择）让定时器的当前值自动重新加载（详见第 15.4.4 节）。

54 位报警值可在 TIMG_TOALARMLO_REG 和 TIMG_TOALARMHI_REG 配置，两者分别代表报警值的低 32 位和高 22 位。但是，只有置位 TIMG_TO_ALARM_EN 字段使能报警功能后，配置的报警值才会生效。为解决报警使能“过晚”（即报警使能时，定时器的值已过报警值），出现以下情况时硬件会立即触发报警：

- 可逆计数器向上计数时，定时器的当前值高于报警值（在一定范围内）
- 可逆计数器向下计数时，定时器的当前值低于报警值（在一定范围内）

表 15.4-1 和表 15.4-2 说明了定时器当前值、报警值与报警触发的关系。假设定时器当前值和报警值如下：

- TIMG_VALUE = {TIMG_TOHI_REG, TIMG_TOLO_REG}
- ALARM_VALUE = {TIMG_TOALARMHI_REG, TIMG_TOALARMLO_REG}

表 15.4-1. 可逆计数器向上计数时的报警场景

场景	范围	报警
1	ALARM_VALUE − TIMG_VALUE $> 2^{53}$	触发
2	$0 <$ ALARM_VALUE − TIMG_VALUE $\leq 2^{53}$	可逆计数器向上计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
3	$0 \leq$ TIMG_VALUE − ALARM_VALUE $< 2^{53}$	触发
4	TIMG_VALUE − ALARM_VALUE $\geq 2^{53}$	可逆计数器向上计数达到最大值时，重新开始从 0 向上计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

表 15.4-2. 可逆计数器向下计数时的报警场景

场景	范围	报警
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	触发
6	0 $<$ TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	可逆计数器向下计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
7	0 $\leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	触发
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	可逆计数器向下计数达到最小值时，重新开始从最大值向下计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

报警时，TIMG_TO_ALARM_EN 字段自动清零，在下次置位 TIMG_TO_ALARM_EN 前不会再次报警。

15.4.4 定时器重新加载

定时器重新加载指将定时器的低 32 位和高 22 位分别更新为 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 字段存储的重新加载值。但是，把重新加载值写入 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 字段不会改变定时器的当前值。写入的重新加载值会被定时器忽视，直到重新加载事件被触发。重新加载事件可由软件即时重新加载或报警时自动重新加载触发。

CPU 在寄存器 TIMG_TOLOAD_REG 写任意值会触发软件即时重新加载，定时器的当前值会立即改变。若此时 TIMG_TO_EN 是置位状态，定时器会继续从新数值开始递增或递减计数。若此时 TIMG_TO_ALARM_EN 置位，仍会根据表 15.4-1 或表 15.4-2 中的关系在对应时刻产生报警。若 TIMG_TO_EN 是清零状态，定时器将保持当前值，直至计数重新使能。

报警时，自动重新加载功能可让定时器在报警时重新加载，从重新加载值开始继续递增或递减计数。该功能通常用于周期性报警时重置定时器的值。要使能报警时自动重新加载，需将 TIMG_TO_AUTORELOAD 字段置 1。如未使能该功能，报警后定时器的值会在过报警值后继续递增或递减。

15.4.5 TIMG0 定时器组慢速时钟频率计算

TIMG0 定时器组可以以 XTAL_CLK 为基准时钟，计算 ESP32-C5 提供的慢速时钟的实际频率，输入给定时器组的具体慢速时钟解释请查看章节 17 实时时钟定时器。计算方式如下：

1. 通过周期性或单次计算的方式启动频率计算模块（详见章节 15.7.5）；
2. 在接收到计算开始的信号后，两个分别工作在 XTAL_CLK 以及慢速时钟的计数器同时开始计数，当慢速时钟的计数器达到设定的计算周期 C0 时，同时停止两个计数器；
3. 通过 XTAL_CLK 的计数器值 C1 即可计算慢速时钟的时钟频率：$f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

请注意慢速时钟频率计算功能使用时应保证输入频率不大于 $\frac{f\_XTAL\_CLK}{2}$。

15.5 事件任务矩阵功能

在 ESP32-C5 中，定时器组支持 ETM 功能，即可以通过任意外设的 ETM 事件触发定时器组的 ETM 任务，或者通过定时器组的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与定时器组相关的 ETM 任务和 ETM 事件。

定时器组可接收的 ETM 任务有：

- TG0_TASK_CNT_START_TIMER0：触发时启用时基计数器。
- TG1_TASK_CNT_START_TIMER0：触发时启用时基计数器。
- TG0_TASK_CNT_STOP_TIMER0：触发时关闭时基计数器。
- TG1_TASK_CNT_STOP_TIMER0：触发时关闭时基计数器。

> **说明：**
> 以上两个 ETM 任务与 APB 配置 TIMG_T0_EN 实现的功能相同。当这些操作同时发生时，各操作优先级由高到低如下：
>
> 1. TG0_TASK_CNT_START_TIMER0 与 TG1_TASK_CNT_START_TIMER0：触发时启用时基计数器
> 2. TG0_TASK_CNT_STOP_TIMER0 与 TG1_TASK_CNT_STOP_TIMER0：触发时关闭时基计数器
> 3. APB 方式配置 TIMG_T0_EN：启用或关闭时基计数器

- TG0_TASK_ALARM_START_TIMER0：触发时启用警报生成。
- TG1_TASK_ALARM_START_TIMER0：触发时启用警报生成。

> **说明：**
> 通过 APB 方式配置 TIMG_ALARM_EN 以及硬件事件触发也可以启用警报生成功能。当这些操作同时发生时，各个操作的优先级从高到低依次为：
>
> 1. TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0：触发时启用警报生成
> 2. 警报事件：触发时关闭警报生成
> 3. APB 方式配置 TIMG_ALARM_EN：启用/关闭警报生成

- TG0_TASK_CNT_CAP_TIMER0：触发时会将当前计数器的值更新至 TIMG_T0LO_REG 和 TIMG_T0HI_REG 寄存器。
- TG1_TASK_CNT_CAP_TIMER0：触发时会将当前计数器的值更新至 TIMG_T0LO_REG 和 TIMG_T0HI_REG 寄存器。
- TG0_TASK_CNT_RELOAD_TIMER0：触发时会用存储在 TIMG_T0_LOAD_LO 和 TIMG_T0_LOAD_HI 中的重新加载值覆盖当前计数器值。
- TG1_TASK_CNT_RELOAD_TIMER0：触发时会用存储在 TIMG_T0_LOAD_LO 和 TIMG_T0_LOAD_HI 中的重新加载值覆盖当前计数器值。

定时器组可产生的 ETM 事件有：

- TG0_EVT_CNT_CMP_TIMER0：表示 TIMG0 的 T0 定时器中断事件。
- TG1_EVT_CNT_CMP_TIMER0：表示 TIMG1 的 T0 定时器中断事件。

只有当 TIMG_ETM_EN 设置为 1 时，ETM 任务和事件才会生效。

在具体应用中，定时器组的 ETM 事件可以用来触发定时器组的 ETM 任务。例如，TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件分别可以触发 TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务，从而实现周期性报警。具体配置步骤请参考 15.7.4 通过 ETM 设置定时器用于周期性报警。

15.6 中断

ESP32-C5 的 TIMG*n* 可以生成以下中断信号并发送给中断矩阵。

- TG*n*_T0_INTR
- TG*n*_WDT_INTR

这些中断信号由 TIMG*n* 的内部中断源生成。表 15.6-1 列出了 TIMG*n* 的内部中断源、中断触发条件、以及生成的中断信号。

表 15.6-1. TIMG*n* 的内部中断源

内部中断源	触发条件	中断信号
TIMG_T0_INT	T0 定时器产生告警	TG*n*_T0_INTR
TIMG_WDT_INT	看门狗定时器产生告警	TG*n*_WDT_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 15.8 寄存器列表。

15.7 配置流程

15.7.1 定时器用作简单时钟

1. 配置时基计数器。
   - 置位或清除 PCR_TGO_TIMER_CLK_SEL 字段选择时钟源。
   - 置位 TIMG_T0_DIVIDER 配置 16 位预分频器。
   - 置位或清除 TIMG_T0_INCREASE 配置定时器方向。
   - 在 TIMG_T0_LOAD_LO 和 TIMG_T0_LOAD_HI 上写初始值设置定时器的初始值，然后在 TIMG_T0LOAD_REG 上写任意值将初始值重新加载进定时器。
2. 置位 TIMG_T0_EN 开启定时器。
3. 获得定时器的当前值。
   - 在 TIMG_T0UPDATE_REG 上写任意值锁存定时器的当前值。
   - 等待硬件将 TIMG_T0UPDATE_REG 清 0。
   - 从 TIMG_T0LO_REG 和 TIMG_T0HI_REG 读取锁存的定时器值。

15.7.2 定时器用于单次报警

1. 按照第 15.7.1 节的第 1 步配置时基计数器。
2. 配置报警。

- 置位 TIMG_TOALARMLO_REG 和 TIMG_TOALARMHI_REG 配置报警值。
- 置位 TIMG_TO_INT_ENA 使能中断。

3. 清零 TIMG_TO_AUTORELOAD 关闭自动重新加载。
4. 置位 TIMG_TO_ALARM_EN 开启报警。
5. 处理报警中断。
   - 置位定时器在 TIMG_TO_INT_CLR 的对应位清除中断。
   - 清零 TIMG_TO_EN 关闭定时器。

15.7.3 通过 APB 设置定时器用于周期性报警

1. 按照第 15.7.1 节的第 1 步配置时基计数器。
2. 按照第 15.7.2 节的第 2 步配置报警。
3. 置位 TIMG_TO_AUTORELOAD 使能自动重新加载，将重新加载值写入 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI。
4. 置位 TIMG_TO_ALARM_EN 开启报警。
5. 处理报警中断（每次报警时重复）。
   - 置位定时器在 TIMG_TO_INT_CLR 的对应位清除中断。
   - 如下一次报警需要新的报警值和重新加载值（即每次都有不同的报警间隔），则应根据需要重新配置 TIMG_TOALARMLO_REG、TIMG_TOALARMHI_REG、TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 。否则，上述寄存器应保持不变。
   - 置位 TIMG_TO_ALARM_EN 重新使能报警。
6. （最后一次报警时）关闭定时器。
   - 置位定时器在 TIMG_TO_INT_CLR 的对应位清除中断。
   - 清零 TIMG_TO_EN 关闭定时器。

15.7.4 通过 ETM 设置定时器用于周期性报警

1. 使能 ETM 模块的时钟
2. 将 ETM 事件映射到相应的 ETM 任务（使用 ETM 事件触发对应的 ETM 任务）
   - 如果 TIMG_TO_AUTORELOAD 置 1，通过一个 ETM 通道映射 TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件到 TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务。
   - 如果 TIMG_TO_AUTORELOAD 置 0，除了映射 TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件到 TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务外，还需要用另一个 ETM 通道映射 TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件到 TG0_TASK_CNT_RELOAD_TIMER0 与 TG1_TASK_CNT_RELOAD_TIMER0 任务。
3. 选择使能一个或两个 ETM 通道。

4. 将 TIMER_ETM_EN 置 1 从而使能定时器组的 ETM 事件和任务。
5. 按照第 15.7.1 节的第 1 步配置时基计数器。
6. 按照第 15.7.2 节的第 2 步配置报警。
7. 通过 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 配置重新加载值。
8. 处理 TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0
   - 当警报生成时，TG0_EVT_CNT_CMP_TIMER0 与 TG1_EVT_CNT_CMP_TIMER0 事件也会产生，此时报警器关闭警报生成。
   - 如果 TIMG_TO_AUTORELOAD 为 1，重新加载值会覆盖当前计数器的值。TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 会重新使能警报生成。
   - 如果 TIMG_TO_AUTORELOAD 为 0，由于 TG0_TASK_CNT_RELOAD_TIMER0 与 TG1_TASK_CNT_RELOAD_TIMER0 任务，重新加载值会覆盖当前计数器的值。TG0_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0 任务会重新使能警报生成。
9. （最后一次报警时）关闭定时器。
   - 关闭用于映射定时器组事件和任务的 ETM 通道。
   - 设置 TIMER_ETM_EN 为 0。
   - 置位定时器在 TIMG_T0_INT_CLR 的对应位清除中断。
   - 清零 TIMG_T0_EN 关闭定时器。

15.7.5 慢速时钟频率计算

1. 单次计算
   - 设置 PCR_32_SEL 选择需要计算频率的时钟，该寄存器设置值与时钟源具体对应关系参考章节 17 实时时钟定时器。
   - 设置 PCR_FOSC_TICK_NUM 对频率较高的 RC_FAST_CLK 进行分频，以满足慢速时钟频率计算的前提条件。
   - 设置 TIMG_RTC_CALI_MAX 配置频率计算时间。
   - 清空 TIMG_RTC_CALI_START_CYCLING 选择单次校准模式，然后配置 TIMG_RTC_CALI_START 开启两个计数器。
   - 等待 TIMG_RTC_CALI_RDY 的值变为 1，读取 TIMG_RTC_CALI_VALUE 获取 XTAL_CLK 计数器值，根据章节 15.4.5 的公式计算慢速时钟的频率。
2. 周期性计算
   - 设置 PCR_32_SEL 选择需要计算频率的时钟。
   - 设置 PCR_FOSC_TICK_NUM 对频率较高的 RC_FAST_CLK 进行分频，以满足慢速时钟频率计算的前提条件。
   - 设置 TIMG_RTC_CALI_MAX 配置频率计算时间。
   - 使能 TIMG_RTC_CALI_START_CYCLING，硬件将不间断进行频率计算过程。

- 只要 TIMG_RTC_CALI_CYCLING_DATA_VLD 为 1，即表示 TIMG_RTC_CALI_VALUE 有效。

3. 超时
   如果慢速时钟的计数器没有在 TIMG_RTC_CALI_TIMEOUT_RST_CNT 的 XTAL_CLK 计数器内完成计数，将置位 TIMG_RTC_CALI_TIMEOUT 标记计算超时。

15.8 寄存器列表

本小节的所有地址均为相对于 定时器组 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

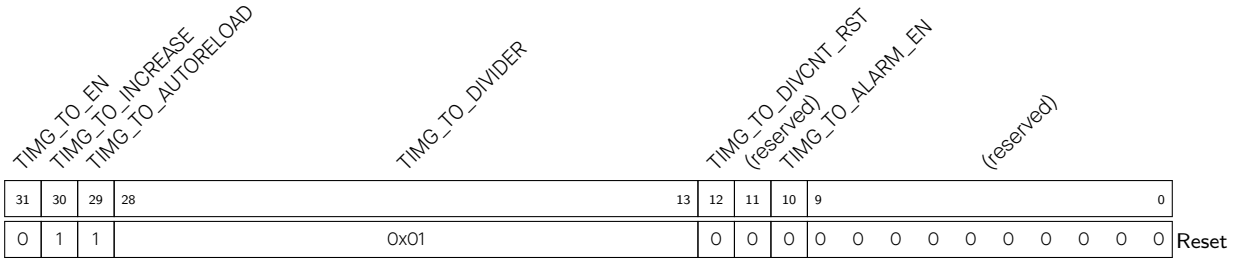
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
定时器 0 控制和配置寄存器			
TIMG_T0CONFIG_REG	定时器 0 配置寄存器	0x0000	varies
TIMG_T0LO_REG	定时器 0 的当前值，低 32 位	0x0004	RO
TIMG_T0HI_REG	定时器 0 的当前值，高 22 位	0x0008	RO
TIMG_T0UPDATE_REG	写值将当前定时器的值复制到 TIMG_T0LO_REG 或 TIMG_T0HI_REG	0x000C	R/ W/ SC
TIMG_T0ALARMLO_REG	定时器 0 的报警值，低 32 位	0x0010	R/W
TIMG_T0ALARMHI_REG	定时器 0 的报警值，高 22 位	0x0014	R/W
TIMG_T0LOADLO_REG	定时器 0 的重新加载值，低 32 位	0x0018	R/W
TIMG_T0LOADHI_REG	定时器 0 的重新加载值，高 22 位	0x001C	R/W
TIMG_T0LOAD_REG	写值从 TIMG_T0LOADLO_REG 或 TIMG_T0LOADHI_REG 上加载定时器	0x0020	WT
看门狗定时器控制和配置寄存器			
TIMG_WDTCONFIG0_REG	看门狗定时器配置寄存器	0x0048	varies
TIMG_WDTCONFIG1_REG	看门狗定时器预分频器寄存器	0x004C	varies
TIMG_WDTCONFIG2_REG	看门狗定时器阶段 0 超时值	0x0050	R/W
TIMG_WDTCONFIG3_REG	看门狗定时器阶段 1 超时值	0x0054	R/W
TIMG_WDTCONFIG4_REG	看门狗定时器阶段 2 超时值	0x0058	R/W
TIMG_WDTCONFIG5_REG	看门狗定时器阶段 3 超时值	0x005C	R/W
TIMG_WDTFEED_REG	写值喂看门狗定时器	0x0060	WT
TIMG_WDTWPROTECT_REG	看门狗写保护寄存器	0x0064	R/W
RTC 校准控制和配置寄存器			
TIMG_RTCCALICFG_REG	RTC 校准配置寄存器	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC 校准配置寄存器 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC 校准配置寄存器 2	0x0080	varies
中断寄存器			
TIMG_INT_ENA_TIMERS_REG	中断使能位	0x0070	R/W
TIMG_INT_RAW_TIMERS_REG	原始中断状态	0x0074	R/ SS/ WTC
TIMG_INT_ST_TIMERS_REG	屏蔽中断状态	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	中断清除位	0x007C	WT
版本寄存器			
TIMG_NTIMERS_DATE_REG	版本控制寄存器	0x00F8	R/W
时钟配置寄存器			
TIMG_REGCLK_REG	定时器组时钟门控寄存器	0x00FC	R/W

15.9 寄存器

本小节的所有地址均为相对于 定时器组 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 15.1. TIMG_T0CONFIG_REG (0x0000)



TIMG_T0_ALARM_EN 配置是否使能定时器 0 的报警功能。报警时，此位自动清零。

0：禁用

1：使能

(R/W/SC)

TIMG_T0_DIVCNT_RST 配置是否复位定时器 0 时钟分频器的计数器。

0：无效

1：复位

(WT)

TIMG_T0_DIVIDER 表示定时器 0 时钟 (T0_clk) 的预分频器值。(R/W)

TIMG_T0_AUTORELOAD 配置是否使能定时器 0 报警时自动重新加载使能。

0：无效

1：使能

(R/W)

TIMG_T0_INCREASE 配置定时器 0 的时基计数器的计数方向。

0：递减

1：递增

(R/W)

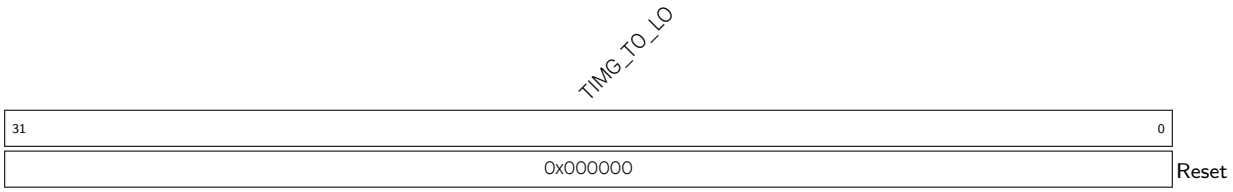
TIMG_T0_EN 配置是否使能定时器 0 的时基计数器。

0：禁用

1：使能

(R/W/SS/SC)

Register 15.2. TIMG_T0LO_REG (0x0004)

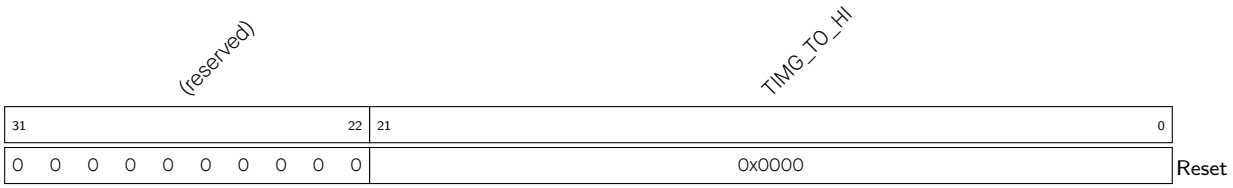


TIMG_T0_LO 表示定时器 0 时基计数器的低 32 位的值。仅在 TIMG_T0UPDATE_REG 写值后有效。

计数单位：T0_clk。

(RO)

Register 15.3. TIMG_T0HI_REG (0x0008)

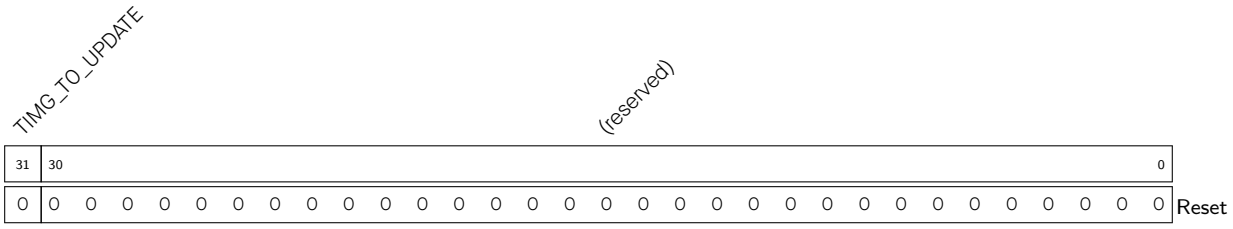


TIMG_T0_HI 表示定时器 0 时基计数器的高 22 位的值。仅在 TIMG_T0UPDATE_REG 写值后有效。

计数单位：T0_clk。

(RO)

Register 15.4. TIMG_T0UPDATE_REG (0x000C)



TIMG_T0_UPDATE 配置后会锁存计数器的值。

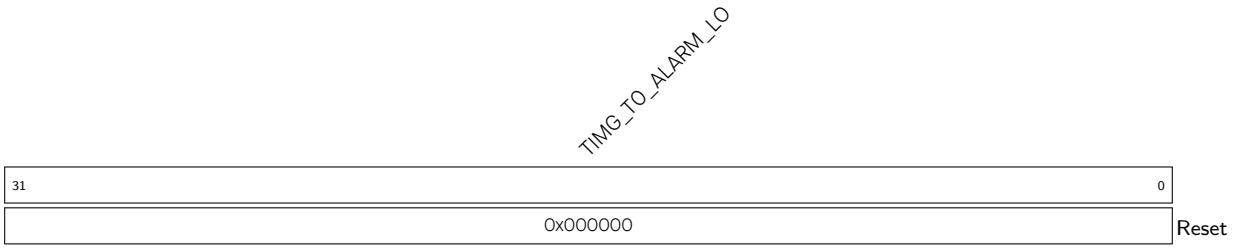
0：锁存

1：锁存

注意：无论该位写 0 或写 1，计数器的值均将被锁存。

(R/W/SC)

Register 15.5. TIMG_T0ALARMLO_REG (0x0010)

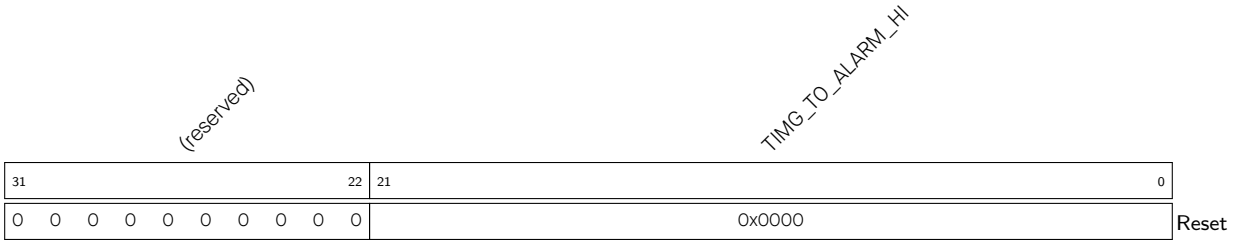


TIMG_T0_ALARM_LO 配置定时器 0 时基计数器触发报警值的低 32 位。仅在 TIMG_T0_ALARM_EN 为 1 时有效。

计数单位：T0_clk。

(R/W)

Register 15.6. TIMG_T0ALARMHI_REG (0x0014)

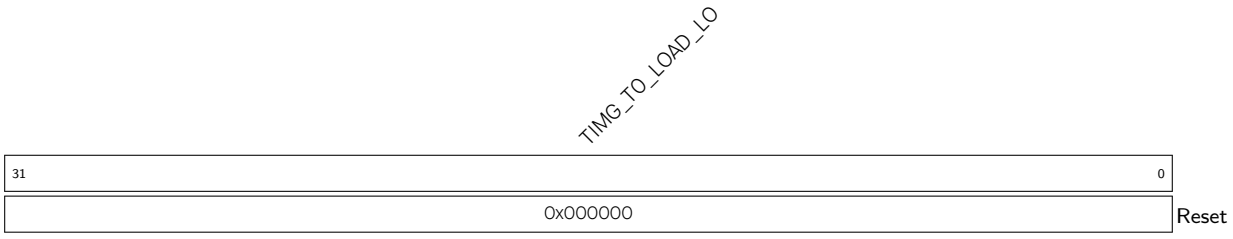


TIMG_T0_ALARM_HI 配置定时器 0 时基计数器触发报警值的高 22 位。仅在 TIMG_T0_ALARM_EN 为 1 时有效。

计数单位：T0_clk。

(R/W)

Register 15.7. TIMG_T0LOADLO_REG (0x0018)

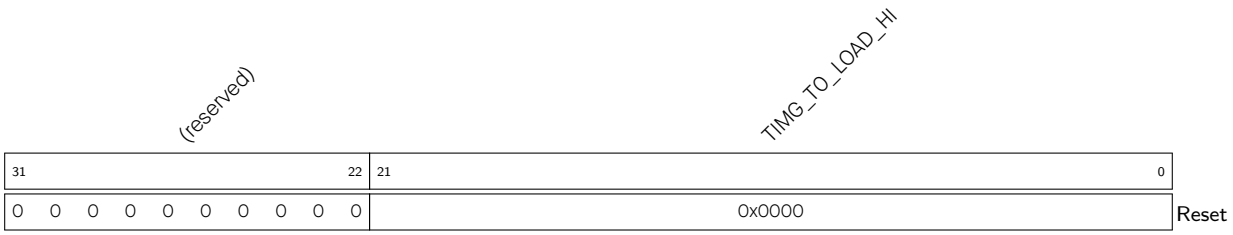


TIMG_T0_LOAD_LO 配置定时器 0 时基计数器重新加载的低 32 位值。

计数单位：T0_clk。

(R/W)

Register 15.8. TIMG_T0LOADHI_REG (0x001C)

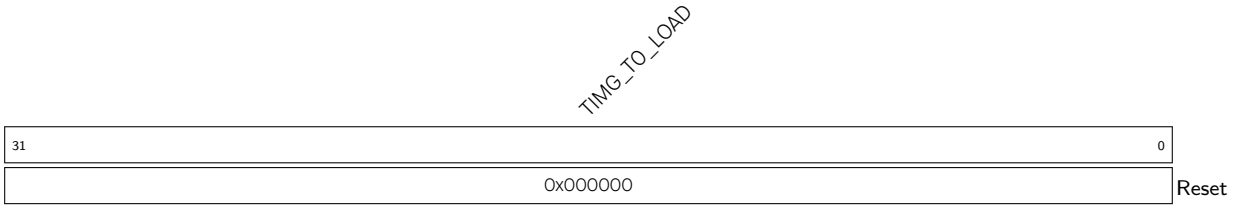


TIMG_T0_LOAD_HI 配置定时器 0 时基计数器重新加载的高 22 位值。

计数单位：T0_clk。

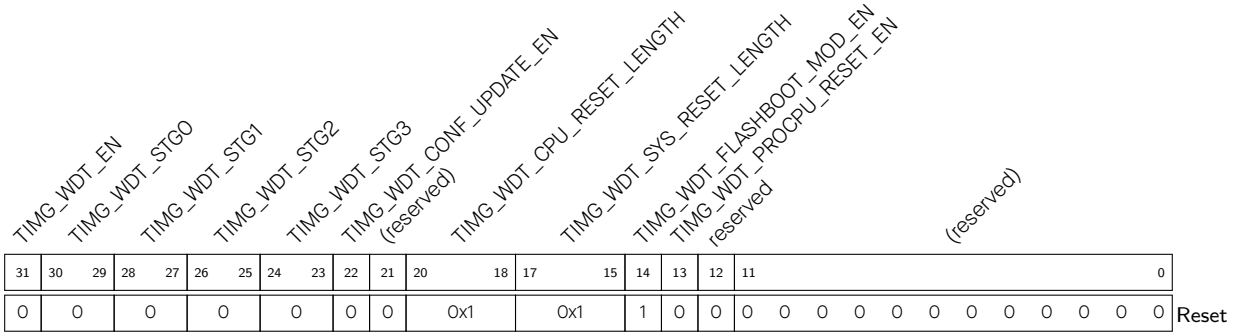
(R/W)

Register 15.9. TIMG_T0LOAD_REG (0x0020)



TIMG_T0_LOAD 写任意值触发定时器 0 时基计数器重新加载。(WT)

Register 15.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_PROCPU_RESET_EN 配置是否屏蔽 MWDT 产生的 CPU 复位。单核默认使用，仅在禁用写保护时有效。

0：屏蔽

1：不屏蔽

(R/W)

TIMG_WDT_FLASHBOOT_MOD_EN 配置是否使能 flash 启动保护。

0：禁用

1：使能

(R/W)

TIMG_WDT_SYS_RESET_LENGTH 配置系统复位信号长度。仅在禁用写保护时有效。

计数单位：mwdt_clk。

0：8

1：16

2：24

3：32

4：40

5：64

6：128

7：256

(R/W)

TIMG_WDT_CPU_RESET_LENGTH 配置 CPU 复位信号长度。仅在禁用写保护时有效。

计数单位：mwdt_clk。

0：8

1：16

2：24

3：32

4：40

5：64

6：128

7：256

(R/W)

见下页...

Register 15.10. USB_SERIAL_JTAG_CONF0_REG (0x0018)

接上页...

TIMG_WDT_CONF_UPDATE_EN 配置以更新看门狗定时器的配置寄存器。

0：无效

1：更新

(WT)

TIMG_WDT_STG3 配置阶段 3 的超时动作。具体可参考 TIMG_WDT_STG0。仅在禁用写保护时有效。

(R/W)

TIMG_WDT_STG2 配置阶段 2 的超时动作。具体可参考 TIMG_WDT_STG0。仅在禁用写保护时有效。

(R/W)

TIMG_WDT_STG1 配置阶段 1 的超时动作。具体可参考 TIMG_WDT_STG0。仅在禁用写保护时有效。

(R/W)

TIMG_WDT_STG0 配置阶段 0 的超时动作。仅在禁用写保护时有效。

0：无效

1：中断

2：复位 CPU

3：复位系统

(R/W)

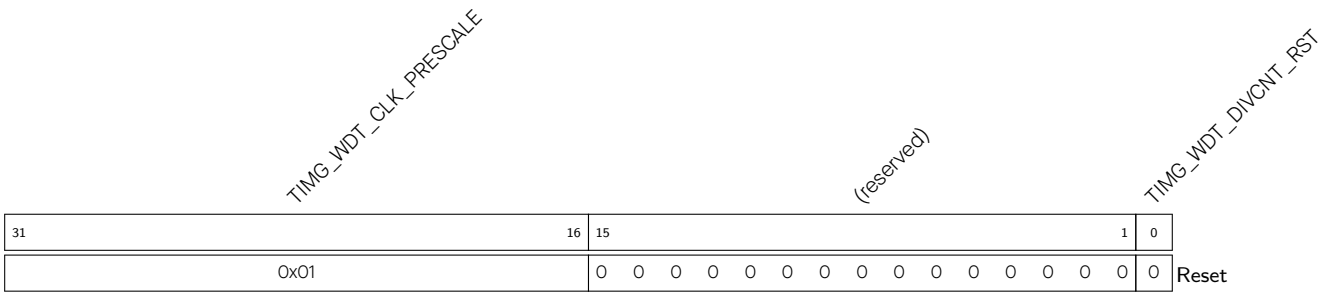
TIMG_WDT_EN 配置是否使能 MWDT。仅在禁用写保护时有效。

0：禁用

1：使能

(R/W)

Register 15.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST 配置是否复位看门狗定时器时钟分频器的计数器。

0：无效

1：复位

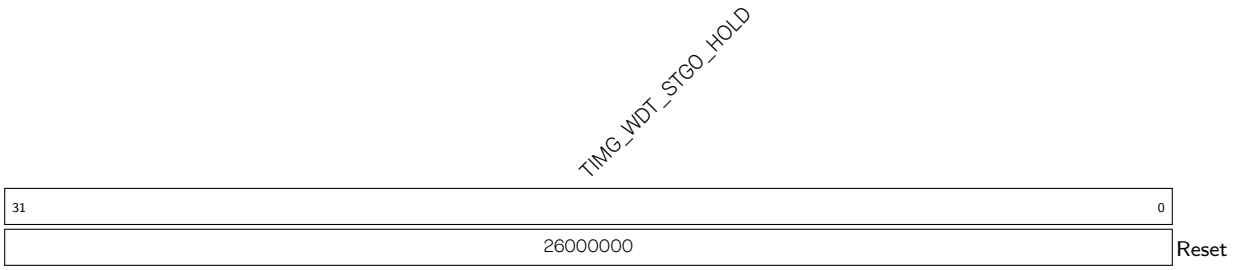
(WT)

TIMG_WDT_CLK_PRESCALE 配置 MWDT 时钟预分频器值。仅在禁用写保护时有效。

MWDT 时钟周期 = MWDT 时钟源周期 * TIMG_WDT_CLK_PRESCALE。

(R/W)

Register 15.12. TIMG_WDTCONFIG2_REG (0x0050)

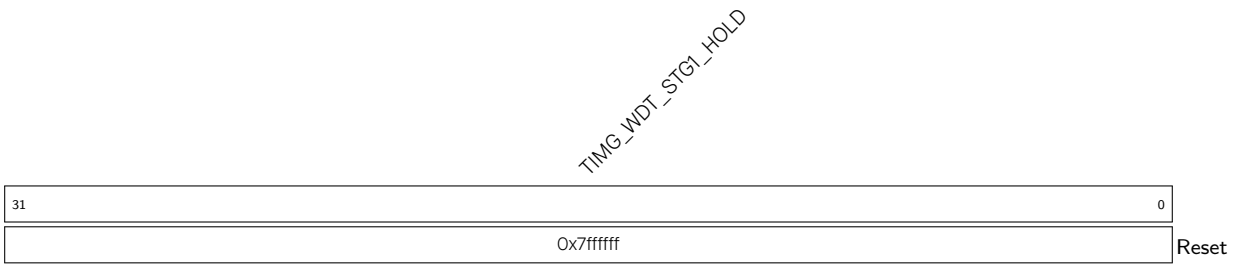


TIMG_WDT_STG0_HOLD 配置阶段 0 超时时间。仅在禁用写保护时有效。

计数单位：mwdt_clk。

(R/W)

Register 15.13. TIMG_WDTCONFIG3_REG (0x0054)

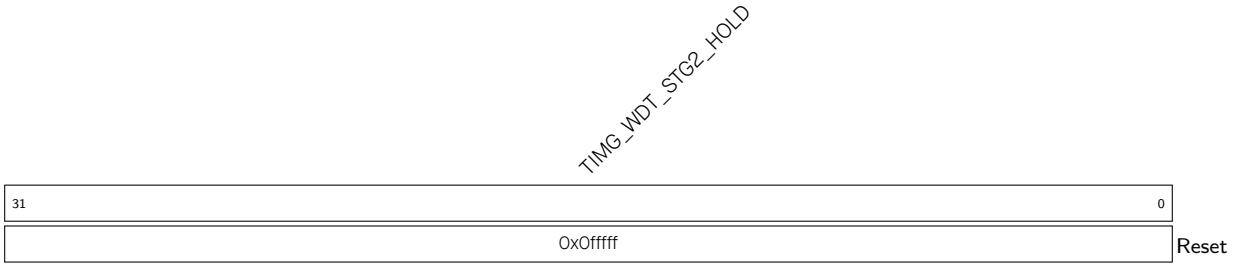


TIMG_WDT_STG1_HOLD 配置阶段 1 超时时间。仅在禁用写保护时有效。

计数单位：mwdt_clk。

(R/W)

Register 15.14. TIMG_WDTCONFIG4_REG (0x0058)

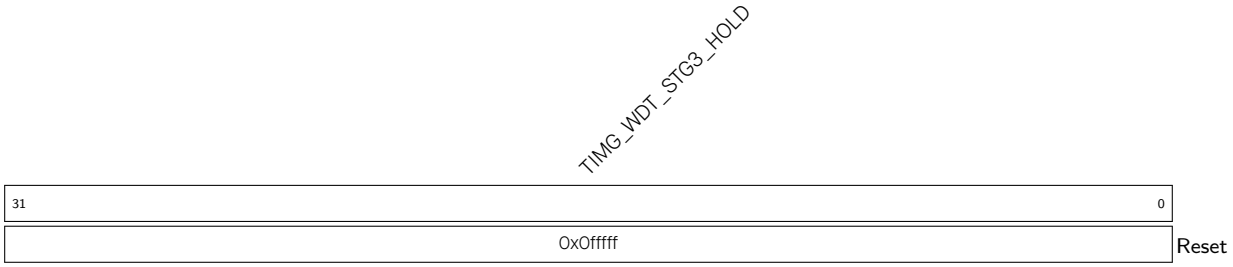


TIMG_WDT_STG2_HOLD 配置阶段 2 超时时间。仅在禁用写保护时有效。

计数单位：mwdt_clk。

(R/W)

Register 15.15. TIMG_WDTCONFIG5_REG (0x005C)

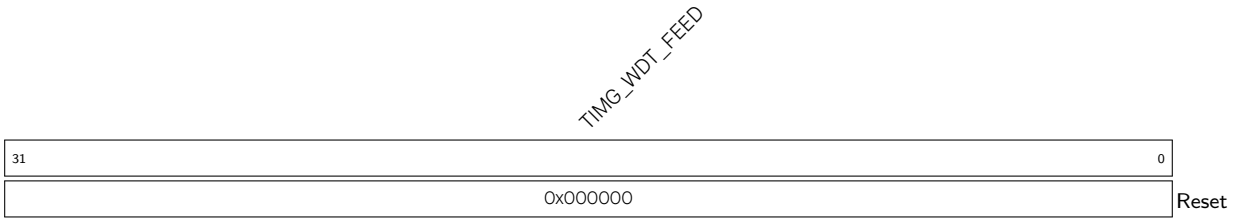


TIMG_WDT_STG3_HOLD 配置阶段 3 超时时间。仅在禁用写保护时有效。

计数单位：mwdt_clk。

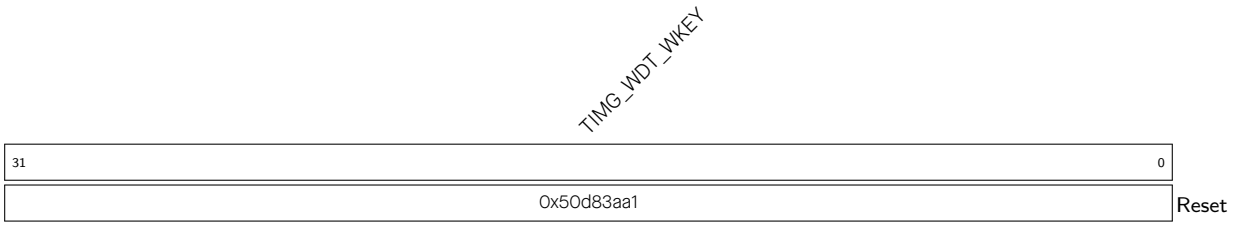
(R/W)

Register 15.16. TIMG_WDTFEED_REG (0x0060)



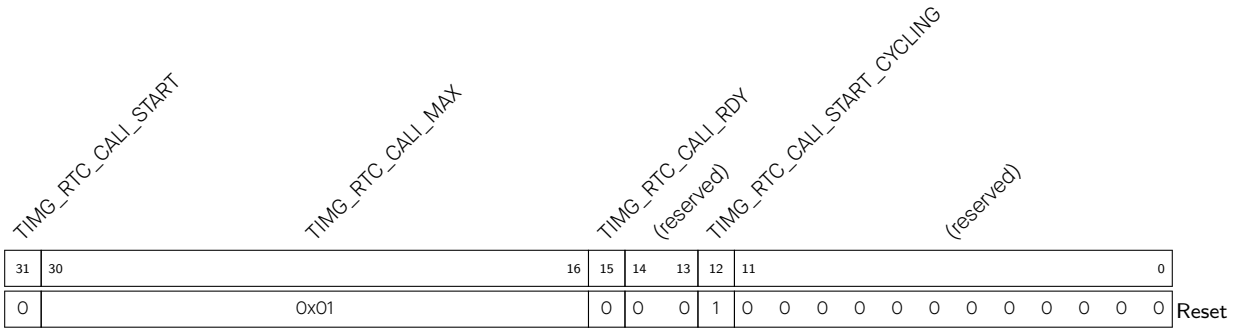
TIMG_WDT_FEED 写任意值喂 MWDT。仅在禁用写保护时有效。(WT)

Register 15.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY 配置一个与复位值不同的值以使能写保护。(R/W)

Register 15.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING 配置频率计算模式。

0：单次频率计算模式；

1：周期性频率计算模式。

(R/W)

TIMG_RTC_CALI_RDY 表示单次频率计算是否完成。

0：未完成

1：完成

(RO)

TIMG_RTC_CALI_MAX 配置计算 RTC 慢速时钟 SLOW_CLK 频率的时间。

计数单位：XTAL_CLK。

(R/W)

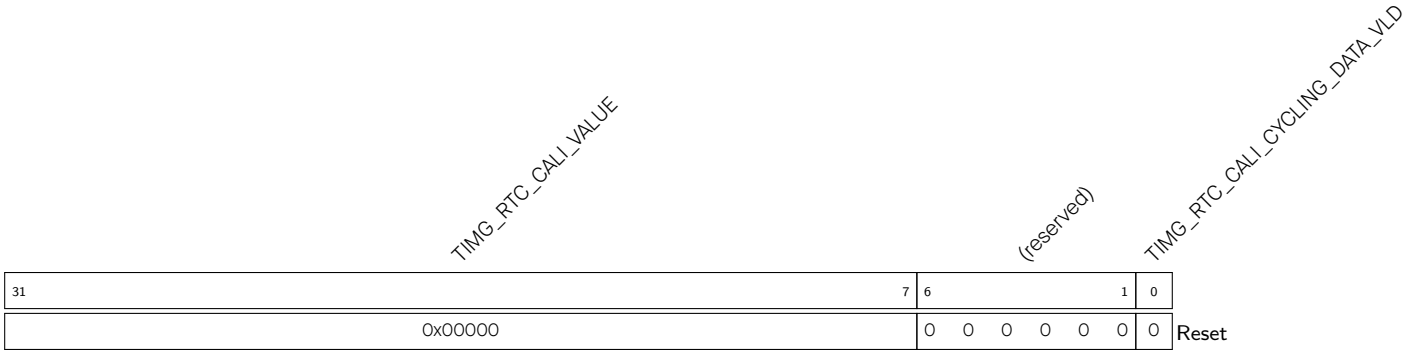
TIMG_RTC_CALI_START 配置是否使能单次频率计算。

0：禁用

1：使能

(R/W)

Register 15.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD 表示周期性频率计算是否完成。

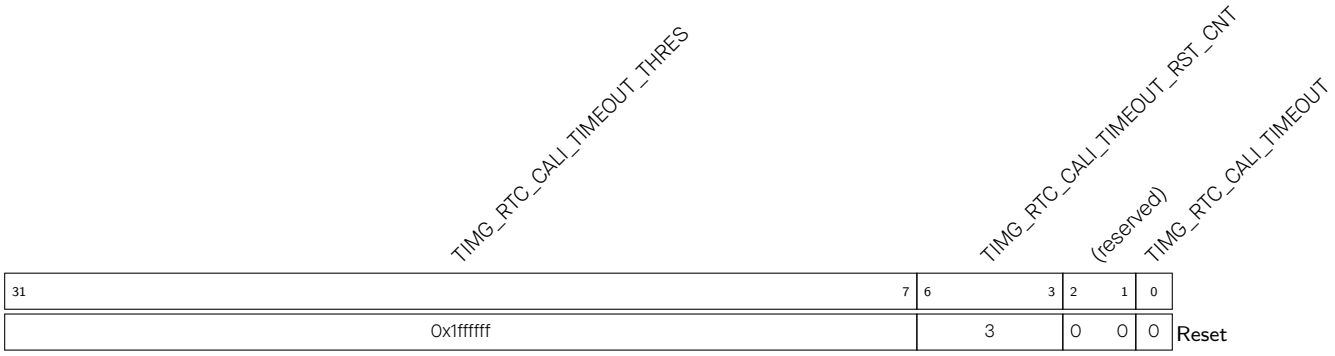
0：未完成

1：完成

(RO)

TIMG_RTC_CALI_VALUE 表示单次或周期性频率计算完成时 XTAL_CLK 计数的值。用于计算 RTC 慢时钟的频率。(RO)

Register 15.20. TIMG_RTCCALICFG2_REG (0x0080)



TIMG_RTC_CALI_TIMEOUT 表示频率计算是否超时。

0：未超时

1：超时

(RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT 配置频率计算超时复位的周期。

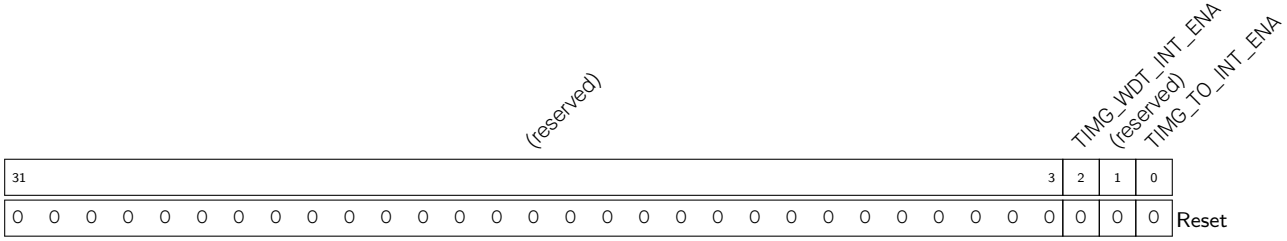
计数单位：XTAL_CLK。(R/W)

TIMG_RTC_CALI_TIMEOUT_THRES 配置 RTC 频率计算定时器的阈值。如果频率计算定时器的值超过此值时触发超时。

计数单位：XTAL_CLK。

(R/W)

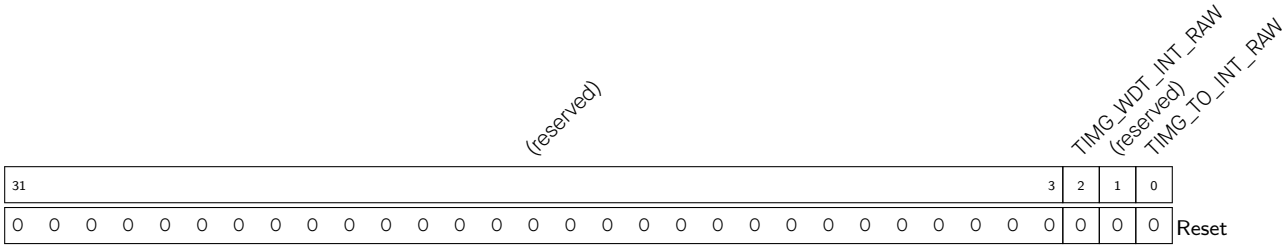
Register 15.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



TIMG_TO_INT_ENA 写 1 使能 TIMG_TO_INT 中断。(R/W)

TIMG_WDT_INT_ENA 写 1 使能 TIMG_WDT_INT 中断。(R/W)

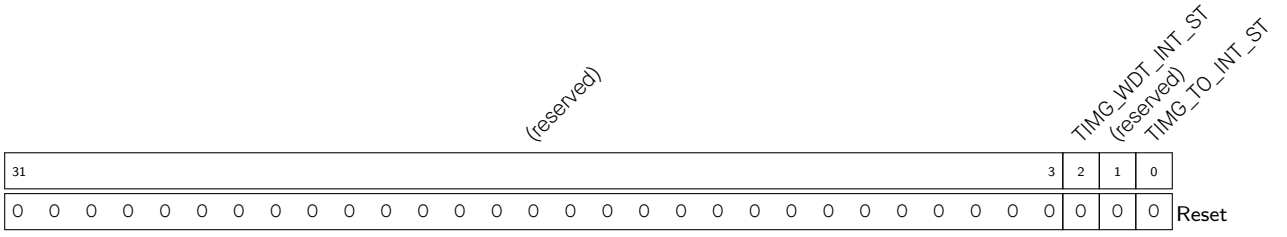
Register 15.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_TO_INT_RAW TIMG_TO_INT 中断的原始中断状态位。(R/SS/WTC)

TIMG_WDT_INT_RAW TIMG_WDT_INT 中断的原始中断状态位。(R/SS/WTC)

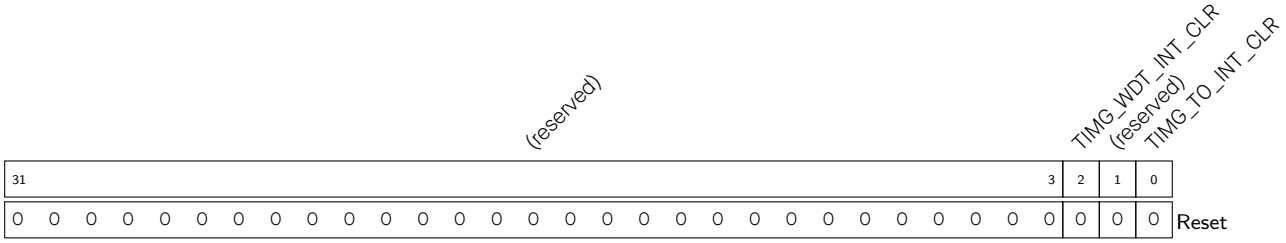
Register 15.23. TIMG_INT_ST_TIMERS_REG (0x0078)



TIMG_TO_INT_ST TIMG_TO_INT 中断的屏蔽中断状态位。(RO)

TIMG_WDT_INT_ST TIMG_WDT_INT 中断的屏蔽中断状态位。(RO)

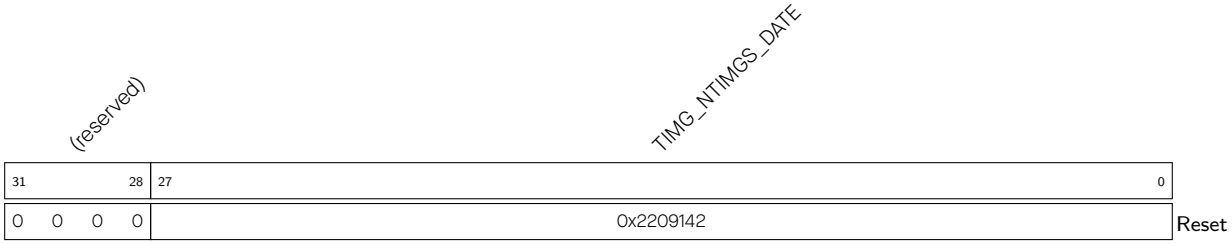
Register 15.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_TO_INT_CLR 写 1 清除 TIMG_TO_INT 中断。(WT)

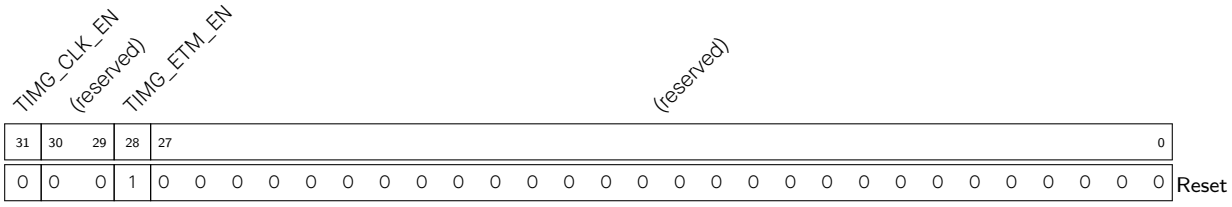
TIMG_WDT_INT_CLR 写 1 清除 TIMG_WDT_INT 中断。(WT)

Register 15.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE 版本控制寄存器。(R/W)

Register 15.26. TIMG_REGCLK_REG (0x00FC)



TIMG_ETM_EN 配置是否使能定时器的 ETM 任务和事件。

0：禁用

1：使能

(R/W)

TIMG_CLK_EN 配置是否使能寄存器的门时钟信号。

0：强制使能寄存器的时钟

1：仅在软件读取或写入寄存器时启动时钟。

(R/W)

第 16 章

看门狗定时器 (WDT)

16.1 概述

看门狗定时器是一种硬件定时器，用于检测和修复故障。软件必须定期喂狗（复位），以防超时。系统或软件若出现不可预知的问题（比如软件卡在某个循环或逾期事件中）将无法按时喂狗，造成看门狗超时。因此，看门狗定时器有助于检测、处理系统或软件的错误行为。

如图 16.1-1 所示，ESP32-C5 中有三个数字看门狗定时器：章节 15 定时器组 *(TIMG)* 描述的两个定时器组中各有一个（称作主系统看门狗定时器，缩写为 MWDT），RTC 模块中有一个（称作 RTC 看门狗定时器，缩写为 RWDT）。数字看门狗在运行期间会经历四个阶段（除非看门狗按时喂狗或者处于关闭状态），每个阶段均可配置单独的超时时间和超时动作，其中 MWDT 支持中断、CPU 复位和内核复位三种超时动作，RWDT 支持中断、CPU 复位、内核复位和系统复位四种超时动作（详见章节 16.2.2.2 阶段与超时动作）。每个阶段的超时时间都可单独设置。

在 flash 引导模式下，RWDT 和定时器组 0 的 MWDT 会默认使能，以检测引导过程中发生的错误，并恢复运行。

ESP32-C5 中还有一个模拟看门狗定时器——超级看门狗 (SWD)。超级看门狗是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。

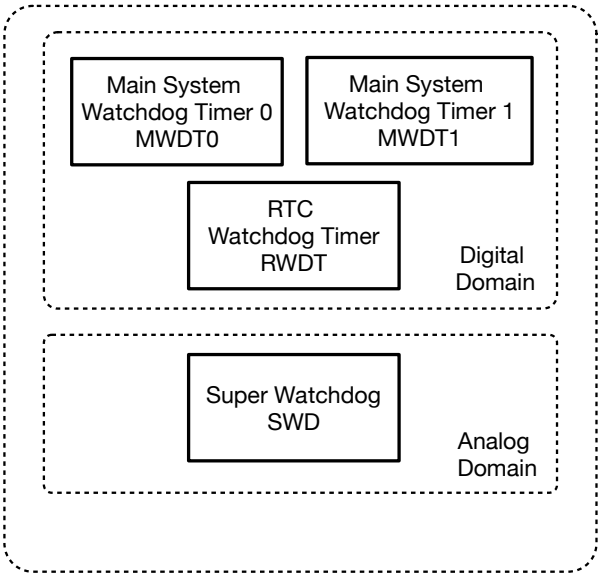


图 16.1-1. 看门狗定时器概览

请注意，本章节提供了看门狗定时器的功能描述，而 MWDT 寄存器部分详见章节 15 定时器组 *(TIMG)*，RWDT 和 SWD 寄存器部分详见 16.5 寄存器列表。

16.2 数字看门狗定时器

16.2.1 主要特性

看门狗定时器具有如下特性：

- 四个阶段，每个阶段都可配置超时时间和超时动作
- 超时动作
  - MWDT：中断、CPU 复位、内核复位
  - RWDT：中断、CPU 复位、内核复位、系统复位
- 阶段 0 Flash 启动保护：
  - MWDT0：超时触发内核复位
  - RWDT：超时触发系统复位
- 写保护，使能时寄存器仅可读取
- 32 位超时计数器
- 时钟源：
  - MWDT：PLL_F80M_CLK、RC_FAST_CLK 或 XTAL_CLK
  - RWDT：RTC_SLOW_CLK

16.2.2 功能描述

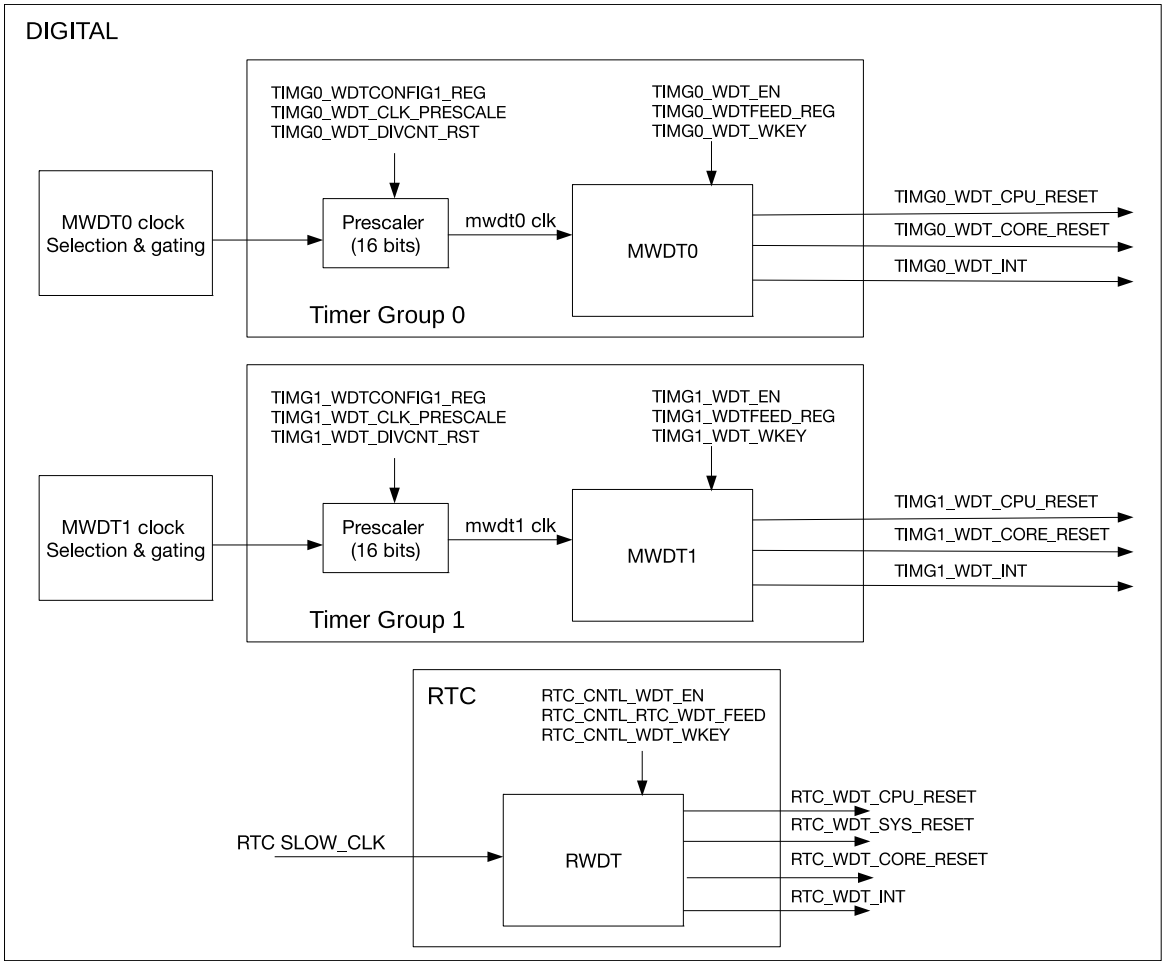


图 16.2-1. ESP32-C5 的数字看门狗定时器

图 16.2-1 为 ESP32-C5 数字系统中的三个看门狗定时器。

16.2.2.1 时钟源与 32 位计数器

看门狗定时器的核心是一个 32 位计数器。

以 MWDT0 为例：

- MWDT0 可通过设置 PCR_TIMERGROUP0_WDT_CLK_CONF_REG 寄存器的 PCR_TG0_WDT_CLK_SEL 字段选择 PLL_F80M_CLK 时钟、RC_FAST_CLK 或 XTAL_CLK （外部时钟）作为时钟源。
- 将 PCR_TIMERGROUP0_WDT_CLK_CONF_REG 寄存器的 PCR_TG0_WDT_CLK_EN 字段置 1 开启时钟，清零关闭时钟。时钟经由可配置的 16 位预分频器分频。详见章节 9 复位和时钟 的表 9.2-1。

MWDT 的 16 位预分频器系数可通过 TIMG_WDTCONFIG1_REG 寄存器的 TIMG_WDT_CLK_PRESCALE 字段配置。TIMG_WDT_DIVCNT_RST 字段置位时，预分频器复位，可立即重新配置。

RWDT 直接将 RTC 慢速时钟 RTC_SLOW_CLK（详见章节 9 复位和时钟）用作时钟源。

MWDT 和 RWDT 看门狗可分别通过设置 TIMG_WDT_EN 和 RTC_WDT_EN 字段使能。看门狗使能后，其内部 32 位计数器的值会在每个时钟源周期内累加 1，直到达到该阶段的超时时间（即在该阶段发生超时）。如发生超时，计数器的值会重置为 0，同时看门狗进入下一阶段。如果软件成功喂狗，看门狗定时器会回到阶段 0，并将计

数器的值重置为 0。软件向 TIMG_WDTFEED_REG 寄存器内写入任意值，便可为 MDWT 喂狗；向 RTC_WDT_FEED 写 1，便可为 RWDT 喂狗。

16.2.2.2 阶段与超时动作

定时器在各阶段可以配置不同的超时时间和对应的超时动作。某一阶段超时会触发对应的超时动作，同时计数器的值被重置为 0，看门狗进入下一阶段。

MWDT 和 RWDT 有四个阶段（称为阶段 0 至阶段 3）。看门狗定时器会循环工作（即从阶段 0 至阶段 3，再回到阶段 0）。

MWDT 每个阶段的超时时间可用 TIMG_WDTCONFIG*i*_REG（i 的范围是 2 到 5）寄存器配置，RWDT 的超时时间可用 RTC_WDT_STG*j*_HOLD（j 的范围是 0 到 3）字段配置。

值得注意的是，RWDT 在阶段 0 的超时时间 ($Thold_0$) 受 eFuse 寄存器 EFUSE_RD_REPEAT_DATA0_REG 的 EFUSE_WDT_DELAY_SEL 字段和 RTC_WDT_STG0_HOLD 字段共同影响，关系如下：

$$T_{hold0} = RTC\_WDT\_STG0\_HOLD << (EFUSE\_WDT\_DELAY\_SEL + 1)$$

其中，<< 为左移运算符。例如，RTC_WDT_STG0_HOLD 配置为 100，EFUSE_WDT_DELAY_SEL为 1，则 $Thold_0$ 为 400 个时钟周期。

如某个阶段超时，下列超时动作之一将会执行：

表 16.2-1. 超时动作

超时动作	描述
中断	触发中断
CPU 复位	详见章节 9 复位和时钟 > 小节 9.1.3 特性
内核复位	
系统复位	
关闭	对系统不产生影响

MWDT 所有阶段的超时动作均在 TIMG_WDTCONFIG0_REG 寄存器中配置。RWDT 的超时动作可在 RTC_WDT_CONFIG0_REG 寄存器配置。

16.2.2.3 写保护

看门狗定时器对于检测和处理系统或软件错误而言至关重要，不应轻易关闭（例如，因写寄存器位置错误而误将看门狗关闭）。因此，MWDT 和 RWDT 引入写保护机制，防止看门狗因无意的写操作而被关闭或篡改。

写保护机制通过每个看门狗定时器的写密钥字段运行（MWDT 看门狗使用 TIMG_WDT_WKEY，RWDT 看门狗使用 RTC_WDT_WKEY）。必须向看门狗定时器的写密钥字段写入 0x50D83AA1，才能修改其它看门狗寄存器。如果写密钥字段的值不是 0x50D83AA1，任何试图向看门狗定时器寄存器（除了向写密钥字段本身）写值的操作都会被忽略。推荐按以下步骤访问看门狗定时器：

1. 将 0x0D83AA1 写入看门狗定时器的写密钥字段，关闭写保护。
2. 根据需要修改看门狗，如喂狗或改变配置。
3. 向看门狗定时器的写密钥字段上写入除 0x50D83AA1 以外的任意值，重新使能写保护。

16.2.2.4 Flash 引导保护

在 flash 引导模式下，MWDT0 和 RWDT 会默认使能。MWDT0 的阶段 0 的默认超时动作为内核复位（复位主系统）。RWDT 的阶段 0 超时动作为系统复位（复位主系统和 RTC）。引导后，应将 TIMG_WDT_FLASHBOOT_MOD_EN 和 RTC_WDT_FLASHBOOT_MOD_EN 位清零，分别关闭 MWDT0 和 RWDT 的 flash 引导保护。然后，软件可以配置 MWDT0 和 RWDT。

16.3 超级看门狗定时器

超级看门狗 (SWD) 是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。SWD 包含一个看门狗电路，需在每个超时阶段（约不足一秒）至少喂狗一次。该电路会在看门狗超时时间约 100 ms 之前发送 WD_INTR 信号提醒系统喂狗。

如果系统不回应 SWD 的喂狗请求，看门狗超时，SWD 会产生系统电平信号 SWD_RSTB，复位芯片上的整个数字电路（即超时动作中的系统复位）。

SWD 的时钟源固定，不可选择。

16.3.1 主要特性

SWD 具有如下特性：

- 独立于数字电路的模拟看门狗，可以工作数字电路时钟和电压异常的环境
- 用中断提醒 SWD 即将超时
- 软件有多种专用的方法喂 SWD，让 SWD 监控整个操作系统的工作状态

16.3.2 SWD 控制器

16.3.2.1 结构

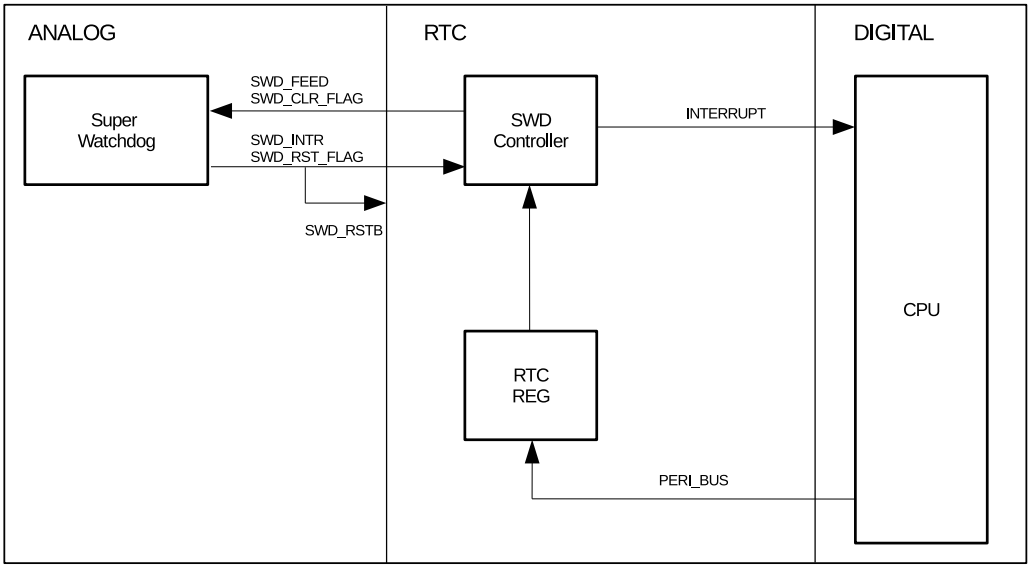


图 16.3-1. SWD 控制器结构

16.3.2.2 工作流程

正常状态下：

- SWD 控制器收到 SWD 的喂狗请求。
- SWD 控制器向 HP CPU 发送中断。
- HP CPU 通过置位 RTC_WDT_SWD_FEED 直接喂狗。
- CPU 喂狗时，需要先向 RTC_WDT_SWD_WKEY 写 0x50D83AA1 关闭 SWD 控制器的写保护。这样做可以防止系统在数字电路异常状态下运行时误喂 SWD。
- 如将 RTC_WDT_SWD_AUTO_FEED_EN 置 1，SWD 控制器也可配置为在不需要 CPU 干预的情况下接收 SWD 发出的中断喂 SWD。

复位后：

- 可查看 LP_CLKRST_RESET_CAUSE[4:0] 获知 CPU 复位原因。
  如 LP_CLKRST_RESET_CAUSE[4:0] == 0x12，则表示上一次复位的原因是 SWD 复位。
- 置位 RTC_WDT_SWD_RST_FLAG_CLR 清除 SWD 复位标志。

16.4 中断

看门狗定时器中断，请前往章节 15 定时器组 *(TIMG)* 的第 15.6 节 中断 查看。

16.5 寄存器列表

本小节的地址均为相对于低功耗看门狗定时器 (RTC_WDT) 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

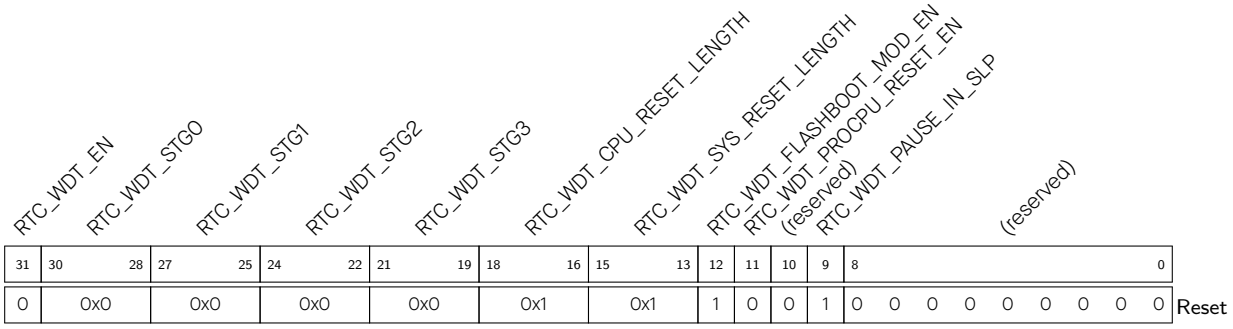
名称	描述	地址	访问
配置寄存器			
RTC_WDT_CONFIG0_REG	RWDT 配置寄存器	0x0000	R/W
RTC_WDT_CONFIG1_REG	配置 RWDT 阶段 0 的超时时间	0x0004	R/W
RTC_WDT_CONFIG2_REG	配置 RWDT 阶段 1 的超时时间	0x0008	R/W
RTC_WDT_CONFIG3_REG	配置 RWDT 阶段 2 的超时时间	0x000C	R/W
RTC_WDT_CONFIG4_REG	配置 RWDT 阶段 2 的超时时间	0x0010	R/W
RTC_WDT_FEED_REG	配置喂狗 RWDT	0x0014	WT
RTC_WDT_WPROTECT_REG	配置锁住 RWDT 的配置寄存器	0x0018	R/W
RTC_WDT_SWD_CONFIG_REG	SWD 配置寄存器	0x001C	varies
RTC_WDT_SWD_WPROTECT_REG	配置锁住 SWD 的配置寄存器	0x0020	R/W
RTC_WDT_INT_RAW_REG	配置是否产生 WDT 超时中断	0x0024	R/WTC/SS
RTC_WDT_INT_ST_REG	WDT 中断状态寄存器	0x0028	RO
RTC_WDT_INT_ENA_REG	WDT 中断使能寄存器	0x002C	R/W

名称	描述	地址	访问
RTC_WDT_INT_CLR_REG	WDT 中断清除寄存器	0x0030	WT
RTC_WDT_DATE_REG	版本控制寄存器	0x03FC	R/W

16.6 寄存器

MWDT 寄存器是定时器组模块的一部分，在章节 15 定时器组 *(TIMG)* 的第 15.8 节 寄存器列表 中有详细描述。本小节包含 RWDT 和 SWD 相关寄存器介绍，其地址均为相对于低功耗看门狗定时器 (RTC_WDT) 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 16.1. RTC_WDT_CONFIG0_REG (0x0000)



RTC_WDT_PAUSE_IN_SLP 配置芯片处于 light sleep 或 deep sleep 模式时是否禁用 RWDT。

0：使能

1：禁用

(R/W)

RTC_WDT_PROCPU_RESET_EN 配置是否使能 RWDT 进行 CPU 复位。

0：禁用

1：使能

(R/W)

RTC_WDT_FLASHBOOT_MOD_EN 配置在 SPI boot 模式下是否自动使能 RWDT。

0：禁用

1：使能

(R/W)

RTC_WDT_SYS_RESET_LENGTH 配置内核复位时间。

单位：LP_DYN_FAST_CLK

(R/W)

RTC_WDT_CPU_RESET_LENGTH 配置 CPU 复位时间。

单位：LP_DYN_FAST_CLK

(R/W)

RTC_WDT_STG3 配置阶段 3 的超时复位动作。

0：无效

1：中断

2：CPU 复位

3：内核复位

4：系统复位

(R/W)

见下页...

Register 16.1. RTC_WDT_CONFIG0_REG (0x0000)

接上页...

RTC_WDT_STG2 配置阶段 2 的超时复位动作。

0：无效
1：中断
2：CPU 复位
3：内核复位
4：系统复位
(R/W)

RTC_WDT_STG1 配置阶段 1 的超时复位动作。

0：无效
1：中断
2：CPU 复位
3：内核复位
4：系统复位
(R/W)

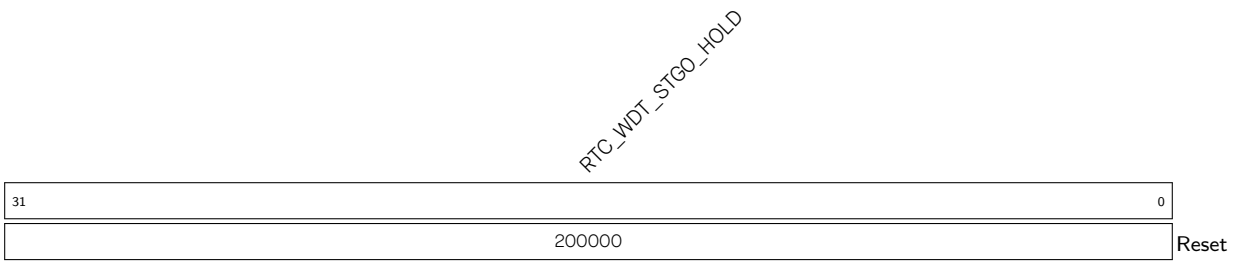
RTC_WDT_STG0 配置阶段 0 的超时复位动作。

0：无效
1：中断
2：CPU 复位
3：内核复位
4：系统复位
(R/W)

RTC_WDT_EN 配置是否使能 RWDT。

0：禁用 RWDT
1：使能 RWDT
(R/W)

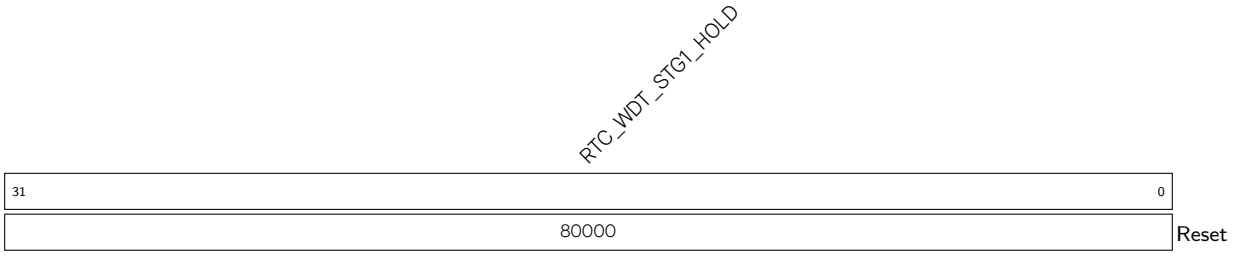
Register 16.2. RTC_WDT_CONFIG1_REG (0x0004)



RTC_WDT_STG0_HOLD 配置阶段 0 的超时时间。

单位：LP_DYN_SLOW_CLK
(R/W)

Register 16.3. RTC_WDT_CONFIG2_REG (0x0008)

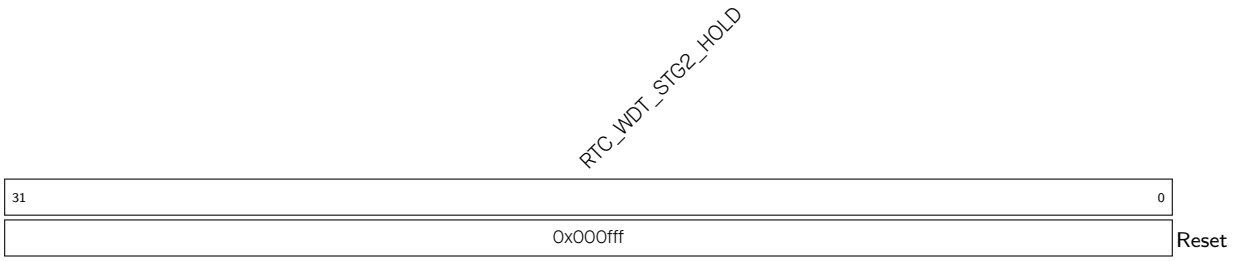


RTC_WDT_STG1_HOLD 配置阶段 1 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 16.4. RTC_WDT_CONFIG3_REG (0x000C)

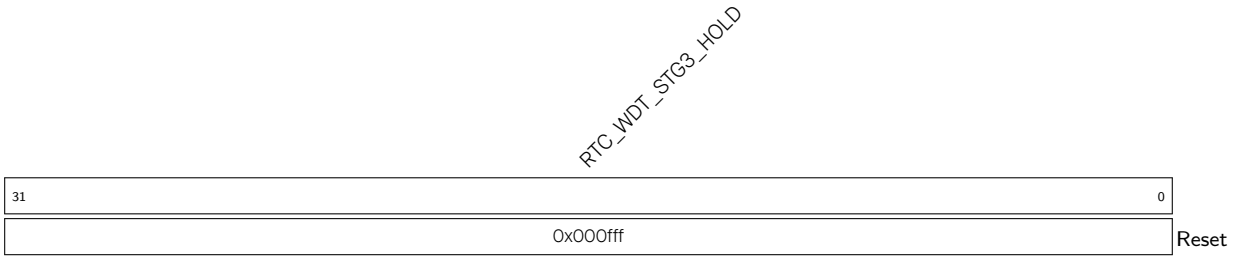


RTC_WDT_STG2_HOLD 配置阶段 2 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 16.5. RTC_WDT_CONFIG4_REG (0x0010)

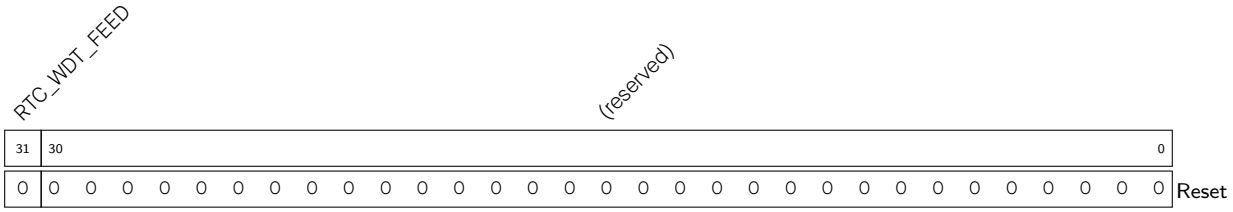


RTC_WDT_STG3_HOLD 配置阶段 3 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 16.6. RTC_WDT_FEED_REG (0x0014)



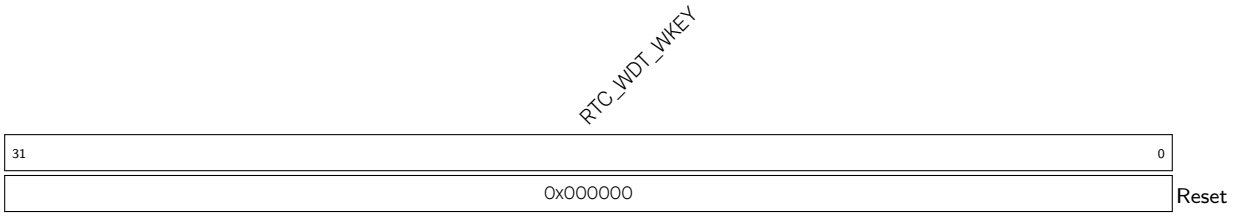
RTC_WDT_FEED 配置此位喂狗 RWDT。

0：无效

1：喂狗

(WT)

Register 16.7. RTC_WDT_WPROTECT_REG (0x0018)



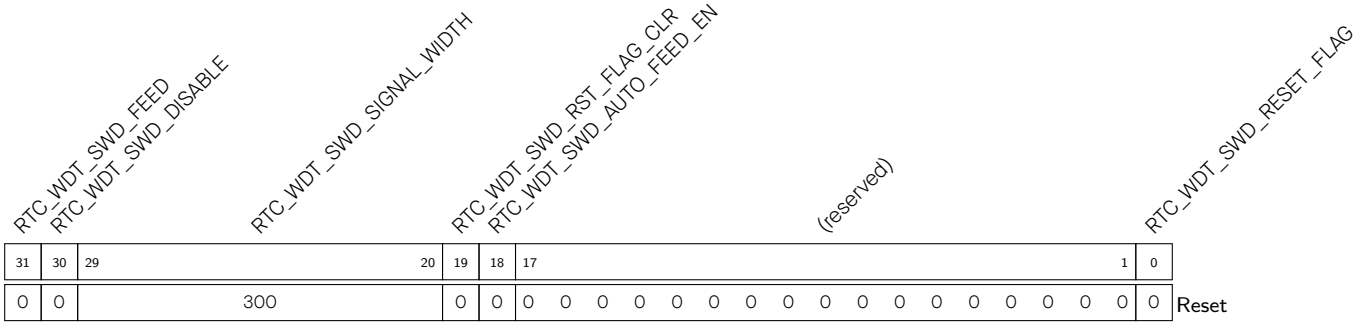
RTC_WDT_WKEY 配置该寄存器来锁住或解锁 RWDT 的配置寄存器。

0x50D83AA1: 解锁 RWDT 的配置寄存器

Others value: 锁住 RWDT 的配置寄存器，软件不能修改 RWDT 的配置

(R/W)

Register 16.8. RTC_WDT_SWD_CONFIG_REG (0x001C)



RTC_WDT_SWD_RESET_FLAG 表示 SWD 是否产生复位信号。

0：否

1：是

(RO)

RTC_WDT_SWD_AUTO_FEED_EN 配置此位使能硬件自动喂狗。

0：禁止硬件自动喂狗

1：使能硬件自动喂狗

(R/W)

RTC_WDT_SWD_RST_FLAG_CLR 配置此位清除 SWD 复位标志。

0：无效

1：清除复位标志

(WT)

RTC_WDT_SWD_SIGNAL_WIDTH 配置输出到模拟电路的信号长度。

单位：LP_DYN_FAST_CLK (R/W)

RTC_WDT_SWD_DISABLE 配置此位禁用 SWD。

0：使能

1：禁用

(R/W)

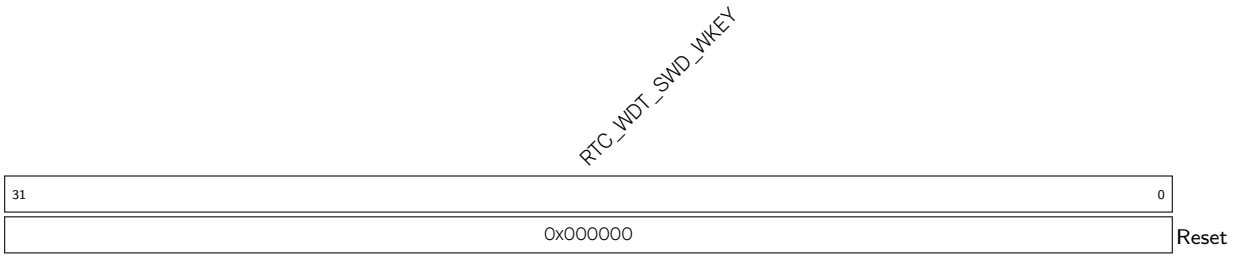
RTC_WDT_SWD_FEED 配置此位喂狗 SWD。

0：无效

1: 喂狗 SWD

(WT)

Register 16.9. RTC_WDT_SWD_WPROTECT_REG (0x0020)



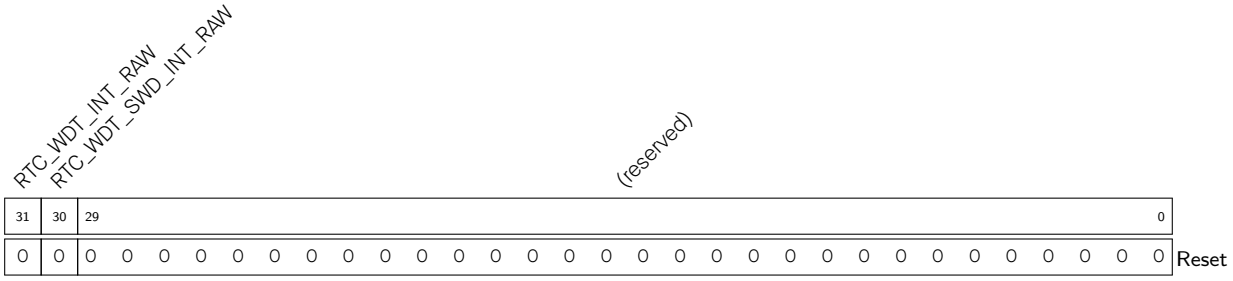
RTC_WDT_SWD_WKEY 配置该寄存器来锁住或解锁 SWDT 的配置寄存器。

0x50D83AA1: 解锁 SWD 的配置寄存器

Others value: 锁住 SWD 的配置寄存器，软件不能修改 SWD 的配置

(R/W)

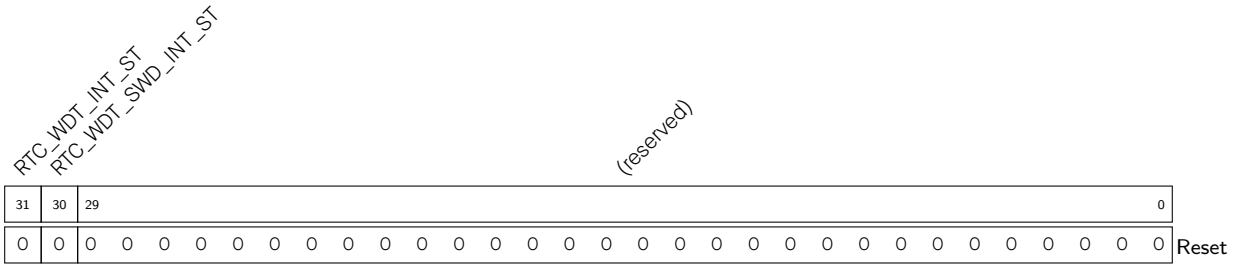
Register 16.10. RTC_WDT_INT_RAW_REG (0x0024)



RTC_WDT_SWD_INT_RAW RTC_WDT_SWD_INT 的原始中断状态。(R/WTC/SS)

RTC_WDT_INT_RAW RTC_WDT_INT 的原始中断状态。(R/WTC/SS)

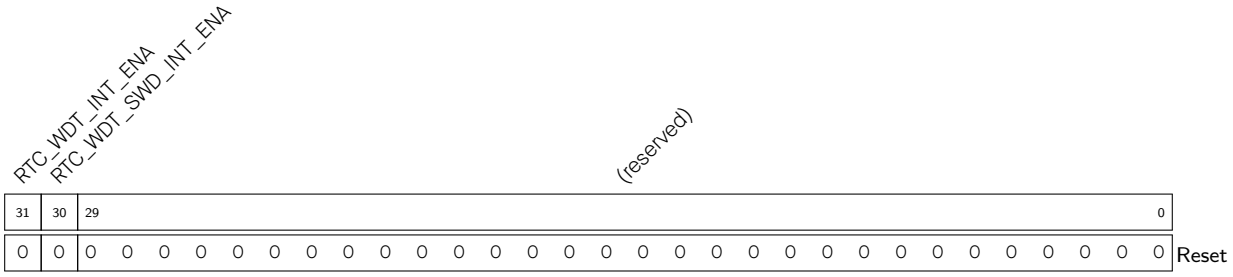
Register 16.11. RTC_WDT_INT_ST_REG (0x0028)



RTC_WDT_SWD_INT_ST RTC_WDT_SWD_INT 的屏蔽中断状态。(RO)

RTC_WDT_INT_ST RTC_WDT_INT 的屏蔽中断状态。(RO)

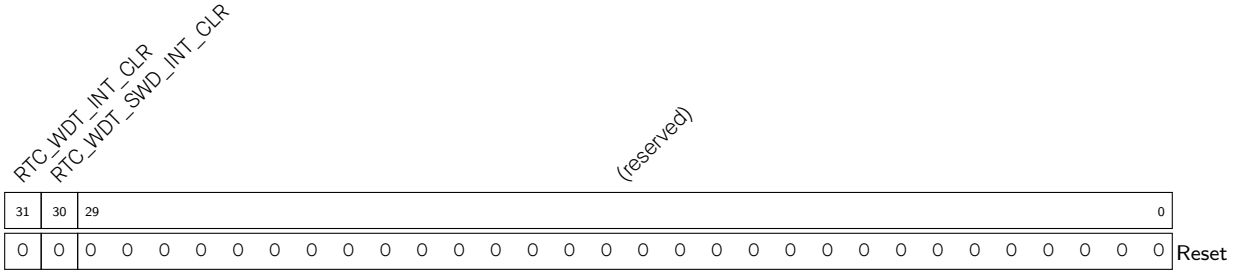
Register 16.12. RTC_WDT_INT_ENA_REG (0x002C)



RTC_WDT_SWD_INT_ENA 写 1 使能 RTC_WDT_SWD_INT. (R/W)

RTC_WDT_INT_ENA 写 1 使能 RTC_WDT_INT. (R/W)

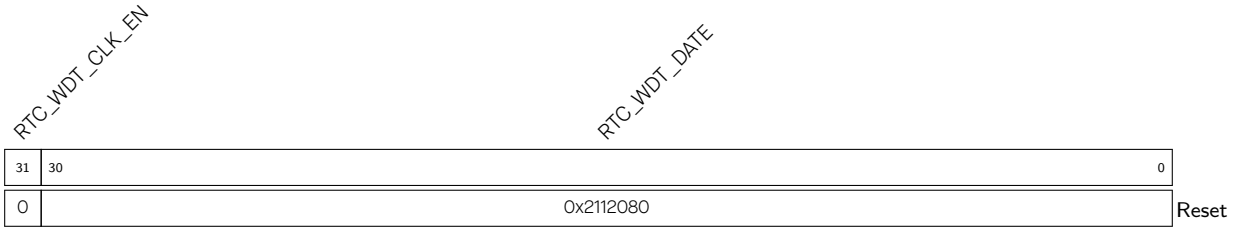
Register 16.13. RTC_WDT_INT_CLR_REG (0x0030)



RTC_WDT_SWD_INT_CLR 写 1 清除 RTC_WDT_SWD_INT. (WT)

RTC_WDT_INT_CLR 写 1 清除 RTC_WDT_INT. (WT)

Register 16.14. RTC_WDT_DATE_REG (0x03FC)



RTC_WDT_DATE 版本控制寄存器 (R/W)

RTC_WDT_CLK_EN 保留 (R/W)

第 17 章

实时时钟定时器

17.1 概述

实时时钟定时器（RTC 定时器）是一个 48 位的可读计时器，能够在芯片的任意功耗模式下工作，当 HP 系统中的定时器外设不可用时，继续为系统提供定时器服务。同时还支持配置定时器中断、记录系统中特定事件发生的时刻。

17.2 主要特性

- 48 位的计时器
- 触发特定事件时可记录事件发生的时刻，支持的特定事件有：
  - HP 系统复位
  - CPU 进入 stall 状态
  - CPU 退出 stall 状态
  - 晶振时钟开启
  - 晶振时钟关闭
- 通过配置寄存器触发 RTC 定时器记录当前时间
- 支持缓存最近两次特定事件的发生的时间
- 支持在目标时刻产生中断，目标时刻可配置，可同时配置两个目标时刻
- 除 LP 系统上电复位外的其余任何复位/睡眠均不会使 RTC 定时器停止或复位
- 可用作唤醒源（详见章节 13 低功耗管理）

17.3 功能描述

- 48 位计时器
  - RTC 定时器的实现基于一个 48 位的计时器，其驱动时钟是由位于 Always-On 电源域的 RTC_SLOW_CLK 提供，采用不间断的循环计时（LP 系统复位除外），当 48 位计时器溢出时会产生溢出中断。关于电源域的更多信息，请查看章节 13 低功耗管理)。
- 记录特定事件的发生时刻
  - RTC 定时器支持记录三类特定事件的发生时刻：
    * HP 系统复位

  * CPU 进入或退出 stall 状态
  * 晶振时钟开启或关闭

- 三类事件都有各自独立的配置寄存器，当配置寄存器使能后，若发生对应的事件，产生的硬件脉冲就会触发 RTC 定时器记录当前时间。对应的配置寄存器见下表：

表 17.3-1. 配置 RTC 定时器记录特定事件的发生时刻

配置寄存器	描述
RTC_TIMER_MAIN_TIMER_SYS_RST	使能 RTC 定时器记录系统复位时刻。
RTC_TIMER_MAIN_TIMER_SYS_STALL	使能 RTC 定时器记录 CPU 进入或退出 stall 状态时刻。
RTC_TIMER_MAIN_TIMER_XTAL_OFF	使能 RTC 定时器记录晶振时钟开启或关闭时刻。

- 通过配置寄存器触发 RTC 定时器记录当前时间
  - 除了上述三类特定事件发生时可以触发 RTC 定时器记录当前时间，还可以通过配置寄存器 RTC_TIMER_MAIN_TIMER_UPDATE 触发 RTC 定时器记录当前时间。
  - 配置 RTC_TIMER_MAIN_TIMER_UPDATE 寄存器会产生一个脉冲信号，触发 RTC 定时器记录当前时间。
- 目标时刻产生计数中断
  - RTC 定时器支持同时配置两个目标时刻，分别为目标时刻 0 和目标时刻 1，当计时器到达目标时刻 0 时会触发 RTC_TIMER_CMP0_INT 中断，当计时器到达目标时刻 1 时会触发 RTC_TIMER_CMP1_INT 中断。
  - 目标时刻的配置寄存器为：

表 17.3-2. RTC 定时器目标时刻配置

配置寄存器	描述
RTC_TIMER_MAIN_TIMER_TAR_EN*n*	*n* = 0 或 1，配置目标时刻使能。
RTC_TIMER_MAIN_TIMER_TAR_LOW*n*	*n* = 0 或 1，配置目标时刻的低 32 位。
RTC_TIMER_MAIN_TIMER_TAR_HIGH*n*	*n* = 0 或 1，配置目标时刻的高 16 位。

- 读取缓存时刻
  - RTC 定时器有两组寄存器用于缓存特定事件的发生时刻，分别为寄存器组 0 和寄存器组 1；
  - 寄存器组 0 包括寄存器 RTC_TIMER_MAIN_BUF0_LOW_REG 和 RTC_TIMER_MAIN_BUF0_HIGH_REG，用于记录 RTC 定时器在当前触发下的计数值；
  - 寄存器组 1 包括寄存器 RTC_TIMER_MAIN_BUF1_LOW_REG 和 RTC_TIMER_MAIN_BUF1_HIGH_REG，用于记录 RTC 定时器在上一次触发下的计数值；
  - 每次有新的触发，上一次触发时的记录将从寄存器组 0 移至寄存器组 1（寄存器组 1 中之前的记录将被覆盖），而本次触发的记录将存储在寄存器组 0。因此，RTC 定时器最多可同时记录两次触发的值。

17.4 事件任务矩阵功能

在 ESP32-C5 中，RTC 定时器支持 ETM 功能，即可以通过 RTC 定时器的 ETM 事件触发任意外设的 ETM 任务。RTC 定时器不支持 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与 RTC 定时器相关的 ETM 事件。

RTC 定时器可产生的 ETM 事件有：

- RTC_EVT_CMP：表示 RTC 定时器计时到达寄存器 RTC_TIMER_MAIN_TIMER_TAR_LOW0 和 RTC_TIMER_MAIN_TIMER_TAR_HIGH0 所配置的目标时刻 0。
- RTC_EVT_OVF：表示 48 位计时器计数溢出。
- RTC_EVT_TICK：表示 RTC 计时器递增 1 时触发的事件。

17.5 中断

ESP32-C5 的 RTC 定时器可以生成以下中断信号并发送给中断矩阵。

- LP_TIMER_REG_0_INTR（该中断供 HP CPU 使用）
- LP_TIMER_REG_1_INTR（该中断供 LP CPU 使用）

这些中断信号由 RTC 定时器的内部中断源生成。表 17.5-1 列出了 RTC 定时器的内部中断源、中断触发条件、以及生成的中断信号。

表 17.5-1. RTC 定时器的内部中断源

内部中断源	触发条件	中断信号
RTC_TIMER_CMP0_INT	RTC 定时器计时到达目标时刻 0	LP_TIMER_REG_0_INTR
RTC_TIMER_CMP1_INT	RTC 定时器计时到达目标时刻 1	LP_TIMER_REG_1_INTR
RTC_TIMER_OVERFLOW_INT	48 位计时器计数溢出	LP_TIMER_REG_0_INTR LP_TIMER_REG_1_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 17.6 寄存器列表。

17.6 寄存器列表

本小节的所有地址均为相对于 RTC 定时器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

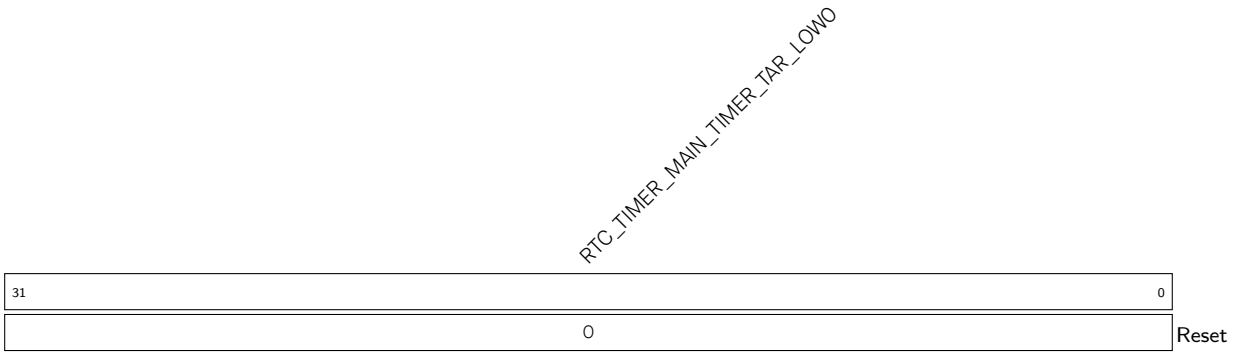
名称	描述	地址	访问
配置寄存器			
RTC_TIMER_TARO_LOW_REG	配置目标时刻 0 的低 32 位	0x0000	R/W
RTC_TIMER_TARO_HIGH_REG	配置目标时刻 0 的高 16 位	0x0004	varies
RTC_TIMER_TAR1_LOW_REG	配置目标时刻 1 的低 32 位	0x0008	R/W
RTC_TIMER_TAR1_HIGH_REG	配置目标时刻 1 的高 16 位	0x000C	varies
RTC_TIMER_UPDATE_REG	配置使能 RTC 定时器锁存当前值	0x0010	varies
RTC_TIMER_MAIN_BUFO_LOW_REG	RTC 定时器计数器缓存值 0 的低 32 位	0x0014	RO
RTC_TIMER_MAIN_BUFO_HIGH_REG	RTC 定时器计数器缓存值 0 的高 16 位	0x0018	RO
RTC_TIMER_MAIN_BUF1_LOW_REG	RTC 定时器计数器缓存值 1 的低 32 位	0x001C	RO
RTC_TIMER_MAIN_BUF1_HIGH_REG	RTC 定时器计数器缓存值 1 的高 16 位	0x0020	RO
中断寄存器			
RTC_TIMER_INT_RAW_REG	计时器到达目标时刻 0 的原始中断状态寄存器	0x0028	R/WTC/SS
RTC_TIMER_INT_ST_REG	计时器到达目标时刻 0 的中断状态寄存器	0x002C	RO
RTC_TIMER_INT_ENA_REG	计时器到达目标时刻 0 的中断使能寄存器	0x0030	R/W
RTC_TIMER_INT_CLR_REG	计时器到达目标时刻 0 的中断清除寄存器	0x0034	WT
RTC_TIMER_LP_INT_RAW_REG	计时器到达目标时刻 1 的原始中断状态寄存器	0x0038	R/WTC/SS
RTC_TIMER_LP_INT_ST_REG	计时器到达目标时刻 1 的中断状态寄存器	0x003C	RO
RTC_TIMER_LP_INT_ENA_REG	计时器到达目标时刻 1 的中断使能寄存器	0x0040	R/W
RTC_TIMER_LP_INT_CLR_REG	计时器到达目标时刻 1 的中断清除寄存器	0x0044	WT
版本寄存器			
RTC_TIMER_DATE_REG	版本控制寄存器	0x03FC	R/W

17.7 寄存器

本小节的所有地址均为相对于 RTC 定时器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

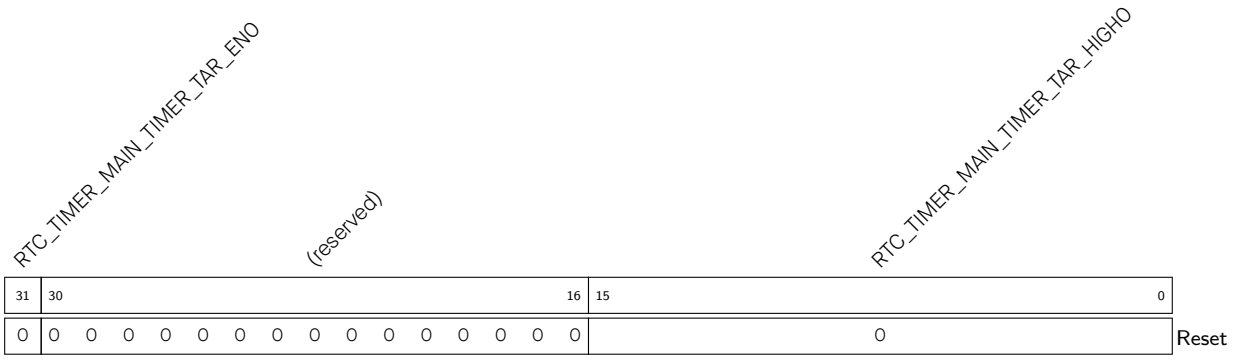
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 17.1. RTC_TIMER_TAR0_LOW_REG (0x0000)



RTC_TIMER_MAIN_TIMER_TAR_LOW0 配置计时器的目标时刻 0 的低 32 位。(R/W)

Register 17.2. RTC_TIMER_TAR0_HIGH_REG (0x0004)



RTC_TIMER_MAIN_TIMER_TAR_HIGH0 配置计时器的目标时刻 0 的高 16 位。(R/W)

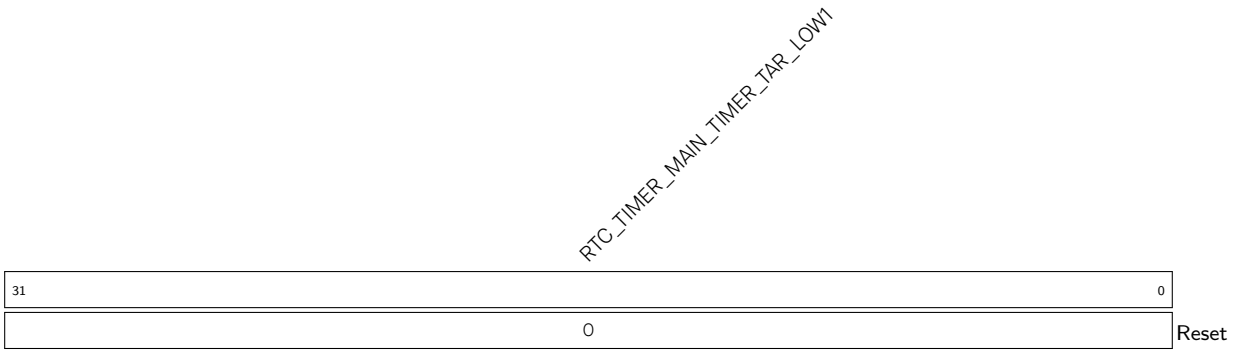
RTC_TIMER_MAIN_TIMER_TAR_EN0 配置是否使能计时器的目标时刻 0。

0：不使能

1：使能

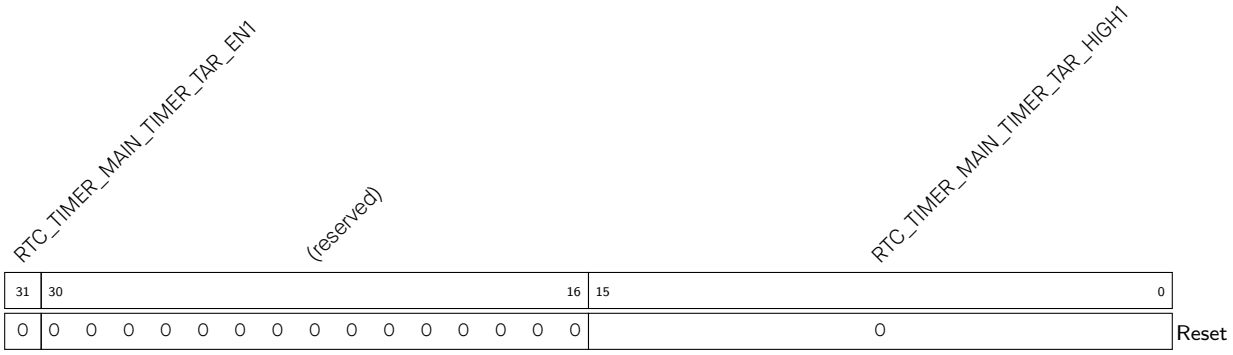
(WT)

Register 17.3. RTC_TIMER_TAR1_LOW_REG (0x0008)



RTC_TIMER_MAIN_TIMER_TAR_LOW1 配置计时器的目标时刻 1 的低 32 位。(R/W)

Register 17.4. RTC_TIMER_TAR1_HIGH_REG (0x000C)



RTC_TIMER_MAIN_TIMER_TAR_HIGH1 配置计时器的目标时刻 1 的高 16 位。(R/W)

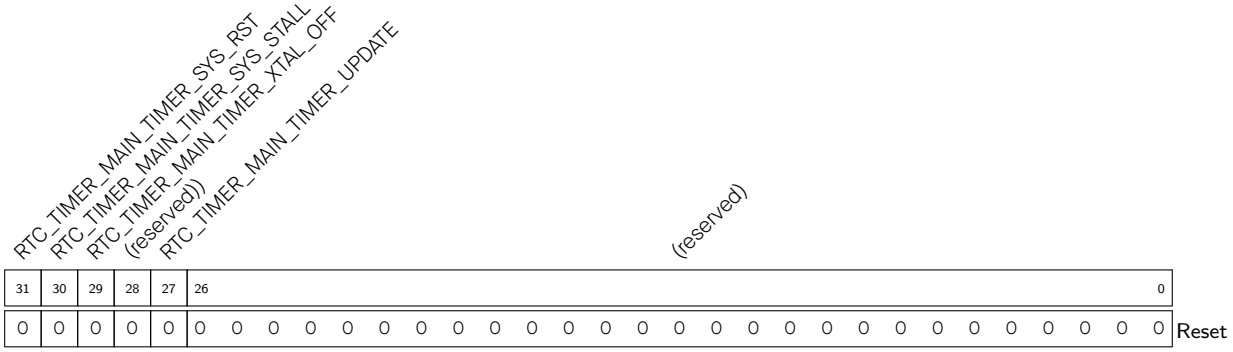
RTC_TIMER_MAIN_TIMER_TAR_EN1 计时器的目标时刻 1 使能。

0：不使能

1：使能

(WT)

Register 17.5. RTC_TIMER_UPDATE_REG (0x0010)



RTC_TIMER_MAIN_TIMER_UPDATE 配置是否通过软件配置记录当前时间。

0：不记录

1：记录

(WT)

RTC_TIMER_MAIN_TIMER_XTAL_OFF 配置是否使能 RTC 定时器记录晶振时钟开启或关闭时刻。

0：不使能

1：使能

(R/W)

RTC_TIMER_MAIN_TIMER_SYS_STALL 配置是否使能 RTC 定时器记录 CPU 进入或退出 stall 状态时刻。

0：不使能

1：使能

(R/W)

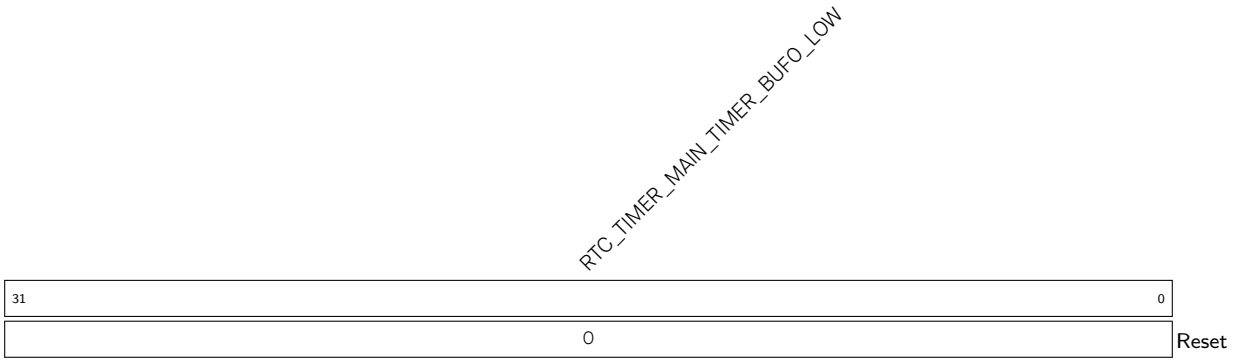
RTC_TIMER_MAIN_TIMER_SYS_RST 配置是否使能 RTC 定时器记录系统复位时刻。

0：不使能

1：使能

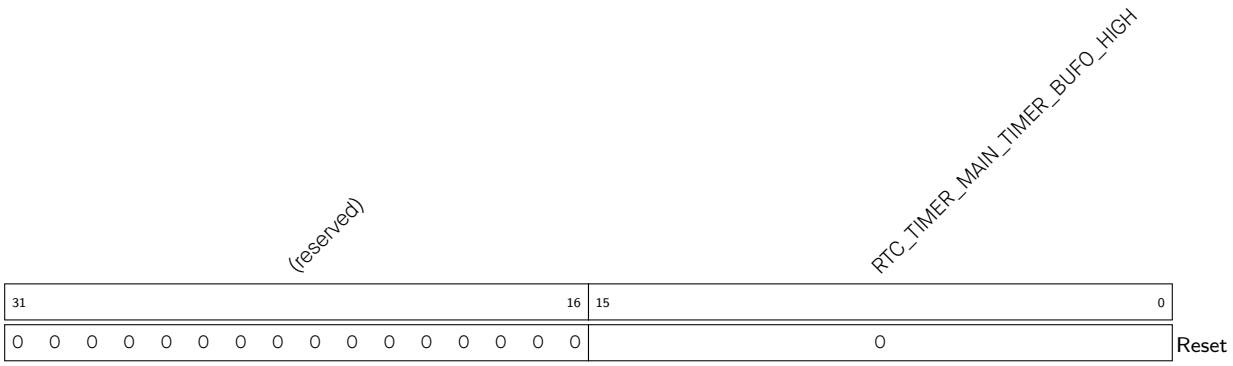
(R/W)

Register 17.6. RTC_TIMER_MAIN_BUF0_LOW_REG (0x0014)



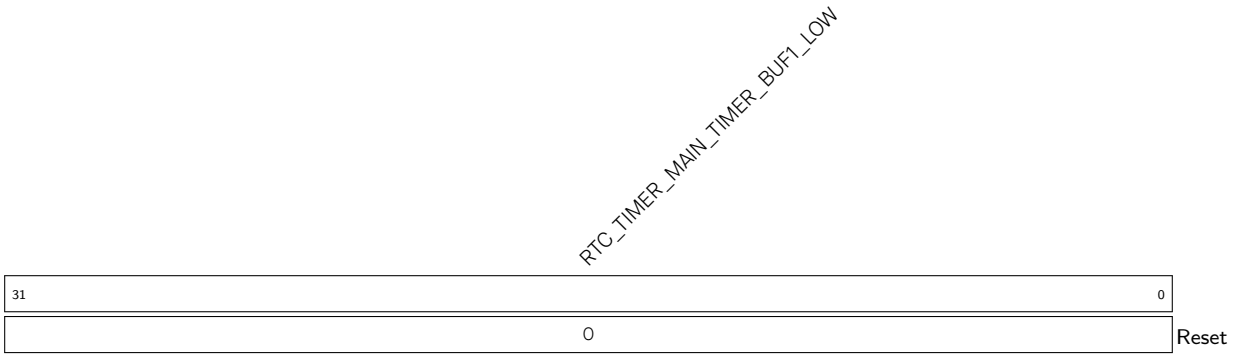
RTC_TIMER_MAIN_TIMER_BUF0_LOW 表示计时器缓存值 0 的低 32 位。(RO)

Register 17.7. RTC_TIMER_MAIN_BUF0_HIGH_REG (0x0018)



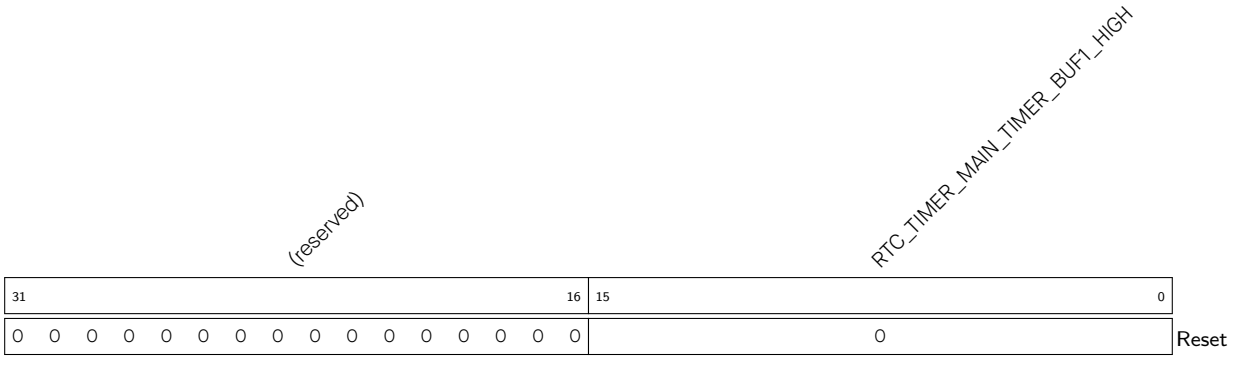
RTC_TIMER_MAIN_TIMER_BUF0_HIGH 表示计时器缓存值 0 的高 16 位。(RO)

Register 17.8. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



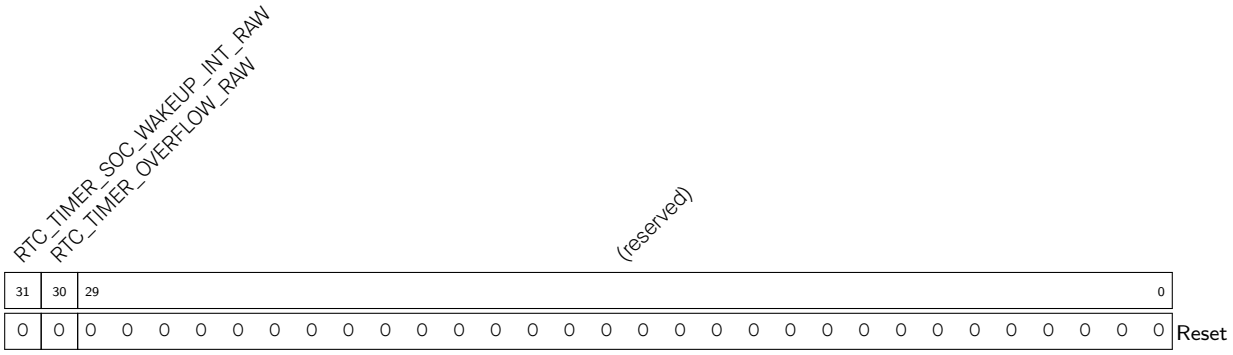
RTC_TIMER_MAIN_TIMER_BUF1_LOW 表示计时器缓存值 1 的低 32 位。(RO)

Register 17.9. RTC_TIMER_MAIN_BUF1_HIGH_REG (0x0020)



RTC_TIMER_MAIN_TIMER_BUF1_HIGH 表示计时器缓存值 1 的高 16 位。(RO)

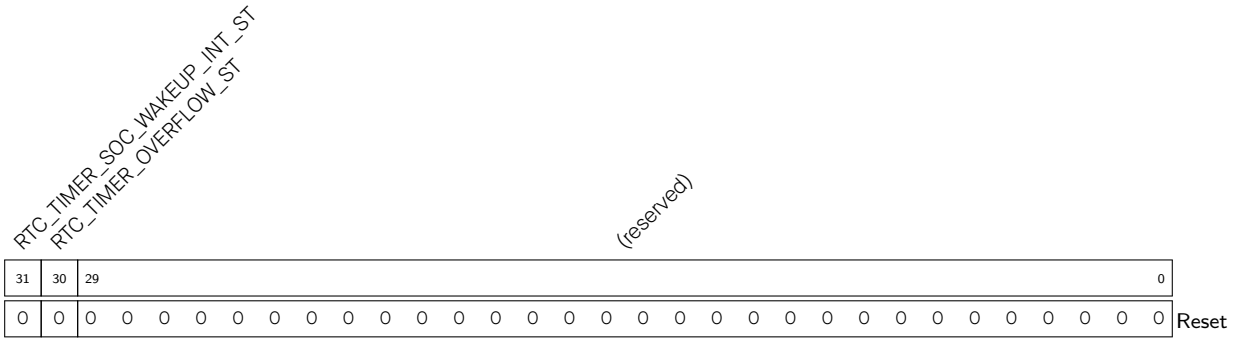
Register 17.10. RTC_TIMER_INT_RAW_REG (0x0028)



RTC_TIMER_OVERFLOW_RAW RTC_TIMER_OVERFLOW_INT (HP CPU) 的原始中断状态。(R/WTC/SS)

RTC_TIMER_SOC_WAKEUP_INT_RAW RTC_TIMER_CMP0_INT 原始中断状态。(R/WTC/SS)

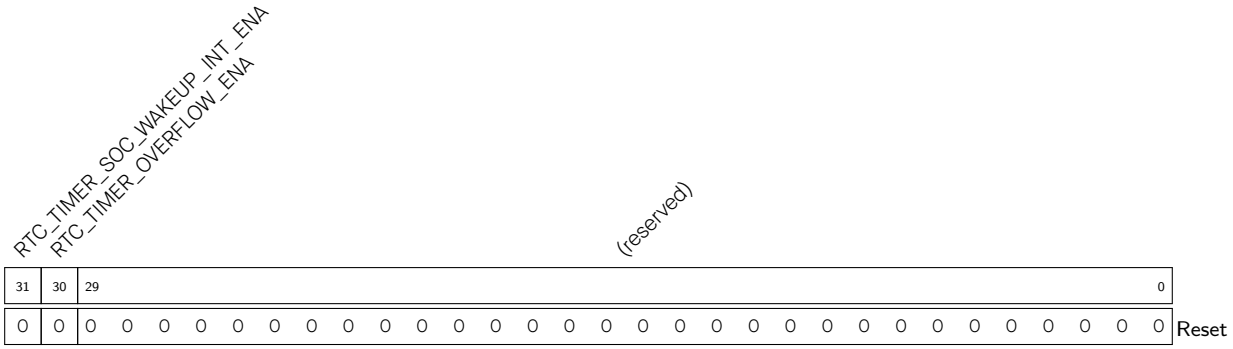
Register 17.11. RTC_TIMER_INT_ST_REG (0x002C)



RTC_TIMER_OVERFLOW_ST RTC_TIMER_OVERFLOW_INT (HP CPU) 的屏蔽中断状态。(RO)

RTC_TIMER_SOC_WAKEUP_INT_ST RTC_TIMER_CMP0_INT 的屏蔽中断状态。(RO)

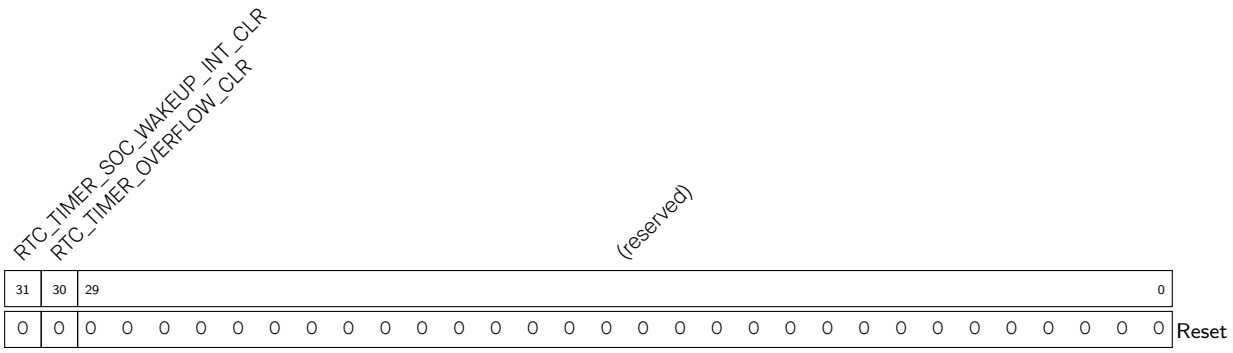
Register 17.12. RTC_TIMER_INT_ENA_REG (0x0030)



RTC_TIMER_OVERFLOW_ENA 写 1 使能 RTC_TIMER_OVERFLOW_INT (HP CPU) 中断。(R/W)

RTC_TIMER_SOC_WAKEUP_INT_ENA 写 1 使能 RTC_TIMER_CMP0_INT 中断。(R/W)

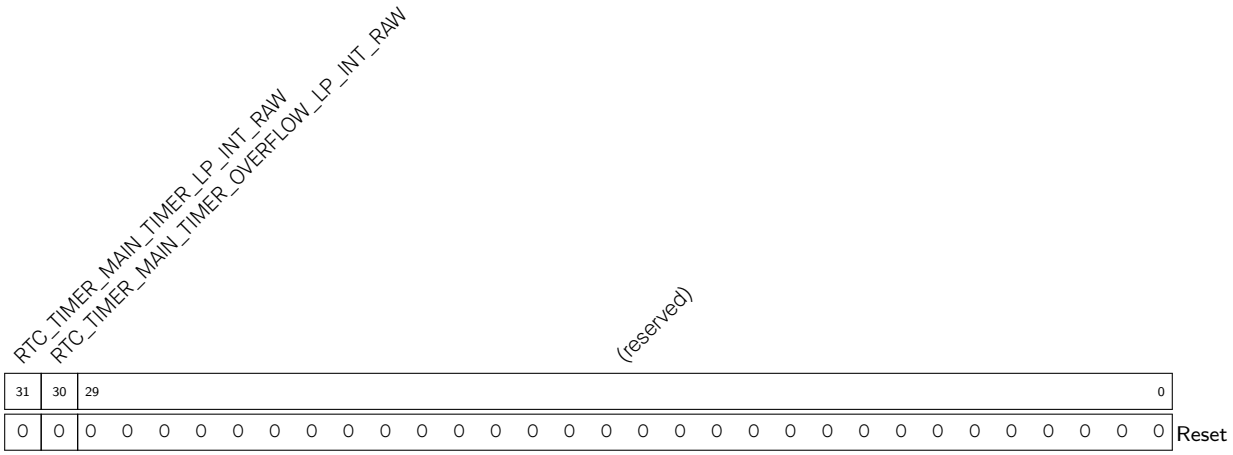
Register 17.13. RTC_TIMER_INT_CLR_REG (0x0034)



RTC_TIMER_OVERFLOW_CLR 写 1 清除 RTC_TIMER_OVERFLOW_INT (HP CPU) 中断。(WT)

RTC_TIMER_SOC_WAKEUP_INT_CLR 写 1 清除 RTC_TIMER_CMP0_INT 中断。(WT)

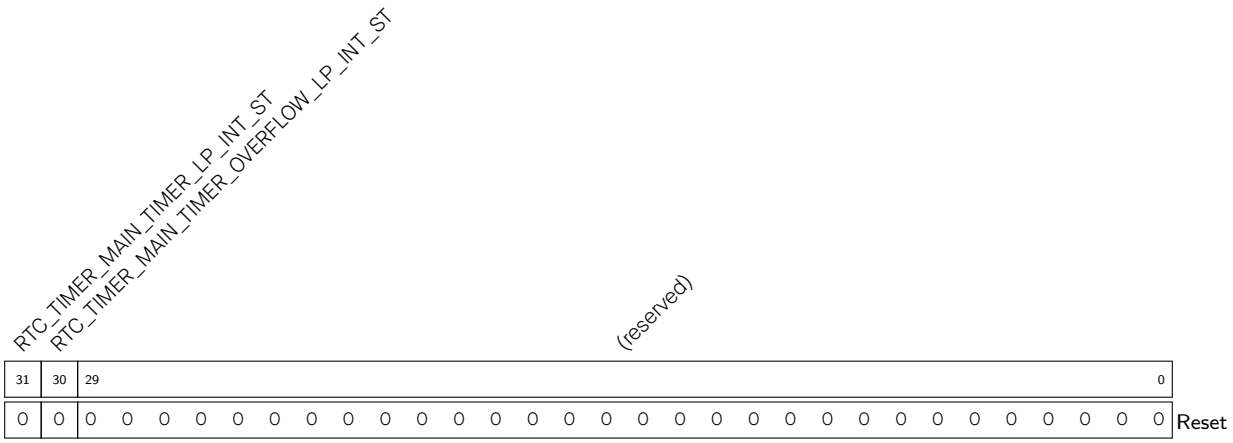
Register 17.14. RTC_TIMER_LP_INT_RAW_REG (0x0038)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_RAW RTC_TIMER_OVERFLOW_INT (LP CPU) 的原始中断状态。(R/WTC/SS)

RTC_TIMER_MAIN_TIMER_LP_INT_RAW RTC_TIMER_CMP1_INT 原始中断状态。(R/WTC/SS)

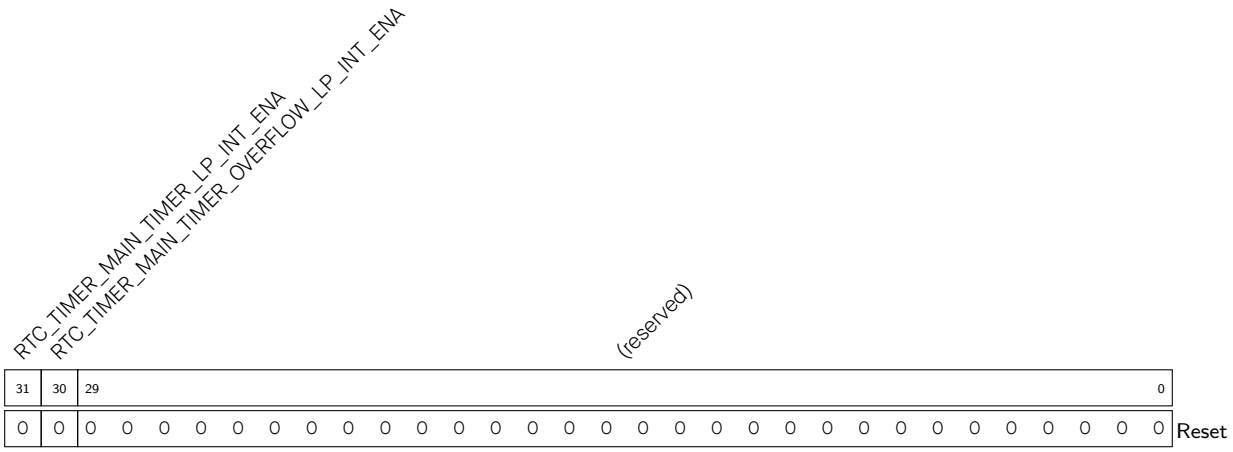
Register 17.15. RTC_TIMER_LP_INT_ST_REG (0x003C)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ST RTC_TIMER_OVERFLOW_INT (LP CPU) 的屏蔽状态中断。(RO)

RTC_TIMER_MAIN_TIMER_LP_INT_ST RTC_TIMER_CMP1_INT 屏蔽中断状态。(RO)

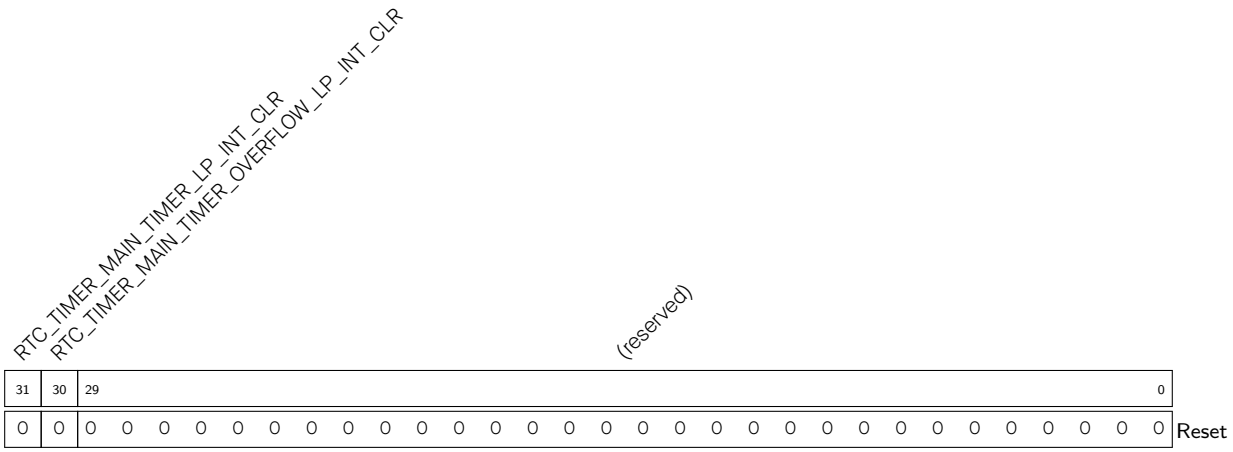
Register 17.16. RTC_TIMER_LP_INT_ENA_REG (0x0040)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ENA 写 1 使能 RTC_TIMER_OVERFLOW_INT (LP CPU) 中断。(R/W)

RTC_TIMER_MAIN_TIMER_LP_INT_ENA 写 1 使能 RTC_TIMER_CMP1_INT 中断。(R/W)

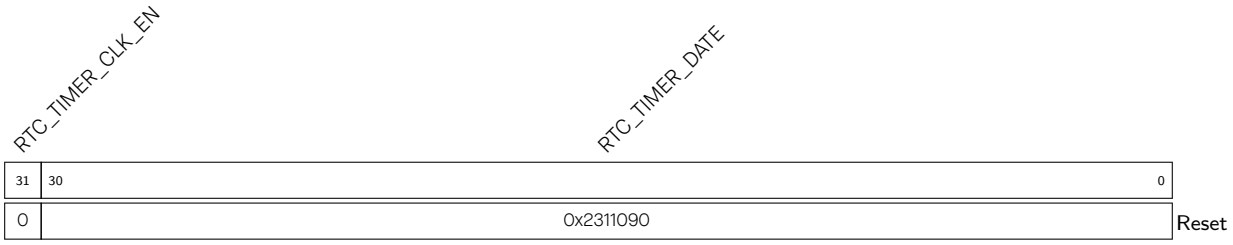
Register 17.17. RTC_TIMER_LP_INT_CLR_REG (0x0044)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_CLR 写 1 清除 RTC_TIMER_OVERFLOW_INT (LP CPU) 中断。(WT)

RTC_TIMER_MAIN_TIMER_LP_INT_CLR 写 1 清除 RTC_TIMER_CMP1_INT 中断。(WT)

Register 17.18. RTC_TIMER_DATE_REG (0x03FC)



RTC_TIMER_DATE 版本控制寄存器。(R/W)

RTC_TIMER_CLK_EN 配置 RTC 定时器时钟门控。

0：仅在应用程序写入寄存器时支持时钟。

1：始终强制打开寄存器的时钟。

(R/W)

第 18 章

权限控制 (PMS)

18.1 概述

ESP32-C5 整个系统的权限控制可以分为两部分：PMP（Physical Memory Protection，物理存储器保护）和 APM（Access Permission Management，访问权限管理）。

PMP 位于 HP CPU 内部，能够管控 HP CPU 访问所有的地址空间。APM 则是一个作用于总线端口处的访问权限管理模块，可以与 PMP 一起管理 HP CPU 访问部分存储器和外设的访问权限，还可以独立管理 LP CPU 访问部分内存和外存的访问权限。此外，APM 还可以独立控制 GDMA，MEM_MONITOR 等可以发起总线读写请求的主设备访问部分存储器的访问权限。

PMP 和 APM 管控的区域分布如表 18.1-1 所示。对于主机 HP CPU，PMP 和 APM 的管控关系见图 18.1-1。

表 18.1-1. PMP 和 APM 管控的区域分布

从机 / 主机	ROM	HP SRAM	LP SRAM	HP CPU_PERI[1]	HP_PERI[2]	LP_PERI[3]	EXT MEM[4]
HP CPU	PMP	PMP+ CPU_APM	PMP + LP_APM/ LP_APM0	PMP + HP_APM + PERI_APM	PMP + HP_APM + PERI_APM	PMP + LP_APM + PERI_APM	PMP
LP CPU	N/A	HP_APM	LP_APM/ LP_APM0	N/A	HP_APM + PERI_APM	LP_APM + PERI_APM	N/A
GDMA	N/A	HP_APM	LP_APM/ LP_APM0	N/A	N/A	N/A	HP_APM
其他主机[5]	N/A	HP_APM	LP_APM/ LP_APM0	N/A	N/A	N/A	N/A

[1] HP CPU 相关外设寄存器，地址范围：0x600C_0000 –0x600C_FFFF
[2] 高性能系统相关外设寄存器，地址范围：0x6000_0000 –0x600A_FFFF
[3] 低功耗系统相关外设寄存器，地址范围：0x600B_0000 –0x600B_FFFF
[4] 外部存储器，如 flash、PSRAM。
[5] 如 TCM_MEM_MONITOR、PSRAM_MEM_MONITOR 等其他可以发起总线读写请求的主机，可参考表 18.5-1。
[6] CPU_APM、HP_APM、LP_APM、LP_APM0、PERI_APM 的介绍参考章节 18.5.2。

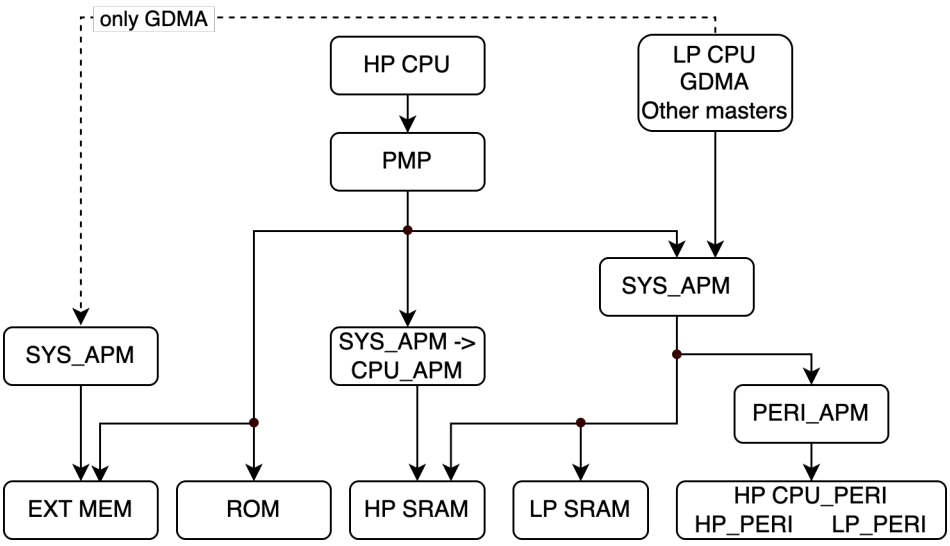


图 18.1-1. PMP 和 APM 的管控关系

如图所示，HP CPU 访问 ROM 和 EXT MEM 只受 PMP 管控，APM 无法控制。HP CPU 访问 HP SRAM 等其他数据通路，则需要先接受 PMP 的权限管控，再接受 APM 的权限管控。如果 PMP 检查不通过，则不会触发 APM 权限管控。对于外设寄存器，会经过两级 APM 权限管控，第一级根据不同地址区域划分管控范围，第二级根据不同外设划分管控范围。

PMP 相关的寄存器位于 HP CPU 内部，需要通过特殊指令进行获取或配置。关于如何配置 PMP，请参考章节 *高性能处理器* > *标准物理存储器保护*。

以下章节将详细介绍 APM 模块的特性、功能和配置。

18.2 APM 简介

APM 访问权限管理模块包含两部分：TEE（Trusted Execution Environment，可信执行环境）控制器和 APM 控制器。他们分别包含自己的寄存器模块：TEE 寄存器和 APM 寄存器。

- TEE 控制器负责配置 ESP32-C5 中主机（如 GDMA）访问存储器或外设寄存器的安全模式，共支持四种安全模式：TEE、REE0 (Rich Execution Environment)、REE1 和 REE2。
- APM 控制器分为两类，SYS_APM 控制器和 PERI_APM 控制器。SYS_APM 控制器的寄存器在 APM 寄存器中，PERI_APM 控制器的寄存器在 TEE 寄存器中。
  - SYS_APM 控制器负责管理主机访问存储器和外设寄存器的权限。通过将预先配置好的地址范围及每个地址范围对应的读、写以及执行权限，与总线上携带的信息，如 ID 号（具体可参考表 18.5-1）、安全模式、访问地址、读写执行等信息做对比，从而判断是否允许访问。
  - PERI_APM 控制器负责管理主机访问外设寄存器的权限。通过将预先配置好的每个外设一组的读、写权限寄存器，与总线上携带的信息，如安全模式、访问地址所属外设、读写执行等信息做对比，从而判断是否允许访问。

TEE 寄存器用于配置每个主机的安全模式。APM 寄存器用于指定每一个安全模式的访问权限和访问地址范围。通过 TEE 控制器和 APM 控制器的配合，可以精确控制所有主机对存储器和外设寄存器的访问权限。

18.3 主要特性

ESP32-C5 共有两个 TEE 控制器：HP_TEE 和 LP_TEE。

- HP_TEE 具有以下特性：
  - 支持 4 种安全模式
  - 支持 32 个主机安全模式配置
- LP_TEE 具有以下特性：
  - 支持 4 种安全模式
  - 支持 1 个主机 (LP CPU) 安全模式配置

ESP32-C5 共有 4 个 SYS_APM 控制器：HP_APM_CTRL、LP_APM_CTRL、LP_APM0_CTRL 和 CPU_APM_CTRL。

- HP_APM_CTRL 控制器具有以下特性：
  - 最多支持 16 组地址范围的配置
  - 支持 HP SRAM、HP CPU 外设寄存器、HP 系统外设寄存器以及外部存储器的访问权限管理
  - 支持中断功能
  - 支持异常信息记录功能
- LP_APM_CTRL 控制器具有以下特性：
  - 最多支持 8 组地址范围的配置
  - 支持 LP SRAM 低速访问的权限管理以及 LP 外设寄存器的访问权限管理
  - 支持中断功能
  - 支持异常信息记录功能
- LP_APM0_CTRL 控制器具有以下特性：
  - 最多支持 8 组地址范围的配置
  - 支持 LP SRAM 高速访问的权限管理
  - 支持中断功能
  - 支持异常信息记录功能
- CPU_APM_CTRL 控制器具有以下特性：
  - 最多支持 8 组地址范围的配置
  - 支持 HP CPU 访问 HP SRAM 的权限管理
  - 支持中断功能
  - 支持异常信息记录功能

ESP32-C5 共有 2 个 PERI_APM 控制器：HP_PERI_APM_CTRL、LP_PERI_APM_CTRL。

- HP_PERI_APM_CTRL 和 LP_PERI_APM_CTRL 具有以下相同特性：
  - 支持对每个外设寄存器单独的权限配置

- 支持通过系统总线返回错误状态到主机的功能
- 支持异常信息记录功能

18.4 TEE 与 REE 术语

TEE 代表可信执行环境 (Trusted Execution Environment)，它是与主操作系统隔离的安全区域，可以为执行敏感操作提供安全环境。

REE 代表常规执行环境 (Rich Execution Environment)，作为一般用途的执行环境，用于运行大多数应用程序。

表 18.4-2. TEE 与 REE 之间的比较

方面	TEE	REE
安全性	安全增强	普通
访问权限	TEE 模式下的主机始终拥有地址范围内的读、写、执行权限。	由用户配置安全权限。REE0/1/2 的安全等级也由用户自定义。

18.5 功能描述

18.5.1 TEE 控制器功能描述

ESP32-C5 芯片支持四种安全模式：TEE、REE0、REE1、REE2。

当 HP CPU 作为主机访问存储器或外设寄存器时，可先配置 HP CPU 的用户模式或机器模式，然后通过配置 HP_TEE 寄存器模块配置其安全模式。关于 CPU 机器模式及用户模式的配置方法请参考 RISC-V 指令集手册 v1.10 第二卷 “特权架构”（RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10）。

- 当 HP CPU 为机器模式时，其安全模式为 TEE 模式。
- 当 HP CPU 为用户模式时，其安全模式为 REE 模式，具体还需设置 TEE_M0_MODE_CTRL_REG 寄存器中的 TEE_M0_MODE 指定 REE0，REE1 或 REE2 模式：
  - 若 TEE_M0_MODE 为 0，即选择 TEE 模式，但是在 HP CPU 的用户模式下，其实际的安全模式为 REE0 模式。
  - 若 TEE_M0_MODE 为 1、2 或 3，则安全模式分别为 REE0，REE1，REE2。

对于 LP CPU 访问存储器或外设寄存器，可通过配置 LP_TEE 寄存器模块的 LP_TEE_M0_MODE 来配置其安全模式。

对于除了 LP CPU 以外的其他主机访问存储器或外设寄存器，可通过配置 HP_TEE 寄存器模块的 TEE_M*n*_MODE 来配置其安全模式，其中 *n* 与表 18.5-1 中的主机 ID 号相对应。

表 18.5-1. HP 系统主机访问来源

n 值	来源
0	HP CPU

值	来源
1	保留
2	保留
3	保留
4	保留
5	TCM_MEM_MONITOR
6	TRACE
7	保留
8	PSRAM_MEM_MONITOR
9 ~ 15	保留
16 ~ 31	详见章节 5 通用 DMA 控制器 (GDMA) > 表 5.4-1 中 0 ~ 15 对应的外设。例如，16 对应的来源为表 5.4-1 中 0 对应的外设，17 对应的来源为表 5.4-1 中 1 对应的外设，以此类推。

可以配置 TEE_M*n*_LOCK 和 LP_TEE_M0_LOCK 锁定主机安全模式的配置，只有芯片复位、系统复位及内核复位可以解除锁定并将 TEE 寄存器所有配置恢复成默认值。

18.5.2 APM 控制器功能描述

18.5.2.1 结构概览

图 18.5-1 为 APM 控制器的结构图，展示了 APM 控制器管控的总线访问路径。

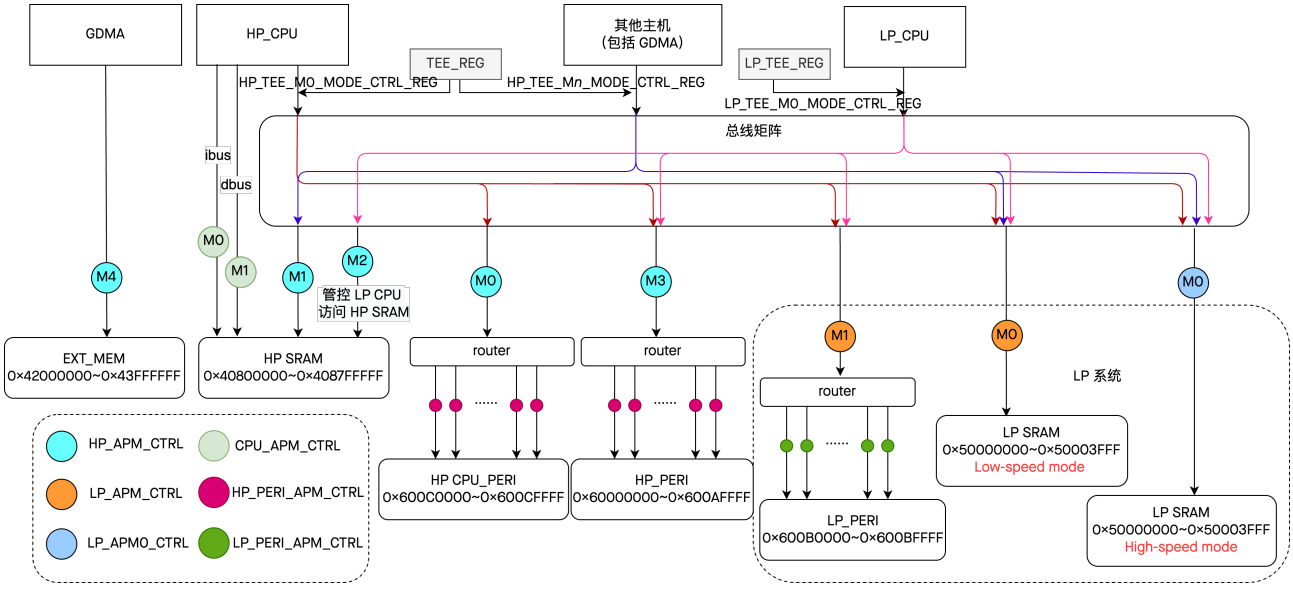


图 18.5-1. APM 控制器结构图

说明：

图中 Low-speed mode 和 High-speed mode 访问区别，请参考章节 6 系统和存储器。

SYS_APM 控制器共有 4 个：

- HP_APM_CTRL，由 HP_APM_REG 寄存器模块配置。

- LP_APM0_CTRL，由 LP_APM0_REG 寄存器模块配置。
- LP_APM_CTRL，由 LP_APM_REG 寄存器模块配置。
- CPU_APM_CTRL，由 CPU_APM_REG 寄存器模块配置。

PERI_APM 控制器共有 2 个：

- HP_PERI_APM_CTRL，由 TEE_REG 寄存器模块配置。
- LP_PERI_APM_CTRL，由 LP_TEE_REG 寄存器模块配置。

下面具体介绍各 APM 控制器的访问路径和权限检查。

- HP_APM_CTRL 管控 5 条访问路径，即图 18.5-1 中蓝色的 M0-M4，可通过配置 HP_APM_FUNC_CTRL_REG 使能各路径的权限检查（默认使能）。
- LP_APM0_CTRL 管控 1 条访问路径，即图 18.5-1 中最右边的 M0，可通过配置 LP_APM0_FUNC_CTRL_REG 使能该路径的权限检查（默认使能）。
- LP_APM_CTRL 管控 2 条访问路径，即图 18.5-1 中橙色的 M0-M1，可通过配置 LP_APM_FUNC_CTRL_REG 使能各路径的权限检查（默认使能）。
- CPU_APM_CTRL 管控 2 条访问路径，即图 18.5-1 中绿色的 M0-M1，可通过配置 CPU_APM_FUNC_CTRL_REG 使能各路径的权限检查（默认使能）。
- HP_PERI_APM_CTRL 管控高性能系统所有外设寄存器访问路径，可通过配置 TEE_*PERI*_CTRL_REG 使能各路径的权限检查。
- LP_PERI_APM_CTRL 管控低功耗系统所有外设寄存器访问路径，可通过配置 LP_TEE_*PERI*_CTRL_REG 使能各路径的权限检查。

各 APM 控制器的详细信息可参考下表 18.5-2：

表 18.5-2. APM 控制器配置信息

APM 控制器	寄存器模块	管控的访问路径数	访问路径权限检查使能	可配地址范围数	地址范围使能
HP_APM_CTRL	HP_APM_REG	5	HP_APM_FUNC_CTRL_REG	16	HP_APM_REGION_FILTER_EN_REG
LP_APM0_CTRL	LP_APM0_REG	1	LP_APM0_FUNC_CTRL_REG	8	LP_APM0_REGION_FILTER_EN_REG
LP_APM_CTRL	LP_APM_REG	2	LP_APM_FUNC_CTRL_REG	8	LP_APM_REGION_FILTER_EN_REG
CPU_APM_CTRL	CPU_APM_REG	2	CPU_APM_FUNC_CTRL_REG	8	CPU_APM_REGION_FILTER_EN_REG
HP_PERI_APM_CTRL	TEE_REG	与高性能系统寄存器模块数量相同	N/A	与高性能系统寄存器模块数量相同	N/A
LP_PERI_APM_CTRL	LP_TEE_REG	与低功耗系统寄存器模块数量相同	N/A	与低功耗系统寄存器模块数量相同	N/A

18.5.2.2 SYS_APM 控制器

18.5.2.2.1 地址范围

- HP_APM_REG 寄存器模块可配置 16 组地址范围（即区域 *n*，*n*=0-15）。HP_APM_REGION*n*_ADDR_START 和 HP_APM_REGION*n*_ADDR_END 分别配置区域 *n* 的起始地址和结束地址。可通过配置 HP_APM_REGION_FILTER_EN_REG 的 bit *n* 使能区域 *n*，默认使能区域 0，即第 1 组的地址范围。
- LP_APMO_REG 寄存器模块可配置 8 组地址范围（即区域 *n*，*n*=0-7）。LP_APMO_REGION*n*_ADDR_START 和 LP_APMO_REGION*n*_ADDR_END 分别配置区域 *n* 的起始地址和结束地址。可通过配置 LP_APMO_REGION_FILTER_EN_REG 的 bit *n* 使能区域 *n*，默认使能区域 0，即第 1 组的地址范围。
- LP_APM_REG 寄存器模块可配置 8 组地址范围（即区域 *n*，*n*=0-7）。LP_APM_REGION*n*_ADDR_START 和 LP_APM_REGION*n*_ADDR_END 分别配置区域 *n* 的起始地址和结束地址。可通过配置 LP_APM_REGION_FILTER_EN_REG 的 bit *n* 使能区域 *n*，默认使能区域 0，即第 1 组的地址范围。
- CPU_APM_REG 寄存器模块可配置 8 组地址范围（即区域 *n*，*n*=0-7）。CPU_APM_REGION*n*_ADDR_START 和 CPU_APM_REGION*n*_ADDR_END 分别配置区域 *n* 的起始地址和结束地址。可通过配置 CPU_APM_REGION_FILTER_EN_REG 的 bit *n* 使能区域 *n*，默认使能区域 0，即第 1 组的地址范围。

配置地址范围时，地址的最小对齐单位是 4 字节，即地址的低两位为 0。例如，地址范围可以设置为 0x4080000C ~ 0x40808774 或 0x600C0008 ~ 0x600CFF70。

18.5.2.2.2 地址范围的访问权限

每个地址范围内，均可以为不同的安全模式单独配置访问权限（读/写/执行）：

- TEE 模式下的主机始终拥有地址范围内的读、写、执行权限。
- REE0、REE1 和 REE2 模式下的主机，根据不同的访问路径，其读写执行权限由 HP_APM_REGION*n*_ATTR_REG、LP_APM_REGION*n*_ATTR_REG 或 LP_APMO_REGION*n*_ATTR_REG 配置。

由同一 APM 寄存器模块管理的所有访问路径共享该模块的配置，即配置的地址范围及其权限。例如上图中 HP_APM_CTRL M0-M4 在进行访问管控时，都依照 HP_APM_REG 寄存器模块中配置的 16 组地址范围及其对应的读、写、执行权限。LP_APM_CTRL M0-M1 在进行访问管控时，都依照 LP_APM_REG 寄存器模块中配置的 4 组地址范围及其对应的读、写、执行权限。

以 HP SRAM 为例。HP CPU 可通过 PMP 和 CPU_APM_CTRL 访问 HP SRAM，LP CPU 可通过 HP_APM_CTRL M2 访问 HP SRAM，其他主机都需要通过 HP_APM_CTRL M1 访问 HP SRAM。假设 HP_APM_M1_FUNC_EN 使能，REE1 安全模式的主机需要通过 HP_APM_CTRL M1 这条访问路径读取 HP SRAM 的数据，具体流程如下：

1. HP_APM_CTRL M1 会先判断请求访问的地址是否在 HP_APM_REG 寄存器模块中配置的 16 组地址范围内。
2. 假设该访问地址在配置的第 2 组地址范围内，则需判断该组地址范围是否使能，即 HP_APM_REGION_FILTER_EN 的 bit 1 是否为 1。
3. 若地址范围已使能，判断主机在 REE1 模式下对第 2 组地址范围是否具有读的权限，即 HP_APM_REGION1_ATTR_REG 中的 HP_APM_REGION1_R1_R 是否有效（即为 1），若有效，则此次读请求会被正确响应，否则读到的数据恒为 0。

当 HP 电源域（请参考章节 13 低功耗管理）掉电重启后，LP CPU 默认没有 HP SRAM 的访问权限。可以在主机 TEE 模式下配置 LP 电源域中的 LP_TEE_FORCE_ACC_HPMEM_EN 寄存器，当该寄存器使能时，LP CPU 可以访问 HP SRAM，且不需要经过 SYS_APM 控制器的权限检查。

说明：

- 芯片上电时，只有 HP CPU 默认处于 TEE 模式，其他主机均处于 REE2 模式。并且默认状态下，SYS_APM 控制器会阻止所有处于 REE0、REE1 和 REE2 模式的主机的访问请求。
- 章节 18.8 寄存器列表 中列出的所有寄存器均只有处于 TEE 安全模式的主机才可以配置。
- 在 APM 寄存器配置的多个区域的地址空间重叠的情况下，例如一个区域设置为不可读，一个区域设置为可读，两个区域的地址重叠，这部分重叠的地址空间的属性是可读的。写以及执行权限适用同样的规则。
- 可以配置 HP_APM_REGION*n*_LOCK、LP_APM_REGION*n*_LOCK 和 LP_APM0_REGION*n*_LOCK 锁定 APM 寄存器中 REGION*n* 的首尾地址及权限的配置，只有芯片复位、系统复位及内核复位可以解除锁定并将 APM 寄存器所有配置恢复成默认值。

18.5.2.3 PERI_APM 控制器

PERI_APM 不是以地址划分权限管控的范围，而是以每个外设寄存器为单位进行权限管控。每一个外设寄存器 *PERI* 都有以下 8 个域：

- HP/LP_TEE_WRITE_TEE_*PERI*
- HP/LP_TEE_WRITE_REE0_*PERI*
- HP/LP_TEE_WRITE_REE1_*PERI*
- HP/LP_TEE_WRITE_REE2_*PERI*
- HP/LP_TEE_READ_TEE_*PERI*
- HP/LP_TEE_READ_REE0_*PERI*
- HP/LP_TEE_READ_REE1_*PERI*
- HP/LP_TEE_READ_REE2_*PERI*

通过上述 8 个域，分别控制从机 *PERI* 在 TEE、REE0、REE1、REE2 四个安全模式下的读/写权限，为 1 表示有对应安全模式下的读/写权限。

复位后默认状态下，每个外设寄存器仅有 HP/LP_TEE_WRITE_TEE_*PERI* 和 HP/LP_TEE_READ_TEE_*PERI* 为 1，其他域均为 0，表示默认状态下仅有 TEE 安全模式下的主机可以配置外设寄存器。

本文所提及的六个寄存器模块，也是受 PERI_APM 权限管控的外设。六个寄存器模块和 PERI_APM 中对应的权限配置寄存器如下所示：

- HP_APM_REG，权限配置寄存器为 TEE_HP_APM_CTRL_REG
- LP_APM0_REG，权限配置寄存器为 TEE_HP_APM_CTRL_REG
- LP_APM_REG，权限配置寄存器为 LP_TEE_LP_APM_CTRL_REG
- CPU_APM_REG，权限配置寄存器为 TEE_CPU_APM_CTRL_REG
- TEE_REG，权限配置寄存器为 TEE_TEE_CTRL_REG
- LP_TEE_REG，权限配置寄存器为 LP_TEE_LP_TEE_CTRL_REG

这些权限配置寄存器中，仅有 HP/LP_TEE_WRITE/READ_TEE_*PERI* 域可配，HP/LP_TEE_WRITE/READ_REE0/1/2_*PERI* 域不可配置且固定为 0，表示这些寄存器不允许安全模式不是 TEE 的主机访问。

18.6 非法访问与中断

18.6.1 SYS_APM 控制器

若总线上携带的信息与配置不一致时，ESP32-C5 将视其为非法访问，并进行如下处理：

- 拒绝该访问请求，并返回默认值，具体表现为：
  - 读取和执行操作时返回 0
  - 写入操作时无效
- 触发中断

SYS_APM 控制器会自动记录非法访问的相关信息，包括主机的 ID、安全模式、访问地址、非法访问原因（地址越限/权限不足）以及各访问路径的权限判定状态。所有这些信息都可以从章节 18.8 寄存器列表 列出的相关寄存器中获取。

以 HP_APM_CTRL M0 数据访问路径为例，当出现非法访问时：

- HP_APM_M0_EXCEPTION_ID 记录访问主机的 ID。
- HP_APM_M0_EXCEPTION_MODE 记录访问安全模式。
- HP_APM_M0_EXCEPTION_ADDR 记录访问地址。
- HP_APM_M0_EXCEPTION_STATUS 记录非法访问原因。
  - 当访问的地址不在 16 组地址范围中配置的区域时，bit 1 会被置 1，表示地址越限。
  - 当访问的地址在某一组或多组地址范围内，但访问主机在这一组或多组地址范围内没有对应的读/写/执行操作权限时，bit 0 会被置 1，表示权限不足。
- HP_APM_M0_EXCEPTION_REGION 记录各地址范围的权限判定状态。该寄存器总共 16 个 bit，对应 16 组地址范围，bit 0 对应第 1 组地址范围。若访问的地址在配置使能的某一组地址范围内，但在这组地址范围内没有要求的读/写/执行操作权限时，本寄存器对应于这组地址范围的 bit 位会被置 1。

SYS_APM 控制器可产生 10 个中断信号，并将其发送给中断矩阵：

- HP_APM_M0_INTR
- HP_APM_M1_INTR
- HP_APM_M2_INTR
- HP_APM_M3_INTR
- HP_APM_M4_INTR
- LP_APM_M0_INTR
- LP_APM_M1_INTR
- LP_APM0_M0_INTR
- CPU_APM_M0_INTR

- CPU_APM_M1_INTR

这 10 个中断信号与图 18.5-1 中受管控的访问路径一一对应。

每个中断都配置一个中断使能寄存器 APM_INT_EN 和一个中断清除寄存器 EXCEPTION_STATUS_CLR。当中断使能寄存器为 1 时，当一个受管控的访问路径有非法访问，则对应的中断会被产生，配置中断清除寄存器，则中断信号会被清除，直至下一次非法访问到来。

18.6.2 PERI_APM 控制器

若总线上携带的信息与配置不一致时，ESP32-C5 将视其为非法访问，并进行如下处理：

- 拒绝该访问请求，并返回默认值，具体表现为：
  - 读取和执行操作时返回 0
  - 写入操作时无效
- 通过总线返回错误信息到主机，使主机进入异常状态

可配置 HP/LP_BUS_ERR_RESP_EN 为 1 使能非法访问返回总线错误信息的功能。

PERI_APM 控制器不会记录非法访问的相关信息。

18.7 配置流程

主机访问存储器或外设寄存器路径中 APM 管控的配置流程为：

1. 配置 HP CPU 到机器模式（即 TEE 模式）。
2. 根据表 18.5-1 中的主机 ID 号，配置 TEE_M*n*_MODE 寄存器，或者 LP_TEE_M*n*_MODE 寄存器，配置 ESP32-C5 中主机的安全模式。
3. 配置 HP_APM_REGION*n*_ADDR_START、HP_APM_REGION*n*_ADDR_END 寄存器，或者 LP_APM_REGION*n*_ADDR_START、LP_APM_REGION*n*_ADDR_END 寄存器，或者 LP_APMO_REGION*n*_ADDR_START、LP_APMO_REGION*n*_ADDR_END 寄存器，或者 CPU_APM_REGION*n*_ADDR_START、CPU_APM_REGION*n*_ADDR_END 寄存器，配置 APM 寄存器中的各组地址范围。
4. 配置 HP_APM_REGION*n*_ATTR_REG 寄存器，或者 LP_APM_REGION*n*_ATTR_REG 寄存器，或者 LP_APMO_REGION*n*_ATTR_REG 寄存器，或者 CPU_APM_REGION*n*_ATTR_REG 寄存器，配置 APM 寄存器中的各组地址范围的访问权限。
5. 配置 HP_APM_REGION_FILTER_ EN_REG 寄存器，或者 LP_APM_REGION_FILTER_ EN_REG 寄存器，或者 LP_APMO_REGION_FILTER_ EN_REG 寄存器，或者 CPU_APM_REGION_FILTER_ EN_REG 寄存器，使能对应地址范围的权限检查。
6. 配置 HP_APM_ FUNC_CTRL_REG、LP_APM _FUNC_CTRL_REG、LP_APMO _FUNC_CTRL_REG 或 CPU_APM _FUNC_CTRL_REG 寄存器使能对应路径的 APM 控制器，默认使能。
7. （仅访问外设寄存器时需要）配置 TEE_*PERI*_CTRL_REG 或 LP_TEE_*PERI*_CTRL_REG，配置 *PERI* 寄存器的访问权限。

以 I2S 通过 GDMA 访问 HP SRAM 为例，假设仅允许其在第 4 组地址范围 0x40805000 ~ 0x4080FFFF 地址范围内读写，则配置流程如下：

1. 配置 HP CPU 为机器模式（即 TEE 模式）。
2. 根据表 18.5-1 中的主机 ID 号，配置 TEE_M19_MODE 寄存器为 1，将 I2S 通过 GDMA 访问的安全模式设置为 REE0 模式。
3. 配置 HP_APM_REGION3_ADDR_START 为 0x40805000，配置 HP_APM_REGION3_ADDR_END 为 0x4080EFFC。
4. 配置 HP_APM_REGION3_RO_W 和 HP_APM_REGION3_RO_R 为 1，使能对地址范围的写入、读取权限。
5. 配置 HP_APM_REGION_FILTER_EN 的 bit 3 为 1，使能区域 3（即第 4 组地址范围）。
6. 配置 HP_APM_M1_FUNC_EN 为 1，使能 HP_APM_CTRL M1 检查权限。

通过以上配置，I2S 可通过 GDMA 在 HP SRAM 中的 0x40805000 ~ 0x4080FFFF 地址范围内读写。

18.8 寄存器列表

18.8.1 HP_APM_REG

本小节的地址均为相对于 HP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
HP_APM_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
HP_APM_REGION0_ADDR_START_REG	配置区域地址寄存器	0x0004	R/W
HP_APM_REGION0_ADDR_END_REG	配置区域地址寄存器	0x0008	R/W
HP_APM_REGION1_ADDR_START_REG	配置区域地址寄存器	0x0010	R/W
HP_APM_REGION1_ADDR_END_REG	配置区域地址寄存器	0x0014	R/W
HP_APM_REGION2_ADDR_START_REG	配置区域地址寄存器	0x001C	R/W
HP_APM_REGION2_ADDR_END_REG	配置区域地址寄存器	0x0020	R/W
HP_APM_REGION3_ADDR_START_REG	配置区域地址寄存器	0x0028	R/W
HP_APM_REGION3_ADDR_END_REG	配置区域地址寄存器	0x002C	R/W
HP_APM_REGION4_ADDR_START_REG	配置区域地址寄存器	0x0034	R/W
HP_APM_REGION4_ADDR_END_REG	配置区域地址寄存器	0x0038	R/W
HP_APM_REGION5_ADDR_START_REG	配置区域地址寄存器	0x0040	R/W
HP_APM_REGION5_ADDR_END_REG	配置区域地址寄存器	0x0044	R/W
HP_APM_REGION6_ADDR_START_REG	配置区域地址寄存器	0x004C	R/W
HP_APM_REGION6_ADDR_END_REG	配置区域地址寄存器	0x0050	R/W
HP_APM_REGION7_ADDR_START_REG	配置区域地址寄存器	0x0058	R/W
HP_APM_REGION7_ADDR_END_REG	配置区域地址寄存器	0x005C	R/W
HP_APM_REGION8_ADDR_START_REG	配置区域地址寄存器	0x0064	R/W
HP_APM_REGION8_ADDR_END_REG	配置区域地址寄存器	0x0068	R/W
HP_APM_REGION9_ADDR_START_REG	配置区域地址寄存器	0x0070	R/W
HP_APM_REGION9_ADDR_END_REG	配置区域地址寄存器	0x0074	R/W
HP_APM_REGION10_ADDR_START_REG	配置区域地址寄存器	0x007C	R/W

名称	描述	地址	访问
HP_APM_REGION10_ADDR_END_REG	配置区域地址寄存器	0x0080	R/W
HP_APM_REGION11_ADDR_START_REG	配置区域地址寄存器	0x0088	R/W
HP_APM_REGION11_ADDR_END_REG	配置区域地址寄存器	0x008C	R/W
HP_APM_REGION12_ADDR_START_REG	配置区域地址寄存器	0x0094	R/W
HP_APM_REGION12_ADDR_END_REG	配置区域地址寄存器	0x0098	R/W
HP_APM_REGION13_ADDR_START_REG	配置区域地址寄存器	0x00A0	R/W
HP_APM_REGION13_ADDR_END_REG	配置区域地址寄存器	0x00A4	R/W
HP_APM_REGION14_ADDR_START_REG	配置区域地址寄存器	0x00AC	R/W
HP_APM_REGION14_ADDR_END_REG	配置区域地址寄存器	0x00B0	R/W
HP_APM_REGION15_ADDR_START_REG	配置区域地址寄存器	0x00B8	R/W
HP_APM_REGION15_ADDR_END_REG	配置区域地址寄存器	0x00BC	R/W
HP_APM_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
HP_APM_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
HP_APM_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
HP_APM_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W
HP_APM_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
HP_APM_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
HP_APM_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
HP_APM_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
HP_APM_REGION8_ATTR_REG	区域访问权限配置寄存器	0x006C	R/W
HP_APM_REGION9_ATTR_REG	区域访问权限配置寄存器	0x0078	R/W
HP_APM_REGION10_ATTR_REG	区域访问权限配置寄存器	0x0084	R/W
HP_APM_REGION11_ATTR_REG	区域访问权限配置寄存器	0x0090	R/W
HP_APM_REGION12_ATTR_REG	区域访问权限配置寄存器	0x009C	R/W
HP_APM_REGION13_ATTR_REG	区域访问权限配置寄存器	0x00A8	R/W
HP_APM_REGION14_ATTR_REG	区域访问权限配置寄存器	0x00B4	R/W
HP_APM_REGION15_ATTR_REG	区域访问权限配置寄存器	0x00C0	R/W
HP_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
HP_APM_M0_STATUS_REG	HP_APM_CTRL M0 状态寄存器	0x00C8	RO
HP_APM_M0_STATUS_CLR_REG	HP_APM_CTRL M0 状态清除寄存器	0x00CC	WT
HP_APM_M0_EXCEPTION_INFO0_REG	HP_APM_CTRL M0 异常信息寄存器	0x00D0	RO
HP_APM_M0_EXCEPTION_INFO1_REG	HP_APM_CTRL M0 异常信息寄存器	0x00D4	RO
HP_APM_M1_STATUS_REG	HP_APM_CTRL M1 状态寄存器	0x00D8	RO
HP_APM_M1_STATUS_CLR_REG	HP_APM_CTRL M1 状态清除寄存器	0x00DC	WT
HP_APM_M1_EXCEPTION_INFO0_REG	HP_APM_CTRL M1 异常信息寄存器	0x00E0	RO
HP_APM_M1_EXCEPTION_INFO1_REG	HP_APM_CTRL M1 异常信息寄存器	0x00E4	RO
HP_APM_M2_STATUS_REG	HP_APM_CTRL M2 状态寄存器	0x00E8	RO
HP_APM_M2_STATUS_CLR_REG	HP_APM_CTRL M2 状态清除寄存器	0x00EC	WT
HP_APM_M2_EXCEPTION_INFO0_REG	HP_APM_CTRL M2 异常信息寄存器	0x00F0	RO
HP_APM_M2_EXCEPTION_INFO1_REG	HP_APM_CTRL M2 异常信息寄存器	0x00F4	RO
HP_APM_M3_STATUS_REG	HP_APM_CTRL M3 状态寄存器	0x00F8	RO
HP_APM_M3_STATUS_CLR_REG	HP_APM_CTRL M3 状态清除寄存器	0x00FC	WT

名称	描述	地址	访问
HP_APM_M3_EXCEPTION_INFO0_REG	HP_APM_CTRL M3 异常信息寄存器	0x0100	RO
HP_APM_M3_EXCEPTION_INFO1_REG	HP_APM_CTRL M3 异常信息寄存器	0x0104	RO
HP_APM_M4_STATUS_REG	HP_APM_CTRL M4 状态寄存器	0x0108	RO
HP_APM_M4_STATUS_CLR_REG	HP_APM_CTRL M4 状态清除寄存器	0x010C	WT
HP_APM_M4_EXCEPTION_INFO0_REG	HP_APM_CTRL M4 异常信息寄存器	0x0110	RO
HP_APM_M4_EXCEPTION_INFO1_REG	HP_APM_CTRL M4 异常信息寄存器	0x0114	RO
中断寄存器			
HP_APM_INT_EN_REG	HP_APM_CTRL M0/1/2/3/4 中断使能寄存器	0x0118	R/W
时钟门控寄存器			
HP_APM_CLOCK_GATE_REG	时钟门控寄存器	0x07F8	R/W
版本控制寄存器			
HP_APM_DATE_REG	版本控制寄存器	0x07FC	R/W

18.8.2 LP_APM_REG

本小节的地址均为相对于 LP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_APM_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
LP_APM_REGION0_ADDR_START_REG	配置区域地址寄存器	0x0004	R/W
LP_APM_REGION0_ADDR_END_REG	配置区域地址寄存器	0x0008	R/W
LP_APM_REGION1_ADDR_START_REG	配置区域地址寄存器	0x0010	R/W
LP_APM_REGION1_ADDR_END_REG	配置区域地址寄存器	0x0014	R/W
LP_APM_REGION2_ADDR_START_REG	配置区域地址寄存器	0x001C	R/W
LP_APM_REGION2_ADDR_END_REG	配置区域地址寄存器	0x0020	R/W
LP_APM_REGION3_ADDR_START_REG	配置区域地址寄存器	0x0028	R/W
LP_APM_REGION3_ADDR_END_REG	配置区域地址寄存器	0x002C	R/W
LP_APM_REGION4_ADDR_START_REG	配置区域地址寄存器	0x0034	R/W
LP_APM_REGION4_ADDR_END_REG	配置区域地址寄存器	0x0038	R/W
LP_APM_REGION5_ADDR_START_REG	配置区域地址寄存器	0x0040	R/W
LP_APM_REGION5_ADDR_END_REG	配置区域地址寄存器	0x0044	R/W
LP_APM_REGION6_ADDR_START_REG	配置区域地址寄存器	0x004C	R/W
LP_APM_REGION6_ADDR_END_REG	配置区域地址寄存器	0x0050	R/W
LP_APM_REGION7_ADDR_START_REG	配置区域地址寄存器	0x0058	R/W
LP_APM_REGION7_ADDR_END_REG	配置区域地址寄存器	0x005C	R/W
LP_APM_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
LP_APM_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
LP_APM_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
LP_APM_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W

名称	描述	地址	访问
LP_APM_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
LP_APM_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
LP_APM_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
LP_APM_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
LP_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
LP_APM_M0_STATUS_REG	LP_APM_CTRL M0 状态寄存器	0x00C8	RO
LP_APM_M0_STATUS_CLR_REG	LP_APM_CTRL M0 状态清除寄存器	0x00CC	WT
LP_APM_M0_EXCEPTION_INFO0_REG	LP_APM_CTRL M0 异常信息寄存器	0x00D0	RO
LP_APM_M0_EXCEPTION_INFO1_REG	LP_APM_CTRL M0 异常信息寄存器	0x00D4	RO
LP_APM_M1_STATUS_REG	LP_APM_CTRL M1 状态寄存器	0x00D8	RO
LP_APM_M1_STATUS_CLR_REG	LP_APM_CTRL M1 状态清除寄存器	0x00DC	WT
LP_APM_M1_EXCEPTION_INFO0_REG	LP_APM_CTRL M1 异常信息寄存器	0x00E0	RO
LP_APM_M1_EXCEPTION_INFO1_REG	LP_APM_CTRL M1 异常信息寄存器	0x00E4	RO
中断寄存器			
LP_APM_INT_EN_REG	LP_APM_CTRL M0/1 中断使能寄存器	0x00E8	R/W
时钟门控寄存器			
LP_APM_CLOCK_GATE_REG	时钟门控寄存器	0x00EC	R/W
版本控制寄存器			
LP_APM_DATE_REG	版本控制寄存器	0x00FC	R/W

18.8.3 LP_APM0_REG

本小节的地址均为相对于 LP_APM0 的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_APM0_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
LP_APM0_REGION0_ADDR_START_REG	配置区域地址寄存器	0x0004	R/W
LP_APM0_REGION0_ADDR_END_REG	配置区域地址寄存器	0x0008	R/W
LP_APM0_REGION1_ADDR_START_REG	配置区域地址寄存器	0x0010	R/W
LP_APM0_REGION1_ADDR_END_REG	配置区域地址寄存器	0x0014	R/W
LP_APM0_REGION2_ADDR_START_REG	配置区域地址寄存器	0x001C	R/W
LP_APM0_REGION2_ADDR_END_REG	配置区域地址寄存器	0x0020	R/W
LP_APM0_REGION3_ADDR_START_REG	配置区域地址寄存器	0x0028	R/W
LP_APM0_REGION3_ADDR_END_REG	配置区域地址寄存器	0x002C	R/W
LP_APM0_REGION4_ADDR_START_REG	配置区域地址寄存器	0x0034	R/W
LP_APM0_REGION4_ADDR_END_REG	配置区域地址寄存器	0x0038	R/W
LP_APM0_REGION5_ADDR_START_REG	配置区域地址寄存器	0x0040	R/W
LP_APM0_REGION5_ADDR_END_REG	配置区域地址寄存器	0x0044	R/W
LP_APM0_REGION6_ADDR_START_REG	配置区域地址寄存器	0x004C	R/W

名称	描述	地址	访问
LP_APM0_REGION6_ADDR_END_REG	配置区域地址寄存器	0x0050	R/W
LP_APM0_REGION7_ADDR_START_REG	配置区域地址寄存器	0x0058	R/W
LP_APM0_REGION7_ADDR_END_REG	配置区域地址寄存器	0x005C	R/W
LP_APM0_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
LP_APM0_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
LP_APM0_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
LP_APM0_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W
LP_APM0_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
LP_APM0_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
LP_APM0_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
LP_APM0_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
LP_APM0_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
LP_APM0_M0_STATUS_REG	LP_APM0_CTRL M0 状态寄存器	0x00C8	RO
LP_APM0_M0_STATUS_CLR_REG	LP_APM0_CTRL M0 状态清除寄存器	0x00CC	WT
LP_APM0_M0_EXCEPTION_INFO0_REG	LP_APM0_CTRL M0 异常信息寄存器	0x00D0	RO
LP_APM0_M0_EXCEPTION_INFO1_REG	LP_APM0_CTRL M0 异常信息寄存器	0x00D4	RO
中断寄存器			
LP_APM0_INT_EN_REG	LP_APM0_CTRL 中断使能寄存器	0x00D8	R/W
时钟门控寄存器			
LP_APM0_CLOCK_GATE_REG	时钟门控寄存器	0x00DC	R/W
版本控制寄存器			
LP_APM0_DATE_REG	版本控制寄存器	0x07FC	R/W

18.8.4 CPU_APM_REG

本小节的地址均为相对于 CPU_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
CPU_APM_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
配置区域地址寄存器			
CPU_APM_REGION0_ADDR_START_REG	配置区域地址寄存器	0x0004	varies
CPU_APM_REGION0_ADDR_END_REG	配置区域地址寄存器	0x0008	varies
CPU_APM_REGION1_ADDR_START_REG	配置区域地址寄存器	0x0010	varies
CPU_APM_REGION1_ADDR_END_REG	配置区域地址寄存器	0x0014	varies
CPU_APM_REGION2_ADDR_START_REG	配置区域地址寄存器	0x001C	varies
CPU_APM_REGION2_ADDR_END_REG	配置区域地址寄存器	0x0020	varies
CPU_APM_REGION3_ADDR_START_REG	配置区域地址寄存器	0x0028	varies
CPU_APM_REGION3_ADDR_END_REG	配置区域地址寄存器	0x002C	varies
CPU_APM_REGION4_ADDR_START_REG	配置区域地址寄存器	0x0034	varies

名称	描述	地址	访问
CPU_APM_REGION4_ADDR_END_REG	配置区域地址寄存器	0x0038	varies
CPU_APM_REGION5_ADDR_START_REG	配置区域地址寄存器	0x0040	varies
CPU_APM_REGION5_ADDR_END_REG	配置区域地址寄存器	0x0044	varies
CPU_APM_REGION6_ADDR_START_REG	配置区域地址寄存器	0x004C	varies
CPU_APM_REGION6_ADDR_END_REG	配置区域地址寄存器	0x0050	varies
CPU_APM_REGION7_ADDR_START_REG	配置区域地址寄存器	0x0058	varies
CPU_APM_REGION7_ADDR_END_REG	配置区域地址寄存器	0x005C	varies
CPU_APM_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
CPU_APM_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
CPU_APM_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
CPU_APM_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W
CPU_APM_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
CPU_APM_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
CPU_APM_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
CPU_APM_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
CPU_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
CPU_APM_M0_STATUS_REG	CPU_APM_CTRL M0 状态寄存器	0x00C8	RO
CPU_APM_M0_STATUS_CLR_REG	CPU_APM_CTRL M0 状态清除寄存器	0x00CC	WT
CPU_APM_M0_EXCEPTION_INFO0_REG	CPU_APM_CTRL M0 异常信息寄存器	0x00D0	RO
CPU_APM_M0_EXCEPTION_INFO1_REG	CPU_APM_CTRL M0 异常信息寄存器	0x00D4	RO
CPU_APM_M1_STATUS_REG	CPU_APM_CTRL M1 状态寄存器	0x00D8	RO
CPU_APM_M1_STATUS_CLR_REG	CPU_APM_CTRL M1 状态清除寄存器	0x00DC	WT
CPU_APM_M1_EXCEPTION_INFO0_REG	CPU_APM_CTRL M1 异常信息寄存器	0x00E0	RO
CPU_APM_M1_EXCEPTION_INFO1_REG	CPU_APM_CTRL M1 异常信息寄存器	0x00E4	RO
中断寄存器			
CPU_APM_INT_EN_REG	CPU_APM_CTRL M0/1 中断使能寄存器	0x0118	R/W
时钟门控寄存器			
CPU_APM_CLOCK_GATE_REG	时钟门控寄存器	0x07F8	R/W
版本控制寄存器			
CPU_APM_DATE_REG	版本控制寄存器	0x07FC	R/W

18.8.5 TEE_REG

本小节的地址均为相对于 TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	地址	权限
配置寄存器			
TEE_M0_MODE_CTRL_REG	安全模式配置寄存器	0x0000	R/W
TEE_M1_MODE_CTRL_REG	安全模式配置寄存器	0x0004	R/W

名称	描述	地址	权限
TEE_M2_MODE_CTRL_REG	安全模式配置寄存器	0x0008	R/W
TEE_M3_MODE_CTRL_REG	安全模式配置寄存器	0x000C	R/W
TEE_M4_MODE_CTRL_REG	安全模式配置寄存器	0x0010	R/W
TEE_M5_MODE_CTRL_REG	安全模式配置寄存器	0x0014	R/W
TEE_M6_MODE_CTRL_REG	安全模式配置寄存器	0x0018	R/W
TEE_M7_MODE_CTRL_REG	安全模式配置寄存器	0x001C	R/W
TEE_M8_MODE_CTRL_REG	安全模式配置寄存器	0x0020	R/W
TEE_M9_MODE_CTRL_REG	安全模式配置寄存器	0x0024	R/W
TEE_M10_MODE_CTRL_REG	安全模式配置寄存器	0x0028	R/W
TEE_M11_MODE_CTRL_REG	安全模式配置寄存器	0x002C	R/W
TEE_M12_MODE_CTRL_REG	安全模式配置寄存器	0x0030	R/W
TEE_M13_MODE_CTRL_REG	安全模式配置寄存器	0x0034	R/W
TEE_M14_MODE_CTRL_REG	安全模式配置寄存器	0x0038	R/W
TEE_M15_MODE_CTRL_REG	安全模式配置寄存器	0x003C	R/W
TEE_M16_MODE_CTRL_REG	安全模式配置寄存器	0x0040	R/W
TEE_M17_MODE_CTRL_REG	安全模式配置寄存器	0x0044	R/W
TEE_M18_MODE_CTRL_REG	安全模式配置寄存器	0x0048	R/W
TEE_M19_MODE_CTRL_REG	安全模式配置寄存器	0x004C	R/W
TEE_M20_MODE_CTRL_REG	安全模式配置寄存器	0x0050	R/W
TEE_M21_MODE_CTRL_REG	安全模式配置寄存器	0x0054	R/W
TEE_M22_MODE_CTRL_REG	安全模式配置寄存器	0x0058	R/W
TEE_M23_MODE_CTRL_REG	安全模式配置寄存器	0x005C	R/W
TEE_M24_MODE_CTRL_REG	安全模式配置寄存器	0x0060	R/W
TEE_M25_MODE_CTRL_REG	安全模式配置寄存器	0x0064	R/W
TEE_M26_MODE_CTRL_REG	安全模式配置寄存器	0x0068	R/W
TEE_M27_MODE_CTRL_REG	安全模式配置寄存器	0x006C	R/W
TEE_M28_MODE_CTRL_REG	安全模式配置寄存器	0x0070	R/W
TEE_M29_MODE_CTRL_REG	安全模式配置寄存器	0x0074	R/W
TEE_M30_MODE_CTRL_REG	安全模式配置寄存器	0x0078	R/W
TEE_M31_MODE_CTRL_REG	安全模式配置寄存器	0x007C	R/W
外设读/写控制寄存器			
TEE_UART0_CTRL_REG	UART0 读/写控制寄存器	0x0088	R/W
TEE_UART1_CTRL_REG	UART1 读/写控制寄存器	0x008C	R/W
TEE_UHCI0_CTRL_REG	UHCI 读/写控制寄存器	0x0090	R/W
TEE_I2C_EXT0_CTRL_REG	I2C 读/写控制寄存器	0x0094	R/W
TEE_I2S_CTRL_REG	I2S 读/写控制寄存器	0x009C	R/W
TEE_PARL_IO_CTRL_REG	PARL_IO 读/写控制寄存器	0x00A0	R/W
TEE_PWM_CTRL_REG	MCPWM 读/写控制寄存器	0x00A4	R/W
TEE_LEDC_CTRL_REG	LEDC 读/写控制寄存器	0x00AC	R/W
TEE_TWAI0_CTRL_REG	TWAI0 读/写控制寄存器	0x00B0	R/W
TEE_USB_SERIAL_JTAG_CTRL_REG	USB_SERIAL_JTAG 读/写控制寄存器	0x00B4	R/W
TEE_RMT_CTRL_REG	RMT 读/写控制寄存器	0x00B8	R/W

名称	描述	地址	权限
TEE_GDMA_CTRL_REG	GDMA 读/写控制寄存器	0x00BC	R/W
TEE_ETM_CTRL_REG	SOC_ETM 读/写控制寄存器	0x00C4	R/W
TEE_INTMTX_CTRL_REG	INTMTX 读/写控制寄存器	0x00C8	R/W
TEE_APB_ADC_CTRL_REG	SAR ADC 读/写控制寄存器	0x00D0	R/W
TEE_TIMERGROUP0_CTRL_REG	TIMG0 读/写控制寄存器	0x00D4	R/W
TEE_TIMERGROUP1_CTRL_REG	TIMG1 读/写控制寄存器	0x00D8	R/W
TEE_SYSTIMER_CTRL_REG	SYSTIMER 读/写控制寄存器	0x00DC	R/W
TEE_PCNT_CTRL_REG	PCNT 读/写控制寄存器	0x00F4	R/W
TEE_IOMUX_CTRL_REG	IO MUX 读/写控制寄存器	0x00F8	R/W
TEE_PSRAM_MEM_MONITOR_CTRL_REG	PSRAM_MEM_MONITOR 读/写控制寄存器	0x00FC	R/W
TEE_MEM_ACS_MONITOR_CTRL_REG	TCM_MEM_MONITOR 读/写控制寄存器	0x0100	R/W
TEE_HP_SYSTEM_REG_CTRL_REG	HP_SYSREG 读/写控制寄存器	0x0104	R/W
TEE_PCR_REG_CTRL_REG	PCR 读/写控制寄存器	0x0108	R/W
TEE_MSPI_CTRL_REG	SPI01 读/写控制寄存器	0x010C	R/W
TEE_HP_APM_CTRL_REG	HP_APM 和 LP_APM0 读/写控制寄存器	0x0110	varies
TEE_CPU_APM_CTRL_REG	CPU_APM_REG 读/写控制寄存器	0x0114	varies
TEE_TEE_CTRL_REG	TEE 读/写控制寄存器	0x0118	varies
TEE_CRYPT_CTRL_REG	CRYPT 读/写控制寄存器，包含 AES 到 ECDSA_DS 地址区间段内的所有安全外设	0x011C	R/W
TEE_TRACE_CTRL_REG	TRACE 读/写控制寄存器	0x0120	R/W
TEE_CPU_BUS_MONITOR_CTRL_REG	BUS_MONITOR 读/写控制寄存器	0x0128	R/W
TEE_INTPRI_REG_CTRL_REG	INTPRI_REG 读/写控制寄存器	0x012C	R/W
TEE_TWAI1_CTRL_REG	TWAI1 读/写控制寄存器	0x0138	R/W
TEE_SPI2_CTRL_REG	SPI2 读/写控制寄存器	0x013C	R/W
TEE_BS_CTRL_REG	BITSCRAMBLER 读/写控制寄存器	0x0140	R/W
TEE_BUS_ERR_CONF_REG	错误信息返回配置寄存器	0x0FF0	R/W
时钟门控寄存器			
TEE_CLOCK_GATE_REG	时钟门控寄存器	0x0FF8	R/W
版本控制寄存器			
TEE_DATE_REG	版本控制寄存器	0x0FFC	R/W

18.8.6 LP_TEE_REG

本小节的地址均为相对于 LP_TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	权限
配置寄存器			
LP_TEE_MO_MODE_CTRL_REG	安全模式配置寄存器	0x0000	R/W
外设读/写控制寄存器			
LP_TEE_EFUSE_CTRL_REG	eFuse 读/写控制寄存器	0x0004	R/W
LP_TEE_PMU_CTRL_REG	PMU 读/写控制寄存器	0x0008	R/W
LP_TEE_CLKRST_CTRL_REG	LP_CLKRST 读/写控制寄存器	0x000C	R/W
LP_TEE_LP_AON_CTRL_CTRL_REG	LP_AON 读/写控制寄存器	0x0010	R/W
LP_TEE_LP_TIMER_CTRL_REG	LP_TIMER 读/写控制寄存器	0x0014	R/W
LP_TEE_LP_WDT_CTRL_REG	LP_WDT 读/写控制寄存器	0x0018	R/W
LP_TEE_LP_PERI_CTRL_REG	LPPERI 读/写控制寄存器	0x001C	R/W
LP_TEE_LP_ANA_PERI_CTRL_REG	LP_ANA_PERI 读/写控制寄存器	0x0020	R/W
LP_TEE_LP_IO_CTRL_REG	LP_GPIO 和 LP_IO_MUX 读/写控制寄存器	0x002C	R/W
LP_TEE_LP_TEE_CTRL_REG	LP_TEE 读/写控制寄存器	0x0034	varies
LP_TEE_UART_CTRL_REG	LP_UART 读/写控制寄存器	0x0038	R/W
LP_TEE_I2C_EXT_CTRL_REG	LP_I2C 读/写控制寄存器	0x0040	R/W
LP_TEE_I2C_ANA_MST_CTRL_REG	I2C_ANA_MST 读/写控制寄存器	0x0044	R/W
LP_TEE_LP_APM_CTRL_REG	LP_APM 读/写控制寄存器	0x004C	varies
LP_TEE_FORCE_ACC_HP_REG	LP CPU 强制访问 HP SRAM 配置寄存器	0x0090	R/W
LP_TEE_BUS_ERR_CONF_REG	错误信息返回配置寄存器	0x00F0	R/W
时钟门控寄存器			
LP_TEE_CLOCK_GATE_REG	时钟门控寄存器	0x00F8	R/W
版本控制寄存器			
LP_TEE_DATE_REG	版本控制寄存器	0x00FC	R/W

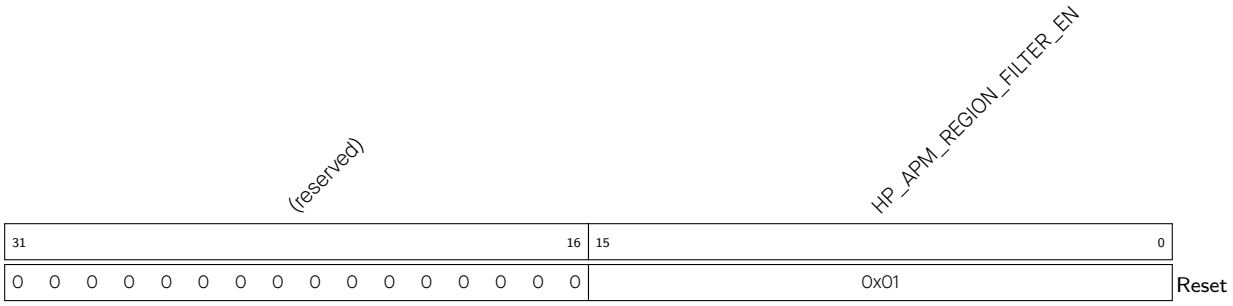
18.9 寄存器

18.9.1 HP_APM_REG

本小节的地址均为相对于 HP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 18.1. HP_APM_REGION_FILTER_EN_REG (0x0000)



HP_APM_REGION_FILTER_EN 配置 bit *n* (0-15) 使能区域 *n* (0-15) 地址范围的权限检查。

0：禁用

1：使能

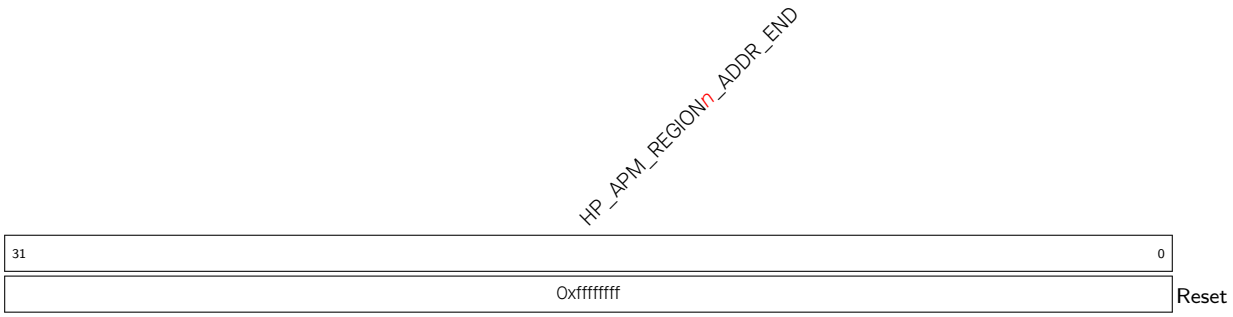
(R/W)

Register 18.2. HP_APM_REGION*n*_ADDR_START_REG (*n*: 0-15) (0x0004+0xC**n*)



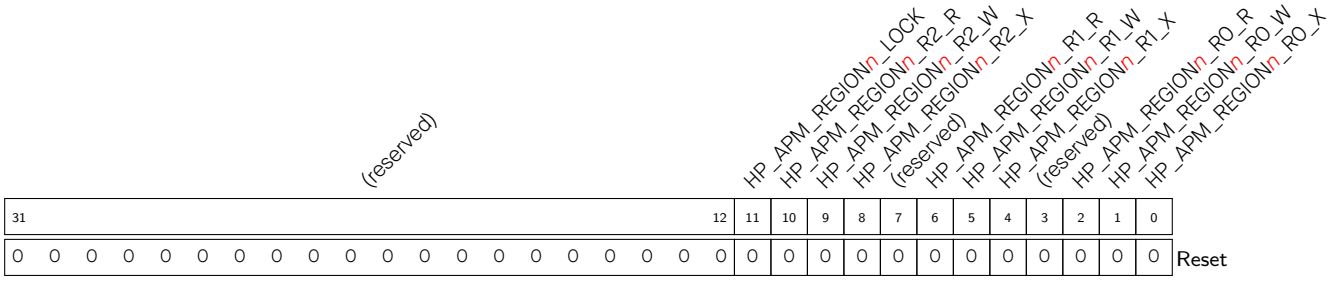
HP_APM_REGION*n*_ADDR_START 配置区域 *n* 的起始地址。(R/W)

Register 18.3. HP_APM_REGIONn_ADDR_END_REG (n: 0-15) (0x0008+0xC*n)



HP_APM_REGIONn_ADDR_END 配置区域 n 的结束地址。(R/W)

Register 18.4. HP_APM_REGIONn_ATTR_REG (n: 0-15) (0x000C+0xC*n)



HP_APM_REGIONn_R0_X 配置 REE0 模式下对区域 n 的执行权限。(R/W)

HP_APM_REGIONn_R0_W 配置 REE0 模式下对区域 n 的写入权限。(R/W)

HP_APM_REGIONn_R0_R 配置 REE0 模式下对区域 n 的读取权限。(R/W)

HP_APM_REGIONn_R1_X 配置 REE1 模式下对区域 n 的执行权限。(R/W)

HP_APM_REGIONn_R1_W 配置 REE1 模式下对区域 n 的写入权限。(R/W)

HP_APM_REGIONn_R1_R 配置 REE1 模式下对区域 n 的读取权限。(R/W)

HP_APM_REGIONn_R2_X 配置 REE2 模式下对区域 n 的执行权限。(R/W)

HP_APM_REGIONn_R2_W 配置 REE2 模式下对区域 n 的写入权限。(R/W)

HP_APM_REGIONn_R2_R 配置 REE2 模式下对区域 n 的读取权限。(R/W)

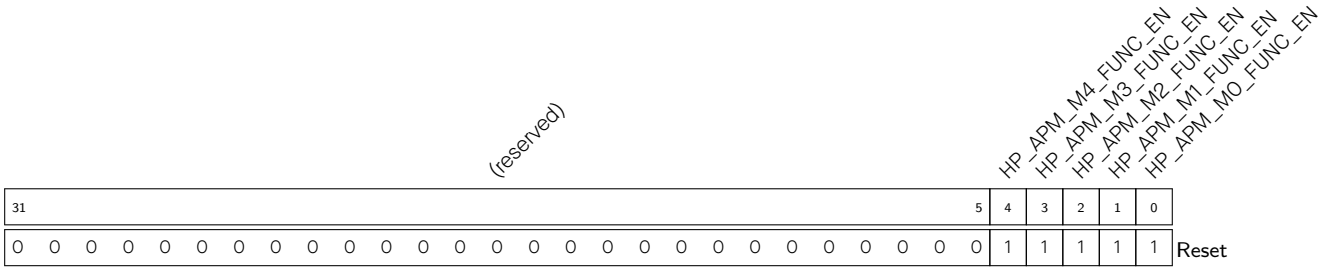
HP_APM_REGIONn_LOCK 锁定区域 n 的首尾地址及权限的配置 (HP_APM_REGIONn_ADDR_START_REG、HP_APM_REGIONn_ADDR_END_REG 和 HP_APM_REGIONn_ATTR_REG)。

0：不锁定

1：锁定

(R/W)

Register 18.5. HP_APM_FUNC_CTRL_REG (0x00C4)



HP_APM_M0_FUNC_EN 配置使能 HP_APM_CTRL M0 权限检查。(R/W)

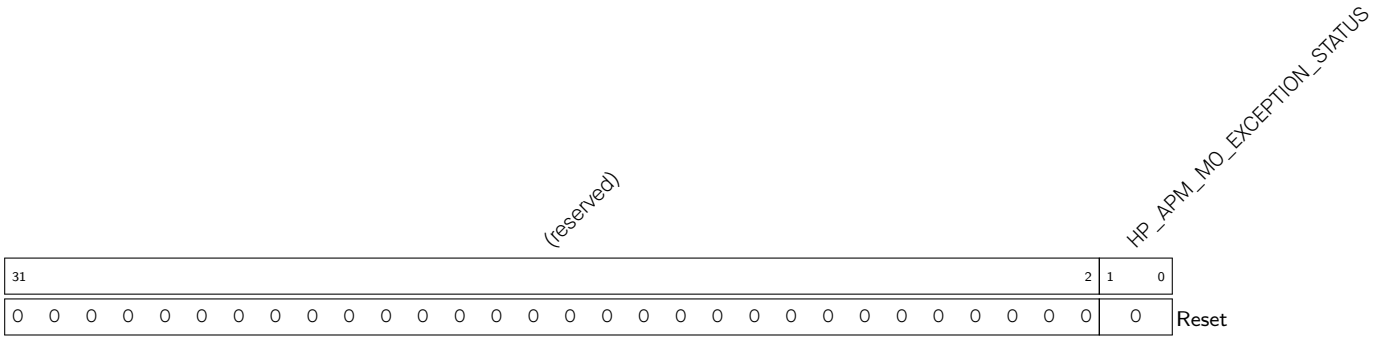
HP_APM_M1_FUNC_EN 配置使能 HP_APM_CTRL M1 权限检查。(R/W)

HP_APM_M2_FUNC_EN 配置使能 HP_APM_CTRL M2 权限检查。(R/W)

HP_APM_M3_FUNC_EN 配置使能 HP_APM_CTRL M3 权限检查。(R/W)

HP_APM_M4_FUNC_EN 配置使能 HP_APM_CTRL M4 权限检查。(R/W)

Register 18.6. HP_APM_M0_STATUS_REG (0x00C8)



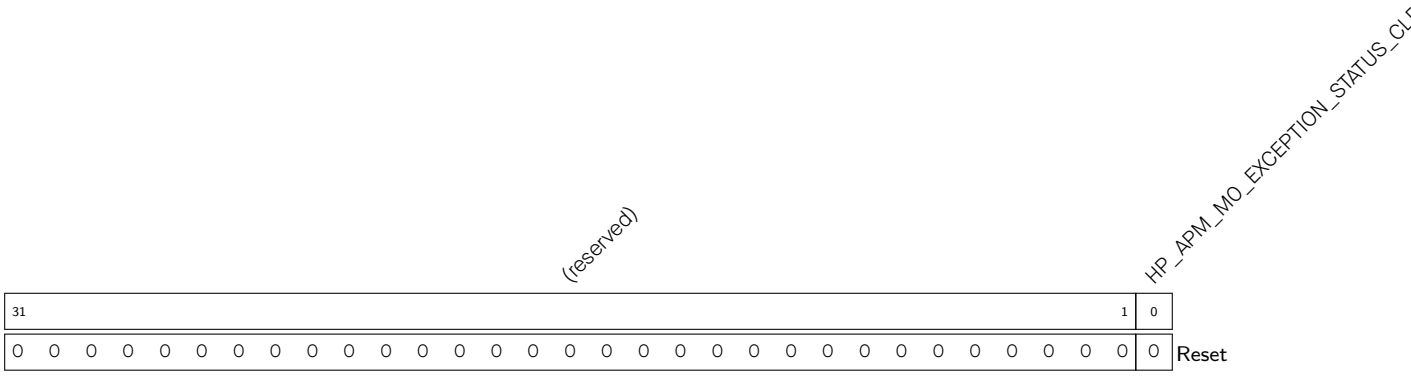
HP_APM_M0_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

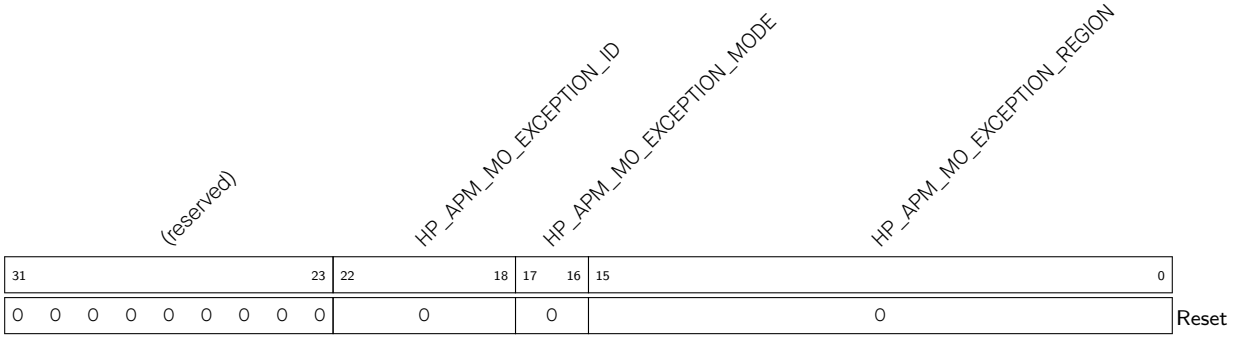
(RO)

Register 18.7. HP_APM_M0_STATUS_CLR_REG (0x00CC)



HP_APM_M0_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.8. HP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

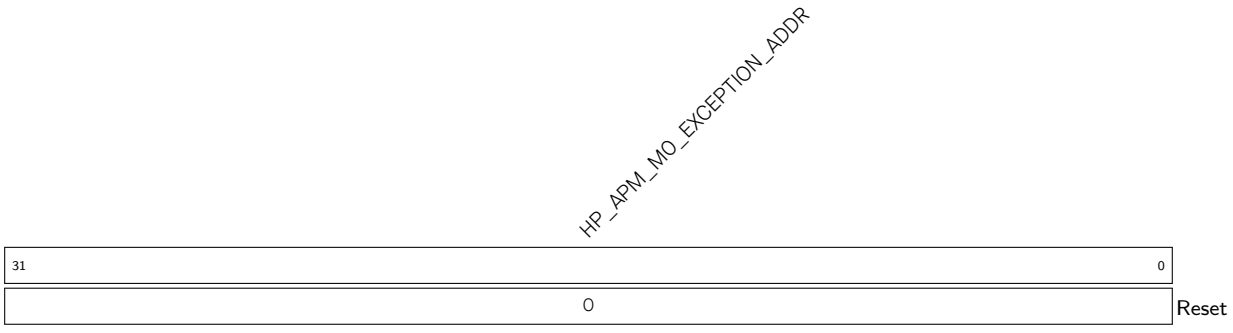


HP_APM_M0_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M0_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

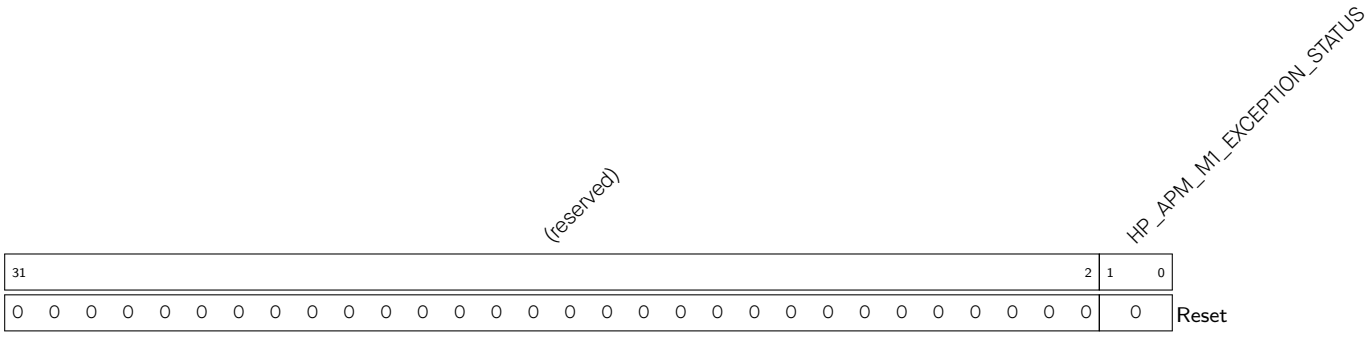
HP_APM_M0_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.9. HP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)



HP_APM_M0_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.10. HP_APM_M1_STATUS_REG (0x00D8)



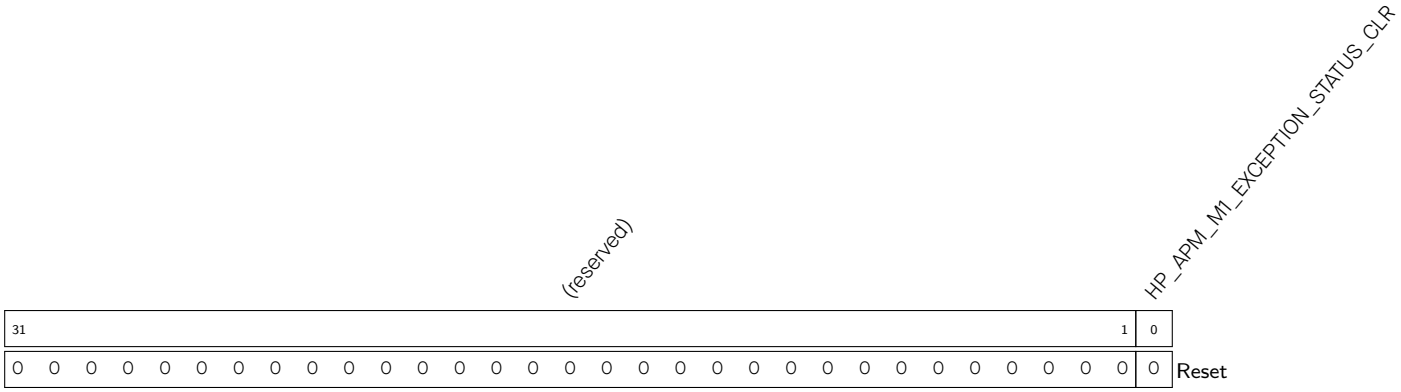
HP_APM_M1_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

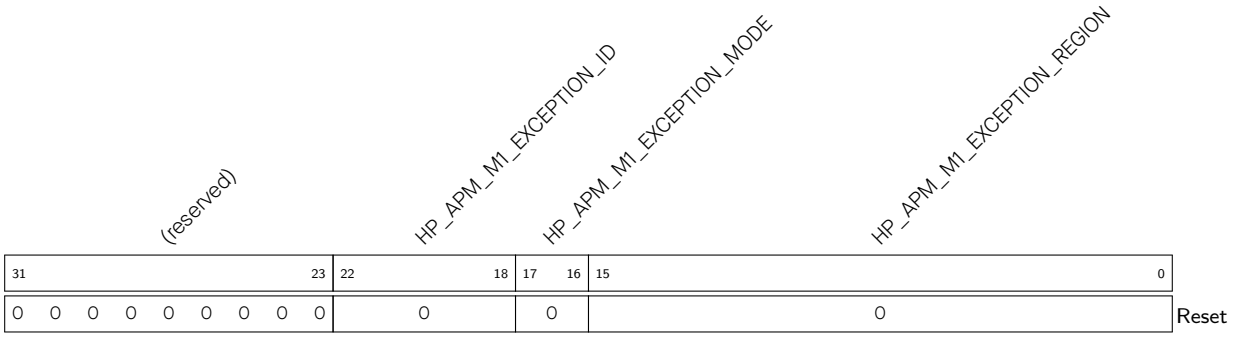
(RO)

Register 18.11. HP_APM_M1_STATUS_CLR_REG (0x00DC)



HP_APM_M1_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.12. HP_APM_M1_EXCEPTION_INFO0_REG (0x00E0)

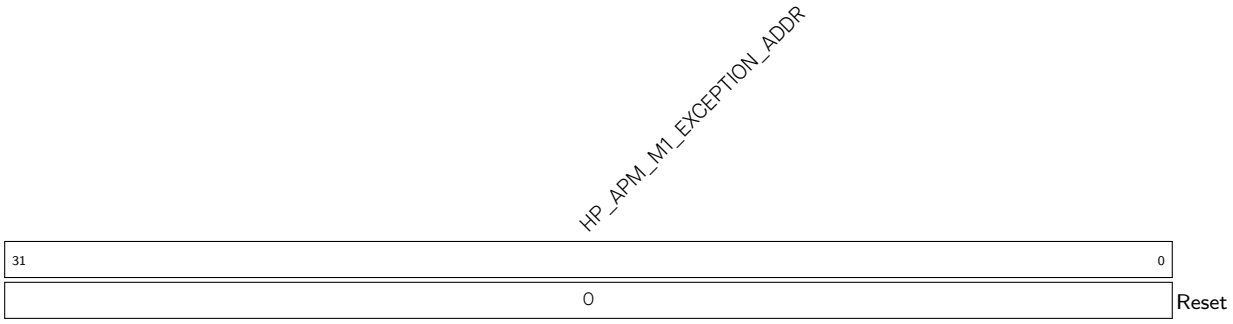


HP_APM_M1_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M1_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

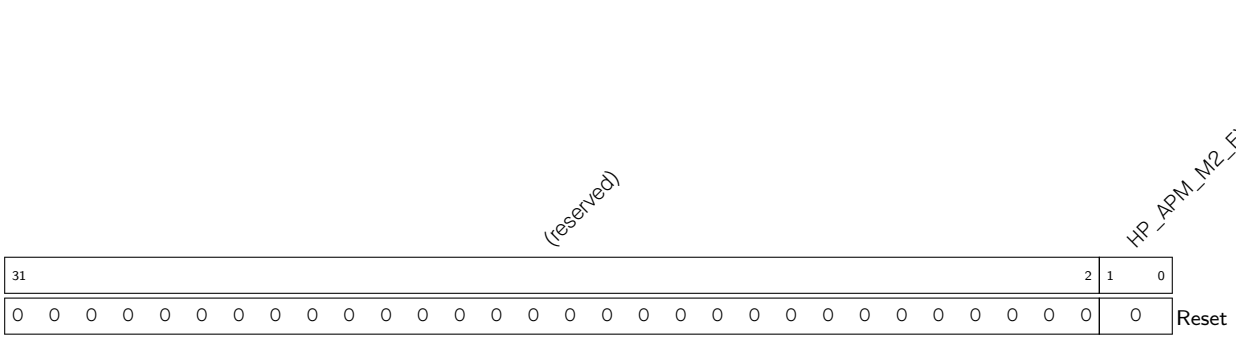
HP_APM_M1_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.13. HP_APM_M1_EXCEPTION_INFO1_REG (0x00E4)



HP_APM_M1_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.14. HP_APM_M2_STATUS_REG (0x00E8)



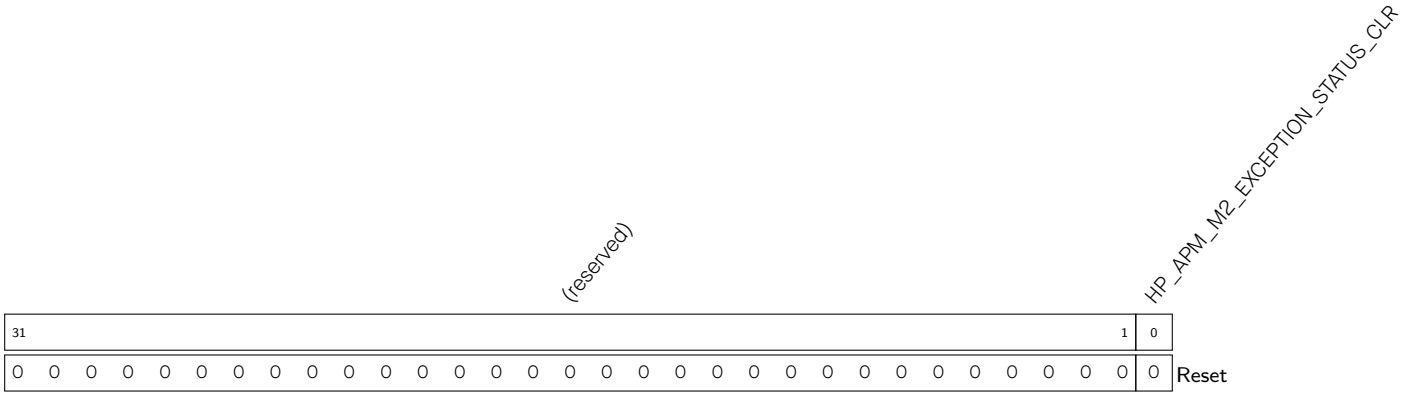
HP_APM_M2_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

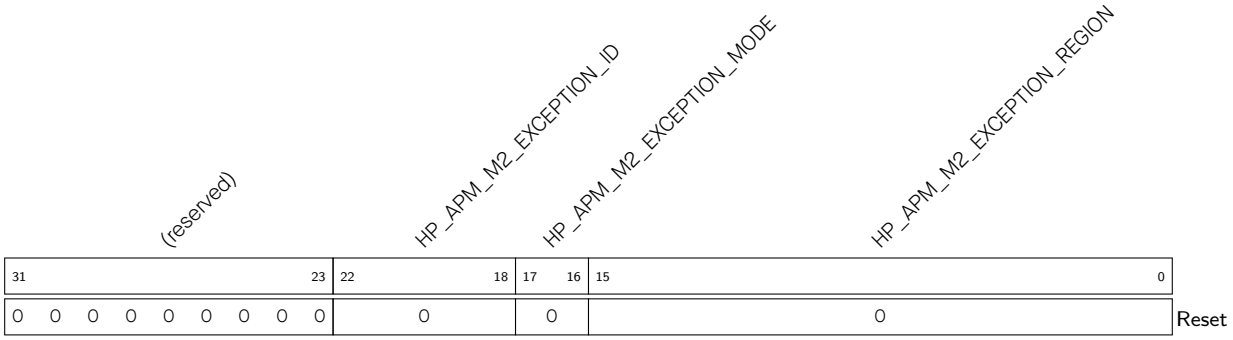
(RO)

Register 18.15. HP_APM_M2_STATUS_CLR_REG (0x00EC)



HP_APM_M2_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.16. HP_APM_M2_EXCEPTION_INFO0_REG (0x00F0)

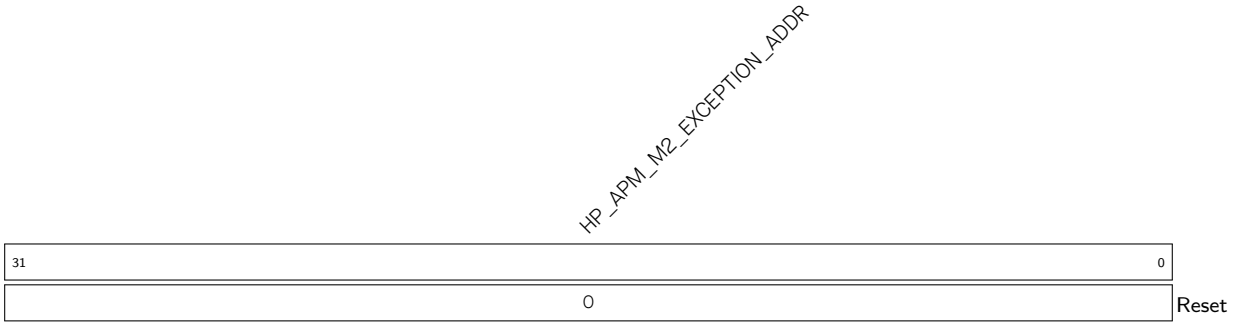


HP_APM_M2_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M2_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

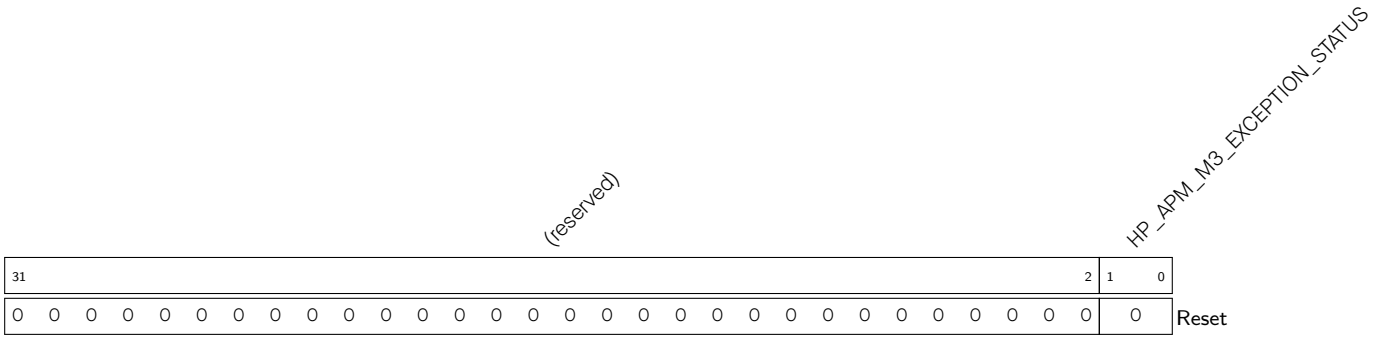
HP_APM_M2_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.17. HP_APM_M2_EXCEPTION_INFO1_REG (0x00F4)



HP_APM_M2_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.18. HP_APM_M3_STATUS_REG (0x00F8)



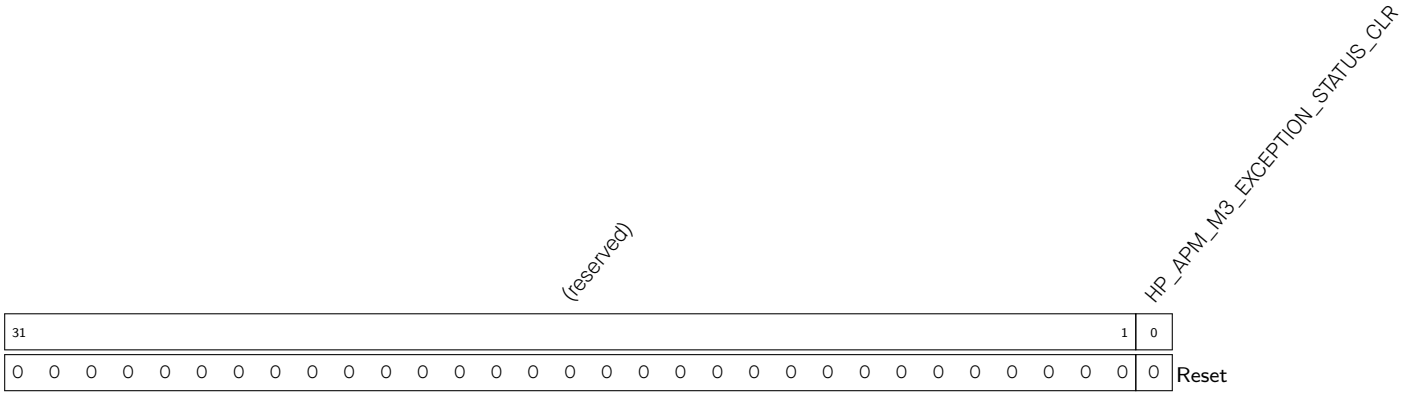
HP_APM_M3_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

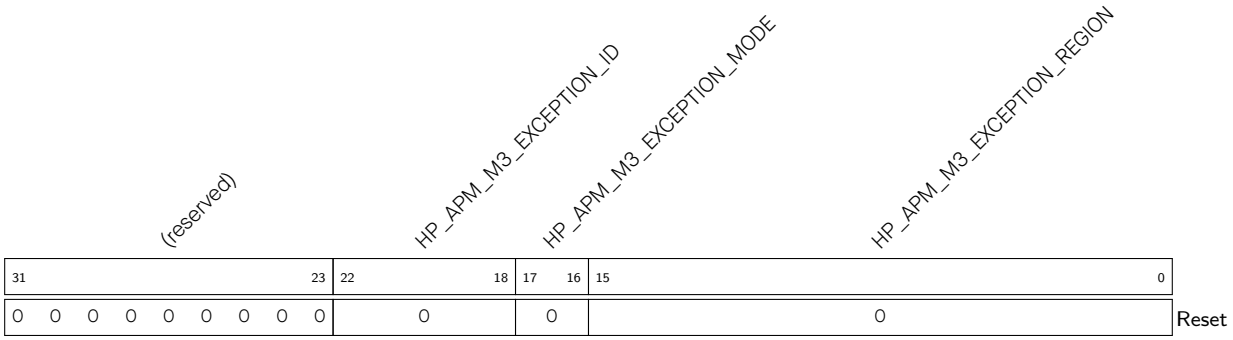
(RO)

Register 18.19. HP_APM_M3_STATUS_CLR_REG (0x00FC)



HP_APM_M3_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.20. HP_APM_M3_EXCEPTION_INFO0_REG (0x0100)

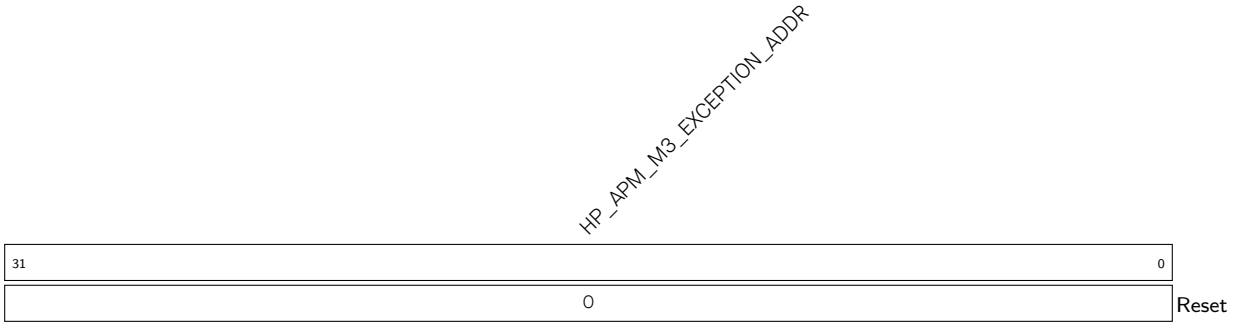


HP_APM_M3_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M3_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

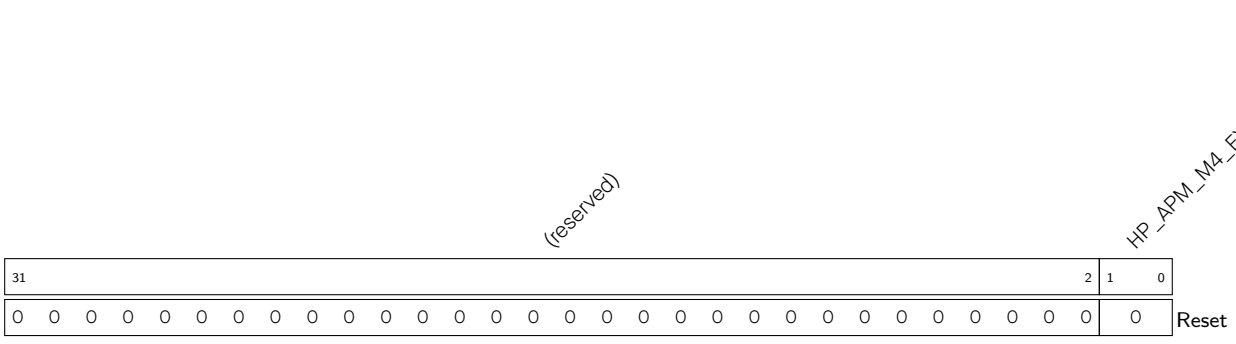
HP_APM_M3_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.21. HP_APM_M3_EXCEPTION_INFO1_REG (0x0104)



HP_APM_M3_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.22. HP_APM_M4_STATUS_REG (0x0108)



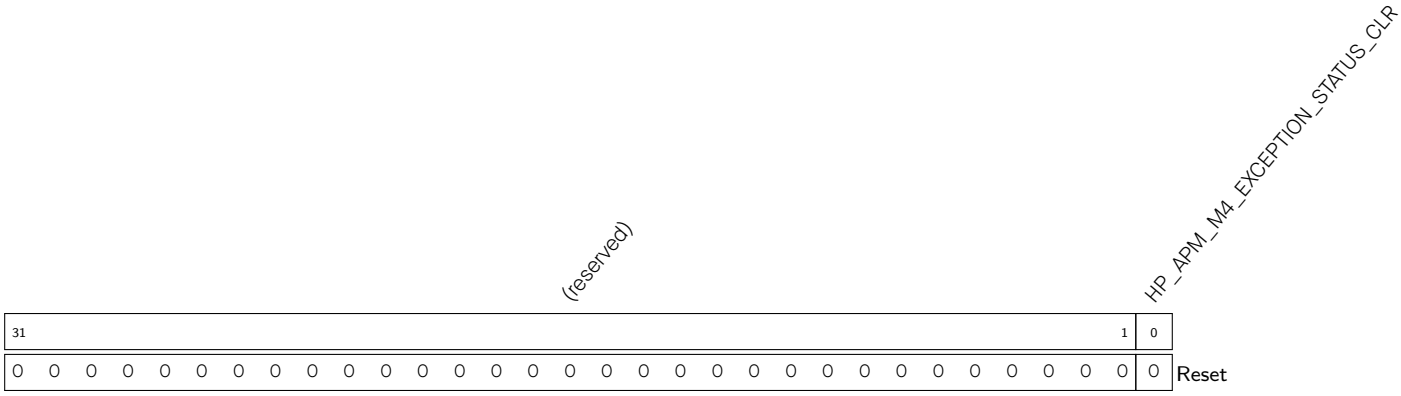
HP_APM_M4_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

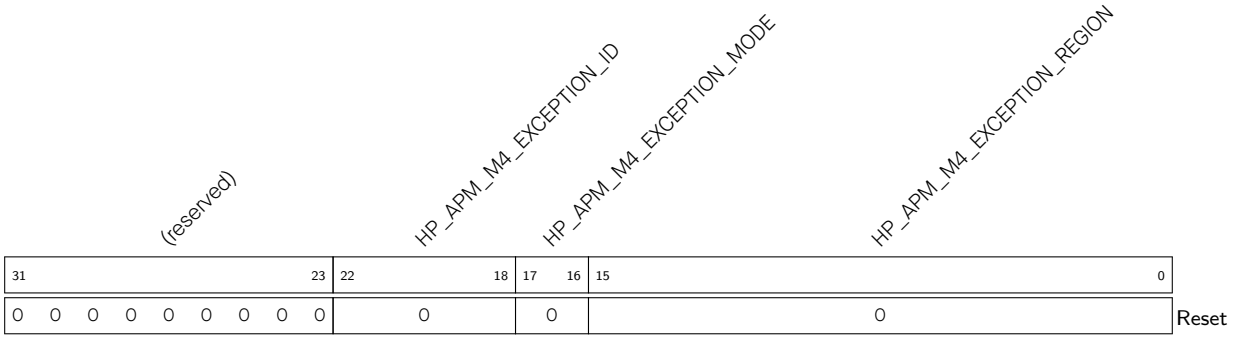
(RO)

Register 18.23. HP_APM_M4_STATUS_CLR_REG (0x010C)



HP_APM_M4_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.24. HP_APM_M4_EXCEPTION_INFO0_REG (0x0110)

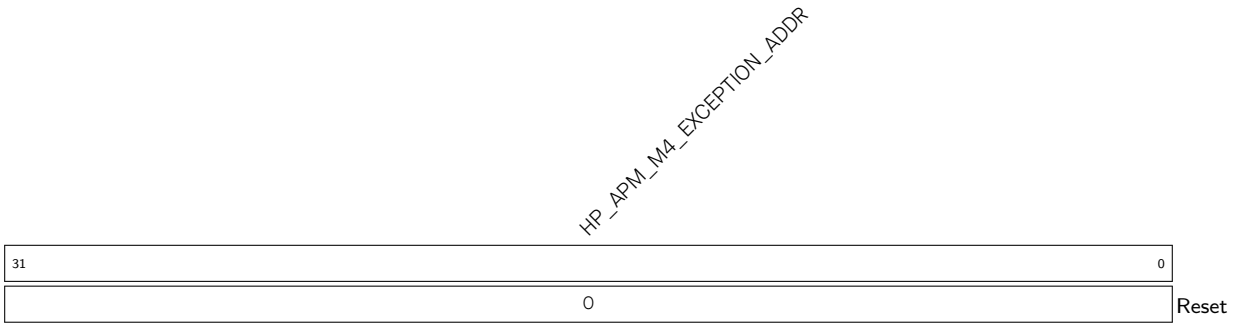


HP_APM_M4_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M4_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

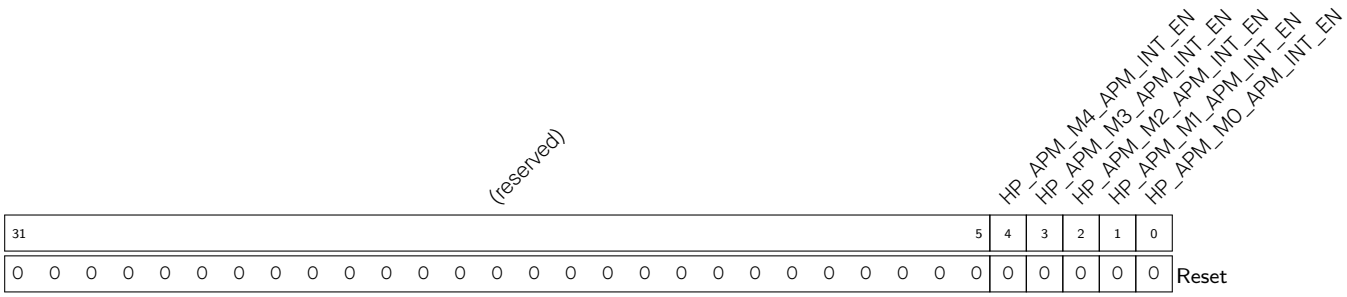
HP_APM_M4_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.25. HP_APM_M4_EXCEPTION_INFO1_REG (0x0114)



HP_APM_M4_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.26. HP_APM_INT_EN_REG (0x0118)



HP_APM_M0_APM_INT_EN 配置是否使能 HP_APM_CTRL M0 中断。

0：禁用

1：使能

(R/W)

HP_APM_M1_APM_INT_EN 配置是否使能 HP_APM_CTRL M1 中断。

0：禁用

1：使能

(R/W)

HP_APM_M2_APM_INT_EN 配置是否使能 HP_APM_CTRL M2 中断。

0：禁用

1：使能

(R/W)

HP_APM_M3_APM_INT_EN 配置是否使能 HP_APM_CTRL M3 中断。

0：禁用

1：使能

(R/W)

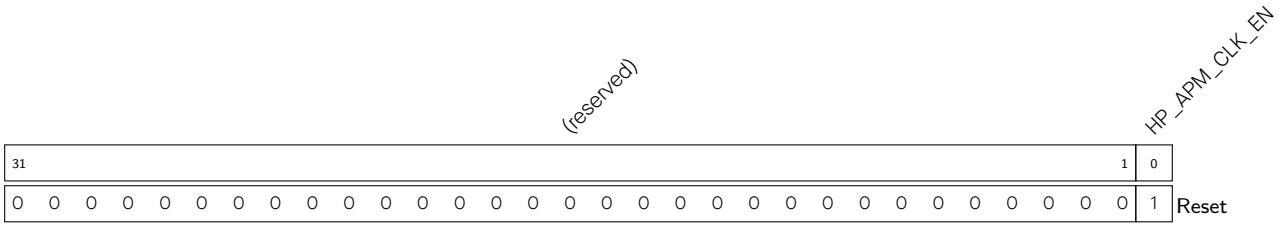
HP_APM_M4_APM_INT_EN 配置是否使能 HP_APM_CTRL M4 中断。

0：禁用

1：使能

(R/W)

Register 18.27. HP_APM_CLOCK_GATE_REG (0x07F8)



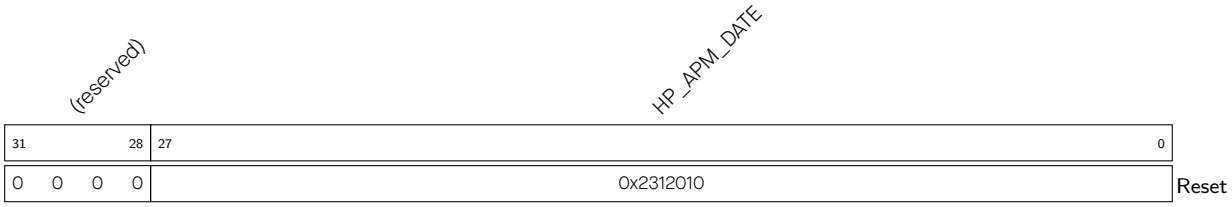
HP_APM_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 18.28. HP_APM_DATE_REG (0x07FC)



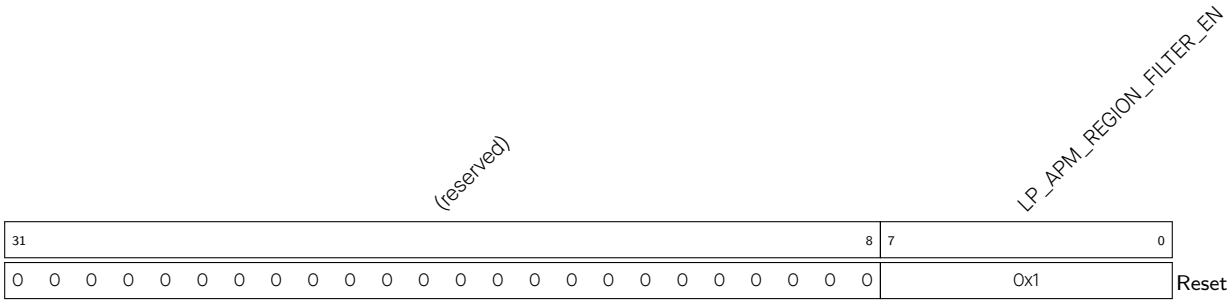
HP_APM_DATE 版本控制寄存器。(R/W)

18.9.2 LP_APM_REG

本小节的地址均为相对于 LP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 18.29. LP_APM_REGION_FILTER_EN_REG (0x0000)



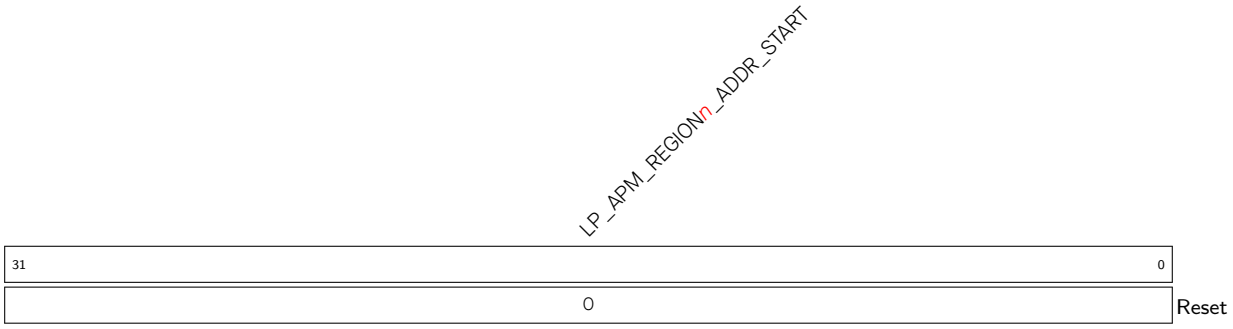
LP_APM_REGION_FILTER_EN 配置 bit *n* (0-7) 使能区域 *n* (0-7) 地址范围的权限检查。

0：禁用

1：使能

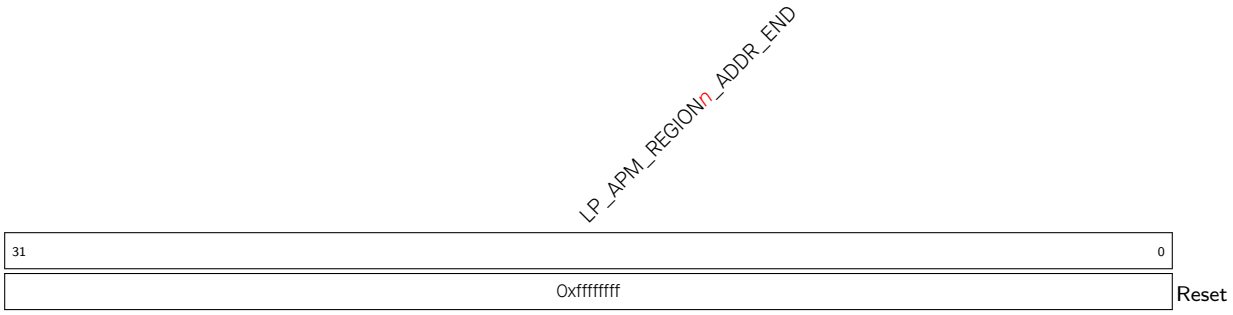
(R/W)

Register 18.30. LP_APM_REGION*n*_ADDR_START_REG (*n*: 0-7) (0x0004+0xC**n*)



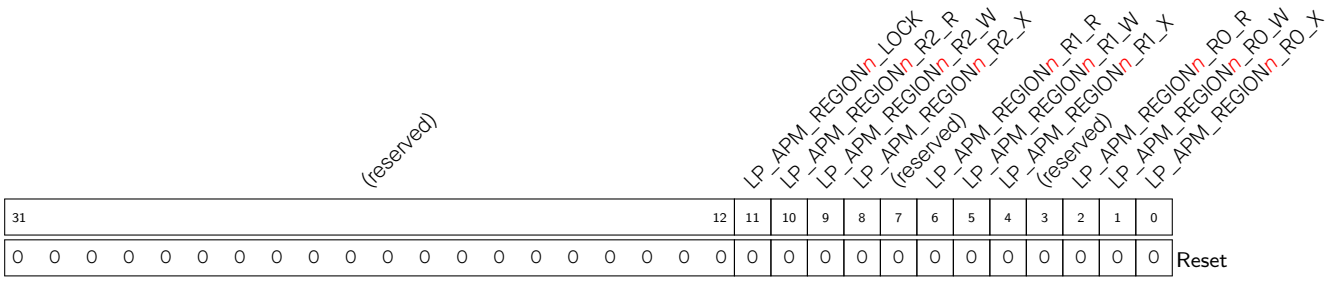
LP_APM_REGION*n*_ADDR_START 配置区域 *n* 的起始地址。(R/W)

Register 18.31. LP_APM_REGION*n*_ADDR_END_REG (*n*: 0-7) (0x0008+0xC**n*)



LP_APM_REGION*n*_ADDR_END 配置区域 *n* 的结束地址。(R/W)

Register 18.32. LP_APM_REGION*n*_ATTR_REG (*n*: 0-7) (0x000C+0xC**n*)



LP_APM_REGION*n*_R0_X 配置 REE0 模式下对区域 *n* 的执行权限。(R/W)

LP_APM_REGION*n*_R0_W 配置 REE0 模式下对区域 *n* 的写入权限。(R/W)

LP_APM_REGION*n*_R0_R 配置 REE0 模式下对区域 *n* 的读取权限。(R/W)

LP_APM_REGION*n*_R1_X 配置 REE1 模式下对区域 *n* 的执行权限。(R/W)

LP_APM_REGION*n*_R1_W 配置 REE1 模式下对区域 *n* 的写入权限。(R/W)

LP_APM_REGION*n*_R1_R 配置 REE1 模式下对区域 *n* 的读取权限。(R/W)

LP_APM_REGION*n*_R2_X 配置 REE2 模式下对区域 *n* 的执行权限。(R/W)

LP_APM_REGION*n*_R2_W 配置 REE2 模式下对区域 *n* 的写入权限。(R/W)

LP_APM_REGION*n*_R2_R 配置 REE2 模式下对区域 *n* 的读取权限。(R/W)

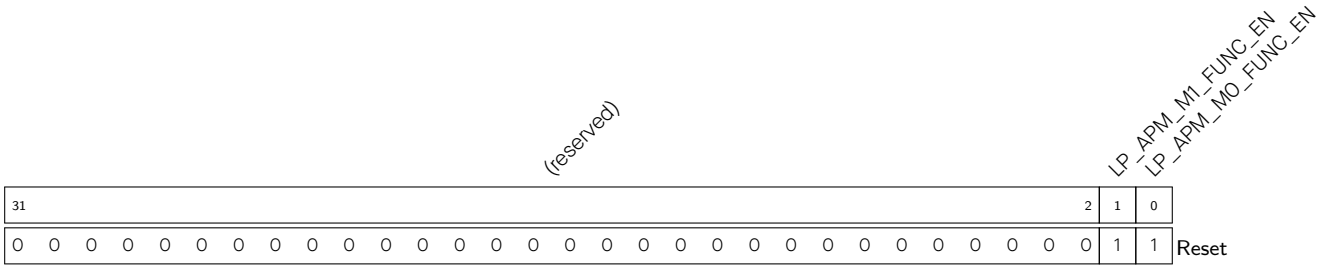
LP_APM_REGION*n*_LOCK 锁定区域 *n* 的首尾地址及权限的配置 (LP_APM_REGION*n*_ADDR_START_REG、LP_APM_REGION*n*_ADDR_END_REG 和 LP_APM_REGION*n*_ATTR_REG)。

0：不锁定

1：锁定

(R/W)

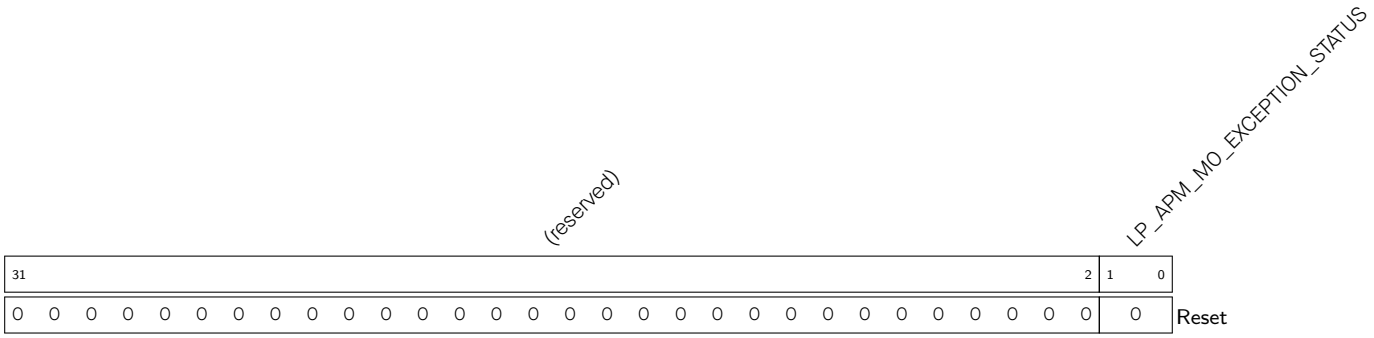
Register 18.33. LP_APM_FUNC_CTRL_REG (0x00C4)



LP_APM_M0_FUNC_EN 配置使能 LP_APM_CTRL M0 权限检查。(R/W)

LP_APM_M1_FUNC_EN 配置使能 LP_APM_CTRL M1 权限检查。(R/W)

Register 18.34. LP_APM_M0_STATUS_REG (0x00C8)



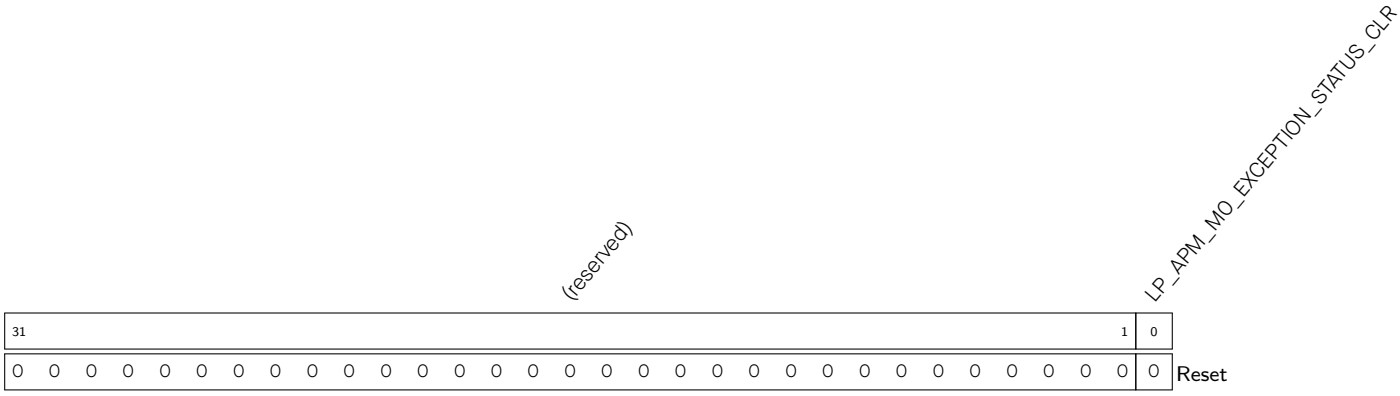
LP_APM_M0_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

(RO)

Register 18.35. LP_APM_M0_STATUS_CLR_REG (0x00CC)



LP_APM_M0_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.36. LP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

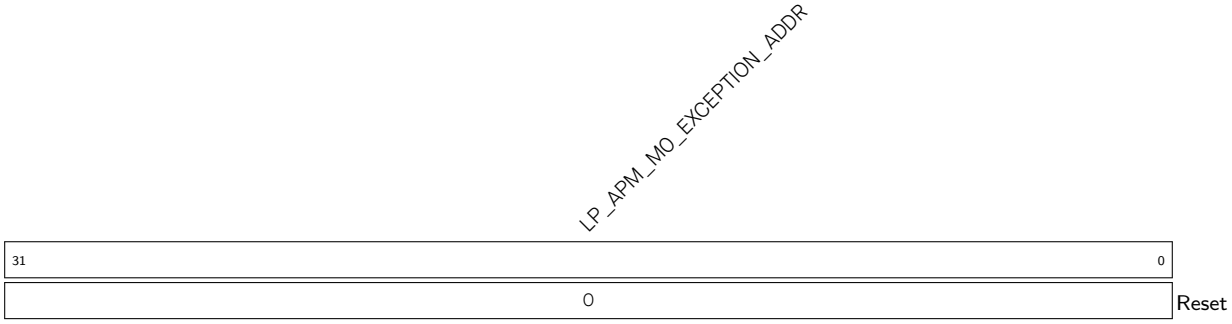


LP_APM_M0_EXCEPTION_REGION 表示异常区域。(RO)

LP_APM_M0_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

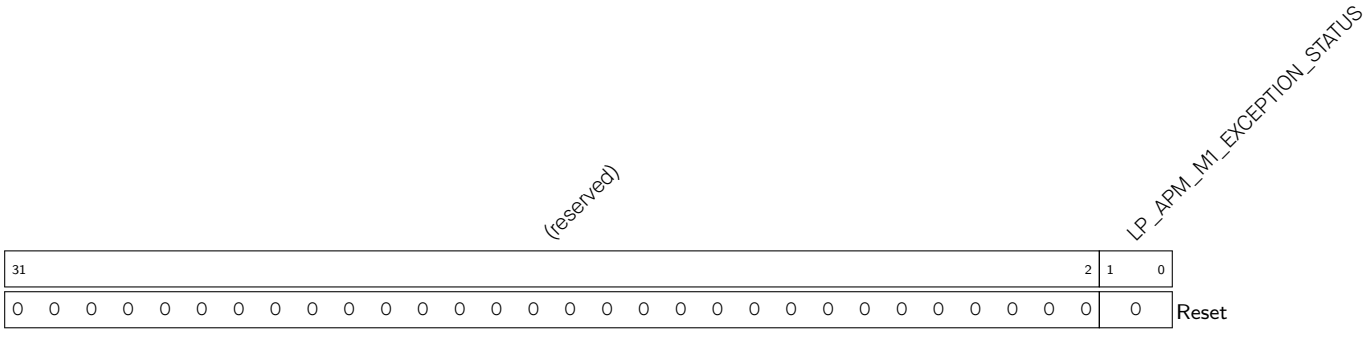
LP_APM_M0_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.37. LP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)



LP_APM_M0_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.38. LP_APM_M1_STATUS_REG (0x00D8)



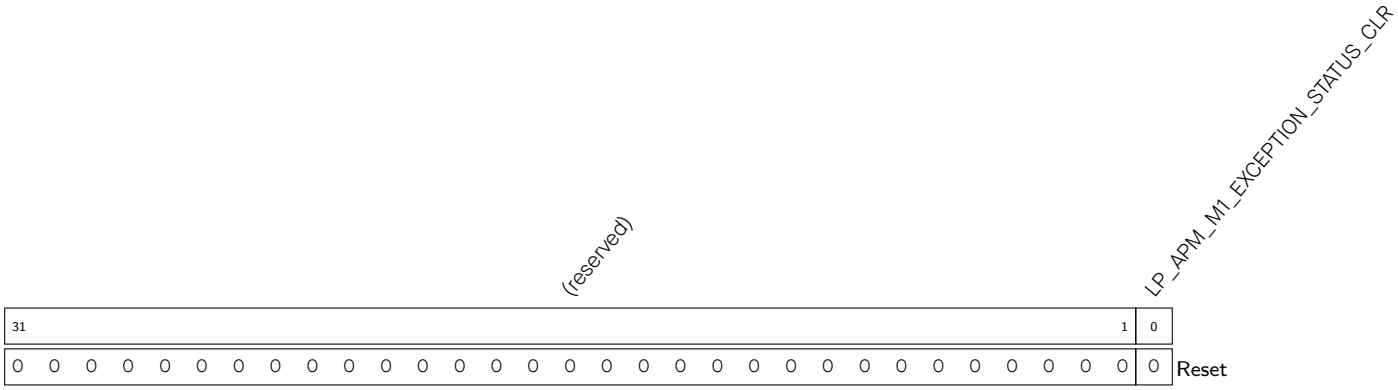
LP_APM_M1_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

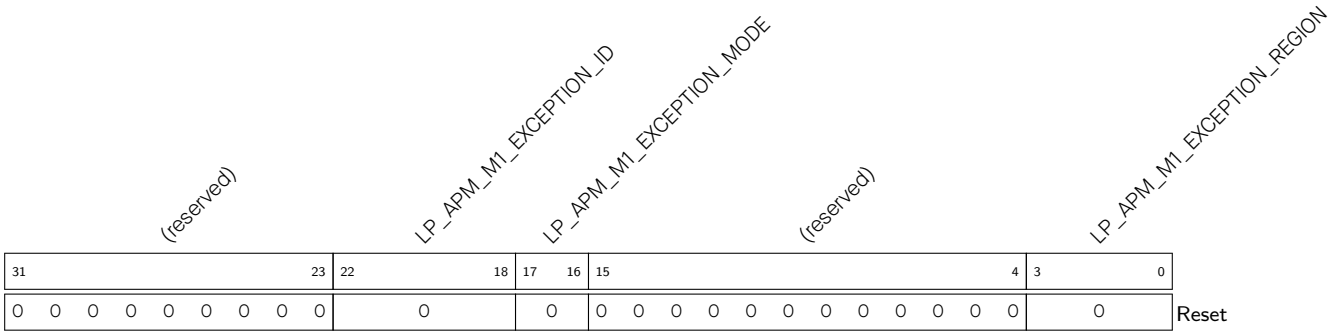
(RO)

Register 18.39. LP_APM_M1_STATUS_CLR_REG (0x00DC)



LP_APM_M1_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.40. LP_APM_M1_EXCEPTION_INFO0_REG (0x00E0)

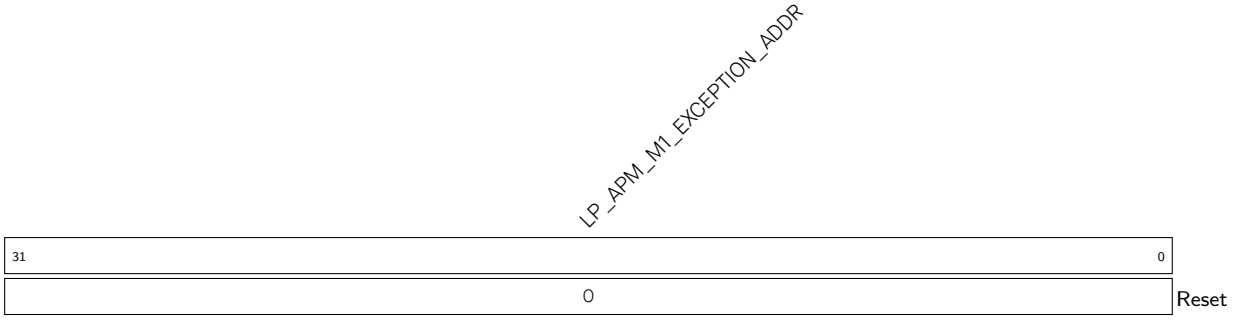


LP_APM_M1_EXCEPTION_REGION 表示异常区域。(RO)

LP_APM_M1_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

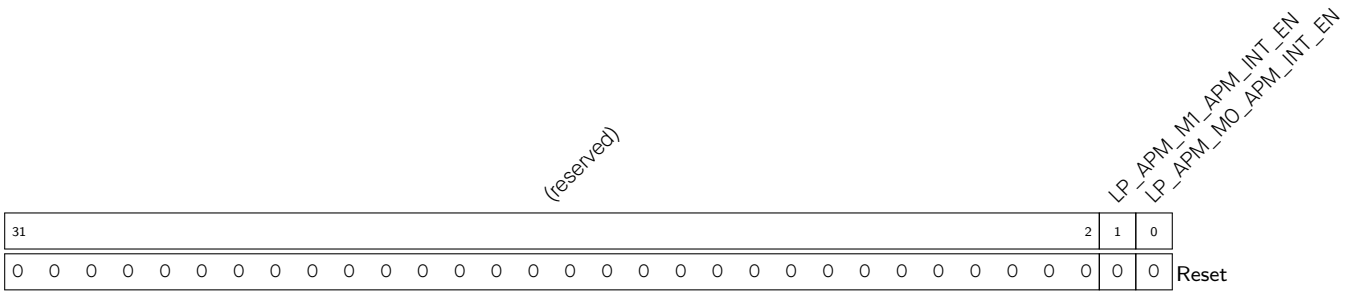
LP_APM_M1_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.41. LP_APM_M1_EXCEPTION_INFO1_REG (0x00E4)



LP_APM_M1_EXCEPTION_ADDR 表示异常地址。(RO)

Register 18.42. LP_APM_INT_EN_REG (0x00E8)



LP_APM_M0_APM_INT_EN 配置使能 LP_APM_CTRL M0 中断。

0：禁用

1：使能

(R/W)

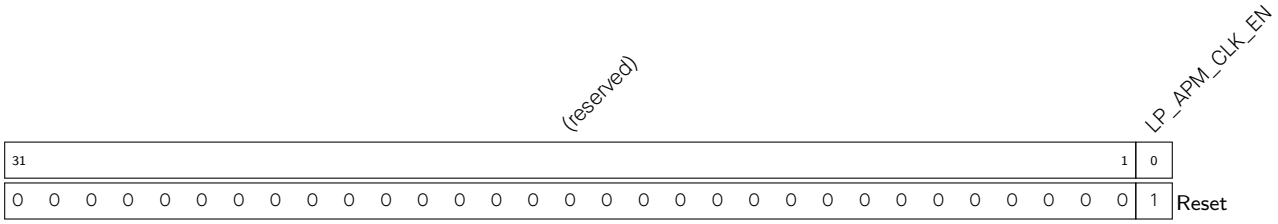
LP_APM_M1_APM_INT_EN 配置使能 LP_APM_CTRL M1 中断。

0：禁用

1：使能

(R/W)

Register 18.43. LP_APM_CLOCK_GATE_REG (0x00EC)



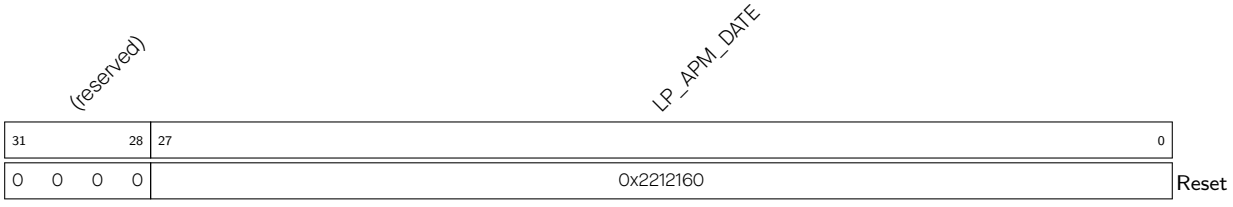
LP_APM_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 18.44. LP_APM_DATE_REG (0x00FC)



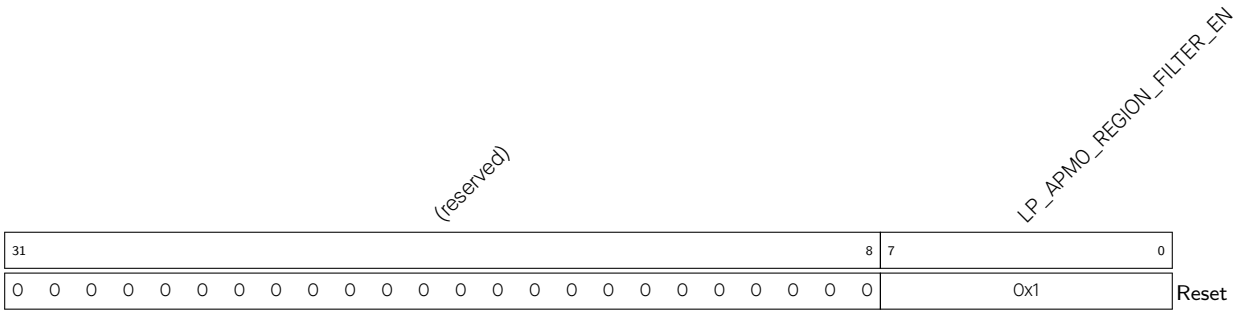
LP_APM_DATE 版本控制寄存器。(R/W)

18.9.3 LP_APM0_REG

本小节的地址均为相对于 LP_APM0 的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 18.45. LP_APM0_REGION_FILTER_EN_REG (0x0000)



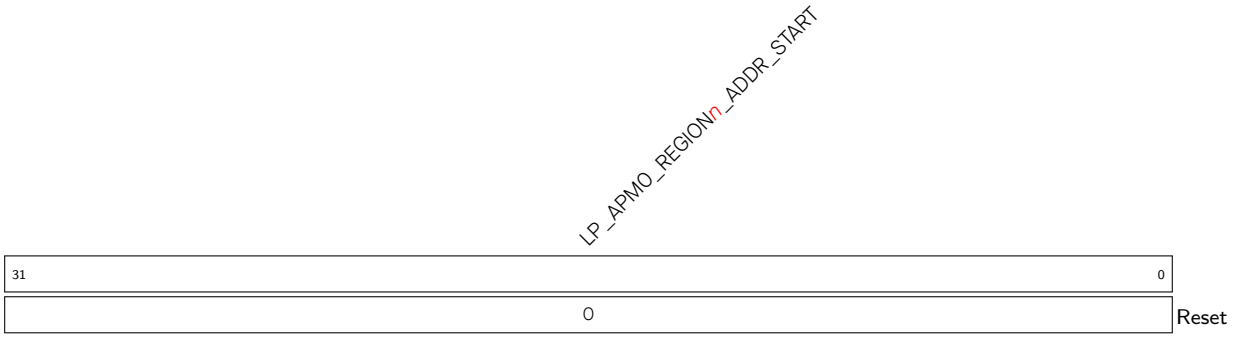
LP_APM0_REGION_FILTER_EN 配置 bit *n* (0-7) 使能区域 *n* (0-7) 地址范围的权限检查。

0：禁用

1：使能

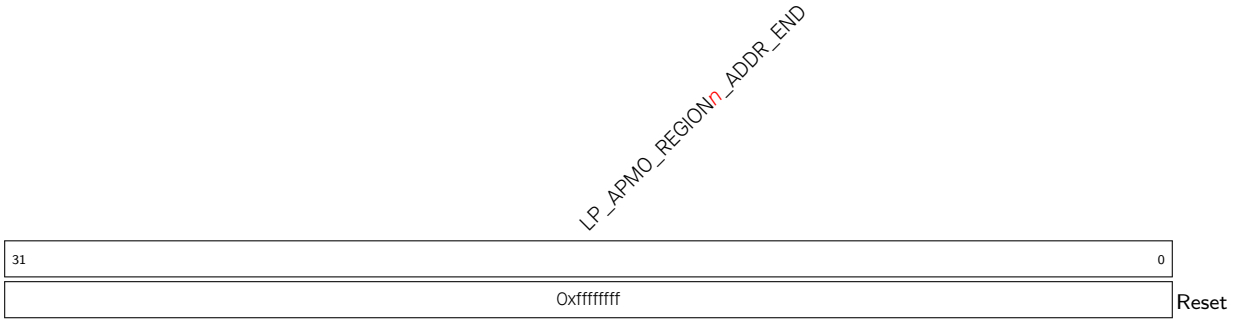
(R/W)

Register 18.46. LP_APM0_REGION*n*_ADDR_START_REG (*n*: 0-7) (0x0004+0xC**n*)



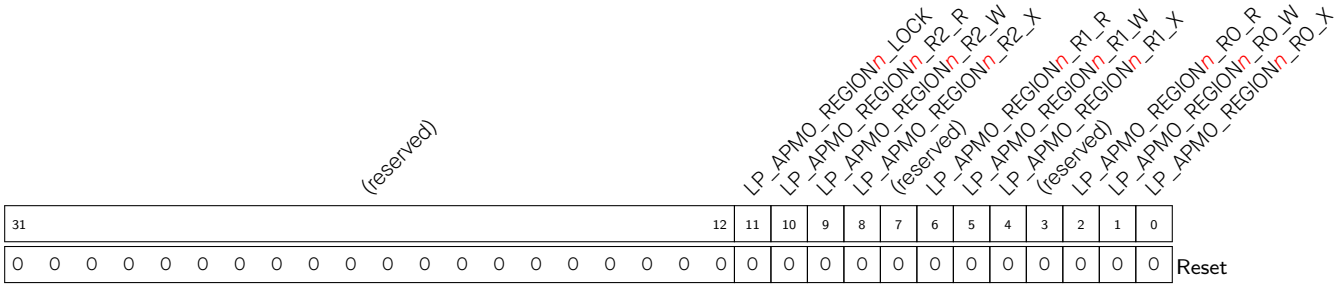
LP_APM0_REGION*n*_ADDR_START 配置区域 *n* 的起始地址。(R/W)

Register 18.47. LP_APM0_REGION*n*_ADDR_END_REG (*n*: 0-7) (0x0008+0xC**n*)



LP_APM0_REGION*n*_ADDR_END 配置区域 *n* 的结束地址。(R/W)

Register 18.48. LP_APM0_REGION*n*_ATTR_REG (*n*: 0-7) (0x000C+0xC**n*)



LP_APM0_REGION*n*_R0_X 配置 REE0 模式下对区域 *n* 的执行权限。(R/W)

LP_APM0_REGION*n*_R0_W 配置 REE0 模式下对区域 *n* 的写入权限。(R/W)

LP_APM0_REGION*n*_R0_R 配置 REE0 模式下对区域 *n* 的读取权限。(R/W)

LP_APM0_REGION*n*_R1_X 配置 REE1 模式下对区域 *n* 的执行权限。(R/W)

LP_APM0_REGION*n*_R1_W 配置 REE1 模式下对区域 *n* 的写入权限。(R/W)

LP_APM0_REGION*n*_R1_R 配置 REE1 模式下对区域 *n* 的读取权限。(R/W)

LP_APM0_REGION*n*_R2_X 配置 REE2 模式下对区域 *n* 的执行权限。(R/W)

LP_APM0_REGION*n*_R2_W 配置 REE2 模式下对区域 *n* 的写入权限。(R/W)

LP_APM0_REGION*n*_R2_R 配置 REE2 模式下对区域 *n* 的读取权限。(R/W)

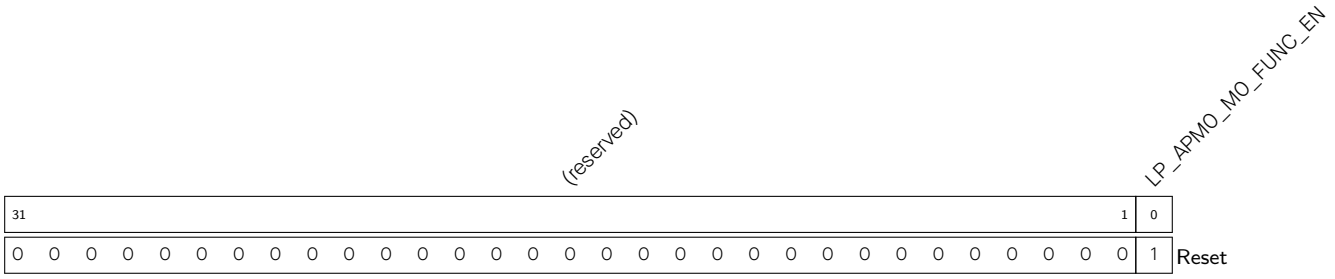
LP_APM0_REGION*n*_LOCK 锁定区域 *n* 的首尾地址及权限的配置 (LP_APM0_REGION*n*_ADDR_START_REG、LP_APM0_REGION*n*_ADDR_END_REG 和 LP_APM0_REGION*n*_ATTR_REG)。

0：不锁定

1：锁定

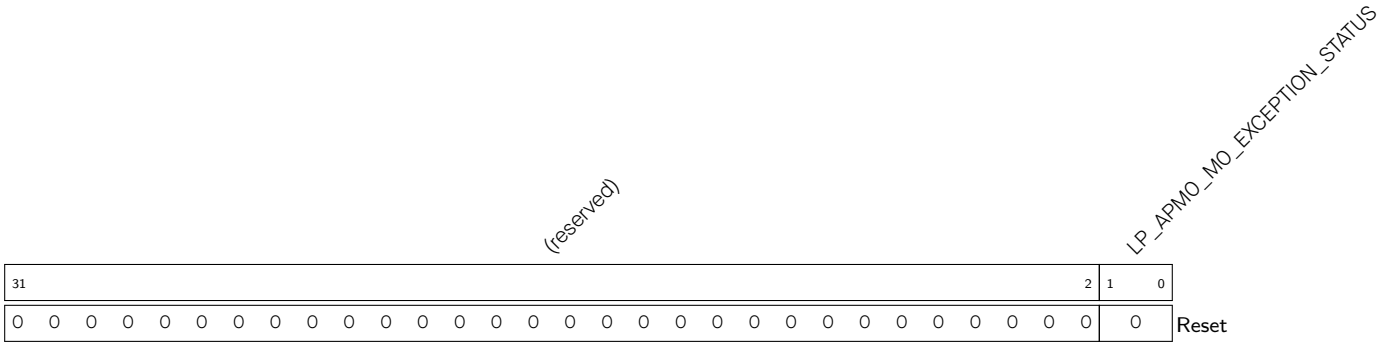
(R/W)

Register 18.49. LP_APM0_FUNC_CTRL_REG (0x00C4)



LP_APM0_M0_FUNC_EN 配置使能 LP_APM0_CTRL M0 权限检查。(R/W)

Register 18.50. LP_APM0_M0_STATUS_REG (0x00C8)



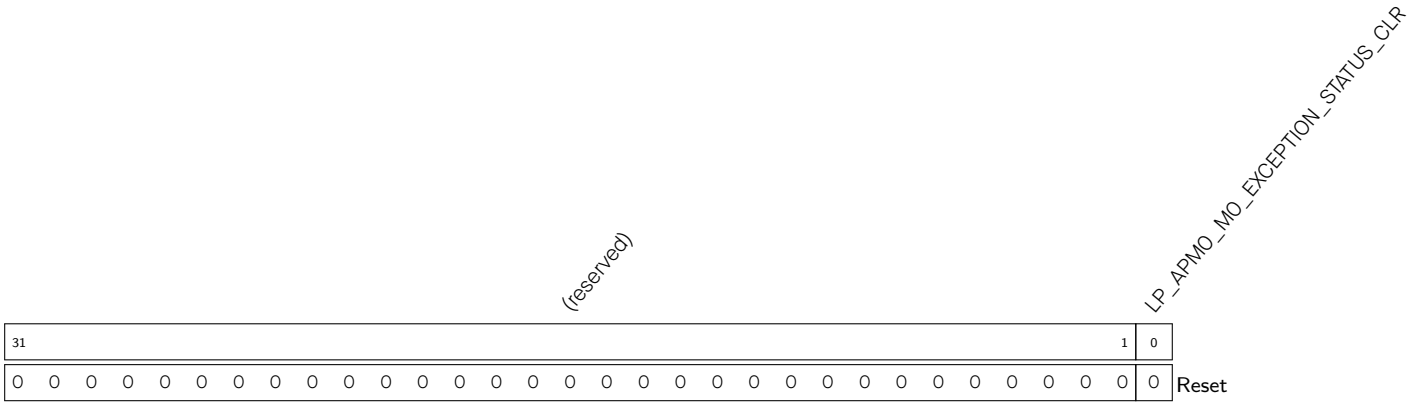
LP_APM0_M0_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

(RO)

Register 18.51. LP_APM0_M0_STATUS_CLR_REG (0x00CC)



LP_APM0_M0_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 18.52. LP_APM0_M0_EXCEPTION_INFO0_REG (0x00D0)

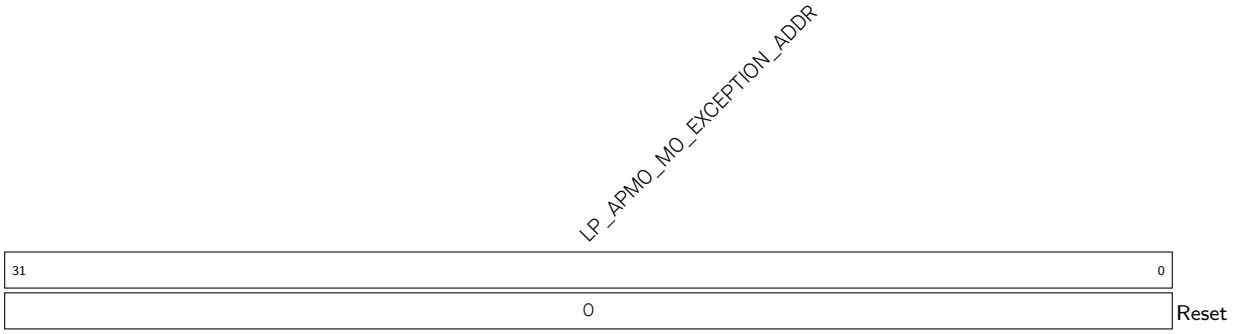


LP_APM0_M0_EXCEPTION_REGION 表示异常区域。(RO)

LP_APM0_M0_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

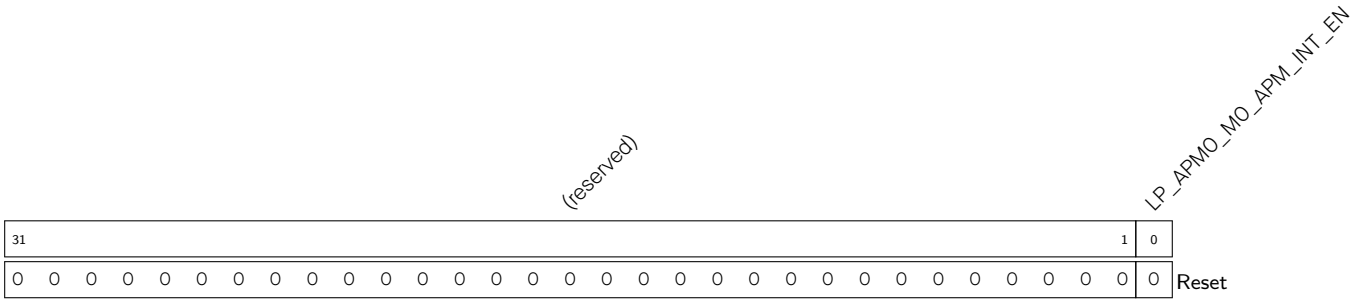
LP_APM0_M0_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.53. LP_APM0_M0_EXCEPTION_INFO1_REG (0x00D4)



LP_APM0_M0_EXCEPTION_ADDR 表示异常地址。(RO)

Register 18.54. LP_APM0_INT_EN_REG (0x00D8)



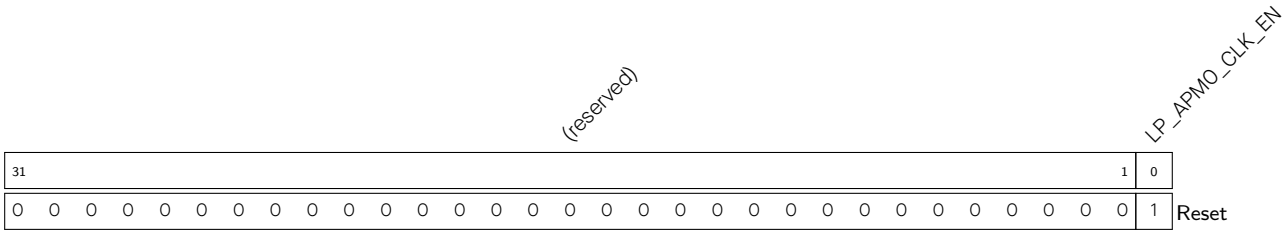
LP_APM0_M0_APM_INT_EN 配置是否使能 LP_APM0_CTRL M0 中断。

0：禁用

1：使能

(R/W)

Register 18.55. LP_APM0_CLOCK_GATE_REG (0x00DC)



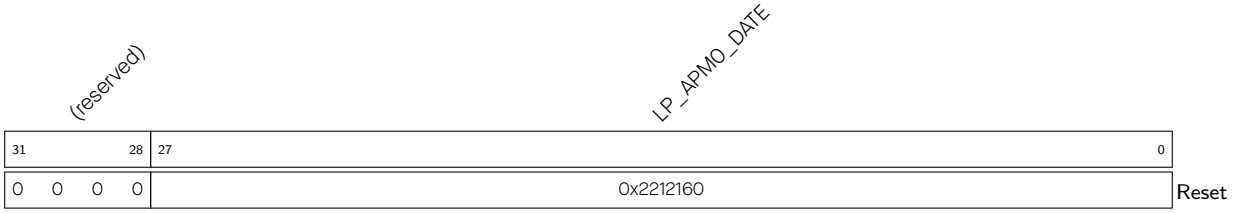
LP_APM0_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 18.56. LP_APM0_DATE_REG (0x07FC)



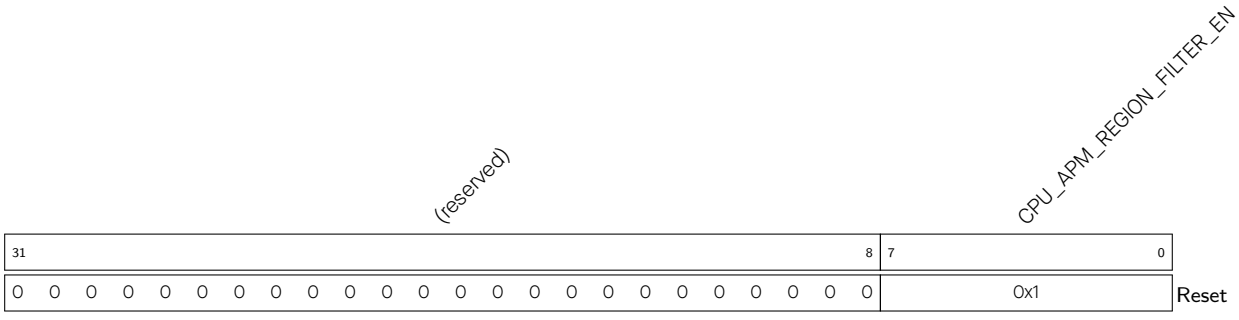
LP_APM0_DATE 版本控制寄存器。(R/W)

18.9.4 CPU_APM_REG

本小节的地址均为相对于 CPU_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

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

Register 18.57. CPU_APM_REGION_FILTER_EN_REG (0x0000)



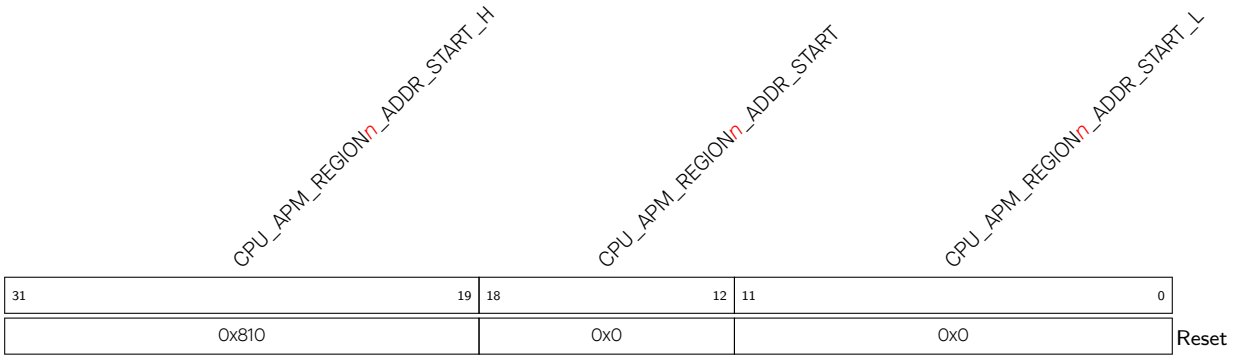
CPU_APM_REGION_FILTER_EN 配置 bit *n* (0-7) 使能区域 *n* (0-7) 地址范围的权限检查。

0：禁用

1：使能

(R/W)

Register 18.58. CPU_APM_REGION*n*_ADDR_START_REG (*n*: 0-7) (0x0004+0xC**n*)



CPU_APM_REGION*n*_ADDR_START_L 起始地址的低 12 位。(HRO)

CPU_APM_REGION*n*_ADDR_START 配置区域 *n* 的起始地址。(R/W)

CPU_APM_REGION*n*_ADDR_START_H 表示区域 *n* 起始地址的高 13 位。(HRO)

Register 18.59. CPU_APM_REGION*n*_ADDR_END_REG (*n*: 0-7) (0x0008+0xC**n*)

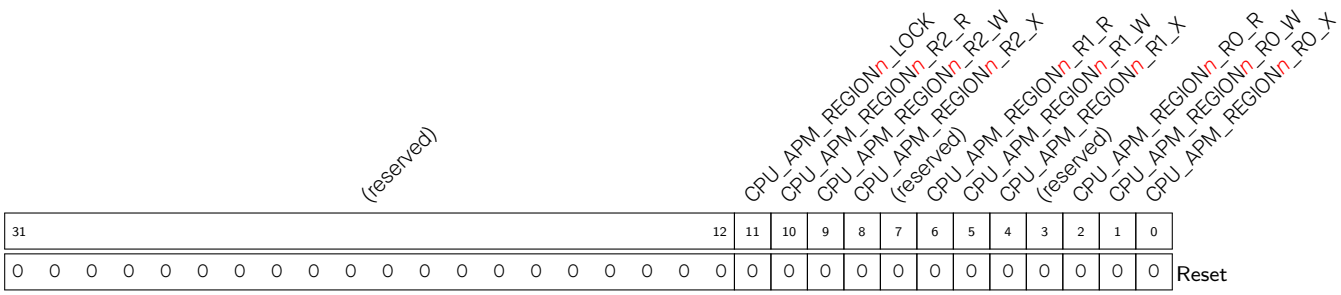


CPU_APM_REGION*n*_ADDR_END_L 表示区域 *n* 结束地址的低 12 位。(HRO)

CPU_APM_REGION*n*_ADDR_END 配置区域 *n* 的结束地址。(R/W)

CPU_APM_REGION*n*_ADDR_END_H 表示区域 *n* 结束地址的高 13 位。(HRO)

Register 18.60. CPU_APM_REGIONn_ATTR_REG (n: 0-7) (0x000C+0xC*n)



CPU_APM_REGIONn_R0_X 配置 REE0 模式下对区域 n 的执行权限。(R/W)

CPU_APM_REGIONn_R0_W 配置 REE0 模式下对区域 n 的写入权限。(R/W)

CPU_APM_REGIONn_R0_R 配置 REE0 模式下对区域 n 的读取权限。(R/W)

CPU_APM_REGIONn_R1_X 配置 REE1 模式下对区域 n 的执行权限。(R/W)

CPU_APM_REGIONn_R1_W 配置 REE1 模式下对区域 n 的写入权限。(R/W)

CPU_APM_REGIONn_R1_R 配置 REE1 模式下对区域 n 的读取权限。(R/W)

CPU_APM_REGIONn_R2_X 配置 REE2 模式下对区域 n 的执行权限。(R/W)

CPU_APM_REGIONn_R2_W 配置 REE2 模式下对区域 n 的写入权限。(R/W)

CPU_APM_REGIONn_R2_R 配置 REE2 模式下对区域 n 的读取权限。(R/W)

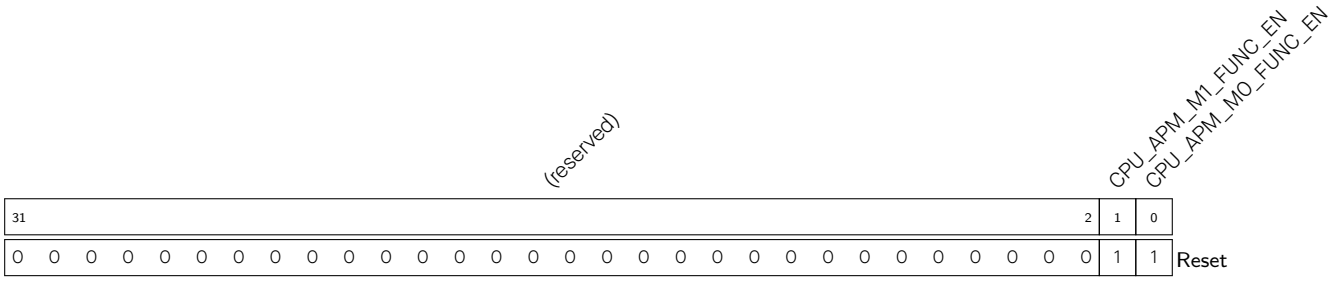
CPU_APM_REGIONn_LOCK 锁定区域 n 的首尾地址及权限的配置 (CPU_APM_REGIONn_ADDR_START_REG、CPU_APM_REGIONn_ADDR_END_REG、和 CPU_APM_REGIONn_ATTR_REG)。

0：不锁定

1：锁定

(R/W)

Register 18.61. CPU_APM_FUNC_CTRL_REG (0x00C4)



CPU_APM_M0_FUNC_EN 配置使能 CPU_APM_CTRL M0 权限检查。

0：禁用

1：使能

(R/W)

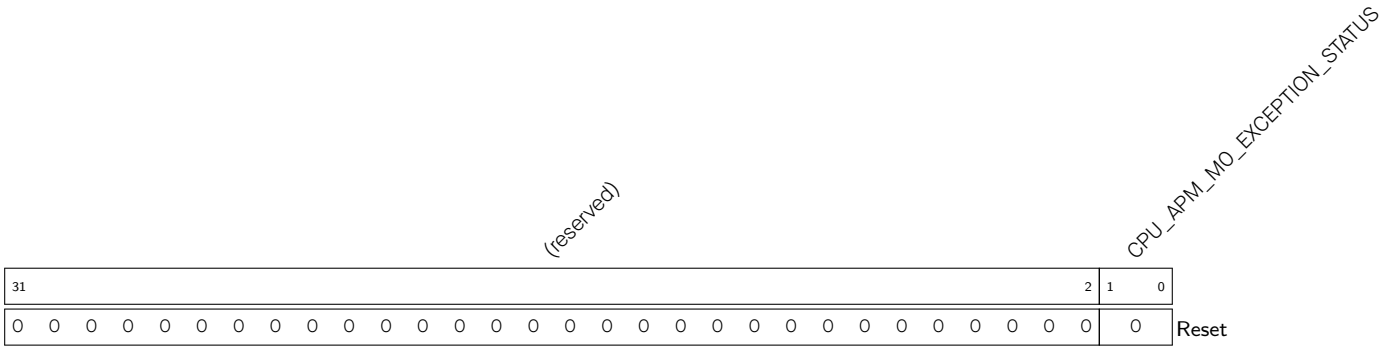
CPU_APM_M1_FUNC_EN 配置使能 CPU_APM_CTRL M1 权限检查。

0：禁用

1：使能

(R/W)

Register 18.62. CPU_APM_M0_STATUS_REG (0x00C8)



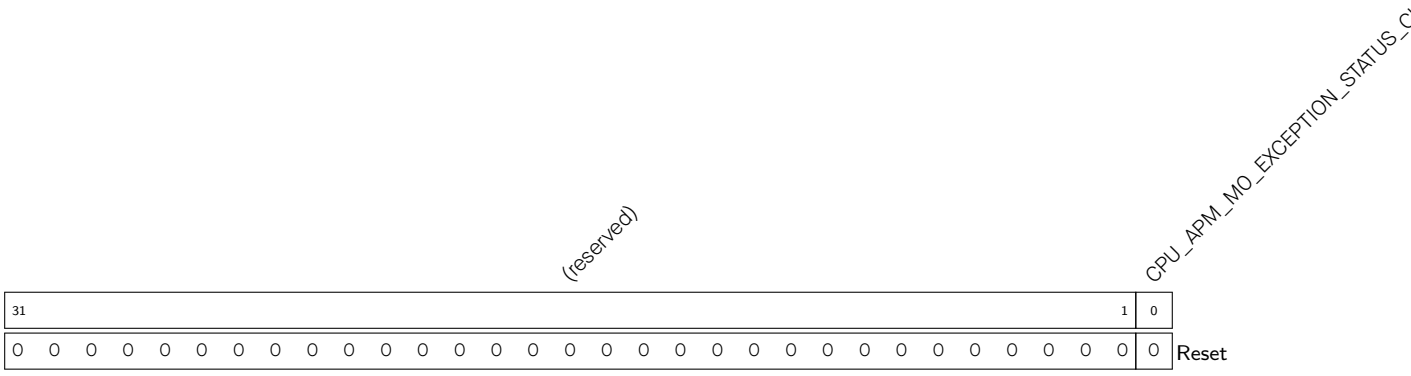
CPU_APM_M0_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

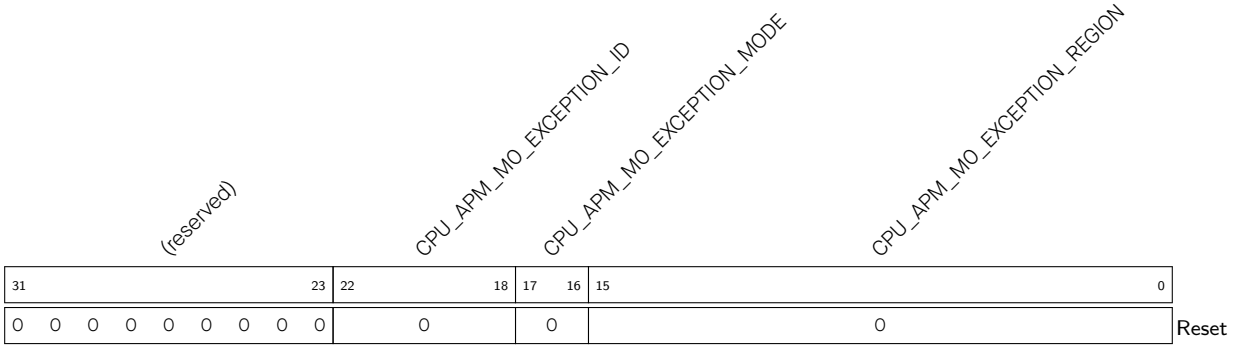
(RO)

Register 18.63. CPU_APM_M0_STATUS_CLR_REG (0x00CC)



CPU_APM_M0_EXCEPTION_STATUS_CLR 写 1 清除异常状态。(WT)

Register 18.64. CPU_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

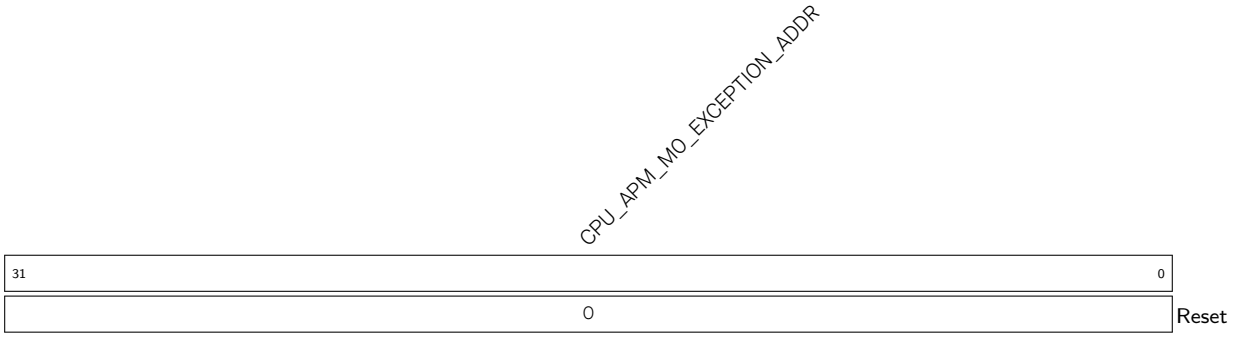


CPU_APM_M0_EXCEPTION_REGION 表示异常区域。(RO)

CPU_APM_M0_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

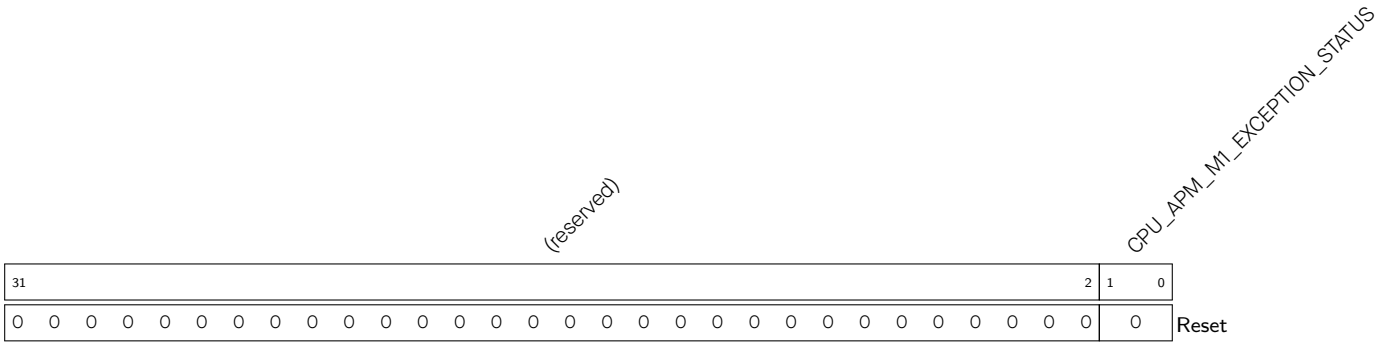
CPU_APM_M0_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.65. CPU_APM_M0_EXCEPTION_INFO1_REG (0x00D4)



CPU_APM_M0_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.66. CPU_APM_M1_STATUS_REG (0x00D8)



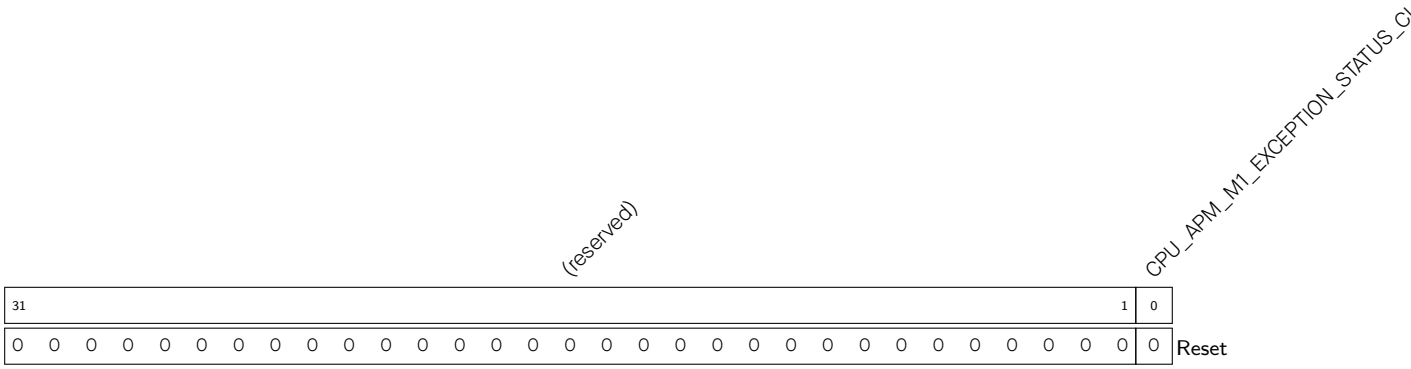
CPU_APM_M1_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

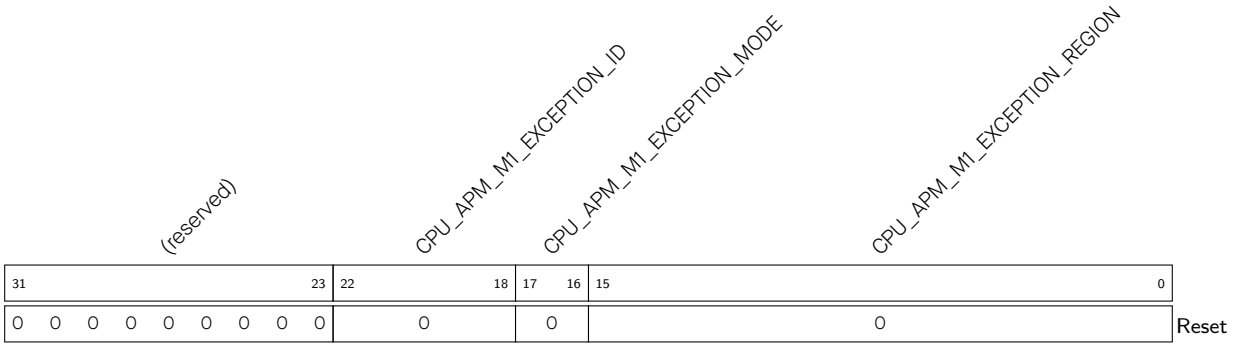
(RO)

Register 18.67. CPU_APM_M1_STATUS_CLR_REG (0x00DC)



CPU_APM_M1_EXCEPTION_STATUS_CLR 写 1 清除异常状态。(WT)

Register 18.68. CPU_APM_M1_EXCEPTION_INFO0_REG (0x00E0)

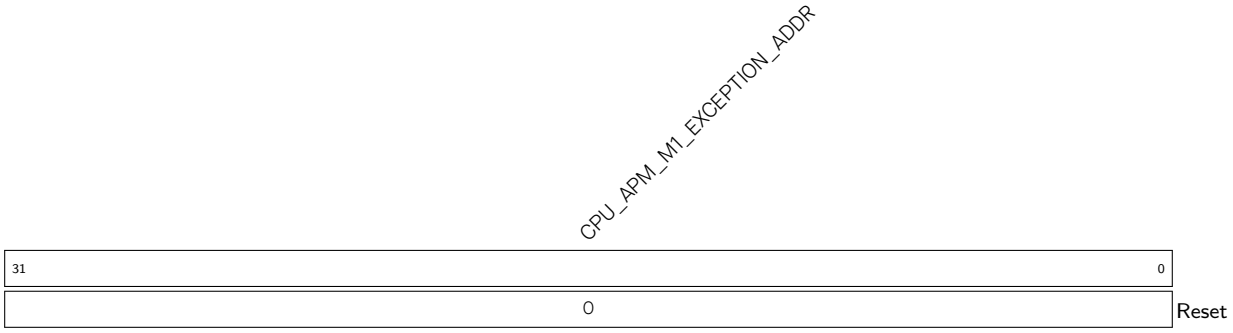


CPU_APM_M1_EXCEPTION_REGION 表示异常区域。(RO)

CPU_APM_M1_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

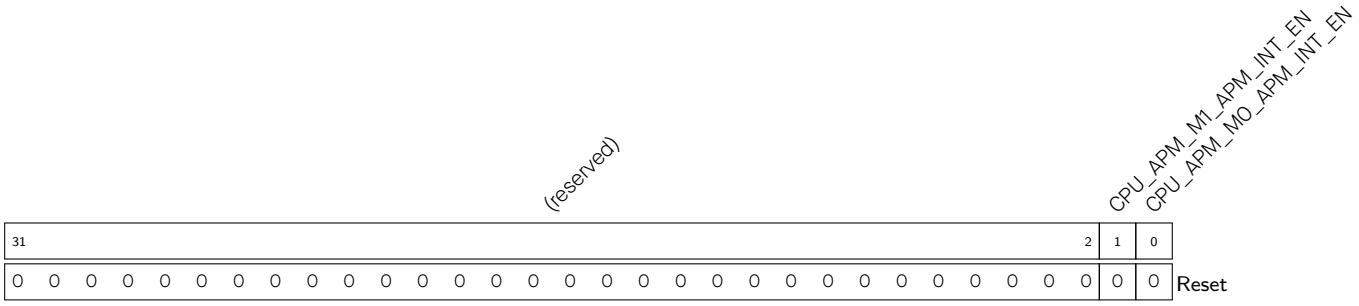
CPU_APM_M1_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 18.69. CPU_APM_M1_EXCEPTION_INFO1_REG (0x00E4)



CPU_APM_M1_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 18.70. CPU_APM_INT_EN_REG (0x0118)



CPU_APM_M0_APM_INT_EN 配置是否使能 CPU_APM_CTRL M0 中断。

0：禁用

1：使能

(R/W)

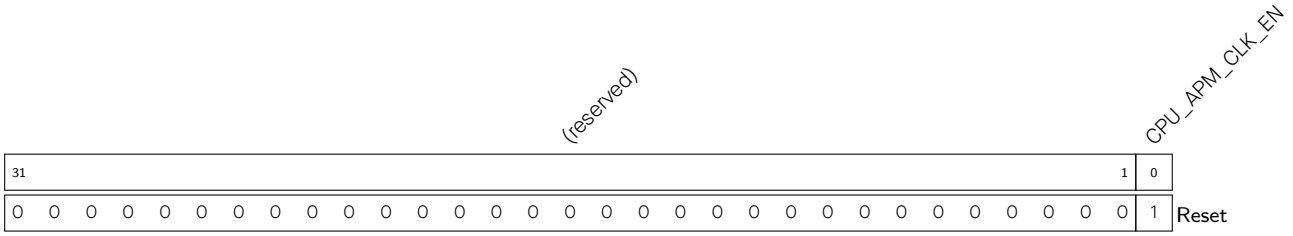
CPU_APM_M1_APM_INT_EN 配置是否使能 CPU_APM_CTRL M1 中断。

0：禁用

1：使能

(R/W)

Register 18.71. CPU_APM_CLOCK_GATE_REG (0x07F8)



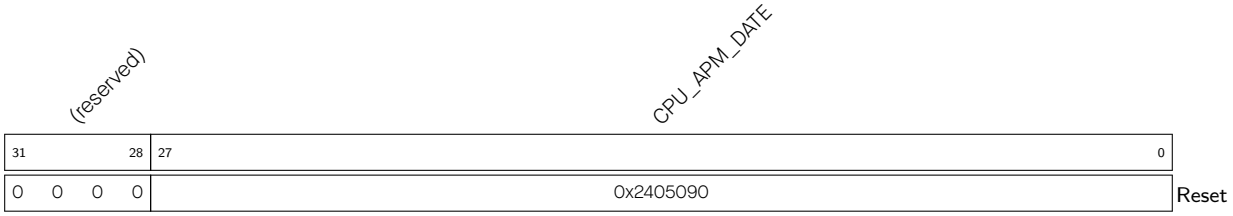
CPU_APM_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 18.72. CPU_APM_DATE_REG (0x07FC)



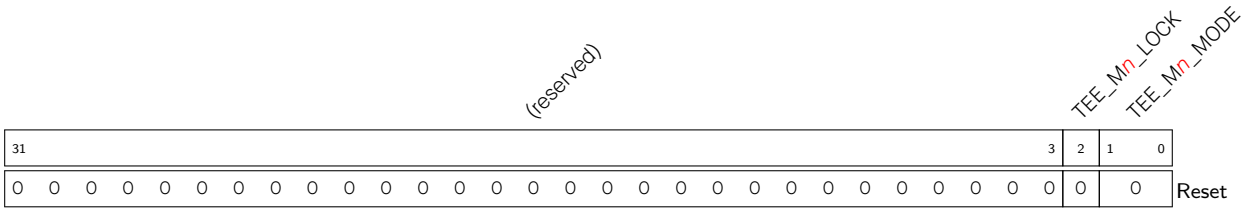
CPU_APM_DATE 版本控制寄存器。(R/W)

18.9.5 TEE_REG

本小节的地址均为相对于 TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 18.73. TEE_M*n*_MODE_CTRL_REG (*n*: 0-31) (0x0000+0x4**n*)



TEE_M*n*_MODE 配置主机 *n* 的安全模式。

0: TEE

1: REE0

2: REE1

3: REE2

(R/W)

TEE_M*n*_LOCK 配置是否锁定 TEE_M*n*_MODE 的值。

0：不锁定

1：锁定

(R/W)

Register 18.74. TEE_UART0_CTRL_REG (0x0088)

Register 18.75. TEE_UART1_CTRL_REG (0x008C)

Register 18.76. TEE_UHCI0_CTRL_REG (0x0090)

Register 18.77. TEE_I2C_EXT0_CTRL_REG (0x0094)

Register 18.78. TEE_I2S_CTRL_REG (0x009C)

Register 18.79. TEE_PARL_IO_CTRL_REG (0x00A0)

Register 18.80. TEE_PWM_CTRL_REG (0x00A4)

Register 18.81. TEE_LEDC_CTRL_REG (0x00AC)

Register 18.82. TEE_TWAI0_CTRL_REG (0x00B0)

Register 18.83. TEE_USB_SERIAL_JTAG_CTRL_REG (0x00B4)

Register 18.84. TEE_RMT_CTRL_REG (0x00B8)

Register 18.85. TEE_GDMA_CTRL_REG (0x00BC)

Register 18.86. TEE_ETM_CTRL_REG (0x00C4)

Register 18.87. TEE_INTMTX_CTRL_REG (0x00C8)

Register 18.88. TEE_APB_ADC_CTRL_REG (0x00D0)

Register 18.89. TEE_TIMERGROUP0_CTRL_REG (0x00D4)

Register 18.90. TEE_TIMERGROUP1_CTRL_REG (0x00D8)

Register 18.91. TEE_SYSTIMER_CTRL_REG (0x00DC)

Register 18.92. TEE_PCNT_CTRL_REG (0x00F4)

Register 18.93. TEE_IOMUX_CTRL_REG (0x00F8)

Register 18.94. TEE_PSRAM_MEM_MONITOR_CTRL_REG (0x00FC)

Register 18.95. TEE_MEM_ACS_MONITOR_CTRL_REG (0x0100)

Register 18.96. TEE_HP_SYSTEM_REG_CTRL_REG (0x0104)

Register 18.97. TEE_PCR_REG_CTRL_REG (0x0108)

Register 18.98. TEE_MSPI_CTRL_REG (0x010C)

Register 18.99. TEE_HP_APM_CTRL_REG (0x0110)

Register 18.100. TEE_CPU_APM_CTRL_REG (0x0114)

Register 18.101. TEE_TEE_CTRL_REG (0x0118)

Register 18.102. TEE_CRYPT_CTRL_REG (0x011C)

Register 18.103. TEE_TRACE_CTRL_REG (0x0120)

Register 18.104. TEE_CPU_BUS_MONITOR_CTRL_REG (0x0128)

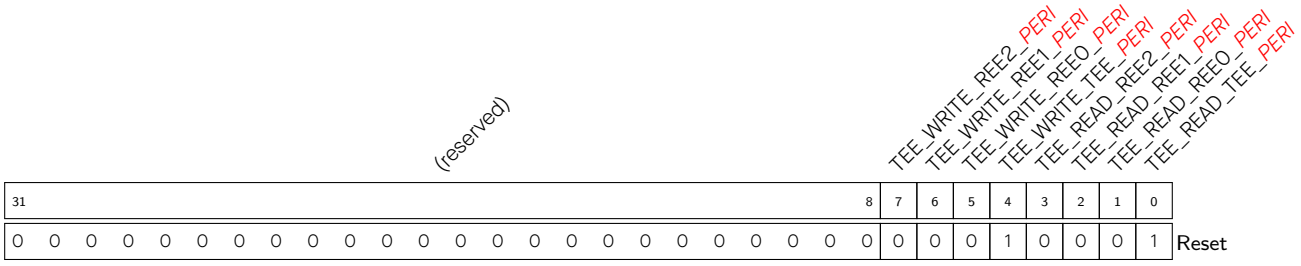
Register 18.105. TEE_INTPRI_REG_CTRL_REG (0x012C)

Register 18.106. TEE_TWAI1_CTRL_REG (0x0138)

Register 18.107. TEE_SPI2_CTRL_REG (0x013C)

Register 18.108. TEE_BS_CTRL_REG (0x0140)

Register 18.109. TEE_*PERI*_CTRL_REG (0x0088-0x0158)



TEE_READ_TEE_*PERI* 配置 TEE 模式下 *PERI* 的读取权限。(R/W)

TEE_READ_REE0_*PERI* 配置 REE0 模式下 *PERI* 的读取权限。(R/W)

TEE_READ_REE1_*PERI* 配置 REE1 模式下 *PERI* 的读取权限。(R/W)

TEE_READ_REE2_*PERI* 配置 REE2 模式下 *PERI* 的读取权限。(R/W)

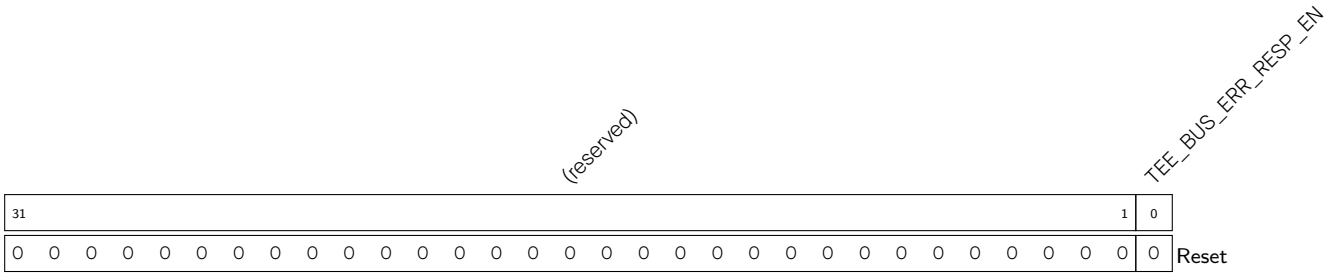
TEE_WRITE_TEE_*PERI* 配置 TEE 模式下 *PERI* 的写入权限。(R/W)

TEE_WRITE_REE0_*PERI* 配置 REE0 模式下 *PERI* 的写入权限。(R/W)

TEE_WRITE_REE1_*PERI* 配置 REE1 模式下 *PERI* 的写入权限。(R/W)

TEE_WRITE_REE2_*PERI* 配置 REE2 模式下 *PERI* 的写入权限。(R/W)

Register 18.110. TEE_BUS_ERR_CONF_REG (0x0FF0)



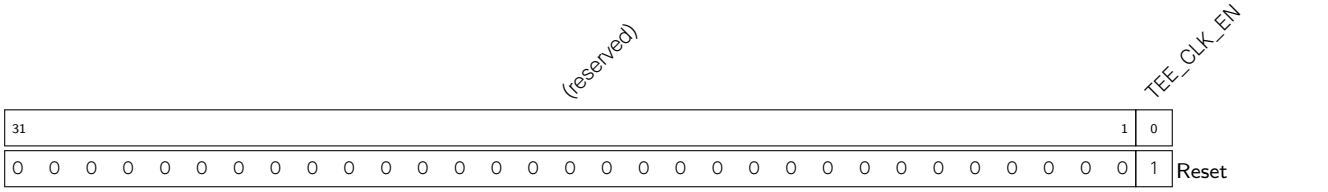
TEE_BUS_ERR_RESP_EN 配置是否使能非法访问返回总线错误信息的功能。

0：禁用

1：使能

(R/W)

Register 18.111. TEE_CLOCK_GATE_REG (0x0FF8)



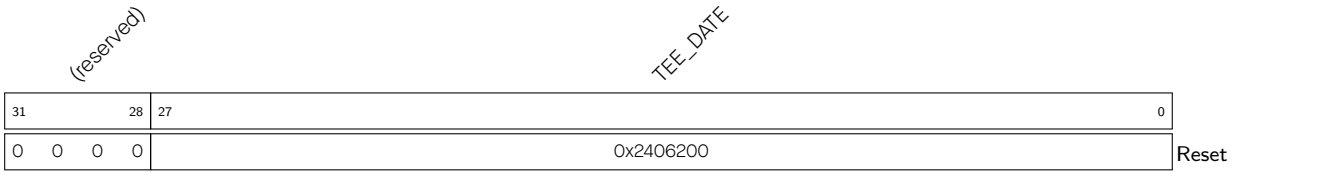
TEE_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 18.112. TEE_DATE_REG (0x0FFC)



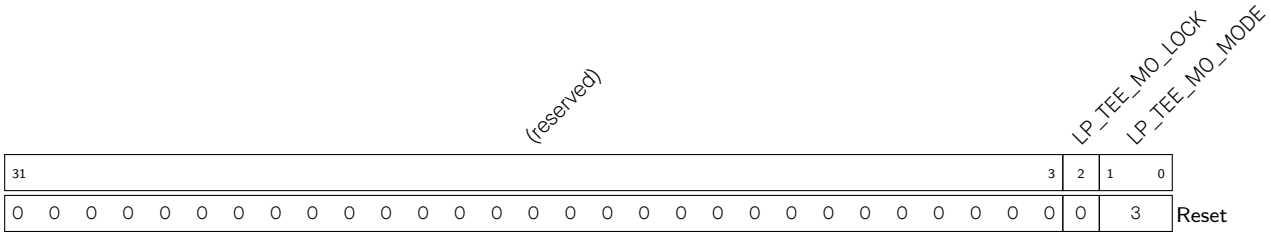
TEE_DATE 版本控制寄存器。(R/W)

18.9.6 LP_TEE_REG

本小节的地址均为相对于 LP_TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

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

Register 18.113. LP_TEE_M0_MODE_CTRL_REG (0x0000)



LP_TEE_M0_MODE 配置 LP CPU 的安全模式。

0: TEE

1: REE0

2: REE1

3: REE2

(R/W)

LP_TEE_M0_LOCK 配置是否锁定 LP_TEE_M0_MODE 的值。

0：不锁定

1：锁定

(R/W)

Register 18.114. LP_TEE_EFUSE_CTRL_REG (0x0004)

Register 18.115. LP_TEE_PMU_CTRL_REG (0x0008)

Register 18.116. LP_TEE_CLKRST_CTRL_REG (0x000C)

Register 18.117. LP_TEE_LP_AON_CTRL_CTRL_REG (0x0010)

Register 18.118. LP_TEE_LP_TIMER_CTRL_REG (0x0014)

Register 18.119. LP_TEE_LP_WDT_CTRL_REG (0x0018)

Register 18.120. LP_TEE_LP_PERI_CTRL_REG (0x001C)

Register 18.121. LP_TEE_LP_ANA_PERI_CTRL_REG (0x0020)

Register 18.122. LP_TEE_LP_IO_CTRL_REG (0x002C)

Register 18.123. LP_TEE_LP_TEE_CTRL_REG (0x0034)

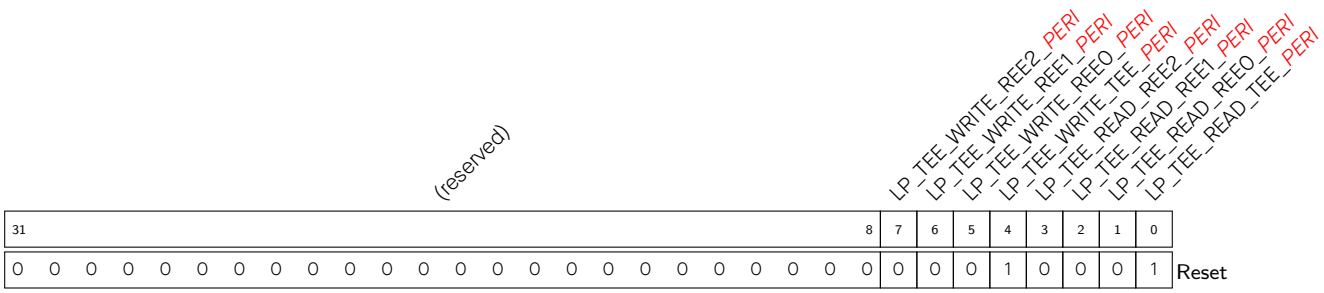
Register 18.124. LP_TEE_UART_CTRL_REG (0x0038)

Register 18.125. LP_TEE_I2C_EXT_CTRL_REG (0x0040)

Register 18.126. LP_TEE_I2C_ANA_MST_CTRL_REG (0x0044)

Register 18.127. LP_TEE_LP_APM_CTRL_REG (0x004C)

Register 18.128. LP_TEE_*PERI*_CTRL_REG (0x0004-0x004C)



LP_TEE_READ_TEE_*PERI* 配置 TEE 模式下 *PERI* 的读取权限。(R/W)

LP_TEE_READ_REE0_*PERI* 配置 REE0 模式下 *PERI* 的读取权限。(R/W)

LP_TEE_READ_REE1_*PERI* 配置 REE1 模式下 *PERI* 的读取权限。(R/W)

LP_TEE_READ_REE2_*PERI* 配置 REE2 模式下 *PERI* 的读取权限。(R/W)

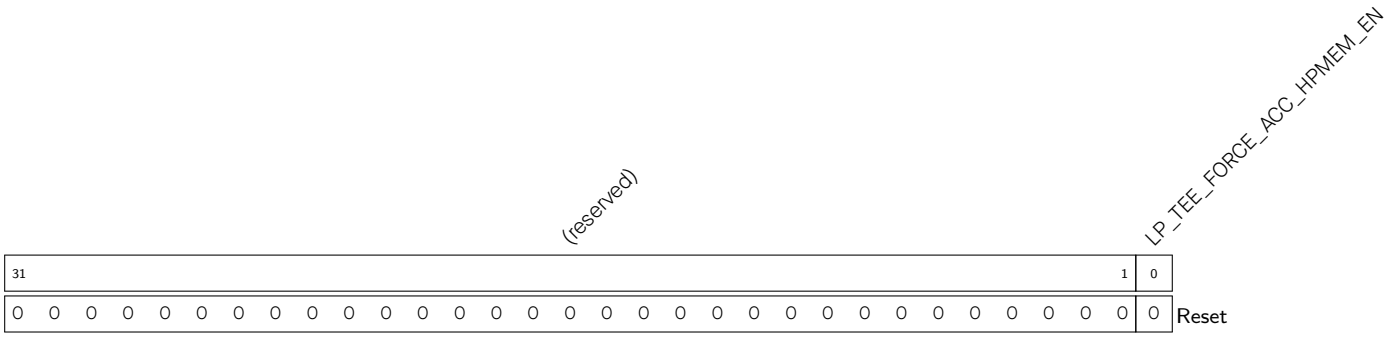
LP_TEE_WRITE_TEE_*PERI* 配置 TEE 模式下 *PERI* 的写入权限。(R/W)

LP_TEE_WRITE_REE0_*PERI* 配置 REE0 模式下 *PERI* 的写入权限。(R/W)

LP_TEE_WRITE_REE1_*PERI* 配置 REE1 模式下 *PERI* 的写入权限。(R/W)

LP_TEE_WRITE_REE2_*PERI* 配置 REE2 模式下 *PERI* 的写入权限。(R/W)

Register 18.129. LP_TEE_FORCE_ACC_HP_REG (0x0090)



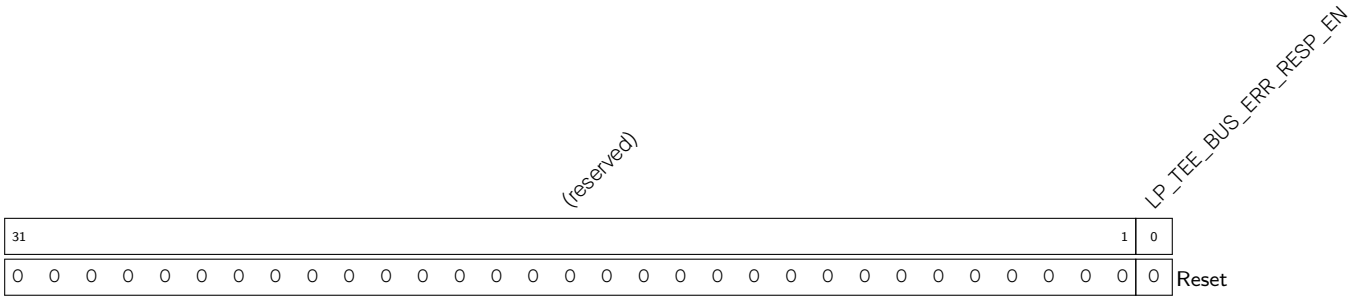
LP_TEE_FORCE_ACC_HPMEM_EN 配置是否允许 LP CPU 不经过 SYS_APM 控制器的权限检查而强制访问 HP SRAM。

0：不允许

1：允许

(R/W)

Register 18.130. LP_TEE_BUS_ERR_CONF_REG (0x00F0)



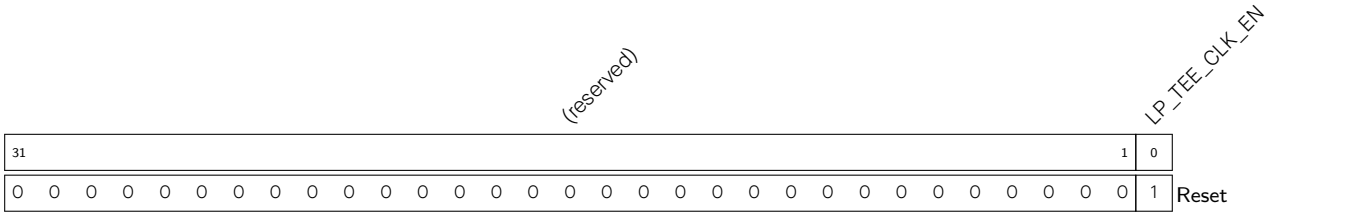
LP_TEE_BUS_ERR_RESP_EN 配置是否使能非法访问返回总线错误信息的功能。

0：禁用

1：使能

(R/W)

Register 18.131. LP_TEE_CLOCK_GATE_REG (0x00F8)



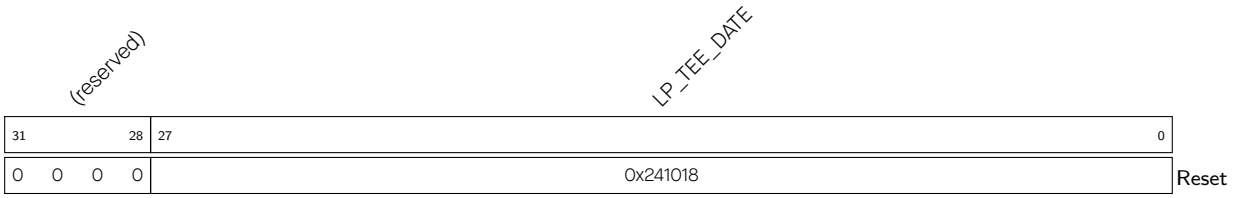
LP_TEE_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 18.132. LP_TEE_DATE_REG (0x00FC)



LP_TEE_DATE 版本控制寄存器。(R/W)

第 19 章

系统寄存器

19.1 概述

ESP32-C5 支持 19.2 主要特性 小节中列出的一系列辅助芯片功能，本章节介绍了如何配置这些功能的寄存器。

19.2 主要特性

ESP32-C5 的系统寄存器可用于控制以下外设和核心模块：

- 外部存储器加密/解密
- 防 DPA 攻击安全
- HP/LP CPU 调试
- 总线超时保护

19.3 功能描述

19.3.1 外部存储器加密/解密配置

寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 配置外部存储器的加密和解密选项。

更多详情，请参考章节 29 片外存储器加密与解密 (XTS_AES)。

19.3.2 防 DPA 攻击安全控制

ESP32-C5 在硬件层面上具有防差分功耗分析 (DPA) 攻击的双重保护机制。

- 首先，ESP32-C5 在对称加密运算过程中引入了掩码 (mask) 机制，通过掩盖运算过程中的真实数据来干扰功耗轨迹。这种安全机制不可关闭。
- 其次，运算时所选用的时钟会动态实时变化，模糊化运算过程中的功耗轨迹。对于此种安全机制，ESP32-C5 提供四种安全级别供用户选用，以适应不同的应用。

表 19.3-1. 安全等级

安全等级	对应值	PLL_CLK (MHz)	XTAL_CLK (MHz)
SEC_DPA_OFF	0	160	48
SEC_DPA_LOW	1	(120 ,160]A	(24,48]A
SEC_DPA_MIDDLE	2	(96,160]A	(16,48]A
SEC_DPA_HIGH	3	(80,160]A	(9.6,48]A

A (x,y] 表示工作频率大于 x Hz，等于或小于 y Hz。

默认情况下，寄存器 HP_SYSTEM_SEC_DPA_CONF_REG 中的字段 HP_SYSTEM_SEC_DPA_CFG_SEL 置 0。此时，安全等级由 eFuse 字段 EFUSE_SEC_DPA_LEVEL 决定。若字段 HP_SYSTEM_SEC_DPA_CFG_SEL 置 1，则安全等级由寄存器 HP_SYSTEM_SEC_DPA_CONF_REG 中的 HP_SYSTEM_SEC_DPA_CFG_LEVEL 决定。

19.3.3 HP/LP CPU 调试控制

寄存器 HP_SYSTEM_CORE_DEBUG_RUNSTALL_ENABLE 可配置 CPU 的 RunStall 功能，便利 HP CPU 和 LP CPU 的调试。一旦使能，当任一个 CPU 进入调试模式时，另一个 CPU 自动暂停工作。

更多详情，请参阅章节 2 高性能处理器 中的 2.10.1调试 小节。

19.3.4 总线超时保护

ESP32-C5 支持超时保护功能，并允许配置超时阈值。当一次传输开始时，超时保护模块内部的计数器会在每个时钟周期递增 1 次。

- 如果在计数值未超过阈值的情况下，从属设备返回响应，计数器将被清零。
- 如果计数值超过设定阈值且未收到从属设备响应，超时保护模块将强制拉高总线返回信号以终止传输。同时，模块会触发中断，并记录异常访问地址及发起访问的主设备 ID (master ID)。

19.3.4.1 CPU 外设超时保护寄存器

寄存器 HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG 用于配置访问 CPU 外设的超时保护功能，即地址段在 0x600C_0000–0x600C_FFFF 之间的外设或模块，详情请见章节 6 系统和存储器 中的 6.3.5 模块/外设地址空间映射 小节。

发生超时异常时，会触发 CPU_PERI_TIMEOUT_INTR 中断。

- HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG：开启超时保护，配置超时阈值。
- HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG：超时发生时，该寄存器将记录超时发生的地址。
- HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG：超时发生时，该寄存器将记录超时发生的 master ID。

19.3.4.2 HP 外设超时保护寄存器

寄存器 HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG 用于配置访问 HP 外设的超时保护功能，即地址段在 0x6000_0000–0x6009_FFFF 之间的外设或模块，详情请见章节 6 系统和存储器 中的 6.3.5 模块/外设地址空间映射 小节。

发生超时异常时，会触发 HP_PERI_TIMEOUT_INTR 中断。

- HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG：开启超时保护，配置超时阈值。
- HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG：超时发生时，该寄存器将记录超时发生的地址。
- HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG：超时发生时，该寄存器将记录超时发生的 master ID。

19.3.4.3 LP 外设超时保护寄存器

寄存器 LP_PERI_BUS_TIMEOUT_CONF_REG 用于配置访问 LP 外设的超时保护功能，即地址段在 0x600B_0000–0x600B_FFFF 之间的外设或模块，详情请见章节 6 系统和存储器 中的 6.3.5 模块/外设地址空间映射 小节。

发生超时异常时，会触发 LP_PERI_TIMEOUT_INTR 中断。

- LP_PERI_BUS_TIMEOUT_CONF_REG：开启超时保护，配置超时阈值。
- LP_PERI_BUS_TIMEOUT_ADDR_REG：超时发生时，该寄存器将记录超时发生的地址。
- LP_PERI_BUS_TIMEOUT_UID_REG：超时发生时，该寄存器将记录超时发生的 master ID。

19.4 寄存器列表

本小节的所有以 HP_SYSTEM 开头的寄存器地址均为相对于高性能系统寄存器基地址的地址偏移量（相对地址），所有以 LP_PERI 开头的寄存器地址均为相对于低功耗外设基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

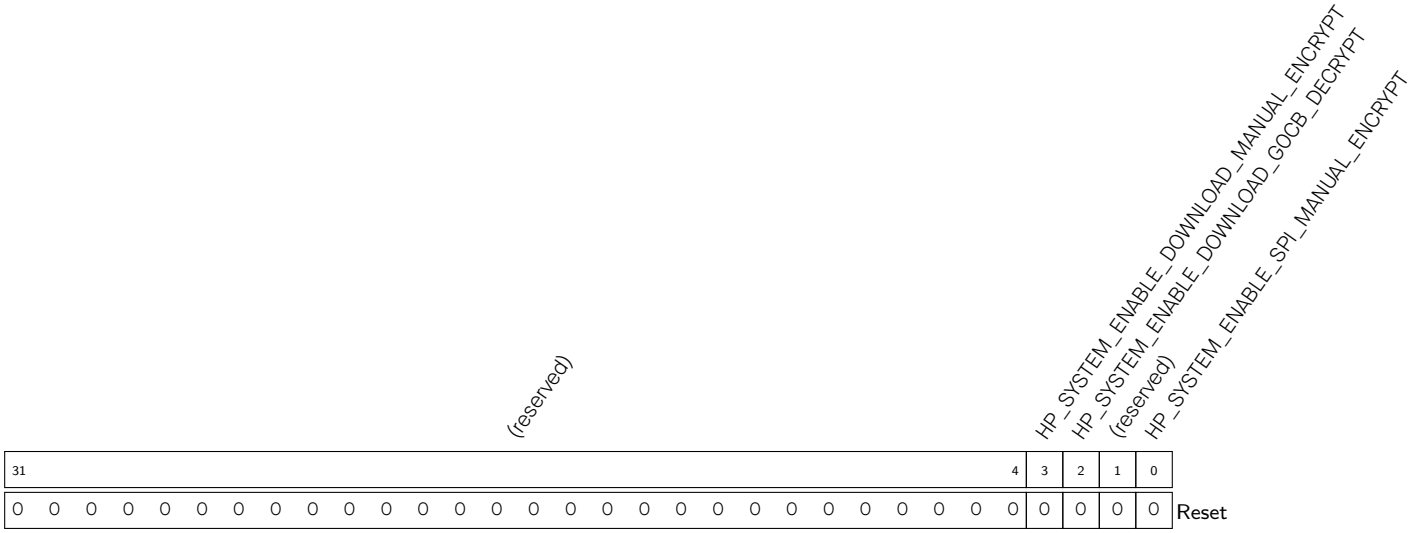
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_-DECRYPT_CONTROL_REG	外部设备加密/解密配置寄存器	0x0000	R/W
HP_SYSTEM_SEC_DPA_CONF_REG	HP 防 DPA 攻击安全配置寄存器	0x0008	R/W
HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_-REG	核心调试 RunStall 配置寄存器	0x0040	R/W
HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG	比特接收器外设选择	0x0080	R/W
CPU 外设超时寄存器			
HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG	超时保护配置寄存器	0x000C	varies
HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG	超时访问地址信息寄存器	0x0010	RO
HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG	Master ID 和 master 权限寄存器	0x0014	WTC
HP 外设超时寄存器			
HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG	超时保护配置寄存器	0x0018	varies
HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG	超时访问地址信息寄存器	0x001C	RO
HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG	Master ID 和 master 权限寄存器	0x0020	WTC
LP 外设超时寄存器			
LP_PERI_BUS_TIMEOUT_CONF_REG	超时保护配置寄存器	0x0010	varies
LP_PERI_BUS_TIMEOUT_ADDR_REG	超时访问地址信息寄存器	0x0014	RO
LP_PERI_BUS_TIMEOUT_UID_REG	Master ID 和 master 权限寄存器	0x0018	WTC
版本寄存器			
HP_SYSTEM_DATE_REG	日期和版本控制寄存器	0x03FC	R/W

19.5 寄存器

本小节的所有以 HP_SYSTEM 开头的寄存器地址均为相对于高性能系统寄存器基地址的地址偏移量（相对地址），所有以 LP_PERI 开头的寄存器地址均为相对于低功耗外设基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 19.1. HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG (0x0000)



HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 配置是否在 SPI Boot 模式下开启 MSPI XTS 手动加密。

0：关闭

1：开启

(R/W)

HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT 配置是否在下载 Boot 模式下开启 MSPI XTS 自动解密。

0：关闭

1：开启

(R/W)

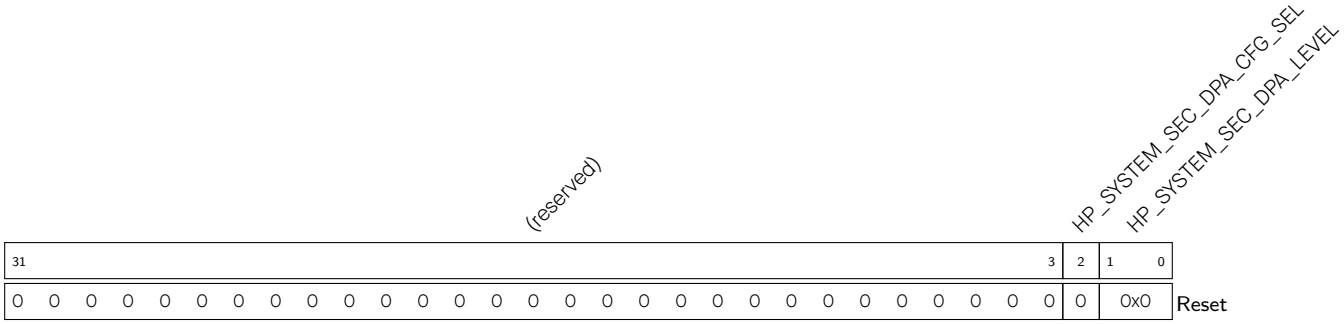
HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 配置是否在下载 Boot 模式下开启 MSPI XTS 手动加密。

0：关闭

1：开启

(R/W)

Register 19.2. HP_SYSTEM_SEC_DPA_CONF_REG (0x0008)



HP_SYSTEM_SEC_DPA_LEVEL 配置是否开启防 DPA 攻击。仅当 HP_SYSTEM_SEC_DPA_CFG_SEL 置 0 时可用。

0：关闭

1-3：开启。值越大，安全等级越高，防 DPA 攻击的能力就越强。不过，硬件加密加速器的计算开销也随之增加。

(R/W)

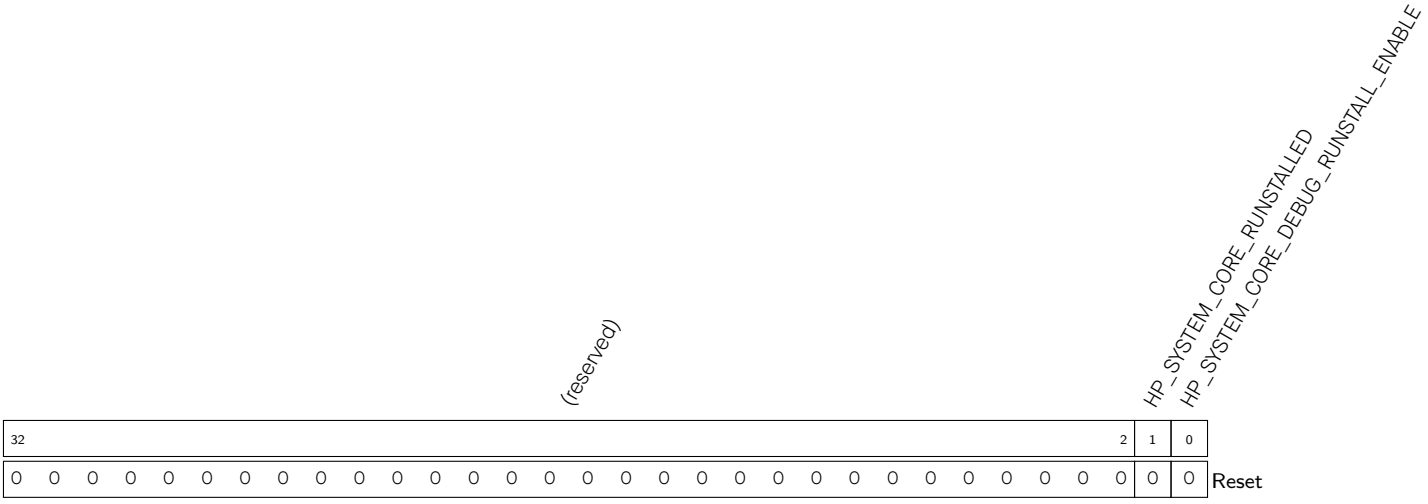
HP_SYSTEM_SEC_DPA_CFG_SEL 配置选择 HP_SYSTEM_SEC_DPA_LEVEL 或是 EFUSE_SEC_DPA_LEVEL (eFuse 寄存器) 来控制 DPA 等级。

0：选择 EFUSE_SEC_DPA_LEVEL

1：选择 HP_SYSTEM_SEC_DPA_LEVEL

(R/W)

Register 19.3. HP_SYSTEM_CORE_DEBUG_RUNSTALL_CONF_REG (0x0040)



HP_SYSTEM_CORE_DEBUG_RUNSTALL_ENABLE 配置是否在启用 HP CPU 和 LP CPU 的 RunStall 功能。一旦启用，当任一个 CPU 进入调试模式时，另一个 CPU 自动暂停工作。

0：关闭

1：开启

(R/W)

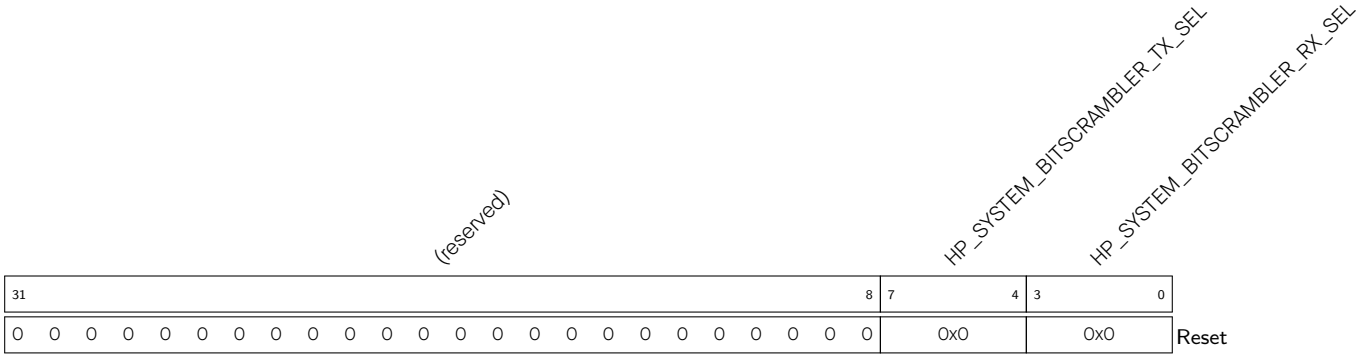
HP_SYSTEM_CORE_RUNSTALLED 表示 HP CPU 的 RunStall 状态。

0：继续工作

1：停止工作

(RO)

Register 19.4. HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG (0x0080)



HP_SYSTEM_BITSCRAMBLER_RX_SEL 配置选择接入比特调节器接收通道的 DMA 外设。

1：GPSPI2
2：UHCI0
3：I2S0
4：AES
5：SHA
6：ADC
7：PARL_IO
其他：NONE
(R/W)

HP_SYSTEM_BITSCRAMBLER_TX_SEL 配置选择接入比特调节器发送通道的 DMA 外设。

1：GPSPI2
2：UHCI0
3：I2S0
4：AES
5：SHA
6：ADC
7：PARL_IO
其他：保留
(R/W)

Register 19.5. HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG (0x000C)



HP_SYSTEM_CPU_PERI_TIMEOUT_THRES 配置总线访问 CPU 外设寄存器的超时阈值，与时钟域的时钟周期数相对应。(R/W)

HP_SYSTEM_CPU_PERI_TIMEOUT_INT_CLEAR 写 1 清除超时中断。(WT)

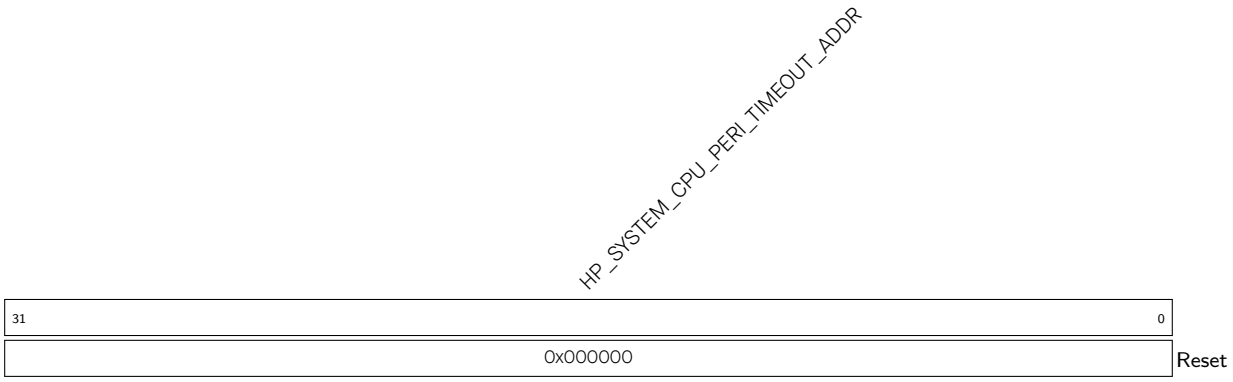
HP_SYSTEM_CPU_PERI_TIMEOUT_PROTECT_EN 配置是否启用访问 CPU 外设寄存器的超时保护。

0：关闭

1：开启

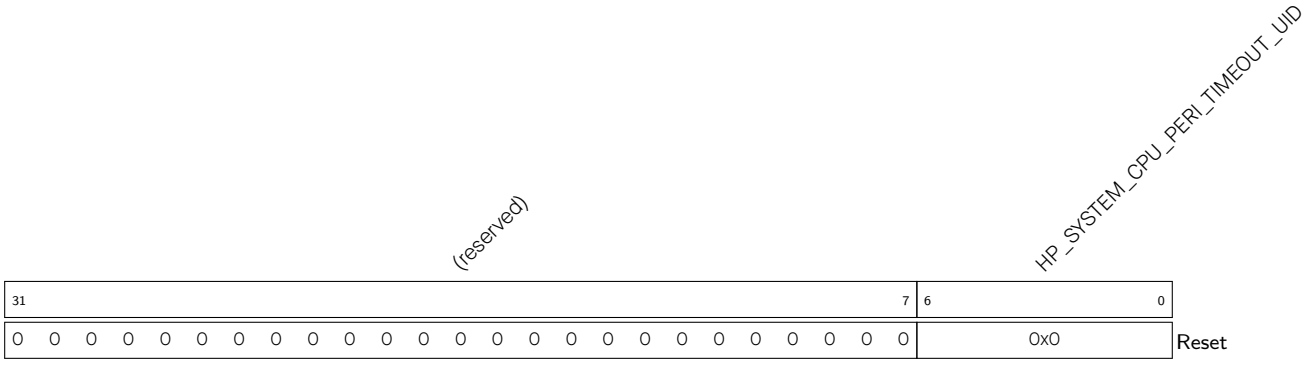
(R/W)

Register 19.6. HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG (0x0010)



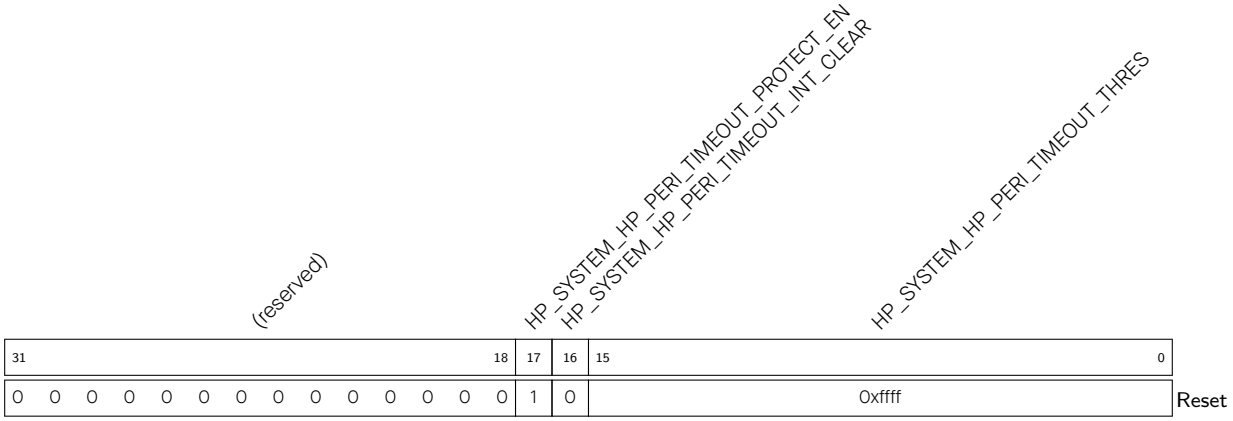
HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR 表示异常访问的地址信息。(RO)

Register 19.7. HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG (0x0014)



HP_SYSTEM_CPU_PERI_TIMEOUT_UID 表示超时发生时的 master ID[4:0] 和 master 权限 [6:5]。该寄存器将在清除超时中断后清除。(WTC)

Register 19.8. HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG (0x0018)



HP_SYSTEM_HP_PERI_TIMEOUT_THRES 配置总线访问 HP 外设寄存器的超时阈值，单位为时钟域的时钟周期数。(R/W)

HP_SYSTEM_HP_PERI_TIMEOUT_INT_CLEAR 写 1 清除超时中断。(WT)

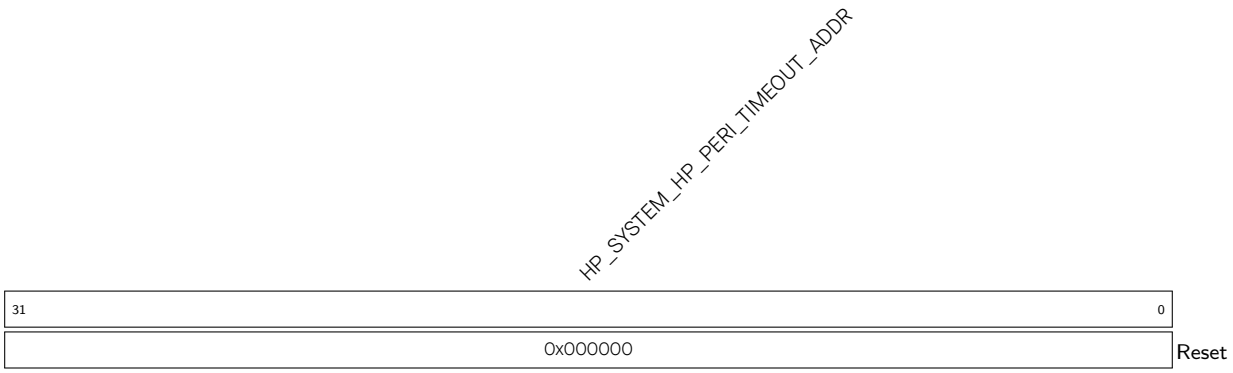
HP_SYSTEM_HP_PERI_TIMEOUT_PROTECT_EN 配置是否启用访问 HP 外设寄存器的超时保护。

0：关闭

1：开启

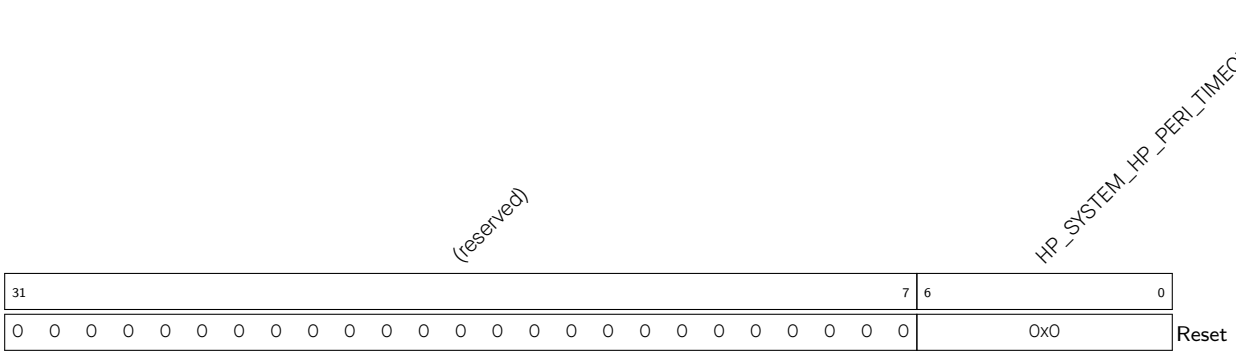
(R/W)

Register 19.9. HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG (0x001C)



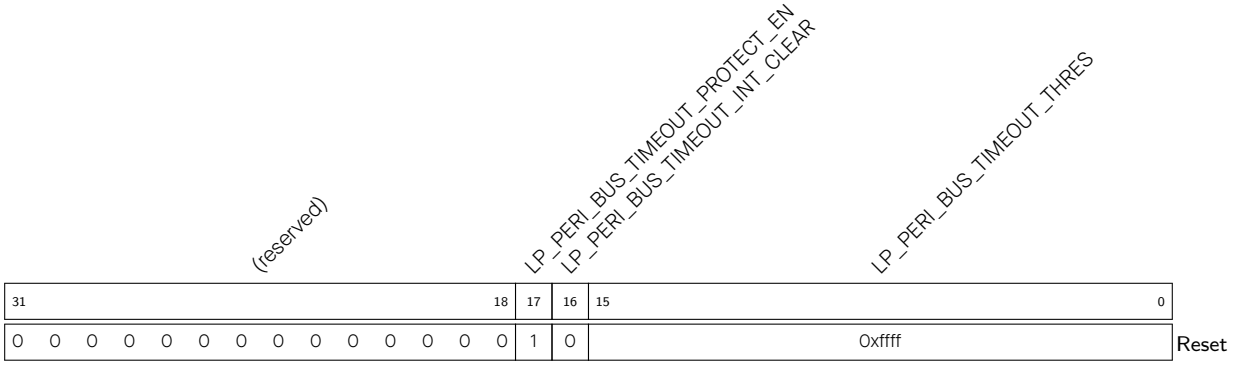
HP_SYSTEM_HP_PERI_TIMEOUT_ADDR 表示异常访问的地址信息。(RO)

Register 19.10. HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG (0x0020)



HP_SYSTEM_HP_PERI_TIMEOUT_UID 表示超时发生时的 master ID[4:0] 和 master 权限 [6:5]。该寄存器将在清除超时中断后清除。(WTC)

Register 19.11. LP_PERI_BUS_TIMEOUT_CONF_REG (0x0010)



LP_PERI_BUS_TIMEOUT_THRES 配置总线访问 LP 外设寄存器的超时阈值，单位为时钟域的时钟周期数。(R/W)

LP_PERI_BUS_TIMEOUT_INT_CLEAR 写 1 清除超时中断。(WT)

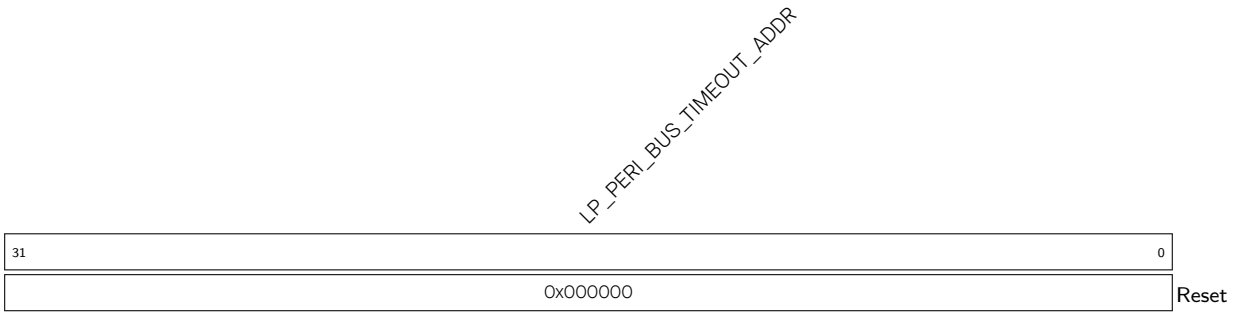
LP_PERI_BUS_TIMEOUT_PROTECT_EN 配置是否启用访问 LP 外设寄存器的超时保护。

0：关闭

1：开启

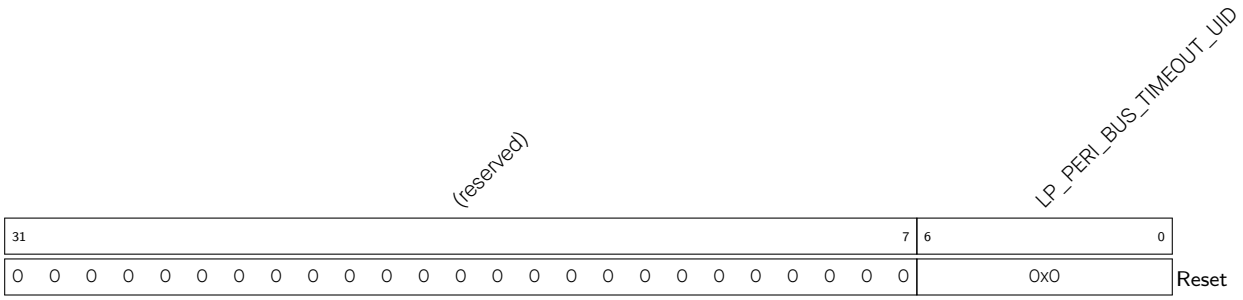
(R/W)

Register 19.12. LP_PERI_BUS_TIMEOUT_ADDR_REG (0x0014)



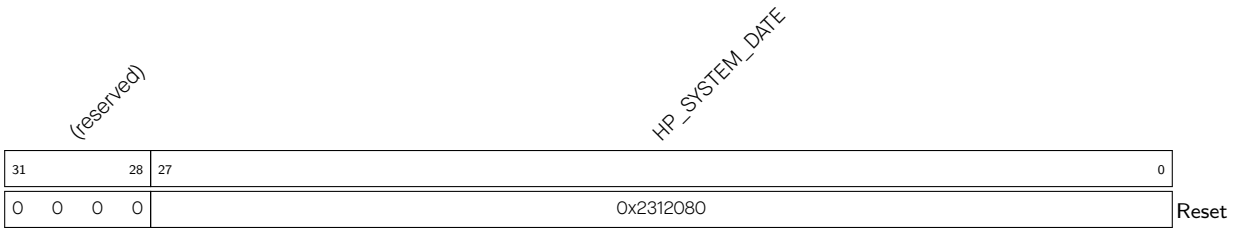
LP_PERI_BUS_TIMEOUT_ADDR 表示异常访问的地址信息。(RO)

Register 19.13. LP_PERI_BUS_TIMEOUT_UID_REG (0x0018)



LP_PERI_BUS_TIMEOUT_UID 表示超时发生时的 master ID[4:0] 和 master 权限 [6:5]。该寄存器将在清除超时中断后清除。(WTC)

Register 19.14. HP_SYSTEM_DATE_REG (0x03FC)



HP_SYSTEM_DATE 版本控制寄存器。(R/W)

第 20 章

辅助调试

20.1 概述

ESP32-C5 辅助调试模块提供一套调试功能，可在软件开发时用于调试，定位问题所在。

20.2 主要特性

辅助调试模块具有以下特性：

- **区域读写监测**：监测高性能处理器 (High-Performance CPU, HP CPU) 总线是否在限定的存储器地址范围内进行读写操作，若在该地址范围内发生读写操作则触发中断。
- **栈指针 (SP) 监测**：监测栈指针是否超出限定的范围，若超出范围则产生中断。
- **程序计数器 (PC) 记录**：记录 PC，可以获取上一次 HP CPU 复位时的 PC 值。
- **总线访问记录**：记录总线访问信息，当 HP CPU、低功耗处理器 (Low-Power CPU, LP CPU) 或 DMA 写了某个特殊值时，会记录此次写操作的总线类型、地址和 PC 值（仅记录 HP CPU 写操作的 PC），并将这些信息记录到 HP SRAM 中。

20.3 功能描述

20.3.1 区域读写监测

为确认 HP CPU 是否在某段地址范围（即区域）进行过读写操作，辅助调试模块能够监测 HP CPU 总线，当总线在监测地址范围进行读写操作时会触发中断，详情请见章节 2 高性能处理器 中的表 2.4-1。

当 HP CPU 总线访问数据空间时，本章节将其称为**数据总线**；而当 HP CPU 总线访问 AHB 外设空间时，本章节将其称为**外设总线**。辅助调试模块可在数据总线上同时监测在两段地址范围内的读写操作，假设命名为 HP CPU 数据总线区域 0 和 HP CPU 数据总线区域 1；也可在外设总线上同时监测两段地址范围，假设命名为 HP CPU 外设总线区域 0 和 HP CPU 外设总线区域 1。区域 0 和区域 1 根据需要进行配置。

详细区域读写监测模式如下。

- 数据总线读写监测
  - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 0 是否进行读操作
  - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 0 是否进行写操作
  - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 1 是否进行读操作
  - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 1 是否进行写操作

- 外设总线读写监测
  - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 0 是否进行读操作
  - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 0 是否进行写操作
  - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 1 是否进行读操作
  - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 1 是否进行写操作

20.3.2 栈指针监测

为防止 HP CPU 栈溢出或者错误的压栈出栈，辅助调试模块能够监测栈指针。当 HP CPU 栈指针超过 HP CPU 栈指针监测区域限定的上下边界时，模块会记录 PC 指针并产生中断，边界由软件进行配置。

详细栈指针监测模式如下。

- 监测 HP CPU 栈指针是否大于 HP CPU 栈指针监测区域上边界
- 监测 HP CPU 栈指针是否小于 HP CPU 栈指针监测区域下边界

20.3.3 PC 记录

在某些情况下，软件开发者希望知道上次 HP CPU 复位时的 PC 指针。比如，程序卡死只能复位时，开发者可能希望读取复位时的 PC 指针，以便知道程序在哪里卡死，然后进行调试。辅助调试模块可以记录 HP CPU 复位时的 PC，方便开发者进行调试。

20.3.4 CPU/DMA 总线访问记录

辅助调试模块能够实时记录 HP CPU 数据总线、LP CPU 总线和 DMA 总线的写操作，当总线在某个范围内发生写操作或者在某个范围内写某个特定的值时，此次操作的总线类型、地址和 PC（仅记录 HP CPU 写操作的 PC）等信息会被记录下来，并按照一定的格式存储到 HP SRAM 中。此处提到的地址范围只能处于 HP SRAM 或外部 RAM 的范围区间。

20.4 中断

ESP32-C5 的辅助调试模块可以生成 BUS_MONITOR_INTR 中断信号并发送给中断矩阵。这一中断信号由以下辅助调试模块的内部中断源生成。

- BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 0 读操作时触发。
- BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 0 写操作时触发。
- BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 1 读操作时触发。
- BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 1 写操作时时触发。
- BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT：当 HP CPU 外设总线在 HP CPU 外设总线区域 0 读操作时触发。

- BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT：当 HP CPU 外设总线在 HP CPU 外设总线区域 0 写操作时时触发。
- BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT：当 HP CPU 外设总线在 HP CPU 外设总线区域 1 读操作时触发。
- BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT：当 HP CPU 外设总线在 HP CPU 外设总线区域 1 写操作时时触发。
- BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT：当 HP CPU 栈指针小于 HP CPU 栈指针监测区域的下边界时触发。
- BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT：当 HP CPU 栈指针大于 HP CPU 栈指针监测区域的上边界时触发。

说明：
对于中断、中断信号、中断源及其关系的详细描述，可以查看章节 11 中断矩阵 > Section 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 20.6 寄存器列表。

20.5 配置流程

20.5.1 区域监测和栈指针监测配置

区域监测和栈指针监测的配置流程如下。

1. 配置监测区域和栈指针。
    - HP CPUO 数据总线区域 0 由 BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX_REG 进行配置。
    - HP CPU 数据总线区域 1 由 BUS_MONITOR_CORE_0_AREA_DRAMO_1_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_DRAMO_1_MAX_REG 进行配置。
    - HP CPU 外设总线区域 0 由 BUS_MONITOR_CORE_0_AREA_PIF_0_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_PIF_0_MAX_REG 进行配置。
    - HP CPU 外设总线区域 1 由 BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG 进行配置。
    - HP CPU 栈指针监测区域由 BUS_MONITOR_CORE_0_SP_MIN_REG 和 BUS_MONITOR_CORE_0_SP_MAX_REG 进行配置。
2. 配置 BUS_MONITOR_CORE_0_INTR_ENA_REG 使能不同模式的中断。
3. 配置 BUS_MONITOR_CORE_0_MONTR_ENA_REG 使能不同的监测模式，可同时使能。
4. 读取 BUS_MONITOR_CORE_0_INTR_RAW_REG 查询不同模式的原始中断状态。
5. 配置 BUS_MONITOR_CORE_0_INTR_CLR_REG 清除不同模式的中断。

比如，若想监测 HP CPU 数据总线地址 A 到地址 B 范围内是否进行过写操作，可通过 HP CPU 数据总线区域 0 或者区域 1 进行监测。以 HP CPU 数据总线区域 0 为例：

1. 配置 BUS_MONITOR_CORE_0_RCD_PDEBUGEN 为 1，使能 HP CPU 更新 PC 信号给辅助调试模块。
2. 配置 BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN_REG 为 A 地址。
3. 配置 BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX_REG 为 B 地址。
4. 配置 BUS_MONITOR_CORE_0_INTR_ENA_REG bit[1] 使能 HP CPU 数据总线区域 0 写监测中断。
5. 配置 BUS_MONITOR_CORE_0_MONTR_ENA_REG bit[1] 使能 HP CPU 数据总线区域 0 写监测。
6. 配置中断矩阵将 BUS_MONITOR_INTR 映射到 HP CPU 中断上（参考章节 11 中断矩阵）。
7. 触发中断后：
   - 读取 BUS_MONITOR_CORE_0_INTR_RAW_REG 确认中断源。
   - 如果是 HP CPU 区域监测引发的中断，读取 BUS_MONITOR_CORE_0_AREA_PC 可获取触发中断时刻的 HP CPU PC 值，读取 BUS_MONITOR_CORE_0_AREA_SP 可获取触发中断时刻的 HP CPU 栈指针。
   - 如果是栈指针监测引发的中断，读取 BUS_MONITOR_CORE_0_SP_PC 可获取触发中断时刻的 HP CPU PC 值。
   - 配置 BUS_MONITOR_CORE_0_INTR_CLR_REG 对应位为 1 可清除中断。

20.5.2 PC 记录配置

配置 BUS_MONITOR_CORE_0_RCD_PDEBUGEN 为 1，使能 HP CPU 更新 PC 信号给辅助调试模块。若 BUS_MONITOR_CORE_0_RCD_RECORDEN 也配置为 1，BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG 会记录 HP CPU PC 信号，同时 BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG 会记录 HP CPU 栈指针，否则保留原值。

当 HP CPU 发生复位时，BUS_MONITOR_CORE_0_RCD_EN_REG 会被复位，但是 BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG 和 BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG 不会被复位，因此这两个寄存器会一直保留 HP CPU 复位时刻的 PC 值和栈指针。

20.5.3 CPU/DMA 总线访问记录配置

20.5.3.1 总线访问记录 1 配置

总线访问记录 1 包括 HP CPU 总线访问记录（HP SRAM 和外部 RAM 监控功能）、LP CPU 总线访问记录（HP SRAM 监控功能）和 DMA 总线访问记录（HP SRAM 监控功能），其配置流程如下。

1. 配置监测地址范围：配置 MEM_MONITOR_LOG_MIN_REG 和 MEM_MONITOR_LOG_MAX_REG 确定地址范围。对于 HP CPU，监测地址范围需处于 HP SRAM (0x4080_0000 ~ 0x4085_FFFF) 或外部 RAM (0x4200_0000 ~ 0x43FF_FFFF) 的地址区间内；对于 LP CPU 或 DMA 监测地址，范围需处于 HP SRAM (0x4080_0000 ~ 0x4085_FFFF) 的地址区间内。
2. 配置监测模式 (MEM_MONITOR_LOG_MODE)
   - 写监测（监测总线是否发生写操作）
   - 字 (word) 监测（监测总线是否写某个特定 word）
   - 半字 (halfword) 监测（监测总线是否写某个特定 halfword）
   - 字节 (byte) 监测（监测总线是否写某个特定 byte）

3. 配置需要监测的特殊值

   - 监测模式为 word 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG 即为监测的特定 word。
   - 监测模式为 halfword 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] 即为监测的特定 halfword。
   - 监测模式为 byte 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] 即为监测的特定 byte。
   - MEM_MONITOR_LOG_DATA_MASK_REG 用于屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 的相应 byte。当某一 byte 被屏蔽，表示该 byte 可为任意值，比如在 word 监测模式下，MEM_MONITOR_LOG_CHECK_DATA_REG 配置为 0x01020304，MEM_MONITOR_LOG_DATA_MASK_REG 配置为 0x1，此时只要总线写 0x010203XX，都会被记录。

4. 配置记录信息的存储地址范围

   - MEM_MONITOR_LOG_MEM_START_REG 和 MEM_MONITOR_LOG_MEM_END_REG 为存储地址范围配置寄存器。记录信息的存储地址范围为 0x4080_0000 ~ 0x4085_FFFF。
   - 置位 MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG，使 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 更新为 MEM_MONITOR_LOG_MEM_START_REG。
   - 配置辅助调试模块对 HP SRAM 的使用权限，只有开启了辅助调试模块对 HP SRAM 的使用权限才能访问 HP SRAM。关于如何配置，详情见章节 18 权限控制 *(PMS)*。

5. 配置记录信息的存储模式，分为 loop 模式和非 loop 模式

   - loop 模式下，将记录信息循环写进配置地址范围内，当写到结束地址时回到起始地址开始写，覆盖之前记录的信息。置位 MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使能 loop 模式。
     例如，存储记录信息的地址范围为 0 ~ 4 ，总线访问过程中有 1 ~ 10 共十次写记录信息的操作，那么第 5 次写操作写到地址 4 后，第 6 次写操作会回到地址 0 开始写，第 6 ~ 10 写操作会覆盖之前的第 1 ~ 5 写操作的数据。
   - 非 loop 模式下，当将记录信息写到结束地址后，会一直停留在结束地址，并不再写记录信息，不会覆盖最开始的记录信息，并且丢弃后面的记录信息。清零 MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使能非 loop 模式。
     例如，存储记录信息的地址范围为 0 ~ 4 ，总线访问过程中有 1 ~ 10 共十次写记录信息的操作，那么第 5 次写操作写到地址 4 后，第 6 ~ 10 次写操作的地址会一直在地址 4 上，但不再进行写操作，因此地址范围 0 ~ 4 保留前 5 次写操作信息，并且丢弃第 6 ~ 10 次写操作信息。

6. 配置总线使能

   - 配置 MEM_MONITOR_LOG_CORE_ENA、MEM_MONITOR_LOG_DMA_1_ENA 和 MEM_MONITOR_LOG_DMA_0_ENA 分别使能 HP CPU、LP CPU 和 DMA 总线访问记录，可同时使能。

辅助调试模块会先将记录信息缓存到一个内部 buffer A 里，然后从 buffer 取出记录信息再存储到配置的存储区域内。当监测行为被连续触发时，连续产生的记录数据包可能会使得 buffer 满而不能缓存新的记录数据包，此时模块会舍弃掉不能及时缓存到 buffer 的记录数据包，并在 buffer 不满的时候缓存一个 LOST 数据包作为替代。此时仅能获取 LOST 数据包生成时所舍弃的记录数据包的总线类型，但无法得知在此之前舍弃了多少个记录数据包。

当总线访问记录结束后需要从内存中读取记录数据并进行解码。记录数据里只有四种包格式，HP CPU 数据包、LP CPU 数据包、DMA 数据包和 LOST 数据包，前三种数据包分别对应 HP CPU 数据总线记录信息、LP CPU 数

据总线记录信息和 DMA 数据总线记录信息，四种包格式如表 20.5-1、20.5-2、20.5-3 和 20.5-4 所示：

表 20.5-1. HP CPU 包格式

Bit[63:33]	Bit[32]	Bit[31:3]	Bit[2:1]	Bit[0]
pc_offset	anchored(1)	addr_offset	format	anchored(0)

表 20.5-2. LP CPU 包格式

Bit[31:8]	Bit[7:3]	Bit[2:1]	Bit[0]
addr_offset	保留	format	anchored(0)

表 20.5-3. DMA 包格式

Bit[31:8]	Bit[7:3]	Bit[2:1]	Bit[0]
addr_offset	dma_source	format	anchored(0)

表 20.5-4. LOST 包格式

Bit[31:7]	Bit[6:3]	Bit[2:1]	Bit[0]
保留	lost_source	format	anchored(0)

从包格式可以看出，HP CPU 数据包共 64 位，其他数据包均为 32 位。以下为各个域的含义：

- **format** 表示此次数据包的类型：
  - 0：HP CPU 数据包
  - 1：DMA 数据包
  - 2：LP CPU 数据包
  - 3：LOST 数据包
- **pc_offset** 记录了 HP CPU 的 PC 指针的偏移量，实际 PC = pc_offset + 0x0000_0000。
- **addr_offset** 记录了此次写操作的地址偏移，实际地址 = addr_offset + {MEM_MONITOR_LOG_MIN_REG[31:2], 2'b0}。
- **dma_source** 记录了 DMA 的访问来源，详见表 20.5-5，其中值 16 ~ 31 对应的来源参考 *5 通用 DMA 控制器 (GDMA)*。
- **lost_source** 记录了 LOST 数据包生成时舍弃了哪些数据包：
  - Bit[4:3]：
    - 0：未舍弃 HP CPU 数据包
    - 3：舍弃了 HP CPU 数据包
    - 其他值：保留
  - Bit[5]：
    - 0：未舍弃 DMA 数据包

    * 1：舍弃了 DMA 数据包
  - Bit[6]：
    * 0：未舍弃 LP CPU 数据包
    * 1：舍弃了 LP CPU 数据包
- **anchored** 表示该 32 位数据在数据包中的位置：
  - 0：该 32 位数据是数据包的低 32 位
  - 1：该 32 位数据是数据包的高 32 位

表 20.5-5. DMA 访问来源

值	来源
0	HP CPU
1	LP CPU
2	保留
3	保留
4	保留
5	MEM_MONITOR
6	TRACE
7	保留
8	PSRAM_MEM_MONITOR
9 ~ 15	保留
16 ~ 31	详见章节 5 通用 *DMA 控制器 (GDMA)* > 表 5.4-1 中 0 ~ 15 对应的外设。例如，16 对应的来源为该表中 0 对应的外设，17 对应的来源为该表中 1 对应的外设，以此类推

模块内部 buffer 的数据宽度是 32 位。若同时使能 HP CPU、LP CPU 和 DMA 总线访问记录，并且在某一时刻同时生成了记录数据，则先将 HP CPU 数据包缓存到 buffer 中，再缓存 DMA 数据包，最后缓存 LP CPU 数据包。辅助调试模块会自动将缓存数据从 buffer 取出，并将取出的数据再以 32 位数据宽度存入指定范围内存中。

当启用 loop 模式时，若在配置的存储地址范围内循环了若干次，可能存在残留数据干扰解析，即 HP CPU 数据包的低 32 位数据被覆盖，使得高 32 位数据成为残留数据，因此确定第一个有效数据包的位置时，必须过滤可能存在的残留数据。读取 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 可确定数据包的起始位置，检查数据包 anchored 位的数值，是 0 则保留，是 1 则舍弃。

数据包解析流程如下。

- 读取 MEM_MONITOR_LOG_MEM_FULL_FLAG 确定数据是否溢出配置范围。
  - 如果未溢出，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG – 4；
  - 若溢出并且开启了 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG 和 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG – 4；

  - 若溢出并且未开启 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG。

- 从起始地址处开始读取数据并解析，每次读取 32 位。

数据解析完成之后需要通过置位 MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 清除 MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

20.5.3.2 总线访问记录 2 配置

总线访问记录 2 包括 DMA 总线访问记录（外部 RAM 监控功能），其配置流程如下。

1. 配置监测地址范围：配置 MEM_MONITOR_LOG_MIN_REG 和 MEM_MONITOR_LOG_MAX_REG 确定地址范围。监测地址范围需处于 0x4200_0000 ~ 0x43FF_FFFF 区间内。
2. 配置 MEM_MONITOR_LOG_MODE 选择监测模式
   - 写监测（监测总线是否发生写操作）
   - 字 (word) 监测（监测总线是否写某个特定 word）
   - 半字 (halfword) 监测（监测总线是否写某个特定 halfword）
   - 字节 (byte) 监测（监测总线是否写某个特定 byte）
3. 配置需要监测的特殊值
   - 监测模式为 word 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG 即为监测的特定 word。
   - 监测模式为 halfword 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] 即为监测的特定 halfword。
   - 监测模式为 byte 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] 即为监测的特定 byte。
   - MEM_MONITOR_LOG_DATA_MASK_REG 用于屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 的相应 byte。当某一 byte 被屏蔽，表示该 byte 可为任意的值，比如 word 监测，MEM_MONITOR_LOG_CHECK_DATA_REG 配置为 0x01020304，MEM_MONITOR_LOG_DATA_MASK_REG 配置为 0x1，因此只要总线写 0x010203XX 都会被记录。
4. 配置记录信息的存储地址范围
   - MEM_MONITOR_LOG_MEM_START_REG 和 MEM_MONITOR_LOG_MEM_END_REG 为存储地址范围配置寄存器。记录信息的存储地址范围为 0x4080_0000 ~ 0x4085_FFFF。
   - 置位 MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG，使 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 更新为 MEM_MONITOR_LOG_MEM_START_REG。
   - 配置辅助调试模块对 HP SRAM 的使用权限，只有开启了辅助调试模块对 HP SRAM 的使用权限才能访问 HP SRAM。关于如何配置，详情见章节 18 权限控制 *(PMS)*。
5. 配置记录信息的存储模式，分为 loop 模式和非 loop 模式
   - loop 模式下，将记录信息循环写进配置地址范围内，当写到结束地址时回到起始地址开始写，覆盖之前记录的信息。置位 MEM_MONITOR_LOG_MEM_LOOP_ENABLE 启用 loop 模式。
   - 非 loop 模式下，当将记录信息写到结束地址后，会一直停留在结束地址，并不再写记录信息，不会覆盖最开始的记录信息，并且丢弃后面的记录信息。清零 MEM_MONITOR_LOG_MEM_LOOP_ENABLE 启用非 loop 模式。

- 示例参考小节 20.5.3.1 的步骤 5。

6. 配置 MEM_MONITOR_LOG_DMA_0_ENA 使能 DMA_0 通道组总线访问记录。

辅助调试模块会先将 DMA 记录信息缓存到一个内部 buffer B 里，然后从 buffer 取出记录信息再存储到配置的存储区域内。当监测行为被连续触发时，连续产生的记录数据包可能会使得 buffer 满而不能缓存新的记录数据包，此时模块会舍弃掉不能及时缓存到 buffer 的记录数据包，并在 buffer 不满的时候缓存一个 DMA LOST 数据包作为替代，此时仅能知道 DMA LOST 数据包生成时所舍弃的记录数据包的总线类型，但不能知道在此之前舍弃了多少个记录数据包。

当总线访问记录结束后，需要从内存中读取记录数据并进行解码。记录数据里只有两种包格式，DMA_0 数据包和 DMA LOST 数据包，DMA_0 数据包对应 DMA_0 通道组数据总线记录信息，两种包格式如表 20.5-6 和 20.5-7 所示：

表 20.5-6. DMA_0 包格式

Bit[31:7]	Bit[6:2]	Bit[1]	Bit[0]
addr_offset	dma_source	format	anchored(0)

表 20.5-7. DMA LOST 包格式

Bit[31:3]	Bit[2]	Bit[1]	Bit[0]
保留	lost_source	format	anchored(0)

从包格式可以看出，DMA_0 和 DMA LOST 数据包均为 32 位。以下为各个域的含义：

- **format** 表示此次数据包的类型：
  - 0：DMA_0 数据包
  - 1：DMA LOST 数据包
- **addr_offset** 记录了此次写操作的地址偏移，实际地址 = addr_offset + {MEM_MONITOR_LOG_MIN_REG[31:2]，2'b0}。
- **dma_source** 记录了 DMA 访问来源，详见表 20.5-5。
- **lost_source** 记录了 DMA LOST 数据包生成时舍弃了哪些数据包：
  - 0：未舍弃 DMA_0 数据包
  - 1：舍弃了 DMA_0 数据包
- **anchored** 表示该 32 位数据在数据包中的位置
  - 0：该 32 位数据是数据包的低 32 位
  - 1：该 32 位数据是数据包的高 32 位

模块内部 buffer 的数据宽度是 32 位。若使能 DMA_0 通道组总线访问记录，并且在某一时刻生成了记录数据，DMA_0 的数据包会缓存到 buffer 中。辅助调试模块会自动将缓存数据从 buffer 取出，并将取出的数据再以 32 位数据宽度存入指定范围内存。

数据包解析流程如下。

- 读取 MEM_MONITOR_LOG_MEM_FULL_FLAG 确定数据是否溢出配置范围。

  - 如果未溢出，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG – 4；
  - 若溢出并且开启了 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG 和 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG – 4；
  - 若溢出并且未开启 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG。
- 从起始地址处开始读取数据并解析，每次读取 32 位。

数据解析完成之后需要通过置位 MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 清除 MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

20.6 寄存器列表

本小节中**总线记录配置寄存器**的地址为相对于存储器访问监控 1 (TCM_MEM_MONITOR) 基地址和存储器访问监控 2 (PSRAM_MEM_MONITOR) 基地址的地址偏移量（相对地址），详见小节 20.6.1，其它寄存器的所有地址均为相对于总线访问监控 (BUS_MONITOR) 基地址的地址偏移量（相对地址），详见小节 20.6.2，具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

20.6.1 总线记录配置寄存器列表

名称	描述	地址	访问
总线记录配置寄存器			
MEM_MONITOR_LOG_SETTING_REG	总线访问记录配置寄存器	0x0000	R/W
MEM_MONITOR_LOG_SETTING1_REG	总线访问记录配置寄存器	0x0004	R/W
MEM_MONITOR_LOG_CHECK_DATA_REG	总线访问监测数据配置寄存器	0x0008	R/W
MEM_MONITOR_LOG_DATA_MASK_REG	总线访问监测数据屏蔽配置寄存器	0x000C	R/W
MEM_MONITOR_LOG_MIN_REG	总线访问监测地址下边界配置寄存器	0x0010	R/W
MEM_MONITOR_LOG_MAX_REG	总线访问监测地址上边界配置寄存器	0x0014	R/W
MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG	HP CPU 总线访问监测范围更新寄存器	0x0018	WT
MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG	DMA_0 总线访问监测范围更新寄存器	0x001C	WT
MEM_MONITOR_LOG_MEM_START_REG	记录数据写入内存的起始地址	0x0020	R/W
MEM_MONITOR_LOG_MEM_END_REG	记录数据写入内存的结束地址	0x0024	R/W
MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	表示下一次写内存的写地址	0x0028	RO
MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	更新下一次写内存的写地址为记录数据写入内存的起始地址	0x002C	WT
MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	记录溢出状态寄存器	0x0030	varies
时钟控制寄存器			
MEM_MONITOR_CLOCK_GATE_REG	寄存器时钟控制	0x0034	R/W
版本寄存器			
MEM_MONITOR_DATE_REG	版本寄存器	0x03FC	R/W

GoBack

20.6.2 其他寄存器列表

名称	描述	地址	访问
监测配置寄存器			
BUS_MONITOR_CORE_0_MONTR_ENA_REG	HP CPU 监测使能配置寄存器	0x0000	R/W
BUS_MONITOR_CORE_0_AREA_DRAM0_0_MIN_REG	数据总线区域 0 监测地址下边界配置寄存器	0x0010	R/W
BUS_MONITOR_CORE_0_AREA_DRAM0_0_MAX_REG	数据总线区域 0 监测地址上边界配置寄存器	0x0014	R/W
BUS_MONITOR_CORE_0_AREA_DRAM0_1_MIN_REG	数据总线区域 1 监测地址下边界配置寄存器	0x0018	R/W
BUS_MONITOR_CORE_0_AREA_DRAM0_1_MAX_REG	数据总线区域 1 监测地址上边界配置寄存器	0x001C	R/W
BUS_MONITOR_CORE_0_AREA_PIF_0_MIN_REG	外设总线区域 0 监测地址下边界配置寄存器	0x0020	R/W
BUS_MONITOR_CORE_0_AREA_PIF_0_MAX_REG	外设总线区域 0 监测地址上边界配置寄存器	0x0024	R/W
BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG	外设总线区域 1 监测地址下边界配置寄存器	0x0028	R/W
BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG	外设总线区域 1 监测地址上边界配置寄存器	0x002C	R/W
BUS_MONITOR_CORE_0_AREA_PC_REG	HP CPU 区域监测 PC 状态寄存器	0x0030	RO
BUS_MONITOR_CORE_0_AREA_SP_REG	HP CPU 区域监测栈指针状态寄存器	0x0034	RO
BUS_MONITOR_CORE_0_SP_MIN_REG	栈指针监测地址下边界配置寄存器	0x0038	R/W
BUS_MONITOR_CORE_0_SP_MAX_REG	栈指针监测地址上边界配置寄存器	0x003C	R/W
BUS_MONITOR_CORE_0_SP_PC_REG	HP CPU 栈指针监测 PC 状态寄存器	0x0040	RO
中断配置寄存器			
BUS_MONITOR_CORE_0_INTR_RAW_REG	HP CPU 监测原始中断状态寄存器	0x0004	RO
BUS_MONITOR_CORE_0_INTR_ENA_REG	HP CPU 监测中断使能配置寄存	0x0008	R/W
BUS_MONITOR_CORE_0_INTR_CLR_REG	HP CPU 监测清除中断配置寄存器	0x000C	WT
PC 记录配置寄存器			
BUS_MONITOR_CORE_0_RCD_EN_REG	PC 记录使能配置寄存器	0x0044	R/W
PC 记录状态寄存器			
BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG	HP CPU PC 记录寄存器	0x0048	RO
BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG	HP CPU 栈指针记录寄存器	0x004C	RO
CPU 状态寄存器			
BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	HP CPU 异常前的最后一条指令的 PC	0x0070	RO

名称	描述	地址	访问
BUS_MONITOR_CORE_0_DEBUG_MODE_REG	HP CPU 处于调试模式的标志寄存器	0x0074	RO
时钟控制寄存器			
BUS_MONITOR_CLOCK_GATE_REG	寄存器时钟控制	0x0108	R/W
版本寄存器			
BUS_MONITOR_DATE_REG	版本寄存器	0x03FC	R/W

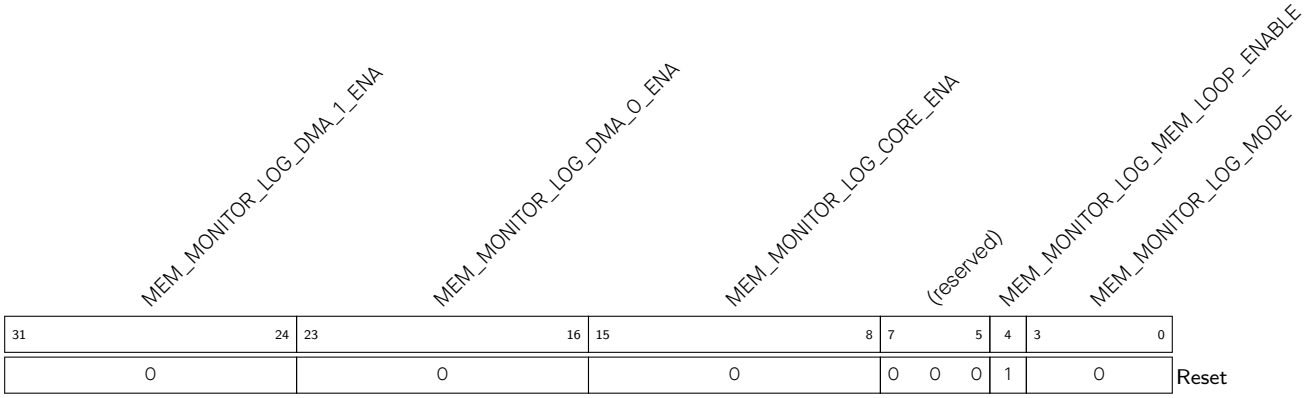
20.7 寄存器

本小节中**总线记录配置寄存器**的地址为相对于存储器访问监控 1 (TCM_MEM_MONITOR) 基地址和存储器访问监控 2 (PSRAM_MEM_MONITOR) 基地址的地址偏移量（相对地址），详见小节 20.7.1，其它寄存器的所有地址均为相对于总线访问监控 (BUS_MONITOR) 基地址的地址偏移量（相对地址），详见小节 20.7.2，具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

20.7.1 总线记录配置寄存器

Register 20.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)



MEM_MONITOR_LOG_MODE 配置监测模式。

1：使能写监测
2：使能 word 监测
4：使能 halfword 监测
8：使能 byte 监测
其他值：无效
(R/W)

MEM_MONITOR_LOG_MEM_LOOP_ENABLE 配置写内存的存储模式。

0：非 loop 模式
1：loop 模式
(R/W)

MEM_MONITOR_LOG_CORE_ENA 配置是否使能 HP CPU 总线访问记录。

0：不使能
1：使能
其他值：无效
(R/W)

见下页...

Register 20.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)

接上页...

MEM_MONITOR_LOG_DMA_0_ENA 配置是否使能 DMA_0 总线访问记录。

0：不使能

1：使能

其他值：无效

(R/W)

MEM_MONITOR_LOG_DMA_1_ENA 配置是否使能 DMA_1 总线访问记录。

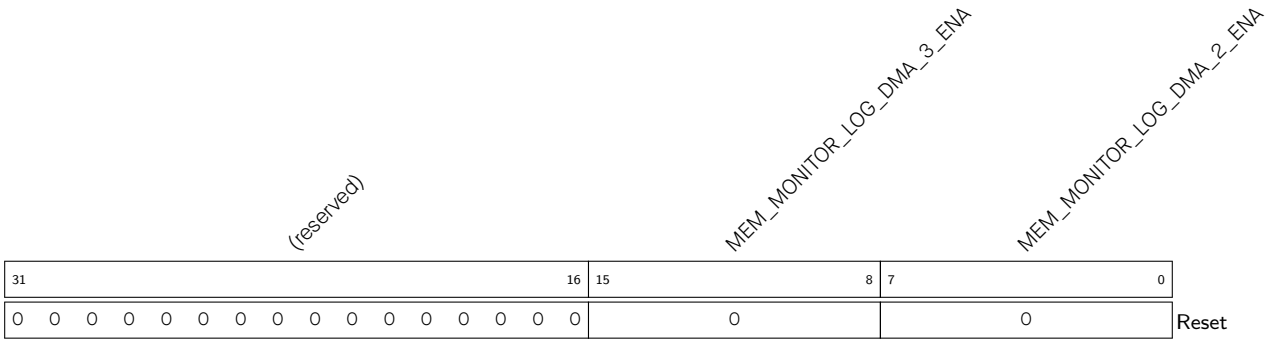
0：不使能

1：使能

其他值：无效

(R/W)

Register 20.2. MEM_MONITOR_LOG_SETTING1_REG (0x0004)



MEM_MONITOR_LOG_DMA_2_ENA 配置是否使能 DMA_2 总线访问记录。

0：不使能

1：使能

其他值：无效

(R/W)

MEM_MONITOR_LOG_DMA_3_ENA 配置是否使能 DMA_3 总线访问记录。

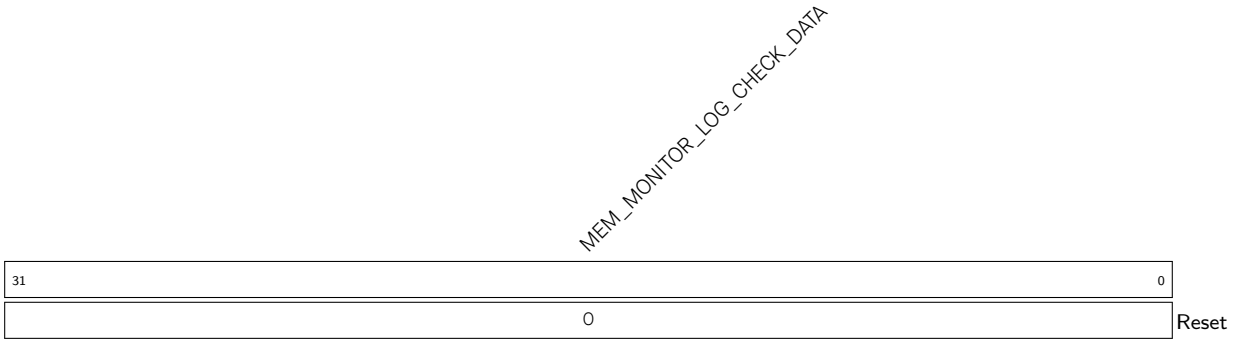
0：不使能

1：使能

其他值：无效

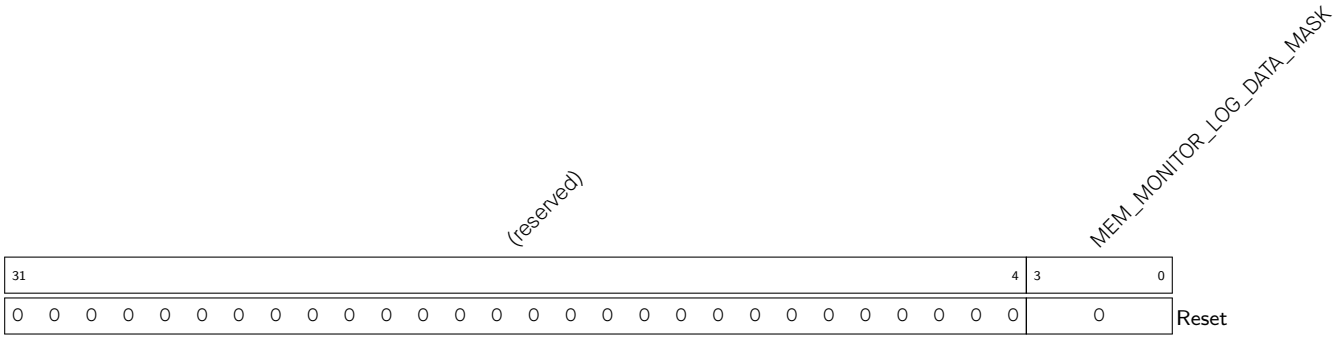
(R/W)

Register 20.3. MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



MEM_MONITOR_LOG_CHECK_DATA 配置总线访问监测的特殊值。(R/W)

Register 20.4. MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



MEM_MONITOR_LOG_DATA_MASK 配置是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 的相应字节。可同时屏蔽多个字节。

bit[0]：配置是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 最低位字节。

0：不屏蔽

1：屏蔽

bit[1]：配置是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 第二个低位字节。

0：不屏蔽

1：屏蔽

bit[2]：配置是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 第二个高位字节。

0：不屏蔽

1：屏蔽

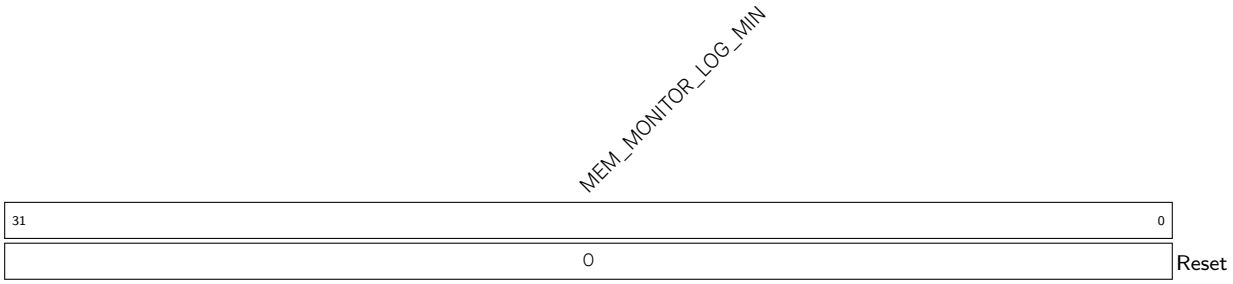
bit[3]：配置是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 最高位字节。

0：不屏蔽

1：屏蔽

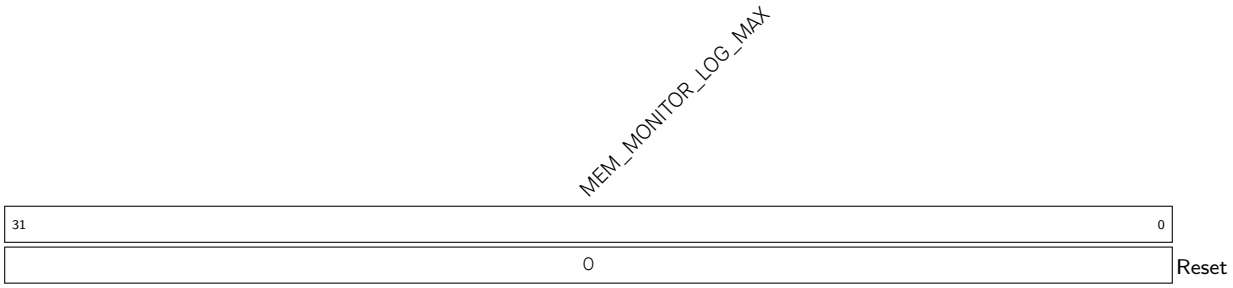
(R/W)

Register 20.5. MEM_MONITOR_LOG_MIN_REG (0x0010)



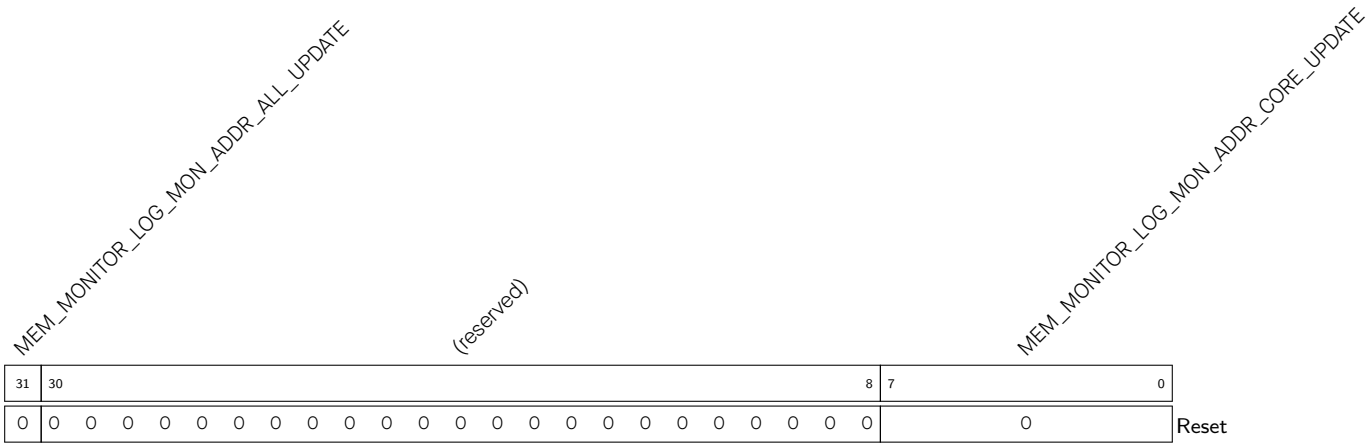
MEM_MONITOR_LOG_MIN 配置监测地址的下边界。(R/W)

Register 20.6. MEM_MONITOR_LOG_MAX_REG (0x0014)



MEM_MONITOR_LOG_MAX 配置监测地址的上边界。(R/W)

Register 20.7. MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG (0x0018)



MEM_MONITOR_LOG_MON_ADDR_CORE_UPDATE 配置是否将 MEM_MONITOR_LOG_MIN_REG 到 MEM_MONITOR_LOG_MAX_REG 的地址更新为 HP CPU 总线的监测地址空间。

0：不更新

1：更新

其他值：无效

(WT)

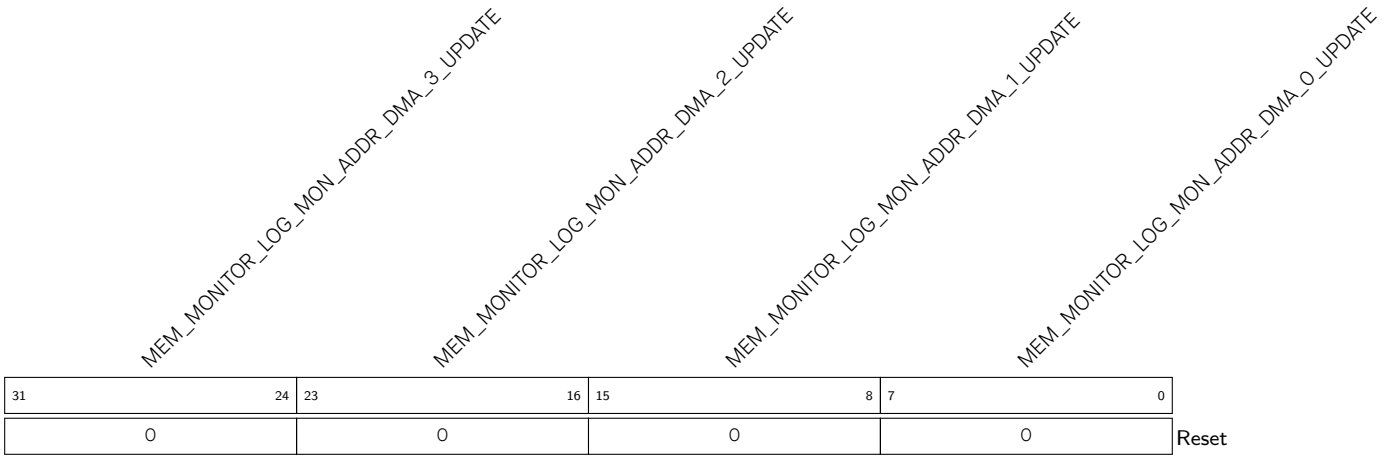
MEM_MONITOR_LOG_MON_ADDR_ALL_UPDATE 配置是否将 MEM_MONITOR_LOG_MIN_REG 到 MEM_MONITOR_LOG_MAX_REG 的地址更新为所有主机的监测地址空间。

0：不更新

1：更新

(WT)

Register 20.8. MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG (0x001C)



MEM_MONITOR_LOG_MON_ADDR_DMA_0_UPDATE MEM_MONITOR_LOG_MIN_REG 到 MEM_MONITOR_LOG_MAX_REG 的地址更新为 DMA_0 总线的监测地址空间。

0：不更新

1：更新

其他值：无效

(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_1_UPDATE MEM_MONITOR_LOG_MIN_REG 到 MEM_MONITOR_LOG_MAX_REG 的地址更新为 DMA_1 总线的监测地址空间。

0：不更新

1：更新

其他值：无效

(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_2_UPDATE MEM_MONITOR_LOG_MIN_REG 到 MEM_MONITOR_LOG_MAX_REG 的地址更新为 DMA_2 总线的监测地址空间。

0：不更新

1：更新

其他值：无效

(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_3_UPDATE MEM_MONITOR_LOG_MIN_REG 到 MEM_MONITOR_LOG_MAX_REG 的地址更新为 DMA_3 总线的监测地址空间。

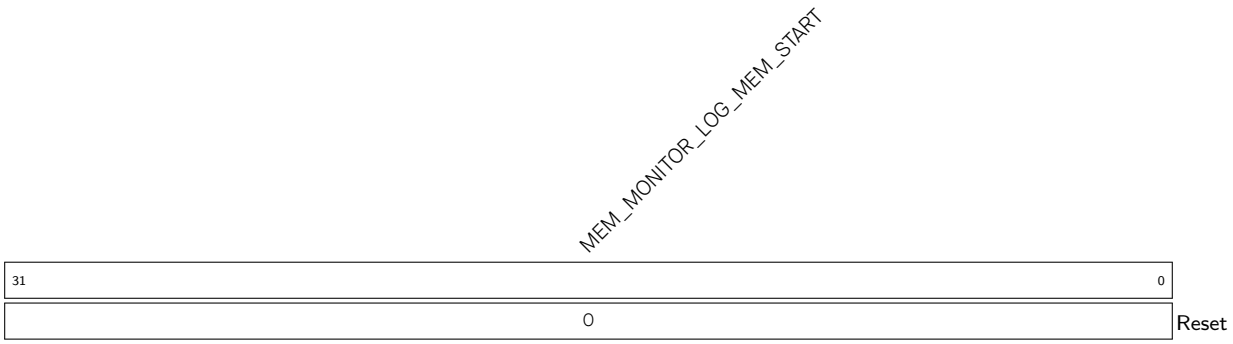
0：不更新

1：更新

其他值：无效

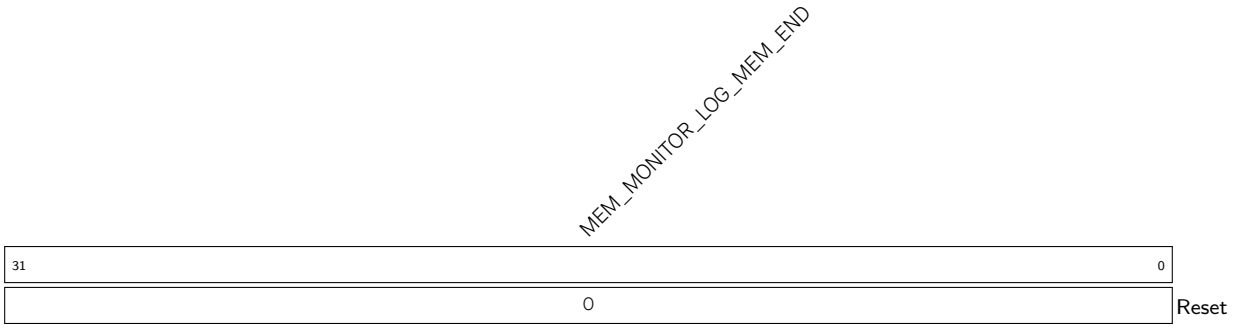
(WT)

Register 20.9. MEM_MONITOR_LOG_MEM_START_REG (0x0020)



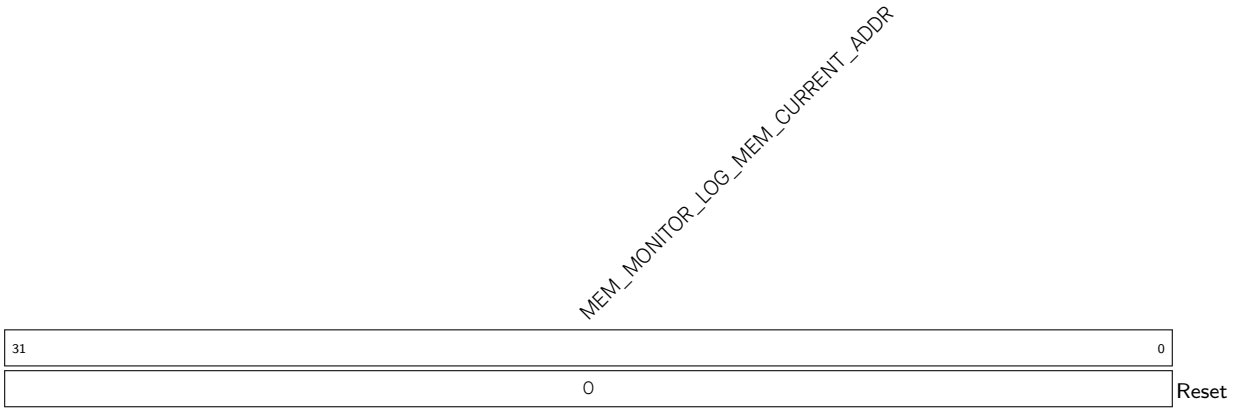
MEM_MONITOR_LOG_MEM_START 配置记录数据写入内存的起始地址。(R/W)

Register 20.10. MEM_MONITOR_LOG_MEM_END_REG (0x0024)



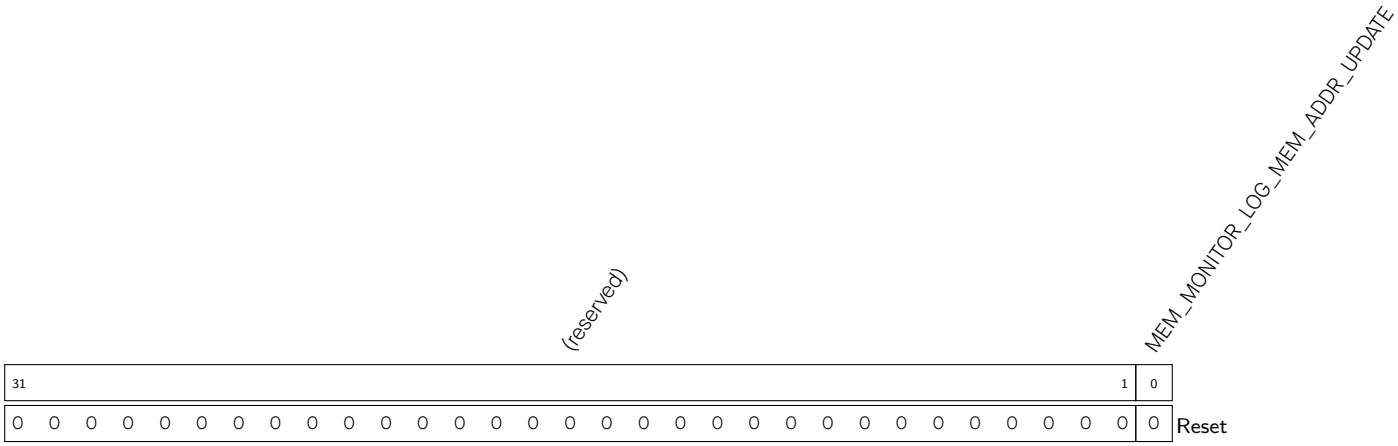
MEM_MONITOR_LOG_MEM_END 配置记录数据写入内存的结束地址。(R/W)

Register 20.11. MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0028)



MEM_MONITOR_LOG_MEM_CURRENT_ADDR 表示下一次写内存的写地址。(RO)

Register 20.12. MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x002C)



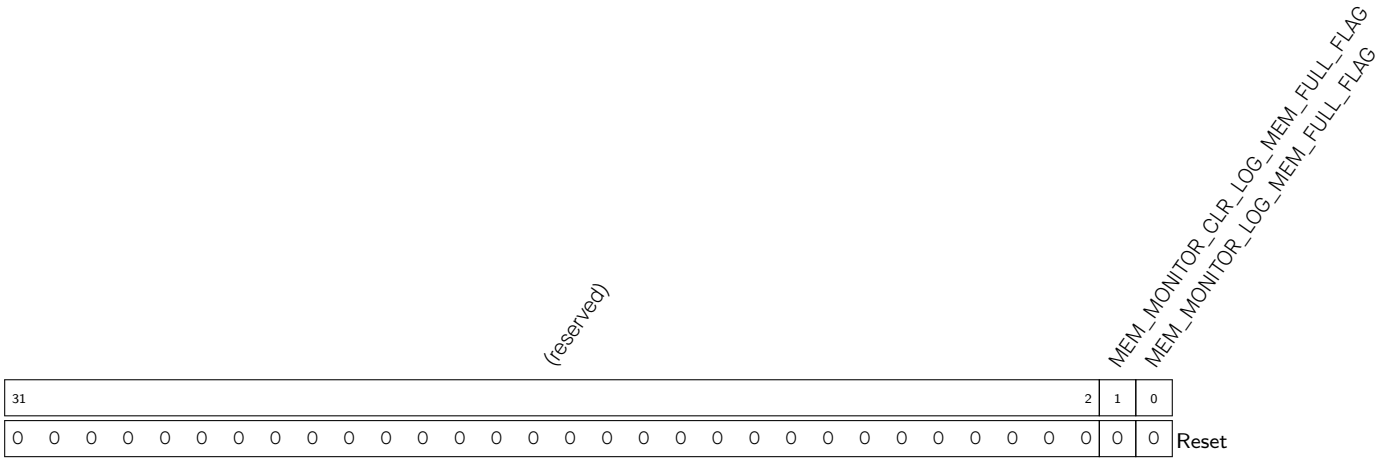
MEM_MONITOR_LOG_MEM_ADDR_UPDATE 配置是否将 MEM_MONITOR_LOG_MEM_START_REG 中的值更新为 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 中的值。

0：不更新

1：更新

(WT)

Register 20.13. MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0030)



MEM_MONITOR_LOG_MEM_FULL_FLAG 表示数据是否溢出存储地址范围。

0：未溢出

1：溢出

(RO)

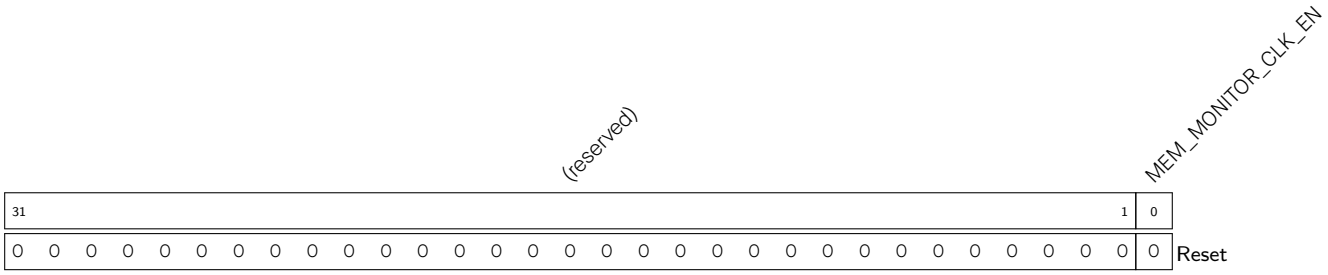
MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 配置是否清除 MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

0：不清除（默认）

1：清除

(WT)

Register 20.14. MEM_MONITOR_CLOCK_GATE_REG (0x0034)



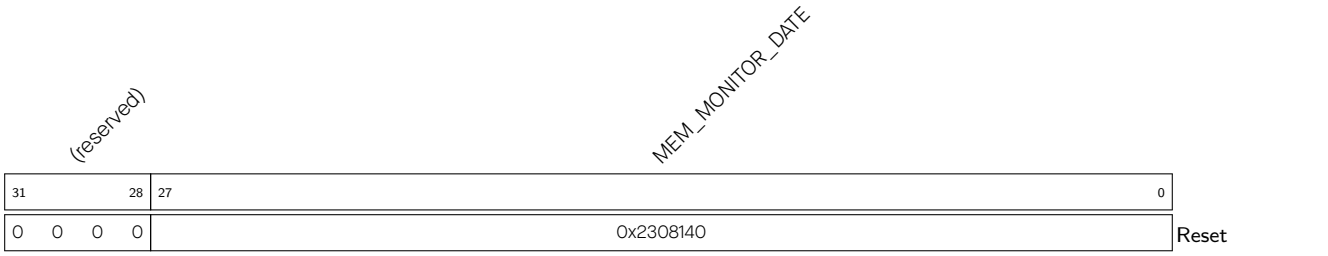
MEM_MONITOR_CLK_EN 配置是否使能寄存器时钟门控。

0：不使能

1：使能

(R/W)

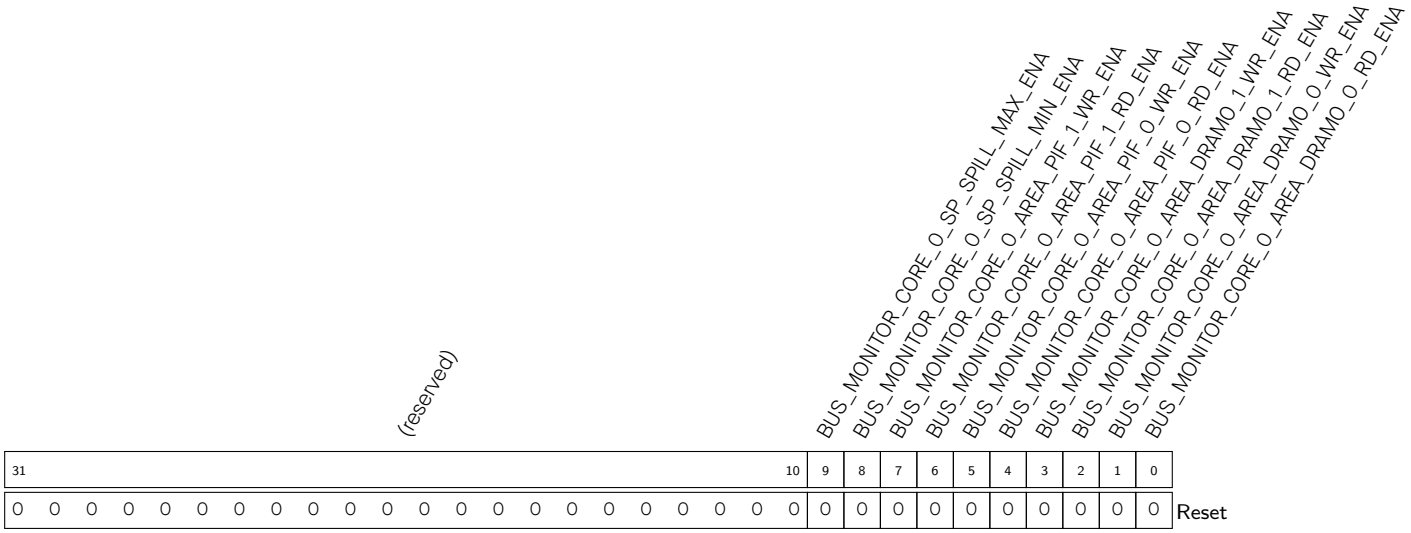
Register 20.15. MEM_MONITOR_DATE_REG (0x03FC)



MEM_MONITOR_DATE 版本控制寄存器。(R/W)

20.7.2 其他寄存器

Register 20.16. BUS_MONITOR_CORE_0_MONTR_ENA_REG (0x0000)



BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_ENA 配置是否监测数据总线在数据总线区域 0 内的读操作。

0：不监测

1：监测

(R/W)

BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_ENA 配置是否监测数据总线在数据总线区域 0 内的写操作。

0：不监测

1：监测

(R/W)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_ENA 配置是否监测数据总线在数据总线区域 1 内的读操作。

0：不监测

1：监测

(R/W)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_ENA 配置是否监测数据总线在数据总线区域 1 内的写操作。

0：不监测

1：监测

(R/W)

BUS_MONITOR_CORE_0_AREA_PIF_0_RD_ENA 配置是否监测外设总线在数据总线区域 0 内的读操作。

0：不监测

1：监测

(R/W)

见下页...

Register 20.16. BUS_MONITOR_CORE_0_MONTR_ENA_REG (0x0000)

接上页...

BUS_MONITOR_CORE_0_AREA_PIF_0_WR_ENA 配置是否监测外设总线在数据总线区域 0 内的写操作。
0：不监测
1：监测
(R/W)

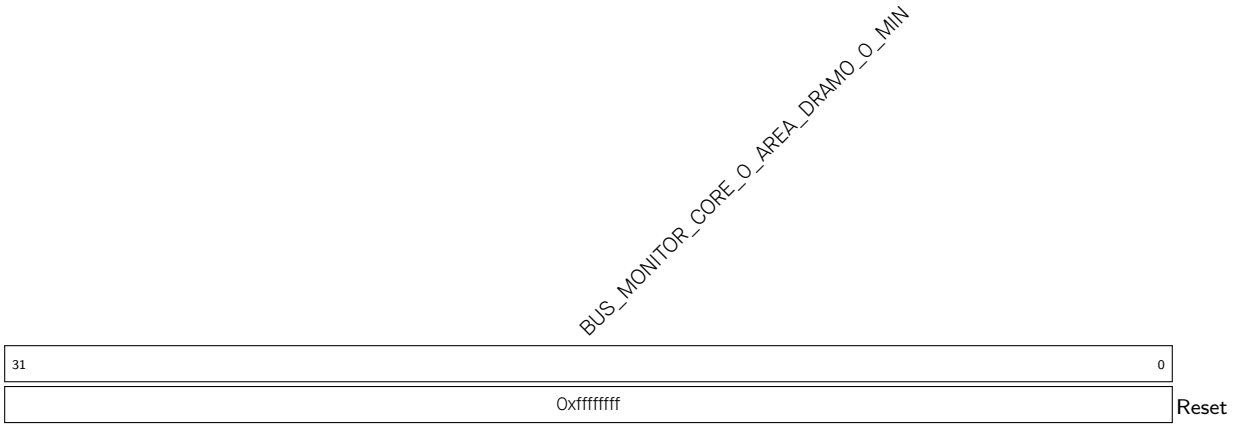
BUS_MONITOR_CORE_0_AREA_PIF_1_RD_ENA 配置是否监测外设总线在数据总线区域 1 内的读操作。
0：不监测
1：监测
(R/W)

BUS_MONITOR_CORE_0_AREA_PIF_1_WR_ENA 配置是否监测外设总线在数据总线区域 1 内的写操作。
0：不监测
1：监测
(R/W)

BUS_MONITOR_CORE_0_SP_SPILL_MIN_ENA 配置是否监测栈指针小于栈指针监测区域的下边界。
0：不监测
1：监测
(R/W)

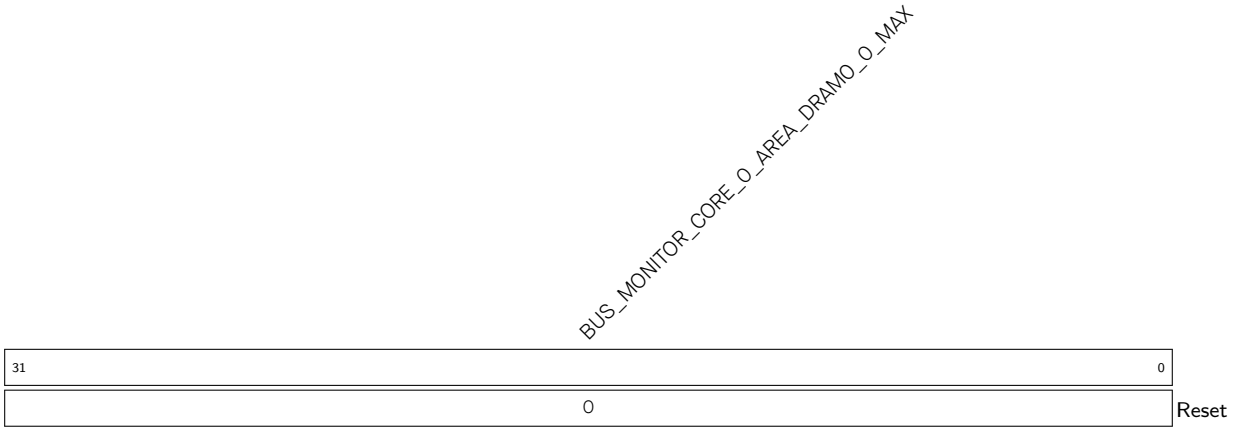
BUS_MONITOR_CORE_0_SP_SPILL_MAX_ENA 配置是否监测栈指针大于栈指针监测区域的上边界。
0：不监测
1：监测
(R/W)

Register 20.17. BUS_MONITOR_CORE_0_AREA_DRAM0_0_MIN_REG (0x0010)



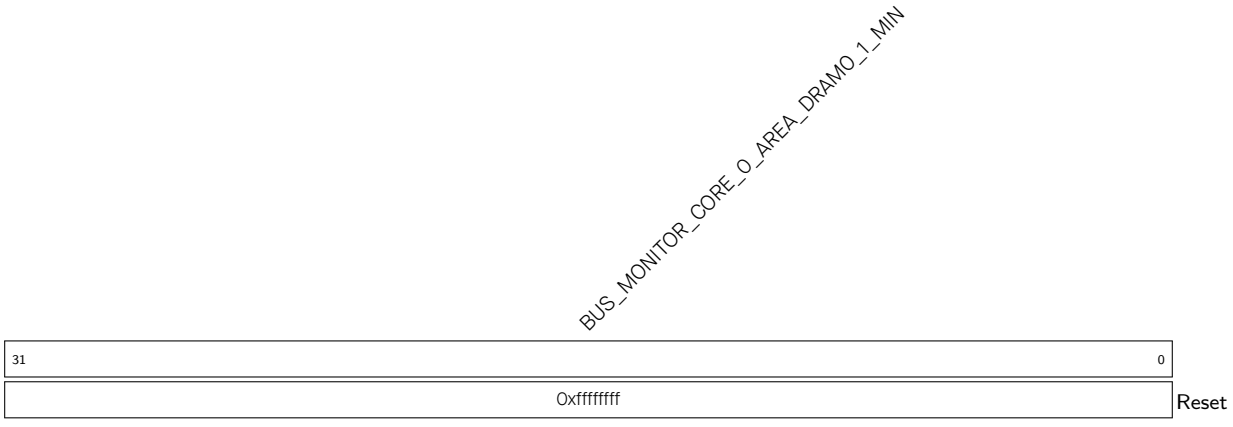
BUS_MONITOR_CORE_0_AREA_DRAM0_0_MIN 配置数据总线区域 0 的下边界。(R/W)

Register 20.18. BUS_MONITOR_CORE_0_AREA_DRAM0_0_MAX_REG (0x0014)



BUS_MONITOR_CORE_0_AREA_DRAM0_0_MAX 配置数据总线区域 0 的上边界。(R/W)

Register 20.19. BUS_MONITOR_CORE_0_AREA_DRAM0_1_MIN_REG (0x0018)



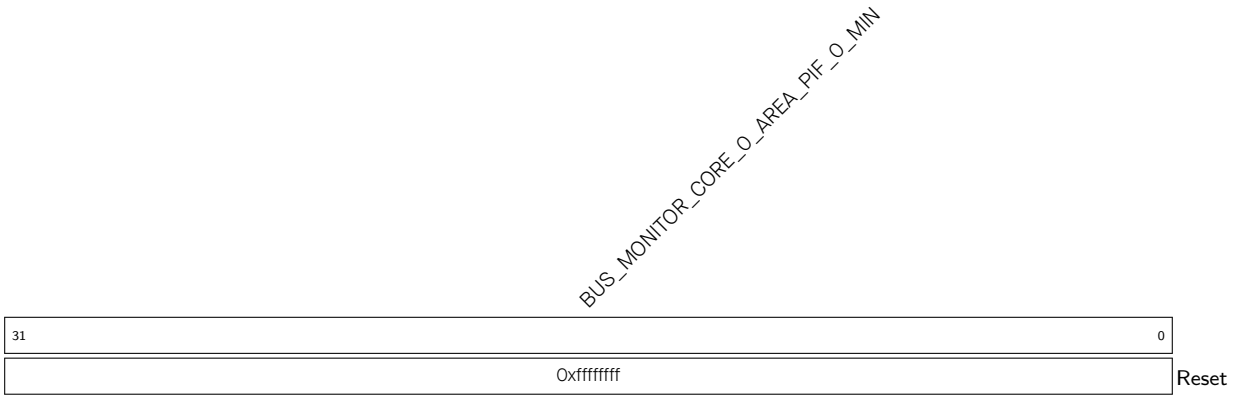
BUS_MONITOR_CORE_0_AREA_DRAM0_1_MIN 配置数据总线区域 1 的下边界。(R/W)

Register 20.20. BUS_MONITOR_CORE_0_AREA_DRAM0_1_MAX_REG (0x001C)



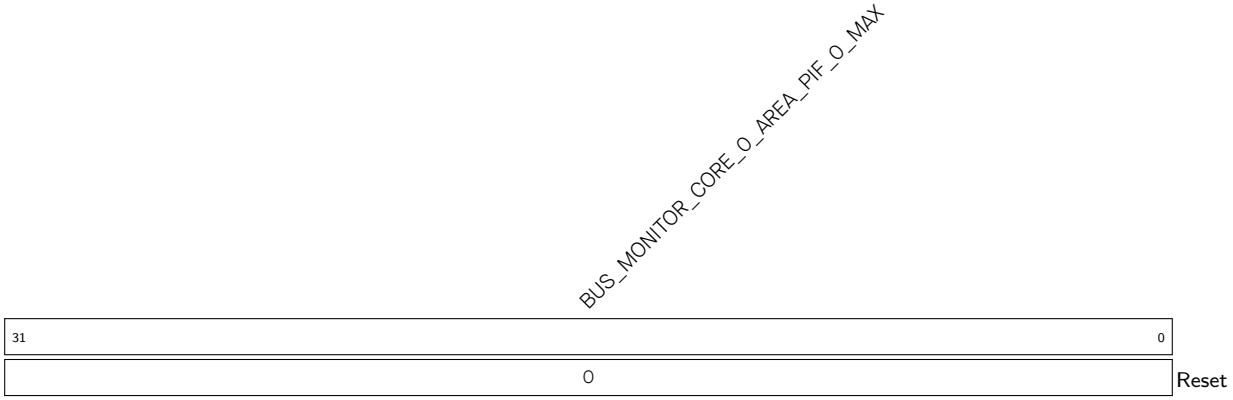
BUS_MONITOR_CORE_0_AREA_DRAM0_1_MAX 配置数据总线区域 1 的上边界。(R/W)

Register 20.21. BUS_MONITOR_CORE_0_AREA_PIF_0_MIN_REG (0x0020)



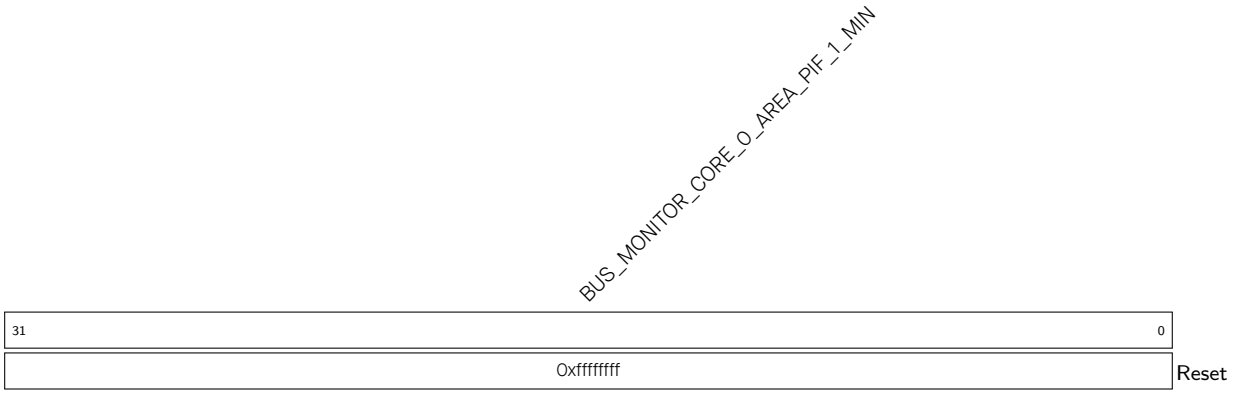
BUS_MONITOR_CORE_0_AREA_PIF_0_MIN 配置外设总线区域 0 的下边界。(R/W)

Register 20.22. BUS_MONITOR_CORE_0_AREA_PIF_0_MAX_REG (0x0024)



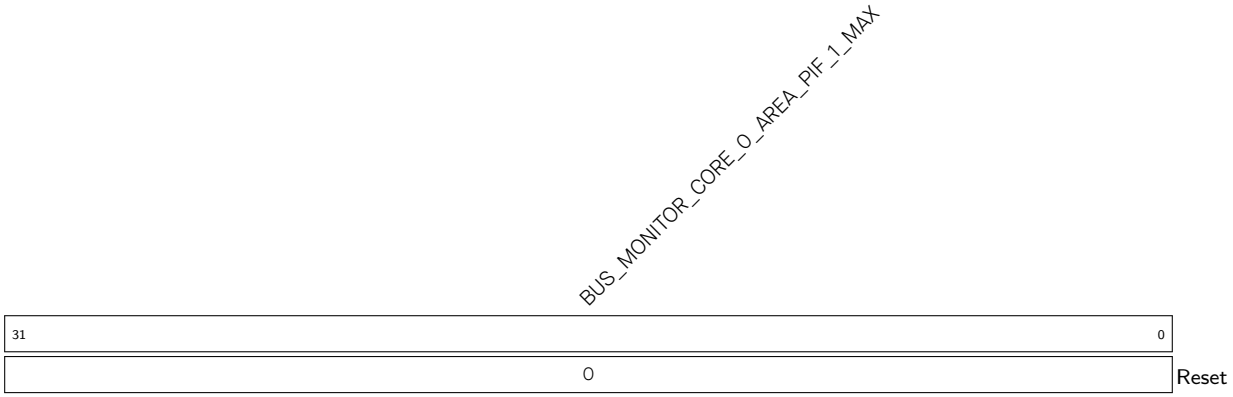
BUS_MONITOR_CORE_0_AREA_PIF_0_MAX 配置外设总线区域 0 的上边界。(R/W)

Register 20.23. BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG (0x0028)



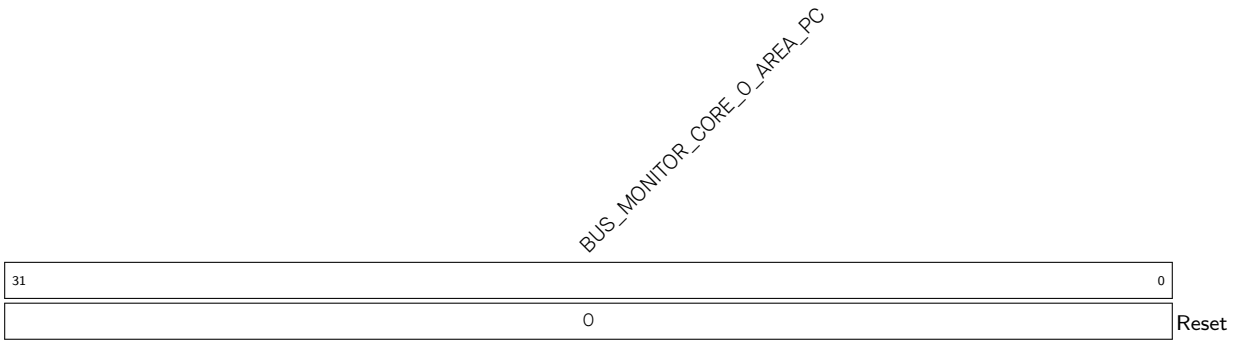
BUS_MONITOR_CORE_0_AREA_PIF_1_MIN 配置外设总线区域 1 的下边界。(R/W)

Register 20.24. BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG (0x002C)



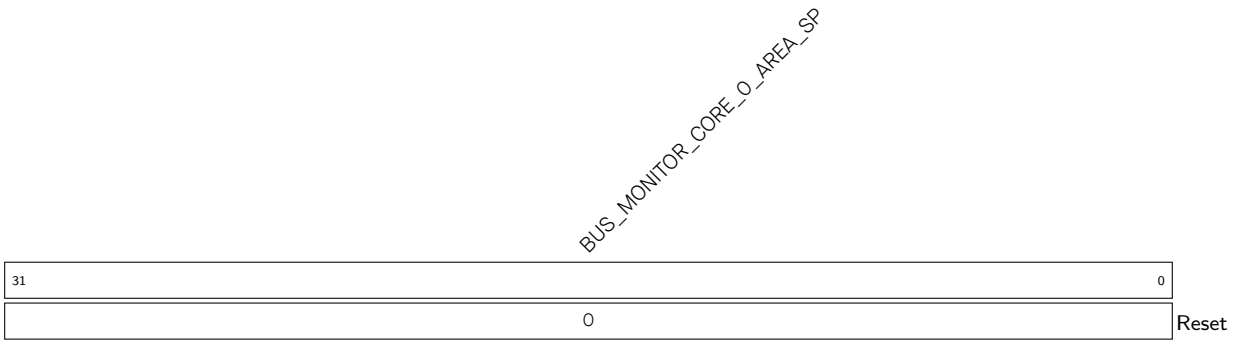
BUS_MONITOR_CORE_0_AREA_PIF_1_MAX 配置外设总线区域 1 的上边界。(R/W)

Register 20.25. BUS_MONITOR_CORE_0_AREA_PC_REG (0x0030)



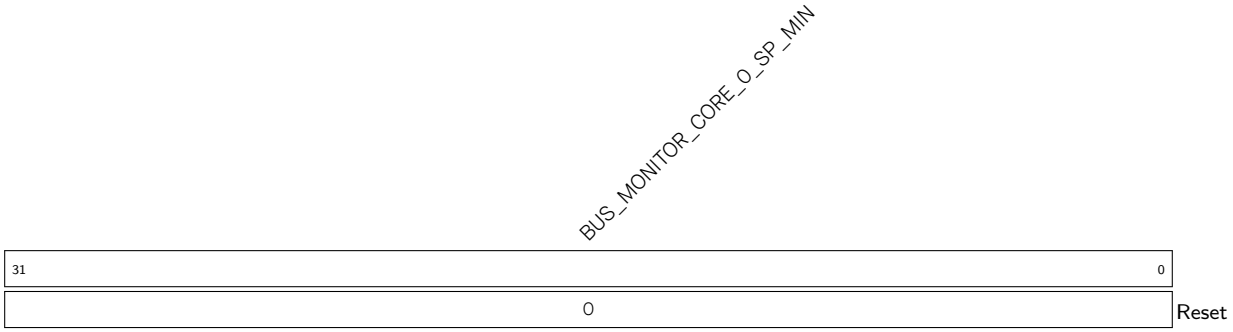
BUS_MONITOR_CORE_0_AREA_PC 表示区域监测下触发中断时刻的 PC 值。(RO)

Register 20.26. BUS_MONITOR_CORE_0_AREA_SP_REG (0x0034)



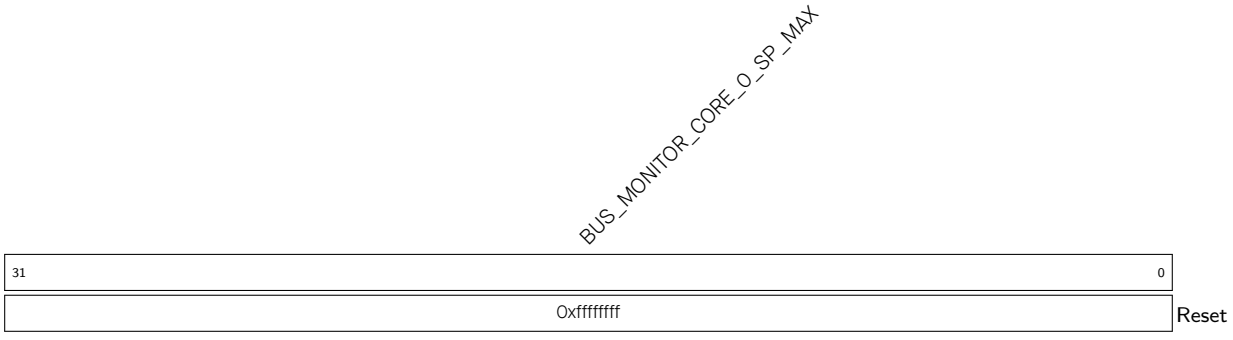
BUS_MONITOR_CORE_0_AREA_SP 表示区域监测下触发中断时刻的栈指针。(RO)

Register 20.27. BUS_MONITOR_CORE_0_SP_MIN_REG (0x0038)



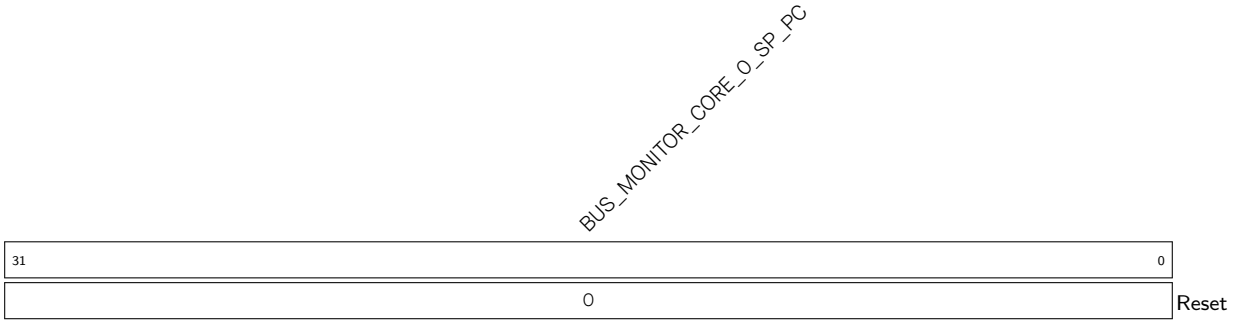
BUS_MONITOR_CORE_0_SP_MIN 配置栈指针监测区域的下边界。(R/W)

Register 20.28. BUS_MONITOR_CORE_0_SP_MAX_REG (0x003C)



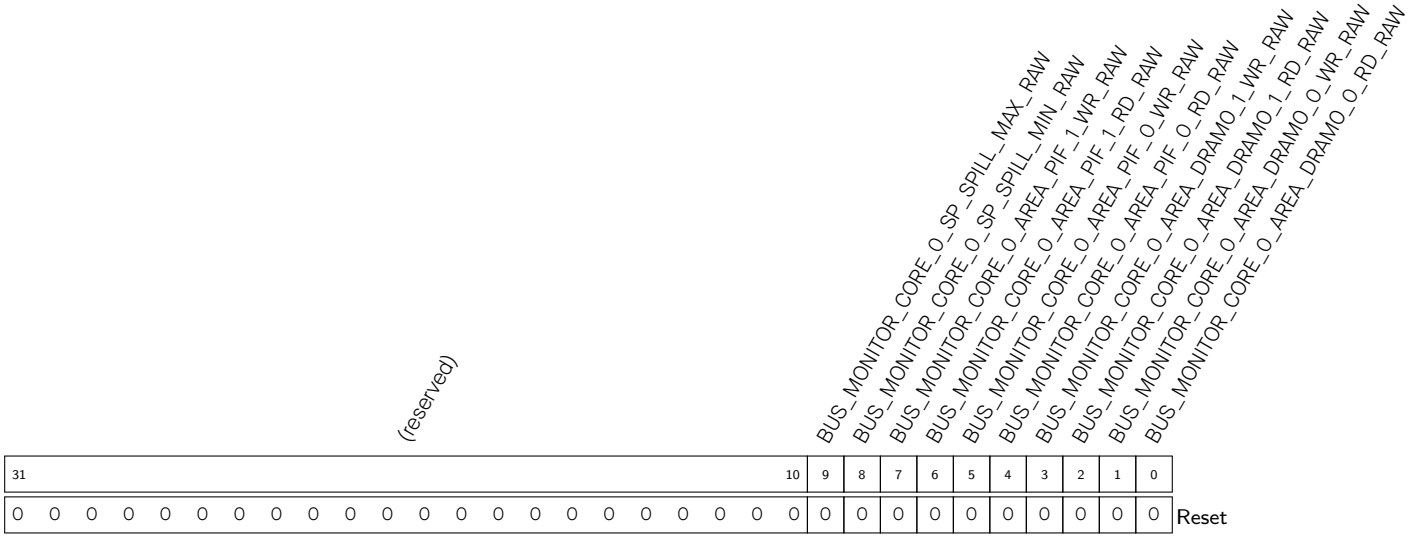
BUS_MONITOR_CORE_0_SP_MAX 配置栈指针监测区域的上边界。(R/W)

Register 20.29. BUS_MONITOR_CORE_0_SP_PC_REG (0x0040)



BUS_MONITOR_CORE_0_SP_PC 表示栈指针监测的 PC 值。(RO)

Register 20.30. BUS_MONITOR_CORE_0_INTR_RAW_REG (0x0004)



BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_RAW BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_RAW BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_RAW BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_RAW BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_AREA_PIF_0_RD_RAW BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_AREA_PIF_0_WR_RAW BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT 的原始中断状态。(RO)

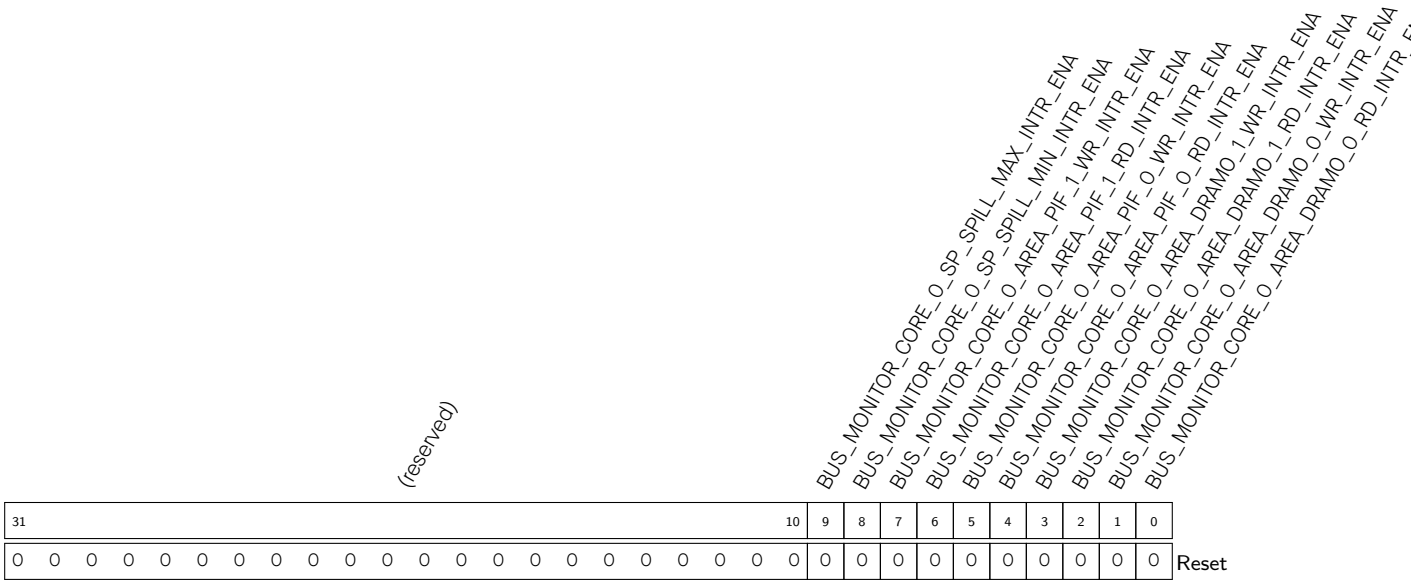
BUS_MONITOR_CORE_0_AREA_PIF_1_RD_RAW BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_AREA_PIF_1_WR_RAW BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_SP_SPILL_MIN_RAW BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT 的原始中断状态。(RO)

BUS_MONITOR_CORE_0_SP_SPILL_MAX_RAW BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT 的原始中断状态。(RO)

Register 20.31. BUS_MONITOR_CORE_0_INTR_ENA_REG (0x0008)



BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT。(R/W)

BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT。(R/W)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT。(R/W)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT。(R/W)

BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT。(R/W)

BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT。(R/W)

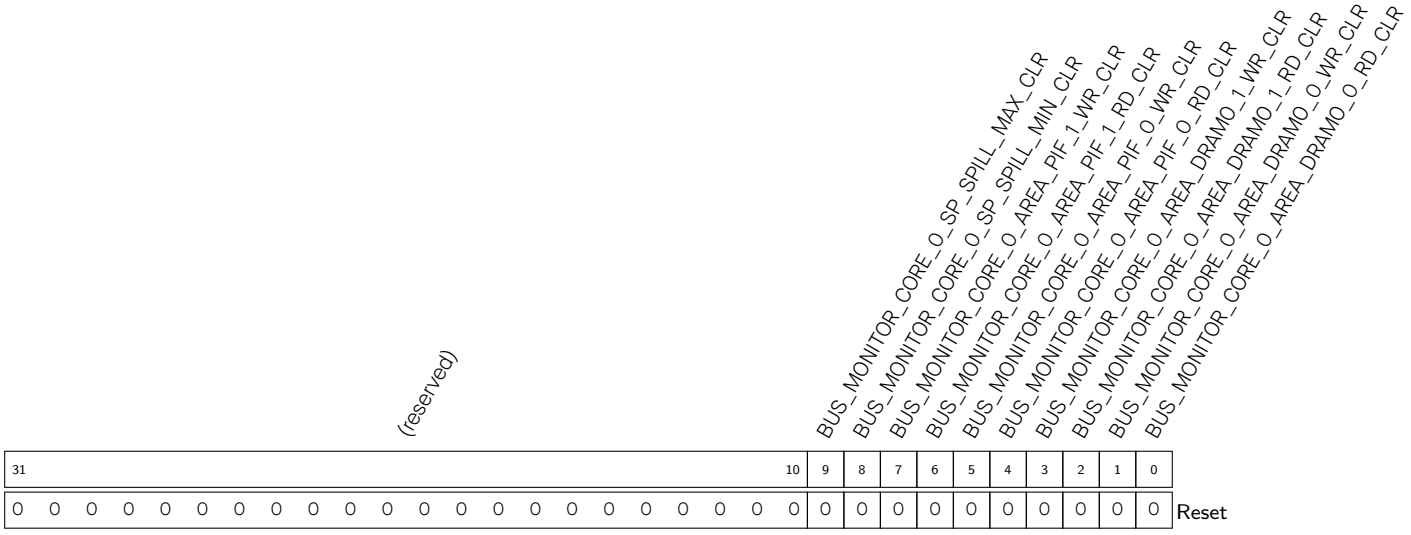
BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT。(R/W)

BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT。(R/W)

BUS_MONITOR_CORE_0_SP_SPILL_MIN_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT。(R/W)

BUS_MONITOR_CORE_0_SP_SPILL_MAX_INTR_ENA 写 1 使能 BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT。(R/W)

Register 20.32. BUS_MONITOR_CORE_0_INTR_CLR_REG (0x000C)



BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_DRAM0_0_RD_INT。(WT)

BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_DRAM0_0_WR_INT。(WT)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_DRAM0_1_RD_INT。(WT)

BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_DRAM0_1_WR_INT。(WT)

BUS_MONITOR_CORE_0_AREA_PIF_0_RD_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT。(WT)

BUS_MONITOR_CORE_0_AREA_PIF_0_WR_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT。(WT)

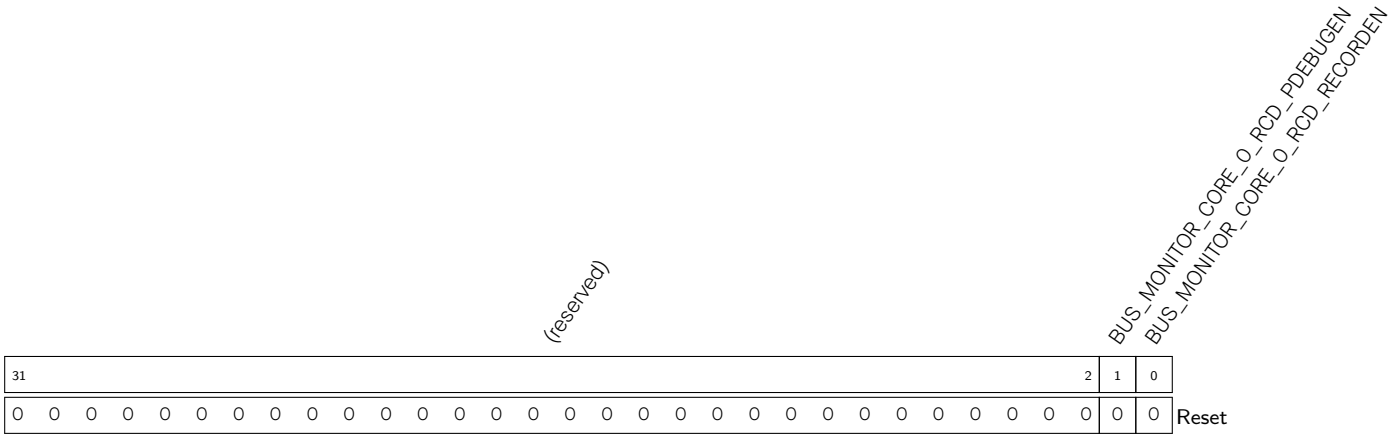
BUS_MONITOR_CORE_0_AREA_PIF_1_RD_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT。(WT)

BUS_MONITOR_CORE_0_AREA_PIF_1_WR_CLR 写 1 清除 BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT。(WT)

BUS_MONITOR_CORE_0_SP_SPILL_MIN_CLR 写 1 清除 BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT。(WT)

BUS_MONITOR_CORE_0_SP_SPILL_MAX_CLR 写 1 清除 BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT。(WT)

Register 20.33. BUS_MONITOR_CORE_0_RCD_EN_REG (0x0044)



BUS_MONITOR_CORE_0_RCD_RECORDEN 配置是否使能 PC 记录。

0：不使能

1：BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG 开始实时记录 PC

(R/W)

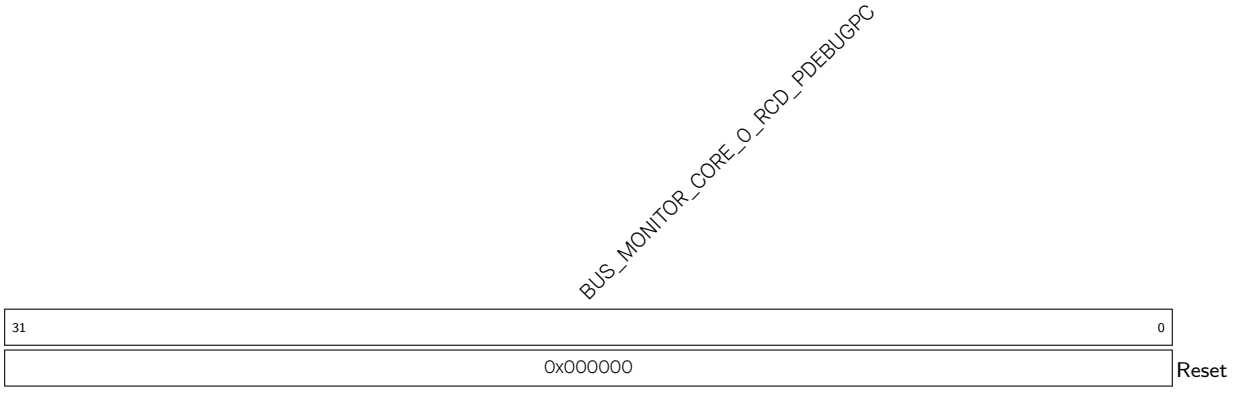
BUS_MONITOR_CORE_0_RCD_PDEBUGEN 配置是否使能 HP CPU 调试。

0：不使能

1：HP CPU 输出 PC

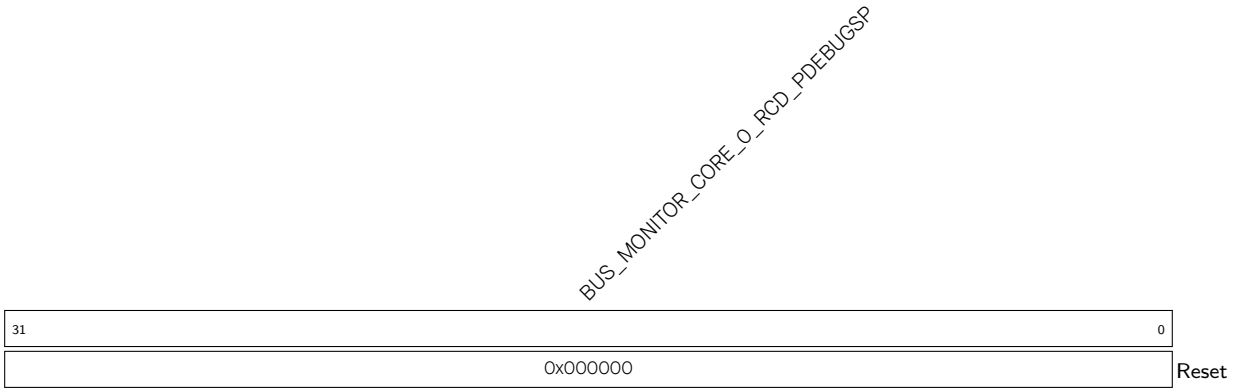
(R/W)

Register 20.34. BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG (0x0048)



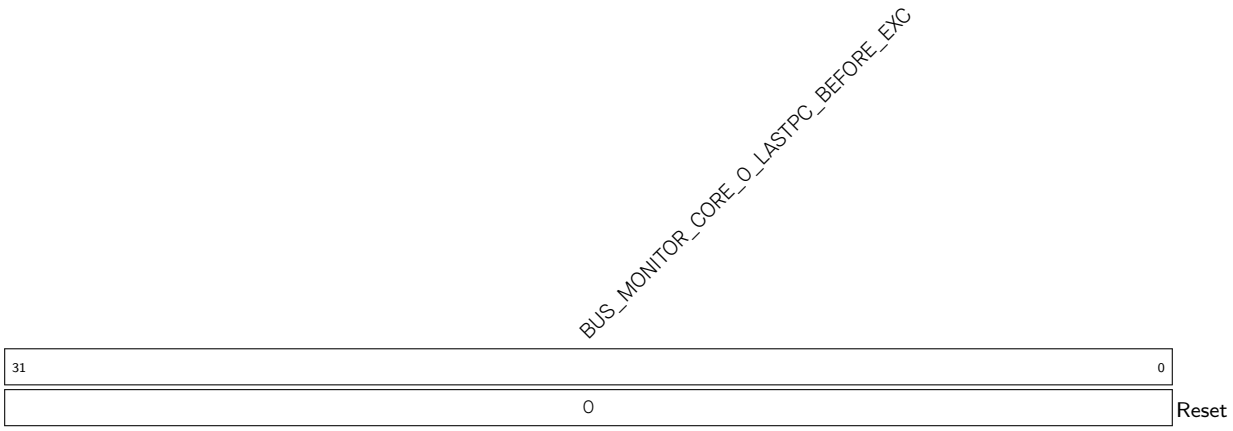
BUS_MONITOR_CORE_0_RCD_PDEBUGPC 表示 HP CPU 复位时刻的 PC 值。(RO)

Register 20.35. BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG (0x004C)



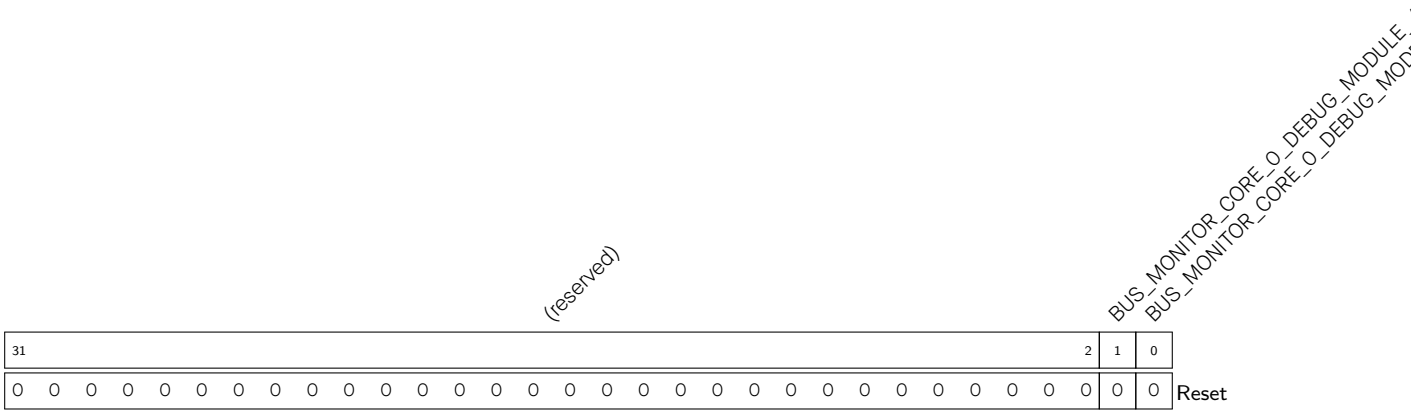
BUS_MONITOR_CORE_0_RCD_PDEBUGSP 表示栈指针。(RO)

Register 20.36. BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)



BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXC 表示 HP CPU 异常前的最后一条指令的 PC。(RO)

Register 20.37. BUS_MONITOR_CORE_0_DEBUG_MODE_REG (0x0074)



BUS_MONITOR_CORE_0_DEBUG_MODE 表示 RISC-V CPU (HP CPU) 是否处于调试模式。

1：处于调试模式

0：未处于调试模式

(RO)

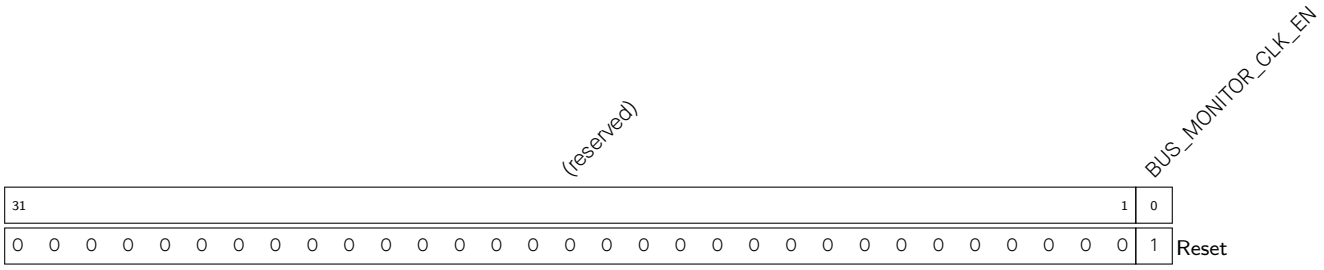
BUS_MONITOR_CORE_0_DEBUG_MODULE_ACTIVE 表示 RISC-V CPU (HP CPU) 调试模块的工作状态。

1：处于工作状态

其它：未处于工作状态

(RO)

Register 20.38. BUS_MONITOR_CLOCK_GATE_REG (0x0108)



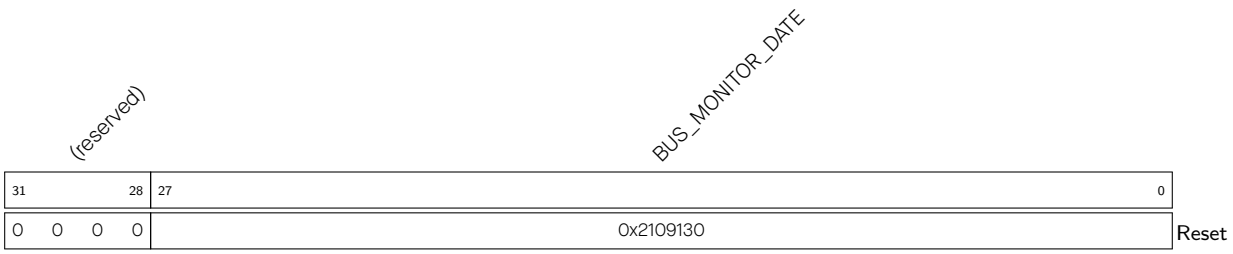
BUS_MONITOR_CLK_EN 配置是否使能寄存器时钟门控。

0：不使能

1：使能

(R/W)

Register 20.39. BUS_MONITOR_DATE_REG (0x03FC)



BUS_MONITOR_DATE 版本控制寄存器。(R/W)

第 21 章

电源检测器

21.1 概述

ESP32-C5 芯片上的电源检测器能够监测片上时钟和数字电路相关的电源管脚上的电压。它包含一个欠压检测器和四个电压毛刺保护器。欠压检测器防止欠压复位，电压毛刺保护器防止电压毛刺攻击。电源检测器属于 always-on（常开）电源区域，可以在任何时刻监控电压。

21.2 主要特性

- 欠压检测器支持两种欠压检测模式（模式 0 和模式 1）
  - 模式 0 支持如下动作：
    * 发出中断
    * 关闭射频电路
    * 请求 flash 进入 suspend 状态
    * 发出系统复位
  - 模式 1 支持如下动作：
    * 发出系统复位直到电压恢复正常
- 电压毛刺保护器能够检测约 50 ns 以上的电压毛刺并触发系统复位。

21.3 功能描述

21.3.1 架构图

图 21.3-1 为电源检测器的架构图。由图可知，电源检测器包含一个欠压检测器 (Brown-out Detector) 和四个电压毛刺保护器 (Voltage Glitch Detector)。

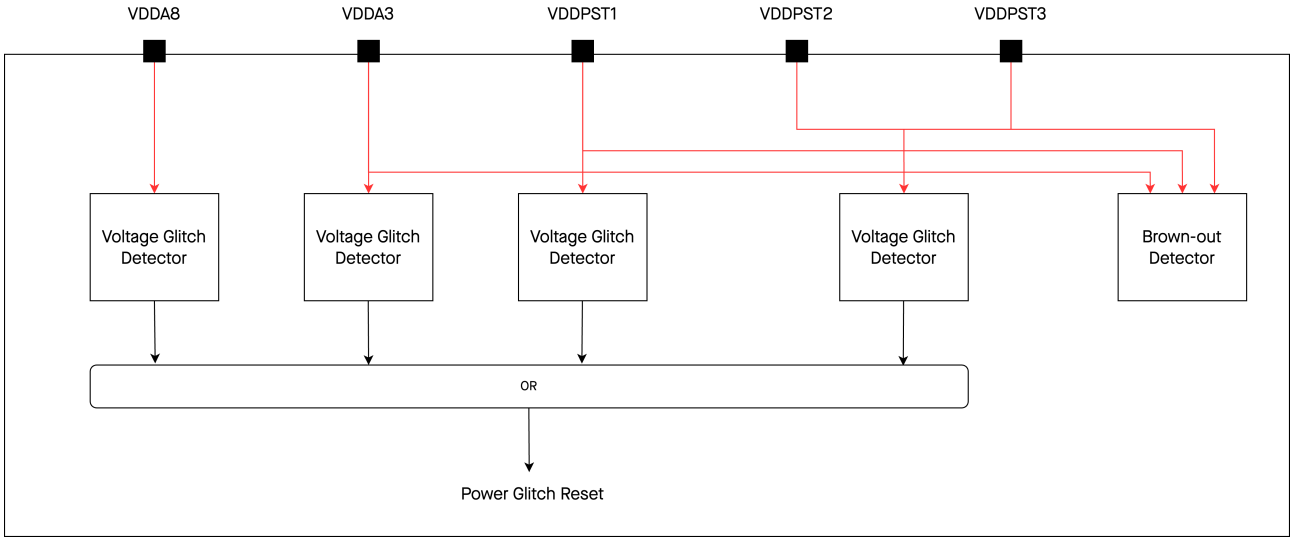


图 21.3-1. 电源检测器架构图

21.3.2 欠压检测器

ESP32-C5 的欠压检测器可以周期性轮流检查管脚 VDDA3、VDDPST1、VDDPST2、VDDPST3 的电压，周期约 280 μs。一旦被检测电压下落至预设阈值（默认为 2.7 V）以下时，欠压检测器发出触发信号，并进行相应处理，关闭功耗较高的射频电路，为数字模块争取更多时间，用以保存、转移重要数据。欠压检测器的功耗非常低，在芯片开启时将永远保持开启。

LP_ANA_BOD_MODE0_INT_RAW 用于指示欠压检测器的输出结果，默认为 0，可在检测管脚电压下降至阈值以下时置位。

欠压检测器检测到欠压信号后，有两种处理模式（默认为模式 1）：

- 模式 0：
  - 此模式需通过 `bod_mode0_en` (LP_ANA_BOD_MODE0_INTR_ENA) 使能。
  - 当欠压计数器达到 Int Comparer (LP_ANA_BOD_MODE0_INTR_WAIT) 中设定的阈值后触发中断，如果使能了 LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA，将会使 flash 进入 SUSPEND 状态；如果使能了 LP_ANA_BOD_MODE0_PD_RF_ENA，将会关闭射频模块。
  - 当欠压计数器达到 Rst Comparer (LP_ANA_BOD_MODE0_RESET_WAIT) 设定的阈值后根据 `bod_mode0_rst_sel` (LP_ANA_BOD_MODE0_RESET_SEL) 的配置选择复位方式。复位需通过 `bod_mode0_rst_en` (LP_ANA_BOD_MODE0_RESET_ENA) 使能。
- 模式 1：直接触发系统复位。

详情如下图所示。

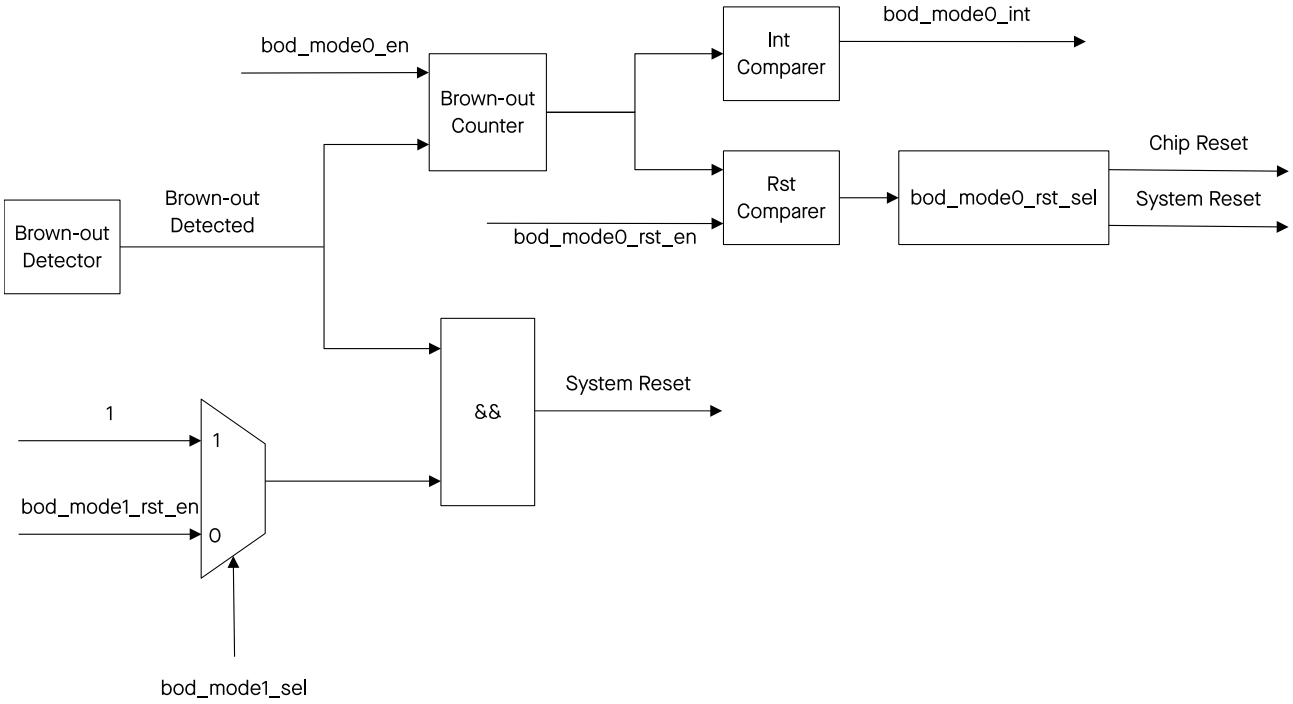


图 21.3-2. 欠压检测器处理方式

图中的选择信号均由寄存器控制，对应关系如下：

- `bod_mode0_en`：LP_ANA_BOD_MODE0_INTR_ENA
- `bod_mode0_rst_en`：LP_ANA_BOD_MODE0_RESET_ENA
- `bod_mode0_rst_sel`：LP_ANA_BOD_MODE0_RESET_SEL 用于选择模式 0 欠压检测器被触发后的复位方式。
  - 0：芯片复位
  - 1：系统复位

  更多有关复位的信息请参考章节 9 复位和时钟。
- `bod_mode1_sel`：LP_ANA_ANA_FIB_ENA 的第一位。
- `bod_mode1_rst_en`：LP_ANA_BOD_MODE1_RESET_ENA

21.3.3 电压毛刺检测器

ESP32-C5 的电压毛刺检测器的结构如下图所示：

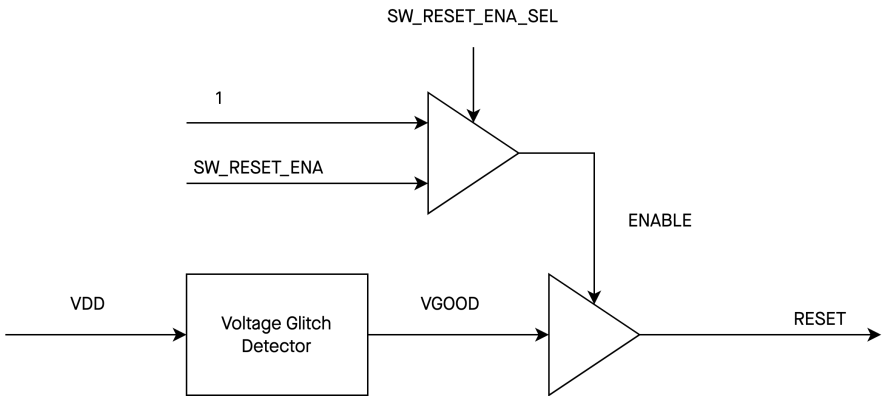


图 21.3-3. 电压毛刺检测器

ESP32-C5 有 4 个电源检测器，每个电源检测器可以检测对应电源上大于 50 ns 的毛刺，包括 VDDPST1、VDDPST2、VDDPST3、VDDA8、VDDA3，其中 VDDPST2 和 VDDPST3 互连，因此可以共用一个电源检测器，所有检测器产生的复位合并为一个 voltage glitch 复位。

电压毛刺检测器的使能控制由 SW_RESET_ENA_SEL 控制，选择为 1 时，由硬件强制开启，选 0 时，由软件寄存器 SW_RESET_ENA 控制。对于不同的电源管脚检测的配置如下表所示：

表 21.3-1. 电源管脚检测的配置

VDD 管脚	SW_RESET_ENA	SW_RESET_ENA_SEL
VDDPST2/3	LP_ANA_PWR_GLITCH_RESET_ENA[0]	LP_ANA_ANA_FIB_ENA[2]
VDDPST1	LP_ANA_PWR_GLITCH_RESET_ENA[1]	LP_ANA_ANA_FIB_ENA[3]
VDDA3	LP_ANA_PWR_GLITCH_RESET_ENA[2]	LP_ANA_ANA_FIB_ENA[4]
VDDA8	LP_ANA_PWR_GLITCH_RESET_ENA[3]	LP_ANA_ANA_FIB_ENA[5]

21.4 中断

ESP32-C5 的电源检测器可可以生成以下中断信号并发送给中断矩阵。

- LP_ANA_INTR（该中断供 HP CPU 使用）
- LP_ANA_LP_INTR（该中断供 LP CPU 使用）

这些中断信号由电源检测器的内部中断源生成。表 21.4-1 列出了电源检测器的内部中断源、中断触发条件、以及生成的中断信号。

表 21.4-1. 电源检测器的内部中断源

内部中断源	触发条件	中断信号
LP_ANA_BOD_MODE0_INT	欠压检测器检测到电压低于阈值时触发	LP_ANA_INTR
LP_ANA_BOD_MODE0_LP_INT	欠压检测器检测到电压低于阈值时触发	LP_ANA_LP_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 21.5 寄存器列表。

21.5 寄存器列表

本小节的所有地址均为相对于电源检测器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

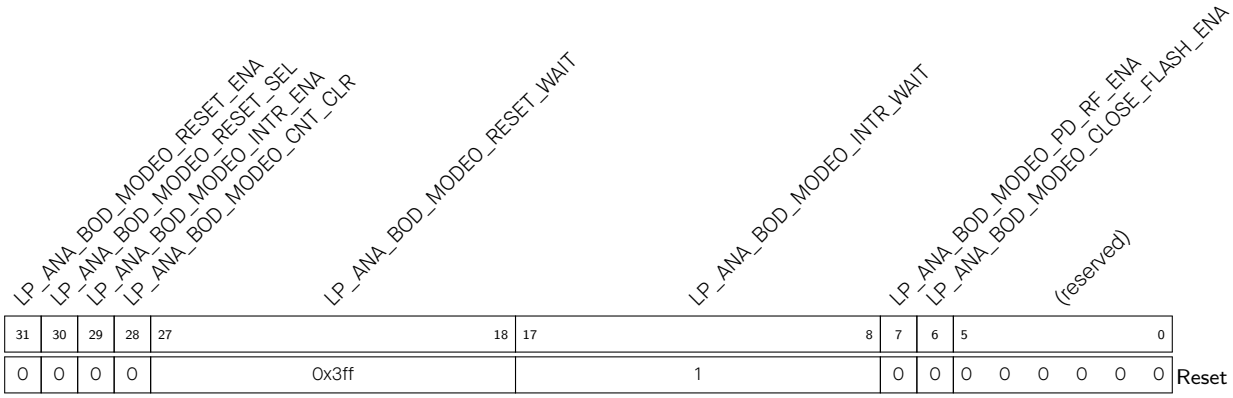
名称	描述	地址	访问
配置寄存器			
LP_ANA_BOD_MODE0_CNTL_REG	配置欠压检测器模式 0	0x0000	R/W
LP_ANA_BOD_MODE1_CNTL_REG	配置欠压检测器模式 1	0x0004	R/W
LP_ANA_POWER_GLITCH_CNTL_REG	电压毛刺控制寄存器	0x0008	R/W
LP_ANA_FIB_ENABLE_REG	电压毛刺检测器使能控制器	0x000C	R/W
LP_ANA_INT_RAW_REG	LP_ANA_BOD_MODE0_INT 原始中断状态	0x0010	R/WTC/SS
LP_ANA_INT_ST_REG	LP_ANA_BOD_MODE0_INT 屏蔽中断状态	0x0014	RO
LP_ANA_INT_ENA_REG	LP_ANA_BOD_MODE0_INT 使能中断状态	0x0018	R/W
LP_ANA_INT_CLR_REG	LP_ANA_BOD_MODE0_INT 中断清除	0x001C	WT
LP_ANA_LP_INT_RAW_REG	LP_ANA_BOD_MODE0_LP_INT 原始中断状态	0x0020	R/WTC/SS
LP_ANA_LP_INT_ST_REG	LP_ANA_BOD_MODE0_LP_INT 屏蔽中断状态	0x0024	RO
LP_ANA_LP_INT_ENA_REG	LP_ANA_BOD_MODE0_LP_INT 使能中断状态	0x0028	R/W
LP_ANA_LP_INT_CLR_REG	LP_ANA_BOD_MODE0_LP_INT 中断清除	0x002C	WT
版本控制寄存器			
LP_ANA_DATE_REG	版本控制寄存器	0x03FC	R/W

21.6 寄存器

本小节的所有地址均为相对于电源检测器 z 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 21.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)



LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA 配置是否使能欠压检测器触发 flash 挂起。

0：禁用

1：使能

(R/W)

LP_ANA_BOD_MODE0_PD_RF_ENA 配置是否使能欠压检测器关闭射频模块。

0：禁用

1：使能

(R/W)

LP_ANA_BOD_MODE0_INTR_WAIT 配置欠压信号有效后，产生中断的时间，单位是 LP_FAST_CLK。(R/W)

LP_ANA_BOD_MODE0_RESET_WAIT 配置欠压信号有效后，产生复位的时间，单位是 LP_FAST_CLK。(R/W)

LP_ANA_BOD_MODE0_CNT_CLR 配置是否清空欠压检测器计数器。

0：撤销

1：清空

(R/W)

LP_ANA_BOD_MODE0_INTR_ENA 配置是否使能欠压检测器模式 0 的中断。此域为 1 时 LP_ANA_BOD_MODE0_INT_RAW 和 LP_ANA_BOD_MODE0_LP_INT_RAW 才有效。

0：禁用

1：使能

(R/W)

LP_ANA_BOD_MODE0_RESET_SEL 配置欠压检测器被触发后的复位方式。

0：芯片复位

1：系统复位

(R/W)

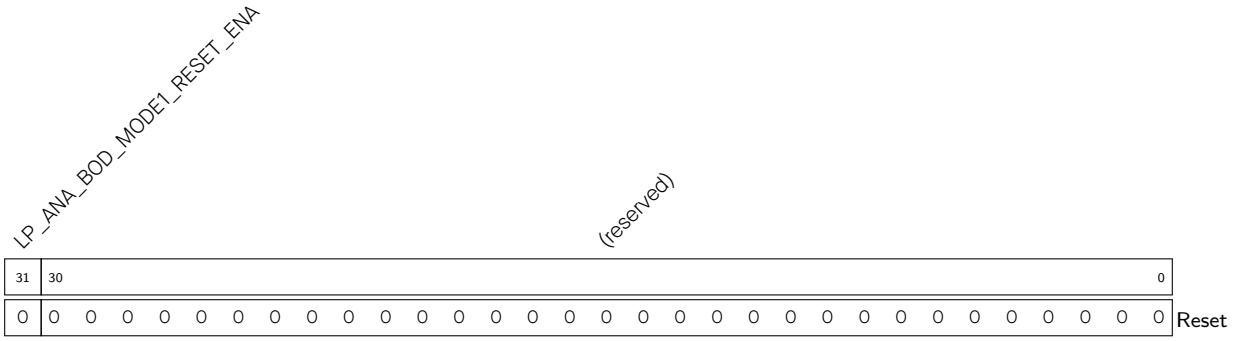
LP_ANA_BOD_MODE0_RESET_ENA 配置是否使能欠压检测器复位。

0：禁用

1：使能

(R/W)

Register 21.2. LP_ANA_BOD_MODE1_CNTL_REG (0x0004)



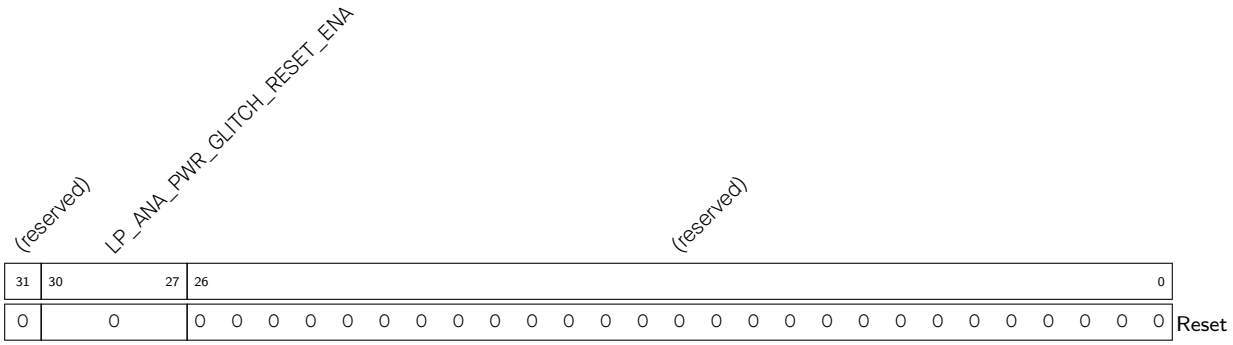
LP_ANA_BOD_MODE1_RESET_ENA 配置是否使能欠压检测器模式 1。

0：禁用

1：使能

(R/W)

Register 21.3. LP_ANA_POWER_GLITCH_CNTL_REG (0x0008)



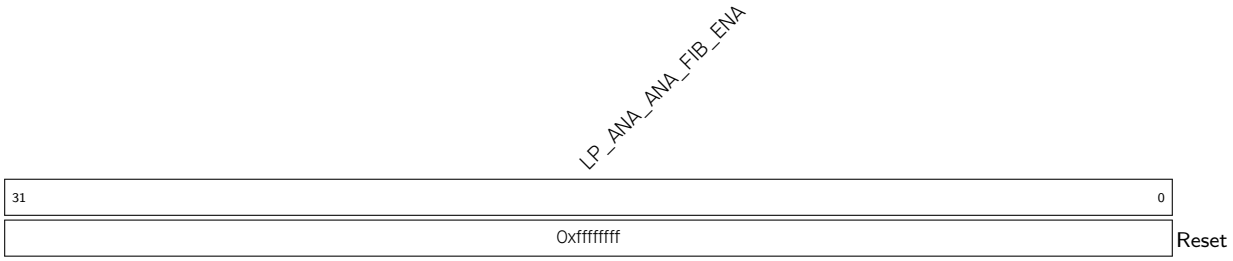
LP_ANA_PWR_GLITCH_RESET_ENA 配置是否使能电压毛刺检测器。其中 bit0，bit1，bit2，bit3 分别对应 VDDPST2/3，VDDPST1，VDDA3，VDDA8。

0：禁用

1：使能

(R/W)

Register 21.4. LP_ANA_FIB_ENABLE_REG (0x000C)



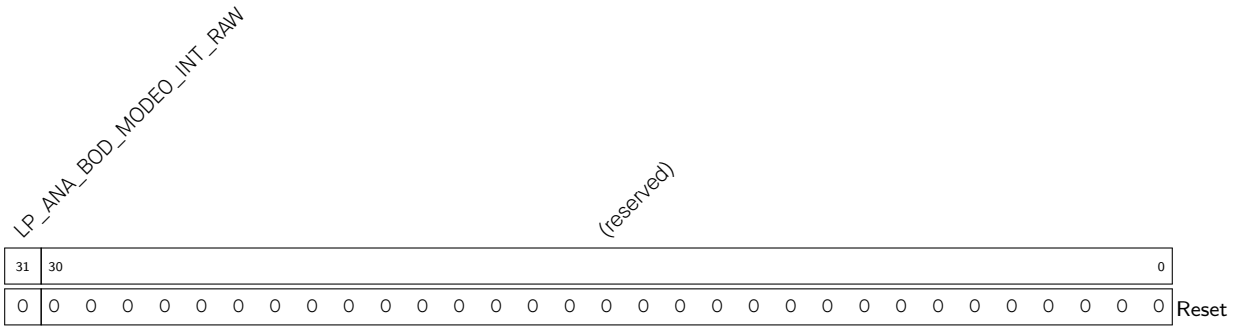
LP_ANA_ANA_FIB_ENA 配置电压毛刺检测器的使能控制。其中 bit2，bit3，bit4，bit5 分别对应 VDDPST2/3，VDDPST1，VDDA3，VDDA8。

0：由 LP_ANA_PWR_GLITCH_RESET_ENA 控制

1：由硬件强制开启

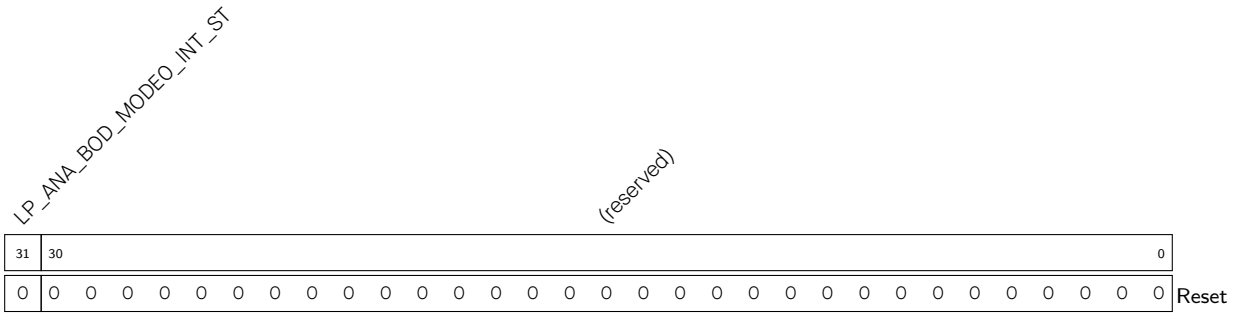
(R/W)

Register 21.5. LP_ANA_INT_RAW_REG (0x0010)



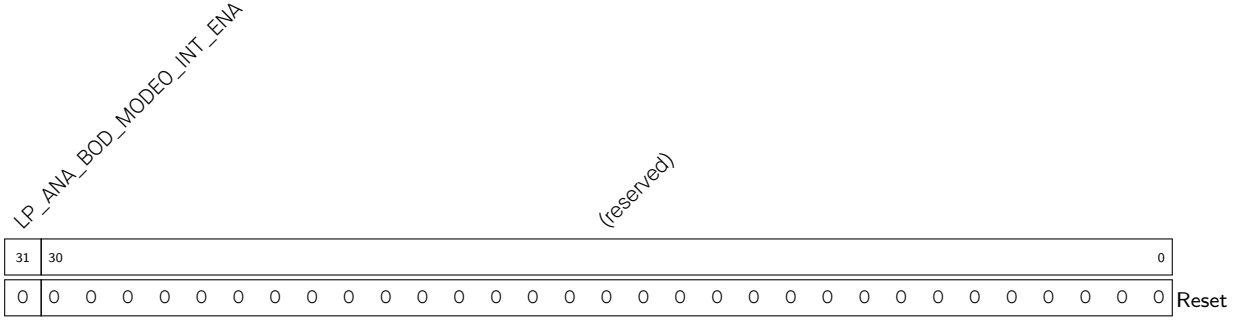
LP_ANA_BOD_MODE0_INT_RAW LP_ANA_BOD_MODE0_INT 的原始中断状态。(R/WTC/SS)

Register 21.6. LP_ANA_INT_ST_REG (0x0014)



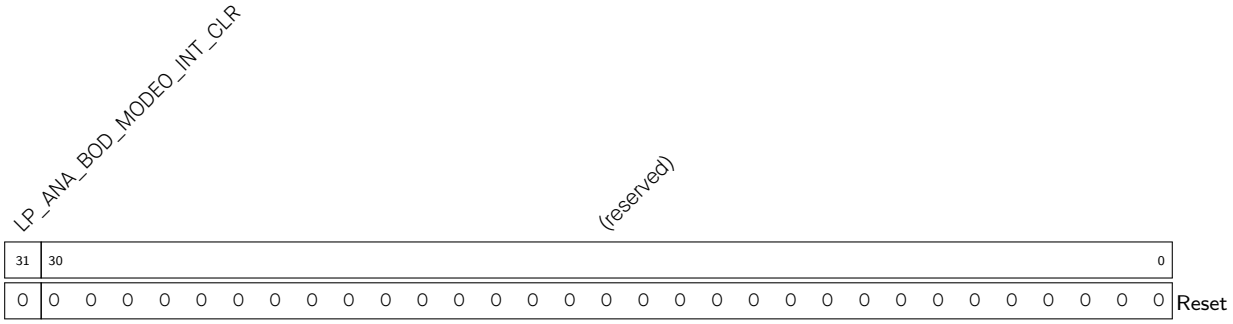
LP_ANA_BOD_MODE0_INT_ST LP_ANA_BOD_MODE0_INT 的屏蔽中断状态。(RO)

Register 21.7. LP_ANA_INT_ENA_REG (0x0018)



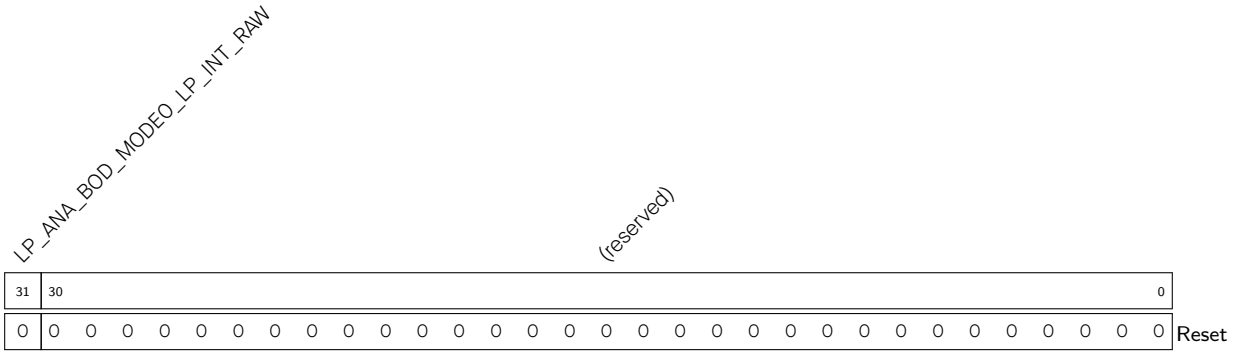
LP_ANA_BOD_MODE0_INT_ENA 写 1 使能 LP_ANA_BOD_MODE0_INT。(R/W)

Register 21.8. LP_ANA_INT_CLR_REG (0x001C)



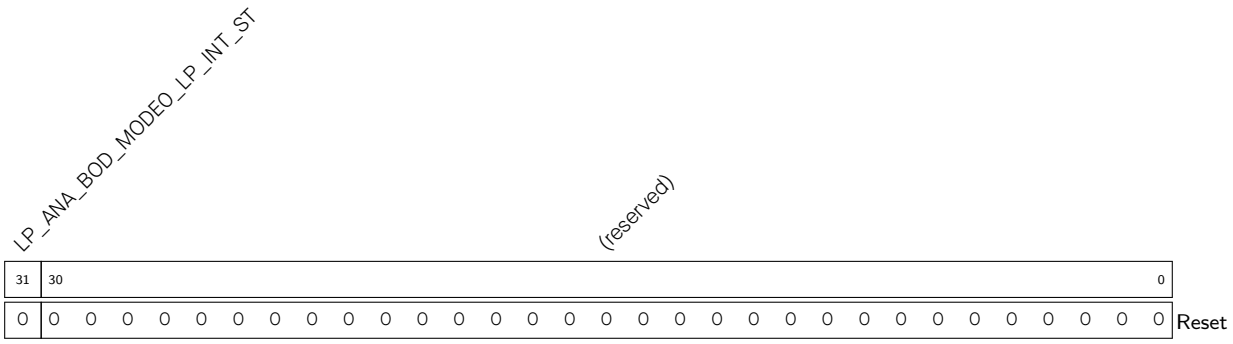
LP_ANA_BOD_MODE0_INT_CLR 写 1 清除 LP_ANA_BOD_MODE0_INT。(WT)

Register 21.9. LP_ANA_LP_INT_RAW_REG (0x0020)



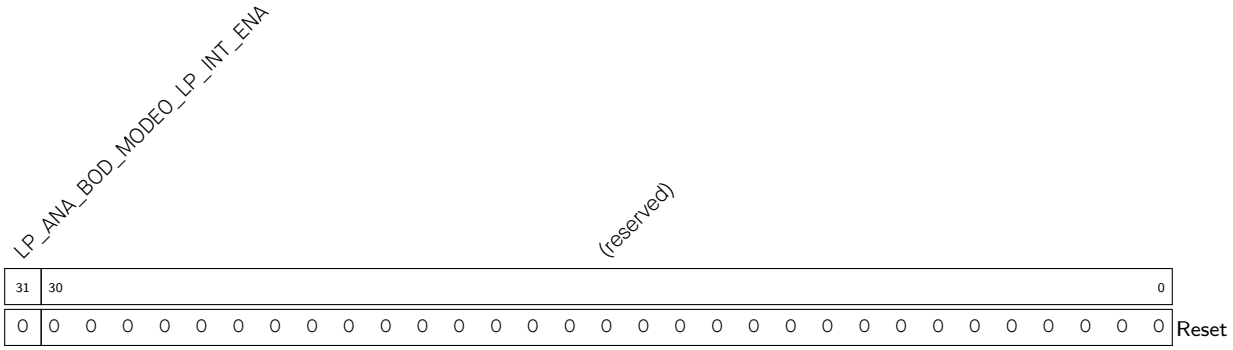
LP_ANA_BOD_MODE0_LP_INT_RAW LP_ANA_BOD_MODE0_LP_INT 的原始中断状态。(R/WTC/SS)

Register 21.10. LP_ANA_LP_INT_ST_REG (0x0024)



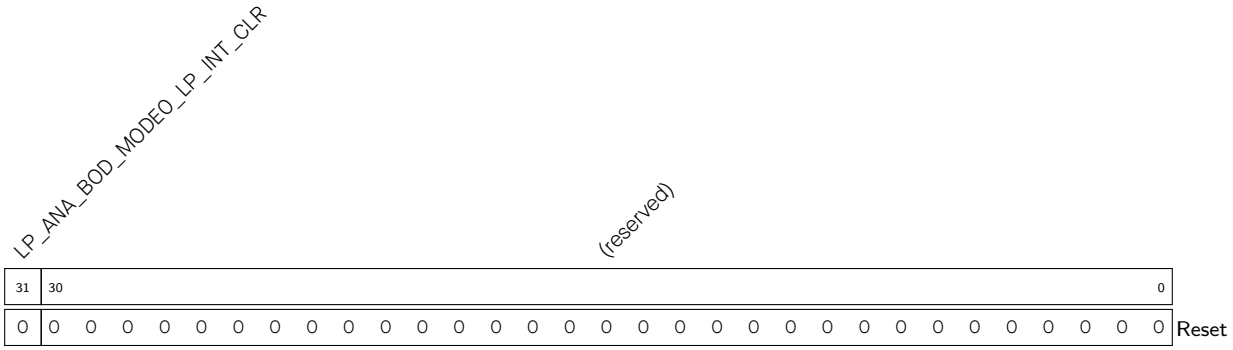
LP_ANA_BOD_MODE0_LP_INT_ST LP_ANA_BOD_MODE0_LP_INT 的屏蔽中断状态。(RO)

Register 21.11. LP_ANA_LP_INT_ENA_REG (0x0028)



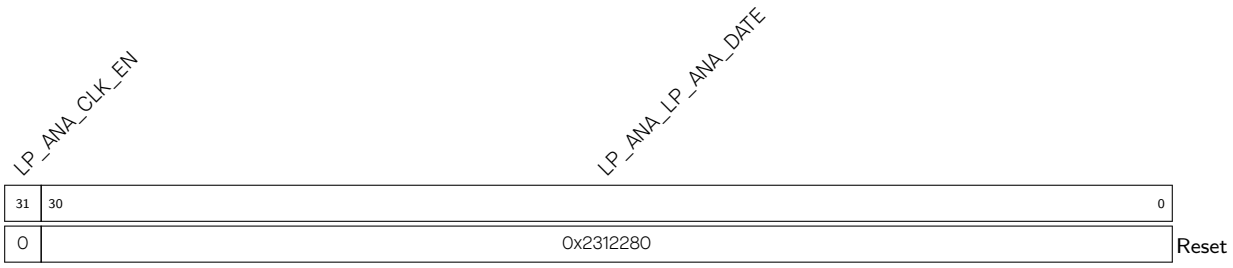
LP_ANA_BOD_MODE0_LP_INT_ENA 写 1 使能 LP_ANA_BOD_MODE0_LP_INT。(R/W)

Register 21.12. LP_ANA_LP_INT_CLR_REG (0x002C)



LP_ANA_BOD_MODE0_LP_INT_CLR 写 1 清除 LP_ANA_BOD_MODE0_LP_INT。(WT)

Register 21.13. LP_ANA_DATE_REG (0x03FC)



LP_ANA_LP_ANA_DATE 版本控制寄存器。(R/W)

LP_ANA_CLK_EN 使能强制打开配置寄存器时钟。

0：自动时钟门控

1：强制打开寄存器时钟

此域的配置不影响寄存器的读写。

(R/W)

第 IV 卷

加密和安全组件

该部分专门介绍芯片的安全功能，包括 AES、ECC 等加密加速器，数字签名、随机数生成、密钥管理、加密/解密算法等安全特性，展示了芯片在加密和安全数据处理方面的性能。

第 22 章

AES 加速器 (AES)

22.1 概述

ESP32-C5 内置 AES（高级加密标准）硬件加速器可使用 AES 算法，完成数据的加解密运算，具有 Typical AES 和 DMA-AES 两种工作模式。整体而言，相比基于纯软件的 AES 运算，AES 硬件加速器能够极大地提高运算速度。

22.2 主要特性

ESP32-C5 支持以下特性：

- Typical AES 工作模式
  - AES-128/AES-256 加解密运算
- DMA-AES 工作模式
  - AES-128/AES-256 加解密运算
  - 块（加密）模式
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - 中断发生

22.3 时钟和复位

AES 加速器的激活需使能 PCR_AES_CONF_REG 寄存器中的 PCR_AES_CLK_EN 位，并同时清零 PCR_AES_RST_EN 位。此外，由于密码学加速器模块之间的资源复用，用户还需要额外清零 PCR_DS_CONF_REG 寄存器中的 PCR_DS_RST_EN 位。

22.4 工作模式简介

ESP32-C5 内置的 AES 加速器支持 Typical AES 和 DMA-AES 两种工作模式。

- Typical AES 工作模式：
  - 支持使用 128 位或 256 位密钥进行加密与解密运算，即 NIST FIPS 197 标准中的 AES-128 和 AES-256 加解密运算。

  这种情况下，明文和密文的读写操作统一通过 CPU 访问完成。
- DMA-AES 工作模式：
  - 支持使用 128 位或 256 位密钥进行加密与解密运算，即 NIST FIPS 197 标准中的 AES-128 和 AES-256 加解密运算；
  - 还支持 NIST SP 800-38A 标准中的 ECB、CBC、OFB、CTR、CFB8、CFB128 块加密模式运算。

  在这种情况下，明文和密文的传输通过硬件上的 DMA 完成，运算完成时会有中断发生。

用户可通过配置 AES_DMA_ENABLE_REG 选择 AES 加速器的工作模式，具体参考表 22.4-1。

表 22.4-1. 工作模式

AES_DMA_ENABLE_REG	工作模式
0	Typical AES
1	DMA-AES

用户可通过配置 AES_MODE_REG 寄存器选择密钥长度和加解密方向，具体可参考表 22.4-2。

表 22.4-2. 密钥长度和加解密方向

AES_MODE_REG[2:0]	密钥长度和加解密方向
0	AES-128 加密
1	保留
2	AES-256 加密
3	保留
4	AES-128 解密
5	保留
6	AES-256 解密
7	保留

有关 Typical AES 和 DMA-AES 两种工作模式的具体介绍，请见下方 22.5 章节和 22.6 章节。

注意：

ESP32-C5 的 RSA 数字签名外设 (RSA_DS) 模块也会调用 AES 加速器。此时，用户无法正常访问 AES 加速器。

22.5 Typical AES 工作模式

在 Typical AES 工作模式下，AES 加速器的状态值可查看寄存器 AES_STATE_REG，具体见表 22.5-1 所示：

表 22.5-1. Typical AES 模式下状态返回值

AES_STATE_REG	描述	状态说明
0	IDLE	加速器空闲或运算完成
1	WORK	加速器忙于运算

22.5.1 密钥、明文、密文

寄存器 AES_KEY_*n*_REG 用于存放密钥，由 8 个 32 位寄存器组成。

- 如果为 AES-128 加解密运算，则 128 位密钥在寄存器 AES_KEY_0_REG ~ AES_KEY_3_REG 中。
- 如果为 AES-256 加解密运算，则 256 位密钥在寄存器 AES_KEY_0_REG ~ AES_KEY_7_REG 中。

寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG 用于存放明文和密文，各由 4 个 32 位寄存器组成。

- 如果为 AES-128 或 AES-256 加密运算，则运算开始之前用明文初始化寄存器 AES_TEXT_IN_*m*_REG。运算完成之后，AES 加速器将把密文更新入寄存器 AES_TEXT_OUT_*m*_REG。
- 如果为 AES-128 或 AES-256 解密运算，则运算开始之前用密文初始化寄存器 AES_TEXT_IN_*m*_REG。运算完成之后，AES 加速器将把明文更新入寄存器 AES_TEXT_OUT_*m*_REG。

22.5.2 字节序

文本字节序

在 Typical AES 工作模式下，AES 加速器可以使用密钥对 128 位的块进行加解密。在操作寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG 中的数据时，用户应遵循表 22.5-2 中定义的文本字节序。

表 22.5-2. Typical AES 文本字节序

明文/密文					
State[1]		c[2]			
		0	1	2	3
r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
	1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
	2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]
	3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

[1] 有关 “State（以及 c 和 r）” 的详细定义，请参考 NIST FIPS 197 中 3.4 The State 章节。
[2] 其中，*x* = IN 或 OUT。

密钥字节序

在 Typical AES 工作模式下，在向寄存器 AES_KEY_*n*_REG 中填入数据时，用户应遵循表 22.5-3 和表 22.5-4 中定义的文本字节序。

表 22.5-3. AES-128 密钥字节序

Bit[1]	w[0]	w[1]	w[2]	w[3][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

[1] Bit 列代表 w[0] ~ w[3] 每个 word 中的各个字节。

[2] w[0] ~ w[3] 符合标准 NIST FIPS 197 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

表 22.5-4. AES-256 密钥字节序

Bit[1]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

[1] Bit 列代表 w[0] ~ w[7] 每个 word 中的各个字节。

[2] w[0] ~ w[7] 符合标准 NIST FIPS 197 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

22.5.3 Typical AES 工作模式的流程

单次运算

1. 对寄存器 AES_DMA_ENABLE_REG 写入 0。
2. 初始化寄存器 AES_MODE_REG、AES_KEY_*n*_REG、AES_TEXT_IN_*m*_REG。
3. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
4. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 0。
5. 从寄存器 AES_TEXT_OUT_*m*_REG 读取结果。

连续运算

在连续运算过程中，每次运算完成之后，只有寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG (*m*: 0-3) 会被 AES 加速器更新，而 AES_DMA_ENABLE_REG、AES_MODE_REG、AES_KEY_*n*_REG 等寄存器中的内容不会变化。所以进行连续运算时可以简化初始化操作。

1. 第一次运算之前对寄存器 AES_DMA_ENABLE_REG 写入 0。
2. 第一次运算之前初始化寄存器 AES_MODE_REG 和 AES_KEY_*n*_REG。
3. 更新寄存器 AES_TEXT_IN_*m*_REG。
4. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
5. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 0。
6. 从寄存器 AES_TEXT_OUT_*m*_REG 读取结果 。返回步骤 3，进行下一轮运算。

22.6 DMA-AES 工作模式

在 DMA-AES 工作模式下，AES 加速器可支持 ECB、CBC、OFB、CTR、CFB8、CFB128 六种块模式运算。用户可通过配置 AES_BLOCK_MODE_REG 寄存器选择具体运算类型，具体可参考表 22.6-1。

表 22.6-1. 块模式选择

AES_BLOCK_MODE_REG[2:0]	块模式
0	ECB (Electronic Code Book)
1	CBC (Cipher Block Chaining)
2	OFB (Output FeedBack)
3	CTR (Counter)
4	CFB8 (8-bit Cipher FeedBack)
5	CFB128 (128-bit Cipher FeedBack)
6	保留
7	保留

AES 加速器的状态值可查看寄存器 AES_STATE_REG，具体见表 22.6-2 所示：

表 22.6-2. DMA-AES 模式下状态返回值

AES_STATE_REG	描述	状态说明
0	IDLE	加速器空闲
1	WORK	加速器忙于运算
2	DONE	加速器运算完成

AES 加速器在 DMA-AES 工作模式下允许中断发生，软件清零。中断功能默认关闭，用户可通过将 AES_INT_ENA_REG 寄存器配置为 1 开启中断。如开启中断功能，AES 加速器在完成运算时，中断发生。

22.6.1 密钥、明文、密文

块运算模式

在块运算模式下，AES 加速器的源数据来自 DMA，结果数据也将被写入 DMA。

- 如果为加密运算，则 DMA 从存储器中读取明文数据流并将其传给 AES。AES 运算出密文后将密文写入 DMA。DMA 再将密文写入存储器。
- 如果为解密运算，则 DMA 从存储器中读取密文数据流并将其传给 AES。AES 运算出明文后将明文写入 DMA。DMA 再将明文写入存储器。

AES 加速器在进行块运算时，结果数据与源数据的大小保持一致。此时，DMA 的数据搬运过程和 AES 的运算过程有所交叠，因此总工作时间有所减少。

值得注意的是，AES 加速器在 DMA-AES 工作模式下要求源数据的大小必须是 128 位的整数倍，否则需要将原始明文封装为 128 位的整数倍，即在原比特串 (bit string) 尾部尽可能少的补 “0”，具体过程见表 22.6-3 所示。

表 22.6-3. TEXT-PADDING

函数： TEXT-PADDING( )
输入： X，比特串。 **输出：** Y = TEXT-PADDING(X)，长度为最接近 128 比特整数倍的值。
步骤： 假设 X 是一个数据流，可以按如下方式分成 n 个部分： $X = X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n$ 此处，X_1□X_2□$\cdots$□X_{n-1} 的长度均为 128 比特，而 X_n 的长度为 t (0<=t<=127)。 如果 $t = 0$，则 TEXT-PADDING(X) = X; 如果 $0 < t <= 127$，定义 X_n^* 块的长度为 128 比特，且 $X_n^* = X_n \| \| 0^{128-t}$，则 TEXT-PADDING($X$) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^*$ = $X \| \| 0^{128-t}$

22.6.2 字节序

在 DMA-AES 工作模式下，源数据和结果数据的传输完全由 DMA 完成，因此不支持字节序的控制调节，但要求它们在存储器中以一定的方式来存放，且要求数据量必须是块的整数倍。

举例说明，假设 DMA 需要搬运源数据：

- 十六进制：0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- 数据大小：相当于 2 个块

假设起始地址为 0x0280，则源数据在存储器中的存放位置如表 22.6-4 所示。结果数据也遵从相同的存放规则，在此不多做介绍。

表 22.6-4. DMA-AES 存储字节序

地址	字节	地址	字节	地址	字节	地址	字节
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

22.6.3 标准增量函数

AES 加速器在进行 CTR 块运算时，还可提供两种标准增量函数供用户选择：INC_{32} 和 INC_{128} 。用户可通过将寄存器 AES_INC_SEL_REG 置为 0 或 1 选择 INC_{32} 或 INC_{128} 标准增量函数。更多有关标准增量函数的内容，请见 NIST SP 800-38A 标准中的 B.1 The Standard Incrementing Function 章节。

22.6.4 块个数

寄存器 AES_BLOCK_NUM_REG 存放明文或密文的块个数 (Block Number)，其值等于 length(**TEXT-PADDING**(P))/128，也等于 length(**TEXT-PADDING**(C))/128。这里的 P 指明文（plaintext），C 指密文（ciphertext）。该寄存器仅在 DMA-AES 工作模式下有意义。

22.6.5 初始向量

存储器 AES_IV_MEM 的空间大小为 16 字节，仅在块运算模式下有效。对于 CBC/OFB/CFB8/CFB128 操作，AES_IV_MEM 用于存放初始向量 (Initialization Vector, IV) 的值。对于 CTR 操作，AES_IV_MEM 存放初始计数器 (Initial Counter Block, ICB) 的值。

IV 和 ICB 都是 128 位的比特串，从左向右被分割成 16 个字节 (Byte0, Byte1, Byte2, ⋯, Byte15)，构成一个字节序列，在 AES_IV_MEM 中存放时需要遵循表 22.6-4 中的字节序规则，即 Byte0 存放在 AES_IV_MEM 中的最低地址中，Byte15 存放在 AES_IV_MEM 中的最高地址中。

更多有关 IV 和 ICB 的信息，请参考 NIST SP 800-38A 标准。

22.6.6 DMA-AES 工作模式的流程

1. 选择一条 DMA 通道与 AES 加速器连接，配置 DMA 链表，而后启动 DMA。详情请见章节 *5 通用 DMA 控制器 (GDMA)*。
2. 配置 AES：
    - 对寄存器 AES_DMA_ENABLE_REG 写入 1。

- 选择是否开启中断。根据需要设置寄存器 AES_INT_ENA_REG 的值。
- 初始化 AES_MODE_REG 和 AES_KEY_*n*_REG 寄存器。
- 配置 AES_BLOCK_MODE_REG 寄存器，选择具体块加密模式。详见表 22.6-1。
- 初始化寄存器 AES_BLOCK_NUM_REG，请参照章节 22.6.4。
- 初始化寄存器 AES_INC_SEL_REG（仅在 CTR 块模式下使用）。
- 初始化存储器 AES_IV_MEM（在 ECB 块模式下不使用）。

3. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
4. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 2。如果开启了中断功能，也可以等待 AES_INT 中断产生。
5. 确认 DMA 完成从 AES 到内存的数据传输。此时，结果数据已经被 DMA 写入 memory，可以直接从中读取。详情请参考章节 5 *通用 DMA 控制器 (GDMA)*。
6. 如果开启了中断，当处理中断程序完成后，请及时对寄存器 AES_INT_CLR_REG 写 1 以清除中断。
7. 对寄存器 AES_DMA_EXIT_REG 写入 1 释放 AES 加速器。之后如果再读取寄存器 AES_STATE_REG 将读到 0。该步操作可以提前完成，但必须在步骤 4 之后。

22.7 抗攻击伪轮函数

为了增强抗攻击性能，ESP32-C5 的 AES 引入了伪轮函数。

启用伪轮函数之后，AES 会在原始操作轮次的前后随机插入伪轮次。在伪轮次中，AES 使用硬件自动生成的伪密钥执行一轮伪操作。伪轮次中的操作不会影响原始 AES 操作的结果。

用户可以将 AES_PSEUDO_EN 配置为 1 来启用伪轮函数。伪轮次的配置如下。

伪轮次总数

以下寄存器字段控制在 AES 操作中随机插入的伪轮次总数：

- pseudo_base：AES_PSEUDO_BASE 的值，配置伪轮次的基本次数。
- pseudo_inc：AES_PSEUDO_INC 的值，配置伪轮次的随机增量次数。

伪轮次总数在以下范围内随机分布：

$$[pseudo\_base, pseudo\_base + (2^{pseudo\_inc} - 1)]$$

随机数更新频率

用户可以通过配置 AES_PSEUDO_RNG_CNT 来确定伪轮函数中随机密钥更新的频率。配置的值越大，更新频率越高。一般来说，建议设置为最大值 7。

22.8 存储器列表

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	大小（比特）	起始地址	结束地址	访问权限
AES_IV_MEM	存储器 IV	16 字节	0x0050	0x005F	(R/W)

22.9 寄存器列表

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

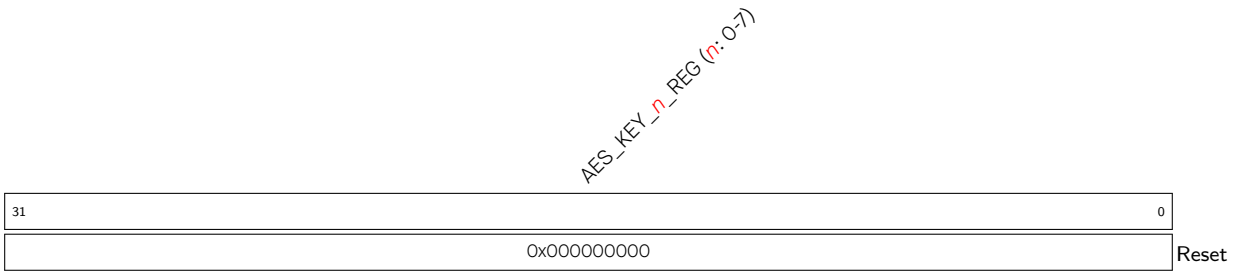
名称	描述	地址	访问
密钥寄存器			
AES_KEY_0_REG	AES 密钥资料寄存器 0	0x0000	R/W
AES_KEY_1_REG	AES 密钥资料寄存器 1	0x0004	R/W
AES_KEY_2_REG	AES 密钥资料寄存器 2	0x0008	R/W
AES_KEY_3_REG	AES 密钥资料寄存器 3	0x000C	R/W
AES_KEY_4_REG	AES 密钥资料寄存器 4	0x0010	R/W
AES_KEY_5_REG	AES 密钥资料寄存器 5	0x0014	R/W
AES_KEY_6_REG	AES 密钥资料寄存器 6	0x0018	R/W
AES_KEY_7_REG	AES 密钥资料寄存器 7	0x001C	R/W
TEXT_IN 寄存器			
AES_TEXT_IN_0_REG	源数据资料寄存器 0	0x0020	R/W
AES_TEXT_IN_1_REG	源数据资料寄存器 1	0x0024	R/W
AES_TEXT_IN_2_REG	源数据资料寄存器 2	0x0028	R/W
AES_TEXT_IN_3_REG	源数据资料寄存器 3	0x002C	R/W
TEXT_OUT 寄存器			
AES_TEXT_OUT_0_REG	结果数据资料寄存器 0	0x0030	RO
AES_TEXT_OUT_1_REG	结果数据资料寄存器 1	0x0034	RO
AES_TEXT_OUT_2_REG	结果数据资料寄存器 2	0x0038	RO
AES_TEXT_OUT_3_REG	结果数据资料寄存器 3	0x003C	RO
控制/配置寄存器			
AES_MODE_REG	密钥长度和加解密方向配置寄存器	0x0040	R/W
AES_TRIGGER_REG	开始运算控制寄存器	0x0048	WT
AES_DMA_ENABLE_REG	工作模式配置寄存器	0x0090	R/W
AES_BLOCK_MODE_REG	块运算模式配置寄存器	0x0094	R/W
AES_BLOCK_NUM_REG	块数量配置寄存器	0x0098	R/W
AES_INC_SEL_REG	标准增量函数配置寄存器	0x009C	R/W
AES_DMA_EXIT_REG	退出运算控制寄存器	0x00B8	WO
AES_PSEUDO_REG	伪轮函数配置寄存器	0x00D0	R/W
状态寄存器			
AES_STATE_REG	运算状态寄存器	0x004C	RO
中断寄存器			
AES_INT_CLR_REG	DMA-AES 中断清除寄存器	0x00AC	WT
AES_INT_ENA_REG	DMA-AES 中断使能寄存器	0x00B0	R/W
版本控制寄存器			
AES_DATE_REG	版本控制寄存器	0x00B4	R/W

22.10 寄存器

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

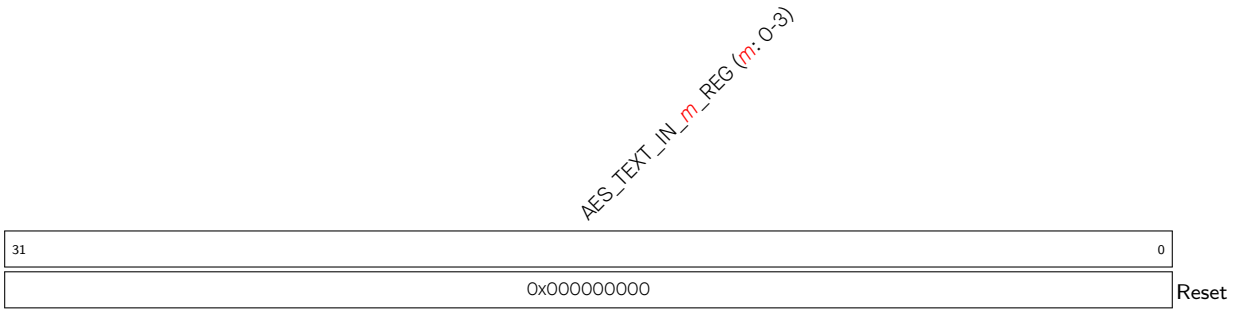
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 22.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**



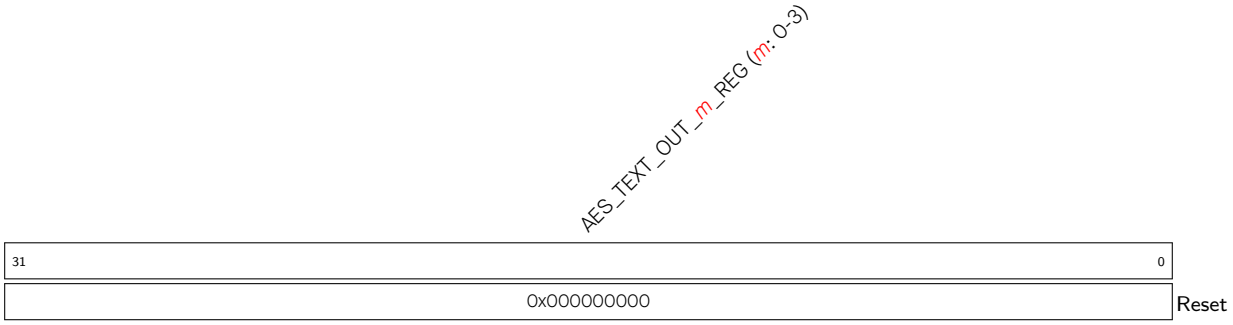
AES_KEY_*n*_REG (*n*: 0-7) 表示 AES 密钥资料。(R/W)

Register 22.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**



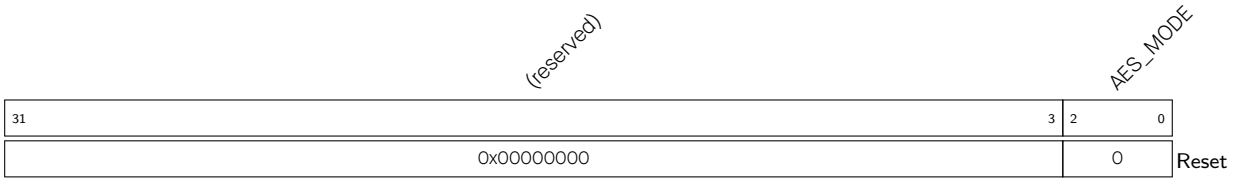
AES_TEXT_IN_*m*_REG (*m*: 0-3) 表示 Typical AES 工作模式下的输入文本。(R/W)

Register 22.3. AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4m*)**



AES_TEXT_OUT_*m*_REG (*m*: 0-3) 表示 Typical AES 工作模式下的输出文本。(RO)

Register 22.4. AES_MODE_REG (0x0040)



AES_MODE 配置 AES 加速器的密钥长度和加解密方向。

0：AES-128 加密

1：保留

2：AES-256 加密

3：保留

4：AES-128 解密

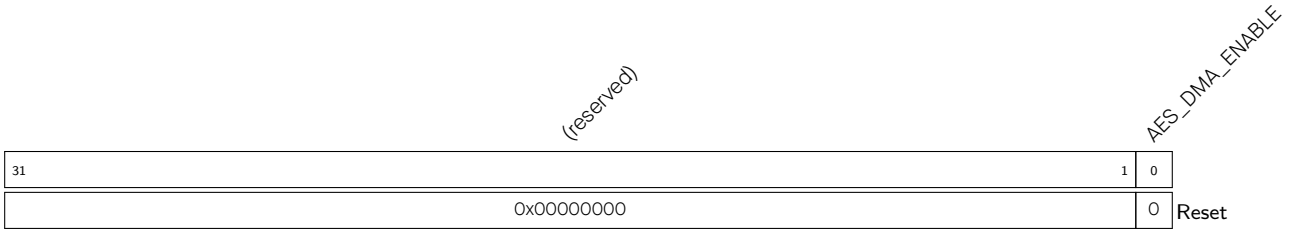
5：保留

6：AES-256 解密

7：保留

(R/W)

Register 22.5. AES_DMA_ENABLE_REG (0x0090)



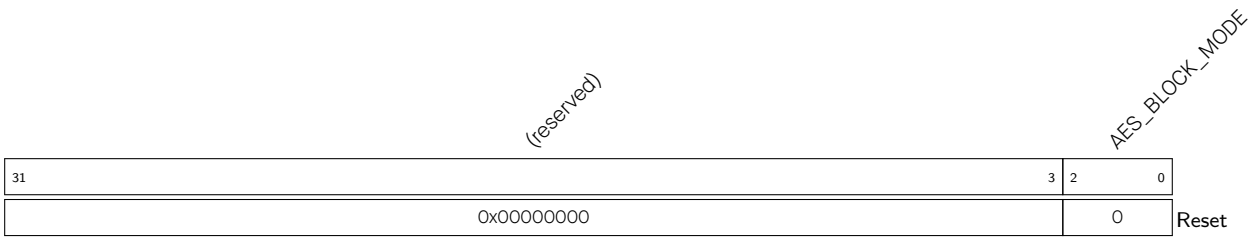
AES_DMA_ENABLE 配置 AES 加速器的工作模式。

0：Typical AES

1：DMA-AES

(R/W)

Register 22.6. AES_BLOCK_MODE_REG (0x0094)



AES_BLOCK_MODE 配置 AES 加速器在 DMA-AES 工作模式下的块模式。

0：ECB (Electronic Code Block)

1：CBC (Cipher Block Chaining)

2：OFB (Output FeedBack)

3：CTR (Counter)

4：CFB8 (8-bit Cipher FeedBack)

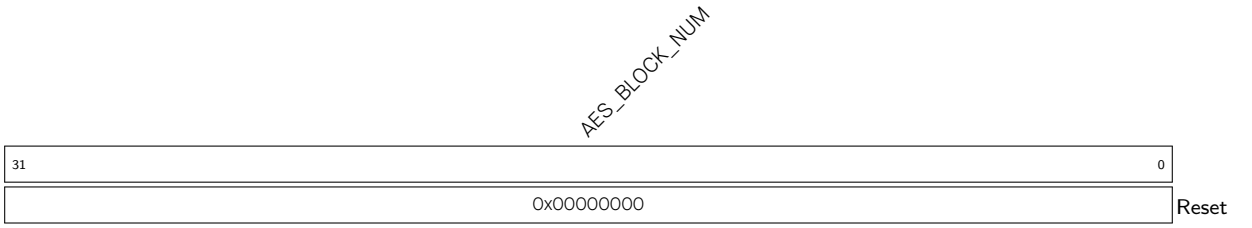
5：CFB128 (128-bit Cipher FeedBack)

6：保留

7：保留

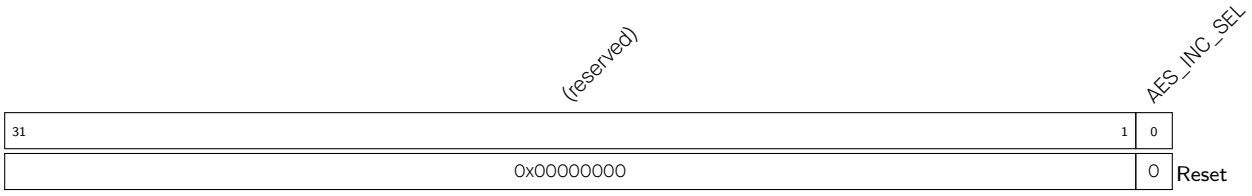
(R/W)

Register 22.7. AES_BLOCK_NUM_REG (0x0098)



AES_BLOCK_NUM 代表 DMA-AES 工作模式下待加解密的文本块数。详情请见章节 22.6.4。(R/W)

Register 22.8. AES_INC_SEL_REG (0x009C)



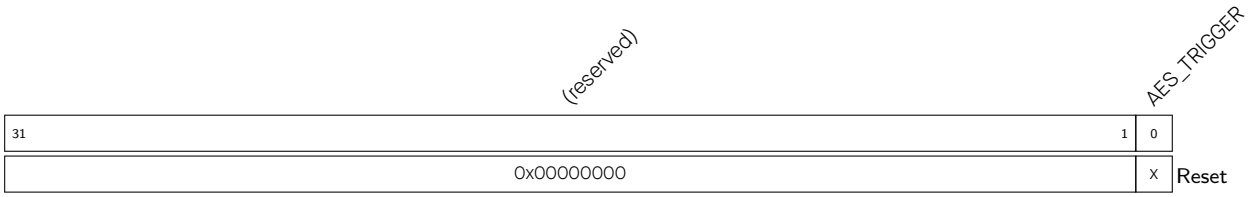
AES_INC_SEL 配置 CTR 块模式使用的标准增量函数。

0：INC_{32} 标准增量函数

1：INC_{128} 标准增量函数

(R/W)

Register 22.9. AES_TRIGGER_REG (0x0048)



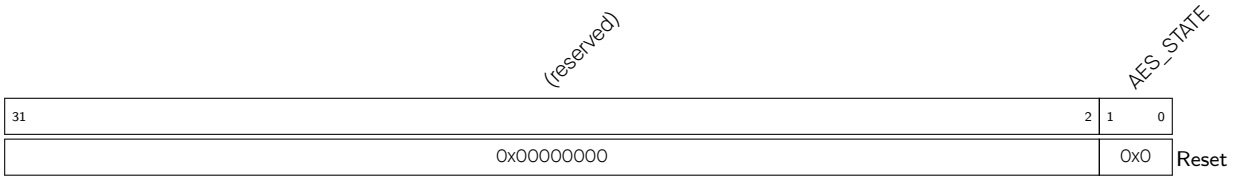
AES_TRIGGER 配置是否启动 AES 运算。

0：无效果

1：启动

(WT)

Register 22.10. AES_STATE_REG (0x004C)



AES_STATE 代表 AES 加速器的状态。

在 Typical AES 工作模式下，

0：IDLE

1：WORK

2：无效果

3：无效果

在 DMA-AES 工作模式下，

0：IDLE

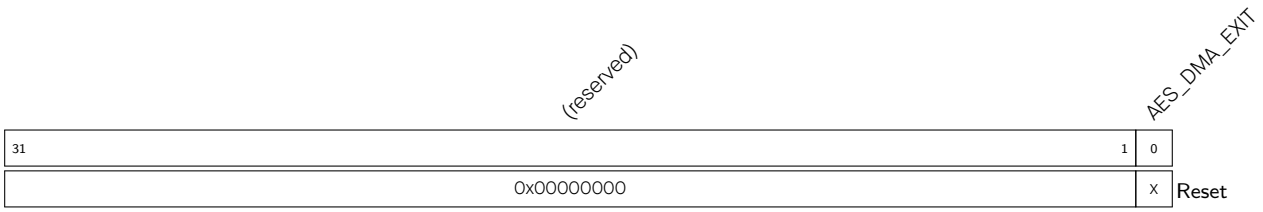
1：WORK

2：DONE

3：无效果

(RO)

Register 22.11. AES_DMA_EXIT_REG (0x00B8)



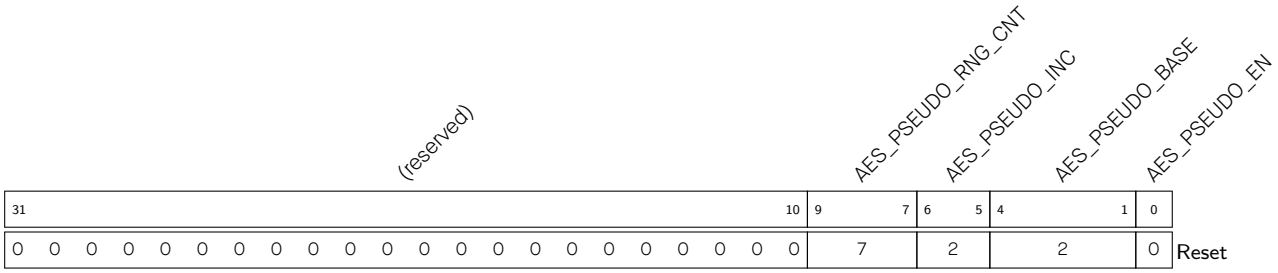
AES_DMA_EXIT 配置是否退出 AES 运算。

0：无效果

1：退出

仅在 DMA-AES 模式下有效。(WO)

Register 22.12. AES_PSEUDO_REG (0x00D0)



AES_PSEUDO_EN 配置是否启用 AES 伪轮函数。

0：禁用

1：启用

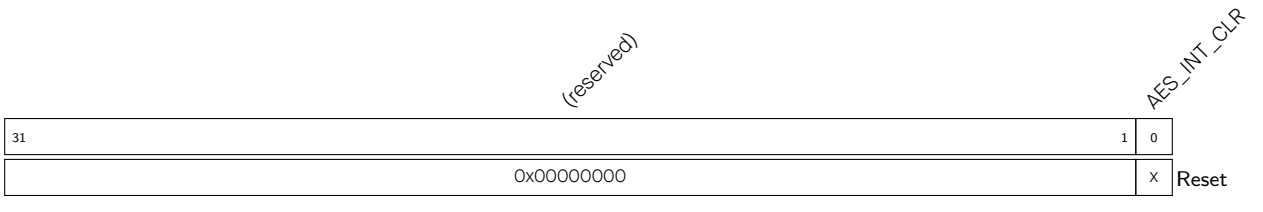
(R/W)

AES_PSEUDO_BASE 配置伪轮次的基本次数。(R/W)

AES_PSEUDO_INC 配置伪轮次的随机增量次数。(R/W)

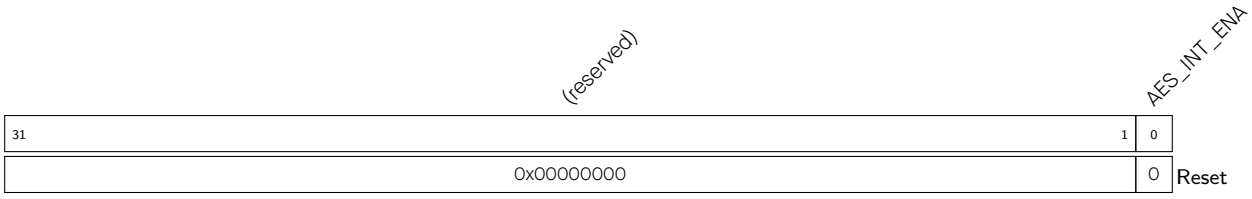
AES_PSEUDO_RNG_CNT 配置伪轮函数中随机密钥更新的频率。(R/W)

Register 22.13. AES_INT_CLR_REG (0x00AC)



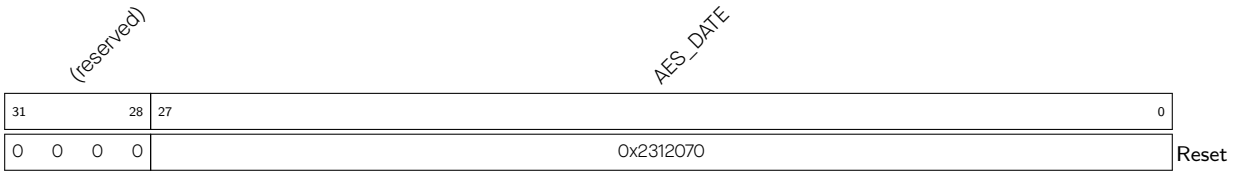
AES_INT_CLR 写 1 清除 AES 中断。(WT)

Register 22.14. AES_INT_ENA_REG (0x00B0)



AES_INT_ENA 写 1 使能 AES 中断。(R/W)

Register 22.15. AES_DATE_REG (0x00B4)



AES_DATE 版本控制寄存器。(R/W)

第 23 章

ECC 加速器 (ECC)

23.1 概述

椭圆曲线密码学 (Elliptic Curve Cryptography, ECC) 是一种基于椭圆曲线数学的公开密钥加密演算法，其优势在于相对于 RSA 算法，使用较小长度的密钥就能够提供相当等级的加密安全性。

ESP32-C5 ECC 硬件加速器支持基于可选曲线的多种基础运算，用以实现对 ECC 基本运算及其衍生算法（如 ECDSA 等算法）的加速。

23.2 主要特性

ESP32-C5 ECC 硬件加速器具备以下特性：

- 支持三种可选 ECC 曲线，即 FIPS 186-5 中定义的 P-192、P-256 和 P-384
- 提供 11 种可选工作模式
- 提升抗攻击性能
- 提供计算完成的中断和中断控制

23.3 ECC 背景知识

本节介绍了与 ECC 硬件加速器相关的背景知识以及章节中会应用到的专业名词。

23.3.1 椭圆曲线与曲线上的点

ECC 是一种基于大素数的有限域椭圆曲线的算法，这一椭圆曲线的数学表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

其中，

- p 是素数，
- a 和 b 为两个小于 p 的非负整数，
- (x, y) 为满足该椭圆曲线方程的点。

23.3.2 仿射坐标系与 Jacobian 坐标系

一条椭圆曲线：

- 在仿射坐标系下的表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

- 在 Jacobian 坐标系下的表达式为：

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

一个曲线上的点在仿射坐标系下的表示 (x, y) 与在 Jacobian 坐标系下的表示 (X, Y, Z) 有如下转换关系：

- 从 Jacobian 坐标系到仿射坐标系的转换：

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- 从仿射坐标系到 Jacobian 坐标系的转换：

$$X = x$$

$$Y = y$$

$$Z = 1$$

23.3.3 内存块

ECC 硬件加速器的内存块用于存储 ECC 运算中所用到的输入数据和输出数据。

表 23.3-1. ECC 硬件加速器内存块

内存块[1]	大小（字节）	起始地址[2]	结束地址[2]	访问权限
ECC_MULT_Mem_k	48	0x100	0x12F	R/W
ECC_MULT_Mem_Px	48	0x130	0x15F	R/W
ECC_MULT_Mem_Py	48	0x160	0x18F	R/W
ECC_MULT_Mem_Qx	48	0x190	0x1BF	R/W
ECC_MULT_Mem_Qy	48	0x1C0	0x1EF	R/W
ECC_MULT_Mem_Qz	48	0x1F0	0x21F	R/W

[1] 内存块在各个工作模式中储存的数据内容不同，详见章节 23.4.2。

[2] 采用相对于 ECC 加速器基地址的偏移量。详见章节 6 系统和存储器 中的表 6.3-2。

23.3.4 数据与数据块

在 ECC 硬件加速器模块中会用到的数据位宽可能为 192、256 或 384 位，这取决于进行运算的曲线。

假设一个数据为 $D[255:0]$，则其可以被划分为八个 32 位的数据块 $D[n][31:0](n = 0, 1, \cdots, 7)$，序号数低的数据块对应二进制低位，即：

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

23.3.5 数据存储

数据存储即将数据存储进一个内存块的操作，该数据为 ECC 算法的输入数据。具体而言，将数据写入一个 ECC 内存块相当于将该数据 $D[n][31:0](n = 0, 1, \cdots, 11)$ 依次写入“该内存块起始地址 + $4 \times n$”：

- 写入 $D[0]$ 至“起始地址”
- 写入 $D[1]$ 至“起始地址 + 4”
- $\cdots$
- 写入 $D[11]$ 至“起始地址 + 44”

说明：
进行数据存储操作时，只需要写入数据长度的数据块即可，不需要在高位补 0。例如，在写入 192 位长度的数据时，只需要存储 192 位的数据，不需要在数据的高位补 0。

23.3.6 数据读取

数据读取即将一个数据从一个内存块读出的操作，该数据为 ECC 算法的输出数据。具体来说，从一个 ECC 内存块读数据相当于从“该内存块起始地址 + $4 \times n$”依次读出 $D[n][31:0](n = 0, 1, \cdots, 11)$：

- 从“起始地址”读出 $D[0]$
- 从“起始地址 + 4”读出 $D[1]$
- $\cdots$
- 从“起始地址 + 44”读出 $D[11]$

说明：
进行数据读取操作时，只需要读取数据长度的数据块即可。例如，在读取 192 位长度的数据时，只需要读取低 192 位（即 6 个数据块）的数据。

23.3.7 标准运算与 Jacobian 运算

ESP32-C5 ECC 硬件加速器中，所有标准运算（包括标准点验证、标准点加和标准点乘）的输入数据以及输出数据中的点均在仿射坐标系中；相对应地，所有 Jacobian 运算（包括 Jacobian 点验证、Jacobian 点加和 Jacobian 点乘）的输入数据以及输出数据中的点均在 Jacobian 坐标系中。

23.4 功能描述

23.4.1 曲线模式

ESP32-C5 ECC 硬件加速器共支持 3 种曲线模式，每种曲线模式唯一对应一条椭圆曲线。用户通过配置 ECC_MULT_CURVE_MODE 可选定密钥长度模式，其与椭圆曲线的对应关系如表 23.4-1 所示。

表 23.4-1. ECC 加速器曲线模式控制

ECC_MULT_CURVE_MODE	对应椭圆曲线*
0	FIPS P-192
1	FIPS P-256
2	FIPS P-384

* FIPS P-192/P-256/P-384 的曲线定义，请参考 FIPS 186-5。

23.4.2 工作模式

ESP32-C5 ECC 硬件加速器共有 11 种工作模式，每种工作模式基于选定曲线进行不同操作。用户通过配置 ECC_MULT_WORK_MODE 可选定工作模式，其值与工作模式的对应关系如表 23.4-2 所示。

表 23.4-2. ECC 硬件加速器工作模式控制

ECC_MULT_WORK_MODE	对应模式
0	标准点乘
1	**保留项，不可用**
2	标准点验证
3	标准点验证 + 标准点乘
4	Jacobian 点乘
5	点加
6	Jacobian 点验证
7	标准点验证 + Jacobian 点乘
8	模加
9	模减
10	模乘
11	模除

说明：
请注意，Jacobian 点乘模式的运算速度比标准点乘模式约快 10%。

每种工作模式的具体计算和输入/输出数据，请参照下述子章节。

23.4.2.1 标准点乘模式

标准点乘模式计算公式如下：

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中，

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px、ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。

- 输出数据：Q_x 和 Q_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。

23.4.2.2 标准点验证模式

标准点验证模式用于计算点 (P_x, P_y) 是否在选定的椭圆曲线上。

- 输入数据：P_x 和 P_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
- 输出数据：点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。

23.4.2.3 标准点验证 + 标准点乘模式

标准点验证 + 标准点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上，如果其在选定的椭圆曲线上，则继续计算如下公式：

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中，

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px、ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。
- 输出数据：
  - 点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。
  - Q_x 和 Q_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - J_x、J_y 和 J_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

23.4.2.4 Jacobian 点乘模式

Jacobian 点乘模式计算公式如下：

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px、ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。
- 输出数据：Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

23.4.2.5 点加模式

点加模式计算公式如下：

$$R = (R_x, R_y) = (J_x, J_y, J_z) = (P_x, P_y, 1) + (Q_x, Q_y, Q_z)$$

其中，

- (R_x, R_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：
  - P_x 和 P_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。
- 输出数据：
  - R_x 和 R_y 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - J_x、J_y 和 J_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

23.4.2.6 Jacobian 点验证模式

Jacobian 点验证模式用于计算点 (Q_x, Q_y, Q_z) 是否在选定的椭圆曲线上。

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。
- 输出数据：点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。

23.4.2.7 标准点验证 + Jacobian 点乘模式

标准点验证 + Jacobian 点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上，如果其在选定的椭圆曲线上，则继续计算如下公式：

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_MULT_Mem_Px, ECC_MULT_Mem_Py 和 ECC_MULT_Mem_k。
- 输出数据：
  - 点验证的结果存储在 ECC_MULT_VERIFICATION_RESULT 中。
  - Q_x、Q_y 和 Q_z 对应的内存块为 ECC_MULT_Mem_Qx、ECC_MULT_Mem_Qy 和 ECC_MULT_Mem_Qz。

23.4.2.8 模加模式

模加模式计算公式如下：

$$R = A + B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
    * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Px。

23.4.2.9 模减模式

模减模式计算公式如下：

$$R = A - B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
    * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Px。

23.4.2.10 模乘模式

模乘模式计算公式如下：

$$R = A \cdot B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
    * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Py。

23.4.2.11 模除模式

模除模式计算公式如下：

$$R = A \cdot B^{-1} \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_MULT_Mem_Px 和 ECC_MULT_Mem_Py。
  - N 的值可通过以下寄存器字段配置：

  * 配置 ECC_MULT_CURVE_MODE 来选择曲线。
  * 配置 ECC_MULT_MOD_BASE 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 ECC_MULT_Mem_Py。

23.4.3 提升抗攻击性能

在执行不同类型的点乘计算（即模式 0、3、4 和 7）时，ESP32-C5 的 ECC 加速器提供了一个选项，可保证所有点乘运算都维持相同的最长运行时间，从而进一步提升其抗攻击性能。

要启用强抗攻击性能，请将 ECC_MULT_SECURITY_MODE 置 1。启用此设置后，ECC 加速器在每次进行点乘计算时：

- 对于每次操作，执行延迟保持恒定。
- 功耗波动最小化。

23.4.4 时钟

ECC 模块有两种时钟：

- clk_ecc：ECC 模块运行所使用的功能时钟
- clk_apb_ecc：用于配置 ECC 模块寄存器的总线时钟

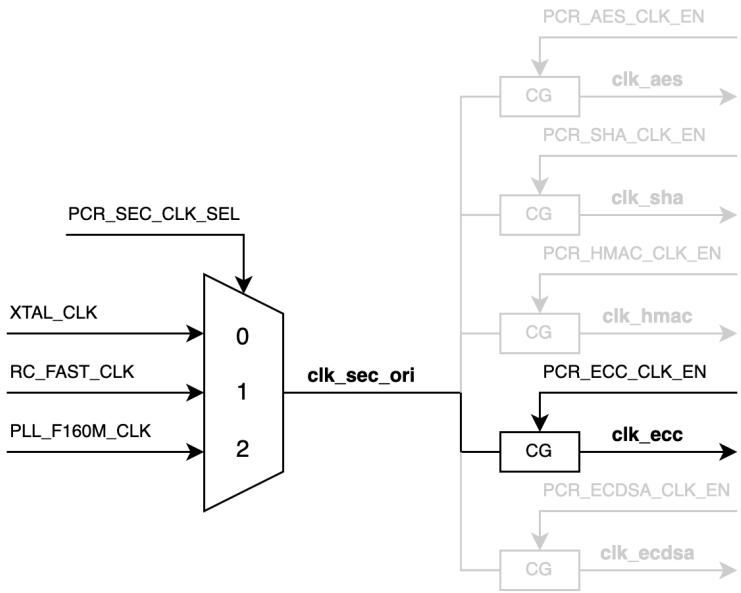


图 23.4-1. clk_ecc 时钟图

如图 23.4-1 *clk_ecc 时钟图* 所示，安全功能源时钟 clk_sec_ori 有以下时钟源：

- XTAL_CLK
- RC_FAST_CLK
- PLL_F160M_CLK

ECC 模块功能时钟 clk_ecc 由时钟门控寄存器 PCR_ECC_CLK_EN 控制，当该寄存器值为 1 时，时钟门控开启，使能 clk_ecc。

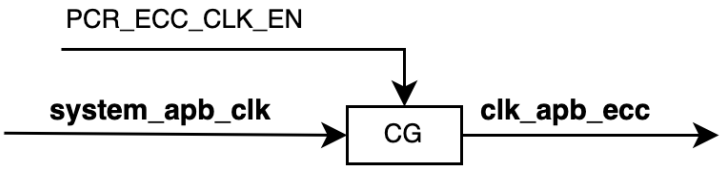


图 23.4-2. clk_apb_ecc 时钟图

如图 23.4-2 *clk_apb_ecc 时钟图* 所示，ECC 模块配置时钟 clk_apb_ecc 的时钟源为 SYSTEM_APB_CLK。

ECC 模块配置时钟 clk_apb_ecc 由时钟门控寄存器 PCR_ECC_CLK_EN 控制，当该寄存器值为 1 时，时钟门控开启，使能 clk_apb_ecc。

有关时钟的更多信息，请查看章节 9 复位和时钟。

23.4.5 复位

ECC 模块支持两种复位方式：

- 自身完全复位：向 PCR_ECC_RST_EN 先写 1 后写 0 使能，可复位整个 ECC 模块。
- 密码学保护复位：向 PCR_ECDSA_RST_EN 先写 1 后写 0 使能，可复位整个 ECC 模块。

说明：
由于密码学加速器之间的资源复用，ECDSA_DS 模块在运算时会复用 ECC 模块中的资源。为了保护 ECDSA_DS 模块的运算数据，在 ECDSA_DS 模块复位时，也会复位 ECC 模块。

23.5 中断

ESP32-C5 ECC 硬件加速器可产生一个中断信号 ECC_INTR，并将其发送给 中断矩阵。

中断信号 ECC_INTR 由 ECC 硬件加速器的内部中断源 ECC_MULT_CALC_DONE_INT 生成。ECC 硬件加速器运算完成即触发该中断源。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。中断源 ECC_MULT_CALC_DONE_INT 由以下寄存器位控制（更多信息请参考 23.7 寄存器列表）：

- ECC_MULT_CALC_DONE_INT_RAW：ECC 硬件加速器运算完成时置 1。
- ECC_MULT_CALC_DONE_INT_ST：表示 ECC 硬件加速器运算完成中断的状态，用 ECC_MULT_CALC_DONE_INT_ENA 使能或屏蔽 ECC_MULT_CALC_DONE_INT_RAW 来生成该位。
- ECC_MULT_CALC_DONE_INT_ENA：用于使能或屏蔽 ECC 硬件加速器运算完成中断状态位。
- ECC_MULT_CALC_DONE_INT_CLR：置 1 此位清除 ECC 硬件加速器运算完成中断，对应的 ECC_MULT_CALC_DONE_INT_RAW 和 ECC_MULT_CALC_DONE_INT_ST 位会清零。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

23.6 配置流程

软件配置 ECC 硬件加速器的流程如下：

1. 配置 ECC 硬件加速器的时钟与复位。配置细节请参考小节 23.4.4 和 23.4.5 。
2. 根据 23.4 小节的描述，按照需求配置 ECC 加速器密钥长度模式和工作模式。
3. 根据 23.5 小节的描述，使能 ECC_MULT_CALC_DONE_INT 中断。
4. 置位 ECC_MULT_START 以启动 ECC 运算。
5. 等待 ECC_MULT_CALC_DONE_INT 中断的产生，即 ECC 运算结束。
6. 根据 23.4 小节的描述，查看运算结果。

23.7 寄存器列表

本小节的所有地址均为相对于 ECC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

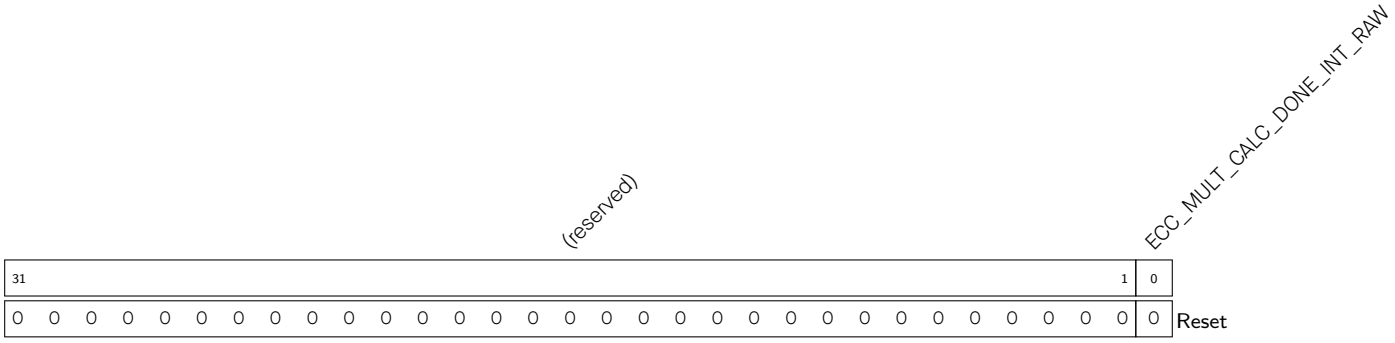
名称	描述	地址	访问权限
中断寄存器			
ECC_MULT_INT_RAW_REG	原始中断状态寄存器	0x000C	R/SS/WTC
ECC_MULT_INT_ST_REG	中断屏蔽状态寄存器	0x0010	RO
ECC_MULT_INT_ENA_REG	中断使能寄存器	0x0014	R/W
ECC_MULT_INT_CLR_REG	中断清除寄存器	0x0018	WT
配置寄存器			
ECC_MULT_CONF_REG	ECC 加速器配置寄存器	0x001C	varies
版本寄存器			
ECC_MULT_DATE_REG	版本控制寄存器	0x00FC	R/W

23.8 寄存器

本小节的所有地址均为相对于 ECC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

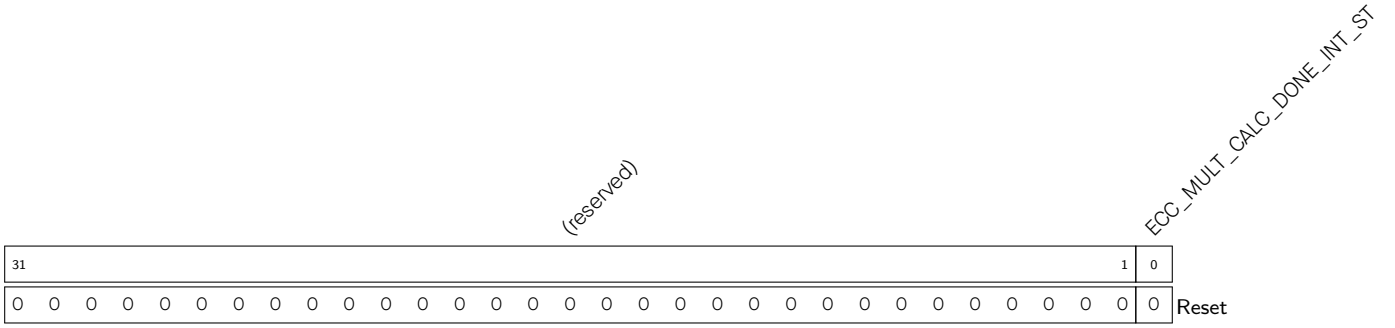
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 23.1. ECC_MULT_INT_RAW_REG (0x000C)



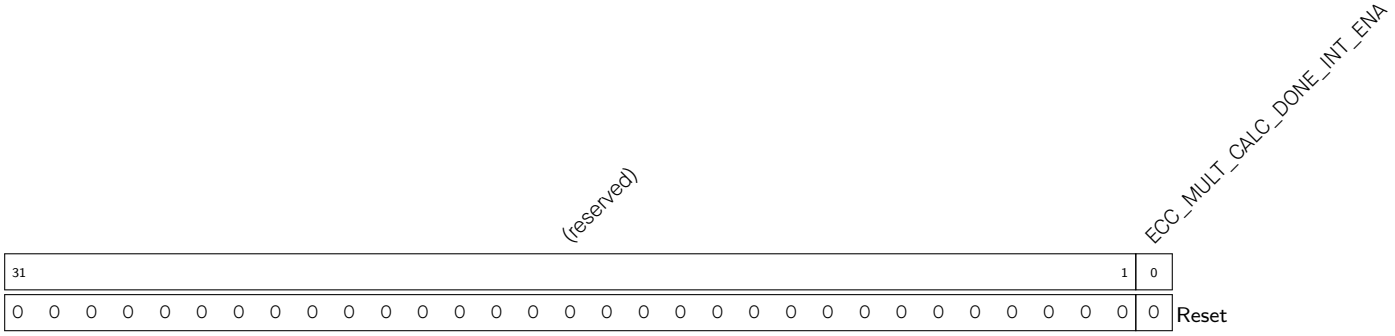
ECC_MULT_CALC_DONE_INT_RAW ECC_MULT_CALC_DONE_INT 的原始中断状态。(R/SS/WTC)

Register 23.2. ECC_MULT_INT_ST_REG (0x0010)



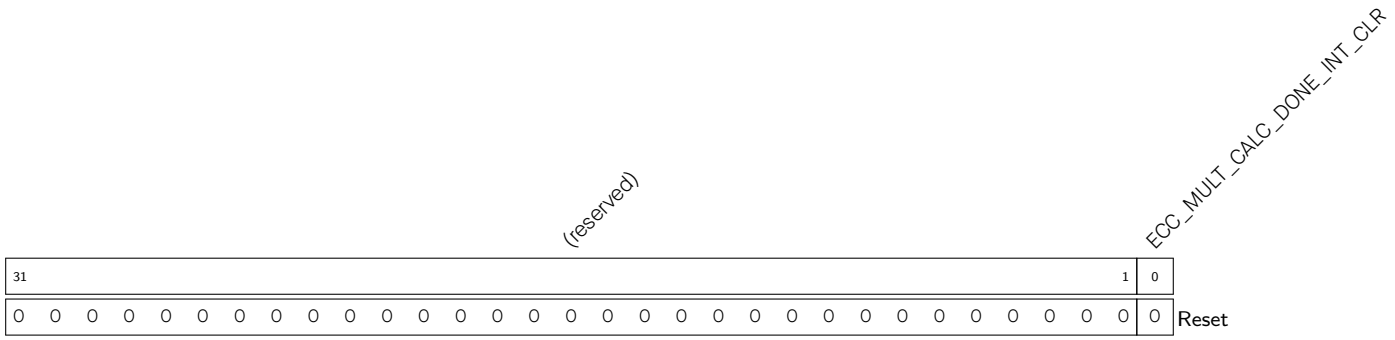
ECC_MULT_CALC_DONE_INT_ST ECC_MULT_CALC_DONE_INT 的屏蔽中断状态。(RO)

Register 23.3. ECC_MULT_INT_ENA_REG (0x0014)



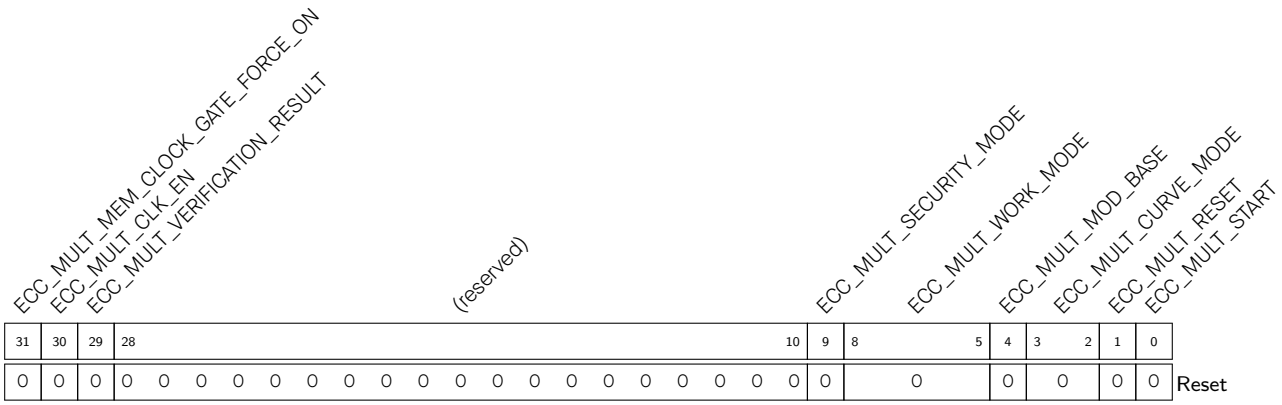
ECC_MULT_CALC_DONE_INT_ENA 写 1 使能 ECC_MULT_CALC_DONE_INT 中断。(R/W)

Register 23.4. ECC_MULT_INT_CLR_REG (0x0018)



ECC_MULT_CALC_DONE_INT_CLR 写 1 清除 ECC_MULT_CALC_DONE_INT 中断。(WT)

Register 23.5. ECC_MULT_CONF_REG (0x001C)



ECC_MULT_START 配置是否启动 ECC 加速器。此位运算结束后自动清 0。

0：无效

1：启动 ECC 加速器

(R/W/SC)

ECC_MULT_RESET 配置是否复位 ECC 加速器。

0：无效

1：复位

(WT)

ECC_MULT_CURVE_MODE 配置 ECC 加速器的曲线模式。

0：P-192

1：P-256

2：P-384

3：无效

(R/W)

ECC_MULT_MOD_BASE 配置在模运算操作中选择使用曲线模数或基点阶数。仅在工作模式 8-11 中有效。

0: n（基点阶数）

1: p（曲线模数）

(R/W)

见下页...

Register 23.5. ECC_MULT_CONF_REG (0x001C)

接上页...

ECC_MULT_WORK_MODE 配置 ECC 加速器的工作模式。

0：标准点乘
1：保留项，不可用
2：标准点验证
3：标准点验证 + 标准点乘
4：Jacobian 点乘
5：点加
6：Jacobian 点验证
7：标准点验证 + Jacobian 点乘
8：模加
9：模减
10：模乘
11：模除
(R/W)

ECC_MULT_SECURITY_MODE 配置是否针对点乘计算提升抗攻击性能。仅在工作模式 0、3、4 和 7 中有效。

0：禁用
1：启用
(R/W)

ECC_MULT_VERIFICATION_RESULT 表示 ECC 加速器的验证结果，仅在运算完成时有效。

0：验证失败
1：验证成功
(R/SS)

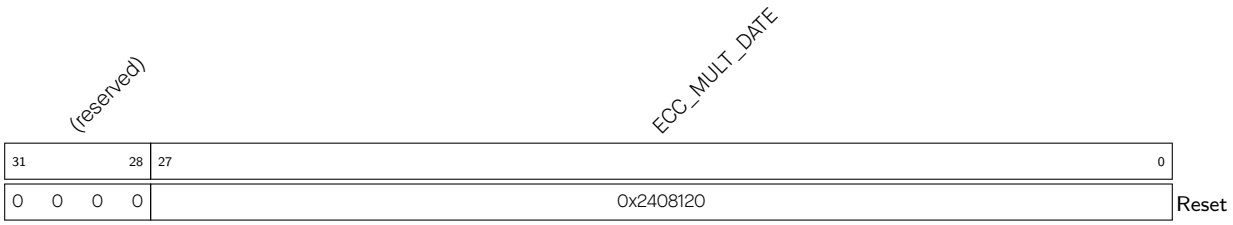
ECC_MULT_CLK_EN 配置是否强制打开寄存器门控。

0：无效
1：强制打开
(R/W)

ECC_MULT_MEM_CLOCK_GATE_FORCE_ON 配置是否强制打开 ECC 内存时钟门控。

0：无效
1：强制打开
(R/W)

Register 23.6. ECC_MULT_DATE_REG (0x00FC)



ECC_MULT_DATE 版本控制寄存器。(R/W)

第 24 章

HMAC 加速器 (HMAC)

24.1 概述

HMAC 模块通过 hash 算法 SHA-256 和密钥计算得到数据信息的信息认证码 (MAC)，采用 RFC 2104 标准。长度为 256 位的 HMAC 密钥存储在 eFuse 的密钥块中，可配置为读保护，即用户无法直接从 HMAC 加速器外部访问密钥。

24.2 主要特性

- 使用标准 HMAC-SHA-256 算法
- 基于 eFuse 密钥的 HMAC-SHA-256 算法计算，
  - 为提高安全性，软件在下行模式下无法访问其计算结果
  - 软件在上行模式下可访问其计算结果
- 下行模式下，支持生成 RSA 数字签名外设 (RSA_DS) 所需的密钥
- 下行模式下，支持重启软禁用的 JTAG

24.3 功能描述

HMAC 模块有两种工作模式，即上行模式和下行模式。上行模式中，由用户提供 HMAC 信息并回读计算结果；下行模式中，HMAC 模块为其他两个可能的内部硬件模块提供输入：一方面，如 JTAG 被暂时禁用，HMAC 可使能 JTAG；另一方面，HMAC 可用作存储在 RSA_DS 外设 flash 中数字签名参数的解密密钥。此外，这些计算在下行模式下自动进行，只要配置正确，就可以确保密钥及派生密钥材料的保密性。

说明：
芯片释放复位信号后，HMAC 模块将检查 eFuse 中是否烧写有 RSA_DS 外设所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行 RSA_DS_KEY 派生模式，完成相应密钥计算。

24.3.1 上行模式

在上行模式下计算 HMAC 值，用户应执行以下步骤：

1. 初始化 HMAC 模块，进入上行模式。
2. 将正确填充的信息写入 HMAC 中，一次写入一个块。
3. 从 HMAC 回读计算结果。

有关此过程的详细步骤，可参见章节 24.5。

说明：

使用上行模式的较多为支持 HMAC-SHA-256 算法的挑战-应答协议。假设挑战-应答协议中的通讯双方分别称为 A 和 B，想要交换的数据信息为 M，则协议的一般验证流程为：

- A 计算出一个特殊的随机数 M。
- A 将 M 发送给 B。
- B（通过 M 和密钥）计算 HMAC 结果，并将结果发送给 A。
- A（通过 M 和密钥）内部计算 HMAC 结果。
- A 比较两次计算结果。如结果相同，则验证通过 B 的身份。

24.3.2 下行模式 - JTAG 启动功能

JTAG 可被 eFuse 暂时关闭，关闭后，用户可使用 HMAC 模块重启 JTAG 功能。（详细步骤可参见章节 7 *eFuse 控制器 (EFUSE)*。）该模式下，用户应提供某 eFuse 密钥的 HMAC 计算结果，HMAC 模块将检查用户提供的数值与计算的数值是否匹配。如果二者相同，JTAG 将被启动，直到用户再次调用 HMAC 模块清除计算结果并关闭 JTAG。

用户可执行以下步骤重启 JTAG：

1. 通过初始化时钟和 HMAC 复位信号启动 HMAC 模块，并通过配置 HMAC_SET_PARA_PURPOSE_REG 位进入下行 JTAG 启动模式，等待计算完成。详细步骤可参见章节 24.5。
2. 将 1 写入 HMAC_SOFT_JTAG_CTRL_REG 寄存器，进入 JTAG 重启模式。
3. 将 256 位的 HMAC 值写入寄存器 HMAC_WR_JTAG_REG。该值是预先在本地使用 SHA-256 和已被写入 eFuse 的密钥对 32 字节的 0x00 进行 HMAC 计算得到的。需以大端字序分 8 次写入，每次 32 位。
4. 如果由 eFuse 中密钥计算的 HMAC 结果与用户在步骤 3 中写入的数值一致，则 JTAG 重启。否则，JTAG 保持关闭状态。
5. 在用户将 1 写入寄存器 HMAC_SET_INVALIDATE_JTAG_REG 或复位重启之后，JTAG 将保持关闭状态。如需再次重启 JTAG，请重复上述步骤。

24.3.3 下行模式 - RSA 数字签名外设密钥派生功能

RSA 数字签名外设 (RSA_DS) 模块使用 AES-CBC 算法加密其参数。HMAC 模块作为密钥导出函数 (KDF) 导出解密上述参数的 AES 密钥。

在启用 RSA_DS 外设之前，用户必须先通过 HMAC 模块计算得到 RSA_DS 外设工作时所需的密钥。详细信息可参见章节 *27 RSA 数字签名外设 (RSA_DS)*。芯片上电后，HMAC 模块将检查 eFuse 中是否烧写有 RSA_DS 外设所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行 RSA_DS_KEY 派生模式，完成相应密钥计算。

24.4 HMAC eFuse 配置

每个烧写入 eFuse 密钥块的 HMAC 密钥都有其功能，指定该密钥用于实现哪一功能。HMAC 中，未配置匹配的功能数值的密钥无法执行计算。当前 HMAC 模块共支持三种功能：下行模式下的 JTAG 重启功能和 RSA_DS 密

钥导出功能，以及上行模式下的 HMAC 计算功能。表 24.4-1 列出了上述各项功能，以及一种可指定密钥同时用于 JTAG 重启和 RSA_DS 密钥导出的功能。

启动 HMAC 模块进行计算之前，用户应先读取章节 7 *eFuse 控制器 (EFUSE)* 中 EFUSE_KEY_PURPOSE_x 的值（基于 eFuse 中共存在 6 个密钥，故 x 的值为 0 ~ 5，其中 EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_1 属于寄存器 EFUSE_RD_REPEAT_DATA1_REG，EFUSE_KEY_PURPOSE_2 ~ EFUSE_KEY_PURPOSE_5 属于寄存器 EFUSE_RD_REPEAT_DATA2_REG），检查 eFuse 中是否已烧写密钥。例如，在上行模式下，如果 EFUSE_KEY_PURPOSE_0~5 中没有 EFUSE_KEY_PURPOSE_HMAC_UP，则说明 eFuse 中没有上行模式可用的密钥。此时，可按照下述步骤将密钥烧写入 eFuse 中：

1. 准备一个 256 位 HMAC 密钥，将其烧写到空的 eFuse 密钥块 y 中。eFuse 中共有 6 个密钥块，分别为 4 ~ 9，因此 y 的值为 4 ~ 9。也就是说，若密钥为 key0，则对应的密钥块为 block4。接着，将功能对应数值烧写至 EFUSE_KEY_PURPOSE_$(y-4)$。例如在上行模式下，烧写密钥后，应将 EFUSE_KEY_PURPOSE_HMAC_UP（对应值为 8）写入 EFUSE_KEY_PURPOSE_$(y-4)$。更多烧写 eFuse 密钥的详细信息，可参见章节 7 *eFuse 控制器 (EFUSE)*。
2. 如有需要，配置 eFuse 密钥块读保护功能，使用户无法读取密钥值。用户应保留步骤中生成的随机密钥副本以便验证。

请注意，功能为 EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL 的密钥既可用于 JTAG 重启，也可用于 RSA_DS 密钥导出。

表 24.4-1. HMAC 功能及配置数值

功能 (m)	模式	数值	描述
JTAG 重启	下行模式	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
RSA_DS 密钥导出	下行模式	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC 计算	上行模式	8	EFUSE_KEY_PURPOSE_HMAC_UP
JTAG 重启和 RSA_DS 密钥导出	下行模式	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

选取 eFuse 的密钥块与 HMAC 功能

eFuse 控制器共有 6 个密钥块，即 KEY0 ~ 5 。用户将编号 n 写入寄存器 HMAC_SET_PARA_KEY_REG，表示选择 KEYn 作为本次 HMAC 模块运行时使用的密钥。

将所使用功能对应的数值写入寄存器 HMAC_SET_PARA_PURPOSE_REG 完成配置（可参见章节 24.5）。需要注意的是，eFuse 存储器中的密钥在烧写时都定义了功能用途，只有当 HMAC 的配置功能与 KEYn 定义的功能用途相匹配时，HMAC 模块才会执行配置好的计算任务。否则，HMAC 模块将返回匹配错误结果并结束当前计算任务。

比如，如果用户选择了 KEY3 作为本次计算的密钥，且烧写入 EFUSE_KEY_PURPOSE_3 中的数值为 6，即功能 EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG。参照表 24.4-1 可知，KEY3 是用于 JTAG 重启的密钥。如果此时寄存器 HMAC_SET_PARA_PURPOSE_REG 中配置的数值也是 6，HMAC 模块才会启动 JTAG 重启功能的计算。

从密钥管理器中选取密钥

用户可以在密钥管理器中部署 HMAC 的密钥。要选择密钥管理器中的该密钥，请将密钥编号 7 写入注册表 HMAC_SET_PARA_KEY_REG。用户从密钥管理器中选取密钥时，注册表中 HMAC 操作的预定目标会自动设置

为"HMAC Calculation"，其他设置都不会生效。

24.5 配置流程

用户调用 ESP32-C5 中 HMAC 流程如下：

24.5.1 启动 HMAC 模块

1. 置位寄存器 PCR_HMAC_CLK_EN 中 HMAC 和 SHA 的外设时钟位，清除寄存器 PCR_HMAC_RST_EN 中相应的外设复位位。关于如何配置时钟和重启信号，请参见章节 9 复位和时钟。
2. 将 1 写入寄存器 HMAC_SET_START_REG。

24.5.2 配置 HMAC 密钥和密钥功能

1. 将密钥功能 m 写入寄存器 HMAC_SET_PARA_PURPOSE_REG。表 24.4-1 列出了 m 对应的数值，可参见章节 24.4。
2. 将数值 n 写入寄存器 HMAC_SET_PARA_KEY_REG，选择 eFuse 存储器中的 KEYn 作为本次计算的密钥（n 的取值范围为 0 ~ 5，7），可参见章节 24.4。
3. 将 1 写入寄存器 HMAC_SET_PARA_FINISH_REG，完成配置。
4. 读取寄存器 HMAC_QUERY_ERROR_REG。如果返回值为 1，表明选取的密钥块与配置的密钥功能不匹配，本次计算任务中止。如果返回值为 0，表明选取的密钥块与配置的密钥功能匹配，可以执行计算流程。
5. 如果 HMAC_SET_PARA_PURPOSE_REG 的数值不为 8，表明 HMAC 模块当前处于下行模式，请参考章节 24.5.3 进行下一步配置。如果数值为 8，表明 HMAC 模块当前处于上行模式，请参考章节 24.5.4 进行下一步配置。

24.5.3 下行模式流程

1. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
2. 如需清除计算结果，并确保后续不再使用相关硬件（JTAG 或 RSA_DS），用户可将 1 写入寄存器 HMAC_SET_INVALIDATE_JTAG_REG，清除 JTAG 密钥生成的结果；也可将 1 写入寄存器 HMAC_SET_INVALIDATE_DS_REG，清除 RSA_DS_KEY 生成的结果。此后，如需重启 JTAG 或 RSA_DS 外设，需重新进行 HMAC 调用流程。

24.5.4 上行模式流程

1. 上行模式下传输数据：
   (a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
   (b) 用户根据章节 24.6.1 的描述对数据进行 SHA 附加填充并获取数据块 Block_n ($n >= 0$)
   (c) 根据数据块 n 的数量，选择其中一种操作：
   - 如果 Block_n 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位：
     i. 将 512 位的数据块 Block_n 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。

ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

iii. 写 1 到寄存器 HMAC_ONE_BLOCK_REG 中，完成数据传输。

- 如果 Block_n 是最后一个填充数据块：

  i. 将 512 位的数据块 Block_0 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。

  ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

  iii. 自动完成数据传输。

- 如果 Block_n 是倒数第二个填充数据块：

  i. 将 512 位的数据块 Block_n 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。

  ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

  iii. 写 1 到寄存器 HMAC_SET_MESSAGE_PAD_REG 中，随后跳转到步骤 c 继续数据传输。

- 如果 Block_n 既不是最后一个也不是倒数第二个数据块：

  i. 将 512 位的数据块 Block_n 写入寄存器 HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。

  ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

  iii. 写 1 到寄存器 HMAC_SET_MESSAGE_ING_REG 中，随后跳转到步骤 c 继续数据传输。

2. 读取上行模式下的结果 hash 值：

   (a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

   (b) 从寄存器 HMAC_RD_RESULT_*n*_REG (*n*: 0-7) 中读取 hash 结果数值。

   (c) 将 1 写入寄存器 HMAC_SET_RESULT_FINISH_REG，结束当次计算，并同时清除计算结果。

   (d) 上行模式下的操作完成。

说明：

RSA_DS 外设和 HMAC 模块可直接调用或在内部使用 SHA 加速器，但两者不能同时共享硬件资源。因此在 HMAC 模块运行过程中，SHA 模块无法被 CPU 和 RSA_DS 外设调用。

24.6 HMAC 算法细节

24.6.1 附加填充位

HMAC 模块中采用 SHA-256 作为加密 hash 算法。该算法中，若待输入数据的总位数不是 512 的倍数，用户须在软件中应用 SHA-256 附加填充算法。SHA-256 附加填充算法与 FIPS PUB 180-4 中章节 *Padding the Message* 所描述相同。下行模式下，用户无需输入数据或进行数据填充，HMAC 模块默认使用 32 字节 0x00 方式重启 JTAG，以及 32-byte 0xff 方式生成 RSA_DS 外设的 AES 密钥。

为方便阅读，以下将简要介绍数据填充的步骤。如图 24.6-1 所示，假设待处理数据长度为 m 个位，填充步骤如下：

1. 在待处理数据末尾附加一个 “1”。
2. 附加 k 个 “0”。其中，k 为满足 $m+1+k \equiv 448 (mod 512)$ 的最小非负数。
3. 附加一个 64 位的整数值作为二进制块。该二进制块的内容为待填充数据作为一个大端二进制整数值 m 的长度。

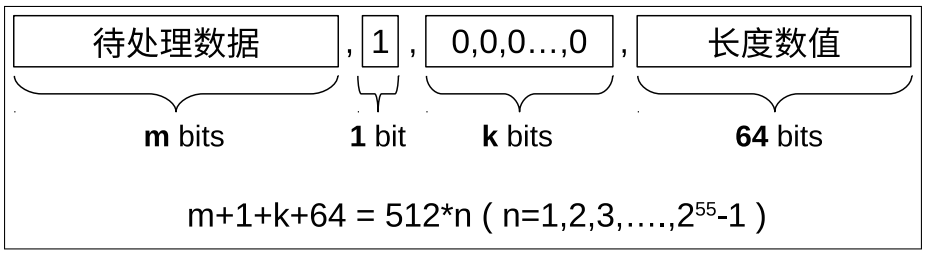


图 24.6-1. HMAC 附加填充位示意图

上行模式下，若待填充数据总位数是 512 的整数倍，则用户可通过将 1 写入 HMAC_SET_MESSAGE_END_REG 配置由硬件完成 SHA 附加填充操作，也可通过将 1 写入 HMAC_SET_MESSAGE_PAD_REG 自行完成填充操作。若待填充数据总位数不是 512 的整数倍，则用户只能自行完成 SHA 附加填充操作。数据填充操作完成后，用户应参照章节 24.5 完成后续配置。

24.6.2 HMAC 算法结构

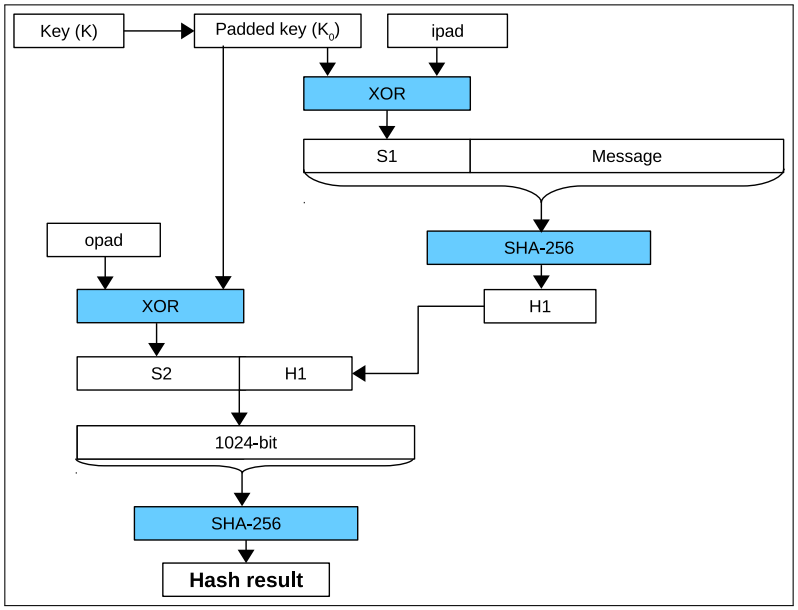


图 24.6-2. HMAC 结构示意图

HMAC 模块中应用的算法结构示意图如 24.6-2 所示。这是 RFC 2104 中描述的标准 HMAC 算法。

图 24.6-2 中，

1. ipad 是由 64 个 0x36 字节组成的 512 位数据块。
2. opad 是由 64 个 0x5c 字节组成的 512 位数据块。

首先，HMAC 模块在 256 位的密钥 K 的位序列后附加上 256 位的 0 序列，得到 512 位的 K_0。再对 K_0 和 ipad 进行异或运算，得到 512 位的 S1。将总位数为 512 倍数的待输入数据附加到 512 位的 S1 数值后，使用 SHA-256 加密算法计算得到 256 位的 H1。

HMAC 模块通过对 K_0 和 opad 进行异或运算得到 S2，将 256 位的 hash 计算结果附加到 512 位的 S2 数值后，得到 768 位长度的序列，使用章节 24.6.1 中描述的 SHA 附加填充算法将该序列填充成 1024 位的序列，最后使用 SHA-256 加密算法计算得到的最终 hash 结果（256 位）。

24.7 寄存器列表

本小节的所有地址均为相对于 HMAC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

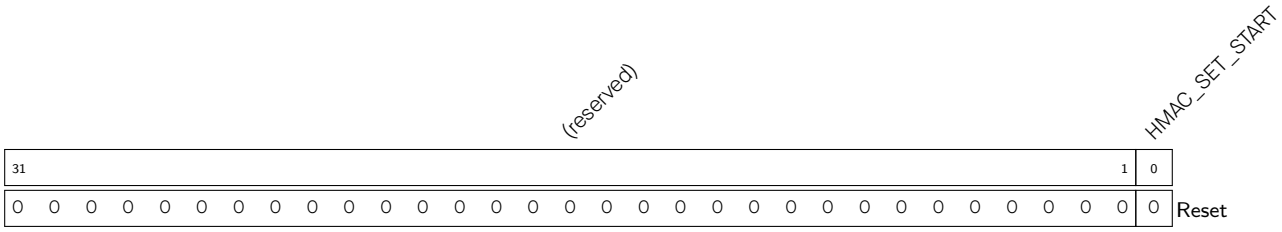
名称	描述	地址	访问
控制/状态寄存器			
HMAC_SET_START_REG	HMAC 开始控制寄存器	0x0040	WS
HMAC_SET_PARA_FINISH_REG	HMAC 配置完成寄存器	0x004C	WS
HMAC_SET_MESSAGE_ONE_REG	HMAC 信息控制寄存器	0x0050	WS
HMAC_SET_MESSAGE_ING_REG	HMAC 信息继续寄存器	0x0054	WS
HMAC_SET_MESSAGE_END_REG	HMAC 信息终止寄存器	0x0058	WS
HMAC_SET_RESULT_FINISH_REG	HMAC 读取结果完成寄存器	0x005C	WS
HMAC_SET_INVALIDATE_JTAG_REG	注销 JTAG 结果寄存器	0x0060	WS
HMAC_SET_INVALIDATE_DS_REG	注销数字签名结果寄存器	0x0064	WS
HMAC_QUERY_ERROR_REG	存储用户配置的密钥和功能的配对结果	0x0068	RO
HMAC_QUERY_BUSY_REG	存储 HMAC 模块的忙碌状态	0x006C	RO
HMAC_SET_MESSAGE_PAD_REG	软件填充寄存器	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block 信息寄存器	0x00F4	WS
配置寄存器			
HMAC_SET_PARA_PURPOSE_REG	HMAC 参数配置寄存器	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC 密钥配置寄存器	0x0048	WO
HMAC_WR_JTAG_REG	重启 JTAG 寄存器 1	0x00FC	WO
HMAC_SOFT_JTAG_CTRL_REG	重启 JTAG 寄存器 0	0x00F8	WS
HMAC 信息块			
HMAC_WR_MESSAGE_*n*_REG	信息寄存器 *n* (*n*: 0-15)	0x0080+4×*n*	WO
HMAC 上行结果			
HMAC_RD_RESULT_*n*_REG	Hash 结果寄存器 *n* (*n*: 0-7)	0x00C0+4×*n*	RO
版本寄存器			
HMAC_DATE_REG	版本控制寄存器	0x01FC	R/W

24.8 寄存器

本小节的所有地址均为相对于 HMAC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 24.1. HMAC_SET_START_REG (0x0040)



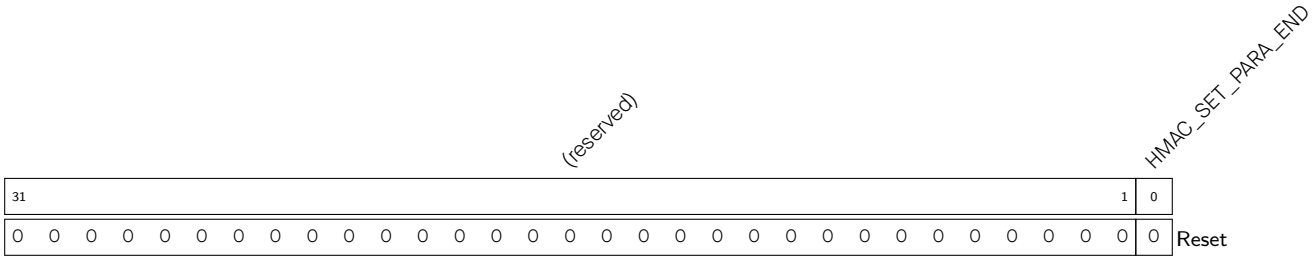
HMAC_SET_START 配置是否启动 HMAC。

0：关闭 HMAC

1：启动 HMAC

(WS)

Register 24.2. HMAC_SET_PARA_FINISH_REG (0x004C)



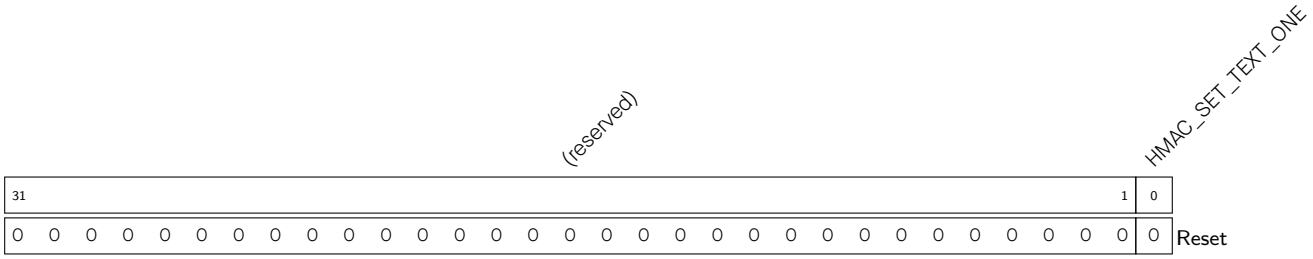
HMAC_SET_PARA_END 配置是否完成 HMAC 配置。

0：无效

1：完成配置

(WS)

Register 24.3. HMAC_SET_MESSAGE_ONE_REG (0x0050)



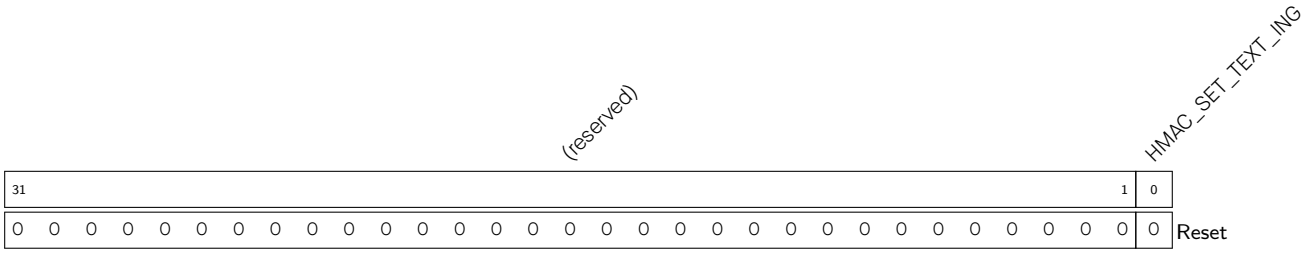
HMAC_SET_TEXT_ONE 调用 SHA 计算一个数据块。

0：无效

1：调用

(WS)

Register 24.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



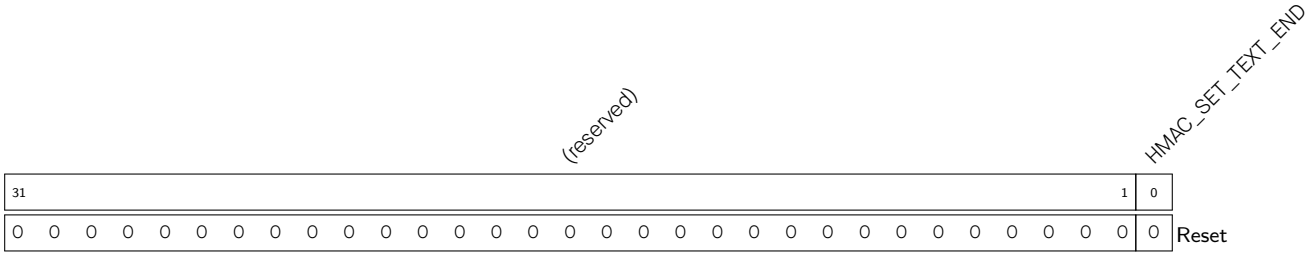
HMAC_SET_TEXT_ING 配置是否存在未处理的数据块。

0：不存在

1：存在

(WS)

Register 24.5. HMAC_SET_MESSAGE_END_REG (0x0058)



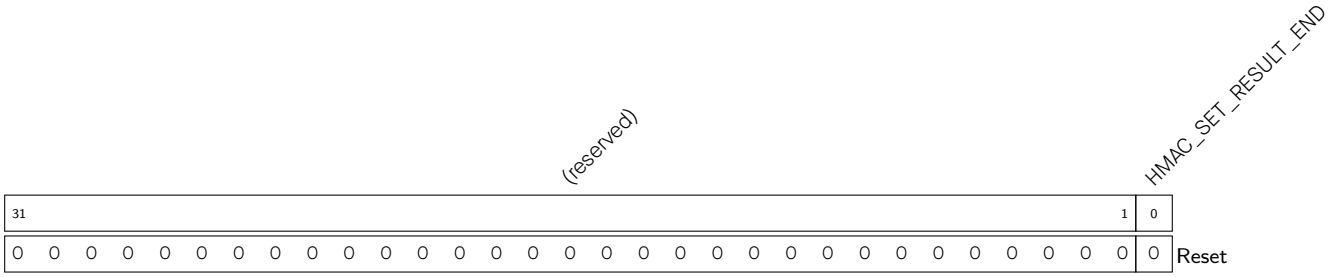
HMAC_SET_TEXT_END 配置是否开始硬件填充。

0：无效

1：开始硬件填充

(WS)

Register 24.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



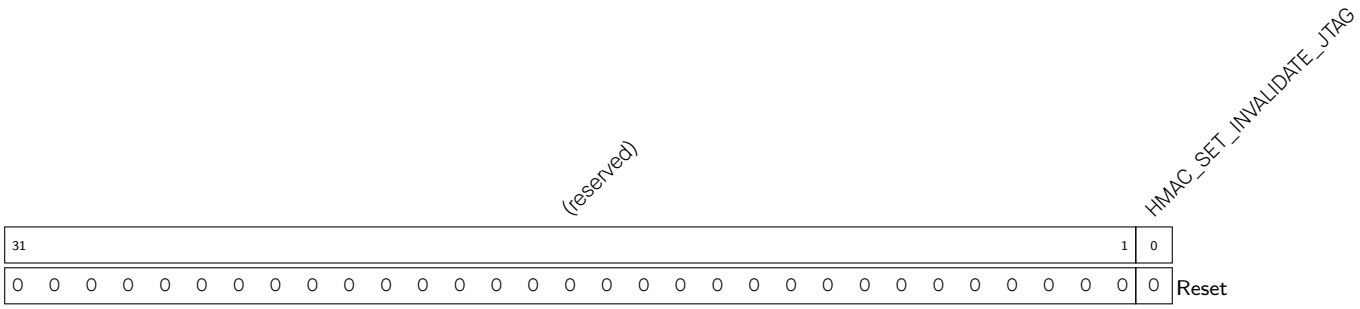
HMAC_SET_RESULT_END 配置是否退出上行模式并清空计算结果。

0：无效

1：退出上行模式并清空计算结果

(WS)

Register 24.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



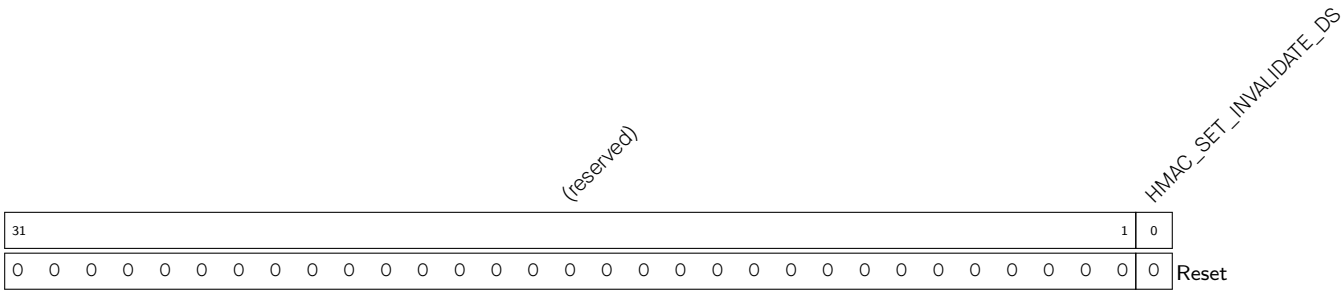
HMAC_SET_INVALIDATE_JTAG 配置是否清空下行模式下 JTAG 重启功能的计算结果。

0：无效

1：清空计算结果

(WS)

Register 24.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



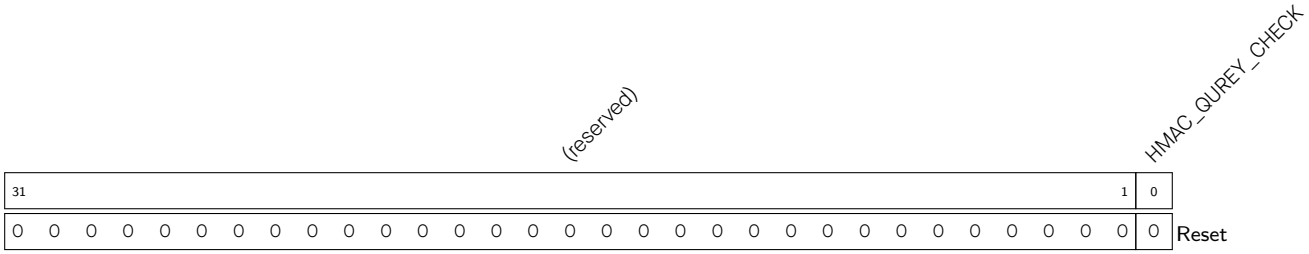
HMAC_SET_INVALIDATE_DS 配置是否清空下行模式下 RSA_DS 功能的计算结果。

0：无效

1：清空计算结果

(WS)

Register 24.9. HMAC_QUERY_ERROR_REG (0x0068)



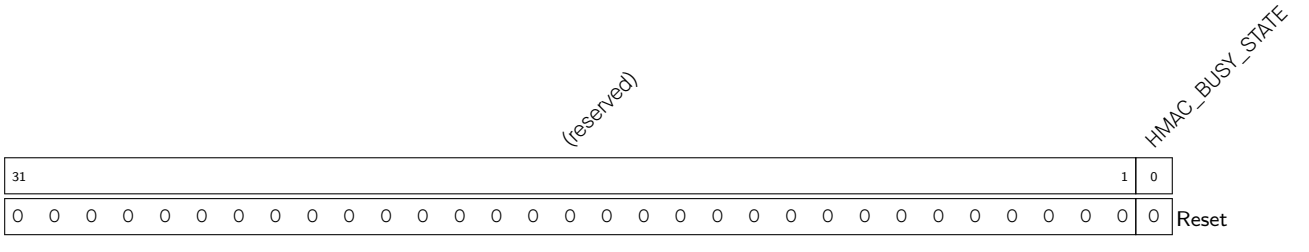
HMAC_QUREY_CHECK 表示 HMAC 密钥与功能是否匹配。

0：HMAC 密钥和功能匹配。

1：不匹配

(RO)

Register 24.10. HMAC_QUERY_BUSY_REG (0x006C)



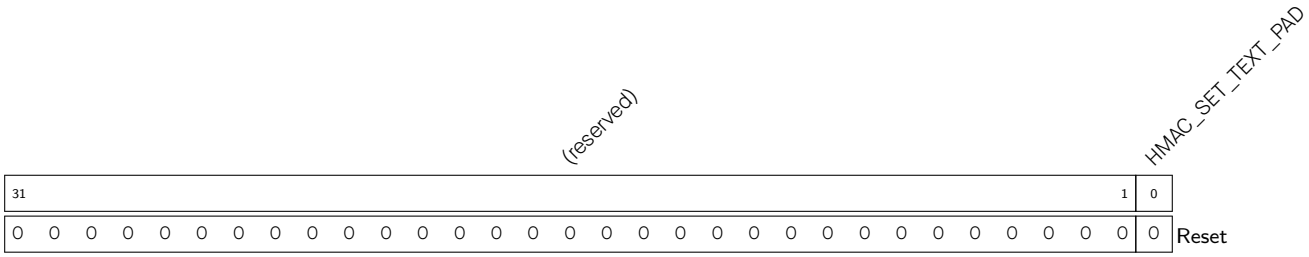
HMAC_BUSY_STATE 表示 HMAC 是否处于忙碌状态。执行计算任务之前，请确保 HMAC 已空闲。

0：空闲

1：HMAC 仍处于工作状态

(RO)

Register 24.11. HMAC_SET_MESSAGE_PAD_REG (0x00F0)



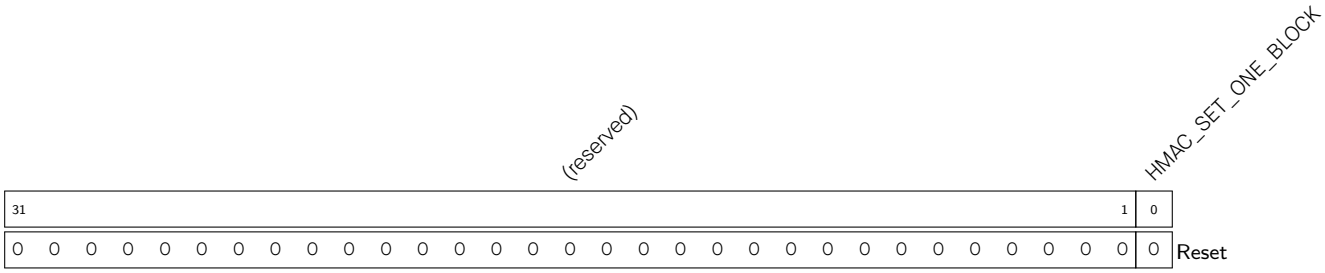
HMAC_SET_TEXT_PAD 表示是否由软件执行填充操作。

0：不由软件执行

1：由软件执行

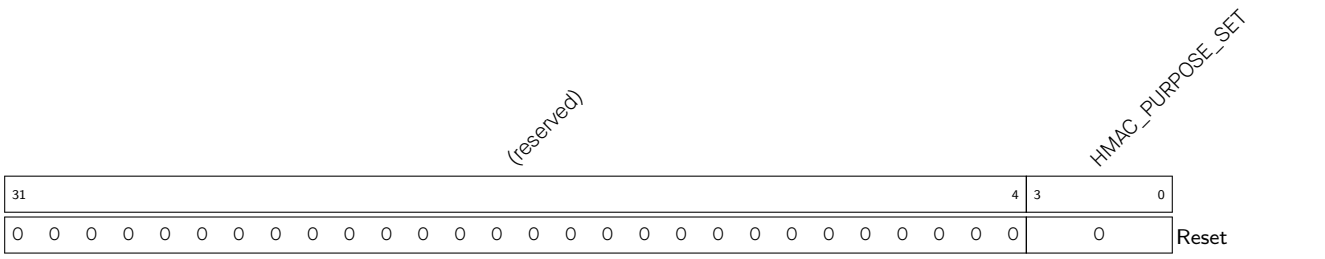
(WO)

Register 24.12. HMAC_ONE_BLOCK_REG (0x00F4)



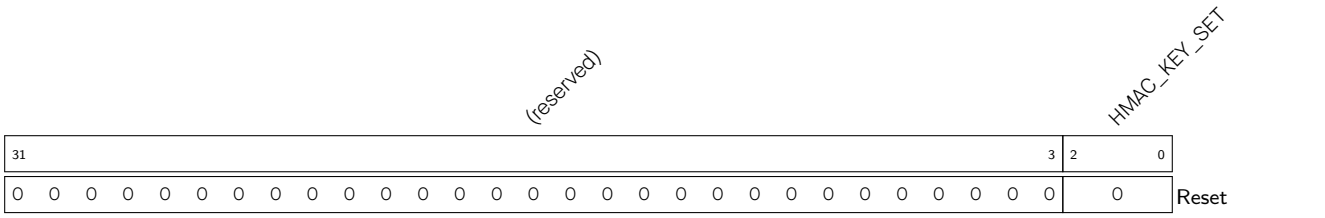
HMAC_SET_ONE_BLOCK 置 1 表示 Block_n 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位，无需填充。(WS)

Register 24.13. HMAC_SET_PARA_PURPOSE_REG (0x0044)



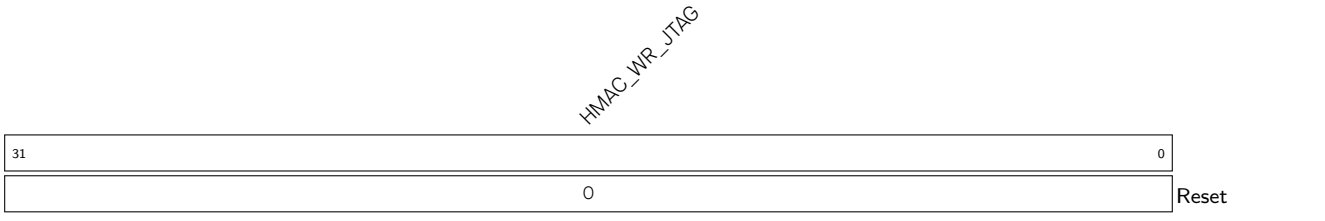
HMAC_PURPOSE_SET 设置 HMAC 功能，请参阅表 24.4-1。(WO)

Register 24.14. HMAC_SET_PARA_KEY_REG (0x0048)



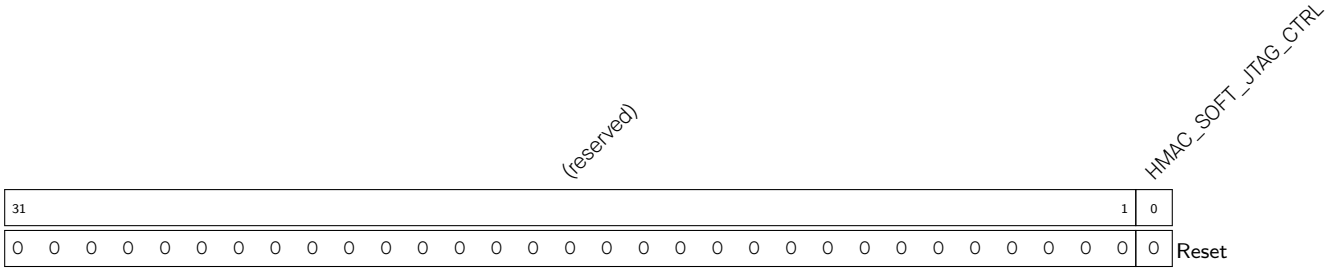
HMAC_KEY_SET 配置 HMAC 密钥。共有 6 个 eFuse 密钥和 1 个 key manager 密钥，编号 0 ~ 5 对应 6 个 eFuse 密钥，编号 7 对应 key manager 密钥，将选择的密钥编号写入该字段即可。(WO)

Register 24.15. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG 写入用于重启 JTAG 的输入比较数值。(WO)

Register 24.16. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)



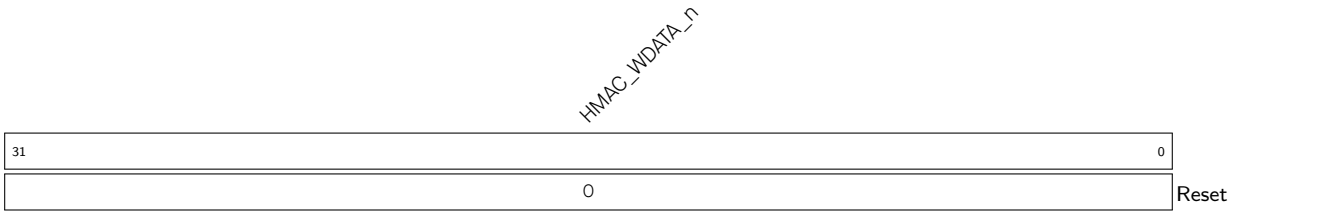
HMAC_SOFT_JTAG_CTRL 配置是否开启 JTAG 验证模式。

0：关闭

1：开启

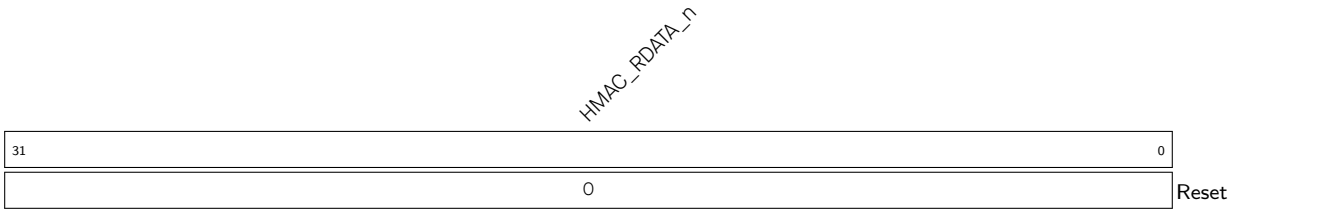
(WS)

Register 24.17. HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4×*n*)



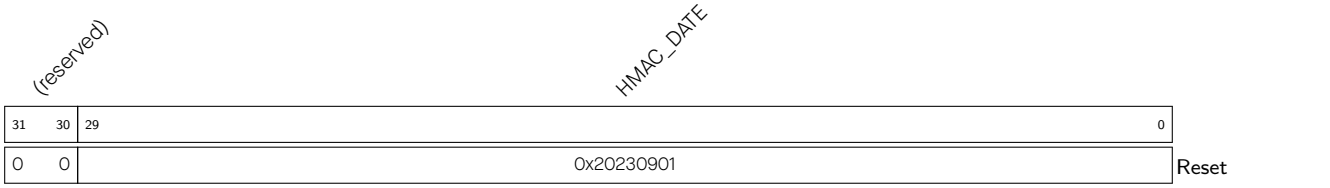
HMAC_WDATA_*n* 表示信息的第 *n* 个 32 位数据信息。(WO)

Register 24.18. HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4×*n*)



HMAC_RDATA_*n* 读取 hash 结果的第 *n* 个 32 位。(RO)

Register 24.19. HMAC_DATE_REG (0x01FC)



HMAC_DATE 版本控制寄存器。(R/W)

第 25 章

RSA 加速器 (RSA)

25.1 概述

RSA 加速器可为多种运用于“RSA 非对称式加密演算法”的高精度计算提供硬件支持，能够极大地降低此类运算的运行时间和软件复杂度。与纯软件 RSA 算法相比，硬件 RSA 加速器的运算速度更快。RSA 加速器还支持多种“运算子长度”，具有很高的灵活性。

25.2 主要特性

RSA 加速器支持以下功能：

- 大数模幂运算（支持两个加速选项）
- 大数模乘运算
- 大数乘法运算
- 多种运算子长度
- 支持在运算完成后触发中断

25.3 功能描述

RSA 加速器的激活需使能 PCR_RSA_CONF_REG 寄存器中的 PCR_RSA_CLK_EN 位，并同时清零 PCR_RSA_RST_EN 位。此外，还需要清零 PCR_DS_RST_EN 和 PCR_ECDSA_RST_EN 以复位 RSA 数字签名外设 (RSA_DS) 和 ECDSA 数字签名外设 (ECDSA_DS)。

不过，RSA 加速器激活后还须等待 RSA 相关存储器 初始化完成后才能开始工作。具体来说，RSA_QUERY_CLEAN_REG 读 0 时初始化开始，读 1 时初始化完成。因此，在复位后首次使用 RSA 加速器时，软件需要先查询 RSA_QUERY_CLEAN_REG 的值是否为 1，以确保 RSA 加速器可正常工作。

此外，RSA 加速器支持中断功能，可对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。RSA 加速器的中断功能默认开启。

注意：

ESP32-C5 的 RSA 数字签名外设 (RSA_DS) 模块在工作期间会调用 RSA 加速器。此时，用户无法正常访问 RSA 加速器。

25.3.1 大数模幂运算

大数模幂运算的算法是 $Z = X^Y \bmod M$，它是基于蒙哥马利乘法 (Montgomery Multiplication) 实现的。因此，对于大数模幂运算，除了需要运算子 X、Y、M 外，还需要额外两个运算子，即参数 $\overline{r}$ 和 M'。这两个参数需要通过软件提前运算得到。

RSA 加速器支持运算子长度为 $N = 32 \times x$ ($x \in \{1, 2, 3, \ldots, 96\}$) 的大数模幂运算。$Z$、$X$、$Y$、$M$ 和 $\overline{r}$ 的位宽为这 96 种中的任意一种，要求它们的位宽必须相同，而 M' 的位宽始终是 32。

设进制数

$$b = 2^{32}$$

则运算子可以由若干个 b 进制数来表示：

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

其中 $Z_{n-1}\cdots Z_0$、$X_{n-1}\cdots X_0$、$Y_{n-1}\cdots Y_0$、$M_{n-1}\cdots M_0$、$\overline{r}_{n-1}\cdots \overline{r}_0$ 分别表示一个 b 进制数，位宽皆为 32。且 Z_{n-1}、X_{n-1}、Y_{n-1}、M_{n-1}、$\overline{r}_{n-1}$ 分别为 Z、X、Y、M、$\overline{r}$ 最高位的 b 进制数，而 Z_0、X_0、Y_0、M_0、$\overline{r}_0$ 分别为 Z、X、Y、M、$\overline{r}$ 最低位的 b 进制数。

另设 $R = b^n$，则计算得参数 $\overline{r} = R^2 \bmod M$。

另外 M' 可使用下方公式计算：

$$M' = -M^{-1} \bmod b$$

其中，M^{-1} 为 M 对 R 的模逆元，可使用扩展二进制 GCD 算法计算。

大数模幂运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。
   (a) 对寄存器 RSA_MODE_REG 写入 ($\frac{N}{32} - 1$)。
   (b) 对寄存器 RSA_M_PRIME_REG 写入 M'。
   (c) 根据需要配置加速选项相关寄存器。请参照章节 25.3.4 获取详细信息。
3. 将 X_i、Y_i、M_i、$\overline{r}_i (i \in \{0, 1, \ldots, n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Y_MEM、RSA_M_MEM、RSA_Z_MEM。每块存储器的容量都是 96 字 (word)。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

   只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 RSA_SET_START_MODEXP 写入 1 启动计算。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生。

6. 从存储器 RSA_Z_MEM 读出运算结果 $Z_i(i \in \{0, 1, \ldots, n-1\})$。
7. 若中断功能已开启，对 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_Y_MEM 中的 Y_i、存储器 RSA_M_MEM 中的 M_i、寄存器 RSA_M_PRIME_REG 中的 M' 都不会变化。但是，存储器 RSA_X_MEM 中的 X_i 与存储器 RSA_Z_MEM 中的 $\overline{r}_i$ 都已经被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

25.3.2 大数模乘运算

大数模乘运算 $Z = X \times Y \bmod M$ 也是基于蒙哥马利乘法实现的。因此，与大数模幂运算类似，也需要预先通过软件计算额外的两个运算子 $\overline{r}$ 和 M'。

RSA 加速器也支持 96 种运算子长度的大数模乘运算。

大数模乘运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。
   (a) 对寄存器 RSA_MODE_REG 写入 $(\frac{N}{32} - 1)$。
   (b) 对寄存器 RSA_M_PRIME_REG 写入 M'。
3. 将 X_i、Y_i、M_i、$\overline{r}_i$ $(i \in \{0, 1, \ldots, n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Y_MEM、RSA_M_MEM、RSA_Z_MEM。每块存储器的容量都是 96 字 (word)。

   每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

   只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 RSA_SET_START_MODMULT 写入 1。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生。
6. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in \{0, 1, \ldots, n-1\})$。
7. 若中断功能已开启，对寄存器 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_X_MEM 中的 X_i、存储器 RSA_Y_MEM 中的 Y_i、存储器 RSA_M_MEM 中的 M_i、寄存器 RSA_M_PRIME_REG 中的 M' 都不会变化。但是，存储器 RSA_Z_MEM 中的 $\overline{r}_i$ 已经被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

25.3.3 大数乘法运算

大数乘法运算实现了 $Z = X \times Y$。其中 Z 的长度是运算子 X、Y 长度的两倍，所以 RSA 加速器只支持运算子 X、Y 长度为 $N = 32 \times x$ $(x \in \{1, 2, 3, \ldots, 48\})$ 的大数乘法运算。运算子 Z 的长度 $\hat{N}$ 为 $2 \times N$。

大数乘法运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。对寄存器 RSA_MODE_REG 写入 $(\frac{\hat{N}}{32} - 1)$，即 $(\frac{N}{16} - 1)$。

3. 将 X_i、Y_i ($i \in \{0, 1, \ldots, n-1\}$) 分别写入存储器 RSA_X_MEM、RSA_Z_MEM。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。n 为 $\frac{N}{32}$。

   X_i ($i \in \{0, 1, \ldots, n-1\}$) 要填充到存储器 RSA_X_MEM 中的第 i 个字对应的地址中，但需要注意的是，Y_i ($i \in \{0, 1, \ldots, n-1\}$) 并不是要填充到存储器 RSA_Z_MEM 中的第 i 个字对应的地址中，而是需要填充到存储器 RSA_Z_MEM 中的第 $n+i$ 个字对应的地址中，即存储器 RSA_Z_MEM 的基地址加上偏移量 $4 \times (n+i)$。

   只需要根据运算子长度，将这两个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。

4. 对 RSA_SET_START_MULT 写入 1。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生。
6. 从存储器 RSA_Z_MEM 读出运算结果 Z_i ($i \in \{0, 1, \ldots, \hat{n}-1\}$)。$\hat{n}$ 为 $2 \times n$。
7. 若中断功能已开启，对 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_X_MEM 中的 X_i 都不会变化。但是，存储器 RSA_Z_MEM 中的 Y_i 已经被覆盖。所以当需要连续运算时，只需要更新必需的寄存器与存储器即可。

25.3.4 控制加速

对于大数模幂运算，ESP32-C5 的 RSA 加速器还特别提供 SEARCH 和 CONSTANT_TIME 两个选项，可进一步提高运算速度。默认情况下，这两个选项均处于不额外加速状态，可以单独使用，也可以同时使用。值得注意的是，即使这两个选项均处于不额外加速状态，硬件 RSA 加速器也比纯软件实现的 RSA 算法快很多。

具体来说，当这两个选项均处于不额外加速状态时，求解 $Z = X^Y \bmod M$ 的时间开销完全由运算子长度决定。否则，只要有某个选项携带有额外加速效果，那么运算的时间开销还与 Y 的 0/1 分布有关。

为了更清楚地说明问题，首先假设 Y 的二进制表示为：

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

其中，

- N 代表 Y 的长度，
- $\widetilde{Y}_t$ 的值为 1，
- $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ 的值均为 0，
- 且 $\widetilde{Y}_{t-1}$, $\widetilde{Y}_{t-2}$, ..., $\widetilde{Y}_0$ 中包括 m 个 0，其余 $t-m$ 全部为 1，即 $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ 的汉明重量 (Hamming weight) 为 $t-m$。

此时，当启动任一额外加速选项时：

- SEARCH 选项（RSA_SEARCH_ENABLE 置 1 开启额外加速）
  - RSA 加速器将忽略所有 $\widetilde{Y}_i$ ($i > \alpha$) 位。其中，加速位置 α 可通过 RSA_SEARCH_POS_REG 寄存器配置。α 在配置时需小于 N-1，否则相当于没有额外加速；且不建议小于 t，否则无法正确求解 $Z = X^Y \bmod M$。当设置 α 为 t 时，加速效果最佳。此时，$\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ 中的 0 位将在运算中全部被忽略。

  - 注意，这个选项会忽略一些位，进而导致密钥长度变短，使安全性降低。因此，不应使用在安全要求高的应用中。

- CONSTANT_TIME 选项（RSA_CONSTANT_TIME_REG 置 0 开启额外加速）
  - RSA 加速器在运算过程中将简化对 Y 中 0 位的处理。因此，Y 中的 0 越多，加速效果越明显。
  - 注意，这个选项的加速效果与密钥中的 0/1 分布密切相关，这可能会被侧信道攻击 (SCA) 等攻击加以利用，使安全性降低。因此，不应使用在安全要求高的应用中。

为了直观地展示这两个选项带来的额外加速效果，下面通过一个典型实例加以说明。

本实例中：

- $Z = X^Y \bmod M$
- N 等于 3072
- Y 等于 65537
- X 和 M 取任意值
- 当 SEARCH 选项开启时，设定 α 为 16，即 RSA_SEARCH_POS_REG 配置为 16。

表 25.3-1 展示了在进行上述 $Z = X^Y \bmod M$ 运算时，开启不同加速选项组合分别对应的时间开销。

表 25.3-1. 加速效果

SEARCH 选项	CONSTANT_TIME 选项	时间开销（时钟周期）
不加速	不加速	174.7×10^6
加速	不加速	1.023×10^6
不加速	加速	0.546×10^6
加速	加速	0.540×10^6

如图 25.3-1 所示：

- 当两个选项均处于不额外加速状态时，时间开销最大。
- 当两个选项均处于额外加速状态时，时间开销最小。
- 相比于不额外加速状态，任一选项处于额外加速状态时的时间开销明显大幅度降低。

25.4 中断

ESP32-C5 的 RSA 加速器可以生成以下中断信号并发送给中断矩阵。

- RSA_INTR

上述中断信号由 RSA 加速器的内部中断源生成。表 25.4-1 列出了 RSA 加速器的内部中断源、中断触发条件、以及生成的中断信号。

表 25.4-1. RSA 的内部中断源

内部中断源	触发条件	中断信号
RSA_CALC_DONE_INT	一次 RSA 运算结束	RSA_INTR

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

25.5 存储器列表

请注意，这里的地址都是相对于 RSA 加速器基地址的地址偏移量（相对地址），详见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	大小（字节）	起始地址	结束地址	访问
RSA_M_MEM	存储器 M	384	0x0000	0x017F	R/W
RSA_Z_MEM	存储器 Z	384	0x0200	0x037F	R/W
RSA_Y_MEM	存储器 Y	384	0x0400	0x057F	R/W
RSA_X_MEM	存储器 X	384	0x0600	0x077F	R/W

25.6 寄存器列表

本小节的所有地址均为相对于 RSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

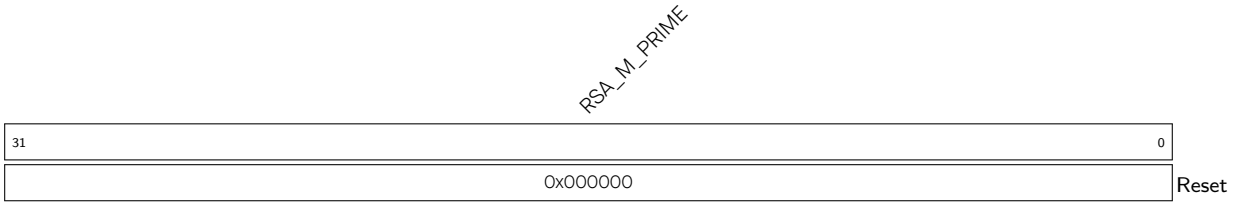
名称	描述	地址	访问
控制/配置寄存器			
RSA_M_PRIME_REG	配置 M'	0x0800	R/W
RSA_MODE_REG	配置 RSA 长度模式	0x0804	R/W
RSA_SET_START_MODEXP_REG	启动模幂运算	0x080C	WT
RSA_SET_START_MODMULT_REG	启动模乘运算	0x0810	WT
RSA_SET_START_MULT_REG	启动乘法运算	0x0814	WT
RSA_QUERY_IDLE_REG	代表 RSA 状态	0x0818	RO
RSA_CONSTANT_TIME_REG	配置 CONSTANT_TIME 选项	0x0820	R/W
RSA_SEARCH_ENABLE_REG	配置 SEARCH 选项	0x0824	R/W
RSA_SEARCH_POS_REG	配置 SEARCH 开始位置	0x0828	R/W
状态寄存器			
RSA_QUERY_CLEAN_REG	表示 RSA 内存初始化状态	0x0808	RO
中断寄存器			
RSA_INT_CLR_REG	RSA 中断清除寄存器	0x081C	WT
RSA_INT_ENA_REG	RSA 中断使能寄存器	0x082C	R/W
版本寄存器			
RSA_DATE_REG	版本控制寄存器	0x0830	R/W

25.7 寄存器

本小节的所有地址均为相对于 RSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

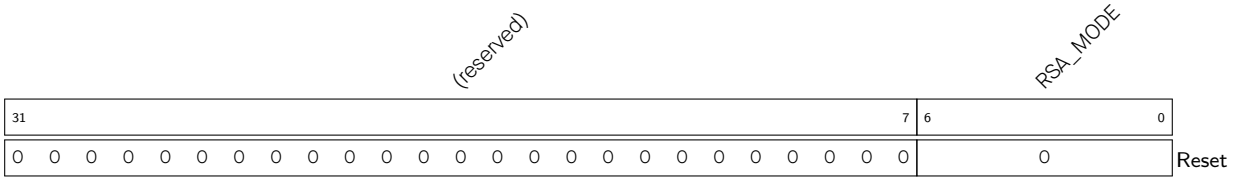
Register 25.1. RSA_M_PRIME_REG (0x0800)



RSA_M_PRIME 配置 M'。

(R/W)

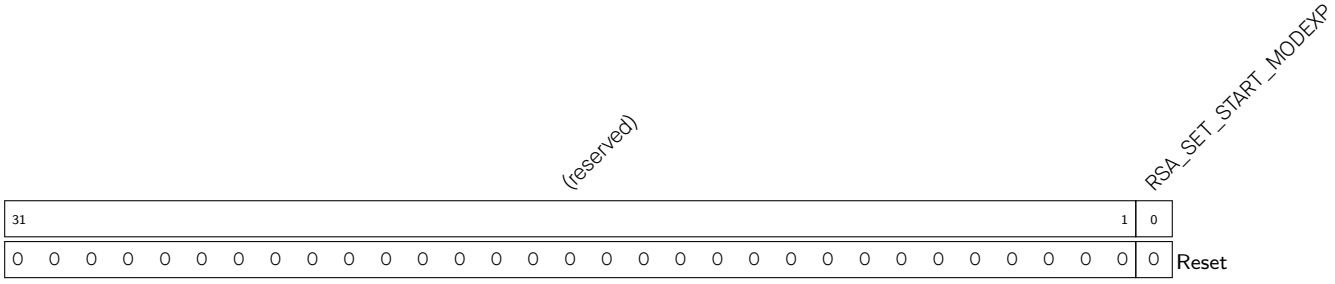
Register 25.2. RSA_MODE_REG (0x0804)



RSA_MODE 配置 RSA 运算子长度。

(R/W)

Register 25.3. RSA_SET_START_MODEXP_REG (0x080C)



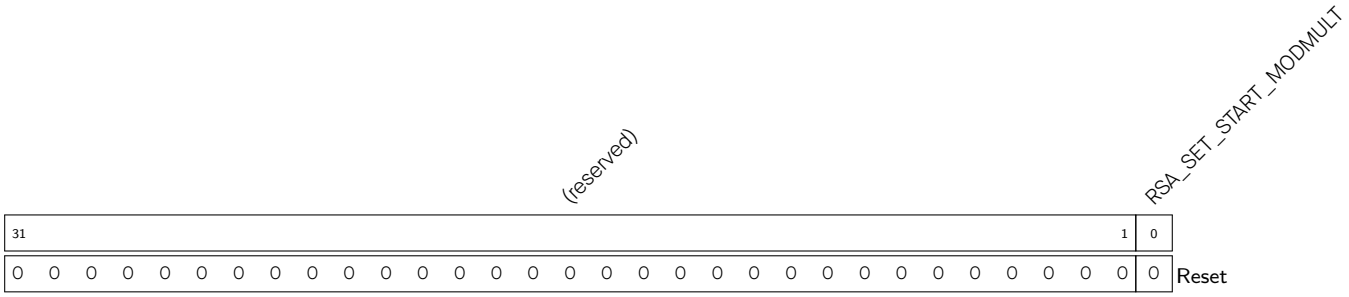
RSA_SET_START_MODEXP 配置是否启动模幂运算。

0：无效果

1：启动

(WT)

Register 25.4. RSA_SET_START_MODMULT_REG (0x0810)



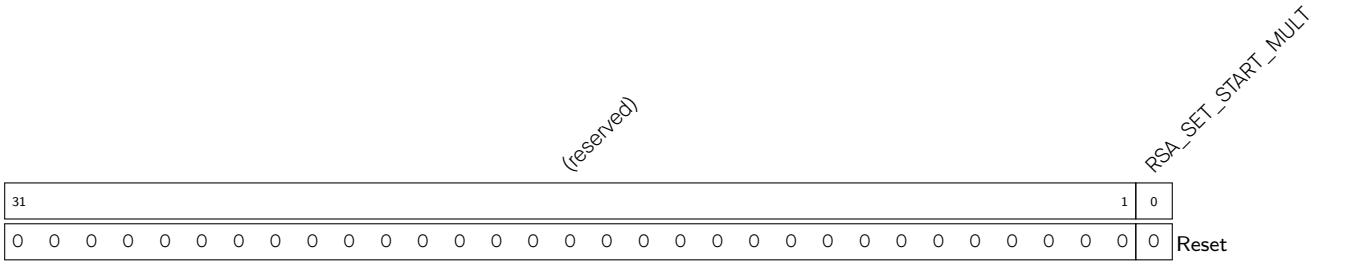
RSA_SET_START_MODMULT 配置是否启动模乘运算。

0：无效果

1：启动

(WT)

Register 25.5. RSA_SET_START_MULT_REG (0x0814)



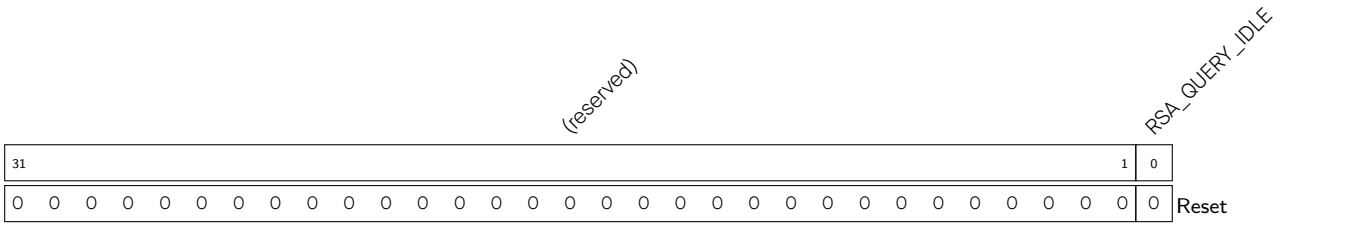
RSA_SET_START_MULT 配置是否启动乘法运算。

0：无效果

1：启动

(WT)

Register 25.6. RSA_QUERY_IDLE_REG (0x0818)



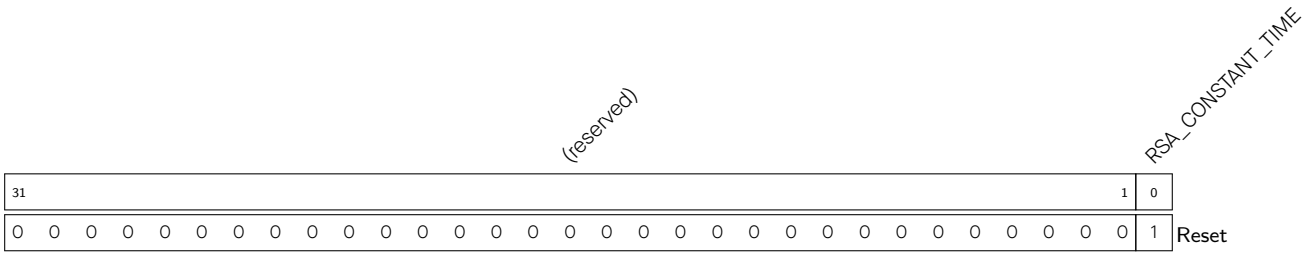
RSA_QUERY_IDLE 代表 RSA 状态。

0：忙碌

1：空闲

(RO)

Register 25.7. RSA_CONSTANT_TIME_REG (0x0820)



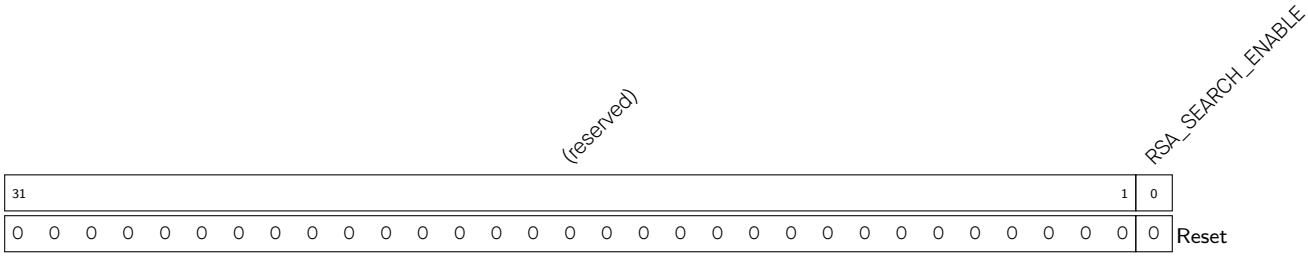
RSA_CONSTANT_TIME 配置 CONSTANT_TIME 选项。

0：加速

1：不加速（默认）

(R/W)

Register 25.8. RSA_SEARCH_ENABLE_REG (0x0824)



RSA_SEARCH_ENABLE 配置 SEARCH 选项。

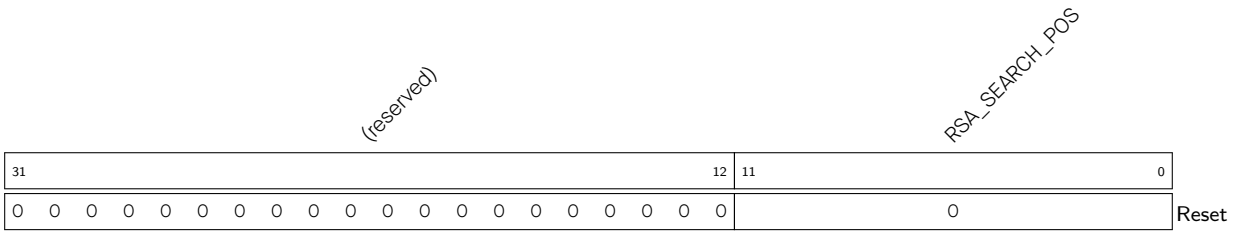
0：不加速（默认）

1：加速

此选项需配合 RSA_SEARCH_POS_REG 使用。

(R/W)

Register 25.9. RSA_SEARCH_POS_REG (0x0828)

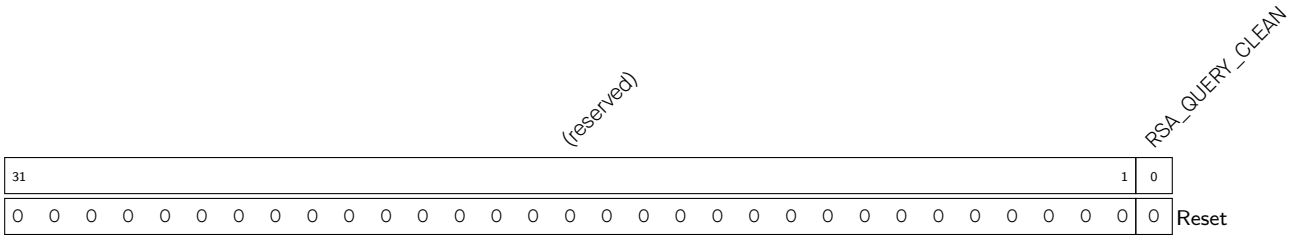


RSA_SEARCH_POS 配置 SEARCH 的开始地址。

此选项需配合 RSA_SEARCH_ENABLE_REG 使用，仅在 RSA_SEARCH_ENABLE 为 1 时有效。

(R/W)

Register 25.10. RSA_QUERY_CLEAN_REG (0x0808)



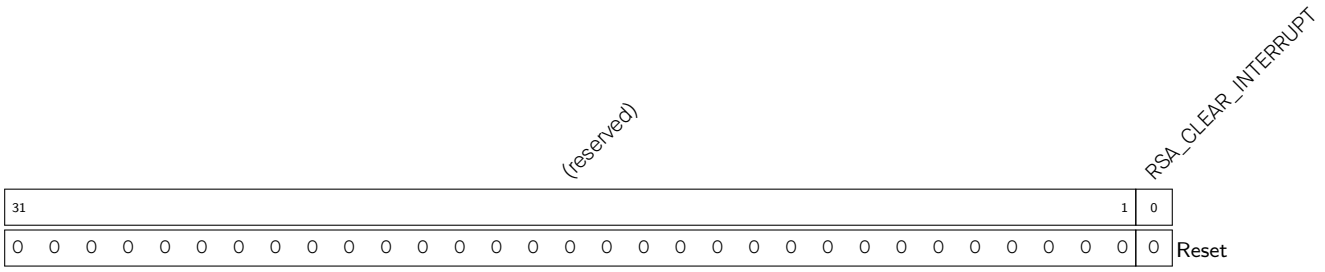
RSA_QUERY_CLEAN 代表 RSA 内存初始化的状态。

0：没有完成

1：已经完成

(RO)

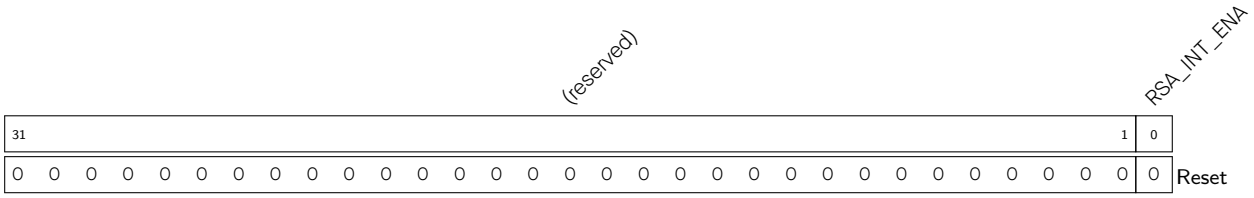
Register 25.11. RSA_INT_CLR_REG (0x081C)



RSA_CLEAR_INTERRUPT 写 1 清除 RSA 中断。

(WT)

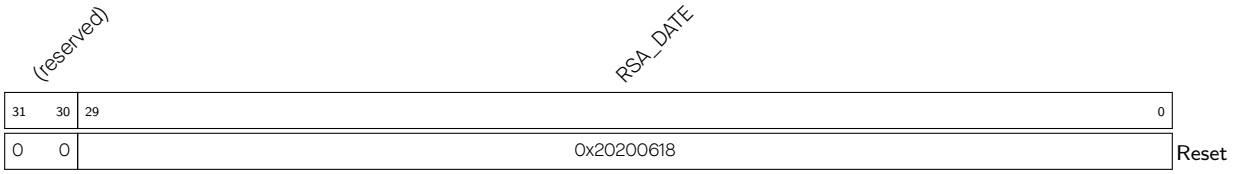
Register 25.12. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA 写 1 使能 RSA 中断。

(R/W)

Register 25.13. RSA_DATE_REG (0x0830)



RSA_DATE 版本控制寄存器。

(R/W)

第 26 章

SHA 加速器 (SHA)

26.1 概述

ESP32-C5 内置 SHA（安全哈希算法）硬件加速器可完成 SHA 运算，具有 Typical SHA 和 DMA-SHA 两种工作模式。整体而言，相比基于纯软件的 SHA 运算，SHA 硬件加速器能够极大地提高运算速度。

26.2 主要特性

ESP32-C5 的 SHA 硬件加速器：

- 支持 FIPS PUB 180-4 规范 中的以下运算标准
  - SHA-1 运算
  - SHA-224 运算
  - SHA-256 运算
- 提供两种工作模式
  - Typical SHA 工作模式
  - DMA-SHA 工作模式
- 允许插入 (interleaved) 功能（仅限 Typical SHA 工作模式）
- 允许中断功能（仅限 DMA-SHA 工作模式）

26.3 工作模式简介

ESP32-C5 内置的 SHA 加速器支持两种工作模式。

- Typical SHA 工作模式：所有数据读写统一通过 CPU 访问完成。
- DMA-SHA 工作模式：所有读数据通过硬件上的 DMA 完成。具体来说，用户可配置 DMA 控制器，由 DMA 控制器提供 SHA 运算过程中所需的数据信息。因此，可以释放 CPU 执行其他任务。

SHA 加速器的激活需要使能 PCR_SHA_CONF_REG 外围时钟的 PCR_SHA_CLK_EN 位，并同时清零 PCR_SHA_RST_EN 位。此外，还需要清零 PCR_DS_RST_EN, PCR_HMAC_RST_EN 和 PCR_ECDSA_RST_EN 复位 RSA 数字签名外设 (RSA_DS)，HMAC 加速器 (HMAC) 和ECDSA 数字签名外设 (ECDSA_DS)。

用户可通过配置 SHA_START_REG 或 SHA_DMA_START_REG 选择 SHA 加速器的工作模式，具体请见表 26.3-1。

表 26.3-1. 工作模式选择

工作模式	选择方式
Typical SHA	SHA_START_REG 置 1
DMA-SHA	SHA_DMA_START_REG 置 1

用户可通过配置 SHA_MODE_REG 寄存器选择 SHA 加速器的运算标准，具体请见表 26.3-2。

表 26.3-2. 运算标准选择

哈希运算标准	SHA_MODE_REG 的配置
SHA-1	0
SHA-224	1
SHA-256	2

注意：

ESP32-C5 的 RSA 数字签名外设 (RSA_DS) 和 HMAC 加速器 (HMAC) 模块在工作时也会调用 SHA 加速器。此时，用户无法正常访问 SHA 加速器。

26.4 功能描述

SHA 加速器可以提取信息摘要 (message digest)，其主要工作流程分为两步：信息预处理 和哈希运算。

26.4.1 信息预处理

信息预处理分为三个主要步骤：附加填充比特、信息解析 和设置初始哈希值。

26.4.1.1 附加填充比特

SHA 加速器仅能处理长度为 512 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件操作将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 m 位，则填充步骤如下：

1. 首先，在待处理信息后填充 1 个 “1”；
2. 随后，再填充 k 个 “0”。其中，k 为满足 $m + 1 + k \equiv 448 \ mod \ 512$ 的最小非负数解；
3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 m 的值。

更多详情，请参考 《FIPS PUB 180-4 规范》 > 章节 “Padding the Message”。

26.4.1.2 信息解析

在完成信息填充后，我们还需将待处理信息（及其填充）解析为 N 个 512 位的信息块，即 $M^{(1)}$、$M^{(2)}$、...、$M^{(N)}$。一个 512 位信息块包括 16 个 32 位的字 (word)，则第 i 个信息块的第一个 32 位字表示为 $M_0^{(i)}$，第二个 32 位字表示为 $M_1^{(i)}$，...，第 16 个 32 位字表示为 $M_{15}^{(i)}$。

SHA 加速器在工作时，每次处理的信息块数据均将按照如下规则写入相应的寄存器中：将 $M_0^{(i)}$ 存放在 SHA_M_0_REG 中，$M_1^{(i)}$ 存放在 SHA_M_1_REG，...，$M_{15}^{(i)}$ 存放在 SHA_M_15_REG 中。

说明：
有关“信息块”及相关概念的描述，请参考 《FIPS PUB 180-4 规范》 > 章节 “Glossary of Terms and Acronyms”。

26.4.1.3 哈希初始值 (Initial Hash Value)

在进行哈希运算前，首先必须设置哈希初始值 $H^{(0)}$，其中 SHA-1、SHA-224 和 SHA-256 运算的哈希初始值为常量 C，且已经固定在硬件中，无需额外配置。

26.4.2 哈希运算流程

在完成信息预处理后，ESP32-C5 SHA 加速器将正式开始哈希运算，最终根据不同运算标准得到不同长度的信息摘要。正如上文所述，ESP32-C5 SHA 加速器支持 Typical SHA 和 DMA-SHA 两种工作模式，下面将对这两种工作模式的具体流程进行介绍。

26.4.2.1 Typical SHA 模式下的运算流程

通常情况下，ESP32-C5 的 SHA 会先处理完当前信息的所有信息块并生成该信息的信息摘要，之后再开始计算新的信息摘要。

不过，ESP32-C5 SHA 加速器还支持插入 (interleaved) 运算（仅 Typical SHA 支持），即在完成当前信息的所有运算前，用户在每个信息块计算完成后均可插入新的运算。

具体来说，用户可以将存储在 SHA_H_*n*_REG 寄存器中的信息摘要暂时保存到其他地方，然后让 SHA 加速器来完成其他优先级更高的运算任务。当插入的运算结束后，用户再将之前暂存的信息摘要重新写入 SHA_H_*n*_REG 中，并继续完成之前中断的计算。

Typical SHA 的具体运算流程

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器，设置运算标准。具体配置，请参考表 26.3-2。
2. 处理当前信息块。
   - 将当前信息块写入 SHA_M_*n*_REG 寄存器。
3. 启动 SHA 加速器 [1]。
   - 如果为首次运算，则对 SHA_START_REG 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器按照步骤 1 中选定的运算标准，使用硬件中固定的哈希初始值进行运算；
   - 如果非首次运算[2]，则对 SHA_CONTINUE_REG 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器使用 SHA_H_*n*_REG 寄存器中的值作为哈希初始值进行运算。
4. 查询当前信息块的处理进度。
   - 轮询寄存器 SHA_BUSY_REG 一直到读回的值为 0，代表 SHA 硬件加速器已完成对当前信息块的计算，进入“空闲”状态[3]。
5. 选择是否有后续的待处理信息块。

- 如果存在后续待处理信息块，则跳回执行步骤 2。
- 否则，继续执行。

6. 获取信息摘要：
   - 从寄存器堆 SHA_H_*n*_REG 取出信息摘要。

说明：

1. 这里，在 SHA 加速器进行硬件运算时，如果存在后续待处理信息块，软件还可以同时将后续信息块写入 SHA_M_*n*_REG 寄存器，以节省时间。
2. 比如重新启动 SHA 加速器完成之前暂停任务的情况。
3. 这里，你可以选择是否需要插入其他任务。如需插入，请前往 插入任务工作流程 具体查看。

如上文所述，ESP32-C5 SHA 加速器**支持在 Typical SHA 模式下“插入”任务**。

具体工作流程如下。

1. 保存插入前任务的以下数据，准备将 SHA 加速器的使用权移交给插入的任务。
   - 读取并保存寄存器 SHA_MODE_REG 中的运算标准类型。
   - 读取并保存寄存器堆 SHA_H_*n*_REG 中的信息摘要。
2. 执行插入的任务。具体按照插入运算类型的不同，请见 Typical SHA 或 DMA-SHA 工作流程。
3. 恢复插入前任务的以下数据，准备将 SHA 加速器的使用权交还给插入前的任务。
   - 将获得使用权前保存的运算标准类型重新写入寄存器 SHA_MODE_REG;
   - 将获得使用权前保存的信息摘要写入寄存器堆 SHA_H_*n*_REG。
4. 将之前任务的下一个待处理信息块写入 SHA_M_*n*_REG 寄存器，并对 SHA_CONTINUE_REG 寄存器置 1，重新启动 SHA 加速器，完成之前暂停的任务。

26.4.2.2 DMA-SHA 模式下的运算流程

ESP32-C5 SHA 加速器在 DMA-SHA 工作模式下不支持插入运算，即用户必须在每次 DMA 运算（可能包括 1 个或多个信息块）全部结束后才能插入新的运算。这种情况下，用户如有插入运算需求，可将较大信息块进行拆分，并进行多次 DMA 运算。每次 DMA 运算之间允许插入其他运算标准的计算任务。单次 DMA 运算最多可以处理 63 个数据块。

与 Typical SHA 不同，SHA 在 DMA-SHA 工作模式下，运算过程中的数据搬运过程均由 DMA 完成。具体配置可见章节 5 *通用 DMA 控制器 (GDMA)*。

DMA-SHA 的具体工作流程

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器，设置运算标准。具体配置，请参考表 26.3-2。
2. 选择是否启用中断。请将 SHA_INT_ENA_REG 寄存器配置为 1 以启动中断。
3. 配置块个数。
   - 将待加密数据的总块数 *M* 写入 SHA_DMA_BLOCK_NUM_REG 寄存器。
4. 开始 DMA-SHA 运算。

- 如果当前 DMA-SHA 运算为接着上一次 DMA-SHA 的运算，需要提前将上一次计算得到的信息摘要写入寄存器堆 SHA_H_*n*_REG 中，随后将 1 写入寄存器 SHA_DMA_CONTINUE_REG；
- 否则，只需要将 1 写入寄存器 SHA_DMA_START_REG。

5. 等待 DMA-SHA 运算结束。判断 DMA-SHA 运算结束有以下两种方法：
   - 轮询寄存器 SHA_BUSY_REG 结果为 0。
   - 等待中断信号产生。此时，应及时通过软件将 SHA_INT_CLEAR_REG 寄存器置为 1 以清除中断。
6. 获取信息摘要
   - 从寄存器堆 SHA_H_*n*_REG 取出信息摘要。

26.4.3 信息摘要存储

哈希运算完成之后，计算得到的信息摘要被 SHA 加速器更新至对应的 SHA_H_*n*_REG (*n*: 0 ~ 7) 寄存器中。不同运算标准得到的信息摘要长度也不同，详情见表 26.4-1：

表 26.4-1. 不同运算标准信息摘要的寄存器占用情况

哈希运算标准	信息摘要长度（位）	寄存器占用情况[1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG

[1] 信息摘要从最高有效位至最低有效位存放，第一个 word 存放在寄存器 SHA_H_0_REG 中，第二个 word 存放在寄存器 SHA_H_1_REG 中，以此类推。

26.4.4 中断

ESP32-C5 的 SHA 加速器可以生成以下中断信号并发送给中断矩阵。

- SHA_INTR

该中断信号由 SHA 加速器的内部中断源生成。表 26.4-2 列出了 SHA 加速器的内部中断源、中断触发条件、以及生成的中断信号。

表 26.4-2. SHA 加速器的内部中断源

内部中断源	触发条件	中断信号
SHA_CALC_DONE_INT	DMA-SHA 模式下完成信息摘要的计算	SHA_INTR

说明：

- 中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。
- 与标准中断寄存器组不同，SHA 的中断寄存器组仅包含 INT_ENA（SHA_INT_ENA_REG）和 INT_CLR（SHA_INT_CLEAR_REG）字段，并不支持用户读取 INT_RAW 和 INT_ST 字段。

26.5 寄存器列表

本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	权限
控制与配置寄存器			
SHA_MODE_REG	配置 SHA 加速器的运算标准	0x0000	R/W
SHA_CONTINUE_REG	继续 SHA 运算（仅用于 Typical SHA 模式）	0x0014	WO
SHA_DMA_START_REG	启动 SHA 加速器的 DMA-SHA 模式	0x001C	WO
SHA_START_REG	启动 SHA 加速器的 Typical SHA 模式	0x0010	WO
SHA_DMA_CONTINUE_REG	继续 SHA 运算（仅用于 DMA-SHA 模式）	0x0020	WO
SHA_DMA_BLOCK_NUM_REG	信息块个数寄存器（仅用于 DMA-SHA 模式）	0x000C	R/W
状态寄存器			
SHA_BUSY_REG	表示 SHA 加速器是否处于“忙碌”状态	0x0018	RO
中断寄存器			
SHA_INT_CLEAR_REG	DMA-SHA 中断清除寄存器	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA 中断使能寄存器	0x0028	R/W
数据寄存器			
SHA_H_0_REG	哈希值	0x0040	R/W
SHA_H_1_REG	哈希值	0x0044	R/W
SHA_H_2_REG	哈希值	0x0048	R/W
SHA_H_3_REG	哈希值	0x004C	R/W
SHA_H_4_REG	哈希值	0x0050	R/W
SHA_H_5_REG	哈希值	0x0054	R/W
SHA_H_6_REG	哈希值	0x0058	R/W
SHA_H_7_REG	哈希值	0x005C	R/W
SHA_M_1_REG	输入信息	0x0084	R/W
SHA_M_2_REG	输入信息	0x0088	R/W
SHA_M_3_REG	输入信息	0x008C	R/W
SHA_M_4_REG	输入信息	0x0090	R/W
SHA_M_5_REG	输入信息	0x0094	R/W
SHA_M_6_REG	输入信息	0x0098	R/W
SHA_M_7_REG	输入信息	0x009C	R/W
SHA_M_8_REG	输入信息	0x00A0	R/W
SHA_M_9_REG	输入信息	0x00A4	R/W
SHA_M_10_REG	输入信息	0x00A8	R/W
SHA_M_11_REG	输入信息	0x00AC	R/W
SHA_M_12_REG	输入信息	0x00B0	R/W
SHA_M_13_REG	输入信息	0x00B4	R/W
SHA_M_14_REG	输入信息	0x00B8	R/W
SHA_M_15_REG	输入信息	0x00BC	R/W
版本寄存器			

名称	描述	地址	权限
SHA_DATE_REG	版本控制寄存器	0x002C	R/W

26.6 寄存器

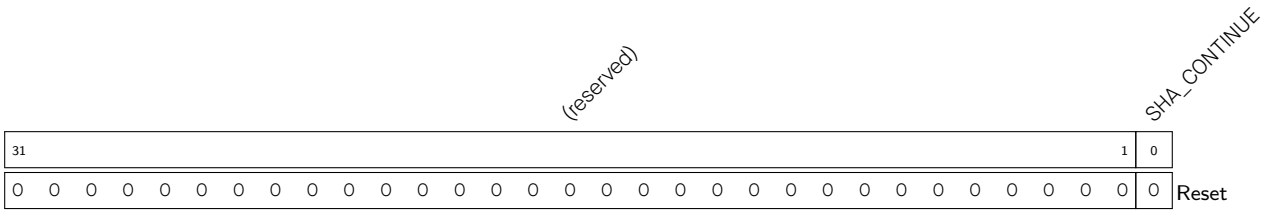
本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 26.1. SHA_START_REG (0x0010)



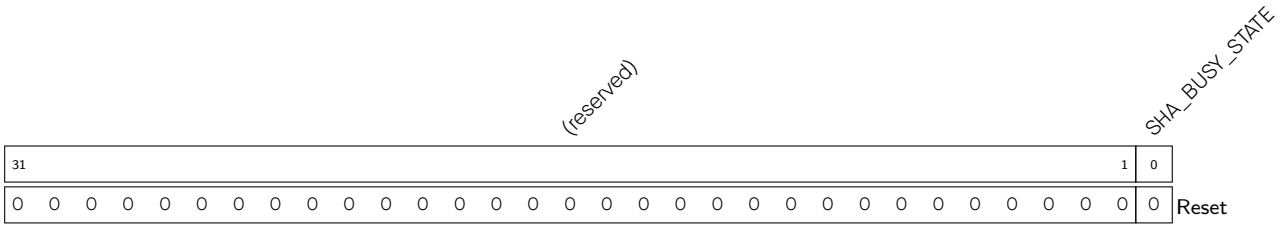
SHA_START 写 1 启动 SHA 加速器的 Typical SHA 模式。(WO)

Register 26.2. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE 写 1 继续 SHA 加速器的 Typical SHA 运算。(WO)

Register 26.3. SHA_BUSY_REG (0x0018)



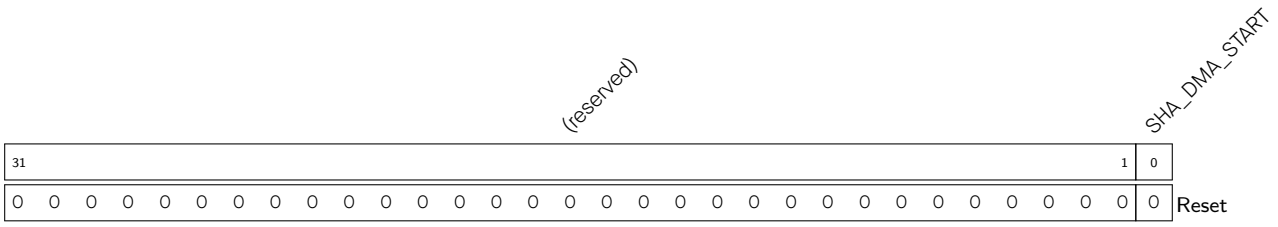
SHA_BUSY_STATE 表示 SHA 是否处于“忙碌”状态。

0：空闲

1：忙碌

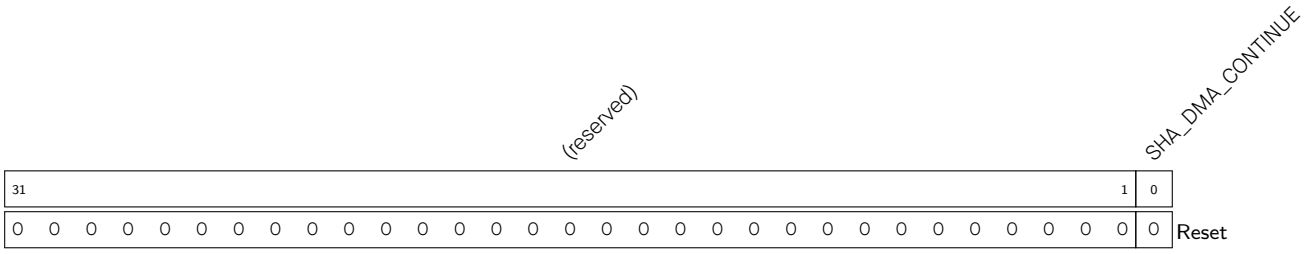
(RO)

Register 26.4. SHA_DMA_START_REG (0x001C)



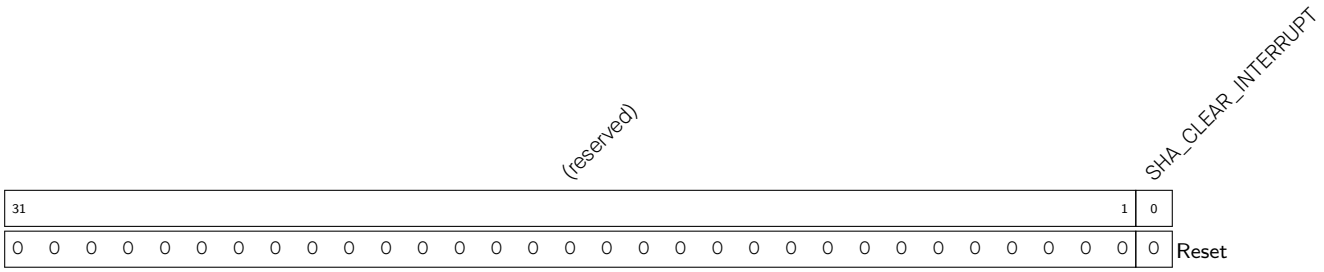
SHA_DMA_START 写 1 启动 SHA 加速器的 DMA-SHA 模式。(WO)

Register 26.5. SHA_DMA_CONTINUE_REG (0x0020)



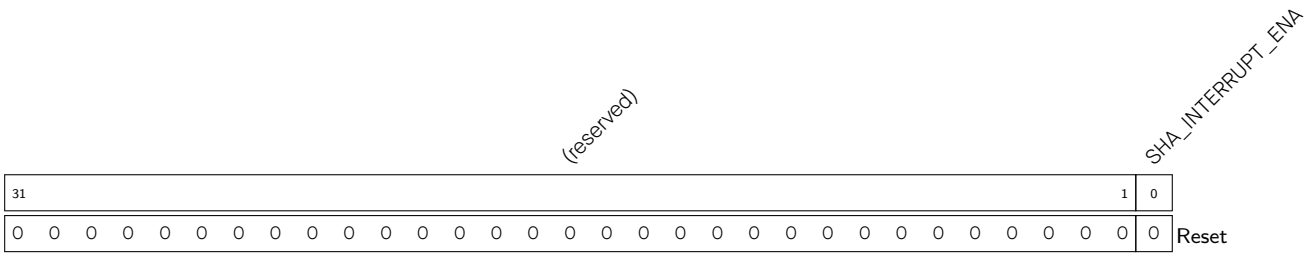
SHA_DMA_CONTINUE 写 1 继续 SHA 加速器的 DMA-SHA 运算。(WO)

Register 26.6. SHA_INT_CLEAR_REG (0x0024)



SHA_CLEAR_INTERRUPT 写 1 清除 DMA-SHA 中断。(WO)

Register 26.7. SHA_INT_ENA_REG (0x0028)



SHA_INTERRUPT_ENA 写 1 使能 DMA-SHA 中断。(R/W)

Register 26.8. SHA_DATE_REG (0x002C)

(reserved)	SHA_DATE	
31 30	29 0	
0 0	0x20201229	Reset

SHA_DATE 版本控制寄存器。(R/W)

Register 26.9. SHA_MODE_REG (0x0000)

(reserved)	SHA_MODE	
31 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

SHA_MODE 配置 SHA 加速器的运算标准。

0：SHA-1

1：SHA-224

2：SHA-256

3 ~ 7: 无效值

(R/W)

Register 26.10. SHA_DMA_BLOCK_NUM_REG (0x000C)

(reserved)	SHA_DMA_BLOCK_NUM	
31 6	5 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

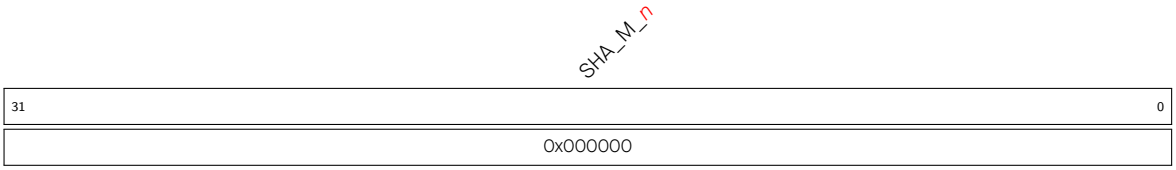
SHA_DMA_BLOCK_NUM 配置 DMA-SHA 工作模式下的信息块个数。(R/W)

Register 26.11. SHA_H_*n*_REG (*n*: 0-7) (0x0040+4**n*)

SHA_H_*n*	
31 0	
0x000000	Reset

SHA_H_*n* 表示第 *n* 个 32 位哈希值。(R/W)

Register 26.12. SHA_M_*n*_REG (*n*: 0-15) (0x0080+4**n*)



SHA_M_*n* 表示第 *n* 个 32 位输入信息。(R/W)

第 27 章

RSA 数字签名外设 (RSA_DS)

27.1 概述

RSA 数字签名外设 (RSA_DS) 使用密码学算法，用于验证消息的真实性和完整性。该算法也可用于向服务器验证设备身份，或验证消息是否经过篡改。

ESP32-C5 包含一个 RSA 数字签名外设 (RSA_DS)，可提供硬件加速，高效生成基于 RSA 的数字签名。HMAC 作为密钥导出函数 (Key Derivation Function, KDF)，使用存储在 eFuse 或密钥管理器生成的密钥作为输入密钥，输出 RSA_DS_KEY。随后，RSA_DS 外设使用 RSA_DS_KEY 解密预先加密的参数，计算出签名。上述过程都发生在硬件层面，因此，在计算过程中，不论是解密 RSA 参数的密钥，还是用于 HMAC 密钥导出函数的输入密钥，都对用户不可见。

27.2 主要特性

- 支持长度最大为 3072 位的 RSA 数字签名密钥
- 支持仅限 RSA_DS 外设读取的加密私钥数据
- 支持 SHA-256 摘要，保护私钥数据免遭攻击者篡改

27.3 功能描述

27.3.1 概述

RSA_DS 外设计算 RSA 签名的方式为 $Z = X^Y \bmod M$，其中 Z 是签名，X 是输入消息，Y 和 M 是 RSA 私钥参数。

私钥参数以密文形式存储在 flash 中。用于解密私钥参数的密钥 (RSA_DS_KEY) 可以来自于 HMAC 模块或密钥管理器模块。当该私钥：

- 由 RSA_DS 外设通过 HMAC 模块计算获取时，生成密钥所需的 HMAC 输入 $HMAC\_KEY$ 存放在 eFuse 中，只允许被 HMAC 模块访问。也就是说，只有 RSA_DS 硬件才能解密私钥密文，软件无法获取私钥明文。关于 eFuse 和 HMAC 的相关细节请参照章节 *7 eFuse 控制器 (EFUSE)* 和章节 *24 HMAC 加速器 (HMAC)*。
- 由 RSA_DS 外设通过密钥管理器模块获取时，该密钥必须以安全的方式由密钥管理器部署而来。

需要签名时，软件直接将输入消息 X 发送到 RSA_DS 外设。RSA 签名计算完成后，软件将读取签名结果 Z。

为方便描述，此处约定几个符号，它们的作用域仅限于本章。

- 1^s: 表示一个完全由 “1” 组成的长度为 s 位的位串。

- $[x]_s$: 一个长度为 s 位的位串，要求 s 为 8 的整数倍。如果 x 是一个数 ($x < 2^s$)，那么其在位串中遵循小端字节序。x 可以是一个变量，例如 $[Y]_{4096}$，或一个十六进制的常数，比如 $[0xOC]_8$。根据需要，$[x]_t$ 右边可以加上$(s-t)$ 个 0，使字串长度扩展成 s 位，得到 $[x]_s$。例如：$[0x05]_8 = 00000101$，$[0x05]_{16} = 0000010100000000$，$[0xOOO5]_{16} = 0000000000000101$，$[0x13]_8 = 00010011$，$[0x13]_{16} = 0001001100000000$，$[0xOO13]_{16} = 0000000000010011$。
- ||: 表示位串粘接操作符，用于将两个位串前后粘成一个较长的位串。

27.3.2 私钥运算子

私钥运算子 Y（私钥指数）和 M（密钥模数）由用户生成。它们具有特定的 RSA 密钥长度（最大为 3072 位）。RSA 签名操作还额外需要两个运算子，即参数 $\bar{r}$ 和 M'，由 Y 和 M 运算得到。

用户加密运算子 Y、M、$\bar{r}$、M' 与验证摘要，并以密文 C 的形式存储。密文 C 输入到 RSA_DS 外设之后，先由硬件解密，然后参与 RSA 签名运算。有关生成 C 的方法，请参考章节 27.3.3。

RSA_DS 外设支持运算子长度为 $N = 32 \times x$ ($x \in \{1, 2, 3, \ldots, 96\}$) 的 RSA 签名运算 $Z = X^Y \bmod M$。其中，参数的位宽需要满足 RSA 计算的运算子长度要求，即 Z、X、Y、M 和 $\bar{r}$ 的位宽必须相同，且为 N 中的任意值，而 M' 的位宽则始终是 32。更多 RSA 计算相关信息，请参考章节 25 *RSA 加速器 (RSA)* 中的 25.3.1 *大数模幂运算* 部分。

27.3.3 软件准备工作

如果用户想使用 RSA_DS 外设进行数字签名，则需要提前做好软件准备工作，如图 27.3-1 所示。图中左半部分列出了在硬件开始 RSA 签名计算之前软件需要做的准备工作，右半部分则列出了硬件在整个签名计算过程中的计算流程。

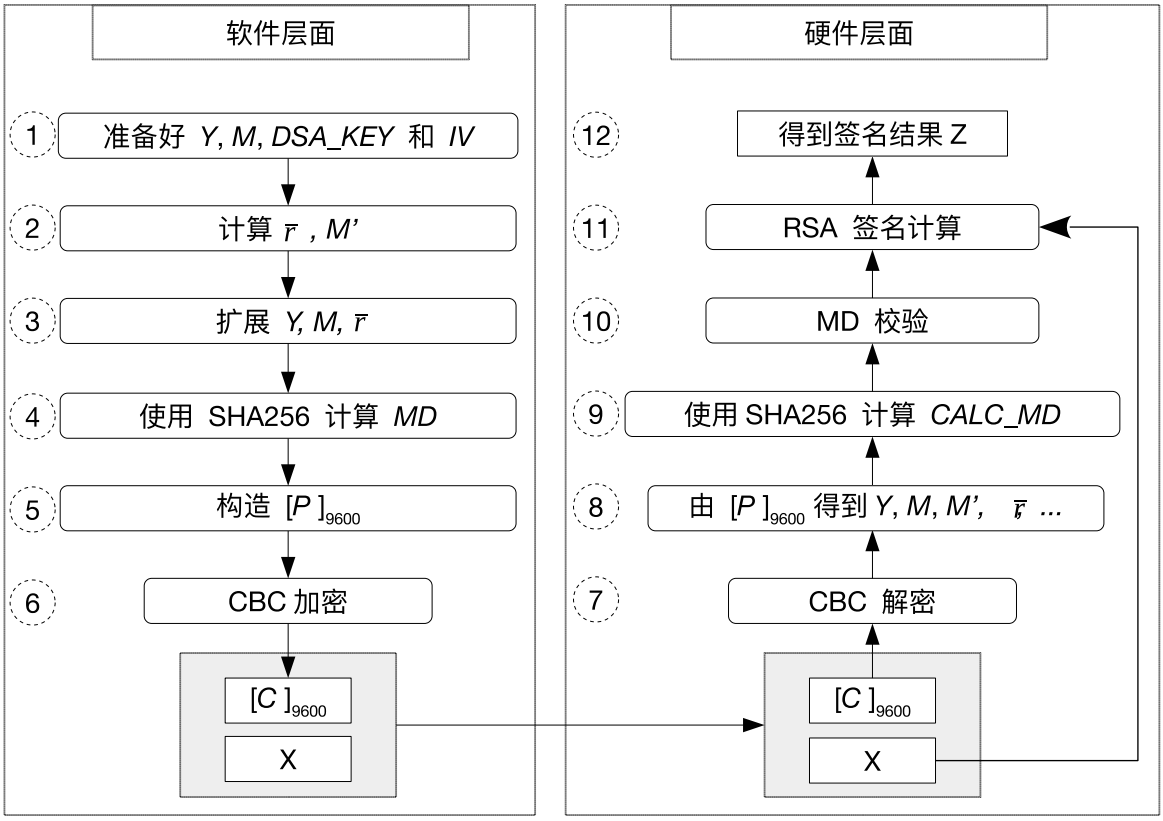


图 27.3-1. 软件准备工作与硬件工作流程

说明：

1. 即便有多次签名计算，也只需进行一次软件准备工作（图 27.3-1 左侧），但每一次签名计算都需要重新进行硬件运算（图 27.3-1 右侧）。
2. 软件准备需要配置时钟复位，相关信息请参考章节 9 复位和时钟。

用户需要依照图 27.3-1 左侧中指定的步骤来计算 C。详细过程如下：

- **步骤 1**：按照章节 27.3.2 所述，准备好符合运算子长度要求的 RSA 私钥运算子 Y 和 M。定义 $[L]_{32} = \frac{N}{32} - 1$（比如，对于 RSA 3072，$[L]_{32}$ == [0x60-1]$_{32}$）。根据私钥的生成方法准备 $[RSA\_DS\_KEY]_{256}$：
  - 当使用 HMAC 作为密钥来源时，准备好 $[HMAC\_KEY]_{256}$，并计算出 $[RSA\_DS\_KEY]_{256}$，即 RSA_DS_KEY = HMAC-SHA256($[HMAC\_KEY]_{256}$, 1^{256})。
  - 当使用密钥管理器作为密钥来源时，准备好 256 位的 RSA_DS_KEY。

  接着，生成一个符合 AES-CBC 块加密算法要求的随机 $[IV]_{128}$。有关 AES 的更多信息，请参考章节 22 AES 加速器 (AES)。
- **步骤 2**：根据 M 求解 $\overline{r}$ 和 M'。
- **步骤 3**：扩展 Y、M 和 $\overline{r}$，得到 $[Y]_{3072}$、$[M]_{3072}$ 和 $[\overline{r}]_{3072}$。此步骤是由于 Y、M 和 $\overline{r}$ 的最大位宽为 3072，故位宽小于 3072 的运算子需要扩展至 3072，而位宽等于 3072 的则不需要扩展。
- **步骤 4**：使用 SHA-256 计算 MD 校验码：$[MD]_{256}$ = SHA256 ($[Y]_{3072}||[M]_{3072}||[\overline{r}]_{3072}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **步骤 5**：构造 $[P]_{9600}$ = ($[Y]_{3072}||[M]_{3072}||[\overline{r}]_{3072}||[Box]_{384}$)。其中，$[Box]_{384}$ = ($[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$)，$[\beta]_{64}$ 是符合 PKCS#7 封装方式的追加码，由 8 个字节码 0x80 组成，即 [0x8080808080808080]$_{64}$，目的在于使 P 的长度为 128 位的整数倍。
- **步骤 6**：计算密文形式的私钥参数 C = $[C]_{9600}$ = AES-CBC-ENC ($[P]_{9600}$, $[RSA\_DS\_KEY]_{256}$, $[IV]_{128}$)，长度为 1200 字节。C 也可以表示为 C = $[C]_{9600}$ = ($[\widehat{Y}]_{3072}||[\widehat{M}]_{3072}||[\widehat{\overline{r}}]_{3072}||[\widehat{Box}]_{384}$)。其中，$[\widehat{Y}]_{3072}$、$[\widehat{M}]_{3072}$、$[\widehat{\overline{r}}]_{3072}$、$[\widehat{Box}]_{384}$ 是 C 的四个子参数，分别对应 $[Y]_{3072}$、$[M]_{3072}$、$[\overline{r}]_{3072}$、$[Box]_{384}$ 的密文形式。

27.3.4 硬件工作流程

每次需要计算 RSA 数字签名时，都会触发硬件操作。硬件需要三个输入信息：预先生成的私钥密文 C、唯一消息 X 以及 IV。

RSA_DS 外设的工作流程可以分为三个阶段：

1. **解析阶段，即图 27.3-1 中的步骤 7 和步骤 8**

   解析过程是图 27.3-1 中步骤 6 的逆过程。RSA_DS 模块将调用 AES 硬件加速器以 CBC 块模式对输入的密文信息 C 进行解密，获取明文信息。该过程可以表示为 P = AES-CBC-DEC (C, RSA_DS_KEY, IV)。其中，IV 就是 $[IV]_{128}$，由用户直接指定；$[RSA\_DS\_KEY]_{256}$ 由：
   - 硬件 HMAC 提供，通过存储在 eFuse 中的 $HMAC\_KEY$ 得到，软件无法获取。
   - 硬件密钥管理器提供时，通过部署在密钥管理器中的 RSA_DS_KEY 得到，软件无法获取。

   通过 P，RSA_DS 外设能够解析出 $[Y]_{3072}$、$[M]_{3072}$、$[\overline{r}]_{3072}$、$[M']_{32}$、$[L]_{32}$、MD 校验码和追加码 $[\beta]_{64}$，这相当于步骤 5 的逆过程。

2. **校验阶段，即图 27.3-1 中的步骤 9 和步骤 10**

   RSA_DS 模块会执行两种校验操作：MD 校验和填充 (padding) 校验。由于填充校验和 MD 校验同步进行，因此填充校验未在图 27.3-1 中体现。

   - MD 校验：RSA_DS 外设调用 SHA-256 进行哈希计算获取哈希值 $[CALC\_MD]_{256}$（即步骤 4）。然后，将 $[CALC\_MD]_{256}$ 与 MD 校验码 $[MD]_{256}$ 作比较，当且仅当二者相同时，MD 校验通过。
   - 填充校验：RSA_DS 外设检查解析阶段解析出的追加码 $[\beta]_{64}$ 是否符合 PKCS#7 标准，当且仅当符合标准时，填充校验通过。

   仅当 MD 校验通过后，RSA_DS 模块才会执行后续计算，否则 RSA_DS 模块将拒绝执行。如果填充校验失败，将生成警告信息，但不会影响 RSA_DS 模块的后续操作。

3. **计算阶段，即图 27.3-1 中的步骤 11 和步骤 12**

   RSA_DS 外设将把用户输入的 X，以及解析得到的 Y、M 和 $\overline{r}$ 都视为大数，结合解析得到的 M'，构成了大数模幂运算 $X^Y \bmod M$ 的所有必要输入参数。大数模幂运算的运算长度由 L 的值唯一指定。RSA_DS 外设调用 RSA 完成大数模幂运算 $Z = X^Y \bmod M$，Z 为签名结果。

27.3.5 软件工作流程

每次需要计算数字签名时，都应执行以下软件操作。输入内容为预先生成的私钥密文 C、唯一的消息 X、IV。本节中的软件步骤会触发章节 27.3.4 中描述的硬件工作流程。
下述流程基于一个假设：软件已经调用了 HMAC 模块，HMAC 已经根据 $HMAC\_KEY$ 计算出了 RSA_DS_KEY。

1. **准备工作**：准备好 C、X、IV。具体方法请参考章节 27.3.3 。
2. **启动 RSA_DS**：对寄存器 DS_SET_START_REG 写 1。
3. **配置 DS_KEY_SOURCE_REG 选择 RSA_DS_KEY 的来源，并检查是否已经准备好**：
   - 配置 DS_KEY_SOURCE_REG 为 1，即选择密钥管理器中部署的 RSA_DS_KEY。
     检查 RSA_DS_KEY 已经成功部署，检查方式为：读取寄存器域 KEYMNG_KEY_DS_VLD，当其值为 1 时，代表 RSA_DS_KEY 已经成功部署并准备好。
   - 配置 DS_KEY_SOURCE_REG 为 0，即选择从 HMAC 模块计算 RSA_DS_KEY。
     轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。

     如果 DS_QUERY_BUSY_REG 超过 1 ms 还没读到 0，则说明 HMAC 初始化出现了问题。此时，软件应当读寄存器 DS_QUERY_KEY_WRONG_REG，根据返回值判断具体是下述哪一种情况：

     - 如果读到零值，说明 HMAC 未被调用。
     - 如果读到非零值 (1 ~ 15)，则说明 HMAC 被调用过，但是 RSA_DS 外设没有成功从 HMAC 模块获取到 RSA_DS_KEY。此种情况可能是因为受到了其他程序的干扰。
4. **将 IV 写入存储器 DS_IV_MEM**：将 IV block 中的内容写入存储器 DS_IV_MEM（大小为 16 字节）。有关 IV block 的更多信息，请参考章节 *22 AES 加速器 (AES)*。
5. **将 X 写入存储器 DS_X_MEM**：将 X_i ($i \in \{0, 1, \ldots, n-1\}$) 写入存储器 DS_X_MEM，容量为 96 个字 (word)，其中 $n = \frac{N}{32}$ 。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。当 X 的长度小于 96 个字时，存储器 DS_X_MEM 中有一部分空间未使用，该部分空间中的数据可以是任意值。
6. **将 C 写入存储器**：即将 C 中的四个子参数分别写入对应的存储器。

- 将 $\widehat{Y_i}$ $(i \in \{0, 1, \ldots, 95\})$ 写入存储器 DS_Y_MEM。
- 将 $\widehat{M_i}$ $(i \in \{0, 1, \ldots, 95\})$ 写入存储器 DS_M_MEM。
- 将 $\widehat{r_i}$ $(i \in \{0, 1, \ldots, 95\})$ 写入存储器 DS_RB_MEM。
- 将 $\widehat{Box_i}$ $(i \in \{0, 1, \ldots, 11\})$ 写入存储器 DS_BOX_MEM。

  其中存储器 DS_Y_MEM、DS_M_MEM、DS_RB_MEM 的容量均为 96 个字，而存储器 DS_BOX_MEM 容量只有 12 个字。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

7. **启动计算**：对寄存器 DS_SET_CONTINUE_REG 写入 1。
8. **等待运算结束**：轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。
9. **检查校验结果**：读寄存器 DS_QUERY_CHECK_REG，根据返回值决定后续操作。
   - 如果返回值为 0，则说明填充校验通过，MD 校验通过，可以继续读取 Z 结果值。
   - 如果返回值为 1，则说明填充校验通过，但 MD 校验失败。Z 结果值无效，跳至步骤 11。
   - 如果返回值为 2，则说明填充校验失败，但 MD 校验通过，用户可以继续读取 Z 结果值。但仍需注意的是，此时数据填充不符合 PKCS#7 封装方式。
   - 如果返回值为 3，则说明填充校验失败，且 MD 校验失败。这种情况说明有致命错误发生。Z 结果值无效。跳至步骤 11。
10. **读出运算结果**：从存储器 DS_Z_MEM 读出运算结果 Z_i $(i \in \{0, 1, \ldots, n-1\})$，其中 $n = \frac{N}{32}$ 。Z 以小端字节序存储在存储器中。
11. **退出计算环境**：对寄存器 DS_SET_FINISH_REG 写入 1，然后轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。

DSA 退出计算环境后，所有输入/输出寄存器和存储器中的数据都已经被抹除（清零）。

27.4 存储器列表

请注意，这里的起始地址和结束地址都是相对于 RSA 数字签名外设 (RSA_DS) 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

名称	描述	大小（字节）	起始地址	结束地址	访问
DS_Y_MEM	存储器 Y	512	0x0000	0x01FF	R/W
DS_M_MEM	存储器 M	512	0x0200	0x03FF	R/W
DS_RB_MEM	存储器 $\bar{r}$	512	0x0400	0x05FF	R/W
DS_BOX_MEM	存储器 Box	48	0x0600	0x062F	R/W
DS_IV_MEM	存储器 IV	16	0x0630	0x063F	R/W
DS_X_MEM	存储器 X	512	0x0800	0x09FF	R/W
DS_Z_MEM	存储器 Box Z	512	0x0A00	0x0BFF	R/W

27.5 寄存器列表

本小节的所有地址均为相对于 RSA 数字签名外设 (RSA_DS) 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

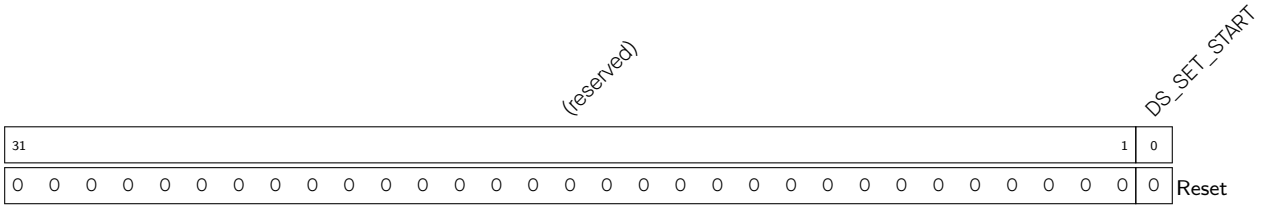
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
状态/控制寄存器			
DS_SET_START_REG	启动 RSA_DS 外设	0x0E00	WT
DS_SET_CONTINUE_REG	继续 RSA_DS 计算	0x0E04	WT
DS_SET_FINISH_REG	结束 RSA_DS 计算	0x0E08	WT
DS_QUERY_BUSY_REG	表示 RSA_DS 外设状态	0x0E0C	RO
DS_QUERY_KEY_WRONG_REG	查询 RSA_DS_KEY 未准备好的原因	0x0E10	RO
DS_QUERY_CHECK_REG	查询 RSA_DS 校验结果	0x0E14	RO
配置寄存器			
DS_KEY_SOURCE_REG	配置 RSA_DS_KEY 来源	0x0E18	R/W
版本控制寄存器			
DS_DATE_REG	版本控制寄存器	0x0E20	R/W

27.6 寄存器

本小节的所有地址均为相对于 RSA 数字签名外设 (RSA_DS) 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 27.1. DS_SET_START_REG (0x0E00)



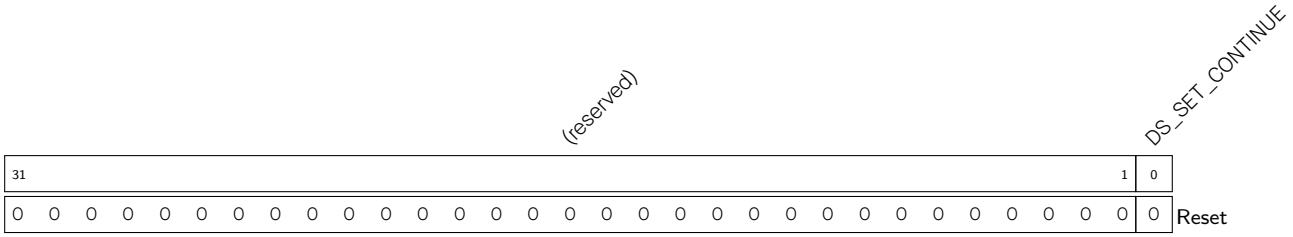
DS_SET_START 配置是否启动 RSA_DS 外设。

0：无效

1：启动 RSA_DS 外设

(WT)

Register 27.2. DS_SET_CONTINUE_REG (0x0E04)



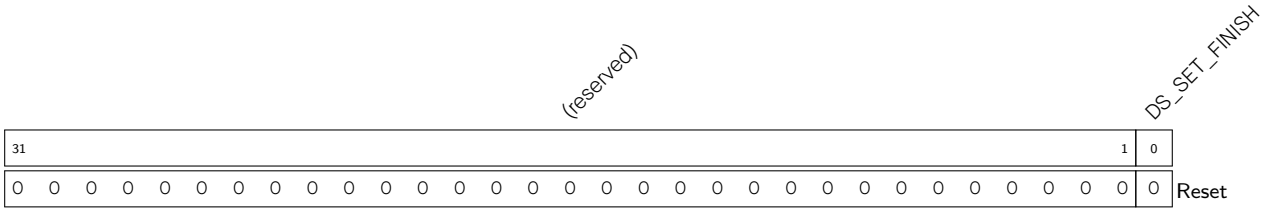
DS_SET_CONTINUE 配置是否继续 RSA_DS 计算。

0：无效

1：继续 RSA_DS 计算

(WT)

Register 27.3. DS_SET_FINISH_REG (0x0E08)



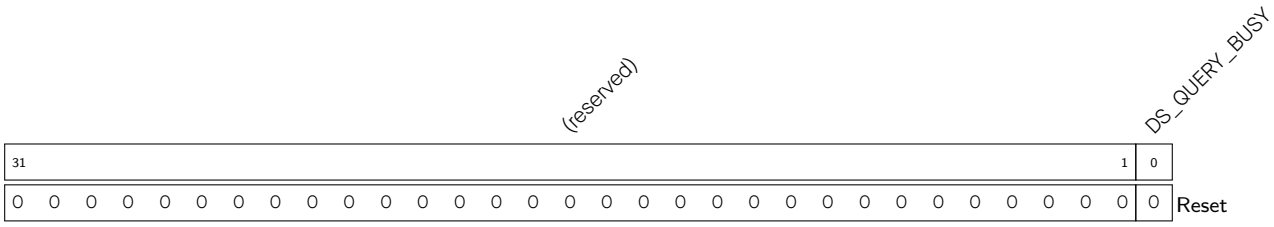
DS_SET_FINISH 配置是否结束 RSA_DS 计算。

0：无效

1：结束 RSA_DS 计算

(WT)

Register 27.4. DS_QUERY_BUSY_REG (0x0E0C)



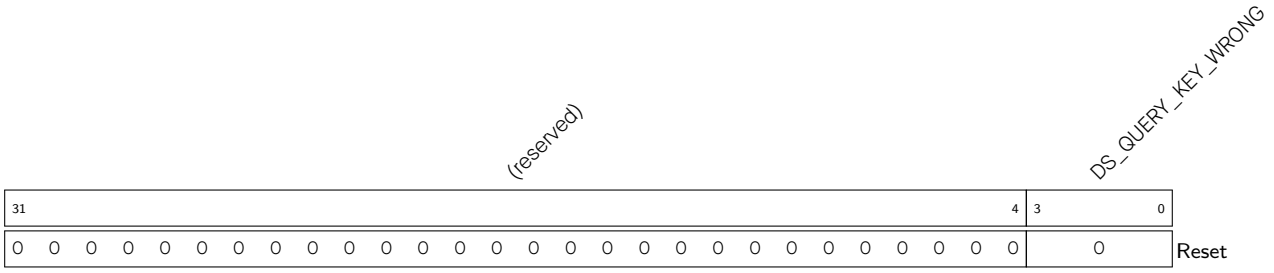
DS_QUERY_BUSY 表示 RSA_DS 外设是否空闲。

0：RSA_DS 外设空闲

1：RSA_DS 外设正忙

(RO)

Register 27.5. DS_QUERY_KEY_WRONG_REG (0x0E10)



DS_QUERY_KEY_WRONG 表示 HMAC 模块的错误情况。

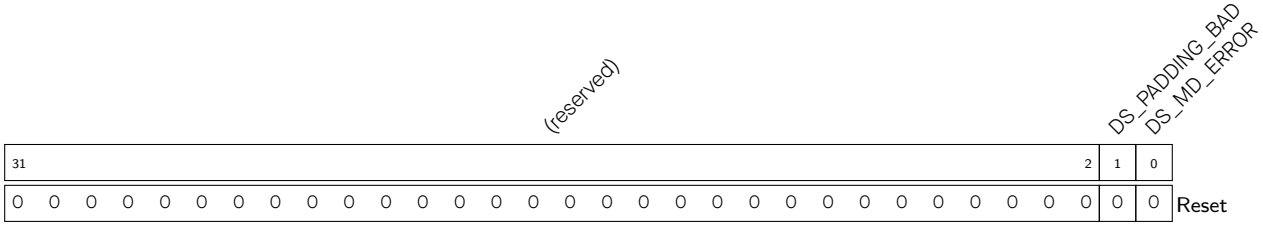
0：HMAC 未被调用

1-15：HMAC 曾被调用，RSA_DS 外设没有成功从 HMAC 模块获取到 RSA_DS_KEY

15 以上：无效

(RO)

Register 27.6. DS_QUERY_CHECK_REG (0x0E14)



DS_MD_ERROR 表示 MD 校验是否通过。

0：MD 校验通过

1：MD 校验失败

(RO)

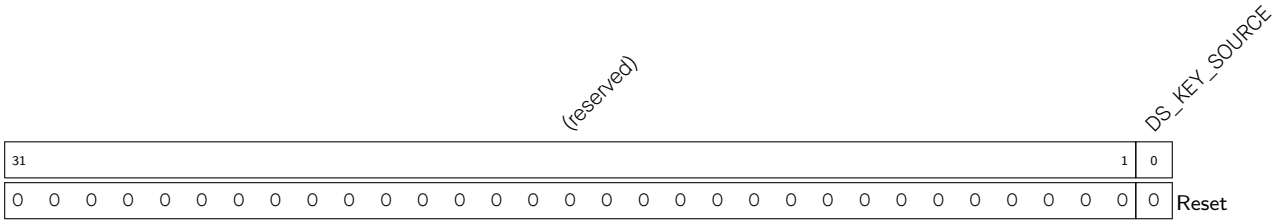
DS_PADDING_BAD 表示填充校验是否通过。

0：填充校验通过

1：填充校验失败

(RO)

Register 27.7. DS_KEY_SOURCE_REG (0x0E18)



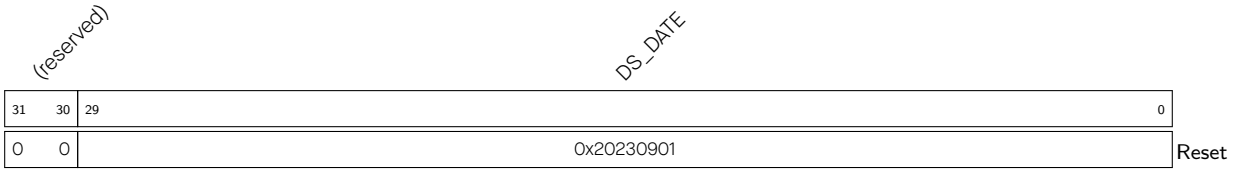
DS_KEY_SOURCE 表示 RSA_DS_KEY 的来源.

0：HMAC

1：密钥管理器

(R/W)

Register 27.8. DS_DATE_REG (0x0E20)



DS_DATE 版本控制寄存器。(R/W)

第 28 章

ECDSA 数字签名外设 (ECDSA_DS)

28.1 概述

在密码学中，椭圆曲线数字签名算法 (ECDSA) 是数字签名算法 (DSA) 基于椭圆曲线密码学的一种变体。

ESP32-C5 的 ECDSA 数字签名外设 (ECDSA_DS) 可在安全高效的环境中计算 ECDSA 签名，不仅可保证签名过程的机密性，还实现了快速计算，非常适合对加密安全性和加密速度有高要求的应用。通过 ECDSA_DS 模块，用户可以在不牺牲性能的情况下保证数据得到保护。

28.2 主要特性

ESP32-C5 的 ECDSA_DS 模块支持：

- 数字签名的生成和验证
- 三种椭圆曲线，分别是 P-192、P-256 和 P-384（具体定义见 FIPS 186-5 规范）
- 多种哈希算法，包括 SHA-224、SHA-256、SHA-384、SHA-512、SHA-512-224 和 SHA-512-256（具体定义见 FIPS PUB 180-4 规范）
- 基于不同工作状态的寄存器访问控制，以确保信息安全

28.3 ECDSA 背景知识

28.3.1 域参数

ECDSA 使用的参数通常被称为域参数，可用于生成密钥、生成签名和验证签名等运算，主要包括有限域中定义的椭圆曲线参数、曲线上的基点和基点阶数等。

ECDSA 使用的域参数主要包括两大类：

- 椭圆曲线域参数：
  - 质数模数 p，指定定义椭圆曲线的有限域的大小。
  - 系数 a 和 b，用于指定椭圆曲线方程。
  - 曲线上的基点 G，用于生成公钥。
  - 基点的阶数 n，代表在椭圆曲线上对基点 G 进行重复点加的次数。
- 哈希函数参数：
  - 哈希算法，用于将要签名的消息生成固定长度的哈希值。
  - 哈希值的长度，决定签名的大小。

这些参数共同定义了 ECDSA 域。

28.3.2 密钥生成

ECDSA 中的密钥生成过程如下所述：

1. 选择一个椭圆曲线域，定义如下参数：
   - 质数模数 p
   - 系数 a 和 b
   - 基点 G 和其阶数 n
2. 生成私钥：
   - 私钥 d 为在 1 到 n-1 之间随机选择的整数。注意，这里必须确保私钥是真正随机的（不可预测或复制）。因此，使用一个安全的随机数发生器 *(RNG)*非常重要。
   - 私钥用于生成签名，需妥善保密保管。
   - 私钥选择完成后，应将其烧录到 ESP32-C5 中的 eFuse 中。(请参考第 7 章 *eFuse 控制器 (EFUSE)*)。
3. 计算公钥：
   - 公钥 Q 的计算方式为 $Q = dG$。
4. 导出公钥：
   - 公钥 Q 通常以 (Qx, Qy) 表示，可公开给他人。

28.3.3 签名生成

ECDSA 签名生成过程如下所述：

1. 选择一条消息 m，用于签名。
2. 计算消息 m 的哈希值 e：e = HASH(m)，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息哈希值 e 的摘要 z：令 z 为 e 的左端 L_n 位，其中 L_n 是基点阶数 n 的位长度。
4. 选择一个随机数 k：范围在 1 到 n-1 之间，其中 n 是椭圆曲线上基点的阶数。
5. 计算签名：签名计算如下：
   (a) 计算点 (x, y) = kG
   (b) 计算 r = x mod n。如果 r 等于零，返回上一步并选择新的 k 值。
   (c) 计算 s = k^{-1} * $(z + d * r)$ mod n。如果 s 等于零，返回步骤 4 并选择新的 k 值。
   (d) 签名为 (r, s)。
6. 发送消息 m 和签名 (r, s) 给接收者。

28.3.4 签名验证

接收方可以使用密钥生成阶段导出的公钥，来验证对应私钥生成的签名。验证过程涉及检查签名是否由对应的私钥生成，以及签名是否对给定的消息有效。

ECDSA 签名验证过程如下所述：

1. 获得公钥 Q，接收消息 m 和签名 (r, s)。
2. 计算消息的哈希值：e 等于 HASH(m)，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息的摘要：令 z 为 e 的左端 L_n 位，其中 L_n 是基点阶数 n 的位长度。
4. 验证签名：签名验证如下所示：
   (a) 验证 r 和 s 是否是介于 1 到 n-1 之间的整数，其中 n 是椭圆曲线上基点的阶数。如果 r 或 s 超出此范围，则签名无效。
   (b) 计算 $u_1 = z * s^{-1} \bmod n$ 和 $u_2 = r * s^{-1} \bmod n$。
   (c) 计算点 $(x_1, y_1) = u_1 * G + u_2 * Q$，其中 G 是椭圆曲线上的基点，Q 是与用于签名生成的私钥相关联的公钥。
   (d) 验证 $r = x_1 \bmod n$。如果 r 不等于 $x_1 \bmod n$，则签名无效。
5. 接受或拒绝签名：如果签名有效，接收方可以确信消息没有被篡改，且消息来自预期的发送方，可以接收。否则，接收方将拒绝并判定为无效消息。

28.4 功能描述

本节描述了 ESP32-C5 的 ECDSA_DS 模块的详细信息。

28.4.1 ECDSA_DS 工作模式

ESP32-C5 内置的 ECDSA_DS 具有三种工作模式，即签名生成、签名验证和公钥导出模式。

用户可以通过配置 ECDSA_WORK_MODE，根据下表 28.4-1 选择 ECDSA_DS 的工作模式。

表 28.4-1. ECDSA_DS 工作模式

ECDSA_WORK_MODE	工作模式
0	签名验证
1	签名生成
2	公钥导出

用户可以通过配置 ECDSA_ECC_CURVE，根据下表 28.4-2 选择 ECDSA_DS 使用的椭圆曲线。

表 28.4-2. ECDSA_DS 椭圆曲线选择

ECDSA_ECC_CURVE	椭圆曲线
0	P-192
1	P-256
2	P-384

用户可以通过配置 ECDSA_SHA_MODE，根据下表 28.4-3 选择用于消息哈希的 SHA 算法。

表 28.4-3. ECDSA_DS SHA 算法

ECDSA_SHA_MODE	SHA 算法
1	SHA-224
2	SHA-256
3	SHA-384
4	SHA-512
5	SHA-512-224
6	SHA-512-256
其他	无效

此外，用户可以通过查询 ECDSA_STATE_REG 寄存器并将返回值与下表 28.4-4 进行比较，来确认 ECDSA_DS 的工作状态。

表 28.4-4. ECDSA_DS 工作状态

ECDSA_STATE_REG	状态	描述
0	IDLE	空闲或已完成的运算，对应于 IDLE 阶段。
1	LOAD	等待用户将信息加载到 ECDSA_DS 中，对应于 LOAD 阶段。
2	GAIN	等待用户从 ECDSA_DS 获取信息，对应于 GAIN 阶段。
3	BUSY	硬件正在运行中，对应于 PREP、PROC 和 POST 阶段。

28.4.2 数据和数据块

ESP32-C5 的 ECDSA_DS 会用到的数据位宽为 192、256、384 或 512 位。

假设一个数据为 $D[255:0]$，则其可以被划分为八个 32 位的数据块 $D[n][31:0](n=0,1,\cdots,7)$，序号数低的数据块对应二进制低位。以 256 位的数据为例：

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

28.4.2.1 数据写入

数据写入即将数据写入进一个内存块的过程，该数据为 ECDSA_DS 的输入数据。具体而言，将数据写入一个 ECDSA_DS 内存块相当于将该数据 $D[n][31:0]$ 依次写入“该内存块起始地址 + $4 \times n$”。

以 256 位的数据为例：

- 将 $D[0]$ 写入“起始地址”
- 将 $D[1]$ 写入“起始地址 + 4”
- $\cdots$
- 将 $D[7]$ 写入“起始地址 + 28”

说明：
进行数据写入操作时，只需要写入数据长度的数据块即可，不需要在数据的高位补 0。例如，在写入 192 位长度的数据时，不需要在数据的高位补 0，只需要存储 192 位的数据。

28.4.2.2 数据读取

数据读取即将一个数据从一个内存块读出的过程，该数据为 ECDSA_DS 的输出数据。具体来说，从一个 ECDSA_DS 内存块读数据相当于从“该内存块起始地址 + $4 \times n$”依次读出 $D[n][31:0]$。以 256 位的数据为例：

- 从“起始地址”读出 $D[0]$
- 从“起始地址 + 4”读出 $D[1]$
- ···
- 从“起始地址 + 28”读出 $D[7]$

说明：
进行数据读取操作时，只需要读取数据长度的数据块即可，例如读取 192 位长度的数据时，只需要读取低 192 位（即 6 个数据块）的数据。

28.4.2.3 消息填充

SHA 加速器仅能处理长度为 512 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件运算将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 L_M 位，则填充步骤如下：

1. 首先，在待处理信息后填充 1 个“1”；
2. 随后，再填充 L_A 个“0”。其中，L_A 为满足 $L_M + 1 + L_A \equiv 448 \ mod \ 512$ 的最小非负数解；
3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 L_M 的值。

更多详情，请参考 《FIPS PUB 180-4 规范》 > 章节 “Padding the Message”。

28.4.2.4 解析消息

在完成信息填充后，我们还需将待处理信息（及其填充）解析为 N 个 512 位的信息块，即 $M^{(1)}$、$M^{(2)}$、...、$M^{(N)}$。

说明：
1. 有关“解析消息”的详细信息，请参考 FIPS PUB 180-4 规范 > 章节 “Parsing the Message”。
2. 有关“信息块”及相关概念的描述，请参考 FIPS PUB 180-4 规范 > 章节 “Glossary of Terms and Acronyms”。

28.4.3 安全功能

为确保运算过程的安全性，ECDSA_DS 模块实现了多种安全功能。

28.4.3.1 抗攻击性能

ESP32-C5 的 ECDSA_DS 模块每次执行操作时，都会利用 ECC 加速器的强抗攻击性能（参考小节 23.4.3 提升抗攻击性能）。这意味着 ECDSA_DS 在每次生成和验证签名时：

- 对于每次操作，执行延迟保持恒定。
- 功耗波动最小化。

这为 ESP32-C5 的 ECDSA_DS 模块提供了强大的抗攻击性能。

28.4.3.2 基于状态的寄存器访问控制

ESP32-C5 的 ECDSA_DS 模块实施了基于状态的寄存器访问控制，可有效防止运行过程中可能发生的非法配置篡改或数据访问，从而保证密钥的安全性。

具体来说，ECDSA_DS 的寄存器在不同工作状态下的访问权限是不同的。例如，ECDSA_CONF_REG 寄存器仅在 ECDSA_DS 处于 IDLE 状态时允许读写运算。这样一来，当 ECDSA_DS 处于其他状态（如 LOAD、GAIN 和 BUSY）时，该寄存器中的配置信息则不允许读取或写入，从而得到很好的保护。有关 ECDSA_DS 所有工作状态的详细信息，请见表 28.4-4。

有关基于状态的 ECDSA_DS 寄存器访问控制的详细信息，请参阅寄存器列表。

28.4.3.3 硬件占用

在运算过程中，ESP32-C5 的 ECDSA_DS 将占用以下硬件模块：

- SHA 加速器
- ECC 加速器

其中，SHA 加速器在触发 ECDSA_SHA_RELEASE_INT 时将释放，而 ECC 加速器在整个 ECDSA_DS 运算中将被占用。

说明：

硬件占用是一种保护复用模块和存储空间的机制。当一个模块被硬件占用时，用户将无法执行以下运算：

- 读取或写入模块的寄存器或存储器数据。
- 关闭模块时钟。
- 复位模块。

在硬件占用结束时，占用的模块将自动复位。此外，当用户对主模块执行软件复位时，所有占用的模块将同时复位。

28.5 配置流程

28.5.1 ECDSA_DS 流程

ECDSA_DS 的完整工作流程包括以下六个阶段。

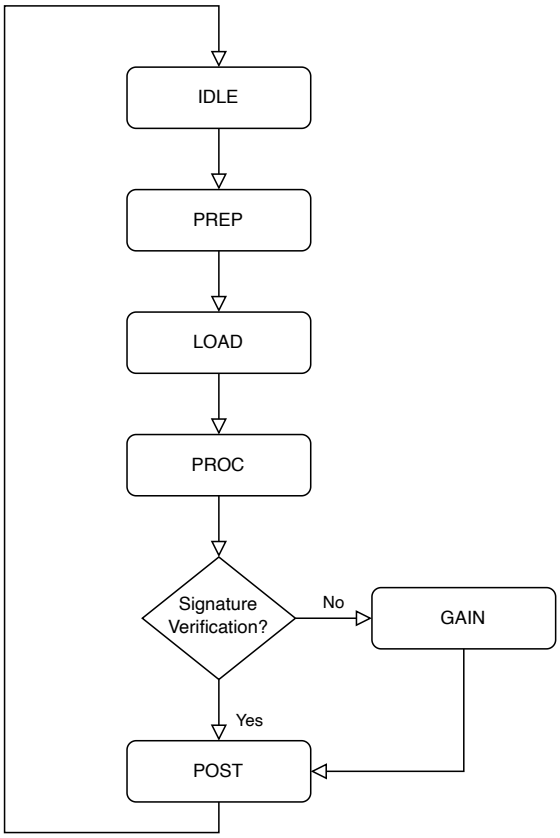


图 28.5-1. ECDSA_DS 流程

每个阶段的详细配置流程在下文中描述。

28.5.1.1 IDLE 阶段

在 IDLE 阶段：

1. 配置静态参数，包括 eFuse 位：
   (a) ECDSA_KEY：ECDSA_DS 中的私钥 d 的值。为了正确配置 eFuse 中的密钥值，用户需要将密钥值写入 KEY*n*（n = 0-5），并将相应的 EFUSE_KEY_PURPOSE_*n* 设置为 ECDSA_KEY。有关更详细的配置步骤，请参阅第 7 章 *eFuse 控制器 (EFUSE)*。
2. 配置 ECDSA_CONF_REG，包括以下字段：
   (a) ECDSA_WORK_MODE：选择 ECDSA_DS 的工作模式。
   (b) ECDSA_ECC_CURVE：选择 ECDSA_DS 的椭圆曲线。
   (c) ECDSA_SOFTWARE_SET_Z：选择是否使用直接输入的 z。
   (d) ECDSA_DETERMINISTIC_K：选择是否使用确定性的 k。
3. 配置寄存器字段 ECDSA_START 以进入 PREP 阶段。

说明：
更多有关“确定性 ECDSA”(Deterministic ECDSA) 的信息，请参考 RFC6979。

28.5.1.2 PREP 阶段

在 PREP 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA_DS 执行准备工作。

通过轮询 ECDSA_BUSY 的状态直到其不为 BUSY，表示 PREP 阶段已结束。然后 ECDSA_DS 将自动进入 LOAD 阶段。

28.5.1.3 LOAD 阶段

在 LOAD 阶段：

1. 使用以下选项之一将输入 z 提供给 ECDSA_DS：
   - 直接输入 z：将 z 写入 ECDSA_MEM_Z。
   - ECDSA_DS SHA 接口：从消息生成 z。有关详细信息，请参阅第 28.5.1.7 节。
2. 根据所选的 ECDSA_WORK_MODE，进行以下配置：
   - 签名验证：
     (a) 将签名 (r, s) 写入 ECDSA_MEM_R 和 ECDSA_MEM_S。
     (b) 将公钥 (Qx, Qy) 写入 ECDSA_MEM_Qx 和 ECDSA_MEM_Qy。
   - 签名生成：
     (a) 不需要其他配置。
   - 公钥导出：
     (a) 不需要其他配置。
3. 将 1 写入 ECDSA_LOAD_DONE，表示配置完成。然后 ECDSA_DS 将自动进入 PROC 阶段。

28.5.1.4 PROC 阶段

在 PROC 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA_DS 根据所选的工作模式执行运算。

通过轮询 ECDSA_BUSY 直到其不为 BUSY，表示 PROC 阶段已结束。然后 ECDSA_DS 将根据所选的工作模式自动进入 GAIN 阶段或 POST 阶段：

- 签名生成：GAIN 阶段
- 签名验证：POST 阶段

28.5.1.5 GAIN 阶段

当选择了签名生成或公钥导出模式时，ECDSA_DS 在 PROC 阶段之后进入 GAIN 阶段：

1. 从 ECDSA_DS 内存中读取数据：
   - 从 ECDSA_MEM_R 和 ECDSA_MEM_S 读取签名 (r, s)，仅在签名生成模式下。
   - 从 ECDSA_MEM_Qx 和 ECDSA_MEM_Qy 读取公钥 (Qx, Qy)，在签名生成或公钥导出模式下。
2. 向 ECDSA_GET_DONE 写入 1，表示 GAIN 阶段已完成。然后 ECDSA_DS 将自动进入 POST 阶段。

28.5.1.6 POST 阶段

在 POST 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA_DS 执行运算的收尾工作。

通过轮询 ECDSA_BUSY 直到不再为 BUSY，表示 POST 阶段已结束。然后 ECDSA_DS 将自动返回 IDLE 阶段。

28.5.1.7 ECDSA_DS SHA 接口

ESP32-C5 的 ECDSA_DS 模块可以自动执行哈希运算并基于直接输入的消息生成 z。

对于消息哈希，ECDSA_DS 支持 SHA 算法 SHA-224（仅当选择 P-192 作为椭圆曲线时有效）和 SHA-256。

要使用 ECDSA_DS SHA 接口，请完成以下步骤：

1. 根据第 28.4.2.3 节中的描述，对消息进行填充。
2. 将消息及其填充解析为消息块。详细信息请参见第 28.4.2.4 节。
3. 处理当前消息块。
   - 将当前消息块写入 ECDSA_MEM_M。
4. 启动 ECDSA_DS SHA 接口 [1]。
   - 如果这是第一次执行此步骤，请将 1 写入 ECDSA_SHA_START 以启动 ECDSA_DS SHA 接口；
   - 如果不是第一次执行此步骤 [2]，请将 1 写入 ECDSA_SHA_CONTINUE 以继续运算。
5. 查询当前信息块的处理进度。
   - 轮询寄存器 ECDSA_SHA_BUSY，直到其为 IDLE，表示接口已完成当前消息块的运算。
6. 处理下一个消息块：
   - 如有下一个消息块，则返回到步骤 3。
   - 如没有更多的消息块，退出。

说明：

1. 在此步骤中，软件还可以在接口启动 SHA 运算时将下一个要处理的消息块（如有）写入寄存器 ECDSA_MEM_M，以节省时间。
2. 即正在继续之前暂停的 ECDSA_DS SHA 接口运算。

28.5.2 时钟

ECDSA_DS 有两种时钟：

- clk_ecdsa：ECDSA_DS 运行所使用的功能时钟
- clk_apb_ecdsa：用于配置 ECDSA_DS 寄存器的总线时钟

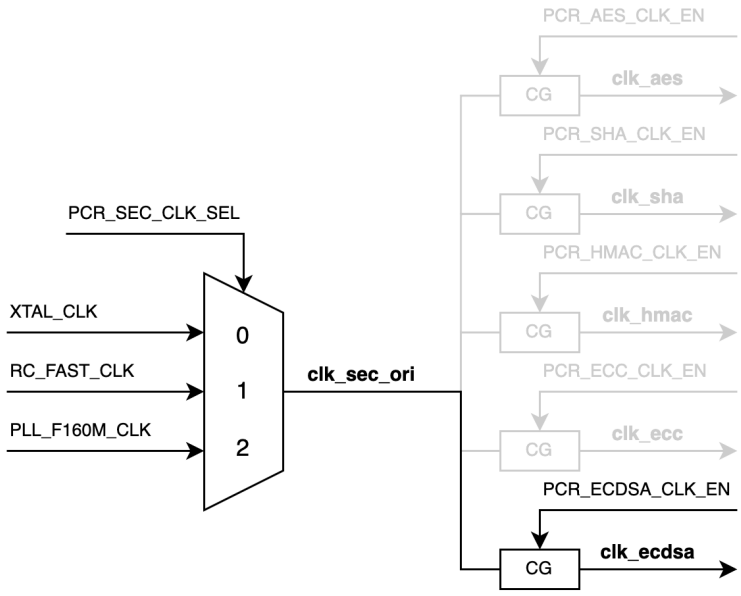


图 28.5-2. clk_ecdsa 时钟图

如图 28.5-2 *clk_ecdsa 时钟图* 所示，安全功能源时钟 clk_sec_ori 有以下时钟源：

- XTAL_CLK
- RC_FAST_CLK
- PLL_F160M_CLK

ECDSA_DS 功能时钟 clk_ecdsa 由时钟门控寄存器 PCR_ECDSA_CLK_EN 控制，当该寄存器值为 1 时，时钟门控开启，使能 clk_ecdsa。

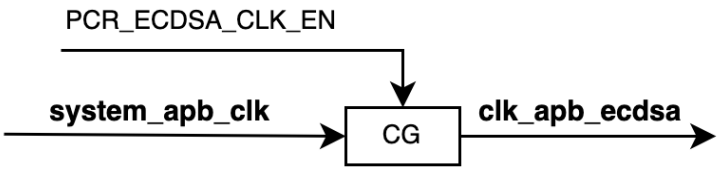


图 28.5-3. clk_apb_ecdsa 时钟图

如图 28.5-3 *clk_apb_ecdsa 时钟图* 所示，ECDSA_DS 配置时钟 clk_apb_ecdsa 的时钟源为 SYSTEM_APB_CLK。

ECDSA_DS 配置时钟 clk_apb_ecdsa 由时钟门控寄存器 PCR_ECDSA_CLK_EN 控制，当该寄存器值为 1 时，时钟门控开启，使能 clk_apb_ecdsa。

有关时钟的更多信息，请查看章节 9 *复位和时钟*。

28.5.3 复位

ECDSA_DS 支持完全复位，向 PCR_ECDSA_RST_EN 先写 1 后写 0 使能，可复位整个 ECDSA_DS 模块。

28.6 中断

ESP32-C5 的 ECDSA_DS 模块可以生成一个中断信号 ECDSA_INTR 并发送给 中断矩阵。

ECDSA_DS 有四个中断源可产生中断信号 ECDSA_INTR，如下所示。每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。

- ECDSA_PREP_DONE_INT：在 PREP 阶段结束时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_PREP_DONE_INT_RAW：存储 ECDSA_PREP_DONE_INT 的原始中断状态。
  - ECDSA_PREP_DONE_INT_ST：代表 ECDSA_PREP_DONE_INT 中断的状态。通过 ECDSA_PREP_DONE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_PREP_DONE_INT_ENA：启用/禁用 ECDSA_PREP_DONE_INT 中断。
  - ECDSA_PREP_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_PREP_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_PREP_DONE_INT_RAW 和 ECDSA_PREP_DONE_INT_ST 将被清除。
- ECDSA_PROC_DONE_INT：在 PROC 阶段结束时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_PROC_DONE_INT_RAW：存储 ECDSA_PROC_DONE_INT 的原始中断状态。
  - ECDSA_PROC_DONE_INT_ST：代表 ECDSA_PROC_DONE_INT 中断的状态。通过 ECDSA_PROC_DONE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_PROC_DONE_INT_ENA：启用/禁用 ECDSA_PROC_DONE_INT 中断。
  - ECDSA_PROC_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_PROC_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_PROC_DONE_INT_RAW 和 ECDSA_PROC_DONE_INT_ST 将被清除。
- ECDSA_POST_DONE_INT：在 POST 阶段结束时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_POST_DONE_INT_RAW：存储 ECDSA_POST_DONE_INT 的原始中断状态。
  - ECDSA_POST_DONE_INT_ST：代表 ECDSA_POST_DONE_INT 中断的状态。通过 ECDSA_POST_DONE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_POST_DONE_INT_ENA：启用/禁用 ECDSA_POST_DONE_INT 中断。
  - ECDSA_POST_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_POST_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_POST_DONE_INT_RAW 和 ECDSA_POST_DONE_INT_ST 将被清除。
- ECDSA_SHA_RELEASE_INT：在 SHA 释放时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_SHA_RELEASE_INT_RAW：存储 ECDSA_SHA_RELEASE_INT 的原始中断状态。
  - ECDSA_SHA_RELEASE_INT_ST：代表 ECDSA_SHA_RELEASE_INT 中断的状态。通过 ECDSA_SHA_RELEASE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_SHA_RELEASE_INT_ENA：启用/禁用 ECDSA_SHA_RELEASE_INT 中断。
  - ECDSA_SHA_RELEASE_INT_CLR：将此位设置为 1 以清除 ECDSA_SHA_RELEASE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_SHA_RELEASE_INT_RAW 和 ECDSA_SHA_RELEASE_INT_ST 将被清除。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

28.7 存储器块

ECDSA_DS 的存储器块用于存储运算的输入数据和输出数据。

表 28.7-1. ECDSA_DS 存储器块

存储器块	大小（字节）	起始地址*	结束地址*	访问权限
ECDSA_MEM_M	128	0x280	0x2FF	R/W
ECDSA_MEM_R	48	0x3E0	0x40F	R/W
ECDSA_MEM_S	48	0x410	0x43F	R/W
ECDSA_MEM_Z	48	0x440	0x46F	R/W
ECDSA_MEM_Qx	48	0x470	0x49F	R/W
ECDSA_MEM_Qy	48	0x4A0	0x4CF	R/W

* 相对于 ECDSA_DS 基地址的地址偏移在第 6 章*系统和存储器* 中的表 6.3-2 中提供。

28.8 寄存器列表

本小节的所有地址均为相对于 ECDSA_DS 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

为了提高安全性，在不同工作状态下，ECDSA_DS 寄存器的读写权限不同。以下是每个工作状态的缩写及其与阶段的对应关系：

- PI：IDLE 状态，对应 IDLE 阶段。
- PL：LOAD 状态，对应 LOAD 阶段。
- PG：GAIN 状态，对应 GAIN 阶段。
- PB：BUSY 状态，对应 PREP、PROC 和 POST 阶段。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

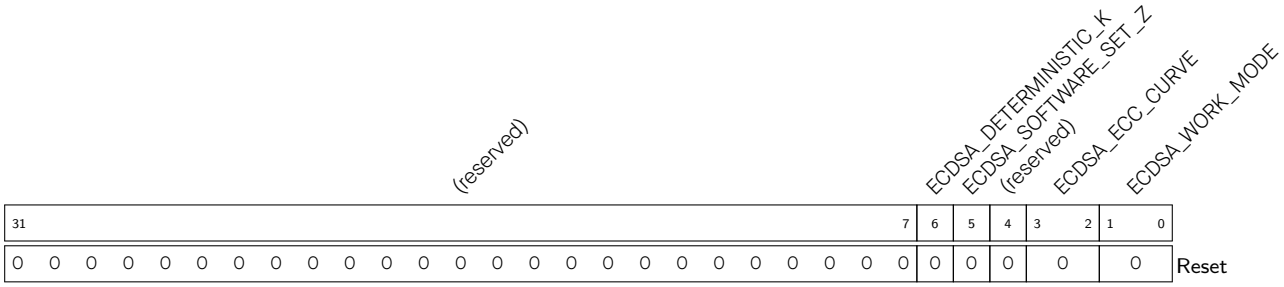
名称	描述	地址	访问权限			
			PI	PL	PG	PB
数据存储器	见表 28.7-1。					
配置寄存器						
ECDSA_CONF_REG	ECDSA_DS 配置寄存器	0x0004	R/W	N/A		
ECDSA_START_REG	ECDSA_DS 启动寄存器	0x001C	WT			N/A
时钟和复位寄存器						
ECDSA_CLK_REG	ECDSA_DS 时钟门控寄存器	0x0008	R/W	N/A		
中断寄存器						
ECDSA_INT_RAW_REG	ECDSA_DS 中断原始状态寄存器	0x000C	RO/WTC/SS			
ECDSA_INT_ST_REG	ECDSA_DS 中断状态寄存器	0x0010	RO			
ECDSA_INT_ENA_REG	ECDSA_DS 中断使能寄存器	0x0014	R/W			
ECDSA_INT_CLR_REG	ECDSA_DS 中断清除寄存器	0x0018	WT			
状态寄存器						
ECDSA_STATE_REG	ECDSA_DS 状态寄存器	0x0020	RO			
结果寄存器						
ECDSA_RESULT_REG	ECDSA_DS 结果寄存器	0x0024	RO/SS			N/A
SHA 寄存器						
ECDSA_SHA_MODE_REG	ECDSA_DS 控制 SHA 寄存器（哈希算法）	0x0200	N/A	R/W	N/A	
ECDSA_SHA_START_REG	ECDSA_DS 控制 SHA 寄存器（运算）	0x0210	N/A	WT	N/A	
ECDSA_SHA_CONTINUE_REG	ECDSA_DS 控制 SHA 寄存器（运算）	0x0214	N/A	WT	N/A	
ECDSA_SHA_BUSY_REG	ECDSA_DS 控制 SHA 状态寄存器	0x0218	N/A	RO	N/A	
版本寄存器						
ECDSA_DATE_REG	版本控制寄存器	0x00FC	R/W	N/A		

28.9 寄存器

本小节的所有地址均为相对于 ECDSA_DS 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 28.1. ECDSA_CONF_REG (0x0004)



ECDSA_WORK_MODE 配置 ECDSA_DS 的工作模式。

0：签名验证模式

1：签名生成模式

2：公钥导出模式

3：无效

(R/W)

ECDSA_ECC_CURVE 配置使用的椭圆曲线。

0：P-192

1：P-256

2：P-384

3：无效

(R/W)

ECDSA_SOFTWARE_SET_Z 配置参数 z 的配置方式。

0：SHA 结果生成

1：软件写入

(R/W)

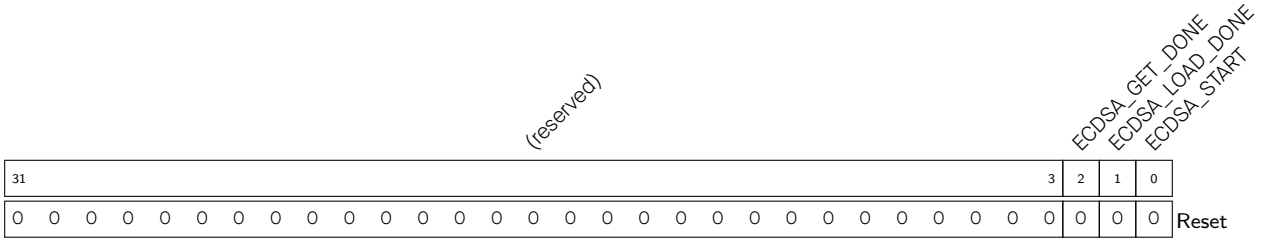
ECDSA_DETERMINISTIC_K 配置参数 k 的配置方式。

0：TRNG 自动生成

1：由确定性推导算法生成

(R/W)

Register 28.2. ECDSA_START_REG (0x001C)



ECDSA_START 配置是否启动 ECDSA_DS 运算。此位配置后自动清 0。

0：无效

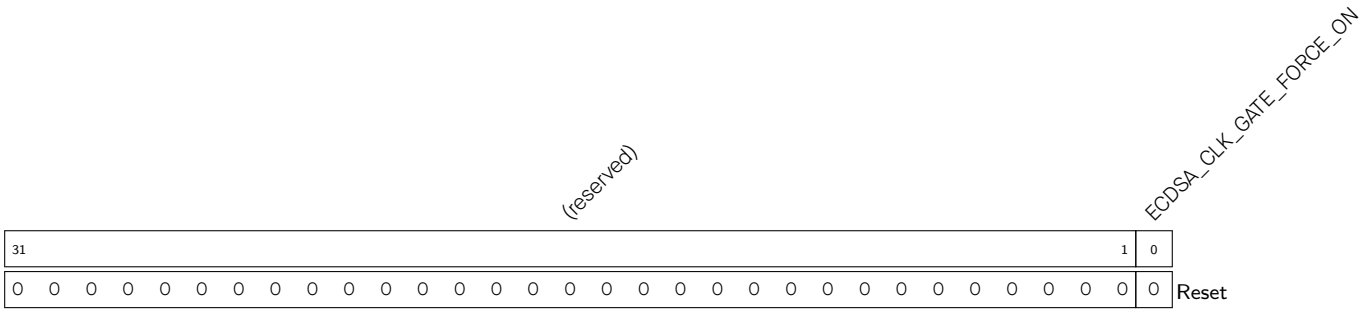
1：启动 ECDSA_DS 运算

(WT)

ECDSA_LOAD_DONE 写 1 生成中断信号，表示 ECDSA_DS 的 LOAD 阶段已完成。此位配置后自动清 0。(WT)

ECDSA_GET_DONE 写 1 生成中断信号，表示 ECDSA_DS 的 GAIN 阶段已完成。此位配置后自动清 0。(WT)

Register 28.3. ECDSA_CLK_REG (0x0008)



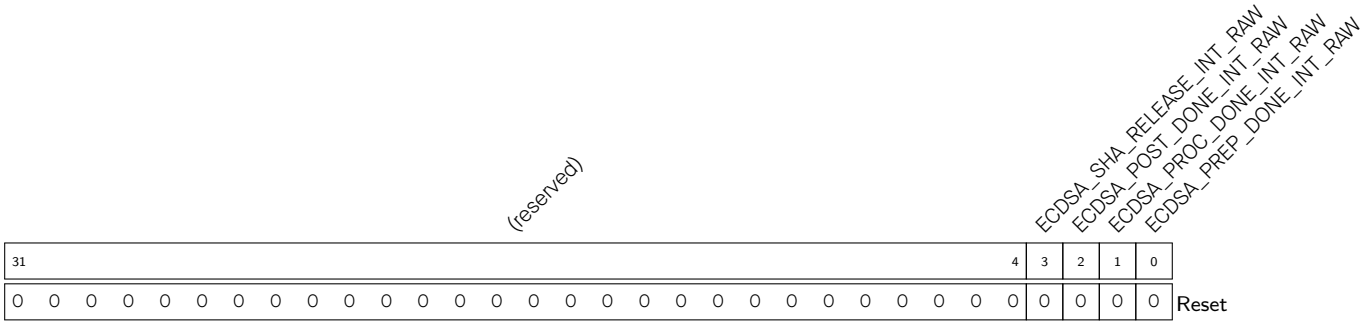
ECDSA_CLK_GATE_FORCE_ON 配置是否强制启动 ECDSA_DS 内存的时钟门控。

0：无效

1：强制启动

(R/W)

Register 28.4. ECDSA_INT_RAW_REG (0x000C)



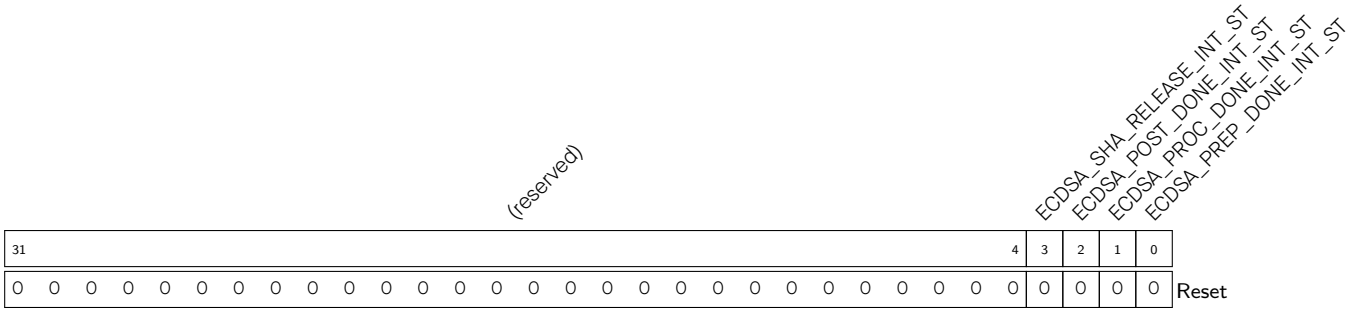
ECDSA_PREP_DONE_INT_RAW ECDSA_PREP_DONE_INT 的原始中断状态。(R/SS/WTC)

ECDSA_PROC_DONE_INT_RAW ECDSA_PROC_DONE_INT 的原始中断状态。(R/SS/WTC)

ECDSA_POST_DONE_INT_RAW ECDSA_POST_DONE_INT 的原始中断状态。(R/SS/WTC)

ECDSA_SHA_RELEASE_INT_RAW ECDSA_SHA_RELEASE_INT 的原始中断状态。(R/SS/WTC)

Register 28.5. ECDSA_INT_ST_REG (0x0010)



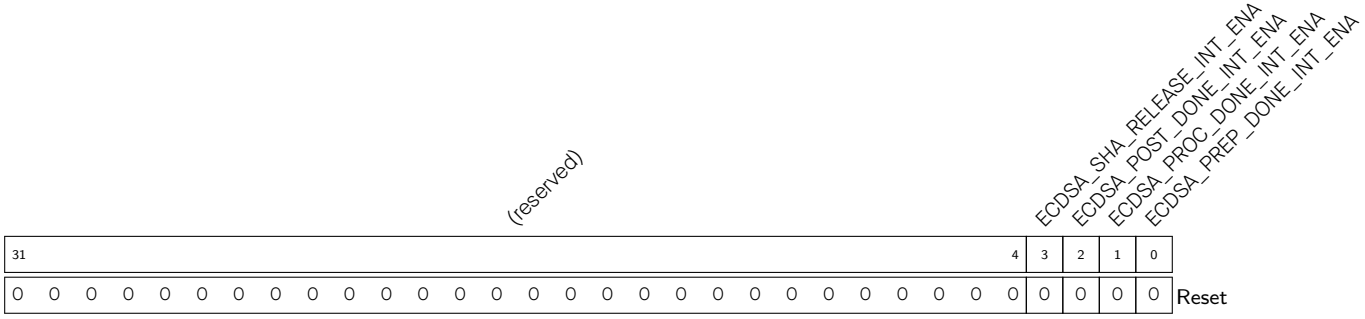
ECDSA_PREP_DONE_INT_ST ECDSA_PREP_DONE_INT 的屏蔽中断状态。(RO)

ECDSA_PROC_DONE_INT_ST ECDSA_PROC_DONE_INT 的屏蔽中断状态。(RO)

ECDSA_POST_DONE_INT_ST ECDSA_POST_DONE_INT 的屏蔽中断状态。(RO)

ECDSA_SHA_RELEASE_INT_ST ECDSA_SHA_RELEASE_INT 的屏蔽中断状态。(RO)

Register 28.6. ECDSA_INT_ENA_REG (0x0014)



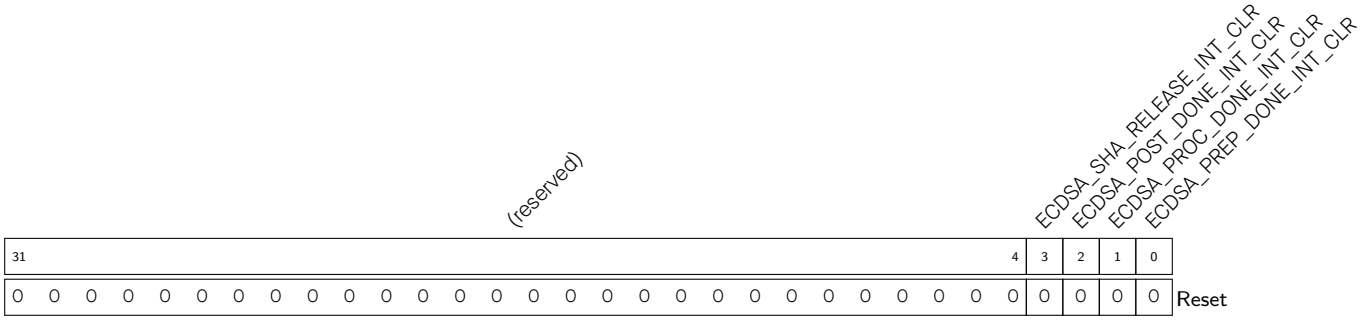
ECDSA_PREP_DONE_INT_ENA 写 1 使能 ECDSA_PREP_DONE_INT 中断。(R/W)

ECDSA_PROC_DONE_INT_ENA 写 1 使能 ECDSA_PROC_DONE_INT 中断。(R/W)

ECDSA_POST_DONE_INT_ENA 写 1 使能 ECDSA_POST_DONE_INT 中断。(R/W)

ECDSA_SHA_RELEASE_INT_ENA 写 1 使能 ECDSA_SHA_RELEASE_INT 中断。(R/W)

Register 28.7. ECDSA_INT_CLR_REG (0x0018)



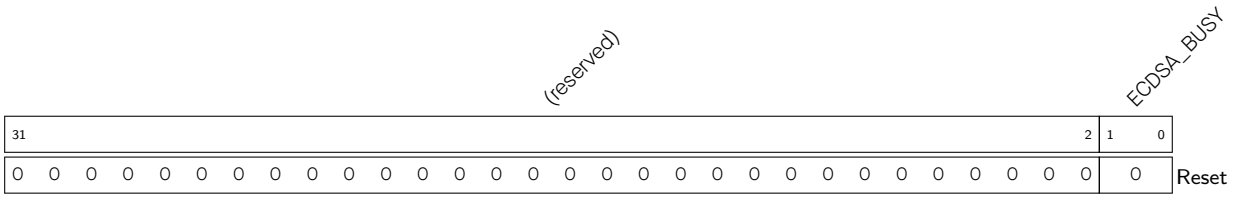
ECDSA_PREP_DONE_INT_CLR 写 1 清除 ECDSA_PREP_DONE_INT 中断。(WT)

ECDSA_PROC_DONE_INT_CLR 写 1 清除 ECDSA_PROC_DONE_INT 中断。(WT)

ECDSA_POST_DONE_INT_CLR 写 1 清除 ECDSA_POST_DONE_INT 中断。(WT)

ECDSA_SHA_RELEASE_INT_CLR 写 1 清除 ECDSA_SHA_RELEASE_INT 中断。(WT)

Register 28.8. ECDSA_STATE_REG (0x0020)



ECDSA_BUSY 表示 ECDSA_DS 的工作状态。

0：IDLE

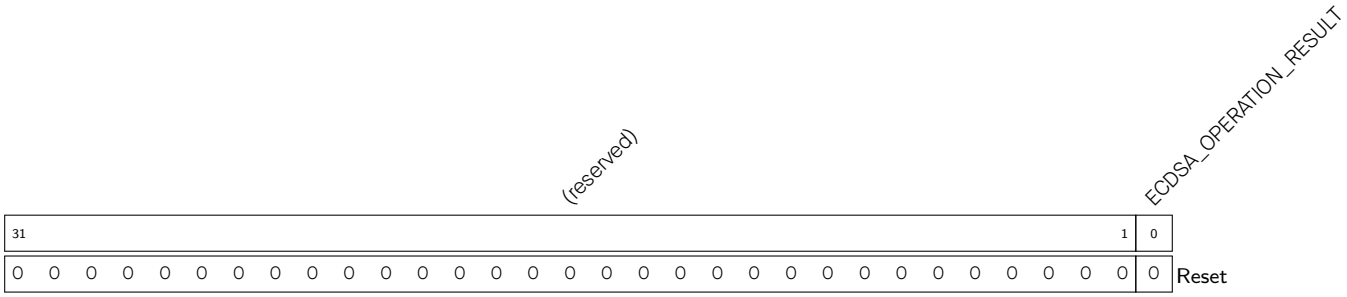
1：LOAD

2：GAIN

3：BUSY

(RO)

Register 28.9. ECDSA_RESULT_REG (0x0024)



ECDSA_OPERATION_RESULT 表示 ECDSA_DS 运算是否成功。

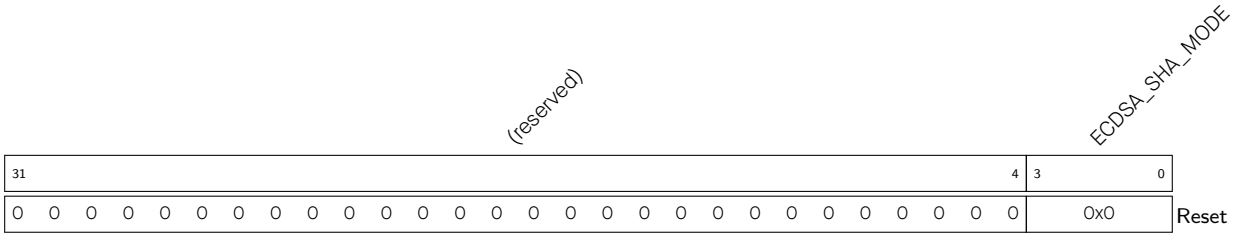
0：失败

1：成功

仅在 ECDSA_DS 运算完成后有效。

(RO/SS)

Register 28.10. ECDSA_SHA_MODE_REG (0x0200)



ECDSA_SHA_MODE 配置哈希运算的 SHA 算法。

1：SHA-224

2：SHA-256

3: SHA-384

4: SHA-512

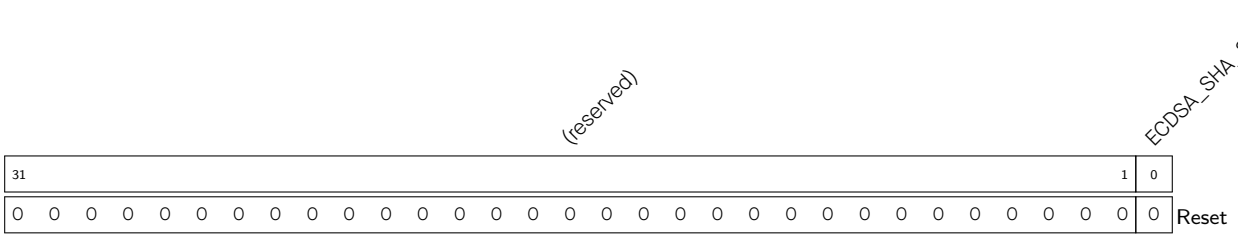
5: SHA-512-224

6: SHA-512-256

其他：无效

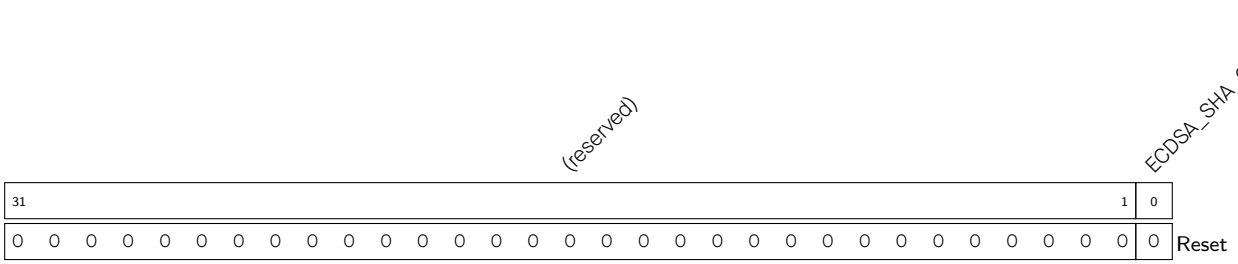
(R/W)

Register 28.11. ECDSA_SHA_START_REG (0x0210)



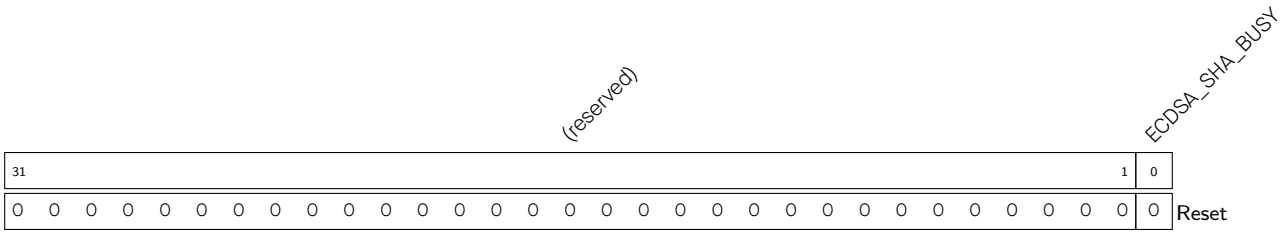
ECDSA_SHA_START 写 1 启动 ECDSA_DS 运算中的首个 SHA 运算。此位配置后自动清 0。(WT)

Register 28.12. ECDSA_SHA_CONTINUE_REG (0x0214)



ECDSA_SHA_CONTINUE 写 1 启动 ECDSA_DS 运算中的后续 SHA 运算。此位配置后自动清 0。(WT)

Register 28.13. ECDSA_SHA_BUSY_REG (0x0218)



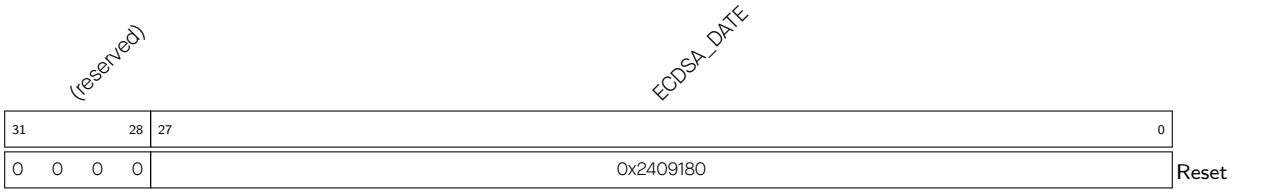
ECDSA_SHA_BUSY 表示 ECDSA_DS 运算过程中 SHA 加速器的工作状态。

0：IDLE

1：BUSY

(RO)

Register 28.14. ECDSA_DATE_REG (0x00FC)



ECDSA_DATE ECDSA_DS 版本控制寄存器。(R/W)

第 29 章

片外存储器加密与解密 (XTS_AES)

29.1 概述

ESP32-C5 芯片集成了片外存储器加密与解密模块，使用 IEEE Std 1619-2007 指定的 XTS-AES 标准算法，为用户存放在片外存储器（flash 和 RAM）的应用代码和数据提供了安全保障。用户可以将专有固件、敏感的用户数据（如用来访问私有网络的证书）存放在片外 flash 中，或将一般数据存放在片外 RAM 中。

29.2 主要特性

- 使用通用 XTS-AES 算法，符合 IEEE Std 1619-2007
- 支持手动加密，需要软件参与
- 支持高速自动加解密，无需软件参与
- 由寄存器配置、eFuse 参数、启动 (boot) 模式共同决定开启/关闭加解密功能
- 支持可配置的抗 DPA 攻击功能

29.3 模块结构

片外存储器加解密模组包含三个模块：手动加密 (Manual Encryption) 模块、自动加密 (Auto Encryption) 模块和自动解密 (Auto Decryption) 模块。模块结构如图 29.3-1 所示。

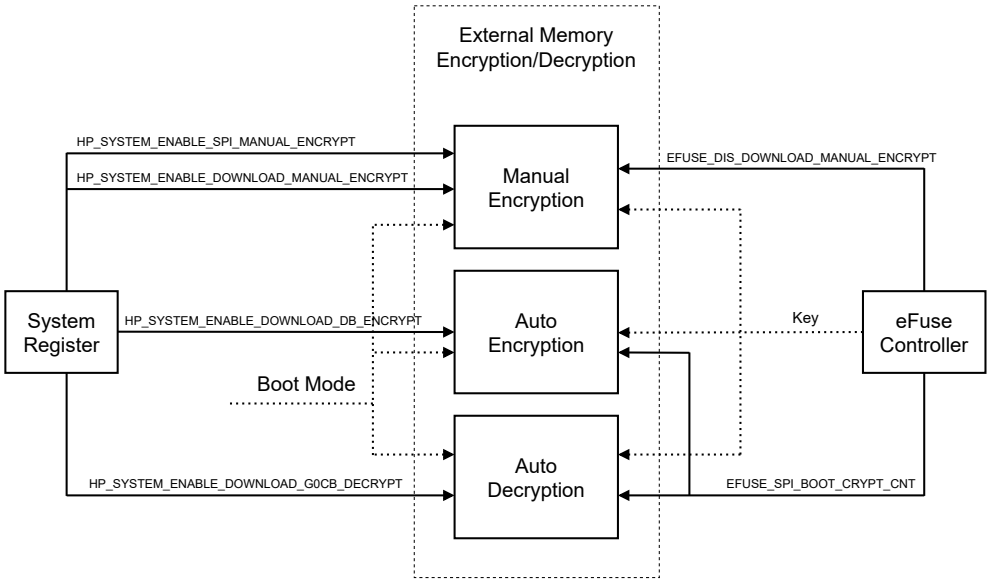


图 29.3-1. 片外存储器加解密结构

手动加密模块能够对指令和数据进行加密，加密后的指令/数据随即以密文状态由 SPI1 写入片外 flash。

当 CPU 通过 cache 向片外 RAM 写入数据时，自动加密模块会首先自动加密数据，然后将数据作为密文写入片外 RAM。

当 CPU 通过高速 cache 从片外 flash 或片外 RAM 读取数据时，自动解密模块将自动解密密文，从而检索指令和数据。

系统寄存器外设中（请参见章节 19 *系统寄存器*），寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 内与片外存储器加解密相关的有以下 3 位：

- HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- HP_SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

此外，片外存储器加解密模组还会从外设 eFuse 控制器中获取两个参数：EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 和 EFUSE_SPI_BOOT_CRYPT_CNT。更多详细信息，请参考章节 7 *eFuse 控制器 (EFUSE)*。

29.4 功能描述

29.4.1 XTS 算法

手动加密模块和自动解密模块均使用 XTS 算法。具体实现中，XTS 算法使用 1024 位为一个数据单元 (data unit)，此处的“数据单元”由 *XTS-AES Tweakable Block Cipher* 标准中的章节 *XTS-AES encryption procedure* 定义。更多关于 XTS-AES 算法的信息，请参考 IEEE Std 1619-2007。

29.4.2 密钥

执行 XTS 算法时，手动加密模块、自动加密模块和自动解密模块共用同一密钥 Key。密钥 Key 来自硬件 eFuse，软件无法访问获取。

Key 的长度为 256 位。Key 的值由 BLOCK4 ~ BLOCK9 中一个 eFuse 块的参数信息决定。为方便阐述如何通过 eFuse 参数信息推导出 Key 的值，现定义：

- Block$_A$：BLOCK4 ~ BLOCK9 中密钥用途为 EFUSE_KEY_PURPOSE_XTS_AES_128_KEY 的 BLOCK（请参考章节 7 eFuse 控制器 (EFUSE) 中的表 7.3-2 密钥用途数值对应的含义）。如果 Block$_A$ 存在，那么 Block$_A$ 中存放着 256 位的 Key_A。

根据 Block$_A$ 是否存在，有两种对应的生成 Key 的方式。任意情况下，Key 值都唯一确定，如表 29.4-1 所示。

表 29.4-1. 根据 Key_A 生成的 Key 值

Block$_A$ 是否存在	Key	Key 长度（位）
是	Key_A	256
否	0^{256}	256

说明：

- “0^{256}” 表示由 256 个位 0 组成的位串。
- 使用 0^{256} 作为密钥 Key 存在风险，建议配置有效密钥。

更多有关密钥用途的信息，请参考章节 7 eFuse 控制器 (EFUSE) 中的表 7.3-2 密钥用途数值对应的含义。

29.4.3 目标空间

目标空间指的是片外存储器中存放首次加密密文的一段连续地址空间。目标空间可由类型、目标大小和目标基地址唯一确定。这三个参数的定义如下：

- 目标类型：目标空间的类型 ($type$)，可为片外 flash 或片外 RAM。值为 0 时代表片外 flash，值为 1 则代表片外 RAM。
- 目标大小：目标空间的大小 ($size$)，即单次加密的数据量。以字节为单位，支持 16 字节、32 字节或 64 字节。
- 目标基地址：目标空间的基地址 ($base\_addr$)。目标基地址为 24 位的物理地址，取值范围为 0x0000_0000 ~ 0x00FF_FFFF，但要求以 $size$ 为单位对齐，即 $base\_addr\%size == 0$。

例如，在一次加密操作中，需要将 16 字节的指令数据加密后写入片外 flash 中的地址段 0x130 ~ 0x13F，则目标空间为 0x130 ~ 0x13F，目标类型为 0（外部 flash），目标大小为 16（字节），目标基地址为 0x130。

对于任意长度（必须是 16 字节的整数倍）的明文指令或数据的加密，整个加密过程可拆分成多次进行，每次加密均具备相应的目标空间和相应参数。

对于自动加解密模块，目标空间等参数由硬件自动调节。对于手动加密模块，目标空间等参数需由用户主动配置。

说明：

IEEE Std 1619-2007 中的章节 *5.1 Data units and tweaks* 中定义的扭曲值是一个 128-bit 的非负整数 ($tweak$)，其值可以通过公式求出：$tweak$ = $type * 2^{30}$ + ($base\_addr$ & 0x3FFFFF80)。$tweak$ 中低 7 位和高 97 位恒为零。

29.4.4 数据写入

对于自动加解密模块，数据写入由硬件自动完成。对于手动加密模块，数据写入需由用户主动配置。手动加密模块中包含一个由 16 个寄存器组成的寄存器块 XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 15)，专用于数据写入，一次可以存放最多 512 位明文指令/数据。

实际上，在手动加密模块中，相对于明文的来源，密文将要存放的位置更为重要。基于明文和密文之间严格的对应关系，为更好地描述明文在寄存器块中的存放方式，现假设明文从一开始就存放在目标空间中，在加密完成后替换为密文。因此，本节中将不再出现“明文”的概念，而用“目标空间”来代替。

目标空间映射到寄存器块的方法：

假设目标空间中某个字的存放地址为 $address$，记 $offset = address\%64$，$n = offset/4$，那么该字将被存放在寄存器 XTS_AES_PLAIN_n_REG 中。

例如，当目标大小为 64 时，寄存器块中的所有寄存器都将被使用，目标空间中的地址与寄存器块之间的映射关系如表 29.4-2 所示。

表 29.4-2. 目标空间与寄存器堆的映射关系

offset	寄存器	*offset*	寄存器
0x00	XTS_AES_PLAIN_0_REG	0x20	XTS_AES_PLAIN_8_REG
0x04	XTS_AES_PLAIN_1_REG	0x24	XTS_AES_PLAIN_9_REG
0x08	XTS_AES_PLAIN_2_REG	0x28	XTS_AES_PLAIN_10_REG
0x0C	XTS_AES_PLAIN_3_REG	0x2C	XTS_AES_PLAIN_11_REG
0x10	XTS_AES_PLAIN_4_REG	0x30	XTS_AES_PLAIN_12_REG
0x14	XTS_AES_PLAIN_5_REG	0x34	XTS_AES_PLAIN_13_REG
0x18	XTS_AES_PLAIN_6_REG	0x38	XTS_AES_PLAIN_14_REG
0x1C	XTS_AES_PLAIN_7_REG	0x3C	XTS_AES_PLAIN_15_REG

29.4.5 手动加密模块

手动加密模块是一个外设模块，自身带有寄存器，可以被 CPU 直接访问。模块内的寄存器、系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，手动加密模块才允许手动加密：

- SPI Boot 模式下：

  当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 位为 1 时，手动加密模块拥有工作权限，否则无法工作。

- Download Boot 模式下：

  当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 位为 1，且 eFuse 参数 EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 为 0 时，手动加密模块拥有工作权限，否则无法工作。

说明：

- 即使 CPU 可以越过 cache，直接读取片外存储器从而得到加密指令/数据，但用户依旧无法获取密钥 Key。

29.4.6 自动加密模块

自动加密模块并非常规外设模块，不带有寄存器，也无法被 CPU 直接访问。由系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，自动加密模块才允许自动加密：

- SPI Boot 模式下：

  当参数 EFUSE_SPI_BOOT_CRYPT_CNT（共 3 位）为 1，2，4 或 7 时（即二进制表示有奇数个 1 时），自动加密模块拥有工作权限，否则无法工作。

- Joint Download Boot 模式下：

  当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT 位为 1 时，自动加密模块拥有工作权限，否则无法工作。

说明：

- 当自动加密模块拥有工作权限时，如果 CPU 向片外 RAM 写入数据，自动加密模块将自动加密该数据，随后将密文写入片外 RAM。加密的整个过程无需软件参与，且对 cache 透明。加密算法过程中，密钥 Key 无法被用户获取。
- 当自动加密模块没有工作权限时，将忽略 CPU 对 cache 的访问请求，不处理数据。此时，数据将以明文形式直接写入片外 RAM。

29.4.7 自动解密模块

自动解密并非传统外设模块，自身不带寄存器，不能被 CPU 直接访问。系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，自动解密模块才允许自动解密：

- SPI Boot 模式下

  当参数 EFUSE_SPI_BOOT_CRYPT_CNT（共 3 位）为 1，2，4 或 7 时（即二进制表示有奇数个 1 时），自动解密模块拥有工作权限，否则无法工作。

- Download Boot 模式下

  当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT 位为 1 时，自动解密模块拥有工作权限，否则无法工作。

说明：

1. 当自动解密模块拥有工作权限时，如果 CPU 通过 cache 读取片外存储器中的指令/数据，自动解密将自动对读取到的密文进行解密以恢复指令/数据。解密的整个过程无需软件参与，且对 cache 透明。解密算法过程中，密钥 Key 无法被软件获取。
2. 当自动解密模块没有工作权限时，无论数据加密与否，自动解密模块不对片外存储器中的数据产生作用。此时，CPU 通过 cache 读取到的是片外存储器中的原始内容。

29.5 软件流程

手动加密模块工作时需要软件参与，软件流程为：

1. 配置 XTS_AES：
   - 将寄存器 XTS_AES_DESTINATION_REG 配置为 0。
   - 将寄存器 XTS_AES_PHYSICAL_ADDRESS_REG 的值设置为 $base\_addr$。
   - 将寄存器 XTS_AES_LINESIZE_REG 的值设置为 $\frac{size}{32}$。

   关于 $base\_addr$ 和 $size$ 的定义，请参考章节 29.4.3。
2. 将明文指令/数据写入至寄存器块 XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 15)。更多详细信息，请参考章节 29.4.4。
   请根据实际需求写入寄存器，未使用的寄存器可为任意值。
3. 等待手动加密模块进入空闲状态。轮询寄存器 XTS_AES_STATE_REG 直到软件读取到 0，表明手动加密模块已进入空闲状态。
4. 向寄存器 XTS_AES_TRIGGER_REG 写入 1，启动手动加密。
5. 等待加密完成。轮询寄存器 XTS_AES_STATE_REG，直到软件读取到 2。
   上述步骤为使用 Key 操作手动加密模块对明文指令/数据进行加密的过程。
6. 向寄存器 XTS_AES_RELEASE_REG 写入 1，使 SPI1 获得密文的访问权限。该操作完成后，寄存器 XTS_AES_STATE_REG 的值将为 3。
7. 调用 SPI1，将密文写入片外 flash（请参阅 ESP-IDF 编程指南 中的 Flash 加密 API 参考）。
8. 向寄存器 XTS_AES_DESTROY_REG 写入 1，销毁密文。该操作完成后，寄存器 XTS_AES_STATE_REG 的值将为 0。

请根据所需加密的明文指令/数据的数量，重复上述步骤。

29.6 抗 DPA 攻击

DPA（Differential Power Analysis, 差分能量分析）是密码学中的一种旁路攻击手段，攻击者可以统计分析从多个加密操作收集的数据，从而计算加密计算中的中间值。ESP32-C5 XTS_AES 模块支持抗 DPA 攻击功能，以抵御来自外界的 DPA 攻击。

根据 IEEE Std 1619-2007，XTS-AES 算法可分为两步：

- 第一步：计算扭曲值。
- 第二步：计算密文/明文。下文中，将此步骤称为“计算数据单元”。

ESP32-C5 允许用户配置在上述两个过程中启用抗 DPA 功能，以增加安全性，详见下文。

29.6.1 时钟抗 DPA 攻击

时钟抗 DPA 攻击是 ESP32-C5 抵御外部 DPA 攻击的首要手段。

用户可以通过寄存器配置实现不同安全等级：

- 首先，为更好地描述各步骤，定义以下参数：

  - $select\_reg$ = XTS_AES_CRYPT_DPA_SELECT_REGISTER
  - $reg\_d\_dpa\_en$ = XTS_AES_CRYPT_CALC_D_DPA_EN
  - $efuse\_dpa\_en$ = EFUSE_CRYPT_DPA_ENABLE
  - $reg\_anti\_dpa\_level$ = XTS_AES_CRYPT_SECURITY_LEVEL
  - $efuse\_anti\_dpa\_level$ = 3

- 配置 XTS_AES 模块抗 DPA 攻击的安全等级：

$$Anti\_DPA\_level = select\_reg\ ?\ (reg\_anti\_dpa\_level) : (efuse\_dpa\_en * efuse\_anti\_dpa\_level)$$

  当 $Anti\_DPA\_level$ 为 0 时，关闭抗 DPA 攻击功能。$Anti\_DPA\_level$ 的值越大，抗 DPA 攻击的能力就越强。

- 配置是否在 XTS-AES 算法计算数据单元时开启抗 DPA 功能：

$$Anti\_DPA\_enabled\_in\_calc\_D = select\_reg\ ?\ reg\_d\_dpa\_en : efuse\_dpa\_en$$

  在 $Anti\_DPA\_level$ 不为 0 的情况下，当 $Anti\_DPA\_enabled\_in\_calc\_D$ 等于 1 时，XTS-AES 算法在计算数据单元时开启抗 DPA 功能。
  在 $Anti\_DPA\_level$ 不为 0 的情况下，XTS-AES 算法计算扭曲值时总是开启抗 DPA 功能。

说明：

- 在 efuse_dpa_en 置 1 后，用户仍可通过配置 $select\_reg = 1$ 及 $reg\_anti\_dpa\_level = 0$ 关闭抗 DPA 功能。
- 配置是否开启抗 DPA 功能会对外存访问带宽造成影响：
  - 在计算数据单元过程中配置开启抗 DPA 功能时，当抗攻击等级 >=4，读写带宽将降低超过 50%。
  - 在计算数据单元过程中配置关闭抗 DPA 功能时，当抗攻击等级 >=6，读写带宽将降低超过 50%。

29.6.2 伪轮次抗 DPA 功能

ESP32-C5 XTS-AES 模块具有伪轮次抗 DPA 功能，可以增加芯片的抵御攻击的性能。当开启伪轮次功能时，XTS-AES 模块将在原始操作轮次之前和之后随机添加伪轮次。在一个伪轮次中，XTS-AES 将使用由硬件自动生成的伪密钥执行伪计算。伪轮次中的计算不会影响原始 XTS-AES 操作的结果。

伪轮次抗 DPA 功能的主要配置参数包括：

伪轮次功能的模式

伪轮次功能的模式 pseudo_mode 可用来选择增加伪轮次计算的具体 XTS-AES 工作步骤：

- 0：计算过程中不增加伪轮次；
- 1：在计算扭曲值时增加伪轮次；
- 2：在计算扭曲值和计算数据单元时的某些轮次增加伪轮次；
- 3：在计算扭曲值和计算数据单元时的所有轮次中增加伪轮次。

伪轮次的总数

在每次 XTS-AES 计算中随机插入伪轮次的具体数量可通过以下配置：

- pseudo_base：该值配置伪轮次功能的基本轮数。

- pseudo_inc：该值配置伪轮次功能的随机递增轮数。

伪轮次的总数将在

$$[pseudo\_base, pseudo\_base + (2^{pseudo\_inc} - 1)]$$

范围内随机生成。

随机数更新频率

pseudo_rnd_cnt 表示伪轮次功能中随机密钥的更新频率。配置的值越大，代表更新频率越高。通常建议将此值配置为最大值 7。

配置

上述参数可通过 eFuse 位或 XTS_AES 相关寄存器完成配置，具体取决于 EFUSE_XTS_DPA_PSEUDO_LEVEL 的值：

- 0：用户可通过 XTS_AES 相关寄存器进行配置
- 1，2 或 3：不允许用户配置

详见表 29.6-1。

表 29.6-1. XTS_AES 伪轮次功能配置

EFUSE_XTS_DPA_PSEUDO_LEVEL	pseudo_mode	pseudo_base	pseudo_inc	pseudo_rnd_cnt
0	XTS_AES_PSEUDO_MODE	XTS_AES_PSEUDO_BASE	XTS_AES_PSEUDO_INC	XTS_AES_PSEUDO_RNG_CNT
1	1	4	2	7
2	2	4	2	7
3	3	4	2	7

29.7 寄存器列表

本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

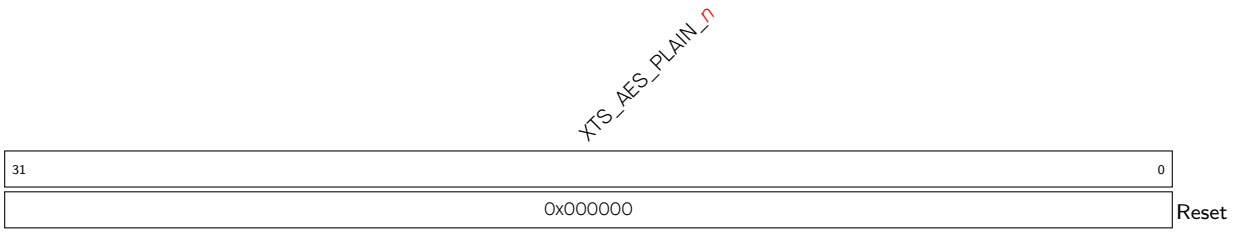
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
明文寄存器堆			
XTS_AES_PLAIN_0_REG	明文寄存器 0	0x0300	R/W
XTS_AES_PLAIN_1_REG	明文寄存器 1	0x0304	R/W
XTS_AES_PLAIN_2_REG	明文寄存器 2	0x0308	R/W
XTS_AES_PLAIN_3_REG	明文寄存器 3	0x030C	R/W
XTS_AES_PLAIN_4_REG	明文寄存器 4	0x0310	R/W
XTS_AES_PLAIN_5_REG	明文寄存器 5	0x0314	R/W
XTS_AES_PLAIN_6_REG	明文寄存器 6	0x0318	R/W
XTS_AES_PLAIN_7_REG	明文寄存器 7	0x031C	R/W
XTS_AES_PLAIN_8_REG	明文寄存器 8	0x0320	R/W
XTS_AES_PLAIN_9_REG	明文寄存器 9	0x0324	R/W
XTS_AES_PLAIN_10_REG	明文寄存器 10	0x0328	R/W
XTS_AES_PLAIN_11_REG	明文寄存器 11	0x032C	R/W
XTS_AES_PLAIN_12_REG	明文寄存器 12	0x0330	R/W
XTS_AES_PLAIN_13_REG	明文寄存器 13	0x0334	R/W
XTS_AES_PLAIN_14_REG	明文寄存器 14	0x0338	R/W
XTS_AES_PLAIN_15_REG	明文寄存器 15	0x033C	R/W
配置寄存器			
XTS_AES_LINESIZE_REG	配置目标空间的大小	0x0340	R/W
XTS_AES_DESTINATION_REG	配置片外存储器的类型	0x0344	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	配置物理地址	0x0348	R/W
XTS_AES_DPA_CTRL_REG	配置抗 DPA 功能	0x0388	R/W
XTS_AES_PSEUDO_ROUND_CONF_REG	配置 XTS-AES 伪轮次功能	0x038C	R/W
控制/状态寄存器			
XTS_AES_TRIGGER_REG	启动 AES 算法	0x034C	WT
XTS_AES_RELEASE_REG	释放控制寄存器	0x0350	WT
XTS_AES_DESTROY_REG	销毁控制寄存器	0x0354	WT
XTS_AES_STATE_REG	状态寄存器	0x0358	RO
版本控制寄存器			
XTS_AES_DATE_REG	版本控制寄存器	0x035C	R/W

29.8 寄存器

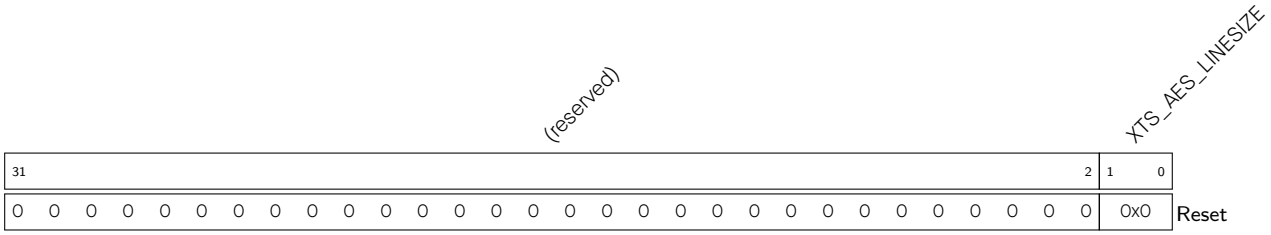
本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 29.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0300+4**n*)



XTS_AES_PLAIN_*n* 配置明文的第 *n* 个 32 位部分。(R/W)

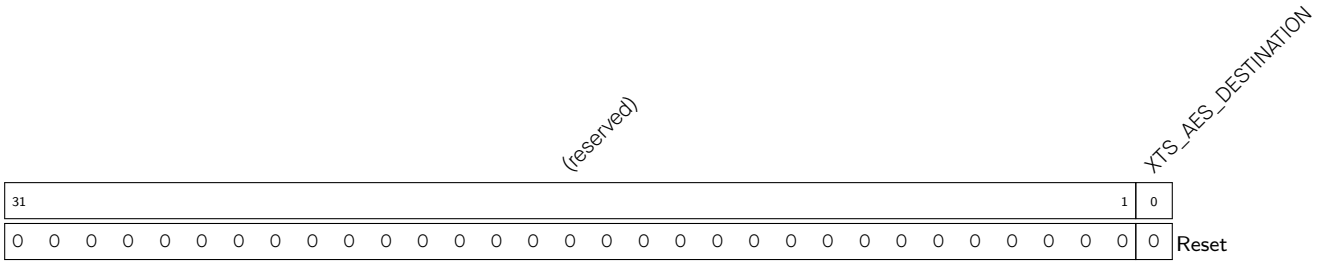
Register 29.2. XTS_AES_LINESIZE_REG (0x0340)



XTS_AES_LINESIZE 配置单次加密的数据大小。

0：加密 16 字节
1：加密 32 字节
2：加密 64 字节
3：无效
(R/W)

Register 29.3. XTS_AES_DESTINATION_REG (0x0344)



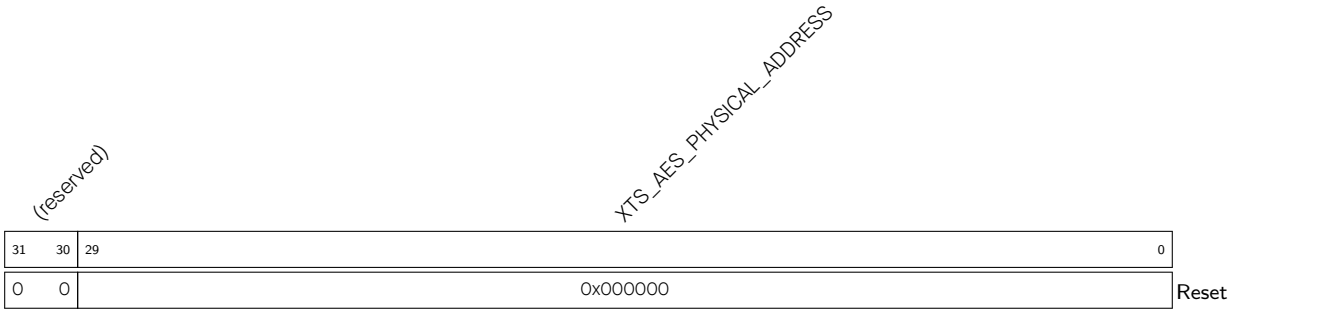
XTS_AES_DESTINATION 配置手动加密的存储器类型。目前只能手动加密 flash，所以只能为 0。用户不能写入 1，否则将发生错误。

0：加密 flash

1：加密片外 RAM

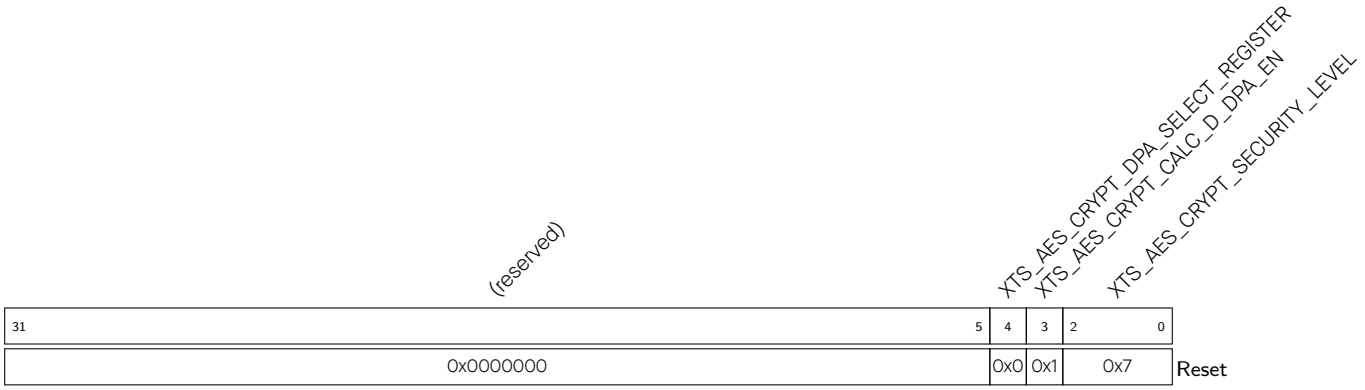
(R/W)

Register 29.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0348)



XTS_AES_PHYSICAL_ADDRESS 配置手动加密使用的物理地址。注意，该值必须按照 XTS_AES_LINESIZE 定义的大小对齐。(R/W)

Register 29.5. XTS_AES_DPA_CTRL_REG (0x0388)



XTS_AES_CRYPT_DPA_SELECT_REGISTER 配置由 eFuse 或寄存器控制时钟抗 DPA 功能。

0：由寄存器控制时钟抗 DPA 功能

1：由 eFuse 控制时钟抗 DPA 功能

(R/W)

XTS_AES_CRYPT_CALC_D_DPA_EN 配置是否在 XTS_AES 算法中启用时钟抗 DPA 功能。

0：仅在计算扭曲值时启用时钟抗 DPA 功能

1：计算扭曲值或是计算数据单元时都启用时钟抗 DPA 功能

请注意，此字段仅在 XTS_AES_CRYPT_SECURITY_LEVEL 不为 0 时有效。

(R/W)

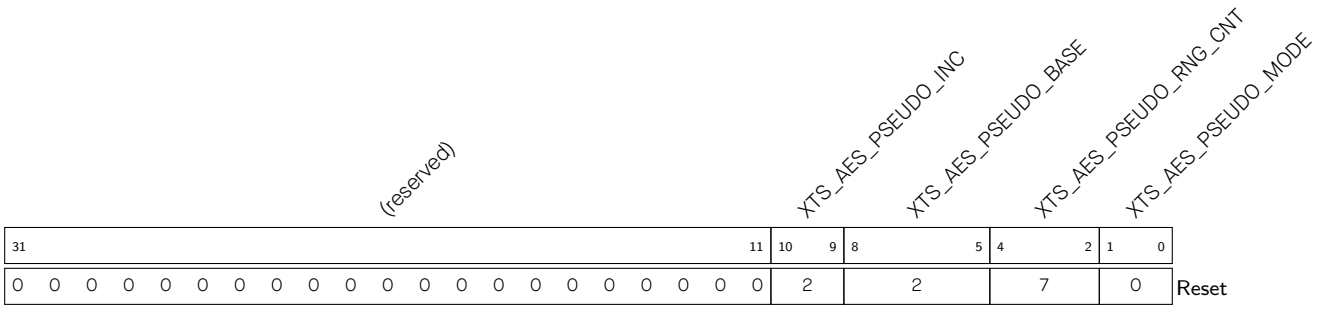
XTS_AES_CRYPT_SECURITY_LEVEL 配置外部加解密的安全等级。

0：时钟抗 DPA 功能已关闭

1-7：数字越大，则加解密安全性更高

(R/W)

Register 29.6. XTS_AES_PSEUDO_ROUND_CONF_REG (0x038C)



XTS_AES_MODE_PSEUDO 配置增加伪轮次抗 DPA 计算的具体 XTS-AES 工作步骤。

0：计算过程中不增加伪轮次

1：在计算扭曲值时增加伪轮次

2：在计算扭曲值时和计算数据单元的某些轮次时增加伪轮次

3：在计算扭曲值时和计算数据单元的所有轮次时增加伪轮次

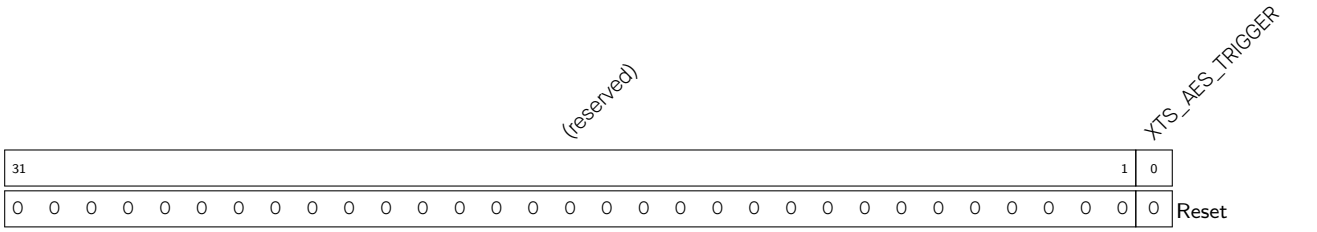
(R/W)

XTS_AES_PSEUDO_RNG_CNT 配置在伪轮次抗 DPA 计算中伪密钥的更新频率。(R/W)

XTS_AES_PSEUDO_BASE 配置伪轮次抗 DPA 计算中的基本轮数。(R/W)

XTS_AES_PSEUDO_INC 配置伪轮次抗 DPA 计算中的递增轮数。(R/W)

Register 29.7. XTS_AES_TRIGGER_REG (0x034C)



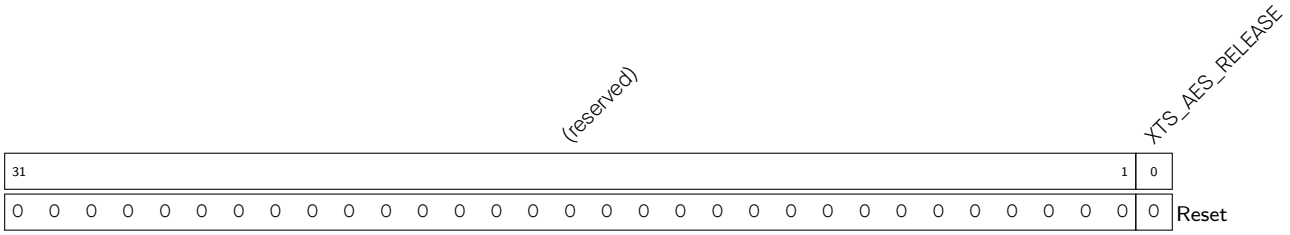
XTS_AES_TRIGGER 配置是否使能手动加密运算。

0：关闭手动加密运算

1：使能手动加密运算

注意，此配置尽在手动加密状态为 0 时有效。此配置生效后，手动加密状态变为 1，计算完成后，手动加密状态变为 2。(WT)

Register 29.8. XTS_AES_RELEASE_REG (0x0350)



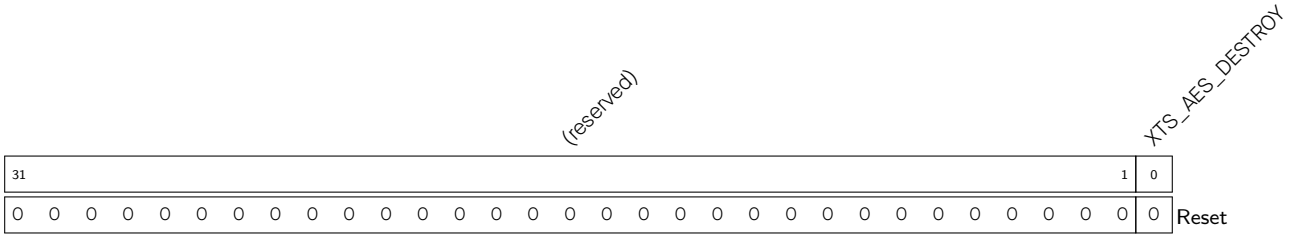
XTS_AES_RELEASE 配置是否向 MSPI 发布加密结果。

0：不发布

1：发布

此配置仅在手动加密状态为 2 时有效。此配置生效后，手动加密状态变为 3。(WT)

Register 29.9. XTS_AES_DESTROY_REG (0x0354)



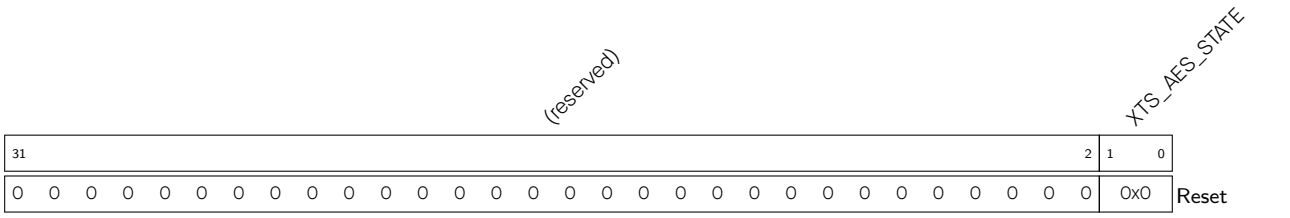
XTS_AES_DESTROY 配置是否销毁加密结果。

0：无效

1：销毁加密结果

此配置仅在手动加密状态为 3 时有效。此配置生效后，手动加密状态变为 0。(WT)

Register 29.10. XTS_AES_STATE_REG (0x0358)



XTS_AES_STATE 表示手动加密模块状态。

0 (XTS_AES_IDLE)：空闲

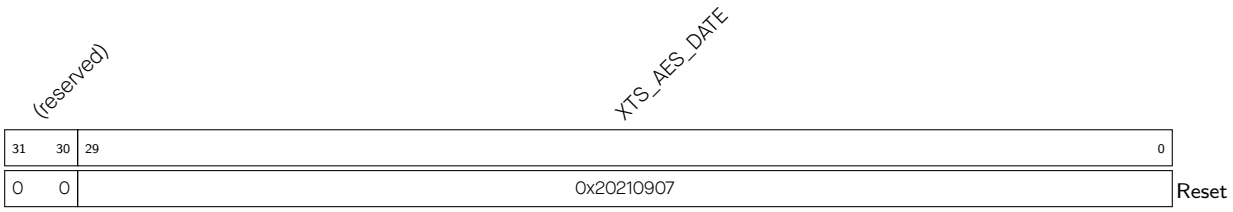
1 (XTS_AES_BUSY)：计算中

2 (XTS_AES_DONE)：计算完成，但手动加密结果数据对 MSPI 不可见

3 (XTS_AES_RELEASE)：手动加密结果对 MSPI 可见

(RO)

Register 29.11. XTS_AES_DATE_REG (0x035C)



XTS_AES_DATE 版本控制寄存器。(R/W)

第 30 章

随机数发生器 (RNG)

30.1 概述

ESP32-C5 内置一个真随机数发生器，其生成的 32 位随机数可作为加密等操作的基础。

30.2 主要特性

ESP32-C5 的随机数发生器可通过物理过程而非算法生成真随机数，所有生成的随机数在特定范围内出现的概率完全一样。

30.3 功能描述

系统可以从随机数发生器的寄存器 LPPERI_RNG_DATA_SYNC_REG 中读取随机数，每个读到的 32 位随机数都是真随机数，噪声源为系统中的**热噪声**、**异步时钟**和**异步计数器**。

- **热噪声**可以来自 SAR ADC 或高速 ADC 或两者兼有。当芯片的 SAR ADC 或高速 ADC 工作时，就会产生比特流，并通过异或 (XOR) 逻辑运算作为随机数种子进入随机数生成器。
- RC_FAST_CLK 是一种**异步时钟源**，会产生电路亚稳态。这种亚稳态也可以作为熵，进入随机数生成器。
- **异步计数器**是利用异步时钟源进行计数，异步时钟源会产生电路亚稳态，可以作为随机数种子进入随机数生成器。异步计数器分为 2 种：
  - 第一种是直接利用片上的 **实时时钟定时器 (RTC_TIMER)** 模块产生的计数值，并通过异或（XOR）逻辑运算作为随机数种子进入随机数生成器；
  - 第二种是使用 **环形缓冲区 (BUF_CHAIN)** 作为噪声源。

  这两种异步计数器可以各自独立工作，也可以多个一起工作。

> **说明：**
> 关于噪声源的限制，请查看 章节编程指南的说明。

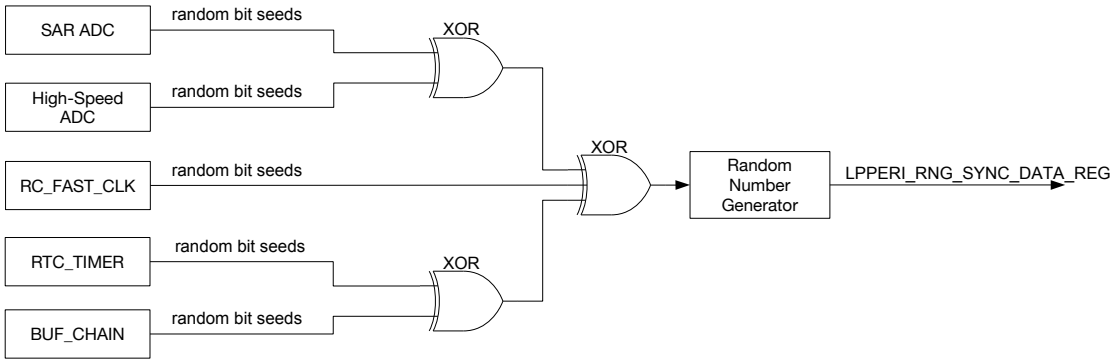


图 30.3-1. 噪声源

当 SAR ADC 打开时，每个 RC_FAST_CLK 时钟周期内（来自内部 RC 振荡器，详见 9 复位和时钟 章节），随机数发生器将获得 2 位的熵。因此，为了获得最大的熵值，建议读取 LPPERI_RNG_DATA_SYNC_REG 寄存器时的速率不超过 1 MHz。

当高速 ADC 打开时，每个 APB 时钟周期（通常为 80 MHz）内，随机数发生器将获得 2 位的熵。因此，为了获得最大的熵值，建议读取 LPPERI_RNG_DATA_SYNC_REG 寄存器时的速率不超过 5 MHz。

我们在仅打开高速 ADC 的状态下，以 5 MHz 的速率从 LPPERI_RNG_DATA_SYNC_REG 读取了 2 GB 的数据样本，并使用 Dieharder 随机数测试套件（版本 3.31.1）对样本进行了测试。最终，样本通过了所有测试。

30.4 编程指南

要开启随机数生成器，需向 LPPERI_RNG_CK_EN 写 1 开启随机数生成器时钟。使用 ECDSA 数字签名外设 (ECDSA_DS) 时，无论 LPPERI_RNG_CK_EN 配置如何，随机数生成器时钟都会自动启动。

在使用 ESP32-C5 的随机数生成器时，如果熵源没有开启，可能会导致产生伪随机数，应注意避免。其中，

- SAR ADC 受控于 DIG ADC 控制器。详见章节 46 *ADC 控制器*。
- 高速 ADC 在 Wi-Fi 或蓝牙开启时自动打开。
- RC_FAST_CLK 处于常开状态。
- 异步计数器
  - RTC 定时器 (RTC_TIMER) 通过设置 LPPERI_RTC_TIMER_EN 开启。
  - 环形缓冲区 (BUF_CHAIN) 通过置位 LPPERI_RNG_SAMPLE_ENABLE 开启。

说明：

1. 在 Wi-Fi 开启时，极端情况下高速 ADC 有读值饱和的可能，这会降低熵值。因此，建议在 Wi-Fi 开启时，同时打开 SAR ADC 产生随机数。
2. RC_FAST_CLK 时钟和异步计数器仅可以提高随机数发生器的熵值。然而，为了保证随机数发生器可以获得足够大的熵值，仍建议在使用随机数发生器时至少保证 SAR ADC 或高速 ADC 处于工作状态。

请多次读取 LPPERI_RNG_DATA_SYNC_REG 寄存器的值，直至获得足够多的随机数。在读取寄存器时，注意控制速率不要超过章节 30.3 的介绍。

30.5 寄存器列表

本小节的所有地址均为相对于随机数生成器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

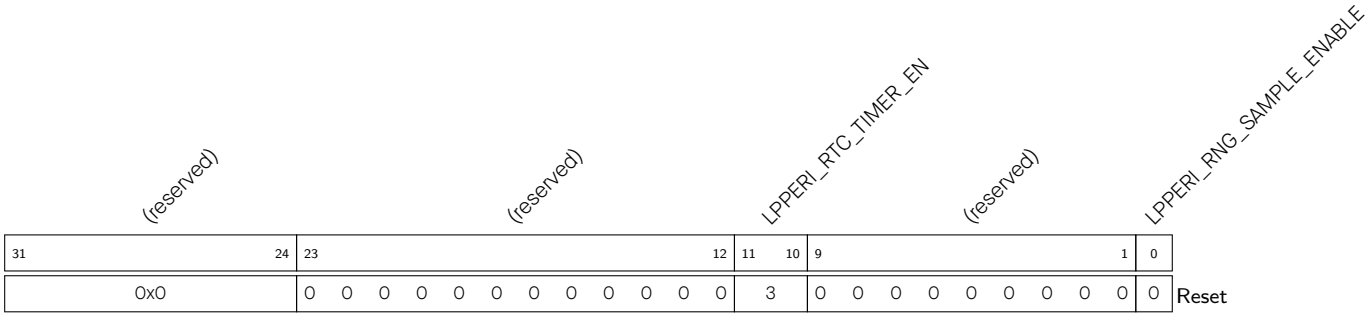
名称	描述	地址	访问
LPPERI_RNG_CFG_REG	RNG 配置寄存器	0x0024	varies
LPPERI_RNG_DATA_SYNC_REG	RNG 数据同步寄存器	0x0028	RO

30.6 寄存器

本小节的所有地址均为相对于随机数生成器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 30.1. LPPERI_RNG_CFG_REG (0x0024)



LPPERI_RNG_SAMPLE_ENABLE 配置是否使能 BUF_CHAIN。

0：关闭

1：使能

(R/W)

LPPERI_RTC_TIMER_EN Bit[0]：配置是否在 CRC 前使能 RTC 定时器。

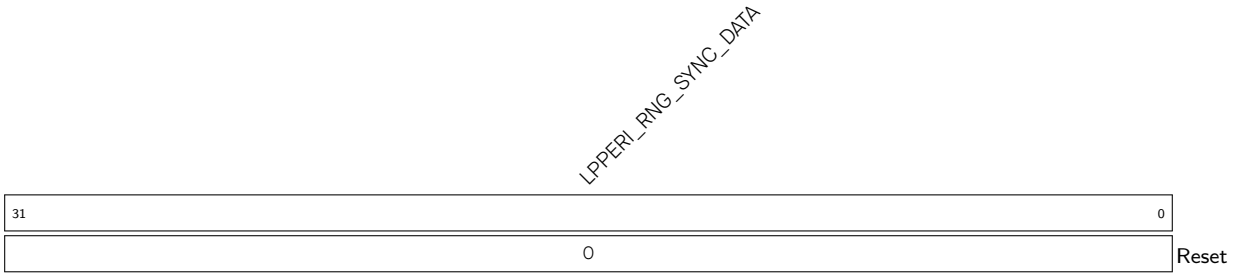
Bit[1]: 配置是否在 CRC 后使能 RTC 定时器。

0：关闭

1：使能

(R/W)

Register 30.2. LPPERI_RNG_DATA_SYNC_REG (0x0028)



LPPERI_RNG_SYNC_DATA 表示 RNG 同步数据。(RO)

第 31 章

密钥管理器

31.1 概述

密钥管理是系统的安全核心，利用密钥管理器和硬件唯一密钥 (HUK) 生成器，可安全地实现密钥的存储和部署。使用芯片的 HUK 作为信任根，芯片的任何区域均不存储明文密钥，因此提高了芯片的整体安全性。

与使用 eFuse 存储密钥相比，密钥管理器和 HUK 生成器可以对系统进行以下改进：

- 环境适应性：支持密钥的安全部署，即使外部环境不可信任，依然可以安全地完成密钥部署。
- 抗攻击能力：每个芯片的 SRAM 都有一个独一无二的默认值，软件无法访问该默认值。明文密钥不存储在芯片的任何物理位置，因此可以抵御切片攻击和复制攻击。例如直接通过切片查看 eFuse 中密钥的方法将会失效。
- 密钥管理灵活性：密钥管理器允许用户将密钥信息（非明文）存储在外部存储器中。与 eFuse 不同，密钥管理器允许用户创建无限数量的密钥，进行动态密钥更改、密钥替换等。

31.2 背景知识

SRAM PUF（物理不可克隆函数）利用 SRAM 单元上电时的天然初始状态，将这些初始状态用作芯片的“指纹”。芯片上电时，每个未初始化的 SRAM 位会因为晶体管在制造过程中的微小差异而倾向于变成 0 或 1。这些差异对每颗芯片都是独一无二的，因此生成的位模式也具有设备唯一性。

利用这一特性可以在芯片上生成一个硬件唯一密钥（HUK），且无需在芯片上存储这个密钥：芯片可以在每次启动时依靠物理特性重新生成密钥，而不需要从 flash 或 eFuse 中读取。HUK 随后可用作安全启动、设备身份与认证、加密密钥以及封装其他密钥的信任根。

但原始的 SRAM PUF 位并不完全稳定。大多数位在每次上电时都会产生相同的值，但某些位可能会因为温度、电压或噪声等因素而出现少量翻转。为了生成稳定的 HUK，芯片在 HUK 生成阶段，会生成具备纠错能力的辅助数据（huk_info）用于恢复 HUK，用户需要将这个辅助数据存储在 flash 中。

在后续的每次启动中，硬件会读取新的（带噪声的）PUF 响应，并利用辅助数据将任何翻转的位纠正回参考值，从而重新构造出部署阶段生成的完全相同的 HUK。随后，HUK 会被加载到加密硬件内部的隐藏密钥槽中（绝不会进入普通 RAM），使设备能够基于 SRAM PUF 获得一把可重复生成、每颗芯片唯一的秘密密钥，即使底层的 PUF 位在每次上电时并非 100% 相同。

31.3 术语

为了更好地说明 HUK 生成器和密钥管理器的功能，本章使用了以下术语及参数。

术语

HUK 硬件唯一密钥 (Hardware Unique Key, HUK)，使用 SRAM PUF 从设备上电时的默认初始值中派生出来的硬件密钥。

PUF 物理不可克隆函数 (Physical Unclonable Function, PUF) 基于 SRAM，属于一种安全技术，利用 SRAM 在上电时的独特且不可预测的行为为设备生成唯一的指纹或加密密钥。

HUK 生成模式 在该模式下，HUK 生成器生成新的 HUK。

HUK 恢复模式 在该模式下，HUK 生成器恢复已部署的 HUK。

随机部署模式 在该模式下，密钥管理器部署随机私钥。随机私钥由随机数生成器生成。

AES 部署模式 在该模式下，密钥管理器部署指定私钥。

ECDHO 部署模式 在该模式下，密钥管理器部署协商私钥 ($k1 * k2 * G$)。该私钥由密钥 $k1$ 和 $k2 * G$ 共同生成。

ECDH1 部署模式 在该模式下，密钥管理器部署协商私钥。该私钥由密钥 $k1$、$k2 * G$、和 $k2\_info$ 共同生成。

私钥恢复模式 在该模式下，密钥管理器恢复已部署的私钥。

key_info 导出模式 在该模式下，密钥管理器重新生成 key_info，用户可将该 key_info 导出到密钥管理器的内部存储器中。

参数

huk_info 包含模糊提取信息和对 HUK 进行 RS 编码后的纠错码信息，不包含任何 HUK 本身的信息。该 huk_info 可用于恢复已部署的 HUK。

key_info 在密钥管理器的所有模式下，key_info 均等于 AES{AES{key,rand_num}||rand_num,HUK}。

$AES\{a, b\}$ 以 b 为密钥对明文 a 进行加密，算法为 AES_ECB，即 AES 模块的典型模式。

$rand\_num$ 随机数

$a||b$ 将两段数据连接成一段数据。

$k1$ 指定私钥

$k2$ 辅助密钥，用于辅助密钥部署，在 AES 部署模式和 ECDH1 部署模式下使用。在 AES 部署模式下，其值不影响部署的密钥；在 ECDH1 部署模式下，其值会影响部署的协商密钥。

$k1\_encrypted$ 使用 $k2$ 对 $k1$ 进行加密而生成的密钥。在 AES 部署模式和 ECDH1 部署模式下使用，$k1\_encrypted$ = AES{k1,k2}

$k2\_info$ $k2$ 的恢复信息，对应辅助密钥 $k2$。在 AES 部署模式和 ECDH1 部署模式下使用，$k2\_info$ = AES{AES{k2,rand_num}||rand_num,init_key}。

$init\_key$ 用于 $k2\_info$ 的生成过程，以及从 $k2\_info$ 恢复为 $k2$ 的过程。其值可以存储在用途为 EFUSE_KM_INIT_KEY 的 eFuse 块中，也可以来自于软件配置。

sw_init_key 软件配置的 $init\_key$。

$k1 * k2 * G$ 协商密钥，通过在椭圆曲线上计算 $k1 * k2 * G$ 生成。

$k1 * G$ 在椭圆曲线上计算 $k1$ 生成的密钥。

$k2 * G$ 辅助公钥，通过在椭圆曲线上计算 $k2 * G$ 生成，该公钥用于生成 $k1 * k2 * G$。

$key\_purpose$ 密钥的用途。

31.4 主要特性

HUK 生成器具有以下特性：

- HUK 生成模式：
  - 生成新 HUK 及其对应的 HUK 恢复信息
- HUK 恢复模式：
  - 使用 HUK 恢复信息来恢复 HUK
- HUK 恢复错误提示
- HUK 风险等级提示

密钥管理器具有以下特性：

- 密钥数量无限
- 指定私钥部署（AES 部署模式）
  - 用户可指定密钥的值
- 协商私钥部署（ECDHO 部署模式）
  - 最高安全模式，无需担心外部渠道的数据泄露
  - 获取私钥需初始化芯片
  - 密钥值为芯片与用户协商所得
- 协商私钥部署（ECDH1 部署模式）
  - 利用辅助密钥部署协商私钥
  - 获取私钥无需启动芯片
- 随机密钥部署（随机部署模式）
  - 部署硬件生成的随机密钥，无人知道其确切值
- 私钥恢复部署
  - 输入部署时生成的密钥信息可恢复完全相同的密钥
- 密钥信息导出
  - 为同一个密钥每次生成不同的密钥信息

31.5 结构概览

密钥管理器和 HUK 生成器的顶层架构如下图所示。

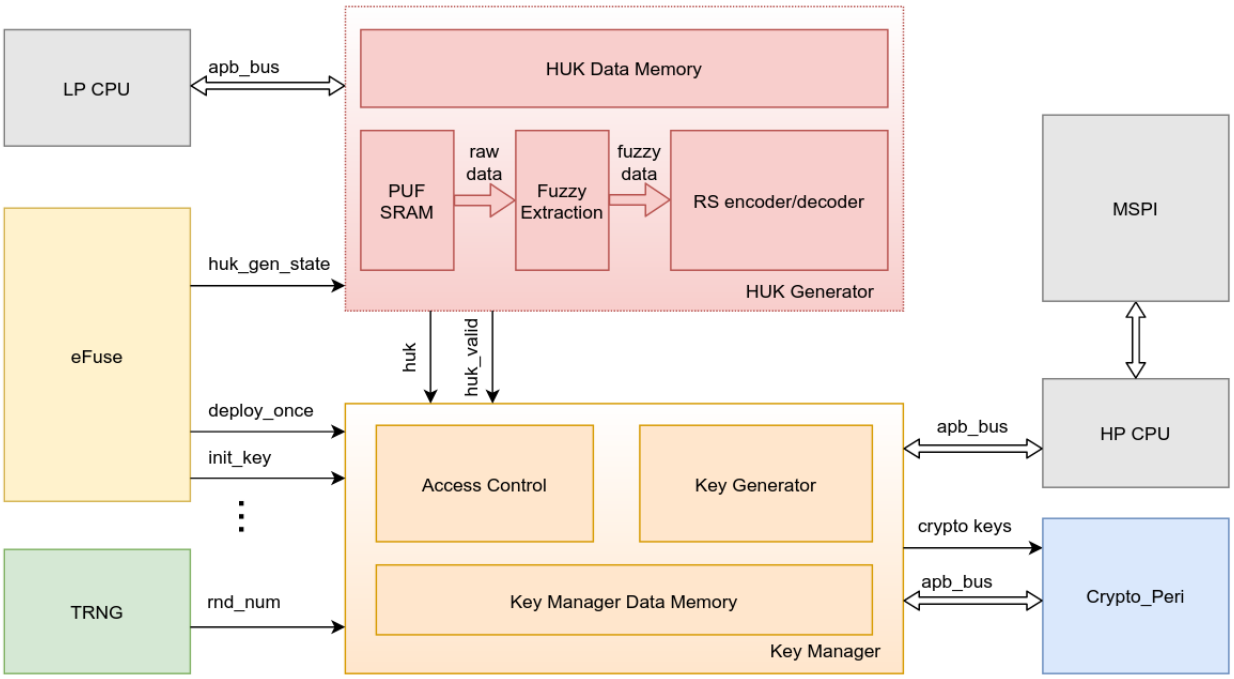


图 31.5-1. 密钥管理器 & HUK 生成器架构

密钥管理器与外部模块的交互路径如下：

- **eFuse 路径：** 使用 eFuse 位来控制密钥管理器的配置。
- **HP APB 路径：** 使用 HP APB 总线来配置寄存器和传输存储器数据。
- **HUK 路径：** 从 HUK 生成器接收 HUK。
- **Crypto 路径：** 使用 Crypto APB 总线控制 ECC/AES 等密码加速器实现密码运算。
- **TRNG 路径：** 从 TRNG 接收真随机数据。

HUK 生成器与外部模块的交互路径如下：

- **eFuse 路径：** 使用 eFuse 位来控制 HUK 生成器的配置。
- **LP APB 路径：** 使用 LP APB 总线来配置寄存器和传输存储器数据。
- **密钥管理器路径：** 发送 HUK 给密钥管理器。

31.6 功能描述

31.6.1 HUK 生成器

HUK 生成器基于 SRAM 物理不可克隆函数 (PUF) 为每个芯片生成独一无二的 HUK。软件无法以任何方式访问此 HUK，因此密钥管理器使用此 HUK 可以保证其安全性可靠。

31.6.1.1 HUK 生成过程

HUK 生成器具有以下模式：

- **HUK 生成模式：**生成新的 HUK 及其对应的 HUK 恢复信息 (*huk_info*)
- **HUK 恢复模式：**根据 *huk_info* 恢复 HUK

EFUSE_KM_HUK_GEN_STATE 和 HUK_MODE 共同控制 HUK 生成器的模式。

表 31.6-1. HUK 生成器工作模式

HUK_MODE	eFuse *	模式
1	配置成 1 的比特数量为偶数	HUK 生成模式
	配置成 1 的比特数量为奇数	HUK 恢复模式
0	—	HUK 恢复模式

* 这里指 EFUSE_KM_HUK_GEN_STATE 中所含 1 的总数。

说明：

- HUK 生成模式结束后，建议向 EFUSE_KM_HUK_GEN_STATE 写入一个 1，此时 eFuse 中配置成 1 的比特数量为奇数。这样，芯片下次启动时，HUK 生成器只能进入 HUK 恢复模式。此时 HUK 恢复模式下恢复的 HUK 将与本次 HUK 生成模式中生成的 HUK 相同。
- 如果在 HUK 生成模式结束后，芯片掉电前没有写入 EFUSE_KM_HUK_GEN_STATE，HUK 仍然可以配置为在下次上电时进入 HUK 生成模式，但新生成 HUK 的值不再与上次用于密钥部署的 HUK 相同。用户可以利用这一点来生成多个 HUK，同时利用其对应的 *huk_info* 恢复所需要的 HUK。
- *huk_info* 仅可被用于同一块芯片，即在芯片 A 生成的 *huk_info* 无法在芯片 B 中恢复 HUK。

31.6.1.2 HUK 状态检查

用户可读取 HUK_STATUS_REG 来判断 HUK 的当前状态，并决定是否需要重新生成 HUK。HUK_STATUS_REG 中各个字段的功能如下：

- HUK_STATUS：表示 HUK 当前的生成状态。
  - 0：HUK 尚未生成。
  - 1：HUK 已成功生成且有效。
  - 2：HUK 已生成但无效。
  - 3：保留。
- HUK_RISK_LEVEL：表示 HUK 的当前风险等级，值为 0~7，值越大，HUK 的风险越高。当值为 7 时，表示 HUK 不再可用。此时 HUK_STATUS 的值也将相应变为 2。

说明：

- HUK 生成器使用 SRAM PUF 的默认值来生成 HUK。HUK 的风险等级指的是 SRAM PUF 的当前默认值与创建 HUK 时 SRAM PUF 的默认值之间的差值。
- 在一些特殊的应用中，比如用户想弃用某个密钥，使对应的 *key_info* 变为无效时，可以选择通过更新 HUK 的方式来实现上述目的。HUK 一旦变更，用户将无法用新的 *huk_info* 结合之前的 *key_info* 恢复密钥，即之前部署的密钥及其对应的 *key_info* 均将变为无效。

31.6.2 密钥管理器

说明：
每次使用密钥管理器进行密钥部署时，均需要确保 HUK 已经处于有效状态。可检查 HUK_STATUS_REG 来确定 HUK 的状态。

密钥管理器实现了包含四种模式的私钥部署流程、统一的私钥恢复流程以及 key_info 导出流程。

- **四种私钥部署模式**
  - 随机部署模式：部署随机私钥
  - AES 部署模式：部署指定私钥
  - ECDH0 部署模式：部署协商私钥
  - ECDH1 部署模式：部署协商私钥

  四种部署模式可以让用户根据需求，在不同安全等级的环境中选择最便捷的部署流程。
- **一种恢复模式**
  - 私钥恢复模式：恢复已部署的私钥
    该模式允许用户从任意数量的已部署的密钥中自由选择所需的密钥进行恢复。
- **一种导出模式**
  - 导出 key_info 模式：生成并输出已部署私钥的 key_info
    该模式允许用户为已部署的密钥重新生成 key_info。

31.6.2.1 随机部署模式

用户可使用该模式来部署随机密钥。该密钥由随机数生成器生成。

说明：
随机部署模式十分便捷，非常适用于不需要知道私钥具体值的应用场景。

31.6.2.2 AES 部署方式

用户可使用该模式来部署指定私钥。该模式需要以下密钥及参数：

- HUK：由 HUK 生成器生成，并发送给密钥管理器。
- $k1$：指定的私钥，待部署到芯片。
- $k2$：辅助密钥，用于加密 $k1$，保证在通讯链路上不存在 $k1$ 的明文，保障数据安全。
- $init\_key$：初始辅助密钥，用于生成和解密 $k2\_info$ 的密钥。
- $k2\_info$：由 $k2$ 和 $init\_key$ 生成，用于恢复得到 $k2$。计算公式为 $k2\_info = AES\{AES\{k2, rand\_num\}||rand\_num, init\_key\}$。

部署流程如下：

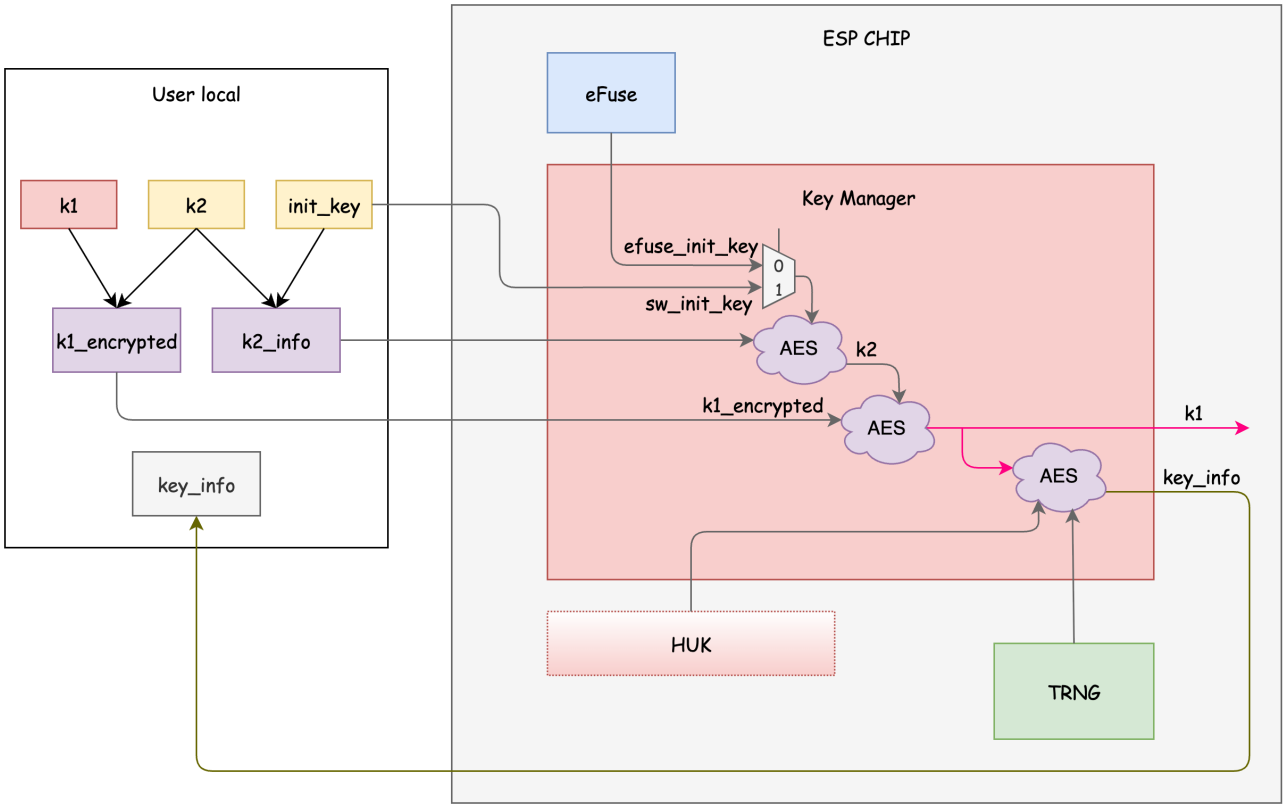


图 31.6-1. AES 部署模式

1. 用户在本地生成私钥 $k1$，辅助密钥 $k2$ 和初始辅助密钥 $init\_key$。
2. 用户本地计算得到 $k2\_info$ 和 $k1\_encrypted$。
3. 在 ESP32-C5 密钥管理器中完成密钥部署过程：
   (a) 根据使用场景，将 $init\_key$ 写入用途为 EFUSE_KM_INIT_KEY 的 eFuse 块或 sw_init_key 中。
   (b) 将 $k2\_info$ 和 $k1\_encrypted$ 写入密钥管理器内部存储器。
   (c) 等待密钥管理器完成部署。
   (d) 用户通过指定地址从密钥管理器内部存储器中读取 key_info。

注意：

- 由于芯片每次上电都需要重新部署密钥，如果下次上电时要使用 key_info 进行密钥恢复，则必须确保芯片断电后 key_info 能够继续保存。一种常见的方法就是将其存储在 flash 中。
- AES 部署模式适用于工厂生产：生产商可以向工厂端提供初始辅助密钥 $init\_key$ 和使用 $k1$ 加密的固件，并在固件中存储 $k2\_info$ 和 $k1\_encrypted$。
  - 这种情况下，建议在固件中加入初次部署私钥的流程，存储 key_info，并在后续芯片上电启动中使用恢复部署模式，而不是每次都使用 AES 部署模式，以有效抵御针对 eFuse 的切片攻击。
  - 注意，这种部署方式需要保证工厂端可信，因为工厂可以获得初始辅助密钥 $init\_key$ 的明文（用于烧录到 eFuse 中），如果工厂不可信，且知道如何使用 $init\_key$ 将 $k2\_info$ 解密为 $k2$，就可能会导致 $k1$ 的泄露。
- 在每一次使用 AES 部署模式部署密钥时，都需要保证 $init\_key$，$k2\_info$ 和 $k1\_encrypted$ 的值不会同时泄露。

- 在使用 EFUSE_KM_INIT_KEY eFuse 块中存储的 $init\_key$ 时，传输 $k2\_info$ 和 $k1\_encrypted$ 时可以处于不可信环境。但这意味着用户必须在本地保留第一次写入 EFUSE_KM_INIT_KEY eFuse 块中的 $init\_key$。
- 在使用 sw_init_key 时，传输 sw_init_key，$k2\_info$ 和 $k1\_encrypted$ 时必须处于可信环境。

31.6.2.3 ECDHO 部署模式

用户可使用该模式来部署协商私钥。该模式需要以下密钥及参数：

- HUK：由 HUK 生成器生成，并发送给密钥管理器。
- $k1$：根密钥，用户自己的私钥，只存在于用户本地。
- $k1 * G$：在椭圆曲线上计算 $k1$*G 生成的密钥。
- $k2 * G$：辅助公钥，通过在椭圆曲线上计算 $k2 * G$ 生成。
- $k1 * k2 * G$：待部署的协商私钥，密钥管理器使用 $k1 * G$ 和 $k2$ 生成协商私钥，用户本地则使用 $k1$ 和 $k2 * G$ 生成协商私钥。
- key_info：$k1 * k2 * G$ 的恢复信息。

该部署过程中，任何用户本地和芯片之外的信息泄露，均不会影响 $k1 * k2 * G$ 的生成或产生泄露风险。部署流程如下：

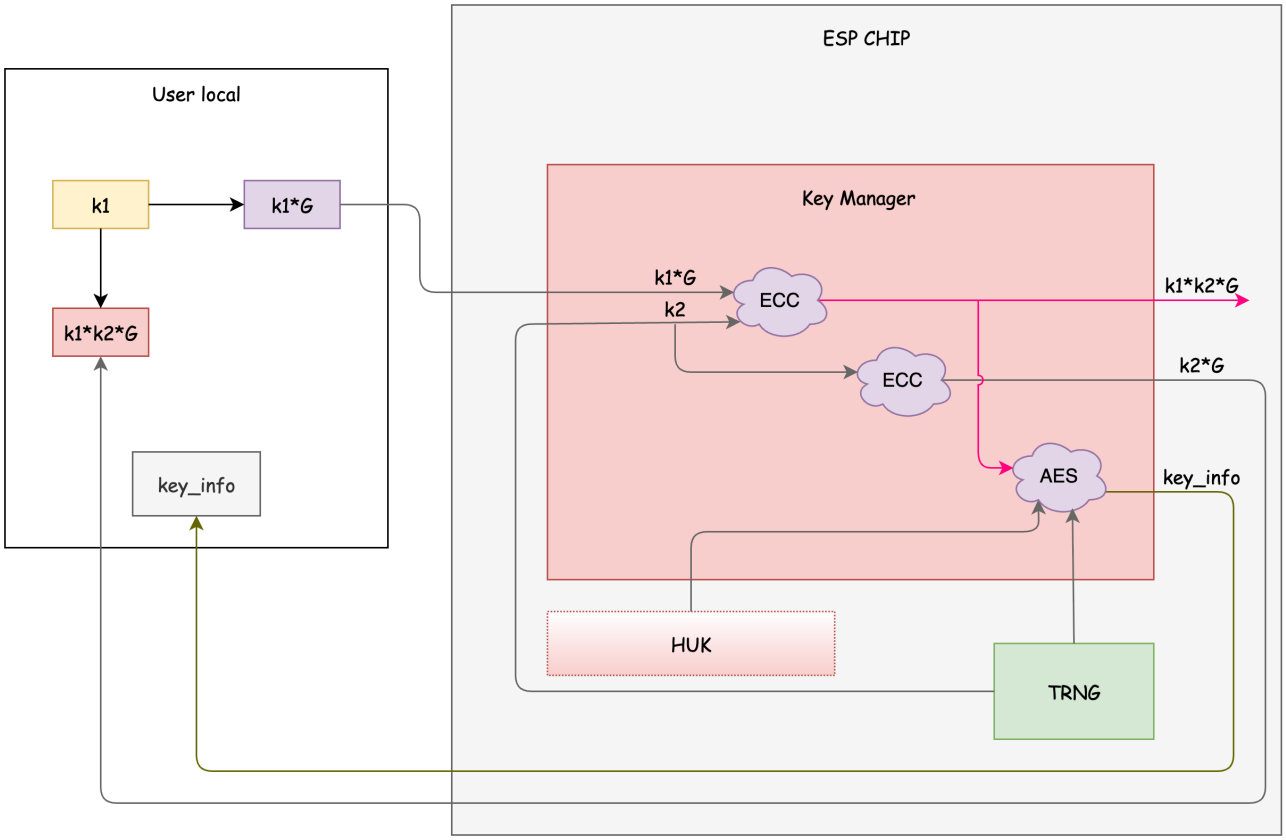


图 31.6-2. ECDHO 部署模式

1. 用户本地生成 $k1$ ，并计算得到 $k1 * G$。
2. 在 ESP32-C5 密钥管理器中完成密钥部署过程：

(a) 将 $k1 * G$ 写入密钥管理器的内部存储器。

(b) 等待密钥管理器完成部署。

(c) 用户通过指定地址从密钥管理器内部存储器中读取 $k2 * G$ 和协商私钥 $k1 * k2 * G$ 对应的 key_info。

3. 将读取的 $k2 * G$ 与用户本地私钥 $k1$ 一起生成协商密钥 $k1 * k2 * G$。

说明：

- ECDHO 部署模式可以保证协商私钥只能在用户本地和密钥管理器内部生成，部署过程中的任何信息泄露都不会导致协商私钥的泄露。因此，使用 ECDHO 部署模式时，传输数据的过程可以处在不可信环境。
- ECDHO 部署模式下，用户不能使用指定值的密钥。
- ECDHO 部署模式下，用户只能在运行 ESP32-C5 密钥管理器后才能获得协商好的私钥。不适用于需要提前获知私钥具体值的应用场景。

31.6.2.4 ECDH1 部署模式

用户可使用该模式来部署协商私钥。该模式需要以下密钥及参数：

- HUK：由 HUK 生成器生成，并发送给密钥管理器。
- $k1$：根密钥，用户自己的私钥，在 ECDH1 部署模式中只存在于用户本地。
- $k1 * G$：在椭圆曲线上计算 $k1$*G 生成的密钥。
- $k2$：辅助私钥，在 ECDH1 部署模式中只存在于用户本地和 ESP32-C5 密钥管理器内部。
- $k2 * G$：辅助公钥，通过在椭圆曲线上计算 $k2 * G$ 生成。
- $k2\_info$：由 $k2$ 和 $init\_key$ 生成，用于恢复得到 $k2$。计算公式为 $k2\_info = AES\{AES\{k2, rand\_num\}||rand\_num, init\_key\}$。
- $k1 * k2 * G$：待部署的协商私钥，密钥管理器使用 $k1 * G$ 和 $k2$ 生成协商私钥，用户本地则使用 $k1$ 和 $k2 * G$ 生成协商私钥。
- key_info：$k1 * k2 * G$ 的恢复信息。

部署流程如下：

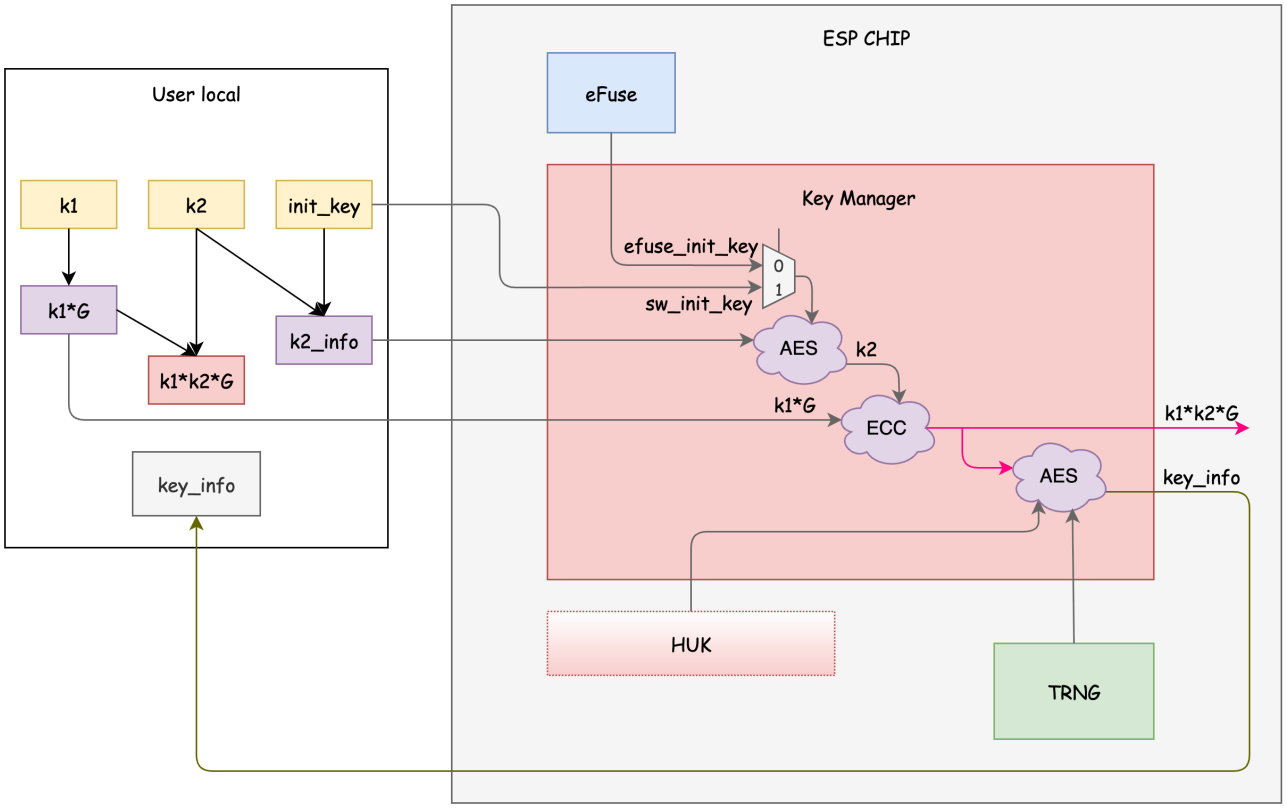


图 31.6-3. ECDH1 部署模式

1. 用户在本地生成私钥 $k1$，辅助密钥 $k2$ 和初始辅助密钥 $init\_key$。
2. 用户本地计算得到 $k1 * G$，$k2 * G$，$k2\_info$，以及协商私钥 $k1 * k2 * G$。
3. 在 ESP32-C5 密钥管理器中完成密钥部署过程：
   (a) 根据使用场景（见下文说明），将 $init\_key$ 写入用途为 EFUSE_KM_INIT_KEY 的 eFuse 块或 sw_init_key 中。
   (b) 将 $k1 * G$ 和 $k2\_info$ 写入密钥管理器的内部存储器。
   (c) 等待密钥管理器完成部署。
   (d) 用户通过指定地址从密钥管理器内部存储器中读取 $k2 * G$ 和协商私钥 $k1 * k2 * G$ 对应的 key_info。

说明：

- ECDH1 部署模式适用于工厂生产：生产商可以向工厂端提供初始辅助密钥 $init\_key$ 和使用 $k1 * k2 * G$ 加密的固件，并在固件中存储 $k2\_info$ 和 $k1 * G$。
  - 这种情况下，建议在固件中加入初次部署私钥的流程，存储 key_info，并在后续芯片上电启动中使用恢复部署模式，而不是每次都使用 ECDH1 部署模式，以有效抵御针对 eFuse 的切片攻击。
  - 注意，这种部署方式需要保证工厂端可信，因为工厂可以获得初始辅助密钥 $init\_key$ 的明文（用于烧录到 eFuse 中），如果工厂不可信，且知道如何使用 $init\_key$ 将 $k2\_info$ 解密为 $k2$，就可能会导致 $k1 * k2 * G$ 的泄露。
- 与 AES 部署模式类似，在每一次使用 ECDH1 部署模式部署密钥时，都需要保证 $init\_key$，$k2\_info$ 和 $k1 * G$ 的值不会同时泄露。

  - 在使用 EFUSE_KM_INIT_KEY eFuse 块中存储的 $init\_key$ 时，传输 $k2\_info$ 和 $k1 * G$ 时可以处于不可信环境。但这意味着用户必须在本地保留第一次写入 EFUSE_KM_INIT_KEY eFuse 块中的 $init\_key$。
  - 在使用 sw_init_key 时，传输 sw_init_key，$k2\_info$ 和 $k1 * G$ 时必须处于可信环境。

- 使用 ECDH1 部署模式更新密钥时，比使用 AES 部署模式拥有更高的安全性能。由于 $k1 * G$ 的值与 $k2$、$init\_key$ 均无关，因此在使用 ECDH1 部署模式更新协商密钥时，可以只更新 $k2\_info$，而不更新 $k1 * G$。用户可以：
  1. 在可信环境下将 $k1 * G$ 写入固件。
  2. 使用 ECDH1 部署模式更新密钥时，只传输 $init\_key$ 和 $k2\_info$。这种情况下，用户可在不可信环境中使用 sw_init_key 进行 ECDH1 模式部署密钥。

31.6.2.5 私钥恢复模式

私钥部署完成后，用户可通过部署时生成的 key_info 来恢复该私钥。具体过程为：

1. 读取外部存储器中的 key_info 并将其存放到密钥管理器内部存储器的固定地址中。
2. 等待密钥管理器完成部署。
3. 重复上述步骤 1 ~ 步骤 2，直到恢复所有需要的私钥。注意，如果恢复了两个或多个相同 $key\_purpose$ 的私钥，则最后一个私钥有效。关于 $key\_purpose$ 的说明，请参考小节 3。

31.6.2.6 key_info 导出模式

私钥部署成功后，用户可使用此模式重新生成并输出 key_info。
具体过程为：

1. 选择部署成功的 $key\_purpose$，进入 key_info 导出模式。
2. 等待密钥管理器完成部署。
3. 用户通过指定地址从密钥管理器内部存储器中读取 key_info。

说明：

- 该模式可用于对同一个密钥输出不同的 key_info。
- 利用该模式也可以只更新 HUK，而不更新私钥。具体操作为：
  - 先使用 HUK 生成模式更新 HUK
  - 再通过 key_info 导出模式更新 key_info。
    这样下次启动时可以使用新的 key_info 在新的 HUK 下恢复相同的私钥。

31.7 编程流程

31.7.1 HUK 生成器

HUK 生成过程分为六个阶段：

1. **IDLE 阶段**：此阶段用户配置模式和静态参数。

2. **PREP 阶段**：此阶段 HUK_STATE 处于 BUSY 状态，HUK 生成器执行 HUK 生成准备。用户只需等待此阶段结束，HUK 生成器将自动进入 LOAD 阶段。
3. **LOAD 阶段**：此阶段用户根据模式将 HUK 信息存入 HUK 生成器的内部存储器中。
4. **PROC 阶段**：此阶段 HUK_STATE 处于 BUSY 状态，HUK 生成器执行 HUK 生成操作。用户只需等待此阶段结束，HUK 生成器将自动进入 GAIN 阶段。
5. **GAIN 阶段**：此阶段用户根据模式从 HUK 生成器的内部存储器中读取所需的 HUK 信息。
6. **POST 阶段**：此阶段 HUK_STATE 处于 BUSY 状态，HUK 生成器执行 HUK 生成的后续工作。用户只需等待该阶段结束，HUK 生成器将自动回到 IDLE 阶段。

31.7.1.1 IDLE 阶段

用户检查 HUK_STATE，确认 HUK 生成器处于 IDLE 阶段。在 IDLE 阶段，用户需要进行如下配置：

1. **配置工作模式**：参考表 31.6-1 配置 HUK 生成器的工作模式。
2. **配置进入下一阶段**：置位 HUK_START，完成 IDLE 阶段，进入 PREP 阶段。

31.7.1.2 PREP 阶段

用户需要等待 PREP 阶段结束。可以通过以下任一方式完成此操作：

- **监测 HUK_STATE**：持续检查 HUK_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 HUK_PREP_DONE_INT，并通过中断函数进行处理。

31.7.1.3 LOAD 阶段

用户检查 HUK_STATE，确认 HUK 生成器处于 LOAD 阶段。在 LOAD 阶段，用户需要进行如下配置：

1. **根据模式进行配置：**
   - **HUK 生成模式**：无需配置。
   - **HUK 恢复模式**：将 HUK 恢复信息 huk_info（185 个字）写入 HUK_INFO_MEM 中。

> **说明：**
> 如果之前已经生成过多个 HUK 恢复信息并存储在外部存储器中，则在恢复模式下使用任意一个 HUK 恢复信息都会恢复当时对应的 HUK。

2. **配置进入下一阶段**：置位 HUK_CONTINUE，完成 LOAD 阶段，进入 PROC 阶段。

31.7.1.4 PROC 阶段

用户需要等待 PROC 阶段结束。可以通过以下任一方式完成此操作：

- **监测 HUK_STATE**：持续检查 HUK_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 HUK_PROC_DONE_INT，并通过中断函数进行处理。

31.7.1.5 GAIN 阶段

用户检查 HUK_STATE，确认 HUK 生成器处于 GAIN 阶段。在 GAIN 阶段，用户根据选择的模式提取所需的密钥信息：

1. **提取信息：**
   - **HUK 生成模式：** 从 HUK_INFO_MEM 中读取 huk_info（185 个字）。
   - **HUK 恢复模式：** 根据 HUK_UPDATE_REQ 判断是否需要更新 huk_info。
     - 需要更新：从 HUK_INFO_MEM 中读取 huk_info（185 个字）。
     - 无需更新：无需读取任何信息。
2. **配置进入下一阶段**：置位 HUK_CONTINUE，完成 GAIN 阶段，进入 POST 阶段。

31.7.1.6 POST 阶段

用户需要等待 POST 阶段结束。可以通过以下任一方式完成此操作：

- **监测 HUK_STATE**：持续检查 HUK_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 HUK_POST_DONE_INT，并通过中断函数进行处理。

31.7.2 密钥管理器

密钥管理器的密钥部署过程分为六个阶段：

1. **IDLE 阶段**：此阶段用户配置模式、密钥用途和静态参数。
2. **PREP 阶段**：此阶段 KEYMNG_STATE 处于 BUSY 状态，密钥管理器执行密钥部署准备工作。用户只需等待此阶段结束，密钥管理器将自动进入 LOAD 阶段。
3. **LOAD 阶段**：此阶段用户根据模式将密钥信息存入密钥管理器的内部存储器中。
4. **PROC 阶段**：此阶段 KEYMNG_STATE 处于 BUSY 状态，密钥管理器执行密钥部署操作。用户只需等待此阶段结束，密钥管理器将自动进入 GAIN 阶段。
5. **GAIN 阶段**：此阶段用户根据模式从密钥管理器的内部存储器中读取所需的密钥信息。
6. **POST 阶段**：此阶段 KEYMNG_STATE 处于 BUSY 状态，密钥管理器执行密钥部署的后续工作。用户只需等待该阶段结束，密钥管理器将自动回到 IDLE 阶段。

31.7.2.1 IDLE 阶段

用户检查 KEYMNG_STATE，确认密钥管理器处于 IDLE 阶段。在 IDLE 阶段，用户需要进行如下配置：

1. **完成 HUK 生成过程：**

   (a) 根据 HUK 是否已经生成，选择以下两种流程之一：

   - **HUK 尚未生成**：按照章节 31.7.1 所述完成 HUK 生成过程。
   - **HUK 已生成**：跳过 HUK 生成过程。

   (b) **验证 HUK 是否有效**：检查 KEYMNG_HUK_VALID：

   - 0，则 HUK 无效，返回步骤 1a，直到其值为 1。
   - 1，HUK 有效。

2. **配置静态参数**：包括 eFuse 位和静态参数寄存器及其对应的寄存器锁：

   - **EFUSE_KM_INIT_KEY**：256 位初始密钥，密钥管理器在 AES 和 ECDH1 部署模式中会用到该密钥。
   - EFUSE_KM_DEPLOY_ONLY_ONCE：控制对应的私钥是否只能在上电后部署一次。
     - **Bit[0]**：配置 ECDSA_DS 密钥是否只能部署一次。
       * 0：可以部署多次
       * 1：只能部署一次
     - **Bit[1]**：配置 FLASH 密钥是否只能部署一次。
       * 0：可以部署多次
       * 1：只能部署一次
     - **Bit[2]**：配置 HMAC 密钥是否只能部署一次。
       * 0：可以部署多次
       * 1：只能部署一次
     - **Bit[3]**：配置 RSA_DS_KEY 是否只能部署一次。
       * 0：可以部署多次
       * 1：只能部署一次
     - **Bit[4]**：配置 PSRAM 密钥是否只能部署一次。
       * 0：可以部署多次
       * 1：只能部署一次
   - EFUSE_FORCE_USE_KEY_MANAGER_KEY：控制相应的私钥是否只能来自密钥管理器。
     - **Bit[0]**：表示 ECDSA_DS 密钥是否只能来自密钥管理器。
       * 0：不必来自密钥管理器
       * 1：必须来自密钥管理器
     - **Bit[1]**：表示 FLASH 密钥是否只能来自密钥管理器。
       * 0：不必来自密钥管理器

    * 1：必须来自密钥管理器
  - **Bit[2]**：表示 HMAC 密钥是否只能来自密钥管理器。
    * 0：不必来自密钥管理器
    * 1：必须来自密钥管理器
  - **Bit[3]**：表示 RSA_DS_KEY 是否只能来自密钥管理器。
    * 0：不必来自密钥管理器
    * 1：必须来自密钥管理器
  - **Bit[4]**：表示 PSRAM 密钥是否只能来自密钥管理器。
    * 0：不必来自密钥管理器
    * 1：必须来自密钥管理器
- KEYMNG_USE_EFUSE_KEY：配置密钥的来源为 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的相应位为 0 时有效。
  - **Bit[0]**：配置 ECDSA_DS 密钥是否来自 eFuse 块。
    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[0] 为 0 时有效。
  - **Bit[1]**：配置 FLASH 密钥是否来自 eFuse 块
    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[1] 为 0 时有效。
  - **Bit[2]**：配置 HMAC 密钥是否来自 eFuse 块
    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[2] 为 0 时有效。
  - **Bit[3]**：配置 RSA_DS_KEY 是否来自 eFuse 块
    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[3] 为 0 时有效。
  - **Bit[4]**：配置 PSRAM 密钥是否来自 eFuse 块
    * 0：无效
    * 1：来自 eFuse 块，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[4] 为 0 时有效。
- **EFUSE_FORCE_DISABLE_SW_INIT_KEY**：控制是否永久禁止使用软件源 $init\_key$。
  - 0：不禁用
  - 1：禁用软件源 $init\_key$
- **KEYMNG_USE_SW_INIT_KEY**：配置使用软件配置的 sw_init_key 而不是用途为 EFUSE_KM_INIT_KEY 的 eFuse 块，仅当 EFUSE_FORCE_DISABLE_SW_INIT_KEY 为 0 时有效。
  - 0：使用用途为 EFUSE_KM_INIT_KEY 的 eFuse 块
  - 1：使用软件写入的初始密钥，即 sw_init_key

- **EFUSE_KM_RND_SWITCH_CYCLE**：配置密钥管理器切换随机数的周期。注意：此切换周期建议配置为 TRNG 模块的时钟周期所对应的密钥管理器时钟周期数。
  - 0：切换周期由 KEYMNG_RND_SWITCH_CYCLE 控制
  - 1：8 个密钥管理器时钟周期
  - 2：16 个密钥管理器时钟周期
  - 3：32 个密钥管理器时钟周期
- **KEYMNG_RND_SWITCH_CYCLE**：配置密钥管理器切换随机数的周期，单位为密钥管理器时钟周期。仅当 EFUSE_KM_RND_SWITCH_CYCLE 为 0 时生效。注意：此切换周期建议配置为 TRNG 模块的时钟周期所对应的密钥管理器时钟周期数。

3. **设置工作模式**：配置 KEYMNG_KGEN_MODE，完成密钥管理器的工作模式配置，具体配置见章节 31.6.2：
   - 0：随机部署模式
   - 1：AES 部署模式
   - 2：ECDH0 部署模式
   - 3：ECDH1 部署模式
   - 4：私钥恢复模式
   - 5：key_info 导出模式
   - 其他值：无效
4. **配置密钥**：配置 KEYMNG_KEY_PURPOSE 完成密钥用途 ($key\_purpose$) 的配置：
   - 1: ecdsa_key
   - 2: flash_256_key_1
   - 3: flash_256_key_2
   - 4: flash_128_key
   - 6: hmac_key
   - 7: dsa_key
   - 8: psram_256_key_1
   - 9: psram_256_key_2
   - 10: psram_128_key
   - 其他值：无效
5. **配置进入下一阶段**：置位 KEYMNG_START，完成 IDLE 阶段，进入 PREP 阶段。

31.7.2.2 PREP 阶段

用户需要等待 PREP 阶段结束。可以通过以下任一方式完成此操作：

- **监测 KEYMNG_STATE**：持续检查 KEYMNG_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 KEYMNG_PREP_DONE_INT，并通过中断函数进行处理。

31.7.2.3 LOAD 阶段

用户检查 KEYMNG_STATE，确认密钥管理器处于 LOAD 阶段。在 LOAD 阶段，用户需要进行如下配置：

1. **根据工作模式进行配置：**
   - **随机部署模式**：无需配置。
   - **AES 部署模式**：
     (a) 如果使用软件配置 $init\_key$ 的值：则将所需配置的值 sw_init_key（256 位）写入 KEYMNG_SW_INIT_KEY_MEM。
     (b) 将 $k2\_info$（512 位）写入 KEYMNG_ASSIST_INFO_MEM。
     (c) 将 $k1\_encrypted$（256 位）写入 KEYMNG_PUBLIC_INFO_MEM。
   - **ECDH0 部署模式：**
     (a) 将 $k1 * G$（512 位）写入 KEYMNG_PUBLIC_INFO_MEM。
   - **ECDH1 部署模式：**
     (a) 如果使用软件配置的 $init\_key$：则将 sw_init_key（256 位）写入 KEYMNG_SW_INIT_KEY_MEM；
     (b) 将 $k2\_info$（512 位）写入 KEYMNG_ASSIST_INFO_MEM。
     (c) 将 $k1 * G$（512 位）写入 KEYMNG_PUBLIC_INFO_MEM。
   - **私钥恢复模式：**
     (a) 将 key_info（512 位）写入 KEYMNG_ASSIST_INFO_MEM。
   - key_info **导出模式**：无需配置。
2. **配置进入下一阶段**：置位 KEYMNG_CONTINUE，完成 LOAD 阶段，进入 PROC 阶段。

31.7.2.4 PROC 阶段

用户需要等待 PROC 阶段结束。可以通过以下任一方式完成此操作：

- **监测 KEYMNG_STATE**：持续检查 KEYMNG_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 KEYMNG_PROC_DONE_INT，并通过中断函数进行处理。

31.7.2.5 GAIN 阶段

用户检查 KEYMNG_STATE，确认密钥管理器处于 GAIN 阶段。在 GAIN 阶段，用户根据选择的模式提取所需的密钥信息：

1. **提取信息：**
   - **随机部署模式：**
     (a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。
   - **AES 部署模式：**
     (a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。
   - **ECDH0 部署模式：**

(a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

(b) 从 KEYMNG_ASSIST_INFO_MEM 中读取 $k2 * G$（512 位）。

- **ECDH1 部署模式：**

(a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

- **私钥恢复模式**：无需读取任何信息。
- key_info **导出模式：**

(a) 从 KEYMNG_PUBLIC_INFO_MEM 中读取 key_info（512 位）。

2. **配置进入下一阶段**：置位 KEYMNG_CONTINUE，完成 GAIN 阶段，进入 POST 阶段。

31.7.2.6 POST 阶段

用户需要等待 POST 阶段结束。可以通过以下任一方式完成此操作：

- **监测 KEYMNG_STATE**：持续检查 KEYMNG_STATE，直到其不为 BUSY。
- **使用中断**：使能中断信号 KEYMNG_POST_DONE_INT，并通过中断函数进行处理。

31.8 编程示例

通过以上章节的介绍，可以总结出一套完整的芯片部署流程。本节主要介绍 HUK 生成器和密钥管理器在两种芯片典型启动过程中所需的操作步骤。

说明：
在本节中，将以简化的方式描述所有配置过程。有关详细信息，请参考第 31.7 小节。

31.8.1 芯片私钥部署阶段的软件流程示例

芯片首次启动和部署私钥对外部存储器进行加密/解密的过程：

1. 芯片进入下载模式。
2. **配置 HUK 生成器**：由于是第一次启动芯片，所以需要配置 HUK 生成器进入 HUK 生成模式：

(a) 配置 IDLE 阶段，根据表 31.6-1 配置 HUK 工作模式为 HUK 生成模式，然后置位 HUK_START。

(b) 进入 LOAD 阶段后，置位 HUK_CONTINUE。

(c) 进入 GAIN 阶段后，读取 huk_info，然后置位 HUK_CONTINUE。

(d) 等待 HUK 生成器回到 IDLE 阶段。

3. **配置 HUK 生成状态**：写入 EFUSE_KM_HUK_GEN_STATE，使其 1 的个数为奇数，以便实现下次启动时只能进入 HUK 恢复模式。
4. **部署密钥**：以 ECDHO 模式下的密钥部署为例：

(a) 配置 IDLE 阶段，配置 KEYMNG_KEY_PURPOSE 为 XTS，然后置位 KEYMNG_START。

(b) 进入 LOAD 阶段后，将本地生成的 $k1 * G$ 写入密钥管理器，然后置位 KEYMNG_CONTINUE。

(c) 进入 GAIN 阶段后，读取 key_info 和 $k2 * G$，然后置位 KEYMNG_CONTINUE。

(d) 等待密钥管理器回到 IDLE 阶段。

5. **上传密钥**：通过通讯接口上传 $k2 * G$ 给用户。
6. **生成并加密数据：**
   - 用户根据获得的 $k2 * G$ 和本地的 $k1$，生成协商私钥 $k1 * k2 * G$。
   - 对数据进行本地加密，最后将密文写入 flash。

31.8.2 芯片正常启动阶段的软件流程示例

当芯片部署完成后，正常启动的完整流程如下：

1. **芯片进入 SPI 启动模式：**
   芯片以 SPI 启动模式启动，首先进入未加密的程序段。
2. **配置 HUK 恢复模式：**
   由于 HUK 已经生成，下面配置 HUK 恢复模式：
   (a) 配置 IDLE 阶段，根据表 31.6-1 配置 HUK 生成器的工作模式为 HUK 恢复模式，然后置位 HUK_START。
   (b) 进入 LOAD 阶段后，将 huk_info 写入 HUK 生成器，然后配置 HUK_CONTINUE。
   (c) 进入 GAIN 阶段后，置位 HUK_CONTINUE。
   (d) 等待 HUK 生成器回到 IDLE 阶段。
3. **配置密钥管理器为私钥恢复模式：**
   (a) 配置 IDLE 阶段，配置 KEYMNG_KEY_PURPOSE 为 XTS，然后置位 KEYMNG_START。
   (b) 进入 LOAD 阶段后，将 key_info 写入密钥管理器，然后置位 KEYMNG_CONTINUE。
   (c) 进入 GAIN 阶段后，置位 KEYMNG_CONTINUE。
   (d) 等待密钥管理器回到 IDLE 阶段。
4. **进入加密程序段**：此时外部存储器中的的加/解密私钥已经生效，系统进入加密程序段。

31.9 中断

ESP32-C5 的密钥管理器和 HUK 生成器可以生成以下中断信号并发送给中断矩阵。

- KEYMNG_INTR
- HUK_INTR

密钥管理器和 HUK 生成器有多个中断源可生成以上中断信号。表 31.9-1 列出了密钥管理器和 HUK 生成器内部的中断源及其触发条件。

表 31.9-1. 密钥管理器和 HUK 生成器内部中断源

内部中断源	触发条件	中断信号
KEYMNG_PREP_DONE_INT	密钥管理器 PREP 阶段完成	KEYMNG_INTR
KEYMNG_PROC_DONE_INT	密钥管理器 PROC 阶段完成	KEYMNG_INTR
KEYMNG_POST_DONE_INT	密钥管理器 POST 阶段完成	KEYMNG_INTR

HUK_PREP_DONE_INT	HUK 生成器 PREP 阶段完成	HUK_INTR
HUK_PROC_DONE_INT	HUK 生成器 PROC 阶段完成	HUK_INTR
HUK_POST_DONE_INT	HUK 生成器 POST 阶段完成	HUK_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

31.10 内存块

本小节的所有地址均为相对于密钥管理器或 HUK 生成器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

表 31.10-1. 密钥管理器内存块

名称	描述	容量（字节）	起始地址	结束地址	访问
KEYMNG_ASSIST_INFO_MEM	存储辅助信息	64	0x0100	0x013F	Varies
KEYMNG_PUBLIC_INFO_MEM	存储公共信息	64	0x0140	0x017F	Varies
KEYMNG_SW_INIT_KEY_MEM	存储 sw_init_key	32	0x0180	0x019F	Varies

表 31.10-2. HUK 生成器内存块

名称	描述	容量（字节）	起始地址	结束地址	访问
HUK_INFO_MEM	存储 huk_info	384	0x0100	0x027F	Varies

31.11 寄存器列表

31.11.1 HUK 寄存器列表

本小节的所有地址均为相对于 HUK 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 Access 寄存器的访问类型，了解**访问**列缩写的含义。

名称	描述	地址	访问
时钟门控寄存器			
HUK_CLK_REG	时钟门控制寄存器	0x0004	R/W
中断寄存器			
HUK_INT_RAW_REG	原始中断寄存器（电平有效）	0x0008	RO/ WTC/ SS
HUK_INT_ST_REG	中断状态寄存器	0x000C	RO
HUK_INT_ENA_REG	中断使能寄存器	0x0010	R/W
HUK_INT_CLR_REG	中断清除寄存器	0x0014	WT

名称	描述	地址	访问
配置寄存器			
HUK_CONF_REG	配置寄存器	0x0020	R/W
控制寄存器			
HUK_START_REG	控制寄存器	0x0024	WT
状态寄存器			
HUK_STATE_REG	HUK 生成器状态寄存器	0x0028	RO
结果寄存器			
HUK_STATUS_REG	HUK 状态寄存器	0x0034	RO
版本寄存器			
HUK_DATE_REG	版本控制寄存器	0x00FC	R/W

31.11.2 密钥管理器寄存器列表

本小节的所有地址均为相对于密钥管理器基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

请查看章节 *Access 寄存器的访问类型*，了解**访问**列缩写的含义。

名称	描述	地址	访问
时钟门控寄存器			
KEYMNG_CLK_REG	时钟门控寄存器	0x0004	R/W
中断寄存器			
KEYMNG_INT_RAW_REG	原始中断寄存器（电平有效）	0x0008	RO/ WTC/ SS
KEYMNG_INT_ST_REG	中断状态寄存器	0x000C	RO
KEYMNG_INT_ENA_REG	中断使能寄存器	0x0010	R/W
KEYMNG_INT_CLR_REG	中断清除寄存器	0x0014	WT
静态配置寄存器			
KEYMNG_STATIC_REG	静态配置寄存器	0x0018	R/W
KEYMNG_LOCK_REG	静态配置锁寄存器	0x001C	R/W1
配置寄存器			
KEYMNG_CONF_REG	配置寄存器	0x0020	R/W
控制寄存器			
KEYMNG_START_REG	控制寄存器	0x0024	WT
状态寄存器			
KEYMNG_STATE_REG	状态寄存器	0x0028	RO
结果寄存器			
KEYMNG_RESULT_REG	运算结果寄存器	0x002C	RO/ SS
KEYMNG_KEY_VLD_REG	密钥状态寄存器	0x0030	RO
KEYMNG_HUK_VLD_REG	HUK 状态寄存器	0x0034	RO
版本寄存器			
KEYMNG_DATE_REG	版本控制寄存器	0x00FC	R/W

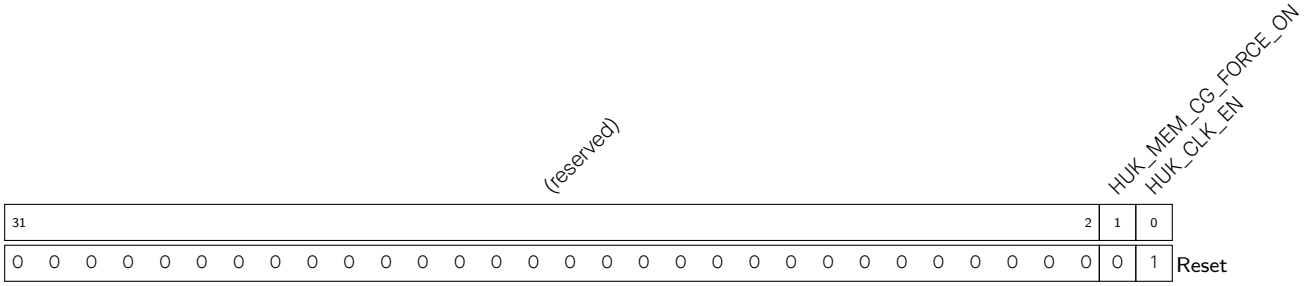
31.12 寄存器

31.12.1 HUK 寄存器

本小节的所有地址均为相对于 HUK 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 31.1. HUK_CLK_REG (0x0004)



HUK_CLK_EN 配置是否强制打开寄存器时钟门控。

0：不强制打开

1：强制打开

(R/W)

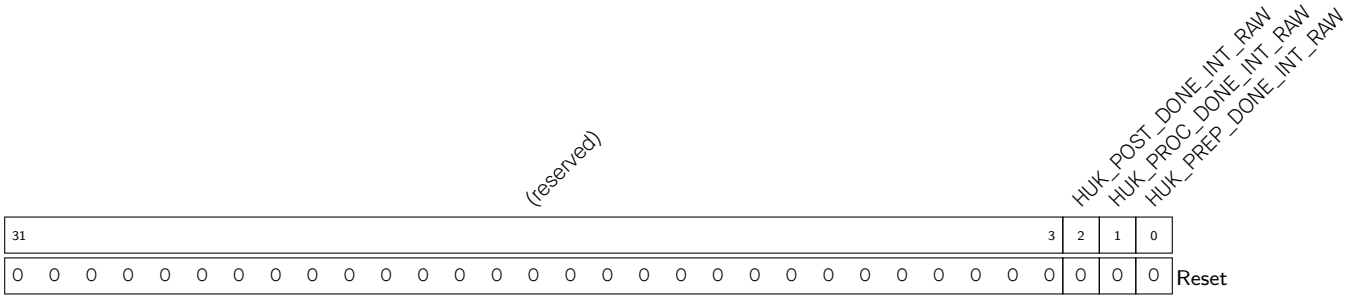
HUK_MEM_CG_FORCE_ON 配置是否强制打开存储器时钟门控。

0：不强制打开

1：强制打开

(R/W)

Register 31.2. HUK_INT_RAW_REG (0x0008)



HUK_PREP_DONE_INT_RAW HUK_PREP_DONE_INT 的原始中断状态。

(RO/WTC/SS)

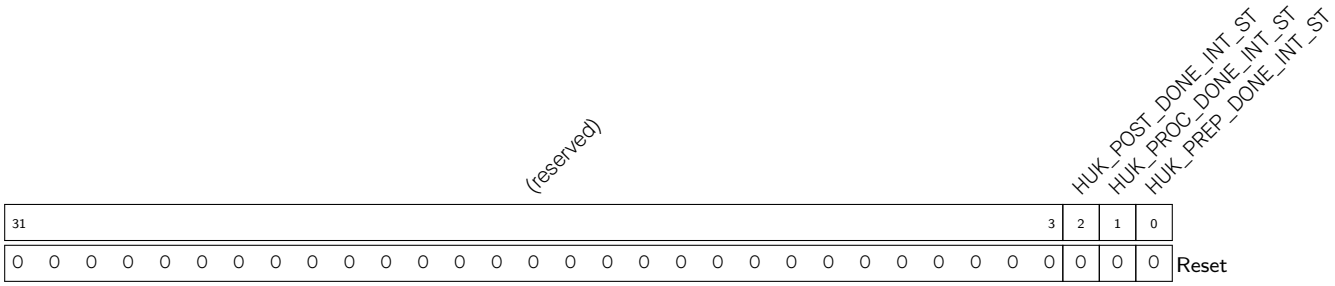
HUK_PROC_DONE_INT_RAW HUK_PROC_DONE_INT 的原始中断状态。

(RO/WTC/SS)

HUK_POST_DONE_INT_RAW HUK_POST_DONE_INT 的原始中断状态。

(RO/WTC/SS)

Register 31.3. HUK_INT_ST_REG (0x000C)

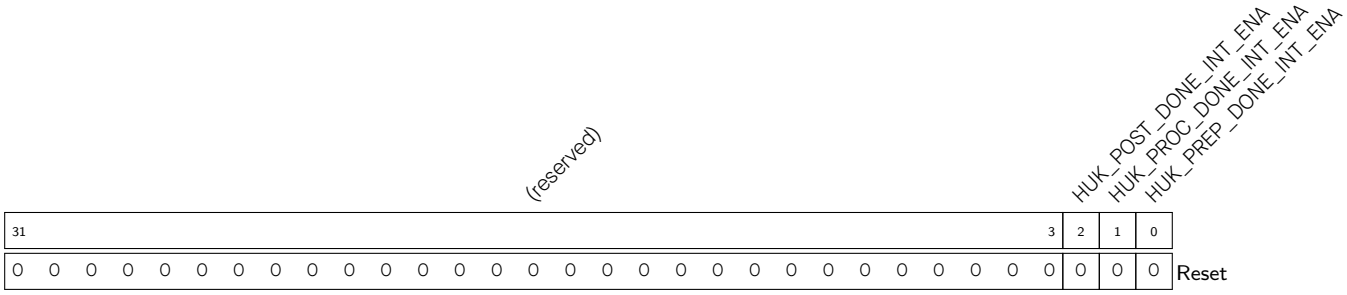


HUK_PREP_DONE_INT_ST HUK_PREP_DONE_INT 的屏蔽中断状态。
(RO)

HUK_PROC_DONE_INT_ST HUK_PROC_DONE_INT 的屏蔽中断状态。
(RO)

HUK_POST_DONE_INT_ST HUK_POST_DONE_INT 的屏蔽中断状态。
(RO)

Register 31.4. HUK_INT_ENA_REG (0x0010)

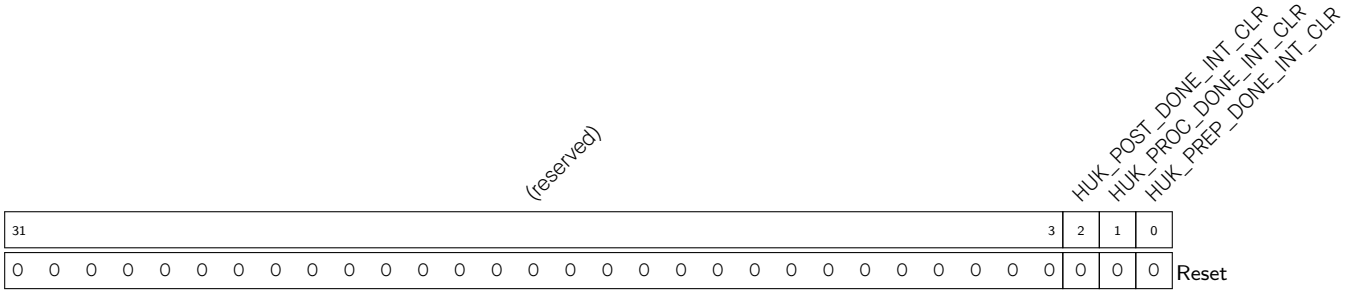


HUK_PREP_DONE_INT_ENA 写 1 使能 HUK_PREP_DONE_INT 中断。
(R/W)

HUK_PROC_DONE_INT_ENA 写 1 使能 HUK_PROC_DONE_INT 中断。
(R/W)

HUK_POST_DONE_INT_ENA 写 1 使能 HUK_POST_DONE_INT 中断。
(R/W)

Register 31.5. HUK_INT_CLR_REG (0x0014)

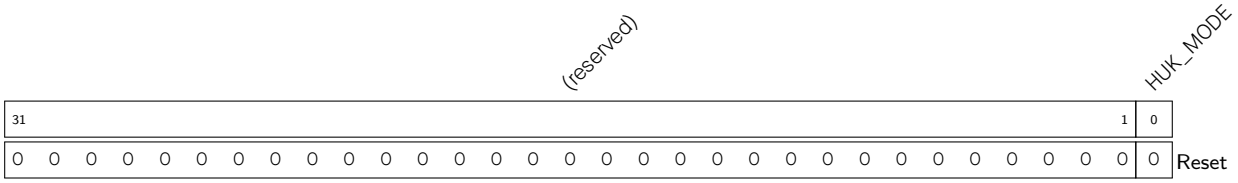


HUK_PREP_DONE_INT_CLR 写 1 清除 HUK_PREP_DONE_INT 中断。
(WT)

HUK_PROC_DONE_INT_CLR 写 1 清除 HUK_PROC_DONE_INT 中断。
(WT)

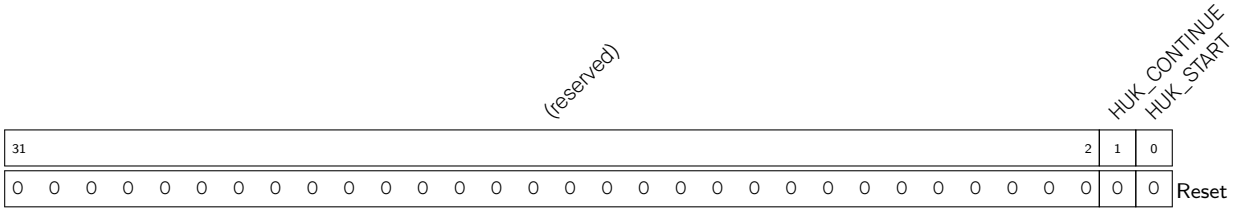
HUK_POST_DONE_INT_CLR 写 1 清除 HUK_POST_DONE_INT 中断。
(WT)

Register 31.6. HUK_CONF_REG (0x0020)



HUK_MODE 配置 HUK 模式。
0：HUK 恢复模式
1：HUK 生成模式
(R/W)

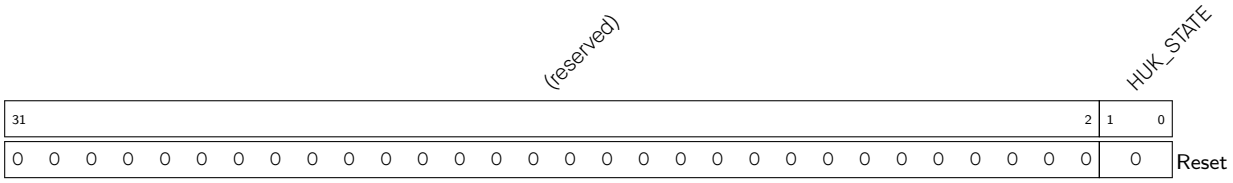
Register 31.7. HUK_START_REG (0x0024)



HUK_START 写入 1 以在 IDLE 状态下启动 HUK 生成器。
(WT)

HUK_CONTINUE 写入 1 继续 LOAD/GAIN 状态下的 HUK 生成器操作。
(WT)

Register 31.8. HUK_STATE_REG (0x0028)



HUK_STATE 表示 HUK 生成器的状态。

0: IDLE

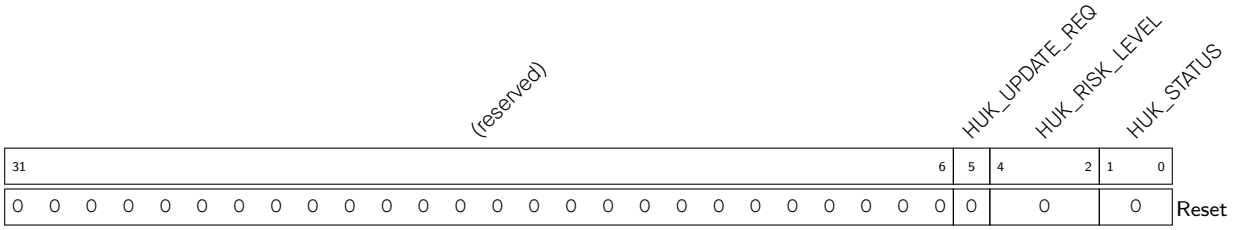
1: LOAD

2: GAIN

3: BUSY

(RO)

Register 31.9. HUK_STATUS_REG (0x0034)



HUK_STATUS 代表 HUK 生成状态。

0：未生成 HUK。

1：HUK 已生成且有效。

2：HUK 已生成但无效。

3：保留。

(RO)

HUK_RISK_LEVEL 代表 HUK 的风险等级。

0~6：风险等级越高，PUF SRAM 中的错误比特越多。

7：等级错误，HUK 无效。

(RO)

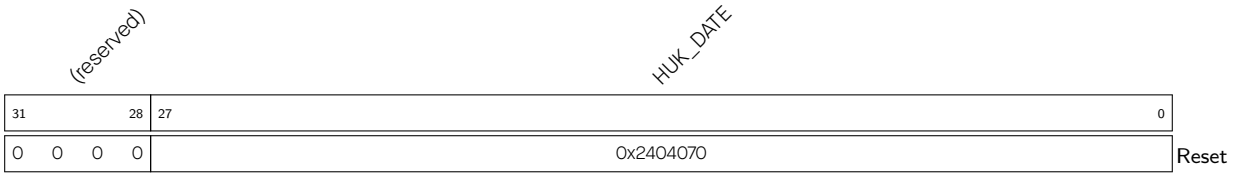
HUK_UPDATE_REQ 表示 huk_info 的更新要求。

0：用户可根据风险等级选择更新 huk_info。

1：HUK info 已过期，用户需更新 huk_info。

(RO)

Register 31.10. HUK_DATE_REG (0x00FC)



HUK_DATE 版本控制寄存器。

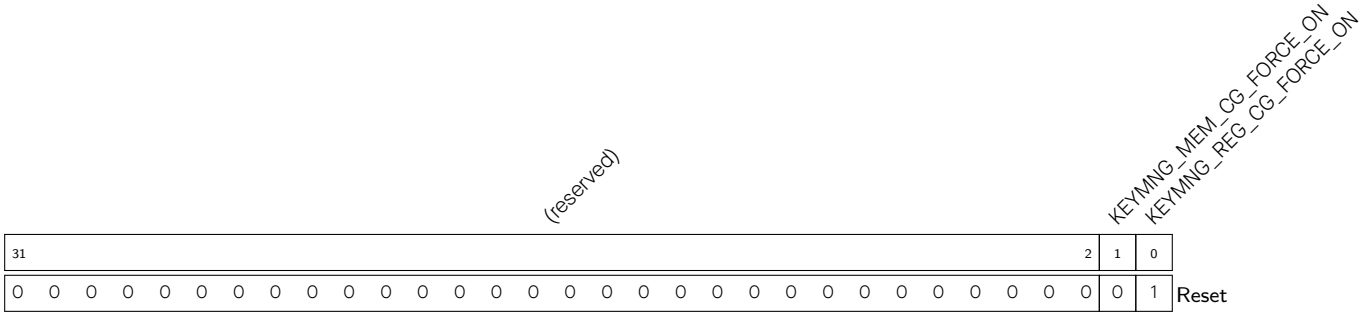
(R/W)

31.12.2 密钥管理器寄存器

本小节的所有地址均为相对于密钥管理器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 31.11. KEYMNG_CLK_REG (0x0004)



KEYMNG_REG_CG_FORCE_ON 配置是否强制打开寄存器时钟门控。

0：不强制打开

1：强制打开

(R/W)

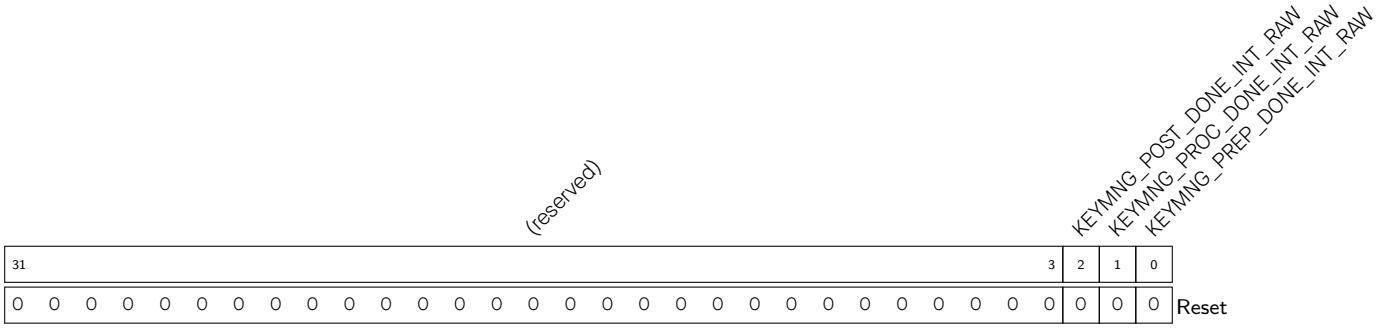
KEYMNG_MEM_CG_FORCE_ON 配置是否强制打开存储器时钟门控。

0：不强制打开

1：强制打开

(R/W)

Register 31.12. KEYMNG_INT_RAW_REG (0x0008)

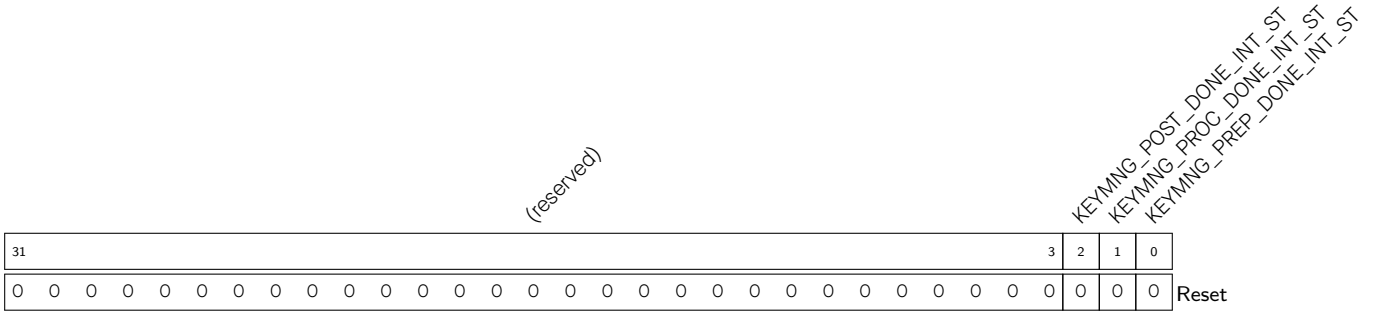


KEYMNG_PREP_DONE_INT_RAW KEYMNG_PREP_DONE_INT 的原始中断状态。
(RO/WTC/SS)

KEYMNG_PROC_DONE_INT_RAW KEYMNG_PROC_DONE_INT 的原始中断状态。
(RO/WTC/SS)

KEYMNG_POST_DONE_INT_RAW KEYMNG_POST_DONE_INT 的原始中断状态。
(RO/WTC/SS)

Register 31.13. KEYMNG_INT_ST_REG (0x000C)

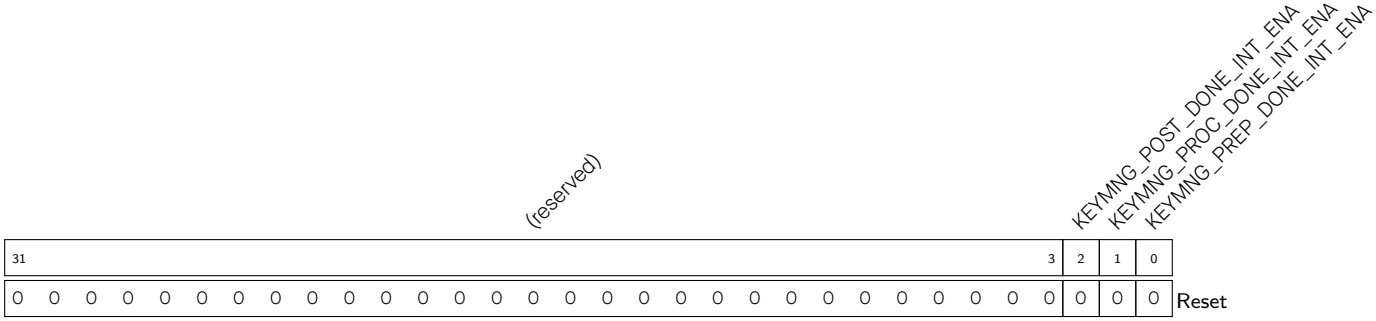


KEYMNG_PREP_DONE_INT_ST KEYMNG_PREP_DONE_INT 的屏蔽中断状态。
(RO)

KEYMNG_PROC_DONE_INT_ST KEYMNG_PROC_DONE_INT 的屏蔽中断状态。
(RO)

KEYMNG_POST_DONE_INT_ST KEYMNG_POST_DONE_INT 的屏蔽中断状态。
(RO)

Register 31.14. KEYMNG_INT_ENA_REG (0x0010)

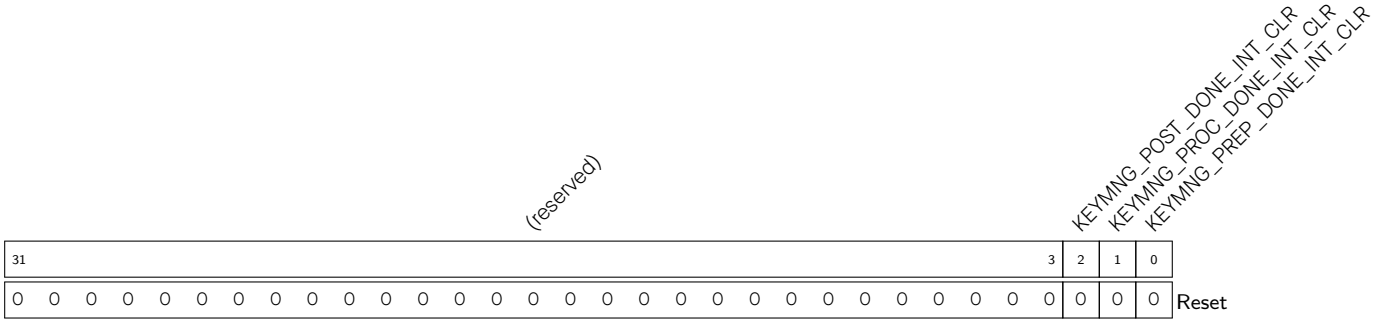


KEYMNG_PREP_DONE_INT_ENA 写 1 使能 KEYMNG_PREP_DONE_INT 中断。(R/W)

KEYMNG_PROC_DONE_INT_ENA 写 1 使能 KEYMNG_PROC_DONE_INT 中断。(R/W)

KEYMNG_POST_DONE_INT_ENA 写 1 使能 KEYMNG_POST_DONE_INT 中断。(R/W)

Register 31.15. KEYMNG_INT_CLR_REG (0x0014)

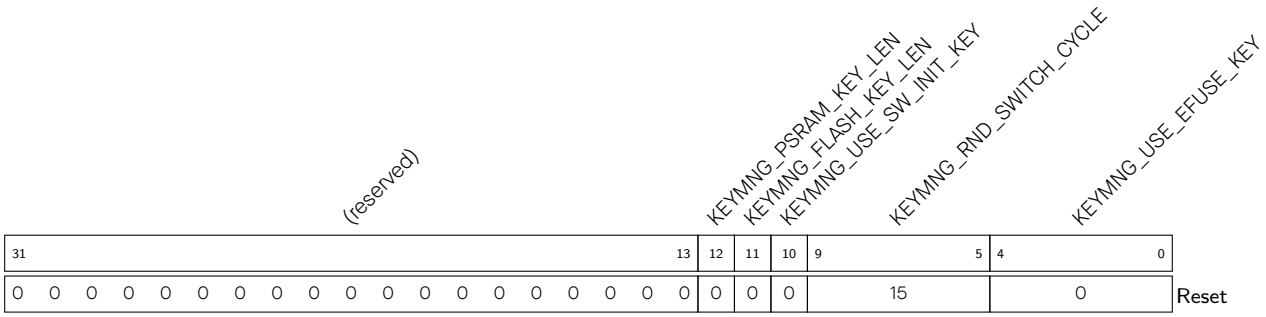


KEYMNG_PREP_DONE_INT_CLR 写 1 清除 KEYMNG_PREP_DONE_INT 中断。(WT)

KEYMNG_PROC_DONE_INT_CLR 写 1 清除 KEYMNG_PROC_DONE_INT 中断。(WT)

KEYMNG_POST_DONE_INT_CLR 写 1 清除 KEYMNG_POST_DONE_INT 中断。(WT)

Register 31.16. KEYMNG_STATIC_REG (0x0018)



KEYMNG_USE_EFUSE_KEY 配置是否使用 eFuse 密钥，而不使用密钥管理器部署的密钥。

- Bit[0]：配置 ECDSA_DS 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[0] 为 0 时有效。
- Bit[1]：配置 flash 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[1] 为 0 时有效。
- Bit[2]：配置 HMAC 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[2] 为 0 时有效。
- Bit[3]：配置 RSA_DS_KEY 使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[3] 为 0 时有效。
- Bit[4]：配置 PSRAM 密钥使用 eFuse 密钥
  0：无效
  1：使用 eFuse 密钥，仅当 EFUSE_FORCE_USE_KEY_MANAGER_KEY 中的 Bit[4] 为 0 时有效。

(R/W)

KEYMNG_RND_SWITCH_CYCLE 配置密钥管理器切换随机数的周期，单位为密钥管理器时钟周期。仅当 EFUSE_KM_RND_SWITCH_CYCLE 为 0 时生效。
注意：此切换周期建议配置为 TRNG 模块的时钟周期所对应的密钥管理器时钟周期数。
(R/W)

KEYMNG_USE_SW_INIT_KEY 配置使用软件配置的 sw_init_key 而不是 EFUSE_KM_INIT_KEY，仅当 EFUSE_FORCE_DISABLE_SW_INIT_KEY 为 0 时有效。
0：使用 EFUSE_KM_INIT_KEY
1：使用软件写入的初始密钥，即 sw_init_key
(R/W)

见下页

Register 31.16. KEYMNG_STATIC_REG (0x0018)

接上页

KEYMNG_FLASH_KEY_LEN 配置 flash 加密的密钥长度。

0：使用 xts-aes-128

1：使用 xts-aes-256

(R/W)

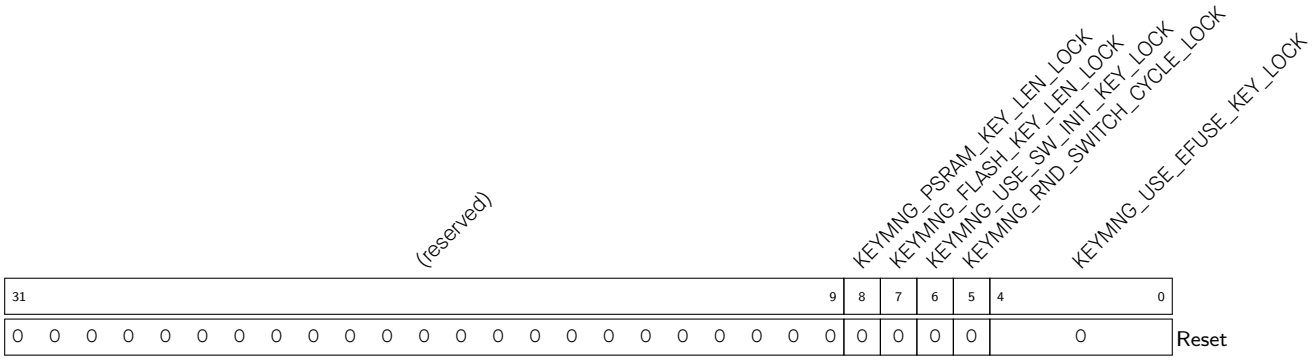
KEYMNG_PSRAM_KEY_LEN 配置 PSRAM 加密的密钥长度。

0：使用 xts-aes-128

1：使用 xts-aes-256

(R/W)

Register 31.17. KEYMNG_LOCK_REG (0x001C)



KEYMNG_USE_EFUSE_KEY_LOCK 写 1 锁定 KEYMNG_USE_EFUSE_KEY。

每一位分别锁定 KEYMNG_USE_EFUSE_KEY 的相应位。

(R/W1)

KEYMNG_RND_SWITCH_CYCLE_LOCK 写 1 锁定 KEYMNG_RND_SWITCH_CYCLE。

(R/W1)

KEYMNG_USE_SW_INIT_KEY_LOCK 写 1 锁定 KEYMNG_USE_SW_INIT_KEY。

(R/W1)

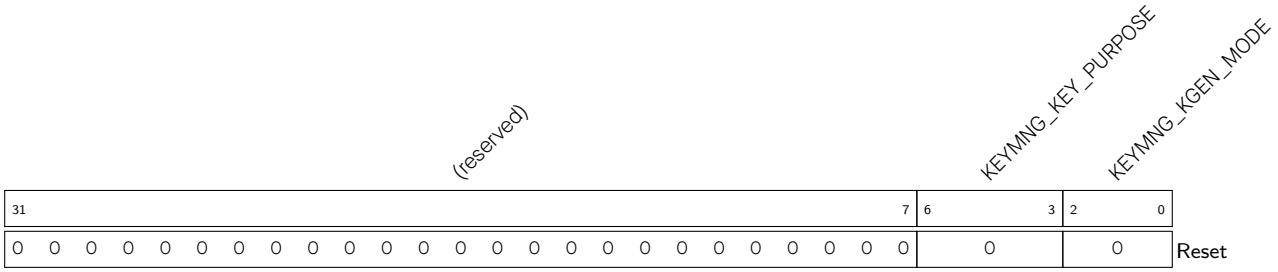
KEYMNG_FLASH_KEY_LEN_LOCK 写 1 锁定 KEYMNG_FLASH_KEY_LEN。

(R/W1)

KEYMNG_PSRAM_KEY_LEN_LOCK 写 1 锁定 KEYMNG_PSRAM_KEY_LEN。

(R/W1)

Register 31.18. KEYMNG_CONF_REG (0x0020)



KEYMNG_KGEN_MODE 配置密钥管理器的部署模式。

0：随机部署模式

1：AES 部署模式

2：ECDH0 部署模式

3：ECDH1 部署模式

4：私钥恢复模式

5：导出 key_info 模式

6~7：保留

(R/W)

KEYMNG_KEY_PURPOSE 配置密钥用途

1: ecdsa_key_192

2: ecdsa_key_256

3: flash_256_1_key

4: flash_256_2_key

5: flash_128_key

6: hmac_key

7: ds_key

8: psram_256_1_key

9: psram_256_2_key

10: psram_128_key

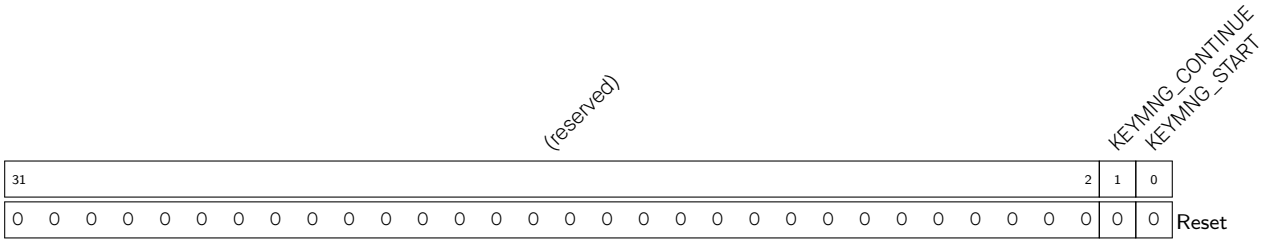
11: ecdsa_key_384_l

12: ecdsa_key_384_h

其他值：保留

(R/W)

Register 31.19. KEYMNG_START_REG (0x0024)



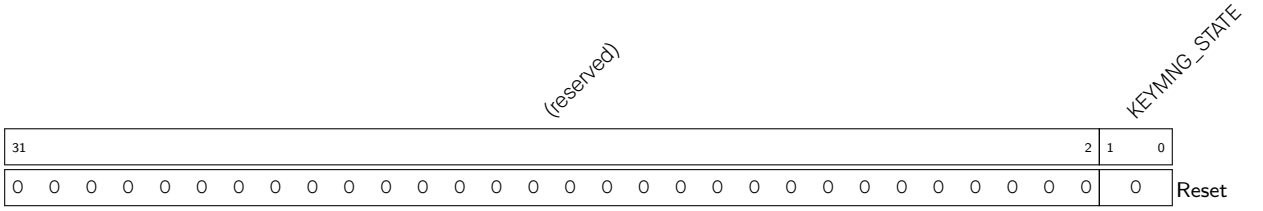
KEYMNG_START 写入 1 以在 IDLE 状态下启动密钥管理器。

(WT)

KEYMNG_CONTINUE 写入 1 继续 LOAD/GAIN 状态下的密钥管理器操作。

(WT)

Register 31.20. KEYMNG_STATE_REG (0x0028)



KEYMNG_STATE 表示密钥管理器的状态。

0: IDLE

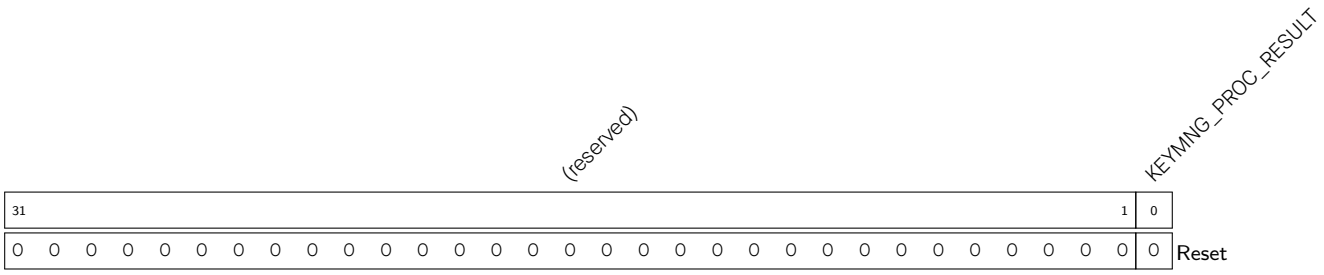
1: LOAD

2: GAIN

3: BUSY

(RO)

Register 31.21. KEYMNG_RESULT_REG (0x002C)



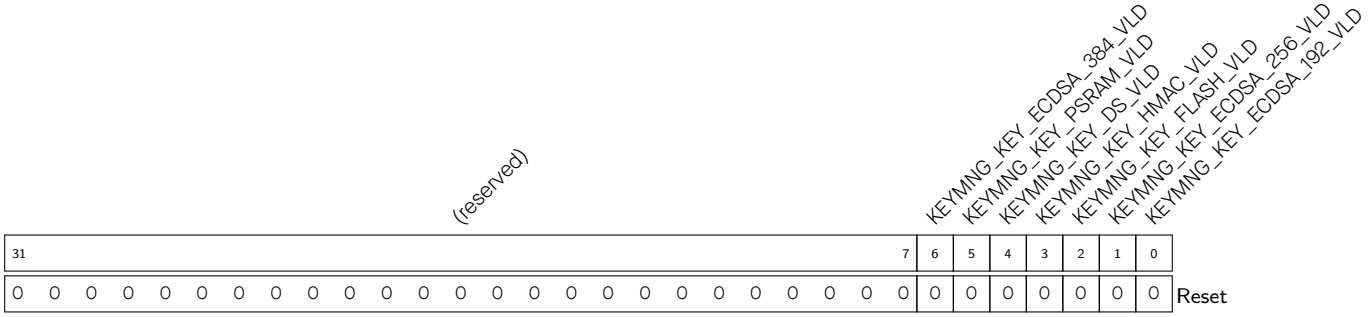
KEYMNG_PROC_RESULT 表示密钥管理器的处理结果，仅当密钥管理器处理完成后才有效。

0：密钥管理器程序失败。

1：密钥管理器程序成功。

(RO/SS)

Register 31.22. KEYMNG_KEY_VLD_REG (0x0030)



KEYMNG_KEY_ECDSA_192_VLD 表示 key_ecdsa_192 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_ECDSA_256_VLD 表示 key_ecdsa_256 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_FLASH_VLD 表示 key_flash 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_HMAC_VLD 表示 key_hmac 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_DS_VLD 表示 key_ds 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

KEYMNG_KEY_PSRAM_VLD 表示 key_psram 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

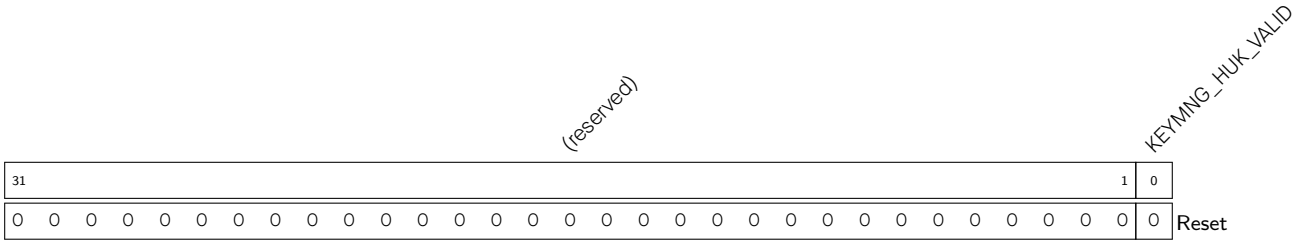
KEYMNG_KEY_ECDSA_384_VLD 表示 key_ecdsa_384 密钥的状态。

0：密钥尚未部署。

1：密钥已正确部署。

(RO)

Register 31.23. KEYMNG_HUK_VLD_REG (0x0034)



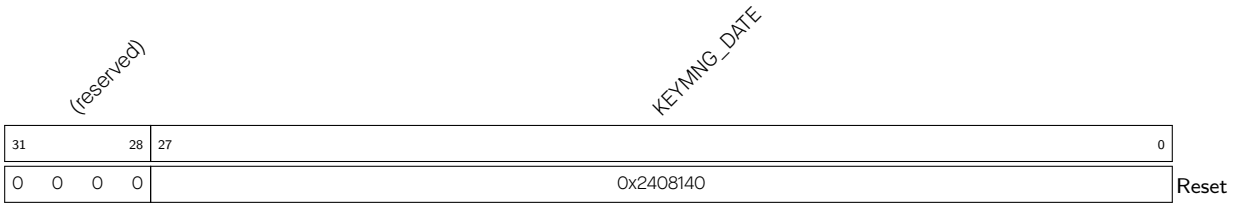
KEYMNG_HUK_VALID 代表 HUK 状态。

0：HUK 无效

1：HUK 有效

(RO)

Register 31.24. KEYMNG_DATE_REG (0x00FC)



KEYMNG_DATE 版本控制寄存器。

(R/W)

第 V 卷

通讯接口

该部分涉及系统的连接性，介绍各种带有通讯接口的模块，如 I2C、I2S、SPI、UART、USB 等，同时还包括其他用于遥控、电机控制、LED 控制等功能的信号生成接口。

第 32 章

UART 控制器 (UART)

32.1 概述

UART 是一种以字符为导向、用于实现设备间通信的异步传输接口。异步传输的意思是不需要在发送数据上添加时钟信息。这也要求发送端和接收端的速率、停止位、奇偶校验位等都要相同，通信才能成功。

一个典型的 UART 帧开始于一个起始位，紧接着是有效数据，然后是奇偶校验位（可有可无），最后是停止位。ESP32-C5 上的 UART 控制器支持多种字符长度和停止位。另外，控制器还支持软硬件流控和 GDMA，可以实现高速的数据传输。开发者可以使用多个 UART 端口，同时又能保证很少的软件开销。

ESP32-C5 芯片中共有三个 UART 控制器，包含两个正常功能的 UART 和一个满足低功耗需求的 LP UART。UART 控制器可以兼容不同的 UART 设备。另外，UART 还可以用于红外数据交换 (IrDA) 或 RS485 调制解调器。本文以 UART*n* 指代两个 UART 控制器，*n* 为 0、1。LP UART 是 UART 的功能剪裁版本，且单独控制一组寄存器，通过表 32.2-1 可以区分 UART 和 LP UART 的特性。

32.2 主要特性

UART 控制器的特性以及与 LP UART 的特性区分见下表：

表 32.2-1. UART 和 LP UART 特性区分

UART 特性	LP UART 特性
可编程收发波特率，最大为 5 MBaud	
一个 UART 的发送通道和接收通道分别有 128 x 8 bit RAM 存储空间	LP UART 的发送 FIFO 和接收 FIFO 共用 20 x 8 bit 存储空间
全双工异步通信	
数据位（5 到 8 位）	
停止位（1、1.5 或 2 位）	
奇偶校验位	
AT_CMD 特殊字符检测	
RS485 协议	—
IrDA 协议	—
GDMA 高速数据通信	—
接收超时	
UART 唤醒模式	
软件流控和硬件流控	

见下页

表 32.2-1 – 接上页

UART 特性	LP_UART 特性
三个可分频的时钟源： 1. XTAL_CLK 2. RC_FAST_CLK 3. PLL_F80M_CLK	三个可分频的时钟源： 1. RC_FAST_CLK 2. XTAL_DIV_CLK 3. PLL_F8M_CLK

下文主要针对正常功能的 UART 进行叙述。

32.3 UART 架构

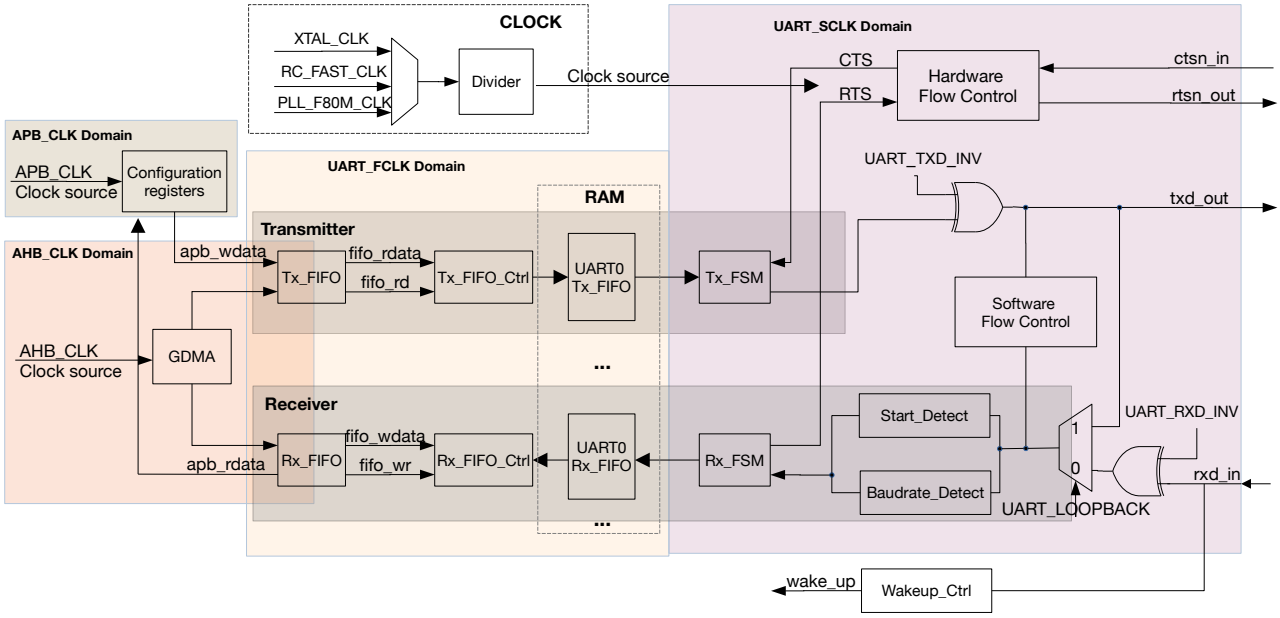


图 32.3-1. UART 基本架构图

图 32.3-1 为 UART 基本架构图。UART 模块工作在 4 个时钟域，分别为 APB_CLK、AHB_CLK、UART_SCLK、UART_FCLK。其中 APB_CLK 与 AHB_CLK 时钟关系为同步不同频（APB_CLK 由 AHB_CLK 分频得到），UART_SCLK 与 UART_FCLK 时钟关系为同步不同频（UART_SCLK 由 UART_FCLK 分频得到）。UART_FCLK 有三个时钟源：PLL_F80M_CLK、RC_FAST_CLK 以及晶振时钟 XTAL_CLK（详情请参考章节 9 复位和时钟）。可以通过配置 PCR_UART*n*_SCLK_SEL 来选择 UART_FCLK 时钟源，之后使用分频器对时钟源进行分频，然后产生时钟信号 UART_SCLK。分频系数的整数部分由 PCR_UART*n*_SCLK_DIV_NUM 配置，小数部分的分母由 PCR_UART*n*_SCLK_DIV_A 配置，小数部分的分子由 PCR_UART*n*_SCLK_DIV_B 配置。支持的分频范围为 1 ~ 256。只有正常功能的 UART 包含此分频器，LP UART 不包含。

UART 控制器可以分为两个功能块：发送块和接收块。

发送块包含一个发送 FIFO 用于缓存待发送的数据。软件可以通过 APB 总线向 Tx_FIFO 写数据，也可以通过 GDMA 将数据搬入 Tx_FIFO。Tx_FIFO_Ctrl 用于控制 Tx_FIFO 的读写过程，当 Tx_FIFO 非空时，Tx_FSM 通过 Tx_FIFO_Ctrl 读取数据，并将数据按照配置的帧格式转化成比特流。比特流输出信号 txd_out 可以通过配置 UART_TXD_INV 寄存器实现取反功能。

接收块包含一个接收 FIFO 用于缓存待处理的数据。输入比特流 rxd_in 可以输入到 UART 控制器。可以通过

UART_RXD_INV 寄存器实现取反。Baudrate_Detect 通过检测最小比特流输入信号的脉宽来测量输入信号的波特率。Start_Detect 用于检测数据的起始位，当检测到起始位之后，Rx_FSM 通过 Rx_FIFO_Ctrl 将帧解析后的数据存入 Rx_FIFO 中。软件可以通过 APB 总线读取 Rx_FIFO 中的数据，也可以使用 GDMA 进行数据接收。

HW_Flow_Ctrl 通过标准 UART RTS 和 CTS（rtsn_out 和 ctsn_in）流控信号来控制 rxd_in 和 txd_out 的数据流。SW_Flow_Ctrl 通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来进行数据流的控制。当 UART 处于 Light-sleep 模式（详情请参考章节 13 低功耗管理）时，可以通过四种不同的方式产生 wake_up 信号给 PMU 模块，由 PMU 来唤醒 ESP32-C5 芯片。具体唤醒方式请参考章节 32.4.8。

32.4 功能描述

32.4.1 时钟与复位

UART 为异步外设。其寄存器配置模块工作在 APB_CLK 时钟域，TX/RX FIFO 作为跨时钟域模块工作在 AHB_CLK 和 UART_FCLK 时钟域。UART RAM 控制单元工作在 UART_FCLK 时钟域，控制 UART 发送与接收的模块工作在 UART_SCLK 时钟域，即 UART Core 模块的时钟域。

在 UART_SCLK 时钟频率满足生成波特率的需求下，可通过预分频使 UART Core 模块工作在较小的时钟频率，从而减小 UART 外设的功耗。通常情况下，UART Core 模块时钟小于 APB_CLK 时钟，并且在满足 UART 波特率的情况下，UART Core 时钟分频系数可以配置到最大值。UART 也支持 UART Core 模块时钟大于 APB_CLK 时钟，此时，UART Core 模块时钟最大为 APB_CLK 的 3 倍。另外，UART Core 的 TX/RX 时钟可以被单独控制。置位 UART_TX_SCLK_EN 使能 UART Core 的 TX 时钟；置位 UART_RX_SCLK_EN 使能 UART Core 的 RX 时钟。

为确保配置寄存器的值成功从 APB_CLK 时钟域同步到 UART_SCLK 时钟域，寄存器配置需要遵循一定的流程，详情请参考章节 32.6。

对整个 UART 的复位，需要遵循如下配置流程：

- 将 PCR_UART*n*_CLK_EN 置 1 打开 UART*n* Core 时钟。
- 向 PCR_UART*n*_RST_EN 位写 1。
- 将 PCR_UART*n*_RST_EN 位清 0。

32.4.2 UART FIFO

UART 控制器的发送块和接收块分别使用一块 128 x 8 位 RAM 存储空间，并且分别通过一个 4 x 8 位的异步 FIFO 接口访问对应的 RAM。发送块和接收块的存储空间和异步 FIFO 接口相互独立，不可复用。

UART 的 Tx_FIFO 可以通过置位 UART_TXFIFO_RST 来复位，Rx_FIFO 可以通过置位 UART_RXFIFO_RST 来复位。

对于 TX FIFO，可以通过 APB 总线或 GDMA 向其写入数据，硬件 Tx_FSM 自动从其中读取数据，数据将按照配置的帧格式转换成比特流；对于 RX FIFO，可以通过 APB 总线或 GDMA 读取其中的数据，并存储到内存，硬件 Rx_FSM 将接收到的比特流转换成字节并写入 RX FIFO。两个 UART 共享同一个 GDMA 通道。

配置 UART_TXFIFO_EMPTY_THRHD 可以设置 Tx_FIFO 空信号阈值，当存储在 Tx_FIFO 中的数据量小于 UART_TXFIFO_EMPTY_THRHD 时会产生中断 UART_TXFIFO_EMPTY_INT；配置 UART_RXFIFO_FULL_THRHD 可以设置 Rx_FIFO 满信号阈值，当储存在 Rx_FIFO 中的数据量大于或等于 UART_RXFIFO_FULL_THRHD 会产生中断 UART_RXFIFO_FULL_INT。另外，当 Rx_FIFO 中储存的数据量超过其能存储的最大值时，会产生 UART_RXFIFO_OVF_INT 中断。

UART*n* 可以通过寄存器 UART_FIFO_REG 访问 FIFO。您可以写 UART_RXFIFO_RD_BYTE 将数据存入 TX FIFO，也可以读 UART_RXFIFO_RD_BYTE 获取 RX FIFO 中的数据。

32.4.3 波特率产生与检测

32.4.3.1 波特率产生

在 UART 发送或接收数据之前，需要配置寄存器来设置波特率。波特率发生器主要通过对输入时钟源的分频来实现，支持小数分频。UART_CLKDIV_SYNC_REG 将分频系数分成两个部分：UART_CLKDIV 用于配置整数部分，UART_CLKDIV_FRAG 用于配置小数部分。在输入时钟为 80 MHz 的情况下，UART 能支持的最大波特率为 5 MBaud。

波特率分频器系数为

$$UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}$$

也就是说，最终波特率为

$$\frac{INPUT\_FREQ}{UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}}$$

其中，INPUT_FREQ 为 UART Core 时钟。例如，若 UART_CLKDIV = 694，UART_CLKDIV_FRAG = 7，则分频系数为

$$694 + \frac{7}{16} = 694.4375$$

UART_CLKDIV_FRAG 为 0 时，分频器为整数分频，每 UART_CLKDIV 个输入脉冲都会产生一个输出脉冲。

UART_CLKDIV_FRAG 不为 0 时，分频器为小数分频，输出波特率脉冲不完全统一。如图 32.4-1 所示，每 16 个输出脉冲，波特率发生器分频 (UART_CLKDIV + 1) 个输入脉冲或 UART_CLKDIV 个输入脉冲。分频 (UART_CLKDIV + 1) 个输入脉冲产生 UART_CLKDIV_FRAG 个输出脉冲，分频 UART_CLKDIV 个输入脉冲产生剩余的 (16 - UART_CLKDIV_FRAG) 个输出脉冲。

如图 32.4-1 所示，输出脉冲相互交错，使得输出时序更加统一。

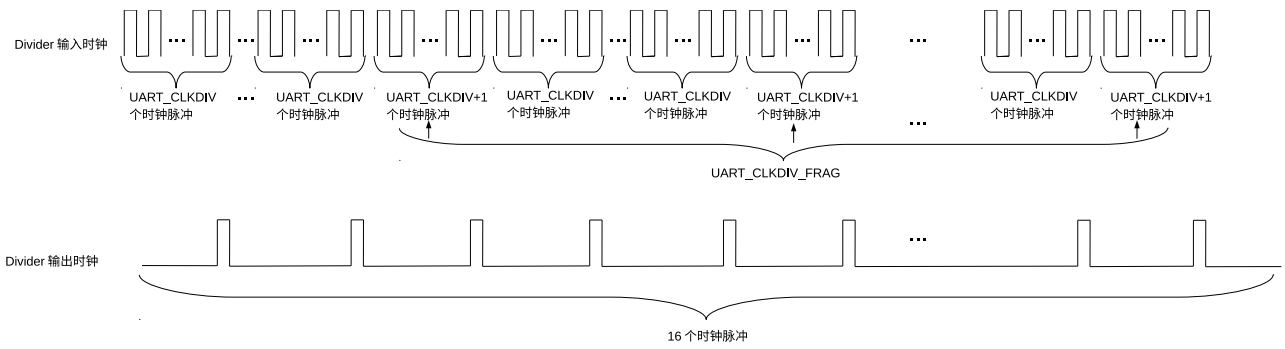


图 32.4-1. UART 控制器分频

为了支持 IrDA（详情见章节 32.4.7），IrDA 小数分频器会产生 16 × UART_CLKDIV_SYNC_REG 分频的时钟用于 IrDA 数据传输。产生 IrDA 数据传输时钟的小数分频器原理与上述小数分频器一样，取UART_CLKDIV/16 作为分频值的整数部分，取 UART_CLKDIV 的低 4 比特作为小数部分。

32.4.3.2 波特率检测

置位 UART_AUTOBAUD_EN 可以开启 UART 波特率自检测功能。图 32.4-2 中的 Baudrate_Detect 可以滤除信号脉宽小于 UART_GLITCH_FILT 的噪声。

在 UART 双方进行通信之前，可以通过发送随机数据，让具有波特率检测功能的数据接收方进行波特率分析。UART_LOWPULSE_MIN_CNT 存储了最小低电平脉冲宽度，UART_HIGHPULSE_MIN_CNT 存储了最小高电平脉冲宽度，UART_POSEDGE_MIN_CNT 存储了两个上升沿之间的最小脉冲宽度，UART_NEGEDGE_MIN_CNT 存储了两个下降沿之间最小的脉冲宽度。软件可以通过读取这四个寄存器获取发送方的波特率。

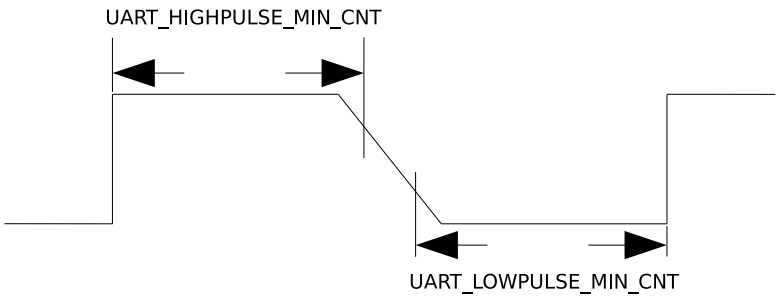


图 32.4-2. UART 信号下降沿较差时序图

波特率的计算分为三种情况：

1. 正常情况下，为防止因亚稳态在上升沿或下降沿附近采样数据错误而导致 UART_LOWPULSE_MIN_CNT 或者 UART_HIGHPULSE_MIN_CNT 不准确，单比特脉冲宽度可以通过将这两个值相加取平均消除误差。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

2. 对于 UART 信号的下降沿信号比较差的情况，如图32.4-2 所示，这时通过取 UART_LOWPULSE_MIN_CNT 与 UART_HIGHPULSE_MIN_CNT 的和平均得到的值不准确，可以通过 UART_POSEDGE_MIN_CNT 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_POSEDGE\_MIN\_CNT} + 1)/2}$$

3. 对于 UART 信号的上升沿信号比较差的情况，可以通过 UART_NEGEDGE_MIN_CNT 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

32.4.4 UART 数据帧

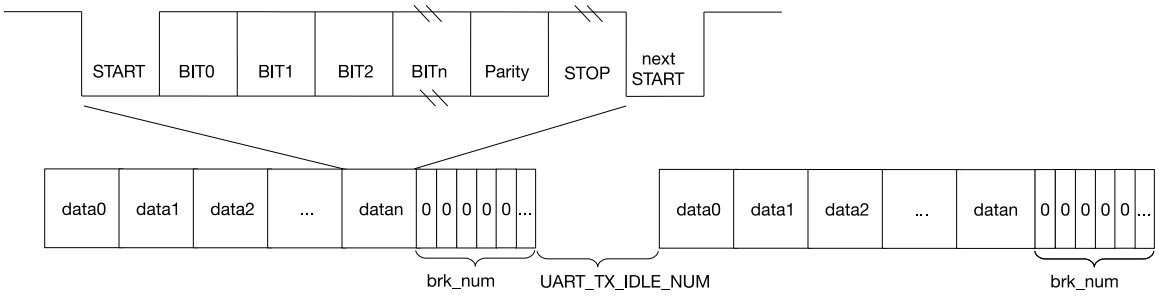


图 32.4-3. UART 数据帧结构

图 32.4-3 所示为基本数据帧格式，数据帧从起始位开始，以停止位结束。起始位占用 1 位，停止位可以通过配置 UART_STOP_BIT_NUM 实现 1、1.5、2 位宽（RS485 模式下可增加转换延时，详见章节 32.4.6.2）。起始位

为低电平，停止位为高电平。

数据位宽 (BIT0 ~ BITn) 为 5 ~ 8 位，可以通过 UART_BIT_NUM 进行配置。当置位 UART_PARITY_EN 时，数据帧会在数据之后添加一位奇偶校验位。UART_PARITY 用于选择奇校验或是偶校验。当接收器检测到输入数据的校验位错误时会产生 UART_PARITY_ERR_INT 中断，输入数据仍会存入 Rx_FIFO。当接收器检测到数据数据帧格式错误时会产生 UART_FRM_ERR_INT 中断，默认情况下，输入数据会被存入 Rx_FIFO。

Tx_FIFO 中数据都发送完成后会产生 UART_TX_DONE_INT 中断。置位 UART_TXD_BRK 时，Tx_FIFO 中数据发送完成后，发送端会进入终止状态 (break condition)，继续发送几个连续的特殊数据帧 NULL。在 NULL 数据帧发送完成后，TX 数据线输出为低电平。NULL 数据帧的数量可由 UART_TX_BRK_NUM 进行配置。发送器发送完所有的 NULL 数据帧之后会产生 UART_TX_BRK_DONE_INT 中断。数据帧之间可以通过配置 UART_TX_IDLE_NUM 保持最小间隔时间。当一帧数据之后的空闲时间大于等于 UART_TX_IDLE_NUM 寄存器的配置值时则产生 UART_TX_BRK_IDLE_DONE_INT 中断。

在传输一个 NULL 数据帧所需的时间内，接收端 RX 数据线若一直检测到低电平，接收端会检测为终止状态，并触发 UART_BRK_DET_INT 中断表示终止状态已结束。

接收端通过 UART_RXFIFO_TOUT_INT 中断检测总线状态。接收端接收到至少一个字节数据后，总线处于空闲状态超过 UART_RX_TOUT_THRHD 位时间时，触发 UART_RXFIFO_TOUT_INT 中断。您可用此中断检测发送端是否已经发送所有数据。

32.4.5 AT_CMD 字符格式

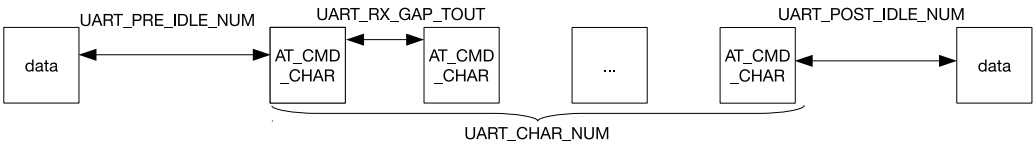


图 32.4-4. AT_CMD 字符格式

图 32.4-4 为一种特殊的 AT_CMD 字符格式。当接收器连续收到 AT_CMD_CHAR 字符且字符之间满足如下条件时将会产生 UART_AT_CMD_CHAR_DET_INT 中断。

- 接收到的第一个 AT_CMD_CHAR 与上一个非 AT_CMD_CHAR 之间间隔至少 UART_PRE_IDLE_NUM 个波特率周期。
- AT_CMD_CHAR 字符之间间隔小于 UART_RX_GAP_TOUT 个波特率周期。
- 接收的 AT_CMD_CHAR 字符个数必须大于等于 UART_CHAR_NUM。
- 接收到的最后一个 AT_CMD_CHAR 字符与下一个非 AT_CMD_CHAR 之间间隔至少 UART_POST_IDLE_NUM 个波特率周期。

注意：由于 AT_CMD_CHAR 字符之间间隔小于 UART_RX_GAP_TOUT 个波特率周期的限制，建议用户使用该功能时，配置的 APB 时钟频率不要低于 8 MHz。

32.4.6 RS485

两个 UART 控制器支持 RS485 通讯模式，RS485 因使用差分信号传输数据，相比于 RS232 具有更远的传输距离及更高的传输速率。RS485 有两线半双工及四线全双工两种选择，UART 模块采用两线半双工模式，并支持侦听总线的功能。

32.4.6.1 驱动控制

如图 32.4-5 所示，RS485 两线 multidrop 系统中，需要一个外部 RS485 收发器实现单端信号与差分信号的转换。RS485 收发器包括一个驱动器与一个接收器。当 UART 不作为发送器时，通过关闭驱动器来断开与差分传输线的连接。DE 为 1 时，使能驱动器；DE 为 0 关闭驱动器。

UART 接收端通过外部接收器将差分信号转为单端信号。RE 作为接收器的使能控制信号，RE 为 0，使能接收器；RE 为 1，关闭接收器。如果 RE 被配置为 0，从而允许 UART 保持侦听总线上的数据，包括 UART 发送的数据。

DE 信号的控制分为软件控制和硬件控制两种方法。为减少软件的开销，DE 信号采用硬件来控制。图 32.4-5 所示，DE 与 UART 的 dtrn_out 相连（详见 32.4.9.1 小节）。

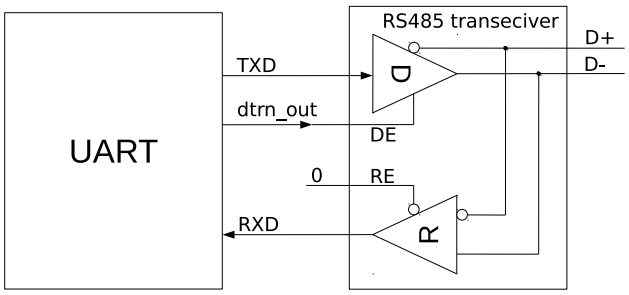


图 32.4-5. RS485 模式驱动控制结构图

32.4.6.2 转换延时

默认情况下，UART 处于接收状态。当从发送转为接收状态时，为保证发送数据被稳定接收，RS485 协议要求在发送停止位之后增加一个波特率的转换延时。UART 发送模块支持在起始位之前或在停止位之后增加一个波特率的延时。置位 UART_DLO_EN，在起始位之前增加一个波特率周期延时；置位 UART_DL1_EN，在停止位之后增加一个波特率周期延时。

32.4.6.3 总线侦听

RS485 两线 multidrop 系统中，当外部 RS485 收发器的 RE 被配置为 0 时，UART 支持侦听总线。默认情况下，不允许 UART 在发送数据时接收数据。置位 UART_RS485TX_RX_EN，允许在发送数据时接收数据，配合图 32.4-5 中外部 RS485 收发器的配置，UART 保持侦听传输总线。另外，默认情况下，不允许 UART 在接收数据时发送数据。置位 UART_RS485RXBY_TX_EN，允许在接收数据时发送数据。

UART 支持侦听 UART 发送的数据。UART 处于发送状态下，当侦听到 UART 发送的数据与 UART 接收的数据冲突时，触发 UART_RS485_CLASH_INT 中断；侦听到发送的数据帧错误时，触发 UART_RS485_FRM_ERR_INT 中断；侦听到发送数据校验位错误时，触发 UART_RS485_PARITY_ERR_INT 中断。

32.4.7 IrDA

IrDA 数据协议由物理层，链路接入层和链路管理层三个基本层协议组成。UART 实现了其物理层协议。在 IrDA 编码模式下，支持最大信号速率到 115.2 Kbit/s，即 SIR 模式。如图 32.4-6 所示，IrDA 编码器将来自 UART 的非归零编码 (NRZ) 信号采用反向归零编码 (RZI) 并输出给外部驱动和红外 LED，用 3/16 Bit Time 的脉宽调制信号表示逻辑 “0”，用低电平表示逻辑 “1”。IrDA 解码器接收来自红外接收器的信号并输出为 UART 的 NRZ 编码。一般情况下，接收端信号空闲时为高电平，编码器输出校验位与解码器输入校验位相反。当检测到低脉冲表示接收到开始信号。

IrDA 使能时，一个位被划分为 16 个时钟周期，在其第 9、10、11 个时钟周期中，当需要发送的比特为 0 时，IrDA 输出为高。

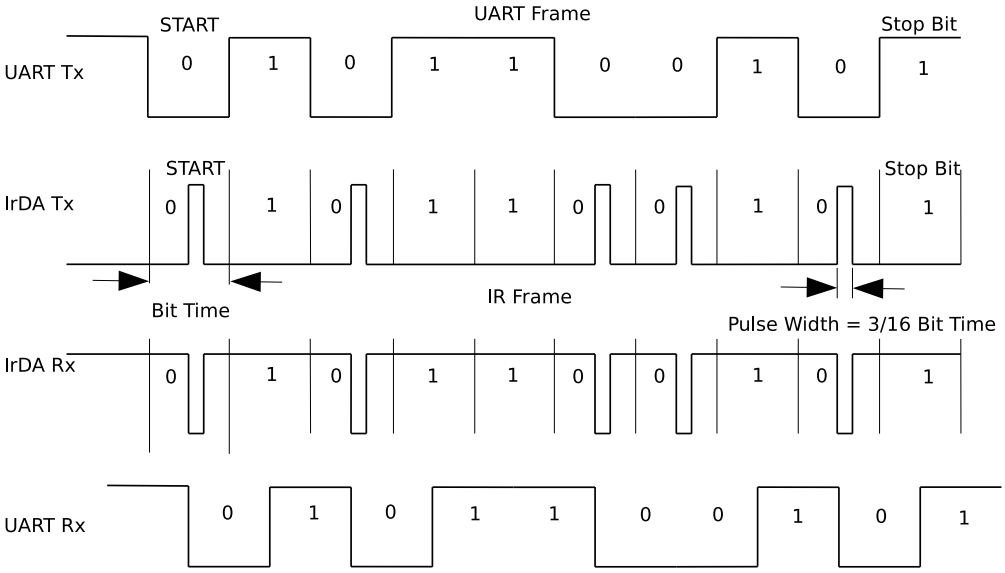


图 32.4-6. SIR 模式编解码时序图

IrDA 为半双工传输协议，不允许同时进行收发。如图32.4-7 所示，置位 UART_IRDA_EN 使能 IrDA 功能。置位 UART_IRDA_TX_EN（置 1）使能 IrDA 发送数据，这时不允许 IrDA 接收数据；复位 UART_IRDA_TX_EN（清 0）使能 IrDA 接收数据，这时不允许 IrDA 发送数据。

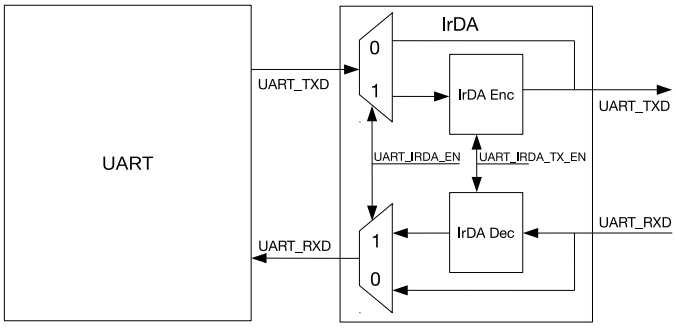


图 32.4-7. IrDA 编解码结构图

32.4.8 唤醒

UART 支持唤醒功能。当 UART 处于 Light-sleep 模式时，可以通过四种不同的方式产生 wake_up 信号给 PMU 模块，由 PMU 来唤醒芯片。

- UART_WK_MODE_SEL=0，当所有的时钟都关闭时，此时可以通过使 RXD 翻转若干周期，当上升沿个数大于等于 UART_ACTIVE_THRESHOLD +3 时唤醒芯片。
- UART_WK_MODE_SEL=1，由于 UART Core 时钟保持工作，因此 UART RX 仍然可以接收数据并将数据暂存在 Rx FIFO 中。当 Rx FIFO 中的数据量超过 UART_RX_WAKE_UP_THRHD 时，可以将芯片从 Light-sleep 中唤醒。
- UART_WK_MODE_SEL=2，当 UART RX 监测到起始位后，唤醒芯片。

- UART_WK_MODE_SEL=3，当 UART RX 接收到特定字符序列后，唤醒芯片。用户通过配置 UART_WK_CHAR0、UART_WK_CHAR1、UART_WK_CHAR2、UART_WK_CHAR3、UART_WK_CHAR4 自定义唤醒字符。UART 监测到由 CHAR0/CHAR1/CHAR2/CHAR3/CHAR4 组成的字符序列流之后唤醒芯片。可以通过配置 UART_CHAR_NUM 和 UART_WK_CHAR_MASK 来设置不同的字符序列。如下表所示，对于最后一个配置，UART 将依次检测 CHAR0 ~ CHAR4。

表 32.4-1. UART_CHAR_WAKEUP 模式配置

UART_CHAR_NAME	UART_WP_CHAR_MASK	检测字符序列
1	0xF	CHAR4
2	0x7	CHAR3/CHAR4
3	0x3	CHAR2/CHAR3/CHAR4
4	0x1	CHAR1/CHAR2/CHAR3/CHAR4
5	0x0	CHAR0/CHAR1/CHAR2/CHAR3/CHAR4

使用 UART 唤醒之后，需要通过在 Active 模式下向 UART 传输数据或是复位整个 UART 模块清除 wake_up 信号，否则下一次唤醒所需的上升沿个数将减少。

32.4.9 流控

UART 控制器有两种数据流控方式：硬件流控和软件流控。硬件流控主要通过输出信号 rtsn_out 以及输入信号 ctsn_in 进行数据流控制。软件流控主要通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来实现数据流控功能。

32.4.9.1 硬件流控

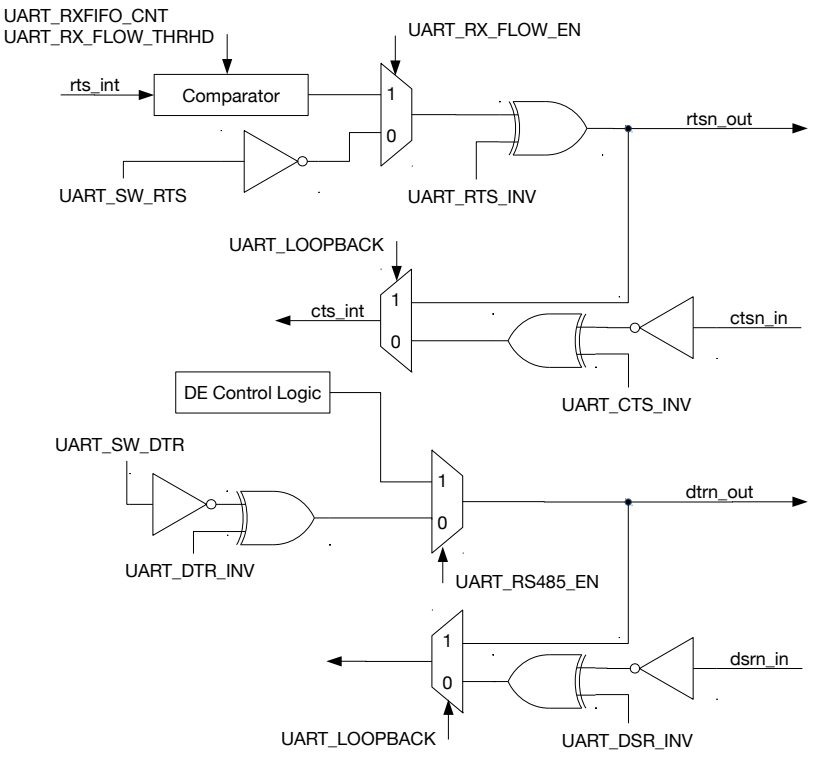


图 32.4-8. 硬件流控图

图 32.4-8 为 UART 硬件流控图。硬件流控的控制信号为输出信号 rtsn_out 及输入信号 ctsn_in。图 32.4-9 为两个 UART 之间硬件流控信号连接图。记 ESP32-C5 UART 为 IUO，External UART 为 EUO，下文将使用这两个标记来区分两个 UART。输出信号 rtsn_out (IUO) 为低电平表示允许对方 (EUO) 发送数据，rtsn_out (IUO) 为高电平表示通知对方 (EUO) 中止数据发送直到 rtsn_out (IUO) 恢复低电平。rtsn_out 输出信号的控制有两种方式。

- 软件控制：将 UART_RX_FLOW_EN 置 0 进入该模式。该模式下通过软件配置 UART_SW_RTS 改变 rtsn_out 的电平。
- 硬件控制：将 UART_RX_FLOW_EN 置 1 进入该模式。该模式下硬件会当 Rx_FIFO 中的数据大于 UART_RX_FLOW_THRHD 时拉高 rtsn_out 的电平。

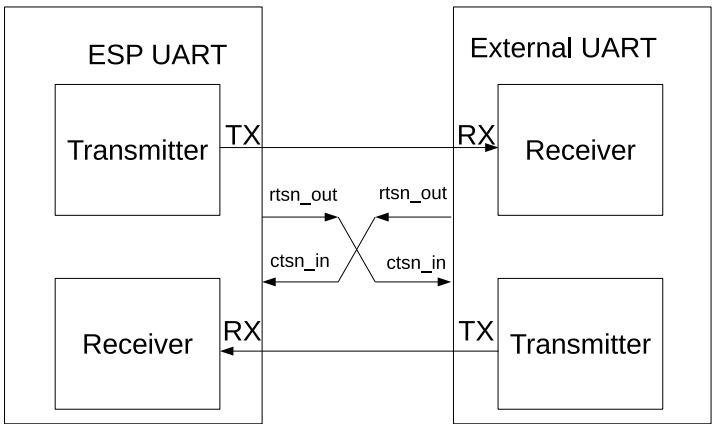


图 32.4-9. 硬件流控信号连接图

输入信号 ctsn_in (IUO) 为低电平表示允许发送端 (IUO) 发送数据；ctsn_in (IUO) 为高电平表示禁止发送端 (IUO) 发送数据。当 UART 检测到输入信号 ctsn_in (IUO) 的沿变化时会产生 UART_CTS_CHG_INT 中断。

UART 发送设备 (IUO) 输出信号 dtrn_out 为高电平表示发送数据已经准备完毕，处于可用状态。dtrn_out 通过配置寄存器 UART_SW_DTR 产生。UART 接收设备 (IUO) 在检测到输入信号 dsrn_in 的沿变化时会产生 UART_DSR_CHG_INT 中断。软件在检测到中断后，通过读取 UART_DSRN 可以获取 dsrn_in 的输入信号电平，UART_DSRN 为高电平时，表示对方设备 (EUO) 处于可用状态。

对于 RS485 两线 multidrop 系统，使用 dtrn_out 来收发转换。置位 UART_RS485_EN 使能 RS485 功能，dtrn_out 由硬件产生。数据开始发送时，dtrn_out 拉高，使能外部驱动器；数据最后一位发送完成后，dtrn_out 拉低，关闭外部驱动器。注意，当使能停止位之后增加一个波特率延时时，dtrn_out 会在延时结束后才拉低。

置位 UART_LOOPBACK 即开启 UART 的回环测试功能。此时 UART 的输出信号 txd_out 和其输入信号 rxd_in 相连，rtsn_out 和 ctsn_in 相连，dtrn_out 和 dsrn_out 相连。当接收的数据与发送的数据相同时表明 UART 能够正常发送和接收数据。

32.4.9.2 软件流控

软件流控不使用硬件的 CTS/RTS 控制线，而是在发送数据流中嵌入 XON/XOFF 字符来通知对方是否可以使用数据发送来实现流控。将 UART_SW_FLOW_CON_EN 置 1 使能软件流控。

在使用软件流控后，硬件会自动检测接收数据流中是否有 XON/XOFF 字符，在检测到相应的字符后会产生 UART_SW_XOFF_INT 或 UART_SW_XON_INT 中断。在检测到接收数据流中有 XOFF 字符后，发送器将会在发送完当前数据后停止发送；在检测到接收数据流中有 XON 字符后，将会使能发送器发送数据。另外，软件可以

通过置位 UART_FORCE_XOFF 来强制发送器停止发送数据，发送器会在发送完当前字节后停止发送；也可以通过置位 UART_FORCE_XON 来使能发送器发送数据。

软件可以根据 Rx_FIFO 中剩余空间大小决定流控字符的发送。置位 UART_SEND_XOFF，发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 UART_XOFF_CHAR 配置；置位 UART_SEND_XON，发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 UART_XON_CHAR 配置。另外，当 UART 接收 FIFO 中的数据量超过 UART_XOFF_THRESHOLD 时，硬件会置位 UART_SEND_XOFF，UART 发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 UART_XOFF_CHAR 配置。当 UART 接收 FIFO 中的数据量小于 UART_XON_THRESHOLD 时，硬件会置位 UART_SEND_XON，UART 发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 UART_XON_CHAR 配置。

当 UART 工作在全双工模式时，如果 UART RX 接收到 XOFF，此时尽管 UART RX 接受的数据超过阈值，UART TX 将被禁止发送包括 XOFF 在内的任何数据。为了避免该种情况导致的软流控死锁或者 overflow，可以使能 UART_XON_XOFF_STILL_SEND，此时允许 UART TX 在被禁止发送时，仍然可以发送一个 XOFF 字符。

32.4.10 GDMA 模式

ESP32-C5 中的两个 UART 接口通过通用主机控制器接口 (UHCI) 共用 1 组 GDMA TX/RX 通道。在 GDMA 模式下，支持对 HCI 协议数据包的解析 (decoder) 及数据包封装 (encoder)。UHCI_UART_SEL 字段用于选择哪个串口占用 GDMA 通道。

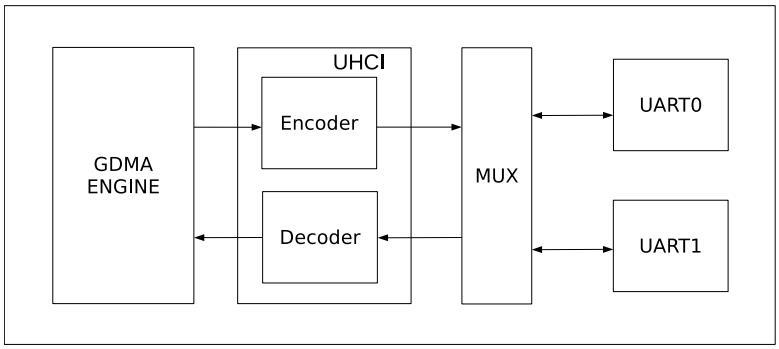


图 32.4-10. GDMA 模式数据传输

图 32.4-10 为 GDMA 方式数据传输图。在 GDMA Rx 通道接收数据前，软件将接收链表准备好。AHB_DMA_INLINK_ADDR_CH*n* 用于指向第一个接收链表描述符。置位 AHB_DMA_INLINK_START_CH*n* 之后，通用主机控制器接口 (UHCI) 会将 UART 接收到的数据传送给 decoder。经过 decoder 解析之后的数据在 GDMA 通道的控制下存入接收链表指定的 RAM 空间。

在 GDMA Tx 通道发送数据前，软件需要将发送链表和发送数据准备好，AHB_DMA_OUTLINK_ADDR_CH*n* 用于指向第一个发送链表描述符。置位 AHB_DMA_OUTLINK_START_CH*n* 之后，GDMA 引擎即从链表中指定的 RAM 地址读取数据，并通过 encoder 进行数据包封装，然后经 UART 的发送模块串行发送出去。

HCI 的数据包格式为（分隔符 + 数据 + 分隔符）。encoder 用于在数据前后加上分隔符，并将数据中和分隔符一样的数据替换为特殊字符。decoder 用于去除数据包前后分隔符，并将数据中的特殊字符替换为分隔符。数据前后的分隔符可以有连续多个。分隔符可由 UHCI_SEPER_CHAR 进行配置，默认值为 0xC0。数据中与分隔符一样的数据可以用 UHCI_ESC_SEQ0_CHAR0（默认为 0xDB）和 UHCI_ESC_SEQ0_CHAR1（默认为 0xDD）进行替换。当数据全部发送完成后，会产生 GDMA_OUT_TOTAL_EOF_CH*n*_INT 中断。当数据接收完成后，会产生 GDMA_IN_SUC_EOF_CH*n*_INT 中断。

32.5 中断

ESP32-C5 的 UART*n* 和 UHCI 可以生成以下中断信号并发送给中断矩阵。

- UART*n*_INTR
- UHCI_INTR

这些中断信号分别由 UART*n* 和 UHCI 的内部中断源生成。

UART*n* 中断源如下所列：

- UART_AT_CMD_CHAR_DET_INT：当接收器检测到 AT_CMD 字符时触发此中断。
- UART_RS485_CLASH_INT：在 RS485 模式下检测到发送器和接收器之间的冲突时触发此中断。
- UART_RS485_FRM_ERR_INT：在 RS485 模式下检测到发送块发送的数据帧错误时触发此中断。
- UART_RS485_PARITY_ERR_INT：在 RS485 模式下检测到发送块发送的数据校验位错误时触发此中断。
- UART_TX_DONE_INT：当发送器发送完 FIFO 中的所有数据时触发此中断。
- UART_TX_BRK_IDLE_DONE_INT：当发送器在最后一个数据发送后保持了最短的间隔时间时触发此中断。
- UART_TX_BRK_DONE_INT：当发送 FIFO 中的数据发送完之后发送器完成了发送 NULL 则触发此中断。
- UART_GLITCH_DET_INT：当接收器在起始位的中点处检测到毛刺时触发此中断。
- UART_SW_XOFF_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XOFF 字符时触发此中断。
- UART_SW_XON_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XON 字符时触发此中断。
- UART_RXFIFO_TOUT_INT：当接收器至少已经接收一个字节的数据，且之后在 UART_RX_TOUT_THRHD 位的时间内，总线一直处于 idle 状态将触发此中断。
- UART_BRK_DET_INT：当接收器在停止位之后检测到一个 NULL（即传输一个 NULL 的时间内保持逻辑低电平）时触发此中断。
- UART_CTS_CHG_INT：当接收器检测到 CTSn 信号的沿变化时触发此中断。
- UART_DSR_CHG_INT：当接收器检测到 DSRn 信号的沿变化时触发此中断。
- UART_RXFIFO_OVF_INT：当接收器接收到的数据量多于 FIFO 的存储量时触发此中断。
- UART_FRM_ERR_INT：当接收器检测到数据帧错误时触发此中断。
- UART_PARITY_ERR_INT：当接收器检测到校验位错误时触发此中断。
- UART_TXFIFO_EMPTY_INT：当发送 FIFO 中的数据量少于 UART_TXFIFO_EMPTY_THRHD 所指定的值时触发此中断。
- UART_RXFIFO_FULL_INT：当接收器接收到的数据多于 UART_RXFIFO_FULL_THRHD 所指定的值时触发此中断。
- UART_WAKEUP_INT：UART 被唤醒时产生此中断。

UHCI 中断源如下所列：

- UHCI_APP_CTRL1_INT：软件置位 UHCI_APP_CTRL1_INT_RAW 时触发此中断。
- UHCI_APP_CTRL0_INT：软件置位 UHCI_APP_CTRL0_INT_RAW 时触发此中断。
- UHCI_OUTLINK_EOF_ERR_INT：当检测到发送链表描述符中的 EOF 有错误时触发此中断。

- UHCI_SEND_A_REG_Q_INT：当使用 always_send 发送一串短包，UHCI 发送了短包后触发此中断。
- UHCI_SEND_S_REG_Q_INT：当使用 single_send 发送一串短包，UHCI 发送了短包后触发此中断。
- UHCI_TX_HUNG_INT：当 UHCI 利用 GDMA Tx 通道从 RAM 中读取数据的时间过长时触发此中断。
- UHCI_RX_HUNG_INT：当 UHCI 利用 GDMA Rx 通道接收数据的时间过长时触发此中断。
- UHCI_TX_START_INT：当检测到分隔符时触发此中断。
- UHCI_RX_START_INT：当分隔符已发送时触发此中断。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 32.7 寄存器列表。

32.6 编程流程

32.6.1 寄存器类型

UART 的所有寄存器都处于 APB_CLK 时钟域。

UART 上的配置寄存器分为两类：第一类寄存器作用在 APB/AHB 时钟域，该类寄存器配置后不需要额外的操作；第二类寄存器作用在 UART_SCLK 时钟域，因此需要进行跨时钟域处理。在配置完该类寄存器后，需要通过写寄存器 UART_REG_UPDATE 使配置的值同步到 Core 时钟域。当同步完成后，硬件会将 UART_REG_UPDATE 自动清零。推荐用户在配置需要同步的寄存器之后，检查 UART_REG_UPDATE 是否为 0，确保上一次的同步操作完成之后再执行接下来的寄存器配置。

为了方便用户区分这两类寄存器，章节 32.7 中将需要做跨时钟域操作的寄存器放在一起，该类寄存器名称附带 _SYNC 后缀。不需要做跨时钟域操作的寄存器放在一起，该类寄存器名称不带 _SYNC 后缀。

32.6.2 具体步骤

图32.6-1 显示了 UART 模块的编程流程。主要包括：初始化、寄存器配置、启动 UART TX/RX 和数据传输结束。

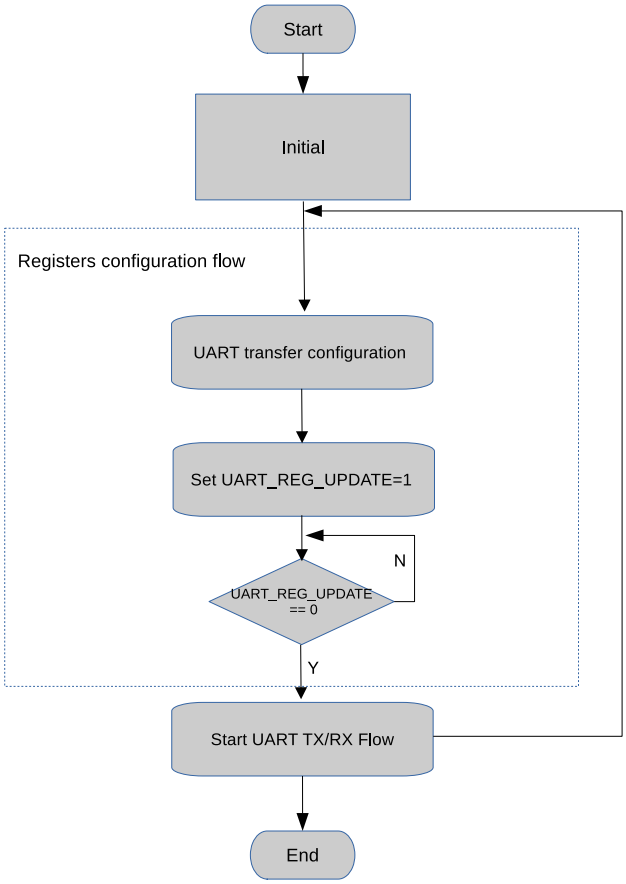


图 32.6-1. UART 编程流程

32.6.2.1 UART*n* 模块初始化

UART*n* 模块初始化流程如下：

- 将 PCR_UART*n*_CLK_EN 置 1，打开 UART*n* 的 APB_CLK；
- 向 PCR_UART*n*_RST_EN 写 1；
- 将 PCR_UART*n*_RST_EN 清 0。

32.6.2.2 UART*n* 通信配置

UART*n* 通信配置流程如下：

- 等待 UART_REG_UPDATE 为 0，确保上一次同步已经完成；
- 配置 PCR_UART*n*_SCLK_SEL 选择时钟源；
- 配置 PCR_UART*n*_SCLK_DIV_NUM、PCR_UART*n*_SCLK_DIV_A、PCR_UART*n*_SCLK_DIV_B 设置预分频器系数；
- 配置 UART_CLKDIV、UART_CLKDIV_FRAG 设置发送波特率；

- 配置 UART_BIT_NUM 设置数据长度；
- 配置 UART_PARITY_EN、UART_PARITY 设置奇偶校验；
- 可选步骤，根据应用不同存在差异；
- 向 UART_REG_UPDATE 写 1，将配置的值同步到 Core 时钟域。

32.6.2.3 启动 UART*n*

启动 UART*n* TX 发送数据：

- 配置 UART_TXFIFO_EMPTY_THRHD，设置 TX FIFO 空阈值；
- 对 UART_TXFIFO_EMPTY_INT_ENA 置 0，关闭 UART_TXFIFO_EMPTY_INT 中断；
- 向 UART_RXFIFO_RD_BYTE 写入需要发送的数据；
- 置位 UART_TXFIFO_EMPTY_INT_CLR，清除 UART_TXFIFO_EMPTY_INT 中断；
- 置位 UART_TXFIFO_EMPTY_INT_ENA，使能 UART_TXFIFO_EMPTY_INT 中断；
- 检测 UART_TXFIFO_EMPTY_INT_ST，等待发送数据结束。

启动 UART*n* RX 数据接收：

- 配置 UART_RXFIFO_FULL_THRHD，设置 RX FIFO 满阈值；
- 置位 UART_RXFIFO_FULL_INT_ENA，使能 UART_RXFIFO_FULL_INT 中断；
- 检测 UART_RXFIFO_FULL_INT_ST，等待 RX FIFO 接收数据满；
- 通过读 UART_RXFIFO_RD_BYTE，从 RX FIFO 中读出数据，并可通过 UART_RXFIFO_CNT 获得当前 RX FIFO 中的接收数据量。

32.7 寄存器列表

32.7.1 UART 寄存器列表

本小节的所有地址均为相对于 UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 配置寄存器			
UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART 阈值和分配配置	0x0064	R/W
UART 中断寄存器			
UART_INT_RAW_REG	原始中断状态	0x0004	R/WTC/SS
UART_INT_ST_REG	屏蔽中断状态	0x0008	RO
UART_INT_ENA_REG	中断使能位	0x000C	R/W
UART_INT_CLR_REG	中断清除位	0x0010	WT
配置寄存器			
UART_CLKDIV_SYNC_REG	时钟分频配置	0x0014	R/W
UART_RX_FILT_REG	RX 滤波器配置	0x0018	R/W
UART_CONF0_SYNC_REG	配置寄存器 0	0x0020	R/W
UART_CONF1_REG	配置寄存器 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	硬件流控配置	0x002C	R/W
UART_SLEEP_CONF0_REG	UART 睡眠配置寄存器 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART 睡眠配置寄存器 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART 睡眠配置寄存器 2	0x0038	R/W
UART_SWFC_CONF0_SYNC_REG	软件流控字符配置	0x003C	varies
UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX 断开字符配置	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	帧结束空闲配置	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 模式配置	0x004C	R/W
UART_CLK_CONF_REG	UART core 时钟配置	0x0088	R/W
UART_REG_UPDATE_REG	UART 寄存器配置更新	0x0098	R/W/SC
UART_ID_REG	UART ID 寄存器	0x009C	R/W
状态寄存器			
UART_STATUS_REG	UART 状态寄存器	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0068	RO
UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x006C	RO
UART_FSM_STATUS_REG	UART 发送和接收状态	0x0070	RO
UART_AFIFO_STATUS_REG	UART 异步 FIFO 状态	0x0090	RO
AT 转义序列检测配置寄存器			
UART_AT_CMD_PRECNT_SYNC_REG	序列发送前的时序配置	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	序列发送后的时序配置	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	超时配置	0x0058	R/W

名称	描述	地址	访问
UART_AT_CMD_CHAR_SYNC_REG	AT 转义序列检测配置	0x005C	R/W
自动波特率检测寄存器			
UART_POSPULSE_REG	自动波特率检测高电平脉冲寄存器	0x0074	RO
UART_NEGPULSE_REG	自动波特率检测低电平脉冲寄存器	0x0078	RO
UART_LOWPULSE_REG	自动波特率检测最短低电平脉冲持续时间寄存器	0x007C	RO
UART_HIGHPULSE_REG	自动波特率检测最短高电平脉冲持续时间寄存器	0x0080	RO
UART_RXD_CNT_REG	自动波特率检测沿变化计数寄存器	0x0084	RO
版本寄存器			
UART_DATE_REG	UART 版本控制寄存器	0x008C	R/W

32.7.2 LP UART 寄存器列表

本小节的所有地址均为相对于 LP UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 配置寄存器			
LP_UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
LP_UART_TOUT_CONF_SYNC_REG	LP UART 阈值和分配配置	0x0064	R/W
LP UART 中断寄存器			
LP_UART_INT_RAW_REG	原始中断状态	0x0004	R/WTC/SS
LP_UART_INT_ST_REG	屏蔽中断状态	0x0008	RO
LP_UART_INT_ENA_REG	中断使能位	0x000C	R/W
LP_UART_INT_CLR_REG	中断清除位	0x0010	WT
配置寄存器			
LP_UART_CLKDIV_SYNC_REG	时钟分频配置	0x0014	R/W
LP_UART_RX_FILT_REG	RX 滤波器配置	0x0018	R/W
LP_UART_CONF0_SYNC_REG	配置寄存器 0	0x0020	R/W
LP_UART_CONF1_REG	配置寄存器 1	0x0024	R/W
LP_UART_HWFC_CONF_SYNC_REG	硬件流控配置	0x002C	R/W
LP_UART_SLEEP_CONF0_REG	LP UART 睡眠配置寄存器 0	0x0030	R/W
LP_UART_SLEEP_CONF1_REG	LP UART 睡眠配置寄存器 1	0x0034	R/W
LP_UART_SLEEP_CONF2_REG	LP UART 睡眠配置寄存器 2	0x0038	R/W
LP_UART_SWFC_CONF0_SYNC_REG	软件流控字符配置	0x003C	varies
LP_UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	R/W
LP_UART_TXBRK_CONF_SYNC_REG	TX 断开字符配置	0x0044	R/W
LP_UART_IDLE_CONF_SYNC_REG	帧结束空闲配置	0x0048	R/W
LP_UART_DELAY_CONF_SYNC_REG	RS485 模式配置	0x004C	R/W
LP_UART_CLK_CONF_REG	LP UART core 时钟配置	0x0088	R/W
LP_UART_REG_UPDATE_REG	LP UART 寄存器配置更新	0x0098	R/W/SC

名称	描述	地址	访问
LP_UART_ID_REG	LP UART ID 寄存器	0x009C	R/W
状态寄存器			
LP_UART_STATUS_REG	LP UART 状态寄存器	0x001C	RO
LP_UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0068	RO
LP_UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x006C	RO
LP_UART_FSM_STATUS_REG	LP UART 发送和接收状态	0x0070	RO
LP_UART_AFIFO_STATUS_REG	LP UART 异步 FIFO 状态	0x0090	RO
AT 转义序列检测配置寄存器			
LP_UART_AT_CMD_PRECNT_SYNC_REG	序列发送前的时序配置	0x0050	R/W
LP_UART_AT_CMD_POSTCNT_SYNC_REG	序列发送前后的时序配置	0x0054	R/W
LP_UART_AT_CMD_GAPTOUT_SYNC_REG	超时配置	0x0058	R/W
LP_UART_AT_CMD_CHAR_SYNC_REG	AT 转义序列检测配置	0x005C	R/W
版本寄存器			
LP_UART_DATE_REG	LP UART 版本控制寄存器	0x008C	R/W

32.7.3 UHCI 寄存器列表

本小节的所有地址均为相对于 UHCI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
UHCI_CONF0_REG	UHCI 配置寄存器	0x0000	R/W
UHCI_CONF1_REG	UHCI 配置寄存器	0x0014	varies
UHCI_ESCAPE_CONF_REG	转义符配置	0x0020	R/W
UHCI_HUNG_CONF_REG	超时配置	0x0024	R/W
UHCI_ACK_NUM_REG	配置 UHCI ACK 值	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI 快速发送配置寄存器	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 快速发送寄存器	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 快速发送寄存器	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 快速发送寄存器	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 快速发送寄存器	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 快速发送寄存器	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 快速发送寄存器	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 快速发送寄存器	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 快速发送寄存器	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 快速发送寄存器	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 快速发送寄存器	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5 WORD0 快速发送寄存器	0x005C	R/W
UHCI_REG_Q5_WORD1_REG	Q5 WORD1 快速发送寄存器	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6 WORD0 快速发送寄存器	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6 WORD1 快速发送寄存器	0x0068	R/W

名称	描述	地址	访问
UHCI_ESC_CONF0_REG	转义序列配置寄存器 0	0x006C	R/W
UHCI_ESC_CONF1_REG	转义序列配置寄存器 1	0x0070	R/W
UHCI_ESC_CONF2_REG	转义序列配置寄存器 2	0x0074	R/W
UHCI_ESC_CONF3_REG	转义序列配置寄存器 3	0x0078	R/W
UHCI_PKT_THRES_REG	包长度配置寄存器	0x007C	R/W
UHCI 中断寄存器			
UHCI_INT_RAW_REG	原始中断状态	0x0004	varies
UHCI_INT_ST_REG	屏蔽中断状态	0x0008	RO
UHCI_INT_ENA_REG	中断使能位	0x000C	R/W
UHCI_INT_CLR_REG	中断清除位	0x0010	WT
UHCI 状态寄存器			
UHCI_STATE0_REG	UHCI 接收状态	0x0018	RO
UHCI_STATE1_REG	UHCI 发送状态	0x001C	RO
UHCI_RX_HEAD_REG	UHCI 包报头寄存器	0x002C	RO
版本寄存器			
UHCI_DATE_REG	UHCI 版本控制寄存器	0x0080	R/W

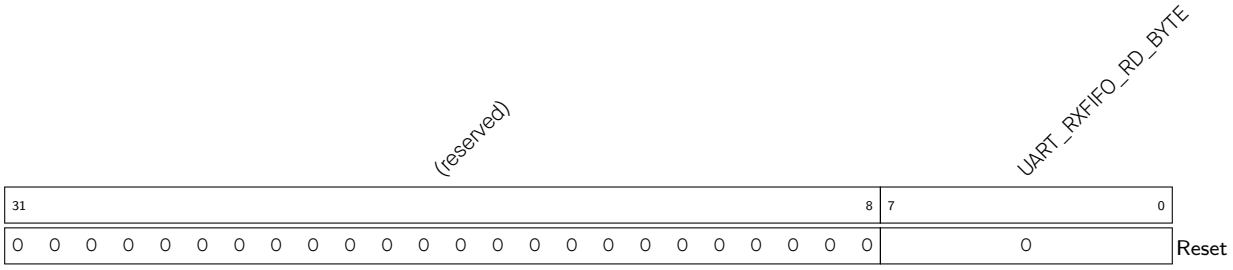
32.8 寄存器

32.8.1 UART 寄存器

本小节的所有地址均为相对于 UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

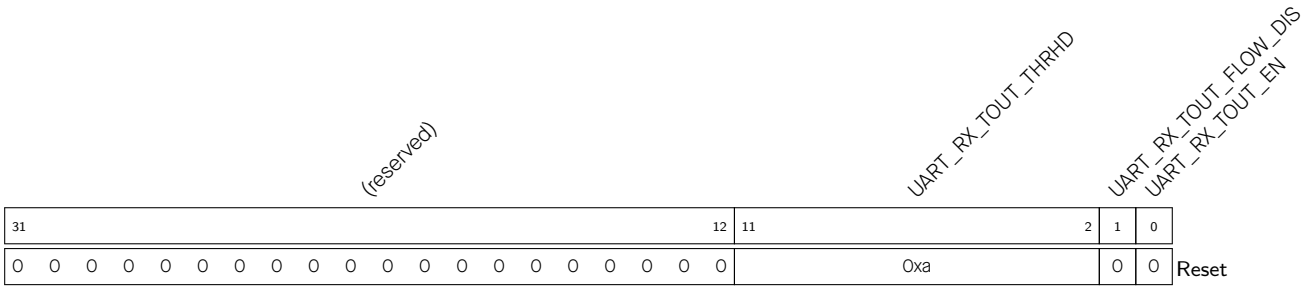
Register 32.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE 表示 UART*n* 从 FIFO 读取的数据。

单位：字节。(RO)

Register 32.2. UART_TOUT_CONF_SYNC_REG (0x0064)



UART_RX_TOUT_EN 配置是否开启 UART 接收器的超时功能。

0：关闭

1：开启

(R/W)

UART_RX_TOUT_FLOW_DIS 配置开启硬件流控时是否关闭 idle 状态计数器计数。

0：开启

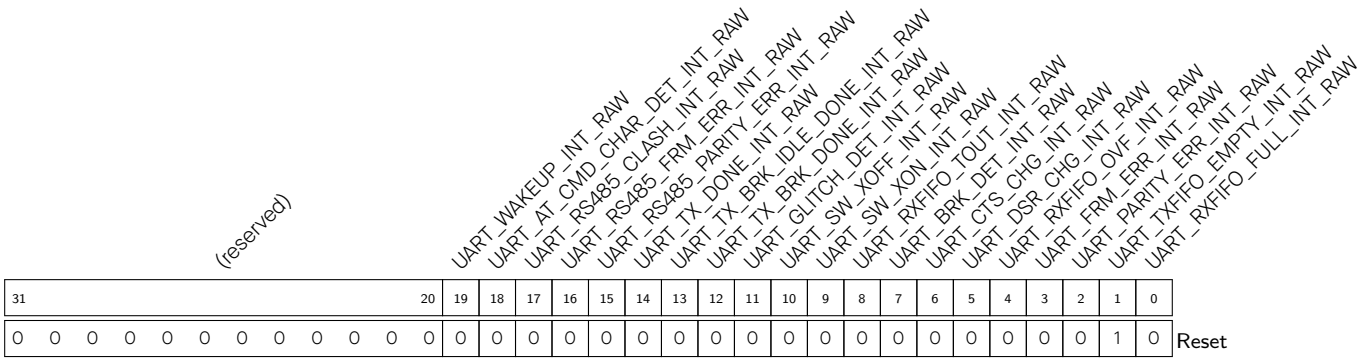
1：关闭

(R/W)

UART_RX_TOUT_THRHD 配置触发超时前总线可保持空闲状态的时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 32.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW UART_RXFIFO_FULL_INT 的原始中断状态。(R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW UART_TXFIFO_EMPTY_INT 的原始中断状态。(R/WTC/SS)

UART_PARITY_ERR_INT_RAW UART_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_FRM_ERR_INT_RAW UART_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW UART_RXFIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

UART_DSR_CHG_INT_RAW UART_DSR_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_CTS_CHG_INT_RAW UART_CTS_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_BRK_DET_INT_RAW UART_BRK_DET_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_TOUT_INT_RAW UART_RXFIFO_TOUT_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XON_INT_RAW UART_SW_XON_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XOFF_INT_RAW UART_SW_XOFF_INT 的原始中断状态。(R/WTC/SS)

UART_GLITCH_DET_INT_RAW UART_GLITCH_DET_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_DONE_INT_RAW UART_TX_BRK_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_IDLE_DONE_INT_RAW UART_TX_BRK_IDLE_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_DONE_INT_RAW UART_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_PARITY_ERR_INT_RAW UART_RS485_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

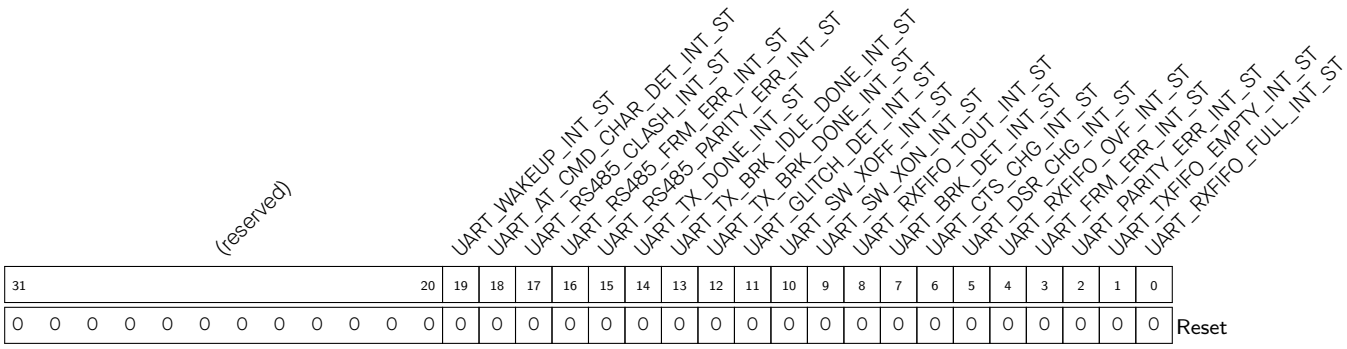
UART_RS485_FRM_ERR_INT_RAW UART_RS485_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_CLASH_INT_RAW UART_RS485_CLASH_INT 的原始中断状态。(R/WTC/SS)

UART_AT_CMD_CHAR_DET_INT_RAW UART_AT_CMD_CHAR_DET_INT 的原始中断状态。(R/WTC/SS)

UART_WAKEUP_INT_RAW UART_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 32.4. UART_INT_ST_REG (0x0008)



UART_RXFIFO_FULL_INT_ST UART_RXFIFO_FULL_INT 的屏蔽中断状态。(RO)

UART_TXFIFO_EMPTY_INT_ST UART_TXFIFO_EMPTY_INT 的屏蔽中断状态。(RO)

UART_PARITY_ERR_INT_ST UART_PARITY_ERR_INT 的屏蔽中断状态。(RO)

UART_FRM_ERR_INT_ST UART_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_OVF_INT_ST UART_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

UART_DSR_CHG_INT_ST UART_DSR_CHG_INT 的屏蔽中断状态。(RO)

UART_CTS_CHG_INT_ST UART_CTS_CHG_INT 的屏蔽中断状态。(RO)

UART_BRK_DET_INT_ST UART_BRK_DET_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_TOUT_INT_ST UART_RXFIFO_TOUT_INT 的屏蔽中断状态。(RO)

UART_SW_XON_INT_ST UART_SW_XON_INT 的屏蔽中断状态。(RO)

UART_SW_XOFF_INT_ST UART_SW_XOFF_INT 的屏蔽中断状态。(RO)

UART_GLITCH_DET_INT_ST UART_GLITCH_DET_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_DONE_INT_ST UART_TX_BRK_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_IDLE_DONE_INT_ST UART_TX_BRK_IDLE_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_DONE_INT_ST UART_TX_DONE_INT 的屏蔽中断状态。(RO)

UART_RS485_PARITY_ERR_INT_ST UART_RS485_PARITY_ERR_INT 的屏蔽中断状态。(RO)

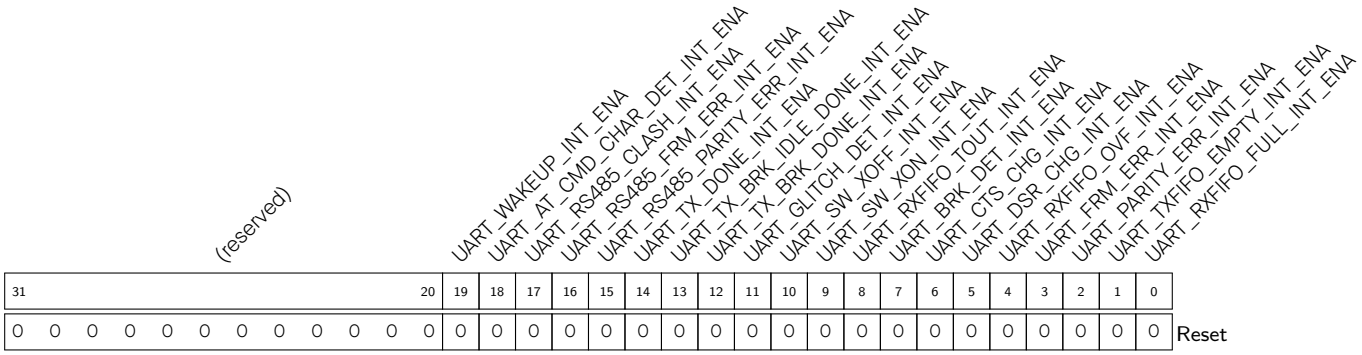
UART_RS485_FRM_ERR_INT_ST UART_RS485_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RS485_CLASH_INT_ST UART_RS485_CLASH_INT 的屏蔽中断状态。(RO)

UART_AT_CMD_CHAR_DET_INT_ST UART_AT_CMD_CHAR_DET_INT 的屏蔽中断状态。(RO)

UART_WAKEUP_INT_ST UART_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 32.5. UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA 写 1 使能 UART_RXFIFO_FULL_INT。(R/W)

UART_TXFIFO_EMPTY_INT_ENA 写 1 使能 UART_TXFIFO_EMPTY_INT。(R/W)

UART_PARITY_ERR_INT_ENA 写 1 使能 UART_PARITY_ERR_INT。(R/W)

UART_FRM_ERR_INT_ENA 写 1 使能 UART_FRM_ERR_INT。(R/W)

UART_RXFIFO_OVF_INT_ENA 写 1 使能 UART_RXFIFO_OVF_INT。(R/W)

UART_DSR_CHG_INT_ENA 写 1 使能 UART_DSR_CHG_INT。(R/W)

UART_CTS_CHG_INT_ENA 写 1 使能 UART_CTS_CHG_INT。(R/W)

UART_BRK_DET_INT_ENA 写 1 使能 UART_BRK_DET_INT。(R/W)

UART_RXFIFO_TOUT_INT_ENA 写 1 使能 UART_RXFIFO_TOUT_INT。(R/W)

UART_SW_XON_INT_ENA 写 1 使能 UART_SW_XON_INT。(R/W)

UART_SW_XOFF_INT_ENA 写 1 使能 UART_SW_XOFF_INT。(R/W)

UART_GLITCH_DET_INT_ENA 写 1 使能 UART_GLITCH_DET_INT。(R/W)

UART_TX_BRK_DONE_INT_ENA 写 1 使能 UART_TX_BRK_DONE_INT。(R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA 写 1 使能 UART_TX_BRK_IDLE_DONE_INT。(R/W)

UART_TX_DONE_INT_ENA 写 1 使能 UART_TX_DONE_INT。(R/W)

UART_RS485_PARITY_ERR_INT_ENA 写 1 使能 UART_RS485_PARITY_ERR_INT。(R/W)

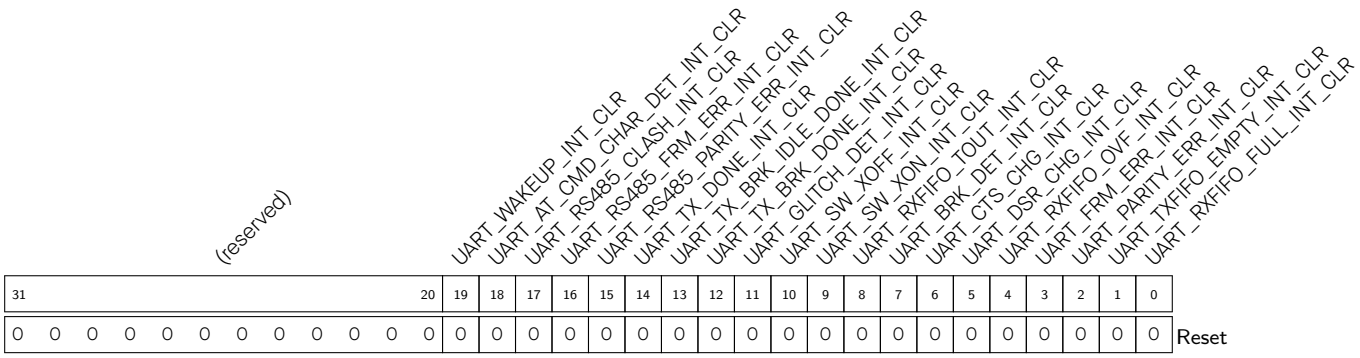
UART_RS485_FRM_ERR_INT_ENA 写 1 使能 UART_RS485_FRM_ERR_INT。(R/W)

UART_RS485_CLASH_INT_ENA 写 1 使能 UART_RS485_CLASH_INT。(R/W)

UART_AT_CMD_CHAR_DET_INT_ENA 写 1 使能 UART_AT_CMD_CHAR_DET_INT。(R/W)

UART_WAKEUP_INT_ENA 写 1 使能 UART_WAKEUP_INT。(R/W)

Register 32.6. UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR 写 1 清除 UART_RXFIFO_FULL_INT。(WT)

UART_TXFIFO_EMPTY_INT_CLR 写 1 清除 UART_TXFIFO_EMPTY_INT。(WT)

UART_PARITY_ERR_INT_CLR 写 1 清除 UART_PARITY_ERR_INT。(WT)

UART_FRM_ERR_INT_CLR 写 1 清除 UART_FRM_ERR_INT。(WT)

UART_RXFIFO_OVF_INT_CLR 写 1 清除 UART_RXFIFO_OVF_INT。(WT)

UART_DSR_CHG_INT_CLR 写 1 清除 UART_DSR_CHG_INT。(WT)

UART_CTS_CHG_INT_CLR 写 1 清除 UART_CTS_CHG_INT。(WT)

UART_BRK_DET_INT_CLR 写 1 清除 UART_BRK_DET_INT。(WT)

UART_RXFIFO_TOUT_INT_CLR 写 1 清除 UART_RXFIFO_TOUT_INT。(WT)

UART_SW_XON_INT_CLR 写 1 清除 UART_SW_XON_INT。(WT)

UART_SW_XOFF_INT_CLR 写 1 清除 UART_SW_XOFF_INT。(WT)

UART_GLITCH_DET_INT_CLR 写 1 清除 UART_GLITCH_DET_INT。(WT)

UART_TX_BRK_DONE_INT_CLR 写 1 清除 UART_TX_BRK_DONE_INT。(WT)

UART_TX_BRK_IDLE_DONE_INT_CLR 写 1 清除 UART_TX_BRK_IDLE_DONE_INT。(WT)

UART_TX_DONE_INT_CLR 写 1 清除 UART_TX_DONE_INT。(WT)

UART_RS485_PARITY_ERR_INT_CLR 写 1 清除 UART_RS485_PARITY_ERR_INT。(WT)

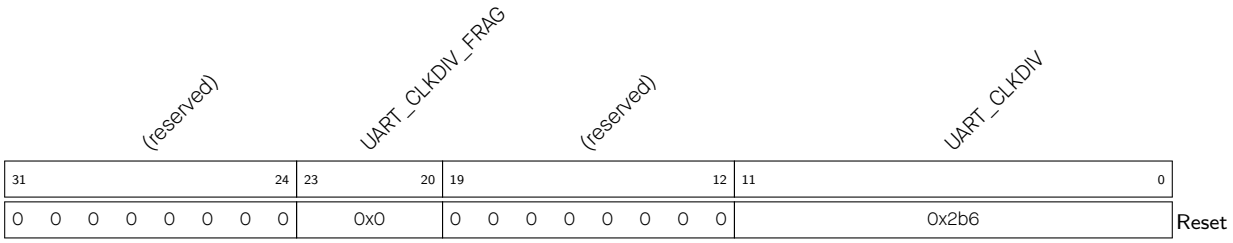
UART_RS485_FRM_ERR_INT_CLR 写 1 清除 UART_RS485_FRM_ERR_INT。(WT)

UART_RS485_CLASH_INT_CLR 写 1 清除 UART_RS485_CLASH_INT。(WT)

UART_AT_CMD_CHAR_DET_INT_CLR 写 1 清除 UART_AT_CMD_CHAR_DET_INT。(WT)

UART_WAKEUP_INT_CLR 写 1 清除 UART_WAKEUP_INT。(WT)

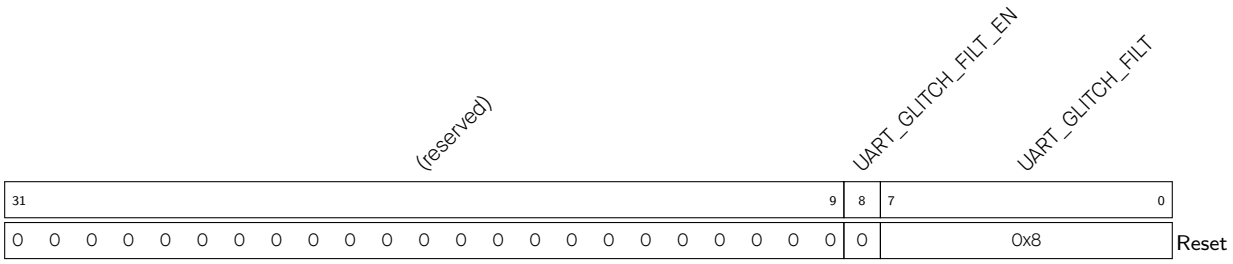
Register 32.7. UART_CLKDIV_SYNC_REG (0x0014)



UART_CLKDIV 配置波特率分频系数的整数部分。(R/W)

UART_CLKDIV_FRAG 配置波特率分频系数的小数部分。(R/W)

Register 32.8. UART_RX_FILT_REG (0x0018)



UART_GLITCH_FILT 配置滤波长度。

单位：UART Core 时钟周期。

宽度小于该数值的输入脉冲会被忽略。(R/W)

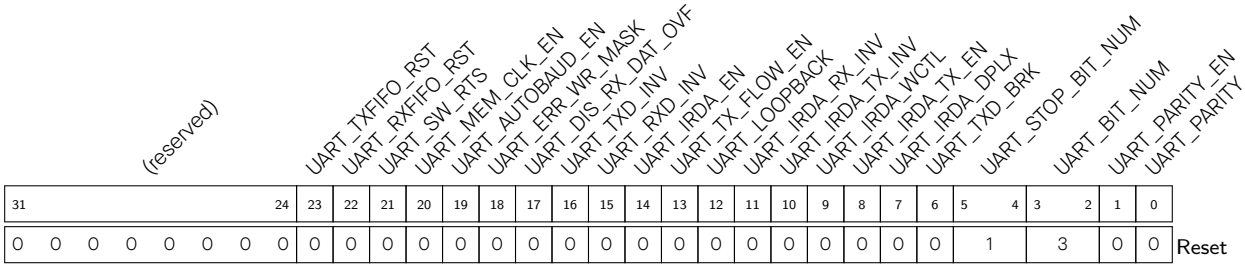
UART_GLITCH_FILT_EN 配置是否开启接收信号滤波器。

0：关闭

1：开启

(R/W)

Register 32.9. UART_CONF0_SYNC_REG (0x0020)



UART_PARITY 配置奇偶检验方式。

0：偶校验

1：奇校验

(R/W)

UART_PARITY_EN 配置是否开启 UART 奇偶校验。

0：关闭

1：开启

(R/W)

UART_BIT_NUM 配置数据位长度。

0：5 位

1：6 位

2：7 位

3：8 位

(R/W)

UART_STOP_BIT_NUM 配置停止位长度。

0: 无效值，没有作用

1: 1 位

2: 1.5 位

3: 2 位

(R/W)

UART_TXD_BRK 配置完成数据发送后，发送器是否发送 NULL。

0：不发送

1：发送

(R/W)

UART_IRDA_DPLX 配置是否开启 IrDA 回环测试模式。

0：关闭

1：开启

(R/W)

UART_IRDA_TX_EN 配置是否开启 IrDA 发送器。

0：关闭

1：开启

(R/W)

见下页...

Register 32.9. UART_CONF0_SYNC_REG (0x0020)

接上页...

UART_IRDA_WCTL 配置 IrDA 发送器的第 11 位。

0：该位为 0

1：该位与第 10 位相同

(R/W)

UART_IRDA_TX_INV 配置是否翻转 IrDA 发送器的电平。

0：不翻转

1：翻转

(R/W)

UART_IRDA_RX_INV 配置是否翻转 IrDA 接收器的电平。

0：不翻转

1：翻转

(R/W)

UART_LOOPBACK 配置是否开启 UART 回环测试模式。

0：关闭

1：开启

(R/W)

UART_TX_FLOW_EN 配置是否开启发送器的流控功能。

0：关闭

1：开启

(R/W)

UART_IRDA_EN 配置是否开启 IrDA。

0：关闭

1：开启

(R/W)

UART_RXD_INV 配置是否翻转 UART RXD 信号电平。

0：不翻转

1：翻转

(R/W)

UART_TXD_INV 配置是否翻转 UART TXD 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DIS_RX_DAT_OVF 配置是否关闭 UART 接收器数据溢出检测。

0：开启

1：关闭

(R/W)

见下页...

Register 32.9. UART_CONF0_SYNC_REG (0x0020)

接上页...

UART_ERR_WR_MASK 配置接收数据有错误时，是否仍将其存储至 FIFO。

0：存储

1：不存储

(R/W)

UART_AUTOBAUD_EN 配置是否开启波特率检测。

0：关闭

1：开启

(R/W)

UART_MEM_CLK_EN 配置是否开启 UART 存储器门控。

0：关闭

1：开启

(R/W)

UART_SW_RTS 配置软件流控使用的 RTS 信号。

0: 禁止发送端发送数据

1: 允许发送端发送数据

(R/W)

UART_RXFIFO_RST 配置是否复位 UART RX FIFO。

0：不复位

1：复位

(R/W)

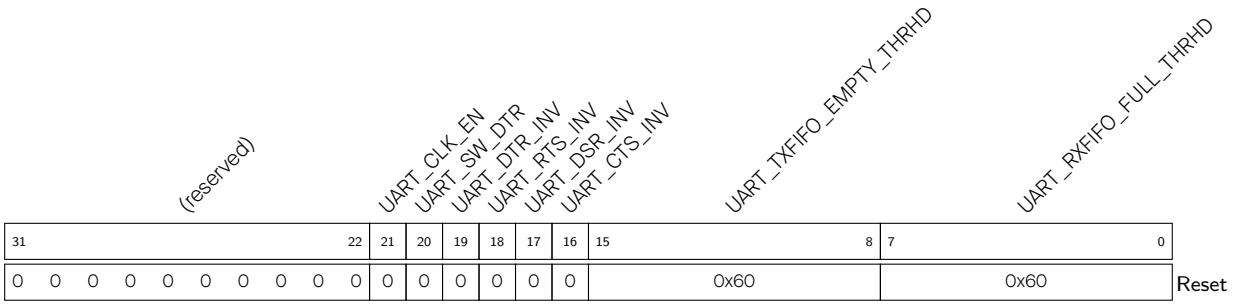
UART_TXFIFO_RST 配置是否复位 UART TX FIFO。

0：不复位

1：复位

(R/W)

Register 32.10. UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD 配置 RX FIFO 为满的阈值。

单位：字节。(R/W)

UART_TXFIFO_EMPTY_THRHD 配置 TX FIFO 为空的阈值。

单位：字节。(R/W)

UART_CTS_INV 配置是否翻转 UART CTS 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DSR_INV 配置是否翻转 UART DSR 信号电平。

0：不翻转

1：翻转

(R/W)

UART_RTS_INV 配置是否翻转 UART RTS 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DTR_INV 配置是否翻转 UART DTR 信号电平。

0：不翻转

1：翻转

(R/W)

UART_SW_DTR 配置软件流控使用的 DTR 信号。

0: 发送数据未准备完毕，处于不可用状态

1: 发送数据准备完毕，处于可用状态

(R/W)

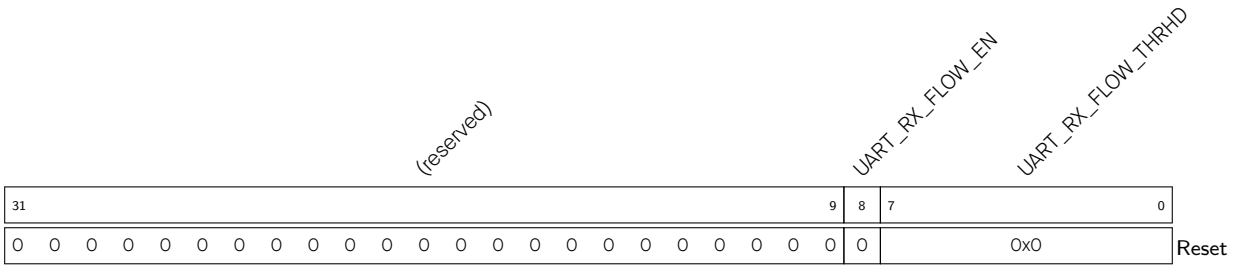
UART_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 32.11. UART_HWFC_CONF_SYNC_REG (0x002C)



UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大数。

单位：字节。(R/W)

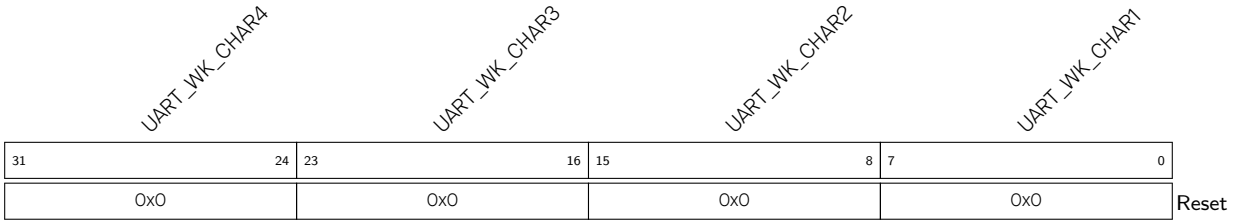
UART_RX_FLOW_EN 配置是否开启 UART 接收器。

0：关闭

1：开启

(R/W)

Register 32.12. UART_SLEEP_CONF0_REG (0x0030)



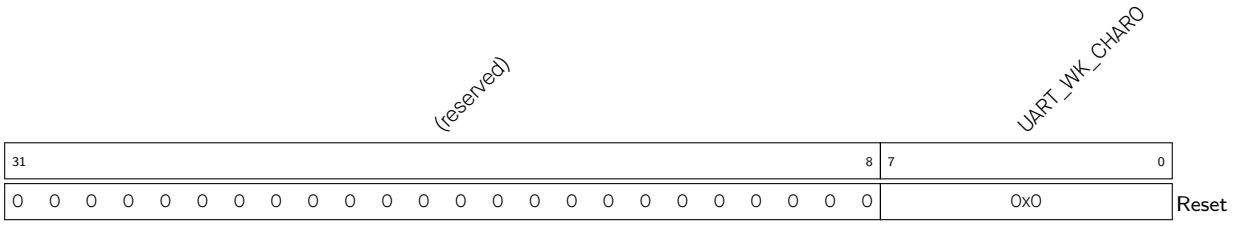
UART_WK_CHAR1 配置唤醒芯片的特定字符 1。(R/W)

UART_WK_CHAR2 配置唤醒芯片的特定字符 2。(R/W)

UART_WK_CHAR3 配置唤醒芯片的特定字符 3。(R/W)

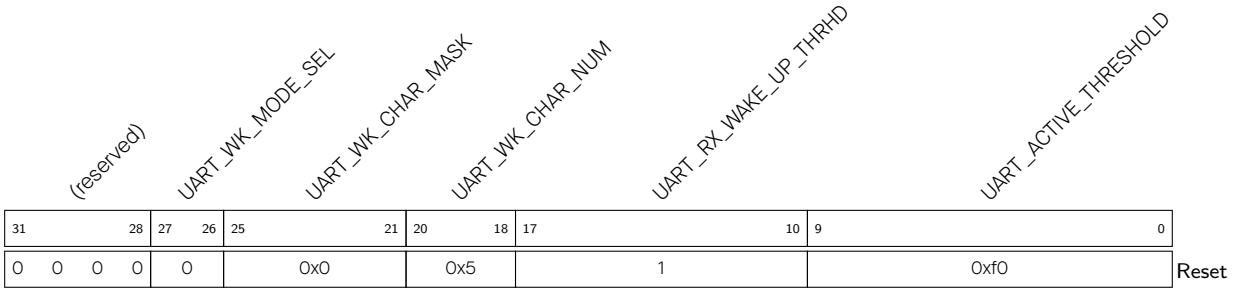
UART_WK_CHAR4 配置唤醒芯片的特定字符 4。(R/W)

Register 32.13. UART_SLEEP_CONF1_REG (0x0034)



UART_WK_CHAR0 配置唤醒芯片的特定字符 0。(R/W)

Register 32.14. UART_SLEEP_CONF2_REG (0x0038)



UART_ACTIVE_THRESHOLD 配置唤醒模式 0 唤醒芯片所需的 RXD 沿变化次数。(R/W)

UART_RX_WAKE_UP_THRHD 配置唤醒模式 1 唤醒芯片所需接收的数据量。

单位：字节。(R/W)

UART_WK_CHAR_NUM 配置选取多少个唤醒字符。(R/W)

UART_WK_CHAR_MASK 配置是否屏蔽唤醒字符。

0：不屏蔽

1：屏蔽

(R/W)

UART_WK_MODE_SEL 配置选取哪个唤醒模式。不同模式的含义详见 32.4.8 唤醒 小节。

0：唤醒模式 0

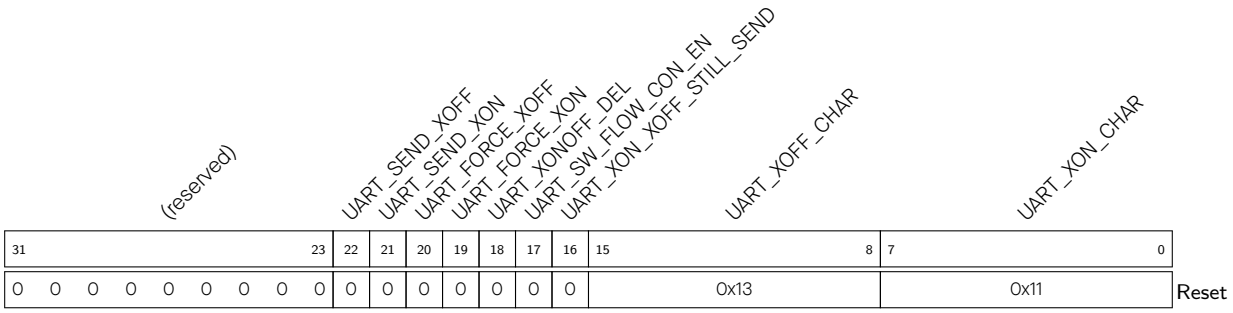
1：唤醒模式 1

2：唤醒模式 2

3：唤醒模式 3

(R/W)

Register 32.15. UART_SWFC_CONF0_SYNC_REG (0x003C)



UART_XON_CHAR 配置 XON 流控字符。(R/W)

UART_XOFF_CHAR 配置 XOFF 流控字符。(R/W)

UART_XON_XOFF_STILL_SEND 配置 UART 发送器关闭时，是否仍能发送 XON 或 XOFF 字符。

0：不能发送

1：可以发送

(R/W)

UART_SW_FLOW_CON_EN 配置是否开启软件流控。

0：关闭

1：开启

(R/W)

UART_XONOFF_DEL 配置是否移除接收数据中的流控字符。

0：不移除

1：移除

(R/W)

UART_FORCE_XON 配置是否让发送器继续发送数据。

0：不发送

1：继续发送

(R/W)

UART_FORCE_XOFF 配置是否让发送器停止发送数据。

0：不停止

1：停止

(R/W)

UART_SEND_XON 配置是否发送 XON 字符。

0：不发送

1：发送

(R/W/SS/SC)

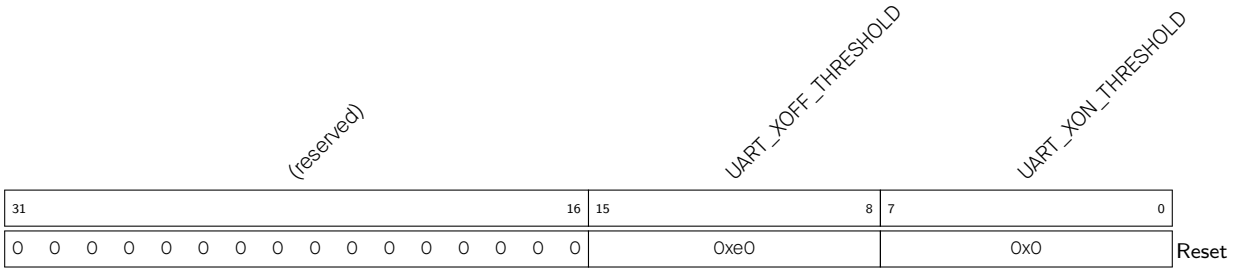
UART_SEND_XOFF 配置是否发送 XOFF 字符。

0：不发送

1：发送

(R/W/SS/SC)

Register 32.16. UART_SWFC_CONF1_REG (0x0040)



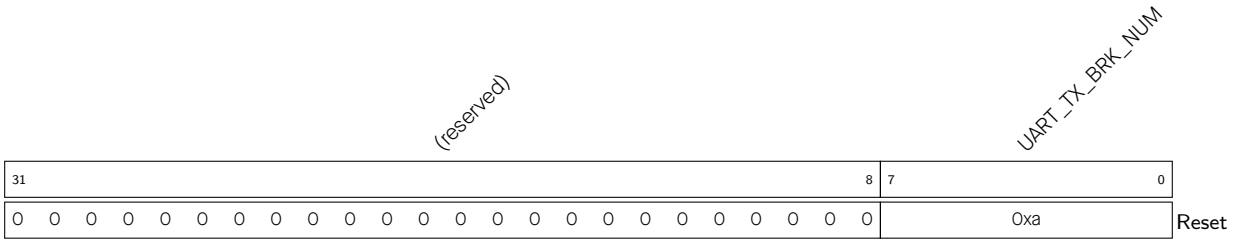
UART_XON_THRESHOLD 配置软件流控时，发送 XON 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

UART_XOFF_THRESHOLD 配置软件流控时，发送 XOFF 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

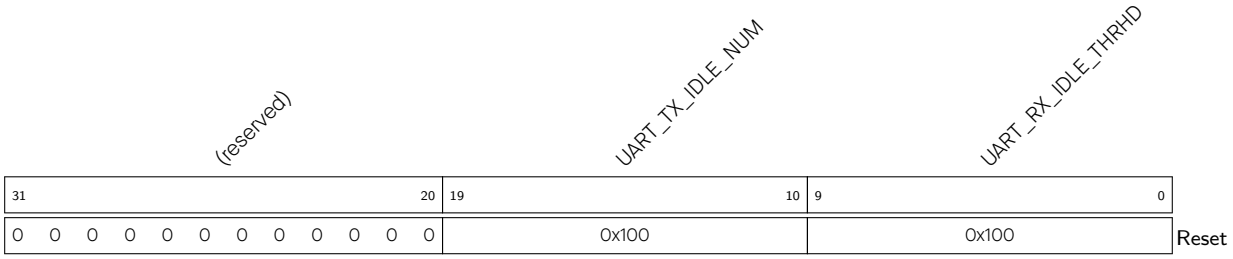
Register 32.17. UART_TXBRK_CONF_SYNC_REG (0x0044)



UART_TX_BRK_NUM 配置完成数据发送后待发 NULL 字符的数量。

仅在 UART_TXD_BRK 为 1 时有效。(R/W)

Register 32.18. UART_IDLE_CONF_SYNC_REG (0x0048)



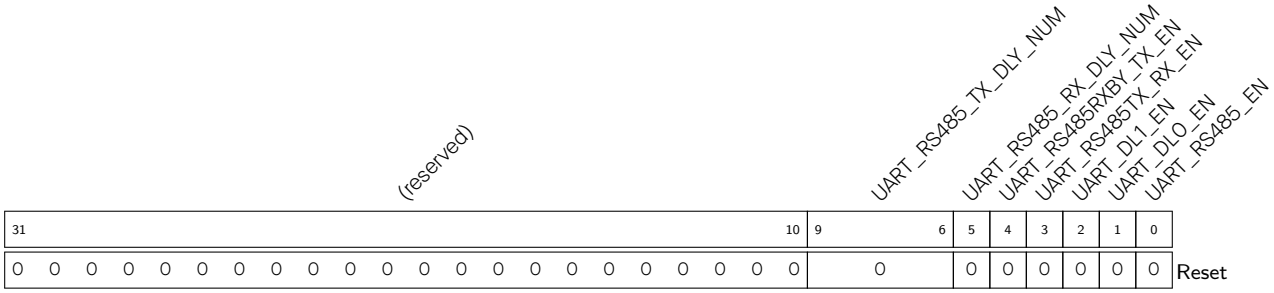
UART_RX_IDLE_THRHD 配置接收一字节数据时间过长、产生帧结束信号的阈值。

单位：位时间（传输一位所需的时间）。(R/W)

UART_TX_IDLE_NUM 配置两次数据传输的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 32.19. UART_RS485_CONF_SYNC_REG (0x004C)



UART_RS485_EN 配置是否开启 RS485 模式。

0：关闭

1：开启

(R/W)

UART_DLO_EN 配置是否在起始位之前增加一位延时。

0：不增加

1：增加

(R/W)

UART_DL1_EN 配置是否在停止位之后增加一位延时。

0：不增加

1：增加

(R/W)

UART_RS485TX_RX_EN 配置发送器在 RS485 模式下发送数据时，是否开启接收器接收数据。

0：关闭

1：开启

(R/W)

UART_RS485RXBY_TX_EN 配置 RS485 接收器线路繁忙时，是否开启 RS485 发送器发送数据。

0：关闭

1：开启

(R/W)

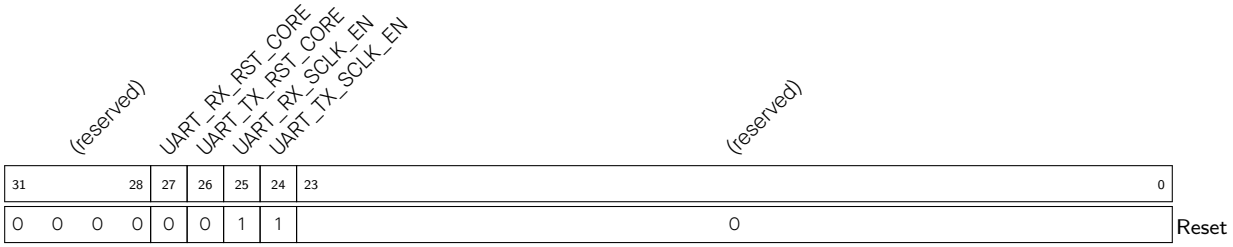
UART_RS485_RX_DLY_NUM 配置接收器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

UART_RS485_TX_DLY_NUM 配置发送器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

Register 32.20. UART_CLK_CONF_REG (0x0088)



UART_TX_SCLK_EN 配置是否开启 UART TX 时钟。

0：关闭

1：开启

(R/W)

UART_RX_SCLK_EN 配置是否开启 UART RX 时钟。

0：关闭

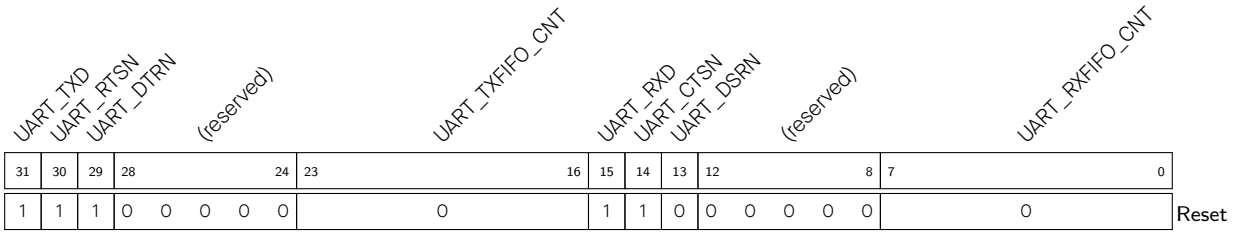
1：开启

(R/W)

UART_TX_RST_CORE 向此位先写 1 后写 0，复位 UART TX。(R/W)

UART_RX_RST_CORE 向此位先写 1 后写 0，复位 UART RX。(R/W)

Register 32.21. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT 表示 RX FIFO 中有效数据量。

单位：字节。(RO)

UART_DSRN 表示内部 UART DSR 信号的电平值。(RO)

UART_CTSN 表示内部 UART CTS 信号的电平值。(RO)

UART_RXD 表示内部 UART RXD 信号的电平值。(RO)

UART_TXFIFO_CNT 表示 TX FIFO 中的数据量。

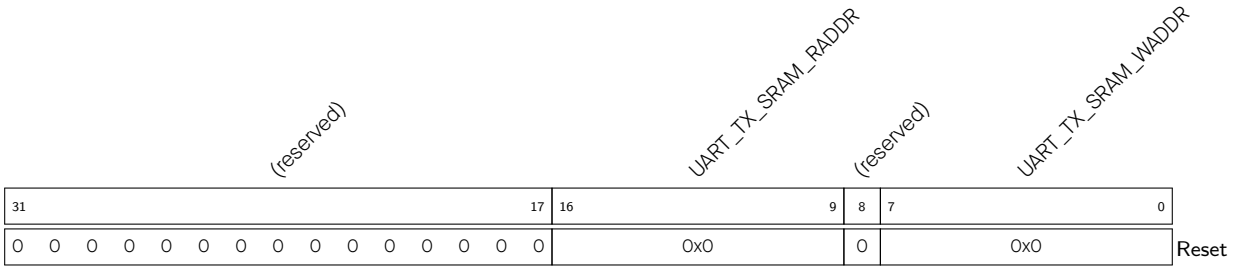
单位：字节。(RO)

UART_DTRN 表示内部 UART DTR 信号的电平。(RO)

UART_RTSN 表示内部 UART RTS 信号的电平。(RO)

UART_TXD 表示内部 UART TXD 信号的电平。(RO)

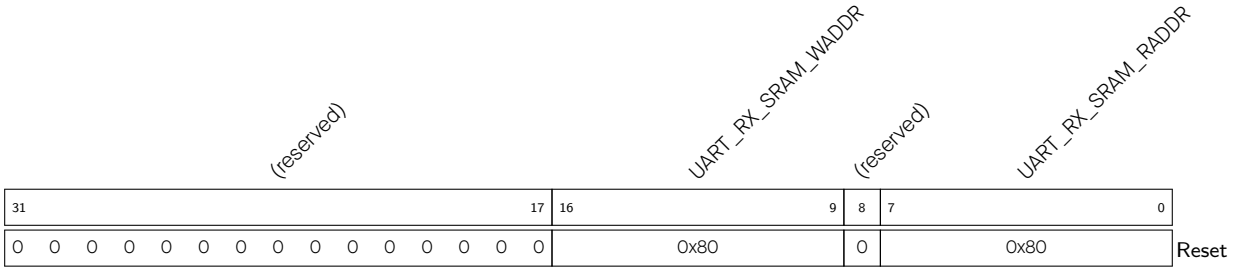
Register 32.22. UART_MEM_TX_STATUS_REG (0x0068)



UART_TX_SRAM_WADDR 表示写 TX FIFO 的偏移地址。(RO)

UART_TX_SRAM_RADDR 表示读 TX FIFO 的偏移地址。(RO)

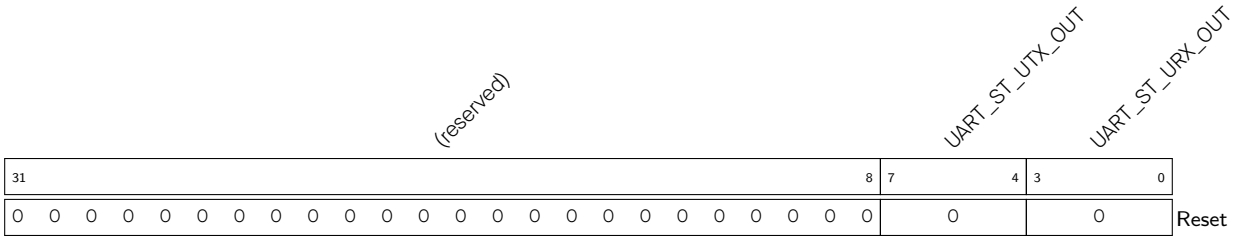
Register 32.23. UART_MEM_RX_STATUS_REG (0x006C)



UART_RX_SRAM_RADDR 表示读 RX FIFO 的偏移地址。(RO)

UART_RX_SRAM_WADDR 表示写 RX FIFO 的偏移地址。(RO)

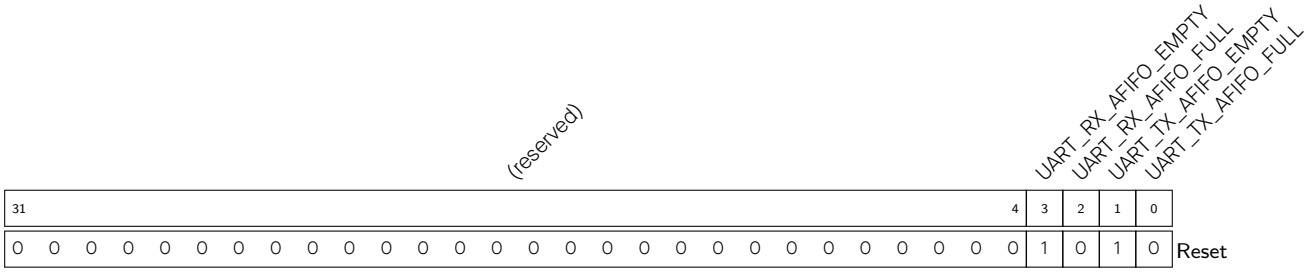
Register 32.24. UART_FSM_STATUS_REG (0x0070)



UART_ST_URX_OUT 表示 UART 接收器的状态。(RO)

UART_ST_UTX_OUT 表示 UART 发送器的状态。(RO)

Register 32.25. UART_AFIFO_STATUS_REG (0x0090)



UART_TX_AFIFO_FULL 表示 APB TX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

UART_TX_AFIFO_EMPTY 表示 APB TX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

UART_RX_AFIFO_FULL 表示 APB RX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

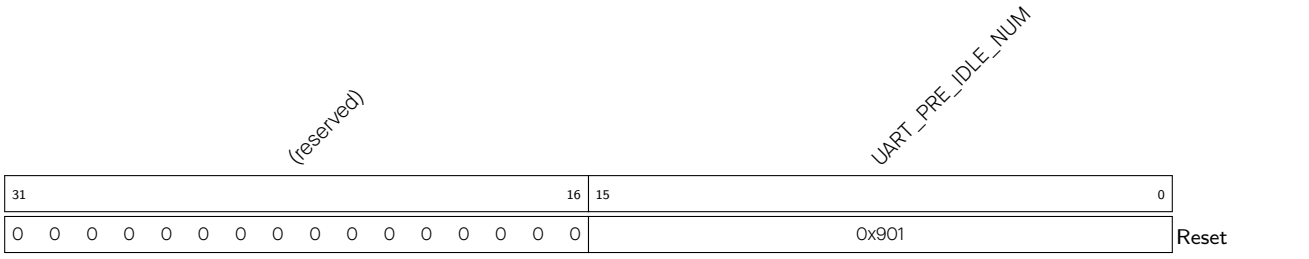
UART_RX_AFIFO_EMPTY 表示 APB RX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

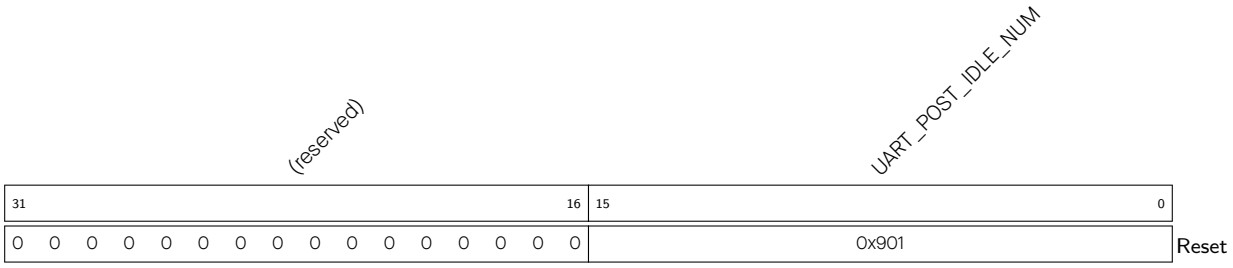
Register 32.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



UART_PRE_IDLE_NUM 配置接收器接收到第一个 AT_CMD 之前的空闲时间。

单位：位时间（传输一位所需的时间）。(R/W)

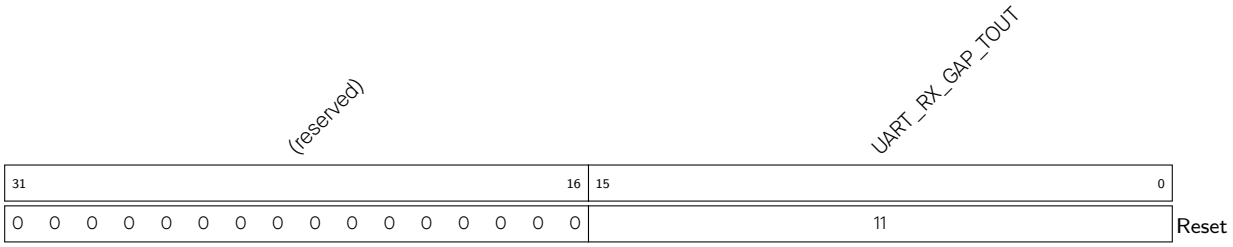
Register 32.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



UART_POST_IDLE_NUM 配置最后一个 AT_CMD 和后续数据之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

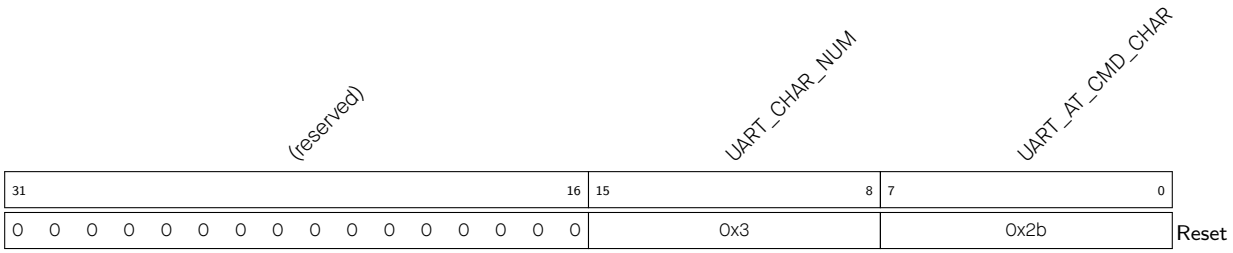
Register 32.28. UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



UART_RX_GAP_TOUT 配置 AT_CMD 之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

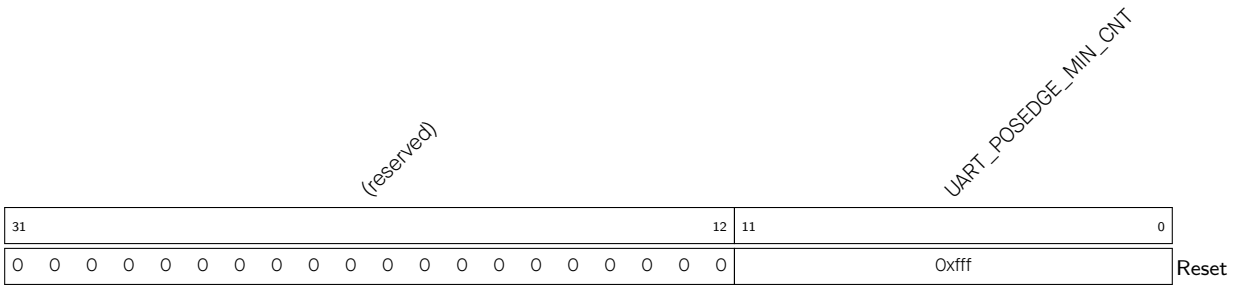
Register 32.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)



UART_AT_CMD_CHAR 配置 AT_CMD 字符。(R/W)

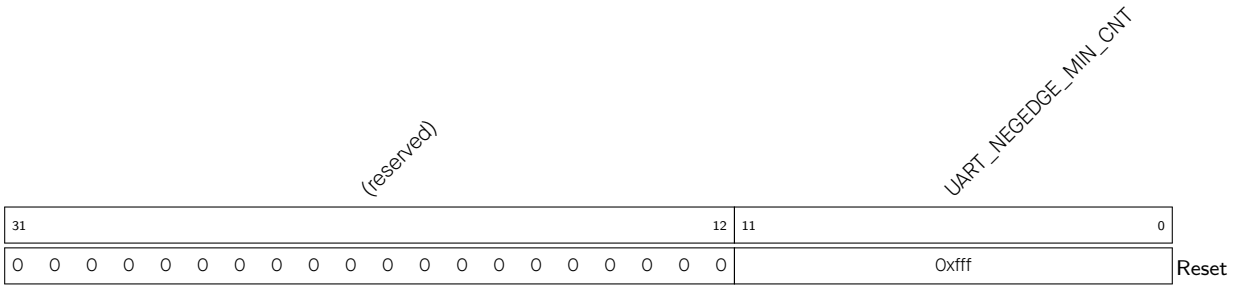
UART_CHAR_NUM 配置接收器可连续接收的 AT_CMD 字符数量。(R/W)

Register 32.30. UART_POSPULSE_REG (0x0074)



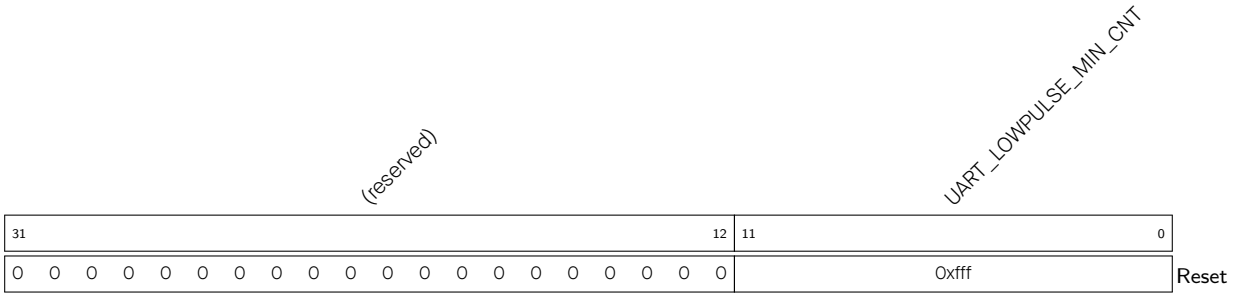
UART_POSEDGE_MIN_CNT 表示两个上升沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 32.31. UART_NEGPULSE_REG (0x0078)



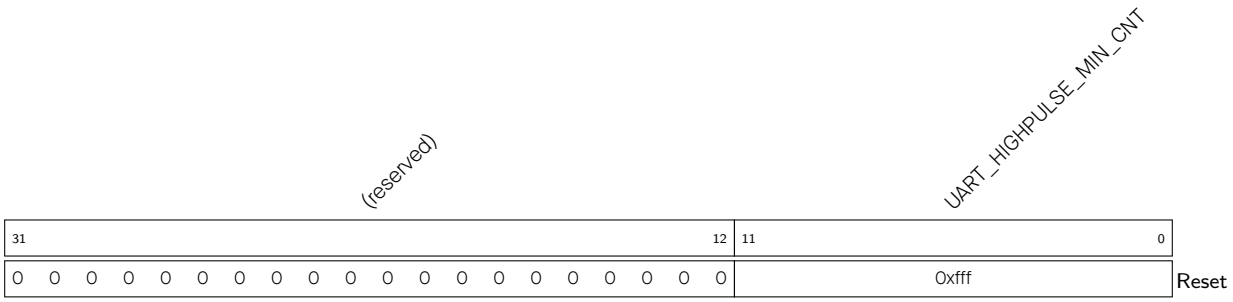
UART_NEGEDGE_MIN_CNT 两个下降沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 32.32. UART_LOWPULSE_REG (0x007C)



UART_LOWPULSE_MIN_CNT 表示低电平脉冲的最短持续时间，用于波特率检测。
单位：APB_CLK 时钟周期。(RO)

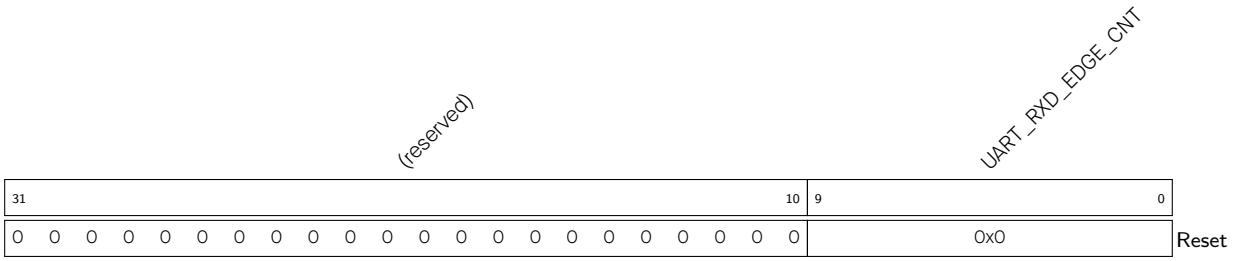
Register 32.33. UART_HIGHPULSE_REG (0x0080)



UART_HIGHPULSE_MIN_CNT 表示高电平脉冲最长持续时间，用于波特率检测。

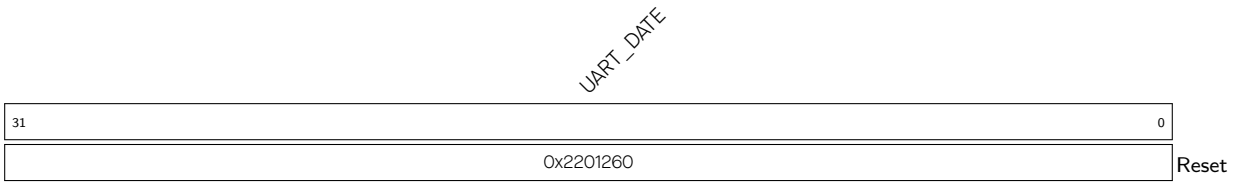
单位：APB_CLK 时钟周期。(RO)

Register 32.34. UART_RXD_CNT_REG (0x0084)



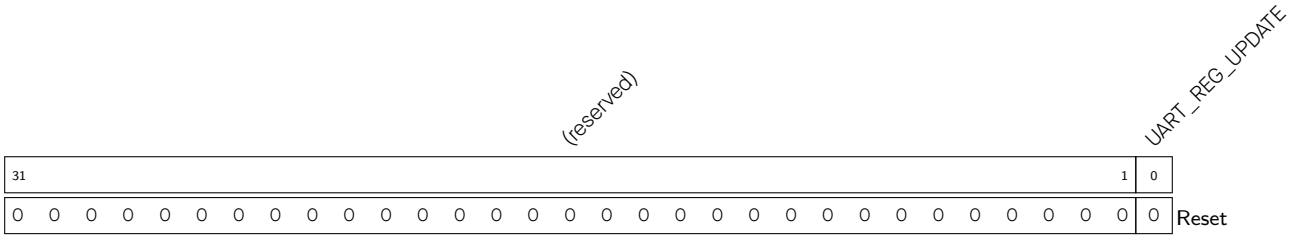
UART_RXD_EDGE_CNT 表示 RXD 沿变化的次数。用于波特率检测。(RO)

Register 32.35. UART_DATE_REG (0x008C)



UART_DATE 版本控制寄存器。(R/W)

Register 32.36. UART_REG_UPDATE_REG (0x0098)



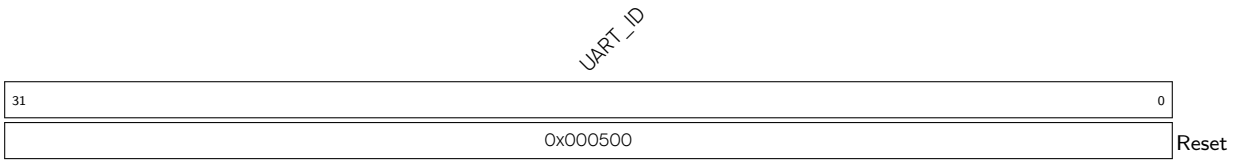
UART_REG_UPDATE 配置是否同步寄存器

0：不同步

1：同步

(R/W/SC)

Register 32.37. UART_ID_REG (0x009C)



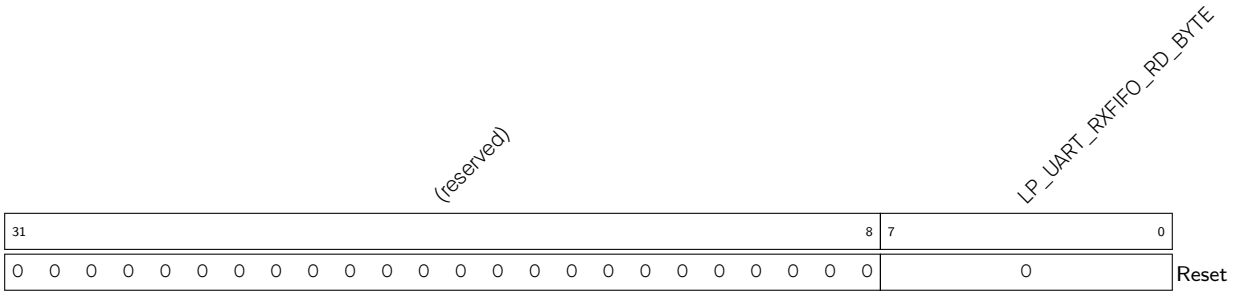
UART_ID 配置 UART ID。(R/W)

32.8.2 LP UART 寄存器

本小节的所有地址均为相对于 LP UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

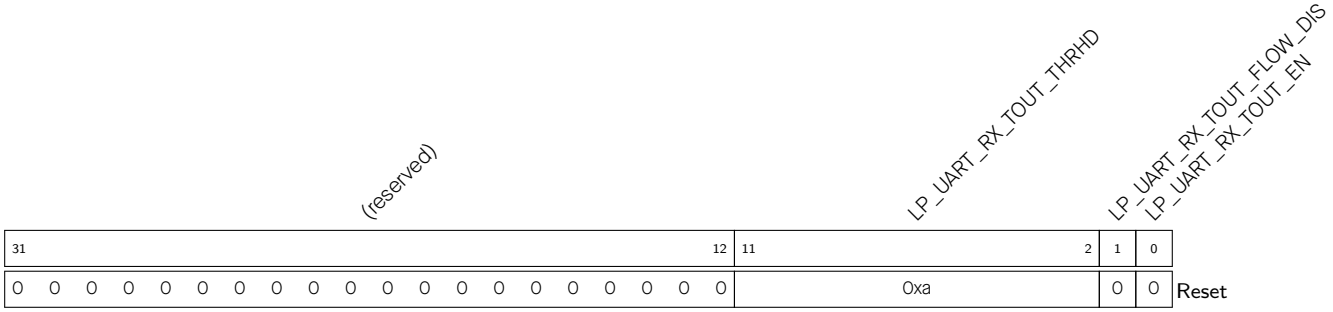
Register 32.38. LP_UART_FIFO_REG (0x0000)



LP_UART_RXFIFO_RD_BYTE 表示 LP UART *n* 从 FIFO 读取的数据。

单位：字节。(RO)

Register 32.39. LP_UART_TOUT_CONF_SYNC_REG (0x0064)



LP_UART_RX_TOUT_EN 配置是否开启 LP UART 接收器的超时功能。

0：关闭

1：开启

(R/W)

LP_UART_RX_TOUT_FLOW_DIS 配置开启硬件流控时是否关闭 idle 状态计数器计数。

0：无效值，没有作用

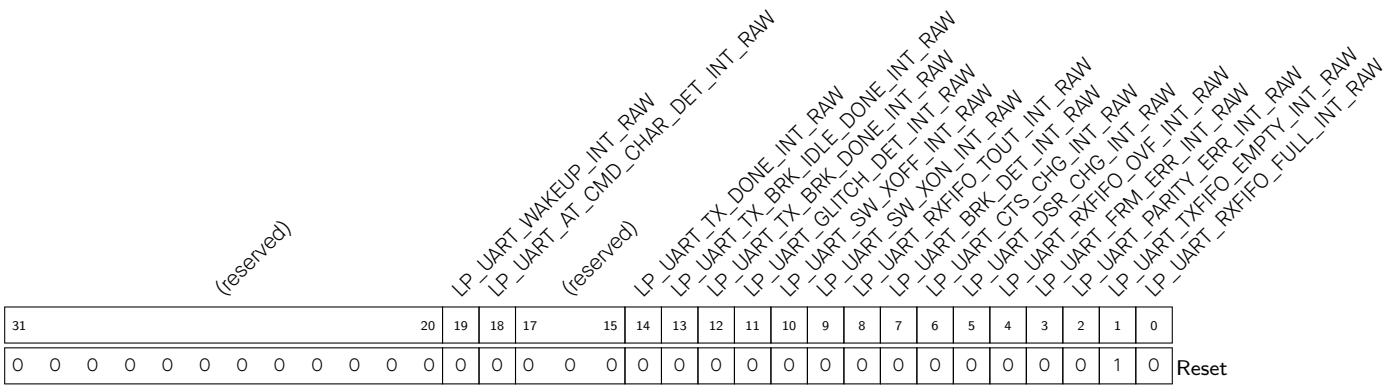
1：关闭

(R/W)

LP_UART_RX_TOUT_THRHD 配置触发超时前总线可保持空闲状态的时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 32.40. LP_UART_INT_RAW_REG (0x0004)



LP_UART_RXFIFO_FULL_INT_RAW LP_UART_RXFIFO_FULL_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TXFIFO_EMPTY_INT_RAW LP_UART_TXFIFO_EMPTY_INT 的原始中断状态。(R/ WTC/SS)

LP_UART_PARITY_ERR_INT_RAW LP_UART_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

LP_UART_FRM_ERR_INT_RAW LP_UART_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

LP_UART_RXFIFO_OVF_INT_RAW LP_UART_RXFIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

LP_UART_DSR_CHG_INT_RAW LP_UART_DSR_CHG_INT 的原始中断状态。(R/WTC/SS)

LP_UART_CTS_CHG_INT_RAW LP_UART_CTS_CHG_INT 的原始中断状态。(R/WTC/SS)

LP_UART_BRK_DET_INT_RAW LP_UART_BRK_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_RXFIFO_TOUT_INT_RAW UART_RXFIFO_TOUT_INT 的原始中断状态。(R/WTC/SS)

LP_UART_SW_XON_INT_RAW LP_UART_SW_XON_INT 的原始中断状态。(R/WTC/SS)

LP_UART_SW_XOFF_INT_RAW LP_UART_SW_XOFF_INT 的原始中断状态。(R/WTC/SS)

LP_UART_GLITCH_DET_INT_RAW LP_UART_GLITCH_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TX_BRK_DONE_INT_RAW UART_TX_BRK_DONE_INT 的原始中断状态。(R/WTC/SS)

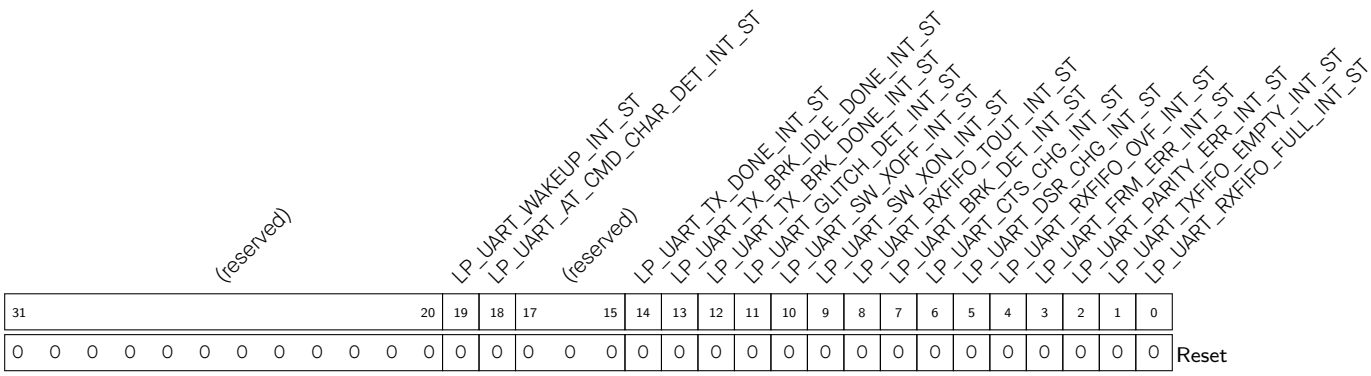
LP_UART_TX_BRK_IDLE_DONE_INT_RAW LP_UART_TX_BRK_IDLE_DONE_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TX_DONE_INT_RAW LP_UART_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

LP_UART_AT_CMD_CHAR_DET_INT_RAW LP_UART_AT_CMD_CHAR_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_WAKEUP_INT_RAW LP_UART_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 32.41. LP_UART_INT_ST_REG (0x0008)



LP_UART_RXFIFO_FULL_INT_ST LP_UART_RXFIFO_FULL_INT 的屏蔽中断状态。(RO)

LP_UART_TXFIFO_EMPTY_INT_ST LP_UART_TXFIFO_EMPTY_INT 的屏蔽中断状态。(RO)

LP_UART_PARITY_ERR_INT_ST LP_UART_PARITY_ERR_INT 的屏蔽中断状态。(RO)

LP_UART_FRM_ERR_INT_ST LP_UART_FRM_ERR_INT 的屏蔽中断状态。(RO)

LP_UART_RXFIFO_OVF_INT_ST LP_UART_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

LP_UART_DSR_CHG_INT_ST LP_UART_DSR_CHG_INT 的屏蔽中断状态。(RO)

LP_UART_CTS_CHG_INT_ST LP_UART_CTS_CHG_INT 的屏蔽中断状态。(RO)

LP_UART_BRK_DET_INT_ST LP_UART_BRK_DET_INT 的屏蔽中断状态。(RO)

LP_UART_RXFIFO_TOUT_INT_ST LP_UART_RXFIFO_TOUT_INT 的屏蔽中断状态。(RO)

LP_UART_SW_XON_INT_ST LP_UART_SW_XON_INT 的屏蔽中断状态。(RO)

LP_UART_SW_XOFF_INT_ST LP_UART_SW_XOFF_INT 的屏蔽中断状态。(RO)

LP_UART_GLITCH_DET_INT_ST LP_UART_GLITCH_DET_INT 的屏蔽中断状态。(RO)

LP_UART_TX_BRK_DONE_INT_ST LP_UART_TX_BRK_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_TX_BRK_IDLE_DONE_INT_ST LP_UART_TX_BRK_IDLE_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_TX_DONE_INT_ST LP_UART_TX_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_AT_CMD_CHAR_DET_INT_ST LP_UART_AT_CMD_CHAR_DET_INT 的屏蔽中断状态。(RO)

LP_UART_WAKEUP_INT_ST LP_UART_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 32.42. LP_UART_INT_ENA_REG (0x000C)

Bits	Field	Reset
31–20	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0
19	LP_UART_WAKEUP_INT_ENA	0
18	LP_UART_AT_CMD_CHAR_DET_INT_ENA	0
17–15	(reserved)	0 0 0
14	LP_UART_TX_DONE_INT_ENA	0
13	LP_UART_TX_BRK_IDLE_DONE_INT_ENA	0
12	LP_UART_TX_BRK_DONE_INT_ENA	0
11	LP_UART_GLITCH_DET_INT_ENA	0
10	LP_UART_SW_XOFF_INT_ENA	0
9	LP_UART_SW_XON_INT_ENA	0
8	LP_UART_RXFIFO_TOUT_INT_ENA	0
7	LP_UART_BRK_DET_INT_ENA	0
6	LP_UART_CTS_CHG_INT_ENA	0
5	LP_UART_DSR_CHG_INT_ENA	0
4	LP_UART_RXFIFO_OVF_INT_ENA	0
3	LP_UART_FRM_ERR_INT_ENA	0
2	LP_UART_PARITY_ERR_INT_ENA	0
1	LP_UART_TXFIFO_EMPTY_INT_ENA	0
0	LP_UART_RXFIFO_FULL_INT_ENA	0

LP_UART_RXFIFO_FULL_INT_ENA 写 1 使能 LP_UART_RXFIFO_FULL_INT。(R/W)

LP_UART_TXFIFO_EMPTY_INT_ENA 写 1 使能 LP_UART_TXFIFO_EMPTY_INT。(R/W)

LP_UART_PARITY_ERR_INT_ENA 写 1 使能 LP_UART_PARITY_ERR_INT。(R/W)

LP_UART_FRM_ERR_INT_ENA 写 1 使能 LP_UART_FRM_ERR_INT。(R/W)

LP_UART_RXFIFO_OVF_INT_ENA 写 1 使能 LP_UART_RXFIFO_OVF_INT。(R/W)

LP_UART_DSR_CHG_INT_ENA 写 1 使能 LP_UART_DSR_CHG_INT。(R/W)

LP_UART_CTS_CHG_INT_ENA 写 1 使能 LP_UART_CTS_CHG_INT。(R/W)

LP_UART_BRK_DET_INT_ENA 写 1 使能 LP_UART_BRK_DET_INT。(R/W)

LP_UART_RXFIFO_TOUT_INT_ENA 写 1 使能 LP_UART_RXFIFO_TOUT_INT。(R/W)

LP_UART_SW_XON_INT_ENA 写 1 使能 LP_UART_SW_XON_INT。(R/W)

LP_UART_SW_XOFF_INT_ENA 写 1 使能 LP_UART_SW_XOFF_INT。(R/W)

LP_UART_GLITCH_DET_INT_ENA 写 1 使能 LP_UART_GLITCH_DET_INT。(R/W)

LP_UART_TX_BRK_DONE_INT_ENA 写 1 使能 LP_UART_TX_BRK_DONE_INT。(R/W)

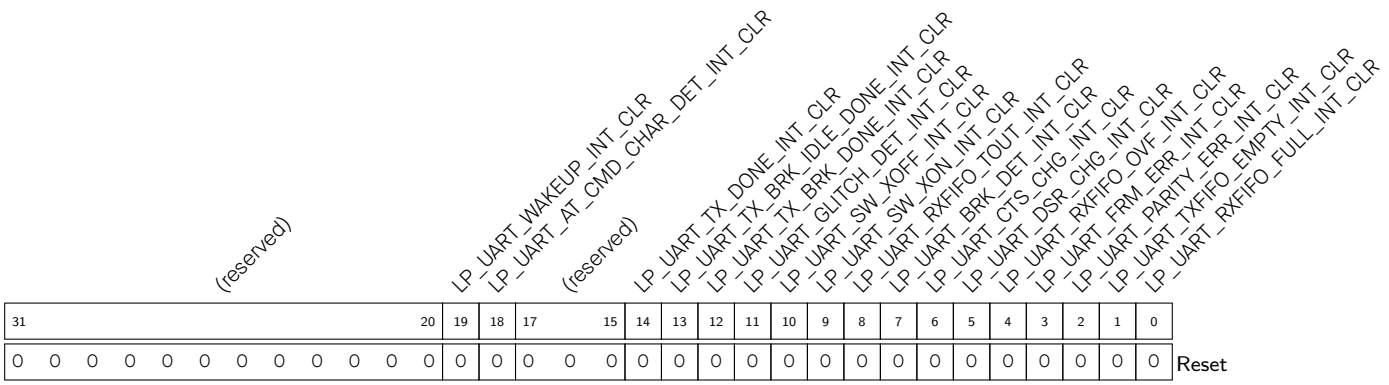
LP_UART_TX_BRK_IDLE_DONE_INT_ENA 写 1 使能 LP_UART_TX_BRK_IDLE_DONE_INT。(R/W)

LP_UART_TX_DONE_INT_ENA 写 1 使能 LP_UART_TX_DONE_INT。(R/W)

LP_UART_AT_CMD_CHAR_DET_INT_ENA 写 1 使能 LP_UART_AT_CMD_CHAR_DET_INT。(R/W)

LP_UART_WAKEUP_INT_ENA 写 1 使能 LP_UART_WAKEUP_INT。(R/W)

Register 32.43. LP_UART_INT_CLR_REG (0x0010)



LP_UART_RXFIFO_FULL_INT_CLR 写 1 清除 LP_UART_RXFIFO_FULL_INT。(WT)

LP_UART_TXFIFO_EMPTY_INT_CLR 写 1 清除 LP_UART_TXFIFO_EMPTY_INT。(WT)

LP_UART_PARITY_ERR_INT_CLR 写 1 清除 LP_UART_PARITY_ERR_INT。(WT)

LP_UART_FRM_ERR_INT_CLR 写 1 清除 LP_UART_FRM_ERR_INT。(WT)

LP_UART_RXFIFO_OVF_INT_CLR 写 1 清除 LP_UART_RXFIFO_OVF_INT。(WT)

LP_UART_DSR_CHG_INT_CLR 写 1 清除 LP_UART_DSR_CHG_INT。(WT)

LP_UART_CTS_CHG_INT_CLR 写 1 清除 LP_UART_CTS_CHG_INT。(WT)

LP_UART_BRK_DET_INT_CLR 写 1 清除 LP_UART_BRK_DET_INT。(WT)

LP_UART_RXFIFO_TOUT_INT_CLR 写 1 清除 LP_UART_RXFIFO_TOUT_INT。(WT)

LP_UART_SW_XON_INT_CLR 写 1 清除 LP_UART_SW_XON_INT。(WT)

LP_UART_SW_XOFF_INT_CLR 写 1 清除 LP_UART_SW_XOFF_INT。(WT)

LP_UART_GLITCH_DET_INT_CLR 写 1 清除 LP_UART_GLITCH_DET_INT。(WT)

LP_UART_TX_BRK_DONE_INT_CLR 写 1 清除 LP_UART_TX_BRK_DONE_INT。(WT)

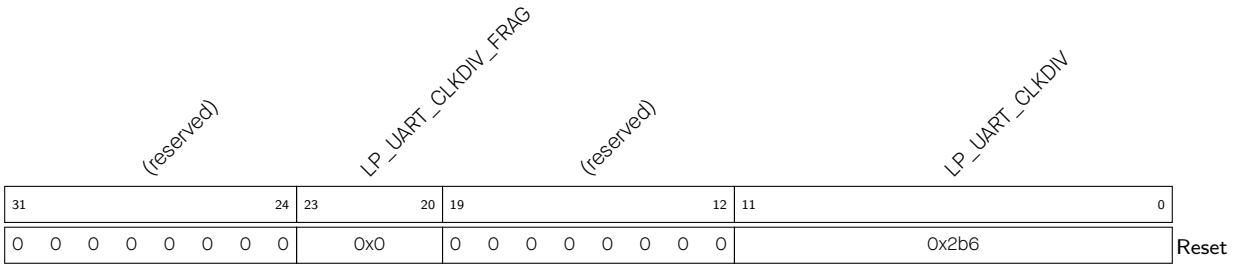
LP_UART_TX_BRK_IDLE_DONE_INT_CLR 写 1 清除 LP_UART_TX_BRK_IDLE_DONE_INT。(WT)

LP_UART_TX_DONE_INT_CLR 写 1 清除 LP_UART_TX_DONE_INT。(WT)

LP_UART_AT_CMD_CHAR_DET_INT_CLR 写 1 清除 LP_UART_AT_CMD_CHAR_DET_INT。(WT)

LP_UART_WAKEUP_INT_CLR 写 1 清除 LP_UART_WAKEUP_INT。(WT)

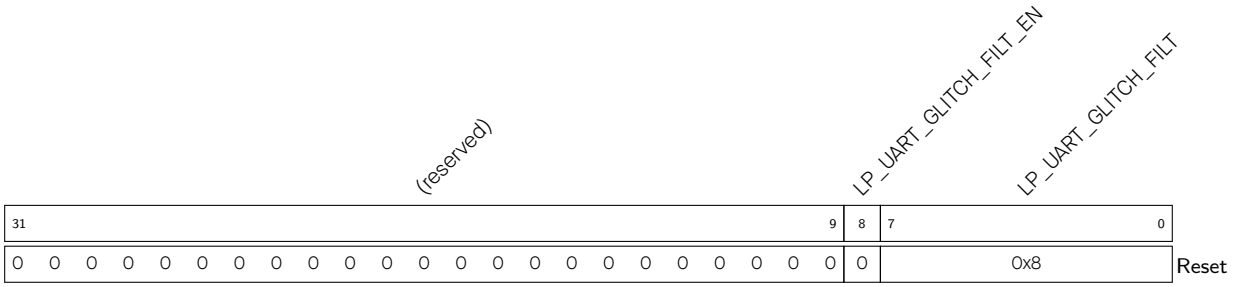
Register 32.44. LP_UART_CLKDIV_SYNC_REG (0x0014)



LP_UART_CLKDIV 配置波特率分频系数的整数部分。(R/W)

LP_UART_CLKDIV_FRAG 配置波特率分频系数的小数部分。(R/W)

Register 32.45. LP_UART_RX_FILT_REG (0x0018)



LP_UART_GLITCH_FILT 配置滤波长度。

单位：UART Core 时钟周期。

宽度小于该数值的输入脉冲会被忽略。(R/W)

LP_UART_GLITCH_FILT_EN 配置是否开启接收信号滤波器。

0：关闭

1：开启

(R/W)

Register 32.46. LP_UART_CONF0_SYNC_REG (0x0020)

Bits	Field	Reset
31:24	(reserved)	0 0 0 0 0 0 0 0
23	LP_UART_TXFIFO_RST	0
22	LP_UART_RXFIFO_RST	0
21	LP_UART_SW_RTS	0
20	LP_UART_MEM_CLK_EN	0
19	(reserved)	0
18	LP_UART_ERR_WR_MASK	0
17	LP_UART_DIS_RX_DAT_OVF	0
16	LP_UART_TXD_INV	0
15	LP_UART_RXD_INV	0
14	(reserved)	0
13	LP_UART_TX_FLOW_EN	0
12	LP_UART_LOOPBACK	0
11:7	(reserved)	0
6	LP_UART_TXD_BRK	0
5:4	LP_UART_STOP_BIT_NUM	1
3:2	LP_UART_BIT_NUM	3
1	LP_UART_PARITY_EN	0
0	LP_UART_PARITY	0

LP_UART_PARITY 配置奇偶检验方式。

0：偶校验

1：奇校验

(R/W)

LP_UART_PARITY_EN 配置是否开启 LP UART 奇偶校验。

0：关闭

1：开启

(R/W)

LP_UART_BIT_NUM 配置数据位长度。

0：5 位

1：6 位

2：7 位

3：8 位

(R/W)

LP_UART_STOP_BIT_NUM 配置停止位长度。

0: 无效值，没有作用

1: 1 位

2: 1.5 位

3: 2 位

(R/W)

LP_UART_TXD_BRK 配置完成数据发送后，发送器是否发送 NULL。

0：不发送

1：发送

(R/W)

LP_UART_LOOPBACK 配置是否开启 LP UART 回环测试模式。

0：关闭

1：开启

(R/W)

LP_UART_TX_FLOW_EN 配置是否开启发送器的流控功能。

0：关闭

1：开启

(R/W)

见下页…

Register 32.46. LP_UART_CONF0_SYNC_REG (0x0020)

接上页...

LP_UART_RXD_INV 配置是否翻转 LP UART RXD 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_TXD_INV 配置是否翻转 LP UART TXD 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_DIS_RX_DAT_OVF 配置是否关闭 LP UART 接收器数据溢出检测。

0：开启

1：关闭

(R/W)

LP_UART_ERR_WR_MASK 配置接收数据有错误时，是否扔将其存储至 FIFO。

0：存储

1：不存储

(R/W)

LP_UART_MEM_CLK_EN 配置是否开启 LP UART 存储器门控。

0：关闭

1：开启

(R/W)

LP_UART_SW_RTS 配置软件流控使用的 RTS 信号。

0: 禁止发送端发送数据

1: 允许发送端发送数据

(R/W)

LP_UART_RXFIFO_RST 配置是否复位 LP UART RX FIFO。

0：不复位

1：复位

(R/W)

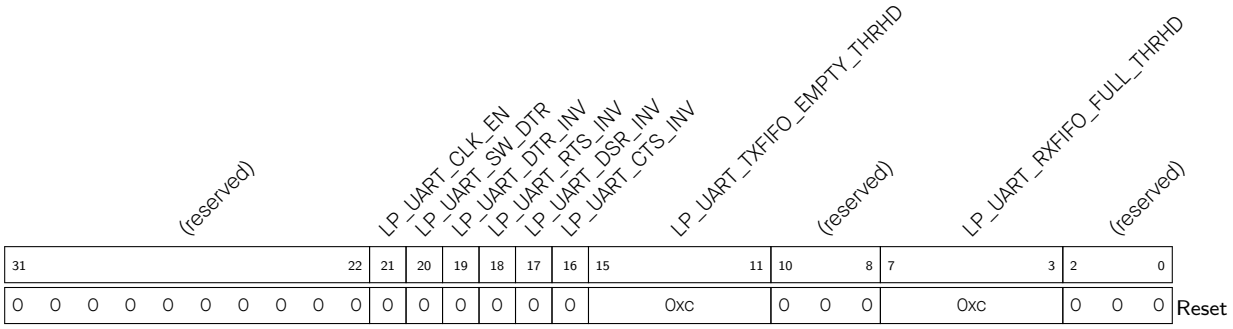
LP_UART_TXFIFO_RST 配置是否复位 LP UART TX FIFO。

0：不复位

1：复位

(R/W)

Register 32.47. LP_UART_CONF1_REG (0x0024)



LP_UART_RXFIFO_FULL_THRHD 配置 RX FIFO 为满的阈值。

单位：字节。(R/W)

LP_UART_TXFIFO_EMPTY_THRHD 配置 TX FIFO 为空的阈值。

单位：字节。(R/W)

LP_UART_CTS_INV 配置是否翻转 LP UART CTS 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_DSR_INV 配置是否翻转 LP UART DSR 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_RTS_INV 配置是否翻转 LP UART RTS 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_DTR_INV 配置是否翻转 LP UART DTR 信号电平。

0：不翻转

1：翻转

(R/W)

LP_UART_SW_DTR 配置软件流控使用的 DTR 信号。

0: 发送数据未准备完毕，处于不可用状态

1: 发送数据准备完毕，处于可用状态

(R/W)

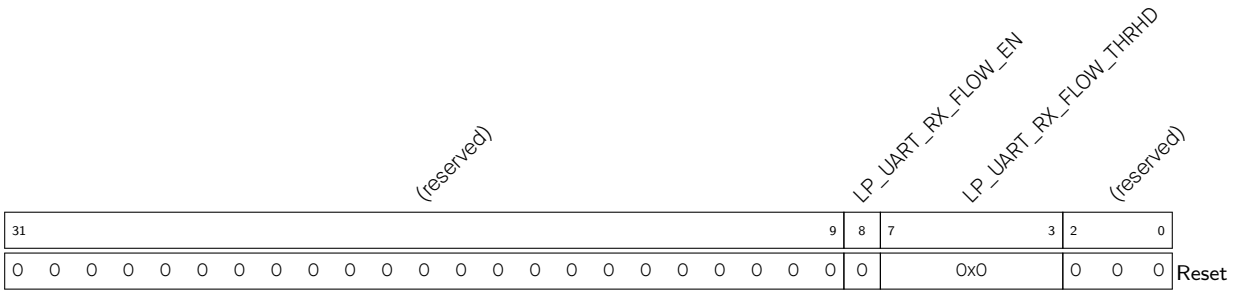
LP_UART_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 32.48. LP_UART_HWFC_CONF_SYNC_REG (0x002C)



LP_UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大数。

单位：字节。(R/W)

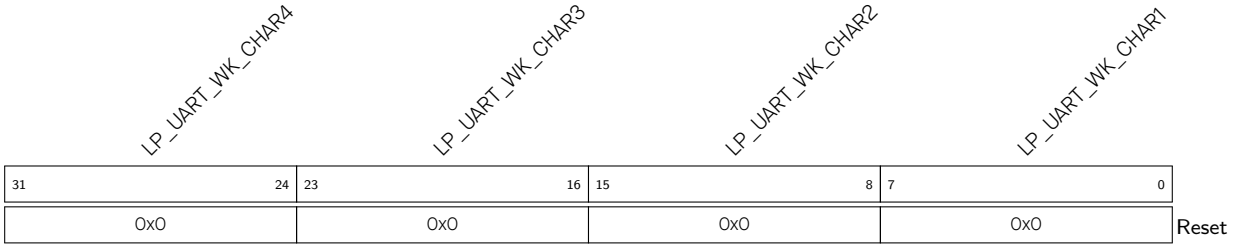
LP_UART_RX_FLOW_EN 配置是否开启 LP UART 接收器。

0：关闭

1：开启

(R/W)

Register 32.49. LP_UART_SLEEP_CONF0_REG (0x0030)



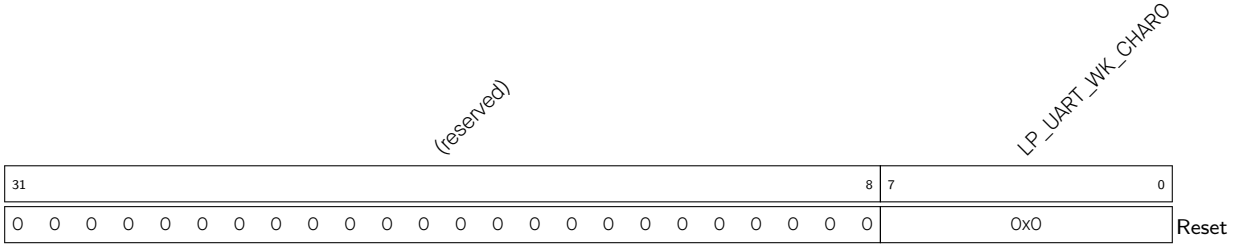
LP_UART_WK_CHAR1 配置唤醒芯片的特定字符 1。(R/W)

LP_UART_WK_CHAR2 配置唤醒芯片的特定字符 2。(R/W)

LP_UART_WK_CHAR3 配置唤醒芯片的特定字符 3。(R/W)

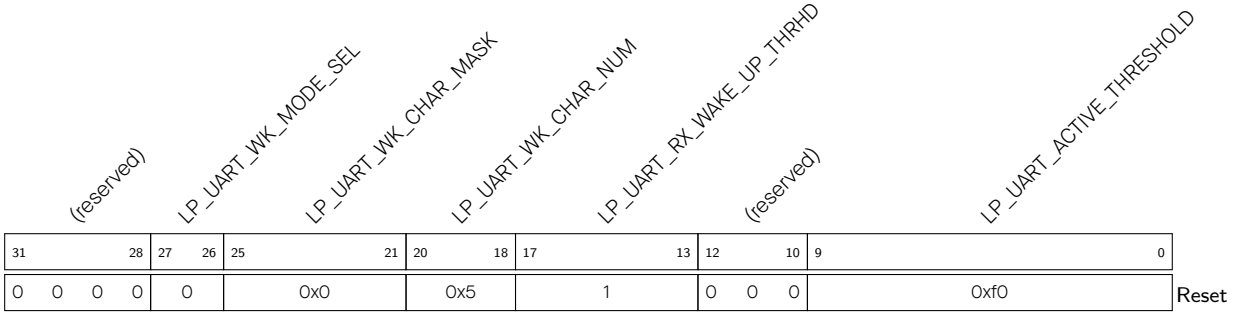
LP_UART_WK_CHAR4 配置唤醒芯片的特定字符 4。(R/W)

Register 32.50. LP_UART_SLEEP_CONF1_REG (0x0034)



LP_UART_WK_CHAR0 配置唤醒芯片的特定字符 0。(R/W)

Register 32.51. LP_UART_SLEEP_CONF2_REG (0x0038)



LP_UART_ACTIVE_THRESHOLD 配置唤醒模式 0 唤醒芯片所需的 RXD 沿变化次数。(R/W)

LP_UART_RX_WAKE_UP_THRHD 配置唤醒模式 1 唤醒芯片所需接收的数据量。

单位：字节。(R/W)

LP_UART_WK_CHAR_NUM 配置选取多少个唤醒字符。(R/W)

LP_UART_WK_CHAR_MASK 配置是否屏蔽唤醒字符。

0：不屏蔽

1：屏蔽

(R/W)

LP_UART_WK_MODE_SEL 配置选取哪个唤醒模式。

0：唤醒模式 0

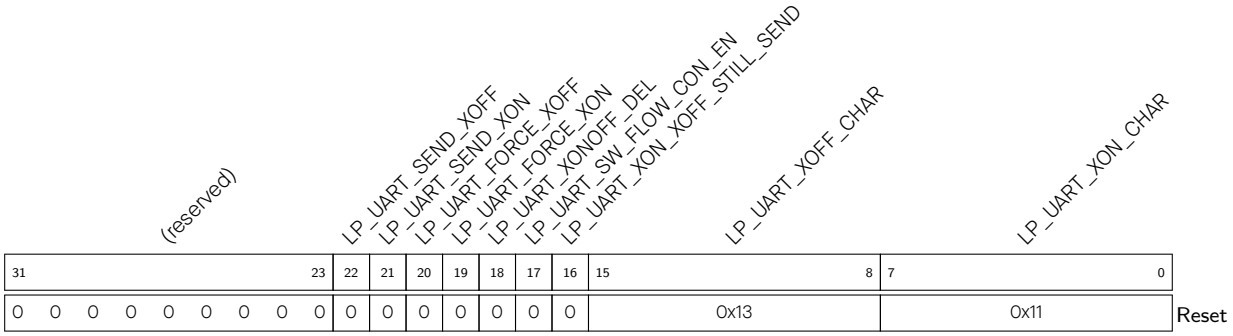
1：唤醒模式 1

2：唤醒模式 2

3：唤醒模式 3

(R/W)

Register 32.52. LP_UART_SWFC_CONF0_SYNC_REG (0x003C)



LP_UART_XON_CHAR 配置 XON 流控字符。(R/W)

LP_UART_XOFF_CHAR 配置 XOFF 流控字符。(R/W)

LP_UART_XON_XOFF_STILL_SEND 配置 LP UART 发送器关闭时，是否仍能发送 XON 或 XOFF 字符。

0：不能发送

1：可以发送

(R/W)

LP_UART_SW_FLOW_CON_EN 配置是否开启软件流控。

0：关闭

1：开启

(R/W)

LP_UART_XONOFF_DEL 配置是否移除接收数据中的流控字符。

0：不移除

1：移除

(R/W)

LP_UART_FORCE_XON 配置是否让发送器继续发送数据。

0：不发送

1：继续发送

(R/W)

LP_UART_FORCE_XOFF 配置是否让发送器停止发送数据。

0：不停止

1：停止

(R/W)

LP_UART_SEND_XON 配置是否发送 XON 字符。

0：不发送

1：发送

(R/W/SS/SC)

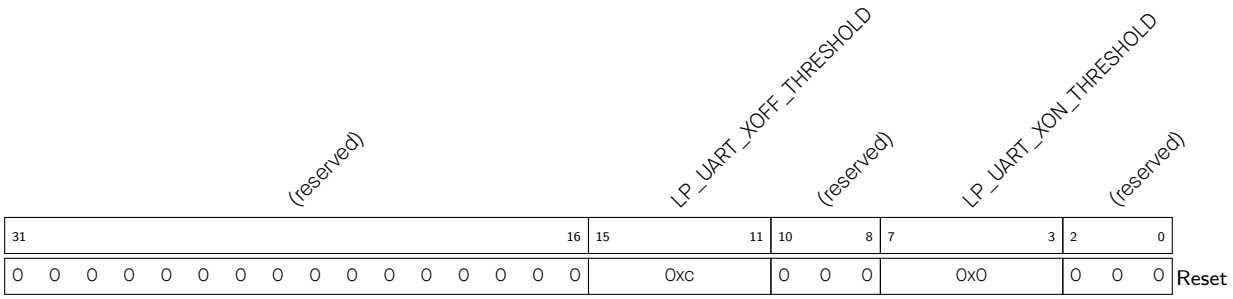
LP_UART_SEND_XOFF 配置是否发送 XOFF 字符。

0：不发送

1：发送

(R/W/SS/SC)

Register 32.53. LP_UART_SWFC_CONF1_REG (0x0040)



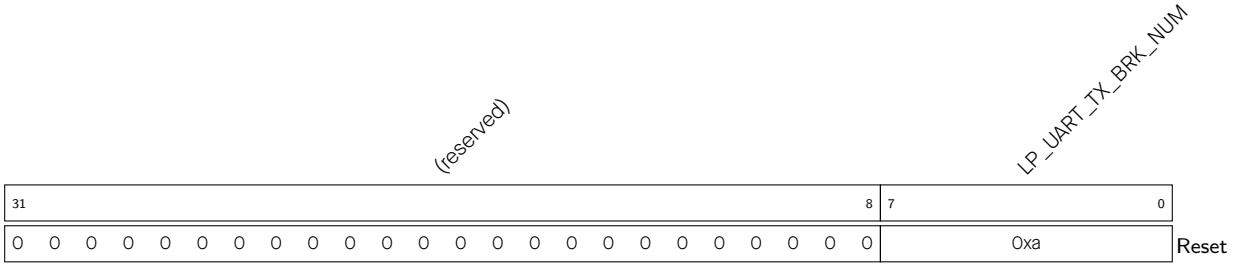
LP_UART_XON_THRESHOLD 配置软件流控时，发送 XON 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

LP_UART_XOFF_THRESHOLD 配置软件流控时，发送 XOFF 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

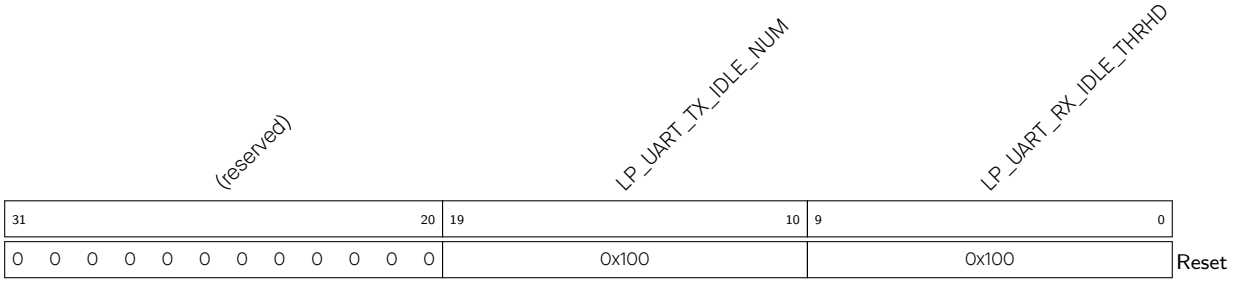
Register 32.54. LP_UART_TXBRK_CONF_SYNC_REG (0x0044)



LP_UART_TX_BRK_NUM 配置完成数据发送后待发 NULL 字符的数量。

仅在 LP_UART_TXD_BRK 为 1 时有效。(R/W)

Register 32.55. LP_UART_IDLE_CONF_SYNC_REG (0x0048)



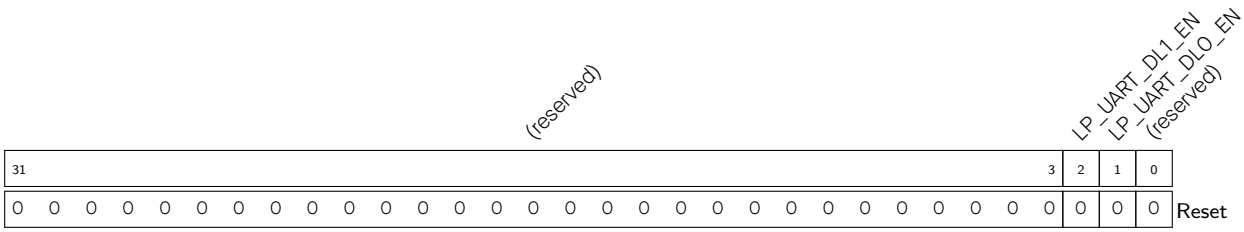
LP_UART_RX_IDLE_THRHD 配置接收一字节数据时间过长、产生帧结束信号的阈值。

单位：位时间（传输一位所需的时间）。(R/W)

LP_UART_TX_IDLE_NUM 配置两次数据传输的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 32.56. LP_UART_DELAY_CONF_SYNC_REG (0x004C)



LP_UART_DL0_EN 配置是否在起始位之前增加一位停止位。

0: 不增加

1: 增加

(R/W)

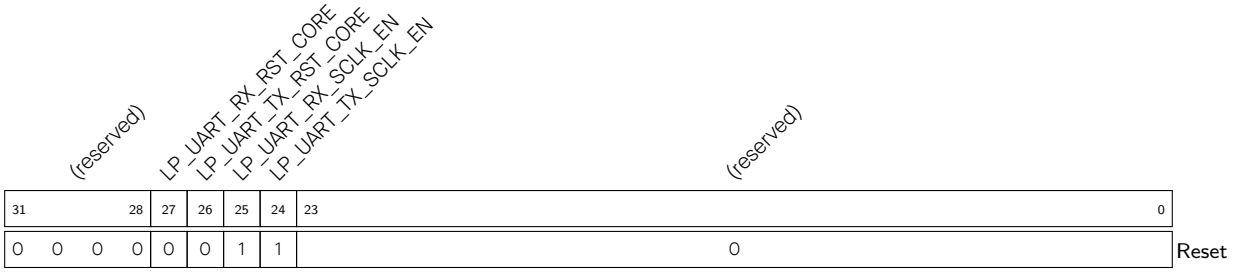
LP_UART_DL1_EN 配置是否在停止位之后增加一位停止位。

0: 不增加

1: 增加

(R/W)

Register 32.57. LP_UART_CLK_CONF_REG (0x0088)



LP_UART_TX_SCLK_EN 配置是否开启 LP UART TX 时钟。

0：关闭

1：开启

(R/W)

LP_UART_RX_SCLK_EN 配置是否开启 LP UART RX 时钟。

0：关闭

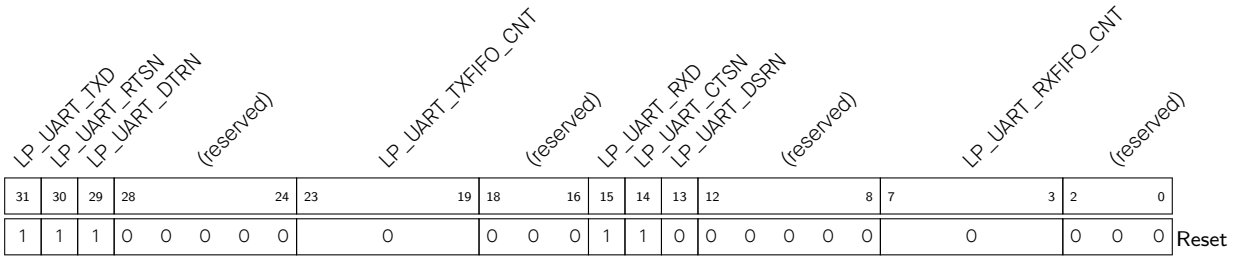
1：开启

(R/W)

LP_UART_TX_RST_CORE 向此位先写 1 后写 0，复位 LP UART TX。(R/W)

LP_UART_RX_RST_CORE 向此位先写 1 后写 0，复位 LP UART RX。(R/W)

Register 32.58. LP_UART_STATUS_REG (0x001C)



LP_UART_RXFIFO_CNT 表示 RX FIFO 中有效数据量。

单位：字节。(RO)

LP_UART_DSRN 表示内部 LP UART DSR 信号的电平值。(RO)

LP_UART_CTSN 表示内部 LP UART CTS 信号的电平值。(RO)

LP_UART_RXD 表示内部 LP UART RXD 信号的电平值。(RO)

LP_UART_TXFIFO_CNT 表示 TX FIFO 中的数据量。

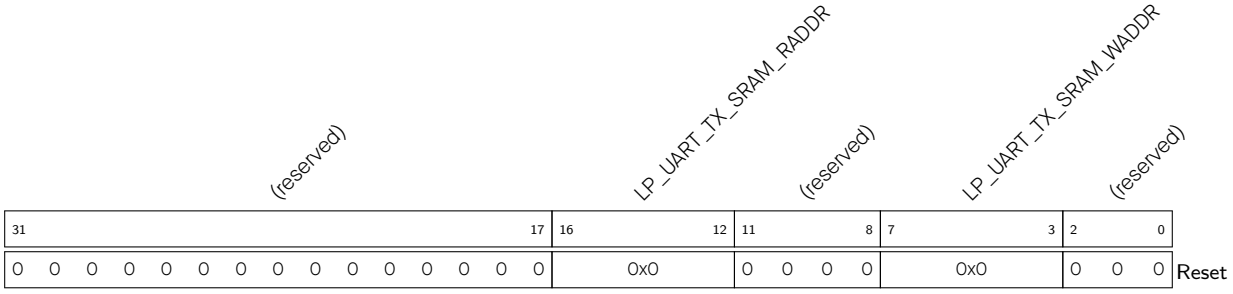
单位：字节。(RO)

LP_UART_DTRN 表示内部 LP UART DTR 信号的电平。(RO)

LP_UART_RTSN 表示内部 LP UART RTS 信号的电平。(RO)

LP_UART_TXD 表示内部 LP UART TXD 信号的电平。(RO)

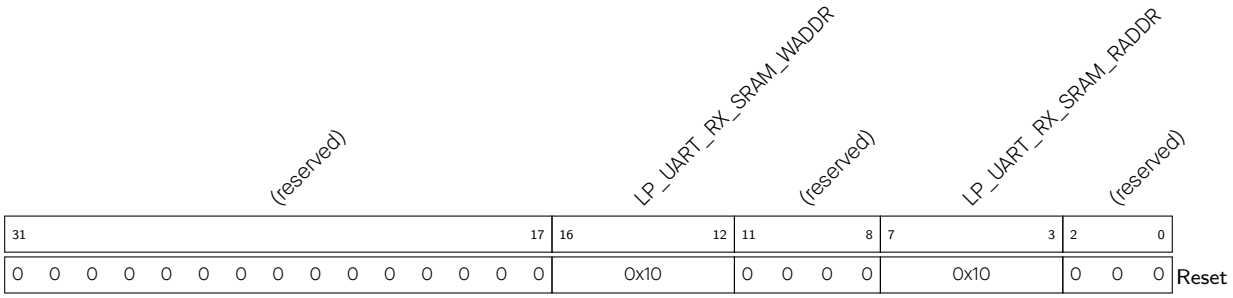
Register 32.59. LP_UART_MEM_TX_STATUS_REG (0x0068)



LP_UART_TX_SRAM_WADDR 表示写 TX FIFO 的偏移地址。(RO)

LP_UART_TX_SRAM_RADDR 表示读 TX FIFO 的偏移地址。(RO)

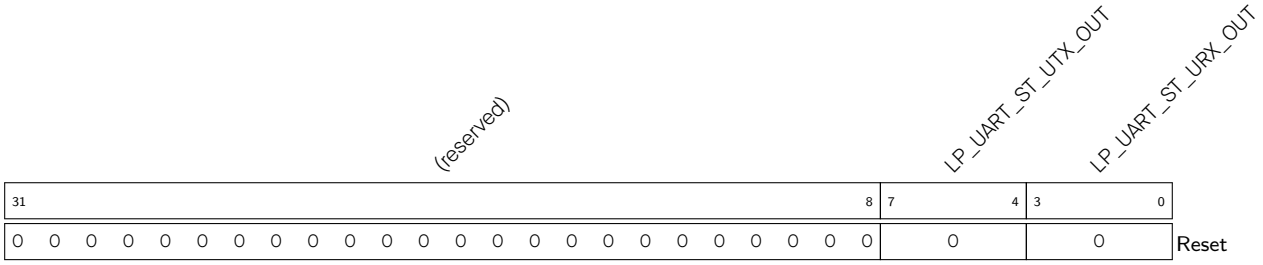
Register 32.60. LP_UART_MEM_RX_STATUS_REG (0x006C)



LP_UART_RX_SRAM_RADDR 表示读 RX FIFO 的偏移地址。(RO)

LP_UART_RX_SRAM_WADDR 表示写 RX FIFO 的偏移地址。(RO)

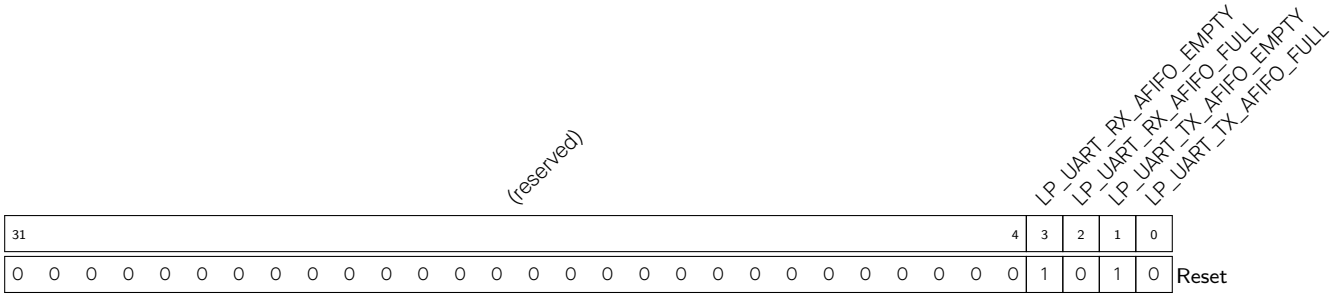
Register 32.61. LP_UART_FSM_STATUS_REG (0x0070)



LP_UART_ST_URX_OUT 表示 LP UART 接收器的状态。(RO)

LP_UART_ST_UTX_OUT 表示 LP UART 发送器的状态。(RO)

Register 32.62. LP_UART_AFIFO_STATUS_REG (0x0090)



LP_UART_TX_AFIFO_FULL 表示 APB TX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

LP_UART_TX_AFIFO_EMPTY 表示 APB TX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

LP_UART_RX_AFIFO_FULL 表示 APB RX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

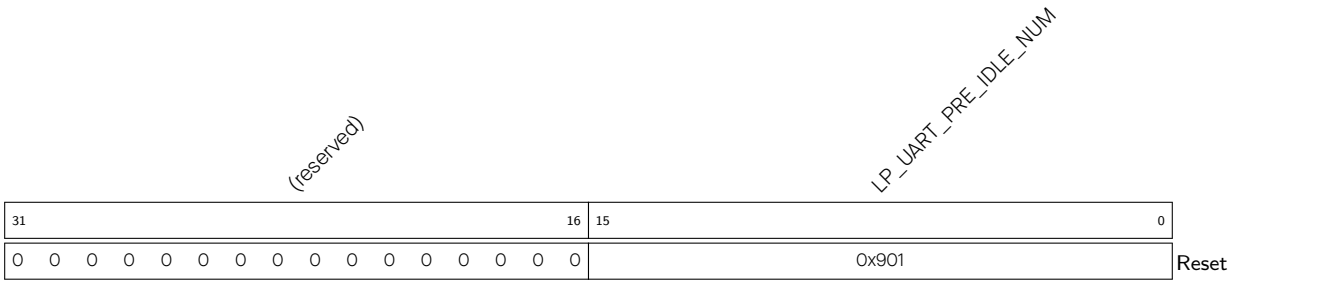
LP_UART_RX_AFIFO_EMPTY 表示 APB RX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

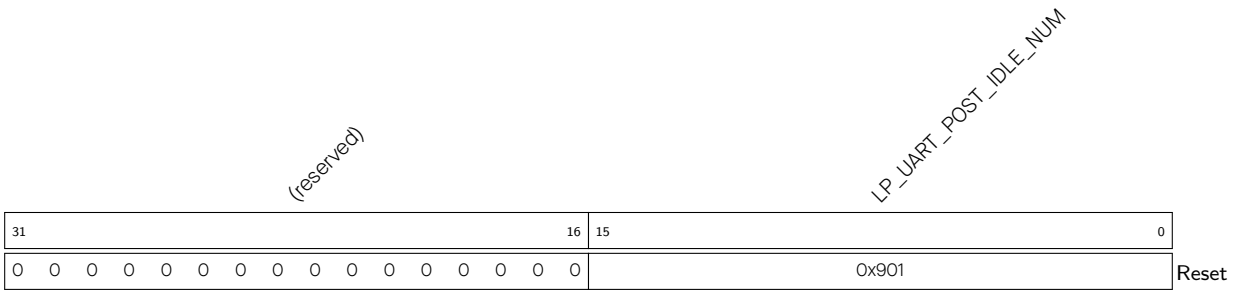
Register 32.63. LP_UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



LP_UART_PRE_IDLE_NUM 配置接收器接收到第一个 AT_CMD 之前的空闲时间。

单位：位时间（传输一位所需的时间）。(R/W)

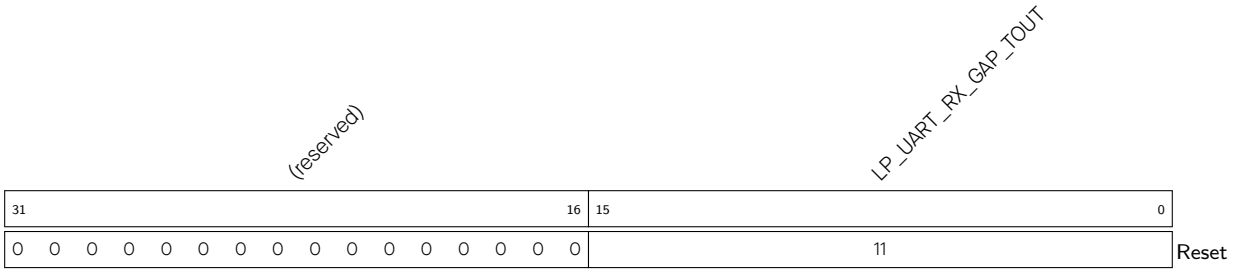
Register 32.64. LP_UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



LP_UART_POST_IDLE_NUM 配置最后一个 AT_CMD 和后续数据之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

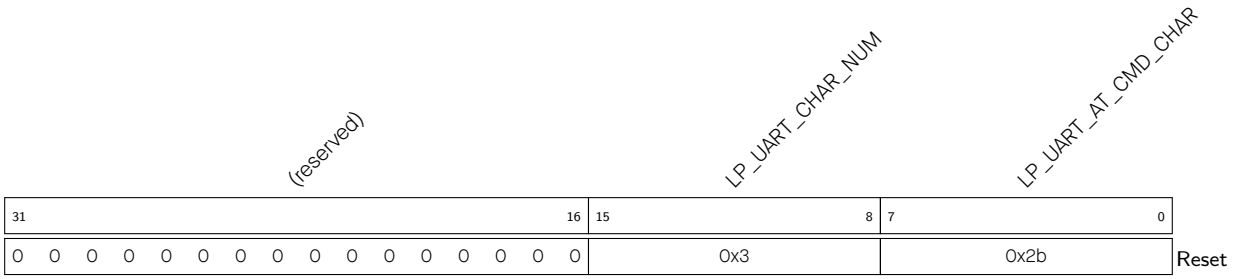
Register 32.65. LP_UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



LP_UART_RX_GAP_TOUT 配置 AT_CMD 之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

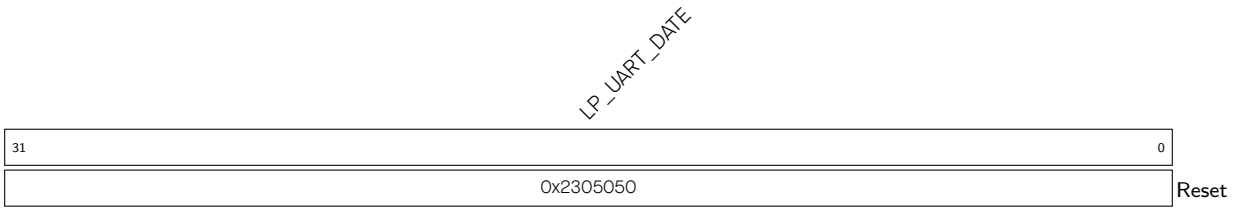
Register 32.66. LP_UART_AT_CMD_CHAR_SYNC_REG (0x005C)



LP_UART_AT_CMD_CHAR 配置 AT_CMD 字符。(R/W)

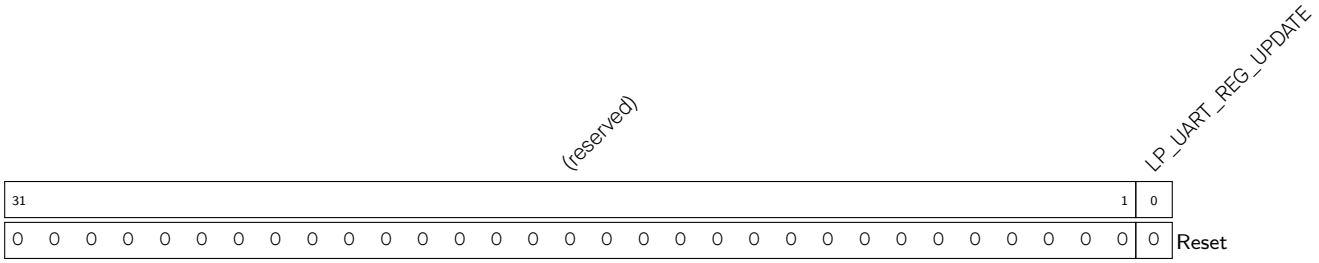
LP_UART_CHAR_NUM 配置接收器可连续接收的 AT_CMD 字符数量。(R/W)

Register 32.67. LP_UART_DATE_REG (0x008C)



LP_UART_DATE 版本控制寄存器。(R/W)

Register 32.68. LP_UART_REG_UPDATE_REG (0x0098)



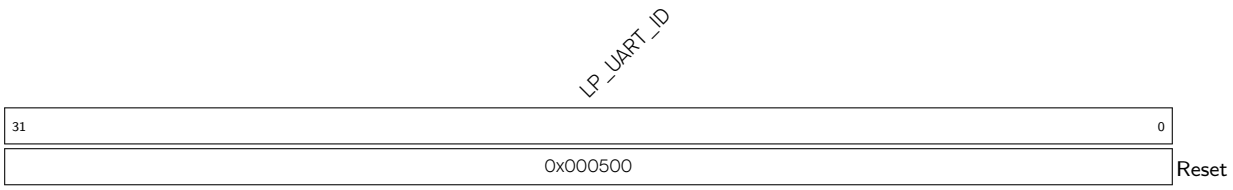
LP_UART_REG_UPDATE 配置是否同步寄存器

0：不同步

1：同步

(R/W/SC)

Register 32.69. LP_UART_ID_REG (0x009C)



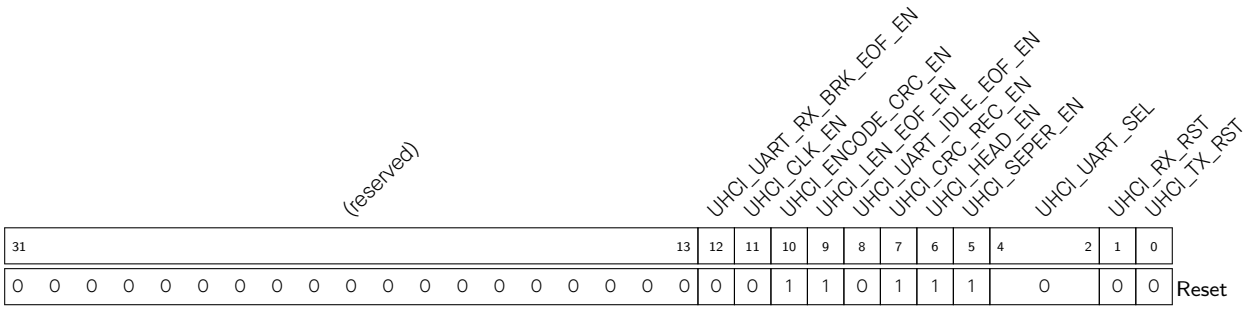
LP_UART_ID 配置 LP UART ID。(R/W)

32.8.3 UHCI 寄存器

本小节的所有地址均为相对于 UHCI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 32.70. UHCI_CONF0_REG (0x0000)



UHCI_TX_RST 先写 1 再写 0 复位解码器状态机。(R/W)

UHCI_RX_RST 先写 1 再写 0 复位编码器状态机。(R/W)

UHCI_UART_SEL 从 UART 0 ~ 1 中选择其一连接到 UHCI。

0：选择 UART0

1：选择 UART1

2 ~ 7：无效值，没有 UART 连接到 UHCI

(R/W)

UHCI_SEPER_EN 配置是否使用特殊字符分隔数据帧。

0：不分隔

1：分隔

(R/W)

UHCI_HEAD_EN 配置是否用格式报头编码数据包。

0：不使用报头

1：使用报头

(R/W)

UHCI_CRC_REC_EN 配置 UHCI 是否开启接收 16 位 CRC。

0：关闭

1：开启

(R/W)

见下页...

Register 32.70. UHCI_CONF0_REG (0x0000)

接上页...

UHCI_UART_IDLE_EOF_EN 配置 UART 空闲时 UHCI 是否停止接收数据。

0：不停止

1：停止

(R/W)

UHCI_LEN_EOF_EN 配置 UHCI 解码器停止接收数据的条件。

0：接收到 0xC0 后停止

1：接收字节数达到指定值时停止。UHCI_HEAD_EN 为 1 时，指定值是 UCHI 数据包报头指定的数据长度。UHCI_HEAD_EN 为 0 时，指定值为配置值。

(R/W)

UHCI_ENCODE_CRC_EN 配置是否开启数据完整性检测，在数据末尾加 16 位 CCITT-CRC。

0：关闭

1：开启

(R/W)

UHCI_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

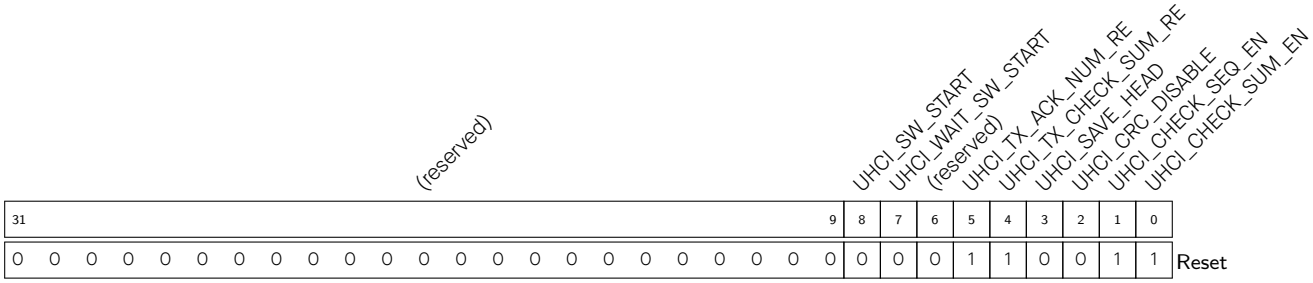
UHCI_UART_RX_BRK_EOF_EN 配置 UART 收到 NULL 帧后，UHCI 是否停止接收数据。

0：不停止

1：停止

(R/W)

Register 32.71. UHCI_CONF1_REG (0x0014)



UHCI_CHECK_SUM_EN 配置 UHCI 接收数据包时，是否开启报头校验和检查。

0：关闭

1：开启

(R/W)

UHCI_CHECK_SEQ_EN 配置 UHCI 接收数据包时，是否开启序列号检查。

0：关闭

1：开启

(R/W)

UHCI_CRC_DISABLE 配置是否开启 CRC 计算。

0：关闭

1：开启

仅在 UHCI 包中的数据完整性检测位为 1 时有效。

(R/W)

UHCI_SAVE_HEAD 配置 UHCI 接收数据包时，是否保存数据包报头。

0：不保存

1：保存

(R/W)

UHCI_TX_CHECK_SUM_RE 配置用校验和编码数据包。

0：不使用

1：使用

(R/W)

UHCI_TX_ACK_NUM_RE 配置准备发送可靠数据包时，是否用 ACK 编码该数据包。

0：不使用

1：使用

(R/W)

UHCI_WAIT_SW_START 配置是否将 UHCI 编码器的状态机调至 ST_SW_WAIT 状态。

0：否

1：是

(R/W)

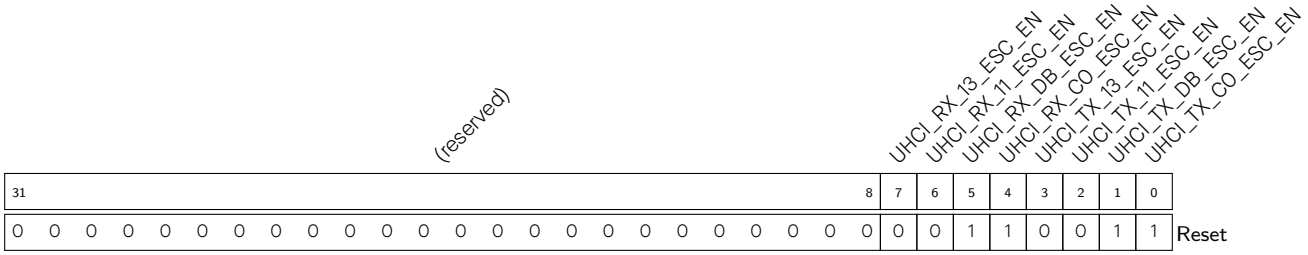
UHCI_SW_START 编码器的状态机为 ST_SW_WAIT 状态时，配置 UHCI 是否开始发送数据包。

0：不发送

1：开始发送

(R/W/SC)

Register 32.72. UHCI_ESCAPE_CONF_REG (0x0020)



UHCI_TX_C0_ESC_EN 配置 DMA 接收数据时，是否开启 0xC0 字符解码。

0：关闭

1：开启

(R/W)

UHCI_TX_DB_ESC_EN 配置 DMA 接收数据时，是否开启 0xDB 字符解码。

0：关闭

1：开启

(R/W)

UHCI_TX_11_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x11 解码。

0：关闭

1：开启

(R/W)

UHCI_TX_13_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x13 解码。

0：关闭

1：开启

(R/W)

UHCI_RX_C0_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换 0xC0。

0：不替换

1：替换

(R/W)

UHCI_RX_DB_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换 0xDB。

0：不替换

1：替换

(R/W)

UHCI_RX_11_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x11。

0：不替换

1：替换

(R/W)

UHCI_RX_13_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x13。

0：不替换

1：替换

(R/W)

Register 32.73. UHCI_HUNG_CONF_REG (0x0024)

(reserved)	UHCI_RXFIFO_TIMEOUT_ENA	UHCI_RXFIFO_TIMEOUT_SHIFT	UHCI_RXFIFO_TIMEOUT	UHCI_TXFIFO_TIMEOUT_ENA	UHCI_TXFIFO_TIMEOUT_SHIFT	UHCI_TXFIFO_TIMEOUT	
31 24	23	22 20	19 12	11	10 8	7 0	
0 0 0 0 0 0 0 0	1	0	0x10	1	0	0x10	Reset

UHCI_TXFIFO_TIMEOUT 配置 DMA 接收数据的超时值。
单位：毫秒。(R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT 配置 TX FIFO 超时计数器的阈值上限。(R/W)

UHCI_TXFIFO_TIMEOUT_ENA 配置是否开启 TX FIFO 接收数据超时。
0：关闭
1：开启
(R/W)

UHCI_RXFIFO_TIMEOUT 配置 DMA 从 RAM 读取数据的超时值。单位：毫秒。(R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT 配置 RX FIFO 超时计数器的阈值上限。(R/W)

UHCI_RXFIFO_TIMEOUT_ENA 配置是否开启 DMA 发送数据超时。
0：关闭
1：开启
(R/W)

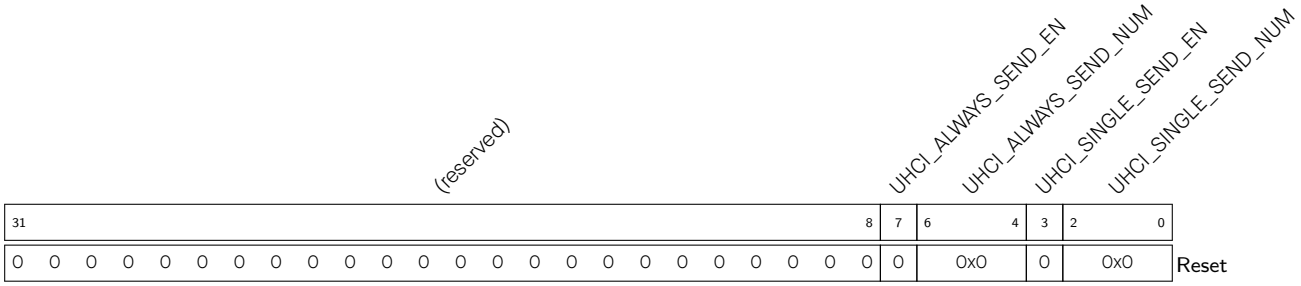
Register 32.74. UHCI_ACK_NUM_REG (0x0028)

(reserved)	UHCI_ACK_NUM_LOAD	UHCI_ACK_NUM	
31 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0x0	Reset

UHCI_ACK_NUM 配置软件流控中使用的 ACK 数量。(R/W)

UHCI_ACK_NUM_LOAD 配置是否加载 ACK。
0：不加载
1：加载
(WT)

Register 32.75. UHCI_QUICK_SENT_REG (0x0030)



UHCI_SINGLE_SEND_NUM 配置 single_send 模式下发送哪个寄存器中的数据。

0：Q0 寄存器
1：Q1 寄存器
2：Q2 寄存器
3：Q3 寄存器
4：Q4 寄存器
5：Q5 寄存器
6：Q6 寄存器
7：无效值，没有作用
(R/W)

UHCI_SINGLE_SEND_EN 配置是否开启 single_send 模式。

0：关闭
1：开启
(R/W/SC)

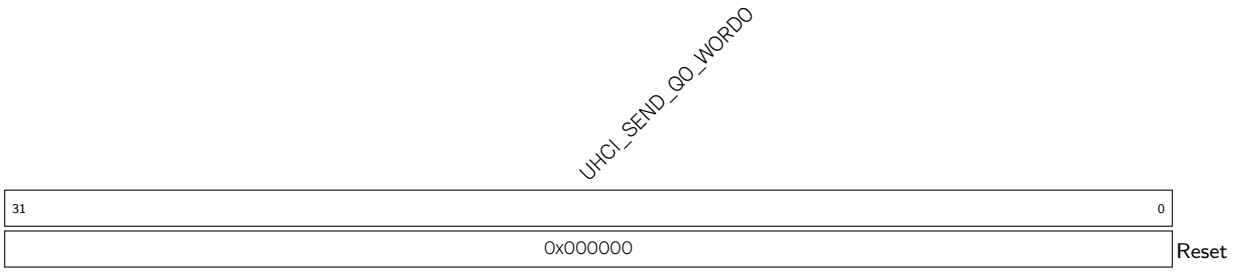
UHCI_ALWAYS_SEND_NUM 配置 always_send 模式下发送哪个寄存器中的数据。

0：Q0 寄存器
1：Q1 寄存器
2：Q2 寄存器
3：Q3 寄存器
4：Q4 寄存器
5：Q5 寄存器
6：Q6 寄存器
7：无效值，没有作用
(R/W)

UHCI_ALWAYS_SEND_EN 配置是否开启 always_send 模式。

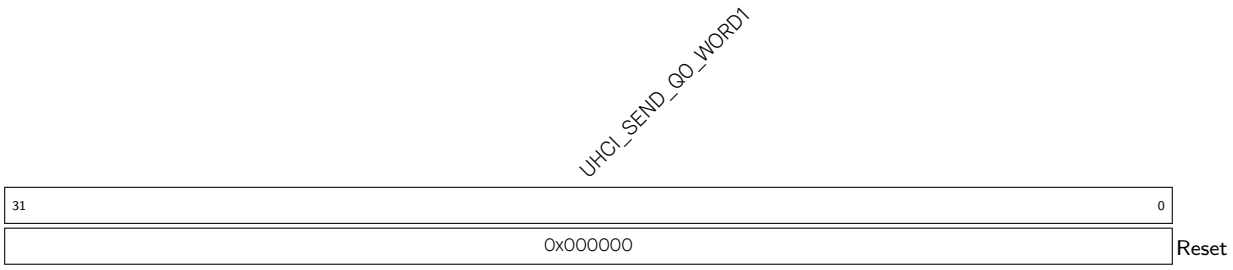
0：关闭
1：开启
(R/W)

Register 32.76. UHCI_REG_Q0_WORD0_REG (0x0034)



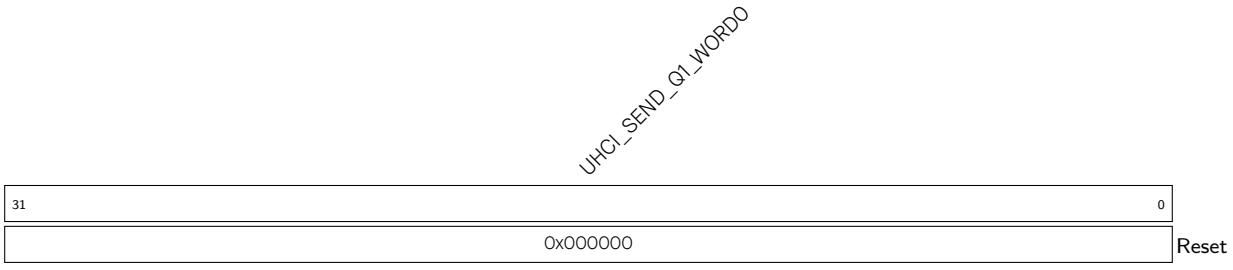
UHCI_SEND_Q0_WORD0 Q0 寄存器待发送数据。(R/W)

Register 32.77. UHCI_REG_Q0_WORD1_REG (0x0038)



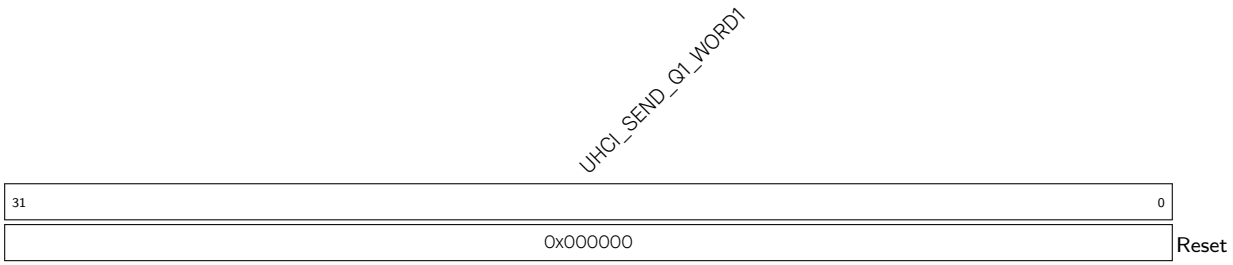
UHCI_SEND_Q0_WORD1 Q0 寄存器待发送数据。(R/W)

Register 32.78. UHCI_REG_Q1_WORD0_REG (0x003C)



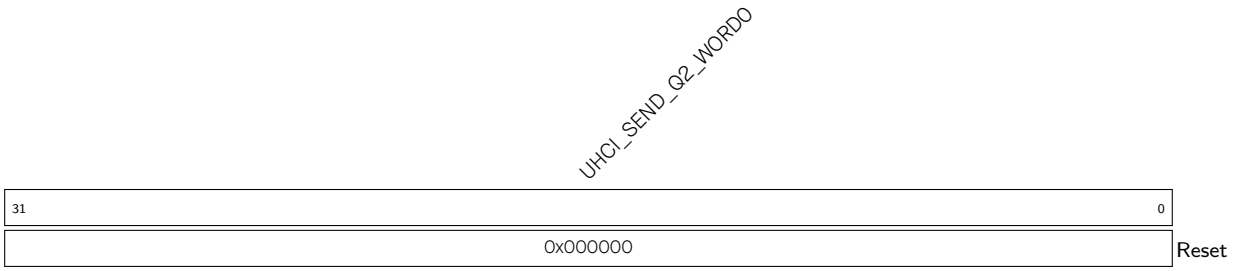
UHCI_SEND_Q1_WORD0 Q1 寄存器待发送数据。(R/W)

Register 32.79. UHCI_REG_Q1_WORD1_REG (0x0040)



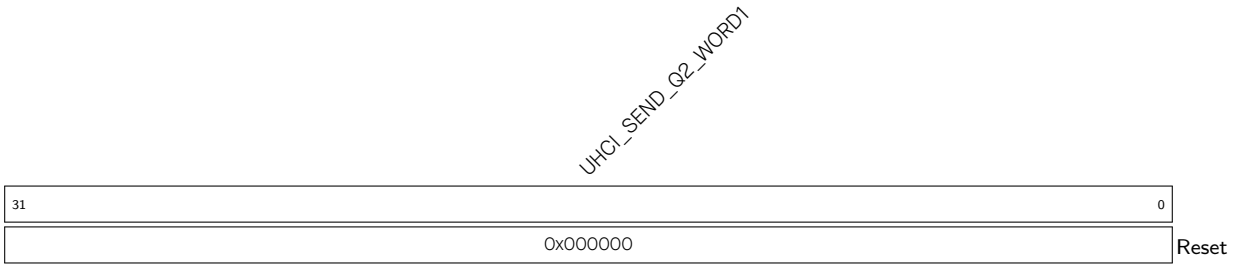
UHCI_SEND_Q1_WORD1 Q1 寄存器待发送数据。(R/W)

Register 32.80. UHCI_REG_Q2_WORD0_REG (0x0044)



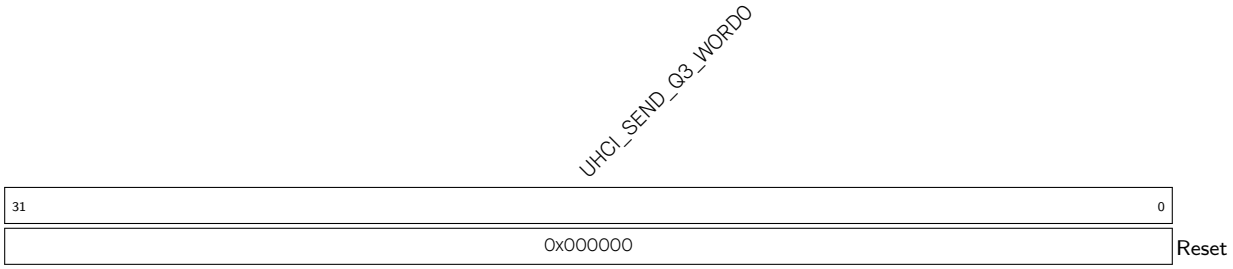
UHCI_SEND_Q2_WORD0 Q2 寄存器待发送数据。(R/W)

Register 32.81. UHCI_REG_Q2_WORD1_REG (0x0048)



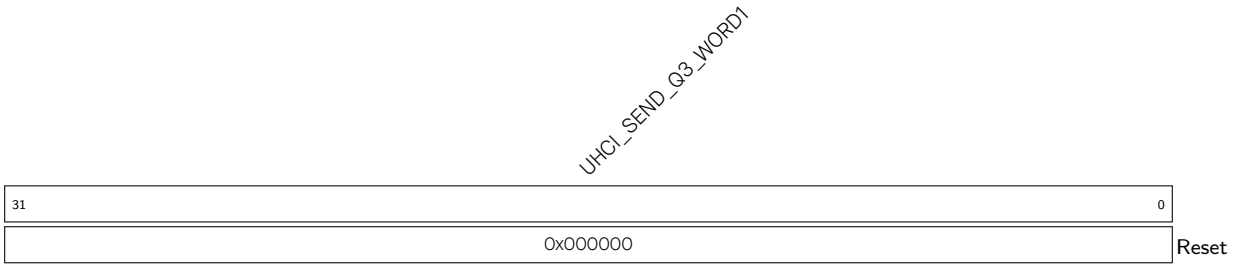
UHCI_SEND_Q2_WORD1 Q2 寄存器待发送数据。(R/W)

Register 32.82. UHCI_REG_Q3_WORD0_REG (0x004C)



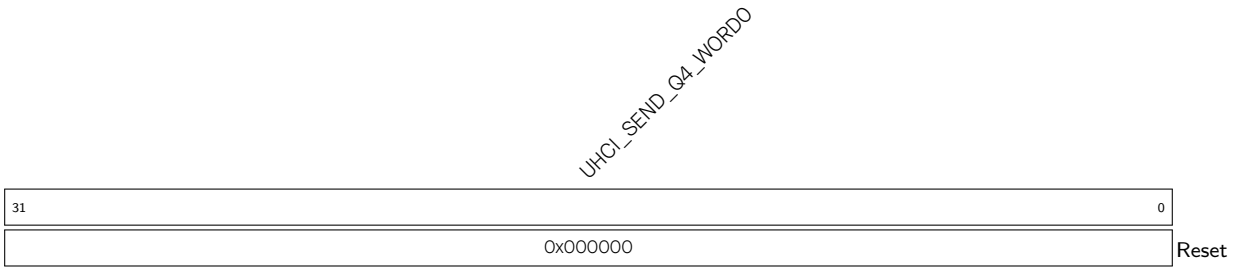
UHCI_SEND_Q3_WORD0 Q3 寄存器待发送数据。(R/W)

Register 32.83. UHCI_REG_Q3_WORD1_REG (0x0050)



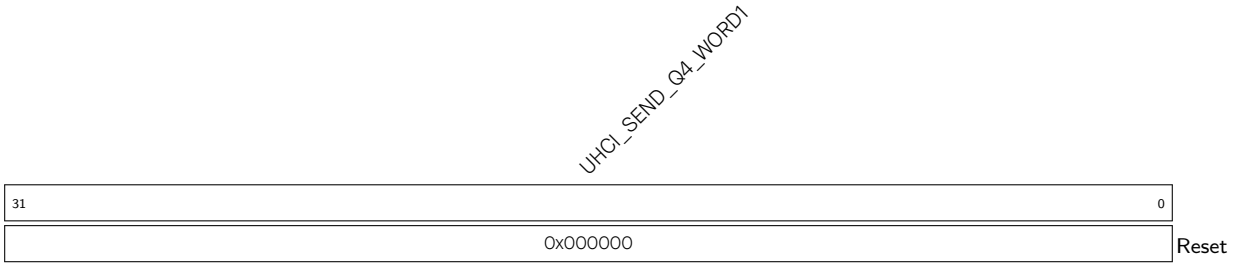
UHCI_SEND_Q3_WORD1 Q3 寄存器待发送数据。(R/W)

Register 32.84. UHCI_REG_Q4_WORD0_REG (0x0054)



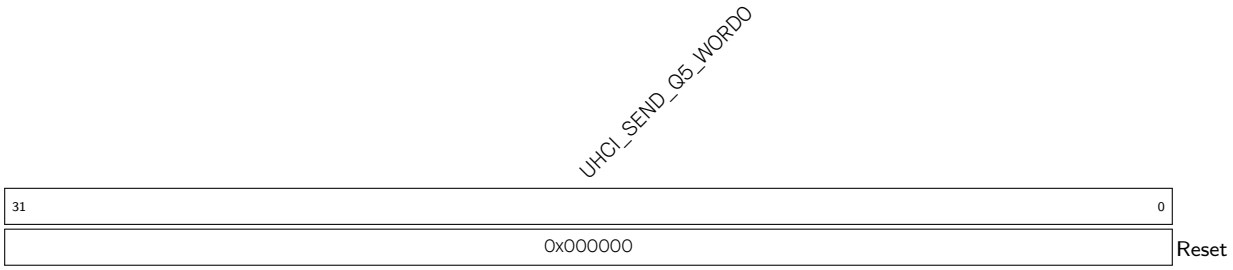
UHCI_SEND_Q4_WORD0 Q4 寄存器待发送数据。(R/W)

Register 32.85. UHCI_REG_Q4_WORD1_REG (0x0058)



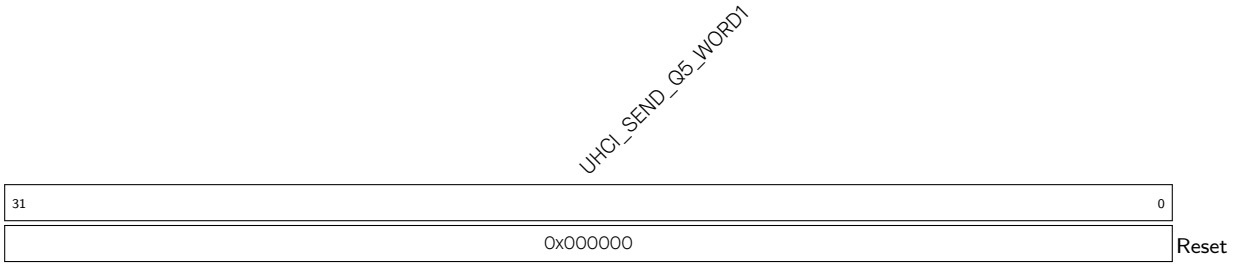
UHCI_SEND_Q4_WORD1 Q4 寄存器待发送数据。(R/W)

Register 32.86. UHCI_REG_Q5_WORD0_REG (0x005C)



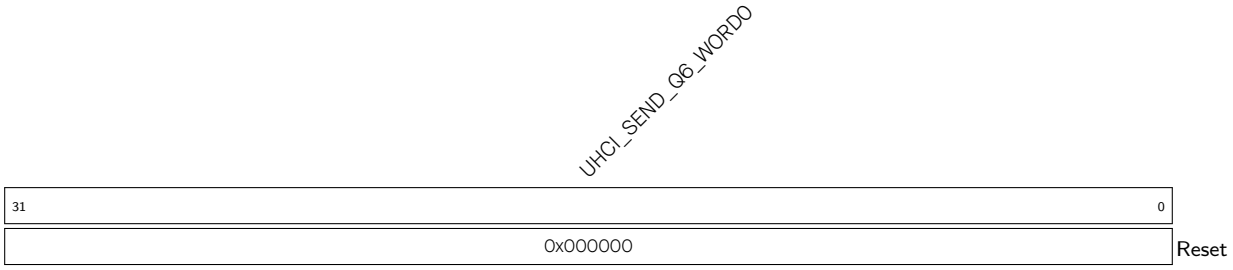
UHCI_SEND_Q5_WORD0 Q5 寄存器待发送数据。(R/W)

Register 32.87. UHCI_REG_Q5_WORD1_REG (0x0060)



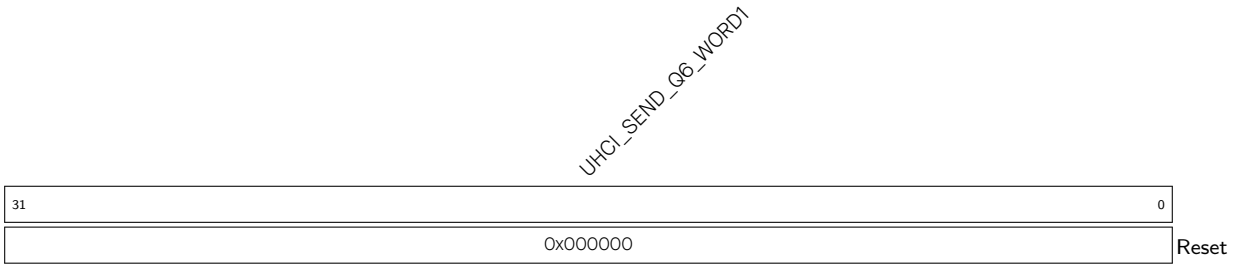
UHCI_SEND_Q5_WORD1 Q5 寄存器待发送数据。(R/W)

Register 32.88. UHCI_REG_Q6_WORD0_REG (0x0064)



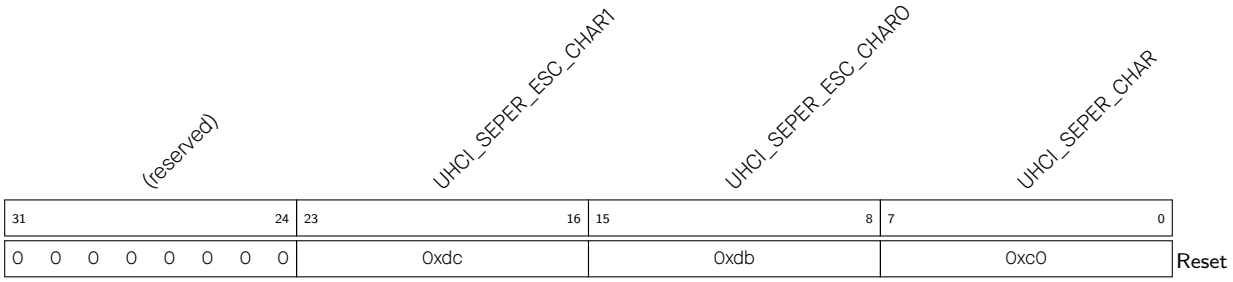
UHCI_SEND_Q6_WORD0 Q6 寄存器待发送数据。(R/W)

Register 32.89. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Q6 寄存器待发送数据。(R/W)

Register 32.90. UHCI_ESC_CONF0_REG (0x006C)

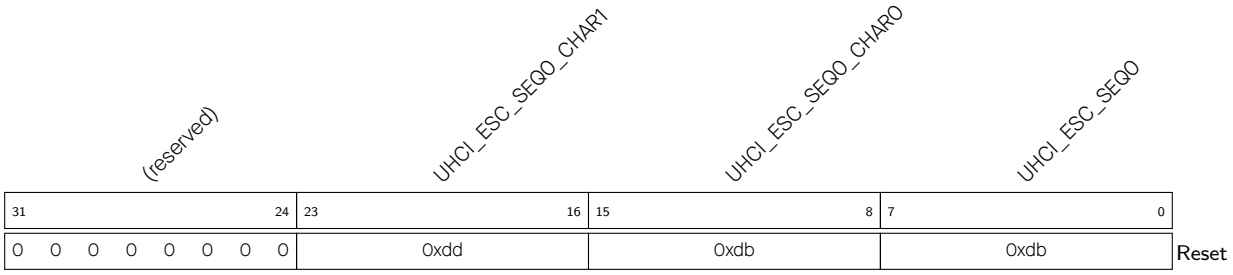


UHCI_SEPER_CHAR 配置编码数据包的分隔符，默认为 0xC0。(R/W)

UHCI_SEPER_ESC_CHAR0 配置 SLIP 转义序列的第一个字符，默认为 0xDB。(R/W)

UHCI_SEPER_ESC_CHAR1 配置 SLIP 转义序列的第二个字符，默认为 0xDC。(R/W)

Register 32.91. UHCI_ESC_CONF1_REG (0x0070)

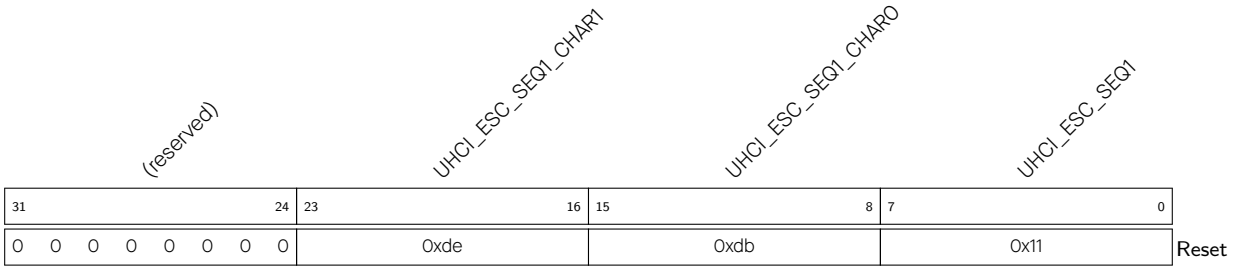


UHCI_ESC_SEQ0 配置需编码的字符，默认为用作 SLIP 转义序列第一个字符的 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ0 默认为 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ0 默认为 0xDD。(R/W)

Register 32.92. UHCI_ESC_CONF2_REG (0x0074)

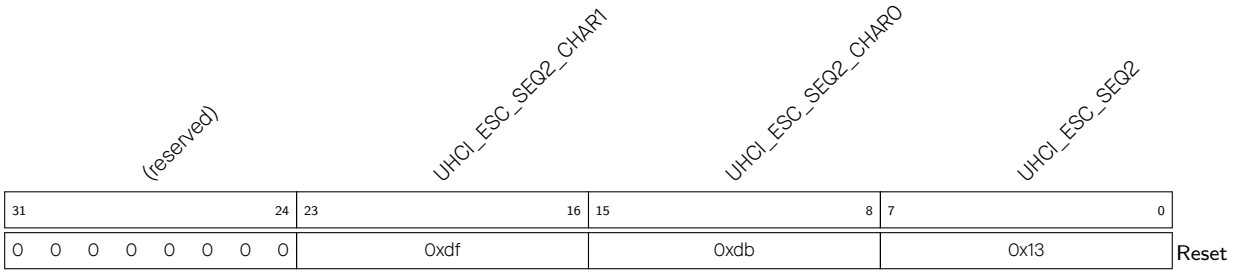


UHCI_ESC_SEQ1 配置需编码的字符，默认为流控字符的 0x11。(R/W)

UHCI_ESC_SEQ1_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ1 默认为 0xDB。(R/W)

UHCI_ESC_SEQ1_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ1 默认为 0xDE。(R/W)

Register 32.93. UHCI_ESC_CONF3_REG (0x0078)

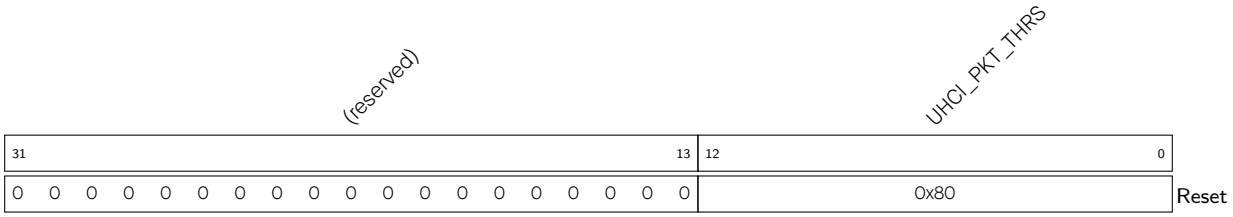


UHCI_ESC_SEQ2 配置需编码的字符，默认为流控字符的 0x13。(R/W)

UHCI_ESC_SEQ2_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ2 默认为 0xDB。(R/W)

UHCI_ESC_SEQ2_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ2 默认为 0xDF。(R/W)

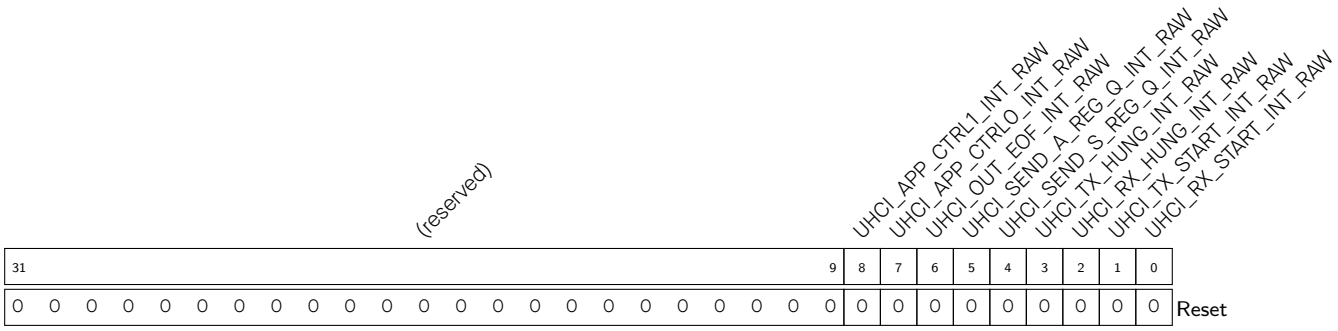
Register 32.94. UHCI_PKT_THRES_REG (0x007C)



UHCI_PKT_THRS 配置数据包的最大长度。

单位：字节。仅在 UHCI_HEAD_EN 为 0 时有效。(R/W)

Register 32.95. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW UHCI_RX_START_INT 的原始中断状态。(R/WTC/SS)

UHCI_TX_START_INT_RAW UHCI_TX_START_INT 的原始中断状态。(R/WTC/SS)

UHCI_RX_HUNG_INT_RAW UHCI_RX_HUNG_INT 的原始中断状态。(R/WTC/SS)

UHCI_TX_HUNG_INT_RAW UHCI_TX_HUNG_INT 的原始中断状态。(R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW UHCI_SEND_S_REG_Q_INT 的原始中断状态。(R/WTC/SS)

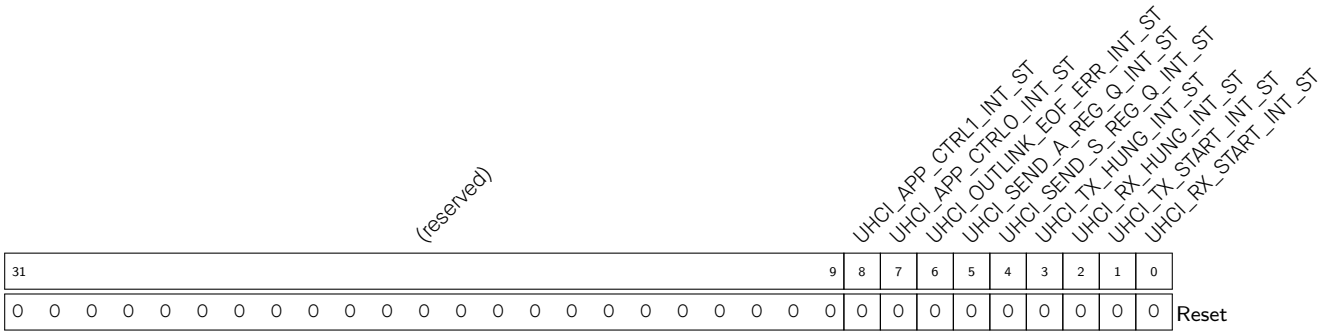
UHCI_SEND_A_REG_Q_INT_RAW UHCI_SEND_A_REG_Q_INT 的原始中断状态。(R/WTC/SS)

UHCI_OUT_EOF_INT_RAW UHCI_OUT_EOF_INT 的原始中断状态。(R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW UHCI_APP_CTRL0_INT 的原始中断状态。(R/W)

UHCI_APP_CTRL1_INT_RAW UHCI_APP_CTRL1_INT 的原始中断状态。(R/W)

Register 32.96. UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST UHCI_RX_START_INT 的屏蔽中断状态。(RO)

UHCI_TX_START_INT_ST UHCI_TX_START_INT 的屏蔽中断状态。(RO)

UHCI_RX_HUNG_INT_ST UHCI_RX_HUNG_INT 的屏蔽中断状态。(RO)

UHCI_TX_HUNG_INT_ST UHCI_TX_HUNG_INT 的屏蔽中断状态。(RO)

UHCI_SEND_S_REG_Q_INT_ST UHCI_SEND_S_REG_Q_INT 的屏蔽中断状态。(RO)

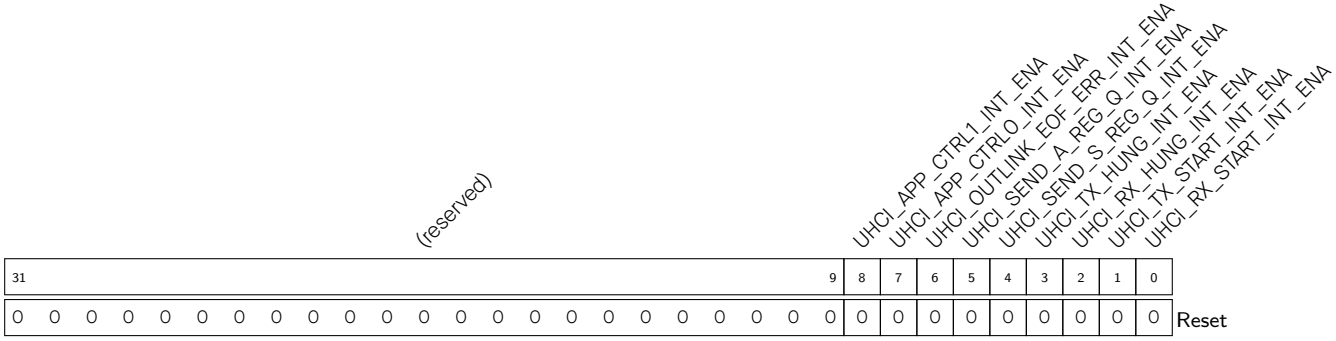
UHCI_SEND_A_REG_Q_INT_ST UHCI_SEND_A_REG_Q_INT 的屏蔽中断状态。(RO)

UHCI_OUTLINK_EOF_ERR_INT_ST UHCI_OUTLINK_EOF_ERR_INT 的屏蔽中断状态。(RO)

UHCI_APP_CTRL0_INT_ST UHCI_APP_CTRL0_INT 的屏蔽中断状态。(RO)

UHCI_APP_CTRL1_INT_ST UHCI_APP_CTRL1_INT 的屏蔽中断状态。(RO)

Register 32.97. UHCI_INT_ENA_REG (0x000C)



UHCI_RX_START_INT_ENA 写 1 使能 UHCI_RX_START_INT。(R/W)

UHCI_TX_START_INT_ENA 写 1 使能 UHCI_TX_START_INT。(R/W)

UHCI_RX_HUNG_INT_ENA 写 1 使能 UHCI_RX_HUNG_INT。(R/W)

UHCI_TX_HUNG_INT_ENA 写 1 使能 UHCI_TX_HUNG_INT。(R/W)

UHCI_SEND_S_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_S_REG_Q_INT。(R/W)

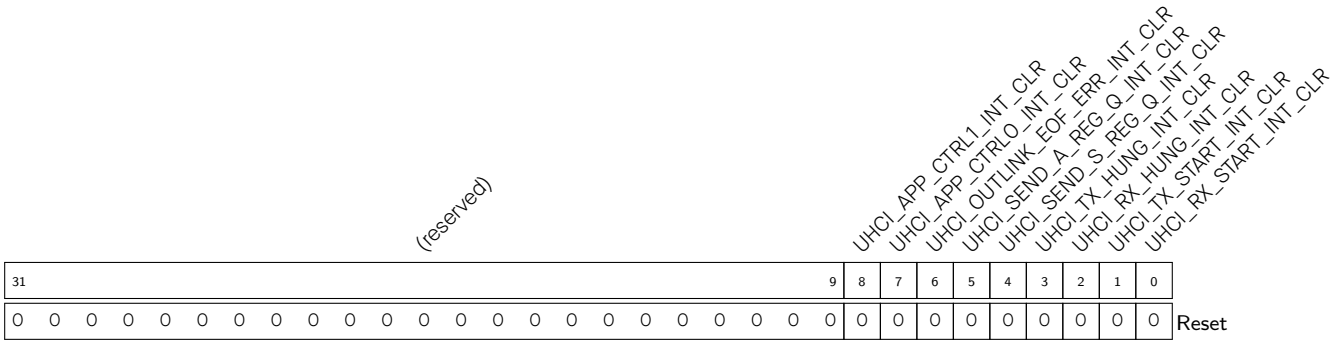
UHCI_SEND_A_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_A_REG_Q_INT。(R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA 写 1 使能 UHCI_OUTLINK_EOF_ERR_INT。(R/W)

UHCI_APP_CTRL0_INT_ENA 写 1 使能 UHCI_APP_CTRL0_INT。(R/W)

UHCI_APP_CTRL1_INT_ENA 写 1 使能 UHCI_APP_CTRL1_INT。(R/W)

Register 32.98. UHCI_INT_CLR_REG (0x0010)



UHCI_RX_START_INT_CLR 写 1 清除 UHCI_RX_START_INT。(WT)

UHCI_TX_START_INT_CLR 写 1 清除 UHCI_TX_START_INT。(WT)

UHCI_RX_HUNG_INT_CLR 写 1 清除 UHCI_RX_HUNG_INT。(WT)

UHCI_TX_HUNG_INT_CLR 写 1 清除 UHCI_TX_HUNG_INT。(WT)

UHCI_SEND_S_REG_Q_INT_CLR 写 1 清除 UHCI_SEND_S_REG_Q_INT。(WT)

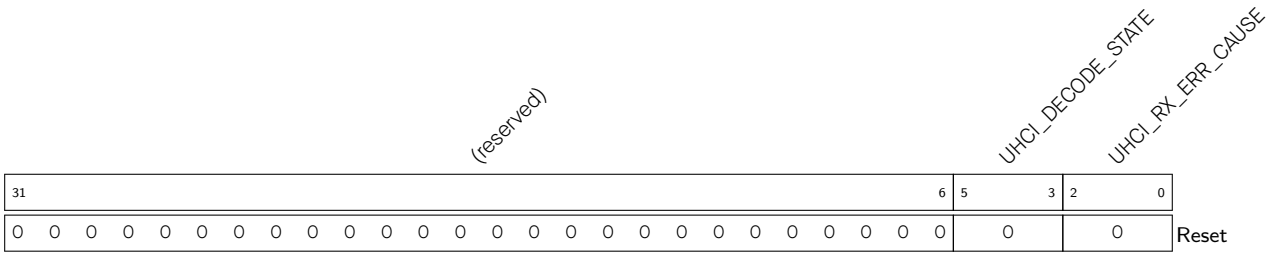
UHCI_SEND_A_REG_Q_INT_CLR 写 1 清除 UHCI_SEND_A_REG_Q_INT。(WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR 写 1 清除 UHCI_OUTLINK_EOF_ERR_INT。(WT)

UHCI_APP_CTRL0_INT_CLR 写 1 清除 UHCI_APP_CTRL0_INT。(WT)

UHCI_APP_CTRL1_INT_CLR 写 1 清除 UHCI_APP_CTRL1_INT。(WT)

Register 32.99. UHCI_STATE0_REG (0x0018)

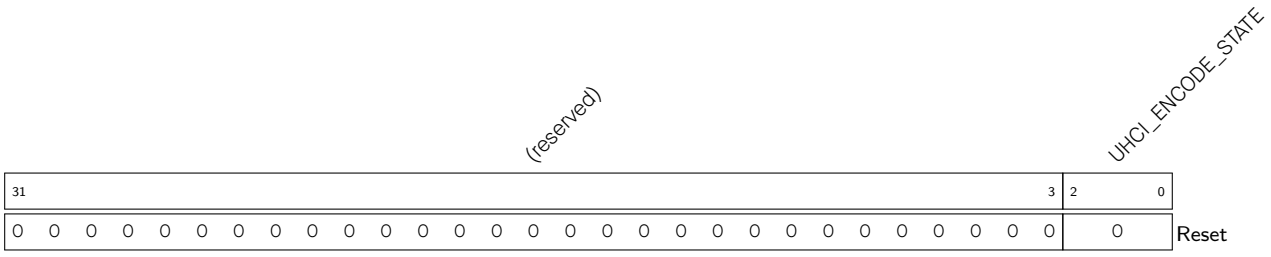


UHCI_RX_ERR_CAUSE 表示 DMA 接收数据错误的原因。

0：无效，没有作用
1：HCI 包校验和错误
2：HCI 包序列号错误
3：HCI 包 CRC 位错误
4：找到 0xC0 但接收的 HCI 包不完整
5：未找到 0xC0 但接收的 HCI 包完整
6：CRC 检测错误
7：无效，没有作用
(RO)

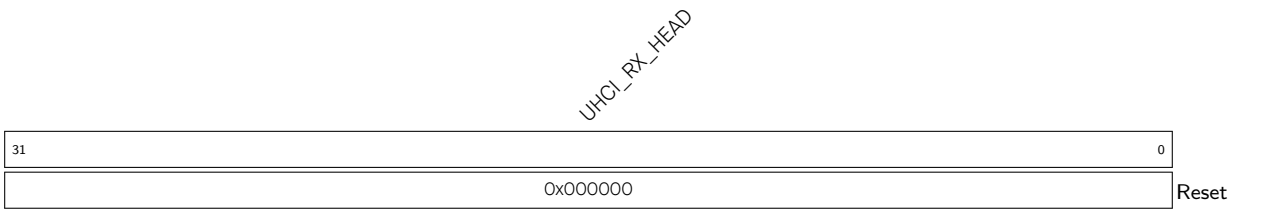
UHCI_DECODE_STATE 表示 UHCI 解码器状态。(RO)

Register 32.100. UHCI_STATE1_REG (0x001C)



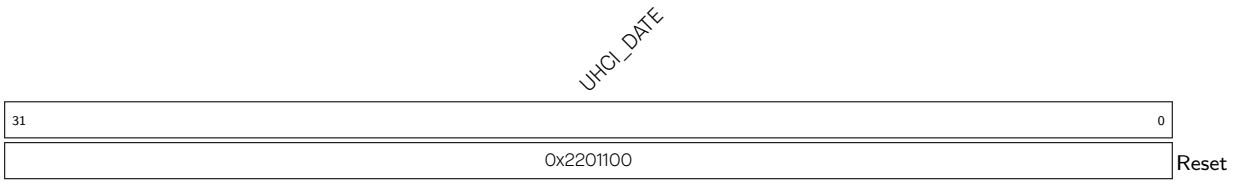
UHCI_ENCODE_STATE 表示 UHCI 编码器状态。(RO)

Register 32.101. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD 表示当前接收数据包的报头。(RO)

Register 32.102. UHCI_DATE_REG (0x0080)



UHCI_DATE 版本控制寄存器。(R/W)

第 33 章

SPI 控制器 (SPI)

33.1 概述

串行外设接口 (SPI) 是一种同步串行接口，可用于与外围设备进行通信。ESP32-C5 芯片集成了三个 SPI 控制器：

- SPI0
- SPI1
- 通用 SPI2，即 GP-SPI2

SPI0 和 SPI1 控制器 (MSPI) 主要供内部使用以访问外部 flash 及 PSRAM。因此，本章节将主要介绍 GP-SPI2 控制器。

33.2 术语

为了更好地说明 GP-SPI2 的功能，本章使用了以下术语。

主机模式	GP-SPI2 用作 SPI 主机，发起 SPI 传输事务。
从机模式	GP-SPI2 用作 SPI 从机，当其 CS 被拉低时，与 SPI 主机进行数据传输。
MISO	主机输入，从机输出。数据从从机发送至主机。
MOSI	主机输出，从机输入。数据从主机发送至从机。
传输事务	一次完整的传输事务：主机拉低从机的 CS 线，开始传输数据，然后再拉高从机的 CS 线。传输事务为原子操作，即不可打断。
SPI 传输	SPI 主机与从机完成数据交换的一次完整过程。一次 SPI 传输可以包含一个或多个 SPI 传输事务。
单次传输	在这种传输模式下，仅包含一次传输事务。
CPU 控制的传输	由 CPU 控制，SPI_W0_REG~SPI_W15_REG 与 SPI 设备之间的数据传输。
DMA 控制的传输	由 DMA 引擎控制，DMA 与 SPI 设备之间的数据传输。
分段配置传输	主机模式下，DMA 控制的数据传输。此类传输包含多个传输事务（分段），每个传输事务均可独立配置。
从机连续传输	从机模式下，DMA 控制的数据传输。此类传输包含多个传输事务（分段）。
全双工	主机与从机之间的发送线和接收线各自独立，发送数据和接收数据同时进行。
半双工	主机和从机只能有一方先发送数据，另一方接收数据。发送数据和接收数据不能同时进行。

四线全双工 四线包括：时钟线、片选线和两条数据线。其中，可使用两条数据线同时发送和接收数据。

四线半双工 四线包括：时钟线、片选线和两条数据线。其中，分时使用两条数据线，不可同时使用。

三线半双工 三线包括：时钟线、片选线和一条数据线。使用数据线分时发送和接收数据。

1-bit SPI 一个时钟周期传输一位数据。

(2-bit) Dual SPI 一个时钟周期传输两个数据位。

Dual Output Read Dual SPI 的一种数据模式，一个时钟周期可传输一位命令、或一位地址、或两位数据。

Dual I/O Read Dual SPI 的另外一种数据模式，一个时钟周期可传输一位命令、或两位地址、或两位数据。

(4-bit) Quad SPI 一个时钟周期传输四个数据位。

Quad Output Read Quad SPI 的一种数据模式，一个时钟周期可传输一位命令、或一位地址、或四位数据。

Quad I/O Read Quad SPI 的另一种数据模式，一个时钟周期可传输一位命令、或四位地址、或四位数据。

QPI 一个时钟周期可传输四位命令、或四位地址、或四位数据。

FSPI Fast SPI，GP-SPI2 输入输出信号的前缀。FSPI 总线信号可通过 GPIO 交换矩阵或 IO MUX 与 GPIO 管脚相连。

33.3 特性

GP-SPI2 具有以下特性：

- 用作主机或用作从机
- 支持半双工通信和全双工通信
- 支持 CPU 控制的传输类型以及 DMA 控制的传输类型
- 支持多种数据模式：
  - 1-bit SPI 模式
  - 2-bit Dual SPI 模式
  - 4-bit Quad SPI 模式
  - QPI 模式
- 时钟频率可配置：
  - 用作主机时：时钟频率可达 80 MHz
  - 用作从机时：时钟频率可达 40 MHz
- 数据长度可配置：
  - 在主机和从机 CPU 控制的传输下：数据长度为 1~64 B
  - 在主机 DMA 控制的单次传输下：数据长度为 1~32 KB
  - 在主机 DMA 控制的分段配置传输下：数据长度字节数无限制
  - 在从机 DMA 控制的单次或连续传输下：数据长度字节数无限制

- 读写数据的比特位顺序可配置
- 为 CPU 控制的传输和 DMA 控制的传输分别提供独立中断
- 时钟极性和相位可配置
- 四种 SPI 时钟模式：模式 0 ~ 模式 3
- 用作主机时，提供六条 CS 线：CS0~CS5
- 支持访问 SPI 接口的传感器、显示屏控制器、flash 或 RAM 芯片

33.4 架构概览

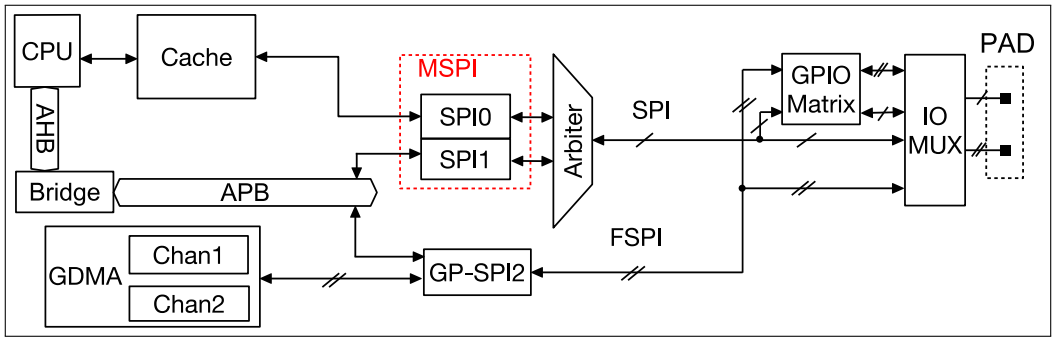


图 33.4-1. SPI 模块概览

图 33.4-1 所示为 SPI 模块的概览。GP-SPI2 通过以下方式与 SPI 设备进行数据交换：

- 在 CPU 控制的传输类型下：CPU ↔ GP-SPI2 ↔ SPI 设备
- 在 DMA 控制的传输类型下：GDMA ↔ GP-SPI2 ↔ SPI 设备

GP-SPI2 输入输出信号的前缀为 “FSPI”。FSPI 总线信号可通过 GPIO 交换矩阵或 IO MUX 与 GPIO 管脚相连。更多信息，见章节 *8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*。

33.5 功能描述

33.5.1 数据模式

GP-SPI2 可配置成主机或从机模式，采用表 33.5-1 所示的数据模式与其他 SPI 设备进行通信。

表 33.5-1. GP-SPI2 支持的数据模式

数据模式		命令阶段	地址阶段	数据阶段
1-bit SPI		1-bit	1-bit	1-bit
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit
	Dual I/O Read	1-bit	2-bit	2-bit
Quad SPI	Quad Output Read	1-bit	1-bit	4-bit
	Quad I/O Read	1-bit	4-bit	4-bit
QPI		4-bit	4-bit	4-bit

GP-SPI2 用作主机时，有效的阶段请参考第 33.5.9 小节；GP-SPI2 用作从机模式时，有效的阶段请参考第 33.5.10 小节。

33.5.2 FSPI 总线信号描述

FSPI 总线信号的功能描述如表 33.5-2。各种 SPI 模式下使用到的信号见表 33.5-3。

表 33.5-2. FSPI 总线信号功能描述

FSPI 总线信号	功能
FSPID	MOSI/SIO0[a]：串行输入输出数据，比特 0
FSPIQ	MISO/SIO1：串行输入输出数据，比特 1
FSPIWP	SIO2：串行输入输出数据，比特 2
FSPIHD	SIO3：串行输入输出数据，比特 3
FSPICLK	用作主机或从机时，输入输出时钟
FSPICS0	用作主机或从机时，输入输出片选信号
FSPICS1~5	用作主机时，输出片选信号

[a] SIO0：全称为 serial data input and output, bit0

表 33.5-3. 各种 SPI 模式下使用到的信号

FSPI 总线信号	用作主机						用作从机					
	1-bit SPI			2-bit Dual SPI	4-bit Quad SPI	QPI	1-bit SPI			2-bit Dual SPI	4-bit Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD				FD	3-line HD	4-line HD			
FSPICLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS1	Y	Y	Y	Y	Y	Y						
FSPICS2	Y	Y	Y	Y	Y	Y						
FSPICS3	Y	Y	Y	Y	Y	Y						
FSPICS4	Y	Y	Y	Y	Y	Y						
FSPICS5	Y	Y	Y	Y	Y	Y						
FSPID	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
FSPIQ	Y		(Y)[3]	Y[4]	Y[5]	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
FSPIWP					Y[5]	Y					Y[8]	Y
FSPIHD					Y[5]	Y					Y[8]	Y

[1] FD：全双工

[2] HD：半双工

[3] 一次只使用两个信号中的一个

[4] 两个信号并行使用

[5] 四个信号并行使用

[6] 一次只使用两个信号中的一个

[7] 两个信号并行使用

[8] 四个信号并行使用

33.5.3 数据位读/写顺序控制

用作主机时：

- GP-SPI2 主机发送的命令、地址和数据的位顺序由 SPI_WR_BIT_ORDER 控制；
- 接收数据的位顺序由 SPI_RD_BIT_ORDER 控制。

用作从机时：

- GP-SPI2 从机发送数据的位顺序由 SPI_WR_BIT_ORDER 控制；
- 从机接收的命令、地址和数据的位顺序由 SPI_RD_BIT_ORDER 控制。

表 33.5-4 所示为 SPI_RD/WR_BIT_ORDER 的功能。

表 33.5-4. GP-SPI2 用作主机和从机时的数据位控制

位模式	FSPI 总线信号	SPI_RD/WR_BIT_ORDER = 0 (MSB)	SPI_RD/WR_BIT_ORDER = 2 (MSB)	SPI_RD/WR_BIT_ORDER = 1 (LSB)	SPI_RD/WR_BIT_ORDER = 3 (LSB)
1-bit 模式	FSPID or FSPIQ	B7→B6→B5→B4→B3→B2→B1→B0	B7→B6→B5→B4→B3→B2→B1→B0	B0→B1→B2→B3→B4→B5→B6→B7	B0→B1→B2→B3→B4→B5→B6→B7
2-bit 模式	FSPIQ	B7→B5→B3→B1	B6→B4→B2→B0	B1→B3→B5→B7	B0→B2→B4→B6
	FSPID	B6→B4→B2→B0	B7→B5→B3→B1	B0→B2→B4→B6	B1→B3→B5→B7
4-bit 模式	FSPIHD	B7→B3	B4→B0	B3→B7	B0→B4
	FSPIWP	B6→B2	B5→B1	B2→B6	B1→B5
	FSPIQ	B5→B1	B6→B2	B1→B5	B2→B6
	FSPID	B4→B0	B7→B3	B0→B4	B3→B7

33.5.4 非字节对齐传输

- 用作主机时，GP-SPI2 主机可以按比特收发数据，比特长度等于 SPI_MS_DATA_BITLEN + 1，并且比特长度在 1/2/4/8 线模式下分别是 1/2/4/8 的倍数。
  - 发送长度不是 8 比特整数倍的数据时，软件需要填充最后不满 8 比特的数据为完整的 1 字节数据。
  - 接收长度不是 8 比特整数倍的数据时，不足 1 字节的部分依然按照 1 字节进行接收。
- 用作从机时，GP-SPI2 从机可以按比特收发数据，并且比特长度在 1/2/4/8 线模式下分别是 1/2/4/8 的倍数。
  - 发送长度不是 8 比特整数倍的数据时，软件需要填充最后不满 8 比特的数据为完整的 1 字节数据。
  - 接收长度不是 8 比特整数倍的数据时，不足 1 字节的部分依然按照 1 字节进行接收，从机接收总长度可以通过 SPI_SLV_DATA_BITLEN 获取，并且最后 1 字节中的有效比特可以通过 SPI_SLV_LAST_BYTE_STRB 确定。

说明：
无论 GP-SPI2 用作主机还是从机，不足 1 字节部分的接收/发送比特顺序仍然按照配置进行。

33.5.5 传输类型

GP-SPI2 用作主机或从机时所支持的传输类型见下表。

表 33.5-5. GP-SPI2 用作主机或从机时所支持的传输类型

模式		CPU 控制的单次传输	DMA 控制的单次传输	DMA 控制的分段配置传输	DMA 控制的从机连续传输
主机	全双工	Y	Y	Y	–
	半双工	Y	Y	Y	–
从机	全双工	Y	Y	–	Y
	半双工	Y	Y	–	Y

以下章节将详细介绍上表中所列的各种传输类型。

33.5.6 CPU 控制的数据传输

GP-SPI2 提供了 16 个 32-bit 的数据 buffer，即 SPI_W0_REG~SPI_W15_REG，见图 33.5-1。CPU 控制的传输表示在该次传输中发送的数据来自数据 buffer 或接收的数据存入数据 buffer。在这种传输类型下，每次传输事务均需要先配置相关寄存器，然后由 CPU 来触发。因此，CPU 控制的传输只能是单次传输，即仅包含一次传输事务。CPU 控制的传输支持全双工通信和半双工通信。

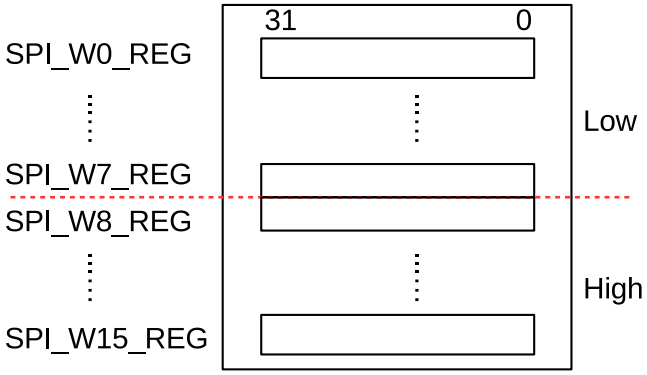


图 33.5-1. CPU 控制的传输中使用的数据 Buffer

33.5.6.1 CPU 控制的主机传输

GP-SPI2 用作主机时，无论全双工通信还是半双工通信，CPU 控制的数据传输均通过 SPI_W0_REG~SPI_W15_REG 完成。此外，用户可选择“高位模式”仅使用其中部分寄存器，具体见下方列表描述。

- **TX 数据：**
  - 未使能“高位模式”（SPI_USR_MOSI_HIGHPART 置 0）：此时，TX 数据取自 SPI_W0_REG~SPI_W15_REG，且每传输一个字节，取 TX 数据的地址即递增 1。**如果数据长度大于 64 字节，SPI_W0_REG ~ SPI_W15_REG 中的数据会被多次发送。**
    以 256 个字节为一个周期：
    * 字节 0~ 字节 63 依次发送 SPI_W0_REG~SPI_W15_REG 的数据。
    * 字节 64~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
    * 字节 256~ 字节 319（另一组数据的前 64 个字节）重新依次发送 SPI_W0_REG~SPI_W15_REG 的数据。以此类推。

    例如：发送 258 个字节（字节 0~ 字节 257），则：
    * 字节 0~ 字节 63 依次发送 SPI_W0_REG~SPI_W15_REG 的数据。
    * 字节 64~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
    * 字节 256~ 字节 257 重新依次发送地址 0 (SPI_W0_REG[7:0]) 和地址 1 (SPI_W0_REG[15:8]) 的数据：
      - 字节 256 取数据的地址为 256 对 64 取模的结果（256 % 64 = 0），即地址 0 (SPI_W0_REG[7:0])。
      - 字节 257 取数据的地址为 257 对 64 取模的结果（257 % 64 = 1），即地址 1 (SPI_W0_REG[15:8])。
  - 使能“高位模式”（SPI_USR_MOSI_HIGHPART 置 1）：此时，TX 数据取自 SPI_W8_REG~SPI_W15_REG，且每传输一个字节，取 TX 数据的地址即递增 1。**如果数据长度大于 32 字节，则 SPI_W8_REG ~ SPI_W15_REG 中的数据会被多次发送。**
    以 256 个字节为一个周期：
    * 字节 0~ 字节 31 依次发送 SPI_W8_REG~SPI_W15_REG 中的数据。
    * 字节 32~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。

* 字节 256~ 字节 287（另一组数据的前 32 个字节）重新依次发送 SPI_W8_REG~SPI_W15_REG 中的数据。以此类推。

例如：发送 258 个字节（字节 0~ 字节 257），则：

* 字节 0~ 字节 31 依次发送 SPI_W8_REG~SPI_W15_REG 的数据。
* 字节 32~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
* 字节 256~ 字节 257 重新依次发送地址 0 (SPI_W8_REG[7:0]) 和地址 1 (SPI_W8_REG[15:8]) 的数据：
  - 字节 256 取数据的地址为 256 对 32 取模的结果（256 % 32 = 0），即地址 0 (SPI_W8_REG[7:0])。
  - 字节 257 取数据的地址为 257 对 32 取模的结果（257 % 32 = 1），即地址 1 (SPI_W8_REG[15:8])。

- **RX 数据：**
  - 未使能“高位模式”（SPI_USR_MISO_HIGHPART 置 0）：此时，RX 数据存入 SPI_WO_REG~SPI_W15_REG，且每传输一个字节，存 RX 数据的地址即递增 1。**如果数据长度大于 64 字节，SPI_WO_REG ~ SPI_W15_REG 中的数据会被覆盖。**
    以 256 个字节为一个周期：
    * 字节 0~ 字节 63 依次存入 SPI_WO_REG~SPI_W15_REG。
    * 字节 64~ 字节 255 重复存入 SPI_W15_REG[31:24]。
    * 字节 256~ 字节 319（另一组数据的前 64 个字节）重新依次存入 SPI_WO_REG~SPI_W15_REG。以此类推。

    例如：接收 258 个字节（字节 0~ 字节 257），则：
    * 字节 0~ 字节 63 依次存入 SPI_WO_REG~SPI_W15_REG。
    * 字节 64~ 字节 255 重复存入 SPI_W15_REG[31:24]。
    * 字节 256~ 字节 257 依次存入地址 0 (SPI_WO_REG[7:0]) 和地址 1 (SPI_WO_REG[15:8])：
      - 字节 256 存数据的地址为 256 对 64 取模的结果 (256 % 64 = 0)，即地址 0 (SPI_WO_REG[7:0])。
      - 字节 257 存数据的地址为 257 对 64 取模的结果 (257 % 64 = 1)，即地址 1 (SPI_WO_REG[15:8])。
  - 使能“高位模式”（SPI_USR_MISO_HIGHPART 置 1）：此时，RX 数据存入 SPI_W8_REG~SPI_W15_REG，且每传输一个字节，存 RX 数据的地址即递增 1。**如果数据长度大于 32 字节，则 SPI_W8_REG ~ SPI_W15_REG 中的数据会被覆盖。**
    以 256 个字节为一个周期：
    * 字节 0~ 字节 31 依次存入 SPI_W8_REG~SPI_W15_REG。
    * 字节 32~ 字节 255 重复存入 SPI_W15_REG[31:24]。
    * 字节 256~ 字节 287（另一组数据的前 32 个字节）依次存入 SPI_W8_REG~SPI_W15_REG。以此类推。

    例如：接收 258 个字节（字节 0~ 字节 257），则：

* 字节 0~ 字节 31 依次存入 SPI_W8_REG~SPI_W15_REG。
* 字节 32~ 字节 255 重复存入 SPI_W15_REG[31:24]。
* 字节 256~ 字节 257 依次存入地址 0 (SPI_W8_REG[7:0]) 和地址 1 (SPI_W8_REG[15:8])：
  - 字节 256 存数据的地址为 256 对 32 取模的结果 (256 % 32 = 0)，即地址 0 (SPI_W8_REG[7:0])。
  - 字节 257 存数据的地址为 257 对 32 取模的结果 (257 % 32 = 1)，即地址 1 (SPI_W8_REG[15:8])。

说明：

- 上述的 TX/RX 数据均按字节寻址。
  - 如果未使能“高位模式”，地址 0 表示 SPI_WO_REG[7:0]，地址 1 表示 SPI_WO_REG[15:8]，以此类推。
  - 如果使能了“高位模式”，地址 0 表示 SPI_W8_REG[7:0]，地址 1 表示 SPI_W8_REG[15:8]，以此类推。

  最大地址为 SPI_W15_REG[31:24]。
- 为避免 TX/RX 数据传输错误，如 TX 数据重复发送或 RX 数据被覆盖等问题，请确保寄存器配置正确。

33.5.6.2 CPU 控制的从机传输

GP-SPI2 用作从机时，无论全双工通信或半双工通信，CPU 控制的数据传输均通过 SPI_WO_REG~SPI_W15_REG 完成，均采用按字节寻址。

- 全双工通信方式下：SPI_WO_REG~SPI_W15_REG 地址从 0 开始，且每传输一个字节，地址即递增 1。如果数据地址大于 63，则 SPI_WO_REG~SPI_W15_REG 中的数据会被覆盖。覆盖规律同主机模式下的非“高位模式”。
- 半双工通信方式下：通信格式中的 ADDR 值即为 RX 数据或 TX 数据的起始地址，对应 SPI_WO_REG~SPI_W15_REG。每传输一个字节，则 RX 或 TX 地址即递增 1。如果地址大于 63（即大于最高地址：SPI_W15_REG[31:24]），SPI_W8_REG~SPI_W15_REG 中的数据会被覆盖。覆盖规律同主机模式下的“高位模式”。

用户可根据具体应用，将 SPI_WO_REG~SPI_W15_REG：

- 全部用作数据 buffer
- 部分用作数据 buffer，部分用作状态 buffer
- 全部用作状态 buffer

33.5.7 DMA 控制的数据传输

在 DMA 控制的传输中，GDMA RX 模块接收数据，GDMA TX 模块发送数据。GP-SPI2 用作主机和从机时均支持这种传输类型。

DMA 控制的传输可以是：

- 一次单次传输，仅包含一次传输事务。GP-SPI2 用作主机和从机时均支持这种单次传输。
- 分段配置传输，包含多个传输事务（即多个分段）。仅有 GP-SPI2 用作主机时支持这种分段配置传输。更多信息，见章节 33.5.9.5。

- 从机连续传输，包含多次传输事务。仅在 GP-SPI2 用作从机时支持这种从机连续传输。更多信息，见章节 33.5.10.3。

DMA 控制的传输只需由 CPU 触发一次即可完成多次传输事务。此类传输一旦被触发，GDMA 引擎从 DMA 链接的内存中发送数据，或将收到的数据存入 DMA 链接的内存中，无需 CPU 的干预。

DMA 控制的传输支持全双工通信、半双工通信以及章节 33.5.9 和章节 33.5.10 所描述的功能。同时，GDMA RX 模块与 GDMA TX 模块互不影响，即支持四种全双工通信：

- 在 DMA 控制模式下接收数据，并在 DMA 控制模式下发送数据；
- 在 DMA 控制模式下接收数据，但在 CPU 控制模式下发送数据；
- 在 CPU 控制模式下接收数据，但在 DMA 控制模式下发送数据；
- 在 CPU 控制模式下接收数据，并在 CPU 控制模式下发送数据。

33.5.7.1 GDMA 配置

- 选择 GDMA 通道 *n*，并配置 GDMA TX/RX 描述符，见章节 5 *通用 DMA 控制器 (GDMA)*；
- 置位 AHB_DMA_INLINK_START_CH*n*/AHB_DMA_OUTLINK_START_CH*n* 启动 GDMA RX/TX 引擎；
- 如果置位 AHB_DMA_OUTLINK_RESTART_CH*n*，则在所有 GDMA TX buffer 用完之前，或在 GDMA TX 引擎重置之前，新的 TX buffer 将会被添加到最后使用中的 TX buffer 结尾；
- GDMA RX buffer 的链接方式与 GDMA TX buffer 的链接方式相同，可通过置位 AHB_DMA_INLINK_START_CH*n*/AHB_DMA_INLINK_RESTART_CH*n* 来实现；
- TX 数据长度和 RX 数据长度分别由 GDMA TX buffer 和 RX buffer 决定，长度范围为：0~32 KB；
- 启动 GDMA 前，先初始化 GDMA 接收链表 (inlink) 和发送链表 (outlink)。请置位寄存器 SPI_DMA_CONF_REG 中的 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA 位，否则读/写数据将存至或取自寄存器 SPI_W0_REG~SPI_W15_REG。

GP-SPI2 用作主机时，如果置位了 AHB_DMA_IN_SUC_EOF_CH*n*_INT_ENA，则一次单次传输结束或一次分段配置传输结束，就会触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。

GP-SPI2 用作从机时，如果置位了 AHB_DMA_IN_SUC_EOF_CH*n*_INT_ENA，则下表中任一情况均可触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。

表 33.5-6. GP-SPI2 用作从机时数据传输的中断触发条件

传输类型	控制位[1]	控制位[2]	触发条件
从机单次传输	0	0	一次单次传输结束即触发该中断。
	1	0	一次单次传输结束，或接收的数据长度等于 SPI_MS_DATA_BITLEN + 1，即触发该中断。
从机连续传输	0	1	正确接收 CMD7 或 End_SEG_TRANS 命令即触发该中断。
	1	1	正确接收 CMD7 或 End_SEG_TRANS 命令、或接收的数据长度等于 SPI_MS_DATA_BITLEN + 1，即触发该中断。

[1] SPI_RX_EOF_EN

[2] SPI_DMA_SLV_SEG_TRANS_EN

33.5.7.2 GDMA TX/RX Buffer 长度控制

配置的 GDMA TX/RX buffer 长度最好应等于实际传输数据的长度。如果配置的 GDMA TX/RX buffer 长度不等于实际传输数据的长度：

- 如果配置的 GDMA TX buffer 长度小于实际传输的数据长度，则多出来的数据将与最后传输的 TX buffer 数据相同。同时触发 SPI_OUTFIFO_EMPTY_ERR_INT 和 AHB_DMA_OUT_EOF_CH*n*_INT 中断。
- 如果配置的 GDMA TX buffer 长度大于实际传输的数据长度，则 TX buffer 中的数据未被完全使用，即使稍后链接了新的 TX buffer，上个 TX buffer 中剩余的数据也将参与后续传输。请特别注意这一点，或保存未使用的数据并重置 GDMA。
- 如果配置的 GDMA RX buffer 长度小于实际传输的数据长度，则多出来的数据将会丢失。同时触发 SPI_INFIFO_FULL_ERR_INT 和 SPI_TRANS_DONE_INT 中断。但不会触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。
- 如果配置的 GDMA RX buffer 长度大于实际传输的数据长度，则 RX buffer 未被使用的部分被丢弃，下次传输直接使用后面链接的 RX buffer。

33.5.8 GP-SPI2 用作主机和从机时的数据流控制

GP-SPI2 用作主机和从机时均支持 CPU 控制的数据传输和 DMA 控制的数据传输。CPU 控制的数据传输发生在 SPI_W0_REG~SPI_W15_REG 和外围 SPI 设备之间。DMA 控制的数据传输发生在配置好的 GDMA TX/RX buffer 和外围 SPI 设备之间。用户可在传输开始之前，配置 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA 来选择需要的传输类型。

33.5.8.1 GP-SPI2 功能块图

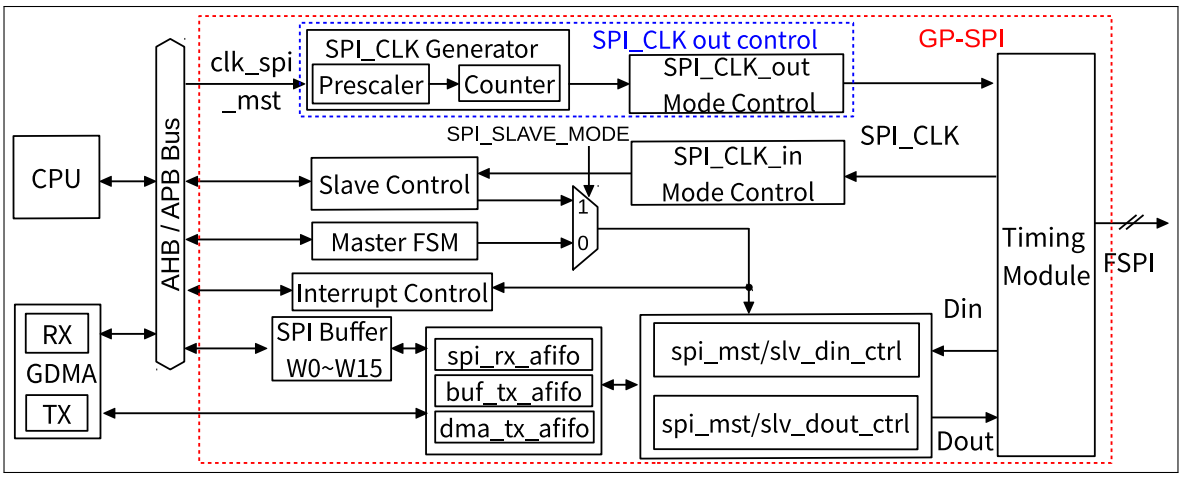


图 33.5-2. GP-SPI2 功能块图

图 33.5-2 所示为 GP-SPI2 主要的功能模块，包括：

- Master FSM：GP-SPI2 的主机状态机。GP-SPI2 用作主机时支持的所有功能，均由该状态机与寄存器共同控制。
- SPI Buffer: SPI_W0_REG~SPI_W15_REG，见图 33.5-1。CPU 控制的传输，数据在 SPI buffer 中准备。
- 时序模块 (Timing Module)：捕获 FSPI 总线上的数据。

- spi_mst/slv_din_ctrl 和 spi_mst/slv_dout_ctrl：用于将 TX/RX 数据转换成字节。
- spi_rx_afifo：暂存接收到的数据。
- buf_tx_afifo：暂存待发送的数据。
- dma_tx_afifo：暂存来自 GDMA 的数据。
- clk_spi_mst：GP-SPI2 模块时钟，由 PLL_CLK 分频所得。在 GP-SPI2 用作主机时用于生成数据传输以及从机所需的 SPI_CLK 信号。
- SPI_CLK 生成器 (SPI_CLK Generator)：对 clk_spi_mst 进行分频生成 SPI_CLK 信号。分频系数由 SPI_CLKCNT_N 和 SPI_CLKDIV_PRE 共同决定，见章节 33.7。
- SPI_CLK_out Mode Control：发送数据传输以及从机所需的 SPI_CLK 信号。
- SPI_CLK_in Mode Control：当 GP-SPI2 用作从机时，用于捕获 SPI 主机发出的 SPI_CLK 信号。

33.5.8.2 GP-SPI2 用作主机时的数据流控制

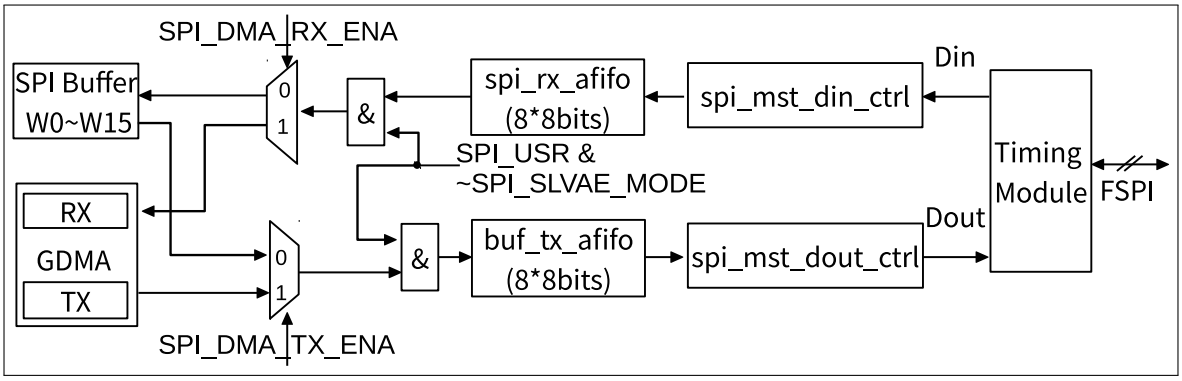


图 33.5-3. GP-SPI2 用作主机时的数据流控制

图 33.5-3 所示为 GP-SPI2 用作主机时的数据流。其控制逻辑如下：

- RX 数据：时序模块捕获 FSPI 总线上的数据，然后 *spi_mst_din_ctrl* 模块将比特数据转化为字节数据，暂存于 *spi_rx_afifo* 中，此后根据控制方式转存至不同的接收位置：
  - CPU 控制：转存至 SPI_W0_REG~SPI_W15_REG
  - DMA 控制：转存至 GDMA RX buffer
- TX 数据：*buf_tx_afifo* 模块暂存待发送数据。根据控制方式不同，待发送数据来自不同的位置：
  - CPU 控制：TX 数据来自 SPI_W0_REG~SPI_W15_REG
  - DMA 控制：TX 数据来自 GDMA TX buffer

buf_tx_afifo 中的数据会由时序模块以 1/2/4-bit 的模式发送出去。具体数据模式由 GP-SPI2 状态机控制。时序模块可用于时序补偿。更多信息，见章节 33.8。

33.5.8.3 GP-SPI2 用作从机时的数据流控制

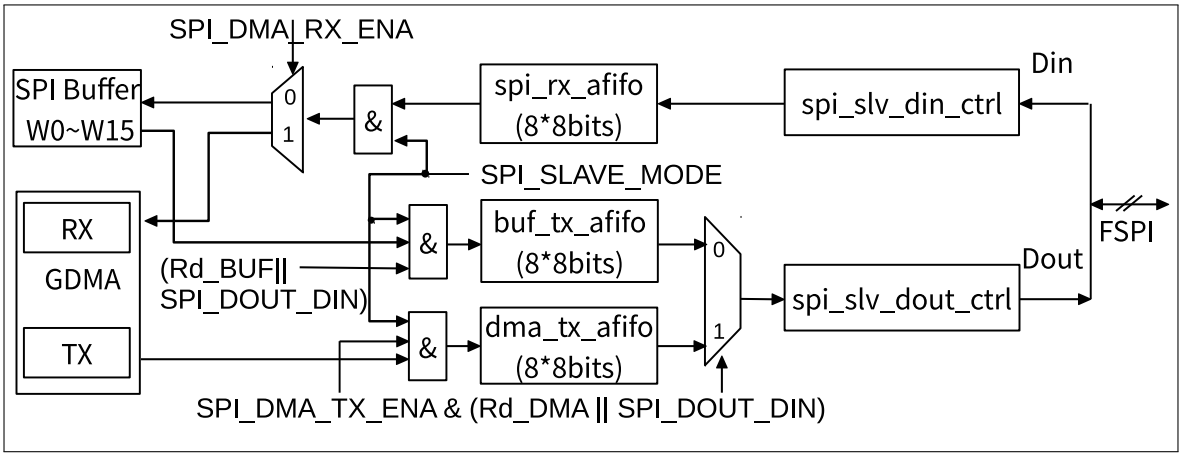


图 33.5-4. GP-SPI2 用作从机时的数据流控制

图 33.5-4 所示为 GP-SPI2 用作从机时的数据流控制。其控制逻辑如下：

- 在 CPU/DMA 控制的全双工/半双工传输下，当外部 SPI 主机发起 SPI 传输后，FSPI 总线上的数据将被捕获，然后由 *spi_slv_din_ctrl* 模块转换为字节，暂存于 *spi_rx_afifo* 中。
  - 在 CPU 控制的全双工传输中，暂存于 *spi_rx_afifo* 中的 RX 数据之后会被相继转存到 SPI_W0_REG~SPI_W15_REG。
  - 在半双工 Wr_BUF 传输中，收到地址值 (SLV_ADDR[7:0]) 后，*spi_rx_afifo* 中暂存的 RX 数据将转存至寄存器 SPI_W0_REG~SPI_W15_REG 的相应地址中。
  - 在 DMA 控制的全双工传输中，或在半双工 Wr_DMA 传输中，*spi_rx_afifo* 中暂存的 RX 数据将转存至配置好的 GDMA RX buffer 中。
- 在 CPU 控制的全双工/半双工传输中，待发送的数据暂存在 *buf_tx_afifo* 中；而在 DMA 控制的全双工/半双工传输中，待发送的数据暂存在 *dma_tx_afifo* 中。因此，在一次从机连续传输中，CPU 控制的 Rd_BUF 传输事务和 DMA 控制的 Rd_DMA 传输事务可同时发生。
  - 在 CPU 控制的全双工传输中，如果置位了 SPI_SLAVE_MODE 和 SPI_DOUTDIN，同时清零了 SPI_DMA_TX_ENA，则 SPI_W0_REG~SPI_W15_REG 中的数据将被转存至 *buf_tx_afifo* 中。
  - 在 CPU 控制的半双工传输中，如果置位了 SPI_SLAVE_MODE，清零了 SPI_DOUTDIN，且收到指令 Rd_BUF 和地址 SLV_ADDR[7:0]，则 SPI_W0_REG~SPI_W15_REG 相应地址中的数据将被转存至 *buf_tx_afifo* 中。
  - 在 DMA 控制的全双工传输中，如果置位了 SPI_SLAVE_MODE、SPI_DOUTDIN 和 SPI_DMA_TX_ENA，则 GDMA TX buffer 中的数据将被转存至 *dma_tx_afifo* 中。
  - 在 DMA 控制的半双工传输中，如果置位了 SPI_SLAVE_MODE，清零了 SPI_DOUTDIN，且收到指令 Rd_DMA，则 GDMA TX buffer 中的数据将被转存至 *dma_tx_afifo* 中。

  buf_tx_afifo 或 *dma_tx_afifo* 的数据将由 *spi_slv_dout_ctrl* 模块以 1/2/4-bit 的模式发送出去。

33.5.9 GP-SPI2 用作主机

清零 SPI_SLAVE_REG 中 SPI_SLAVE_MODE 位可将 GP-SPI2 配置成主机。此时，GP-SPI2 提供时钟信号（GP-SPI2 模块时钟的分频时钟）和六条 CS 线 (CS0~CS5)。

33.5.9.1 主机状态机

GP-SPI2 用作主机时，状态机在数据传输中控制其各个阶段，包括配置阶段 (CONF)、准备阶段 (PREP)、命令阶段 (CMD)、地址阶段 (ADDR)、空闲阶段 (DUMMY)、发送数据阶段 (DOUT) 和接收数据阶段 (DIN)。GP-SPI2 主要用于访问 1/2/4-bit SPI 设备，如 flash、外部 RAM 等。因此，GP-SPI2 各个阶段的命名规则应与 flash 以及外部 RAM 的时序名称保持一致。每个阶段的描述如下，GP-SPI2 状态机的工作流程见图 33.5-5。

1. 空闲阶段 (IDLE)：GP-SPI2 未处于工作状态或正在用作从机。
2. 配置阶段 (CONF)：仅用于 DMA 控制的分段配置传输。置位 SPI_USR 和 SPI_USR_CONF 使能该阶段。如果未使能该阶段，则说明当前传输为单次传输。
3. 准备阶段 (PREP)：准备 SPI 传输事务，控制 SPI CS 建立时间。置位 SPI_USR 和 SPI_CS_SETUP 使能该阶段。
4. 命令阶段 (CMD)：发送命令序列。置位 SPI_USR 和 SPI_USR_COMMAND 使能该阶段。
5. 地址阶段 (ADDR)：发送地址序列。置位 SPI_USR 和 SPI_USR_ADDR 使能该阶段。
6. 等待阶段 (DUMMY)：发送 DUMMY 序列。置位 SPI_USR 和 SPI_USR_DUMMY 使能该阶段。
7. 传输数据阶段 (DATA)：传输数据。
   - DOUT：发送数据。置位 SPI_USR 和 SPI_USR_MOSI 使能该阶段。
   - DIN：接收数据。置位 SPI_USR 和 SPI_USR_MISO 使能该阶段。
8. 结束阶段 (DONE)：控制 SPI CS 保持时间。置位 SPI_USR 使能该阶段。

注意：

主机状态机启动必须配置 SPI_USR。SPI_MST_FD_WAIT_DMA_TX_DATA 可以控制 SPI_USR 生效的时刻：

- 当 SPI_MST_FD_WAIT_DMA_TX_DATA 为 0 时，状态机相关状态在配置 SPI_USR 和其他控制寄存器后立即生效；
- 当 SPI_MST_FD_WAIT_DMA_TX_DATA 为 1 时，如果状态机也配置了 DOUT 状态，则会等待 *buf_tx_fifo* 中至少有数据后，SPI_USR 和其他控制寄存器才会生效，状态机开始启动。

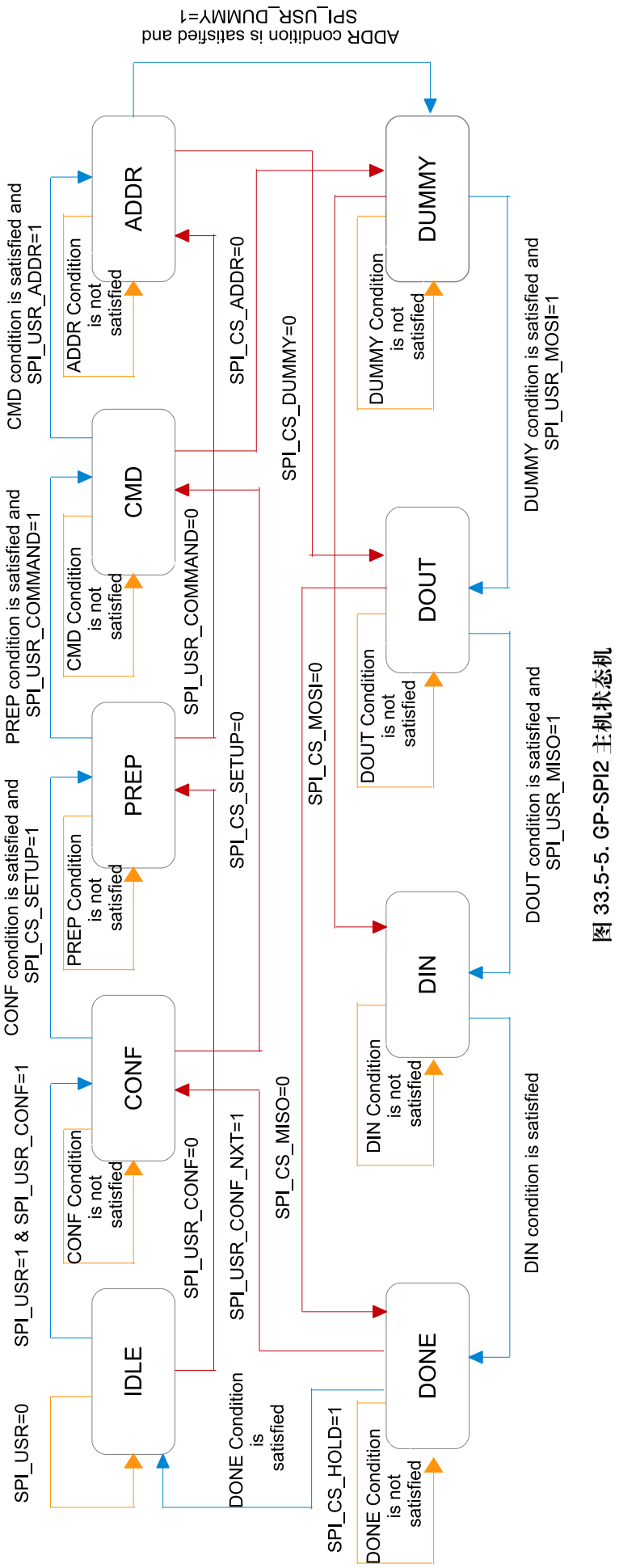


图 33.5-5. GP-SPI2 主机状态机

图标说明：

- ——：表示相应的状态条件不满足，重复当前状态。
- ——：表示相应的寄存器已配置，状态条件已满足，将进行下一个状态。
- ——：表示相应的寄存器未配置，跳过下一个状态，或跳过后续多个状态。

上图中的各个状态条件描述如下：

- CONF condition: gpc[17:0] >= SPI_CONF_BITLEN[17:0]
- PREP condition: gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]
- CMD condition: gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]
- ADDR condition: gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]
- DUMMY condition: gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]
- DOUT condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DIN condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DONE condition: (gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)

状态机中用到了一个计数器 (gpc[17:0]) 来控制每个状态的周期长度。CONF、PREP、CMD、ADDR、DUMMY、DOUT 和 DIN 各状态可单独使能或禁用，也可以单独配置其周期长度。

33.5.9.2 状态控制和位模式控制寄存器

概述

表 33.5-7 列出了与 GP-SPI2 状态控制相关的寄存器配置。如需使能 GP-SPI2 的 QPI 模式，请置位寄存器 SPI_USER_REG 中 SPI_QPI_MODE 位。

表 33.5-7. 1/2/4-bit 模式下状态控制寄存器

状态	1-bit FSPI 控制寄存器	2-bit FSPI 控制寄存器	4-bit FSPI 控制寄存器
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_MS_DATA_BITLEN	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_DUAL	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_QUAD
DOUT	SPI_USR_MOSI SPI_MS_DATA_BITLEN	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_DUAL	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_QUAD

如表 33.5-7 所示，如果希望在表格第一栏所示的状态中将 FSPI 总线设置为相应的位模式（见表头），则需要配置该行中每一单元格的寄存器。

配置

例如，当 GP-SPI2 读取数据时，且希望实现：

- CMD 为 1-bit 模式
- ADDR 为 2-bit 模式
- DUMMY 为 8 个时钟周期
- DIN 为 4-bit 模式

则具体的寄存器配置如下：

1. 配置 CMD 状态相关寄存器。
   - 配置 SPI_USR_COMMAND_VALUE 为需要的命令值；
   - 配置 SPI_USR_COMMAND_BITLEN。SPI_USR_COMMAND_BITLEN 为所需要的命令位长 - 1；
   - 置位 SPI_USR_COMMAND；
   - 清除 SPI_FCMD_DUAL 和 SPI_FCMD_QUAD。
2. 配置 ADDR 状态相关寄存器。
   - 配置 SPI_USR_ADDR_VALUE 为需要的地址值；
   - 配置 SPI_USR_ADDR_BITLEN。SPI_USR_COMMAND_BITLEN 为所需要的地址位长 - 1；
   - 置位 SPI_USR_ADDR 和 SPI_FADDR_DUAL；
   - 清除 SPI_FADDR_QUAD。
3. 配置 DUMMY 状态相关寄存器。
   - 在 SPI_USR_DUMMY_CYCLELEN 中配置 DUMMY 周期，其中 SPI_USR_DUMMY_CYCLELEN 的值等于 DUMMY 阶段所需要的时钟周期数 - 1；
   - 置位 SPI_USR_DUMMY。
4. 配置 DIN 状态相关寄存器。
   - 在 SPI_MS_DATA_BITLEN 中配置读数据的位长；SPI_MS_DATA_BITLEN 的值等于所需要的位长 - 1；
   - 置位 SPI_FREAD_QUAD 和 SPI_USR_MISO；
   - 清除 SPI_FREAD_DUAL；
   - 如果选择了 DMA 控制的传输类型，则需要配置 GDMA。如果选择了 CPU 控制的传输类型，则无需任何操作；
5. 清除 SPI_USR_MOSI；
6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer。
7. 置位 SPI_USR 开始 GP-SPI2 传输。

说明：
本章节涉及主机模式下的寄存器配置，都需要置位 SPI_UPDATE 以将配置数据从 AHB_CLK 更新到 clk_spi_mst 时钟域，从机模式无需关注。具体的配置方式如上例。

写数据时 (DOUT)，需要配置 SPI_USR_MOSI，同时清除 SPI_USR_MISO。输出数据的位长等于 SPI_MS_DATA_BITLEN 加 1。在 CPU 控制的传输类型下，需要在数据 buffer (SPI_W0_REG~SPI_W15_REG) 中准备数据；在 DMA 控制的数据传输下，需要在 GDMA TX buffer 中准备输出数据。

需特别注意 SPI_USR_COMMAND_VALUE 中的命令值以及 SPI_USR_ADDR_VALUE 中的地址值。

命令值的配置如下：

表 33.5-8. 命令值的发送顺序

COMMAND_BITLEN[1]	COMMAND_VALUE[2]	BIT_ORDER[3]	命令值的发送顺序
0~7	[7:0]	1	先发送 COMMAND_VALUE[COMMAND_BITLEN:0]。
		0	先发送 COMMAND_VALUE[7:7−COMMAND_BITLEN]。
8~15	[15:0]	1	先发送 COMMAND_VALUE[7:0]，再发送 COMMAND_VALUE[COMMAND_BITLEN:8]。
		0	先发送 COMMAND_VALUE[7:0]，再发送 COMMAND_VALUE[15:15−COMMAND_BITLEN]。

[1] SPI_USR_COMMAND_BITLEN：用于配置命令的位长。
[2] SPI_USR_COMMAND_VALUE：命令值写入的字段，见上表。
[3] SPI_WR_BIT_ORDER：请参考表 33.5-4 中的配置方法。

地址值配置如下：

表 33.5-9. 地址值的发送顺序

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	地址发送顺序
0~7	[31:24]	1	先发送 ADDR_VALUE[ADDR_BITLEN + 24:24]。
		0	先发送 ADDR_VALUE[31:31−ADDR_BITLEN]。
8~15	[31:16]	1	先发送 ADDR_VALUE[31:24]，再发送 ADDR_VALUE[ADDR_BITLEN + 8:16]。
		0	先发送 ADDR_VALUE[31:24]，再发送 ADDR_VALUE[23:31−ADDR_BITLEN]。
16~23	[31:8]	1	先发送 ADDR_VALUE[31:16]，再发送 ADDR_VALUE[ADDR_BITLEN−8:8]。
		0	先发送 ADDR_VALUE[31:16]，再发送 ADDR_VALUE[15:31−ADDR_BITLEN]。
24~31	[31:0]	1	先发送 ADDR_VALUE[31:8]，再发送 ADDR_VALUE[ADDR_BITLEN−24:0]。
		0	先发送 ADDR_VALUE[31:8]，再发送 ADDR_VALUE[7:31−ADDR_BITLEN]。

[1] SPI_USR_ADDR_BITLEN：用于配置地址值的位长。
[2] SPI_USR_ADDR_VALUE：地址值写入的字段，见上表。
[3] SPI_WR_BIT_ORDER：请参考表 33.5-4 中的配置方法。

33.5.9.3 主机全双工通信（仅支持 1-bit 模式）

概述

GP-SPI2 支持 SPI 全双工通信。在这种通信方式下，SPI 主机提供 CLK 和 CS 信号，然后与从机使用 1-bit 模式同时交换数据：MOSI（FSPID，发送），MISO（FSPIQ，接收）。用户可通过置位寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位使能全双工通信。GP-SPI2 与从机使用全双工通信时的连接方式见图 33.5-6。

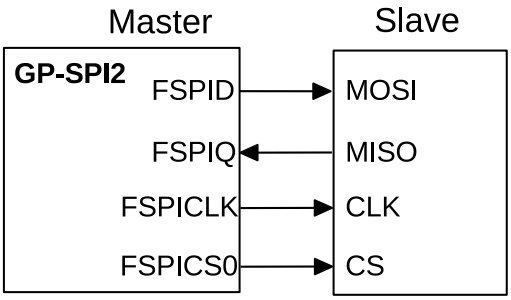


图 33.5-6. GP-SPI2 主机使用全双工模式与 SPI 从机通信框图

在全双工通信中，CMD、ADDR、DUMMY、DOUT 和 DIN 各个状态的具体行为可配置。通常，全双工通信跳过 CMD、ADDR 和 DUMMY 状态。传输数据的位长可在 SPI_MS_DATA_BITLEN 中配置。通信中使用的实际位长等于 (SPI_MS_DATA_BITLEN + 1)。

配置

按照以下操作步骤，开始数据传输：

- 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
- 配置 AHB 时钟、APB 时钟（即 AHB_CLK、APB_CLK，见章节 9 复位和时钟），并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；
- 置位 SPI_DOUTDIN 同时清除 SPI_SLAVE_MODE，使能主机全双工通信方式；
- 配置表 33.5-7 中所列的 GP-SPI2 寄存器；
- 配置 SPI CS 建立时间和保持时间，见章节 33.6；
- 设置 FSPICLK 的极性，见章节 33.7；
- 根据选定的传输类型准备数据：
  - 如果选择的传输类型为 CPU 控制的 MOSI 传输，则需要在 SPI_W0_REG~SPI_W15_REG 中准备数据。
  - 如果选择了 DMA 控制的传输类型，则需要：
    * 配置 SPI_DMA_RX_ENA/SPI_DMA_TX_ENA；
    * 配置 GDMA TX/RX 链表；
    * 启动 GDMA TX/RX 引擎，更多描述见章节 33.5.7 和章节 33.5.8。
- 配置中断，然后等待 SPI 从机做好传输准备；
- 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
- 置位寄存器 SPI_CMD_REG 中 SPI_USR 位，开始数据传输，然后等待之前配置的中断。

33.5.9.4 主机半双工通信（支持 1/2/4-bit 模式）

概述

在半双工通信下，GP-SPI2 发送 CLK 和 CS 信号。在同一时刻，SPI 主机或从机只能有一个可以发送数据，另一个接收数据。用户可通过清除寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位使能半双工通信。SPI 半双工通信的通用格式为 CMD + [ADDR +] [DUMMY +] [DOUT 或 DIN]。其中，ADDR、DUMMY、DOUT 和 DIN 状态非必选，可单独禁用或启用。

如章节 33.5.9.2 所述，CMD、ADDR、DUMMY、DOUT 和 DIN 各个状态的周期、具体值和并行总线位模式等可独立配置。更多寄存器配置信息，见表 33.5-7。

半双工 GP-SPI2 的详细属性如下：

1. CMD：0~16 位，主机发送，从机接收 (MOSI)。
2. ADDR：0~32 位，主机发送，从机接收 (MOSI)。
3. DUMMY：0~256 个 FSPICLK 周期，主机发送，从机接收。
4. DOUT：在 CPU 控制的传输中，可传输 0~512 位（64 字节）数据；在 DMA 控制的单次传输中，可传输 0~256 Kbit (32 KB)；在 DMA 控制的分段配置传输中，可传输的数据长度无限制。主机发送，从机接收。
5. DIN：在 CPU 控制的传输中，可传输 0~512 位（64 字节）数据；在 DMA 控制的单次传输中，可传输 0~256 Kbit (32 KB)；在 DMA 控制的分段配置传输中，可传输的数据长度无限制。主机接收，从机发送。

配置

具体的寄存器配置如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟、APB 时钟（即 AHB_CLK、APB_CLK），并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；
3. 清除 SPI_DOUTDIN 和 SPI_SLAVE_MODE 位，使能主机半双工通信方式；
4. 配置表 33.5-7 中所列的 GP-SPI2 寄存器；
5. 配置 SPI CS 建立时间和保持时间，见章节 33.6；
6. 设置 FSPICLK 的极性，见章节 33.7；
7. 根据选定的传输类型准备数据：
   - 如果选择的传输类型为 CPU 控制的 MOSI 传输，则需要在 SPI_W0_REG~SPI_W15_REG 中准备数据。
   - 如果选择了 DMA 控制的传输类型，则需要：
     - 配置 SPI_DMA_RX_ENA/SPI_DMA_TX_ENA；
     - 配置 GDMA TX/RX 链表；
     - 启动 GDMA TX/RX 引擎，更多描述见章节 33.5.7 和章节 33.5.8。
8. 配置中断，然后等待 SPI 从机做好传输准备；
9. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
10. 置位寄存器 SPI_CMD_REG 中 SPI_USR 位，开始数据传输，然后等待之前配置的中断。

应用示例

以下示例展示了 GP-SPI2 如何在主机半双工模式下访问 flash 和外部 RAM。

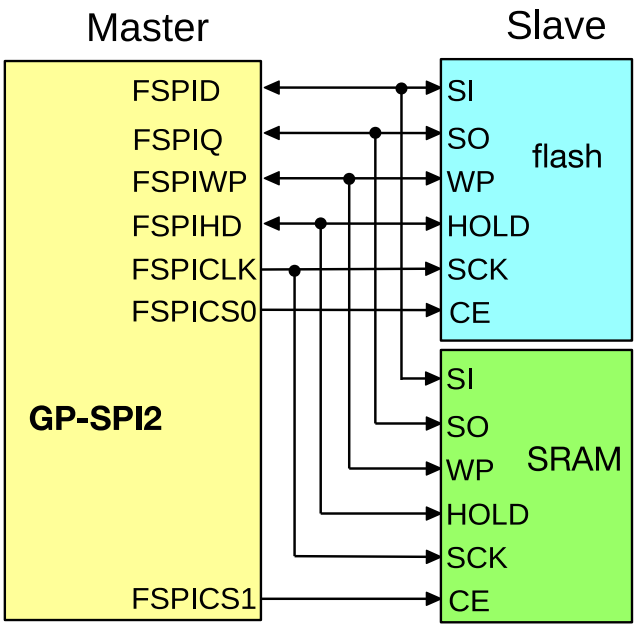


图 33.5-7. 4-bit 模式下 GP-SPI2 与 Flash 以及外部 RAM 的连接方式

图 33.5-8 所示为 GP-SPI2 按照标准 flash 规范进行 Quad I/O Read 操作。其他 GP-SPI2 命令序列可以根据 SPI 从机的要求实现。

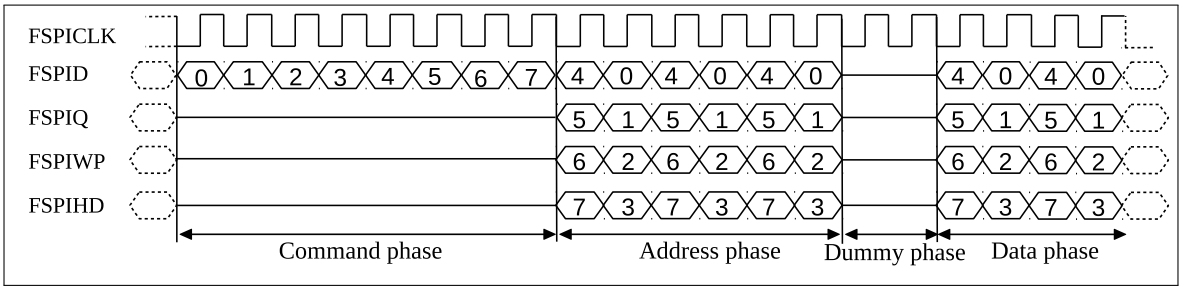


图 33.5-8. GP-SPI2 发送到 Flash 的 SPI Quad I/O 命令序列

33.5.9.5 DMA 控制的分段配置传输

说明：

注意，由于跳过 CONF 阶段即可实现单次传输，因此不再另起章节单独介绍如何在 GP-SPI2 用作主机时配置单次传输。

概述

GP-SPI2 用作主机时，可采用 DMA 控制的分段配置传输。

DMA 控制的主机传输可以是：

- 一次单次传输，仅包含一次传输事务。
- 分段配置传输，包括多个传输事务（即多个分段）。

如果选择了分段配置传输，则在每个分段中，寄存器均可单独配置。在分段配置传输下，仅需 CPU 触发一次，即可完成多次传输事务。具体工作流程见图 33.5-9。

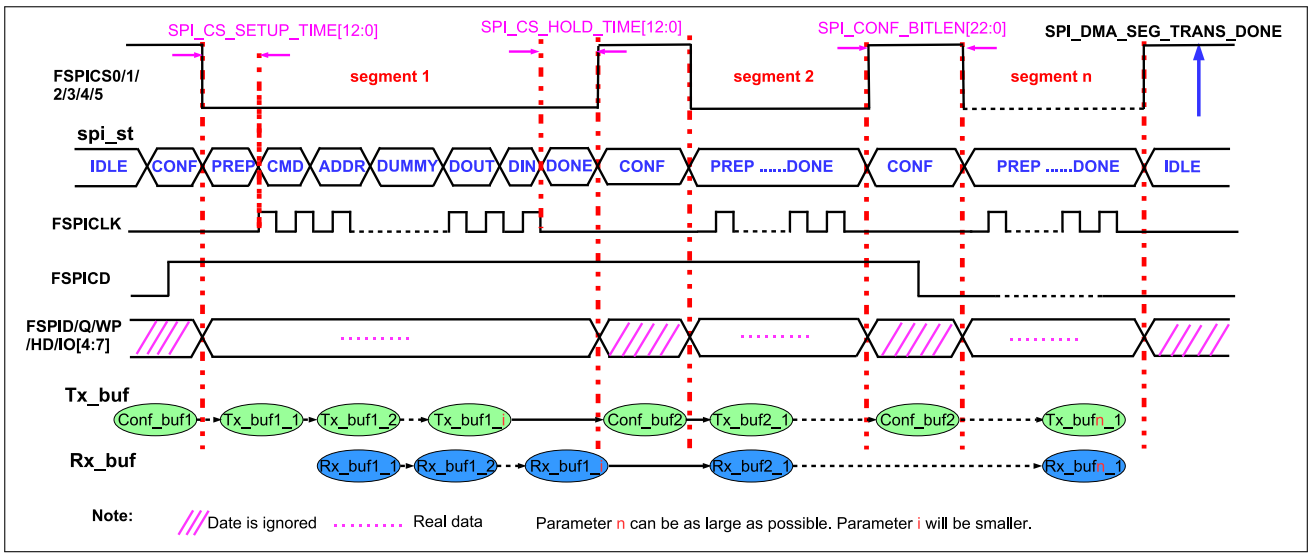


图 33.5-9. DMA 控制的分段配置传输

如图 33.5-9 所示，在分段配置传输中的某个单次传输事务 (segment *n*) 开始前，GP-SPI2 可在 CONF 阶段将寄存器重新按照 Conf_buf*n* 定义的内容进行配置。

建议为每个传输事务的 CONF 阶段提供单独的 GDMA CONF 链表和 CONF buffer（即图 33.5-9 中的 Conf_buf*i*）。GDMA TX 链表将所有的 CONF buffer 和 TX data buffer（即图 33.5-9 中的 Tx_buf*i*）链接起来，因此可以独立控制每个传输事务中的 FSPI 总线行为。

例如，在一次完整的分段配置传输中，传输事务 *i*、传输事务 *j* 和传输事务 *k* 可分别配置为全双工、半双工 MISO 和半双工 MOSI 模式。*i*、*j* 和 *k* 均为整数变量，代表传输事务的编号。

同时，每个传输事务中，GP-SPI2 所使用到的各个阶段、各个阶段的相关值和 FSPI 总线周期长、以及 GDMA 行为等，均可独立配置。当整个 DMA 控制的分段配置传输（包括多个传输事务）完成后，即触发 GP-SPI2 中断 SPI_DMA_SEG_TRANS_DONE_INT。

配置

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟、APB 时钟（AHB_CLK、APB_CLK），并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；
3. 清除 SPI_DOUTDIN 和 SPI_SLAVE_MODE 位，使能主机半双工通信方式；
4. 配置表 33.5-7 中所列的 GP-SPI2 寄存器；
5. 配置 SPI CS 建立时间和保持时间，见章节 33.6；
6. 设置 FSPICLK 的相位和极性，见章节 33.7；
7. 为每个传输事务准备 GDMA CONF buffer 描述符和 TX data 描述符（可选）。把 CONF buffer 描述符和几次传输事务需要的 TX buffer 链接成一个链表；
8. 同样，为每个传输事务准备 RX buffer 描述符，并链接成一个链表；
9. 在该 DMA 控制的分段配置传输开始之前，为每个传输事务配置所需的 CONF buffer、TX buffer 和 RX buffer；
10. 配置 AHB_DMA_OUTLINK_ADDR_CH*n* 指向 CONF 和 TX buffer 描述符链表的首地址，之后置位 AHB_DMA_OUTLINK_START_CH*n*，启动 TX GDMA；

11. 清除 SPI_DMA_CONF_REG 中 SPI_RX_EOF_EN 位。配置 AHB_DMA_INLINK_ADDR_CH*n* 指向 RX buffer 描述符链表的首地址，之后置位 AHB_DMA_INLINK_START_CH*n* 启动 RX GDMA；
12. 置位 SPI_USR_CONF 使能 CONF 阶段；
13. 置位 SPI_DMA_SEG_TRANS_DONE_INT_ENA 使能 SPI_DMA_SEG_TRANS_DONE_INT 中断。如需配置其他中断，请参考章节 33.9；
14. 等待所有从机做好传输准备；
15. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
16. 置位 SPI_USR 开始本次 DMA 控制的分段配置传输；
17. 等待 SPI_DMA_SEG_TRANS_DONE_INT 中断，即 DMA 分段配置传输结束，数据已存储至相应内存。

说明：

在 PREP 阶段需要准备好本次分段传输发送的数据。应当保证：

$$(SPI\_CS\_SETUP\_TIME + 1) \times T_{SPI\_CLK} >= 4 \times (T_{AHB\_CLK} + Tclk\_spi\_mst)$$

配置 CONF Buffer 和 Magic 值

在 GP-SPI2 分段配置传输中，仅有较上次传输事务有变动的寄存器会在 CONF 阶段被重新配置。为节省时间和芯片资源，其他寄存器配置则保持不变。

GDMA CONF buffer*i* 中第一个字，即 SPI_BIT_MAP_WORD，记录传输事务 *i* 中，寄存器是否有改动。SPI_BIT_MAP_WORD 和待更新的 GP-SPI2 寄存器的对应关系见表 33.5-10，即位图 (BM) 表。如果位图表中某一位为 1，则在此次传输事务中，该位对应寄存器的值将被更新。如果其他寄存器不需要修改，则位图表中相应位应置为 0。

表 33.5-10. CONF 阶段 BM 位图

BM 位	寄存器	BM 位	寄存器
0	SPI_ADDR_REG	7	SPI_MISC_REG
1	SPI_CTRL_REG	8	SPI_DIN_MODE_REG
2	SPI_CLOCK_REG	9	SPI_DIN_NUM_REG
3	SPI_USER_REG	10	SPI_DOUT_MODE_REG
4	SPI_USER1_REG	11	SPI_DMA_CONF_REG
5	SPI_USER2_REG	12	SPI_DMA_INT_ENA_REG
6	SPI_MS_DLEN_REG	13	SPI_DMA_INT_CLR_REG

所有待修改的寄存器新值应紧跟在 SPI_BIT_MAP_WORD 之后，在 CONF buffer 中用连续的字表示。

为确保每个 CONF buffer 中内容正确，SPI_BIT_MAP_WORD[31:28] 位将用作 Magic 值，与寄存器 SPI_SLAVE_REG 中 SPI_DMA_SEG_MAGIC_VALUE 的值进行比较。SPI_DMA_SEG_MAGIC_VALUE 的值应在此 DMA 控制的分段配置传输开始之前配置，且在任何传输事务过程中均不可更改。

- 经比较，如果 SPI_BIT_MAP_WORD[31:28] == SPI_DMA_SEG_MAGIC_VALUE，则分段配置传输继续正常进行，整个传输过程结束则触发 SPI_DMA_SEG_TRANS_DONE_INT 中断。
- 如果 SPI_BIT_MAP_WORD[31:28] != SPI_DMA_SEG_MAGIC_VALUE，则 GP-SPI2 状态，即 spi_st 将返回至 IDLE 状态，分段配置传输立即结束。同时触发 SPI_DMA_SEG_TRANS_DONE_INT 中断，SPI_SEG_MAGIC_ERR_INT_RAW 位也将置 1。

CONF Buffer 配置示例

在一次分段配置传输中，传输事务 *i* 有 SPI_ADDR_REG、SPI_CTRL_REG、SPI_CLOCK_REG、SPI_USER_REG 和 SPI_USER1_REG 五个寄存器需要更新，则其 CONF buffer*i* 具体的配置示例见表 33.5-11 和表 33.5-12。

表 33.5-11. 传输事务 *i* 中 CONF buffer*i* 配置示例

CONF buffer*i*	说明
SPI_BIT_MAP_WORD	Buffer 中的第一个字。如果 SPI_DMA_SEG_MAGIC_VALUE 设置为 0xA，则本示例中该字的值为 0xA000001F。由表 33.5-12 可知，被置 1 的位有第 0、1、2、3 和 4 位，表示下列寄存器将被更新
SPI_ADDR_REG	CONF buffer*i* 的第二个字，存储 SPI_ADDR_REG 寄存器的更新值
SPI_CTRL_REG	CONF buffer*i* 的第三个字，存储 SPI_CTRL_REG 寄存器的更新值
SPI_CLOCK_REG	CONF buffer*i* 的第四个字，存储 SPI_CLOCK_REG 寄存器的更新值
SPI_USER_REG	CONF buffer*i* 的第五个字，存储 SPI_USER_REG 寄存器的更新值
SPI_USER1_REG	CONF buffer*i* 的第六个字，存储 SPI_USER1_REG 寄存器的更新值

表 33.5-12. BM 位图与待更新的寄存器

位	值	寄存器	位	值	寄存器
0	1	SPI_ADDR_REG	7	0	SPI_MISC_REG
1	1	SPI_CTRL_REG	8	0	SPI_DIN_MODE_REG
2	1	SPI_CLOCK_REG	9	0	SPI_DIN_NUM_REG
3	1	SPI_USER_REG	10	0	SPI_DOUT_MODE_REG
4	1	SPI_USER1_REG	11	0	SPI_DMA_CONF_REG
5	0	SPI_USER2_REG	12	0	SPI_DMA_INT_ENA_REG
6	0	SPI_MS_DLEN_REG	13	0	SPI_DMA_INT_CLR_REG

说明

使用 DMA 分段配置传输功能时，应注意以下寄存器相关位：

- SPI_USR_CONF：在置位 SPI_USR 之前，需先置位 SPI_USR_CONF，以使能本次传输。
- SPI_USR_CONF_NXT：如果传输事务 *i* 不是本次 DMA 控制的分段配置传输中的最后一次传输事务，则需要置位 SPI_USR_CONF_NXT。
- SPI_CONF_BITLEN：此外，在每个单独的传输事务中，GP-SPI2 的 CS 建立时间和保持时间可独立编程，更多配置信息见章节 33.6。在每次传输事务中，CS 保持高电平的时长约为：

$$(SPI\_CONF\_BITLEN + 5) \times T_{AHB\_CLK}$$

  f_{AHB_CLK} 为 80 MHz 时，CONF 阶段的 CS 高电平时长可配置为 62.5 ns~3.2768 ms。如果 SPI_CONF_BITLEN 大于 0x3FFFA，(SPI_CONF_BITLEN + 5) 将溢出 (0x40000 - SPI_CONF_BITLEN - 5)。

33.5.10 GP-SPI2 用作从机

GP-SPI2 可用作从机与另一 SPI 主机进行通信。用作从机时，GP-SPI2 支持特定格式的 1-bit SPI、2-bit Dual SPI、4-bit Quad SPI 和 QPI 模式。用户可置位寄存器 SPI_SLAVE_REG 中 SPI_SLAVE_MODE 位使能 GP-SPI2 从机。

在传输过程中，CS 信号应保持低电平，CS 信号的下降沿和上升沿代表一次单次传输或从机连续传输的开始和结束。以 CS 低电平有效为例，GP-SPI2 用作从机时，每次 SPI 传输之间的 CS 高电平无效持续时间应当不小于 8 T_{AHB_CLK}，保证每次传输都能正常结束。

33.5.10.1 可配置的通信格式

GP-SPI2 用作从机时支持全双工通信和半双工通信。用户可配置寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位选择需要的通信方式。

全双工通信下，传输一开始，则数据同时输入和输出。此时，所有数据位均被视为输入/输出数据，即不需要命令、地址或 DUMMY 阶段。传输结束即触发 SPI_TRANS_DONE_INT 中断。

在半双工通信下，通信格式为 CMD+ADDR+DUMMY+DATA（DIN 或 DOUT）。

- “DIN” 表示 SPI 主机从 GP-SPI2 中读取数据；
- “DOUT” 表示 SPI 主机向 GP-SPI2 中写入数据。

每个阶段的详细特性如下：

1. CMD：
   - 表明 SPI 从机用于何种功能；
   - 一个字节，主机输出，从机输入；
   - 仅支持表 33.5-13 和表 33.5-14 所列的命令值；
   - 以 1-bit SPI 模式或 4-bit QPI 模式发送。
2. ADDR：
   - 在 CPU 控制的传输中，可以为 Wr_BUF 和 Rd_BUF 命令提供地址，或在其他命令中用作占位符，具体由应用定义；
   - 一个字节，主机输出，从机输入；
   - 可根据命令，以 1-bit、2-bit 或 4-bit 模式发送；
3. DUMMY：
   - DUMMY 的值无实际意义；SPI 从机在这个阶段准备数据；
   - FSPI 总线的位模式在这里也没有实际意义；
   - 持续八个 SPI_CLK 时钟周期。
4. DIN 或 DOUT：
   - 在 CPU 控制的传输下，可传输 0~64 字节数据；在 DMA 控制的传输下，传输数据长度无限制。
   - 可根据具体的 CMD 值，以 1-bit、2-bit 或 4-bit 模式发送。

说明：
半双工通信下，ADDR 和 DUMMY 阶段不可跳过。

半双工传输结束后，传输的 CMD 和 ADDR 的值分别锁存至 SPI_SLV_LAST_COMMAND 和 SPI_SLV_LAST_ADDR。如果 GP-SPI2 不支持传输的 CMD 值，SPI_SLV_CMD_ERR_INT_RAW 将被置位。SPI_SLV_CMD_ERR_INT_RAW 仅可由软件清零。

33.5.10.2 半双工通信支持的 CMD 值

在半双工传输中，CMD 定义的值将决定传输类型。不支持的 CMD 值及其相关数据传输均被忽略，且 SPI_SLV_CMD_ERR_INT_RAW 将被置 1。传输格式为：CMD（8 位）+ ADDR（8 位）+ DUMMY（8 个 SPI_CLK 周期）+ DATA，其中，DATA 的单位为字节。CMD[3:0] 的详细说明如下：

- 0x1 (Wr_BUF)：CPU 控制的写操作。主机发送数据，GP-SPI2 接收数据。数据将存储至相应地址的寄存器 SPI_W0_REG~SPI_W15_REG。
- 0x2 (Rd_BUF)：CPU 控制的读操作。主机接收 GP-SPI2 发送的数据。数据来自相应地址的寄存器 SPI_W0_REG~SPI_W15_REG。
- 0x3 (Wr_DMA)：DMA 控制的写操作。主机发送数据，GP-SPI2 接收数据。数据将存储至 GP-SPI2 的 GDMA RX buffer 中。
- 0x4 (Rd_DMA)：DMA 控制的读操作。主机接收 GP-SPI2 发送的数据。数据来自 GP-SPI2 的 GDMA TX buffer。
- 0x7 (CMD7)：用于生成 SPI_SLV_CMD7_INT 中断。在从机连续传输下，使用 DMA RX 链表时，也可用于生成 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。但不会结束 GP-SPI2 的从机连续传输。
- 0x8 (CMD8)：仅用于生成 SPI_SLV_CMD8_INT 中断，但不会结束 GP-SPI2 的从机连续传输。
- 0x9 (CMD9)：仅用于生成 SPI_SLV_CMD9_INT 中断，但不会结束 GP-SPI2 的从机连续传输。
- 0xA (CMDA)：仅用于生成 SPI_SLV_CMDA_INT 中断，但不会结束 GP-SPI2 的从机连续传输。

CMD7、CMD8、CMD9 和 CMDA 的具体用途可由用户自定义。这些命令可用作握手信号、某些特定功能的密码、或某些用户自定义操作的触发信号等。

CMD、ADDR 和 DATA 阶段均支持 1/2/4-bit 模式，具体由 CMD[7:4] 决定。DUMMY 仅支持 1-bit 模式，且持续八个 SPI_CLK 时钟周期。CMD[7:4] 的具体定义如下：

- 0x0：CMD、ADDR 和 DATA 阶段均为 1-bit 模式。
- 0x1：CMD 和 ADDR 均为 1-bit 模式。DATA 为 2-bit 模式。
- 0x2：CMD 和 ADDR 均为 1-bit 模式。DATA 为 4-bit 模式。
- 0x5：CMD 为 1-bit 模式。ADDR 和 DATA 均为 2-bit 模式。
- 0xA：CMD 为 1-bit 模式。ADDR 和 DATA 均为 4-bit 模式，或 QPI 模式。

此外，CMD[7:0] 的值为 0x05、0xA5、0x06 和 0xDD 时，将跳过 DUMMY 和 DATA 阶段。CMD[7:0] 的具体定义如下：

- 0x05 (End_SEG_TRANS)：主机发送 0x05 命令，结束 SPI 模式下从机连续传输。
- 0xA5 (End_SEG_TRANS)：主机发送 0xA5 命令，结束 QPI 模式下从机连续传输。
- 0x06 (En_QPI)：GP-SPI2 接收到 0x06 命令后，进入 QPI 模式。此时，寄存器 SPI_USER_REG 中 SPI_QPI_MODE 置位。
- 0xDD (Ex_QPI)：GP-SPI2 接收到 0xDD 命令后，退出 QPI 模式。此时，SPI_QPI_MODE 位清零。

GP-SPI2 支持的所有 CMD 值见表 33.5-13 和表 33.5-14。注意，DUMMY 仅支持 1-bit 模式，且持续八个 SPI_CLK 时钟周期。

表 33.5-13. GP-SPI2 从机 SPI 模式支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
Wr_BUF	0x01	1-bit 模式	1-bit 模式	1-bit 模式
	0x11	1-bit 模式	1-bit 模式	2-bit 模式
	0x21	1-bit 模式	1-bit 模式	4-bit 模式
	0x51	1-bit 模式	2-bit 模式	2-bit 模式
	0xA1	1-bit 模式	4-bit 模式	4-bit 模式
Rd_BUF	0x02	1-bit 模式	1-bit 模式	1-bit 模式
	0x12	1-bit 模式	1-bit 模式	2-bit 模式
	0x22	1-bit 模式	1-bit 模式	4-bit 模式
	0x52	1-bit 模式	2-bit 模式	2-bit 模式
	0xA2	1-bit 模式	4-bit 模式	4-bit 模式
Wr_DMA	0x03	1-bit 模式	1-bit 模式	1-bit 模式
	0x13	1-bit 模式	1-bit 模式	2-bit 模式
	0x23	1-bit 模式	1-bit 模式	4-bit 模式
	0x53	1-bit 模式	2-bit 模式	2-bit 模式
	0xA3	1-bit 模式	4-bit 模式	4-bit 模式
Rd_DMA	0x04	1-bit 模式	1-bit 模式	1-bit 模式
	0x14	1-bit 模式	1-bit 模式	2-bit 模式
	0x24	1-bit 模式	1-bit 模式	4-bit 模式
	0x54	1-bit 模式	2-bit 模式	2-bit 模式
	0xA4	1-bit 模式	4-bit 模式	4-bit 模式
CMD7	0x07	1-bit 模式	1-bit 模式	-
	0x17	1-bit 模式	1-bit 模式	-
	0x27	1-bit 模式	1-bit 模式	-
	0x57	1-bit 模式	2-bit 模式	-
	0xA7	1-bit 模式	4-bit 模式	-
CMD8	0x08	1-bit 模式	1-bit 模式	-
	0x18	1-bit 模式	1-bit 模式	-
	0x28	1-bit 模式	1-bit 模式	-
	0x58	1-bit 模式	2-bit 模式	-
	0xA8	1-bit 模式	4-bit 模式	-
CMD9	0x09	1-bit 模式	1-bit 模式	-
	0x19	1-bit 模式	1-bit 模式	-
	0x29	1-bit 模式	1-bit 模式	-
	0x59	1-bit 模式	2-bit 模式	-
	0xA9	1-bit 模式	4-bit 模式	-
CMDA	0x0A	1-bit 模式	1-bit 模式	-
	0x1A	1-bit 模式	1-bit 模式	-
	0x2A	1-bit 模式	1-bit 模式	-
	0x5A	1-bit 模式	2-bit 模式	-
	0xAA	1-bit 模式	4-bit 模式	-
End_SEG_TRANS	0x05	1-bit 模式	-	-
En_QPI	0x06	1-bit 模式	-	-

表 33.5-14. QPI 模式支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
Wr_BUF	0xA1	4-bit 模式	4-bit 模式	4-bit 模式
Rd_BUF	0xA2	4-bit 模式	4-bit 模式	4-bit 模式
Wr_DMA	0xA3	4-bit 模式	4-bit 模式	4-bit 模式
Rd_DMA	0xA4	4-bit 模式	4-bit 模式	4-bit 模式
CMD7	0xA7	4-bit 模式	4-bit 模式	-
CMD8	0xA8	4-bit 模式	4-bit 模式	-
CMD9	0xA9	4-bit 模式	4-bit 模式	-
CMDA	0xAA	4-bit 模式	4-bit 模式	-
End_SEG_TRANS	0xA5	4-bit 模式	4-bit 模式	-
Ex_QPI	0xDD	4-bit 模式	4-bit 模式	-

GP-SPI2 收到主机发送的 0x06 CMD (En_QPI) 命令后，将进入 QPI 模式。GP-SPI2 在 QPI 模式下支持的传输类型，其后续所有阶段均为 4-bit 模式。如果收到 0xDD CMD (Ex_QPI)，则 GP-SPI2 从机将返回到 SPI 模式。

未在表 33.5-13 和表 33.5-14 中列出的传输类型将被忽略掉。但如果 CS 低电平持续时长大于 2 个 APB_CLK 时钟周期，则将触发 SPI_TRANS_DONE_INT 中断。有关传输结束时触发的中断信息，请参考章节 33.9。

33.5.10.3 从机单次传输和从机连读传输

GP-SPI2 用作从机时，支持由 DMA 和 CPU 控制的全双工和半双工通信。DMA 控制的从机传输，可以是一次单次传输，也可以是从机连续传输（包含多次传输事务）。CPU 控制的传输只能是单次传输，因为每次传输均需由 CPU 触发。

一次从机连续传输包含多个传输事务，每个传输事务可以是表 33.5-13 和表 33.5-14 列出的任一传输类型。即在一次完整的连续传输过程中，可以包含 CPU 控制的数据传输，也可以包含 DMA 控制的数据传输。

在一次完整的连续传输过程中，推荐操作如下：

- CPU 控制的数据传输可用于握手通信以及少量数据传输；
- DMA 控制的数据传输可用于大量数据传输。

33.5.10.4 配置从机单次传输

用作从机时，GP-SPI2 支持 CPU 控制的和 DMA 控制的全/半双工单次传输。具体的寄存器配置如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟、APB 时钟（AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 使能从机传输；
4. 配置 SPI_DOUTDIN：
    - 1：使能全双工通信；
    - 0：使能半双工通信。
5. 准备数据：

- 如果选择的传输类型为 CPU 控制的传输，且 GP-SPI2 发送数据，则在寄存器 SPI_W0_REG~SPI_W15_REG 中准备数据。
- 如果选择的传输类型为 DMA 控制的传输，则需要：
  - 配置 SPI_DMA_RX_ENA/SPI_DMA_TX_ENA 和 SPI_RX_EOF_EN；
  - 配置 GDMA TX/RX 链表；
  - 启动 GDMA TX/RX 引擎，更多描述见章节 33.5.7 和章节 33.5.8。

6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
7. 清零寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN 使能从机单次传输；
8. 置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_TRANS_DONE_INT_ENA，使能中断，并等待 SPI_TRANS_DONE_INT 中断。在 DMA 控制的传输下，使用 DMA RX buffer 时，推荐等待 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断，即数据已存储至相应内存。其他中断见章节 33.9。

33.5.10.5 配置半双工通信下从机连续传输

此时必须使用 GDMA。具体的寄存器配置如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟、APB 时钟（AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 使能从机传输；
4. 清除 SPI_DOUTDIN 使能半双工通信方式；
5. 根据需求，确定是否需要在寄存器 SPI_W0_REG~SPI_W15_REG 中准备数据；
6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
7. 置位 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA。清零 SPI_RX_EOF_EN。配置 GDMA TX/RX 链表，并启动 GDMA TX/RX 引擎，更多描述见章节 33.5.7 和章节 33.5.8；
8. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN，使能从机连续传输；
9. 置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_DMA_SEG_TRANS_DONE_INT_ENA，使能中断，并等待 SPI_DMA_SEG_TRANS_DONE_INT 中断。中断发生，即表明从机连续传输已结束，且数据已放入相应的内存中。其他中断见章节 33.9。

GP-SPI2 收到 End_SEG_TRANS 命令（SPI 模式下为 0x05，QPI 模式下为 0xA5），从机连续传输结束，并触发 SPI_DMA_SEG_TRANS_DONE_INT 中断。

33.5.10.6 配置全双工通信下从机连续传输

在这一传输中，必须使用 GDMA。数据从 GDMA buffer 中输入输出。传输结束，触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。具体的配置程序如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB 时钟、APB 时钟（AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 和 SPI_DOUTDIN，使能从机全双工通信；
4. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；

5. 置位 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA。配置 GDMA TX/RX 链表，并启动 GDMA TX/RX 引擎，更多描述见章节 33.5.7 和章节 33.5.8；
6. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_RX_EOF_EN。在寄存器 SPI_MS_DLEN_REG 的 SPI_MS_DATA_BITLEN[17:0] 中配置 DMA 接收数据长度（单位：字节）；
7. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN，使能从机连续传输；
8. 置位 AHB_DMA_IN_SUC_EOF_CH*n*_INT_ENA 使能中断，然后等待 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。

33.6 CS 建立时间和保持时间控制

SPI CS 建立时间和保持时间对于满足各种 SPI 设备（如 flash 或 PSRAM）的时序要求非常重要。

CS 建立时间为 CS 下降沿至 SPI_CLK 第一个锁存边沿的时间。模式 0 和模式 3 的第一锁存边沿为上升沿，模式 1 和模式 2 的第一锁存边沿为下降沿。

CS 保持时间为 SPI_CLK 最后一个锁存边沿到 CS 上升沿之间的时间。

用作从机时，CS 建立时间和保持时间应大于 0.5 x T_SPI_CLK，否则 SPI 传输可能出错。这里的 T_SPI_CLK 指 SPI_CLK 时钟周期。

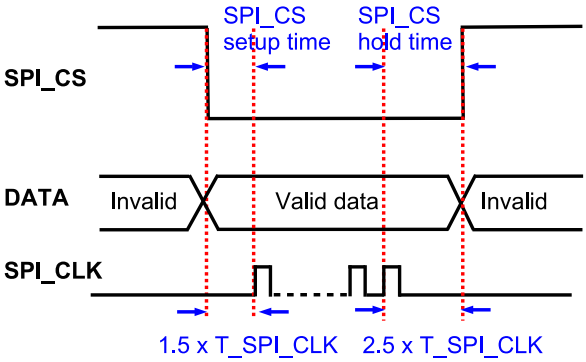
用作主机时，CS 建立时间由寄存器 SPI_USER_REG 中的 SPI_CS_SETUP 位和寄存器 SPI_USER1_REG 中的 SPI_CS_SETUP_TIME 位控制：

- 清零 SPI_CS_SETUP，则 SPI CS 建立时间为 0.5 x T_SPI_CLK；
- 置位 SPI_CS_SETUP，则 SPI CS 建立时间为 (SPI_CS_SETUP_TIME + 1.5) x T_SPI_CLK。

CS 保持时间由寄存器 SPI_USER_REG 中的 SPI_CS_HOLD 位和寄存器 SPI_USER1_REG 中的 SPI_CS_HOLD_TIME 位控制：

- 清零 SPI_CS_HOLD，则 SPI CS 保持时间为 0.5 x T_SPI_CLK；
- 置位 SPI_CS_HOLD，则 SPI CS 保持时间为 (SPI_CS_HOLD_TIME + 1.5) x T_SPI_CLK。

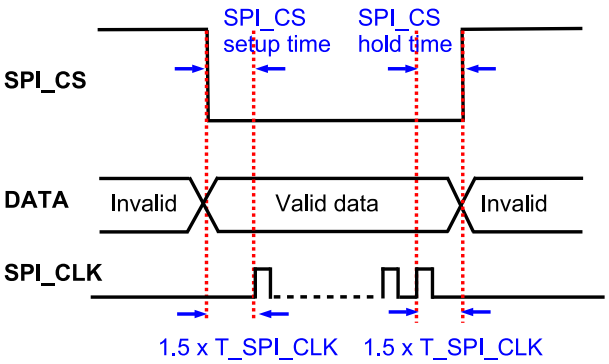
图 33.6-1 和图 33.6-2 所示为访问外部 RAM 和 flash 时推荐的 CS 时序配置和寄存器配置。



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 1.

图 33.6-1. GP-SPI2 访问外部 RAM 时推荐的 CS 时序配置



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 0.

图 33.6-2. GP-SPI2 访问 Flash 时推荐的 CS 时序配置

33.7 GP-SPI2 时钟控制

GP-SPI2 中有以下四个时钟：

- clk_hclk：GP-SPI2 模块的时序补偿时钟，与 clk_spi_mst 同源，一般是 clk_spi_mst 的 2 倍频；
- clk_spi_mst：GP-SPI2 模块时钟，在 GP-SPI2 用作主机时用于生成数据传输以及从机所需的 SPI_CLK 信号，时钟频率最高不超过 80MHz；
- SPI_CLK：主机输出时钟；
- AHB_CLK：用于寄存器配置的时钟。

其中，clk_spi_mst 与 clk_hclk 时钟的开关寄存器为 PCR_SPI2_CLK_EN，时钟源选择寄存器为 PCR_SPI2_CLKMST_SEL，clk_hclk 时钟频率与如下源时钟保持相同，clk_spi_mst 可通过

PCR_SPI2_CLKM_DIV_NUM 设置：

- 0：时钟源为 XTAL_CLK；
- 1：时钟源为 PLL_F160M_CLK；
- 2：时钟源为 FOSC_CLK;
- 3：时钟源为 PLL_F120M_CLK。

用作主机时，GP-SPI2 最高输出时钟频率为 $f_{clk\_spi\_mst}$。如果需要较低的时钟频率，可以采用如下分频方式：

$$f_{SPI\_CLK} = \frac{f_{clk\_spi\_mst}}{(SPI\_CLKCNT\_N + 1)(SPI\_CLKDIV\_PRE + 1)}$$

用户可配置寄存器 SPI_CLOCK_REG 中 SPI_CLKCNT_N 和 SPI_CLKDIV_PRE 设置分频系数。寄存器 SPI_CLOCK_REG 中 SPI_CLK_EQU_SYSCLK 位置 1 时，GP-SPI 的输出时钟频率为 $f_{clk\_spi\_mst}$。如果采用其他整数分频，则 SPI_CLK_EQU_SYSCLK 应置 0。

用作从机时，GP-SPI2 支持的输入时钟频率为 $f_{SPI\_CLK} <= f_{AHB\_CLK}$。

33.7.1 时钟相位和极性

SPI 协议支持四种时钟模式，即模式 0~3，见图 33.7-1 和图 33.7-2。

说明：
图片来源于 SPI 协议，两图中 SAMPLE 信号的黑色表示标准采样沿，红色表示 GP-SPI 的默认采样沿。

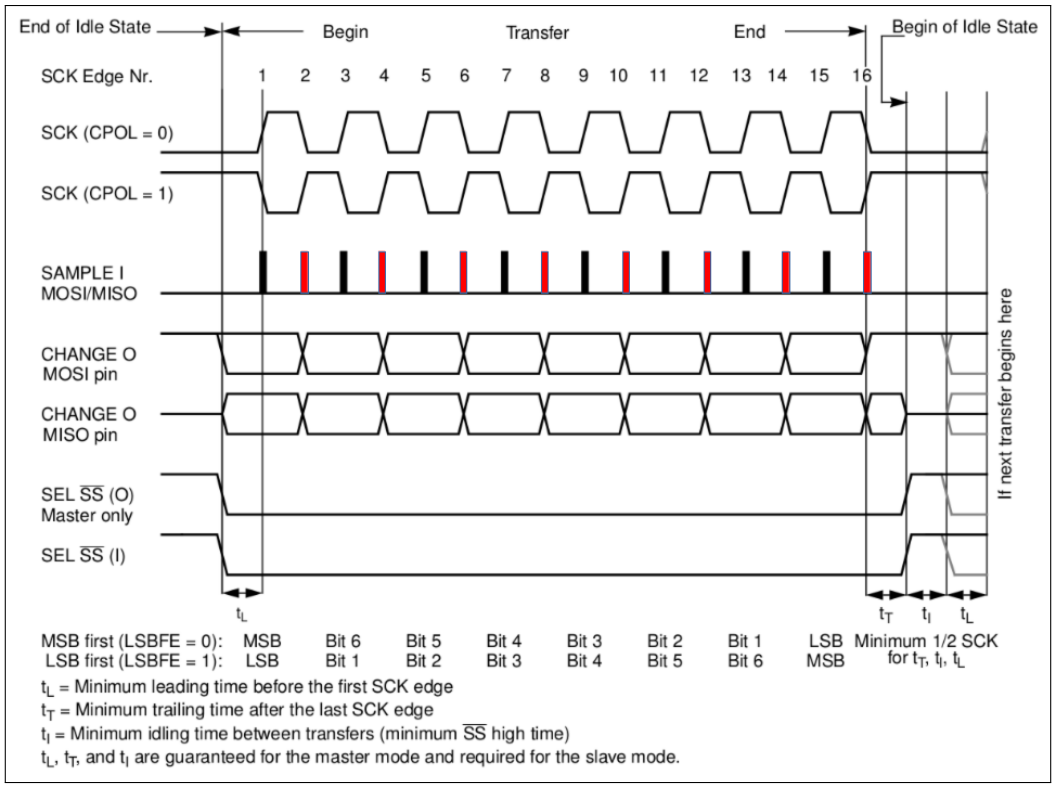


图 33.7-1. SPI 时钟模式 0 和时钟模式 2

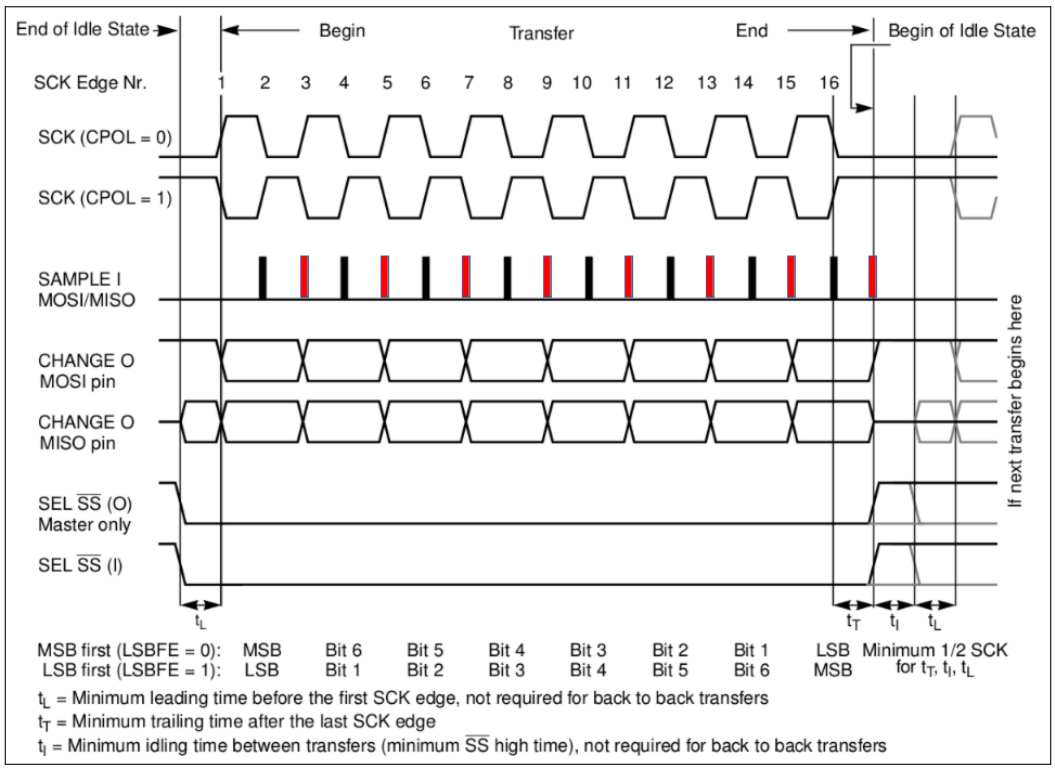


图 33.7-2. SPI 时钟模式 1 和时钟模式 3

1. 模式 0：CPOL = 0，CPHA = 0；SPI 处于空闲模式时，SCK 为 0；数据在 SCK 下降沿变化，在上升沿采样。第一个数据在 SCK 的第一个下降沿之前被移出。
2. 模式 1：CPOL = 0，CPHA = 1；SPI 处于空闲模式时，SCK 为 0；数据在 SCK 上升沿变化，在下降沿采样。
3. 模式 2：CPOL = 1，CPHA = 0；SPI 处于空闲模式时，SCK 为 1；数据在 SCK 上升沿变化，在下降沿采样。第一个数据在 SCK 的第一个上升沿之前被移出。
4. 模式 3：CPOL = 1，CPHA = 1；SPI 处于空闲模式时，SCK 为 1；数据在 SCK 下降沿变化，在上升沿采样。

33.7.2 主机时钟控制

GP-SPI2 主机支持多种 SPI 时钟模式：模式 0~3。GP-SPI2 极性和相位由寄存器 SPI_MISC_REG 中 SPI_CK_IDLE_EDGE 位和寄存器 SPI_USER_REG 中 SPI_CK_OUT_EDGE 位控制。SPI 时钟模式 0~3 的寄存器配置见表 33.7-1，可根据应用的路径延迟进行更改。

表 33.7-1. 主机时钟相位和极性配置

寄存器控制位	模式 0	模式 1	模式 2	模式 3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0

此外，SPI_CLK_MODE 可用于选择 CS 拉高时 SPI_CLK 的上升沿个数：0、1、2 或 SPI_CLK 一直有效。

说明：

SPI_CLK_MODE 配置成 1 或 2 时，必须置位 SPI_CS_HOLD 且 SPI_CS_HOLD_TIME 的值需大于 1。

33.7.3 从机时钟控制

GP-SPI2 从机也支持四种 SPI 时钟模式：即模式 0~3。寄存器 SPI_USER_REG 中 SPI_TSCK_I_EDGE 和 SPI_RSCK_I_EDGE 位可用于配置时钟极性和相位。数据的输出沿则由寄存器 SPI_SLAVE_REG 中的 SPI_CLK_MODE_13 位控制。寄存器具体配置见表 33.7-2。

表 33.7-2. 从机时钟相位和极性配置

寄存器控制位	模式 0	模式 1	模式 2	模式 3
SPI_TSCK_I_EDGE	0	1	1	0
SPI_RSCK_I_EDGE	0	1	1	0
SPI_CLK_MODE_13	0	1	0	1

33.8 GP-SPI2 时序补偿

概述根据 GP-SPI2 在用户 SPI 传输系统中的角色（主机或从机），提供如下时序补偿方案：

- 在主机模式下，默认采用延后半拍 SPI_CLK 的时序补偿方案，如图 33.7-1 和图 33.7-2 所示；也可通过配置寄存器切换为标准采样模式。
- 若使用串联寄存器补偿方案，则在默认延后半拍的基础上，进一步增强高速传输下的时序补偿能力。
- 在从机模式下，默认按标准 SPI 协议发送数据，可以通过寄存器设置提前半拍发送数据。

GP-SPI2 用作主机时，默认延后半拍 SPI_CLK 采样从机返回的数据，可以有效补偿 GP-SPI2 IO 时序，也可置位 SPI_CLK_EDGE_SEL 切换为符合 SPI 标准协议的采样模式。请注意，SPI_CLK_EQU_SYSCLK 位置 1 时，固定延后半拍 SPI_CLK 采样从机返回的数据，此时可采用其他方法进行 IO 时序补偿。

SPI 输入输出信号可通过 GPIO 矩阵或 IO MUX 映射到芯片管脚，但 IO MUX 不支持时序调整。输入输出数据在 GPIO 矩阵模块中，可在上升沿或下降沿延迟 1 或 2 个 IO MUX 运行时钟周期。更多寄存器配置信息，见章节 8 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*。

图 33.8-1 所示为 GP-SPI2 用作主机时的时序补偿控制，包括以下路径：

- “CLK”：GP-SPI2 总线时钟信号的输出路径。时钟由 SPI_CLK 输出控制模块发送，经过 GPIO 矩阵或 IO MUX，然后到达外部 SPI 设备。
- “IN”：GP-SPI2 的数据输入路径，即图 33.8-1 中路径 3（紫色）。来自外部 SPI 设备的输入数据通过 GPIO 矩阵或 IO MUX，由时序模块（见图 33.5-2）进行调整，最后存储到 *spi_rx_afifo*。
- “OUT”：GP-SPI2 的数据输出路径，即图 33.8-1 中路径 2（玫红色）。输出数据发送到时序模块，经过 GPIO 矩阵或 IO MUX，最后由外部 SPI 设备捕获。

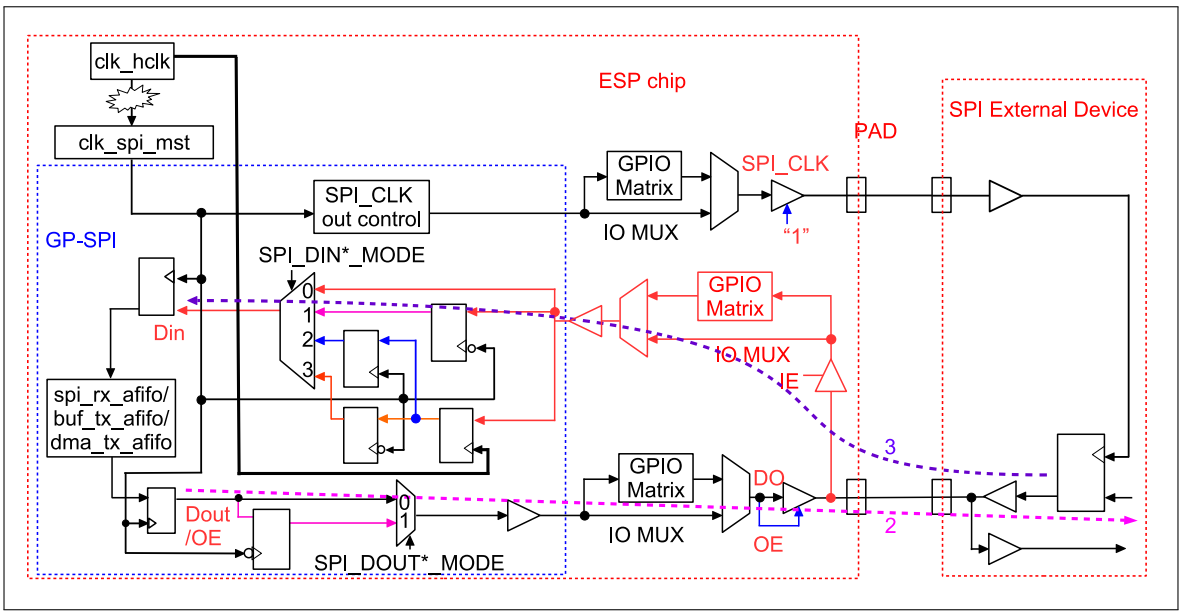


图 33.8-1. GP-SPI2 主机时序补偿控制图

时序调制模块同时适用于输入数据和输出数据，数据可在时钟上升沿或下降沿延迟整数个 clk_spi_mst 周期，即整数个 $T_{clk\_spi\_mst}$。

关键寄存器

- SPI_DIN_MODE_REG：用于选择输入数据的锁存沿；
- SPI_DIN_NUM_REG：用于选择输入数据的延迟周期；
- SPI_DOUT_MODE_REG：用于选择输出数据的锁存沿。

时序补偿应用示例

图 33.8-2 所示为 GP-SPI2 用作主机时的时序补偿示例。同时，DUMMY 周期长可更改，用于补偿实际的 I/O 线路延迟，从而提高 GP-SPI2 性能。

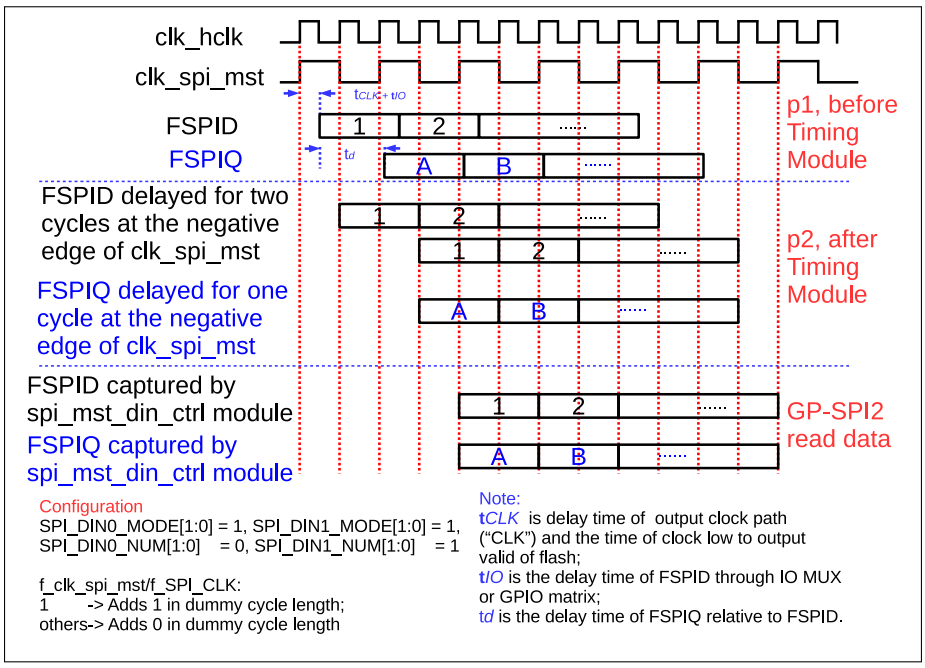


图 33.8-2. GP-SPI2 主机时序补偿示例

图 33.8-2 中，“p1” 为时序模块输入数据的时间点；“p2” 为输出数据的时间点。由于 FSPIQ 输入数据并未与 FSPID 输入数据对齐，如果没有时间补偿的话，GP-SPI2 读取数据将会出错。

为了正确读取数据，需要进行如下配置。其中，假设 $f_{clk\_spi\_mst}$ 等于 f_{SPI_CLK}：

- 将 FSPID 在 clk_spi_mst 下降沿延迟两个时钟周期
- 将 FSPIQ 在 clk_spi_mst 下降沿延迟一个时钟周期
- 增加一个额外的 DUMMY 周期

GP-SPI2 用作从机时，如果寄存器 SPI_SLAVE_REG 中 SPI_RSCK_DATA_OUT 置 1，则在锁存沿发送输出数据，即提前半个 SPI 时钟周期。上述功能可用于从机时序补偿。

33.9 中断

33.9.1 中断描述

GP-SPI2 可以生成如下外部中断信号：

- SPI_INTR

SPI_INTR 可通过中断矩阵映射至 CPU0。中断信号由 GP-SPI2 模块的内部中断源生成。表 33.9-1 列出了 GP-SPI2 的内部中断源、中断触发条件、以及生成的中断信号。

表 33.9-1. GP-SPI2 的内部中断源

内部中断源	触发条件	中断信号
SPI_DMA_INFIFO_FULL_ERR_INT	GDMA RX FIFO 小于实际传输的数据长度	SPI_INTR
SPI_DMA_OUTFIFO_EMPTY_ERR_INT	GDMA TX FIFO 小于实际传输的数据长度	SPI_INTR

见下页

表 33.9-1 – 接上页

内部中断源	触发条件	中断信号
SPI_SLV_EX_QPI_INT	GP-SPI2 从机模式下，正确接收 Ex_QPI 命令，且 SPI 传输结束	SPI_INTR
SPI_SLV_EN_QPI_INT	GP-SPI2 从机模式下，正确接收 En_QPI 命令，且 SPI 传输结束	SPI_INTR
SPI_SLV_CMD7_INT	GP-SPI2 从机模式下，正确接收 CMD7 命令，且 SPI 传输结束	SPI_INTR
SPI_SLV_CMD8_INT	GP-SPI2 从机模式下，正确接收 CMD8 命令，且 SPI 传输结束	SPI_INTR
SPI_SLV_CMD9_INT	GP-SPI2 从机模式下，正确接收 CMD9 命令，且 SPI 传输结束	SPI_INTR
SPI_SLV_CMDA_INT	GP-SPI2 从机模式下，正确接收 CMDA 命令，且 SPI 传输结束	SPI_INTR
SPI_SLV_RD_DMA_DONE_INT	从机模式下，Rd_DMA 传输结束	SPI_INTR
SPI_SLV_WR_DMA_DONE_INT	从机模式下，Wr_DMA 传输结束	SPI_INTR
SPI_SLV_RD_BUF_DONE_INT	从机模式下，Rd_BUF 传输结束	SPI_INTR
SPI_SLV_WR_BUF_DONE_INT	从机模式下，Wr_BUF 传输结束	SPI_INTR
SPI_TRANS_DONE_INT	主从机模式下，SPI 总线传输结束	SPI_INTR
SPI_DMA_SEG_TRANS_DONE_INT	GP-SPI2 从机连续传输模式下，END_SEG_TRANS 传输结束。主机模式下，分段配置传输结束	SPI_INTR
SPI_SEG_MAGIC_ERR_INT	在主机分段配置传输模式下，CONF buffer 中的 Magic 值有误	SPI_INTR
SPI_MST_RX_AFIFO_WFULL_ERR_INT	GP-SPI 主机模式下，发生 RX AFIFO write-full 错误	SPI_INTR
SPI_MST_TX_AFIFO_REMPTY_ERR_INT	GP-SPI 主机模式下，发生 TX AFIFO read-empty 错误	SPI_INTR
SPI_SLV_CMD_ERR_INT	GP-SPI 从机模式下，接收到 GP-SPI 不支持的命令值	SPI_INTR
SPI_APP2_INT	置位 SPI_APP2_INT_SET（仅用于用户自定义的功能）	SPI_INTR
SPI_APP1_INT	置位 SPI_APP1_INT_SET（仅用于用户自定义的功能）	SPI_INTR

33.9.2 GP-SPI2 用作主机和从机时分别用到的中断

表 33.9-2 和表 33.9-3 分别列出了 GP-SPI2 用作主机和用作从机时用到的中断。置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_*_INT_ENA 位，使能相应中断，并等待 SPI_INT 中断。传输结束时，将触发相关中断。注意，在下次传输之前，需软件清除中断。

表 33.9-2. GP-SPI2 用作主机时用到的中断

传输类型	通信模式	控制方式	中断
单次传输	全双工	DMA	AHB_DMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	半双工主机输出从机输入	DMA	SPI_TRANS_DONE_INT
		CPU	SPI_TRANS_DONE_INT
	半双工主机输入从机输出	DMA	AHB_DMA_IN_SUC_EOF_CH*n*_INT

接上页

表 33.9-2 – 见下页

传输类型	通信模式	控制方式	中断
		CPU	SPI_TRANS_DONE_INT
分段配置传输	全双工	DMA	SPI_DMA_SEG_TRANS_DONE_INT [3]
		CPU	不支持
	半双工主机输出从机输入	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	不支持
	半双工主机输入从机输出	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	不支持

[1] 如果触发了 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断，则表示 GP-SPI2 的所有 RX 数据已保存至 RX buffer，且所有 TX 数据已发送至从机。

[2] CS 拉高，则将触发 SPI_TRANS_DONE_INT 中断，表明主机与从机已完成 SPI_W0_REG ~ SPI_W15_REG 的数据交换。

[3] 如果触发了 SPI_DMA_SEG_TRANS_DONE_INT 中断，则表明整个分段配置传输，包括若干个传输事务，已完成。即 RX 数据已全部存入 RX buffer 且所有 TX 数据已发送完毕。

表 33.9-3. GP-SPI2 用作从机时用到的中断

传输类型	通信模式	控制方式	中断
单次传输	全双工	DMA	AHB_DMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	半双工主机输出从机输入	DMA (Wr_DMA)	AHB_DMA_IN_SUC_EOF_CH*n*_INT [3]
		CPU (Wr_BUF)	SPI_TRANS_DONE_INT [4]
	半双工主机输入从机输出	DMA (Rd_DMA)	SPI_TRANS_DONE_INT [5]
		CPU (Rd_BUF)	SPI_TRANS_DONE_INT [6]
从机连续传输	全双工	DMA	AHB_DMA_IN_SUC_EOF_CH*n*_INT [7]
		CPU	不支持 [8]
	半双工主机输出从机输入	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [9]
		CPU (Wr_BUF)	不支持 [10]
	半双工主机输入从机输出	DMA (Rd_DMA)	SPI_DMA_SEG_TRANS_DONE_INT [11]
		CPU (Rd_BUF)	不支持 [12]

[1] 如果触发了 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断，则表示所有 RX 数据已保存至 RX buffer，且所有 TX 数据已发送至从机。
[2] CS 拉高，则将触发 SPI_TRANS_DONE_INT 中断，表明主机与从机已完成 SPI_W0_REG ~ SPI_W15_REG 的数据交换。
[3] 触发 SPI_SLV_WR_DMA_DONE_INT 中断仅表示 SPI 总线上的数据传输已完成，但并不能保证所有入栈数据已存至 RX buffer。因此，推荐使用 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。
[4] 或等待 SPI_SLV_WR_BUF_DONE_INT 中断。
[5] 或等待 SPI_SLV_RD_DMA_DONE_INT 中断。
[6] 或等待 SPI_SLV_RD_BUF_DONE_INT 中断。
[7] 传输开始前，从机应在 SPI_MS_DATA_BITLEN 中设置读数据的总长度。并在中断程序结束前，置位 SPI_RX_EOF_EN。
[8] 主机和从机需定义连续传输结束的方式，比如配置 GPIO 用作中断等。
[9] 主机发送 End_SEG_TRAN 结束连续传输，或从机在 SPI_MS_DATA_BITLEN 中配置总的读数据长度，然后等待 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。
[10] 半双工 Wr_BUF 单次传输也可用于 DMA 控制的从机连续传输中。
[11] 主机发送 End_SEG_TRAN 结束从机连续传输。
[12] 半双工 Rd_BUF 单次传输也可用于 DMA 控制的从机连续传输中。

33.10 寄存器列表

本小节的所有地址均为相对于 GP-SPI2 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	地址	访问
自定义控制寄存器			
SPI_CMD_REG	命令控制寄存器	0x0000	varies
SPI_ADDR_REG	地址值寄存器	0x0004	R/W
SPI_USER_REG	SPI 用户控制寄存器	0x0010	varies
SPI_USER1_REG	SPI 用户控制寄存器 1	0x0014	R/W
SPI_USER2_REG	SPI 用户控制寄存器 2	0x0018	R/W
控制和配置寄存器			
SPI_CTRL_REG	SPI 控制寄存器	0x0008	varies
SPI_MS_DLEN_REG	SPI 数据位长控制寄存器	0x001C	R/W
SPI_MISC_REG	SPI MISC 寄存器	0x0020	varies
SPI_DMA_CONF_REG	SPI DMA 控制寄存器	0x0030	varies
SPI_SLAVE_REG	SPI 从机控制寄存器	0x00E0	varies
SPI_SLAVE1_REG	SPI 从机控制寄存器 1	0x00E4	R/W/SS
时钟控制寄存器			
SPI_CLOCK_REG	SPI 时钟控制寄存器	0x000C	R/W
SPI_CLK_GATE_REG	SPI 模块时钟和寄存器时钟控制寄存器	0x00E8	R/W
时序控制寄存器			
SPI_DIN_MODE_REG	SPI 输入延迟模式配置寄存器	0x0024	varies

名称	描述	地址	访问
SPI_DIN_NUM_REG	SPI 输入延迟周期配置寄存器	0x0028	varies
SPI_DOUT_MODE_REG	SPI 输出延迟模式配置寄存器	0x002C	varies
中断寄存器			
SPI_DMA_INT_ENA_REG	SPI 中断使能寄存器	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI 中断清除寄存器	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI 原始中断寄存器	0x003C	R/WTC/SS
SPI_DMA_INT_ST_REG	SPI 中断状态寄存器	0x0040	RO
SPI_DMA_INT_SET_REG	SPI 中断软件置位寄存器	0x0044	WT
CPU 数据 buffer			
SPI_W0_REG	SPI CPU 控制的 buffer 0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU 控制的 buffer 1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU 控制的 buffer 2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU 控制的 buffer 3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU 控制的 buffer 4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU 控制的 buffer 5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU 控制的 buffer 6	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU 控制的 buffer 7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU 控制的 buffer 8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU 控制的 buffer 9	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU 控制的 buffer 10	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU 控制的 buffer 11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU 控制的 buffer 12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU 控制的 buffer 13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU 控制的 buffer 14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU 控制的 buffer 15	0x00D4	R/W/SS
版本寄存器			
SPI_DATE_REG	版本控制寄存器	0x00F0	R/W

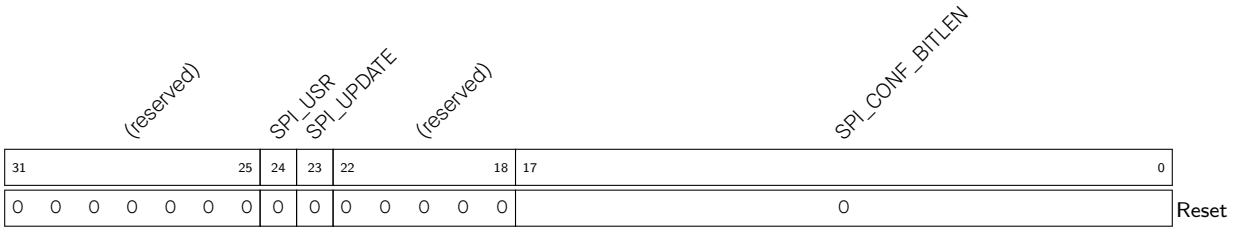
33.11 寄存器

注意：
ESP32-C5 暂不支持本章节中权限为 HRO 的字段所涉及的功能。

本小节的所有地址均为相对于 GP-SPI2 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

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

Register 33.1. SPI_CMD_REG (0x0000)



SPI_CONF_BITLEN 配置 SPI CONF 阶段的 SPI_CLK 周期。

单位：SPI_CLK 时钟周期。

可在 CONF 阶段配置。

(R/W)

SPI_UPDATE 配置是否将 SPI 寄存器从 APB 时钟域同步到 SPI 模块时钟域。

0：不同步

1：同步

仅用于 SPI 主机。

(WT)

SPI_USR 配置是否使能用户自定义命令。

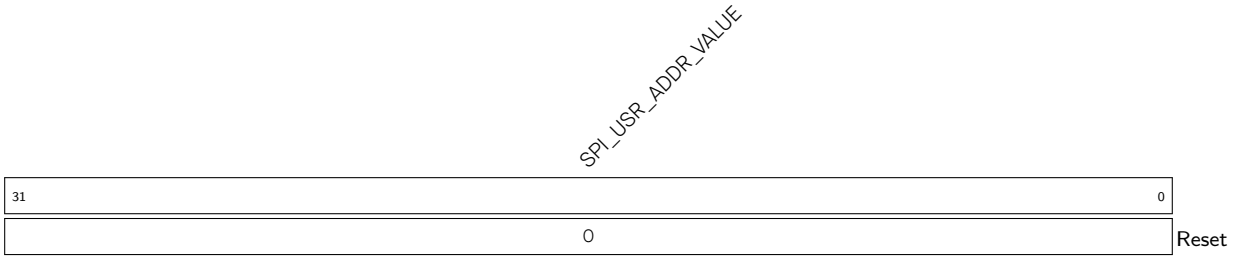
0：不使能

1：使能

置位此位将触发一次 SPI 操作。操作结束后此位被自动清零。CONF_buf 不可更改该配置。

(R/W/SC)

Register 33.2. SPI_ADDR_REG (0x0004)

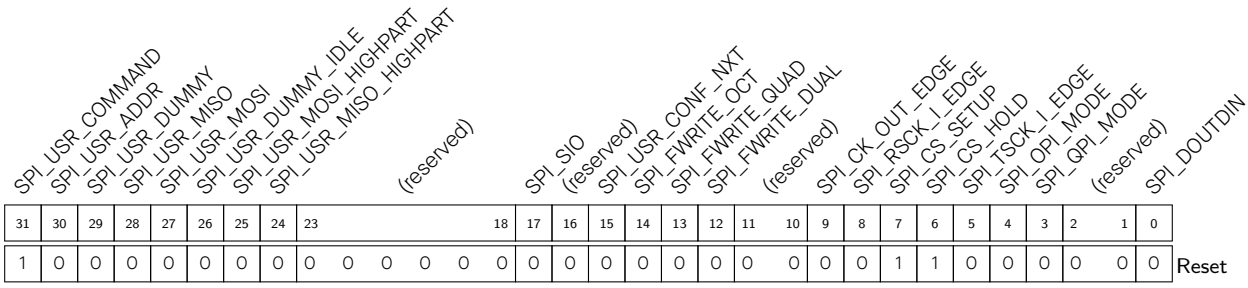


SPI_USR_ADDR_VALUE 配置从机地址。

可在 CONF 阶段配置。

(R/W)

Register 33.3. SPI_USER_REG (0x0010)



SPI_DOUTDIN 配置是否使能全双工通信。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_QPI_MODE 配置是否使能 QPI 模式。

0：不使能

1：使能

SPI 主机和从机均支持该配置。可在 CONF 阶段配置。

(R/W/SS/SC)

SPI_OPI_MODE 保留 (HRO)

SPI_TSCK_I_EDGE 配置 SPI 从机传输中是否更改 TSCK 极性。

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

SPI_CS_HOLD 配置在 SPI 处于完成 (DONE) 阶段时是否保持 SPI CS 拉低。

0：不保持拉低

1：保持拉低

可在 CONF 阶段配置。

(R/W)

SPI_CS_SETUP 配置在 SPI 处于准备 (PREP) 阶段时是否使能 SPI CS。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

见下页

Register 33.3. SPI_USER_REG (0x0010)

接上页

SPI_RSCK_I_EDGE 配置 SPI 从机传输中是否更改 RSCK 极性。

0: RSCK = !SPI_CK_I

1: RSCK = SPI_CK_I

(R/W)

SPI_CK_OUT_EDGE 配置 SPI 时钟模式，与 SPI_CK_IDLE_EDGE 搭配使用。

可在 CONF 阶段配置。更多信息见章节 33.7.2。

(R/W)

SPI_FWRITE_DUAL 配置在写操作 (DOUT) 阶段时读数据的方式是否为 2-bit 方式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FWRITE_QUAD 配置在写操作 (DOUT) 阶段时读数据的方式是否为 4-bit 方式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FWRITE_OCT 保留 (HRO)

见下页

Register 33.3. SPI_USER_REG (0x0010)

接上页

SPI_USR_CONF_NXT 配置是否使能下一次传输事务的 CONF 阶段。

0：当前传输事务结束后，本次分段配置传输结束。或者，当前的传输模式不是分段配置传输。

1：本次分段配置传输继续进行，开始下一次传输事务。

可在 CONF 阶段配置。

(R/W)

SPI_SIO 配置是否使能 3 线半双工通信，其中 MOSI 和 MISO 信号共享一个管脚。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_MISO_HIGHPART 配置在读数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_MOSI_HIGHPART 配置在写数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_DUMMY_IDLE 配置是否在 DUMMY 阶段禁用 SPI 时钟。

0：不禁用

1：禁用

可在 CONF 阶段配置。

(R/W)

SPI_USR_MOSI 配置是否使能一次操作的写数据 (DOUT) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_MISO 配置是否使能一次操作的读数据 (DIN) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

见下页

Register 33.3. SPI_USER_REG (0x0010)

接上页

SPI_USR_DUMMY 配置是否使能一次操作的 DUMMY 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_ADDR 配置是否使能一次操作的地址 (ADDR) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_USR_COMMAND 配置是否使能一次操作的命令 (CMD) 阶段。

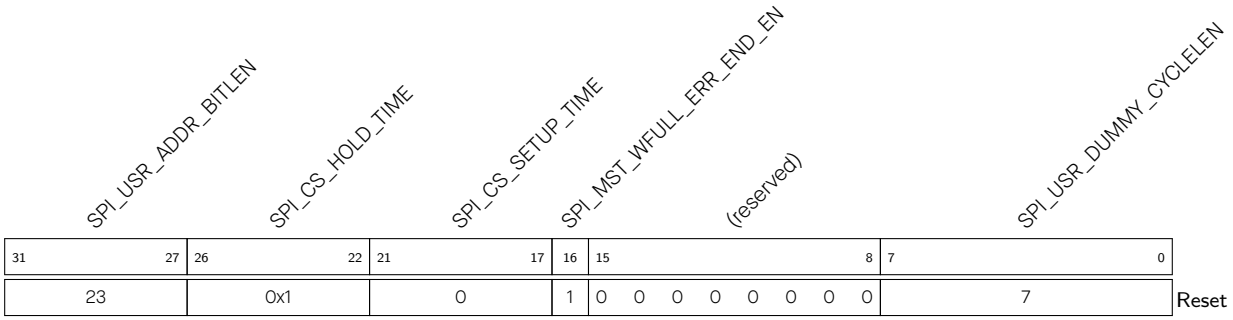
0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

Register 33.4. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN 配置 DUMMY 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。

可在 CONF 阶段配置。

(R/W)

SPI_MST_WFULL_ERR_END_EN 配置在主机全双工或半双工通信中，如果发生 RX AFIFO 满错误，是否终止 SPI 传输。

0：不终止

1：终止

(R/W)

SPI_CS_SETUP_TIME 配置准备 (PREP) 阶段的时长。

单位：SPI_CLK 时钟周期。

该值等于预期周期数 - 1。此字段与 SPI_CS_SETUP 搭配使用。

可在 CONF 阶段配置。

(R/W)

SPI_CS_HOLD_TIME 配置 CS 管脚的延迟周期。

单位：SPI_CLK 时钟周期。

此字段与 SPI_CS_HOLD 搭配使用。

可在 CONF 阶段配置。

(R/W)

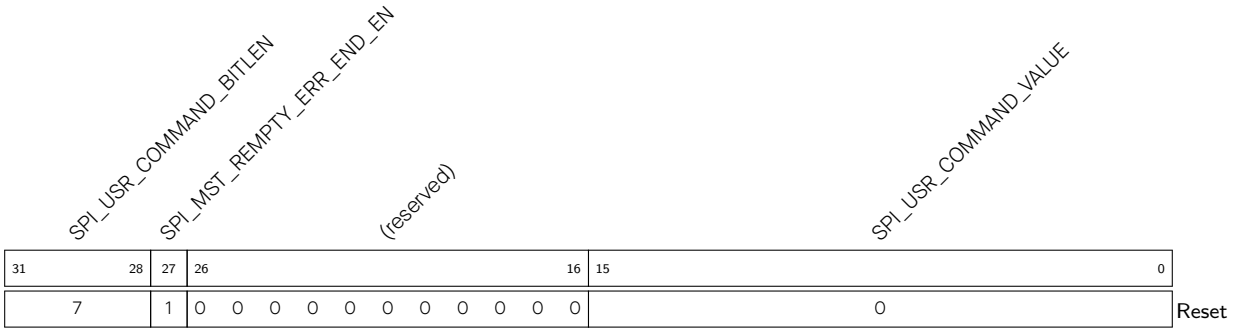
SPI_USR_ADDR_BITLEN 配置地址阶段的位长。

该值等于预期值 - 1。

可在 CONF 阶段配置。

(R/W)

Register 33.5. SPI_USER2_REG (0x0018)



SPI_USR_COMMAND_VALUE 配置命令值。

可在 CONF 阶段配置。

(R/W)

SPI_MST_REMPTY_ERR_END_EN 配置在主机全双工或半双工通信下，如果发生 TX AFIFO 空错误，是否终止 SPI 传输。

0：不终止

1：终止

(R/W)

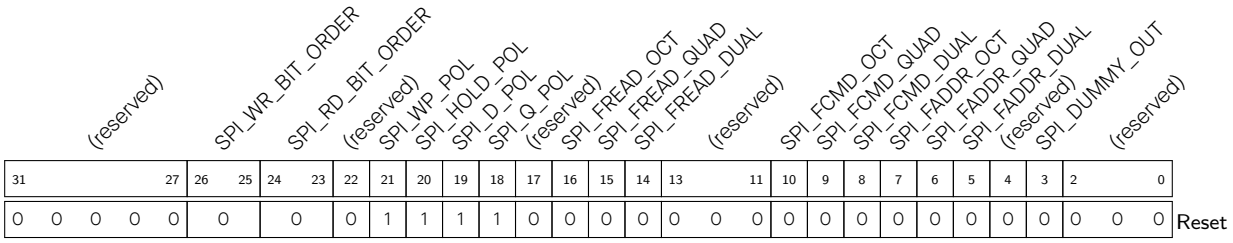
SPI_USR_COMMAND_BITLEN 配置命令阶段的位长。

该值等于预期值 - 1。

可在 CONF 阶段配置。

(R/W)

Register 33.6. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT 配置在 DUMMY 阶段是否输出 FSPI 总线信号。

0：不输出

1：输出

可在 CONF 阶段配置。

(R/W)

SPI_FADDR_DUAL 配置在地址 (ADDR) 阶段时是否使能 2-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FADDR_QUAD 配置在地址 (ADDR) 阶段时是否使能 4-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FADDR_OCT 保留 (HRO)

SPI_FCMD_DUAL 配置在命令 (CMD) 阶段时是否使能 2-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FCMD_QUAD 配置在命令 (CMD) 阶段时是否使能 4-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

见下页

Register 33.6. SPI_CTRL_REG (0x0008)

接上页

SPI_FCMD_OCT 保留 (HRO)

SPI_FREAD_DUAL 配置在读数据 (DIN) 阶段是否使能 2-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FREAD_QUAD 配置在读数据 (DIN) 阶段是否使能 4-bit 模式。

0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

SPI_FREAD_OCT 保留 (HRO)

SPI_Q_POL 配置 MISO 的极性。

0：低

1：高

可在 CONF 阶段配置。

(R/W)

SPI_D_POL 配置 MOSI 的极性。

0：低

1：高

可在 CONF 阶段配置。

(R/W)

SPI_HOLD_POL 配置在 SPI 空闲状态下 SPI_HOLD 的输出值。

0：输出低电平

1：输出高电平

可在 CONF 阶段配置。

(R/W)

见下页

Register 33.6. SPI_CTRL_REG (0x0008)

接上页

SPI_WP_POL 配置在 SPI 空闲状态下 WP 信号的输出值。

0：输出低电平

1：输出高电平

可在 CONF 阶段配置。

(R/W)

SPI_RD_BIT_ORDER 配置读数据 (MISO) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

2：先读高有效位

3：先读低有效位

可在 CONF 阶段配置，详细配置值参考表 33.5-4。

(R/W)

SPI_WR_BIT_ORDER 配置命令 (CMD)、地址 (ADDR) 和写数据 (MOSI) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

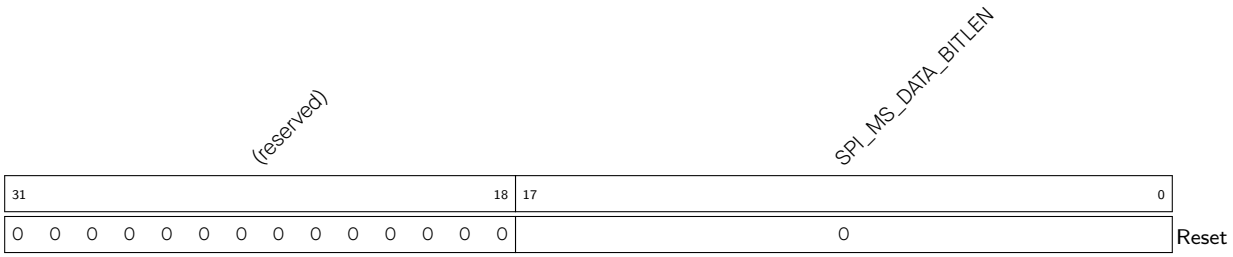
2：先读高有效位

3：先读低有效位

可在 CONF 阶段配置，详细配置值参考表 33.5-4。

(R/W)

Register 33.7. SPI_MS_DLEN_REG (0x001C)



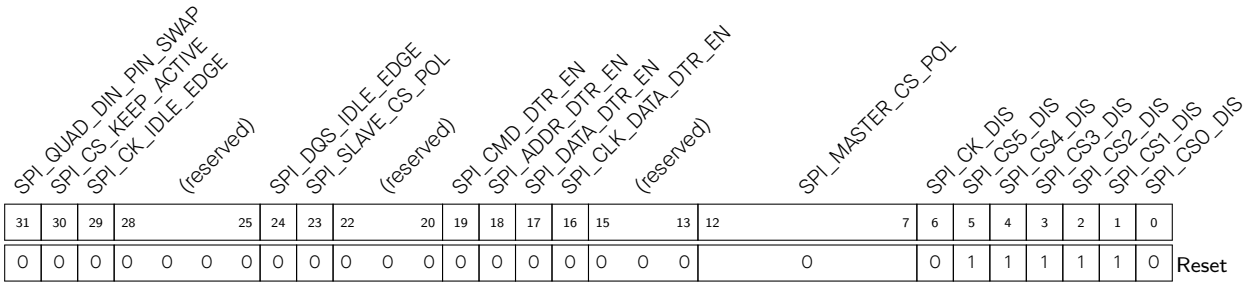
SPI_MS_DATA_BITLEN 配置 DMA 控制或 CPU 控制的主机传输的数据位长。或配置 DMA 控制的从机传输中接收数据的位长。

该值等于需要的位长 - 1。

可在 CONF 阶段配置。

(R/W)

Register 33.8. SPI_MISC_REG (0x0020)



SPI_CS0_DIS 配置是否禁用 SPI_CS0 管脚。

0：SPI_CS0 信号来自 SPI_CS0 管脚或输出至 SPI_CS0 管脚

1：禁用 SPI_CS0 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS1_DIS 配置是否禁用 SPI_CS1 管脚。

0：SPI_CS1 信号来自 SPI_CS1 管脚或输出至 SPI_CS1 管脚

1：禁用 SPI_CS1 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS2_DIS 配置是否禁用 SPI_CS2 管脚。

0：SPI_CS2 信号来自 SPI_CS2 管脚或输出至 SPI_CS2 管脚

1：禁用 SPI_CS2 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS3_DIS 配置是否禁用 SPI_CS3 管脚。

0：SPI_CS3 信号来自 SPI_CS3 管脚或输出至 SPI_CS3 管脚

1：禁用 SPI_CS3 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS4_DIS 配置是否禁用 SPI_CS4 管脚。

0：SPI_CS4 信号来自 SPI_CS4 管脚或输出至 SPI_CS4 管脚

1：禁用 SPI_CS4 管脚

可在 CONF 阶段配置。

(R/W)

SPI_CS5_DIS 配置是否禁用 SPI_CS5 管脚。

0：SPI_CS5 信号来自 SPI_CS5 管脚或输出至 SPI_CS5 管脚

1：禁用 SPI_CS5 管脚

可在 CONF 阶段配置。

(R/W)

见下页

Register 33.8. SPI_MISC_REG (0x0020)

接上页

SPI_CK_DIS 配置是否禁用 SPI_CLK 输出信号。

0：不禁用

1：禁用

可在 CONF 阶段配置。

(R/W)

SPI_MASTER_CS_POL 配置主机 SPI_CS*n* (*n* = 0 ∼ 5) 的极性。

0：SPI_CS*n* 低电平有效

1：SPI_CS*n* 高电平有效

可在 CONF 阶段配置。

(R/W)

SPI_CLK_DATA_DTR_EN 保留 (HRO)

SPI_DATA_DTR_EN 保留 (HRO)

SPI_ADDR_DTR_EN 保留 (HRO)

见下页

Register 33.8. SPI_MISC_REG (0x0020)

接上页

SPI_CMD_DTR_EN 保留 (HRO)

SPI_SLAVE_CS_POL 配置是否反相 SPI 从机输入信号 CS 的极性。

0：保持不变

1：反相

可在 CONF 阶段配置。

(R/W)

SPI_DQS_IDLE_EDGE 保留 (HRO)

SPI_CK_IDLE_EDGE 配置 SPI_CLK 线在 GP-SPI2 空闲状态时的电平。

0：低电平

1：高电平

可在 CONF 阶段配置。

(R/W)

SPI_CS_KEEP_ACTIVE 配置 SPI_CS 是否保持低电平。

0：不保持低电平

1：保持低电平

可在 CONF 阶段配置。

(R/W)

SPI_QUAD_DIN_PIN_SWAP 配置是否交换 SPI Quad 输入管脚。

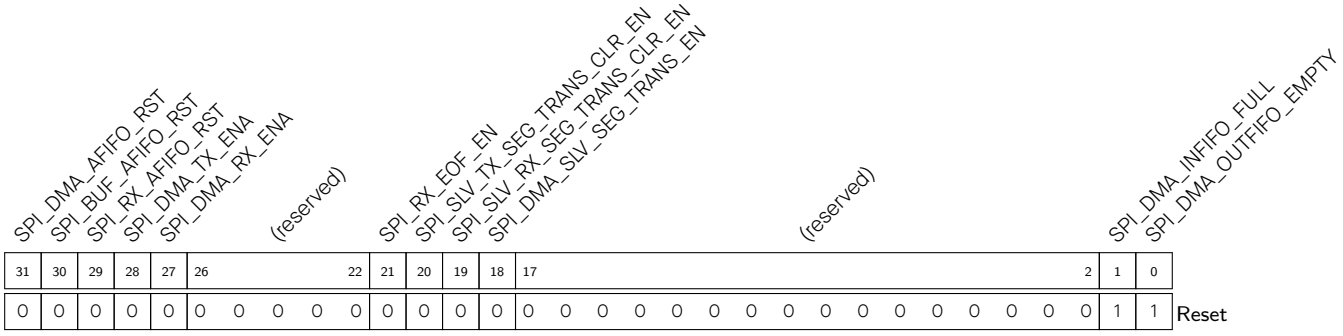
0：不交换

1：FSPID 与 FSPIQ 交换，FSPIWP 与 FSPIHD 交换

可在 CONF 阶段配置。

(R/W)

Register 33.9. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY 表示 DMA TX FIFO 是否就绪。

0：DMA TX FIFO 已就绪，可以发送数据

1：DMA TX FIFO 尚未就绪，不能发送数据

(RO)

SPI_DMA_INFIFO_FULL 表示 DMA RX FIFO 是否就绪。

0：DMA RX FIFO 已就绪，可以接收数据

1：DMA RX FIFO 尚未就绪，不能接收数据

(RO)

SPI_DMA_SLV_SEG_TRANS_EN 配置是否使能半双工通信方式下，DMA 控制的从机连续传输。

0：不使能

1：使能

(R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA RX buffer 小于实际接收的数据长度：

1：后续 Wr_DMA 传输事务中传输的数据都不接收

0：当前 Wr_DMA 传输事务中传输的数据不接收，但在后续的 Wr_DMA 传输事务中：

- 如果 DMA RX buffer 长度不为 0，则后续 Wr_DMA 传输事务中传输的数据会被接收。
- 如果 DMA RX buffer 长度为 0，则后续 Wr_DMA 传输事务中传输的数据不会被接收。

(R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA TX buffer 小于实际发送的数据长度：

1：后续传输事务中传输的数据都不更新，即旧数据被重复发送

0：当前传输事务中传输的数据不更新，但在后续的传输事务中：

- 如果有新的数据填充到 DMA TX FIFO，则将发送新数据。
- 如果没有新的数据填充到 DMA TX FIFO，则没有新数据被发送。

(R/W)

SPI_RX_EOF_EN 配置 AHB_DMA_IN_SUC_EOF_CH*n*_INT_RAW 中断的触发源。

0：在单次传输中，AHB_DMA_IN_SUC_EOF_CH*n*_INT_RAW 由 SPI_TRANS_DONE_INT 事件置位；或在分段配置传输及从机连续传输中，由 SPI_DMA_SEG_TRANS_DONE_INT 事件置位。

1：在 DMA 控制的数据传输过程中，如果 DMA 传输的数据比特数等于 (SPI_MS_DATA_BITLEN + 1)，则硬件会置位 AHB_DMA_IN_SUC_EOF_CH*n*_INT_RAW。

(R/W)

见下页

Register 33.9. SPI_DMA_CONF_REG (0x0030)

接上页

SPI_DMA_RX_ENA 配置是否使能 DMA 控制的接收数据模式。

0：不使能

1：使能

(R/W)

SPI_DMA_TX_ENA 配置是否使能 DMA 控制的发送数据模式。

0：不使能

1：使能

(R/W)

SPI_RX_AFIFO_RST 配置是否复位 spi_rx_afifo。spi_rx_afifo 将在 SPI 主机和从机传输中用于接收数据，见图 33.5-3 和图 33.5-4。

0：不复位

1：复位

(WT)

SPI_BUF_AFIFO_RST 配置是否复位 buf_tx_afifo。buf_tx_afifo 将在 CPU 控制的从机传输或主机传输中用于发送数据，见图 33.5-3 和图 33.5-4。

0：不复位

1：复位

(WT)

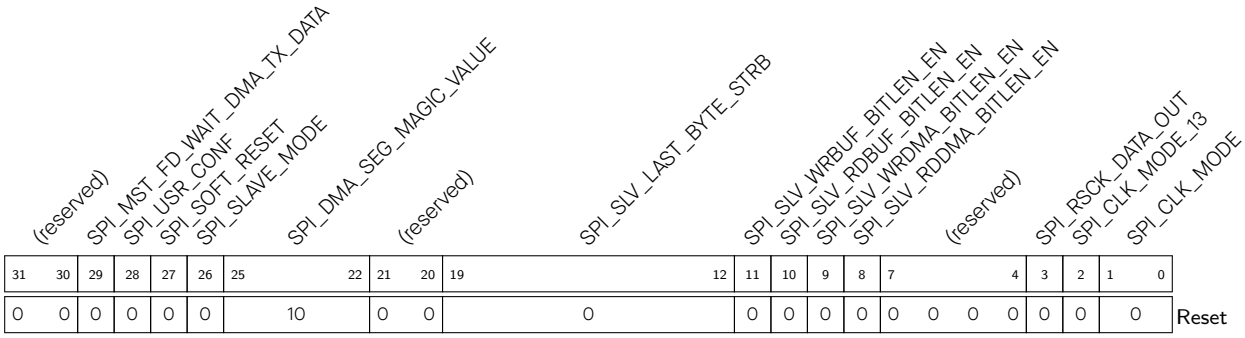
SPI_DMA_AFIFO_RST 配置是否复位 dma_tx_afifo。dma_tx_afifo 在 DMA 控制的从机传输中用于发送数据，见图 33.5-3 和图 33.5-4。

0：不复位

1：复位

(WT)

Register 33.10. SPI_SLAVE_REG (0x00E0)



SPI_CLK_MODE 配置 SPI 时钟模式。

0：CS 信号无效时，SPI 时钟关闭

1：CS 信号无效后，SPI 时钟延迟一个时钟周期

2：CS 信号无效后，SPI 时钟延迟两个时钟周期

3：SPI 时钟一直有效

可在 CONF 阶段配置。

(R/W)

SPI_CLK_MODE_13 配置时钟模式。

0：支持 SPI 时钟模式 0 或 2，见表 33.7-2。

1：支持 SPI 时钟模式 1 或 3，见表 33.7-2。

(R/W)

SPI_RSCK_DATA_OUT 配置输出数据的时钟沿。

0：在 TSCK 上升沿输出数据

1：在 RSCK 上升沿输出数据

(R/W)

SPI_SLV_RDDMA_BITLEN_EN 配置在 DMA 控制的 Rd_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

SPI_SLV_WRDMA_BITLEN_EN 配置在 DMA 控制的 Wr_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_SLV_RDBUF_BITLEN_EN 配置在 CPU 控制的 Rd_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

见下页

Register 33.10. SPI_SLAVE_REG (0x00E0)

接上页

SPI_SLV_WRBUF_BITLEN_EN 配置在 CPU 控制的 Wr_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_SLV_LAST_BYTE_STRB 代表在 SPI 主机全双工传输和半双工传输中，上次接收的字节数据的有效位。

(R/SS)

SPI_DMA_SEG_MAGIC_VALUE 配置 DMA 控制的分段配置传输中位图表的 Magic 值。

(R/W)

SPI_SLAVE_MODE 配置 SPI 工作模式。

0：主机

1：从机

(R/W)

SPI_SOFT_RESET 配置是否软件复位 SPI 时钟线、CS 线和数据线。

0：不复位

1：复位

可在 CONF 阶段配置。

(WT)

SPI_USR_CONF 配置是否使能当前 DMA 控制分段配置传输的 CONF 阶段。

0：无效，表明当前传输不是分段配置传输

1：使能，开始分段配置传输

(R/W)

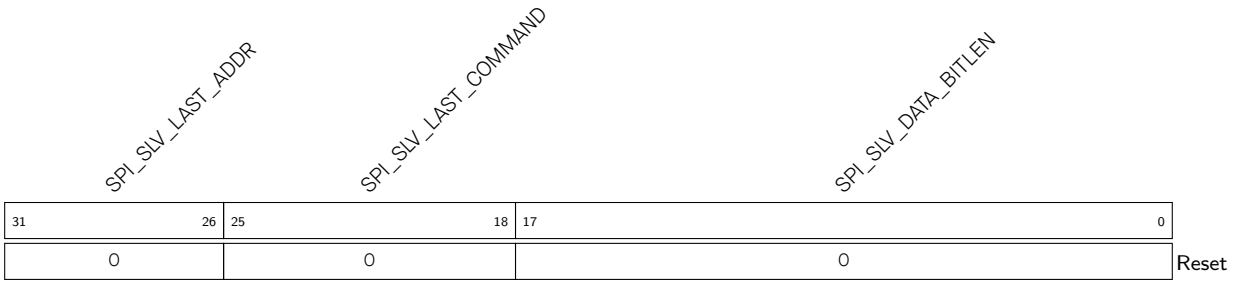
SPI_MST_FD_WAIT_DMA_TX_DATA 配置在主机全双工通信下 SPI 是否先等待 DMA TX 数据准备好之后，再开始 SPI 传输。

0：无需等待 DMA TX 数据即可开始 SPI 传输

1：等待

(R/W)

Register 33.11. SPI_SLAVE1_REG (0x00E4)

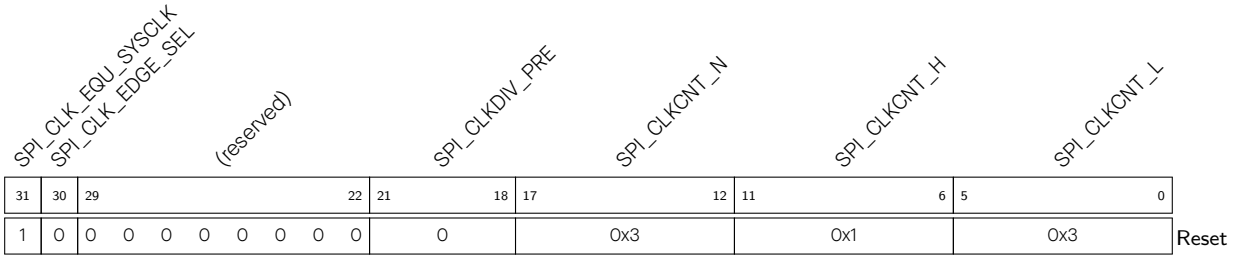


SPI_SLV_DATA_BITLEN 配置在 SPI 从机全双工和半双工传输中，传输的数据位长。
(R/W/SS)

SPI_SLV_LAST_COMMAND 配置从机的命令值。
(R/W/SS)

SPI_SLV_LAST_ADDR 配置从机的地址值。
(R/W/SS)

Register 33.12. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L 配置时钟占空比，与 SPI_CLKCNT_H 和 SPI_CLKCNT_N 配合使用。

用作主机时，必须与 SPI_CLKCNT_N 相等。

在从机模式下，必须为 0。

可在 CONF 阶段配置。

(R/W)

SPI_CLKCNT_H 配置用作主机时 SPI_CLK（高电平）的占空比。

建议将此值配置为 floor((SPI_CLKCNT_N + 1)/2 - 1)。floor() 表示向下取整值，例如 floor(2.2) = 2。

用作从机时，必须配置为 0。

可在 CONF 阶段配置。

(R/W)

SPI_CLKCNT_N 配置用作主机时 SPI_CLK 的分频系数。

SPI_CLK 频率为 $f_{clk\_spi\_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1)。

可在 CONF 阶段配置。

(R/W)

SPI_CLKDIV_PRE 配置用作主机时 SPI_CLK 的预分频系数。

可在 CONF 阶段配置。

(R/W)

SPI_CLK_EQU_SYSCLK 配置用作主机时 SPI_CLK 频率是否与 clk_spi_mst 频率相同。

0：SPI_CLK 为 clk_spi_mst 的分频时钟

1：SPI_CLK 与 clk_spi_mst 频率相同

可在 CONF 阶段配置。

(R/W)

SPI_CLK_EDGE_SEL 在主机传输中配置使用标准时钟采样边缘还是将采样边缘延迟半个周期。

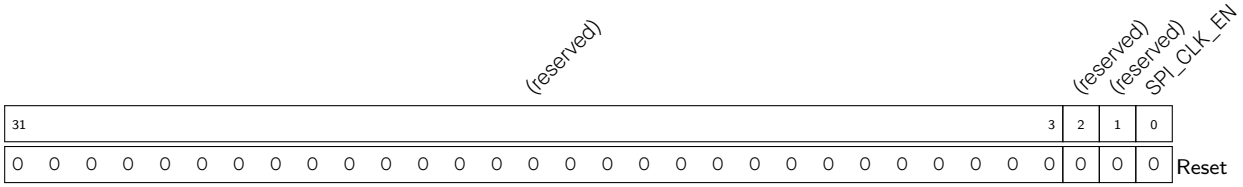
0：时钟采样边缘延迟半个周期。

1：标准时钟采样边缘。

可以在 CONF 状态中配置。

(R/W)

Register 33.13. SPI_CLK_GATE_REG (0x00E8)



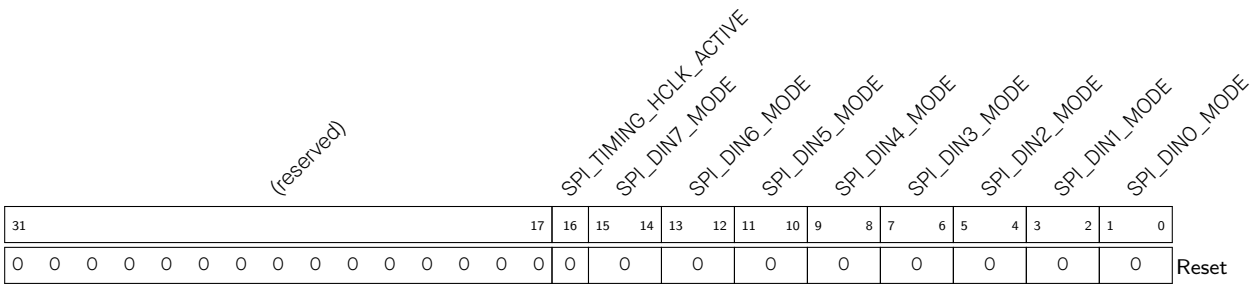
SPI_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能

(R/W)

Register 33.14. SPI_DIN_MODE_REG (0x0024)



SPI_DIN0_MODE 配置 FSPID 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN0_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN1_MODE 配置 FSPIQ 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN1_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN2_MODE 配置 FSPIWP 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN2_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

见下页

Register 33.14. SPI_DIN_MODE_REG (0x0024)

接上页

SPI_DIN3_MODE 配置 FSPIHD 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN3_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN3_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN3_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。

(R/W)

SPI_DIN4_MODE 保留 (HRO)

SPI_DIN5_MODE 保留 (HRO)

SPI_DIN6_MODE 保留 (HRO)

SPI_DIN7_MODE 保留 (HRO)

SPI_TIMING_HCLK_ACTIVE 配置是否使能 SPI 输入信号时序模块的高频时钟 HCLK。

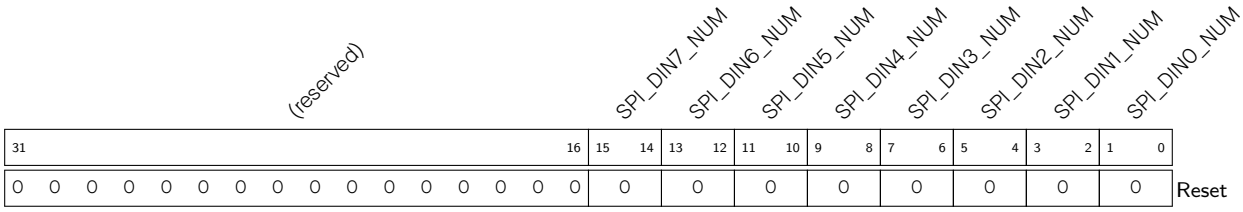
0：不使能

1：使能

可在 CONF 阶段配置。

(R/W)

Register 33.15. SPI_DIN_NUM_REG (0x0028)



SPI_DIN0_NUM 配置输入信号 FSPID 的延迟周期数。具体的延迟模式由 SPI_DIN0_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN1_NUM 配置输入信号 FSPIQ 的延迟周期数。具体的延迟模式由 SPI_DIN1_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN2_NUM 配置输入信号 FSPIWP 的延迟周期数。具体的延迟模式由 SPI_DIN2_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

SPI_DIN3_NUM 配置输入信号 FSPIHD 的延迟周期数。具体的延迟模式由 SPI_DIN3_MODE 配置。

0：延迟 1 个时钟周期
1：延迟 2 个时钟周期
2：延迟 3 个时钟周期
3：延迟 4 个时钟周期
可在 CONF 阶段配置。
(R/W)

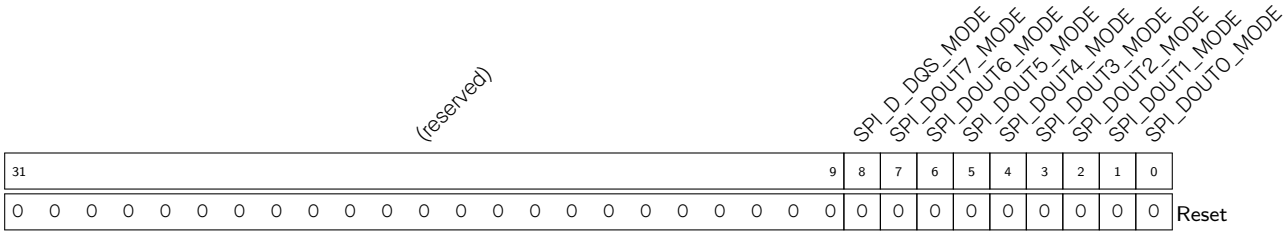
SPI_DIN4_NUM 保留 (HRO)

SPI_DIN5_NUM 保留 (HRO)

SPI_DIN6_NUM 保留 (HRO)

SPI_DIN7_NUM 保留 (HRO)

Register 33.16. SPI_DOUT_MODE_REG (0x002C)



SPI_DOUT0_MODE 配置 FSPID 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT1_MODE 配置 FSPIQ 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT2_MODE 配置 FSPIWP 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT3_MODE 配置 FSPIHD 信号的输出模式。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

可在 CONF 阶段配置。

(R/W)

SPI_DOUT4_MODE 保留 (HRO)

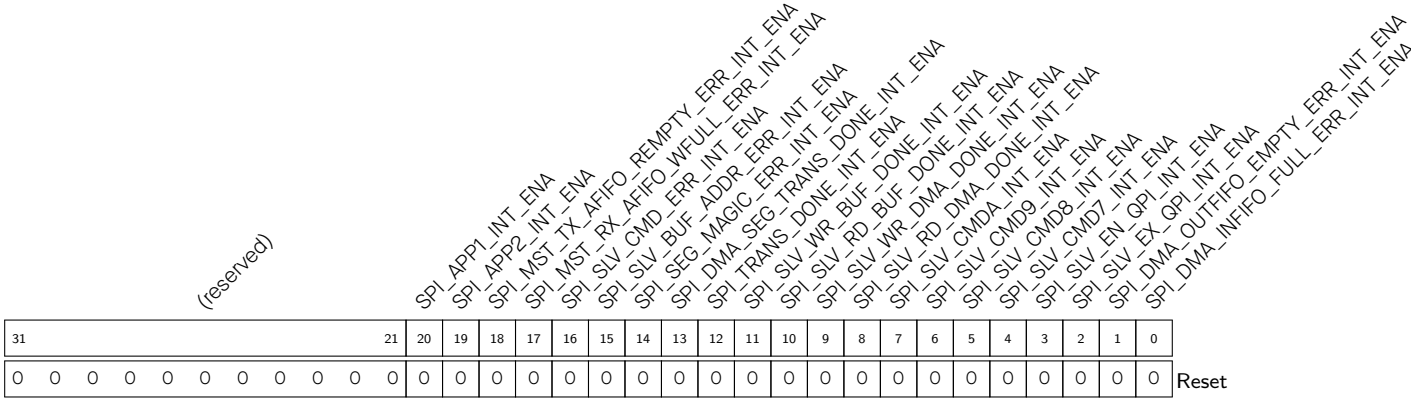
SPI_DOUT5_MODE 保留 (HRO)

SPI_DOUT6_MODE 保留 (HRO)

SPI_DOUT7_MODE 保留 (HRO)

SPI_D_DQS_MODE 保留 (HRO)

Register 33.17. SPI_DMA_INT_ENA_REG (0x0034)



SPI_DMA_INFIFO_FULL_ERR_INT_ENA 写 1 使能 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA 写 1 使能 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(R/W)

SPI_SLV_EX_QPI_INT_ENA 写 1 使能 SPI_SLV_EX_QPI_INT 中断。(R/W)

SPI_SLV_EN_QPI_INT_ENA 写 1 使能 SPI_SLV_EN_QPI_INT 中断。(R/W)

SPI_SLV_CMD7_INT_ENA 写 1 使能 SPI_SLV_CMD7_INT 中断。(R/W)

SPI_SLV_CMD8_INT_ENA 写 1 使能 SPI_SLV_CMD8_INT 中断。(R/W)

SPI_SLV_CMD9_INT_ENA 写 1 使能 SPI_SLV_CMD9_INT 中断。(R/W)

SPI_SLV_CMDA_INT_ENA 写 1 使能 SPI_SLV_CMDA_INT 中断。(R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_DMA_DONE_INT 中断。(R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_DMA_DONE_INT 中断。(R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_BUF_DONE_INT 中断。(R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_BUF_DONE_INT 中断。(R/W)

见下页

Register 33.17. SPI_DMA_INT_ENA_REG (0x0034)

接上页

SPI_TRANS_DONE_INT_ENA 写 1 使能 SPI_TRANS_DONE_INT 中断。
(R/W)

SPI_DMA_SEG_TRANS_DONE_INT_ENA 写 1 使能 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(R/W)

SPI_SEG_MAGIC_ERR_INT_ENA 写 1 使能 SPI_SEG_MAGIC_ERR_INT 中断。
(R/W)

SPI_SLV_BUF_ADDR_ERR_INT_ENA 写 1 使能 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(R/W)

SPI_SLV_CMD_ERR_INT_ENA 写 1 使能 SPI_SLV_CMD_ERR_INT 中断。
(R/W)

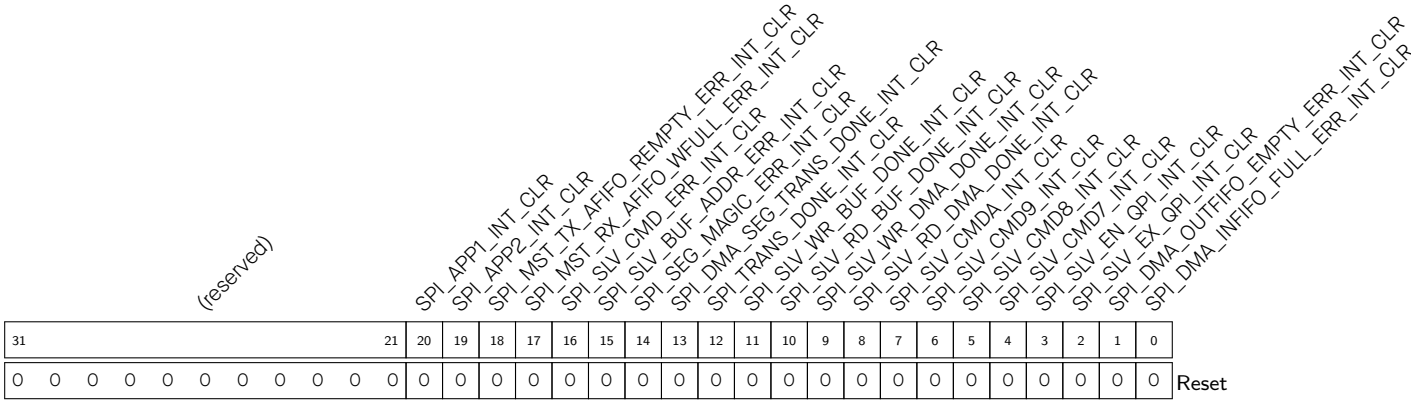
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA 写 1 使能 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA 写 1 使能 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(R/W)

SPI_APP2_INT_ENA 写 1 使能 SPI_APP2_INT 中断。
(R/W)

SPI_APP1_INT_ENA 写 1 使能 SPI_APP1_INT 中断。
(R/W)

Register 33.18. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR 写 1 清除 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR 写 1 清除 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(WT)

SPI_SLV_EX_QPI_INT_CLR 写 1 清除 SPI_SLV_EX_QPI_INT 中断。(WT)

SPI_SLV_EN_QPI_INT_CLR 写 1 清除 SPI_SLV_EN_QPI_INT 中断。(WT)

SPI_SLV_CMD7_INT_CLR 写 1 清除 SPI_SLV_CMD7_INT 中断。(WT)

SPI_SLV_CMD8_INT_CLR 写 1 清除 SPI_SLV_CMD8_INT 中断。(WT)

SPI_SLV_CMD9_INT_CLR 写 1 清除 SPI_SLV_CMD9_INT 中断。(WT)

SPI_SLV_CMDA_INT_CLR 写 1 清除 SPI_SLV_CMDA_INT 中断。(WT)

SPI_SLV_RD_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_DMA_DONE_INT 中断。(WT)

SPI_SLV_WR_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_DMA_DONE_INT 中断。(WT)

SPI_SLV_RD_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_BUF_DONE_INT 中断。(WT)

SPI_SLV_WR_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_BUF_DONE_INT 中断。(WT)

见下页

Register 33.18. SPI_DMA_INT_CLR_REG (0x0038)

接上页

SPI_TRANS_DONE_INT_CLR 写 1 清除 SPI_TRANS_DONE_INT 中断。
(WT)

SPI_DMA_SEG_TRANS_DONE_INT_CLR 写 1 清除 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(WT)

SPI_SEG_MAGIC_ERR_INT_CLR 写 1 清除 SPI_SEG_MAGIC_ERR_INT 中断。
(WT)

SPI_SLV_BUF_ADDR_ERR_INT_CLR 写 1 清除 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(WT)

SPI_SLV_CMD_ERR_INT_CLR 写 1 清除 SPI_SLV_CMD_ERR_INT 中断。
(WT)

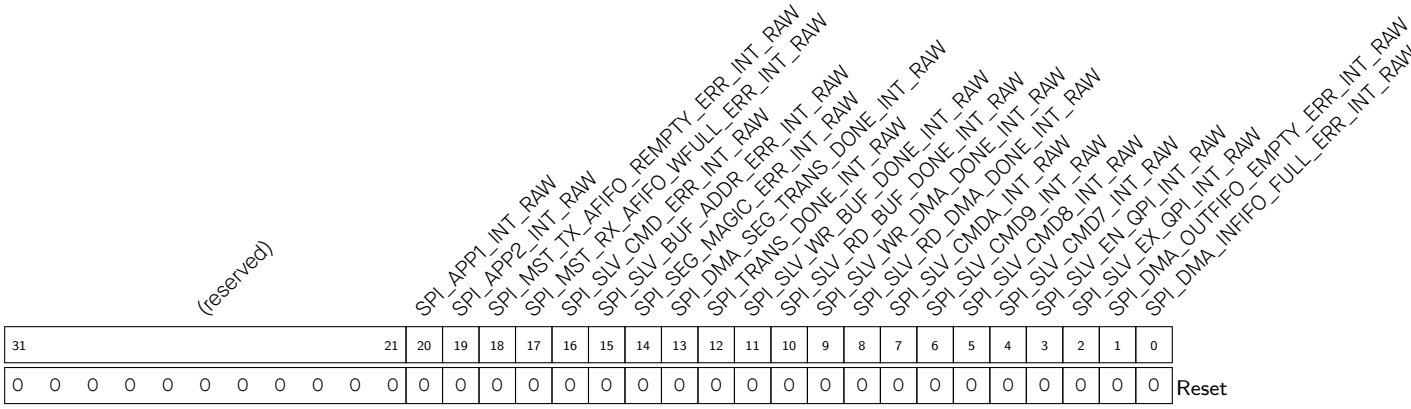
SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR 写 1 清除 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR 写 1 清除 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(WT)

SPI_APP2_INT_CLR 写 1 清除 SPI_APP2_INT 中断。
(WT)

SPI_APP1_INT_CLR 写 1 清除 SPI_APP1_INT 中断。
(WT)

Register 33.19. SPI_DMA_INT_RAW_REG (0x003C)



SPI_DMA_INFIFO_FULL_ERR_INT_RAW SPI_DMA_INFIFO_FULL_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW SPI_SLV_EN_QPI_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD7_INT_RAW SPI_SLV_CMD7_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD8_INT_RAW SPI_SLV_CMD8_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD9_INT_RAW SPI_SLV_CMD9_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMDA_INT_RAW SPI_SLV_CMDA_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW SPI_SLV_RD_DMA_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW SPI_SLV_WR_DMA_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW SPI_SLV_RD_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

见下页

Register 33.19. SPI_DMA_INT_RAW_REG (0x003C)

接上页

SPI_SLV_WR_BUF_DONE_INT_RAW SPI_SLV_WR_BUF_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_TRANS_DONE_INT_RAW SPI_TRANS_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_DMA_SEG_TRANS_DONE_INT_RAW SPI_DMA_SEG_TRANS_DONE_INT 的原始中断状态。
(R/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW SPI_SEG_MAGIC_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_BUF_ADDR_ERR_INT_RAW SPI_SLV_BUF_ADDR_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW SPI_SLV_CMD_ERR_INT 的原始中断状态。
(R/WTC/SS)

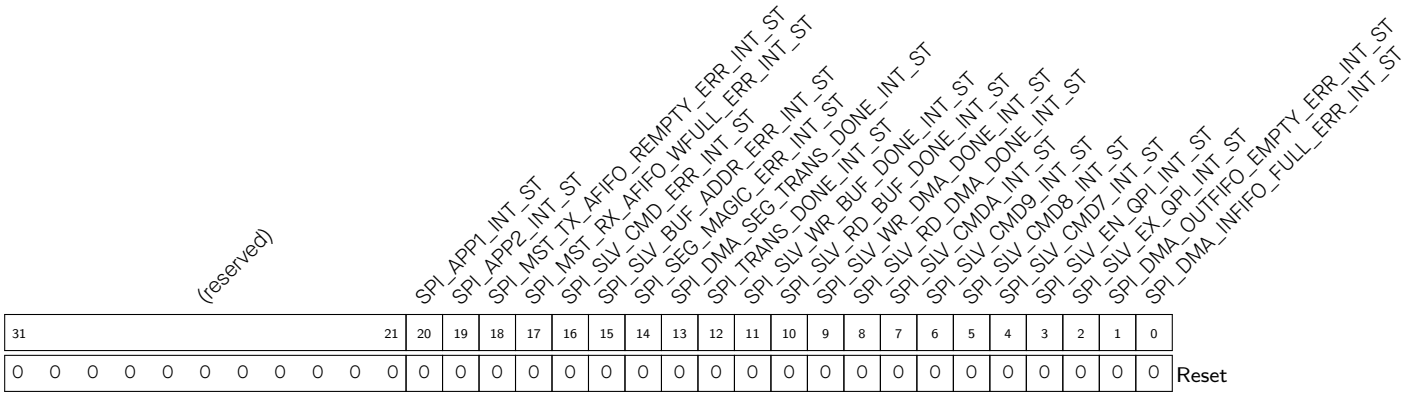
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW SPI_MST_RX_AFIFO_WFULL_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的原始中断状态。
(R/WTC/SS)

SPI_APP2_INT_RAW SPI_APP2_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

SPI_APP1_INT_RAW SPI_APP1_INT 的原始中断状态。该值仅由应用控制。
(R/WTC/SS)

Register 33.20. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST SPI_DMA_INFIFO_FULL_ERR_INT 的中断状态位。
(RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的中断状态位。
(RO)

SPI_SLV_EX_QPI_INT_ST SPI_SLV_EX_QPI_INT 的中断状态位。
(RO)

SPI_SLV_EN_QPI_INT_ST SPI_SLV_EN_QPI_INT 的中断状态位。
(RO)

SPI_SLV_CMD7_INT_ST SPI_SLV_CMD7_INT 的中断状态位。
(RO)

SPI_SLV_CMD8_INT_ST SPI_SLV_CMD8_INT 的中断状态位。
(RO)

SPI_SLV_CMD9_INT_ST SPI_SLV_CMD9_INT 的中断状态位。
(RO)

SPI_SLV_CMDA_INT_ST SPI_SLV_CMDA_INT 的中断状态位。
(RO)

SPI_SLV_RD_DMA_DONE_INT_ST SPI_SLV_RD_DMA_DONE_INT 的中断状态位。
(RO)

SPI_SLV_WR_DMA_DONE_INT_ST SPI_SLV_WR_DMA_DONE_INT 的中断状态位。
(RO)

SPI_SLV_RD_BUF_DONE_INT_ST SPI_SLV_RD_BUF_DONE_INT 的中断状态位。
(RO)

SPI_SLV_WR_BUF_DONE_INT_ST SPI_SLV_WR_BUF_DONE_INT 的中断状态位。
(RO)

见下页

Register 33.20. SPI_DMA_INT_ST_REG (0x0040)

接上页

SPI_TRANS_DONE_INT_ST SPI_TRANS_DONE_INT 的中断状态位。
(RO)

SPI_DMA_SEG_TRANS_DONE_INT_ST SPI_DMA_SEG_TRANS_DONE_INT 的中断状态位。
(RO)

SPI_SEG_MAGIC_ERR_INT_ST SPI_SEG_MAGIC_ERR_INT 的中断状态位。
(RO)

SPI_SLV_BUF_ADDR_ERR_INT_ST SPI_SLV_BUF_ADDR_ERR_INT 的中断状态位。
(RO)

SPI_SLV_CMD_ERR_INT_ST SPI_SLV_CMD_ERR_INT 的中断状态位。
(RO)

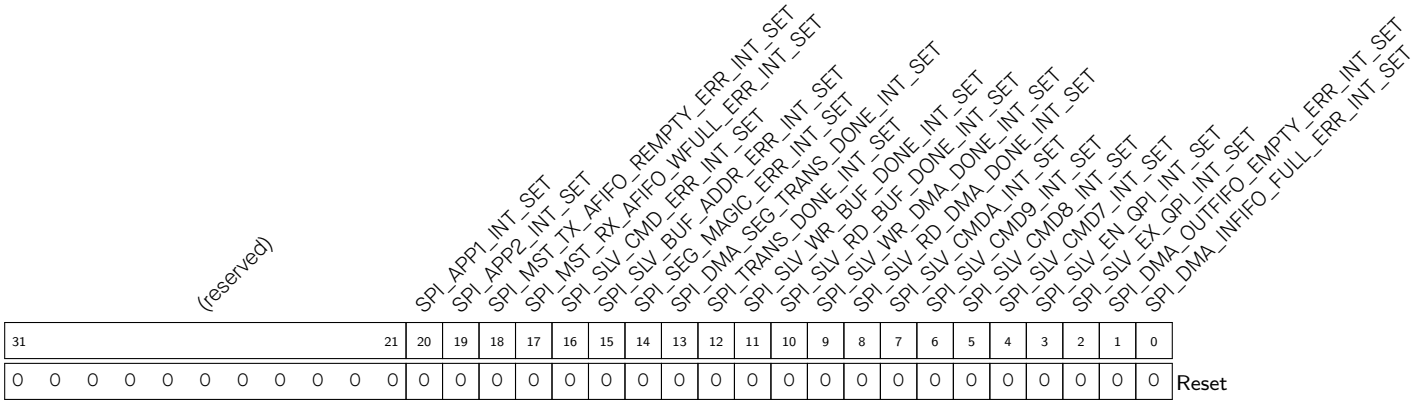
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断状态位。
(RO)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断状态位。
(RO)

SPI_APP2_INT_ST SPI_APP2_INT 的中断状态位。
(RO)

SPI_APP1_INT_ST SPI_APP1_INT 的中断状态位。
(RO)

Register 33.21. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET 写 1 置位 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET 写 1 置位 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(WT)

SPI_SLV_EX_QPI_INT_SET 写 1 置位 SPI_SLV_EX_QPI_INT 中断。(WT)

SPI_SLV_EN_QPI_INT_SET 写 1 置位 SPI_SLV_EN_QPI_INT 中断。(WT)

SPI_SLV_CMD7_INT_SET 写 1 置位 SPI_SLV_CMD7_INT 中断。(WT)

SPI_SLV_CMD8_INT_SET 写 1 置位 SPI_SLV_CMD8_INT 中断。(WT)

SPI_SLV_CMD9_INT_SET 写 1 置位 SPI_SLV_CMD9_INT 中断。(WT)

SPI_SLV_CMDA_INT_SET 写 1 置位 SPI_SLV_CMDA_INT 中断。(WT)

SPI_SLV_RD_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_RD_DMA_DONE_INT 中断。(WT)

SPI_SLV_WR_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_WR_DMA_DONE_INT 中断。(WT)

SPI_SLV_RD_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_RD_BUF_DONE_INT 中断。(WT)

SPI_SLV_WR_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_WR_BUF_DONE_INT 中断。(WT)

SPI_TRANS_DONE_INT_SET 写 1 置位 SPI_TRANS_DONE_INT 中断。(WT)

见下页

Register 33.21. SPI_DMA_INT_SET_REG (0x0044)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_SET 写 1 置位 SPI_DMA_SEG_TRANS_DONE_INT 中断。
(WT)

SPI_SEG_MAGIC_ERR_INT_SET 写 1 置位 SPI_SEG_MAGIC_ERR_INT 中断。
(WT)

SPI_SLV_BUF_ADDR_ERR_INT_SET 写 1 置位 SPI_SLV_BUF_ADDR_ERR_INT 中断。
(WT)

SPI_SLV_CMD_ERR_INT_SET 写 1 置位 SPI_SLV_CMD_ERR_INT 中断。
(WT)

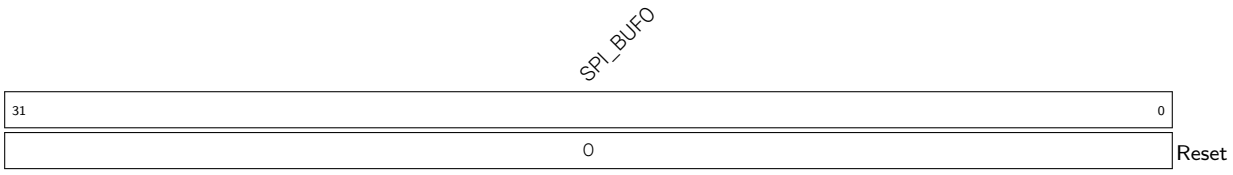
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET 写 1 置位 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。
(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET 写 1 置位 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。
(WT)

SPI_APP2_INT_SET 写 1 置位 SPI_APP2_INT 中断。
(WT)

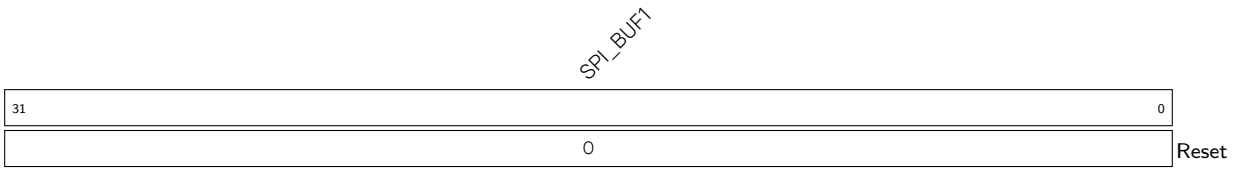
SPI_APP1_INT_SET 写 1 置位 SPI_APP1_INT 中断。
(WT)

Register 33.22. SPI_W0_REG (0x0098)



SPI_BUF0 数据 buffer 0，32 位。
(R/W/SS)

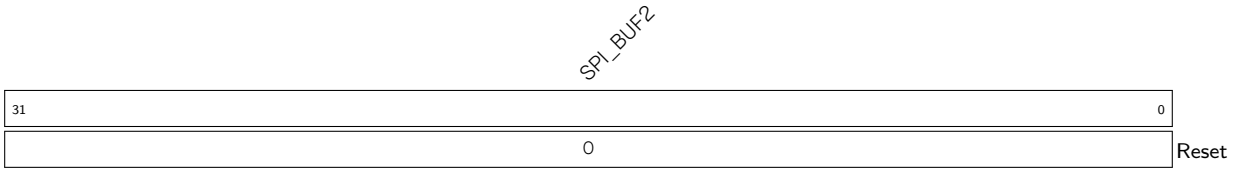
Register 33.23. SPI_W1_REG (0x009C)



SPI_BUF1 数据 buffer 1，32 位。

(R/W/SS)

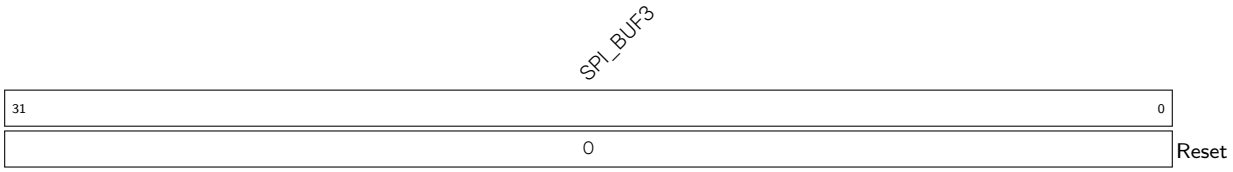
Register 33.24. SPI_W2_REG (0x00A0)



SPI_BUF2 数据 buffer 2，32 位。

(R/W/SS)

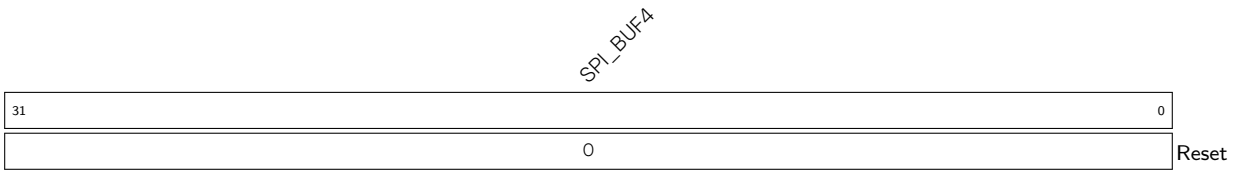
Register 33.25. SPI_W3_REG (0x00A4)



SPI_BUF3 数据 buffer 3，32 位。

(R/W/SS)

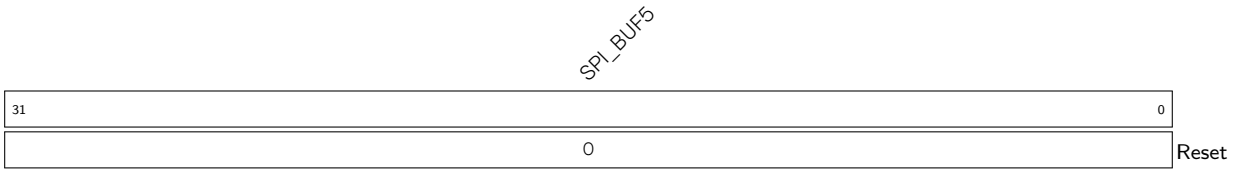
Register 33.26. SPI_W4_REG (0x00A8)



SPI_BUF4 数据 buffer 4，32 位。

(R/W/SS)

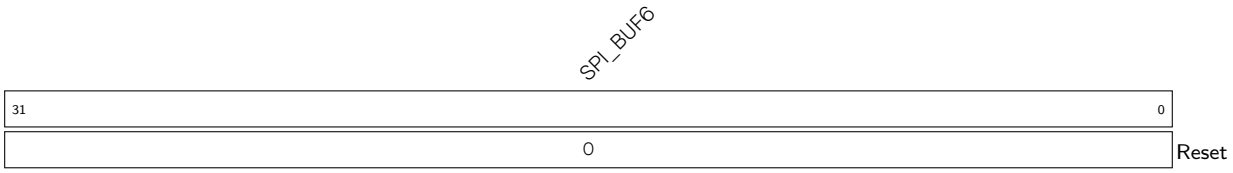
Register 33.27. SPI_W5_REG (0x00AC)



SPI_BUF5 数据 buffer 5，32 位。

(R/W/SS)

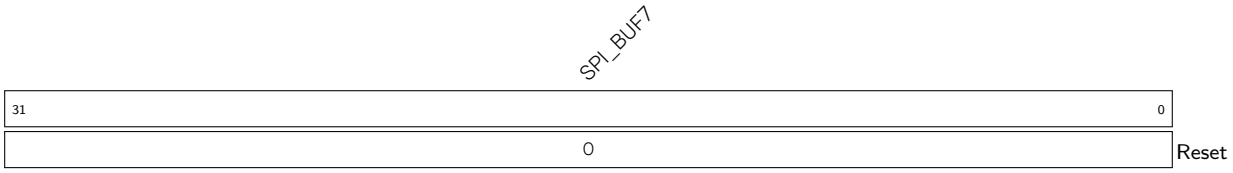
Register 33.28. SPI_W6_REG (0x00B0)



SPI_BUF6 数据 buffer 6，32 位。

(R/W/SS)

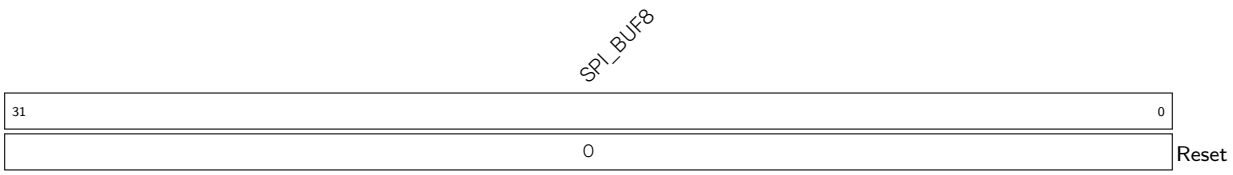
Register 33.29. SPI_W7_REG (0x00B4)



SPI_BUF7 数据 buffer 7，32 位。

(R/W/SS)

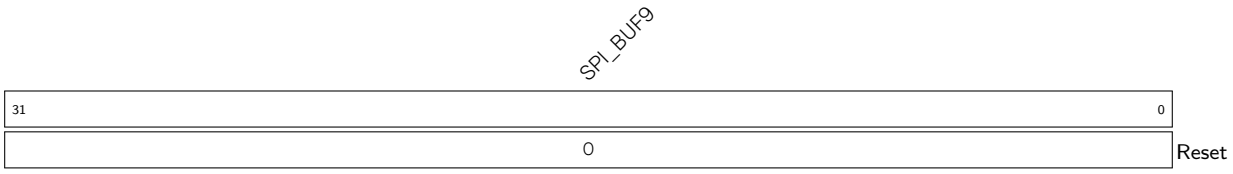
Register 33.30. SPI_W8_REG (0x00B8)



SPI_BUF8 数据 buffer 8，32 位。

(R/W/SS)

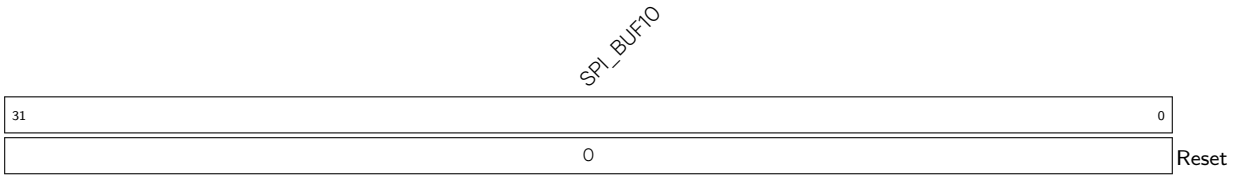
Register 33.31. SPI_W9_REG (0x00BC)



SPI_BUF9 数据 buffer 9，32 位。

(R/W/SS)

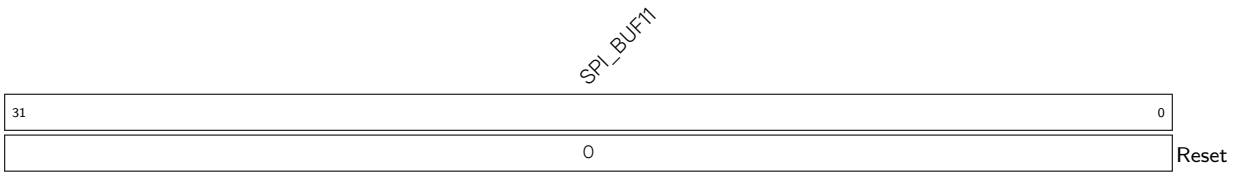
Register 33.32. SPI_W10_REG (0x00C0)



SPI_BUF10 数据 buffer 10，32 位。

(R/W/SS)

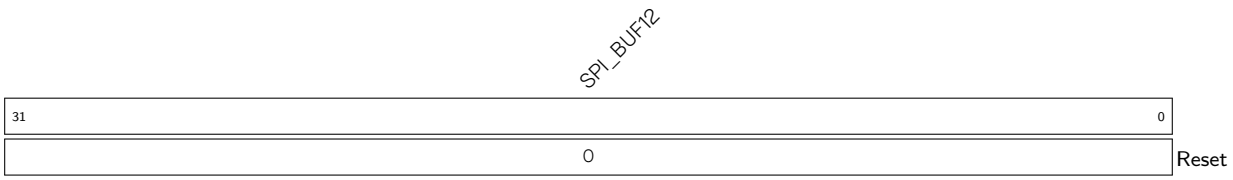
Register 33.33. SPI_W11_REG (0x00C4)



SPI_BUF11 数据 buffer 11，32 位。

(R/W/SS)

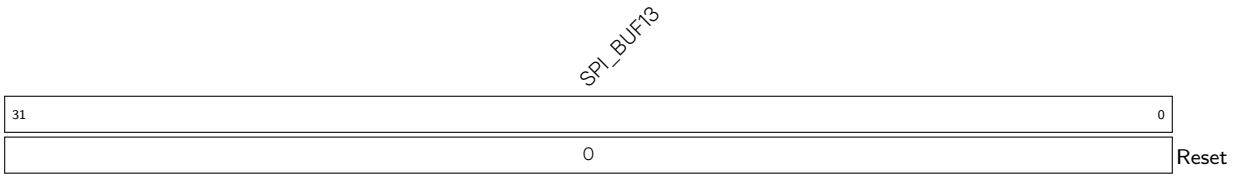
Register 33.34. SPI_W12_REG (0x00C8)



SPI_BUF12 数据 buffer 12，32 位。

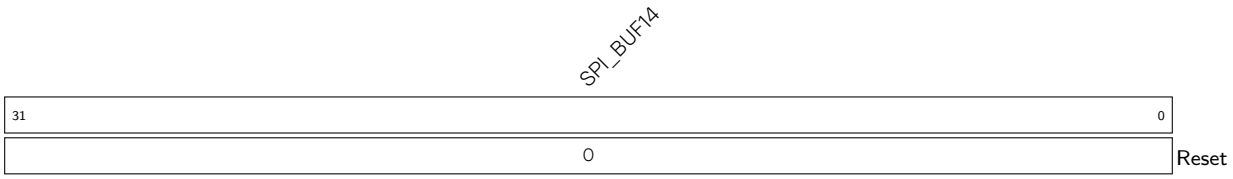
(R/W/SS)

Register 33.35. SPI_W13_REG (0x00CC)



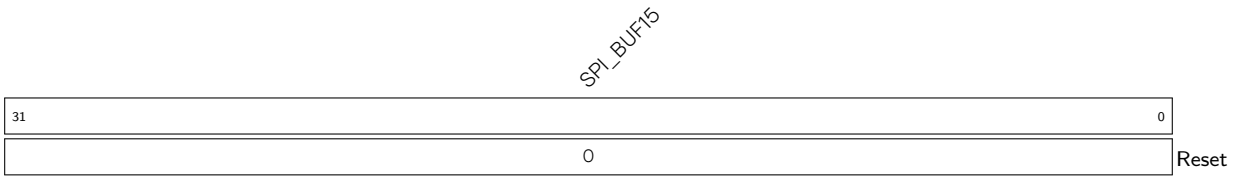
SPI_BUF13 数据 buffer 13，32 位。
(R/W/SS)

Register 33.36. SPI_W14_REG (0x00D0)



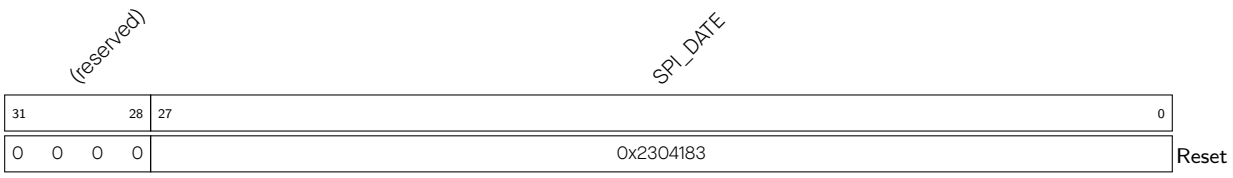
SPI_BUF14 数据 buffer 14，32 位。
(R/W/SS)

Register 33.37. SPI_W15_REG (0x00D4)



SPI_BUF15 数据 buffer 15，32 位。
(R/W/SS)

Register 33.38. SPI_DATE_REG (0x00F0)



SPI_DATE 版本控制寄存器。
(R/W)

第 34 章

I2C 控制器 (I2C)

I2C (Inter-Integrated Circuit) 总线用于使 ESP32-C5 和多个外部设备进行通信。多个外部设备可以共用一个 I2C 总线。

ESP32-C5 在主系统有一个 I2C 控制器，在低功耗系统有一个 I2C 控制器。其中，主系统中的 I2C 控制器既可作为主机又可作为从机（下文以 I2C 指代），低功耗系统中的 I2C 控制器则只可作为主机使用，在主系统休眠时仍能工作（下文以 LP_I2C 指代）。

34.1 概述

I2C 是一个两线总线，由 SDA 线和 SCL 线构成。这些线设置为漏极开漏 (open-drain) 输出。因此，I2C 总线上可以挂载多个外设，通常是和一个或多个主机以及一个或多个从机。但同一时刻只有一个主机能占用总线访问一个从机。

主机发出开始信号，即主机在 SCL 为高电平时拉低 SDA 线，代表通讯开始。随后主机通过 SCL 线发出 9 个时钟脉冲，前 8 个脉冲用于传输 7 位地址和 1 个读 / 写位。如果从机地址与该 7 位地址一致，那么从机可以通过在第 9 个脉冲拉低 SDA 线来应答。接下来，根据读 / 写标志位，主机和从机之间可以传输更多的数据。根据应答位 (ACK) 的逻辑电平决定是否停止发送数据。在数据传输中，SDA 线仅在 SCL 线为低电平时才发生变化。当主机完成通讯，发送一个停止信号：主机在 SCL 为高电平时，拉高 SDA 线。如果一次通信中主机既有写操作又有读操作，则主机需在读写操作变化前，发送一个重新开始信号、从机地址和读 / 写标志位。重新开始信号不仅用于一次通信中切换方向，也用于切换设备模式（主机或从机模式）。

34.2 主要特性

ESP32-C5 I2C 控制器具有以下几个特点：

- 支持主机模式和从机模式
- 支持多主机和从机通信
- 支持标准模式 (100 Kbit/s)
- 支持快速模式 (400 Kbit/s)
- 支持 7 位以及 10 位地址寻址
- 从机模式下支持拉低 SCL 时钟实现连续数据传输
- 支持可编程数字噪声滤波功能
- 支持从机地址和从机内存或寄存器地址的双寻址模式

34.3 结构概览

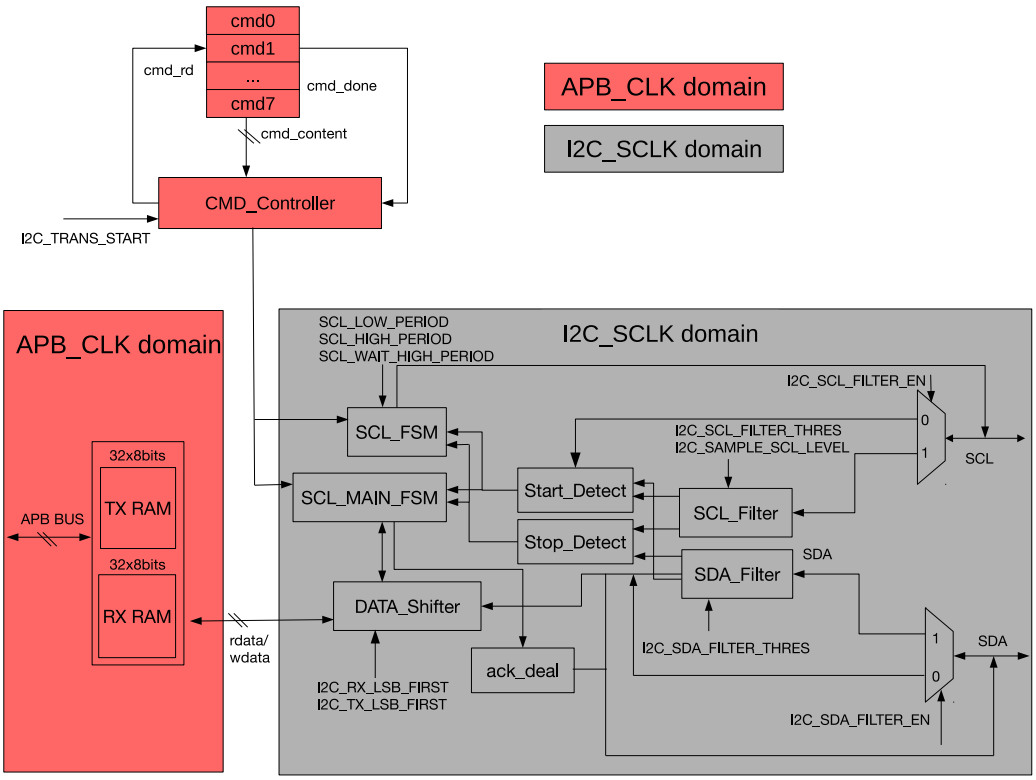


图 34.3-1. I2C 主机基本架构

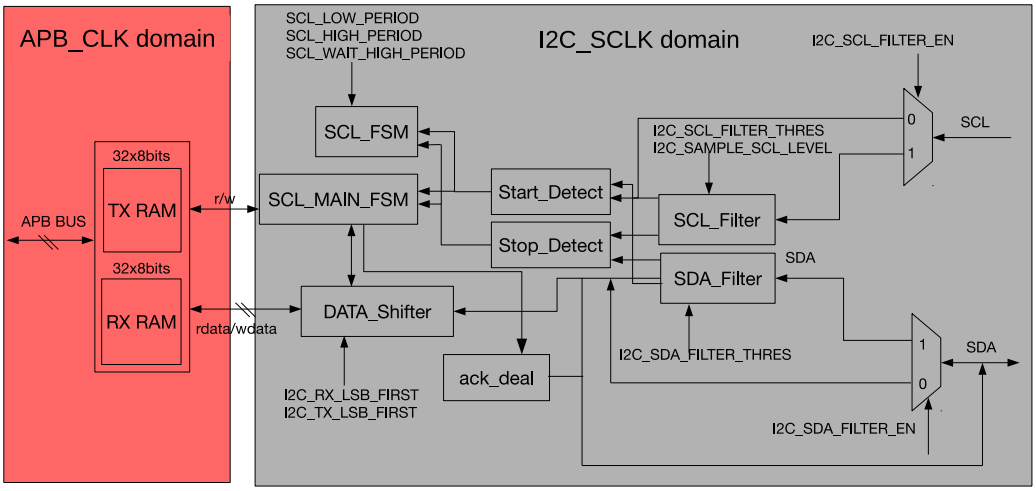


图 34.3-2. I2C 从机基本架构

I2C 控制器可以工作于主机模式或者从机模式，I2C_MS_MODE 寄存器用于模式选择。图 34.3-1 为 I2C 主机基本架构图，图 34.3-2 为 I2C 从机基本架构图。I2C 控制器内部包括的模块主要有：

- 接收和发送存储器 TX/RX RAM：分别用来存储 I2C 要发送和接收到的数据
- 命令控制器 CMD_Controller：产生 RSTART、STOP、WRITE、READ 和 END 指令
- SCL 时钟控制器 SCL_FSM：用来产生满足 I2C 协议的 SCL 时钟。图 34.3-3 和表 34.3-1 是 I2C 协议的时序图和对应的参数表。

- SDA 数据控制器 SCL_MAIN_FSM：用来控制 I2C 指令的执行，和 SDA 线的数据序列，还控制 ack_deal 模块生成 ACK 位并检测 SDA 线上 ACK 位的电平
- 串并转换器 DATA_Shifter：用来完成串行数据和并行数据之间的转换
- SCL 滤波器 SCL_Filter：用于消除 SCL 输入信号上的噪声
- SDA 滤波器 SDA_Filter：用于消除 SDA 输入信号上的噪声
- ACK 位控制器 ack_deal：在 SCL_MAIN_FSM 控制下生成 ACK 位并检测 SDA 线上 ACK 位的电平。

另外，还有产生 I2C 内部时钟的时钟模块，以及在 APB 总线和 I2C 模块之间同步的同步模块。

时钟模块的作用是进行时钟源选择、时钟开关和时钟分频。同步模块用来同步不同时钟域之间信号的传输。

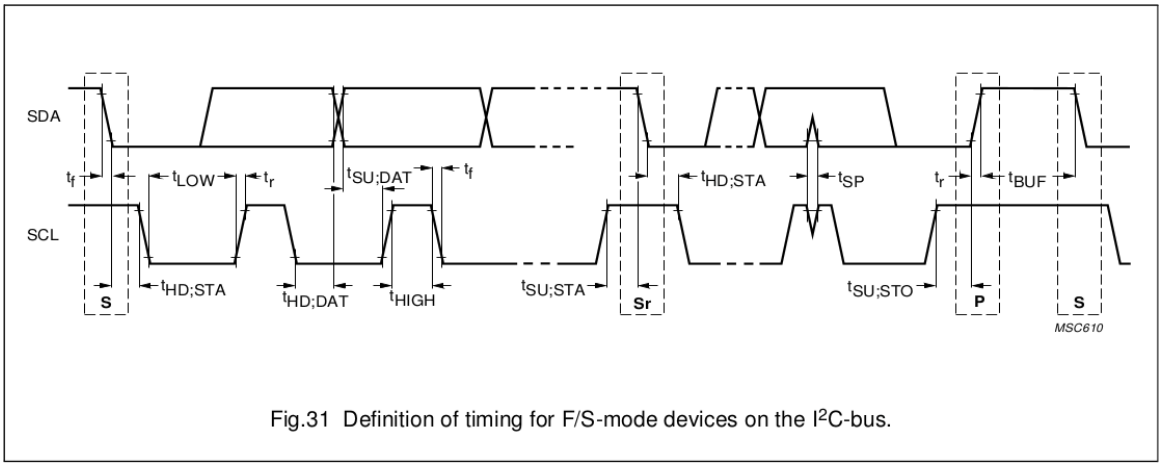


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

图 34.3-3. I2C 协议时序（引自 The I2C-bus specification Version 2.1 Fig.31）

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	20 + 0.1C_b(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	20 + 0.1C_b(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs

表 34.3-1. I2C 时序参数（引自 The I2C-bus specification Version 2.1 Table 5）

34.4 功能描述

正如前文介绍，I2C 总线上可以挂载一个或多个主机以及一个或多个从机。下文中介绍的是 ESP32-C5 I2C 控制器的操作方法，请注意 I2C 总线上其他主机或者从机的操作方法可能与 ESP32-C5 I2C 操作方法不同，具体请参考各个 I2C 控制器的技术规格书。

34.4.1 时钟配置

寄存器配置和 TX/RX RAM 部分的时钟域为 APB_CLK。I2C 主要逻辑部分，包括 SCL_FSM、SCL_MAIN_FSM、SCL_FILTER、SDA_FILTER 和 DATA_SHIFTER 都为 I2C_SCLK 时钟域。

用户可以通过 PCR_I2C_SCLK_SEL，将 HP 系统中 I2C 的时钟源 I2C_SCLK 配置为 XTAL_CLK 或 RC_FAST_CLK。对于 LP 系统中 LP_I2C 的时钟源 I2C_SCLK，则可以通过 LP_CLKRST_LP_I2C_CLK_SEL，将其配置为 CLK_XTALD2 或 CLK_ROOT_FAST。

配置 I2C 时钟源的步骤如下：

- PCR_I2C_SCLK_EN 为 1 时，启用 I2C I2C_SCLK 时钟源。
- PCR_I2C_SCLK_SEL 为 0 时，配置时钟源为 XTAL_CLK。
- PCR_I2C_SCLK_SEL 为 1 时，配置时钟源为 RC_FAST_CLK。

配置 LP_I2C 时钟源的步骤如下：

- LPPERI_LP_EXT_I2C_CK_EN 为 1 时，启用 LP_I2C I2C_SCLK 时钟源。
- LP_CLKRST_LP_I2C_CLK_SEL 为 0 时，配置时钟源为 CLK_ROOT_FAST。
- LP_CLKRST_LP_I2C_CLK_SEL 为 1 时，配置时钟源为 CLK_XTALD2。

选择后的时钟经过小数分频，得到 I2C 的工作时钟 I2C_SCLK：

$$I2C\_SCLK\_DIV\_NUM + 1 + \frac{I2C\_SCLK\_DIV\_A}{I2C\_SCLK\_DIV\_B}$$

式中 I2C_SCLK_DIV_NUM 代表分频系数的整数部分，I2C_SCLK_DIV_A 代表分频系数小数部分的分子，I2C_SCLK_DIV_B 代表分频系数小数部分的分母。根据时序参数的限制，分频后的 I2C_SCLK 的频率需至少大于 SCL 频率 20 倍。
在 I2C 中：

- 配置 PCR_I2C_SCLK_DIV_NUM 调整 I2C_SCLK_DIV_NUM
- 配置 PCR_I2C_SCLK_DIV_A 调整 I2C_SCLK_DIV_A
- 配置 PCR_I2C_SCLK_DIV_B 调整 I2C_SCLK_DIV_B

34.4.2 滤除 SCL 和 SDA 噪声

SCL_Filter 和 SDA_Filter 滤波器模块实现方式相同，用于滤除 SCL 及 SDA 输入信号上的噪声。通过配置 I2C_SCL_FILTER_EN 以及 I2C_SDA_FILTER_EN 寄存器可以开启或关闭滤波器。

以 SCL_Filter 为例，当使能 SCL_Filter 功能时，滤波器会连续采样输入信号 SCL，如果输入信号在连续

I2C_SCL_FILTER_THRES 个 I2C_SCLK 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，SCL_Filter 和 SDA_Filter 滤波器分别会过滤脉冲宽度小于 I2C_SCL_FILTER_THRES 和 I2C_SDA_FILTER_THRES 个 I2C_SCLK 时钟周期的线路毛刺。

34.4.3 SCL 时钟拉伸

从机模式下，可以通过拉低 SCL 线实现时钟拉伸，在此期间数据传输暂停，可以用来给软件足够的时间处理数据。置位 I2C_SLAVE_SCL_STRETCH_EN 位使能 SCL 时钟拉伸功能，配置 I2C_STRETCH_PROTECT_NUM 字段来控制 SCL 拉伸后释放的时长以防出现时序错误。

出现以下四种情况时从机可以选择拉低 SCL 线实现时钟拉伸：

1. 地址命中：从机模式下，从机地址与主机发送到 SDA 线上的地址相匹配，且读写标志位为 1。
2. 写满：从机模式下，I2C 控制器的 RX RAM 为满。注意，从机在接收少于 FIFO 深度个字节时（ESP32-C5 I2C 的 FIFO 深度为 32 字节），可以不开启时钟拉伸功能；当要接收大于等于 FIFO 深度个字节时，可以通过 FIFO 阈值中断写 RAM 的乒乓操作，或者开始时钟拉伸功能，给软件提供处理时间。开启时钟拉伸功能时，必须将 I2C_RX_FULL_ACK_LEVEL 置 0，来保证功能正确，否则可能会出现不可预计的后果。
3. 读空：从机模式下，I2C 控制器要发送数据，但 TX RAM 为空。
4. 发送 ACK 时：从机模式下置位 I2C_SLAVE_BYTE_ACK_CTL_EN，从机会在发送 ACK 时拉低 SCL。软件在此阶段进行一些操作，如数据校验，并通过配置 I2C_SLAVE_BYTE_ACK_LVL 控制将要发送的 ACK 的电平高低。要注意的是，当出现从机接收的 RX RAM 满时，要发送的 ACK 电平将由 I2C_RX_FULL_ACK_LEVEL 而不是 I2C_SLAVE_BYTE_ACK_LVL 决定。此时同样需要将 I2C_RX_FULL_ACK_LEVEL 置 0，以保证 SCL 时钟拉伸功能的正常产生。

产生时钟拉伸时，软件可读取 I2C_STRETCH_CAUSE 位获取 SCL 时钟拉伸的原因。置位 I2C_SLAVE_SCL_STRETCH_CLR 位关闭 SCL 时钟拉伸。

34.4.4 SCL 空闲时产生 SCL 脉冲

通常情况下，在 I2C 总线空闲时，SCL 线一直为高。ESP32-C5 I2C 支持在 I2C 主机处于空闲状态时，可编程配置产生 SCL 脉冲的功能。这个功能仅在 I2C 控制器作为主机时有效。置位 I2C_SCL_RST_SLV_EN，硬件会发送 I2C_SCL_RST_SLV_NUM 个 SCL 脉冲，之后 I2C_SCL_RST_SLV_EN 位会自动清零。

34.4.5 同步

I2C 的寄存器配置用 APB 时钟，I2C 主模块用 I2C_SCLK，这之间存在异步处理，需要增加同步的步骤将配置寄存器的值更新进入 I2C 主模块。步骤为先写配置寄存器，再向 I2C_CONF_UPGATE 位写 1。需要通过这种方法更新的配置寄存器详见表 34.4-1。

表 34.4-1. I2C 同步寄存器

配置寄存器	寄存器域	地址
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	I2C_ADDR_10BIT_RW_CHECK_EN	
	I2C_ADDR_BROADCASTING_EN	
	I2C_SDA_FORCE_OUT	
	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	
I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SLAVE_ADDR_REG	I2C_ADDR_10BIT_EN	0x0010
	I2C_SLAVE_ADDR	
I2C_FIFO_CONF_REG	I2C_FIFO_ADDR_CFG_EN	0x0018
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_STRETCH_CONF_REG	I2C_SLAVE_BYTE_ACK_CTL_EN	0x0084
	I2C_SLAVE_BYTE_ACK_LVL	
	I2C_SLAVE_SCL_STRETCH_EN	
	I2C_STRETCH_PROTECT_NUM	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038
	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	
	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

34.4.6 漏级开路输出

SCL 及 SDA 线采用漏级开路的驱动方式。I2C 控制器有两种配置方式实现漏级开路驱动方式：

1. 置位 I2C_SCL_FORCE_OUT、I2C_SDA_FORCE_OUT 并配置相应 SCL 及 SDA PAD 的 GPIO_PIN*n*_PAD_DRIVER 寄存器为漏级开路驱动。
2. 清零 I2C_SCL_FORCE_OUT 以及 I2C_SDA_FORCE_OUT。

SCL 和 SDA 配置成开漏方式时，从低电平转向高电平的时间会较长，这个转变时间由线上的上拉电阻以及电容共同决定。开漏模式下，I2C 输出频率的占空比受限于 SCL 上拉速度，主要受 SCL 的速度限制。

另外，在 I2C_SCL_FORCE_OUT 和 I2C_SCL_PD_EN 置 1 时，可以强制拉低 SCL 线；在 I2C_SDA_FORCE_OUT 和 I2C_SDA_PD_EN 置 1 时，可以强制拉低 SDA 线。

34.4.7 时序参数配置

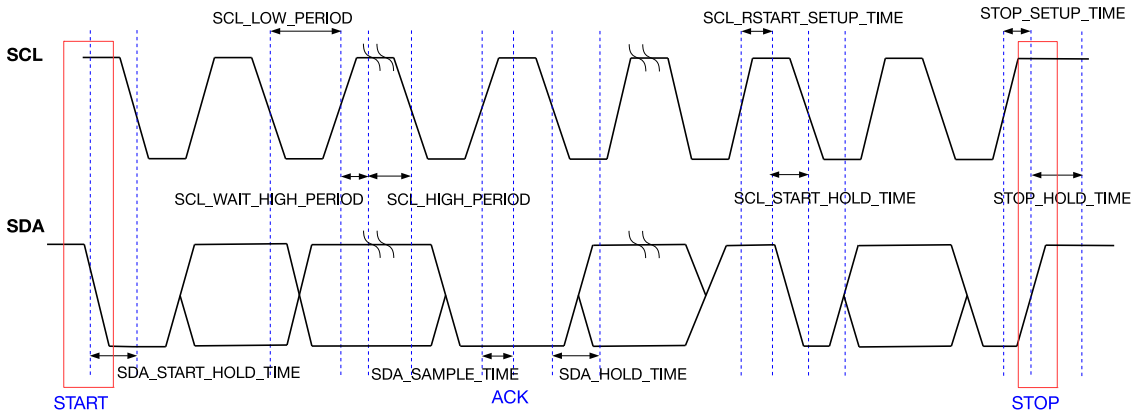


图 34.4-1. I2C 时序图

图 34.4-1 为实现 I2C 协议的 I2C 主机的时序图，图中的寄存器均用来配置时序参数。I2C 控制器的 START 位、STOP 位、数据保持时间、数据采样时间、SCL 上升沿等待时间等时序均可以通过图 34.4-1 中所示的寄存器进行配置。这些寄存器以模块时钟 (I2C_SCLK) 为单位，与各时序参数的对应关系为：

1. $t_{LOW} = (I2C\_SCL\_LOW\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
2. $t_{HIGH} = (I2C\_SCL\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
3. $t_{SU:STA} = (I2C\_SCL\_RSTART\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
4. $t_{HD:STA} = (I2C\_SCL\_START\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
5. $t_{r} = (I2C\_SCL\_WAIT\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
6. $t_{SU:STO} = (I2C\_SCL\_STOP\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
7. $t_{BUF} = (I2C\_SCL\_STOP\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
8. $t_{HD:DAT} = (I2C\_SDA\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
9. $t_{SU:DAT} = (I2C\_SCL\_LOW\_PERIOD - I2C\_SDA\_HOLD\_TIME) \cdot T_{I2C\_SCLK}$

根据在何种模式下有意义，下列时序寄存器可分为两组：

- 主机模式：

1. I2C_SCL_START_HOLD_TIME：生成 I2C 协议中的 START 位时，SDA 信号拉低到 SCL 信号拉低的时间间隔。该时间间隔为（I2C_SCL_START_HOLD_TIME +1）个模块时钟周期。仅控制器工作在主机模式时有意义。
2. I2C_SCL_LOW_PERIOD：SCL 低电平持续时间。SCL 低电平时间为（I2C_SCL_LOW_PERIOD + 1）个模块时钟周期。该寄存器仅在控制器工作在主机模式时有意义。

   SCL 低电平时间会在以下情况下延长：

   - 当 I2C 为主机时，外设拉低 SCL
   - I2C 控制器执行 END 命令拉低 SCL
   - I2C 为从机时，发生 SCL 时钟拉伸
3. I2C_SCL_WAIT_HIGH_PERIOD：等待 SCL 线拉高的模块时钟周期数。请确保在该时间内 SCL 线可以完成上拉。否则会导致 SCL 高电平持续时间不可预测。仅控制器工作在主机模式时有意义。
4. I2C_SCL_HIGH_PERIOD：SCL 线拉高后维持高电平的模块时钟周期数。仅控制器工作在主机模式时有意义。当 SCL 线在 I2C_SCL_WAIT_HIGH_PERIOD + 1 个模块时钟内完成上拉，则 SCL 线的频率为：$f_{scl} = \frac{f_{I2C\_SCLK}}{I2C\_SCL\_LOW\_PERIOD + I2C\_SCL\_HIGH\_PERIOD + I2C\_SCL\_WAIT\_HIGH\_PERIOD + 3 + I2C\_SCL\_FILTER\_THRES}$

   其中 3 个时钟周期是对 SCL 进行同步所需的时间，如果打开了 SCL 滤波功能，则还需要加上滤波带来的延迟 I2C_SCL_FILTER_THRES。用 I2C_SCL_WAIT_HIGH_PERIOD + 1 个模块时钟模拟的 SCL 上拉时间受到上拉电阻、IO 驱动能力、SCL 线上电容等等的影响，可能出现测试的实际频率和理论频率有偏差的情况。此时可以通过调整 I2C_SCL_WAIT_HIGH_PERIOD 的值来使实际频率和理论频率接近。

- 主机模式和从机模式：
  1. I2C_SDA_SAMPLE_TIME：SCL 上升沿到采样 SDA 线电平值的时间间隔。推荐设置在 SCL 高电平持续时间的中间值，以保证能够正确采样到 SDA 线上电平。控制器工作在主机模式及从机模式时都有意义。
  2. I2C_SDA_HOLD_TIME：SDA 输出数据变化与 SCL 下降沿的时间间隔。控制器工作在主机模式及从机模式时都有意义。

根据时序参数的限制，对时序寄存器的配置范围也有约束。

1. $\frac{f_{I2C\_SCLK}}{f_{SCL}} > 20$
2. $3 \times f_{I2C\_SCLK} \leq (I2C\_SDA\_HOLD\_TIME - 4) \times f_{APB\_CLK}$
3. I2C_SDA_HOLD_TIME + I2C_SCL_START_HOLD_TIME > SDA_FILTER_THRES + 3
4. I2C_SCL_WAIT_HIGH_PERIOD < I2C_SDA_SAMPLE_TIME < I2C_SCL_HIGH_PERIOD
5. I2C_SDA_SAMPLE_TIME < I2C_SCL_WAIT_HIGH_PERIOD + I2C_SCL_START_HOLD_TIME +I2C_SCL_RSTART_SETUP_TIME
6. I2C_STRETCH_PROTECT_NUM + I2C_SDA_HOLD_TIME > I2C_SCL_LOW_PERIOD

34.4.8 超时控制

I2C 内部有三种超时控制，分别是对 SCL_FSM 状态的超时控制、SCL_MAIN_FSM 状态的超时控制和对 SCL 线状态的超时控制。其中前两种是一直使能的，第三种是可编程配置的。

当 SCL_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C\_SCL\_ST\_TO\_I2C+1}-1$ 个时钟周期后，会触发 I2C_SCL_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_ST_TO_I2C 的配置值最大为 23，即最迟在等待超过 $2^{24}-1$ 个 I2C_SCLK 时钟周期后会产生超时中断。

当 SCL_MAIN_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C+1}-1$ 个 I2C_SCLK 时钟周期后，会触发 I2C_SCL_MAIN_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_MAIN_ST_TO_I2C 的配置值最大为 23，即最迟在等待超过 $2^{24}-1$ 个时钟周期后会产生超时中断。

使能 I2C_TIME_OUT_EN 打开 SCL 线状态的超时控制。当 SCL 线状态长时间维持同一电平不变，且时间超过 $2^{I2C\_TIME\_OUT\_VALUE+1}-1$ 个时钟周期后，会触发 I2C_TIME_OUT_INT 中断，I2C 总线回到空闲状态。

34.4.9 指令配置

I2C 控制器工作于主机模式时，CMD_Controller 会依次从八个命令寄存器中读出命令并按照命令来控制 SCL_FSM 及 SCL_MAIN_FSM。

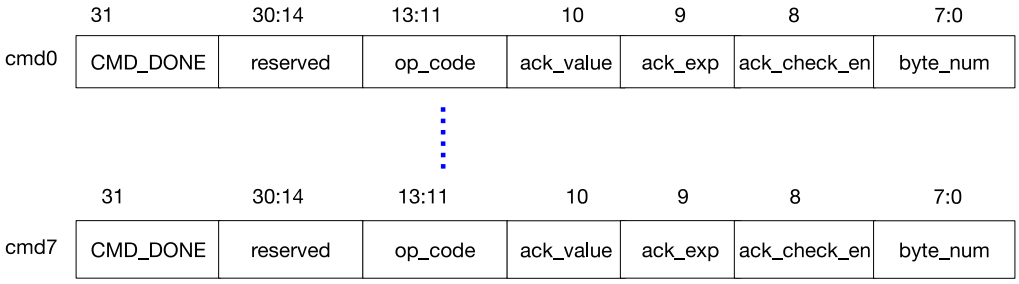


图 34.4-2. I2C 命令寄存器结构

命令寄存器只在 I2C 控制器工作于主机模式时才有效，其内部结构如图 34.4-2 所示。命令寄存器的参数为:

1. CMD_DONE: 命令执行完成标识。每条命令执行完硬件会将对应命令寄存器中的 CMD_DONE 置 1。软件可以通过读取每条命令的 CMD_DONE 位来判断该命令是否执行完毕。每次更新命令时，软件需要将 CMD_DONE 位清零。
2. op_code: 命令编码，共有五种命令:
   - WRITE: op_code 等于 1 时为 WRITE 命令，该命令指示 I2C 控制器向从机发送从机地址、被访问的寄存器地址（仅限双寻址模式）、数据。
   - STOP: op_code 等于 2 时为 STOP 命令，该命令指示 I2C 控制器发送 I2C 协议中的 STOP 位。此条命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件再次启动 CMD_Controller 后，会重新从命令寄存器 0 开始去取指令。
   - READ: op_code 等于 3 时为 READ 命令，该命令指示 I2C 控制器从从机读取数据。
   - END: op_code 等于 4 时为 END 命令，该命令指示 I2C 控制器将 SCL 信号拉低，暂停 I2C 通信。该命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件在更新命令寄存器和 RAM 数据后可重新启动 CMD_Controller，继续进行 I2C 协议传输。再次启动后 CMD_Controller 会重新从命令寄存器 0 开始去取指令。
   - RSTART: op_code 等于 6 时为 RSTART 命令，该命令指示 I2C 控制器发送 I2C 协议中的 START 位或 RESTART 位。

3. ack_value: 该位设置读操作时 I2C 控制器在 I2C 协议中的 ACK 位发送的电平值。RSTART、STOP、END、WRITE 命令中该位没有意义。
4. ack_exp: 该位用于设置写操作时 I2C 控制器在 I2C 协议中的 ACK 位期望接收的电平值。RSTART、STOP、END、READ 命令中该位没有意义。
5. ack_check_en: 该位使能写操作中 I2C 控制器检查从机发送的 ACK 位电平与命令中的 ack_exp 是否一致。如果接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致时，I2C 主机会产生 I2C_NACK_INT 中断，停止发送数据并产生 STOP。此位为 1 时，检测从机发送的 ACK 位电平；此位为 0 时，不检测从机发送的 ACK 位电平。RSTART、STOP、END、READ 命令中该位没有意义。
6. byte_num: 读写数据的长度（单位字节），最大为 255，最小为 1。RSTART、STOP、END 命令中 byte_num 无意义。

每次命令序列的执行都是从命令寄存器 0 开始，到 STOP 或 END 命令结束。所以需要保证每个命令序列中必须有 STOP 或 END 命令。

一次完整的 I2C 协议传输应该起始于 START 命令，结束于 STOP 命令。可通过 END 命令将一次 I2C 协议传输分为多个命令序列来完成。每个命令序列可以改变数据传输的方向、时钟频率、从机地址和数据长度等。这样可以弥补 RAM 大小不足的问题，也可以实现更灵活的 I2C 通信。

34.4.10 TX/RX RAM 数据存储

TX/RX RAM 大小均为 32 x 8 位。TX RAM 和 RX RAM 均可以通过 FIFO 和直接地址 (non-FIFO) 两种方式访问。将 I2C_NONFIFO_EN 位设置成 0，为 FIFO 方式；I2C_NONFIFO_EN 位设置成 1 时为直接地址方式。

TX RAM 用于存储 I2C 控制器需要发送的数据。在 I2C 通信的过程中，当 I2C 控制器需要发送数据时（不包括 ACK 位响应），会依次读出 TX RAM 中的数据并串行输出到 SDA 线上。当 I2C 控制器工作于主机模式时，所有需要发送给从机的数据都必须按照发送顺序依次存储在 TX RAM 中。包括被访问的从机地址、读写标志位、被访问的寄存器地址（仅限双地址寻址模式下）、写数据。当 I2C 控制器工作于从机模式时，TX RAM 中只存放写数据。

TX RAM 可被 CPU 读写。CPU 可通过两种方式写 TX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 I2C_DATA_REG 写 TX RAM，硬件自动进行 TX RAM 写地址自增。直接地址访问是通过地址段 (I2C 基地址 + 0x100) ~(I2C 基地址 + 0x17C) 直接访问 TX RAM。TX RAM 的每一个字节占据一个字 (word) 的地址。因此，第一个字节访问地址为 I2C 基地址 + 0x100，第二字节访问地址为 I2C 基地址 + 0x104，第三字节访问地址为 I2C 基地址 + 0x108，以此类推。CPU 只可通过直接地址访问方式读 TX RAM，且 CPU 可以通过直接地址读回写入到 TX RAM 的字节。读 TX RAM 的地址和写 TX RAM 的地址相同。

RX RAM 存储的是 I2C 通信过程中，I2C 控制器接收到的数据。当 I2C 控制器工作于从机模式时，主机发送的从机地址及被访问的寄存器地址（仅限双地址寻址模式下）都不会存储在 RX RAM 中。软件可以在 I2C 通信结束后，读出 RX RAM 的值。

RX RAM 只可被 CPU 读。CPU 可通过两种方式读 RX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 I2C_DATA_REG 读 RX RAM，硬件自动完成 RX RAM 读地址自增。直接地址访问是通过地址段 (I2C 基地址 + 0x180) ~(I2C 基地址 + 0x1FC) 直接访问 RX RAM。RX RAM 的每一个字节占据一个字的地址。因此，第一个字节访问地址为 I2C 基地址 + 0x180，第二字节访问地址为 I2C 基地址 + 0x184，第三字节访问地址为 I2C 基地址 + 0x188，以此类推。

在 FIFO 模式下可以对 TX RAM 进行乒乓操作，来实现发送大于 FIFO 深度的数据。置位 I2C_FIFO_PRT_EN，当 RAM 中剩下的待发送数据字节数小于 I2C_TXFIFO_WM_THRHD 时，会产生 I2C_TXFIFO_WM_INT 中断。软件收

到中断后，继续向 I2C_DATA_REG（主机）中写数。需要保证向 TX RAM 写数或更新数据的操作提前于 I2C 发送数据的动作，否则会产生不可预计的后果。

在 FIFO 模式下也可以对 RX RAM 进行乒乓操作，来实现接收大于 FIFO 深度的数据。置位 I2C_FIFO_PRT_EN，将 I2C_RX_FULL_ACK_LEVEL 置 0。当 RAM 中收到的数据字节数大于等于 I2C_RXFIFO_WM_THRHD（从机）时，会产生 I2C_RXFIFO_WM_INT 中断。软件收到中断后，从 I2C_DATA_REG（从机）中读数。

34.4.11 数据转换

DATA_Shifter 模块用于串并转换，当 I2C 发送数据时，将 TX RAM 中的字节数据转化成比特流；当 I2C 接收数据时，将比特流转化成字节数据并存入 RX RAM。I2C_RX_LSB_FIRST 和 I2C_TX_LSB_FIRST 用于配置最高有效位或最低有效位的优先储存或传输。

34.4.12 寻址模式

ESP32-C5 I2C 支持 7 位寻址模式和 10 位寻址模式，7 位寻址和 10 位寻址可结合使用。除此之外，ESP32-C5 I2C 还支持双地址寻址模式。

假设从机地址为 SLV_ADDR。ESP32-C5 I2C 控制器可以使用 7 位寻址（SLV_ADDR[6:0]），也可以使用 10 位寻址（SLV_ADDR[9:0]）。

对于主机模式而言，7 位寻址下只要发送一个字节地址段 SLV_ADDR[6:0] 和读写标志。7 位寻址模式下，有种特殊情况是广播寻址。在从机中，将 I2C_ADDR_BROADCASTING_EN 置 1，开启广播寻址模式。当接收到主机发送的地址为广播地址 (0x00) 且读写标志位为 0 时，此时无论从机自己的地址是多少，都会响应主机。

10 位寻址需要发送两字节地址段。第一个要发送的数是从机地址的第一个 7 位 slave_addr_first_7bits 和读写标志，slave_addr_first_7bits 的值应该配置为 (0x78 | SLV_ADDR[9:8])。第二个要发送的数是 slave_addr_second_byte，slave_addr_second_byte 的值为 SLV_ADDR[7:0]。

在从机中，可以通过配置 I2C_ADDR_10BIT_EN 寄存器开启 10 位寻址模式。I2C_SLAVE_ADDR 用于配置 I2C 从机地址。I2C_SLAVE_ADDR[14:7] 的值应配置为 SLV_ADDR[7:0]，I2C_SLAVE_ADDR[6:0] 的值应配置为 (0x78 | SLV_ADDR[9:8])。由于 10 位从机地址比 7 位地址多一个字节，所以 WRITE 命令对应的 byte_num 以及 RAM 中数据数量都相应增加 1。详细说明可参考 编程示例。

控制器处于从机模式时，还支持双地址寻址模式。双地址的第一个地址是 I2C 从机地址，第二个地址是 I2C 从机的内存地址。双地址模式下，RAM 必须采用直接地址 (non-FIFO) 的方式访问。

通过置位 I2C_FIFO_ADDR_CFG_EN 来使能双地址访问功能。当从机接收到的从机地址和内部配置的从机地址不一致时，会产生 I2C_SLAVE_ADDR_UNMATCH 中断。

34.4.13 10 位寻址的读写标志位检查

在 10 位寻址模式下，将 I2C_ADDR_10BIT_RW_CHECK_EN 置 1，会对发送的第一个数 slave_addr_first_7bits 和读写标志做检查。当读写标志不是写，此时与协议不符合，会结束传输。若不打开此功能，当读写标志不是写，还能支持继续传输，但可能出现传输错误。

34.4.14 启动控制器

对于主机，配置成主机模式和命令寄存器等相关配置后，通过向 I2C_TRANS_START 写 1，启动主机解析，执行命令序列。一组命令总是从命令寄存器 0 开始，顺序执行到 STOP 或者 END 命令。当另一组命令需要从命令寄存器 0 开始执行时，需要重新向 I2C_TRANS_START 写 1 来更新命令。

对于从机，配置成从机模式并配置其他相关寄存器之后，向 I2C_TRANS_START 写 1。从机将会在与主机所发送的地址匹配时，响应主机。

对于从机，请注意：

- 每次接收传输前必须置位 I2C_TRANS_START，否则即使地址匹配，从机也不会响应主机。
- 如果主机发起 STOP 提前中止了和从机的传输，那么在下一次传输前，从机必须先置位 I2C_SLV_TX_AUTO_START_EN，再置位 I2C_TRANS_START，最后清零 I2C_SLV_TX_AUTO_START_EN，否则从机的 TX FIFO 将会出现指针异常。

34.5 LP_I2C 和 I2C 的功能差异

LP_I2C 可以作为主机在主系统休眠时和外部设备通信，它包含所有 ESP32-C5 I2C 主机部分的功能，但不包含任何 ESP32-C5 I2C 从机部分的功能。因此，LP_I2C 不包含任何与从机功能相关的寄存器，详细的寄存器列表见 34.8.2 LP_I2C 寄存器列表。

LP_I2C 和 I2C 主机在设计上的区别有下列几点：

- LP_I2C 中 TX/RX RAM 为 16x8 位大小，即 TX/RX FIFO 深度变为 16 字节。
- LP_I2C 的 APB_CLK 的时钟源为 CLK_AON_FAST。通过配置 LP_CLKRST_LP_I2C_CLK_SEL 选择 I2C_SCLK 的时钟源。具体时钟源配置为：
  - 0：CLK_ROOT_FAST
  - 1：CLK_XTALD2

  配置 LPPERI_LP_EXT_I2C_CK_EN 为高电平来使能 I2C_SCLK 的时钟源。时钟频率发生变化时要注意对时序寄存器做出相应调整。

LP_I2C 的编程示例参见 34.7 中的主机部分。

34.6 中断

ESP32-C5 的 I2C*n* 可以生成 I2C*n*_INTR 中断信号并发送给中断矩阵。这一中断信号由以下 I2C*n* 的内部中断源生成：

- I2C_SLAVE_STRETCH_INT：从机模式下，当从机时钟拉伸时，产生此中断。
- I2C_DET_START_INT：主机或从机检测到 I2C START 信号，触发此中断。
- I2C_SCL_MAIN_ST_TO_INT：当 I2C 主状态机 SCL_MAIN_FSM 保持某个状态超过 $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C+1}$ 个模块时钟周期时，触发此中断。
- I2C_SCL_ST_TO_INT：当 I2C 状态机 SCL_FSM 保持某个状态超过 $2^{I2C\_SCL\_ST\_TO\_I2C+1}$ 个模块时钟周期时，触发此中断。
- I2C_RXFIFO_UDF_INT：当 I2C 通过 APB 总线读取 RX FIFO，但 RX FIFO 为空时，触发该中断。
- I2C_TXFIFO_OVF_INT：当 I2C 通过 APB 总线写 TX FIFO，但 TX FIFO 为满时，触发该中断。
- I2C_NACK_INT: 当 I2C 配置为主机时，接收到的 ACK 与命令中期望的 ACK 值不一致时，即触发该中断；当 I2C 配置为从机时，接收到的 ACK 值为 1 时即触发该中断。
- I2C_TRANS_START_INT：当 I2C 发送一个 START 位时，即触发该中断。

- I2C_TIME_OUT_INT：在传输过程中，当 I2C SCL 保持为高或为低电平的时间超过 $2^{I2C\_TIME\_OUT\_VALUE+1}$ 个模块时钟后，即触发该中断。
- I2C_TRANS_COMPLETE_INT：当 I2C 检测到 STOP 位时，即触发该中断。
- I2C_MST_TXFIFO_UDF_INT：当 I2C 主机的 TX FIFO 下溢时，触发此中断。
- I2C_ARBITRATION_LOST_INT：当 I2C 主机的 SCL 为高电平，SDA 输出值与输入值不相等时，即触发该中断。
- I2C_BYTE_TRANS_DONE_INT：当 I2C 发送或接收一个字节，即触发该中断。
- I2C_END_DETECT_INT：当 I2C 主机命令的 op_code 为 END，且检测到 I2C END 状态时，触发此中断。
- I2C_RXFIFO_OVF_INT：当 I2C RX FIFO 上溢时，触发此中断。
- I2C_TXFIFO_WM_INT：I2C TX FIFO 水标中断。当I2C_FIFO_PRT_EN 为 1，且 TX FIFO 指针小于 I2C_TXFIFO_WM_THRHD[4:0] 时，触发此中断。
- I2C_RXFIFO_WM_INT：I2C RX FIFO 水标中断。当I2C_FIFO_PRT_EN 为 1，且 RX FIFO 指针大于 I2C_RXFIFO_WM_THRHD[4:0] 时，触发此中断。
- I2C_GENERAL_CALL_INT：从机模式下，接收到通用广播地址时，触发此中断。
- I2C_SLAVE_ADDR_UNMATCH_INT：从机模式下，接收到的从机地址和内部配置的从机地址不一致时，触发此中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 41.5 寄存器列表。

34.7 配置流程

本节提供一些典型通信场景的编程示例。ESP32-C5 中有两个 I2C 控制器，为了便于描述，下文所有图示中的 I2C 主机和从机都假定为 ESP32-C5 I2C 外设控制器。

34.7.1 I2C 主机写入从机，7 位寻址，单次命令序列

34.7.1.1 场景介绍

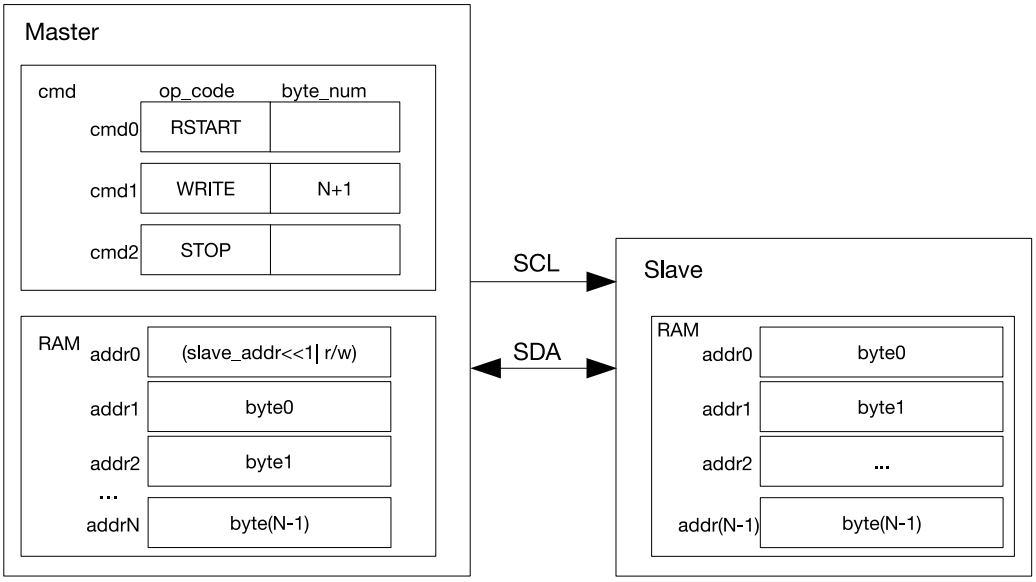


图 34.7-1. I2C 主机写 7 位寻址的从机

图 34.7-1 为 I2C 主机采用 7 位寻址写 N 个字节数据到 I2C 从机的寄存器或 RAM。如图 34.7-1 所示，主机 RAM 中第一个字节数据为 7 位从机地址 + 1 位读写标志位，其中读写标志位为 0 时表示写操作，接下来的连续空间存储待发送的数据。cmd 框中包含了相应的命令序列。

对于主机，在软件配置好命令序列以及 RAM 数据后，置位 I2C_TRANS_START 寄存器启动控制器进行数据传输。控制器的行为可分为四步：

1. 等待 SCL 线为高电平，以避免 SCL 线被其他主机或者从机占用。
2. 通过发送 START 位来执行 RSTART 命令。
3. 执行 WRITE 命令从 RAM 的首地址开始取出 N+1 个字节并依次发送给从机，其中第一个字节为地址。
4. 发送 STOP。当 I2C 主机完成 STOP 位的传输后，会产生 I2C_TRANS_COMPLETE_INT 中断。

34.7.1.2 配置示例

1. 参照章节 34.4.7，配置 I2C 主机和 I2C 从机的时序参数寄存器。
2. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2（主机）	STOP	—	—	—	—

5. 参考章节 34.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。
6. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。

7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
10. I2C 主机发送数据，并根据 ack_check_en（主机）的配置决定是否对 ACK 信号进行检测。
11. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 34.4.10。
12. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作，具体做法参照章节 34.4.10。

    若接收数据 N 大于 RX FIFO 深度，另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
13. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.7.2 I2C 主机写入从机，10 位寻址，单次命令序列

34.7.2.1 场景介绍

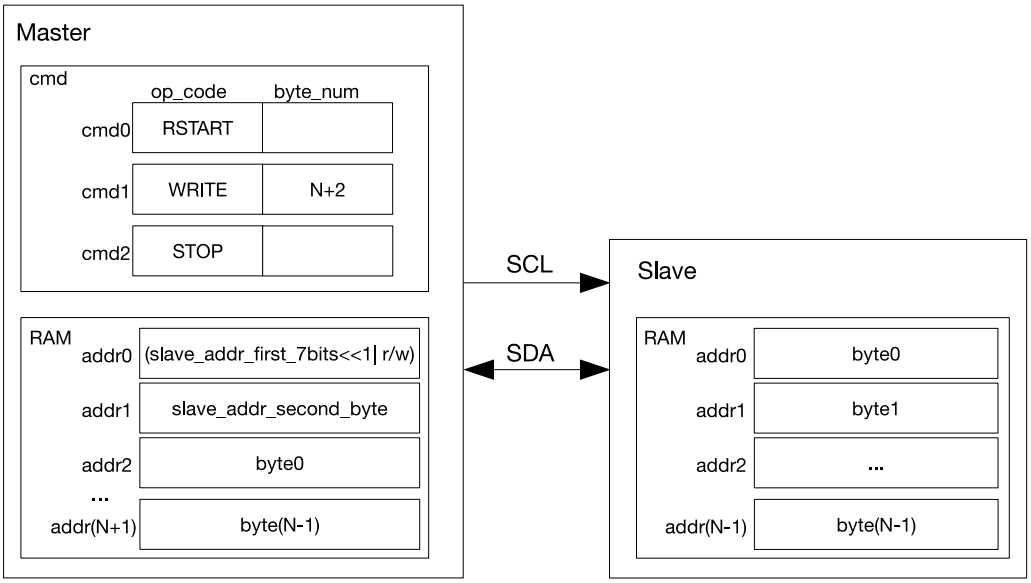


图 34.7-2. I2C 主机写 10 位寻址的从机

图 34.7-2 为 I2C 主机写 N 个字节到 10 位地址 I2C 从机的配置图。整个配置和传输过程都和 34.7.1 中类似，除了在传输的开始主机在 10 位寻址模式下需要发送两字节地址段。由于 10 位从机地址比 7 位地址多一个字节，

所以 WRITE 命令对应的 byte_num 以及 TX RAM 中数据数量都相应增加 1。

34.7.2.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2（主机）	STOP	—	—	—	—

4. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的 10 位从机地址，并将 I2C_ADDR_10BIT_EN（从机）置 1 使能 10 位寻址模式。
5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 ((0x78 | I2C_SLAVE_ADDR[9:8])«1)|R/W，第二个地址字节是I2C_SLAVE_ADDR[7:0]。之后就是要发送的数据，可选 FIFO 方式和直接访问方式。
6. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
7. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
9. I2C 主机发送数据，并根据 ack_check_en（主机）的配置判断是否进行 ACK 检测。
10. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 34.4.10。
11. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作，具体做法参照章节 34.4.10。

    若接收数据 N 大于 RX FIFO 深度，另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
12. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.7.3 I2C 主机写入从机，7 位双地址寻址，单次命令序列

34.7.3.1 场景介绍

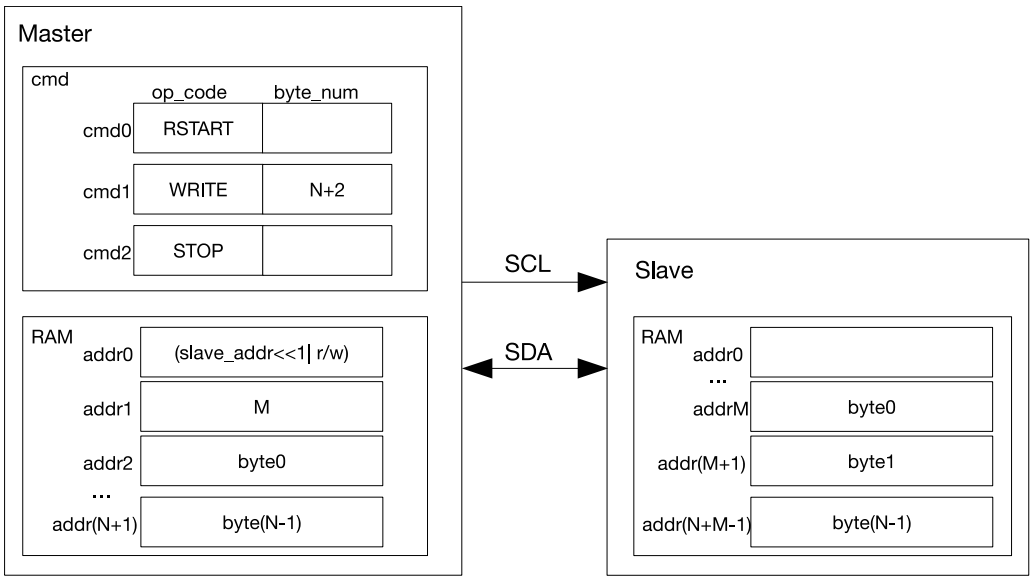


图 34.7-3. I2C 主机写 7 位双地址寻址从机

图34.7-3 为 I2C 主机采用 7 位双寻址模式写 N 个字节数据到 I2C 从机的寄存器或 RAM 的值。整个配置和传输过程都和章节 34.7.1 中类似，区别是传输的开始主机在 7 位双寻址模式下需要发送两个字节。双地址的第一个地址是 7 位从机地址，第二个地址是 I2C 从机的内存地址（即图 34.7-3 中的 addrM）。双地址模式下，RX RAM 必须采用 non-FIFO 方式访问。另一个区别是，I2C 从机将接收到的数据 byte0 ~ byte (N-1) 从 RX RAM 中的 addrM 开始依次存储，addrM 就是主机发送的 I2C 内存地址。当超出地址 31 后会从地址 0 开始继续存储。

34.7.3.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 设置I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2（主机）	STOP	—	—	—	—

5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，可以用 FIFO 方式或直接访问方式。
6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。

- 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
- 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。

10. I2C 从机接收到 I2C 主机发送的内存地址，完成 RX RAM 的地址偏移。
11. I2C 主机发送数据，并根据 ack_check_en（主机）的配置判断是否进行 ACK 检测。
12. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 34.4.10。
13. 若接收数据 N 大于 RX FIFO 深度，置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
14. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.7.4 I2C 主机写入从机，7 位寻址，多次命令序列

34.7.4.1 场景介绍

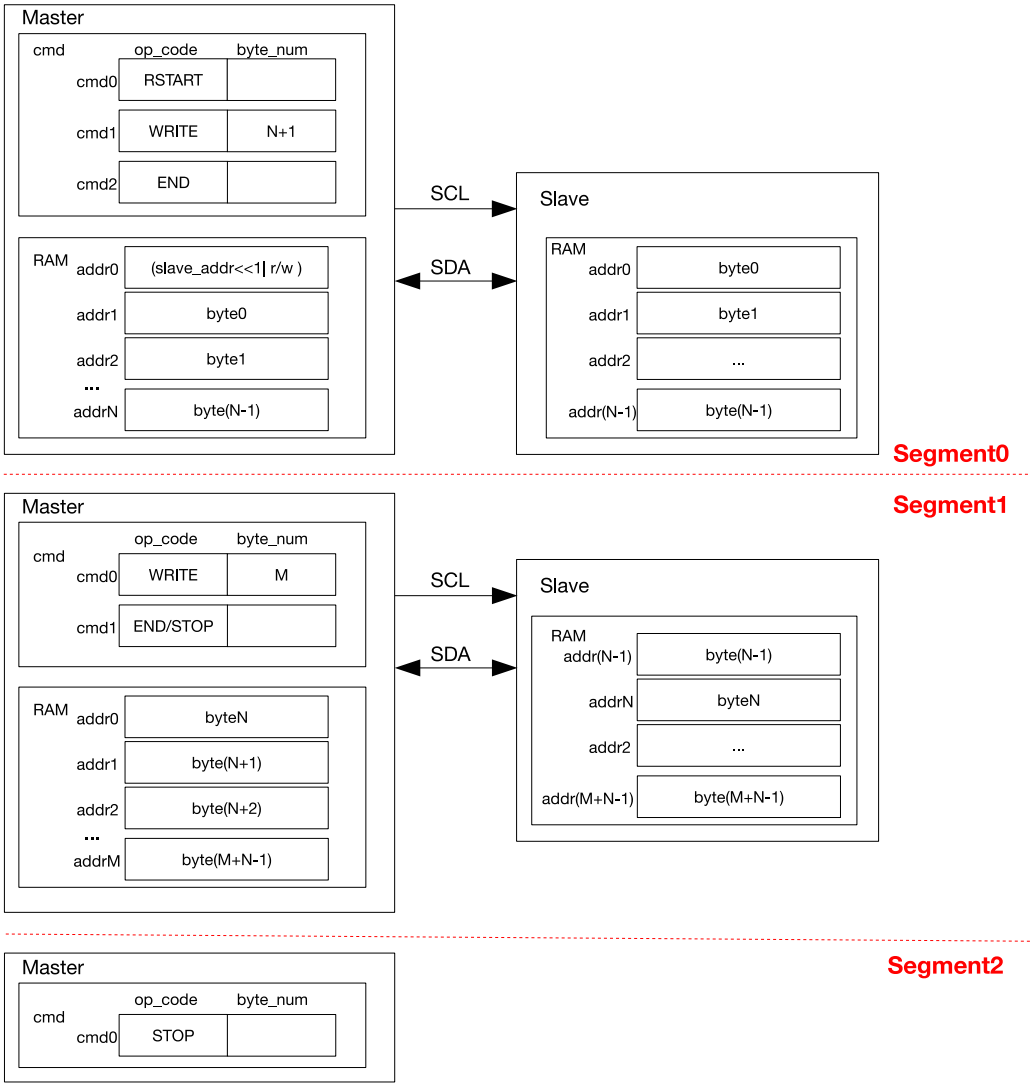


图 34.7-4. I2C 主机分段写 7 位寻址的从机

RAM 的大小只有 TX/RX FIFO 深度，当进行大量的数据传输时，建议使用多次命令序列进行分段传输。每次命令序列以 END 命令结束。当控制器执行 END 命令时，将拉低 SCL 线，软件此时可以更新命令序列寄存器和 RAM 的内容以用于下一次命令序列的传输。

以两段和三段传输为例，如图 34.7-4 所示为 I2C 主机分成三段或者两段写从机。配置 I2C 主机的命令序列如第一段所示，并且在主机的 RAM 中准备好数据，置位 I2C_TRANS_START，I2C 主机即开始数据传输。在执行到 END 命令后，I2C 主机会关闭 SCL 时钟，并将 SCL 线拉低来防止其他设备占用 I2C 总线。此时控制器产生 I2C_END_DETECT_INT 中断。

在检测到 I2C_END_DETECT_INT 中断后，软件可以更新命令序列以及 RAM 中的内容如第二段所示，并清除 I2C_END_DETECT_INT 中断。当第二段中 cmd1 为 STOP 时，不需要第三段，即为两段写从机。置位 I2C_TRANS_START 后，I2C 主机继续发送数据，并在最后发送 STOP 位。

当为三段写从机时，I2C 主机在第二段发送完数据，并检测到 I2C 主机的 I2C_END_DETECT_INT 中断后，即可配置 cmd 如第三段所示。置位 I2C_TRANS_START 后，I2C 主机即产生 STOP 位，从而停止传输。

请注意，在两个分段之间，I2C 总线上的其他主机设备不会占用总线。只有在发送了 STOP 命令后总线才会被

释放。任何情况下，置位 I2C_FSM_RST 可复位 I2C 控制器，该寄存器硬件自清 I2C_FSM_RST。

34.7.4.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2（主机）	END	—	—	—	—

4. 参考章节 34.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。
5. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
6. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
7. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
9. I2C 主机发送数据，并根据 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
10. 等到 I2C_END_DETECT_INT（主机）中断产生后，设置I2C_END_DETECT_INT_CLR（主机）为 1 来清除中断。
11. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	WRITE	ack_value	ack_exp	1	M
I2C_COMMAND1（主机）	END/STOP	—	—	—	—

12. 向 I2C 主机的 TX RAM 写入 M 个要发送的数据，可以用 FIFO 方式或直接访问方式。
13. 向 I2C_TRANS_START（主机）位写 1 开始传输，并重复步骤 9 的流程。
14. 若指令为 STOP，I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。
15. 若I2C_COMMAND1（主机）的指令为 END，则重复步骤 10。
16. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1（主机）	STOP	—	—	—	—

17. 向 I2C_TRANS_START（主机）位写 1 开始传输。
18. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.7.5 I2C 主机读取从机，7 位寻址，单次命令序列

34.7.5.1 场景介绍

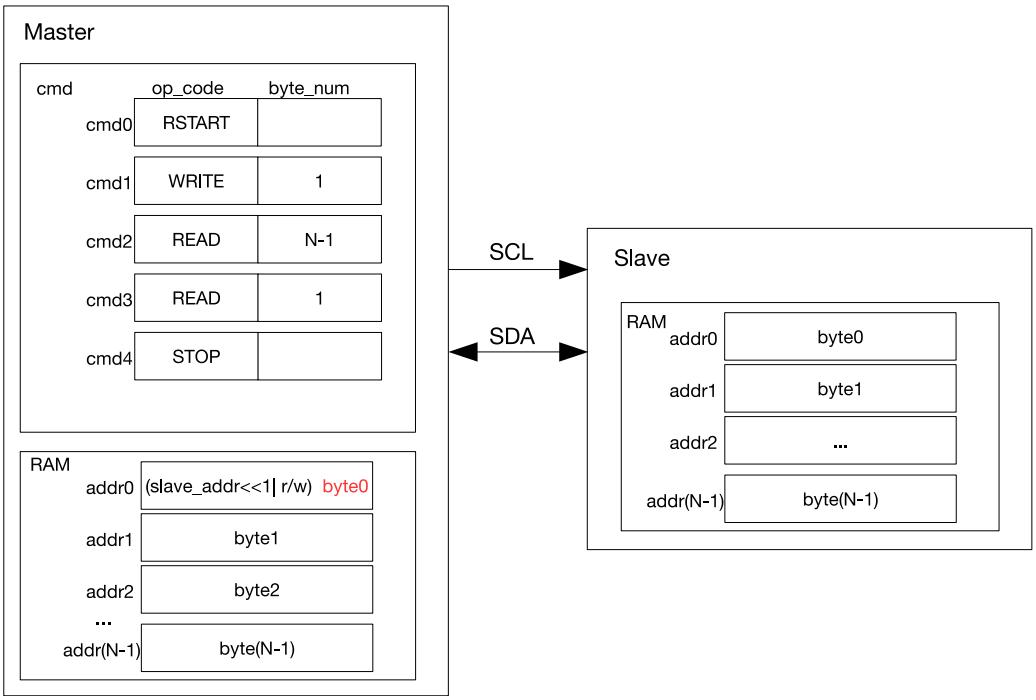


图 34.7-5. I2C 主机读 7 位寻址的从机

图 34.7-5 I2C 主机从 7 位寻址 I2C 从机读取 N 个字节数据的寄存器或 RAM 的值。cmd1 为 WRITE 命令，I2C 主机会将 I2C 从机的地址发送出去。该命令发送的字节是 7 位 I2C 从机地址以及读写标志位。读写标志位为 1 表示读操作。I2C 从机在地址匹配成功之后即开始发送数据给 I2C 主机。I2C 主机根据 READ 命令中设置的 ack_value，在接收完一个字节的数据之后回复 ACK。

图 34.7-5 中 READ 分成两次，I2C 主机对 cmd2 中 N-1 个数据均回复 ACK，对 cmd3 中的数据即传输的最后一个数据回复 NACK，实际使用时可以根据需要进行配置。在存储接收的数据时，I2C 主机从 RX RAM 的首地址开始存储，即为图中红色 byte0 存储的位置。

34.7.5.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。

4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	1
I2C_COMMAND2（主机）	READ	0	0	1	N-1
I2C_COMMAND3（主机）	READ	1	0	1	1
I2C_COMMAND4（主机）	STOP	—	—	—	—

5. 参考章节 34.4.10，向 I2C 主机的 TX RAM 写入从机地址。可选 FIFO 方式和直接访问方式。
6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 34.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. 等待 I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
12. 参考章节 34.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
13. 将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
14. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
15. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
16. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 I2C_NACK_INT 中断，停止发送。
17. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.7.6 I2C 主机读取从机，10 位寻址，单次命令序列

34.7.6.1 场景介绍

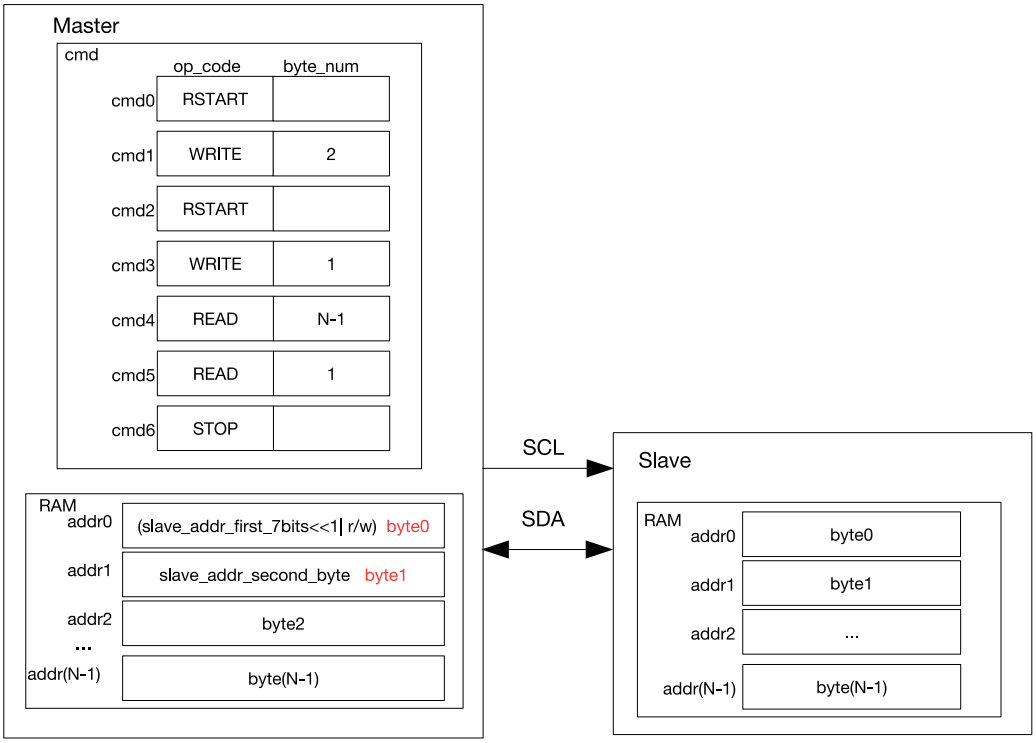


图 34.7-6. I2C 主机读 10 位寻址的从机

图 34.7-6 为 I2C 主机从 10 位寻址的 I2C 从机中读取数据的寄存器或 RAM 的值。相比于 7 位寻址，I2C 主机的第一写命令的字节数为两个字节，相应 TX RAM 中存储两个字节的 I2C 从机 10 位地址，且第一个地址字节的 R/W 位为 W（主机）。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位为 R（从机）。之后主机从从机中读取数据。两个字节地址的配置方式与章节 34.7.2 的相同。

34.7.6.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照 I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1
I2C_COMMAND5（主机）	READ	1	0	1	1
I2C_COMMAND6（主机）	STOP	—	—	—	—

5. 在 I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的 10 位从机地址，并将 I2C_ADDR_10BIT_EN（从机）置 1 使能 10 位寻址模式。
6. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 ((0x78 | I2C_SLAVE_ADDR[9:8])«1)|R/W，R/W 位为 W（主机）；第二个地址字节是I2C_SLAVE_ADDR[7:0]。第三个字节是重复的第一个地址字节加上 R/W 位，其中 R/W 位为 R（从机）。可选 FIFO 方式和直接访问方式。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 34.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. I2C 主机发送 RSTART 命令，并发送 TX RAM 里的第三个字节，即为重复的地址字节和 R 位。
12. I2C 从机重复执行步骤 10，若地址匹配，继续后面的步骤。
13. 等待 I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
14. 参考章节 34.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
15. 将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
16. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
17. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
18. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 I2C_NACK_INT 中断，停止发送。
19. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.7.7 I2C 主机读取从机，7 位双寻址，单次命令序列

34.7.7.1 场景介绍

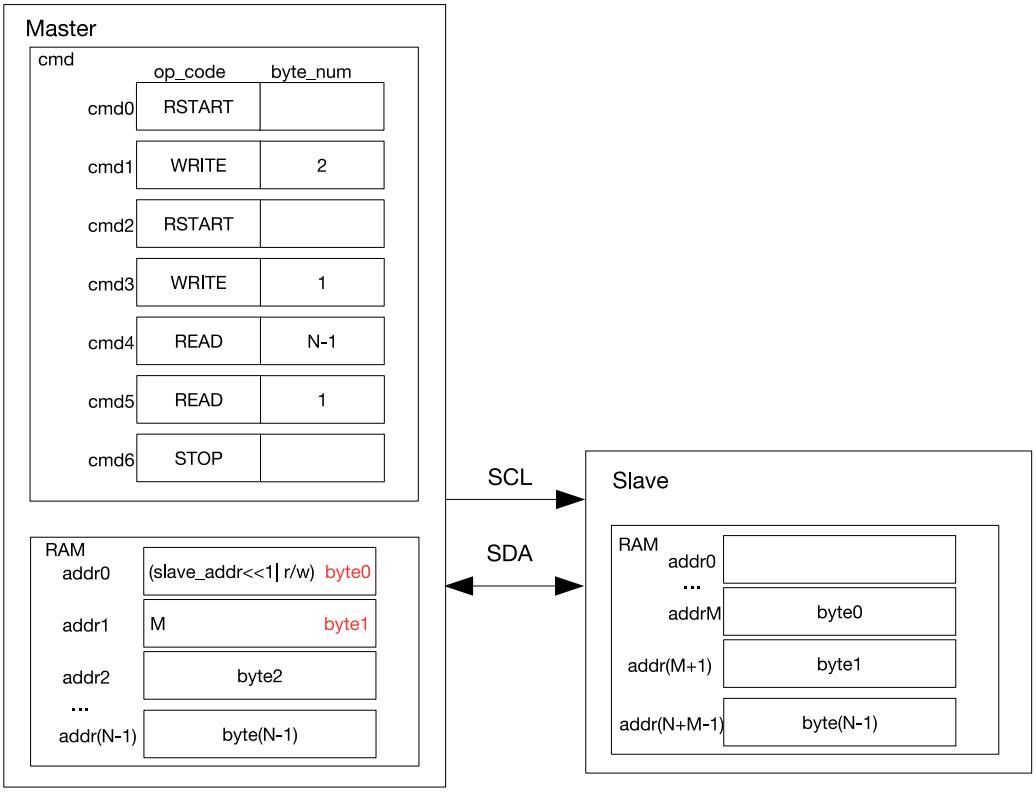


图 34.7-7. I2C 主机从 7 位寻址从机的 M 地址读取 N 个数据

图 34.7-7 为 I2C 主机从 I2C 从机中指定地址读取数据的寄存器或 RAM 的值。主机在传输开始发送 2 个地址字节，第一个地址字节是从机的 7 位地址加 R/W 位，R/W 位为 W（主机）；第二个地址字节是从机的内存地址 M。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位变为 R（从机）。之后主机从从机的 AddrM 地址开始读取数据。

34.7.7.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 设置 I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
4. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
5. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1

I2C_COMMAND5（主机）	READ	1	0	1	1
I2C_COMMAND6（主机）	STOP	—	—	—	—

6. 在I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的 7 位地址，I2C_ADDR_10BIT_EN（从机）置 0 使能 7 位寻址模式。
7. 参考章节 34.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 (I2C_SLAVE_ADDR[6:0])«1)|R/W，R/W 位为 W（主机）；第二个地址字节是 I2C 从机的内存地址 M。第三个字节是重复的第一个地址字节加上 R/W 位，其中 R/W 位为 R（从机）。可选 FIFO 方式和直接访问方式。
8. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
9. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
10. 参考章节 34.4.14，启动 I2C 从机的传输。
11. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
12. I2C 从机接收到 I2C 主机发送的内存地址，完成 TX RAM 的地址偏移。
13. I2C 主机发送 RSTART 命令，并发送 TX RAM 里的第三个字节，即为重复的地址字节和 R 位。
14. I2C 从机重复执行步骤 11，若地址匹配，继续后面的步骤。
15. 等待 I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
16. 参考章节 34.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
17. 将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
18. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
19. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1，清除中断；I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
20. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 NACK 中断，停止发送。
21. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.7.8 I2C 主机读取从机，7 位寻址，多次命令序列

34.7.8.1 场景介绍

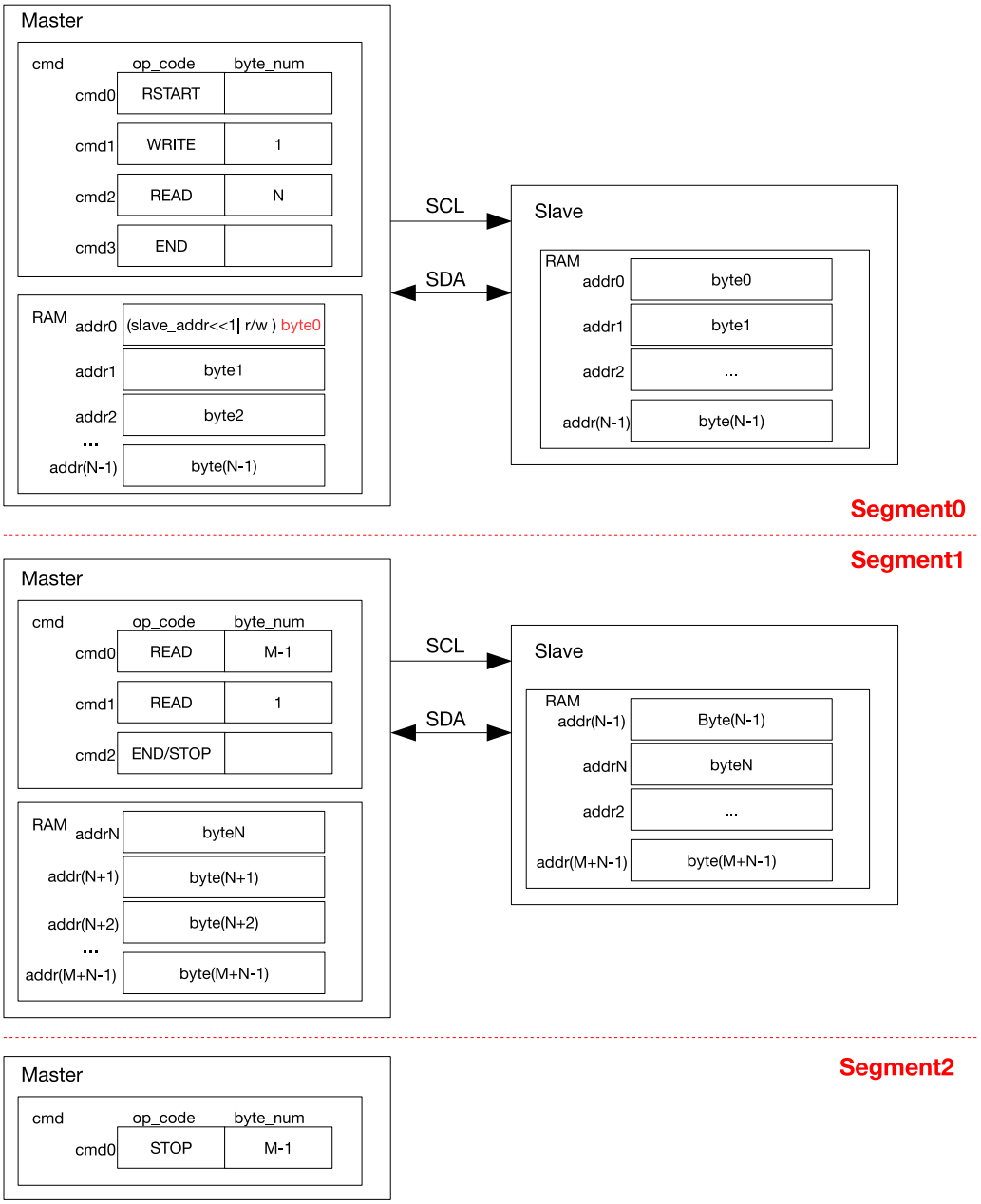


图 34.7-8. I2C 主机分段读 7 位寻址的从机

图 34.7-8 为 I2C 主机通过 END 命令分三段或者分两段，从 I2C 从机读取 N+M 个数据的流程图。

1. 第一段流程和 34.7-5 类似，只是最后一个命令变为 END。
2. 接着在从机的 TX RAM 中准备好数据，置位 I2C_TRANS_START，当执行到 END 命令时，I2C 主机可以更新命令寄存器和 RAM 的内容，如第二段所示，并且清零其对应的 I2C_END_DETECT_INT 中断。当第二段中 cmd2 为 STOP 时，即两段读 I2C 从机，置位 I2C_TRANS_START，I2C 主机继续传输数据，最后发送 STOP 位来停止传输。
3. 当第二段中 cmd2 为 END 时，在 I2C 主机完成第二次数据传输，并检测到 I2C 主机的 I2C_END_DETECT_INT 中断后，配置 cmd 如第三段所示。置位 I2C_TRANS_START，I2C 主机发送 STOP 位停止传输。

34.7.8.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照 I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	1
I2C_COMMAND2（主机）	READ	0	0	1	N
I2C_COMMAND3（主机）	END	—	—	—	—

5. 向 I2C 主机的 TX RAM 写入从机地址，可选 FIFO 方式和直接访问方式。
6. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 34.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. 等待 I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
12. 参考章节 34.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
13. 将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
14. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
15. 若 I2C 主机一个 READ 指令要读取的数 N 或 M 超过 I2C 从机的 TX FIFO 深度个字节，当 I2C 从机的 TX RAM 中发送数据全部发完，为空时产生I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
16. 等到一次 READ 指令完成，I2C 主机执行 END 指令，I2C_END_DETECT_INT（主机）中断产生后，设置 I2C_END_DETECT_INT_CLR（主机）为 1 来清除中断。

17. 更新 I2C 主机的指令寄存器，有两种设置方式：

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	READ	ack_value	ack_exp	1	M
I2C_COMMAND1（主机）	END	—	—	—	—

或者

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	READ	0	0	1	M-1
I2C_COMMAND1（主机）	READ	1	0	1	1
I2C_COMMAND2（主机）	STOP	—	—	—	—

18. 向 I2C 从机的 TX RAM 写入 M 个字节要发送的数据，若 M 大于 TX FIFO 深度，重复步骤 12，可以用 FIFO 方式或直接访问方式。
19. 向 I2C_TRANS_START（主机）位写 1 开始传输，并重复步骤 14 的流程。
20. 若最后一个指令为 STOP，则当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 NACK 中断，停止发送。I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。
21. 若最后一个指令为 END，则重复步骤 16，并在完成后继续下面的步骤。
22. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1（主机）	STOP	—	—	—	—

23. 向 I2C_TRANS_START（主机）位写 1 开始传输。
24. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

34.8 寄存器列表

34.8.1 I2C 寄存器列表

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时序寄存器			
I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平的保持时间	0x0000	R/W
I2C_SDA_HOLD_REG	配置 SCL 下降沿后的数据保持时间	0x0030	R/W
I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平保持时间	0x0038	R/W
I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟时间	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	配置 STOP 命令生成时 SDA 和 SCL 上升沿之间的间隔时间	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x007C	R/W
配置寄存器			
I2C_CTR_REG	传输配置寄存器	0x0004	varies
I2C_TO_REG	数据接收超时控制寄存器	0x000C	R/W
I2C_SLAVE_ADDR_REG	从机地址配置寄存器	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	R/W
I2C_FILTER_CFG_REG	SCL 和 SDA 滤波配置寄存器	0x0050	R/W
I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
I2C_SCL_STRETCH_CONF_REG	配置 I2C 从机 SCL 时钟拉伸	0x0084	varies
状态寄存器			
I2C_SR_REG	I2C 的工作状态寄存器	0x0008	RO
I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
I2C_DATA_REG	Rx FIFO 读取数据寄存器	0x001C	HRO
中断寄存器			
I2C_INT_RAW_REG	原始中断状态	0x0020	R/ SS/ WTC
I2C_INT_CLR_REG	中断清除位	0x0024	WT
I2C_INT_ENA_REG	中断使能位	0x0028	R/W
I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
I2C_COMD0_REG	I2C 命令寄存器 0	0x0058	varies
I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	varies

名称	描述	地址	访问
I2C_COMD2_REG	I2C 命令寄存器 2	0x0060	varies
I2C_COMD3_REG	I2C 命令寄存器 3	0x0064	varies
I2C_COMD4_REG	I2C 命令寄存器 4	0x0068	varies
I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	varies
I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	varies
I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	varies
版本寄存器			
I2C_DATE_REG	版本寄存器	0x00F8	R/W
地址寄存器			
I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO 基地址寄存器	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO 基地址寄存器	0x0180	HRO

34.8.2 LP_I2C 寄存器列表

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时序寄存器			
LP_I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平的保持时间	0x0000	R/W
LP_I2C_SDA_HOLD_REG	配置 SCL 下降沿后的数据保持时间	0x0030	R/W
LP_I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
LP_I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平保持时间	0x0038	R/W
LP_I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
LP_I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟时间	0x0044	R/W
LP_I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	R/W
LP_I2C_SCL_STOP_SETUP_REG	配置 STOP 命令生成时 SDA 和 SCL 上升沿之间的间隔时间	0x004C	R/W
LP_I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0078	R/W
LP_I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x007C	R/W
配置寄存器			
LP_I2C_CTR_REG	传输配置寄存器	0x0004	varies
LP_I2C_TO_REG	数据接收超时控制寄存器	0x000C	R/W
LP_I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	R/W
LP_I2C_FILTER_CFG_REG	SCL 和 SDA 滤波配置寄存器	0x0050	R/W
LP_I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
状态寄存器			
LP_I2C_SR_REG	I2C 的工作状态寄存器	0x0008	RO
LP_I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
LP_I2C_DATA_REG	RX FIFO 读取数据寄存器	0x001C	RO

名称	描述	地址	访问
中断寄存器			
LP_I2C_INT_RAW_REG	原始中断状态	0x0020	R/ SS/ WTC
LP_I2C_INT_CLR_REG	中断清除位	0x0024	WT
LP_I2C_INT_ENA_REG	中断使能位	0x0028	R/W
LP_I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
LP_I2C_COMD0_REG	I2C 命令寄存器 0	0x0058	varies
LP_I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	varies
LP_I2C_COMD2_REG	I2C 命令寄存器 2	0x0060	varies
LP_I2C_COMD3_REG	I2C 命令寄存器 3	0x0064	varies
LP_I2C_COMD4_REG	I2C 命令寄存器 4	0x0068	varies
LP_I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	varies
LP_I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	varies
LP_I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	varies
版本寄存器			
LP_I2C_DATE_REG	版本寄存器	0x00F8	R/W
地址寄存器			
LP_I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO 基地址寄存器	0x0100	HRO
LP_I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO 基地址寄存器	0x0180	HRO

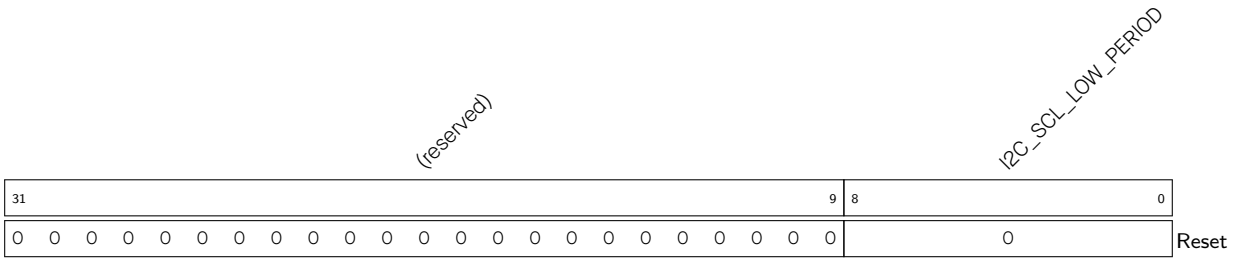
34.9 寄存器

34.9.1 I2C 寄存器

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 34.1. I2C_SCL_LOW_PERIOD_REG (0x0000)

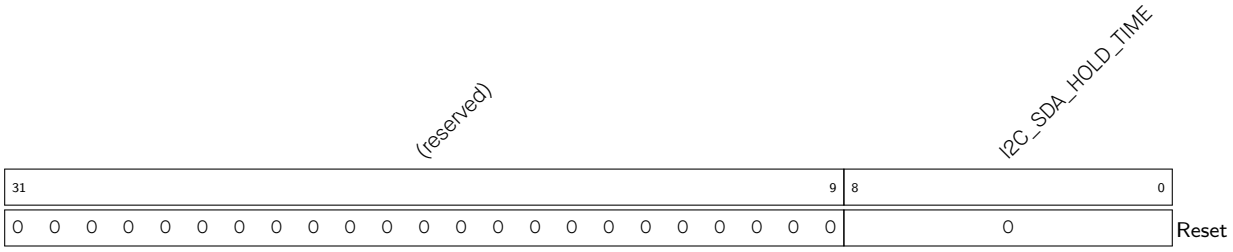


I2C_SCL_LOW_PERIOD 配置主机模式下 SCL 低电平的保持时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.2. I2C_SDA_HOLD_REG (0x0030)

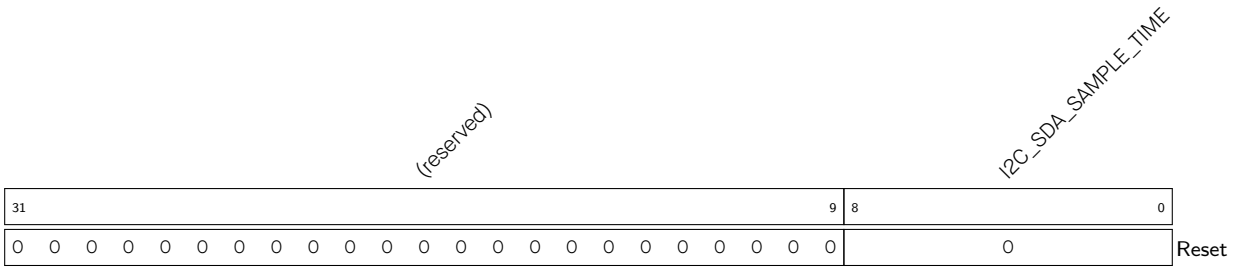


I2C_SDA_HOLD_TIME 配置 SCL 下降沿后的数据保持时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.3. I2C_SDA_SAMPLE_REG (0x0034)

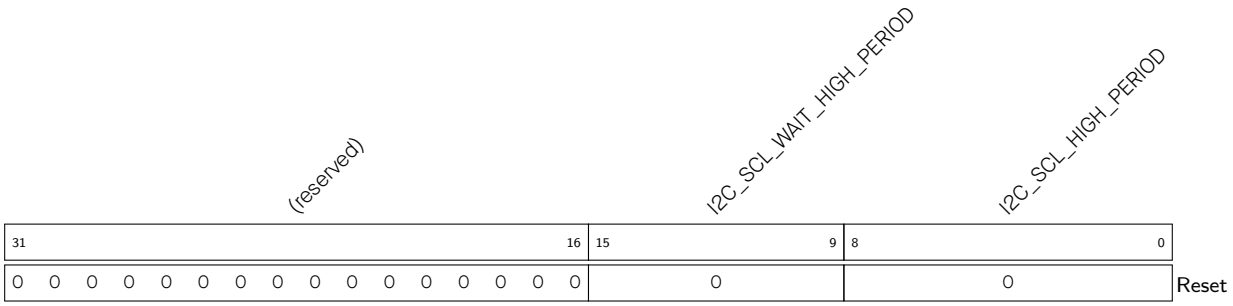


I2C_SDA_SAMPLE_TIME 配置采样 SDA 的时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD 配置 SCL 在主机模式下保持高电平的时间。

单位：I2C_SCLK 时钟周期数

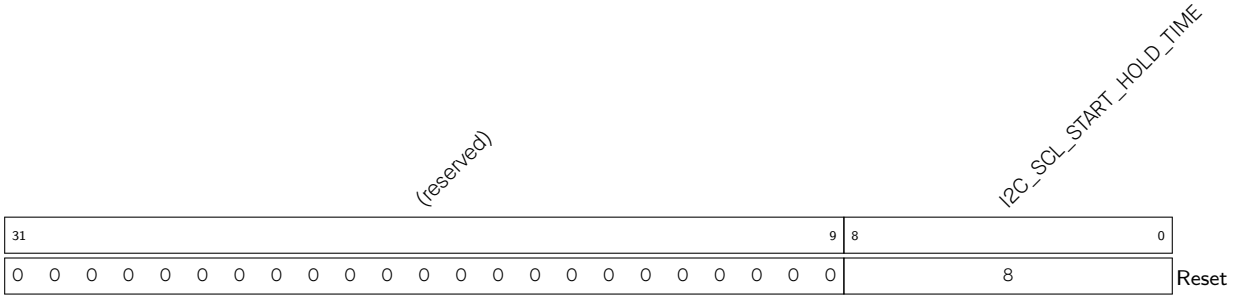
(R/W)

I2C_SCL_WAIT_HIGH_PERIOD 配置 SCL_FSM 等待 SCL 在主机模式下翻转至高电平的时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.5. I2C_SCL_START_HOLD_REG (0x0040)

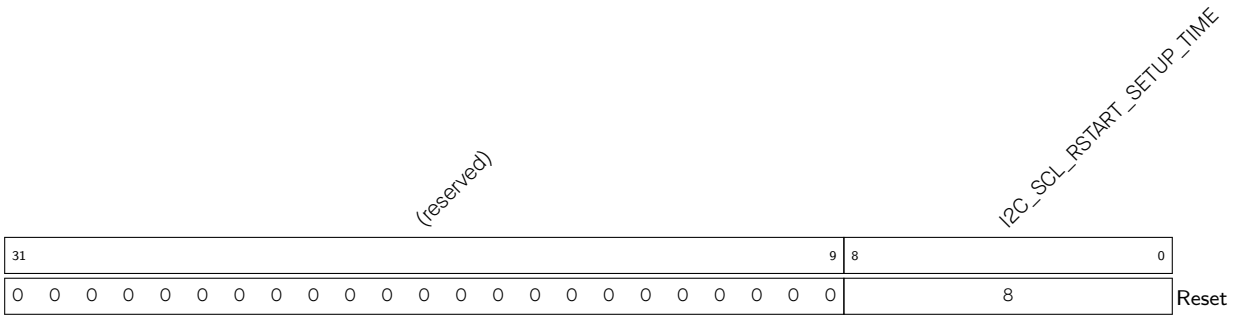


I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.6. I2C_SCL_RSTART_SETUP_REG (0x0044)

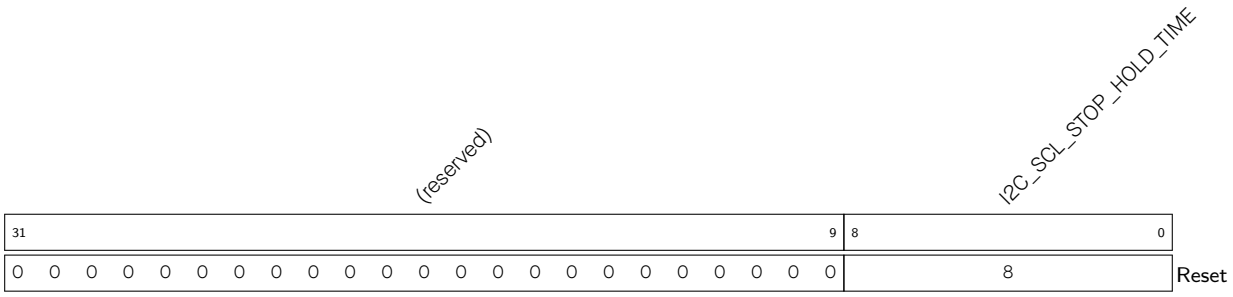


I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.7. I2C_SCL_STOP_HOLD_REG (0x0048)

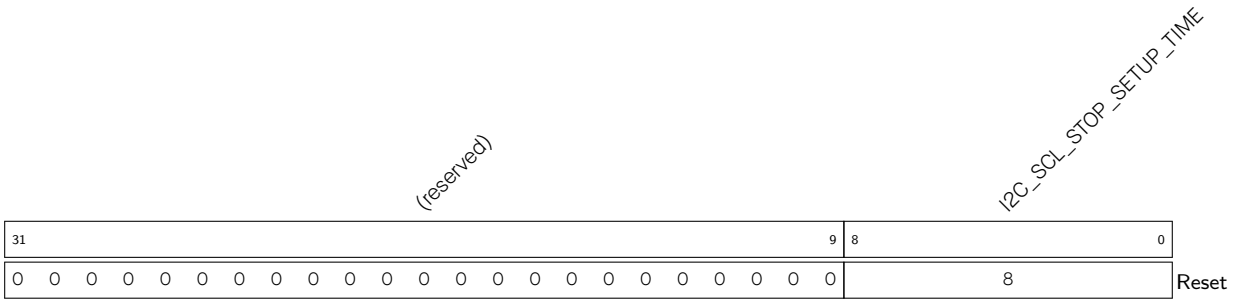


I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.8. I2C_SCL_STOP_SETUP_REG (0x004C)

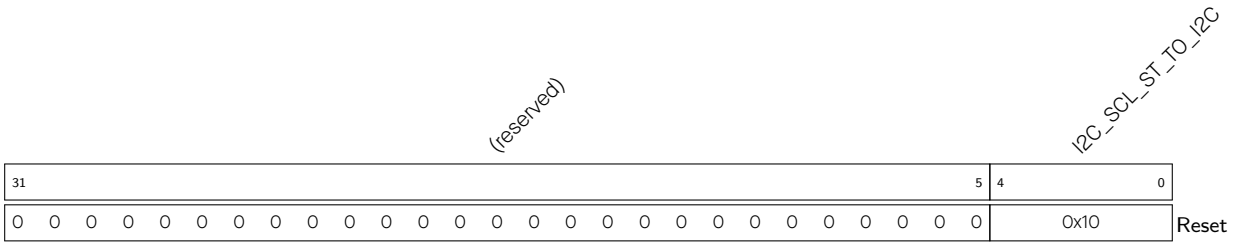


I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



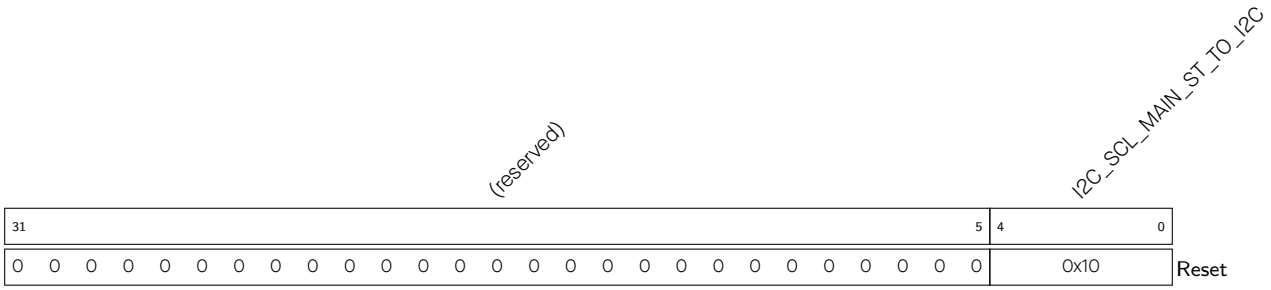
I2C_SCL_ST_TO_I2C 配置 SCL_FSM 状态保持不变的最大时间。该值应该大于 1，小于等于 23。

$$\text{最大时钟周期数} = 2^{\text{I2C\_SCL\_ST\_TO\_I2C}+1} - 1$$

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

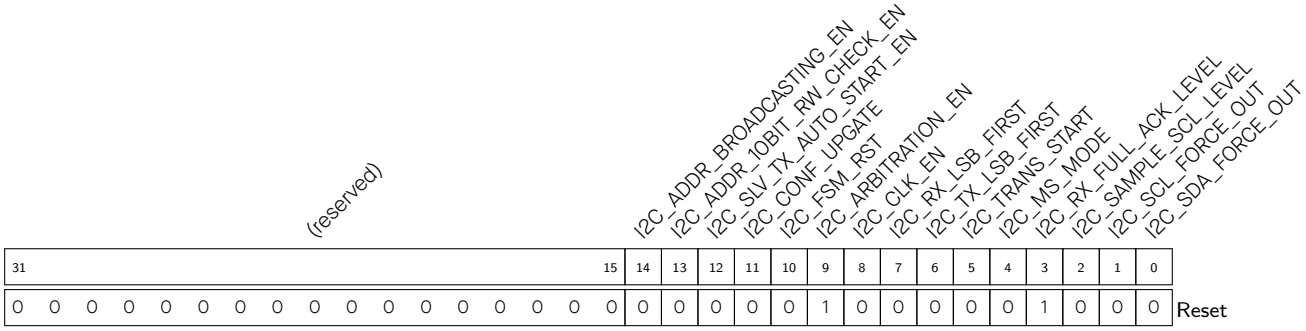


I2C_SCL_MAIN_ST_TO_I2C 配置 SCL_MAIN_FSM 状态保持不变的最大时间，不能大于 23。实际周期是 2^(I2C_SCL_MAIN_ST_TO_I2C+1)–1。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.11. I2C_CTR_REG (0x0004)



I2C_SDA_FORCE_OUT 配置 SDA 输出模式。

0：开漏输出

1：直接输出

(R/W)

I2C_SCL_FORCE_OUT 配置 SCL 输出模式。

0：开漏输出

1：直接输出

(R/W)

I2C_SAMPLE_SCL_LEVEL 配置 SDA 的采样模式。

0：SCL 为高电平时采样 SDA 数据

1：SCL 为低电平时采样 SDA 数据

(R/W)

I2C_RX_FULL_ACK_LEVEL 用于配置主机在接收的 RX RAM 达到阈值时需发送的 ACK 电平值。(R/W)

I2C_MS_MODE 配置该模块为 I2C 主机或从机。

0：从机

1：主机

(R/W)

I2C_TRANS_START 配置从机是否开始发送 TX FIFO 中的数据。

0：无效

1：开始

(WT)

I2C_TX_LSB_FIRST 控制待发送数据的发送顺序。

0：从最高有效位开始发送数据

1：从最低有效位开始发送数据

(R/W)

I2C_RX_LSB_FIRST 控制接收数据的存储顺序。

0：从最高有效位开始接收数据

1：从最低有效位开始接收数据

(R/W)

见下页...

Register 34.11. I2C_CTR_REG (0x0004)

... 接上页

I2C_CLK_EN 门控寄存器的时钟信号。

0：仅当通过软件读写寄存器时支持时钟

1：强制寄存器时钟开启

(R/W)

I2C_ARBITRATION_EN 配置使能 I2C 总线仲裁检测。

0：无效

1：使能

(R/W)

I2C_FSM_RST 配置复位 SCL_FSM。

0：无效

1：复位

(WT)

I2C_CONF_UPGATE 配置此位实现同步。

0：无效

1：同步

(WT)

I2C_SLV_TX_AUTO_START_EN 配置从机自动发送数据的使能位。

0：禁用

1：使能

(R/W)

I2C_ADDR_10BIT_RW_CHECK_EN 配置使能 10 位寻址模式的读写标志位检查功能，检查读写标志是否符合协议。

0：不检查

1：检查

(R/W)

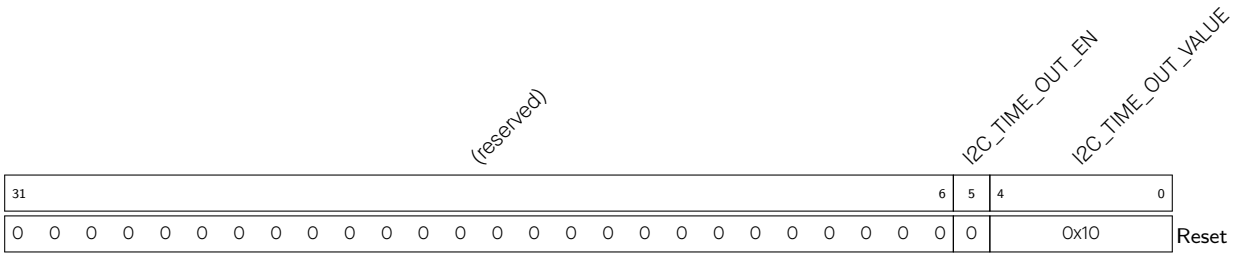
I2C_ADDR_BROADCASTING_EN 配置是否支持 7 位寻址模式的广播功能。

0：不支持

1：支持

(R/W)

Register 34.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE 配置 SCL 在高电平或低电平停滞的超时阈值周期。实际周期是 2^(I2C_TIME_OUT_VALUE+1)–1。

单位：I2C_SCLK 时钟周期数

(R/W)

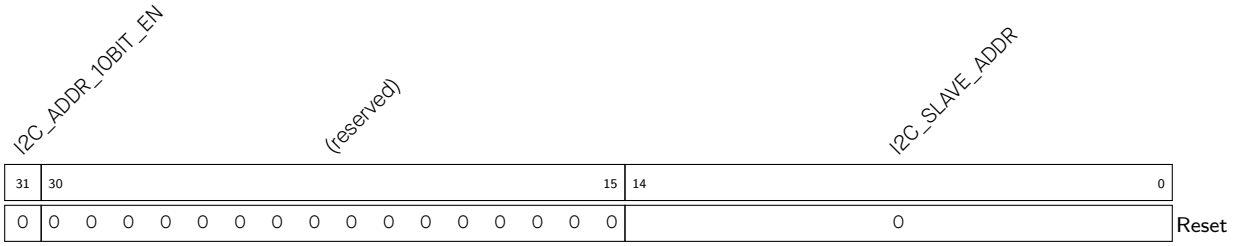
I2C_TIME_OUT_EN 配置使能超时控制。

0：无效

1：使能

(R/W)

Register 34.13. I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR 配置 I2C 从机的从机地址。(R/W)

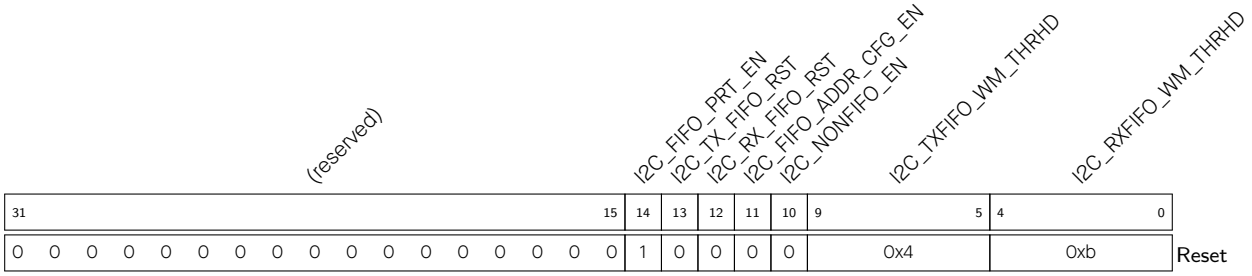
I2C_ADDR_10BIT_EN 配置在主机模式下使能从机的 10 位寻址模式。

0：无效

1：使能

(R/W)

Register 34.14. I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD 配置在直接访问模式下，RX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 I2C_RXFIFO_WM_THRHD[4:0] 时，I2C_RXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_TXFIFO_WM_THRHD 配置在直接访问模式下，TX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 TX FIFO 计数值大于 I2C_TXFIFO_WM_THRHD[4:0] 时，I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_NONFIFO_EN 配置使能 APB 直接访问。(R/W)

I2C_FIFO_ADDR_CFG_EN 配置从机是否使能双寻址模式。当该模式使能时，在 I2C 地址字节之后接收的字节代表 I2C 从机 RAM 中的偏移地址。

0：禁用

1：使能

(R/W)

I2C_RX_FIFO_RST 配置复位 RX FIFO。

0：无效

1：复位

(R/W)

I2C_TX_FIFO_RST 配置复位 TX FIFO。

0：无效

1：复位

(R/W)

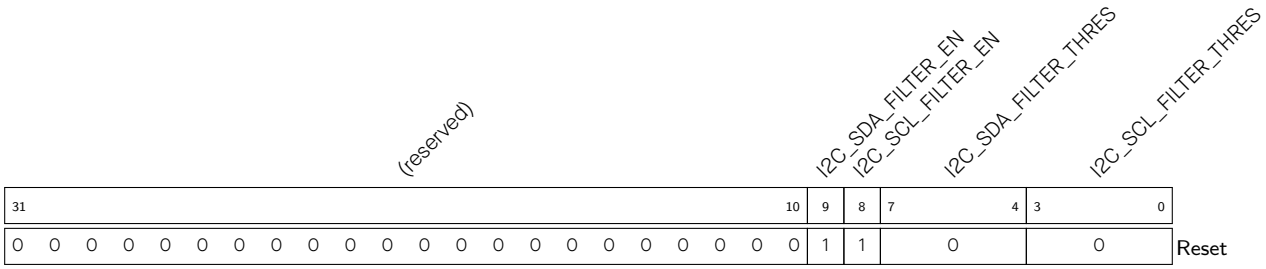
I2C_FIFO_PRT_EN 配置在直接访问模式下是否使能 FIFO 指针。该位控制有效位以及 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的中断。

0：无效

1：使能

(R/W)

Register 34.15. I2C_FILTER_CFG_REG (0x0050)



I2C_SCL_FILTER_THRES 配置 SCL 过滤脉冲宽度的阈值。当 SCL 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SDA_FILTER_THRES 配置 SDA 过滤脉冲宽度的阈值。当 SDA 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SCL_FILTER_EN 配置使能 SCL 的滤波功能。

0：无效

1：复位

(R/W)

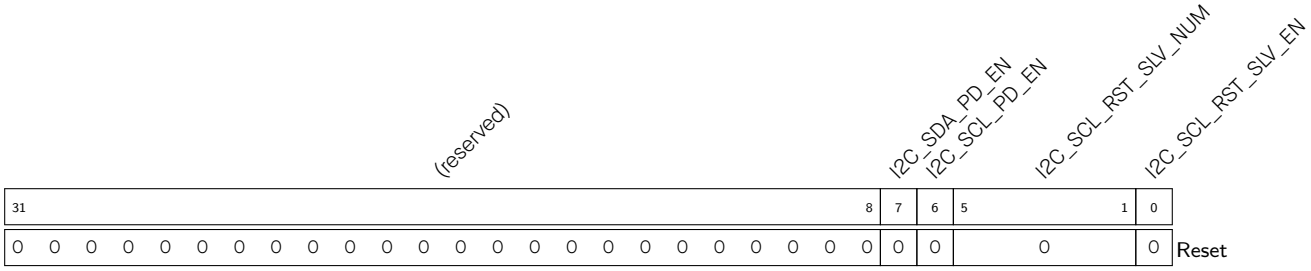
I2C_SDA_FILTER_EN 配置使能 SDA 的滤波功能。

0：无效

1：复位

(R/W)

Register 34.16. I2C_SCL_SP_CONF_REG (0x0080)



I2C_SCL_RST_SLV_EN 在 I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 I2C_SCL_RST_SLV_NUM。

0：无效

1：发送 SCL 脉冲

(R/W/SC)

I2C_SCL_RST_SLV_NUM 配置 I2C 主机模式下生成的 SCL 脉冲。

仅在 I2C_SCL_RST_SLV_EN 为 1 时有效。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SCL_PD_EN 配置降低 I2C SCL 输出功耗。

0：正常功耗

1：不工作，降低功耗

仅在 I2C_SCL_FORCE_OUT 为 1 时有效。

(R/W)

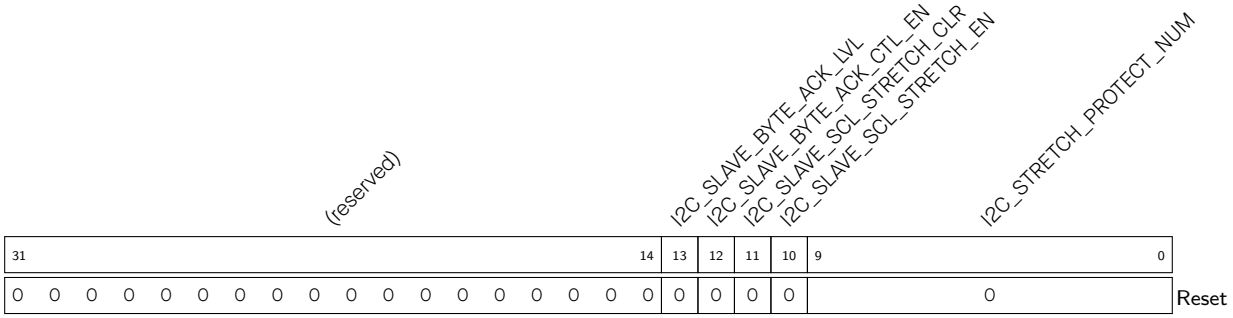
I2C_SDA_PD_EN 配置降低 I2C SDA 输出功耗。

0：正常功耗

1：不工作，降低功耗

仅在 I2C_SDA_FORCE_OUT 为 1 时有效。(R/W)

Register 34.17. I2C_SCL_STRETCH_CONF_REG (0x0084)



I2C_STRETCH_PROTECT_NUM 配置 SCL 时钟拉伸释放后的保护时间，通常设置为大于 SDA 的建立时间。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SLAVE_SCL_STRETCH_EN 配置使能从机的 SCL 时钟拉伸。I2C_SLAVE_SCL_STRETCH_EN 为 1，且出现可触发时钟拉伸的事件时，拉伸 SCL 时钟。SCL 时钟拉伸的原因可见 I2C_STRETCH_CAUSE。

0：禁用

1：使能

(R/W)

I2C_SLAVE_SCL_STRETCH_CLR 配置清除从机 SCL 时钟拉伸。

0：无效

1：清除

(WT)

I2C_SLAVE_BYTE_ACK_CTL_EN 配置使能从机控制 ACK 电平。

0：禁用

1：使能

(R/W)

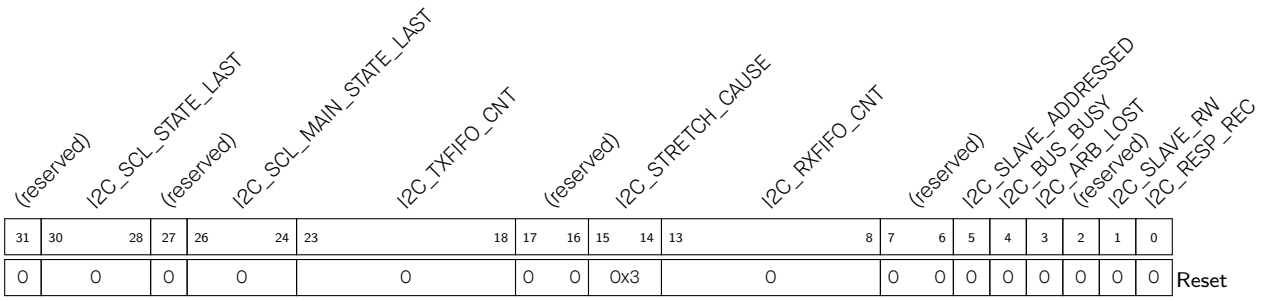
I2C_SLAVE_BYTE_ACK_LVL 从机控制 ACK 电平使能时，通过此位设置 ACK 的电平值。

0：低电平

1：高电平

(R/W)

Register 34.18. I2C_SR_REG (0x0008)



I2C_RESP_REC 表示主机模式或从机模式下接收的 ACK 电平值。

0：ACK

1：NACK

(RO)

I2C_SLAVE_RW 表示在从机模式下数据传输方向。

0：主机向从机写入数据

1：主机读取从机数据

(RO)

I2C_ARB_LOST 表示 I2C 控制器是否控制 SCL 线。

0：仲裁未丢失

1：仲裁丢失

(RO)

I2C_BUS_BUSY 表示 I2C 总线状态。

0：I2C 总线处于空闲状态

1：I2C 总线正在传输数据

(RO)

I2C_SLAVE_ADDRESSED 表示主机发送的地址是否与从机的地址相同。

仅当模块被配置为 I2C 从机时有效。

0：不相同

1：相同

(RO)

I2C_RXFIFO_CNT 表示 RAM 接收数据的字节数。(RO)

I2C_STRETCH_CAUSE 表示从机模式下 SCL 时钟拉伸的原因。

0：主机开始读取数据时拉伸 SCL 时钟

1：从机模式下 I2C TX FIFO 读空时拉伸 SCL 时钟

2：从机模式下 I2C RX FIFO 写满时拉伸 SCL 时钟

3：无效

(RO)

I2C_TXFIFO_CNT 表示需发送数据的字节数。(RO)

见下页...

Register 34.18. I2C_SR_REG (0x0008)

... 接上页

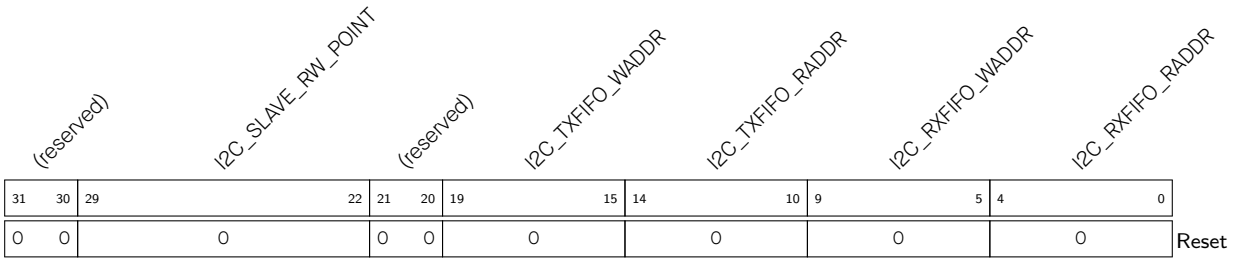
I2C_SCL_MAIN_STATE_LAST 表示 I2C 控制器状态机的状态。

0：空闲
1：地址偏移
2：ACK 地址
3：接收数据
4：发送数据
5：发送 ACK
6：等待 ACK
7：无效
(RO)

I2C_SCL_STATE_LAST 表示用于生成 SCL 的状态机状态。

0：空闲
1：开始
2：下降沿
3：低电平
4：上升沿
5：高电平
6：停止
7：无效
(RO)

Register 34.19. I2C_FIFO_ST_REG (0x0014)



I2C_RXFIFO_RADDR 表示 APB 总线读 RX FIFO 的偏移地址。(RO)

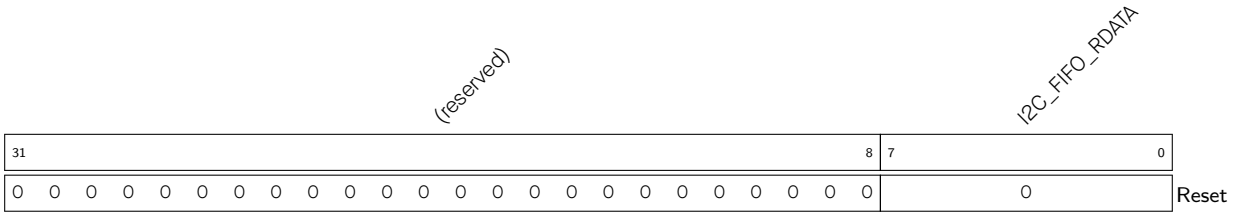
I2C_RXFIFO_WADDR 表示 I2C 控制器接收数据和写 RX FIFO 的偏移地址。(RO)

I2C_TXFIFO_RADDR 表示 I2C 控制器读 TX FIFO 的偏移地址。(RO)

I2C_TXFIFO_WADDR 表示 APB 总线写 TX FIFO 的偏移地址。(RO)

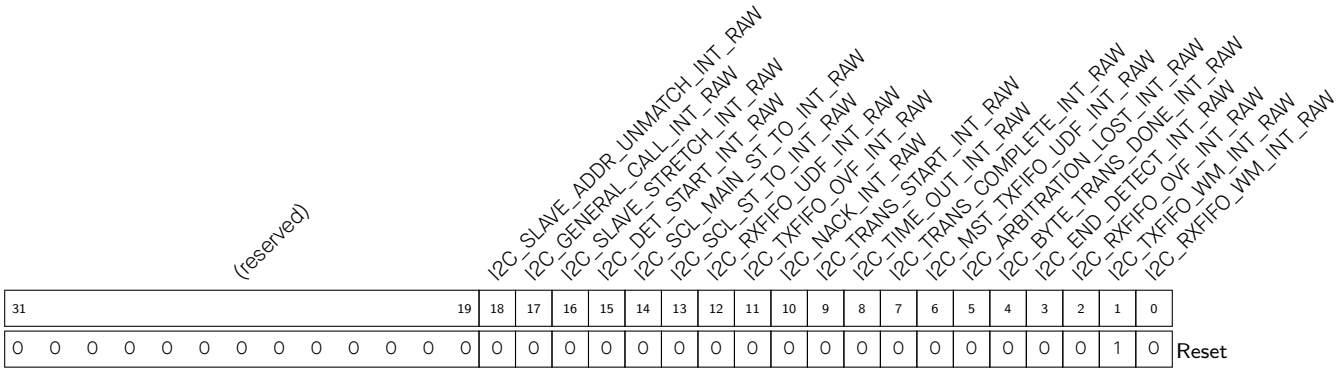
I2C_SLAVE_RW_POINT 表示在 I2C 从机模式下由 I2C 主机寻址的 I2C 从机 RAM 的偏移地址。(RO)

Register 34.20. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA 表示 RX FIFO 读取数据的值。(RO)

Register 34.21. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW I2C_RXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW I2C_TXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW I2C_RXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_END_DETECT_INT_RAW I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW I2C_BYTE_TRANS_DONE_INT 的原始中断状态。(R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW I2C_ARBITRATION_LOST_INT 的原始中断状态。(R/SS/WTC)

I2C_MST_TXFIFO_UDF_INT_RAW I2C_MST_TXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

I2C_TIME_OUT_INT_RAW I2C_TIME_OUT_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_START_INT_RAW I2C_TRANS_START_INT 的原始中断状态。(R/SS/WTC)

I2C_NACK_INT_RAW I2C_NACK_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW I2C_TXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW I2C_RXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW I2C_SCL_ST_TO_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW I2C_SCL_MAIN_ST_TO_INT 的原始中断状态。(R/SS/WTC)

I2C_DET_START_INT_RAW I2C_DET_START_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_STRETCH_INT_RAW I2C_SLAVE_STRETCH_INT 的原始中断状态。(R/SS/WTC)

I2C_GENERAL_CALL_INT_RAW I2C_GENARAL_CALL_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_ADDR_UNMATCH_INT_RAW I2C_SLAVE_ADDR_UNMATCH_INT 的原始中断状态。(R/SS/WTC)

Register 34.22. I2C_INT_CLR_REG (0x0024)

Bit	31..19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	(reserved)	I2C_SLAVE_ADDR_UNMATCH_INT_CLR	I2C_GENERAL_CALL_INT_CLR	I2C_SLAVE_STRETCH_INT_CLR	I2C_DET_START_INT_CLR	I2C_SCL_MAIN_ST_TO_INT_CLR	I2C_SCL_ST_TO_INT_CLR	I2C_RXFIFO_UDF_INT_CLR	I2C_TXFIFO_OVF_INT_CLR	I2C_NACK_INT_CLR	I2C_TRANS_START_INT_CLR	I2C_TIME_OUT_INT_CLR	I2C_TRANS_COMPLETE_INT_CLR	I2C_MST_TXFIFO_UDF_INT_CLR	I2C_ARBITRATION_LOST_INT_CLR	I2C_BYTE_TRANS_DONE_INT_CLR	I2C_END_DETECT_INT_CLR	I2C_RXFIFO_OVF_INT_CLR	I2C_TXFIFO_WM_INT_CLR	I2C_RXFIFO_WM_INT_CLR
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2C_RXFIFO_WM_INT_CLR 写 1 清除 I2C_RXFIFO_WM_INT 中断。(WT)

I2C_TXFIFO_WM_INT_CLR 写 1 清除 I2C_TXFIFO_WM_INT 中断。(WT)

I2C_RXFIFO_OVF_INT_CLR 写 1 清除 I2C_RXFIFO_OVF_INT 中断。(WT)

I2C_END_DETECT_INT_CLR 写 1 清除 I2C_END_DETECT_INT 中断。(WT)

I2C_BYTE_TRANS_DONE_INT_CLR 写 1 清除 I2C_BYTE_TRANS_DONE_INT 中断。(WT)

I2C_ARBITRATION_LOST_INT_CLR 写 1 清除 I2C_ARBITRATION_LOST_INT 中断。(WT)

I2C_MST_TXFIFO_UDF_INT_CLR 写 1 清除 I2C_MST_TXFIFO_UDF_INT 中断。(WT)

I2C_TRANS_COMPLETE_INT_CLR 写 1 清除 I2C_TRANS_COMPLETE_INT 中断。(WT)

I2C_TIME_OUT_INT_CLR 写 1 清除 I2C_TIME_OUT_INT 中断。(WT)

I2C_TRANS_START_INT_CLR 写 1 清除 I2C_TRANS_START_INT 中断。(WT)

I2C_NACK_INT_CLR 写 1 清除 I2C_NACK_INT 中断。(WT)

I2C_TXFIFO_OVF_INT_CLR 写 1 清除 I2C_TXFIFO_OVF_INT 中断。(WT)

I2C_RXFIFO_UDF_INT_CLR 写 1 清除 I2C_RXFIFO_UDF_INT 中断。(WT)

I2C_SCL_ST_TO_INT_CLR 写 1 清除 I2C_SCL_ST_TO_INT 中断。(WT)

I2C_SCL_MAIN_ST_TO_INT_CLR 写 1 清除 I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

I2C_DET_START_INT_CLR 写 1 清除 I2C_DET_START_INT 中断。(WT)

I2C_SLAVE_STRETCH_INT_CLR 写 1 清除 I2C_SLAVE_STRETCH_INT 中断。(WT)

I2C_GENERAL_CALL_INT_CLR 写 1 清除 I2C_GENARAL_CALL_INT 中断。(WT)

I2C_SLAVE_ADDR_UNMATCH_INT_CLR 写 1 清除 I2C_SLAVE_ADDR_UNMATCH_INT 中断。(WT)

Register 34.23. I2C_INT_ENA_REG (0x0028)

Bit	Field	Reset
31–19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	I2C_SLAVE_ADDR_UNMATCH_INT_ENA	0
17	I2C_GENERAL_CALL_INT_ENA	0
16	I2C_SLAVE_STRETCH_INT_ENA	0
15	I2C_DET_START_INT_ENA	0
14	I2C_SCL_MAIN_ST_TO_INT_ENA	0
13	I2C_SCL_ST_TO_INT_ENA	0
12	I2C_RXFIFO_UDF_INT_ENA	0
11	I2C_TXFIFO_OVF_INT_ENA	0
10	I2C_NACK_INT_ENA	0
9	I2C_TRANS_START_INT_ENA	0
8	I2C_TIME_OUT_INT_ENA	0
7	I2C_TRANS_COMPLETE_INT_ENA	0
6	I2C_MST_TXFIFO_UDF_INT_ENA	0
5	I2C_ARBITRATION_LOST_INT_ENA	0
4	I2C_BYTE_TRANS_DONE_INT_ENA	0
3	I2C_END_DETECT_INT_ENA	0
2	I2C_RXFIFO_OVF_INT_ENA	0
1	I2C_TXFIFO_WM_INT_ENA	0
0	I2C_RXFIFO_WM_INT_ENA	0

I2C_RXFIFO_WM_INT_ENA 写 1 使能 I2C_RXFIFO_WM_INT 中断。(R/W)

I2C_TXFIFO_WM_INT_ENA 写 1 使能 I2C_TXFIFO_WM_INT 中断。(R/W)

I2C_RXFIFO_OVF_INT_ENA 写 1 使能 I2C_RXFIFO_OVF_INT 中断。(R/W)

I2C_END_DETECT_INT_ENA 写 1 使能 I2C_END_DETECT_INT 中断。(R/W)

I2C_BYTE_TRANS_DONE_INT_ENA 写 1 使能 I2C_BYTE_TRANS_DONE_INT 中断。(R/W)

I2C_ARBITRATION_LOST_INT_ENA 写 1 使能 I2C_ARBITRATION_LOST_INT 中断。(R/W)

I2C_MST_TXFIFO_UDF_INT_ENA 写 1 使能 I2C_MST_TXFIFO_UDF_INT 中断。(R/W)

I2C_TRANS_COMPLETE_INT_ENA 写 1 使能 I2C_TRANS_COMPLETE_INT 中断。(R/W)

I2C_TIME_OUT_INT_ENA 写 1 使能 I2C_TIME_OUT_INT 中断。(R/W)

I2C_TRANS_START_INT_ENA 写 1 使能 I2C_TRANS_START_INT 中断。(R/W)

I2C_NACK_INT_ENA 写 1 使能 I2C_NACK_INT 中断。(R/W)

I2C_TXFIFO_OVF_INT_ENA 写 1 使能 I2C_TXFIFO_OVF_INT 中断。(R/W)

I2C_RXFIFO_UDF_INT_ENA 写 1 使能 I2C_RXFIFO_UDF_INT 中断。(R/W)

I2C_SCL_ST_TO_INT_ENA 写 1 使能 I2C_SCL_ST_TO_INT 中断。(R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA 写 1 使能 I2C_SCL_MAIN_ST_TO_INT 中断。(R/W)

I2C_DET_START_INT_ENA 写 1 使能 I2C_DET_START_INT 中断。(R/W)

I2C_SLAVE_STRETCH_INT_ENA 写 1 使能 I2C_SLAVE_STRETCH_INT 中断。(R/W)

I2C_GENERAL_CALL_INT_ENA 写 1 使能 I2C_GENARAL_CALL_INT 中断。(R/W)

I2C_SLAVE_ADDR_UNMATCH_INT_ENA 写 1 使能 I2C_SLAVE_ADDR_UNMATCH_INT 中断。(R/W)

Register 34.24. I2C_INT_STATUS_REG (0x002C)

(reserved)	I2C_SLAVE_ADDR_UNMATCH_INT_ST	I2C_GENERAL_CALL_INT_ST	I2C_SLAVE_STRETCH_INT_ST	I2C_DET_START_INT_ST	I2C_SCL_MAIN_ST_TO_INT_ST	I2C_SCL_ST_TO_INT_ST	I2C_RXFIFO_UDF_INT_ST	I2C_TXFIFO_OVF_INT_ST	I2C_NACK_INT_ST	I2C_TRANS_START_INT_ST	I2C_TIME_OUT_INT_ST	I2C_TRANS_COMPLETE_INT_ST	I2C_MST_TXFIFO_UDF_INT_ST	I2C_ARBITRATION_LOST_INT_ST	I2C_BYTE_TRANS_DONE_INT_ST	I2C_END_DETECT_INT_ST	I2C_RXFIFO_OVF_INT_ST	I2C_TXFIFO_WM_INT_ST	I2C_RXFIFO_WM_INT_ST	
31 … 19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2C_RXFIFO_WM_INT_ST I2C_RXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_WM_INT_ST I2C_TXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_RXFIFO_OVF_INT_ST I2C_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

I2C_END_DETECT_INT_ST I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

I2C_BYTE_TRANS_DONE_INT_ST I2C_BYTE_TRANS_DONE_INT 的屏蔽中断状态。(RO)

I2C_ARBITRATION_LOST_INT_ST I2C_ARBITRATION_LOST_INT 的屏蔽中断状态。(RO)

I2C_MST_TXFIFO_UDF_INT_ST I2C_MST_TXFIFO_UDF_INT 的屏蔽中断状态。(RO)

I2C_TRANS_COMPLETE_INT_ST I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

I2C_TIME_OUT_INT_ST I2C_TIME_OUT_INT 的屏蔽中断状态。(RO)

I2C_TRANS_START_INT_ST I2C_TRANS_START_INT 的屏蔽中断状态。(RO)

I2C_NACK_INT_ST I2C_NACK_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_OVF_INT_ST I2C_TXFIFO_OVF_INT 的屏蔽中断状态。(RO)

I2C_RXFIFO_UDF_INT_ST I2C_RXFIFO_UDF_INT 的屏蔽中断状态。(RO)

I2C_SCL_ST_TO_INT_ST I2C_SCL_ST_TO_INT 的屏蔽中断状态。(RO)

I2C_SCL_MAIN_ST_TO_INT_ST I2C_SCL_MAIN_ST_TO_INT 的屏蔽中断状态。(RO)

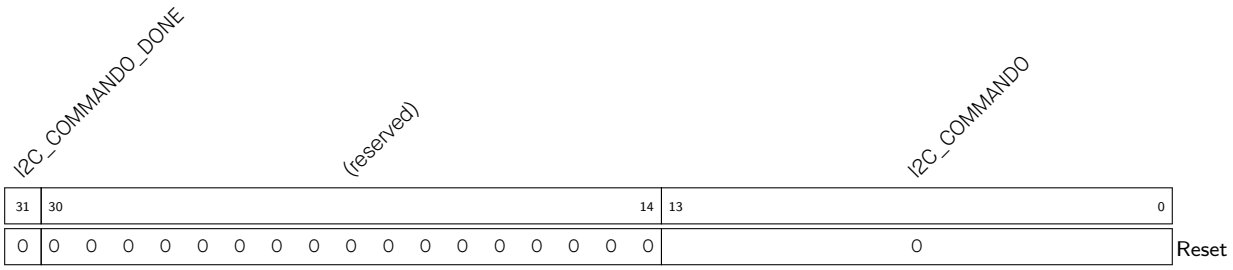
I2C_DET_START_INT_ST I2C_DET_START_INT 的屏蔽中断状态。(RO)

I2C_SLAVE_STRETCH_INT_ST I2C_SLAVE_STRETCH_INT 的屏蔽中断状态。(RO)

I2C_GENERAL_CALL_INT_ST I2C_GENARAL_CALL_INT 的屏蔽中断状态。(RO)

I2C_SLAVE_ADDR_UNMATCH_INT_ST I2C_SLAVE_ADDR_UNMATCH_INT 的屏蔽中断状态。(RO)

Register 34.25. I2C_COMD0_REG (0x0058)



I2C_COMMAND0 配置命令寄存器 0 的内容。

命令寄存器包含三个部分：

op_code 表示命令：

1：WRITE

2：STOP

3：READ

4：END

6：RSTART

Byte_num 表示需要发送或接收数据的字节数。

ack_check_en, ack_exp 和 ack 用于控制 ACK 位。详细信息，请参阅 I2C CMD 结构 34.4-2 。

(R/W)

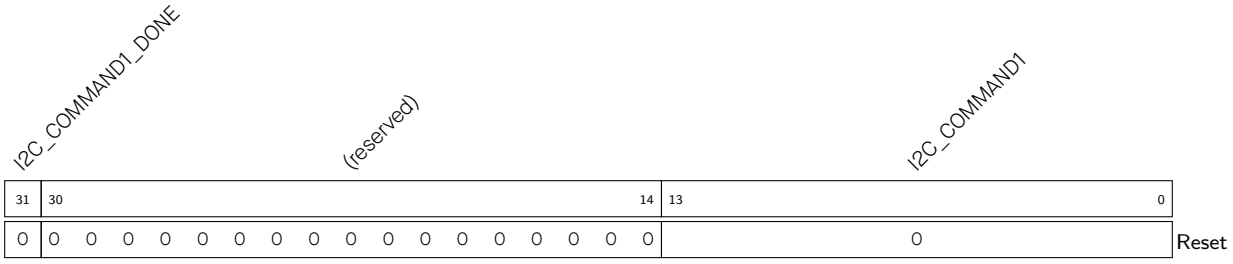
I2C_COMMAND0_DONE 表示在 I2C 主机模式下，命令 0 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.26. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 配置命令寄存器 1 的内容。具体可参考 I2C_COMMAND0。(R/W)

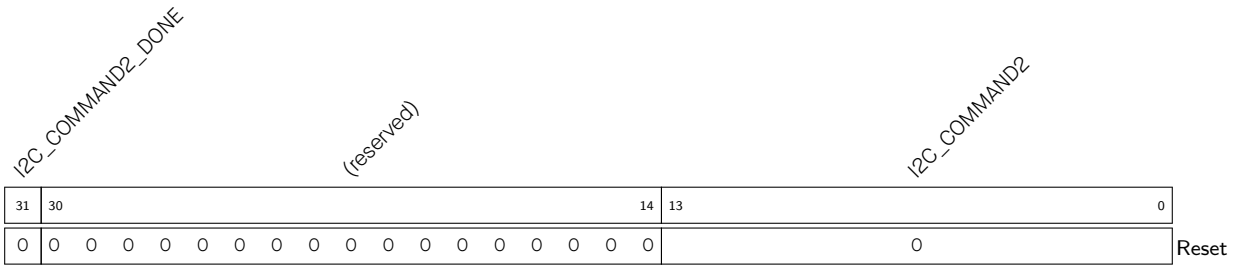
I2C_COMMAND1_DONE 表示在 I2C 主机模式下，命令 1 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.27. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 配置命令寄存器 2 的内容。具体可参考 I2C_COMMAND0。(R/W)

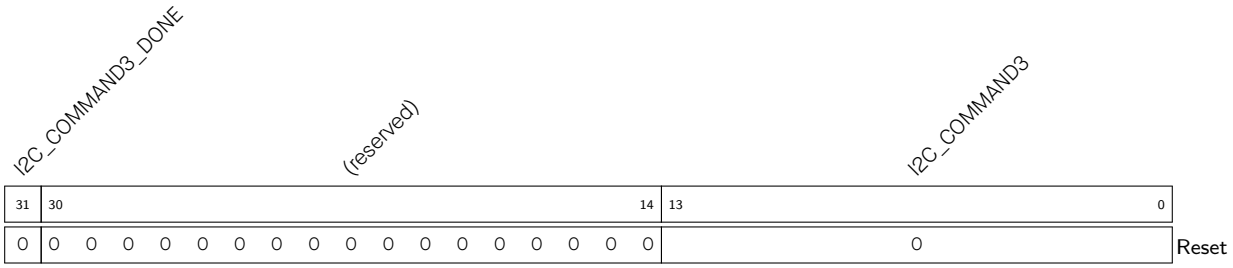
I2C_COMMAND2_DONE 表示在 I2C 主机模式下，命令 2 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.28. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 配置命令寄存器 3 的内容。具体可参考 I2C_COMMAND0。(R/W)

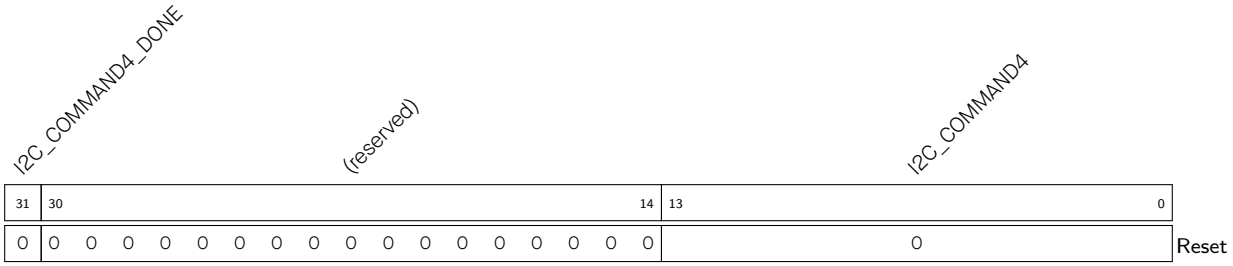
I2C_COMMAND3_DONE 表示在 I2C 主机模式下，命令 3 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.29. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 配置命令寄存器 4 的内容。具体可参考 I2C_COMMAND0。(R/W)

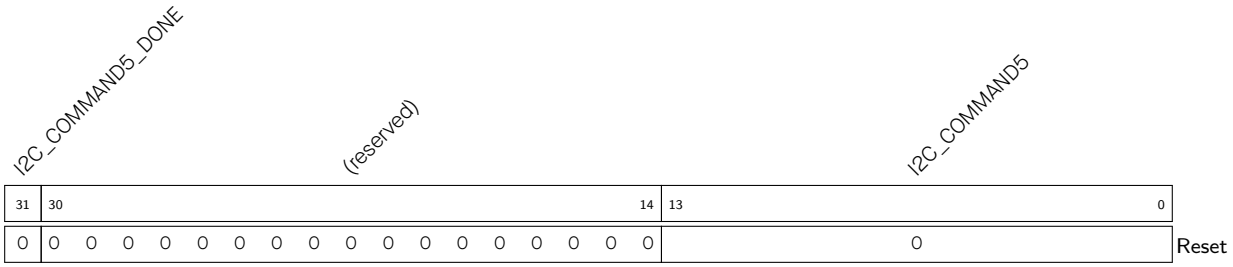
I2C_COMMAND4_DONE 表示在 I2C 主机模式下，命令 4 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.30. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 配置命令寄存器 5 的内容。具体可参考 I2C_COMMAND0。(R/W)

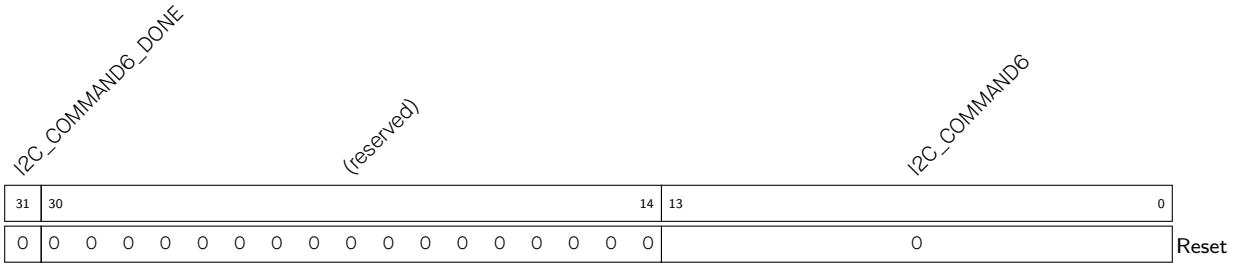
I2C_COMMAND5_DONE 表示在 I2C 主机模式下，命令 5 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.31. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 配置命令寄存器 6 的内容。具体可参考 I2C_COMMAND0。(R/W)

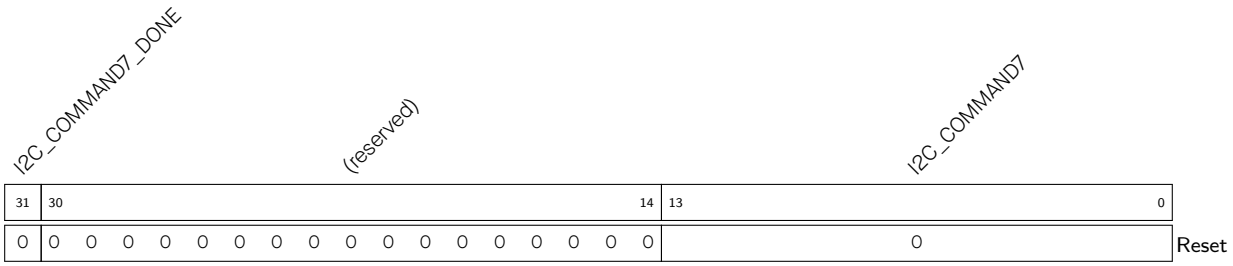
I2C_COMMAND6_DONE 表示在 I2C 主机模式下，命令 6 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.32. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 配置命令寄存器 7 的内容。具体可参考 I2C_COMMAND0。(R/W)

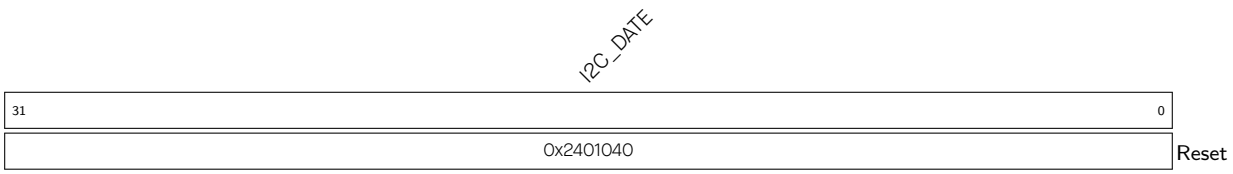
I2C_COMMAND7_DONE 表示在 I2C 主机模式下，命令 7 是否已完成。

0：未完成

1：已完成

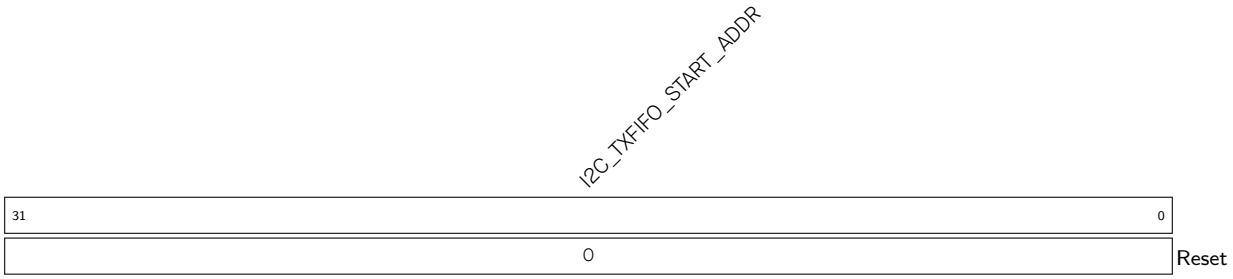
(R/W/SS)

Register 34.33. I2C_DATE_REG (0x00F8)



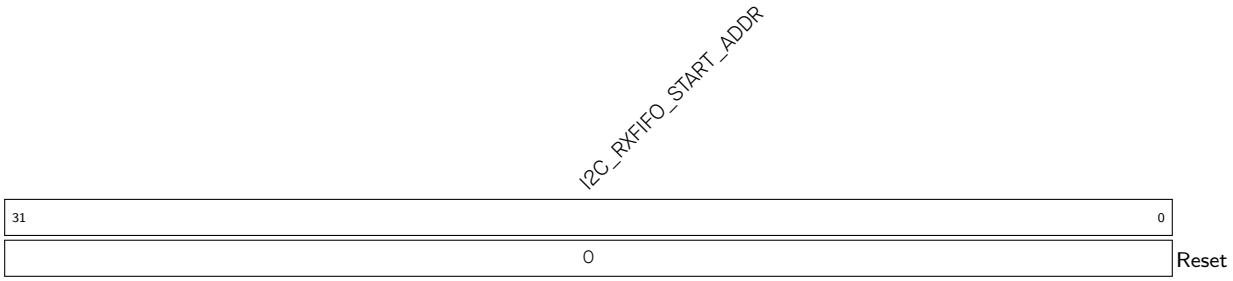
I2C_DATE 版本控制寄存器。(R/W)

Register 34.34. I2C_TXFIFO_START_ADDR_REG (0x0100)



I2C_TXFIFO_START_ADDR 表示 I2C TX FIFO 首地址。(RO)

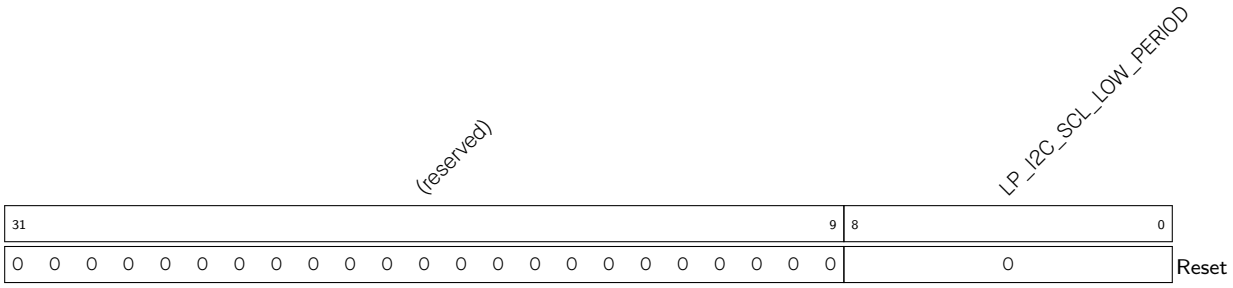
Register 34.35. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR 表示 I2C RX FIFO 首地址。(RO)

34.9.2 LP_I2C 寄存器

Register 34.36. LP_I2C_SCL_LOW_PERIOD_REG (0x0000)

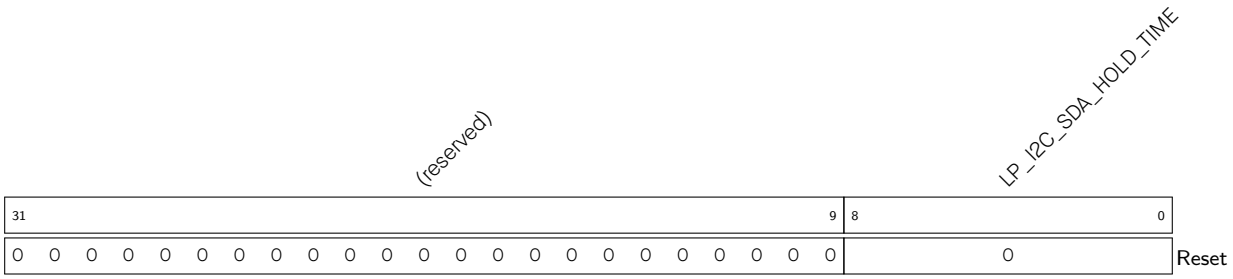


LP_I2C_SCL_LOW_PERIOD 配置主机模式下 SCL 低电平的保持时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.37. LP_I2C_SDA_HOLD_REG (0x0030)

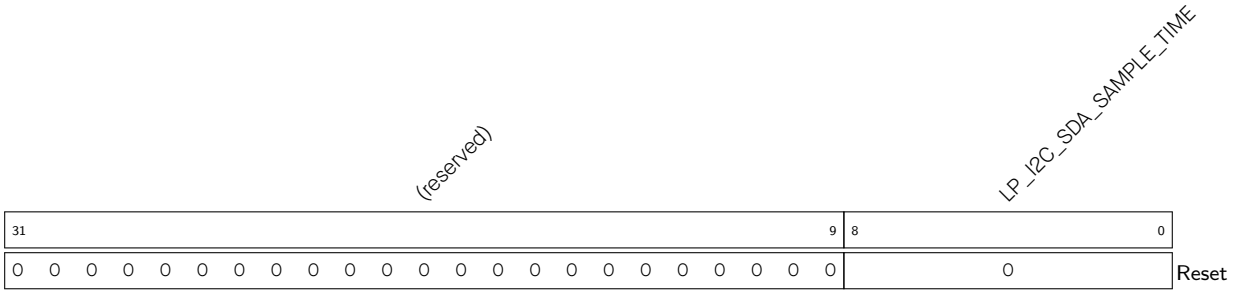


LP_I2C_SDA_HOLD_TIME 配置 SCL 下降沿后的数据保持时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.38. LP_I2C_SDA_SAMPLE_REG (0x0034)

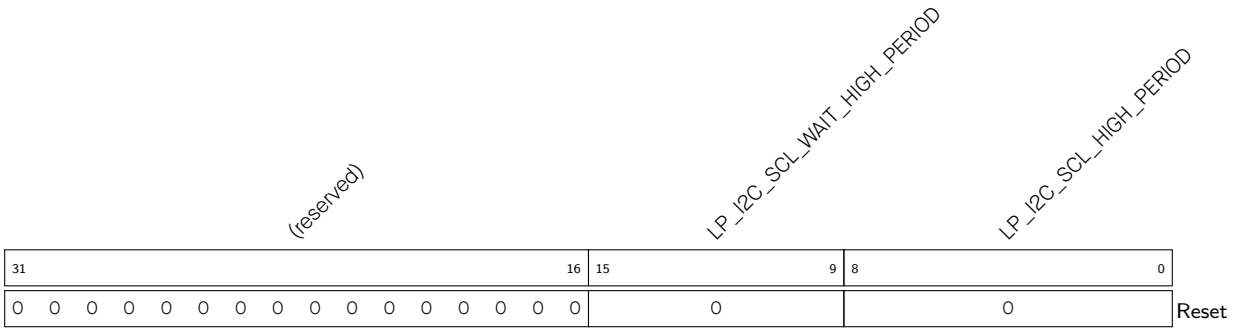


LP_I2C_SDA_SAMPLE_TIME 配置采样 SDA 的时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.39. LP_I2C_SCL_HIGH_PERIOD_REG (0x0038)



LP_I2C_SCL_HIGH_PERIOD 配置 SCL 在主机模式下保持高电平的时间。

单位：I2C_SCLK 时钟周期数

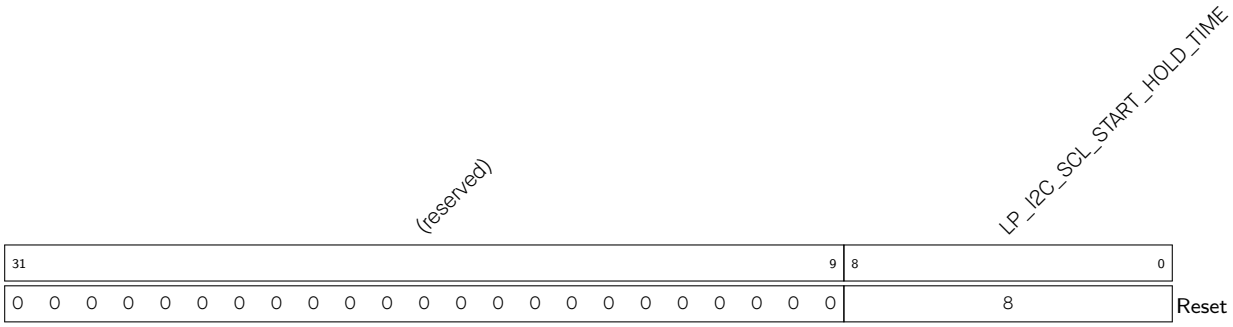
(R/W)

LP_I2C_SCL_WAIT_HIGH_PERIOD 配置 SCL_FSM 等待 SCL 在主机模式下翻转至高电平的时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.40. LP_I2C_SCL_START_HOLD_REG (0x0040)

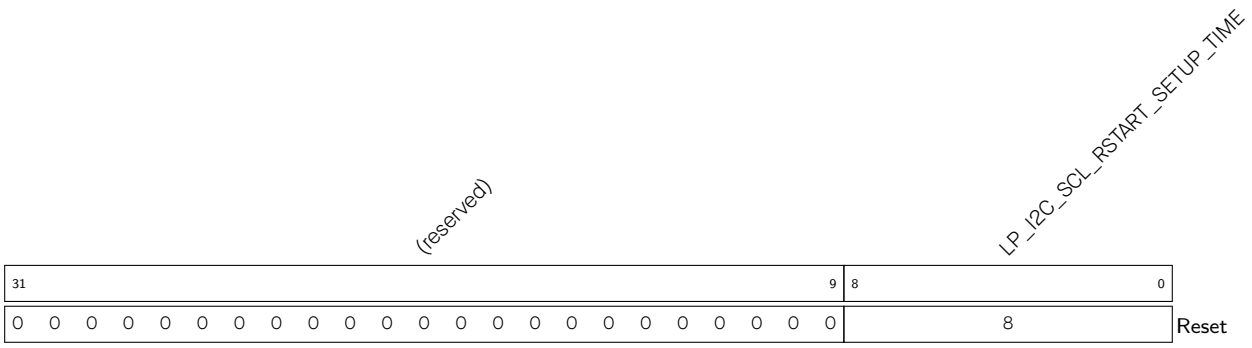


LP_I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.41. LP_I2C_SCL_RSTART_SETUP_REG (0x0044)

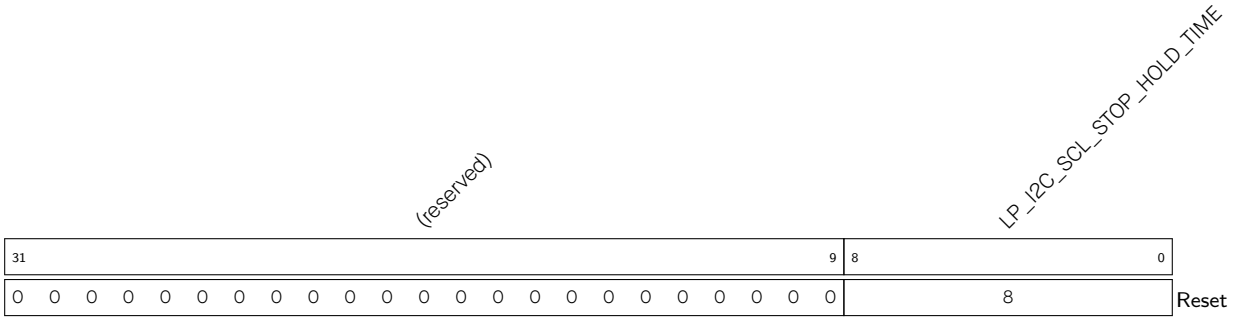


LP_I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.42. LP_I2C_SCL_STOP_HOLD_REG (0x0048)

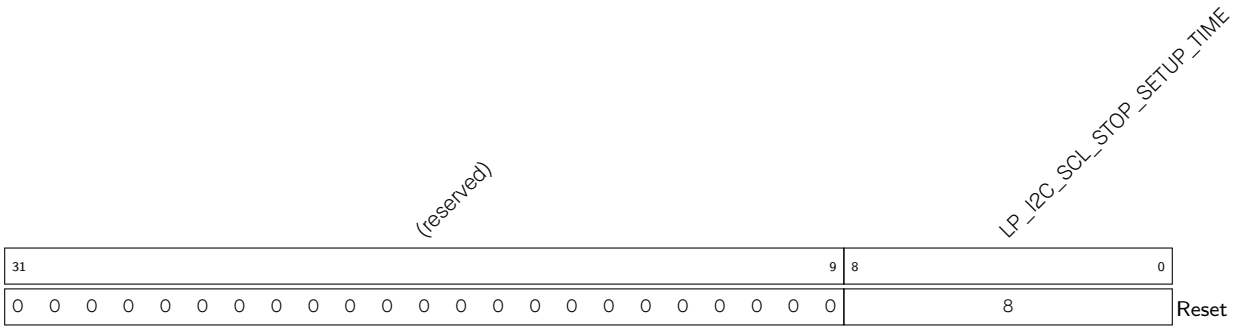


LP_I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.43. LP_I2C_SCL_STOP_SETUP_REG (0x004C)

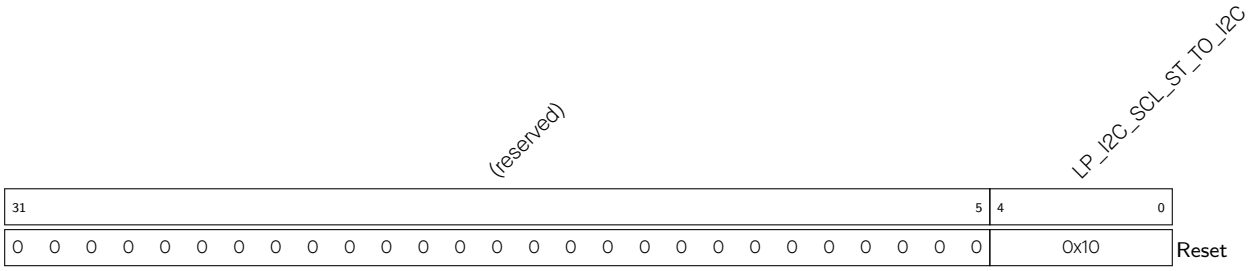


LP_I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.44. LP_I2C_SCL_ST_TIME_OUT_REG (0x0078)

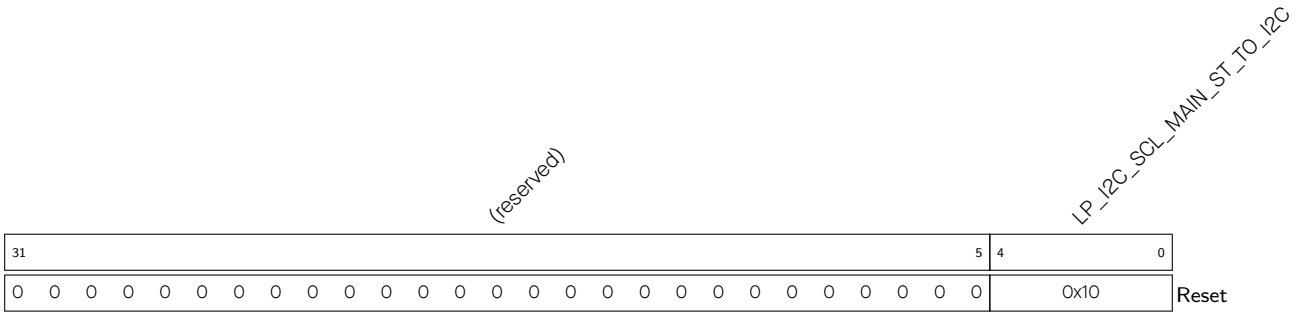


LP_I2C_SCL_ST_TO_I2C 配置 SCL_FSM 状态保持不变的最大时间，不能大于 23。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.45. LP_I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

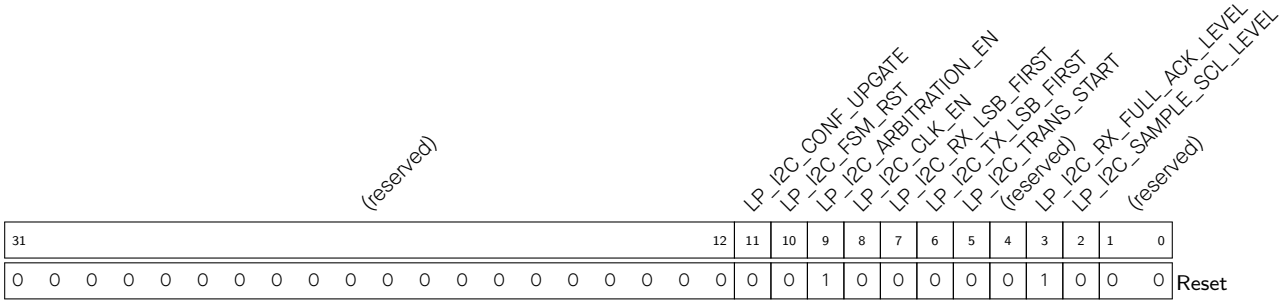


LP_I2C_SCL_MAIN_ST_TO_I2C 配置 SCL_MAIN_FSM 状态保持不变的最大时间，不能大于 23。

单位：I2C_SCLK 时钟周期数

(R/W)

Register 34.46. LP_I2C_CTR_REG (0x0004)



LP_I2C_SAMPLE_SCL_LEVEL 配置 SDA 的采样模式。

0: SCL 为高电平时采样 SDA 数据

1: SCL 为低电平时采样 SDA 数据

(R/W)

LP_I2C_RX_FULL_ACK_LEVEL 用于配置主机在 rx_fifo_cnt 达到阈值时需发送的 ACK 电平值。(R/W)

LP_I2C_TRANS_START 配置从机是否开始发送 TX FIFO 中的数据。

0：无效

1：开始

(WT)

LP_I2C_TX_LSB_FIRST 控制待发送数据的发送顺序。

0：从最高有效位开始发送数据

1：从最低有效位开始发送数据

(R/W)

LP_I2C_RX_LSB_FIRST 控制接收数据的存储顺序。

0：从最高有效位开始接收数据

1：从最低有效位开始接收数据

(R/W)

LP_I2C_CLK_EN 门控寄存器的时钟信号。

0：仅当通过软件读写寄存器时支持时钟

1：强制寄存器时钟开启。

(R/W)

Continued on the next page...

Register 34.46. I2C_CTR_REG (0x0004)

Continued from the previous page...

LP_I2C_ARBITRATION_EN 配置使能 I2C 总线仲裁检测。

0：无效

1：使能

(R/W)

LP_I2C_FSM_RST 配置复位 SCL_FSM。

0：无效

1：复位

(WT)

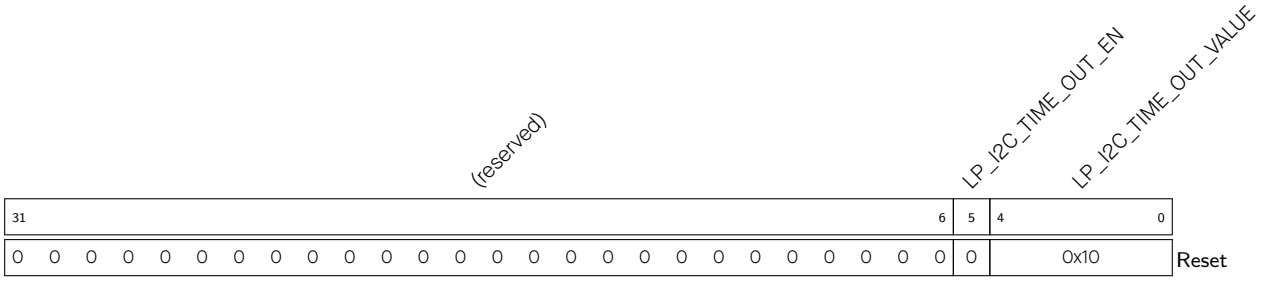
LP_I2C_CONF_UPGATE 配置此位实现同步。

0：无效

1：同步

(WT)

Register 34.47. LP_I2C_TO_REG (0x000C)



LP_I2C_TIME_OUT_VALUE 配置 SCL 在高电平或低电平停滞的超时阈值周期。实际周期是 2^(LP_I2C_TIME_OUT_VALUE+1)–1。

单位：I2C_SCLK 时钟周期数

(R/W)

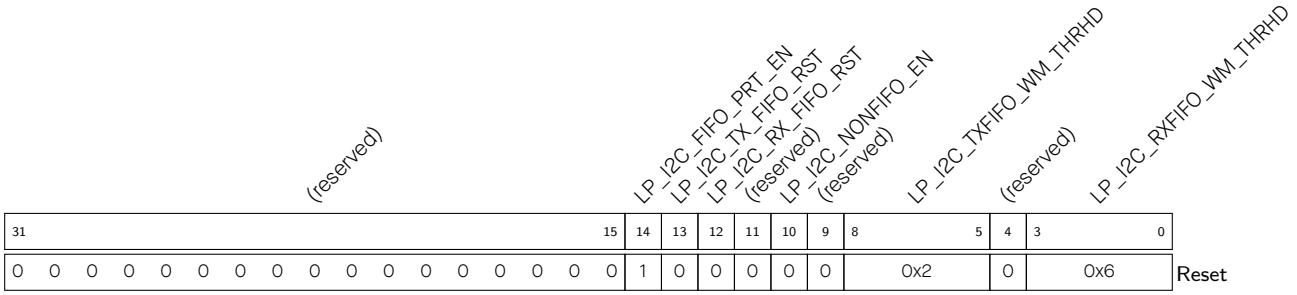
LP_I2C_TIME_OUT_EN 配置使能超时控制。

0：无效

1：使能

(R/W)

Register 34.48. LP_I2C_FIFO_CONF_REG (0x0018)



LP_I2C_RXFIFO_WM_THRHD 配置在直接访问模式下，RX FIFO 的水标阈值。当 LP_I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 LP_I2C_RXFIFO_WM_THRHD[4:0]，LP_I2C_RXFIFO_WM_INT_RAW 位有效。(R/W)

LP_I2C_TXFIFO_WM_THRHD 配置在直接访问模式下，TX FIFO 的水标阈值。当 LP_I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 LP_I2C_TXFIFO_WM_THRHD[4:0] 时，LP_I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

LP_I2C_NONFIFO_EN 配置使能 APB 直接访问。(R/W)

LP_I2C_RX_FIFO_RST 配置复位 RX FIFO。

0：无效

1：复位

(R/W)

LP_I2C_TX_FIFO_RST 配置复位 TX FIFO。

0：无效

1：复位

(R/W)

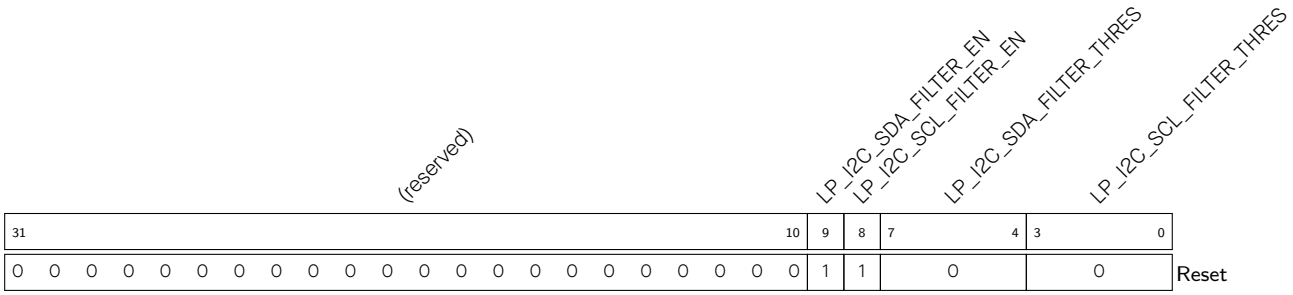
LP_I2C_FIFO_PRT_EN 配置在直接访问模式下是否使能 FIFO 指针。该位控制有效位以及 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的中断。

0：无效

1：使能

(R/W)

Register 34.49. LP_I2C_FILTER_CFG_REG (0x0050)



LP_I2C_SCL_FILTER_THRES 配置 SCL 过滤脉冲宽度的阈值。当 SCL 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK 时钟周期数

(R/W)

LP_I2C_SDA_FILTER_THRES 配置 SDA 过滤脉冲宽度的阈值。当 SDA 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK 时钟周期数

(R/W)

LP_I2C_SCL_FILTER_EN 配置使能 SCL 的滤波功能。

0：无效

1：复位

(R/W)

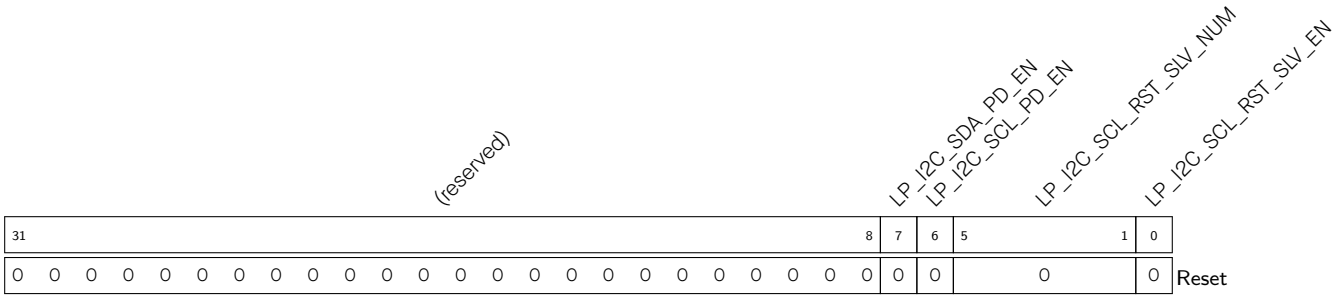
LP_I2C_SDA_FILTER_EN 配置使能 SDA 的滤波功能。

0：无效

1：复位

(R/W)

Register 34.50. LP_I2C_SCL_SP_CONF_REG (0x0080)



LP_I2C_SCL_RST_SLV_EN 在 I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 LP_I2C_SCL_RST_SLV_NUM. (R/W/SC)

LP_I2C_SCL_RST_SLV_NUM 配置 I2C 主机模式下生成的 SCL 脉冲。

仅在 LP_I2C_SCL_RST_SLV_EN 为 1 时有效。

单位：I2C_SCLK 时钟周期数

(R/W)

LP_I2C_SCL_PD_EN 配置降低 I2C SCL 输出功耗。

0：正常功耗

1：不工作，降低功耗

仅在 LP_I2C_SCL_FORCE_OUT 为 1 时有效。

(R/W)

LP_I2C_SDA_PD_EN 配置降低 I2C SDA 输出功耗。

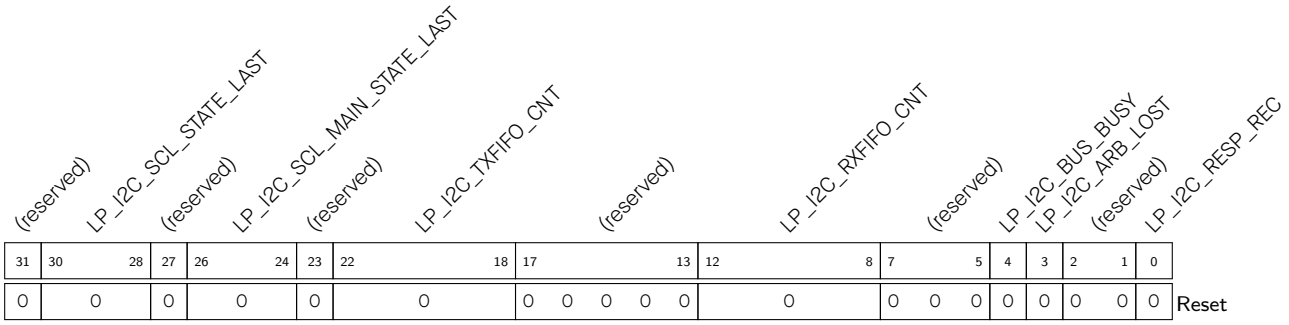
0：正常功耗

1：不工作，降低功耗

仅在 LP_I2C_SDA_FORCE_OUT 为 1 时有效。

(R/W) deleted these two registers based on JLW's request.

Register 34.51. LP_I2C_SR_REG (0x0008)



LP_I2C_RESP_REC 表示主机模式或从机模式下接收的 ACK 电平值。

0: ACK

1: NACK

(RO)

LP_I2C_ARB_LOST 表示 I2C 控制器是否控制 SCL 线。

0: 仲裁未丢失

1: 仲裁丢失

(RO)

LP_I2C_BUS_BUSY 表示 I2C 总线状态。

0：I2C 总线处于空闲状态

1：I2C 总线正在传输数据

(RO)

LP_I2C_RXFIFO_CNT 表示 RAM 接收数据的字节数。(RO)

LP_I2C_TXFIFO_CNT 表示需发送数据的字节数。(RO)

LP_I2C_SCL_MAIN_STATE_LAST 表示 I2C 控制器状态机的状态。

0：空闲

1：地址偏移

2：ACK 地址

3：接收数据

4：发送数据

5：发送 ACK

6：等待 ACK

(RO)

LP_I2C_SCL_STATE_LAST 表示用于生成 SCL 的状态机状态。

0：空闲

1：开始

2：下降沿

3：低电平

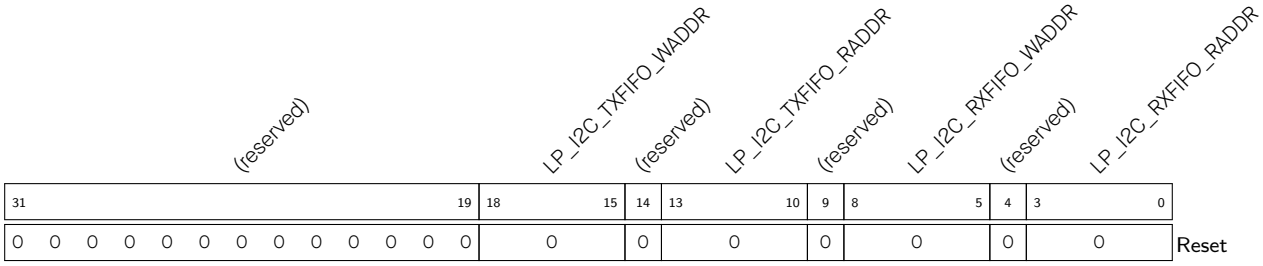
4：上升沿

5：高电平

6：停止

(RO)

Register 34.52. LP_I2C_FIFO_ST_REG (0x0014)



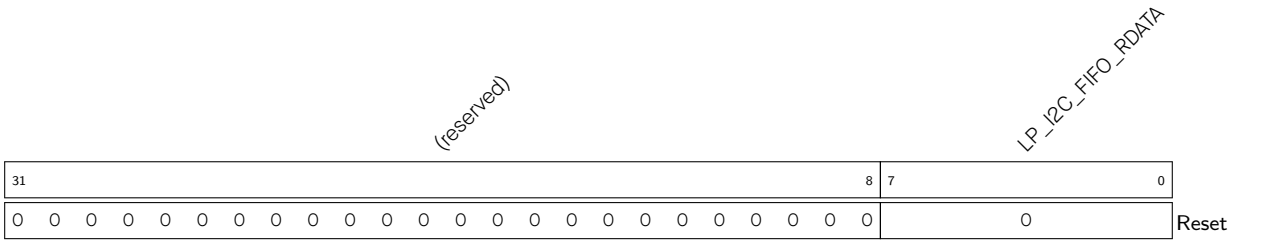
LP_I2C_RXFIFO_RADDR 表示 APB 总线读 RX FIFO 的偏移地址。(RO)

LP_I2C_RXFIFO_WADDR 表示 I2C 控制器接收数据和写 RX FIFO 的偏移地址。(RO)

LP_I2C_TXFIFO_RADDR 表示 I2C 控制器读 TX FIFO 的偏移地址。(RO)

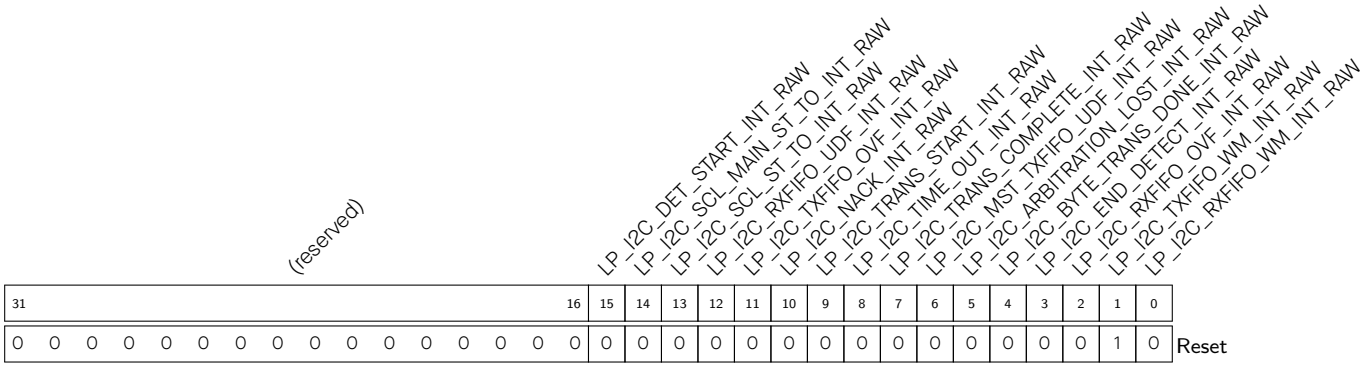
LP_I2C_TXFIFO_WADDR 表示 APB 总线写 TX FIFO 的偏移地址。(RO)

Register 34.53. LP_I2C_DATA_REG (0x001C)



LP_I2C_FIFO_RDATA 表示 RX FIFO 读取数据的值。(RO)

Register 34.54. LP_I2C_INT_RAW_REG (0x0020)



LP_I2C_RXFIFO_WM_INT_RAW LP_I2C_RXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TXFIFO_WM_INT_RAW LP_I2C_TXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_RXFIFO_OVF_INT_RAW LP_I2C_RXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_END_DETECT_INT_RAW LP_I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_BYTE_TRANS_DONE_INT_RAW LP_I2C_BYTE_TRANS_DONE_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_ARBITRATION_LOST_INT_RAW LP_I2C_ARBITRATION_LOST_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_MST_TXFIFO_UDF_INT_RAW LP_I2C_MST_TXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TRANS_COMPLETE_INT_RAW LP_I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TIME_OUT_INT_RAW LP_I2C_TIME_OUT_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TRANS_START_INT_RAW LP_I2C_TRANS_START_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_NACK_INT_RAW LP_I2C_NACK_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TXFIFO_OVF_INT_RAW LP_I2C_TXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

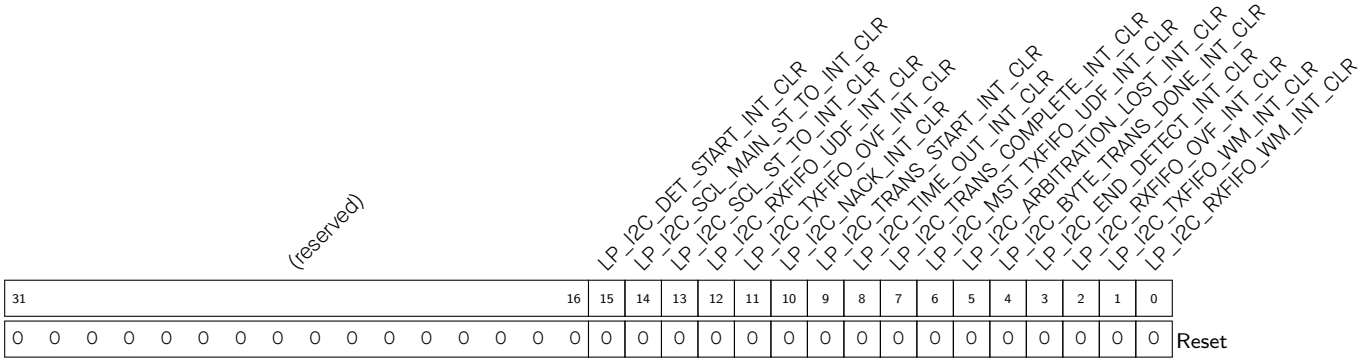
LP_I2C_RXFIFO_UDF_INT_RAW LP_I2C_RXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_SCL_ST_TO_INT_RAW LP_I2C_SCL_ST_TO_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_SCL_MAIN_ST_TO_INT_RAW LP_I2C_SCL_MAIN_ST_TO_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_DET_START_INT_RAW LP_I2C_DET_START_INT 的原始中断状态。(R/SS/WTC)

Register 34.55. LP_I2C_INT_CLR_REG (0x0024)



LP_I2C_RXFIFO_WM_INT_CLR 写 1 清除 LP_I2C_RXFIFO_WM_INT 中断。(WT)

LP_I2C_TXFIFO_WM_INT_CLR 写 1 清除 LP_I2C_TXFIFO_WM_INT 中断。(WT)

LP_I2C_RXFIFO_OVF_INT_CLR 写 1 清除 LP_I2C_RXFIFO_OVF_INT 中断。(WT)

LP_I2C_END_DETECT_INT_CLR 写 1 清除 LP_I2C_END_DETECT_INT 中断。(WT)

LP_I2C_BYTE_TRANS_DONE_INT_CLR 写 1 清除 LP_I2C_BYTE_TRANS_DONE_INT 中断。(WT)

LP_I2C_ARBITRATION_LOST_INT_CLR 写 1 清除 LP_I2C_ARBITRATION_LOST_INT 中断。(WT)

LP_I2C_MST_TXFIFO_UDF_INT_CLR 写 1 清除 LP_I2C_MST_TXFIFO_UDF_INT 中断。(WT)

LP_I2C_TRANS_COMPLETE_INT_CLR 写 1 清除 LP_I2C_TRANS_COMPLETE_INT 中断。(WT)

LP_I2C_TIME_OUT_INT_CLR 写 1 清除 LP_I2C_TIME_OUT_INT 中断。(WT)

LP_I2C_TRANS_START_INT_CLR 写 1 清除 LP_I2C_TRANS_START_INT 中断。(WT)

LP_I2C_NACK_INT_CLR 写 1 清除 LP_I2C_NACK_INT 中断。(WT)

LP_I2C_TXFIFO_OVF_INT_CLR 写 1 清除 LP_I2C_TXFIFO_OVF_INT 中断。(WT)

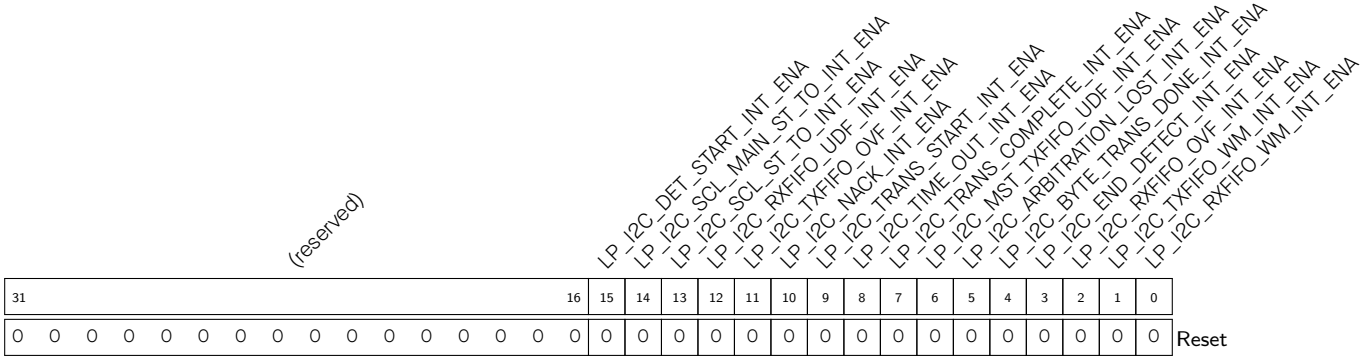
LP_I2C_RXFIFO_UDF_INT_CLR 写 1 清除 LP_I2C_RXFIFO_UDF_INT 中断。(WT)

LP_I2C_SCL_ST_TO_INT_CLR 写 1 清除 LP_I2C_SCL_ST_TO_INT 中断。(WT)

LP_I2C_SCL_MAIN_ST_TO_INT_CLR 写 1 清除 LP_I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

LP_I2C_DET_START_INT_CLR 写 1 清除 LP_I2C_DET_START_INT 中断。(WT)

Register 34.56. LP_I2C_INT_ENA_REG (0x0028)



LP_I2C_RXFIFO_WM_INT_ENA 写 1 使能 LP_I2C_RXFIFO_WM_INT 中断。(R/W)

LP_I2C_TXFIFO_WM_INT_ENA 写 1 使能 LP_I2C_TXFIFO_WM_INT 中断。(R/W)

LP_I2C_RXFIFO_OVF_INT_ENA 写 1 使能 LP_I2C_RXFIFO_OVF_INT 中断。(R/W)

LP_I2C_END_DETECT_INT_ENA 写 1 使能 LP_I2C_END_DETECT_INT 中断。(R/W)

LP_I2C_BYTE_TRANS_DONE_INT_ENA 写 1 使能 LP_I2C_BYTE_TRANS_DONE_INT 中断。(R/W)

LP_I2C_ARBITRATION_LOST_INT_ENA 写 1 使能 LP_I2C_ARBITRATION_LOST_INT 中断。(R/W)

LP_I2C_MST_TXFIFO_UDF_INT_ENA 写 1 使能 LP_I2C_MST_TXFIFO_UDF_INT 中断。(R/W)

LP_I2C_TRANS_COMPLETE_INT_ENA 写 1 使能 LP_I2C_TRANS_COMPLETE_INT 中断。(R/W)

LP_I2C_TIME_OUT_INT_ENA 写 1 使能 LP_I2C_TIME_OUT_INT 中断。(R/W)

LP_I2C_TRANS_START_INT_ENA 写 1 使能 LP_I2C_TRANS_START_INT 中断。(R/W)

LP_I2C_NACK_INT_ENA 写 1 使能 LP_I2C_NACK_INT 中断。(R/W)

LP_I2C_TXFIFO_OVF_INT_ENA 写 1 使能 LP_I2C_TXFIFO_OVF_INT 中断。(R/W)

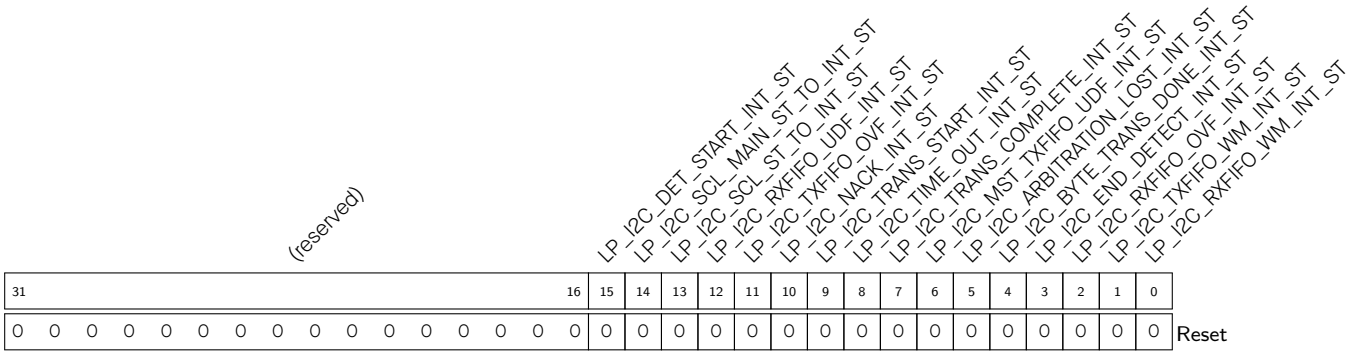
LP_I2C_RXFIFO_UDF_INT_ENA 写 1 使能 LP_I2C_RXFIFO_UDF_INT 中断。(R/W)

LP_I2C_SCL_ST_TO_INT_ENA 写 1 使能 LP_I2C_SCL_ST_TO_INT 中断。(R/W)

LP_I2C_SCL_MAIN_ST_TO_INT_ENA 写 1 使能 LP_I2C_SCL_MAIN_ST_TO_INT 中断。(R/W)

LP_I2C_DET_START_INT_ENA 写 1 使能 LP_I2C_DET_START_INT 中断。(R/W)

Register 34.57. LP_I2C_INT_STATUS_REG (0x002C)



LP_I2C_RXFIFO_WM_INT_ST LP_I2C_RXFIFO_WM_INT 的屏蔽中断状态。(RO)

LP_I2C_TXFIFO_WM_INT_ST LP_I2C_TXFIFO_WM_INT 的屏蔽中断状态。(RO)

LP_I2C_RXFIFO_OVF_INT_ST LP_I2C_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

LP_I2C_END_DETECT_INT_ST LP_I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

LP_I2C_BYTE_TRANS_DONE_INT_ST LP_I2C_BYTE_TRANS_DONE_INT 的屏蔽中断状态。(RO)

LP_I2C_ARBITRATION_LOST_INT_ST LP_I2C_ARBITRATION_LOST_INT 的屏蔽中断状态。(RO)

LP_I2C_MST_TXFIFO_UDF_INT_ST LP_I2C_MST_TXFIFO_UDF_INT 的屏蔽中断状态。(RO)

LP_I2C_TRANS_COMPLETE_INT_ST LP_I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

LP_I2C_TIME_OUT_INT_ST LP_I2C_TIME_OUT_INT 的屏蔽中断状态。(RO)

LP_I2C_TRANS_START_INT_ST LP_I2C_TRANS_START_INT 的屏蔽中断状态。(RO)

LP_I2C_NACK_INT_ST LP_I2C_NACK_INT 的屏蔽中断状态。(RO)

LP_I2C_TXFIFO_OVF_INT_ST LP_I2C_TXFIFO_OVF_INT 的屏蔽中断状态。(RO)

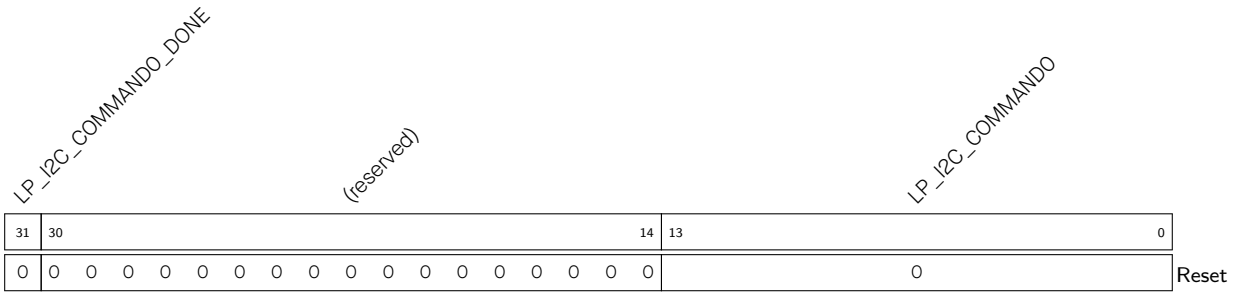
LP_I2C_RXFIFO_UDF_INT_ST LP_I2C_RXFIFO_UDF_INT 的屏蔽中断状态。(RO)

LP_I2C_SCL_ST_TO_INT_ST LP_I2C_SCL_ST_TO_INT 的屏蔽中断状态。(RO)

LP_I2C_SCL_MAIN_ST_TO_INT_ST LP_I2C_SCL_MAIN_ST_TO_INT 的屏蔽中断状态。(RO)

LP_I2C_DET_START_INT_ST LP_I2C_DET_START_INT 的屏蔽中断状态。(RO)

Register 34.58. LP_I2C_COMD0_REG (0x0058)



LP_I2C_COMMAND0 配置命令寄存器 0 的内容。

命令寄存器包含三个部分：

op_code 表示命令：

1：WRITE

2：STOP

3：READ

4：END

6：RSTART

Byte_num 表示需要发送或接收数据的字节数。

ack_check_en, ack_exp 和 ack 用于控制 ACK 位。详细信息，请参阅 I2C CMD 结构 34.4-2 。

(R/W)

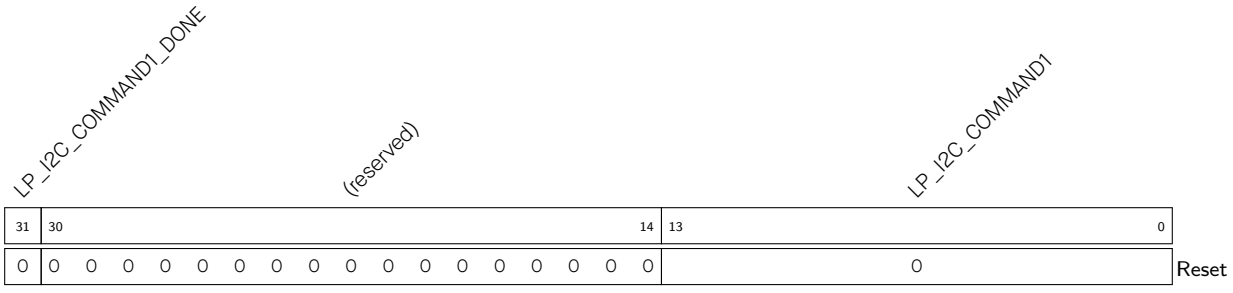
LP_I2C_COMMAND0_DONE 表示在 I2C 主机模式下，命令 0 是否已完成。

0：未完成

1：已完成

(R/W/SS)

Register 34.59. LP_I2C_COMD1_REG (0x005C)



LP_I2C_COMMAND1 配置命令寄存器 1 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

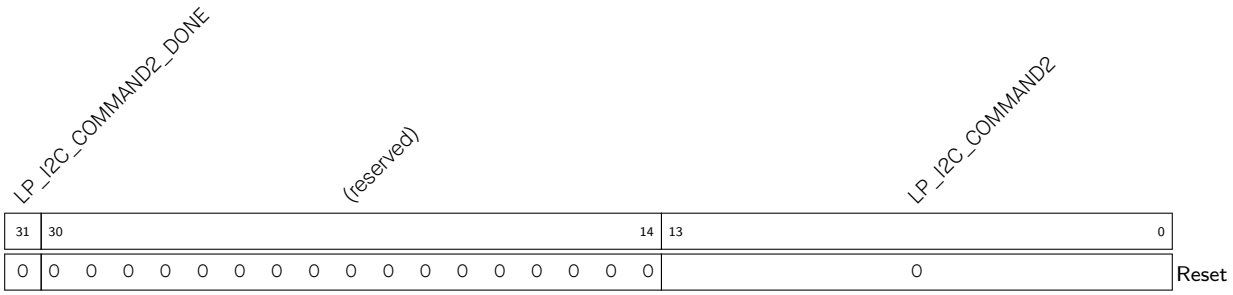
LP_I2C_COMMAND1_DONE 表示在 I2C 主机模式下，命令 1 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 34.60. LP_I2C_COMD2_REG (0x0060)



LP_I2C_COMMAND2 配置命令寄存器 2 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

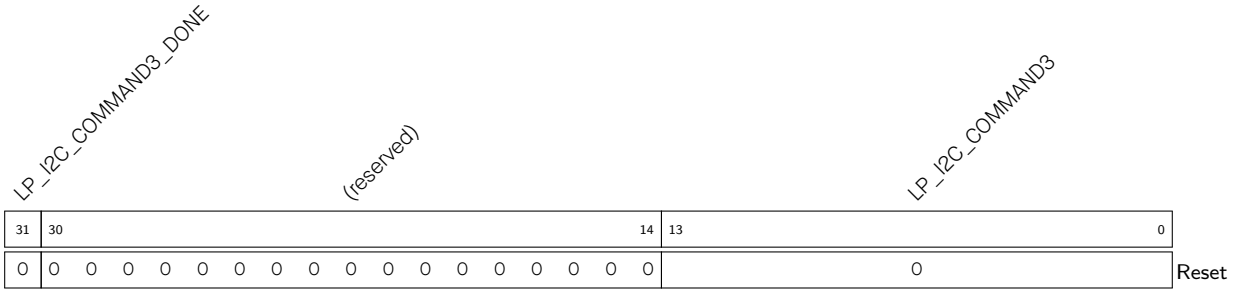
LP_I2C_COMMAND2_DONE 表示在 I2C 主机模式下，命令 2 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 34.61. LP_I2C_COMD3_REG (0x0064)



LP_I2C_COMMAND3 配置命令寄存器 3 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

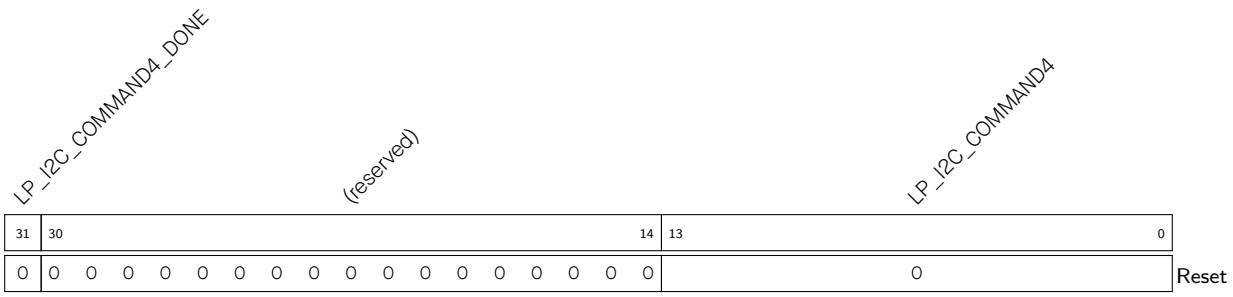
LP_I2C_COMMAND3_DONE 表示在 I2C 主机模式下，命令 3 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 34.62. LP_I2C_COMD4_REG (0x0068)



LP_I2C_COMMAND4 配置命令寄存器 4 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

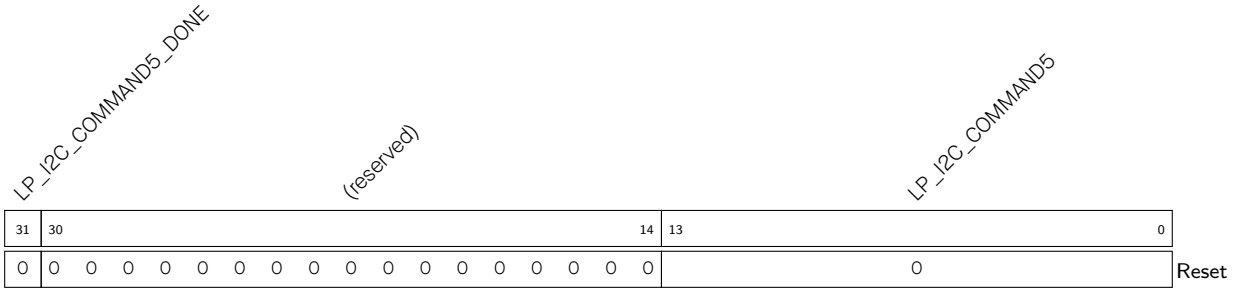
LP_I2C_COMMAND4_DONE 表示在 I2C 主机模式下，命令 4 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 34.63. LP_I2C_COMD5_REG (0x006C)



LP_I2C_COMMAND5 配置命令寄存器 5 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

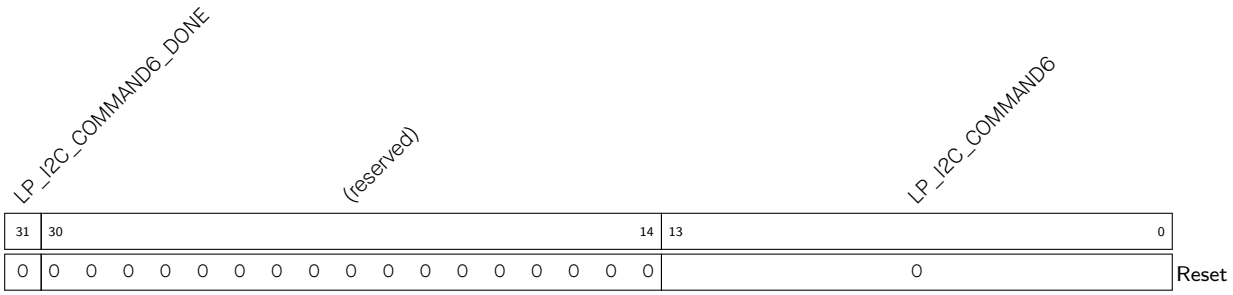
LP_I2C_COMMAND5_DONE 表示在 I2C 主机模式下，命令 5 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 34.64. LP_I2C_COMD6_REG (0x0070)



LP_I2C_COMMAND6 配置命令寄存器 6 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

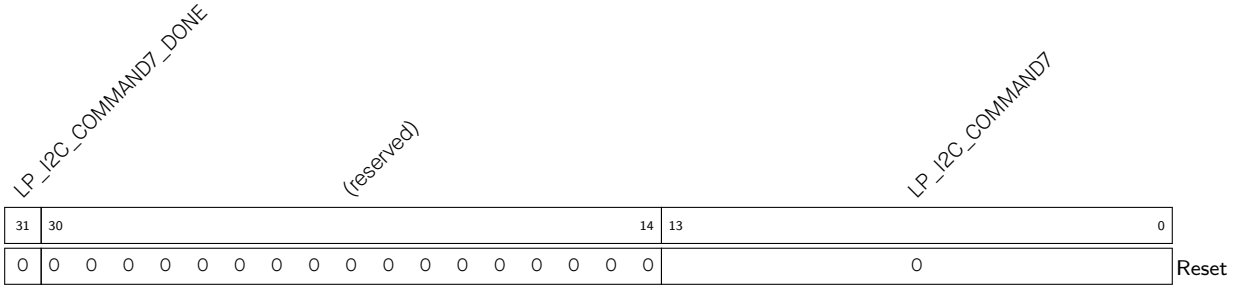
LP_I2C_COMMAND6_DONE 表示在 I2C 主机模式下，命令 6 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 34.65. LP_I2C_COMD7_REG (0x0074)



LP_I2C_COMMAND7 配置命令寄存器 7 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

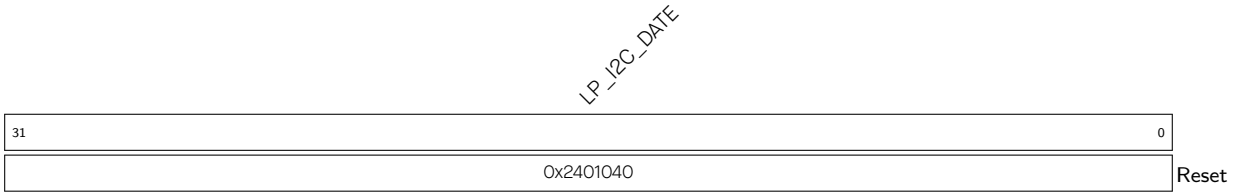
LP_I2C_COMMAND7_DONE 表示在 I2C 主机模式下，命令 7 是否已完成。

0: 未完成

1: 已完成

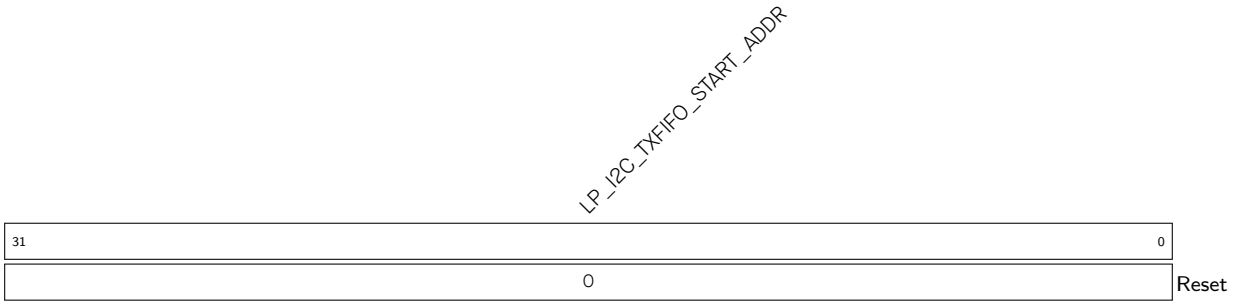
(R/W/SS)

Register 34.66. LP_I2C_DATE_REG (0x00F8)



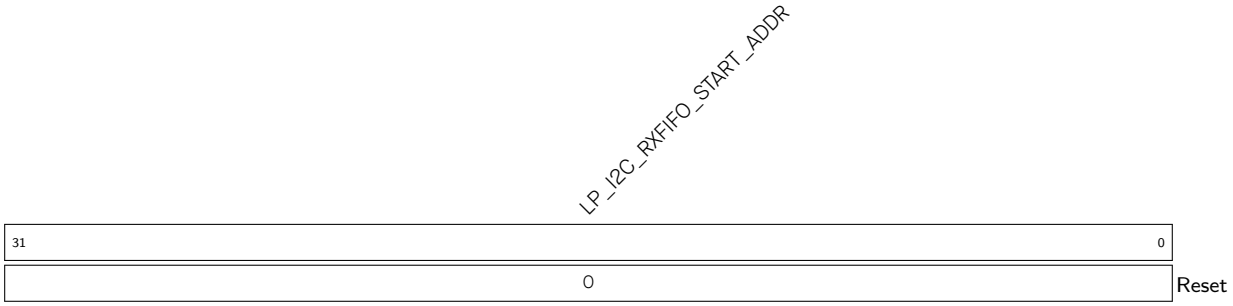
LP_I2C_DATE 版本控制寄存器。(R/W)

Register 34.67. LP_I2C_TXFIFO_START_ADDR_REG (0x0100)



LP_I2C_TXFIFO_START_ADDR 表示 I2C TX FIFO 首地址。(RO)

Register 34.68. LP_I2C_RXFIFO_START_ADDR_REG (0x0180)



LP_I2C_RXFIFO_START_ADDR 表示 I2C RX FIFO 首地址。(RO)

第 35 章

I2S 控制器 (I2S)

35.1 概述

ESP32-C5 内置一个 I2S 接口，为多媒体应用，尤其是为数字音频应用提供了灵活的数据通信接口。

I2S 标准总线定义了三种信号——串行时钟信号 BCK、字选择信号 WS 和串行数据信号 SD。一个基础的 I2S 数据总线包含一个主机和一个从机。主机和从机的角色在通信过程中保持不变。ESP32-C5 的 I2S 模块包含独立的 TX 模块和 RX 模块，能够保证优良的通信性能。

35.2 术语

为了更好地说明 I2S 的功能，本章使用了以下术语。

主机模式 I2S 作为主机驱动 BCK/WS，向从机发送或从其接收数据。

从机模式 I2S 作为从机由 BCK/WS 驱动，从主机接收或向其发送数据。

全双工 主机与从机之间的发送线和接收线各自独立，发送数据和接收数据同时进行。

半双工 主机和从机只能有一方先发送数据，另一方接收数据。发送数据和接收数据不能同时进行。

A 律和 μ 律 A 律和 μ 律是数字脉冲编码调制 (PCM) 非均匀量化中的压缩/解压缩算法，可以有效地改善信号的量化信噪比。

TDM RX 模式 即 TDM 输入模式，该模式利用时分复用方式接收脉冲编码调制 (PCM) 数据。信号线包括 BCK、WS 和 DATA。可以接收最多 16 个通道的数据。用户可配置数据格式，支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。

PDM RX 模式 即 PDM 输入模式，该模式接收脉冲密度调制 (PDM) 数据。信号线包括 WS 和 DATA。用户可配置数据格式，支持 PDM 标准格式等。

TDM TX 模式 即 TDM 输出模式，该模式利用时分复用方式发送脉冲编码调制 (PCM) 数据。信号线包括 BCK、WS 和 DATA，可以发送最多 16 个通道的数据。用户可配置数据格式，支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。

PDM TX 模式 即 PDM 输出模式，该模式发送脉冲密度调制 (PDM) 数据。信号线包括 WS 和 DATA。用户可配置数据格式，支持 PDM 标准格式等。

PCM-to-PDM 数据格式转换器 将 PCM 数据转换为 PDM 数据的格式转换器（后文简称为 PCM-to-PDM 转换器），可以在 PDM TX 主机模式下启用。启用后，I2S 通过 GDMA 从存储器中取得脉冲编码调制 (PCM) 数据，将其转换为脉冲密度调制 (PDM) 数据并发送。

35.3 特性

I2S 模块具有以下特性：

- 支持主机模式和从机模式
- 支持全双工和半双工通信
- 支持 TX 模块和 RX 模块独立工作或同时工作
- 支持多种音频标准：
  - TDM Philips 标准
  - TDM MSB 对齐标准
  - TDM PCM 标准
  - PDM 标准
- 支持多种 TX/RX 模式
  - TDM TX 模式，最多支持 16 通道
  - TDM RX 模式，最多支持 16 通道
  - PDM TX 模式
    - 支持原始 PDM 数据发送
    - 支持将 PCM 数据转换为 PDM 数据发送，最多支持 2 通道
  - PDM RX 模式
    - 支持原始 PDM 数据接收
- 可配置时钟源，支持最高频率为 240 MHz
- 可配置高精度采样时钟，支持多种采样频率（详见下文注释）
- 支持 8/16/24/32 位的数据位宽
- TX 模式支持同步计数器
- 支持 ETM 功能
- 支持 GDMA
- 支持 I2S 接口中断

说明：
从机模式下，为了保障采样的准确性，模块时钟频率必须大于等于 BCK 时钟频率的 8 倍，因此 I2S 支持的采样频率受到数据位宽和声道数的限制。例如，时钟源频率最高为 240 MHz，即模块时钟最高能配置为 120 MHz，能支持的 BCK 时钟频率最高为 15 MHz，因此传输双声道 32 位宽数据时，能够支持 234.375 kHz 以下的采样频率，例如 8 kHz、16 kHz、32 kHz、44.1 kHz、48 kHz、88.2 kHz、96 kHz、128 kHz、192 kHz。具体请参考章节 35.6 *I2S TX/RX 模块时钟*。

35.4 系统架构

图 35.4-1 为 ESP32-C5 I2S 模块的结构框图。ESP32-C5 I2S 模块包含：

- 独立的 TX 模块
- 独立的 RX 模块
- 输入输出时序调节单元（I/O 同步）
- 时钟分频器（时钟发生器）
- 64 x 32 位 TX FIFO
- 64 x 32 位 RX FIFO
- 压缩/解压缩模块

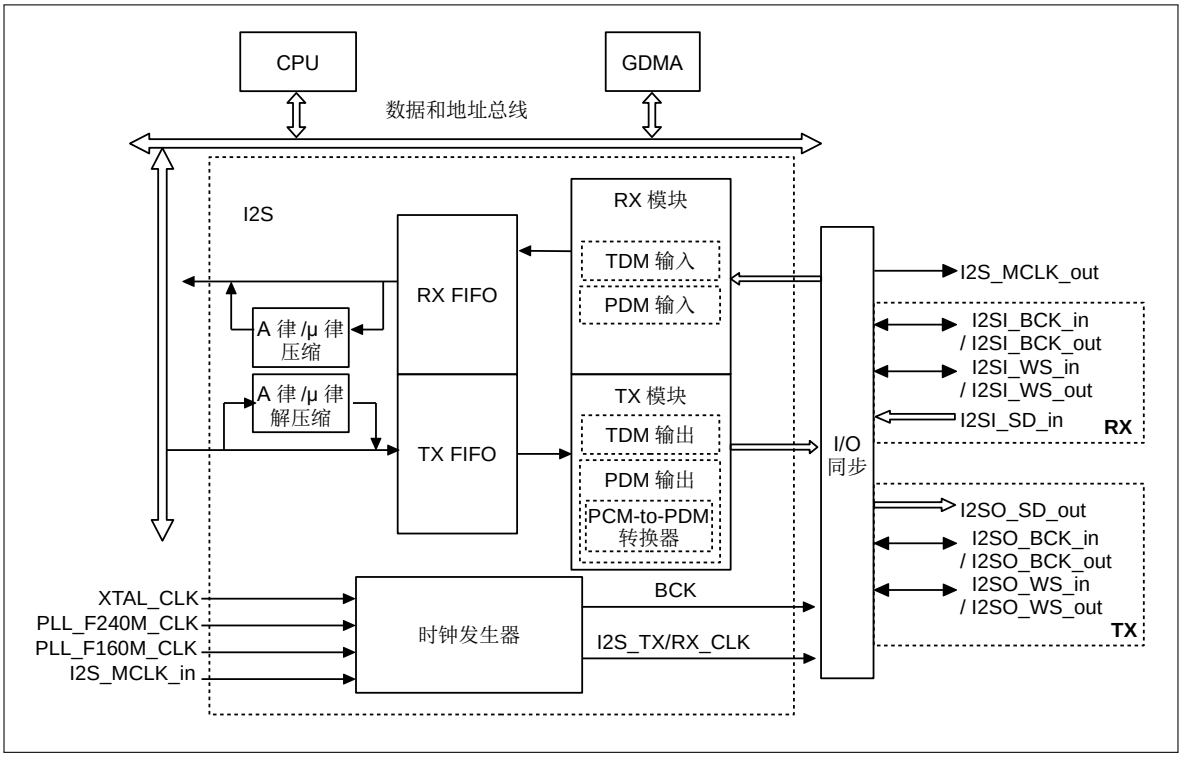


图 35.4-1. ESP32-C5 I2S 系统框图

ESP32-C5 I2S 模块支持 GDMA，可访问内部存储器，更多信息见章节 5 *通用 DMA 控制器 (GDMA)*。

TX 模块和 RX 模块各自有一组三线接口，分别为串行时钟线 BCK、字选择线 WS 和串行数据线 SD。其中，TX 模块的 SD 线固定为输出，RX 模块的 SD 线固定为接收。主机模式下，TX 模块和 RX 模块的 BCK 和 WS 信号线为输出，从机模式下，TX 模块和 RX 模块的 BCK 和 WS 信号线为输入。

图 35.4-1 中的右半部分展示了 I2S 模块的信号总线。RX 和 TX 模块的信号命名规则为：I2SA_B_C，例如 I2SI_BCK_in。其中：

- “A” 表示信号属于 I2S TX 或 RX 模块，包括：
  - “I” 表示信号输入至/输出自 RX 模块
  - “O” 表示信号输入至/输出自 TX 模块

- “B” 表示信号功能，包括：
  - BCK
  - WS
  - SD
- “C” 表示该信号的方向，包括：
  - “in” 表示该信号输入 I2S 模块
  - “out” 表示该信号自 I2S 模块输出

I2S 各信号的具体描述见表 35.4-1。

表 35.4-1. 模块信号描述

信号	方向	功能
I2SI_BCK_in	输入	I2S 从机模式下，输入 BCK 信号，用于 RX 模块
I2SI_BCK_out	输出	I2S 主机模式下，输出 BCK 信号，用于 RX 模块
I2SI_WS_in	输入	I2S 从机模式下，输入 WS 信号，用于 RX 模块
I2SI_WS_out	输出	I2S 主机模式下，输出 WS 信号，用于 RX 模块
I2SI_SD_in	输入	I2S RX 模块的串行输入数据线
I2SO_SD_out	输出	I2S TX 模块的串行输出数据线
I2SO_BCK_in	输入	I2S 从机模式下，输入 BCK 信号，用于 TX 模块
I2SO_BCK_out	输出	I2S 主机模式下，输出 BCK 信号，用于 TX 模块
I2SO_WS_in	输入	I2S 从机模式下，输入 WS 信号，用于 TX 模块
I2SO_WS_out	输出	I2S 主机模式下，输出 WS 信号，用于 TX 模块
I2S_MCLK_in	输入	I2S 从机模式下，来自外部芯片的时钟源
I2S_MCLK_out	输出	I2S 主机模式下，作为外部芯片的时钟源
I2SO_SD1_out	输出	TX 模块在启用 PCM-to-PDM 转换器时的串行输出数据线

* I2S 的所有信号均需要经过 GPIO 交换矩阵映射到芯片的管脚。更多信息请参考章节 8 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*。

35.5 I2S 模块支持的音频协议

ESP32-C5 I2S 模块支持多种音频标准，包括 TDM Philips 标准、TDM MSB 对齐标准、TDM PCM 标准以及 PDM 标准。

用户可通过配置以下寄存器，选择所需的音频标准：

- I2S_TX/RX_TDM_EN
  - 0：禁用 TDM 模式
  - 1：选择 TDM 模式
- I2S_TX/RX_PDM_EN
  - 0：禁用 PDM 模式
  - 1：选择 PDM 模式

- I2S_TX/RX_MSB_SHIFT
  - 0：配置 WS 信号和 SD 信号同时开始变化，即选择 MSB 对齐标准
  - 1：配置 WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即选择 Philips 标准或 PCM 标准

说明：
不可将 I2S_TX/RX_TDM_EN 和 I2S_TX/RX_PDM_EN 同时配置为 1 或同时配置为 0，否则会导致 ESP32-C5 I2S 以既非 TDM 模式也非 PDM 模式的状态错误发送数据。

35.5.1 TDM Philips 标准模式

在 Philips 标准下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并在当前通道数据发送结束前一个 BCK 时钟周期开始变化。SD 信号线上首先传输音频数据的最高位。

与普通 Philips 标准相比，TDM Philips 标准支持的通道更多，见图 35.5-1。

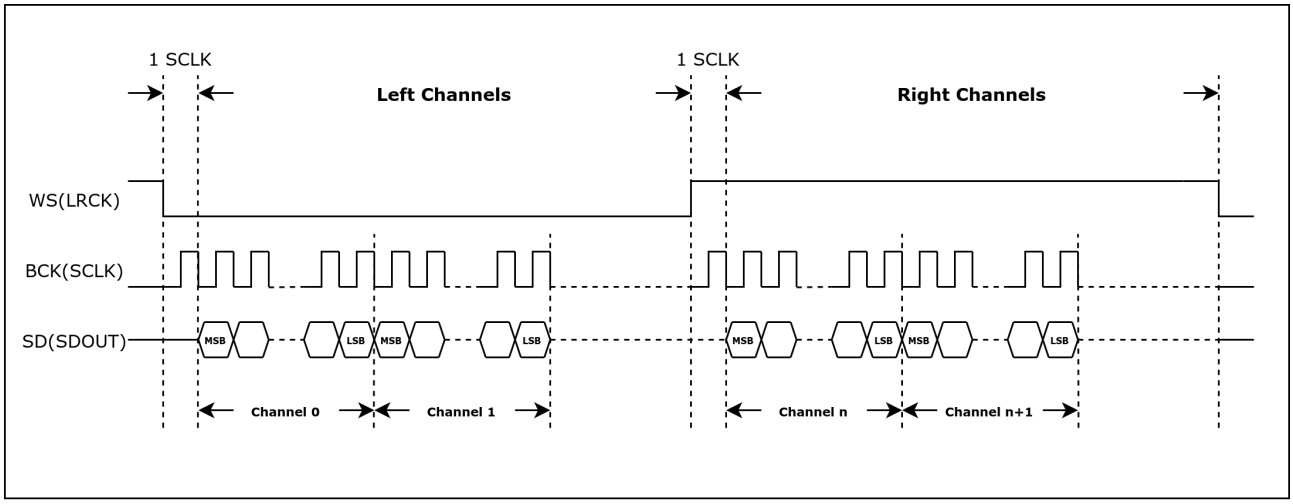


图 35.5-1. 时序图 – TDM Philips 标准

35.5.2 TDM MSB 对齐标准模式

MSB 对齐标准下，在 BCK 下降沿，WS 信号和 SD 信号同时变化。WS 持续到当前通道数据发送结束，SD 信号线上首先传输音频数据的最高位。

与普通 MSB 对齐标准相比，TDM MSB 对齐标准支持更多通道，见图 35.5-2。

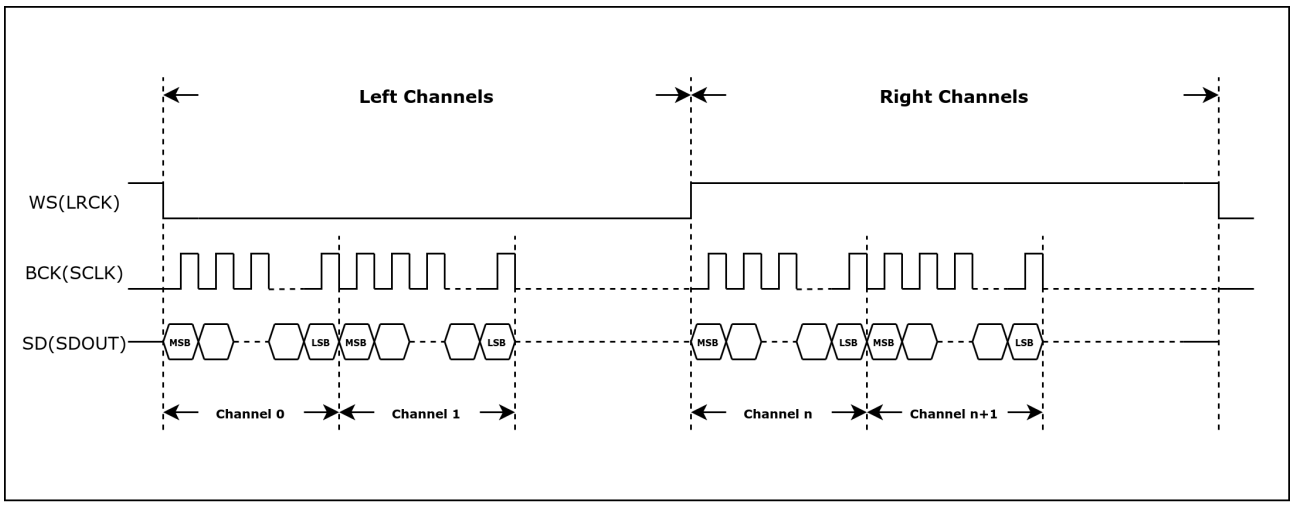


图 35.5-2. 时序图 – TDM MSB 对齐标准

35.5.3 TDM PCM 标准模式

在 PCM 标准的短帧同步模式下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并持续一个 BCK 时钟周期。SD 信号线上首先传输音频数据的最高位。

与普通 PCM 标准相比，TDM PCM 标准支持更多通道，见图 35.5-3 所示。

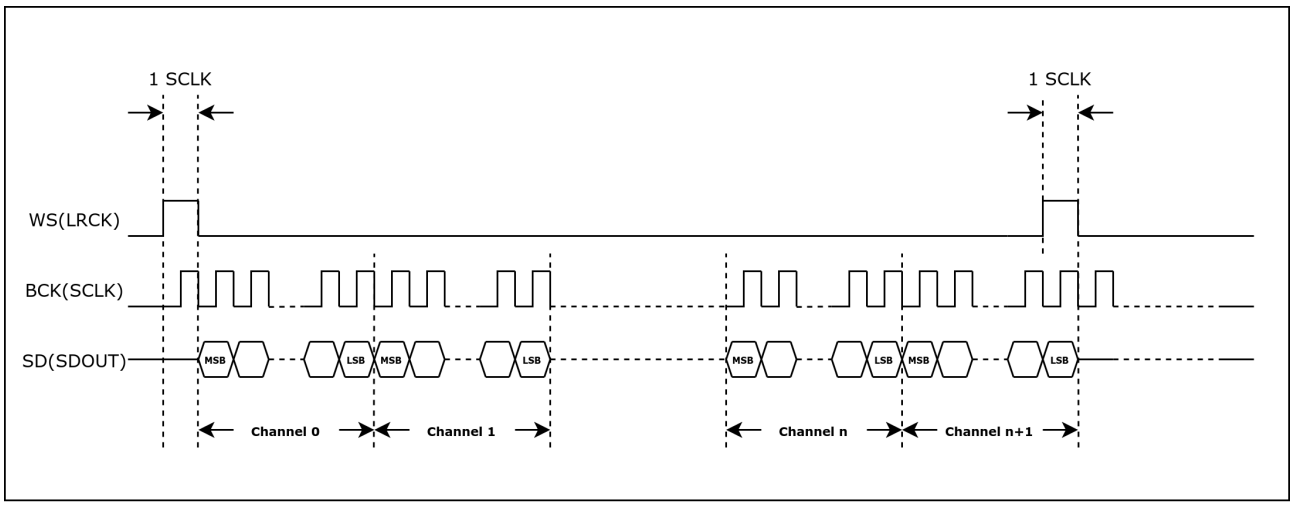


图 35.5-3. 时序图 – TDM PCM 标准

35.5.4 PDM 标准模式

如图 35.5-4 所示，在 PDM 标准下，WS 代表左/右声道，WS 与 SD 同时变化。WS 在数据发送过程中持续变化，WS 的高低对应两个声道。请参考图 35.5-4。

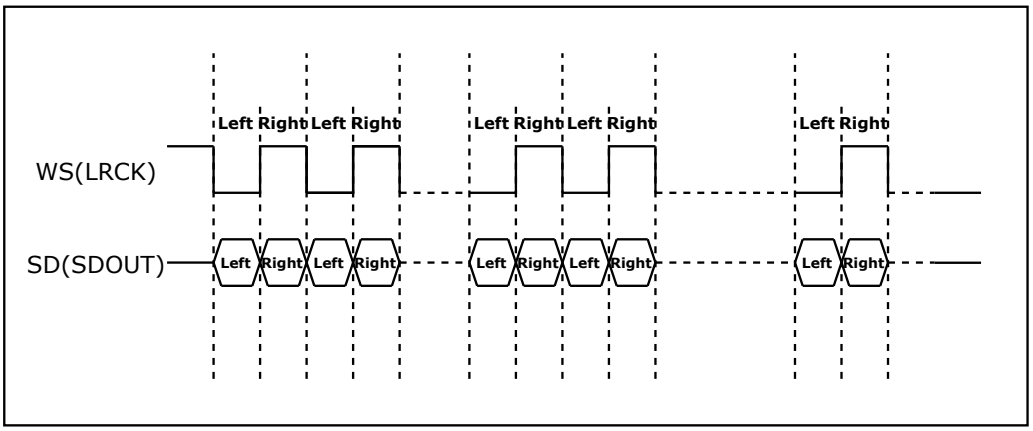


图 35.5-4. 时序图 – PDM 标准

35.6 I2S TX/RX 模块时钟

I2S_TX/RX_CLK 作为 I2S TX/RX 模块的主时钟，可由以下时钟分频所得：

- 48 MHz XTAL_CLK
- 240 MHz PLL_F240M_CLK
- 160 MHz PLL_F160M_CLK
- 外部输入时钟 I2S_MCLK_in

I2S TX/RX 模块的串行时钟 BCK 再由 I2S_TX/RX_CLK 分频获得，如图 35.6-1 所示。PCR_I2S_TX/RX_CLKM_SEL 用于选择 I2S TX/RX 的时钟源，PCR_I2S_TX/RX_CLKM_EN 用于使能或者关闭 I2S TX/RX 模块的时钟源。

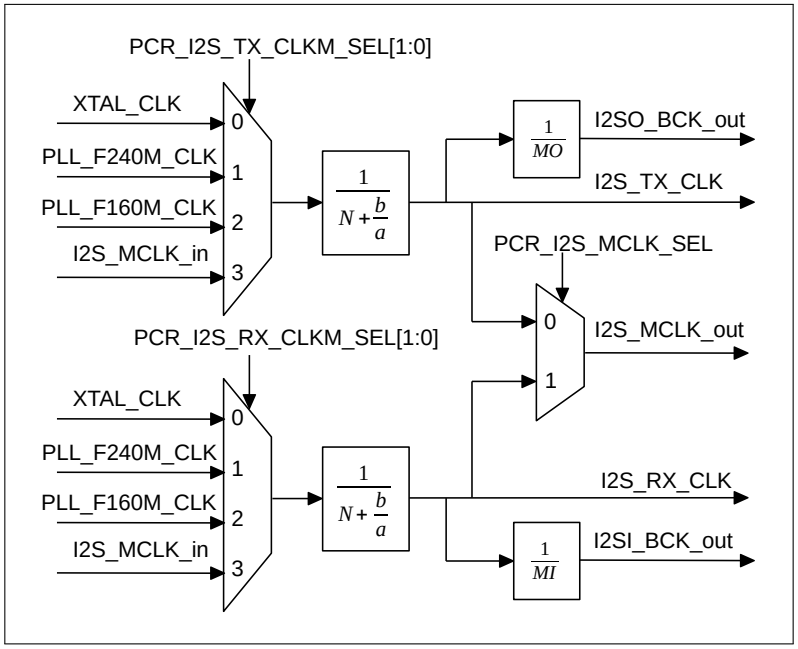


图 35.6-1. I2S 时钟分频器

I2S_TX/RX_CLK 的频率 f_{I2S_TX/RX_CLK} 与分频器时钟源频率 $f_{I2S\_CLK\_S}$ 间的关系如下：

$$f_{I2S\_TX/RX\_CLK} = \frac{f_{I2S\_CLK\_S}}{N + \frac{b}{a}}$$

其中，2 <= N <= 256（N 为整数），N 对应寄存器域 PCR_I2S_TX/RX_CLKM_DIV_NUM 的值，具体为：

- PCR_I2S_TX/RX_CLKM_DIV_NUM = 0 时，N = 256；
- PCR_I2S_TX/RX_CLKM_DIV_NUM = 1 时，N = 2；
- PCR_I2S_TX/RX_CLKM_DIV_NUM 为其他值时，N = PCR_I2S_TX/RX_CLKM_DIV_NUM 的值。

分数部分分频系数 a、b 唯一对应系数 x、y、z、yn1。系数对应公式为：

- 当 b <= $\frac{a}{2}$ 时，yn1 = 0，x = $floor([\frac{a}{b}]) - 1$，y = a%b，z = b；
- 当 b > $\frac{a}{2}$ 时，yn1 = 1，x = $floor([\frac{a}{a-b}]) - 1$，y = a%(a - b)，z = a - b。

系数 x、y、z、yn1 在 PCR_I2S_TX/RX_CLKM_DIV_X、PCR_I2S_TX/RX_CLKM_DIV_Y、PCR_I2S_TX/RX_CLKM_DIV_Z 和 PCR_I2S_TX/RX_CLKM_DIV_YN1 中配置。

对于整数分频，PCR_I2S_TX/RX_CLKM_DIV_X 和 PCR_I2S_TX/RX_CLKM_DIV_Z 清零，PCR_I2S_TX/RX_CLKM_DIV_Y 置为 1。

说明：
使用小数分频功能可能会产生时钟抖动。

在主机发送模式下，I2S TX 模块的串行时钟 BCK 为 I2SO_BCK_out 信号，由 I2S_TX_CLK 分频获得。即：

$$f_{I2SO\_BCK\_out} = \frac{f_{I2S\_TX\_CLK}}{MO}$$

其中，MO 的值为：

$$MO = I2S\_TX\_BCK\_DIV\_NUM + 1$$

说明：
注意，I2S_TX_BCK_DIV_NUM 不可配置为 1。

在主机接收模式下，I2S RX 模块的串行时钟 BCK 为 I2SI_BCK_out 信号，由 I2S_RX_CLK 分频获得。即：

$$f_{I2SI\_BCK\_out} = \frac{f_{I2S\_RX\_CLK}}{MI}$$

其中 MI 的值为：

$$MI = I2S\_RX\_BCK\_DIV\_NUM + 1$$

说明：

- I2S_RX_BCK_DIV_NUM 不可配置为 1；
- 当模块处于 I2S 从机模式时，必须保证 f_{I2S_TX/RX_CLK} >= 8 * f_{BCK}。另外模块可以输出 I2S_MCLK_out 作为外部设备的主时钟。

35.7 I2S 模块复位

I2S 模块中各个单元以及 FIFO 可通过配置相关位进行复位：

- I2S TX/RX 单元：可配置 I2S_TX_RESET 和 I2S_RX_RESET 进行复位；
- I2S TX/RX FIFO：可配置 I2S_TX_FIFO_RESET 和 I2S_RX_FIFO_RESET 进行复位。

说明：
在模块和 FIFO 复位之前，需要先配置 I2S 模块时钟。

35.8 I2S 主/从机模式

ESP32-C5 I2S 模块可作为主机或从机，两种模式均支持半双工通信或全双工通信。用户可配置 I2S_RX_SLAVE_MOD 和 I2S_TX_SLAVE_MOD 选择需要的模式。

- I2S_TX_SLAVE_MOD
  - 0：主机发送模式
  - 1：从机发送模式
- I2S_RX_SLAVE_MOD
  - 0：主机接收模式
  - 1：从机接收模式

35.8.1 主/从机发送模式

- 主机发送模式
  - 置位 I2S_TX_START 启动一次发送操作。置位该位，TX 模块会一直输出时钟信号和串行数据。
  - 置位 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，则主机停止发送数据和时钟信号。
  - 清零 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，发送模块将一直发送最后一帧数据和时钟信号。
  - 当 I2S_TX_START 被清零时，主机停止发送数据。
- 从机发送模式
  - 置位 I2S_TX_START。TX 模块等待主机 BCK 时钟，来启动发送操作。
  - 置位 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，则从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。
  - 清零 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，TX 模块将一直发送最后一帧数据。
  - 当 I2S_TX_START 被清零时，从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。

35.8.2 主/从机接收模式

- 主机接收模式
  - 置位 I2S_RX_START 启动一次接收操作。RX 模块会一直输出时钟信号，并对输入数据进行采样。
  - 配置 I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    * 0：只有 I2S_RX_START 被清除，RX 模块才会暂停接收。
    * 1：RX 模块会在 I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收。当 I2S_RX_START 未被清除时，可通过置位 I2S_RX_RESET 来重新启动接收。
    * 2：RX 模块会在 I2S_RX_START 被清除或 GDMA RX FIFO 满时暂停接收。当 I2S_RX_START 未被清除且 GDMA RX FIFO 不再满时，可通过置位 I2S_RX_RESET 重新启动接收。
  - 清零 I2S_RX_START，RX 模块停止接收数据。
- 从机接收模式
  - 置位 I2S_RX_START。等待主机 BCK 时钟，来启动接收操作。
  - 配置 I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    * 0：只有 I2S_RX_START 被清除，RX 模块才会暂停接收；
    * 1：RX 模块会在 I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收。当 I2S_RX_START 未被清除时，可通过置位 I2S_RX_RESET 来重新开始接收；
    * 2：RX 模块会在 I2S_RX_START 被清除或 GDMA RX FIFO 满时暂停接收。当 I2S_RX_START 未被清除且 GDMA RX FIFO 不再满时，可通过置位 I2S_RX_RESET 恢复接收。
  - 清零 I2S_RX_START，RX 模块停止接收数据。

35.9 发送数据

> **说明：**
> 本小节以及后续小节所述的配置，均需要通过置位 I2S_TX_UPDATE 的方式来进行更新，从而将 I2S TX 寄存器数据从 APB 时钟域同步到 I2S TX 时钟域。详细配置见第 35.13.1 小节。

I2S 发送数据时，从 GDMA 读取数据，经过数据格式控制和通道模式控制，从外设输出信号输出对应数据。

35.9.1 数据格式控制

数据格式控制分为三个阶段：

- 第一阶段：从内存中读出有效数据并写入 TX FIFO；
- 第二阶段：将待发送数据从 TX FIFO 中读出，并进行输出数据模式转换；
- 第三阶段：将待发送数据转换为串行数据流输出。

35.9.1.1 通道有效数据位宽

I2S_TX_BITS_MOD 和 I2S_TX_24_FILL_EN 决定了每个通道的有效数据位宽，其可取值和对应的有效数据位宽如下表：

表 35.9-1. 通道有效数据位宽控制

通道有效数据位宽	I2S_TX_BITS_MOD	I2S_TX_24_FILL_EN
32	31	x*
	23	1
24	23	0
16	15	x
8	7	x

* “x”表示该值被忽略（适用于本章所有表格）。

35.9.1.2 通道有效数据字节序

I2S 通过 GDMA 读取数据之后，I2S_TX_BIG_ENDIAN 用于控制从 GDMA 读取数据的字节序。表 35.9-2 描述了不同通道有效数据位宽下，该寄存器对读取数据的控制。

表 35.9-2. 通道有效数据字节序控制

通道有效数据位宽	原始数据	控制后数据字节序	I2S_TX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

说明：
B0、B1、B2、B3 各代表一个 8 位数据，符号 {} 表示这些字节被组合到一起。例如，{B3, B2, B1, B0} 代表一个 32 位的数据，其中 B0 代表 0-7 位，B1 代表 8-15 位，B2 代表 16-23 位，B3 代表 24-31 位。

35.9.1.3 A 律/μ 律压缩/解压缩

ESP32-C5 I2S 按照 32 位（缺省高位补 0*）的方式对排列好字节序的有效数据进行 A 律/μ 律压缩/解压缩。

说明：
缺省高位补 0，即如果有效数据位宽小于 32，则待压缩/解压缩数据的 [31: 有效位宽] 位自动填充 0。

配置 I2S_TX_PCM_BYPASS 为：

- 0：进行压缩/解压缩
- 1：不进行压缩/解压缩

配置 I2S_TX_PCM_CONF 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2：μ 律解压缩
- 3：μ 律压缩

至此，数据格式控制的第一阶段完成。

35.9.1.4 通道发送数据位宽

ESP32-C5 I2S 中，I2S_TX_TDM_CHAN_BITS 决定了每个通道发送数据的位宽。

- 当每个通道发送数据的位宽大于有效数据位宽时，会在剩余的位置补充 0。此时，配置 I2S_TX_LEFT_ALIGN 为：
  - 0：有效数据位于发送数据低位，在数据高位补零；
  - 1：有效数据位于发送数据高位，在数据低位补零。
- 当每个通道发送数据的位宽小于有效数据位宽时，每次发送数据仅发送低位有效数据部分，高位部分将被舍弃。

至此，数据格式控制的第二阶段完成。

35.9.1.5 通道数据比特顺序

ESP32-C5 I2S 通道数据比特顺序由 I2S_TX_BIT_ORDER 控制：

- 0：不调换有效数据比特顺序；
- 1：调换有效数据比特顺序。

至此，数据格式控制部分全部完成。经过数据格式控制的数据将从高位向低位依次发送。图 35.9-1 所示即为一次完整的 TX 数据格式控制过程。

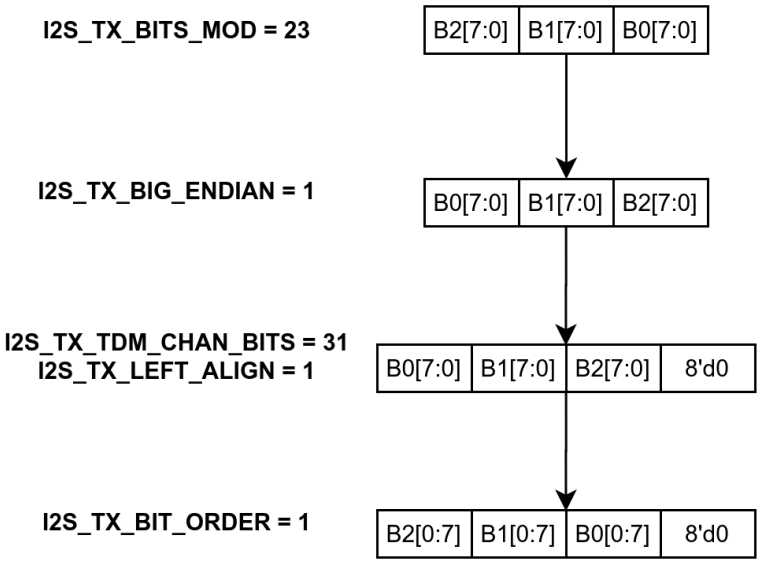


图 35.9-1. TX 数据格式控制

35.9.2 通道模式控制

ESP32-C5 I2S 支持 TDM 和 PDM 两种发送模式。置位 I2S_TX_TDM_EN 使能 TDM 输出模式，置位 I2S_TX_PDM_EN 使能 PDM 输出模式。

> **说明：**
> I2S_TX_TDM_EN 和 I2S_TX_PDM_EN 不能同时置位或同时清零。

35.9.2.1 TDM 输出模式

在 TDM 输出模式下，I2S 最多支持 16 个发送通道。发送通道数由 I2S_TX_TDM_TOT_CHAN_NUM 控制。例如，配置 I2S_TX_TDM_TOT_CHAN_NUM 为 5，则六个通道（通道 0 ~ 5）将用于发送数据，见图 35.9-2。

> **说明：**
> 将 I2S 模块设置为 TDM 双通道模式，可实现控制大多数 I2S 双声道编解码器。

在发送数据的通道中，如果其对应的 I2S_TX_TDM_CHANn_EN 为：

- 1：则该通道发送通道数据；
- 0：则该通道发送数据由 I2S_TX_CHAN_EQUAL 控制，当该值为：
  - 1：则发送前一个通道的数据；
  - 0：则发送 I2S_SINGLE_DATA 的值。

当 I2S 处于主机 TDM 输出模式下，WS 信号由 I2S_TX_WS_IDLE_POL 和 I2S_TX_TDM_WS_WIDTH 控制：

- I2S_TX_WS_IDLE_POL 的值为 WS 信号的默认电平；
- I2S_TX_TDM_WS_WIDTH 的值为在发送所有通道数据的过程中，WS 为默认电平的周期数。

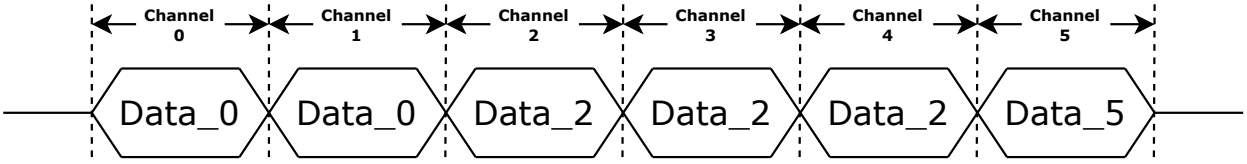
另外，I2S_TX_HALF_SAMPLE_BITS 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

TDM 通道配置示例

在本示例中，寄存器的配置如下：

- 配置 I2S_TX_TDM_CHAN_NUM 为 5，即选择使用通道 0 ~ 5 进行数据发送。
- 配置 I2S_TX_CHAN_EQUAL 为 1，即如果相应通道的 I2S_TX_TDM_CHAN*n*_EN (*n* = 0 ~ 5) 被清除，则该通道将发送前一通道的数据。
- 配置 I2S_TX_TDM_CHAN0/2/5_EN 为 1，即这些通道将用于发送通道数据。
- 配置 I2S_TX_TDM_CHAN1/3/4_EN 为 0，则这些通道将发送前一个通道的数据。

配置完成后，则数据将按下图方式进行发送。



I2S_TX_TDM_CHAN_NUM = 5; I2S_TX_CHAN_EQUAL = 1;
I2S_TX_TDM_CHAN0_EN = 1; I2S_TX_TDM_CHAN1_EN = 0; I2S_TX_TDM_CHAN2_EN = 1;
I2S_TX_TDM_CHAN3_EN = 0; I2S_TX_TDM_CHAN4_EN = 0; I2S_TX_TDM_CHAN5_EN = 1;

图 35.9-2. TDM 通道控制

35.9.2.2 PDM 输出模式

在 PDM 输出模式下，I2S 可以发送原始 PDM 数据，或者将 PCM 数据转换为 PDM 数据后发送。

在 PDM 输出模式下，从 GDMA 取得数据的过程由 I2S_TX_MONO 和 I2S_TX_MONO_FST_VLD 控制，具体如下表。请根据内存中存储的数据为单/双声道数据来配置该寄存器。

表 35.9-3. PDM 输出模式下 I2S 取数逻辑

取数逻辑	模式	I2S_TX_MONO	I2S_TX_MONO_FST_VLD
每个 WS 沿都向 GDMA 发起取数请求	双声道	0	x
只在 WS 后半周期向 GDMA 发起取数请求	单声道	1	0
只在 WS 前半周期向 GDMA 发起取数请求	单声道	1	1

当 I2S 处于主机 PDM 输出模式时，WS 信号的默认电平由 I2S_TX_WS_IDLE_POL 控制，WS 信号频率为 BCK 信号频率的一半。请参考 35.6 小节配置 BCK 信号的方式来配置 WS 信号，见图 35.9-3。

在**发送原始 PDM 数据**时，I2S 通道数据由 I2S_TX_CHAN_MOD 和 I2S_TX_WS_IDLE_POL 控制，具体如下表。

表 35.9-4. 原始 PDM 数据的 I2S 通道模式控制

模式	左声道	右声道	模式控制字段[1]	声道选择位[2]
双声道	发送左通道数据	发送右通道数据	0	x
单声道	发送左通道数据	发送左通道数据	1	0
	发送右通道数据	发送右通道数据	1	1
	发送右通道数据	发送右通道数据	2	0
	发送左通道数据	发送左通道数据	2	1
	发送 “single” 值[3]	发送右通道数据	3	0
	发送左通道数据	发送 “single” 值	3	1
	发送左通道数据	发送 “single” 值	4	0
	发送 “single” 值	发送右通道数据	4	1

[1] I2S_TX_CHAN_MOD
[2] I2S_TX_WS_IDLE_POL
[3] single 值等于 I2S_SINGLE_DATA 的值。

在 **启用 PCM-to-PDM 转换器**时，可以将 GDMA 中的 PCM 数据转为 PDM 数据，并按照 PDM 信号格式输出，配置 I2S_PCM2PDM_CONV_EN 启动该转换器。PCM-to-PDM 转换器的寄存器配置如下：

- 配置一线 PDM 输出格式或一/二线 DAC 输出格式，具体如下表。

表 35.9-5. I2S PCM-to-PDM 数据输出

通道输出格式	I2S_TX_PDM_DAC_MODE_EN	I2S_TX_PDM_DAC_2OUT_EN
一线 PDM 输出格式[1]	0	x
一线 DAC 输出格式[2]	1	0
二线 DAC 输出格式	1	1

说明：

1. 此处定义的 PDM 输出格式是指一个 WS 周期发送两个通道的 SD 数据。
2. 此处定义的 DAC 输出格式是指一个 WS 周期发送一个通道的 SD 数据。
3. 一线 PDM/DAC 输出格式使用 I2SO_Data_out 作为数据输出线。
4. 二线 DAC 输出格式使用 I2SO_Data_out 和 I2SO_Data1_out 作为数据输出线。
5. 从应用上来说，PDM 输出格式针对的是需要时钟信号、可以解码 PDM 格式编解码器的应用场景，DAC 输出格式针对的是不依赖时钟信号、没有 PDM 格式编解码器、且通过低通滤波可以直接恢复成模拟波形的应用场景。

- 配置采样频率和上采样率。
  I2S 启用 PCM-to-PDM 转换器进行数据输出时，PDM 时钟频率即为 BCK 时钟频率。采样频率（$f_{Sampling}$，即 PCM 数据采样频率）与 BCK 时钟频率（PDM 数据采样频率）的关系如下：

$$f_{Sampling} = \frac{f_{BCK}}{OSR}$$

其中，上采样率 OSR 和 I2S_TX_PDM_SINC_OSR2 的关系为：

$$OSR = I2S\_TX\_PDM\_SINC\_OSR2 \times 64$$

采样频率 $f_{Sampling}$ 和 I2S_TX_PDM_FS 的对应关系为：

$$f_{Sampling} = I2S\_TX\_PDM\_FS \times 100$$

请根据需要的采样频率、上采样率以及 PDM 时钟频率进行寄存器配置。

PDM 通道配置示例

本示例针对 I2S，示例中寄存器的配置如下：

- 配置 I2S_PCM2PDM_CONV_EN 为 0，即选择输出原始 PDM 数据。
- 配置 I2S_TX_MONO 为 0，即 I2S 在 WS 的高低电平均通过 GDMA 从存储器取数。
- 配置 I2S_TX_CHAN_MOD 为 2，即选择单声道模式。
- 配置 I2S_TX_WS_IDLE_POL 为 1，则左声道和右声道均发送左通道数据，右声道数据将被舍弃。

配置完成后，假定存储器中的数据**经过数据格式控制之后**为：

Left	Right	Left	Right	...	Left	Right

说明：

1. 此处的数据并非存储器中的原始数据，而是经过数据格式控制之后的数据。
2. 表中的“Left”和“Right”代表通道数据，其位宽为通道有效数据位宽，请参考章节 35.9.1。

则数据经过通道模式控制后，将按照下图方式进行发送。

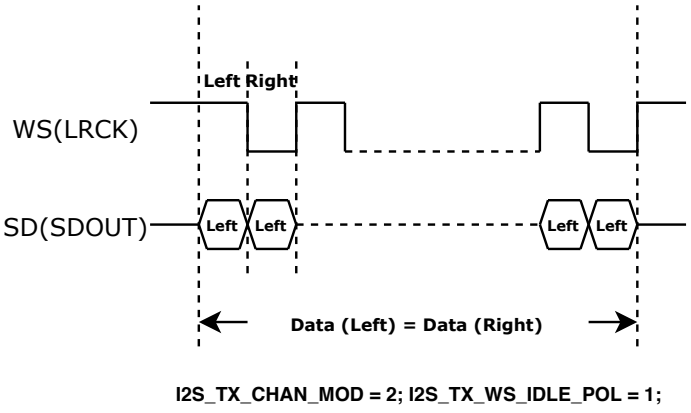


图 35.9-3. PDM 通道控制

35.9.3 同步计数器

ESP32-C5 I2S TX 模块包含同步计数器，用于反映当前 I2S 的数据发送情况。同步计数器可用于检测 I2S 发送数据错误，或在多个设备使用 I2S 发送数据时进行同步。

ESP32-C5 I2S 共包含两种类型的同步计数器：

- I2S_TX_FIFO_CNT
  - 每次 I2S TX FIFO 成功读取一个数据，该计数器加 1。
  - 置位 I2S_TX_FIFO_CNT_RST 可复位该同步计数器。
  - 该计数器位宽为 31 比特，溢出时自动归零。
- I2S_TX_BCK_CNT
  - 每次 I2S TX 发送一个 BCK 周期，该计数器加 1。
  - 可以通过置位 I2S_TX_BCK_CNT_RST 来复位该同步计数器。
  - 该计数器位宽为 31 比特，溢出时自动归零。

说明：

- 每次 I2S TX FIFO 发送一个数据，均意味着发送了一个通道数据，因此，当 I2S 正确发送数据时，I2S_TX_BCK_CNT = I2S_TX_FIFO_CNT * 通道发送数据位宽（请参考小节 35.9.1.4）。
- 由于 I2S TX 模块发送数据时，从 FIFO 取出数据到把数据发送至线上存在一些延迟，因此读到 I2S_TX_BCK_CNT 的值与 I2S_TX_FIFO_CNT 的值可能略微不对应。

应用场景举例

1. 多个设备 I2S TX 同步
   多个设备同时通过 I2S TX 发送数据时，由于时钟源的微小频率差，长时间发送数据后会出现同步问题。此时，可以通过读取每个设备的 I2S_TX_FIFO_CNT 值来判断发送音频数据的位置，从而进行同步操作。
2. I2S TX 稳定性检查
   当设备处于 GDMA 占用较多的应用场景时，可能会由于多路仲裁，导致 I2S TX 出现偶尔缺数。此时，检查 I2S_TX_BCK_CNT 与 I2S_TX_FIFO_CNT 的比例关系，可以确认设备运行 I2S TX 时的稳定性。

35.10 接收数据

I2S 接收数据时，从外设接口读取数据，经过通道模式控制和数据格式控制，通过 GDMA 输入数据至内存。

35.10.1 通道模式控制

ESP32-C5 I2S 支持 TDM 和 PDM 两种输入模式。置位 I2S_RX_TDM_EN 使能 TDM 输入模式，置位 I2S_RX_PDM_EN 使能 PDM 输入模式。

说明：
I2S_RX_TDM_EN 和 I2S_RX_PDM_EN 不能同时置位或同时清零。

35.10.1.1 TDM 输入模式

在 TDM 输入模式下，I2S 最多支持 16 个接收通道。接收通道数由 I2S_RX_TDM_TOT_CHAN_NUM 控制。例如，配置 I2S_RX_TDM_TOT_CHAN_NUM 为 5，则通道 0 ~ 5 接收数据。

在接收数据的通道中，如果其对应的 I2S_RX_TDM_CHANn_EN 为：

- 0：则该通道数据无效，不会存入 RX FIFO；

- 1：则该通道数据有效，会被存入 RX FIFO。

当 I2S 处于主机 TDM 输入模式下，WS 信号由 I2S_RX_WS_IDLE_POL 和 I2S_RX_TDM_WS_WIDTH 控制：

- I2S_RX_WS_IDLE_POL 的值为 WS 信号的默认电平；
- I2S_RX_TDM_WS_WIDTH 的值为在接收所有通道的过程中，WS 为默认电平的周期数。

另外，I2S_RX_HALF_SAMPLE_BITS 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

35.10.1.2 PDM 输入模式

在 PDM 输入模式下，I2S 可以接收原始 PDM 数据，将通道中的串行数据转换为待输入数据。

当 I2S 处于主机 PDM 输入模式下，WS 信号的默认电平由 I2S_RX_WS_IDLE_POL 控制，WS 信号频率为 BCK 信号频率的一半，请参考 35.6 小节配置 BCK 信号的方式配置 WS 信号。注意，在 PDM 输入模式下，请配置 I2S_RX_HALF_SAMPLE_BITS 的值与 I2S_RX_BITS_MOD 的值相同。

35.10.2 数据格式控制

数据格式控制分为两个阶段：

- 第一阶段：从串行数据流输入转换为待输入数据，并存入 RX FIFO；
- 第二阶段：将待输入数据从 RX FIFO 中读出，并进行输入数据模式转换。

35.10.2.1 通道数据比特顺序

ESP32-C5 I2S 通道数据将从高位到低位依次存入待输入数据，并由 I2S_RX_BIT_ORDER 控制是否调换比特序，当该值为：

- 0：不调换待输入数据的比特序；
- 1：调换待输入数据的比特序。

至此，数据格式控制的第一阶段完成，经过比特顺序控制的待输入数据被存入 RX FIFO。

35.10.2.2 通道储存数据位宽

I2S_RX_BITS_MOD 和 I2S_RX_24_FILL_EN 决定了每个通道的储存数据位宽，其可取值和对应的储存数据位宽如下表：

表 35.10-1. 通道储存数据位宽控制

通道储存数据位宽	I2S_RX_BITS_MOD	I2S_RX_24_FILL_EN
32	31	x
	23	1
24	23	0
16	15	x
8	7	x

35.10.2.3 通道接收数据位宽

I2S_RX_TDM_CHAN_BITS 决定了每个通道接收数据的位宽。

- 当每个通道储存数据位宽小于接收数据的位宽时，接收时仅会在接收数据中取储存位宽作为储存数据。此时，配置 I2S_RX_LEFT_ALIGN 为：
  - 0：储存数据位于接收数据低位；
  - 1：储存数据位于接收数据高位。
- 当每个通道接收数据的位宽小于储存数据位宽时，会将接收数据高位补 0，作为储存数据。

35.10.2.4 通道储存数据字节序

通道接收数据经过截取/补全后，成为了待储存数据，I2S_RX_BIG_ENDIAN 用于控制待储存数据的字节序。下表描述了不同通道储存数据位宽下，该寄存器对待储存数据的控制。

表 35.10-2. 通道储存数据字节序控制

通道储存数据位宽	原始数据	控制后数据	I2S_RX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

35.10.2.5 A 律/μ 律压缩/解压缩

I2S 对排列好字节序的待储存数据会按照 32 位（缺省高位补 0）的方式进行 A 律/μ 律压缩/解压缩。

配置 I2S_RX_PCM_BYPASS 为：

- 0：进行压缩/解压缩
- 1：不进行压缩/解压缩

配置 I2S_RX_PCM_CONF 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2：μ 律解压缩
- 3：μ 律压缩

至此，数据格式控制部分全部完成，数据通过 GDMA 存入内存。

35.11 事件任务矩阵功能

在 ESP32-C5 中，I2S 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 I2S 的 ETM 任务，或者通过 I2S 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里

仅介绍与 I2S 相关的 ETM 任务和 ETM 事件。

I2S 可接收的 ETM 任务有：

- I2S0_TASK_START_TX：触发时启动 I2S TX 数据传输。
- I2S0_TASK_START_RX：触发时启动 I2S RX 数据传输。
- I2S0_TASK_STOP_TX：触发时停止 I2S TX 数据传输。
- I2S0_TASK_STOP_RX：触发时停止 I2S RX 数据传输。

I2S 可产生的 ETM 事件有：

- I2S0_EVT_TX_DONE：表示 I2S TX 发送完成，即 TX FIFO 中的数据全部发送完毕。
- I2S0_EVT_RX_DONE：根据寄存器 I2S_RX_STOP_MODE 的配置值决定此事件的触发条件：
  - 0：不会触发此事件；
  - 1：触发时表示 I2S RX 模块接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度；
  - 2：触发时表示 GDMA RX FIFO 满。
- I2S0_EVT_X_WORDS_SENT：表示 I2S TX 发送字数大于等于 I2S_ETM_TX_SEND_WORD_NUM 的值。
- I2S0_EVT_X_WORDS_RECEIVED：表示 I2S RX 接收字数大于等于 I2S_ETM_RX_RECEIVE_WORD_NUM 的值。

在具体应用中，可以配置 I2S 的 ETM 事件来触发 I2S 的 ETM 任务，例如配置 I2S0_EVT_X_WORDS_SENT 事件触发 I2S0_TASK_STOP_TX 任务，从而以 ETM 的方式完成停止 I2S 的操作。

35.12 I2S 中断

ESP32-C5 的 I2S 可以生成 I2S_INTR 中断信号并发送给中断矩阵。这一中断信号由以下 I2S 的内部中断源生成。

- I2S_TX_HUNG_INT：当发送数据超时，触发此中断。例如，I2S 配置为从机发送模式，但主机长时间未提供 BCK 或 WS 信号，则将触发该中断。超时配置见寄存器 I2S_LC_HUNG_CONF_REG。
- I2S_RX_HUNG_INT：当接收数据超时，触发此中断。例如，I2S 配置为从机接收模式，但主机长时间未发送数据，则将触发该中断。超时配置见寄存器 I2S_LC_HUNG_CONF_REG。
- I2S_TX_DONE_INT：当发送数据完成，即 I2S TX FIFO 中的数据全部发送完毕时，触发此中断。
- I2S_RX_DONE_INT：根据寄存器 I2S_RX_DONE_MODE 的配置值决定中断触发方式：
  - 0：当 I2S RX 模块接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时，触发此中断；
  - 1：当 GDMA RX FIFO 满时，触发此中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 35.14 寄存器列表。

35.13 软件配置流程

35.13.1 软件配置 I2S 发送流程

软件配置 I2S 发送的流程如下：

1. 根据 35.6 小节的描述，配置时钟。
2. 根据表 35.4-1 的描述，配置信号管脚。
3. 根据主从机模式配置 I2S_TX_SLAVE_MOD：
   - 0：主机发送模式
   - 1：从机发送模式
4. 根据 35.9 小节的描述，配置正确的发送数据模式和发送通道模式，置位 I2S_TX_UPDATE。
5. 根据 35.7 小节的描述，复位 TX 模块和发送 FIFO。
6. 根据 35.12 小节的描述使能相应的中断。
7. 配置 GDMA 发送链表。
8. 根据需要置位 I2S_TX_STOP_EN，更多信息见章节 35.8.1。
9. 开始发送数据：
   - 主机模式下，等待 I2S 从设备配置完成后，置位 I2S_TX_START 开始发送数据；
   - 从机模式下，置位 I2S_TX_START。I2S 主设备提供 BCK 和 WS 信号后，开始发送数据。
10. 等待步骤 6 设置的中断信号，或查询 I2S_TX_IDLE 检查传输是否结束：
    - 0：发送设备为工作状态；
    - 1：发送设备为空闲状态。
11. 清零 I2S_TX_START 停止发送数据。

35.13.2 软件配置 I2S 接收流程

软件配置 I2S 接收模式的流程如下：

1. 根据 35.6 小节的描述，配置时钟。
2. 根据表 35.4-1 的描述，配置信号管脚。
3. 配置 I2S_RX_SLAVE_MOD 选择需要的模式：
   - 0：主机接收模式
   - 1：从机接收模式
4. 根据 35.10 小节的描述，配置正确的接收通道模式和接收数据模式，置位 I2S_RX_UPDATE。
5. 根据 35.7 小节的描述，复位 RX 模块和接收 FIFO。
6. 根据 35.12 小节的描述使能相应的中断。
7. 配置 GDMA 接收链表，并在 I2S_RX_EOF_NUM_REG 中配置接收数据长度。

8. 开始接收数据：
   - 在主机模式下，等待从机准备好后，置位 I2S_RX_START 开始接收数据；
   - 在从机模式下，置位 I2S_RX_START，等待主机提供 BCK 和 WS 信号后开始接收数据。
9. 接收的数据由 GDMA 根据配置存到存储器的指定地址。最终产生步骤 6 中设置的中断。

35.14 寄存器列表

本小节的所有地址均为相对于 I2S 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

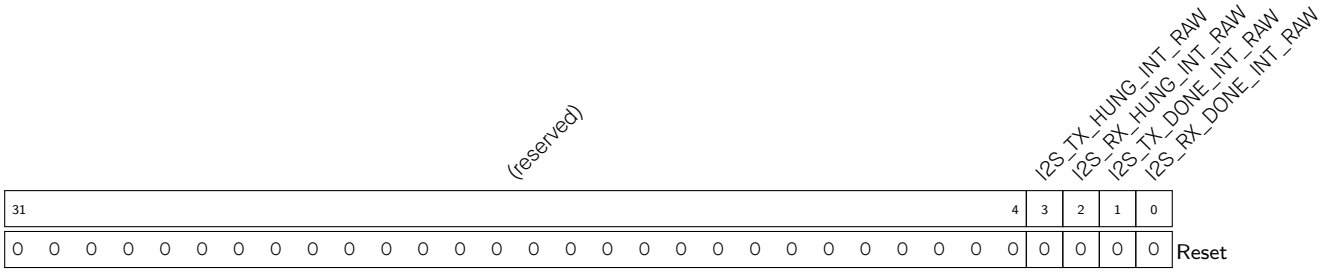
名称	描述	地址	访问
中断寄存器			
I2S_INT_RAW_REG	I2S 原始中断寄存器	0x000C	R/SS/WTC
I2S_INT_ST_REG	I2S 中断状态寄存器	0x0010	RO
I2S_INT_ENA_REG	I2S 中断使能寄存器	0x0014	R/W
I2S_INT_CLR_REG	I2S 中断清除寄存器	0x0018	WT
RX 控制和配置寄存器			
I2S_RX_CONF_REG	I2S RX 配置寄存器	0x0020	varies
I2S_RX_CONF1_REG	I2S RX 配置寄存器 1	0x0028	R/W
I2S_RX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0050	R/W
I2S_RXEOF_NUM_REG	I2S RX 数据大小控制寄存器	0x0064	R/W
TX 控制和配置寄存器			
I2S_TX_CONF_REG	I2S TX 配置寄存器	0x0024	varies
I2S_TX_CONF1_REG	I2S TX 配置寄存器 1	0x002C	R/W
I2S_TX_PCM2PDM_CONF_REG	I2S TX PCM-to-PDM 配置寄存器	0x0044	R/W
I2S_TX_PCM2PDM_CONF1_REG	I2S TX PCM-to-PDM 配置寄存器	0x0048	R/W
I2S_TX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0054	R/W
RX 时钟和时序寄存器			
I2S_RX_TIMING_REG	I2S RX 时序控制寄存器	0x0058	R/W
TX 时钟和时序寄存器			
I2S_TX_TIMING_REG	I2S TX 时序控制寄存器	0x005C	R/W
控制和配置寄存器			
I2S_LC_HUNG_CONF_REG	I2S 超时配置寄存器	0x0060	R/W
I2S_CONF_SIGLE_DATA_REG	I2S single 数据寄存器	0x0068	R/W
TX 状态寄存器			
I2S_STATE_REG	I2S TX 状态寄存器	0x006C	RO
ETM 寄存器			
I2S_ETM_CONF_REG	I2S ETM 配置寄存器	0x0070	R/W
同步计数器寄存器			
I2S_FIFO_CNT_REG	I2S 同步计数器寄存器	0x0074	varies
I2S_BCK_CNT_REG	I2S 同步计数器寄存器	0x0078	varies
时钟寄存器			
I2S_CLK_GATE_REG	时钟门控寄存器	0x007C	R/W
版本控制寄存器			
I2S_DATE_REG	版本控制寄存器	0x0080	R/W

35.15 寄存器

本小节的所有地址均为相对于 I2S 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 35.1. I2S_INT_RAW_REG (0x000C)



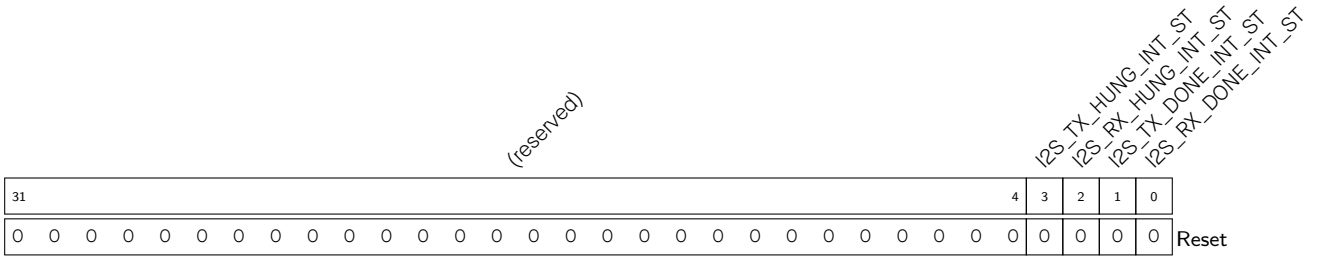
I2S_RX_DONE_INT_RAW I2S_RX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

I2S_TX_DONE_INT_RAW I2S_TX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

I2S_RX_HUNG_INT_RAW I2S_RX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)

I2S_TX_HUNG_INT_RAW I2S_TX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)

Register 35.2. I2S_INT_ST_REG (0x0010)



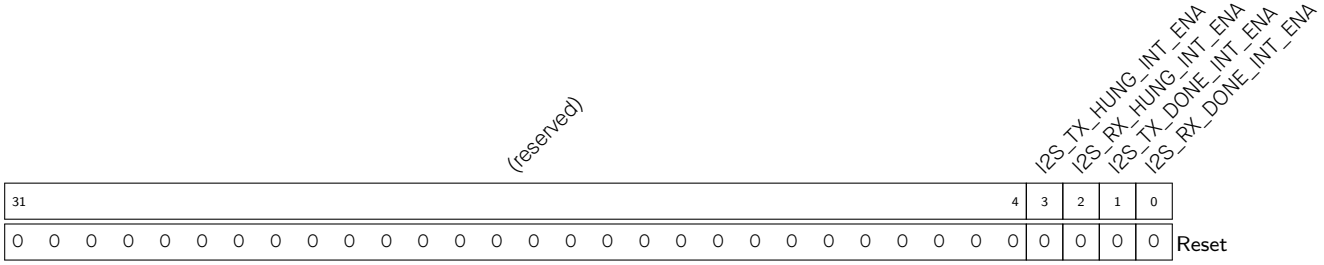
I2S_RX_DONE_INT_ST I2S_RX_DONE_INT 中断的屏蔽中断状态。(RO)

I2S_TX_DONE_INT_ST I2S_TX_DONE_INT 中断的屏蔽中断状态。(RO)

I2S_RX_HUNG_INT_ST I2S_RX_HUNG_INT 中断的屏蔽中断状态。(RO)

I2S_TX_HUNG_INT_ST I2S_TX_HUNG_INT 中断的屏蔽中断状态。(RO)

Register 35.3. I2S_INT_ENA_REG (0x0014)



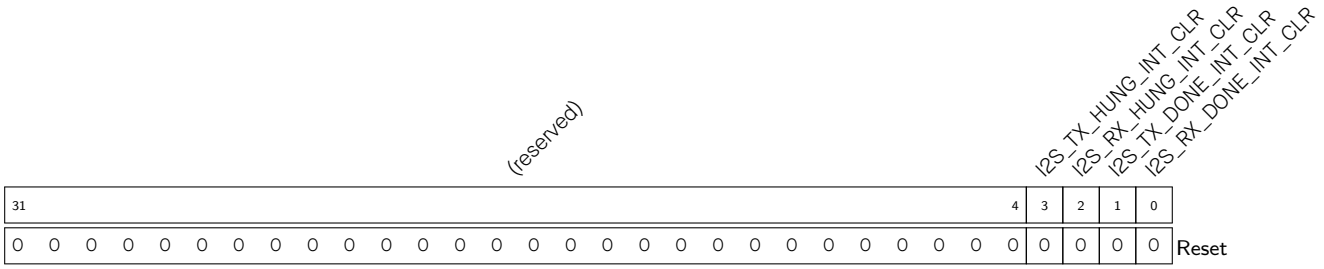
I2S_RX_DONE_INT_ENA 写 1 使能 I2S_RX_DONE_INT。(R/W)

I2S_TX_DONE_INT_ENA 写 1 使能 I2S_TX_DONE_INT。(R/W)

I2S_RX_HUNG_INT_ENA 写 1 使能 I2S_RX_HUNG_INT。(R/W)

I2S_TX_HUNG_INT_ENA 写 1 使能 I2S_TX_HUNG_INT。(R/W)

Register 35.4. I2S_INT_CLR_REG (0x0018)



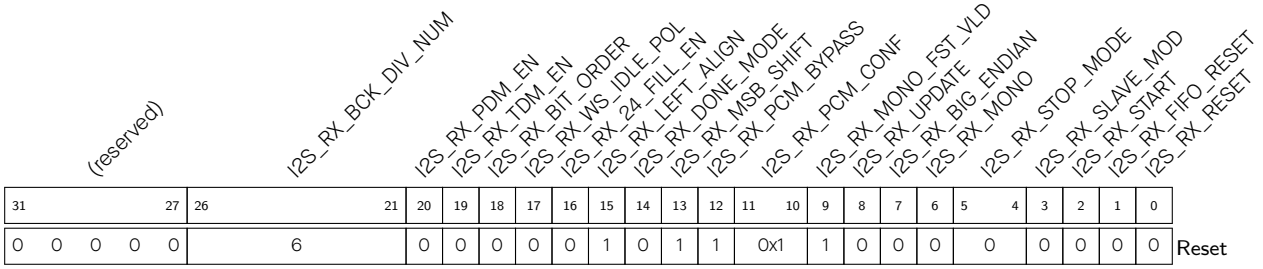
I2S_RX_DONE_INT_CLR 写 1 清除 I2S_RX_DONE_INT。(WT)

I2S_TX_DONE_INT_CLR 写 1 清除 I2S_TX_DONE_INT。(WT)

I2S_RX_HUNG_INT_CLR 写 1 清除 I2S_RX_HUNG_INT。(WT)

I2S_TX_HUNG_INT_CLR 写 1 清除 I2S_TX_HUNG_INT。(WT)

Register 35.5. I2S_RX_CONF_REG (0x0020)



I2S_RX_RESET 配置是否复位接收单元。

0：无效

1：复位

(WT)

I2S_RX_FIFO_RESET 配置是否复位 RX FIFO。

0：无效

1：复位

(WT)

I2S_RX_START 配置是否开始接收数据。

0：无效

1：开始

(R/W/SC)

I2S_RX_SLAVE_MOD 配置是否使能从机接收模式。

0：使能主机模式

1：使能从机模式

(R/W)

I2S_RX_STOP_MODE 配置 I2S RX 何时暂停接收。

0：只有 I2S_RX_START 被清除时暂停接收

1：I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收

2：接收单元会在 I2S_RX_START 被清除，或者 GDMA RX FIFO 满时暂停接收

(R/W)

I2S_RX_MONO 配置是否使能接收单元的单声道模式。

0：禁用

1：使能

(R/W)

I2S_RX_BIG_ENDIAN 配置 I2S RX 字节序。

0：低字节数据写入低位地址

1：低字节数据写入高位地址

(R/W)

见下页...

Register 35.5. I2S_RX_CONF_REG (0x0020)

接上页...

I2S_RX_UPDATE 配置是否将 I2S RX 寄存器从 APB 时钟域同步到 I2S RX 时钟域。寄存器完成更新后，该位将由硬件清除。

0：无效

1：同步

(R/W/SC)

I2S_RX_MONO_FST_VLD 配置 I2S RX 单声道模式下的有效数据通道。

0：第二个通道数据有效

1：第一个通道数据有效

(R/W)

I2S_RX_PCM_CONF 配置 I2S RX 压缩/解压缩模式。

0 (atol)：A 律解压缩

1 (ltoa)：A 律压缩

2 (utol)：μ 律解压缩

3 (ltou)：μ 律压缩

(R/W)

I2S_RX_PCM_BYPASS 配置接收数据是否将绕过压缩/解压缩单元。

0：无效

1：绕过压缩/解压缩单元

(R/W)

I2S_RX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0：上升沿对齐

1：相隔一个周期

(R/W)

I2S_RX_DONE_MODE 配置触发 I2S_RX_DONE_INT 中断的条件。

0：RX FIFO 满时

1：IN_SUC_EOF 为 1 时

(R/W)

I2S_RX_LEFT_ALIGN 配置 I2S RX 对齐模式。

0：右对齐模式

1：左对齐模式

(R/W)

I2S_RX_24_FILL_EN 配置 24 位通道数据位的储存数据位数。

0：将 24 位通道数据位保存到 24 位储存数据

1：将 24 位通道数据位保存到 32 位储存数据（多余的位填充为 0）

(R/W)

见下页...

Register 35.5. I2S_RX_CONF_REG (0x0020)

接上页...

I2S_RX_WS_IDLE_POL 配置 WS 与接收数据通道的关系。

0：WS 为 0 时，接收左通道数据；WS 为 1 时，接收右通道数据

1：WS 为 1 时，接收左通道数据；WS 为 0 时，接收右通道数据

(R/W)

I2S_RX_BIT_ORDER 配置是否调换 I2S RX 待接收数据的位顺序。

0：不调换

1：调换

(R/W)

I2S_RX_TDM_EN 配置是否使能 I2S TDM RX 模式。

0：禁用

1：使能

(R/W)

I2S_RX_PDM_EN 配置是否使能 I2S PDM RX 模式。

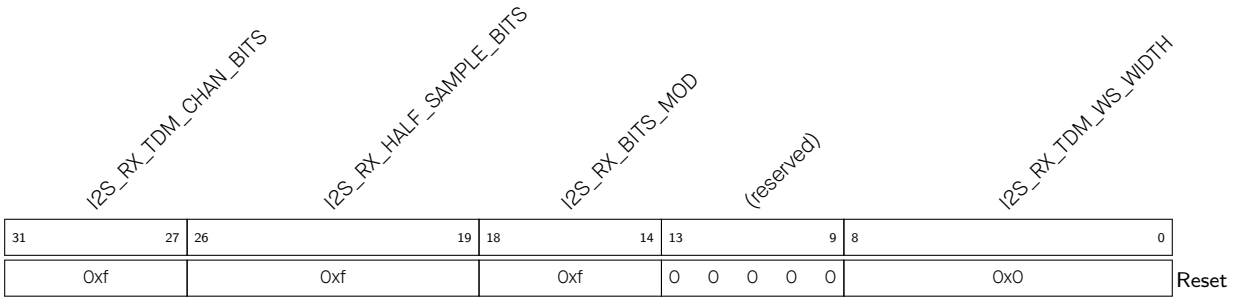
0：禁用

1：使能

(R/W)

I2S_RX_BCK_DIV_NUM 配置在 RX 模式下 BCK 时钟的分频系数。注意，不可配置为 1。(R/W)

Register 35.6. I2S_RX_CONF1_REG (0x0028)



I2S_RX_TDM_WS_WIDTH 配置 I2SI_WS_out（WS 默认电平）空闲时在 TDM 模式下的时长。I2SI_WS_out 空闲时在 TDM 模式下的时长 = (I2S_RX_TDM_WS_WIDTH[8:0] + 1) x T_BCK。(R/W)

I2S_RX_BITS_MOD 配置在 RX 模式下接收通道的有效数据位长度。

7：所有有效的通道数据均为 8 位模式

15：所有有效的通道数据均为 16 位模式

23：所有有效的通道数据均为 24 位模式

31：所有有效的通道数据均为 32 位模式

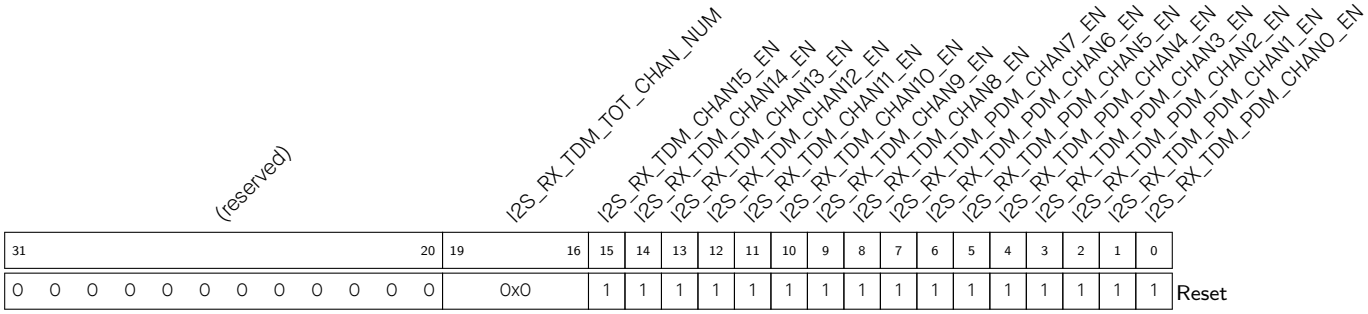
其他值无效。

(R/W)

I2S_RX_HALF_SAMPLE_BITS 配置 I2S RX 单次采样比特数。一个 WS 信号中 BCK 持续的周期 = I2S_RX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_RX_TDM_CHAN_BITS 配置在 TDM RX 模式下每个通道的 RX 数据位长。RX 数据位长 = I2S_RX_TDM_CHAN_BITS + 1。(R/W)

Register 35.7. I2S_RX_TDM_CTRL_REG (0x0050)



I2S_RX_TDM_PDM_CHAN*n*_EN (*n*：0-7) 配置是否使能 I2S TDM 或 PDM RX 通道 *n* 的有效输入数据。

0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

I2S_RX_TDM_CHAN*n*_EN (*n*：8-15) 配置是否使能 I2S TDM RX 通道 *n* 的有效输入数据。

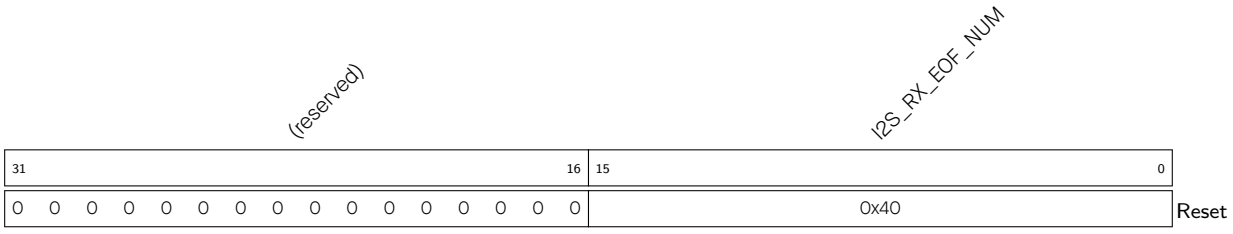
0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

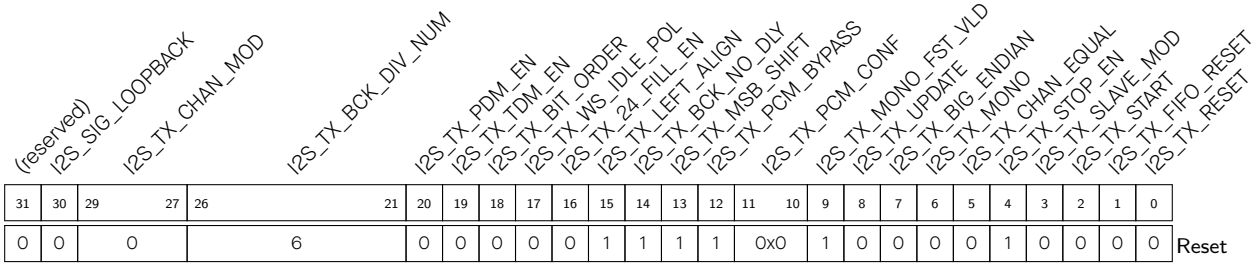
I2S_RX_TDM_TOT_CHAN_NUM 配置在 I2S TDM RX 模式下使用的通道总数。使用的通道总数 = I2S_RX_TDM_TOT_CHAN_NUM + 1。(R/W)

Register 35.8. I2S_RXEOF_NUM_REG (0x0064)



I2S_RX_EOF_NUM 配置接收数据的长度。接收数据长度 = (I2S_RX_BITS_MOD + 1) x (I2S_RX_EOF_NUM + 1)。如果接收数据的长度达到该值，则将在已配置的 GDMA RX 通道中触发 AHB_DMA_IN_SUC_EOF_CH*n*_INT 中断。(R/W)

Register 35.9. I2S_TX_CONF_REG (0x0024)



I2S_TX_RESET 配置是否复位发送单元。

0：无效

1：复位

(WT)

I2S_TX_FIFO_RESET 配置是否复位 TX FIFO。

0：无效

1：复位

(WT)

I2S_TX_START 配置是否开始发送数据。

0：无效

1：开始发送

(R/W/SC)

I2S_TX_SLAVE_MOD 配置是否使能从机发送模式。

0：使能主机模式

1：使能从机模式

(R/W)

I2S_TX_STOP_EN 配置当 TX FIFO 为空时，发送单元是否停止输出 BCK 和 WS 信号。

0：无效

1：停止

(R/W)

I2S_TX_CHAN_EQUAL 配置在 I2S TX 单声道模式或 TDM 模式下，是否使能左声道数据等于右声道数据。

0：在 I2S TX 单声道模式或 TDM 模式下，I2S_SINGLE_DATA 为无效的通道数据

1：在 I2S TX 单声道模式或 TDM 模式下，左声道数据等于右声道数据

(R/W)

I2S_TX_MONO 配置是否使能发送单元的单声道模式。

0：禁用

1：使能

(R/W)

I2S_TX_BIG_ENDIAN 配置 I2S TX 字节序。

0：低字节数据写入低位地址

1：低字节数据写入高位地址

(R/W)

见下页...

Register 35.9. I2S_TX_CONF_REG (0x0024)

接上页...

I2S_TX_UPDATE 配置是否将 I2S TX 寄存器从 APB 时钟域同步到 I2S TX 时钟域。寄存器更新完成后，此位将由硬件清除。

0：无效

1：同步

(R/W/SC)

I2S_TX_MONO_FST_VLD 配置在 I2S TX 单声道模式下的有效数据通道。

0：第二个通道数据有效

1：第一个通道数据有效

(R/W)

I2S_TX_PCM_CONF 配置 I2S TX 压缩/解压缩模式。

0 (atol)：A 律解压缩

1 (ltoa)：A 律压缩

2 (utol)：μ 律解压缩

3 (ltou)：μ 律压缩

(R/W)

I2S_TX_PCM_BYPASS 配置是否发送数据将绕过压缩/解压缩单元。

0：无效

1：绕过

(R/W)

I2S_TX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0：上升沿对齐

1：相隔一个周期

(R/W)

I2S_TX_BCK_NO_DLY 配置在主机模式下，BCK 上升沿和下降沿生成来源。

0：用 I2SO_BCK_in 输入的 BCK 构建上升沿和下降沿

1：用 I2S TX 模块时钟分频构建上升沿和下降沿

(R/W)

I2S_TX_LEFT_ALIGN 配置 I2S TX 对齐模式。

0：右对齐模式

1：左对齐模式

(R/W)

I2S_TX_24_FILL_EN 配置 24 位数据的发送格式。

0：将 24 位数据以 24 位的格式发送出去

1：将 24 位数据以 32 位的格式发送出去（不足的位用 0 填充）

(R/W)

见下页...

Register 35.9. I2S_TX_CONF_REG (0x0024)

接上页...

I2S_TX_WS_IDLE_POL 配置 WS 与发送数据通道的关系。

0：WS 为 0 时，发送左通道数据；WS 为 1 时，发送右通道数据

1：WS 为 1 时，发送左通道数据；WS 为 0 时，发送右通道数据

I2S_TX_BIT_ORDER 配置是否调换 I2S TX 待发送有效数据的位顺序。

0：不调换

1：调换

(R/W)

I2S_TX_TDM_EN 配置是否使能 I2S TDM TX 模式。

0：禁用

1：使能

(R/W)

I2S_TX_PDM_EN 配置是否使能 I2S PDM TX 模式。

0：禁用

1：使能

(R/W)

I2S_TX_BCK_DIV_NUM 配置在 TX 模式下 BCK 时钟的分频系数。注意，不可配置为 1。(R/W)

I2S_TX_CHAN_MOD 配置 I2S 发送单元通道，更多信息见表 35.9-4。(R/W)

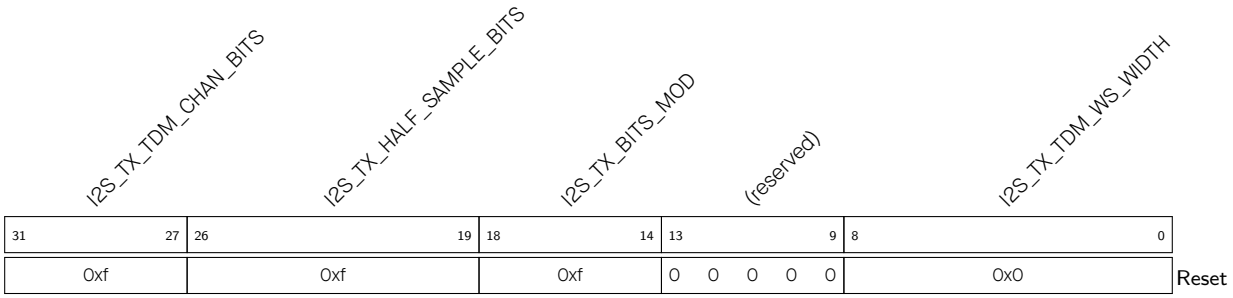
I2S_SIG_LOOPBACK 配置是否使能发送单元和接收单元共享 WS 和 BCK 信号。

0：禁用

1：使能

(R/W)

Register 35.10. I2S_TX_CONF1_REG (0x002C)



I2S_TX_TDM_WS_WIDTH 配置 I2SO_WS_out（WS 默认电平）空闲时在 TDM 模式下的时长。I2SO_WS_out 空闲时在 TDM 模式下的时长 = (I2S_TX_TDM_WS_WIDTH[8:0] +1) x T_BCK。(R/W)

I2S_TX_BITS_MOD 配置 TX 模式下发送通道的有效数据位长度。

7：所有有效的通道数据均为 8 位模式

15：所有有效的通道数据均为 16 位模式

23：所有有效的通道数据均为 24 位模式

31：所有有效的通道数据均为 32 位模式

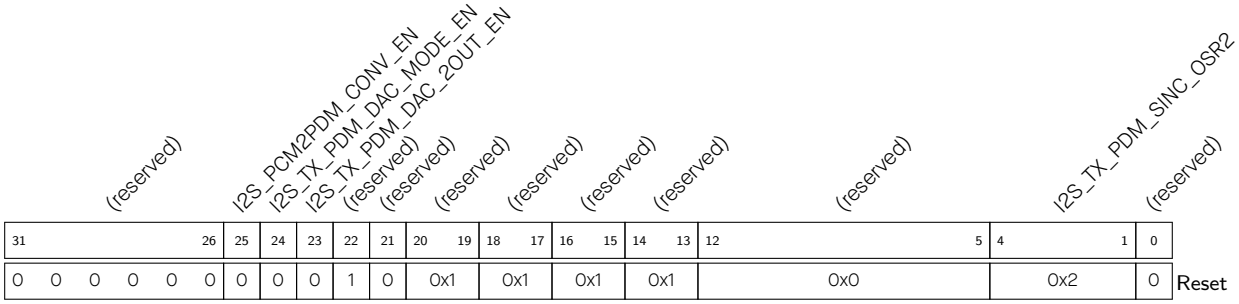
其他值无效。

(R/W)

I2S_TX_HALF_SAMPLE_BITS 配置 TX 单次采样比特数。一个 WS 信号中 BCK 持续的周期长 = I2S_TX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_TX_TDM_CHAN_BITS 配置在 TDM TX 模式下每个通道的 TX 数据位。TX 数据位 = I2S_TX_TDM_CHAN_BITS + 1。(R/W)

Register 35.11. I2S_TX_PCM2PDM_CONF_REG (0x0044)



I2S_TX_PDM_SINC_OSR2 配置 I2S TX PDM 的 OSR 值。(R/W)

I2S_TX_PDM_DAC_2OUT_EN 配置 DAC 输出模式。

0：使能 1-line DAC 输出模式

1：使能 2-line DAC 输出模式

仅在 I2S_TX_PDM_DAC_MODE_EN 为 1 时有效。

(R/W)

I2S_TX_PDM_DAC_MODE_EN 配置使能 1-line PDM 输出模式或 DAC 输出模式。

0：使能 1-line PDM 输出模式

1：使能 DAC 输出模式

(R/W)

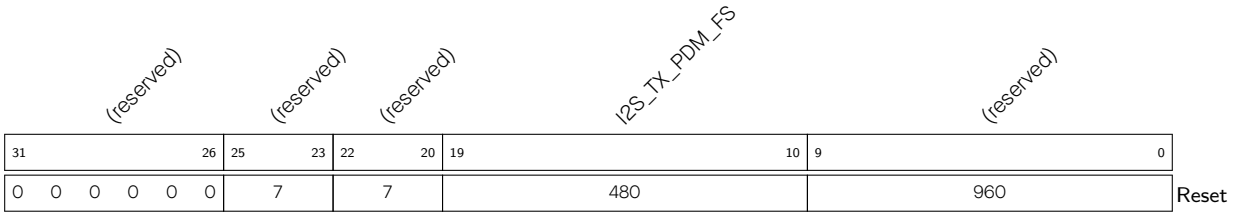
I2S_PCM2PDM_CONV_EN 配置是否使能 I2S TX PCM-to-PDM 转换器。

0: 禁用

1: 使能

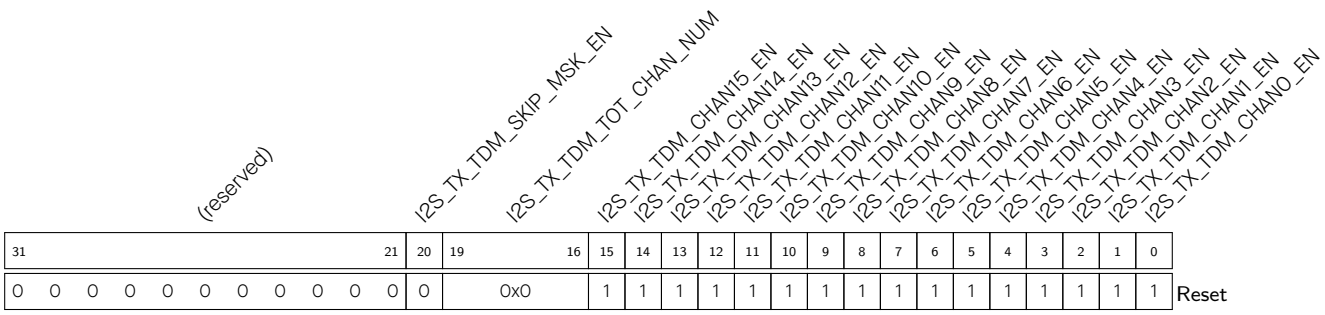
(R/W)

Register 35.12. I2S_TX_PCM2PDM_CONF1_REG (0x0048)



I2S_TX_PDM_FS 配置 I2S TX PDM 上采样率。(R/W)

Register 35.13. I2S_TX_TDM_CTRL_REG (0x0054)



I2S_TX_TDM_CHAN*n*_EN (*n*: 0-15) 配置是否使能 I2S TDM TX 通道 *n* 的有效输出数据。

0：通道发送数据由 I2S_TX_CHAN_EQUAL 和 I2S_SINGLE_DATA 控制，参考章节 35.9.2.1

1：使能

(R/W)

I2S_TX_TDM_TOT_CHAN_NUM 配置在 I2S TDM TX 模式下使用的通道总数。通道总数 = I2S_TX_TDM_TOT_CHAN_NUM + 1。(R/W)

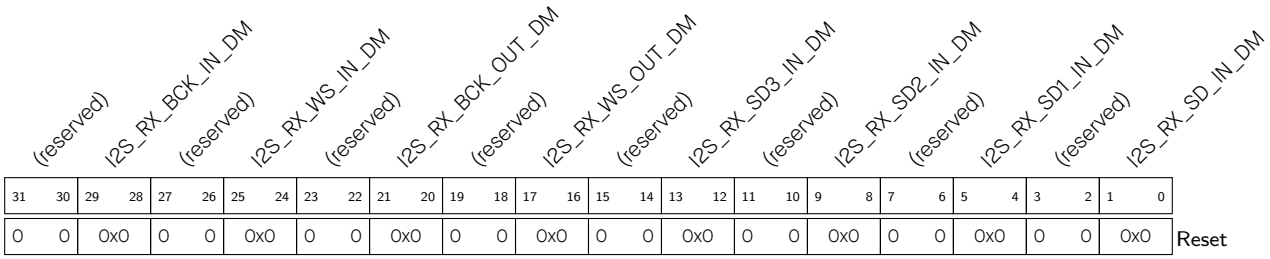
I2S_TX_TDM_SKIP_MSK_EN 配置 GDMA TX buffer 的发送数据。

0：GDMA TX buffer 中的数据都用于已启用的通道，未启用的通道发送时不会读取 GDMA TX buffer 中的数据

1：GDMA TX buffer 中的数据在所有通道发送时均会被读取，因此在未启用的通道发送时 GDMA TX buffer 中的数据会被跳过

(R/W)

Register 35.14. I2S_RX_TIMING_REG (0x0058)



I2S_RX_SD_IN_DM 配置 I2S RX SD 输入信号的延迟模式。

0：旁路

1：上升沿延迟

2：下降沿延迟

3：无效

(R/W)

I2S_RX_SD1_IN_DM 配置 I2S RX SD1 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_SD2_IN_DM 配置 I2S RX SD2 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_SD3_IN_DM 配置 I2S RX SD3 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

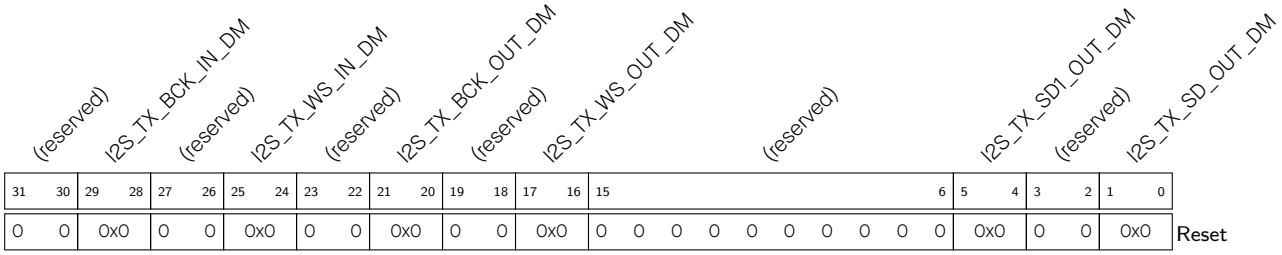
I2S_RX_WS_OUT_DM 配置 I2S RX WS 输出信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_BCK_OUT_DM 配置 I2S RX BCK 输出信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_WS_IN_DM 配置 I2S RX WS 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_BCK_IN_DM 配置 I2S RX BCK 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

Register 35.15. I2S_TX_TIMING_REG (0x005C)



I2S_TX_SD_OUT_DM 配置 I2S TX SD 输出信号的延迟模式。

0：旁路

1：上升沿延迟

2：下降沿延迟

3：无效

(R/W)

I2S_TX_SD1_OUT_DM 配置 I2S TX SD1 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

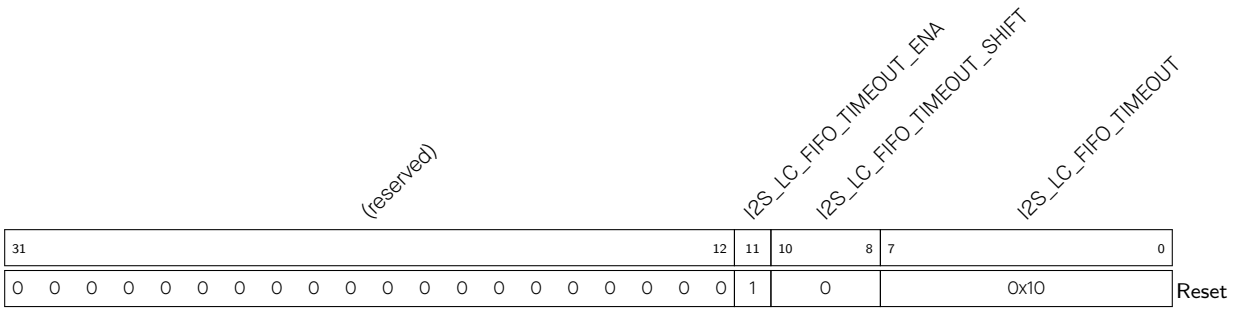
I2S_TX_WS_OUT_DM 配置 I2S TX WS 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_OUT_DM 配置 I2S TX BCK 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_WS_IN_DM 配置 I2S TX WS 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_IN_DM 配置 I2S TX BCK 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

Register 35.16. I2S_LC_HUNG_CONF_REG (0x0060)



I2S_LC_FIFO_TIMEOUT 配置 FIFO 的超时阈值。FIFO Hung 计数器在 I2S 处于错误状态，并且每次分频滴答计数器达到滴答计数器的阈值时会加一。FIFO Hung 计数器等于 FIFO 的超时阈值时，将触发 I2S_TX_HUNG_INT 中断或 I2S_RX_HUNG_INT 中断。(R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT 配置分频滴答计数器的阈值。分频滴答计数器会在 I2S 处于错误状态，并且每个系统 (APB) 时钟上升沿加一。计数器值大于等于 $88000/2^{I2S\_LC\_FIFO\_TIMEOUT\_SHIFT}$ 时，复位滴答计数器。(R/W)

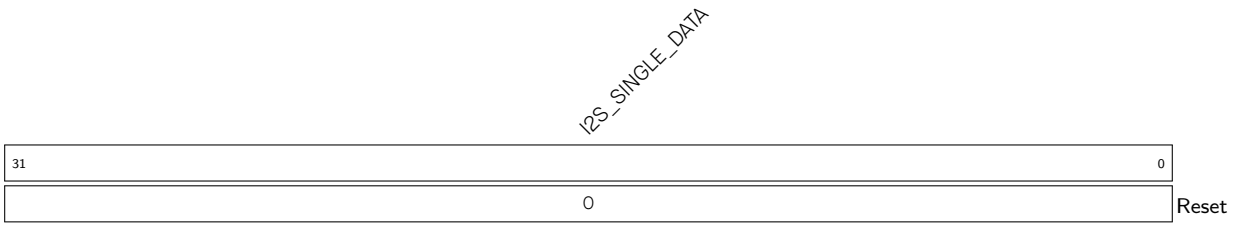
I2S_LC_FIFO_TIMEOUT_ENA 配置是否使能 FIFO 超时。

0：禁用

1：使能

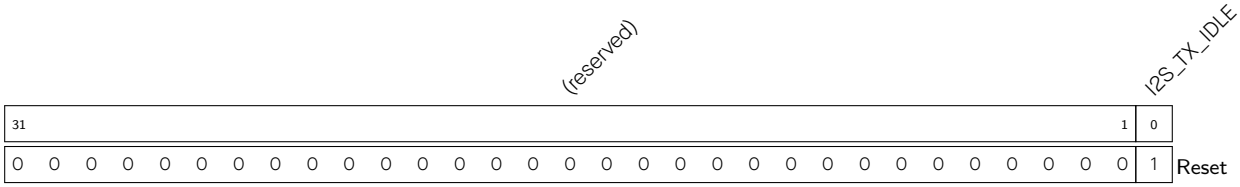
(R/W)

Register 35.17. I2S_CONF_SIGLE_DATA_REG (0x0068)



I2S_SINGLE_DATA 配置用于发送的通道常量数据。(R/W)

Register 35.18. I2S_STATE_REG (0x006C)



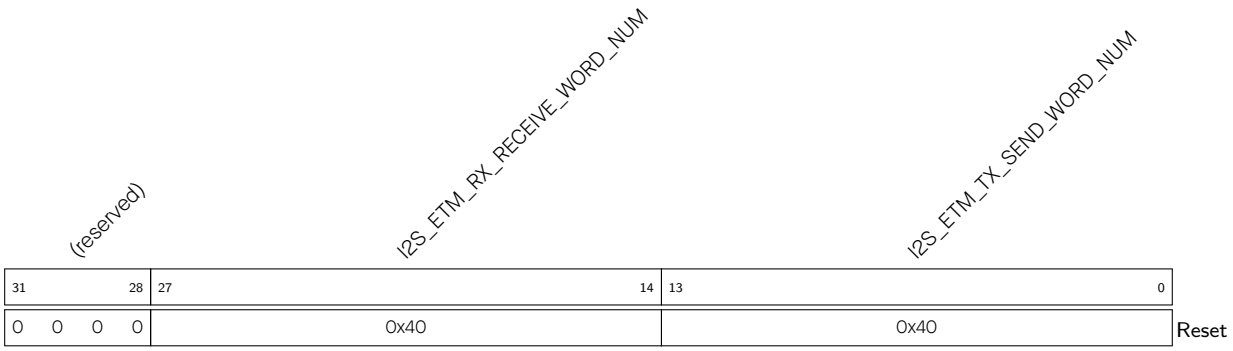
I2S_TX_IDLE 表示 I2S TX 单元的工作状态。

0：I2S TX 单元处于工作状态

1：I2S TX 单元处于空闲状态

(RO)

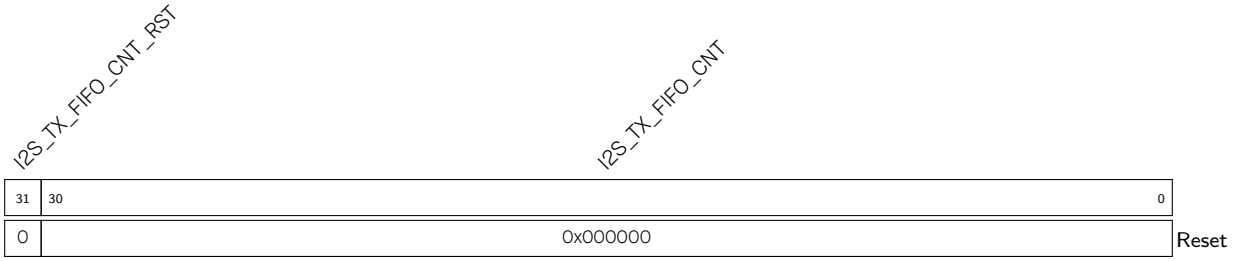
Register 35.19. I2S_ETM_CONF_REG (0x0070)



I2S_ETM_TX_SEND_WORD_NUM 配置触发 I2S_TX_X_WORDS_SENT 事件的阈值。当发送字数为 I2S_ETM_TX_SEND_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

I2S_ETM_RX_RECEIVE_WORD_NUM 配置触发 ETM I2S_RX_X_WORDS_RECEIVED 事件的阈值。当接收字数为 I2S_ETM_RX_RECEIVE_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

Register 35.20. I2S_FIFO_CNT_REG (0x0074)



I2S_TX_FIFO_CNT 配置 TX FIFO 计数值。(RO)

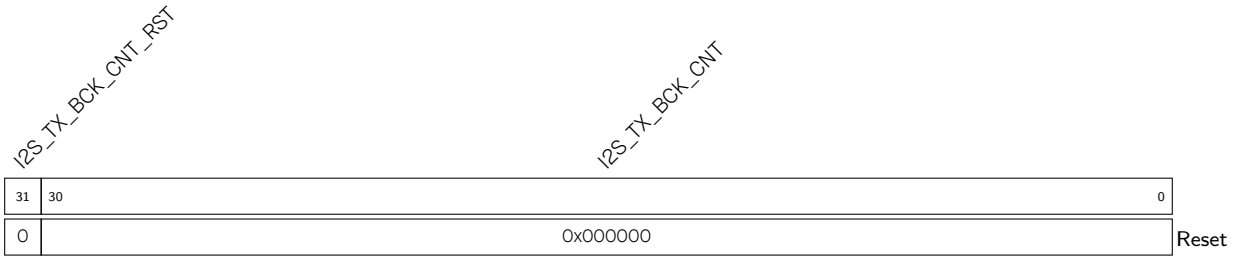
I2S_TX_FIFO_CNT_RST 配置是否复位 TX FIFO 计数器。

0：无效

1：复位

(WT)

Register 35.21. I2S_BCK_CNT_REG (0x0078)



I2S_TX_BCK_CNT 配置 TX BCK 计数值。(RO)

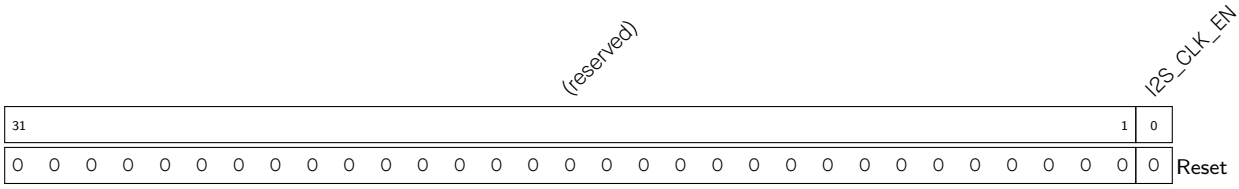
I2S_TX_BCK_CNT_RST 配置是否复位 TX BCK 计数器。

0：无效

1：复位

(WT)

Register 35.22. I2S_CLK_GATE_REG (0x007C)



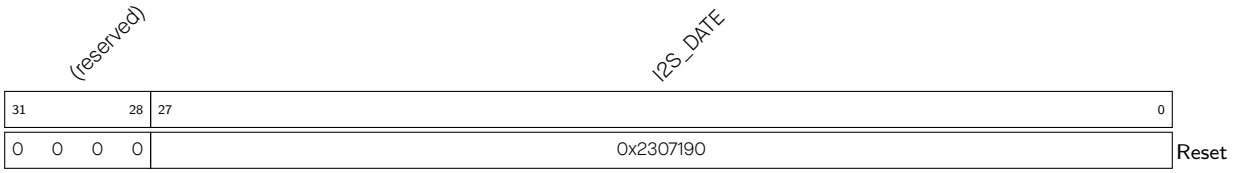
I2S_CLK_EN 配置是否启用时钟门控。

0：禁用

1：启用

(R/W)

Register 35.23. I2S_DATE_REG (0x0080)



I2S_DATE 版本控制寄存器。(R/W)

第 36 章

脉冲计数控制器 (PCNT)

脉冲计数控制器 (Pulse Count Controller, PCNT) 用于对输入脉冲计数，通过记录输入脉冲信号的上升沿或下降沿进行递增或递减计数。PCNT 有四个称为“单元”的独立脉冲计数控制器，这些单元拥有自己的寄存器。PCNT 模块仅有一个时钟，为 APB_CLK。下文描述中 *n* 表示单元编号 0 ~ 3。

每个单元有两个通道（ch0 和 ch1），可以独立配置为递增或递减计数。两个通道功能相同，下文以通道 0 (ch0) 为例进行介绍。

如图 36.0-1 所示，每个通道有两个输入信号：

1. 一个脉冲输入信号（如 sig_ch0_u*n* 为单元 *n* ch0 的脉冲输入信号）
2. 一个控制信号（如 ctrl_ch0_u*n* 为单元 *n* ch0 的控制信号）

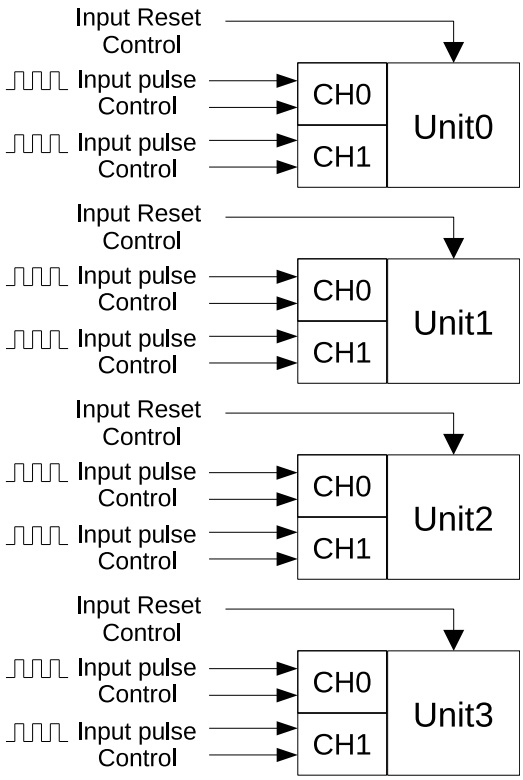


图 36.0-1. PCNT 框图

36.1 主要特性

PCNT 有如下特性：

- 四个脉冲计数控制器（单元），各自独立工作，计数范围是 1 ~ 65535

- 每个单元有两个独立的通道，共用一个脉冲计数控制器
- 所有通道均有输入脉冲信号（如 sig_ch0_u*n*）和相应的控制信号（如 ctrl_ch0_u*n*）
- 滤波器独立工作，过滤每个单元输入脉冲信号（sig_ch0_u*n* 和 sig_ch1_u*n*）控制信号（ctrl_ch0_u*n* 和 ctrl_ch1_u*n*）的毛刺
- 每个通道参数如下：
  1. 选择在输入脉冲信号的上升沿或下降沿计数
  2. 在控制信号为高电平或低电平时可将计数模式配置为递增、递减或停止计数
  3. 通过设置递增/递减计数步长阈值实现步长计数警报
  4. 置位清零寄存器或通过 GPIO 输入清零信号可清除脉冲计数控制器的值
  5. 每个计数模式都可以生成并记录相应事件信号，这些事件信号可以配置为中断
- 最大输入脉冲频率：$\frac{f_{APB\_CLK}}{2}$

36.2 功能描述

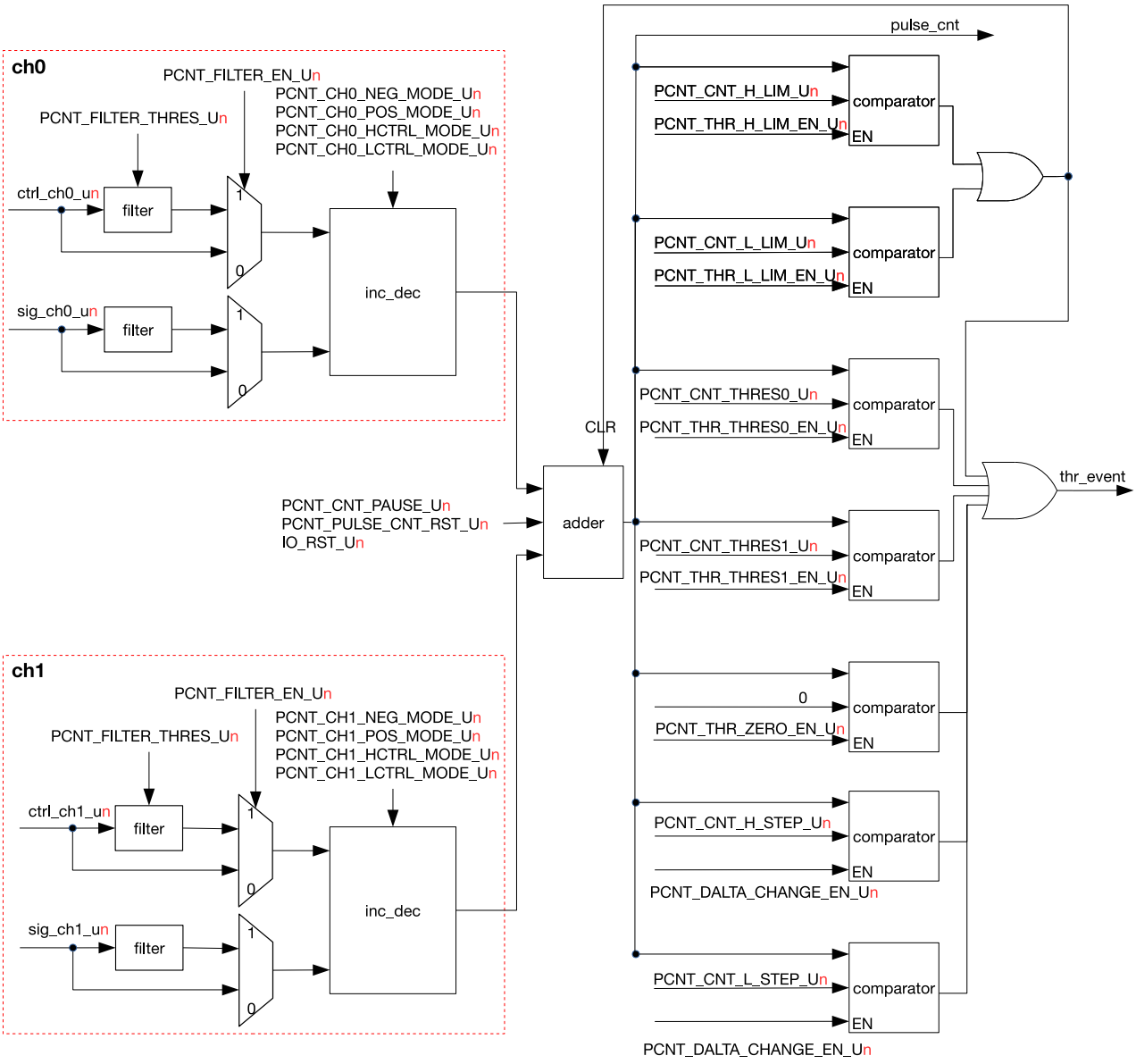


图 36.2-1. PCNT 单元基本架构图

图 36.2-1 为 PCNT 单元的基本架构图。如上所述，ctrl_ch0_u*n* 为单元 *n* ch0 的控制信号，控制信号 ctrl_ch0_u*n* 高电平或低电平可分别配置为不同的计数模式，用于在输入脉冲信号 sig_ch0_u*n* 的上升沿和下降沿计数。可选计数模式如下：

- 递增模式：通道检测到 sig_ch0_u*n* 的有效边沿（该边沿可通过软件配置）时，计数器的值 pulse_cnt 加 1。当 pulse_cnt 的值达到 PCNT_CNT_H_LIM_U*n* 时被清零。如果在 pulse_cnt 达到 PCNT_CNT_H_LIM_U*n* 前修改了该通道的计数模式或 PCNT_CNT_PAUSE_U*n* 置 1，则 pulse_cnt 停止计数，并切换计数模式。
- 递减模式：通道检测到 sig_ch0_u*n* 的有效边沿（该边沿可通过软件配置）时，计数器的值 pulse_cnt 减 1。pulse_cnt 的值达到 PCNT_CNT_L_LIM_U*n* 时被清零。如果在 pulse_cnt 达到 PCNT_CNT_L_LIM_U*n* 前修改了该通道的计数模式或 PCNT_CNT_PAUSE_U*n* 置 1，则 pulse_cnt 停止计数，并切换计数模式。

- 停止计数：计数停止，计数器的值 pulse_cnt 保持不变。

表 36.2-1 至表 36.2-4 说明了如何配置通道 0 的计数模式。

表 36.2-1. 控制信号为低电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 36.2-2. 控制信号为高电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 36.2-3. 控制信号为低电平时输入脉冲信号下降沿的计数模式

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 36.2-4. 控制信号为高电平时输入脉冲信号下降沿的计数模式

PCNT_CHO_NEG_MODE_U*n*	PCNT_CHO_HCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

每个单元均有一个滤波器，用于该单元的所有控制信号和输入脉冲信号。置位 PCNT_FILTER_EN_U*n* 位使能滤波器。滤波器监测信号，滤除脉冲宽度小于 PCNT_FILTER_THRES_U*n* 个 APB 时钟周期的线路毛刺。

如前文所述，每个单元包含通道 0 和通道 1，分别处理不同的输入脉冲信号，并通过各自的 inc_dec 模块进行加减计数。之后，两个通道将计数值发送给加法器 adder 模块，该模块有一个带符号位的 16 位宽寄存器。软件可以设置寄存器 PCNT_CNT_PAUSE_U*n* 来暂停加法器的计数行为，也可以通过设置寄存器 PCNT_PULSE_CNT_RST_U*n* 清零加法器：

脉冲计数控制器（PCNT）可以设置七个计数事件观察点，这些观察点共享一个中断。每个观察点都有独立的中断使能信号，用于开启或屏蔽中断。

以下列出 PCNT 可以配置上报的计数事件的触发方式和对应的中断事件：

- 最大计数值：当 pulse_cnt 大于等于 PCNT_CNT_H_LIM_U*n* 时，产生上限中断，同时 PCNT_CNT_THR_H_LIM_LAT_U*n* 为高。
- 最小计数值：当 pulse_cnt 小于等于 PCNT_CNT_L_LIM_U*n* 时，产生下限中断，同时 PCNT_CNT_THR_L_LIM_LAT_U*n* 为高。
- 两个中间阈值：当 pulse_cnt 等于 PCNT_CNT_THRESO_U*n* 或者 PCNT_CNT_THRES1_U*n* 时，产生中断，同时 PCNT_CNT_THR_THRESO_LAT_U*n* 或 PCNT_CNT_THR_THRES1_LAT_U*n* 为高。
- 零：当 pulse_cnt 等于 0 时，产生中断，同时 PCNT_CNT_THR_ZERO_LAT_U*n* 有效。
- 递增计数步长阈值：如果 PCNT_DALTA_CHANGE_EN_U*n* 被设置为高电平，当在计数递增过程中 pulse_cnt 每增加 PCNT_CNT_H_STEP_U*n* 时，将触发递增计数步长中断，并将 PCNT_CNT_THR_H_STEP_LAT_U*n* 置为高电平。

  例如如果 PCNT_CNT_H_STEP_U*n* 设置为 100，那么当 pulse_cnt 脉冲数达到达到 100、200、300 等整数倍值，将分别触发中断一次。此类中断可用于周期性地检测输入脉冲数量是否达到预定倍数，便于实现定量采样、数据统计或或定期事件响应等功能。
- 递减计数步长阈值：如果 PCNT_DALTA_CHANGE_EN_U*n* 被设置为高电平，当在计数递减过程中 pulse_cnt 每减少 PCNT_CNT_L_STEP_U*n* 时，将触发递减计数步长中断，并将 PCNT_CNT_THR_L_STEP_LAT_U*n* 置为高电平。

  例如，如果 PCNT_CNT_L_STEP_U*n* 设置为 100，那么当 pulse_cnt 脉冲数递减至 100、200、300 等整数倍时，将分别触发中断一次。此类中断适用于周期性检测计数递减是否达到预定倍数，便于实现定量采样、数据统计或定期事件响应等功能。

在 PCNT 模块工作过程中，软件如果重新配置 PCNT_CNT_H_LIM_U*n* 和/或 PCNT_CNT_L_LIM_U*n*，则新的配置值会在 pulse_cnt 达到上述七个观察点的任意一个后再生效；软件如果重新配置 PCNT_CNT_THRESO_U*n*、

PCNT_CNT_THRES1_U*n*、PCNT_CNT_H_STEP_U*n* 和/或 PCNT_CNT_L_STEP_U*n*，则新的配置值会立即生效。

36.3 应用实例

每个单元的通道 0 和通道 1 可配置为独立工作或一起工作。下文详细说明了通道 0 独自递增计数、通道 0 独自递减计数和两个通道一起递增计数的应用实例。本节中未详述的通道工作模式（如通道 1 独自递减或递增、双通道一增一减），可参考这三种模式。

36.3.1 通道 0 独自递增计数

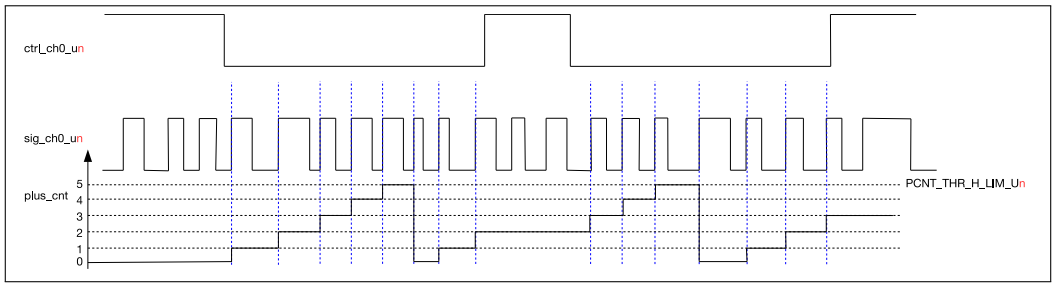


图 36.3-1. 通道 0 递增计数图

图 36.3-1 为通道 0 在 sig_ch0_u*n* 上升沿独立递增计数的示意图，此时通道 1 关闭（请参阅 36.2 一节查看如何关闭通道 1）。通道 0 的配置如下所示。

- PCNT_CH0_LCTRL_MODE_U*n*=0：当 ctrl_ch0_u*n* 为低电平时，递增计数。
- PCNT_CH0_HCTRL_MODE_U*n*=2：当 ctrl_ch0_u*n* 为高电平时，停止计数。
- PCNT_CH0_POS_MODE_U*n*=1：在 sig_ch0_u*n* 的上升沿递增计数。
- PCNT_CH0_NEG_MODE_U*n*=0：在 sig_ch0_u*n* 的下降沿不计数。
- PCNT_CNT_H_LIM_U*n*=5：pulse_cnt 的值递增至 PCNT_CNT_H_LIM_U*n* 时被清零。

36.3.2 通道 0 独自递减计数

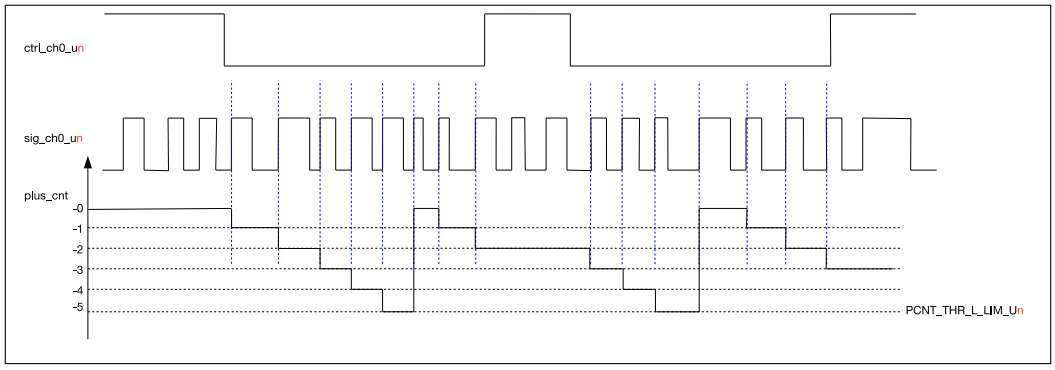


图 36.3-2. 通道 0 递减计数图

图 36.3-2 为通道 0 在 sig_ch0_u*n* 上升沿独立递减计数的示意图，此时通道 1 关闭。此时通道 0 的配置与图 36.3-1 相比有如下区别：

- PCNT_CH0_POS_MODE_U*n*=2：即在 sig_ch0_u*n* 的上升沿递减计数。
- PCNT_CNT_L_LIM_U*n*=-5：pulse_cnt 的值递减到 PCNT_CNT_L_LIM_U*n* 时被清零。

36.3.3 通道 0 和通道 1 同时递增计数

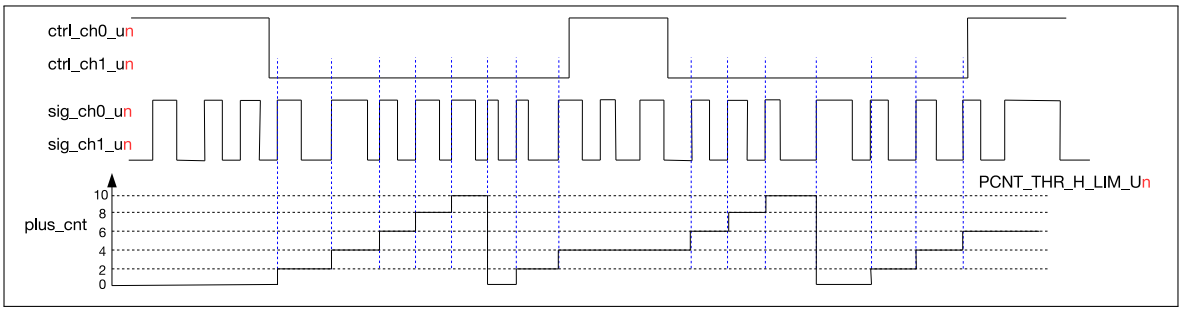


图 36.3-3. 双通道递增计数图

图 36.3-3 为通道 0 和通道 1 分别在 sig_ch0_u*n* 和 sig_ch1_u*n* 上升沿同时递增计数的示意图。如图 36.3-3 所示，控制信号 ctrl_ch0_u*n* 与 ctrl_ch1_u*n* 的波形一致，输入脉冲信号 sig_ch0_u*n* 和 sig_ch1_u*n* 波形一致。具体配置如下：

- 通道 0：
  - PCNT_CH0_LCTRL_MODE_U*n*=0：当 ctrl_ch0_u*n* 为低电平时，递增计数。
  - PCNT_CH0_HCTRL_MODE_U*n*=2：当 ctrl_ch0_u*n* 为高电平时，停止计数。
  - PCNT_CH0_POS_MODE_U*n*=1：在 sig_ch0_u*n* 的上升沿递增计数。
  - PCNT_CH0_NEG_MODE_U*n*=0：在 sig_ch0_u*n* 的下降沿不计数。
- 通道 1：
  - PCNT_CH1_LCTRL_MODE_U*n*=0：当 ctrl_ch1_u*n* 为低电平时，递增计数。
  - PCNT_CH1_HCTRL_MODE_U*n*=2：当 ctrl_ch1_u*n* 为高电平时，停止计数。
  - PCNT_CH1_POS_MODE_U*n*=1：在 sig_ch1_u*n* 的上升沿递增计数。
  - PCNT_CH1_NEG_MODE_U*n*=0：在 sig_ch1_u*n* 的下降沿不计数。
- PCNT_CNT_H_LIM_U*n*=10：pulse_cnt 递增至 PCNT_CNT_H_LIM_U*n* 时被清零。

36.4 寄存器列表

本小节的所有地址均为相对于 脉冲计数控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 6 *系统和存储器* 中的表 6.3-2。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
PCNT_U0_CONF0_REG	单元 0 的配置寄存器 0	0x0000	R/W
PCNT_U0_CONF1_REG	单元 0 的配置寄存器 1	0x0004	R/W

名称	描述	地址	访问
PCNT_U0_CONF2_REG	单元 0 的配置寄存器 2	0x0008	R/W
PCNT_U0_CONF3_REG	单元 0 的步长值配置寄存器	0x000C	R/W
PCNT_U1_CONF0_REG	单元 1 的配置寄存器 0	0x0010	R/W
PCNT_U1_CONF1_REG	单元 1 的配置寄存器 1	0x0014	R/W
PCNT_U1_CONF2_REG	单元 1 的配置寄存器 2	0x0018	R/W
PCNT_U1_CONF3_REG	单元 1 的步长值配置寄存器	0x001C	R/W
PCNT_U2_CONF0_REG	单元 2 的配置寄存器 0	0x0020	R/W
PCNT_U2_CONF1_REG	单元 2 的配置寄存器 1	0x0024	R/W
PCNT_U2_CONF2_REG	单元 2 的配置寄存器 2	0x0028	R/W
PCNT_U2_CONF3_REG	单元 2 的步长值配置寄存器	0x002C	R/W
PCNT_U3_CONF0_REG	单元 3 的配置寄存器 0	0x0030	R/W
PCNT_U3_CONF1_REG	单元 3 的配置寄存器 1	0x0034	R/W
PCNT_U3_CONF2_REG	单元 3 的配置寄存器 2	0x0038	R/W
PCNT_U3_CONF3_REG	单元 3 的步长值配置寄存器	0x003C	R/W
PCNT_CTRL_REG	所有计数器的控制寄存器	0x0070	R/W
状态寄存器			
PCNT_U0_CNT_REG	单元 0 的计数器值	0x0040	RO
PCNT_U1_CNT_REG	单元 1 的计数器值	0x0044	RO
PCNT_U2_CNT_REG	单元 2 的计数器值	0x0048	RO
PCNT_U3_CNT_REG	单元 3 的计数器值	0x004C	RO
PCNT_U0_STATUS_REG	脉冲计数器单元 0 的状态寄存器	0x0060	RO
PCNT_U1_STATUS_REG	脉冲计数器单元 1 的状态寄存器	0x0064	RO
PCNT_U2_STATUS_REG	脉冲计数器单元 2 的状态寄存器	0x0068	RO
PCNT_U3_STATUS_REG	脉冲计数器单元 3 的状态寄存器	0x006C	RO
中断寄存器			
PCNT_INT_RAW_REG	原始中断状态寄存器	0x0050	R/ WTC/ SS
PCNT_INT_ST_REG	中断状态寄存器	0x0054	RO
PCNT_INT_ENA_REG	中断使能寄存器	0x0058	R/W
PCNT_INT_CLR_REG	中断清除寄存器	0x005C	WT
版本控制寄存器			
PCNT_DATE_REG	脉冲计数器的版本控制寄存器	0x00FC	R/W

36.5 寄存器

本小节的所有地址均为相对于 脉冲计数控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 36.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0x10**n*)



PCNT_FILTER_THRES_U*n* 配置滤波器的最大阈值。滤波器启动时，任何小于该值的脉冲都会被过滤。

单位：APB_CLK 时钟周期。

(R/W)

PCNT_FILTER_EN_U*n* 单元 *n* 输入滤波器的使能位。(R/W)

PCNT_THR_ZERO_EN_U*n* 单元 *n* 过零比较器的使能位。(R/W)

PCNT_THR_H_LIM_EN_U*n* 单元 *n* 上限比较器的使能位，控制上限中断事件的产生。(R/W)

PCNT_THR_L_LIM_EN_U*n* 单元 *n* 下限比较器的使能位，控制下限中断事件的产生。(R/W)

PCNT_THR_THRESO_EN_U*n* 单元 *n* 阈值 0 比较器的使能位。(R/W)

PCNT_THR_THRES1_EN_U*n* 单元 *n* 阈值 1 比较器的使能位。(R/W)

PCNT_CH0_NEG_MODE_U*n* 配置通道 0 输入信号检测下降沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH0_POS_MODE_U*n* 配置通道 0 输入信号检测上升沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH0_HCTRL_MODE_U*n* 控制信号为高电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

见下页...

Register 36.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0x10**n*)

接上页...

PCNT_CH0_LCTRL_MODE_U*n* 控制信号为低电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

PCNT_CH1_NEG_MODE_U*n* 配置通道 1 输入信号检测下降沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH1_POS_MODE_U*n* 配置通道 1 输入信号检测上升沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH1_HCTRL_MODE_U*n* 控制信号为高电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

PCNT_CH1_LCTRL_MODE_U*n* 控制信号为低电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

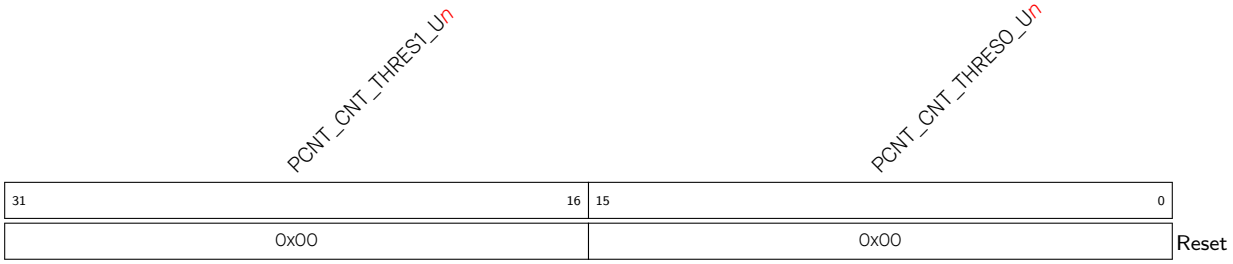
0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

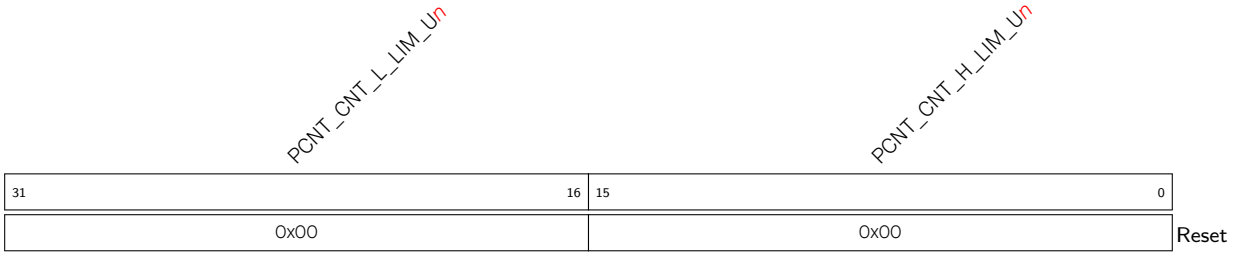
Register 36.2. PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+0x10*n*)



PCNT_CNT_THRES0_U*n* 配置单元 *n* 阈值 0 的值。(R/W)

PCNT_CNT_THRES1_U*n* 配置单元 *n* 阈值 1 的值。(R/W)

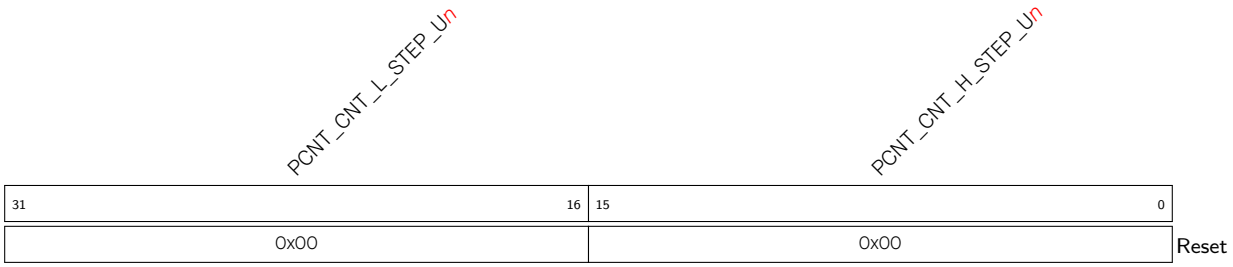
Register 36.3. PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+0x10*n*)



PCNT_CNT_H_LIM_U*n* 配置单元 *n* 的计数上限阈值。当脉冲计数达到上限阈值，计数会被清零。(R/W)

PCNT_CNT_L_LIM_U*n* 配置单元 *n* 的计数下限阈值。当脉冲计数达到下限阈值，计数会被清零。(R/W)

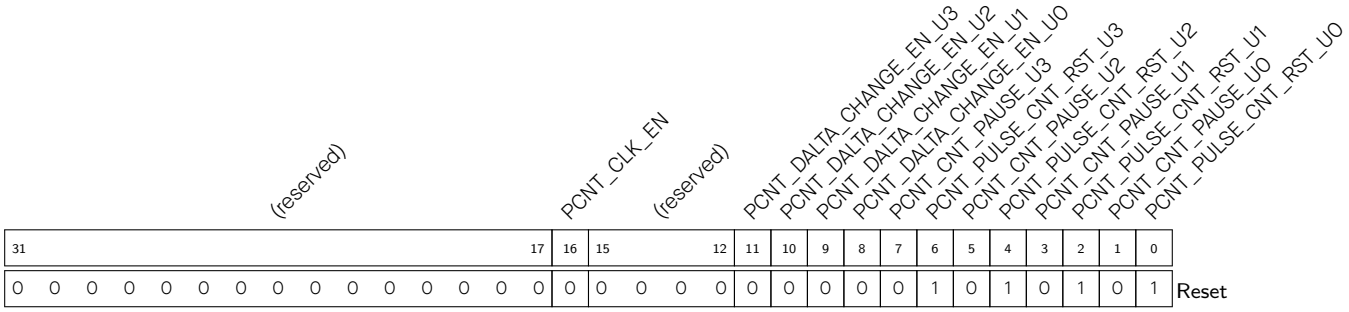
Register 36.4. PCNT_U*n*_CONF3_REG (*n*: 0-3) (0x000C+0x10*n*)



PCNT_CNT_H_STEP_U*n* 配置单元 *n* 的递增计数步长阈值。(R/W)

PCNT_CNT_L_STEP_U*n* 配置单元 *n* 的递减计数步长阈值。(R/W)

Register 36.5. PCNT_CTRL_REG (0x0070)



PCNT_PULSE_CNT_RST_U*n* (*n*: 0-3) 写 1 清零单元 *n* 的计数器。

0：无效

1：清零

(R/W)

PCNT_CNT_PAUSE_U*n* (*n*: 0-3) 写 1 暂停单元 *n* 的计数器。

0：无效

1：暂停

(R/W)

PCNT_DALTA_CHANGE_EN_U*n* (*n*: 0-3) 配置是否使能单元 *n* 的步长计数比较器。

0：无效

1：使能比较器

(R/W)

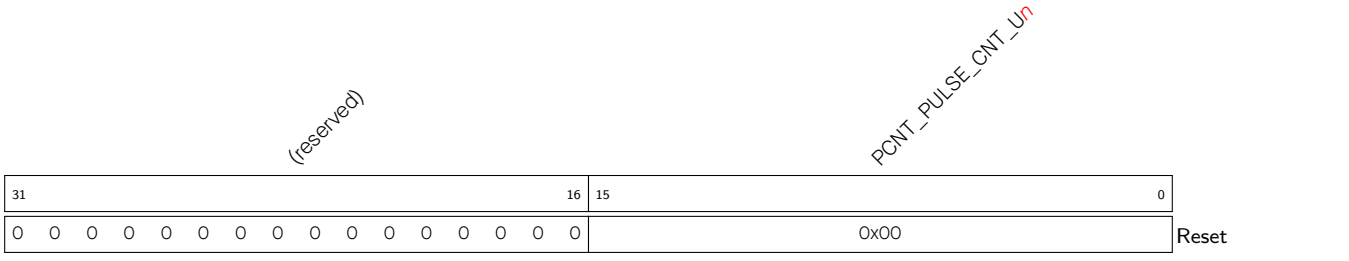
PCNT_CLK_EN 配置是否使能脉冲计数器模块寄存器时钟门控。

0：寄存器时钟仅在有读写操作时开启

1：寄存器时钟一直开启

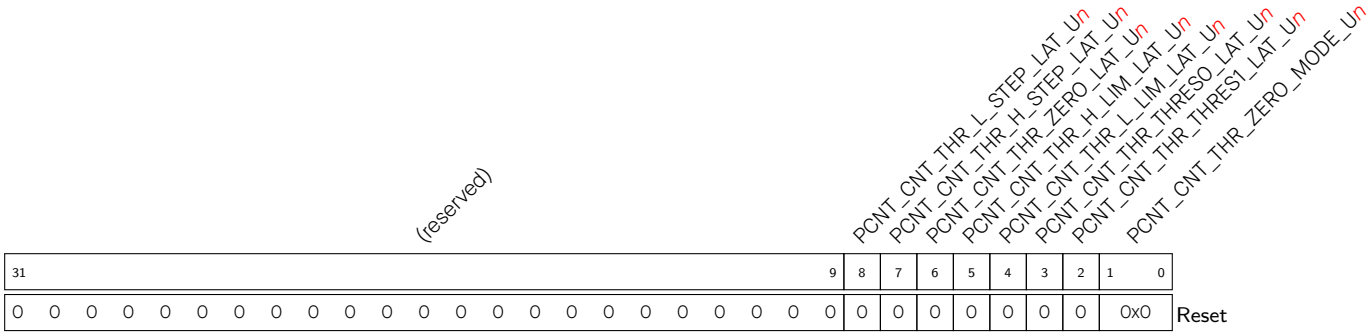
(R/W)

Register 36.6. PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0040+0x4**n*)



PCNT_PULSE_CNT_U*n* 表示单元 *n* 脉冲计数器的当前值。(RO)

Register 36.7. PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0060+0x4**n*)



PCNT_CNT_THR_ZERO_MODE_U*n* 表示 PCNT_U*n* 为 0 时的脉冲计数器状态。

0：脉冲计数器的值由正数减至 0

1：脉冲计数器的值由负数增至 0

2：脉冲计数器为负

3：脉冲计数器为正

(RO)

PCNT_CNT_THR_THRES1_LAT_U*n* 表示阈值中断有效时，PCNT_U*n* 阈值 1 的锁存值。

0：其他

1：脉冲计数器的当前值与阈值 1 相等，阈值 1 有效

(RO)

PCNT_CNT_THR_THRES0_LAT_U*n* 表示阈值中断有效时，PCNT_U*n* 阈值 0 的锁存值。

0：其他

1：脉冲计数器的当前值与阈值 0 相等，阈值 0 有效

(RO)

PCNT_CNT_THR_L_LIM_LAT_U*n* 表示下限中断有效时，PCNT_U*n* 下限的锁存值。

0：其他

1：脉冲计数器的当前值与下限阈值相等，下限有效

(RO)

PCNT_CNT_THR_H_LIM_LAT_U*n* 表示上限中断有效时，PCNT_U*n* 上限的锁存值。

0：其他

1：脉冲计数器的当前值与上限阈值相等，上限有效

(RO)

PCNT_CNT_THR_ZERO_LAT_U*n* 表示阈值中断有效时，PCNT_U*n* 阈值 0 的锁存值。

0：其他

1：脉冲计数器的当前值为 0，阈值 0 有效

(RO)

见下页...

Register 36.7. PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+0x4*n*)

接上页...

PCNT_CNT_THR_H_STEP_LAT_U*n* 表示在步长计数器事件中断有效时，PCNT_U*n* 的步长计数器事件锁存值。

1：当前脉冲计数器的减少量等于 reg_cnt_step，且步长计数器事件有效。

0：其他情况。

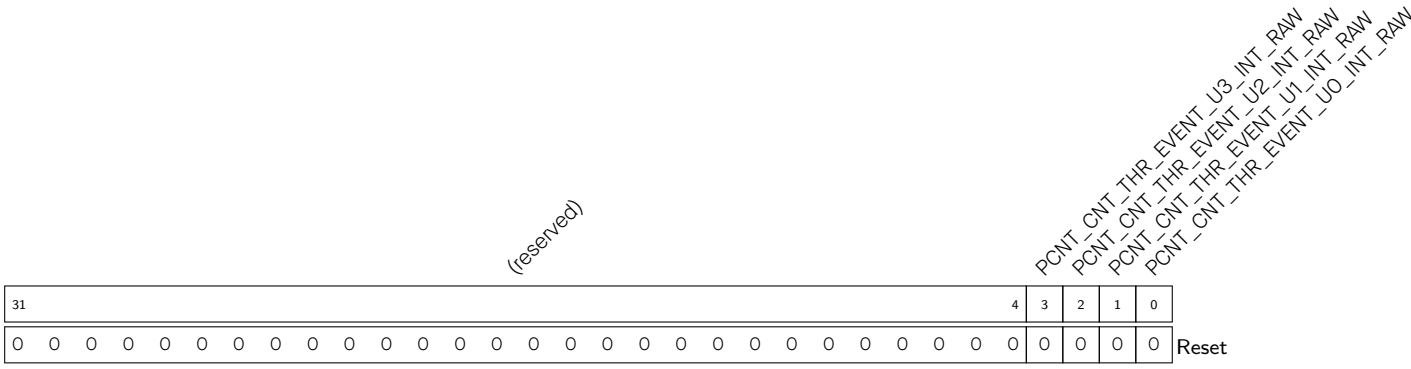
(RO)

PCNT_CNT_THR_L_STEP_LAT_U*n* 表示在步长进计数器事件中断有效时，PCNT_U*n* 的步长计数器事件锁存值。1：当前脉冲计数器的增量等于 reg_cnt_step，且步长计数器事件有效。

0：其他情况。

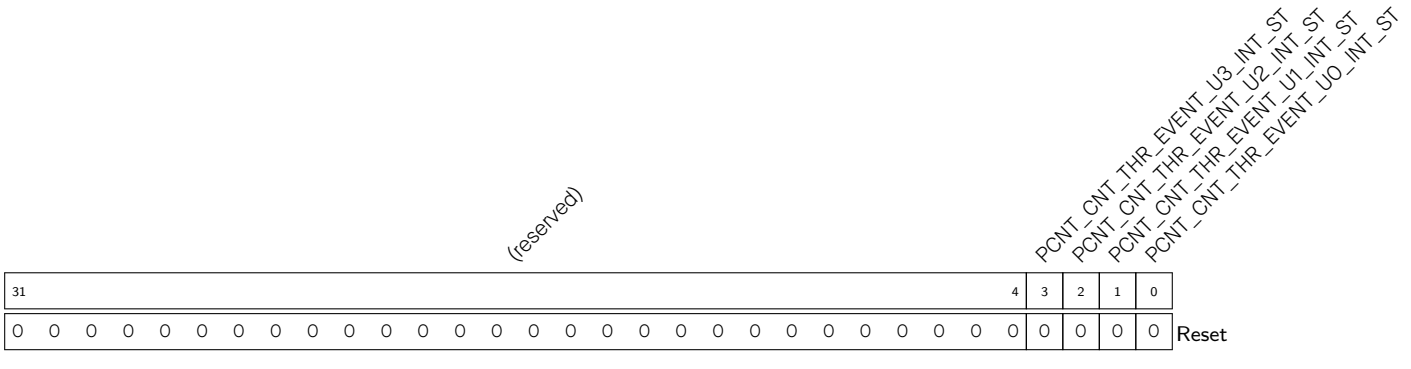
(RO)

Register 36.8. PCNT_INT_RAW_REG (0x0050)



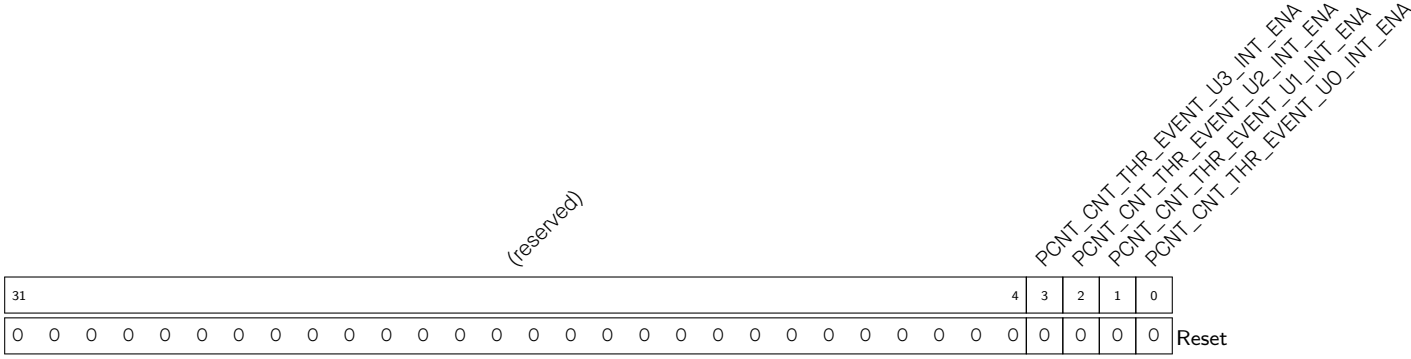
PCNT_CNT_THR_EVENT_U*n*_INT_RAW (*n*: 0-3) 单元 *n* 事件中断的原始中断状态。(R/WTC/SS)

Register 36.9. PCNT_INT_ST_REG (0x0054)



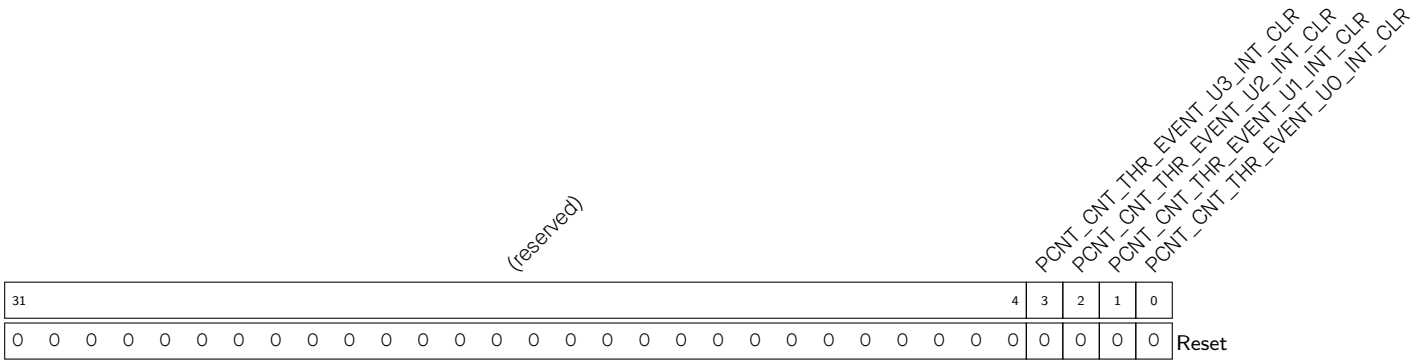
PCNT_CNT_THR_EVENT_U*n*_INT_ST (*n*: 0-3) 单元 *n* 事件中断的屏蔽中断状态。(RO)

Register 36.10. PCNT_INT_ENA_REG (0x0058)



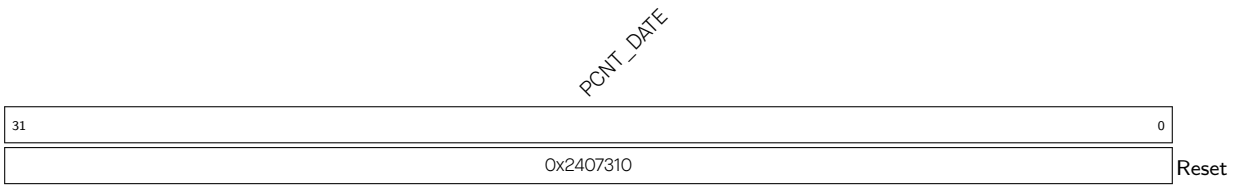
PCNT_CNT_THR_EVENT_U*n*_INT_ENA (*n*: 0-3) 写 1 使能单元 *n* 事件中断。(R/W)

Register 36.11. PCNT_INT_CLR_REG (0x005C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR (*n*: 0-3) 写 1 清除单元 *n* 事件中断。(WT)

Register 36.12. PCNT_DATE_REG (0x00FC)



PCNT_DATE 版本控制寄存器。(R/W)

第 37 章

USB 串口/JTAG 控制器

ESP32-C5 中包含一个 USB 串口/JTAG 控制器，可用于烧录芯片的外部 flash、读取程序输出的数据以及将调试器连接到正在运行的程序中。任何带有 USB 主机（下文将简称为“主机”）的计算机都可以实现上述功能，无需其他外部组件辅助。

37.1 概述

使用串口通信和 JTAG 调试端口通信来编程和调试 ESP32-C5 项目都是可行的，但也有其限制。首先，串口通信和 JTAG 调试端口通信都占用 IO 管脚，减少了软件中可用于控制外部信号的管脚数量。此外，串口通信和 JTAG 调试端口通信都需借助外部芯片或适配器与主机连接，因此，如果需要整合这两个功能，则需额外使用外部芯片或调试适配器。

为解决上述问题，ESP32-C5 提供了一个 USB 串口/JTAG 控制器，同时集成 USB-串口转换器和 USB-JTAG 适配器功能。由于该模块仅使用 USB 2.0 所需的两条数据线直接连接外部 USB 主机，因此 ESP32-C5 仅需占用两个管脚用于调试。

37.2 主要特性

USB 串口/JTAG 控制器具有如下特性：

- 支持 USB 全速标准；集成 CDC-ACM （通信设备类抽象控制模型）和 JTAG 适配器固定功能
- CDC-ACM：
  - 配置虚拟串行功能，在大多数现代操作系统上可实现即插即用
  - 支持主机控制芯片复位和进入下载模式
- JTAG 适配器：
  - 支持使用紧凑的 JTAG 指令实现与 CPU 调试内核的快速通信
- 共配置一个控制端点、一个虚拟中断端点、两个批量输入端点和两个批量输出端点，可实现最大 64 字节的数据载荷
- 集成内部 PHY：基本无需其他外部组件连接主机计算机

说明：

SDIO 从机控制器在单线模式下可以与 USB 串口/JTAG 控制器同时使用，但在四线模式下不能与 USB 串口/JTAG 控制器同时使用。

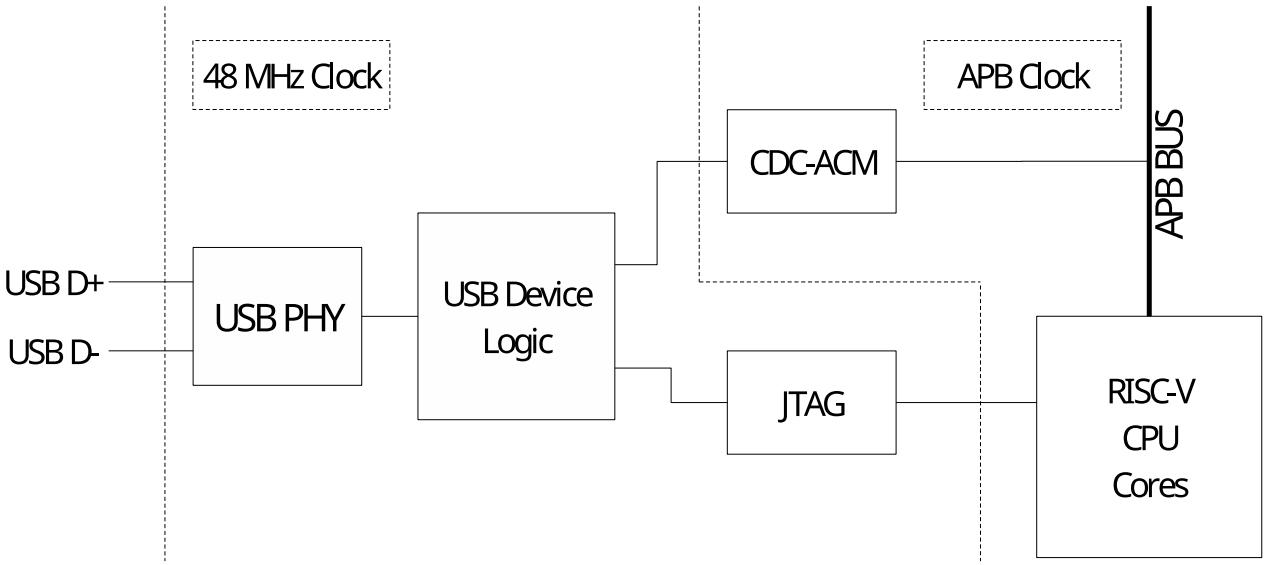


图 37.2-1. USB 串口/JTAG 控制器高层框图

如图 37.2-1 所示，USB 串口/JTAG 控制器包含 USB PHY、USB 设备接口、JTAG 命令处理器、响应捕捉单元以及若干个 CDC-ACM 寄存器。PHY 和设备接口使用主 PLL (BBPLL) 时钟产生的 48 MHz 时钟作为时钟源，CDC-ACM 块中软件可访问的部分则使用 APB_CLK 作为时钟源。JTAG 命令处理器与 ESP32-C5 主处理器中的 JTAG 调试单元相连；CDC-ACM 寄存器则连接至 APB 总线，主 CPU 上运行的软件可对其进行读写访问。

请注意，USB 串口/JTAG 控制器仅支持 USB 2.0 全速标准 (12 Mbps)，不支持 USB 2.0 标准的其他模式（如 480 Mbps 的高速模式）。

图 37.2-2 展示了 USB 串口/JTAG 控制器中 USB 部分的内部详细结构。USB 串口/JTAG 控制器由一个 USB 2.0 全速设备组成，包含一个控制端点、一个虚拟中断端点、两个批量输入端点和两个批量输出端点。这些端点共同组成该复合型 USB 设备，具体可分为 CDC-ACM USB 设备和实现 JTAG 接口功能的供应商特定设备。JTAG 接口直接与芯片的 RISC-V CPU 的调试接口相连，可对运行在该 CPU 上的程序进行调试。同时，CDC-ACM 中包含一组寄存器，CPU 上运行的程序可对其进行读写操作。此外，芯片上的 ROM 启动代码可允许用户通过使用该接口重新烧录 flash。

> **说明：**
> 如需确定某个功能使用的具体端点，请解析设备返回的相关 USB 描述符。这样可以保证与功能相同但实现方式不同的设备兼容。

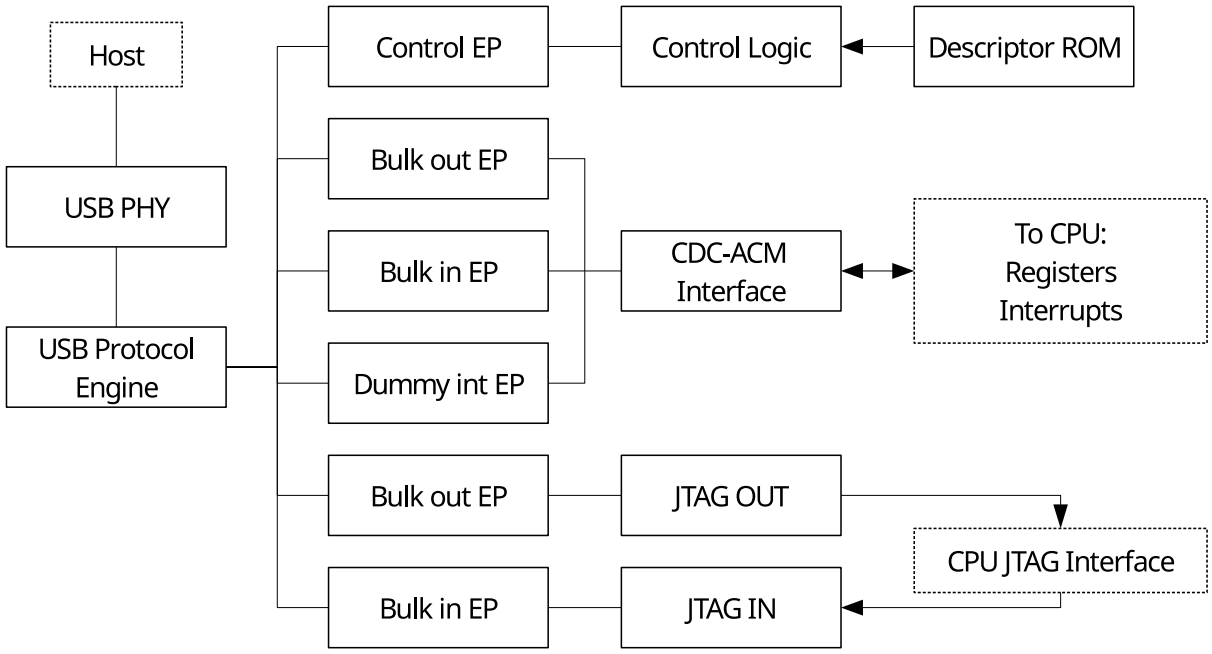


图 37.2-2. USB 串口/JTAG 控制器框图

37.3 功能描述

USB 串口/JTAG 控制器一边与 USB 主机处理器连接，另一边与 CPU 调试硬件以及通过 CDC-ACM 端口通信的软件连接。

37.3.1 CDC-ACM USB 接口描述

CDC-ACM 接口遵循标准 USB CDC-ACM 类别进行虚拟串口通信，包含一个虚拟中断端点（不会发送任何事件，无使用需求）以及一个批量输入端点 (Bulk IN) 和一个批量输出端点 (Bulk OUT) 进行数据接收和发送。这些端点一次可以处理最高 64 字节的数据包，实现高吞吐量。CDC-ACM 为标准的 USB 设备类型，主机一般无需任何特殊安装程序就能正常工作，也就是说，当 USB 调试设备正确连接至主机时，操作系统应能在片刻后显示新的串口信息。

CDC-ACM 接口可以接收以下标准 CDC-ACM 控制请求：

表 37.3-1. 标准 CDC-ACM 控制请求

命令	操作
SEND_BREAK	接收但忽略（虚拟命令）
SET_LINE_CODING	接收，发送值在软件中可读
GET_LINE_CODING	默认返回 9600 baud，无奇偶校验，8 个数据位，1 个停止位（可通过软件更改）
SET_CONTROL_LINE_STATE	设置 RTC/DTR 线的状态，如表 37.3-2 所示

除了通用的通信之外，CDC-ACM 接口还可以复位 ESP32-C5 并选择使其进入下载模式，从而烧录新的固件。这一功能可通过设置虚拟串口的 RTS 和 DTR 线来实现。

表 37.3-2. CDC-ACM 中 RTS 和 DTR 的设置

RTS	DTR	操作
0	0	清除下载模式标志
0	1	置位下载模式标志
1	0	复位 ESP32-C5
1	1	无操作

请注意，当 ESP32-C5 复位时，如果下载模式标志已置位，则 ESP32-C5 重启时将直接进入下载模式；如果下载模式标志已清除，则 ESP32-C5 将从 flash 启动。具体操作流程，请参见章节 37.5。除此之外，也可以通过烧写相应 eFuse 来禁用上述功能，详细信息请参见章节 7 *eFuse 控制器 (EFUSE)*。

37.3.2 CDC-ACM 固件接口描述

由于 USB 串口/JTAG 控制器与 ESP32-C5 的内部 APB 总线相连，因此 CPU 可直接与该模块交互，主要用于对连接主机上的虚拟串口进行读写操作。

CPU 向主机发送并从主机接收 0 ~ 64 字节大小的 USB CDC-ACM 串口数据包。主机已积累足够多的 CDC-ACM 数据时，将向 CDC-ACM 的接收端点发送一个数据包。如果 USB 串行/JTAG 控制器中有空闲缓冲区，该缓冲区将接收这一数据包。反之，主机会定期检查 USB 串口/JTAG 控制器内是否有待向主机发送的数据包，如果有，主机将接收这个数据包。

固件可通过以下两种方式之一获知是否有来自主机的新数据：第一，只要缓冲区中还有来自主机的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位将保持为 1；第二，如果有新的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断将被触发。当新数据可用时，固件可通过重复从 USB_SERIAL_JTAG_EP1_REG 读取字节来获取该数据。读取每个字节后，可通过检查 USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位来看是否还有其他可读取的数据，从而确定需要读取的总字节数。读取完所有数据后，USB 设备会自动做好准备，以接收来自主机的新数据包。

一般来说，发送数据后，连接的主机会阻塞，直到固件通过上述方法之一读取数据。然而，也可能存在某些固件根本不读取数据的情况。通常来说，用于与独立 USB 串口转换器通信的主机程序不会出现这种情况，例如，程序会停止执行直到数据被读取。ESP32-C5 的 USB 串口/JTAG 转换器可以模拟独立 USB 串口/JTAG 转换器的行为：通过向 USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX 写入超时时间并向 USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN 写入 1，任何未读取的数据将在设定的超时时间后自动刷新。

当固件需要发送数据时，可将待发送数据置于发送缓冲区并触发刷写，从而使主机以 USB 数据包的形式接收该数据。在此之前，需确保发送缓冲区有可用空间以存储待发送数据。固件可通过读取 USB_REG_SERIAL_IN_EP_DATA_FREE 位检查发送缓冲区是否有可用空间；当该值为 1 时，发送缓冲区有可用空间。此时，固件可通过向 USB_SERIAL_JTAG_EP1_REG 寄存器中写入字节，从而向缓冲区中写入数据。但是，数据写入后并不会立即触发向主机发送数据，需首先对缓冲区执行刷写操作。刷写操作后，整个缓冲区将准备好被 USB 主机立即接收。可通过两种方式触发刷写：1) 将第 64 个字节写入缓冲区后，USB 硬件会自动将缓冲区刷写到主机；2) 固件可通过向 USB_REG_SERIAL_WR_DONE 写入 1 来触发刷写。

不论以何种方式触发刷写操作，在此期间固件都无法向缓冲区写入数据，直到缓冲区中的所有数据都已被主机读取完成。主机读取完成后，将触发 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断，此时可向缓冲区内写入新的 64 字节数据。

在软件中可以处理一些带外串行请求，例如主机设置 DTR 和 RTS，以及改变线路状态。如果 CDC-ACM 接口收到 SET_LINE_CODING 请求，可将外设配置为触发 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断，此时可以从 USB_SERIAL_JTAG_SET_LINE_CODE_WO_REG 寄存器读取线路编码。类似的，如果

SET_CONTROL_LINE_STATE 请求导致线路状态改变，将触发 USB_SERIAL_JTAG_RTS_CHG_INT 和 USB_SERIAL_JTAG_DTR_CHG_INT 中断。此后，软件可以通过 USB_SERIAL_JTAG_RTS 和 USB_SERIAL_JTAG_DTR 位来读取具体状态。注意，如前所述，某些 RTS/DTR 序列会导致 ESP32-C5 硬件复位。软件可以通过置位 USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS 位来禁止硬件识别这些 DTR/RTS 序列，从而允许软件自由解读信号。

最后，主机可以通过 GET_LINE_CODING 读取当前线路状态。此事件将回送 USB_SERIAL_JTAG_GET_LINE_CODE_WO_REG 寄存器中的数据，并触发 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。

37.3.3 USB-JTAG 接口：JTAG 命令处理器

USB-JTAG 接口使用一种供应商特定接口类型实现命令解析的功能。它由两个端点组成：一个用于接收命令，一个用于发送响应。此外，一些对时效性要求不高的命令也可以作为控制请求发出。

JTAG 命令处理器负责解析从主机到 JTAG 接口的命令。JTAG 命令处理器内部包含一个全四线 JTAG 总线，包括发送信号到 RISC-V CPU 的 TCK、TMS 和 TDI 输出线，以及从 CPU 返回信号至 JTAG 响应捕捉单元的 TDO 线。这些信号都符合 IEEE 1149.1 JTAG 标准。此外，还有一条 SRST 线用于复位 ESP32-C5。

另外，软件也可以置位 USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN，将这些信号通过 GPIO 交换矩阵连接至 ESP32-C5 的 IO 焊盘上，此操作也将允许通过 USB 串口/JTAG 控制器对外部设备进行调试。

JTAG 命令处理器会将每个接收到的半字节（4 位）解析为一条命令。由于 USB 以 8 位为一个字节接收数据，这就意味着每个字节中都包含两条命令。USB 命令处理器将先解析高 4 位字节，然后再解析低 4 位字节。这些命令用于控制内部 JTAG 总线的 TCK、TMS、TDI、SRST 线，以及向 JTAG 响应捕捉单元发出信号，说明需要捕捉 TDO 线（由 CPU 调试逻辑驱动）的状态。

JTAG 总线中，TCK、TMS、TDI 和 TDO 线直接与 RISC-V CPU 的 JTAG 调试逻辑相连。上文提到的 SRST 线则与 ESP32-C5 数字电路中的复位逻辑相连，该线电平拉高，芯片将进行芯片复位。请注意，SRST 线并不会对 USB 串口/JTAG 控制器模块产生影响。

1 个半字节中可包含以下命令：

表 37.3-3. 半字节中的命令

位	3	2	1	0
CMD_CLK	0	cap	tms	tdi
CMD_RST	1	0	0	srst
CMD_FLUSH	1	0	1	0
CMD_RSV	1	0	1	1
CMD_REP	1	1	R1	R0

- CMD_CLK：将 TDI 和 TMS 设置为指示值，并在 TCK 上发出一个时钟脉冲。如果 CAP 位为 1，将指示 JTAG 响应捕捉单元捕捉 TDO 线的状态。该指令构成了 JTAG 通信的基础。
- CMD_RST：将 SRST 线的状态设置为指示值。该命令可用于复位 ESP32-C5。
- CMD_FLUSH：指示 JTAG 响应捕捉单元对接收到的所有位的缓冲区进行刷写操作，以便主机读取这些位。请注意在某些情况下，一次 JTAG 通信会结束于第奇数个命令，即结束于第奇数个半字节。此时，可重复执行该命令直到获得偶数个半字节，使其组成整数个字节。
- CMD_RSV：该版本中保留。ESP32-C5 在接收到该命令时会自动忽略。

- CMD_REP：将上一条指令（非 CMD_REP）重复一定次数，以压缩多次重复 CMD_CLK 的命令流。因此，CMD_CLK 命令后可能跟随着多个 CMD_REP。一次 CMD_REP 命令产生的重复次数可表示为 $repetition\_count = (R1 \times 2 + R0) \times (4^{cmd\_rep\_count})$，其中 cmd_rep_count 表示该命令之前的 CMD_REP 数量。请注意，CMD_REP 仅用于重复 CMD_CLK 命令。也就是说，如果在 CMD_FLUSH 后使用该命令，USB 设备将无法响应，需进行 USB 复位后才可恢复正常。

37.3.4 USB-JTAG 接口：CMD_REP 使用示例

下列命令用于演示如何使用 CMD_REP 命令。请注意，该示例中每个命令为半字节，命令流的每个字节为 0x0D 0x5E 0xCF。

1. 0x0 (CMD_CLK: cap=0, tdi=0, tms=0)
2. 0xD (CMD_REP: R1=0, R0=1)
3. 0x5 (CMD_CLK: cap=1, tdi=0, tms=1)
4. 0xE (CMD_REP: R1=1, R0=0)
5. 0xC (CMD_REP: R1=0, R0=0)
6. 0xF (CMD_REP: R1=1, R0=1)

每一步骤的具体操作为：

1. TCK 上发出一个时钟脉冲，TDI 和 TMS 设置为 0。未捕捉到任何数据。
2. TCK 上再次发出 $(0 \times 2 + 1) \times (4^0) = 1$ 个时钟脉冲，其他设置与步骤 1 相同。
3. TCK 上发出一个时钟脉冲，TDI 设置为 0，TMS 设置为 1。捕捉到 TDO 线上的数据。
4. TCK 上再次发出 $(1 \times 2 + 0) \times (4^0) = 2$ 个时钟脉冲，其他设置与步骤 3 相同。
5. 未发生任何动作：$(0 \times 2 + 0) \times (4^1) = 0$。请注意，该操作将增加下一步骤中的 cmd_rep_count 数值。
6. TCK 上再次发出 $(1 \times 2 + 1) \times (4^2) = 48$ 个时钟脉冲，其他设置与步骤 3 相同。

换言之，该命令流示例的操作结果等同于执行两次命令 1，然后执行 51 次命令 3 的效果。

37.3.5 USB-JTAG 接口：响应捕捉单元

响应捕捉单元首先读取内部 JTAG 总线的 TDO 线，并在命令处理器执行 CMD_CLK (cap=1) 命令时捕捉 TDO 线的值，然后将这个值放入内部移位寄存器中，且在接收到 8 位时向 USB 缓冲区写入 1 个字节。这 8 位中的最低有效位即为最先从 TDO 线读取的值。

一旦接收到 64 字节 (512 位) 数据或执行 CMD_FLUSH 命令后，响应捕捉单元将使缓冲区可被主机接收。请注意，USB 逻辑的接口为双缓冲。因此，只要 USB 的吞吐量充足，响应捕捉单元就可以随时接收更多数据，即当一个缓冲区等待发送给主机时，另一个缓冲区可以继续接收数据。当主机从缓冲区成功接收数据且响应捕捉单元对缓冲区执行刷写操作后，这两个缓冲区便可以交换位置。

同时，这也意味着一个命令流可导致最多 128 字节（若该命令流中有刷写命令，则该数字会减小）的捕捉数据生成，而无需主机主动接收这些数据。如果还是生成了超过该阈值数量的捕捉数据，则命令流将暂停，且设备在这些数据被读取之前不会接收其他命令。

另需注意，一般情况下，响应捕捉单元会尽量不发送 0 字节响应。例如，当发送一系列 CMD_FLUSH 命令后，并不会产生一系列 0 字节 USB 响应。但在当前版本中，一些特殊情况下也可能产生 0 字节响应。建议直接忽略这些响应信息。

37.3.6 USB-JTAG 接口：控制传输请求

除命令处理器和响应捕捉单元之外，USB-JTAG 接口也可接收一些控制请求，具体如下表所示：

表 37.3-4. USB-JTAG 控制请求

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000000b	0 (VEND_JTAG_SETDIV)	[divider]	接口	0	None
01000000b	1 (VEND_JTAG_SETIO)	[iobits]	接口	0	None
11000000b	2 (VEND_JTAG_GETTDO)	0	接口	1	[iostate]
10000000b	6 (GET_DESCRIPTOR)	0x2000	0	256	[jtag cap desc]

- VEND_JTAG_SETDIV：设置使用的分频器。该命令直接影响 TCK 时钟脉冲的持续时间。TCK 时钟脉冲来自于由内部分频器向下分频得到的 48 MHz 基准时钟。该控制请求允许主机设置此内部分频器。请注意，该分频器在启动时的初始值为 2，即 TCK 时钟速率一般为 24 MHz。
- VEND_JTAG_SETIO：跳过 JTAG 命令处理器，直接将内部 TDI、TDO、TMS 和 SRST 线设置为指定值。这些指定值在 wValue 字段中以 11'b0, srst, trst, tck, tms, tdi 的格式编码。
- VEND_JTAG_GETTDO：跳过 JTAG 响应捕捉单元，直接读取内部 TDO 信号。该请求返回 1 个字节，其中的最低有效位代表 TDO 线的状态。
- GET_DESCRIPTOR：为标准 USB 请求，该请求也可使用供应商专用的 0x2000 wValue 获取 JTAG 功能描述符。该请求返回一定字节，具体字节数所代表的 USB-JTAG 适配器功能如表 37.3-5 所示。这一固定结构允许主机软件自动支持未来的硬件新版本，无需再次更新。

ESP32-C5 包含的 JTAG 功能描述符如下表所示。请注意，所有 16 位值都为小端序存储。

表 37.3-5. JTAG 功能描述符

字节	数值	描述
0	1	JTAG 协议功能结构的版本
1	10	JTAG 协议功能长度
2	1	结构类型：1 代表高速功能结构类型
3	8	该高速功能结构长度
4 ~ 5	4800	以 10 kHz 为增量的 JTAG 基准时钟速度，其最大速度为该值的一半
6 ~ 7	1	最小分频系数，可通过 VEND_JTAG_SETDIV 设置
8 ~ 9	255	最大分频系数，可通过 VEND_JTAG_SETDIV 设置

37.4 中断

ESP32-C5 的 USB 串口/JTAG 控制器的下列内部中断源可生成中断信号 USB_SERIAL_JTAG_INTR，并发送给中断矩阵。

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT：JTAG 批量输出端点接收到刷写命令时触发。
- USB_SERIAL_JTAG_SOF_INT：接收到 SOF 帧时触发。
- USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT：串口输出端点接收到 1 个数据包时触发。
- USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT：串口输入端点为空时触发。

- USB_SERIAL_JTAG_PID_ERR_INT：检测到 PID 错误时触发。
- USB_SERIAL_JTAG_CRC5_ERR_INT：检测到 CRC5 错误时触发。
- USB_SERIAL_JTAG_CRC16_ERR_INT：检测到 CRC16 错误时触发。
- USB_SERIAL_JTAG_STUFF_ERR_INT：检测到位填充错误时触发。
- USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT：输入端点 1 接收到一个 IN 令牌时触发。
- USB_SERIAL_JTAG_USB_BUS_RESET_INT：检测到 USB 总线复位时触发。
- USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT：输出端点 1 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT：输出端点 2 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_RTS_CHG_INT：USB 串行通道的 RTS 电平改变时触发。
- USB_SERIAL_JTAG_DTR_CHG_INT：USB 串行通道的 DTR 电平改变时触发。
- USB_SERIAL_JTAG_GET_LINE_CODE_INT：收到 GET_LINE_CODING 请求的电平时触发。
- USB_SERIAL_JTAG_SET_LINE_CODE_INT：收到 SET_LINE_CODING 请求的电平时触发。

37.5 配置流程

使用 USB 串口/JTAG 控制器之前，几乎不需要多余的配置。除了主机操作系统已经完成的标准 USB 初始化之外，USB-JTAG 硬件本身不需要进行任何配置。此外，CDC-ACM 虚拟串口在主机端也支持即插即用。

固件方面也几乎不需要初始化。USB 硬件是自初始化的，在其启动后，如果固件连接了一台主机并在 CDC-ACM 接口上监听，除非固件设置了中断处理程序，否则无需任何特定设置就可以实现前文所述的数据交换。

需要注意的是，可能会出现主机未连接或 CDC-ACM 虚拟串口未打开的情况。在这种情况下，发送至主机的数据包永远无法被接收，发送缓冲区也永远不会为空。因而，必须对这种情况进行检测并执行超时操作，以便清楚地检测主机侧的端口是否关闭。

其次，需要注意 USB 设备依赖于产生 48 MHz USB PHY 时钟的 BBPLL。如果此 PLL 被禁用，USB 通信将停止。

上述情况可能发生在 Deep-sleep 期间。在 Deep-sleep 模式下，USB 串口/JTAG 控制器（及其连接的 RISC-V CPU）将完全断电。如果有设备需要在这两种模式下进行调试，最好使用外部 JTAG 调试器和串行接口。

CDC-ACM 接口还可用于复位芯片，使其进入或退出下载模式。产生正确的握手信号序列则有些复杂，因为大多数操作系统仅支持分别设置或重置 DTR 和 RTS，无法同时进行。此外，一些驱动程序（如 Windows 系统上的标准 CDC-ACM 驱动程序）须先设置 RTS 后才可设置 DTR，此时用户必须先明确设置 RTS，才能传播 DTR 的值。推荐遵循以下步骤进行设置。

复位芯片使其进入下载模式：

表 37.5-1. 复位芯片进入下载模式

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	初始化以获取数值
清除 RTS	RTS=0, DTR=0	-
设置 DTR	RTS=0, DTR=1	设置下载模式标志
清除 RTS	RTS=0, DTR=1	传播 DTR
设置 RTS	RTS=1, DTR=1	-
清除 DTR	RTS=1, DTR=0	复位芯片
设置 RTS	RTS=1, DTR=0	传播 DTR
清除 RTS	RTS=0, DTR=0	清除下载标志

复位 SoC 使其从 flash 启动：

表 37.5-2. 复位 SoC 从 flash 启动

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	-
清除 RTS	RTS=0, DTR=0	清除下载标志
设置 RTS	RTS=1, DTR=0	复位 SoC
清除 RTS	RTS=0, DTR=0	退出复位

37.6 寄存器列表

本小节的所有地址均为相对于 USB 串口/JTAG 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
USB_SERIAL_JTAG_EP1_REG	CDC-ACM 输入/输出端点 FIFO 访问配置寄存器	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	CDC-ACM FIFO 配置寄存器	0x0004	varies
USB_SERIAL_JTAG_CONF0_REG	PHY 硬件配置寄存器	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	PHY 调试寄存器	0x001C	varies
USB_SERIAL_JTAG_MISC_CONF_REG	时钟使能控制寄存器	0x0044	R/W
USB_SERIAL_JTAG_MEM_CONF_REG	存储器配置寄存器	0x0048	R/W
USB_SERIAL_JTAG_CHIP_RST_REG	CDC-ACM 芯片重置寄存器	0x004C	varies
USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG	GET_LINE_CODING 命令的 W0	0x0058	R/W
USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG	GET_LINE_CODING 命令的 W1	0x005C	R/W
USB_SERIAL_JTAG_CONFIG_UPDATE_REG	配置寄存器更新寄存器	0x0060	WT
USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG	串口 AFIFO 配置寄存器	0x0064	varies
USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG	USB UART 输出端点超时配置	0x006C	varies
USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG	USB UART 输出端点超时配置	0x0070	R/W
中断寄存器			
USB_SERIAL_JTAG_INT_RAW_REG	中断原始状态寄存器	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	中断状态寄存器	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	中断使能状态寄存器	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	中断清除状态寄存器	0x0014	WT
状态寄存器			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO 状态与控制寄存器	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	接收 SOF 帧索引寄存器	0x0024	RO
USB_SERIAL_JTAG_IN_EP0_ST_REG	输入端点状态信息寄存器	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM 输入端点状态信息寄存器	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM 中断输入端点状态信息寄存器	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG 输入端点状态信息寄存器	0x0034	RO
USB_SERIAL_JTAG_OUT_EP0_ST_REG	输出端点状态信息寄存器	0x0038	RO
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM 输出端点状态信息寄存器	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG 输出端点状态信息寄存器	0x0040	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	SET_LINE_CODING 命令的 W0	0x0050	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG	SET_LINE_CODING 命令的 W1	0x0054	RO
USB_SERIAL_JTAG_BUS_RESET_ST_REG	USB 总线重置状态寄存器	0x0068	RO
版本寄存器			

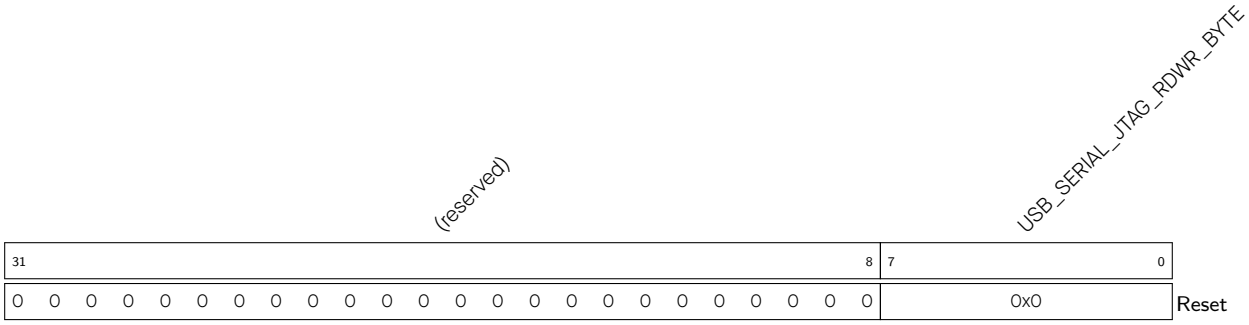
名称	描述	地址	访问
USB_SERIAL_JTAG_DATE_REG	版本寄存器	0x0080	R/W

37.7 寄存器

本小节的所有地址均为相对于 USB 串口/JTAG 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 37.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



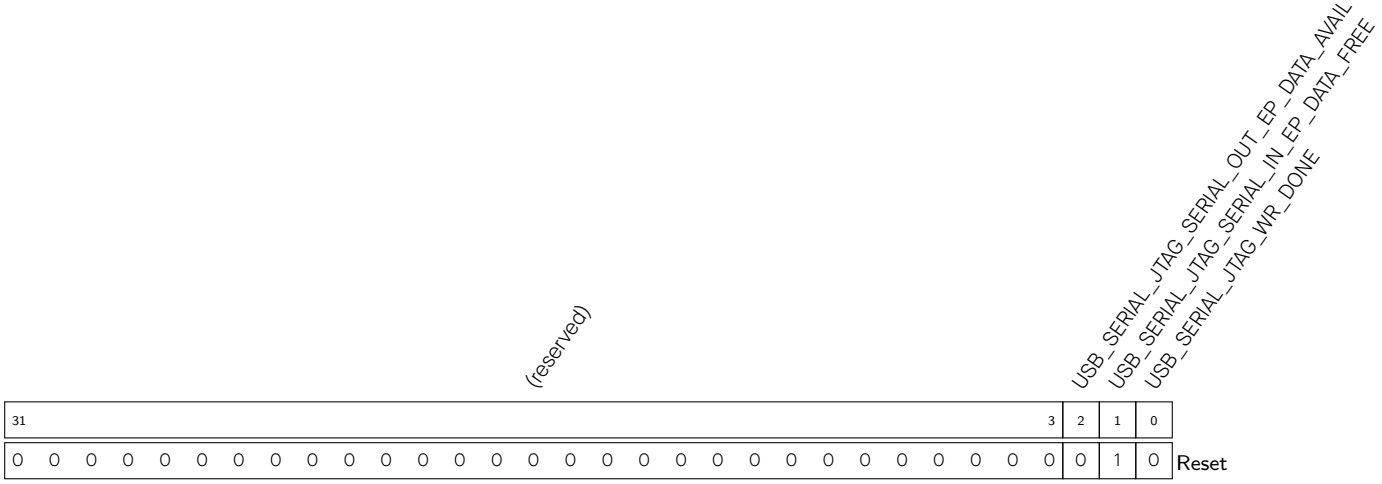
USB_SERIAL_JTAG_RDWR_BYTE 写入该位即为将数据写入 CDC-ACM TX FIFO；读取该位即为从 CDC-ACM RX FIFO 读取一个字节并返回该字节。

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 置位时，用户可向 CDC-ACM TX FIFO 中写入数据（最大 64 字节）。

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 置位时，用户可通过查看 USB_SERIAL_JTAG_OUT_EP1_WR_ADDR 的值获知接收到的数据量，然后从 CDC-ACM RX FIFO 中读取这些数据。

(R/W)

Register 37.2. USB_SERIAL_JTAG_EP1_CONF_REG (0x0004)



USB_SERIAL_JTAG_WR_DONE 配置是否表明已完成向 CDC-ACM TX FIFO 写入字节数据。

0: 无效

1: 表明已完成向 CDC-ACM TX FIFO 写入字节数据

该位将保持为 0，直到 USB 主机读取 CDC-ACM TX FIFO 中的数据。

(WT)

USB_SERIAL_JTAG_SERIAL_IN_EP_DATA_FREE 表示 CDC-ACM TX FIFO 是否可以写入数据。

0：CDC-ACM TX FIFO 已写满，不可写入数据

1：CDC-ACM TX FIFO 未写满且可以写入数据

写入 USB_SERIAL_JTAG_WR_DONE 后，该位将置为 0，直到其中的数据已发送至 USB 主机。

(RO)

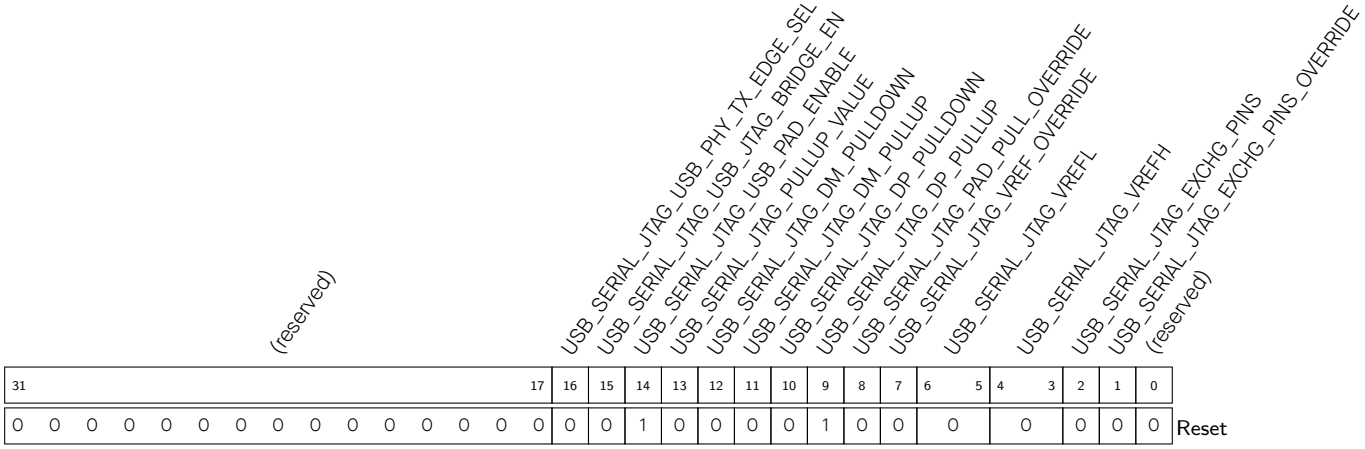
USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 表示 CDC-ACM RX FIFO 中是否有数据。

0: CDC-ACM RX FIFO 中没有数据

1: CDC-ACM RX FIFO 中有数据

(RO)

Register 37.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)



USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚交换。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS 配置是否交换 USB D+ 和 D- 管脚。

0：无效

1：交换

(R/W)

USB_SERIAL_JTAG_VREFH 配置单端输入高阈值。

0：1.76 V

1：1.84 V

2：1.92 V

3：2.00 V

(R/W)

USB_SERIAL_JTAG_VREFL 配置单端输入低阈值。

0：0.80 V

1：0.88 V

2：0.96 V

3：1.04 V

(R/W)

USB_SERIAL_JTAG_VREF_OVERRIDE 配置是否使能软件控制输入阈值。

0：不使能

1：使能

(R/W)

见下页...

Register 37.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

接上页...

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚的上下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DP_PULLUP 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D+ 管脚的上拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D+ 管脚的下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DM_PULLUP 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D- 管脚的上拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DM_PULLDOWN 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D- 管脚的下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时的上拉数值。

0：2.2 K

1：1.1 K

(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE 配置是否使能 USB 管脚。

0：不使能

1：使能

(R/W)

见下页...

Register 37.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

接上页...

USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN 配置是否断开 USB_JTAG 和内部 JTAG 的连接。

0：USB_JTAG 连接到 CPU 的内部 JTAG 端口

1：USB_JTAG 和内部 JTAG 之间断开连接，MTMS、MTDI 和 MTCK 为通过 GPIO 交换矩阵的输出，MTDO 为通过 GPIO 交换矩阵的输入

(R/W)

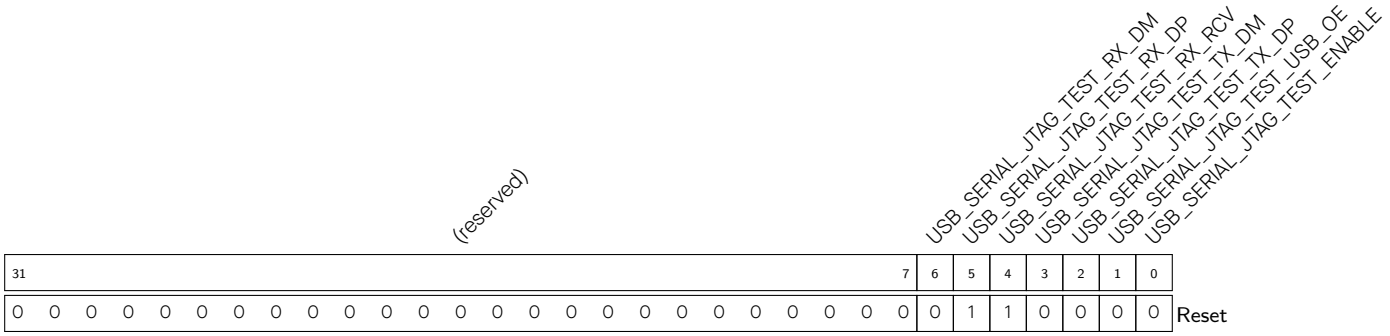
USB_SERIAL_JTAG_USB_PHY_TX_EDGE_SEL 配置将 DP 和 DM 信号发送到 USB PHY 的时钟沿。

0：在时钟下降沿发送

1：在时钟上升沿发送

(R/W)

Register 37.4. USB_SERIAL_JTAG_TEST_REG (0x001C)



USB_SERIAL_JTAG_TEST_ENABLE 配置是否使能 USB 管脚测试模式。

0：不使能，恢复正常模式

1：使能 USB 管脚测试模式

使能 USB 管脚测试模式将允许使用该寄存器的其他位来控制/读取 USB 管脚。

(R/W)

USB_SERIAL_JTAG_TEST_USB_OE 配置是否使能 USB 管脚输出。

0：将 D+ 和 D- 设为高阻抗

1：在 D+ 和 D- 管脚上输出 USB_SERIAL_JTAG_TEST_TX_DP 和 USB_SERIAL_JTAG_TEST_TX_DM 中设置的值

(R/W)

USB_SERIAL_JTAG_TEST_TX_DP 当 USB_SERIAL_JTAG_TEST_USB_OE 为 1 时，设置测试模式下 USB D+ 管脚的值。(R/W)

USB_SERIAL_JTAG_TEST_TX_DM 当 USB_SERIAL_JTAG_TEST_USB_OE 为 1 时，设置测试模式下 USB D- 管脚的值。(R/W)

USB_SERIAL_JTAG_TEST_RX_RCV 表示在测试模式下 USB D- 和 USB D+ 管脚之间电压差的当前逻辑水平。

0：USB D- 电压高于 USB D+

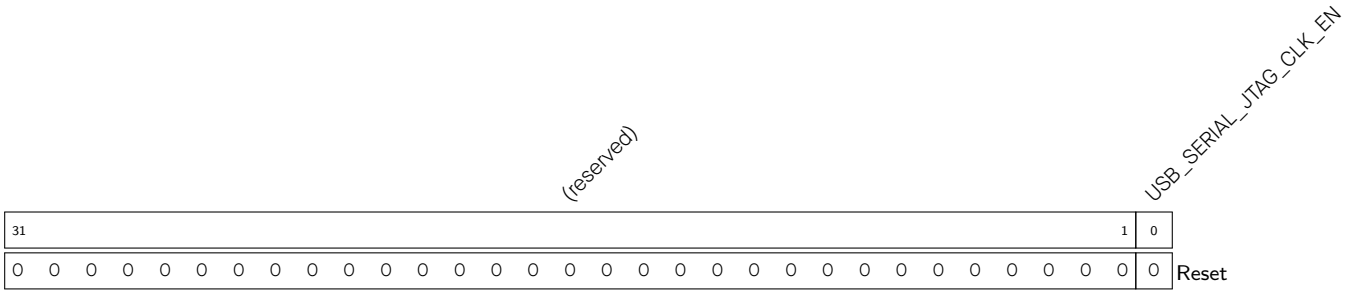
1：USB D- 电压低于 USB D+

(RO)

USB_SERIAL_JTAG_TEST_RX_DP 表示在测试模式下 USB D+ 管脚的逻辑电平。(RO)

USB_SERIAL_JTAG_TEST_RX_DM 表示在测试模式下 USB D- 管脚的逻辑电平。(RO)

Register 37.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



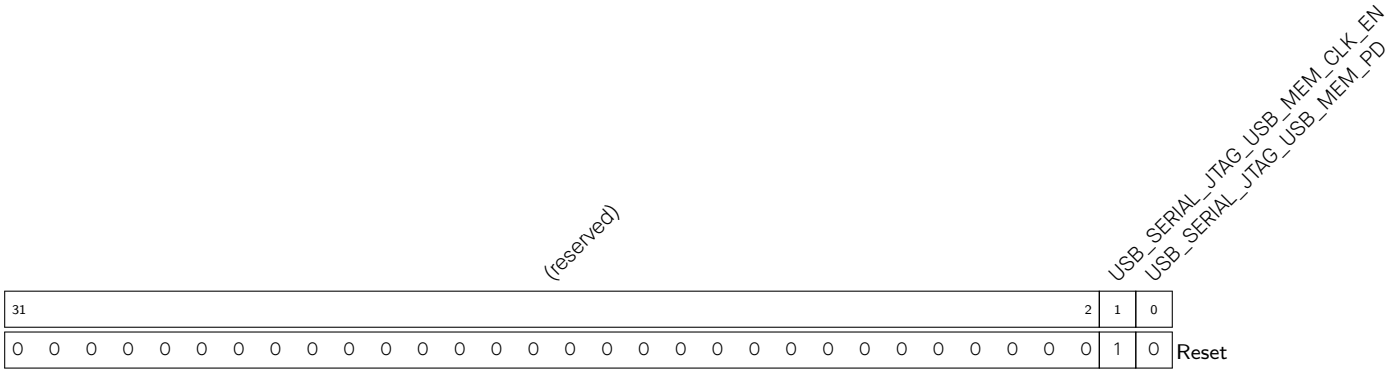
USB_SERIAL_JTAG_CLK_EN 配置是否强制打开寄存器的时钟。

0：支持仅当应用程序向寄存器写入数据时打开时钟

1：强制打开寄存器的时钟

(R/W)

Register 37.6. USB_SERIAL_JTAG_MEM_CONF_REG (0x0048)



USB_SERIAL_JTAG_USB_MEM_PD 配置是否关闭 USB 存储器。

0：无效

1：关闭

(R/W)

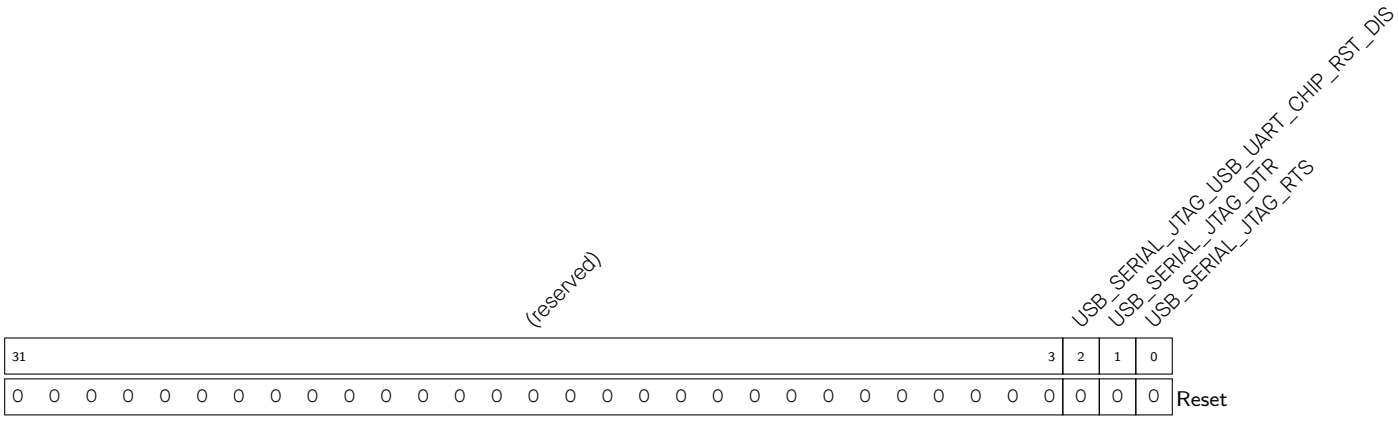
USB_SERIAL_JTAG_USB_MEM_CLK_EN 配置是否强制对 USB 存储器进行时钟分频。

0：无效

1：强制

(R/W)

Register 37.7. USB_SERIAL_JTAG_CHIP_RST_REG (0x004C)



USB_SERIAL_JTAG_RTS 表示最近的 SET_LINE_CODING 命令所设定的 RTS 信号的状态。(RO)

USB_SERIAL_JTAG_DTR 表示最近的 SET_LINE_CODING 命令所设定的 DTR 信号的状态。(RO)

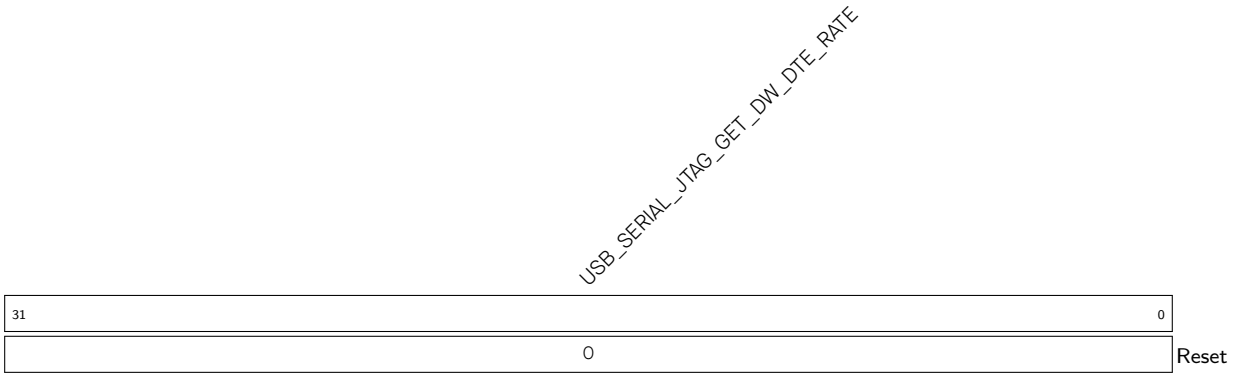
USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS 配置是否禁用来自 USB 串行通道的芯片复位。

0: 无效

1: 禁用

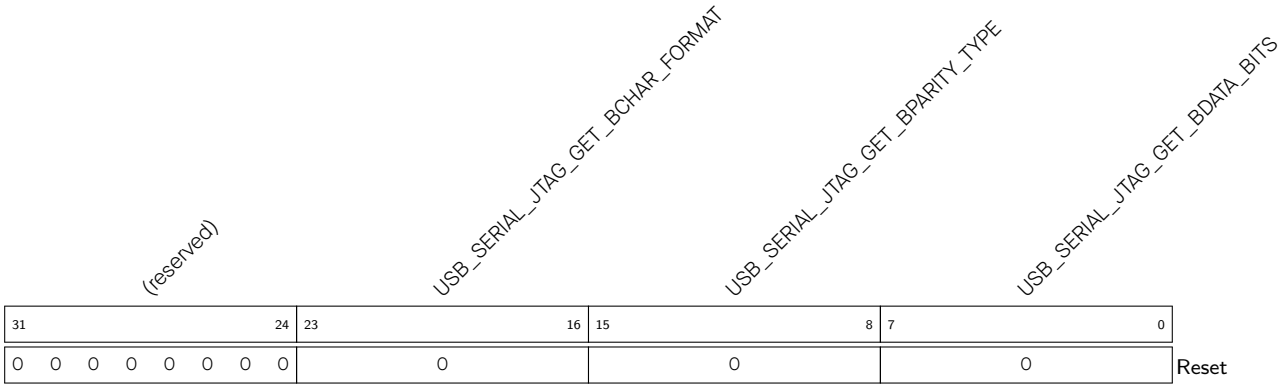
(R/W)

Register 37.8. USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG (0x0058)



USB_SERIAL_JTAG_GET_DW_DTE_RATE 配置软件设置的 dwDTERate 值，该值由 GET_LINE_CODING 命令请求。(R/W)

Register 37.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

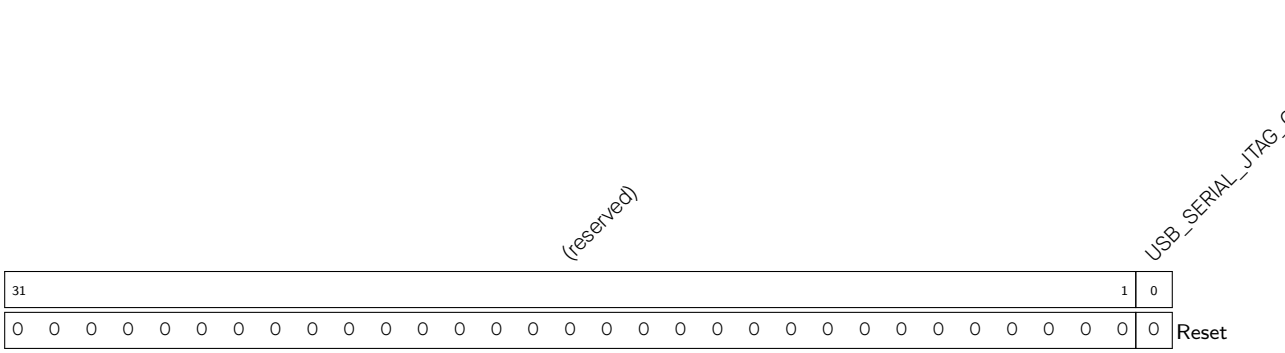


USB_SERIAL_JTAG_GET_BDATA_BITS 配置软件设置的 bDataBits 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BPARITY_TYPE 配置软件设置的 bParityType 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BCHAR_FORMAT 配置软件设置的 bCharFormat 值，该值由 GET_LINE_CODING 命令请求。(R/W)

Register 37.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



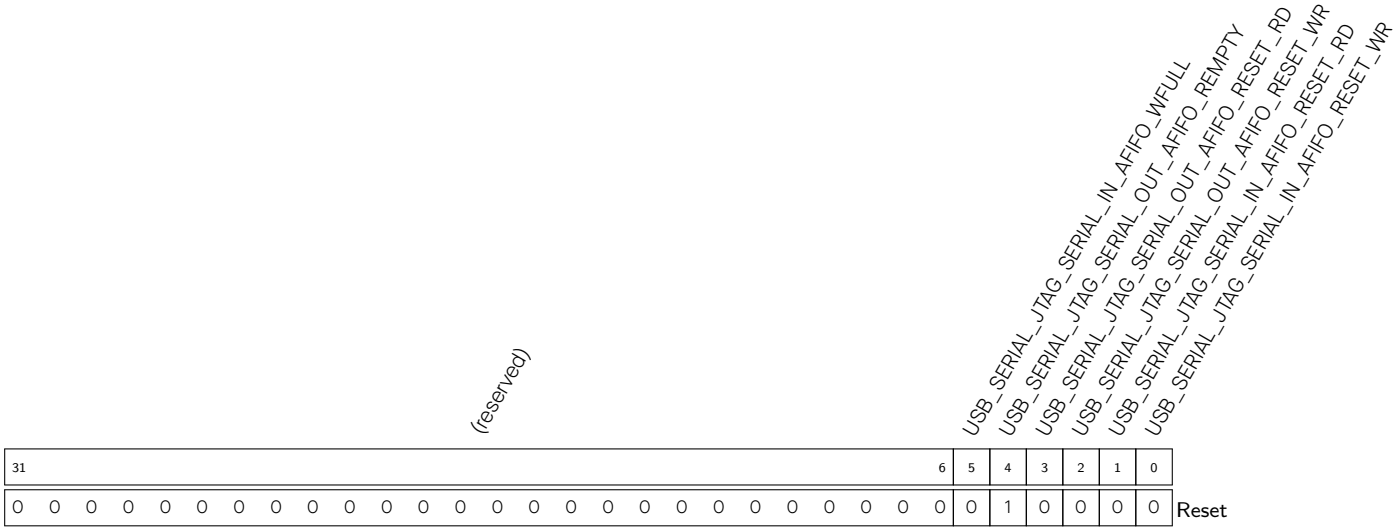
USB_SERIAL_JTAG_CONFIG_UPDATE 配置是否将配置寄存器的值从 APB 时钟域更新到 48 MHz 时钟域。

0: 无效

1: 更新

(WT)

Register 37.11. USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG (0x0064)



USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_WR 配置是否复位 CDC_ACM IN 异步 FIFO 写入时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_RD 配置是否复位 CDC_ACM IN 异步 FIFO 读取时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_WR 配置是否复位 CDC_ACM OUT 异步 FIFO 写入时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_RD 配置是否复位 CDC_ACM OUT 异步 FIFO 读取时钟域。

0: 无效

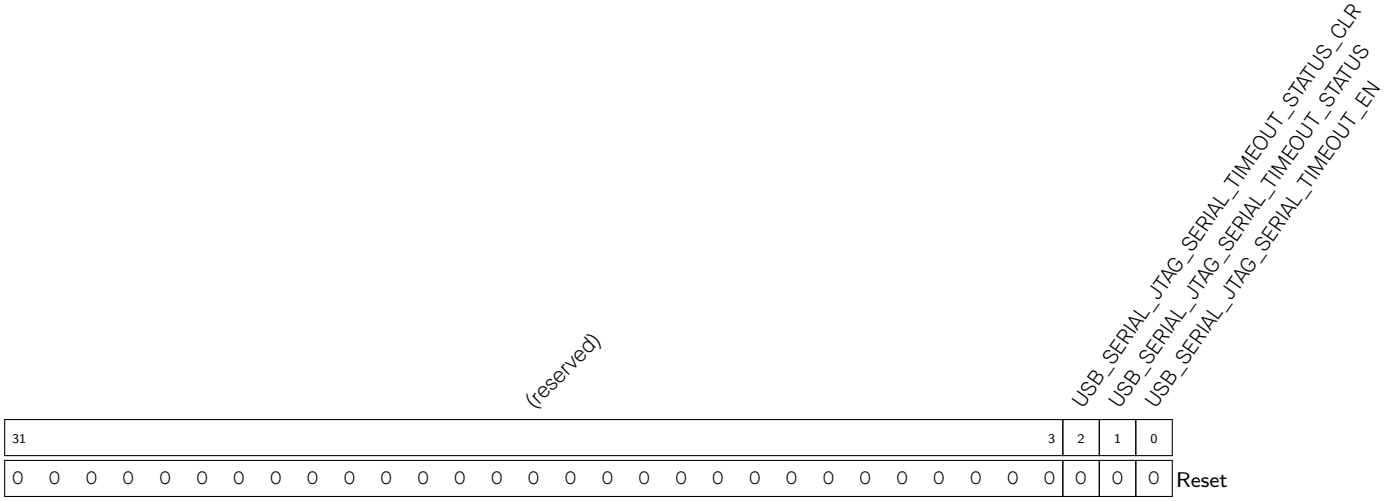
1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_REMPTY 表示 CDC_ACM OUT 读取时钟域的异步 FIFO 空信号。(RO)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_WFULL 表示 CDC_ACM IN 写入时钟域的异步 FIFO 满信号。(RO)

Register 37.12. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG (0x006C)



USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN 使能 USB 串口输出端点超时功能。超时事件发生时，串口输出端点缓冲区自动清除，USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS 被置位。

0：禁用超时，缓冲区不会自动清除

1：使能超时

(R/W)

USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS 超时状态。

0：未发生超时

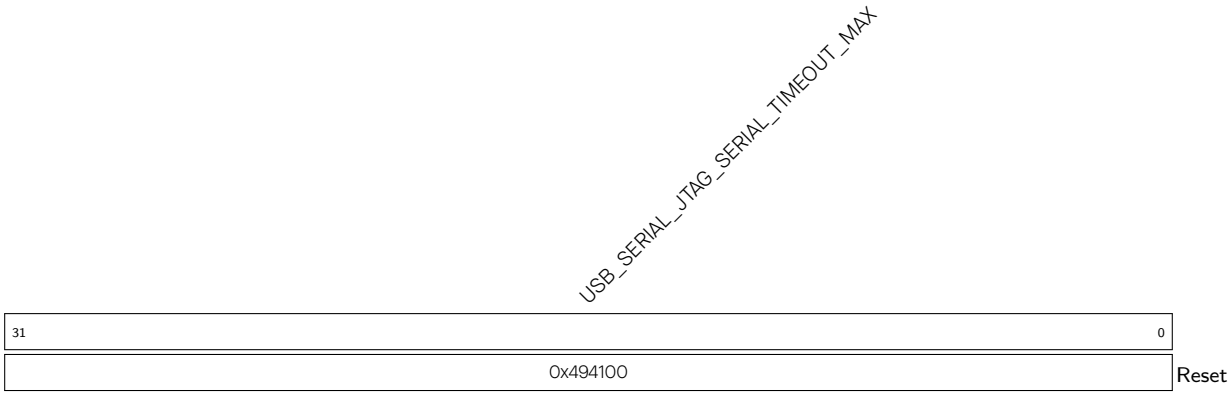
1：串口输出端点触发了超时事件

(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS_CLR 写 1 清除 USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS。

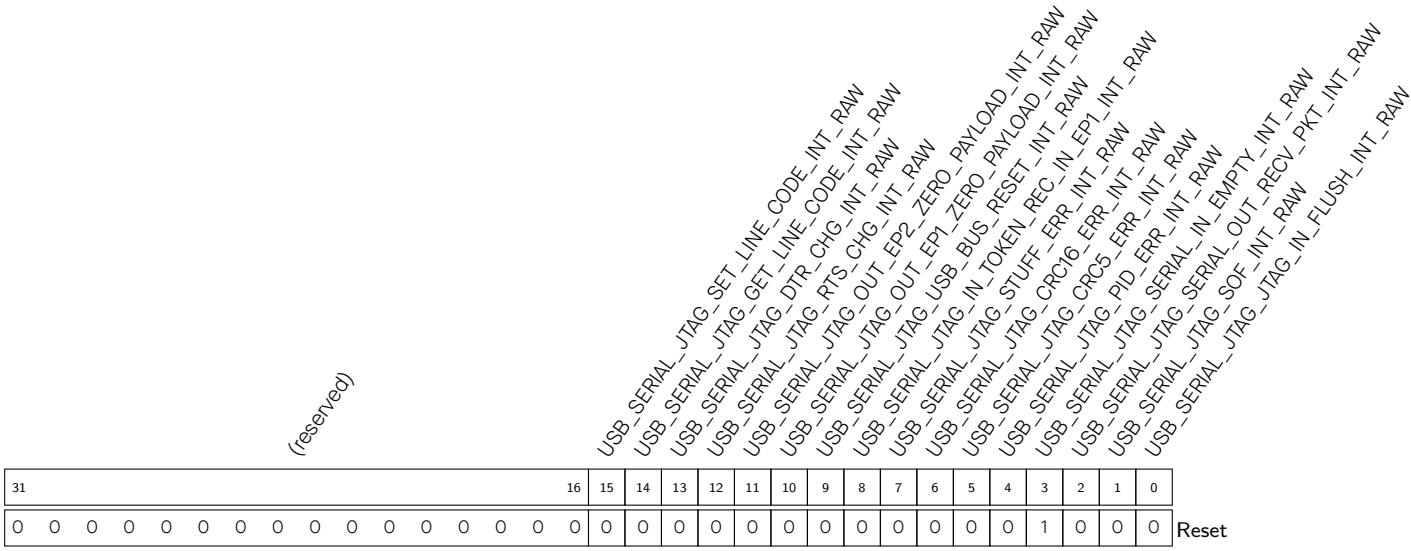
(WT)

Register 37.13. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG (0x0070)



USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX USB 串口输出端点的最大超时数值，表示等待芯片读取数据的最长等待时间，单位为 20.83 ns。(R/W)

Register 37.14. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW USB_SERIAL_JTAG_SOF_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW USB_SERIAL_JTAG_PID_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW USB_SERIAL_JTAG_CRC5_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW USB_SERIAL_JTAG_CRC16_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW USB_SERIAL_JTAG_STUFF_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW USB_SERIAL_JTAG_USB_BUS_RESET_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 的原始中断状态。(R/WTC/SS)

见下页...

Register 37.14. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

接上页...

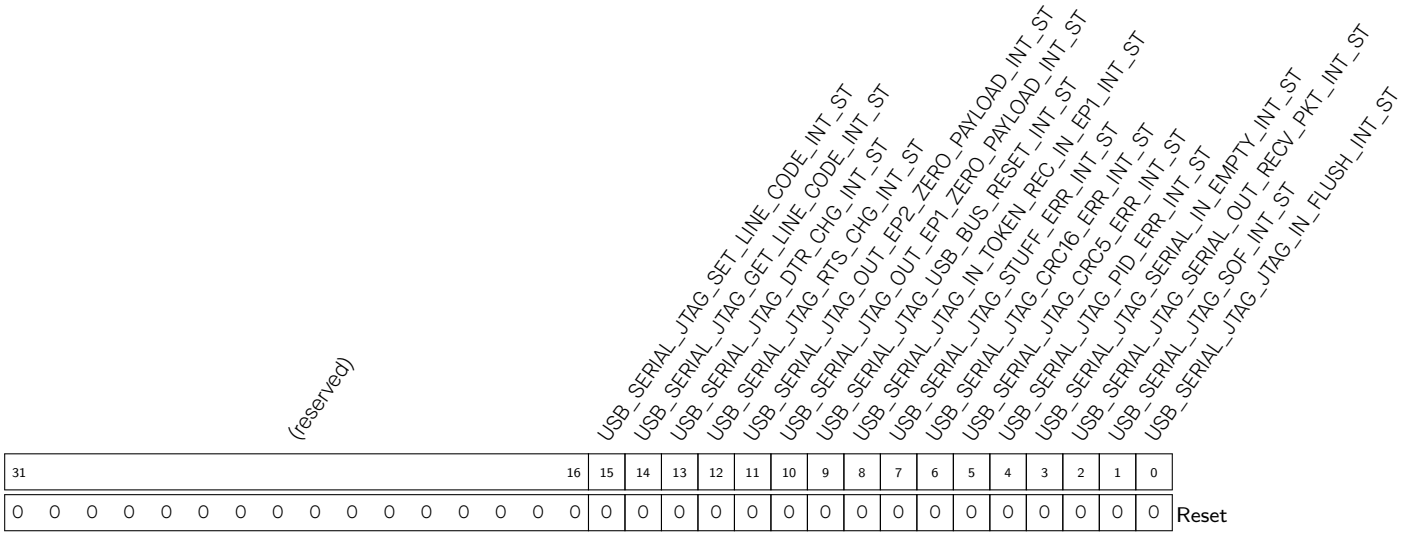
USB_SERIAL_JTAG_RTS_CHG_INT_RAW USB_SERIAL_JTAG_RTS_CHG_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW USB_SERIAL_JTAG_DTR_CHG_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_GET_LINE_CODE_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_SET_LINE_CODE_INT 的原始中断状态。(R/WTC/SS)

Register 37.15. USB_SERIAL_JTAG_INT_ST_REG (0x000C)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SOF_INT_ST USB_SERIAL_JTAG_SOF_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST USB_SERIAL_JTAG_PID_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST USB_SERIAL_JTAG_CRC5_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST USB_SERIAL_JTAG_CRC16_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST USB_SERIAL_JTAG_STUFF_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST USB_SERIAL_JTAG_USB_BUS_RESET_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

见下页...

Register 37.15. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

接上页...

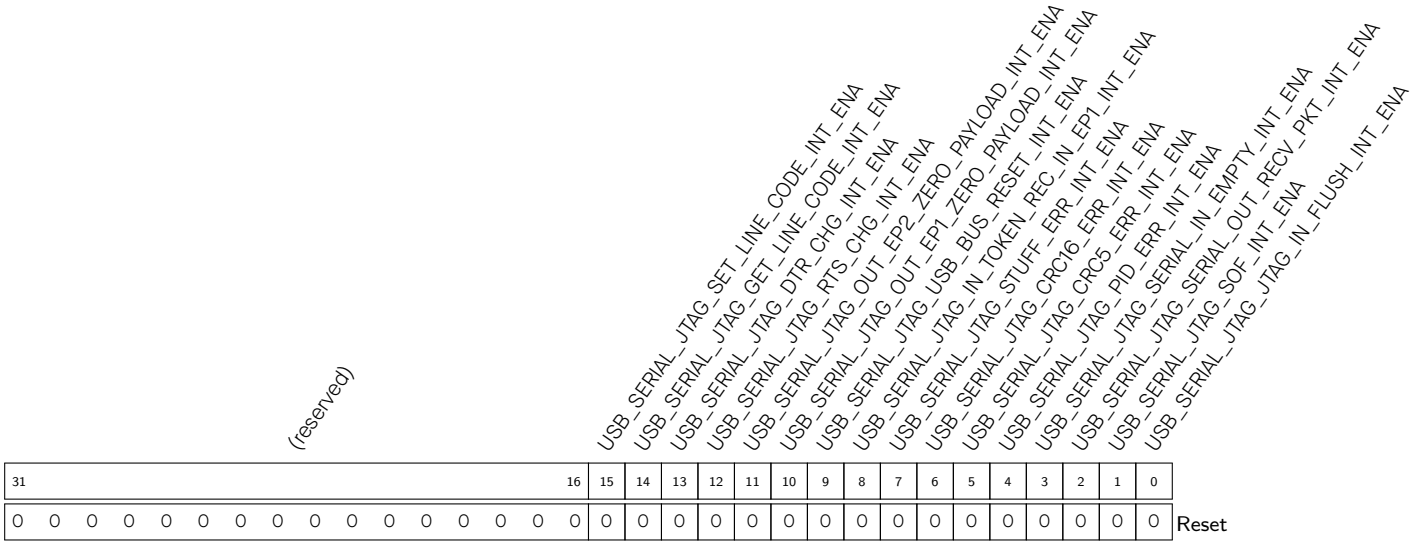
USB_SERIAL_JTAG_RTS_CHG_INT_ST USB_SERIAL_JTAG_RTS_CHG_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST USB_SERIAL_JTAG_DTR_CHG_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST USB_SERIAL_JTAG_GET_LINE_CODE_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST USB_SERIAL_JTAG_SET_LINE_CODE_INT 的屏蔽中断状态。(RO)

Register 37.16. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA 写 1 使能 USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 中断。(R/W)

USB_SERIAL_JTAG_SOF_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SOF_INT 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断。(R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_PID_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_CRC5_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_CRC16_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_STUFF_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA 写 1 使能 USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 中断。(R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA 写 1 使能 USB_SERIAL_JTAG_USB_BUS_RESET_INT 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA 写 1 使能 USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA 写 1 使能 USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 中断。(R/W)

见下页...

Register 37.16. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

接上页...

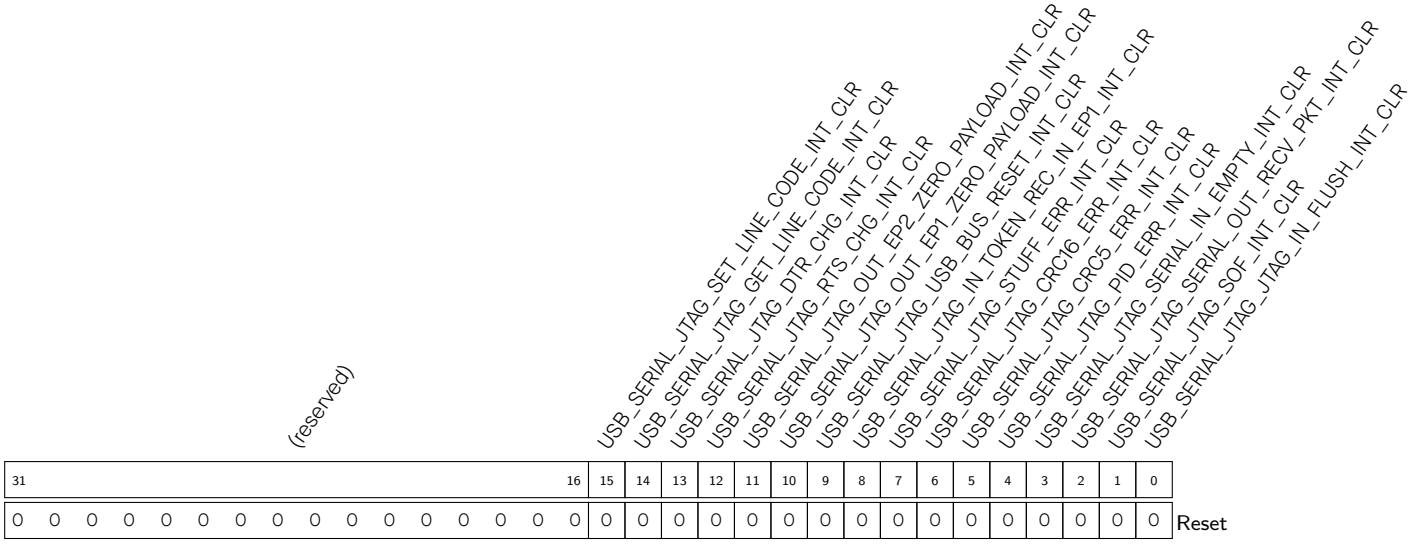
USB_SERIAL_JTAG_RTS_CHG_INT_ENA 写 1 使能 USB_SERIAL_JTAG_RTS_CHG_INT 中断。(R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA 写 1 使能 USB_SERIAL_JTAG_DTR_CHG_INT 中断。(R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA 写 1 使能 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。(R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断。(R/W)

Register 37.17. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR 写 1 清除 USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 中断。(WT)

USB_SERIAL_JTAG_SOF_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SOF_INT 中断。(WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断。(WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断。(WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_PID_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_CRC5_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_CRC16_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_STUFF_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR 写 1 清除 USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 中断。(WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR 写 1 清除 USB_SERIAL_JTAG_USB_BUS_RESET_INT 中断。(WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR 写 1 清除 USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 中断。(WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR 写 1 清除 USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 中断。(WT)

见下页...

Register 37.17. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

接上页...

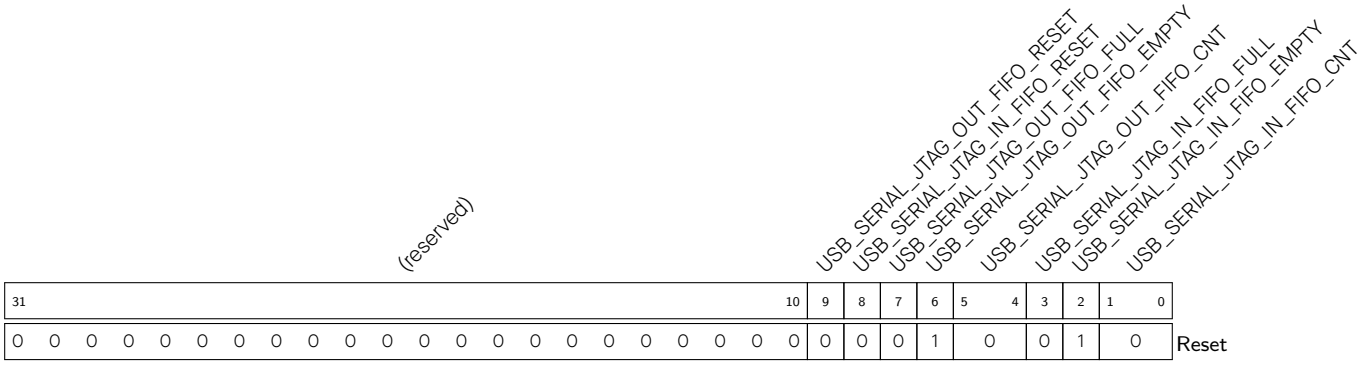
USB_SERIAL_JTAG_RTS_CHG_INT_CLR 写 1 清除 USB_SERIAL_JTAG_RTS_CHG_INT 中断。(WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR 写 1 清除 USB_SERIAL_JTAG_DTR_CHG_INT 中断。(WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR 写 1 清除 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。(WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断。(WT)

Register 37.18. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT 表示 JTAG IN FIFO 计数器。(RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY 表示 JTAG IN FIFO 是否为空。

0：JTAG IN FIFO 不为空

1：JTAG IN FIFO 为空

(RO)

USB_SERIAL_JTAG_IN_FIFO_FULL 表示 JTAG IN FIFO 是否为满。

0：JTAG IN FIFO 不为满

1：JTAG IN FIFO 为满

(RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT 表示 JTAG OUT FIFO 计数器。(RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY 表示 JTAG OUT FIFO 是否为空。

0：JTAG OUT FIFO 不为空

1：JTAG OUT FIFO 为空

(RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL 表示 JTAG OUT FIFO 是否为满。

0：JTAG OUT FIFO 不为满

1：JTAG OUT FIFO 为满

(RO)

USB_SERIAL_JTAG_IN_FIFO_RESET 配置是否复位 JTAG IN FIFO。

0：无效

1：复位

(R/W)

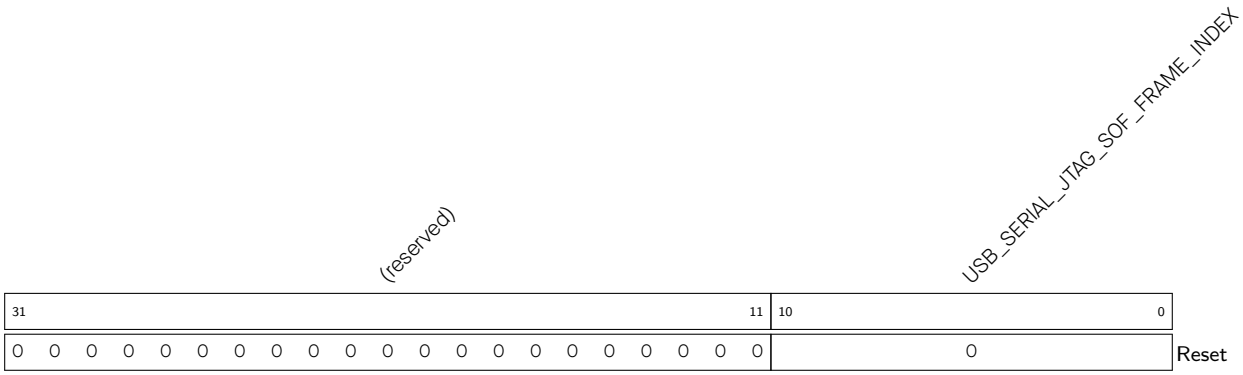
USB_SERIAL_JTAG_OUT_FIFO_RESET 配置是否复位 JTAG OUT FIFO。

0：无效

1：复位

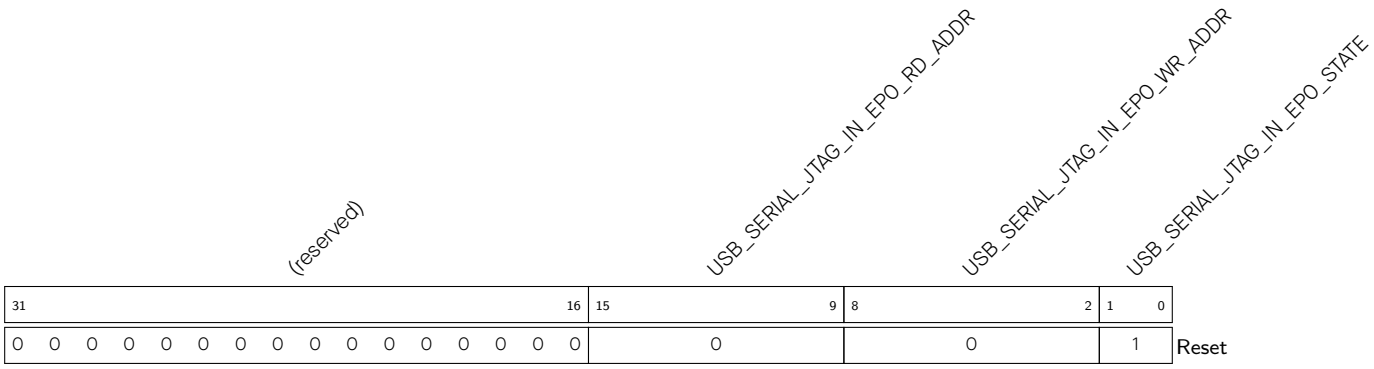
(R/W)

Register 37.19. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)



USB_SERIAL_JTAG_SOF_FRAME_INDEX 接收 SOF 帧的帧索引。(RO)

Register 37.20. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

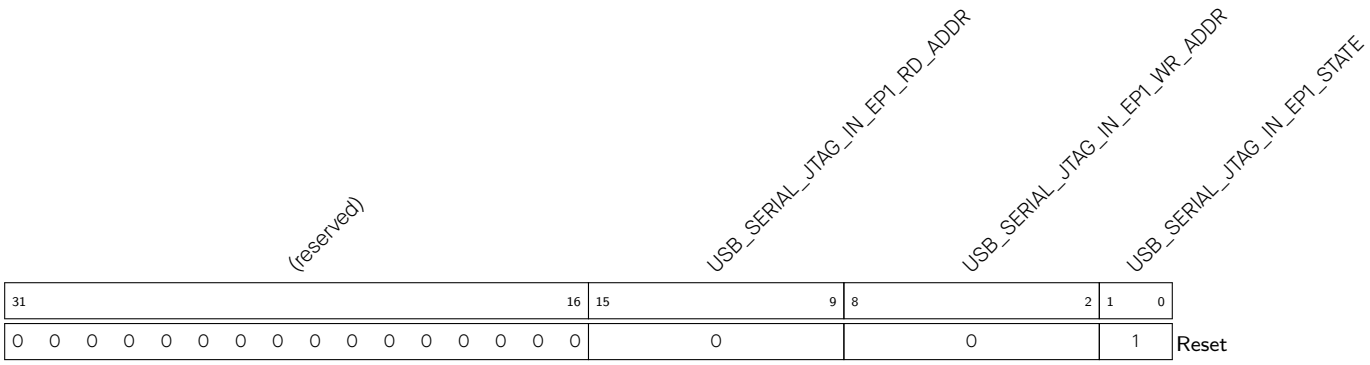


USB_SERIAL_JTAG_IN_EP0_STATE 表示输入端点 0 的状态。(RO)

USB_SERIAL_JTAG_IN_EP0_WR_ADDR 表示输入端点 0 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP0_RD_ADDR 表示输入端点 0 的读取数据地址。(RO)

Register 37.21. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

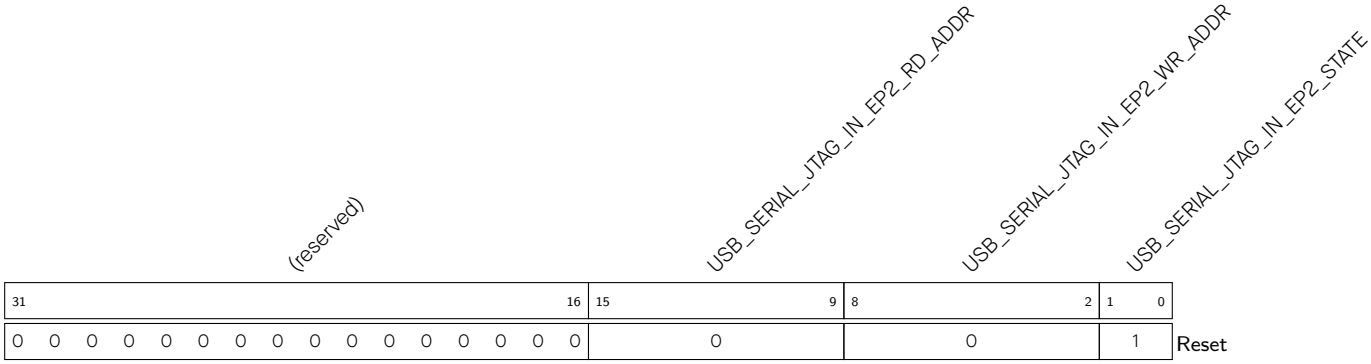


USB_SERIAL_JTAG_IN_EP1_STATE 表示输入端点 1 的状态。(RO)

USB_SERIAL_JTAG_IN_EP1_WR_ADDR 表示输入端点 1 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP1_RD_ADDR 表示输入端点 1 的读取数据地址。(RO)

Register 37.22. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

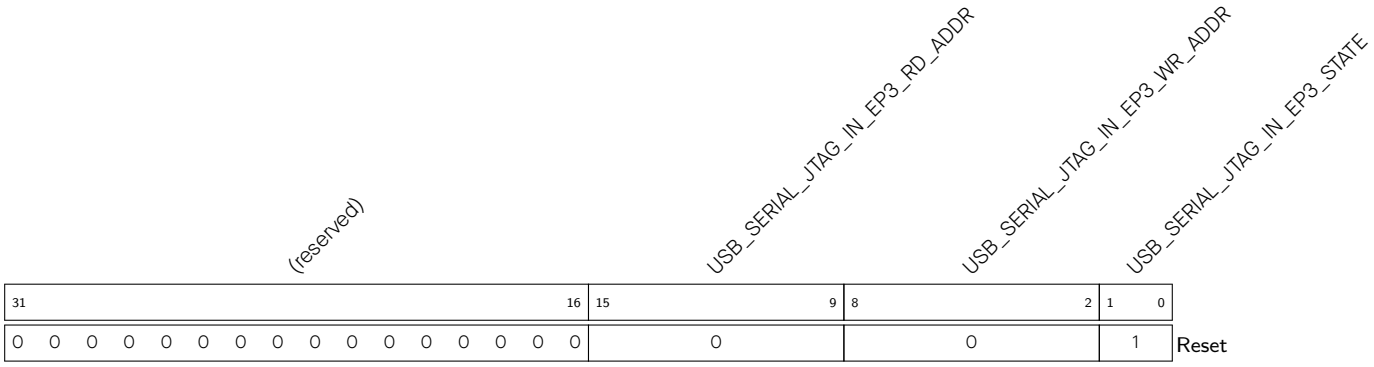


USB_SERIAL_JTAG_IN_EP2_STATE 表示输入端点 2 的状态。(RO)

USB_SERIAL_JTAG_IN_EP2_WR_ADDR 表示输入端点 2 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP2_RD_ADDR 表示输入端点 2 的读取数据地址。(RO)

Register 37.23. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

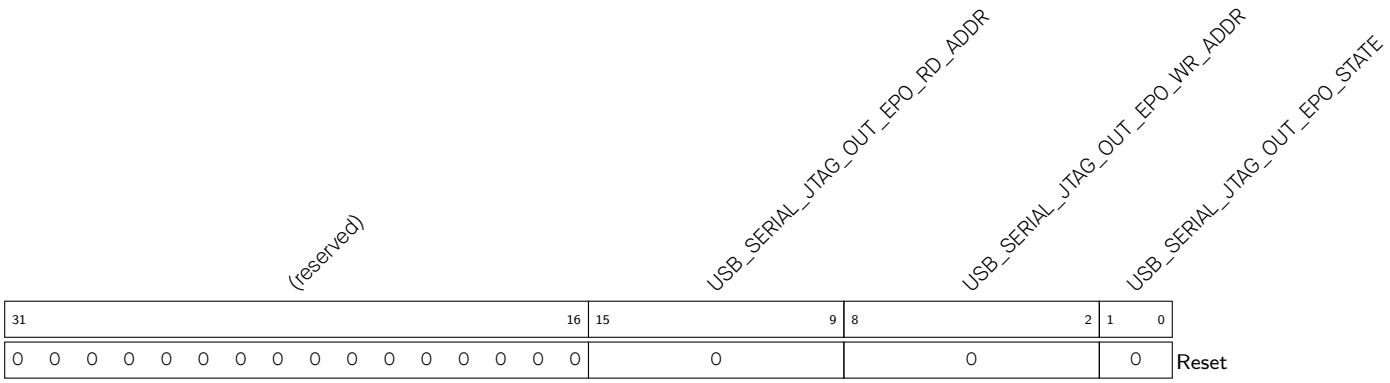


USB_SERIAL_JTAG_IN_EP3_STATE 表示输入端点 3 的状态。(RO)

USB_SERIAL_JTAG_IN_EP3_WR_ADDR 表示输入端点 3 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP3_RD_ADDR 表示输入端点 3 的读取数据地址。(RO)

Register 37.24. USB_SERIAL_JTAG_OUT_EP0_ST_REG (0x0038)

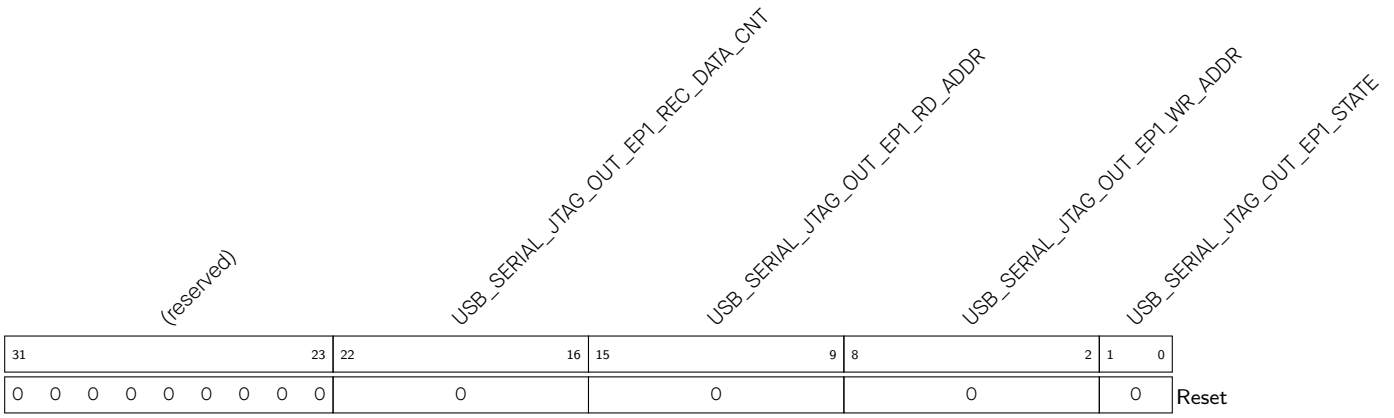


USB_SERIAL_JTAG_OUT_EP0_STATE 表示输出端点 0 的状态。(RO)

USB_SERIAL_JTAG_OUT_EP0_WR_ADDR 表示输出端点 0 的写入数据地址。
当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 0 中有 (USB_SERIAL_JTAG_OUT_EP0_WR_ADDR − 2) 个字节数据。
(RO)

USB_SERIAL_JTAG_OUT_EP0_RD_ADDR 表示输出端点 0 的读取数据地址。(RO)

Register 37.25. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



USB_SERIAL_JTAG_OUT_EP1_STATE 表示输出端点 1 的状态。(RO)

USB_SERIAL_JTAG_OUT_EP1_WR_ADDR 表示输出端点 1 的写入数据地址。

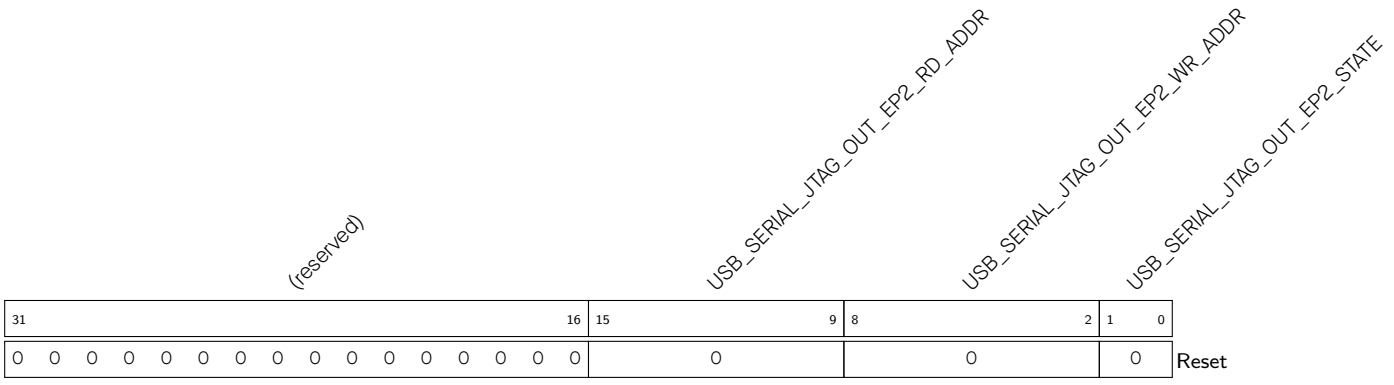
当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 1 中有 (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR – 2) 个字节数据。

(RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR 表示输出端点 1 的读取数据地址。(RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT 当接收到一个数据包时输出端点 1 中的数据计数器。(RO)

Register 37.26. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)



USB_SERIAL_JTAG_OUT_EP2_STATE 表示输出端点 2 的状态。(RO)

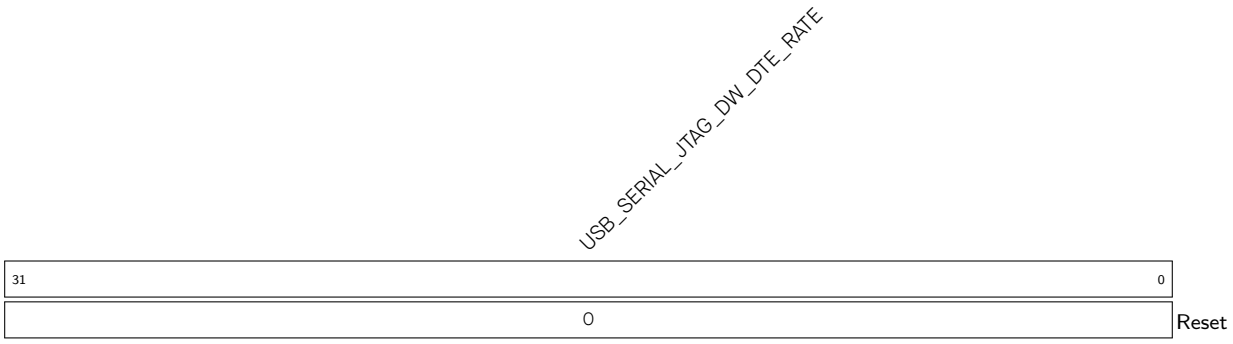
USB_SERIAL_JTAG_OUT_EP2_WR_ADDR 表示输出端点 2 的写入数据地址。

当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 2 中有 (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR – 2) 个字节数据。

(RO)

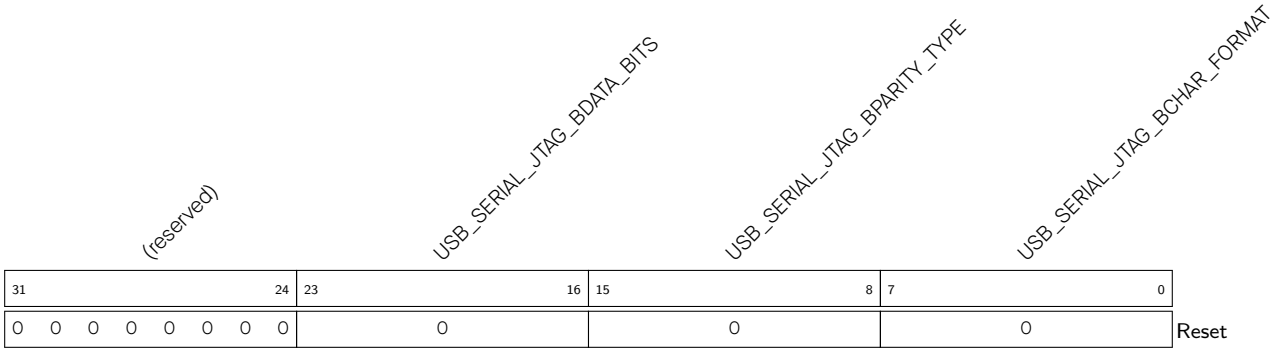
USB_SERIAL_JTAG_OUT_EP2_RD_ADDR 表示输出端点 2 的读取数据地址。(RO)

Register 37.27. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE 表示主机通过 SET_LINE_CODING 命令设置的 dwDTERate 值。(RO)

Register 37.28. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

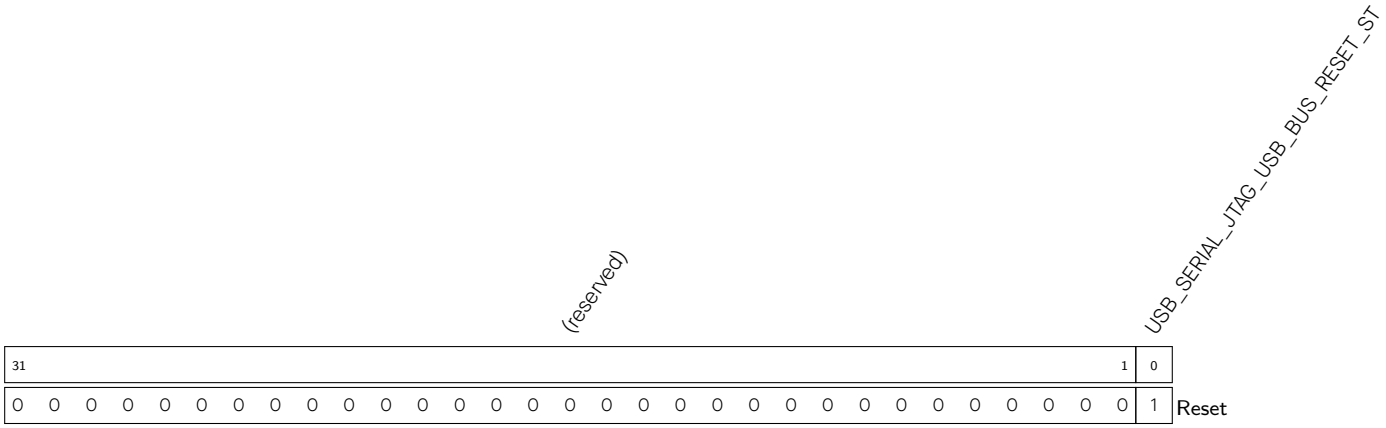


USB_SERIAL_JTAG_BCHAR_FORMAT 表示主机通过 SET_LINE_CODING 命令设置的 bCharFormat 值。(RO)

USB_SERIAL_JTAG_BPARITY_TYPE 表示主机通过 SET_LINE_CODING 命令设置的 bParityTpye 值。(RO)

USB_SERIAL_JTAG_BDATA_BITS 表示主机通过 SET_LINE_CODING 命令设置的 bDataBits 值。(RO)

Register 37.29. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



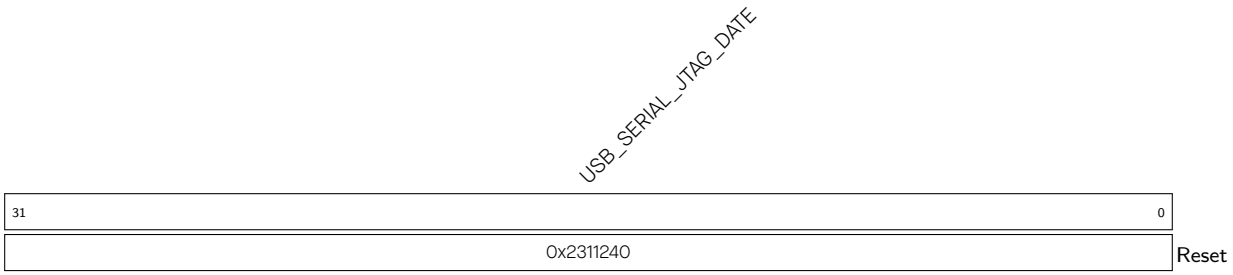
USB_SERIAL_JTAG_USB_BUS_RESET_ST 表示 USB 总线复位是否被释放。

0：USB 串口/JTAG 控制器处于 USB 总线复位状态

1：USB 总线复位被释放

(RO)

Register 37.30. USB_SERIAL_JTAG_DATE_REG (0x0080)



USB_SERIAL_JTAG_DATE 版本控制寄存器。(R/W)

第 38 章

CAN FD

CAN FD（Controller Area Network Flexible Data-Rate，控制器局域网灵活数据速率）协议是一种多主机、多播的通信协议，具有错误检测、发送错误信号以及内置报文优先仲裁等功能，遵循 ISO11898-1 规定的 CAN 协议。CAN 协议适用于汽车和工业应用。

ESP32-C5 中集成的 CAN FD 控制器与 CAN FD 规范协议兼容，但尚未获得正式认证。除非另有说明，本章中提到的 CAN FD 控制器均指兼容 CAN FD 协议的控制器。

38.1 概述

ESP32-C5 包含两个 CAN FD 控制器，可以通过外部收发器连接到 CAN FD 总线。CAN FD 支持许多高级功能，应用场景广泛，如汽车产品、工业自动化控制、楼宇自动化等。

本章节提供了 CAN FD 的功能描述、程序模型和参数，旨在为软件驱动开发人员提供参考。

38.2 主要特性

ESP32-C5 的 CAN FD 具有以下特性：

- 兼容 ISO11898-1:2015 标准
- 具有 128 字的 RX FIFO（6 个 64 字节载荷的 CAN FD 帧，21 个 8 字节载荷的 CAN/CAN FD 帧）
- 4 个 TX buffer（每个 TX buffer 中有 1 个 CAN FD 帧）
- 32 位 APB 配置接口
- 支持 ISO 和非 ISO CAN FD 协议
- 支持时间戳和定时发送
- 中断
- 内置 3 个屏蔽过滤器和 1 个范围过滤器
- 回环模式、总线监控模式、禁止确认模式、自检模式、受限操作模式

38.3 功能描述

38.3.1 时钟

CAN FD 采用单一时钟设计，唯一的时钟是 TWAI_CLK，所有时间参数均由此派生。

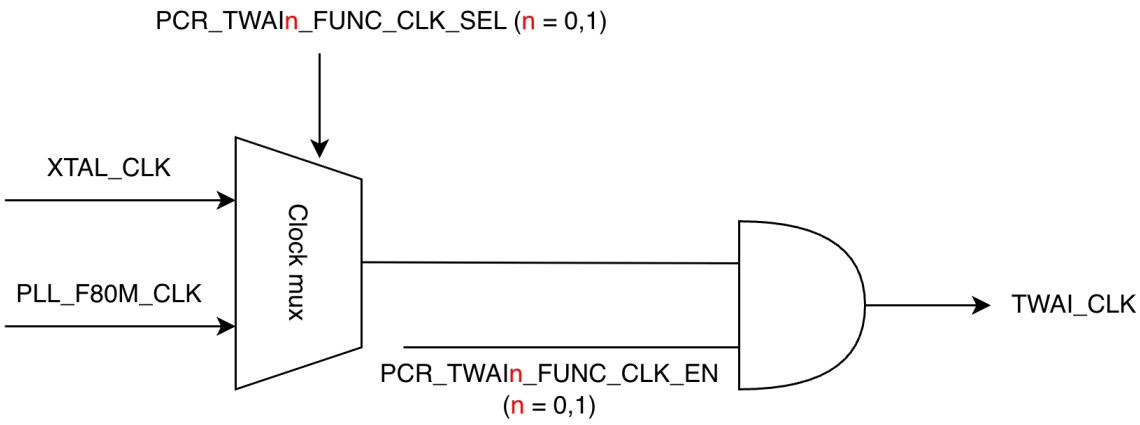


图 38.3-1. TWAI_CLK Diagram

ESP32-C5 CAN FD 支持 XTAL_CLK 与 PLL_F80M_CLK 两种时钟源选择：

- XTAL_CLK (外部晶振)：高精度时钟源，支持最高频率为 48 MHz。
- PLL_F80M_CLK (锁相环时钟)：倍频时钟源，最高工作频率可达 80 MHz。

可通过 PCR_TWAI0_FUNC_CLK_SEL 和 PCR_TWAI1_FUNC_CLK_SEL 寄存器选择两个 CAN FD 控制器的时钟源。

- 当数据速率 ≤ 5 Mbit/s 时：推荐选用 XTAL_CLK，以获得更高的频率稳定度。
- 当数据速率 > 5 Mbit/s 时：推荐选用 PLL_F80M_CLK，以满足高带宽需求。

具体的位速率（相关介绍详见 38.3.7.1 位速率）设定，请参考 TWAIFD_BTR_FD_REG 和 TWAIFD_BTR_REG 寄存器的参数计算说明。

38.3.2 复位

上电后，CAN FD 应通过硬件或软件进行复位。将 1 写入 TWAIFD_RST 可以实现软件复位。如需硬件复位，则 CAN FD 在系统时钟的两个时钟周期内不可访问。例如，如果 CAN FD 系统时钟为 100 MHz，软件应在硬件复位释放后等待 20 ns 再访问 CAN FD。如需软件复位，则不必等待。硬件复位和软件复位作用相同。无论采用哪种复位方式，CAN FD 都将进入以下状态：

- 禁用 CAN FD，不能在 CAN 总线上通信（离线状态）。
- CAN FD 所有内存寄存器取复位值。
- CAN FD 内存（TX buffer 和 RX buffer）未复位。

38.3.3 时间基准

CAN FD 集成了独立的 32 位定时器，可以产生时间戳，为定时发送或接收帧的提供时间基准。时间基准定时器可以通过配置 TWAIFD_TIMER_CE 启用，然后通过 TWAIFD_TIMESTAMP_HIGH 和 TWAIFD_TIMESTAMP_LOW 读取其当前值。

时间基准是一个向上计数的无符号定时器，TWAIFD_TIMER_STEP 配置计时步长，TWAIFD_TIMER_LD_VAL_L 配置定时器的初始值，TWAIFD_TIMER_CT_VAL_L 配置定时器的终到值。时间基准的有效位数可通过 TWAIFD_TS_BITS 寄存器读取或配置，有效范围为 1 到 32。

38.3.4 工作模式

复位后，CAN FD 处于禁用状态，不参与 CAN 总线通信（不发送、接收或监控数据）。在启用 CAN FD 之前，必须按照章节 38.3.7 *CAN 总线配置* 中的说明完成必要配置。配置完成之后，对 TWAIFD_ENA 写 1 启用 CAN FD。当 TWAIFD_ENA = 1 时，CAN FD 开始接入 CAN 总线，并在接收到 11 个连续的隐性位后正式加入 CAN 总线通信。加入总线后，CAN FD 进入主动错误状态（在接入过程中处于离线状态），开始在 CAN 总线上进行数据交互。CAN FD 加入 CAN 总线通信的时刻可以通过 FCS 中断以及后续读取 TWAIFD_EWL_ERP_FAULT_STATE_REG 来确定（详见章节 38.4 *中断*）。CAN FD 的基本工作模式如图 38.3-2 所示。

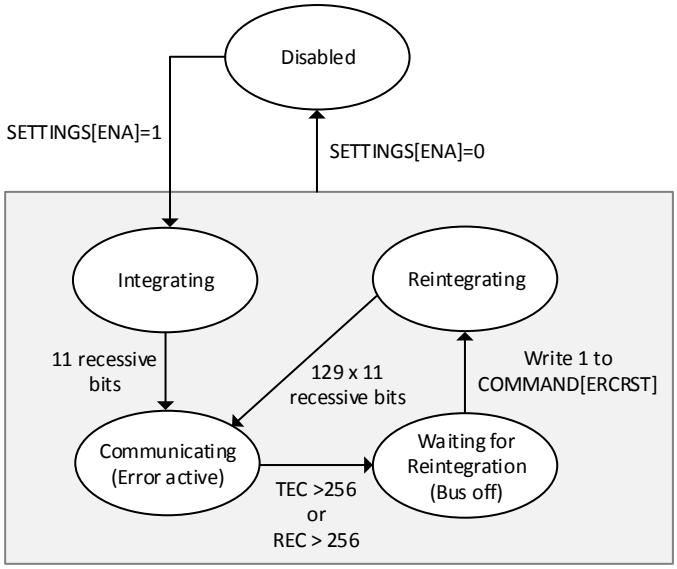


图 38.3-2. 工作模式

CAN FD 处于主动错误状态时，仍然参与 CAN 总线通信。如果 CAN FD 变为被动错误状态并随后进入离线状态，则会停止在 CAN 总线上通信，直至收到错误计数器复位命令（向 TWAIFD_ERCRST 写 1）。收到此命令后，CAN FD 开始重新接入总线，直到 CAN FD 检测到 129 组由 11 个连续隐性位组成的序列（实现层面的选择，超过 CAN 规范规定的最小值 128 个序列），之后再次进入主动错误状态。

清零 TWAIFD_ENA 寄存器，可以随时禁用 CAN FD。在这种情况下：

- CAN FD 立即停止在 CAN 总线上通信，并仅发送隐性位。
- TX/RX 错误计数器 (TEC/REC) 复位为 0，CAN FD 进入离线状态。
- 所有 TX buffer 切换至 empty 状态（见图 38.3-8），TX buffer RAM 中的内容仍然有效（内存未复位）。
- RX buffer 刷新（详见章节 38.3.9.5 *刷新*）。

建议在清零 TWAIFD_ENA 禁用 CAN FD 之前，避免发送任何帧，防止 CAN 总线上其他节点发送错误帧。因此，操作 CAN FD 的软件驱动程序应确保 CAN FD 内的任何 TX buffer 不处于 ready、TX in progress 或 abort in progress 状态（详见章节 38.3.8 *CAN 帧发送*）。

TWAIFD_ERCRST 是一个“粘性”命令。这意味着如果 CAN FD 尚未关闭总线且此命令已发出，CAN FD 将记录此命令，在进入离线状态的瞬间自动触发重新接入总线。这样，软件驱动程序可以在通信期间提前发送该命令，从而在离线后立即触发重新接入，避免额外的软件延迟。

38.3.5 初始化顺序

初始化顺序包括以下步骤：

1. 复位（硬件或软件复位）
2. 配置 CAN FD：
   (a) 按照章节 38.4 中断 配置中断
   (b) 按照章节 38.3.7.1 位速率 配置位速率
   (c) 配置过滤器、特殊模式等其他功能
3. 对 TWAIFD_ENA 写 1 启用 CAN FD。
4. 轮询 TWAIFD_EWL_ERP_FAULT_STATE_REG 寄存器，或等待故障限制状态触发中断 TWAIFD_FCSIINTST。TWAIFD_ERA = 1（CAN FD 进入主动错误状态）表示接入完成。
5. 初始化完成，软件驱动程序可以发送和接收帧。

38.3.6 反初始化顺序

反初始化顺序包括以下步骤：

1. 确保没有任何 TX buffer 处于 ready、TX in progress 或 abort in progress 状态：向 TX buffer 发出 set abort 命令（详见章节 38.3.8 *CAN 帧发送*），并且避免将新的帧插入 TX buffer 进行传输。
2. 清零 TWAIFD_ENA。

38.3.7 CAN 总线配置

38.3.7.1 位速率

CAN 总线上的位速率来源于系统时钟（参见章节 38.3.1 时钟）。CAN 总线上的基本时间单位是时间定额 (time quanta)，计算方式为将系统时钟频率除以位速率预分频器的值。CAN FD 具有独立的预分频器用于标称位速率（通过 TWAIFD_BRP 配置）和数据位速率（通过 TWAIFD_BRP_FD 配置）。配置位速率，可通过 TWAIFD_BTR_REG（标称位速率）和 TWAIFD_BTR_FD_REG（数据位速率）寄存器指定 Prop_Seg、Phase_Seg1 和 Phase_Seg2 的持续时间，如图 38.3-3 所示。

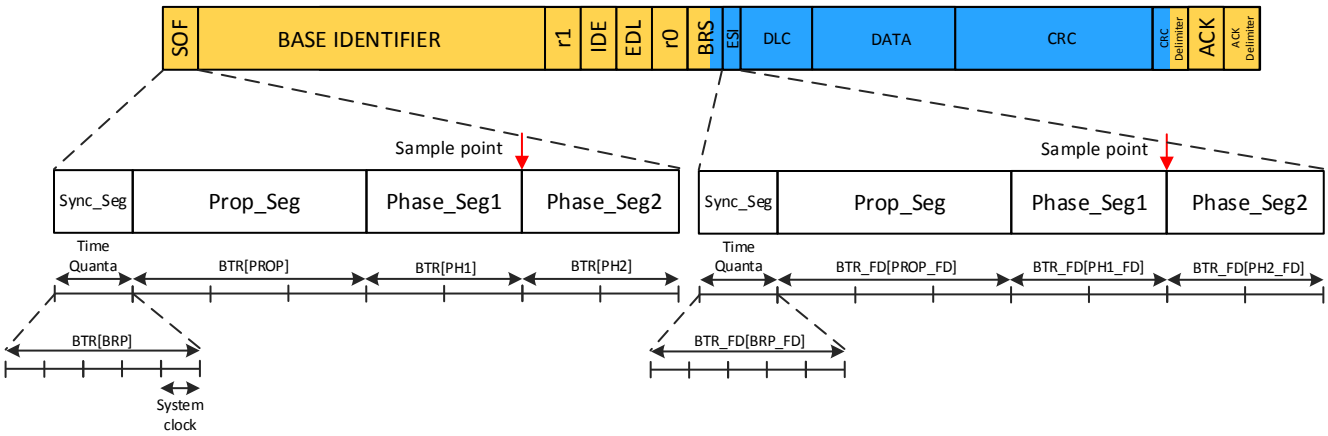


图 38.3-3. 位时间

配置流程详见章节 38.5.1 *500 kbit/4 Mbit 示例*。

38.3.7.2 发送器延迟

发送器延迟是 CAN FD 在 CAN_TX 输出上发送的信号返回 CAN_RX 输入的传播延迟，如图 38.3-4 所示。此延迟包括信号传播到物理层收发器的延迟、收发器本身的延迟，以及从收发器到 CAN_RX 输入的延迟。无论帧中是否切换位速率，CAN FD 在发送 CAN FD 帧时会测量其自身的发送器延迟。测量位置在 FDF (EDL) 位和 RO 位之间由隐性到显性的边沿，如图 38.3-5 所示。发送器延迟可以通过 TWAIFD_TRV_DELAY_VALUE 进行测量，其单位为系统时钟周期。

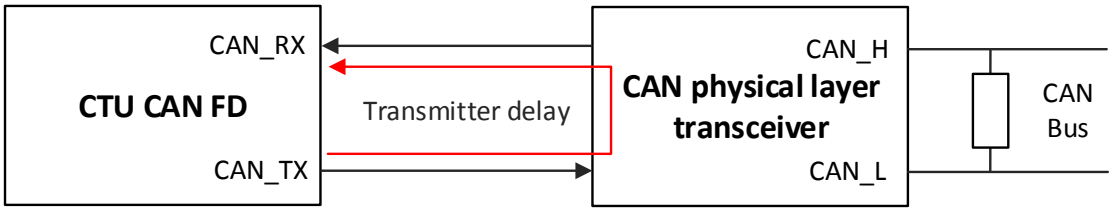


图 38.3-4. 发送器延迟

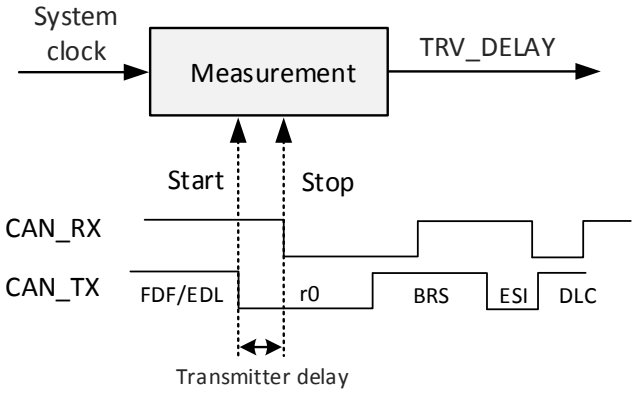


图 38.3-5. 发送器延迟测量

测量的发送器延迟包括 CAN FD 的输入延迟，即系统时钟的 2 个周期。因此，测量的发送器延迟将始终比从 CAN_TX 到 CAN_RX 的实际延迟多两个周期。例如，如果从 CAN_TX 到 CAN_RX 的信号传播需要 110 ns（在 100 MHz 时为 11 个系统时钟周期），测量的发送器延迟将为 13 个周期。

发送器延迟的测量有最大值限制，127 个系统时钟周期饱和。如果 CAN_TX 和 CAN_RX 之间的延迟超过此值，测量结果将为 127 个周期。当系统时钟频率为 100 MHz 时，这意味着最大可测量的发送器延迟为 1.27 μs，此值大于大多数 CAN 收发器所需的延迟。

38.3.7.3 二次采样点

二次采样点用于在数据位速率期间由 CAN FD 检测比特错误。二次采样点的位置从位时间的开始 (Sync_Seg) 起，表示为系统时钟周期的延迟（而不是时间定额）。二次采样点位置可以是固定的（通过 TWAIFD_SSP_OFFSET 配置），从发送器延迟派生（通过 TWAIFD_SSP_OFFSET 和 TWAIFD_TRV_DELAY_VALUE 配置），或者可以禁用（无 SSP），如图 38.3-6 所示。当禁用二次采样点时，CAN FD 在数据位速率下发送时使用 hyperref[regdesc:TWAIFDBTRREG]TWAIFD_BTR_REG 配置的常规采样点。

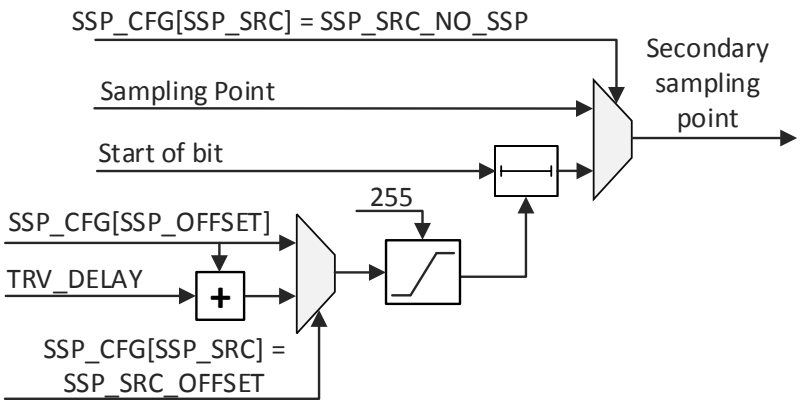


图 38.3-6. 二次采样点

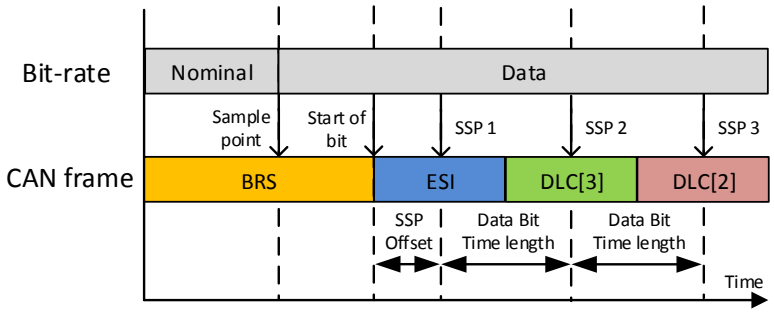


图 38.3-7. 二次采样点 2

二次采样点偏移量 (TWAIFD_SSP_OFFSET) 可配置范围为 0 ~ 255，二次采样点位置的内部范围同为 0 ~ 255。如果出于某些原因计算出的二次采样点位置超过 255 个时钟周期（例如测量的发送器延迟较大），则其值最大为 255。

由于 CAN FD 具有 2 个系统时钟周期的输入延迟（最小时间定额），建议二次采样点偏移量至少设置为 2 以补偿该延迟。如果 TWAIFD_SSP_OFFSET < 3 且 TWAIFD_SSP_SRC = SSP_SRC_OFFSET，则 CAN FD 无法检测自身在 CAN FD 发送帧的比特错误。

CAN FD 在使用二次采样点时最多可以处理 CAN_TX 和 CAN_RX 管脚之间的 4 个飞行比特。例如，在系统时钟频率为 100 MHz、数据位速率为 5 Mbit/s 的情况下，一个数据位时间对应 20 个系统时钟周期。因此，二次采样点的最晚可能位置应设定为 4 × 20 = 80 个系统时钟周期。用户在配置二次采样点时，应确保其最终位置（包括 TWAIFD_SSP_OFFSET/TWAIFD_TRV_DELAY_VALUE）不超过 4 个数据位时间。

38.3.7.4 CAN FD 支持

CAN FD 支持选择 ISO 和非 ISO 版本的 CAN FD 协议，可以通过 TWAIFD_NISOFD 配置。选择 ISO 协议版本时，CAN FD 符合 ISO11898-1:2015 规范。选择非 ISO 版本时，CAN FD 符合 CAN FD 1.0 规范。只有在 CAN FD 禁用时 (TWAIFD_ENA = 0)，才能配置 TWAIFD_NISOFD。

38.3.7.5 协议异常处理

CAN FD 支持协议异常检测，可以通过 TWAIFD_PEX 启用。只有在 CAN FD 禁用时 (TWAIFD_ENA = 0)，才能配置 TWAIFD_PEX。

协议异常的具体行为取决于 CAN 的实现类型，详见表 38.3-1。当 TWAIFD_PEX = 1 且 CAN FD 检测到协议异常，模块将进入总线接入状态，并等待在 CAN_RX 信号上连续检测到 11 个隐性位。

此过程中，协议异常不会影响 REC/TEC 计数器的值，CAN FD 的故障限制状态也保持不变。检测到协议异常将会置位 TWAIFD_PEXS 标志，可通过对 TWAIFD_CPEXS 写 1 来清除该标志。

若 TWAIFD_PEX = 0 （即未启用协议异常检测）且协议异常仍然存在，CAN FD 将发送错误帧。

38.3.7.6 实现类型

ISO11898-1:2015 定义了三种 CAN 协议实现类型：经典 CAN、CAN FD 容忍 (CAN FD tolerant) 和 CAN FD 启用 (CAN FD enabled)。CAN FD 支持三种实现类型。通过 TWAIFD_FDE 和 TWAIFD_PEX 可以配置符合实现类型的协议，上述字段仅在 CAN FD 禁用时 (TWAIFD_ENA = 0) 才能配置。

表 38.3-1. CAN 实现类型

实现类型	TWAIFD_FDE	TWAIFD_PEX	行为
经典 CAN	0	0	CAN FD 检测到隐性 FDF 位（基本帧中 IDE 之后的位和扩展帧中 RTR/R1 之后的位）时，会发送错误帧。
CAN FD 容忍	0	1	CAN FD 检测到隐性 FDF 位时，将视为协议异常，并进入总线接入状态。
CAN FD 启用	1	0	CAN FD 能够接收和发送 CAN FD 帧。CAN FD 在 res 位（FDF 之后的位）检测到隐性值时，会发送错误帧。
CAN FD 启用且协议异常	1	1	CAN FD 能够接收和发送 CAN FD 帧。CAN FD 在 res 位（FDF 之后的位）检测到隐性值时，将视为协议异常，并进入总线接入状态。此配置兼容 CAN FD 协议的未来扩展（例如 CAN XL）。

当 CAN FD 配置为经典 CAN/CAN FD 容忍节点 (TWAIFD_FDE = 0) 时，若用户尝试发送 CAN FD 帧 (TX buffer 中的 FRAME_FORMAT_W[FDF_BIT] = 1)，则模块将忽略该帧的类型，并发送 CAN 2.0 格式的帧。

当 CAN FD 配置为经典 CAN/CAN FD 容忍节点时，TWAIFD_NISOFD 寄存器无效。

根据 ISO11898-1:2015 的第 10.9.10 节，不应将 CAN FD 启用型实现配置为行为类似于 CAN FD 容忍型实现的模式。因此，用户应仅在评估或调试阶段使用该配置选项，并自行承担相关责任。

此外，根据原 CAN 2.0 规范，接收器应接受 R0 和 R1 位的任意取值；然而在 ISO11898-1:2015 中，规范要求经典 CAN 实现在检测到此类情况时发送错误帧。这一规范差异已在本实现中得到修正，以符合 ISO11898-1:2015 的最新要求。

38.3.7.7 最小位时间/最大位速率

系统时钟周期等于最小时间定额，从而影响 CAN 总线上可实现的最小位速率。CAN FD 具有以下限制：

- Phase_Seg2 >= 2 个最小时间定额，适用于标称位速率和数据位速率。
- Sync_Seg + Prop_Seg + Phase_Seg1 > 2 个最小时间定额，适用于标称位速率和数据位速率。

在满足上述条件时，每个位时间的长度最小可为 5 个时间定额。请注意，对于标称位速率而言，5 个时间定额虽然可行，但仍建议每个位时间至少使用 8 个时间定额，以获得更好的通信稳定性。对于数据位速率，可以设置为 5 个时间定额。

例如，当标称位速率为 250 kbit/s、数据位速率为 1 Mbit/s 时，最小可用的系统时钟频率为 5 MHz。但请注意，这已是极限值，采样点位置的调整空间极小。因此，建议在该配置下使用至少 10 MHz 的系统时钟频率，确保有更安全的设计裕量。

38.3.8 CAN 帧发送

CAN 帧通过 CAN FD 从 TX buffer 发起传输。CAN FD 包含 4 个 TX buffer。当前 TX buffer 的数量可从 TWAIFD_TX_COMMAND_TXTB_INFO_REG 中读取。若存在 N 个缓冲区，则缓冲区编号固定为 1 至 N。每个 TX buffer 可以处于图 38.3-8 中描述的状态之一，从 TWAIFD_TX_STATUS_REG 中读取。软件能够通过 TWAIFD_TX_COMMAND_TXTB_INFO_REG 控制 TX buffer 状态，支持以下三种命令：

set ready 请求将 TX buffer 设置为 ready 状态。

set abort 请求将 TX buffer 设置为 aborted 或 abort in progress 状态。

set empty 请求将 TX buffer 设置为 empty 状态。

每个 TX buffer 可存储一帧 CAN 帧。整个 64 字节的 CAN FD 帧可写入单个 TX buffer。TX buffer 为只写寄存器（不支持读取回写入的帧），且仅在 TX buffer 处于 Empty、TX OK、TX failed 或 aborted 状态时可写入。CAN 帧写入 TX buffer 时使用的内存映射区域如下。

内存区域	地址偏移
控制寄存器	0x000
TX buffer 1	0x100
TX buffer 2	0x200
TX buffer 3	0x300
TX buffer 4	0x400

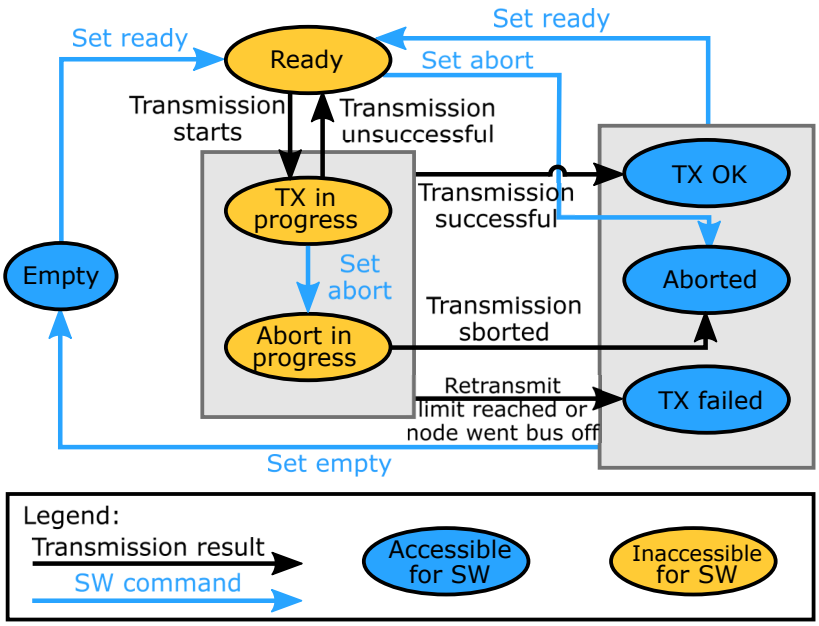


图 38.3-8. TX buffer 状态

当驱动程序将 CAN 帧写入 TX buffer 后，需向该 buffer 发出 set ready 命令以请求发送，此时 TX buffer 进入 ready 状态。在间歇字段的第三位采样到显性位（此时跳过 SOF），或总线空闲时，CAN FD 将从 ready 进入 TX

in progress 状态并开始发送。请注意，在定时发送的模式下行为有所不同，详见 38.3.8.2）。

当 CAN FD 处于被动错误状态，且是上一帧的发送节点时，则会在连续发送前暂停 8 个位时间。当帧发送成功（即无仲裁丢失、无错误帧）时，TX buffer 转为 TX OK 状态。若出现错误帧或仲裁丢失，TX buffer 将重新进入 ready 状态，并在下一个间歇字段或总线空闲时重新尝试发送。

在总线监控模式 (TWAIFD_BMM = 1) 或受限操作模式 (TWAIFD_ROM = 1) 下，即使发出 set ready 命令，CAN FD 也将进入 TX failed 状态，不会尝试发送。

38.3.8.1 TX buffer 选择

如有多个 TX buffer 处于 ready 状态，CAN FD 将从中选择优先级最高的 TX buffer 发送 CAN 帧，优先级由 TWAIFD_TX_PRIORITY_REG 配置。如果两个 TX buffer 的优先级相同，则索引较小的 buffer 优先。发送流程如图38.3-9 所示。

TWAIFD_TX_PRIORITY_REG[TX*P] 值越大表示 TX buffer * 优先级越高。例如，如果 TWAIFD_TX_PRIORITY_REG[TX1P] = 2 且 TWAIFD_TX_PRIORITY_REG[TX2P] = 5，则 TX buffer 2 的优先级为 5，优先级高于 TX buffer 1。两个 TX buffer 都处于 ready 状态时，CAN FD 优先选择 TX buffer 2。

当 TWAIFD_TXBBM = 1 时，候补 TX buffer 不通过其自身的 TWAIFD_TX_PRIORITY_REG[TX*P] 配置优先级，而是由原 buffer 的对应字段配置。详见 38.3.11.3。

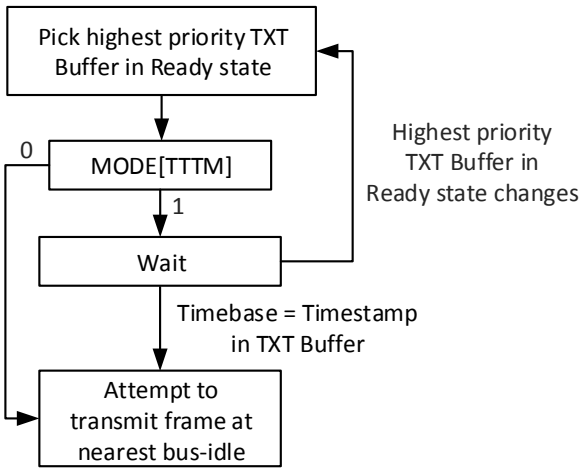


图 38.3-9. TX buffer 选择

38.3.8.2 定时发送模式

CAN FD 支持定时发送模式，对 TWAIFD_TTTM 写 1 可启用。在此模式下，仅当时间基准（参见章节 38.3.3）的值达到 TX buffer 中 TIMESTAMP_L_W 和 TIMESTAMP_U_W 所存储的时间戳时，CAN FD 才会尝试从优先级最高的 TX buffer 开始发送。时间基准是一个向上计数的无符号定时器，当其值达到设定的时间戳时，CAN FD 将发送位于 ready 状态下优先级最高 TX buffer 中的帧，如图 38.3-10 所示。请注意，这并不意味着立即发送该帧，CAN FD 仍需等待总线空闲。如果 TX buffer 处于 ready 状态，而时间基准尚未达到发送时间，CAN FD 会等待直到条件满足。如果在此期间总线上已有其他节点开始发送，CAN FD 会作为接收方参与通信。

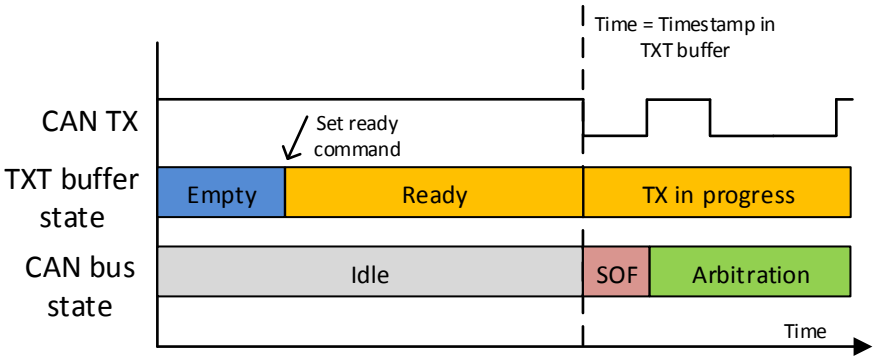


图 38.3-10. 定时发送

若某帧应立即发送（即不依赖时间触发机制），则应将 TIMESTAMP_L_W、TIMESTAMP_U_W 配置为 0x00000000。请注意，定时发送仅考虑 ready 状态下优先级最高的 TX buffer。TX buffer 优先级始终高于发送时间，具体行为如下：

- 如果 TX buffer A 的优先级高于 B，即使其发送时间晚于 B，CAN FD 也会优先考虑 A。
- 如果优先级发生变化，导致 ready 状态下优先级最高的 buffer 改变，则 CAN FD 将选择发送新的最高优先级 buffer 中的帧。只要此前选中的 buffer 仍在等待时间基准达到发送时间，这一原则有效。一旦开始发送，将不再考虑优先级，不会切换 buffer 帧。

38.3.8.3 发送 CAN 帧的类型

CAN FD 发送的 CAN 帧类型由 TX buffer 中 FRAME_FORMAT_W 字段和 CAN FD 的相关配置共同决定，如图 38.3-11 所示。

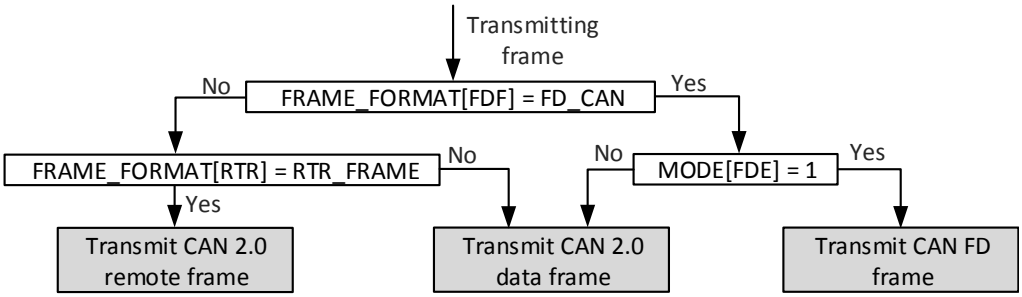


图 38.3-11. TX 帧类型

当 FRAME_FORMAT_W[FDF] = FD_CAN 且TWAIFD_FDE = 0 时，CAN FD 发送 CAN 2.0 帧。如果此时 TX buffer 中为超过 8 字节的数据负载（CAN FD 格式），则超出 8 字节的部分不会发送。

当 FRAME_FORMAT_W[RTR] = RTR_FRAME 且 FRAME_FORMAT_W[FDF] = FD_CAN 时，将忽略 RTR 标志，CAN FD 发送 CAN FD 帧（因为 CAN FD 协议不支持远程帧）。

38.3.8.4 重发限制

CAN FD 可以限制单个 TX buffer 的重发次数，对 TWAIFD_RTRLE 写 1 启。重发次数通过 TWAIFD_RTRTH 配置，首次尝试发送 CAN 帧不计为重发。配置选项如表 38.3-3 所示。

表 38.3-3. 重发限制配置

TWAIFD_RTRTH	TWAIFD_RTRTH	行为
-	0	发送尝试没有任何限制，直到发送成功或进入离线。
0	1	仅尝试发送一次，失败后不重发，即一次性模式。
1 ~ 15	1	尝试发送帧 TWAIFD_RTRTH + 1 次（首次发送 + TWAIFD_RTRTH 次重发）。

如果单个 TX buffer 连续重发 TWAIFD_RTRTH 次不成功（出现错误帧或仲裁丢失），则该 buffer 进入 TX failed 状态。如果两次发送之间 TX buffer 改变（例如优先级变化），内部重发计数器清零，新 TX buffer 的帧仍有 TWAIFD_RTRTH + 1 次发送机会。如果再次使用原 TX buffer 发送，则重新开始计数，CAN FD 将尝试发送 CAN 帧 TWAIFD_RTRTH + 1 次。若当前的 TX buffer 进入 aborted 状态，内部重发计数器也将清零。如果 buffer 再次进入 ready 状态，CAN FD 将尝试发送 TWAIFD_RTRTH + 1 次。每个帧的发送尝试次数由一个内部计数器追踪，通过 TWAIFD_RETR_CTR_VAL 读取。

38.3.8.5 中止

如果软件驱动程序先前通过 set ready 命令请求发送 CAN 帧，则可通过 set abort 命令中止该请求。若 TX buffer 接收 set abort 命令时仍处于 ready 状态（尚未开始发送），则会立即转为 aborted 状态。若 TX buffer 已处于 TX in progress 状态（已经开始发送），则转为 abort in progress 状态。在此情况下，一旦出现错误帧或仲裁丢失，将转换至 Aborted 状态。请注意，当 TX buffer 处于 abort in progress 状态时，如果当前发送成功，则进入 TX OK 状态；如果达到重发次数上限，则进入 TX failed 状态。

38.3.8.6 TX buffer 离线行为

当 CAN FD 由于 TEC > 255 而离线时，TX buffer 可根据配置做出以下两种响应：

1. 所有处于 ready、TX in progress 或 abort in progress 状态的 TX buffer 立即进入 TX failed 状态。对 TWAIFD_TBFBO 写 1 启用，为默认配置。
2. CAN FD 离线时使用的 TX buffer 将按照常规错误帧处理，通过清零 TWAIFD_TBFBO 启用。若未发出 set abort 命令且未达到重发上限，相关 buffer 会重新进入 ready 状态。在 CAN FD 重新接入总线（详见章节 38.3.4）后，将根据优先级规则继续尝试发送。此配置支持模块离线与重新接入的自动化，无需软件主动干预 TX buffer。

配置流程详见章节 38.5.1.1 *CAN 帧发送*。

38.3.9 CAN 帧接收

CAN FD 包含一个独立的 RX buffer，用于存储接收到的 CAN 帧。RX buffer 的大小是 32 位字的整数倍，通过 RX_MEM_INFO 读取。RX buffer 采用 FIFO 结构组织。当 CAN 总线上成功接收到一个 CAN 帧（未检测到错误帧）时，该 CAN 帧便存入 RX buffer。软件通过连续读取 TWAIFD_RX_DATA 寄存器，从 RX buffer 中取出 CAN 帧。每次从 TWAIFD_RX_DATA 寄存器读取，会从 RX buffer 读出一个字。RX buffer 支持以下两种操作模式：

- 自动模式 - 读取 TWAIFD_RX_DATA 时 RX FIFO 的读取指针自动递增，仅在通过 32 位访问方式读取 TWAIFD_RX_DATA 时使用。此时，对 TWAIFD_RXRPMV 写 1 无效。
- 手动模式 - 读取 TWAIFD_RX_DATA 时读取指针不会自动移动，需由软件对 TWAIFD_RXRPMV 写 1 手动增加。该模式适用于通过 8 位、16 位或 32 位访问方式读取 RX buffer，可借助多个小访问组合读取一个完整的字（如 4 次 8 位访问或 2 次 16 位访问）。

RX buffer 的模式由 TWAIFD_RXBAM 配置。RX buffer 内的 CAN 帧格式可参考图 38.3-12，该图展示了可能存储在 RX buffer 中的可选 CAN 帧格式（互斥）。RX buffer 中的每个 CAN 帧占用 4 到 20 个字，其大小计算公式为：

RX 帧的字数 = 4 + ceil (数据字段长度/4)

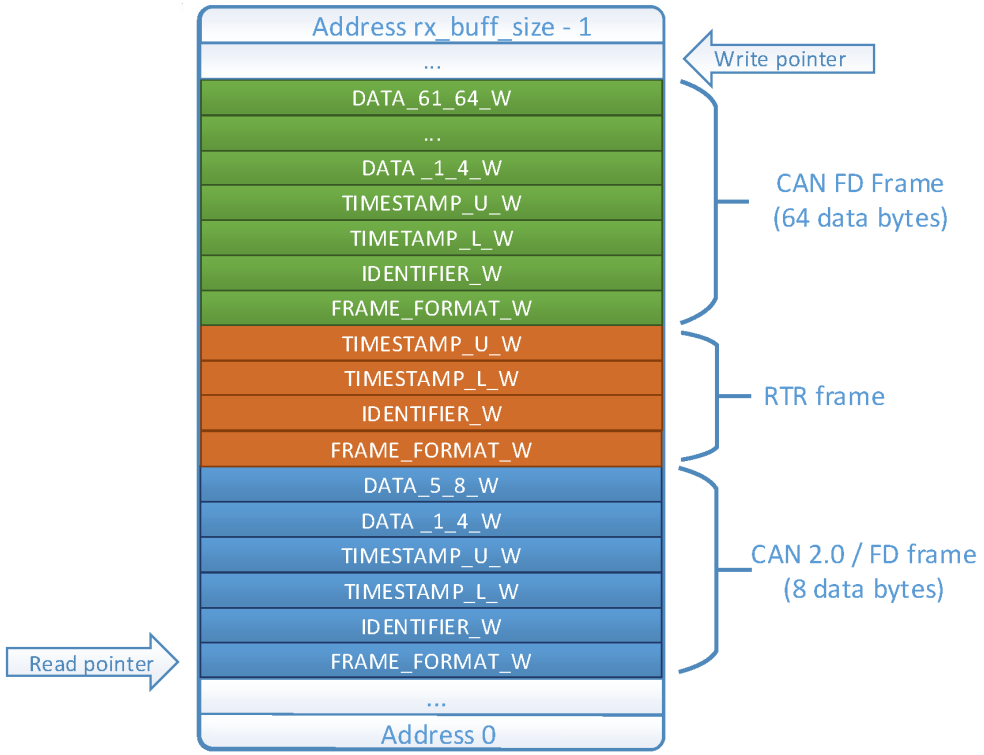


图 38.3-12. RX buffer

38.3.9.1 帧数量

RX buffer 内含一个帧计数器，统计 buffer 中存储的 CAN 帧数量，通过 TWAIFD_RXFRC 读取。帧存入 RX buffer 时，计数器递增；从 RX buffer 读取到某个 CAN 帧的最后一个字时，计数器递减。

38.3.9.2 RX buffer 内存

RX buffer 内存提供以下状态信息：

- 可用内存字数，通过 TWAIFD_RX_FREE 读取。
- 写指针位置，通过 TWAIFD_RX_WPP 读取。
- 读指针位置，通过 TWAIFD_RX_RPP 读取。

38.3.9.3 RX buffer 状态

没有存储任何 CAN 帧的 RX buffer 为空。在此状态下，TWAIFD_RXE = 1。所有字都被 CAN 帧占用的 RX buffer 为满。在此状态下，TWAIFD_RXF = 1。需要注意的是，即使 RX buffer 状态未满，例如还有 2 个空闲字，但可能此时最小的 CAN 帧也无法存入。

38.3.9.4 溢出

如果在接收 CAN 帧期间，RX buffer 没有足够的空闲空间存储当前接收的 CAN 帧，则会发生溢出，导致该帧被丢弃（RX buffer FIFO 溢出），同时溢出标志被置位。溢出标志为粘性标志（保持置位状态直至清除），可以通过 TWAIFD_DOR 读取，通过对 TWAIFD_CDO 写 1 清除。

38.3.9.5 刷新

对 TWAIFD_RRB 写 1 可以刷新 RX buffer。刷新将保留 buffer 中的内容（内存未被擦除），但会清零读写指针和帧计数器。此时，RX buffer 状态和未接收到任何帧相同。如果在 CAN 帧接收过程中执行此命令，当前正在接收的帧也会被丢弃。

38.3.9.6 不一致保护

从 RX buffer 读取 CAN 帧需要多次访问 TWAIFD_RX_DATA 寄存器，每次读取会产生副作用，即 RX buffer 内的读取指针自动递增。如果在读取过程中发生错误（如总线错误或 ECC 错误），则已读取的数据可能丢失，且读取指针已经递增，导致软件驱动无法重新读取该字，从而软件与 RX buffer 状态不一致。软件驱动程序可以读取 TWAIFD_RXMOF 来检测和恢复。当 RX buffer 的读取指针指向帧中间某个字（即下一次读取将返回除 FRAME_FORMAT_W 以外的字时），TWAIFD_RXMOF 会被置 1。如果软件驱动出现不一致，可反复读取 TWAIFD_RX_DATA，直至 TWAIFD_RXMOF 返回 0。此时，TWAIFD_RX_DATA 指针会定位到新帧的 FRAME_FORMAT_W 字，或者 RX buffer 为空（若在读取 RX buffer 中唯一帧时发生错误）。

38.3.9.7 时间戳

当 CAN FD 接收到 CAN 帧时，会将其时间戳存储在 RX buffer 中的 TIMESTAMP_L_W 和 TIMESTAMP_U_W 字，时间戳为采样外部时间基准的值。接收帧的时间戳可以在帧起始位采样点，或帧结束后的第 6 位（依据 ISO11898-1:2015 标准中定义的 CAN 帧有效接收时间）进行采样。时间戳采样位置由 TWAIFD_RTSOP 配置。

38.3.9.8 帧过滤

硬件过滤器用于筛选接收到的 CAN 帧。CAN FD 包含两种类型的过滤器：屏蔽过滤器和范围过滤器，共有三个屏蔽过滤器实例 (A、B、C) 和一个范围过滤器实例。接收到的 CAN 帧通过任一过滤器（即过滤器结果按逻辑“或”计算），才会存入 RX buffer。帧过滤器在启用接受过滤器模式 (TWAIFD_AFM = 1) 时生效，应用于接收到的帧标识符。TWAIFD_AFM 仅可在 CAN FD 禁用时修改 (TWAIFD_ENA = 0)。若禁用接受过滤器模式，过滤器功能无效，所有接收的 CAN 帧均会直接存入 RX buffer。

每个过滤器可选择配置为仅接受特定类型的 CAN 帧（CAN 2.0 或 CAN FD）及标识符类型（基本标识符或扩展标识符），通过 TWAIFD_FILTER_CONTROL 配置。如需禁用某过滤器，可将其对应的 TWAIFD_FILTER_CONTROL 寄存器全部字段清零。接受过滤器实例的内部工作机制如图 38.3-13 所示。

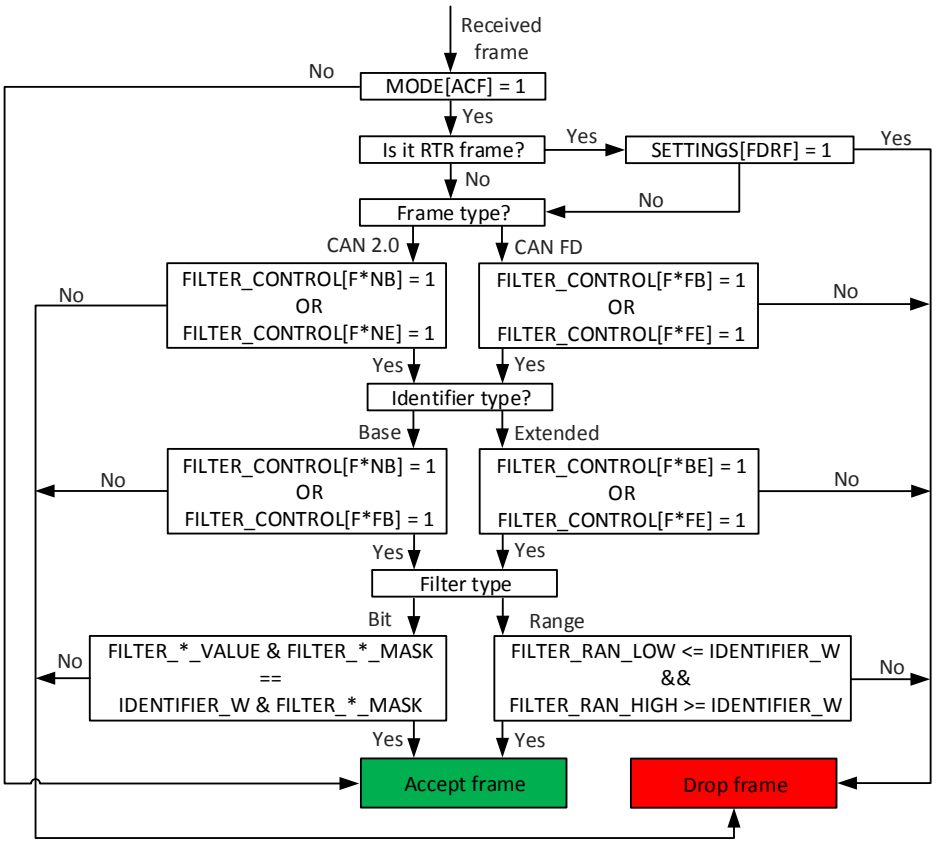


图 38.3-13. 帧过滤器工作流程（* 以 A/B/C/R 类型过滤器为例）

38.3.9.9 屏蔽过滤器

屏蔽过滤器会检查接收到的 CAN 帧标识符是否与 TWAIFD_FILTER_X_VAL_REG 配置的值匹配（X 表示过滤器实例 A、B 或 C），仅比较 TWAIFD_FILTER_X_MASK_REG 指定的屏蔽值。

若仅用屏蔽过滤器过滤带基本标识符的帧，应配置 TWAIFD_FILTER_X_MASK_REG[17:0] 为 0b00000000000000000。

38.3.9.10 范围过滤器

范围过滤器判断接收帧的标识符是否落在下限 TWAIFD_FILTER_RAN_LOW_REG 至上限 TWAIFD_FILTER_RAN_HIGH_REG 定义的十进制范围内。

若仅用屏蔽过滤器过滤带基本标识符的帧，应配置 TWAIFD_FILTER_RAN_LOW_REG[17:0] 为 0b00000000000000000 和 TWAIFD_FILTER_RAN_HIGH_REG[17:0] 为 0b11111111111111111。

配置流程详见章节 38.5.1.2 *CAN 帧接收*。

38.3.10 故障限制

CAN FD 的故障限制状态可以通过 TWAIFD_EWL_ERP_FAULT_STATE_REG 读取，状态转换见图 38.3-14。故障限制计数器包括 RX 错误计数器 (REC) 和 TX 错误计数器 (TEC)，二者符合 ISO11898-1 标准定义，计数器值可分别从 REC 和 TEC 读取。此外，CAN FD 提供区分标称位速率与数据位速率的错误计数器。标称位速率错误计数器 (ERR_NORM) 通过 TWAIFD_ERR_NORM_VAL 读取，每检测到一次错误计数加 1。数据位速率错误计数器 (ERR_FD) 通过 TWAIFD_ERR_FD_VAL 读取，每检测到一次错误计数加 1。

所有四个计数器（REC、TEC、ERR_NORM、ERR_FD）均可由软件控制。由于此功能直接影响 CAN FD 对 ISO11898-1 的合规性，因此仅在测试模式启用时 (TWAIFD_TSTM = 1) 时才可配置，所有计数器的值由 TWAIFD_CTR_PRES_REG 配置。错误警告限制 (EWL) 和被动错误限制 (ERP) 由 TWAIFD_EWL 和 TWAIFD_ERP_LIMIT 配置，默认值符合 ISO11898-1 标准。在测试模式下，EWL 和 ERP 字段支持写操作。

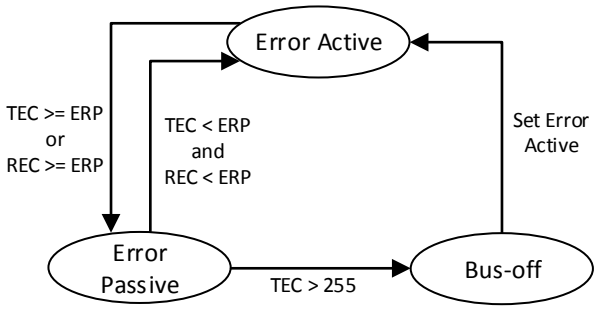


图 38.3-14. 故障限制

38.3.11 容错

CAN FD 实现了以下容错机制：

- RX buffer RAM 的奇偶校验保护
- TX buffer RAM 的奇偶校验保护
- TX buffer 备份模式 (TWAIFD_TXBBM)

运行上述机制必须满足以下条件：

- 软件驱动程序需对 TWAIFD_PCHKE 写 1 以启用奇偶校验错误检测。仅在 TWAIFD_ENA = 0 时可配置 TWAIFD_PCHKE。

38.3.11.1 RX buffer RAM 的奇偶校验保护

CAN FD 接收 CAN 帧并将其存储到 RX buffer RAM 时，会为 RX buffer RAM 的每个字添加一个奇偶校验位。软件驱动程序从 RX buffer RAM 读取帧时，可以通过读取 TWAIFD_RXPE 位检测奇偶校验错误。若 RX buffer RAM 中某字的奇偶校验位与计算结果不符，CAN FD 会在每次读取 TWAIFD_RX_DATA 寄存器时置位 TWAIFD_RXPE。

RX buffer RAM 中的伪单事件翻转 (SEU) 可能会修改 RX buffer RAM 中接收帧的 FRAME_FORMAT_W[DLC] 字，影响软件驱动程序看到的 RX 帧长度，进而引发 RX buffer 状态不一致。此时软件只读取了部分接收帧，后续从 RX buffer 读取的所有帧均被破坏。

38.3.11.2 TX buffer RAM 的奇偶校验保护

软件将 CAN 帧存储到 TX buffer 时，CAN FD 会为 TX buffer RAM 的每个字附加一个奇偶校验位。当 CAN FD 尝试从 TX buffer 发送帧并检测到 TX buffer RAM 中的奇偶校验错误时，其行为如下：

1. 如果 CAN FD 在 FRAME_FORMAT_W、IDENTIFIER_W、TIMESTAMP_U_W 或 TIMESTAMP_L_W 中检测到奇偶校验错误，则不会尝试发送该 CAN 帧。
2. 如果上述 TX buffer 字无奇偶校验错误，则尝试发送该 CAN 帧。

3. 如果在发送 CAN 帧时 CAN FD 在 DATA_1_4_W - DATA_61_64_W 的任意字检测到奇偶校验错误，则会发送错误帧。

当 CAN FD 在 TX buffer RAM 中检测到步骤 1 - 3 所述的奇偶校验错误时，该 TX buffer 进入 parity error 状态，如图 38.3-8 所示，并置位 TWAIFD_TXPE。

若检测到奇偶校验错误，软件应在再次尝试发送前，将整个 CAN 帧重新写入 TX buffer。

对 TWAIFD_CTXPE 写 1 可清除 TWAIFD_TXPE 位。

当 TWAIFD_PCHKE = 0 时，CAN FD 将忽略在 TX buffer 中的奇偶校验错误。此时不会配置 TWAIFD_TXPE，TWAIFD_CTXPE 无效，TX buffer 不会进入 parity error 状态。

CAN FD 不会检测 FRAME_TEST_W 中的奇偶校验错误。FRAME_TEST_W 用于故意破坏发送帧（例如测试 CAN 总线上的错误场景），该功能在需要奇偶校验保护的高可靠性容错应用中通常不适用。

38.3.11.3 TX buffer 备份模式

TWAIFD_TXBBM = 1 时，CAN FD 进入 TX buffer 备份模式。在该模式下，具有相邻索引的两个 TX buffer 组成一对，如图 38.3-15 所示。例如，TWAIFD_TXT_BUFFER_COUNT = 8 时，CAN FD 包含 8 个 TX buffer，共有 4 对 TX buffer，分别为 1-2、3-4、5-6、7-8。



图 38.3-15. TX buffer 对

备份模式通过备份 TX buffer 为原 TX buffer 提供容错支持，buffer 对中索引较高的 buffer 作为备份，用于在原 buffer 发生奇偶校验错误时替代发送。工作流程如图 38.3-16 所示。

当 TWAIFD_TXBBM = 1 且原 TX buffer 存在奇偶校验错误时，该 TX buffer 会进入 parity error 状态，CAN FD 将尝试从对应的备份 TX buffer 发送帧（如 TX buffer 3 出错时，尝试从 TX buffer 4 发送）。若原 TX buffer 发送成功，其备份 TX buffer 则进入 aborted 状态，CAN FD 不会发送备份帧。

当 CAN FD 由于原 TX buffer 中的奇偶校验错误而从备份 TX buffer 发送帧时，如果在备份 TX buffer RAM 中也检测到奇偶校验错误，则会置位 TWAIFD_TXDPE，表示存在双重奇偶校验错误。

在备份模式下，软件对 TX buffer 的控制与非备份模式存在差异：

- TX buffer 对中的两个 TX buffer 的优先级相同，由原始 TX buffer 的 TWAIFD_TX_PRIORITY_REG[TX*P] 决定。例如，TX buffer 1 和 2 的优先级由 TWAIFD_TX_PRIORITY_REG[TX1P] 决定，TWAIFD_TX_PRIORITY_REG[TX2P] 无效。
- 软件发出的命令将自动应用于每个原 TX buffer 及其对应的备份 TX buffer。例如，如果软件对 TX buffer 1 发出命令（TX_COMMAND[TXB1] = 1），CAN FD 会自动将其应用于 TX buffer 2。

软件存储相同的帧至 TX buffer 对中的两个 buffer，备份模式可在原 buffer 出现奇偶校验错误时，从备份 buffer 发送相应的帧。

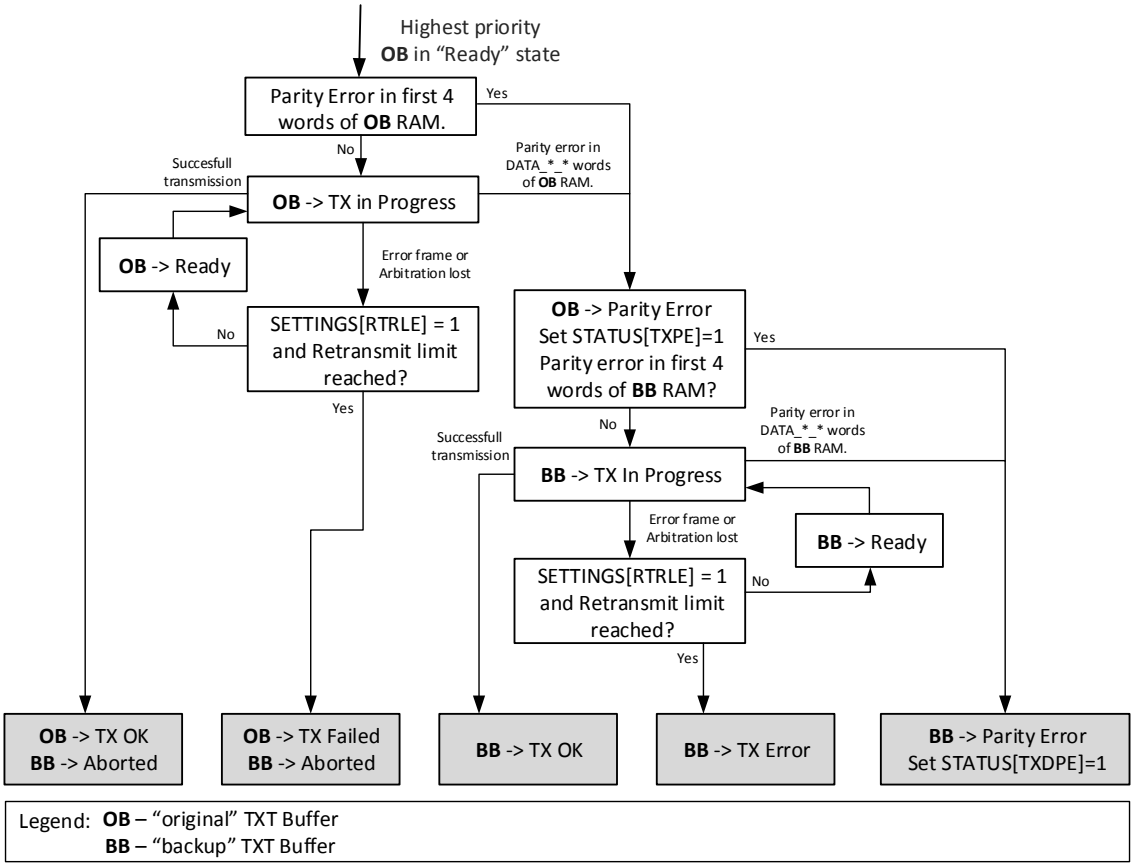


图 38.3-16. TX buffer 备份模式工作流程

通过单独的内存访问将相同的帧存储到两个 TX buffer 是设计使然。CAN FD 不会自动将帧复制到两个 TX buffer，以避免在写入过程中发生 SEU。如果发生 SEU，可能导致两个 TX buffer 中存储的帧已包含奇偶校验错误。

不必将完全相同的帧存储到 TX buffer 对中的两个 buffer。软件也可以仅将任意一帧存入备份 TX buffer，以便在原 TX buffer 出现奇偶校验错误时使用该备份帧进行发送。

软件应同时将 TWAIFD_TXBBM 和 TWAIFD_PCHKE 置 1。如果 TWAIFD_TXBBM = 1，TWAIFD_PCHKE = 0，CAN FD 会忽略原 TX buffer 中的奇偶校验错误，且不会从备份 TX buffer 发送帧。

在 CAN 帧发送期间，如果 CAN FD 在原 TX buffer 中检测到奇偶校验错误，且存在另一个优先级高于当前所用 TX buffer 对的 TX buffer 已因软件发出 set ready 命令而进入 ready 状态），CAN FD 将在下一次发送时优先选择优先级更高的 TX buffer，忽略备份 TX buffer。

CAN FD 仅在包含偶数个 TX buffer 时支持 TX buffer 备份模式。如果 TX buffer 数量为奇数，则存在一个没有备份 buffer 的 TX buffer。在此情况下，当 TWAIFD_TXBBM = 1 时，软件不得使用这一无备份的原 TX buffer，否则 CAN FD 行为将不可预测。

38.3.12 特殊模式

38.3.12.1 回环模式

回环模式下，CAN FD 从任意 TX buffer 发送 CAN 帧后，若帧发送成功且通过帧过滤器，则将存储至 RX buffer。需注意，尽管 CAN FD 可将自身发送的帧存入 RX buffer，但其仍作为发送端，并不会确认自身的帧。要成功发

送帧，须由 CAN 总线上的其他节点完成确认。回环模式通过对 TWAIFD_ILBP 写 1 启用，仅在 CAN FD 禁用 (TWAIFD_ENA = 0) 时可以修改。

38.3.12.2 自检模式

自检模式下，即使确认槽未收到显性位，CAN FD 仍判定发送帧为有效。该模式可与回环模式同时启用，用于验证 CAN FD 作为单节点在总线上的运行情况。自检模式通过对 TWAIFD_STM 写 1 启用，仅在 CAN FD 禁用 (TWAIFD_ENA = 0) 时可以修改。

38.3.12.3 禁止确认模式

禁止确认模式启用后，CAN FD 作为 CAN 帧接收端时，即使接收到的 CRC 校验无误，也不会向确认槽发送显性位。禁止确认模式通过向 TWAIFD_ACF 写 1 启用，仅在 CAN FD 禁用 (TWAIFD_ENA = 0) 时可以修改。

38.3.12.4 总线监控模式

总线监控模式下，CAN FD 只接收、不发送任何帧。将 CAN 帧写入 TX buffer 并发出 set ready 命令时，不会发送该帧，且 TX buffer 会立即进入 TX failed 状态。在此模式下，CAN FD 不在总线上发出任何显性位；当需发出显性位（如 ACK 或错误帧）时，显性位仅作为内部信号由 CAN FD 接收，CAN 总线上其他节点无法检测到该字段。总线监控模式通过向 TWAIFD_BMM 写 1 启用，仅在 CAN FD 禁用 (TWAIFD_ENA = 0) 时可以修改。

38.3.12.5 受限操作模式

受限操作模式下，CAN FD 能够接收 CAN 总线上的帧，但不发送任何帧。将 CAN 帧写入 TX buffer 并发出 set ready 命令时，不会发送该帧，且 TX buffer 会立即进入 TX failed 状态。在此模式下，CAN FD 会确认有效帧，但不发送错误帧或过载帧。当检测到错误或过载时，CAN FD 进入总线接入状态，并等待 11 个连续的隐性位。REC 与 TEC 计数器均不会被修改，CAN FD 始终处于主动错误状态。受限操作模式通过向 TWAIFD_ROM 写 1 启用，仅在 CAN FD 禁用 (TWAIFD_ENA = 0) 时可以修改。

38.3.13 其他功能

38.3.13.1 错误代码捕获

CAN FD 配备了错误代码捕获寄存器，用于记录导致发送错误帧最后一次错误的类型和比特位置。检测到错误信号时，其采样点会实时更新该寄存器。用户可通过 TWAIFD_ERR_CAPT 读取错误捕获信息。需要注意，CAN FD 标准未将错误类型定义为互斥（例如：某个填充位损坏时可能同时满足位错误和填充错误条件）。在这种情况下，错误代码捕获机制仅记录优先级最高的错误。错误类型的优先级如下，数字越小优先级越高（格式错误最高）：

优先级	1	2	3	4	5
错误类型	格式错误	位错误	CRC 错误	ACK 错误	填充错误

填充 CAN FD 帧固定位时，发生的填充错误会在错误代码捕获寄存器中报告为格式错误。

上述错误优先级顺序有一个例外。如果在仲裁字段期间发送显性填充位时采样到隐性值，则此错误被捕获为填充错误，而不是位错误。

38.3.13.2 仲裁丢失捕获

CAN FD 包含仲裁丢失捕获寄存器 (ALC)，用于记录 CAN FD 最近一次在 CAN 仲裁字段丢失仲裁的比特位置。

38.3.13.3 流量计数器

CAN FD 支持统计在 CAN 总线上发送和接收的 CAN 帧。每成功发送一个 CAN 帧，TX_COUNTER 加 1。每成功接收一个 CAN 帧，RX_COUNTER 加 1。通过向 TWAIFD_TXFCRST写 1 可清除 TX_COUNTER，通过向 TWAIFD_RXFCRST 写 1 可清除 RX_COUNTER。当 CAN FD 处于回环模式并且自身发送的帧存入 RX buffer 时，RX_COUNTER 也会增加。流量计数器为 CAN FD 的可选功能，可通过读取 TWAIFD_STCNT 来判断其是否使能。

38.3.13.4 调试寄存器

CAN FD 包含调试寄存器 TWAIFD_DEBUG_REG，直接反映当前正在发送或接收的 CAN 帧的相关字段内容。

38.4 中断

ESP32-C5 的 CAN FD 可以生成以下中断信号并发送给 中断矩阵。

- IRQ
- IRQ_TIMER

以下中断源可以中断源可以生成上述中断信号，具体取决于中断源：

- TWAIFD_TXBHCI_INT：当 TX buffer 收到来自 CAN 核心的硬件命令，将 TX buffer 状态更改为 TX OK、error 或 aborted 时触发。
- TWAIFD_RBNEI_INT：当 RX buffer 非空时触发。清除此中断且不通过 TWAIFD_RX_DATA 读取 RX buffer 将重新激活中断。
- TWAIFD_BSI_INT：当位速率发生变化时触发。
- TWAIFD_RXFI_INT：当 RX buffer 为满时触发。
- TWAIFD_OFI_INT：当开始发送过载帧时触发。
- TWAIFD_BEI_INT：当开始发送错误帧时触发。
- TWAIFD_ALI_INT：当仲裁丢失时触发。
- TWAIFD_FCSI_INT：当故障限制状态改变时触发，即节点状态从主动错误变为被动错误，从被动错误变为离线，从离线后重新接入或被动错误变为主动错误。
- TWAIFD_DOI_INT：当数据溢出时触发。在清除此中断之前，必须清除 TWAIFD_DOI_INT_ST 以避免再次触发此中断。
- TWAIFD_EWLI_INT：当 TX 错误计数器 (TEC) 或 RX 错误计数器 (REC) 达到错误警告限制 (EWL) 时触发。当中断被清除且 REC 或 TEC 仍等于或高于 EWL 时，不会再次产生中断。
- TWAIFD_TXI_INT：当发送帧时触发。
- TWAIFD_RXI_INT：当接收帧时触发。
- TWAIFD_TIMER_OVERFLOW_INT：当定时器溢出时触发。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 38.6 寄存器列表。

38.5 配置示例

38.5.1 500 kbit/4 Mbit 示例

在汽车行业中，CAN 总线常见的配置为：标称位速率 500 kbit/s，数据位速率 4 Mbit/s。对于 GPIO 选择无特定要求，收发器需选用支持 CAN FD 的型号（如 TD301MCANFD）。以下伪代码展示了在系统时钟源为 PLL_F80M_CLK 且采样点设为每个位的 80% 时的典型配置方法：

```
uint32 btr;
btr = (4 << 19);     // BRP = 4 (Tq = 4/80MHz = 50ns)
btr |= 14;           // PROP = 14
btr |= (17 << 7);    // SEG1 = 17
btr |= (8 << 13);   // SEG2 = 8
btr |= (2 << 27);    // SJW = 2
REG32_WR(TWAIFD_BTR_REG, btr); // (1+14+17+8)*50ns = 40*50ns = 2μs = 500kbit/s
uint32 btr_fd;
btr_fd = (1 << 19);     // BRP = 1 (Tq = 1/80MHz = 12.5ns)
btr_fd |= 3;           // PROP = 3
btr_fd |= (11 << 7);    // SEG1 = 11
btr_fd |= (5 << 13);   // SEG2 = 5
btr_fd |= (2 << 27);    // SJW = 2
REG32_WR(TWAIFD_BTR_FD_REG, btr_fd); // (1+3+11+5)*12.5ns = 20*12.5ns = 250ns = 4Mbit
    /s
```

按上述伪代码配置寄存器 TWAIFD_BTR_REG 和 TWAIFD_BTR_FD_REG。如收发器选用 TD301MCANFD，建议配置 TWAIFD_SSP_OFFSET 为 14。

38.5.1.1 CAN 帧发送

38.5.1.1.1 示例代码

```
#define CAN_FD_BASE TWAIFD_DEVICE_ID_VERSION_REG
#define TX_COMMAND_ADDR (CAN_FD_BASE + 0x74)
#define TXT_BUFFER_1_BASE (CAN_FD_BASE + 0x100)
/* 将 CAN 帧插入 TX buffer  1 */
uint32_t frame_format_word = 0;
frame_format_word |= 4;                                    // DLC = 4
frame_format_word |= (1 << 7);                             // CAN FD 帧
frame_format_word |= (1 << 9);                             // 切换位速率
REG32_WR(TXT_BUFFER_1_BASE, frame_format_word);            // 存储帧格式字
uint32_t id_word = (55 << 18);                             // 标识符：55
REG32_WR(TXT_BUFFER_1_BASE + 0x4, id_word);                // 存储标识符字
REG32_WR(TXT_BUFFER_1_BASE + 0x8, 1000);
REG32_WR(TXT_BUFFER_1_BASE + 0xC, 0);                      // 在时间 1000 发送
REG32_WR(TXT_BUFFER_1_BASE + 0x10, 0xAABBCCDD);            // 数据：0xAA 0xBB 0xCC 0xDD
/* 发出 set ready 命令 */
uint32_t command = 0;
command |= 0x2;                                            // set ready 命令
command |= (1 << 8);                                       // 选择 TX buffer 1
REG32_WR(TX_COMMAND_ADDR, command);                        // 发出命令
```

向 TWAIFD_ENA 写 1 使能 CAN FD 时，在接入 CAN 总线期间模块仍保持离线状态。如果此时向 TX buffer 下发 set ready 命令，并且 TWAIFD_TBFBO = 1，TX buffer 会立即进入 aborted 状态。在向任意 TX buffer 发送 set ready 命令之前，软件应等待节点进入错误主动状态，可通过轮询 TWAIFD_EWL_ERP_FAULT_STATE_REG 或 FCS 中断判断。

TX buffer 未初始化或重置。因此，在发出 set ready 命令之前，软件必须先将有效的 CAN 帧填充至相应的 TX buffer。

CAN FD 仅在检测到总线过载时主动发送过载帧，不会因内部原因自行发送过载帧。

38.5.1.2 CAN 帧接收

38.5.1.2.1 示例代码 - 自动模式下的帧接收（32 位访问方式）

```
#define CAN_FD_BASE TWAIFD_DEVICE_ID_VERSION_REG
#define RX_DATA_ADDR (CAN_FD_BASE + 0x6C)
#define RX_STATUS_ADDR (CAN_FD_BASE + 0x68)
/* 轮询 RX buffer 直到其中有帧 */
uint32_t rx_status;
do {
    rx_status = REG32_RD(RX_STATUS_ADDR);
} while ((rx_status \& 0x1) == 0)
/* 从 RX buffer 读取帧 */
uint8_t data[64];
uint32_t tmp;
uint32_t ffw = REG32_RD(RX_DATA_ADDR);
uint32_t id = REG32_RD(RX_DATA_ADDR);
uint32_t ts_l = REG32_RD(RX_DATA_ADDR);
uint32_t ts_h = REG32_RD(RX_DATA_ADDR);
uint32_t rwcnt = (ffw >> 11) \& 0x1F;
for(int i = 0; i < rwcnt; i++){
    tmp = REG32_RD(RX_DATA_ADDR);
    data[i*4] = tmp \& 0xFF;
    data[i*4+1] = (tmp >> 8) \& 0xFF;
    data[i*4+2] = (tmp >> 16) \& 0xFF;
    data[i*4+3] = (tmp >> 24) \& 0xFF;
}
```

38.6 寄存器列表

本小节的所有地址均为相对于 CAN FD 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
ID 寄存器			
TWAIFD_DEVICE_ID_VERSION_REG	设备 ID 状态寄存器	0x0000	RO
配置寄存器			
TWAIFD_MODE_SETTINGS_REG	模式设置寄存器	0x0004	varies
TWAIFD_COMMAND_REG	命令寄存器	0x000C	WO
TWAIFD_BTR_REG	位时间寄存器	0x0024	R/W
TWAIFD_BTR_FD_REG	数据段的位时间寄存器	0x0028	R/W
TWAIFD_TRV_DELAY_SSP_CFG_REG	发送延迟和二次采样点配置寄存器	0x0080	varies
TWAIFD_TIMER_CLK_EN_REG	定时器时钟强制使能寄存器	0x0FD4	R/W
TWAIFD_TIMER_CFG_REG	定时器配置寄存器	0x0FE8	varies
TWAIFD_TIMER_LD_VAL_L_REG	定时器初始值寄存器	0x0FEC	R/W
TWAIFD_TIMER_CT_VAL_L_REG	定时器终到值寄存器	0x0FF4	R/W
状态寄存器			
TWAIFD_STATUS_REG	状态寄存器	0x0008	RO
TWAIFD_RX_MEM_INFO_REG	RX 内存信息寄存器	0x0060	RO
TWAIFD_RX_POINTERS_REG	RX 内存指针信息寄存器	0x0064	RO
TWAIFD_RX_STATUS_RX_SETTINGS_REG	RX 状态和设置寄存器	0x0068	varies
TWAIFD_TX_STATUS_REG	TX buffer 状态寄存器	0x0070	RO
TWAIFD_ERR_CAPT_RETR_CTR_ALC_TS_INFO_REG	错误捕获、重发计数器、仲裁丢失、时间戳信息寄存器	0x007C	RO
TWAIFD_TIMESTAMP_LOW_REG	时间基准低位寄存器	0x0094	RO
TWAIFD_TIMESTAMP_HIGH_REG	时间基准高位寄存器	0x0098	RO
中断寄存器			
TWAIFD_INT_STAT_REG	屏蔽中断状态寄存器	0x0010	R/W1C
TWAIFD_INT_ENA_SET_REG	中断使能寄存器	0x0014	R/W1S
TWAIFD_INT_ENA_CLR_REG	中断清除寄存器	0x0018	WO
TWAIFD_INT_MASK_SET_REG	屏蔽中断状态寄存器	0x001C	R/W1S
TWAIFD_INT_MASK_CLR_REG	屏蔽中断清除寄存器	0x0020	WO
TWAIFD_TIMER_INT_RAW_REG	定时器原始中断状态寄存器	0x0FD8	R/ SS/ WTC
TWAIFD_TIMER_INT_ST_REG	定时器屏蔽中断状态寄存器	0x0FDC	RO
TWAIFD_TIMER_INT_ENA_REG	定时器中断使能寄存器	0x0FE0	R/W
TWAIFD_TIMER_INT_CLR_REG	定时器中断清除寄存器	0x0FE4	WT
错误限制寄存器			
TWAIFD_EWL_ERP_FAULT_STATE_REG	错误阈值和状态寄存器	0x002C	varies
TWAIFD_REC_TEC_REG	错误计数器状态寄存器	0x0030	RO

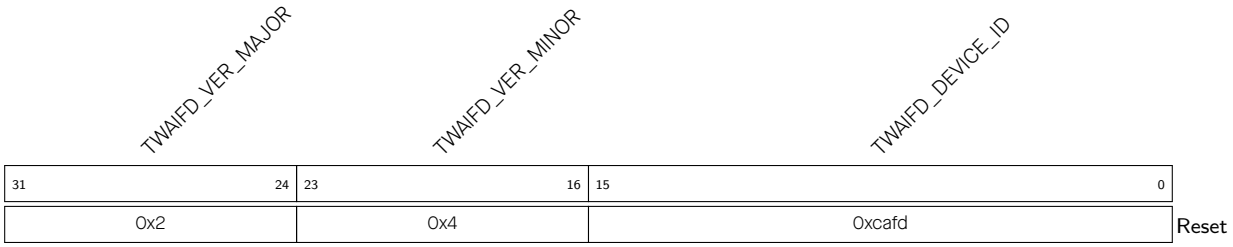
名称	描述	地址	访问
TWAIFD_ERR_NORM_ERR_FD_REG	特殊错误计数器状态寄存器	0x0034	RO
TWAIFD_CTR_PRES_REG	错误计数器预定义配置寄存器	0x0038	varies
过滤器寄存器			
TWAIFD_FILTER_A_MASK_REG	过滤器 A 屏蔽值寄存器	0x003C	R/W
TWAIFD_FILTER_A_VAL_REG	过滤器 A 位值寄存器	0x0040	R/W
TWAIFD_FILTER_B_MASK_REG	过滤器 B 屏蔽值寄存器	0x0044	R/W
TWAIFD_FILTER_B_VAL_REG	过滤器 B 位值寄存器	0x0048	R/W
TWAIFD_FILTER_C_MASK_REG	过滤器 C 屏蔽值寄存器	0x004C	R/W
TWAIFD_FILTER_C_VAL_REG	过滤器 C 位值寄存器	0x0050	R/W
TWAIFD_FILTER_RAN_LOW_REG	过滤器范围阈值下限寄存器	0x0054	R/W
TWAIFD_FILTER_RAN_HIGH_REG	过滤器范围阈值上限寄存器	0x0058	R/W
TWAIFD_FILTER_CONTROL_FILTER_STATUS_REG	过滤器控制寄存器	0x005C	varies
接收寄存器			
TWAIFD_RX_DATA_REG	接收数据寄存器	0x006C	RO
发送寄存器			
TWAIFD_TX_COMMAND_TXTB_INFO_REG	TX buffer 命令和信息寄存器	0x0074	varies
TWAIFD_TX_PRIORITY_REG	TX buffer 优先级寄存器	0x0078	R/W
控制器寄存器			
TWAIFD_RX_FR_CTR_REG	接收帧计数寄存器	0x0084	RO
TWAIFD_TX_FR_CTR_REG	发送帧计数寄存器	0x0088	RO
TWAIFD_DEBUG_REG	调试寄存器	0x008C	RO
版本寄存器			
TWAIFD_DATE_VER_REG	版本控制寄存器	0x0FFC	R/W

38.7 寄存器

本小节的所有地址均为相对于 CAN FD 基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 38.1. TWAIFD_DEVICE_ID_VERSION_REG (0x0000)

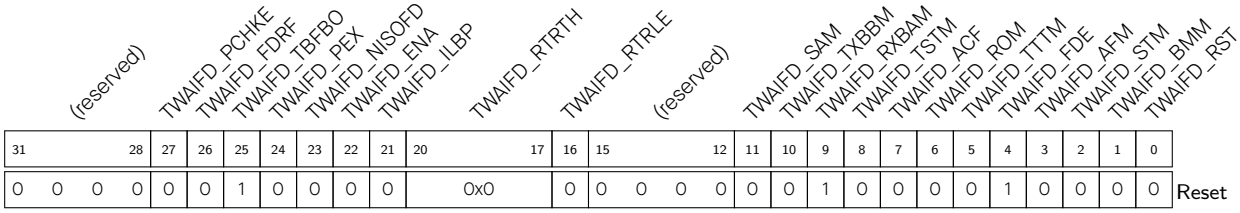


TWAIFD_DEVICE_ID 表示 CAN IP 功能是否正确映射到其基地址。(RO)

TWAIFD_VER_MINOR 表示 TWAI FD IP 小版本号。(RO)

TWAIFD_VER_MAJOR 表示 TWAI FD IP 大版本号。(RO)

Register 38.2. TWAIFD_MODE_SETTINGS_REG (0x0004)



TWAIFD_RST 配置是否软件复位。写 1 会复位 CAN FD。写 1 后不需要再写 0，该字段会自动清除。

0：无效

1：复位

(WO)

TWAIFD_BMM 配置是否启用总线监控模式。

0：禁用

1：启用

(R/W)

TWAIFD_STM 配置是否启用自检模式。

0：禁用

1：启用

(R/W)

TWAIFD_AFM 配置是否启用接受过滤器模式。

0：禁用

1：启用

(R/W)

TWAIFD_FDE 配置是否启用灵活数据速率。启用后，CAN FD 能够识别 CAN FD 帧（即 FDF 位为 1）。

0：禁用

1：启用

(R/W)

TWAIFD_TTTM 配置是否启用定时发送模式。

0：禁用

1：启用

(R/W)

TWAIFD_ROM 配置是否启用受限操作模式。

0：禁用

1：启用

(R/W)

TWAIFD_ACF 配置是否启用禁止确认模式。

0：禁用

1：启用

(R/W)

见下页...

Register 38.2. TWAIFD_MODE_SETTINGS_REG (0x0004)

... 接上页

TWAIFD_TSTM 配置是否启用测试模式。

0：禁用

1：启用

(R/W)

TWAIFD_RXBAM 配置是否启用 RX buffer 自动模式。

0：禁用

1：启用

(R/W)

TWAIFD_TXBBM 配置是否启用 TX buffer 备份模式。

0：禁用

1：启用

(R/W)

TWAIFD_SAM 配置是否启用自主确认模式。

0：禁用

1：启用

(R/W)

TWAIFD_RTRLE 配置是否启用重发限制。

0：禁用

1：启用

(R/W)

TWAIFD_RTRTH 配置重发限制阈值。TWAIFD_RTRLE 启用时，最大重发次数。(R/W)

TWAIFD_ILBP 配置是否启用回环模式。

0：禁用

1：启用

(R/W)

TWAIFD_ENA 配置是否启用 CAN FD。

0：禁用

1：启用

(R/W)

TWAIFD_NISOFD 配置 CAN FD 是否符合 ISO 协议。仅在 TWAIFD_ENA = 0 时修改。

0：符合 ISO

1：符合 CAN FD 1.0

(R/W)

TWAIFD_PEX 配置是否启用协议异常处理。仅在 TWAIFD_ENA = 0 时修改。

0：禁用

1：启用

(R/W)

见下页...

Register 38.2. TWAIFD_MODE_SETTINGS_REG (0x0004)

... 接上页

TWAIFD_TBFBO 配置 CAN FD 离线时，所有 TX buffer 是否进入 TX failed 状态。

0：不进入 TX failed

1：进入 TX failed

(R/W)

TWAIFD_FDRF 配置帧过滤器是否丢弃 RTR 帧。

0：接受

1：丢弃

(R/W)

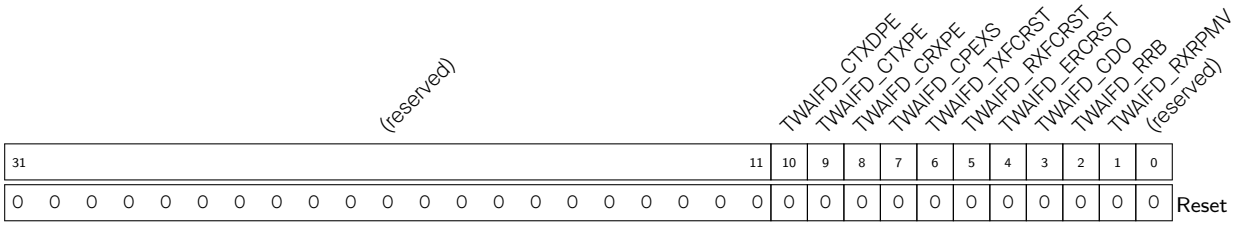
TWAIFD_PCHKE 配置是否启用 TX buffer 和 RX buffer 的奇偶校验检查。

0：禁用

1：启用

(R/W)

Register 38.3. TWAIFD_COMMAND_REG (0x000C)



TWAIFD_RXRPMV 配置是否将 RX buffer 读取指针向前移动。

0：无操作

1：向前移动

(WO)

TWAIFD_RRB 配置是否刷新 RX buffer 并重置其内存指针。

0：无效

1：刷新

(WO)

TWAIFD_CDO 配置是否清除 RX buffer 的数据溢出标志。

0：无效

1：清除

(WO)

TWAIFD_ERCRST 配置是否复位错误计数器。若 CAN FD 不处于离线状态，或因禁用（TWAIFD_ENA = 0）导致离线，此命令无效。

0：无效

1：复位

(WO)

TWAIFD_RXFCRST 配置是否清除 RX 总线流量计数器 RX_COUNTER。

0：无效

1：清除

(WO)

TWAIFD_TXFCRST 配置是否清除 TX 总线流量计数器 TX_COUNTER。

0：无效

1：清除

(WO)

TWAIFD_CPEXS 配置是否清除协议异常标志。

0：无效

1：清除

(WO)

见下页...

Register 38.3. TWAIFD_COMMAND_REG (0x000C)

... 接上页

TWAIFD_CRXPE 配置是否清除 TWAIFD_RXPE 标志。

0：无效

1：清除

(WO)

TWAIFD_CTXPE 配置是否清除 TWAIFD_TXPE 标志。

0：无效

1：清除

(WO)

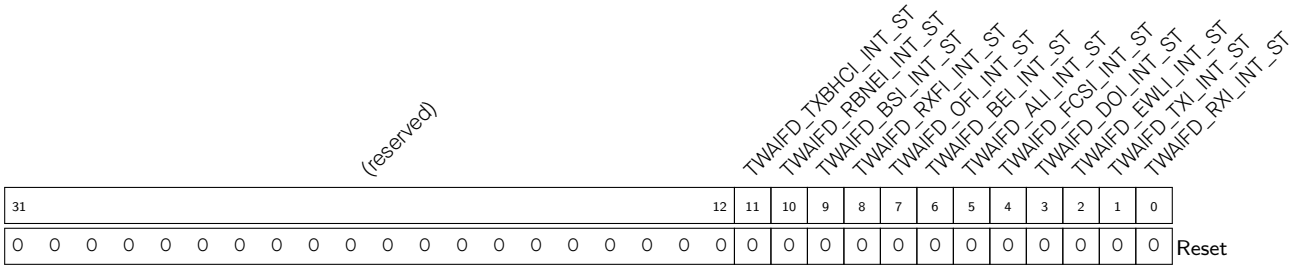
TWAIFD_CTXDPE 配置是否清除双重奇偶校验错误标志。

0：无效

1：清除

(WO)

Register 38.4. TWAIFD_INT_STAT_REG (0x0010)



TWAIFD_RXI_INT_ST TWAIFD_RXI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_TXI_INT_ST TWAIFD_TXI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_EWLI_INT_ST TWAIFD_EWLI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_DOI_INT_ST TWAIFD_DOI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_FCSI_INT_ST TWAIFD_FCSI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_ALI_INT_ST TWAIFD_ALI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_BEI_INT_ST TWAIFD_BEI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_OFI_INT_ST TWAIFD_OFI_INT 的屏蔽中断状态。(R/W1C)

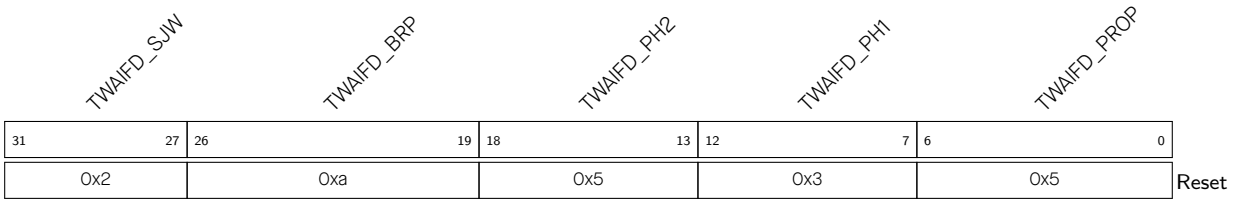
TWAIFD_RXFI_INT_ST TWAIFD_RXFI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_BSI_INT_ST TWAIFD_BSI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_RBNEI_INT_ST TWAIFD_RBNEI_INT 的屏蔽中断状态。(R/W1C)

TWAIFD_TXBHCI_INT_ST TWAIFD_TXBHCI_INT 的屏蔽中断状态。(R/W1C)

Register 38.5. TWAIFD_BTR_REG (0x0024)



TWAIFD_PROP 配置标称位速率的传播段。

测量单位：时间定额。(R/W)

TWAIFD_PH1 配置标称位速率的相位 1 段。

测量单位：时间定额。(R/W)

TWAIFD_PH2 配置标称位速率的相位 2 段。

测量单位：时间定额。(R/W)

TWAIFD_BRP 配置标称位速率的波特率预分频器。

测量单位：系统时钟周期。(R/W)

TWAIFD_SJW 配置标称位时间内的同步跳宽。

测量单位：时间定额。(R/W)

Register 38.6. TWAIFD_BTR_FD_REG (0x0028)



TWAIFD_PROP_FD 配置数据位速率的传播段。

测量单位：时间定额。(R/W)

TWAIFD_PH1_FD 配置数据位速率的相位 1 段。

测量单位：时间定额。(R/W)

TWAIFD_PH2_FD 配置数据位速率的相位 2 段。

测量单位：时间定额。(R/W)

TWAIFD_BRP_FD 配置数据位速率的波特率预分频器。

测量单位：核心时钟周期。(R/W)

TWAIFD_SJW_FD 配置数据位时间内的同步跳宽。

测量单位：时间定额。(R/W)

Register 38.7. TWAIFD_TRV_DELAY_SSP_CFG_REG (0x0080)



TWAIFD_TRV_DELAY_VALUE 表示以最小时间定额的倍数测量的发送器延迟。(RO)

TWAIFD_SSP_OFFSET 表示以最小时间定额的倍数测量的二次采样点偏移。(R/W)

TWAIFD_SSP_SRC 表示二次采样点来源。

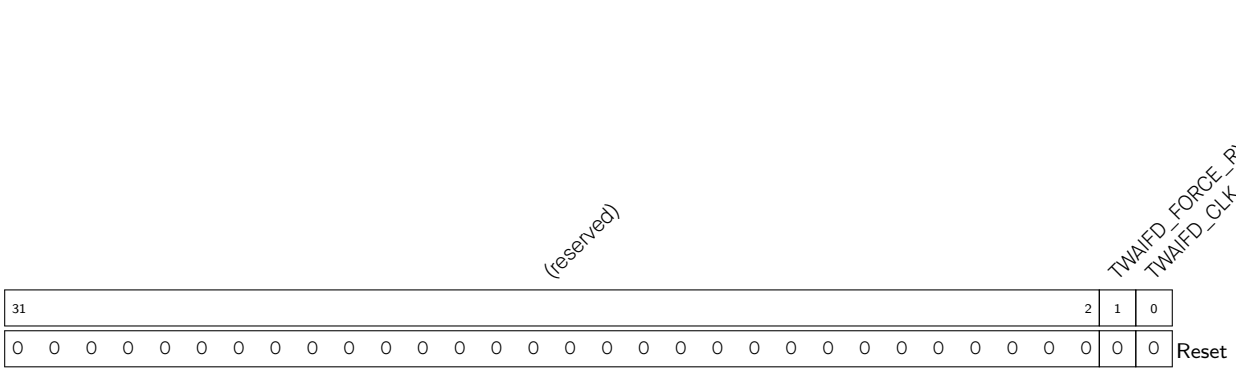
0：SSP_SRC_MEAS_N_OFFSET，SSP 位置 = TRV_DELAY（测量的发送器延迟）+ SSP_OFFSET。

1：SSP_SRC_NO_SSP，不使用 SSP。发送器在数据位速率阶段使用常规采样点。

2：SSP_SRC_OFFSET，SSP 位置 = SSP_OFFSET。忽略测量的发送器延迟。

(R/W)

Register 38.8. TWAIFD_TIMER_CLK_EN_REG (0x0FD4)



TWAIFD_CLK_EN 配置是否强制启用寄存器配置时钟信号。

0：禁用

1：启用

(R/W)

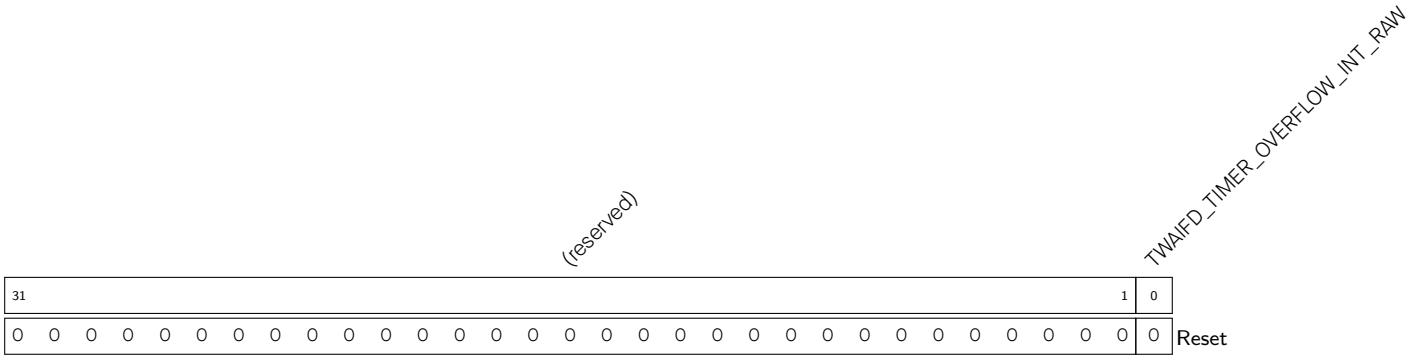
TWAIFD_FORCE_RXBUF_MEM_CLK_ON 配置是否强制启用 RX buffer RAM 时钟信号。

0：禁用

1：启用

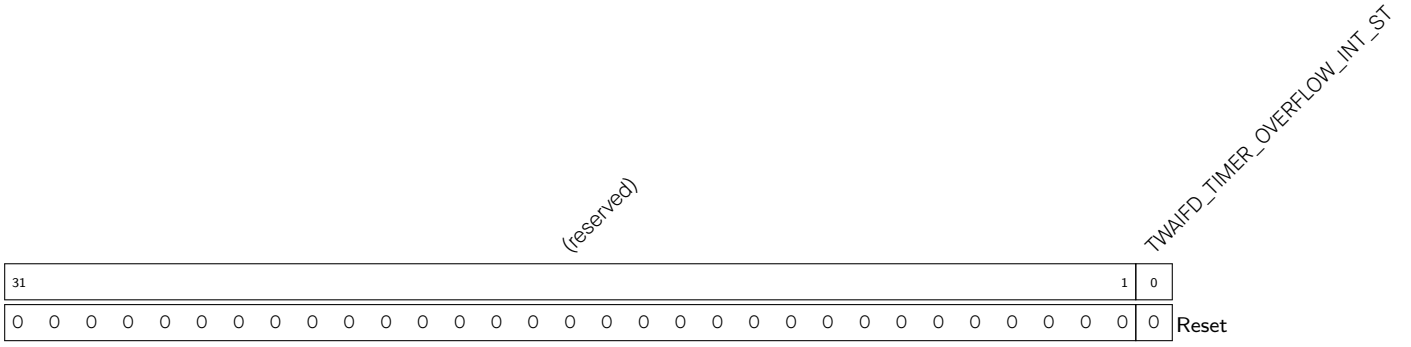
(R/W)

Register 38.9. TWAIFD_TIMER_INT_RAW_REG (0x0FD8)



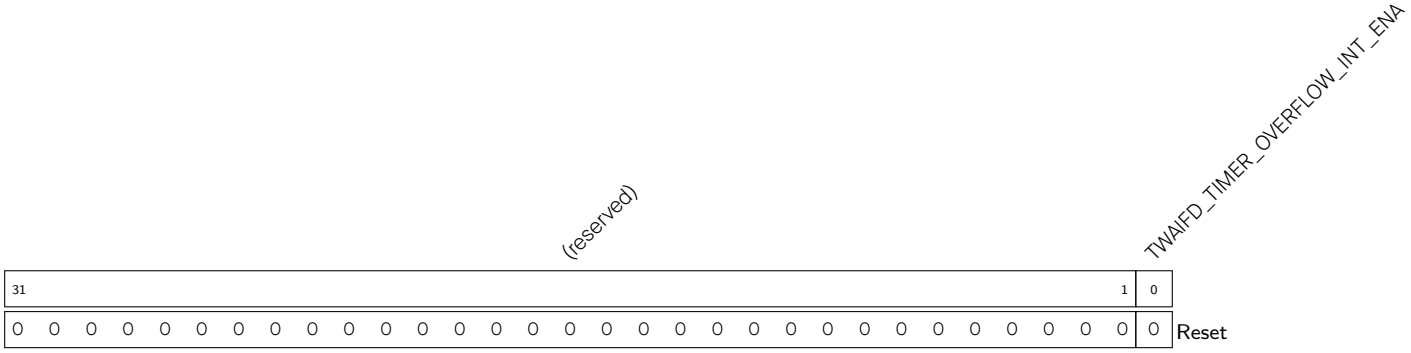
TWAIFD_TIMER_OVERFLOW_INT_RAW 定时器溢出中断的原始中断状态。(R/SS/WTC)

Register 38.10. TWAIFD_TIMER_INT_ST_REG (0x0FDC)



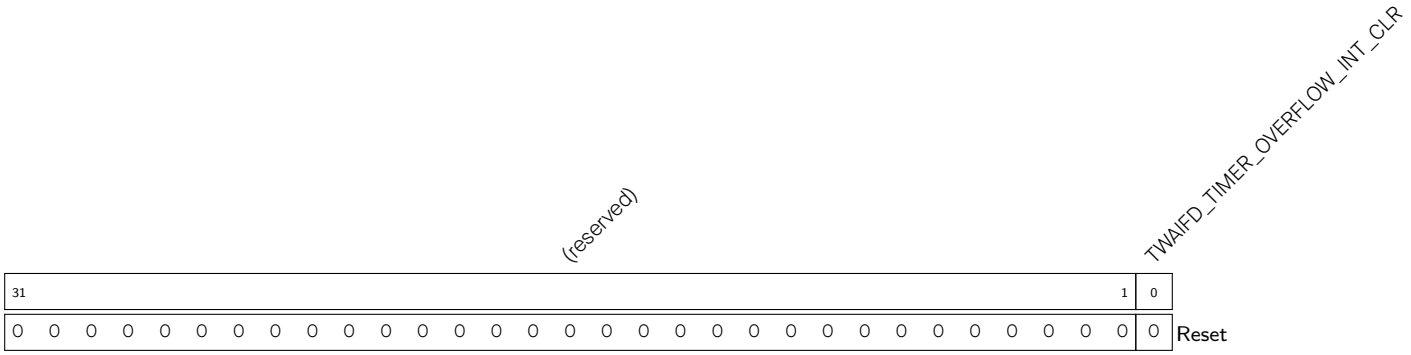
TWAIFD_TIMER_OVERFLOW_INT_ST 定时器溢出中断的屏蔽中断状态。(RO)

Register 38.11. TWAIFD_TIMER_INT_ENA_REG (0x0FE0)



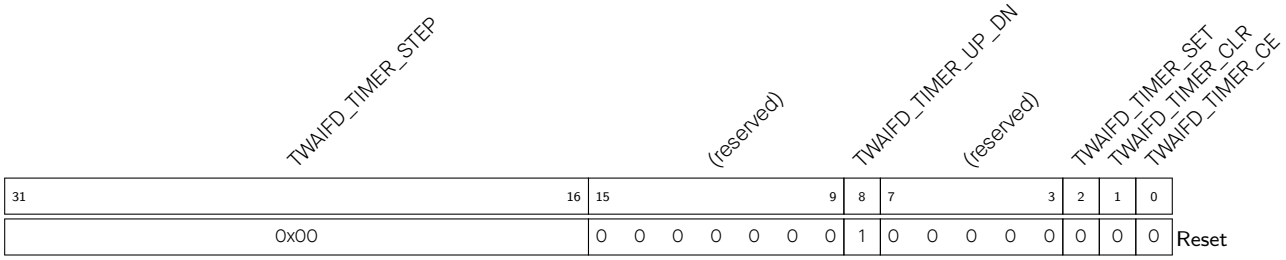
TWAIFD_TIMER_OVERFLOW_INT_ENA 写 1 启用定时器溢出中断。(R/W)

Register 38.12. TWAIFD_TIMER_INT_CLR_REG (0x0FE4)



TWAIFD_TIMER_OVERFLOW_INT_CLR 写 1 清除定时器溢出中断。(WT)

Register 38.13. TWAIFD_TIMER_CFG_REG (0x0FE8)



TWAIFD_TIMER_CE 配置是否启用定时器。

0：禁用

1：启用

(R/W)

TWAIFD_TIMER_CLR 配置是否清除定时器。

0：不清除

1：清除

(WT)

TWAIFD_TIMER_SET 配置是否设置定时器。

0：不设置

1：设置值为 ld_val

(WT)

TWAIFD_TIMER_UP_DN 配置定时器向上或向下计数。

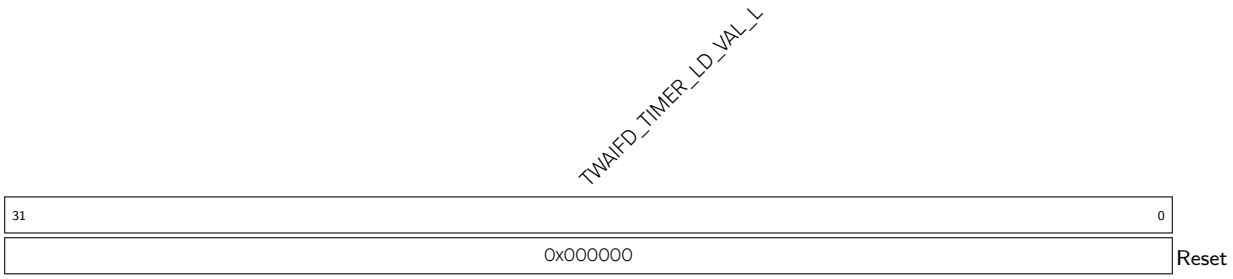
0：向下

1：向上

(R/W)

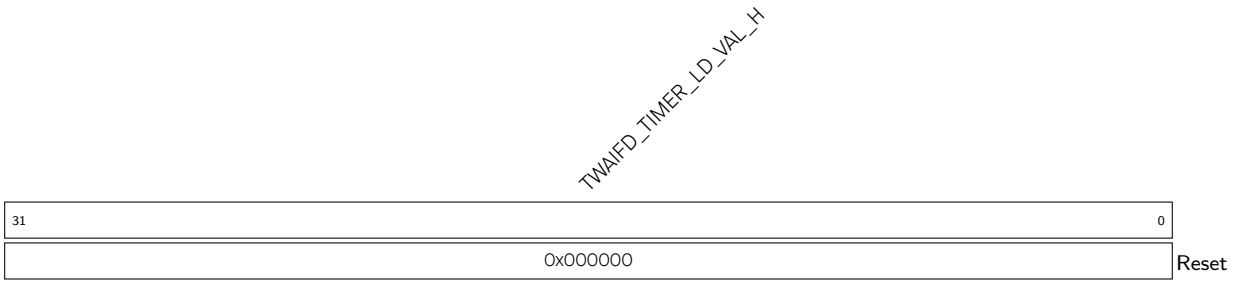
TWAIFD_TIMER_STEP 配置定时器计数步长，步长 = TWAIFD_TIMER_STEP + 1。(R/W)

Register 38.14. TWAIFD_TIMER_LD_VAL_L_REG (0x0FEC)



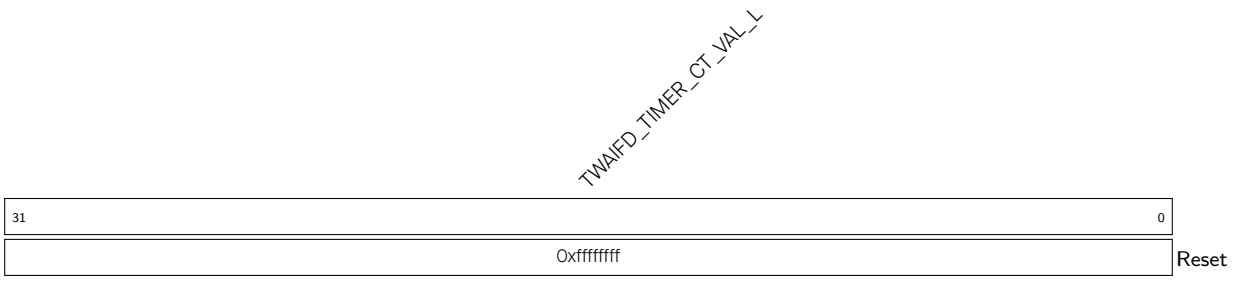
TWAIFD_TIMER_LD_VAL_L 配置定时器初始值的低位。(R/W)

Register 38.15. TWAIFD_TIMER_LD_VAL_H_REG (0x0FF0)



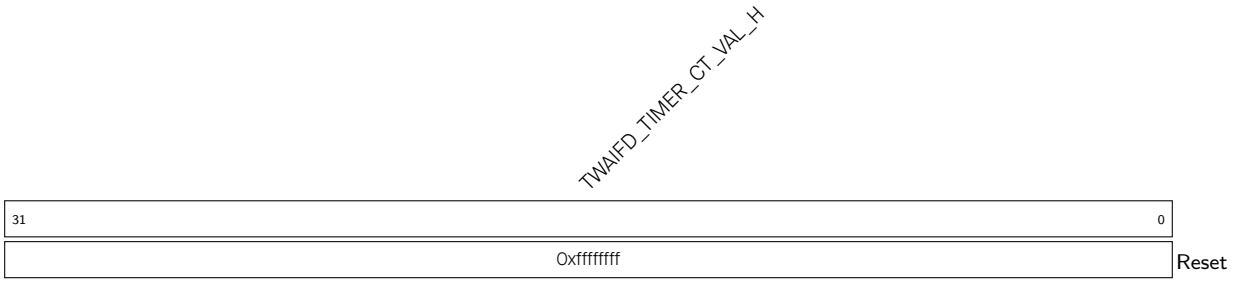
TWAIFD_TIMER_LD_VAL_H 配置定时器初始值的高位。如果时间戳有效位宽小于 33，则此字段被忽略。(R/W)

Register 38.16. TWAIFD_TIMER_CT_VAL_L_REG (0x0FF4)



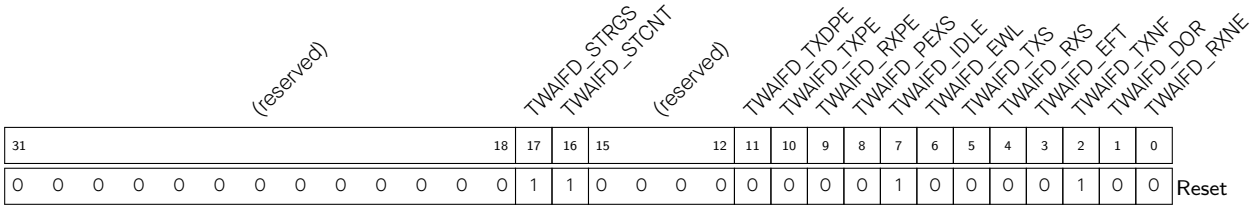
TWAIFD_TIMER_CT_VAL_L 配置定时器终到值的低位。(R/W)

Register 38.17. TWAIFD_TIMER_CT_VAL_H_REG (0x0FF8)



TWAIFD_TIMER_CT_VAL_H 配置定时器终到值的高位。如果时间戳有效位宽小于 33，则此字段被忽略。(R/W)

Register 38.18. TWAIFD_STATUS_REG (0x0008)



TWAIFD_RXNE 表示 RX buffer 不为空。当 buffer 中存储至少一帧数据时，该字段为 1。

0：为空

1：不为空

(RO)

TWAIFD_DOR 表示数据溢出标志。当由于 RX buffer 空间不足导致帧被丢弃时，该字段被置位。可通过 TWAIFD_RRB 清除。

0：未溢出

1：溢出

(RO)

TWAIFD_TXNF 表示 TX buffer 的状态。当至少有一个 TX buffer 为空时，该字段被置位。

0：未满

1：已满

(RO)

TWAIFD_EFT 表示是否正在发送错误帧。

0：未发送

1：正在发送

(RO)

TWAIFD_RXS 表示 CAN FD 是否为 CAN 帧的接收方。

0：未接收

1：正在接收

(RO)

TWAIFD_TXS 表示 CAN FD 是否为 CAN 帧的发送方。

0：未发送

1：正在发送

(RO)

TWAIFD_EWL 表示 TEC 或 REC 是否达到（大于或等于）EWL。

0：未达到

1：达到

(RO)

TWAIFD_IDLE 表示是否总线处于空闲状态（不在发送或接收帧）或 CAN FD 处于离线状态。

0：非空闲

1：空闲

(RO)

见下页...

Register 38.18. TWAIFD_STATUS_REG (0x0008)

... 接上页

TWAIFD_PEXS 表示是否发生协议异常。

0：未发生

1：发生，可通过向 TWAIFD_CPEXS 写 1 清除

(RO)

TWAIFD_RXPE 表示通过 TWAIFD_RX_DATA 从 RX buffer 读取 CAN 帧时是否检测到奇偶校验错误。

0：未检测到

1：已检测到

(RO)

TWAIFD_TXPE 表示从 TX buffer 发送帧时是否检测到奇偶校验错误。

0：未检测到

1：已检测到

(RO)

TWAIFD_TXDPE 表示在 TX buffer 备份模式下，备份 TX buffer 是否检测到双重奇偶校验错误。

0：未检测到

1：已检测到

(RO)

TWAIFD_STCNT 表示是否启用流量计数器。

0：禁用

1：启用

(RO)

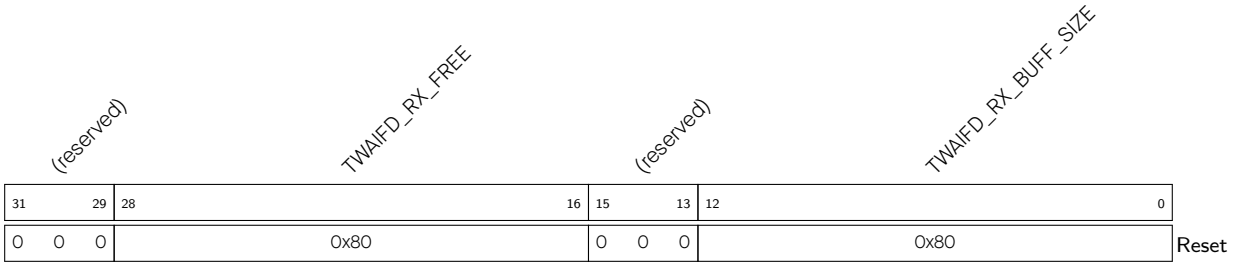
TWAIFD_STRGS 表示是否启用用于存储器可测试性的测试寄存器。

0：禁用

1：启用

(RO)

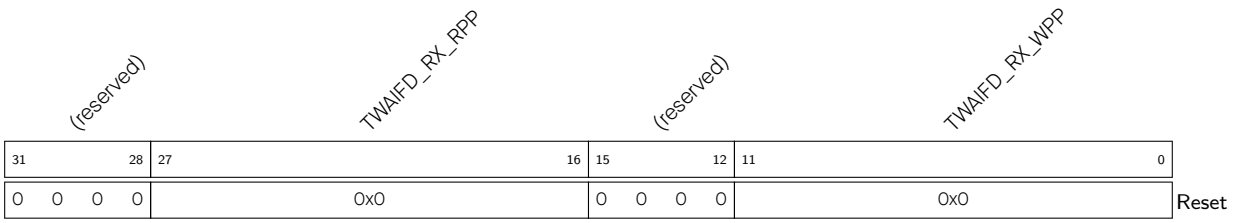
Register 38.19. TWAIFD_RX_MEM_INFO_REG (0x0060)



TWAIFD_RX_BUFF_SIZE 表示 RX buffer 的大小，以 32 位字为单位。(RO)

TWAIFD_RX_FREE 表示 RX buffer 空闲区域的大小，以 32 位字为单位。(RO)

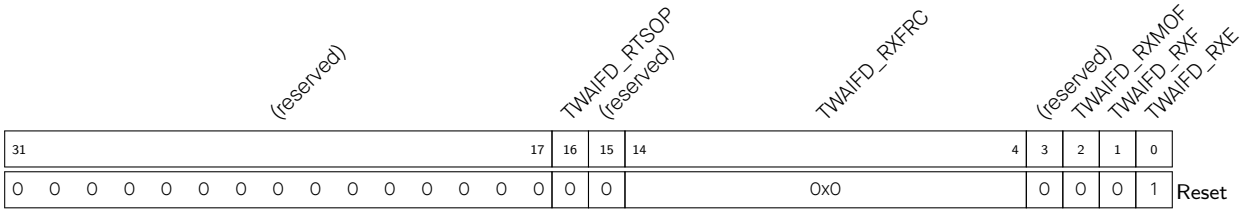
Register 38.20. TWAIFD_RX_POINTERS_REG (0x0064)



TWAIFD_RX_WPP 表示 RX buffer 中的写指针位置。在存储接收到的帧时，写指针会更新。(RO)

TWAIFD_RX_RPP 表示 RX buffer 中的读指针位置。在读取接收到的帧时，读指针会更新。(RO)

Register 38.21. TWAIFD_RX_STATUS_RX_SETTINGS_REG (0x0068)



TWAIFD_RXE 表示 RX buffer 是否为空。

0：不为空

1：为空

(RO)

TWAIFD_RXF 表示 RX buffer 是否已满。

0：未满

1：已满

(RO)

TWAIFD_RXMOF 表示 RX buffer 中接收到的帧数。RXMOF 表示 RX buffer 正在读取一帧数据。当该字段为 1 时，下次从 TWAIFD_RX_DATA 读取的数据将不是 CAN 帧的第一个字 (FRAME_FORMAT_W)。(RO)

TWAIFD_RXFRC 表示 RX buffer 当前存储的 CAN 帧数量。(RO)

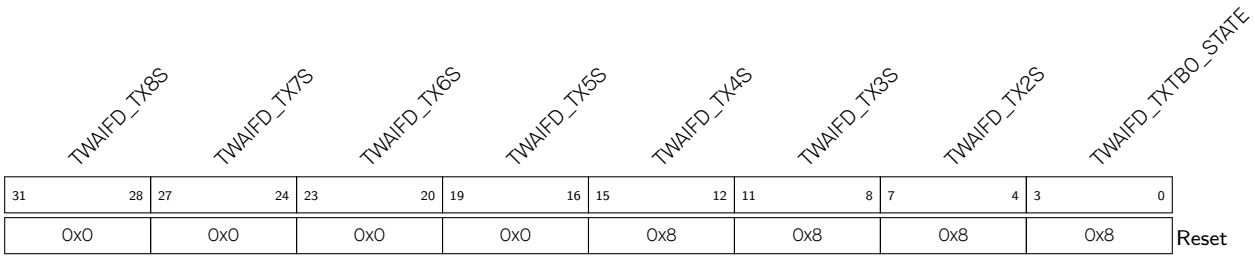
TWAIFD_RTSOP 表示 RX buffer 时间戳选项。仅在 TWAIFD_ENA = 0 时修改。

0：RTS_END，RX FIFO 接收到的帧的时间戳记录在 EOF 字段的最后一位

1：RTS_BEG，RX FIFO 接收到的帧的时间戳记录在 SOF 字段

(R/W)

Register 38.22. TWAIFD_TX_STATUS_REG (0x0070)



TWAIFD_TXTBO_STATE 表示 TX buffer 1 的状态。

0：TXT_NOT_EXIST，TX buffer 在核心中不存在（适用于 CAN FD 有少于 8 个 TX buffer）

1：TXT_RDY，TX buffer 处于 ready 状态，等待 CAN FD 启动发送

2：TXT_TRAN，TX buffer 处于 TX in progress 状态，CAN FD 正在发送帧

3：TXT_ABTP，TX buffer 处于 abort in progress 状态

4：TXT_TOK，TX buffer 处于 TX OK 状态

6：TXT_ERR，TX buffer 处于 failed 状态

7：TXT_ABT，TX buffer 处于 aborted 状态

8：TXT_ETY，TX buffer 处于 empty 状态

(RO)

TWAIFD_TX2S 表示 TX buffer 2 的状态。字段含义参考 TWAIFD_TXTBO_STATE。(RO)

TWAIFD_TX3S 表示 TX buffer 3 的状态。字段含义参考 TWAIFD_TXTBO_STATE。(RO)

TWAIFD_TX4S 表示 TX buffer 4 的状态。字段含义参考 TWAIFD_TXTBO_STATE。(RO)

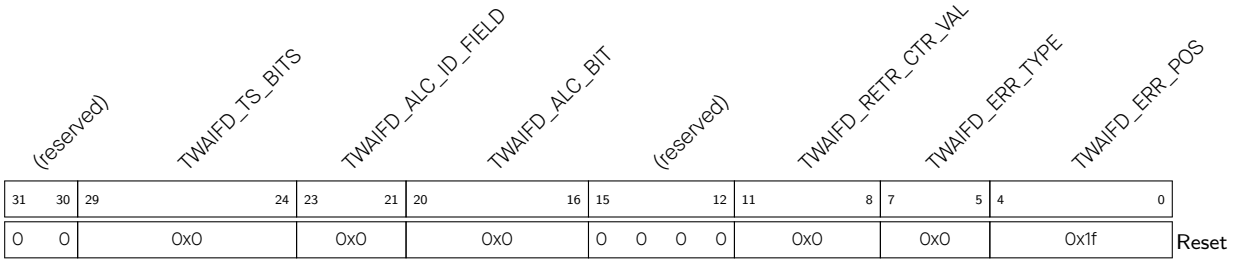
TWAIFD_TX5S 表示 TX buffer 5 的状态。字段含义参考 TWAIFD_TXTBO_STATE。(RO)

TWAIFD_TX6S 表示 TX buffer 6 的状态。字段含义参考 TWAIFD_TXTBO_STATE。(RO)

TWAIFD_TX7S 表示 TX buffer 7 的状态。字段含义参考 TWAIFD_TXTBO_STATE。(RO)

TWAIFD_TX8S 表示 TX buffer 8 的状态。字段含义参考 TWAIFD_TXTBO_STATE。(RO)

Register 38.23. TWAIFD_ERR_CAPT_RETR_CTR_ALC_TS_INFO_REG (0x007C)



TWAIFD_ERR_POS 表示最后一个错误的位置。0：ERC_POS_SOF，起始帧错误
1：ERC_POS_ARB，仲裁段错误
2：ERC_POS_CTRL，控制段错误
3：ERC_POS_DATA，数据段错误
4：ERC_POS_CRC，CRC 段错误
5：ERC_POS_ACK，CRC 分隔符、ACK 或 ACK 分隔符错误
6：ERC_POS_EOF，帧结束段错误
7：ERC_POS_ERR，错误帧期间出现错误
8：ERC_POS_OVRL，过载帧错误
31：ERC_POS_OTHER，其他位置错误
(RO)

TWAIFD_ERR_TYPE 表示最后一个错误的类型。
0：ERC_BIT_ERR，位错误
1：ERC_CRC_ERR，CRC 错误
2：ERC_FRM_ERR，格式错误
3：ERC_ACK_ERR，ACK 错误
4：ERC_STUF_ERR，填充错误
(RO)

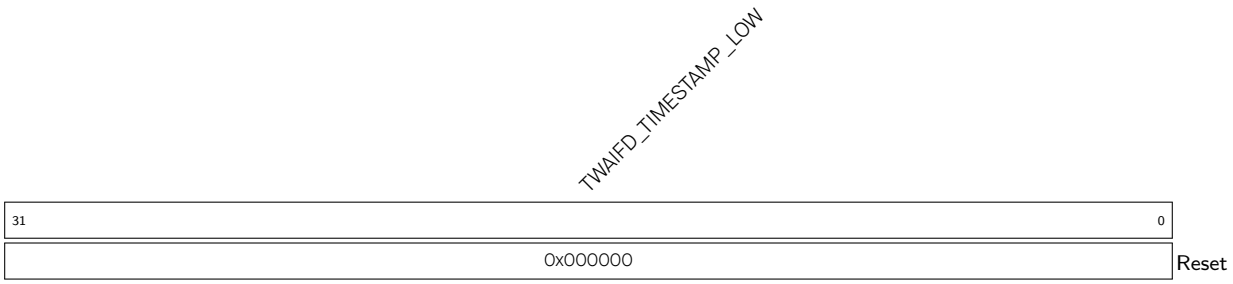
TWAIFD_RETR_CTR_VAL 表示重发计数器的当前值。(RO)

TWAIFD_ALC_BIT 表示仲裁丢失时的位捕获位置。若该字段为 ALC_BASE_ID，则仲裁丢失的基本标识符位索引为 11 - ALC_VAL。如果 ALC_ID_FIELD = ALC_EXTENSION，则仲裁丢失的扩展标识符位索引为 18 - ALC_VAL。其他值无效。(RO)

TWAIFD_ALC_ID_FIELD 表示仲裁丢失时 CAN 标识符中的部分。
0：ALC_RSVD，自上次复位后未丢失过仲裁
1：ALC_BASE_ID，在基本标识符期间丢失仲裁
2：ALC_SRR_RTR，在基本标识符之后的第一个位丢失仲裁（扩展帧的 SRR 位，CAN 2.0 Base 帧的 RTR 位）
3：ALC_IDE，在 IDE 位期间丢失仲裁
4：ALC_EXTENSION，在拓展标识符期间丢失仲裁
5：ALC_RTR，在拓展标识符之后的 RTR 位期间丢失仲裁
(RO)

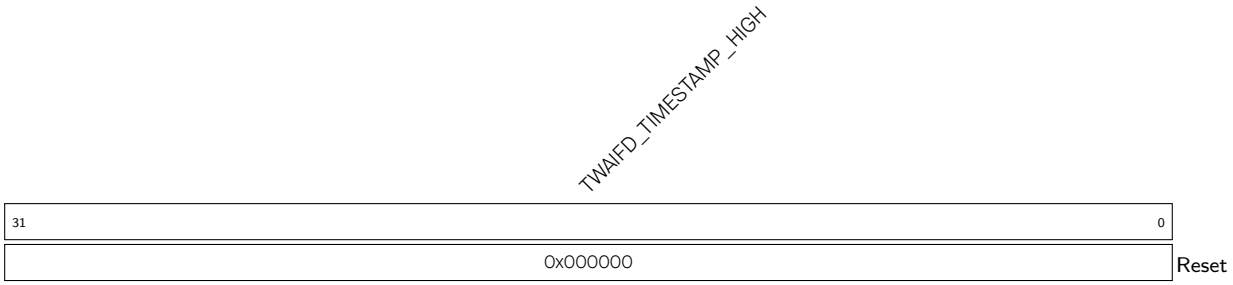
TWAIFD_TS_BITS 表示时间基准的有效位数减 1（0x3F = 64 位时间基准）。(RO)

Register 38.24. TWAIFD_TIMESTAMP_LOW_REG (0x0094)



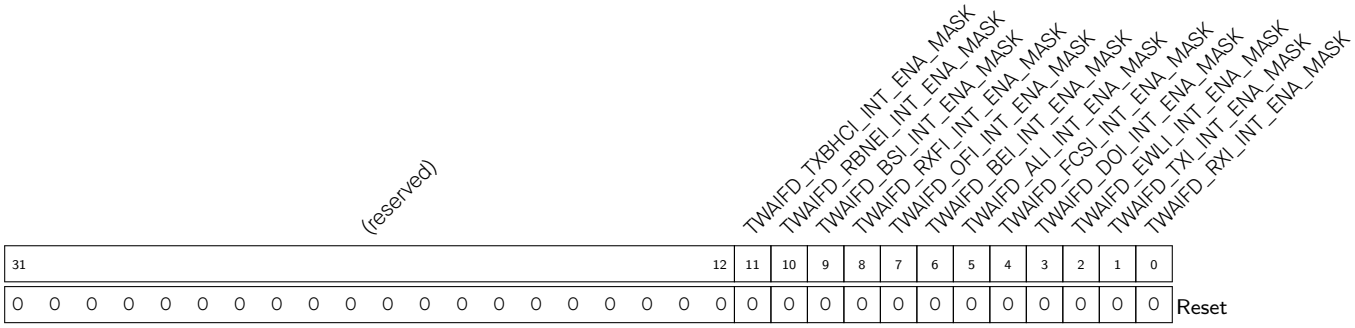
TWAIFD_TIMESTAMP_LOW 表示时间基准的 0:31 位。(RO)

Register 38.25. TWAIFD_TIMESTAMP_HIGH_REG (0x0098)



TWAIFD_TIMESTAMP_HIGH 表示时间基准的 32:63 位。(RO)

Register 38.26. TWAIFD_INT_ENA_SET_REG (0x0014)



TWAIFD_RXI_INT_ENA_MASK 写 1 使能 TWAIFD_RXI_INT。(R/W1S)

TWAIFD_TXI_INT_ENA_MASK 写 1 使能 TWAIFD_TXI_INT。(R/W1S)

TWAIFD_EWLI_INT_ENA_MASK 写 1 使能 TWAIFD_EWLI_INT。(R/W1S)

TWAIFD_DOI_INT_ENA_MASK 写 1 使能 TWAIFD_DOI_INT。(R/W1S)

TWAIFD_FCSI_INT_ENA_MASK 写 1 使能 TWAIFD_FCSI_INT。(R/W1S)

TWAIFD_ALI_INT_ENA_MASK 写 1 使能 TWAIFD_ALI_INT。(R/W1S)

TWAIFD_BEI_INT_ENA_MASK 写 1 使能 TWAIFD_BEI_INT。(R/W1S)

TWAIFD_OFI_INT_ENA_MASK 写 1 使能 TWAIFD_OFI_INT。(R/W1S)

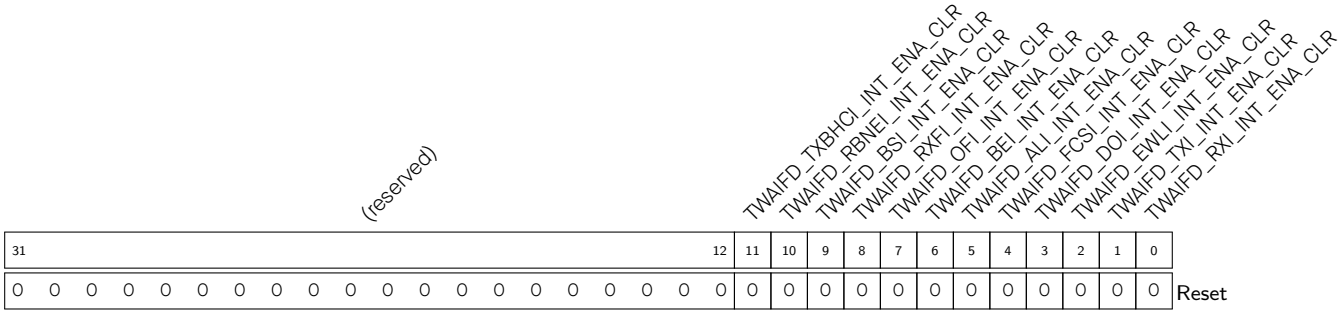
TWAIFD_RXFI_INT_ENA_MASK 写 1 使能 TWAIFD_RXFI_INT。(R/W1S)

TWAIFD_BSI_INT_ENA_MASK 写 1 使能 TWAIFD_BSI_INT。(R/W1S)

TWAIFD_RBNEI_INT_ENA_MASK 写 1 使能 TWAIFD_RBNEI_INT。(R/W1S)

TWAIFD_TXBHCI_INT_ENA_MASK 写 1 使能 TWAIFD_TXBHCI_INT。(R/W1S)

Register 38.27. TWAIFD_INT_ENA_CLR_REG (0x0018)



TWAIFD_RXI_INT_ENA_CLR 写 1 清除 TWAIFD_RXI_INT_ENA。(WO)

TWAIFD_TXI_INT_ENA_CLR 写 1 清除 TWAIFD_TXI_INT_ENA。(WO)

TWAIFD_EWLI_INT_ENA_CLR 写 1 清除 TWAIFD_EWLI_INT_ENA。(WO)

TWAIFD_DOI_INT_ENA_CLR 写 1 清除 TWAIFD_DOI_INT_ENA。(WO)

TWAIFD_FCSI_INT_ENA_CLR 写 1 清除 TWAIFD_FCSI_INT_ENA。(WO)

TWAIFD_ALI_INT_ENA_CLR 写 1 清除 TWAIFD_ALI_INT_ENA。(WO)

TWAIFD_BEI_INT_ENA_CLR 写 1 清除 TWAIFD_BEI_INT_ENA。(WO)

TWAIFD_OFI_INT_ENA_CLR 写 1 清除 TWAIFD_OFI_INT_ENA。(WO)

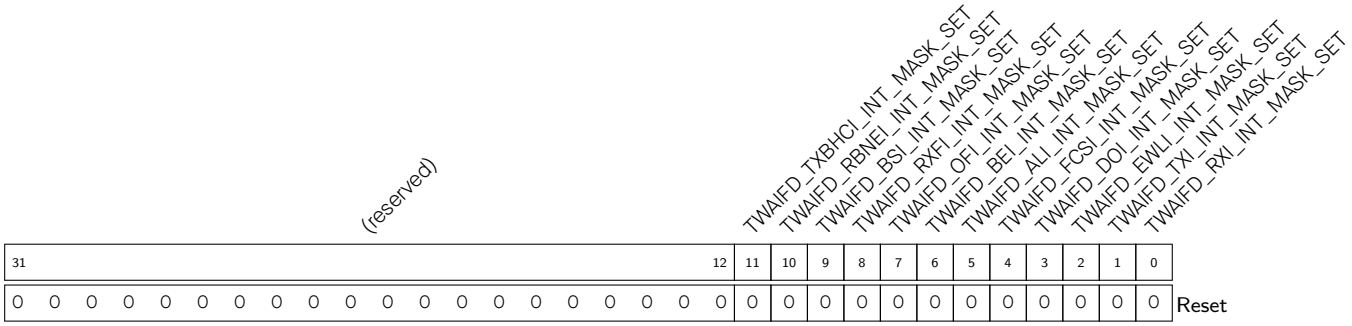
TWAIFD_RXFI_INT_ENA_CLR 写 1 清除 TWAIFD_RXFI_INT_ENA。(WO)

TWAIFD_BSI_INT_ENA_CLR 写 1 清除 TWAIFD_BSI_INT_ENA。(WO)

TWAIFD_RBNEI_INT_ENA_CLR 写 1 清除 TWAIFD_RBNEI_INT_ENA。(WO)

TWAIFD_TXBHCI_INT_ENA_CLR 写 1 清除 TWAIFD_TXBHCI_INT_ENA。(WO)

Register 38.28. TWAIFD_INT_MASK_SET_REG (0x001C)



TWAIFD_RXI_INT_MASK_SET 写 1 屏蔽 TWAIFD_RXI_INT 的中断状态。(R/W1S)

TWAIFD_TXI_INT_MASK_SET 写 1 屏蔽 TWAIFD_TXI_INT 的中断状态。(R/W1S)

TWAIFD_EWLI_INT_MASK_SET 写 1 屏蔽 TWAIFD_EWLI_INT 的中断状态。(R/W1S)

TWAIFD_DOI_INT_MASK_SET 写 1 屏蔽 TWAIFD_DOI_INT 的中断状态。(R/W1S)

TWAIFD_FCSI_INT_MASK_SET 写 1 屏蔽 TWAIFD_FCSI_INT 的中断状态。(R/W1S)

TWAIFD_ALI_INT_MASK_SET 写 1 屏蔽 TWAIFD_ALI_INT 的中断状态。(R/W1S)

TWAIFD_BEI_INT_MASK_SET 写 1 屏蔽 TWAIFD_BEI_INT 的中断状态。(R/W1S)

TWAIFD_OFI_INT_MASK_SET 写 1 屏蔽 TWAIFD_OFI_INT 的中断状态。(R/W1S)

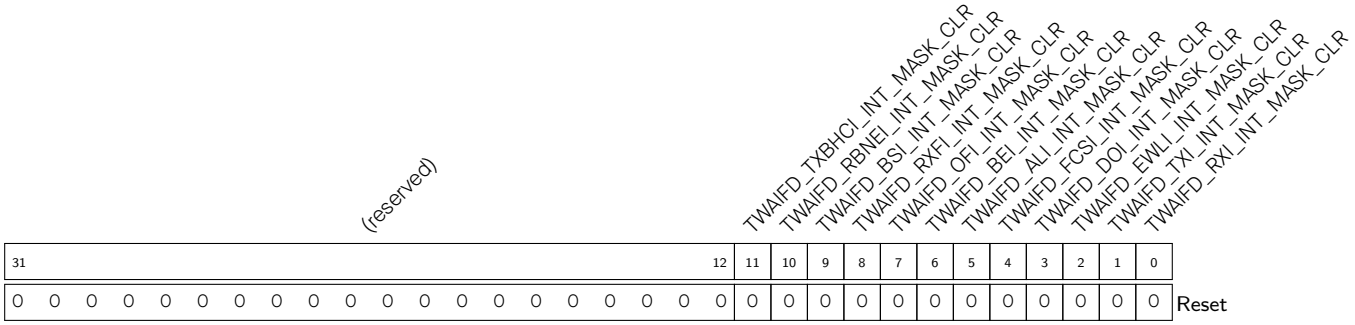
TWAIFD_RXFI_INT_MASK_SET 写 1 屏蔽 TWAIFD_RXFI_INT 的中断状态。(R/W1S)

TWAIFD_BSI_INT_MASK_SET 写 1 屏蔽 TWAIFD_BSI_INT 的中断状态。(R/W1S)

TWAIFD_RBNEI_INT_MASK_SET 写 1 屏蔽 TWAIFD_RBNEI_INT 的中断状态。(R/W1S)

TWAIFD_TXBHCI_INT_MASK_SET 写 1 屏蔽 TWAIFD_TXBHCI_INT 的中断状态。(R/W1S)

Register 38.29. TWAIFD_INT_MASK_CLR_REG (0x0020)



TWAIFD_RXI_INT_MASK_CLR 写 1 清除 TWAIFD_RXI_INT_MASK_CLR。(WO)

TWAIFD_TXI_INT_MASK_CLR 写 1 清除 TWAIFD_TXI_INT_MASK_CLR。(WO)

TWAIFD_EWLI_INT_MASK_CLR 写 1 清除 TWAIFD_EWLI_INT_MASK_CLR。(WO)

TWAIFD_DOI_INT_MASK_CLR 写 1 清除 TWAIFD_DOI_INT_MASK_CLR。(WO)

TWAIFD_FCSI_INT_MASK_CLR 写 1 清除 TWAIFD_FCSI_INT_MASK_CLR。(WO)

TWAIFD_ALI_INT_MASK_CLR 写 1 清除 TWAIFD_ALI_INT_MASK_CLR。(WO)

TWAIFD_BEI_INT_MASK_CLR 写 1 清除 TWAIFD_BEI_INT_MASK_CLR。(WO)

TWAIFD_OFI_INT_MASK_CLR 写 1 清除 TWAIFD_OFI_INT_MASK_CLR。(WO)

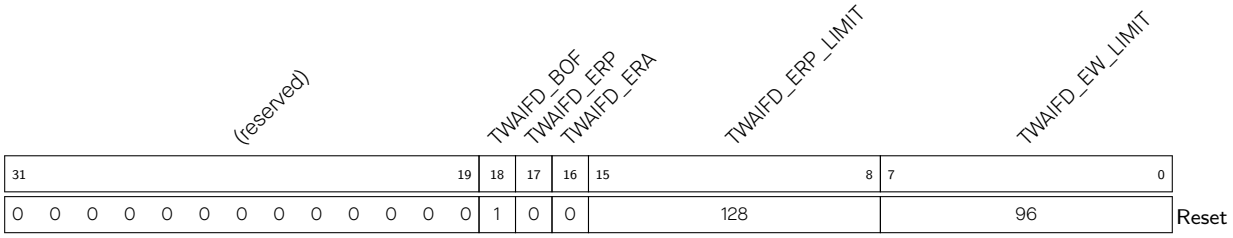
TWAIFD_RXFI_INT_MASK_CLR 写 1 清除 TWAIFD_RXFI_INT_MASK_CLR。(WO)

TWAIFD_BSI_INT_MASK_CLR 写 1 清除 TWAIFD_BSI_INT_MASK_CLR。(WO)

TWAIFD_RBNEI_INT_MASK_CLR 写 1 清除 TWAIFD_RBNEI_INT_MASK_CLR。(WO)

TWAIFD_TXBHCI_INT_MASK_CLR 写 1 清除 TWAIFD_TXBHCI_INT_MASK_CLR。(WO)

Register 38.30. TWAIFD_EWL_ERP_FAULT_STATE_REG (0x002C)



TWAIFD_EW_LIMIT 配置错误警告限制。当达到错误警告阈值时，可以产生中断。错误警告阈值表示总线干扰严重。(R/W)

TWAIFD_ERP_LIMIT 配置被动错误限制。当任一错误计数器 (REC/TEC) 超过该值时，故障限制状态将变为被动错误状态。(R/W)

TWAIFD_ERA 表示主动错误故障状态。

0：未出现主动错误故障状态

1：出现主动错误故障状态

(RO)

TWAIFD_ERP 表示被动错误故障状态。

0：未出现被动错误故障状态

1：出现被动错误故障状态

(RO)

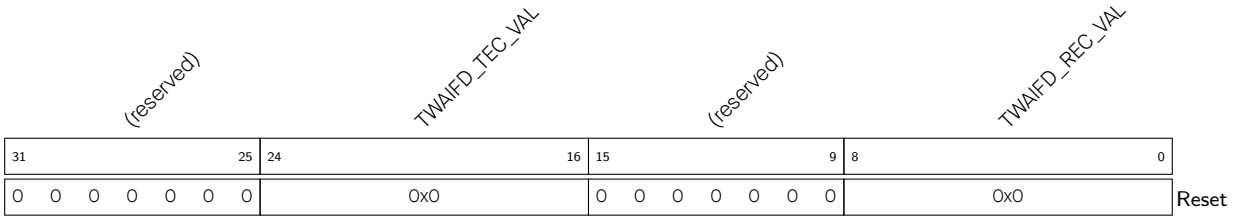
TWAIFD_BOF 表示离线故障状态。

0：未出现离线故障状态

1：出现离线故障状态

(RO)

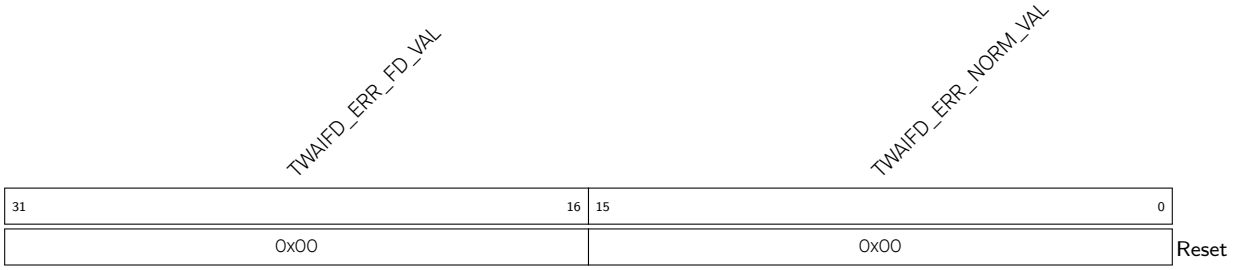
Register 38.31. TWAIFD_REC_TEC_REG (0x0030)



TWAIFD_REC_VAL 表示接收器错误计数器的值。(RO)

TWAIFD_TEC_VAL 表示发送器错误计数器的值。(RO)

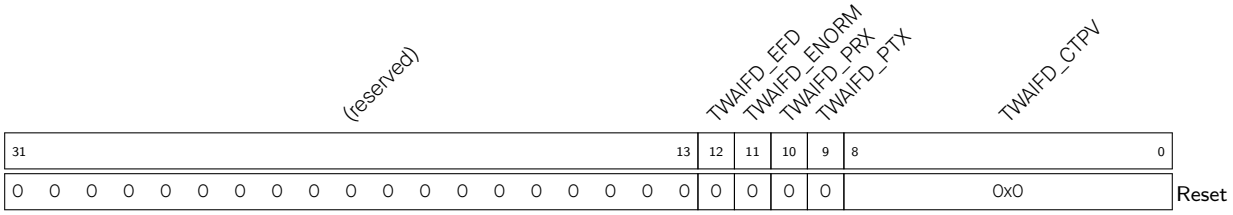
Register 38.32. TWAIFD_ERR_NORM_ERR_FD_REG (0x0034)



TWAIFD_ERR_NORM_VAL 表示标称位时间内的错误数量。(RO)

TWAIFD_ERR_FD_VAL 表示数据位时间内的错误数量。(RO)

Register 38.33. TWAIFD_CTR_PRES_REG (0x0038)



TWAIFD_CTPV 配置预设值用于设置错误计数器。(WO)

TWAIFD_PTX 配置是否将接收器错误计数器设置为预定义值。

0：无效

1：设置

(WT)

TWAIFD_PRX 配置是否将发送器错误计数器设置为预定义值。

0：无效

1：设置

(WT)

TWAIFD_ENORM 配置是否清除标称位时间的错误计数器。

0：无效

1：清除

(WO)

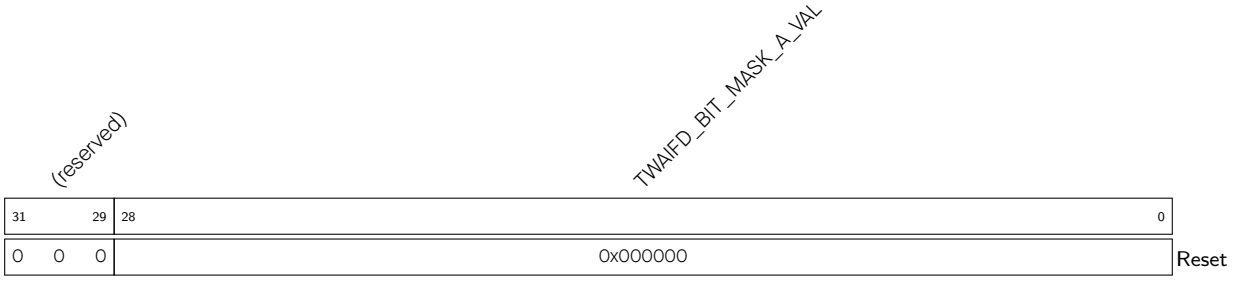
TWAIFD_EFD 配置是否清除数据位时间的错误计数器。

0：无效

1：清除

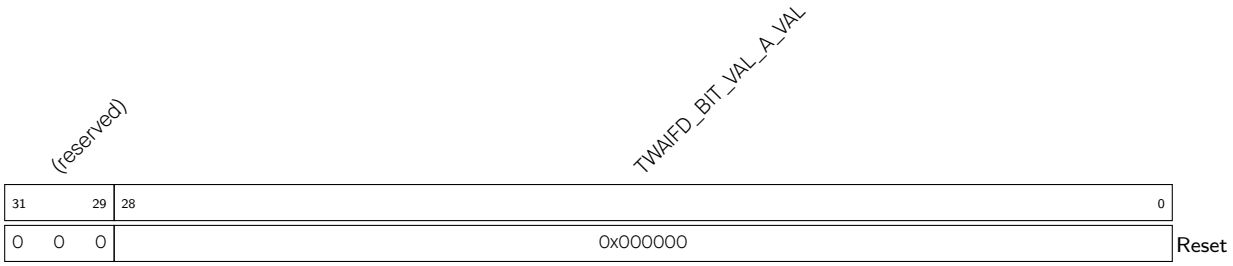
(WO)

Register 38.34. TWAIFD_FILTER_A_MASK_REG (0x003C)



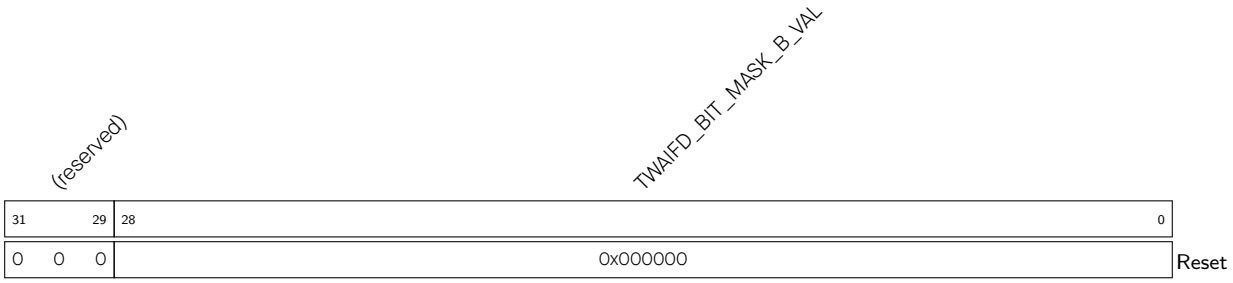
TWAIFD_BIT_MASK_A_VAL 配置过滤器 A 的屏蔽值。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果过滤器 A 不存在，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.35. TWAIFD_FILTER_A_VAL_REG (0x0040)



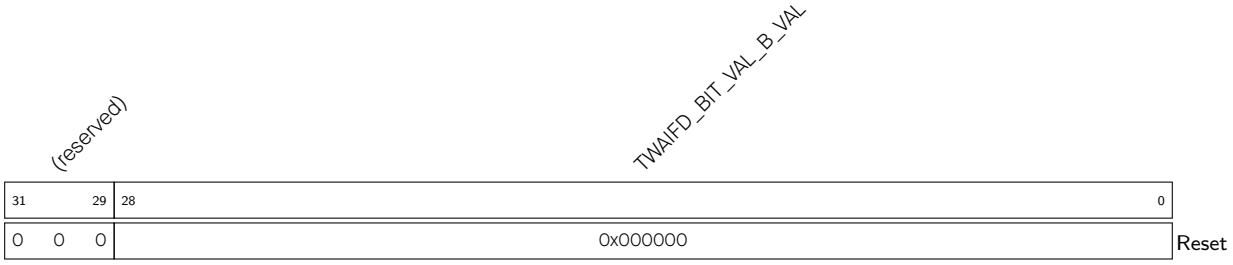
TWAIFD_BIT_VAL_A_VAL 配置过滤器 A 的位值。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果过滤器 A 不存在，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.36. TWAIFD_FILTER_B_MASK_REG (0x0044)



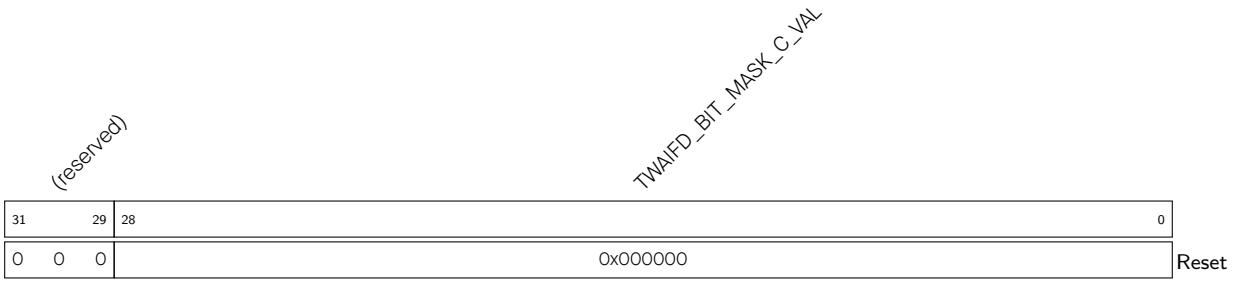
TWAIFD_BIT_MASK_B_VAL 配置过滤器 B 的屏蔽值。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果过滤器 B 不存在，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.37. TWAIFD_FILTER_B_VAL_REG (0x0048)



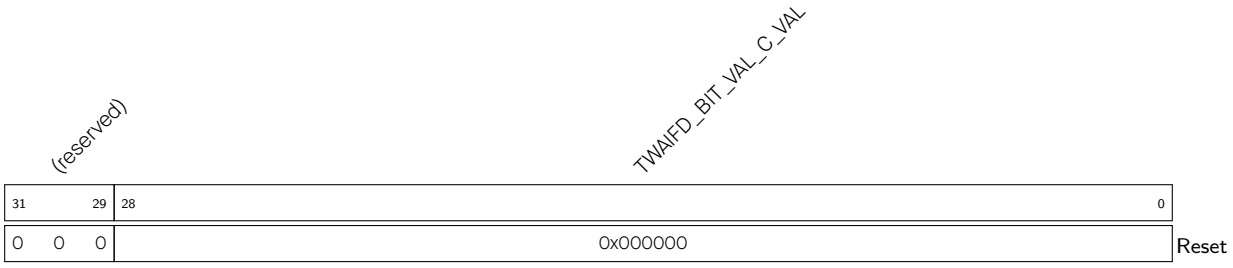
TWAIFD_BIT_VAL_B_VAL 配置过滤器 B 的位值。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果过滤器 B 不存在，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.38. TWAIFD_FILTER_C_MASK_REG (0x004C)



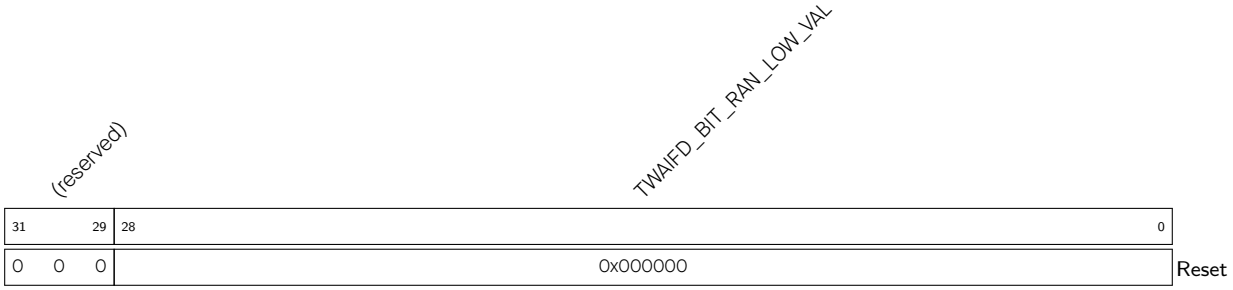
TWAIFD_BIT_MASK_C_VAL 配置过滤器 C 的屏蔽值。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果过滤器 C 不存在，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.39. TWAIFD_FILTER_C_VAL_REG (0x0050)



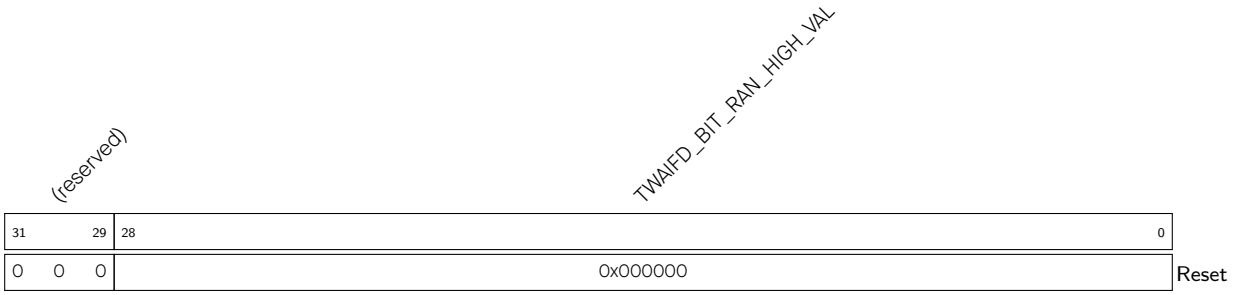
TWAIFD_BIT_VAL_C_VAL 配置过滤器 C 的位值。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果过滤器 C 不存在，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.40. TWAIFD_FILTER_RAN_LOW_REG (0x0054)



TWAIFD_BIT_RAN_LOW_VAL 配置范围过滤器的阈值下限。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果不支持范围过滤器，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.41. TWAIFD_FILTER_RAN_HIGH_REG (0x0058)



TWAIFD_BIT_RAN_HIGH_VAL 配置范围过滤器的阈值上限。标识符格式与 TX 或 RX buffer 的 IDENTIFIER_W 相同。如果不支持范围过滤器，写入此寄存器无效，读取将返回全零。(R/W)

Register 38.42. TWAIFD_FILTER_CONTROL_FILTER_STATUS_REG (0x005C)



TWAIFD_FANB 配置过滤器 A 是否接收 CAN 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FANE 配置过滤器 A 是否接收 CAN 扩展帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FAFB 配置过滤器 A 是否接收 CAN FD 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FAFE 配置过滤器 A 是否接收 CAN FD 扩展帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FBNB 配置过滤器 B 是否接收 CAN 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FBNE 配置过滤器 B 是否接收 CAN 扩展帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FBFB 配置过滤器 B 是否接收 CAN FD 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FBFE 配置过滤器 B 是否接收 CAN FD 扩展帧。

0：未接收

1：已接收

(R/W)

见下页...

Register 38.42. TWAIFD_FILTER_CONTROL_FILTER_STATUS_REG (0x005C)

... 接上页

TWAIFD_FCNB 配置过滤器 C 是否接收 CAN 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FCNE 配置过滤器 C 是否接收 CAN 扩展帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FCFB 配置过滤器 C 是否接收 CAN FD 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FCFE 配置过滤器 C 是否接收 CAN FD 扩展帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FRNB 配置范围过滤器是否接收 CAN 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FRNE 配置范围过滤器是否接收 CAN 扩展帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FRFB 配置范围过滤器是否接收 CAN FD 基本帧。

0：未接收

1：已接收

(R/W)

TWAIFD_FRFE 配置范围过滤器是否接收 CAN FD 扩展帧。

0：未接收

1：已接收

(R/W)

TWAIFD_SFA 表示过滤器 A 是否可用。

0：不可用

1：可用

(RO)

见下页...

Register 38.42. TWAIFD_FILTER_CONTROL_FILTER_STATUS_REG (0x005C)

... 接上页

TWAIFD_SFB 表示过滤器 B 是否可用。

0：不可用

1：可用

(RO)

TWAIFD_SFC 表示过滤器 C 是否可用。

0：不可用

1：可用

(RO)

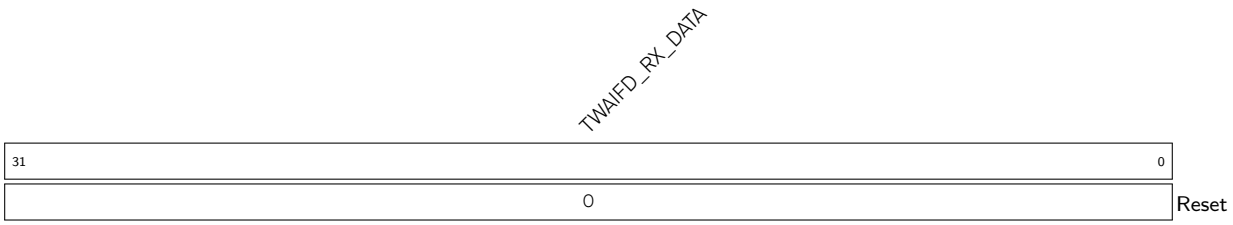
TWAIFD_SFR 表示范围过滤器是否可用。

0：不可用

1：可用

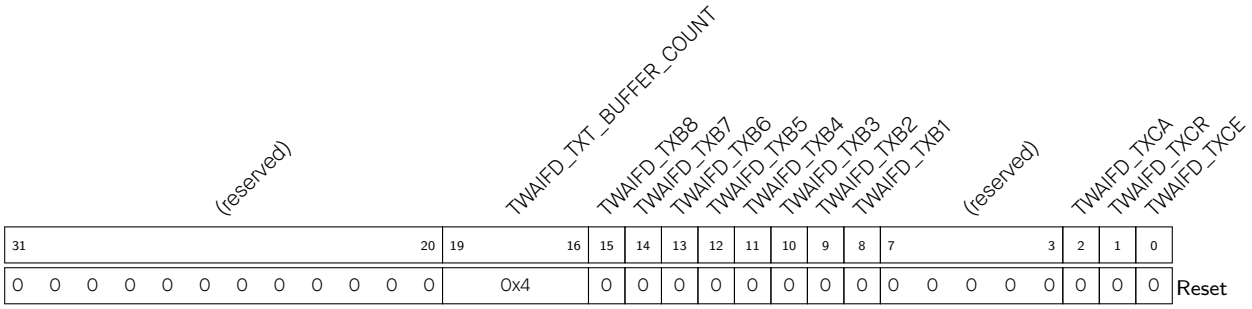
(RO)

Register 38.43. TWAIFD_RX_DATA (0x006C)



TWAIFD_RX_DATA 表示 FIFO 中读取指针位置的 RX buffer 数据。每次读取此寄存器后，只要 RX buffer 中还有下一个数据字，读取指针会自动增加。缓冲区存储的第一个字为 FRAME_FORMAT_W，接下来是 IDENTIFIER_W，依次类推。此寄存器应通过 32 位访问方式读取。(RO)

Register 38.44. TWAIFD_TX_COMMAND_TXTB_INFO_REG (0x0074)



TWAIFD_TXCE 配置是否发出 set empty 命令。

0：不发出

1：发出

(WO)

TWAIFD_TXCR 配置是否发出 set ready 命令。

0：不发出

1：发出

(WO)

TWAIFD_TXCA 配置是否发出 set abort 命令。

0：不发出

1：发出

(WO)

TWAIFD_TXB1 配置是否发出命令到 TX buffer 1。

0：不发出

1：发出

(WO)

TWAIFD_TXB2 配置是否发出命令到 TX buffer 2。

0：不发出

1：发出

(WO)

TWAIFD_TXB3 配置是否发出命令到 TX buffer 3。如果 TX buffer 数量少于 3，此字段保留且无功能。

0：不发出

1：发出

(WO)

TWAIFD_TXB4 配置是否发出命令到 TX buffer 4。如果 TX buffer 数量少于 4，此字段保留且无功能。

0：不发出

1：发出

(WO)

见下页...

Register 38.44. TWAIFD_TX_COMMAND_TXTB_INFO_REG (0x0074)

... 接上页

TWAIFD_TXB5 配置是否发出命令到 TX buffer 5。如果 TX buffer 数量少于 5，此字段保留且无功能。

0：不发出

1：发出

(WO)

TWAIFD_TXB6 配置是否发出命令到 TX buffer 6。如果 TX buffer 数量少于 6，此字段保留且无功能。

0：不发出

1：发出

(WO)

TWAIFD_TXB7 配置是否发出命令到 TX buffer 7。如果 TX buffer 数量少于 7，此字段保留且无功能。

0：不发出

1：发出

(WO)

TWAIFD_TXB8 配置是否发出命令到 TX buffer 8。如果 TX buffer 数量少于 8，此字段保留且无功能。

0：不发出

1：发出

(WO)

TWAIFD_TXT_BUFFER_COUNT 表示 CAN FD 中存在的 TX buffer 数量。最低缓冲区始终为 1。最高缓冲区的索引等于存在的缓冲区数量。(RO)

Register 38.45. TWAIFD_TX_PRIORITY_REG (0x0078)



TWAIFD_TXT1P 配置 TX buffer 1 的优先级，数值越高，优先级越高。(R/W)

TWAIFD_TXT2P 配置 TX buffer 2 的优先级，数值越高，优先级越高。(R/W)

TWAIFD_TXT3P 配置 TX buffer 3 的优先级，数值越高，优先级越高。如果 TX buffer 数量少于 3，此字段保留且无功能。(R/W)

TWAIFD_TXT4P 配置 TX buffer 4 的优先级，数值越高，优先级越高。如果 TX buffer 数量少于 4，此字段保留且无功能。(R/W)

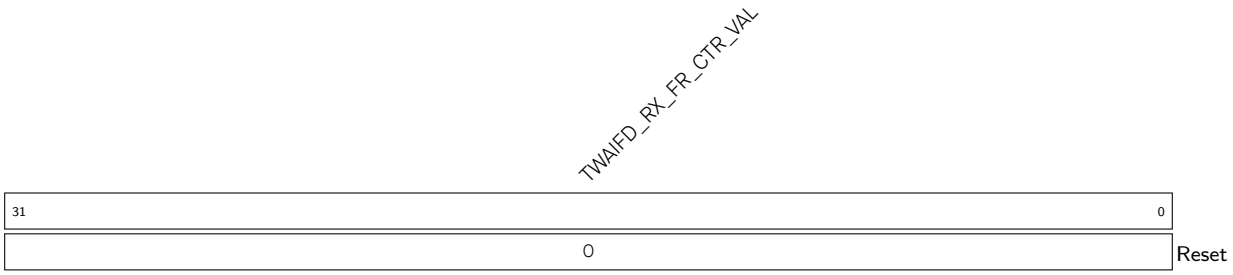
TWAIFD_TXT5P 配置 TX buffer 5 的优先级，数值越高，优先级越高。如果 TX buffer 数量少于 5，此字段保留且无功能。(R/W)

TWAIFD_TXT6P 配置 TX buffer 6 的优先级，数值越高，优先级越高。如果 TX buffer 数量少于 6，此字段保留且无功能。(R/W)

TWAIFD_TXT7P 配置 TX buffer 7 的优先级，数值越高，优先级越高。如果 TX buffer 数量少于 7，此字段保留且无功能。(R/W)

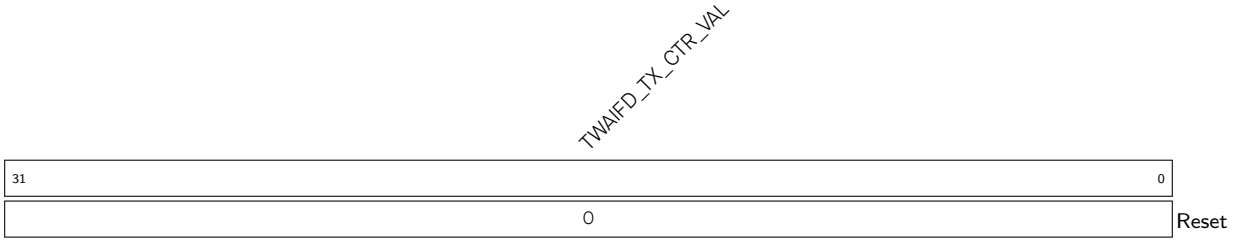
TWAIFD_TXT8P 配置 TX buffer 8 的优先级，数值越高，优先级越高。如果 TX buffer 数量少于 8，此字段保留且无功能。(R/W)

Register 38.46. TWAIFD_RX_FR_CTR_REG (0x0084)



TWAIFD_RX_FR_CTR_VAL 表示接收的帧数。(RO)

Register 38.47. TWAIFD_TX_FR_CTR_REG (0x0088)



TWAIFD_TX_CTR_VAL 表示发送的帧数。(RO)

Register 38.48. TWAIFD_DEBUG_REG (0x008C)

Bits	31:19	18	17	16	15	14	13	12	11	10	9	8	7	6	5:3	2:0
Field	(reserved)	TWAIFD_PC_SOF	TWAIFD_PC_OVR	TWAIFD_PC_SUSP	TWAIFD_PC_INT	TWAIFD_PC_EOF	TWAIFD_PC_ACKD	TWAIFD_PC_ACK	TWAIFD_PC_CRCD	TWAIFD_PC_CRC	TWAIFD_PC_STC	TWAIFD_PC_DAT	TWAIFD_PC_CON	TWAIFD_PC_ARB	TWAIFD_DESTUFF_COUNT	TWAIFD_STUFF_COUNT
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TWAIFD_STUFF_COUNT 表示当前的填充位计数，对 8 取模，符合 ISO FD 协议定义。每个 CAN 帧开始时填充计数被重置，在到达 ISO FD 帧的填充计数字段之前，每检测到一个填充位加一，之后该值保持不变，直到下一帧开始。在非 ISO FD 或标准 CAN 中，填充位的计数会一直持续到帧结束。请注意，此字段并未采用 ISO FD 标准中规定的格雷码编码。填充计数仅在控制器主动在总线上收发数据时更新；在接收过程中，此值保持不变。(RO)

TWAIFD_DESTUFF_COUNT 表示当前的去填充位计数，对 8 取模，符合 ISO FD 协议定义。每个帧开始时去填充计数被重置，在到达 ISO FD 帧的填充计数字段之前，每检测到一个去填充位加一，之后该值保持不变，直到下一帧开始。在非 ISO FD 或标准 CAN 中，去填充位的计数会一直持续到帧结束。请注意，此字段并未采用 ISO FD 标准中规定的格雷码编码。去填充计数在发送和接收过程中都会被更新。(RO)

TWAIFD_PC_ARB 表示协议控制状态机是否处于仲裁字段。
0：不处于
1：处于
(RO)

TWAIFD_PC_CON 表示协议控制状态机处于控制字段。
0：不处于
1：处于
(RO)

TWAIFD_PC_DAT 表示协议控制状态机处于数据字段。
0：不处于
1：处于
(RO)

TWAIFD_PC_STC 表示协议控制状态机处于填充计数字段。
0：不处于
1：处于
(RO)

TWAIFD_PC_CRC 表示协议控制状态机处于 CRC 字段。
0：不处于
1：处于
(RO)

见下页...

Register 38.48. TWAIFD_DEBUG_REG (0x008C)

... 接上页

TWAIFD_PC_CRCD 表示协议控制状态机处于 CRC 分隔符字段。

0：不处于

1：处于

(RO)

TWAIFD_PC_ACK 表示协议控制状态机处于 ACK 字段。

0：不处于

1：处于

(RO)

TWAIFD_PC_ACKD 表示协议控制状态机处于 ACK 分隔符字段。

0：不处于

1：处于

(RO)

TWAIFD_PC_EOF 表示协议控制状态机处于文件结束字段。

0：不处于

1：处于

(RO)

TWAIFD_PC_INT 表示协议控制状态机处于间歇字段。

0：不处于

1：处于

(RO)

TWAIFD_PC_SUSP 表示协议控制状态机处于暂停发送字段。

0：不处于

1：处于

(RO)

TWAIFD_PC_OVR 表示协议控制状态机处于过载字段。

0：不处于

1：处于

(RO)

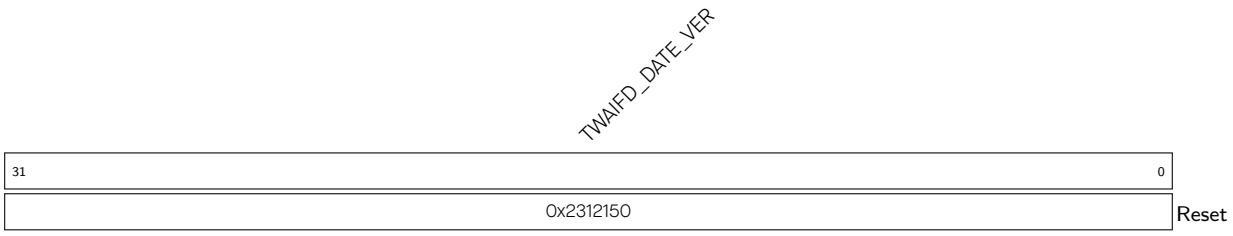
TWAIFD_PC_SOF 表示协议控制状态机处于帧开始字段。

0：不处于

1：处于

(RO)

Register 38.49. TWAIFD_DATE_VER_REG (0x0FFC)



TWAIFD_DATE_VER 版本控制寄存器。(R/W)

第 39 章

SDIO 从机控制器 (SDIO)

39.1 概述

ESP32-C5 支持安全数字输入输出 (SDIO) 设备接口，符合 SDIO 规范 V2.00。主机 (host) 可以通过 SDIO 总线协议访问 ESP32-C5。

主机可以直接访问 ESP32-C5 SDIO 接口寄存器，或通过使用直接存储访问 (DMA) 引擎访问设备上的共享存储器，在保证性能的同时减少了处理器开销。

39.2 主要特性

SDIO 从机控制器具有以下特性：

- 符合 SDIO 物理层规范 V2.00 和 SDIO 规范 V2.00
- 支持 2 个 IO functions（除 function 0 外）
- 支持 SPI、1-bit SDIO 和 4-bit SDIO 传输模式
- 0~50 MHz 时钟范围
- 采样时钟沿或驱动时钟沿可配置
- 为信息交互设定的特定寄存器
- 支持 SDIO 中断机制
- 支持自动填充 SDIO 总线上的发送数据，同样支持自动丢弃 SDIO 总线上的填充数据
- 高达 512 字节的块大小
- 主机与从机间有中断向量可以相互中断对方
- 带有数据传输的 DMA
- 支持在保持连接的状态下进行休眠唤醒

说明：
SDIO 从机控制器在 1-bit SDIO 传输模式下可以与 USB 串口/JTAG 控制器同时使用，但在 4-bit SDIO 传输模式下不能与 USB 串口/JTAG 控制器同时使用。

39.3 结构概览

SDIO 从机的功能结构图如图 39.3-1 所示。

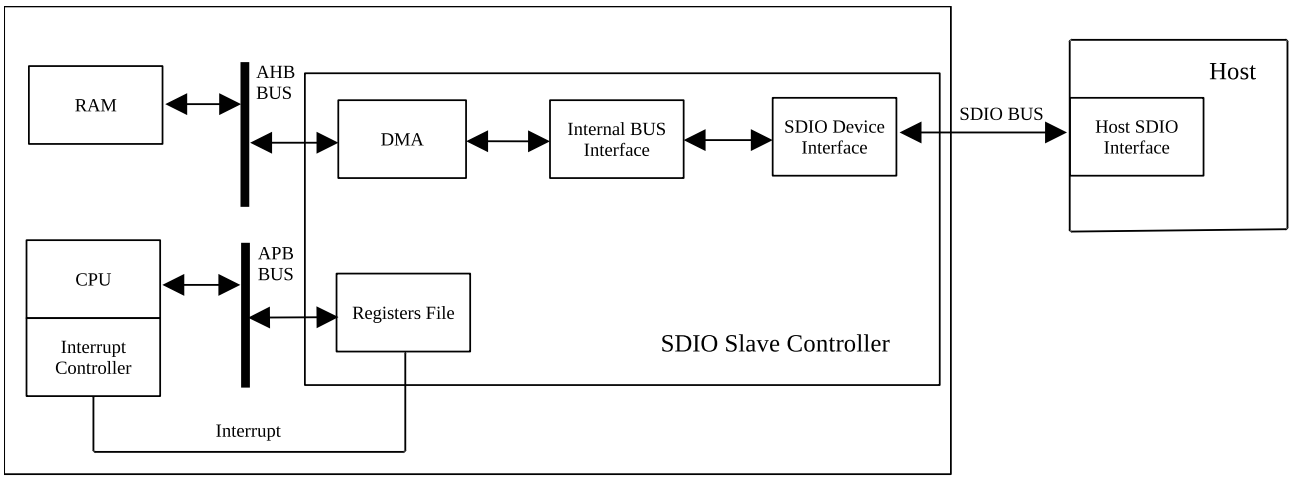


图 39.3-1. SDIO 从机功能结构图

上图中，Host（主机）代表任一符合 SDIO 规范 V2.00 的主机设备。主机通过标准 SDIO 总线与作为 SDIO 从机的 ESP32-C5 互动。SDIO 设备接口模块通过直接提供 SDIO 接口寄存器并使能 DMA 操作来与主机通信，实现高级性能总线 (AHB) 上的高速数据传输，并且不需要 CPU 的参与。

39.4 协议标准

ESP32-C5 SDIO 从机控制器符合以下协议标准：

- 《SD 卡规范第一部分：物理层规范，版本 2.00》（SD Specifications Part 1 Physical Layer Specification Version 2.00，本章中简称为“物理层规范 V2.00”）
- 《SD 卡规范 E1 部分：SDIO 规范，版本 2.00》（SD Specifications Part E1 SDIO Specification Version 2.00, January 30, 2007，本章中简称为“SDIO 规范 V2.00”）

39.5 功能描述

39.5.1 物理总线

- 总线模式：支持 SPI、1-bit SDIO 和 4-bit SDIO 传输模式。
- 总线信号：标准 SDIO 规范 V2.00 的物理总线信号。SPI 传输模式主要包括 CS/DI/SCLK/DO/IRQ 信号；1-bit SDIO 传输模式主要包括 CMD/CLK/DATA/IRQ 信号；4-bit SDIO 传输模式主要包括 CMD/CLK/DAT[3:0]。
- 总线速度模式：全速模式 0~50 MHz 时钟范围；低速模式 0~400 kHz 时钟范围。
- IO functions：除 function 0 外，还支持 2 个 IO function。其中，function 0 只用于 CCCR/FBR/CIS 操作；function 1 和 function 2 可同时用于应用数据传输（如 Wi-Fi 包和蓝牙包）和访问 SLC Host 寄存器。

详见“物理层规范 V2.00”和“SDIO 规范 V2.00”。

39.5.2 支持命令

SDIO 从机控制器支持的主要数据传输命令为 IO_RW_DIRECT (CMD52) 和 IO_RW_EXTENDED (CMD53)。

命令 IO_RW_DIRECT (CMD52) 可以用来访问寄存器和传输数据。其组成如图 39.5-1 所示，各部分具体含义请查看 SDIO 规范 V2.00。

S	D	Command Index 110100b	R/W flag	Function Number	RAW flag	Stuff	Register Address	Stuff	Write Data of Stuff Bits	CRC7	E
1	1	6	1	3	1	1	17	1	8	7	1

图 39.5-1. CMD52 内容

命令 IO_RW_EXTENDED (CMD53) 用来发起任意长度的包传输。其组成如图 39.5-2 所示，各部分具体含义请查看 SDIO 规范 V2.00。

S	D	Command Index 110101b	R/W flag	Function Number	Block Mode 1b	OP Code	Register Address	Byte/Block Count Roundup (Packet_length/Block_Size)	CRC7	E
1	1	6	1	3	1	1	17	9	7	1

图 39.5-2. CMD53 内容

39.5.3 I/O function 0 地址空间

I/O function 0 只用于卡通用控制寄存器 (CCCR) 、功能基本寄存器 (FBR) 、卡信息结构寄存器 (CIS) 的操作，它的地址空间如图 39.5-3 所示。I/O function 0 的地址空间是 SDIO 规范规定的，各部分具体含义请查看 SDIO 规范 V2.00。

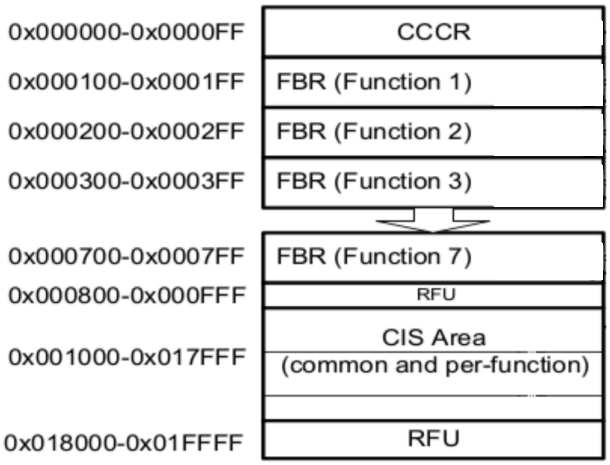


图 39.5-3. function 0 地址空间

CCCR、FBR、CIS 分别是 SDIO 协议规定的通用控制寄存器、各 function 的控制配置寄存器、以及有关版本、

功耗、制造商等卡信息的状态寄存器。这些控制或配置寄存器对应的功能是可选的，其含义详见 SDIO 规范。ESP32-C5 SDIO 从机的 CCCR 设置如表 39.5-1 所示，FBR 设置如表 39.5-2 所示。

表 39.5-1. SDIO 从机的 CCCR 设置

地址	寄存器名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	CCCR/ SDIO Revision	通过 HINF_CFG_DATA1_REG 中的 HINF_SDIO_VER[7:4] 设置读取值 (SDIO bit[3:0])				通过 HINF_CFG_DATA1_REG 中的 HINF_SDIO_VER[3:0] 设置读取值 (CCCR bit[3:0])			
0x01	SD Specification Revision	0 (RFU)				通过 HINF_CFG_DATA1_REG 中的 HINF_SDIO_VER[11:8] 设置读取值 (SD bit[3:0])			
0x02	I/O Enable	0 (IOE[7:3])					读/写 (IOE[2:1])		0 (RFU)
0x03	I/O Ready	0 (IOR[7:3])					读 (IOR[2:1])		0 (RFU)
0x04	Int Enable	0 (IEN[7:3])					读/写 (IEN[2:1])		读/写 (IENM)
0x05	Int Pending	0 (INT[7:3])					读 (INT[2:1])		0 (RFU)
0x06	I/O Abort	0 (RFU)				写 (RES)	写 (AS[2:0])		
0x07	Bus Interface Control	读/写 (CD Disable)	1 (SCSI)	读/写 (ECSI)	0 (RFU)			读/写 (Bus Width[1:0])	
0x08	Card Capability	0 (4BLS)	0 (LSC)	读/写 (E4MI)	1 (S4MI)	0 (SBS)	1 (SRW)	1 (SMB)	1 (SDC)
0x09-0x0B	Common CIS Pointer	Address 0x09: 0x0; Address 0x0A: 0x10; Address 0x0B: 0x0 (Pointer to card's common CIS)							
0x0C	Bus Suspend	0 (RFU)						0 (BR)	0 (BS)
0x0D	Function Select	0 (DF)	0 (RFU)			0 (FS[3:0])			
0x0E	Exec Flags	0 (EX[7:1])							0 (EXM)
0x0F	Ready Flags	0 (RF[7:1])							0 (RFM)
0x10-0x11	FN0 Block Size	读/写 (支持范围 0~512) (I/O block size for Function 0)							
0x12	Power Control	0 (RFU)						读/写 (EMPC)	1 (SMPC)
0x13	High-Speed	0 (RFU)						读/写 (EHS)	Note [a] (SHS)

[a] 通过 HINF_CFG_DATA1_REG 中的 HINF_HIGHSPEED_ENABLE 设置读取值。

表 39.5-2. SDIO 从机的 FBR 设置

地址	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x100	0 (Function 1 CSA enable)	0 (Function 1 supports CSA)	0 (RFU)		0 (Function 1 Standard SDIO Function interface code)			
0x101	0 (Function 1 Extended standard SDIO Function interface code)							
0x102	0 (RFU)						读/写 (EPS)	0 (SPS)
0x109-0x10B	Address 0x109: 0x0; Address 0x10A: 0x11; Address 0x10B: 0x0 (Pointer to Function 1 CIS)							
0x110-0x111	读/写 (支持范围 0~512) (I/O block size for Function 1)							
0x200	0 (Function 2 CSA enable)	0 (Function 2 supports CSA)	0 (RFU)		0x2 (Function 2 Standard SDIO Function interface code)			
0x201	0 (Function 2 Extended standard SDIO Function interface code)							
0x202	0 (RFU)						读/写 (EPS)	0 (SPS)
0x209-0x20B	Address 0x209: 0x0; Address 0x20A: 0x12; Address 0x20B: 0x0 (Pointer to Function 2 CIS)							
0x210-0x211	读/写 (支持范围 0~512) (I/O block size for Function 2)							

39.5.4 I/O function 1/2 地址空间

I/O function 1 和 function 2 的功能与权限完全相同，二者可同时或独立传输应用数据（例如 Wi-Fi 包和蓝牙包），都可以传输固定地址包和递增地址包。function 1 和 function 2 还可以访问同一组 SLC Host 寄存器，它们的地址空间如图 39.5-4 所示。主机可通过 I/O function 1 和 function 2 访问图中的所有地址段。

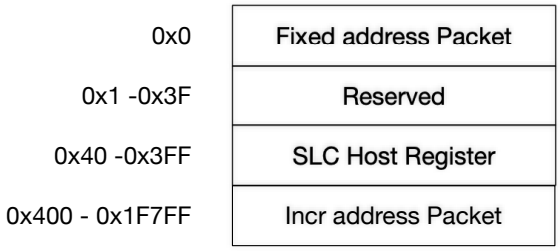


图 39.5-4. function 1/2 地址空间

39.5.4.1 访问 SLC HOST 寄存器空间

为了方便主机与从机之间的信息交互，主机可以通过 I/O function 1/2 访问从机中的特定寄存器。这些寄存器在 0x40 –0x3FF 的连续地址段内，主机访问时只需将 CMD52 或 CMD53 中的寄存器地址设置为对应寄存器地址的低 10 位。主机可以使用 CMD53 同时访问多个寄存器，提高了数据传输的速率。

SLCHOST_CONF_W0_REG 到 SLCHOST_CONF_W15_REG 共有 52 个字节的区域可供主机和从机任意访问和修改，方便了主机与从机之间的信息交互。

主机与从机侧的软件可以同时访问 SLC Host 寄存器空间，因此需要用户设计上层机制保证主机与从机侧的软件不会同时访问相同的寄存器地址，避免两侧同时访问同一寄存器字段而造成的错误。

39.5.4.2 使用递增地址传输包

主机使用地址 0x400 - 0x1F7FF 连续传输多个应用数据包（如 Wi-Fi 包）时，CMD53 中的地址字段应设置为递增模式，OP Code 字段应固定为 1。

例如：主机希望使用 CMD53，从起始地址 0x500 开始，传输 3 个数据块。那么主机应该将 CMD53 中的字段 Block Mode 设为 1，表示以数据块为单位进行数据传输；将字段 OP Code 设为 1，表示递增地址模式；将 Register Address 字段设为 0x500，表示起始地址为 0x500；将字段 Byte/Block Count 设为 0x3，表示传输 3 个数据块；其余字段依据 SDIO 规范设置。

主机与从机间通过 CMD53 传输数据包时，从机会判定当前数据包的有效数据是否结束，从而进行无效数据的填充与丢弃。数据填充与丢弃机制详见 39.5.5.3 所述。

39.5.4.3 使用固定地址传输包

主机使用地址 0x0 传输应用数据包（如蓝牙包）时，CMD53 中的地址字段和 OP Code 字段应固定为 0。

例如：主机希望使用 CMD53，从固定地址 0x0，传输 3 个数据块。那么主机应该将 CMD53 中的字段 Block Mode 设为 1，表示以数据块为单位进行数据传输；将字段 OP Code 设为 0，表示固定地址模式；将 Register Address 字段设为 0x0，表示固定地址为 0x0；将字段 Byte/Block Count 设为 0x3，表示传输 3 个数据块；其余字段依据 SDIO 规范设置。

主机与从机间通过 CMD53 传输数据包时，从机会判定当前数据包的有效数据是否结束，从而进行无效数据的填充与丢弃。数据填充与丢弃机制详见 39.5.5.3 所述。

39.5.5 DMA

SDIO 从机有一个专门的 DMA 用于从系统存储 (RAM) 空间获取或存储传输数据，如图 39.3-1 所示，DMA 通过 AHB 总线访问 RAM。SDIO 从机控制器有权限访问的 RAM 空间详见章节 6 系统和存储器。SDIO 从机控制器一

次传输可以访问的存储地址范围，可以通过配置 SLC 寄存器中的 *SHAREMEM*_REG 相关字段控制，详见 39.8 节所述。

DMA 有 2 个通道，SLC0 和 SLC1，传输固定地址包时，会通过 DMA 的 SLC1 通道完成；传输递增地址包时，会通过 DMA 的 SLC0 通道完成。为了用户使用上的方便，SDIO 从机提供了 function 1/2 用于数据传输，I/O function 1/2 的地址范围详见 39.5.4 节所述。建议通过 function 1 传输递增地址包，使用 DMA 的 SLC0 通道；通过 function 2 传输固定地址包，使用 DMA 的 SLC1 通道。

DMA 通过 AHB 总线访问 RAM，可以通过配置 SDIO_SLCCONF0_REG 和 SDIO_SLC_BURST_LEN_REG 中的相关字段，设置 AHB 接口是否可以使用 Burst 操作，以及可以使用的 Burst 操作类型，详见 39.8 节所述。

39.5.5.1 链表

从机软件可以通过挂载链表的方式来使用 DMA 引擎。DMA 根据 RX（从机到主机）链表中的内容将相应 RAM 中的数据发送出去，也可根据 TX（主机到从机）链表中的内容将接收的数据存入指定 RAM 地址空间。每张链表由若干个描述符组成。

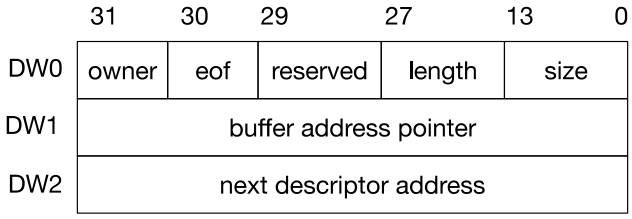


图 39.5-5. SDIO 从机 DMA 链表描述符结构

TX 链表描述符与 RX 链表描述符结构相同，图 39.5-5 所示为描述符的结构图，一个描述符由 3 个字组成。每一字段的意义如下：

- owner (DW0)[31]：表示当前描述符对应的 buffer 允许的操作者。
  0：允许的操作者为 CPU；
  1：允许的操作者为 DMA 引擎。
  从机软件创建描述符时应将该字段置为 1；使能 DMA 回写权限并且 DMA 使用完对应 buffer 后，会将该字段清 0。
- eof (DW0)[30]：表示数据包结束标志。
  0：当前描述符不是数据包的最后一个描述符；
  1：当前描述符为数据包的最后一个描述符。
  主机向从机发送包，从机软件创建描述符时应将该字段置为 0，DMA 会在一个包结束时将对应描述符的该字段置 1；主机从从机接收包，从机软件创建描述符时根据包结束情况进行设置。
- reserved (DW0)[29:28]：reserved。
  从机软件需将该字段设置为 0x0。
- length (DW0)[27:14]：表示当前描述符对应的 buffer 中的有效字节数。DMA 引擎从 buffer 中读取数据时表示能够读取的字节数；DMA 引擎向 buffer 中存储数据时表示已存数据的字节数。
  主机向从机发送包，从机软件创建描述符时应将该字段置为 0x0，DMA 会在对应 buffer 使用完后回写该字段；主机从从机接收包，从机软件创建描述符时应将该字段置为对应 buffer 中能够读取的字节数。
- size (DW0)[13:0]：表示当前描述符对应的 buffer 的大小，单位为字节。
  从机软件创建描述符时设置。
  注意：大小必须字对齐。

- buffer address pointer (DW1)：buffer 地址指针。
  从机软件创建描述符时设置。
  注意：地址必须字对齐。
- next descriptor address (DW2)：下一个描述符的地址。下一个描述符不存在时，该值为 0。
  从机软件创建描述符时设置。

DMA 回写链表描述符字段详见 39.5.5.2 所述。

从机软件可以将多个描述符通过 next descriptor address (DW2) 字段组成链表，SDIO 从机 DMA 链表如图 39.5-6 所示：

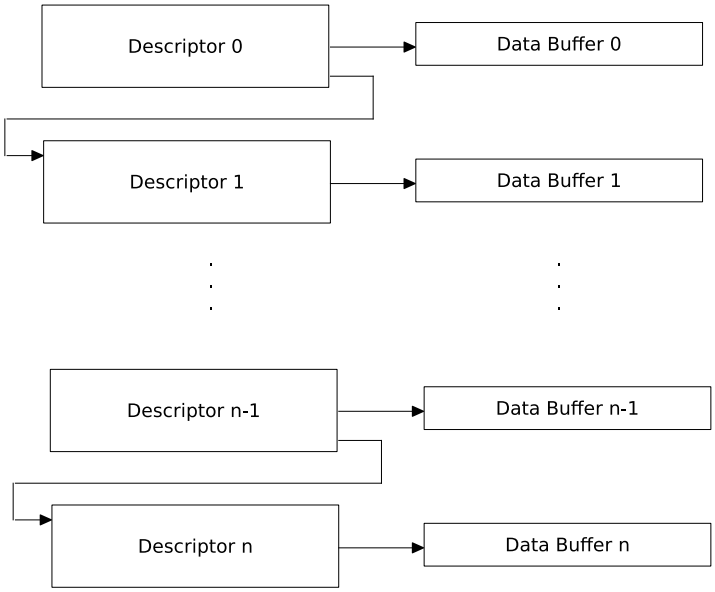


图 39.5-6. SDIO 从机 DMA 链表

为了更清晰的理解链表和 eof 位，举例如下：假设从机侧软件构造了链表，包含 3 个描述符。

- 描述符 0 指向的数据大小为 500 字节，eof 位为 0。
- 描述符 1 指向的数据大小为 200 字节，eof 位为 1。
- 描述符 2 指向的数据大小为 200 字节，eof 位为 1。

1. 若第 1 个 CMD53 命令要求读取 400 字节的数据，则 DMA 会将描述符 0 指向的前 400 字节数据发送给主机。
2. 若第 2 个 CMD53 命令要求读取 400 字节的数据，则 DMA 会先将描述符 0 指向的后 100 字节剩余数据发送给主机；之后再将描述符 1 指向的 200 字节数据发送给主机；之后由于描述符 1 的 eof 位为 1，DMA 判定当前命令的有效数据结束，从而向主机发送 100 字节的无效数据 0x0。
3. 若第 3 个 CMD53 命令要求读取 400 字节的数据，则 DMA 会先将描述符 2 指向的 200 字节数据发送给主机；之后由于描述符 2 的 eof 位为 1，DMA 判定当前命令的有效数据结束，从而向主机发送 200 字节的无效数据 0x0。

39.5.5.2 链表回写

主机向从机发送包的过程中，当一个链表描述符指定的 buffer 被写满，或者一个包传输结束时，DMA 引擎需要跳到下一个描述符来存储后续的数据。在跳转之前，DMA 会对当前的描述符进行回写。DMA 可以回写描述符

的 DW0，将 eof 和 length 位写入最新的值，owner 位的值由 SDIO_SLCCONF0_REG 中的 SDIO_SLC0/1_TX_LOOP_TEST 决定。

主机从从机接收包的过程中，当一个链表描述符指定的 buffer 被读空时，DMA 引擎需要跳到下一个描述符来读取后续的数据。在此之前，从机侧软件可将 SDIO_SLCCONF0_REG 中的 SDIO_SLC0/1_RX_AUTO_WRBACK 置 1，DMA 会对当前的描述符进行回写。可以回写描述符的 DW0，将 owner 位写入最新的值，owner 位的值由 SDIO_SLCCONF0_REG 中的 SDIO_SLC0/1_RX_LOOP_TEST 决定，而其他位的值保持不变。

相关寄存器字段详见 39.8 节所述。

39.5.5.3 数据填充与丢弃

为了实现块传输，主机和从机都需要将 SDIO 总线上发送的数据填充为整个块。从机在发包时会自动填充数据，在收包时自动丢弃填充数据。

- 主机向从机通过 CMD53 发送数据包时，当数据量达到数据包长度时，SDIO 从机判定当前数据包有效数据结束。此时 DMA 会回写当前链表描述符，将当前描述符的 eof 位置 1，并产生 SLC0/1_TX_SUC_EOF_INT 中断。判定有效数据结束后，当前数据包的剩余数据会被认为是无效数据，不再经 DMA 收入 buffer。直至下一个 CMD53，从机才会重新开始将数据接收至下一个 buffer。
  - 对于递增地址包，从机判断有效数据是由地址决定的，当数据地址大于等于 0x1F800 后会被判定为无效数据从而丢弃。因此，主机应将 CMD53 起始地址字段设为 0x1F800 –Packet_length（Packet_length 的单位是字节）。递增地址包在 SDIO 总线上的数据流如图 39.5-7 所示。

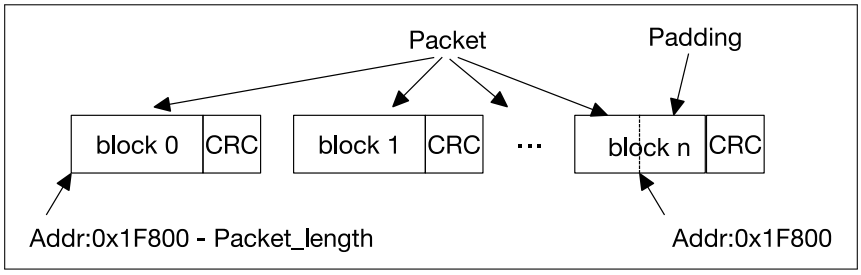


图 39.5-7. 主机向从机发送递增地址包的数据流

  - 对于固定地址包，从机使用数据包的前 3 个字节作为包长度（包长度包括前 3 个字节，并且前 3 个字节也会被存入链表指定的 buffer 中），当收到的数据长度达到包长度后，之后的数据会被认为是无效数据而丢弃。
- 主机通过 CMD53 从从机获取数据包（包括递增地址包和固定地址包）时，当 DMA 读取到某个 buffer 的最后一个字节，且 DMA 链表描述符的 eof 位为 1 时，从机判定当前包的有效数据结束。此时 DMA 会回写当前描述符，并触发 SLC0/1_RX_EOF_INT 中断。判定有效数据结束后，当前数据包的剩余数据会被无效数据 0x0 填充，不再经 DMA 从 buffer 读取。直至主机发送下一个 CMD53，从机才会重新开始经 DMA 从 buffer 读取数据。

  注意：主机从从机获取数据包时，无论递增地址包和固定地址包都使用 DMA 链表描述符的 eof 位作为数据结束的判断依据，而不是地址 0x1F800。因此，主机下发多个 CMD53 获取多个数据包，只要 DMA 没有遇到描述符 eof 位为 1，就会按照链表对应的 buffer 顺序依次获取数据，依次传输给主机；当 DMA 遇到描述符 eof 位为 1 时，会从对应的 buffer 中取完数据，然后填充无效数据完成当前 CMD53 命令，下一个 CMD53 命令将会使用下一个描述符指向的 buffer 数据。

39.5.6 SDIO 总线时序

主机与从机间是通过 PCB 走线连接，所以延迟大。为了保证总线上时序正确，从机支持调整 SDIO 总线输入的采样时钟沿和输出的驱动时钟沿。

当从主机来的数据变化的时刻靠近时钟的上升沿时，从机会选择时钟的下降沿进行采样。采样时序图如图 39.5-8 所示：

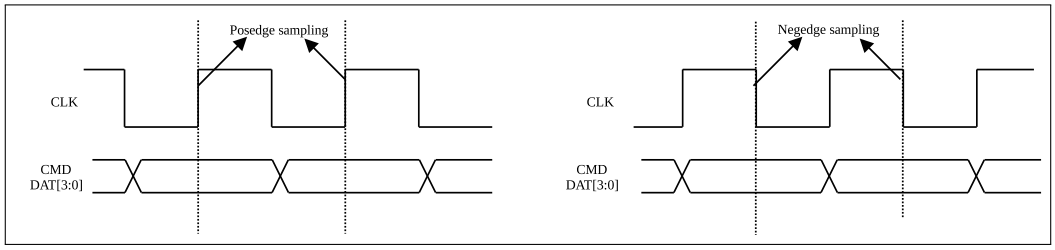


图 39.5-8. 采样时序图

从机引脚采样沿默认由 GPIO25 strapping 引脚电平决定，但通过配置 SLCHOST_CONF_REG 寄存器可以强制决定模式（优先级从高到低）：(1) 置位 SLCHOST_FRC_POS_SAMP，对应引脚强制上升沿采样；(2) 置位 SLCHOST_FRC_NEG_SAMP，对应引脚强制下降沿采样。

SLCHOST_FRC_POS_SAMP 和 SLCHOST_FRC_NEG_SAMP 的位宽均为 5 bit，5 个 bit 分别对应 CMD 线和 4 个 DATA 线 (0-3) 。置位这两个域使得从机对相应的线在时钟上升沿或下降沿上进行采样。

从机还可以选择时钟的上升沿还是下降沿对数据输出线进行驱动，以适应不同的延迟。时序图如图 39.5-9 所示：

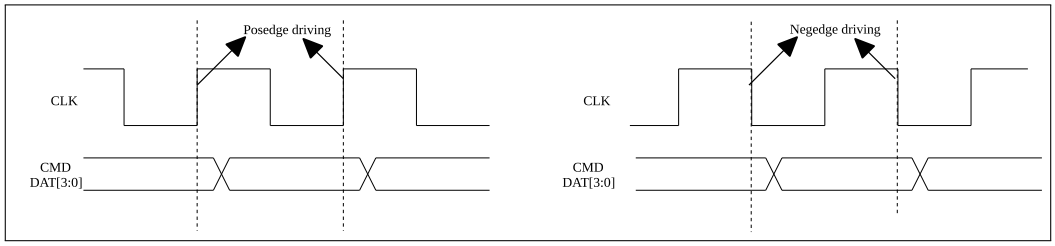


图 39.5-9. 输出时序图

从机引脚输出沿默认由 MTDI strapping 引脚电平决定，但通过配置以下寄存器可以强制决定模式（优先级从高到低）：(1) 置位 SLCHOST_CONF_REG 中的 SLCHOST_FRC_SDIO11，对应引脚强制下降沿输出；(2) 置位 SLCHOST_CONF_REG 中的 SLCHOST_FRC_SDIO22，对应引脚强制上升沿输出；(3) 置位 HINF_CFG_DATA1_REG 中的 HINF_HIGHSPEED_ENABLE 和 SLCHOST_CONF_REG 中的 SLCHOST_HSPEED_CON_EN 后，主机通过配置 CCCR 中的 EHS (Enable High-Speed) 为 1 强制上升沿输出。

SLCHOST_FRC_SDIO11 和 SLCHOST_FRC_SDIO22 的位宽均为 5 bit，5 个 bit 分别对应 CMD 线和 4 个 DATA 线 (0-3) 。置位这两个域使得从机在时钟下降沿或上升沿上对相应的线进行驱动。

关于优先级的说明：无论输入还是输出，strapping 引脚的配置也被包含在优先级范围内（优先级最低）。每一级有效的前提是比它高的优先级都不生效。例如 GPIO25 strapping 的配置决定对应引脚的采样沿，生效的前提是 SCLHOST_FRC_POS_SAMP 没有置 1，SCLHOST_FRC_NEG_SAMP 也没有置 1。

39.6 中断

主机和从机间可以通过配置中断向量灵活地中断对方。主机和从机 DMA 每个 SLC 通道之间各有 8 个中断向量可用于中断对方。在配置中断向量寄存器后就会向对方发送中断（配置相应的中断使能寄存器）。

39.6.1 主机侧中断

- *SLCHOST_SLC0/1_RX_NEW_PACKET_INT*：SDIO Slave 有数据包需要向主机发送。以下任意一种情况可触发该中断：
  - 软件配置 SDIO_SLC0RX_LINK_REG 或 SDIO_SLC1RX_LINK_REG 中的 SDIO_SLC0/1_RXLINK_START 字段启动 DMA 时
  - DMA 处理完 eof 字段为 1 的 RX 链表描述符，之后又有新的 RX 链表描述符时
  - 数据包需要重传时
- *SLCHOST_SLC0/1_TX_OVF_INT*：从机 TX buffer 溢出中断
- *SLCHOST_SLC0/1_RX_UDF_INT*：从机 RX buffer 下溢中断
- *SLCHOST_SLC0/1_TOHOST_BITn_INT*（*n*: 0 ~ 7）：从机中断主机

39.6.2 从机侧中断

- *SLC0/1_RX_DSCR_ERR_INT*：从机 RX 链表描述符错误中断
- *SLC0/1_TX_DSCR_ERR_INT*：从机 TX 链表描述符错误中断
- *SLC0/1_RX_EOF_INT*：从机 RX 操作完成中断
- *SLC0/1_RX_DONE_INT*：单个 buffer 由从机 RX 完成的中断
- *SLC0/1_TX_SUC_EOF_INT*：从机 TX 操作完成中断
- *SLC0/1_TX_DONE_INT*：单个 buffer 在从机 TX 操作时填满了的中断
- *SLC0/1_TX_OVF_INT*：从机 TX buffer 溢出中断
- *SLC0/1_RX_UDF_INT*：从机 RX buffer 下溢中断
- *SLC0/1_TX_START_INT*：从机 TX 操作开始中断
- *SLC0/1_RX_START_INT*：从机 RX 操作开始中断
- *SLC_FRHOST_BITn_INT*（*n*: 0~15）：主机中断从机，其中中断向量 Bit[7:0] 通过 DMA SLC0 通道处理中断，中断向量 Bit[15:8] 通过 DMA SLC1 通道处理中断

39.7 包的发送和接收流程

SDIO 主机与从机间的包传输需要两者按照特定的流程配合完成。在基于 SDIO 规范协议的基础上，ESP32-C5 针对 Wi-Fi/蓝牙等更高抽象层次的数据传输应用场景，需要额外遵循如下特定的数据传输过程。

39.7.1 从机向主机发送包

包的发送是由从机发起，通过中断来通知主机（中断实现方式参看 SDIO 协议）。主机从从机读取相关信息后发起 SDIO 总线传输。整个流程如图 39.7-1 所示：

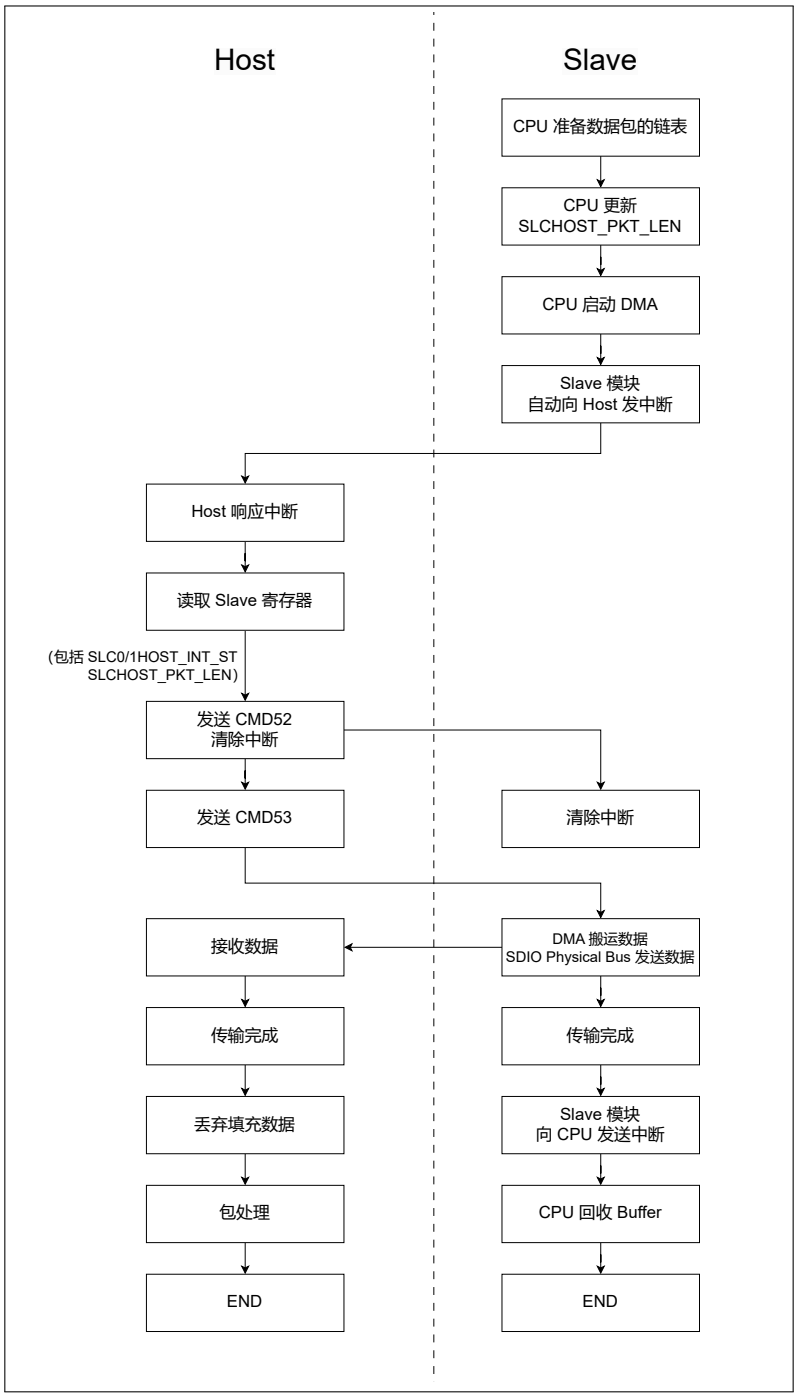


图 39.7-1. 从机向主机发送包的流程

1. 从机 CPU 根据传输数据包的具体情况，构建链表。更多关于构建链表的信息请参考 39.5.5.1 小节。
2. 从机 CPU 通过寄存器 SDIO_SLCO_LEN_CONF_REG 向主机更新要发送的数据长度信息。
3. 从机 CPU 启动 DMA。需要先将链表第一个描述符的 32 位地址写到寄存器 SDIO_SLCORX_LINK_ADDR_REG 或 SDIO_SLC1RX_LINK_ADDR_REG 中，再配置 SDIO_SLCORX_LINK_REG 或 SDIO_SLC1RX_LINK_REG 的 SDIO_SLCO/1_RXLINK_START 字段来启动

DMA。更多关于 DMA 的信请参考 39.5.5 小节。

4. 从机 DMA 向主机发送中断。
5. 主机收到中断后，通过访问 SLCHOST_SLC0HOST_INT_ST_REG、SLCHOST_SLC1HOST_INT_ST_REG 和 SLCHOST_PKT_LEN_REG 这 3 个寄存器从从机读取相关信息。
   - SLCHOST_SLC0/1HOST_INT_ST_REG：中断状态寄存器，其中 bit SLCHOST_SLC0/1_RX_NEW_PACKET_INT_ST 为 1 表明中断的原因是从机发包。
   - SLCHOST_PKT_LEN_REG：从机发送包长度累加寄存器，主机用本次读取值减去上次读取值就可以得到本次发送包的长度。
6. 主机通过 CMD52 命令写到寄存器 SLCHOST_SLC0/1HOST_INT_CLR_REG 清除中断。
7. 主机通过 CMD53 命令从从机处获取数据包。数据传输过程中，当从机判定当前包的有效数据结束时，之后的数据会使用无效数据 0x0 填充。关于从机如何判定有效数据结束的信息，请参考 39.5.5.3 小节。
8. 数据包发送完成后，从机 DMA 会向 CPU 发送中断，这时 CPU 可以回收 buffer。

注意：

- 不建议链表中 eof 位全设置为 0。如果链表中没有设置 eof 位为 1，DMA 可能会将下一个包的数据发送给当前命令，造成错误。若链表不含 eof 位为 1 的描述符，Slave 侧软件应该负责填充数据，将每个数据包的长度对齐到数据块的大小，避免 DMA 将下一个包的数据发送给当前命令。Host 发送 CMD53 读取数据时应该准确控制每个包的数据块个数，避免多取或少取数据，同时 Host 应该有识别填充数据的能力。
- 建议用户一个 CMD53 命令只传输一个数据包，每个数据包只使用一个链表，避免复杂传输造成未知异常。
- 不建议通过一个链表发送多个数据包，Host 及软件可能很难对数据包进行分割。如果软件希望通过一个链表发送多个数据包，应该在构造链表时通过 eof 字段对数据包进行分割，方便 DMA 根据链表对不同的数据包做数据填充（建议每个数据包的长度都要对齐到数据块的大小，可以避免 DMA 填充数据），同时 Host 应该有识别填充数据的能力。

39.7.2 从机从主机接收包

包的接收是由主机发起，从机通过 DMA 接收数据并存储到 RAM 中，传输完成后通过中断通知 CPU 进行数据处理。整个流程如图 39.7-2 所示：

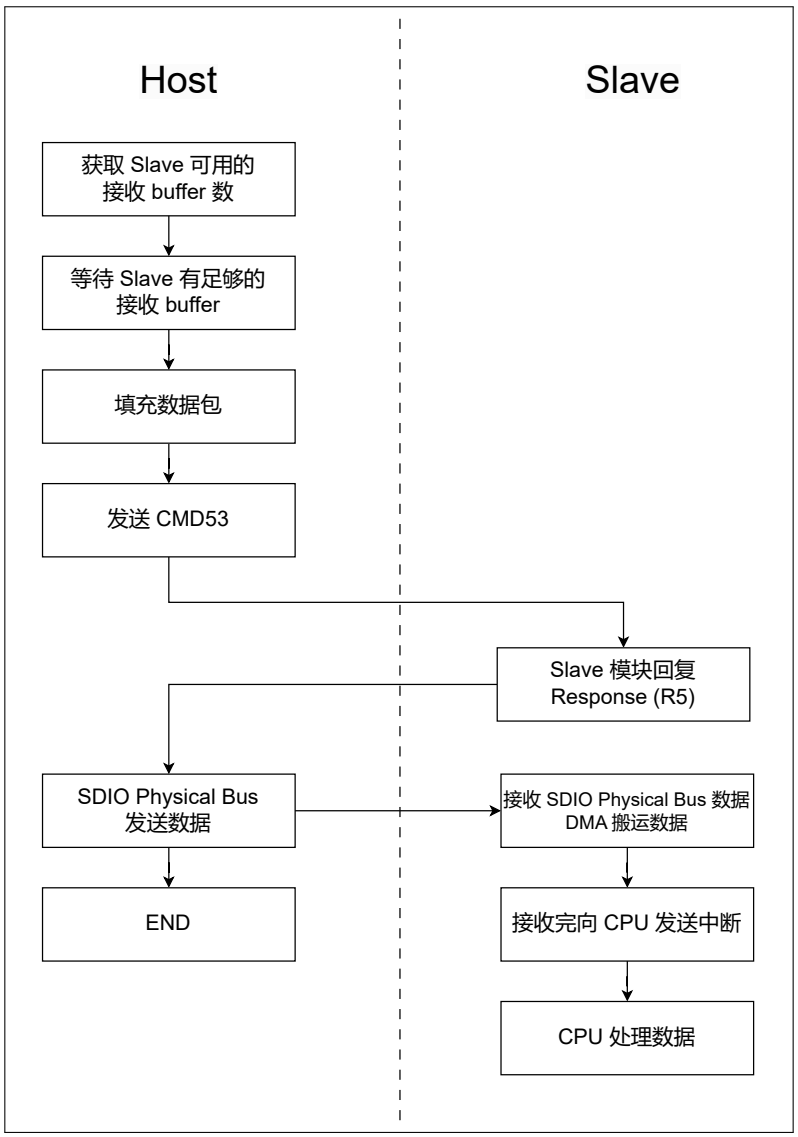


图 39.7-2. 从机从主机接收包的流程

主机通过访问寄存器 SLCHOST_SLCOHOST_TOKEN_RDATA_REG 或 SLCHOST_SLC1HOST_TOKEN_RDATA_REG 获取从机侧可用于接收包的 buffer 数。从机侧的 CPU 要在准备好用于接收包的 DMA 链表后更新这个值。

SLCHOST_HOSTSLCHOST_SLCO_TOKEN1 或 SLCHOST_HOSTSLCHOST_SLC1_TOKEN1 存放可用 buffer 的累计值。主机通过该字段的值减去已用掉的 buffer 数就得到了当前可用的 buffer 数。当 buffer 不够用时，主机需要通过不停地访问该寄存器直到 buffer 够用为止。

主机通过 CMD53 命令向从机发送数据包的过程中，当一个 DMA 链表描述符指定的区域被写满，或者一个包传输结束时，DMA 便会跳到下一个描述符来存储后续的数据。从机判定当前数据包有效数据结束时，会丢弃当前包的无效填充数据，DMA 会将当前的描述符 eof 位回写为 1，并产生 SLCO/1_TX_SUC_EOF_INT中断。

DMA 功能、链表、以及数据丢弃详见 39.5.5 小节所述。

为了保证有充足的 buffer 来接收包，从机的 CPU 必须不停地在接收链表上挂载 buffer。挂载的流程如图 39.7-3 所示。

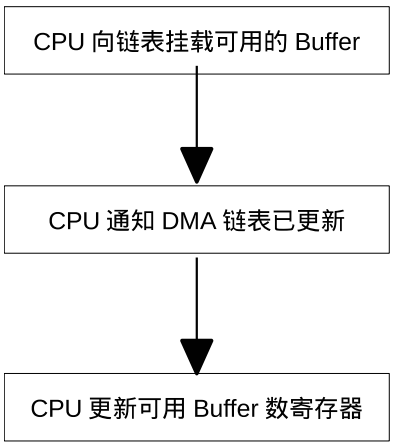


图 39.7-3. 从机 CPU 挂载 buffer 的流程

CPU 首先需要将新的 buffer 组成链表并且挂载在 DMA 正在使用的链表结尾。

然后，CPU 需要通知 DMA 链表已更新，可以通过置位 SDIO_SLCO_TXLINK_RESTART 或 SDIO_SLC1_TXLINK_RESTART 字段来实现。注意，CPU 在首次启动 DMA 用于收包时需置位 SDIO_SLCO_TXLINK_START 或 SDIO_SLC1_TXLINK_START 字段为 1。

RESTART 字段说明：DMA RESTART 只适用于 2 种情况：

- 配置相关寄存器中的 *_STOP 字段暂停了 DMA 的工作，之后可以通过配置 *_RESTART 字段使 DMA 继续工作
- 因为缺乏链表描述符导致 DMA 暂停工作，补充描述符后可以通过配置 *_RESTART 字段使 DMA 继续工作。

最后，CPU 通过配置 SDIO_SLCOTOKEN1_REG 或 SDIO_SLC1TOKEN1_REG 寄存器更新可用的 buffer。

39.8 寄存器列表

本小节的所有地址均为相对于 SDIO 从机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

39.8.1 HINF 寄存器列表

名称	描述	地址偏移	访问
配置寄存器			
HINF_CFG_DATA0_REG	SDIO CIS 配置	0x0000	R/W
HINF_CFG_DATA1_REG	SDIO 配置	0x0004	R/W
HINF_CFG_DATA7_REG	SDIO 配置	0x001C	varies
HINF_CIS_CONF_W*n*_REG(*n*: 0-7)	SDIO CIS 配置	0x0020+0x4**n*	R/W
HINF_CFG_DATA16_REG	SDIO CIS 配置	0x0040	R/W
状态寄存器			
HINF_CONF_STATUS_REG	SDIO CIS function 0 config0 状态	0x0054	RO

39.8.2 SLC 寄存器列表

名称	描述	地址	访问
配置寄存器			
SDIO_SLCCONF0_REG	DMA 配置	0x0000	R/W
SDIO_SLC0RX_LINK_REG	SCL0 RX 链表配置	0x003C	varies
SDIO_SLC0RX_LINK_ADDR_REG	SCL0 RX 链表地址	0x0040	R/W
SDIO_SLC0TX_LINK_REG	SCL0 TX 链表配置	0x0044	varies
SDIO_SLC0TX_LINK_ADDR_REG	SCL0 RX 链表地址	0x0048	R/W
SDIO_SLC1RX_LINK_REG	SCL1 RX 链表配置	0x004C	varies
SDIO_SLC1RX_LINK_ADDR_REG	SCL1 RX 链表地址	0x0050	R/W
SDIO_SLC1TX_LINK_REG	SCL1 TX 链表配置	0x0054	varies
SDIO_SLC1TX_LINK_ADDR_REG	SCL1 TX 链表地址	0x0058	R/W
SDIO_SLC0TOKEN1_REG	SLC0 接收 buffer 配置	0x0064	varies
SDIO_SLC1TOKEN1_REG	SLC1 接收 buffer 配置	0x006C	varies
SDIO_SLCCONF1_REG	DMA 配置	0x0070	R/W
SDIO_SLC_RX_DSCR_CONF_REG	DMA slave 到主机配置寄存器	0x00A8	R/W
SDIO_SLC0_LEN_CONF_REG	传输包长度控制	0x00F4	varies
SDIO_SLC0_TX_SHAREMEM_START_REG	SLC0 AHB TX 地址范围起始边界	0x0154	R/W
SDIO_SLC0_TX_SHAREMEM_END_REG	SLC0 AHB TX 地址范围结束边界	0x0158	R/W
SDIO_SLC0_RX_SHAREMEM_START_REG	SLC0 AHB RX 地址范围起始边界	0x015C	R/W
SDIO_SLC0_RX_SHAREMEM_END_REG	SLC0 AHB RX 地址范围结束边界	0x0160	R/W
SDIO_SLC1_TX_SHAREMEM_START_REG	SLC1 AHB TX 地址范围起始边界	0x0164	R/W
SDIO_SLC1_TX_SHAREMEM_END_REG	SLC1 AHB TX 地址范围结束边界	0x0168	R/W
SDIO_SLC1_RX_SHAREMEM_START_REG	SLC1 AHB RX 地址范围起始边界	0x016C	R/W

名称	描述	地址	访问
SDIO_SLC1_RX_SHAREMEM_END_REG	SLC1 AHB RX 地址范围结束边界	0x0170	R/W
SDIO_SLC_BURST_LEN_REG	DMA AHB burst type 配置	0x017C	R/W
中断寄存器			
SDIO_SLCOINT_RAW_REG	SLC0 到从机原始中断状态	0x0004	varies
SDIO_SLCOINT_ST_REG	SLC0 到从机屏蔽中断状态	0x0008	RO
SDIO_SLCOINT_ENA_REG	SLC0 到从机中断使能	0x000C	R/W
SDIO_SLCOINT_CLR_REG	SLC0 到从机中断清除	0x0010	WT
SDIO_SLC1INT_RAW_REG	SLC1 到从机原始中断状态	0x0014	varies
SDIO_SLC1INT_CLR_REG	SLC1 到从机中断清除	0x0020	WT
SDIO_SLCINTVEC_TOHOST_REG	从机到主机中断向量设置	0x005C	WT
SDIO_SLC1INT_ST1_REG	SLC1 到从机屏蔽中断状态	0x014C	RO
SDIO_SLC1INT_ENA1_REG	SLC1 到从机中断使能	0x0150	R/W
状态寄存器			
SDIO_SLC0_LENGTH_REG	传输包长度	0x00F8	RO

39.8.3 SLC Host 寄存器列表

名称	描述	地址	访问
配置寄存器			
SLCHOST_CONF_REG	沿配置	0x01F0	R/W
中断寄存器			
SLCHOST_SLC0HOST_INT_RAW_REG	SLC0 中断主机的原始中断状态	0x0050	varies
SLCHOST_SLC1HOST_INT_RAW_REG	SLC1 中断主机的原始中断状态	0x0054	varies
SLCHOST_SLC0HOST_INT_ST_REG	SLC0 中断主机的屏蔽中断状态	0x0058	RO
SLCHOST_SLC1HOST_INT_ST_REG	SLC1 中断主机的屏蔽中断状态	0x005C	RO
SLCHOST_CONF_W7_REG	主机中断从机的中断向量集	0x008C	R/W
SLCHOST_SLC0HOST_INT_CLR_REG	SLC0 中断主机的中断清除	0x00D4	WT
SLCHOST_SLC1HOST_INT_CLR_REG	SLC1 中断主机的中断清除	0x00D8	WT
SLCHOST_SLC0HOST_FUNC1_INT_ENA_REG	SLC0 中断主机的中断使能	0x00DC	R/W
SLCHOST_SLC1HOST_FUNC1_INT_ENA_REG	SLC1 中断主机的中断清除	0x00E0	R/W
状态寄存器			
SLCHOST_SLC0HOST_TOKEN_RDATA_REG	SLC0 接收 buffer 的累计值	0x0044	RO
SLCHOST_PKT_LEN_REG	传输数据包的长度	0x0060	RO
SLCHOST_SLC1HOST_TOKEN_RDATA_REG	SCL1 接收 buffer 的累计值	0x00C4	RO
通信寄存器			
SLCHOST_CONF_W*n*_REG(*n*: 0-2)	主机从机通信寄存器	0x006C+0x4**n*	R/W
SLCHOST_CONF_W3_REG	主机从机通信寄存器	0x0078	R/W
SLCHOST_CONF_W4_REG	主机从机通信寄存器	0x007C	R/W
SLCHOST_CONF_W6_REG	主机从机通信寄存器	0x0088	R/W
SLCHOST_CONF_W*n*_REG(*n*: 8-15)	主机从机通信寄存器	0x009C+0x4*(*n*-8)	R/W

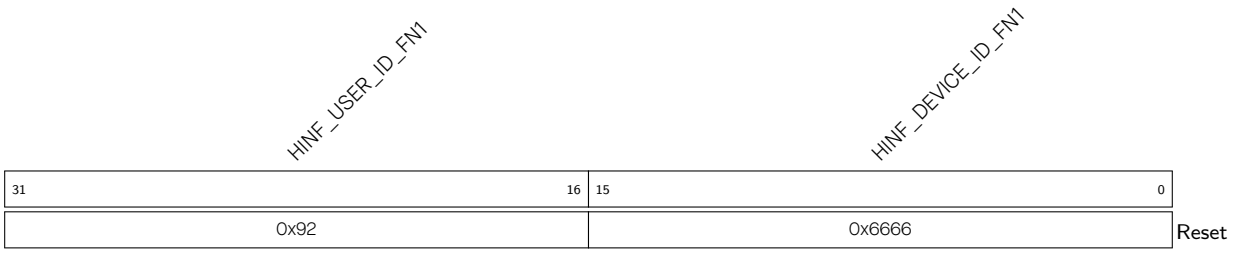
39.9 寄存器

本小节的所有地址均为相对于 SDIO 从机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

39.9.1 HINF 寄存器

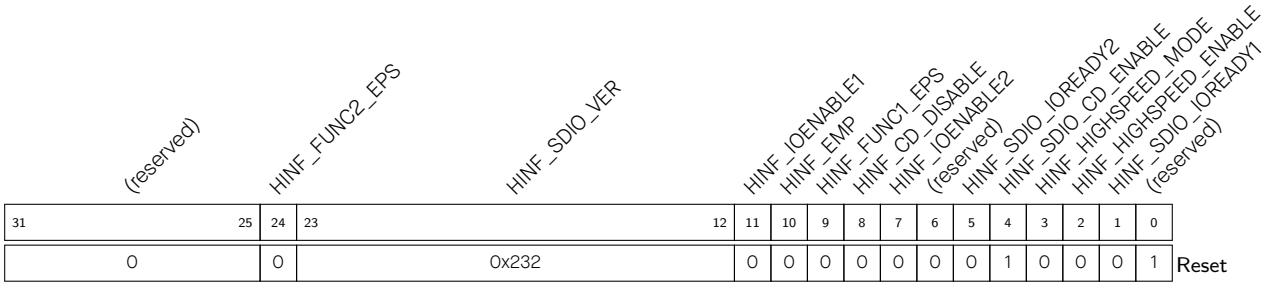
Register 39.1. HINF_CFG_DATA0_REG (0x0000)



HINF_DEVICE_ID_FN1 配置 function 1 在 SDIO CIS 寄存器中 device ID 字段的值。(R/W)

HINF_USER_ID_FN1 配置 function 1 在 SDIO CIS 寄存器中 user ID 字段的值。(R/W)

Register 39.2. HINF_CFG_DATA1_REG (0x0004)



HINF_SDIO_IOREADY1 配置 SDIO CCCR 寄存器中 IOR1 字段，以及 CIS 寄存器中 function 1 ready 字段的值。

0：function 1 未准备就绪

1：function 1 已准备就绪

详情请参考 SDIO 规范。

(R/W)

HINF_HIGHSPEED_ENABLE 配置是否支持 SDIO CCCR 寄存器中 SHS 字段的值。

0：不支持高速模式

1：支持高速模式

详情请参考 SDIO 规范。

(R/W)

HINF_HIGHSPEED_MODE 表示 SDIO CCCR 寄存器中 EHS 字段的值。

0：未使能

1：使能

详情请参考 SDIO 规范。

(RO)

HINF_SDIO_CD_ENABLE 配置是否使能 SDIO 卡检测。

0：不使能

1：使能

(R/W)

HINF_SDIO_IOREADY2 配置 CCCR 寄存器中的 IOR2 字段，以及 CIS 寄存器中的 function 2 ready 字段的值。

0：function 2 未准备就绪

1：function 2 已准备就绪

详情请参考 SDIO 规范。

(R/W)

HINF_IOENABLE2 表示 SDIO CCCR 寄存器中的 IOE2 是否使能。

0：function 2 未使能

1：function 2 已使能

详情请参考 SDIO 规范。

(RO)

见下页……

Register 39.2. HINF_CFG_DATA1_REG (0x0004)

接上页……

HINF_CD_DISABLE 表示 SDIO CCCR 寄存器中的 CD 是否禁用。

0：未禁用

1：禁用

详情请参考 SDIO 规范。

(RO)

HINF_FUNC1_EPS 表示 SDIO FBR 寄存器中 function 1 的 EPS 字段的值。

0：function 1 运行在高电流模式下

1：function 1 运行在低电流模式下

详情请参考 SDIO 规范。

(RO)

HINF_EMP 表示 SDIO CCCR 寄存器中 EMPC 字段的值。

0：主机功耗控制未使能

1：主机功耗控制已使能

详情请参考 SDIO 规范。

(RO)

HINF_IOENABLE1 表示 SDIO CCCR 寄存器中 IOE1 字段的值。

0：function 1 未使能

1：function 1 已使能

详情请参考 SDIO 规范。

(RO)

HINF_SDIO_VER 配置 SDIO CCCR 中 SD bit[3:0]、SDIO bit[3:0]、CCCR bit[3:0] 的值。

HINF_SDIO_VER[11:8] 映射至 SD bit[3:0]

HINF_SDIO_VER[7:4] 映射至 SDIO bit[3:0]

HINF_SDIO_VER[3:0] 映射至 CCCR bit[3:0]

详情请参考 SDIO 规范。

(R/W)

HINF_FUNC2_EPS 表示 SDIO FBR 寄存器中 function 2 的 EPS 字段的值。

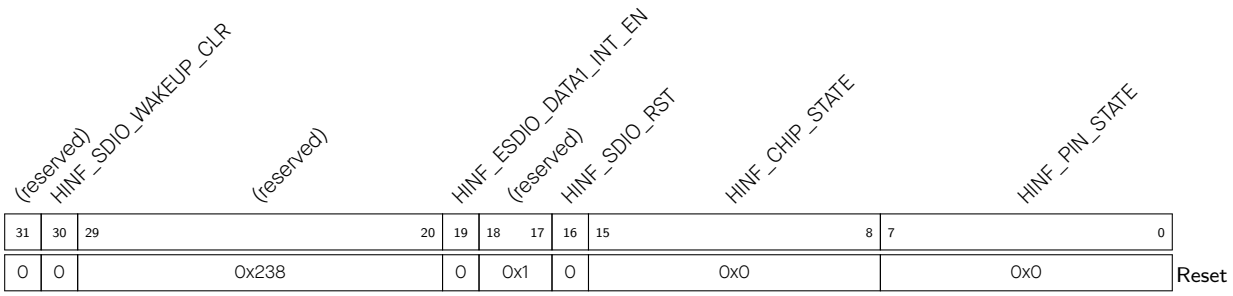
0：function 2 运行在高电流模式下

1：function 2 运行在低电流模式下

详情请参考 SDIO 规范。

(RO)

Register 39.3. HINF_CFG_DATA7_REG (0x001C)



HINF_PIN_STATE 配置 SDIO CIS 寄存器地址 318 和 574 的值，详情请参考 SDIO 规范。(R/W)

HINF_CHIP_STATE 配置 SDIO CIS 寄存器地址 312、315、568、571 的值，详情请参考 SDIO 规范。(R/W)

HINF_SDIO_RST 配置是否重置 SDIO 从机模块。

0：没有作用

1：重置

(R/W)

HINF_ESDIO_DATA1_INT_EN 配置是否使能在 data1 线上的 SDIO 中断。

0：不使能

1：使能

(R/W)

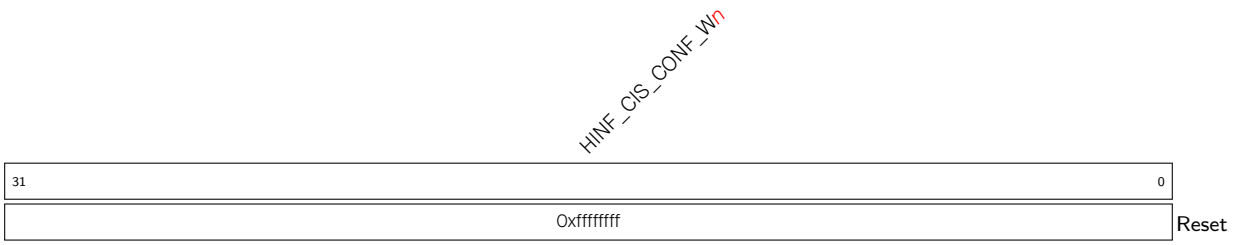
HINF_SDIO_WAKEUP_CLR 配置是否在 SDIO 从机模块唤醒芯片系统后清除唤醒信号。

0：没有作用

1：清除

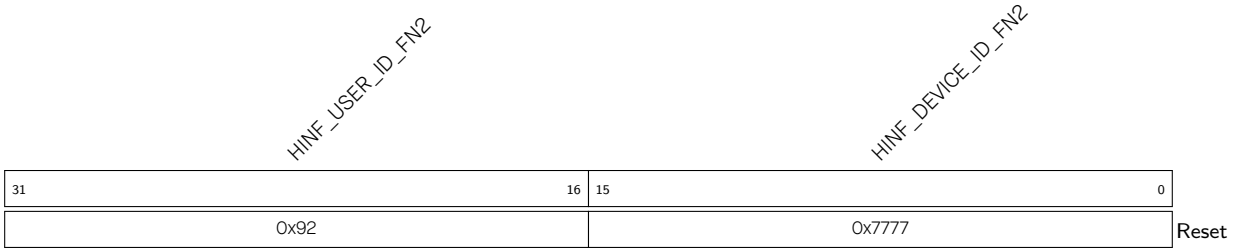
(WT)

Register 39.4. HINF_CIS_CONF_W*n*_REG(*n*: 0-7) (0x0020+0x4**n*)



HINF_CIS_CONF_W*n* 配置 SDIO CIS 寄存器地址 (39+4***n*) ~ (36+4***n*) 的值，详情请参考 SDIO 规范。(R/W)

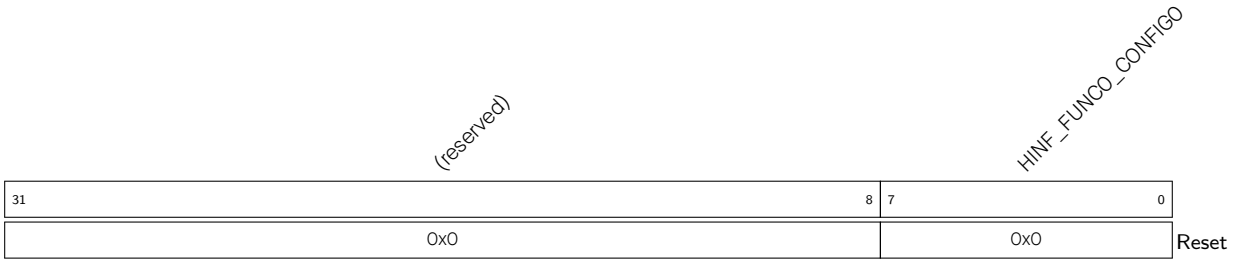
Register 39.5. HINF_CFG_DATA16_REG (0x0040)



HINF_DEVICE_ID_FN2 配置 function 2 在 SDIO CIS 寄存器中 device ID 字段的值。(R/W)

HINF_USER_ID_FN2 配置 function 2 在 SDIO CIS 寄存器中 user ID 字段的值。(R/W)

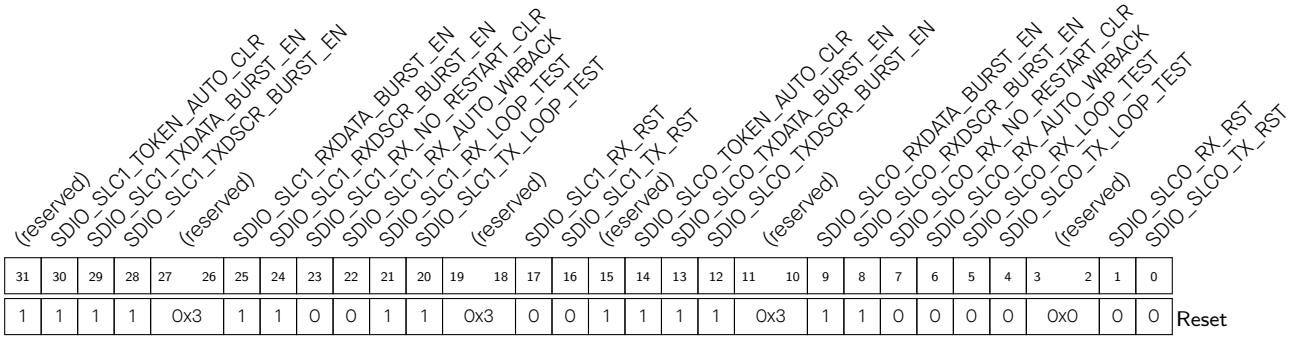
Register 39.6. HINF_CONF_STATUS_REG (0x0054)



HINF_FUNC0_CONFIG0 表示 SDIO CIS 寄存器 function 0 config0 (addr: 0x20f0) 字段状态，详情请参考 SDIO 规范。(RO)

39.9.2 SLC 寄存器

Register 39.7. SDIO_SLCCONF0_REG (0x0000)



SDIO_SLCO_TX_RST 配置是否重置 SLCO TX（主机至从机）有限状态机 (FSM)。

0：没有作用

1：重置

(R/W)

SDIO_SLCO_RX_RST 配置是否重置 SLCO RX（从机至主机）FSM。

0：没有作用

1：重置

(R/W)

SDIO_SLCO_TX_LOOP_TEST 配置当从机 buffer 从主机接收完数据包时，SLCO 是否进行循环。

0：不循环

1：循环，且硬件不改变链表中的 owner 位

(R/W)

SDIO_SLCO_RX_LOOP_TEST 配置当从机 buffer 向主机发送完数据包时，SLCO 是否进行循环。

0：不循环

1：循环，且硬件不改变链表中的 owner 位

(R/W)

SDIO_SLCO_RX_AUTO_WRBACK 配置 SCLO 是否回写 RX 链表的的 owner 位。

0：不回写

1：回写

(R/W)

SDIO_SLCO_RX_NO_RESTART_CLR 请初始化为 1，且不要修改。(R/W)

SDIO_SLCO_RXDSCR_BURST_EN 配置从存储器读取 RX 链表时，SLCO 是否允许使用 AHB burst 操作。

0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

见下页……

Register 39.7. SDIO_SLCCONF0_REG (0x0000)

接上页……

SDIO_SLC0_RXDATA_BURST_EN 配置从存储器读取数据时，SLC0 是否允许使用 AHB burst 操作。

0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

SDIO_SLC0_TXDSCR_BURST_EN 配置从存储器读取 TX 链表时，SLC0 是否允许使用 AHB burst 操作。

0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

SDIO_SLC0_TXDATA_BURST_EN 配置向存储器发送数据时，SLC0 是否允许使用 AHB burst 操作。

0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

SDIO_SLC0_TOKEN_AUTO_CLR 请初始化为 0，且不要修改。(R/W)

SDIO_SLC1_TX_RST 配置是否重置 SCL1 TX FSM。

0：没有作用

0：重置

(R/W)

SDIO_SLC1_RX_RST 配置是否重置 SCL1 RX FSM。

0：没有作用

0：重置

(R/W)

SDIO_SLC1_TX_LOOP_TEST 配置当从机 buffer 从主机接收完数据包时，SLC1 是否进行循环。

0：不循环

1：循环，且硬件不改变链表中的 owner 位

(R/W)

SDIO_SLC1_RX_LOOP_TEST 配置当从机 buffer 向主机发送完数据包时，SLC1 是否进行循环。

0：不循环

1：循环，且硬件不改变链表中的 owner 位

(R/W)

见下页……

Register 39.7. SDIO_SLCCONF0_REG (0x0000)

接上页……

SDIO_SLC1_RX_AUTO_WRBACK 配置 SCL1 是否回写 RX 链表的的 owner 位。

0：不回写

1：回写

(R/W)

SDIO_SLC1_RX_NO_RESTART_CLR 请初始化为 1，且不要修改。(R/W)

SDIO_SLC1_RXDSCR_BURST_EN 配置从存储器读取 RX 链表时，SLC1 是否允许使用 AHB burst 操作。

0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_RXDATA_BURST_EN 配置从存储器读取数据时，SLC1 是否允许使用 AHB burst 操作。

0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_TXDSCR_BURST_EN 配置从存储器读取 TX 链表时，SLC1 是否允许使用 AHB burst 操作。

0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_TXDATA_BURST_EN 配置向存储器发送数据时，SLC1 是否允许使用 AHB burst 操作。

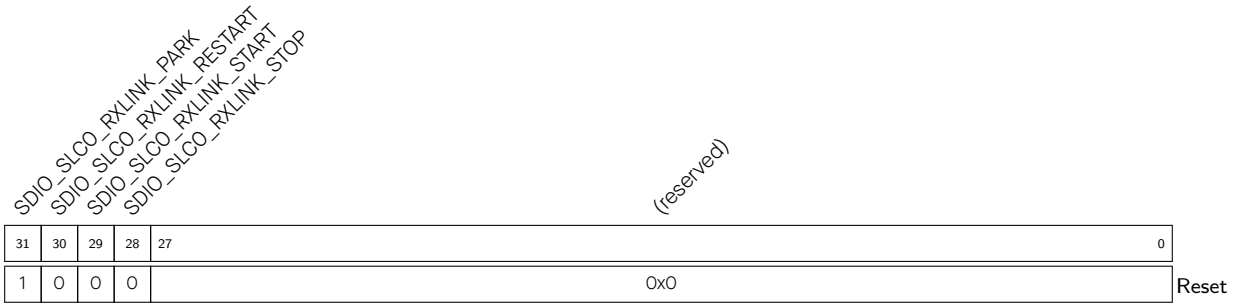
0：仅使用单次 (single) 操作

1：允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_TOKEN_AUTO_CLR 请初始化为 0，且不要修改。(R/W)

Register 39.8. SDIO_SLC0RX_LINK_REG (0x003C)



SDIO_SLC0_RXLINK_STOP 配置是否停止 SLC0 RX 链表操作。

0：没有作用

1：停止操作

(R/W/SC)

SDIO_SLC0_RXLINK_START 配置是否从 SDIO_SLC0_RXLINK_ADDR 指定的地址开始 SLC0 RX 链表操作。

0：没有作用

1：开始操作

(R/W/SC)

SDIO_SLC0_RXLINK_RESTART 配置是否重新启动并继续执行 SLC0 RX 链表操作。

0：没有作用

1：重新启动操作

(R/W/SC)

SDIO_SLC0_RXLINK_PARK 表示 SLC0 RX 链表 FSM 状态。

0：FSM 不处于空闲状态

1：FSM 处于空闲状态

(RO)

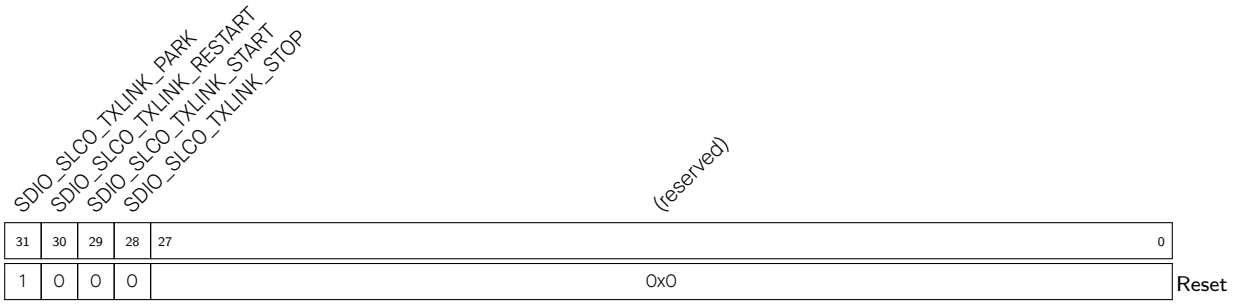
Register 39.9. SDIO_SLC0RX_LINK_ADDR_REG (0x0040)



SDIO_SLC0_RXLINK_ADDR 表示 SLC0 RX 链表初始地址。

(R/W)

Register 39.10. SDIO_SLC0TX_LINK_REG (0x0044)



SDIO_SLC0_TXLINK_STOP 配置是否停止 SLC0 TX 链表操作。

0：没有作用

1：停止操作

(R/W/SC)

SDIO_SLC0_TXLINK_START 配置是否从 SDIO_SLC0_TXLINK_ADDR 指定的地址开始 SLC0 TX 链表操作。

0：没有作用

1：开始操作

(R/W/SC)

SDIO_SLC0_TXLINK_RESTART 配置是否重新启动并继续执行 SLC0 TX 链表操作。

0：没有作用

1：重新启动操作

(R/W/SC)

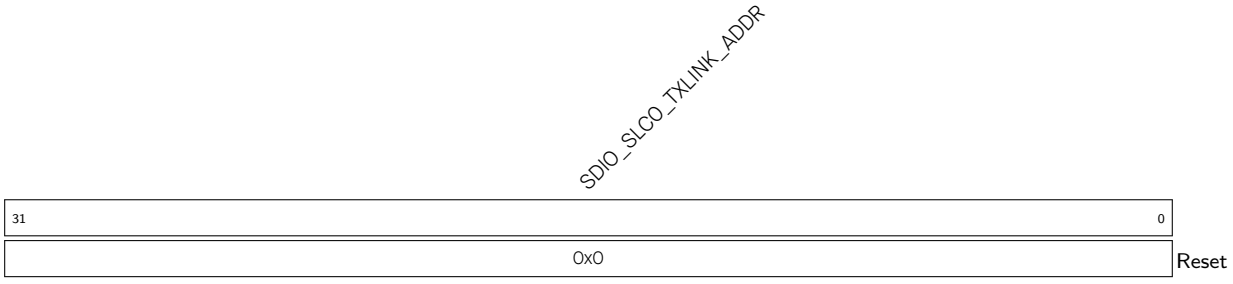
SDIO_SLC0_TXLINK_PARK 表示 SLC0 TX 链表 FSM 状态。

0：FSM 不处于空闲状态

1：FSM 处于空闲状态

(RO)

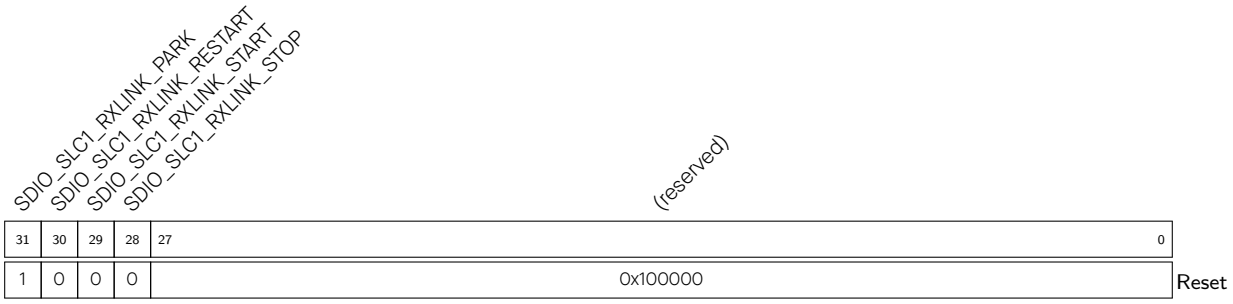
Register 39.11. SDIO_SLC0TX_LINK_ADDR_REG (0x0048)



SDIO_SLC0_TXLINK_ADDR 表示 SLC0 TX 链表初始地址。

(R/W)

Register 39.12. SDIO_SLC1RX_LINK_REG (0x004C)



SDIO_SLC1_RXLINK_STOP 配置是否停止 SLC1 RX 链表操作。

0：没有作用

1：停止操作

(R/W/SC)

SDIO_SLC1_RXLINK_START 配置是否从 SDIO_SLC1_RXLINK_ADDR 指定的地址开始 SLC1 RX 链表操作。

0：没有作用

1：开始操作

(R/W/SC)

SDIO_SLC1_RXLINK_RESTART 配置是否重新启动并继续执行 SLC1 RX 链表操作。

0：没有作用

1：重新启动操作

(R/W/SC)

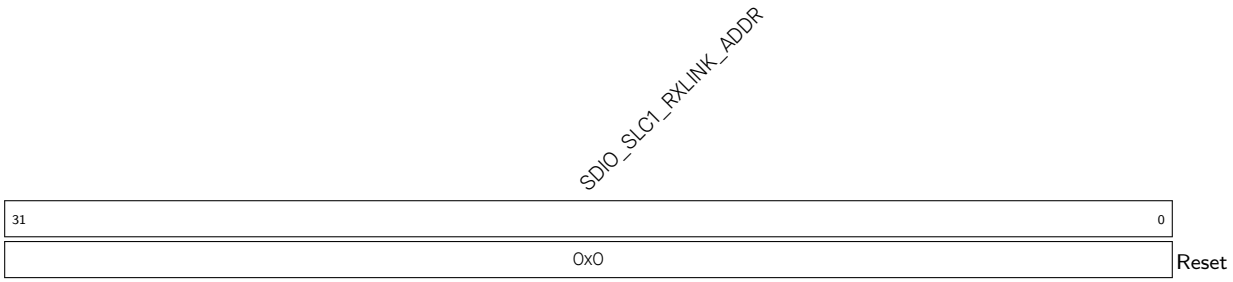
SDIO_SLC1_RXLINK_PARK 表示 SLC1 RX 链表 FSM 状态。

0：FSM 不处于空闲状态

1：FSM 处于空闲状态

(RO)

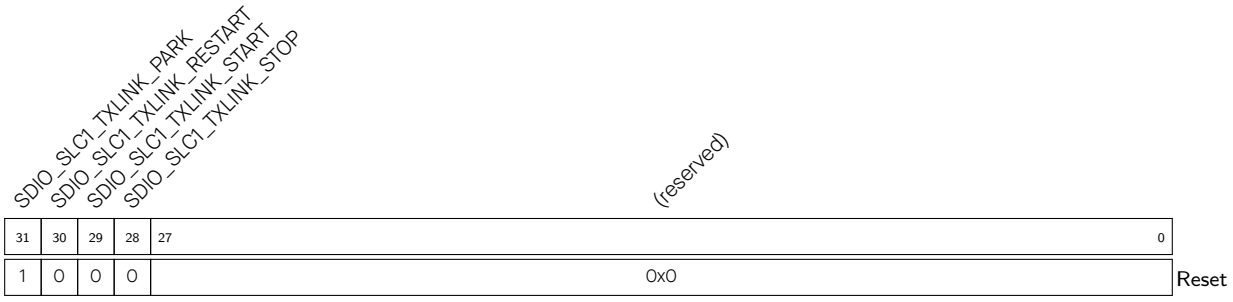
Register 39.13. SDIO_SLC1RX_LINK_ADDR_REG (0x0050)



SDIO_SLC1_RXLINK_ADDR 表示 SLC1 RX 链表初始地址。

(R/W)

Register 39.14. SDIO_SLC1TX_LINK_REG (0x0054)



SDIO_SLC1_TXLINK_STOP 配置是否停止 SLC1 TX 链表操作。

0：没有作用

1：停止操作

(R/W/SC)

SDIO_SLC1_TXLINK_START 配置是否从 SDIO_SLC1_TXLINK_ADDR 指定的地址开始 SLC1 TX 链表操作。

0：没有作用

1：开始操作

(R/W/SC)

SDIO_SLC1_TXLINK_RESTART 配置是否重新启动并继续执行 SLC1 TX 链表操作。

0：没有作用

1：重新启动操作

(R/W/SC)

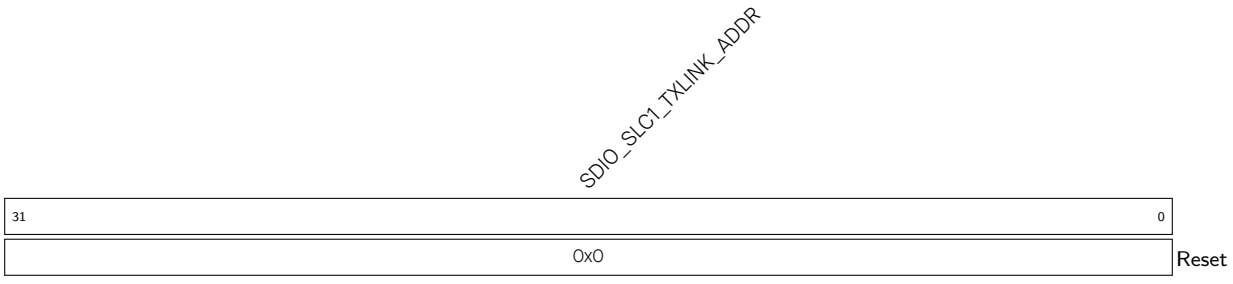
SDIO_SLC1_TXLINK_PARK 表示 SLC1 TX 链表 FSM 状态。

0：FSM 不处于空闲状态

1：FSM 处于空闲状态

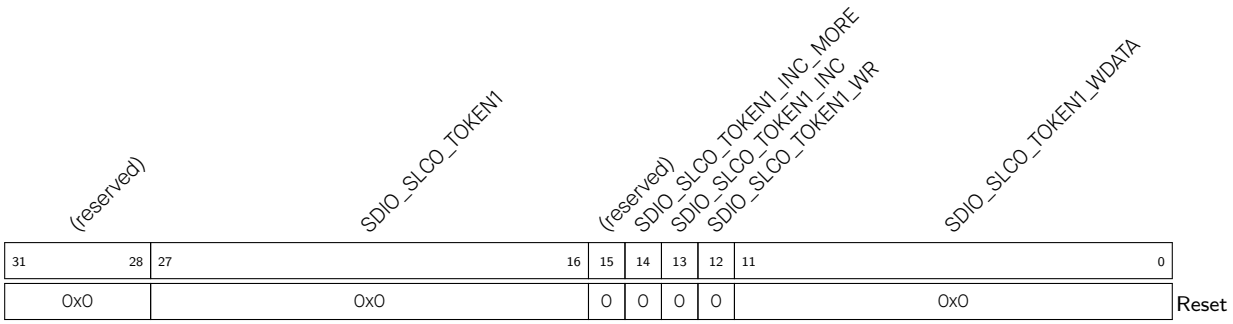
(RO)

Register 39.15. SDIO_SLC1TX_LINK_ADDR_REG (0x0058)



SDIO_SLC1_TXLINK_ADDR 表示 SLC1 TX 链表初始地址。(R/W)

Register 39.16. SDIO_SLC0TOKEN1_REG (0x0064)



SDIO_SLC0_TOKEN1_WDATA 配置 SLC0 token1 的值。(WT)

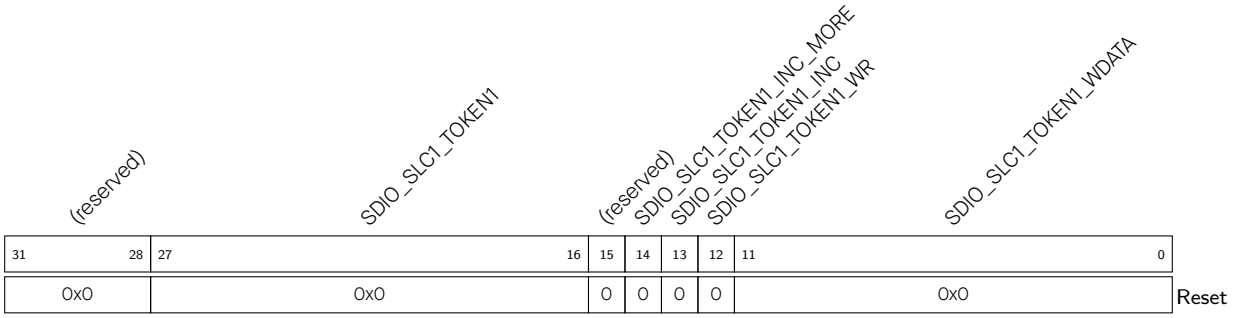
SDIO_SLC0_TOKEN1_WR 设置该位为 1 后，SDIO_SLC0_TOKEN1_WDATA 的值将写入 SDIO_SLC0_TOKEN1。(WT)

SDIO_SLC0_TOKEN1_INC 设置该位为 1 后，SDIO_SLC0_TOKEN1 的值将加 1。(WT)

SDIO_SLC0_TOKEN1_INC_MORE 设置该位为 1 后，SDIO_SLC0_TOKEN1_WDATA 的值将与 SDIO_SLC0_TOKEN1 的值相加。(WT)

SDIO_SLC0_TOKEN1 表示可用于通过 SLC0 通道接收数据包的 buffer 的累计值。(RO)

Register 39.17. SDIO_SLC1TOKEN1_REG (0x006C)



SDIO_SLC1_TOKEN1_WDATA 配置 SLC1 token1 的值。(WT)

SDIO_SLC1_TOKEN1_WR 设置该位为 1 后，SDIO_SLC1_TOKEN1_WDATA 的值将写入 SDIO_SLC1_TOKEN1。(WT)

SDIO_SLC1_TOKEN1_INC 设置该位为 1 后，SDIO_SLC1_TOKEN1 的值将加 1。(WT)

SDIO_SLC1_TOKEN1_INC_MORE 设置该位为 1 后，SDIO_SLC1_TOKEN1_WDATA 的值将与 SDIO_SLC1_TOKEN1 的值相加。(WT)

SDIO_SLC1_TOKEN1 表示可用于通过 SLC1 通道接收数据包的 buffer 的累计值。(RO)

Register 39.18. SDIO_SLCCONF1_REG (0x0070)



SDIO_SDIO_CMD_HOLD_EN 请初始化为 0，且不要修改。(R/W)

SDIO_SLC0_LEN_AUTO_CLR 请初始化为 0，且不要修改。(R/W)

SDIO_SLC0_TX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

SDIO_SLC0_RX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

SDIO_HOST_INT_LEVEL_SEL 配置发送至主机的中断的极性。

0：低电平中断有效

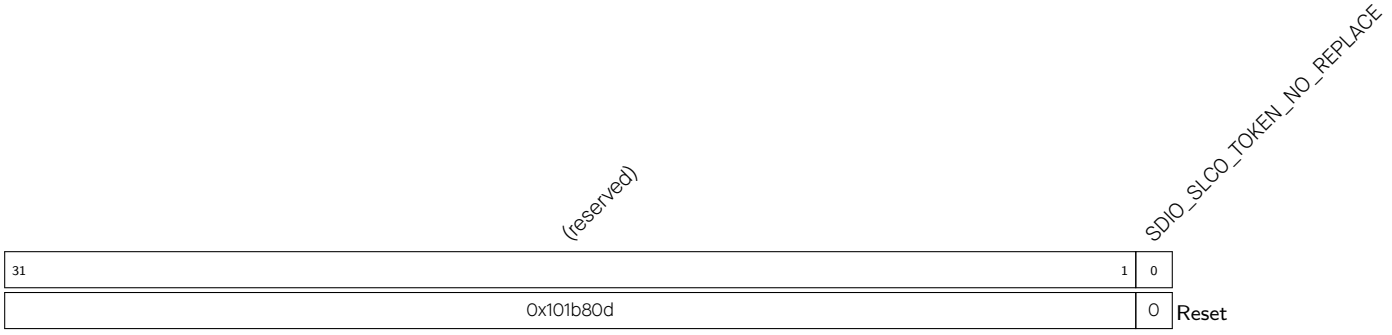
1：高电平中断有效

(R/W)

SDIO_SLC1_TX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

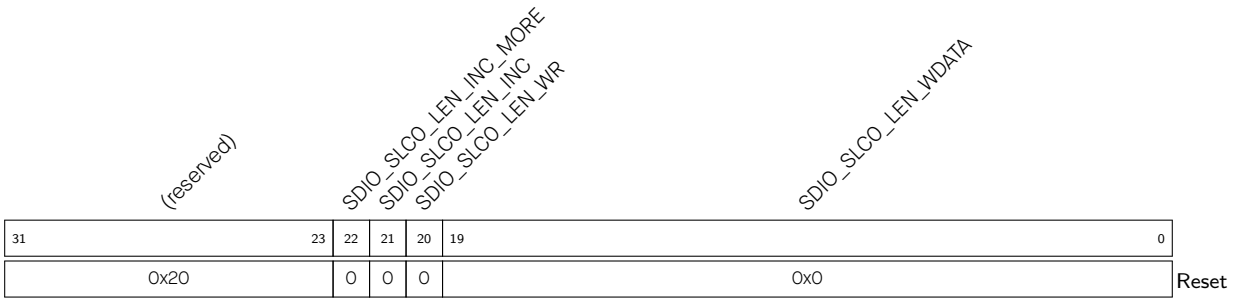
SDIO_SLC1_RX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

Register 39.19. SDIO_SLC_RX_DSCR_CONF_REG (0x00A8)



SDIO_SLC0_TOKEN_NO_REPLACE 请初始化为 1，且不要修改。(R/W)

Register 39.20. SDIO_SLCO_LEN_CONF_REG (0x00F4)



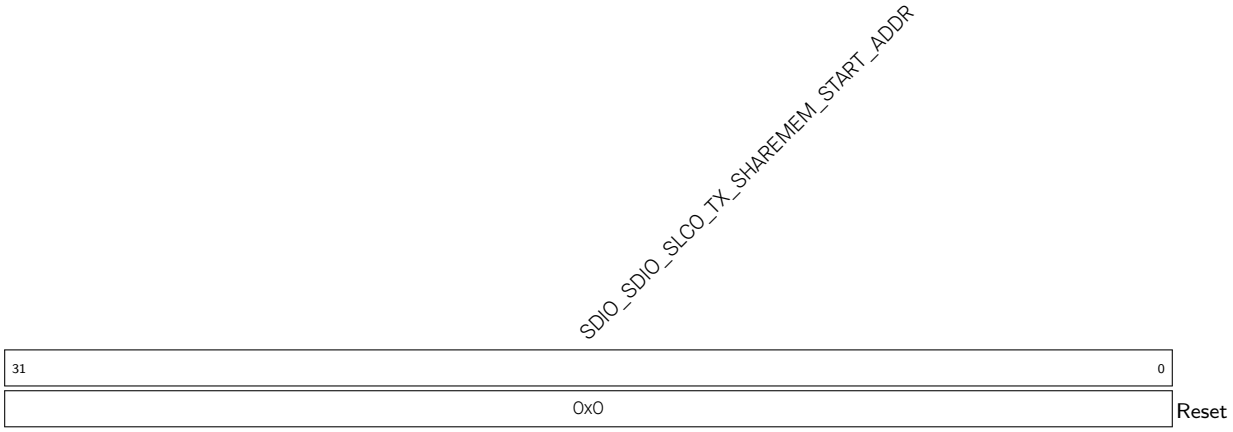
SDIO_SLCO_LEN_WDATA 配置从机想要发送的数据的长度。(WT)

SDIO_SLCO_LEN_WR 设置该位为 1 后，SDIO_SLCO_LEN_WDATA 的值将写入 SDIO_SLCO_LEN 和 SLCHOST_HOSTSLCHOST_SLCO_LEN。(WT)

SDIO_SLCO_LEN_INC 设置该位为 1 后，SDIO_SLCO_LEN 和 SLCHOST_HOSTSLCHOST_SLCO_LEN 的值将分别加 1。(WT)

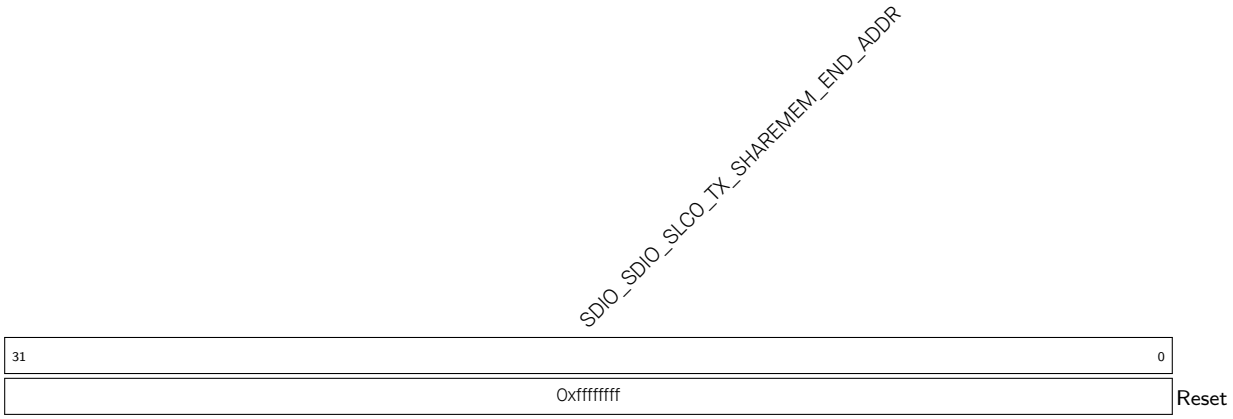
SDIO_SLCO_LEN_INC_MORE 设置该位为 1 后，SDIO_SLCO_LEN 和 SLCHOST_HOSTSLCHOST_SLCO_LEN 的值将分别加上 SDIO_SLCO_LEN_WDATA 的值。(WT)

Register 39.21. SDIO_SLCO_TX_SHAREMEM_START_REG (0x0154)



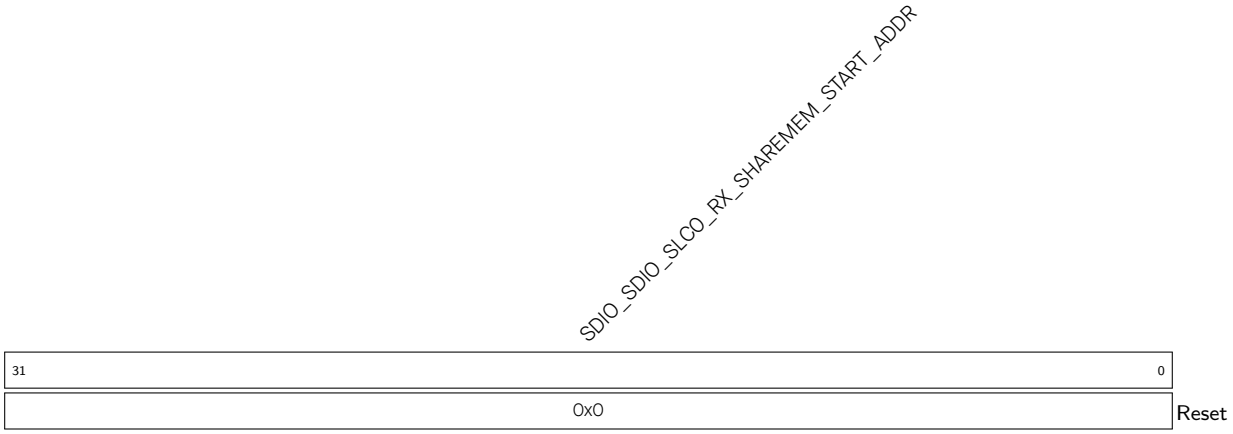
SDIO_SDIO_SLCO_TX_SHAREMEM_START_ADDR 配置 SLCO 主机至从机通道 AHB 起始地址边界。(R/W)

Register 39.22. SDIO_SLC0_TX_SHAREMEM_END_REG (0x0158)



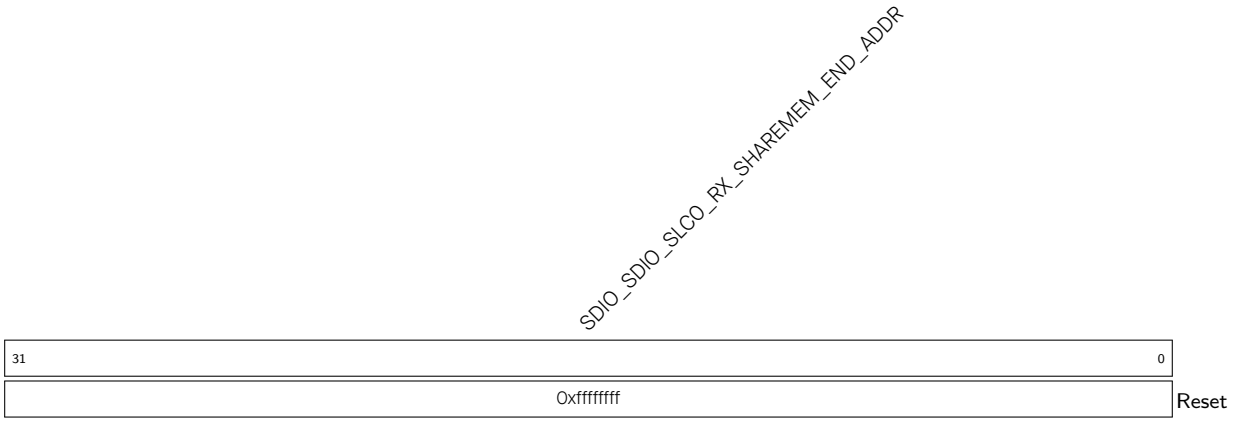
SDIO_SDIO_SLC0_TX_SHAREMEM_END_ADDR 配置 SLC0 主机至从机通道 AHB 结束地址边界。(R/W)

Register 39.23. SDIO_SLC0_RX_SHAREMEM_START_REG (0x015C)



SDIO_SDIO_SLC0_RX_SHAREMEM_START_ADDR 配置 SLC0 从机至主机通道 AHB 起始地址边界。(R/W)

Register 39.24. SDIO_SLC0_RX_SHAREMEM_END_REG (0x0160)



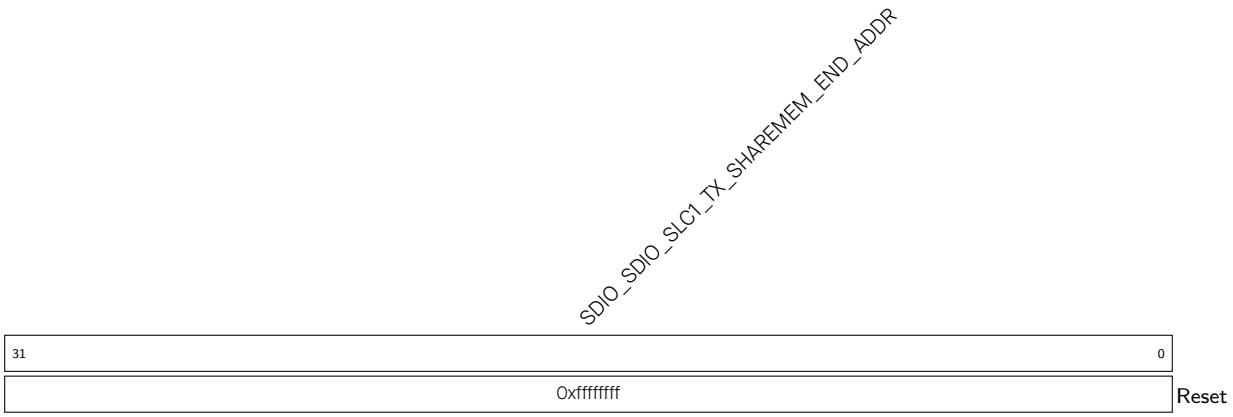
SDIO_SDIO_SLC0_RX_SHAREMEM_END_ADDR 配置 SLC0 从机至主机通道 AHB 结束地址边界。(R/W)

Register 39.25. SDIO_SLC1_TX_SHAREMEM_START_REG (0x0164)



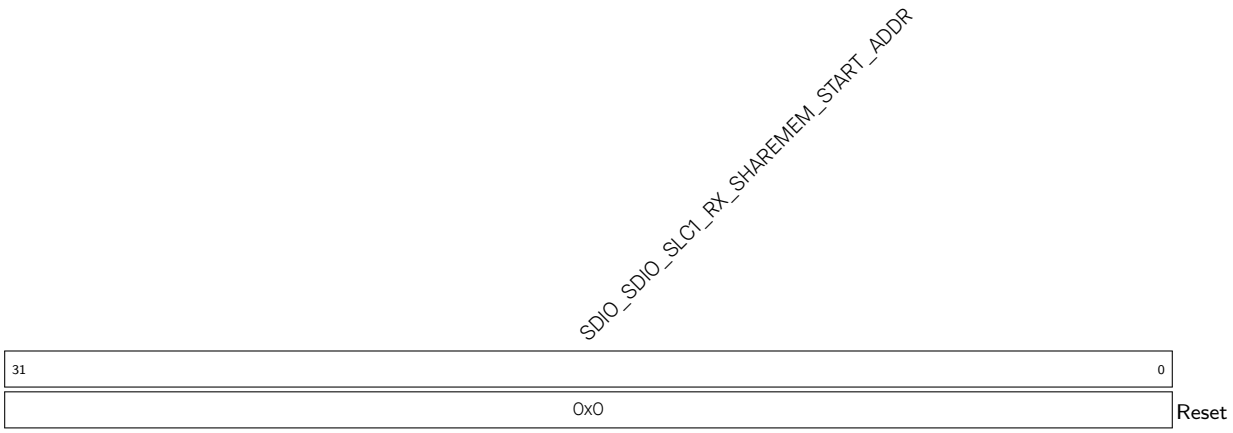
SDIO_SDIO_SLC1_TX_SHAREMEM_START_ADDR 配置 SLC1 主机至从机通道 AHB 起始地址边界。(R/W)

Register 39.26. SDIO_SLC1_TX_SHAREMEM_END_REG (0x0168)



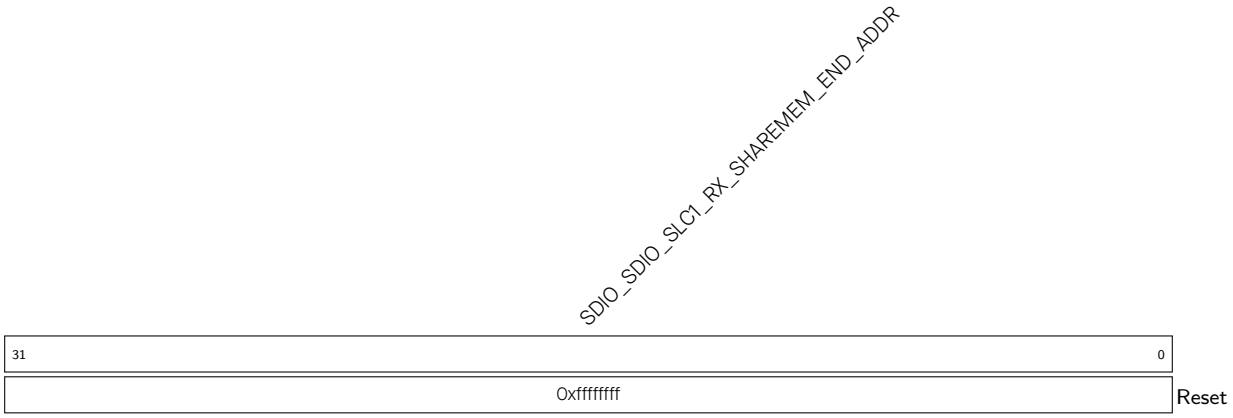
SDIO_SDIO_SLC1_TX_SHAREMEM_END_ADDR 配置 SLC1 主机至从机通道 AHB 结束地址边界。(R/W)

Register 39.27. SDIO_SLC1_RX_SHAREMEM_START_REG (0x016C)



SDIO_SDIO_SLC1_RX_SHAREMEM_START_ADDR 配置 SLC1 从机至主机通道 AHB 起始地址边界。(R/W)

Register 39.28. SDIO_SLC1_RX_SHAREMEM_END_REG (0x0170)



SDIO_SDIO_SLC1_RX_SHAREMEM_END_ADDR 配置 SLC1 从机至主机通道 AHB 结束地址边界。(R/W)

Register 39.29. SDIO_SLC_BURST_LEN_REG (0x017C)



SDIO_SLC0_TXDATA_BURST_LEN 配置 SLC0 主机至从机通道 AHB burst 类型。

0：可以使用 incr4

1：可以使用 incr8

(R/W)

SDIO_SLC0_RXDATA_BURST_LEN 配置 SLC0 从机至主机通道 AHB burst 类型。

0：可以使用 incr 和 incr4

1：可以使用 incr 和 incr8

(R/W)

SDIO_SLC1_TXDATA_BURST_LEN 配置 SLC1 主机至从机通道 AHB burst 类型。

0：可以使用 incr4

1：可以使用 incr8

(R/W)

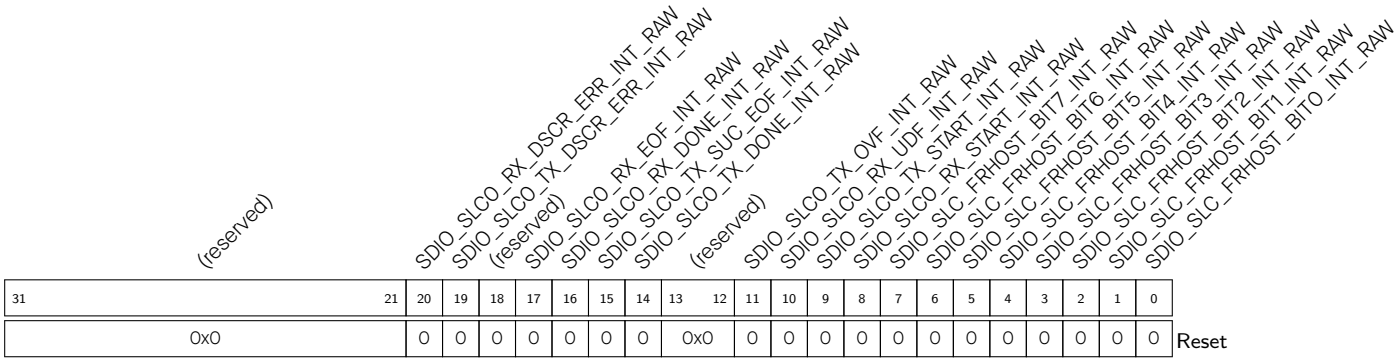
SDIO_SLC1_RXDATA_BURST_LEN 配置 SLC1 从机至主机通道 AHB burst 类型。

0：可以使用 incr 和 incr4

1：可以使用 incr 和 incr8

(R/W)

Register 39.30. SDIO_SLC0INT_RAW_REG (0x0004)



SDIO_SLC_FRHOST_BIT*n*_INT_RAW (*n*: 0-7) SLC_FRHOST_BIT*n*_INT (*n*: 0-7) 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_START_INT_RAW SLC0_RX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_START_INT_RAW SLC0_TX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_UDF_INT_RAW SLC0_RX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_OVF_INT_RAW SLC0_TX_OVF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_DONE_INT_RAW SLC0_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_SUC_EOF_INT_RAW SLC0_TX_SUC_EOF_INT 的原始中断状态。(R/WTC/SS)

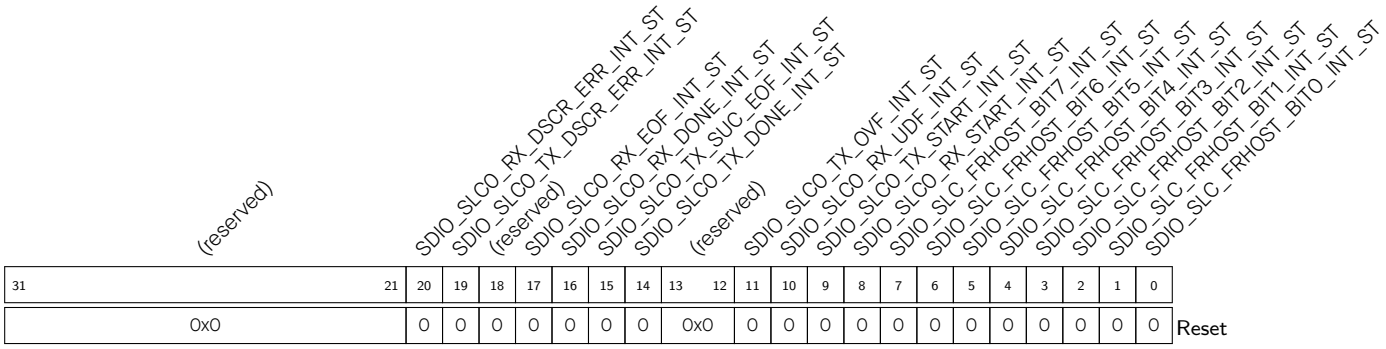
SDIO_SLC0_RX_DONE_INT_RAW SLC0_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_EOF_INT_RAW SLC0_RX_EOF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_DSCR_ERR_INT_RAW SLC0_TX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_DSCR_ERR_INT_RAW SLC0_RX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

Register 39.31. SDIO_SLCOINT_ST_REG (0x0008)



SDIO_SLC_FRHOST_BIT*n*_INT_ST (*n*: 0-7) SLC_FRHOST_BIT*n*_INT (*n*: 0-7) 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_START_INT_ST SLC0_RX_START_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_START_INT_ST SLC0_TX_START_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_UDF_INT_ST SLC0_RX_UDF_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_OVF_INT_ST SLC0_TX_OVF_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_DONE_INT_ST SLC0_TX_DONE_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_SUC_EOF_INT_ST SLC0_TX_SUC_EOF_INT 的屏蔽中断状态。(RO)

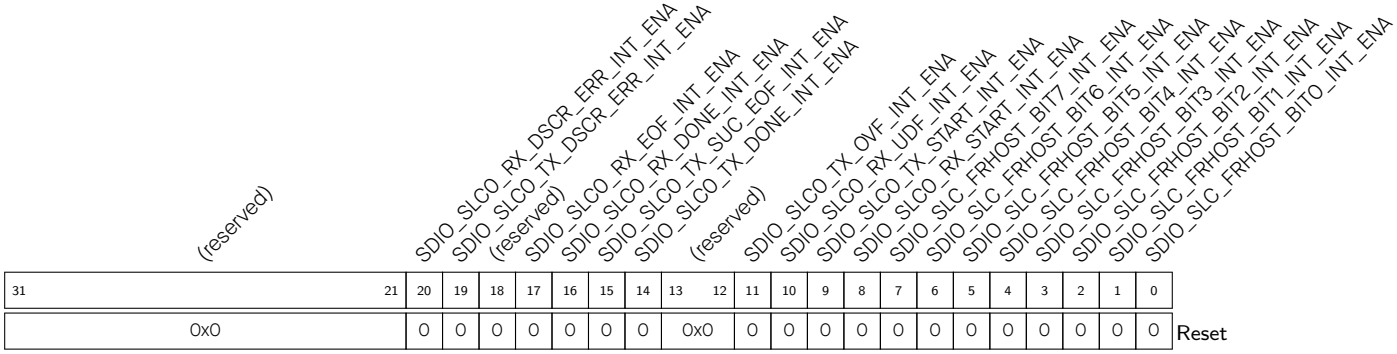
SDIO_SLC0_RX_DONE_INT_ST SLC0_RX_DONE_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_EOF_INT_ST SLC0_RX_EOF_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_DSCR_ERR_INT_ST SLC0_TX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_DSCR_ERR_INT_ST SLC0_RX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

Register 39.32. SDIO_SLCOINT_ENA_REG (0x000C)



SDIO_SLC_FRHOST_BIT*n*_INT_ENA (*n*: 0-7) 写 1 使能 SLC_FRHOST_BIT*n*_INT (*n*: 0-7) 中断。(R/W)

SDIO_SLCO_RX_START_INT_ENA 写 1 使能 SLCO_RX_START_INT 中断。(R/W)

SDIO_SLCO_TX_START_INT_ENA 写 1 使能 SLCO_TX_START_INT 中断。(R/W)

SDIO_SLCO_RX_UDF_INT_ENA 写 1 使能 SLCO_RX_UDF_INT 中断。(R/W)

SDIO_SLCO_TX_OVF_INT_ENA 写 1 使能 SLCO_TX_OVF_INT 中断。(R/W)

SDIO_SLCO_TX_DONE_INT_ENA 写 1 使能 SLCO_TX_DONE_INT 中断。(R/W)

SDIO_SLCO_TX_SUC_EOF_INT_ENA 写 1 使能 SLCO_TX_SUC_EOF_INT 中断。(R/W)

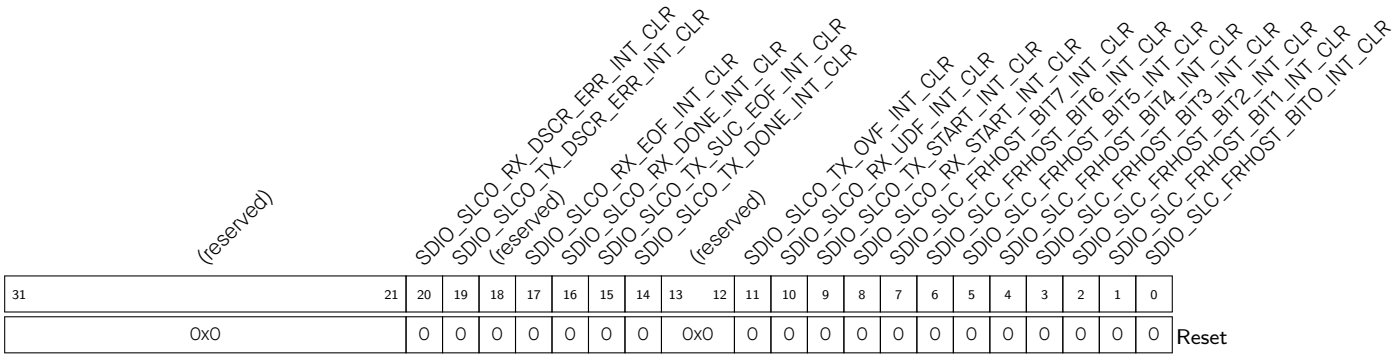
SDIO_SLCO_RX_DONE_INT_ENA 写 1 使能 SLCO_RX_DONE_INT 中断。(R/W)

SDIO_SLCO_RX_EOF_INT_ENA 写 1 使能 SLCO_RX_EOF_INT 中断。(R/W)

SDIO_SLCO_TX_DSCR_ERR_INT_ENA 写 1 使能 SLCO_TX_DSCR_ERR_INT 中断。(R/W)

SDIO_SLCO_RX_DSCR_ERR_INT_ENA 写 1 使能 SLCO_RX_DSCR_ERR_INT 中断。(R/W)

Register 39.33. SDIO_SLC0INT_CLR_REG (0x0010)



SDIO_SLC_FRHOST_BIT*n*_INT_CLR (*n*: 0-7) 写 1 清除中断 SLC_FRHOST_BIT*n*_INT (*n*: 0-7)。(WT)

SDIO_SLC0_RX_START_INT_CLR 写 1 清除中断 SLC0_RX_START_INT。(WT)

SDIO_SLC0_TX_START_INT_CLR 写 1 清除中断 SLC0_TX_START_INT。(WT)

SDIO_SLC0_RX_UDF_INT_CLR 写 1 清除中断 SLC0_RX_UDF_INT。(WT)

SDIO_SLC0_TX_OVF_INT_CLR 写 1 清除中断 SLC0_TX_OVF_INT。(WT)

SDIO_SLC0_TX_DONE_INT_CLR 写 1 清除中断 SLC0_TX_DONE_INT。(WT)

SDIO_SLC0_TX_SUC_EOF_INT_CLR 写 1 清除中断 SLC0_TX_SUC_EOF_INT。(WT)

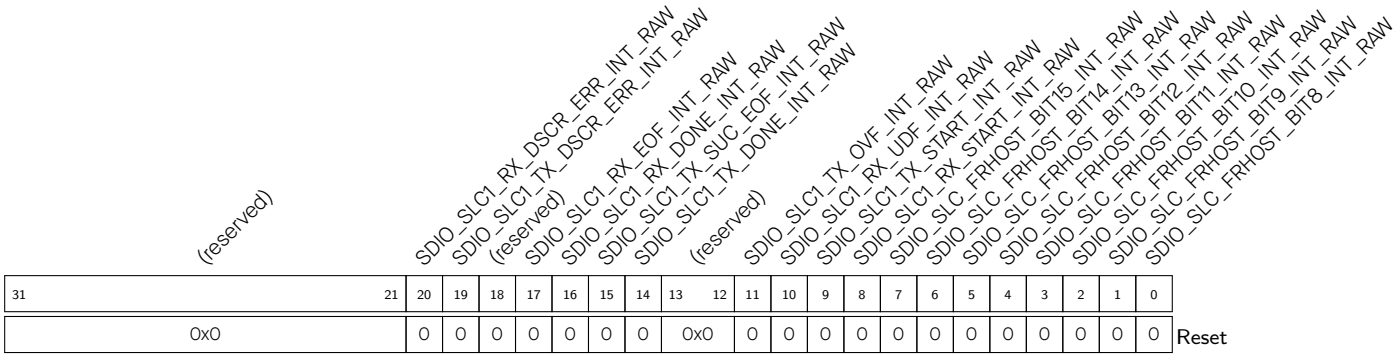
SDIO_SLC0_RX_DONE_INT_CLR 写 1 清除中断 SLC0_RX_DONE_INT。(WT)

SDIO_SLC0_RX_EOF_INT_CLR 写 1 清除中断 SLC0_RX_EOF_INT。(WT)

SDIO_SLC0_TX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC0_TX_DSCR_ERR_INT。(WT)

SDIO_SLC0_RX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC0_RX_DSCR_ERR_INT。(WT)

Register 39.34. SDIO_SLC1INT_RAW_REG (0x0014)



SDIO_SLC_FRHOST_BIT*n*_INT_RAW (*n*: 8-15) SLC_FRHOST_BIT*n*_INT (*n*: 8-15) 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_START_INT_RAW SLC1_RX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_START_INT_RAW SLC1_TX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_UDF_INT_RAW SLC1_RX_UDF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_OVF_INT_RAW SLC1_TX_OVF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_DONE_INT_RAW SLC1_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_SUC_EOF_INT_RAW SLC1_TX_SUC_EOF_INT 的原始中断状态。(R/WTC/SS)

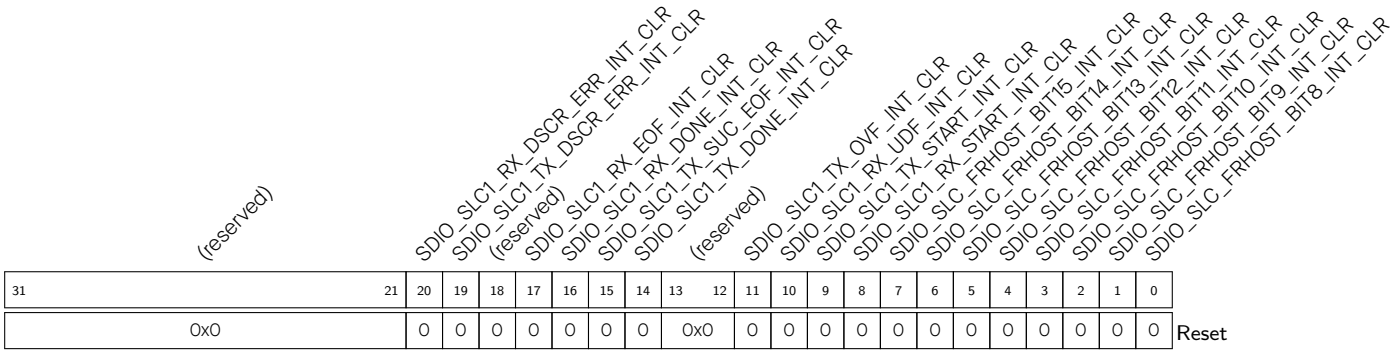
SDIO_SLC1_RX_DONE_INT_RAW SLC1_RX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_EOF_INT_RAW SLC1_RX_EOF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_DSCR_ERR_INT_RAW SLC1_TX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_DSCR_ERR_INT_RAW SLC1_RX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

Register 39.35. SDIO_SLC1INT_CLR_REG (0x0020)



SDIO_SLC_FRHOST_BIT*n*_INT_CLR (*n*: 8-15) 写 1 清除中断 SLC_FRHOST_BIT*n*_INT (*n*: 8-15)。(WT)

SDIO_SLC1_RX_START_INT_CLR 写 1 清除中断 SLC1_RX_START_INT。(WT)

SDIO_SLC1_TX_START_INT_CLR 写 1 清除中断 SLC1_TX_START_INT。(WT)

SDIO_SLC1_RX_UDF_INT_CLR 写 1 清除中断 SLC1_RX_UDF_INT。(WT)

SDIO_SLC1_TX_OVF_INT_CLR 写 1 清除中断 SLC1_TX_OVF_INT。(WT)

SDIO_SLC1_TX_DONE_INT_CLR 写 1 清除中断 SLC1_TX_DONE_INT。(WT)

SDIO_SLC1_TX_SUC_EOF_INT_CLR 写 1 清除中断 SLC1_TX_SUC_EOF_INT。(WT)

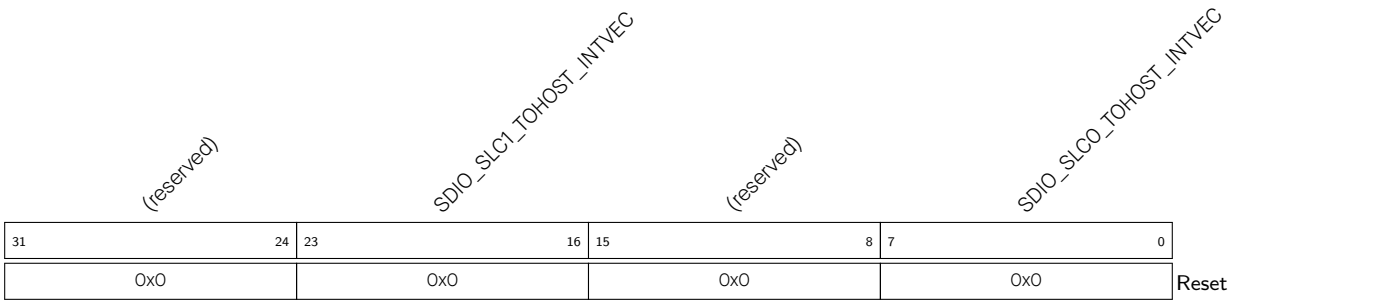
SDIO_SLC1_RX_DONE_INT_CLR 写 1 清除中断 SLC1_RX_DONE_INT。(WT)

SDIO_SLC1_RX_EOF_INT_CLR 写 1 清除中断 SLC1_RX_EOF_INT。(WT)

SDIO_SLC1_TX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC1_TX_DSCR_ERR_INT。(WT)

SDIO_SLC1_RX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC1_RX_DSCR_ERR_INT。(WT)

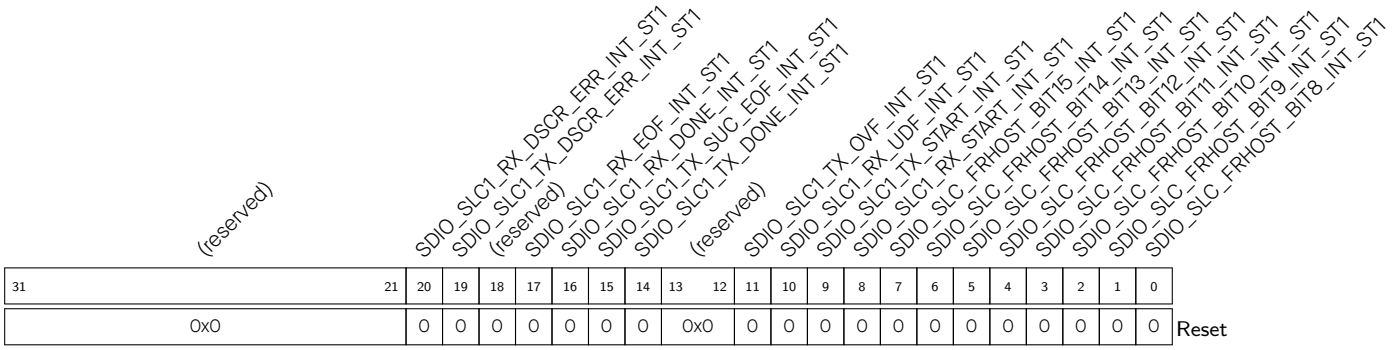
Register 39.36. SDIO_SLCINTVEC_TOHOST_REG (0x005C)



SDIO_SLC0_TOHOST_INTVEC SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7) 中断设置位。这些位不会被自动清除。(WT)

SDIO_SLC1_TOHOST_INTVEC SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7) 中断设置位。这些位不会被自动清除。(WT)

Register 39.37. SDIO_SLC1INT_ST1_REG (0x014C)



SDIO_SLC_FRHOST_BIT*n*_INT_ST1 (*n*: 8-15) SLC_FRHOST_BIT*n*_INT (*n*: 8-15) 的屏蔽中断状态。(RO)

SDIO_SLC1_RX_START_INT_ST1 SLC1_RX_START_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_START_INT_ST1 SLC1_TX_START_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_RX_UDF_INT_ST1 SLC1_RX_UDF_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_OVF_INT_ST1 SLC1_TX_OVF_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_DONE_INT_ST1 SLC1_TX_DONE_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_SUC_EOF_INT_ST1 SLC1_TX_SUC_EOF_INT 的屏蔽中断状态。(RO)

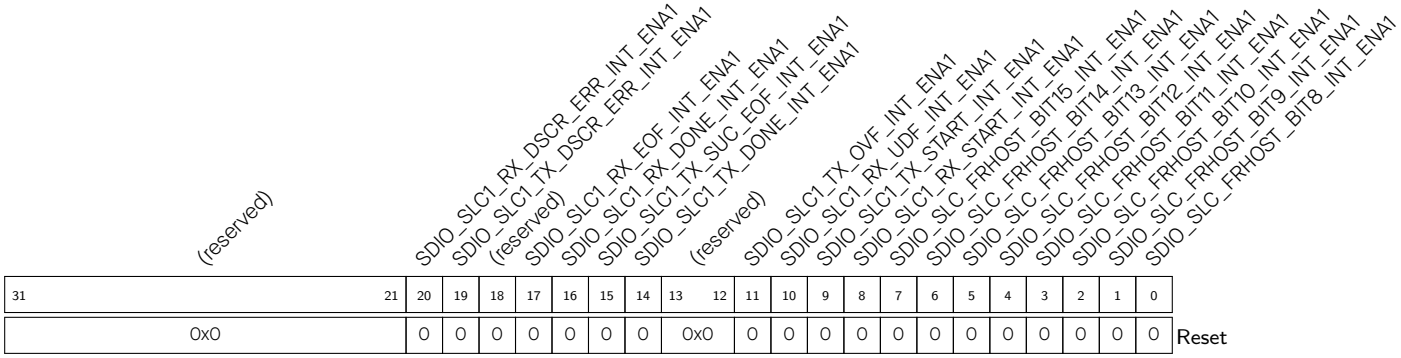
SDIO_SLC1_RX_DONE_INT_ST1 SLC1_RX_DONE_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_RX_EOF_INT_ST1 SLC1_RX_EOF_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_DSCR_ERR_INT_ST1 SLC1_TX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_RX_DSCR_ERR_INT_ST1 SLC1_RX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

Register 39.38. SDIO_SLC1INT_ENA1_REG (0x0150)



SDIO_SLC_FRHOST_BIT*n*_INT_ENA1 (*n*: 8-15) 写 1 使能 SLC_FRHOST_BIT*n*_INT (*n*: 8-15) 中断。(R/W)

SDIO_SLC1_RX_START_INT_ENA1 写 1 使能 SLC1_RX_START_INT 中断。(R/W)

SDIO_SLC1_TX_START_INT_ENA1 写 1 使能 SLC1_TX_START_INT 中断。(R/W)

SDIO_SLC1_RX_UDF_INT_ENA1 写 1 使能 SLC1_RX_UDF_INT 中断。(R/W)

SDIO_SLC1_TX_OVF_INT_ENA1 写 1 使能 SLC1_TX_OVF_INT 中断。(R/W)

SDIO_SLC1_TX_DONE_INT_ENA1 写 1 使能 SLC1_TX_DONE_INT 中断。(R/W)

SDIO_SLC1_TX_SUC_EOF_INT_ENA1 写 1 使能 SLC1_TX_SUC_EOF_INT 中断。(R/W)

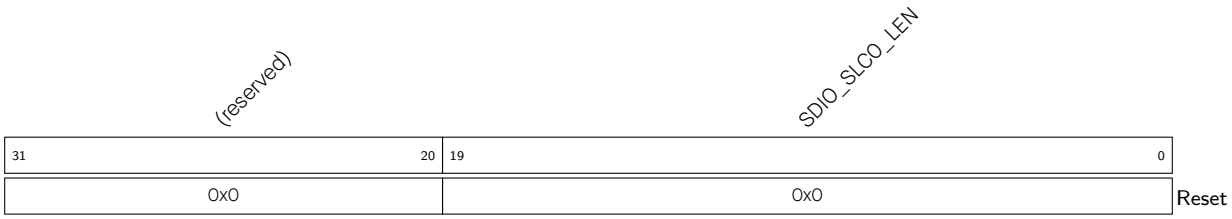
SDIO_SLC1_RX_DONE_INT_ENA1 写 1 使能 SLC1_RX_DONE_INT 中断。(R/W)

SDIO_SLC1_RX_EOF_INT_ENA1 写 1 使能 SLC1_RX_EOF_INT 中断。(R/W)

SDIO_SLC1_TX_DSCR_ERR_INT_ENA1 写 1 使能 SLC1_TX_DSCR_ERR_INT 中断。(R/W)

SDIO_SLC1_RX_DSCR_ERR_INT_ENA1 写 1 使能 SLC1_RX_DSCR_ERR_INT 中断。(R/W)

Register 39.39. SDIO_SLC0_LENGTH_REG (0x00F8)



SDIO_SLC0_LEN 表示从机希望发送的数据总长度。(RO)

39.9.3 SLC Host 寄存器

Register 39.40. SLCHOST_CONF_REG (0x01F0)



SLCHOST_FRC_SDIO11 将 bit[4] 设置为 1，以此强制在时钟下降沿驱动 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟下降沿驱动 DAT[3:0] 信号对应 bit。(R/W)

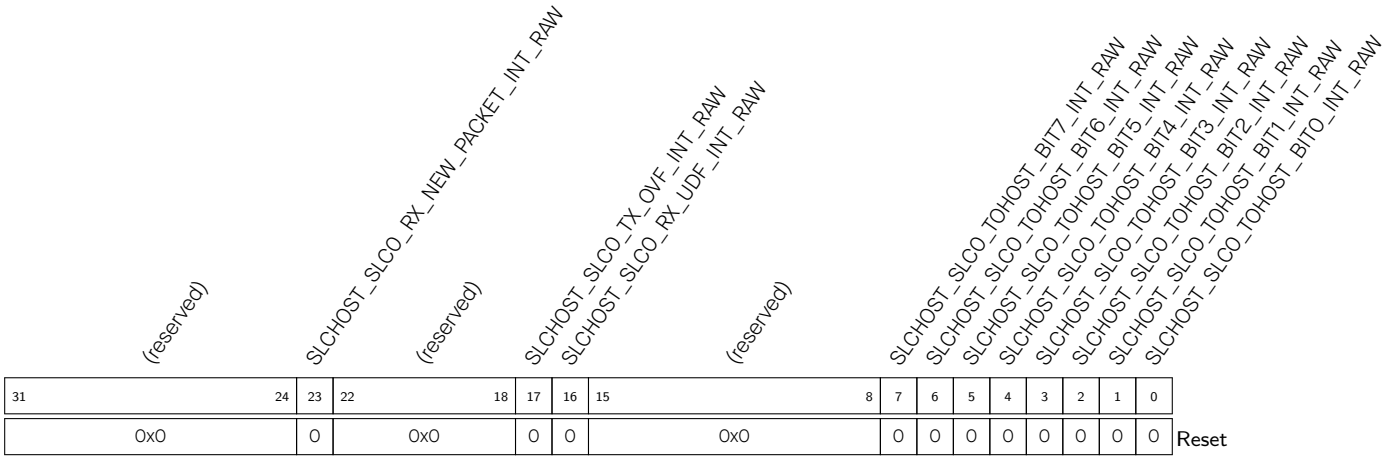
SLCHOST_FRC_SDIO20 将 bit[4] 设置为 1，以此强制在时钟上升沿驱动 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟上升沿驱动 DAT[3:0] 信号对应 bit。(R/W)

SLCHOST_FRC_NEG_SAMP 将 bit[4] 设置为 1，以此强制在时钟下降沿采样 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟下降沿采样 DAT[3:0] 信号对应 bit。(R/W)

SLCHOST_FRC_POS_SAMP 将 bit[4] 设置为 1，以此强制在时钟上升沿采样 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟上升沿采样 DAT[3:0] 信号对应 bit。(R/W)

SLCHOST_HSPEED_CON_EN 将该位和 HINF_HIGHSPEED_ENABLE 设置为 1，然后主机将 CCCR 寄存器中的 EHS 置为 1，以此强制在时钟上升沿驱动 CMD 和 DAT 信号。(R/W)

Register 39.41. SLCHOST_SLCOHOST_INT_RAW_REG (0x0050)



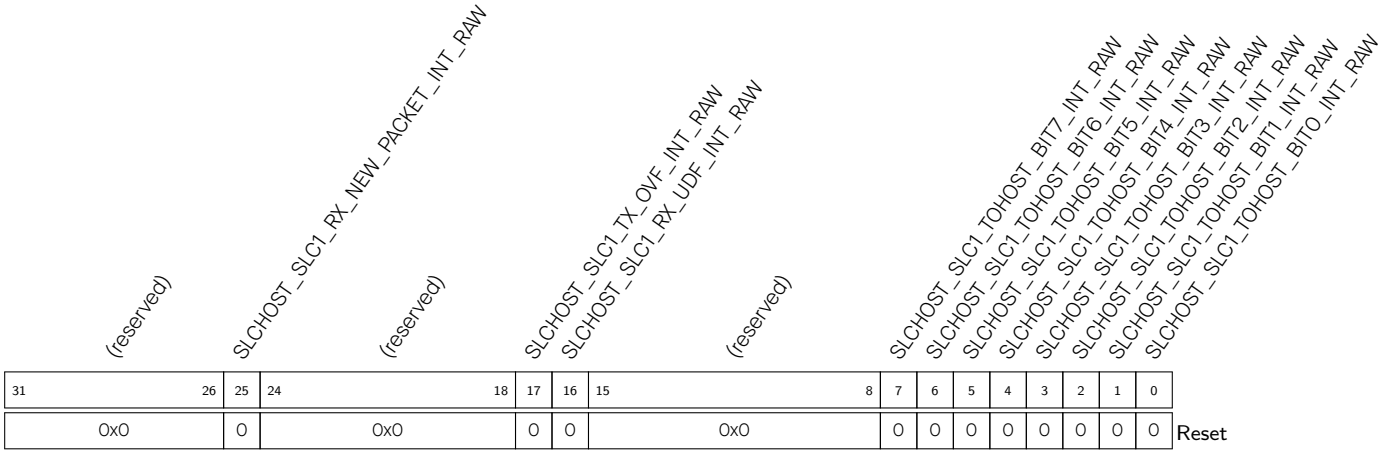
SLCHOST_SLCO_TOHOST_BIT*n*_INT_RAW (*n*: 0-7) SLCHOST_SLCO_TOHOST_BIT*n*_INT (*n*: 0-7) 的原始中断状态。(R/WTC/SS)

SLCHOST_SLCO_RX_UDF_INT_RAW SLCHOST_SLCO_RX_UDF_INT 的原始中断状态。(R/WTC/SS)

SLCHOST_SLCO_TX_OVF_INT_RAW SLCHOST_SLCO_TX_OVF_INT 的原始中断状态。(R/WTC/SS)

SLCHOST_SLCO_RX_NEW_PACKET_INT_RAW SLCHOST_SLCO_RX_NEW_PACKET_INT 的原始中断状态。(R/WTC/SS)

Register 39.42. SLCHOST_SLC1HOST_INT_RAW_REG (0x0054)



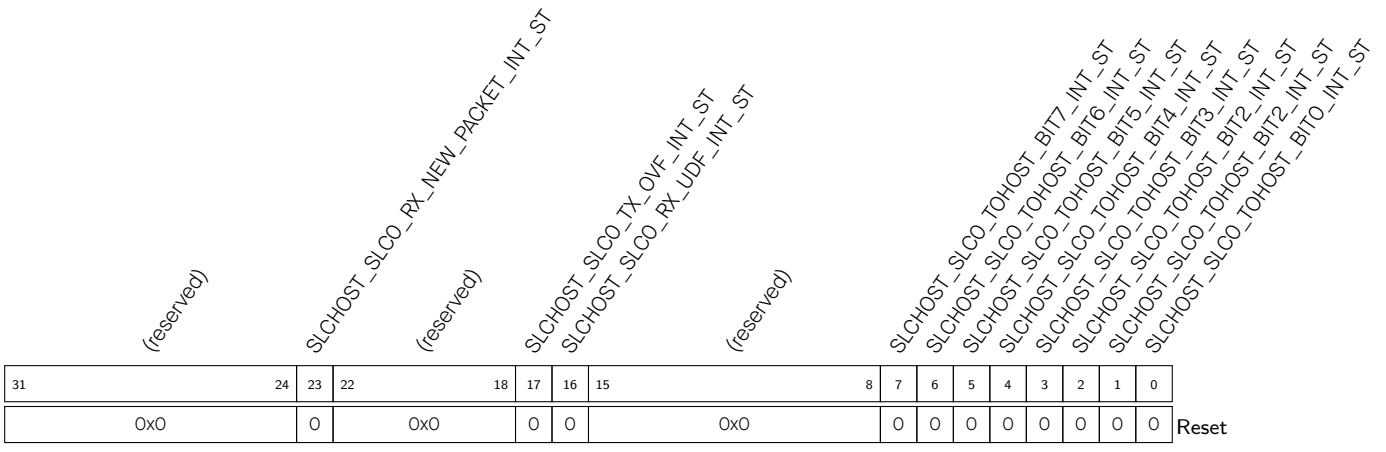
SLCHOST_SLC1_TOHOST_BIT*n*_INT_RAW (*n*: 0-7) SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7) 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC1_RX_UDF_INT_RAW SLCHOST_SLC1_RX_UDF_INT 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC1_TX_OVF_INT_RAW SLCHOST_SLC1_TX_OVF_INT 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC1_RX_NEW_PACKET_INT_RAW SLCHOST_SLC1_RX_NEW_PACKET_INT 的原始中断状态。(R/WTC/SS)

Register 39.43. SLCHOST_SLC0HOST_INT_ST_REG (0x0058)



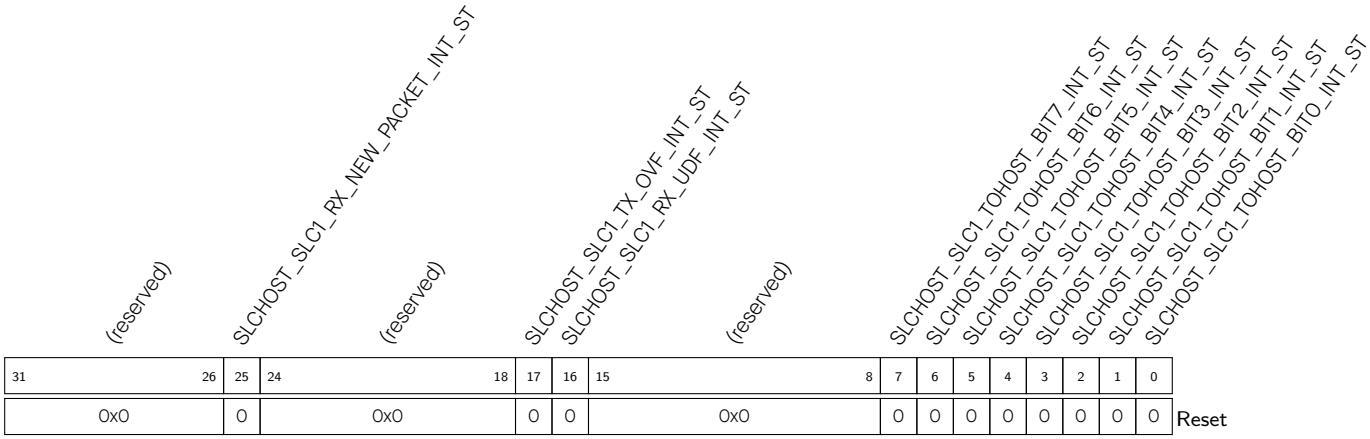
SLCHOST_SLC0_TOHOST_BIT*n*_INT_ST (*n*: 0-7) SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7) 的屏蔽中断状态。(RO)

SLCHOST_SLC0_RX_UDF_INT_ST SLCHOST_SLC0_RX_UDF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC0_TX_OVF_INT_ST SLCHOST_SLC0_TX_OVF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC0_RX_NEW_PACKET_INT_ST SLCHOST_SLC0_RX_NEW_PACKET_INT 的屏蔽中断状态。(RO)

Register 39.44. SLCHOST_SLC1HOST_INT_ST_REG (0x005C)



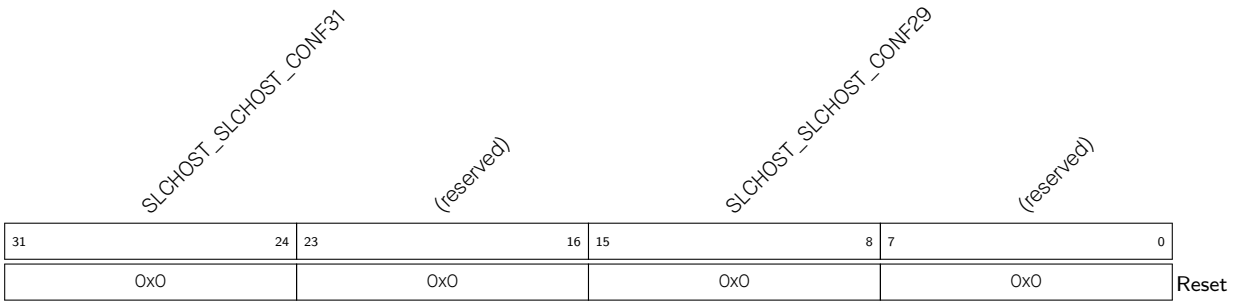
SLCHOST_SLC1_TOHOST_BIT*n*_INT_ST (*n*: 0-7) SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7) 的屏蔽中断状态。(RO)

SLCHOST_SLC1_RX_UDF_INT_ST SLCHOST_SLC1_RX_UDF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC1_TX_OVF_INT_ST SLCHOST_SLC1_TX_OVF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC1_RX_NEW_PACKET_INT_ST SLCHOST_SLC1_RX_NEW_PACKET_INT 的屏蔽中断状态。(RO)

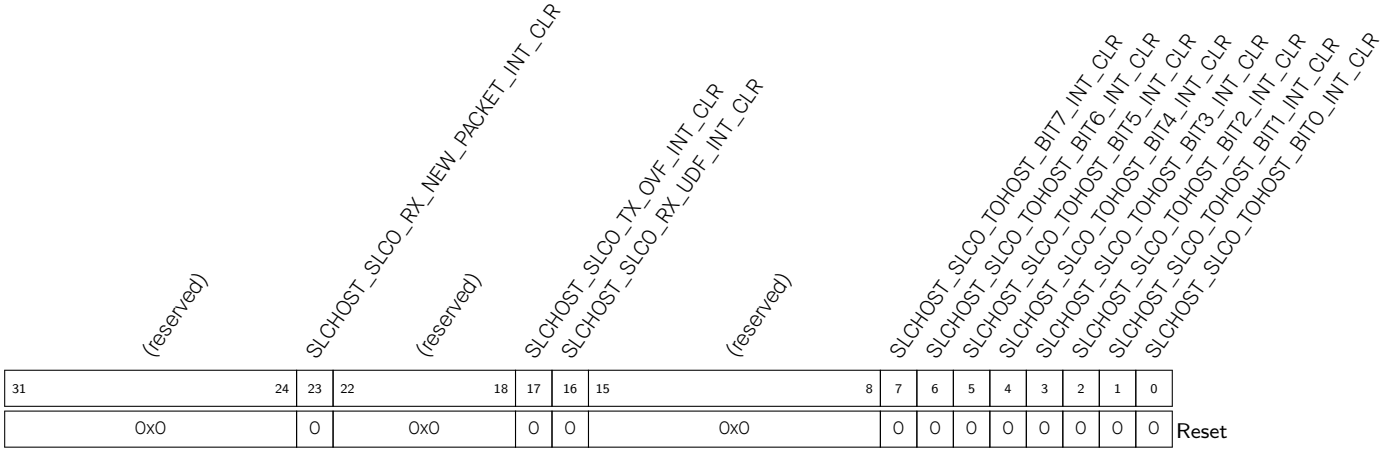
Register 39.45. SLCHOST_CONF_W7_REG (0x008C)



SLCHOST_SLCHOST_CONF29 SLCINT_SLC_FRHOST_BIT*n*_INT (*n*: 0-7) 的中断设置位，这些位不会自动清除。(R/W)

SLCHOST_SLCHOST_CONF31 SLCINT_SLC_FRHOST_BIT*n*_INT (*n*: 8-15) 的中断设置位，这些位不会自动清除。(R/W)

Register 39.46. SLCHOST_SLC0HOST_INT_CLR_REG (0x00D4)



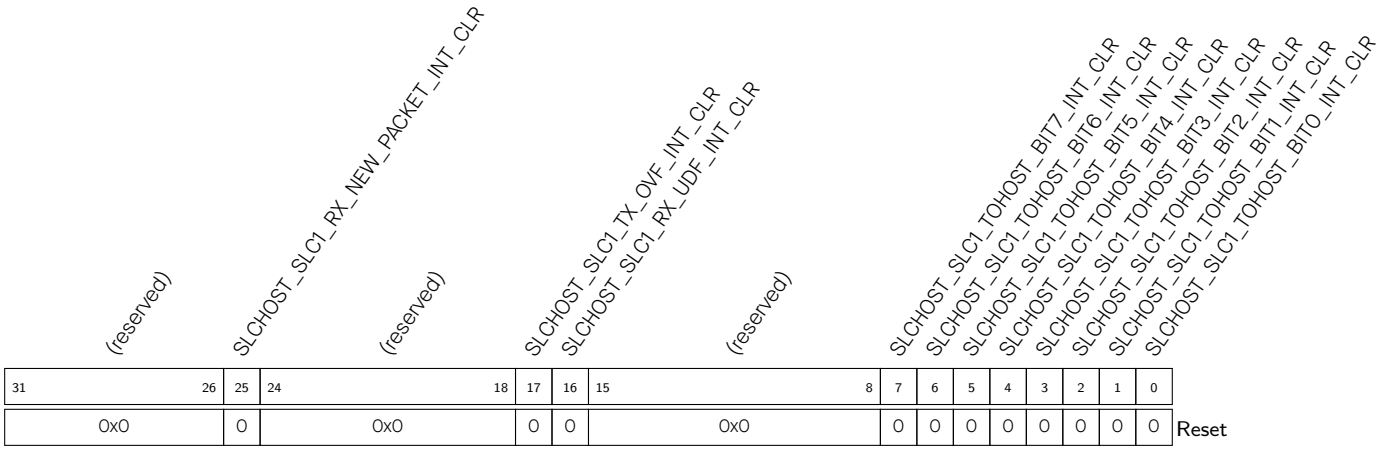
SLCHOST_SLC0_TOHOST_BIT*n*_INT_CLR (*n*: 0-7) 写 1 清除 SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7)。(WT)

SLCHOST_SLC0_RX_UDF_INT_CLR 写 1 清除 SLCHOST_SLC0_RX_UDF_INT。(WT)

SLCHOST_SLC0_TX_OVF_INT_CLR 写 1 清除 SLCHOST_SLC0_TX_OVF_INT。(WT)

SLCHOST_SLC0_RX_NEW_PACKET_INT_CLR 写 1 清除 SLCHOST_SLC0_RX_NEW_PACKET_INT。(WT)

Register 39.47. SLCHOST_SLC1HOST_INT_CLR_REG (0x00D8)



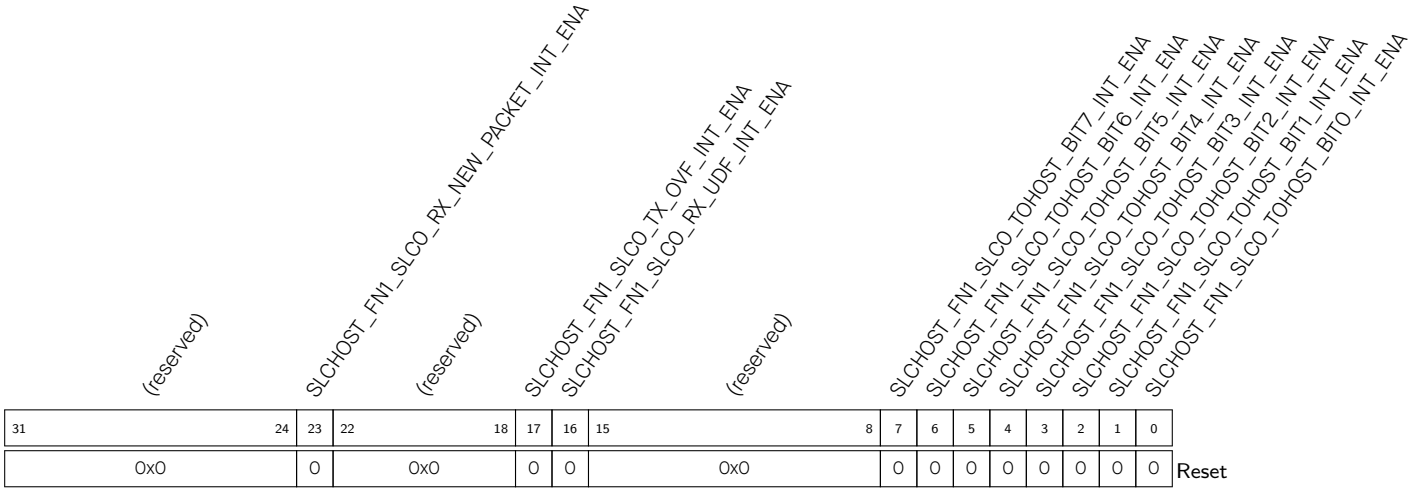
SLCHOST_SLC1_TOHOST_BIT*n*_INT_CLR (*n*: 0-7) 写 1 清除 SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7)。(WT)

SLCHOST_SLC1_RX_UDF_INT_CLR 写 1 清除 SLCHOST_SLC1_RX_UDF_INT。(WT)

SLCHOST_SLC1_TX_OVF_INT_CLR 写 1 清除 SLCHOST_SLC1_TX_OVF_INT。(WT)

SLCHOST_SLC1_RX_NEW_PACKET_INT_CLR 写 1 清除 SLCHOST_SLC1_RX_NEW_PACKET_INT。(WT)

Register 39.48. SLCHOST_SLC0HOST_FUNC1_INT_ENA_REG (0x00DC)



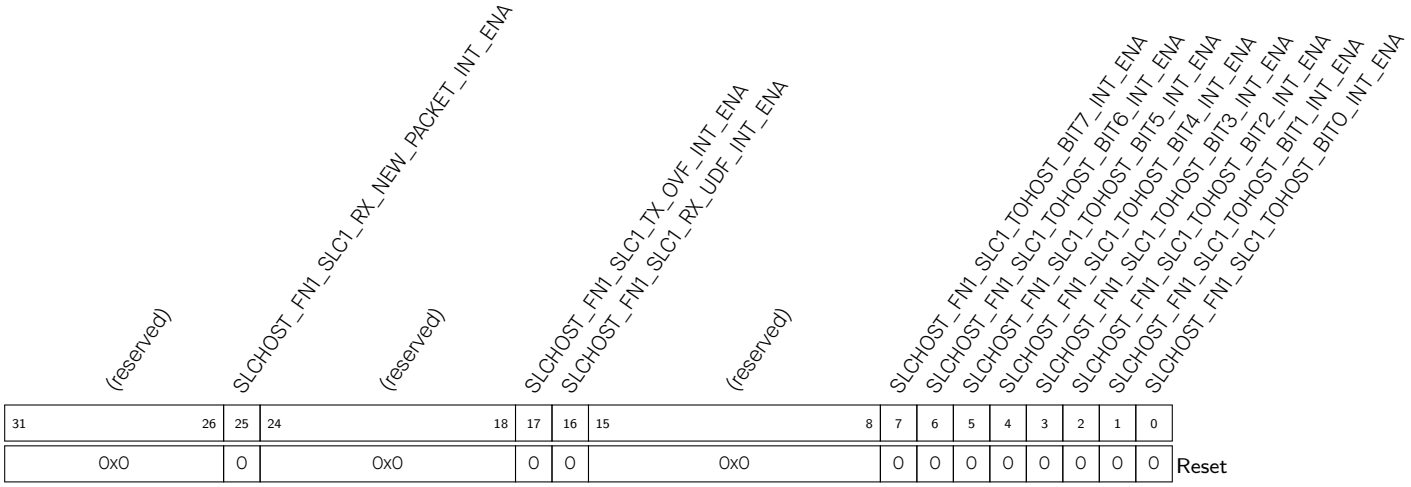
SLCHOST_FN1_SLC0_TOHOST_BIT*n*_INT_ENA (*n*: 0-7) 写 1 使能 SLCHOST_SLC0_TOHOST_BIT*n*_INT (*n*: 0-7)。(R/W)

SLCHOST_FN1_SLC0_RX_UDF_INT_ENA 写 1 使能 SLCHOST_SLC0_RX_UDF_INT。(R/W)

SLCHOST_FN1_SLC0_TX_OVF_INT_ENA 写 1 使能 SLCHOST_SLC0_TX_OVF_INT。(R/W)

SLCHOST_FN1_SLC0_RX_NEW_PACKET_INT_ENA 写 1 使能 SLCHOST_SLC0_RX_NEW_PACKET_INT。(R/W)

Register 39.49. SLCHOST_SLC1HOST_FUNC1_INT_ENA_REG (0x00E0)



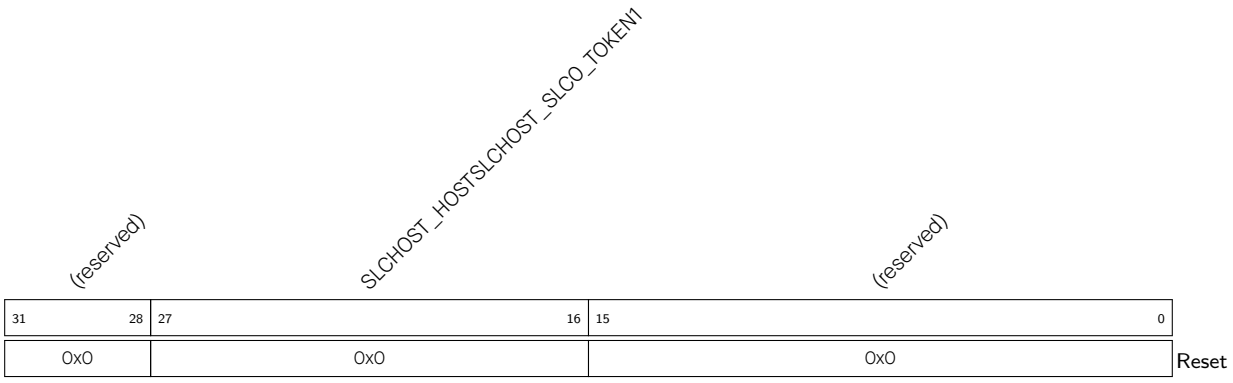
SLCHOST_FN1_SLC1_TOHOST_BIT*n*_INT_ENA (*n*: 0-7) 写 1 使能 SLCHOST_SLC1_TOHOST_BIT*n*_INT (*n*: 0-7)。(R/W)

SLCHOST_FN1_SLC1_RX_UDF_INT_ENA 写 1 使能 SLCHOST_SLC1_RX_UDF_INT。(R/W)

SLCHOST_FN1_SLC1_TX_OVF_INT_ENA 写 1 使能 SLCHOST_SLC1_TX_OVF_INT。(R/W)

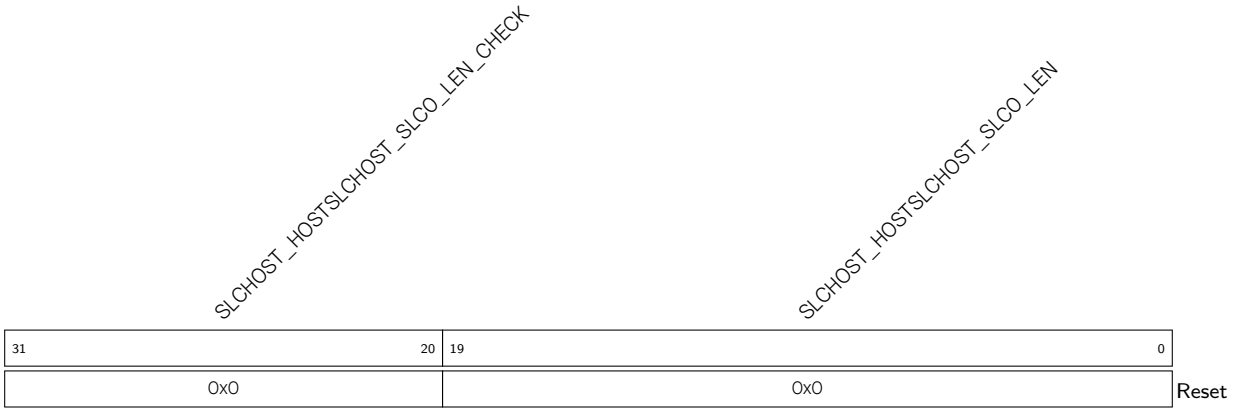
SLCHOST_FN1_SLC1_RX_NEW_PACKET_INT_ENA 写 1 使能 SLCHOST_SLC1_RX_NEW_PACKET_INT。(R/W)

Register 39.50. SLCHOST_SLC0HOST_TOKEN_RDATA_REG (0x0044)



SLCHOST_HOSTSLCHOST_SLC0_TOKEN1 表示可用于通过 SLC0 通道接收数据的 buffer 的累计值。(RO)

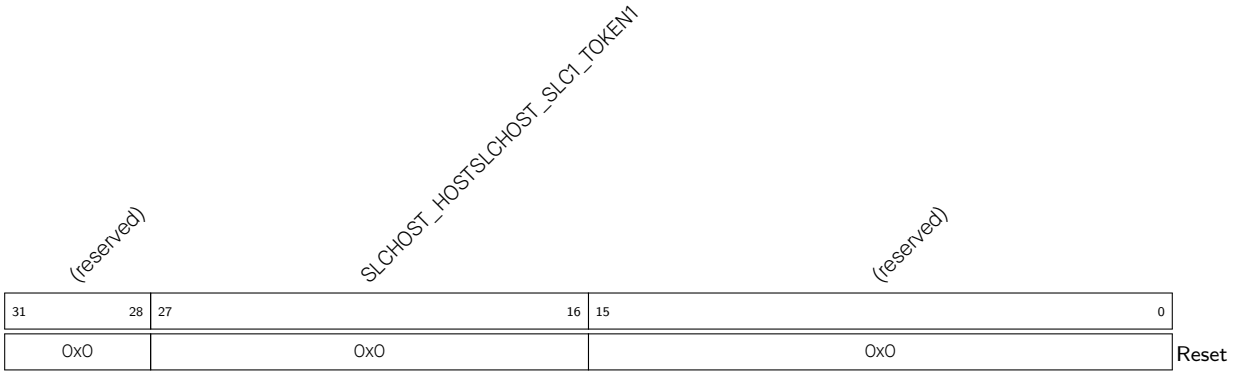
Register 39.51. SLCHOST_PKT_LEN_REG (0x0060)



SLCHOST_HOSTSLCHOST_SLCO_LEN 表示从机总共希望发送的数据量，该字段的值只在被主机读取时更新。(RO)

SLCHOST_HOSTSLCHOST_SLCO_LEN_CHECK 校验 SLCHOST_HOSTSLCHOST_SLCO_LEN 字段的值，该字段的值等于 SLCHOST_HOSTSLCHOST_SLCO_LEN bit[9:0] 加上 bit[19:10]。(RO)

Register 39.52. SLCHOST_SLC1HOST_TOKEN_RDATA_REG (0x00C4)



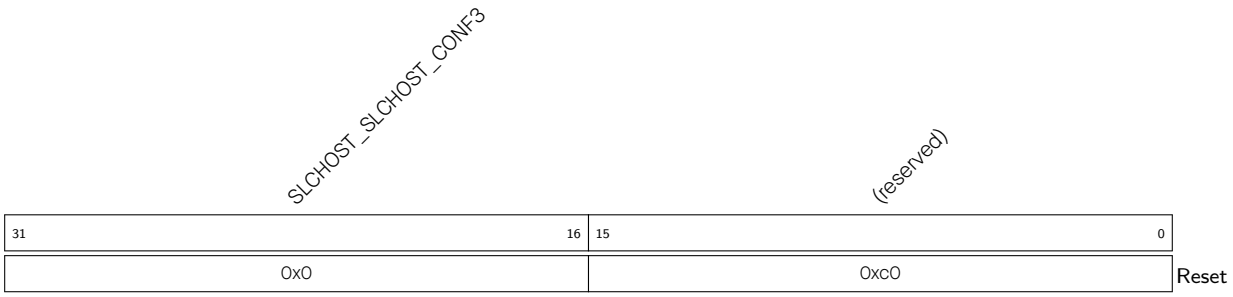
SLCHOST_HOSTSLCHOST_SLC1_TOKEN1 表示可用于通过 SLC1 通道接收数据的 buffer 的累计值。(RO)

Register 39.53. SLCHOST_CONF_W*n*_REG(*n*: 0-2) (0x006C+0x4**n*)



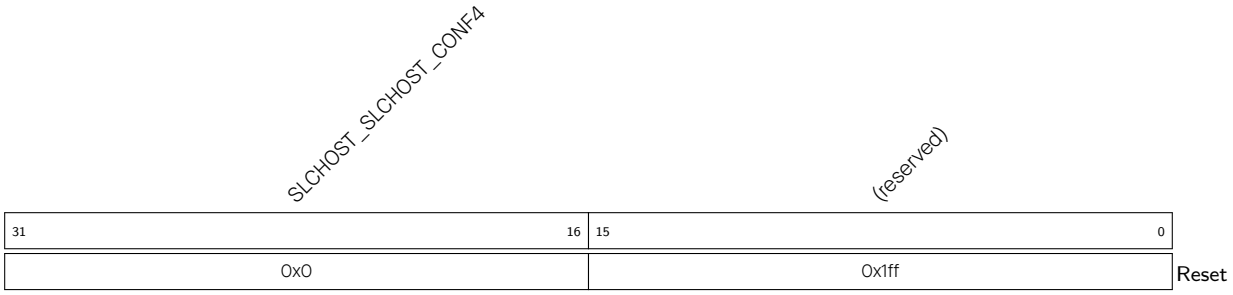
SLCHOST_SLCHOST_CONF*n* 主机与从机之间的信息交互寄存器。(R/W)

Register 39.54. SLCHOST_CONF_W3_REG (0x0078)



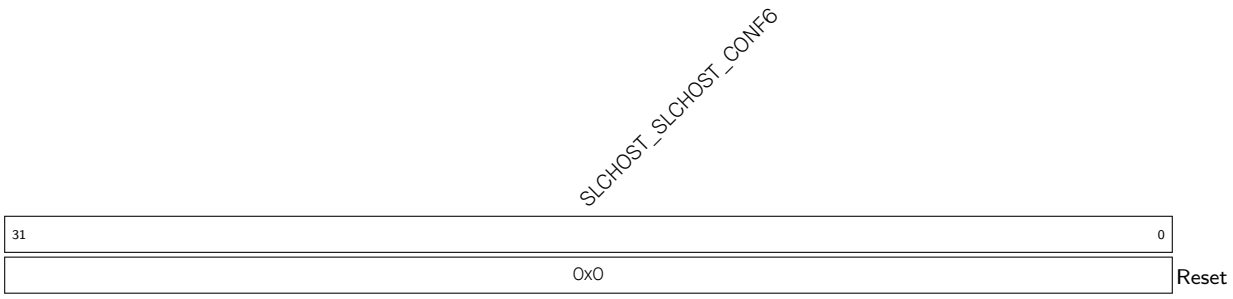
SLCHOST_SLCHOST_CONF3 主机与从机之间的信息交互寄存器。(R/W)

Register 39.55. SLCHOST_CONF_W4_REG (0x007C)



SLCHOST_SLCHOST_CONF4 主机与从机之间的信息交互寄存器。(R/W)

Register 39.56. SLCHOST_CONF_W6_REG (0x0088)



SLCHOST_SLCHOST_CONF6 主机与从机之间的信息交互寄存器。(R/W)

Register 39.57. SLCHOST_CONF_W*n*_REG(*n*: 8-15) (0x009C+0x4*(*n*-8))



SLCHOST_SLCHOST_CONF*n* 主机与从机之间的信息交互寄存器。(R/W)

第 40 章

LED PWM 控制器 (LEDC)

40.1 概述

LED PWM 控制器用于生成控制 LED 的脉冲宽度调制信号 (PWM)，具有占空比自动渐变等功能。该外设也可生成 PWM 信号用作其他用途。

40.2 主要特性

LED PWM 控制器具有如下特性:

- 6 个独立的 PWM 生成器（即 6 个通道）
- PWM 占空比最大精度为 20 位
- 4 个独立定时器，可实现小数分频
- PWM 输出信号相位可调节
- PWM 占空比微调
- 占空比自动渐变，即 PWM 信号占空比可逐渐增加或减小，无须处理器干预，渐变完成时产生中断
- 每个 PWM 生成器包含 16 个占空比渐变区间，用于生成占空比伽马曲线渐变的信号。每个区间都可以独立配置占空比变化方向（增加或减少）、变化步长（占空比每次增加或减少的量）、变化次数（占空比在一个范围内的渐变次数）以及变化频率
- 低功耗模式 (Light-sleep mode) 下可输出 PWM 信号
- 可以生成 ETM（事件任务矩阵）外设相关的事件，可以接收 ETM 外设相关的任务

由于 4 个定时器具有相同的功能和运行方式，下文将 4 个定时器统称为定时器*x*（*x* 的范围是 0 到 3）。同样的，6 个 PWM 生成器的功能和运行方式也相同，因此下文将统称为 PWM*n*（*n* 的范围是 0 到 5）。

40.3 结构概览

图 40.3-1为 LED PWM 控制器的架构。

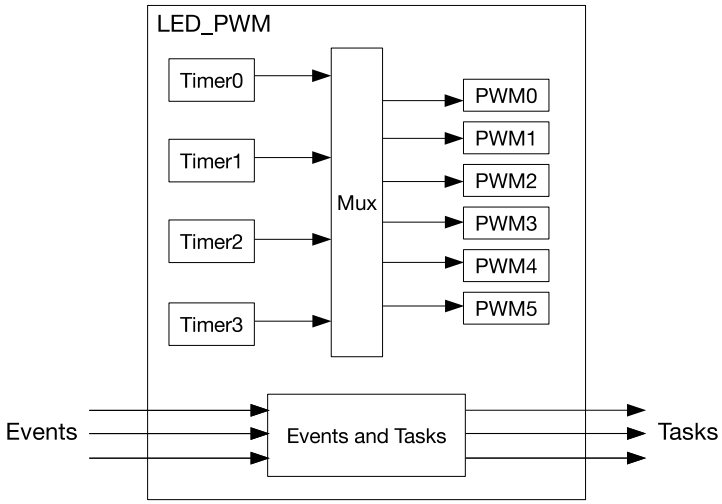


图 40.3-1. LED PWM 控制器架构

4 个定时器每个内部都有一个时基计数器（即基于基准时钟周期计数的计数器），因此每个定时器都可以独立配置（可配置时钟分频器和计数器最大值）。每个 PWM 生成器通过配置 LEDC_TIMER_SEL_CH*n* 可以在 4 个定时器中择一，并以该定时器的计数值 timer*x*_cnt 为基准生成 PWM 信号。

图 40.3-2 为定时器和 PWM 生成器的主要功能块。

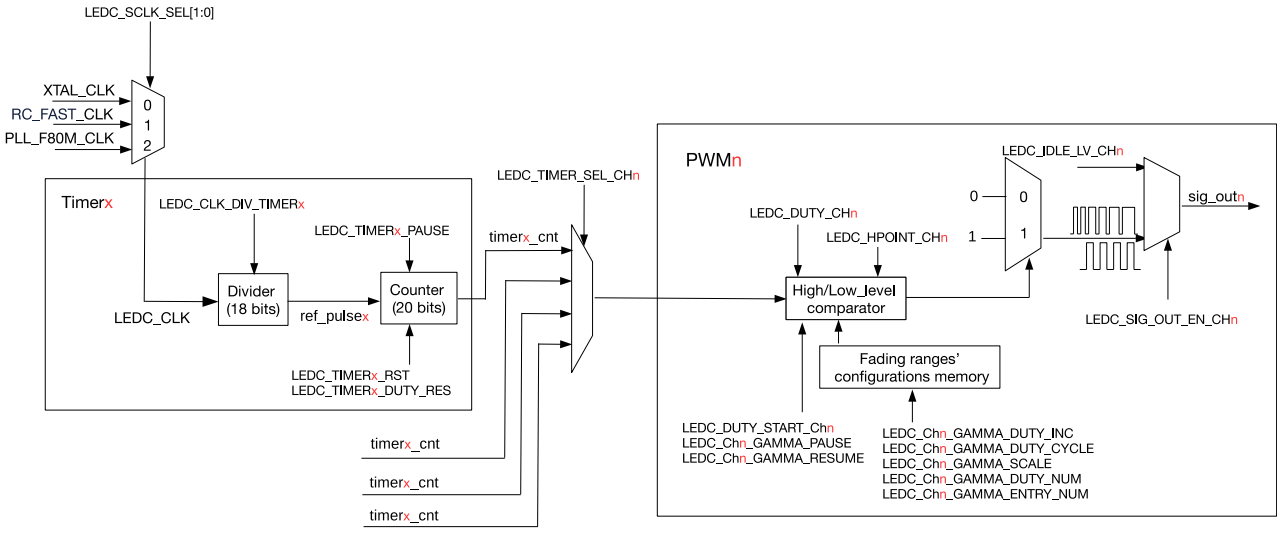


图 40.3-2. 定时器和 PWM 生成器功能框图

40.4 功能描述

40.4.1 定时器

LED PWM 控制器的每个定时器内部都有一个时基计数器。图 40.3-2 中时基计数器使用的时钟信号称为 ref_pulse*x*。所有定时器使用同一个时钟源信号 LEDC_CLK，该时钟源信号经分频器分频后产生 ref_pulse*x* 供计数器使用。

40.4.1.1 时钟源

软件可配置的 LED PWM 寄存器由 APB_CLK 时钟驱动。要使用 LED PWM 控制器，需使能 LED PWM 的 APB_CLK 时钟信号。APB_CLK 时钟信号可通过置位 PCR_LEDC_CONF_REG 寄存器的 PCR_LEDC_CLK_EN 字段使能。LED PWM 的 LEDC_CLK 时钟信号可通过置位 PCR_LEDC_SCLK_CONF_REG 寄存器的 PCR_LEDC_SCLK_EN 字段使能。LED PWM 可通过软件置位 PCR_LEDC_CONF_REG 寄存器的 PCR_LEDC_RST_EN 字段复位。

LED PWM 控制器的定时器从下述时钟信号中选择一个作为其时钟源信号：XTAL_CLK、RC_FAST_CLK 和 PLL_F80M_CLK。为 LEDC_CLK 选择时钟源信号的配置如下：

- XTAL_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 0
- RC_FAST_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 1
- PLL_F80M_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 2

之后，LEDC_CLK 信号会进入时钟分频器。

更多信息，请参阅章节 9 复位和时钟。

40.4.1.2 时钟分频器配置

LEDC_CLK 信号传输到时钟分频器，产生 ref_pulse*x* 信号供计数器使用。ref_pulse*x* 的频率等于 LEDC_CLK 的频率经分频系数 LEDC_CLK_DIV 分频后的结果（见图 40.3-2）。

由于时钟分频器为小数分频，因此分频系数 LEDC_CLK_DIV 可为非整数，其可根据下列等式配置：

$$\text{LEDC\_CLK\_DIV} = A + \frac{B}{256} \tag{40.1}$$

- 整数部分 *A* 为 LEDC_CLK_DIV_TIMER*x* 字段的高 10 位（即 LEDC_TIMER*x*_CONF_REG[22:13]）
- 小数部分 *B* 为 LEDC_CLK_DIV_TIMER*x* 字段的低 8 位（即 LEDC_TIMER*x*_CONF_REG[12:5]）

小数部分 *B* 为 0 时，LEDC_CLK_DIV 的值为整数（整数分频）。也就是说，每 *A* 个 LEDC_CLK 时钟周期产生一个 ref_pulse*x* 时钟脉冲。

小数部分 *B* 不为 0 时，LEDC_CLK_DIV 的值为非整数。时钟分频器按照 *A* 个 LEDC_CLK 时钟周期和 (*A*+1) 个 LEDC_CLK 时钟周期轮流产生 ref_pulse*x* 时钟脉冲，实现非整数分频。这样一来，ref_pulse*x* 时钟脉冲的平均频率便会是理想值（非整数分频的频率）。每 256 个 ref_pulse*x* 时钟脉冲中：

- 有 *B* 个 ref_pulse*x* 时钟脉冲以 (*A*+1) 个 LEDC_CLK 时钟周期分频
- 有 (256-*B*) 个 ref_pulse*x* 时钟脉冲以 *A* 个 LEDC_CLK 时钟周期分频
- 以 (*A*+1) 个 LEDC_CLK 时钟周期分频的 ref_pulse*x* 时钟脉冲均匀分布在以 *A* 个 LEDC_CLK 时钟周期分频的 ref_pulse*x* 时钟脉冲中

图 40.4-1 展示了分频系数 LEDC_CLK_DIV 非整数时，LEDC_CLK 时钟脉冲和 ref_pulse*x* 时钟脉冲的关系。

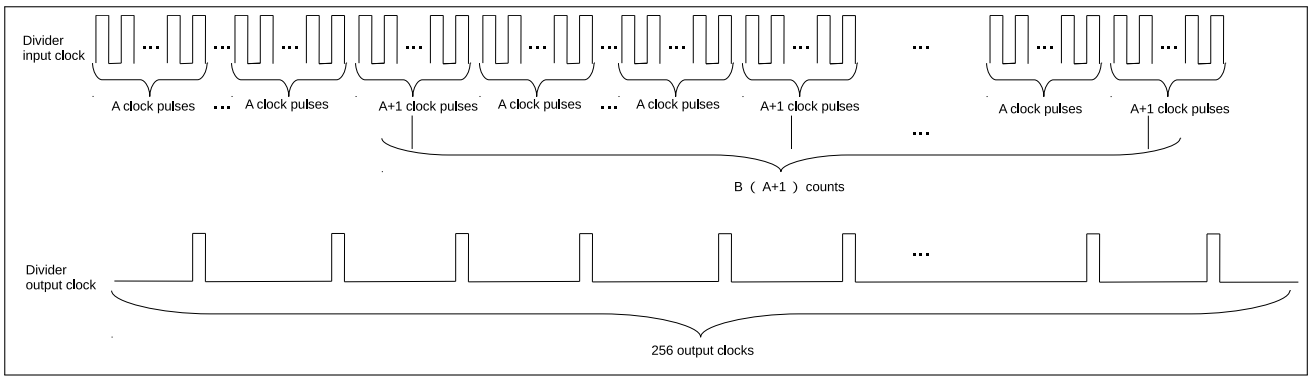


图 40.4-1. LEDC_CLK_DIV 为非整数时的分频

在运行时改变定时器时钟的分频系数，需先配置 LEDC_CLK_DIV_TIMER*x* 字段，然后置位 LEDC_TIMER*x*_PARA_UP 字段应用新配置。新配置会在计数器下次溢出时生效。LEDC_TIMER*x*_PARA_UP 字段由硬件自动清除。

40.4.1.3 20 位计数器

每个定时器有一个以 ref_pulse*x* 为基准时钟的 20 位时基计数器（见图 40.3-2）。LEDC_TIMER*x*_DUTY_RES 字段用于配置 20 位计数器的实际使用位宽。因此，PWM 信号的最大精度为 20 位。计数器最大可计数至 $2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} - 1$，然后溢出并重新从 0 开始计数。软件可以读取、复位、暂停计数器。计数器和 PWM 信号精度的关系如图 40.4-2 所示。

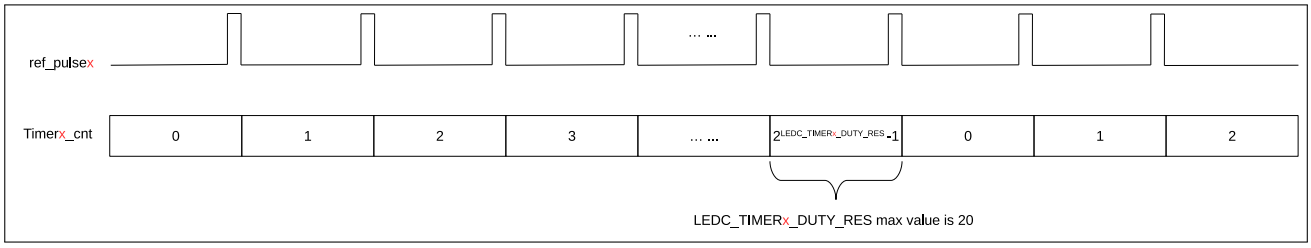


图 40.4-2. 计数器和 PWM 信号精度关系

计数器可在每次溢出时触发 LEDC_TIMER*x*_OVF_INT 中断（这个中断为硬件自动产生，不需要配置）。计数器也可配置为在溢出 LEDC_OVF_NUM_CH*n* + 1 次时触发 LEDC_OVF_CNT_CH*n*_INT 中断，该中断配置步骤如下：

1. 配置 LEDC_TIMER_SEL_CH*n* 为 PWM 生成器选择该定时器
2. 置位 LEDC_OVF_CNT_EN_CH*n* 使能溢出次数计数器
3. 把 LEDC_OVF_NUM_CH*n* 的值设为触发中断的计数器溢出次数减 1
4. 置位 LEDC_OVF_CNT_CH*n*_INT_ENA 使能溢出中断
5. 配置 LEDC_TIMER*x*_DUTY_RES 为所选的定时器 Timer*x* 指定计数器的位宽，等待 LEDC_OVF_CNT_CH*n*_INT 中断产生

在运行时改变计数器的最大值，需先配置 LEDC_TIMER*x*_DUTY_RES 字段，然后置位 LEDC_TIMER*x*_PARA_UP 字段。新的配置在计数器下一次溢出时生效。如果重新配置 LEDC_OVF_CNT_EN_CH*n* 字段，需置位 LEDC_PARA_UP_CH*n* 应用新配置。总之，更改配置时需置位 LEDC_TIMER*x*_PARA_UP 或 LEDC_PARA_UP_CH*n* 应用新配置。

LEDC_TIMER*x*_PARA_UP 和 LEDC_PARA_UP_CH*n* 字段由硬件自动清除。

如图 40.3-2 所示，PWM 生成器输出信号 sig_out*n* 的频率取决于定时器时钟源 LEDC_CLK 的频率、时钟分频系数 LEDC_CLK_DIV 以及占空比精度（计数器位宽）LEDC_TIMER*x*_DUTY_RES：

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}}}{\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}} \tag{40.2}$$

上述公式变形后，可得到以下公式计算预期的占空比精度：

$$\text{LEDC\_TIMER}x\text{\_DUTY\_RES} = \log_2\left(\frac{f_{\text{LEDC\_CLK}}}{f_{\text{PWM}} \cdot \text{LEDC\_CLK\_DIV}}\right) \tag{40.3}$$

表 40.4-1 列出了常用配置频率及其对应精度。

表 40.4-1. 常用配置频率及精度

LEDC_CLK	PWM 频率	最高精度（位）[1]	最低精度（位）[2]
REF_80M_CLK (80 MHz)	1 kHz	16	6
REF_80M_CLK (80 MHz)	5 kHz	13	3
REF_80M_CLK (80 MHz)	10 kHz	12	2
XTAL_48M_CLK (48 MHz)	1 kHz	15	6
XTAL_48M_CLK (48 MHz)	4 kHz	13	4
FOSC_20M_CLK (20 MHz)	1 kHz	14	4
FOSC_20M_CLK (20 MHz)	2 kHz	13	3

[1] 最高精度指时钟分频系数 LEDC_CLK_DIV 为 1 时的精度。如果经公式计算出的最高精度超过了计数器位宽 20 位，则最高精度为 20。

[2] 最低精度指时钟分频系数 LEDC_CLK_DIV 为 $1023 + \frac{255}{256}$ 时的精度。如果经公式计算出的最低精度小于 0，则最低精度为 1。

40.4.2 PWM 生成器

要生成 PWM 信号，PWM 生成器 (PWM*n*) 需选择一个定时器 (Timer*x*)。每个 PWM 生成器均可通过置位 LEDC_TIMER_SEL_CH*n* 单独配置，在 4 个定时器中选择一个用于生成 PWM 信号。

如图 40.3-2 所示，每个 PWM 生成器主要包括一个高低电平比较器和两个选择器。PWM 生成器将定时器的 20 位计数值 (Timer*x*_cnt) 与高低电平比较器的值 Hpoint*n* 和 Lpoint*n* 比较。如果定时器的计数值等于 Hpoint*n* 或 Lpoint*n*，PWM 信号可以输出高低电平：

- 如果 Timer*x*_cnt == Hpoint*n*，则 sig_out*n* 为 1。
- 如果 Timer*x*_cnt == Lpoint*n*，则 sig_out*n* 为 0。

图 40.4-3 展示了如何使用 Hpoint*n* 和 Lpoint*n* 生成占空比固定的 PWM 信号。

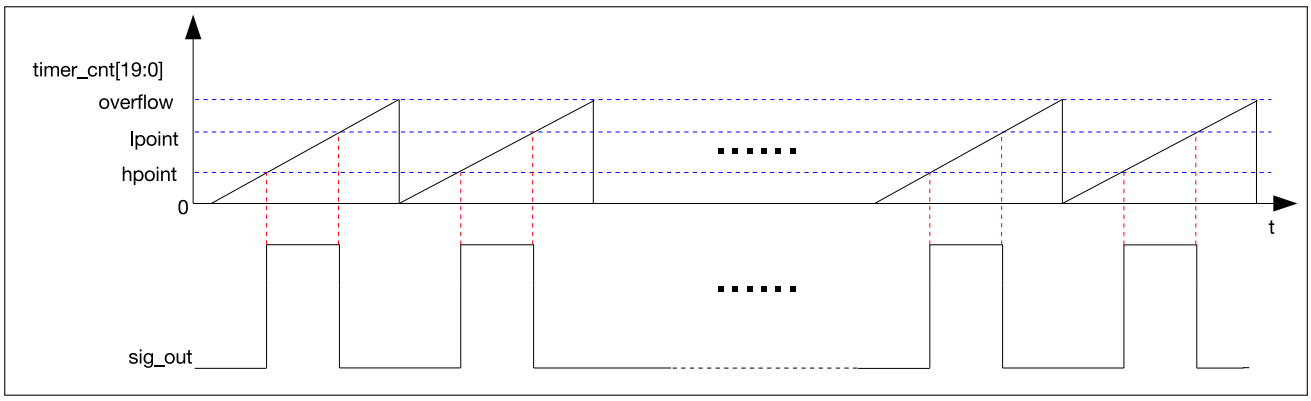


图 40.4-3. LED PWM 输出信号图

每当所选定时器的计数器溢出时，PWM 生成器 (PWM*n*) 的 Hpoint*n* 值更新为 LEDC_HPOINT_CH*n*。Lpoint*n* 的值同样在计数器每次溢出时更新，为 LEDC_DUTY_CH*n*[24:4] 和 LEDC_HPOINT_CH*n* 的和。通过配置 LEDC_DUTY_CH*n*[24:4] 和 LEDC_HPOINT_CH*n* 两个字段可对 Hpoint*n* 和 Lpoint*n* 进行设置，进而可设置 PWM 输出信号的相对相位和占空比。

置位 LEDC_SIG_OUT_EN_CH*n*，开启 PWM 信号 (sig_out*n*) 输出；清除 LEDC_SIG_OUT_EN_CH*n*，关闭 PWM 信号输出，输出信号 (sig_out*n*) 输出恒定电平，电平值为 LEDC_IDLE_LV_CH*n*。

LEDC_DUTY_CH*n*[3:0] 通过周期性改变 PWM 输出信号 sig_out*n* 的占空比实现微调。如 LEDC_DUTY_CH*n*[3:0] 不为 0，那么 sig_out*n* 每 16 个周期中，有 LEDC_DUTY_CH*n*[3:0] 个周期的 PWM 脉冲占空比要比 (16 - LEDC_DUTY_CH*n*[3:0]) 个周期的脉冲占空比多一个定时器的计数周期。比如，如果 LEDC_DUTY_CH*n*[24:4] 设为 10，LEDC_DUTY_CH*n*[3:0] 设为 5，则 16 个周期中，有 5 个周期的 PWM 脉冲占空比为 11，剩余 11 个周期的 PWM 脉冲占空比为 10。16 个周期的平均占空比为 10.3125。

如果重新配置 LEDC_TIMER_SEL_CH*n*、LEDC_HPOINT_CH*n*、LEDC_DUTY_CH*n*[24:4] 和 LEDC_SIG_OUT_EN_CH*n* 字段，需置位 LEDC_PARA_UP_CH*n* 应用新配置。新配置在计数器下次溢出时生效。LEDC_PARA_UP_CH*n* 字段由硬件自动清除。

40.4.3 占空比渐变

PWM 生成器可以渐变 PWM 输出信号的占空比（即由一种占空比逐渐变为为另一种占空比）。每个 PWM 生成器最多可以包含 16 个占空比渐变区间，每个区间可以独立配置占空比渐变方向（增加或减少）、渐变步长、渐变次数和渐变频率。如果开启占空比渐变功能，每个区间的 Lpoint*n* 的值会根据该区间的渐变配置进行变化。

请注意在占空比渐变过程中，如果 PWM 输出信号 (PWM*n*) 的占空比达到 100%，则应将 LEDC_HPOINT_CH*n* 设置为 0，以确保 Lpoint*n* 正确。

40.4.3.1 线性占空比渐变

通过配置第一个占空比渐变区间的渐变方向、渐变步长、渐变次数和渐变频率，可以产生线性占空比渐变 PWM 信号。

配置步骤请见 40.7。

当上述配置步骤完成后，每当计数器溢出 LEDC_CH*n*_GAMMA_DUTY_CYCLE 次后，PWM 生成器将 PWM 信号的占空比渐变一次。PWM 信号每次渐变时，其 Lpoint*n* 会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）LEDC_CH*n*_GAMMA_SCALE，相应地占空比会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）。

$$\frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} \tag{40.4}$$

PWM 信号的占空比会渐变 LEDC_CH*n*_GAMMA_DUTY_NUM 次。完成该次数的渐变后，PWM 信号的占空比停止变化并保持以该占空比输出。由于每次渐变时占空比增加/减少的值都相同，所以产生的 PWM 信号为线性占空比渐变信号。

图 40.4-4 展示了一个 PWM 信号占空比线性渐变的例子。

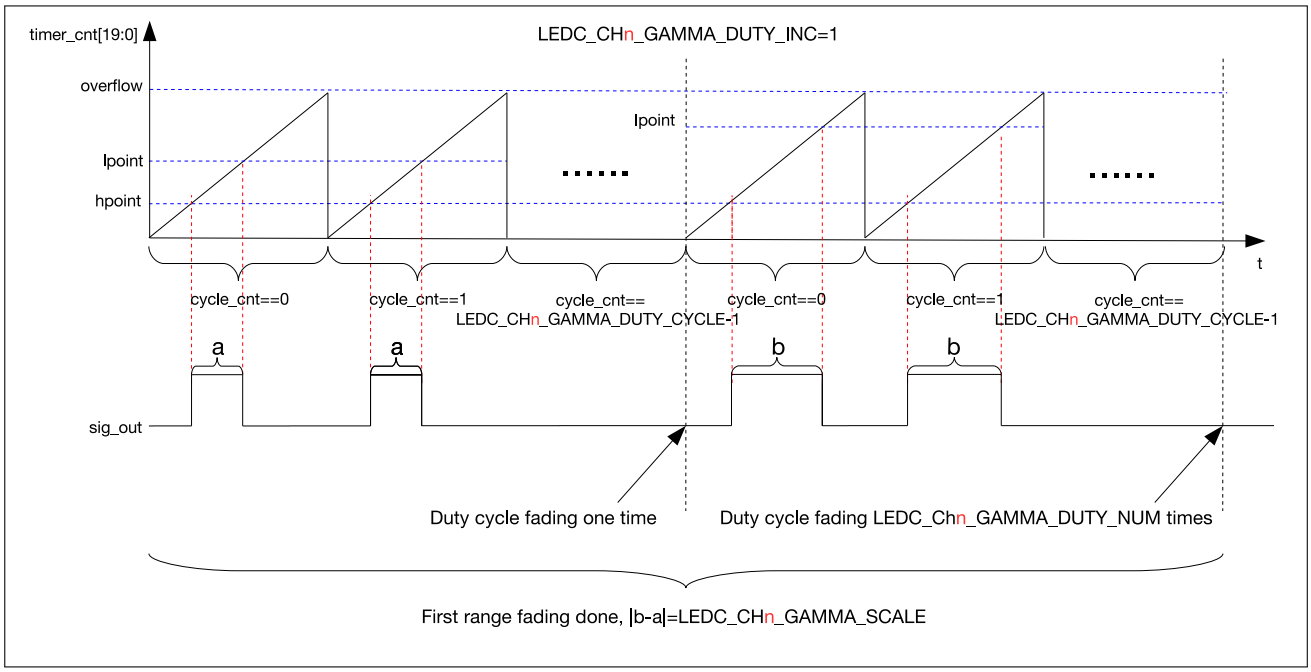


图 40.4-4. 输出信号占空比线性渐变图

40.4.3.2 伽马曲线渐变

通过配置多个占空比渐变区间的占空比渐变方向、渐变步长、渐变次数和渐变频率，PWM 生成器可以产生伽马曲线渐变的 PWM 信号。

配置步骤请见 40.7。

经过上述所有的配置步骤，PWM 生成器会产生包含 LEDC_CH*n*_GAMMA_ENTRY_NUM 个占空比渐变区间的 PWM 信号。首先，PWM 信号会根据序号为 0 的占空比渐变区间的配置进行占空比渐变，渐变完成后，按照序号为 1 的占空比渐变区间的配置进行占空比渐变，直到序号为 (LEDC_CH*n*_GAMMA_ENTRY_NUM − 1) 的占空比渐变区间（即最后一个区间）完成渐变。在每个区间中，PWM 信号都是独立变化的。对于每个区间，PWM 信号会在计数器每溢出 LEDC_CH*n*_GAMMA_DUTY_CYCLE 次后，进行一次占空比渐变，对于每次渐变，其 Lpoint*n* 会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）LEDC_CH*n*_GAMMA_SCALE，相对的，占空比会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 配置）

$$\frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} \tag{40.5}$$

在该区间内，占空比会渐变 LEDC_CH*n*_GAMMA_DUTY_NUM 次，当指定次数的占空比渐变完成后，视为当前区间的占空比渐变完成。

当 LEDC_CHn_GAMMA_ENTRY_NUM 个区间的占空比渐变都完成后，PWM 信号会停止占空比渐变，其占空比会保持最后一次渐变的占空比。由于每个占空比渐变区间的渐变过程都是独立不同的线性渐变，因此多个区间的线性占空比渐变就可以拟合成一个特定的伽马曲线占空比渐变。

图 40.4-5 展示了一个伽马曲线渐变 PWM 信号。

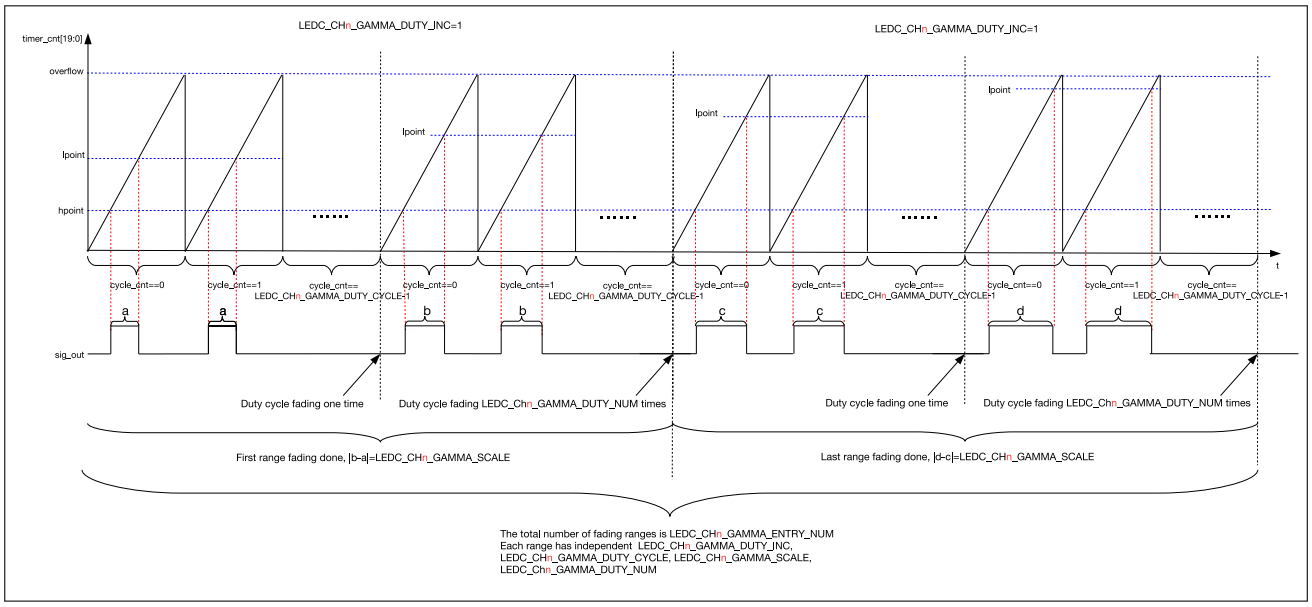


图 40.4-5. 输出信号占空比伽马曲线渐变图

40.4.3.3 占空比渐变暂停和恢复

向 LEDC_CHn_GAMMA_CONF_REG 寄存器的 LEDC_CHn_GAMMA_PAUSE 字段写 1 可以暂停已经开始的占空比渐变过程。一旦将 LEDC_CHn_GAMMA_PAUSE 字段置 1，PWM 信号将会保持最近一次渐变后的占空比。

向 LEDC_CHn_GAMMA_CONF_REG 寄存器的 LEDC_CHn_GAMMA_RESUME 字段写 1 可以恢复被暂停的占空比渐变过程。一旦将 LEDC_CHn_GAMMA_RESUME 字段置 1，PWM 信号将会按照暂停时所在的占空比渐变区间的配置，从暂停时的占空比继续进行占空比渐变，直到所有有效的区间都完成指定的占空比渐变（LEDC_CHn_GAMMA_RESUME 字段置 1 后，硬件会自动清除 LEDC_CHn_GAMMA_PAUSE 字段）。

40.5 事件任务矩阵功能

在 ESP32-C5 中，LEDC 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 LEDC 的 ETM 任务，或者通过 LEDC 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与 LEDC 相关的 ETM 任务和 ETM 事件。

ETM 外设相关的事件和任务的使能由寄存器 LEDC_EVT_TASK_EN0_REG、LEDC_EVT_TASK_EN1_REG 和 LEDC_EVT_TASK_EN2_REG 的相关字段所配置。事件和任务与寄存器 LEDC_EVT_TASK_EN0_REG、LEDC_EVT_TASK_EN1_REG、LEDC_EVT_TASK_EN2_REG 的每个字段的对应关系参照章节 40.9。

LEDC 可接收的 ETM 任务有：

- LEDC_TASK_DUTY_SCALE_UPDATE_CHn：如果 LEDC_TASK_DUTY_SCALE_UPDATE_CHn_EN 字段使能，当接收到 LEDC_TASK_DUTY_SCALE_UPDATE_CHn 任务后，相应的 LEDC 通道（PWMn）将会根据最新配置的 LEDC_CHn_GAMMA_SCALE 的值，重新开始生成占空比渐变 PWM 信号。

- LEDC_TASK_TIMER*x*_RES_UPDATE：如果 LEDC_TASK_TIMER*x*_RES_UPDATE_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RES_UPDATE 任务后，相应的定时器（Timer*x*）将会根据最新配置的 LEDC_TIMER*x*_DUTY_RES 字段的值，在下一次计数器溢出时，更新计数器的最大计数值为该值。
- LEDC_TASK_TIMER*x*_CAP：如果 LEDC_TASK_TIMER*x*_CAP_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_CAP 任务时，相应的定时器（Timer*x*）将会抓取当前计数器的计数值。该计数值将会存储在寄存器 LEDC_TIMER*x*_CNT_CAP_REG 的 LEDC_TIMER*x*_CNT_CAP 字段。
- LEDC_TASK_SIG_OUT_DIS_CH*n*：如果 LEDC_TASK_SIG_OUT_DIS_CH*n*_EN 字段使能，当接收到 LEDC_TASK_SIG_OUT_DIS_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）的 PWM 信号将停止输出，该通道的输出信号 (sig_out*n*) 将会变为 LEDC_IDLE_LV_CH*n* 字段所配置电平的固定电平信号，如图 40.3-2 所示。
- LEDC_TASK_OVF_CNT_RST_CH*n*：如果 LEDC_TASK_OVF_CNT_RST_CH*n*_EN 字段使能，当接收到 LEDC_TASK_OVF_CNT_RST_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）定时器的计数器溢出次数的计数器值将会被复位为 0。
- LEDC_TASK_TIMER*x*_RST：如果 LEDC_TASK_TIMER*x*_RST_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RST 任务时，相应的定时器（Timer*x*）的计数器值将会被复位为 0。
- LEDC_TASK_TIMER*x*_RESUME 和 LEDC_TASK_TIMER*x*_PAUSE：如果 LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RESUME 和 LEDC_TASK_TIMER*x*_PAUSE 任务时，相应的定时器（Timer*x*）将会被交替暂停或恢复。即，当该任务被接收一次时，定时器（Timer*x*）将会暂停，当该任务再被接收一次时，定时器（Timer*x*）将会恢复。
- LEDC_TASK_GAMMA_RESTART_CH*n*：如果 LEDC_TASK_GAMMA_RESTART_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_RESTART_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会重新开始产生占空比渐变 PWM 信号。
- LEDC_TASK_GAMMA_PAUSE_CH*n*：如果 LEDC_TASK_GAMMA_PAUSE_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_PAUSE_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会在计数器下一次溢出时停止占空比的渐变。即，当该任务被接收后，相应的 LEDC 通道（PWM*n*）的 PWM 信号的占空比将不会再进行渐变。
- LEDC_TASK_GAMMA_RESUME_CH*n*：如果 LEDC_TASK_GAMMA_RESUME_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_RESUME_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会在计数器下一次溢出时恢复占空比的渐变。即，当该任务被接收后，相应的 LEDC 通道（PWM*n*）的 PWM 信号将会按照暂停时的占空比渐变配置继续进行占空比的渐变。

LEDC 可产生的 ETM 事件有：

- LEDC_EVT_DUTY_CHNG_END_CH*n*：如果 LEDC_EVT_DUTY_CHNG_END_CH*n*_EN 字段使能，当相应的 LEDC 通道（PWM*n*）占空比渐变完成后，将会产生 LEDC_EVT_DUTY_CHNG_END_CH*n* 事件。
- LEDC_EVT_OVF_CNT_PLS_CH*n*：如果 LEDC_EVT_OVF_CNT_PLS_CH*n*_EN 字段使能，当相应的 LEDC 通道（PWM*n*）的定时器的计数器溢出 LEDC_OVF_NUM_CH*n* + 1 次后，将会产生 LEDC_EVT_OVF_CNT_PLS_CH*n* 事件。
- LEDC_EVT_TIME_OVF_TIMER*x*：如果 LEDC_EVT_TIME_OVF_TIMER*x*_EN 字段使能，当相应的定时器（Timer*x*）的计数器溢出时，将会产生 LEDC_EVT_TIME_OVF_TIMER*x* 事件。
- LEDC_EVT_TIMER*x*_CMP：如果 LEDC_EVT_TIMER*x*_CMP_EN 字段使能，当相应的定时器（Timer*x*）的计数器值等于寄存器 LEDC_TIMER*x*_CMP_REG 的 LEDC_TIMER*x*_CMP 字段的值时，将会产生 LEDC_EVT_TIMER*x*_CMP 事件。

在具体应用中，LEDC 的 ETM 事件可以用来触发 LEDC 的 ETM 任务，例如，LEDC_EVT_DUTY_CHNG_END_CH*n* 事件可以触发 LEDC_TASK_GAMMA_RESTART_CH*n* 任务，从而在一次渐变完成后直接开始下一次渐变。

40.6 中断

ESP32-C5 的 LEDC 模块会产生 LEDC_INT 中断信号，该信号会被发送到 中断矩阵。

下列内部中断源可以产生 LEDC_INT 中断信号：

- LEDC_OVF_CNT_CH*n*_INT：定时器计数器溢出 LEDC_OVF_NUM_CH*n* + 1 次且寄存器 LEDC_OVF_CNT_EN_CH*n* 置 1 时触发中断。
- LEDC_DUTY_CHNG_END_CH*n*_INT：PWM*n* 渐变完成后触发中断。
- LEDC_TIMER*x*_OVF_INT：定时器达到最大计数值时触发中断。

只有当中断使能寄存器 *XXX*_ENA 置位为 1 时，才能触发上述中断源。置位 *XXX*_CLR 可清除中断（*XXX* 为中断源）。

40.7 配置流程

LED PWM 控制器生成 PWM 信号的配置流程为：

1. 参照 40.4.1 对定时器 Timer*x* 进行配置。配置完成后，时钟源、分频器与计数器就配置完成了。
2. 参照 40.4.2 对 PWM*n* 生成器进行配置。配置完成后，PWM*n* 从四个定时器中选择其中一个，并设置 PWM*n* 输出的相位和占空比。
3. 根据不同的占空比渐变类型选择相应的配置过程：
   (a) 线性占空比渐变：
      i. 配置寄存器字段 LEDC_DUTY_CH*n*。该字段用来设置 Lpoint*n* 的初始值。
      ii. 置位寄存器字段 LEDC_DUTY_START_CH*n*。当该字段置位/清除，占空比渐变会使能/关闭。
      iii. 将线性占空比渐变配置写入寄存器（LEDC_CH*n*_MEM 起始地址 + 占空比区间值 * 4），LEDC_CH*n*_MEM 起始地址包含在表 40.8-1 中。

      请注意以下的字段名称只是为了方便叙述，寄存器中并不存在这些字段。

      A. 第 0 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_INC）用于配置方向。当该字段置位或清零时，当前所配置的占空比渐变区间的 Lpoint*n* 会增加/减少。

      B. 第 1 ~ 10 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_CYCLE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每增加/减少一次，计数器溢出的次数。即当计数器溢出次数为配置次数，Lpoint*n*将会增加/减少一次。

      C. 第 11 ~ 20 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_SCALE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每次增加/减少的量。

      D. 第 21 ~ 30 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_NUM）用来设置当前所配置的占空比渐变区间的渐变次数。

E. 占空比范围值（从 0 到 15）用于指定上述配置的属性属于哪个占空比渐变区间。由于线性占空比渐变只需要配置第一个占空比渐变区间即可，所以占空比范围值应配置为 0。

iv. 将寄存器 LEDC_CH*n*_GAMMA_CONF_REG 的 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为 1。该字段用来指定一次渐变过程中有效的占空比渐变区间的个数（当指定数目的区间都渐变完成后，PWM 信号会停止占空比渐变，PWM 生成器会触发 LEDC_DUTY_CHNG_END_CH*n*_INT 中断）。由于线性占空比渐变只包含一个渐变区间（第一个占空比渐变区间），所以 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为 1。

v. 置位寄存器字段 LEDC_PARA_UP_CH*n* 应用上述步骤的配置。当该字段被置位后，上述的占空比渐变配置会在下一次计数器溢出时生效，PWM 生成器会输出指定配置的线性占空比渐变 PWM 信号（LEDC_PARA_UP_CH*n* 字段会被硬件自动清除）。

(b) 伽马曲线渐变：

i. 与章节 40.4.3.1 的步骤 1 相同。

ii. 与章节 40.4.3.1 的步骤 2 相同。

iii. 配置多个占空比渐变区间：将伽马曲线渐变配置写入寄存器（LEDC_CH*n*_MEM 起始地址 + 占空比区间值 * 4），LEDC_CH*n*_MEM 起始地址包含在表 40.8-1 中。

A. 第 0 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_INC）用于配置方向。当该字段置位或清零时，当前所配置的占空比渐变区间的 Lpoint*n* 会增加/减少。

B. 第 1 ~ 10 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_CYCLE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每增加/减少一次，计数器溢出的次数。即当计数器溢出次数为配置次数，Lpoint*n*将会增加/减少一次。

C. 第 11 ~ 20 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_SCALE）用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每次增加/减少的量。

D. 第 21 ~ 30 位（为方便叙述，以下称为 LEDC_CH*n*_GAMMA_DUTY_NUM）用来设置当前所配置的占空比渐变区间的渐变次数。

E. 占空比范围值 (0 ~ 15) 用于指定上述配置的属性属于哪个占空比渐变区间。对于伽马曲线渐变，所配置的占空比渐变区间的序号必须从 0 开始，每配置完一个占空比渐变区间，下一个要配置的占空比渐变区间的序号需要加 1。

F. 上述步骤配置完毕后，一个占空比渐变区间的配置视为完成，然后可以重复上述的步骤配置其他的占空比渐变区间的参数。每个占空比渐变区间的配置都是独立的，拥有有效配置的区间个数 (0 ~ 15) 可以是任意的。

iv. 当所需的占空比渐变区间都配置完毕后，将 LEDC_CH*n*_GAMMA_CONF_REG 寄存器的 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为步骤 3 中配置过的占空比渐变区间的总个数。

v. 置位寄存器的 LEDC_PARA_UP_CH*n* 字段，应用以上步骤的配置。当该字段被置位后，上述步骤的占空比渐变配置会在下一次计数器溢出时生效，PWM 生成器会按照配置输出伽马曲线渐变的 PWM 信号（LEDC_PARA_UP_CH*n* 字段会自动被硬件清除）。

按照上述步骤配置完成后，LED PWM 控制器会生成所需的 PWM 信号。

占空比渐变随时都可以暂停或恢复，详情请参考 40.4.3.3。如果需要在 PWM 信号停止渐变之前（即 LEDC_DUTY_CHNG_END_CH*n*_INT 尚未触发）更改占空比渐变配置，则需要在更改配置之前暂停占空比渐变。但是，如果需要在 PWM 信号停止渐变后（即 LEDC_DUTY_CHNG_END_CH*n*_INT 已被触发）更改占空比渐变配置，则可以直接进行更改。

软件随时都可以启用或禁用 PWM 信号输出，也可对输出信号的电平进行更改。详情请参考 40.4.2。

40.8 存储空间

每个 LEDC 通道都有一块存储空间用于存放占空比渐变的配置数据。每个存储空间最多可储存 16 个区间的配置数据。本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

表 40.8-1. LEDC 存储空间

名称	描述	大小	起始地址	终止地址	访问权限
LEDC_CH*n*_MEM	PWM*n* 内存	64 B	0x400 + PWM*n* * 64	0x400 + (PWM*n* + 1) * 64	R/W

40.9 寄存器列表

本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LEDC_CH0_CONF0_REG	通道 0 的配置寄存器 0	0x0000	varies
LEDC_CH0_HPOINT_REG	通道 0 的 Hpoint 寄存器	0x0004	R/W
LEDC_CH0_DUTY_REG	通道 0 的初始占空比	0x0008	R/W
LEDC_CH0_CONF1_REG	通道 0 的配置寄存器 1	0x000C	R/W/SC
LEDC_CH1_CONF0_REG	通道 1 的配置寄存器 0	0x0014	varies
LEDC_CH1_HPOINT_REG	通道 1 的 Hpoint 寄存器	0x0018	R/W
LEDC_CH1_DUTY_REG	通道 1 的初始占空比	0x001C	R/W
LEDC_CH1_CONF1_REG	通道 1 的配置寄存器 1	0x0020	R/W/SC
LEDC_CH2_CONF0_REG	通道 2 的配置寄存器 0	0x0028	varies
LEDC_CH2_HPOINT_REG	通道 2 的 Hpoint 寄存器	0x002C	R/W
LEDC_CH2_DUTY_REG	通道 2 的初始占空比	0x0030	R/W
LEDC_CH2_CONF1_REG	通道 2 的配置寄存器 1	0x0034	R/W/SC
LEDC_CH3_CONF0_REG	通道 3 的配置寄存器 0	0x003C	varies
LEDC_CH3_HPOINT_REG	通道 3 的 Hpoint 寄存器	0x0040	R/W
LEDC_CH3_DUTY_REG	通道 3 的初始占空比	0x0044	R/W
LEDC_CH3_CONF1_REG	通道 3 的配置寄存器 1	0x0048	R/W/SC
LEDC_CH4_CONF0_REG	通道 4 的配置寄存器 0	0x0050	varies
LEDC_CH4_HPOINT_REG	通道 4 的 Hpoint 寄存器	0x0054	R/W
LEDC_CH4_DUTY_REG	通道 4 的初始占空比	0x0058	R/W
LEDC_CH4_CONF1_REG	通道 4 的配置寄存器 1	0x005C	R/W/SC
LEDC_CH5_CONF0_REG	通道 5 的配置寄存器 0	0x0064	varies
LEDC_CH5_HPOINT_REG	通道 5 的 Hpoint 寄存器	0x0068	R/W
LEDC_CH5_DUTY_REG	通道 5 的初始占空比	0x006C	R/W
LEDC_CH5_CONF1_REG	通道 5 的配置寄存器 1	0x0070	R/W/SC
LEDC_TIMER0_CONF_REG	配置定时器 0	0x00A0	varies
LEDC_TIMER1_CONF_REG	配置定时器 1	0x00A8	varies
LEDC_TIMER2_CONF_REG	配置定时器 2	0x00B0	varies
LEDC_TIMER3_CONF_REG	配置定时器 3	0x00B8	varies
LEDC_CH0_GAMMA_CONF_REG	通道 0 伽马配置寄存器	0x0100	varies
LEDC_CH1_GAMMA_CONF_REG	通道 1 伽马配置寄存器	0x0104	varies
LEDC_CH2_GAMMA_CONF_REG	通道 2 伽马配置寄存器	0x0108	varies
LEDC_CH3_GAMMA_CONF_REG	通道 3 伽马配置寄存器	0x010C	varies
LEDC_CH4_GAMMA_CONF_REG	通道 4 伽马配置寄存器	0x0110	varies
LEDC_CH5_GAMMA_CONF_REG	通道 5 伽马配置寄存器	0x0114	varies
LEDC_EVT_TASK_EN0_REG	LEDC 事件任务使能寄存器 0	0x0120	R/W
LEDC_EVT_TASK_EN1_REG	LEDC 事件任务使能寄存器 1	0x0124	R/W

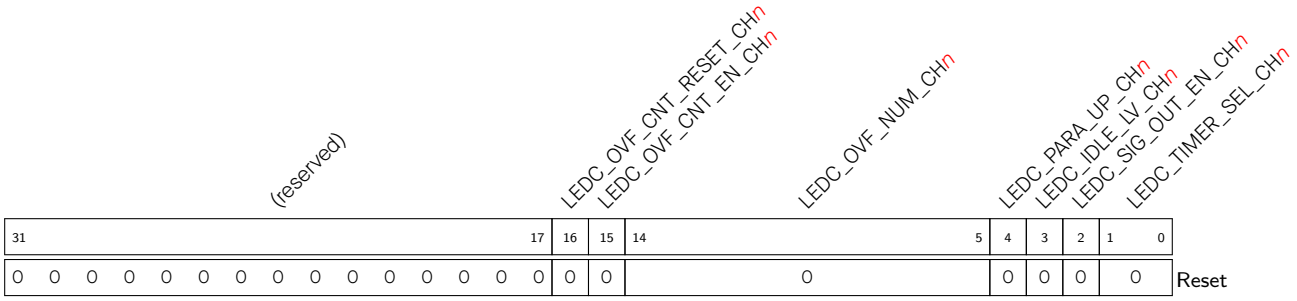
名称	描述	地址	访问
LEDC_EVT_TASK_EN2_REG	LEDC 事件任务使能寄存器 2	0x0128	R/W
LEDC_TIMER0_CMP_REG	定时器 0 的比较值	0x0140	R/W
LEDC_TIMER1_CMP_REG	定时器 1 的比较值	0x0144	R/W
LEDC_TIMER2_CMP_REG	定时器 2 的比较值	0x0148	R/W
LEDC_TIMER3_CMP_REG	定时器 3 的比较值	0x014C	R/W
LEDC_CONF_REG	LEDC 全局配置寄存器	0x0170	R/W
状态寄存器			
LEDC_CH0_DUTY_R_REG	通道 0 的当前占空比	0x0010	RO
LEDC_CH1_DUTY_R_REG	通道 1 的当前占空比	0x0024	RO
LEDC_CH2_DUTY_R_REG	通道 2 的当前占空比	0x0038	RO
LEDC_CH3_DUTY_R_REG	通道 3 的当前占空比	0x004C	RO
LEDC_CH4_DUTY_R_REG	通道 4 的当前占空比	0x0060	RO
LEDC_CH5_DUTY_R_REG	通道 5 的当前占空比	0x0074	RO
LEDC_TIMER0_VALUE_REG	配置定时器 0 的当前计数值	0x00A4	RO
LEDC_TIMER1_VALUE_REG	配置定时器 1 的当前计数值	0x00AC	RO
LEDC_TIMER2_VALUE_REG	配置定时器 2 的当前计数值	0x00B4	RO
LEDC_TIMER3_VALUE_REG	配置定时器 3 的当前计数值	0x00BC	RO
LEDC_TIMER0_CNT_CAP_REG	抓取定时器 0 的计数值	0x0150	RO
LEDC_TIMER1_CNT_CAP_REG	抓取定时器 1 的计数值	0x0154	RO
LEDC_TIMER2_CNT_CAP_REG	抓取定时器 2 的计数值	0x0158	RO
LEDC_TIMER3_CNT_CAP_REG	抓取定时器 3 的计数值	0x015C	RO
中断寄存器			
LEDC_INT_RAW_REG	原始中断寄存器	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	屏蔽中断寄存器	0x00C4	RO
LEDC_INT_ENA_REG	中断使能寄存器	0x00C8	R/W
LEDC_INT_CLR_REG	中断清除寄存器	0x00CC	WT
版本寄存器			
LEDC_DATE_REG	版本控制寄存器	0x0174	R/W

40.10 寄存器

本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 40.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14**n*)



LEDC_TIMER_SEL_CH*n* 配置通道 *n* 选取的定时器。

0：定时器 0

1：定时器 1

2：定时器 2

3：定时器 3

(R/W)

LEDC_SIG_OUT_EN_CH*n* 配置是否开启通道 *n* 的信号输出。

0：关闭

1：开启

(R/W)

LEDC_IDLE_LV_CH*n* 配置通道 *n* 关闭（即 LEDC_SIG_OUT_EN_CH*n* 为 0）时的输出电平值。

0：低电平

1：高电平

(R/W)

LEDC_PARA_UP_CH*n* 配置是否更新以下字段：LEDC_HPOINT_CH*n*、LEDC_DUTY_START_CH*n*、LEDC_SIG_OUT_EN_CH*n*、LEDC_TIMER_SEL_CH*n*、LEDC_OVF_CNT_EN_CH*n* 和占空比区间配置

0：无效值，没有作用

1：更新

该字段由硬件自清

(WT)

LEDC_OVF_NUM_CH*n* 配置定时器的最大溢出次数减 1。

通道 *n* 的定时器溢出次数达到 (LEDC_OVF_NUM_CH*n* + 1 次时，触发 LEDC_OVF_CNT_CH*n*_INT 中断。(R/W)

LEDC_OVF_CNT_EN_CH*n* 配置是否开启通道 *n* 的溢出次数计数器。

0：关闭

1：开启

(R/W)

见下页...

Register 40.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14**n*)

接上页...

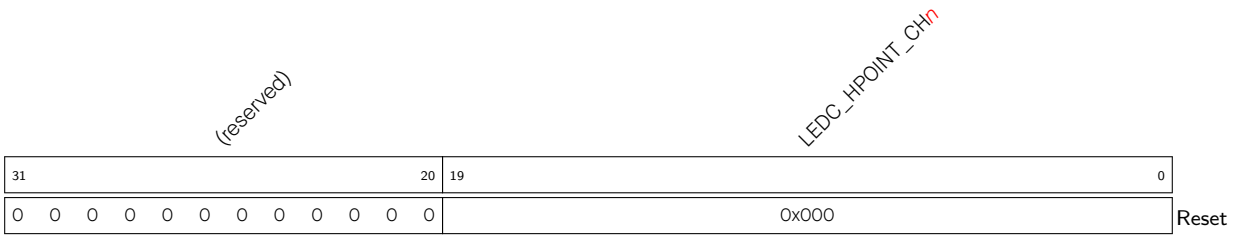
LEDC_OVF_CNT_RESET_CH*n* 配置是否复位通道 *n* 的溢出次数计数器。

0：无效值，没有作用

1：复位

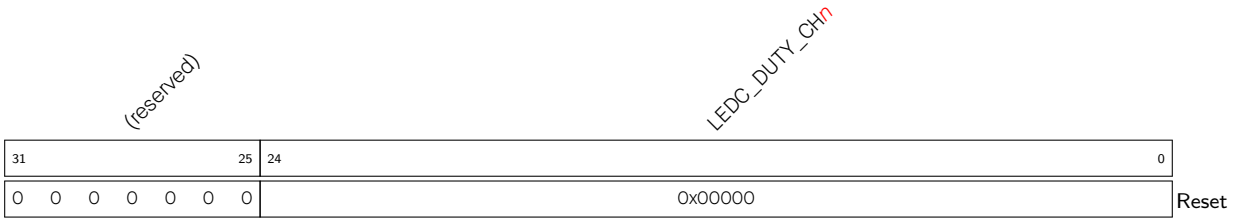
(WT)

Register 40.2. LEDC_CH*n*_HPOINT_REG (*n*: 0-5) (0x0004+0x14**n*)



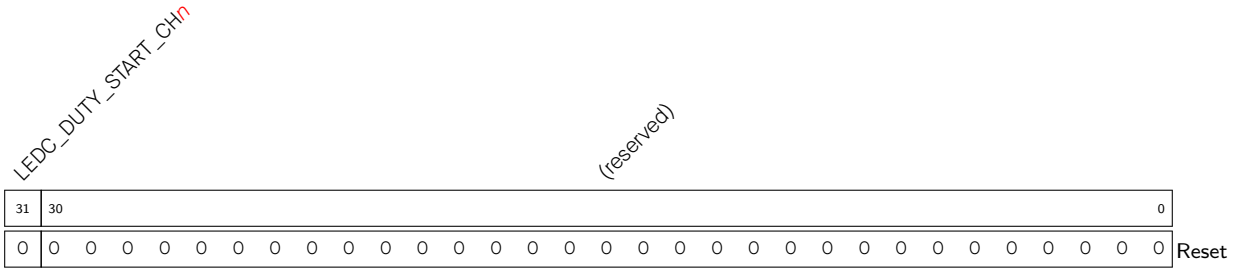
LEDC_HPOINT_CH*n* 配置通道 *n* 输出的 Hpoint 值。当定时器达到该寄存器的规定值时，输出值变为高电平。(R/W)

Register 40.3. LEDC_CH*n*_DUTY_REG (*n*: 0-5) (0x0008+0x14**n*)



LEDC_DUTY_CH*n* 配置通道 *n* 输出信号的占空比。(R/W)

Register 40.4. LEDC_CH*n*_CONF1_REG (*n*: 0-5) (0x000C+0x14**n*)



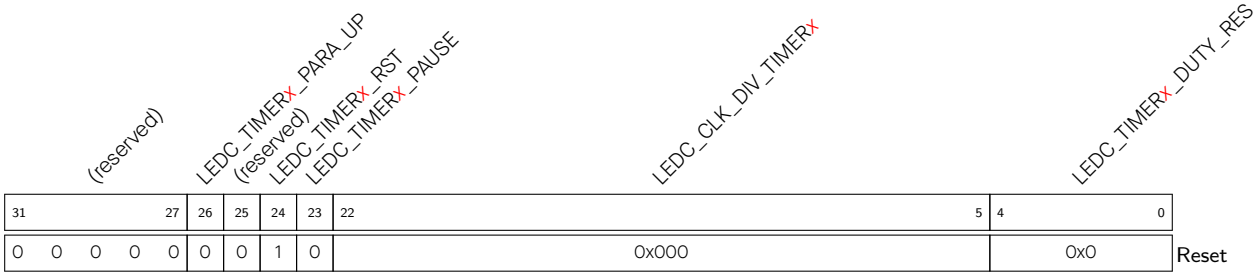
LEDC_DUTY_START_CH*n* 配置占空比渐变配置是否生效。

0：不生效

1：生效

(R/W/SC)

Register 40.5. LEDC_TIMER*x*_CONF_REG (*x*: 0-3) (0x00A0+0x8**x*)



LEDC_TIMER*x*_DUTY_RES 配置定时器 *x* 计数器的位宽，有效值为 1 到 20。(R/W)

LEDC_CLK_DIV_TIMER*x* 配置定时器 *x* 中分频器的分频系数。

低 8 位为小数部分，高 10 位为整数部分。(R/W)

LEDC_TIMER*x*_PAUSE 配置是否暂停定时器 *x* 的计数器。

0：不暂停

1：暂停

(R/W)

LEDC_TIMER*x*_RST 配置是否复位定时器 *x* 的计数器（复位后，计数器的值为 0。）。

0：不复位

1：复位

(R/W)

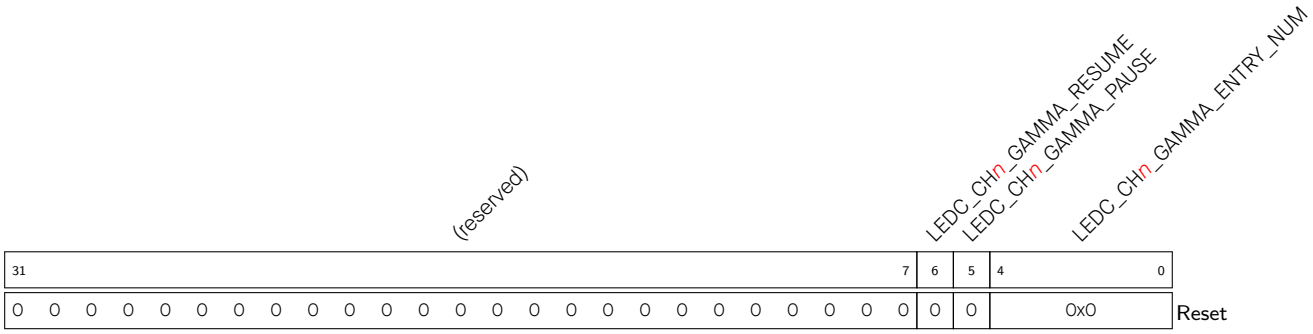
LEDC_TIMER*x*_PARA_UP 配置是否更新 LEDC_CLK_DIV_TIMER*x* 和 LEDC_TIMER*x*_DUTY_RES 字段。

0：无效值，没有作用

1：更新

(WT)

Register 40.6. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-5) (0x0100+0x4**n*)



LEDC_CH*n*_GAMMA_ENTRY_NUM 配置通道 *n* 的占空比渐变区间数量。(R/W)

LEDC_CH*n*_GAMMA_PAUSE 配置是否暂停占空比渐变。

0：无效值，没有作用

1：暂停

(WT)

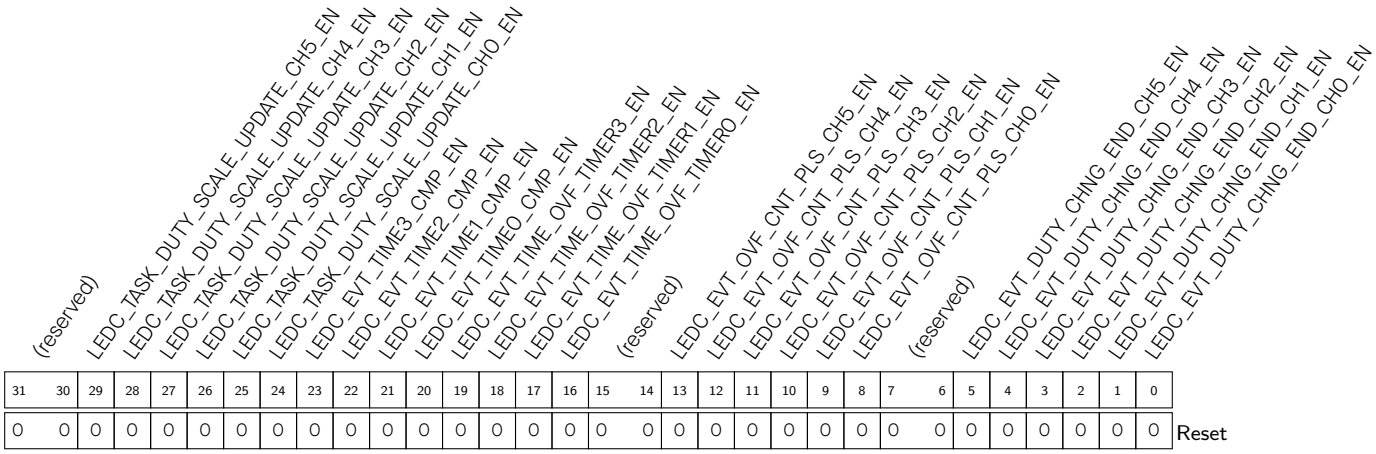
LEDC_CH*n*_GAMMA_RESUME 配置是否恢复已暂停的占空比渐变。

0：无效值，没有作用

1：恢复

(WT)

Register 40.7. LEDC_EVT_TASK_EN0_REG (0x0120)



LEDC_EVT_DUTY_CHNG_END_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_EVT_DUTY_CHNG_END_CH*n* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_OVF_CNT_PLS_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_EVT_OVF_CNT_PLS_CH*n* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_TIME_OVF_TIMER*x*_EN (*x*: 0-3) 配置是否使能 LEDC_EVT_TIME_OVF_TIMER*x* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_TIMER*x*_CMP_EN (*x*: 0-3) 配置是否使能 LEDC_EVT_TIMER*x*_CMP 事件。

0：关闭

1：使能

(R/W)

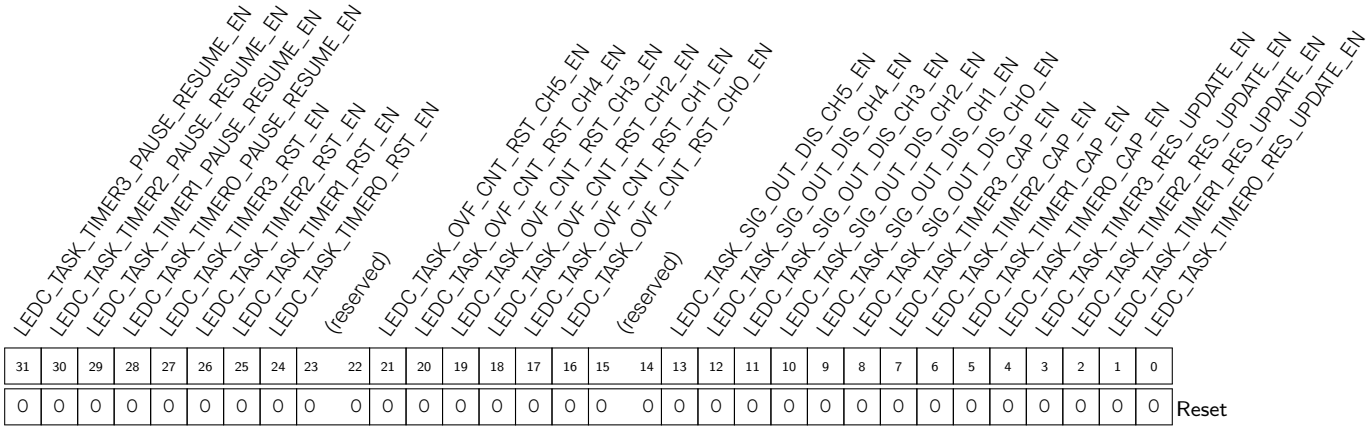
LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* 事件。

0：关闭

1：使能

(R/W)

Register 40.8. LEDC_EVT_TASK_EN1_REG (0x0124)



LEDC_TASK_TIMER*x*_RES_UPDATE_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_RES_UPDATE 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_TIMER*x*_CAP_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_CAP 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_SIG_OUT_DIS_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_TASK_SIG_OUT_DIS_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_OVF_CNT_RST_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_TASK_OVF_CNT_RST_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_TIMER*x*_RST_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_RST 任务。

0：关闭

1：使能

(R/W)

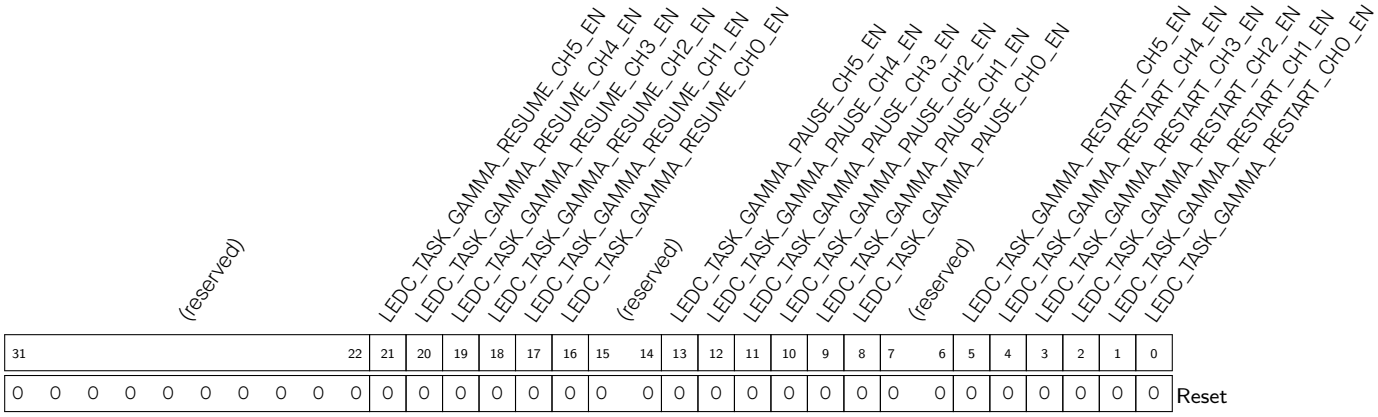
LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN (*x*: 0-3) 配置是否使能 LEDC_TASK_TIMER*x*_PAUSE 和 LEDC_TASK_TIMER*x*_RESUME 任务。

0：关闭

1：使能

(R/W)

Register 40.9. LEDC_EVT_TASK_EN2_REG (0x0128)



LEDC_TASK_GAMMA_RESTART_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_TASK_GAMMA_RESTART_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_GAMMA_PAUSE_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_TASK_GAMMA_PAUSE_CH*n* 任务。

0：关闭

1：使能

(R/W)

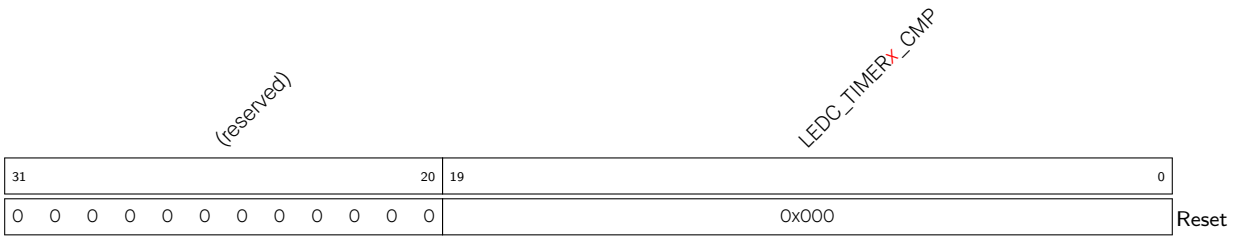
LEDC_TASK_GAMMA_RESUME_CH*n*_EN (*n*: 0-5) 配置是否使能 LEDC_TASK_GAMMA_RESUME_CH*n* 任务。

0：关闭

1：使能

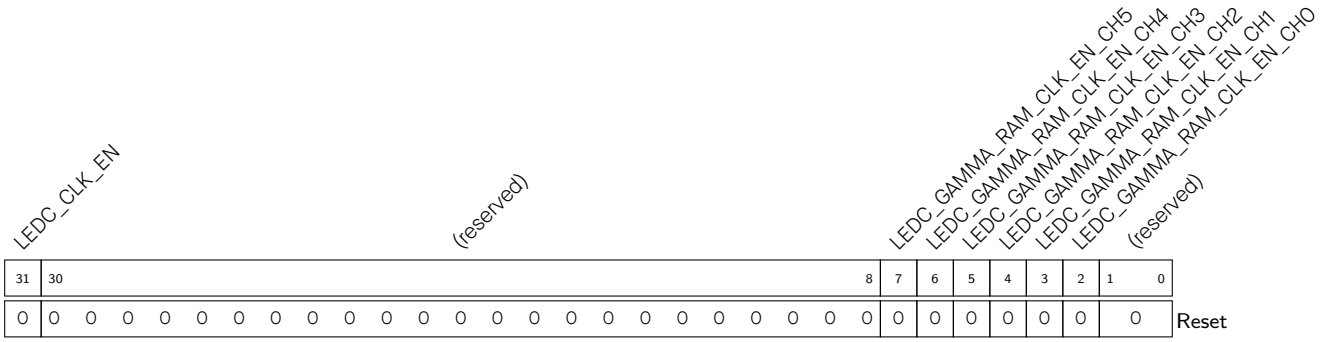
(R/W)

Register 40.10. LEDC_TIMER*x*_CMP_REG (*x*: 0-3) (0x0140+0x4**x*)



LEDC_TIMER*x*_CMP 配置 LEDC 定时器 *x* 的比较值。(R/W)

Register 40.11. LEDC_CONF_REG (0x0170)



LEDC_GAMMA_RAM_CLK_EN_CH*n* (*n*: 0-5) 配置是否开启通道 *n* 伽马 RAM 时钟门控。

0：仅在应用读或写通道 *n* 的伽马 RAM 时开启时钟门控

1：强制为通道 *n* 伽马 RAM 开启时钟门控

(R/W)

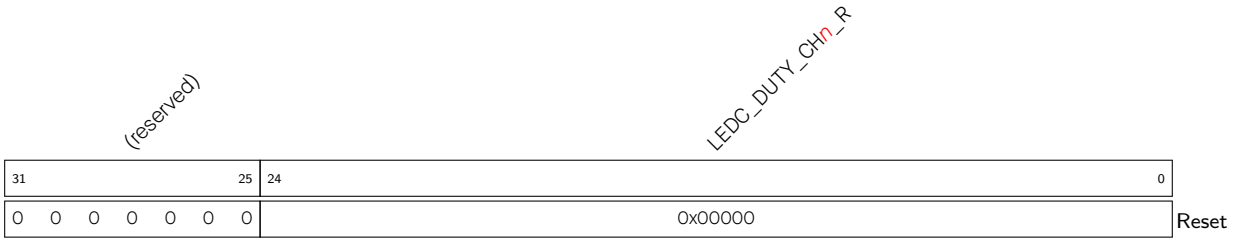
LEDC_CLK_EN 配置是否开启寄存器门控。

0：仅在应用写寄存器时开启时钟门控

1：强制为寄存器开启时钟门控

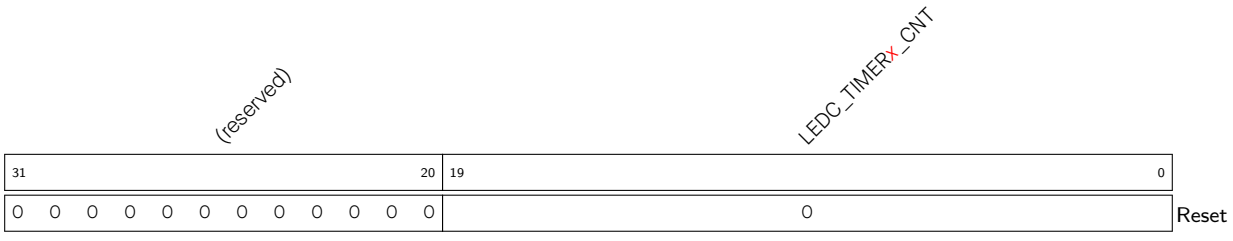
(R/W)

Register 40.12. LEDC_CH*n*_DUTY_R_REG (*n*: 0-5) (0x0010+0x14*n)



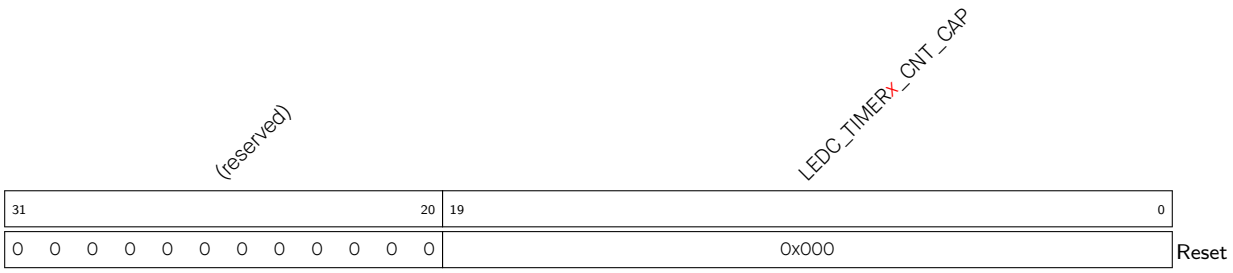
LEDC_DUTY_CH*n*_R 通道 *n* 输出信号的当前占空比。(RO)

Register 40.13. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8*x)



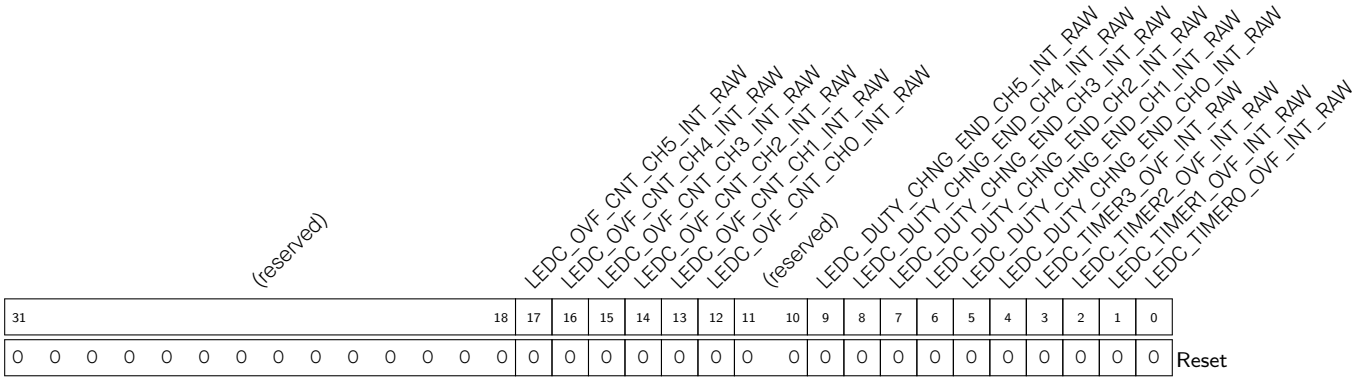
LEDC_TIMER*x*_CNT 定时器 *x* 计数器的当前数值。(RO)

Register 40.14. LEDC_TIMERx_CNT_CAP_REG (x: 0-3) (0x0150+0x4*x)



LEDC_TIMERx_CNT_CAP 抓取的定时器 x 的计数器数值。(RO)

Register 40.15. LEDC_INT_RAW_REG (0x00C0)

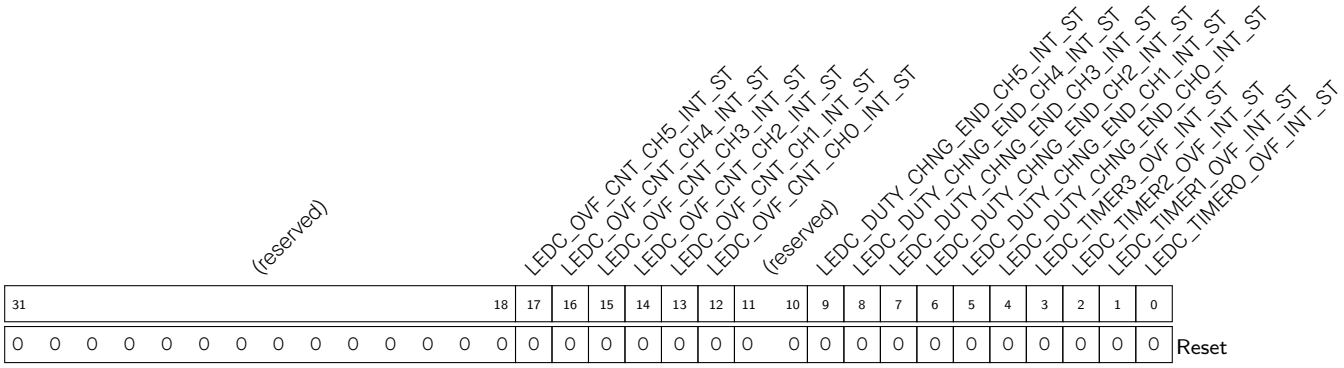


LEDC_TIMERx_OVF_INT_RAW (x: 0-3) 原始状态位：LEDC_TIMERx_OVF_INT 的原始中断状态。定时器 x 的计数器到达最大值时，触发该中断。(R/WTC/SS)

LEDC_DUTY_CHNG_END_CHn_INT_RAW (n: 0-5) 原始状态位：LEDC_DUTY_CHNG_END_CHn_INT 的原始中断状态。占空比渐变结束时会触发该中断。(R/WTC/SS)

LEDC_OVF_CNT_CHn_INT_RAW (n: 0-5) 原始状态位：LEDC_OVF_CNT_CHn_INT 的原始中断状态。当 ovf_cnt 达到 LEDC_OVF_NUM_CHn 的设定值时，触发该中断。(R/WTC/SS)

Register 40.16. LEDC_INT_ST_REG (0x00C4)



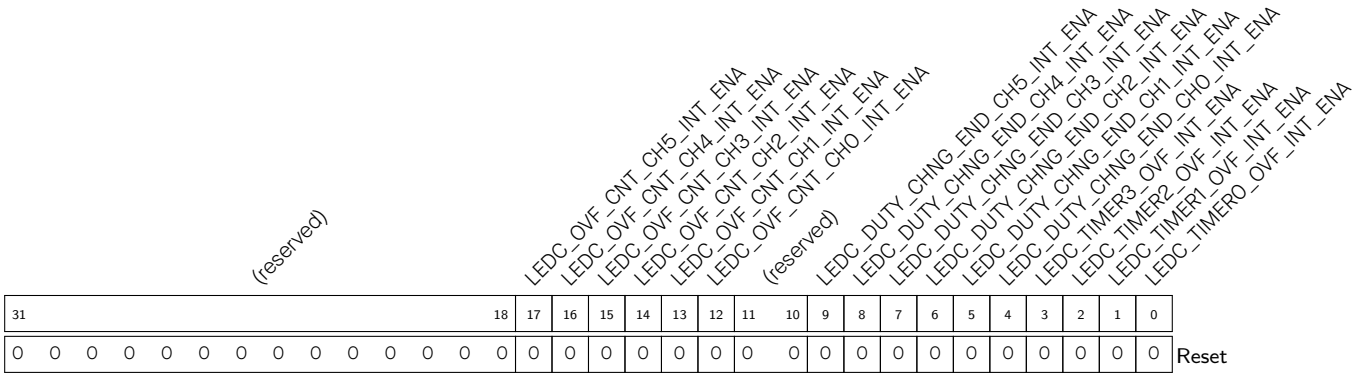
LEDC_TIMER*x*_OVF_INT_ST (*x*: 0-3) 屏蔽状态位：LEDC_TIMER*x*_OVF_INT 的屏蔽中断状态。仅在 LEDC_TIMER*x*_OVF_INT_ENA 为 1 时有效。(RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST (*n*: 0-5) 屏蔽状态位：LEDC_DUTY_CHNG_END_CH*n*_INT 的屏蔽中断状态。

仅在 LEDC_DUTY_CHNG_END_CH*n*_INT_ENA 为 1 时有效。(RO)

LEDC_OVF_CNT_CH*n*_INT_ST (*n*: 0-5) 屏蔽状态位：LEDC_OVF_CNT_CH*n*_INT 的屏蔽中断状态。仅在 LEDC_OVF_CNT_CH*n*_INT_ENA 为 1 时有效。(RO)

Register 40.17. LEDC_INT_ENA_REG (0x00C8)

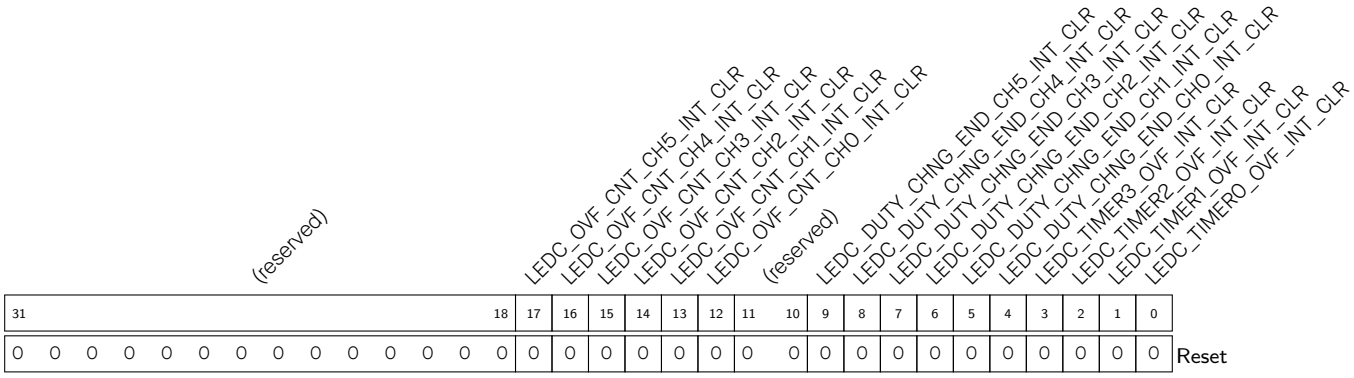


LEDC_TIMER*x*_OVF_INT_ENA (*x*: 0-3) 使能位：写 1 使能 LEDC_TIMER*x*_OVF_INT。(R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA (*n*: 0-5) 使能位：写 1 使能 LEDC_DUTY_CHNG_END_CH*n*_INT。(R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA (*n*: 0-5) 使能位：写 1 使能 LEDC_OVF_CNT_CH*n*_INT。(R/W)

Register 40.18. LEDC_INT_CLR_REG (0x00CC)

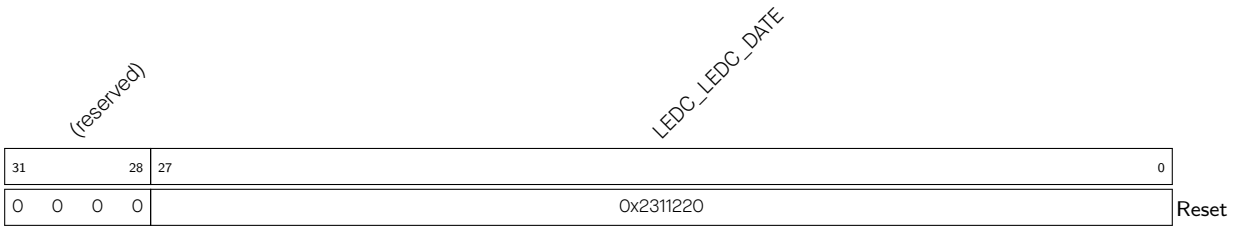


LEDC_TIMER*x*_OVF_INT_CLR (*x*: 0-3) 清零位：写 1 清零 LEDC_TIMER*x*_OVF_INT。(WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR (*n*: 0-5) 清零位：写 1 清零 LEDC_DUTY_CHNG_END_CH*n*_INT。(WT)

LEDC_OVF_CNT_CH*n*_INT_CLR (*n*: 0-5) 清零位：写 1 清零 LEDC_OVF_CNT_CH*n*_INT。(WT)

Register 40.19. LEDC_DATE_REG (0x0174)



LEDC_LEDC_DATE 版本控制寄存器。(R/W)

第 41 章

电机控制脉宽调制器 (MCPWM)

41.1 概述

电机控制脉宽调制器 (MCPWM) 外设用于电机和电源控制。该外设提供了 6 个 PWM 输出，可在几种拓扑结构中运行。常见的拓扑结构之一是用一对 PWM 输出来驱动 H 桥以控制电机旋转速度和旋转方向。

MCPWM 外设主要分为五个子模块：PWM 定时器、PWM 操作器、捕获模块、事件矩阵 (Event Task Matrix, ETM) 模块、故障检测模块。PWM 定时器提供参考时序，可以自由运行，或同步到其他定时器或外部源。每个 PWM 操作器具有生成一个 PWM 通道波形对所需的所有控制资源。捕获模块用于需要精确定时外部事件的系统。ETM 模块对输入到 MCPWM 的任务做出相应的响应动作，根据不同的运动状态产生相应的事件。故障检测模块用于捕获外部故障，使得系统可以选择做出响应。

ESP32-C5 有一个 MCPWM 外设，即 MCPWM0。

41.2 主要特性

MCPWM 外设有一个时钟分频器（预分频器）、三个 PWM 定时器、三个 PWM 操作器、一个捕获模块、一个 ETM 模块和一个故障检测模块。MCPWM 的主时钟可以从三个时钟源中任选其一，分别为：PLL_F160M_CLK, XTAL_CLK 和 RC_FAST_CLK（通过 PCR_PWM_CLK_CONF_REG 寄存器的 PCR_PWM_CLKM_SEL 字段配置）。图 41.2-1 描述了 MCPWM 内的模块和接口上的信号。PWM 定时器用于生成定时参考，PWM 操作器将根据生成的定时参考生成所需的波形。通过配置，任一 PWM 操作器可以使用任一 PWM 定时器的定时参考。不同的 PWM 操作器可以使用相同的 PWM 定时器的定时参考产生 PWM 信号，也可以使用不同的 PWM 定时器的值来生成单独的 PWM 信号。不同的 PWM 定时器可以进行同步。

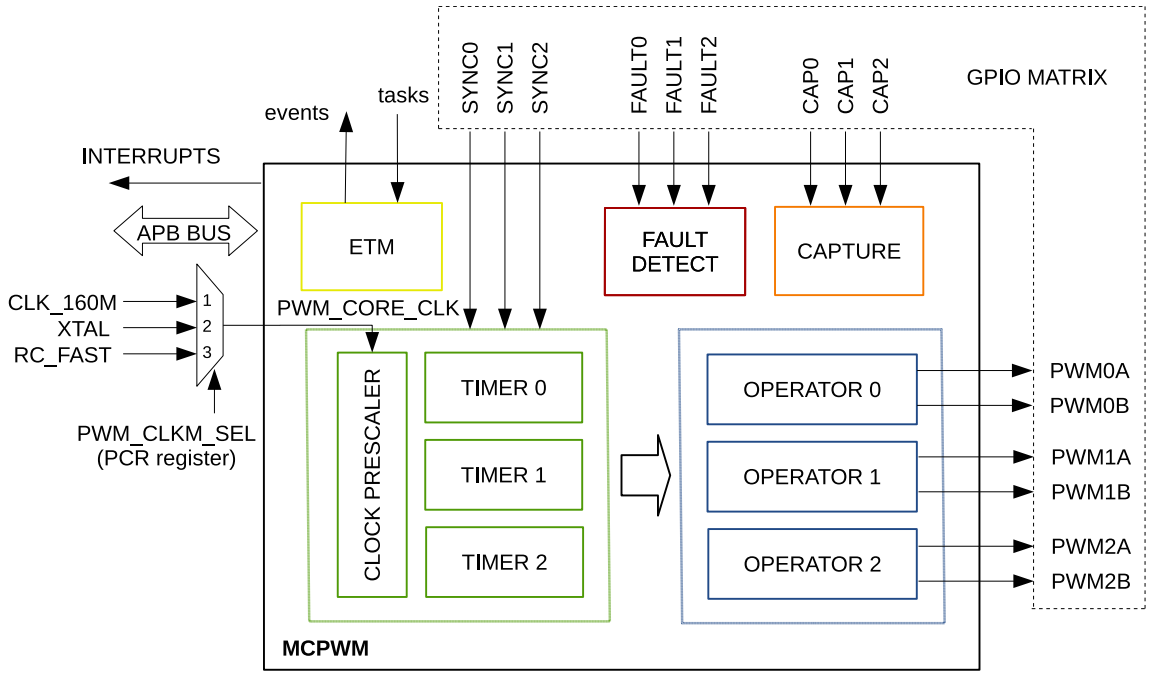


图 41.2-1. MCPWM 外设概览

以下是图 41.2-1 中模块的功能概述：

- PWM 定时器 0、1 和 2：
  - 每个 PWM 定时器都有一个专用的 8 位时钟预分频器。
  - PWM 定时器中的 16 位计数器的工作模式包括：递增计数模式，递减计数模式，递增递减循环计数模式。
  - 硬件/软件同步可以触发 PWM 定时器重载，重载值位于相位寄存器中；同时触发预分频的重启，从而同步定时器的时钟。硬件同步源可以来自任何 GPIO 或任何其他 PWM 定时器的 sync_out 信号。软件同步源来自于向 MCPWM_TIMER*n*_SYNC_SW 位写入取反值。
- PWM 操作器 0、1 和 2：
  - 每个 PWM 操作器有两个 PWM 输出（PWM*x*A 和 PWM*x*B），可以在对称和非对称配置中独立工作。
  - 可以通过异步方式更新对 PWM 信号的控制。
  - 死区时间在上升沿和下降沿可配置，并可分别设置。
  - 所有事件都可触发 CPU 中断。
  - 通过高频载波信号调制 PWM 输出，在使用变压器隔离栅极驱动器时可发挥巨大作用。
  - 周期、时间戳寄存器和其他主要的控制寄存器有影子寄存器，具有灵活的更新方式。
- 故障检测模块：
  - 出现故障时，可选择在逐周期模式或一次性模式下处理。
  - 出现故障时可强制 PWM 输出高或低电平。
- 捕获模块：
  - 捕获模块的时钟为 MCPWM 的主时钟。

  - 旋转电机的速度测量。
  - 位置传感器脉冲之间的间隔时间测量。
  - 脉冲序列信号的周期和占空比测量。
  - 从电流/电压传感器的占空比编码信号导出的解码电流或电压振幅。
  - 3 个独立的捕获通道，各具备一个 32 位的时间戳寄存器。
  - 输入捕获信号可以预分频，边沿极性可选。
  - 捕获定时器可以与 PWM 定时器或外部信号同步。
  - 3 个捕获通道上都可以产生中断。
- ETM 模块：
  - 根据每个定时器和操作器不同的运行状态，产生不同的事件。
  - 每个定时器和操作器响应其对应的任务，自动进行相应的操作。
  - 每个事件和任务都可以独立使能。某个事件不使能时，不产生对应的事件；某个任务不使能时，不响应对应的任务。

41.3 模块

41.3.1 概述

每个 MCPWM 包含的关键模块为定时器模块、操作器模块、故障检测模块、捕获模块和 ETM 模块，关键模块的功能详见下文。

41.3.1.1 定时器模块

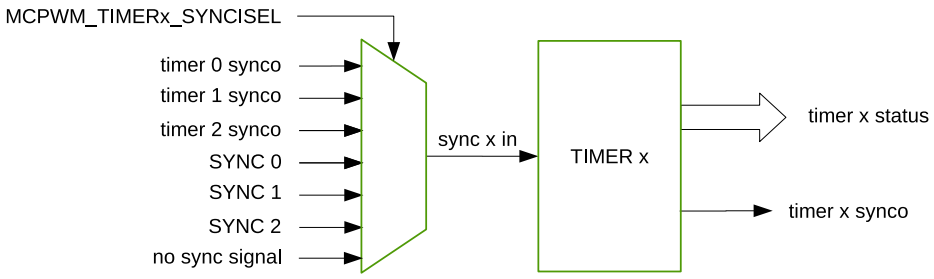


图 41.3-1. 定时器模块

该模块包含一个定时器，它可以按照指定的周期进行计数，它可以工作在递增计数模式，递减计数模式或递增递减循环计数模式下，它可以选择不同的同步输入源（共 7 个可选输入源）来进行计数值和方向的重载，从而使多个定时器彼此同步，它还可以输出一个同步输出源（共 4 个可选输出源）供其他定时器使用。

图中的 timer*x* status 指定时器模块输出的定时器状态，如计数模式，计数值等于 0 或周期等。

41.3.1.2 操作器模块

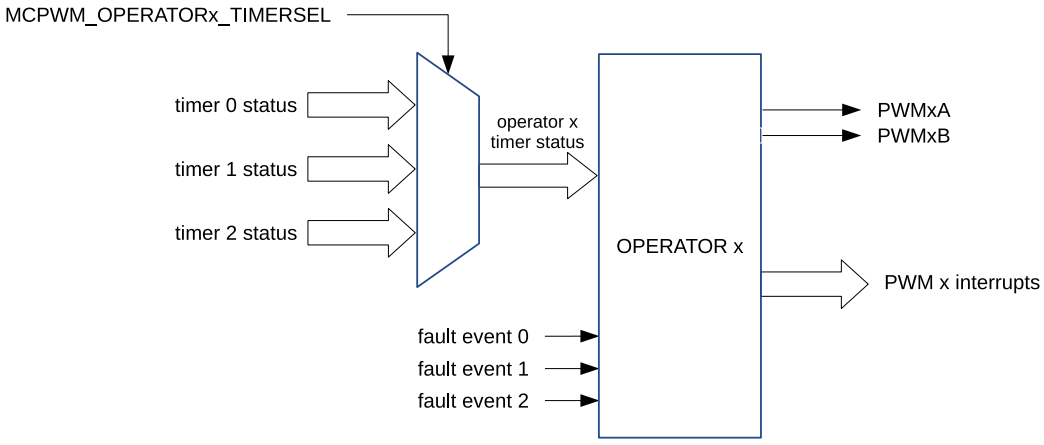


图 41.3-2. 操作器模块

一个 PWM 操作器包含一个 PWM 生成器、一个死区生成器、一个故障检测模块以及一个 PWM 载波模块，它们的功能如表 41.3-1 所示：

表 41.3-1. 操作器模块包含的各个子模块功能

子模块	功能
PWM 生成器	按照指定的占空比和波形，生成 PWM 信号
死区生成器	可以选择对 PWM 生成器产生的 PWM 信号的上升沿和下降沿添加死区（即对上升沿和下降沿添加延迟）以及进行反相等操作
PWM 载波	可以选择对死区生成器产生的 PWM 信号添加载波，载波的频率可以配置，并且添加载波后的 PWM 波形的第一个脉冲持续时间可以配置
故障检测	检测故障并生成对应的故障事件，按照指定的间隔模式（一次性模式或逐周期模式）响应故障事件以及发生故障事件时，按照指定的方式对 PWM 载波模块产生的 PWM 信号采取操作

41.3.1.3 捕获模块

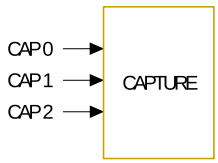


图 41.3-3. 捕获模块

该模块包含一个可以进行同步的定时器，当输入的捕获信号有效时，将捕获定时器的计数值。

41.3.1.4 ETM 模块

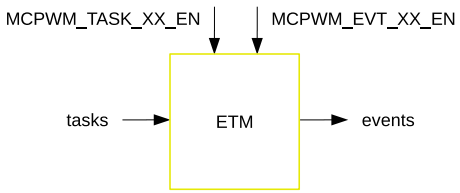


图 41.3-4. ETM 模块

该模块对接收到的任务进行响应，或产生事件并输出，可以独立配置每个事件和任务的使能。

41.3.2 PWM 定时器模块

MCPWM 外设有三个 PWM 定时器模块。它们中的任何一个都可以决定三个 PWM 操作器模块中任意一个的必要事件时序。通过使用 GPIO 交换矩阵的同步信号，内置同步逻辑允许一个或多个 MCPWM 外设中的多个 PWM 定时器模块作为一个系统协同工作。

41.3.2.1 PWM 定时器模块的配置

用户可配置 PWM 定时器模块的以下功能：

- 通过指定 PWM 定时器频率或周期来控制事件发生的频率

- 配置某个 PWM 定时器与其他 PWM 定时器或模块进行相位同步
- 配置定时器计数模式：递增，递减，或递增递减循环计数模式
- 使用预分频器更改 PWM 定时器时钟 (PT_CLK) 的速率。每个定时器都有自己的预分频器，通过寄存器 MCPWM_TIMER0_CFG0_REG 的 MCPWM_TIMER*n*_PRESCALE 配置。PWM 定时器根据该寄存器的设置以较慢的速度递增或递减。当定时器停止后并再次开始计数时，新的MCPWM_TIMER*n*_PRESCALE 配置值才会生效。

41.3.2.2 PWM 定时器工作模式和定时事件生成

PWM 定时器有三种工作模式，由 PWM*x* 定时器模式字段配置：

- 递增计数模式：
  定时器从零增加到周期字段中配置的值。达到周期值后，PWM 定时器清零，并再次开始递增。PWM 周期 = 周期寄存器中的周期值 + 1。
  说明：周期寄存器为 MCPWM_TIMER*n*_PERIOD (*x* = 0, 1, 2)，对应 MCPWM_TIMER0_PERIOD、MCPWM_TIMER1_PERIOD、MCPWM_TIMER2_PERIOD。
- 递减计数模式：
  PWM 定时器从周期寄存器中的值开始递减到零。达到零后，将恢复为周期值，并再次开始递减。在这种情况下，PWM 周期 = 周期寄存器中的周期值 + 1。
- 递增-递减循环模式：
  此模式结合了上述两种模式。PWM 定时器从零开始递增，直到达到周期值，再次递减为零。PWM 定时器按照此模式循环递增递减。PWM 周期 = 周期寄存器的周期值 × 2。

图 41.3-5 至 41.3-8 显示不同模式下的 PWM 定时器波形，包括同步事件期间的定时器行为。递增计数模式中，同步后永远保持递增计数；递减计数模式中，同步后永远保持递减计数；递增-递减循环模式中，同步后由
MCPWM_TIMER*n*_PHASE_DIRECTION 配置计数方向。

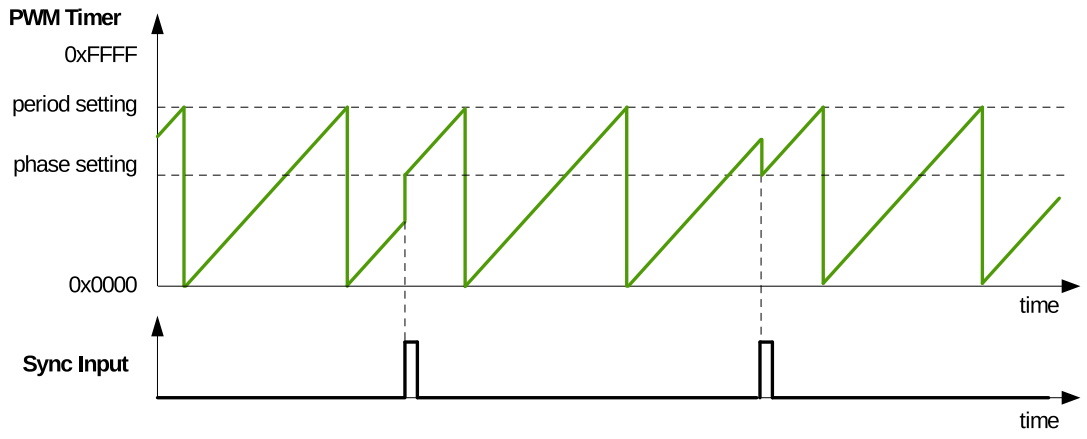


图 41.3-5. 递增计数模式波形

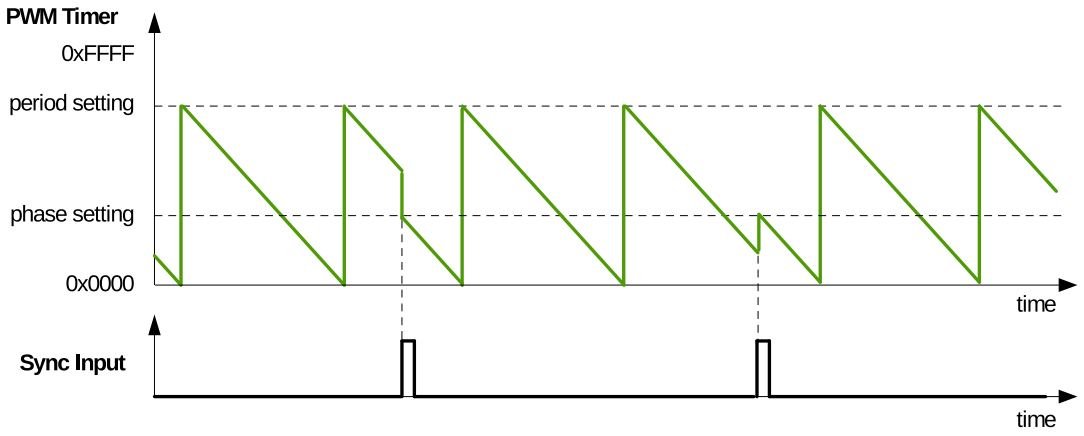


图 41.3-6. 递减计数模式波形

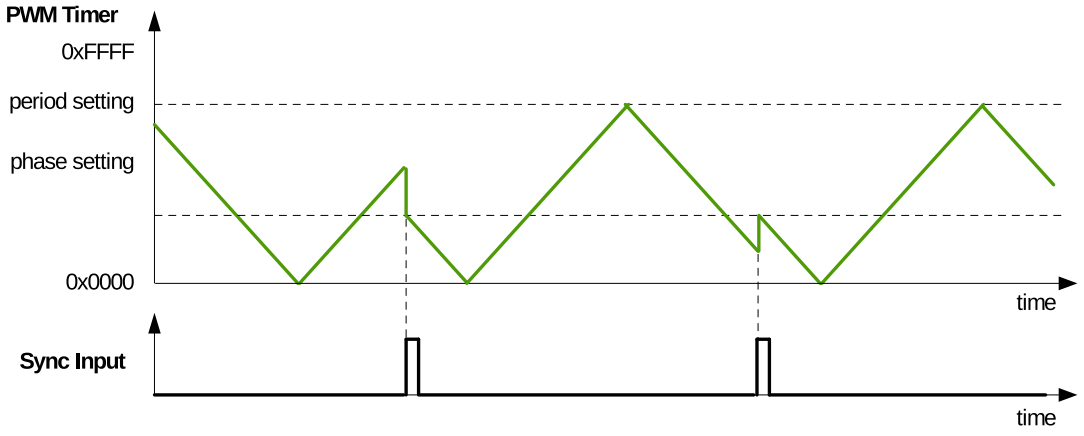


图 41.3-7. 递增递减循环模式波形，同步事件后递减

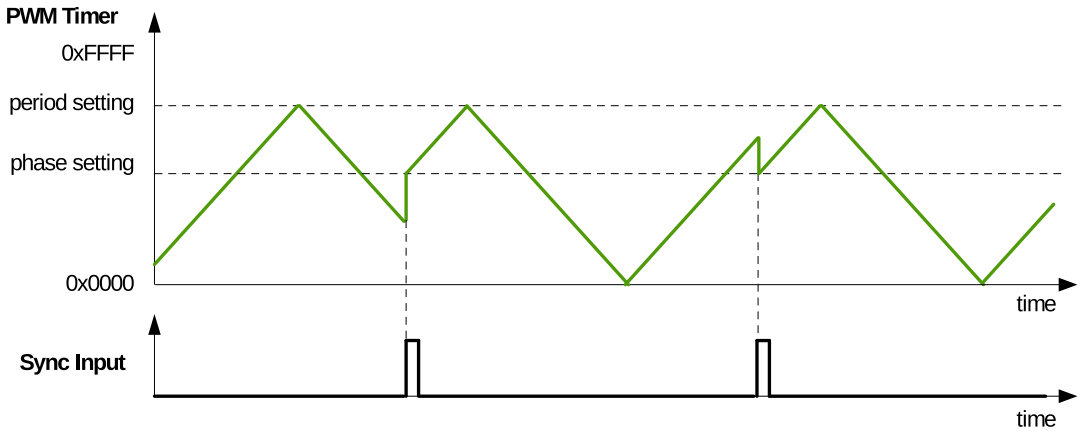


图 41.3-8. 递增递减循环模式波形，同步事件后递增

PWM 定时器运行时，定期自动生成以下定时事件：

- UTEP：当 PWM 定时器等于周期字段的值 (MCPWM_TIMER*n*_PERIOD) 且 PWM 定时器递增计数时，生成的定时事件。
- UTEZ：当 PWM 定时器等于零且 PWM 定时器递增计数时，生成的定时事件。
- DTEP：当 PWM 定时器等于周期字段的值 (MCPWM_TIMER*n*_PERIOD) 且 PWM 定时器递减时，生成的定时事件。
- DTEZ：当 PWM 定时器等于零且 PWM 定时器递减时，生成的定时事件。

图 41.3-9 至 41.3-11 为 U/DTEP 和 U/DTEZ 定时事件的时序波形。

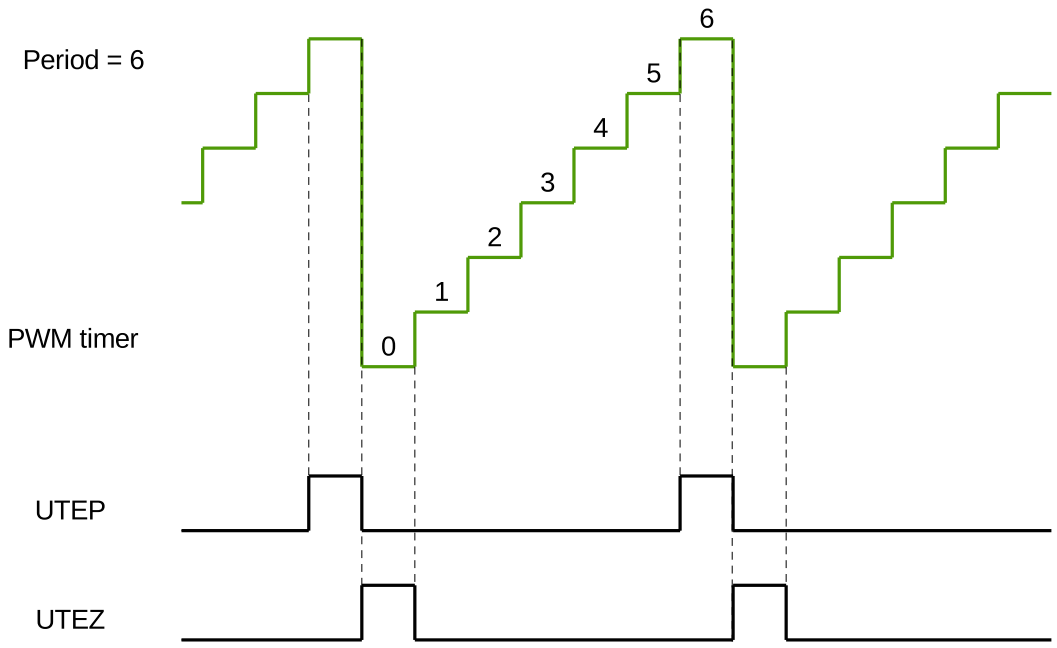


图 41.3-9. 递增模式中生成的 UTEP 和 UTEZ

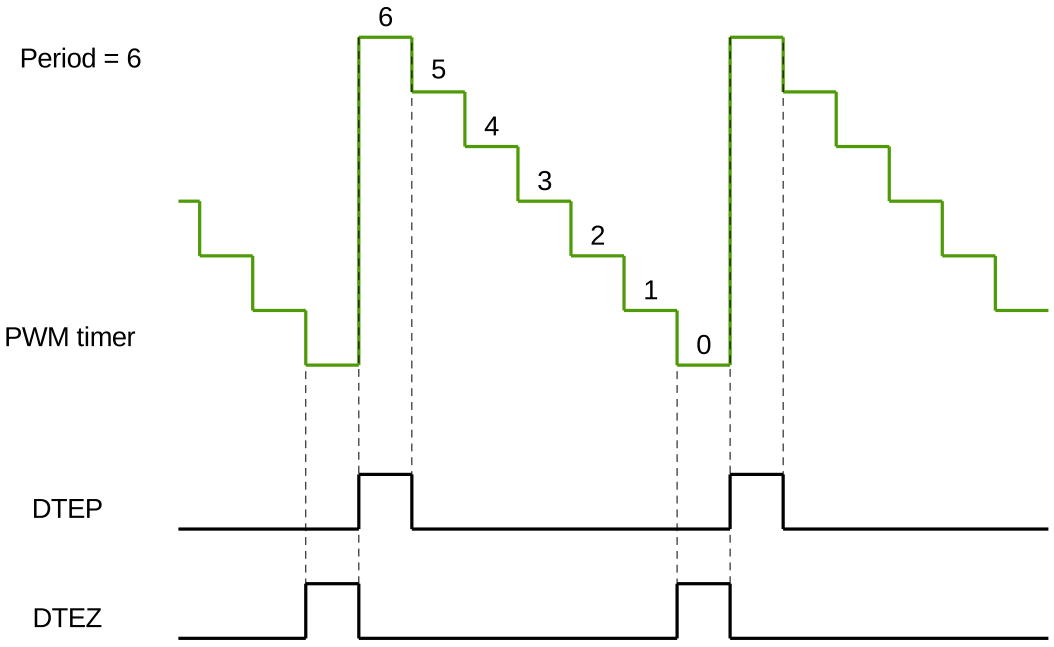


图 41.3-10. 递减模式中生成的 DTEP 和 DTEZ

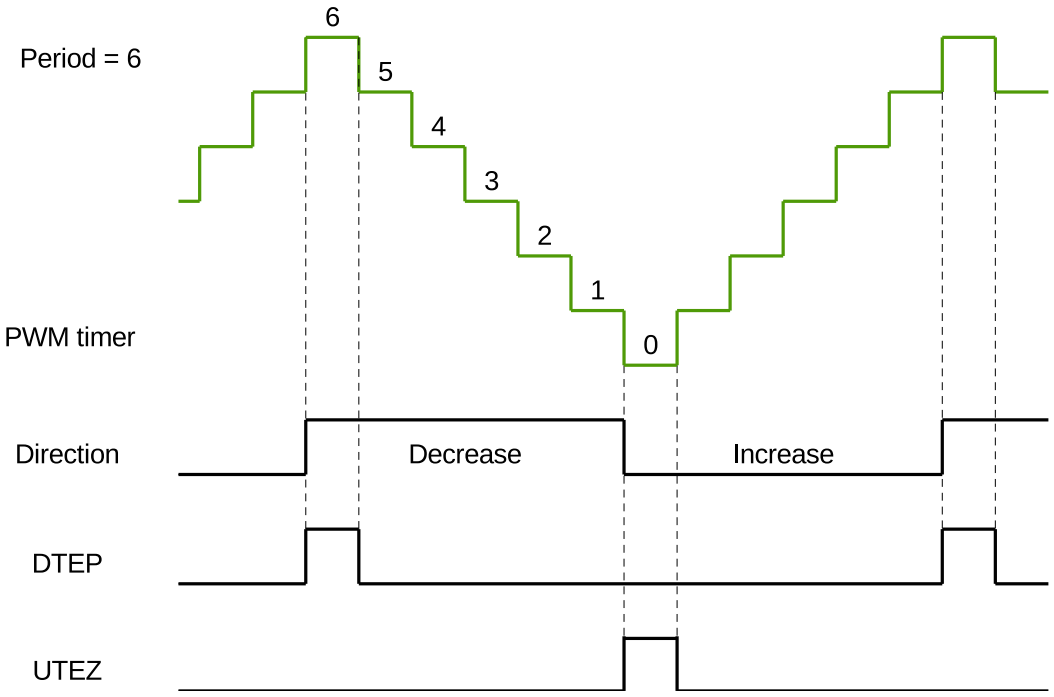


图 41.3-11. 递增递减模式中生成的 DTEP 和 UTEZ

需注意，在递增递减模式下，当定时器递增计数时，定时器范围为 [0，周期值 - 1]；递减计数时，定时器范围为 [周期值，1]。也就是说，在该模式下，将定时器同步为 0 时，则不支持递减计数，即 MCPWM_TIMER*n*_PHASE_DIRECTION 不能为 1；定时器同步为周期值时，也不支持递增计数，即 MCPWM_TIMER*n*_PHASE_DIRECTION 不能为 0。所以，当定时器同步为 0 时，计数方向只能为递增，即将 MCPWM_TIMER*n*_PHASE_DIRECTION 配置为 0；定时器同步为周期值时，计数方向只能为递减，即将 MCPWM_TIMER*n*_PHASE_DIRECTION 配置为 1。

41.3.2.3 PWM 定时器影子寄存器

PWM 定时器周期寄存器和 PWM 定时器时钟预分频器寄存器配有影子寄存器。影子寄存器能够备份即将写入有效寄存器的值，并在硬件同步的特定时刻写入有效寄存器。有效寄存器和影子寄存器的功能如下：

- 有效寄存器：直接控制硬件执行的所有操作。
- 影子寄存器：作为临时缓冲区，存储即将写入有效寄存器的值。在用户配置的某个时间点，影子寄存器中的值被写入有效寄存器。在此之前，影子寄存器的内容对受控硬件没有任何直接影响，防止寄存器由软件异步修改时可能发生的错误硬件操作。影子寄存器和有效寄存器具有相同的存储器地址。软件始终在影子寄存器中进行读写操作。

  当定时器开始工作时，时钟预分频器的有效寄存器更新。当 MCPWM_GLOBAL_UP_EN 置 1 时，可通过以下方式选择更新有效寄存器的时间点：

  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 0，立即更新。
  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 1，当 PWM 定时器值为 0 时，开始更新。
  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 2，当 PWM 定时器同步时，开始更新。
  - 将 MCPWM_TIMER*n*_PERIOD_UPMETHOD 置 3，当 PWM 定时器值为 0 时或同步时，开始更新。
  - 软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。

41.3.2.4 PWM 定时器同步和锁相

PWM 模块采用灵活的同步方法。每个 PWM 定时器都有一个同步输入和一个同步输出。同步输入可以从 GPIO 矩阵的三个同步输出和三个 PWM 定时器的三个同步输出信号中选择。同步输出可以在 PWM 定时器等于周期、PWM 定时器等于零、同步输入有效或软件同步时生成。因此，每个 PWM 定时器都可以灵活的选择同步源，从而实现多种相位同步方式。在同步时，PWM 定时器的预分频器会复位其计数器，并重新开始对时钟的分频，以保证定时器的正确性。

41.3.3 PWM 操作器模块

PWM 操作器模块具备以下功能：

- 根据相应 PWM 定时器的定时参考生成 PWM 信号对。
- PWM 信号对的每个信号都可以独立设置特定的死区（即对 PWM 信号的上升沿和下降沿添加延迟）。
- 可通过配置将载波叠加到 PWM 信号上。
- 故障条件下处理响应。

图 41.3-12 为 PWM 操作器的框图。

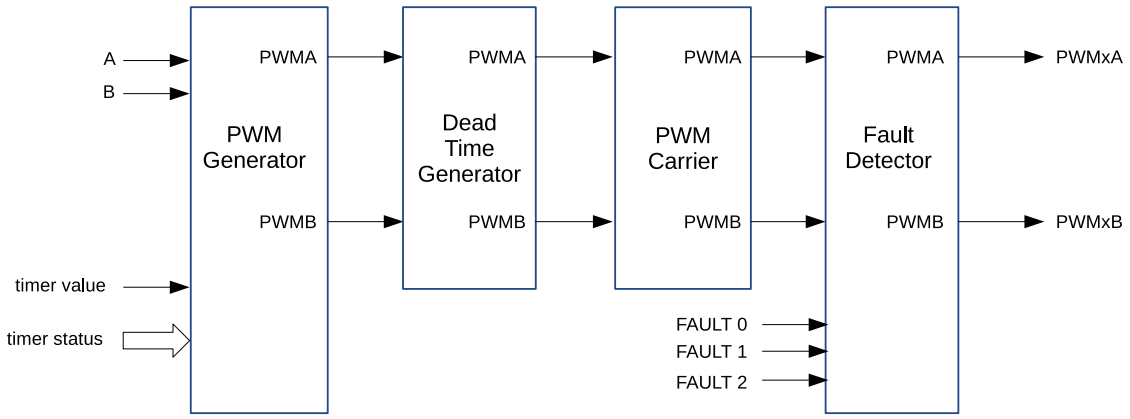


图 41.3-12. PWM 操作器的子模块

41.3.3.1 PWM 生成器模块

PWM 生成器模块的作用

此模块可以生成或导入重要的定时事件，转化为特定操作后，在 PWM*x*A 和 PWM*x*B 输出处生成所需的波形。

PWM 生成器模块执行以下操作：

- 基于使用寄存器 A 和 B 配置的时间戳生成定时事件。满足以下条件时发生定时事件（寄存器 A 和 B 的配置方法在章节 41.3.3.1 描述）：
  - UTEA：PWM 定时器递增计数并且其值等于寄存器 A。
  - UTEB：PWM 定时器递增计数并且其值等于寄存器 B。
  - DTEA：PWM 定时器递减计数并且其值等于寄存器 A。
  - DTEB：PWM 定时器递减计数并且其值等于寄存器 B。
- 基于故障或同步事件生成 U/DTO、U/DT1 定时事件。
  - UTO：定时器递增计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_TO_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_TO_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_TO_SEL 置 2）或同步（字段 MCPWM_GEN*n*_TO_SEL 置 3）。
  - UT1：定时器递增计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_T1_SEL 置 2）或同步（字段 MCPWM_GEN*n*_T1_SEL 置 3）。
  - DTO：定时器递减计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_TO_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_TO_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_TO_SEL 置 2）或同步（字段 MCPWM_GEN*n*_TO_SEL 置 3）。
  - DT1：定时器递减计数并且检测到 FAULTO 事件（字段 MCPWM_GEN*n*_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GEN*n*_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GEN*n*_T1_SEL 置 2）或同步（字段 MCPWM_GEN*n*_T1_SEL 置 3）。
- 当这些定时事件同时发生时管理优先级。
- 基于定时事件产生置 1、置 0 和取反操作。

- 根据 PWM 生成器模块的配置来控制 PWM 占空比。
- 使用影子寄存器处理新的时间戳值，以防止 PWM 波形中的毛刺干扰。

PWM 生成器影子寄存器

硬件使用的时间戳寄存器 A 和 B 都配有影子寄存器，为 MCPWM_GEN*n*_A_REG 和 MCPWM_GEN*n*_B_REG。影子寄存器提供了一种与硬件同步更新寄存器的方法。

当 MCPWM_GLOBAL_UP_EN 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。MCPWM_GEN*n*_A_REG 和 MCPWM_GEN*n*_B_REG 的更新方式字段为 MCPWM_GEN*n*_CFG_UPMETHOD。软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。更多关于影子寄存器的描述，请参考章节 41.3.2.3。

定时事件

表 41.3-2 概括了所有定时信号和事件。

表 41.3-2. PWM 生成器中的所有定时事件

信号	事件描述	PWM 定时器操作
DTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递减计数
DTEZ	PWM 定时器的值等于 0	
DTEA	PWM 定时器的值等于寄存器 A	
DTEB	PWM 定时器的值等于寄存器 B	
DTO 事件	基于故障或同步事件	
DT1 事件	基于故障或同步事件	
UTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递增计数
UTEZ	定时器的值等于 0	
UTEA	PWM 定时器的值等于寄存器 A	
UTEB	PWM 定时器的值等于寄存器 B	
UTO 事件	基于故障或同步事件	
UT1 事件	基于故障或同步事件	
软件强制事件	软件触发的异步事件	-

软件强制事件用于在 PWM*x*A 和 PWM*x*B 的输出上施加非连续或连续的强制电平。此更改是异步完成的。软件强制事件由寄存器 MCPWM_GEN*n*_FORCE_REG 控制。

PWM 生成器模块中 TO/T1 的选择和配置独立于故障处理模块中的故障事件的配置，某个特定的跳闸事件可以不被配置为在故障处理器模块中引起跳闸动作，而是用于由 PWM 生成器用于触发 TO/T1 以控制 PWM 波形。

需要注意的是，当 PWM 定时器处于递增递减循环计数模式时，它将在 TEP 事件后递减，在 TEZ 事件后递增。因此，在此模式下，将出现 DTEP 和 UTEZ，但不会出现 UTEP 和 DTEZ。

PWM 生成器可以同时处理多个事件。事件优先级由硬件决定，详见表 41.3-3 和表 41.3-4。优先级从 1（最高）到 7（最低）排列。需要注意的是，TEP 和 TEZ 事件的优先级取决于 PWM 定时器的计数模式。

如果 A 或 B 的值设置为大于周期，则 U/DTEA 和 U/DTEB 将永远不会发生。

表 41.3-3. PWM 定时器递增计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	UTEP
3	UTO
4	UT1
5	UTEB
6	UTEA
7（最低）	UTEZ

表 41.3-4. PWM 定时器递减计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	DTEZ
3	DTO
4	DT1
5	DTEB
6	DTEA
7（最低）	DTEP

说明：

1. UTEP 和 UTEZ 不同时发生。当 PWM 定时器处于递增计数模式，UTEP 将始终比 UTEZ 提前一个周期发生，如图 41.3-9 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，UTEP 不会发生。
2. DTEP 和 DTEZ 不同时发生。当 PWM 定时器处于递减计数模式时，DTEZ 始终比 DTEP 早一个周期发生，如图 41.3-10 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，DTEZ 不会发生。

PWM 信号生成

当某个定时事件发生时，PWM 生成器控制输出 PWM*x*A 和 PWM*x*B 的电平。定时事件通过 PWM 定时器计数模式（递增或递减）进一步限定。根据定时器计数方向，模块可以对 PWM 定时器递增或递减计数的阶段执行不同的操作。

可以在 PWM*x*A 和 PWM*x*B 输出上配置以下操作：

- 置为高电平：将 PWM*x*A 或 PWM*x*B 的输出配置为高电平。
- 置为低电平：通过将 PWM*x*A 或 PWM*x*B 的输出配置为低电平，来清除 PWM*x*A 或 PWM*x*B 的输出。
- 取反：将 PWM*x*A 或 PWM*x*B 的当前输出电平配置为相反的值。如果它当前被拉高，则拉低，或反之。
- 不进行操作：保持 PWM*x*A 和 PWM*x*B 输出电平不变。在这种状态下，仍然可以触发中断。

输出上的操作通过寄存器 MCPWM_GEN*n*_A_REG 和 MCPWM_GEN*n*_B_REG 配置。每一次输出的操作都独立配置。此外，可以根据事件，灵活地在输出上执行不同操作。表 41.3-2 中列举的任何事件都可以作用于 PWM*x*A 或 PWM*x*B 输出上。关于生成器 0，1 或 2 的寄存器信息，请参考章节 41.5。

常见配置的波形

在下面各种波形的配置中，period 指 PWM 操作器所选 PWM 定时器的周期寄存器的配置值（即 MCPWM_TIMER*n*_PERIOD (*x* = 0, 1, 2)），配置方法参考章节 41.3.2。A 和 B 指硬件使用的时间戳寄存器 A 和 B，配置方法参考章节 41.3.3.1。UTEA/B、UTEZ/P、DTEA/B、DTEZ/P、DTO/1 和 UTO/1 指 PWM 生成器的各种定时事件，其定义参考章节 41.3.3.1。向上的箭头、向下的箭头、T 标识分别代表定时事件发生时将 PWM 波形置为高电平、置为低电平、取反，配置方法参考章节 41.3.3.1。PWM*x*A 和 PWM*x*B 是 PWM 生成器 *x* 的输出信号。

图 41.3-13 为 PWM 定时器在递增递减循环计数时生成的对称 PWM 波形。该模式下的直流 0%-100% 调制可由以下公式获得：

$$Duty = (Period - A) \div Period$$

如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递增，则 PWM 输出被拉高。如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递减，则 PWM 输出被拉低。

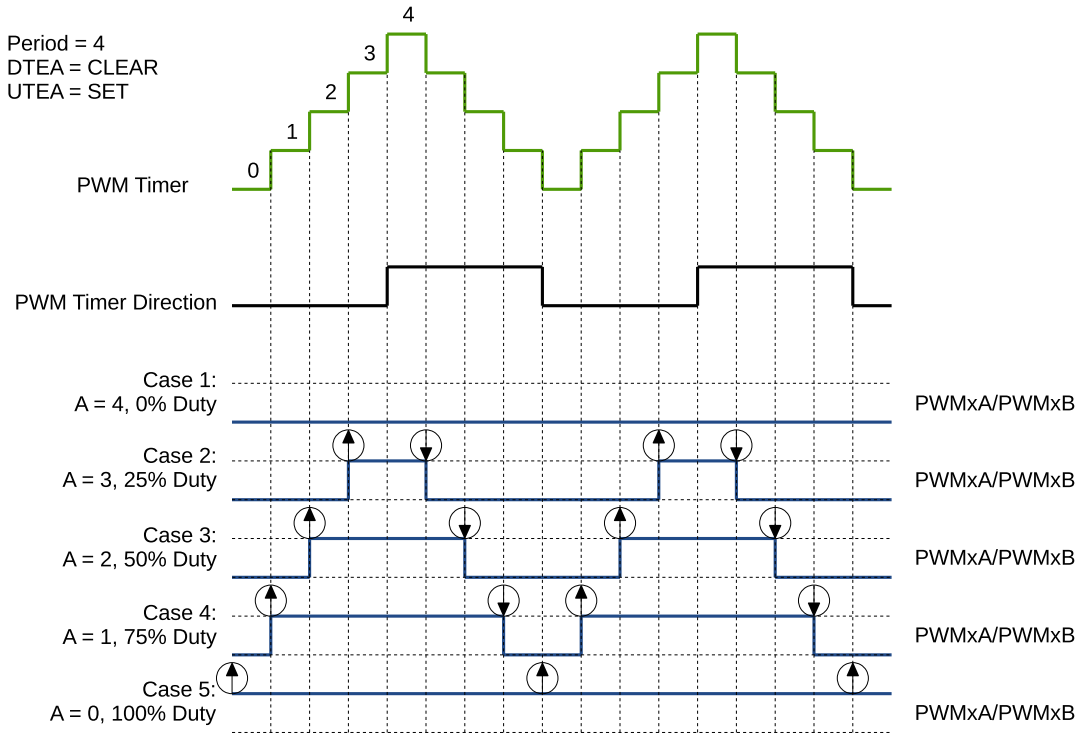


图 41.3-13. 递增递减模式下的对称波形

图 41.3-14 至图 41.3-17 的 PWM 波形描述了常见的 PWM 生成器配置。

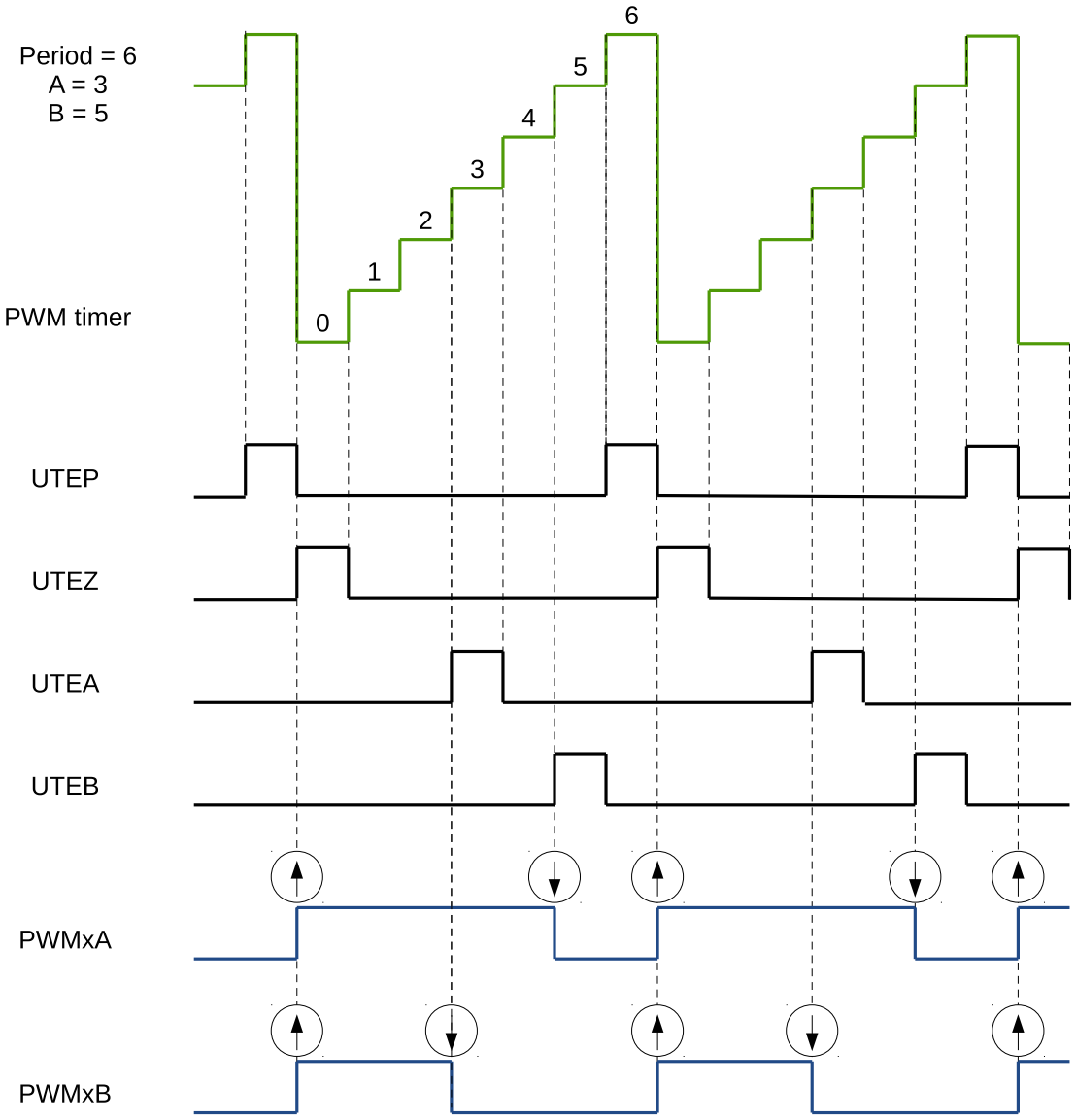


图 41.3-14. 递增计数模式，单边不对称波形，PWM*x*A 和 PWM*x*B 独立调制–高电平

PWM*x*A 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWM*x*B 的占空比调制由 A 设置，高电平有效，与 A 成正比。

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

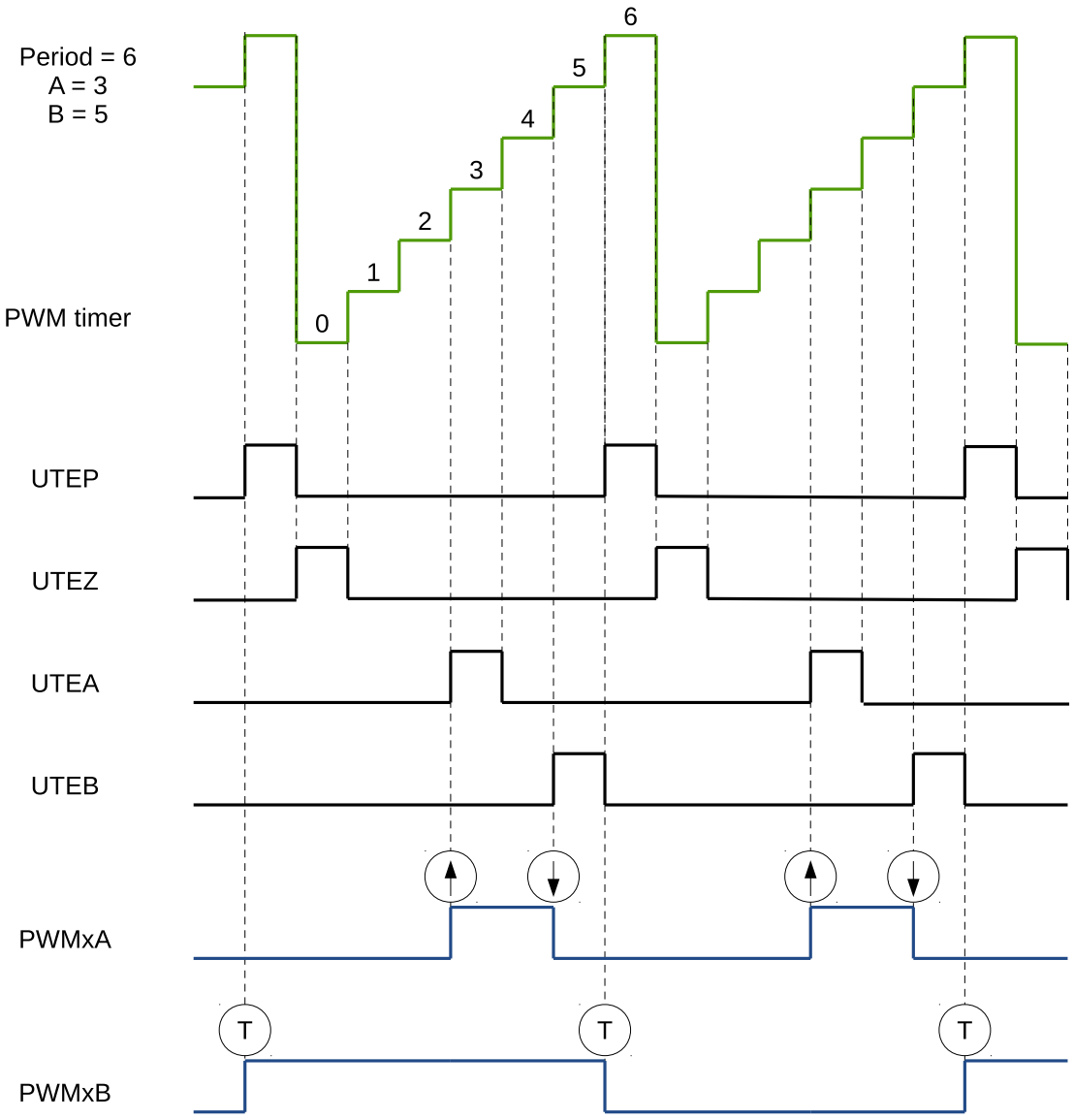


图 41.3-15. 递增计数模式，脉冲位置不对称波形，PWM*x*A 独立调制

脉冲可以在 PWM 波形内（零至周期值之间）的任何地方生成。
PWM*x*A 占空比与 (B – A) 成正比。

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

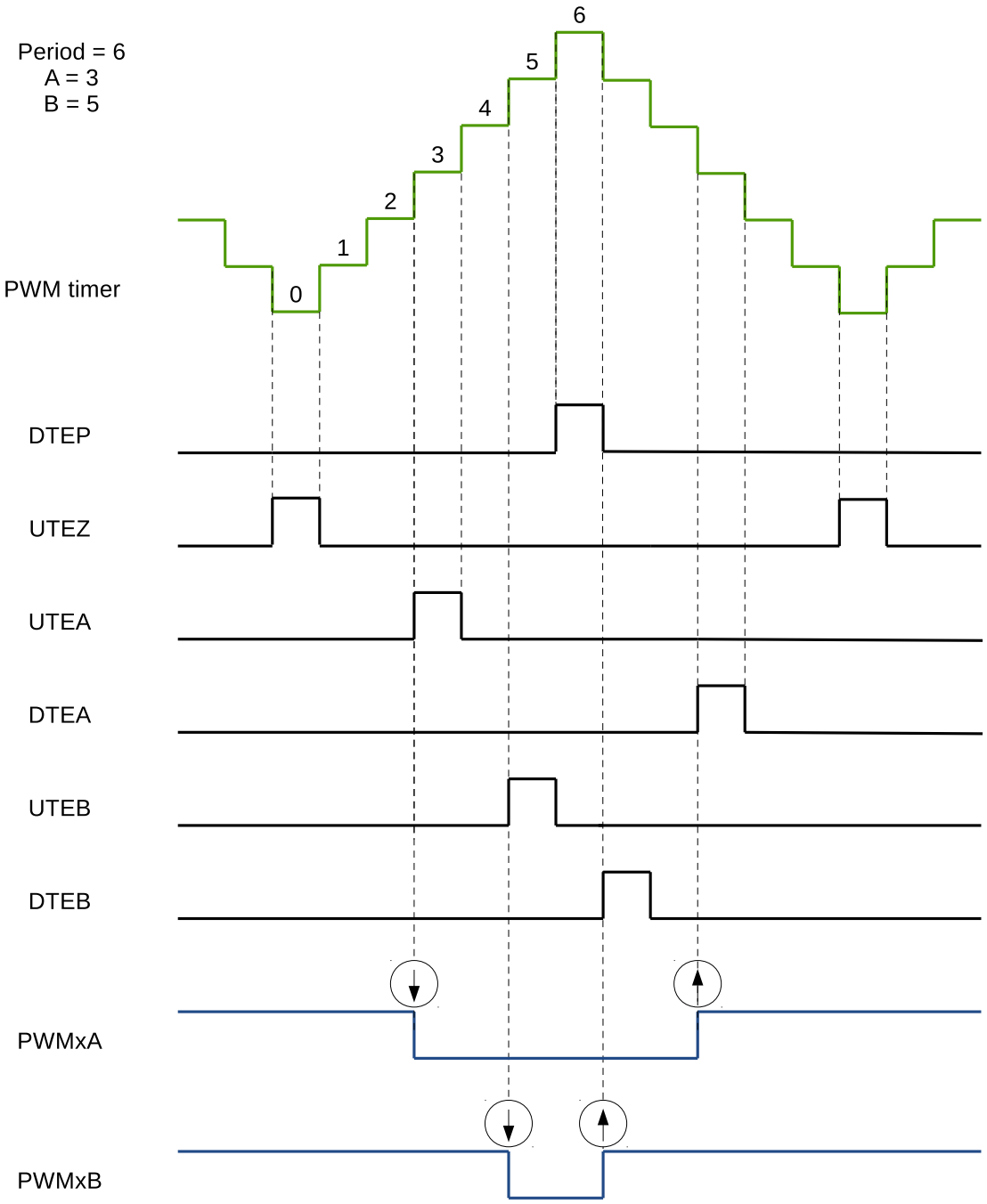


图 41.3-16. 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–高电平有效

PWM*x*A 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWM*x*B 的占空比调制由 B 设置，高电平有效，与 B 成正比。
输出 PWM*x*A 和 PWM*x*B 可驱动不同开关。

$$Period = (2 \times MCPWM\_TIMERn\_PERIOD) \times T_{PT\_clk}$$

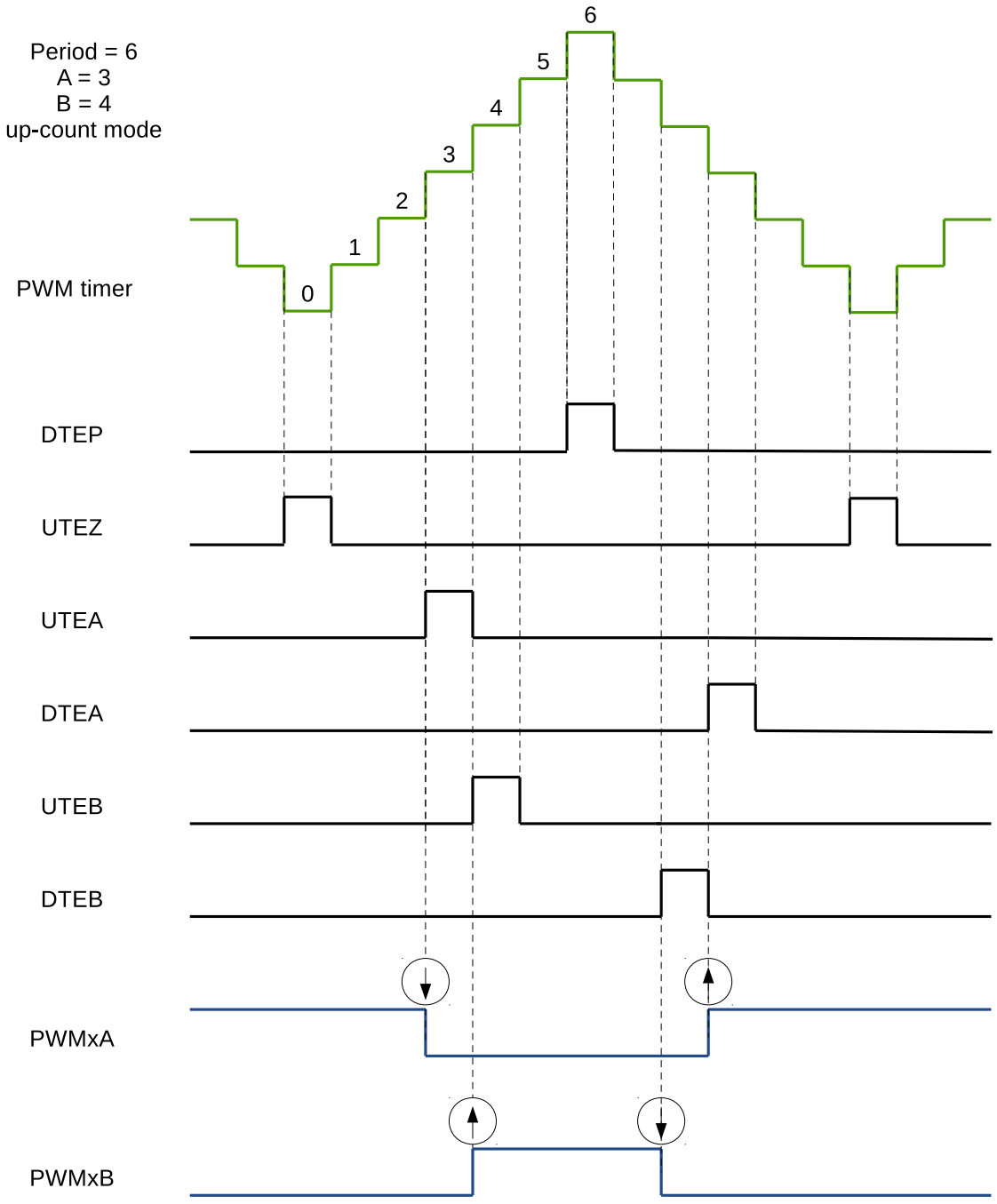


图 41.3-17. 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–互补

PWM*x*A 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWM*x*B 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWM*x*A/B 输出可驱动上/下（互补）开关。
死区 = B – A，边沿位置完全可由软件配置。必要时，可使用死区生成器模块设置其他边沿延迟方式。

$$Period = (2 \times MCPWM\_TIMERn\_PERIOD) \times T_{PT\_clk}$$

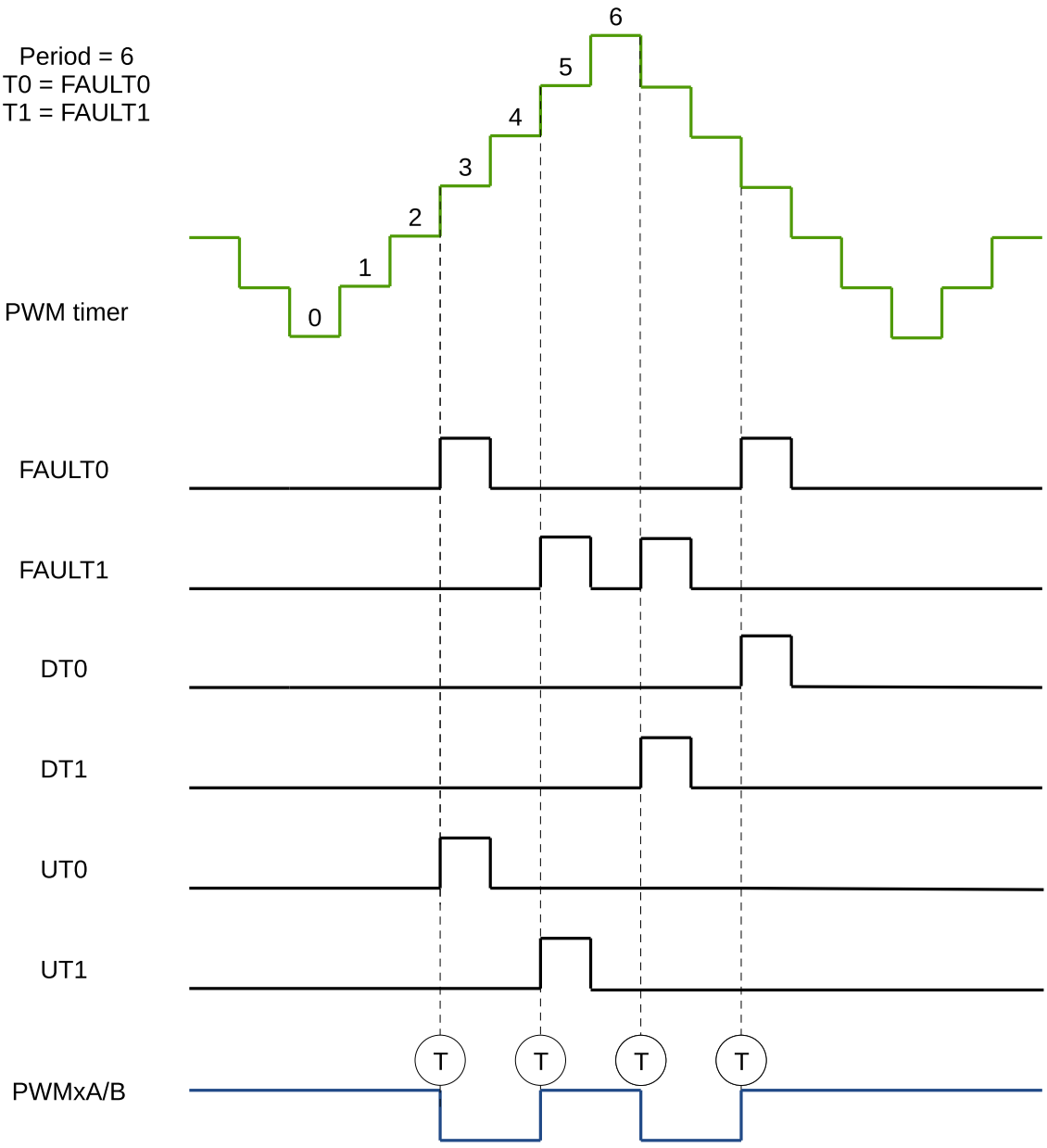


图 41.3-18. 递增递减循环计数模式，错误或同步事件，在 PWM*x*A 和 PWM*x*B 上相同调制

图 41.3-18 为当 UTO/1 和 DTO/1 事件产生时的波形。在本例中，TO 选择 FAULTO ，T1 选择 FAULT1。TO 和 T1 所选择的事件可以独立配置，这些事件可以是 FAULTO、FAULT1、FAULT2 或同步事件。具体配置方法见章节 41.3.3.1。

软件强制事件

在 PWM 生成器内有 2 种软件强制事件：

- 非连续即时 (NCI) 软件强制事件：当由软件触发时，这类事件在 PWM 输出上立即生效。强制是不连续的，意味着下一个激活的定时事件能够改变 PWM 输出。
- 连续 (CNTU) 软件强制事件：这类事件是连续的。强制 PWM 将持续输出，直到通过软件释放。事件触发器可配置。这类事件可配置为定时或即时发生。

对于 NCI 软件强制事件，向寄存器 MCPWM_GEN*n*_A_NCIFORCE 或 MCPWM_GEN*n*_B_NCIFORCE 写 1 再写

0，触发 PWM*x*A 或 PWM*x*B 的 NCI 软件强制事件。配置寄存器 MCPWM_GEN*n*_A_NCIFORCE_MODE 或 MCPWM_GEN*n*_B_NCIFORCE_MODE 指定当 NCI 软件强制事件发生时，对 PWM*x*A 或 PWM*x*B 波形的操作，配置为 0 或 3 表示无操作，配置为 1 表示置为低电平，配置为 2 表示置为高电平。

对于 CNTU 软件强制事件，配置寄存器 MCPWM_GEN*n*_CNTUFORCE_UPMETHOD，指定 CNTU 软件强制事件的触发方式，所有 bit 配置为 0，表示立即触发 CNTU 软件强制事件，bit 0 配置为 1，表示 TEZ 事件触发 CNTU 软件强制事件，bit 1 配置为 1，表示 TEP 事件触发 CNTU 软件强制事件，bit 2 配置为 1，表示 TEA 事件触发 CNTU 软件强制事件，bit 3 配置为 1，表示 TEB 事件触发 CNTU 软件强制事件，bit 4 配置为 1，表示 PWM 生成器所选的 PWM 定时器同步时触发 CNTU 软件强制事件，bit 5 配置为 1，表示不触发 CNTU 软件强制事件。配置寄存器 MCPWM_GEN*n*_A_CNTUFORCE_MODE 或 MCPWM_GEN*n*_B_CNTUFORCE_MODE 指定当 CNTU 软件强制事件发生时，对 PWM*x*A 或 PWM*x*B 波形的操作，配置为 0 或 3 表示无操作，配置为 1 表示置为低电平，配置为 2 表示置为高电平。

图 41.3-19 为 NCI 软件强制事件的一种波形。NCI 用于单独强制 PWM*x*A 输出为低电平，PWM*x*B 不受强制事件控制。

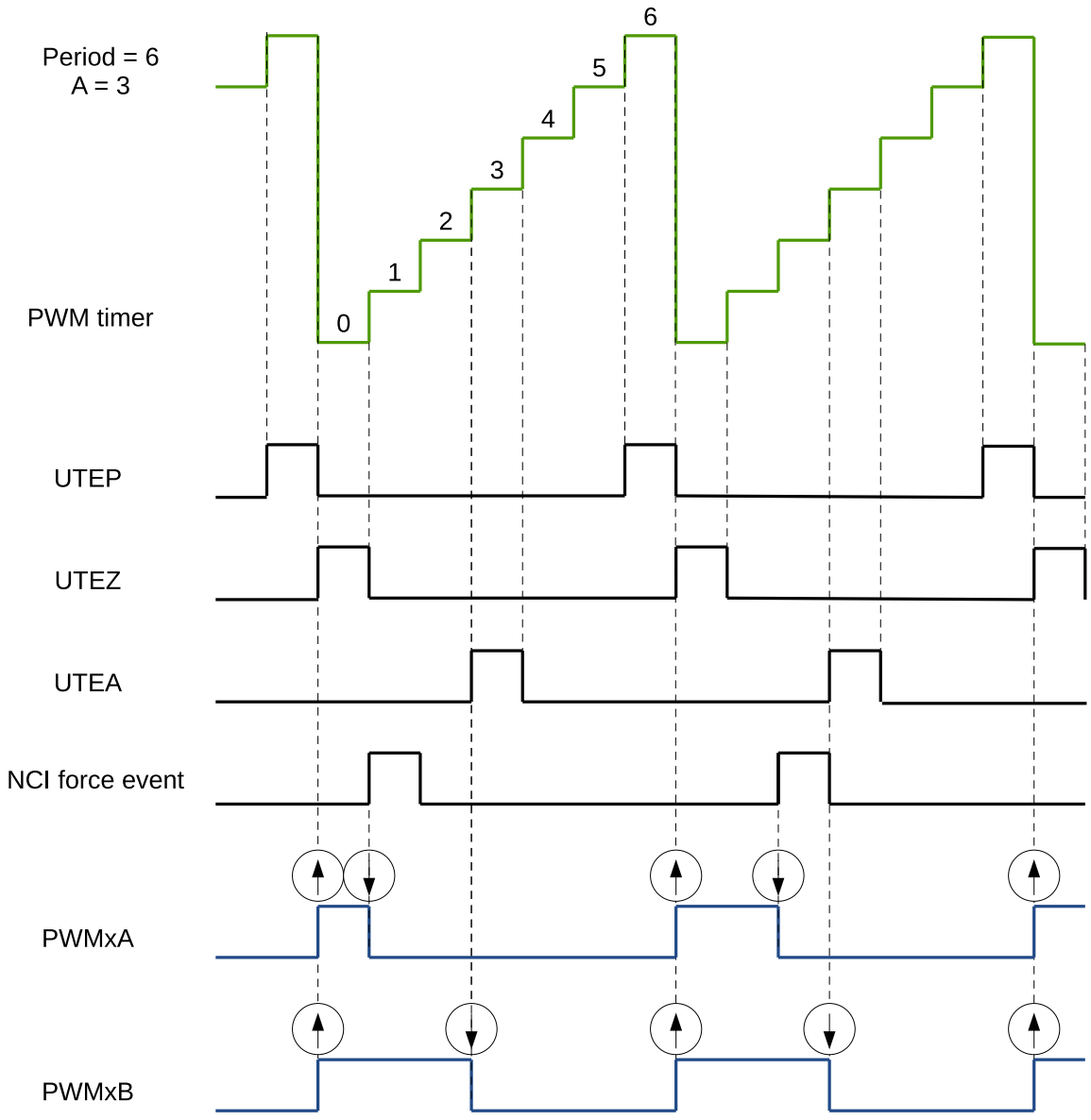


图 41.3-19. NCI 在 PWM*x*A 输出上软件强制事件示例

图 41.3-20 为 CNTU 软件强制事件的波形。TEZ 事件被选为 CNTU 软件强制事件的触发器。CNTU 用于单独强制 PWM*x*B 输出为低电平，但 PWM*x*A 不受强制。

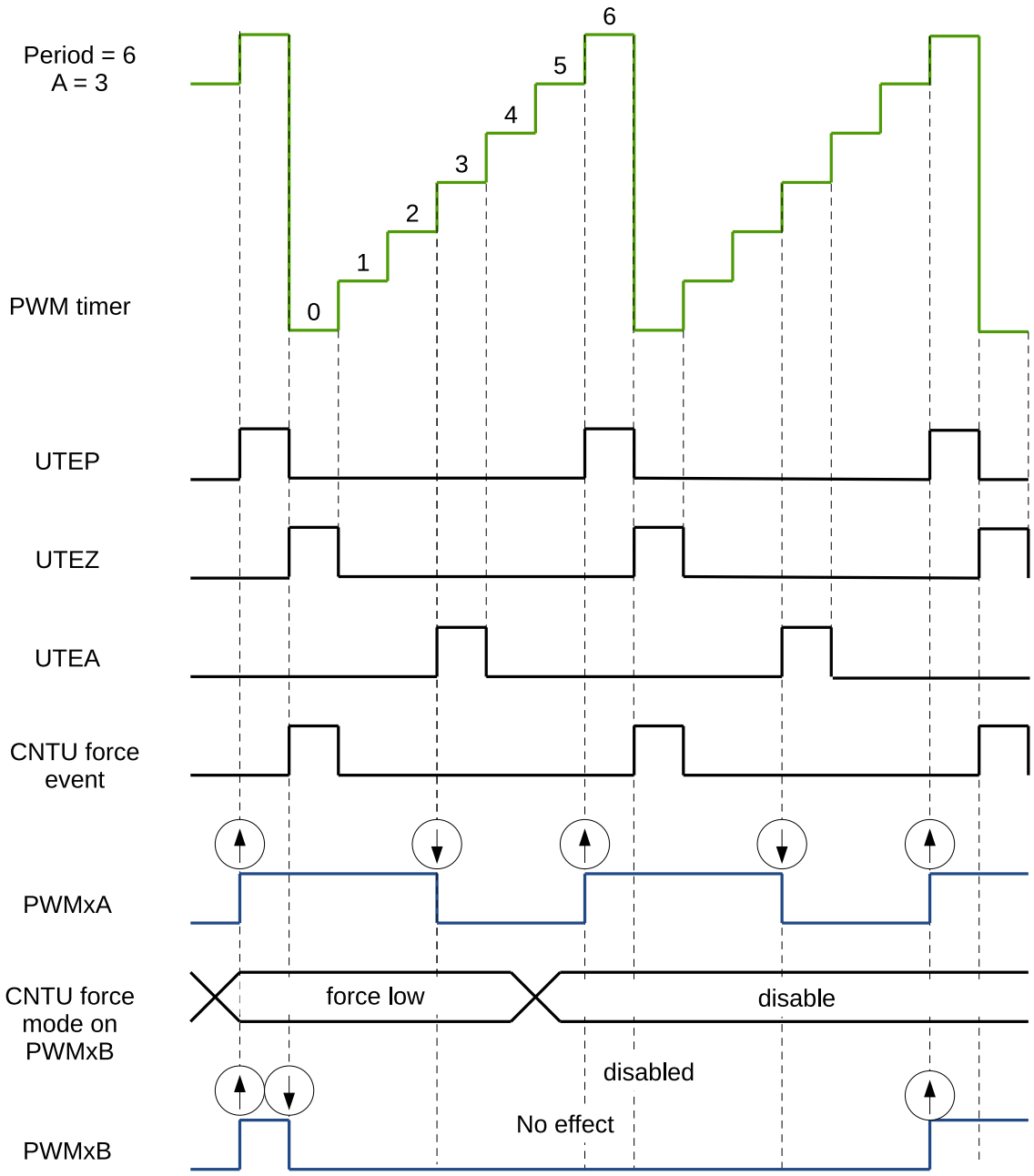


图 41.3-20. CNTU 在 PWM*x*B 输出上软件强制事件示例

41.3.3.2 死区生成器模块

死区生成器模块作用

章节 41.3.3.1 介绍了在 PWM*x*A 和 PWM*x*B 输出上生成信号的几种方式/事件，包括信号边沿的特定位置。通过改变信号之间的边沿位置以及设置信号的占空比，可获得所需的死区。此外，也可以使用专门的死区生成器模块控制死区。

死区生成器模块的主要功能如下：

- 可以将输入的 PWM*x*A 或 PWM*x*B 信号作为公共源来生成具有死区的 PWM*x*A 和 PWM*x*B 信号对输出
- 通过在信号边沿增加延迟来生成死区：
  - 上升沿延迟 (RED)
  - 下降沿延迟 (FED)
- 可以生成多种 PWM 波形，经典的死区配置为：
  - 高电平有效互补 (AHC)
  - 低电平有效互补 (ALC)
  - 高电平有效 (AH)
  - 低电平有效 (AL)
- 如果死区已经在生成器中配置，死区发生器也可以不进行配置。

死区模块生成器影子寄存器

延迟寄存器 RED 和 FED 的影子寄存器为 MCPWM_DT*n*_RED_CFG_REG 和 MCPWM_DT*n*_FED_CFG_REG。
MCPWM_GLOBAL_UP_EN 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。
MCPWM_DT*n*_RED_CFG_REG 的更新方式寄存器为 MCPWM_DB*n*_RED_UPMETHOD；
MCPWM_DT*n*_FED_CFG_REG 的更新方式寄存器为 MCPWM_DB*n*_FED_UPMETHOD。软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。寄存器描述详见章节 41.3.2.3。

死区生成器模块的操作要点

图 41.3-21 描述了创建死区模块的开关拓扑。

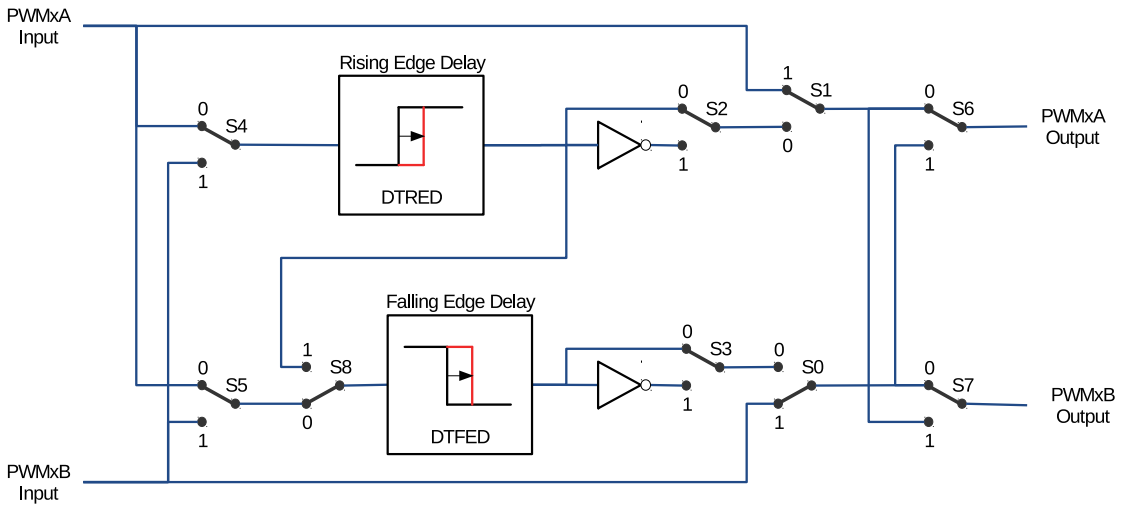


图 41.3-21. 死区模块的开关拓扑

上图中的 S0 - S8 开关由 MCPWM_DT*n*_CFG_REG 寄存器中的字段控制，详细信息见表 41.3-5。

表 41.3-5. 控制死区时间生成器开关的字段

开关	字段
S0	MCPWM_DB*n*_B_OUTBYPASS
S1	MCPWM_DB*n*_A_OUTBYPASS
S2	MCPWM_DB*n*_RED_OUTINVERT
S3	MCPWM_DB*n*_FED_OUTINVERT
S4	MCPWM_DB*n*_RED_INSEL
S5	MCPWM_DB*n*_FED_INSEL
S6	MCPWM_DB*n*_A_OUTSWAP
S7	MCPWM_DB*n*_B_OUTSWAP
S8	MCPWM_DB*n*_DEB_MODE

所有开关组合皆受支持，但不是所有的开关模式都是典型的使用模式。表 41.3-6 列举了一些典型的死区配置。在这些配置中，S4 和 S5 的开关位置将 PWM*x*A 设置为下降沿和上升沿延迟的公共源。表 41.3-6 中的模式可分为以下几类：

- **模式 1：绕过下降沿 (FED) 和上升沿 (RED) 的延迟**
  该模式下，死区模块被关闭。PWM*x*A 和 PWM*x*B 信号的波形无变化。
- **模式 2-5：经典死区极性设置**
  这些模式为典型极性配置，涵盖工业电源栅极驱动器中的高 / 低电平有效模式。图 41.3-22 至 41.3-25为典型波形。
- **模式 6 和 7：绕过下降沿 (FED) 或上升沿 (RED) 的延迟**
  这两种模式下，绕过上升沿延迟 (RED) 或下降沿延迟 (FED) 。因此，无需使用对应延迟。

表 41.3-6. 死区生成器的典型操作模式

模式	描述	S0	S1	S2	S3
1	PWM*x*A 和 PWM*x*B 波形无变化	1	1	X	X
2	高电平有效互补 (AHC)，参见图 41.3-22	0	0	0	1
3	低电平有效互补 (ALC)，参见图 41.3-23	0	0	1	0
4	高电平有效 (AH)，参见图 41.3-24	0	0	0	0
5	低电平有效 (AL)，参见图 41.3-25	0	0	1	1
6	PWM*x*A 输出 = PWM*x*A 输入（无延迟） PWM*x*B 输出 = PWM*x*A 输入，下降沿延迟	0	1	0 或 1	0 或 1
7	PWM*x*A 输出 = PWM*x*A 输入，上升沿延迟 PWM*x*B 输出 = PWM*x*B 输入（无延迟）	1	0	0 或 1	0 或 1

说明：

以上所有模式中，S4 - S8 的开关位置都置 0。

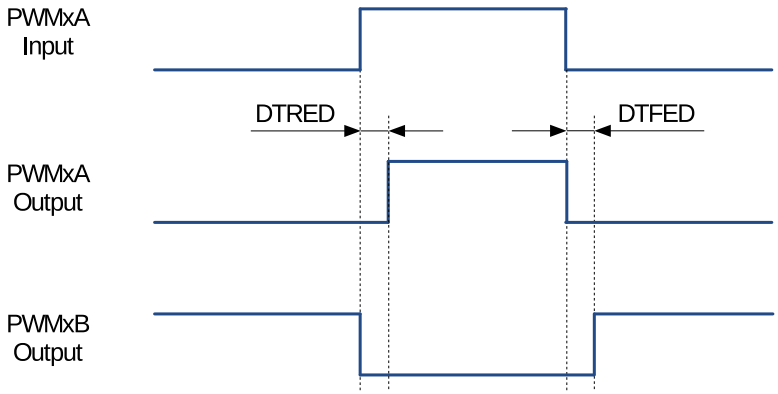


图 41.3-22. 高电平有效互补 (AHC) 死区波形

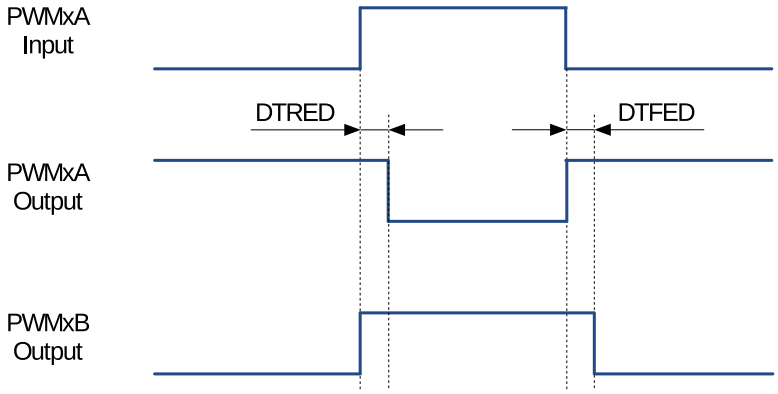


图 41.3-23. 低电平有效互补 (ALC) 死区波形

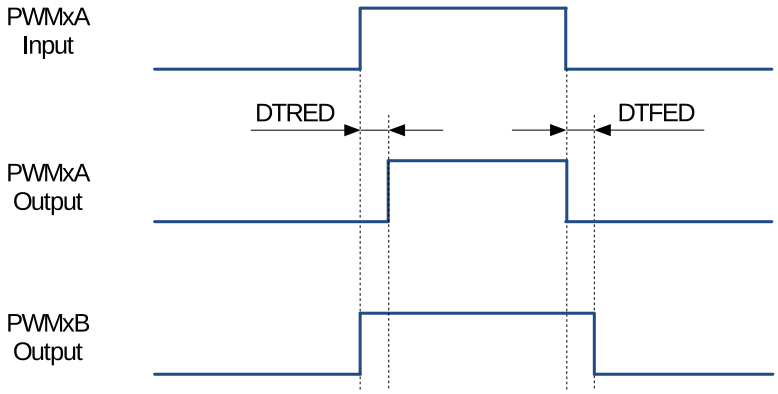


图 41.3-24. 高电平有效 (AH) 死区波形

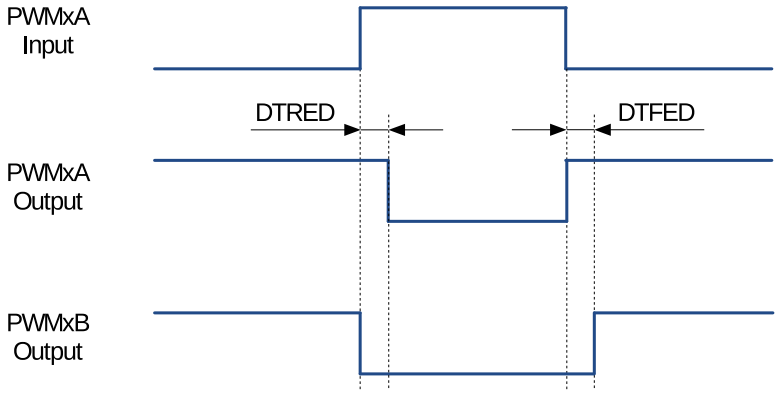


图 41.3-25. 低电平有效 (AL) 死区波形

上升沿延迟 (RED) 和下降沿延迟 (FED) 可分别设置。延迟的值通过 16 位寄存器 MCPWM_DT*x*_*RED* 和 MCPWM_DT*x*_*FED* 配置。寄存器值表示一个信号边沿可以延迟的 DT_clk 时钟周期值。DT_clk 可通过寄存器 MCPWM_DB*n*_CLK_SEL 从 PWM_clk 或 PT_clk 中选择。

通过以下公式计算下降沿延迟 (FED) 和上升沿延迟 (RED) 的值:

$$FED = MCPWM\_DTn\_FED \times T_{DT\_clk}$$

$$RED = MCPWM\_DTn\_RED \times T_{DT\_clk}$$

41.3.3.3 PWM 载波模块

将 PWM 输出耦合到电机驱动器可能需要使用变压器隔离。变压器只提供交流信号，而 PWM 信号的占空比可能在 0% 到 100% 之间变化。PWM 载波模块可以通过使用高频载波对其进行调制，将该信号传递给变压器。

功能概述

此模块的以下关键功能可配置:

- 载波频率
- 第一个脉冲的脉宽
- 第二个以及之后的脉冲的占空比
- 开启/关闭载波

操作要点

PWM 载波时钟 (PC_clk) 来自于 PWM_CLK。通过寄存器 MCPWM_CARRIER*n*_CFG_REG 的 MCPWM_CHOPPER*n*_PRESCALE 和 MCPWM_CHOPPER*n*_DUTY 位配置频率和占空比。一次性脉冲的功能在于提供高能量脉冲以接通电源开关。随后的脉冲用于保持上电的状态。一次性脉冲宽度可通过 MCPWM_CHOPPER*n*_OSHTWTH 位进行配置。通过 MCPWM_CHOPPER*n*_EN 位来使能/禁用载波模块。

载波示例

图 41.3-26 描述了载波叠加在原始 PWM 脉冲上的示例波形。图中未显示第一个脉冲和占空比控制，相关详细信息将在后两节中介绍。

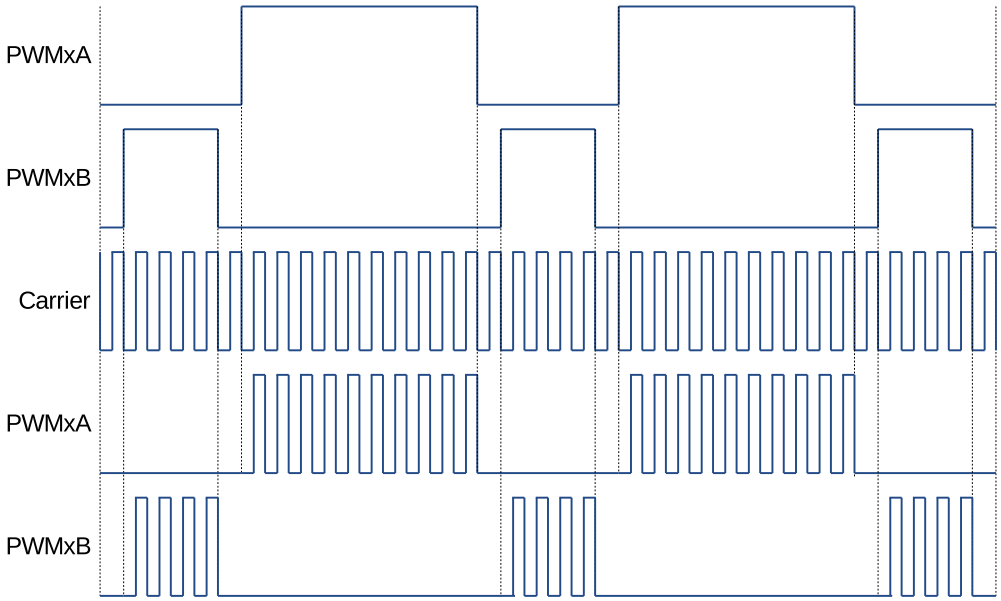


图 41.3-26. PWM 载波操作的波形示例

第一个脉冲

第一个脉冲的宽度可配置，其值有 16 种可能，可通过下公式计算：

$$T_{1stpulse} = T_{PWM\_clk} \times 8 \times (MCPWM\_CARRIERn\_PRESCALE + 1) \times (MCPWM\_CARRIERn\_OSHTWTH + 1)$$

其中：

- T_{PMW_clk} 为 PWM 时钟周期 (PWM_clk)
- $(MCPWM\_CARRIERn\_OSHTWTH + 1)$ 为一次性脉冲宽度值（取值范围：1-16）
- $(MCPWM\_CARRIERn\_PRESCALE + 1)$ PWM 载波时钟 (PC_clk) 预分频值

图 41.3-27 展示了第一个脉冲和之后持续的脉冲。

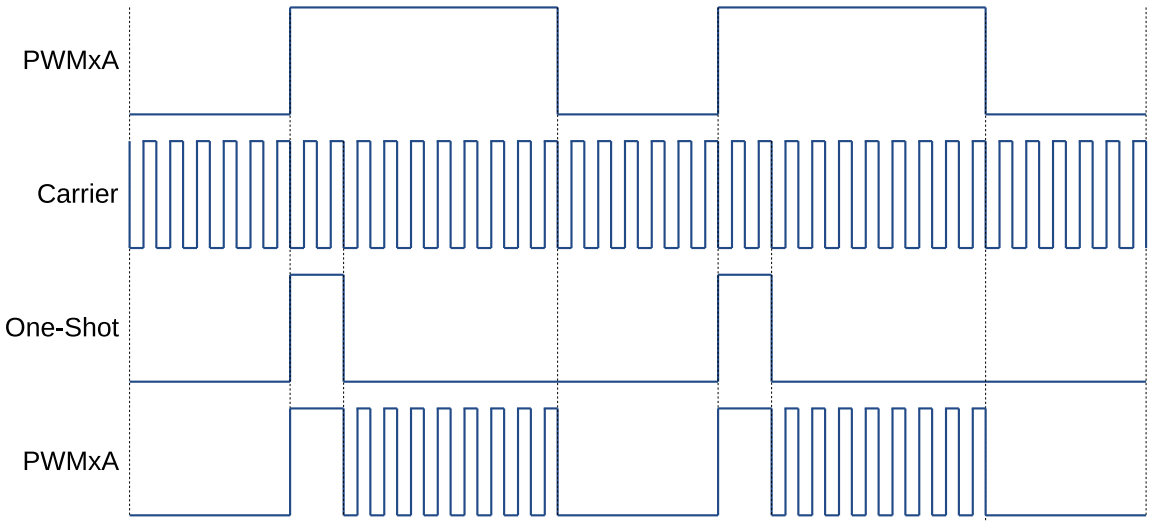


图 41.3-27. 载波模块的第一个脉冲和之后持续的脉冲示例

占空比控制

在发出第一个一次性脉冲之后，根据载波频率调制剩余的 PWM 信号。用户可配置该信号的占空比。在一定情况下，调整占空比可使信号通过隔离变压器后仍然可以开启或关闭电动机驱动器，改变电机旋转速度和方向。

占空比通过 MCPWM_CHOPPER*n*_DUTY 或寄存器 MCPWM_CARRIER*n*_CFG_REG 的 [7:5] 位设置，其值有 7 种可能性。

占空比的值可通过以下方式计算：

$$Duty = MCPWM\_CARRIERn\_DUTY \div 8$$

图 41.3-28 为所有 7 种占空比设置。

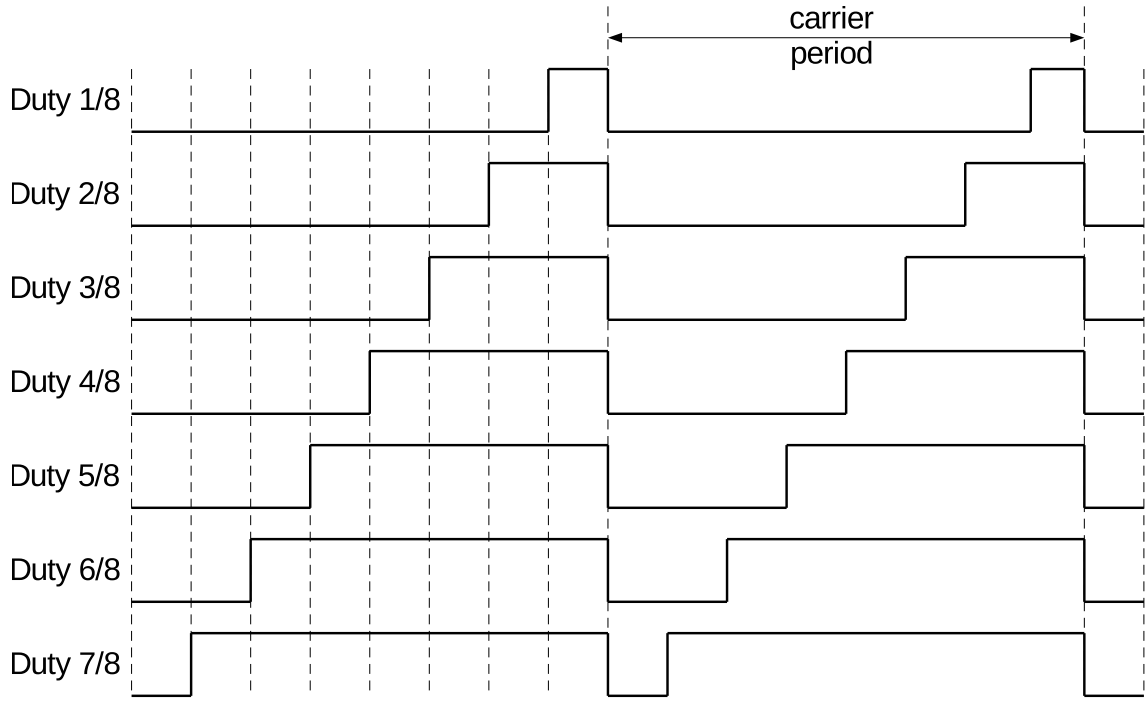


图 41.3-28. PWM 载波模块中持续脉冲的 7 种占空比设置

41.3.3.4 故障检测模块

每个 MCPWM 外设都连接来自 GPIO 矩阵的 3 个故障信号（FAULT0、FAULT1 和 FAULT2）。这些信号用于指示外部故障状况，并且可由故障检测模块预处理后生成故障事件。故障事件通过执行用户代码，针对特定故障调整 MCPWM 输出。

故障检测模块功能

故障检测模块的主要功能为：

- 在检测到故障时强制 PWM*x*A 和 PWM*x*B 输出信号进入以下状态之一：
  - 高
  - 低
  - 取反
  - 无
- 在检测到过电流/短路时执行一次性跳闸（OST）。
- 逐周期跳闸（CBC）以提供限流操作。
- 每个故障信号单独分配一次性或逐周期操作。
- 每个故障输入都生成中断。
- 支持软件强制跳闸。
- 根据需要开启或关闭模块功能。

操作与配置要点

本节介绍故障处理模块的操作要点和配置选项。

来自管脚的故障信号在 GPIO 矩阵中采样和同步。为了保证故障脉冲成功采样，每个脉冲持续时间必须至少为 2 个 APB 时钟周期。故障检测模块使用 PWM_clk 对故障信号进行采样，因此，来自 GPIO 矩阵的故障脉冲持续时间必须至少为 1 个 PWM_clk 周期。换句话说，无论 APB 时钟周期和 PWM_clk 周期的大小关系如何，管脚上的故障信号脉冲的宽度必须至少等于两个 APB 时钟周期与一个 PWM_clk 周期的和。

故障处理器模块可以使用故障信号 FAULT0 至 FAULT2 中的高电平或低电平来生成故障事件 fault_event0 至 fault_event2。每个故障事件可以单独配置为进行 CBC 操作、OST 操作或无操作。

- **逐周期 (CBC) 操作：**
  触发 CBC 操作时，PWM*x*A 和 PWM*x*B 的状态立即根据字段 MCPWM_TZ*n*_A_CBC_U/D 和 MCPWM_TZ*n*_B_CBC_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 CBC 操作中断。通过状态字段 MCPWM_TZ*n*_CBC_ON 开启或关闭 CBC 操作。在没有故障事件时，将在指定时间点，即发生 D/UTEP 或 D/UTEZ 事件时，清除 PWM*x*A/B 上的 CBC 操作。由字段 MCPWM_TZ*n*_CBCPULSE 决定 PWM*x*A 和 PWM*x*B 恢复正常的事件。因此，在此模式下，每个 PWM 循环后，CBC 操作都将清除或刷新。
- **一次性（OST）操作：**
  触发 OST 操作时，PWM*x*A 和 PWM*x*B 的状态立即根据字段 MCPWM_TZ*n*_A_OST_U/D 和 MCPWM_TZ*n*_B_OST_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 OST 操作中断。通过状态字段 MCPWM_TZ*n*_OST_ON 开启或关闭 OST 操作。在没有故障事件时，PWM*x*A/B 上的 OST 操作无法自动清除。须通过将 MCPWM_TZ*n*_CLR_OST 位置 1 来清除一次性操作。

41.3.4 捕获模块

41.3.4.1 介绍

捕获模块包含 3 个完整的捕获通道。通道输入信号 CAP0、CAP1 和 CAP2 来自于 GPIO 矩阵。由于 GPIO 矩阵的灵活性，CAP0、CAP1 和 CAP2 可以通过任一管脚输入配置。多个捕获通道可以来自同一管脚输入，每个通道的预分频可以分别设置。同时，每个捕获通道也可以来自不同的管脚输入。因此，可以通过后台硬件用多种方式处理捕获信号，而不直接由 CPU 处理。

捕捉模块有以下独立资源：

- 一个 32 位定时器（计数器），可与 PWM 定时器、另一个模块或软件同步。
- 三个捕获通道，每个通道配有一个 32 位时间戳和一个捕获预分频器。
- 任何捕获通道的边沿极性（上升/下降沿）可独立选择。
- 输入捕获信号预分频（分频取值范围：1 ~ 256）。
- 三个捕获事件都有中断功能。

41.3.4.2 捕获定时器

捕获定时器是一个 32 位计数器，使能时不断递增计数。将 MCPWM_CAP_TIMER_EN 置 1 使能捕获寄存器。其操作时钟源为 MCPWM 模块的主时钟。配置 MCPWM_CAP_SYNCI_EN 后，发生同步事件时，存储在

MCPWM_CAP_TIMER_PHASE_REG 寄存器中的相位将被加载至计数器中。同步事件可来自 PWM 定时器同步输出或 PWM 模块同步输入，通过 MCPWM_CAP_SYNCI_SEL 进行配置。也可将 MCPWM_CAP_SYNC_SW 置 1 生成同步事件。捕获定时器为所有三个捕获通道提供定时参考。

41.3.4.3 捕获通道

必要时，到达捕获通道的捕获信号可先被反相，然后预分频。每个捕获通道都有一个预分频寄存器 MCPWM_CAP*n*_PRESCALE。最后，预处理后的捕获信号的指定边沿将触发捕获事件。将 MCPWM_CAP*n*_EN 置 1 使能捕获通道。捕获事件将在 MCPWM_CAP*n*_MODE 配置的时间发生。在捕获事件发生时，捕获定时器的值存储在时间戳寄存器 MCPWM_CAP_CH*n*_REG 中。捕获事件中的不同捕获通道可生成不同的中断。触发捕获事件的边沿储存在寄存器 MCPWM_CAP*n*_EDGE 中。捕获事件可通过将 MCPWM_CAP*n*_SW 置 1 由软件强制发生。

41.3.5 ETM 模块

41.3.5.1 介绍

在 ESP32-C5 中，MCPWM 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 MCPWM 的 ETM 任务，或者通过 MCPWM 的 ETM 事件触发任意外设的 ETM 任务。MCPWM 的捕获模块、故障检测模块、三个定时器以及三个操作器都可以独立产生事件或响应任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与 MCPWM 相关的 ETM 任务和 ETM 事件。

41.3.5.2 MCPWM 可产生的 ETM 事件

将使能字段置 1 后，满足对应生成条件时，将产生对应 ETM 事件，详见表 41.3-7：

表 41.3-7. MCPWM 可产生的 ETM 事件

使能字段	生成条件	产生事件
MCPWM_EVT_CAP*n*_EN	发生 CAP*n* 捕获事件	MCPWM*0*_EVT_CAP*n*
MCPWM_EVT_TZ*n*_OST_EN	PWM 操作器 *n* 执行一次性跳闸 (OST) 操作	MCPWM*0*_EVT_TZ*n*_OST
MCPWM_EVT_TZ*n*_CBC_EN	PWM 操作器 *n* 执行逐周期跳闸 (CBC) 操作	MCPWM*0*_EVT_TZ*n*_CBC
MCPWM_EVT_F*n*_CLR_EN	清除故障事件 fault_event*n*	MCPWM*0*_EVT_F*n*_CLR
MCPWM_EVT_F*n*_EN	产生故障事件 fault_event*n*	MCPWM*0*_EVT_F*n*
MCPWM_EVT_OP*n*_TEB_EN	PWM 操作器 *n* 选择的定时器计数值等于时间戳 B 的值	MCPWM*0*_EVT_OP*n*_TEB
MCPWM_EVT_OP*n*_TEA_EN	PWM 操作器 *n* 选择的定时器计数值等于时间戳 A 的值[1]	MCPWM*0*_EVT_OP*n*_TEA
MCPWM_EVT_OP*n*_TEE1_EN	PWM 操作器 *n* 选择的定时器计数值等于寄存器 MCPWM_OP*n*_TSTMP_E1_REG 的值	MCPWM*0*_EVT_OP*n*_TEE1
MCPWM_EVT_OP*n*_TEE2_EN	PWM 操作器 *n* 选择的定时器计数值等于寄存器 MCPWM_OP*n*_TSTMP_E2_REG 的值	MCPWM*0*_EVT_OP*n*_TEE2
MCPWM_EVT_TIMER*n*_TEP_EN	定时器 *n* 的计数值等于周期值 MCPWM_TIMER*n*_PERIOD	MCPWM*0*_EVT_TIMER*n*_TEP
MCPWM_EVT_TIMER*n*_TEZ_EN	定时器 *n* 的计数值等于 0	MCPWM*0*_EVT_TIMER*n*_TEZ
MCPWM_EVT_TIMER*n*_STOP_EN	定时器 *n* 停止计数	MCPWM*0*_EVT_TIMER*n*_STOP

[1] 有关时间戳 A 和时间戳 B 的描述，详见章节 41.3.3.1。

41.3.5.3 MCPWM 可接收的 ETM 任务

将使能字段置 1 后，接收对应有效任务后，将产生对应的响应操作，详见表 41.3-8：

表 41.3-8. MCPWM 可接收的 ETM 任务

使能字段	接收的有效任务	响应操作
MCPWM_TASK_CAP*n*_EN	MCPWM0_TASK_CAP*n*	CAP*n* 通道进行捕获操作
MCPWM_TASK_CLR*n*_OST_EN	MCPWM0_TASK_CLR*n*_OST	清除 PWM 操作器 *n* 的一次性跳闸 (OST) 操作
MCPWM_TASK_TZ*n*_OST_EN	MCPWM0_TASK_TZ*n*_OST	PWM 操作器 *n* 执行一次性跳闸 (OST) 操作
MCPWM_TASK_TIMER*n*_PERIOD_UP_EN	MCPWM0_TASK_TIMER*n*_PERIOD_UP	定时器 *n* 的周期更新为周期寄存器 MCPWM_TIMER*n*_PERIOD 配置的值
MCPWM_TASK_TIMER*n*_SYNC_EN	MCPWM0_TASK_TIMER*n*_SYN	定时器 *n* 执行一次同步操作
MCPWM_TASK_GEN_STOP_EN	MCPWM0_TASK_GEN_STOP	所有定时器停止计数，所有的 PWM 操作器输出的 PWM 信号保持不变
MCPWM_TASK_CMPR*n*_B_UP_EN	MCPWM0_TASK_CMPR*n*_B_UP	将 PWM 操作器 *n* 的时间戳 B 更新为时间戳 B 影子寄存器 MCPWM_GEN*n*_B 的值
MCPWM_TASK_CMPR*n*_A_UP_EN	MCPWM0_TASK_CMPR*n*_A_UP	将 PWM 操作器 *n* 的时间戳 A 更新为时间戳 A 影子寄存器 MCPWM_GEN*n*_A 的值

41.4 中断

ESP32-C5 的 MCPWM0 会分别生成 PWM0_INTR 中断信号，并发送给中断矩阵。

PWM0_INTR 中断信号由以下内部中断源生成：

- MCPWM_TIMER*x*_STOP_INT：定时器*x* 停止后触发。
- MCPWM_TIMER*x*_TEZ_INT：由 PWM 定时器*x* TEZ 事件触发。
- MCPWM_TIMER*x*_TEP_INT：由 PWM 定时器*x* TEP 事件触发。
- MCPWM_FAULT*x*_INT：fault_event*x* 开始后触发。
- MCPWM_FAULT*x*_CLR_INT：fault_event*x* 结束后触发。
- MCPWM_CMPR*x*_TEA_INT：由 PWM 操作器*x* TEA 事件触发。
- MCPWM_CMPR*x*_TEB_INT：由 PWM 操作器*x* TEB 事件触发。
- MCPWM_TZ*x*_CBC_INT：由 PWM*x* 逐周期操作触发。
- MCPWM_TZ*x*_OST_INT：由 PWM*x* 一次性操作触发。
- MCPWM_CAP*x*_INT：由信道*x* 上捕获事件触发。

上述中断源仅在对应的中断寄存器使能字段 *interrupt_source_name*_ENA 置 1 时才能触发，将 *interrupt_source_name*_CLR 字段置 1 即清除对应中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 41.5 寄存器列表。

41.5 寄存器列表

本小节的所有地址均为相对于电机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
预分频器配置			
MCPWM_CLK_CFG_REG	配置预分频器	0x0000	R/W
MCPWM 定时器 0 配置与状态			
MCPWM_TIMER0_CFG0_REG	定时器周期与更新方法	0x0004	R/W
MCPWM_TIMER0_CFG1_REG	工作模式与开始/停止控制	0x0008	varies
MCPWM_TIMER0_SYNC_REG	PWM 定时器 0 同步功能配置寄存器	0x000C	R/W
MCPWM_TIMER0_STATUS_REG	PWM 定时器 0 状态	0x0010	RO
MCPWM 定时器 1 配置与状态			
MCPWM_TIMER1_CFG0_REG	定时器更新方式与周期	0x0014	R/W
MCPWM_TIMER1_CFG1_REG	工作模式与开始/停止控制	0x0018	varies
MCPWM_TIMER1_SYNC_REG	同步设置	0x001C	R/W
MCPWM_TIMER1_STATUS_REG	PWM 定时器 1 状态	0x0020	RO
MCPWM 定时器 2 配置与状态			
MCPWM_TIMER2_CFG0_REG	定时器方式与周期	0x0024	R/W
MCPWM_TIMER2_CFG1_REG	工作模式与开始/停止控制	0x0028	varies
MCPWM_TIMER2_SYNC_REG	同步设置	0x002C	R/W
MCPWM_TIMER2_STATUS_REG	PWM 定时器 2 状态	0x0030	RO
MCPWM 定时器常见配置			
MCPWM_TIMER_SYNCI_CFG_REG	定时器同步输入选择	0x0034	R/W
MCPWM_OPERATOR_TIMERSEL_REG	为 PWM 操作器选择特定的计时器	0x0038	R/W
MCPWM 操作器 0 配置与状态			
MCPWM_GEN0_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x003C	varies
MCPWM_GEN0_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0040	R/W
MCPWM_GEN0_TSTMP_B_REG	寄存器 B 的影子寄存器	0x0044	R/W
MCPWM_GEN0_CFG0_REG	故障事件 T0 和 T1 处理	0x0048	R/W
MCPWM_GEN0_FORCE_REG	软件强制 PWM0A 和 PWM0B 输出	0x004C	R/W
MCPWM_GEN0_A_REG	PWM0A 输出上事件触发的操作	0x0050	R/W
MCPWM_GEN0_B_REG	PWM0B 输出上事件触发的操作	0x0054	R/W
MCPWM_DT0_CFG_REG	死区配置	0x0058	R/W
MCPWM_DT0_FED_CFG_REG	FED 的影子寄存器	0x005C	R/W
MCPWM_DT0_RED_CFG_REG	RED 的影子寄存器	0x0060	R/W
MCPWM_CARRIER0_CFG_REG	载波使能与配置	0x0064	R/W
MCPWM_FH0_CFG0_REG	故障事件中 PWM0A 和 PWM0B 上的操作	0x0068	R/W
MCPWM_FH0_CFG1_REG	故障处理的软件触发	0x006C	R/W
MCPWM_FH0_STATUS_REG	故障事件状态	0x0070	RO
MCPWM 操作器 1 配置与状态			
MCPWM_GEN1_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x0074	varies

名称	描述	地址	访问
MCPWM_GEN1_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0078	R/W
MCPWM_GEN1_TSTMP_B_REG	寄存器 B 的影子寄存器	0x007C	R/W
MCPWM_GEN1_CFG0_REG	故障事件 T0 和 T1 处理	0x0080	R/W
MCPWM_GEN1_FORCE_REG	软件强制 PWM1A 和 PWM1B 输出	0x0084	R/W
MCPWM_GEN1_A_REG	PWM1A 输出上的事件触发的操作	0x0088	R/W
MCPWM_GEN1_B_REG	PWM1B 输出上的事件触发的操作	0x008C	R/W
MCPWM_DT1_CFG_REG	死区配置	0x0090	R/W
MCPWM_DT1_FED_CFG_REG	FED 的影子寄存器	0x0094	R/W
MCPWM_DT1_RED_CFG_REG	RED 的影子寄存器	0x0098	R/W
MCPWM_CARRIER1_CFG_REG	使能与配置载波	0x009C	R/W
MCPWM_FH1_CFG0_REG	故障事件中 PWM1A 和 PWM1B 输出上的操作	0x00A0	R/W
MCPWM_FH1_CFG1_REG	故障处理的软件触发	0x00A4	R/W
MCPWM_FH1_STATUS_REG	故障事件状态	0x00A8	RO
MCPWM 操作器 2 的配置与状态			
MCPWM_GEN2_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x00AC	varies
MCPWM_GEN2_TSTMP_A_REG	寄存器 A 的影子寄存器	0x00B0	R/W
MCPWM_GEN2_TSTMP_B_REG	寄存器 B 的影子寄存器	0x00B4	R/W
MCPWM_GEN2_CFG0_REG	故障事件 T0 和 T1 处理	0x00B8	R/W
MCPWM_GEN2_FORCE_REG	软件强制 PWM2A 和 PWM2B 输出	0x00BC	R/W
MCPWM_GEN2_A_REG	PWM2A 输出上的事件触发的操作	0x00C0	R/W
MCPWM_GEN2_B_REG	PWM2B 输出上的事件触发的操作	0x00C4	R/W
MCPWM_DT2_CFG_REG	死区类型的选择与配置	0x00C8	R/W
MCPWM_DT2_FED_CFG_REG	FED 影子寄存器	0x00CC	R/W
MCPWM_DT2_RED_CFG_REG	RED 影子寄存器	0x00D0	R/W
MCPWM_CARRIER2_CFG_REG	使能与配置载波	0x00D4	R/W
MCPWM_FH2_CFG0_REG	故障事件中 PWM2A 和 PWM2B 输出上的操作	0x00D8	R/W
MCPWM_FH2_CFG1_REG	故障处理的软件触发	0x00DC	R/W
MCPWM_FH2_STATUS_REG	故障事件状态	0x00E0	RO
故障检测与配置			
MCPWM_FAULT_DETECT_REG	故障检测与配置	0x00E4	varies
捕获配置与状态			
MCPWM_CAP_TIMER_CFG_REG	配置捕获定时器	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	捕获定时器同步相位	0x00EC	R/W
MCPWM_CAP_CH0_CFG_REG	捕获通道 0 的配置与使能	0x00F0	varies
MCPWM_CAP_CH1_CFG_REG	捕获通道 1 的配置与使能	0x00F4	varies
MCPWM_CAP_CH2_CFG_REG	捕获通道 2 的配置与使能	0x00F8	varies
MCPWM_CAP_CH0_REG	捕获通道 0 值的状态	0x00FC	RO
MCPWM_CAP_CH1_REG	捕获通道 1 值的状态	0x0100	RO
MCPWM_CAP_CH2_REG	捕获通道 2 值的状态	0x0104	RO
MCPWM_CAP_STATUS_REG	上一次捕获触发器的边沿	0x0108	RO
使能有效寄存器的更新			
MCPWM_UPDATE_CFG_REG	使能更新	0x010C	R/W
管理中断			

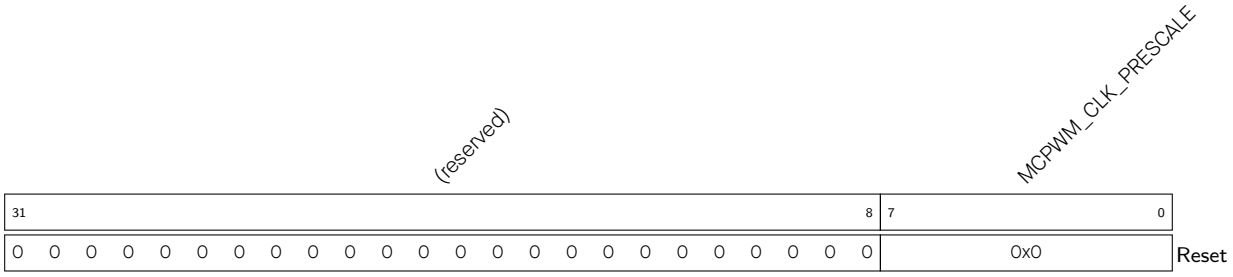
名称	描述	地址	访问
MCPWM_INT_ENA_REG	中断使能位	0x0110	R/W
MCPWM_INT_RAW_REG	原始中断状态	0x0114	R/WTC/SS
MCPWM_INT_ST_REG	屏蔽中断状态	0x0118	RO
MCPWM_INT_CLR_REG	中断清除位	0x011C	WT
使能 MCPWM 事件			
MCPWM_EVT_EN_REG	使能 MCPWM 事件	0x0120	R/W
MCPWM_EVT_EN2_REG	使能 MCPWM 事件	0x0128	R/W
使能 MCPWM 任务			
MCPWM_TASK_EN_REG	使能 MCPWM 任务	0x0124	R/W
MCPWM 生成器配置			
MCPWM_OP0_TSTMP_E1_REG	配置生成器 0 时间戳 E1 值	0x012C	R/W
MCPWM_OP0_TSTMP_E2_REG	配置生成器 0 时间戳 E2 值	0x0130	R/W
MCPWM_OP1_TSTMP_E1_REG	配置生成器 1 时间戳 E1 值	0x0134	R/W
MCPWM_OP1_TSTMP_E2_REG	配置生成器 1 时间戳 E2 值	0x0138	R/W
MCPWM_OP2_TSTMP_E1_REG	配置生成器 2 时间戳 E1 值	0x013C	R/W
MCPWM_OP2_TSTMP_E2_REG	配置生成器 2 时间戳 E2 值	0x0140	R/W
MCPWM APB 配置			
MCPWM_CLK_REG	MCPWM APB 配置	0x0144	R/W
版本寄存器			
MCPWM_VERSION_REG	版本控制寄存器	0x0148	R/W

41.6 寄存器

本小节的所有地址均为相对于电机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

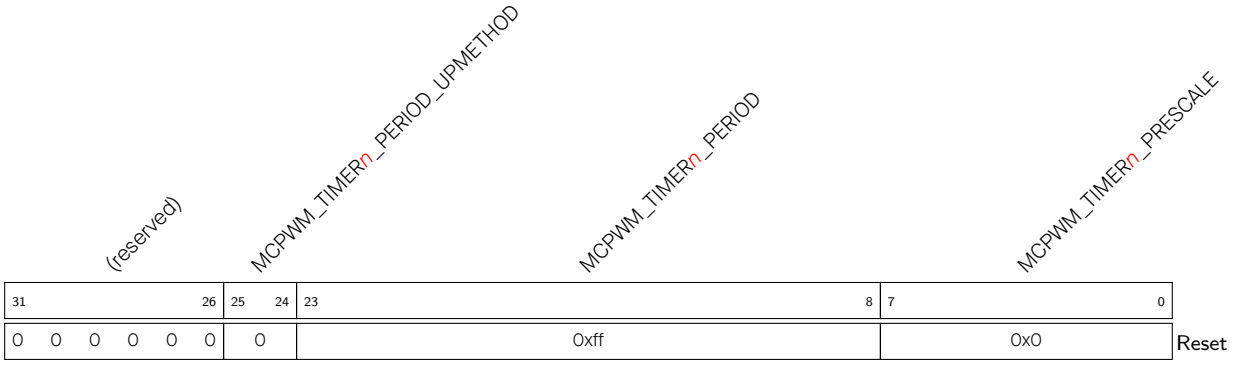
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 41.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE 配置时钟预分频系数，使得 PWM_CLK 的周期为 6.25 ns * (PWM_CLK_PRESCALE + 1)。(R/W)

Register 41.2. MCPWM_TIMER*n*_CFG0_REG (*n*: 0-2) (0x0004+0x10**n*)

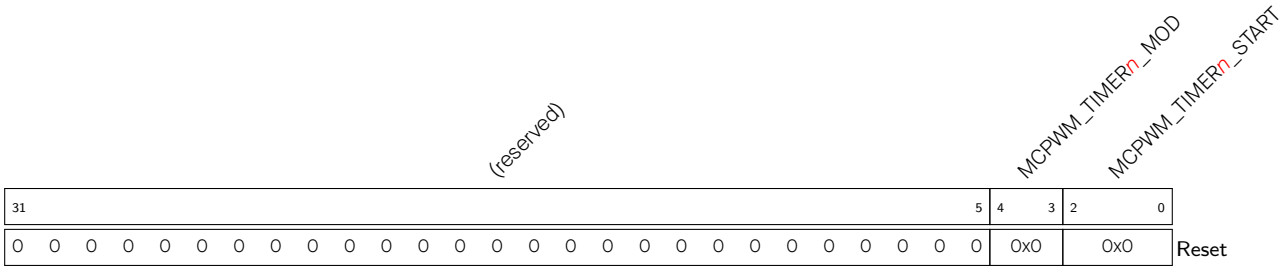


MCPWM_TIMER*n*_PRESCALE 配置定时器*n* 的预分频系数，使得 PTO_CLK 的周期 = PWM_CLK 的周期 * (PWM_TIMER*n*_PRESCALE + 1)。
(R/W)

MCPWM_TIMER*n*_PERIOD 配置定时器*n* 的影子周期。
(R/W)

MCPWM_TIMER*n*_PERIOD_UPMETHOD 配置 PWM 定时器*n* 周期有效寄存器的更新方式。
0：立即更新
1：发生 TEZ 事件时更新
2：同步时更新
3：发生 TEZ 事件或同步时更新
本文档中，TEZ 指定时器为 0 时的事件。
(R/W)

Register 41.3. MCPWM_TIMER*n*_CFG1_REG(*n*: 0-2) (0x0008+0x10**n*)



MCPWM_TIMER*n*_START 控制 PWM 定时器的开启与关闭。

0：如果开启，在 TEZ 事件发生时停止

1：如果开启，在 TEP 事件发生时停止

2：开启

3：开启，并在下一个 TEZ 事件发生时停止

4：开启，并在下一个 TEP 事件发生时停止

其他值：无效值，不产生影响

本文档中，TEP 指定时器为周期值时发生的事件。

(R/W/SC)

MCPWM_TIMER*n*_MOD 配置 PWM 定时器 0 的工作模式。

0：暂停

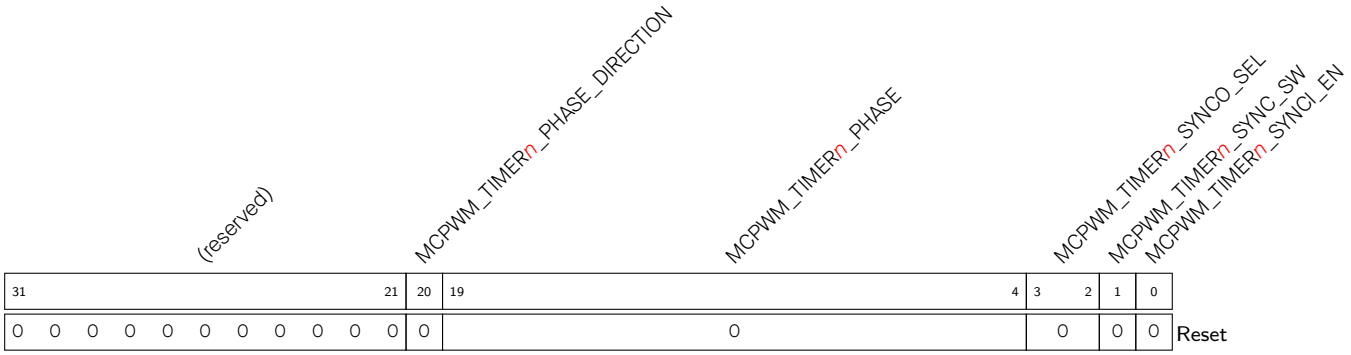
1：递增模式

2：递减模式

3：递增递减循环模式

(R/W)

Register 41.4. MCPWM_TIMER*n*_SYNC_REG (*n*: 0-2) (0x000C+0x10**n*)



MCPWM_TIMER*n*_SYNCI_EN 配置是否开启同步输入事件发生时的定时器*n* 相位重载。

0：关闭

1：开启

(R/W)

MCPWM_TIMER*n*_SYNC_SW 配置是否触发软件同步。

0：无效

1：触发软件同步

(R/W)

MCPWM_TIMER*n*_SYNCO_SEL 选择 PWM 定时器*n* 的同步输出来源。

0：同步。取反 MCPWM_TIMER*n*_SYNC_SW 位时始终生成同步输出。

1：TEZ

2：TEP

3：无效

(R/W)

MCPWM_TIMER*n*_PHASE 同步事件中定时器*n* 重载的相位。

(R/W)

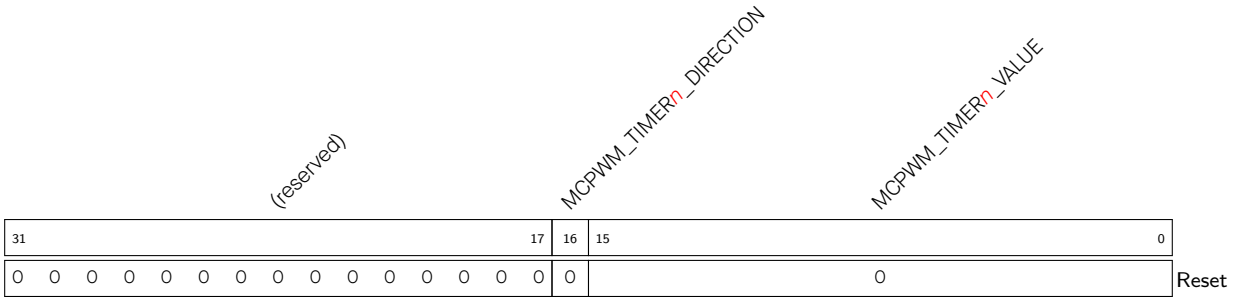
MCPWM_TIMER*n*_PHASE_DIRECTION 当定时器*n* 为递增-递减循环模式时，配置该定时器方向。

0：递增

1：递减

(R/W)

Register 41.5. MCPWM_TIMER*n*_STATUS_REG(*n*: 0-2) (0x0010+0x10**n*)



MCPWM_TIMER*n*_VALUE 表示当前 PWM 定时器*n* 计数器的值。(RO)

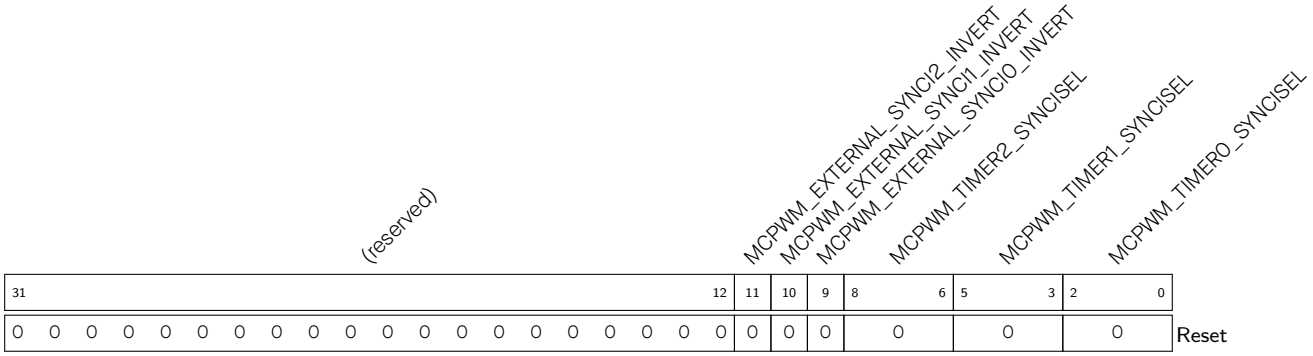
MCPWM_TIMER*n*_DIRECTION 配置当前 PWM 定时器*n* 的计数器模式。

0：递增模式

1：递减模式

(RO)

Register 41.6. MCPWM_TIMER_SYNCI_CFG_REG (0x0034)



MCPWM_TIMER*n*_SYNCISEL 选择 PWM 定时器*n* 的同步输入来源。

1：PWM 定时器 0 同步输出

2：PWM 定时器 1 同步输出

3：PWM 定时器 2 同步输出

4：来自 GPIO 矩阵的 SYNC0

5：来自 GPIO 矩阵的 SYNC1

6：来自 GPIO 矩阵的 SYNC2

其他值：未选择任何同步输入

(R/W)

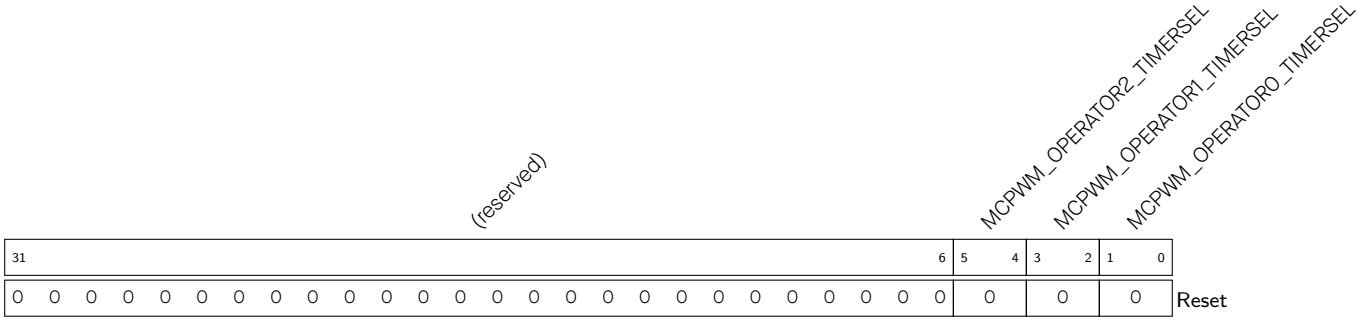
MCPWM_EXTERNAL_SYNCI*n*_INVERT 配置是否将来自 GPIO 矩阵的 SYNC*n* 反相。

0：不反相

1：反相

(R/W)

Register 41.7. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR*n*_TIMERSEL 选择 PWM 操作器*n* 的定时参考来源。

0：定时器 0

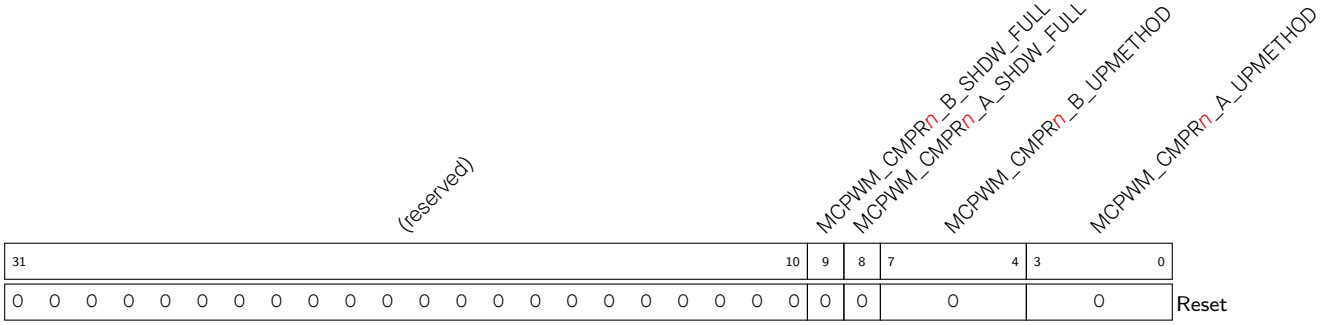
1：定时器 1

2：定时器 2

3：无效，将选择定时器 2

(R/W)

Register 41.8. MCPWM_GEN*n*_STMP_CFG_REG (*n*: 0-2) (0x003C+0x38**n*)



MCPWM_CMPR*n*_A_UPMETHOD 配置 PWM 生成器*n* 时间戳寄存器 A 的有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_CMPR*n*_B_UPMETHOD 配置 PWM 生成器*n* 时间戳寄存器 B 的有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_CMPR*n*_A_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 A 中

1：将值写入 PWM 生成器*n* 时间戳寄存器 A 的影子寄存器中，等待传输给有效寄存器 A。

(R/SC/WTC)

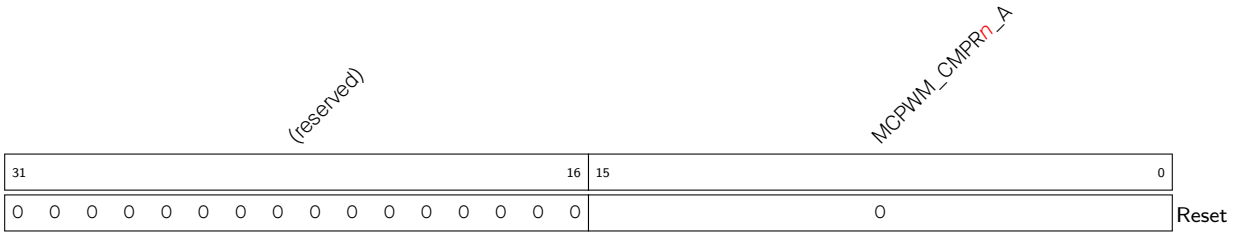
MCPWM_CMPR*n*_B_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 B 中

1：将值写入 PWM 生成器*n* 时间戳寄存器 B 的影子寄存器中，等待传输给有效寄存器 B。

(R/SC/WTC)

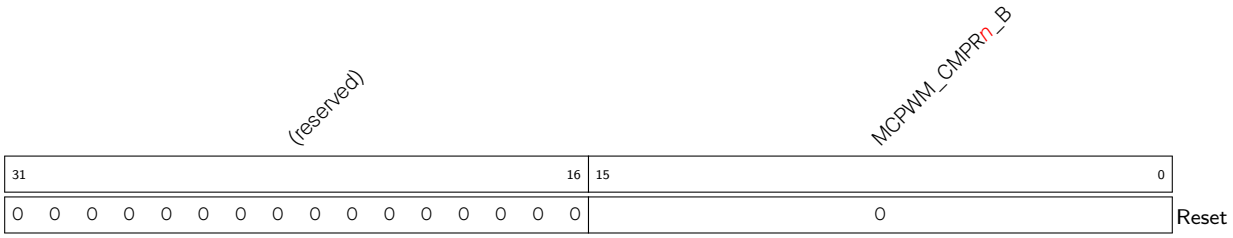
Register 41.9. MCPWM_GEN*n*_TSTMP_A_REG(*n*: 0-2) (0x0040+0x38**n*)



MCPWM_CMPR*n*_A 配置 PWM 生成器*n* 时间戳寄存器 A 的影子寄存器。

(R/W)

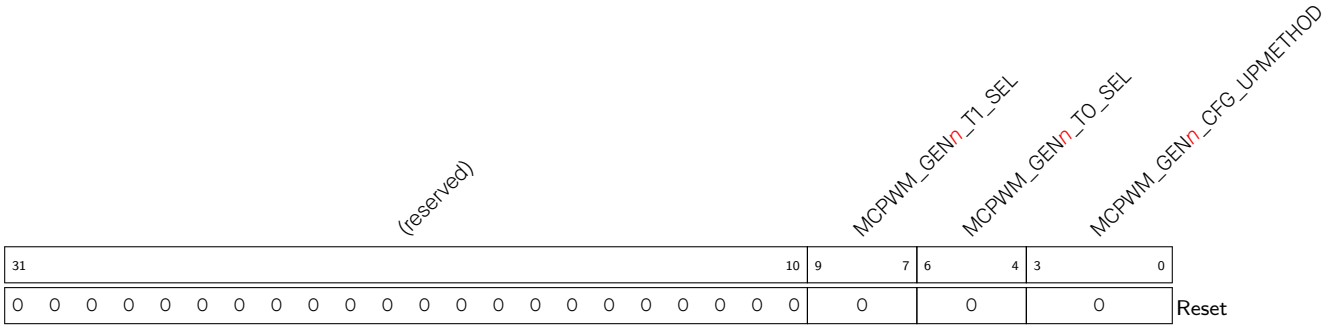
Register 41.10. MCPWM_GEN*n*_TSTMP_B_REG(*n*: 0-2) (0x0044+0x38**n*)



MCPWM_CMPR*n*_B 配置 PWM 生成器*n* 时间戳寄存器 B 的影子寄存器。

(R/W)

Register 41.11. MCPWM_GEN*n*_CFG0_REG(*n*: 0-2) (0x0048+0x38**n*)



MCPWM_GEN*n*_CFG_UPMETHOD 配置 PWM 生成器*n* 有效配置寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN*n*_T0_SEL 选择 PWM 生成器*n* event_t0 的信号源，立即生效。

0：fault_event0

1：fault_event1

2：fault_event2

3：sync_taken

4：无

(R/W)

MCPWM_GEN*n*_T1_SEL 选择 PWM 生成器*n* event_t1 的信号源，立即生效。

0：fault_event0

1：fault_event1

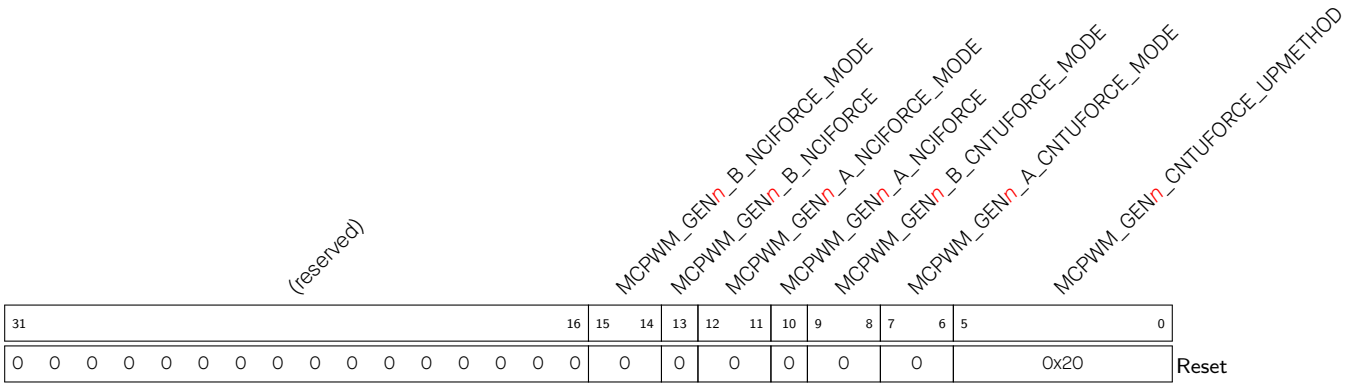
2：fault_event2

3：sync_taken

4：无

(R/W)

Register 41.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)



MCPWM_GEN*n*_CNTUFORCE_UPMETHOD 配置生成器*n* 的连续软件强制事件更新方式。

所有 bit 为 0 时：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生 TEA 事件时更新

bit3 为 1：发生 TEB 事件时更新

bit4：发生同步事件时更新

bit5 为 1：关闭更新

本文档中，TEA/B 指定时器值为寄存器 A/B 的值时生成的事件。

(R/W)

MCPWM_GEN*n*_A_CNTUFORCE_MODE 配置 PWM*n* A 的连续软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN*n*_B_CNTUFORCE_MODE 配置 PWM*n* B 的连续软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN*n*_A_NCIFORCE 配置是否触发 PWM*n* A 上的非连续即时软件强制事件。

0：无效

1：触发 PWM*n* A 上的非连续即时软件强制事件

(R/W)

见下页...

Register 41.12. MCPWM_GEN*n*_FORCE_REG(*n*: 0-2) (0x004C+0x38**n*)

... 接上页

MCPWM_GEN*n*_A_NCIFORCE_MODE 配置用于 PWM*n* A 的非连续即时软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN*n*_B_NCIFORCE 配置是否触发 PWM*n* B 上的非连续即时软件强制事件。

0：无效

1：触发 PWM*n* B 上的非连续即时软件强制事件

(R/W)

MCPWM_GEN*n*_B_NCIFORCE_MODE 配置用于 PWM*n* B 的非连续即时软件强制模式。

0：关闭

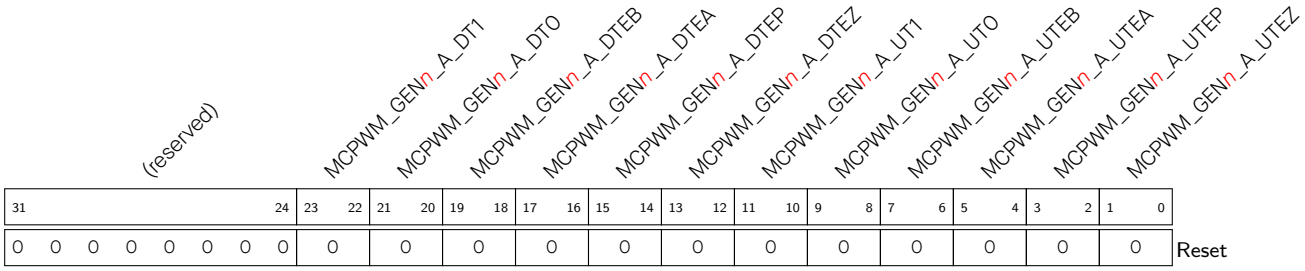
1：低电平

2：高电平

3：关闭

(R/W)

Register 41.13. MCPWM_GEN*n*_A_REG (*n*: 0-2) (0x0050+0x38**n*)



MCPWM_GEN*n*_A_UTEZ 配置定时器递增时，TEZ 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_UTEP 配置定时器递增时，TEP 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_UTEA 配置定时器递增时，TEA 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_UTEB 配置定时器递增时，TEB 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

见下页...

Register 41.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

... 接上页

MCPWM_GEN*n*_A_UT0 配置定时器递增时，event_t0 在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_UT1 配置定时器递增时，event_t1 在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_DTEZ 配置定时器递减时，TEZ 事件在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_DTEP 配置定时器递减时，TEP 事件在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_A_DTEA 配置定时器递减时，TEA 事件在 PWM*n*A 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 41.13. MCPWM_GEN*n*_A_REG(*n*: 0-2) (0x0050+0x38**n*)

... 接上页

MCPWM_GEN*n*_A_DTEB 配置定时器递减时，TEB 事件在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_DT0 配置定时器递减时，event_t0 在 PWM*n*A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_A_DT1 配置定时器递减时，event_t1 在 PWM*n*A 上触发的操作。

0：波形无改变

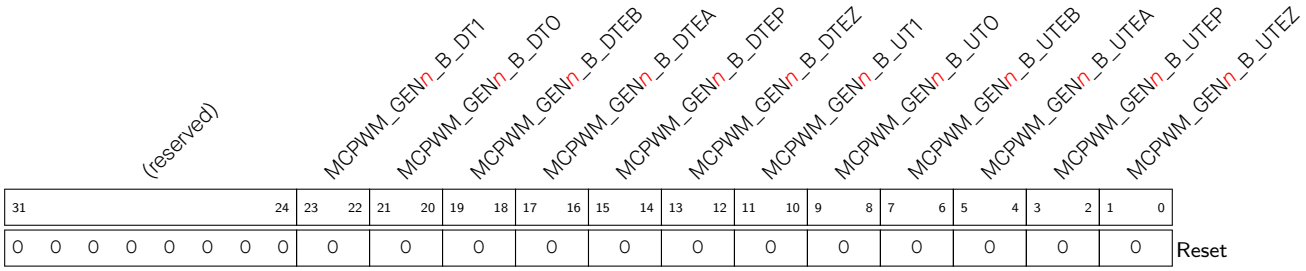
1：拉低

2：拉高

3：取反

(R/W)

Register 41.14. MCPWM_GEN*n*_B_REG (*n*: 0-2) (0x0054+0x38**n*)



MCPWM_GEN*n*_B_UTEZ 定时器递增时，TEZ 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_UTEP 定时器递增时，TEP 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_UTEA 定时器递增时，TEA 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_UTEB 定时器递增时，TEB 在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

见下页...

Register 41.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

... 接上页

MCPWM_GEN*n*_B_UT0 定时器递增时，event_t0 在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_UT1 定时器递增时，event_t1 在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_DTEZ 定时器递减时，TEZ 事件在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_DTEP 定时器递减时，TEP 事件在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN*n*_B_DTEA 定时器递减时，TEA 事件在 PWM*n*B 上触发的操作。

0：波形无改变
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 41.14. MCPWM_GEN*n*_B_REG(*n*: 0-2) (0x0054+0x38**n*)

... 接上页

MCPWM_GEN*n*_B_DTEB 定时器递减时，TEB 事件在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_DT0 定时器递减时，event_t0 事件在 PWM*n*B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN*n*_B_DT1 定时器递减时，event_t1 在 PWM*n*B 上触发的操作。

0：波形无改变

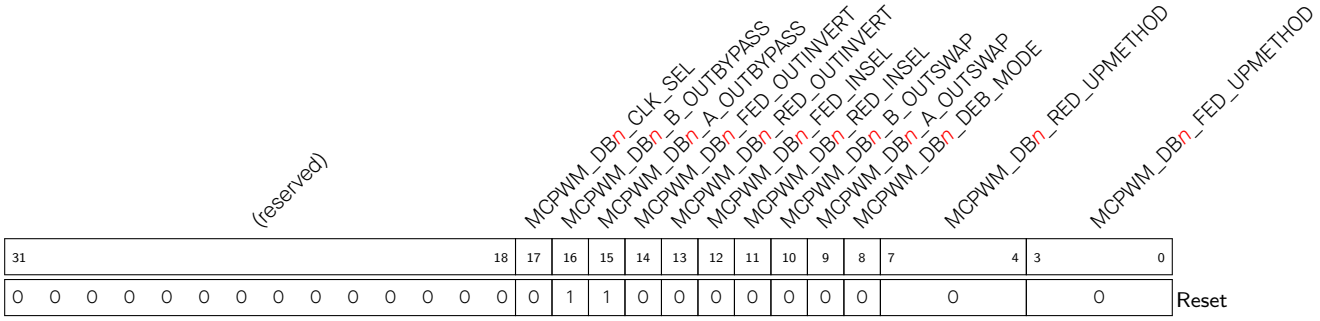
1：拉低

2：拉高

3：取反

(R/W)

Register 41.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)



MCPWM_DB*n*_FED_UPMETHOD 配置下降沿延迟有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DB*n*_RED_UPMETHOD 配置上升沿延迟有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DB*n*_DEB_MODE 配置表 41.3-5中的 S8 开关。详细配置信息请参考表 41.3-6。

0：FED/RED 分别在不同路径上生效

1：FED/RED 在 B 路径上生效，A 输出为旁路或 dulpB 模式

(R/W)

MCPWM_DB*n*_A_OUTSWAP 配置表 41.3-5中的 S6 开关。详细配置信息请参考表 41.3-6。(R/W)

MCPWM_DB*n*_B_OUTSWAP 配置表 41.3-5中的 S7 开关。详细配置信息请参考表 41.3-6。(R/W)

MCPWM_DB*n*_RED_INSEL 配置表 41.3-5中的 S4 开关。详细配置信息请参考表 41.3-6。(R/W)

MCPWM_DB*n*_FED_INSEL 配置表 41.3-5中的 S5 开关。详细配置信息请参考表 41.3-6。(R/W)

MCPWM_DB*n*_RED_OUTINVERT 配置表 41.3-5中的 S2 开关。详细配置信息请参考表 41.3-6。(R/W)

MCPWM_DB*n*_FED_OUTINVERT 配置表 41.3-5中的 S3 开关。详细配置信息请参考表 41.3-6。(R/W)

MCPWM_DB*n*_A_OUTBYPASS 配置表 41.3-5中的 S1 开关。详细配置信息请参考表 41.3-6。(R/W)

MCPWM_DB*n*_B_OUTBYPASS 配置表 41.3-5中的 S0 开关。详细配置信息请参考表 41.3-6。(R/W)

见下页...

Register 41.15. MCPWM_DT*n*_CFG_REG(*n*: 0-2) (0x0058+0x38**n*)

... 接上页

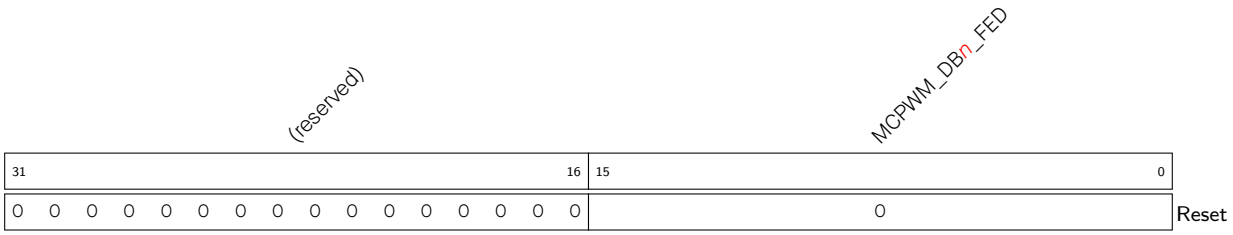
MCPWM_DB*n*_CLK_SEL 选择死区时间生成器 0 的时钟。

0：PWM_CLK

1：PT_CLK

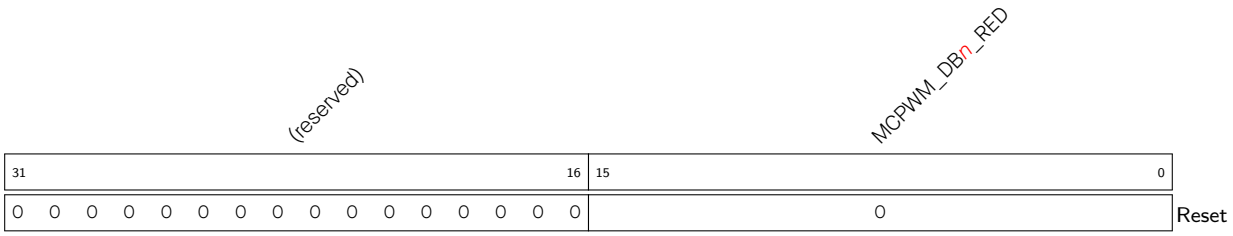
(R/W)

Register 41.16. MCPWM_DT*n*_FED_CFG_REG(*n*: 0-2) (0x005C+0x38**n*)



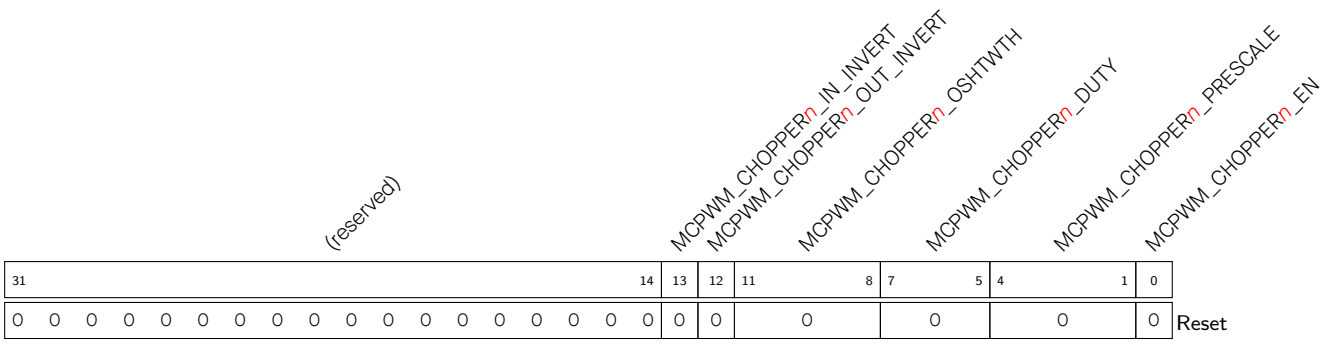
MCPWM_DB*n*_FED 配置 FED 的影子寄存器。(R/W)

Register 41.17. MCPWM_DT*n*_RED_CFG_REG(*n*: 0-2) (0x0060+0x38**n*)



MCPWM_DB*n*_RED 配置 RED 的影子寄存器。(R/W)

Register 41.18. MCPWM_CARRIER*n*_CFG_REG(*n*: 0-2) (0x0064+0x38**n*)



MCPWM_CHOPPER*n*_EN 配置是否使能载波*n*。

0：绕过

1：使能

(R/W)

MCPWM_CHOPPER*n*_PRESCALE 配置 PWM 载波*n* 时钟 (PC_CLK) 的预分频值。PC_CLK 周期 = PWM_CLK 周期 * (PWM_CARRIER*n*_PRESCALE + 1)。(R/W)

MCPWM_CHOPPER*n*_DUTY 配置载波占空比。占空比 = PWM_CARRIER*n*_DUTY/8。(R/W)

MCPWM_CHOPPER*n*_OSHTWTH 配置载波第一个脉冲的宽度，单位为载波周期。(R/W)

MCPWM_CHOPPER*n*_OUT_INVERT 配置是否将此模块的 PWM*n*A 和 PWM*n*B 输出反相。

0：正常输出

1：反相

(R/W)

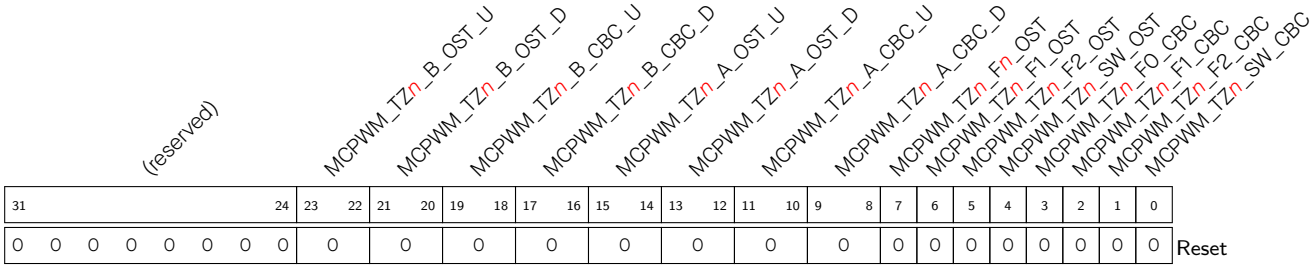
MCPWM_CHOPPER*n*_IN_INVERT 配置是否将此模块的 PWM*n*A 和 PWM*n*B 输入反相。

0：正常输入

1：反相

(R/W)

Register 41.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)



MCPWM_TZ*n*_SW_CBC 配置是否使能软件强制逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F2_CBC 配置 fault_event2 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F1_CBC 配置 fault_event1 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F0_CBC 配置 fault_event0 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_SW_OST 配置是否使能软件强制一次性模式操作。

0：关闭

1：使能

(R/W)

MCPWM_TZ*n*_F2_OST 配置 fault_event2 是否触发一次性模式操作。

0：关闭

1：使能

(R/W)

见下页...

Register 41.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

... 接上页

MCPWM_TZ*n*_F1_OST 配置 fault_event1 是否触发一次性模式操作。

0：关闭
1：使能
(R/W)

MCPWM_TZ*n*_F*n*_OST 配置 fault_event0 是否触发一次性模式操作。

0：关闭
1：使能
(R/W)

MCPWM_TZ*n*_A_CBC_D 配置定时器递减计数并且发生故障事件时，PWM*n*A 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_A_CBC_U 配置定时器递增计数并且发生故障事件时，PWM*n*A 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_A_OST_D 配置定时器递减计数并且发生故障事件时，PWM*n*A 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

见下页...

Register 41.19. MCPWM_FH*n*_CFG0_REG(*n*: 0-2) (0x0068+0x38**n*)

... 接上页

MCPWM_TZ*n*_A_OST_U 配置定时器递增计数并且发生故障事件时，PWM*n*A 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_B_CBC_D 配置定时器递减计数并且发生故障事件时，PWM*n*B 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_B_CBC_U 配置定时器递增计数并且发生故障事件时，PWM*n*B 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

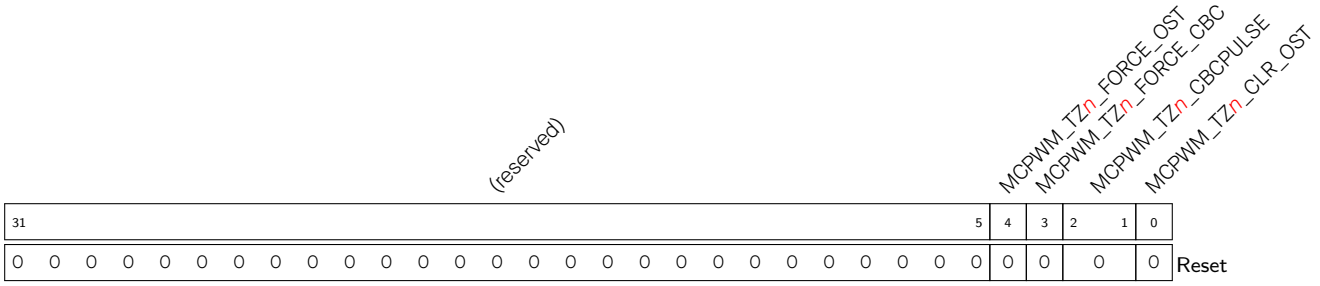
MCPWM_TZ*n*_B_OST_D 配置定时器递减计数并且发生故障事件时，PWM*n*B 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_TZ*n*_B_OST_U 配置定时器递增计数并且发生故障事件时，PWM*n*B 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

Register 41.20. MCPWM_FH*n*_CFG1_REG(*n*: 0-2) (0x006C+0x38**n*)



MCPWM_TZ*n*_CLR_OST 配置是否通过软件清除持续一次性模式操作。

0：无效

1：通过软件清除持续一次性模式操作

(R/W)

MCPWM_TZ*n*_CBCPULSE 配置逐周期模式操作更新的时间点。

bit0 为 1：发生 TEZ 事件时

bit1 为 1：发生 TEP 事件时

(R/W)

MCPWM_TZ*n*_FORCE_CBC 配置是否触发逐周期模式的操作。

0：无效

1：触发

(R/W)

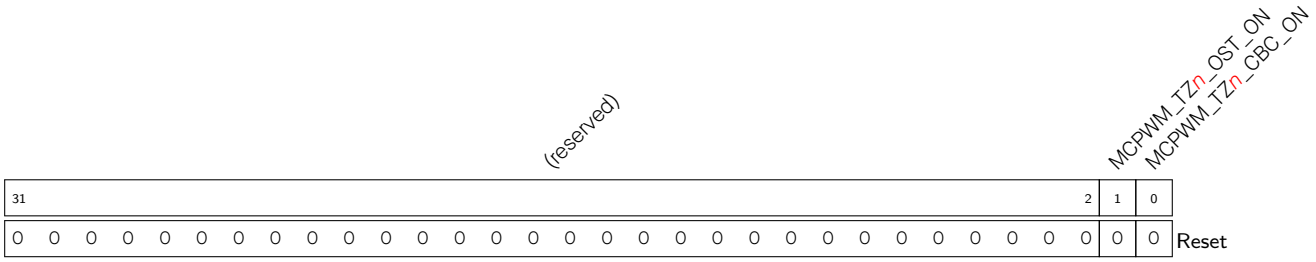
MCPWM_TZ*n*_FORCE_OST 配置是否触发一次性模式的操作。

0：无效

1：触发

(R/W)

Register 41.21. MCPWM_FH*n*_STATUS_REG(*n*: 0-2) (0x0070+0x38**n*)



MCPWM_TZ*n*_CBC_ON 表示是否正在进行逐周期模式的操作。

0：不存在正在进行的逐周期模式操作

1：逐周期模式的操作正在进行

(RO)

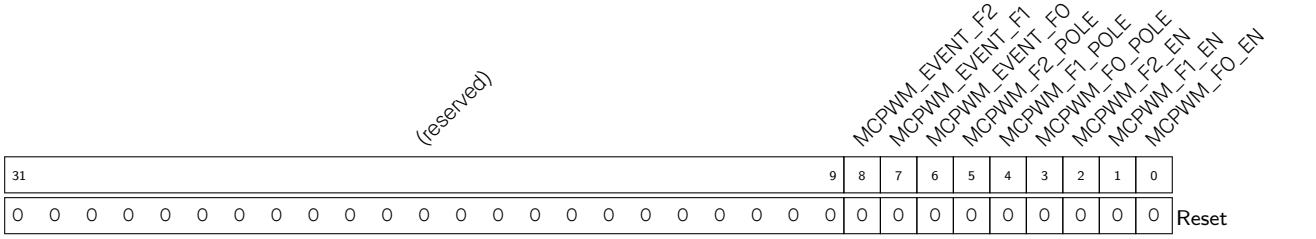
MCPWM_TZ*n*_OST_ON 表示是否正在进行一次性模式的操作。

0：不存在正在进行的一次性模式操作

1：一次性模式的操作正在进行

(RO)

Register 41.22. MCPWM_FAULT_DETECT_REG (0x00E4)



MCPWM_F*n*_EN 配置是否使能生成 fault_event*n*。

0：无效

1：使能

(R/W)

MCPWM_F*n*_POLE 配置来自 GPIO 矩阵的 FAULT*n* 信号源触发 fault_event*n* 时极性。

0：低电平触发

1：高电平触发

(R/W)

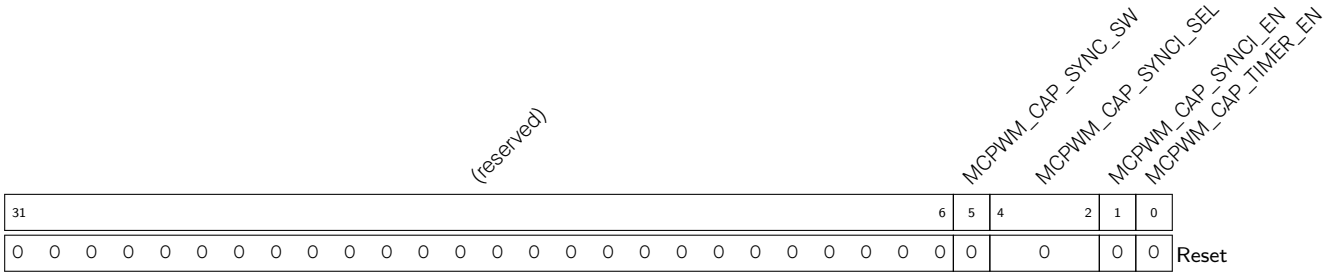
MCPWM_EVENT_F*n* 表示 fault_event*n* 事件的状态。此字段由硬件置 1 和清零。

0：fault_event*n* 事件停止

1：fault_event*n* 事件持续

(RO)

Register 41.23. MCPWM_CAP_TIMER_CFG_REG (0x00E8)



MCPWM_CAP_TIMER_EN 配置是否使能捕获定时器在 APB_CLK 下的递增。

0：无效

1：使能捕获定时器在 APB_CLK 下的递增

(R/W)

MCPWM_CAP_SYNCI_EN 配置是否使能捕获定时器同步。

0：无效

1：使能捕获定时器同步

(R/W)

MCPWM_CAP_SYNCI_SEL 配置捕获模块的同步输入。

0：无

1：定时器 0 的同步输出

2：定时器 1 的同步输出

3：定时器 2 的同步输出

4：来自 GPIO 矩阵的 SYNC0

5：来自 GPIO 矩阵的 SYNC1

6：来自 GPIO 矩阵的 SYNC2

7：无

(R/W)

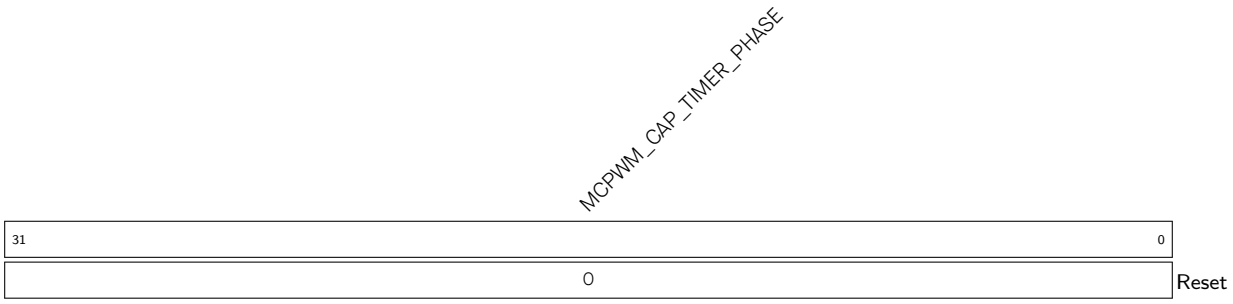
MCPWM_CAP_SYNC_SW 当 MCPWM_CAP_SYNCI_EN 置位时，配置是否同步捕获定时器，在捕获定时器中写入相位寄存器的值。

0：无效

1：同步捕获定时器

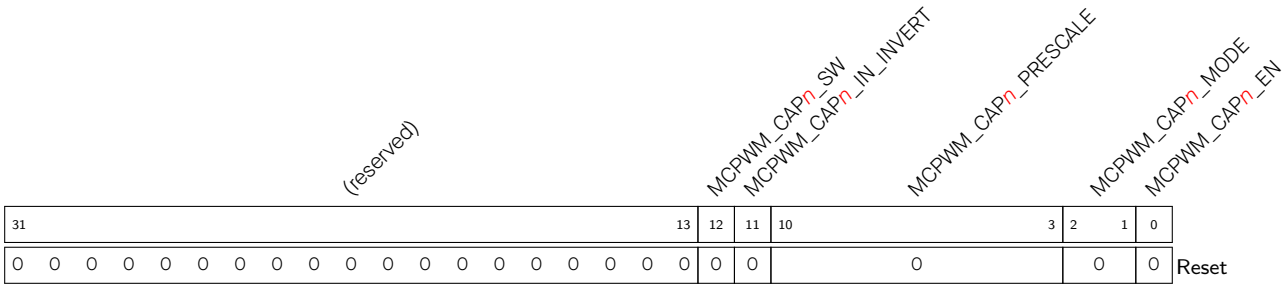
(WT)

Register 41.24. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM_CAP_TIMER_PHASE 配置捕获定时器同步操作的相位值。(R/W)

Register 41.25. MCPWM_CAP_CH*n*_CFG_REG (*n*: 0-2) (0x00F0+0x4*n*)



MCPWM_CAP*n*_EN 配置是否使能信道 *n* 上的捕获事件。

0：不使能

1：使能

(R/W)

MCPWM_CAP*n*_MODE 配置预分频后信道 *n* 上的捕获沿。

0：无

bit0 为 1：使能下降沿捕获。

bit1 为 1：使能上升沿捕获。

(R/W)

MCPWM_CAP*n*_PRESCALE 配置 CAP*n* 上升沿的预分频值。预分频值 = PWM_CAP*n*_PRESCALE + 1。(R/W)

MCPWM_CAP*n*_IN_INVERT 配置是否在预分频前反相来自 GPIO 矩阵的 CAP*n*。

0：无效

1：在预分频前反相来自 GPIO 矩阵的 CAP*n*

(R/W)

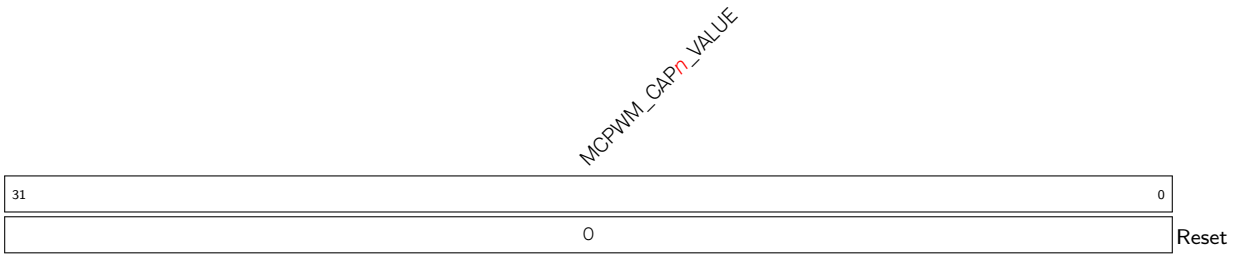
MCPWM_CAP*n*_SW 配置是否触发信道 0 上的软件强制捕获事件。

0：不触发

1：触发

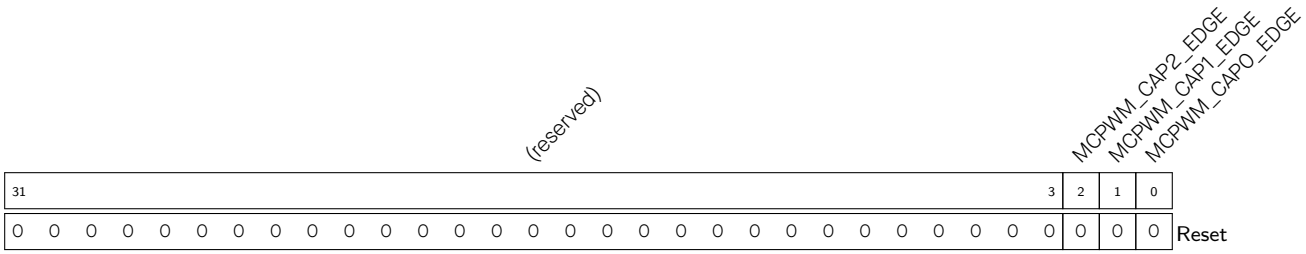
(WT)

Register 41.26. MCPWM_CAP_CH*n*_REG (*n*: 0-2) (0x00FC+0x4**n*)



MCPWM_CAP*n*_VALUE 表示信道 *n* 上一次捕获的值。(RO)

Register 41.27. MCPWM_CAP_STATUS_REG (0x0108)



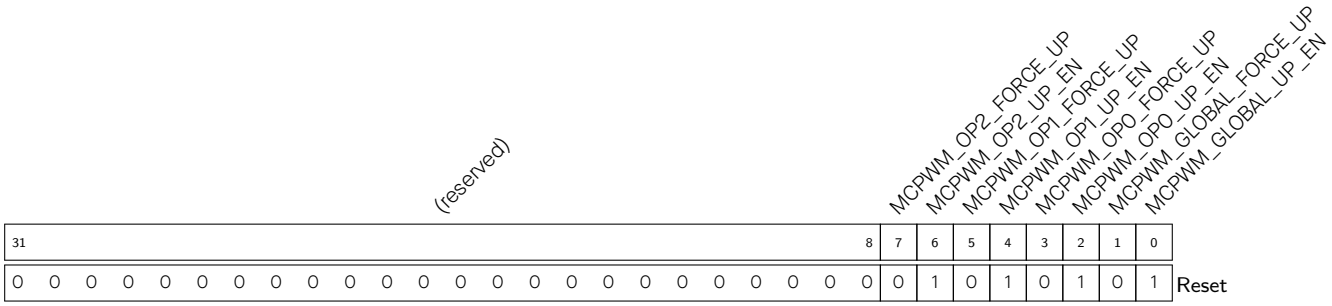
MCPWM_CAP*n*_EDGE 信道*n* 上一次捕获触发事件的边沿。

0：上升沿

1：下降沿

(RO)

Register 41.28. MCPWM_UPDATE_CFG_REG (0x010C)



MCPWM_GLOBAL_UP_EN 配置是否更新 MCPWM 模块的所有有效寄存器。

0：无效

1：更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_GLOBAL_FORCE_UP 配置是否强制更新 MCPWM 模块的所有有效寄存器。

0：无效

1：强制更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_OP0_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 0 中的有效寄存器。

0：无效

1：更新 PWM 操作器 0 中的有效寄存器

(R/W)

MCPWM_OP0_FORCE_UP 配置是否强制更新 PWM 操作器 0 中的有效寄存器。

0：无效

1：强制更新

(R/W)

MCPWM_OP1_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 1 中的有效寄存器。

0：无效

1：更新 PWM 操作器 1 中的有效寄存器

(R/W)

MCPWM_OP1_FORCE_UP 配置是否强制更新 PWM 操作器 1 中的有效寄存器。

0：无效

1：强制更新

(R/W)

MCPWM_OP2_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 2 中的有效寄存器。

0：无效

1：更新 PWM 操作器 2 中的有效寄存器

(R/W)

见下页...

Register 41.28. MCPWM_UPDATE_CFG_REG (0x010C)

... 接上页

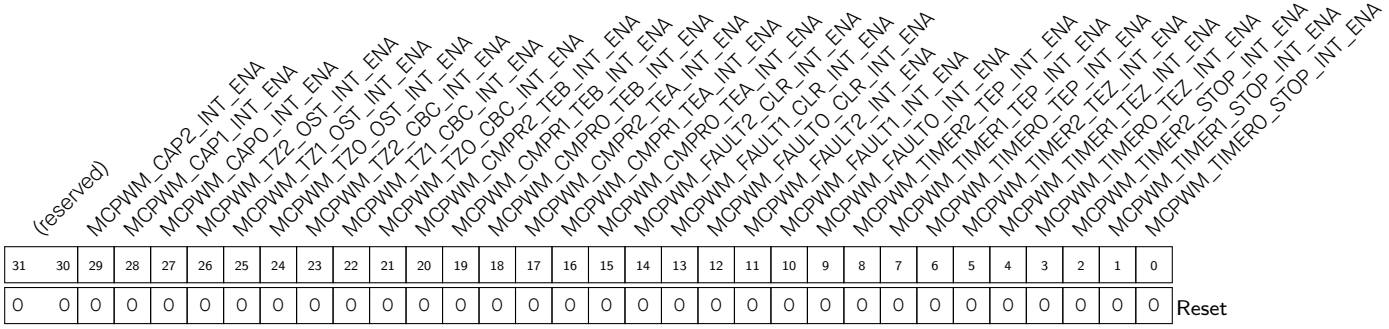
MCPWM_OP2_FORCE_UP 配置是否强制更新 PWM 操作器 2 中的有效寄存器。

0：无效

1：强制更新

(R/W)

Register 41.29. MCPWM_INT_ENA_REG (0x0110)



31 30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_ENA 写 1 使能 MCPWM_TIMER0_STOP_INT。(R/W)

MCPWM_TIMER1_STOP_INT_ENA 写 1 使能 MCPWM_TIMER1_STOP_INT。(R/W)

MCPWM_TIMER2_STOP_INT_ENA 写 1 使能 MCPWM_TIMER2_STOP_INT。(R/W)

MCPWM_TIMER0_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER0_TEZ_INT。(R/W)

MCPWM_TIMER1_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER1_TEZ_INT。(R/W)

MCPWM_TIMER2_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER2_TEZ_INT。(R/W)

MCPWM_TIMER0_TEP_INT_ENA 写 1 使能 MCPWM_TIMER0_TEP_INT。(R/W)

MCPWM_TIMER1_TEP_INT_ENA 写 1 使能 MCPWM_TIMER1_TEP_INT。(R/W)

MCPWM_TIMER2_TEP_INT_ENA 写 1 使能 MCPWM_TIMER2_TEP_INT。(R/W)

MCPWM_FAULT0_INT_ENA 写 1 使能 MCPWM_FAULT0_INT。(R/W)

MCPWM_FAULT1_INT_ENA 写 1 使能 MCPWM_FAULT1_INT。(R/W)

MCPWM_FAULT2_INT_ENA 写 1 使能 MCPWM_FAULT2_INT。(R/W)

MCPWM_FAULT0_CLR_INT_ENA 写 1 使能 MCPWM_FAULT0_CLR_INT。(R/W)

MCPWM_FAULT1_CLR_INT_ENA 写 1 使能 MCPWM_FAULT1_CLR_INT。(R/W)

MCPWM_FAULT2_CLR_INT_ENA 写 1 使能 MCPWM_FAULT2_CLR_INT。(R/W)

MCPWM_CMPR0_TEA_INT_ENA 写 1 使能 MCPWM_CMPR0_TEA_INT。(R/W)

MCPWM_CMPR1_TEA_INT_ENA 写 1 使能 MCPWM_CMPR1_TEA_INT。(R/W)

MCPWM_CMPR2_TEA_INT_ENA 写 1 使能 MCPWM_CMPR2_TEA_INT。(R/W)

见下页...

Register 41.29. MCPWM_INT_ENA_REG (0x0110)

... 接上页

MCPWM_CMPR0_TEB_INT_ENA 写 1 使能 MCPWM_CMPR0_TEB_INT。(R/W)

MCPWM_CMPR1_TEB_INT_ENA 写 1 使能 MCPWM_CMPR1_TEB_INT。(R/W)

MCPWM_CMPR2_TEB_INT_ENA 写 1 使能 MCPWM_CMPR2_TEB_INT。(R/W)

MCPWM_TZ0_CBC_INT_ENA 写 1 使能 MCPWM_TZ0_CBC_INT。(R/W)

MCPWM_TZ1_CBC_INT_ENA 写 1 使能 MCPWM_TZ1_CBC_INT。(R/W)

MCPWM_TZ2_CBC_INT_ENA 写 1 使能 MCPWM_TZ2_CBC_INT。(R/W)

MCPWM_TZ0_OST_INT_ENA 写 1 使能 MCPWM_TZ0_OST_INT。(R/W)

MCPWM_TZ1_OST_INT_ENA 写 1 使能 MCPWM_TZ1_OST_INT。(R/W)

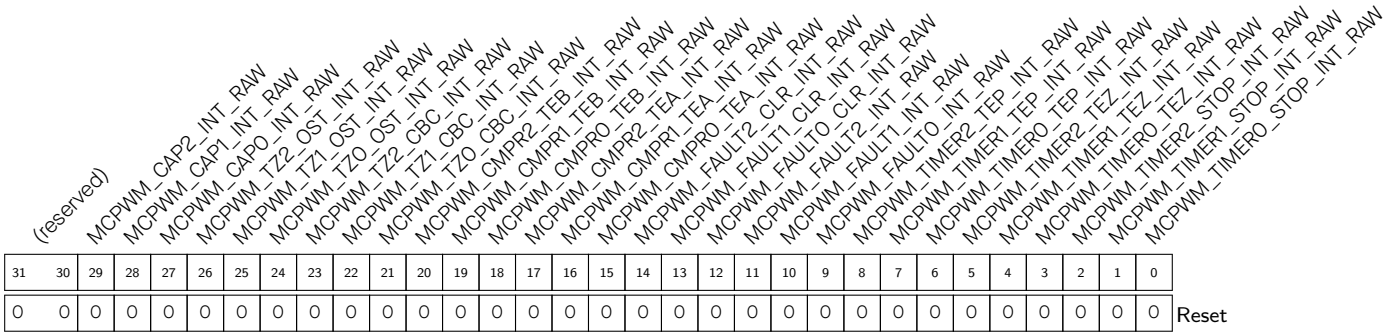
MCPWM_TZ2_OST_INT_ENA 写 1 使能 MCPWM_TZ2_OST_INT。(R/W)

MCPWM_CAP0_INT_ENA 写 1 使能 MCPWM_CAP0_INT。(R/W)

MCPWM_CAP1_INT_ENA 写 1 使能 MCPWM_CAP1_INT。(R/W)

MCPWM_CAP2_INT_ENA 写 1 使能 MCPWM_CAP2_INT。(R/W)

Register 41.30. MCPWM_INT_RAW_REG (0x0114)



MCPWM_TIMER0_STOP_INT_RAW MCPWM_TIMER0_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_STOP_INT_RAW MCPWM_TIMER1_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_STOP_INT_RAW MCPWM_TIMER2_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEZ_INT_RAW MCPWM_TIMER0_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEZ_INT_RAW MCPWM_TIMER1_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEZ_INT_RAW MCPWM_TIMER2_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEP_INT_RAW MCPWM_TIMER0_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEP_INT_RAW MCPWM_TIMER1_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEP_INT_RAW MCPWM_TIMER2_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_INT_RAW MCPWM_FAULT0_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_INT_RAW MCPWM_FAULT1_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_INT_RAW MCPWM_FAULT2_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_CLR_INT_RAW MCPWM_FAULT0_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_CLR_INT_RAW MCPWM_FAULT1_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_CLR_INT_RAW MCPWM_FAULT2_CLR_INT 的原始状态位。(R/WTC/SS)

见下页...

Register 41.30. MCPWM_INT_RAW_REG (0x0114)

... 接上页

MCPWM_CMPR0_TEA_INT_RAW MCPWM_CMPR0_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEA_INT_RAW MCPWM_CMPR1_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEA_INT_RAW MCPWM_CMPR2_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR0_TEB_INT_RAW MCPWM_CMPR0_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEB_INT_RAW MCPWM_CMPR1_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEB_INT_RAW MCPWM_CMPR2_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ0_CBC_INT_RAW MCPWM_TZ0_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_CBC_INT_RAW MCPWM_TZ1_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ2_CBC_INT_RAW MCPWM_TZ2_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ0_OST_INT_RAW MCPWM_TZ0_OST_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_OST_INT_RAW MCPWM_TZ1_OST_INT 的原始状态位。(R/WTC/SS)

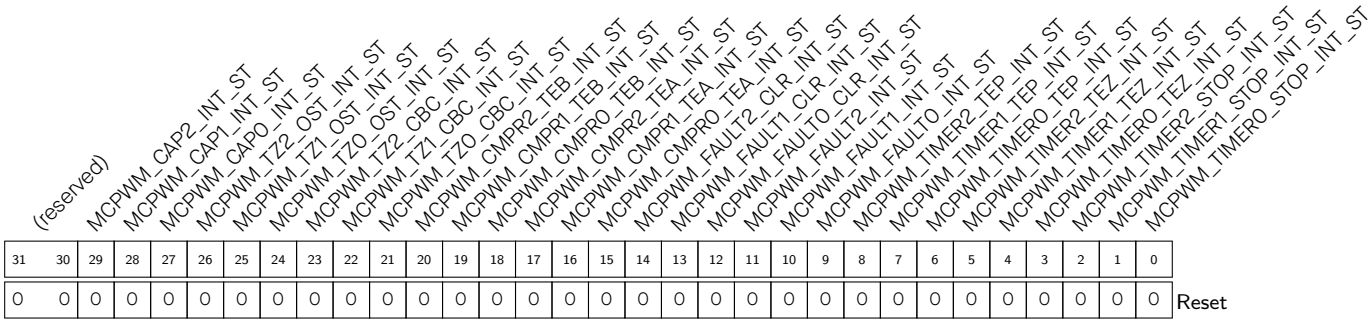
MCPWM_TZ2_OST_INT_RAW MCPWM_TZ2_OST_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP0_INT_RAW MCPWM_CAP0_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP1_INT_RAW MCPWM_CAP1_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP2_INT_RAW MCPWM_CAP2_INT 的原始状态位。(R/WTC/SS)

Register 41.31. MCPWM_INT_ST_REG (0x0118)



MCPWM_TIMER0_STOP_INT_ST MCPWM_TIMER0_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_STOP_INT_ST MCPWM_TIMER1_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_STOP_INT_ST MCPWM_TIMER2_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEZ_INT_ST MCPWM_TIMER0_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEZ_INT_ST MCPWM_TIMER1_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEZ_INT_ST MCPWM_TIMER2_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEP_INT_ST MCPWM_TIMER0_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEP_INT_ST MCPWM_TIMER1_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEP_INT_ST MCPWM_TIMER2_TEP_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_INT_ST MCPWM_FAULT0_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_INT_ST MCPWM_FAULT1_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_INT_ST MCPWM_FAULT2_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_CLR_INT_ST MCPWM_FAULT0_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_CLR_INT_ST MCPWM_FAULT1_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_CLR_INT_ST MCPWM_FAULT2_CLR_INT 的屏蔽状态位。(RO)

MCPWM_CMPR0_TEA_INT_ST MCPWM_CMPR0_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEA_INT_ST MCPWM_CMPR1_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEA_INT_ST MCPWM_CMPR2_TEA_INT 的屏蔽状态位。(RO)

见下页...

Register 41.31. MCPWM_INT_ST_REG (0x0118)

... 接上页

MCPWM_CMPR0_TEB_INT_ST MCPWM_CMPR0_TEB_INT 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEB_INT_ST MCPWM_CMPR1_TEB_INT 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEB_INT_ST MCPWM_CMPR2_TEB_INT 的屏蔽状态位。(RO)

MCPWM_TZ0_CBC_INT_ST MCPWM_TZ0_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ1_CBC_INT_ST MCPWM_TZ1_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ2_CBC_INT_ST MCPWM_TZ2_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ0_OST_INT_ST MCPWM_TZ0_OST_INT 的屏蔽状态位。(RO)

MCPWM_TZ1_OST_INT_ST MCPWM_TZ1_OST_INT 的屏蔽状态位。(RO)

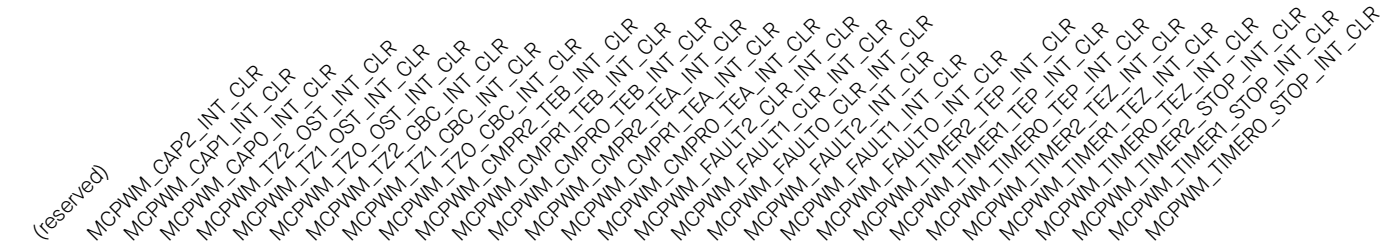
MCPWM_TZ2_OST_INT_ST MCPWM_TZ2_OST_INT 的屏蔽状态位。(RO)

MCPWM_CAP0_INT_ST MCPWM_CAP0_INT 的屏蔽状态位。(RO)

MCPWM_CAP1_INT_ST MCPWM_CAP1_INT 的屏蔽状态位。(RO)

MCPWM_CAP2_INT_ST MCPWM_CAP2_INT 的屏蔽状态位。(RO)

Register 41.32. MCPWM_INT_CLR_REG (0x011C)



MCPWM_TIMER0_STOP_INT_CLR 写 1 清除 MCPWM_TIMER0_STOP_INT。(WT)

MCPWM_TIMER1_STOP_INT_CLR 写 1 清除 MCPWM_TIMER1_STOP_INT。(WT)

MCPWM_TIMER2_STOP_INT_CLR 写 1 清除 MCPWM_TIMER2_STOP_INT。(WT)

MCPWM_TIMER0_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER0_TEZ_INT。(WT)

MCPWM_TIMER1_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER1_TEZ_INT。(WT)

MCPWM_TIMER2_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER2_TEZ_INT。(WT)

MCPWM_TIMER0_TEP_INT_CLR 写 1 清除 MCPWM_TIMER0_TEP_INT。(WT)

MCPWM_TIMER1_TEP_INT_CLR 写 1 清除 MCPWM_TIMER1_TEP_INT。(WT)

MCPWM_TIMER2_TEP_INT_CLR 写 1 清除 MCPWM_TIMER2_TEP_INT。(WT)

MCPWM_FAULT0_INT_CLR 写 1 清除 MCPWM_FAULT0_INT。(WT)

MCPWM_FAULT1_INT_CLR 写 1 清除 MCPWM_FAULT1_INT。(WT)

MCPWM_FAULT2_INT_CLR 写 1 清除 MCPWM_FAULT2_INT。(WT)

MCPWM_FAULT0_CLR_INT_CLR 写 1 清除 MCPWM_FAULT0_CLR_INT。(WT)

MCPWM_FAULT1_CLR_INT_CLR 写 1 清除 MCPWM_FAULT1_CLR_INT。(WT)

MCPWM_FAULT2_CLR_INT_CLR 写 1 清除 MCPWM_FAULT2_CLR_INT。(WT)

见下页...

Register 41.32. MCPWM_INT_CLR_REG (0x011C)

... 接上页

MCPWM_CMPR0_TEA_INT_CLR 写 1 清除 MCPWM_CMPR0_TEA_INT。(WT)

MCPWM_CMPR1_TEA_INT_CLR 写 1 清除 MCPWM_CMPR1_TEA_INT。(WT)

MCPWM_CMPR2_TEA_INT_CLR 写 1 清除 MCPWM_CMPR2_TEA_INT。(WT)

MCPWM_CMPR0_TEB_INT_CLR 写 1 清除 MCPWM_CMPR0_TEB_INT。(WT)

MCPWM_CMPR1_TEB_INT_CLR 写 1 清除 MCPWM_CMPR1_TEB_INT。(WT)

MCPWM_CMPR2_TEB_INT_CLR 写 1 清除 MCPWM_CMPR2_TEB_INT。(WT)

MCPWM_TZ0_CBC_INT_CLR 写 1 清除 MCPWM_TZ0_CBC_INT。(WT)

MCPWM_TZ1_CBC_INT_CLR 写 1 清除 MCPWM_TZ1_CBC_INT。(WT)

MCPWM_TZ2_CBC_INT_CLR 写 1 清除 MCPWM_TZ2_CBC_INT。(WT)

MCPWM_TZ0_OST_INT_CLR 写 1 清除 MCPWM_TZ0_OST_INT。(WT)

MCPWM_TZ1_OST_INT_CLR 写 1 清除 MCPWM_TZ1_OST_INT。(WT)

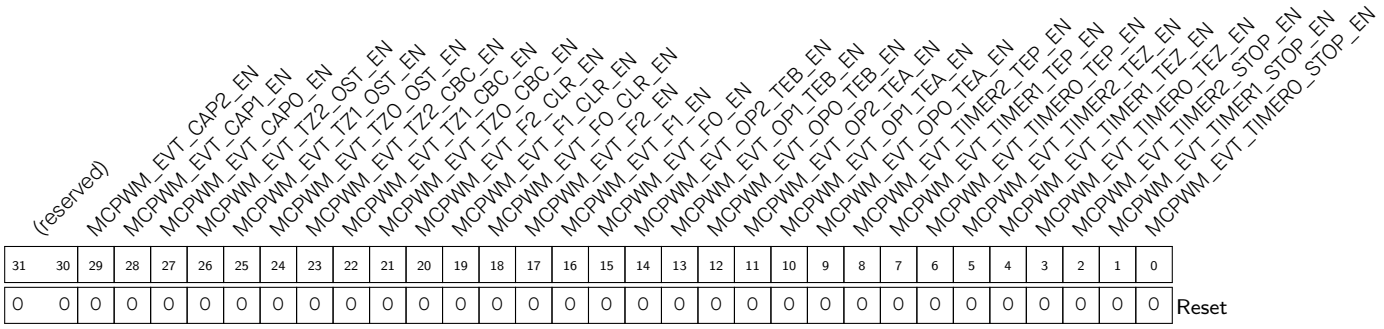
MCPWM_TZ2_OST_INT_CLR 写 1 清除 MCPWM_TZ2_OST_INT。(WT)

MCPWM_CAP0_INT_CLR 写 1 清除 MCPWM_CAP0_INT。(WT)

MCPWM_CAP1_INT_CLR 写 1 清除 MCPWM_CAP1_INT。(WT)

MCPWM_CAP2_INT_CLR 写 1 清除 MCPWM_CAP2_INT。(WT)

Register 41.33. MCPWM_EVT_EN_REG (0x0120)



MCPWM_EVT_TIMER0_STOP_EN 配置是否使能当定时器 0 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_STOP_EN 配置是否使能当定时器 1 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER2_STOP_EN 配置是否使能当定时器 2 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER0_TEZ_EN 配置是否使能当定时器 0 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_TEZ_EN 配置是否使能当定时器 1 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER2_TEZ_EN 配置是否使能当定时器 2 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER0_TEP_EN 配置是否使能当定时器 0 等于周期时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_TEP_EN 配置是否使能当定时器 1 等于周期时的事件生成。

0：关闭

1：使能

(R/W)

见下页...

Register 41.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TIMER2_TEP_EN 配置是否使能当定时器 2 等于周期时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP0_TEA_EN 配置是否使能当生成器 0 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP1_TEA_EN 配置是否使能当生成器 1 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP2_TEA_EN 配置是否使能当生成器 2 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP0_TEB_EN 配置是否使能当生成器 0 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP1_TEB_EN 配置是否使能当生成器 1 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP2_TEB_EN 配置是否使能当生成器 2 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F0_EN 配置是否使能 FAULT0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F1_EN 配置是否使能 FAULT1 事件生成。

0：关闭
1：使能
(R/W)

见下页...

Register 41.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_F2_EN 配置是否使能 FAULT2 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F0_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F1_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F2_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ0_CBC_EN 配置是否使能 CBC0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ1_CBC_EN 配置是否使能 CBC1 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ2_CBC_EN 配置是否使能 CBC2 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ0_OST_EN 配置是否使能 OST0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ1_OST_EN 配置是否使能 OST1 事件生成。

0：关闭
1：使能
(R/W)

见下页...

Register 41.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TZ2_OST_EN 配置是否使能 OST2 事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_CAP0_EN 配置是否使能 capture0 事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_CAP1_EN 配置是否使能 capture1 事件生成。

0：关闭

1：使能

(R/W)

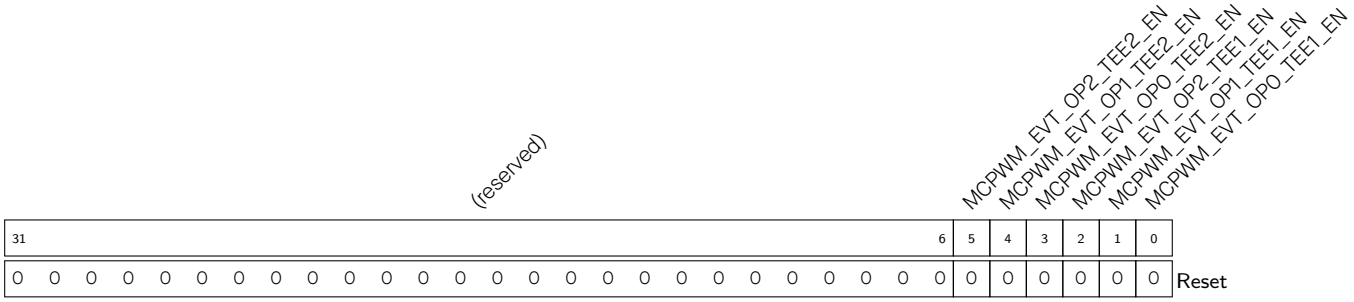
MCPWM_EVT_CAP2_EN 配置是否使能 capture2 事件生成。

0：关闭

1：使能

(R/W)

Register 41.34. MCPWM_EVT_EN2_REG (0x0128)



MCPWM_EVT_OP*n*_TEE1_EN 配置是否在 PWM 生成器计数器的值等于 MCPWM_OP*n*_TSTMP_E1_REG 寄存器的值时，生成 MCPWM_EVT_OP*n*_TEE1 事件。

0：不生成

1：生成

(R/W)

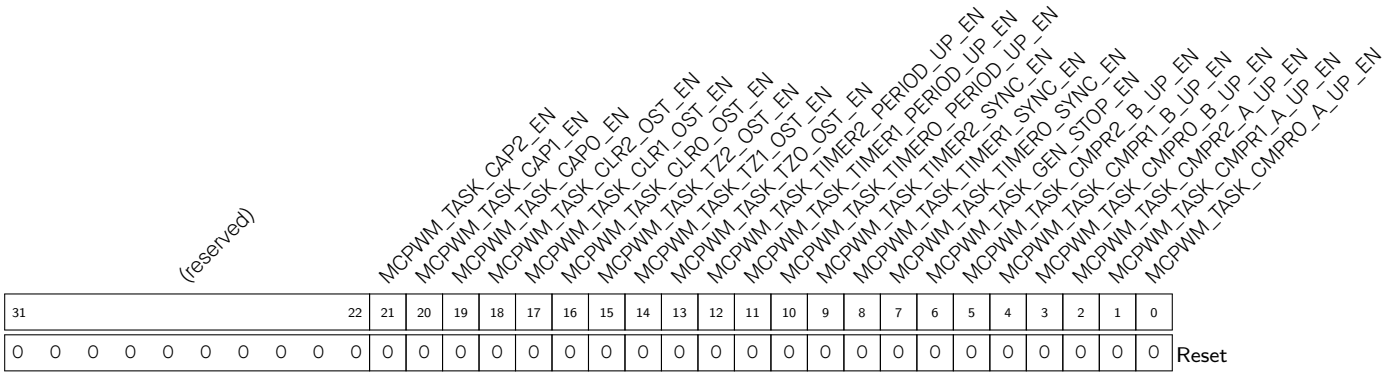
MCPWM_EVT_OP*n*_TEE2_EN 配置是否在 PWM 生成器计数器的值等于 MCPWM_OP*n*_TSTMP_E2_REG 寄存器的值时，生成 MCPWM_EVT_OP*n*_TEE2 事件。

0：不生成

1：生成

(R/W)

Register 41.35. MCPWM_TASK_EN_REG (0x0124)



MCPWM_TASK_CMPR0_A_UP_EN 配置是否接收 PWM 生成器 0 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR1_A_UP_EN 配置是否接收 PWM 生成器 1 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR2_A_UP_EN 配置是否接收 PWM 生成器 2 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR0_B_UP_EN 配置是否接收 PWM 生成器 0 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR1_B_UP_EN 配置是否接收 PWM 生成器 1 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR2_B_UP_EN 配置是否接收 PWM 生成器 2 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

见下页...

Register 41.35. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_GEN_STOP_EN 配置是否接收所有 PWM 生成器的停止任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER0_SYNC_EN 配置是否接收定时器 0 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER1_SYNC_EN 配置是否接收定时器 1 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER2_SYNC_EN 配置是否接收定时器 2 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER0_PERIOD_UP_EN 配置是否接收定时器 0 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER1_PERIOD_UP_EN 配置是否接收定时器 1 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER2_PERIOD_UP_EN 配置是否接收定时器 2 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TZ0_OST_EN 配置是否接收 OST0 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TZ1_OST_EN 配置是否接收 OST1 任务。

0：无效

1：接收

(R/W)

见下页...

Register 41.35. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_TZ2_OST_EN 配置是否接收 OST2 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR0_OST_EN 配置是否接收 OST0 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR1_OST_EN 配置是否接收 OST1 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR2_OST_EN 配置是否接收 OST2 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CAP0_EN 配置是否接收 capture0 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CAP1_EN 配置是否接收 capture1 任务。

0：无效

1：接收

(R/W)

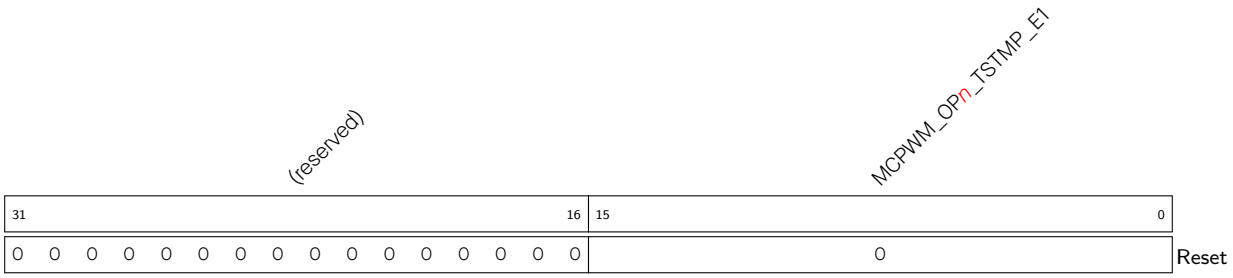
MCPWM_TASK_CAP2_EN 配置是否接收 capture2 任务。

0：无效

1：接收

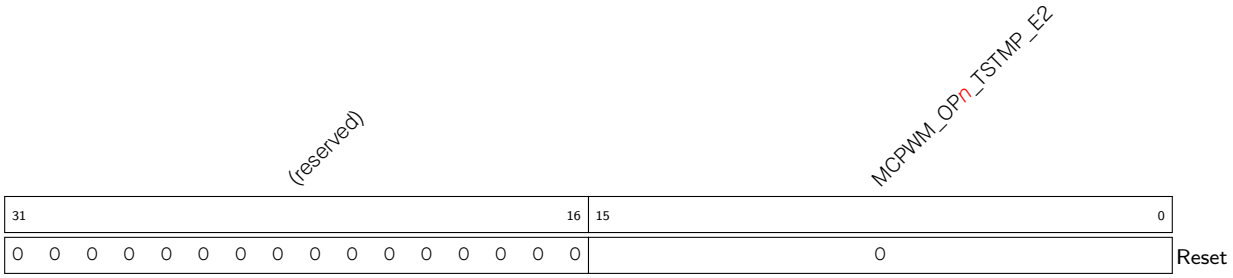
(R/W)

Register 41.36. MCPWM_OP*n*_TSTMP_E1_REG (*n*: 0-2) (0x012C+0x8**n*)



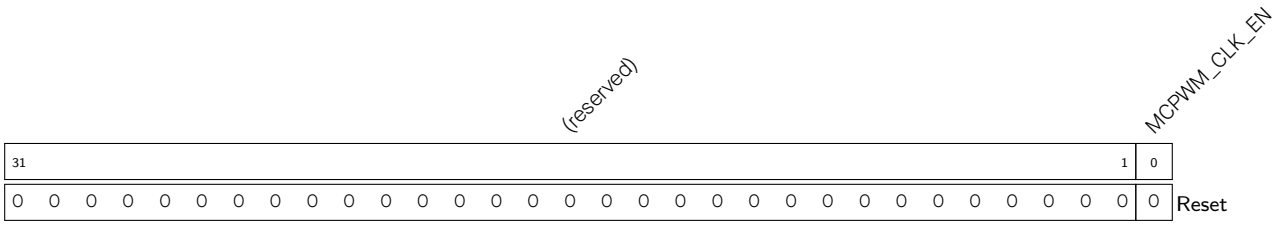
MCPWM_OP*n*_TSTMP_E1 配置生成器*n* 时间戳 E1 值。(R/W)

Register 41.37. MCPWM_OP*n*_TSTMP_E2_REG(*n*: 0-2) (0x0130+0x8**n*)



MCPWM_OP*n*_TSTMP_E2 配置生成器*n* 时间戳 E2 值。(R/W)

Register 41.38. MCPWM_CLK_REG (0x0144)



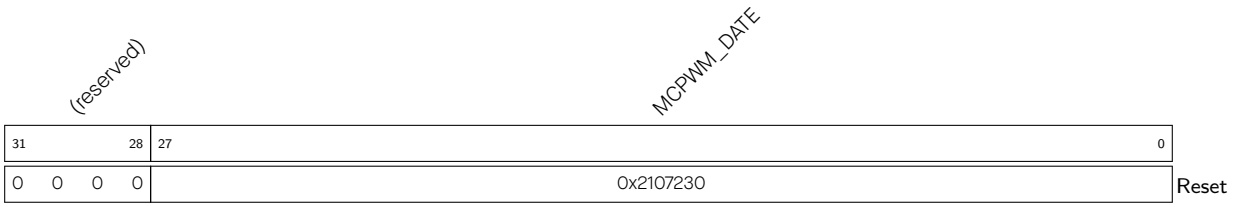
MCPWM_CLK_EN 配置是否强制使能时钟。

0：无效

1：强制使能时钟

(R/W)

Register 41.39. MCPWM_VERSION_REG (0x0148)



MCPWM_DATE 版本控制寄存器。(R/W)

第 42 章

红外遥控 (RMT)

42.1 概述

红外遥控 (RMT) 是一个红外发送和接收控制器，可通过软件编解码多种红外协议。RMT 模块可以将模块内置 RAM 中的脉冲编码转换为信号输出，或将模块的输入信号转换为脉冲编码存入 RAM 中。此外，RMT 模块也支持对输出信号进行载波调制或对输入信号进行解调和去噪处理。

RMT 共有四个通道，编码为 0 ~ 3，各通道可独立用于发送或接收信号：

- 0 ~ 1 通道专门用于信号发送；
- 2 ~ 3 通道专门用于信号接收。

每个发送通道和接收通道分别有一组功能相同的寄存器。为了方便叙述，以 *n* 表示各个发送通道，以 *m* 表示各个接收通道。

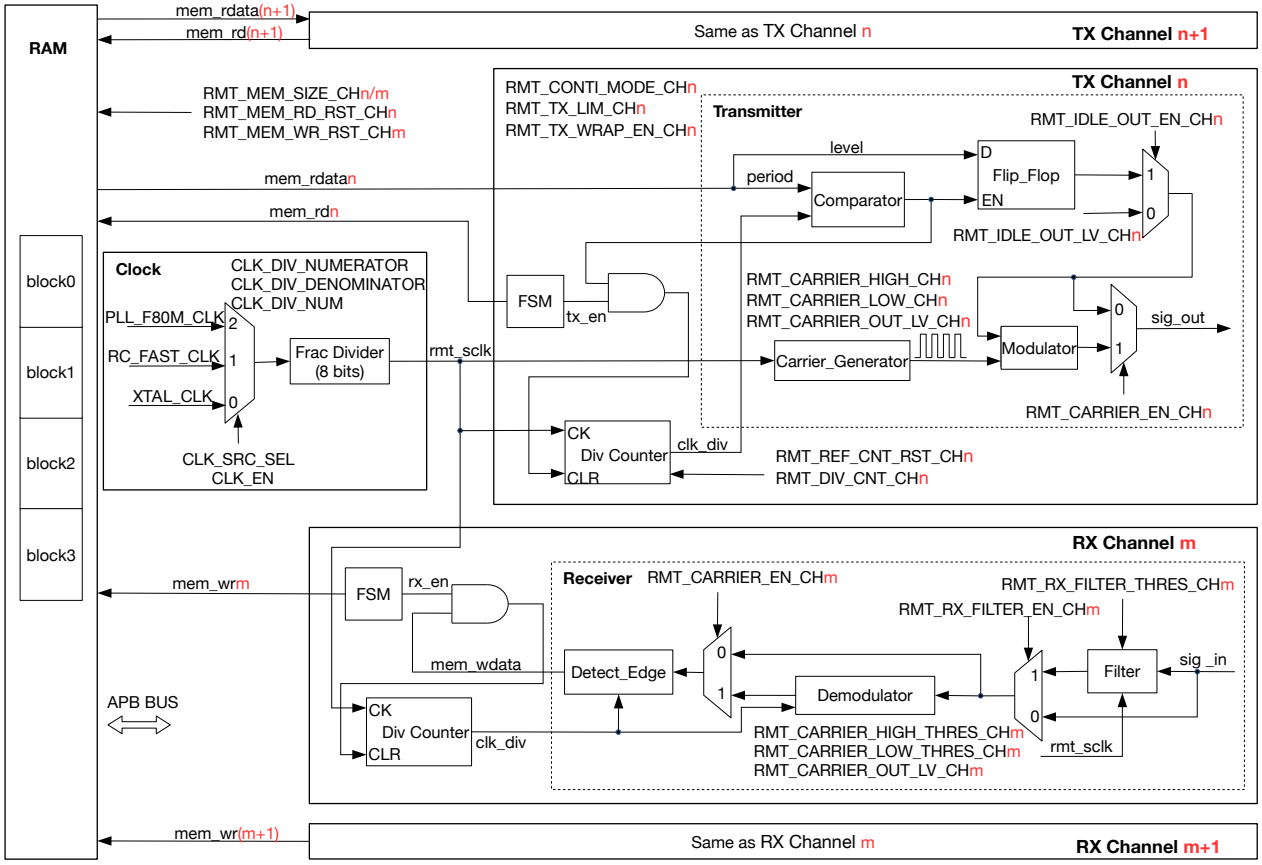
42.2 主要特性

RMT 控制器具有如下特性：

- 共配置四个通道：
  - 0 ~ 1 通道支持发送
  - 2 ~ 3 通道支持接收
  - 四个通道共享 192 x 32 位的 RAM
- 发射器支持以下模式：
  - 普通发送模式
  - 乒乓发送模式
  - 载波调制
  - 持续发送模式
  - 多通道同时发送
- 接收器支持以下模式：
  - 普通接收模式
  - 乒乓接收模式
  - 接收滤波
  - 载波解调

42.3 功能描述

42.3.1 RMT 架构



注：上图中，n = 0，m = 2。

图 42.3-1. RMT 结构框图

如图 42.3-1 所示，每个发送通道内部各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个发射器 (Transmitter)

每个接收通道内部也各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个接收器 (Receiver)

四个通道共享一块 192 x 32 位的 RAM。

图中 “Clock” 部分需要配置的 RMT 时钟相关寄存器具体如下：

- CLK_EN：PCR_RMT_SCLK_EN，用于使能 rmt_sclk 时钟

- CLK_SRC_SEL：PCR_RMT_SCLK_SEL，用于选择 rmt_sclk 时钟
- CLK_DIV_NUM：PCR_RMT_SCLK_DIV_NUM，用于配置 rmt_sclk 时钟的分频系数整数部分
- CLK_DIV_NUMERATOR：PCR_RMT_SCLK_DIV_A，用于配置 rmt_sclk 时钟分频系数小数部分的分子
- CLK_DIV_DENOMINATOR：PCR_RMT_SCLK_DIV_B，用于配置 rmt_sclk 时钟分频系数小数部分的分母

42.3.2 RAM

42.3.2.1 RAM 结构

RAM 中脉冲编码结构如图 42.3-2 所示。每个脉冲编码为 16 位，由 level 与 period 两部分组成。其中 level 表示输入或输出信号的逻辑电平值（0 或 1），period 表示该电平信号持续的时钟（见图 42.3-1 中的 clk_div）周期数。

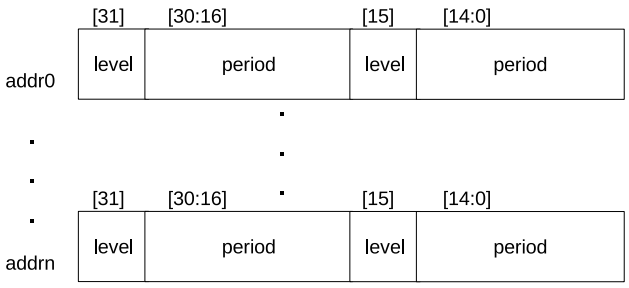


图 42.3-2. RAM 中脉冲编码结构

period 的最小值为 0，period 为 0 是一次传输的结束标志。对于非 0 的 period，即非结束标志的 period，其值需要满足与 APB 时钟和 RMT 时钟相关的限制条件，具体请参考以下公式：

$$5 \times T_{apb\_clk} + 6 \times T_{rmt\_sclk} < period \times T_{clk\_div} \tag{42.1}$$

说明：

RMT 时钟与 APB 时钟需满足以下关系：

$$1.5 \times T_{apb\_clk} < 9 \times T_{rmt\_sclk} \tag{42.2}$$

42.3.2.2 RAM 使用说明

RAM 按照 48 x 32 位分成四个 block。默认情况下，每个通道只能使用一个 block（固定为通道 0 使用 block 0，通道 1 使用 block 1，以此类推）。

当发送通道 *n* 或接收通道 *m* 单次传输的脉冲编码数大于一个 block 时，可以：

- 置位 RMT_MEM_TX/RX_WRAP_EN_CH*n*/*m* 使能乒乓操作；
- 或通过配置 RMT_MEM_SIZE_CH*n*/*m* 寄存器，允许该通道占用多个 block。

当设置 RMT_MEM_SIZE_CH*n*/*m* > 1 时，通道 *n*/*m* 将占用 block (*n*/*m*) ~ block (*n*/*m*+RMT_MEM_SIZE_CH*n*/*m*–1) 的存储空间，此时通道 *n*/*m*+1 ~ *n*/*m*+RMT_MEM_SIZE_CH*n*/*m*–1 因为对应的 RAM block 被占用而无法使用。例如，如果通道 0 配置使用了 block 0 和 block 1，则通道 1 无法使用，通道 2 和通道 3 不受影响。

注意，每个通道使用 RAM 的空间是根据地址从低到高进行映射的，因此通道 0 可以通过配置 RMT_MEM_SIZE_CH0 寄存器来使用通道 1、2、3 的 RAM 空间，但是通道 3 不能使用通道 0、1 或 2 的 RAM 空间。因此 RMT_MEM_SIZE_CH*n* 的最大值不应超过 (4–*n*)，RMT_MEM_SIZE_CH*m* 的最大值不应超过 (2–*m*)。

RAM 可被 APB 总线及通道的发射器或接收器访问。为了防止接收通道访问 RAM 和 APB 访问时发生冲突（例如 APB 读 RAM 时，通道向 RAM 发起写操作），可以通过配置 RMT_MEM_OWNER_CH*m* 来决定当前 RAM 的使用权。当接收通道发生越权访问时会产生 RMT_MEM_OWNER_ERR_CH*m* 标志信号。

42.3.2.3 RAM 访问方式

APB 总线访问 RAM 有 FIFO 和 NONFIFO（直接地址）两种模式：

- RMT_APB_FIFO_MASK 置 0 时，选择 FIFO 模式；
- RMT_APB_FIFO_MASK 置 1 时，选择 NONFIFO 模式。

FIFO 模式

在 FIFO 模式下，APB 通过固定地址 RMT_CH*n*/*m*DATA_REG 向 RAM 写数据或从 RAM 读数据。

NONFIFO 模式

在 NONFIFO 模式下，APB 向连续地址段写入数据，或从连续地址段读出数据：

- 发送通道 *n* 对应的写地址段的首地址是 (RMT 基地址 + 0x400 + *n*x48)，第 2 个数据的访问地址是（RMT 基地址 + 0x400 + *n*x48 + 0x4），以此类推，后面的地址依次加上 0x4。
- 接收通道 *m* 对应的读地址段的首地址是 (RMT 基地址 + 0x460 + *m*x48)，第 2 个数据的访问地址是（RMT 基地址 + 0x460 + *m*x48 + 0x4），以此类推，后面的地址依次加上 0x4。

42.3.3 时钟

用户可以通过配置 PCR_RMT_SCLK_SEL 选择 RMT 的时钟源：PLL_F80M_CLK、RC_FAST_CLK 或 XTAL_CLK，配置 PCR_RMT_SCLK_EN 为高电平来打开 RMT 的时钟。选择后的时钟经过小数分频得到 RMT 的工作时钟（见图 42.3-1 中的 rmt_sclk），分频系数为 PCR_RMT_SCLK_DIV_NUM + 1 + PCR_RMT_SCLK_DIV_A / PCR_RMT_SCLK_DIV_B。

更多信息，请参考章节 9 复位和时钟。RMT_DIV_CNT_CH*n*/*m* 用于配置 RMT 通道内部的时钟分频器的分频系数，除 0 表示 256 分频外，其他分频数等同于寄存器 RMT_DIV_CNT_CH*n*/*m* 的值。时钟分频器可以通过配置 RMT_REF_CNT_RST_CH*n*/*m* 进行复位。时钟分频器的分频时钟可供计数器使用，见图 42.3-1。

42.3.4 发射器

> **说明：**
> 本小节以及后续小节所述的配置，均需要通过置位 RMT_CONF_UPDATE_CH*n*/*m* 的方式来进行更新，详情见第 42.4 小节。

42.3.4.1 普通发送模式

当 RMT_TX_START_CH*n* 置为 1 时，通道 *n* 的发射器开始从通道对应 RAM block 的起始地址，按照地址从低到高依次读取脉冲编码进行发送。当遇到结束标志（period 等于 0）时，发射器将结束发送返回空闲状态，并产生 RMT_CH*n*_TX_END_INT 中断。配置 RMT_TX_STOP_CH*n* 可以使发射器立刻停止发送并进入空闲状态。发

射器空闲状态发送的电平由结束标志中的 level 段或者是 RMT_IDLE_OUT_LV_CH*n* 决定。用户可以配置 RMT_IDLE_OUT_EN_CH*n* 来选择这两种方式：

- RMT_IDLE_OUT_EN_CH*n* 置 0 时，发射器空闲状态发送的电平由结束标志中的 level 段决定；
- RMT_IDLE_OUT_EN_CH*n* 置 1 时，发射器空闲状态发送的电平由 RMT_IDLE_OUT_LV_CH*n* 决定。

42.3.4.2 乒乓发送模式

当发送的脉冲编码较多时，可通过置位 RMT_MEM_TX_WRAP_EN_CH*n* 使能通道 *n* 的乒乓模式。在乒乓操作模式下，发射器会循环从通道对应的 RAM 区域取出脉冲编码进行发送，直至遇到结束标识为止。例如，当 RMT_MEM_SIZE_CH*n* = 1 时，发射器将从 48 * *n* 地址开始发送，然后对应 RAM 的地址递增。发完 (48 * (*n*+1)–1) 地址的数据后，下次继续从 48 * *n* 地址开始递增发送数据，依此类推，遇到结束标识时停止发送。RMT_MEM_SIZE_CH*n* > 1 的情形下，乒乓操作同样适用。

每当发射器发送的脉冲编码数大于等于 RMT_TX_LIM_CH*n* 时，会产生 RMT_CH*n*_TX_THR_EVENT_INT 中断。在乒乓模式下，可以设置 RMT_TX_LIM_CH*n* 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 RMT_CH*n*_TX_THR_EVENT_INT 中断之后，可以回收已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

42.3.4.3 发送加载波

此外，发射器还可以对输出信号进行载波调制，置位 RMT_CARRIER_EN_CH*n* 可以使能该功能。载波的波形可配置。一个载波周期中高电平持续时间为 (RMT_CARRIER_HIGH_CH*n* + 1) 个 rmt_sclk 时钟周期，低电平持续的时间为 (RMT_CARRIER_LOW_CH*n* + 1) 个 rmt_sclk 时钟周期。置位 RMT_CARRIER_OUT_LV_CH*n* 时在输出信号高电平上加载波信号，清零 RMT_CARRIER_OUT_LV_CH*n* 时在输出信号低电平上加载波信号。同时，在进行载波调制时，载波可以一直加载在输出信号上，也可以仅加载在有效的脉冲编码（RAM 中的数据）上。通过配置 RMT_CARRIER_EFF_EN_CH*n* 寄存器，可以选择这两种模式：

- RMT_CARRIER_EFF_EN_CH*n* 置 0 时，在所有信号上加载波。
- RMT_CARRIER_EFF_EN_CH*n* 置 1 时，在有效信号上加载波。

42.3.4.4 持续发送模式

置位 RMT_TX_CONTI_MODE_CH*n* 可以使能发射器的持续发送功能。置位后，发射器会循环发送 RAM 中的脉冲编码。持续发送模式下：

- 如果遇到结束标志，则重新从该通道 RAM 中的第一个数据开始发送；
- 如果发送到 RAM 中的最后一个数都没有结束标志，存在两种情况。普通发送模式下（RMT_MEM_TX_WRAP_EN_CH*n* = 0），会因为 RAM 被读空没有可以继续发送的数而产生错误中断；乒乓发送模式下（RMT_MEM_TX_WRAP_EN_CH*n* = 1），会在发送到最后一个数据处回卷，重新开始发送第一个数据。

配置 RMT_TX_LOOP_CNT_EN_CH*n* 后，发射器每遇到一次结束标志，循环发送的次数会加 1。当该次数达到 RMT_TX_LOOP_NUM_CH*n* 设定的值时，会产生 RMT_CH*n*_TX_LOOP_INT 中断。如果置位 RMT_LOOP_STOP_EN_CH*n*，则发送会在产生 RMT_CH*n*_TX_LOOP_INT 中断后立即停止，否则会继续发送。持续发送模式下，如果遇到的结束标志类型是 period[14:0] 为 0，那么这个结束标志前一个数据的 period 需要满足：

$$10 \times T_{apb\_clk} + 19 \times T_{rmt\_sclk} < period \times T_{clk\_div} \tag{42.3}$$

其他数据的 period，满足 非 0 period 关系式 即可。

42.3.4.5 多通道同时发送

RMT 模块支持多通道同时发送，具体配置步骤如下所示：

1. 首先配置 RMT_TX_SIM_CH*n*，选择同步发送的通道；
2. 然后置位 RMT_TX_SIM_EN，使能发射器多个通道同步发送的功能；
3. 最后将所选同步发送通道的 RMT_TX_START_CH*n* 置为 1。

最后一个通道完成配置时，多个通道会同时启动发送。由于硬件条件的限制，两条通道不能保证完全同时开始发送，两条通道发送的时间间隔在 3 x T_{clk_div} 以内。

42.3.5 接收器

42.3.5.1 普通接收模式

RMT_RX_EN_CH*m* 置为 1 时接收器开始工作，置为 0 时会停止接收。接收器会从信号的第一个跳变沿开始计数，并检测信号电平及其持续的时钟周期数，将其按照脉冲编码的格式存入 RAM 中。当信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时，接收器结束接收过程，返回空闲状态，并产生 RMT_CH*m*_RX_END_INT 中断。RMT_IDLE_THRES_CH*m* 的值需要大于输入信号的高电平或低电平的最大时钟周期数，否则接收器会将该信号误判为空闲信号而进入空闲状态。当接收数据存满了接收通道设置的 RAM 空间时，会停止接收，在非乒乓模式下会产生 RMT_CH*m*_ERR_INT 中断（由 RAM 满事件触发）。

42.3.5.2 乒乓接收模式

当接收的脉冲编码较多时，可通过置位 RMT_MEM_RX_WRAP_EN_CH*m* 使能通道 *m* 的乒乓模式。在乒乓操作模式下，接收器会将接收到的脉冲编码循环存入通道对应的 RAM 区域，当信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时，接收器结束接收过程，返回空闲状态，并产生 RMT_CH*m*_RX_END_INT 中断。例如，当 RMT_MEM_SIZE_CH*m* = 1 时，接收器将从 48 * *m* 地址开始接收，然后对应 RAM 的地址递增。收完 (48 x (*m*+1)−1) 地址的数据后，下次继续从 48 * *m* 地址开始递增接收数据，以此类推，遇到信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时停止接收。RMT_MEM_SIZE_CH*m* > 1 的情形下，乒乓操作同样适用。

每当接收器接收的脉冲编码数大于或等于 RMT_RX_LIM_CH*m* 时，会产生 RMT_CH*m*_RX_THR_EVENT_INT 中断。在乒乓模式下，可以设置 RMT_RX_LIM_CH*m* 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 RMT_CH*m*_RX_THR_EVENT_INT 中断之后，可以回收已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

42.3.5.3 接收滤波

置位 RMT_RX_FILTER_EN_CH*m* 可使能通道 *m* 的接收器对输入信号进行滤波。滤波器的功能为连续采样输入信号，如果输入信号在连续 RMT_RX_FILTER _THRES_CH*m* 个 rmt_sclk 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，滤波器会滤除脉冲宽度小于 RMT_RX_FILTER_THRES_CH*m* 个 rmt_sclk 时钟周期的线路毛刺。

42.3.5.4 接收去载波

此外，接收器还可以对输入信号或滤波后的信号进行去载波调制，置位 RMT_CARRIER_EN_CH*m* 可以使能该功能。去载波可以对高电平调制或低电平调制的载波进行滤除：

- RMT_CARRIER_OUT_LV_CH*m* 置 0 时，配置滤除低电平载波；
- RMT_CARRIER_OUT_LV_CH*m* 置 1 时，配置滤除高电平载波。

配置 RMT_CARRIER_HIGH_THRES_CH*m* 和 RMT_CARRIER_LOW_THRES_CH*m* 可设置去载波的高电平阈值和低电平阈值。当信号的高电平持续时间小于 RMT_CARRIER_HIGH_THRES_CH*m* 个 clk_div 分频时钟周期，或者信号的低电平持续时间小于 RMT_CARRIER _LOW_THRES_CH*m* 个 clk_div 分频时钟周期，则信号会被认为是载波而被滤除。

42.4 配置参数更新

RMT 的配置寄存器均需要配置各自通道的 RMT_CONF_UPDATE_CH*n*/*m* 位来更新进入各自通道，更新方法是向 RMT_CONF_UPDATE_CH*n*/*m* 写入高电平。发送通道和接收通道需要通过这种方法更新的配置参数见表 42.4-1。

表 42.4-1. 更新配置参数

配置寄存器	配置参数
发送通道	
RMT_CH*n*CONF0_REG	RMT_CARRIER_OUT_LV_CH*n*
	RMT_CARRIER_EN_CH*n*
	RMT_CARRIER_EFF_EN_CH*n*
	RMT_DIV_CNT_CH*n*
	RMT_IDLE_OUT_EN_CH*n*
	RMT_IDLE_OUT_LV_CH*n*
	RMT_TX_CONTI_MODE_CH*n*
RMT_CH*n*CARRIER_DUTY_REG	RMT_CARRIER_HIGH_CH*n*
	RMT_CARRIER_LOW_CH*n*
RMT_CH*n*_TX_LIM_REG	RMT_TX_LOOP_CNT_EN_CH*n*
	RMT_TX_LOOP_NUM_CH*n*
	RMT_TX_LIM_CH*n*
RMT_TX_SIM_REG	RMT_TX_SIM_EN
接收通道	
RMT_CH*m*CONF0_REG	RMT_CARRIER_OUT_LV_CH*m*
	RMT_CARRIER_EN_CH*m*
	RMT_IDLE_THRES_CH*m*
	RMT_DIV_CNT_CH*m*
RMT_CH*m*CONF1_REG	RMT_RX_FILTER_THRES_CH*m*
	RMT_RX_EN_CH*m*
RMT_CH*m*_RX_CARRIER_RM_REG	RMT_CARRIER_HIGH_THRES_CH*m*
	RMT_CARRIER_LOW_THRES_CH*m*
RMT_CH*m*_RX_LIM_REG	RMT_RX_LIM_CH*m*
RMT_REF_CNT_RST_REG	RMT_REF_CNT_RST_CH*m*

42.5 中断

ESP32-C5 的 RMT 控制器可以生成 RMT_INTR 中断信号并发送给 中断矩阵，该中断信号由 RMT 的内部中断源生成。

以下中断源生成 RMT_INTR 中断信号：

- RMT_CH*n*/*m*_ERR_INT：当通道 *n*/*m* 发生读写数据不正确，或内存空满错误时，即触发此中断。
- RMT_CH*n*_TX_THR_EVENT_INT：发射器每发送 RMT_CH*n*_TX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH*m*_RX_THR_EVENT_INT：接收器每接收 RMT_CH*m*_RX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH*n*_TX_END_INT：当发射器停止发送信号时，即触发此中断。
- RMT_CH*m*_RX_END_INT：当接收器停止接收信号时，即触发此中断。
- RMT_CH*n*_TX_LOOP_INT：发射器处于循环发送模式时，当循环次数达到 RMT_TX_LOOP_NUM_CH*n* 的值后，会产生此中断。

> **说明：**
> 中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 42.6 寄存器列表。

42.6 寄存器列表

本小节的所有地址均为相对于红外遥控基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 读/写寄存器			
RMT_CH0DATA_REG	通道 0 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0000	HRO
RMT_CH1DATA_REG	通道 1 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0004	HRO
RMT_CH2DATA_REG	通道 2 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0008	HRO
RMT_CH3DATA_REG	通道 3 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x000C	HRO
配置寄存器			
RMT_CH0CONF0_REG	通道 0 的配置寄存器 0	0x0010	varies
RMT_CH1CONF0_REG	通道 1 的配置寄存器 0	0x0014	varies
RMT_CH2CONF0_REG	通道 2 的配置寄存器 0	0x0018	R/W
RMT_CH2CONF1_REG	通道 2 的配置寄存器 1	0x001C	varies
RMT_CH3CONF0_REG	通道 3 的配置寄存器 0	0x0020	R/W
RMT_CH3CONF1_REG	通道 3 的配置寄存器 1	0x0024	varies
RMT_SYS_CONF_REG	RMT APB 配置寄存器	0x0068	R/W
RMT_REF_CNT_RST_REG	RMT 时钟分频器复位寄存器	0x0070	WT
状态寄存器			
RMT_CH0STATUS_REG	通道 0 的状态寄存器	0x0028	RO
RMT_CH1STATUS_REG	通道 1 的状态寄存器	0x002C	RO
RMT_CH2STATUS_REG	通道 2 的状态寄存器	0x0030	RO
RMT_CH3STATUS_REG	通道 3 的状态寄存器	0x0034	RO
中断寄存器			
RMT_INT_RAW_REG	原始中断状态寄存器	0x0038	R/ WTC/ SS
RMT_INT_ST_REG	屏蔽中断状态寄存器	0x003C	RO
RMT_INT_ENA_REG	中断使能寄存器	0x0040	R/W
RMT_INT_CLR_REG	中断清零寄存器	0x0044	WT
载波占空比寄存器			
RMT_CH0CARRIER_DUTY_REG	通道 0 的占空比配置寄存器	0x0048	R/W
RMT_CH1CARRIER_DUTY_REG	通道 1 的占空比配置寄存器	0x004C	R/W
RMT_CH2_RX_CARRIER_RM_REG	通道 2 的去载波寄存器	0x0050	R/W
RMT_CH3_RX_CARRIER_RM_REG	通道 3 的去载波寄存器	0x0054	R/W
TX 事件配置寄存器			
RMT_CH0_TX_LIM_REG	通道 0 的 TX 事件配置寄存器	0x0058	varies
RMT_CH1_TX_LIM_REG	通道 1 的 TX 事件配置寄存器	0x005C	varies

名称	描述	地址	访问
RMT_TX_SIM_REG	RMT TX 同步发送寄存器	0x006C	R/W
RX 事件配置寄存器			
RMT_CH2_RX_LIM_REG	通道 2 RX 事件配置寄存器	0x0060	R/W
RMT_CH3_RX_LIM_REG	通道 3 RX 事件配置寄存器	0x0064	R/W
版本寄存器			
RMT_DATE_REG	版本控制寄存器	0x00CC	R/W

42.7 寄存器

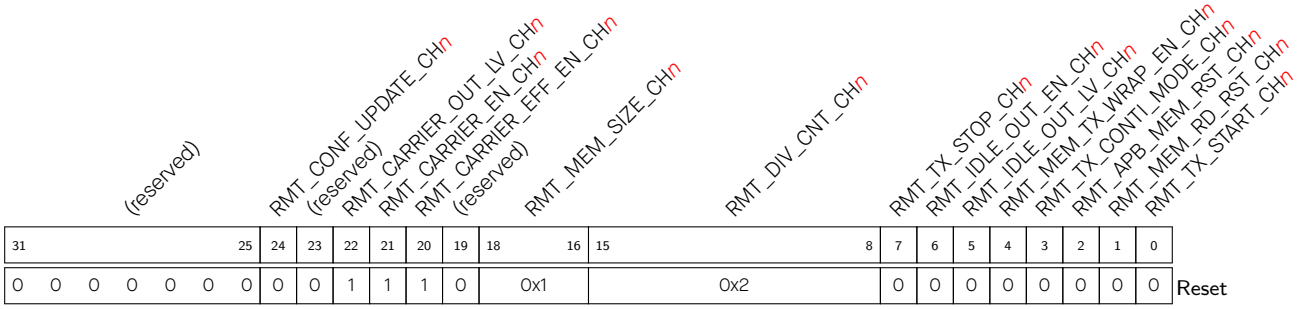
本小节的所有地址均为相对于红外遥控基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 42.1. RMT_CH*n*DATA_REG (*n*: 0-3) (0x0000+0x4**n*)



RMT_CH*n*DATA 通道 *n* 通过 APB FIFO 进行读写操作时用到的数据寄存器。(HRO)

Register 42.2. RMT_CH*n*CONF0_REG (*n*: 0-1) (0x0010+0x4**n*)



RMT_TX_START_CH*n* 配置是否使能通道 *n* 开始发送数据。

0：无效

1：使能

(WT)

RMT_MEM_RD_RST_CH*n* 配置是否复位通道 *n* 通过发射器访问的 RAM 读地址。

0：无效

1：复位

(WT)

RMT_APB_MEM_RST_CH*n* 配置是否复位通道 *n* 通过 APB FIFO 访问的读/写 RAM 地址。

0：无效

1：复位

(WT)

见下页...

Register 42.2. RMT_CH*n*CONF0_REG (*n*: 0-1) (0x0010+0x4**n*)

接上页...

RMT_TX_CONTI_MODE_CH*n* 配置是否使能通道 *n* 的持续发送模式。

0：无效

1：使能

在这种模式下，发射器从第一个数据开始发送，如果遇到结束标志，则再次发送第一个数据；如果没有遇到结束标志，发送完最后一个数据后，回卷到第一个数据再次继续发送。

(R/W)

RMT_MEM_TX_WRAP_EN_CH*n* 配置是否使能通道 *n* 的乒乓发送模式。

0：无效

1：使能

在这种模式下，如果待发送的数据长度大于该通道的 RAM Block 长度，则发射器将继续从第一个数据开始循环发送。

(R/W)

RMT_IDLE_OUT_LV_CH*n* 配置通道 *n* 在空闲模式下的输出信号电平。(R/W)

RMT_IDLE_OUT_EN_CH*n* 配置是否使能通道 *n* 在空闲状态下的输出模式。

0：无效

1：使能

(R/W)

RMT_TX_STOP_CH*n* 配置通道 *n* 的发射器是否停止发送数据。

0：无效

1：停止

(R/W/SC)

RMT_DIV_CNT_CH*n* 配置通道 *n* 的时钟分频系数。

测量单位：rmt_sclk

(R/W)

RMT_MEM_SIZE_CH*n* 配置通道 *n* 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EFF_EN_CH*n* 配置是否仅在发送数据状态下对输出信号载波调制。

0：配置通道 *n* 对发送数据状态和空闲状态均加载载波

1：配置通道 *n* 仅在发送数据状态下对输出信号载波调制

仅在 RMT_CARRIER_EN_CH*n* 为 1 时有效。

(R/W)

RMT_CARRIER_EN_CH*n* 配置是否使能通道 *n* 对输出信号进行载波调制。

0：不使能

1：使能

(R/W)

见下页...

Register 42.2. RMT_CH*n*CONF0_REG (*n*: 0-1) (0x0010+0x4**n*)

接上页...

RMT_CARRIER_OUT_LV_CH*n* 配置通道 *n* 的载波调制方式。

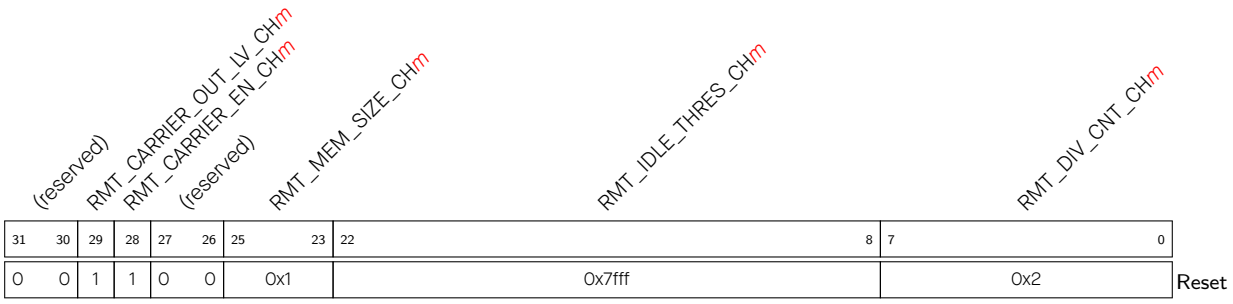
0：载波加载在低电平上

1：载波加载在高电平上

(R/W)

RMT_CONF_UPDATE_CH*n* 通道 *n* 的同步位。(WT)

Register 42.3. RMT_CH*m*CONF0_REG (*m*: 2-3) (0x0018+0x8*(*m*-2))



RMT_DIV_CNT_CH*m* 配置通道 *m* 的时钟分频时钟。

测量单位：rmt_sclk

(R/W)

RMT_IDLE_THRES_CH*m* 配置接收阈值。

接收器长时间检测不到信号变化，且持续的时间大于 RMT_IDLE_THRES_CH*m* 的值，则接收器停止接收过程。

测量单位：clk_div

(R/W)

RMT_MEM_SIZE_CH*m* 配置通道 *m* 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EN_CH*m* 配置是否使能通道 *m* 对输出信号进行载波调制和解调。

0：不使能

1：使能

(R/W)

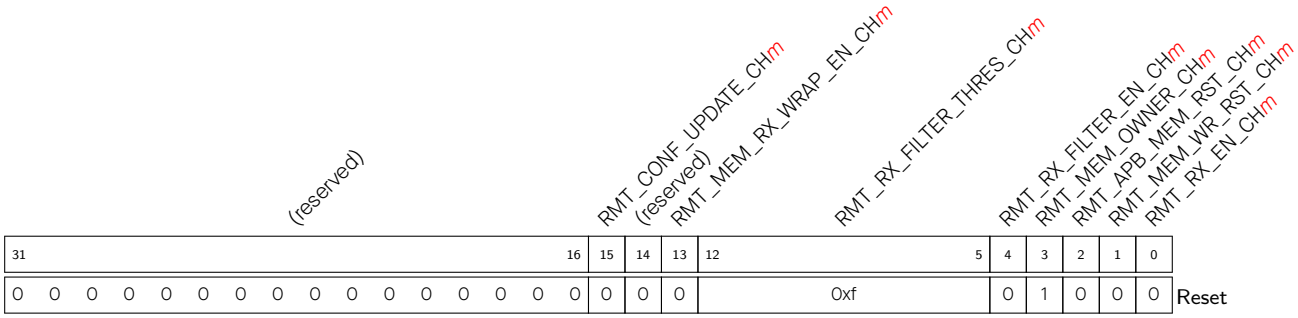
RMT_CARRIER_OUT_LV_CH*m* 配置通道 *m* 的载波调制方式。

0：载波加载在低电平上

1：载波加载在高电平上

(R/W)

Register 42.4. RMT_CH*m*CONF1_REG (*m*: 2-3) (0x001C+0x8*(*m*-2))



RMT_RX_EN_CH*m* 配置是否使能通道 *m* 的接收器，使接收器开始接收数据。

0：不使能

1：使能

(R/W)

RMT_MEM_WR_RST_CH*m* 配置是否复位通道 *m* 通过接收器访问的 RAM 写地址。

0：无效

1：复位

(WT)

RMT_APB_MEM_RST_CH*m* 配置是否复位通道 *m* 通过 APB FIFO 访问的读/写 RAM 地址。

0：无效

1：复位

(WT)

RMT_MEM_OWNER_CH*m* 配置通道 *m* 的 RAM 使用权。

0：APB 总线有权使用该通道的 RAM

1：接收器有权使用该通道的 RAM

(R/W/SC)

RMT_RX_FILTER_EN_CH*m* 配置是否使能通道 *m* 的接收器滤波功能。

0：不使能

1：使能

(R/W)

RMT_RX_FILTER_THRES_CH*m* 配置是否在接收模式下，忽略宽度小于 RMT_RX_FILTER_THRES_CH*m* 个 rmt_sclk 周期的输入脉冲。

0：无效

1：忽略

(R/W)

RMT_MEM_RX_WRAP_EN_CH*m* 配置是否使能通道 *m* 的乒乓接收模式。

0：不使能

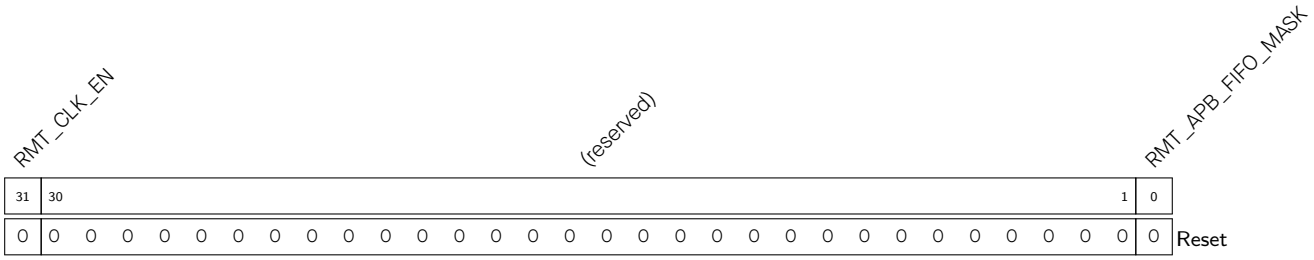
1：使能

在这种模式下，如果待接收的数据长度大于该通道的 RAM Block 长度，则接收器将继续把接收数据存入 RAM 的第一个地址，依次循环。

(R/W)

RMT_CONF_UPDATE_CH*m* 通道 *m* 的同步位。(WT)

Register 42.5. RMT_SYS_CONF_REG (0x0068)



RMT_APB_FIFO_MASK 配置 RAM 的访问模式。

0：通过 FIFO 访问 RAM

1：直接访问 RAM

(R/W)

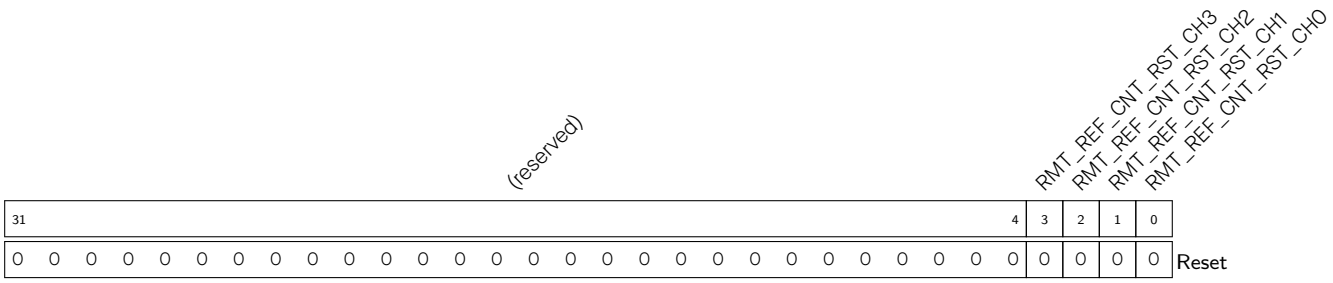
RMT_CLK_EN 配置是否使能 RMT 寄存器的时钟门控。

0：关闭寄存器的驱动时钟

1：打开寄存器的驱动时钟

(R/W)

Register 42.6. RMT_REF_CNT_RST_REG (0x0070)



RMT_REF_CNT_RST_CH*n* (*n*: 0-1) 配置是否复位通道 *n* 的时钟分频器。

0：无效

1：复位

(WT)

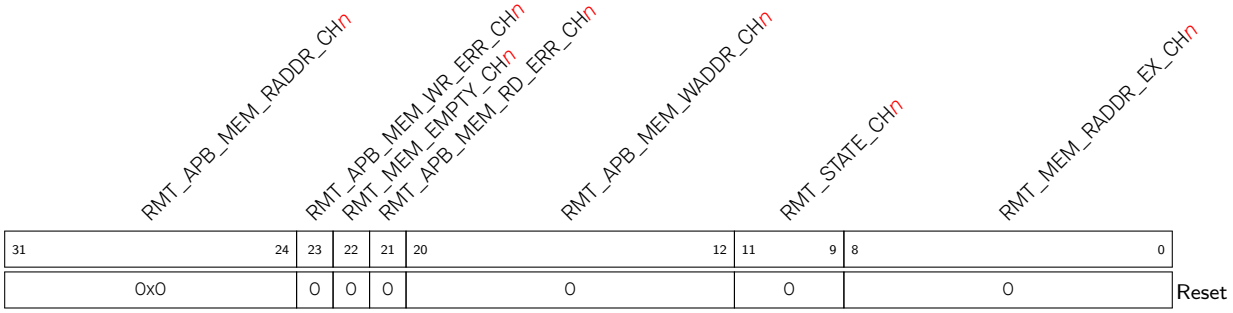
RMT_REF_CNT_RST_CH*m* (*m*: 2-3) 配置是否复位通道 *m* 的时钟分频器。

0：无效

1：复位

(WT)

Register 42.7. RMT_CH*n*STATUS_REG (*n*: 0-1) (0x0028+0x4**n*)



RMT_MEM_RADDR_EX_CH*n* 表示通道 *n* 发射器使用 RAM 时的地址偏移量。(RO)

RMT_STATE_CH*n* 表示通道 *n* 的 FSM 状态。(RO)

RMT_APB_MEM_WADDR_CH*n* 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)

RMT_APB_MEM_RD_ERR_CH*n* 表示 RMT 使用 APB 总线进行读 RAM 操作时，偏移地址是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

RMT_MEM_EMPTY_CH*n* 表示是否发送的数据长度是否溢出 RAM Block，且乒乓模式未启用。

0：未溢出

1：溢出

(RO)

RMT_APB_MEM_WR_ERR_CH*n* 表示 RMT 使用 APB 总线进行写 RAM 操作时，偏移地址是否溢出 RAM Block。

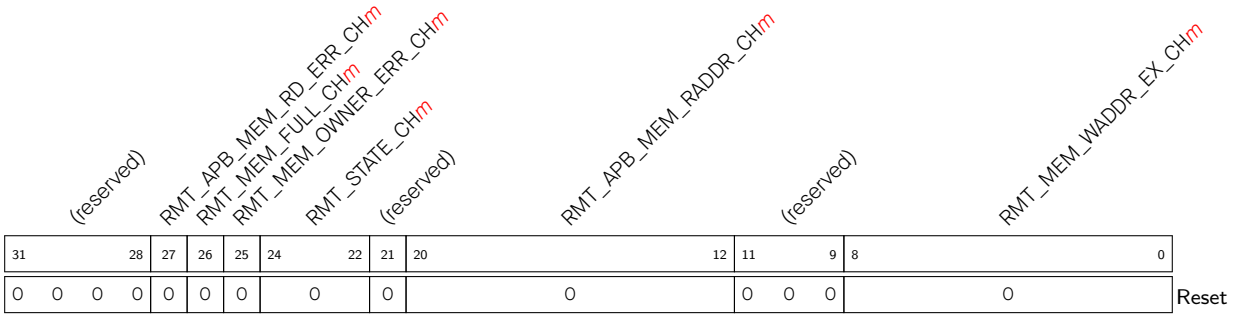
0：未溢出

1：溢出

(RO)

RMT_APB_MEM_RADDR_CH*n* 表示 RMT 使用 APB 总线读 RAM 时的地址偏移量。(RO)

Register 42.8. RMT_CH*m*STATUS_REG (*m*: 2-3) (0x0030+0x4*(*m*-2))



RMT_MEM_WADDR_EX_CH*m* 表示通道 *m* 接收器使用 RAM 时的地址偏移量。(RO)

RMT_APB_MEM_RADDR_CH*m* 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)

RMT_STATE_CH*m* 表示通道 *m* 的 FSM 状态。(RO)

RMT_MEM_OWNER_ERR_CH*m* 表示 RAM Block 使用权是否发生错误。

0：无错误

1：发生错误

(RO)

RMT_MEM_FULL_CH*m* 表示接收器接收的数据长度是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

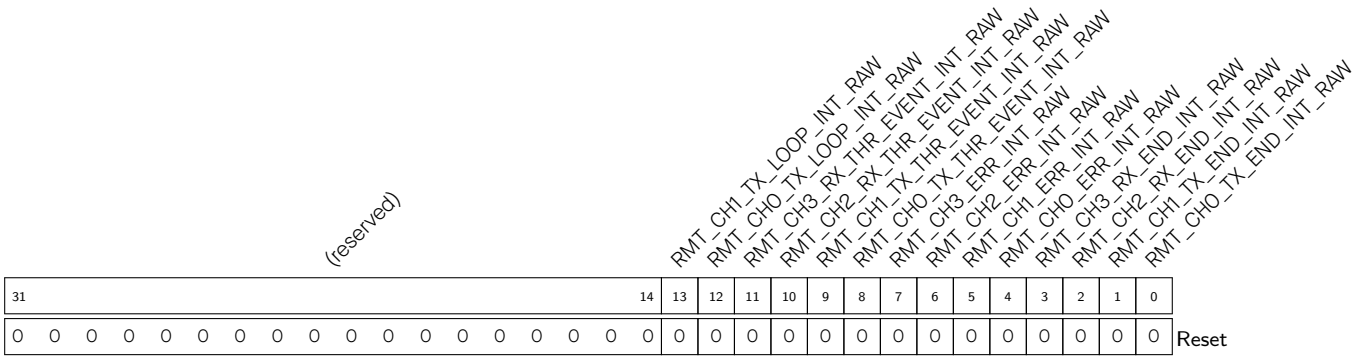
RMT_APB_MEM_RD_ERR_CH*m* 表示 RMT 使用 APB 总线执行 RAM 读操作时，偏移地址是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

Register 42.9. RMT_INT_RAW_REG (0x0038)



RMT_CH*n*_TX_END_INT_RAW (*n*: 0-1) RMT_CH*n*_TX_END_INT 的原始中断状态。发送完成时触发。(R/WTC/SS)

RMT_CH*m*_RX_END_INT_RAW (*m*: 2-3) RMT_CH*m*_RX_END_INT 的原始中断状态。接收完成时触发。(R/WTC/SS)

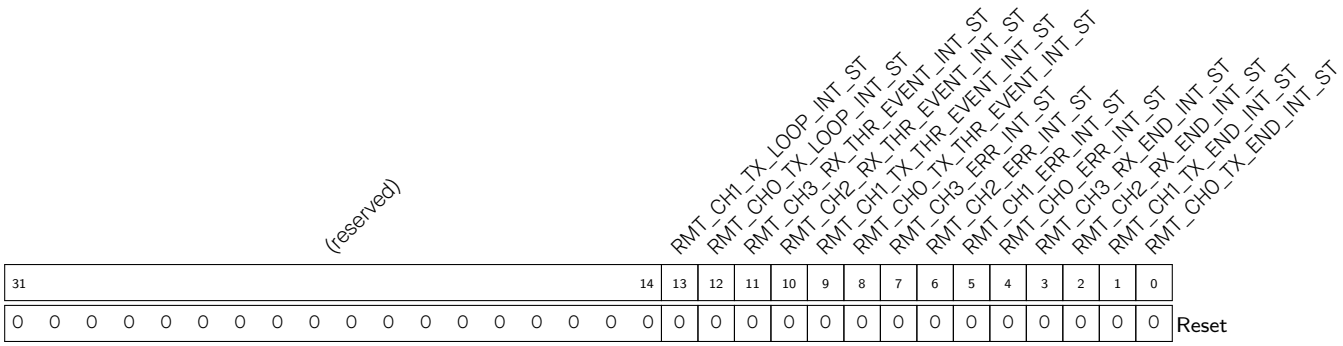
RMT_CH*n*_ERR_INT_RAW (*n*: 0-3) RMT_CH*n*/*m*_ERR_INT 的原始中断状态。发生错误时触发。(R/WTC/SS)

RMT_CH*n*_TX_THR_EVENT_INT_RAW (*n*: 0-1) RMT_CH*n*_TX_THR_EVENT_INT 的原始中断状态。发送器发送多于配置值时触发。(R/WTC/SS)

RMT_CH*m*_RX_THR_EVENT_INT_RAW (*m*: 2-3) RMT_CH*m*_RX_THR_EVENT_INT 的原始中断状态。接收器接收多于配置值时触发。(R/WTC/SS)

RMT_CH*n*_TX_LOOP_INT_RAW (*n*: 0-1) RMT_CH*n*_TX_LOOP_INT 的原始中断状态。循环计数次数到达配置阈值时触发。(R/WTC/SS)

Register 42.10. RMT_INT_ST_REG (0x003C)



RMT_CH*n*_TX_END_INT_ST (*n*: 0-1) RMT_CH*n*_TX_END_INT 的屏蔽中断状态。(RO)

RMT_CH*m*_RX_END_INT_ST (*m*: 2-3) RMT_CH*m*_RX_END_INT 的屏蔽中断状态。(RO)

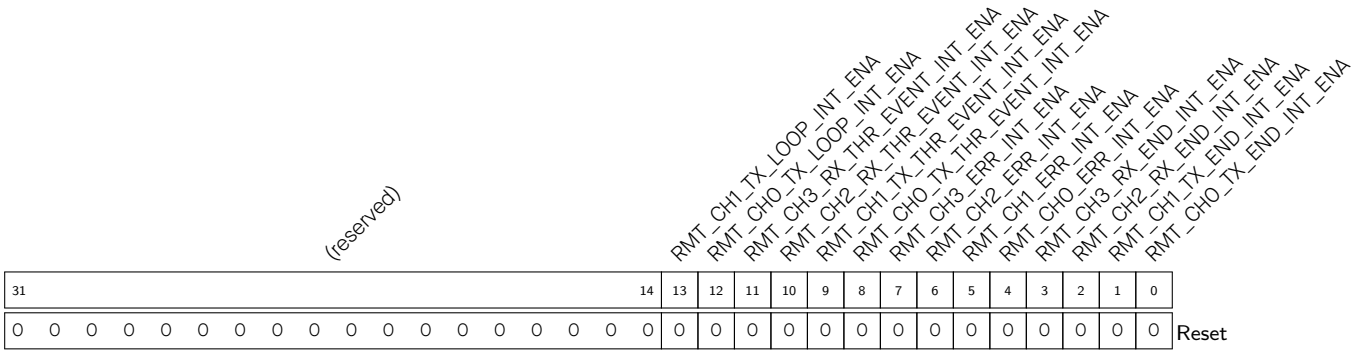
RMT_CH*n*_ERR_INT_ST (*n*: 0-3) RMT_CH*n*/*m*_ERR_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST (*n*: 0-1) RMT_CH*n*_TX_THR_EVENT_INT 的屏蔽中断状态。(RO)

RMT_CH*m*_RX_THR_EVENT_INT_ST (*m*: 2-3) RMT_CH*m*_RX_THR_EVENT_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_TX_LOOP_INT_ST (*n*: 0-1) RMT_CH*n*_TX_LOOP_INT 的屏蔽中断状态。(RO)

Register 42.11. RMT_INT_ENA_REG (0x0040)



RMT_CH*n*_TX_END_INT_ENA (*n*: 0-1) 写 1 使能 RMT_CH*n*_TX_END_INT。(R/W)

RMT_CH*m*_RX_END_INT_ENA (*m*: 2-3) 写 1 使能 RMT_CH*m*_RX_END_INT。(R/W)

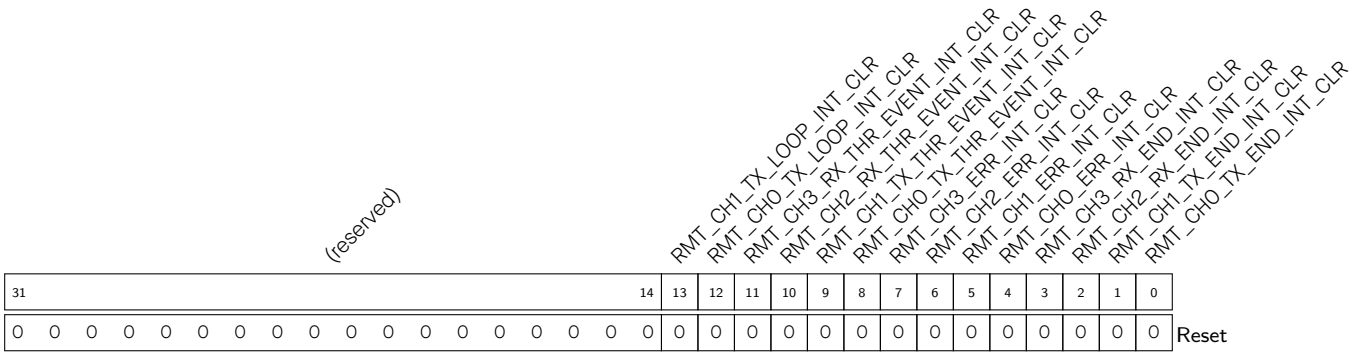
RMT_CH*n*_ERR_INT_ENA (*n*: 0-3) 写 1 使能 RMT_CH*n*/*m*_ERR_INT。(R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA (*n*: 0-1) 写 1 使能 RMT_CH*n*_TX_THR_EVENT_INT。(R/W)

RMT_CH*m*_RX_THR_EVENT_INT_ENA (*m*: 2-3) 写 1 使能 RMT_CH*m*_RX_THR_EVENT_INT。(R/W)

RMT_CH*n*_TX_LOOP_INT_ENA (*n*: 0-1) 写 1 使能 RMT_CH*n*_TX_LOOP_INT。(R/W)

Register 42.12. RMT_INT_CLR_REG (0x0044)



RMT_CH*n*_TX_END_INT_CLR (*n*: 0-1) 写 1 清除 RMT_CH*n*_TX_END_INT。(WT)

RMT_CH*m*_RX_END_INT_CLR (*m*: 2-3) 写 1 清除 RMT_CH*m*_RX_END_INT。(WT)

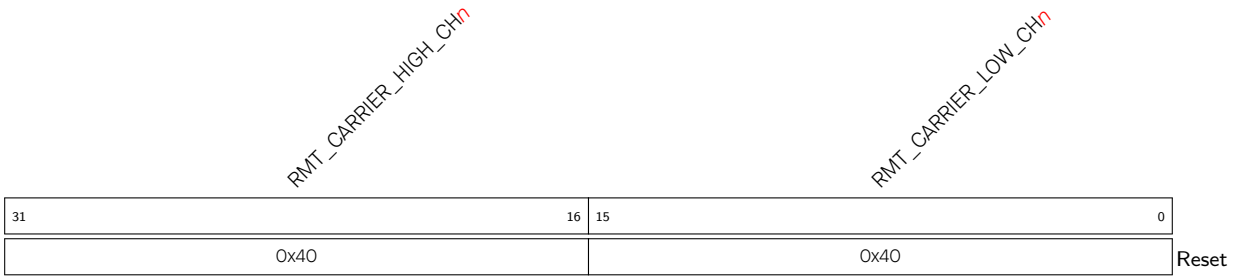
RMT_CH*n*_ERR_INT_CLR (*n*: 0-3) 写 1 清除 RMT_CH*n*/*m*_ERR_INT。(WT)

RMT_CH*n*_TX_THR_EVENT_INT_CLR (*n*: 0-1) 写 1 清除 RMT_CH*n*_TX_THR_EVENT_INT。(WT)

RMT_CH*m*_RX_THR_EVENT_INT_CLR (*m*: 2-3) 写 1 清除 RMT_CH*m*_RX_THR_EVENT_INT。(WT)

RMT_CH*n*_TX_LOOP_INT_CLR (*n*: 0-1) 写 1 清除 RMT_CH*n*_TX_LOOP_INT。(WT)

Register 42.13. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-1) (0x0048+0x4**n*)



RMT_CARRIER_LOW_CH*n* 配置通道 *n* 载波的低电平时钟周期。

测量单位：rmt_sclk

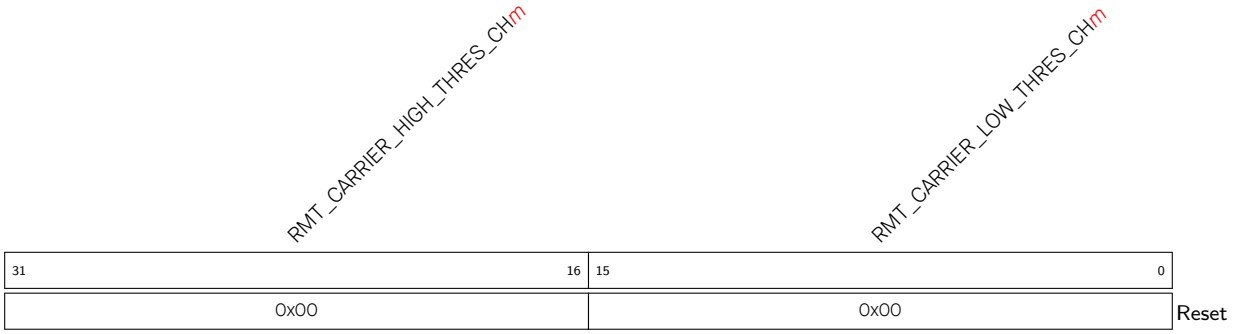
(R/W)

RMT_CARRIER_HIGH_CH*n* 配置通道 *n* 载波的高电平时钟周期。

测量单位：rmt_sclk

(R/W)

Register 42.14. RMT_CH*m*_RX_CARRIER_RM_REG (*m*: 2-3) (0x0050+0x4*(*m*-2))



RMT_CARRIER_LOW_THRES_CH*m* 配置载波调制模式下通道 *m* 的低电平周期。

载波调制模式下，通道 *m* 低电平周期为 RMT_CARRIER_LOW_THRES_CH*m* + 1。

测量单位：clk_div

(R/W)

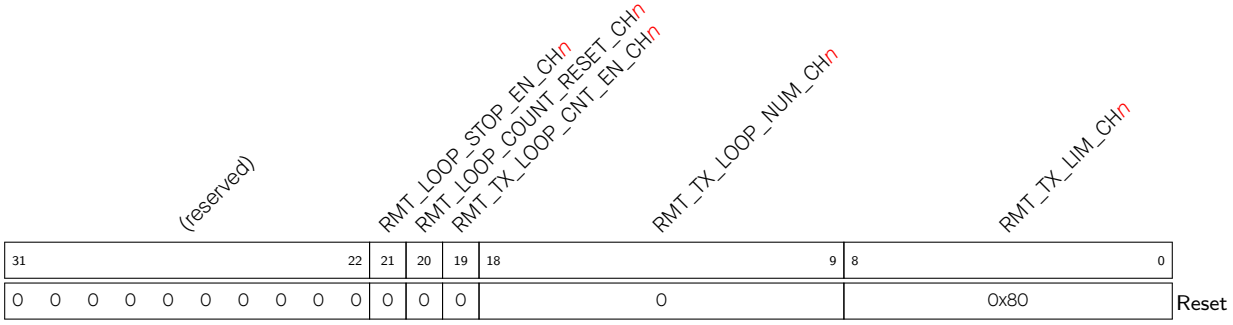
RMT_CARRIER_HIGH_THRES_CH*m* 配置载波调制模式下通道 *m* 的高电平周期。

载波调制模式下，通道 *m* 高电平周期为 RMT_CARRIER_HIGH_THRES_CH*m* + 1。

测量单位：clk_div

(R/W)

Register 42.15. RMT_CH*n*_TX_LIM_REG (*n*: 0-1) (0x0058+0x4**n*)



RMT_TX_LIM_CH*n* 配置通道 *n* 发送脉冲编码数量的上限值。(R/W)

RMT_TX_LOOP_NUM_CH*n* 配置持续发送模式下最大循环发送次数。(R/W)

RMT_TX_LOOP_CNT_EN_CH*n* 配置是否使能循环次数计数。

0：无效

1：使能

(R/W)

RMT_LOOP_COUNT_RESET_CH*n* 配置是否重置持续发送模式下的循环计数器。

0：无效

1：重置

(WT)

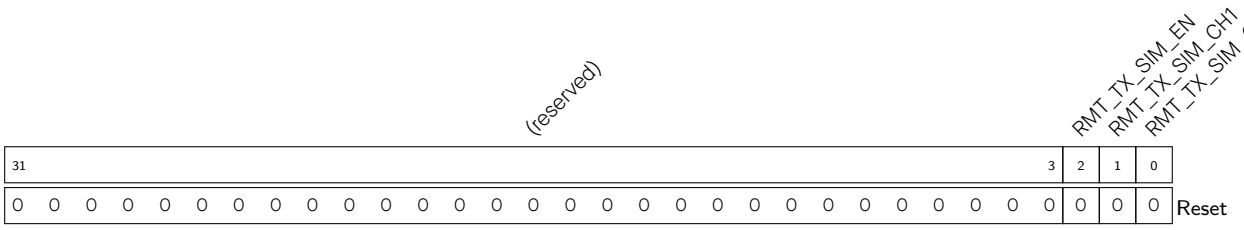
RMT_LOOP_STOP_EN_CH*n* 配置是否在通道 *n* 的循环计数器记到循环数后使能循环发送停止功能。

0：无效

1：使能

(R/W)

Register 42.16. RMT_TX_SIM_REG (0x006C)



RMT_TX_SIM_CH*n* (*n*: 0-1) 配置是否使能通道 *n* 与其它启用的通道同步开始发送数据。

0: 不使能

1: 使能

(R/W)

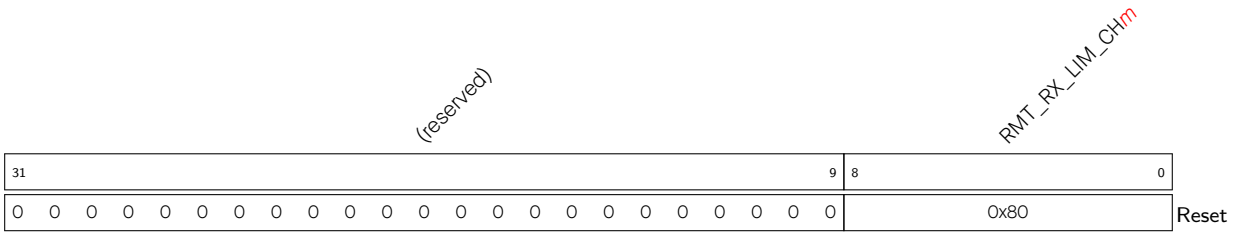
RMT_TX_SIM_EN 配置是否使能多个通道开始同步发送数据。

0: 不使能

1: 使能

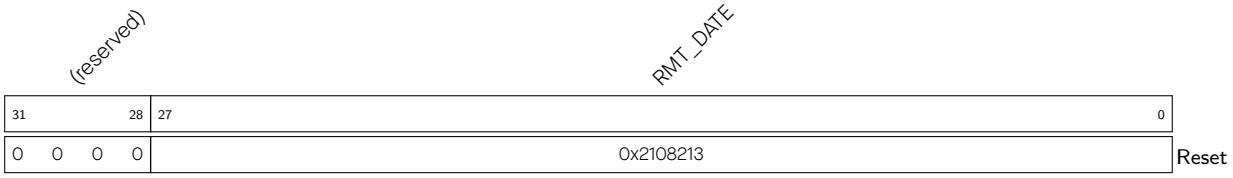
(R/W)

Register 42.17. RMT_CH*m*_RX_LIM_REG (*m*: 2-3) (0x0060+0x4*(*m*-2))



RMT_RX_LIM_CH*m* 配置通道 *m* 最大可接收的脉冲编码数量。(R/W)

Register 42.18. RMT_DATE_REG (0x00CC)



RMT_DATE 版本控制寄存器。(R/W)

第 43 章

并行 IO 控制器 (PARLIO)

43.1 概况

ESP32-C5 包含一个并行 IO 控制器 (PARLIO)，支持通过 GDMA 在并行总线上实现外部设备和内部存储器之间的数据通信。它由 TX 和 RX 两个子模块组成，分别作为发送器和接收器。PARLIO 支持全双工通信。

鉴于 IO 数据的灵活性，PARLIO 可用作通用接口，连接各种外围设备。例如可以把 SPI 作为主机，PARLIO 作为从机，实现点对点传输。详细的应用实例，请参考小节 43.8。

43.2 术语

为了更好地说明 PARLIO 的功能，本章使用了以下术语。

RX 模块 PARLIO 子模块，从外部并行总线接收数据并将其存储在内部存储器中。

TX 模块 PARLIO 子模块，将数据从内部存储器传输到外部并行总线上。

RXD 从 RX 模块的 IO 接口接收的并行数据。

TXD 从 TX 模块的 IO 接口发送的并行数据。

帧 传输数据的单位，从设置 START 信号时起，到收到 EOF 信号止。

自由运行时钟 持续翻转的时钟。若非自由运行时钟，则时钟只在收到有效数据的期间翻转，其余时间停止翻转。

GDMA SUC EOF GDMA 帧传输成功结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧正确，且数据接收结束。

GDMA ERR EOF GDMA 帧传输错误结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧错误，且数据接收结束。

CDC 跨时钟域处理 (Clock Domain Crossing)。

43.3 特性

PARLIO 模块具备以下特性：

- 支持多种时钟源可选：
  - 包括外部 IO 时钟 PAD_CLK_TX/RX、内部系统时钟 XTAL_CLK、PLL_F240M_CLK 和 RC_FAST_CLK
  - 支持最大时钟频率 40 MHz
  - 支持对时钟频率进行整数分频
- 支持将传输数据总线位宽配置为 1/2/4/8 位
- 支持 8 位全双工传输

- 总线位宽为 1/2/4 位时，支持在一个字节范围内对比特数据顺序进行翻转
- 包含用于接收 IO 并行数据的 RX 子模块：
  - 支持对输出时钟进行门控
  - 支持 RX 子模块输入时钟和输出时钟分别取反
  - 支持多种接收模式
  - 支持配置 GDMA SUC EOF 信号生成模式
  - 支持配置外部使能信号的 IO 管脚
- 包含用于发送 IO 并行数据的 TX 子模块：
  - 支持片选功能
  - 支持对输出时钟进行门控
  - 支持 TX 子模块输入时钟和输出时钟分别取反
  - 支持配置总线空闲值

43.4 系统架构

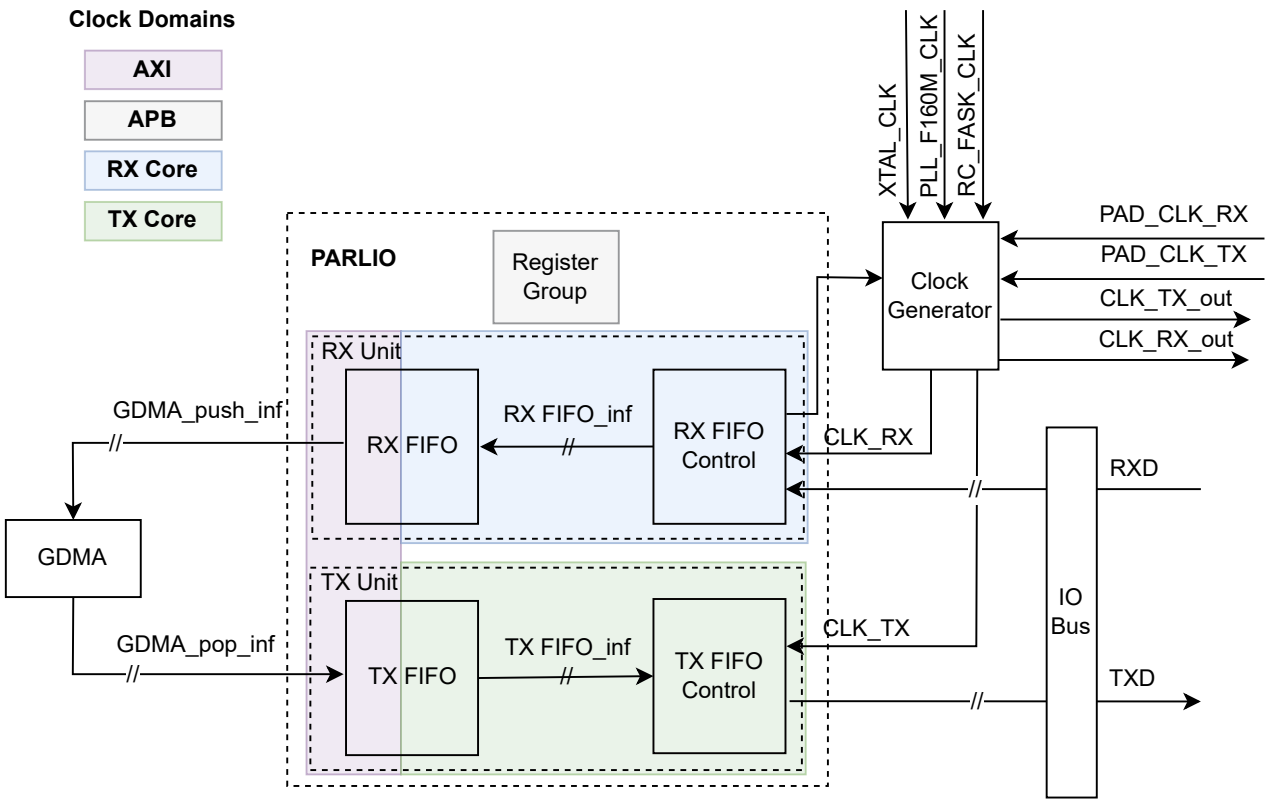


图 43.4-1. PARLIO 系统框图

图 43.4-1 展示了 PARLIO 的系统内部架构。除了 RX 模块和 TX 模块外，PARLIO 还包括一组状态配置寄存器。RX 模块接收 RXD 信号，并在完成串并转换后将其存入异步 FIFO，然后将这些数据通过 GDMA 存到内部存储器。

TX 模块通过 GDMA 从内部存储器获取数据并存入异步 FIFO，将这些数据进行并串转换后，通过 TXD 信号从 IO 总线输出。

43.5 功能描述

43.5.1 时钟生成器

PARLIO 模块有四个输入时钟域，即 RX Core、TX Core、AHB 和 APB，如图 43.4-1 所示。

状态配置寄存器组位于 APB 时钟域内。

GDMA 接口逻辑位于 AHB 时钟域内。

RX Core 和 TX Core 时钟域各有四个时钟源可供选择，分别为内部系统时钟源 XTAL_CLK、RC_FAST_CLK、PLL_F240M_CLK 和外部时钟源 (PAD_CLK_TX/RX)，如图 43.5-1 所示。配置 PCR_PARL_CLK_RX_SEL 和 PCR_PARL_CLK_TX_SEL 选择时钟源，配置 PCR_PARL_CLK_RX_DIV_NUM 和 PCR_PARL_CLK_TX_DIV_NUM 对时钟进行分频，分频系数最大可配置为 $(2^{16} - 1)$。

RX 模块和 TX 模块支持对输入时钟进行取反，同时其工作时钟可以输出到 IO，并在输出前支持对输出时钟进行取反。此外 RX 模块和 TX 模块还支持对输出时钟进行门控。

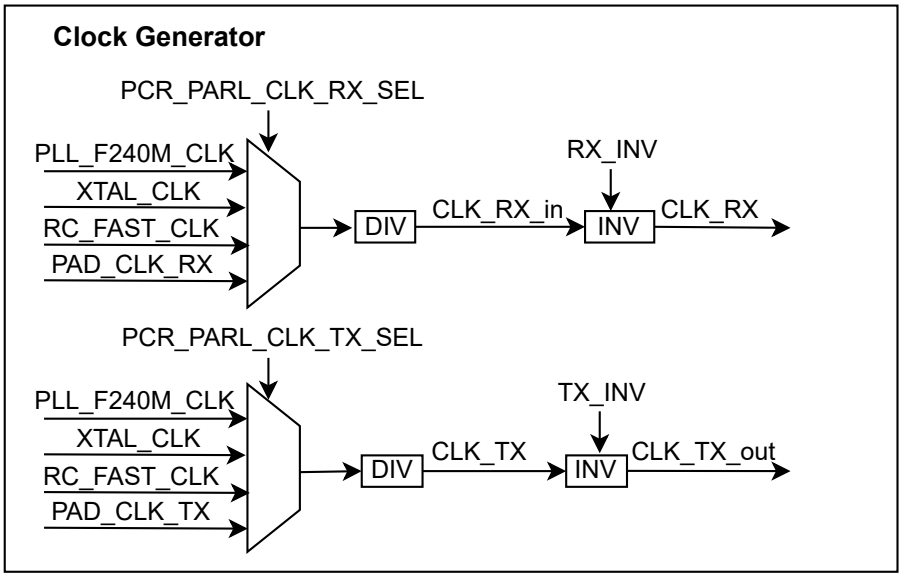


图 43.5-1. PARLIO 时钟生成

43.5.2 时钟复位使用限制

由于 PARLIO 模块的通用性，其 PAD 时钟 (PAD_CLK_TX/RX) 可能来自不同的主机（外部设备或内部时钟源），这些时钟可能是自由运行时钟，也可能不是。如果外部运行时钟不是自由运行时钟，PARLIO 内部的一些控制信号将无法进行 CDC，因此对操作过程会有一定限制。

1. 异步 FIFO 复位时，需要两个时钟周期才能初始化 FIFO。因此如果需要在使用非自由运行时钟时复位 AHB 时钟域，需保证提前两个时钟周期进行复位同步。具体的操作步骤见下表。

表 43.5-1. 时钟限制下复位 AHB 时钟域操作步骤

时钟限制		具体操作
当前帧传输时钟	下一帧传输时钟	
自由运行时钟	非自由运行时钟	在切换到非自由运行时钟之前复位下一帧传输，并在复位完成后配置时钟切换。
非自由运行时钟	自由运行时钟	只需确保在复位和正式开始传输期间有两个时钟周期的间隔。
非自由运行时钟	非自由运行时钟	如需复位下一帧传输，需要先切换到内部的自由运行时钟，复位完成后再切换回实际时钟。

2. 由于非自由运行时钟造成的限制，PARL_IO_RX_START 和 PARL_IO_TX_START 无法进行 CDC，所以在操作时必须等到 PARL_IO_RX_START 和 PARL_IO_TX_START 稳定后再开始数据传输，否则可能会发生亚稳态的问题。

   RX 模块中的具体操作步骤为：

   (a) 清除 PCR_PARL_CLK_RX_EN 关闭 RX Core 时钟域；

   (b) 向 PARL_IO_RX_START 写 1；

   (c) 置位 PCR_PARL_CLK_RX_EN 打开 RX Core 时钟域；

   (d) 操作外部设备，使其开始发送数据；

   (e) 清除 PCR_PARL_CLK_RX_EN 关闭 RX Core 时钟域；

   (f) 向 PARL_IO_RX_START 写 0。

   TX 模块中的具体操作步骤为：

   (a) 清除 PCR_PARL_CLK_TX_EN 关闭 TX Core 时钟域；

   (b) 向 PARL_IO_TX_START 写 1；

   (c) 置位 PCR_PARL_CLK_TX_EN 打开 TX Core 时钟域；

   (d) 操作外部设备，使其开始接收数据；

   (e) 清除 PCR_PARL_CLK_TX_EN 关闭 TX Core 时钟域；

   (f) 向 PARL_IO_TX_START 写 0。

3. 复位需遵循以下要求：

   - 芯片启动时，时钟复位应遵循以下顺序：

     (a) 复位 APB 时钟域；

     (b) 复位 AHB 时钟域；

     (c) 复位 Core 时钟域。

   - 帧间传输需要复位 Core 时钟域和异步 FIFO。

43.5.3 主从机模式

RX 模块和 TX 模块均支持主从机模式。

当 TX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。TX 模块在时钟的上升沿驱动 TXD。

当 TX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 43.5-2. 时钟限制下 TX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求。
非自由运行时钟	正向波形（图 43.5-2）	主机需要在下降沿时采集。
非自由运行时钟	负向波形（图 43.5-3）	主机在输出前需要对原时钟进行取反，将其转换成正向波形。

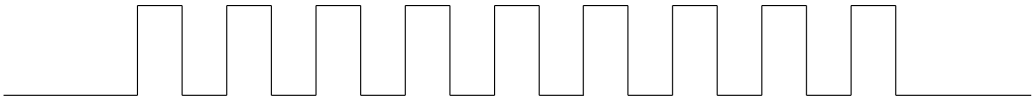


图 43.5-2. 主机时钟正向波形

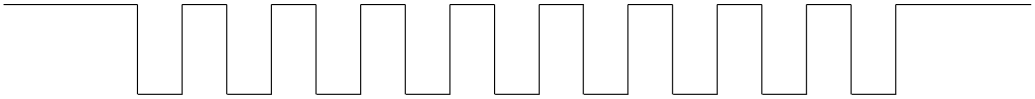


图 43.5-3. 主机时钟负向波形

当 RX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。RX 模块在时钟的上升沿驱动 RXD。

当 RX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 43.5-3. 时钟限制下 RX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求，采集的有效数据以外部使能信号为准。
非自由运行时钟	正向波形（图 43.5-2）	主机需在上升沿驱动数据，RX 模块需在下降沿采集数据（即对主时钟进行翻转）。
非自由运行时钟	负向波形（图 43.5-3）	主机需在下降沿驱动数据，RX 模块需在上升沿采集数据（即使用原始主时钟）

43.5.4 RX 模块接收模式

PARLIO 的 RX 模块有 8 种接收模式，可根据使能信号规格分为三大类：

- 电平使能模式：接收的数据由外部信号电平使能；
- 脉冲使能模式：接收的数据由外部信号脉冲使能；
- 软件使能模式：数据接收的使能信号可由用户直接配置。

RX 模块还支持对外部使能信号进行翻转。如果外部使能信号为低电平或负脉冲，可通过置位 PARL_IO_RX_EXT_EN_INV 使能该功能，转换为以下相应的接收模式。

43.5.4.1 电平使能模式

电平使能模式如图 43.5-4 所示。在这种模式下，外部使能信号的有效电平必须与有效数据对齐。由于外部电平使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 7 个。

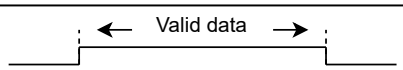
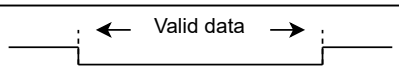
Mode	Sub-mode	Description
LEVEL_ENABLE	sub-mode 1	***signal level high*** 
	sub-mode 2	***signal level low*** 

图 43.5-4. RX 模块电平使能子模式

43.5.4.2 脉冲使能模式

根据脉冲有效电平及其与有效数据的对应性，脉冲使能模式可分为 6 种子模式，详细分类可参见图 43.5-5。

子模式 1 ~ 4 都包含起始脉冲和结束脉冲，不同之处在于起始脉冲和结束脉冲是否与有效数据对齐。

子模式 5 ~ 6 只包含起始脉冲，通过配置 PARL_IO_RX_BITLEN 可表示有效数据传输结束。

由于外部脉冲使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 7 个。但在子模式 4 和子模式 6 下，数据在脉冲的第一个边沿后和最后一个边沿前被认为有效，可以实现使能信号的 IO 管脚与数据 IO 管脚的复用，此时 RXD 可使用 8 个 IO 管脚。

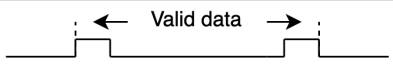
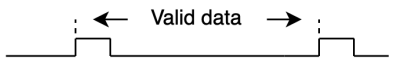
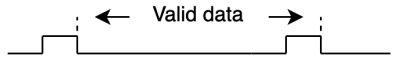
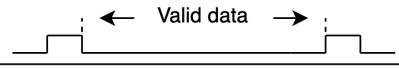
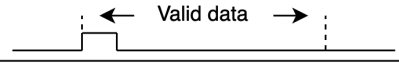
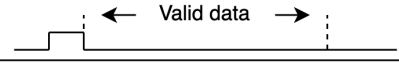
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	***pulse start(data bit include) && pulse end(data bit include)*** 
	sub-mode2	***pulse start(data bit include) && pulse end(data bit exclude)*** 
	sub-mode3	***pulse start(data bit exclude) && pulse end(data bit include)*** 
	sub-mode4	***pulse start(data bit exclude) && pulse end(data bit exclude)*** 
	sub-mode5	***pulse start(data bit include) && length end*** 
	sub-mode6	***pulse start(data bit exclude) && length end*** 

图 43.5-5. RX 模块脉冲使能子模式

43.5.4.3 软件使能模式

软件使能模式下的使能信号由内部配置寄存器决定。用户切换到此模式时，只有同时写 1 到 PARL_IO_RX_SW_EN 和 PARL_IO_RX_START，才能开始接收数据。

由于使能信号不占用接口 IO 管脚，RXD 在接口上的最大可用 IO 管脚数量为 8。由于时钟域不同，使能信号无法与有效数据对齐，因此只能通过有效时钟沿来判定有效数据。此时，RX Core 时钟需要与有效数据一一对应。

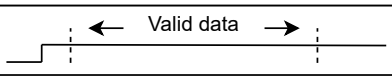
Mode	Sub-mode	Description
SW_ENABLE	/	/ 

图 43.5-6. RX 模块软件使能子模式

43.5.5 RX 模块 GDMA SUC EOF 信号生成

RX 模块产生一个 GDMA SUC EOF 信号，表示当前帧传输结束，并将其发送到 GDMA 接口。GDMA SUC EOF 信号可由外部使能信号生成，也可基于内部配置的比特长度触发。

- 当 GDMA SUC EOF 信号由内部配置的比特长度生成时，接收模式的选择不受限制。但在这种情况下必须配置 PARL_IO_RX_BITLEN。如果 PARL_IO_RX_BITLEN 的配置值小于实际接收的数据，将提前触发 GDMA SUC EOF，此时 RX 模块停止从 FIFO 读取数据，但 FIFO 会继续接收外部数据，直到 RX_FIFO_WOVF_INT 中断被触发。
- 当 GDMA SUC EOF 信号由外部使能信号产生时，只可选脉冲使能模式的子模式 1、3。在这种模式下，传输不受 PARL_IO_RX_BITLEN 值的影响，并且传输帧的数据长度不受限制。

43.5.6 RX 模块超时

RX 模块支持接收超时功能。当 RX 模块 FIFO 长时间未接收到有效数据，超时将被触发，产生一个 GDMA ERR EOF 信号，并发送至 GDMA 接口以表示结束数据接收。配置 PARL_IO_RX_TIMEOUT_THRES 可设置超时阈值。

默认情况下启用超时功能，用户也可以选择禁用该功能。超时阈值的配置上限为 AHB 时钟域的 $(2^{16}-1)$ 周期，下限取决于 AHB 时钟域和 RX Core 时钟域之间的相对频率关系。建议将 PARL_IO_RX_TIMEOUT_THRES 设置为较大的数值，以避免生成非期望的 GDMA ERR EOF 信号。

43.5.7 TX 模块无限制长度数据传输

默认情况下，TX 模块的传输结束标志由 PARL_IO_TX_BITLEN 的配置值决定。当发送的数据达到配置值时，系统会生成一个 TX EOF 中断信号，标志着该数据帧的传输结束。此时，单帧传输的最大值受到 PARL_IO_TX_BITLEN 的位宽限制，使其无法实现无限长度的数据传输。

为了实现更大长度的单帧传输，可以将 PARL_IO_TX_EOF_GEN_SEL 设置为 1。在这种配置下，硬件将使用 GDMA 发送链表配置的 EOF 信号作为单帧传输结束的标志。此时，数据传输不再受限于 PARL_IO_TX_BITLEN 的位宽，从而能够支持更大长度的数据传输。

43.5.8 TX 片选功能

为模拟 LCD 的 I8080 协议，TX 模块配备了一个可输出的片选信号，在拉低时有效。用户可通过配置 PARL_IO_TX_CS_START_DELAY 和 PARL_IO_TX_CS_STOP_DELAY 来控制 DMA 输出数据和片选信号的相位关系。需要注意的是，在开启时钟门控、且使用片选信号作为时钟门控控制源的情况下，对上述两个寄存器的操作在实现基础功能的同时，还可以控制输出时钟和片选信号的相位关系。

43.5.9 TX 模块输出时钟门控

TX 模块支持对输出时钟进行门控。时钟门控功能默认关闭，可通过软件置位 PARL_IO_TX_GATING_EN 使能该功能。

目前 TXD 最高位有两个控制源可配置，分别为 DMA 输出数据和片选信号 (CS)。将 PARL_IO_TX_VALID_OUTPUT_EN 配置为 0 即选择 DMA 输出数据为控制源，此时必须将 TX 模块配置为最大位宽，因为当控制源为 DMA 输出数据时，门控信号固定为 TXD 最高位，该位为高电平时，时钟信号可以翻转。需要注意的是，在选择 DMA 输出数据作为控制源时，如果外设处于空闲状态，则此时时钟门控将被用户配置的总线空闲值的最高位控制，应参照小节 43.5.10 *TX 模块总线空闲值* 中的注意事项进行配置；若选择 DMA 输出数据作为门控功能控制源时启用时钟门控功能，门控信号将占用一个 IO，此时 TXD 可以使用的 IO 管脚最多为 7 个。

将 PARL_IO_TX_VALID_OUTPUT_EN 配置为 1 即选择片选信号为控制源，此时对位宽无限制。

43.5.10 TX 模块总线空闲值

TX 模块不传输数据时被认为保持在空闲状态。TX 模块支持总线空闲值可配置。

注意，应避免配置的空闲值与其他已开启的功能产生冲突，例如当使用 TXD 的最高有效位作为有效信号时，应避免将空闲值的最高有效位配置为 1。

43.5.11 单帧数据传输

RX/TX 模块以比特为单位进行传输，即单帧至少传输 1 个比特的数据。

当 RX 模块通过比特长度生成 GDMA EOF 信号时，单帧传输的最大长度是 $(2^{19} - 1)$ 比特。当 RX 模块通过外部设备的使能信号产生 GDMA EOF 信号时，单帧传输的比特数没有限制。

TX 模块仅通过比特长度生成 EOF 信号，因此单帧传输的最大长度为 $(2^{19} - 1)$ 比特。

由于 PARLIO 在 GDMA 一侧以字节为单位进行传输，因此当 IO 一侧的数据不与字节对齐时，PARLIO 会对数据进行特殊处理。

- 在接收状态下，通过 GDMA 存入内存的数据将自动在高位补 0，从而与字节对齐。
- 在发送状态下，从内存中取出的数据将根据配置的 PARL_IO_TX_BITLEN 值被截断，超过该配置值的数据不会被发送。

当配置的数据总线宽度为 2/4/8 位时，比特长度必须配置为对应总线宽度的倍数。

43.5.12 字节范围内数据顺序翻转

总线位宽为 1/2/4 位时，可以对一个字节范围内的数据顺序进行翻转。以 RX 模块为例，当配置总线位宽为 2 比特时，数据在被写入 RX FIFO 之前需要被打包成一个字节。

假设原始的比特数据排列顺序为：

$$\{\{b_0, b_1\}, \{b_2, b_3\}, \{b_4, b_5\}, \{b_6, b_7\}\}$$

则启用该功能后，数据顺序将更改为：

$$\{\{b_6, b_7\}, \{b_4, b_5\}, \{b_2, b_3\}, \{b_0, b_1\}\}$$

43.6 中断

ESP32-C5 的 PARLIO 模块可以生成以下中断信号并发送给中断矩阵。

- PARL_IO_TX_INTR
- PARL_IO_RX_INTR

这些中断信号由 PARLIO 模块的内部中断源生成。表 43.6-1 列出了 PARLIO 的内部中断源、中断触发条件、以及生成的中断信号。

表 43.6-1. PARLIO 的内部中断源

内部中断源	触发条件	中断信号
TX_FIFO_REMPTY_INT	TX 模块 FIFO 空（此时 TX 发送的数据可能出现错误）	PARL_IO_TX_INTR
RX_FIFO_WOVF_INT	RX 模块 FIFO 满（此时 RX 接收的数据可能出现错误）	PARL_IO_RX_INTR
TX_EOF_INT	TX 模块发送完成完整的一帧数据	PARL_IO_TX_INTR

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 43.9 寄存器列表。

43.7 配置流程

43.7.1 数据接收操作流程

本节介绍 RX 模块接收数据的配置流程。参照以下流程，从连接到外部设备的 IO 管脚接收并行数据，并存储在内部存储器中。关于 RX 模块时钟和复位的详细操作限制，请参考小节 43.5.2。

1. 复位 RX 模块。具体复位方法与顺序，请参考小节 43.5.2。
2. 置位 PARL_IO_RX_FIFO_WOVF_INT_CLR 和 PARL_IO_RX_FIFO_WOVF_INT_ENA。
3. 选择 RXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置 PCR 寄存器，选择时钟源并对时钟进行分频。
5. 关闭 RX 模块 Core 时钟域的时钟。
6. 按照小节 43.3 和 43.5 中的描述，选择所需接收模式并启用相关功能。
7. 配置 GDMA 接收链表。

8. 置位 PARL_IO_RX_REG_UPDATE，同步寄存器信号。
9. 置位 PARL_IO_RX_START。
10. 打开 RX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其发送数据。
12. 轮询 GDMA SUC EOF 中断信号。
13. 清除 GDMA SUC EOF 中断信号。
14. 关闭 RX 模块 Core 时钟域的时钟。
15. 清除 PARL_IO_RX_START。

43.7.2 数据发送操作流程

本节介绍 TX 模块发送数据的编程流程。参照以下流程，将内部存储器中的并行数据传输到与外部设备相连的 IO 管脚。关于 TX 模块时钟和复位的详细操作限制，请参考小节 43.5.2。

1. 复位 TX 模块。具体复位方法与顺序，请参考小节 43.5.2。
2. 依次置位 PARL_IO_TX_FIFO_REMPTY_INT_CLR、PARL_IO_TX_EOF_INT_CLR、PARL_IO_TX_FIFO_REMPTY_INT_ENA 和 PARL_IO_TX_EOF_INT_ENA。
3. 选择 TXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置 PCR 寄存器，选择时钟源并对时钟进行分频。
5. 关闭 TX 模块 Core 时钟域的时钟。
6. 按照小节 43.5 中的描述，选择开启所需功能。
7. 配置 GDMA 发送链表。
8. 轮询 PARL_IO_TX_READY。
9. 置位 PARL_IO_TX_START。
10. 打开 TX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其接收数据。
12. 轮询 PARL_IO_TX_EOF_INT_ST 中断信号。
13. 置位 PARL_IO_TX_EOF_INT_CLR 中断信号。
14. 关闭 TX 模块 Core 时钟域的时钟。
15. 清除 PARL_IO_TX_START。

43.8 应用示例

本节介绍 PARLIO 模块的应用示例及其具体操作过程。示例中使用的外围设备都来自 ESP 系列芯片，这些外设均可以和 PARLIO 模块组成完整的数据通路。

> **说明：**
> 示例中所构建的数据通路可能不是最佳方案，例如，在实际应用中，用户可使用两颗相同 ESP 芯片上的 SPI 外设来完

成点对点传输，而无需使用 PARLIO。但本小节中的示例在一定程度上证明了 PARLIO 接口的灵活性。

43.8.1 与 SPI 进行数据传输

在本示例中，外部 SPI 作为主机发送数据，PARLIO RX 模块作为从机接收数据，与此同时，PARLIO TX 模块作为主机发送数据，外部 SPI 作为从机接收数据，从而实现点对点串行数据传输。

- 实现 SPI 发送、PARLIO 接收，请参考以下操作过程：
  1. 配置 SPI 时钟。
  2. 将 SPI 配置为主机。
  3. 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_RX，FSPICS0 管脚连接至 RXD[7]，FSPID 管脚连接至 RXD[0]。
  4. 将发送的数据写入 SPI 缓冲器中，并配置发送数据的位长。
  5. 置位 SPI_UPDATE，更新配置的寄存器值。
  6. 复位 PARLIO RX 模块。
  7. 配置 PARLIO RX 模块时钟。
  8. 关闭 PARLIO RX 模块 Core 时钟域。
  9. 将 PARLIO 接收模式配置为电平使能模式的子模式 1，配置 RX 模块数据总线宽度为 1 位，然后根据 SPI 的发送数据长度配置 PARL_IO_RX_BITLEN，最后置位 PARL_IO_RX_REG_UPDATE。
  10. 配置 PARLIO GDMA 接收链表。
  11. 置位 PARL_IO_RX_START。
  12. 打开 PARLIO RX 模块 Core 时钟域。
  13. 置位 SPI_USR，开始传输 SPI 的数据。
  14. 轮询 GDMA SUC EOF 中断信号。
  15. 清除 PARL_IO_RX_START。
- 实现 PARLIO 发送、SPI 接收，请参考以下操作过程：
  1. 配置 SPI 时钟。
  2. 将 SPI 配置为从机。
  3. 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_TX，FSPICS0 管脚连接至 TXD[7]，FSPID 管脚连接至 TXD[0]。
  4. 置位 SPI_RD_BIT_ORDER，翻转比特数据顺序。
  5. 置位 SPI_UPDATE，更新配置的寄存器值。
  6. 复位 PARLIO TX 模块。
  7. 置位 PARL_IO_TX_EOF_INT_CLR 和 PARL_IO_TX_EOF_INT_ENA。
  8. 配置 PARLIO TX 模块时钟。
  9. 关闭 PARLIO TX 模块 Core 时钟域。

10. 配置数据总线宽度为 1 位，写 1 至 PARL_IO_TX_VALID_OUTPUT_EN，然后配置 PARL_IO_TX_BITLEN。
11. 配置 PARLIO GDMA 发送链表。
12. 轮询 PARL_IO_TX_READY。
13. 写 1 至 PARL_IO_TX_START。
14. 打开 PARLIO TX 模块 Core 时钟域。
15. 开始数据传输。
16. 轮询 PARL_IO_TX_EOF_INT_ST。
17. 置位 PARL_IO_TX_EOF_INT_CLR。
18. 关闭 PARLIO TX 模块 Core 时钟域。
19. 清除 PARL_IO_TX_START。

43.8.2 与 I2S 进行数据传输

在本示例中，外部 I2S 作为主机发送数据，PARLIO RX 模块作为从机接收数据。PARLIO 支持 I2S TDM MSB 对齐标准模式和 TDM PCM 标准模式下的传输。当 I2S 传输协议为 TDM MSB 对齐标准模式时，需将 PARLIO 的接收模式配置为电平使能模式。当 I2S 传输协议为 TDM PCM 标准模式时，需将 PARLIO 的接收模式配置为脉冲使能模式的子模式 4，并置位 PARL_IO_RX_EXT_EN_INV。

本节以 TDM PCM 标准模式为例，具体操作过程如下：

1. 配置 I2S 时钟。
2. 配置信号管脚，将 I2SO_BCK_out 连接至 PAD_CLK_RX，I2SO_WS_out 连接至 RXD[7]，I2SO_Data_out 管脚连接至 RXD[0]。
3. 将 I2S 配置为主机。
4. 配置所需 I2S TX 数据和通道模式，并置位 I2S_TX_UPDATE。
5. 复位 I2S TX 模块和 TX FIFO。
6. 使能 I2S_TX_DONE_INT。
7. 配置 I2S GDMA 发送链表。
8. 置位 I2S_TX_STOP_EN。
9. 复位 PARLIO RX 模块。
10. 配置 PARLIO RX 模块时钟。
11. 关闭 PARLIO RX 模块 Core 时钟域。
12. 将 PARLIO 接收模式配置为脉冲使能模式的子模式 4，配置 RX 模块数据总线宽度为 1 位，然后根据 I2S 的发送数据长度配置 PARL_IO_RX_BITLEN，最后置位 PARL_IO_RX_REG_UPDATE。
13. 配置 PARLIO GDMA 接收链表。
14. 置位 PARL_IO_RX_START。
15. 打开 PARLIO RX 模块 Core 时钟域。

16. 置位 I2S_TX_START，开始发送数据。
17. 轮询 I2S_TX_DONE_INT。
18. 轮询 GDMA SUC EOF 中断信号。
19. 清除 I2S_TX_START。
20. 清除 PARL_IO_RX_START。

43.9 寄存器列表

本小节的所有地址均为相对于并行 IO 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

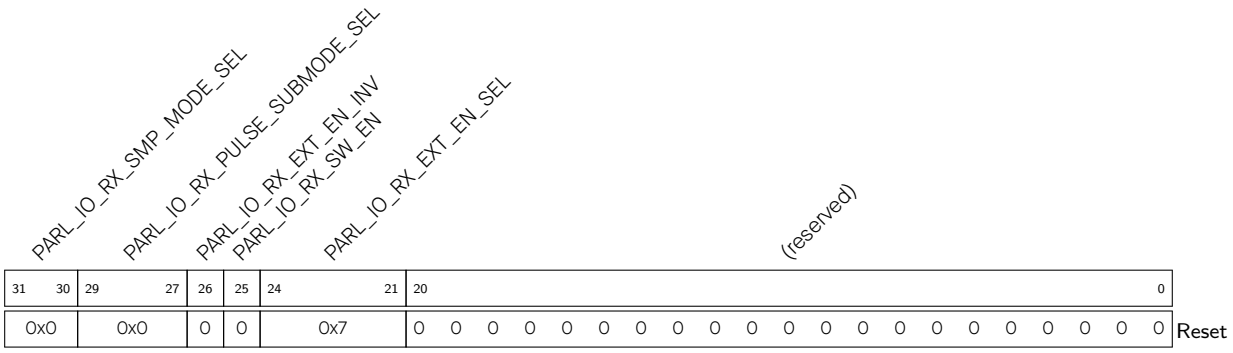
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
PARLIO RX 配置寄存器			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX 采样模式配置寄存器	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX 数据配置寄存器	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX 通用配置寄存器	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX 开始配置寄存器	0x000C	R/W
PARLIO TX 配置寄存器			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX 数据配置寄存器	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX 开始配置寄存器	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX 通用配置寄存器	0x0018	R/W
PARLIO 配置与状态寄存器			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO 配置寄存器	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO 寄存器更新配置寄存器	0x0020	WT
PARL_IO_ST_REG	PARLIO 状态寄存器	0x0024	RO
PARLIO 中断配置与状态寄存器			
PARL_IO_INT_ENA_REG	PARLIO 中断使能寄存器	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO 原始中断寄存器	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO 中断状态寄存器	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO 中断清除寄存器	0x0034	WT
PARLIO RX/TX 状态寄存器			
PARL_IO_RX_ST0_REG	PARLIO RX 状态寄存器 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX 状态寄存器 1	0x003C	RO
PARL_IO_TX_ST0_REG	PARLIO TX 状态寄存器 0	0x0040	RO
PARLIO 时钟配置寄存器			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX 时钟配置寄存器	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX 时钟配置寄存器	0x0048	R/W
PARL_IO_CLK_REG	PARLIO 时钟配置寄存器	0x0120	R/W
PARLIO TX 片选配置寄存器			
PARL_IO_TX_CS_CFG_REG	PARLIO TX 片选配置寄存器	0x004C	R/W
PARLIO 版本寄存器			
PARL_IO_VERSION_REG	版本控制寄存器	0x03FC	R/W

43.10 寄存器

本小节的所有地址均为相对于并行 IO 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 43.1. PARL_IO_RX_MODE_CFG_REG (0x0000)



PARL_IO_RX_EXT_EN_SEL 配置 RX 外部使能信号的 IO 管脚。(R/W)

PARL_IO_RX_SW_EN 配置是否使能软件数据采集。

0：不使能

1：使能

(R/W)

PARL_IO_RX_EXT_EN_INV 配置是否翻转外部使能信号。

0：无效

1：翻转

(R/W)

PARL_IO_RX_PULSE_SUBMODE_SEL 配置 RXD 脉冲使能子模式。

0：正脉冲起始（包括数据位）& 正脉冲结束（包括数据位）

1：正脉冲起始（包括数据位）& 正脉冲结束（不包括数据位）

2：正脉冲起始（不包括数据位）& 正脉冲结束（包括数据位）

3：正脉冲起始（不包括数据位）& 正脉冲结束（不包括数据位）

4：正脉冲起始（包括数据位）& 长度结束

5：正脉冲起始（不包括数据位）& 长度结束

6 ~ 7：无效值

(R/W)

PARL_IO_RX_SMP_MODE_SEL 配置 RXD 采集模式。

0：外部电平使能模式

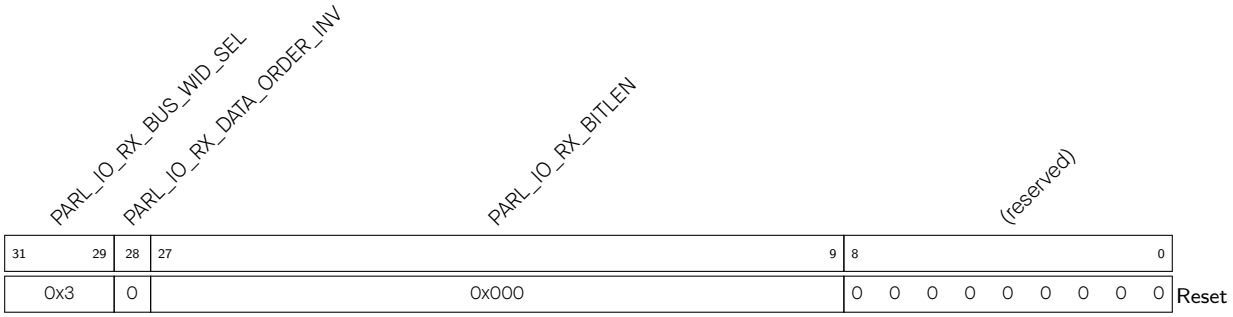
1：外部脉冲使能模式

2：内部软件使能模式

3：无效值

(R/W)

Register 43.2. PARL_IO_RX_DATA_CFG_REG (0x0004)



PARL_IO_RX_BITLEN 配置 RXD 的比特长度。(R/W)

PARL_IO_RX_DATA_ORDER_INV 配置是否翻转从 RX FIFO 发送到 GDMA 的数据的字节范围内比特顺序。

0：无效

1：翻转数据顺序

(R/W)

PARL_IO_RX_BUS_WID_SEL 配置 RXD 总线位宽。

0：1 位

1：2 位

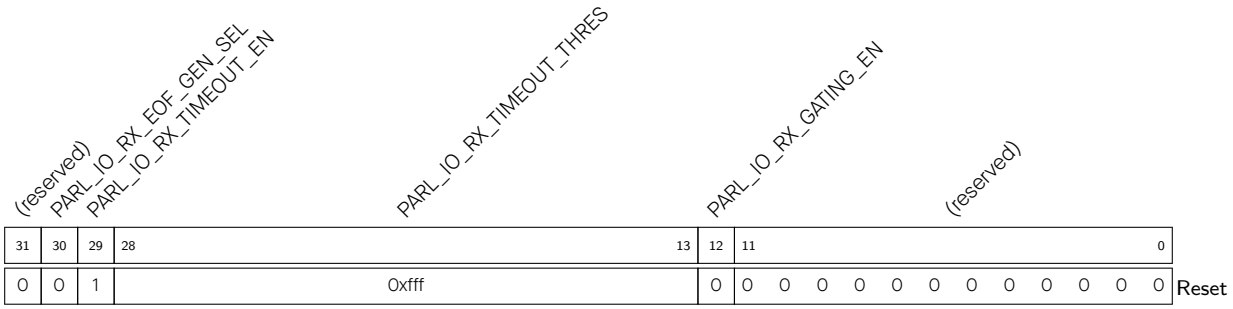
2：4 位

3：8 位

4 ~ 15：无效值，会引发错误

(R/W)

Register 43.3. PARL_IO_RX_GENRL_CFG_REG (0x0008)



PARL_IO_RX_GATING_EN 配置是否使能 RX 输出时钟的时钟门控。

0：不使能

1：使能

(R/W)

PARL_IO_RX_TIMEOUT_THRES 配置 RX 超时计数器的阈值。(R/W)

PARL_IO_RX_TIMEOUT_EN 配置是否使能超时功能来生成 GDMA ERR EOF。

0：不使能

1：使能

(R/W)

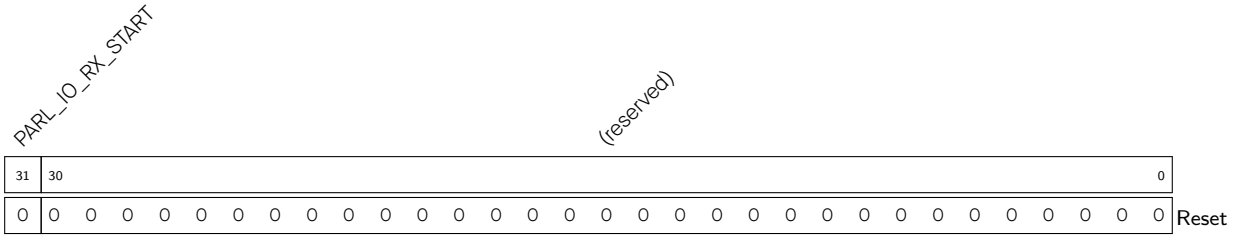
PARL_IO_RX_EOF_GEN_SEL 配置 GDMA SUC EOF 的生成模式。

0：通过配置的数据比特长度生成 GDMA SUC EOF

1：通过外部使能信号生成 GDMA SUC EOF

(R/W)

Register 43.4. PARL_IO_RX_START_CFG_REG (0x000C)



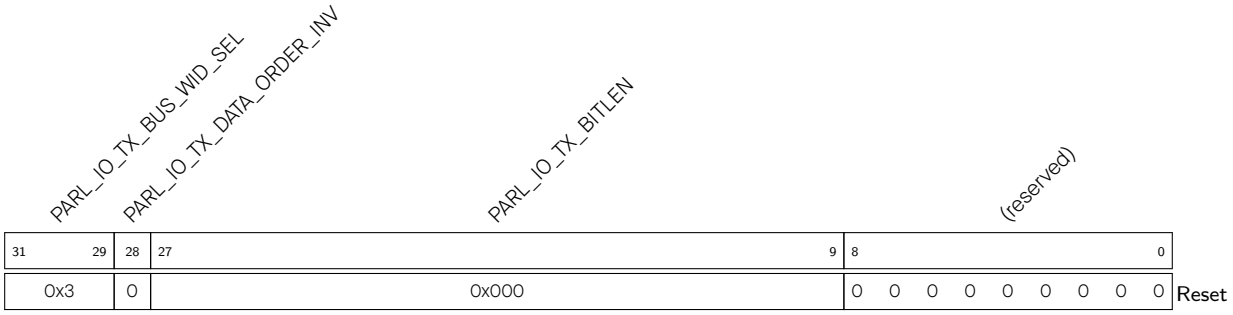
PARL_IO_RX_START 配置是否开始 RX 数据采集。

0：无效

1：开始

(R/W)

Register 43.5. PARL_IO_TX_DATA_CFG_REG (0x0010)



PARL_IO_TX_BITLEN 配置 TXD 的比特长度。(R/W)

PARL_IO_TX_DATA_ORDER_INV 配置是否翻转从 TX_FIFO 发送到 IO 的数据的字节范围内比特顺序。

0：无效

1：翻转数据顺序

(R/W)

PARL_IO_TX_BUS_WID_SEL 配置 TXD 总线位宽。

0：1 位

1：2 位

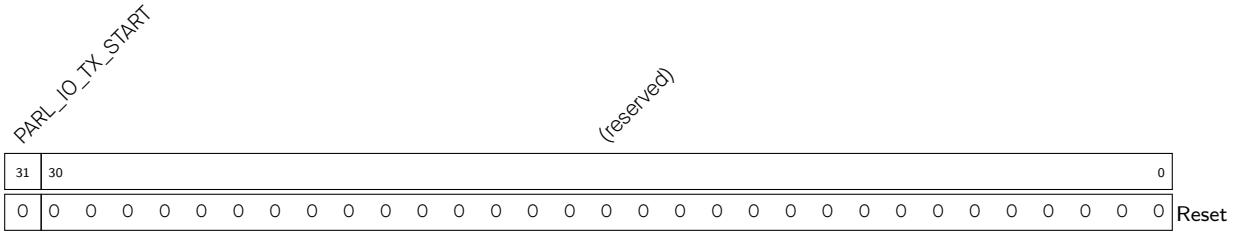
2：4 位

3：8 位

4 ~ 15：无效值，会引发错误

(R/W)

Register 43.6. PARL_IO_TX_START_CFG_REG (0x0014)



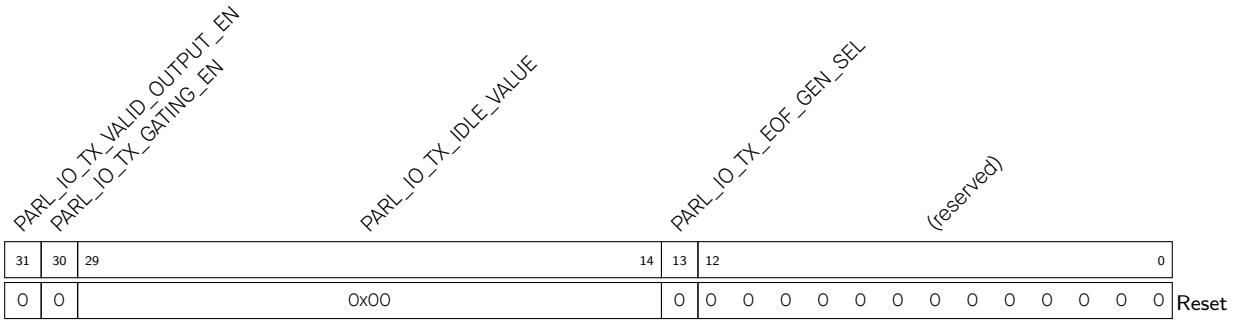
PARL_IO_TX_START 配置是否开始 TX 数据输出。

0：无效

1：开始

(R/W)

Register 43.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)



PARL_IO_TX_EOF_GEN_SEL 配置 TX EOF 的生成模式。

0：通过配置的数据比特长度生成 TX EOF

1：通过 GDMA EOF 信号生成 TX EOF

(R/W)

PARL_IO_TX_IDLE_VALUE 配置 TX 总线上在未发送状态下的数据值。(R/W)

PARL_IO_TX_GATING_EN 配置是否使能 TX 输出时钟的时钟门控。

0：不使能

1：使能

(R/W)

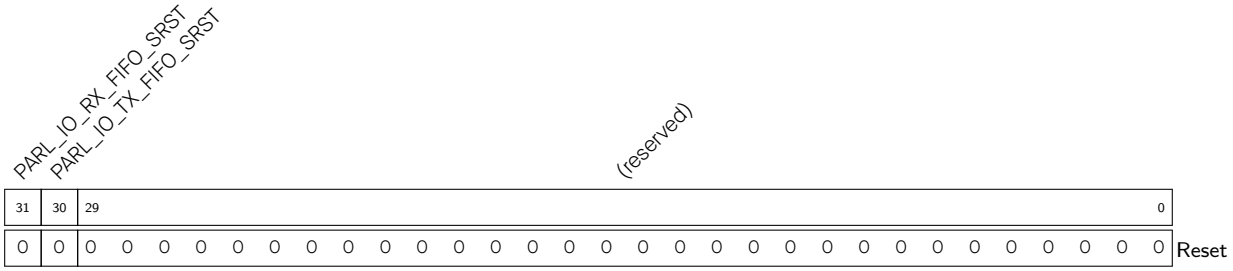
PARL_IO_TX_VALID_OUTPUT_EN 配置使能 DMA 输出数据或者片选信号作为 TXD 最高位控制源。

0：选择 DMA 输出数据为控制源

1：选择片选信号为控制源

(R/W)

Register 43.8. PARL_IO_FIFO_CFG_REG (0x001C)



PARL_IO_TX_FIFO_SRST 配置是否复位 TX 异步 FIFO。

0：无效

1：复位

(R/W)

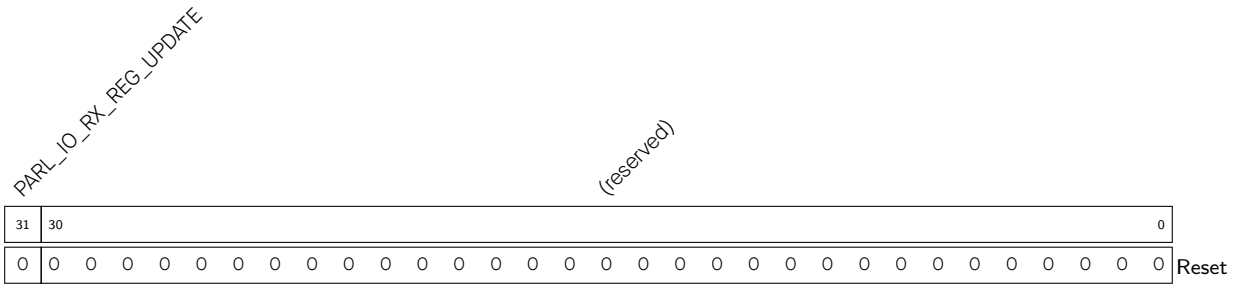
PARL_IO_RX_FIFO_SRST 配置是否复位 RX 异步 FIFO。

0：无效

1：复位

(R/W)

Register 43.9. PARL_IO_REG_UPDATE_REG (0x0020)



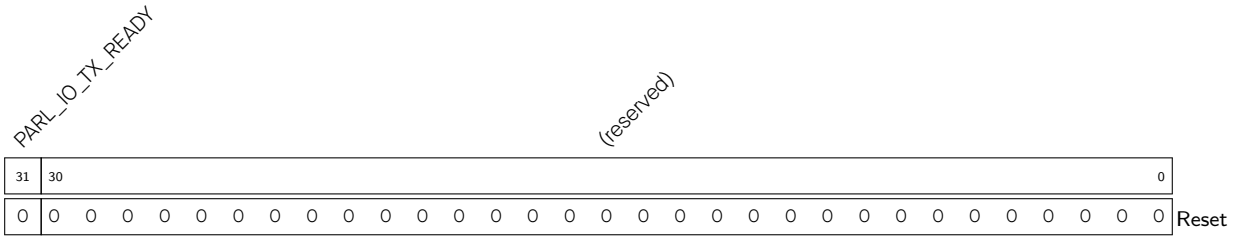
PARL_IO_RX_REG_UPDATE 配置是否更新 RX 寄存器配置。

0：无效

1：更新

(WT)

Register 43.10. PARL_IO_ST_REG (0x0024)



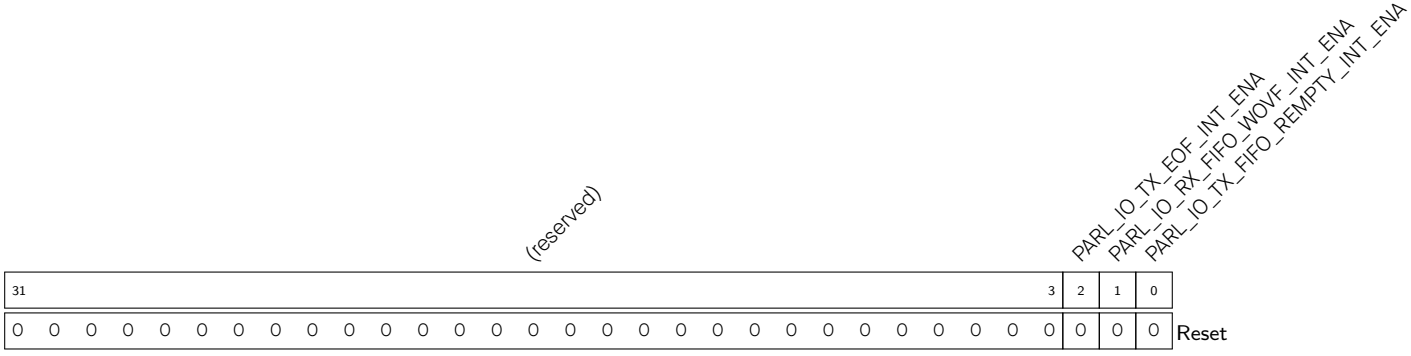
PARL_IO_TX_READY 表示 TX 模块是否已准备好传输数据。

0：未准备好

1：已准备好

(RO)

Register 43.11. PARL_IO_INT_ENA_REG (0x0028)

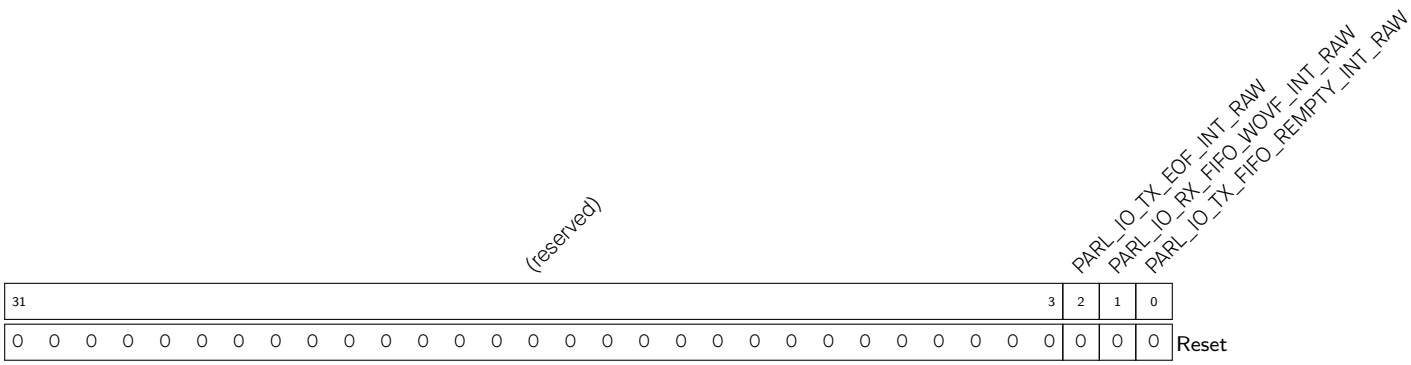


PARL_IO_TX_FIFO_REMPTY_INT_ENA 写 1 使能 TX_FIFO_REMPTY_INT 的屏蔽中断状态。(R/W)

PARL_IO_RX_FIFO_WOVF_INT_ENA 写 1 使能 RX_FIFO_WOVF_INT 的屏蔽中断状态。(R/W)

PARL_IO_TX_EOF_INT_ENA 写 1 使能 TX_EOF_INT 的屏蔽中断状态。(R/W)

Register 43.12. PARL_IO_INT_RAW_REG (0x002C)

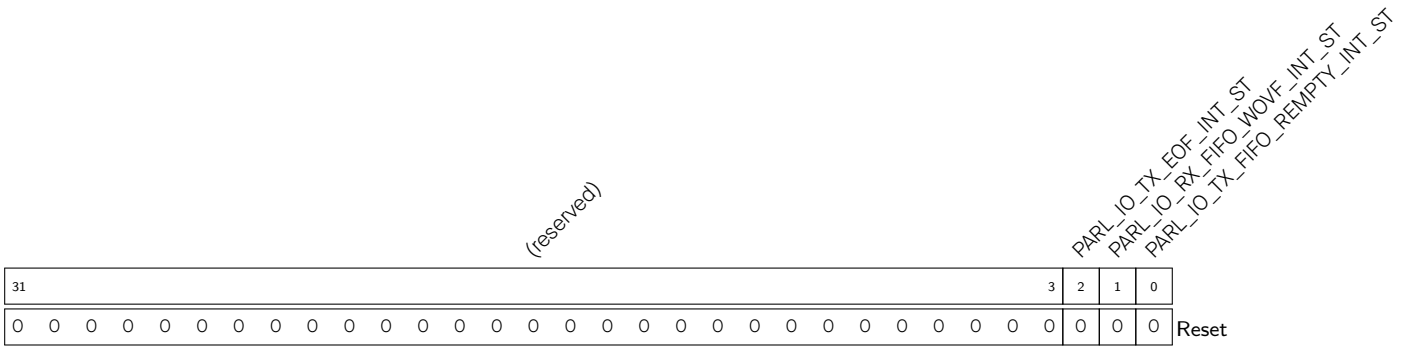


PARL_IO_TX_FIFO_REMPTY_INT_RAW TX_FIFO_REMPTY_INT 的原始中断状态。(R/SS/WTC)

PARL_IO_RX_FIFO_WOVF_INT_RAW RX_FIFO_WOVF_INT 的原始中断状态。(R/SS/WTC)

PARL_IO_TX_EOF_INT_RAW TX_EOF_INT 的原始中断状态。(R/SS/WTC)

Register 43.13. PARL_IO_INT_ST_REG (0x0030)

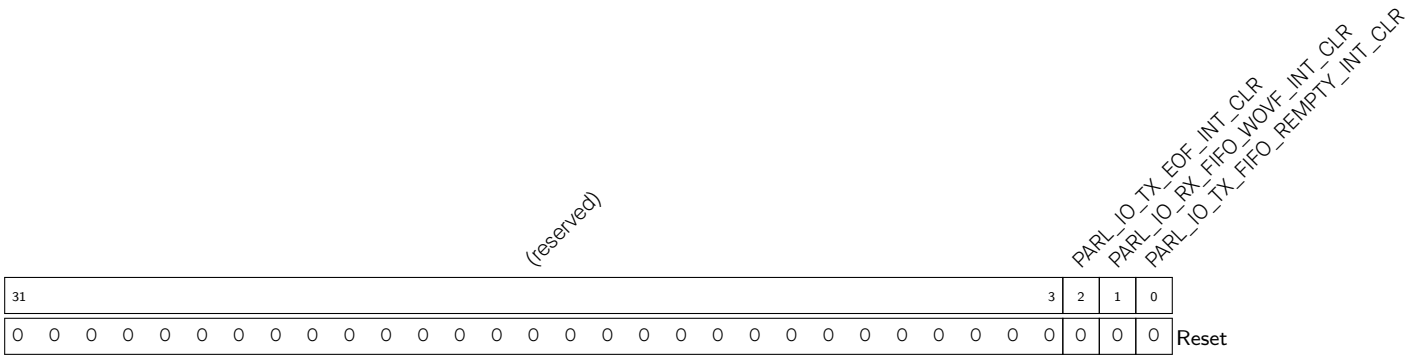


PARL_IO_TX_FIFO_REMPTY_INT_ST TX_FIFO_REMPTY_INT 的屏蔽中断状态。(RO)

PARL_IO_RX_FIFO_WOVF_INT_ST RX_FIFO_WOVF_INT 的屏蔽中断状态。(RO)

PARL_IO_TX_EOF_INT_ST TX_EOF_INT 的屏蔽中断状态。(RO)

Register 43.14. PARL_IO_INT_CLR_REG (0x0034)

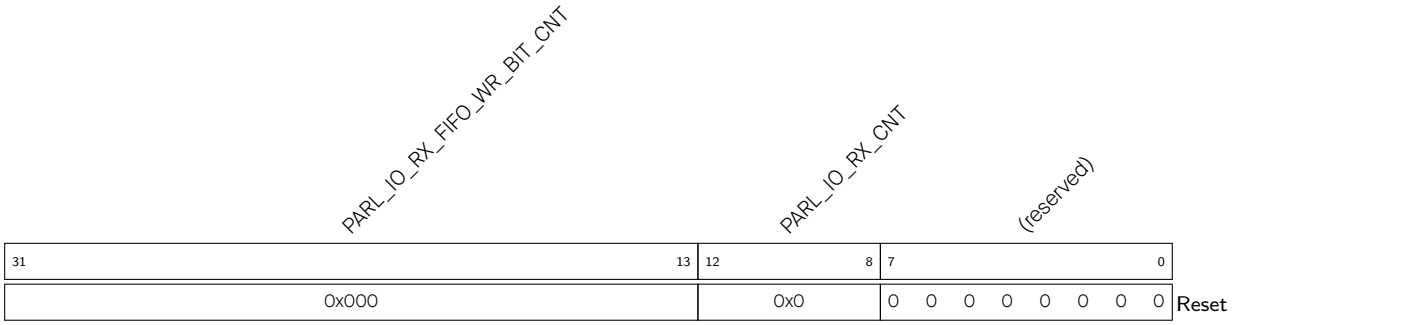


PARL_IO_TX_FIFO_REMPTY_INT_CLR 写 1 清除 TX_FIFO_REMPTY_INT。(WT)

PARL_IO_RX_FIFO_WOVF_INT_CLR 写 1 清除 RX_FIFO_WOVF_INT。(WT)

PARL_IO_TX_EOF_INT_CLR 写 1 清除 TX_EOF_INT。(WT)

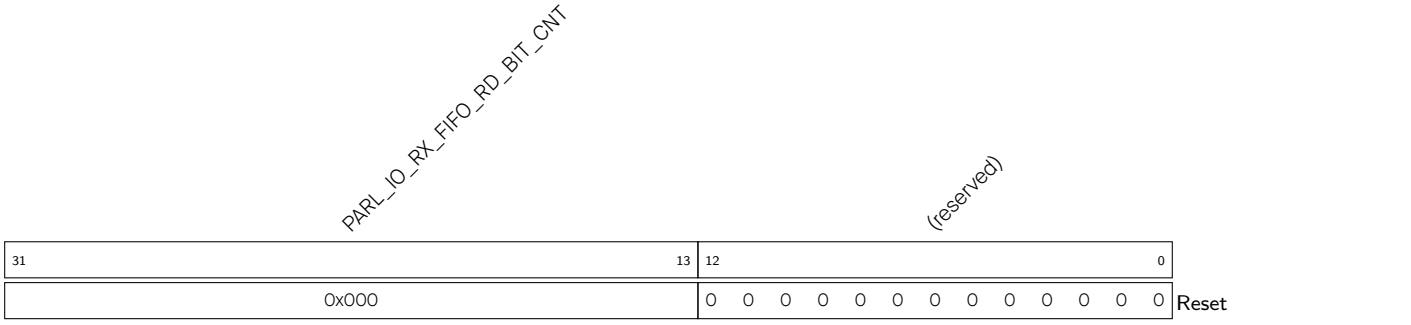
Register 43.15. PARL_IO_RX_ST0_REG (0x0038)



PARL_IO_RX_CNT 表示读取 RX FIFO 的时钟周期数。(RO)

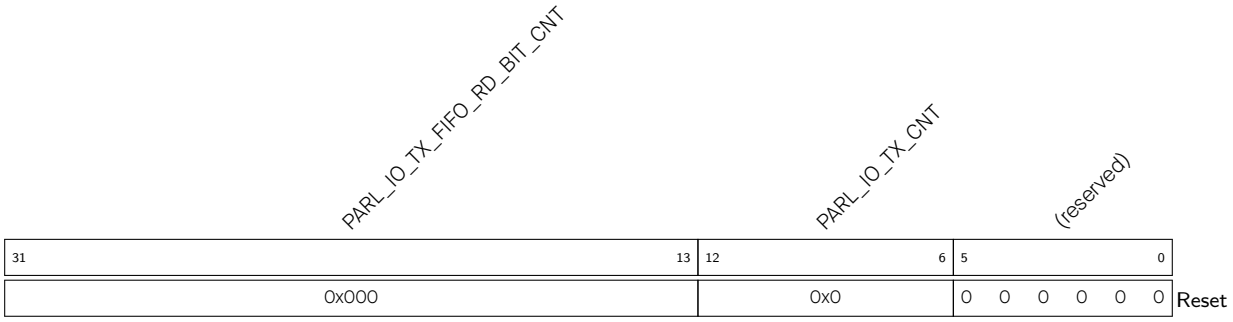
PARL_IO_RX_FIFO_WR_BIT_CNT 表示当前写入 RX FIFO 的位数。(RO)

Register 43.16. PARL_IO_RX_ST1_REG (0x003C)



PARL_IO_RX_FIFO_RD_BIT_CNT 表示当前从 RX FIFO 中读取的位数。(RO)

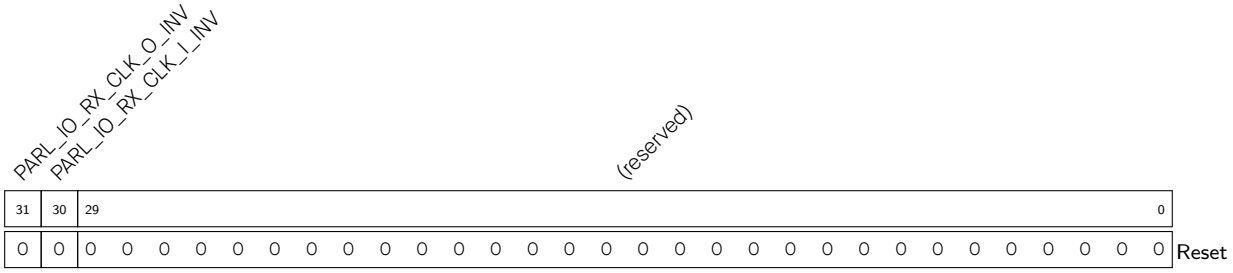
Register 43.17. PARL_IO_TX_ST0_REG (0x0040)



PARL_IO_TX_CNT 表示读取 TX FIFO 的周期数。(RO)

PARL_IO_TX_FIFO_RD_BIT_CNT 表示当前从 TX FIFO 中读取的位数。(RO)

Register 43.18. PARL_IO_RX_CLK_CFG_REG (0x0044)



PARL_IO_RX_CLK_I_INV 配置是否翻转 RX 的输入 Core 时钟。

0：无效

1：翻转

(R/W)

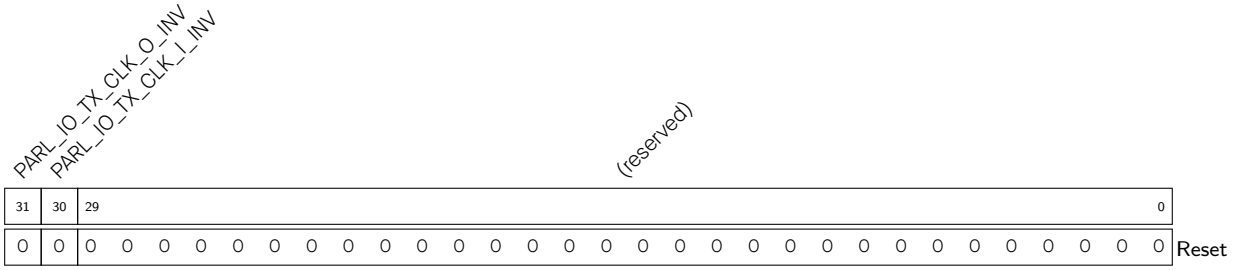
PARL_IO_RX_CLK_O_INV 配置是否翻转 RX 的输出 Core 时钟。

0：无效

1：翻转

(R/W)

Register 43.19. PARL_IO_TX_CLK_CFG_REG (0x0048)



PARL_IO_TX_CLK_I_INV 配置是否翻转 TX 的输入 Core 时钟。

0：无效

1：翻转

(R/W)

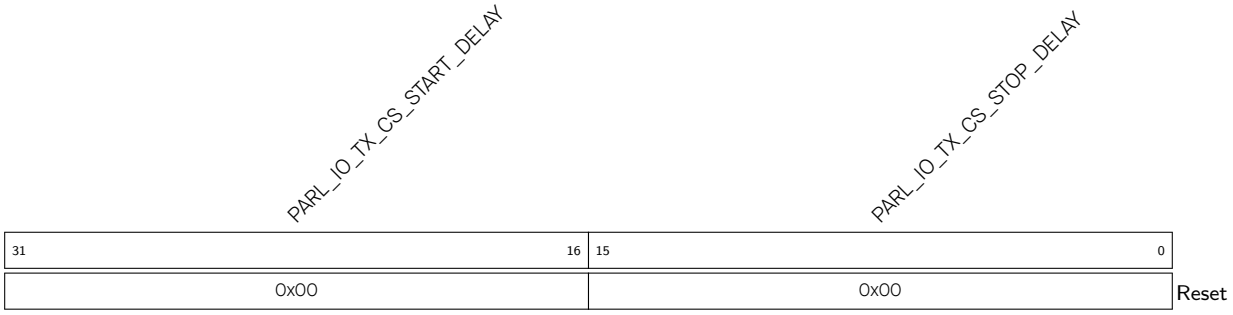
PARL_IO_TX_CLK_O_INV 配置是否翻转 TX 的输出 Core 时钟。

0：无效

1：翻转

(R/W)

Register 43.20. PARL_IO_TX_CS_CFG_REG (0x004C)



PARL_IO_TX_CS_STOP_DELAY 配置 TX 数据发送结束与 TX_CS_O 上升沿之间的延迟。

单位：TX Core 时钟周期

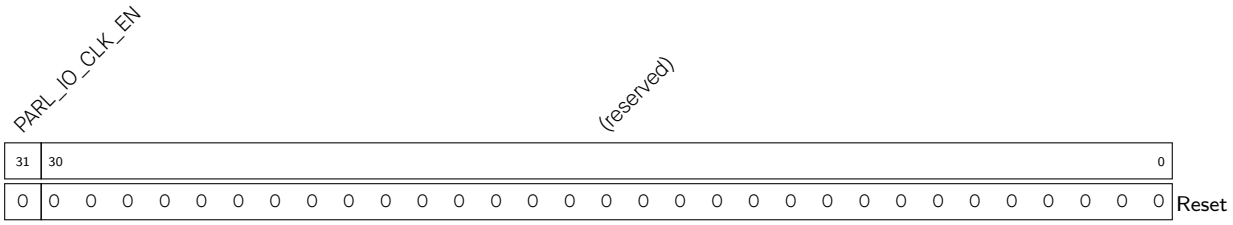
(R/W)

PARL_IO_TX_CS_START_DELAY 配置 TX_CS_O 下降沿与 TX 数据开始传输之间的延迟。

单位：TX Core 时钟周期

(R/W)

Register 43.21. PARL_IO_CLK_REG (0x0120)



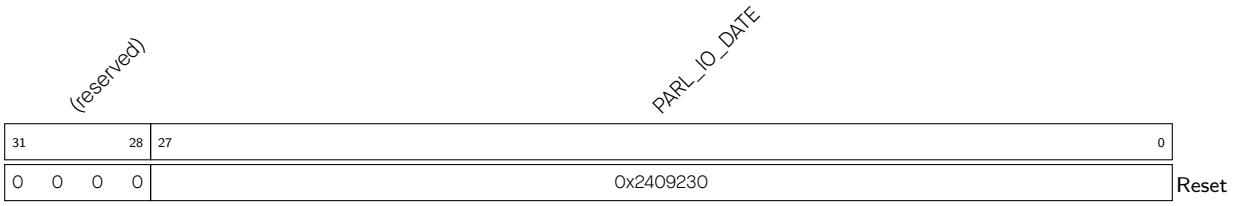
PARL_IO_CLK_EN 寄存器时钟门控。

0：只在读写寄存器时开启寄存器时钟

1：始终开启寄存器时钟

(R/W)

Register 43.22. PARL_IO_VERSION_REG (0x03FC)



PARL_IO_DATE 版本控制寄存器。(R/W)

第 44 章

比特调节器

ESP32-C5 中有大量支持 DMA（直接存储器访问）的外设（详情请参考章节 5 通用 DMA 控制器 (GDMA)），它们可以在 CPU 不参与的情况下将数据从存储器传输至外设或从外设传输至存储器，但这需要外设传输的数据格式与软件支持的数据格式相同，如果格式不同，则需要 CPU 重写数据格式，如交换字节、反转字节和左右移位数据。

由于位操作通常相当耗费 CPU 资源，而设计 DMA 的初衷是在传输过程中避免使用 CPU，因此 ESP32-C5 集成了一个比特调节器 (BitScrambler)，专门用于修改存储器和外设之间数据传输的格式，支持的传输方向包括存储器到外设、存储器到存储器、外设到存储器。除此之外，比特调节器还是一个灵活的可编程状态机，能够执行更高级的操作。

44.1 概述

在 ESP32-C5 中，支持 DMA 的外设是指能够通过 DMA 通道直接从存储器读取数据和/或直接向存储器写入数据的外设。这些数据的读取/写入格式取决于外设，灵活性较低。对于 ESP32-C5 中大多数支持 DMA 的外设来说，可以在这些路径中挂载比特调节器来提高灵活性。

当某个外设挂载比特调节器后，比特调节器就可以修改与该外设连接的 DMA 通道上的任何数据。写入 DMA 通道的数据会出现在比特调节器的输入端，而比特调节器输出端的数据将沿着 DMA 通道继续传输。比特调节器内置指令存储器，存储的指令用于控制输入和输出数据格式的映射关系。因此，通过编写合适的比特调节器程序，可以让比特调节器改写源数据。

如图 44.1-1 所示，比特调节器可以放置在 DMA 的 TX 路径中，修改从存储器流向外设的数据；也可以放置在 DMA 的 RX 路径中，修改从外设流向存储器的数据；还可以配置为环回模式，用于修改从存储器流向存储器的数据。

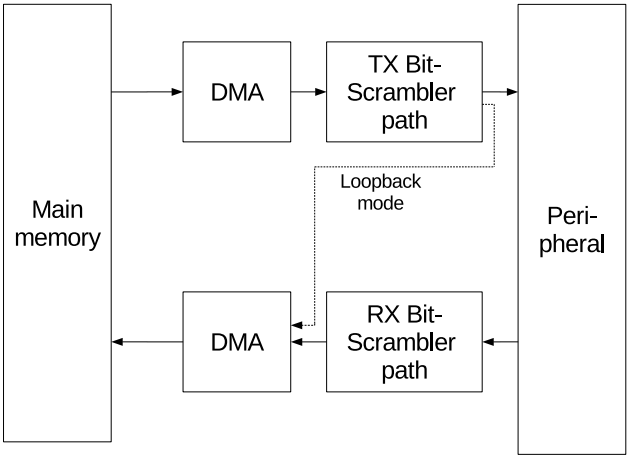


图 44.1-1. 比特调节器系统架构图

44.2 特性

比特调节器有以下特性：

- 一个比特调节器核心，可用于 RX 单元（外设到存储器）或 TX 单元（存储器到外设）
- 支持存储器到存储器的传输
- 每个 DMA 时钟周期最多可处理 32 位数据
- 数据路径由存储在指令存储器中的比特调节器程序控制
- 输入寄存器每个时钟周期可读取 0、8、16 或 32 位
- 输出寄存器：
  - 每个时钟周期可写入 0、8、16 或 32 位
  - 输出寄存器位的数据源：64 位输入数据、两个计数器、LUT RAM 数据、上个周期的数据输出、比较器
  - 32 位输出寄存器位中的每一位可以来自数据源的任意位
- 8 x 257 位指令存储器，用于存储八条指令，配置控制流和数据路径
- 2048 字节查找表（LUT）存储器，可配置为不同的字宽

44.3 应用示例

- 数据中字节、半字节或位顺序的交换
- 将基于调色板的位图转换为全 RGB 位图
- 数据的游程编码 (run-length encoding) 或解码
- 生成复杂波形，例如用于 WS2811 芯片

44.4 架构

比特调节器可以分为数据路径、控制路径以及寄存器，CPU 可通过这些寄存器配置数据路径和控制路径。数据路径在 DMA 输入流、内部寄存器和 DMA 输出流之间传输比特。控制路径解析指令存储器中的指令，以控制数据路径，并处理程序流程。

44.4.1 数据路径

比特调节器的数据路径从 DMA 输入流 (DMA incoming stream) 中获取数据，根据存储在指令存储器中的程序代码处理数据，然后将其写入 DMA 输出流 (DMA outgoing stream) 中。

1. 数据从 DMA 输入流保存到一个 64 位寄存器中。在每个指令周期结束时，根据指令的配置从 DMA 输入流中读取 N 位（N 可以是 0、8、16 或 32）。
2. 寄存器中的数据向寄存器的最低有效位（LSB）方向移动，最低有效位的 N 位消失。
3. 将最新读取的数据保存在最高的 N 位。

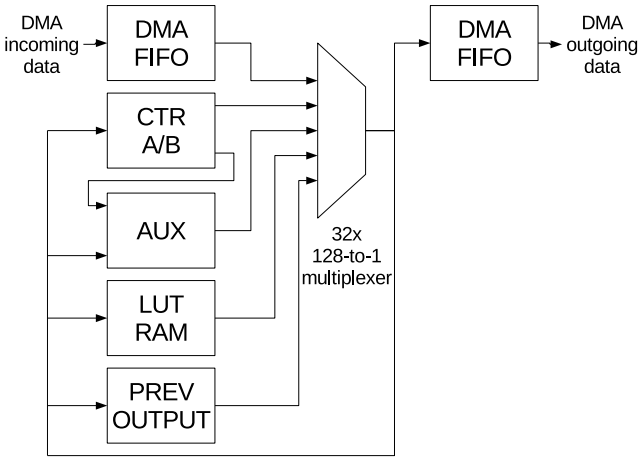


图 44.4-1. 比特调节器数据路径图

或者在比特调节器启动时，可以选择预先自动读取前 64 位到寄存器中。

在比特调节器的另一端，数据存入一个 32 位输出寄存器。在每个指令周期结束时，根据指令的配置将最低的 0、8、16 或 32 位写入 DMA 输出流。

比特调节器的名称来源于其可以将输入比特放置在输出中的任意位置，这是通过 32 个 128 选 1 多路复用器 (128-to-1 multiplexer) 来实现的。对于输出寄存器中的 32 位的每一位，都有一个多路复用器根据比特调节器控制路径的输出，从相应的数据源中选择输入信号。这些数据源包括：

- 从输入 FIFO 移入的 64 位数据（即图 44.4-1 中左侧的 DMA FIFO 模块）。如前所述，来自 DMA 输入流的数据被存储在此处。为了便于使用循环来遍历输入的 64 位数据，指令可以启用相对寻址。在这种寻址模式下，任何从输入 FIFO 寄存器获取比特的多路复用器实际上会获取偏移计数器 A 寄存器值的比特（即图 44.4-1 中的 CTR A）。
- 从上一个周期发送到输出的 32 位数据（即图 44.4-1 中的 PREV OUTPUT 模块）。发送到输出寄存器的数据将在一个时钟周期后出现在这个寄存器中。这使得比特调节器可以在多个时钟周期中“记住”数据：即使某个比特没有发送到输出 FIFO，只要是输出到了输出寄存器，下一个时钟周期仍然可以通过选择这个寄存器来访问该比特。通过再次从这个寄存器中选择比特进行输出，可以实现比特的长期记忆，使它们在随后的周期中出现在此寄存器中。
- 由 LUT RAM 输出的 8、16 或 32 位数据。当 LUT 宽度为 32 位、16 位、8 位时，则数据地址分别是上一周期输出数据的第 16 到 24、25、26 位。LUT RAM 是比特调节器的一部分，它首先接收到某个地址，然后在 LUT RAM 中查找该地址对应的数据，最后输出该数据。比特调节器初始化时可填充 LUT RAM，但比特调节器不能修改 LUT RAM 。可根据需求将 LUT RAM 配置成以下模式进行初始化：
  - 512 x 32 位字
  - 1024 x 16 位字
  - 2048 x 8 位字

  LUT RAM 可以用于数据的转换，例如用于某传感器的校准曲线，也可用于实现数学函数：将地址输入拆分成两个或两个以上的输入变量，数据输出作为函数的输出；存储器应加载每个输入值对应的函数输出。
- 2 x 16 位计数器提供的 32 位数据（即图 44.4-1 中的 CTR A/B 模块）。该寄存器包含两个 16 位计数器，分别命名为计数器 A 和 B，可通过控制路径指令对它们进行加载、递增和递减等操作。
- 计数器 B 和之前输出数据（对应图 44.4-1 中的 PREV OUT）之间的比较器中包含的 30 位数据，属于 AUX

模块的一部分比较结果可用于输出或程序流程，例如在解码游程编码流 (run-length encoded stream) 或生成 PWM 信号时非常有用。

- 2 位数据（属于图 44.4-1 中的 AUX 模块），一位固定为高电平，一位固定为低电平。使用场景：某协议需要输出一个始终为高电平或低电平的位，不受输入数据影响。

请注意，某些数据源无法同时访问。具体而言，指令无法同时从输入 FIFO 源寄存器的高 32 位以及计数器寄存器中获取数据。当从 LUT 读取数据时，对于宽度为 N 的 LUT，输入 FIFO 寄存器的最高 N 位和计数器寄存器的最高 N 位不可用。

44.4.2 控制路径

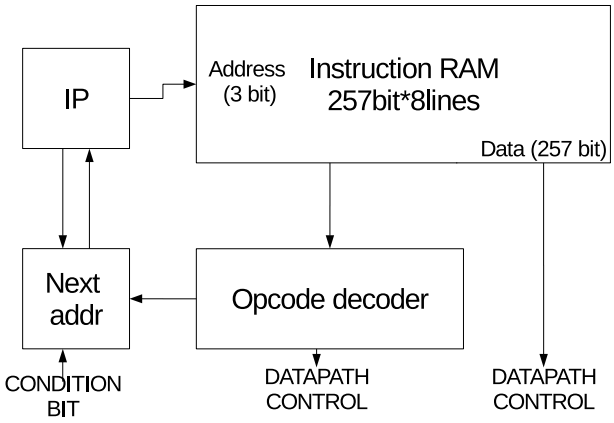


图 44.4-2. 比特调节器控制路径图

比特调节器的行为由存储在指令 RAM 中的程序控制。指令 RAM 最多可以存储 8 条指令，每条指令为 257 位。这些指令可配置该指令运行时钟周期内的数据路径，以及控制程序流程的操作码。

比特调节器的程序和微处理器的程序类似，每个周期都会执行指令存储器中的下一条指令，除非操作码明确指示需要跳转到另一个位置。具体来说，这种行为是使用指令指针（IP，参见图 44.4-2）来实现的，指向当前正在执行的指令。除此之外，操作码还可以影响计数器寄存器。

在配置数据路径的指令位中，一些位用于配置 32 个多路复用器，另一些位用于配置计数器位或 LUT 输出是否可用。指令还可以将多路复用器设置为相对模式，在这种模式下，从 DMA FIFO 获取的比特的位置将被偏移计数器 B 位。

44.5 功能描述

比特调节器是一种灵活的设备，其行为可通过加载到控制器内部的程序来控制。此外，一些配置寄存器也可以全局配置控制器的一些行为。因此，比特调节器程序包括 257 位 x 8 的指令 RAM 以及各种配置寄存器的设置，下文将对二者进行介绍。

44.5.1 指令

表 44.5-1 为 257 位指令的结构。

表 44.5-1. 指令结构

位	域	位	域
0 - 6	CTL_MUX_SEL_0	133 - 139	CTL_MUX_SEL_19
7 - 13	CTL_MUX_SEL_1	140 - 146	CTL_MUX_SEL_20
14 - 20	CTL_MUX_SEL_2	147 - 153	CTL_MUX_SEL_21
21 - 27	CTL_MUX_SEL_3	154 - 160	CTL_MUX_SEL_22
28 - 34	CTL_MUX_SEL_4	161 - 167	CTL_MUX_SEL_23
35 - 41	CTL_MUX_SEL_5	168 - 174	CTL_MUX_SEL_24
42 - 48	CTL_MUX_SEL_6	175 - 181	CTL_MUX_SEL_25
49 - 55	CTL_MUX_SEL_7	182 - 188	CTL_MUX_SEL_26
56 - 62	CTL_MUX_SEL_8	189 - 195	CTL_MUX_SEL_27
63 - 69	CTL_MUX_SEL_9	196 - 202	CTL_MUX_SEL_28
70 - 76	CTL_MUX_SEL_10	203 - 209	CTL_MUX_SEL_29
77 - 83	CTL_MUX_SEL_11	210 - 216	CTL_MUX_SEL_30
84 - 90	CTL_MUX_SEL_12	217 - 223	CTL_MUX_SEL_31
91 - 97	CTL_MUX_SEL_13	224 - 249	OPCODE
98 - 104	CTL_MUX_SEL_14	250 - 251	CTL_READ_IN
105 - 111	CTL_MUX_SEL_15	252 - 253	CTL_WR_OUT
112 - 118	CTL_MUX_SEL_16	254	CTL_MUX_REL
119 - 125	CTL_MUX_SEL_17	255	CTL_CTR_SRC_SEL
126 - 132	CTL_MUX_SEL_18	256	CTL_LUT_SRC_SEL

下文介绍指令各域的含义。

- CTL_MUX_SEL_n：包含 7 位，选择输出寄存器中第 n 位的来源。详情参见表 44.5-3。
- OPCODE：操作码，该域的位格式请参见表 44.5-4。
- CTL_READ_IN：在指令结束时选择从输入 FIFO 读取的位数。
  - 0：不读取任何数据
  - 1：读取 8 位数据
  - 2：读取 16 位数据
  - 3：读取 32 位数据
- CTL_WR_OUT：选择从输出寄存器写入到输出 FIFO 的位数：
  - 0：不写入任何数据
  - 1：写入 8 位数据
  - 2：写入 16 位数据
  - 3：写入 32 位数据
- CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL：配置哪些位可以通过 CTL_MUX_SEL 域来选择。
  - CTRL_CTR_SRC_SEL（控制计数器源选择）配置计数器 A 和 B 的位是否可用。

- CTL_LUT_SRC_SEL（控制 LUT 源选择）配置 LUT RAM 的输出值是否可用。详见表 44.5-3。
- 当 CTL_MUX_REL（控制多路复用器相对寻址）为 1 时，且 CTL_MUX_SEL_n 小于等于 63 时，比特调节器将使用计数器 A 重新计算有效的 CTL_MUX_SEL_n。具体行为取决于 CTR_SRC_SEL，详见表 44.5-2。

表 44.5-2. 当 CTL_MUX_REL 为 1 时 CTL_MUX_SEL_n 生效的值

条件	CTL_MUX_SEL 生效的值
CTL_MUX_SEL_n >= 64	CTL_MUX_SEL（不变）
CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 0	(CTL_MUX_SEL_n + CTRA) & 63
CTL_MUX_SEL_n < 31 and CTL_CTR_SRC_SEL = 1	(CTL_MUX_SEL_n + CTRA) & 31
32 <= CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 1	((CTL_MUX_SEL_n + CTRA) & 31) + 32

这些域中，大部分是数据路径配置项，其中 OPCODE 是影响控制路径的配置项，但 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 可通过调整 IF/IFN 操作码的来源来影响某些操作码的工作方式。

CTL_MUX_SEL_n: 每个 CTL_MUX_SEL_n 域都可用于选择输出寄存器中与之相对应的位的来源。具体选择哪个位在一定程度上取决于 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置。假设 CTL_MUX_SRC 的值为 N，选择该位的方式如表 44.5-3 所示。请注意，IF/IFN 操作码的数据源选择也采用相同的方式。

表 44.5-3. CTL_MUX_SEL 的值

位	描述
0-31	该位来自输入 FIFO 寄存器中的第 N 位。
32-63	该位的来源取决于 CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置 如果 CTL_CTR_SRC_SEL = 0，该位来源于输入 FIFO 寄存器中的第 N 位。 如果 CTL_CTR_SRC_SEL = 1，该位来源于计数器寄存器中的第 N-32 位。具体来说，如果 N 是 32 到 47，则该位来源于计数器 A 的第 N-32 位；如果 N 是 48 到 63，则该位来源于计数器 B 的第 N-48 位。 如果 CTL_LUT_SRC_SEL = 1，根据 LUT 的宽度（8、16 或 32 位），读取最高 8、16 或 32 位将返回来自 LUT 的数据，而不是 CTL_CTR_SRC_SEL 选择的数据。具体来说，假定 LUT 宽度为 N，63 将返回 LUT 输出的第 N-1 位，62 将返回 LUT 输出的第 N-2 位，依此类推。
64	CounterB[7:0] =< last[7:0]
65	CounterB[7:0] > last[7:0]
66	CounterB[7:0] = last[7:0]
67	CounterB[7:0] =< last[15:8]
68	CounterB[7:0] > last[15:8]
69	CounterB[7:0] = last[15:8]
70	CounterB[7:0] =< last[23:16]
71	CounterB[7:0] > last[23:16]
72	CounterB[7:0] = last[23:16]
73	CounterB[7:0] =< last[31:24]
74	CounterB[7:0] > last[31:24]
75	CounterB[7:0] = last[31:24]
76	CounterB[15:8] =< last[7:0]
77	CounterB[15:8] > last[7:0]
78	CounterB[15:8] = last[7:0]
79	CounterB[15:8] =< last[15:8]
80	CounterB[15:8] > last[15:8]
81	CounterB[15:8] = last[15:8]
82	CounterB[15:8] =< last[23:16]
83	CounterB[15:8] > last[23:16]
84	CounterB[15:8] = last[23:16]
85	CounterB[15:8] =< last[31:24]
86	CounterB[15:8] > last[31:24]
87	CounterB[15:8] = last[31:24]
88	CounterB[15:0] =< last[15:0]
89	CounterB[15:0] > last[15:0]
90	CounterB[15:0] = last[15:0]
91	CounterB[15:0] =< last[31:16]
92	CounterB[15:0] > last[31:16]
93	CounterB[15:0] = last[31:16]
94	始终为 0
95	始终为 1
96-127	这些位与上一周期结束时输出寄存器上的位相同

比特调节器可识别七种操作码，编码格式如表 44.5-4 所示。

表 44.5-4. 指令操作码

位 / 操作码	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOOP	1	c	tgt			end_val																ctr_add				
ADD	0	c	h	l	0	0	0	0	0	0	ctr_add															
IF	0	0	tgt			0	0	0	0	1	0	0	0	0	0	0	0	0	0	src						
IFN	0	0	tgt			0	0	0	1	0	0	0	0	0	0	0	0	0	0	src						
LDCTD	0	c	h	l	0	0	0	0	1	1	ctr_set															
LDCTI	0	c	h	l	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADDCTI	0	c	h	l	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

操作码域的定义如下：

- c：配置使用哪个计数器。
  - 0：计数器 A
  - 1：计数器 B
- end_val, ctr_add, ctr_set：16 位（或 5 位）值。这些域的含义请参阅下面的指令描述。
- tgt：指令的位置。范围：0 到 7。
- h：如果需要将高 8 位写回计数器，则为 1（如果操作码以 H 结尾）。
- l：如果需要将低 8 位写回计数器，则为 1（如果操作码以 L 结尾）。

请注意，如果操作码没有指定 H 或 L，则默认需要将 16 位全部写回，此时 h 和 l 位都会为 1。

下方表格主要介绍这七个操作码如何影响比特调节器的状态。

表 44.5-5. LOOP 操作码

`LOOPc end_val, ctr_add, tgt`	
使用 c 参数设置的计数器进行循环，将 ctr_add 加到计数器中，并循环回到 tgt 位置，直到计数器达到 end_val。当计数器达到 end_val 时，重置计数器并继续执行。	
参数	c：计数器 A 或 B，表示要使用的计数器 tgt：3 位值，表示目标指令地址 end_val：16 位值，用于与计数器进行比较 ctr_add：5 位有符号值，用于加到计数器上

伪代码	`if ((unsigned)ctrC < end_val) begin`
	`  IP <= tgt`
	`  ctr_c <= ctr_c + sign_extend(ctr_add)`
	`end else begin`
	`  IP <= IP + 1`
	`  ctr_c <= 0`
	`end`

表 44.5-6. ADD 操作码

ADDc[H|L] ctr_add	
将立即数加到计数器上，可以选择只写回高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 ctr_add：16 位值，加到计数器上的值 如果操作码是 ADDcH，则只写回最高 8 位 如果操作码是 ADDcL，则只写回最低 8 位 如果操作码是 ADDc，则 16 位全部都将被写回（h=1，l=1）

伪代码	`tmp <= ctr_c + ctr_add`
	`if (h) begin`
	`  ctr_c[15:8] <= tmp[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= tmp[7:0]`
	`end`
	`IP <= IP + 1`

说明	NOP（无操作）伪操作码为 “ADDA 0”。

表 44.5-7. IF 操作码

IF ctl_cond_src, tgt	
如果 src 域对应的多路复用器值为 1，则跳转到目标。	
参数	ctl_cond_src：多路复用器来源，可参考表 CTR_MUX_SRC 来源 tgt：3 位值，表示目标指令地址

伪代码	`if (ctl_cond_src) begin`
	`  IP <= tgt`
	`end else begin`
	`  IP <= IP + 1`
	`end`

说明	JMP（始终跳转）伪操作码为 “IF 95,tgt”，使用 AUX 位中始终为 1 的位。

表 44.5-8. IFN 操作码

IFN ctl_cond_src, tgt	
如果指定的多路复用器源位未设置，则跳转到目标（注：与 IF 操作相同，但条件取反）。	
参数	ctl_cond_src：多路复用器来源，可参考表 CTR_MUX_SRC sources tgt：3 位值，表示目标指令地址

伪代码	`if (ctl_cond_src) begin`
	`  IP <= IP + 1`
	`end else begin`
	`  IP <= tgt`
	`end`

表 44.5-9. LDCTD 操作码

LDCTDc[H|L] ctr_set	
将立即数直接加载到计数器，可以选择只加载计数器的高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 ctr_set：16 位值，要设置的计数器的值 h：如果操作码是 LDCTDcH，则只设置最高 8 位 l：如果操作码是 LDCTDcL，则只设置最低 8 位 如果操作码是 LDCTDc，则 16 位全部都将被设置（h=1，l=1）

伪代码	`if (h) begin`
	`  ctr_c[15:8] <= ctr_set[15:8]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= ctr_set[7:0]`
	`end`
	`IP <= IP + 1`

表 44.5-10. LDCTI 操作码

LDCTIc[H|L]	
将多路复用器输出的高 16 位间接加载到计数器，可以选择只加载到计数器的高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 h：如果操作码是 LDCTIcH，则只设置最高 8 位 l：如果操作码是 LDCTIcL，则只设置最低 8 位 如果操作码是 LDCTIc，则 16 位全部都将被设置（h=1，l=1）

伪代码	`if (h) begin`
	`  ctr_c[15:8] <= mux_out[31:24]`
	`end`
	`if (l) begin`
	`  ctr_c[7:0] <= mux_out[23:16]`
	`end`
	`IP <= IP + 1`

表 44.5-11. ADDCTI Opcode

ADDCTIc[H|L]	
将多路复用器输出的高 16 位间接加到计数器，可以选择只加到计数器的高 8 位或低 8 位。	
参数	c：计数器 A 或 B，表示要使用的计数器 h：如果操作码是 ADDCTIcH，将多路复用器输出的 24–31 位加到计数器的高 8 位 l：如果操作码是 ADDCTIcL，将多路复用器输出的 16–23 位加到计数器的低 8 位 如果操作码是 ADDCTIc，将多路复用器输出的 16–31 位加到计数器的全部 16 位 (h=1, l=1)

伪代码	`if (h) begin`
	`  ctr_c[15:8] <= ctr_c[15:8] + mux_out[31:24]`
	`end else if (l) begin`
	`  ctr_c[7:0] <= ctr_c[7:0] + mux_out[23:16]`
	`end else begin`
	`  ctr_c[15:0] <= ctr_c[15:0] + mux_out[31:16]`
	`end`
	`IP <= IP + 1`

44.5.2 配置寄存器

比特调节器的配置是通过一组寄存器实现的。这些寄存器可向比特调节器写入程序、写入 LUT RAM、配置控制器行为、以及启动和停止控制器的操作，还可将控制器挂载到外设的 DMA 路径。

ESP32-C5 只有一个比特调节器核心，可放置在 TX 路径（数据从存储器发送到外设）或 RX 路径（数据从外设发送到存储器）。为了兼容其他 ESP 系列芯片，当比特调节器核心处于 TX 路径时，则由 BITSCRAMBLER_TX_ 为前缀的寄存器控制；处于 RX 路径时，则由 BITSCRAMBLER_RX_ 为前缀的寄存器控制。哪个路径处于激活状态由 BITSCRAMBLER_RX_ENA 和 BITSCRAMBLER_TX_ENA 决定，且在任意时刻只允许其中一个处于激活状态。

说明：
为方便阐述，本节仅使用 TX 路径寄存器进行描述，若需 RX 路径寄存器的说明，只需替换描述中的前缀即可。

当比特调节器修改某外设的输入或输出时，需要挂载到该外设的 DMA 路径。为此，需将该外设的对应值写入 HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG 寄存器的 HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 域或 HP_SYSTEM_BITSCRAMBLER_PERI_RX_SEL 域。

比特调节器也可以在存储器到存储器模式下运行，但需要将 BITSCRAMBLER_SYS_REG 寄存器中的 BITSCRAMBLER_LOOP_MODE 域置为 1。该模式下，TX 比特调节器进入环回模式，而 RX 比特调节器则未使用且无法使用。并且比特调节器仍然需要连接到外设，但该外设无需配置，而且对应的外设 DMA 路径将无法使用。

主处理器无法直接访问比特调节器的程序存储器和 LUT RAM，但可以间接访问，即在一个寄存器中设置地址，然后在另一个寄存器中写入数据。

具体而言，对于 LUT RAM，地址写入 BITSCRAMBLER_TX_LUT_CFG0_REG 的 BITSCRAMBLER_TX_LUT_IDX 域，数据写入 BITSCRAMBLER_TX_LUT_CFG1_REG 寄存器，或从该寄存器读取数据。LUT 的位宽对比特调节器可见的，可以使用 BITSCRAMBLER_TX_LUT_MODE 域进行配置。需要注意的是，当主处理器以上述方式写入 LUT 时，地址和数据大小需要符合配置的 LUT 位宽。

同样，对于指令存储器，地址写入 BITSCRAMBLER_TX_INST_CFG0_REG，数据写入或从 BITSCRAMBLER_TX_INST_CFG1_REG 中读取。由于比特调节器指令为 257 位，因此一个指令需要写入 9 个 32 位字，第 257 位位于第 9 个字的最低有效位。为此，BITSCRAMBLER_TX_INST_CFG0_REG 寄存器被分为两个域，指令的位置写入 BITSCRAMBLER_TX_INST_IDX 域，而指令内的字偏移写入 BITSCRAMBLER_TX_INST_POS 域。

由于比特调节器生成的数据可以比输入多也可以比输入少，所以它控制的 DMA 流可以以一种或两种方式结束。第一种情况是接收器（在 RX 比特调节器的情况下是存储器，而在 TX 比特调节器的情况下是外设）停止接收数据，因为它的配置是只接收一定数量的字节。这种情况比较简单，比特调节器因为无法再写入数据而停止。另一种方式是发射器（在 RX 比特调节器的情况下是外设，在 TX 比特调节器的情况下是存储器）停止发送数据。这种情况更加复杂，因为比特调节器可能正在处理需要写入输出的一些数据，也可能不在处理。

ESP32-C5 比特调节器设计了两种方法来处理输入数据流结束后的一定数量的数据，目的是为了完整地处理这些数据。这两种方法依赖于 BITSCRAMBLER_RX_TAILING_BITS_REG 寄存器设置的具体值 N：

- EOF-on-N-reads（读取 N 字节后结束）：在输入数据流结束后，比特调节器从输入读取 N 个虚拟字节。这些字节的值为零。在读取完第 N 个字节后，比特调节器停止，DMA 流结束。
- EOF-on-N-writes（写入 N 字节后结束）：在输入数据流结束后，比特调节器允许向输出写入 N 个字节。在此期间，任何输入流上的读取都会返回值为零的虚拟字节。在写入完第 N 个字节后，比特调节器停止，DMA 流结束。

请按照下表内容配置上述两种模式。

表 44.5-12. EOF 模式设置

寄存器	EOF-on-N-reads	EOF-on-N-writes
BITSCRAMBLER_TX_RD_DUMMY	1	1
BITSCRAMBLER_TX_FETCH_MODE	0	0
BITSCRAMBLER_TX_EOF_MODE	1	0
BITSCRAMBLER_TX_HALT_MODE	1	1

44.6 配置流程

请注意，下文描述的是 TX 路径使能时比特调节器的配置流程，路径方向为存储器到外设方向或者处于环回模式下存储器到存储器方向。对于外设到存储器方向的路径，则使用比特调节器的 RX 路径。除非另作说明，否则将下文描述中的 TX 替换为 RX 即为比特调节器 RX 路径的配置流程。

1. 配置 HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG 寄存器中的 HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 域将比特调节器挂载到外设。请注意，即使在环回模式下，此步骤也是必须的。
2. 配置 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_ENA 域使能比特调节器。请确保任意时刻 BITSCRAMBLER_TX_ENA 和 BITSCRAMBLER_RX_ENA 中只能有一个寄存器的值为 1。
3. 根据需要决定是否配置环回模式。如果需要，将 BITSCRAMBLER_SYS_REG 寄存器的 BITSCRAMBLER_LOOP_MODE 域置 1，否则清零。
4. 将程序加载到比特调节器中。对于每条指令和指令内的每个 32 位字，设置 BITSCRAMBLER_TX_INST_CFG0_REG 寄存器中的 BITSCRAMBLER_TX_INST_IDX 和

BITSCRAMBLER_TX_INST_POS 域，然后将字写入 BITSCRAMBLER_TX_INST_CFG1_REG 寄存器。注意，第 257 位写入到 BITSCRAMBLER_TX_INST_CFG1_REG 的第 0 位。

5. 如果需要，将 LUT 挂载到比特调节器中：
   (a) 配置 BITSCRAMBLER_TX_LUT_CFG0_REG 寄存器中 BITSCRAMBLER_TX_LUT_MODE 域来设置 LUT 宽度
   (b) 将地址写入 BITSCRAMBLER_TX_LUT_CFG0_REG
   (c) 将数据字或字节写入 BITSCRAMBLER_TX_LUT_CFG1_REG

   请注意，在写入数据之后和启动比特调节器之前更改 LUT 宽度都是可以的。
6. 配置比特调节器。根据比特调节器程序的需求，设置或清除 BITSCRAMBLER_TX_CTRL_REG 寄存器中的相应位，也可将 BITSCRAMBLER_TX_TAILING_BITS_REG 设置为适当的值。
7. 清除 BITSCRAMBLER_TX_CTRL_REG 寄存器的BITSCRAMBLER_TX_HALT 域，以启动比特调节器。
8. 启动 DMA 传输。有关 DMA 子系统的更多信息，请参阅章节 5 *通用 DMA 控制器 (GDMA)*，如果未使用环回模式，请参阅各外设章节。
9. 等待 DMA 传输完成。
10. 设置 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_HALT 域停止比特调节器。
11. 等待比特调节器处于停止状态。当 BITSCRAMBLER_TX_STATE_REG 寄存器中的 BITSCRAMBLER_TX_IN_IDLE 位被置位时，比特调节器停止。
12. 在 BITSCRAMBLER_TX_CTRL_REG 寄存器中的 BITSCRAMBLER_TX_FIFO_RST 位先写入 1 然后再写入 0 来复位 FIFO。
13. 比特调节器可以使用当前的程序和 LUT 配置开始进行新的传输，从步骤 7 开始即可。

44.7 寄存器列表

本小节的所有地址均为相对于比特调节器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

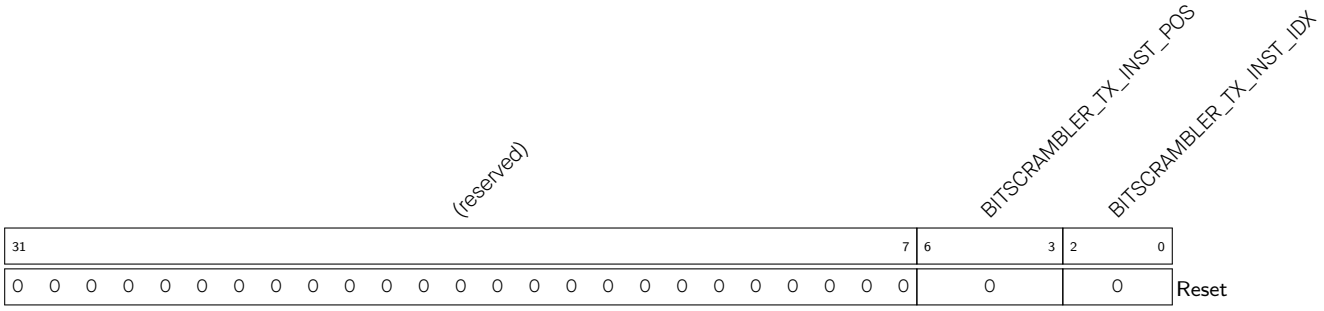
名称	描述	地址	访问
控制和配置寄存器			
BITSCRAMBLER_TX_INST_CFG0_REG	比特调节器 TX 路径指令存储器地址选择寄存器	0x0000	R/W
BITSCRAMBLER_TX_INST_CFG1_REG	比特调节器 TX 路径指令存储器访问寄存器	0x0004	R/W
BITSCRAMBLER_RX_INST_CFG0_REG	比特调节器 RX 路径指令存储器地址选择寄存器	0x0008	R/W
BITSCRAMBLER_RX_INST_CFG1_REG	比特调节器 RX 路径指令存储器访问寄存器	0x000C	R/W
BITSCRAMBLER_TX_LUT_CFG0_REG	比特调节器 TX 路径 LUT 存储器地址选择寄存器	0x0010	R/W
BITSCRAMBLER_TX_LUT_CFG1_REG	比特调节器 TX 路径 LUT 存储器访问寄存器	0x0014	R/W
BITSCRAMBLER_RX_LUT_CFG0_REG	比特调节器 RX 路径 LUT 存储器地址选择寄存器	0x0018	R/W
BITSCRAMBLER_RX_LUT_CFG1_REG	比特调节器 RX 路径 LUT 存储器访问寄存器	0x001C	R/W
配置寄存器			
BITSCRAMBLER_TX_TAILING_BITS_REG	比特调节器 TX 路径额外数据长度寄存器	0x0020	R/W
BITSCRAMBLER_RX_TAILING_BITS_REG	比特调节器 RX 路径额外数据长度寄存器	0x0024	R/W
BITSCRAMBLER_TX_CTRL_REG	比特调节器 TX 路径控制寄存器	0x0028	varies
BITSCRAMBLER_RX_CTRL_REG	比特调节器 RX 路径控制寄存器	0x002C	varies
BITSCRAMBLER_SYS_REG	环回控制寄存器	0x00F8	R/W
状态寄存器			
BITSCRAMBLER_TX_STATE_REG	比特调节器 TX 路径状态寄存器	0x0030	varies
BITSCRAMBLER_RX_STATE_REG	比特调节器 RX 路径状态寄存器	0x0034	varies
版本寄存器			
BITSCRAMBLER_VERSION_REG	版本控制寄存器	0x00FC	R/W

44.8 寄存器

本小节的所有地址均为相对于比特调节器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

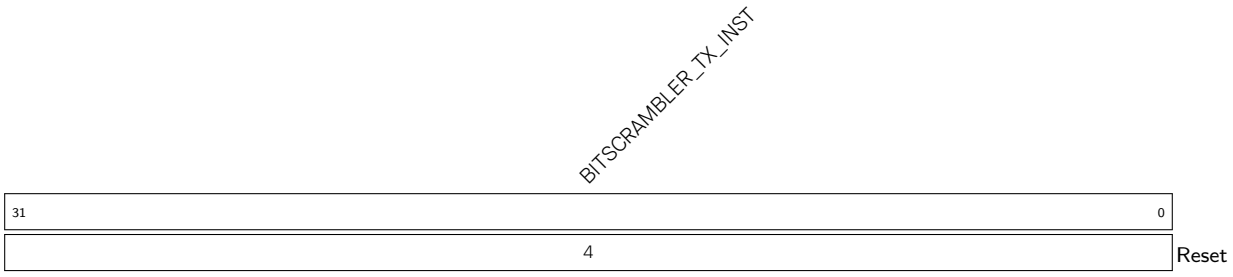
Register 44.1. BITSCRAMBLER_TX_INST_CFG0_REG (0x0000)



BITSCRAMBLER_TX_INST_IDX 配置比特调节器指令索引（通过 BITSCRAMBLER_TX_INST_CFG1_REG 访问指令）。(R/W)

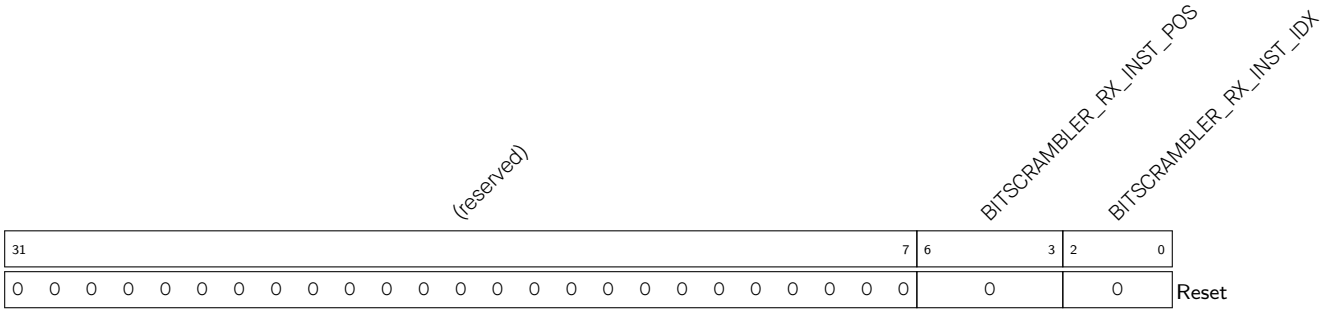
BITSCRAMBLER_TX_INST_POS 配置比特调节器 257 位指令的偏移量（以 32 位增量，通过 BITSCRAMBLER_TX_INST_CFG1_REG 访问指令）。(R/W)

Register 44.2. BITSCRAMBLER_TX_INST_CFG1_REG (0x0004)



BITSCRAMBLER_TX_INST 配置在 BITSCRAMBLER_TX_INST_CFG0_REG 指定的地址处访问的指令。(R/W)

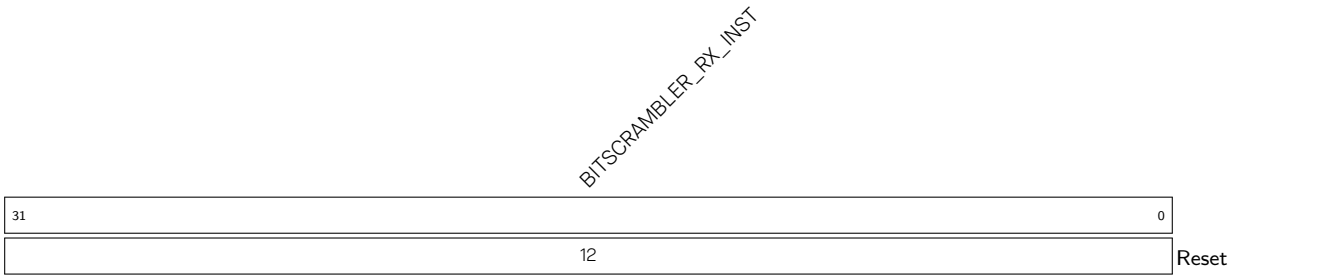
Register 44.3. BITSCRAMBLER_RX_INST_CFG0_REG (0x0008)



BITSCRAMBLER_RX_INST_IDX 配置比特调节器指令索引（通过 BITSCRAMBLER_RX_INST_CFG1_REG 访问指令）。(R/W)

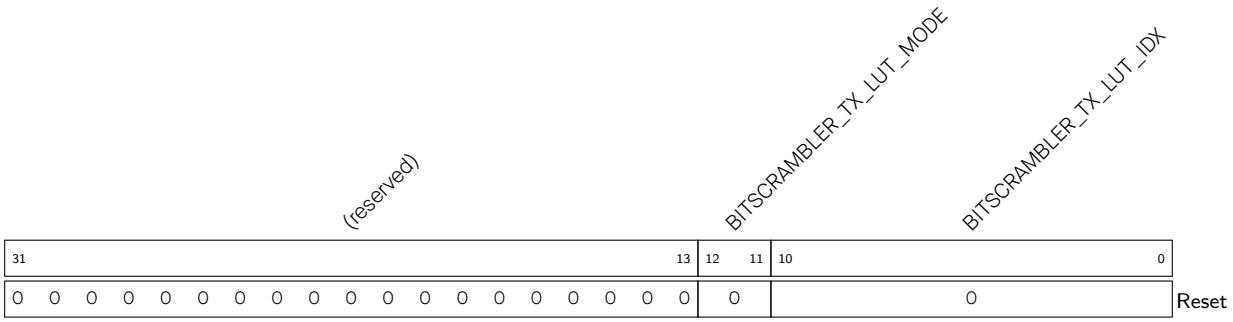
BITSCRAMBLER_RX_INST_POS 配置比特调节器 257 位指令的偏移量（以 32 位为增量，通过 BITSCRAMBLER_RX_INST_CFG1_REG 访问指令）。(R/W)

Register 44.4. BITSCRAMBLER_RX_INST_CFG1_REG (0x000C)



BITSCRAMBLER_RX_INST 配置在 BITSCRAMBLER_RX_INST_CFG0_REG 指定的地址处访问的指令。(R/W)

Register 44.5. BITSCRAMBLER_TX_LUT_CFG0_REG (0x0010)

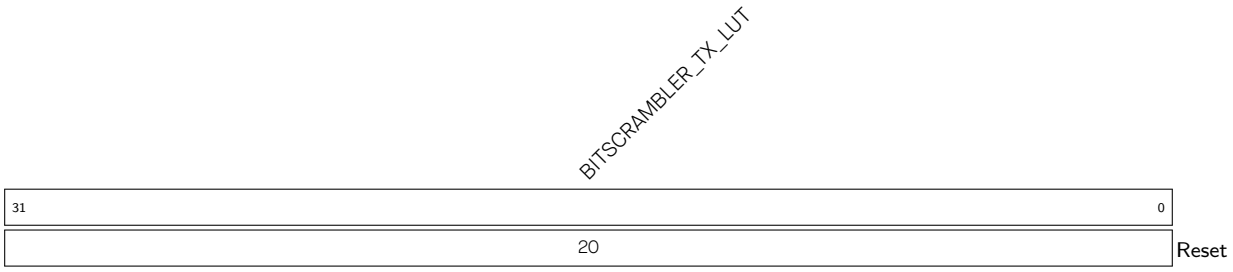


BITSCRAMBLER_TX_LUT_IDX 配置访问 LUT RAM 的位置。单位：BITSCRAMBLER_TX_LUT_MODE 配置的字大小。(R/W)

BITSCRAMBLER_TX_LUT_MODE 为比特调节器程序配置 LUT RAM 的字大小。

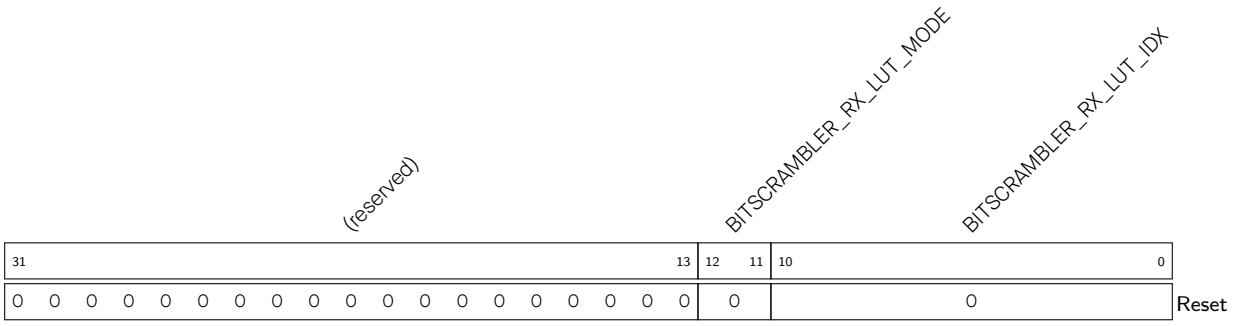
0：1 字节
1：2 字节
2：4 字节
3：保留
(R/W)

Register 44.6. BITSCRAMBLER_TX_LUT_CFG1_REG (0x0014)



BITSCRAMBLER_TX_LUT 配置在 BITSCRAMBLER_TX_LUT_CFG0_REG 指定的位置访问的 LUT 表项。(R/W)

Register 44.7. BITSCRAMBLER_RX_LUT_CFG0_REG (0x0018)



BITSCRAMBLER_RX_LUT_IDX 配置访问 LUT RAM 的位置。单位：BITSCRAMBLER_RX_LUT_MODE 配置的字大小。(R/W)

BITSCRAMBLER_RX_LUT_MODE 为比特调节器程序配置 LUT RAM 的字大小。

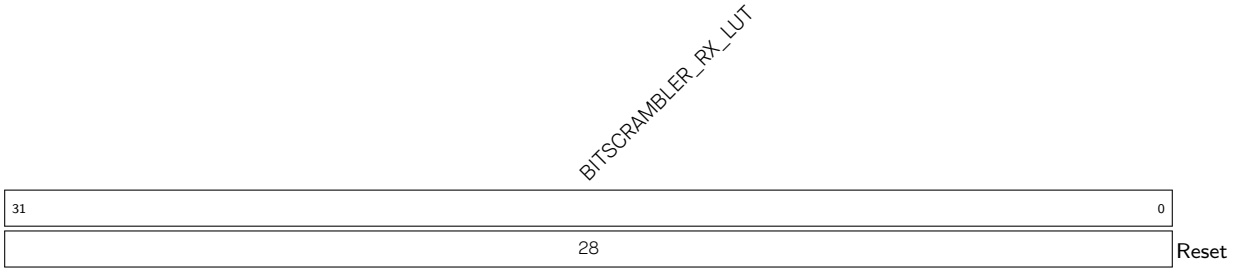
0：1 字节

1：2 字节

2：4 字节

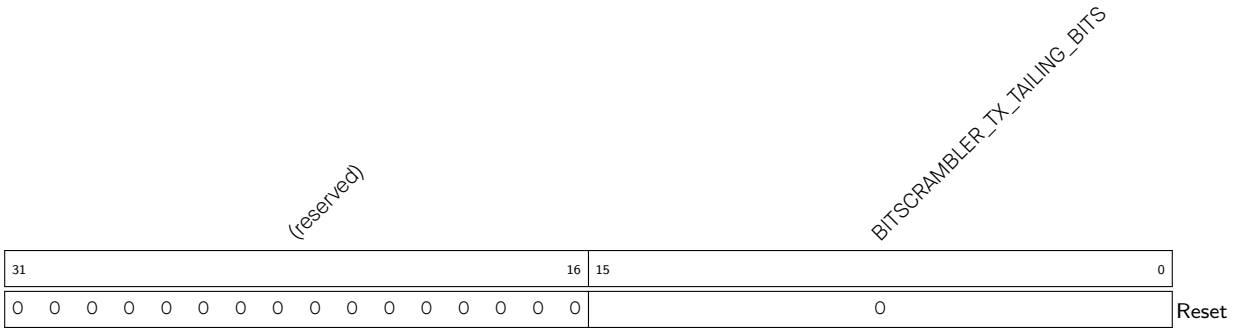
3：保留

Register 44.8. BITSCRAMBLER_RX_LUT_CFG1_REG (0x001C)



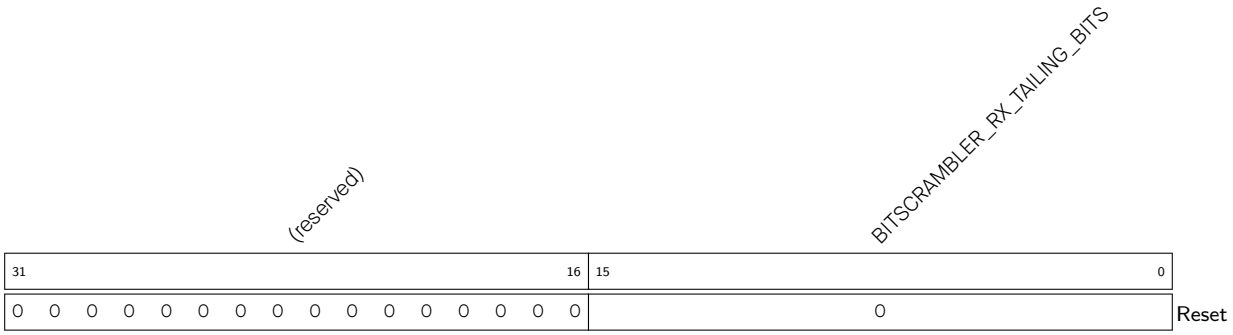
BITSCRAMBLER_RX_LUT 配置在 BITSCRAMBLER_RX_LUT_CFG0_REG 指定的位置访问的 LUT 表项。(R/W)

Register 44.9. BITSCRAMBLER_TX_TAILING_BITS_REG (0x0020)



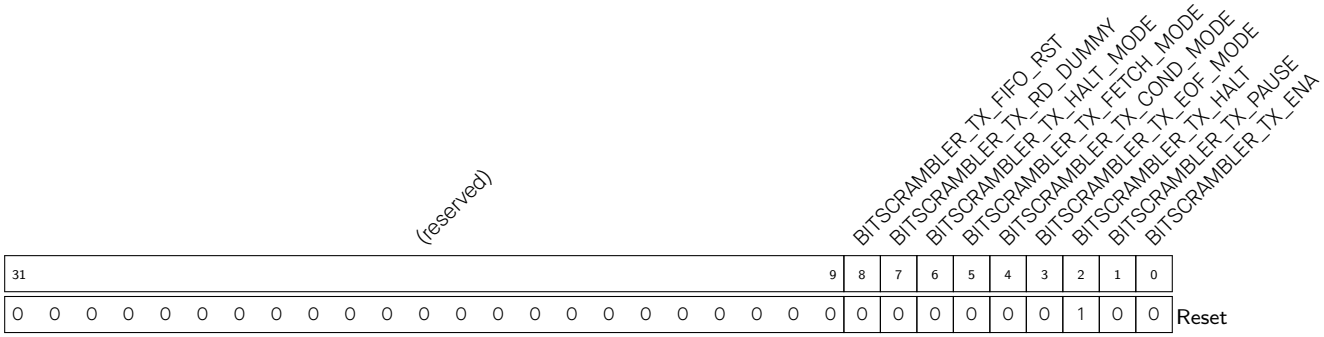
BITSCRAMBLER_TX_TAILING_BITS 配置比特调节器 TX 路径收到 EOF 后仍可处理额外数据的长度。单位：比特。(R/W)

Register 44.10. BITSCRAMBLER_RX_TAILING_BITS_REG (0x0024)



BITSCRAMBLER_RX_TAILING_BITS 配置比特调节器 RX 路径收到 EOF 后仍可处理额外数据的长度。单位：比特。(R/W)

Register 44.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)



BITSCRAMBLER_TX_ENA 配置是否在 TX 路径上启用比特调节器。

0：禁用

1：启用

注意，比特调节器核心只能在 RX 或 TX 路径中的一个路径中启用，不能同时启用。(R/W)

BITSCRAMBLER_TX_PAUSE 配置是否暂停比特调节器 TX 路径。暂停的路径可取消暂停并恢复执行。

0：不暂停

1：暂停

(R/W)

BITSCRAMBLER_TX_HALT 配置是否停止比特调节器 TX 路径。停止的 TX 路径将刷新 FIFO 并且无法恢复，需要重置 FIFO 并重新启动。

0：不停止

1：停止

(R/W)

BITSCRAMBLER_TX_EOF_MODE 配置比特调节器 TX 路径 EOF 信号生成模式，与 BITSCRAMBLER_TX_TAILING_BITS 结合使用。

0：对写入输出 FIFO 的数据进行计数，生成延迟的 EOF

1：对从输入 FIFO 读取的数据进行计数，生成延迟的 EOF

(R/W)

BITSCRAMBLER_TX_COND_MODE 配置比特调节器 TX LOOP 指令条件模式。

0：使用小于运算符获取条件

1：使用不等于运算符获取条件

(R/W)

BITSCRAMBLER_TX_FETCH_MODE 配置比特调节器 TX 路径获取指令模式。

0：通过复位预取

1：通过指令获取

(R/W)

BITSCRAMBLER_TX_HALT_MODE 当 BITSCRAMBLER_TX_HALT 置 1 时，配置比特调节器 TX 路径暂停模式。

0：等待写回数据完成

1：忽略回写数据

(R/W)

见下页……

Register 44.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)

接上页……

BITSCRAMBLER_TX_RD_DUMMY 当接收到 EOF 时，配置比特调节器 TX 路径读取数据模式。

0：等待读取数据

1：忽略读取数据

(R/W)

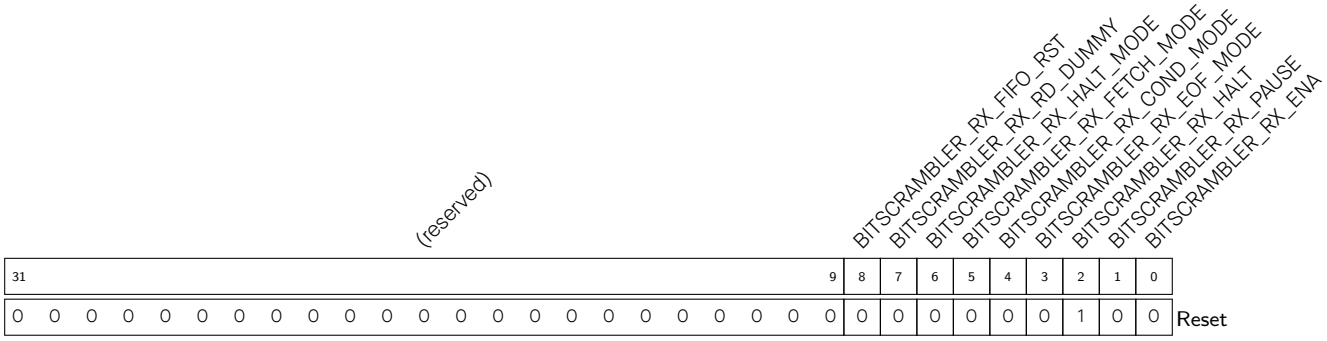
BITSCRAMBLER_TX_FIFO_RST 配置是否重置比特调节器 TX FIFO。

0：不复位

1：复位

(WT)

Register 44.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)



BITSCRAMBLER_RX_ENA 配置是否在 RX 路径上启用比特调节器。

0：禁用

1：启用

注意，比特调节器核心只能在 RX 或 TX 路径中的一个路径中启用，不能同时启用。(R/W)

BITSCRAMBLER_RX_PAUSE 配置是否暂停比特调节器 RX 路径。暂停的路径可取消暂停并恢复执行。

0：不暂停

1：暂停

(R/W)

BITSCRAMBLER_RX_HALT 配置是否停止比特调节器 RX 路径。停止的 RX 路径将刷新 FIFO 并且无法恢复，需要重置 FIFO 并重新启动。

0：不停止

1：停止

(R/W)

BITSCRAMBLER_RX_EOF_MODE 配置比特调节器 RX 路径 EOF 信号生成模式，与 BITSCRAMBLER_RX_TAILING_BITS 结合使用。

0：对写入输出 FIFO 的数据进行计数，生成延迟的 EOF

1：对从输入 FIFO 读取的数据进行计数，生成延迟的 EOF

(R/W)

BITSCRAMBLER_RX_COND_MODE 配置比特调节器 RX LOOP 指令条件模式。

0：使用小于运算符获取条件

1：使用不等于运算符获取条件

(R/W)

BITSCRAMBLER_RX_FETCH_MODE 配置比特调节器 RX 路径获取指令模式。

0：通过复位预取

1：通过指令获取

(R/W)

BITSCRAMBLER_RX_HALT_MODE 当 BITSCRAMBLER_RX_HALT 置 1 时，配置比特调节器 RX 路径暂停模式。

0：等待写回数据完成

1：忽略回写数据

(R/W)

见下页……

Register 44.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)

接上页……

BITSCRAMBLER_RX_RD_DUMMY 当接收到 EOF 时，配置比特调节器 RX 路径读取数据模式。

0：等待读取数据

1：忽略读取数据

(R/W)

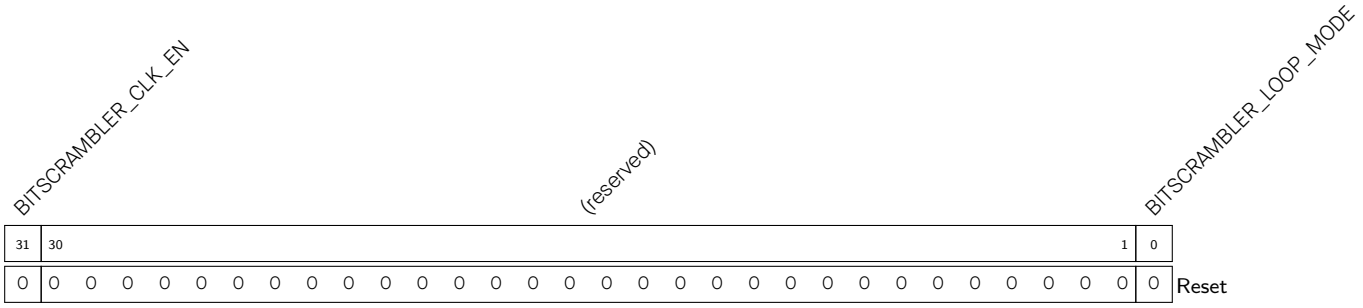
BITSCRAMBLER_RX_FIFO_RST 配置是否重置比特调节器 RX FIFO。

0：不复位

1：复位

(WT)

Register 44.13. BITSCRAMBLER_SYS_REG (0x00F8)



BITSCRAMBLER_LOOP_MODE 配置是否启用比特调节器 TX 到 DMA RX 环回模式

0：禁用

1：启用

(R/W)

BITSCRAMBLER_CLK_EN 配置是否常开配置寄存器时钟。

0：访问时打开

1：常开

(R/W)

Register 44.14. BITSCRAMBLER_TX_STATE_REG (0x0030)



BITSCRAMBLER_TX_IN_IDLE 表示比特调节器 TX 路径是否停止。

0：未停止

1：停止

(RO)

BITSCRAMBLER_TX_IN_RUN 表示比特调节器 TX 路径是否正在运行。

0：未运行

1：运行

(RO)

BITSCRAMBLER_TX_IN_WAIT 表示比特调节器 TX 路径是否正在等待写回完成。

0：不等待

1：等待

(RO)

BITSCRAMBLER_TX_IN_PAUSE 表示比特调节器 TX 路径是否暂停。

0：未暂停

1：暂停

(RO)

BITSCRAMBLER_TX_FIFO_EMPTY 表示比特调节器 TX FIFO 是否为空。

0：不为空

1：空

(RO)

BITSCRAMBLER_TX_EOF_GET_CNT 表示收到 EOF 后比特调节器 TX 路径的字节计数。(RO)

BITSCRAMBLER_TX_EOF_OVERLOAD 表示比特调节器 TX 路径是否尝试处理多个 EOF。

0：不尝试处理多个 EOF

1：尝试处理多个 EOF

(RO)

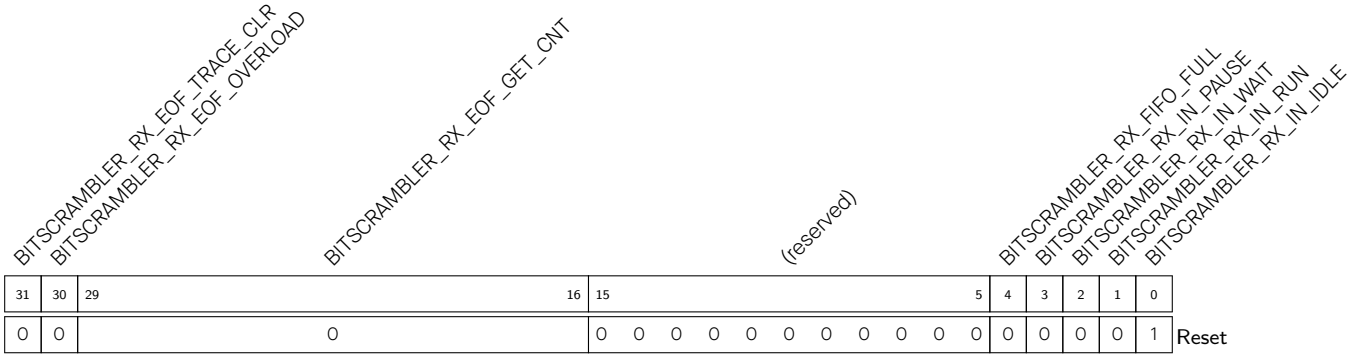
BITSCRAMBLER_TX_EOF_TRACE_CLR 配置是否清除 BITSCRAMBLER_TX_EOF_OVERLOAD 和 BITSCRAMBLER_TX_EOF_GET_CNT。

0：不清楚

1：清除

(WT)

Register 44.15. BITSCRAMBLER_RX_STATE_REG (0x0034)



BITSCRAMBLER_RX_IN_IDLE 表示比特调节器 RX 路径是否停止。

0：未停止

1：停止

(RO)

BITSCRAMBLER_RX_IN_RUN 表示比特调节器 RX 路径是否正在运行。

0：未运行

1：运行

(RO)

BITSCRAMBLER_RX_IN_WAIT 表示比特调节器 RX 路径是否正在等待写回完成。

0：不等待

1：等待

(RO)

BITSCRAMBLER_RX_IN_PAUSE 表示比特调节器 RX 路径是否暂停。

0：未暂停

1：暂停

(RO)

BITSCRAMBLER_RX_FIFO_FULL 表示比特调节器 RX FIFO 是否已满。

0：未满

1：已满

(RO)

BITSCRAMBLER_RX_EOF_GET_CNT 表示收到 EOF 后比特调节器 RX 路径的字节计数。(RO)

BITSCRAMBLER_RX_EOF_OVERLOAD 表示比特调节器 RX 路径是否尝试处理多个 EOF。

0：不尝试处理多个 EOF

1：尝试处理多个 EOF

(RO)

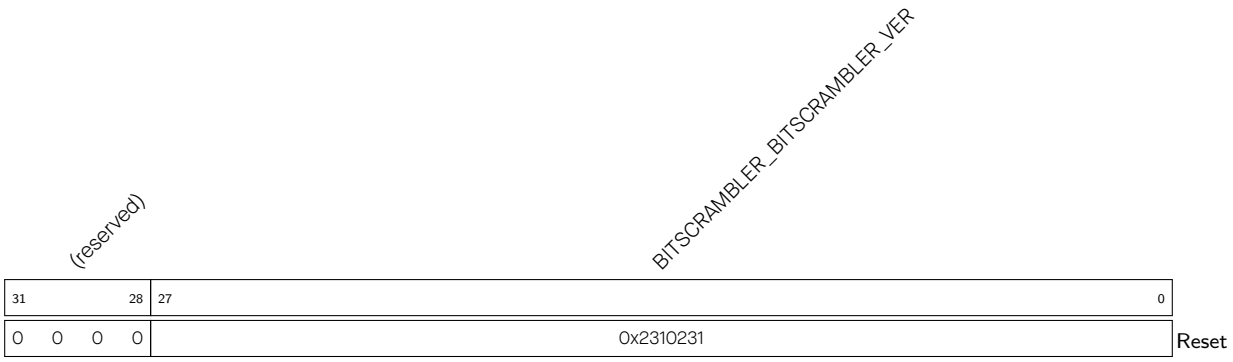
BITSCRAMBLER_RX_EOF_TRACE_CLR 配置是否清除 BITSCRAMBLER_RX_EOF_OVERLOAD 和 BITSCRAMBLER_RX_EOF_GET_CNT。

0：不清楚

1：清除

(WT)

Register 44.16. BITSCRAMBLER_VERSION_REG (0x00FC)



BITSCRAMBLER_BITSCRAMBLER_VER 版本控制寄存器。(R/W)

第 VI 卷

模拟信号处理

该部分描述了与模数转换、电压比较、片上传感器以及温度感应等功能相关的模块，展示了系统在处理模拟信号方面的性能。

第 45 章

温度传感器

45.1 介绍

ESP32-C5 搭载了一个温度传感器，用于实时测量芯片内部温度。温度传感器可将输出的电压转换成数字值，并且带有补偿温度偏移的功能。

45.2 特性

温度传感器的主要特性包括：

- 支持软件触发测量温度，且一旦触发后，传感器可持续测量温度，软件可实时读取数据
- 支持硬件触发自动监测温度
- 支持两种自动监测模式且发送中断
- 支持根据使用环境配置温度偏移，提高测试精度
- 温度测量范围可配置
- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

45.3 结构概览

温度传感器的内部结构如图 45.3-1 所示。

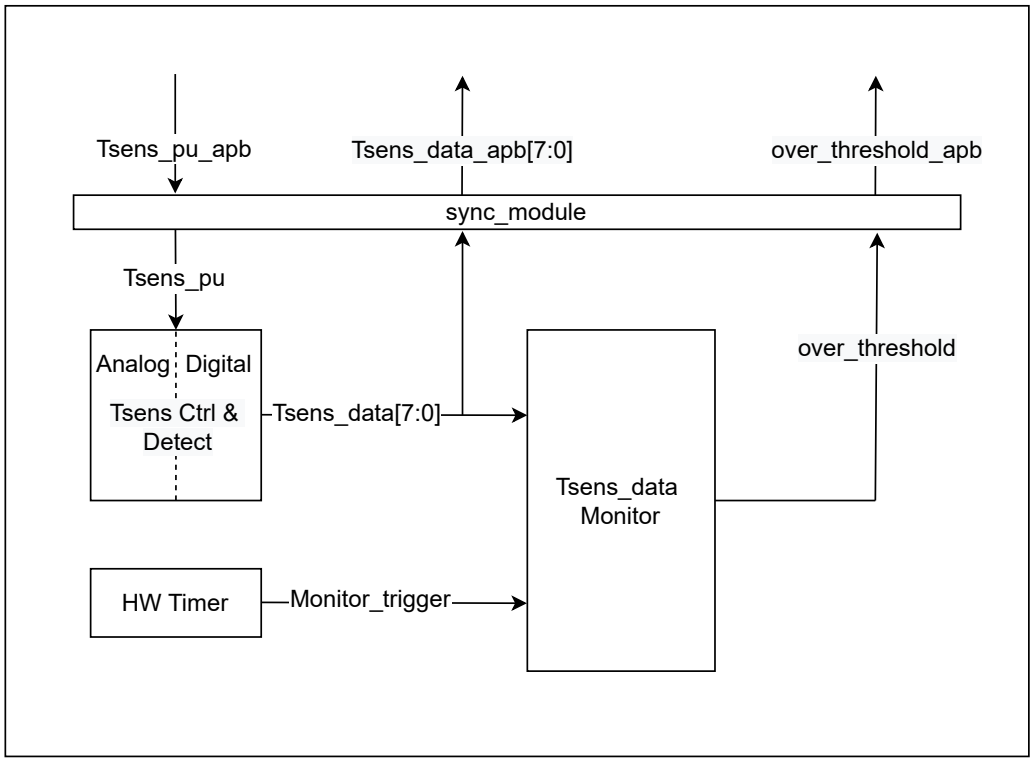


图 45.3-1. 温度传感器结构框图

根据图 45.3-1，温度传感器模块主要包括以下功能模块：

- Tsens Ctrl & Detect：温度传感器
- HW Timer（硬件定时器）：用于触发温度监测
- Tsens_data 监测器：用于监测温度是否超出阈值
- sync_module：APB 时钟域与温度传感器时钟域间的同步模块

45.4 功能描述

45.4.1 温度传感器上电

通过设置 APB_SARADC_TSENS_PU 给温度传感器上电。

45.4.2 时钟管理

温度传感器有两个时钟源：RC_FAST_CLK 和 XTAL_CLK。通过寄存器 PCR_TSENS_CLK_SEL 选择时钟源。时钟分频由寄存器 APB_SARADC_TSENS_CLK_DIV 配置。

45.4.3 自动监测模式

温度传感器自动监测模式有两种：绝对值模式和变化量模式，通过寄存器 APB_SARADC_WAKEUP_MODE 进行选择。

- 绝对值模式：

  - 用于监测当前温度的绝对值。配置 APB_SARADC_WAKEUP_TH_LOW 和 APB_SARADC_WAKEUP_TH_HIGH 来决定温度阈值，温度超过高阈值或小于低阈值时将触发中断。

- 变化量模式：
  - 用于监测温度的变化量。若连续两次采样的温度值增量超过 APB_SARADC_WAKEUP_TH_HIGH 的配置值或减少量超过 APB_SARADC_WAKEUP_TH_LOW 的配置值，将触发中断。例如当 APB_SARADC_WAKEUP_TH_LOW 的值为 8 时，若连续两次温度采样分别为 28 和 19，即温度减少量为 9，将触发中断。

45.4.4 温度测量范围与温度偏移

为了提高测试精度，温度传感器的测量范围被设计为 5 个范围，如表 45.4-1 所示。每个测量范围都有一定的测温误差。用户无需关注测温误差，只需选择相应的温度偏移，即可获得校准后的数据。

表 45.4-1. 温度测量范围与温度偏移

测量范围 (°C)	温度偏移
50 ~ 125	–2
20 ~ 100	–1
–10 ~ 80	0
–30 ~ 50	1
–40 ~ 20	2

45.4.5 数据转换

温度传感器的输出值存储在寄存器 APB_SARADC_TSENS_OUT 中。用户可使用以下公式将输出值 $VALUE$ 转换成实际的温度值 T (°C)。转换公式如下：

$$T = 0.4386 \times VALUE - 27.88 \times offset - 20.52$$

其中，$offset$ 即表 45.4-1 所示温度偏移。

45.5 配置流程

温度传感器可由软件启动，具体配置如下：

1. 置位 APB_SARADC_TSENS_PU 使温度传感器上电；
2. 置位 PCR_TSENS_CLK_EN 使能温度传感器时钟；
3. 配置 APB_SARADC_TSENS_CLK_SEL 选择传感器时钟；
4. 等待 APB_SARADC_TSENS_XPD_WAIT 个时钟周期后，温度传感器的复位释放，开始测量环境温度；
5. 等待 100 μs 后（输出值会随着测量时间的增加而逐渐逼近真实的温度值），直接从 APB_SARADC_TSENS_OUT 中读取温度值。

如要使用硬件自动温度监测功能，则可在给温度传感器上电之前增加如下步骤：

1. 配置 APB_SARADC_TSENS_SAMPLE_RATE 设置采样率；

2. 配置 APB_SARADC_WAKEUP_MODE 选择温度监测模式；
3. 配置 APB_SARADC_WAKEUP_TH_HIGH/LOW 设置温度高低阈值；
4. 置位 APB_SARADC_WAKEUP_EN 启用温度监测；
5. 置位 APB_SARADC_TSENS_SAMPLE_EN 自动触发温度监测。

自动温度监测工作模式下，温度传感器的输出值不会存储，但用户可以随时读取寄存器 APB_SARADC_TSENS_OUT 获取当前输出值。

45.6 中断

ESP32-C5 的温度传感器可以生成以下中断信号并发送给 中断矩阵。

- LP_TSENS_INTR

这个中断信号由以下内部中断源生成：

- APB_SARADC_TSENS_INT：温度采样值超过阈值，即触发此中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 45.8 寄存器列表。

45.7 事件任务矩阵功能

在 ESP32-C5 中，温度传感器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发温度传感器的 ETM 任务，或者通过温度传感器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 *(ETM)*。这里仅介绍与温度传感器相关的 ETM 任务和 ETM 事件。

温度传感器可接收的 ETM 任务有：

- TMPSNSR_TASK_START_SAMPLE：该任务被触发时，温度传感器开始采样。
- TMPSNSR_TASK_STOP_SAMPLE：该任务被触发时，温度传感器停止采样。

温度传感器可产生的 ETM 事件有：

- TMPSNSR_EVT_OVER_LIMIT：温度传感器发现温度超过阈值时生成。

在具体应用中，温度传感器的 ETM 事件可以用来触发温度传感器的 ETM 任务，例如，TMPSNSR_EVT_OVER_LIMIT 事件可以触发 TMPSNSR_TASK_STOP_SAMPLE 任务。

45.8 寄存器列表

本小节的所有地址均为相对于温度传感器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

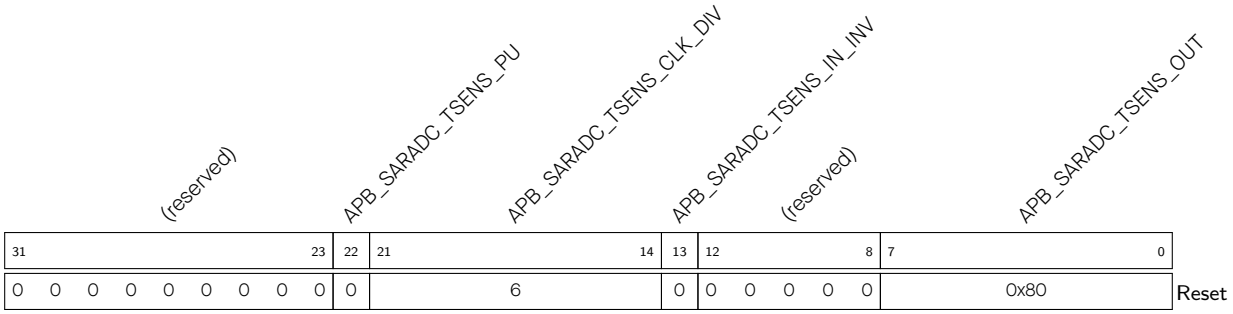
名称	描述	地址	访问
配置寄存器			
APB_SARADC_INT_ENA_REG	SAR ADC / 温度传感器中断使能寄存器	0x0040	R/W
APB_SARADC_INT_RAW_REG	SAR ADC / 温度传感器原始中断寄存器	0x0044	R/WTC/SS
APB_SARADC_INT_ST_REG	SAR ADC / 温度传感器中断状态寄存器	0x0048	RO
APB_SARADC_INT_CLR_REG	SAR ADC / 温度传感器中断清除寄存器	0x004C	WT
APB_SARADC_APB_TSENS_CTRL_REG	温度传感器控制寄存器 1	0x0058	varies
APB_SARADC_APB_TSENS_CTRL2_REG	温度传感器控制寄存器 2	0x005C	R/W
APB_TSENS_WAKE_REG	温度传感器自动监测功能配置寄存器	0x0064	varies
APB_TSENS_SAMPLE_REG	温度传感器采样配置寄存器	0x0068	R/W
版本控制寄存器			
APB_SARADC_APB_CTRL_DATE_REG	版本控制寄存器	0x03FC	R/W

45.9 寄存器

本小节的所有地址均为相对于温度传感器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

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

Register 45.1. APB_SARADC_APB_TSENS_CTRL_REG (0x0058)



APB_SARADC_TSENS_OUT 表示温度传感器的输出值。(RO)

APB_SARADC_TSENS_IN_INV 配置是否反转温度传感器的输入值。

0：无效

1：反转

(R/W)

APB_SARADC_TSENS_CLK_DIV 表示温度传感器的时钟分频系数。(R/W)

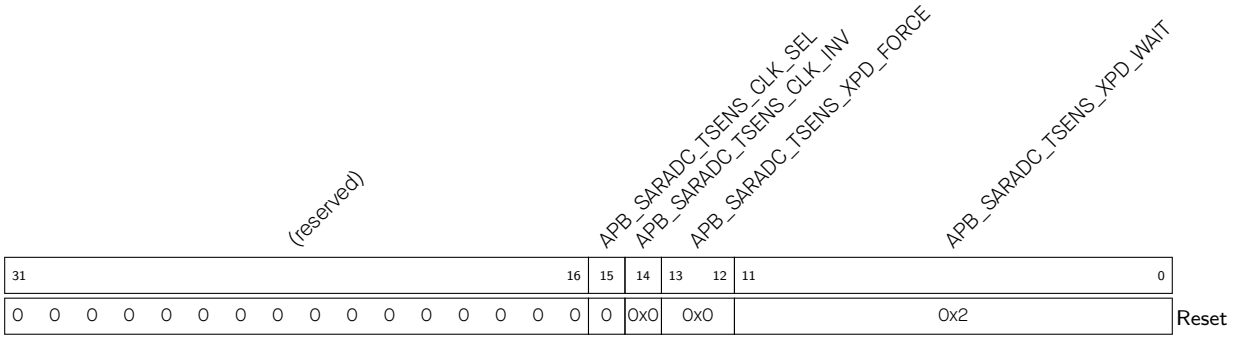
APB_SARADC_TSENS_PU 配置是否使能温度传感器上电。

0：无效

1：使能

(R/W)

Register 45.2. APB_SARADC_APB_TSENS_CTRL2_REG (0x005C)



APB_SARADC_TSENS_CLK_SEL 配置温度传感器时钟。

0：RC_FAST_CLK

1：XTAL_CLK

(R/W)

APB_SARADC_TSENS_CLK_INV 配置传感器采样时钟的相位。(R/W)

APB_SARADC_TSENS_XPD_FORCE 配置是否使能强制上电/断电温度传感器。

0：禁用强制上电功能

1：禁用强制断电功能

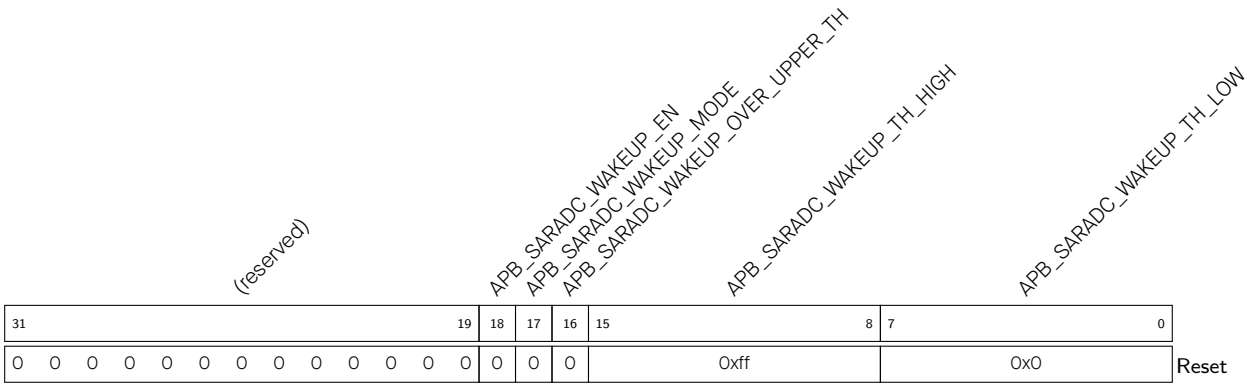
2：强制上电温度传感器

3：强制断电温度传感器

(R/W)

APB_SARADC_TSENS_XPD_WAIT 配置温度传感器的复位释放前需要等待的时钟周期。(R/W)

Register 45.3. APB_TSENS_WAKE_REG (0x0064)



APB_SARADC_WAKEUP_TH_LOW 配置温度传感器自动监测功能的低阈值。(R/W)

APB_SARADC_WAKEUP_TH_HIGH 配置温度传感器自动监测功能的高阈值。(R/W)

APB_SARADC_WAKEUP_OVER_UPPER_TH 表示输出值是否超过阈值。

0：输出值低于低阈值

1：输出值高于高阈值

(RO)

APB_SARADC_WAKEUP_MODE 配置温度监测模式。仅当 APB_SARADC_WAKEUP_EN=1 时有效。

0：绝对值模式

1：变化量模式

(R/W)

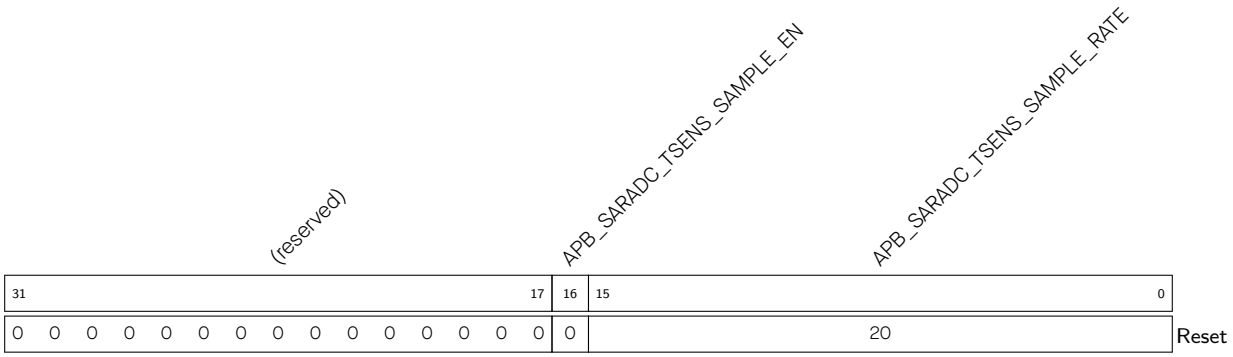
APB_SARADC_WAKEUP_EN 配置是否启用温度监测功能。

0：禁用

1：启用

(R/W)

Register 45.4. APB_TSENS_SAMPLE_REG (0x0068)



APB_SARADC_TSENS_SAMPLE_RATE 配置硬件自动温度监测的采样率。采样周期 = 配置值 × 传感器工作时钟周期。(R/W)

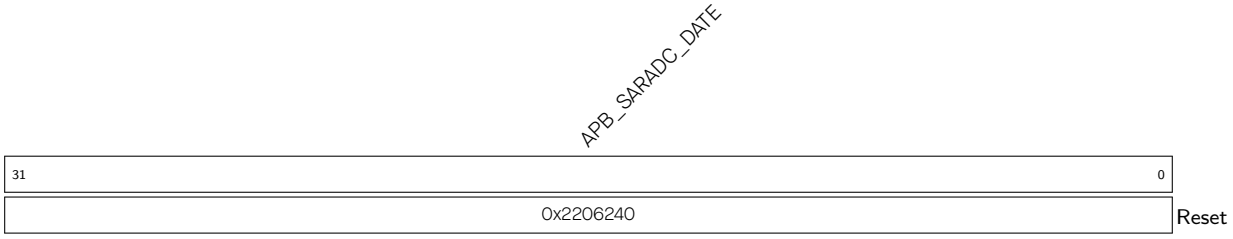
APB_SARADC_TSENS_SAMPLE_EN 配置是否自动触发持续温度监测。

0：不使能

1：使能

(R/W)

Register 45.5. APB_SARADC_CTRL_DATE_REG (0x03FC)



APB_SARADC_DATE SAR ADC（请参考章节 46 *ADC 控制器* 获取详细信息）和温度传感器的版本控制寄存器。(R/W)

第 46 章

ADC 控制器

46.1 概述

ESP32-C5 搭载了一个 12 位逐次逼近型模拟数字转换器 (SAR ADC)，用于测量最多来自 6 个管脚上的模拟信号。

SAR ADC 由 DIG ADC 控制器控制，该控制器驱动电压采样，支持单次采样和多通道采样。

46.2 术语表

下列有关 SAR ADC 的术语是在《ESP32-C5 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

SAR ADC： 包含模拟 ADC 电路与数字 ADC 控制器。

单次采样模式： 在此模式下，ADC 一次对一个通道进行采样。

多通道采样模式： 在此模式下，ADC 依次对一组通道进行采样，或者连续对单个通道进行采样。

转换结果： 转换结果是模拟到数字转换后得到的二进制数字值。有时也被称为转换数据。

滤波数据： 滤波数据是转换数据经过滤波器处理得到的结果。

采样： 该术语通常指的是对模拟输入进行采样、转换采样数据，并将转换结果传输到内存的整个过程。在特定语境和流程中，“采样”也可能指 SAR ADC 从模拟信号中采集数据。

46.3 特性

SAR ADC 具有以下特性：

- 12 位分辨率
- 支持采集最多 6 个管脚上的模拟信号
- 支持单次采样模式和多通道采样模式
- 在多通道采样模式下，支持：
  - 自定义采样通道顺序
  - 两个滤波器，滤波系数可配
  - 阈值监控，滤波后数据大于设置的高阈值或小于设置的低阈值将产生中断
  - GDMA 连续数据搬运

- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

46.4 结构概览

SAR ADC 的主要子模块与连接情况见图 46.4-1。

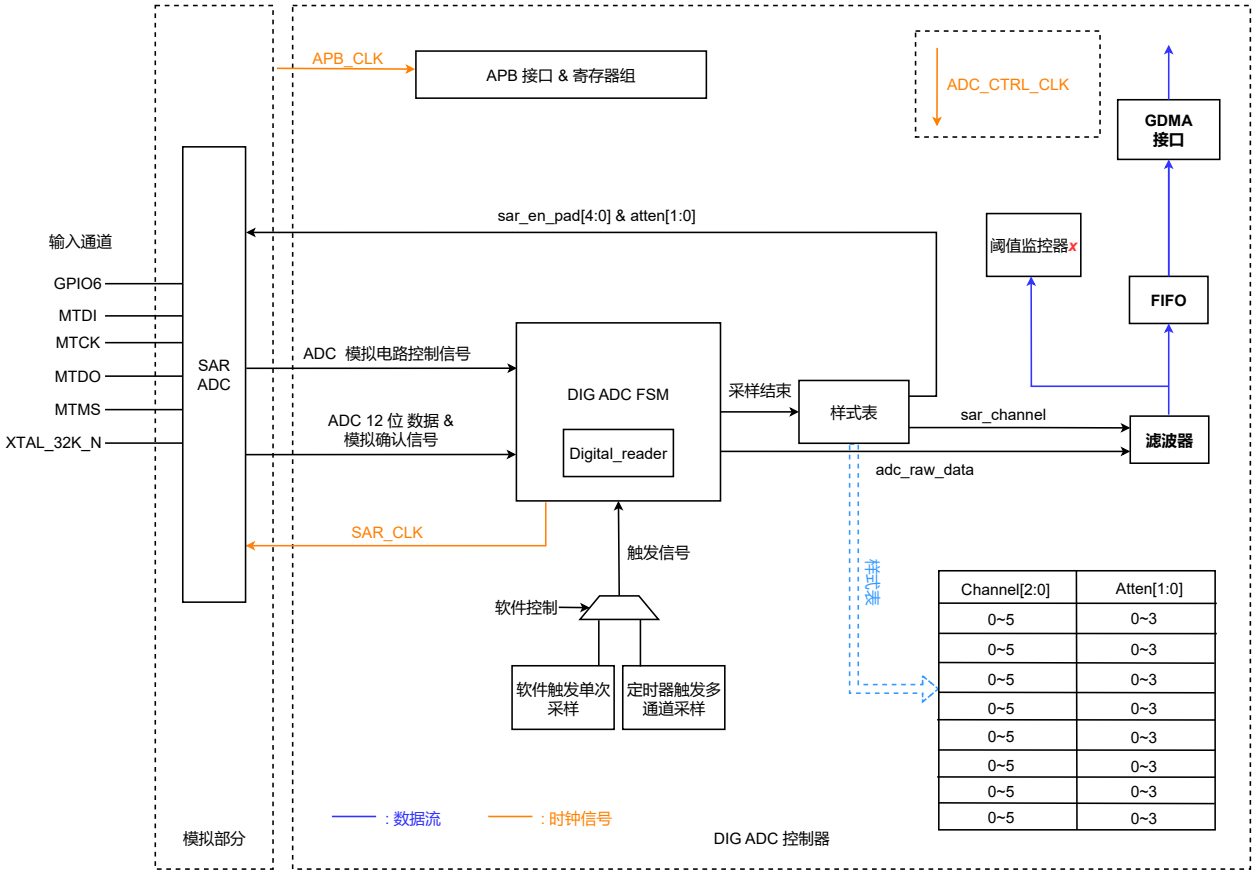


图 46.4-1. SAR ADC 结构图

根据图 46.4-1，按照从左至右、从上到下的顺序，SAR ADC 模块主要包括以下功能模块：

- 6 个通道，连接 6 个芯片管脚
- SAR ADC：SAR ADC 的模拟部分
- DIG ADC 控制器：SAR ADC 的数字部分，主要包括：
  - 时钟管理模块：对时钟源进行选择和分频
  - DIG ADC FSM（有限状态机）：生成整个 ADC 采样过程中所需的各种信号
  - Digital_reader：读取 SAR ADC 的数值，由 DIG ADC FSM 驱动
  - 滤波器：用于在多通道采样模式下对目标通道的转换数据进行滤波
  - Threshold Monitor*x*：阈值监控器 1 和阈值监控器 2。可在采样值大于设定的高阈值，或小于设定的低阈值时触发中断。

46.5 功能描述

46.5.1 ADC 上电

置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 寄存器给 ADC 上电。上电后即可进行采样，硬件设计上已考虑等待时间，因此用户无需特意等待。

46.5.2 ADC 通道

SAR ADC 具有 6 个通道，连接芯片的 6 个管脚。ADC 需首先通过内部多路器选择待测量的模拟管脚，然后才能采样模拟信号。

表 46.5-1 列出了用作 ADC 通道的芯片管脚和相应的 GPIO 编号。

表 46.5-1. SAR ADC 通道

芯片管脚名称	GPIO 编号	ADC 通道编号
XTAL_32K_N	GPIO1	0
MTMS	GPIO2	1
MTDI	GPIO3	2
MTCK	GPIO4	3
MTDO	GPIO5	4
GPIO6	GPIO6	5

46.5.3 ADC 时钟

图 46.5-1 为 SAR ADC 的时钟结构图。

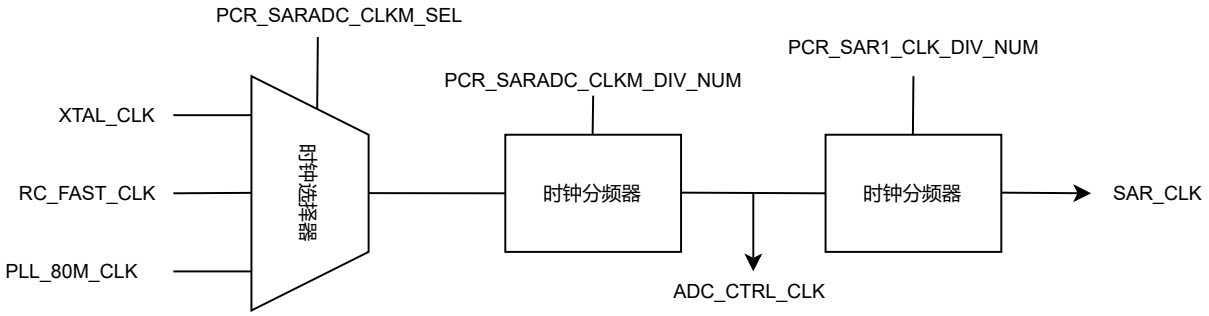


图 46.5-1. SAR ADC 时钟结构图

SAR ADC 的系统时钟为 ADC_CTRL_CLK，它是 DIG ADC FSM 和其他逻辑电路的工作时钟（APB 接口和 Digital_reader 除外）。

ADC_CTRL_CLK 可选择三个时钟源：XTAL_CLK，RC_FAST_CLK，或 PLL_80M_CLK。通过寄存器 PCR_SARADC_CLKM_SEL 选择时钟源。

SAR ADC 的分频时钟为 SAR_CLK，由 ADC_CTRL_CLK 分频而得。它是 SAR ADC 和 Digital_reader 的工作时钟，频率不得超过 5 MHz。

更多有关时钟的信息请参考章节 9 复位和时钟。

46.5.4 单次采样模式

单次采样模式下，ADC 只在一个通道进行一次采样。该模式由软件通过寄存器 APB_SARADC_ONETIME_SAMPLE 触发。一旦采样完成，转换数据将被存储在 APB_SARADC_DATA 中。如需转换采样通道，可继续配置 APB_SARADC_ONETIME_SAMPLE 寄存器。

46.5.5 多通道采样模式

多通道采样模式由专用定时器触发。在该模式下，ADC 会按照指定的顺序采样一组通道。通道顺序在样式表中进行定义。多通道采样模式也可以配置成连续采样一个通道。

用户通过配置寄存器 APB_SARADC_TIMER_EN 使能定时器，配置 APB_SARADC_TIMER_TARGET 设置定时器的触发周期。当定时器计数到配置周期数的 2 倍时，触发采样。定时器的工作时钟是 ADC_CTRL_CLK。

采样完成后，定时器复位为 0，重新开始计数。数据通过 GDMA 连续搬运至内存空间。

说明：
单次采样和多通道采样不能同时使用。

46.5.6 ADC 转换和衰减

SAR ADC 转换模拟信号时，转换电压范围为 0 mV ~ V_{ref}。V_{ref} 为 ADC 内部参考电压，出厂设定为 1100 mV。ADC 的转换结果是个 12 位的数字值，是原始值。用户可使用以下公式将原始值 $data$ 转换成模拟电压 V_{data}：

$$V_{data} = \frac{V_{ref}}{k} \times \frac{data}{4095}$$

其中 k 为衰减对应的系数。

如需转换大于 V_{ref} 的电压，信号输入 SAR ADC 前可进行衰减。使用寄存器 APB_SARADC_ONETIME_ATTEN 可配置以下衰减：

- 0 dB (k≈100%)
- 2.5 dB (k≈75%)
- 6 dB (k≈50%)
- 12 dB (k≈25%)

46.5.7 DIG ADC FSM

DIG ADC FSM（下文简称 FSM）在多通道采样模式下工作，用于生成整个 ADC 采样过程中所需的各种信号。图 46.5-2 展示了 FSM 的工作原理。

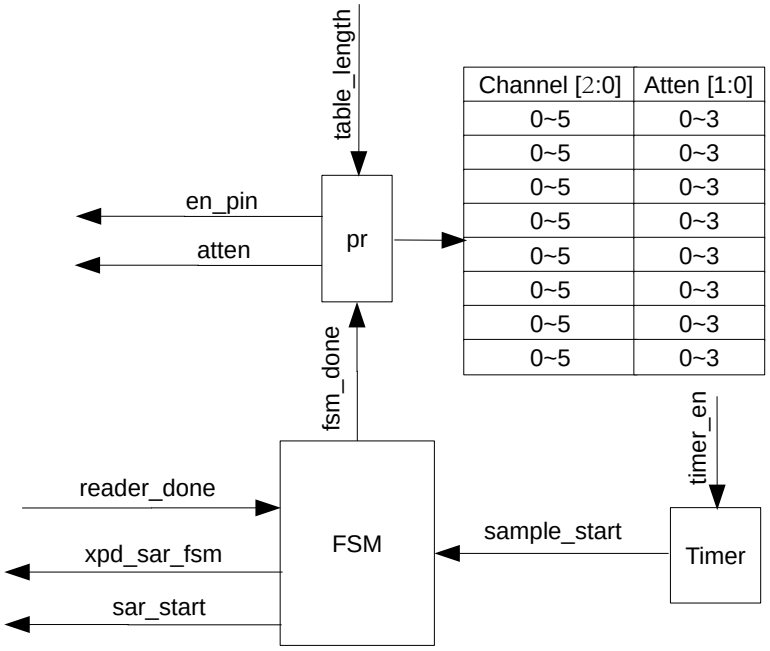


图 46.5-2. DIG ADC FSM 概况

其中，

- Timer（定时器）：表示 DIG ADC 控制器的专用定时器，用于触发多通道采样模式。定时器可产生 sample_start 信号，表示开始采样。
- pr：样式表指针，FSM 将根据该指针指向的样式配置，发送相应信号。

用户通过置位 APB_SARADC_TIMER_EN 使能定时器。定时器超时将触发 sample_start 信号驱动 FSM 模块开始采样。FSM 模块收到 sample_start 信号后，执行以下操作：

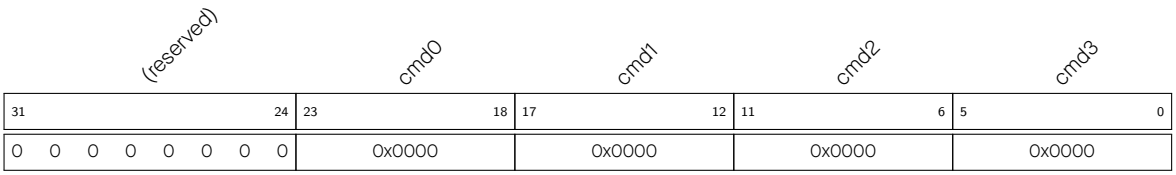
- 开启 SAR ADC 电源；
- 根据当前 pr 指向的样式，决定该次采样的 ADC 通道以及衰减；
- 根据配置信息，输出相应的 en_pin（使能管脚）以及 atten（衰减）信号到模拟端；
- 发起 sar_start 信号，开启采样。

FSM 收到 Digital_reader 返回的 reader_done 信号后，执行以下操作：

- 结束采样；
- 将数据传输给滤波器，然后阈值监控器通过 GDMA 将数据传输给内存（见图 46.4-1）；
- 更新样式表指针 pr，等待下一次采样。pr 在 0 - APB_SARADC_SAR_PATT_LEN (table_length) 之间循环计数。

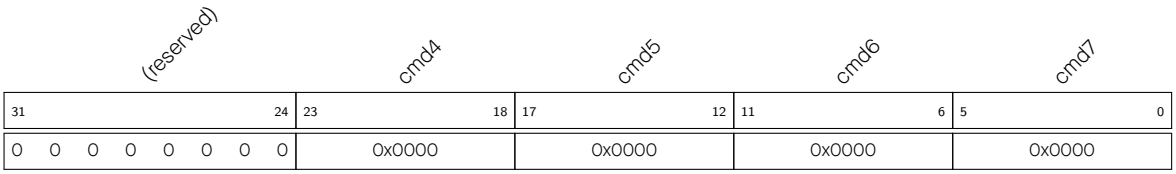
46.5.8 样式表

FSM 包含一个样式表，由 APB_SARADC_SAR_PATT_TAB1_REG 和 APB_SARADC_SAR_PATT_TAB2_REG 两个寄存器组成，每个寄存器包含四个样式，每个样式长度为六位，如图 46.5-3 和图 46.5-4 所示。



cmd *x* 表示样式表中的样式，即样式 0 ~ 样式 3。

图 46.5-3. APB_SARADC_SAR_PATT_TAB1_REG 包含样式 0 - 3



cmd *x* 表示样式表中的样式，即样式 4 ~ 样式 7。

图 46.5-4. APB_SARADC_SAR_PATT_TAB2_REG 包含样式 4 - 7

每个样式长度为六位，共包含三个字段，每个字段存储的是采样规则。图 46.5-5 所示为样式结构，字段描述参见后文。



图 46.5-5. 样式结构

atten 配置衰减：

0：0 dB
1：2.5 dB
2：6 dB
3：12 dB

ch_sel 配置通道：

0：通道 0
1：通道 1
2：通道 2
3：通道 3
4：通道 4
5：通道 5

(reserved) 保留字段

样式配置示例

注意：使用多通道连续采样时，每个通道的衰减必须设置成一致；例如，希望实现如下所示的多通道采样方式：

- 采样 SAR ADC 的通道 0，且衰减配置为 2.5 dB；

- 采样 SAR ADC 的通道 2，且衰减配置为 2.5 dB。

则具体的配置如下：

- 配置第一个样式 cmd0，如下图所示：

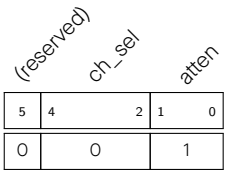


图 46.5-6. cmd0 配置示例

atten 配置该字段的值为 1，即衰减配置为 2.5 dB。

ch_sel 配置该字段的值为 0，即选择通道 0。

- 配置第二个样式 cmd1，如下图所示：



图 46.5-7. cmd1 配置示例

atten 配置该字段的值为 1，即衰减配置为 2.5 dB。

ch_sel 配置该字段的值为 2，即选择通道 2。

- 配置 APB_SARADC_SAR_PATT_LEN 为 1，即选择使用上述配置好的样式 0 和样式 1；
- 使能定时器，则 DIG ADC 控制器将根据上述样式配置周期性采样两个通道。

46.5.9 ADC 滤波器

DIG ADC 控制器提供两个滤波器，用于在多通道采样模式下对目标通道的转换数据进行滤波。两个滤波器均可配置给 SAR ADC 的任一通道，但不能配置给同一通道。如果配置给同一通道，则第一次配置的滤波器生效。

滤波公式如下所示：

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} + 0.5$$

- $data_{cur}$：滤波后数据
- $data_{in}$：ADC 转换数据
- $data_{prev}$：上次滤波数据
- k：滤波系数

配置滤波器如下：

- 配置 APB_SARADC_FILTER_CHANNEL*x* 设置滤波器 *x* (*x*=0, 1) 作用的 ADC 通道；
- 配置 APB_SARADC_FILTER_FACTOR*x* 设置滤波器 *x* 的滤波系数 k。

46.5.10 阈值监控器

DIG ADC 控制器包含两个阈值监控器，用于在多通道采样模式下对滤波之后的数据进行监控。当数据大于设定的高阈值，则触发高阈值中断；当数据小于设定的低阈值，则触发低阈值中断。两个监控器可配置给 SAR ADC 的任一通道，但不能配置给同一通道。

阈值监控配置如下：

- 置位 APB_SARADC_THRES*x*_EN 使能阈值监控器 *x* (*x*=0, 1)；
- 配置 APB_SARADC_THRES*x*_LOW 设置低阈值；
- 配置 APB_SARADC_THRES*x*_HIGH 设置高阈值；
- 配置 APB_SARADC_THRES*x*_CHANNEL 设置监控的通道；
- 配置 APB_SARADC_THRES_ALL_EN 使能阈值监控。

46.5.11 GDMA 支持

由于 ADC 仅有一个数据寄存器 APB_SARADC_DATA 用于存储单次采样的转换数据，因此当转换多个通道时需要使用 GDMA，通过 GDMA 接口将转换数据连续存储至内存。

由 DIG ADC 控制器的专用定时器产生 GDMA 功能触发信号。用户可通过软件配置 APB_SARADC_APB_ADC_TRANS 将 GDMA 的数据通路切换到 DIG ADC 控制器。关于 GDMA 的具体配置，请参考章节 5 *通用 DMA 控制器 (GDMA)*。

GDMA 数据格式

ADC 最终向 GDMA 传递 32 位数据，如下图所示：

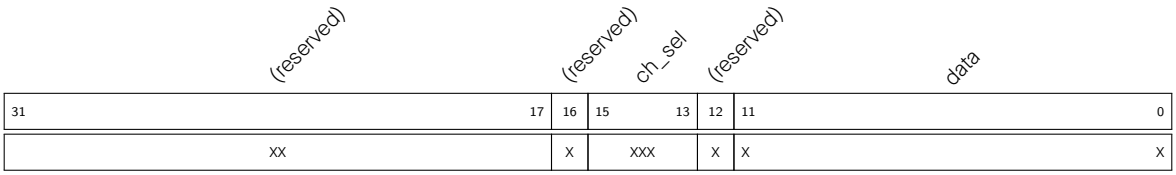


图 46.5-8. GDMA 数据格式

data ADC 转换结果，12 位

ch_sel 通道信息，3 位

46.6 事件任务矩阵功能

在 ESP32-C5 中，SAR ADC 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 SAR ADC 的 ETM 任务，或者通过 SAR ADC 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与 SAR ADC 相关的 ETM 任务和 ETM 事件。

SAR ADC 可接收的 ETM 任务有：

- ADC_TASK_SAMPLE0：该任务被触发时，ADC 开始进行单次采样。
- ADC_TASK_START0：该任务被触发时，ADC 开始进行多通道采样。
- ADC_TASK_STOP0：该任务被触发时，ADC 停止采样。

SAR ADC 可产生的 ETM 事件有：

- ADC_EVT_CONV_CMPLTO：在单次采样模式和多通道采样模式下，ADC 每完成一次采样时生成。
- ADC_EVT_EQ_ABOVE_THRESH*x*：ADC 过滤后数据高于阈值时生成。*x*=0, 1，代表阈值监控器 0, 1。
- ADC_EVT_EQ_BELOW_THRESH*x*: ADC 过滤后数据低于阈值时生成。*x*=0, 1，代表阈值监控器 0, 1。
- ADC_EVT_STARTEDO：ADC 开始采样时生成，单次采样不会触发该事件。
- ADC_EVT_STOPPEDO：ADC 停止采样时生成，单次采样不会触发该事件。

在具体应用中，SAR ADC 的 ETM 事件可以用来触发 SAR ADC 的 ETM 任务，例如，ADC_EVT_EQ_ABOVE_THRESH*x* 事件可以触发 ADC_TASK_STOPO 任务。

46.7 中断

ESP32-C5 的 SAR ADC 可以生成 APB_ADC_INTR 中断信号并发送给中断矩阵。这一中断信号由以下 ADC 的内部中断源生成。

- APB_SARADC_ADC_DONE_INT：SAR ADC 完成一次转换，即触发此中断。
- APB_SARADC_THRES*x*_HIGH_INT：过滤值超过阈值监控器 *x* 的高阈值，即触发此中断。
- APB_SARADC_THRES*x*_LOW_INT：过滤值低于阈值监控器 *x* 的低阈值，即触发此中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 46.9 寄存器列表。

46.8 配置流程

46.8.1 单次采样配置

软件驱动的单次采样的具体配置如下：

1. 置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 给 ADC 上电；
2. 配置 PCR_SARADC_CLKM_SEL 选择时钟源；
3. 配置时钟分频寄存器 PCR_SARADC_CLKM_DIV_NUM 和 PCR_SAR1_CLK_DIV_NUM；
4. 置位 PCR_SARADC_CLKM_EN 使能 SAR ADC 时钟；
5. 置位 APB_SARADC_ONETIME_SAMPLE 使能单次采样；
6. 配置 APB_SARADC_ONETIME_CHANNEL 选择采样通道；
7. 根据需要，配置 APB_SARADC_ONETIME_ATTEN 选择衰减；
8. 置位 APB_SARADC_ONETIME_START 启动单次采样。

采样结束即触发 APB_SARADC_ADC_DONE_INT_RAW 中断。软件检测到该中断后，可启用从 APB_SARADC_ADC_DATA 中读取样本值。如需切换采样通道无需重复 1 至 5 步骤。

46.8.2 多通道采样配置

专用定时器驱动的多通道采样的具体配置如下：

1. 置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 给 ADC 上电；
2. 配置 PCR_SARADC_CLKM_SEL 选择时钟源；
3. 配置时钟分频寄存器 PCR_SARADC_CLKM_DIV_NUM 和 PCR_SAR1_CLK_DIV_NUM；
4. 置位 PCR_SARADC_CLKM_EN 使能 SAR ADC 时钟；
5. 按照章节 46.5.8 *样式表* 所述步骤配置样式表；
6. 按照章节 46.5.9 *ADC 滤波器* 所述步骤配置滤波器作用的通道和滤波系数；
7. 根据需要，按照章节 46.5.10 *阈值监控器* 所述步骤设置阈值监控；
8. 置位 APB_SARADC_APB_ADC_TRANS 选择使用 GDMA；
9. 配置 APB_SARADC_TIMER_TARGET 设置 DIG ADC 定时器的触发周期；
10. 置位 APB_SARADC_TIMER_EN 使能定时器。

定时器超时则将驱动 DIG ADC FSM 根据样式表进行采样。数据通过 GDMA 自动存储到内存空间中。当采集到 APB_SARADC_APB_ADC_EOF_NUM 寄存器设置的次数时，采样停止。

采样完成将产生 APB_SARADC_ADC_DONE_INT_RAW 中断。

46.9 寄存器列表

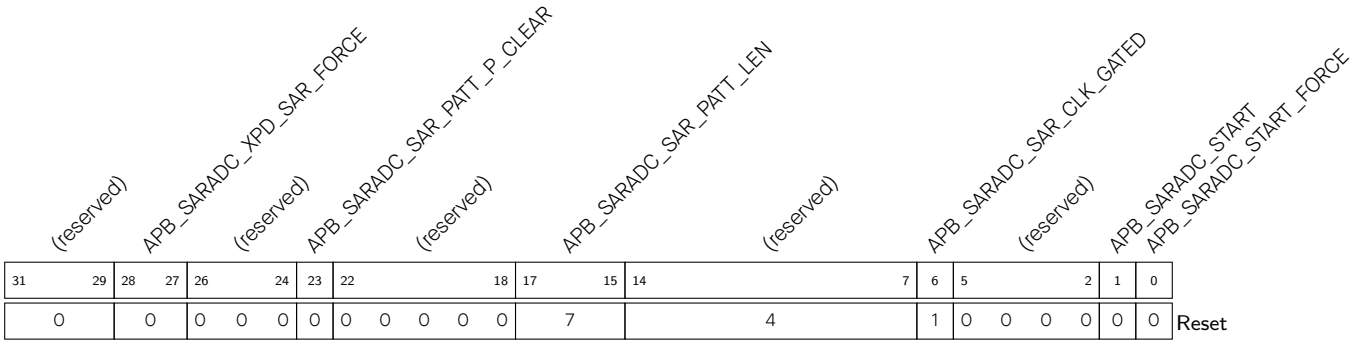
本小节的所有地址均为相对于 SAR ADC 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
APB_SARADC_CTRL_REG	SAR ADC 控制寄存器 1	0x0000	R/W
APB_SARADC_CTRL2_REG	SAR ADC 控制寄存器 2	0x0004	R/W
APB_SARADC_FILTER_CTRL1_REG	滤波控制寄存器 1	0x0008	R/W
APB_SARADC_SAR_PATT_TAB1_REG	样式表寄存器 1	0x0018	R/W
APB_SARADC_SAR_PATT_TAB2_REG	样式表寄存器 2	0x001C	R/W
APB_SARADC_ONETIME_SAMPLE_REG	单次采样配置寄存器	0x0020	R/W
APB_SARADC_FILTER_CTRL0_REG	滤波控制寄存器 0	0x0028	R/W
APB_SARADC_SAR1DATA_STATUS_REG	SAR ADC 转换数据存储寄存器	0x002C	RO
APB_SARADC_THRES0_CTRL_REG	采样阈值控制寄存器 0	0x0034	R/W
APB_SARADC_THRES1_CTRL_REG	采样阈值控制寄存器 1	0x0038	R/W
APB_SARADC_THRES_CTRL_REG	采样阈值控制寄存器	0x003C	R/W
APB_SARADC_INT_ENA_REG	SAR ADC 中断使能寄存器	0x0040	R/W
APB_SARADC_INT_RAW_REG	SAR ADC 原始中断寄存器	0x0044	R/WTC/SS
APB_SARADC_INT_ST_REG	SAR ADC 中断状态寄存器	0x0048	RO
APB_SARADC_INT_CLR_REG	SAR ADC 中断清除寄存器	0x004C	WT
APB_SARADC_DMA_CONF_REG	SAR ADC DMA 配置寄存器	0x0050	R/W
APB_SARADC_APB_CTRL_DATE_REG	版本控制寄存器	0x03FC	R/W

46.10 寄存器

本小节的所有地址均为相对于 SAR ADC 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 6 系统和存储器 中的表 6.3-2。

Register 46.1. APB_SARADC_CTRL_REG (0x0000)



APB_SARADC_START_FORCE 配置是否使用软件启动 SAR ADC 采样。

0：选择使用 FSM 启动 SAR ADC 采样

1：选择使用软件启动 SAR ADC 采样

(R/W)

APB_SARADC_START 配置是否使用软件启动 SAR ADC。

0：无效

1：使用软件启动 SAR ADC

仅当 APB_SARADC_START_FORCE = 1 时有效。

(R/W)

APB_SARADC_SAR_CLK_GATED 配置是否使能 SAR ADC 时钟门控。

0：不使能

1：使能

(R/W)

APB_SARADC_SAR_PATT_LEN 配置需要使用的样式数量。

0：仅使用 cmd0

1：使用 cmd0 和 cmd1

n：依次使用 cmd0 至 cmdn, n 最大取 7

(R/W)

APB_SARADC_SAR_PATT_P_CLEAR 配置是否清除 DIG ADC 控制器样式表指针。

0：无效

1：清除

(R/W)

APB_SARADC_XPD_SAR_FORCE 配置是否强制上电 SAR ADC。

0：无效

1：强制上电

(R/W)

Register 46.2. APB_SARADC_CTRL2_REG (0x0004)



APB_SARADC_MEAS_NUM_LIMIT 配置是否使能 SAR ADC 最大转换次数限制。

0：不使能

1：使能

(R/W)

APB_SARADC_MAX_MEAS_NUM 配置 SAR ADC 的最大转换次数。(R/W)

APB_SARADC_SAR1_INV 配置是否反转 SAR ADC 数据。

0：无效

1：反转 SAR ADC 数据

(R/W)

APB_SARADC_TIMER_TARGET 设置 SAR ADC 定时器目标，即定时器的触发周期。(R/W)

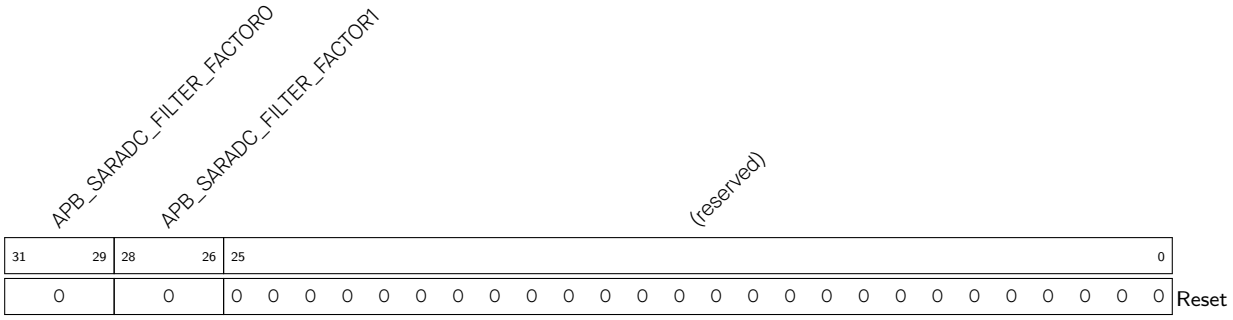
APB_SARADC_TIMER_EN 配置是否使能 SAR ADC 定时器触发。

0：不使能

1：使能

(R/W)

Register 46.3. APB_SARADC_FILTER_CTRL1_REG (0x0008)



APB_SARADC_FILTER_FACTOR1 设置 SAR ADC 滤波器 1 的滤波系数 k。

0: k=0（即滤波器关闭）

1: k=2

2: k=4

3: k=8

4: k=16

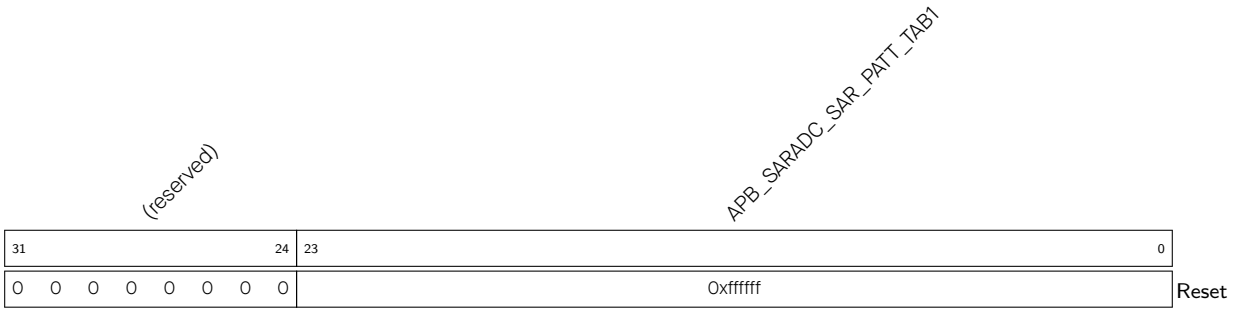
5: k=32

6: k=64

(R/W)

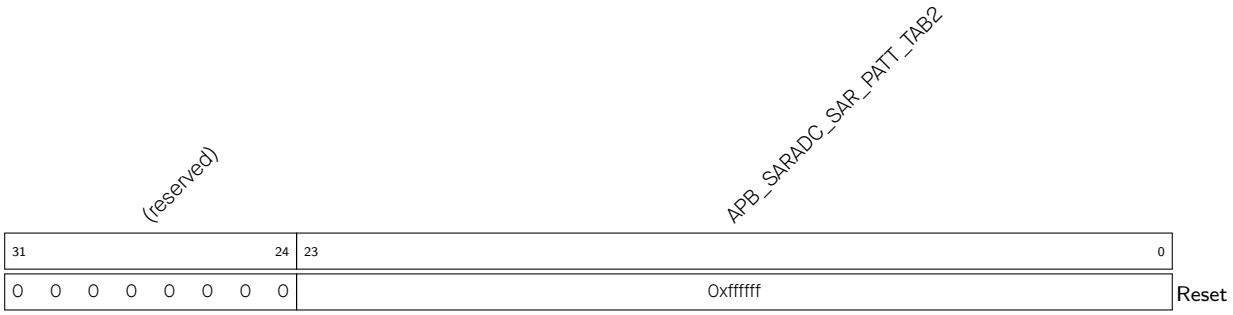
APB_SARADC_FILTER_FACTOR0 设置 SAR ADC 滤波器 0 的滤波系数 k（配置同上）。(R/W)

Register 46.4. APB_SARADC_SAR_PATT_TAB1_REG (0x0018)



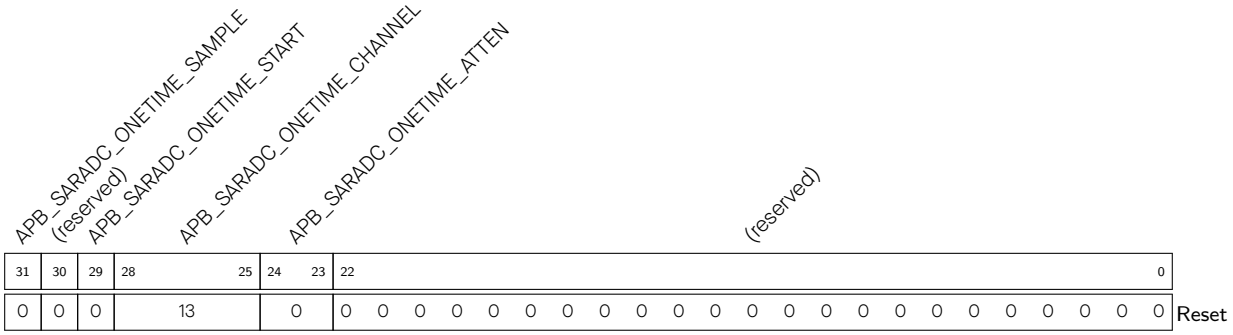
APB_SARADC_SAR_PATT_TAB1 配置样式表的第 0 ~ 3 个样式。每个样式占六位。具体见小节 46.5.8 样式表。(R/W)

Register 46.5. APB_SARADC_SAR_PATT_TAB2_REG (0x001C)



APB_SARADC_SAR_PATT_TAB2 配置样式表的第 4 ~ 7 个样式，每个样式占六位。具体见小节 46.5.8 样式表。(R/W)

Register 46.6. APB_SARADC_ONETIME_SAMPLE_REG (0x0020)



APB_SARADC_ONETIME_ATTEN 配置单次采样的衰减。

0：0 dB

1：2.5 dB

2：6 dB

3：12 dB

(R/W)

APB_SARADC_ONETIME_CHANNEL 配置单次采样的通道。

0：通道 0

1：通道 1

2：通道 2

3：通道 3

4：通道 4

5：通道 5

(R/W)

APB_SARADC_ONETIME_START 配置是否启动 SAR ADC 单次采样。

0：无效

1：启动

(R/W)

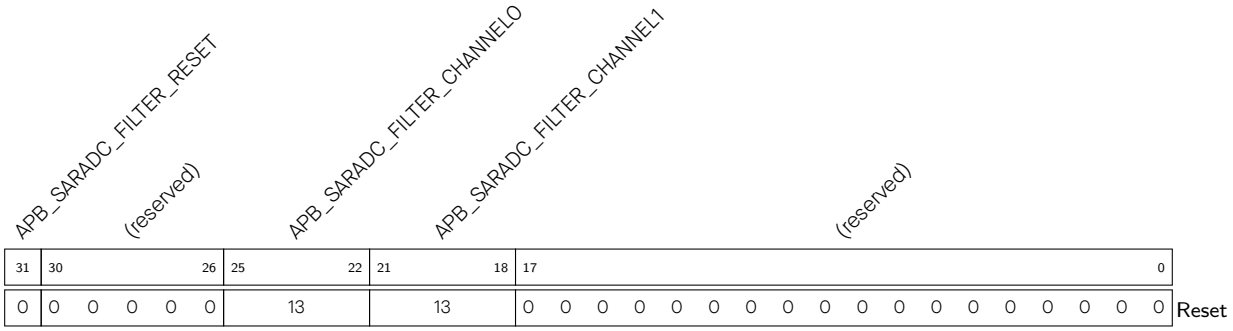
APB_SARADC_ONETIME_SAMPLE 配置是否使能 SAR ADC 单次采样。

0：不使能

1：使能

(R/W)

Register 46.7. APB_SARADC_FILTER_CTRL0_REG (0x0028)



APB_SARADC_FILTER_CHANNEL1 配置 SAR ADC 滤波通道 1。(R/W)

APB_SARADC_FILTER_CHANNEL0 配置 SAR ADC 滤波通道 0。(R/W)

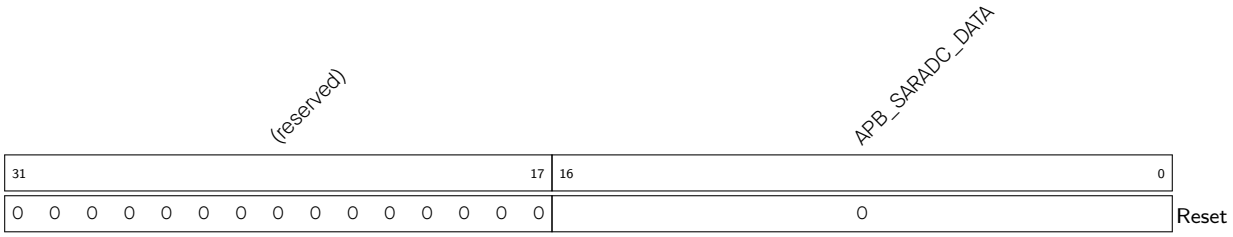
APB_SARADC_FILTER_RESET 配置是否复位 SAR ADC 滤波器。

0：无效

1：复位

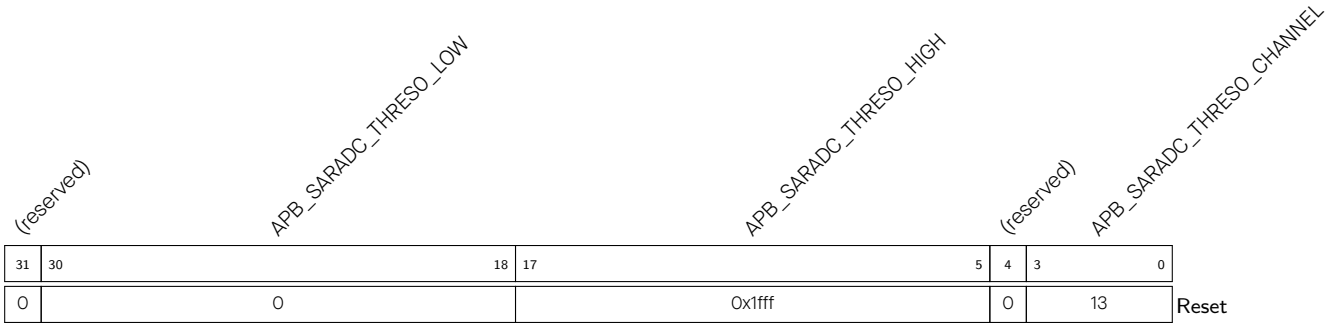
(R/W)

Register 46.8. APB_SARADC_1_DATA_STATUS_REG (0x002C)



APB_SARADC_DATA 存储 SAR ADC 单次采样的转换数据。(RO)

Register 46.9. APB_SARADC_THRES0_CTRL_REG (0x0034)

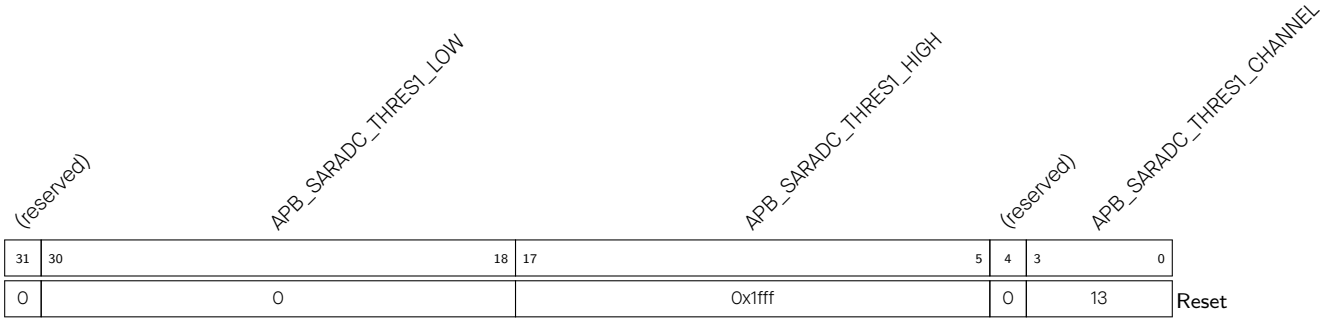


APB_SARADC_THRES0_CHANNEL 配置 SAR ADC 阈值监控器 0 需要监控的通道。(R/W)

APB_SARADC_THRES0_HIGH 配置 SAR ADC 阈值监控器 0 的高阈值。(R/W)

APB_SARADC_THRES0_LOW 配置 SAR ADC 阈值监控器 0 的低阈值。(R/W)

Register 46.10. APB_SARADC_THRES1_CTRL_REG (0x0038)

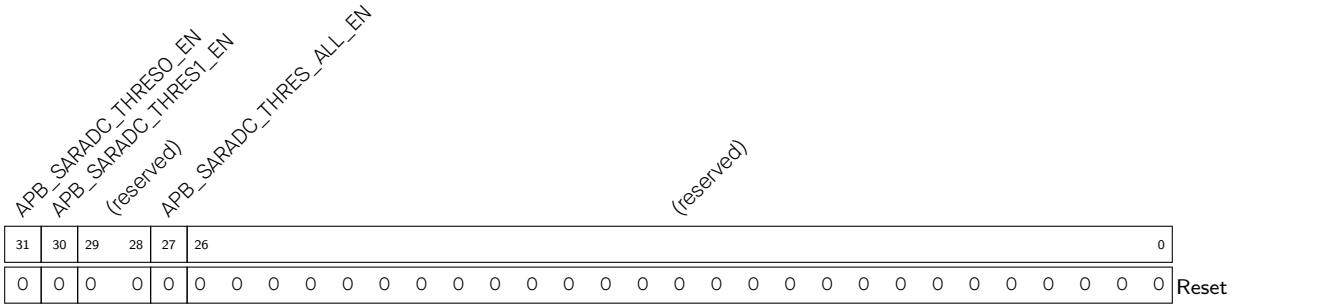


APB_SARADC_THRES1_CHANNEL 配置 SAR ADC 阈值监控器 1 需要监控的通道。(R/W)

APB_SARADC_THRES1_HIGH 配置 SAR ADC 阈值监控器 1 的高阈值。(R/W)

APB_SARADC_THRES1_LOW 配置 SAR ADC 阈值监控器 1 的低阈值。(R/W)

Register 46.11. APB_SARADC_THRES_CTRL_REG (0x003C)



APB_SARADC_THRES_ALL_EN 配置是否使能所有已配置通道的阈值监控。

0：不使能

1：使能

(R/W)

APB_SARADC_THRES1_EN 配置是否使能阈值监控器 1。

0：不使能

1：使能

(R/W)

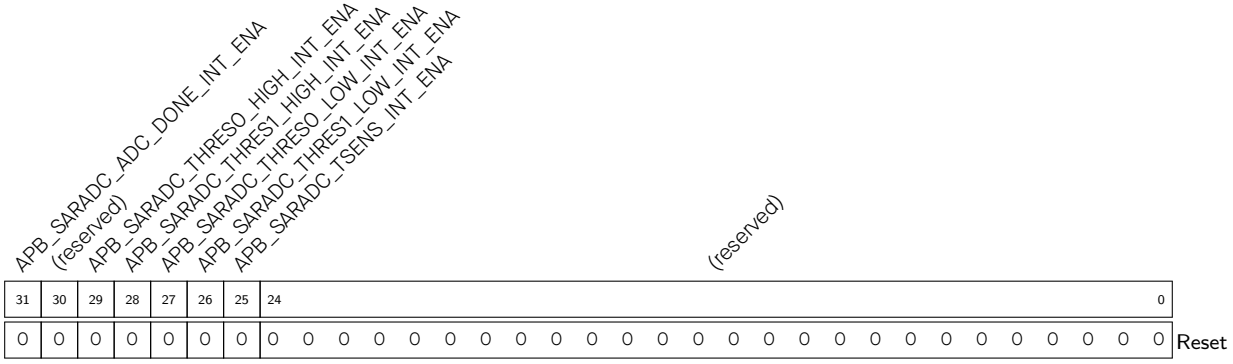
APB_SARADC_THRES0_EN 配置是否使能阈值监控器 0。

0：不使能

1：使能

(R/W)

Register 46.12. APB_SARADC_INT_ENA_REG (0x0040)



APB_SARADC_TSENS_INT_ENA 写 1 使能 APB_SARADC_TSENS_INT [TBA] 中断。(R/W)

APB_SARADC_THRES1_LOW_INT_ENA 写 1 使能 APB_SARADC_THRES1_LOW_INT 中断。(R/W)

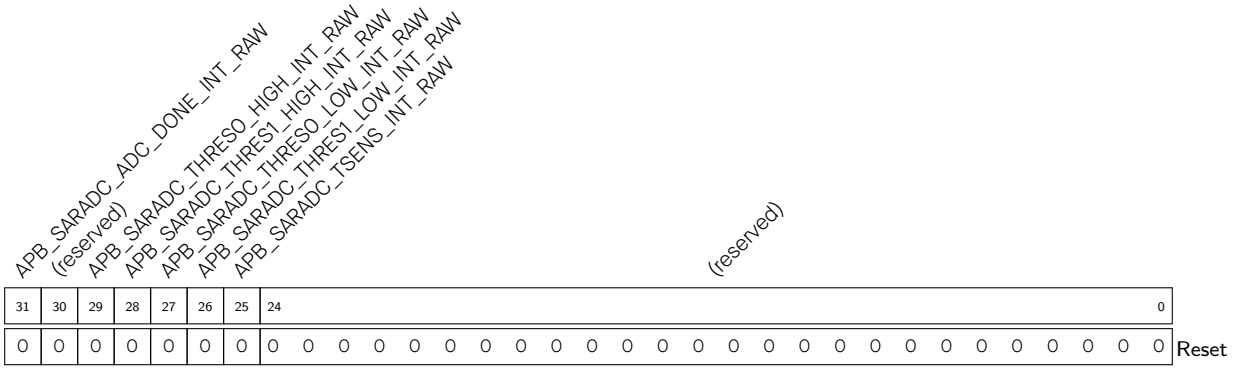
APB_SARADC_THRESO_LOW_INT_ENA 写 1 使能 APB_SARADC_THRESO_LOW_INT 中断。(R/W)

APB_SARADC_THRES1_HIGH_INT_ENA 写 1 使能 APB_SARADC_THRES1_HIGH_INT 中断。(R/W)

APB_SARADC_THRESO_HIGH_INT_ENA 写 1 使能 APB_SARADC_THRESO_HIGH_INT 中断。(R/W)

APB_SARADC_ADC_DONE_INT_ENA 写 1 使能 APB_SARADC_ADC_DONE_INT 中断。(R/W)

Register 46.13. APB_SARADC_INT_RAW_REG (0x0044)



APB_SARADC_TSENS_INT_RAW APB_SARADC_TSENS_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRES1_LOW_INT_RAW APB_SARADC_THRES1_LOW_INT 的原始中断状态。(R/WTC/SS)

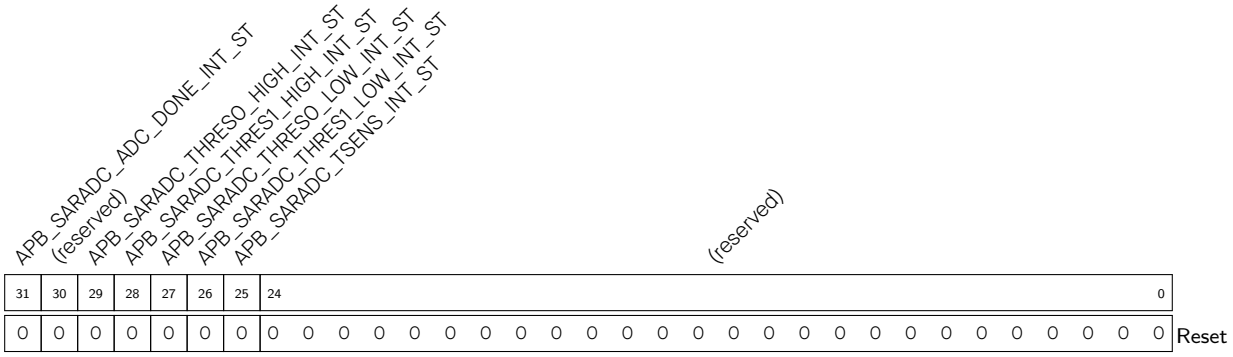
APB_SARADC_THRES0_LOW_INT_RAW APB_SARADC_THRES0_LOW_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRES1_HIGH_INT_RAW APB_SARADC_THRES1_HIGH_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRES0_HIGH_INT_RAW APB_SARADC_THRES0_HIGH_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_ADC_DONE_INT_RAW APB_SARADC_ADC_DONE_INT 的原始中断状态。(R/WTC/SS)

Register 46.14. APB_SARADC_INT_ST_REG (0x0048)



APB_SARADC_TSENS_INT_ST APB_SARADC_TSENS_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES1_LOW_INT_ST APB_SARADC_THRES1_LOW_INT 的屏蔽中断状态。(RO)

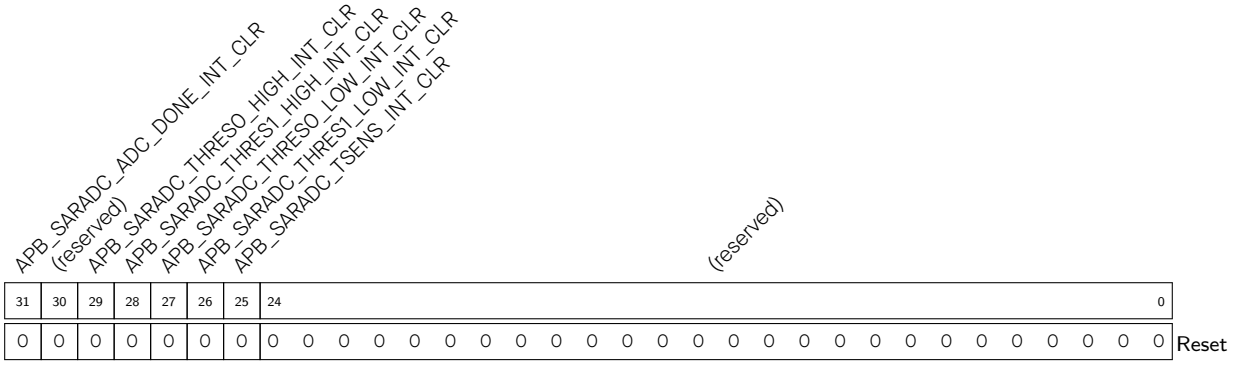
APB_SARADC_THRES0_LOW_INT_ST APB_SARADC_THRES0_LOW_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES1_HIGH_INT_ST APB_SARADC_THRES1_HIGH_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES0_HIGH_INT_ST APB_SARADC_THRES0_HIGH_INT 的屏蔽中断状态。(RO)

APB_SARADC_ADC_DONE_INT_ST APB_SARADC_ADC_DONE_INT 的屏蔽中断状态。(RO)

Register 46.15. APB_SARADC_INT_CLR_REG (0x004C)



APB_SARADC_TSENS_INT_CLR 写 1 清除 APB_SARADC_TSENS_INT 中断。(WT)

APB_SARADC_THRES1_LOW_INT_CLR 写 1 清除 APB_SARADC_THRES1_LOW_INT 中断。(WT)

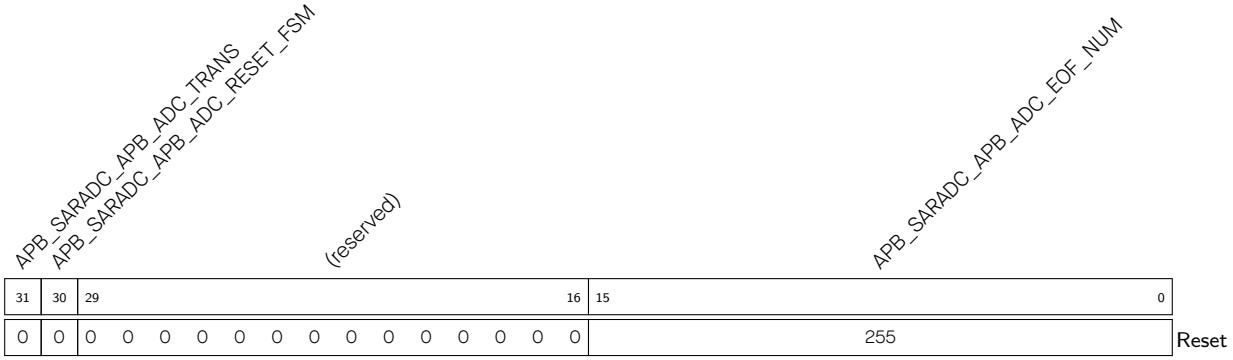
APB_SARADC_THRES0_LOW_INT_CLR 写 1 清除 APB_SARADC_THRES0_LOW_INT 中断。(WT)

APB_SARADC_THRES1_HIGH_INT_CLR 写 1 清除 APB_SARADC_THRES1_HIGH_INT 中断。(WT)

APB_SARADC_THRES0_HIGH_INT_CLR 写 1 清除 APB_SARADC_THRES0_HIGH_INT 中断。(WT)

APB_SARADC_ADC_DONE_INT_CLR 写 1 清除 APB_SARADC_ADC_DONE_INT 中断。(WT)

Register 46.16. APB_SARADC_DMA_CONF_REG (0x0050)



APB_SARADC_APB_ADC_EOF_NUM 当 ADC 采样得到的数据个数等于该寄存器配置值时，发送给 GDMA 的 EOF 标志位将被拉高。

(R/W)

APB_SARADC_APB_ADC_RESET_FSM 配置是否复位 DIG ADC 控制器状态。

0：无效

1：复位

(R/W)

APB_SARADC_APB_ADC_TRANS 配置 DIG ADC 控制器是否使用 GDMA。

0：无效

1：使用 GDMA

(R/W)

第 47 章

模拟电压比较器

47.1 概述

ESP32-C5 芯片集成了一个模拟电压比较器。模拟电压比较器依靠支持电压比较功能的特殊芯片焊盘 (PAD) 实现，用于监测 PAD 上的电压变化。模拟电压比较器有外部两个电压接入点，分为主电压和参考电压。模拟电压比较器输出的电压比较结果可以作为 ETM（事件任务矩阵）事件来驱动其他外设的 ETM 任务，也可以触发中断。

47.2 主要特性

模拟电压比较器具有以下主要特性：

- 支持电压比较功能
  - 电压比较模式可配置
  - 内部参考电压值可配置
- 支持电压比较中断
- 支持 ETM 事件

47.3 系统架构

图 47.3-1 展示了模拟电压比较器的系统架构。

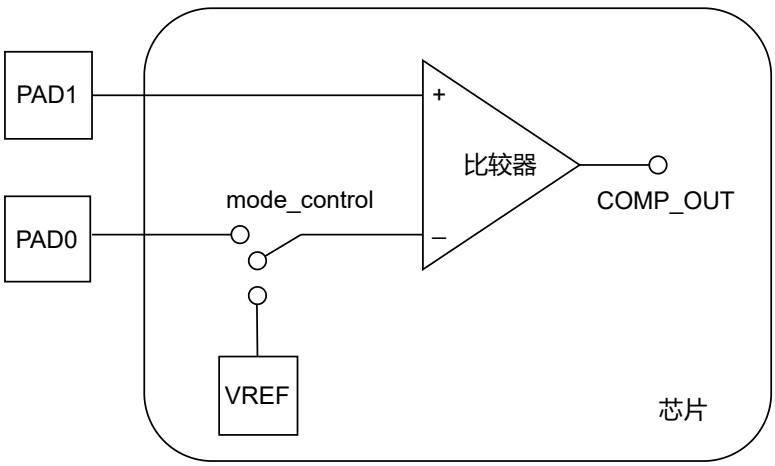


图 47.3-1. 模拟电压比较器系统框图

如上图所示，模拟电压比较器的结构如下：

- 两个 PAD 接口：
  - PAD1：主电压接口
  - PAD0：外部参考电压接口
- VREF：内部参考电压接口
- mode_control：寄存器控制的选择开关，用于选择参考电压的输入源
- COMP_OUT：比较器输出的电压比较结果，为片上内部信号，外部无法直接获取
  - 若主电压高于参考电压，COMP_OUT 信号输出为高电平
  - 若主电压低于参考电压，COMP_OUT 信号输出为低电平

47.4 功能描述

模拟电压比较器具有以下功能：

- **电压比较**

  模拟电压比较器采用的是支持电压比较功能的特殊 PAD，这种特殊 PAD 由两个通用 PAD 组成。下表列出了模拟电压比较器的 PAD 与 GPIO 管脚之间的映射关系。

  表 47.4-1. 模拟电压比较器 PAD 和 GPIO 之间的映射关系

模拟电压比较器	PAD0	PAD1
模拟电压比较器	GPIO8	GPIO9

  电压比较 PAD 的比较模式和参考电压可配置，具体配置流程见小节 47.7 配置流程。

  电压比较结果输出为 COMP_OUT 信号。

- **电压比较模式控制**

  模拟电压比较器将主电压与参考电压进行比较，参考电压可选内部参考电压和外部参考电压。根据参考电压的选择不同，电压比较模式分为两种：
  - 主电压与外部参考电压比较；
  - 主电压与内部参考电压比较；

  电压比较模式可通过 GPIO_EXT_PAD_COMP_CONFIG_0_REG 配置。

- **参考电压配置**

  内部参考电压范围为 0 ~ (0.7 × VDDPST1) V，步长为 0.1 × VDDPST1 V，其中 VDDPST1 为模拟电压比较器的供电电压，等于芯片的电源电压（通常为 3.3 V）。内部参考电压值可通过寄存器 GPIO_EXT_DREF_COMP_0 配置。

  外部参考电压直接从 PAD 接入，无需配置，输入范围与内部参考电压一致，为 0 ~ (0.7 x VDDPST1) V。

- **电压比较中断处理**

  电压比较的输出值 COMP_OUT 信号为高电平或低电平，只要输出值发生变化，就会产生对应的中断信号。

47.5 事件任务矩阵

在 ESP32-C5 中，模拟电压比较器支持 ETM 功能，即可以通过模拟电压比较器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 12 事件任务矩阵 (ETM)。这里仅介绍与模拟电压比较器相关的 ETM 事件。

模拟电压比较器可产生的 ETM 事件有：

- GPIO_EVT_ZERO_DET_POS：表示 COMP_OUT 信号由低电平变为高电平，即主电压从低于参考电压变为高于参考电压。
- GPIO_EVT_ZERO_DET_NEG：表示 COMP_OUT 信号由高电平变为低电平，即主电压从高于参考电压变为低于参考电压。

模拟电压比较器不支持任何 ETM 任务。

47.6 中断

ESP32-C5 的模拟电压比较器可以生成 GPIO_PAD_COMP_INT 中断信号并发送给中断矩阵。

这一中断信号由模拟电压比较器的内部中断源生成。表 47.6-1 列出了模拟电压比较器的内部中断源、中断触发条件、以及生成的中断信号。

表 47.6-1. 模拟电压比较器的内部中断源

内部中断源	触发条件	中断信号
GPIO_COMP_ALL_INT	COMP_OUT 信号发生跳变	GPIO_PAD_COMP_INT
GPIO_COMP_NEG_INT	COMP_OUT 信号由高电平变为低电平	GPIO_PAD_COMP_INT
GPIO_COMP_POS_INT	COMP_OUT 信号由低电平变为高电平	GPIO_PAD_COMP_INT

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 11 中断矩阵 > 11.2 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 8.18.1 *HP GPIO 交换矩阵寄存器列表*。

47.7 配置流程

软件配置电压比较器功能的流程如下：

说明：
有关本节提及的寄存器字段的更多信息，请参阅章节 8 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*。

1. 将 PCR_IOMUX_FUNC_CLK_EN 设置为 1。
2. 将 GPIO_EXT_XPD_COMP_0 设置为 1，启用电压比较器功能。

3. 禁用 PAD1 的正常数字 PAD 功能，包括 IE、OE、WPU 和 WPD，如果使用外部参考电压，则同时禁用 PAD0 的正常数字 PAD 功能。具体的配置方式请参考章节 8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)。
4. 配置比较模式，将 GPIO_EXT_PAD_COMP_CONFIG_0_REG 设置为：
   - 0：选择主电压和内部参考电压进行比较；
   - 1：选择主电压和外部参考电压进行比较。
5. 通过 GPIO_EXT_DREF_COMP_0 配置内部参考电压，可调电压范围为 0 ~ (0.7 × VDDPST1) V，步长为 0.1 × VDDPST1 V。

第 VII 卷

附录

附录部分包括以下内容：

- 相关文档和资源
- 词汇列表
- 如何配置寄存器的保留域
- 中断配置寄存器
- 修订历史

相关文档和资源

相关文档

- 《ESP32-C5 系列芯片技术规格书》- 提供 ESP32-C5 芯片的硬件技术规格。
- 《ESP32-C5 硬件设计指南》– 提供基于 ESP32-C5 芯片的产品设计规范。
- 《ESP32-C5 系列芯片勘误表》– 描述 ESP32-C5 系列芯片的已知错误。
- 技术参考手册 LaTeX 源码 – 乐鑫芯片技术参考手册 LaTeX 源码的公开仓库。
  https://github.com/espressif/esp-technical-reference-manual-latex
- 证书
  https://espressif.com/zh-hans/support/documents/certificates
- ESP32-C5 产品/工艺变更通知 (PCN)
  https://espressif.com/zh-hans/support/documents/pcns?keys=ESP32-C5
- ESP32-C5 公告 – 提供有关安全、bug、兼容性、器件可靠性的信息
  https://espressif.com/zh-hans/support/documents/advisories?keys=ESP32-C5
- 文档更新和订阅通知
  https://espressif.com/zh-hans/support/download/documents

开发者社区

- 《ESP32-C5 ESP-IDF 编程指南》– ESP-IDF 开发框架的文档中心。
- ESP-IDF 及 GitHub 上的其它开发框架
  https://github.com/espressif
- ESP32 论坛 – 工程师对工程师 (E2E) 的社区，您可以在这里提出问题、解决问题、分享知识、探索观点。
  https://esp32.com/
- *ESP-FAQ* – 由乐鑫官方推出的针对常见问题的总结。
  https://espressif.com/projects/esp-faq/zh_CN/latest/index.html
- *The ESP Journal* – 分享乐鑫工程师的最佳实践、技术文章和工作随笔。
  https://blog.espressif.com/
- SDK 和演示、App、工具、AT 等下载资源
  https://espressif.com/zh-hans/support/download/sdks-demos

产品

- ESP32-C5 系列芯片 – ESP32-C5 全系列芯片。
  https://espressif.com/zh-hans/products/socs?id=ESP32-C5
- ESP32-C5 系列模组 – ESP32-C5 全系列模组。
  https://espressif.com/zh-hans/products/modules?id=ESP32-C5
- ESP32-C5 系列开发板 – ESP32-C5 全系列开发板。
  https://espressif.com/zh-hans/products/devkits?id=ESP32-C5
- ESP Product Selector（乐鑫产品选型工具）– 通过筛选性能参数、进行产品对比快速定位您所需要的产品。
  https://products.espressif.com/#/product-selector?language=zh

联系我们

- 商务问题、技术支持、电路原理图 & PCB 设计审阅、购买样品（线上商店）、成为供应商、意见与建议
  https://espressif.com/zh-hans/contact-us/sales-questions

词汇列表

外设相关词汇

ADC	ADC 控制器
AES	AES 加速器
CAN FD	控制器局域网灵活数据速率
RSA_DS	RSA 数字签名外设
ECC	ECC 加速器
ECDSA_DS	ECDSA 数字签名外设
eFuse	eFuse 控制器
ETM	事件任务矩阵
GDMA	通用 DMA 控制器
HMAC	HMAC 加速器
I2C	I2C 控制器
I2S	I2S 控制器
LEDC	LED PWM 控制器
MCPWM	电机控制脉宽调制器
PARLIO	并行 IO 控制器
PCNT	脉冲计数器控制器
PMS	权限控制
RMT	红外遥控
RNG	随机数生成器
RSA	RSA 加速器
SDIO	SDIO 从机控制器
SHA	SHA 加速器
SPI	SPI 控制器
TIMG	定时器组
TRACE	RISC-V Trace 编码器
TSENS	温度传感器
UART	UART 控制器
WDT	看门狗定时器
XTS_AES	片外存储器加密与解密

寄存器相关缩写

REG	**寄存器**。
SYSREG	**系统寄存器**是一组控制系统复位、存储器、时钟、软件中断、电源管理、时钟门控等的寄存器。
ISO	**隔离**。如果外设或其他芯片组件断电，其输出信号的管脚（若有）将会浮空。ISO 寄存器会隔离上述引脚并令其保持在某个确定值，以使连接到这些引脚的其他非断电外设/设备免受影响。
NMI	**非屏蔽中断**是一种 CPU 指令无法禁用或忽略的硬件中断。出现此类中断说明发生严重错误。

W1TS 添加到寄存器/字段名称中的缩写，表示此类寄存器/字段用于置位名称相似寄存器中的相应字段。例如，寄存器 GPIO_ENABLE_W1TS_REG 用于置位寄存器 GPIO_ENABLE_REG 中的相应字段。

W1TC 与 *W1TS* 相同，但用于清除相应寄存器中的字段。

寄存器的访问类型

TRM 章节 寄存器列表 和 寄存器 详述了寄存器及其字段的访问类型。

常用访问类型及组合如下：

- RO
- WO
- WT
- R/W
- R/W1
- WL
- R/W/SC
- R/W/SS
- R/W/SS/SC
- R/WC/SS
- R/WC/SC
- R/WC/SS/SC
- R/WS/SC
- R/WS/SS
- R/WS/SS/SC
- R/SS/WTC
- R/SC/WTC
- R/SS/SC/WTC
- RF/WF
- R/SS/RC
- varies

下文提供了所有访问类型的具体描述。

R	**软件可读**。用户软件可以读取此寄存器/字段；通常与其他访问类型结合使用。
RO	**软件只读**。用户软件只可读取此寄存器/字段。
HRO	**硬件只读**。仅硬件可以读取此寄存器/字段；用于存储变量参数的默认设置。
W	**软件可写**。用户软件可以写入此寄存器/字段；通常与其他访问类型结合使用。
WO	**软件只写**。用户软件只可写入此寄存器/字段。
W1	**软件只写一次**。用户软件只可写入一次此寄存器/字段；只允许写入 1，写入 0 无效。
SS	**硬件置位**。在指定事件中，硬件自动将 1 写入此寄存器/字段；与一位字段一同使用。
SC	**硬件清零**。在指定事件中，硬件自动将 0 写入此寄存器/字段；与一位和多位字段一同使用。
SM	**硬件修改**。在指定事件中，硬件自动将指定值写入此寄存器/字段；与多位字段一同使用。
SU	**硬件更新**。在指定事件中，硬件自动更新此寄存器/字段；与多位字段一同使用。
RS	**软件读置位**。如果用户软件读取此寄存器/字段，硬件会自动写 1。
RC	**软件读清零**。如果用户软件读取此寄存器/字段，硬件会自动写 0。
RF	**软件读 FIFO**。如果用户软件将新数据写入 FIFO，寄存器/字段会自动读取。
WF	**软件写 FIFO**。如果用户软件将新数据写入此寄存器/字段，寄存器/字段会自动通过 APB 总线将数据传递到 FIFO。
WS	**软件写置位**。如果用户软件写入此寄存器/字段，硬件会自动置位此寄存器/字段。
W1S	**软件写 1 置位**。如果用户软件将 1 写入此寄存器/字段，硬件会自动置位此寄存器/字段。
WOS	**软件写 0 置位**。如果用户软件将 0 写入此寄存器/字段，硬件会自动置位此寄存器/字段。
WC	**软件写清零**。如果用户软件写入此寄存器/字段，硬件会自动清零此寄存器/字段。
W1C	**软件写 1 清零**。如果用户软件将 1 写入此寄存器/字段，硬件会自动清零此寄存器/字段。
WOC	**软件写 0 清零**。如果用户软件将 0 写入此寄存器/字段，硬件会自动清零此寄存器/字段。
WT	**软件写产生边沿触发信号**。如果用户软件将 1 写入此字段，将会产生边沿触发信号（APB 总线中的脉冲）或清除相应的 WTC 字段（详见 WTC）。

WTC **软件写其他寄存器位清零本寄存器位**。如果用户软件将 1 写入相应的 WT 字段，硬件会自动清除此字段（详见 WT）。

W1T **软件写 1 取反**。如果用户软件将 1 写入此字段，硬件会自动取反相应字段，否则不会取反。

W0T **软件写 0 取反**。如果用户软件将 0 写入此字段，硬件会自动取反相应字段，否则不会取反。

WL **软件仅在锁禁用时写**。如果锁被禁用，用户软件可以写入此寄存器/字段。

varies **访问类型不定**。此寄存器中的不同字段访问类型可能不同。

如何配置寄存器的保留域

概述

寄存器的保留域指的是寄存器中不对用户开放、或者配置为非默认值时会导致不可预测的结果的域。

如何配置保留域

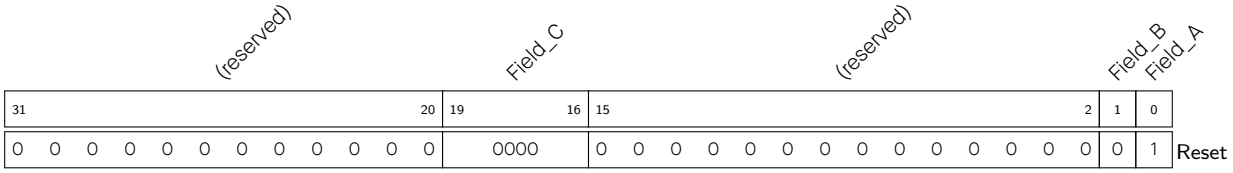
保留域的值不能修改。由于写寄存器时必须整体写，不能只写部分域，因此，在写带有保留域的寄存器时需要特别注意，只能采用以下两种方式：

1. 读取寄存器的值，仅修改需要配置的域，然后将修改后的值以及其他未修改的值一起写回寄存器，这样保留域的值就会保持不变。

   或者

2. 仅修改需要配置的域，然后将保留域的默认值写回寄存器。默认值即寄存器图表中的“Reset”值。例如，寄存器 X 中 Field_A 的默认值为 1。

Register 47.1. 寄存器 X (地址)



假设您要将 寄存器 X 的 Field_A、Field_B 和 Field_C 设置为 0x0、0x1 和 0x2，您可以：

- 使用第 1 种方式，修改这三个域的值，然后把读到的值写回寄存器。假设寄存器读取的值为 0x0000_0003，修改三个域的值后，将值 0x0002_0002 写入寄存器。
- 使用第 2 种方式，修改这三个域的值，然后把保留域的默认值写回寄存器，即把 0x0002_0002 写入寄存器。

中断配置寄存器

大部分外设的内部中断源都有以下配置寄存器：

- **RAW**（原始状态）寄存器：该寄存器指示原始中断状态，每个位对应一个内部中断源。当中断源触发时，其 RAW 位为 1。
- **ENA**（使能）寄存器：该寄存器用于启用或禁用内部中断源，每个位对应一个内部中断源。

  通过操作 ENA 寄存器，可以根据需要屏蔽或取消屏蔽某个内部中断源。当中断源被屏蔽（禁用）时，它不会生成中断信号，但仍可以从 RAW 寄存器中读取其值。
- **ST**（状态）寄存器：该寄存器指示中断源的屏蔽状态，每个位对应一个内部中断源。ST 位为 1 代表 RAW 位和 ENA 位都为 1，即中断源已生成且未被屏蔽。RAW 位和 ENA 位的值为其他组合时，ST 位为 0。

  ENA/RAW/ST 寄存器的配置见表 47.7-4。
- **CLR**（清除）寄存器：CLR 寄存器负责清除内部中断源。写 1 将清除该位对应的中断源。

表 47.7-4. ENA/RAW/ST 寄存器的配置

ENA 位的值	RAW 位的值	ST 位的值
0	忽略	0
1	0	0
	1	1

修订历史

日期	版本	发布说明
2026-04-24	v1.1	更新以下章节： • 章节 2 高性能处理器：新增 mhcr 寄存器；更新 mintthresh 寄存器域；更正 CPU 时钟频率 • 章节 9 复位和时钟：更新 PCR_32K_SEL 的描述；新增 PCR_SPI2_CLKM_DIV_NUM 域的描述 • 章节 24 HMAC 加速器 (HMAC)：新增寄存器 HMAC_WR_MESSAGE_*n*_REG 和 HMAC_RD_RESULT_*n*_REG • 章节 27 RSA 数字签名外设 (RSA_DS)：将模块从数字签名算法 (DSA) 更名为 RSA 数字签名外设 (RSA_DS) • 章节 28 ECDSA 数字签名外设 (ECDSA_DS)：将模块从椭圆曲线数字签名算法 (ECDSA) 更名为 ECDSA 数字签名外设 (ECDSA_DS) • 章节 32 UART 控制器 (UART)：更新 UART*n* 模块初始化和 UART*n* 通信配置编程流程 • 章节 38 CAN FD：补充关于 CAN FD 协议兼容性的说明；更新 38.2 主要特性；补充时钟源选择建议；更新图 38.3-8 TX buffer 状态 和图 38.3-1 TWAI_CLK Diagram；更新 38.5 配置示例 • 章节 42 红外遥控 (RMT)：更新 RMT 时钟频率限制相关公式
2026-01-27	v1.0	新增以下章节： • 章节 1 总线架构 • 章节 2 高性能处理器 • 章节 13 低功耗管理 • 章节 31 密钥管理器 • 章节 38 CAN FD 更新以下章节： • 章节 5 通用 DMA 控制器 (GDMA)：新增 总线错误响应信息记录 和 自动时钟门控 相关描述和寄存器 • 章节 8 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)：新增 GPIO_SDIO_INT 中断描述 • 章节 9 复位和时钟 和章节 40 LED PWM 控制器 (LEDC)：更新 XTAL_CLK 时钟频率 • 章节 10 芯片 Boot 控制：移除晶振频率选择相关信息

见下页

接上页

日期	版本	发布说明
2025-08-22	v0.2	新增以下章节： • 章节 7 *eFuse 控制器 (EFUSE)* • 章节 11 中断矩阵 • 章节 19 系统寄存器 • 章节 33 *SPI 控制器 (SPI)* • 章节 36 脉冲计数控制器 *(PCNT)* 更新以下章节： • 章节 4 低功耗处理器：更新 LP CPU 和 HP CPU 原子访问的限制 • 章节 8 *GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)*：更新寄存器 GPIO_STRAP_REG 的描述 • 章节 39 *SDIO 从机控制器 (SDIO)* 和章节 37 *USB 串口/JTAG 控制器*：说明 SDIO 从机控制器与 USB 串口/JTAG 控制器的复用限制
2025-05-20	v0.1	首次发布



免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，乐鑫不对信息的准确性、真实性做任何保证。

乐鑫不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他乐鑫提案、规格书或样品在他处提到的任何保证。

乐鑫不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

版权归 © 2026 乐鑫信息科技（上海）股份有限公司。保留所有权利。

www.espressif.com