

ESP8266

低功耗解决方案



版本 1.5
版权 © 2019

关于本手册

本文向用户介绍了 ESP8266 系列芯片的睡眠模式和低功耗解决方案。

发布说明

日期	版本	发布说明
2015.06	V1.0	首次发布。
2016.04	V1.1	增加章节 4.5 低功耗解决方案。
2016.07	V1.2	增加关于 Light-sleep 的注意事项。
2016.08	V1.3	更新表 1-1 三种睡眠模式比较； 增加对于 Deep-sleep 特性的说明事项； 增加章节 4.3.1 自动唤醒。
2018.05	V1.4	更新章节 3.3 外部唤醒。
2019.08	V1.5	更新表 1-1 三种睡眠模式比较。

文档变更通知

用户可通过[乐鑫官网](#)订阅技术文档变更的电子邮件通知。

证书下载

用户可通过[乐鑫官网](#)下载产品证书。

目录

1. 概述	1
2. Modem-sleep	2
2.1. 特性.....	2
2.2. 接口说明.....	2
2.2.1. 自动休眠.....	2
2.2.2. 强制休眠.....	2
2.3. 应用.....	2
3. Light-sleep	3
3.1. 特性.....	3
3.2. 接口说明.....	3
3.2.1. 自动休眠.....	3
3.2.2. 强制休眠.....	3
3.3. 外部唤醒.....	3
3.4. 应用.....	4
4. Deep-sleep	5
4.1. 特性.....	5
4.2. 接口说明.....	5
4.2.1. 使能 Deep-sleep.....	5
4.2.2. 配置 Deep-sleep.....	5
4.3. 唤醒.....	6
4.3.1. 自动唤醒.....	6
4.3.2. 外部唤醒.....	6
4.4. 应用.....	6
4.5. 低功耗解决方案.....	6



1.

概述

ESP8266 系列芯片提供三种可配置的睡眠模式，针对这些睡眠模式，我们提供了多种低功耗解决方案，用户可以结合具体需求选择睡眠模式并进行配置。三种睡眠模式如下：

- Modem-sleep
- Light-sleep
- Deep-sleep

三种模式的区别如表 1-1 所示。

表 1-1. 三种睡眠模式比较

项目	Modem-sleep		Light-sleep		Deep-sleep
	自动	强制	自动	强制	强制
Wi-Fi 连接	保持	断连	保持	断连	断连
GPIO 状态	未定义		未定义		未定义 (2 μ A)
Wi-Fi	关闭		关闭		关闭
系统时钟	开启		关闭		关闭
RTC	开启		开启		开启
CPU	开启		暂停		关闭
衬底电流	15 mA		0.4 mA		~ 20 μ A
平均电流	DTIM = 1	16.2 mA	1.8 mA		
	DTIM = 3	15.4 mA	0.9 mA		-
	DTIM = 10	15.2 mA	0.55 mA		

说明：

- 对于 Modem-sleep 和 Light-sleep 模式，SDK 提供接口来使能睡眠模式，并由系统底层决定何时进入睡眠。具体请参考第 2 章 Modem-sleep 和第 3 章 Light-sleep。
- 在 Deep-sleep 模式下，何时进入睡眠由用户控制，调用接口函数就可立即进入 Deep-sleep 模式。具体请参考第 4 章 Deep-sleep。
- RTC (Real-Time Clock)：实时时钟。
- DTIM (Delivery Traffic Indication Message)：使用无线路由器时无线发送数据包的频率。



2. Modem-sleep

2.1. 特性

目前 ESP8266 的 Modem-sleep 仅工作在 Station 模式下，连接路由器后生效。ESP8266 通过 Wi-Fi 的 DTIM Beacon 机制与路由器保持连接。

说明：

一般路由器的 DTIM Beacon 间隔为 100 ms ~ 1,000 ms。

在 Modem-sleep 模式下，ESP8266 会在两次 DTIM Beacon 间隔时间内，关闭 Wi-Fi 模块电路，达到省电效果，在下次 Beacon 到来前自动唤醒。睡眠时间由路由器的 DTIM Beacon 时间决定。睡眠同时可以保持与路由器的 Wi-Fi 连接，并通过路由器接受来自手机或者服务器的交互信息。

2.2. 接口说明

2.2.1. 自动休眠

系统通过以下接口进入 Modem-sleep 模式。

```
wifi_set_sleep_type(MODEM_SLEEP_T)
```

说明：

在 Modem-sleep 模式下，系统可以自动被唤醒，无需配置接口。

2.2.2. 强制休眠

用户可以使系统强制进入 Modem-sleep 模式，即调用强制休眠接口，强制关闭射频。关于强制休眠接口的说明，请参考 [《ESP8266 Non-OS SDK API 参考》](#) 中的 [3.7 节 强制休眠接口](#) 和 [ESP8266 RTOS SDK API Reference](#) 中的 [4.12 节 Force Sleep APIs](#)。

注意：

强制休眠接口调用后，并不会立即休眠，而是等到系统空闲任务执行时才进入休眠。

2.3. 应用

Modem-sleep 一般用于必须打开芯片 CPU 的应用场景，例如 PWM 彩灯，需要 CPU 实时控制。



3. Light-sleep

3.1. 特性

Light-sleep 的工作模式与 Modem-sleep 相似，不同的是，除了关闭 Wi-Fi 模块电路以外，在 Light-sleep 模式下，还会关闭时钟并暂停内部 CPU，比 Modem-sleep 功耗更低。

⚠ 注意：

在 Light-sleep 之前把处于输出状态的管脚改为输入状态，比如：MTDO、U0TXD、GPIO0，消除管脚上的漏电，可使 Light-sleep 的功耗更低。

3.2. 接口说明

3.2.1. 自动休眠

系统通过以下接口进入 Light-sleep 模式。

```
wifi_set_sleep_type(LIGHT_SLEEP_T)
```

📖 说明：

在 Wi-Fi 连接后，并且 CPU 处于空闲状态时，会自动进入 Light-sleep 状态。

3.2.2. 强制休眠

用户可以使系统强制进入 Light-sleep 模式，即调用强制休眠接口，强制关闭射频。关于强制休眠接口的说明，请参考 [《ESP8266 Non-OS SDK API 参考》](#) 中的 [3.7 节 强制休眠接口](#) 和 [ESP8266 RTOS SDK API Reference](#) 中的 [4.12 节 Force Sleep APIs](#)。

⚠ 注意：

强制休眠接口调用后，并不会立即休眠，而是等到系统空闲任务执行时才进入休眠。

3.3. 外部唤醒

在 Light-sleep 模式下，CPU 在暂停状态下不会响应来自外围硬件接口的信号与中断，因此需要通过外部 GPIO 信号将 ESP8266 唤醒，硬件唤醒过程大约为 3 ms。由于 Wi-Fi 初始化过程需要大约 1 ms，所以建议用户 5 ms 之后再对芯片进行操作。

通过 GPIO 唤醒只能配置为电平触发模式，接口如下。

```
void wifi_enable_gpio_wakeup(uint32 i, GPIO_INT_TYPE intr_state);
```



例如：设置 GPIO12 为唤醒 GPIO 管脚。

```
GPIO_DIS_OUTPUT(12);
PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U, FUNC_GPIO12);
wifi_enable_gpio_wakeup(12, GPIO_PIN_INTR_LOLEVEL);
```

uint32 i 唤醒功能的 IO 序号。

GPIO_INT_TYPE
intr_state 唤醒的触发模式。

- GPIO_PIN_INTR_LOLEVEL
- GPIO_PIN_INTR_HILEVEL

说明：

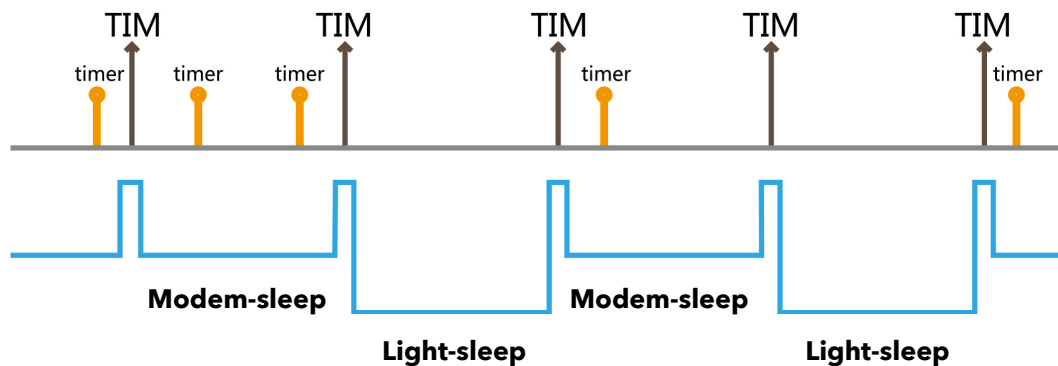
GPIO16 不能用于唤醒。

3.4. 应用

Light-sleep 模式可用于需要保持与路由器的连接，可以实时响应路由器发来的数据的情况。并且在未接收到命令时，CPU 可以处于空闲状态。比如 Wi-Fi 开关的应用，大部分时间 CPU 都是空闲的，直到收到控制命令，CPU 才需要进行 GPIO 的操作。

说明：

若系统应用中有小于 DTIM Beacon 间隔时间的循环定时，系统将不能进入 Light-sleep 模式，如下图所示。





4. Deep-sleep

4.1. 特性

相对于其他两种模式，系统无法自动进入 Deep-sleep，需要由用户调用接口函数 `system_deep_sleep` 来控制。在该模式下，芯片会断开所有 Wi-Fi 连接与数据连接，进入睡眠模式，只有 RTC 模块仍然工作，负责芯片的定时唤醒。

说明：

在 Deep-sleep 状态下，GPIO 电平状态可以保持，具有 2 μ A 的驱动能力。

4.2. 接口说明

4.2.1. 使能 Deep-sleep

通过以下接口使能 Deep-sleep。

```
void system_deep_sleep(uint32 time_in_us)
```

参数说明：

<code>uint32 time_in_us = 0</code>	不会定时唤醒，即不会主动醒来。
<code>uint32 time_in_us \neq 0</code>	会在设定的时间后，自动唤醒（单位： μ s）。

4.2.2. 配置 Deep-sleep

可以通过以下接口配置 Deep-sleep 唤醒时的软件工作流程，从而影响长期运行的平均功耗。

```
bool system_deep_sleep_set_option(uint8 option)
```

<code>deep_sleep_set_option(0)</code>	由 <code>init</code> 参数的第 108 字节控制 Deep-sleep 唤醒时是否作射频校准。
<code>deep_sleep_set_option(1)</code>	表示下一次 Deep-sleep 唤醒时要作射频校准，功耗较大。
<code>deep_sleep_set_option(2)</code>	表示下一次 Deep-sleep 唤醒时不作射频校准，功耗较小。
<code>deep_sleep_set_option(4)</code>	表示下一次 Deep-sleep 唤醒时不打开射频，和 Modem-sleep 一样，电流最小。

**说明：**

init 参数即 `esp_init_data_default.bin` 内的参数值。比如，将第 108 字节的数据改为 8，并且调用 `deep_sleep_set_option(0)`，则表示芯片每 8 次 Deep-sleep 唤醒才会进行射频校准。用户可以参考 ESP8266 低功耗传感器示例应用，链接如下：

https://github.com/EspressifSystems/low_power_voltage_measurement/wiki。

4.3. 唤醒

4.3.1. 自动唤醒

在 Deep-sleep 状态下，可以将 GPIO16 (XPD_DCDC) 连接至 EXT_RSTB。计时到达睡眠时间后，GPIO16 输出低电平给 EXT_RSTB 管脚，芯片即可被重置并被唤醒。

4.3.2. 外部唤醒

在 Deep-sleep 状态下，可以通过外部 IO 在芯片 EXT_RSTB 管脚上产生一个低电平脉冲，芯片即可被重置并被唤醒。

注意：

如果自动唤醒与外部唤醒须要同时作用，须要在外部电路设计时，使用合适的线逻辑操作电路。

4.4. 应用

Deep-sleep 可以用于低功耗的传感器应用，或者大部分时间都不需要进行数据传输的情况。设备可以每隔一段时间从 Deep-sleep 状态醒来测量数据并上传，之后继续进入 Deep-sleep。也可以将多个数据存储于 RTC memory（RTC memory 在 Deep-sleep 模式下仍然可以保存数据），然后一次发送出去。

4.5. 低功耗解决方案

我们提供了以下 8 种方案来减少 Deep-sleep 模式的功耗。

1. 设置立即进入 Deep-sleep，缩短设备进入 Deep-sleep 的时间。

函数定义：

```
void system_deep_sleep_instant(uint32 time_in_us)
```

示例代码：

```
//Deep-sleep for 5 seconds, and then wake up  
system_deep_sleep_instant(5000*1000);
```

**说明：**

函数 `system_deep_sleep_instant` 未外部声明，但可以直接调用。

2. 设置 Deep-sleep 唤醒后不进行射频校准，以缩短芯片初始化的时间和降低芯片初始化时的电流。

```
system_deep_sleep_set_option(2);
```

3. 适当降低射频功耗。

如果应用场景不需要较高的 Tx 的峰值，可以适当降低射频功耗。

使用 [ESP8266 Download Tool](#) (v1.2 及以上版本)，在 **RF InitConfig** 页签修改射频功耗。修改后生成的 `esp_init_data_setting.bin` 文件，可以替换 `esp_init_data_default.bin` 文件并下载。

4. 使用以下指令修改 BIN 文件，缩短 flash 初始化的时间并降低 flash 初始化时的电流。

```
python add_low-power_deepsleep_cmd.py ./bin file
```

若使用支持在线升级 (OTA) 的固件，使用脚本修改 `boot_v1.5.bin` 后生成 `boot_v1.5_low_power.bin` 下载到 `0x0` 地址。

如果使用不支持在线升级 (Non-OTA) 的固件，使用脚本修改 `eagle.flash.bin` 后生成 `eagle.flash_low_power.bin` 下载到 `0x0` 地址。

说明：

用户可以通过如下链接下载脚本文件 `Add_Low-power_Cmd`：

<https://www.espressif.com/en/support/download/other-tools?keys=ESP8266+Add+Low-power+Cmd>

5. 选择 Flash 型号和工作模式。

选择特殊型号的 flash，可明显缩短从 flash 加载固件的时间，例如 ISSI-IS25LQ025。适当的工作模式，也可以缩短从 flash 加载固件的时间，建议 flash 工作模式尽量选择四线工作模式。

6. 设置清空 UART FIFO，减少打印时间。

FIFO（先入先出队列）是 UART 的缓存器，强制将串行通信的每个字节按照接收的顺序进行传送。FIFO 打印消耗时间较多，尽量避免大量打印。所以在设置休眠前应清空 UART FIFO，否则，系统会等待 UART FIFO 打印结束才进入睡眠，将消耗较多时间。

```
SET_PERI_REG_MASK(UART_CONF0(0), UART_TXFIFO_RST); // RESET FIFO
```

```
CLEAR_PERI_REG_MASK(UART_CONF0(0), UART_TXFIFO_RST);
```

7. 集中发包。



因为发包动作持续时间短（与睡眠醒来的时间相比）、消耗能量少，建议集中发送数据，在 Deep-sleep 醒来之后，一次发送多个数据包。

8. **esp_iot_sdk_v1.4.0**、**esp_iot_rtos_sdk_v1.3.0** 及其之后版本优化了降低功耗的能力，请使用新版本的 SDK。

说明：

实际测试中发现，由于 Light-sleep 硬件唤醒时间短（约为 3 ms），如果应用场景中设备睡眠时间 $< 2s$ ，宜采用 Light-sleep 工作模式，更节省功耗；如果睡眠时间 $> 2s$ ，则宜采用 Deep-sleep 工作模式，更节省功耗。



免责声明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，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© 2019 乐鑫所有。保留所有权利。