ESP8266
Low-Power Solutions

Version 1.5

Copyright © 2019

About This Guide

This document introduces the ESP8266 sleep modes and some relevant low-power solutions.

Release Notes

Date	Version	Release notes
2015.06	V1.0	First release.
2016.04	V1.1	Added **Section 4.5 Low Power Solutions**.
2016.07	V1.2	Added Notice for Light-sleep.
2016.08	V1.3	Updated **Table 1-1 Differences Between the Three Sleep Modes**; Added notes on Deep-sleep; Added **Section 4.3.1 Automatic Wake-up**.
2018.05	V1.4	Updated **Section 3.3 External Wake-up**.
2019.08	V1.5	Updated **Table 1-1. Differences Between the Three Sleep Modes.**

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe *here*.

Certification

Download certificates for Espressif products from *here*.

Table of Contents

1. Overview

ESP8266 supports three configurable sleep modes, as well as a number of low-power solutions. Users can choose and configure the sleep mode, as required. The three sleep modes are:

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

Table 1-1 shows the differences between the three sleep modes.

Table 1-1. Differences Between the Three Sleep Modes

Item		Modem-sleep		Light-sleep		Deep-sleep
		Automatic	Forced	Automatic	Forced	Forced
Wi-Fi connectivity		Connected	Disconnected	Connected	Disconnected	Disconnected
GPIO state		Undefined		Undefined		Undefined (2 μA)
Wi-Fi		OFF		OFF		OFF
System clock		ON		OFF		OFF
RTC		ON		ON		ON
CPU		ON		Pending		OFF
Substrate current		15 mA		0.4 mA		~ 20 μA
Average current	DTIM = 1	16.2 mA		1.8 mA		
	DTIM = 3	15.4 mA		0.9 mA		-
	DTIM = 10	15.2 mA		0.55 mA		

📖 **Notes:**

- The SDK provides interfaces to enable Modem-sleep and Light-sleep modes, while the system's bottom layer determines when the entire system gets into sleep mode. For details, please refer to **Chapter 2 Modem-sleep** and **Chapter 3 Light-sleep** of the present document.

- The Deep-sleep mode is controlled by users. Users can call the function to enable Deep-sleep instantly. For details, please refer to **Chapter 4 Deep-sleep**.

- RTC: Real-Time Clock.

- DTIM: Delivery Traffic Indication Message.

2.   Modem-sleep

2.1.   Features

Modem-sleep mode is enabled only when ESP8266 connects to the router in station mode. ESP8266 stays connected to the router through the DTIM beacon mechanism.

> 📖 **Note:**
>
> The DTIM Beacon interval of the router usually ranges from 100 ms to 1,000 ms.

In Modem-sleep mode, ESP8266 will close the Wi-Fi module circuit between the two DTIM Beacon intervals in order to save power. ESP8266 will be automatically woken up before the next Beacon arrival. The sleep time is determined by the DTIM Beacon interval time of the router. During sleep mode, ESP8266 can stay connected to the Wi-Fi and receive the interactive information from a mobile phone or server.

2.2.   Interface

2.2.1.   Auto Modem-sleep

The system gets into Modem-sleep via the following interface:

```
wifi_set_sleep_type(MODEM_SLEEP_T)
```

> 📖 **Note:**
>
> In Modem-sleep, the system can be woken up automatically. Users do not need to configure the interface.

2.2.2.   Forced Modem-sleep

Users can enable Modem-sleep by calling force-sleep APIs and turning off RF. For details on force-sleep APIs, please refer to **Section 3.7 Force Sleep APIs** in *ESP8266 Non-OS SDK API Reference* and **Section 4.12 Force Sleep APIs** in *ESP8266 RTOS SDK API Reference*.

> ⚠️ **Notice:**
>
> The system will not enter sleep mode instantly when force-sleep APIs are called, but only after executing an idle task.

2.3.   Application

Modem-sleep is generally used in the applications that need the CPU powered on. An example of such applications is the Pulse Width Modulation (PWM) light, which needs real-time CPU control.

3.   Light-sleep

3.1. Features

The functioning mode of Light-sleep mode is similar to that of Modem-sleep. The only difference is that during Light-sleep mode, ESP8266 powers off both the clock and the Wi-Fi circuit, and also suspends the internal CPU, which results in even less power consumption than in Modem-sleep mode.

> ⚠ **Notice:**
>
> Set the pins with an output status to input status, i.e., MTDO, U0TXD and GPIO0, before enabling Light-sleep to eliminate the leakage current, so that the power consumption becomes even lower.

3.2. Interface

3.2.1. Auto Light-sleep

The system goes into Light-sleep mode via the following interface:

```
wifi_set_sleep_type(LIGHT_SLEEP_T)
```

> 📖 **Note:**
>
> ESP8266 automatically enters Light-sleep mode when getting connected to Wi-Fi, as the CPU becomes idle at that point.

3.2.2. Forced Light-sleep

Users can enable Light-sleep by calling force-sleep APIs and turning off RF. For details on force-sleep APIs, please refer to **Section 3.7 Force Sleep APIs** in *ESP8266 Non-OS SDK API Reference* and **Section 4.12 Force Sleep APIs** in *ESP8266 RTOS SDK API Reference*.

> ⚠ **Notice:**
>
> The system will not enter sleep mode instantly when force-sleep APIs are called, but only after executing an idle task.

3.3. External Wake-up

During Light-sleep, the CPU is suspended and does not respond to the signals and interrupts from the peripheral hardware interfaces. Therefore, ESP8266 needs to be woken up via an external GPIO. The hardware wake-up process lasts for approximately 3 ms. However, since the Wi-Fi initialization process takes about 1 ms, it is recommended that users operate the chip after 5 ms.

The GPIO wake-up function can only be enabled by level triggers. The interface is as follows:

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

For example, GPIO12 can be set as the external wake-up pin via the following interface:

```
GPIO_DIS_OUTPIT(12);

PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U, FUNC_GPIO12);

wifi_enable_gpio_wakeup(12, GPIO_PIN_INTR_LOLEVEL);
```

|	
uint32 i	The IO serial number of the wake-up function.
GPIO_INT_TYPE intr_state	The wake-up trigger: • GPIO_PIN_INTR_LOLEVEL • GPIO_PIN_INTR_HILEVEL

📖 **Note:**

GPIO16 cannot be used for wake-ups.

3.4. Application

Light-sleep mode can be used in cases where certain applications need to stay connected to the router, so that they can respond to the data being sent by the router in real time. The CPU can be idle before receiving any commands. A relevant example is the Wi-Fi switch whose CPU is idle for most of the time and only performs GPIO operations until receiving the control commands.

📖 **Note:**

If a task interval is shorter than the DTIM Beacon interval, the system cannot get into Light-sleep mode, as the figure below shows.
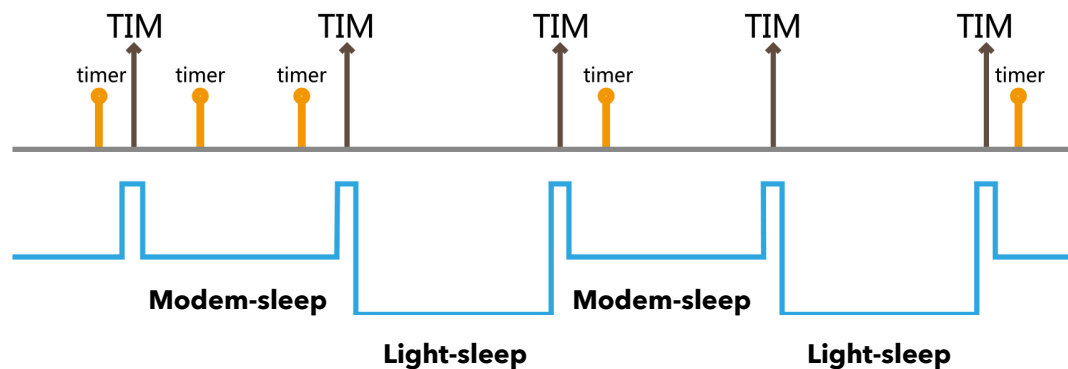
4.   Deep-sleep

4.1.   Features

Unlike the other two modes, the system cannot get into Deep-sleep automatically. Users can call the interface function `system_deep_sleep` to immediately enable Deep-sleep immediately. In this mode, the chip turns off Wi-Fi connectivity and data connection; only the RTC module continues working, and is responsible for periodic wake-ups.

> 📖 **Note:**
>
> In Deep-sleep mode, GPIO maintains its level state which has a 2-µA drive strength.

4.2.   Interface

4.2.1.   Enable Deep-sleep

The system goes into Deep-sleep mode via the following interface:

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

Parameters:

uint32 time_in_us = 0	The chip will not wake up at regular intervals, i.e., it will not wake up automatically.
uint32 time_in_us ≠ 0	The chip will be automatically woken up at regular intervals (unit: µs).

4.2.2.   Configure Deep-sleep

Users can configure the software workflow during the wake-up from Deep-sleep via the following interface, thus adjusting the average power consumption during the chip's long-running operation.

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

deep_sleep_set_option(0)	The 108th byte of the `init` parameter determines whether the RF calibration will be performed after the chip wakes up from Deep-sleep.
deep_sleep_set_option(1)	The chip will carry out the RF calibration during waking up from Deep-sleep. Power consumption is high.
deep_sleep_set_option(2)	The chip will not perform the RF calibration during waking up from Deep-sleep. Power consumption is low.
deep_sleep_set_option(4)	The chip will not turn on the RF during waking up from Deep-sleep. Power consumption is the lowest, same as in Modem-sleep.

> 📖 **Note:**
>
> The `init` parameter is the parameter value in **esp_init_data_default.bin**. For example, to change the 108th byte of the data to 8, and call `deep_sleep_set_option (0)`, means that the chip will undertake RF calibration after every eighth wake-up from Deep-sleep. For an example related to ESP8266's low-power sensor application, please refer to:
> https://github.com/EspressifSystems/low_power_voltage_measurement/wiki.

4.3. Wake-up

4.3.1. Automatic Wake-up

In Deep-sleep mode, GPIO16 (XPD_DCDC) can be connected to EXT_RSTB. When the time for sleep is up, the chip can be woken up and initialized by a low-level pulse generated on the EXT_RSTB pin via GPIO16.

4.3.2. External Wake-up

In Deep-sleep mode, the chip can be woken up and initialized by a low-level pulse generated on the EXT_RSTB pin via an external IO.

> ⚠️ **Notice:**
>
> If the automatic wake-up and the external wake-up are to be enabled at the same time, users need the right line logic operation circuit when designing the external circuit.

4.4. Application

Deep-sleep can be used in low-power sensor applications or in cases where data transmission is not required for most of the time. The device wakes up from Deep-sleep at intervals to measure and upload data, and then goes to Deep-sleep again. The device can also store data in the RTC memory (which can still save data in Deep-sleep mode) and then send it at one time.

4.5. Low-Power Solutions

Espressif provides the following eight solutions to reduce the power consumption in Deep-sleep:

1. Set the device to enter Deep-sleep mode instantly to reduce the time for it to actually enter Deep-sleep.

   Function:

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

   Sample code:

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

> 📖 **Note:**
>
> The function `system_deep_sleep_instant` is undeclared, but can be called directly.

2. Call the following function so that the chip will not perform RF calibration after waking up from Deep-sleep. This will reduce both the initialization time and current consumption.

```
system_deep_sleep_set_option(2);
```

3. Reduce RF power consumption.

   If the application does not require a high Tx peak value, users can lower the RF power consumption.

   Please make sure the ESP8266 Download Tool is V1.2, or a later version. RF power consumption can be modified in the **RF InitConfig** tab. Please replace **esp_init_data_default.bin** with the newly generated bin file **esp_init_data_setting.bin**.

4. Modify the binaries with the following command to reduce the time and current consumption during flash initialization.

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

   If OTA firmware is used, users need to modify **boot_v1.5.bin** and download the generated **boot_v15_low_power.bin** to the address `0x0`.

   If non-OTA firmware is used, users need to modify **eagle.flash.bin** and download the generated **eagle.flash_low_power.bin** to the address `0x0`.

> 📖 **Note:**
>
> Users can download **Add_Low-power_Cmd** from: https://www.espressif.com/en/support/download/other-tools?keys=ESP8266+Add+Low-power+Cmd.

5. Select the flash type and working mode.

   The right flash, e.g., ISSI-IS25LQ025, can greatly reduce the firmware loading time. The appropriate working mode of flash can also reduce the firmware loading time. Four-line working mode is recommended.

6. Clear up UART FIFO to reduce printing time.

   A FIFO (First In First Out) is a UART buffer that forces each Byte of the serial communication to be passed on in the received order. To reduce time consumption, printing out excessive information should be avoided. Therefore, all UART FIFO should be erased before the chip enters Deep-sleep mode; otherwise, the system will not get into Deep-sleep until all UART FIFO information has been printed out.

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

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

7. Synchronous data transmission.

   Data transmission consumes less time than the device's wake-up mode, and it also needs less power. It is recommended that users sending multiple data packets at a time, when ESP8266 is woken up from Deep-sleep mode.

8. **esp_iot_sdk_v1.4.0**, **esp_iot_rtos_sdk_v1.3.0** and the later versions of the SDK have largely optimized the power consumption capability. Please make sure that the SDK used is up to date.

> 📖 **Note:**
>
> Due to the minuscule amount of time required for hardware to wake up from Light-sleep mode (approximately 3 ms), for power-saving purposes, Light-sleep mode is preferred for applications that get into sleep mode for less than two seconds. If, however, an application sleeps for more than two seconds, then Deep-sleep mode is recommended.

Espressif IOT Team

www.espressif.com