ESP8266
Application Note
Firmware Download Protocol

Version 1.0

About This Guide

This document introduces ESP8266 firmware download protocol with a structure as follows.

Chapter	Title	Subject
Chapter 1	Overview	Introduction to the hardware preparations and procedure for downloading firmware.
Chapter 2	Transmission Protocol	Introduction to the data transmission format when downloading firmware into flash.
Chapter 3	Firmware Image Format	Introduction to the firmware image format in flash.
Appendix I	Programming Examples	Related programming examples.

Release Notes

Date	Version	Release Notes
2016.05	V1.0	First release.

Related Documents

Please download related documents via the following links.

Official website: *http://www.espressif.com/support/download/documents*

Official BBS: *http://bbs.espressif.com/viewtopic.php?f=67&t=225*

Document Category	Documents
HDK Guides	***ESP8266 Hardware Description***
	ESP-WROOM-02 Datasheet
SDK Guides	***ESP8266 SDK Getting Started Guide***
	ESP8266 Non-OS SDK AT Instruction Set

Table of Contents

1. Overview

1.1. Hardware Preparations

When ESP8266 is in UART download mode, you can download firmware from external MCU to ESP8266.

1.1.1. Hardware Settings

The hardware settings are shown in Table 1-1.

Table 1-1. Hardware Settings

Item	Value
UART download mode	GPIO0 and GPIO15: pulled down GPIO2: pulled high
Baud rate	Auto-bauds
Data bit	8
Stop bit	1
Parity bit	None
Flow control	Disabled

1.1.2. Hardware Connection

The hardware connection is shown in Figure 1-1.

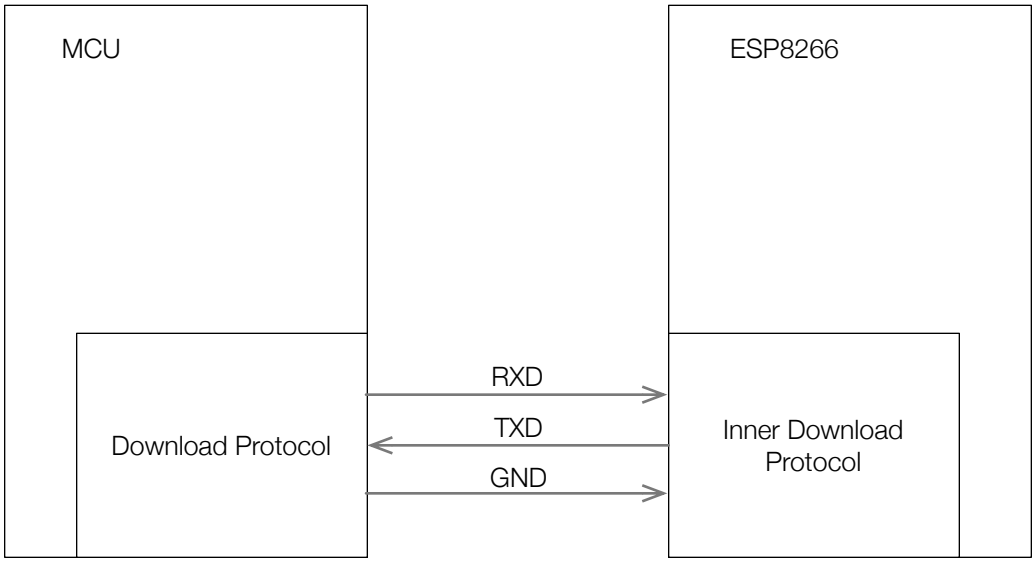

Figure 1-1. Hardware Connection

1.2. Download Procedure

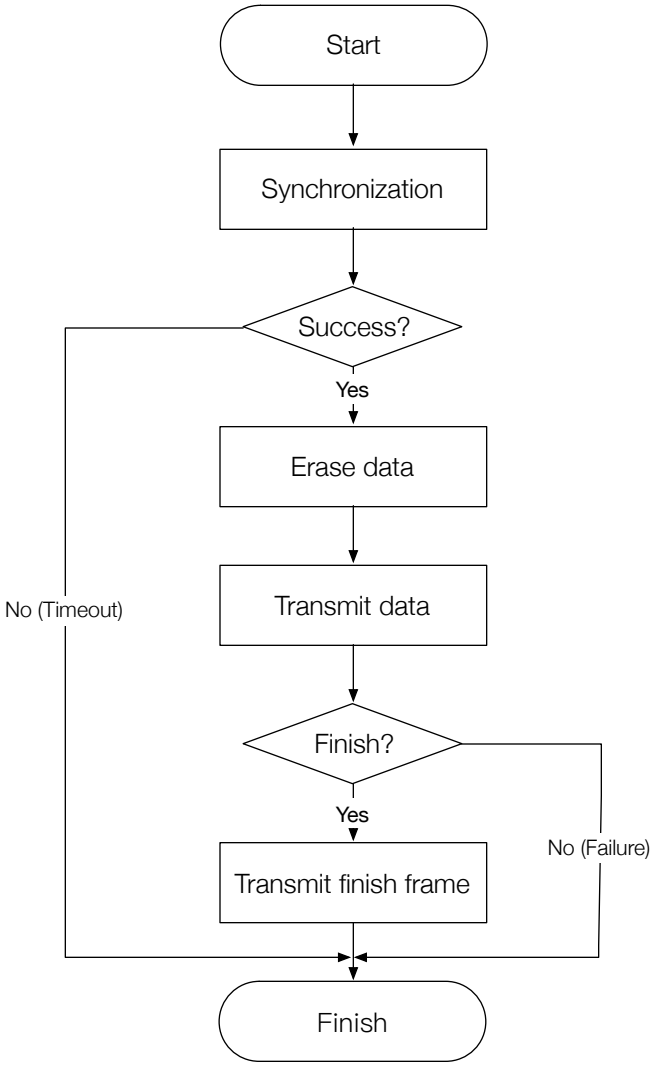

Figure 1-2. Download Procedure

- Synchronization: transmit sync frame to synchronize the baud rate.
- Erase data: erase the related flash sectors according to the size and address of the firmware to download.
- Transmit data: encapsulate the firmware into multiple frames and transmit them to ESP8266.
- Transmit finish frame: transmit download finish frame to ESP8266.

2. Transmission Protocol

The transmission protocol uses *SLIP* framing.

- Each packet begin and end with `0xC0`.
- All occurrences of `0xC0` and `0xDB` inside the packet are replaced with `0xDB 0xDC` and `0xDB 0xDD`, respectively.
- Inside the frame, the packet consists of a header and a variable length body as shown in Figure 2-1.
- All multi-byte fields are little-endian.

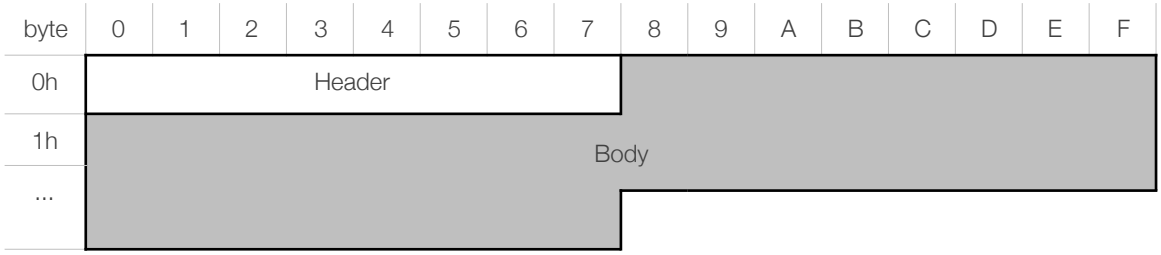

Figure 2-1. Packet Format

> 📖 ***Note:***
>
> *Data size in header is the length of packet body before being replaced.*

2.1. Packet Header

The format of packet header is shown in Table 2-1.

Table 2-1. Packet Header Format

Data type	Byte	Request	Response
Type	0	Always `0x00`.	Always `0x01`.
Command	1	Operation code. Please refer to Table 2-2 for details.	
Data size	2~3	The size of packet body.	
Checksum/ Response	4~7	XOR checksum of payload (the firmware data stored after the 16th byte of the packet body). For checksum algorithm, please refer to ***Appendix - Programming Examples***.	Response data.
Body	8~n	Depends on operation	
Status	8	-	Status flag, success (0) or failure (1).
Error	9	-	Success (null) or failure (error code).

Table 2-2. Operation Code

Codes	Name	Description
02	Flash DownLoad Start	Erase the data in the flash. • **Word0**: the number of erasing flash sectors. Each sector is 4096 bytes. • **Word1**: the number of transmitting packet. • **Word2**: packet size, e.g., `0x400`. • **Word3**: offset address. ***Note:*** *For the sample codes of erasing data, please refer to* ***Appendix - Programming Examples****.*
03	File Packet Send	Transmit data. • **Word0**: the size of writing data (filled with `0x400`). • **Word1**: the sequence number of transmitting packet. • **Word2**: `0x0` • **Word3**: `0x0`
04	Flash DownLoad Stop	Stop transmitting data.

08	Sync Frame Send	`sync_frame[36] = { 0x07, 0x07, 0x12, 0x20,`
		`0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,`
		`0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,`
		`0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55,`
		`0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55 };`

2.2. Packet Body

The packet body format is shown in Figure 2-2.

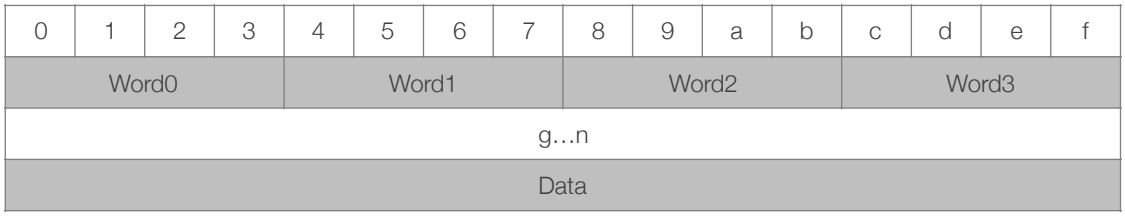

Figure 2-2. Packet Body Format

The first 16 bytes (Word0~Word3) is the description of packet body, which is different when executing different commands.

3. Firmware Image Format

The firmware consists of a file header, a variable number of data blocks (the size of blocks may be different) as shown in Figure 3-1. Multi-byte fields are little-endian.

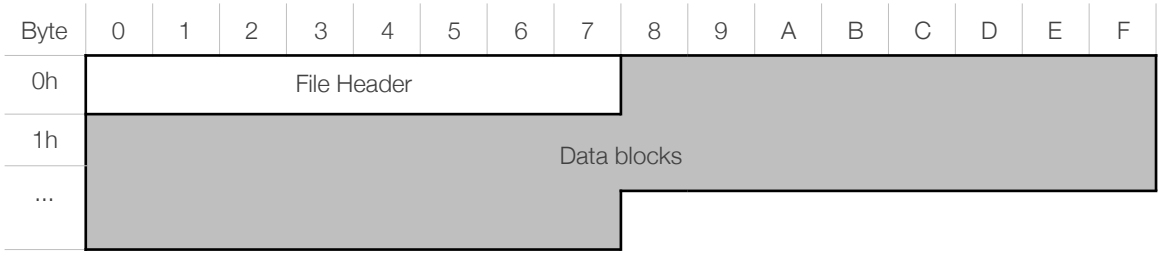

Figure 3-1. Firmware Image Format

The format of file header is shown in Table 3-1.

Table 3-1. Firmware Format Description

Byte	Data type	Description
0	Magic Code	The value is always `0XE9`.
1	Block Number	The number of blocks.
2	SPI Mode	The SPI working mode. • `0x00`: QIO mode • `0x01`: QOut mode • `0x02`: DIO mode • `0x03`: DOut mode
3	SPI Flash Info	SPI flash size and frequency. High 4 bits: `0x0` = 512 kB; `0x1` = 256 kB; `0x2` = 1 MB; `0x3` = 2 MB; `0x4` = 4 MB Low 4 bits: `0x0` = 40 MHz; `0x1` = 26 MHz; `0x2` = 20 MHz; `0xF` = 80 MHz
4~7	Entry Address	CPU entry address.

I. Appendix - Examples

I.I. checksum

```
uint32_t espcomm_calc_checksum(unsigned char *data, uint16_t
data_size)
{
    uint16_t cnt;
    uint32_t result;
    result = 0xEF;
    for(cnt = 0; cnt < data_size; cnt++)
    {
        result ^= data[cnt];
    }
    return result;
}
```

I.II. erase flash

```
#define  BLOCKSIZE_FLASH  0x400
#define  FLASH_DOWNLOAD_BEGIN  0x02
uint32 flash_packet[];
//uint32_t size:firmware real size,  uint32_t address: download
offset address
int erase_flash(uint32_t size, uint32_t address)
{
const int sector_size = 4096;
const int sectors_per_block  = 16;
const int first_sector_index = address / sector_size;
const int total_sector_count = ((size % sector_size) == 0) ?
                                  (size / sector_size) : (size /
sector_size + 1);
const int max_head_sector_count  = sectors_per_block -
(first_sector_index % sectors_per_block);
const int head_sector_count = (max_head_sector_count >
total_sector_count) ?
```

```
                                    total_sector_count :
max_head_sector_count;
// SPIEraseArea function in the esp8266 ROM has a bug which causes
extra area to be erased.
// If the address range to be erased crosses the block boundary,
// then extra head_sector_count sectors are erased.
// If the address range doesn't cross the block boundary,
// then extra total_sector_count sectors are erased.
const int adjusted_sector_count = (total_sector_count > 2 *
head_sector_count) ?
   (total_sector_count - head_sector_count):
   (total_sector_count + 1) / 2;
erase_size = adjusted_sector_count * sector_size;
flash_packet[0] = erase_size;
flash_packet[1] = (size + BLOCKSIZE_FLASH - 1) / BLOCKSIZE_FLASH;
flash_packet[2] = BLOCKSIZE_FLASH;
flash_packet[3] = address;
espcomm_send_command(FLASH_DOWNLOAD_BEGIN, (unsigned char*)
&flash_packet, 16);
}
```

I.III. References

(1) igrr/esptool-ck url: *https://github.com/igrr/esptool-ck*

(2) themadinventor/esptool url: *https://github.com/themadinventor/esptool*

Espressif IOT Team

www.espressif.com