ESP8266 Flash RW Operation

Version 1.0

Espressif Systems IOT Team

Table of Contents

1. Preambles

This document introduces flash read/write APIs, and related matters needing attention, last but not least, introduces the flash RW protection method used in IOT_Demo in ESP8266_NONOS_SDK, and some other methods as reference.

2. Flash APIs

APIs below can read/write/erase the whole flash, flash sectors start counting from 0, 4Kbytes per sector:

`spi_flash_erase_sector`	:	erase flash sector
`spi_flash_write`	:	write data to flash
`spi_flash_read`	:	read data from flash

Notice:

- Flash sector needs to be erased first, then to be written.
- Flash read/write has to be 4 bytes aligned.

Example:

```
uint32 data[M];

// TODO: fit in the data

spi_flash_erase_sector (N);
spi_flash_write (N*4*1024, data, M*4);
```

Return:

```
Typedef enum{
      SPI_FLASH_RESULT_OK,
      SPI_FLASH_RESULT_ERR,
      SPI_FLASH_RESULT_TIMEOUT
}SpiFlashOpResult;
```

2.1. spi_flash_erase_sector

Function:	erase flash sector
Prototype:	`SpiFlashOpResult spi_flash_erase_sector (uint16 sec)`
Parameter:	`uint16 sec` – sector number, start counting from sector 0, 4Kbytes per sector
Return:	`SpiFlashOpResult`

2.2. spi_flash_write

Function:	Write data into flash. Please call `spi_flash_erase_sector` to erase the target flash sector first.
Prototype:	`SpiFlashOpResult spi_flash_write (uint32 des_addr, uint32 *src_addr, uint32 size)`
Parameter:	`uint32 des_addr` – destination address on flash `uint32 *src_addr` – source address of data `Uint32 size` – data length, uint : byte.
Return:	`SpiFlashOpResult`

2.3. spi_flash_read

Function:	Read data from flash
Prototype:	`SpiFlashOpResult spi_flash_read(uint32 src_addr, uint32 * des_addr, uint32 size)`
Parameter:	`uint32 src_addr` – source address on flash `uint32 *des_addr` – destination address to keep data `Uint32 size` – data length, uint : byte.
Return:	`SpiFlashOpResult`

3. Flash layout

Please pay attention on follow flash areas:

- Program area：store bin files generated by compilation, please don't RW this area；
- System parameter area：store system parameters, please don't RW this area；
- User parameter area：store user parameters in application, users can RW this area. Users can refer to the chapter "Flash Map" in documentation "2A-ESP8266__IOT_SDK_User_Manual".

Download 2A-ESP8266__IOT_SDK_User_Manual : http://bbs.espressif.com/viewtopic.php?f=51&t=1024

firmware do not support FOTA（none boot）		
Program area	eagle.flash.bin	start from 0x00000
	eagle.irom0text.bin	start from 0x40000
System parameter area	The last 4 sectors（16 KBytes) on flash	

firmware can support FOTA（with boot）	
Program area	boot.bin starts from 0x00000
	user1.bin starts from 0x01000
	user2.bin For 512KB + 512KB Flash Map, starts from 0x81000 For 1024KB + 1024KB Flash Map, starts from 0x101000
System parameter area	The last 4 sectors（16 KBytes) on flash

Note：

1. "Program area" only mentions start address, the space it costs depends on the size of each bin.
2. System parameter area：

   esp_init_data_default.bin ：starts from the forth sector from the last on flash

   blank.bin ：starts from the second sector from the last on flash

   For example, 512KB Flash:

   esp_init_data_default.bin downloads to flash 0x7C000;

   blank.bin downloads to flash 0x7E000

4. Flash RW Protection

4.1. Foreword

Flash is erased sector by sector, which means it has to erase 4Kbytes one time at least. When you want to change some data in flash, you have to erase the whole sector, and then write it back with the new data.

So, if power off during the flash writing, data of the whole sector will be missing.

According to that, Espressif gives an example of flash RW protection and some other advices about flash RW protection as reference.

In IOT_Demo, there is an example of flash RW protection in `user_esp_platform.c`.

4.2. Basic Principle

Flash RW protection example in IOT_Demo, uses 3 sectors to provide a reliable storage of 4Kbytes.

Use sector 1 & sector 2 as data sector to save the same data of this time and the last time, this two sector alternate writing, so that there is always a sector to be backup. Use sector 3 as flag sector to keep the flag which sector (sector 1 or 2) saved the latest data.

1. Default sector is sector 2, so at first, copy data from sector 2 to RAM.
2. Then, the first time data changes, save the new data in sector 1.

- If power off unexpectedly during this phase, writing sector 1 will fail, but sector 2 & 3 are all fine; then power on, it will still copy data from sector 2 to RAM.

3. Modify sector 3 to change flag to 0， means sector 1 saved the latest data.

- If power off unexpectedly during this phase, writing sector 3 will fail, but sector 1 & 2 are all fine; then power on, because of the invalid data in sector 3, it will still copy data from sector 2 to RAM by default. Although data in sector 2 is not the latest data, but it can still work, what we lost is just the latest update.

4. The next time we want to change data in flash, we have to read flag from sector 3 first, if flag is 0, means the latest data stored in sector 1 , so we write data to sector 2 this time; if flag is not 0, assume the latest data stored in sector 2, we will write data to sector 1 this time.

- If power off unexpectedly during this phase, please look back to step 2 & 3 as reference.

5. Only after writing data sector (sector 1 or 2) accomplished, we will write flag sector (sector 3) to change the flag.

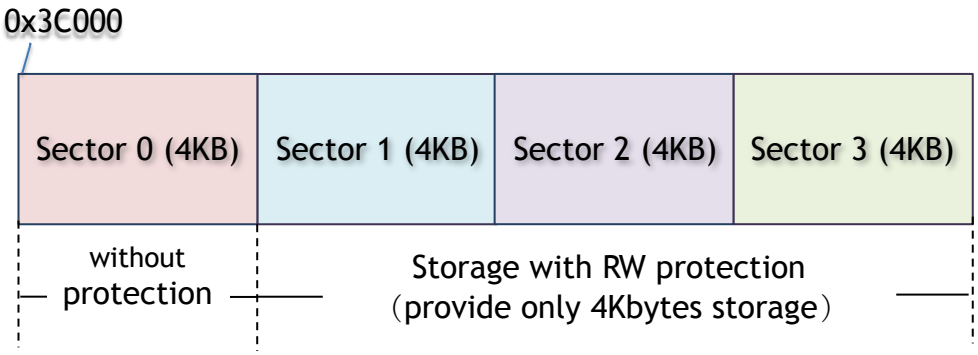
Note：

Write data sector first, then flag sector. This order ensure the integrity of data.

4.3. Example in IOT_Demo

`esp_platform_saved_param`, In IOT_Demo，use 4 sectors which starts at 0x3C000 as user data area.

In user data area, use 3 sectors(0x3D000、0x3E000、0x3F000) to provide flash RW protection, offer 4Kbytes safety storage.



Use APIs : `user_esp_platform_load_param` and `user_esp_platform_save_param` to access to the "Storage with RW protection" area in above picture.

`user_esp_platform_load_param` - read data from flash user parameter area

`user_esp_platform_save_param` - write data to flash user parameter area

`struct esp_platform_saved_param` is the parameter Espressif used that stored in flash user parameter area.

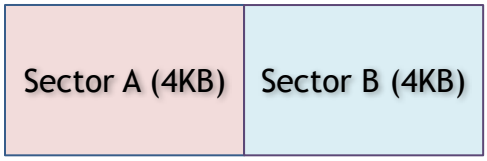
If you want to use this storage, add your parameters in the `struct esp_platform_saved_param` and call the two APIs above to read/write.

4.4. Flash RW Protection Advices

1. Advice A

Principle: "data sector switch" + "counter in head" + "check-code in the end"

It takes 2 sectors to provide storage with RW protection of 4Kbytes.



Details:

Two sectors switch to store the user data without flag sector.

Keep counters in the first byte of both data sectors. Every time writing data, count add 1 and write into the first byte of data sector. Compare the value of counter in both data sectors to distinguish

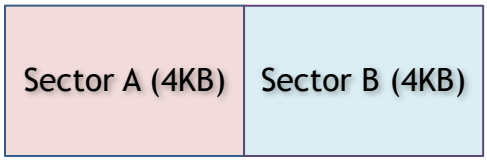
which one store the latest data. Keep check-code at the end of both data sectors, such as CRC or checksum, ensure the integrity of data. For example:

1. Data stores in sector A by default, the value of counter in sector A is 0xFF, copy data from sector A to RAM.
2. The first data changing will be written into sector B, the value of counter in sector B will be 1, check-code at the end of sector B.
3. Next time we need to save data, we will compare the value of counter between two sectors, get that the latest data was stored in sector B, so data goes to sector A this time, counter in sector A records 2, check-code at the end.
4. If power off unexpectedly, data of sector which is writing maybe lose. When power on, compare the value of counters between sector A and B, read data from the one has large count, check its integrity with the check-code, if pass, use this sector, otherwise, use the other sector, check, and load data.

2. Advice B

Principle: "sector backup" + "check-code in the end"

It takes 2 sectors to provide storage with RW protection of 4Kbytes.



Details:

Always write data to sector A, after that, write the same data to sector B as backup, keep check-codes (such as CRC、checksum) both in the end of sector A and sector B.

1. Read data from sector A，use check-code to confirm its integrity.
2. Write data to sector A, check-code at the end of it.
3. After finish writing sector A, write the same data to sector B as backup, check-code at the end of it.
4. If power off unexpectedly, data of sector which is writing maybe lose. Then power on, read data from sector A, use check-code to confirm its integrity. If pass, all goes as normal; if fail, read data from sector B, check, and program goes on.