ESP8266 Flash 读写说明

Version 0.2

Espressif Systems IOT Team

Copyright (c) 2015

免责申明和版权公告

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。

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© 2015 乐鑫信息科技（上海）有限公司所有。保留所有权利。

Table of Contents

1. 前言

本文介绍 ESP8266 的 Flash 读写接口，Flash 读写时的注意事项，以及 esp_iot_sdk 中 IOT_Demo 提供的 Flash 读写保护示例。

2. Flash 读写接口

一个扇区（sector）为 4KB，从扇区 0 开始计数，以下接口可读写整个 Flash 的任意区域：

`spi_flash_erase_sector` ： 擦除 Flash 的某个扇区（sector）。

`spi_flash_write` ： 将数据写入Flash。

`spi_flash_read` ： 读取Flash中的数据。

返回值：

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

2.1. spi_flash_erase_sector

功能：	擦除 Flash 的某个扇区。
函数定义：	`SpiFlashOpResult spi_flash_erase_sector (uint16 sec)`
参数：	`uint16 sec` - 扇区号，从扇区 0 开始计数，每扇区 4KB
返回值：	`SpiFlashOpResult`

2.2. spi_flash_write

功能：	将数据写入 Flash 。
函数定义：	`SpiFlashOpResult spi_flash_write (uint32 des_addr, uint32 *src_addr, uint32 size)`

参数：	uint32 des_addr - 写入Flash 的地址，起始位置。 uint32 *src_addr - 写入 Flash 的数据指针。 Uint32 size - 写入数据长度
返回值：	SpiFlashOpResult

2.3. spi_flash_read

功能：	从 Flash 读取数据。
函数定义：	SpiFlashOpResult spi_flash_read(uint32 src_addr, uint32 * des_addr, uint32 size)
参数：	uint32 src_addr - 写入Flash 的地址，起始位置。 uint32 *des_addr - 写入 Flash 的数据指针。 Uint32 size - 写入数据长度
返回值：	SpiFlashOpResult

3. Flash layout

由于上述软件接口，可以读写 Flash 的整个区域，请注意 Flash 以下三个区域划分：

- 程序区：代码编译生成的 bin 文件，烧录到 Flash 占用的区域，请勿改写；
- 系统参数区：esp_iot_sdk 中底层用于存放系统参数的区域，请勿改写；
- 用户参数区：上层应用程序存储用户参数的区域，在 IOT_Demo 中定义为 0x3C000 开始的 16KBytes，用户可以自行更改

软件不支持云端升级（none boot）		
程序区	eagle.flash.bin	起始于 0x00000 地址
	eagle.irom0text.bin	起始于 0x40000 地址
系统参数区	Flash 最后 4 个扇区（即 Flash 最后 16 KBytes）	

软件支持云端升级（boot）		
程序区	boot.bin	起始于 0x00000 地址
	user1.bin	起始于 0x01000 地址
	user2.bin	若为 512KB Flash ，起始于 0x41000 地址； 若为 1MB 及以上容量的 Flash，起始于 0x81000 地址
系统参数区	Flash 最后 4 个扇区（即 Flash 最后 16 KBytes）	

补充说明：

1. 上表程序区仅表明起始位置，占用空间由编译烧录的 bin 大小决定。
2. 系统参数区，存储系统级数据：

   esp_init_data_default.bin ：位于 Flash 倒数第 4 个 sector

   blank.bin ：位于 Flash 倒数第 2 个 sector

   例如，对于 512KB Flash，

   esp_init_data_default.bin 烧到 0x7C000; blank.bin 烧到 0x7E000
3. Flash 读写编程可参考 IOT_Demo 中 user_light.c，在 IoT_Demo 将应用级数据存储在了 0x3C000 开始的 4 x 4 KB （4 个扇区）区域。

   例如，master_device_key.bin (用户参数，仅在需使用 Espressif Cloud 的情况需烧录) 烧录在 0x3E000地址

4. Flash 读写保护示例

4.1. 问题描述

Flash 擦除的最小单位为一个扇区（4KB），当存储在某个扇区的数据需要改写时，流程是先擦掉整个扇区，再将该扇区的数据写回去。

那么，如果在扇区的擦写过程中突然断电，就有可能导致该扇区的数据丢失。

针对上述问题，Espressif 提供了Flash 读写保护机制的示例，并给出其他读写保护方法的建议。客户可以直接使用 Espressif 提供的 Flash 读写保护机制，也可以实现自己的 Flash 读写保护机制。

在 IOT_Demo 中 `user_esp_platform.c` 提供了 Flash 读写保护的样例。

4.2. 实现原理

Espressif Flash 读写保护示例，使用三个 sector（扇区）实现（每 sector 4KB），提供 4KB 的可靠存储空间。

将 sector 1 和 sector 2 作为数据 sector，轮流读写，始终分别存放"本次"数据和"前一次"数据，确保了至少有一份数据存储安全；sector 3 作为 flag sector，标志最新的数据存储 sector。

保护机制如下：

1. 初始上电时，数据存储在 sector 2 中，从 sector 2 中将数据读到 RAM。
2. 第一次写数据时，将数据写入 sector 1。

此时若突然掉电，sector 1写入失败，sector 2 & 3数据未改变；重新上电时，仍是从 sector 2 中读取数据，不影响使用。

3. 改写 sector 3，将标志置为 0，表示数据存于 sector 1。

此时若突然掉电，sector 3 写入失败，sector 1 & 2 均存有一份完整数据；重新上电时，因 sector 3 无有效 flag，默认从 sector 2 中读取数据，则仍能正常使用，只是未能包含掉电前对 sector 1 写入的数据。

4. 再一次写数据时，先从 sector 3 读取 flag，若 flag 为0，则上次数据存于 sector 1，此次应将数据写入 sector 2；若 flag 为非 0，则认为上次数据存于 sector 2，此次应将数据写入 sector 1。

此时若写数据出错，请参考步骤 2、3的说明，同理。

5. 写入 sector 1（或 sector 2）完成后，才会写 sector 3，重置 flag。

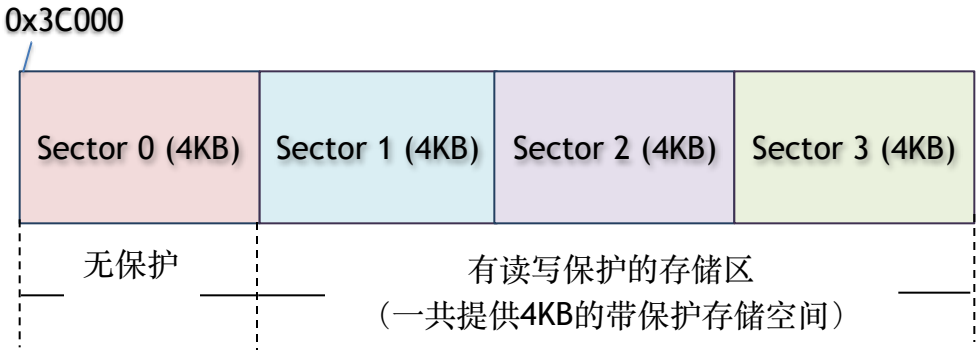
注意：

只有数据扇区（sector 1或 sector 2）写完之后，才会写 flag sector（sector 3），这样即使 flag sector 写入出错，两个数据扇区都已存有完整数据内容，目前默认会读取 sector 2。

4.3. 软件示例

在 IOT_Demo 中，使用 0x3C000 开始的 4 个 sector（每 sector 4KB），作为用户参数存储区。其中 0x3D000、0x3E000、0x3F000 这 3 个 sector 实现了读写保护的功能，并存储了应用级参数 `esp_platform_saved_param`。

如下图：



图中"有读写保护的存储区"，IOT_Demo 中建议调用 `user_esp_platform_load_param` 和 `user_esp_platform_save_param` 进行读写。

`user_esp_platform_load_param` - 读 Flash 用户参数区数据

`user_esp_platform_save_param` - 写 Flash 用户参数区数据

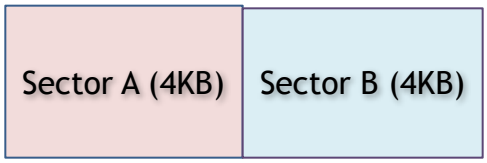
参数 `struct esp_platform_saved_param` 定义了目前乐鑫存储于 Flash 的用户应用级数据，用户只需将自己要存储的数据添加到结构体 `struct esp_platform_saved_param` 后面，调用上述两个函数进行 Flash 读写即可。

4.4. 其他建议

1. Flash 读写保护参考一

方法："轮流写入"+"首部记数"+"尾部校验"

占用空间：2 个 sector，共计 8KB；提供 4KB 的带数据保护存储空间。



原理：

仍然采用两个数据 sector 轮流写入来做备份数据保护，只是不再专门设立 flag sector。记一个 counter，写入数据 sector 的首部，每次写入时计数加一，用记数比较来判别下一次应写入哪个 sector；在数据尾部加入校验码（CRC、checksum 等任一种校验方式），用以验证数据的完整性。

(1) 假设初次上电，数据存储在 sector A，sector A 的记数为初始值 0xFF，从 sector A 将数据读入 RAM。

(2) 第一次数据写入 sector B，则在 sector B 首部信息中记录 counter 为 1，尾部加入校验码。

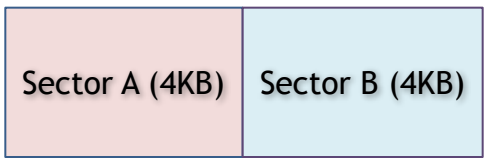
(3) 再次写入数据时，先分别读取 sector A/B 的 counter 值进行比较，此次应当将数据写入 sector A，sector A 首部记录 counter 为 2，尾部加入校验码。

(4) 若发生突然掉电，当前正在写入的 sector 数据丢失，重新上电时，先比较 sector A/B 的 counter 值，读取 counter 值较大的完整 sector，根据 sector 尾部的校验码进行校验，当前 sector 数据是否可靠，若校验通过，则继续执行；若校验失败，则读取另一个 sector 的数据，校验，并执行。

2. Flash 读写保护参考二

方法："备份扇区"+"尾部校验"

占用空间：2 个 sector，共计 8KB；提供 4KB 的带数据保护存储空间。



原理：

始终往 sector A 读写数据，每次写入时，同样写一遍 sector B 作为 sector A 的备份扇区，每个 sector 尾部均加入校验码（CRC、checksum等任一种校验方式）。

(1) 从 sector A 读取数据，并进行校验。

(2) 数据写入 sector A，尾部为校验码。

(3) sector A 写入完成后，同样的数据也写入 sector B 进行备份。

(4) 若发生突然掉电，当前正在写入的 sector 数据丢失，重新上电时，先从 sector A 读取数据，根据尾部的校验码进行校验，sector A 数据是否可靠，若校验通过，则继续执行；若校验失败，则读取 sector B 的数据，校验，并执行。