PHY62XX FS 应用指南

Phyplus Confidential

Phyplus Inc.

2019/01/23

版本控制信息

版本/状态	作者	参与者	起止日期	备注
V0.3	付晓亮		01/23/2019	文档初稿

Phyplus Confidential

目录

1 简介 5
2 API 5
2.1 枚举&宏 5
2.1.1 FS_SETTING 5
2.1.2 FS_ITEM_LEN 5
2.1.3 FS_ITEM_HEAD_LEN 5
2.1.4 FS_ITEM_DATA_LEN 6
2.1.5 FS_SECTOR_ITEM_NUM 6
2.1.6 FS_SECTOR_NUM_BUFFER_SIZE 6
2.1.7 FS_ABSOLUTE_ADDR 6
2.1.8 item_pro 6
2.1.9 item_frame 6
2.1.10 FS_FLASH_TYPE 6
2.1.11 search_type 7
2.2 数据结构 7
2.2.1 fs_cfg_t 7
2.2.2 fs_item_t 7
2.2.3 fs_t 8
2.3 API 8
2.3.1 int hal_fs_init(uint32_t fs_start_address,uint8_t sector_num) 8
2.3.2 int hal_fs_item_read(uint16_t id,uint8_t* buf,uint16_t buf_len,uint16_t* len) 8
2.3.3 int hal_fs_item_write(uint16_t id,uint8_t* buf,uint16_t len) 9
2.3.4 uint32_t hal_fs_get_free_size(void) 9
2.3.5 int hal_fs_get_garbage_size(uint32_t* garbage_file_num) 9
2.3.6 int hal_fs_item_del (uint16_t id) 10
2.3.7 int hal_fs_garbage_collect(void) 10
2.3.8 int hal_fs_format (uint32_t fs_start_address,uint8_t sector_num) 10
2.3.9 bool hal_fs_initialized(void) 11
2.4 存储示意图 11
2.5 FS 初始化流程图 12
3 示例说明 12
3.1 初始化 12
3.2 写文件 14
3.3 读文件 15
3.4 删除文件 15
3.5 垃圾回收 16
4 效率参考 17
4.1 测试参考数据 17
4.1.1 FS 容量大小得当 19
4.1.2 在 CPU 空闲时执行耗时操作 19

图表

图表 1: FS 逻辑结构示意图 11
图表 2: FS 初始化流程图 12
图表 3:首次初始化后的 FS 14
图表 4:写入文件后的 FS 物理存储 15
图表 5:删除文件后的 FS 物理存储 16

Phyplus Confidential

图表 6:FS 执行垃圾回收后的物理存储 17
图表 7FLASH API 驱动耗时测试 18
图表 8FS API 耗时测试 19

Phyplus Confidential

1 简介

本文档介绍了 PHY62XX FS 模块(以下简称 FS)的原理和使用方法，它能够帮助您理解 FS 提供的 API，您可以参考样例，快速进行实际产品开发。

FS 在物理结构上是 FLASH 上的一块连续区域，使用时需要保证 FS 存储空间和 CODE 存储空间没有交叉冲突。

FLASH 有如下特性，FS 使用时需要注意。

- 物理上可分为页、扇区、块。
- 容量上有 512KB~8MB。
- 存储空间是线性的，可以对其进行随机读写访问。
- 全新 FLASH 数据为 1，写入数据只能将 1 变 0，不能将 0 变 1。
- 擦除操作只能以块和扇区为单位进行，或者整片擦除，擦除后数据变 1。
- 有写/擦除寿命，最小 10 万次。

FS 为小型文件系统，有如下特点：

- 线性结构顺序存储，不支持文件夹。
- 每个文件的 ID 是唯一的。
- 支持文件的读、写、删除。
- 支持垃圾回收。
- 文件存储采用均衡保存。

注：使用 FS 时，FS 源文件无需修改，也不建议修改。

2 API

2.1 枚举&宏

2.1.1 FS_SETTING

FS 每条记录长度，该宏定义可以在工程的配置中设置，如果设置默认每条记录长度 16 字节。

FS_SETTING 取值范围如下：

- FS_ITEM_LEN_16BYTE：每条记录长度 16 字节
- FS_ITEM_LEN_32BYTE：每条记录长度 32 字节
- FS_ITEM_LEN_64BYTE：每条记录长度 64 字节

2.1.2 FS_ITEM_LEN

FS 每条记录存储长度。

2.1.3 FS_ITEM_HEAD_LEN

FS 每条记录文件头长度。

Phyplus Confidential

2.1.4 FS_ITEM_DATA_LEN

FS 每条记录数据区长度。

2.1.5 FS_SECTOR_ITEM_NUM

每个扇区 4096 字节包含的 FS 记录数量。

2.1.6 FS_SECTOR_NUM_BUFFER_SIZE

FS 最大的扇区数量。

2.1.7 FS_ABSOLUTE_ADDR

相对地址对应的绝对地址。

2.1.8 item_pro

FS 每条记录的存储属性。

ITEM_DEL	该文件记录被删除。
ITEM_UNUSED	该记录是全新未用。
ITEM_USED	该记录是已用有效。
ITEM_RESERVED	预留。

2.1.9 item_frame

FS 每条记录的帧属性。

ITEM_SF	单帧，文件小于等于 12 字节。
ITEM_MF_F	多帧，首帧。
ITEM_MF_C	多帧，continue 帧。
ITEM_MF_E	多帧，尾帧。

2.1.10 FS_FLASH_TYPE

FS 状态。

FLASH_UNCHECK	FS 未初始化，内容未知。
FLASH_NEW	全新的 FS，内容全部为 0xFF。

Phyplus Confidential

FLASH_ORIGINAL_ORDER	FS 有数据，数据存储为原始顺序。
FLASH_NEW_ORDER	FS 有数据，数据存储为非原始顺序。
FLASH_CONTEXT_ERROR	FS 数据无效，FS 结构和预期不一致。

2.1.11 search_type

查找类型。

SEARCH_FREE_ITEM	查找空闲位置，初始化时会查找一次。
SEARCH_APPOINTED_ITEM	查找指定的文件。
SEARCH_DELETED_ITEMS	查找删除的文件。

2.2 数据结构

2.2.1 fs_cfg_t

FS 配置信息。

类型	参数名	说明
uint32_t	sector_addr	FS 起始扇区地址，需要 4096 对齐，地址分配不能有冲突。
uint8_t	sector_num	FS 扇区数量，最小值为 3，最大值为 78。
uint8_t	item_len	文件记录长度，默认 16。
uint8_t	index	该扇区在整个 FS 中的偏移索引。
uint8_t	reserved[9]	预留。

2.2.2 fs_item_t

FS 每条记录文件头信息。

类型	参数名	说明
uint32_t :16	id	文件 id。
uint32_t :2	pro	文件记录的存储属性。
uint32_t :2	frame	文件记录的帧属性。
uint32_t :12	len	文件长度，最大长度 0xFFF。

Pinyplus confidential

2.2.3 fs_t

FS 全局控制结构体，对应变量为 fs。

类型	参数名	说明
fs_cfg_t	cfg	FS 配置信息。
uint8_t	current_sector	FS 空闲扇区索引。
uint8_t	exchange_sector	FS 交换扇区索引。
uint16_t	offset	FS 空闲扇区内空闲位置偏移。

2.3 API

2.3.1 int hal_fs_init(uint32_t fs_start_address,uint8_t sector_num)

用配置参数初始化文件系统，该函数需要在系统初始化时候设置，具体请参考例程。

- 参数

类型	参数名	说明
uint32_t	fs_start_address	FS 起始地址。 需要 4096 字节对齐，空间上不能和其他使用有冲突。
uint8_t	sector_num	FS 扇区数量，有效值 3~78。 举例： 将 FS 分配 4 个扇区，起始地址为 0x11005000 hal_fs_init(0x11005000,4)

- 返回值

PPlus_SUCCESS	初始化成功。
其他	参考<error.h>

2.3.2 int hal_fs_item_read(uint16_t id,uint8_t* buf,uint16_t buf_len,uint16_t* len)

读取 FS 文件。

- 参数

类型	参数名	说明
uint16_t	id	读取文件的 id。
uint8_t*	buf	传入 buffer 起始地址。

Phyplus confidential

buf_len	buf_len	传入 buffer 起始长度
uint16_t*	len	文件实际长度

- 返回值

PPlus_SUCCESS	初始化成功。
其他	参考<error.h>

2.3.3 int hal_fs_item_write(uint16_t id,uint8_t* buf,uint16_t len)

写入 FS 文件。

- 参数

类型	参数名	说明
uint16_t	id	写入文件的 id。
uint8_t*	buf	传入 buffer 起始地址。
uint16_t	len	传入 buffer 起始长度

- 返回值

PPlus_SUCCESS	初始化成功。
其他数值	参考<error.h>

2.3.4 uint32_t hal_fs_get_free_size(void)

FS 可用来存储文件数据的空间大小，单位为字节。

- 参数

无

- 返回值

FS 可用存储文件的空间，单位为字节。

2.3.5 int hal_fs_get_garbage_size(uint32_t* garbage_file_num)

FS 已删除文件数据区大小，单位为字节。

- 参数

类型	参数名	说明
uint32_t*	garbage_file_num	已删除文件的数量。

- 返回值

Phyplus Confidential

FS 已删除文件所占空间，单位为字节。

2.3.6 int hal_fs_item_del (uint16_t id)

删除文件。

- 参数

类型	参数名	说明
uint16_t	id	删除文件的 id。

- 返回值

PPlus_SUCCESS	成功。
其他数值	参考<error.h>

2.3.7 int hal_fs_garbage_collect(void)

垃圾回收，将 FS 中已经删除的文件所占有的空间释放。

该函数会遍历整个 FS，还会对多个扇区进行擦除操作，耗时相对较多。

建议在 CPU 空闲时且 garbage 较多时执行，执行时间和主频、FS 大小都有关系。

- 参数

无

- 返回值

PPlus_SUCCESS	初始化成功。
其他数值	参考<error.h>

2.3.8 int hal_fs_format (uint32_t fs_start_address,uint8_t sector_num)

格式化 FS，所有文件会被擦除，使用需谨慎。

如果必须调用，建议在 CPU 空闲时调用，执行时间和主频、FS 大小都有关系。

- 参数

类型	参数名	说明
uint32_t	fs_start_address	FS 起始地址。 需要 4096 字节对齐，空间上不能和其他使用有冲突。
uint8_t	sector_num	FS 扇区数量，有效值 3~78。 举例： 将 FS 分配 4 个扇区，起始地址为 0x11005000

Phyplus Confidential

		hal_fs_init(0x11005000,4)

- 返回值

PPlus_SUCCESS	成功。
其他数值	参考<error.h>

2.3.9 bool hal_fs_initialized(void)

查询 FS 初始化状态。

- 参数

无

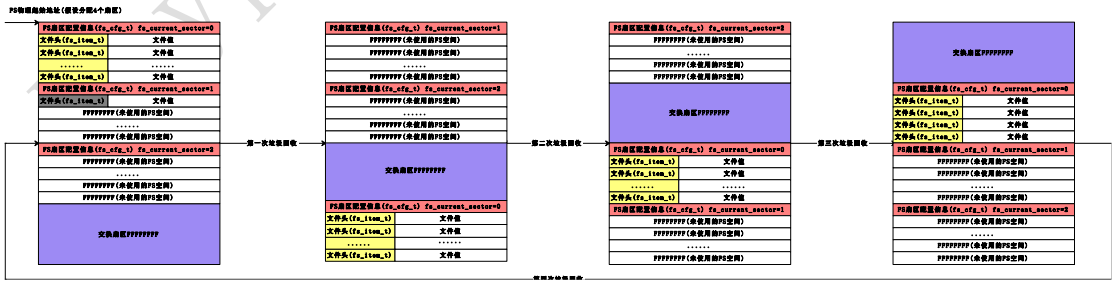
- 返回值

true	已初始化，可以使用。
false	未初始化，不能使用。

2.4 存储示意图

FS 有如下特点：

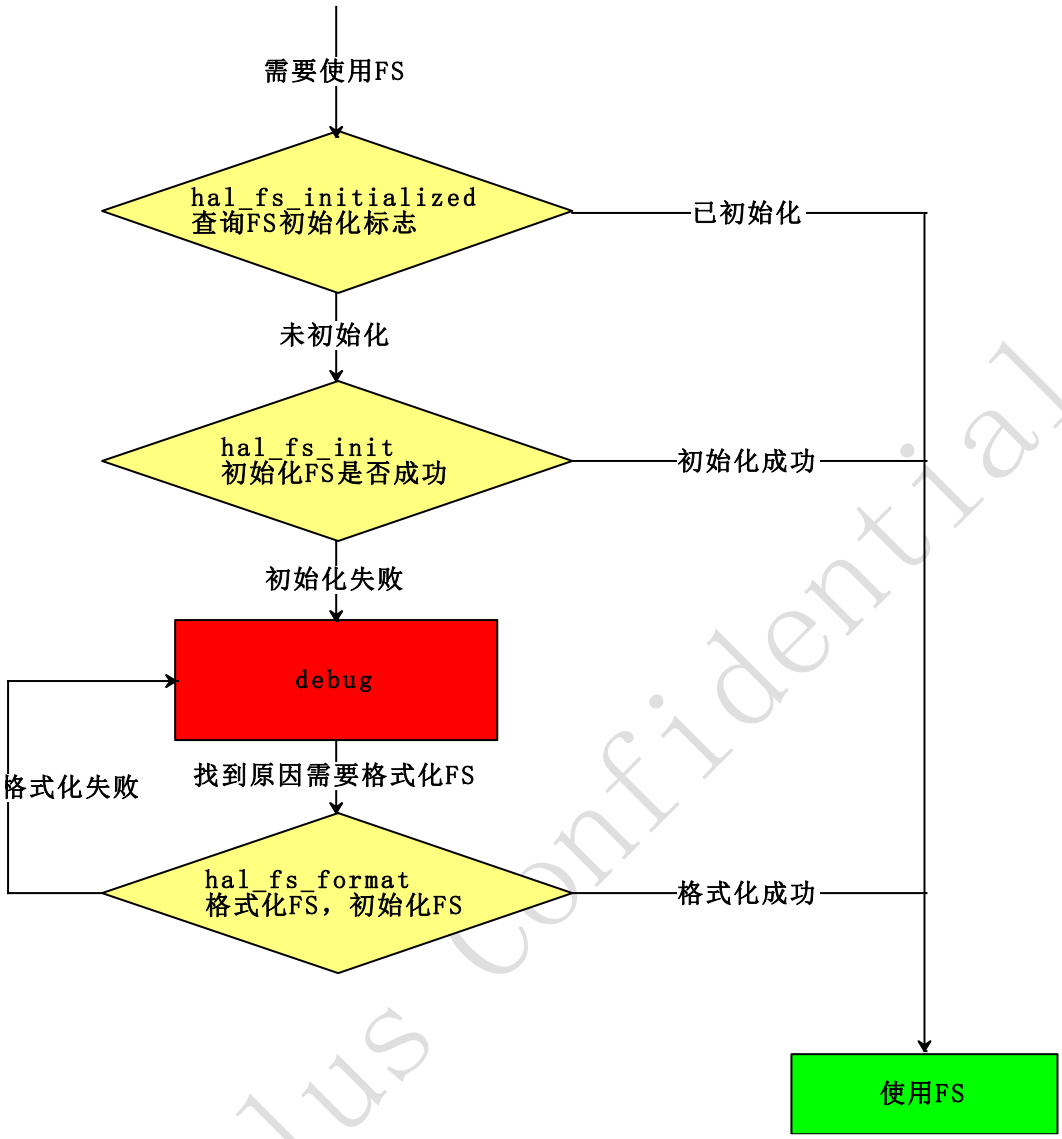
- FS 扇区分数据区和交换区，数据区存储文件数据，交换区用于垃圾回收中转区域。
- FS 使用每个数据扇区的配置信息和交换区，其余区域存储文件数据。
- 文件在每个扇区内自上而下存储，分文件头和文件区，文件头存储长度、ID、属性等信息，文件区存储文件数据。
- 可以读写文件，同一个 ID 的文件只支持一个。
- 可以删除文件，删除时只修改属性，垃圾回收时将释放其空间，参见下图第一次垃圾回收，灰色文件表示被删除文件。



图表 1: FS 逻辑结构示意图

zhuhai-jieli Confidential

2.5 FS 初始化流程图



图表 2: FS 初始化流程图

注：如果 hal_fs_init 失败，一定要排查清楚原因，特别是 FS 内容被破坏时，不能简单使用 hal_fs_format 将问题绕过。

3 示例说明

PHY62XX SDK 目录 example\peripheral 下的 fs 工程，包含了 FS 的测试框架、FS 的示例、FS 效率测试。

如果您在项目开发过程中需要用到 FS，可以参考 fs_example。

这里介绍下 FS 的示例代码。

3.1 初始化

Phyplus Confidential

上层调用：

以固定频率调用 fs_example 100 次，每个测试周期内，fs_example 包含了 FS 初始化一次、写文件两次、读文件两次、删除文件两次、垃圾回收一次。

FS 初始化：

上电后第一次使用 FS，FS 未初始化，对 FS 进行初始化。

```
if(hal_fs_initialized() == FALSE){
    ret = hal_fs_init(0x11005000,4);
    if(PPlus_SUCCESS != ret)
        LOG("error!\n");
}
```

如果 FS 是全 0xFF，则会按照配置的参数来写入 FS，类似图 2。

如果 FS 存在数据，则会进行合法性检查。

Phyplus Confidential

```
Memory 1
Address: 0x11005000
0x11005000: 11005000 FF001004 FFFFFFFF FFFFFFFF
0x11005010: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11005020: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11005030: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11005040: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11005050: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11005060: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11006000: 11005000 FF011004 FFFFFFFF FFFFFFFF
0x11006010: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11006020: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11006030: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11006040: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11006050: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11007000: 11005000 FF021004 FFFFFFFF FFFFFFFF
0x11007010: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11007020: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11007030: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11007040: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11007050: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11007060: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008000: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008010: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008020: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008030: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008040: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008050: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008060: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x11008070: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

图表 3:首次初始化后的 FS

3.2 写文件

FS 写入文件：

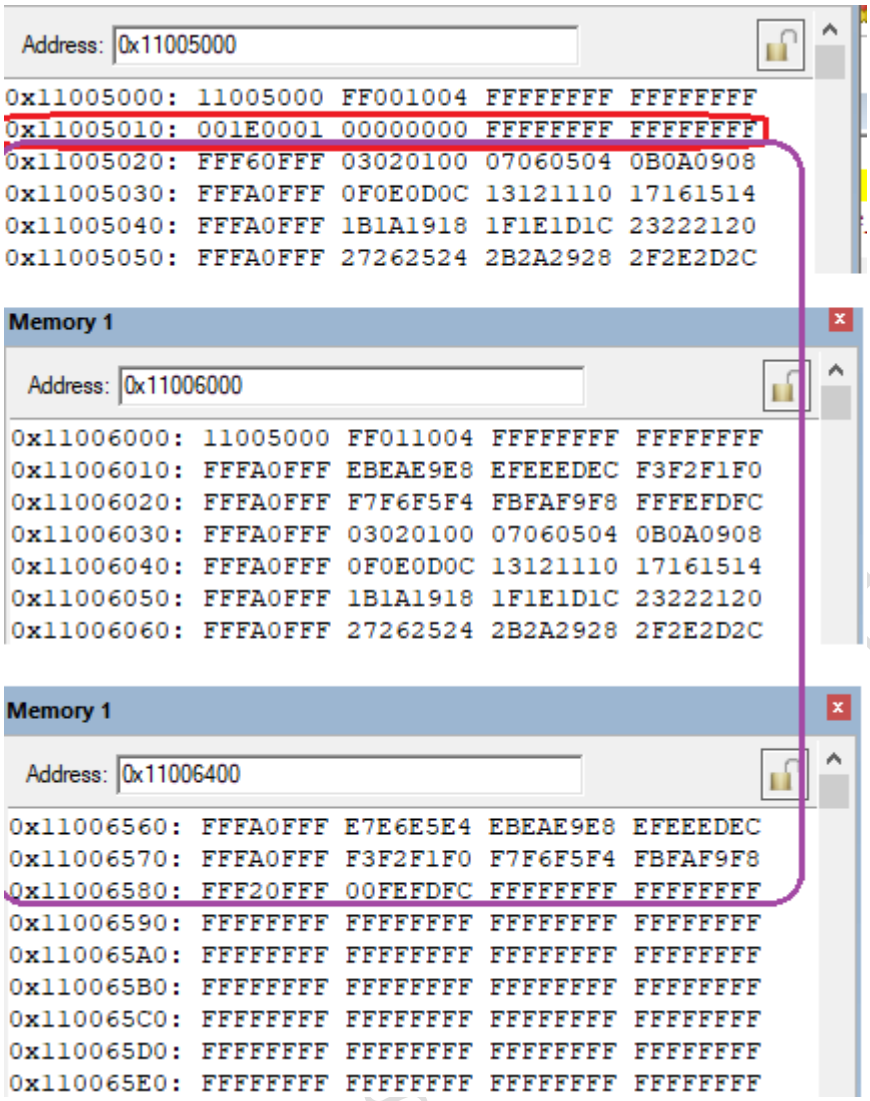
分别向 FS 写入两个文件，第一个文件 id 和长度都是 1，第二个文件 id 和长度都是 4095。

向 FS 写入文件时，需要 FS 空闲空间足够，否则会报空闲不足无法写入错误。

如果 FS 存在同样 id 的文件，则 FS 会覆盖旧的文件，旧的文件会被删除。

FS 写入文件时，会在 Flash 空闲区域依次写入数据。

Phyplus Confidential



图表 4:写入文件后的 FS 物理存储

3.3 读文件

FS 读取文件：

分别向 FS 读取两个文件，第一个文件 id 1，第二个文件 id 是 4095。

从 FS 读取文件时，需要传递 id、buffer 地址、buffer 长度给 api，api 会将读取的文件写入指定 buffer 中，并将文件长度通过参数告知上层。

3.4 删除文件

FS 删除文件：

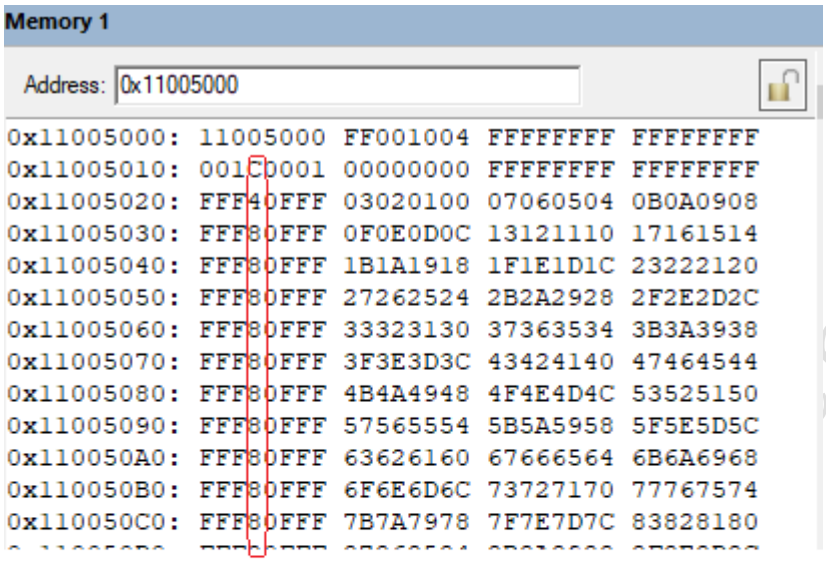
Phyplus Confidential

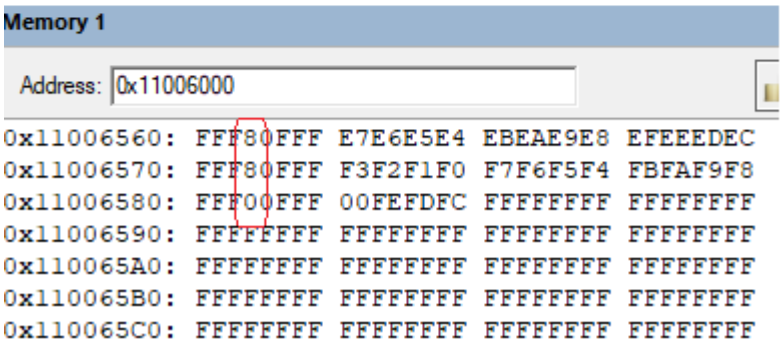
删除 FS 两个文件，第一个文件 id 1，第二个文件 id 是 4095。

文件删除后，将无法读取此 id 对应的数据。

删除后的文件存储空间将在垃圾回收后释放调。

文件删除后，文件的标识会由有效变为删除，空间在垃圾回收时释放。





图表 5:删除文件后的 FS 物理存储

3.5 垃圾回收

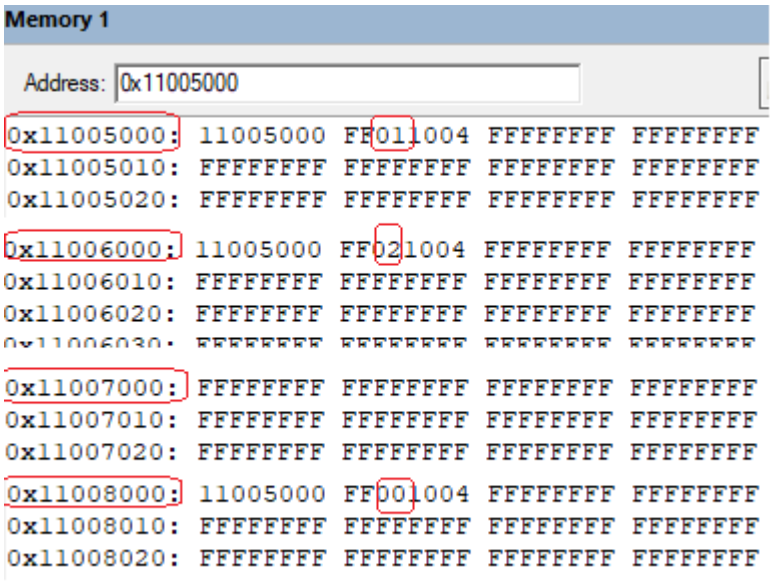
垃圾回收：

垃圾回收会对 FS 进行遍历，只保留有效的文件，被删除的文件所占空间会被释放。

在遍历过程中，扇区索引会变化，原交换区会变成新第一个数据区，原第一个数据区变成新的第二个数据区，原最后一个数据区变成新的交换区。

垃圾回收会擦除扇区，对 FS 有效文件进行搬运，因为耗时较多，建议在 CPU 空闲时操作。

执行垃圾回收后，扇区索引会变化，被删除文件会释放。



图表 6:FS 执行垃圾回收后的物理存储

4 效率参考

4.1 测试参考数据

FS 物理存储介质是 Nor Flash，FS 操作用时依赖于 Flash 底层访问速度和 FS 算法实现。

FS 访问速度和系统主频、FS 扇区数量、ITEM 长度等配置有关，下面数据测试场景是主频 48Mhz、扇区数量为 3、ITEM 长度为 16，仅供参考。

操作	测试代码	

擦除 1 个扇区	hal_gpio_write(TOGGLE_GPIO,0);
	flash_sector_erase(0x11005000);
	hal_gpio_write(TOGGLE_GPIO,1);
		112ms

写 4 个字节	hal_gpio_write(TOGGLE_GPIO,0);
	WriteFlash(0x11005000,0x12345678);
	hal_gpio_write(TOGGLE_GPIO,1);
		61us

读 4 个字节	hal_gpio_write(TOGGLE_GPIO,0); osal_memcpy((uint8_t*)read_buf,0x11005000,4);	26us

Phyplus Confidential

	hal_gpio_write(TOGGLE_GPIO,1);	

图表 7Flash API 驱动耗时测试

操作	测试代码	

格式化 FS	hal_gpio_write(TOGGLE_GPIO,0);
	
	ret = hal_fs_format(0x11005000,3);
	
	hal_gpio_write(TOGGLE_GPIO,1);
		308ms

	格式化 FS 包含了 FS 扇区每个扇区的擦除和 FS 初始化，主要却决于前者，和 FS 扇区数量有关。	

FS 初始化	hal_gpio_write(TOGGLE_GPIO,0);
	
	ret = hal_fs_init(0x11005000,3);
	
	hal_gpio_write(TOGGLE_GPIO,1);
		170us(FS 为空)
		
		248us (包含 10 个单帧文件)

	FS 初始化包含下面两大功能： ➢ FS 各扇区的头信息，与设定值相比较，看是否合法，无擦除操作。 ➢ 如果 FS 非空，查找并记录 FS 的空闲位置。	
写文件	FS 全新的，第一次写，文件长度为 1 字节	160us
	FS 前面有 1 个文件，长度为 1，所写文件长度为 100 字节	2.00ms
	FS 前面有 2 个文件，长度分别为 1 和 100，所写文件长度为 100 字节	2.04ms
	文件写操作耗时和存储文件数量、所写文件长度都有关系。	
读文件	只存在一个文件，第一次就读到，文件长度为 1 字节	46us
	FS 前面有 1 个文件，长度为 1，所读文件长度为 100 字节	276us
	FS 前面有 2 个文件，长度分别为 1 和 100，所读文件长度为 100 字节	306us
	文件读操作耗时和存储文件数量、所写文件长度都有关系。	
删除文件	删除一个文件，第一次就找到位置，文件长度为 1 字节	116us

Phyplus Confidential

	删除一个文件，其前面有一文件长度为 1，所删除文件长度为 1 字节	588us
	删除一个文件，其前面有两文件长度为 1 和 100，所删除文件长度为 100 字节	604us
	文件的删除操作耗时和本文件的长度、文件在 FS 中位置有关。	

垃圾容量统计	对上述测试项三个删除的文件进行垃圾容量统计
	
	hal_gpio_write(TOGGLE_GPIO,0);
	
	garbage_size = hal_fs_get_garbage_size(&garbage_num);
	
	hal_gpio_write(TOGGLE_GPIO,1);
		128us

	垃圾统计时间取决于文件系统中使用文件的大小，当垃圾文件和有效文件数量以及比例。	
垃圾回收	对上述测试项三个删除的文件进行垃圾回收	226ms
	垃圾回收会遍历整个 FS 的有效目录项和删除目录项，并依次擦除扇区并搬运数据。 垃圾回收的时间取决于文件系统中使用文件的大小，当垃圾文件和有效文件数量以及比例。	

图表 8FS API 耗时测试

4.1.1 FS 容量大小得当

- 太小的 FS，会遇到没有足够空间可写的场景。
- 太大的 FS，则会浪费存储空间。
- 建议根据自身项目需求配置 FS 的大小。

4.1.2 在 CPU 空闲时执行耗时操作

- hal_fs_format 和 hal_fs_garbage_collect 会对扇区进行擦除，耗时相对较多。
- 建议在 CPU 空闲时执行 hal_fs_format 和 hal_fs_garbage_collect。
- hal_fs_format 会擦除 FS 所有数据，再初始化 FS，使用需谨慎。
- hal_fs_garbage_collect 会释放已删除文件占有的存储空间，当垃圾空间足够大且有效空间足够小时，可能会因为空间不够影响新文件写入，可以在 CPU 空闲时执行此操作。

Phytium Confidential