PHY62XX

FS Application Note

Version 0.3

Author: Fu Xiaoliang

Security: Public

Date: 2021.3

PhyPlus

Revision History

Revision	Author	Participant	Date	Description
v0.3	Fu Xiaoliang		01/23/2019	Draft file

Table of Contents

Figures and Tables

1 Introduction

This file introduces the principle and usage of the PHY62XX FS module (hereinafter referred to as FS). It can help you understand the API provided by FS. You can refer to the sample to quickly develop the actual product.

FS is a continuous area on FLASH in physical structure. When using, it is necessary to ensure that there is no cross conflict between FS storage space and CODE storage space.

FLASH has the following characteristics, you need to pay attention when using FS.

- Physically can be divided into pages, sectors and blocks.
- Capacity is 512KB~8MB.
- Storage space is linear, and it can be accessed randomly.
- New FLASH data is 1, the write data can only change 1 to 0, not 0 to 1.
- Erasing operation can only be performed in units of blocks and sectors, or entire chip erasing, after erasing, the data becomes 1.
- Has write/erase life, minimum 100,000 times.

FS is a small file system with the following characteristics:

- Sequential storage in linear structure, folders are not supported.
- The ID of each file is unique.
- Support reading, writing and deleting of files.
- Garbage collection is no longer support for Mac apps.
- File storage adopts balanced storage.

Note: When using FS, the FS source file does not need to be modified, nor is it recommended to modify.

2 API

2.1 Enumeration & Macro

2.1.1 FS_SETTING

Length of each record of FS, this macro definition can be set in the configuration of the project, if set the default length of each record is 16 bytes.
Value range of FS_SETTING is as follows:

- FS_ITEM_LEN_16BYTE: Length of each record is 16 bytes.
- FS_ITEM_LEN_32BYTE: Length of each record is 32 bytes.
- FS_ITEM_LEN_64BYTE: Length of each record is 64 bytes.

2.1.2 FS_ITEM_LEN

FS storage length of each record.

2.1.3 FS_ITEM_HEAD_LEN

FS Length of each record file header.

2.1.4 FS_ITEM_DATA_LEN

Length of each FS record data area.

2.1.5 FS_SECTOR_ITEM_NUM

Number of FS records contained in 4096 bytes per sector.

2.1.6 FS_SECTOR_NUM_BUFFER_SIZE

FS maximum number of sectors.

2.1.7 FS_ABSOLUTE_ADDR

Absolute address corresponding to relative address.

2.1.8 item_pro

FS storage attributes of each record.

ITEM_DEL	File record is deleted.
ITEM_UNUSED	Record is brand new and unused.
ITEM_USED	Record is valid.
ITEM_RESERVED	Reserved.

2.1.9 item_frame

FS Frame attributes of each record.

ITEM_SF	Single frame, the file is less than or equal to 12 bytes.
ITEM_MF_F	Multi-frame, first frame.
ITEM_MF_C	Multi-frame, continue frame.
ITEM_MF_E	Multi-frame, tail frame.

2.1.10 FS_FLASH_TYPE

FS status.

FLASH_UNCHECK	FS is not initialized, the content is unknown.
FLASH_NEW	Brand new FS, all content is 0xFF.
FLASH_ORIGINAL_ORDER	FS has data, and the data is stored in the original order.
FLASH_NEW_ORDER	FS has data, and data is stored in non-original order.
FLASH_CONTEXT_ERROR	The FS data is invalid, and the FS structure is inconsistent with expectations.

2.1.11 search_type

Find the type.

SEARCH_FREE_ITEM	Find a free location, it will be searched once during initialization.
SEARCH_APPOINTED_ITEM	Find the specified file.
SEARCH_DELETED_ITEMS	Find deleted files.

2.2 Data structure

2.2.1 fs_cfg_t

FS configuration information.

Type	Parameter name	Description
uint32_t	sector_addr	FS starting sector address, need 4096 alignment, address allocation can not conflict.
uint8_t	sector_num	Number of FS sectors, the minimum is 3 and the maximum is 78.
uint8_t	item_len	File record length, the default is 16.
uint8_t	index	Offset index of the sector in the entire FS.
uint8_t	reserved[9]	Reserved.

2.2.2 fs_item_t

FS each record file header information.

Type	Parameter name	Description
uint32_t :16	id	File id.
uint32_t :2	pro	Storage properties of file records.
uint32_t :2	frame	Frame attributes of the file record.
uint32_t :12	len	File length, maximum length 0xFFF.

2.2.3 fs_t

FS global control structure, the corresponding variable is fs.

Type	Parameter name	Description
fs_cfg_t	cfg	FS configuration information.
uint8_t	current_sector	FS free sector index.
uint8_t	exchange_sector	FS swap sector index.
uint16_t	offset	Free position offset in the FS free sector.

2.3 API

2.3.1 int hal_fs_init(uint32_t fs_start_address,uint8_t sector_num)

Initialize the file system with configuration parameters. This function needs to be set during system initialization. For details, please refer to the routine.

- Parameter

Type	Parameter name	Description
uint32_t	fs_start_address	FS start address. 4096-byte alignment is required, and space cannot conflict with other uses.
uint8_t	sector_num	Number of FS sectors, the effective value is 3~78. Example: Assign FS 4 sectors, starting address is 0x11005000 hal_fs_init(0x11005000,4)

- Return value

PPlus_SUCCESS	Initialization succeeded.
Others	Reference<error.h>

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

Read FS file.

- Parameter

Type	Parameter name	Description
uint16_t	id	Read the id of the file.
uint8_t*	buf	Incoming buffer start address.
buf_len	buf_len	Incoming buffer start length
uint16_t*	len	Actual file length

- Return value

PPlus_SUCCESS	Initialization succeeded.
Others	Reference<error.h>

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

Write FS file.

- Parameter

Type	Parameter name	Description
uint16_t	id	Write file id.
uint8_t*	buf	Incoming buffer start address.
uint16_t	len	Incoming buffer start length

- Return value

PPlus_SUCCESS	Initialization succeeded.
Other values	Reference<error.h>

2.3.4 uint32_t hal_fs_get_free_size(void)

Size of the space that FS can use to store file data, in bytes.

- Parameter

  None.

- Return value

  FS can store the file space, the unit is byte.

2.3.5 int hal_fs_get_garbage_size(uint32_t* garbage_file_num)

Size of the data area of the FS deleted file, in bytes.

- Parameter

Type	Parameter name	Description
uint32_t*	garbage_file_num	Number of files deleted.

- Return value

  FS Space occupied by deleted files, in bytes.

2.3.6 int hal_fs_item_del (uint16_t id)

Delete a file.

- Parameter

Type	Parameter name	Description
uint16_t	id	Delete file id.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

2.3.7 int hal_fs_garbage_collect(void)

Garbage collection releases the space occupied by deleted files in FS.
This function will traverse the entire FS and erase multiple sectors, which takes a relatively long time.
It is recommended to execute when the CPU is idle and there are more garbage. Execution time is related to the main frequency and the size of the FS.

- Parameter
  None.

- Return value

PPlus_SUCCESS	Initialization succeeded.
Other values	Reference<error.h>

2.3.8 int hal_fs_format (uint32_t fs_start_address,uint8_t sector_num)

Format FS, all files will be erased, use caution.
If it must be called, it is recommended to call it when the CPU is idle. Execution time is related to the main frequency and FS size.

- Parameter

Type	Parameter name	Description
uint32_t	fs_start_address	FS start address. 4096-byte alignment is required, and space cannot conflict with other uses.
uint8_t	sector_num	Number of FS sectors, the effective value is 3~78. Example: Assign FS 4 sectors, starting address is 0x11005000 hal_fs_init(0x11005000,4)

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

2.3.9 bool hal_fs_initialized(void)

Query the initialization status of FS.

- Parameter

  None.

- Return value

0	It has been initialized and can be used.
false	It is not initialized and cannot be used.

2.4 Storage diagram

FS has the following characteristics:

- FS sector is divided into data area and exchange area, data area stores file data, and exchange area is used for garbage collection transit area.
- FS uses the configuration information and exchange area of each data sector, and the rest of the area stores file data.
- Files are stored from top to bottom in each sector, divided into file header and file area. File header stores information such as length, ID, attributes, and the file area stores file data.
- Can read and write files, only one file with the same ID is supported.
- Files can be deleted, only the attributes are modified when deleting, and its space will be released during garbage collection. Refer to the figure below for the first garbage collection. Gray files indicate deleted files.

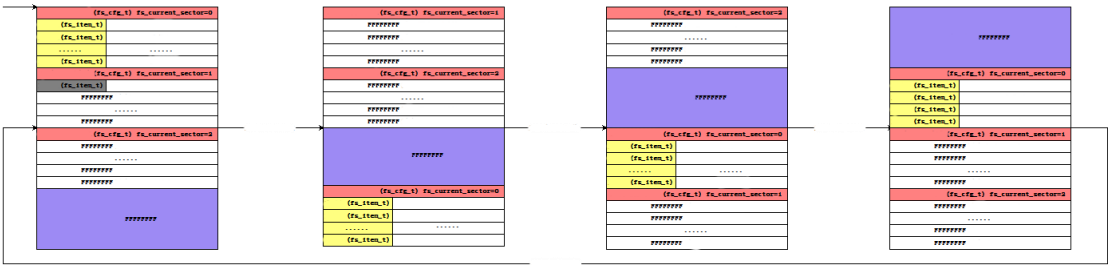

Figure 1: Schematic Diagram of FS Logical Structure

2.5 FS initialization flow chart

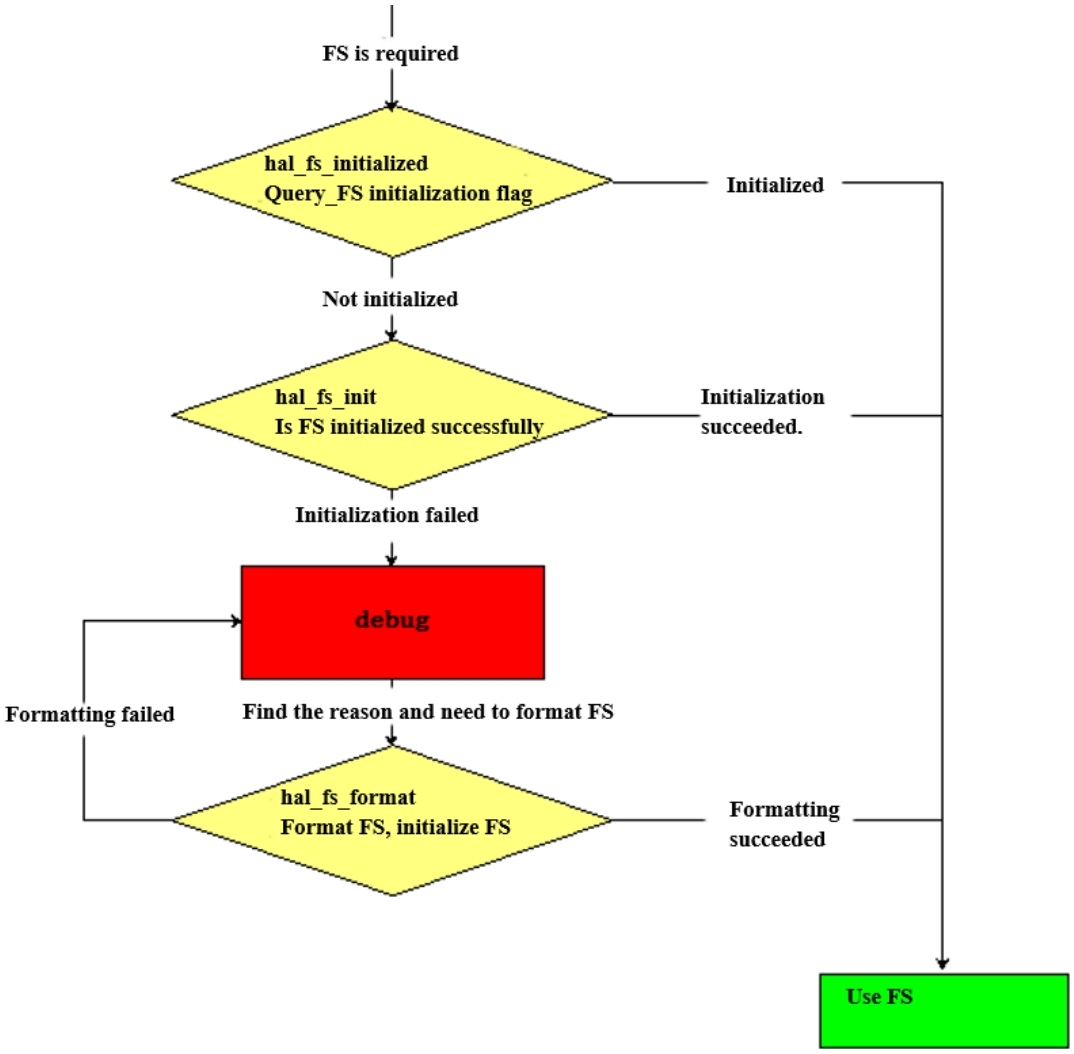

Figure 2: FS Initialization Flow Chart

Note: If hal_fs_init fails, you must investigate the cause, especially when the FS content is damaged, you cannot simply use hal_fs_format to bypass the problem.

3 Example description

Fs item under the PHY62XX SDK directory example\peripheral contains the FS test framework, FS examples, and FS efficiency tests.

If you need to use FS during item development, you can refer to fs_example.

Here is the sample code of FS.

3.1 Initialization

Upper call:

Call fs_example 100 times at a fixed frequency. In each test cycle, fs_example includes FS initialization once, file writing twice, file reading twice, file deletion twice, and garbage collection once.

FS initialization:

The FS is used for the first time after power-on, the FS is not initialized, and the FS is initialized.

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

If FS is all 0xFF, it will be written to FS according to the configured parameters, similar to **Figure 2**.

If the FS has data, it will conduct a legality check.

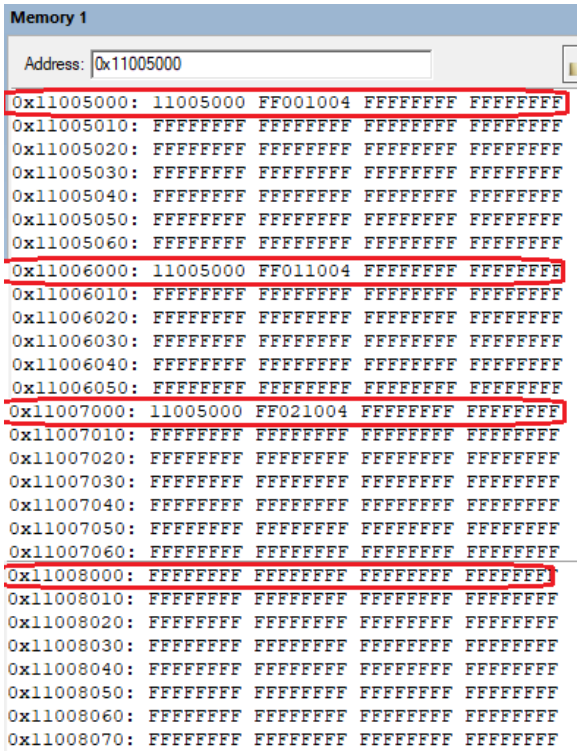

Figure 3: FS after Initial Initialization

3.2 Write files

FS writes file: Write two files to FS respectively, the first file id and length are both 1, and the second file id and length are both 4095. When writing a file to FS, you need enough FS free space, otherwise it will report insufficient free space and cannot write error. If a file with the same id exists in the FS, the FS will overwrite the old file and the old file will be deleted.

When FS writes a file, it will write data sequentially in the free area of Flash.

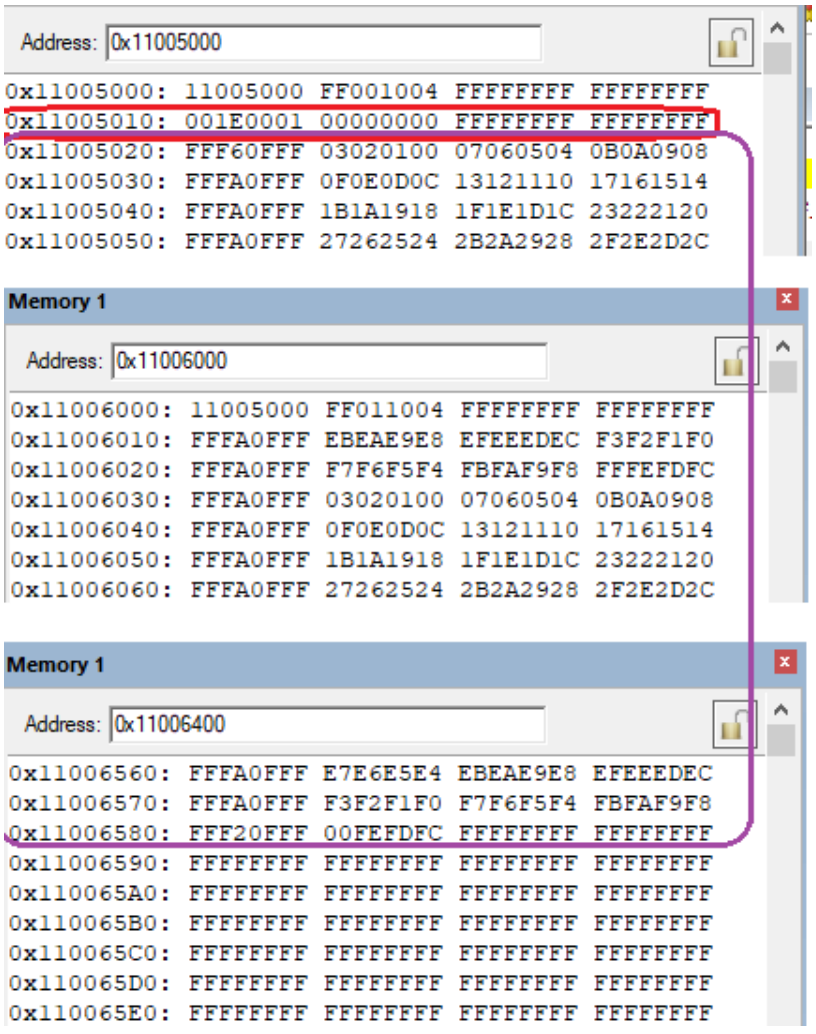

Figure 4: FS Physical Storage after File Writing

3.3 Read files

FS reads file: Read two files to FS respectively, the first file id is 1, and the second file id is 4095. When reading a file from FS, you need to pass the id, buffer address, and buffer length to the api. Api will write the read file into the specified buffer and inform the upper layer of the file length through parameters.

3.4 Delete files

FS deleted files: Delete two FS files, the first file id is 1, and the second file id is 4095. After the file is deleted, the data corresponding to this id cannot be read. Deleted file storage space will be released after garbage collection.

After the file is deleted, the identification of the file will change from valid to deleted, and the space will be released during garbage collection.

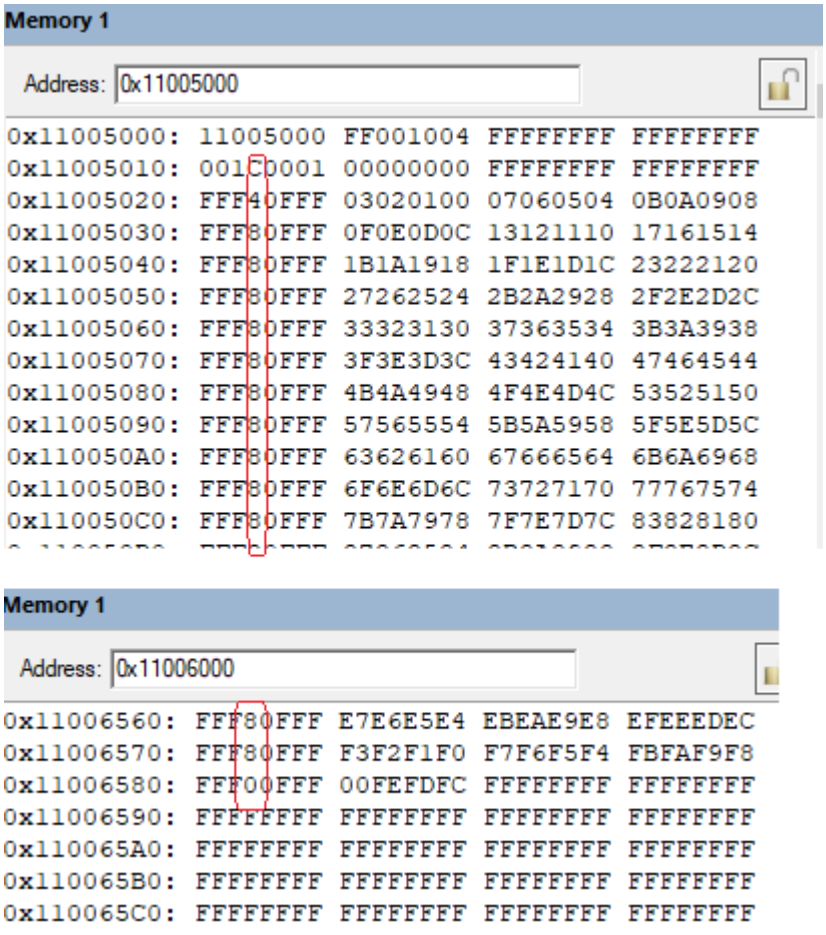

Figure 5: FS Physical Storage after File Deleting

3.5 Garbage collection

Garbage collection: Garbage collection will traverse the FS and only keep valid files, and the space occupied by deleted files will be released. During the traversal process, the sector index will change, the original exchange area will become the new first data area, the original first data area will become the new second data area, and the original last data area will become the new exchange Area. Garbage collection will erase sectors and move FS valid files. Because it takes more time, it is recommended to operate when the CPU is idle.

After garbage collection is performed, the sector index will change and the deleted files will be released.

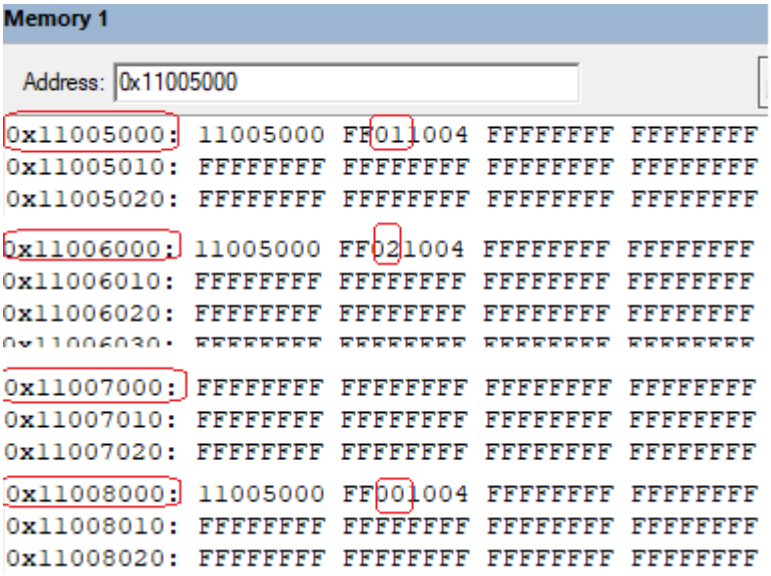

Figure 6: Physical Storage after FS Performs Garbage Collection

4 Efficiency reference

4.1 Test reference data

The FS physical storage medium is Nor Flash, and the FS operation time depends on the access speed of the underlying Flash and the implementation of the FS algorithm.
The FS access speed is related to the configuration of the system's main frequency, the number of FS sectors, and the length of ITEM. The following data test scenario is that the main frequency is 48Mhz, the number of sectors is 3, and the ITEM length is 16, for reference only.

Operation	Test code	Ref. Access Speed

Erase 1 sector	hal_gpio_write(TOGGLE_GPIO,0);
	flash_sector_erase(0x11005000);
	hal_gpio_write(TOGGLE_GPIO,1);
		112ms

Write 4 bytes	hal_gpio_write(TOGGLE_GPIO,0);
	WriteFlash(0x11005000,0x12345678);
	hal_gpio_write(TOGGLE_GPIO,1);
		61us

Read 4 bytes	hal_gpio_write(TOGGLE_GPIO,0);
	osal_memcpy((uint8_t*)read_buf,0x11005000,4);
	hal_gpio_write(TOGGLE_GPIO,1);
		26us

Table 1: Flash API Driver Time-consuming Test

Operation	Test code	Ref. Speed	Description

Format FS	hal_gpio_write(TOGGLE_GPIO,0);
	ret = hal_fs_format(0x11005000,3);
	hal_gpio_write(TOGGLE_GPIO,1);
		308ms	Formatting FS includes erasing and FS initialization of each sector of the FS sector, but mainly depends on the former and is related to the number of FS sectors.

FS initialization	hal_gpio_write(TOGGLE_GPIO,0);
	ret = hal_fs_init(0x11005000,3);
	hal_gpio_write(TOGGLE_GPIO,1);
		170us
		(FS is empty)
		248us
		(including 10 single frame files)
			FS initialization includes the following two functions: Header information of each sector of the FS is compared with the set value to see if it is legal and there is no erase operation. If FS is not empty, find and record the free position of FS.

Write file	FS is brand new, written for the first time, the file length is 1 byte	160us	File write operation time is related to the number of stored files and the length of the written file.
	There is 1 file in front of FS, the length is 1, the length of the written file is 100 bytes	2.00ms	
	There are 2 files in front of FS, the length is 1 and 100 respectively, the length of the written file is 100 bytes	2.04ms	

Read file	There is only one file, read it for the first time, the file length is 1 byte	46us	File read operation time is related to the number of stored files and the length of the written file.
	There is 1 file in front of FS, the length is 1, the length of the read file is 100 bytes	276us	
	There are 2 files in front of FS, the length is 1 and 100 respectively, the length of the read file is 100 bytes	306us	
Delete file	Delete a file, find the location the first time, the file length is 1 byte	116us	Time-consuming operation of file deletion is related to the length of the file and the location of the file in the FS.
	Delete a file, the length of the file in front of it is 1, and the length of the deleted file is 1 byte	588us	
	Delete a file, the length of the two files before it is 1 and 100, the length of the deleted file is 100 bytes	604us	

Garbage capacity statistics	Perform garbage capacity statistics on the three deleted files of the above test items
	hal_gpio_write(TOGGLE_GPIO,0);
	garbage_size = hal_fs_get_garbage_size(&garbage_num);
	hal_gpio_write(TOGGLE_GPIO,1);
		128us	Garbage statistics time depends on the size of the files used in the file system, the number and proportion of garbage files and valid files.

Garbage collection	Garbage collection of the three deleted files of the above test items	226ms	Garbage collection will traverse the effective directory entries and delete directory entries of the entire FS, erase sectors and move data in turn. Garbage collection time depends on the size of the files used in the file system, the number and ratio of garbage files and valid files.

Table 2: FS API Time-consuming Test

4.1.1 FS capacity is appropriate

- If the FS is too small, there will be scenarios where there is not enough space to write.
- Too big FS will waste storage space.
- It is recommended to configure the size of FS according to your own item requirements.

4.1.2 Perform time-consuming operations when the CPU is idle

- hal_fs_format and hal_fs_garbage_collect will erase sectors, which takes a relatively long time.
- It is recommended to execute hal_fs_format and hal_fs_garbage_collect when the CPU is idle.
- hal_fs_format will erase all data of FS, and then initialize FS, use it with caution.
- hal_fs_garbage_collect will release the storage space occupied by the deleted files. When the garbage space is large enough and the available space is small enough, the writing of new files may be affected due to insufficient space. This operation can be performed when the CPU is idle.