Pi file system (πfs)

Peter Ivanov <ivanovp@gmail.com>

December 27, 2017

Contents

1 Introduction

This file system was developed for embedded systems which use NOR flash as storage. It was designed mainly for ARM Cortex processors, but it can be ported to any 16/32-bit processor. The development started as teach-myself project and I released the code in hope that it will be useful for someone else as well.

NOR flash ICs have very low price nowadays (2017) and can be used to store files. But there are few problems to consider when designing a file system as they are not working like magnetic devices (hard disk drives or floppy disks):

- NOR flashes can be programmed (set bits to zero) by pages, but only can be erased (set bits to one) in a larger quantity, which are mostly called block in the data sheets. One page is usually 256 or 512 bytes, one block consists of 16, 256, 1024, etc. pages. So typical block sizes are 4 KiB, 64 KiB, 256 KiB.
- Blocks can be erased 10,000–100,000 times. After that data retention is not guaranteed. Therefore all blocks should be erased uniformly. This method is called wear leveling.

πfs can be scaled from 4 Mbit (512 KiB), 256 Mbit (32 MiB), theoretically up to 1 Gbit (128 MiB) memory sizes.

The code is not fully MISRA compatible, but I've written MISRA code earlier and I tend to use the rules. So I tried to use '`return`' only at the end of functions, use '`break`' once in '`while`' cycles or not use it at all and call one function from expressions (or not call functions at all from expressions). I didn't use '`goto`' in the sources of file system, but you may find them in STM32 HAL's sources.

1.1 Features of the πfs

- Small memory footprint (3-4 KiB static, 1-3 KiB stack/task)
- Files can be opened for update ("r+", "w+", "a+" modes are supported)
- Compatible with standard C functions: `fopen()`, `fwrite()`, `fread()`, `fclose()`
- Size of logical page is user-defined
- Cache buffer for page (currently only one page is cached)

- Directory handling
- Dynamic wear-leveling
- Static wear-leveling
- User data can be added for files: permissions, owner IDs, etc.
- At the beginning of flash memory reserved blocks can be defined, which are not used by the file system
- File names are case-sensitive

1.2 Limitations of the πfs

- Only one flash chip can be used (one volume is supported)
- Memory and file system configuration cannot be changed during run-time
- One directory can only store pre-defined number of files or directories
- Partial OS support: file system can be used from multiple tasks, but there is only one working directory for all tasks
- Incompatible with FAT file system, therefore cannot be used for USB mass storage devices

2 Definitions, Acronyms and Abbreviations

NOR flash	Special type of EEPROMs, which manufactured using NOR gates.
Page	Array of several bytes in the flash memory. Number of bytes is power of two, usually 256 or 512 bytes. Data can be programmed in page size units. The same page can be programmed multiple times and bits can be programmed one by one this way. Pages cannot be erased individually. See Figure 1.
Block	Composed of several pages. Usually a block contain 16, 256 or 1024 pages. Data can be erased in block size units. See Figure 1.
Logical page	Allocation unit of file system. User configurable option. Larger logical page needs more RAM and less pages in management area. Smaller logical page needs less RAM and more pages in management area. It shall be larger or equal to page size. Define: `PIFS_LOGICAL_PAGE_SIZE_BYTE`
Erase	Change data bits to one (logical high level) is called erasing. Only a whole block can be erased, which means all bits of the block are set to one.
Program	Change data bits to zero (logical low level) is called programming. One page can be programmed at a time, but the same page can be programmed multiple times. Therefore each bit of a page can be programmed individually.
Block address	Index of block in flash memory. Type: `pifs_block_address_t`
Page address	Index of a page in a block. It can be flash (physical) page address for flash functions or logical page address for file system's functions. Type: `pifs_page_address_t`
Page offset	Index of a byte in a page. It can be flash (physical) page offset for flash functions or logical page offset for file system's functions. Type: `pifs_page_offset_t`
Data area	Blocks which hold the file's content. See Figure 2.
Management area	Blocks which hold the file system's internal data, how much space is allocated, where are the data pages of files, etc. There are two types of management area: primary (active) management area, secondary (next) management area. See Figure 2.
File system header	First page of active management area contains the header of file system. It describes address of free space bitmap, entry list, delta map, wear level list, etc. Type: `pifs_header_t`, variable: `pifs.header`.

FSBM	Free space bitmap. It stores information about all logical pages of flash memory: 1 bit stores whether page is free, 1 bit stores if page is to be released.
Delta page	If content of a file is overwritten and original page cannot be overwritten because bits should be changed from 0 to 1, delta pages are added. When the original page is read from a given address the content of delta page is provided from a different address.
Map page	Management page which is used to describe data pages of a file.
Entry list	Technically a directory of file system. If directories are disabled, only root entry list exist.
Entry	One file or directory in the directory.
TBR	To be released. A page which was used, but now it can erased then allocated. If all pages in a block marked TBR, the block can be erased during next merge.
Merge	During merge management area is written, blocks can be erased, delta pages are resolved and entry list is compacted. The data is copied from primary to secondary management area (secondary management are became primary management area), old primary management area is erased and the next secondary management area is selected.

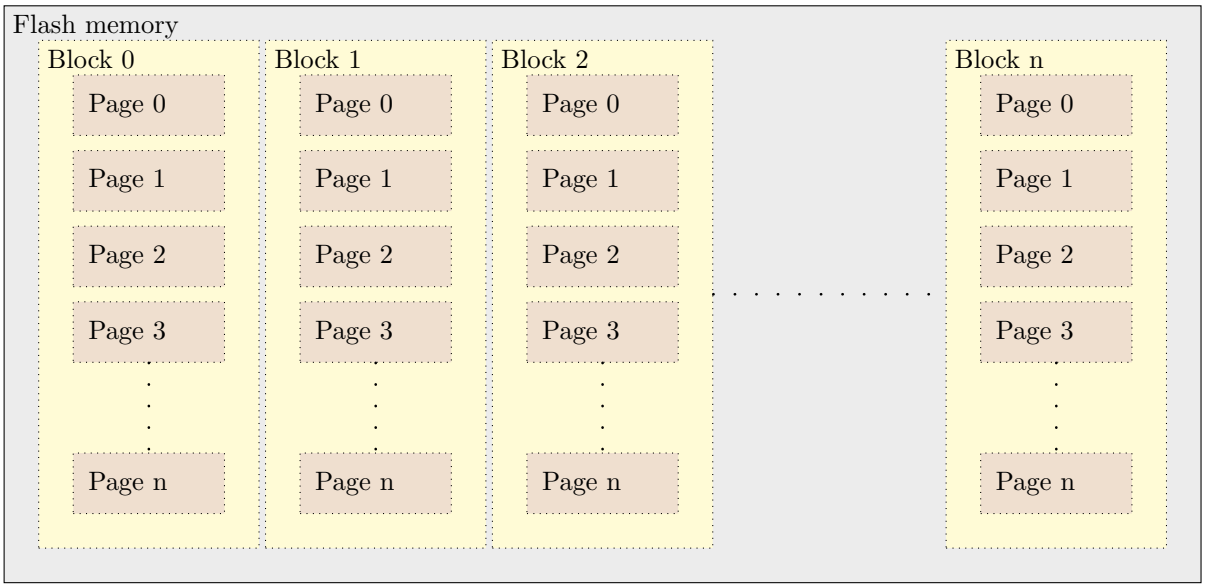

Figure 1: Flash memory layout

3 Demonstration

In the 'demo' directory four applications can be found. I developed them under Linux, but probably they can be compiled on Windows/cygwin as well.

1. pc_emu: PC application which uses NOR flash emulator. It can be compiled with GNU make an GCC.
2. maple_mini_pifs: Demo running on Maple Mini board, MCU is STM32F103RCBT6. There is no NOR flash, UART and debugger on board. So prototype should be created as can be seen in Figure 3. It can be compiled with GNU make and GCC or with STM32 System Workbench.
3. nucleo-f413zh_pifs: Demo running on Nucleo-F413ZH board, MCU is STM32F413ZH. There is no NOR flash memory on the board, some prototype hardware should be created as can be seen in Figure 4. It has internal debugger and virtual serial port. It can be compiled with GNU make and GCC or with STM32 System Workbench.

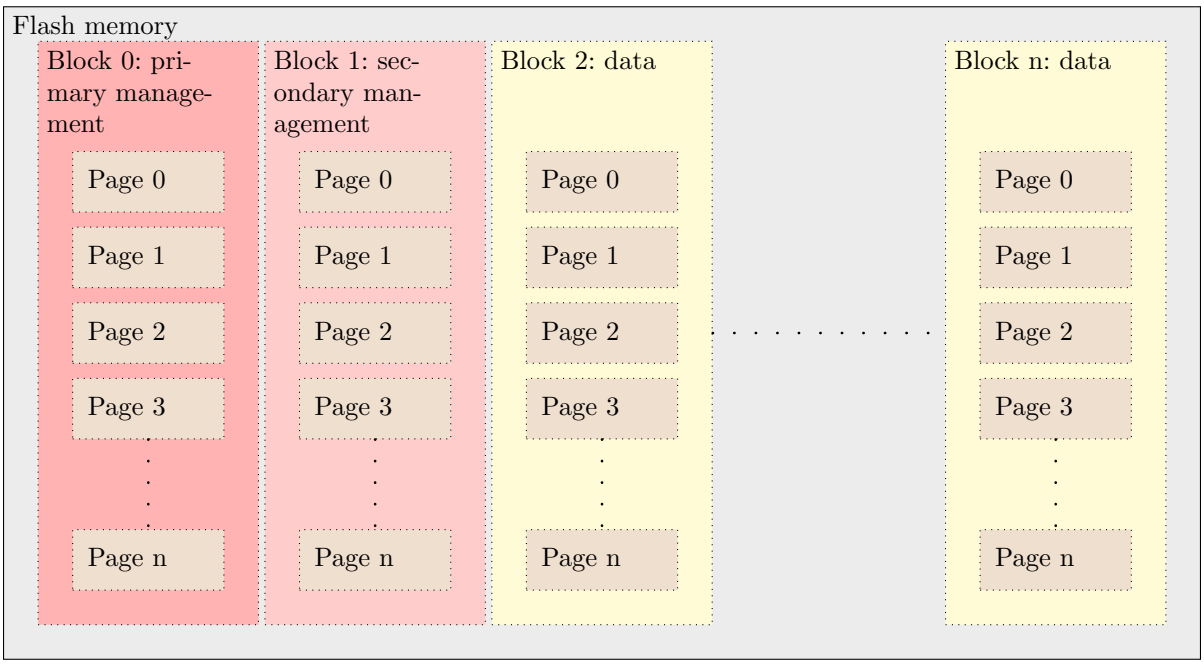

Figure 2: Flash memory layout when πfs installed

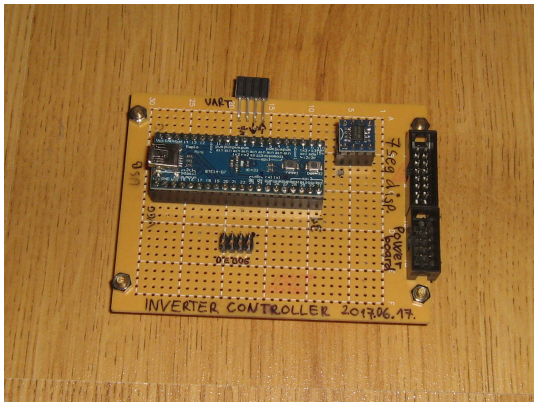

Figure 3: Maple Mini development board

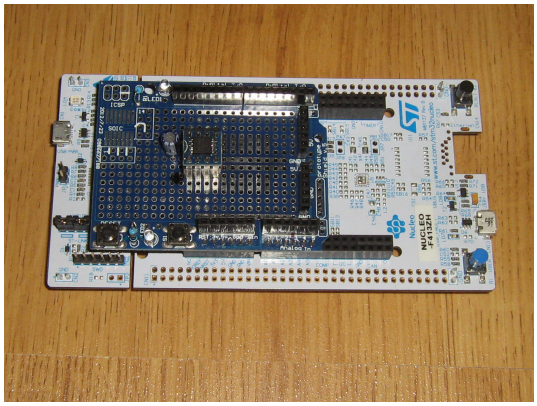

Figure 4: Nucleo-F413ZH development board

4. stm32_f4ve_pifs: The most complex demo is running on STM32-F4VE board, MCU is STM32F407VE. There is Winbond W25Q16DV NOR flash soldered on the board, but it has not got UART. So external debugger and USB-serial converter is needed. See Figure 5. This application uses DHT22 sensor to collect humidity and temperature data and store them to NOR flash. The data can be copied to MMC/SD card. It can be compiled with GNU make and GCC or with STM32 System Workbench.

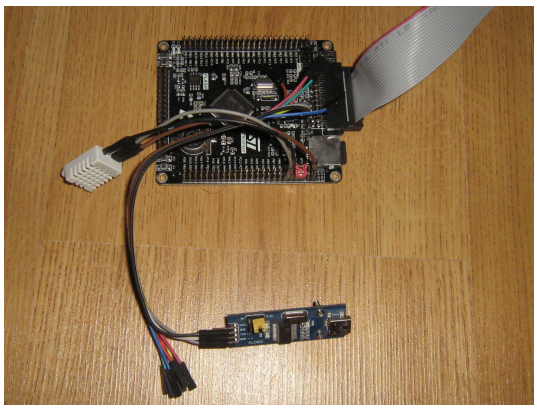

Figure 5: STM32-F4VE development board

3.1 Using demo

All applications based on a command interpreter which communicates over UART or standard input/output when PC is used. There are commands to list directory, create and dump files, run some tests and examine the file system.

Addresses are displayed in two formats at the same time:

1. `BAx/PAy` Block address (BA), page address (PA) format. Where 'x' is index of block and 'y' is index of page in the block. See Figure 6 Note: when logical page size is not equal to flash's page size it should be be checked which part of software uses the address.

2. `@0xnnnn` Address in hexadecimal format in flash memory. Logical and physical page size does not affect this address.

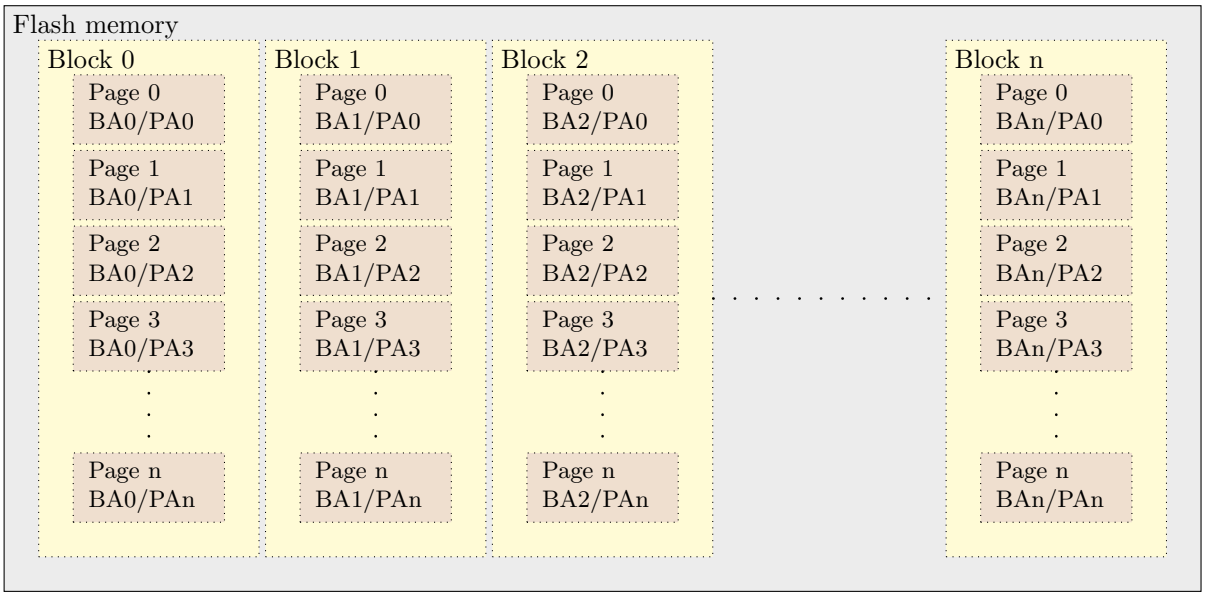

Figure 6: Addressing of flash memory

The commands:

ls	List directory. Switches: `-l` display file size, `-e` examine file (for debugging). `-e` switch displays address of first map page of file or entry list of directory.
l	List directory with switches: -l and -e (for debugging).
rm	Remove file. Switch: -a remove all files in the directory.
dumpf	Dump file in hexadecimal format.
df	Alias to `dumpf`.

cat	Read and print file to console.
create	Create file. You can type file until 'q' is entered in the first character of a line.
append	Append to file. You can type file until 'q' is entered in the first character of a line.
cd	Change directory. Only available if directories are enabled.
mkdir	Make directory. Only available if directories are enabled.
rmdir	Remove an empty directory. Only available if directories are enabled.
cwd	Get current working directory. Only available if directories are enabled.
dump	Read and show a flash page in hexadecimal format. First parameter is the address, second optional parameter in number of flash pages to display. Note: this command works with flash page size other ones work with logical page size! Example: `dump 0x10000 2`
d	Alias to `dump`.
map	Decode and display a map page. The parameter should be the address of map page. Example: `map 0x10D00`
m	Alias to `map`.
fd	Find delta pages of a page. Parameter is address of page.
info	Print information about file system.
i	Alias to `info`.
free	Print info of free space
f	Alias to `free`.
bi	Print info of block.
pi	Check if page is free, to be released or erased.
w	Print wear level list, which is stored in the file system's header.
lw	Print least weared blocks' list.
mw	Print most weared blocks' list.
eb	Empty block: copy files which resides on the specified block. This command should be used only on blocks which has no free pages at all. Used during developing of static wear leveling.
sw	Perform static wear leveling if necessary.
fs	Print flash's statistics. Available only on emulated NOR flash memory.
erase	Erase flash. It destroys file system!
tstflash	Test flash. It destroys file sytem during test!
tstpifs	Test Pi file system: run all file system tests (except large file).
tp	Alias to `tstpifs`
tb	Test Pi file system: basic. File name can be changed if parameter is given.
ts	Test Pi file system: small files.
tl	Test Pi file system: large file.
tf	Test Pi file system: full write.
tfrag	Test Pi file system: fragment.
tsk	Test Pi file system: seek.
td	Test Pi file system: delta.

tdir	Test Pi file system: directories. Only available if directories are enabled.
y	Debug command.
quit	Quit.
q	Alias to `quit`
noprompt	Prompt will not be displayed.
p	Parameter test.
help	Print help.
?	Alias to `help`.

Example use of commands on a flash M25P40, where logical page size is configured to the size of flash page size (256 bytes). First it generates small files with 'ts', then examines the map page of file 'small0.tst', then dumps the pages of file with 'd'. After that it dumps the file with 'df' which opens the file with `pifs_fopen()` and read with `pifs_fread()`. Then it prints some information about blocks, file system and free space.

```
> ts
...
> l
List directory '.'
small0.tst                              512  BA1/PA13 @0x10D00
small1.tst                              512  BA1/PA14 @0x10E00
small2.tst                              512  BA1/PA15 @0x10F00
small3.tst                              512  BA1/PA16 @0x11000
small4.tst                              512  BA1/PA17 @0x11100
small5.tst                              512  BA1/PA18 @0x11200
...
> m 0x10D00
Map page BA1/PA13 @0x10D00

Previous map: BA255/PA65535 @0x1FEFF00
Next map:     BA255/PA65535 @0x1FEFF00

BA2/PA0 @0x20000  page count: 2
> d 0x20000 2
Dump page BA2/PA0 @0x20000
00020000 46 69 6C 65 3A 20 73 6D 61 6C 6C 30 2E 74 73 74   File: small0.tst
00020010 2C 20 73 65 71 75 65 6E 63 65 3A 20 30 23 00 00   , sequence: 0#..
00020020 10 00 11 00 12 00 13 00 14 00 15 00 16 00 17 00   ................
00020030 18 00 19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00   ................
00020040 20 00 21 00 22 00 23 00 24 00 25 00 26 00 27 00    .!."#.$.%.&.'.
00020050 28 00 29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00   (.).*.+.,.-.../.
00020060 30 00 31 00 32 00 33 00 34 00 35 00 36 00 37 00   0.1.2.3.4.5.6.7.
00020070 38 00 39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00   8.9.:.;.<.=.>.?.
00020080 40 00 41 00 42 00 43 00 44 00 45 00 46 00 47 00   @.A.B.C.D.E.F.G.
00020090 48 00 49 00 4A 00 4B 00 4C 00 4D 00 4E 00 4F 00   H.I.J.K.L.M.N.O.
000200A0 50 00 51 00 52 00 53 00 54 00 55 00 56 00 57 00   P.Q.R.S.T.U.V.W.
000200B0 58 00 59 00 5A 00 5B 00 5C 00 5D 00 5E 00 5F 00   X.Y.Z.[.\.].^._.
000200C0 60 00 61 00 62 00 63 00 64 00 65 00 66 00 67 00   `.a.b.c.d.e.f.g.
000200D0 68 00 69 00 6A 00 6B 00 6C 00 6D 00 6E 00 6F 00   h.i.j.k.l.m.n.o.
000200E0 70 00 71 00 72 00 73 00 74 00 75 00 76 00 77 00   p.q.r.s.t.u.v.w.
000200F0 78 00 79 00 7A 00 7B 00 7C 00 7D 00 7E 00 7F 00   x.y.z.{.|.}.~..

Dump page BA2/PA1 @0x20100
00020100 80 00 81 00 82 00 83 00 84 00 85 00 86 00 87 00   ................
00020110 88 00 89 00 8A 00 8B 00 8C 00 8D 00 8E 00 8F 00   ................
00020120 90 00 91 00 92 00 93 00 94 00 95 00 96 00 97 00   ................
00020130 98 00 99 00 9A 00 9B 00 9C 00 9D 00 9E 00 9F 00   ................
```

```
00020140 A0 00 A1 00 A2 00 A3 00 A4 00 A5 00 A6 00 A7 00   ................
00020150 A8 00 A9 00 AA 00 AB 00 AC 00 AD 00 AE 00 AF 00   ................
00020160 B0 00 B1 00 B2 00 B3 00 B4 00 B5 00 B6 00 B7 00   ................
00020170 B8 00 B9 00 BA 00 BB 00 BC 00 BD 00 BE 00 BF 00   ................
00020180 C0 00 C1 00 C2 00 C3 00 C4 00 C5 00 C6 00 C7 00   ................
00020190 C8 00 C9 00 CA 00 CB 00 CC 00 CD 00 CE 00 CF 00   ................
000201A0 D0 00 D1 00 D2 00 D3 00 D4 00 D5 00 D6 00 D7 00   ................
000201B0 D8 00 D9 00 DA 00 DB 00 DC 00 DD 00 DE 00 DF 00   ................
000201C0 E0 00 E1 00 E2 00 E3 00 E4 00 E5 00 E6 00 E7 00   ................
000201D0 E8 00 E9 00 EA 00 EB 00 EC 00 ED 00 EE 00 EF 00   ................
000201E0 F0 00 F1 00 F2 00 F3 00 F4 00 F5 00 F6 00 F7 00   ................
000201F0 F8 00 F9 00 FA 00 FB 00 FC 00 FD 00 FE 00 FF 00   ................
> df small0.tst
File size: 512 bytes
Dump file 'small0.tst'
00000000 46 69 6C 65 3A 20 73 6D 61 6C 6C 30 2E 74 73 74   File: small0.tst
00000010 2C 20 73 65 71 75 65 6E 63 65 3A 20 30 23 00 00   , sequence: 0#..
00000020 10 00 11 00 12 00 13 00 14 00 15 00 16 00 17 00   ................
00000030 18 00 19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00   ................
00000040 20 00 21 00 22 00 23 00 24 00 25 00 26 00 27 00    .!."#.$.%.&.'.
00000050 28 00 29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00   (.).*.+.,.-.../.
00000060 30 00 31 00 32 00 33 00 34 00 35 00 36 00 37 00   0.1.2.3.4.5.6.7.
00000070 38 00 39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00   8.9.:.;.<.=.>.?.
00000080 40 00 41 00 42 00 43 00 44 00 45 00 46 00 47 00   @.A.B.C.D.E.F.G.
00000090 48 00 49 00 4A 00 4B 00 4C 00 4D 00 4E 00 4F 00   H.I.J.K.L.M.N.O.
000000A0 50 00 51 00 52 00 53 00 54 00 55 00 56 00 57 00   P.Q.R.S.T.U.V.W.
000000B0 58 00 59 00 5A 00 5B 00 5C 00 5D 00 5E 00 5F 00   X.Y.Z.[.\.].^._.
000000C0 60 00 61 00 62 00 63 00 64 00 65 00 66 00 67 00   `.a.b.c.d.e.f.g.
000000D0 68 00 69 00 6A 00 6B 00 6C 00 6D 00 6E 00 6F 00   h.i.j.k.l.m.n.o.
000000E0 70 00 71 00 72 00 73 00 74 00 75 00 76 00 77 00   p.q.r.s.t.u.v.w.
000000F0 78 00 79 00 7A 00 7B 00 7C 00 7D 00 7E 00 7F 00   x.y.z.{.|.}.~..

00000100 80 00 81 00 82 00 83 00 84 00 85 00 86 00 87 00   ................
00000110 88 00 89 00 8A 00 8B 00 8C 00 8D 00 8E 00 8F 00   ................
00000120 90 00 91 00 92 00 93 00 94 00 95 00 96 00 97 00   ................
00000130 98 00 99 00 9A 00 9B 00 9C 00 9D 00 9E 00 9F 00   ................
00000140 A0 00 A1 00 A2 00 A3 00 A4 00 A5 00 A6 00 A7 00   ................
00000150 A8 00 A9 00 AA 00 AB 00 AC 00 AD 00 AE 00 AF 00   ................
00000160 B0 00 B1 00 B2 00 B3 00 B4 00 B5 00 B6 00 B7 00   ................
00000170 B8 00 B9 00 BA 00 BB 00 BC 00 BD 00 BE 00 BF 00   ................
00000180 C0 00 C1 00 C2 00 C3 00 C4 00 C5 00 C6 00 C7 00   ................
00000190 C8 00 C9 00 CA 00 CB 00 CC 00 CD 00 CE 00 CF 00   ................
000001A0 D0 00 D1 00 D2 00 D3 00 D4 00 D5 00 D6 00 D7 00   ................
000001B0 D8 00 D9 00 DA 00 DB 00 DC 00 DD 00 DE 00 DF 00   ................
000001C0 E0 00 E1 00 E2 00 E3 00 E4 00 E5 00 E6 00 E7 00   ................
000001D0 E8 00 E9 00 EA 00 EB 00 EC 00 ED 00 EE 00 EF 00   ................
000001E0 F0 00 F1 00 F2 00 F3 00 F4 00 F5 00 F6 00 F7 00   ................
000001F0 F8 00 F9 00 FA 00 FB 00 FC 00 FD 00 FE 00 FF 00   ................

End position: 512 bytes
> bi
      | Type    | Wear  | Free pages  | TBR pages
Block |         | Level | Data | Mgmt | Data | Mgmt
------+---------+-------+------+------+------+------
    0 | Data    |     1 |  229 |    0 |   10 |    0
    1 | PriMgmt |     1 |    0 |  190 |    0 |    4
    2 | Data    |     0 |    0 |    0 |  224 |    0
    3 | SecMgmt |     1 |    0 |    0 |    0 |    0
    4 | Data    |     1 |  256 |    0 |    0 |    0
    5 | Data    |     1 |  256 |    0 |    0 |    0
    6 | Data    |     1 |  256 |    0 |    0 |    0
```

```
    7 | Data    |     1 |  256 |    0 |    0 |    0
> i
Geometry of flash memory
------------------------
Size of flash memory (all):         524288 bytes, 512 KiB
Size of flash memory (used by FS):  524288 bytes, 512 KiB
Size of block:                      65536 bytes
Size of page:                       256 bytes
Number of blocks (all):             8
Number of blocks (used by FS)):     8
Number of pages/block:              256
Number of pages (all):              2048
Number of pages (used by FS)):      2048

Geometry of file system
-----------------------
Size of logical page:               256 bytes
Block address size:                 1 bytes
Page address size:                  2 bytes
Header size:                        71 bytes, 1 logical pages
Entry size:                         48 bytes
Entry size in a page:               240 bytes
Entry list size:                    1792 bytes, 7 logical pages
Free space bitmap size:             512 bytes, 2 logical pages
Map header size:                    6 bytes
Map entry size:                     4 bytes
Number of map entries/page:         62
Delta entry size:                   6 bytes
Number of delta entries/page:       42
Number of delta entries:            84
Delta map size:                     512 bytes, 2 logical pages
Wear level entry size:              3 bytes
Number of wear level entries/page:  85
Number of wear level entries:       8
Wear level map size:                24 bytes, 1 logical pages
Minimum management area:            13 logical pages, 1 blocks
Recommended management area:        41 logical pages, 1 blocks
Full reserved area for management:  131072 bytes, 512 logical pages
Size of management area:            65536 bytes, 256 logical pages

File system in RAM:                 2456 bytes
Counter: 1
Entry list at BA1/PA1 @0x10100
Free space bitmap at BA1/PA8 @0x10800
Delta page map at BA1/PA10 @0x10A00
Wear level list at BA1/PA12 @0x10C00
> f
Free data area:                     320768 bytes, 1253 pages
Free management area:               48640 bytes, 190 pages
To be released data area:           59904 bytes, 234 pages
To be released management area:     1024 bytes, 4 pages
Free entries:                       10
To be released entries:             4
```

4 Using the file system

The API of file system is very similar to standard C API defined in `stdio.h`. The difference is that `pifs_fopen()` should be called instead of `fopen()` and before opening a file calling of `pifs_init()` is necessary. Furthermore only binary mode is supported, opening file in mode "r" is always "rb".

Example:

```
#include <stdint.h>
#include "api_pifs.h"

int main(void)
{
    pifs_status_t pifs_status;
    P_FILE      * file;
    uint8_t       test_buf_w[512];
    size_t        written_size;

    pifs_status = pifs_init();
    if (pifs_status == PIFS_SUCCESS)
    {
        file = pifs_fopen("demo.bin", "w");
        if (file)
        {
            written_size = pifs_fwrite(test_buf_w, 1, sizeof(test_buf_w), file);
            if (written_size != sizeof(test_buf_w))
            {
                /* Error */
            }
            if (pifs_fclose(file))
            {
                /* Error */
            }
        }
        pifs_status = pifs_delete();
    }
}
```

5 Porting flash driver

Defined types used by the flash driver:

pifs_block_address_t Unsigned integer type to address a block in the flash memory.

pifs_page_address_t Unsigned integer type to address a page in the block.

pifs_page_offset_t Unsigned integer type to address a byte in the page.

These types are automatically defined in `flash.h` according to geometry of flash. If maximum count of blocks is 4096, `pifs_block_address_t` will be `uint16_t`, if there are 64 blocks, it will be `uint8_t`.

The following functions shall be implemented to access the flash memory.

```
pifs_status_t pifs_flash_init(void);
```

Initialize flash driver. It shall return `PIFS_SUCCESS` if flash memory is identified and initialized.

```
pifs_status_t pifs_flash_delete(void);
```

De-initialize flash driver. It shall return `PIFS_SUCCESS` if flash memory was successfully de-initialized.

```
pifs_status_t pifs_flash_read(pifs_block_address_t a_block_address,
        pifs_page_address_t a_page_address, pifs_page_offset_t a_page_offset,
        void * const a_buf, size_t a_buf_size);
```

Read from flash memory. See Figure 6 how addressing works. Arguments:

a_block_address Address of block. Input.

a_page_address Address of the page in block. Input.

a_page_offset Offset in page. Input.

a_buf Buffer to fill. Output.

a_buf_size Size of buffer. Input.

It shall return PIFS_SUCCESS if read successfully finished.

```
pifs_status_t pifs_flash_write(pifs_block_address_t a_block_address,
        pifs_page_address_t a_page_address, pifs_page_address_t a_page_offset,
        const void * const a_buf, size_t a_buf_size);
```

Write to flash memory. See Figure 6 how addressing works. Arguments:

a_block_address Address of block. Input.

a_page_address Address of the page in block. Input.

a_page_offset Offset in page. Input.

a_buf Buffer to write. Input.

a_buf_size Size of buffer. Input.

It shall return PIFS_SUCCESS if write successfully finished.

```
pifs_status_t pifs_flash_erase(pifs_block_address_t a_block_address);
```

Erase a block. See Figure 6 how addressing works. Arguments:

a_block_address Address of block. Input.

It shall return PIFS_SUCCESS if block was erased successfully.

```
void pifs_flash_print_stat(void);
```

Called by the terminal to print information about flash memory. The body of function can be empty or can print useful data.