Pi file system (πfs)

Peter Ivanov <ivanovp@gmail.com>

August 19, 2017

1 Introduction

This file system was developed for embedded systems which use NOR flash as storage. 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:

- 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 datasheets. One page is usually 256 or 512 bytes, one block consits 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.

1.1 Features of the πfs

- Small memory footprint
- Files can be opened for update ("r+", "w+", "a+" modes are supported)
- 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 (limited, work-in-progress)
- 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

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
- No OS support yet, file system can be used only from one task
- 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. See Figure 1.
Block	Composed of several pages. Usually a block contain 16, 256 or 1024 pages. See Figure 1.
Erase	Change data bits to one (logical high level). 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). 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. Type: pifs_page_address_t
Page offset	Index of a byte in a page. 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 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.
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.
Merge	During merge management area is written, blocks are erased, delta pages are resolved and entry list is compacted. The data is copied from primary to secondary management area and the next secondary management area is selected.

3 Demonstration

In the 'demo' directory three applications can be found.

1. pc_emu: PC application which uses NOR flash emulator. It can be compiled with GNU make an GCC.
2. nucleo-f413zh_pifs: Demo running on Nucleo-F413ZH board. There is no flash memory on the board, some prototype hardware should be created. It has internal debugger and virtual serial port. It can be compiled with GNU make and GCC or with STM32 System Workbench.
3. stm32_f4ve_pifs: Demo running on STM32-F4VE board. There is W25Q16DV NOR flash soldered on the board, but it has not got UART. So external debugger and USB-serial converter is needed. It can be compiled with GNU make and GCC or with STM32 System Workbench.

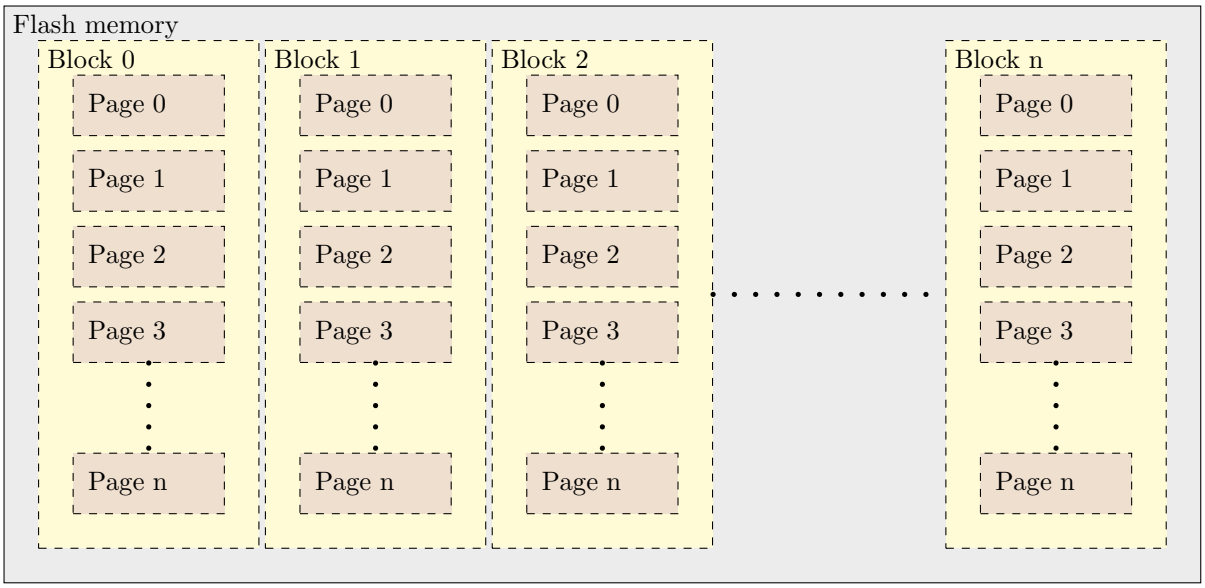

Figure 1: Flash memory layout

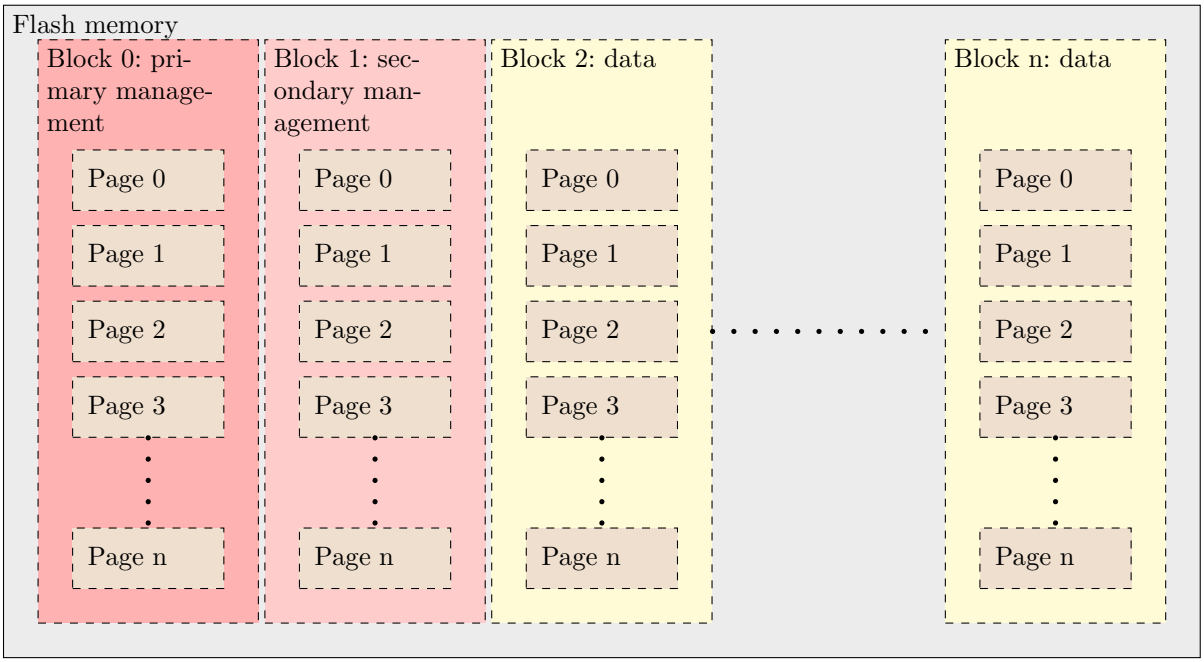

Figure 2: Flash memory layout when πfs installed