Embedding it better...

µTasker Document

µTasker – Fail-Safe SPI Flash FAT with Wear Levelling

Table of Contents

1. Introduction....3
2. Reading and Writing SPI Flash....4
3. Erasing SPI Flash....4
4. Block Management....5
5. Block Management Storage Concept....7
1.RAM Arrays....7
2.Non-volatile SPI Flash Information....7
3.Assigning FAT Sectors to Physical Flash FAT Sectors to FAT Sectors....9
4.Cleaning Up “Dirty” Unassigned Physical Flash Sectors....9
6. Conclusion....11

1. Introduction

SPI based NOR Flash is popular in embedded systems due to the fact that it offers a good amount of reliable Flash memory (several hundred kBytes to several tens of Mbytes) at an attractive price. Typically the devices are in SO-8 packages and are thus small and require only a 4 line SPI interface to be controlled by any embedded microcontroller. Multiple SPI Flash devices can be connected to the same SPI bus by assigning one chip select (CS) line to each, which allows increasing the available memory size when needed.

A typical SPI Flash can be written in units of a byte or multiple bytes, whereby usually cumulative writing of '0's to a single byte is possible. Writes may internally be organises as pages, even if the concept of pages is not necessarily relevant for application layer software.

The flash can be deleted in blocks which are typically referred to as Flash sectors. Possibly the most important characteristic of the flash is its sector size due to the fact that it is the smallest area of Flash memory that can be deleted at one time. Since individual bits in flash that are programmed to '0' can only be programmed back to '1' by erasing the Flash sector that they belong to, the erasure of at least this amount of memory is necessary as soon as any bit in the area needs to be returned to the '1' state.

Usually each sector can withstand at least 100'000 or more programming/erase cycles (endurance) in its lifetime.

This document details the implementation of a file system (utFAT) in SPI based Flash which ensures safe operation in embedded system environments where power loss or other interference (resulting in system resets) cannot result in serious data loss or FAT corruption. Furthermore it explains the implementation of block management and wear levelling, which ensures that the programming/erasure load is spread over the available Flash sectors to avoid premature wear during normal FAT operation.

The Spansion S25FL164K 64Mbit (8MByte) SPI Flash device is used throughout the discussion but the operation is also valid for most general SPI Flash devices.

2. Reading and Writing SPI Flash

The Spansion S25FL164K supports reading of a byte from any internal address. Multiple bytes can be read by continuously reading data once an address has been set, which means that it is possible to read the entire SPI Flash content by commanding a read starting at its internal address 0 and reading 8 Mbytes of content. *It is possible to read at 50Mb/s from the SPI Flash*.

FAT operations often require reading a FAT sector (512 bytes) from the storage medium and is therefore performed by commanding the internal address of the start of the sector in question and reading this amount of data from it.

The Spansion S25FL164K supports page writes. A page is 256 bytes in size and is aligned to the internal address of the page's boundary; 0x000, 0x100, 0x200 are thus the internal start addresses of the first three pages in the device. *A page write can only be performed after a "write enable" command has first been sent to the SPI Flash and this write enable is automatically reset after each page write has terminated*. A page write can address any byte within the page to be written but can only write up to the end of that page in a single page write operation. This means that 256 bytes could be written in one go from the internal start address 0x000 but only 1 could be written from the internal start address 0x0ff. To write more data from 0x100 would need a second page write starting at that address. Programming 256 bytes of data to a single Flash page takes typically 700us to complete.

FAT operations often require writing a FAT sector (512 bytes) to the storage medium and so is performed by commanding two page writes of 256 bytes each in two contiguous SPI Flash pages. The internal start address of each page is automatically aligned to SPI Flash pages.

It is to be noted that bits in the SPI Flash can be programmed from '1' to '0' but not from '0' to '1' without previously erasing the complete SPI Flash sector that it resides in – see the next section for more details.

3. Erasing SPI Flash

The Spansion S25FL164K's Flash is divided into 2048 x 4k sectors which can be erased individually. This means that whenever a bit within any byte in a 4k sector needs to be changed from '0' to '1' the complete Flash sector that it resides in must be erased and subsequently reprogrammed. *It takes typically 50ms to erase a single Flash sector.*

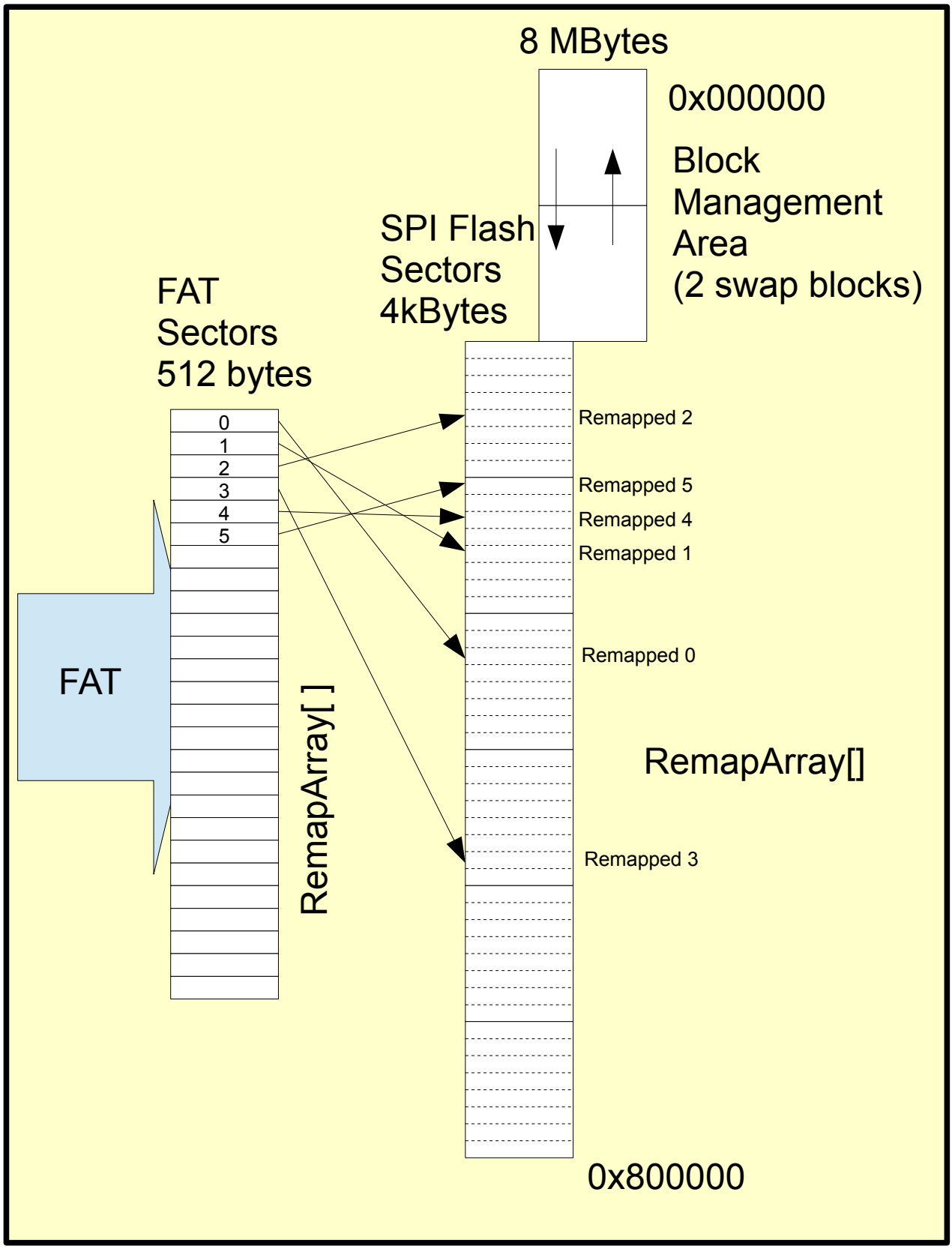
Since FAT works with FAT sectors of 512 bytes a single Flash sector contains 8 FAT sectors. When any one FAT sector needs to be deleted (or any bit of its content needs to be changed from '0' to '1') another 7 neighbouring FAT sectors also potentially need to be erased and reprogrammed.

4. Block Management

A basic analysis of the operation of SPI Flash and FAT results in the conclusion that a FAT sector should not be fixed at a particular Flash page/sector. This is due to the fact that file systems will concentrate activity at certain FAT sectors, which would then result in a certain SPI Flash page/sector needing to be frequently programmed and erased. Although few other SPI Flash pages/sectors may ever have been used the ones with high FAT sector utilisation could already have been subjected to high levels of stress and prematurely fail.

Furthermore, since SPI Flash sectors are larger than FAT sectors it is practical to be able to move the location of a FAT sector in the SPI Flash memory map when it needs to be modified in order to avoid having to erase and reprogram neighbouring FAT sectors.

Block management should therefore allow FAT sectors to be mapped to SPI Flash pages/sectors (including remapping to allow individual FAT sectors to be moved to different physical SPI Flash locations). By intelligently remapping the FAT sectors to physical SPI Flash locations that have been subjected to little programming/erase cycles the block management also forms the foundation of wear levelling too.


8 MBytes
0x000000
Block
Management
Area
(2 swap blocks)
SPI Flash
Sectors
4kBytes
FAT
Sectors
512 bytes
0
1
2
3
4
5
Remapped 2
Remapped 5
Remapped 4
Remapped 1
Remapped 0
Remapped 3
FAT
RemapArray[ ]
RemapArray[]
0x800000

5. Block Management Storage Concept

Fail safe operation includes FAT and also its underlying block management/wear levelling. If, for example, the FAT were intact but the block management were to lose track of the relationship between the FAT and the physical SPI Flash storage it would also result in overall corruption. This quickly leads to the realisation that the SPI Flash is not used only for non-volatile FAT storage but is also used for non-volatile storage of all critical data related to its block management!

In order to keep track of the blocks as their mapping is changed and which are valid or need deleting there are various variable arrays that need to be maintained. During operation these arrays are generally located in RAM so that they can be quickly accessed and modified. At the same time some of the new data has to be retained across resets or power cycles and so also needs to be committed to SPI flash based storage too, although not necessarily in the same form as the RAM arrays. These different types of variables are discussed here.

1. RAM Arrays

During operation there is need for an array which keeps track of the mapping of FAT sectors to SPI FAT sectors. This is called `RemapArray[]` and has one entry for each possible FAT sector used by the file system. It contains a reference to the present location (or address) of this FAT sector in the SPI Flash.

Each physical SPI Flash sector (4k in size) has an entry in the `WearLevelArray[]` which holds the number of times that it has been used (programmed and erased) since a certain point in time. A single byte is allocated for each sector which allows 255 program/erase cycles to be counted but each disk has a long word `ulWearLevelBaseCount` which holds the base program/erase cycle count of the complete flash disk. The exact value is therefore equal to the base erase cycle count plus the individual sector's erase cycle count. The wear-levelling operation attempts to keep the erase count across all sectors as equal as possible and the difference value in the wear level array is reduced whenever the the base count value is increased.

Each SPI FAT sector (used to map FAT sectors to) has a boolean (bit) entry in the `DirtySectors[]` array which tracks the content is unused (erased) or not. Unused SPI FAT sectors can be used to write FAT sector content to since there are no restrictions to writing to them.

Each SPI FAT sector (used to map FAT sectors to) has a boolean (bit) entry in the `UsedSectors[]` array which tracks whether it is actively remapped to a FAT sector or not. These spare physical sectors are available for use for new remapping as long as they are not marked as dirty (not erased). *There is a need for more physical FAT sectors than FAT sectors used by the file system because there will inevitably be some old mapped physical sectors that are no longer used but are still dirty and so can not yet be reused*.

2. Non-volatile SPI Flash Information

A part of the SPI Flash's content is used to store non-volatile information concerning the arrays used at run time. Each time the run-time RAM arrays change there is a potential requirement that the new state needs to be maintained across the next reset/power cycle so that it can be used again after the next restart.

Since the RAM content can change a great many times and there is a limited amount of the SPI Flash space that should be used for this task, plus the fact that this area needs also to be used as little as possible to avoid wearing it out, the change information is kept as small as possible. These pieces of change information are each time tagged onto the present management block, utilising the fact that it is possible to contiguously write to the block content which is originally erased (at 0xff). Erases of the area are avoided until necessary (when the block becomes completely full with change information) but it is important that a power failure or reset taking place when the block management is deleted doesn't result in losing any information, which would then lead to file system corruption.

This is secured by having a swap block in the Block Management Area. When one becomes exhausted (cannot accept any more change information) the present status is written to the swap block in a consolidated form before the other active block is deleted, leaving the consolidated one as base for further change information to be written to. The swap block thus ensures that a reset during an update of the Block Management Area can not result in loss of information and potential loss of data in the file system.

The content of the Block Management Area (each of the two swap blocks) consists of

- a status so that it an be detected whether its content is valid or not
- a copy of the number of erase cycles that each physical 4k Flash sector has been subjected to when this Block Management Area was originally created
- a copy of the remap array when this Block Management Area was originally created
- a number of change notices which are originally empty. The number of change notices that can be added depends on the remaining space in the Block Management Area and thus on its physical size that is defined by `MANAGEMENT_BLOCK_SIZE` (*this is a multiple of the physical Flash sector's size since it needs to be deleted as a unit*).

Each time there is a change made during run-time, a change notice may be added to the Block Management Area so that the RAM table can always be reconstructed from the content whenever needed. Each change notice entry is of a fixed size but is kept as small as possible so that the maximum number of change notices can be written before the Block Management Area becomes exhausted and a consolidated swap has to be performed.

3. Assigning FAT Sectors to Physical Flash FAT Sectors to FAT Sectors

Initially, when the Flash disk has not yet been formatted, no physical FAT sectors are mapped to FAT sector. Erased Flash (assuming each mapping entry is 16 bits in size and can thus map up to 65534 FAT sectors) is 0xffff, which is interpreted as not being mapped.

Each time the file system writes a FAT sector that has no physical storage allocated to it a new one is assigned. Any empty (erased) physical Flash FAT sector can be used but preference is for a physical FAT sector in a Flash sector that has the lowest erase count.

The physical FAT sector is mapped to the FAT sector, marked as in use and also as not being empty (dirty) because it is assumed that the write will not be of 512 x 0xff. The data content is then written to the physical FAT sector before a change notice is written to the management area.

Should the content be written to the newly assigned physical FAT sector but the change notice not be posted (eg. due to power loss in the sequence) the result is that the physical FAT sector will be marked as a “dirty” unused block that can be erased at a suitable time.

Each time the content of a FAT sector is written to when there is an assigned physical Flash FAT sector belonging to it, it will cause a new physical Flash FAT sector to be allocated. Again a free one will be chosen from a Flash sector with the lowest erase count. The one that was used previously is marked as being no longer allocated; it is “dirty” but not assigned to the FAT.

The data content is written to the new physical Flash FAT sector and then the change notice is written to the management area. The change notice informs of the new mapping, which is automatically informing of the fact that the original one is no longer in use.

Should the content be written to the newly assigned physical FAT sector but the change notice not be posted (eg. due to power loss in the sequence) the result is that the physical FAT sector will be marked as a “dirty” unused block that can be erased at a suitable time. The original physical Flash FAT sector is still valid since the change didn't complete meaning that no data loss could take place.

As long as there are free physical Flash FAT sectors available to be assigned to a write of modified data to a FAT sector this results in a write without any erasures. Since Flash sector erasures require much longer than a page write, multiple FAT sector writes to the same sector require only the SPI Flash write time (typically 1.4ms per FAT sector) rather than an erase (typically 50ms) plus a write time.

Note that an attempt to read a FAT sector that is not assigned to a physical Flash FAT sector results in data content consisting of all '0' to be returned.

4. Cleaning Up “Dirty” Unassigned Physical Flash Sectors

When the file system starts operation there will be few allocated FAT sectors and so there will be enough unassigned physical Flash FAT sector to allow writing initial FAT sector data content or overwriting FAT sector data.

With time there will be ever more physical Flash FAT sectors that are marked as being “dirty” and unused. At some point there could be a situation when no further unused/empty physical Flash FAT sectors exist and so no further writes are possible until these are freed up (erased).

The following rules can be determined by studying the basic operation:

1. There must be *at least* one Flash sector worth of free physical Flash FAT sectors somewhere in the physical Flash FAT area (4k in the case of the S25FL164K) in order to be able to recuperate a Flash sector for further remapping use. This allows the content of a single FAT sector to be copied and remapped to them in order to create a complete Flash sector of unused but dirty physical Flash FAT sectors, which can then be erased to result in a Flash sector worth of newly available physical Flash FAT sectors.
2. When all physical Flash FAT sectors in a single Flash sector are unused but dirty (when the last one is remapped to another location) an erasure of the Flash sector should be performed. This frees up the sector for reuse.
3. When the full FAT content is used it is advantageous to have a number of spare physical Flash FAT sectors available to simplify (make more efficient) subsequent FAT sector remapping when required.
4. If there are FAT sectors who's content never changes it improves overall wear levelling when they are nevertheless moved around if the Flash sectors they occupy are found to be much less used than other Flash sectors.
5. Since Flash sector erases take a long time (50ms typically) it is best to perform them when the file system is not being used (eg. as a background clean-up operation when there is no activity). *If there are no free physical Flash FAT sectors available when needed a clean-up as mentioned in rule 1 will be required, which results in a slow write due to the fact that it must also wait for the erasure to complete before being able to terminate the write. The background clean-up operation attempts to avoid this occurring by cleaning up before it becomes urgent.*
6. SPI Flash devices usually support suspend/resume operations which allows a Flash Sector erasure that is presently in operation to be temporarily interrupted so that other sectors can be read or programmed. Since Flash sector erases are normally performed exclusively as a background clean-up operation the SPI FAT read/write interface implementation should make use of this to avoid being blocked for the long erasure time in case a Flash sector delete is in progress when a new FAT operation is required.

6. Conclusion

Open.

Modifications:

V0.00 23.04.2015: