smxFLog™
Flash Logger

smxFLog provides high-speed, reliable data logging to NAND or NOR flash memory.

Logging data is a common operation in embedded systems, and warrants a good solution. It is a sequential operation consisting of appending data to a file. This is not efficient in FAT file systems writing to flash media.

The Problem with FAT File Systems for Use with Flash Memory

The problem with a using a FAT file system for logging data to flash is that, unless a full cluster is appended each time, the current partial cluster must be read to RAM from flash, the new data appended, and the new partial cluster written back to a clean area of flash. Each time a new cluster is written, the FAT must be updated to point to the new cluster and the file's directory entry size must be updated. Since flash cannot be overwritten, the FAT and directory records must be read to RAM, modified, then written to clean areas of flash.

This is too much overhead to add a small amount of data to a data log. It hurts performance and increases flash wear. It also means *garbage collection* (i.e. freeing up and erasing flash blocks) is frequently required. Garbage collection is a lengthy operation that can temporarily stall further logging. Hence, although a FAT file system can be used for data logging with magnetic media (which can be overwritten) it is not a good choice for data logging with flash memory.

Features

- Works with NAND, NOR, or serial NOR flash.
- Can be used with any size flash memory.
- Supports multi-chip flash arrays.
- Uses the same low-level drivers as smxFS and smxFFS.
- Efficient and fast:
  - Read vs. raw speed: NOR 98%, NAND 86%
  - Write vs. raw speed: NOR 88%, NAND 69%
- Designed for reliable use.
  - ECC mode.
  - Read back and verify mode.
  - Simple, safe API.
  - Power fail safe.
- Supports multitasking
  - Optimized for SMX®.
  - Can be ported to any RTOS or run standalone.
- Small:
  - ROM: 4 KB with ECC, 2 KB without ECC.
  - RAM: 288 bytes with ECC: 32 bytes without ECC.
- Full source code included
- Can share flash with smxFS, smxFFS, boot code, and application code

Writing

By comparison, smxFLog can append new records to a data log stored in flash without moving any data. Also, it has no FAT, no directories, nor any mapping tables requiring updating. It starts from the beginning of the flash partition that has been allocated to it and writes

one flash record at a time, in sequential order. If read back verification is enabled, it compares the data it wrote, and if it does not match, the record is marked bad and it is written to the next location and compared again. Wear leveling is accomplished automatically by the sequential write process. smxFLog can be configured to either stop at the end of the partition allocated to it, or to cycle back to the beginning, erase the oldest blocks, and reuse them. This is called *recycle* mode. A write pointer is maintained in RAM to point to the next available flash record to write.

Reading

A read pointer is maintained in RAM to point to the next flash record to read. Reading and writing may occur simultaneously. Any number of records may be read out at once up to the record currently being written, or the end of the flash partition, if not in recycle mode.

Flash Records

The flash record size must be a power of 2. In addition, it is limited by the nature of the flash memory. Most NAND flash chips, have a *small* page, which is 512 bytes. This is true for small to medium NAND flash chips. Large flash chips use a 2048 byte, *large* page. However, the same 512 byte flash record size applies to them also. The limiting factor is that most NAND flash chips permit only 3 writes to a small page or 12 writes to a large page. These limitations apply separately to data and spare areas. For reliable write operation, it is necessary to write twice to the status byte, in addition to possibly setting the read flag. For the rare flash chips that can tolerate more writes per page, smxFLog will support a smaller flash record (e.g. 256 bytes if 6 writes are permitted to a small page). In NAND flash the status byte and ECC bytes are stored in the *spare* area of each page and do not reduce space for data storage.

NOR flash does not have pages and hence does not have the above limitation on flash record size. However, the status byte and ECC bytes must be stored at the end of the flash record, since there is no spare area. They require 4 bytes. Hence, a 32 byte flash record loses 4 bytes (12.5%) of data. This is probably a practical lower limit. For a record size of 256, overhead is only 1.56%. However, a small record size may be needed for NOR flash since there may not be much of it in a typical system.

Matching Data Records to Flash Records

Data record sizes are determined by the application. If they are very small, it is desirable to buffer several in order to fill a flash record as full as possible. In other cases (e.g. audio records) the data record may be much larger than the flash record and will be spread over several flash records. In general, you should design NOR flash into your system for logging small data records or NAND flash for larger ones, as discussed in the previous section. If you have multiple synchronized data streams, you may be able to create a data record structure that has entries for each. If the data streams are asynchronous you may need to add headers to your data records to identify what data each contains. Then data from the data streams can be interleaved. This may not work if you wish to read back one stream at a time, but it will work fine if you plan to transfer all streams together to a host computer which will then sort them out.

Using a simple data logger like smxFLog in a complex data collection environment may not be optimum. In that case, smxFFS or smxFS file system might be the better choice. However, the performance of a FAT file system may not be sufficient. In that case, a combination of smxFLog in one partition and a file system in another partition may best meet your requirements.

API

The smxFLog API is simple and designed to help minimize programming errors.

int **sfl_Init**(uint iFlag)

Calls the low-level flash driver to initialize the flash chip and to retrieve basic information, such as the block size and the total number of blocks. Then, as determined by iFlag, initializes smxFLog pointers or erases the flash.

int **sfl_Release**(void)

Resets the smxFLog pointers and calls the low-level flash driver to release the hardware resources.

int **sfl_Read**(u8 *pRecord, uint iNumRecords)

Reads one or more records from the flash memory starting at the record pointed to by the read pointer. Bad and partial records are skipped.

int **sfl_ReadPtrMark**(bool iEraseOldBlocks)

Marks the flash record to which the read pointer (stored in RAM) points, by setting the read status flag. This avoids reading records again should power be lost, then restored. If iEraseOldBlocks is true, preceding old blocks will be erased

int **sfl_ReadPtrRestore**(void)

Restores the read pointer to the last position marked in flash so the application can restart reading from the last known read pointer. This is useful if data is lost during transmission.

int **sfl_Write**(u8 *pRecord)

Writes a new record to the flash log following the last record. After writing the record, the write pointer is advanced to the next free record.

int **sfl_Erase**(uint iFlag)

Erases the oldest block, all old blocks, or all flash blocks in the partition, as specified by iFlag.

Reliability Features

We recognize that for a logger to fail when deep down a drill hole, in the middle of the ocean, or in some other remote place can be expensive. Therefore, we have designed smxFLog to be reliable.

API

As mentioned above, the API has been structured to try to prevent common programming errors due to not understanding the peculiarities of flash, especially NAND flash. In addition, not shown above, extensive error checking is performed in order to permit defensive programming. All call return values should be tested and identifed errors handled at the application level.

Read Back and Verify Write Mode

This mode of operation can be enabled at compile time and should be used if the logging rate permits. If enabled, smxFLog reads the data just written to the flash and compares it with the data it tried to write. If they don't match, it marks the record bad and tries to write the record to the next location in flash, and compound again.

ECC

smxFLog uses a 22-bit Error Correction Code, which is capable of correcting a single bit error in up to 256 bytes of data. For 512 byte records, two ECCs are required. Enabling ECC is recommended particularly if NAND flash is being used since it can start losing bits as it ages.

The ECC is generated before the data is actually written to the flash chip. When data is read back from the flash chip, if it has a correctable error, the corrected data is returned. If the data has an error that cannot be corrected, an error is reported.

Power-Fail Safety

It is important in data logging applications that loss of power not result in the loss of data already stored in the flash memory. This cannot be guaranteed with a FAT file system, such as smxFS, because such a file system is vulnerable to massive data loss whenever the FAT is in the process of being changed. This weakness can be overcome by means of journaling or other mechanisms at the expense of even more complexity and overhead and lower performance.

Some non-standard file systems, such as smxFFS, are designed to be power-fail-safe but they require a lot of RAM to do so. This makes them unsuitable for many low-end SoCs with only on-chip SRAM and no external RAM.

smxFLog is power-fail safe because of its simplicity. Records are written sequentially, beginning at the start of the flash partition. There are no FAT, directories, maps, or other control structures that might be corrupted. A flag in the flash record is set when starting to write a flash record. A second flag is set when the write is complete. If power fails in between, the record will be recognized as a partial record, when later

read, and skipped over. Hence, the most that can be lost is the last record being written. The data records that fill flash records should be time stamped or sequence numbered so that the application can recognize when records have been lost due to power failure or ECC error detection.

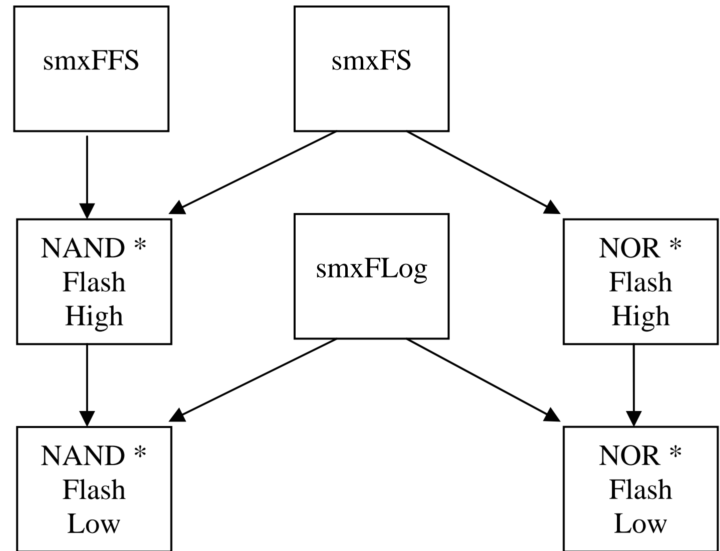
Combining smxFLog with Other File Systems

File systems are useful because they allow storing multiple files and directories. Moreover, a DOS/Windows FAT file system, such as smxFS, allows media to be shared with other computers. Even non-removable media in an embedded target can be read by the file system on a PC if the target is running the smxUSBD device stack with the mass storage function driver or the smxNS TCP/IP stack with FTP. In the first case, a disk stored in resident flash will look like a disk to a PC connected via USB.

For the above reasons, it can be beneficial to use smxFLog and smxFS in the same system. Data can be logged reliably by smxFLog and periodically, or at the end of a run, offloaded to the file system. smxFLog and smxFS can coexist in the same flash, in separate partitions, and they share the same low-level flash driver, so there is only one driver to port and no driver duplicated code.

Although smxFFS is not DOS/Windows compatible, it is a file system, and there may be advantages to combining it with smxFLog. It is simpler and lower-cost than smxFS, and it is power-fail safe, so it might be useful, for example to log many streams of low-speed data via smxFFS while smxFLog logs one or more high-speed data stream. If there are pauses in the high-speed data stream(s), it might be desirable to offload data from smxFLog to smxFFS in order to better organize it. As with smxFS, the low-level flash driver is shared.

The diagram below illustrates the relationship between all these data storage products.



*Drivers

Size

C.1 Code Size

Code size varies depending upon CPU, compiler, and optimization level.

	ARM7/9 IAR	ColdFire CodeWarrior
smxFLog (without ECC)	2 KB	2.5 KB
smxFLog (with ECC)	4 KB	5 KB

C.2 Data Size

smxFLog was designed to minimize RAM usage:

smxFLog core	32 B
smxFLog ECC (disabled by	256 B

default)	
smxFLog readback verify (disabled by default)	flash record size

Performance

NAND: LPC2468

Record Size, Bytes	Raw Data Read/Write KB/s	smxFLog Read/Write KB/s
512	1795/1638	1438/851
1024	2184/1956	1657/1352

NAND: MCF5282

Record Size, Bytes	Raw Data Read/Write KB/s	smxFLog Read/Write KB/s
512	2730/1260	2338/712
1024	2730/1260	2338/910

Note: Performance is reduced by a slow (220 ms) processor bus on the test board

NOR: LPC2468

Record Size, Bytes	Raw Data Read/Write KB/s	smxFLog Read/Write KB/s
64	2048/195	793/155
128	2048/195	1333/169
256	2048/195	1338/185
512	2048/195	2000/185

NOR: MCF5485

Record Size, Bytes	Raw Data Read/Write KB/s	smxFLog Read/Write KB/s
64	5461/264	4000/215
128	5461/264	5376/232
256	5461/277	5397/240
512	5461/282	5397/247