smxNOR™

Flash Driver

The smxNOR flash driver makes NOR flash memory appear to a file system like a disk drive.

General

smxNOR works with both the smxFFS Flash File System and the smxFS FAT file system. It is a two-level driver. The high-level driver presents a sector-oriented interface to a file system. The low-level driver presents a hardware-independent interface, or abstraction layer, to the high-level driver. The standard low-level driver interfaces directly to standard NOR flash chips and serial NOR flash chips. Small modifications are normally required to adapt it to specific hardware configurations. smxNOR can be restricted to operate within a *partition* of NOR flash memory, so that other partitions can be used for other purposes, such as for execute-in-place code.

smxNOR is designed for use in embedded systems. It has a small code footprint and a very small RAM footprint. Performance is moderate. This is normally adequate because NOR flash does not have fast write speed. The algorithm chosen favors small- to medium-size NOR flash memories in order to reduce RAM usage. smxNOR is best used to store static data such as downloaded code, tables, HTML pages, etc. It can also be used to log data being acquired at low rates. If high-speed operation or large capacity is needed, the smxNAND flash driver is recommended. For simple data logging operations, with very small flash and RAM space, smxFLog may be the best choice.

Features

- Small RAM usage – typically 700 bytes.
- 10KB code footprint.
- Moderate performance – 1500 KB/s read, 100 KB/s write for typical NOR flash and processors.
- Power fail safe.
- Block reclaim.
- Static and dynamic wear leveling.
- Read-back verification.
- Works with smxFFS and smxFS.
- Easily portable to other file systems.
- Two-level structure facilitates supporting different NOR devices.
- Support for standard and serial NOR devices
- Supports Common Flash Interface (CFI) NOR chips and others, and Hardware Interface Layer (HIL) is easily modifiable for other NOR devices
- Can be restricted to a partition of flash memory
- Sector-oriented interface to file system.

Definitions

- **Block**: Minimum erasable unit of a flash chip. (Note: some flash chip vendors use the term *sector*, instead.)
- **Flash partition**: The portion of the NOR flash memory allocated to smxNOR.
- **Sector**: Minimum unit of data for a file system. It is 512 bytes, by default.
- **Free Sector**: An erased sector ready to store new data. (Each byte contains 0xFF.)

NOR Flash Requirements

- Must support block erase (changes all cells to 0xFF)
- Each byte or word can be written at least 3 times, changing selected bits from ‘1’ to ‘0’ each time, without disturbing other bits.

Limitations

- Uses only full size flash blocks. Small contiguous blocks can be merged together to form full blocks that start on full block boundaries.
- Intended for moderate performance. Use smxNAND or smxFLog for data streaming.
- A portion of each block is used for control information, so the total data capacity is somewhat less than the total media size.

Structure

smxNOR is a two-level flash driver:

- High-Level Driver
- Low-Level Driver or Hardware Interface Layer (HIL)

High-Level Driver

The high-level driver performs most of the functions of smxNOR and it makes a NOR flash memory partition look like a disk drive to a file system.

API

The following API is presented to the file system and the application: (smxFS includes a wrapper to map its standard driver interface to this interface – see the smxFS data sheet. Some functions, such as nor_BlockReclaim() may be called directly by the application.)

int **nor_FlashInit (id)**
Initializes the NOR flash driver internal data structures and the flash chip hardware. Also does power up check and does the recovery, if necessary.

int **nor_FlashRelease (id)**
Releases the NOR flash driver when the file system un-mounts this disk.

int **nor_BlockReclaim (id, eesn)**
Does a block reclaim. Should be called only when the system is idle. smxFS calls this function through a custom sfs_ioctl() function.

int **nor_SectorRead (id, rp, si)**
Reads one data sector from the flash.

int **nor_SectorWrite (id, wp, si)**
Writes one sector of data to the flash.

int **nor_SectorDiscard (id, si)**
Discards one sector. Used when deleting a file.

u32 **nor_SectorNum (id)**
Returns the number of sectors in the flash partition.

u32 **nor_SectorSize (id)**
Returns the sector size.

u32 **nor_WearLeveling (id)**
Does static wear leveling based on wear counters and a configurable threshold.

id = chip id, si = sector index, rp = read buffer pointer, wp = write buffer pointer,
eesn = expected empty sector number.

Operation

The high-level driver performs the following functions:

- Converts each logical sector address to a NOR memory physical sector address.
- Sector map caching.
- Power fail-safe sector writes.
- Power failure recovery on power up.
- Block reclaim.

- Static and dynamic wear leveling.
- Read back verification.

Sector Translation. The main function of the high-level driver is to convert a logical sector address into a physical sector address. To do so, smxNOR utilizes and maintains a *sector map*. The sector map also contains status information for each physical sector that indicates if it is valid, erased, etc. For 512 byte sectors, the maximum memory size is 8 GB. For sizes larger than this, smxNAND is recommended.

Sector Map Cache. The sector map is stored in flash so that it is preserved if power is lost. However, it is not stored in a form convenient for normal operation. Hence, a sector map cache is maintained in RAM to improve performance. An algorithm has been chosen to achieve the best performance under normal conditions. The cache size is determined at compile time and can be adjusted depending upon the NOR flash partition size. Default size is 16 entries.

Power Fail-Safe Write Operation. A pointer is maintained to the next free sector. Before starting to write, the free sector status is set to *being written*. Then data is written into the sector. After the sector write is complete, the status of the old sector, if any, is set to *discarded*, and the status of the new sector is set to *valid*.

Power Failure Recovery. On power up the entire flash partition is scanned for a sector with *being written* status. If found then the flash partition is scanned to find the old sector having the same logical address. If the old sector status is *valid*, then the new sector is deemed to be only partially written and it is discarded. But if the old sector status is *discarded*, then the new sector status is changed to *valid*.

Block Reclaim. Block reclaim is the process of erasing blocks so that they may be reused. It is invoked under two conditions: (1) by the user when the system is idle, or (2) after a sector write when the number of free sectors drops below a pre-defined level. To do a block reclaim the entire sector map is scanned to find the block with the most discarded sectors. Then valid sectors are moved from that block to another block and the block is erased. During this time, the NOR flash driver is busy and will not accept read or write operations. To minimize downtime, only one flash block is reclaimed, at a time.

Static and Dynamic Wear Leveling. Static wear leveling is done by the API function nor_WearLeveling(), which uses wear counters and a configurable threshold. Dynamic wear leveling is a natural result of always moving pointers through flash memory in an increasing direction and recycling from the bottom when the top of the flash partition reached.

Read Back Verification. This option may be enabled at compile time to guarantee that data has been written to NOR flash correctly. If an error is found, the sector is marked *discarded* and the data is rewritten to the next free sector.

Hardware Interface Layer (HIL)

The low-level driver is referred to as the Hardware Interface Layer (HIL) because it creates a hardware-independent abstraction layer for the high-level driver.

API

int **nor_IO_FlashInit (id, pdi)**
Initializes all necessary flash chip hardware settings, such as chip selects. Also loads the fields of the device info structure so that the high-level driver knows the basic flash information.

int **nor_IO_FlashRelease (id)**
Does any clean up work when the NOR flash driver is no longer needed.

int **nor_IO_PageRead (id, rp, si)**
Reads one sector of data from sector si of the flash to the RAM buffer at rp.

int **nor_IO_PageWrite (id, wp, si)**
Writes one sector of data from the RAM buffer at wp to sector si of the flash. The target sector is guaranteed to be empty by the high-level driver algorithm.

int **nor_IO_BlockErase (id, bi)**
Erases the selected block – i.e. changes all bytes in the block to 0xFF.

int **nor_IO_InfoWrite (id, pdi, ds, bi, os)**
Writes specified information into the selected block's first sector.

int **nor_IO_InfoRead (id, pdi, ds, bi, os)**
Reads information from the specified block's first sector.

id = chip id, bi = block index, ds = data size, os = offset, pdi = device info pointer, si = sector index, rp = read pointer, wp = write pointer.

Operation

The HIL performs the following functions:

- Tells the high-level driver the total number of blocks and the block size through a pre-defined device info structure so that it does not need to know the details of each NOR flash chip.
- Blocks until the current operation is completed.
- Handles sector requests that differ from true physical sector size. For example, if a NOR chip allows reading only 256 bytes, at a time, then HIL translates a 512-byte sector read operation into two successive 256-byte read operations.
- Handles different interface widths, transparently, such as 16-bit, 8-bit, and serial.
- Handles different interface types, transparently, such as processor bus, GPIO, and SPI.

Available Drivers

The following low-level drivers (HILs) are currently available:

- Common Flash Interface (CFI)
- Atmel AT45D Data Flash
- Intel 28F Strata Flash
- Spansion S29AL
- SST 39VF
- STMicro M25P

More drivers are being added, so contact us if you need a different driver. In most cases, one of the above will suffice with small modifications. (Small modifications are usually necessary, anyway, due differences in hardware implementations.)

Size and Performance

The following is the smxNOR code size. The STMicro flash is a serial flash. The Intel flash is a parallel bus flash.

ARM7/9 IAR STMicro M25P16	ColdFire CodeWarrior Intel 28F128K3
5.0 KB	7.5 KB

smxNOR needs less than 1KB RAM for buffer and stack.

The following table shows performance for the smxNOR flash driver. The tests wrote 1MB files and were done on a Freescale M5485EVB, using Intel 28F128K3 StrataFlash, and on an Avnet M5282 board using an SPI interface to an STMicro M25P16 serial flash. These measurements were made with smxFS

Flash Type	Read KB/sec	Write KB/sec
Intel 28F128K3 StrataFlash	950	110
STMicro M25P16 serial flash	150	30