NAND FLASH

NAND vs. NOR

Beside the different silicon cell design, the most important difference between NAND and NOR Flash is the bus interface. NOR Flash is connected to a address / data bus direct like other memory devices as SRAM etc. NAND Flash uses a multiplexed I/O Interface with some additional control pins. NAND flash is a sequential access device appropriate for mass storage applications, while NOR flash is a random access device appropriate for code storage application.NOR Flash can be used for code storage and code execution. Code stored on NAND Flash can't be executed frome there. It must be loaded into RAM memory and executed from there.

	NOR	**NAND**
Interface	Bus	I/O
Cell Size	Large	Small
Cell Cost	High	Low
Read Time	Fast	Slow
Program Time single Byte	Fast	Slow
Program Time multi Byte	Slow	Fast
Erase Time	Slow	Fast
Power consumption	High	Low, but requires additional RAM
Can execute code	Yes	No, but newer chips can execute a small loader out of the first page
Bit twiddling	nearly unrestricted	1-3 times, also known as "partial page program restriction"
Bad blocks at ship time	No	Allowed

Some facts about write speed.
NAND is typically faster than NOR for large writes. A typical NOR write is 10uS per word, which results in 1280uS per 512 bytes on a 32-bit bus. A typical NAND write is 50nS per byte + 10uS page seek + 200uS program which results in 236uS per 512 bytes on a 8 bit bus.

As NAND Flash is cheaper than NOR Flash and has a very slim interface it was selected as the optimum solution for large nonvolatile storage applications such as solid state file storage, digital audio/voice recorder, digital still camera and portable applications requiring non-volatility.

NAND Types

There are various types of NAND Flash available. Bare NAND chips, SmartMediaCards, DiskOnChip.

SmartMediaCards are bare NAND chips covered by thin plastic. They are very common in digital cameras and MP3 players. The card itself contains nothing smart at all. It gets smart by software.

DiskOnChip is NAND Flash with additional glue logic as a drop in replacement for NOR Flash chips. The glue logic provides direct memory access to a small address window, which contains a boot loader stub, which loads the real boot code from the NAND device. The logic contains also control registers for the static NAND chip control lines and a hardware ECC generator.

NAND technical view

The memory is arranged as an array of pages. A page consists of 256 / 512 Byte data and 8 / 16 Byte spare (out of band) area. Newer chips have 2048 Bytes data and and 64 Bytes spare area sizes. The spare area is used to store ECC (error correction code), bad block information and filesystem dependend data. n pages build one block. The read / write access to data is on a per page basis. Erase is done on a per block basis. The commands to read / write / erase the chip is given by writing to the chip with the Command Latch Enable pin high. Address is given by writing with the Address Latch Enable pin high.

There are only a few lines neccecary to access NAND Flashmemory.

16 bit buswidth chips are supported.

Pin(s)	Function
I/O 0-7(15)	Data Inputs/Outputs
/CE	Chip Enable
CLE	Command Latch Enable
ALE	Address Latch Enable
/RE	Read Enable
/WE	Write Enable
/WP	Write Protect
/SE	Spare area Enable
R/B	Ready / Busy Output

As it is neccecary to use the spare area, the /SE (Spare area Enable) pin should be tied to GND. /CE, CLE and ALE should be GPIO pins or latched signals. It's possible to use address lines for ALE and CLE, but you have to take care about the timing restrictions of the chip !

/RE and /WE can be tied to the corresponding lines of the CPU. Make sure, that they are logicaly combined with the corresponding chipselect. You can also use two different chipselects for /RE and /WE, but be aware of data hold time constraints of your NAND chip. Data hold time after rising edge of /WE is different to data hold time after rising edge of chipselect lines!

I/O 0-7(15) are connected to the databus D0-D7(D15). The /WP pin can be used for write protection or connected to VCC to enable writes unconditionally. As NAND flash uses a command driven programming and erasing, an accidential write or erase is not likely to happen. The Ready / Busy output is not neccecary for operation, but it can be tied to a GPIO or an interrupt line.

Filesystems supporting NAND

One major problem for using NAND Flash is, that you cannot write as often as you want to a page. The consecutive writes to a page, before erasing it again, are restricted to 1-3 writes, depending on the manufacturers specifications. This applies similar to the spare area. This makes it neccecary for the filesystem to handle a writebuffer, which contains data, that is less than a page

At the moment there are only a few filesystems, which support NAND

- JFFS2 and YAFFS for bare NAND Flash and SmartMediaCards
- NTFL for DiskOnChip devices
- TRUEFFS from M-Systems for DiskOnChip devices
- SmartMedia DOS-FAT as defined by the SSFDC Forum

JFFS2 and NTFL are Open Source, while TRUEFFS is a proprietary solution. SmartMedia DOS-Fat is a specification from SSFDC forum. It is somewhat open under a non disclosure agreement with Toshiba, who owns all rights on this specifications. NTFL is designed for the usage of DiskOnChip devices. JFFS2 supports raw NAND chips and SmartMediaCards at the moment. A JFFS2 support for DiskOnChip devices, based on the NAND code, is planned. There are some other Open Source projects for NAND filesystem support, but there's no other working solution than JFFS2 and YAFFS at the moment of this writing. YAFFS is available from YAFFS-Homepage. YAFFS is faster than JFFS2 and consumes less RAM, JFFS2 provides on the fly file compression and decompression, which is very helpfull for small FLASHs.

There is currently no support for the wide spread SmartMedia DOS-FAT filesystem, mainly because it's not a reliable filesystem for industrial usage. It's ok for multimedia applications. The hardware support layer is designed to support an implementation of SmartMedia DOS-FAT. There are some efforts to implement it, but it's in an early stage. There are a couple of SmartMedia Card adaptors for USB, PCMCIA, FireWire ... with Linux drivers available, which support the SmartMedia DOS-FAT.

JFFS2 and YAFFS include bad block management, wear leveling, error correction and provide reliable filesystems for industrial use on top of NAND Flash.

JFFS2 specific information

JFFS2 Out of Band usage

JFFS2 uses the default autoplacement scheme. The only JFFS2 specific usage of the oob area is the storage of the cleanmarker

Nand chips with 256 byte pagesize and 8 byte OOB size

Offset	Content	Comment
0x06	Clean marker byte 0	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x85. In the remaining pages this byte is reserved
0x07	Clean marker byte 1	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x19. In the remaining pages this byte is reserved

Nand chips with 512 byte pagesize and 16 byte OOB size

Offset	Content	Comment
0x08	Clean marker byte 0	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x85. In the remaining pages this byte is reserved
0x09	Clean marker byte 1	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x19. In the remaining pages this byte is reserved
0x0a	Clean marker byte 2	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x03. In the remaining pages this byte is reserved
0x0b	Clean marker byte 3	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x20. In the remaining pages this byte is reserved
0x0c	Clean marker byte 4	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x08. In the remaining pages this byte is reserved
0x0d	Clean marker byte 5	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x00. In the remaining pages this byte is reserved
0x0e	Clean marker byte 6	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x00. In the remaining pages this byte is reserved
0x0f	Clean marker byte 7	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x00. In the remaining pages this byte is reserved

Nand chips with 2048 byte pagesize and 64 byte OOB size

Offset	Content	Comment
0x10	Clean marker byte 0	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x85. In the remaining pages this byte is reserved
0x11	Clean marker byte 1	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x19. In the remaining pages this byte is reserved
0x12	Clean marker byte 2	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x03. In the remaining pages this byte is reserved
0x13	Clean marker	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x20. In the remaining pages

	byte 3	this byte is reserved
0x14	Clean marker byte 4	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x08. In the remaining pages this byte is reserved
0x15	Clean marker byte 5	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x00. In the remaining pages this byte is reserved
0x16	Clean marker byte 6	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x00. In the remaining pages this byte is reserved
0x17	Clean marker byte 7	This byte indicates that a block was erased under JFFS2 control. If the page was succesfully erased this byte in the first page of a block is programmed to 0x00. In the remaining pages this byte is reserved

HOWTO implement NAND support

Where can you get the code ?

The latest changes to JFFS2 and the underlying NAND code are not in the kernel code at the moment. The latest code is available from CVS and daily snapshots

There are four layers of software

1. JFFS2: filesystem driver
2. MTD: Memory Technology Devices driver
3. NAND: generic NAND driver
4. Hardware specific driver

the MTD driver just provides a mount point for JFFS2. The generic NAND driver provides all functions, which are neccecary to identify, read, write and erase NAND Flash. The hardware dependend functions are provided by the hardware driver. They provide mainly the hardware access informations and functions for the generic NAND driver. For YAFFS applies the same.

API Documentation

A complete API documentation is available as DocBook template in the Documentation/DocBook directory of the MTD source tree.

Read the API documentation online

Supported chips

Most NAND chips actually available should be supported by the current code. If you have a chip, which is not supported, you can easily add it by extending the chiplist in drivers/mtd/nand/nand_ids.c. The chip name does not longer contain cryptic part numbers, as the device ID is just an information about size, erase block size, pagesize and operating voltage. Add an entry, which contains following information:
{ name, id, pagesize, chipsize, erasesize, options }

ref	**comment**
name	string: "NAND 'size' 'voltage' 'bus-width'"
id	chip device code. This code is read during nand_scan. Check datasheet for the code of your chip
pagesize	Page size (0,256,512). 0 indicates that the pagesize can be read out from the chip in the extended ID
chipsize	The total size of the chip in MiB
erasesize	the erasesize of your chip in bytes. 0 for chips with extended ID
options	Options. Bitfield to enable chip specific options. See nand.h

Please contact NAND driver maintainer to include it in the public source tree.

Manufacturer codes are scanned during nand_scan too. If the code is one of the known codes in the manufacturer ID table, the name of the manufacturer is printed out, else "Unknown" is printed. This happens

when your hardware driver is loaded and calls nand_scan. Add codes, which are new and contact NAND driver maintainer to include it

Config settings

The following config switches have to be set. JFFS2 on NAND **does not** work, if one of these settings is missing.

CONFIG_MTD=y
CONFIG_MTD_PARTITIONS=y
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLOCK=y
CONFIG_MTD_NAND=y
CONFIG_MTD_NAND_YOURBOARD=y
CONFIG_JFFS2_FS=y
CONFIG_JFFS2_FS_DEBUG=0
CONFIG_JFFS2_FS_NAND=y

Make sure that fs/Config.in contains the following lines:
dep_tristate 'Journalling Flash File System v2 (JFFS2) support' CONFIG_JFFS2_FS $CONFIG_MTD
if ["$CONFIG_JFFS2_FS" = "y" -o "$CONFIG_JFFS2_FS" = "m"] ; then
int 'JFFS2 debugging verbosity (0 = quiet, 2 = noisy)' CONFIG_JFFS2_FS_DEBUG 0
bool 'JFFS2 support for NAND chips' CONFIG_JFFS2_FS_NAND
fi

FAQ

Please see the NAND section in MTD FAQ's

References:

Open Source

JFFS2 and NTFL are located on this website.
YAFFS is located at YAFFS-Homepage.

Hardware

Toshiba

Samsung

SSFDC Forum

M-Systems

Maintainers

JFFS2 is maintained by David Woodhouse

The generic NAND driver is maintained by Thomas Gleixner

Please don't contact them direct. Ask your questions on the mtd-mailing-list.

Any suggestions, improvements, bug-reports and bug-fixes are welcome

---

Thomas Gleixner
$Id: nand.html,v 1.1 2005/03/12 13:43:49 gleixner Exp $