

# Linux in embedded systems

Peter De Schrijver (p2@mind.be)  
Mind Linux Solutions

12th November 2001

## 1 abstract

Using Linux in an environment which has a limited amount of RAM and boots or runs from flash, requires some special care in setting up the system. There are a number of approaches varying from running everything from RAM using a ramdisk to using execute in place and a flash filesystem to run as much as possible from flash. The appropriate choice is based on the amount of RAM you have, how much writes are necessary, what kind of persistency the application needs,...

## 2 Overview of common types of Flash memory

A flash chip consists of a number of erase blocks. Each block can be erased individually. The erase operation will reset the contents of the block to all 1's. Writing can change individual bits back to 0.

### 2.1 NOR Flash

NOR flash is the most expensive flash variant. It is however directly accessible as memory by the CPU so it can easily be used as the primary bootstrap and to directly execute code from (XIP). It doesn't scale as well as NAND and erase blocksize are typically quite large (64Kbyte). NOR flash chips also don't have problems with bad blocks.

### 2.2 NAND Flash

NAND flash has some interesting advantages. It's less expensive than NOR flash. NAND flash is about 3 times the price of SDRAM whereas NOR flash is 10 times the price of SDRAM. It's also scales better than NOR flash. You can put from 16Mbit to 1Gbit on the same board space with the same pin layouts. NAND flash also has smaller erase block sizes than NOR (16Kbyte versus upto 64Kbyte). NAND flash has a removable media variant called SmartMedia. NAND flash however is not directly accessible as memory by the CPU. It acts

| client           | NOR flash | NAND flash |
|------------------|-----------|------------|
| character device | yes       | yes        |
| block device     | yes       | yes        |
| NFTL             | no        | yes        |
| FTL              | yes       | no         |
| JFFS             | yes       | yes        |
| JFFS2            | yes       | no         |

Table 1: MTD client support

more like a disk device. It is always accessed in blocks of 512bytes. To read a page, you first have to write the block address into an address register. Then you have to issue a read page command which will instruct the chip to copy the desired page from the flash storage to an onchip buffer. Then you can copy this buffer to main memory. Writing a page is similar, except that you first have to copy the data from main memory to the onchip buffer and then issue the write page command. This means it cannot be used as the primary bootstrap method and it cannot be used for XIP. NAND flash chips can also develop badblocks over time. This means software has to verify writes and save data to spare blocks in case of a verify failure.

### 2.3 Compact flash

Compact flash is accessed via an IDE-like interface. There is no way to directly access the flash chip(s) inside. Therefore from a linux point of way it is a disk, which happens to be implemented using chips instead of platters.

## 3 Linux support for flash storage : MTD

MTD is the memory technology devices framework which has been introduced in the 2.4 kernel to support directly accessible flash storage. See figure 1 for an overview of the architecture. The framework supports drivers for different kinds of NOR and NAND flash chips and also main memory for testing purposes. On top of the MTD layer various modules export services towards the rest of the kernel and userland applications. There is a module which offers a character device for raw flash access. There is a block device module which allows the use of block based filesystems on the flash. There is a NFTL module which implements the NAND Flash Translation Layer and there is a FTL which implements the Flash Translation Layer. These two layers export a block device interface to the rest of the kernel. There is also a Journaling Flash Filesystem which directly uses the MTD interface. Table 1 illustrates client support for specific flash types.

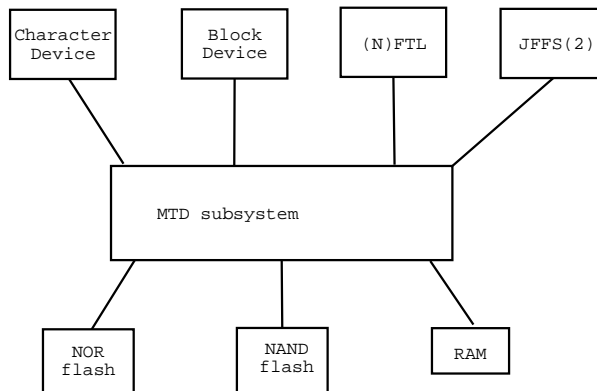


Figure 1: MTD architecture overview

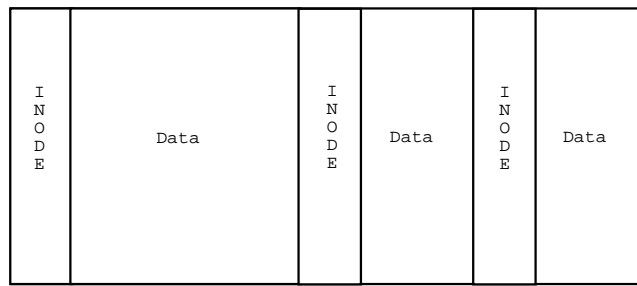


Figure 2: JFFS

## 4 Flash storage organization

### 4.1 No filesystem

If your system does very little writes to stable storage, it might make sense to write the data raw on the flash, without using a filesystem. This saves overhead and makes the bootloader simpler. On the other hand you lose a lot of flexibility for upgrading the system or writing data to stable storage.

### 4.2 Journaling flash filesystems

JFFS is a journaling flash filesystem which has been designed for use on flash devices. JFFS2 is the second, improved version of it. JFFS is a log structured filesystem. A log structured filesystem views the storage medium as a big circular buffer. Filesystem data and meta data are written in the log consecutively. (see figure 2). From time to time a garbage collection process will look for deleted or overwritten data and reclaim their space in the log by compact-

ing the log. This has the advantage of being resilient against powerfailure and providing a crude form of wearleveling. The main disadvantage is the mount time. At mount time an in-memory data structure has to be built containing filesystem metadata. Improvements in JFFS2 include file compression and better wearleveling. NAND flash is supported by JFFS now. JFFS2 does not yet support NAND flash however.

## 5 optimization

The criteria which we will try to optimize are :

- cost
- boottime
- capacity
- read performance
- write performance

### 5.1 cost

#### 5.1.1 Compression

To reduce the cost of flash memory, it can be useful to compress the data stored in it. The main disadvantages are extra CPU load, and read/write performance. If the majority of the files are read only, a ramdisk can be used to store the read only data. This ramdisk can be set up from a compressed image stored in the flash. Data that needs write access can be stored on a separate part of the flash in uncompressed form. The ramdisk can also be used to store data which does not need to survive powercycles (eg. /tmp) JFFS2 supports transparent block level compression. This can be useful if there is not enough RAM available to store the files in uncompressed form. However, the compression will impact all read accesses, while the compressed ramdisk approach only impacts the boottime.

#### 5.1.2 NAND flash

As NAND flash is only 3 times the price of SDRAM (versus NOR flash's 10 times), the use of NAND flash to store data instead of NOR flash could be a good idea. As you can not bootstrap from NAND flash, you still need some NOR flash. Also NAND flash needs a more complicated filesystem or translation layer to handle bad blocks.

## 5.2 boottime

The fastest boottimes are obtained using XIP as the code doesn't have to be copied or decompressed into RAM first. On the other hand execution times are longer when running from NOR flash compared to running from RAM. How much longer depends on the bus width, cache size, flash chip speed, etc. Measurements on a 386EX based system equipped with intel Strataflash indicate a 20% slowdown when running from NOR flash compared to running from RAM. XIP requires the code to reside uncompressed in expensive NOR flash too.

## 5.3 capacity

NAND flash has the highest capacities in general. Compression can also be used to increase capacity.

## 5.4 read performance

NOR flash clearly has a read performance lead compared to NAND flash. Copying a page from the NAND flash storage to the onchip buffer takes about  $2\mu\text{s}$ . Every read cycle to the onchip buffer takes another 50ns. So copying 1 page from NAND flash to main memory takes  $36\mu\text{s}$  on a NAND flash device with an 8bit databus. On Intel's Synchronous Strataflash, which has a 16bit databus, it takes about  $3\mu\text{s}$ . This assumes the use of the synchronous burst mode.

## 5.5 write performance

Looking to write performance, however, NAND flash is the fastest. Writing 512 bytes on an Intel Synchronous Strataflash takes about  $54\mu\text{s}$ , while the same operation a NAND flash chip takes  $26\mu\text{s}$ . Erasing a 16Kbyte block on a NAND flash chip typically takes 2ms, while erasing a 64Kbyte block on an Intel Synchronous Strataflash, takes about 1000ms.

## 5.6 Conclusions

- If you have enough RAM and few writes (e.g. only configuration updates), use a compressed RAM disk image.
- If boottime is very important and you're willing to sacrifice execution speed, you could consider using XIP.
- If you need a lot of storage, use NAND flash.
- If write performance or cost are important, use NAND flash. Unfortunately MTD support for NAND flash is not yet as mature as NOR flash support.
- If you don't need much storage, use NOR flash. NOR flash support is currently better than NAND flash support and you can use part of the NOR flash for the bootcode.

## 6 References

Linux MTD <http://www.linux-mtd.infradead.org/>  
Intel Strataflash <http://developer.intel.com/>  
Samsung NAND flash <http://samsungelectronics.com/semiconductors/Flash/Flash.htm>