Building Embedded Linux Systems using PCM7230
Textbook

“Building Embedded Linux Systems”

- Author: Karim Yaghmour
- Publisher: O’Reilly & Associates, Inc.
- Date: April 2003
- Version: 1st Edition

Professor Advise:

- It is **required** to have this book at hand if you want to become an embedded Linux developer
Overview

- Much of currently available doc relies heavily on a number of prepackaged, ready-to-use cross-platform tools, and binaries.

This book makes no such assumption
  - Only required: an Internet connection
  - A development host (e.g. your PC)
  - A target hardware (e.g. Intel Xscale)
  - Docs on your target HW
Demonstrated Hardware & Required Software
PCM7230
PCM7230: CPU:Xscale PXA255
PCM7230: 64MB SDRAM + 32MB Flash
PCM7230: JTAG/MULTI-ICE
PCM7230: CF Card Slot
PCM7230: PCMCIA slot
PCM7230: CRT cable
PCM7230: *Reset Buttons*
PCM7230: Serial Port
PCM7230: USB ports
PCM7230: Power Switch
PCM7230: Power Supply Cable
Software Version

- **Development host:**
  - PC or notebook
  - Windows XP + VMWare + Red Hat Linux
  - I was using RedHat Linux 9 / 2.4.20.8

- **Target embedded Linux system**
  - Linux 2.4.18 kernel
  - Over 40 different open source and free software packages
    - Each maintained independently at different places
Resources (Appendix B)
Resource Listing

- **This book web site:**
  - Updates: [http://www.embeddedtux.org/](http://www.embeddedtux.org/)

- **Online**
  - LinuxAutomation
    - A well-organized of links to various resources
  - LinuxDevices.com
    - [http://www.linuxdevices.com/](http://www.linuxdevices.com/)
    - The most visible embedded Linux site around
  - Linux Documentation Project
    - [http://www.tldp.org/](http://www.tldp.org/)
    - The main repository for HOWTOs, FAQs, and others
  - uCdot
    - A news and community site for uClinux users
Reference Books

- **The Art of Designing Embedded Systems**
  - By Jack Ganssle (Newnes Press)
  - Experiences from system designers

- **Embedded System Design**
  - By Arnold S. Berger (CMP Books)
  - From both HW and SW perspective

- **Linux Device Drivers**
  - By A. Rubini and J. Corbet (O’Reilly)
  - A must read for any Linux device driver developer

- **Running Linux**
  - By Matt Welsh, et al. (O’Reilly)
  - All you need to know in using Linux, very handy book

- **Understanding the Linux Kernel**
  - By Daniel Bovet and Marco Cesati (O’Reilly)
  - Well researched and structured book on Linux kernel
Publications & Organizations & HW Projects

- **Embedded Systems Programming**
  - [http://www.embedded.com/mag.html](http://www.embedded.com/mag.html)
  - Main magazine for ESW programmers

  - Most well-established Linux publication

  - Another well-established Linux publication


- **Free Software Foundation ([http://www.fsf.org/](http://www.fsf.org/))**


- **Simputer ([http://www.simputer.org/](http://www.simputer.org/))**
  - Developing an inexpensive reference HW running Linux

- **uClinux boards ([http://www.uclinux.org/](http://www.uclinux.org/))**
  - HW projects aimed at building ES running Linux
  - MMM-less port of Linux comes from this project
Term Definition
Linux Nutshell

- Public release: 1991
  - Initially for workstations, servers, and clusters
  - Contribute to the open-source movement

- Nowadays:
  - Linux becomes the preferred OS for embedded devices
  - Flexibility, robustness, price tag, large developer community, vendor support
Linux Definition (1/3)

- Correct pronunciation of Linux
  “Hello, this is Linux Torvalds, and I pronounce Linux as Linux!”

- This message was recorded by Linus
  ➢ Should be considered as a right and correct pronunciation
Term Definition (2/3)

- **Linux kernel:**
  - “Linux” itself refers to the kernel maintained by Linus Torvalds
  - Kernel only, no utilities (e.g. X Windows)
  - The kernel controls all HW and provides processes, sockets, and files, etc
  - **2.4.18:** version 2.4, release 18
    - Odd version: development kernels
    - Even version: stable kernels
  - **2.4.18-rmk3-hh24**
    - A modified kernel by Familiar project with private version “rmk3-hh24”
Term Definition (3/3)

- **Linux system:**
  - A HW system running the Linux kernel and various utilities
  - Utilities: GNU C library, X windows, RTAI real-time addition, etc

- **Linux distribution**
  - A set of files and an automatic procedure to install Linux and utilities on your system
  - Red Hat, Mandrake, SuSE, Debian, Slackware, Caldera, MontaVista, Embedix, BlueCat, PeeWeeLinux, etc
Embedded Linux?

- **Embedded Linux kernel:**
  - No such thing distributed by Linus
  - It means you do not need a special version of kernel for your embedded systems
  - Often, a modified kernel is distributed by a 3rd party or vendor
  - Example: **S3C2410 Linux kernel** =
    - 2.4.18 Linux kernel +
    - Open-source ARM patch +
    - Samsung special patches (2)
Embedded Linux Distribution

- Linux kernel and patches
- Development framework for embedded Linux systems
  - Installed at the development host
  - Source browsers
  - Cross-compilers
  - Debuggers
  - Boot image builders
Real-Time Linux

- Initially, it means the RTLlinux project released in 1996
- Today, there are many more similar projects
  - RTAI, Kurt, Linux/RK, etc
- Most of them are soft-real-time
16-bit Linux?

- In general, Linux runs on processors with 32 bits or more
- ELKS: Embedded Linux Kernel Subset project
  - Intends to run Linux on 16-bit processors such as Intel 8086 and 286
Market Shares
ELS Example: Ericsson “blip”

- **Ericsson “blip” project**
  - A commercial product that discontinues already
  - Bluetooth Local Infotainment Point
  - A tiny networked computer that communicates with any Bluetooth device
    - PDA, phones, etc

- **Linux system**
  - uClinux distribution
  - 2.0.38 for MMU-less ARM
  - uClibc
ELS Example: Sharp Zaurus

- Zaurus is a PDA completely based on Linux
- SL-5500: Intel StrongARM 206MHz
  - 64MB RAM, 16MB flash
- SL-5600: Intel XScale 400 MHz
  - 32MB RAM, 64MB flash
  - Based on Lineo’s Embedix embedded Linux distribution
- Developer web site:
  - http://developer.sharpsec.com/
Market Surveys (1/2)

- **2000 survey by Embedded Systems Programming magazine**
  - 38% readers considering using Linux in their ES
  - 12% already used Linux in their ES

- **2000 survey by LinuxDevices.com**
  - ELS platforms: x86, ARM, PPC
  - Main distributions: Red Hat, Debian, MontaVista
  - Resources: more than 8MB RAM and 8MB persistent storage
Market Survey (2/2)

- **Survey by Venture Development Corp (VDC)**
  - Leading vendors: Lineo, MontaVista, Red Hat

- **Summary:**
  - Linux is getting popular in embedded systems
  - Linux is expected to be used as Windows in ES
  - Debian is one of favorite embedded distributions
Reasons for Favoring Linux (1/2)

- Quality and reliability of code
  - Open source development model
  - A good book to read: “The Cathedral and the Bazaar” by O’Reilly
- Availability of code
  - GNU and FSF are big contributors
- Hardware support
  - Many drivers are maintained by the Linux community itself
- Communication protocol and SW standards
  - Make it easier for a Linux system to work with others
Reasons for Favoring Linux (2/2)

- **Available tools**
  - Freshmeat ([http://www.freshmeat.net](http://www.freshmeat.net))
  - SourceForge ([http://www.sourceforge.net](http://www.sourceforge.net))

- **Community support**
  - An email to the appropriate mailing list to point you to the person who wrote the code

- **Licensing**
  - GPL will be described later

- **Cost**
  - Probably only for purchasing packaged distribution
Major Industrial Players

- **Red Hat**
  - One of first Linux distributions
  - Provide a development package for developing embedded Linux systems
  - [http://sources.redhat.com/](http://sources.redhat.com/)

- **MontaVista**
  - Founded by Jim Ready
  - Leaders in services and porting

- **LynuxWorks**
  - Used to be known as Lynx Real-Time Systems
  - Lynx: continues to be a hard-real-time OS
  - BlueCat: their embedded Linux distribution
Copyright and Patent Issues
GPL and LGPL License

- **GPL** is mainly used for applications and **LGPL** is mainly used for libraries
  - E.g. kernel, binary utilities, gcc compiler, gdb debugger are under GPL
  - C library is under LGPL
- **What is GPL license?**
  - Free to make copies
  - It provides no warranty whatsoever
  - Binary must be distributed along with source code
  - No further restriction can be placed on GPL code
  - Any code that modified or includes GLP code must be also distributed under GPL
    - This is so called virus-like license
  - Packaging unmodified GPL code for just running it is okay
More on GPL License

- **Running v.s. modifying GPL SW**
  - Software compiled by unmodified gcc is not covered by GPL
  - SW compiled by modified gcc is under GPL
  - A new compiler generated by modified gcc is under GPL
  - Any program compiled by this new compiler is also under GPL

- **Static & dynamic linking**
  - Your app that dynamically (or statically) links to GPL SW is still under GPL
  - E.g. make gcc as a dynamic linked library and your new compiler links to it ➔ your new compiler is still under GPL

- **A derived work combining GPL code and non-GPL code is still under GPL license**
LGPL License

- LGPL allows you to use *unmodified* LGPL code in your program → not under LGPL
  - But if you use modified LGPL code, the new program is then under LGPL
- GNU C library is under LGPL
  - Any code that links unmodified C library is not under LGPL
Complications & Confusion

- How about a kernel module running at Linux kernel? It is integrated into kernel space?
- How about user programs running on Linux system?
- How about binary-only modules?
- Well, it is too complicated for us (you and me)
  - If you plan to make money out of open-source code, I strongly suggest to hire a good lawyer to clear up all these issues
Building Your Own Embedded Linux System
Using Distributions (1/2)

- First, it is not necessary to use any form of distribution to build an ELS
  - E.g. We rely on no distribution for S3C2410
- But, using a prepackaged distribution will save you a lot of time 😊
  - Finding matched package versions is time-consuming
- Open-source or commercial distribution?
  - Some commercial distributions come with valued-added packages that are not under GPL
    - Distributing them is an illegal act
  - Some sort of support is provided by commercial ones
  - Better documentation support by commercial ones
Avoid depending solely on distribution
- Don’t be locked on a particular distribution
- Why do you use Linux at all?

Documentation issues
- Open-source developers prefer to write more code than documentation
- Linus advise: “Use the source, Luke!”
- How do you learn Nachos? By tracing code
Creating Target Linux System

- There are five main steps to create a target Linux system
  - 1. Determine system components
    - Chapter 3
  - 2. Setup cross-platform development environment
    - Chapter 4
  - 3. Configure and build kernel
    - Chapter 5
  - 4. Build root file system
    - Chapter 6
  - 5. Set up bootloader and configuration
    - Chapters 7, 8, 9
Host / Target Environment

Host:
- Cross-platform development environment

Target:
- Cross-platform development environment
- Bootloader
- Kernel
- Root filesystem

Kernel 透過 TFTP 從 host 取得
root filesystem 透過 NFS 由 host mount 上來
Porting Roadmap

Setup cross-platform development tools
- Setup workspace
- Download toolchain source
- Setup kernel headers
- Setup binary utilities
- Setup bootstrap compiler
- Setup C library
- Setup full compiler

Compile kernel
- Setup root filesystem
  - Create fs hierarchy
  - Install libraries
  - Install kernel modules
  - Install kernel images
  - Create device files
  - Install system utilities

Setup RAM disk
- Setup Bootloader

Module-4: Embedded Linux Introduction
Host/Target Development Setups
Type of Hosts (1/2)

- **Linux workstations**
  - Most common type of hosts for ELS
  - **HW:** standard PC, Mac, etc
  - **Distribution:** Red Hat, Debian, etc
  - Large disk space (>3GB), RAM (128MB RAM)

- **Unix workstations**
  - E.g. Sun Solaris workstations
Type of Hosts (2/2)

- Windows (2000, XP, ME, etc) workstations
  - Red Hat: Cygwin (Windows-compatible GNU toolchain)
  - Linux (virtual) environment on Windows: VMWare or Connectix

- Case study:
  - Mumit Khan detailed the procedure to build a cross-platform dev toolchain for an i386 Linux target
Host/Target Setups

- Three kinds of setups are available for developing ELS
  - The linked setup (S3C2410 case study)
  - The removable storage setup
  - The standalone setup
Linked Setup (1/2)

- Cross-platform development environment
  - Host
  - Target
  - Bootloader
  - Kernel
  - Root filesystem
Linked Setup (2/2)

- Target and host are permanently linked using a physical cable
  - A serial cable or an Ethernet link or an USB cable

- Alternatives
  - Root FS be accessed by NFS-mounted
  - Kernel be accessed by TFTP

- It is common to have another physical link for debugging purpose
  - E.g. Ethernet link for file transfer, serial cable for debugging purpose
Removable Storage Setup (1/2)

Figure 2-2. Host/Target removable storage setup

- **Host**
  - Cross-platform development environment

- **Target**
  - Bootloader
  - Secondary bootloader
  - Kernel
  - Root filesystem
Removable Storage Setup (2/2)

- There is no direct physical link between host and target
- Files are transferred through a storage device such as CompactFlash IDE
- Target contains
  - only a minimal bootloader to load a secondary bootloader from storage device
  - Or exactly nothing
Standalone Setup (1/2)

Figure 2-3. Host/Target standalone setup

<table>
<thead>
<tr>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Bootloader</td>
</tr>
<tr>
<td>*Kernel</td>
</tr>
<tr>
<td>*Full root filesystem</td>
</tr>
<tr>
<td>*Native development environment</td>
</tr>
</tbody>
</table>
Standalone Setup (2/2)

- Target is a self-contained dev system
  - It requires no cross-platform dev toolchain
- Popular with developers building high-end PC-based embedded systems
  - E.g. high-availability systems
- Frequent reboots may happen due to instability of tested kernels
  - Still suggest for a target machine
Host/Target **Debug Setup**

- **Three types of interfaces to link a target to a host for debugging**
  - Serial line / USB line
  - Networking interface
  - Special debugging hardware

- **Serial line**
  - The simplest way to debug target from host
  - Speed is rather limited
  - If there is only one serial line, you cannot do file transfers and debugging at the same time
Networking Debug Interface

- Example: TCP/IP over Ethernet
  - Much higher bandwidth than serial link
  - Many networking connections between target and host can be used at the same time
    - You can debug and interact at the same time
  - But, such an interface cannot be used to debug kernel itself
    - TCP/IP is part of kernel itself
    - Solution: use serial line for debugging and networking for interacting
Special Debugging Hardware

- **The preferred** way to obtain direct control over HW is to use BDM or JTAG interface
  - It is also a common practice to **debug Linux kernel** over a development target board
- **This requires purchase of special HW and SW**
  - Evaluate **its compatibility with GNU toolchain before you purchase them**
  - A good such tool should deal with standard GNU toolchain and generated binary files
Embedded Linux Architecture
Generic Linux System Architecture (1/2)

Figure 2-4. Architecture of a generic Linux system
Generic Linux System Architecture (2/2)

- Linux requires at least 32-bit CPU with MMU
  - uCLinux is Linux for MMU-less architecture
- The kernel requires at least one properly structure file system, the root filesystem
  - Kernel loads the first app from root filesystem
- Root filesystem location
  - Be stored on a real storage device, or
  - Loaded into RAM during system startup, or
  - Mounted through NFS
- Libraries (such C lib) are typically dynamically liked with applications
System Startup

- Three SW modules participate in system startup
  - Bootloader (Chapter 9)
  - Kernel (next semester)
  - Init process (Chapter 6)

- Bootloader
  - The first SW module to run during startup
  - Highly dependent on target HW
  - Conduct low-level HW initialization and jump to kernel’s startup code

- Kernel internal startup and initialization
  - Will be discussed in length next semester
Boot Configuration

- All CPUs fetch their 1st instruction from an address preassigned by their manufacture
  - The address refers to a solid state storage device such as a masked ROM or a flash chip
  - The code at the address is for bootstrapping system

- Workstations & servers: boot software is responsible only for loading OS from disk

- For embedded systems, there are few agreed upon purposes, if any, for boot software
Embedded System Boot Software

- It may be the only SW that runs throughout system life’s time
- It may be a simply monitor that loads rest of system software
- It may even load additional bootloaders, as is often the case with x86 PCs
ELS Bootstrap Setup

- Three different setups for bootstrapping ELS (details in Chapter 9)
  - Solid state storage media setup
  - Disk setup
  - Network setup

- Solid state storage media is initially programmed by JTAG or BDM tool

*Figure 2-5. Typical solid state storage device layout*
Disk Setup & Network Setup

- **Disk setup**
  - Mostly used in workstations and servers
  - Kernel and root FS are located on a disk
  - Initial bootloader at masked ROM either
    - Loads a secondary bootloader off disk, or
    - Fetches kernel itself directly from disk

- **Network setup**
  - Kernel resides on a local solid state storage or a disk and root FS is mounted via NFS
  - Bootloader on local storage, kernel is downloaded via TFTP, root FS is mounted via NFS
Embedded Linux
Hardware Support
Chapter Outline

- This chapter discusses the embedded hardware supported by Linux.
- Use it as a starting point for your research in:
  - Identifying components to be included in your system.
  - Judging the amount of effort needed to get Linux running on such hardware.
- It only covers HW supported by open source and free software communities.
Type of Hardware Discussed

- Processor Architectures
  - x86, ARM, PowerPC, MIPS, Hitachi SuperH, Motorola 68000,

- Buses and Interfaces
  - ISA, PCI, PCMCIA, PC/104, VME, CompactPCI, parallel port, SCSI, USB, IEEE1394, GPIB, I^2C

- I/O
  - Serial port, parallel port, modem, keyboard, mouse, display, sound, printer

- Storage
  - MTD, ATA-ATAPI (IDE), SCSI

- General Purpose Networking
  - Ethernet, IrDA, IEEE 802.11, Bluetooth,
Processor Architectures

- Only MMU-equipped CPUs are discussed
  - MMU-less CPUs: [http://www.uclinux.org/](http://www.uclinux.org/)
    - Motorola MMU-less 68K CPUs, MMU-less ARM, Intel’s i960, etc
- x86: Linux kernel was first written for i386
- ARM
  - All ARM processors share same ARM instruction set
  - Complete and up-to-date list of ARM supported
    - [http://www.arm.linux.org.uk/developer/machines](http://www.arm.linux.org.uk/developer/machines)
  - Many projects have ARM support
    - Hard real-time from RTAI project
  - However, no kernel debugger support for ARM
    - You have to use JTAG debugger
Flash Storage

- All embedded systems require at least one form of persistent storage to start booting an EOS.
- In Linux terminology, MTD (memory technology devices) include all memory devices:
  - ROM, RAM, flash, DiskOnChip (DOC), etc.
- Linux kernel includes MTD subsystem:
  - Provides a unified and uniform layer enabling a seamless combination of low-level MTD drivers.
  - “MTD user modules” refer to SW within kernel that enables access to low-level MTD drivers.
Flash Memory Device

- Low-power, high density non-volatile storage
- Bit resets only in “erase blocks”
- Limited lifetime: typically 100,000 erase cycles
- NOR flash and NAND flash
  - NAND with smaller erase blocks
- Traditional answer (FTL and NFTL)
  - Emulate a standard block device
  - Use a normal file system on top of that
The MTD Subsystem

Kernel virtual filesystem layer

Disk-style filesystem

**MTD “user modules”**

- JFFS2
- JFFS
- Char device
- Block device
- Read-only block device
- NFTL
- FTL

**Memory Technology Devices “glue logic”**

**MTD chip drivers**

- DiskOnChip flash
- JEDEC-compliant flash
- Uncached system RAM
- RAM, ROM, and absent chips
- Virtual memory
- Block device

**Virtual devices for testing and evaluation**

**Memory devices hardware**

Figure 3-1. The MTD subsystem
More on Linux MTD

- **Goal:** make it simple to provide a driver for new hardware
  - By providing a generic interface between the HW driver and the upper layers of the system
  - HW drivers need to know nothing about storage format used
    - e.g. block, char, or JFF2, etc
  - HW drivers only need to provide simple routines for *read*, *write*, and *erase*
MTD Chip Drivers supported by Linux

- DiskOnChip 2000 and Millennium
- Common Flash Interface (CFI)
- PCMCIA flash (not CompactFlash)
- JEDEC
- Non-DOC NAND flash
- Old non-CFI flash
- RAM, ROM, and absent chips
- Uncached RAM
MTD User Modules

- MTD subsystem allows for memory devices to be divided into multiple partitions
  - Each partition can be managed by a different MTD user module
  - Just like your hard disk device
- MTD user modules are not fully interoperable with all MTD drivers
  - E.g., JFFS2 cannot deal with NAND flash chips (but things are changing very fast)
MTD User Module List

- http://www.linux-mtd.infradead.org/
- JFFS2 (Journaling Flash File System V2)
- NFTL (NAND Flash Translation Layer)
- JFFS
- FTL (Flask Translation Layer)
- Char device (Unix-like char device)
- Caching block device (usual workstation and server FS can be used on)
- Real-only block device
Flash-Based File System
FTL – Flash Translation Layer (1/2)

Figure 1. FTL Sector Relocation
FTL – Flash Translation Layer (2/2)

Figure 3. Erase Unit Divided into Read/Write Blocks. Each Read/Write Block Is the Same Size as a Virtual Block (Sector) Used by the Software Layers above FTL.
JFFS2

- Red Hat’s Journalling Flash File System
- It does not provide a translation layer enabling use of a traditional FS
  - It uses a log-structured FS on a MTD device
- It can gracefully deal with power failures
- It provides wear leveling (100,000 times limited) by ensuring uniform usage on all blocks
- It offers compression and decompression between MTD device and RAM
- It has been widely adopted as the FS for MTD devices
JFFS

- Improvement on NFTL
  - A journalling file system specifically for flash devices, with built-in wear leveling
- JFFS2 is a log-structured file system with direct operations on flash devices
- GPL’d code for Linux 2.0
- Data stored on medium in no particular location
- Packets written sequentially to a log which records all changes
Log-Structured File System (1/4)

<table>
<thead>
<tr>
<th>Storage Medium</th>
<th>User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 1</td>
<td>Write 200 bytes ‘A’ at offset zero in file</td>
</tr>
<tr>
<td>offset: 0</td>
<td></td>
</tr>
<tr>
<td>len: 200</td>
<td></td>
</tr>
<tr>
<td>data: AAAAA...</td>
<td></td>
</tr>
<tr>
<td>version: 2</td>
<td>Write 200 bytes ‘B’ at offset 200 in file</td>
</tr>
<tr>
<td>offset: 200</td>
<td></td>
</tr>
<tr>
<td>len: 200</td>
<td></td>
</tr>
<tr>
<td>data:BBBBB...</td>
<td></td>
</tr>
<tr>
<td>version: 3</td>
<td>Write 50 bytes ‘C’ at offset 175</td>
</tr>
<tr>
<td>offset: 175</td>
<td></td>
</tr>
<tr>
<td>len: 50</td>
<td></td>
</tr>
<tr>
<td>data:CCCCC...</td>
<td></td>
</tr>
</tbody>
</table>
Log-Structured File System (2/4)

To read the file system, the log nodes are played back in version order, to recreate a map of where each range of data is located on the physical medium.

<table>
<thead>
<tr>
<th>Node playback</th>
<th>List State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node version 1:</td>
<td>0–200: v1</td>
</tr>
<tr>
<td>200 bytes @ 0</td>
<td>0–200: v1</td>
</tr>
<tr>
<td></td>
<td>200–400: v2</td>
</tr>
<tr>
<td>Node version 2:</td>
<td></td>
</tr>
<tr>
<td>200 bytes @ 200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0–175: v1</td>
</tr>
<tr>
<td></td>
<td>175–225: v3</td>
</tr>
<tr>
<td></td>
<td>225–400: v2</td>
</tr>
<tr>
<td>Node version 3:</td>
<td></td>
</tr>
<tr>
<td>50 bytes @ 175</td>
<td></td>
</tr>
</tbody>
</table>
Log-Structured File System (3/4)

Some nodes are completely obsoleted by later writes to the same location in the file. They create “dirty space” within the file system.
Log-Structured File System (4/4)

So far so good. But soon the log reaches the end of the medium. At this point we need to start to reclaim some of the dirty space.

So we copy the still-valid data from the beginning of the log to the remaining space at the end...

...until we can erase a block at the start.
JFFS vs. JFFS2

- JFFS
  - Poor garbage collection performance
  - No compression

- JFFS2
  - Adding compression
  - Non-sequential log structure
  - Improved memory usage
DiskOnChip

- [http://www.m-sys.com/](http://www.m-sys.com/)
  - uDiskOnChip: USB 2.0, 64MB - 2GB
  - DiskOnChip 2000: DIP, 16MB - 1GB
  - iDiskOnChip: IDE bus, 16MB - 1536MB

Features

- Plug-and-play on major OSes
  - CE, XP, VxWorks, Linux, QNX, DOS
- Full boot capability
- Built-in TrueFFS for full disk emulation
Summary

- Many off-the-shelf HW are supported by Linux
- Many drivers of such HW are maintained by Linux and open-source community
- Incompatibility is an issue that should be considered when selecting HW/SW packages
- This book (BELS) provides a comprehensive list of HW components