FH Technikum Wien

DIPLOMA THESIS

for an academic degree

"Master of Science in Engineering"

An Open Source based Software Stack for Embedded SIP Applications

written by Anton Schmidbauer, BSc.
3500, Krems a.d. Donau, Bahnhofplatz 8/9

1. Examiner: Dipl. Ing. (FH) Matthias Kastner
2. Examiner: Dipl. Ing. Johannes Egger

Vienna, March 4, 2009

FACHHOCHSCHULE TECHNIKUM WIEN

Written at University of Applied Sciences Technikum Vienna
Study Programme Telecommunications & Internet Technologies

Affidavit

„I hereby declare by oath that I have written this paper myself. Any ideas and concepts taken from other sources either directly or indirectly have been referred to as such. The paper has neither in the same nor similar form been handed in to an examination board, nor has it been published. “

Place, Date

Signature

Abstract

Building a software stack for applications running on embedded systems is a complex task. The task is even more complex for newcomers to the embedded system world. Because of differences to standard PC systems many questions arise while working on the project. Within this thesis we propose a software stack that is based on Open Source Software. We introduce the reader to embedded systems in general, the Linux operating system running on our reference platform and we also provide a reference implementation for our proposed stack.

Keywords: embedded system, linux, open embedded, bitbake, open source software

Acknowledgements

I would like to thank Dr. Elke Mayer and Dipl. Ing. Johannes Egger for their never ending patience.

Contents

1 Task and Motivation

The study program Intelligent Transport Systems at the University for Applied Science Technikum Wien has an ongoing project where the Session Initiation Protocol (SIP) event framework is used for car to car communication. One of the challenges within this project was to find a suitable embedded system platform to implement this particular SIP based application. A requirement was to base all applications written within this project on Open Source Software (OSS) . We have a strong interest in OSS, embedded systems and SIP. Because we already had experience with Linux but not as an operating system on embedded platforms, it was interesting to see how a newcomer to the embedded world would handle the upcoming problems.

Our assignment was to create a software stack which enables software developers to write SIP applications for embedded systems. The SIP event framework is the only relevant part of the SIP stack used in this project, however our software stack allows developers to use all available SIP protocol features. The use of Open Source Software where feasible was a project requirement.

We introduce the reader to embedded systems fundamentals and the Linux operating system running on these embedded systems. We cover topics that distinguish embedded systems from standard PC systems, including the boot process, different kinds of non-volatile memory, filesystems and software development.

We also describe the evaluation criteria we used to decide which software tools and hardware components to use for the project[1]. It was not required to implement a SIP client or to demonstrate the usage of the SIP event framework. This thesis does not contain a single line of C source code and is not an introduction on how to develop software using the `PJSIP` library. To understand this thesis a basic understanding of the Linux operating system and computing in general is required.

[1] We include evaluation criteria for the embedded system board, even though we had to use the CompuLab cm-x270 for this project.

2 Introduction

The Session Initiation Protocol (SIP) is becoming more and more important in the Internet world. But Voice over IP applications are just one case where SIP can be used. The university of Applied Science Technikum Wien decided to use the SIP event framework in a car to car communication project. The application implemented within the project should be based on Open Source Software running on an embedded system platform.

One of the tasks within this project was deciding which platform and toolchain to use for application development. Because our team had little experience with embedded systems in general we also needed to provide some evaluation guidelines to select suitable components for such a project. Finding the right tools is often the hardest part when working with a new environment. Developing software for embedded systems is quite different from writing code that is going to run on standard desktop computers. On a standard PC only a compiler and an editor is needed. But things change when faced with an embedded system board that has a different CPU architecture, no harddrive and very limited memory. In the project we had to answer many questions: How to compile code for a CPU architecture that is different from the platform the compiler runs on? How to install the resulting binary application on the destination platform? What kind of operating system is going to run on the embedded board? These are questions we had to answer and this thesis provides some of the answers.

This thesis has two major parts. The first part (chapters 3-6), "Fundamentals", presents the reader with an introduction to the embedded system world, focusing on providing a theoretical background of embedded systems. The second part (chapter 7) "Reference Implementation" describes the practical implementation of our proposed software stack. We provide the reader with a step-by-step guide on how to build the software stack for one particular embedded system. The guide is also applicable to other embedded systems.

Chapter 3 "Fundamentals" provides an overview about embedded systems and Linux as an operating system for embedded platforms. "Embedded System Basics" (section 3.5) provides a definition of the term embedded system and describes major differences between embedded and non-embedded systems. The following section 3.6 on "Linux" covers the operating system used for the project, describes what open source software means and details the boot process. Section 3.7 "Session Initiation Protocol" gives an overview about the protocol.

Chapter 4 "Choosing an Embedded Platform" provides certain rules that should be followed when evaluating embedded platforms. We are also providing an overview of the embedded board used for this project.

The next chapter "Choosing a Toolchain" focuses on finding the right tools for application

development. Having little experience with embedded system software development it was important to find tools that would enable the project team to concentrate on writing the application and not on how to deploy the application and using the toolchain.

Chapter 6 "Choosing a SIP Library" describes the evaluation criteria used for deciding which SIP library should be used for the project. A stable implementation of the SIP protocol stack is crucial. Application development should be focused on the application itself, not on implementation details of the protocol. It is important to understand SIP and the event framework, not how they are implemented in a specific library. Problems with the SIP library could put the project at risk.

In the last chapter 7 "Reference Implementation" we provide a detailed description how the platform, the toolchain and the SIP library were used for this project.

3 Fundamentals

In the following sections we discuss fundamentals necessary to understand this master's thesis. We give an overview of our proposed software stack, cover the basics of Open Source Software, describe embedded system fundamentals as well as cross compilation and embedded system memory. We finish this chapter with an overview of the Linux operating system and special fileystems used in embedded systems.

3.1 Project Background

In this section we explain the intended application for the proposed software stack. Our software stack is not restricted to this particular application but it demonstrates one possible usage.

The plan is to implement a car to car communication system. Figure 3.1 depicts the system:

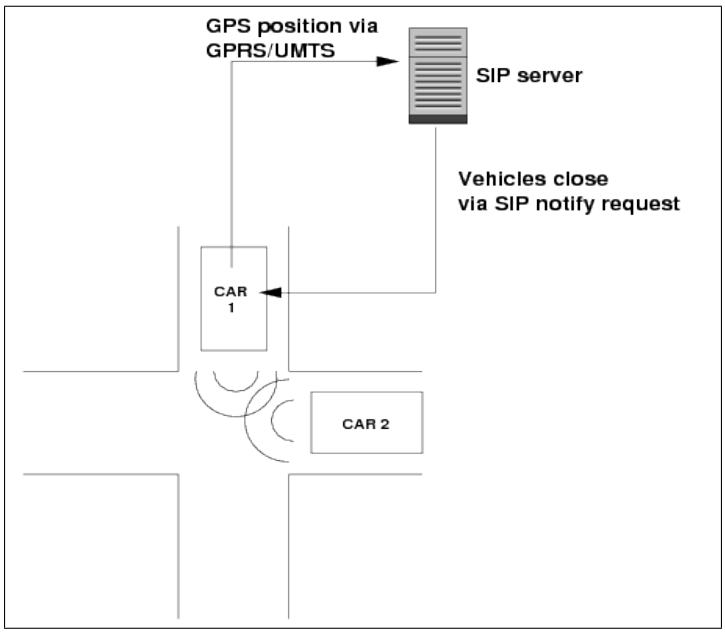

Figure 3.1: Intended Application

Car 1 and Car 2 are equipped with an embedded system that includes a GSM/WLAN module and a GPS receiver. Car 1 and 2 send its GPS location data via a GPRS or

UMTS[1] up-link to a central SIP server. The SIP event framework is used to publish GPS coordinates to the server. The SIP server informs Car 1 and 2 via SIP notify requests about other vehicles in proximity. As an add on it is also possible to establish a direct communication channel between Car 1 and 2 via the WLAN interface.

The intended use case for this type of application is to alarm drivers of other vehicles in vicinity, even if there is no direct visual contact.

3.2 Overview

Figure 3.2 provides an overview of our proposed software stack from bottom to top.

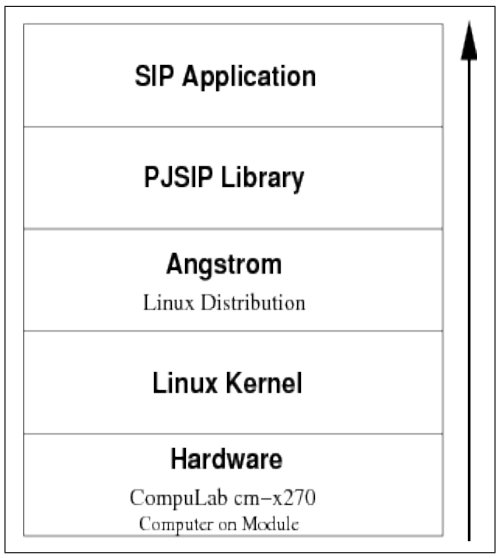

Figure 3.2: Embedded SIP Application Stack

The hardware for our embedded system is a CompuLab cm-x270 Computer on Module (CoM). The hardware can be replaced with little modification to the upper layers required. The portable Linux operating system kernel runs on the system hardware. The kernel provides hardware management and networking services to the upper layers. The Linux distribution `Angstrom` provides standard applications and interfaces like the C library and networking tools. For more information about Linux see section 3.6 “Linux”. The `PJSIP` library implements the standard SIP protocol and is used by SIP Applications in the top layer. Section 3.7 “Session Initiation Protocol” describes the basics of the SIP protocol. “Choosing a SIP library” gives an overview about `PJSIP`. All software layers presented in figure 3.2 are based on Open Source Software.

[1] For more information about GPRS and UMTS networks see the official 3GPP web-site at `http://www.3gpp.org/specifications`

3.3 Requirements for an Embedded System

It is necessary to use an embedded system for our application because of environmental requirements in vehicles. Compliance with the following acceptance standards is required

- Temperature (cf. MIL-STD-810F-502.4-C1)
- Change of Temperature (cf. MIL-STD-810F-504.3-1)
- Dust/Dirt and Splash water (cf. DIN-VDE-0470-IP-55)
- Vibration (cf. MIL-STD-810F-514.5-20)
- Shock (cf. MIL-STD-810F-516.5-I-V)

3.4 Open Source Software

In Open Source Software, the source code which is the basis of all computer software is freely available and can be modified by everyone. At the other end of the spectrum are proprietary software products, where the end user is only allowed to use the application but has no right to view or modify the source code. All Open Source Software is covered by a license approved by the Open Source Initiative (OSI)[2]. For a complete definition of the term Open Source Software see [OSI07].

OSI lists all software licenses that are appropriate for Open Source Software on their website. In this thesis, only licenses that apply to our project are covered.

GNU General Public License is a license by the Free Software Foundation (`http://www.fsf.org`). Currently two major versions are available: GPLv2 and GPLv3. GPLv3 was a major update to v2 and was released in June 2007. The GPL is well known in the software developing world and most developers at least know the license by name. Prominent software covered by the GPL is the Linux Kernel and the GNU Emacs editor which was used to write this thesis. The major points of the GPL are

- The program can be used without any restrictions. Commercial use of the software is explicitly allowed.
- Copies of the program covered by the GPL can be distributed for free or for money but the source code has to be included.
- The inner workings of the program can be studied and modified to fit one's needs.
- Modified versions of the program have to be distributed under the terms mentioned above. The source code of the modified version has to be made freely available.

One important aspect of the GPL is that any derivative work automatically falls under the GPL as well. Exactly what "derivative work" means is a point of contention between legal experts. Some claim that if a program uses a library that is covered by the GPL,

[2] see also `http://www.opensource.org/`

the program using this library also has to be licensed under the GPL. This issue has never been decided in a court of law.

Lesser GNU General Public License The major difference between GPL and LGPL is that if a program uses a library covered by the LGPL, it does not have to be licensed under the GPL. The library itself has the same conditions as software under the standard GPL license, but programs that use the LGPL library do not have to fall under the GPL.

One example of LGPL software is the GNU C library. Because the GNU C library is used by almost all GNU and non-GNU software in the Open Source world it is covered under the LGPL to allow different licenses for programs that just link to the C library.

BSD License has its roots at the University of Berkeley in California. BSD stands for Berkeley Software Distribution and is also a software license approved by OSI. The difference between the GPL variants of software licenses and the BSD license is that developers are not obliged to release software products that use pieces of BSD covered software under the same license. It is possible to use software under a BSD license in proprietary software products. Releasing the source code of the resulting application is not necessary. The only requirement is to leave the license statement in the original software.

3.4.1 Why Open Source Software

In this section we make a case for using Open Source Software in software projects and why releasing software covered by any Open Source license is important. The points mentioned below are by no means complete or based on scientific facts, but emphasize our decision to use Open Source Software.

Reliability We believe that source code that is viewable and usable by everyone is more robust than proprietary code. The more people use a particular piece of software the more bugs will be discovered.

Code quality improves if more people have access to our work. More people reading the code also means that more questions will be asked about the code and this results in the revelation of more bugs.

Openness is an important point in the academic field. Not only should academic papers be published but the source code of software written for these papers should be made freely available. If more people have access to academic work the public will benefit from that fact.

3.5 Embedded System Basics

In this section we explain common terms used in the embedded world. First we give a definition of the term "embedded system" and what distinguishes it from a non-embedded system. We answer the question "What is an embedded system" in section 3.5.1. Section 3.5.2 "How do we create code for embedded systems?" describes the tasks necessary to develop software for an embedded hardware architecture. Because most embedded systems use some form of Flash memory for permanent storage instead of harddisks, "NOR vs. NAND Flash Memory" (section 3.5.3) explains the most important differences between NOR and NAND flash memory.

3.5.1 What is an embedded system?

What are the differences between an embedded system and a desktop PC system? This question is not easy to answer because there is no clear line between the two. Embedded systems come in various sizes from very small computers embedded into a watch to large data storage systems. [Hal07] defines attributes that are specific to embedded systems:

- An embedded system contains a processing engine. This can be a general purpose microprocessor or processor specialized for a given task.
- Embedded systems are designed for a specific task.
- Embedded systems have a simple user interface or none at all.
- Resources such as physical memory are often limited.
- Embedded systems are usually not a general purpose computing platform.
- Hardware and software are often pre-integrated.
- Embedded systems often run without user intervention.

Limited resources is often the clearest distinction between embedded and non-embedded systems. On many occasions embedded platforms have limited memory and CPU resources and do not contain a hard drive. The user interface is scarce (e.g. a serial port or an LED display) and sometimes there is no user interface at all.

3.5.2 How do we create code for embedded systems?

When developing software on a PC system, it is usually written and compiled for execution on the same machine or CPU architecture. The question is how does this work if the target platform has a different CPU architecture? The resulting binary will not run on a different microprocessor. This is where a `cross compiler` comes into play.

A cross compiler creates executable code for a particular platform which is different to it's own. For example we compile code for our project on an notebook computer with an Intel x86 compatible processor and the compiler creates binaries that run only on platforms with an ARM CPU.

When talking about cross compilation we must distinguish between host, build and target platform.

Host is the platform where the compiler itself is compiled.

Build identifies the platform where the compiler is used to create binary code.

Target is the platform on which the compiled code runs.

A simple example will clarify the different terms. Let us consider three separate systems A, B and C. Each of them is based on a different CPU architecture: A is Intel based, B runs an ARM CPU and C executes only PowerPC binary code. Using a compiler that runs on `host` A, we create a second compiler that runs natively on our `build` system B but creates code for the `target` system C.

If all three systems are based on three different CPU architectures as in our example, we are talking about a `Canadian Cross`. Most of the time system A and B (the host and build systems) are the same.

More about cross compilation can be found in [Yag03] and [vH04].

3.5.3 NOR vs. NAND Flash Memory

Because embedded systems often do not contain a harddrive but instead use some form of flash memory we will provide an overview of NOR and NAND Flash memory. After reading this section the major differences between these two variants and why they are used in the embedded world should be clear.

Flash memory is a type of Electrically Erasable Programmable Read-Only Memory (EEPROM) . It is non-volatile and is programmed or erased in large blocks called `Erase Blocks`. Reading flash memory is fast but write times are in the range of milliseconds to seconds depending on the type used. Flash memory allows only a finite number of write/erase cycles per block, this property is called `Memory Wear`. Wear leveling, distributing write/erase operations evenly over all blocks has to be done by the flash memory controller or via special filesystems (see section 3.6.7).

When deleting Flash memory all bits within a block have to be set to one. Single bits can be set to zero. Deleting single Bits (setting them to 1) is **not** possible. It is necessary to delete whole blocks, so this blocks are called `Erase Blocks`.

NOR Flash has a typical size between one and four MB. An important property of NOR Flash is that it is addressable byte wise via a so called SRAM interface. CPU's have direct access to this type of memory, so NOR Flash looks like standard memory to the CPU. This is also called `eXecute In Place (XIP)`. A major disadvantage of NOR flash are the slow write times which are around 5 seconds per block.

NAND Flash comes in sizes between four MB and 16 GB. The major differences to NOR Flash memory is the complex I/O interface which varies from manufacturer to manufacturer. Under Linux NAND Flash normally looks like a standard block device[3] (e.g. /dev/sda). Deleting a block on NAND flash only takes a few milliseconds in comparison to seconds with NOR flash.

3.6 Linux

Linux is the operating system we chose for our embedded platform. In the following sections we examine the history of Linux, introduce the operating system, explain the difference between the Linux kernel and a Linux distribution, cover the boot process and clarify the difference between block and character device. We finish this chapter with a short intruction to the "Journaling Flash Filesystem Version 2".

3.6.1 An Introduction to Linux

In 1991 the creator of Linux, Linus Torvalds introduced the Linux kernel to the public with a post to the newsgroup comp.os.minix[4]. At the time it was seen by Linus as a hobby project and not as a professional operating system. Because it was released under an Open Source license and everybody had access to the source code, people started to contribute to Linux.

The first version ran on an Intel based i386 processor and had only support for AT harddisks. Today Linux supports many different processor architectures like ARM, SH, PowerPC and MIPS. A complete listing can be found in the Linux kernel source tree[5] in the directory `arch/`. It is now one of most portable Operating Systems on the market.

Nowadays there are large companies behind the development of the Linux Kernel. RedHat and Novell are specialized around Linux and sell so called Linux Distributions (see also section 3.6.2). Companies like IBM, Intel or Oracle are also big contributors to Linux and employ professional software developers who write code for the Linux kernel.

Linux follows a distributed software development model. Linux Torvalds is the maintainer of the so called mainline kernel. The mainline kernel is the version accessible on the web site `http://kernel.org`. Everybody who wants to contribute to the development of the kernel is free to download and modify the source tree. To include that modification in the mainline kernel one sends a patch[6] to the Linux kernel mailing list. The kernel mailing list is the main communication platform for all things related to the Linux kernel. The patches are going to be discussed on the mailing list and if they are considered good enough they will be included in one of the next releases of the kernel. For more information about the kernel mailing list see `http://vger.kernel.org/`.

[3] see also section 3.6.6

[4] Minix is another Open Source operating system see http://www.minix3.org

[5] The Linux source can be found at http://kernel.org

[6] A patch includes only the differences between the modified and the original source code. See also http://www.gnu.org/software/patch/

Linux is a monolithic kernel where device drivers and kernel extensions run in kernel space (ring 0 on many CPU architectures). Despite being monolithic Linux supports loading of modules (kernel extensions) at runtime. This is a very convenient feature because it is possible to load device drivers at runtime.

3.6.2 Linux Kernel vs. Linux Distribution

One of the more confusing points about Linux is the question what Linux exactly is. Linux is an operating system kernel. A kernel manages the underlying computer hardware and provides software with a well-defined application programming interface to interact with the kernel.

A complete operating system also needs tools like a shell (e.g. bash), a window manager and an editor. The kernel and all software tools necessary for a complete operating system form a distribution. Most of the time when referring to Linux, people mean a distribution and not the kernel.

Some of the more well known Linux distributions are RedHat, SuSE or Debian. RedHat and SuSE are managed by companies (SuSE is owned by Novell Inc.) and the Debian distribution is a volunteer project.

3.6.3 Linux Kernel Compilation

This section gives a short introduction on how to compile the Linux kernel. It is easier to understand how an embedded board boots up if you know how to create a kernel image.

The first step in compiling the Linux kernel is to download the source code from the official Linux kernel web page at `http://kernel.org`. At the time of this writing version 2.6.28 was the latest version and therefore used for this introduction.

Listing 3.1 demonstrates the download process using the Debian distribution.

```
1  $ wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.28.tar.bz2
   --2009-01-02 19:12:11--  http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.28.tar.bz2
3  Resolving murus.stderror.at... 83.65.196.90
   Connecting to murus.stderror.at|83.65.196.90|:8080... connected.
5  Proxy request sent, awaiting response... 200 OK
   Length: 52665364 (50M) [application/x-bzip2]
7  Saving to: 'linux-2.6.28.tar.bz2'

9  100%[======================================>] 52,665,364   174K/s   in 5m 51s

11 2009-01-02 19:18:02 (147 KB/s) - 'linux-2.6.28.tar.bz2' saved [52665364/52665364]

13 $ tar jxf linux-2.6.28.tar.bz2
   $ cd linux-2.6.28
15 $ make menuconfig
   $ make
```

Listing 3.1: downloading the Linux kernel

We use the tool `wget` to download the kernel source tree. If `wget` is not available it can be installed with the command `apt-get install wget`. The command in line 13 extracts the source tree into the directory `linux-2.6.28`.

The Linux kernel build system provides four main interfaces for configration:

1. `config`: is the simplest tool available and asks all options on a step by step basis.
2. `menuconfig`: is a text-based menu system based on the ncurses[7] library.
3. `xconfig`: starts a QT based graphical tool and requires a running X server[8].
4. `gconfig`: starts the same graphical tool as `xconfig` but is based on the GTK library.

We change into the directory kernel directory and call GNU `make`[9] with the target `menuconfig`. This starts the building of the kernel configuration tool which is used to set various options the kernel provides, e.g. device drivers to build, hardware support, processor type etc.

The command `make help` lists all available make targets. Figure 3.3 depicts the kernel configuration tool.

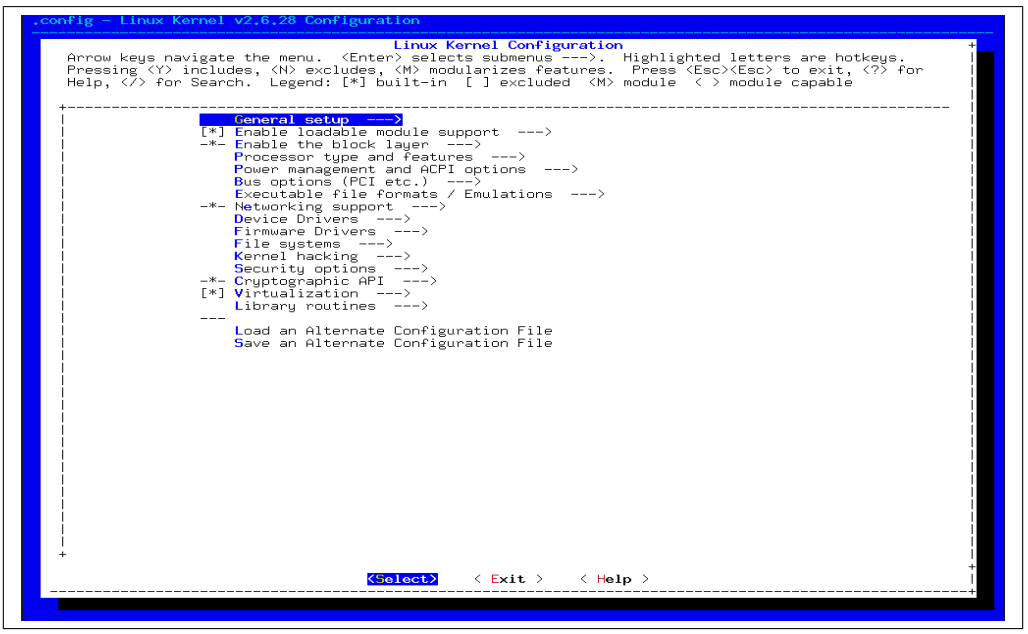

Figure 3.3: Linux Kernel configuration tool

To cross compile (see section 3.5.2) the Linux kernel it is necessary to specify the target architecture. For example

```
make ARCH=arm menuconfig
```

enables all options that are specific to the ARM processor architecture.

After configuring the kernel, start the compilation process with the command

```
make
```

[7] For more information about the ncurses library see `http://www.gnu.org/software/ncurses/`
[8] Simple put, a X server provides the operating system graphical interface.
[9] For more information about GNU make see `http://www.gnu.org/software/make/`

This builds a bootable kernel image which can be found in the directory `arch/x86/boot` if the kernel was built for an Intel compatible processor.

3.6.4 Linux Kernel Image

The result of compiling the kernel is the Linux kernel image. This binary file `vmlinux` represents the executable kernel. Figure 3.4 depicts the various parts of the image. This is a modified version of the same figure found in the excellent book [Hal07].

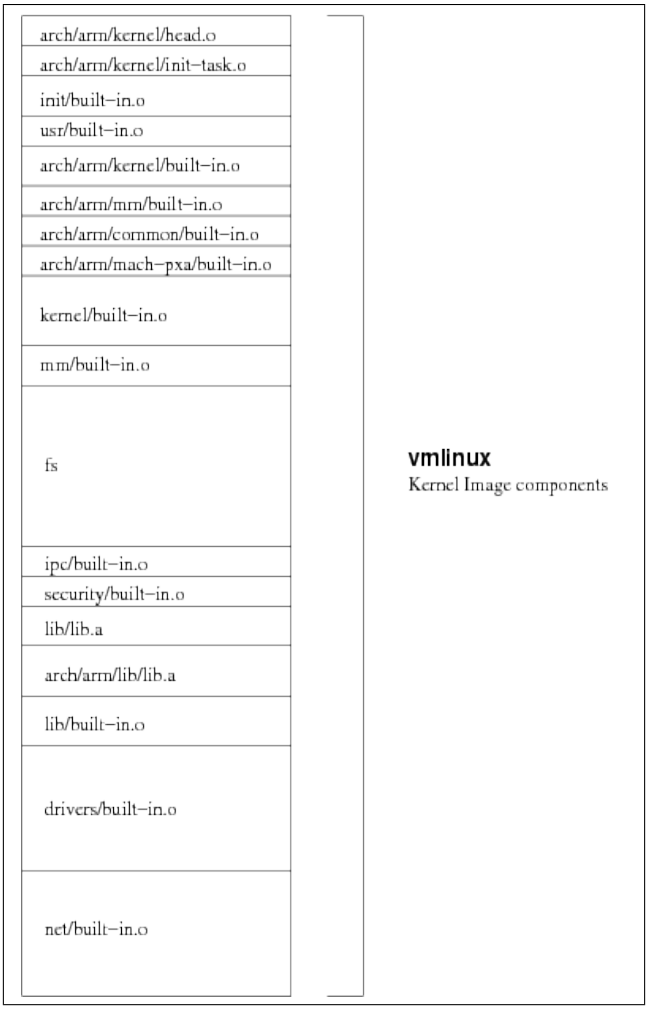

Figure 3.4: components of vmlinux kernel image [Hal07]

Table 3.1 gives a short description of the various components of the kernel. This table was also taken from [Hal07] and modified to fit the kernel built for the cm-x270.

Creating a bootable kernel image involves various steps and different formats of the resulting binary. Figure 3.5 depicts the process of creating the kernel image. All steps are performed automatically by the kernel build process.

`vmlinux` is a binary file in ELF format (see also [Lev00]). Because not all bootloaders are

Component	Description
arch/arm/kernel/head.o	Kernel architecture-specific startup code
arch/arm/kernel/init-task.o	Initial thread and task structs
init/built-in.o	Main kernel-initialization code
arch/arm/kernel/built-in.o	Architecture specific kernel code
arch/arm/mm/built-in.o	Architecture specific memory-management code
arch/arm/common/built-in.o	Architecture specific generic code
arch/arm/mach-pxa/built-in.o	Machine specific code, usually initialization
kernel/built-in.o	Common components of the kernel itself
mm/built-in.o	Common components of memory-management code
ipc/built-in.o	Interprocess communication helper functions
security/built-in.o	Linux security components
lib/lib.a	Archive of miscellaneous helper functions
drivers/built-in.o	All the built-in drivers - not loadable modules
net/built-in.o	Linux networking
fs	Filesystem code

Table 3.1: Linux kernel image components

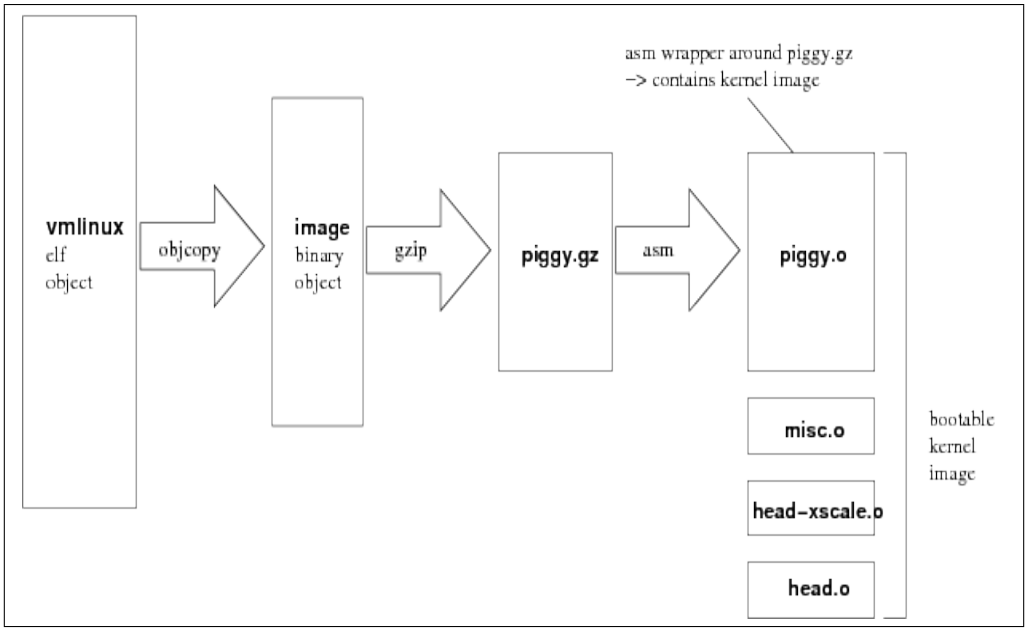

Figure 3.5: Linux Kernel Image [Hal07]

able to directly load and run an object file in ELF format, the image is converted further. The first step is to remove the ELF sections `.note` and `.comment` and strip debugging

symbols[10]. This is done using the tool `objcopy`[11]. Next `gzip` is used to compress the binary kernel image to a file named `piggy.gz`. Now the following files are compiled:

- **piggy.o** This is compiled from the assembly language file `piggy.S` that includes a reference to the compressed kernel image.
- **misc.o** This object file includes functions to decompress the kernel image.
- **head-xscale.o** Processor initialization code for the Xscale (Intel ARM implementation) CPU.
- **head.o** ARM specific startup code. This object is passed control by the boot loader.

The last step is linking all the files mentioned above to the final image named `zImage`.

3.6.5 Boot Process

In this section we describe the boot process of the ARM based CompuLab cm-x270 embedded system. Figure 3.6 is a general overview of the steps involved when booting the cm-x270.

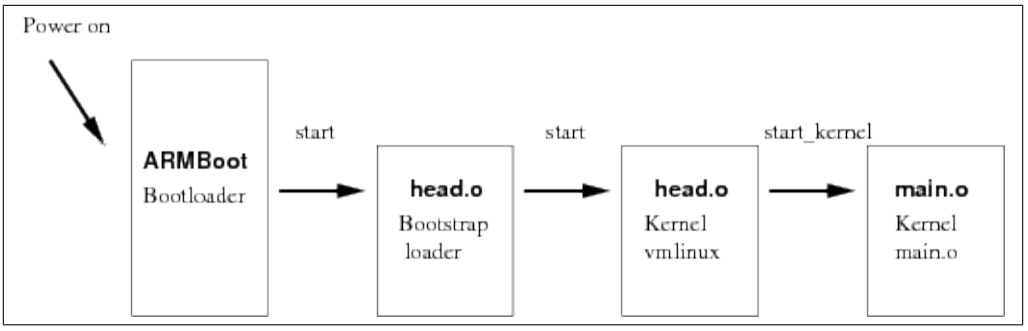

Figure 3.6: Linux Kernel boot control flow [Hal07]

On power on the ARM CPU on the cm-x270 loads the ARMboot bootloader from the address `0x00000000` (see also section 4.2.3) which is within the NOR flash memory. The ARMboot command `bootos` loads the Linux kernel image (`zImage` in our case) from address `0x00040000`. When the bootloader passes control to Linux kernel the first code executed can be found in the module `head.o`. As explained in the previous section this object file contains general ARM specific startup code (e.g. initialization of CPU caches). `head.o` also calls code found in the modules `head-xcale.o` and `misc.o`. After CPU initialization the compressed kernel image is decompressed into system memory. When booting the Linux kernel the following message should be printed to the system console

[10] A binary in ELF format is split in different sections. For example there is a TEXT section which holds the executable code, a DATA section for initialized global variables and a BSS section for uninitialized data.

[11] see also the UNIX manual page for objcopy(1).

```
Uncompressing Linux...done, booting the kernel.
```

On the cm-x270 with the proprietary bootloader ARMmon the above message is not shown. We think this is a problem with the system console on the cm-x270. The following messages are shown after the `bootos` command:

```
Linux image detected in flash at 0x00040000.
Loading linux kernel from flash... Done.
Booting...
Kernel parameters:
KERNEL_PARAMS: 0xA0000100
```

After decompressing the kernel image control is passed to the object `head.o` [12] found within the decompressed kernel image. This is not to be confused with the other `head.o` object in the bootstrap loader which recieved control from the ARMboot bootloader. The main task of `head.o` is to do further architecture and CPU specific setup before entering the main kernel code. It creates the initial page table entries, sets up the processor memory management unit (MMU) and finally starts the main kernel loop in `main.o`. One thing worth noting is that before `main.o` executes, the CPU is in real mode. In real mode (a term mainly found on x86 based architectures) there is a one-to-one mapping between logical and physical memory address space. `head.o` switches the CPU to protected mode (also a x86 term) by turning on the memory management unit. After this point the CPU uses virtual addresses which are translated by the MMU to physical addresses.

The following messages are shown after entering `head.o` in the uncompressed kernel:

```
Linux version 2.6.26.2-dirty (pinhead@bluebook) (gcc version 4.2.4) #1 PREEMPT Sat Aug 16 17:51:31 CEST 2008
CPU: XScale-PXA270 [69054117] revision 7 (ARMv5TE), cr=0000397f
Machine: Compulab CM-x270
ATAG_INITRD is deprecated; please update your bootloader.
Memory policy: ECC disabled, Data cache writeback
Run Mode clock: 208.00MHz (*16)
Turbo Mode clock: 520.00MHz (*2.5, active)
Memory clock: 208.00MHz (/2)
System bus clock: 208.00MHz
CPU0: D VIVT undefined 5 cache
CPU0: I cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets
CPU0: D cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 16256
Kernel command line: console=ttyS1,38400 monitor=1 mem=64M
mtdparts=physmap-flash.0:256k(boot)ro,0x180000(kernel), \\
   -(root);cm-x270-nand:64m(app),-(data) rdinit=/sbin/init root=mtd3 rootfstype=jffs2
PID hash table entries: 256 (order: 8, 1024 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 64MB = 64MB total
Memory: 57728KB available (2796K code, 197K data, 108K init)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
net_namespace: 192 bytes
NET: Registered protocol family 16
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
```

[12] `arch/arm/kernel/head.S` is the source code for `head.o`

```
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
NET: Registered protocol family 1
checking if image is initramfs...it isn't (bad gzip magic numbers); looks like an initrd
Freeing initrd memory: 2285K
JFFS2 version 2.2. (NAND) (SUMMARY)  Â© 2001-2006 Red Hat, Inc.
msgmni has been set to 117
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered
io scheduler cfq registered (default)
Console: switching to colour frame buffer device 80x30
pxa2xx-uart.0: ttyS0 at MMIO 0x40100000 (irq = 22) is a FFUART
pxa2xx-uart.1: ttyS1 at MMIO 0x40200000 (irq = 21) is a BTUART
console [ttyS1] enabled
pxa2xx-uart.2: ttyS2 at MMIO 0x40700000 (irq = 20) is a STUART
brd: module loaded
loop: module loaded
dm9000 Ethernet Driver, V1.30
eth0: dm9000 at c4856000,c4858008 IRQ 74 MAC: 00:01:c0:02:4b:2f (chip)
usbcore: registered new interface driver rt73usb
Driver 'sd' needs updating - please use bus_type methods
physmap platform flash device: 00400000 at 00000000
physmap-flash.0: Found 1 x16 devices at 0x0 in 16-bit bank
 Amd/Fujitsu Extended Query Table at 0x0040
number of CFI chips: 1
cfi_cmdset_0002: Disabling erase-suspend-program due to code brokenness.
3 cmdlinepart partitions found on MTD device physmap-flash.0
Creating 3 MTD partitions on "physmap-flash.0":
0x00000000-0x00040000 : "boot"
0x00040000-0x001c0000 : "kernel"
0x001c0000-0x00400000 : "root"
NAND device: Manufacturer ID: 0x2c, Chip ID: 0xdc (Micron NAND 512MiB 3,3V 8-bit)
Scanning device for bad blocks
Bad eraseblock 2138 at 0x10b40000
Bad eraseblock 3127 at 0x186e0000
Bad eraseblock 4067 at 0x1fc60000
2 cmdlinepart partitions found on MTD device cm-x270-nand
Using command line partition definition
Creating 2 MTD partitions on "cm-x270-nand":
0x00000000-0x04000000 : "app"
0x04000000-0x20000000 : "data"
usbmon: debugfs is not available
pxa27x-ohci pxa27x-ohci: PXA27x OHCI
pxa27x-ohci pxa27x-ohci: new USB bus registered, assigned bus number 1
pxa27x-ohci pxa27x-ohci: irq 3, io mem 0x4c000000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 3 ports detected
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
mice: PS/2 mouse device common for all mice
Registered led device: cm-x270:red
Registered led device: cm-x270:green
usbcore: registered new interface driver usbhid
usbhid: v2.6:USB HID core driver
TCP cubic registered
NET: Registered protocol family 17
XScale iWMMXt coprocessor detected.
Empty flash at 0x026288b8 ends at 0x02629000
VFS: Mounted root (jffs2 filesystem).
Freeing init memory: 108K
INIT: version 2.86 booting
```

Messages listed below are an important part of the boot process and require further explanation:

ATAG_INITRD: When booting the linux kernel on the ARM architecure there is a need to pass options describing the hardware (memory layout, size of memory pages and so on) from the boot loader to the kernel. This is done by placing these parameters (c structures) into system memory. The kernel searches for these structures at the start of physical memory plus `0x100` or the start address is placed in the CPU register R2. One of these parameters points to the initial ramdisk `ATAG_INITRD`. For more information see the file `Documention/arm/Setup` within the kernel source tree.

Kernel command line: console=ttyS1,38400...: The kernel command line is used

to pass user options to the kernel. For example which device should be used for the system console (in our case ttyS1, the second serial port), IP address configuration that is needed to mount the root filesystem from a nfs server etc. Most of these user definable parameters are documented in the file `Documentation/kernel-parameters.txt` within the kernel source tree.

3 cmdlinepart partitions found on MTD device physmap-flash.0: MTD stands for Memory Technology Devices which is a Linux kernel subsystem to support different types of memory-like devices like Flash memory. Because MTD devices support partitions, the partition layout has to be communicated to the kernel. No partition tables are stored on MTD devices. In our case the partition layout is passed to the kernel via the kernel command line options:

```
mtdparts=physmap-flash.0:256k(boot)ro,0x180000(kernel), \\
   -(root);cm-x270-nand:64m(app),-(data) root=mtd3
```

After booting the embedded board the following command displays the partition layout:

```
root@cm-x270:~# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00040000 00010000 "boot"
mtd1: 00180000 00010000 "kernel"
mtd2: 00240000 00010000 "root"
mtd3: 04000000 00020000 "app"
mtd4: 1c000000 00020000 "data"
root@cm-x270:~#
```

The parameter `root=mtd3` tells the kernel where to find the root filesystem.

INIT: version 2.86 booting: After mounting the root filesystem kernel startup is finished and the last step is to fork the so called `INIT` process. `INIT` always has the process id 1 in Linux and it is the first process that runs after kernel initialization. It is `INIT's` responsibility to further set up the system, for example start various background processes and present the user with the login screen. The kernel has a hardcoded search path to find `INIT`, the following paths are tried in order:

```
run_init_process("/sbin/init");
run_init_process("/etc/init");
run_init_process("/bin/init");
run_init_process("/bin/sh");
```

3.6.6 Block vs. Character Device

This section provides a short explanation of the terms `block` and `character` device.

Block device is accessed in blocks, the size of one block depends on the underlying device but is typical 512 Bytes for hardisks. Linux uses block devices for accessing standard disks. Filesystems are created on block devices. The block size of a block device is not to be confused with the block size of a filesytem which is equal to or larger than it.

Character devices allow byte wise access to the underlying device. Modems are typical character devices.

3.6.7 Journaling Flash File System Version 2

The Journaling Flash File System Version 2 (JFFS2) is a filesystem developed by RedHat especially to be used on NAND flash memory. It is an enhanced version of the Journaling Flash Filesystem (JFFS) by Axis Communictions AB.

Before the development of a special filesystem for flash memory under Linux a so called Flash Translation Layer (FTL) [Coo98] or a NAND Flash Translation Layer (NFTL) were used to store data in NAND memory. The FTL emulates a standard block device that can be used with any available filesystem. Therefore the FTL hides the characteristics of flash memory (e.g. erase only possible on whole blocks, see also section 3.5.3) and allows the usage of standard filesystems like NTFS or EXT3.

A problem that arises with filesystems like NTFS or EXT3 is that they were written with standard block devices in mind. There is no such thing as wear leveling (see section 3.5.3) on magnetic disk drives. Using a standard fileystem is not the best solution for NAND flash memory.

The JFFS2 filesystem was is a Log-Structured Filesystem (LFS) . LFS is a filesystem that is written like a log file. New data is appended to the filesystem instead of overwriting entries that have changed. For more information on Log-Structured Filesystems see [MR91].

The original version of JFFS had some downsides so RedHat decided to implement a second version. There were improvements to the garbage collector which is reponsible to delete old entries, the original version had no support for compression of the filesystem and unix hard links were not supported. For more information to the JFFS2 filesystem see [Woo01].

3.7 Session Initiation Protocol

Because the session initiation protocol is an integral component of the software stack this section covers the basics of the protocol. SIP is a complex protocol and going into more detail would cover a complete master thesis alone. For a complete description of the protocol please see RFC3261 [RSC+02].

Before we look at the protocol we cover basic SIP terminology:

User Agent: SIP terminals are called User Agents. A SIP User Agent is a client if it sends SIP requests and a SIP server if it answers requests.

Address of Record: Every SIP User Agent is identified by an address of record in the form of an Universal Resource Identifier (URI). The SIP URI scheme is similar to an e-mail address and has the form `sip:<username>@<domain>` for example `sip:toni@stderror.at`.

Contact Address: The contact address is the temporary physical address of a SIP user agent. When an user agent moves from one network to the other the contact address changes but the address of record stays the same. The contact address has the form `sip:<username>@<IP address>`, for example `sip:toni@85.96.196.90`.

Outbound Proxy: If a User Agent client wants to contact to a User Agent server it has to forward all SIP request to the outbound proxy. It is the responsibility of the outbound proxy to find the inbound proxy (see below) and to forward the request to the inbound proxy.

Inbound Proxy: The inbound proxy queries the location database with the Address of Record to determine the Contact Address of the user agent server. After finding the contact address it forwards the request to the user agent server.

Location Database: The mapping between Address of Record and Contact Address is done with the help of the location database. Before an user agent is able to receive SIP `INVITE` requests it has to register itself with the SIP method `REGISTER`.

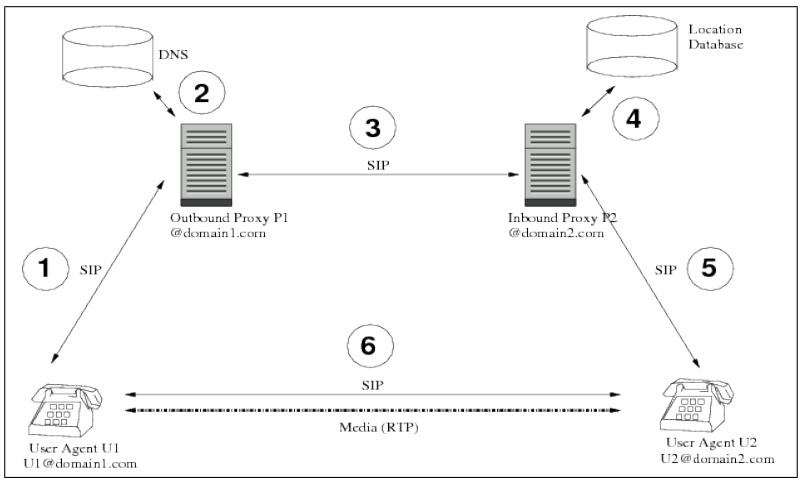

Figure 3.7: SIP Trapezoid

Figure 3.7 was taken from RFC3261 [RSC+02] and describes how two SIP user agents connect to each other.

User agent U1 wants to establish a real time protocol (RTP) (see RFC 3550 [SCFJ03]) session with user agent U2.

1. The first step for U1 in creating this connection is to send a SIP `INVITE` request to the so called SIP outbound proxy.
2. Via DNS lookups (SRV and NAPTR records) the outbound proxy searches for the IP address of the inbound proxy for user agent U2.
3. Upon finding a valid address the outbound proxy forwards the SIP `INVITE` request to the inbound proxy of U2.
4. Now the inbound proxy does a lookup in the location database where the mapping between the address of record (AoR) and the contact address is stored.
5. If the lookup was successful the request gets forwarded to user agent U2. User agent U2 replies with an SIP `OK` answer. This `OK` request travels form user agent U2 to the inbound proxy, then to the outbound proxy of user agent U1 and finally to user agent U1.
6. When user agent U1 receives the `OK` request it starts the media session to user agent U2.

The important point here is the direct connection between user agent U1 and U2 for the media stream. SIP is a signaling protocol and its only responsibility is session establishment. SIP is not a media transport protocol.

4 Choosing an Embedded Platform

In this section we provide evaluation criteria for choosing an embedded system platform. When starting a new embedded software project selecting the right hardware components is a difficult task. Section 4.1 provides criteria that should be followed to make the selection process easier.

Section 4.2 describes one particular embedded system that was used in our reference implementation.

4.1 Evaluation Criteria

In this section we provide important evaluation criteria that should be used when comparing different platforms and which will help to made the right choice.

Documentation: Documentation is one of the most important evaluation criteria. The project team had little experience with embedded systems; therefore it was very important to find a well documented system.

There are various embedded platforms on the market and high quality products are well documented. The following questions should be helpful in deciding whether the provided documentation is adequate or not:

- Is a complete description of the hardware available (interfaces, bus topology etc.)?
- Is the physical layout documented (dimensions, connectors location)?
- Is the boot process documented?
- Is the boot loader documented?
- What kind of flash memory is available on the board?
- Is the operating system installation process covered?

Preconfiguration: Starting with a new platform is easier if a preconfigured operating system image is already available. It is not required to use this image on a production system, but getting to know the platform requires less effort.

Price: In our opinion this should not be the most important selection criteria when evaluating embedded platforms. But platforms with good documentation and support will definitely cost more than others.

Support by the Manufacturer: Getting support from the manufacturer of the embedded board in case of hardware defects or other difficulties (e.g. problems with the operating system, applications not working as expected) is an important issue. In some cases good documentation and a large user base is not sufficient to find solutions for difficult problems. Having a capable support team is sometimes the only way to find a solution.

User Base: A large user base using the same embedded platform in a variety of different projects is a definitive plus. If the project team is the first to use an embedded board, it is likely that the team will face difficulties. In such a situation manufacturer support is even more important.

Community: A strong community around a particular embedded platform can be very helpful if problems arise. Active Mailing lists and web forums are an indicator of such a community. Most smaller problems that emerge during the development process might be solved via mailing list postings or web forums.

4.2 CompuLab cm-x270

The cm-x270 is a so-called computer on a module, built and sold by the Israeli company CompuLab. "Computer on a Module" means that all components needed for a full-fledged computer are built onto a single circuit board. This particular board was chosen by Technikum Wien as a platform for a software project involving car to car communication via the SIP event framework. We had no influence in the decision process and had to use the cm-x270. Nevertheless as we will see the cm-x270 is a capable embedded platform for this kind of project.

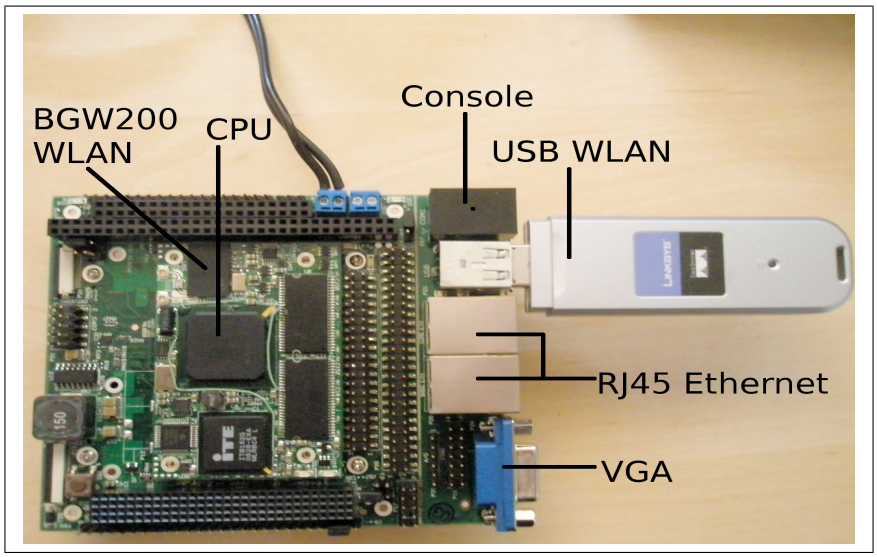

Figure 4.1: CompuLab cm-x270 with wireless usb stick

4.2.1 Hardware description

The cm-x270 consists of the following components:

- Intel PXA 270 32bit CPU 520Mhz
- ARMv5 Instruction Set
- 64MB SDRAM 100Mhz
- 4MB NOR Flash
- 64MB NAND FLASH
- 2x 10/100 Ethernet Interface
- 2x USB 1.1 (OHCI)
- 1x serial Interface for serial console access
- 1x WLAN Interface BGW200 (802.11b)
- GSM/GPRS Modem
- PXA270 graphic chip

4.2.2 Memory Layout

One important point in understanding an embedded platform is its memory layout. The documentation provided by CompuLab covers the basics and is presented in the following table:

Memory address range	Usage
0x00000000 - 0x0002FFFF	ARMMON bootloader
0x00030000 - 0x0003FFFF	Setup Block
0x00040000 - 0x001BFFFF	OS Kernel Image
0x001C0000 - NOR flash top	OS Ramdisk
0xA0000000 - 0xA3FFFFFF	SDRAM 64 MB

Table 4.1: cm-x270 memory layout

The address range `0x00000000 - 0x9FFFFFFF` covers the NOR flash memory. Within this range from address `0x00000000` to `0x0002FFFF` is the ARMMON boot loader. When the cm-x270 is powered on the CPU loads the first instruction from memory address `0x00000000`. As a first step the bootloader is started. ARMMON is preinstalled and should not be overwritten otherwise booting becomes impossible.

Following the boot loader is the so called setup block from address range `0x00030000` to `0x0003FFFF`. It stores persistent ARMMON configuration options like IP addresses and boot parameters.

The operating system kernel is stored within `0x00040000 - 0x001BFFFF`. CompuLab provides precompiled kernel binaries for Windows CE and Linux.

Starting with address `0x001C0000` until the end of the NOR flash is the so called operating system ramdisk. Linux uses the ramdisk as an intermediate root file system to load drivers that are required to load the real root file system (e.g. drivers for a storage subsystem). These drivers do not have to be included in the kernel binary and can be loaded as modules.

The rest of the addressable memory range is used as main memory for the CPU and has a size of 64MB.

Boot Loader

`ARMMON` is a proprietary boot loader used by CompuLab for the cm-x270. It is the responsibility of the boot loader to initialize the hardware and to load the operating system. Furthermore `ARMMON` provides the capability to format and write to NOR and NAND memory. `ARMMON` includes a minimal TFTP client for downloading the kernel and root filesystems. So the boot loader includes a minimal IP stack that is quite capable. A serial interface is provided to connect to `ARMMON`.

CompuLab provides extensive documentation about the boot loader and all available options.

4.2.3 Booting the cm-x270

One question that always arises when working with a new embedded platform is, how does the board boot? This section should provide an overview of the boot process. Booting an embedded board is always very hardware specific. Not only does the boot process change with different CPU architectures, but also with different hardware manufactures. This section is very specific to the CompuLab cm-x270.

After the system is powered on and CPU initialization is complete, the first instruction is fetched from memory address `0x00000000`. In case of the cm-x270 this means the first instruction is loaded from NOR memory. The boot loader `ARMMON` starts at this memory address and is loaded. When `ARMMON` is executed the user sees the following messages (Listing 4.1) via the attached serial console:

```
Welcome to ARMmonitor running on Compulab CM-X270.
Copyright Compulab 2002-2007 (c).

Built at: Wed May 30 18:51:19 IDT 2007 on 89

CM-X270 hardware configuration:
-------------------------------
SDRAM size ........................... 128MB
NOR flash ............................ AMD or compatible, 4MB
NAND flash ........................... Generic NAND, 512MB
PCI bridge ........................... present
Ethernet on CORE ..................... present
Ethernet on BASE ..................... present
AC'97 CODEC .......................... present
RTC .................................. present

ARMmon >
```

Listing 4.1: boot screen

ARMMON is now waiting for user input. If an operating system kernel and root file system are already installed on the board the command bootos loads the kernel from address 0x00040000 (see section 4.2.2) and starts executing the kernel. There is also a configuration option to boot directly into the operating system without ARMMON waiting for input (setboot os).

4.2.4 NOR and NAND Flash on the cm-x270

The cm-x270 includes both NOR and NAND flash memory variants. NOR flash is about 4MB in size and stores the boot loader ARMMON, configuration settings for the boot loader, the kernel image and a minimal root filesystem. Because of size constraints it is not possible to store a complete Linux root filestem in NOR memory. For this purpose a different flash memory variant is necessary.

64MB of NAND flash memory provide enough space for a full fledged root filessytem. Unfortunately the standard cm-x270 setup uses a proprietary format for the NAND flash. It is only usable with a non-standard Linux driver. Fortunately reformating the NAND memory is possible and we describe this solution in section 7.6.

4.2.5 Wireless LAN Interface

A problem with the cm-x270 is the wireless LAN interface used on the chip. It is manufactured by Philips and only a binary driver BGW200 under a proprietary license is available. CompuLab includes the driver with the system but only for kernel version 2.6.16. The toolchain used for this project uses kernel version 2.6.26.2 so the wireless LAN interface was not supported.

The project team used a USB wireless LAN adapter as a solution to this problem. The cm-x270 provides two USB 1.1 ports with a maxium speed of 12MB/s which is sufficient for 802.11b wireless LAN access. Linksys provides an adapter (WUSB54GC) that uses a RaLink chipset which is well supported under Linux kernel 2.6.26.2.

5 Choosing a Toolchain

A toolchain is the infrastructure necessary to develop software for an embedded system. It is the responsibility of the toolchain to provide the software developer with tools to compile and deploy software to the embedded system. First we are going to provide some guidelines on how to choose an appropriate toolchain for an embedded system project. We then provide a short introduction to the toolchain we chose for our project.

5.1 Evaluation Criteria

Within this section we provide the reader with guidelines on how to choose the right toolchain for developing software on an embedded system.

Complexity: Software development for embedded systems is not an easy task. The learning curve is a steep one for inexperienced developers. Two options are available when starting a new project and evaluating a toolchain:

- Build you own, which means building a cross compiler, the necessary libraries for your application and a standardized process to deploy your application to the embedded system board.
- Use an existing toolchain and hope that it fits your needs and its usability is better than building your own.

When starting a new project it is easier to use an existing solution (if available). We started our project with an existing toolchain and customized it to our needs.

Documentation: When using a toolchain that is not well documented the time needed to understand and use it will be nearly the same as the roll-your-own case. It is very important to find existing examples that are well documented when learning a new software tool.

User Base: Active mailing lists and web forums are also very important. If many people use the same toolchain the probability of getting helpful answers to a problem rises.

5.2 OpenEmbedded

Creating a Linux distribtion from scratch is a complex task. Not only is it necessary to customize the operating system kernel, but also to use a toolchain to deploy the application onto the embedded system board. This toolchain includes a cross compiler to create binaries the destination platform understands, libraries to support the application (e.g. the C standard library libc) and tools to deploy the application on the embedded system.

OpenEmbedded (OE) is a toolchain to create complete Linux distributions for embedded systems. It was choosen as a toolchain for this particular project because the `cm-x270` is well supported. Creating a bootable kernel and a standard root filesystem image is quite easy and the learning curve is not too steep. The `cm-x270` is only one of many supported systems. So OpenEmbedded is well suited for many different platforms and projects.

OpenEmbedded itself contains metadata that describes how a Linux distribution has to be built. This covers creating the cross compilation environment, the C cross compiler and necessary libraries. There is metadata that describes which Linux kernel should be used for a particular embedded system board and how the kernel should be built. There is also metadata that covers which tools and libraries should be included in the Linux distribution and which format should be used for the root filesystem.

OpenEmbedded distinguishes between three different forms of metadata:

- configuration files
- `.bb` files
- classes

Configuration files include settings that are important for all software packages that have to be built. Configuration files can also be nested, so one file can include other configuration files.

`.bb` files are the so called recipes. BitBake (see section 5.2.1) uses these recipes to get information on how a particular software package has to be compiled and installed.

Classes describe the different steps BitBake has to take when compiling a software package. For example there is a class `autotools` which is responsible for compiling packages that use the GNU AutoTools[1].

The user has to create a configuration file that describes the destination platform. OpenEmbedded then builds a standard distribution that includes all necessary compenents to boot the embedded system. Customization is also possible if the distribution should include custom applications.

[1] `http://www.gnu.org/projects/autotools`

For a detailed description on how to setup OpenEmbedded, starting the compliation process and the customization process see section 7.1.

5.2.1 BitBake

`BitBake` is the so called "Task Executor" used by OpenEmbedded. BitBake reads the configuration files and executes the listed operations. It can be compared to GNU Make (`http://www.gnu.org/projects/make`) and `Portage` which is used by the Linux distribution Gentoo for creating software packages. It is written in Python and covered under the open source license GPLv2.

BitBake is responsible for the following tasks:

- cross compilation of software packages
- resolving dependencies between software packages
- downloading, applying patches and compiling the source code

For example the command

```
bitbake -c build-package-<packagename>
```

creates a new package for the software `packagename`. The `packagename` tells BitBake which configuration files have to be used for downloading, compiling and installing the software package.

BitBake not only includes tasks to create separate software packages, but also tasks for creating Linux distributions and filesystem images with all necessary tools.

To create a new software package the developer has to create a new configuration file that describes where the source code can be found (e.g. URL) and how the source code has to be compiled (e.g. GNU AutoTools or a standalone Makefile). It is BitBake's responiblity to download, patch, compile and create a suitable package.

The format of the finished software packages is `tar.gz` or `ipkg`. This is a configuration option.

5.2.2 Ångström

`Ångström` or `Angstrom`[2] is the Linux distribution we are going to build and deploy with the help of OpenEmbedded. It was started as a distribution for the OpenZaurus and OpenSimpad project and is well integrated into OpenEmbedded. For detailed explanation of what a Linux distribution is see section 3.6.2 "Linux Kernel vs. Linux Distribution".

To quote the `Angstrom` homepage:

[2] http://www.angstrom-distribution.org/

Ångström is verstatile, it scales down to devices with 4MB of flash to devices with terabytes of RAID storage. Someday it might even run on a toaster :)

6 Choosing a SIP Library

The CompuLab cm-x270 embedded board is used as a SIP platform in this particular project. Because of the complexity of the SIP stack, implementing the SIP protocol stack within the project was not an option. One assignment of the project team was to search for pre-existing implementions that would fit project needs. Section 6.1 describes some evalution criteria we used when evaluating different libraries. Later on section 6.2 `PJSIP` gives a short introduction into the PJSIP library we selected for the project.

6.1 Evaluation Criteria

The criteria listed below are nowhere complete, the main purpose is to provide hints to future projects with similar requirements.

License: A strict requirement for this project was to base all infrastructure (operating system of the embedded board, libraries, toolchain) on software which is available under an Open Source license. For a short introduction on Open Source software and two example OSS licenses see section 3.4.

Documentation: Documentation is an important point, especially when using an open source SIP library. Unclear or missing documentation is a problem often seen in open source projects. Nevertheless there are open source projects with excellent documentation. This is even more important when starting a new project. The project team should not waste time because of missing documentation.

Complexity of Integration: One task is to integrate the library within the toolchain and to deploy it on the embedded board. There are libraries available with embedded systems already in mind. These are generally easier to port to the destination platform. External dependencies should also be checked. There is the possiblity that the chosen library has dependencies to other libraries which must also be ported.

Resource Requirements: Limited resources are often a problem with embedded platforms. This includes physical memory, CPU power and disk/flash storage. Checking the resource requirements before deciding for a particular library is important.

Sample Applications: If sample applications are available using the library is easier. Custom applications can be derived from sample implementions.

User Base and Community: When using Open Source Software an active user base and a large community is often an indicator for the quality of the project. There is no agressive markting behind open source projects, what counts are satisfied users. If the library is badly written and barely usable no one would choose it for their software. Asking around on mailing lists and IRC channels is often a good way to get a first impression.

6.2 PJSIP

We decided to use the SIP library `PJSIP`. `PJSIP` is an open source implementation of the SIP protocol stack covered by the GPL. All code using `PJSIP` has to use the same license. The library not only supports Voice over IP (VoIP) services but also presence[1] and instant messaging.

The major advantages of `PJSIP` are the portability (the library was written with embedded systems in mind), small resource requirements and excellent documentation.

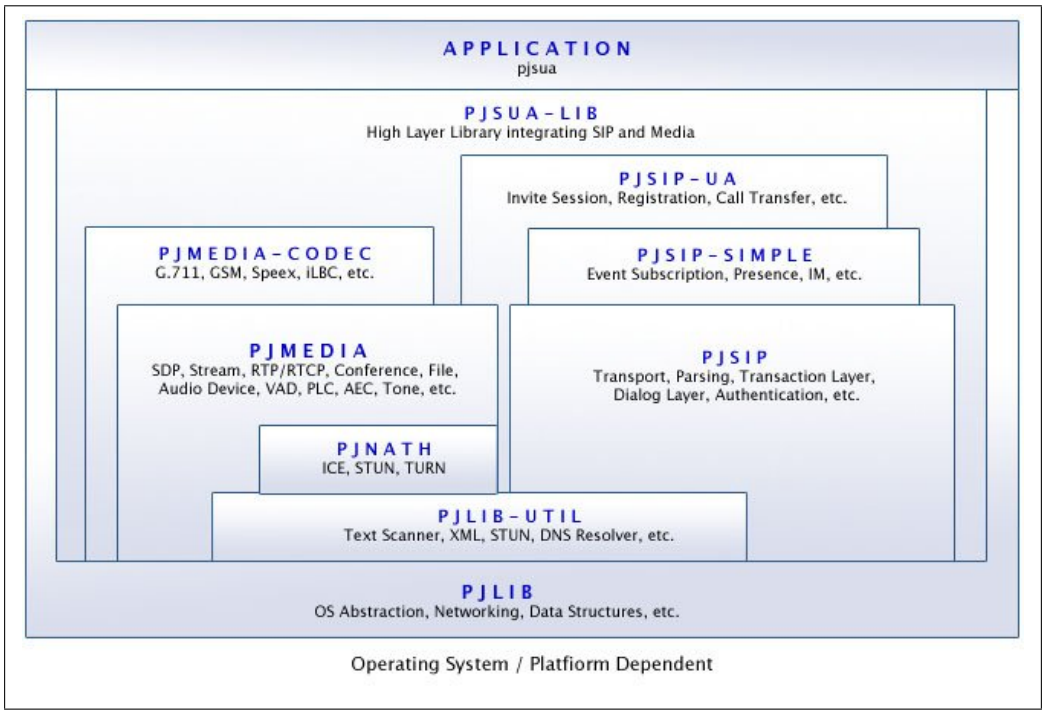

Figure 6.1: PJSIP Stack [PJS09]

The overall architecure of the library is depicted in figure 6.1. `PJSIP` is not a single library, there are different layers of libraries which implement all aspects of the SIP protocol.

[1] Presence is a SIP service that enables a user to determine the online status of other users.

PJLIB is at the bottom and contains basic functionalities. It is actually meant as a replacement for the standard C library.

PJLIB-UTIL implements utility functions like encryption algorithms, functions for text scanning and XML parsing as well as a DNS resolver. (see RFC 3489 [RWHM03]).

PJNATH includes utilities for Network Address Translation (NAT) (see RFC 3022 [SE01]). This covers implementations of Session Traversal Utilities for NAT (STUN) (RFC 3489 [RWHM03]), Traversal Using Relays around NAT (TURN) ([RMM08]) and Interactive Connectivity Establishment (ICE) ([Ros07]).

PJSIP core library realizes the core SIP protocol. The core library has been activly developed since 1999 and is considered very stable.

PJMEDIA and PJMEDIA-CODEC implement a complete media stack which is required for Voice over IP applications. This covers parsing of Session Description Protocol (SDP) ([HJ98]) messages and an implementation of the Realtime Transport Protocol (RTP) ([GSC+96]).

PJSIP-SIMPLE is the implementation of the SIP event and presence framework. For more information about the SIP event and presence framework see RFC 3265 [Roa02] and RFC 3856 [Ros04].

PJSIP-UA The SIP user agent library provides a high level abstraction for sending `INVITE` requests, user agent registration via the SIP `REGISTER` method and an implementation of RFC 3515 [Spa03] `SIP REFER` for call transfer.

PJSUA-SIMPLE is the highest abstraction layer of the PJSIP library stack. It is a wrapper around signaling and media functionalities. `PJSUA-SIMPLE` also provides bindings for the Python programming language, so implementing the SIP Client in C is not a requirement.

The complete `PJSIP` stack has already been ported to Linux, BSD, Mac OS, Solaris, Symbian and Windows. There is also a SIP Client for the Nintendo DS game console which uses the `PJSIP` framework.

Version 0.9 which was released in June 2008 also supports IPv6.

7 Reference Implementation

This chapter is describes the reference implementation of a working Linux distribution for the cm-x270. Section 7.1 "OpenEmbedded Environment" explains how to prepare the OpenEmbedded Toolchain for creating the required Linux kernel and the root filesystem images. The following section covers updating the toolchain if new packages are available. The next section "Customizing the Distribution" details the steps necessary for creating a custom application that should be built with the OpenEmbedded Toolchain but is not included in the standard OpenEmbedded distribution. "Building a minimal Image" and "Building a Distribution Image" (section 7.4 and 7.5) describe the necessary tasks for creating the actual images to boot the cm-x270. Following image creation, section 7.6 "Flashing the cm-x270" describes the deployment of the kernel and root filesystem on the cm-x270.

7.1 OpenEmbedded Environment

Creating the OpenEmbedded Environment for the toolchain is split into two major tasks: Installing the OpenEmbedded Task Executor BitBake, which is further explained in section 7.1.2, and cloning the OpenEmbedded Repositories containing all metadata files necessary for building the Linux distribution (section 7.1.3).

7.1.1 OpenEmbedded Directory Layout

OpenEmbedded has a complex directory layout. There are about 450,000 files in the OpenEmbedded root and its subdirectories. Table 7.1 provides an overview of the most important directories.

Directory	Description
`$HOME/oe/`	This is the root directory. All other directories in this table are relative to the root directory.
`bitbake/`	contains BitBake, the Task-Executor (section 5.2.1)
`build/`	configuration files, images and all other files OpenEmbedded creates when building a distribution
`build/tmp/staging/`	binary files OpenEmbedded creates during the build process
`build/tmp/work/`	extracted source code archives of packages to be built and temporary files that are required for package creation
`build/tmp/rootfs/`	template root filesystem. all files in this directory are included in the resulting filesystem image
`build/tmp/cross/`	cross compiler
`build/tmp/stamps/`	files in this directory tell BitBake which packages have already been built, in case of a restart
`build/tmp/deploy/`	ipkg package files
`build/conf/`	configuration files for OpenEmbedded (local.conf)
`build/tmp/deploy/... uclibc/images/cm-x270`	file system and kernel images
`local/`	local overlay 7.3.1
`openembedded/`	contains all OpenEmbedded metadata, i.e the BitBake recipes
`sources/`	Source code archives BitBake downloads before building the package

Table 7.1: Overview directory layout

7.1.2 BitBake

The OpenEmbedded Task-Executor BitBake is responsible for parsing the metadata files and creating the actual Linux distribution. It is hosted in a publicly accessible subversion repository. Subversion[1] is a source code version control system used by the BitBake developers. A Subversion client is required to download BitBake. We used the Debian Linux distribution on our development workstation and installing a Subversion (svn) client is very easy:

```
1 $ apt-get install subversion
```

[1] `http://subversion.tigris.org/`

Listing 7.1: Installing a Subversion client under Debian

Next we create the necessary directory structure and check out the Task Executor using the subversion client:

```
$ mkdir -p oe/build/conf
$ cd oe/
$ svn co svn://svn.berlios.de/bitbake/branches/bitbake-1.8/ bitbake
```

Listing 7.2: bitbake

The command in the first line of listing 7.2 creates required directories for BitBake. `build/conf` stores the BitBake configuration which is explained in section 7.1.4. After changing directory to `oe` the Subversion command line client `svn` is used to download version 1.8 of BitBake. Files downloaded by the Subversion client are stored in the directory `bitbake`.

7.1.3 OpenEmbedded BitBake Recipes

After installing the Task Executor BitBake, the next step is downloading the necessary OpenEmbedded metadata files. The OpenEmbedded developers decided to use a different type of version control system (`GIT`) to manage the metadata. This is a minor inconvenience when using OpenEmbedded. `GIT` is also used by the Linux kernel developers and can be installed under the Debian distribution via the command in listing 7.3.

```
$ apt-get install git-core
```

Listing 7.3: Installation of GIT under Debian GNU/Linux

After installing the `GIT` command line tool, the next step is to retrieve the files:

```
$ cd oe/
$ git clone git://git.openembedded.net/openembedded
```

Listing 7.4: Using GIT to download BitBake recipes

The clone command in listing 7.4 tells GIT to download a copy of the original repository into the directory `openembedded`.

7.1.4 BitBake Configuration

Before building the Linux distribution BitBake needs to be told where to find the metadata installed in section 7.1.3, the hardware platform the distribution is supposed to run on and the type of root filesystem to build.

First we copy a sample configuration file (see listing 7.5):

```
$ cp openembedded/conf/local.conf.sample build/conf/local.conf
$ vi build/conf/local.conf
```

Listing 7.5: copy and customize sample configuration

Line one in listing 7.5 copies the sample configuration into the directory

`build/conf/local.conf`

This is the standard directory where BitBake searches for configuration files. Line two opens the configuration file with the editor vi to customize the configuration.

Listing 7.6 contains the complete configuration file needed for the CompuLab cm-x270 to build a minimal image.

```
DL_DIR = "${OEDIR}/sources"
BBFILES := "${OEDIR}/openembedded/packages/*/*.bb ${OEDIR}/local/packages/*/*.bb"
BBINCLUDELOGS = "yes"

BBFILE_COLLECTIONS = "upstream overlay"

BBFILE_PATTERN_upstream = "^${OEDIR}/openembedded/"
BBFILE_PRIORITY_upstream = "10"

BBFILE_PATTERN_overlay = "^${OEDIR}/local/"
BBFILE_PRIORITY_overlay = "20"

MACHINE = "cm-x270"
DISTRO = "angstrom-2008.1"
ANGSTROMLIBC = "uclibc"
IMAGE_FSTYPES = "jffs2 tar cpio.gz"
```

Listing 7.6: local.conf

The first line in listing 7.6 tells BitBake where to store source code archives (.eg. tar.gz, tar.bz2 or .zip files) downloaded during the build process. In line two the variable `BBFILES` describes the location of metadata files. Because we also want to build custom packages we add the location `${OEDIR}/local/packages/*/*.bb` to the BitBake search path. `BBINCLUDELOGS` tells BitBake to include error messages that appeared during the build of a package in the build output.

Sometimes a package has to be customized further. It is possible to edit the files in the standard directory `openembedded`, but this has the disadvantage that all changes will be lost when an update is done (see also section 7.2). OpenEmbedded offers a solution to this problem called `OVERLAYS`. An `OVERLAY` tells BitBake where to search for package metadata that may provide a updated configuration in comparison to the standard package. Priorities tell BitBake which version of the package should be used during the build process. Line five defines two so-called BitBake collections named `upstream` and `overlay`. Each collection stores the necessary package metadata and a priority. `upstream` is the standard directory for BitBake files and `overlay` holds customized package data. The collection `overlay` has a higher priority (20) than the collection `upstream` so if there are two versions of a package, `upstream` is used.

Next BitBake has to be told for which type of machine the kernel and the root filesystem have to be built. This information is stored in the variable `MACHINE` in line 13. Because OpenEmbedded supports multiple Linux distributions, `DISTRO` selects angstrom-2008.1 (see section 5.2.2).

The option `ANGSTROMLIBC = "uclibc"` limits the size of the image. The minimal image has to be stored in NOR flash which is only 4MB in size. uLibc is a minimal standard C library. With the GNU C library the resulting image would be too large to fit into NOR flash.

Option `IMAGE_FSTYPES` selects the type of root filesystem BitBake should create after building all necessary packages. JFFS2 is the filesystem used for the cm-x270, but for testing purposes .tar and cpio.gz were also selected.

There are some environment variables that have to be set before executing BitBake. Listing 7.7 displays the file `oe/setup-env` which should be sourced with the command

```
. setup-env
```

before calling BitBake.

```
export OEDIR=$HOME/oe
export PATH=$OEDIR/bitbake/bin:$PATH
export BBPATH=$OEDIR/build:$OEDIR/openembedded
export http_proxy=http://murus.stderror.at:8080/
export ftp_proxy=http://murus.stderror.at:8080/
export HTTP_PROXY=http://murus.stderror.at:8080/
export FTP_PROXY=http://murus.stderror.at:8080/
```

Listing 7.7: BitBake Environment

Option `OEDIR` in Listing 7.7 specifies the root directory for OpenEmbedded. In line two `PATH` extends the standard search path of the shell used with the location of BitBake. `BBPATH` is used internally by BitBake. The variables `http_proxy` in line four and eight and `ftp_proxy` in line five / nine tell BitBake to use a proxy server when downloading source code archives.

7.2 Applying OpenEmbedded Updates

OpenEmbedded is a very dynamic project and updates to packages are available on a daily basis. Getting new files from the master repository is very straight forward:

```
$ cd oe
$ git pull git://git.openembedded.net/openembedded master
```

Listing 7.8: local.conf

Line one in listing 7.8 changes to the directory were the toolchain is located. The next line fetches updated metadata files from the git repository at `git.openembedded.net`.

Updating the environment is a necessary task and should be done on a regular basis. If the local build environment and the master repository are too out of sync, updating can become difficult because of changes to many packages.

As a last step the distribution and kernel images should be recompiled as described in section 7.5.

7.3 Customizing the Distribution

As explained in section 7.1.4 sometimes it is necessary to extend OpenEmbedded with custom packages. Our project had the requirement to include the SIP library `PJSIP` in

the distribution image and to build a custom kernel to support the USB wireless LAN stick described in section 4.2.5.

Section 7.3.1 explains how to build and include `PJSIP` in the image and section 7.3.2 details the required steps to build a custom kernel with the help of OpenEmbedded.

7.3.1 Custom Packages

This section describes how to build a custom package and how to include it in the distribution.

First create a directory structure that supports the package infrastructure. This is listed below:

```
mkdir -p local/conf local/classes \
         local/packages local/packages/images \
         local/packages/tasks
```

Now it is possible to create a new BitBake recipe that describes the package to be built. Listing 7.9 depicts the configuration file for building the SIP library `PJSIP`.

```
DESCRIPTION = "PJSIP Library"
AUTHOR = "pjsip.org"
HOMEPAGE = "http://pjsip.org"
SECTION = "libs"
MAINTAINER = "Toni Schmidbauer"
PRIORITY = "optional"
LICENSE = "GPL"

# where can we find the source package for downloading?
SRC_URI = "http://pjsip.org/release/0.8.0/pjproject-0.8.0.tar.gz"

# pjsip does not work with parallel make, so disable it
PARALLEL_MAKE = ""

# we just want the archive files included in our package
# headers and so on are not required for running our application
FILES_${PN} = "/usr/local/lib/lib*.a"

# pjsip uses ./configure aka autotools, so tell bitbake to use it
inherit autotools
```

Listing 7.9: pjproject_0_8_0.bb

Lines one to seven are standard variables that should be set for every new package. BitBake not only creates the distribution image but also one IPKG file for every package built. IPKG (Itsy Package Management System) is a package format especially designed for embedded Linux distributions. The variables in line one to seven are also used as metadata for the IPKG version of the package.

`SRC_URI` specifies the URI where the source code for the SIP library is located. BitBake downloads the specified tar.gz archive as a first step during the build process.

`PARALLEL_MAKE` is explicitly disabled because `PJSIP` does not support parallel make execution (e.g. make -j4)[2].

[2]Parallel make exection is used to compile source code files in parallel.

The regular expression[3] in line 17 specifies which files should be included in the image and in the resulting IPKG file. We are only interested in the library files themselves.

Because `PJSIP` needs GNU autotools during the build, we tell BitBake about it.

Now it is possible to start the build process for the new package with the command

```
make build-package-pjproject
```

If the build is successful we find the new package in

`build/tmp/deploy/uclibc/ipk/cm-x270`.

7.3.2 Custom Kernel

Building a custom kernel for the cm-x270 is basically the same process as building a custom package. The only difference is that there is no IPKG file for the kernel.

OpenEmbedded supports the CompuLab cm-x270 out of the box, but with an older kernel version. Because of the used wireless LAN USB stick it was necessary to upgrade the kernel to version 2.6.26.2 (see section 4.2.5).

The first step to build a custom kernel with the help of OpenEmbedded the first step is to add md5 and sha256 checksums of the kernel source package to the file `oe/openembedded/conf/checksum.ini`. OpenEmbedded compares the checksums of the downloaded file (in our case `linux-2.6.26.2.tar.bz2`) with the checksums stored in the .ini file.

```
$ cd oe/openembedded/conf/
$ cp checksum.ini checksum.ini.orig
```

Next add the following lines to `checksum.ini`:

```
[http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.26.2.tar.bz2]
md5=82472622bd26bed0054790ec3e60ee00
sha256=355ccb35ae53bc3835b7fff0eac7eee91d054063e6a832d48798d032b9d93377
```

Create the directory structure necessary for the custom kernel:

```
mkdir -p local/packages/linux-2.6.26.2/cm-x270
```

OpenEmbedded includes some patches for the standard Linux kernel to support the cm-x270. Copy the patches from the standard OpenEmbedded packages directory to the custom kernel directory:

[3] Actually it is a glob.

```
cd local/packages/linux-2.6.26.2/cm-x270
cp ~/oe/openembedded/packages/linux/linux-2.6.25/cm-x270/*patch .
```

The file `defconfig` in the directory `oe/openembedded/packages/linux/linux-2.6.25/cm-x270/` is the standard Linux kernel configuration used by OpenEmbedded. Because we are using a newer kernel version and we would like to include the driver for our wireless USB card, we have to create a new kernel configuration. This means

- downloading the Linux kernel we plan to use (2.6.26.2) into a temporary directory;
- unpacking the kernel source tree;
- copying the OpenEmbedded configuration for kernel 2.6.26.2 into our new source tree;
- customizing the configuration;
- copying the new configuration file back into the OpenEmbedded local tree.

Listing 7.10 lists the necessary steps:

```
$ mkdir ~/tmp && cd ~/tmp
$ wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.26.2.tar.bz2
$ tar jxf linux-2.6.26.2.tar.bz2
$ cd linux-2.6.26.2
$ cp ~/oe/openembedded/packages/linux/linux-2.6.25/cm-x270/defconfig .config
$ make ARCH=arm oldconfig

Answer questions, important is the ARM system type (PXA2xx/PXA3xx-based
(ARCH_PXA) and the target board (CompuLab CM-X270 modules)

$ make ARCH=arm menuconfig

Select necessary components to support the usb wireless adapter, and exit
the config utility

$ cp .config ~/oe/local/packages/linux/linux-2.6.26.2/defconfig
```

Listing 7.10: Configure Kernel 2.6.26.2

Appendix A lists the complete Linux kernel configuration used in this project.

7.4 Building a Minimal Image

The minimal filesystem image has to be flashed to NOR memory before deploying the standard linux image. It is not possible to flash the standard filesystem directly because the cm-x270 uses a proprietary format for the NAND flash. Linux has no support for reading and writing this format. Since the NOR memory is only 4MB there is a special BitBake target to build the minimal filesystem.

Listing 7.11 explains the build process:

```
$ cd oe/build/
$ source ../setup-env
$ bitbake minimal-image-with-mtd-utils
```

Listing 7.11: Building the minimal image

We start by changing to the build directory `oe/build` as shown in line one. Before calling BitBake it is necessary to set up some shell environment variables and this is done in line two. Now it is possible to execute BitBake and start the build process.

If the build finished successfully the filesystem image can be found in `build/tmp/deploy/uclibc/images/cm-x270`.

7.5 Building the Distribution Image

To build the distribution image we create a slightly different BitBake configuration file (listing 7.12).

```
1  DL_DIR = "${OEDIR}/sources"
   BBFILES := "${OEDIR}/openembedded/packages/*/*.bb ${OEDIR}/local/packages/*/*.bb"
3  BBINCLUDELOGS = "yes"

5  BBFILE_COLLECTIONS = "upstream overlay"

7  BBFILE_PATTERN_upstream = "^${OEDIR}/openembedded/"
   BBFILE_PRIORITY_upstream = "10"
9
   BBFILE_PATTERN_overlay = "^${OEDIR}/local/"
11 BBFILE_PRIORITY_overlay = "20"

13 MACHINE = "cm-x270"
   DISTRO = "angstrom-2008.1"
15 ANGSTROMLIBC = "glibc"
   IMAGE_FSTYPES = "jffs2 tar cpio.gz"
17
   DISTRO_EXTRA_RDEPENDS="pciutils pjproject"
```

Listing 7.12: local.conf

The main difference to the configuration file for our minimal image is the use of the GNU standard C library instead of uClibc. `ANGSTROMLIBC` tells BitBake to use this version of the C library.

Option `DISTRO_EXTRA_RDEPENDS` adds two additional packages to the distribution: `pciutils` which includes tools to get information of installed PCI devices and `pjproject` which is the SIP library used for the project. `pjproject` is also a local package because it is not included in the standard OpenEmbedded metadata files. For more information on how to build custom packages see section 7.3.1 “Customizing the Distribution”.

Listing 7.13 covers the necessary steps to start the build process:

```
$ cd oe/build/
2 $ source ../setup-env
  $ bitbake console-image
```

Listing 7.13: Building the distribution image

After a successful build the kernel and the distribution images can be found in `build/tmp/deploy/uclibc/images/cm-x270`.

7.6 Flashing the cm-x270

After building the minimal image in section 7.4 and the distribution image in section 7.5 we can now continue with flashing the images to the cm-x270.

We start with the minimal image because this is a prerequisite before deploying the distribution image. Section 7.6.2 "Distribution Image" covers this last step before the cm-x270 can finally boot into our custom built Linux distribution.

7.6.1 Minimal Image

After BitBake has successfully created the minimal image in section 7.4 it is located in the directory `oe/build/tmp/deploy/ulibc/images/cm-x270`. `minimalist-image-mtdutils-cm-x270.cpio.gz` is the image file name we are looking for.

The cm-x270 supports the trivial file transfer protocol (TFTP) (see also RFC 1350 [Sol92]) for downloading images to the embedded board. You need a TFTP server installed on a computer connected to the cm-x270 via a LAN[4]. Installing the `TFTP` package under Debian can be done using the command `apt-get install tftpd`. After installing the TFTP server copy the image to the directory `/srv/tftpd`. The TFTP server daemon serves all files in this location to TFTP clients, in our case the cm-x270.

Next create a serial connection to the cm-x270 to access the serial console. We used the program `kermit` for this purpose. Appendix B (section B) lists a configuration for kermit which connects via a USB serial dongle to the cm-x270.

After connecting the cm-x270 serial port to our USB dongle on the notebook, executing the command `kermit cm-x270.kermit` starts the serial console. Listing 7.14 depicts the output:

```
1  Connecting to /dev/ttyUSB0, speed 38400
    Escape character: Ctrl-\ (ASCII 28, FS): enabled
3  Type the escape character followed by C to get back,
   or followed by ? to see other options.
5  ----------------------------------------------------

7  Welcome to ARMmonitor running on Compulab CM-X270.
   Copyright Compulab 2002-2007 (c).
9
   Built at: Wed May 30 18:51:19 IDT 2007 on 89
11
   CM-X270 hardware configuration:
13 --------------------------------
   SDRAM size ........................... 128MB
15 NOR flash ............................ AMD or compatible, 4MB
   NAND flash ........................... Generic NAND, 512MB
17 PCI bridge ........................... present
   Ethernet on CORE ..................... present
19 Ethernet on BASE ..................... present
   AC'97 CODEC .......................... present
21 RTC .................................. present

23 ARMmon >
```

Listing 7.14: Connecting to the serial console with kermit

[4] Actually, we used a crossover cable.

The commands in listing 7.15 describe the process of configuring the cm-x270 and downloading the minimal image to the NOR flash:

```
1 ARMmon > setip ip 83.65.196.94
  ARMmon > setip mask 255.255.255.248
3 ARMmon > download ramdisk tftp minimalist-image-mtdutils-cm-x270.cpio.gz 83.65.196.92
  ARMmon > flash ramdisk
5 ARMmon > download kernel tftp zimage-2.6.26.2-r3-cm-x270.bin 83.65.196.92
  ARMmon > flash kernel
```

Listing 7.15: Downloading the minimal image to NOR flash

Line one and two in listing 7.15 configure the on-board Ethernet port of the cm-x270. Next we download the file `minimalist-image-mtdutils-cm-x270.cpio.gz` from the TFTP server to the embedded board. After a successful download we write the minimal image to the NOR flash with the command `flash ramdisk`. The cm-x270 boot-loader ARMmon knows at which address to store the image into NOR memory because we specify the target `ramdisk`. This initial ramdisk is used by the Linux kernel as a preliminary root filesystem when booting. This root filesystem provides all the utilities we need to format the NAND flash memory and to write the full distribution image into NAND. Line 5 downloads the Linux kernel from the TFTP server and line 6 writes the kernel image to NOR memory.

We are now able to boot the cm-x270 with our kernel and minimal ramdisk image. Listing 7.16 describes the necessary commands:

```
ARMmon > ramdisk on
2 ARMmon > bootos
```

Listing 7.16: Booting the cm-x270 into the minimal image

In line one we tell the boot-loader ARMmon to use an initial ramdisk when booting the kernel. Line two boots the Linux kernel.

7.6.2 Distribution Image

After deploying the Linux kernel and the minimal image onto the cm-x270 we are able to boot the system and reformat the NAND flash. The last step is deploying the distribution image and booting the system into our Linux distribution.

We boot the cm-x270 from NOR flash with the commands:

```
ARMmon > ramdisk on
2 ARMmon > bootos
```

The Linux kernel prints informational messages during the boot process and presents the login screen as depicted in listing 7.17.

```
 .-------.
2 |       |                  .-.
  |   |   |-----.-----.-----.| |   .----.-----.-----.
4 |       |     | __  |  ---'| '--.|  .-'|     |     |
  |   |   |  |  | ____|---  ||  --'|  |  |  '  | | | |
6 '---'---'--'--'--'__.  |-----''-----''--'  '-----'-'-'-'
                   _.'  |
8                  '---'

10 The Angstrom Distribution cm-x270 ttyS1
```

```
12 Angstrom 2008.1-test-20080417 cm-x270 ttyS1

14 cm-x270 login: root
   Alignment trap: login (559) PC=0x4000255c Instr=0xe1c62df8 Address=0x4000520c FSR 0x813
16 Apr 17 22:59:08 login[559]: root login  on 'ttyS1'Alignment trap: sh (559)
   PC=0x4000255c Instr=0xe1c62df8 Address=0x40005204 FSR 0x813
18

20 Alignment trap: sh (559) PC=0x4000255c Instr=0xe1c62df8 Address=0x40005314 FSR 0x813
   Alignment trap: sh (559) PC=0x40003b40 Instr=0xe1c42df8 Address=0x40005544 FSR 0x813
22 Alignment trap: id (563) PC=0x4000255c Instr=0xe1c62df8 Address=0x40005204 FSR 0x813
   Alignment trap: id (563) PC=0x4000255c Instr=0xe1c62df8 Address=0x40005314 FSR 0x813
24 Alignment trap: id (563) PC=0x40003b40 Instr=0xe1c42df8 Address=0x40005544 FSR 0x813
   root@cm-x270:~#
```

Listing 7.17: Linux Login

Login with the username `root`. The user `root` has no initial password so we get the standard shell prompt.

After executing commands (e.g. `ls`) the Linux kernel prints the following message to the system console,

```
1 Alignment trap: ls (564) PC=0x4000255c Instr=0xe1c62df8 Address=0x40005204 FSR 0x813
```

To prevent these kind of messages execute:

```
1 echo 2 > /proc/cpu/alignment
```

We are now ready to configure the Ethernet interface of the cm-x270 and start downloading the distribution image. Listing 7.18 includes the necessary commands.

```
1 root@cm-x270:~# ifconfig eth0 83.65.196.94
  root@cm-x270:~# tftp -g -r console-image-tw-cm-x270.jffs2 83.65.196.92
```

Listing 7.18: network configuration and image download

Line one in listing 7.18 configures the network interface eth0 with the IP address `83.65.196.94`. The following line executes the `tftp` command to download the distribution image with the name `console-image-tw-cm-x270.jffs2` from our TFTP server.

After downloading the image we are able to re-format the NAND flash memory and deploy the standard distribution image.

```
  root@cm-x270:~# flash_eraseall -j /dev/mtd3
2 Erasing 128 Kibyte @ 3fe0000 -- 99 % complete. Cleanmarker written at 3fe0000.
  root@cm-x270:~# nandwrite /dev/mtd3 console-image-tw-cm-x270.jffs2
```

Listing 7.19: re-format and writing the distribution image to the NAND flash

The first line in listing 7.19 re-formats the NAND flash memory on device `/dev/mtd3`. Line two writes the image file `console-image-tw-cm-x270.jffs2` to the same device.

After executing the command `nandwrite` we reboot the cm-x270 into the ARMmon boot-loader. Before booting the kernel with our newly flashed distribution image we have to disable the ramdisk otherwise the boot-loader would once again load the kernel with the minimal root filesystem. The command

```
ramdisk off
```

disables the ramdisk and `bootos` starts the kernel and loads the distribution image from NAND memory as it's root filesystem.

8 Conclusion and Future Work

This last section sums up the topics covered in this thesis and discusses the problems we faced during our work. It also describes possible future work.

We covered the difference between embedded systems and standard PC systems in the section "Fundemantals". Embedded systems is a broad field and the learning curve for newcomers is steep. The literature that is available on the subject only provides an overview (see [Hal07] and [Yag03]).

A well documented Open Source operating system and a platform used by many people make things easier. Nevertheless a lot of trial and error is necessary to get to a working solution.

We documented the process of creating our software stack in section "Reference Implementation". Even though we focused on one particular embedded system it should be possible to recreate the proposed software stack on another embedded platform. The only requirement is that the hardware is supported by the OpenEmbedded toolchain. Setting up the toolchain is easy however compiling and deploying the Linux distribution may be difficult.

A lot of time is required to understand the OpenEmbedded toolchain. It took us about four to five months to get a first prototype ready.

Nevertheless we think that our proposed solution is a good starting point for future work. Linux as an operating system has proven to be very stable, and `PJSIP` is very reliable and well-integrated into Linux. The documentation for `PJSIP` was an excellent help when integrating the library into the cm-x270.

8.1 Problems

Finding the right components (embedded board, toolchain and SIP library) is not an easy task. Many embedded boards are available on the market and the decision process of finding the most suitable for a particular task is difficult. We did not have to perform this task because using the cm-x270, was a requirement within the car to car communication project.

If we had to choose an embedded board we would consider the following issues to determine the requirements:

1. power consumption
2. CPU power necessary

3. required memory
4. type(s) of persistent storage (harddisks, flash)
5. size of persistent storage (gigabytes, megabytes, kilobytes)
6. environment the board should run in (e.g. temperature, humidity etc.)
7. operating system

Another problem we faced was the complexity of the OpenEmbedded toolchain. It is easy to setup the basic infrastructure but customizing and solving problems like compilation errors is hard. There is documentation available but it is scattered all over the Internet and most documention only contains a basic introduction.

For example when we applied the latest OpenEmbedded GIT updates the minimal image became too large (about 71MB). Because of time constraints it was not possible to find a solution to this problem before finishing this thesis. More investigation on how the root filesystem is built within OpenEmbedded is necessary. The build process of the root image is done by **many** BitBake recipes and to solve compilation problems it is necessary to understand each of them.

Knowledge of the Python[1] programming language is also necessary to better understand BitBake. We had no experience with Python so debugging BitBake was not easy.

To better understand BitBake recipes and all the possibilities they offer it is necessary to study all existing recipes. There is no available single complete documentation of BitBake recipes.

8.2 Future Work

In this last section we would like to give a perspective of future tasks within this project. Because of time constraints it was possible to implement all required features.

The basic software stack is working and it is now possible to start building a SIP application.

Nevertheless more knowledge about the OpenEmbedded toolchain is required to solve the problems we faced. The Linux distribution should be futher customized to include only the tools necessary to run the application.

It is advisable to run stress tests on the hardware to see how stable the stack is. A working application is required to test the stack. As a first step CPU and network stress tests should be run.

A process on how to develop and deploy applications to the embedded board should also be defined. Flashing the root filesystem after every application update (edit - compile - run cycle) is not feasible. For example OpenEmbedded offers the possibility to build a

[1] For more information about the Python programming language visit the official website at `http://www.python.org`

root filesystem that can be shared via NFS[2]. The cross compiled application can then be copied onto the shared filesystem.

[2]NFS is the so called Network File System. A network server provides a filesystem that is mountable form a network client. For more information about NFS see [Mic89], [CPS95] and [SCR+03].

Bibliography

[Coo98] Intel Cooperation. *Understanding the Flash Translation Layer (FTL) Specification.* Intel, 1998.

[CPS95] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification. RFC 1813, Internet Engineering Task Force, June 1995.

[GSC+96] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, Internet Engineering Task Force, January 1996.

[Hal07] Christopher Hallinan. *Embedded Linux Primer.* Prentice Hall, 2007.

[HJ98] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327, Internet Engineering Task Force, April 1998.

[Lev00] John R. Levine. *Linkers & Loaders.* Morgen Kaufmann Publishers, 2000.

[Mic89] Sun Microsystems. NFS: Network File System Protocol specification. RFC 1094, Internet Engineering Task Force, March 1989.

[MR91] John K. Ousterhout Mendel Rosenblum. *The Design and Implementation of a Log-Structured File System.* Technical report, Electrical Engineering and Computer Sciences, Computer Science Division, University of California, 1991.

[OSI07] OSI. The open source definition. `http://www.opensource.org/docs/osd`, 2007.

[PJS09] PJSIP. `http://www.pjsip.org/`, 2009.

[RMM08] J. Rosenberg, R. Mahy, and P. Matthews. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Internet-Draft draft-ietf-behave-turn-10, Internet Engineering Task Force, September 2008. Work in progress.

[Roa02] A. B. Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265, Internet Engineering Task Force, June 2002.

[Ros04] J. Rosenberg. A Presence Event Package for the Session Initiation Protocol (SIP). RFC 3856, Internet Engineering Task Force, August 2004.

[Ros07] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. Internet-Draft draft-ietf-mmusic-ice-19, Internet Engineering Task Force, October 2007. Work in progress.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[RWHM03] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, Internet Engineering Task Force, March 2003.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, Internet Engineering Task Force, July 2003.

[SCR+03] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network File System (NFS) version 4 Protocol. RFC 3530, Internet Engineering Task Force, April 2003.

[SE01] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022, Internet Engineering Task Force, January 2001.

[Sol92] K. Sollins. The TFTP Protocol (Revision 2). RFC 1350, Internet Engineering Task Force, July 1992.

[Spa03] R. Sparks. The Session Initiation Protocol (SIP) Refer Method. RFC 3515, Internet Engineering Task Force, April 2003.

[vH04] Kurt Hall / William von Hagen. *The Definitiv Guide to GCC.* Apress, 2004.

[Woo01] David Woodhouse. *JFFS : The Journalling Flash File System.* Red Hat, Inc., 2001.

[Yag03] Karim Yaghmour. *Building Embedded Linux Systems.* O'Reilly Media, 03 2003.

List of Figures

List of Tables

Listings

Glossary

BSD	Berkeley Software Distribution
CoM	Computer on Module
EEPROM	Electrically Erasable Programmable Read-Only Memory
FTL	Flash Translation Layer
ICE	Interactive Connectivity Establishment
IPKG	Itsy Package Managenment System
JFFS	Journaling Flash File System
LFS	Log Structured Filesystem
MMU	Memory Management Unit
MTD	Memory Technologie Devices
NAT	Network Address Translation
NFS	Network File System
NFTL	NAND Flash Translation Layer
OE	OpenEmbedded
OSI	Open Source Initiative
OSS	Open Source Software
RTP	Real-Time Transport Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
STUN	Session Traversal Utilities for NAT
SVN	Subversion
TFTP	Trivial File Transfer Protocol
TFTP	Trivial File Transfer Protocol
TURN	Traversal Using Relays around NAT
URI	Uniform Resource Identifier
VoIP	Voice over IP

A Appendix A
Linux 2.6.26.2 kernel config

```
#
# Automatically generated make config: don't edit
# Linux kernel version: 2.6.26.2
# Sat Aug 16 11:11:02 2008
#
CONFIG_ARM=y
CONFIG_SYS_SUPPORTS_APM_EMULATION=y
CONFIG_GENERIC_GPIO=y
CONFIG_GENERIC_TIME=y
CONFIG_GENERIC_CLOCKEVENTS=y
CONFIG_MMU=y
# CONFIG_NO_IOPORT is not set
CONFIG_GENERIC_HARDIRQS=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
CONFIG_HARDIRQS_SW_RESEND=y
CONFIG_GENERIC_IRQ_PROBE=y
CONFIG_RWSEM_GENERIC_SPINLOCK=y
# CONFIG_ARCH_HAS_ILOG2_U32 is not set
# CONFIG_ARCH_HAS_ILOG2_U64 is not set
CONFIG_GENERIC_HWEIGHT=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_ARCH_SUPPORTS_AOUT=y
CONFIG_ZONE_DMA=y
CONFIG_ARCH_MTD_XIP=y
CONFIG_VECTORS_BASE=0xffff0000
CONFIG_DEFCONFIG_LIST="/lib/modules/$UNAME_RELEASE/.config"

#
# General setup
#
CONFIG_EXPERIMENTAL=y
CONFIG_BROKEN_ON_SMP=y
CONFIG_LOCK_KERNEL=y
CONFIG_INIT_ENV_ARG_LIMIT=32
CONFIG_LOCALVERSION="-cm-x270"
# CONFIG_LOCALVERSION_AUTO is not set
CONFIG_SWAP=y
CONFIG_SYSVIPC=y
CONFIG_SYSVIPC_SYSCTL=y
# CONFIG_POSIX_MQUEUE is not set
# CONFIG_BSD_PROCESS_ACCT is not set
# CONFIG_TASKSTATS is not set
# CONFIG_AUDIT is not set
CONFIG_IKCONFIG=y
CONFIG_IKCONFIG_PROC=y
CONFIG_LOG_BUF_SHIFT=14
# CONFIG_CGROUPS is not set
# CONFIG_GROUP_SCHED is not set
CONFIG_SYSFS_DEPRECATED=y
CONFIG_SYSFS_DEPRECATED_V2=y
# CONFIG_RELAY is not set
# CONFIG_NAMESPACES is not set
CONFIG_BLK_DEV_INITRD=y
CONFIG_INITRAMFS_SOURCE=""
CONFIG_CC_OPTIMIZE_FOR_SIZE=y
CONFIG_SYSCTL=y
CONFIG_EMBEDDED=y
CONFIG_UID16=y
CONFIG_SYSCTL_SYSCALL=y
CONFIG_SYSCTL_SYSCALL_CHECK=y
CONFIG_KALLSYMS=y
# CONFIG_KALLSYMS_EXTRA_PASS is not set
CONFIG_HOTPLUG=y
CONFIG_PRINTK=y
CONFIG_BUG=y
CONFIG_ELF_CORE=y
CONFIG_COMPAT_BRK=y
CONFIG_BASE_FULL=y
CONFIG_FUTEX=y
CONFIG_ANON_INODES=y
```

```
CONFIG_EPOLL=y
CONFIG_SIGNALFD=y
CONFIG_TIMERFD=y
CONFIG_EVENTFD=y
CONFIG_SHMEM=y
CONFIG_VM_EVENT_COUNTERS=y
CONFIG_SLAB=y
# CONFIG_SLUB is not set
# CONFIG_SLOB is not set
# CONFIG_PROFILING is not set
# CONFIG_MARKERS is not set
CONFIG_HAVE_OPROFILE=y
# CONFIG_KPROBES is not set
CONFIG_HAVE_KPROBES=y
CONFIG_HAVE_KRETPROBES=y
# CONFIG_HAVE_DMA_ATTRS is not set
CONFIG_PROC_PAGE_MONITOR=y
CONFIG_SLABINFO=y
CONFIG_RT_MUTEXES=y
# CONFIG_TINY_SHMEM is not set
CONFIG_BASE_SMALL=0
CONFIG_MODULES=y
# CONFIG_MODULE_FORCE_LOAD is not set
CONFIG_MODULE_UNLOAD=y
# CONFIG_MODULE_FORCE_UNLOAD is not set
# CONFIG_MODVERSIONS is not set
# CONFIG_MODULE_SRCVERSION_ALL is not set
CONFIG_KMOD=y
CONFIG_BLOCK=y
# CONFIG_LBD is not set
# CONFIG_BLK_DEV_IO_TRACE is not set
# CONFIG_LSF is not set
# CONFIG_BLK_DEV_BSG is not set

#
# IO Schedulers
#
CONFIG_IOSCHED_NOOP=y
CONFIG_IOSCHED_AS=y
CONFIG_IOSCHED_DEADLINE=y
CONFIG_IOSCHED_CFQ=y
# CONFIG_DEFAULT_AS is not set
# CONFIG_DEFAULT_DEADLINE is not set
CONFIG_DEFAULT_CFQ=y
# CONFIG_DEFAULT_NOOP is not set
CONFIG_DEFAULT_IOSCHED="cfq"
CONFIG_CLASSIC_RCU=y

#
# System Type
#
# CONFIG_ARCH_AAEC2000 is not set
# CONFIG_ARCH_INTEGRATOR is not set
# CONFIG_ARCH_REALVIEW is not set
# CONFIG_ARCH_VERSATILE is not set
# CONFIG_ARCH_AT91 is not set
# CONFIG_ARCH_CLPS7500 is not set
# CONFIG_ARCH_CLPS711X is not set
# CONFIG_ARCH_CO285 is not set
# CONFIG_ARCH_EBSA110 is not set
# CONFIG_ARCH_EP93XX is not set
# CONFIG_ARCH_FOOTBRIDGE is not set
# CONFIG_ARCH_NETX is not set
# CONFIG_ARCH_H720X is not set
# CONFIG_ARCH_IMX is not set
# CONFIG_ARCH_IOP13XX is not set
# CONFIG_ARCH_IOP32X is not set
# CONFIG_ARCH_IOP33X is not set
# CONFIG_ARCH_IXP23XX is not set
# CONFIG_ARCH_IXP2000 is not set
# CONFIG_ARCH_IXP4XX is not set
# CONFIG_ARCH_L7200 is not set
# CONFIG_ARCH_KS8695 is not set
# CONFIG_ARCH_NS9XXX is not set
# CONFIG_ARCH_MXC is not set
# CONFIG_ARCH_ORION5X is not set
# CONFIG_ARCH_PNX4008 is not set
CONFIG_ARCH_PXA=y
# CONFIG_ARCH_RPC is not set
# CONFIG_ARCH_SA1100 is not set
# CONFIG_ARCH_S3C2410 is not set
# CONFIG_ARCH_SHARK is not set
# CONFIG_ARCH_LH7A40X is not set
# CONFIG_ARCH_DAVINCI is not set
# CONFIG_ARCH_OMAP is not set
# CONFIG_ARCH_MSM7X00A is not set

#
# Intel PXA2xx/PXA3xx Implementations
#
```

```
# CONFIG_ARCH_GUMSTIX is not set
# CONFIG_ARCH_LUBBOCK is not set
# CONFIG_MACH_LOGICPD_PXA270 is not set
# CONFIG_MACH_MAINSTONE is not set
# CONFIG_ARCH_PXA_IDP is not set
# CONFIG_PXA_SHARPSL is not set
# CONFIG_ARCH_PXA_ESERIES is not set
# CONFIG_MACH_TRIZEPS4 is not set
# CONFIG_MACH_EM_X270 is not set
# CONFIG_MACH_COLIBRI is not set
# CONFIG_MACH_ZYLONITE is not set
# CONFIG_MACH_LITTLETON is not set
CONFIG_MACH_ARMCORE=y
# CONFIG_MACH_MAGICIAN is not set
# CONFIG_MACH_PCM027 is not set
CONFIG_PXA27x=y

#
# Boot options
#

#
# Power management
#

#
# Processor Type
#
CONFIG_CPU_32=y
CONFIG_CPU_XSCALE=y
CONFIG_CPU_32v5=y
CONFIG_CPU_ABRT_EV5T=y
CONFIG_CPU_PABRT_NOIFAR=y
CONFIG_CPU_CACHE_VIVT=y
CONFIG_CPU_TLB_V4WBI=y
CONFIG_CPU_CP15=y
CONFIG_CPU_CP15_MMU=y

#
# Processor Features
#
CONFIG_ARM_THUMB=y
# CONFIG_CPU_DCACHE_DISABLE is not set
# CONFIG_OUTER_CACHE is not set
CONFIG_IWMMXT=y
CONFIG_XSCALE_PMU=y

#
# Bus support
#
# CONFIG_PCI is not set
# CONFIG_PCI_SYSCALL is not set
# CONFIG_ARCH_SUPPORTS_MSI is not set
# CONFIG_PCCARD is not set

#
# Kernel Features
#
CONFIG_TICK_ONESHOT=y
# CONFIG_NO_HZ is not set
CONFIG_HIGH_RES_TIMERS=y
CONFIG_GENERIC_CLOCKEVENTS_BUILD=y
CONFIG_PREEMPT=y
CONFIG_HZ=100
CONFIG_AEABI=y
CONFIG_OABI_COMPAT=y
# CONFIG_ARCH_DISCONTIGMEM_ENABLE is not set
CONFIG_SELECT_MEMORY_MODEL=y
CONFIG_FLATMEM_MANUAL=y
# CONFIG_DISCONTIGMEM_MANUAL is not set
# CONFIG_SPARSEMEM_MANUAL is not set
CONFIG_FLATMEM=y
CONFIG_FLAT_NODE_MEM_MAP=y
# CONFIG_SPARSEMEM_STATIC is not set
# CONFIG_SPARSEMEM_VMEMMAP_ENABLE is not set
CONFIG_PAGEFLAGS_EXTENDED=y
CONFIG_SPLIT_PTLOCK_CPUS=4096
# CONFIG_RESOURCES_64BIT is not set
CONFIG_ZONE_DMA_FLAG=1
CONFIG_BOUNCE=y
CONFIG_VIRT_TO_BUS=y
CONFIG_ALIGNMENT_TRAP=y

#
# Boot options
#
CONFIG_ZBOOT_ROM_TEXT=0x0
CONFIG_ZBOOT_ROM_BSS=0x0
CONFIG_CMDLINE="console=ttyS1,38400 monitor=1 mem=64M\
  mtdparts=physmap-flash.0:256k(boot)ro,0x180000(kernel),\
```

```
  -(root);cm-x270-nand:64m(app),-(data) rdinit=/sbin/init root=mtd3 rootfstype=jffs2"
# CONFIG_XIP_KERNEL is not set
# CONFIG_KEXEC is not set

#
# CPU Frequency scaling
#
# CONFIG_CPU_FREQ is not set

#
# Floating point emulation
#

#
# At least one emulation must be selected
#
# CONFIG_FPE_NWFPE is not set
# CONFIG_FPE_FASTFPE is not set

#
# Userspace binary formats
#
CONFIG_BINFMT_ELF=y
# CONFIG_BINFMT_AOUT is not set
# CONFIG_BINFMT_MISC is not set

#
# Power management options
#
# CONFIG_PM is not set
CONFIG_ARCH_SUSPEND_POSSIBLE=y

#
# Networking
#
CONFIG_NET=y

#
# Networking options
#
CONFIG_PACKET=y
# CONFIG_PACKET_MMAP is not set
CONFIG_UNIX=y
CONFIG_XFRM=y
# CONFIG_XFRM_USER is not set
# CONFIG_XFRM_SUB_POLICY is not set
# CONFIG_XFRM_MIGRATE is not set
# CONFIG_XFRM_STATISTICS is not set
# CONFIG_NET_KEY is not set
CONFIG_INET=y
# CONFIG_IP_MULTICAST is not set
# CONFIG_IP_ADVANCED_ROUTER is not set
CONFIG_IP_FIB_HASH=y
CONFIG_IP_PNP=y
CONFIG_IP_PNP_DHCP=y
CONFIG_IP_PNP_BOOTP=y
# CONFIG_IP_PNP_RARP is not set
# CONFIG_NET_IPIP is not set
# CONFIG_NET_IPGRE is not set
# CONFIG_ARPD is not set
# CONFIG_SYN_COOKIES is not set
# CONFIG_INET_AH is not set
# CONFIG_INET_ESP is not set
# CONFIG_INET_IPCOMP is not set
# CONFIG_INET_XFRM_TUNNEL is not set
# CONFIG_INET_TUNNEL is not set
CONFIG_INET_XFRM_MODE_TRANSPORT=y
CONFIG_INET_XFRM_MODE_TUNNEL=y
CONFIG_INET_XFRM_MODE_BEET=y
# CONFIG_INET_LRO is not set
CONFIG_INET_DIAG=y
CONFIG_INET_TCP_DIAG=y
# CONFIG_TCP_CONG_ADVANCED is not set
CONFIG_TCP_CONG_CUBIC=y
CONFIG_DEFAULT_TCP_CONG="cubic"
# CONFIG_TCP_MD5SIG is not set
# CONFIG_IPV6 is not set
# CONFIG_NETWORK_SECMARK is not set
# CONFIG_NETFILTER is not set
# CONFIG_IP_DCCP is not set
# CONFIG_IP_SCTP is not set
# CONFIG_TIPC is not set
# CONFIG_ATM is not set
# CONFIG_BRIDGE is not set
# CONFIG_VLAN_8021Q is not set
# CONFIG_DECNET is not set
# CONFIG_LLC2 is not set
# CONFIG_IPX is not set
# CONFIG_ATALK is not set
# CONFIG_X25 is not set
```

```
# CONFIG_LAPB is not set
# CONFIG_ECONET is not set
# CONFIG_WAN_ROUTER is not set
# CONFIG_NET_SCHED is not set
CONFIG_NET_SCH_FIFO=y

#
# Network testing
#
# CONFIG_NET_PKTGEN is not set
# CONFIG_HAMRADIO is not set
# CONFIG_CAN is not set
# CONFIG_IRDA is not set
# CONFIG_BT is not set
# CONFIG_AF_RXRPC is not set

#
# Wireless
#
CONFIG_CFG80211=y
CONFIG_NL80211=y
CONFIG_WIRELESS_EXT=y
CONFIG_MAC80211=y

#
# Rate control algorithm selection
#
CONFIG_MAC80211_RC_DEFAULT_PID=y
# CONFIG_MAC80211_RC_DEFAULT_NONE is not set

#
# Selecting 'y' for an algorithm will
#

#
# build the algorithm into mac80211.
#
CONFIG_MAC80211_RC_DEFAULT="pid"
CONFIG_MAC80211_RC_PID=y
# CONFIG_MAC80211_MESH is not set
# CONFIG_MAC80211_LEDS is not set
# CONFIG_MAC80211_DEBUG_PACKET_ALIGNMENT is not set
# CONFIG_MAC80211_DEBUG is not set
# CONFIG_IEEE80211 is not set
# CONFIG_RFKILL is not set
# CONFIG_NET_9P is not set

#
# Device Drivers
#

#
# Generic Driver Options
#
CONFIG_UEVENT_HELPER_PATH="/sbin/hotplug"
CONFIG_STANDALONE=y
CONFIG_PREVENT_FIRMWARE_BUILD=y
CONFIG_FW_LOADER=y
# CONFIG_SYS_HYPERVISOR is not set
# CONFIG_CONNECTOR is not set
CONFIG_MTD=y
# CONFIG_MTD_DEBUG is not set
# CONFIG_MTD_CONCAT is not set
CONFIG_MTD_PARTITIONS=y
# CONFIG_MTD_REDBOOT_PARTS is not set
CONFIG_MTD_CMDLINE_PARTS=y
# CONFIG_MTD_AFS_PARTS is not set
# CONFIG_MTD_AR7_PARTS is not set

#
# User Modules And Translation Layers
#
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLKDEVS=y
CONFIG_MTD_BLOCK=y
# CONFIG_FTL is not set
# CONFIG_NFTL is not set
# CONFIG_INFTL is not set
# CONFIG_RFD_FTL is not set
# CONFIG_SSFDC is not set
# CONFIG_MTD_OOPS is not set

#
# RAM/ROM/Flash chip drivers
#
CONFIG_MTD_CFI=y
# CONFIG_MTD_JEDECPROBE is not set
CONFIG_MTD_GEN_PROBE=y
# CONFIG_MTD_CFI_ADV_OPTIONS is not set
CONFIG_MTD_MAP_BANK_WIDTH_1=y
```

```
CONFIG_MTD_MAP_BANK_WIDTH_2=y
CONFIG_MTD_MAP_BANK_WIDTH_4=y
# CONFIG_MTD_MAP_BANK_WIDTH_8 is not set
# CONFIG_MTD_MAP_BANK_WIDTH_16 is not set
# CONFIG_MTD_MAP_BANK_WIDTH_32 is not set
CONFIG_MTD_CFI_I1=y
CONFIG_MTD_CFI_I2=y
# CONFIG_MTD_CFI_I4 is not set
# CONFIG_MTD_CFI_I8 is not set
# CONFIG_MTD_CFI_INTELEXT is not set
CONFIG_MTD_CFI_AMDSTD=y
# CONFIG_MTD_CFI_STAA is not set
CONFIG_MTD_CFI_UTIL=y
# CONFIG_MTD_RAM is not set
# CONFIG_MTD_ROM is not set
# CONFIG_MTD_ABSENT is not set
# CONFIG_MTD_XIP is not set

#
# Mapping drivers for chip access
#
# CONFIG_MTD_COMPLEX_MAPPINGS is not set
CONFIG_MTD_PHYSMAP=y
CONFIG_MTD_PHYSMAP_START=0x0
CONFIG_MTD_PHYSMAP_LEN=0x400000
CONFIG_MTD_PHYSMAP_BANKWIDTH=2
# CONFIG_MTD_ARM_INTEGRATOR is not set
# CONFIG_MTD_SHARP_SL is not set
# CONFIG_MTD_PLATRAM is not set

#
# Self-contained MTD device drivers
#
# CONFIG_MTD_SLRAM is not set
# CONFIG_MTD_PHRAM is not set
# CONFIG_MTD_MTDRAM is not set
# CONFIG_MTD_BLOCK2MTD is not set

#
# Disk-On-Chip Device Drivers
#
# CONFIG_MTD_DOC2000 is not set
# CONFIG_MTD_DOC2001 is not set
# CONFIG_MTD_DOC2001PLUS is not set
CONFIG_MTD_NAND=y
# CONFIG_MTD_NAND_VERIFY_WRITE is not set
# CONFIG_MTD_NAND_ECC_SMC is not set
# CONFIG_MTD_NAND_MUSEUM_IDS is not set
# CONFIG_MTD_NAND_H1900 is not set
CONFIG_MTD_NAND_IDS=y
# CONFIG_MTD_NAND_DISKONCHIP is not set
# CONFIG_MTD_NAND_SHARPSL is not set
CONFIG_MTD_NAND_CM_X270=y
# CONFIG_MTD_NAND_NANDSIM is not set
# CONFIG_MTD_NAND_PLATFORM is not set
# CONFIG_MTD_ALAUDA is not set
# CONFIG_MTD_ONENAND is not set

#
# UBI - Unsorted block images
#
# CONFIG_MTD_UBI is not set
# CONFIG_PARPORT is not set
CONFIG_BLK_DEV=y
# CONFIG_BLK_DEV_COW_COMMON is not set
CONFIG_BLK_DEV_LOOP=y
# CONFIG_BLK_DEV_CRYPTOLOOP is not set
# CONFIG_BLK_DEV_NBD is not set
# CONFIG_BLK_DEV_UB is not set
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=16
CONFIG_BLK_DEV_RAM_SIZE=4096
# CONFIG_BLK_DEV_XIP is not set
# CONFIG_CDROM_PKTCDVD is not set
# CONFIG_ATA_OVER_ETH is not set
CONFIG_MISC_DEVICES=y
# CONFIG_EEPROM_93CX6 is not set
# CONFIG_ENCLOSURE_SERVICES is not set
CONFIG_HAVE_IDE=y
# CONFIG_IDE is not set

#
# SCSI device support
#
# CONFIG_RAID_ATTRS is not set
CONFIG_SCSI=y
CONFIG_SCSI_DMA=y
# CONFIG_SCSI_TGT is not set
# CONFIG_SCSI_NETLINK is not set
CONFIG_SCSI_PROC_FS=y
```

```
#
# SCSI support type (disk, tape, CD-ROM)
#
CONFIG_BLK_DEV_SD=y
# CONFIG_CHR_DEV_ST is not set
# CONFIG_CHR_DEV_OSST is not set
# CONFIG_BLK_DEV_SR is not set
# CONFIG_CHR_DEV_SG is not set
# CONFIG_CHR_DEV_SCH is not set

#
# Some SCSI devices (e.g. CD jukebox) support multiple LUNs
#
# CONFIG_SCSI_MULTI_LUN is not set
# CONFIG_SCSI_CONSTANTS is not set
# CONFIG_SCSI_LOGGING is not set
# CONFIG_SCSI_SCAN_ASYNC is not set
CONFIG_SCSI_WAIT_SCAN=m

#
# SCSI Transports
#
# CONFIG_SCSI_SPI_ATTRS is not set
# CONFIG_SCSI_FC_ATTRS is not set
# CONFIG_SCSI_ISCSI_ATTRS is not set
# CONFIG_SCSI_SAS_LIBSAS is not set
# CONFIG_SCSI_SRP_ATTRS is not set
CONFIG_SCSI_LOWLEVEL=y
# CONFIG_ISCSI_TCP is not set
# CONFIG_SCSI_DEBUG is not set
# CONFIG_ATA is not set
# CONFIG_MD is not set
CONFIG_NETDEVICES=y
# CONFIG_NETDEVICES_MULTIQUEUE is not set
# CONFIG_DUMMY is not set
# CONFIG_BONDING is not set
# CONFIG_MACVLAN is not set
# CONFIG_EQUALIZER is not set
# CONFIG_TUN is not set
# CONFIG_VETH is not set
# CONFIG_PHYLIB is not set
CONFIG_NET_ETHERNET=y
CONFIG_MII=y
# CONFIG_AX88796 is not set
# CONFIG_SMC91X is not set
CONFIG_DM9000=y
CONFIG_DM9000_DEBUGLEVEL=4
# CONFIG_SMC911X is not set
# CONFIG_IBM_NEW_EMAC_ZMII is not set
# CONFIG_IBM_NEW_EMAC_RGMII is not set
# CONFIG_IBM_NEW_EMAC_TAH is not set
# CONFIG_IBM_NEW_EMAC_EMAC4 is not set
# CONFIG_B44 is not set
# CONFIG_NETDEV_1000 is not set
# CONFIG_NETDEV_10000 is not set

#
# Wireless LAN
#
# CONFIG_WLAN_PRE80211 is not set
CONFIG_WLAN_80211=y
# CONFIG_LIBERTAS is not set
# CONFIG_USB_ZD1201 is not set
# CONFIG_USB_NET_RNDIS_WLAN is not set
# CONFIG_RTL8187 is not set
# CONFIG_P54_COMMON is not set
# CONFIG_IWLWIFI_LEDS is not set
# CONFIG_HOSTAP is not set
# CONFIG_B43 is not set
# CONFIG_B43LEGACY is not set
# CONFIG_ZD1211RW is not set
CONFIG_RT2X00=y
CONFIG_RT2X00_LIB=y
CONFIG_RT2X00_LIB_USB=y
CONFIG_RT2X00_LIB_FIRMWARE=y
# CONFIG_RT2500USB is not set
CONFIG_RT73USB=y
# CONFIG_RT73USB_LEDS is not set
# CONFIG_RT2X00_DEBUG is not set

#
# USB Network Adapters
#
CONFIG_USB_CATC=m
CONFIG_USB_KAWETH=m
CONFIG_USB_PEGASUS=m
CONFIG_USB_RTL8150=m
CONFIG_USB_USBNET=m
CONFIG_USB_NET_AX8817X=m
```

```
CONFIG_USB_NET_CDCETHER=m
# CONFIG_USB_NET_DM9601 is not set
CONFIG_USB_NET_GL620A=m
CONFIG_USB_NET_NET1080=m
CONFIG_USB_NET_PLUSB=m
CONFIG_USB_NET_MCS7830=m
# CONFIG_USB_NET_RNDIS_HOST is not set
# CONFIG_USB_NET_CDC_SUBSET is not set
# CONFIG_USB_NET_ZAURUS is not set
# CONFIG_WAN is not set
# CONFIG_PPP is not set
# CONFIG_SLIP is not set
# CONFIG_NETCONSOLE is not set
# CONFIG_NETPOLL is not set
# CONFIG_NET_POLL_CONTROLLER is not set
# CONFIG_ISDN is not set

#
# Input device support
#
CONFIG_INPUT=y
# CONFIG_INPUT_FF_MEMLESS is not set
# CONFIG_INPUT_POLLDEV is not set

#
# Userland interfaces
#
CONFIG_INPUT_MOUSEDEV=y
CONFIG_INPUT_MOUSEDEV_PSAUX=y
CONFIG_INPUT_MOUSEDEV_SCREEN_X=1024
CONFIG_INPUT_MOUSEDEV_SCREEN_Y=768
# CONFIG_INPUT_JOYDEV is not set
CONFIG_INPUT_EVDEV=m
# CONFIG_INPUT_EVBUG is not set

#
# Input Device Drivers
#
CONFIG_INPUT_KEYBOARD=y
CONFIG_KEYBOARD_ATKBD=y
# CONFIG_KEYBOARD_SUNKBD is not set
# CONFIG_KEYBOARD_LKKBD is not set
# CONFIG_KEYBOARD_XTKBD is not set
# CONFIG_KEYBOARD_NEWTON is not set
# CONFIG_KEYBOARD_STOWAWAY is not set
# CONFIG_KEYBOARD_PXA27x is not set
# CONFIG_KEYBOARD_GPIO is not set
CONFIG_INPUT_MOUSE=y
CONFIG_MOUSE_PS2=y
# CONFIG_MOUSE_PS2_ALPS is not set
# CONFIG_MOUSE_PS2_LOGIPS2PP is not set
# CONFIG_MOUSE_PS2_SYNAPTICS is not set
# CONFIG_MOUSE_PS2_LIFEBOOK is not set
# CONFIG_MOUSE_PS2_TRACKPOINT is not set
# CONFIG_MOUSE_PS2_TOUCHKIT is not set
# CONFIG_MOUSE_SERIAL is not set
# CONFIG_MOUSE_APPLETOUCH is not set
# CONFIG_MOUSE_VSXXXAA is not set
# CONFIG_MOUSE_GPIO is not set
# CONFIG_INPUT_JOYSTICK is not set
# CONFIG_INPUT_TABLET is not set
CONFIG_INPUT_TOUCHSCREEN=y
# CONFIG_TOUCHSCREEN_FUJITSU is not set
# CONFIG_TOUCHSCREEN_GUNZE is not set
# CONFIG_TOUCHSCREEN_ELO is not set
# CONFIG_TOUCHSCREEN_MTOUCH is not set
# CONFIG_TOUCHSCREEN_MK712 is not set
# CONFIG_TOUCHSCREEN_PENMOUNT is not set
# CONFIG_TOUCHSCREEN_TOUCHRIGHT is not set
# CONFIG_TOUCHSCREEN_TOUCHWIN is not set
CONFIG_TOUCHSCREEN_UCB1400=m
# CONFIG_TOUCHSCREEN_WM97XX is not set
# CONFIG_TOUCHSCREEN_USB_COMPOSITE is not set
# CONFIG_INPUT_MISC is not set

#
# Hardware I/O ports
#
CONFIG_SERIO=y
CONFIG_SERIO_SERPORT=y
CONFIG_SERIO_LIBPS2=y
# CONFIG_SERIO_RAW is not set
# CONFIG_GAMEPORT is not set

#
# Character devices
#
CONFIG_VT=y
CONFIG_VT_CONSOLE=y
CONFIG_HW_CONSOLE=y
```

```
# CONFIG_VT_HW_CONSOLE_BINDING is not set
CONFIG_DEVKMEM=y
# CONFIG_SERIAL_NONSTANDARD is not set

#
# Serial drivers
#
# CONFIG_SERIAL_8250 is not set

#
# Non-8250 serial port support
#
CONFIG_SERIAL_PXA=y
CONFIG_SERIAL_PXA_CONSOLE=y
CONFIG_SERIAL_CORE=y
CONFIG_SERIAL_CORE_CONSOLE=y
CONFIG_UNIX98_PTYS=y
# CONFIG_LEGACY_PTYS is not set
# CONFIG_IPMI_HANDLER is not set
CONFIG_HW_RANDOM=y
# CONFIG_NVRAM is not set
# CONFIG_R3964 is not set
# CONFIG_RAW_DRIVER is not set
# CONFIG_TCG_TPM is not set
CONFIG_I2C=m
CONFIG_I2C_BOARDINFO=y
CONFIG_I2C_CHARDEV=m

#
# I2C Hardware Bus support
#
# CONFIG_I2C_GPIO is not set
CONFIG_I2C_PXA=m
# CONFIG_I2C_PXA_SLAVE is not set
# CONFIG_I2C_OCORES is not set
# CONFIG_I2C_PARPORT_LIGHT is not set
# CONFIG_I2C_SIMTEC is not set
# CONFIG_I2C_TAOS_EVM is not set
# CONFIG_I2C_STUB is not set
# CONFIG_I2C_TINY_USB is not set
# CONFIG_I2C_PCA_PLATFORM is not set

#
# Miscellaneous I2C Chip support
#
# CONFIG_DS1682 is not set
# CONFIG_SENSORS_EEPROM is not set
# CONFIG_SENSORS_PCF8574 is not set
# CONFIG_PCF8575 is not set
# CONFIG_SENSORS_PCF8591 is not set
# CONFIG_TPS65010 is not set
# CONFIG_SENSORS_MAX6875 is not set
# CONFIG_SENSORS_TSL2550 is not set
# CONFIG_I2C_DEBUG_CORE is not set
# CONFIG_I2C_DEBUG_ALGO is not set
# CONFIG_I2C_DEBUG_BUS is not set
# CONFIG_I2C_DEBUG_CHIP is not set
# CONFIG_SPI is not set
CONFIG_HAVE_GPIO_LIB=y

#
# GPIO Support
#

#
# I2C GPIO expanders:
#
# CONFIG_GPIO_PCA953X is not set
# CONFIG_GPIO_PCF857X is not set

#
# SPI GPIO expanders:
#
# CONFIG_W1 is not set
# CONFIG_POWER_SUPPLY is not set
CONFIG_HWMON=y
# CONFIG_HWMON_VID is not set
# CONFIG_SENSORS_AD7418 is not set
# CONFIG_SENSORS_ADM1021 is not set
# CONFIG_SENSORS_ADM1025 is not set
# CONFIG_SENSORS_ADM1026 is not set
# CONFIG_SENSORS_ADM1029 is not set
# CONFIG_SENSORS_ADM1031 is not set
# CONFIG_SENSORS_ADM9240 is not set
# CONFIG_SENSORS_ADT7470 is not set
# CONFIG_SENSORS_ADT7473 is not set
# CONFIG_SENSORS_ATXP1 is not set
# CONFIG_SENSORS_DS1621 is not set
# CONFIG_SENSORS_F71805F is not set
# CONFIG_SENSORS_F71882FG is not set
```

```
# CONFIG_SENSORS_F75375S is not set
# CONFIG_SENSORS_GL518SM is not set
# CONFIG_SENSORS_GL520SM is not set
# CONFIG_SENSORS_IT87 is not set
# CONFIG_SENSORS_LM63 is not set
# CONFIG_SENSORS_LM75 is not set
# CONFIG_SENSORS_LM77 is not set
# CONFIG_SENSORS_LM78 is not set
# CONFIG_SENSORS_LM80 is not set
# CONFIG_SENSORS_LM83 is not set
# CONFIG_SENSORS_LM85 is not set
# CONFIG_SENSORS_LM87 is not set
# CONFIG_SENSORS_LM90 is not set
# CONFIG_SENSORS_LM92 is not set
# CONFIG_SENSORS_LM93 is not set
# CONFIG_SENSORS_MAX1619 is not set
# CONFIG_SENSORS_MAX6650 is not set
# CONFIG_SENSORS_PC87360 is not set
# CONFIG_SENSORS_PC87427 is not set
# CONFIG_SENSORS_DME1737 is not set
# CONFIG_SENSORS_SMSC47M1 is not set
# CONFIG_SENSORS_SMSC47M192 is not set
# CONFIG_SENSORS_SMSC47B397 is not set
# CONFIG_SENSORS_ADS7828 is not set
# CONFIG_SENSORS_THMC50 is not set
# CONFIG_SENSORS_VT1211 is not set
# CONFIG_SENSORS_W83781D is not set
# CONFIG_SENSORS_W83791D is not set
# CONFIG_SENSORS_W83792D is not set
# CONFIG_SENSORS_W83793 is not set
# CONFIG_SENSORS_W83L785TS is not set
# CONFIG_SENSORS_W83L786NG is not set
# CONFIG_SENSORS_W83627HF is not set
# CONFIG_SENSORS_W83627EHF is not set
# CONFIG_HWMON_DEBUG_CHIP is not set
# CONFIG_WATCHDOG is not set

#
# Sonics Silicon Backplane
#
CONFIG_SSB_POSSIBLE=y
# CONFIG_SSB is not set

#
# Multifunction device drivers
#
# CONFIG_MFD_SM501 is not set
# CONFIG_MFD_ASIC3 is not set
# CONFIG_HTC_EGPIO is not set
# CONFIG_HTC_PASIC3 is not set

#
# Multimedia devices
#

#
# Multimedia core support
#
# CONFIG_VIDEO_DEV is not set
# CONFIG_DVB_CORE is not set
# CONFIG_VIDEO_MEDIA is not set

#
# Multimedia drivers
#
# CONFIG_DAB is not set

#
# Graphics support
#
# CONFIG_VGASTATE is not set
# CONFIG_VIDEO_OUTPUT_CONTROL is not set
CONFIG_FB=y
# CONFIG_FIRMWARE_EDID is not set
# CONFIG_FB_DDC is not set
CONFIG_FB_CFB_FILLRECT=y
CONFIG_FB_CFB_COPYAREA=y
CONFIG_FB_CFB_IMAGEBLIT=y
# CONFIG_FB_CFB_REV_PIXELS_IN_BYTE is not set
# CONFIG_FB_SYS_FILLRECT is not set
# CONFIG_FB_SYS_COPYAREA is not set
# CONFIG_FB_SYS_IMAGEBLIT is not set
# CONFIG_FB_FOREIGN_ENDIAN is not set
# CONFIG_FB_SYS_FOPS is not set
# CONFIG_FB_SVGALIB is not set
# CONFIG_FB_MACMODES is not set
# CONFIG_FB_BACKLIGHT is not set
# CONFIG_FB_MODE_HELPERS is not set
# CONFIG_FB_TILEBLITTING is not set
```

```
#
# Frame buffer hardware drivers
#
# CONFIG_FB_S1D13XXX is not set
CONFIG_FB_PXA=y
# CONFIG_FB_PXA_SMARTPANEL is not set
CONFIG_FB_PXA_PARAMETERS=y
# CONFIG_FB_MBX is not set
# CONFIG_FB_W100 is not set
# CONFIG_FB_AM200EPD is not set
# CONFIG_FB_VIRTUAL is not set
# CONFIG_BACKLIGHT_LCD_SUPPORT is not set

#
# Display device support
#
# CONFIG_DISPLAY_SUPPORT is not set

#
# Console display driver support
#
# CONFIG_VGA_CONSOLE is not set
CONFIG_DUMMY_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE_DETECT_PRIMARY=y
# CONFIG_FRAMEBUFFER_CONSOLE_ROTATION is not set
# CONFIG_FONTS is not set
CONFIG_FONT_8x8=y
CONFIG_FONT_8x16=y
CONFIG_LOGO=y
# CONFIG_LOGO_LINUX_MONO is not set
# CONFIG_LOGO_LINUX_VGA16 is not set
CONFIG_LOGO_LINUX_CLUT224=y

#
# Sound
#
CONFIG_SOUND=m

#
# Advanced Linux Sound Architecture
#
CONFIG_SND=m
CONFIG_SND_TIMER=m
CONFIG_SND_PCM=m
# CONFIG_SND_SEQUENCER is not set
# CONFIG_SND_MIXER_OSS is not set
# CONFIG_SND_PCM_OSS is not set
# CONFIG_SND_DYNAMIC_MINORS is not set
CONFIG_SND_SUPPORT_OLD_API=y
CONFIG_SND_VERBOSE_PROCFS=y
# CONFIG_SND_VERBOSE_PRINTK is not set
# CONFIG_SND_DEBUG is not set

#
# Generic devices
#
CONFIG_SND_AC97_CODEC=m
# CONFIG_SND_DUMMY is not set
# CONFIG_SND_MTPAV is not set
# CONFIG_SND_SERIAL_U16550 is not set
# CONFIG_SND_MPU401 is not set

#
# ALSA ARM devices
#
CONFIG_SND_PXA2XX_PCM=m
CONFIG_SND_PXA2XX_AC97=m

#
# USB devices
#
# CONFIG_SND_USB_AUDIO is not set
# CONFIG_SND_USB_CAIAQ is not set

#
# System on Chip audio support
#
# CONFIG_SND_SOC is not set

#
# ALSA SoC audio for Freescale SOCs
#

#
# SoC Audio for the Texas Instruments OMAP
#

#
# Open Sound System
```

```
#
# CONFIG_SOUND_PRIME is not set
CONFIG_AC97_BUS=m
CONFIG_HID_SUPPORT=y
CONFIG_HID=y
# CONFIG_HID_DEBUG is not set
# CONFIG_HIDRAW is not set

#
# USB Input Devices
#
CONFIG_USB_HID=y
# CONFIG_USB_HIDINPUT_POWERBOOK is not set
# CONFIG_HID_FF is not set
# CONFIG_USB_HIDDEV is not set
CONFIG_USB_SUPPORT=y
CONFIG_USB_ARCH_HAS_HCD=y
CONFIG_USB_ARCH_HAS_OHCI=y
# CONFIG_USB_ARCH_HAS_EHCI is not set
CONFIG_USB=y
# CONFIG_USB_DEBUG is not set
# CONFIG_USB_ANNOUNCE_NEW_DEVICES is not set

#
# Miscellaneous USB options
#
CONFIG_USB_DEVICEFS=y
# CONFIG_USB_DEVICE_CLASS is not set
# CONFIG_USB_DYNAMIC_MINORS is not set
# CONFIG_USB_OTG is not set
# CONFIG_USB_OTG_WHITELIST is not set
# CONFIG_USB_OTG_BLACKLIST_HUB is not set

#
# USB Host Controller Drivers
#
# CONFIG_USB_C67X00_HCD is not set
# CONFIG_USB_ISP116X_HCD is not set
# CONFIG_USB_ISP1760_HCD is not set
CONFIG_USB_OHCI_HCD=y
# CONFIG_USB_OHCI_BIG_ENDIAN_DESC is not set
# CONFIG_USB_OHCI_BIG_ENDIAN_MMIO is not set
CONFIG_USB_OHCI_LITTLE_ENDIAN=y
# CONFIG_USB_SL811_HCD is not set
# CONFIG_USB_R8A66597_HCD is not set

#
# USB Device Class drivers
#
# CONFIG_USB_ACM is not set
# CONFIG_USB_PRINTER is not set
# CONFIG_USB_WDM is not set

#
# NOTE: USB_STORAGE enables SCSI, and 'SCSI disk support'
#

#
# may also be needed; see USB_STORAGE Help for more information
#
CONFIG_USB_STORAGE=y
# CONFIG_USB_STORAGE_DEBUG is not set
# CONFIG_USB_STORAGE_DATAFAB is not set
# CONFIG_USB_STORAGE_FREECOM is not set
# CONFIG_USB_STORAGE_ISD200 is not set
# CONFIG_USB_STORAGE_DPCM is not set
# CONFIG_USB_STORAGE_USBAT is not set
# CONFIG_USB_STORAGE_SDDR09 is not set
# CONFIG_USB_STORAGE_SDDR55 is not set
# CONFIG_USB_STORAGE_JUMPSHOT is not set
# CONFIG_USB_STORAGE_ALAUDA is not set
# CONFIG_USB_STORAGE_ONETOUCH is not set
# CONFIG_USB_STORAGE_KARMA is not set
# CONFIG_USB_STORAGE_CYPRESS_ATACB is not set
# CONFIG_USB_LIBUSUAL is not set

#
# USB Imaging devices
#
# CONFIG_USB_MDC800 is not set
# CONFIG_USB_MICROTEK is not set
CONFIG_USB_MON=y

#
# USB port drivers
#
# CONFIG_USB_SERIAL is not set

#
# USB Miscellaneous drivers
```

```
#
# CONFIG_USB_EMI62 is not set
# CONFIG_USB_EMI26 is not set
# CONFIG_USB_ADUTUX is not set
# CONFIG_USB_AUERSWALD is not set
# CONFIG_USB_RIO500 is not set
# CONFIG_USB_LEGOTOWER is not set
# CONFIG_USB_LCD is not set
# CONFIG_USB_BERRY_CHARGE is not set
# CONFIG_USB_LED is not set
# CONFIG_USB_CYPRESS_CY7C63 is not set
# CONFIG_USB_CYTHERM is not set
# CONFIG_USB_PHIDGET is not set
# CONFIG_USB_IDMOUSE is not set
# CONFIG_USB_FTDI_ELAN is not set
# CONFIG_USB_APPLEDISPLAY is not set
# CONFIG_USB_LD is not set
# CONFIG_USB_TRANCEVIBRATOR is not set
# CONFIG_USB_IOWARRIOR is not set
# CONFIG_USB_TEST is not set
# CONFIG_USB_ISIGHTFW is not set
# CONFIG_USB_GADGET is not set
CONFIG_MMC=m
# CONFIG_MMC_DEBUG is not set
# CONFIG_MMC_UNSAFE_RESUME is not set

#
# MMC/SD Card Drivers
#
CONFIG_MMC_BLOCK=m
CONFIG_MMC_BLOCK_BOUNCE=y
# CONFIG_SDIO_UART is not set
# CONFIG_MMC_TEST is not set

#
# MMC/SD Host Controller Drivers
#
CONFIG_MMC_PXA=m
CONFIG_NEW_LEDS=y
CONFIG_LEDS_CLASS=y

#
# LED drivers
#
# CONFIG_LEDS_GPIO is not set
CONFIG_LEDS_CM_X270=y

#
# LED Triggers
#
# CONFIG_LEDS_TRIGGERS is not set
CONFIG_RTC_LIB=y
# CONFIG_RTC_CLASS is not set
# CONFIG_UIO is not set

#
# File systems
#
CONFIG_EXT2_FS=y
# CONFIG_EXT2_FS_XATTR is not set
# CONFIG_EXT2_FS_XIP is not set
CONFIG_EXT3_FS=y
CONFIG_EXT3_FS_XATTR=y
# CONFIG_EXT3_FS_POSIX_ACL is not set
# CONFIG_EXT3_FS_SECURITY is not set
# CONFIG_EXT4DEV_FS is not set
CONFIG_JBD=y
CONFIG_FS_MBCACHE=y
# CONFIG_REISERFS_FS is not set
# CONFIG_JFS_FS is not set
# CONFIG_FS_POSIX_ACL is not set
# CONFIG_XFS_FS is not set
# CONFIG_OCFS2_FS is not set
CONFIG_DNOTIFY=y
CONFIG_INOTIFY=y
CONFIG_INOTIFY_USER=y
# CONFIG_QUOTA is not set
# CONFIG_AUTOFS_FS is not set
# CONFIG_AUTOFS4_FS is not set
# CONFIG_FUSE_FS is not set

#
# CD-ROM/DVD Filesystems
#
# CONFIG_ISO9660_FS is not set
# CONFIG_UDF_FS is not set

#
# DOS/FAT/NT Filesystems
#
```

```
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
CONFIG_VFAT_FS=y
CONFIG_FAT_DEFAULT_CODEPAGE=437
CONFIG_FAT_DEFAULT_IOCHARSET="iso8859-1"
# CONFIG_NTFS_FS is not set

#
# Pseudo filesystems
#
CONFIG_PROC_FS=y
CONFIG_PROC_SYSCTL=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
# CONFIG_TMPFS_POSIX_ACL is not set
# CONFIG_HUGETLB_PAGE is not set
# CONFIG_CONFIGFS_FS is not set

#
# Miscellaneous filesystems
#
# CONFIG_ADFS_FS is not set
# CONFIG_AFFS_FS is not set
# CONFIG_HFS_FS is not set
# CONFIG_HFSPLUS_FS is not set
# CONFIG_BEFS_FS is not set
# CONFIG_BFS_FS is not set
# CONFIG_EFS_FS is not set
CONFIG_JFFS2_FS=y
CONFIG_JFFS2_FS_DEBUG=0
CONFIG_JFFS2_FS_WRITEBUFFER=y
# CONFIG_JFFS2_FS_WBUF_VERIFY is not set
CONFIG_JFFS2_SUMMARY=y
# CONFIG_JFFS2_FS_XATTR is not set
# CONFIG_JFFS2_COMPRESSION_OPTIONS is not set
CONFIG_JFFS2_ZLIB=y
# CONFIG_JFFS2_LZO is not set
CONFIG_JFFS2_RTIME=y
# CONFIG_JFFS2_RUBIN is not set
# CONFIG_CRAMFS is not set
# CONFIG_VXFS_FS is not set
# CONFIG_MINIX_FS is not set
# CONFIG_HPFS_FS is not set
# CONFIG_QNX4FS_FS is not set
# CONFIG_ROMFS_FS is not set
# CONFIG_SYSV_FS is not set
# CONFIG_UFS_FS is not set
CONFIG_NETWORK_FILESYSTEMS=y
# CONFIG_NFS_FS is not set
# CONFIG_NFSD is not set
# CONFIG_SMB_FS is not set
# CONFIG_CIFS is not set
# CONFIG_NCP_FS is not set
# CONFIG_CODA_FS is not set
# CONFIG_AFS_FS is not set

#
# Partition Types
#
# CONFIG_PARTITION_ADVANCED is not set
CONFIG_MSDOS_PARTITION=y
CONFIG_NLS=y
CONFIG_NLS_DEFAULT="iso8859-1"
CONFIG_NLS_CODEPAGE_437=y
# CONFIG_NLS_CODEPAGE_737 is not set
# CONFIG_NLS_CODEPAGE_775 is not set
# CONFIG_NLS_CODEPAGE_850 is not set
# CONFIG_NLS_CODEPAGE_852 is not set
# CONFIG_NLS_CODEPAGE_855 is not set
# CONFIG_NLS_CODEPAGE_857 is not set
# CONFIG_NLS_CODEPAGE_860 is not set
# CONFIG_NLS_CODEPAGE_861 is not set
# CONFIG_NLS_CODEPAGE_862 is not set
# CONFIG_NLS_CODEPAGE_863 is not set
# CONFIG_NLS_CODEPAGE_864 is not set
# CONFIG_NLS_CODEPAGE_865 is not set
# CONFIG_NLS_CODEPAGE_866 is not set
# CONFIG_NLS_CODEPAGE_869 is not set
# CONFIG_NLS_CODEPAGE_936 is not set
# CONFIG_NLS_CODEPAGE_950 is not set
# CONFIG_NLS_CODEPAGE_932 is not set
# CONFIG_NLS_CODEPAGE_949 is not set
# CONFIG_NLS_CODEPAGE_874 is not set
# CONFIG_NLS_ISO8859_8 is not set
# CONFIG_NLS_CODEPAGE_1250 is not set
# CONFIG_NLS_CODEPAGE_1251 is not set
# CONFIG_NLS_ASCII is not set
CONFIG_NLS_ISO8859_1=y
# CONFIG_NLS_ISO8859_2 is not set
# CONFIG_NLS_ISO8859_3 is not set
```

```
# CONFIG_NLS_ISO8859_4 is not set
# CONFIG_NLS_ISO8859_5 is not set
# CONFIG_NLS_ISO8859_6 is not set
# CONFIG_NLS_ISO8859_7 is not set
# CONFIG_NLS_ISO8859_9 is not set
# CONFIG_NLS_ISO8859_13 is not set
# CONFIG_NLS_ISO8859_14 is not set
# CONFIG_NLS_ISO8859_15 is not set
# CONFIG_NLS_KOI8_R is not set
# CONFIG_NLS_KOI8_U is not set
# CONFIG_NLS_UTF8 is not set
# CONFIG_DLM is not set

#
# Kernel hacking
#
# CONFIG_PRINTK_TIME is not set
CONFIG_ENABLE_WARN_DEPRECATED=y
CONFIG_ENABLE_MUST_CHECK=y
CONFIG_FRAME_WARN=1024
# CONFIG_MAGIC_SYSRQ is not set
# CONFIG_UNUSED_SYMBOLS is not set
# CONFIG_DEBUG_FS is not set
# CONFIG_HEADERS_CHECK is not set
# CONFIG_DEBUG_KERNEL is not set
# CONFIG_DEBUG_BUGVERBOSE is not set
CONFIG_FRAME_POINTER=y
# CONFIG_SAMPLES is not set
# CONFIG_DEBUG_USER is not set

#
# Security options
#
# CONFIG_KEYS is not set
# CONFIG_SECURITY is not set
# CONFIG_SECURITY_FILE_CAPABILITIES is not set
CONFIG_CRYPTO=y

#
# Crypto core or helper
#
CONFIG_CRYPTO_ALGAPI=y
CONFIG_CRYPTO_BLKCIPHER=y
CONFIG_CRYPTO_MANAGER=y
# CONFIG_CRYPTO_GF128MUL is not set
# CONFIG_CRYPTO_NULL is not set
# CONFIG_CRYPTO_CRYPTD is not set
# CONFIG_CRYPTO_AUTHENC is not set
# CONFIG_CRYPTO_TEST is not set

#
# Authenticated Encryption with Associated Data
#
# CONFIG_CRYPTO_CCM is not set
# CONFIG_CRYPTO_GCM is not set
# CONFIG_CRYPTO_SEQIV is not set

#
# Block modes
#
# CONFIG_CRYPTO_CBC is not set
# CONFIG_CRYPTO_CTR is not set
# CONFIG_CRYPTO_CTS is not set
CONFIG_CRYPTO_ECB=y
# CONFIG_CRYPTO_LRW is not set
# CONFIG_CRYPTO_PCBC is not set
# CONFIG_CRYPTO_XTS is not set

#
# Hash modes
#
# CONFIG_CRYPTO_HMAC is not set
# CONFIG_CRYPTO_XCBC is not set

#
# Digest
#
# CONFIG_CRYPTO_CRC32C is not set
# CONFIG_CRYPTO_MD4 is not set
# CONFIG_CRYPTO_MD5 is not set
# CONFIG_CRYPTO_MICHAEL_MIC is not set
# CONFIG_CRYPTO_SHA1 is not set
# CONFIG_CRYPTO_SHA256 is not set
# CONFIG_CRYPTO_SHA512 is not set
# CONFIG_CRYPTO_TGR192 is not set
# CONFIG_CRYPTO_WP512 is not set

#
# Ciphers
#
```

```
CONFIG_CRYPTO_AES=y
# CONFIG_CRYPTO_ANUBIS is not set
CONFIG_CRYPTO_ARC4=y
# CONFIG_CRYPTO_BLOWFISH is not set
# CONFIG_CRYPTO_CAMELLIA is not set
# CONFIG_CRYPTO_CAST5 is not set
# CONFIG_CRYPTO_CAST6 is not set
# CONFIG_CRYPTO_DES is not set
# CONFIG_CRYPTO_FCRYPT is not set
# CONFIG_CRYPTO_KHAZAD is not set
# CONFIG_CRYPTO_SALSA20 is not set
# CONFIG_CRYPTO_SEED is not set
# CONFIG_CRYPTO_SERPENT is not set
# CONFIG_CRYPTO_TEA is not set
# CONFIG_CRYPTO_TWOFISH is not set

#
# Compression
#
# CONFIG_CRYPTO_DEFLATE is not set
# CONFIG_CRYPTO_LZO is not set
# CONFIG_CRYPTO_HW is not set

#
# Library routines
#
CONFIG_BITREVERSE=y
# CONFIG_GENERIC_FIND_FIRST_BIT is not set
# CONFIG_GENERIC_FIND_NEXT_BIT is not set
# CONFIG_CRC_CCITT is not set
# CONFIG_CRC16 is not set
CONFIG_CRC_ITU_T=y
CONFIG_CRC32=y
# CONFIG_CRC7 is not set
# CONFIG_LIBCRC32C is not set
CONFIG_ZLIB_INFLATE=y
CONFIG_ZLIB_DEFLATE=y
CONFIG_PLIST=y
CONFIG_HAS_IOMEM=y
CONFIG_HAS_IOPORT=y
CONFIG_HAS_DMA=y
```

B Appendix B
Kermit configuration

```
set line /dev/ttyUSB4
set speed 38400
set carrier-watch off
set flow-control none
set parity none
connect
```