Hardware and software development of a uClinux Voice over IP telephone platform

A Master of Science thesis performed in Electronics Systems
by
Sven Johnsson

LiTH-ISY-EX--07/3961--SE
Norrköping 2007
Hardware and software development of a uClinux Voice over IP telephone platform

Master of Science thesis
Division of Electronics System
Department of Electrical Engineering
Linköping Institute of Technology

Sven Johnsson
LiTH-ISY-EX--07/3961--SE

Supervisor: Markus Englund, Häger Scandinavia AB
Examiner / supervisor: Kent Palmkvist

Norrköping, June 14th 2007
Abstract

Voice over IP technology (VoIP) has recently gained popularity among consumers. Many popular VoIP services exist only as software for PCs. The need of taking such services out of the PC, into a stand-alone device has been discovered, and this thesis work deals with the development of such a device. The thesis work is done for Häger Scandinavia AB, a Swedish telephone manufacturer. This thesis work covers the design of a complete prototype of a table-top VoIP telephone running an embedded Linux Operating system. Design areas include product development, hardware design and software design. The result is a working prototype with hardware and corresponding Linux device drivers. The prototype can host a Linux application adapted to it. Conclusions are that the first hardware version has worked well and that using an open-source operating system is very useful. Further work consists of implementing a complete telephony software application in the system, evaluation of system requirements and adapting the prototype for a commercial design.

Keywords

VoIP, Voice over IP, uClinux, Linux, telephone, Coldfire
Abstract

Voice over IP technology (VoIP) has recently gained popularity among consumers. Many popular VoIP services exist only as software for PCs. The need of taking such services out of the PC, into a stand-alone device has been discovered, and this thesis work deals with the development of such a device. The thesis work is done for Häger Scandinavia AB, a Swedish telephone manufacturer. This thesis work covers the design of a complete prototype of a table-top VoIP telephone running an embedded Linux Operating system. Design areas include product development, hardware design and software design. The result is a working prototype with hardware and corresponding Linux device drivers. The prototype can host a Linux application adapted to it. Conclusions are that the first hardware version has worked well and that using an open-source operating system is very useful. Further work consists of implementing a complete telephony software application in the system, evaluation of system requirements and adapting the prototype for a commercial design.
Preface

This report covers a Master's thesis work performed at the Division of Electronics Systems at Linköping Institute of Technology for Häger Scandinavia AB.

Various persons have assisted me with ideas throughout my work. Industrial design skills have been provided by Kenneth Bringzén, whom I would like to thank for his unique designs and crazy ideas. CAD engineering for a future exterior design have been provided by Markus Strömberg, whose amazing Pro/ENGINEER skills I envy. I would like to thank Attila Szelp for his excellent web design skills and devoted cooperation with the website of the project. As the final participant of the voluntary design team, I would like to thank Therese Forsberg for manufacturing the circuit boards and helping me to mount some critical components.

I would like to thank my examiner and technical supervisor Kent Palmkvist for answering my questions during the thesis project. I am especially grateful to my supervisor at Häger, Markus Englund, for believing in my idea and to support me during this project. Finally, I would also like to thank Adam Kalinsky at Freescale Semiconductor for supplying me with the evaluation board.

Sven Johnsson,

June 2007
Abbreviations

ADC  Analog to Digital Converter
ALSA  Advanced Linux Sound Architecture
API  Application Programmer's Interface
ARM  Advanced RISC Machine
ATA  Analog Telephone Adapter
BGA  Ball Grid Array
BSP  Board Support Package
CAN  Controller Area Network
CODEC  Coder-Decoder
CPU  Central Processing Unit
DAC  Digital to Analog Converter
DC  Direct Current
DMA  Direct Memory Access
DSP  Digital Signal Processor
EMC  Electromagnetic Compatibility
EVB  Evaluation Board
I²C  Inter-IC
IAX  Inter-Asterisk eXchange
IC  Integrated Circuit
iLBC  internet Low Bit Rate Codec
IP  Internet Protocol
iSAC  internet Speech Audio Codec
LAN  Local Area Network
LCD  Liquid Crystal Display
LED  Light Emitting Diode
MIPS  Million Instructions Per Second
MMU  Memory Management Unit
NDA  Non-Disclosure Agreement
NFS  Network File System
ODM  Original Design Manufacturer
OSS  Open Sound System
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>POTS</td>
<td>Plain Old Telephone Service</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse Width Modulation</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computer</td>
</tr>
<tr>
<td>RoHS</td>
<td>Restriction of Hazardous Substances</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory</td>
</tr>
<tr>
<td>ROMFS</td>
<td>ROM File System</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Synchronous Dynamic Random Access Memory</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SSI</td>
<td>Synchronous Serial Interface</td>
</tr>
<tr>
<td>STN</td>
<td>Super-Twisted Nematic display</td>
</tr>
<tr>
<td>TACT</td>
<td>TACTile</td>
</tr>
<tr>
<td>TFTP</td>
<td>Trivial File Transfer Protocol</td>
</tr>
<tr>
<td>THD</td>
<td>Total Harmonic Distortion</td>
</tr>
<tr>
<td>uClinux</td>
<td>Micro Controller Linux</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>Wireless Fidelity</td>
</tr>
</tbody>
</table>
Contents
1. Introduction ........................................................................................................................................ 1
   1.1 Background .................................................................................................................................. 1
   1.2 Purpose ......................................................................................................................................... 1
   1.3 Method .......................................................................................................................................... 1
   1.4 Limitations .................................................................................................................................... 2
   1.5 Confidential information .............................................................................................................. 2
   1.6 Structure of the report .................................................................................................................. 2
2. Preparations and pre-studies .............................................................................................................. 3
   2.1 Starting up and finding a company ............................................................................................... 3
   2.2 Gathering design requirements ................................................................................................... 3
   2.3 Re-defining the purpose ................................................................................................................. 3
   2.4 Linux studies .................................................................................................................................. 4
3. Requirements ....................................................................................................................................... 5
   3.1 Primary design requirements ......................................................................................................... 5
   3.2 Secondary design requirements ..................................................................................................... 5
   3.3 Primary project requirements ........................................................................................................ 6
   3.4 Secondary project requirements .................................................................................................... 6
4. Technology background ....................................................................................................................... 7
   4.1 Voice over IP (VoIP) ..................................................................................................................... 7
      4.1.1 Basic concepts ...................................................................................................................... 7
      4.1.2 Protocols ............................................................................................................................. 8
      4.1.3 Voice CODECs ..................................................................................................................... 8
   4.2 uClinux .......................................................................................................................................... 9
      4.2.1 Basic facts about uClinux ..................................................................................................... 9
      4.2.2 The lack of a memory management unit .............................................................................. 9
      4.2.3 Linux device drivers ........................................................................................................... 10
         4.2.3.1 General information .................................................................................................... 10
         4.2.3.2 Sound device drivers in Linux .................................................................................... 11
      4.2.4 The board support package (BSP) ...................................................................................... 12
   4.3 Electronics background ................................................................................................................ 12
4.3.1 Audio CODECs ................................................................. 12
4.3.2 The I2C bus ................................................................. 12
4.3.3 The SSI bus ................................................................. 12

5. Design and implementation .................................................. 13

5.1 Hardware ........................................................................ 13
5.1.1 System design ............................................................ 13
5.1.2 Selection of central components .................................... 16
  5.1.2.1 Processor and evaluation board ............................... 17
    5.1.2.1.1 Requirements .................................................. 17
    5.1.2.1.2 Freescale Semiconductors: Coldfire ................... 18
    5.1.2.1.3 5V Technologies: 5VT 1312 ............................. 19
    5.1.2.1.4 Conclusion .................................................... 20
  5.1.2.2 CODEC ................................................................. 20
  5.1.2.3 Power amplifier ...................................................... 20
  5.1.2.4 I2C expanders ....................................................... 21
  5.1.2.5 Hookswitch .......................................................... 21
  5.1.2.6 LCD module ......................................................... 21
  5.1.2.7 Switches .............................................................. 22

5.1.3 Hardware design ......................................................... 22
  5.1.3.1 Development environment .................................... 22
  5.1.3.2 Schematic capture of the motherboard ....................... 22
  5.1.3.3 PCB layout of the motherboard ............................... 27
  5.1.3.4 Schematic capture of the interface board .................. 31
  5.1.3.5 PCB layout of the interface board ........................... 34
  5.1.3.6 The hookswitch board ......................................... 34

5.2 Software ......................................................................... 35
  5.2.1 Development environment ....................................... 35
    5.2.1.1 Host system ...................................................... 35
    5.2.1.2 Target system .................................................. 36
  5.2.2 I2C driver ............................................................... 37
  5.2.3 Keypad driver ......................................................... 38
  5.2.4 LED driver ............................................................. 39
5.2.5 LCD driver........................................................................................................ 39
5.2.6 Sound driver.................................................................................................... 40
5.3 Demonstration system..................................................................................... 42

6. Results / Verification and testing..................................................................... 43
   6.1 Processor execution speed........................................................................... 43
   6.2 Hardware and software............................................................................... 43

7. Conclusions ......................................................................................................... 45

8. Further work........................................................................................................... 46

References................................................................................................................ 47
List of Figures

Figure 1: The central components of a VoIP system.......................................................... 7
Figure 2: Overview of the system.......................................................................................... 14
Figure 3: A more detailed view of the system....................................................................... 15
Figure 4: A complete overview of the system....................................................................... 16
Figure 5: Top level schematic of the motherboard................................................................. 22
Figure 6: Top level schematic of the interface board.............................................................. 31
Figure 7: The device driver system...................................................................................... 36
List of Tables

Table 1: Processor requirements from Global IP Sound.................................................. 9
Table 2: Coldfire processors.......................................................................................... 18
Table 3: Coldfire evaluation boards................................................................................ 19
Table 4: Selection of supply voltages for the CODEC.................................................... 24
Table 5: Parameters for estimating the need of a heatsink.............................................. 29
1. Introduction

1.1 Background

Until recently, Voice over IP (VoIP) technology has mainly been used by companies. Today, the technology has become very popular among consumers.

Most VoIP services require a PC running a specific software. This means that the computer has to be switched on for the user to be available on his/her Internet telephone. To avoid this dependence a stand-alone VoIP telephone is required. Such telephones exist, but are often bound to specific standards, and thus do not support all VoIP services. The very popular VoIP service Skype uses a proprietary protocol to communicate, making those stand-alone phones useless for their service. This is the situation for many software phone clients.

To make VoIP available to more people and to eliminate the dependence of a PC, a new stand-alone VoIP telephone is required. This telephone needs to be able to run software from different service providers.

1.2 Purpose

The purpose of this Master's thesis is to develop and evaluate a fully functional prototype of a stand-alone Voice Over IP (VoIP) telephone with the possibility to run different VoIP softwares. The development includes circuit design, software design and some exterior design as well. The idea behind this Master's thesis project comes from the author and the result will be used by Häger Scandinavia AB, a Swedish telephone manufacturer.

1.3 Method

Various books about embedded Linux have been studied to give the required knowledge before the project was started. General information about Voice over IP services and equipment has been gathered. Design requirements have been collected. Hardware for a telephone prototype has been created by choosing components, drawing schematics, drawing printed circuit board layouts, manufacturing circuit boards and assembling the circuit boards. A target system has been put together by using the previously constructed hardware and an evaluation board containing a processor running an embedded Linux operating system. Linux device drivers have been designed for the hardware and have been tested continuously with the hardware to achieve the proper functionality. Application software for demonstration of the prototype functionality has been created. Both hardware and software has been evaluated for future revisions.
1.4 Limitations
A complete software application for Internet telephony and instant messaging has not been integrated into the telephone prototype. The processor and its peripherals are not integrated into a circuit board of the prototype. Instead, an evaluation board containing the processor and its peripherals is used. Only one revision of the hardware is created. The prototype hardware is not integrated into a rapid prototype plastic casing. The sound device driver has been written using OSS, an older standard for Linux sound device drivers.

1.5 Confidential information
The results of this thesis work (hardware and software) is property of Häger Scandinavia AB. Because of this, information such as schematics, selected components and source code is not available in this report.

1.6 Structure of the report
The structure of this report is based on the actual work flow of the project. Chapter 2 to 4 cover preparations and basic theory while chapter 5 to 7 covers the design, the results and conclusions. Chapter 8 deals with recommendations for further work.
2. Preparations and pre-studies

This Master's thesis work has required many preparations before its actual start. Preparations include finding a company, gathering design requirements and literature studies.

2.1 Starting up and finding a company

At first, the plan was to develop a prototype of a stand-alone Skype telephone. The idea for this thesis project was conceived by the author during the winter of 2005. After some emails to Skype and a brief phone call to the CEO of Skype, Niklas Zennström, one thing was made clear: Skype can not host such a thesis project since they do not make any hardware themselves. The general advice was to speak to one of Skype's ODM (original design manufacturer) partners. The only Nordic ODM (RTX) was asked regarding hosting this thesis work, and their answer was no. Finally, a Swedish telephone manufacturer, Häger Scandinavia AB was contacted. Their answer was very positive, and the outcome was that this thesis work is made in cooperation with them.

2.2 Gathering design requirements

The next step was to define the new product. With the help of an Austrian web designer, a website was created. Its aim was to collect opinions from Skype users all over the world, so that the future telephone would appeal to as many consumers as possible. The website was launched at www.standalone-skype.com in the 16 of March 2006. Its main contents is a brief project description and a survey allowing visitors to describe their ideal stand-alone Skype telephones. In May of 2007, 380 submitted surveys had been received.

Information from collected surveys, conversations over Skype, various web sites and discussions with Häger Scandinavia AB finally lead to the design requirements of this project.

An industrial designer was contacted regarding the exterior design of the telephone. This resulted in a cooperation where a complete concept design for this project was created. This design has created various requirements for the prototype hardware.

2.3 Re-defining the purpose

In the beginning, the purpose was to develop a prototype for a stand-alone Skype telephone. Efforts have been made by Häger Scandinavia AB to initiate a partnership with Skype to make this project possible. In May of 2006, no answer had been received from Skype. A Swedish competitor to Skype named Woize was contacted instead. Their answer was very positive, and the new plan was to develop a telephone for their service. Woize announced plans to launch software for Linux in Q4 of 2006. This software could be rewritten to work in the prototype designed in this project. Until today (May 2007), this software has still not been released to the public. The current conclusion is
that there will not be any possibility of integrating Woize software into the prototype during this thesis project.

All previous events mentioned has lead to the following decision: The results of this thesis project will be available as a reference design from Häger Scandinavia AB. The design is not customized for any specific VoIP service.

### 2.4 Linux studies

Good knowledge in Linux has been required when working on this project. The author's knowledge in Linux was almost non-existent in the beginning of the project. This problem has been resolved by massive training in the Linux operating system by reading books, studying websites, running Linux at home and writing programs for Linux using the C programming language.

Books studied include

- Linux Bible [1]
- Embedded Linux System Design and Development [2]
- Linux Device Drivers [3]
- Vägen till C [4]
3. Requirements

There are two types of requirements for this project: design requirements and project requirements. The design requirements mostly cover the desired physical contents of the future product while the project requirements state the desired achievements of the project.

3.1 Primary design requirements

- A processor running an embedded Linux operating system
- Software portability: a telephony application for Linux should be easy to port to the processor used in this telephone.
- Ethernet connection
- Receiver connector
- Headset connector
- Display
- Navigation keys and alphanumeric keys
- LEDs: indicators and white backlight for the telephone body
- Speaker for playing ring tones
- Table-top / wall-mount design
- Sample rates: 8 kHz to 16 kHz
- 16 bit sample format
- Reduce component costs whenever possible
- Linux device drivers for all hardware functionality
- The processor shall have enough computational capacity for VoIP
- Operation shall be possible in a temperature range of 0 °C to 60 °C

3.2 Secondary design requirements

- Speaker phone capabilities
- Bluetooth headset compatibility
- Wi-Fi adapter via USB
- A sound device driver for ALSA (Advanced Linux Sound Architecture), which is used by most Linux applications today.
3.3 Primary project requirements

- The prototype should be able to demonstrate the different hardware functions by using software constructed within the thesis work. This demonstrator does not have to be a telephony application, although this would be the optimal application to test the system.

- Evaluation of the results regarding signal quality, design decisions and components.

3.4 Secondary project requirements

- Testing by using a VoIP application ported to the current platform.

- Evaluation of different CODECs, especially wideband CODECs such as iSAC from Global IP Sound.

- Evaluation of the processing capabilities of the hardware and possible optimizations in software.

- Evaluation of system requirements for today's VoIP software.
4. Technology background

This chapter explains some basic concepts applied in this Master's thesis work. The reader is assumed to have basic engineering knowledge in electronics and computer technology. Areas covered are Voice over IP technology, the embedded uClinux operating system and some electronics concepts. However, this chapter does not cover all theory applied in this project, and the interested reader is encouraged to obtain additional information in books and on the Internet.

4.1 Voice over IP (VoIP)

This chapter gives a brief introduction to the Voice over IP technology. Both technical information and basic facts are presented.

4.1.1 Basic concepts

VoIP is short for Voice over Internet Protocol and is a technology that allows telephone calls to travel over the Internet.

The central technical components of the Voice over IP technology are protocols and voice CODECs. The voice CODEC codes (compresses) outgoing audio samples and decodes (decompresses) incoming audio data, while the protocol determines the way that data is being transmitted over the network. A simplified schematic of these components is shown below.

There are generally three ways of using this technology: using a computer, a special IP telephone or an analog telephone adapter (ATA).

If a computer is used, it needs to run a Voice over IP application and there is a large variety of software to choose from. Most software available is bound to a certain VoIP provider. This often means the following things:

- An account has to be created at the provider
- Calls to other users of the same provider are unlimited and free of charge
- Calls to ordinary telephones are not free of charge, and the pricing varies between different providers.
The VoIP software today has become instant messengers as well, meaning that other common functions are text chat, file transfer and contact lists with online presence. There is also VoIP software that is not bound to a certain provider, but to specific standard VoIP protocols. Such applications contain the basic telephony functionality but most often no additional features such as file transfer and instant messaging.

An IP telephone is a telephone that can be connected directly to an IP network. The telephone contains software that handles its functionality. This software is often limited to certain protocols and has a limited set of functions. Each telephone has got its own software, also known as firmware. Different firmware versions with different functionality is not very common in IP telephones. Most IP telephones today are used by companies. Consumer IP phones can be divided into two groups:

- Generic IP phones complying with certain protocols without being bound to a provider
- IP phones customized for specific providers using proprietary protocols

An analog telephone adapter (ATA) is a box that is connected between the Internet and regular telephones. By using such a device, any common telephone can be used for calling over the Internet. Such adapters use common VoIP protocols such as SIP. Some providers focus their services around this technology and give subscribers a normal telephone number, making this service a replacement for the POTS (plain old telephone service).

When a service provider uses a standard protocol, any device being compliant with that protocol can be used for the service, including a PC, an IP telephone or an ATA.

### 4.1.2 Protocols

A protocol is a detailed description of how information is transmitted. Popular protocols within VoIP technology are:

- SIP (Session Initiation Protocol)
- IAX (Inter-Asterisk eXchange protocol)
- H.323

### 4.1.3 Voice CODECs

A voice CODEC is a piece of software that compresses outgoing speech signals and decompresses incoming speech signals. The compression is used because the data transmission rate is limited. There is a variety of voice CODECs available with differences in data rates, compression rates and bandwidths. Some voice CODECs are free of charge and have their source code open, while other CODECs do not.

The bandwidth of typical telephony today is limited to between 300 Hz and 3400 Hz [5]. There are so-called wideband CODECs having a larger bandwidth, which makes the perceived sound quality better. Another important parameter affecting the perceived
quality is the amount of signal compression. The iSAC CODEC from Global IP Sound is a popular wideband CODEC which gives a very natural reproduction of both voice and music signals. It is used by Skype and other large soft phone applications.

According to Global IP Sound, approximate requirements for running their voice CODECs on Coldfire v.4 processors are [6]:

<table>
<thead>
<tr>
<th>Codec</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard requirement</td>
<td>120</td>
</tr>
<tr>
<td>Minimal requirement</td>
<td>60</td>
</tr>
<tr>
<td>iLBC</td>
<td>125</td>
</tr>
<tr>
<td>iSAC</td>
<td>200</td>
</tr>
<tr>
<td>iSAC+Echo cancelling</td>
<td>300</td>
</tr>
</tbody>
</table>

*Table 1: Processor requirements from Global IP Sound*

The values above are only accurate for a narrow range of processors, but will still be used as a hint when choosing a processor in this project.

4.2 uClinux

This chapter gives an introduction into the world of Linux and uClinux.

4.2.1 Basic facts about uClinux

uClinux is an operating system derived from the Linux operating system intended for microcontrollers without a memory management unit (MMU). uClinux is open source and therefore free for everyone to use. Readers seeking for more detailed information about Linux, uClinux and Linux device drivers are referred to the books mentioned in the previous chapter on Linux studies.

Since uClinux is derived from Linux, many Linux applications can be ported easily to run on an uClinux system. A simple text-based application like “Hello World” is ported just by recompiling with a cross-compiler for the desired architecture. More advanced software can be ported with more or less effort, depending on issues like hardware dependencies and graphic API:s.

4.2.2 The lack of a memory management unit

Since uClinux runs on microcontrollers without memory management units (MMU) some problems arise. An MMU is used to create virtual memory addresses so that each application can get its own continuous address space, even if it consists of data segments from different parts of the memory.

This means that virtual memory does not exist in uClinux, and some limitations are thus added. The software will access the memory using physical addresses without further control, meaning that badly written software can crash other applications or worse, the entire operating system.
When memory is allocated and deallocated during runtime the memory will contain a mix of free and occupied segments. With virtual memory the free segments will be mapped into a virtually continuous memory segment meaning that the application does not see the mix of free and occupied segments. The lack of MMU and virtual memory, however, requires care to be taken when designing software for uClinux. Dynamic allocation of memory should be used with care, because a heavily fragmented memory is useless in uClinux, since the software needs continuous regions to store its data. Assume for example that every even byte is allocated in a 2 MB memory. Theoretically, there is 1 MB of free memory, but the largest continuous memory block is one byte. This means that the software can only allocate blocks of one byte, nothing more. However, if all applications have allocated their memory at start-up, there is no need to worry. Programs written in C++ can heavily fragment the memory since each new call will allocate new data during runtime [2].

4.2.3 Linux device drivers

Linux device drivers are very important components in a computer running Linux, since they form the bridge between the software and the hardware. This chapter briefly explains some basic concepts within this area.

4.2.3.1 General information

Applications in Linux that need to communicate with hardware (buttons, LEDs, etc.) require an interface to the hardware, called a device driver. A device driver consists of code containing various methods for handling requests from applications.

Examples of requests are

- Write “Hello World” on an LCD screen.
- Read data from the keypad
- Blink a LED

There are two ways of integrating a device driver into the Linux kernel,

- Compiling a driver directly into the kernel
- Compiling into an object format that can be loaded and unloaded into the kernel at runtime (a loadable kernel module)

The advantage of compiling a driver into the kernel is that it is instantly available and that nothing more needs to be done to activate it. Drawbacks are that the memory usage of the kernel increases, and that updating a driver requires rebuilding the kernel and rebooting before it can execute [7].

The kernel module, on the other hand, can be re-loaded at any time. This is very useful during driver development, since the driver needs to be tested regularly. One small drawback of using modules is that they have to be inserted into the kernel before use,
and something must make this happen. This can be solved by a script at boot time or by manual insertion using the \texttt{insmod} call.

When inside the kernel, the driver becomes a part of the operating system and operates invisibly in the background. Care must be taken while programming device drivers because there are many pitfalls. One is the concurrency issue that arises when two applications access one kernel module simultaneously.

There are different types of device drivers in Linux: \textit{character}, \textit{block} and \textit{network} device drivers are the most common types. The drivers created in this project are character device drivers. This type of device driver handles data streams of bytes (characters).

Compiling a device driver is different from compiling an application. Since a device driver is a part of the operating system it must be compiled in a special manner. This is solved by sending module-specific parameters to the compiler in the Makefile for the device driver.

Device drivers are often accessed by using device file nodes in the file system. Such device files are located in the \texttt{dev/} directory. Examples are \texttt{dev/dsp} for the audio driver and \texttt{dev/hda} for hard disk drives. These device files allow file operations such as \texttt{open}, \texttt{close}, \texttt{read}, \texttt{write} and \texttt{ioctl}. Device files are accessed in the same manner as plain data files. The file operations supported by a device file is defined by which operations are implemented by the corresponding device driver.

Device files can not be used for the communication between different drivers. When communicating inside the kernel, another approach is used: exporting symbols to the kernel symbol table. Symbols exported to the table are often functions. A function exported to the symbol table by one driver can be called by another driver.

4.2.3.2 Sound device drivers in Linux

The Open Sound System (OSS) has been employed as the standard sound driver architecture for Linux. OSS provides a simple API, originally designed for old sound cards containing 16 bit playbacks and captures. The simpleness has become a problem with many modern sound cards, since they contain advanced functions that require a more advanced sound driver API.

The Advanced Linux Sound Architecture (ALSA) was designed with the modern sound cards in mind to support more features and to overcome other limitations of the existing sound drivers in Linux.

A PC with multiple sound cards using many channels must use ALSA drivers, so that all functionality can be used properly. OSS, on the other hand, will work fine on a simple system only containing basic sound functionality. Over all, OSS provides a simple and easy API for programming sound applications. The same applies to writing sound drivers for OSS [8].
4.2.4 The board support package (BSP)

A board support package is the part of an operating system that contains support for the hardware that it is running on (CPU, memory etc.). This is realized by various configuration files and drivers to support basic functionality. The term BSP often also includes a Linux distribution and a compiler toolchain for the target system.

4.3 Electronics background

Except for the software, this Master's thesis work contains lots of electronics. This chapter covers some important concepts within this area.

4.3.1 Audio CODECs

An audio CODEC is an integrated circuit containing a digital-to-analog converter (DAC) and an analog-to-digital converter (ADC). Its purpose is to serve as an audio interface between a processor and its analog surroundings. Many CODECs use serial data lines to transfer audio samples.

4.3.2 The I²C bus

The I²C (Inter IC) bus is a bi-directional data bus designed by Philips Semiconductors to decrease the amount of I/O lines in a digital system. Only two wires are used: SDA (serial data) and SCL (serial clock). The maximum clock frequency is 3.4 Mbit/s and each data block is 8 bits wide. The I²C bus supports the interconnection of multiple bus master devices and slaves devices on the same two wires. Since only one data line is used, all transfers are half-duplex [10]. There is a variety of digital circuits today supporting the I²C bus.

4.3.3 The SSI bus

The version of SSI (Synchronous Serial Interface) used in this project is a full-duplex serial interface containing the following signals:

- Bit clock
- Frame sync (indicating the arrival of a new data frame)
- Transmit data
- Receive data
5. Design and implementation

This chapter describes the development of hardware and software in this project. The most important design decisions are explained to give a deeper insight into the development process. The hardware is treated first, since it constitutes the condition for the software to work. The last part of this chapter deals with the final test of the design and implementation: the demonstration system.

5.1 Hardware

The hardware has required much work in this project. Tasks include system design, selection of components, schematic design, PCB layout, PCB manufacturing and assembling the circuit boards.

5.1.1 System design

The term system refers to all electronics included in the telephone prototype. It was established that an evaluation board should be used for the processor and its peripherals. This is motivated by the following facts:

- Most processors use a BGA package which is hard to solder by hand, and this requires a circuit board of four layers which can not be manufactured at campus.
- Using an evaluation board saves the work of choosing peripheral components for the processor. It also saves the work of designing schematics and a circuit board layout for the processor and its peripherals.
- Using an evaluation board with a Linux board support package eliminates the need for creating one, which is a big task for a beginner in Linux.

Hence, the system will consist of the following parts:

- An evaluation board containing a processor running an embedded Linux operating system, and a network connection
- An application system containing the remaining functionality to create a VoIP telephone

An overview of the system is depicted below.
A hardware user interface has been created in collaboration with an industrial designer. The outcome was a panel containing a numeric keypad, TACT switches for navigating menus, and other purposes, a character based backlit LCD module and various LEDs for practical and aesthetic purposes.

The choice of switches will be different in a mass-produced product and will probably be rubber pad switches instead, because of their low price. This has not been considered possible in this prototype design, however. A similar issue applies to the LCD module, where it will be more cost effective to use a tailor-made display and to integrate its peripheral components directly on the telephone PCB instead of using an LCD module. The LEDs will probably remain the same in a commercial design because they will not get any simpler than they already are.

It was decided that the switches should be connected in a matrix which can be read by selecting a column and reading a row. This way each switch has its own coordinate to point it out. This approach requires about $2\sqrt{\text{Number of switches}}$ I/O pins instead of using one digital input pin for each switch. This solution has been chosen because of its low consumption of I/O channels.

It was decided that one circuit board should be dedicated for the user interface and this board is named “Interface board”. This board should be connected to a circuit board containing the remaining circuitry, and this board is named “Motherboard”. The motherboard should connect to the evaluation board containing the processor. This approach is illustrated below.
The interface board must communicate with the processor and the question is how this should be done. The options are:

- **General purpose I/O ports:**
  
  Requires about \( N = 2\sqrt{\text{Number of switches}} + 11\text{ LCD bits} + \text{Number of LEDs} \) I/O lines connected to the processor. The drawback is the large amount of connections between the interface panel and the processor.

- **I²C I/O expanders:**
  
  Requires the same number of I/O lines as the general purpose I/O ports. The lines are connected to nearby placed I/O expanders communicating with the processor via the I²C interface requiring only 2 signals (clock and data) connected to the processor. The drawback is the increased cost and the space occupied by these circuits.

The I²C option was chosen because of its low number of connections to the processor:

Using an I²C expander simplifies the PCB layout since it is desirable to keep the amount of connections to the processor as low as possible. Fewer connections and shorter I/O lines between the interface board and the processor also decreases the amount of potential EMC problems in the system.
To meet the system design requirements, the motherboard has to contain the following circuits:

- CODEC
- Crystal clock oscillator for the CODEC
- Power amplifier for the speaker
- Linear power regulators for the application system
- Various jumper switches to evaluate different possibilities
- Connection to a hookswitch

To detect whether the telephone receiver is lifted or not, a hookswitch is required. When the receiver is lifted, the receiver microphone should be activated, and the processor should get a signal indicating the receiver is lifted. In the opposite case, the headset microphone is activated. It is decided that the hookswitch is placed on a small circuit board that can be put in different positions to evaluate its placement. This board is named “Hookswitch board”. The complete system is illustrated below.

![Diagram](image)

*Figure 4: A complete overview of the system*

### 5.1.2 Selection of central components

Selecting components is an important task which will affect the remaining design process. Components for the telephone prototype have been chosen with low cost in mind. RoHS-classed components have been selected as far as possible because of the regulations on consumer electronics today. This chapter covers the selection process.
5.1.2.1 Processor and evaluation board

The processor used in the prototype has to be able to run an embedded Linux operating system. There are two general groups of processor solutions to choose from:

- A chip containing both an application processor and a digital signal processor (DSP). This allows demanding software such as voice CODECs to execute on the DSP without loading the main CPU.
- A chip containing only one processor. This requires a more powerful processor capable of handling both application software and voice CODEC algorithms.

The dual processor solution is attractive since it increases the chances that more advanced voice CODEC algorithms can be used without demanding a more advanced application processor. A possible drawback is that the portability decreases: new software may contain CODEC algorithms that are not portable to the DSP co-processor. Dual processor systems appear to be popular in Voice over IP solutions from various semiconductor companies. The dual processor solution has not been investigated further, but is worth looking at in future work. It is assumed that running all software on one processor simplifies the porting of new software to the system. According to web browsing done by the author, the availability of evaluation boards with Linux board support packages (BSP) is larger for single processor solutions.

The current alternatives are single processor solutions and come from two manufacturers, Freescale Semiconductor and 5V Technologies, both supplying evaluation boards with Linux board support packages.

5.1.2.1.1 Requirements

An existing stand-alone Skype telephone, Topcom Web@lker 5000 is used as a reference when looking at memory requirements. It has 32 MB of flash memory and 64 MB of SDRAM memory [11]. This telephone uses a graphical user interface on an LCD screen, which is assumed to consume a non-negligible amount of system resources. Examples of this are storage of graphics and frame buffers. This memory capacity will be kept in mind when choosing an evaluation board for a certain processor.

According to table 1 on page 9, the processor requirement for the iSAC wideband voice CODEC is about 200 MIPS. It is assumed that the voice CODEC is the most computational demanding software component and 200 MIPS will be used as a guideline when choosing a processor. It is worth mentioning that the stand-alone Skype telephones released so far (2007-05-01) only seem to use the iLBC CODEC, which consumes about 125 MIPS according to table 1. This has been tested by the author by calling such telephones from Skype and observing which voice CODEC is used. It is assumed that this arises from the fact that the embedded version of Skype does not yet support the embedded version of iSAC, maybe because iLBC is freeware while iSAC is not.
It is considered unrealistic to choose a processor without a board support package, since this would require the work to write a new one, which can be quite a task by itself.

5.1.2.1.2 Freescale Semiconductors: Coldfire

The Coldfire processor is a 32 bit microprocessor based on the popular 68K microprocessor family from Motorola (Freescale Semiconductor is the former semiconductor section of Motorola).

Freescale Semiconductor currently offers 4 different evaluation boards with Linux board support packages, and the choice of processor strongly depends on the availability of board support packages (BSPs). The choice of processor in a future product will probably be different, since a new BSP has to be created for the new hardware anyway. The Linux BSPs from Freescale Semiconductors use the uClinux operating system for their MMU-less processors.

The processors with Linux BSPs available from Freescale Semiconductor are presented briefly in the table below. DMIPS or Dhrystone MIPS is a benchmark figure comparing a processor's performance with that of a reference processor. This figure is used for comparing processor performance [13].

<table>
<thead>
<tr>
<th>Supported processor</th>
<th>fclk_max</th>
<th>DMIPS_max</th>
<th>Serial interfaces</th>
<th>Other</th>
<th>MMU</th>
<th>10k price</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCF5208</td>
<td>166 MHz</td>
<td>159 DMIPS</td>
<td>I2C, QSPI</td>
<td>No</td>
<td></td>
<td>$6.49</td>
</tr>
<tr>
<td>MCF5329</td>
<td>240 MHz</td>
<td>211 DMIPS</td>
<td>I2C, QSPI, SSI</td>
<td>No</td>
<td></td>
<td>$14</td>
</tr>
<tr>
<td>MCF54xx</td>
<td>166-200 MHz</td>
<td>300-400 DMIPS</td>
<td>I2C, DSPI</td>
<td>Floating point unit</td>
<td>Yes</td>
<td>$17-$27</td>
</tr>
</tbody>
</table>

Table 2: Coldfire processors

The first processor does not meet the requirement of 200 MIPS and it will not be chosen. The second processor, MCF5329 meets the MIPS requirement and has a lower price than the last group of processors in the table above. Its drawback is the lack of a memory management unit (MMU), but since the uClinux operating system can run on this processor, it is considered to be the best choice of the Coldfire processors presented above.

The evaluation boards for Coldfire processors are described in the table below.
<table>
<thead>
<tr>
<th>Evaluation board</th>
<th>RAM memory</th>
<th>Flash memory</th>
<th>Budgetary Price($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M5208EVB</td>
<td>32 MB SDRAM</td>
<td>2 MB</td>
<td>$348.98</td>
</tr>
<tr>
<td>M5329EVB</td>
<td>32 MB SDRAM</td>
<td>16 MB NAND flash, 2 MB boot flash</td>
<td>$698.96</td>
</tr>
<tr>
<td>MCF5475/85 EVB</td>
<td>64 MB SDRAM</td>
<td>16 MB</td>
<td>$849.95</td>
</tr>
<tr>
<td>MCF5474/84 LITE</td>
<td>64 MB SDRAM</td>
<td>4 MB</td>
<td>$349.98</td>
</tr>
</tbody>
</table>

Table 3: Coldfire evaluation boards

The evaluation board of the MCF5329 processor has about half the memory capacity of the Topcom Skype telephone mentioned in the previous requirements chapter. This evaluation board will, however, be the choice if a Coldfire processor is selected.

5.1.2.1.3 5V Technologies: 5VT 1312

This is an ARM based processor supported by the commercial MontaVista embedded Linux operating system.

Features include:

- ARM 926EJS 300 MHz / 330 MIPS
- Memory management unit (MMU)
- Voice Channel x 2
- 10/100 MAC x 2
- Codec Support: G.711/G.723.1/G.729AB/G.726/iLBC
- Echo Cancellation: G.167/G.168

An evaluation board exists but its price was considered too high compared to other alternatives. No more details about the processor and the evaluation board can be mentioned in this report because this information is protected under a non-disclosure agreement (NDA).
5.1.2.1.4 Conclusion

The Freescale MCF5329 Coldfire processor has been chosen for a variety of reasons:

- The MIPS requirement is met
- Lowest price for the required capabilities compared to other Coldfire options
- Good support resources from Freescale Semiconductor
- uClinux BSP, open-source operating system
- A heavily subsidized evaluation board is available

The processor from 5V Technologies has advantages, such as an MMU, higher MIPS capacity and a lower price compared to the chosen processor. However, since this is a thesis work and the author is rather unskilled in embedded Linux technology, the processor from Freescale Semiconductor has been chosen because it was considered to be a safer choice. This assumption is based on experiences of previous relations with Freescale Semiconductor.

5.1.2.2 CODEC

The required CODEC should meet the following requirements:

- A sample rate of 8 kHz to 16 kHz
- A low price compared to other CODECs
- A serial interface being able to communicate with the selected processor

A CODEC has been chosen successfully from the above requirements. The selected CODEC uses a synchronous serial interface, which can communicate with the corresponding interface on the processor to transmit and receive sound data. The I²C bus can be used to receive commands from the processor such as data format, output volume, etc. The selected CODEC has a headphone driver and a microphone amplifier that enables direct interfacing to a receiver or a headset, only requiring a few discrete components. This is preferable, because it saves the cost and physical space of external headphone and microphone interface circuitry.

5.1.2.3 Power amplifier

The required power amplifier should meet the following requirements:

- Possibility of driving one 8 Ω speaker at 62.5 mW
- A mute input port enabling other hardware to turn the amplifier on and off
- Single supply voltage of 3.3 V
At first, the goal was to build a power amplifier from discrete components. After some simulations and research this was considered to be an inefficient solution, since it would require a large amount of transistors to achieve an acceptable performance. Instead, a small integrated power amplifier has been chosen successfully from the above requirements. The selected power amplifier is a bridged power amplifier requiring only a few external components to work. It has a shut-down control to turn the amplification on and off. Allowed values for the supply voltage are 2.0 V to 5.5 V. The input is differential, giving the possibility to supply a single-ended or a differential signal.

5.1.2.4 I²C expanders
The required interface circuits should meet the following requirements:
- Capability of driving and fading 8 LEDs
- 19 digital output lines
- 5 digital input lines
- One interrupt output line to signal when any input data channel changes
- Capable of using a power supply of 3.3 V.

Two I²C interface circuits have been chosen successfully for the interface board: one I/O expander and one LED driver. The LED driver has a built-in PWM functionality for fading LEDs. Its 16 pins can also be used as digital outputs or inputs (inputs require a pull-up resistor to work). The I/O expander has an interrupt output line that indicates changes of data at the input pins, and 16 general purpose I/O ports. Both circuits accept a power supply of 3.3 V.

5.1.2.5 Hookswitch
A hookswitch commonly used in telephones by Häger Scandinavia AB has been selected. It has two poles and outputs both for open and closed states.

5.1.2.6 LCD module
The required LCD module should meet the following requirements:
- 4 rows with 20 characters each
- Backlight
- Good viewing characteristics
- Black characters on a white background

An LCD module has been chosen successfully from the above requirements. The selected module is a transreflective STN LCD module with 20 x 4 black characters on a white backlit background. Its supply and I/O voltage is 5.0 V, which is not completely
compatible with the rest of the system. Further research showed that the LCD module accepts 3.3 V as a logic “1”. The compatibility is evaluated further in the hardware design chapter. The LCD controller chip on the module is the NT3881D, and the software interfacing the display must take this into account.

5.1.2.7 Switches
Switches have been chosen to achieve a feeling of quality and robustness when used. The operating force of a switch $F$ has been evaluated in cooperation with an industrial designer for this purpose. It has been decided that the desired value of $F$ should be about 1.6 N. A switch with $F = 1$ N was finally selected because of the availability.

5.1.3 Hardware design
The actual design of the hardware includes work done on schematic capture and layout design of the printed circuit boards.

5.1.3.1 Development environment
The hardware has been developed using Altium DXP 2004 for schematic capture and PCB layout.

5.1.3.2 Schematic capture of the motherboard
Some components could be found in existing libraries for Altium DXP, other components had to be created in a new library.

The schematic has been divided into smaller schematic documents to simplify the design process. The top level schematic is illustrated below:

![Figure 5: Top level schematic of the motherboard](image_url)
Supply voltages in a mixed digital and analog system has to be handled with care because digital circuits generate noise that can propagate through the supply voltage line to other circuits. The following measures has been taken in the design:

- Separate voltage regulators for analog and digital supply voltages to keep the digital noise from travelling to analog circuits through the supply voltage line.
- Separate ground nets for analog and digital circuits, where the digital ground is connected directly to the ground of the voltage regulators. The two ground nets are connected in only one place to make sure that noisy digital return currents cannot travel into the analog ground net and back again and thereby inducing noise currents in the analog circuits. The connection is made with a ferrite bead used to block high frequency noise. Ferrite beads are also used to filter the analog supply voltages.

The power supply consists of the following voltages/regulators:

- 5 V (digital) – Supply voltage for the LCD module and LEDs
- 3.3 V (digital) – The digital supply voltage
- 3.3 V (analog) – The analog supply voltage
- 1.8 V (digital) – Core supply voltage for the CODEC

The analog supply voltage is divided into two nets, VDD_A and VDD_DRV, where the first voltage is the supply voltage for the analog part of the CODEC while the second voltage is used by the power amplifiers in the CODEC and the external power amplifier. These two voltages come from the same regulator and both use a serial ferrite bead to block high frequency interferences.

The power supply input port of the application system uses a DC plug connector followed by filter capacitors, and a diode to prevent an erroneous connection of supply voltage to damage the circuits. Each regulator has two filter capacitors connected close to its package. LEDs are connected to all regulators, except for the 1.8 V regulator, to indicate the status of the regulators.

The required input voltage for the system has been calculated: the dropout voltage of the 5 V regulator is about 1.2 V at its maximum output current. Since a diode is connected between the voltage input and the regulator, some of the voltage falls over the diode. The forward voltage of the diode is about 1.1 V. This yields an input voltage of

\[5.0 + 1.2 + 1.1 = 7.3 \text{ V} \]

The diode is used for safety reasons only, and can probably be removed in a future product. In such a case, the required input voltage will be 6.2 V. The voltage supplied to the voltage input jack of the prototype is 7.5 V.

The connections of the CODEC have mostly been based on reference design schematics from its data sheet. It has been assumed that the type of microphones used in headsets today are condenser microphones, and the design has been adapted for such
microphones only. Another choice regarding the microphone had to be made: single-ended or pseudo-differential mode. A jumper switch has been added, so that both modes could be evaluated at a later time.

The voltage supplies are filtered with multiple capacitors close to the CODEC, and it is possible that all filter capacitors are not really needed. This can be evaluated by multiple measurements where different amounts of capacitors are present on the circuit board.

The clock source for the CODEC was not determined at start, since there existed two alternatives: the on-board clock oscillator and a bus clock output from the processor. This decision could not be made during the circuit design, and a jumper switch has thus been added so that the choice of clock source cold be made later on. The CODEC can operate in two modes, master mode and slave mode. This heavily depends on whether the CODEC should be a clock master or not and thus depends on the choice of clock source. A jumper switch has been added here as well.

Signals between the CODEC and the processor are:

- SDA – I²C data line
- SCL – I²C clock line
- RESET
- SCLK – Serial data clock line
- FS – Frame sync signal to synchronise the arrival of data words
- DOUT – Data output line from CODEC
- DIN – Data input line to CODEC
- MCLK – Master clock signal routed through a jumper switch

The supply voltages of the CODEC have been selected as follows:

<table>
<thead>
<tr>
<th>Voltage type</th>
<th>Input voltage range</th>
<th>Selected voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog supply</td>
<td>2.7 – 3.6 V</td>
<td>3.3 V</td>
</tr>
<tr>
<td>Digital supply</td>
<td>1.65 – 1.95 V</td>
<td>1.8 V</td>
</tr>
<tr>
<td>I/O supply</td>
<td>1.1 – 3.6 V</td>
<td>3.3 V</td>
</tr>
</tbody>
</table>

*Table 4: Selection of supply voltages for the CODEC*

The I/O supply voltage could also have been chosen to 1.8 V, but 3.3 V has been chosen since this also is the I/O voltage on the processor evaluation board.

The connections of the power amplifier are based fully on a reference design schematic from its data sheet. One obvious drawback of that design is the fact that it is single-ended, which is not optimal when designing to reduce noise in the amplifier, since the differential mode eliminates noise induced in the signal path to the amplifier. One benefit of using a single-ended signal is that only one signal line is required. Since the
signal path to the amplifier will be quite short (< 10 cm) the amplitude of induced noise
is assumed not to be alarmingly high, so the single-ended connection will be given a try.

The amplification is determined by two resistors, and one of these (the feedback
resistor) has been chosen to be a potentiometer, so that the amplification can be
evaluated at a later time. The following parameters were decided for the amplifier:

- \( R_{in} = 20 \, \text{k}\Omega \)
- \( R_{load} = 8 \, \Omega \)
- \( P_{out,\, \text{max}} = 62.5 \, \text{mW} \) (average power determined through subjective testing using a
  speaker and a signal generator)

According to the data sheet, the output power can be 80 mW at most to obtain a value
for the total harmonic distortion + noise (THD+N) of 4% when using a supply voltage
of 3 V. There are two reasons for choosing a supply voltage of 3.3 V for the amplifier.
The first reason is that the CODEC already uses 3.3 V for its analog supply voltage and
it is wise to take advantage of the existing voltage regulator for this purpose. The other
reason is that the supply voltage has to be compatible with the shutdown signal from the
rest of the system, which uses 3.3 V as digital I/O voltage. It was decided that the
amplifier should share the analog supply voltage of 3.3 V with the CODEC.

The clock oscillator circuit consists of two discrete digital inverters, a crystal, two
resistors and two capacitors. Four filter capacitors are connected to the supply voltage
close to the inverters to eliminate noise from the oscillator so that it does not propagate
through the supply voltage to the rest of the system. The amount of filter capacitors
should be evaluated in a future design and it is assumed that this amount can be
decreased without affecting the system performance negatively. The desired serial bit
clock rate is 256 kHz (16 bits at 16 kHz), so this frequency must be a factor of the
crystal frequency. This constraint has been used when the crystal was selected. The
selected crystal frequency was 4.096 MHz and it can be divided by 16 to give 256 kHz.

The hookswitch connection header is used to connect the motherboard to the
hookswitch board through a planar cable.

The board interconnection header is used to connect the interface board and the
motherboard both electrically and physically. This header distributes the following
signals between the two boards:
- Hookswitch on/off
- 5 V (digital)
- 3.3 V (digital)
- Digital ground
- SCL
- SDA
- PWM (optional PWM source for white LEDs)
- Speaker on/off
- Keyboard interrupt

The EVB connection header is used to connect the whole application system to the processor evaluation board through a planar cable. To eliminate crosstalk between wires containing high frequency signals, almost each odd-numbered wire is connected to digital ground. The signals in this header are:

- Digital ground
- Keypad interrupt
- Optional PWM signal for white LEDs
- CODEC reset
- Optional master clock for CODEC
- Serial clock for CODEC
- Frame sync signal for CODEC
- Data output from CODEC
- Data input to CODEC
- SCL clock signal for the I²C bus
- SDA data signal for the I²C bus
The hookswitch has to control which microphone that should be active depending on the status of the telephone receiver. It is not obvious how this should be solved, and several options arise:

- Using an analog multiplexer controlled from the processor
- Using multiple input channels on the CODEC
- Routing the bias voltage to the selected microphone directly through the hookswitch

The multiplexer option has not been chosen because the smallest multiplexer found has 8 channels and 16 pins, and this was considered unnecessary because of the price and the space occupied by such a circuit. Using one CODEC input for each microphone would be the best alternative if the CODEC had two microphone inputs, which is not the case here. The existing analog inputs are two plain analog input ports and one microphone input port. Thus, one solution is to connect one of the microphones to the microphone input and connecting the other microphone to a plain input port. This may require an additional integrated amplifier for the microphone connected to the plain input port.

The method of using the hookswitch to route the bias voltage to the selected microphone is based on the fact that a condenser microphone needs a bias voltage to operate. This is is a clever alternative, since it does not introduce any additional components. The hookswitch used here will do the job since it is both double pole and double throw. One possible drawback of this solution is that the bias voltage has to travel through a cable from the system to the hookswitch board and back, and this cable is susceptible to interferences which are added to the bias voltage. This is a potential problem and it was not established during the design phase whether the induced interferences would stay on an acceptable level or not. It was finally decided that the last alternative will be used since it requires the least amount of additional components.

5.1.3.3 PCB layout of the motherboard

The prototype circuit boards were manufactured in the PCB lab at Linköping Institute of Technology, Campus Norrköping. This lab has certain process parameters that has to be taken into account when designing a circuit board:

- Minimum line width: 254 μm (200 μm under lucky circumstances)
- Minimum hole size: 400 μm
- Number of layers: two
- Through-hole plating
The constraints in the PCB document have been set:

- Minimum line width: 300 μm
- Minimum spacing between lines: 200 μm
- Vias:
  - Hole size: 500 μm
  - Diameter: 1.27 mm

First of all, each component in the motherboard was assigned a matching footprint. Most of the footprints have been created from start using the data sheets as a source of information. The footprints have been designed such that the surface mounted components would be easy to solder by hand. This has been made possible by expanding the pads and thus creating a larger surface for soldering. The through-hole footprints have been adapted to compensate for scaling errors in the drilling. This has been accomplished by enlarging both the hole size and the pad width.

The footprints of the voltage regulators require some extra attention, because there is a potential need of integrating heatsinks. The following discussion is based on the recommendations in the data sheet of using a heatsink for the voltage regulators. A heatsink is required for a voltage regulator if its required junction-to-ambient thermal resistance $\theta_{JA}$ is below $136 \, ^\circ\text{C}/\text{W}$. The following figures has to be defined before further discussions.

Power dissipation, $P_D = (V_{IN} - V_{OUT})I_L + V_{IN}I_G$

Estimated power dissipation, $P_{D}' = (V_{IN} - V_{OUT})I_L < P_D$

Input voltage, $V_{IN} = 7.5 \, \text{V}$.

Maximum allowable temperature rise, $T_{R,\text{max}} = T_{J,\text{max}} - T_{A,\text{max}}$

Maximum junction and ambient temperature, $T_{J,\text{max}} = 125 \, ^\circ\text{C}$, $T_{A,\text{max}} = 60 \, ^\circ\text{C}$

Junction-to-ambient thermal resistance, $\theta_{JA} = \frac{T_{R,\text{max}}}{P_D}$

The value of the maximum ambient temperature is taken from the primary design requirements of this project.
As can be seen in the table above, the required junction-to-ambient thermal resistance for the 5.0 V-regulator lies below 136 °C/W, so a heatsink must be used for this regulator. A rectangular double-sided heatsink with the dimensions 7 mm x 8 mm has been created as a part of the footprint for this voltage regulator.

Surface mount packages have been chosen for all components as far as possible. The exceptions are connectors, headers, jumper switches and potentiometers. Examples of selected package types are:

- Resistors and capacitors: 0603 (0.06” x 0.03” = 1.5 x 0.8 mm)
- CODEC: DBT package with a pitch of 500 μm and a pad width of 220 μm.

The package sizes have been chosen as small as possible but still at a size where components can be placed by hand. The reason for selecting small packages is to learn about modern package types and to be able to create a compact and realistic prototype design. The CODEC is the most tricky component to handle, both during placement and during the manufacturing of the motherboard, since it requires very thin spacing between lines. It is, however, preferable to have too small spacing and wider lines than having thin lines and larger spacing because thin lines can break while copper can be removed easily from wider lines after the board is manufactured.

Before any components can be placed, the edge of the board must be drawn. This has been achieved by using dimensions measured on a model of the exterior design, so that the board should be able to fit into a future plastic rapid prototype. The next move is to update the PCB document with the components. When this is finished, Altium DXP places all components in the PCB document in a more or less random fashion.

When the PCB document is updated with the components, floor planning is the next move. The placement of external connectors was determined first, because this was more or less decided by the exterior design prototype. The floor planning flow can be described as follows:
1. Placement of connectors that affect the exterior design
2. Placement of power supply circuits close to the DC input connector
3. Placement and rotation of the CODEC
4. Placement of board interconnection header
5. Placement of external components for the CODEC
6. Placement of power amplifier and crystal oscillator
7. Placement of jumper switches

Another thing affecting the floor planning is the fact that separate ground planes are used for analog and digital ground nets. This implies that the analog components have to be placed within an area with borders that can be drawn using simple lines (the analog part of the board). The same applies to the digital part. This means that when ground planes are placed, the border between the planes must be possible to draw. If any analog components are placed inside the digital area, the border will be very hard, or even impossible to draw.

Some basic design rules have been used when designing the PCB:

- Two lines should not run in parallel close to each other and if this is necessary, the parallel distance has to be as short as possible. This applies to lines on the same layer, but especially on lines running on different layers. This design rule eliminates crosstalk.
- If two lines cross, they should do this perpendicularly to eliminate crosstalk.
- Lines should not be drawn in arcs, since a circular line behaves like an inductor.
- Small capacitors should be connected between the supply voltage and ground close to each package, so that interferences created in each circuit are eliminated at the source. This should be combined with a larger capacitor in parallel to keep the supply voltage stable when temporary higher currents are drawn by the circuit.
- Whenever possible, via holes should be avoided.
- Ground planes for both analog and digital ground nets should be used to shield the system from both external and internal interferences.

The rules for placing lines and via holes have been followed as far as possible while the last rule could be followed completely. Another decision regarding the ground planes also had to be made. A ground plane can consist of either a completely filled copper area or a pattern, in this case a grid. One advantage of the grid is that the capacitance between two ground planes (top layer and bottom layer) will decrease if it is used. After looking at various graphic cards and sound cards for PCs, the conclusion was that that
the filled ground plane is the most common type used. This lead to the decision of using filled ground planes. According to the CODEC data sheet, the analog and digital ground nets should be connected as close to the CODEC as possible, and the surface beneath it should be covered with the analog ground plane. This was easily accomplished when placing the ground planes.

The layout and placement of components had to be evaluated before a feasible solution was obtained. Finally, a complete layout was accomplished. The next step was to place the ground planes and the ground via holes to connect the ground planes on both layers. It is important that the ground planes are connected in many places throughout the board, so that a return currents can travel the shortest possible path to ground. Finally, a ground plane was placed on the entire board. This ground plane had to be split into two planes, the analog and digital ground planes. A border between the two planes was drawn in a way that divides the analog and the digital part of the board. This process has been repeated for both layers.

5.1.3.4 Schematic capture of the interface board

Some components could be found in existing libraries for *Altium DXP*, other components had to be created in a new library.

The schematic has been divided into smaller schematic documents to simplify the schematic design process. The top level schematic is illustrated below:

![Interface board schematic](image)

*Figure 6: Top level schematic of the interface board*

The power supply for this board comes from the motherboard. Since this board only consists of digital circuits, only one ground plane has been used here.

All functions on the interface board are controlled by its two I²C transceivers which are both connected to the same bus as the CODEC. The bus addresses of the transceivers have been set by connecting three pins: A0, A1 and A2 to unique combinations of $V_{DD}$ and ground for each transceiver.

The I/O expander has been used for the LCD module, the rows of the switch matrix, and the hookswitch. Its interrupt output line has been connected directly to an interrupt input port on the CPU via the board interconnection header and the EVB connector. The LED driver has been used to drive and to fade the LEDs and the LCD backlight, controlling
the columns of the switch matrix and to turn the speaker amplifier on and off. The power supply of both I2C circuits is decoupled with three capacitors at both circuits, as close as possible to the packages.

It was not obvious how to connect the LCD module to the I/O expander, since it was not determined whether all signals should be used or not. The signal in question is the R/W signal which is used to set the direction of the data transfer between the system and the LCD module. Since the LCD is an output device, most data is transferred to it, not from it. There are, however, some exceptions:

- When the processor reads the busy flag to find out if the LCD module is ready for a new transfer
- When the processor reads the address counter to receive the position of the LCD cursor
- When the processor reads the data located at an address (a character code or a row of pixels of a user-defined character)

The first two alternatives can be useful: waiting for the busy flag to change to “not busy” before communicating with the LCD module assures that the software will not wait any more than what is necessary. Software often needs to know the position of the LCD cursor, and reading it directly from the LCD module is a convenient way of solving this. There are two drawbacks of using a bidirectional data port:

- Badly written software can turn both the LCD port and the I/O expander port into outputs. Such a case results in a short-circuit on the lines where the LCD pin and the corresponding I/O expander port pin does not have the same output level, which will damage the circuits.
- The R/W signal requires an extra I/O line, which could be used for something else.

It was finally decided that the R/W line should be connected, so that it could be evaluated at a later time.

The supply and I/O voltage of the display is 5.0 V, which does not match the rest of the system which uses 3.3 V. As mentioned before, the LCD module accepts 3.3 V as a logic “1”. The remaining problem is whether the I2C I/O expander can handle an input voltage of 5.0 V. It's data sheet states that it has got “5 V tolerant I/Os”. It also says that the input voltage may be raised above $V_{DD}$ to a maximum of 5.5 V. With this information in mind, it is safe to say that the LCD module can be connected directly to the I/O expander even if its supply voltage is 5 V.

A potentiometer has been connected between $5V_{DD}$ and ground to deliver an adjustable voltage to the display to be able to change its contrast.

All LEDs are connected to $5V_{DD}$ through resistors with values that allows the light to have an intensity that is expected to be perceived as comfortable for a viewer. The
reason for using $5V_{dd}$ as the driving voltage for the white LEDs is that their forward voltage is about 3.5 V, so the driving voltage must be higher than 3.3 V. The remaining LEDs could have been driven with 3.3 Volts, but have been connected to $5V_{dd}$ because of the following reasons:

- Using a LED driving voltage that is higher than the supply voltage for the driver IC decreases the power consumption in the LED driver when the LEDs are off, according to the data sheet of the LED driver.

- The power dissipation for a certain current is lowest when it is drawn through the voltage regulator having the highest output voltage $V_{out}$, since $P_D = (V_{in} - V_{out})I_L + V_{in}I_G$ grows with higher load current $I_L$ and smaller output voltage $V_{out}$. The term power dissipation refers to the power dissipated in a voltage regulator.

The current limiting resistor for the LCD backlight LEDs has been chosen to a value that limits the current to the highest allowable value that the LED driver circuit can sink. This value is lower than the highest allowable diode current of the backlight LEDs, and increasing the backlight current and thereby its intensity any further would require additional components. Driving the backlight with the LED driver circuit has been considered to be the best option because it does not require any additional components. The achieved intensity will be evaluated for decisions regarding future versions of the hardware that may contain an extra transistor to drive the backlight LEDs.

The white backlight LEDs for the telephone body are driven by a common DFET N-channel transistor. A jumper switch is used to select which signal source that should control the transistor of the telephone backlight LEDs. The available signal sources are:

- One channel of the LED driver circuit
- A PWM output channel on the processor

Both options have been kept for evaluation purposes. The main reason for using a PWM output channel on the processor is that fading of the telephone backlight can be carried out without any data transfer on the I²C bus. An I²C data transfer takes much more time than just setting the ratio of the processor's PWM channel. This means that the highest possible fading rate is decreased in the I²C case. The LED driver has 256 intensity levels, while the PWM channel of the processor has a maximum of 65536 levels. This also implies that the resolution is increased if the processor's PWM channel is used. The issue to evaluate here is whether a higher resolution really is necessary. If it is not, the LED driver can be used. The advantage of this solution is that it eliminates the need of another signal from the processor to the system, thus simplifying the layout. These two options will be evaluated further in software, since they are accomplished in different ways on a programming level.

The telephone backlight LEDs are connected to the circuit board with thin cables connected to terminal blocks at the circuit board. This allows the position of these LEDs to be evaluated.
All switches except the hookswitch are connected in a switch matrix of 4 rows and 7 columns. Every row has a common line that is connected to the second pin of each switch in the row. In a column, a common line is connected to the first pin of each switch in the column. To obtain both logic output levels, pull-up resistors are connected between every row and $V_{DD}$, allowing a row to have a logic one when no switch is pushed. Each row serves as a digital output to the I/O expander while each column is a digital input to the switch matrix connected to the LED driver that selects which column to read. A column is selected when the corresponding output pin on the LED driver has a level of 0 V.

Since the hookswitch is pressed constantly when the receiver is “on hook”, this will indicate that the corresponding switch matrix row is activated. This will inhibit the I/O expander to see when other switches on the same row are pressed, because the row is already active. Because of this, the hookswitch is connected to a dedicated input on the I/O expander, so that its operation does not affect the switch matrix.

5.1.3.5 PCB layout of the interface board

The same restrictions for creating a PCB layout as for the motherboard has been used. The shape of the interface board PCB is critical, since it must fit into a plastic casing based on the exterior design study. This board is placed at the front, and many parts of it will interact with the design of the casing much more than the motherboard PCB does. The shape of the interface board PCB was determined by using measurements made on a hand-made model of the telephone body design. The board is shaped like the front of the model, and has margins to be able to fit into a casing. All non-linear shapes have been approximated using arcs.

Most footprints have been reused from the motherboard PCB, while some new footprints have been created. The footprint of the LCD module contains both a header for the connection, and also the shape of the module and its mounting holes. Since components can be placed on the PCB underneath the LCD module, some clearance rules in the CAD software had to be ignored while designing this PCB.

The placement of components depend heavily of the exterior design, since most components on this board is a part of the user interface. The placement of the LCD, switches and LEDs has been decided in cooperation with an industrial designer. This board has got much unused space because of its design requirements, so the placement and connection of components on this PCB is much easier than on the motherboard PCB.

5.1.3.6 The hookswitch board

This is a very small and simple circuit board containing the hookswitch and soldering pads for connecting a planar cable directly to the motherboard. One of the internal switches redirect the bias voltage to one of the microphones while the other internal switch only is used to give a logic signal to indicate whether the receiver is lifted or not.
5.2 Software

Most software developed in this Master's thesis work is drivers for various parts of the hardware. This chapter covers the development of driver software for the telephone prototype, and also some application programming.

5.2.1 Development environment

The development environment consists of two computer systems: the host system and the target system. The host system is the computer where the software is written and compiled, while the target system is the embedded computer where the software is executed.

5.2.1.1 Host system

The host system consists of a PC running Linux (Kubuntu 6.06 Dapper Drake). The compiler environment comes from Freescale Semiconductor and is pre-configured for the MCF5329 Coldfire processor. Simple applications are compiled from the terminal by typing

```
m68k-uclinux-gcc code.c -o code
```

Advanced applications and Linux device drivers are compiled using a Makefile that is executed in the terminal:

```
/code/module/sound$ make
```

The Makefile contains various parameters for the compiler. This enables the use of multiple source files being compiled by a single command. This is compulsory when compiling device drivers, since they differ in many ways from application software.

The code is written using Kate, which is an advanced text editor that comes with the Kubuntu Linux distribution. More advanced development software suites like Eclipse and Kdevelop have been considered, but have not been chosen. The reason for this is the fact that the applications and drivers written in this thesis project mostly consist of one or two source files. The learning period of such a development software suite was considered too big compared to its advantages.

To communicate with the target system, a terminal program called minicom is used. This enables terminal access to the target's operating system just like accessing the terminal on the host.

The file system of the target is located on the hard drive of the host system. This is made possible by running an NFS server on the host and making sure that the Ethernet interfaces of both the host and target can communicate (an Ethernet switch has been used here). NFS is an acronym for Network File System and enables a folder on one computer to be accessed like a local folder on another computer, as long as they are connected via a local area network (LAN) or the Internet. The computer hosting the file system must run a NFS server application, while the other computer has to run an NFS client application.
5.2.1.2 Target system

The target system consists of an evaluation board with the MCF5329 Coldfire processor from Freescale Semiconductors connected to the prototype hardware designed in this thesis project. The uClinux operating system, version 2.6.16 is used as the operating system.

The target system has the Freescale dBug bootloader (version v4b.1a.1d) pre-loaded into its boot flash memory. The bootloader starts automatically when the target board is powered up. The bootloader is a small program that enables the developer to run programs and to debug applications. In this thesis project, the bootloader is used to download the operating system into memory and to start it.

The operating system is loaded into SDRAM by typing `dn` in the target terminal window. This operation transfers the operating system from the host to the target via the network using a TFTP (trivial FTP) service. When the transfer is completed, uClinux is started by typing `go 0x40020000` in the target terminal window. The `go` command executes the code located at the given address, where the operating system is located.

The device driver system is depicted below. The thin lines illustrate communication between the drivers while the wider lines illustrate serial data transfers between the processor and the circuits in the prototype. Each device driver has its file system node depicted above it.

![Device Driver System Diagram](image)

*Figure 7: The device driver system*

At first, driver code was designed in userspace applications to simplify the testing. Normally, it would be cumbersome to access registers in the processor from userspace, but it is easy to do here since no memory protection is present (no virtual memory).
Header files from Freescale Semiconductor containing address definitions for registers have been used to simplify the programming. An example of such a definition is

```
#define MCF532x_I2C_I2DR (volatile u8 *)(0xFC058010)
```

This row defines a pointer to the I²C data register which is eight bits wide.

### 5.2.2 I²C driver

The development of this driver has delayed the whole project more than expected. The reason is that no I²C code – no matter the design, did work at all. It was finally determined that the error was located somewhere in the hardware. Measurements were made, and it was clear that one signal did not behave as expected: the SDA (serial data) pin. After many measurements and lots of thinking it was finally established that the SDA pin of the processor has an alternative function as the receive pin for the processor's CAN module. Because of this, that pin is connected to the output of a CAN transceiver circuit on the evaluation board which can not be disabled or disconnected in any simple way. This explains the malfunction of the pin, since two logic outputs never should be connected to each other. The problem was finally resolved by physically removing the corresponding pin on the CAN transceiver IC.

Efforts have been made to use the device driver from the BSP supporting the Linux I²C subsystem. It was finally determined, however, that creating a new simpler driver would be easier because of the complexity of the Linux I²C subsystem. The complexity arises from the fact that this subsystem is made very portable to work in many processor environments. This driver should only work for the Coldfire CPU used here, so such portability is not necessary.

This driver is not attached to a device file and is only accessed by methods exported to the kernel symbol table. The following methods have been created and exported:

- **i2c_sendByte(u8 data, u8 address, u8 id)**
  
  This method transmits `data` to the register at `address` at the device with the address `id`.

- **i2c_receiveByte(u8* data, u8 address, u8 id)**
  
  This method receives data to the address of the `data` pointer from the register at `address` at the device with the address `id`.

- **i2c_receiveBytes(int n, u8* data, u8 address, u8 id)**
  
  This method receives `n` bytes of data to the address of the `data` pointer from the register at `address` at the device with the address `id`.

Early versions of this driver were based on code found on the Freescale forums for the 68K/Coldfire microprocessors. The current version is optimised and built to follow the I²C programming examples of the processor's data sheet.

One mutex semaphore has been used to restrict the access to the exported methods, so that no I²C method can be called as long as any other I²C method is executing. Since it is forbidden for a process to sleep in atomic context (e.g. interrupt handlers), a check is
performed before the mutex is taken. If the method is called from atomic context and the mutex is already taken, an error code is returned instead of putting the process to sleep. If the mutex is not yet taken, the process will not sleep anyway, so the mutex is taken and the method is executed.

When necessary, delays have been inserted in the code to wait for the I²C hardware to finish a task. In both receive and send methods, `udelay(4)` have been called once to generate a delay of approximately 4 microseconds. This delay is needed when I²C methods are called sequentially without any delays in between.

When a byte is being transferred, the driver polls the I²C interrupt flag to discover when the transfer is finished. Since polling takes up more system resources, an interrupt driven approach would be more preferable. In such a case, the process is put to sleep in a wait queue and awakened by the I²C interrupt. This would be forbidden if called from atomic context. The interrupt driven approach has not been used here because the solution of polling works, and because of efforts to make the code simple. Using interrupts is a possible way of optimising the code further in future versions.

5.2.3 Keypad driver

The main purpose of this driver is to receive interrupts from the keypad, scan the keypad and finally return the corresponding ASCII codes to applications reading from this device driver. The driver is accessed through `dev/keypad`. The `read` method is used to receive characters into a buffer and to copy it to the reading application. The calling method is put to sleep if data is not available (blocking read).

The following two methods have also been implemented:

- **ioctl** - The following commands are accepted:
  - `KEYPAD_SET_CHARTABLE` - uploads a character table for multi-tap character input.
  - `KEYPAD_UPPERCASE` - puts the driver in a upper case character mode.
  - `KEYPAD_LOWERCASE` - puts the driver in a lower case character mode.
  - `KEYPAD_NUMERIC` - puts the driver in a numeric input mode.
  - `KEYPAD_TELL_TIMEOUT` - sets the time out value for multi-tap character input.

- **poll** - checks whether data is available and provides a possibility for the calling process to sleep until data has arrived.

Switches are debounced by letting the scanning process sleep for 14 milliseconds before the scanning is executed. The keypad is scanned by selecting a column and reading the rows. This process is repeated for all columns. This method is executed when a keypad interrupt has arrived. It is executed in a work queue by the interrupt handler, allowing the key scan method to sleep.
When operated in character mode, a multi-tap mode is used. Similar functionality is found on cellular phones: many characters are available at every key. The set of characters for each key is stored in a customised file. A small application for reading such a file and transmitting the contents to the keypad driver has been created. Characters can not be previewed when using this feature and if this is desirable, the multi-tap feature must be included in the application and not by the driver. In such a case this driver is set to numeric mode.

The multi-tap time out is implemented using a kernel timer.

This driver exports one method, `keypad_set_pin_LS1(u8 pin, u8 data)`. The reason for exporting this method is that it is used by the sound device driver to control the power amplifier.

### 5.2.4 LED driver

The purpose of this driver is to control all LEDs in the application system. This driver is accessed through the `dev/led` node in the file system. The following `ioctl` commands can be called for each LED:

- `LED_TELL_FADE_ON`
- `LED_TELL_FADE_OFF`
- `LED_TELL_BLINK`
- `LED_TELL_ON`
- `LED_TELL_OFF`

The first three commands use PWM generators for fading the transitions. The fading rate and resolution is set in the driver and can not be accessed from applications.

The LED driver circuit has got two PWM sources. Each LED can be driven by one of the PWM sources or a constant level of "on" or "off". This also means that a maximum of two LEDs can be faded simultaneously.

The fading is managed by using a kernel timer to obtain regular updating intervals.

Each LED is represented by using a structure containing all necessary information for controlling it.

### 5.2.5 LCD driver

The purpose of this driver is to receive characters from applications and to print them on the LCD display module. User defined characters can be transferred to the driver and printed on the display as ordinary characters. This driver is available through the `dev/lcd` node in the file system.
The `write` method simply receives characters for printing and transmits them to the display. `ioctl` calls implemented in the driver are

- `LCD_CLEAR`
- `LCD_ON`
- `LCD_OFF`
- `LCD_SET_POS` – sets the position of the cursor.
- `LCD_CURSOR_ON`
- `LCD_CURSOR_OFF`
- `LCD_STEP_FORWARD`
- `LCD_STEP_BACK`
- `LCD_SET_CHARACTER` – stores a user-defined character in the LCD module.

The driver implements the new line character `\n`, so that when it is sent to the driver, a new line is created.

### 5.2.6 Sound driver

The aim is to create a sound driver being compliant with a known standard. This means that the choices are OSS and ALSA for Linux systems.

ALSA was considered at first because of its growing popularity, but finally OSS was chosen because of its simpleness and the limited capabilities of the sound system used in this project.

Another decision to make is whether memory-mapped transfer or read/write transfer should be used in the driver. According to [8], memory-mapped mode is suitable for real-time applications with small buffers which are more sensitive to transfer-related latency. However, memory-mapping is not completely supported on systems without virtual memory. For example, write-enabled mappings are not supported on uClinux [2].

Another reason for not choosing memory-mapped mode is that it is simply not recommended by 4Front Technologies, the maintainer of the official OSS drivers. They state that using the memory-mapped mode only makes an application 0.01% faster than with the normal read/write method and that it should only be used in custom applications for special purposes where it is possible to select a sound device that supports the method properly [9]. The conclusion is that read/write transfers are used here.

The OSS audio subsystem contains a generic layer to handle common tasks for a sound device driver. This layer has not been used here because it is adapted to larger systems. A workaround was considered simpler in this project.
A sound device driver compliant with OSS was finally designed. Both read and write calls have been implemented for recording and playback. The sound device driver is accessed through the dev/dsp node in the file system. Necessary IOCTL commands have been implemented to support the PortAudio API for interfacing sound drivers. A separate mixer driver has not been created since this was considered unnecessary for a one-channel system. Volume controls are implemented as IOCTL commands instead. The hardware specific controls are

- **SNDCTL_DSP_SPEED** – sets the sample rate of the CODEC
- **SNDCTL_DSP_SETPLAYVOL** – sets the DAC gain
- **SNDCTL_DSP_SETRECVOL** – sets the ADC gain
- **SNDCTL_HEADSET** – selects the headset as playback device
- **SNDCTL_RECEIVER** – selects the receiver as playback device
- **SNDCTL_SPEAKER** – selects the speaker as playback device

The common way to handle sound samples in memory is the Little-Endian (LE) format, since this is supported by Intel processors. Coldfire processors use Big-Endian (BE) format instead, and the driver has been given the ability to convert between these two formats so that both can be used.

The on-board crystal oscillator has been used as the clock source for the CODEC, since the processor bus clock does not have a suitable frequency for this purpose. The Synchronous Serial Interface (SSI) has been chosen as the physical interface between the processor and the CODEC, since this interface can use a clock source different from the processor bus clock.

Playback and capture of sound require a high rate of data transfers between the memory and the hardware. Leaving the processor to do this work alone would lead to a higher CPU load, which is bad for the overall system performance. Instead, DMA transfers have been used for this task. The Board Support Package from Freescale Semiconductor did not contain drivers for the processor's DMA module, so such drivers have been created in this project instead. The SSI module which interfaces the CODEC circuit has a FIFO buffer consisting of eight positions. This feature is used to decrease latency requirements when the DMA buffer is reloaded with new data. Using a double-buffering approach would also decrease the latency requirements, but it is assumed that the FIFO does the job well enough.

Connecting the CODEC to the SSI receive pin of the processor proved to be complicated, since the output of another CODEC on the evaluation board is connected to the same pin. The other CODEC could not be removed easily because of its BGA casing. The SSI receive pin of the processor has been moved to another pin by changing the contents of the corresponding pin assignment register in the driver software. The new pin was already connected to the output of another circuit, but the connected pin of that circuit could be physically removed.
5.3 Demonstration system

The hardware and software just described were tested together in a demonstration system. This was accomplished by running software which uses all functionality in the prototype.

The goal was to create a demonstration system having the following features:

- Linphone, an Open-Source SIP telephony application. A command-line version is available and should be used to simplify the integration.

- The ROMFS file system is used instead of the NFS file system. In practice, the on-board flash memory is used instead of the hard drive of a host PC. This eliminates the need of a PC for hosting the file system of the embedded Linux system. It also simplifies the connection to the Internet.

Test applications have also been created successfully during this project. Important applications are

- interface.c – tests and demonstrates the functionality of the user interface (LEDs, display and keypad).
- playback.c – plays a wave file from the file system.
- echo.c – records ten seconds of audio and plays it back.

Finally, an interface demo was created to demonstrate all prototype features except for the network connection.
6. Results / Verification and testing

This chapter covers the practical results of the work in this project. The results consist of observations on performance and the evaluations performed with or without jumper switches. Design mistakes and their solutions are also presented.

6.1 Processor execution speed

The execution speed for the MCF5329 processor is 211 Dhrystone 2.1 MIPS according to its data sheet [12]. It is assumed that this has been measured in a system running nothing but the Dhrystone benchmark application.

A new Dhrystone 2.1 benchmark was executed in this project in uClinux, to investigate if the operating system would affect the benchmark results. After configuring and compiling for the target system, the benchmark was executed. The result was 45 454 Dhrystones per second. Recalculated in Dhrystone MIPS (DMIPS) [13], the number is

$$\frac{45454}{1757} \approx 25.9 \text{ DMIPS}$$

It is suspected that this low result may depend on that the cache memory is not activated properly. This requires further investigation and is left as future work.

6.2 Hardware and software

The intensities of the LEDs have been evaluated, and all intensities have been perceived as expected except for the “alert” LED where it was far too high. The intensity of the LCD backlight was expected to be too low, but it seemed to be high enough.

Using an output of the I²C LED driver to drive the telephone backlight LEDs works fine, and using a PWM channel of the processor has not been chosen because of problems with the software. The PWM resolution is considered to be good enough for fading LEDs.

The CODEC circuit has been chosen to be both the SSI clock master and bus master, and its input clock signal is generated by the crystal oscillator on the application board. The microphone has been connected using pseudo-differential mode without any noticeable problems. Activating the microphones by routing the bias voltage through the hookswitch works fine, and no interferences induced in the planar cable have been noticed. The signal level of the microphone input was perceived as a bit low, even at the maximum internal amplification. This can be compensated for in software and has not been considered to be a problem.

Audio playback works fine in the headset, the receiver and in the speaker. Using the single-ended input of the power amplifier gives a fairly good result. One design mistake has been discovered at the CODEC: its reset line is sensitive to transients in the power supply. Connecting another device in the same power outlet induces a reset signal in the CODEC. The solution to this problem is to connect at least one capacitor between the...
reset line and ground to keep the CODEC from resetting itself during transients on the power supply.

The output volumes of the headset and the receiver can be set to a level that is perceived as high. The outputs for the headset and the receiver have been set to differential mode (this can be changed easily in software). The output volume of the speaker is a bit low when a proper headset volume is used, because of the amplification set by the potentiometer of the power amplifier. One solution for this is to change the value of the input resistor and the feedback resistor to obtain a suitable amplification. An alternative solution is to set the DAC gain of the CODEC to a higher value when only the speaker is used. Tests have shown that this gain cannot be set to its maximum value because this distorts the signal. A maximum value of about 1 dB of DAC gain has been estimated.

Different speakers have been connected to the power amplifier with very varying results. The conclusion is that speakers without a proper casing gives a poor result and distorts the sound easily. A speaker from a HiFi set gave a very good result, which excluded former suspicions about bad signal quality.

The keypad and the switches work as expected and require no further design efforts.

Using the LCD module without reading from it has worked fine. Because of this, a possible future solution is to connect the R/W signal to a constant “0” level.

All drivers designed for the telephone prototype have been tested successfully and only one is considered to be unstable: the I²C driver. Some I²C transfers fail, and it seems like adding more delay before each transfer minimizes the amount of failed transfers. The sound device driver does not work properly when the CPU is heavily loaded, and it is assumed that adding double buffering would solve this problem.
7. Conclusions

The prototype hardware and software has worked very well with only a few design mistakes. The selected components have proved to work in an expected manner.

It has proven to be easier writing a new driver than using a driver that is too advanced for the current purpose. In general, the simpler solutions have often been the best ones.

Another important conclusion is that it is wrong to assume that evaluation boards are free from design mistakes.

The choice of processor is very important and can be optimized in many ways. Since a processor containing an MMU supports virtual memory, this can prevent memory related bugs which can be very hard to solve. Even if this has not been evaluated fully, it is assumed that a processor using an MMU is the best choice for a complete telephony application. There has, however, not been any problems with the lack of MMU during this project.

Using an open source operating system has proved useful in many ways: it is free of charge and it has a big community of skilled and helpful people that like to help other programmers, mostly through mailing lists.

The workload of this project has been a bit too big for one person, and it should have been smaller, or shared with at least one more student.
8. Further work

There are several areas of further work for this project:

- Implementation of a complete telephony application using all hardware of the prototype
- Improvement of the I²C driver using interrupts instead of polling
- Improvement of the sound device driver with double buffering
- Evaluation of computational capacity for the current processor
- Evaluation of system requirements for applications and voice CODEC algorithms
- Evaluation of processors containing MMU and processors with DSP co-processors
- Evaluation of different amounts of filter capacitors

Adapting the telephone design into a real consumer product requires changes to be made in the hardware. Important changes are:

- The motherboard should also contain the processor and its peripherals
- The LCD module PCB should be integrated into the interface board
- TACT switches may be replaced by rubber pad switches, which is a cheaper alternative
- A new BSP must be created

There is much work remaining before a commercial product is obtained, but the achievements of this project is a very good start.
References

Printed sources


Unprinted sources


47
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/