Dulkóðuð leyndartut

Glenn Ruben Árthun Bakke
Pål Driveklepp
Åsmund Eldhuset
Olav Aanes Fagerlund
Stig Fjellskaalnes
Daniel Haugen
Olav Morken
Gunnar Rangøy
Øystein Ingmar Skartsæterhagen
Rolf Anders Syvertsen

November 23, 2007

Hofstadter's Law:
It always takes longer than you expect, even when you take into account Hofstadter's Law.

— Douglas R. Hofstadter

The road to wisdom? – Well, it's plain
and simple to express:
Err
and err
and err again
but less
and less
and less.

— Piet Hein

If it's not broken – fix it until it is!

— Anonymous

Abstract

This report documents the project work of ten students taking the course *TDT4295 Datamaskiner prosjektarbeid* (English: *Computers project work*) at NTNU in the fall term of 2007.

The goal of the project was to create two identical units able to communicate speech securely over a common mobile phone network. The project introduced a variety of distinct tools and design techniques. Most of these tools were new to most of the participants of this project. We got to experience important aspects of hardware design, notably Printed Circuit Board (PCB) design, the use and programming of a microcontroller and Field Programmable Gate Array (FPGA) logic design. The project emphasized learning of various techniques and the process of designing a system rather than an optimal design.

We produced two prototypes, as we intended. These prototypes unfortunately had problems with the audio. By the delivery deadline of this report, we could not set up a full duplex encrypted conversation due to problems with this part. However, the systems are capable of secure communication, by sending encrypted Short Message Service (SMS) messages back and forth.

Preface

The project must have a name, and to find a name that everyone could agree upon was rather difficult. Why did we choose Dulkóðuð leyndartut?

A somewhat more natural choice could be to base the name on such words as "cryptography" and "telephone". Indeed, many different variations over the basic idea of a portmanteau of these two words were suggested as the project's name, including most of the strings matching the regular expression [CK]rypto(ph|f)on(e?) – and even this regex itself. However, each of these had its eager proponents who refused to use any other variant; with the result that the early documentation was, regarding the name of the project, consistent only in being inconsistent.

In a final effort to find a name everyone could agree on, we decided to try to write our name in Icelandic, based on the observation that everything expressed in Icelandic sounds intriguing. Aided by dictionaries and one or two bursts of creativity, we finally arrived at the name Dulkóðuð leyndartut, and there was much rejoicing.

- dulkóðuð (Icelandic) – encrypted
- leyndar (Icelandic) – secret
- tut (quasi Icelandic) – telephone

Contents

1 Introduction **1**
- 1.1 Assignment . . . 2
- 1.2 Interpretation . . . 3
- 1.3 Requirements Specification . . . 3
  - 1.3.1 Functional . . . 3
  - 1.3.2 Non-Functional . . . 5
- 1.4 Prototype vs Production Model . . . 5

2 Background **7**
- 2.1 Historical aspect and the situation today; availability of solutions . . . 7
- 2.2 Encryption . . . 8
  - 2.2.1 Triple DES . . . 9
  - 2.2.2 AES (Rijndael) . . . 10
  - 2.2.3 Modes of operation . . . 10
- 2.3 FPGA . . . 13
- 2.4 VHDL . . . 14
- 2.5 ATmega microcontroller . . . 14
  - 2.5.1 External memory interface . . . 15
- 2.6 Audio . . . 15
  - 2.6.1 Analog/digital audio conversion . . . 15
  - 2.6.2 Analog audio . . . 15
  - 2.6.3 Compression . . . 16
- 2.7 GSM/GPRS . . . 16
  - 2.7.1 GSM . . . 16
  - 2.7.2 GPRS . . . 17
  - 2.7.3 Administration . . . 18

3 Design **19**
- 3.1 Process overview . . . 19
- 3.2 Main functional units . . . 20
- 3.3 Choice of main components . . . 20
  - 3.3.1 Audio . . . 21
  - 3.3.2 Encryption . . . 24
  - 3.3.3 Main controller . . . 27
  - 3.3.4 Transmission . . . 27
- 3.4 Power supply components . . . 31

3.5 Choice of Peripherals . . . 32
3.5.1 Display . . . 32
3.5.2 Keypad . . . 32
3.5.3 Flash memory . . . 33
3.5.4 Extra memory . . . 33
3.6 PCB . . . 34
3.7 Software design . . . 34
3.7.1 The menu system . . . 34
3.8 Casing . . . 34

4 Implementation **37**
4.1 Implemented features . . . 37
4.2 Physical interfaces . . . 37
4.2.1 AVR . . . 37
4.2.2 GSM module . . . 38
4.2.3 SD card . . . 39
4.2.4 FPGA . . . 41
4.3 Support electronics . . . 45
4.3.1 Power supply . . . 45
4.3.2 Oscillators and crystals . . . 45
4.3.3 Battery charger . . . 45
4.3.4 Reset circuit . . . 46
4.3.5 Analog audio processing . . . 46
4.4 PCB . . . 47
4.4.1 Layers . . . 49
4.4.2 Power Grid . . . 49
4.4.3 Signalling Layers . . . 50
4.4.4 Ground . . . 50
4.4.5 PCB Holes . . . 50
4.4.6 Soldered PCB . . . 50
4.4.7 Debug and workaround possibilities . . . 51
4.5 Workarounds . . . 53
4.5.1 Error in Capacitor Polarity Marking . . . 54
4.5.2 Error in Audio Circuit Design . . . 54
4.5.3 Error in Audio Output Plug Footprint . . . 54
4.5.4 Missing JTAG Connector for PROM and JTAG . . . 54
4.5.5 Oscillator . . . 54
4.6 AVR software . . . 57
4.6.1 SMS . . . 57
4.6.2 External memory interface . . . 58
4.6.3 Interrupts . . . 59
4.6.4 SD card . . . 59
4.6.5 LCD GUI . . . 60
4.7 FPGA implementation . . . 61
4.7.1 AMBE interface . . . 61
4.7.2 Keypad . . . 62

4.8 Casing 62

5 Testing **67**
5.1 Hardware 67
5.1.1 Audio 67
5.1.2 AVR 69
5.1.3 FPGA 69
5.1.4 Power 70
5.1.5 PCB card 1 70
5.1.6 PCB card 2 73
5.1.7 PCB card 3 73
5.2 Software 76
5.2.1 High-level test of main program 76
5.2.2 GSM 76

6 Tools **79**
6.1 PCB Design 79
6.2 VHDL 79
6.3 AVR programming 80
6.3.1 Software tools 80
6.3.2 Hardware tools 80
6.4 LaTeX 80
6.5 SVN 80
6.6 Laboratory tools 81

7 Discussion **83**
7.1 Security 83
7.2 Things we wish we had done 84
7.2.1 Splitting the VCC plane 84
7.2.2 In-System-Programming Connector 84
7.2.3 More GND 84
7.3 Audio codec chip 84
7.4 Group dynamics 85
7.4.1 Leadership 85
7.4.2 Groups 85
7.4.3 Difficult decisions 85
7.5 Research & Development 86
7.5.1 Budget 86
7.5.2 Making the final product 86

8 Conclusion **89**

9 Acknowledgements **91**
9.1 GSM 91
9.2 Course staff 91
9.3 Atmel 91

10 Abbreviations **93**

A Datasheets/Specifications **99**
A.1 GSM 99
A.1.1 Siemens MC75 99
A.1.2 Round Solutions GM862 101
A.1.3 Multitech SocketModem EDGE (based on Siemens MC75) 102

B Schematics **105**

C Pin mappings **116**
C.1 FPGA 117

D Budget **119**
D.1 Budget table 119

E GSM AT commands **121**
E.1 Commands to test 121

F Memory mapped I/O addresses **123**

G Code listings **125**
G.1 VHDL code 125
G.2 C code 205
G.3 C# 438

List of Figures

1.1 Diagram showing the role of the Dulkóðuð leyndartut 1

2.1 Top level structure of DES 10
2.2 Structure of the DES Feistel function 11
2.3 Structure of Triple DES 11
2.4 Encryption in cipher block chaining mode 12
2.5 Decryption in cipher block chaining mode 12
2.6 A FPGAs logic block 13
2.7 The pins of the logic block 13
2.8 Several logic blocks and the communication channels 14
2.9 PCM audio conversion 15
2.10 GSM architecture 17
2.11 GSM/GPRS architecture 18

3.1 Processes involved 19
3.2 The data flow through the system when audio is sent and received 20
3.3 Block diagram showing the main components and interconnections 21
3.4 Picture of the AMBE-2000 voice codec chip 22
3.5 Picture of a ATmega128 microcontroller 27
3.6 Picture of the Siemens MC75 GSM module 28
3.7 Top and bottom picture of the SocketModem EDGE module 30
3.8 Picture of the antenna recommended by Acte, the reseller of the SocketModem EDGE 30
3.9 Picture of a CAP-XX supercapacitor 32
3.10 Front and back picture of the GTC-40045-YS6L0S display 32
3.11 Picture of the numeric keypad 33
3.12 Diagram showing the menu system of the Dulkóðuð leyndartut. 35

4.1 The interfaces between the AVR and other parts of the system 37
4.2 Resistive divider diagram 38
4.4 Physical interface between AVR and LCD-display 39
4.3 Signal lines between the AVR and the GSM-module. All lines from the GSM to the AVR are indirectly connected via voltage dividers. 39
4.5 SD card interface reference design 40
4.6 The SD card interface. 40
4.7 The ATmega128L's physical interface for external memory 41
4.8 Physically interfaces of the FPGA and the other devices 42

4.9 PROM and FPGA programming lines . . . 42
4.10 Connection between FPGA and AMBE-2000 . . . 43
4.11 Physical interface between the PCM3500 and the AMBE-2000 . . . 43
4.12 Signal lines between FPGA and external memory . . . 44
4.13 Schematic of inner workings of the keypad . . . 44
4.14 LED and button connected to FPGA . . . 45
4.15 Typical crystal circuit . . . 45
4.16 Oscillator connected to the FPGA . . . 46
4.17 The reset circuit . . . 46
4.18 The PCB card . . . 47
4.19 The PCB with silkscreen top and holes . . . 48
4.20 Power Grid. . . . 49
4.21 Signal layer 1 — Top. . . . 50
4.22 Signal layer 4 — Bottom. . . . 51
4.23 Ground Grid. . . . 52
4.24 Card 1 with all the components. . . . 53
4.25 JTAG connections for the FPGA on the PCB backside . . . 55
4.26 The function of the different FPGA JTAG vias . . . 55
4.27 PROM JTAG connections on the PCB backside . . . 56
4.28 The function of the different PROM JTAG vias . . . 56
4.29 Picture of the oscillator workaround . . . 57
4.30 Memory space configuration . . . 58
4.31 Xmem interface timing diagram . . . 59
4.32 The menu . . . 60
4.33 FPGA units . . . 61
4.34 Images of the outside of Dulkóðuð leyndartut. . . . 63
4.35 Images of the inside of Dulkóðuð leyndartut. . . . 64
4.36 The Dulkóðuð leyndartut in use. . . . 65

5.1 Pictures of Card no. 1 . . . 71
5.2 Test setup for GSM-module . . . 77

List of Tables

4.1 FPGA and PROM workaround 54

5.1 Wave testing 68
5.2 Testing with mic 68
5.3 Testing with audio from PC 68
5.4 Testing of AMBE-2000 69
5.5 AVR tests 69
5.6 FPGA tests 70
5.7 Power circuit for PCB card 1 72
5.8 Audio test for PCB card 1 72
5.9 Circuits test for PCB card 1 73
5.10 User interface test for PCB card 1 73
5.11 Power circuit for PCB card 3 74
5.12 Audio test for PCB card 3 74
5.13 Circuits test for PCB card 3 75
5.14 User interface test for PCB card 3 75
5.15 Test for main program 76
5.16 GSM-module tests 78

E.1 AT commands used 122

Listings

G.1 top_level.vhd . . . 126
G.2 sram_controller.vhd . . . 134
G.3 tdes_control.vhd . . . 135
G.4 buttons_controller.vhd . . . 141
G.5 clock_divider.vhd . . . 143
G.6 codec_rx_channel.vhd . . . 144
G.7 codec_rx_control.vhd . . . 146
G.8 keyboard.vhd . . . 147
G.9 keyboard_controller.vhd . . . 149
G.10 keyboard_debounce.vhd . . . 151
G.11 testbench.vhd . . . 152
G.12 add_key.vhd . . . 162
G.13 add_left.vhd . . . 162
G.14 block_top.vhd . . . 163
G.15 des_cipher_top.vhd . . . 168
G.16 des_top.vhd . . . 171
G.17 e_expansion_function.vhd . . . 177
G.18 key_schedule.vhd . . . 178
G.19 p_box.vhd . . . 184
G.20 s1_box.vhd . . . 185
G.21 s2_box.vhd . . . 187
G.22 s3_box.vhd . . . 188
G.23 s4_box.vhd . . . 190
G.24 s5_box.vhd . . . 191
G.25 s6_box.vhd . . . 193
G.26 s7_box.vhd . . . 195
G.27 s8_box.vhd . . . 196
G.28 s_box.vhd . . . 198
G.29 tdes_top.vhd . . . 201
G.30 src/init.c . . . 205
G.31 src/crypt.c . . . 208
G.32 src/delay.c . . . 210
G.33 src/error.c . . . 210
G.34 src/fat16.c . . . 211
G.35 src/fifo.c . . . 258
G.36 src/fs.c . . . 269

G.37 src/fs.c . . . 272
G.38 src/jtag.c . . . 274
G.39 src/keyboard.c . . . 274
G.40 src/keyboard_int.c . . . 275
G.41 src/lcd.c . . . 277
G.42 src/memory.c . . . 289
G.43 src/menu.c . . . 289
G.44 src/message.c . . . 297
G.45 src/modem.c . . . 298
G.46 src/output.c . . . 309
G.47 src/partition.c . . . 309
G.48 src/phone_book.c . . . 312
G.49 src/scheduler.c . . . 316
G.50 src/sd_raw.c . . . 318
G.51 src/sleep.c . . . 336
G.52 src/sms.c . . . 337
G.53 src/spi.c . . . 346
G.54 src/tests.c . . . 349
G.55 src/thread.c . . . 352
G.56 src/thread_switch.S . . . 357
G.57 src/time.c . . . 360
G.58 src/timeout.c . . . 362
G.59 src/ui.c . . . 363
G.60 src/usart.c . . . 368
G.61 src/wait_queue.c . . . 372
G.62 inc/assert.h . . . 373
G.63 inc/avrlibdefs.h . . . 374
G.64 inc/avrlibtypes.h . . . 376
G.65 inc/config.h . . . 378
G.66 inc/crypt.h . . . 381
G.67 inc/delay.h . . . 382
G.68 inc/error.h . . . 382
G.69 inc/fat.h . . . 382
G.70 inc/fat16.h . . . 391
G.71 inc/fat16_config.h . . . 393
G.72 inc/fatconf.h . . . 395
G.73 inc/fifo.h . . . 396
G.74 inc/fpga.h . . . 397
G.75 inc/fs.h . . . 398
G.76 inc/global.h . . . 398
G.77 inc/jtag.h . . . 399
G.78 inc/keyboard.h . . . 399
G.79 inc/keyboard_int.h . . . 400
G.80 inc/lcd.h . . . 400
G.81 inc/memory.h . . . 400
G.82 inc/menu.h . . . 400

G.83 inc/message.h . . . 402
G.84 inc/mmc.h . . . 402
G.85 inc/mmcconf.h . . . 406
G.86 inc/modem.h . . . 406
G.87 inc/output.h . . . 407
G.88 inc/partition.h . . . 407
G.89 inc/partition_config.h . . . 412
G.90 inc/phone_book.h . . . 412
G.91 inc/scheduler.h . . . 413
G.92 inc/sd-reader_config.h . . . 413
G.93 inc/sd_raw.h . . . 414
G.94 inc/sd_raw_config.h . . . 417
G.95 inc/sleep.h . . . 419
G.96 inc/sms.h . . . 419
G.97 inc/spi.h . . . 419
G.98 inc/tests.h . . . 420
G.99 inc/thread.h . . . 421
G.100 inc/thread_switch.h . . . 421
G.101 inc/time.h . . . 421
G.102 inc/timeout.h . . . 422
G.103 inc/ui.h . . . 422
G.104 inc/usart.h . . . 423
G.105 inc/wait_queue.h . . . 423
G.106 gsmsandbox.c . . . 423
G.107 gsmsandbox.h . . . 429
G.108 gsmsandbox_tryout.c . . . 430
G.109 sdfiller.c . . . 437
G.110 test_vector_generator.cs . . . 438

Chapter 1

Introduction

The task at hand was to create two identical communication devices. They were to communicate over the mobile phone network using an encrypted connection. The two devices use a Liquid Crystal Display (LCD) and a numeric keyboard as human interface devices. A loudspeaker provides a ringing signal when one of the devices is dialed. And, of course, a headset is needed to be able to conduct a conversation. A Global System for Mobile communications (GSM) module attached to our PCB via pin connections, is used for the communication part. Figure 1.1 gives a quick overview.

The report begins in section 1.1 with an interpretation of the assignment we were given. This includes a detailed requirements specification. Then appropriate background information regarding the most fundamental technologies used is presented in an overview-style manner in chapter 2. This is followed by chapter 3 regarding the design phase, explaining high level design choices before submerging into details. Implementation details follows in chapter 4, this includes low level design choices. A test plan for the various parts of the design is presented in chapter 5. The results are then commented. Chapter 6 gives an overview of the tools used in this projects different phases. A discussion of things the group, in retrospect, wished were done is part of the discussion in chapter 7. This chapter also includes research and development thoughts.

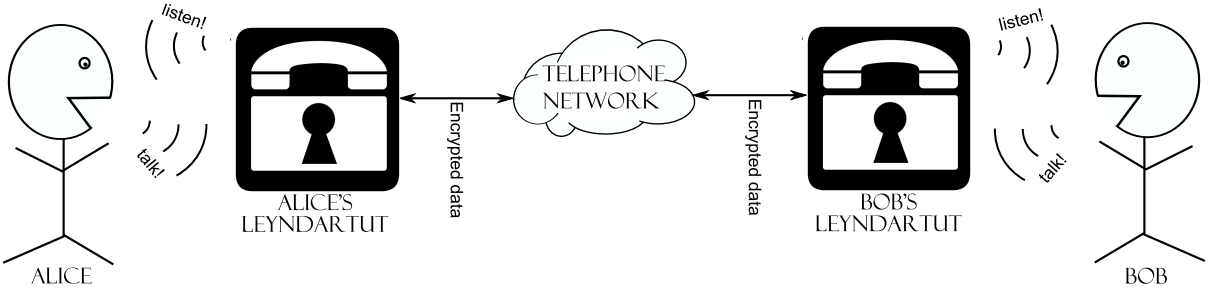


Figure 1.1: Diagram showing the role of the Dulkóðuð leyndartut

1.1 Assignment

Oppgave: Kryptert mobiltelefoni
Industrispionasje og identitetstyveri er et økende problem. Å sikre slike sensitive opplysninger blir ofte en avveining mellom hva som er praktisk og hva som er sikkert. Mobiltelefoner er allemannseie og svært praktisk for kjapp informasjonsutveksling. GSM er i dag beskyttet av en svak kryptering, men denne krypteringer er i praksis ikke til å stole på. Årets oppgave i TDT4295 er å konstruere et system for kryptering av tale over det regulære GSM / GPRS / 3G mobilnettet. Det forventes at det lages to enheter som kan kommunisere i begge retninger til demonstrasjon. Systemet skal være basert på åpne og kjente krypteringsstandarder, slik som AES. Systemet skal implementere symetriske kryptering for datastrømmen, men det er valgfritt om man vil også implementere privat / offentlig kryptering for utveksling av symetriske nøkler. Systemet skal ta utgangspunkt i at mobiltelefonen i seg selv ikke er til å stole på og må derfor ha sine egne A/D konverterere for sampling av lyd og D/A kretser for avspilling av lyd.
Andre krav:
Prosjektet skal minst benytte seg av en Atmel AVR mikrokontroller og minst en Xilinix FPGA/CPLD. Budsjettet for utviklingen av prototypen er på 23000 NOK, som skal dekke komponentutgifter og PCB-produksjon. Ut over dette må enheten konstrueres innenfor de rammer som til enhver tid settes av fagstaben. Tidsfristen for levering av ferdig PCB-utlegg er 2. Oktober. Presentasjon av ferdig prosjekt er 21. November.

Evaluering
Oppgaven vil evalueres av ekstern sensor. Til grunn for evalueringen ligger prosjektrapport og muntlig presentasjon av arbeidet med demonstrasjon av prototypen. Gruppen får samlet karakter med mindre det forekommer stor variasjon i individuell arbeidsinnsats.

Problem description
Given task, freely translated from Norwegian.

Industrial espionage and identity theft is an increasing problem. To secure such sensitive information is often a ponderation between practical and secure solutions. Cellular phones are "common property" and very practical for fast information exchange. GSM is today proteced by a weak encryption, but this encryption is in practical matters not trustworthy. The task of the year in TDT4295 is to construct a system for encryption of speech over the regular GSM/GPRS/3G mobile network. It is expected that two units which can communicate in both directions are built for demonstration.

The system shall be based on open and well known encryption standards, such as AES. The system shall implement symmetric encryption for the data stream, but it is optional if you also implement private/public encryption for exchange of symmetric keys.

The system shall be based on the assumption that the cellular phone itself is not to be trusted, thus it needs its own A/D converters for sampling of sound and D/A circuits for playback of sound.

1.2 Interpretation

The requirement specification given to us was fairly ambiguous and flexible. We were also encouraged to sculpt the specification ourselves. The following shows how we interpreted our given task.

- Two devices able to connect to each other via a mobile telecommunication network.
- Microphone input and speaker output. A headset is suitable.
- All audio communication should be encrypted using strong symmetric key encryption algorithms based on open and well-known standards.
- Design realized by using at least one AVR microcontroller and one FPGA or Complex Programmable Logic Device (CPLD).
- Conversion of speech between analog and digital must be done by A/D and D/A converters in the system.
- Available budget: 23000 NOK
- Initial deadline for PCB design: 02-10-2007
- Final deadline for project report: 21-11-2007
- Final deadline for project presentation: 23-11-2007

The Dulkóðuð leyndartut uses Triple DES (3DES) encryption. This will ensure a secure connection between the two parties, given that our implementation is correct, as well not containing any critical weaknesses. We first set out to use the 128 bit Advanced Encryption Standard (AES). This was changed due to the lack of space on the FPGA. AES would have improved the security level. We will come back to this matter, discussing the encryption in section 3.3.2.

1.3 Requirements Specification

Two prototype units are to be produced. They should conform to the following requirements:

1.3.1 Functional

Misc

- The units must communicate using a standard mobile network (GSM or UMTS).
- The units must support full duplex speech communication.
- The units should have audio input/output ports for receiver (3.5mm jack or RJ-11).
- The units should be able to use either a power adapter or batteries, with a minimum battery life time of 3 hours in standby mode or 15 min in use (optional).

User Interface

- The units should have a user interface similar to that of a cellular phone, with features such as dial, hang-up and stored numbers (with corresponding encryption keys).
- It must be possible to hang up (end a connection attempt/close a connection).
- The units should have a numeric keypad.
- It should be possible to adjust the output volume.
- The units should have a bell or beeper to indicate incoming calls.
- The units must have a bistable power switch.
- The units should have a LED for indicating status of power (and optionally a LED indicating charging of the batteries).
- It should be possible to store and load at least 10 telephone numbers with associated encryption keys.
- The units should have a backlit character display for showing status (incoming call, caller's number, recipient unavailable, recipient busy, power level and signal strength).

Security

- The audio data must be encrypted using strong symmetrical encryption (AES or Serpent in CBC or cipher feedback mode).
- Encryption will be done in the FPGA/CPLD.

1.3.2 Non-Functional

- It should be easy to solder the components onto the PCB. DIP sockets should be used where possible.
- The sound quality should be good enough for speech.
- The total cost for the components used in the production of the two units should not exceed 23 000 NOK.
- The units shall use at least one Atmel AVR microcontroller.
- The units shall use at least one Xilinx FPGA/CPLD.
- A/D and D/A conversion in both units.
- The units should be able to gracefully handle continuous data loss for a few seconds.

Testability Requirements

- Important signals on the PCB will be connected to testpods, for easier debugging and logic analysis.
- Vital (and not too rapidly changing) signals will have status LEDs attached to them on the PCB.
- VCC and ground layers will have pins attached at several different locations on the PCB, to make multimeter measurements easier.
- It should be possible to test each main component independently.

1.4 Prototype vs Production Model

The requirement specification above is intended for the prototype units built in this project. For a production model, there would be several changes to the requirements.

Size, power usage and durability are not considered important at this stage, and requirements for these are weak if at all present. For a production model, however, they would be essential.

The testability requirements are relevant for the prototype only and would be removed in a production requirement specification.

Chapter 2

Background

The background chapter will try to build up a supporting background for the following chapters and general information about cryptography, microcontrollers, FPGAs, audio and GSM will be given.

2.1 Historical aspect and the situation today; availability of solutions

The need to communicate securely has a long history, as described in [11]. It dates back 2000 years, when the Romans ciphered their messages sent by messengers in order to protect the vital information if their messenger was captured. At these times the cryptographic methods were of course not as advanced as today. The methods they used are called the substitution system. Each letter is substituted, by rule, with another letter. This new letter can either come from the same alphabet, or from another alphabet. This last method was in use by the Romans, the latin alphabet was swapped by the greek alphabet. Each letter, in this case, is a message since each letter is encrypted alone. The key is the chosen substitution function (which shows which letter corresponds to which new letter, in any given alphabet). Modern cryptography methods depends on computers. The methods are far more advanced, and any digital content can be encrypted.

The following are newer examples of the use of encryption. A discotheque in Berlin, Germany, was the target of a terrorist attack in 1986. Two American soldiers died. No organization would admit responsibility. The terrorists thought they did not leave any traces, confirmed by a phone conversation saying the words: "Mission accomplished. We did not leave any traces.". They did. Few minutes after the call was made, computers belonging to the American National Security Agency (NSA) had received the conversation data. It was a call going from the Libyan embassy in Berlin, to the embassy of the same country in Rome. It was encrypted, but it did not take the NSA long to decrypt the suspicious information transfered only moments after the explosion. Days later president Reagan ordered the bombing of selected targets in Tripoli. In the same manner decrypted messages from Teheran and to the Iranian embassy in Paris revealed the assassins of the former Iranian foreign minister Chapour Bakhtiar. These examples from the real world shows the power of being able to attain information one were not meant to possess in the first place. It was a clarifying event to get this information. But there also exists groups that are after information they should not be obliged to attain. It shows that governments want to communicate secure, but at the same time be able of decrypting other governments

communication. Put in another perspective, this is also true for the commercial world of firms. They want to stay at the top of the information, adding benefit to their own firms.

The ability to make secure calls is of great importance these days, due to the increased availability of GSM interception equipment[11]. This kind of equipment has become much more affordable the last years, making it more widespread and enabling surveillance of even more firms of smaller sizes where contracts are of less value, but still can be of some importance for competitors. Secure communication has thus become of a greater need than before.

> The use of wiretapping has become so widespread, simple and uncontrolled that you must assume that records of your private calls may end up in the wrong hands. Equipment for wireless interception of mobile phone calls has become available at such low prices that it is deployed frequently even in comparatively small business conflicts. So using encryption to protect your privacy is the prudent choice.
>
> — *Mobile firm GMSK regarding their CryptoPhone series*

There exists commercial products. The german firm GMSK for example sell several models in their CryptoPhone series. They use, as proposed in our assignment, AES – and also Twofish[1]. Calls on these phones are encrypted with 256-bit keys using AES and Twofish running as counter mode stream ciphers. This is described in [5]. They release the algorithms as open source, thereby enabling anyone to confirm the level of security. Other products tend to have an undocumented "black-box" that encrypts the stream, unable or maybe unwilling to document the security of their device. The solutions from CryptoPhone seems to be among the most secure, available commercially to the common man. American firms were until recently[2] not able to deliver products in this market with the same level of security, due to American export restrictions. Products implementing high level of encryption were denied by law to be exported from the US, placing cryptographic software in the classified military hardware domain. On the noncommercial side there are many different solutions. As an example, one of the most successful and widespread were initiated by the NSA in the United States of America, described in [24]. They initiated the Future Secure Voice System (FSVS) in 1984. The aim was to make high level communication more secure by the end of the 80s. Secure Telephone Unit, Third generation (STU-III) was developed and produced by 1987. It was designed to be the size of a conventional telephone desk set. It was also relatively user-friendly and low cost. A 15 seconds wait was needed going into "secure mode" when two STU-IIIs were communicating. There were produced hundreds of thousands of these units, contracted to General Electric, Motorola and AT&T. Even mobile editions were produced. These devices secured the calls of the U.S. Government at all levels; the President and his Cabinet, Congress, the military, civil agencies, law enforcement, government contractors and research institutions. Today, newer generations of encrypted phone systems has replaced the STU-III.

2.2 Encryption

Since cell phone signals are transmitted wirelessly, it is possible to wiretap the communication. In order to ensure that no one except the intended recipient can obtain the data that is being

[1] Twofish was one of the AES finalists (see section 2.2.2), but it did not win

[2] A relaxation regarding cryptographic software export has found place since 2000

transferred, the data must be encrypted – that is, transformed in such a way that it becomes unintelligible to anyone who is not in possession of a secret *key*. The key is, of course, assumed to be known only by the intended recipient (and possibly also by the sender, depending on how the encryption works).

Different *ciphers* (encryption algorithms) can be classified as having have different characteristics There are several categories of ciphers. The most important ones are:

- Symmetric block ciphers
- Symmetric stream ciphers
- Asymmetric ciphers (public-key cryptography)

Block ciphers operate on fixed-size data chunks, while stream ciphers can process arbitrarily long data sequences (and do not require all of the data to be present before encryption can start). Simple modifications enable block ciphers to process longer sequences as well. Symmetric ciphers use the same key for encryption and decryption, which requires that both the sender and the recipient are in possession of the same key. With asymmetric ciphers, each participant has a *public key* that may be known by anybody (and which the sender must use to encrypt the message), and a *private key* that can decrypt messages that have been encrypted with the corresponding public key.

The two algorithms that we have worked the most with, AES and 3DES, are presented in the following subsections.

2.2.1 Triple DES

Triple DES is based on the Data Encryption Standard (DES) (Data Encryption Standard) cipher. Like DES and AES, it is a symmetric block cipher. Its operation consists of sixteen *rounds* in which the same set of transformations is applied to the output from the previous round. In each round, the input is divided into two. One half is sent through a so-called *Feistel function*, which expands the input data by duplicating some of the bits, then XORs it with the *subkey* for the current round (which has been generated based on the original key). The result is transformed by substituting each byte with another byte according to a lookup table called an *S-box*, and then the bytes are permuted. The result is now XOR'ed with the other half of the input block and is sent to the next round. The Feistel function is illustrated in Figure 2.2, and the overall structure is depicted in Figure 2.1. For a more thorough explanation, please consult the DES specification[15].

Whereas the structure of algorithm itself has resisted effective cryptographic attacks, its vulnerability has turned out to be the key size of 56 bits[23]. Parallel computers designed specifically for the purpose of breaking DES (such as the Electronic Frontier Foundation DES Cracker) are capable of deciphering an encrypted message in a matter of days. In order to circumvent this problem without having to devise an entirely new algorithm, Triple DES was introduced. There are several versions of the algorithm, and we use the one that is known as 3TDES or 3TDEA, in EDE mode. Its operation is as follows: There are three distinct keys, k_1, k_2, and k_3, each containing 64 bits. However, eight of the bits in each key are parity bits, so the effective key size is $3 \times 56 = 168$. A data block consists of 64 bits. The data block is first encrypted with k_1, and the result is "decrypted" with k_2 (which will only further scramble the data rather than actually decrypting, since $k_1 \neq k_2$). Finally, the new result is decrypted with k_3. This is depicted in Figure 2.3.

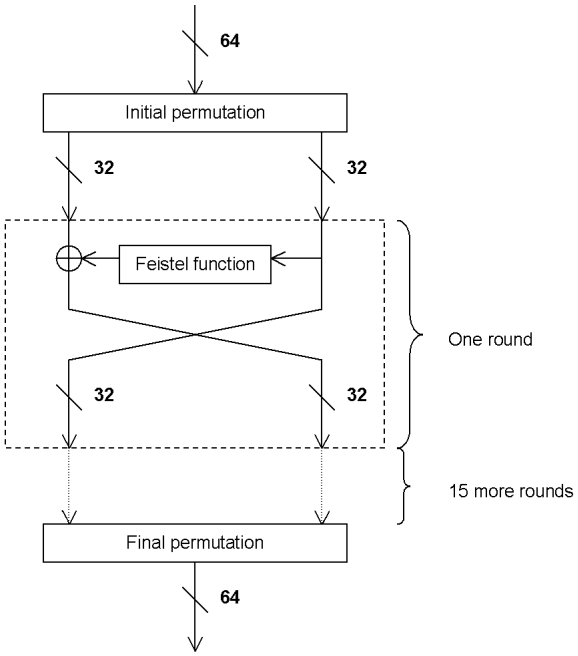


Figure 2.1: Top level structure of DES

2.2.2 AES (Rijndael)

With DES having been cracked and Triple DES not being particularily fast, the National Institute of Standards and Technology started a process for selecting a new algorithm to supersede DES[22]. Several renowned cryptographers contributed with suggested algorithms. Eventually, the Rijndael algorithm was chosen to become the AES - Advanced Encryption Standard.

AES can operate on key sizes of 128, 192 or 256 bits, but it always uses a block size of 128 bits. It consists of a number of rounds that is given by the key size; for 128 bit keys, the number is ten. The algorithm starts off by generating a number of subkeys, and then fills out a *state array* with the plaintext. Each round consists of four steps that manipulate the state array: byte substitution through S-boxes, circular shifting of the array rows, combination of the elements in each column to form new column values, and XOR'ing the state array with a subkey. Again, please consult the AES specification[16] for further elaboration.

2.2.3 Modes of operation

Using block ciphers to transmit portions of data larger than one block is risky, since data blocks that are equal will always produce the same ciphertext block. In order to prevent this, some *mode of operation* must be employed. We have chosen to use *cipher block chaining* (CBC), which appeared to be the simplest one to implement. It makes each ciphertext block dependent on all the plaintext blocks from the beginning of the plaintext sequence, by combining the previous ciphertext block with the current plaintext block before encryption. See Figures 2.4 and 2.5 for illustrations of the encryption process and the decryption process, respectively.

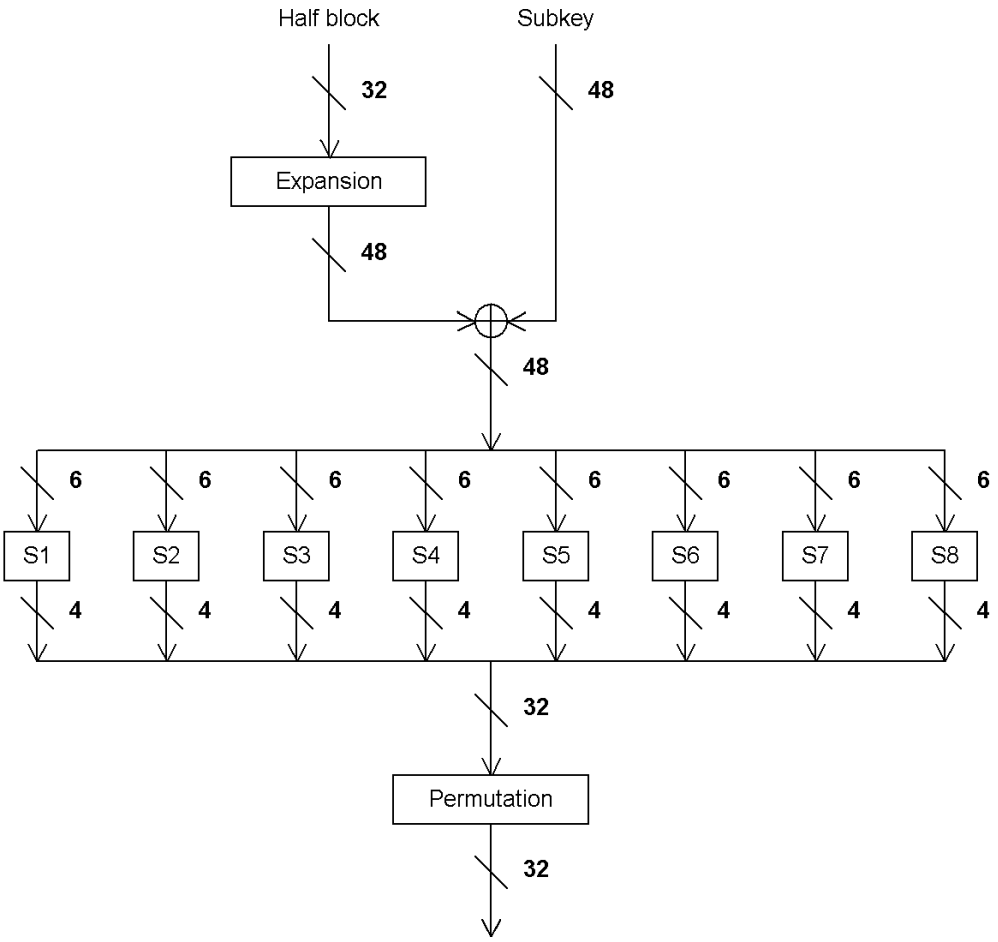


Figure 2.2: Structure of the DES Feistel function

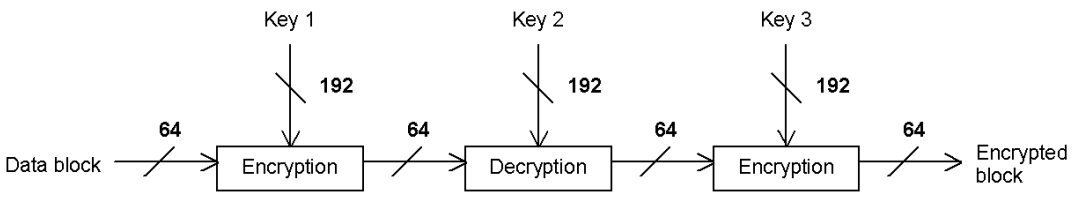


Figure 2.3: Structure of Triple DES

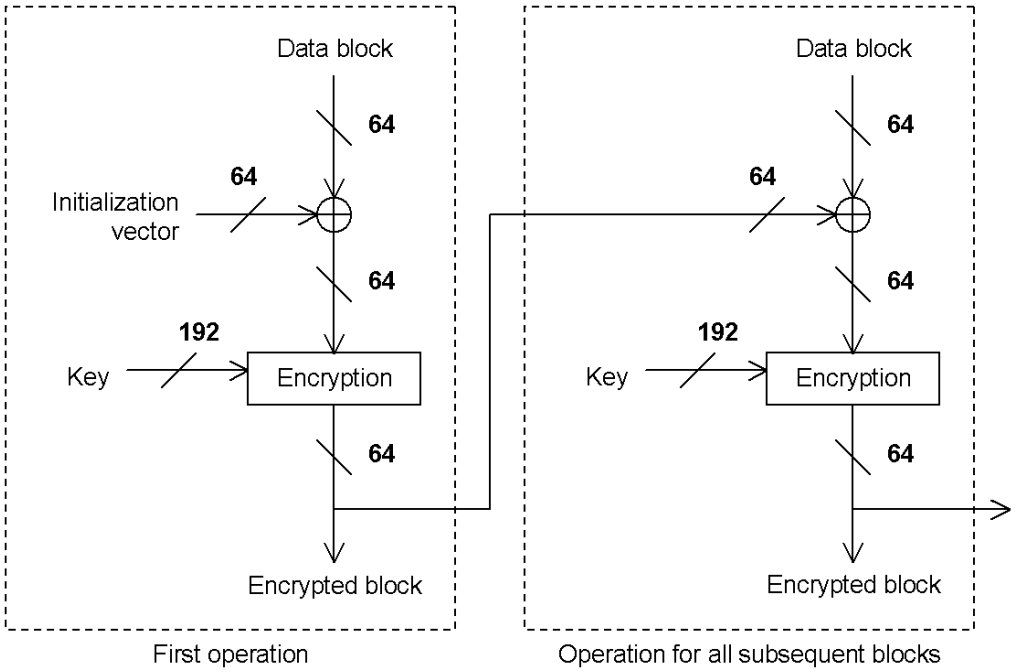


Figure 2.4: Encryption in cipher block chaining mode

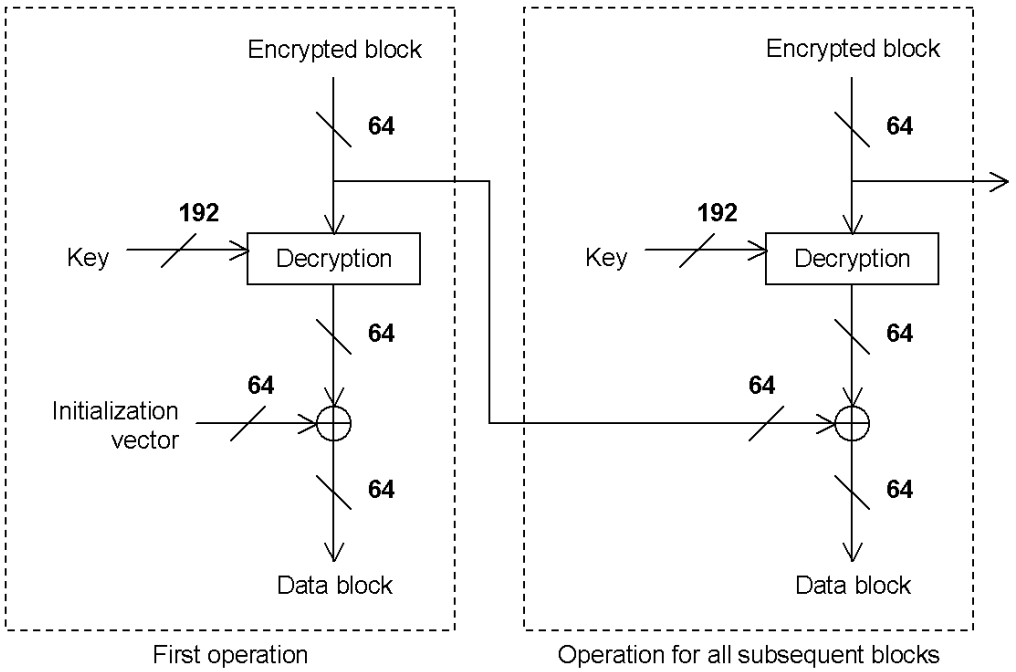


Figure 2.5: Decryption in cipher block chaining mode

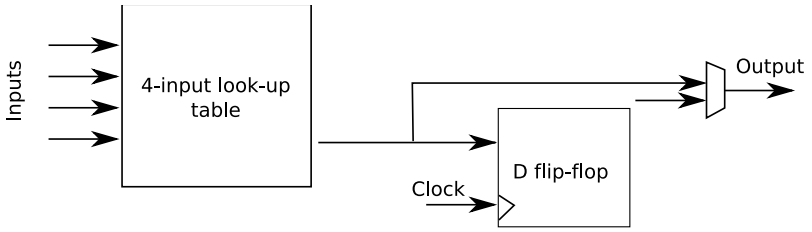


Figure 2.6: A FPGAs logic block

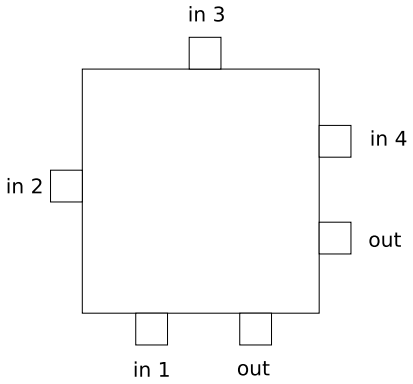


Figure 2.7: The pins of the logic block

2.3 FPGA

FPGAs are semiconductor devices deriving from CPLDs. CPLDs saw the light of day in the start of the 80s. They are both programmable semiconductor devices containing programmable logic elements. While the CPLD contains from some thousands to tens of thousands of logic elements, the FPGA can contain from tens of thousands to several millions. The main difference between the two is architectural differences. The FPGA is far more flexible, regarding the range of different designs possible to implement. At the same time this makes the FPGA more complex to develop for (more choices; more complexity). FPGAs can have embedded functions such as adders and multipliers, and memory. The most costly models from Xilinx.[3]

As shown in figure 2.6 a common FPGA logic block is made up of a 4-input Lookup Table (LUT) and a flip-flop. Figure 2.7 shows the pins of such a logical block. Each output pin of the logic block is able to connect to any of the wiring segments in the channels adjacent to it. Each input is accessible from one side of the logic block. The length of each wiring segment is the same as one side of the logic block. At each end it terminates into a switch box. These switches are programmable, so longer paths can be constructed by the flip of a switch. There exists FPGAs that uses longer routing lines, spanning multiple logic blocks, making higher speed interconnection possible. Newer high end FPGA designs uses 6 input pins for the logic blocks, delivering higher performance. Figure 2.8 shows the logic blocks and the wiring between them. The programmable switches between wires are placed where the wires cross, at the corners of the logic blocks.

[3]The market leader together with Altera regarding FPGAs even contain a PowerPC (PPC) Central Processing Unit (CPU) core that can work as part of the design.

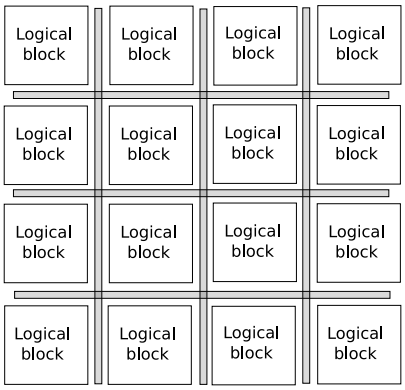


Figure 2.8: Several logic blocks and the communication channels

2.4 VHDL

VHDL (V stands for Very-High-Speed Integrated Circuits (VHSIC)) is commonly used to describe hardware in FPGA and in other intergrated circuits. The history of VHSIC Hardware Description Language (VHDL) [25] goes far back to the eighties at the US Department of Defence. In order to document the behavior of the integrated circuits suppliers were including in equipment, they made a language for documenting the circuits. As a result of this they were now able to simulate the behavior of integrated circuits based on the documentation. The choose of Ada as the base of the language made it possible to avoid re-inventing concepts that had already been thoroughly tested in the development of Ada. As with Ada, VHDL is a strong typed language and is case insensitive.

The language needs a simulator to run its code. It can read and write to files on the host computer, which actually enables VHDL to produce code to be incorporated into the design being developed. It is possible to write *testbenches* to verify the functionality of the design using files on the computer, get input from the user and compare the result from the production with the expected results.

VHDL is not a programming language but it is a description language for hardware. This means it is not meant to be executed in any way but rather describe the gates and wires. To be able to transfer the language down to this level it has to be synthesized.

2.5 ATmega microcontroller

The ATmega AVR microcontroller can perform approximately 1 instruction per clock cycle, giving a 8 MHz ATmega128[4] a performance of about 8 Million Instructions Per Second (MIPS). The family is well supported by the GNU Compiler Collection (GCC), contributing to its popularity. The AVRs in this family have onboard flash memory and Random Access Memory (RAM), and many popular interfaces. For example support for serial communication, and A/D-conversion. These are systems on chip, so to speak. These microcontrollers are based on the Harvard architecture. This means they got separate program and data memory.

[4] As used in our design

2.5.1 External memory interface

The ATmega128 microcontroller has 4096 bytes of internal Static Random Access Memory (SRAM).

This memory can be augmented by attaching an external memory chip to the external memory interface on the AVR. This can increase the amount of memory accessible from the microcontroller from 4096 bytes to 65280 bytes. The external memory interface consists of between 11 an 19 signal lines. All 19 lines must be used to address the full range of external memory.

2.6 Audio

To deal with audio from analog sources in an digital circuit we will first have to convert it to a digital bitstream which describes the audio accurate enough to get the sound quality we need. To play back digital audio we have to do the inverse conversion to get the analog audio back. To prepare the analog signal to conversion some signal processing has to be done.

2.6.1 Analog/digital audio conversion

Pulse-Code Modulation (PCM) [26] is a method for sampling audio where the sound wave is represented by a binary code which represents its magnitude at the time of sampling as shown in figure 2.9. This sampling happens at uniformly divided intervals defined by the sampling rate. The Shannon-Nyquist sampling theorem states sampling of signal can only capture frequencies less than or equal to half the sampling rate. Frequencies above this limit is dissorted through aliasing effects. Speech can be captured at a rate of 8kHz without substantial loss of quality if the frequencies above 4 kHz is removed by a filter.

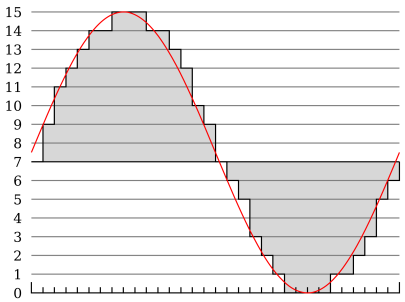


Figure 2.9: PCM audio conversion

2.6.2 Analog audio

To prepare the audio signal for Analog-to-Digital (AD)-conversion a few measures has to be taken. The audio has usally to be amplified and frequencies above half the samplingrate filtred. The amplifing happens in a operational amplifier (op-amp) and the filtering is done using capacitators and resistors. See [10] for an example circuit.

2.6.3 Compression

To be able to transmit audio over a narrow bandwidth link which usally is the case in a cell phone, it is nescessary to compress it to a datarate the datalink has capability to transmit. Audio sampled at a sample rate of 8 kHz with a sample resolution of 16 bits and no encryption, will give us 8000 samples each second each at 16 bits. To transmit this audio stream we will need a link with a bandwidth of 128 kbps. As you can see in the next section, a typical mobile network does not have that kind of bandwidth available.

Several compression algorithms designed for compressing speech exist. Speex [27] is a open source compression algorithm.

Digital voice systems inc. has developed an algorithm based on the Multi-Band Excitation (MBE) model, which can give very good speech quality at rates between 2 and 5 kbps. Se [9] for more details about this.

Other speech compression include Adaptive Differential Pulse Coded Modulation (ADPCM), Low-Delay Code Excited Linear Prediction (LD-CELP), Conjugate-Structured Algebraic Code Excited Linear Prediction (CS-ACELP), Code Excited Linear Prediction (CELP), Linear Predictive Coding (LPC10) scheme[2].

2.7 GSM/GPRS

In the next section, we will introduce some common standards for mobile networks that we evaluated. First we will introduce GSM, followed by General Packet Radio Service (GPRS) and then Enhanced Data rates for GSM Evolution (EDGE).

2.7.1 GSM

The most widespread mobile network in Norway today is called GSM. This is a Time Division Multiple Access (TDMA)-based mobile network. Unlike Institute of Electrical and Electronics Engineers (IEEE) 802.11a/b/g/n (referred to as Wireless Local Area Network (WLAN)), GSM supports roaming, or handover. This means that a mobile terminal (usually a mobile phone) can move to another cell in a call or data transfer without losing the connection. A cell corresponds to the signal coverage of a Base Transceiver Station (BTS). This is not possible with standard WLAN, since WLAN doesn't support the roaming capability, and can therefore not be called a mobile system.

However, roaming capability within WLAN is being implemented in the form of IEEE 802.11f, but is still on testing phase. The GSM mobile network is digital and works the same way as Integrated Services Digital Network (ISDN). GSM supports speech and data transfer with a bandwidth of 14.4 kbps, which is a fairly low bandwidth today. The bandwidth corresponds to a single TDMA channel.

When a mobile terminal sets up a call or data transfer to another mobile terminal using GSM-speech or GSM-data, the terminal connects to a BTS. Several BTSes are connected to a Base Station Controller (BSC), and further, an Mobile-services Switching Centre (MSC). An MSC controls several BSCs with their underlying BTSes. The MSC are connected to a gateway exchange, in other words, to the rest of the telephone network. This is the path a conversation or GSM-data transfer must travel to reach the recipient when setting up a call or data transfer. The path is shown in figure 2.10.

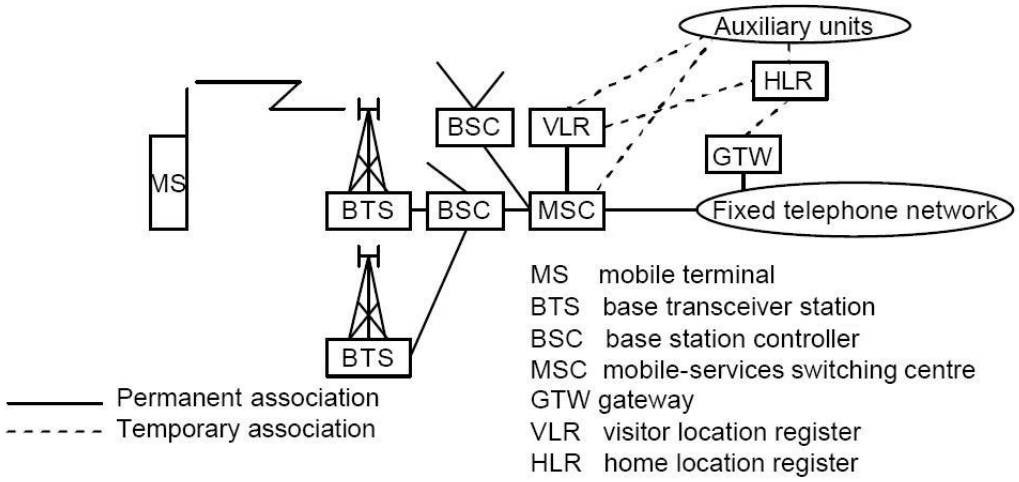


Figure 2.10: GSM architecture

2.7.1.1 How secure is GSM?

With GSM only the air signals are encrypted. "Man in the middle" attacks are possible as the Mobile Stations (MSs) authenticates towards the base stations (BTS), but the reverse is not needed. This enables the possibility of setting up a false base station. This false station can send false data in both directions, or simply just listen to the data passing. In addition, the ciphering key used for encryption of the aired signals is only 54 bits long. End-to-end security is not provided. Also, Subscriber Identity Module (SIM) cloning is evidently possible.

2.7.2 GPRS

As mentioned earlier, a pure GSM connection (speech or data transfer) has a very low bandwidth. This is because GSM only uses a single TDMA channel. This is where the GPRS technology comes in. The GPRS technology allows a mobile terminal to gain much higher bandwidth over the GSM mobile network than pure GSM does. In contrast to GSM-speech and GSM-data, GPRS uses several TDMA channels, which increases the bandwidth up to 115.2 kbps when using eight TDMA channels (GPRS class 8).

Better equipment has allowed us to use even more than eight TDMA channels for GPRS connections. The source we used to document GSM/GPRS was fetched from the course material in the course TTM4105 Access and transport networks [6].

A GPRS connection doesn't follow the same connection path as the GSM-speech and GSM-data does. In the BSC which is expanded with a Packet Control Unit (PCU), the GPRS connection is sent to a Serving GPRS Support Node (SGPN) and then to the internet, not to an MSC, which would send the data to the telephone network. The PCU provides support for packet switching in the mobile network, which is circuit switched. This means that we cannot connect directly to a mobile terminal just using the mobile phone number. We need to know the Internet Protocol (IP) address of the counterpart if we want to set up a direct GPRS-connection between two mobile terminals. Figure 2.11 shows the connection path for GPRS connections.

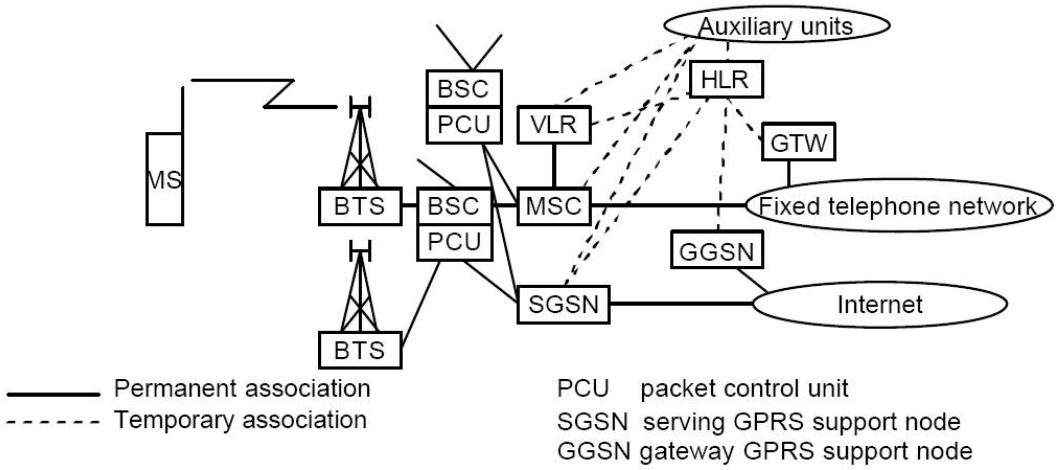


Figure 2.11: GSM/GPRS architecture

2.7.2.1 EDGE (E-GPRS)

By using more complex coding and by altering the modulation scheme of GSM, even higher bandwidth in each time-slot can be achieved than by the scheme of regular GPRS. This technique is called EDGE, or Enhanced General Packet Radio Service (E-GPRS). The bandwidth is not far below that of a Universal Mobile Telecommunications System (UMTS) network (not High-Speed Downlink Packet Access (HSDPA)). Today, both GPRS and EDGE/E-GPRS are available in most of the GSM mobile network.

2.7.3 Administration

Since GSM is a telephone system, we can use known administration techniques to set up a call or a data connection. Modems are administrated with a set of commands, called AT commands, the Hayes AT command set. These commands are also used to administrate a mobile terminal, since mobile terminals also are modems. The AT commands can vary from manufacturer to manufacturer, and when in need of administering such a device, one needs to obtain the documentation for that specific device to be sure which commands will work. This is one of the most important factors we have to consider when we begin working with our mobile device.

Chapter 3

Design

The first section of this chapter presents the processes to be performed by the unit and the next section splits them into functional units focusing on the data flow. Section 3.3 presents all the main parts of the system and the specific choice of components.

The next section contains information about the design of the power supply. This section is followed by a section about the various peripherals. PCB design goals comes in section 3.6. Software design comes next, in section 3.7. Last we talk about the casing for the device.

3.1 Process overview

This section describes the different processes needed for transferring encrypted voice over a telecommunications network. Figure 3.1 outlines the independent processes required. Each process is discussed in separate subsections below.

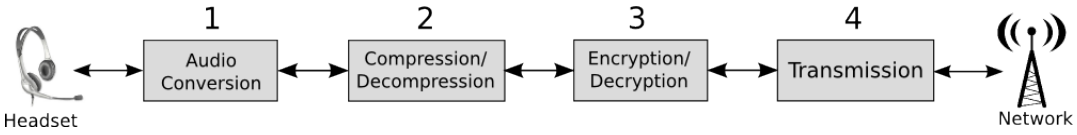


Figure 3.1: Processes involved

Audio (1)
The audio circuit does the analog preprocessing and converts between analog and digital signals. The preprocessing is mainly to remove frequencies which are above the maximum frequency the AD/Digital-to-Analog (DA) converter can handle.

Compression (2)
The compression module is responsible for compressing the digital audio to a rate which can be transmitted over the narrow bandwidth we have available, and to decompress the audio received over our transmission link. In addition to compression some filters are applied to eliminate information in the sound that is not important to the human ear.

Encryption (3)
The data that is going to be transmitted (or has been received) is handled in blocks of the right size for the encryption/decryption. From a black-box perspective, encryption

is a simple process where data is simply fed into the box and encrypted data comes out. Decryption works the same way except that encrypted data is fed in, and decrypted data comes out.

Transmission (4)
The transmission unit is connected to the microcontroller with a serial interface. The microcontroller receives data from the encryption unit and pushes it to the transmission module. We receive data the same way.

User interface
In addition to the functional processes outlined above, the design includes peripheral components for input/output to control the Dulkóðuð leyndartut. We will use a display and a keyboard to provide the interactive user interface, and a removable storage unit is used for inserting encryption keys. In need of a way to handle delicate and rather large encryption keys, we decided to integrate a flash memory card reader onto our PCB. In this way, generating new keys can be done easily using a PC with a memory card reader integrated or connected. Since flash memory cards are popular in many applications from photography and media players to cell-phones and more, most memory cards are easy to obtain and are typically in the size range from 512MB to 4GB.

3.2 Main functional units

The main data flow through the system occurs during a conversation. It is full duplex, so the flow is perceived by humans to go both ways simultaneously. This is shown in figure 3.2. We will go into more details about the components later in the chapter.

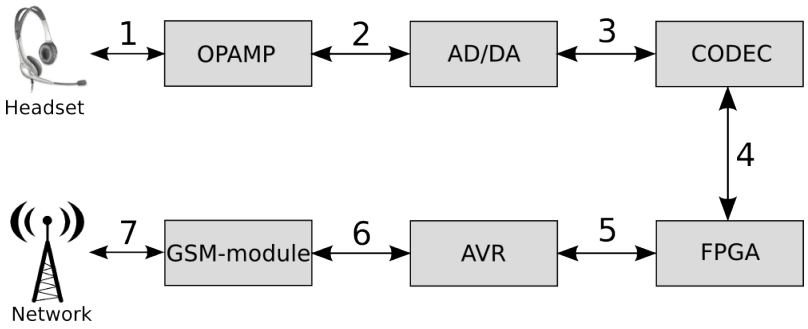


Figure 3.2: The data flow through the system when audio is sent and received

3.3 Choice of main components

During the planning phase a lot of choices had to be made, such as which chips and modules were going to be used. This section contains descriptions of the various alternatives found for each component and arguments behind the final decisions. Because of the great variety in features, cost and availability of GSM modules and compression Integrated Circuit (IC)s, a lot of research and consideration had to be done. Figure 3.3 shows all the main components chosen and which components that are connected to each other. This section will show what opportunities we found and what choices we took to get to the components shown in the figure.

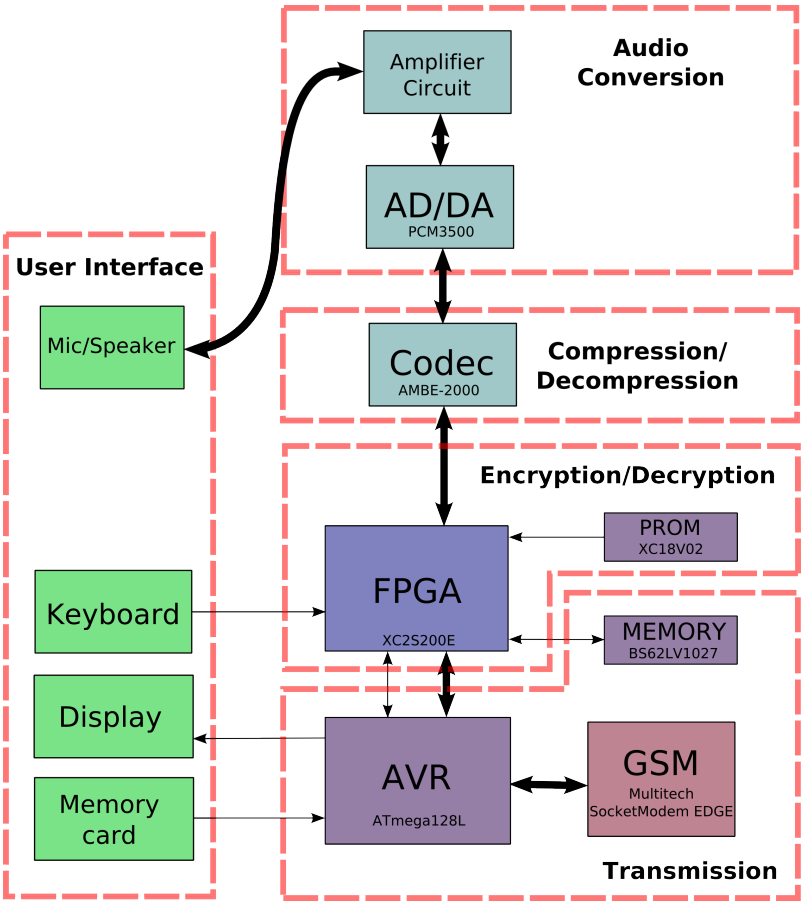


Figure 3.3: Block diagram showing the main components and interconnections

3.3.1 Audio

To deliver a product that fulfills the requirement specification we had to find a way to convert between analog and digital audio, and compress/decompress the audio stream. We saw that to implement the compression in AVR would push a bigger load to both the CPU and the project team than we could allow.

So a couple of approaches could be used to do this, and we first tried to find a chip that both compressed and did the AD/DA conversion, but did not find anything suitable in our price range. There are a few ways to do this process that might could be applicable to this project. We could use a AD/DA converter combined with a Digital Signal Processor (DSP), AD/DA with a compression codec chip or AD/DA and an FPGA.

3.3.1.1 Audio compression

Several approaches can be used to compress and we tried to evaluate our most compelling possibilities.

Complete solution, DVSINC VC-55 A complete board that does both the AD/DA-conversion and compression is the DVSINC VC-55[4] from Digital Voice Systems Inc.

- **Pros**
  - A tested complete solution
  - Good compression (3.6 or 7.2 kilobit per second (kbps))
  - An easy solution that makes our job a lot easier.
  - Nice speech quality with speech enhancements like echo cancellation etc.
- **Cons**
  - Price (US$925.00) is too high, since two devices would eat over half our budget.
  - Gives a more clumsy design, by adding a external component.

AMBE-2000 Vocoder Chip The AMBE-2000 was the only available chip with a data rate which suits our needs we found. The chip uses a compression algorithm with an extreamly low data rate.

Figure 3.4: Picture of the AMBE-2000 voice codec chip

- **Pros**
  - Extremely low compression rate (down to 2.0 kbps)
  - Nice speech quality with speech enhancements like echo cancellation etc.
  - Gives us better control of our design.
  - Feature rich
- **Cons**
  - Needs a seperate AD/DA converter (compared to VC-55)
  - We have to implement another bus (compared to using FPGA)
  - Not as easy as VC-55

We considered a Hardware Development Kit (HDK) for this chip but the price (US$975.00) would make it impossible for us to keep withing budget.

DSP It is possible to implement compression in a DSP chip mounted on our PCB.

- **Pros**
  - There are DSP-code for many compression algorithms out there (speex, MELP/CELP)
  - Makes it possible for us to test different compression algorithms
  - A DSP can do a lot of things and might come in handy
- **Cons**
  - DVSInc claims that AMBE-2000 outperforms MELP/CELP
  - None of the group members has any prior knowledge of DSP programming
  - Much of the existing code for DSP's has strict licenses.
  - A lot of hard work to program the DSP.

FPGA The compression can be implemented on an FPGA, and since we have to include a FPGA or CPLD this might be a possible implementation.

- **Pros**
  - Reduces the number of components as we can do both compression/decompression and encryption/decryption on the FPGA
  - Reduces the need for datashuffling around on the board.
- **Cons**
  - A lot of hard work to make a FPGA firmware.
  - Increaeses the risk (if we don't get the FPGA up and running, it reduces our posibilty to work around it in some clever way.)

Conclusion Even though to use a DSP or FPGA would have given us more streamlined product, we have to put that up against how much it would cost us in hours of work. The AMBE chip should (in theory) work without doing much more than putting it in the right circuit and wait for it to compress/decompress audio. Since the AMBE datasheet provides us with a pretty complete schematic for a circuit we see this as a even stronger option, since it eases the schematic design of the audio part and therefore we choose this component.

3.3.1.2 AD/DA

We need a chip to that modulates the analog sound into digtal signals (PCM) at a rate which matches the chip we chosen (16 bits at 8 kHz).

The AMBE-2000 datasheet [10]:

- AD73311 (Analog Devices)
- TLV320AIC10 (Texas Instruments)
- PCM3500 (Texas Instruments)

The AMBE user manual mentions the PCM3500 chip as a good option: "The Texas Instruments PCM3500 codec chip presents a simple low cost solution for use with DVSInc's AMBE-2000™or AMBE-2020™vocoder chips. This application note provides information on alternative methods of interfacing these components." ([10], p. 57)

3.3.2 Encryption

To be able to handle the cryptographical work and high speed data transfer we were adviced from the problem description to use either an FPGA or a CPLD.

An advantage by choosing a FPGA over a CPLD is its size. We needed a device able to implement a well known encryption algorithm. To define such an algorithm in VHDL we needed enough configurable units in the FPGA or CPLD. We first looked around to see if we could find some other projects to use as a base. As we found a project that implemented the AES in VHDL.

Based on the FPGAs and the CPLD available at the computer laboratory the FPGAs had much more pin outs than the compared to the CPLDs. Because of the need of a great number of pin outs to interface with a keyboard, a microcontroller, a led array, a button array and a external memory we had a great benefit by chosing a FPGA. The FPGA we ended up with was a Xilinx XC2S200E.

With the choosen FPGA we also need a Programmable Read-Only Memory (PROM) for coding the FPGA when booting to avoid reflashing the FPGA with JTAG every time we reset the circuit. The PROM had to be large enough to hold the whole circuit definition of the FPGA. We used what was already on the computer laboratory, Xilinx XC18V02. This has an inbuilt interface to communicate with the family of FPGA we are using. There are other possible ways to program the FPGA with its firmware, like connecting it to another device with builtin flash memory. We decided to use this solution because this kind of PROM already have a well configured protocol to upload the definition file to the FPGA.

3.3.2.1 Choice of encryption algorithm

The only requirement in the task specification was that a well-known and open "strong encryption" algorithm should be used. We were also told by our advisors that this had to me implemented in the FPGA using VHDL. We still had many possible options, and those who had some knowledge about encryption suggested that we should use a (symmetric) block cipher in *cipher block chaining* or cipher *feedback* mode and therefore this was added to the requirements specification. The rationale is that stream ciphers cannot use the same key twice which would require us to deal with the difficult problem of key distribution, and that asymmetric ciphers are considerably slower than symmetric ciphers (also, the task specification dictates the use of a symmetric cipher). We considered using an asymmetric cipher for exchanging the symmetric keys (as mentioned by the task specification), which is a common method, but concluded that we most likely would not get the time to implement this. When using block ciphers to encrypt a data stream, one should avoid the *electronic codebook* mode (which consists of dividing the stream into blocks and process them independently of each other), which can reveal patterns in the data.

There is a large number of block ciphers to choose between, and with limited knowledge about encryption, we figured it would be safest to go with one of the most well-known ciphers, such as AES/Rijndael, blowfish, or twofish. AES was the algorithm that we most frequently

found freely available VHDL implementations of when we searched on the internet. We also found several implementations of other algorithms, and the issues that have been taken into account when selecting which algorithm and which implementation to use include:

- Readability and understandability of the code
- The language must be VHDL (we found Verilog implementations, but when we asked our advisors if we could use them, they said we were not allowed to use Verilog due to the lack of personell skilled in VHDL)
- The extent to which the code seems modifiable
- Whether or not the code included test bench (preferably for FIPS standard tests)
- Which mode(s) of operation had been implemented (Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CF))

We initially decided to go with a block size and key size of 128 bits, since this was the only size supported by nearly all the implementations we found. Since the requirements do not mention any specific level of security, we trust that any key size from 128 bits and up will be sufficient for our purposes. We designed our solution in such a way that if a different block or key size would be needed at some point, making such a change will be fairly straightforward (ideally just changing a few constants and modifying the VHDL for the buffer layer between the crypto core and the external interface of the FPGA). It turned out that this was a wise decision (see the next paragraph).

We have not found any particularly positive or negative aspects with any of the algorithms we studied (except for 3DES, which is more or less superseded by AES[22]). Since we found most AES implementations, we initially decided to go with one of those. We did most of the initial work on the "AES128" code (see below), since it was quite neatly structured and readable/understandable. However, in order to keep all possibilities open, we decided not to bind ourselves to that particular implementation, so we took care to define the external interface between the AVR and the FPGA in such a way that its operation was not dependent on the cipher implementation and as independent as possible of the actual cipher. This came into use when we discovered that the synthesized core took up more than 300% of the resources of the Spartan II, even with heavy area optimization by the synthesis tool. This should have been discovered much earlier, but it had not crossed anyone's mind (not even of our advisors) that we might run out of space, and we didn't actually synthesize the encryption implementations until very late in the project (this was obviously a huge mistake). As a result, we needed to find an implementation (possibly of another algorithm) that did not use too much space. The choice fell on the Triple DES implementation listed below, due to its compactness and ease of use. Even though Triple DES is inferior to AES, it is still considered a secure cipher. This involved a change of key size (from 128 to 3×64) and block size (from 128 to 64). Thanks to our memory mapped interface, this change was trivial. We had to implement cipher block chaining, but that amounted to a fairly straightforward wrapping of the algorithm in some VHDL code that handled manipulation of incoming and outgoing data.

3.3.2.2 Implementation candidates

OpenCores: "AES128" [20]

Language: VHDL

Licence: LGPL

Key sizes: 128

Block sizes: 128

Pros: Somewhat small amount of code. Quite clean and understandable code. FIPS compliant test bench included.

Cons: Uses Electronic Code Book, not Cipher Block Chaining (but it should not be too much work to wrap it in some CBC code)

OpenCores: "twofish 128/192/256" [14]

Language: VHDL

Licence: GPL

Key sizes: 128,192,256

Pros: Test bench included. Supports Cipher Block Chaining.

Cons: Huge amounts of code due to support for different key sizes

OpenCores: "3DES (Triple DES) / DES (VHDL)" [13]

Language: VHDL

Licence: Free use and distribution when including original copyright notice

Key sizes: 3x64

Pros: Acceptable amount of code, small footprint

Cons: Somewhat outdated algorihtm (but still considered secure) that is slower than AES. Lacks a test bench.

"VHDL AES128 Encryption/Decryption" [12] (project from students at Bradley University)

Language: VHDL

Licence: None given

Key sizes: 128

Block sizes: 128

Pros: No obvious ones

Cons: A lot of messy code. Built for manual entry of keys and data via keypad.

Figure 3.5: Picture of a ATmega128 microcontroller

OpenCores: "AES (Rijndael) IP Core" [21]

Language: Verilog

Licence: Free use and distribution when including original copyright notice

Key sizes: 128

Block sizes: 128

Pros: Very simple interface

Cons: Written i Verilog

3.3.3 Main controller

We needed a controller to control the transmission of the encrypted audio data. This controller is placed between the encryption unit and the transmission unit. The main task is to control the transmission unit. It has to decide where to send data, and what to do with received data.

In addition, this controller handles the user interface part of the device. This means that the controller has to communicate with several differend devices.

One of our requirements was to use the one of Atmel's microcontrolles. In the previous years, the ATmega128 microcontroller has been the "default" microcontroller. It was recommended by the advisors and we knew it was both a popular and powerful microcontroller. See figure 3.5 for an image of the ATmega128.

The ATmega128 microcontroller comes in two flavors — the ATmega128 and the ATmega128L. The ATmega128 runs at speeds up to 16 MHz on 5 volts, and the ATmega128L runs at speeds up to 8 MHz on 3.3 vols.

To interface with the other devices in the system, and in particular the FPGA, we chose to use the low-speed and low-voltage version. We were a bit worried about whether it was quick enough, and did some quick calculations of the throughput. In the end we were quite certain that it was quick enough for our use.

3.3.4 Transmission

There were several alternatives that we could use to implement the transmission. The choice stood between using a fully featured GSM/UMTS module for the communication part. Another choice we had was to just use a bluetooth module to communicate with an ordinary cellular phone. We also discussed the option using a serial connector between the device and mobile phone. Universal Serial Bus (USB) was also considered, but USB turned out to be too time

consuming to implement. The next choice we had to make was wether we wanted to use regular GSM or UMTS. UMTS today does have not the same coverage rates as GSM has. We want to be able to use the cryptophone almost anywhere, therefore we chose to use the GSM network instead of UMTS. And to reduce complexity we finally decided to use a fully featured GSM module as our choice for transmission.

3.3.4.1 Siemens MC75

One GSM-module we took into consideration was the generic MC75 module produced by Siemens. It is a GSM-module that also supports GPRS and EDGE for higher bandwidth than pure GSM-data. It supports quite a few interesting features as well. The product specifications are located in the appendix.

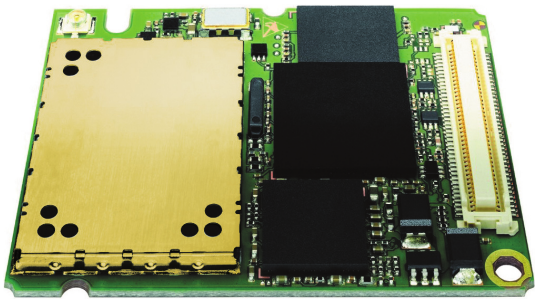


Figure 3.6: Picture of the Siemens MC75 GSM module

Evaluation of the module

- **Pros**
  - A generic solution which can be integrated in many combinations
  - Lots of different interfaces
  - EDGE support
  - Good price (below 1000 NOK)
- **Cons**
  - A complex connector, 80 pin application port which makes PCB layout complex
  - Nothing is integrated on the module, we'll have to add everything we need (SIM card reader, memory card...), which makes PCB layout even more complex and time-consuming

Conclusion As the specification tells us, this GSM-module has quite a few features. It supports both USB and serial connection. The module also has a Interface for Secure Digtal flashram (Secure Digital (SD)-card) that makes it possible to store the phonebook on a SD-card. It also has a Transmission Control Protocol (TCP)/IP stack that can be configured through AT-commands. The module provides us with functionality that are not available with a plain GSM-module. It can be bought through one of The Norwegian University of Science

and Technology (NTNU)'s standard suppliers. Link is provided in the end of this text. The possibility to create a direct connection between two mobile units over GPRS/EDGE must be researched as well as what the latency will be in that case. If this can be done we will have much more bandwidth than if we only use GSM-data. But because of the complex 80-pin connector, this module was not chosen for our project. For more information, please see the MC75 Product Brief[3].

3.3.4.2 Round Solutions GM862

Another GSM-module we took into consideration was the same GSM-module the Priori project used, the Round Solutions GM862. Specifications are located in the appendix.

Evaluation of the module

- **Pros**
  - The module has been used in an earlier computer project (Priori)
  - Standard pinout without any obscure connector like the 80pin application port on MC75
  - Acessible by AT commands
- **Cons**
  - Doesn't support EDGE
  - Even if the pinout isn't as complex as in the 80 pin application port, the layout is rather complex compared to SocketModem EDGE
  - As for the Priori group the power consumption specified in the datasheet is inaccurate. It uses more power than the specification tells us
  - Non-generic AT commands
  - Difficult to acquire. The Priori group had to wait several weeks for delivery
  - Outdated

Conclusion The reason to use this module was because this module was used in an earlier project, the Priori workout computer, which sent pulse, velocity and Global Position System (GPS) position over GPRS to a web server. We thought this would save us some work. But after talking to Jahre, one of the participans of Priori, we learnt that this was not the case. The GM862 used a lot of power, the AT commands gave them a bit trouble, especially the GPRS connection, which was managed through an "EASY GPRS" AT command extension, which turned out not to be so easy. In addition they had to wait a long time for delivery of this module. Besides, the Priori report contained very little documentation for the GM862, so we wouldn't had saved any work by choosing this module. This module was actually chosen at an early stage of the project, but in light of the information from Priori, this choice was undone.

3.3.4.3 Multitech SocketModem EDGE

The module that we chose is the Multitech SocketModem EDGE. It is based on the MC75 EDGE-module described earlier. The main difference between the generic MC75 and the SocketModem EDGE is that the SocketModem is an integrated solution with simple connectors (standard pinout) for RS232, power, and reset, separated into groups. It uses the Universal Socket connectivity that can easily be integrated on our PCB. A SIM card reader is also integrated on this module, and we don't have to deal with the higly complex 80pin application connector that the generic MC75 module uses, which will lighten our workload. The feature list can be found in the appendix. Figure 3.7 shows how the modem looks like, and figure 3.8 shows us the corresponding antenna we have to use with the modem.

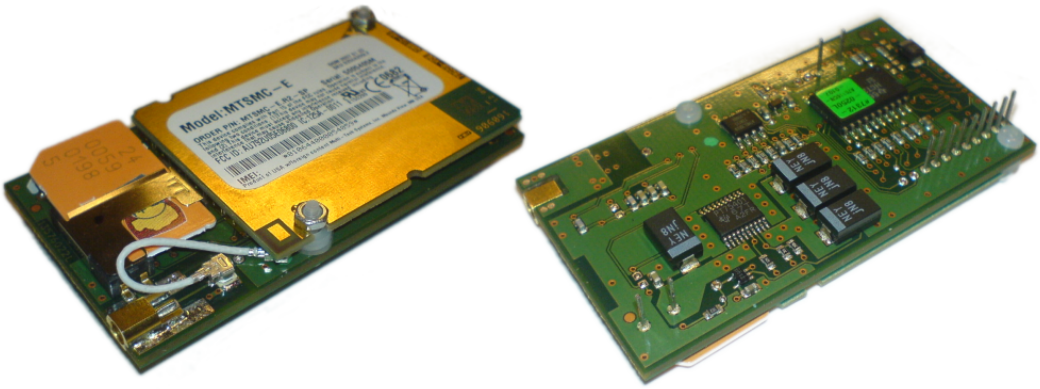


Figure 3.7: Top and bottom picture of the SocketModem EDGE module

Figure 3.8: Picture of the antenna recommended by Acte, the reseller of the SocketModem EDGE

Evaluation of the module

- **Pros**
  - SIM-card reader integrated to module
  - Based on Siemens MC75

  - EDGE support
  - Simple pin-out which makes it fairly easy to integrate it on the PCB

- **Cons**

  - Does not take fully advantage of the generic MC75 features

Conclusion Of the alternatives we evaluated, the Multitech SocketModem EDGE based on the Siemens MC75, was the best choice. This module replaced the GM862. It combines a rich feature list, including EDGE, with simple connectors/pinout and integraded SIM card reader.

3.4 Power supply components

Battery and Charge Controller This was probably one of the most unchanged parts from last year's project that were integrated into our device. None of the participants on the project had much knowledge about how to make a battery charger. We therefore decided to use the design from last years project with a 7.2V NiMH 2000MAh battery, and we found two batteries in the laboratory that we could use. This design uses a MAX712 battery charge-controller IC from Maxim that is specifically designed for charging NiCd/NiMH batteries from 1 to 16 cells at rates up to 4C[1]. The only thing that we changed was that we have added a 1.8V circuit. And we chose to supply the reset circuit with 3.3V instead of 5.0V.

LM1084 Voltage Converters The choice of voltage converters was delayed until we had decided on most other components. In the start we were not sure if the AVR was going to run on 3.3V or 5V, but we knew that we would need voltage regulators for both voltage levels since the GSM-module required 5V and the FPGA required 3.3V. The FPGA also needed 1.8V, so we needed a regulator for that voltage too. All the voltage regulation was done by using three 5A low dropout positive regulators called LM1084 from National Semiconductors. We used one LM1084-it-5.0 for 5.0V, one LM1084-it-3.3 for 3.3V and one LM1084-IT-ADJ for the 1.8V.

DS1233 Brown out detector This chip makes sure that the voltage gets held low long enough for the chips to register that the RESET button has been pushed. This chip needs to be the 3.3V version or else the voltage will be held low.

CAP-XX GS203F The reason why we choose this specific capacitor was because it was recommended for use with the Multitech SocketModem EDGE by our retailer, ACTE AS[1]. The purpose of using this capacitor is to offload the Power Supply Unit (PSU) because the GSM-module creates drain burst up to 2A. Figure 3.9 shows a supercapacitor of this type. Highlights of the specifications are placed in the appendix.

[1] Charge rates are typically normalized to the capacity of the batteries. A 1Ah battery charged completely in 15 minutes is charged with a charge rate of 4C.

Figure 3.9: Picture of a CAP-XX supercapacitor

3.5 Choice of Peripherals

3.5.1 Display

We needed some kind of user interface and to keep things simple, we quickly decided to use the same display as last year's project group. This simplified our design process because we did not need to order it, wait for it, or do any research to find an alternative. We did not need a very advanced or big display since it was only going to be used for displaying status messages and show input feedback. The GTC-40045-YS6L0S has four lines able to display 40 characters each. The display also has a backlight-feature which was very nice since we wanted the final product to be usable in dark environments. Many early commercial cell-phones had just one line of numerical display so four lines of 40 characters each should be sufficient. Figure 3.10 shows the front and back of the display.

Figure 3.10: Front and back picture of the GTC-40045-YS6L0S display

3.5.2 Keypad

We needed input buttons to dial, hang up and navigate through the user interface. We found some simple numeric keypads on the laboratory and because of these keypads' relatively simple interface and fit look, we quickly agreed to use them.

The keypads have buttons for the decimal digits and the letters A to D, but not E and F. If we are going to enter crypto keys manually, we must either assign two other buttons the role of E and F in order to use hexadecimal, or simply use another number base.

Figure 3.11: Picture of the numeric keypad

3.5.3 Flash memory

MMC and SD-cards are very popular to use in microcontroller applications because they support a rather simple interface called Serial Peripheral Interface Bus (SPI). In systems involving AVR microcontrollers and large external storage, SD or Multi Media Card (MMC)-cards are extremely often the format of choice since many of Atmel's microcontrollers have an SPI-interface implemented in hardware. Because of this we were able to find numerous well documented projects demonstrating such interfacing. We finally decided to choose SD over MMC since cards and sockets of the high availability and low price.

3.5.4 Extra memory

We were uncertain about the amount of memory we would need to implement our software, and decided to include some extra memory on our board in case it would become necessary. We had several criteria for our choice of memory:

- Voltage — the memory should run on the same voltage as the microcontroller and the FPGA.
- Size — the memory should have a size of at least 64 KB.
- Form factor — to ease installation and debugging, the memory should have a standard Dual In-line Package (DIP) form factor.
- Availability — we were going to order some components from Farnell, and we limited our search to Farnell.

After looking at the available memory chips, we settled for the BS62LV1027 memory chip from Brilliance Semiconductor, inc. This is a SRAM memory chip has a size of 128 KB and comes in a standard wide DIP package [8].

3.6 PCB

The goals for the PCB is as follows, all the components has to have footprints that are correct. Also there has to be enough space between the components making it easy to solder them, and changing them if it has to be replaced. The components has to be placed out in a logical way that makes the routing as easy as possible. The components need to be placed in a way that groups the components with different voltages. To make it easy to debug the board it needs a lot of jumpers that makes it easy to sample the signal, insert a signal or reroute the signal. All extra pins should also be routed out to jumpers, making it easy to connect buttons, Light-Emitting Diodes (LEDs) or other things that might be needed. Connection points for the power cable, speaker, microphone, GSM and SD card should be placed close to each other and on one side of the board if possible.

To reduce the price of printing the PCB the number of different hole sizes should be as small as possible. The size of the signal routes should be at least 6 mil[2]. The area of the board also inflicts the prize, so the board should not be bigger than necessary.

3.7 Software design

The different hardware components need to have a way to communicate and work together as expected. The way this is solved is by controlling the hardware with the help of software.

The software takes care of the user interface and sends commands to the other components on behalf of the user.

The software is built up from low-level drivers for the display, keyboard, GSM module and encryption. On top of these, a main program is built which controls the user interface and makes calls to the low-level modules as needed.

3.7.1 The menu system

The user interface consists of a set of menus, between which the user may navigate. Figure 3.12 shows the high level structure of the menu system.

In addition to the main task of making encrypted phone calls, the menu design includes support for SMS messages as an optional possibility.

3.8 Casing

Since the Dulkóðuð leyndartut is a GSM phone it should be portable. A way to make it portable is to put it into something that is easy to carry, like a suitcase. The Dulkóðuð leyndartut should therefore be placed into a suitcase with holes to the components that needs to be plugged in and out. The case should be as close to the size of the PCB board as possible.

[2] **A thou, also known as a mil**, is a unit of length equal to 0.001 international inches (1 international inch is equal to 1,000 thou)

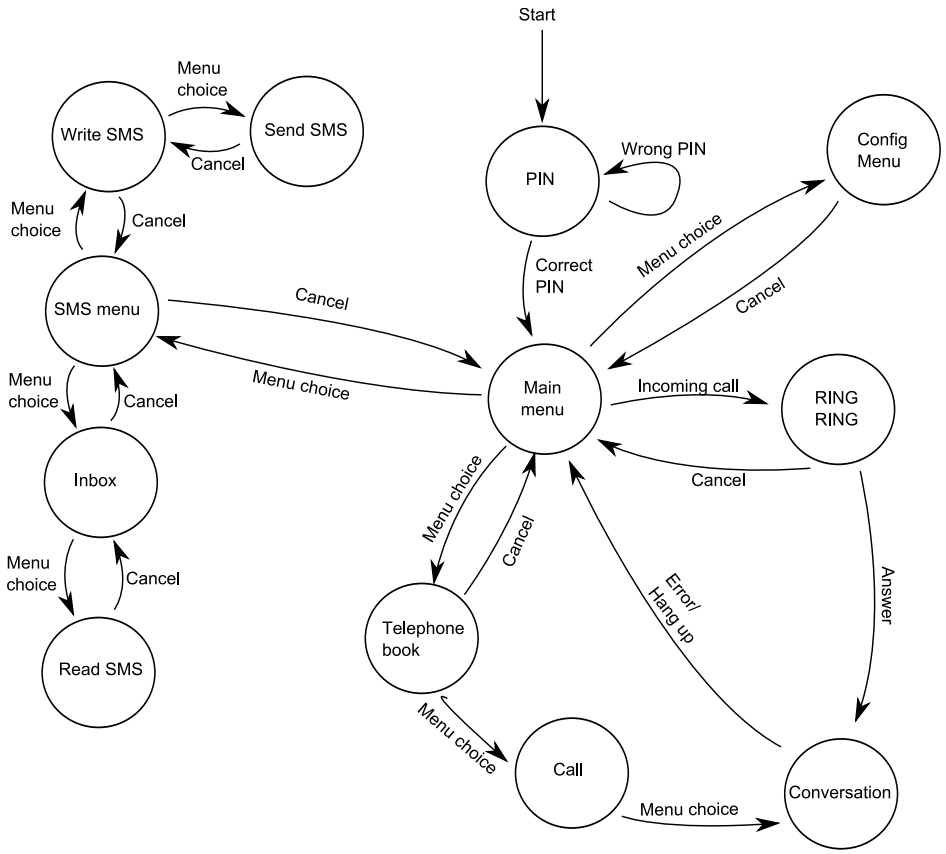


Figure 3.12: Diagram showing the menu system of the Dulkóðuð leyndartut.

Chapter 4

Implementation

This chapter describes the details of the actual implementations done. The first section explains which features that are implemented and section 4.2 through 4.8 describes in detail the implementations done.

4.1 Implemented features

The implementation is not complete with respect to the design described in the previous chapter, because of lack of time.

Most of the main data path, including the encryption, is implemented; however, the interface between the FPGA and the audio codec chip is not implemented. Since the sound system is not complete, the software implementation focuses on sending of encrypted SMS messages instead of encrypted calls.

4.2 Physical interfaces

In this section, all the interfaces between the main components are discussed. This includes principal schematics and the actual schematics that were implemented.

4.2.1 AVR

The AVR is the master in the system and controls the four main units it is connected to. Figure 4.1 shows the parts of the system the AVR is connected to.

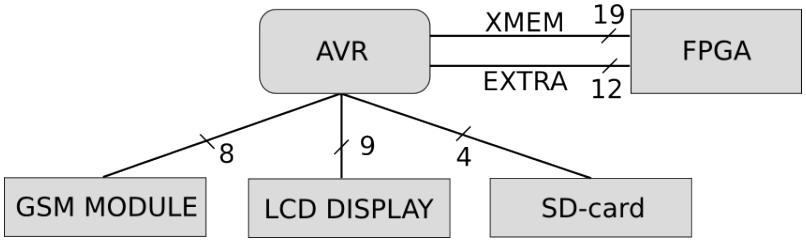


Figure 4.1: The interfaces between the AVR and other parts of the system

4.2.2 GSM module

The SocketModem EDGE communicates with the ATmega128L over a serial UART interface. The UART interface consists of a transmission and a reception line, and in addition, the module also has pins for flow control. Since the GSM module operates at 5V and the ATmega128L at 3.3V, we connected resistive voltage dividers to the output pins of the module to reduce the signal voltage from the GSM-module to the ATmega128L. The voltage dividers only work one way but this is not a problem since none of the communication lines between the ATmega128L and the GSM module are bidirectional. According to the DC characteristics listed in the datasheets, the output voltage from an asserted output pin on the ATmega128L is at least 2.2V when its VCC is 3V, and the minimum input voltage to the GSM-module is 2.0V. See figure 4.2 for a principal diagram of a resistive voltage divider. Figure 4.3 shows all the communication lines between the AVR and the GSM-module.

The SocketModem EDGE is supplied with power from our 5V regulator circuit and shares this only with the LCD-display. We placed a CAP-XX GS203F super capacitor in parallel with and physically close to the SocketModem EDGE to compensate for transient, high current demand periods, to prevent the voltage on the power supply rail from being pulled down by the momentary current load. The GS203F has a capacitance at 0.2F, which according to our reseller should be more than enough to achieve this. According to the GS203F datasheet and information from the manufacturer the capacitor should be balanced with two 22kΩ resistors (see schematic for GSM).

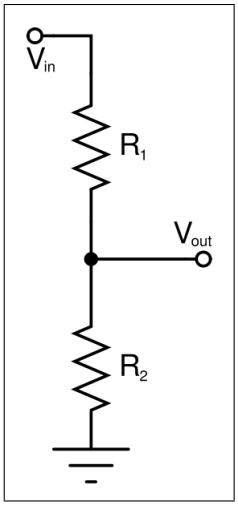


Figure 4.2: Resistive divider diagram

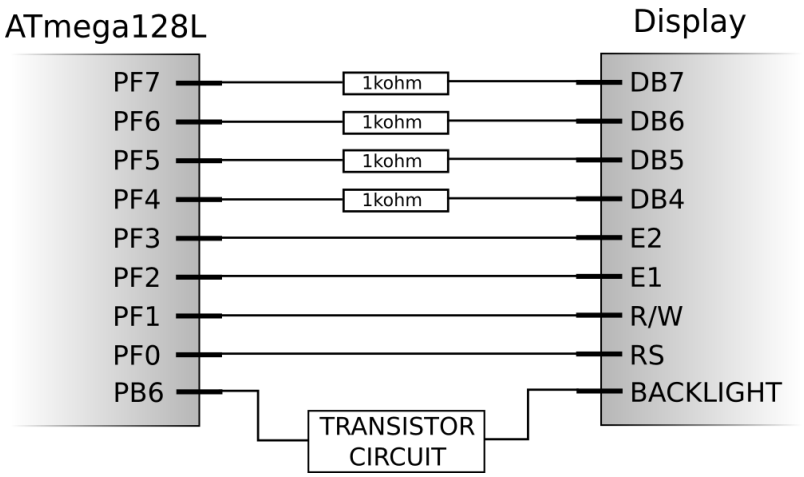


Figure 4.4: Physical interface between AVR and LCD-display

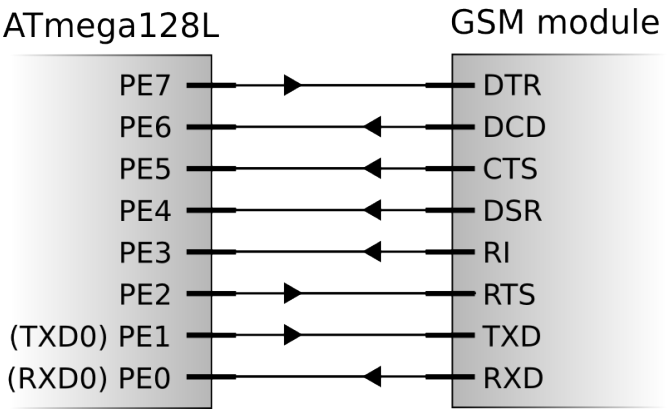


Figure 4.3: Signal lines between the AVR and the GSM-module. All lines from the GSM to the AVR are indirectly connected via voltage dividers.

4.2.2.1 LCD display

The LCD display we chose has a relatively standard interface needing four pins for control and at least four pins for data. We chose to run the display in 4-bit mode because we wanted to save the AVR's I/O pins for other things and because we did not need the speed achievable in 8-bit mode. Figure 4.4 lists the lines that connect the AVR to the LCD module.

4.2.3 SD card

The simplicity of the physical interface between the AVR and the SD card was one of the reasons why we chose it. The AVR has a built-in SPI-interface and all standard SD cards support the SPI protocol. SD cards run on 3.3V supply and input and output signals operate at this level too. This fits perfectly when running an AVR at the same voltage and eliminates the need for even more voltage converters. Figure 4.5 shows the reference design we used to implement the interface and figure 4.6 lists the pin names and functions of the pins when the SD card is in SPI-mode. VCC and GND are connected to 3.3V and ground. All others are unconnected. The

reference design was found in the manual for the The Embedded Filesystems Library (EFSL) project[1].

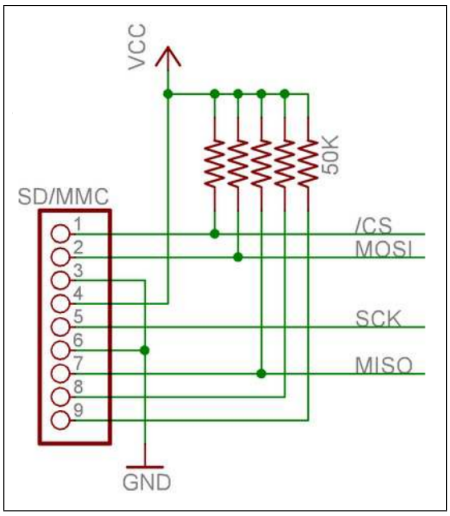


Figure 4.5: SD card interface reference design

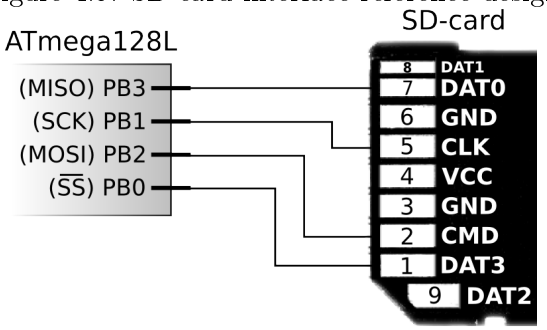


Figure 4.6: The SD card interface.

4.2.3.1 FPGA interface

We had several different possibilities for the connection between the microcontroller and the FPGA. We considered to use the second serial port on the microcontroller for communication. Also considered was the possibility to use our own custom bus. Thirdly we considered to attach the FPGA with the external memory interface of the ATmega128.

In the end we choose the external memory interface. There were several reasons for this. One reason was that we considered to connect an external memory chip to the microcontroller, and to do this we required some sort of latch between the microcontroller and the memory chip. The FPGA could do this job.

However, the main reason was that the memory interface was simple to implement on the FPGA side, and simple to use on the microcontroller side. All we had to do on the ATmega128 microcontroller was to set some I/O registers, and the memory from address 4352 and up to 65535 would be routed out to the memory bus.

All we had to do on the FPGA side was to implement an address latch, and trigger events when we received a read or a write request.

[1]The Embedded File systems Library http://efsl.be/

The external memory interface allowed us to control the FPGA in the same way that we controlled the devices internal to the microcontroller - through Input/Output (I/O) registers. By writing to special memory addresses we could control the different units on the FPGA.

The memory bus The memory consists of 19 signal lines between the microcontroller and the FPGA. These signal lines are shown in figure 4.7. Eight of the lines are combined data and address lines. Up to eight other lines contains the high bits of the address. The remaining three lines are control signals [7].

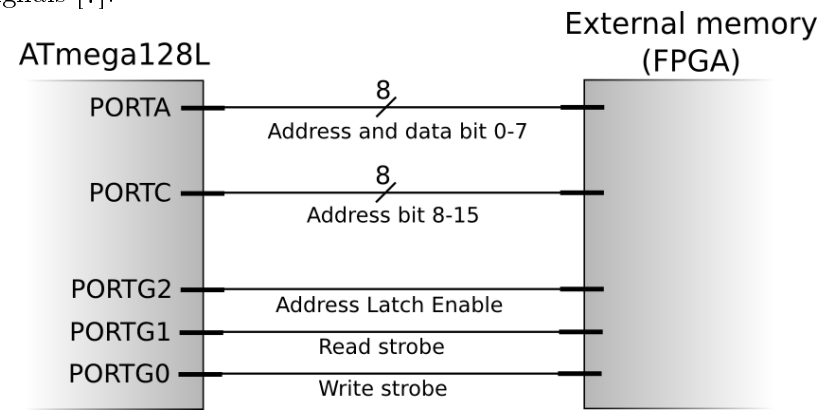


Figure 4.7: The ATmega128L's physical interface for external memory

- ALE controls the address latch. When this signal is high, the eight combined address and data lines contain the lower eight bits of the address.
- RD is the read strobe. When this signal is low, the FPGA should set the eight combined address and data lines to the value of the memory location specified in the address.
- WR is the write strobe. When this signal is low, the FPGA should store the data on the eight combined address and data lines to the memory location specified in the address.

Interrupt lines To allow the FPGA to notify the microcontroller of events, such as keypresses, we allocated several interrupt lines between the FPGA and the microcontroller. These lines also served as backup if it should become necessary to change the interface.

4.2.4 FPGA

The FPGA is one of the main components of Dulkóðuð leyndartut. The fact that the firmware is reconfigurable make it a value part of the system. As we used it we connected all our debuggin leds and buttons to it. This made it possible to check whether it was working at all, but also make it as a debugging platform for the microcontroller.

When the appropriate firmware is in the FPGA it is possible to route the keyboard for the microcontroller trough it, and make the leds available as input for the AVR microcontroller. In addition we put out connection pins for a additional keyboard, which means we can also route the main data input trough. Figure 4.2.4 shows how the different interfaces are connected to the FPGA with corresponding bus sizes. There is still two implicit buses which are not drawn on this figure, the Joint Test Action Group (JTAG) connection and the programming data lines where the PROM could be connected.

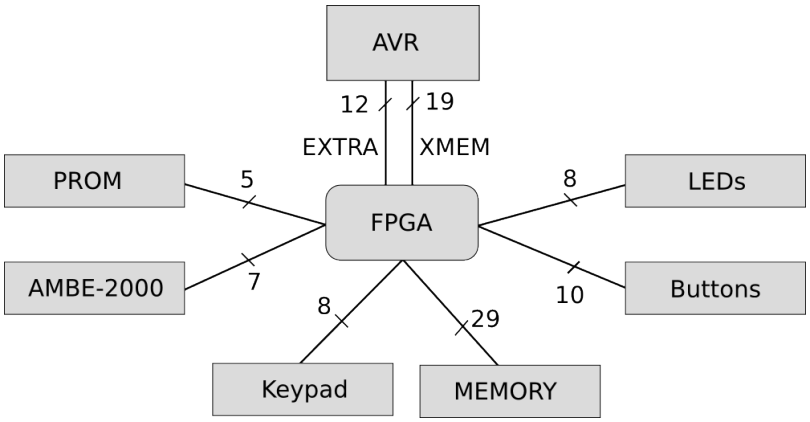


Figure 4.8: Physically interfaces of the FPGA and the other devices

4.2.4.1 PROM

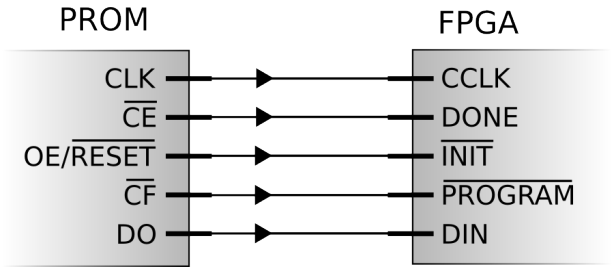


Figure 4.9: PROM and FPGA programming lines

The PROM is a support device for the FPGA for programming it with a definition file. To be able to know how the circuit in the FPGA is defined we need to either upload this definition file via a JTAG cable connected to a Personal Computer (PC) or upload it in other ways. As mentioned before we could have programed the FPGA with its bit file via other devices like an microcontroller with a flash memory. This oppertunity is still possible by using connecting a cable from the extra pins of the AVR to the corrosponding pins of the PROM programming lines shown in figure 4.9. By setting the jumpers over the pins on this programming line we will connect the PROM to the FPGA on its programming lines, activating a serial transfer of the bit file.

4.2.4.2 Speech codec

The AMBE-2000, which was the speech codec we decided to use, uses a serial transmission protocol with three lines in each direction for communication. One of the lines is a clock, one is a signal to synchronize the start of a word, and the last line is the data line [10].

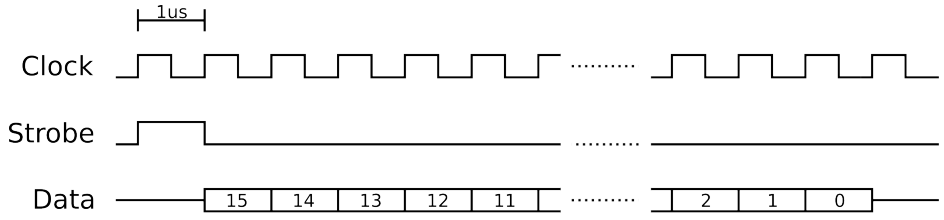


Figure 4.10: Connection between FPGA and AMBE-2000

The clock and the strobe are driven by the FPGA for both transmission directions. The speech codec chip only controls the data line for transmission from the codec chip to the FPGA.

The communication channel transmits 16 bit words, where each bit is set on the rising edge of the clock and read at the falling edge of the clock. The start of a word is marked by a strobe signal. This mode of communication is shown in figure 4.10.

When the FPGA wishes to read a word from the speech codec chip, it raises the strobe signal for one cycle. The AMBE-2000 will then transmit each bit, starting with the most significant bit, in the following 16 cycles.

To send a word, the FPGA has to raise the strobe signal for one cycle. Then each bit should be set on the data line for the next 16 clock cycles.

The AMBE-2000 restricts the speed of the serial channel to 2 MHz. To be well below this line, we selected 1 MHz. This gave us a clock cycle of 1μs.

AD/DA Figure 4.11 shows the simple physical interface between the PCM3500 AD/DA converter and the AMBE-2000 compression chip.

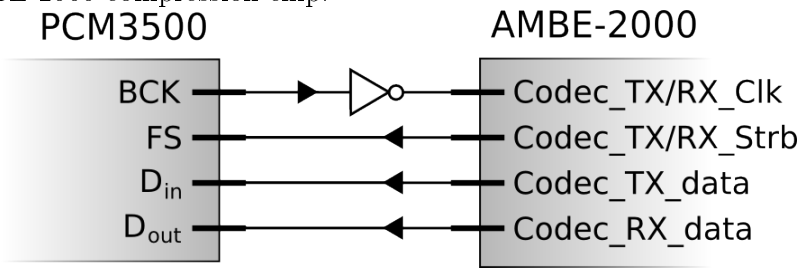


Figure 4.11: Physical interface between the PCM3500 and the AMBE-2000

4.2.4.3 External memory

We wanted to add extra memory in case it we would need it while developing software. Since the FPGA was connected to the external memory interface of the microcontroller, it was simple to add an extra memory chip to the FPGA and route requests for memory through the FPGA.

The memory chip we selected was 128 kB large [8], but we can access only 57088 bytes of this because of address space limitations.

The memory chip has 29 signal lines. These signal lines are shown in figure 4.12.

Address The 17 address lines determine which byte should be read or written.

Data The data lines contain the byte which is read or written.

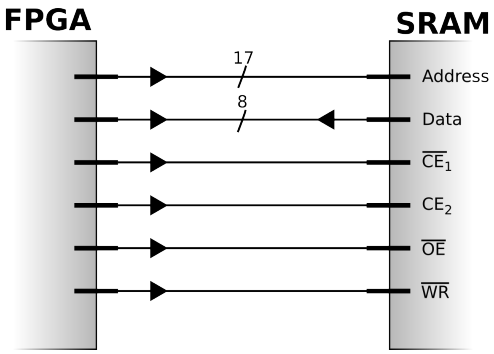


Figure 4.12: Signal lines between FPGA and external memory

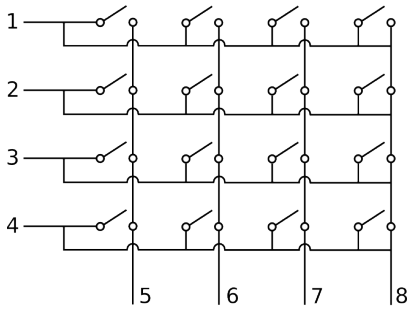


Figure 4.13: Schematic of inner workings of the keypad

CE_1 and CE_2 These lines are used to enable the chip. One of them is active high and one is active low. Both must be enabled to access the memory chip.

WE This signal enables writing to the memory. The data is copied from the data lines to the memory.

RD This signal enables reading from the memory. The byte addressed by the address lines is copied from the memory to the data lines.

4.2.4.4 Keypad

Figure 4.13 shows the schematics of the button array inside the keypad. The output of the keypad is connected directly to I/O pins on the FPGA.

4.2.4.5 LEDs and Buttons

To simplify debugging, 10 buttons and 8 LEDs connected to the FPGA were placed in the PCB design. Pull-down resistors are used on the buttons and the LEDs have current-limiting resistors connected in series. A single LED and button connected in this way is shown in figure 4.14. For the full schematics of all the LEDs and buttons see appendix B.

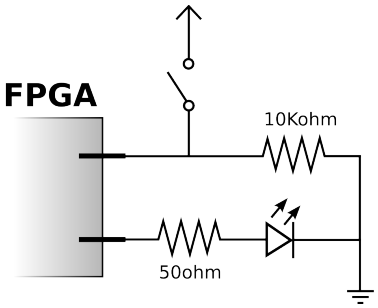


Figure 4.14: LED and button connected to FPGA

4.3 Support electronics

This section describes the electronics that the main units on the board need to function correctly. This includes voltage regulators, decoupling capacitors, power conditioning capacitors, reset circuitry and more.

4.3.1 Power supply

The power supply is a Direct Current (DC) supply of around 12 volts. It needs to be more than 10 volts to charge the battery. On the input to the battery charger circuit there is a rectifier bridge, so that the polarity of the input can be reversed or even alternating.

4.3.2 Oscillators and crystals

Three crystals and one oscillator is included in the design. The oscillator is connected to the FPGA and the different crystals are connected to the PCM3500, the AMBE-2000 and the AVR. The oscillators and crystals are connected to their host units according to guidelines and reference designs in their respective datasheets. All the crystal circuits are designed in the same fashion as shown in figure 4.15. The oscillator connected to the FPGA shown in figure 4.16 is a more self-contained device and requires less peripherals.

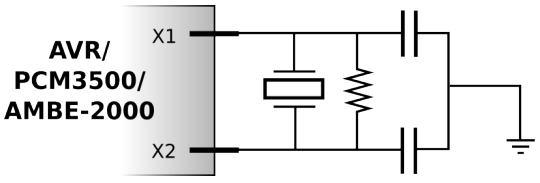


Figure 4.15: Typical crystal circuit

4.3.3 Battery charger

The battery charger consists of a MAX712[18] battery charger circuit from Maxim that supports charging of Nickel-Metal Hydride battery (NiMH) batteries.

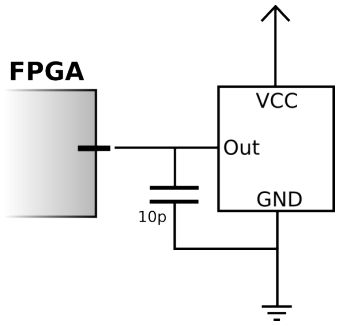


Figure 4.16: Oscillator connected to the FPGA

4.3.4 Reset circuit

The device has a general reset button which resets most of the components, and a separate reset button for the FPGA. Figure 4.17 shows the reset connection.

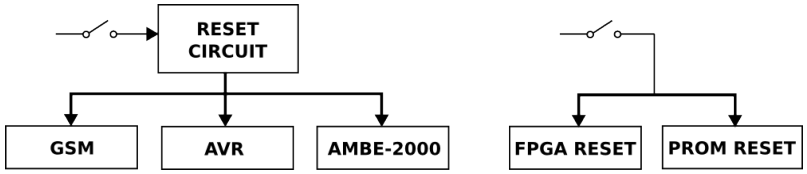


Figure 4.17: The reset circuit

The general reset button controls the signal `RESET_LOW`, which is active low and connected to the microcontroller, the GSM unit and the audio encoding chip. To make sure that the reset signal gets held low long enough for the reset to be registered there is a chip to force this, this is the EconoReset chip DS1233[17].

The FPGA reset button resets the FPGA and its associated PROM.

4.3.5 Analog audio processing

We based our design on an reference design from the AMBE datasheet[10] (in this paragraph refered to as datasheet). The design is calculated for use with a standard telephone handset. A typical handset has according to the datasheet a very small gain in the microphone (voltage levels in the order of 50 mV peak to peak). The PCM3500 expects voltages with a 2 volt peak to peak. The reference design amplifies the input with a gain of 22 dB.

According to the datasheet you have to filter the output from the PCM3500 for maximum voice quality, and the output section is designed with a low pass filter with a gain of 1. Capacitors E3 and E4 (see appendix B) creates a low noise DC bias which lifts the signal to be a positive signal as needed for both the AD and PCM3500.

The oscillator connected to PCM3500 sets which sampling frequency the conversion is done at. The sampling rate is the frequency of the oscillator divided by 512. As the AMBE-2000 expects a sampling frequency of 8 kHz. We connected a 4MHz oscillator (giving 8 kHz sampling rate), which will feed the AMBE with data at the rate it expects.

As this circuit is made for a handset telephone handset we originally included a RJ11 connector on our board. This connector has been replaced with a mini jack from several reasons.

First of all, the availability of telephone handset was not as we initially expected. The handsets we found were integrated with a lot of electronics, and not as clean as they used to be in the good old days. Second is a flaw in the footprint we used for the RJ11 connector.

4.4 PCB

The PCB board has become rather large: 250 mm × 235 mm. The reason for this is that we have connected all the unused FPGA ports and almost all the connections between the different blocks to jumpers. This makes it easy for us to test, and to reroute faulty parts, but it makes the board big, and the wiring a mess. The FPGA and the AMBE2000 chips have their pins so close together that we had to use nets with the thickness of only 5mil[2], this made the PCB a lot more expensive. Figure 4.18 shows the PCB without any components and the layout for the different parts. Figure 4.19 shows a drawing of the PCB with the two signalling layers, silkscreen top and the holes. Layer 1 is in blue, layer 4 is in red.

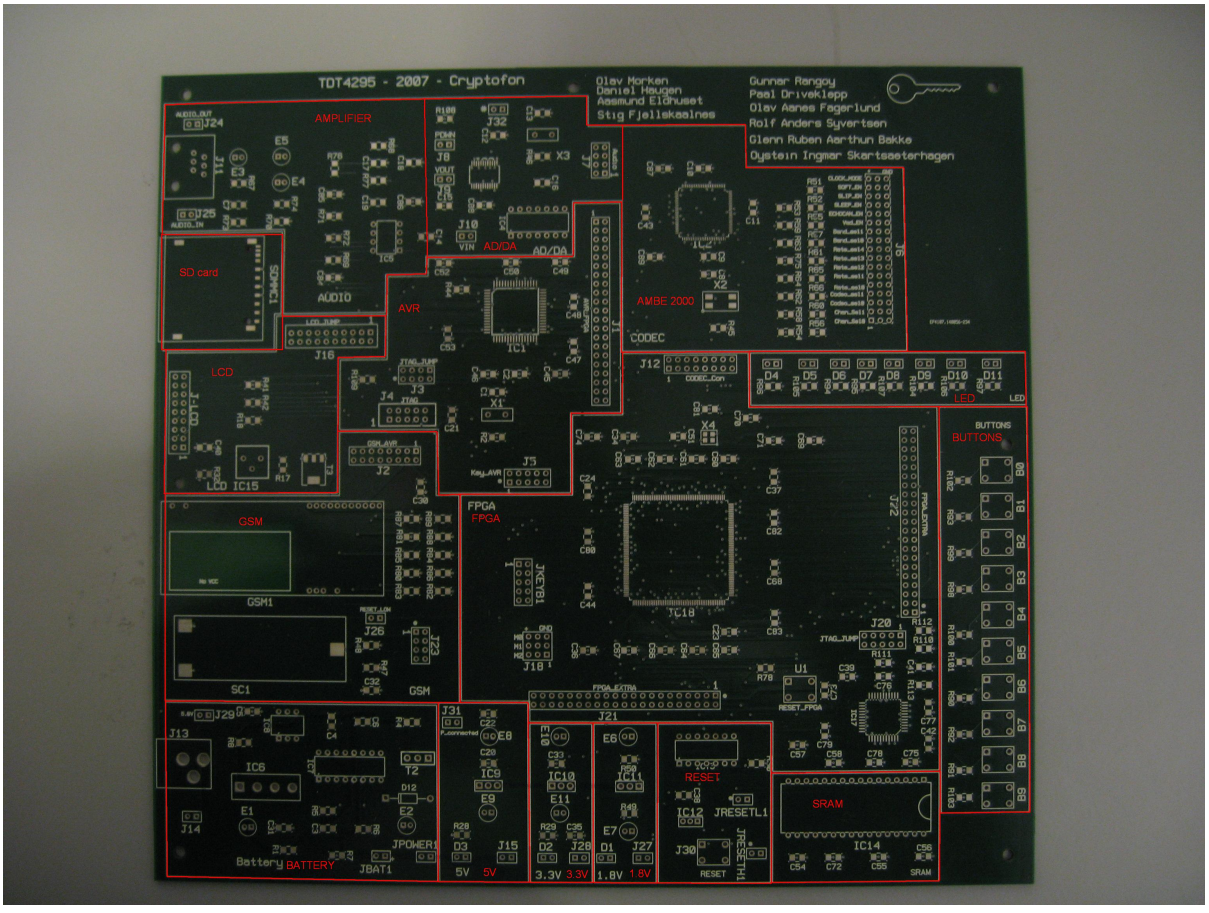


Figure 4.18: The PCB card

[2] **A thou, also known as a mil**, is a unit of length equal to 0.001 international inches (1 international inch is equal to 1,000 thou)

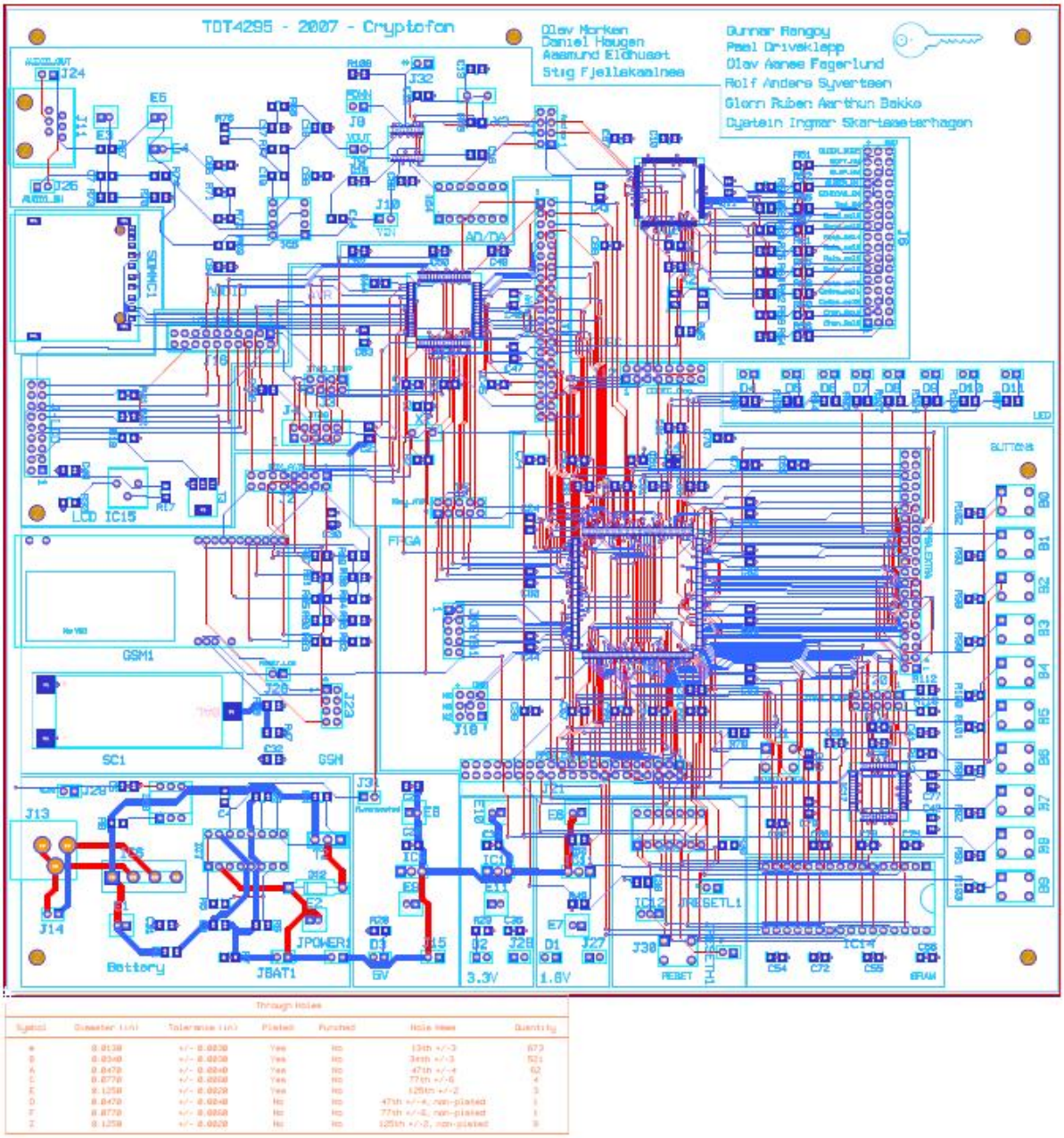


Figure 4.19: The PCB with silkscreen top and holes

4.4.1 Layers

The board consists of 4 layers. Layer 1 and 4 are the signalling layers where all the routed connections are. Layer 2 is the Power Grid and layer 3 is Ground.

4.4.2 Power Grid

On layer two the power grid is laid out. Because of all the different components on our board, we need three different power grids. They are:

- 5.0V: Used by GSM and LCD
- 1.8V: FPGA
- 3.3V: Everything else

Figure 4.20 shows the power grid. The 5.0V layer is on the bottom left in pink, the 1.8V is the yellow area in the lower middle and the blue rest is the 3.3V layer.

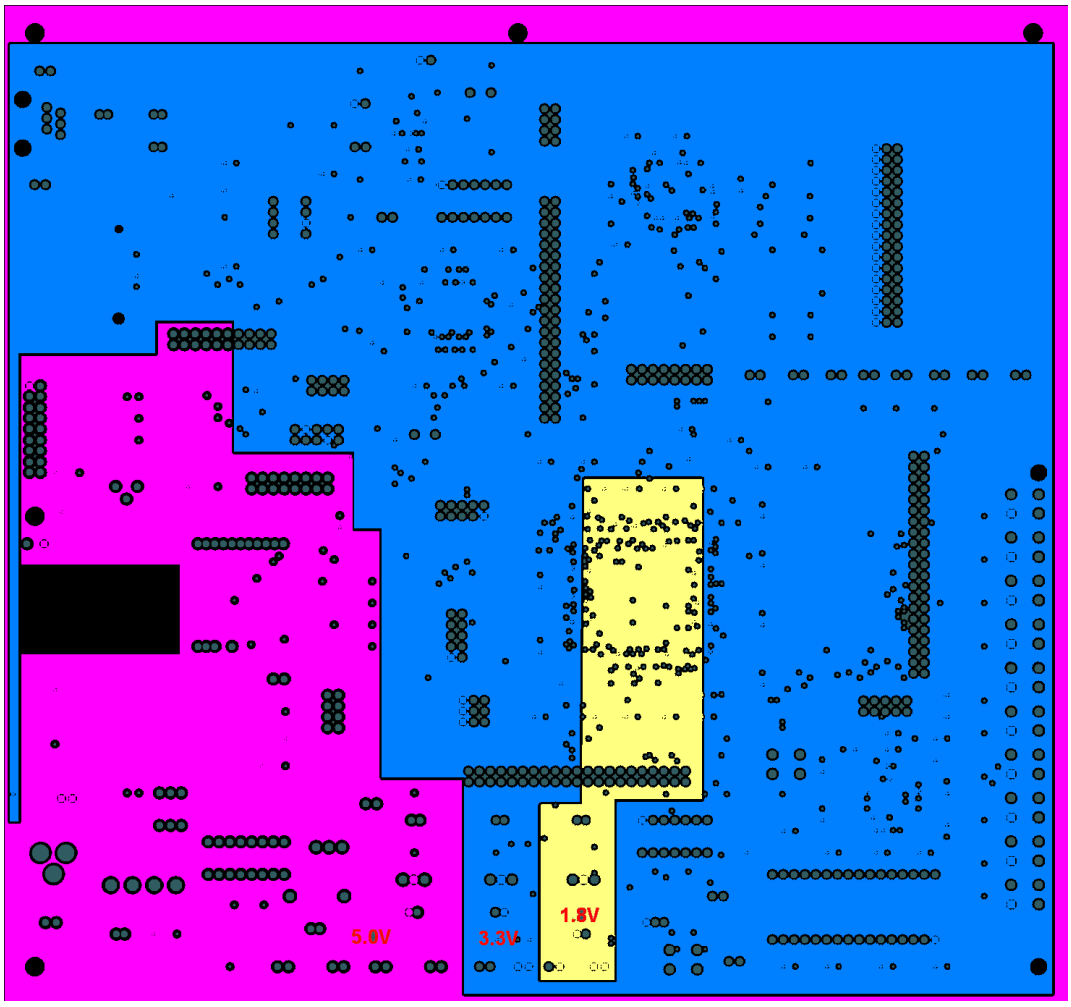


Figure 4.20: Power Grid.

4.4.3 Signalling Layers

Figure 4.21 and Figure 4.22 shows the signalling layers.

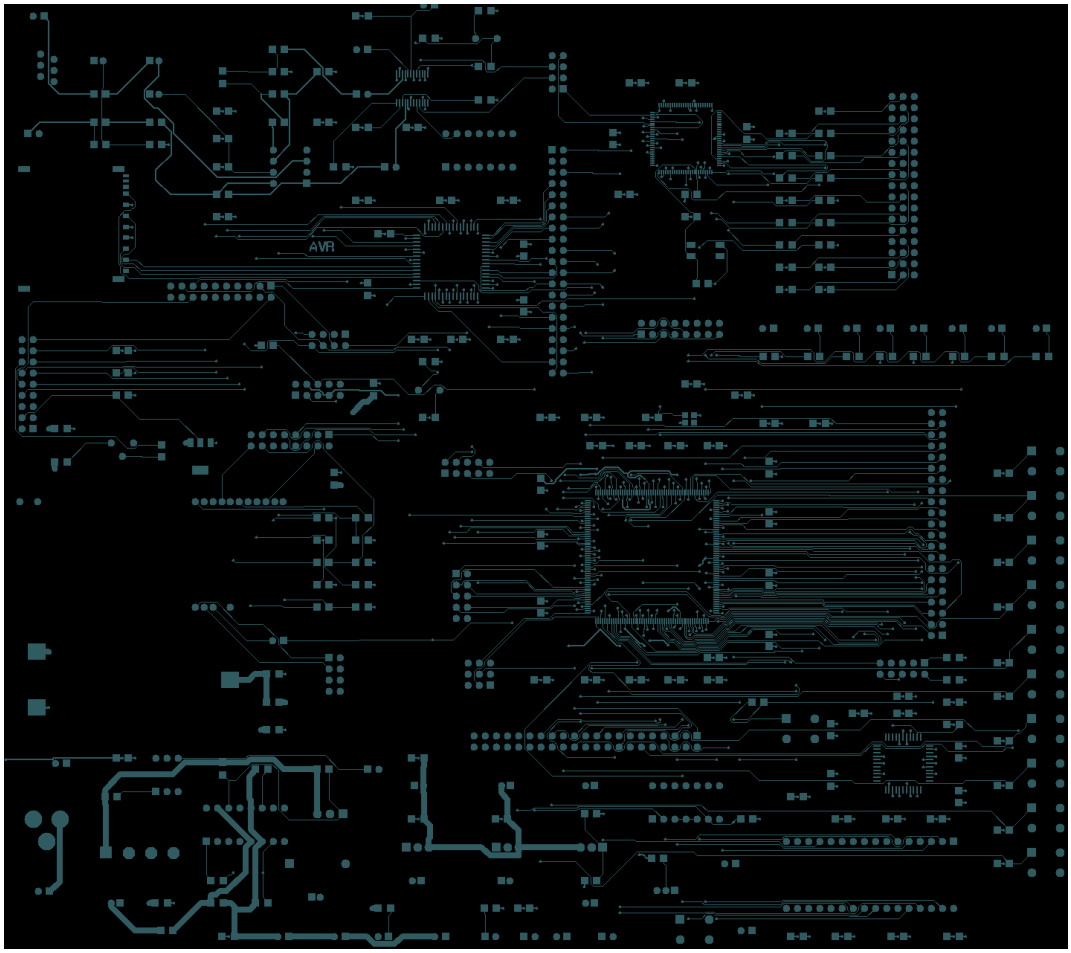


Figure 4.21: Signal layer 1 — Top.

4.4.4 Ground

Figure 4.23 shows the Ground layer.

4.4.5 PCB Holes

There are 8 different hole sizes on the board. We have limited the number of hole sizes as much as possible, but there are a lot of different components with different mounting and soldering holes. The holes are shown in Figure 4.19.

4.4.6 Soldered PCB

The finished card is shown in Figure 4.24

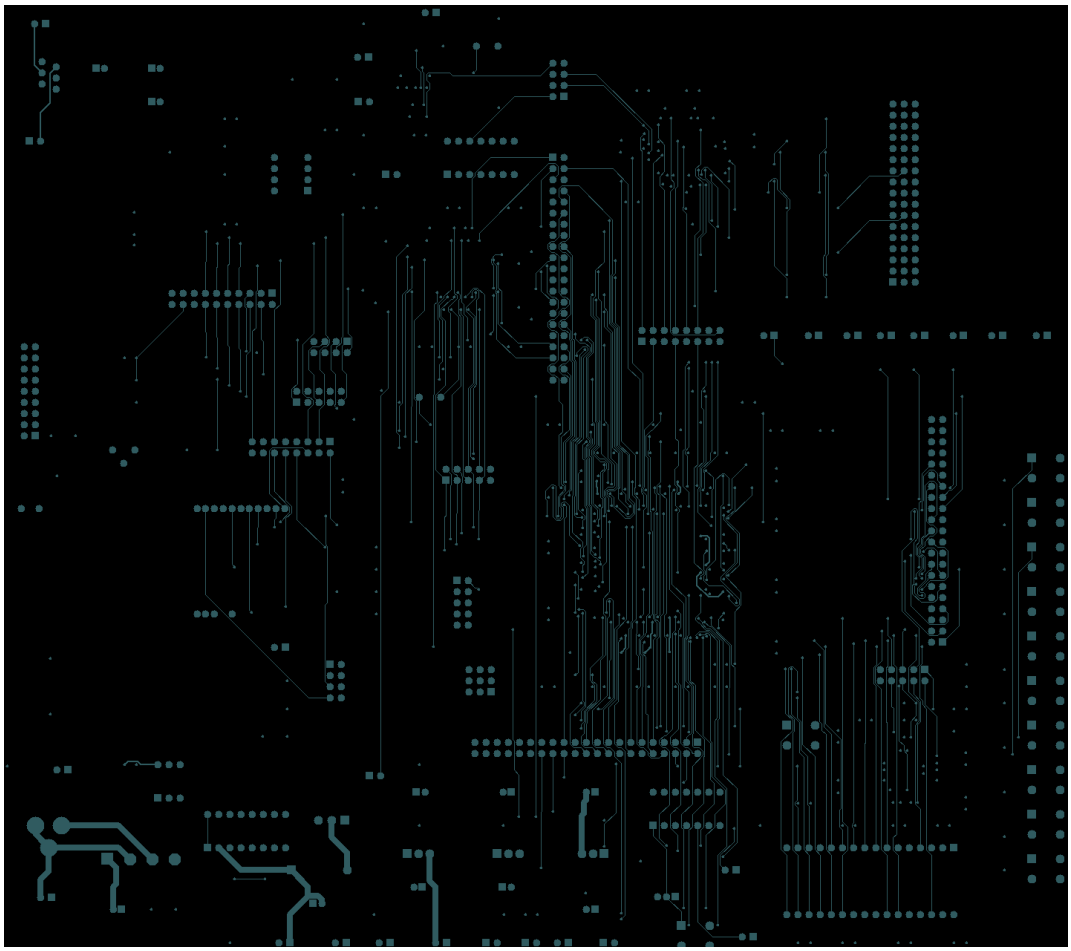


Figure 4.22: Signal layer 4 — Bottom.

4.4.7 Debug and workaround possibilities

The PCB has a lot of workaround possibilities. This has been done to make us less dependent on each part, makes it possible and not to hard to replace parts that does not work.

4.4.7.1 FPGA extra pins

The FPGA has a lot of extra unconnected pins, they are routed out to jumpers, making us able to connect extra components. These pins made the connection from the buttons be possible, but also would serve as a set of extra input/output pins for the FPGA.

4.4.7.2 AVR extra pins

The AVR has a port connected to the FPGA and a backup keyboard connector, making it possible to connect the keyboard directly to the AVR, if the FPGA fails. This port also has interrupt possibilities for sending interrupts from the FPGA to the AVR.

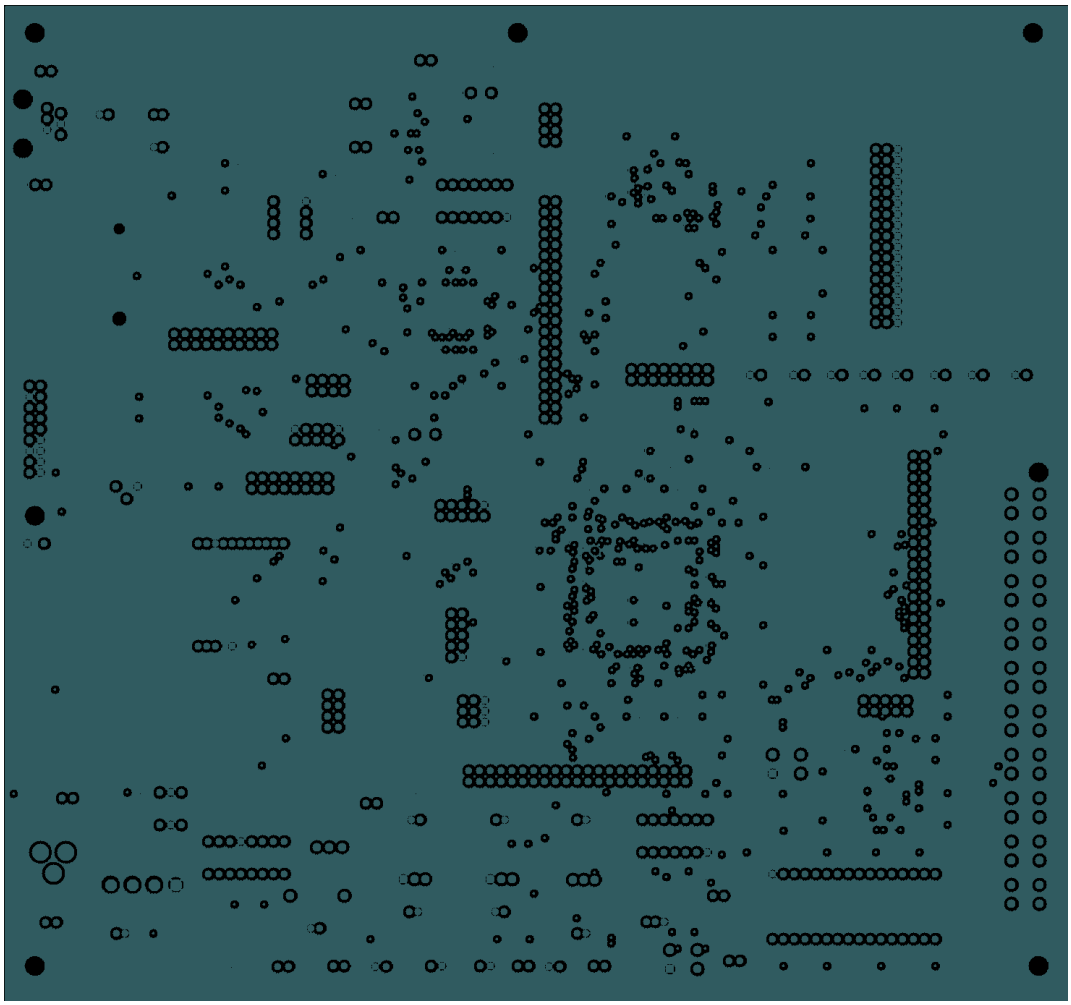


Figure 4.23: Ground Grid.

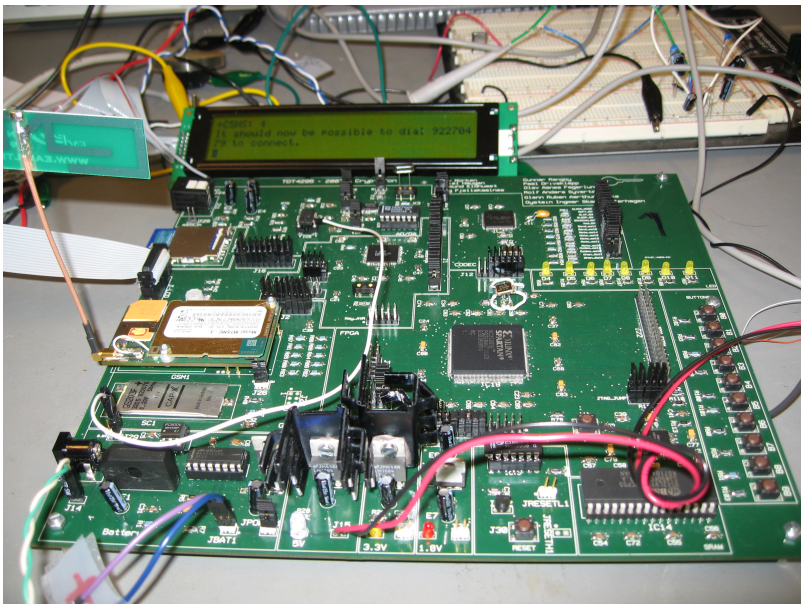


Figure 4.24: Card 1 with all the components.

4.4.7.3 FPGA PROM connection pins

To connect FPGA with the prom. Due to uncertainty we had to make it possible to disconnect the PROM from the FPGA jtag chain. Only if these pins are set we can short circuit the signals from the prom to the programming pins on the FPGA. This is also briefly discussed in section 4.4.7.5.

4.4.7.4 FPGA AMBE connection pins

All the communication pins used for data transfer between the FPGA and the codec chip was put out as a seperate row of pins. These pins was used to debug the AMBE circuit. When the clock from the FPGA was sending signals to the AMBE we could see in a scope that we did not get any data back. Also we could simulate signal by putting a frequence generator on the pins to simulate a clock.

4.4.7.5 JTAGs

We added two jtags for the FPGA and its PROM to keep the jtag chains from each other. But we assured we had pinouts to merge the two jtag chains by short circuit TDO from the first device in the chain to the TDI in the second device. We made put out pins to make it possible to connect all the pins of the FPGA and the corrosponding PROM in a JTAG chain. This chain would not have excluded the microcontroler as it also have its own JTAG connection.

4.5 Workarounds

In this section the workarounds that were made are described in detail.

Pin name	TDI	TMS	TCK	TDO
Pin # on FPGA	159	2	207	157
Pin # on PROM	3	5	7	31
Wire colour	Dark gray or brown	Blue	Light gray	Orange

Table 4.1: FPGA and PROM workaround

4.5.1 Error in Capacitor Polarity Marking

During soldering we found out that the pins on the footprint for the electrolytic capacitors were switched: pin 1 on the component was pin 2 on the board and vice versa. Once discovered, this problem was easily solved by soldering on the capacitors the opposite way.

4.5.2 Error in Audio Circuit Design

The 220Ω resistor R67 had to be replaced by a 0Ω resistor, and a potentiometer could placed on jumper 24 to make it possible to adjust the output volume.

The op-amp circuit did not work with the orginal design. Connecting VCC pin of the op-amp to 5V instead of 3.3V made it work, although the signal is still distorted.

4.5.3 Error in Audio Output Plug Footprint

The J11 cell did not match a RJ11 plug as intended. Instead we soldered in pins and connected them to mini jack connectors.

4.5.4 Missing JTAG Connector for PROM and JTAG

To our luck, we found that vias going straight through the card were made for each of these pins, close to the pin. We were therefore able to connect the needed pins to cables using the vias in order to get the chips programmed, avoiding the ugly workaround of connecting small cables straight onto the needed pins. The pins that the wires were connected to are the following:

To clarify this further we have made two pictures, showing the vias that correspond to the JTAG interface. Both the pictures are mirrored. When you turn the PCB sideways to get to the backside, you will have to mirror the PCB layout to map the right pins. This is already done in the pictures, to make it look just like as if you flipped the board on the side edge. We have marked the actual vias with a white-black-white circle. Figure 4.25 show were the wires are connected on the PCB. Figure 4.27 show the wire coupling for the PROM.

In addition we have also made a pattern map figure. Which indicates which of the vias corresponding to a certain functionality in the FPGA. This is shown in figure 4.26 and 4.28. Note that this is also a backside view. Which means it is mirrored from the pcb schematics.

4.5.5 Oscillator

The oscillator unit connected to the FPGA had soldering pads underneath it and was therefore rather difficult to solder. During testing it was found that the oscillator on one of the cards drained a lot of current and heated up very quickly. It was believed that the reason for this was bad soldering and therefore the oscillator was resoldered many times. Later it turned out

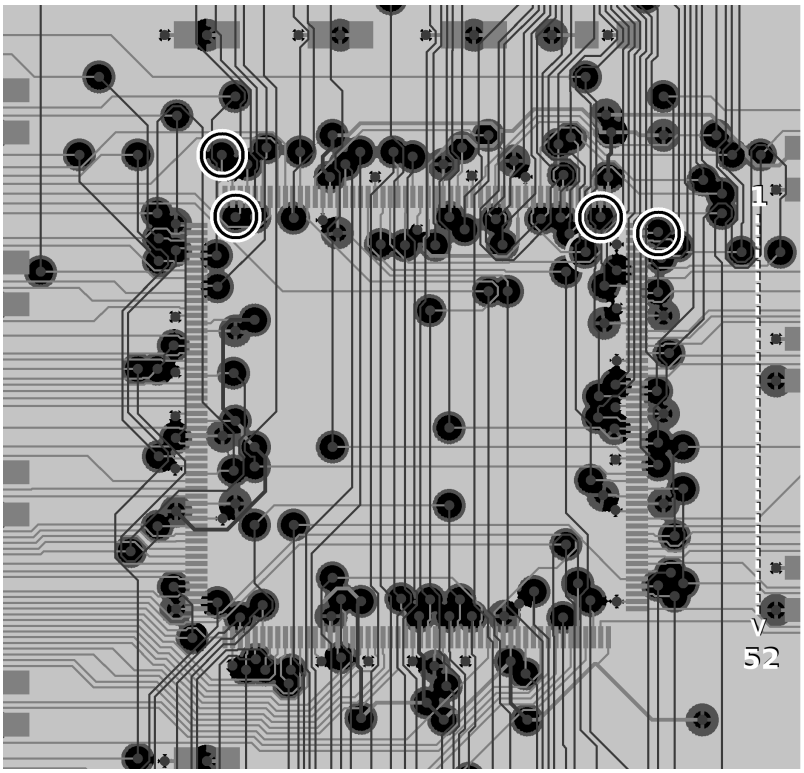


Figure 4.25: JTAG connections for the FPGA on the PCB backside

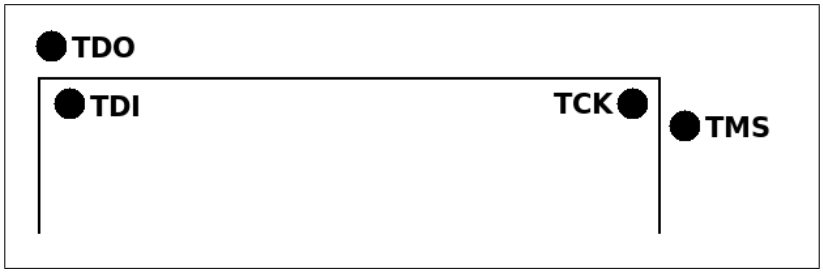


Figure 4.26: The function of the different FPGA JTAG vias

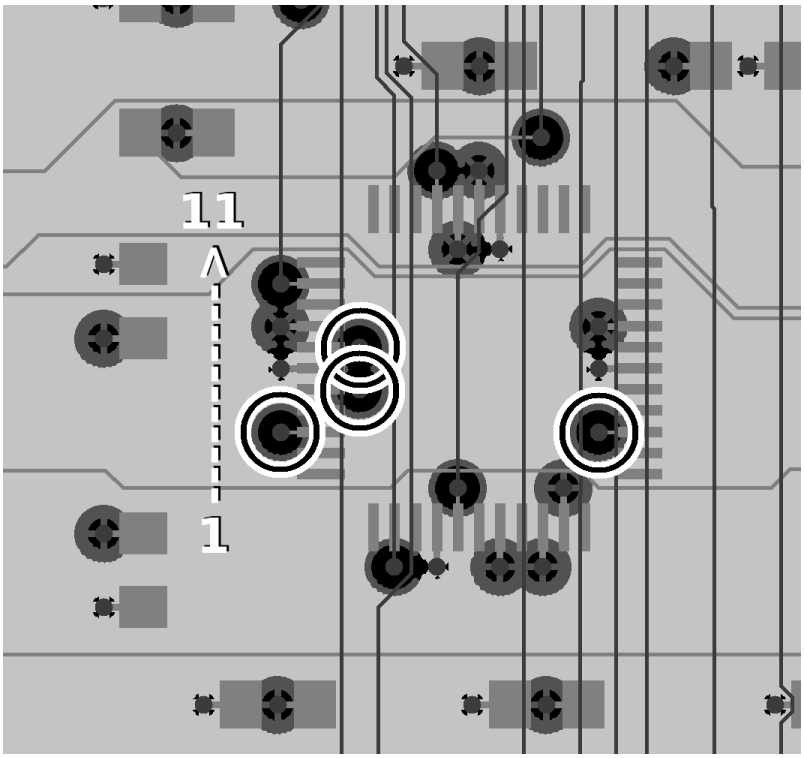


Figure 4.27: PROM JTAG connections on the PCB backside

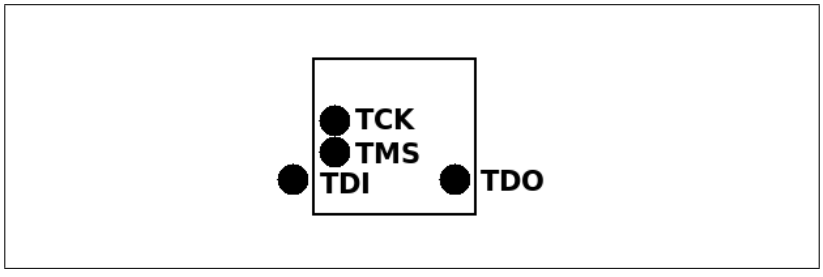


Figure 4.28: The function of the different PROM JTAG vias

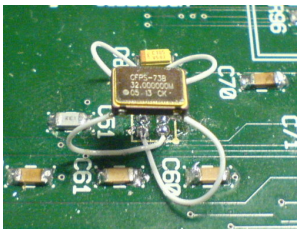


Figure 4.29: Picture of the oscillator workaround

that the oscillator only heated up when it in fact was properly soldered. After an examination of the footprint and the rather unclear description of it in the datasheet we concluded that our footprint was mirrored. We therefore took a different oscillator with a different footprint found at the lab and soldered it to the erroneous footprint using thin wires. Figure 4.29 shows the final result.

4.6 AVR software

The software for the AVR microcontroller is mainly written in C, with some parts in AVR assembly language. A listing of the code appears in section G.2.

The software uses a non-preemptive threading system. There are two main threads, one for the user interface and one for communicating with the GSM module, along with several threads performing supporting functions. The threads communicate by passing messages to each other.

4.6.1 SMS

Sending and receiving of encrypted SMS messages was originally planned as an extra feature. When it was discovered that we would probably not be able to make the audio circuit work in time, however, this became the main focus of the programming.

The messages are encrypted with keys found in the phone book on the SD memory card. When sending a message, the user may either type a number to send an unencrypted message or select an entry from the phone book to make the message encrypted. Upon receipt of a message, the phone book is searched for an entry with the number of the sender and the message decrypted with the key found there. If no such entry is found or if the message is not in a form attainable with the encoding we use (described below), the message is assumed to be unencrypted.

For writing of text messages on the numeric keypad, we have implemented an input method similar to that found on most cell phones. Each of the keys '2' to '9' is used for three to four letters each, and multiple keypresses in rapid succession on the same key switch between the characters it represents.

To make the encrypted data (which may include any eight-bit values) safe to send as text in an SMS message, we use a simple (though space-inefficient) coding scheme. Each byte of encrypted data is encoded as two characters, representing the upper and lower four bits of the byte, respectively. The ASCII values for '0' (0x30) to '?' (0x3F) represent the values 0 to 16.

4.6.2 External memory interface

We have connected the FPGA to the microcontroller through the external memory interface of the ATmega128. See chapter 4.2.3.1 for more details about the physical connection.

The address range for external memory begins at 4352 bytes, where the internal memory ends, and ends at 65535 bytes, which is the limit of the 16-bit address space. This gives us 61184 bytes of external memory, which is much more than we need for memory mapped I/O.

We decided to split the address space at 61440 bytes. The addresses from 4352 to 61439 were to be routed through the FPGA to an external memory chip. This gave us 57088 bytes of external memory accessible to the microcontroller.

The addresses from 61440 to 65535 (4096 bytes in total) were reserved for memory mapped I/O. The microcontroller would write to different addresses in this range to control different units on the FPGA.

This memory layout is shown in figure 4.30. The addresses we use for memory mapped I/O are listed in appendix F.

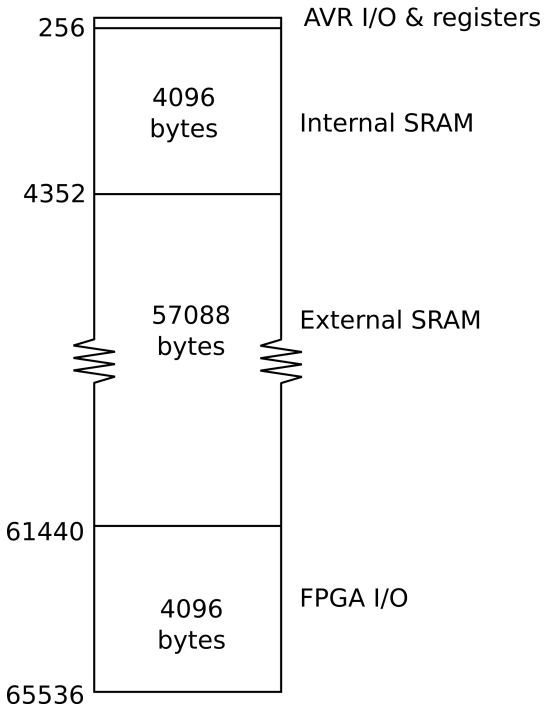


Figure 4.30: Memory space configuration

4.6.2.1 Memory timings

Figure 4.31 show the timing diagram of the xmem interface of the microcontroller interfacing with the implementation of external memory in the FPGA. As you can see there is a little gap between the write signal from the microcontroller and the actual write sigal to the memory. This delay is caused by the internal logic of the FPGA.

It is possible to configure the ATmega128 microcontroller to use different memory timings. We did some calculations about how fast the FPGA would have to respond to the different memory speeds. With the microcontroller configured to use the fastest possible timings, the FPGA has 65 ns to handle to a read request and 110 ns to handle to a write request. With the

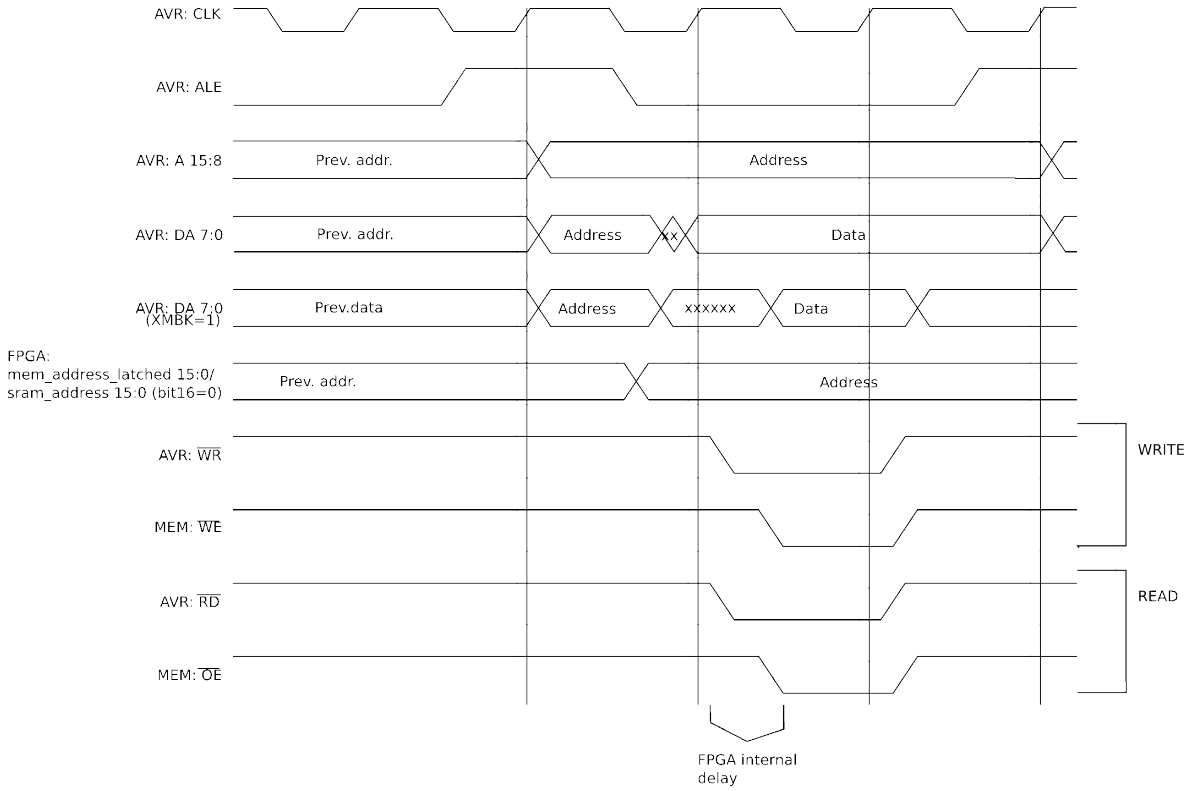


Figure 4.31: Xmem interface timing diagram

slowest timing, the speeds were 315 ns for read requests and 360 ns for write requests.

The read time was measured from the microcontroller set the read signal to the data had to be available at the data lines. The write time was measured as the length of the write signal from the microcontroller. To avoid having any bugs as a result of the FPGA responding to slow to requests, we decided to use the slowest timing.

4.6.3 Interrupts

The microcontroller receives interrupts from the FPGA when a key on the keyboard is pressed or released and when an encryption or decryption task is finished. Since this implementation only encrypts SMS messages and not voice data and therefore does not need high throughput, the encryption/decryption interrupts are ignored and a simple polling scheme used instead.

The microcontroller also receives interrupts when the GSM module is ready to receive data or has data to transmit.

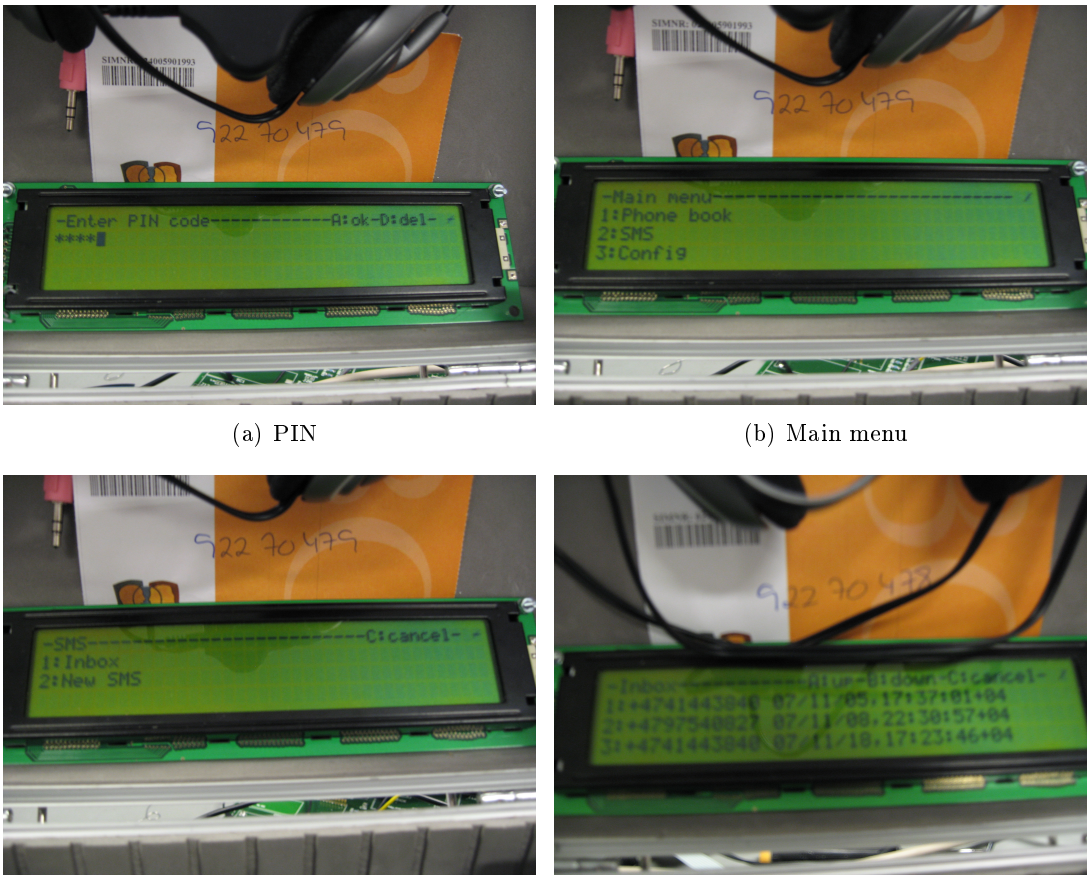
4.6.4 SD card

The SD card is used for storing a phone book with encryption keys. We use the File Allocation Table (FAT) file system and store the phone book as a single file in a simple format.

For reading the SD card, we use a free (GNU General Public License (GPL)-licensed) driver for FAT on SD cards[19], with some minor modifications.

The phone book is not editable from the Dulkóðuð leyndartut. We have written a simple program for use on a PC which lets the user enter the data for a phone book and writes it to an SD card. Each entry in the phone book consists of a name, a phone number and an encryption key used when sending messages to and receiving messages from the corresponding number.

4.6.5 LCD GUI



(a) PIN

(b) Main menu

(c) SMS

(d) Inbox

Figure 4.32: The menu

The graphical user interface of the Dulkóðuð leyndartut is organized into *menus* between which the user may navigate. Each menu fills the entire screen, displaying a title, the choices the user can make and optionally some arbitrary data, such as the text of an SMS message.

Each menu choice has an associated key used for activating it. The 'C' key is always used for exiting the current menu.

The first line of the display shows the menu title and any choices bound to the 'A' to 'D' keys. The remaining three lines show either the choices bound to the numeric keys (in one to

three columns, depending on the number of choices) or arbitrary data; menus which display the latter bind all their choices to the alphabetic keys.

4.7 FPGA implementation

We decided to implement the FPGA as several independent functional units. The different functional units are connected to an interrupt controller and a memory controller. The interrupt controller and the memory controller are connected to the microcontroller. See figure 4.7 for an overview of the functional units we wanted to have in the FPGA.

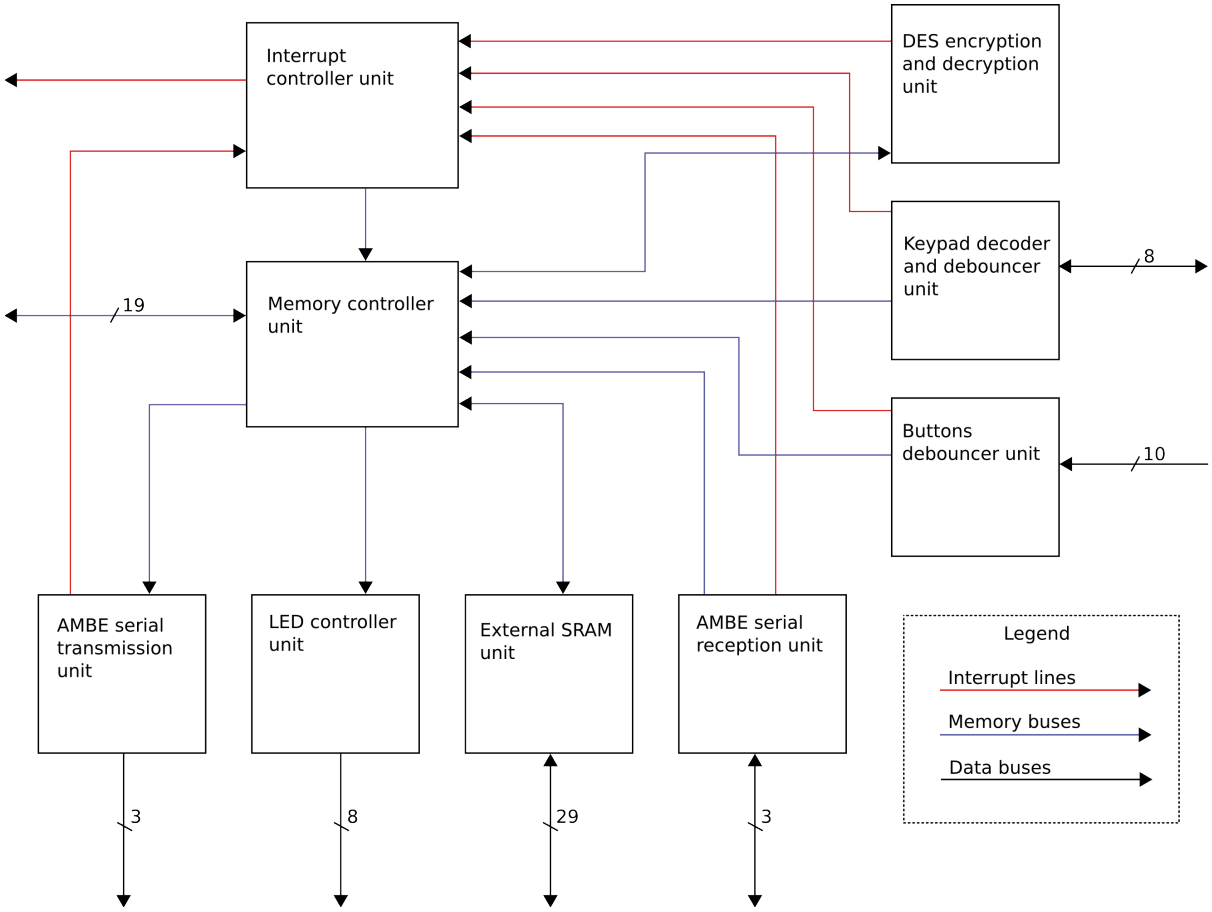


Figure 4.33: FPGA units

The AMBE serial transmission unit is unimplemented and the reception unit was only partially implemented. It lacks a serial port interface. Instead, we can only read the last word we received. All other functional units are working as they should.

4.7.1 AMBE interface

The AMBE-2000 speech codec chip is connected to the FPGA on six pins. See chapter 4.2.4.2 for more information about the connection.

We began implementing the read-side of this connection. We had created a decoder for the serial channel format which read words from the channel continously and stored them in a register. This register was made available to the microcontroller through the memory mapped I/O registers.

Testing this protocol showed that there was no response from the AMBE-2000 on serial line. In the limited amount of time we had remaining we were unable to determine where the fault was. As far as we were able to tell, the FPGA sent the correct signals to the speech codec chip.

4.7.2 Keypad

The keypad is connected to the FPGA with 8 lines. 4 of these are output to the rows and 4 are inputs from the columns.

To read the status of a row of keys, we set the voltage of the row to high and set all other rows to "high-impedance". "high-impedance" means that the FPGA doesn't put any voltage out to those rows. This is necessary to avoid a short circuit between the rows.

After setting the voltage on a row, the keyboard decoder unit waits for several cycles to give the voltages on the pins connected to the columns time to stabilize. After the wait, the column status are read and stored in a register, and the same procedure is repeated for the next row.

The key state from the keyboard decoder is read by a debouncer. The debouncer waits for the keys state to stabilize before giving this information to the controller unit. The controller gives access to the key status through the memory controller. It also raises an interrupt every time the key state changes.

4.8 Casing

The selected suitcase has the dimensions of 290 mm $\times$ 260 mm outside, inside it is 275 mm $\times$ 245 mm. The size is almost perfect for this since the PCB is 250 $\times$ 235 mm. By placing it inside the suitcase the phone has become portable. Figure 4.34 shows the outside of the Dulkóðuð leyndartut.

To get access to the power charger plug, the microphone, the speaker and the SD memory card there has been made holes in the left side of the case. Inside the case the PCB has been mounted in the lower left corner and the battery in the upper right. The summer that beeps when a call or a SMS is coming is placed on the outside of the suitcase in the middle of the top. The power button and the GSM antenna are on the left side of the box. On the right the keyboard is in the upper corner. The LCD has been placed in an angle on the backside to improve the visibility. On the top plate there has been attached screws with mounting to hold the headset in place when moving. All the parts for the case have been found in the lab except the suitcase and the headset which have been bought. Images from the inside of the Dulkóðuð leyndartut can be seen in Figure 4.35. In Figure 4.36 the Dulkóðuð leyndartut is shown in use as a portable phone.

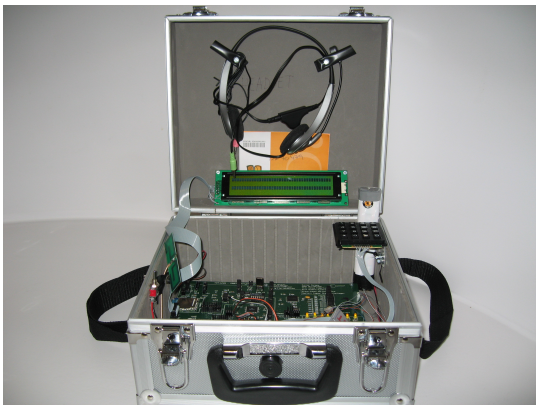
(a) Closed standing

(b) Closed laying

(c) Open

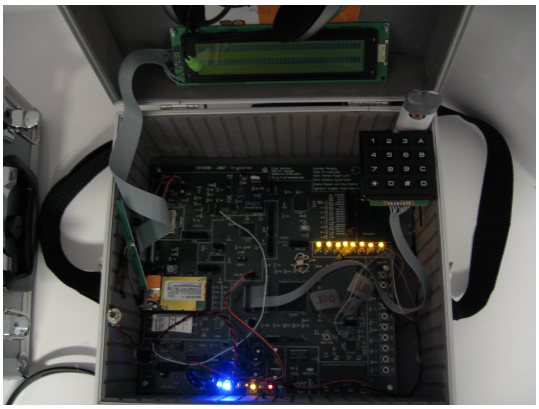
(d) Open with headset connected

Figure 4.34: Images of the outside of Dulkóðuð leyndartut.

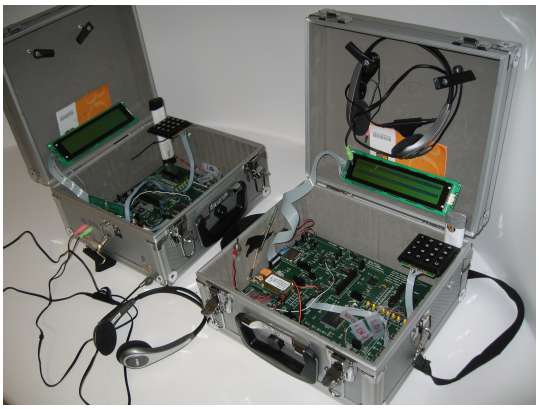


(a) Front

(b) Up and in



(c) With power on



(d) The two prototypes

Figure 4.35: Images of the inside of Dulkóðuð leyndartut.

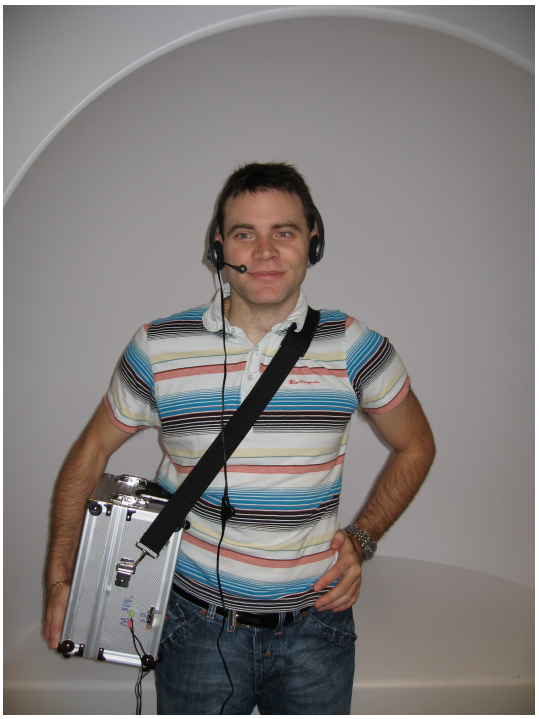


Figure 4.36: The Dulkóðuð leyndartut in use.

Chapter 5

Testing

Before we could connect the whole card into a working implementation of our design, we had to test each of the components individually.

5.1 Hardware

We tested the different components with focus on physical functionalities. An important factor here is the different electrical characteristics of the components.

5.1.1 Audio

A couple of jumpers were included to enable splitting of the audio circuit into isolated parts for individual testing. By tactical placement of jumpers, the input op-amp, output op-amp, PCM or AMBE could be tested independently.

5.1.1.1 Testing of the analog and AD/DA part

The jumper J9 can be used to split out the output circuit, J10 to split out the input, J7 to split the interface between PCM and AMBE and J12 to split AMBE and FPGA.

J32 sets the PCM in loopback mode, which is simply a mode where it feeds the generated digital audio back to the DA converter. In this mode the input is delivered back as a regenerated analog signal, and also as a digital signal out from the PCM.

The audio circuit has different testingpoints which makes testing possible through the path. The first test point is the Audio_in which is the input to the pre-amp. Output of the pre-amp can be tested on the left pin of VIN, and input to the PCM3500 can be injected in the pin to the right of VIN. The output from the PCM can be probed on the rightmost pin of VOUT, and signal can be applied on the left pin of VOUT. Audio_out enables us to probe the signal after output op-amp.

Test results from testing with Frequency generator with different frequencies, amplitudes and modulations is shown in 5.1, with microphone in 5.2 and audio from computer in 5.3. Each test is performed by probing with oscilloscope and listening through speaker/headset.

Even though most of the audio tests is marked as "Passed", they did not pass without remarks. The signal we the PCM delivered was quite dissorted, but they are marked as "Passed" because there is a signal through and it let us continue with our development.

Testing of loop through PCM without preamp failed because the voltage levels for the microphones are too lower than the PCM expects (the PCM expects 2 volt peak to peak [10]). Since the microphone's voltage range is so much lower than expected we did not regard this as a problem.

Test	**Expected result**	**Status**
Preamp	Amplified signal	Passed
Output op-amp	Signal passing through	Passed
Loop through PCM w/amp	Signal out from PCM	Passed
Loop through PCM w/preamp	Signal out from PCM	Passed
Loop through PCM w/output opamp	Signal out from opamp	Passed

Table 5.1: Wave testing

Test	**Expected result**	**Status**
Preamp	Amplified signal	Passed
Output op-amp	Signal passing through	Failed
Loop through PCM wo/amp	Signal out from PCM	Passed
Loop through PCM w/preamp	Signal out from PCM	Passed
Loop through PCM w/output opamp	Signal out from opamp	Failed
Loop through whole circuit	Signal out from opamp	Passed

Table 5.2: Testing with mic

Test	**Expected result**	**Status**
Preamp	Amplified signal	Passed
Output op-amp	Signal passing through	Passed
Loop through PCM w/amp	Signal out from PCM	Passed
Loop through PCM w/preamp	Signal out from PCM	Passed
Loop through PCM w/output opamp	Signal out from opamp	Passed

Table 5.3: Testing with audio from PC

5.1.1.2 Testing of digital circuits (PCM, AMBE, FPGA)

When we had a partially working AD/DA conversion, our next goal was to compress/decompress the digital audiostream and transfer it to/from the FPGA.

The first obvious step to assure that the audio circuit worked as planned was to apply audio to the PCM, and connect an oscilloscope to the digital output pin.

The next important step was to see whether or not we got any signal out from the AMBE-chip. We applied the the strobe and clock signal in to the AMBE, and connected the oscilloscope to the output and analyzed the it.

Test	Expected result	Status
Digital out from PCM	Square wave on oscilloscope	Passed
Digital out from AMBE	Square wave on oscilloscope	Failed

Table 5.4: Testing of AMBE-2000

The AMBE test failed as shown in table 5.4.What we got was just a few milivolts of noise. We saw something which probably was crosstalk from the strobe or clock, as the noise formed a square pulse matching the clock frequency.

5.1.2 AVR

The AVR is a pretty independent and robust chip that only needs VCC and GND connected to function. We started out with checking that this was correctly routed on the PCB by using a multimeter and then we tested all the different connections to other parts of the system.

Test	Expected result	Status
AVR power supply	3.3V between pin 53 and 52	Passed
JTAG connector	Correct signature should be possible to read out	Passed
Physical interface to LCD	Writing "Hello world" to display	Passed
Physical interface to SD-card	Read "helloworld.txt" from the SD-card	Passed
Physical XMEM interface to FPGA	Reading specific address returned data set by VHDL code	Passed

Table 5.5: AVR tests

The AVR has also been tested in conjunction with the units connected to it. See section 5.1.3 for more information.

5.1.3 FPGA

When the FPGA and the AVR had been soldered onto the board (and we had created the JTAGs – see section 4.5.4), the first step was to verify that we would be able to program the FPGA. Since the FPGA is directly connected to the eight LEDs, this was done by writing a VHDL program that simply asserted all the LED signals. We then modified the program to assert one LED at a time in order to make sure that the FPGA could control the LEDs individually – since the LEDs would be our primary way of checking the results of the more advanced tests, it is nice to know that they actually work the way you expect. In order to test the buttons, we wrote a program that would turn on only the LEDs corresponding to the buttons that were pushed.

When we knew that the FPGA could correctly control the LEDs, we moved on to the AVR memory bus. Before actually trying to access the SRAM, we wanted to make sure that the FPGA received the signals correctly (and also that it was able to latch the lower address byte, since the address bus and the data bus share lines); this was also done by writing the data to the LEDs. We then proceeded to actually route the signals through to the SRAM, and see if we could store data in the memory and read it back.

The aforementioned tests were quite ad hoc, and we did not keep the code - rather, the test code gradually was developed into the actual production code. We have, however, written a test bench (listed in the appendix) that tests the encryption. Ideally, we should have used the test vectors from NIST[1] (the "AES128"[20] code contains a nice test bench that reads test vectors from files, and we intended to retrofit it to the 3DES implementation), but we did not get the time to do this. Instead, we generated a few test vectors with a C# program (also listed in the appendix) and entered them directly into the test bench. The test bench is hastily written and not particularly pretty (and it only tests a few of the vectors generated by the C# program, and result verification must be done by manually comparing the output to that of the C# program), but it shows that both encryption and decryption in cipher block chaining mode work, and it demonstrates the memory mapped interface.

Test	**Expected result**	**Status**
The FPGA can be programmed	No error messages from the programming device	Passed
The FPGA can control the LEDs	The bit pattern that the FPGA sends out should be displayed on the LEDs	Passed
The FPGA can react to button presses	The LEDs indicate which buttons are pressed	Passed
The FPGA can receive data that is sent by the AVR	The data is shown on the LEDs	Passed
The FPGA can route memory requests through to the SRAM	Data that is written to the SRAM can be read back at a later time	Passed
The FPGA can encrypt data in ECB mode	We get back the expected ciphertext block	Passed
The FPGA can decrypt data in ECB mode	We get back the original plaintext block	Passed
The FPGA can encrypt data in CBC mode	We get back the expected sequence of ciphertext blocks	Passed
The FPGA can decrypt data in CBC mode	We get back the original sequence of plaintext blocks	Passed

Table 5.6: FPGA tests

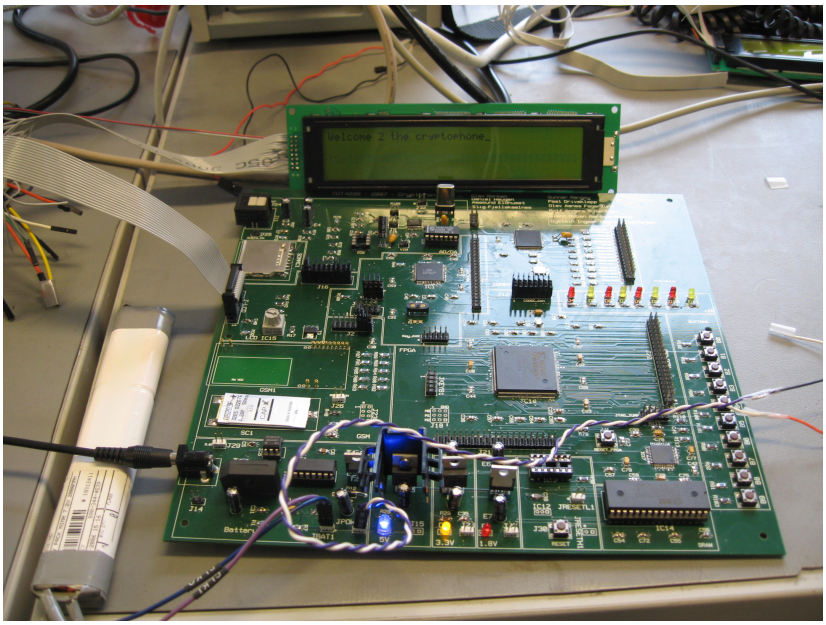
5.1.4 Power

The battery charger and power circuits work as they should. The battery is charged and the 1.8V circuit delivers 1.8V, the 3.3V delivers 3.3V and the 5.0V delivers 5.0V.

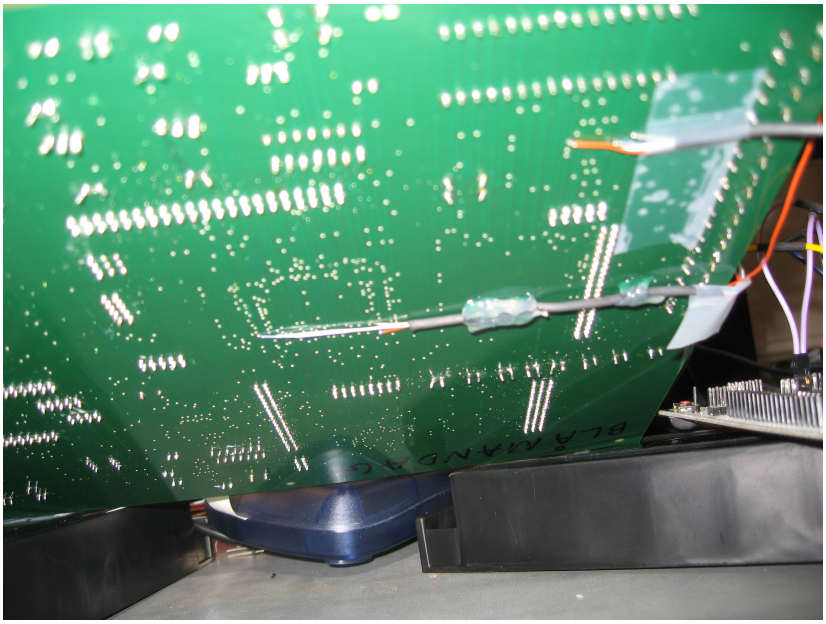
5.1.5 PCB card 1

Figure 5.1 shows what the cards look like. The tables 5.7, 5.8, 5.9, 5.10 lists the test results from card 1.

[1] http://csrc.nist.gov/publications/nistpubs/800-20/800-20.pdf



(a) Top View of Card no. 1



(b) Bottom View of Card no. 1

Figure 5.1: Pictures of Card no. 1

Test	**Expected result**	**Status**
Battery charger	Delivering 10V to the battery	Tested OK
5.0V Power circuit	Delivers 5.0V	Tested OK
3.3V Power circuit	Delivers 3.3V	Tested OK
1.8V Power circuit	Delivers 1.8V	Tested OK
Power consumption without LCD and GSM	0.3A	Tested OK
Power consumption without GSM	0.3A	Tested OK
Power consumption with all things connected	0.3A	Tested OK
Power consumption incoming call	0.6A	Tested OK
Power consumption when an encrypted call is running		N/A

Table 5.7: Power circuit for PCB card 1

Test	**Expected result**	**Status**
Audio codec	Compresses the digital audio	Not working
DA converter	Converts the digital signals to analog	Tested OK in loop-back mode
AD converter	Converts the analog signal to digital	Tested OK in loop-back mode
Audio out amplifier	Amplifies the audio out	Tested OK
Audio in amplifier	Amplifies the audio in	Tested OK

Table 5.8: Audio test for PCB card 1

Test	Expected result	Status
Reset circuit	Delivers 3.3V on RESET_LOW and 0V when RESET is pushed	Tested OK
GSM circuit	The GSM can communicate with the AVR	Tested OK
AVR	All pins connected	Tested OK
FPGA	All pin connected	Tested OK
FPGA programmable	It can be programmed via the external JTAG connector	Tested OK
FPGA PROM programmable	It can be programmed via the external JTAG connector	Tested OK
AVR communication with FPGA	AVR and FPGA can communicate over the memory bus	Tested OK

Table 5.9: Circuits test for PCB card 1

Test	Expected result	Status
FPGA LED's	LED's working	Tested OK
FPGA buttons	The buttons are working	Tested OK
Keyboard	Keyboard connected and communicating via the FPGA	Tested OK
LCD circuit	Working	Tested OK
AVR writing to LCD	The AVR can write text to the LCD	Tested OK

Table 5.10: User interface test for PCB card 1

5.1.6 PCB card 2

Card two died because of an not yet located short circuit between 3.3V and gnd.

5.1.7 PCB card 3

Table 5.11, 5.12, 5.13, 5.14 lists the test results from card 3.

Test	**Expected**	**Result**
Battery charger	Delivering 10V to the battery	Tested OK
5.0V Power supply	Delivers 5.0V	Tested OK
3.3V Power supply	Delivers 3.3V	Tested OK
1.8V Power supply	Delivers 1.8V	Tested OK
Power consumption without LCD and GSM	0.3A	Tested OK
Power consumption without GSM	0.3A	Tested OK
Power consumption with all things connected	0.3A	Tested OK
Power consumption incoming call	0.6A	Tested OK
Power consumption when a encrypted call is running		N/A

Table 5.11: Power circuit for PCB card 3

Test	**Expected result**	**Status**
Audio codec	Compresses the digital audio	Not working
DA converter	Converts the digital signals to analog	Tested OK in loop-back mode
AD converter	Converts the analog signal to digital	Tested OK in loop-back mode
Audio out amplifier	Amplifies the audio out	Tested OK
Audio in amplifier	Amplifies the audio in	Tested OK

Table 5.12: Audio test for PCB card 3

Test	Expected result	Status
Reset circuit	Delivers 3.3V on RESET_LOW and 0V when RESET is pushed	Tested OK
GSM circuit	The GSM can communicate with the AVR	Tested OK
AVR	All pins connected	Tested OK
FPGA	All pin connected	Tested OK
FPGA programmable	It can be programmed via the external JTAG connector	Tested OK
FPGA PROM programmable	It can be programmed via the external JTAG connector	Tested OK
AVR communication with FPGA	AVR and FPGA can communicate over the memory bus	Tested OK

Table 5.13: Circuits test for PCB card 3

Test	Expected result	Status
FPGA LED's	LED's working	Tested OK
FPGA buttons	The buttons are working	Tested OK
Keyboard	Keyboard connected and communicating via the FPGA	Tested OK
LCD circuit	Working	Tested OK
AVR writing to LCD	The AVR can write text to the LCD	Tested OK

Table 5.14: User interface test for PCB card 3

5.2 Software

We tested the software for the AVR microcontroller. Here we tried to determine if the software performed the tasks as it should.

5.2.1 High-level test of main program

We tested that our main program was able to perform the tasks we had implemented. We performed the following tests:

- Start up – We tested that the phone was able to start up, and that we were able to enter the pin code.
- Phone book – We tested that we could open the phone book and show the contacts who were stored on the memory card.
- Send unencrypted SMS – We tested that we were able to send an unencrypted SMS to a number we typed in.
- Send encrypted SMS – We tested that we were able to send an encrypted SMS to a contact in the phone book.
- Receive unencrypted SMS – We tested that we were able to receive an unencrypted SMS by sending a SMS to the unit from a mobile phone.
- Receive encrypted SMS – We tested that we were able to receive an encrypted SMS by sending a SMS from the other unit.

The test results are shown in table 5.15. All the parts of the program we had implemented worked as they should.

Test	**Expected result**	**Status**
Start up	Main menu on display	Tested OK
Phone book	Be able to browse the phone book	Tested OK
Send unencrypted SMS	Unencrypted SMS received on phone	Tested OK
Send encrypted SMS	Scrambled SMS received on phone	Tested OK
Receive unencrypted SMS	Reception and display of SMS sent from phone	Tested OK
Receive encrypted SMS	Reception and display of SMS sent from the other unit	Tested OK

Table 5.15: Test for main program

5.2.2 GSM

The serial interface SocketModem EDGE uses is not RS-232 compliant, so we had to connect the GSM module directly to an Atmel AVR Starter Kit and Development System (STK500) with an Expansion Module for the Atmel STK500 (STK501) expansion card. This was necessary to perform testing and get familiar with the module with minimal work. This way we could use the RS-232 to Universal Asynchronous Receiver/Transmitter (UART)-converter on the STK500 to

send serial data to the AVR which in turn forwarded the data in the UART format to the GSM module. This worked both ways and enabled us to use a client over COM1 to communicate directly with the GSM-module from a PC. See figure 5.2 for a principal diagram of this test setup.

We have specified in the requirement specification that our unit must be able to communicate over a standard mobile network, GSM or UMTS. As mentioned earlier, we chose the GSM mobile network. Therefore, we have to test any possible functionality that we might have a use for. Examples of such functionality is calling, GPRS, SMS and so on.

To make sure the needed AT commands worked as expected, we had to test each one on this test bench. To achieve this we tested the AT commands as they were listed in the GSM module documentation. When we tested the AT commands we defined certain tasks which involved one or several AT commands. Then we easily could see if they worked, and that the output was as expected. The tasked we defined is listed in table 5.16.

Figure 5.2: Test setup for GSM-module

Test	**Expected response**	Status
PIN code authentication	OK	Passed
GPRS connection profile setup	OK	Passed
GPRS service profile setup	OK	Passed
Activate GPRS service	OK	Passed
Check GPRS connection status	Status	Passed
Closing a GPRS connection	OK	Passed
Read/Write data from connection	Output/OK	Passed
Set SMS format to text	OK	Passed
Set preferred SMS storage	+CPMS: 2,40,0,15,2,40 OK	Passed
Send an SMS	> text to be sent <CTRL-Z> <ESC>	Passed
List SMS stored in a category	+CMGL: (SMS header) SMS data OK	Passed
List SMS categories	+CMGL: (list of supported categories) OK	Passed
Read a single SMS	+CMGL: (SMS header) SMS data OK	Passed
Delete a SMS	OK	Passed
Turn on new SMS notification	OK	Passed
SMS notifications appears when new SMS is delivered	+CMTI: <memory>,<index>	Passed
Check signal status	+CSQ: <signal strength>, <channel bit error rate> OK	Passed

Table 5.16: GSM-module tests

Chapter 6

Tools

This chapter describes the software and hardware tools we have used for designing, building and testing the devices, as well as for report writing and version control.

6.1 PCB Design

For the PCB design process we used the suit by Mentor Graphics, mainly Design Capture and PCB Expedition. The former is a tool for making the schematics with paths between the components and attaching their appropriate footprints. The resistors, capacitors and diodes are inserted with their appropriate values, so these schematics can be used in the manual assembly of the physical cards later on, as an assembly guide.

The latter, PCB Expedition, is a tool that imports the schematics from Design Capture. It is used to generate the correct gerber files needed. It has a powerful auto router that ease the job of setting up the connections between the parts on the actual PCB. We used this tool set for a 4-layer design; VCC, GND and to layers for the connections. The VCC layer included our three different operating voltages at 5.0, 3.3 and 1.8 volts. In PCB Expedition we set up the actual placement of the components, making sure that parts belonging to each other were grouped together and marked with the appropriate subsystem name (audio, codec, FPGA etc.). The use of this tool resulted in the gerber files that were to be sent to ElPrint in Oslo for the manufacturing of the boards. In addition a drilling file was generated so the correct holes would be drilled.

6.2 VHDL

To make the FPGA work we needed to program it. This is done by giving it a bit file which defines the FPGAs LUTs (for more, see section 2.3) and how the signals inside the FPGA is routed. By loading a PROM with the bit file we can prevent the FPGA to be programmed from a cable each time we reboot. We therefor make a mcs file for the PROM which upload its program to the FPGA each reboot. The generation of this file is based on the bit file produced for the FPGA.

To be able to make a bit file for the FPGA we need write a program for the FPGA. This is done by using VHDL (for more info see section 2.4). For writing VHDL we have used several tools. From vim and emacs to Xilinx ISE. It does not matter which text editor we write the VHDL in, but we have to do some more than just writing text files. To make a bit file from the

VHDL we used Xilinx development tools. This tools are bundle together in a package known as the Xilinx ISE.

The programs in the Xilinx ISE took care of all from synthezise the vhdl to mapping and routing all the signals in the FPGA.

Besides the Xilinx ISE we needed a programming device to actually transfer the program we had produced to the target device. For this we used a parallel port JTAG cable which was allready available for us at the computer labratory. To be more precise we used the Xilinx Parallell Cable IV programming cable.

6.3 AVR programming

This section describes the software and hardware tools used for developing and uploading software for the AVR.

6.3.1 Software tools

To compile C code for the AVR we used AVR-GCC and for uploading the binary files, single step and debug we used AVRdude and AVRStudio. AVR-GCC is a freeware compiler and assembler made available by the GNU project. AVRStudio is a free[1] Integrated Development Environment (IDE) for writing and debugging software for AVR applications and supports a wide range of Atmel's hardware tools. It can also auto-detect and utilize an AVR-GCC installation to ease the process of compiling. The downside is that it only runs in 9x/NT/2000/XP[2] environments. AVRdude is a freeware uploader and downloader for AVRs and was used when uploading from a UNIX environment.

6.3.2 Hardware tools

To upload the compiled software, set fuses in the AVR, single step and debug, we used Atmel's own programming tool called JTAGICE MKII and in testing phases we also used Atmel's starter kit STK500 with an extension named STK501 for programming AVRs in TQFP-64 packaging.

6.4 LaTeX

We have chosen to use LaTeX for the report primarily because it is well suited, considering the way we are editing the document. Other formats that could have been used are Microsoft Word or OpenOffice documents. These formats are more layout focused, and not particulary well suited when the document is edited (often simultaneously) by different persons. There are surely other feasible formats, but since many of the persons involved already knew LaTeX it was a natural choice.

6.5 SVN

To keep record of changes and be able go back in history of changes when necessary, we chose to use a version control system, namely Subversion (SVN). SVN is a system that allows users

[1] Registration form submission required to download
[2] Windows 95, Windows98, Windows NT, Windows 2000 or Windows XP

to keep track of changes to source files, and other types of file formats, but is mainly suited for text based data. This is because SVN cannot merge binary files without any knowledge of the binary format, without considerable effort. This way, anyone on the team could work separately on each file at any time, and when feeling ready to commit, they could share the result with others. If any conflict arises while commiting, they could work it out by merging the contents. That is, if someone else had edited the same file.

6.6 Laboratory tools

A lab was completely reserved for the course and there we found most electrical equipment and hardware tools needed for assembling the cards. The most essential of these were soldering irons of different types, solder, wires, different types of wire cutters, a microscope, oscilloscopes, a signal generator and various generic components like capacitors and resistors.

Chapter 7

Discussion

In this chapter we discuss various aspects of the Dulkóðuð leyndartut prototype and the process of designing and building it, as well as the security of the product.

This chapter begins with section 7.1, where we discuss some topics about the security of our devices. Section 7.2 mentions some choices we have made which turned out to be less than optimal and suggests how we could have avoided the problems caused by these. In section 7.3 we write about our problems with the audio codec chip. Section 7.4 comments on the way we worked together in the group. Section 7.5 presents the economical results of the project and gives some ideas as to the commercial potential of a future development of our prototype.

7.1 Security

Triple DES is a well-known cipher that has been extensively analyzed, and no feasible attacks have been found (this applies to AES, which is what we originally intended to use, as well). In cipher block chaining mode, it is safe to use for messages that are longer than one block. However, there is one weakness in our way of using the cipher: we always use the same initialization vector (namely one containing only zero bits). This means that identical messages will produce the same ciphertexts. It could have been solved quite easily, though (we just ran out of time at the end), by always prefixing a message with a block containing random data, which the receiver will just discard. This is equivalent to using a random initialization vector and sending only zeroes in the first plaintext block.

Our product does not provide message integrity or authenticity[1] - that is, messages can be tampered with and forged. This should not be much of a problem in practice, since the attacker (who does not have the key) would not be able to produce data that are meaningful when decrypted. Modifications of an encrypted message will cause corruption of the corresponding plaintext blocks plus the block that follows the tampered blocks (after that, the cipher block chaining mode will recover), so tampering should be evident to the recipient. However, our product is vulnerable to replay attacks. If an attacker has intercepted a message, he can resend it at a later time, and the recipient will not be able to tell whether this is a duplicate or a new (but identical) message. Again, with slightly more time on our hands, this could have been solved as well, by including a timestamp or sequence number inside the encrypted message.

[1] The task specification suggested that we might implement a public key cryptosystem to provide message signing, but we did not have the time to do this.

Note that we have not taken into account the problem of key distribution and management, which was not part of the task.

7.2 Things we wish we had done

During the development we found that our design decisions in the earlier phases were not optimal. This section summarizes the decisions that we now regret, and tells what we would done different if we had seen these potential problems earlier.

7.2.1 Splitting the VCC plane

If we had split the VCC plane into several independent sections, it would have been easier to isolate faults. We found it problematic to isolate short circuits due to the large voltage planes. Had we isolated each voltage plane into several, for example four separate planes for each voltage, the job of finding such short circuits had been much easier.

7.2.2 In-System-Programming Connector

While planning the top-level system design we decided to move the display to the same port on the AVR as the JTAG-interface. The argument for this was that we thought that we would not need to use both simultaneously, and that switching between the two would be easy. It turned out that the display would not function when the JTAGEN fuse in the AVR were programmed and when this fuse was unprogrammed it could not be re-programmed because it required the JTAG interface to already be enabled.

7.2.3 More GND

More GND pins on different places on the board would have made testing easier. In addition the GND layer should have been divided into a GND layer for the digital part, and one for the analog part. This is a well known good design principle to follow if problems arise with noise on the analog part. We did not implement this in our PCB due to our inexperience. In retrospect this is something we should have done. This could have given better sound quality from the audio part. Electrical noise from the noisy parts of the circuit can get to the parts of the circuit which should be free of noise, as mentioned the audio part, resulting in terrible audio quality in a A/D converter.

7.3 Audio codec chip

We had problems with making the AMBE-2000 audio codec chip work. The audio codec chip sends frames containing 24 words, with each word consisting of 16 bits.

To receive data from the AMBE-2000, we have to give it a clock. The clock should have a speed up to 2 MHz. We decided to use 1 MHz. To request that the AMBE-2000 transmits a single word, we have to transmit a “strobe” signal. The AMBE-2000 transmits the word after the falling edge of this strobe signal.

We managed to implement a simple test code for creating the clock and sending the strobe signal. We also created code which should read the data the AMBE-2000 transmitted. However, we were unable to make the AMBE-2000 transmit a response. When we were unable to receive a

signal, we attached an oscilloscope to the channel. We were able to verify that the clock had the correct frequency, and the strobe signal was transmitted correctly. We did not register anything on the data channel from the AMBE-2000. We also tried to connect the logic analyzer, but got the same result.

We considered connecting the AD/DA converter directly to the FPGA. Because of the data rate to/from the AD/DA converter, we would have needed to perform some simple compression of the datastream in the FPGA. However, by this time, we had limited time remaining, and we decided to focus on other parts of our project.

7.4 Group dynamics

In this section we describe how we organized the group and how we handled some of the more difficult decisions.

7.4.1 Leadership

One of our main problems throughout the project progress was the lack of a natural leader. Because none of us had any strong wish of taking the leadership, there has not been a formal leader. Due to this, no single person had a full picture of the project's state at any given time. In retrospect, we realize that it would have been beneficial to have one person responsible for coordinating the work.

7.4.2 Groups

Initially we divided the group into natural subgroups focusing on, respectively: PCB design (three persons), VHDL code for FPGA (two persons), microcontroller programming (two persons), research and testing regarding GSM (two persons) and audio research (one person). During the project these groups become increasingly diffuse. Some groups needed assistance from time to time and hired in some help from the others to finish their tasks. Additionaly, the PCB group had finished its task when the PCB was delivered to production; the persons in this group were then moved to other groups. Some tasks, such as soldering, were not assigned to any group and were instead performed by several persons from different groups.

7.4.3 Difficult decisions

Some difficult decisions had to be made. The decision with most controversy was about how we should implement the mobile platform.

7.4.3.1 Mobile platform

As mentioned in section 3.3.4 we initially decided to use the same module as the earlier project Priori (GM862 from RoundSolutions). This was made through a democratic decision, and the first voting gave a majority for GM862. A module from Siemens (MC75) was fronted as an opponent, but this motion was voted down mainly because of a more difficult hardware interface (80 pin connector and the sim card reader was excluded). Not all members of the group were happy with this choice and some continued investigating for other available modules.

After a new round of research a new module, the SocketModem Egde was found. A new voting was held and we decided to use this module instead.

7.4.3.2 Audio design

Deciding which units should be used and how the audio module should be designed was rather difficult. None of the group members had implemented anything similar, and we only had a vague understanding on how this could be put together.

Searching for possible modules was time consuming and difficult, partly because of the lack of knowledge about the terminology. When we found the webpage for the module we ended up using, we tried to send an e-mail to the address they had listed on their webpage but they has not yet given any response to this. When we realized that an answer never would come, we tried to call them but we got forwarded to a Bob, but found an dead end in his voice mail. A new round of searching was done and we found his e-mail address and managed to place an order at last.

Even though the chip we found probably was the best choice between the options we found, there is a high probability that there exists a chip more fit for our needs that we could not find.

7.5 Research & Development

We have developed a prototype for secure communication via the cell phone network. To develop this prototype we had 23000 NOK available for components, PCB print etc.

The prototype has been developed by ten workers, each working about 250 hours on this project. If this development is done as an investment from an company the prototype would cost 750 000 NOK in salaries (assuming that the salary from the company is 300 NOK).

7.5.1 Budget

As we had a upper limit at 23000 NOK on our expenses, we were in the initial planning a bit worried that we would break that limit. We therefore used some time on considering cheaper parts, and what parts we actually needed. We also used components like LCD, keyboard and so on from the lab.

After the prototype was built, the component expenses had accumulated to 20000 NOK. For a more detailed account overview, see D.1

7.5.2 Making the final product

Even if we have a working prototype, further improvements are needed before we have a final production line model. And we must also find out what kind of market this product appeals to.

Production line model If we are going to mass produce this unit as a successful secure mobile phone we would have to make the unit much smaller and lighter than it is now. The user interface must also be much more user friendly, as this product may be used by people with various technical skills. And it must of course support both encrypted SMS/MMS and speech. Support for both GSM and UMTS must also be implemented together with color monitor, MMS, WAP, bluetooth and so on. And we have to use a much stronger encryption algorithm than 3DES which we used in our prototype. In other words, the production model must have the same functionality and physical dimensions as a regular cellular phone in addition to cryptology abilities.

Target groups This product may be very useful for military purposes in high-security environments. Other target groups may be in big corporations, mainly the top executives that need to communicate in a secure way. Unfortunately, criminals may also draw benefit of this product because they will be able to communicate without risk of being phone tapped by the authorities.

Chapter 8

Conclusion

The main goals of this projects were to gain basic knowledge about – and experience in – practical hardware design, and to build two working prototype units of a mobile telephone with strong symmetric encryption of the sound data.

With most of us having little or no prior experience in hardware design, we may say that we have achieved the first of these goals. Through the practical work we have gained experience in areas such as PCB design, writing VHDL, low-level programming of a microcontroller, and choosing appropriate electronic components for a design. We have performed much of the implementation and testing of a complete system with several components acting together.

The goal of creating prototypes was, however, not completely reached, mainly due to unforeseen problems with the audio codec chip (the AMBE-2000). Without the AMBE speech codec chip we are unable to compress the audio data, and we cannot transmit speech. We considered trying to connect the AD/DA converter directly to the FPGA, but with little time remaining, we decided to focus our efforts elsewhere.

Virtually all of the other components work as they should. These include the battery circuit, the three power circuits, the reset circuit, the GSM modem, the LCD circuit, the microcontroller, the FPGA, the external memory, the buttons and LEDs, the amplifier and the AD/DA (even though the latter creates a lot of noise). The microcontroller and the FPGA have been successfully programmed to make all the components, except the AMBE-2000, work together. The microcontroller has a user interface running on the LCD that is controlled by the keyboard. We can send 3DES encrypted SMS messages to other mobile phones, and such messages can only be decrypted by someone in possession of the key with which the message was encrypted. Since we have two functioning devices, they can communicate transparently without the user noticing the encryption, since decryption of incoming messages happens automatically on the Dulkóðuð leyndartut.

If time had allowed, a second revision of the phone could have been produced, hopefully correcting the audio noise problem and making the AMBE codec chip work. We have experimented with setting up EDGE connections between the phone and a computer, which suggests that it should be possible to set up connections between two phones as well. Thus, we are confident that the initial goal of transferring encrypted sound is within reach.

Chapter 9

Acknowledgements

9.1 GSM

For the GSM part of the project there were several actors who contributed.

Netcom AS We want to thank Netcom AS v/ Gina Langnes Burœy Olsen, Product Manager for Smarttalk, for giving us access to two GSM subscriptions together with Connect Premium free of charge for our project. And we also want to thank Netcom generally for giving us information about GPRS connections (certain GPRS access point).

Magnus Jahre He gave us valuble information which made us able to avoid the problems that Priori had with their GSM module, the Round Solutions GM862.

Acte AS A big thank to Leif Thorset at Acte AS for providing more than a sufficient amount of documentation which made it much easier for us to work with the GSM module.

9.2 Course staff

The course staff has been a good support to the project, in form of guidance through the project. We have also been able to deliver several drafts of our report, and they have come with constructive critics helping us to refine our final report for evaluation.

9.3 Atmel

Thanks to Atmel for helping us solder two integrated circuits – the FPGA and the audio codec chip.

Chapter 10

Abbreviations

3DES	Triple DES
AD	Analog-to-Digital
AES	Advanced Encryption Standard
BTS	Base Transceiver Station
BSC	Base Station Controller
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
DA	Digital-to-Analog
DC	Direct Current
DES	Data Encryption Standard
DIP	Dual In-line Package
DSP	Digital Signal Processor
EDGE	Enhanced Data rates for GSM Evolution
EFSL	The Embedded Filesystems Library
E-GPRS	Enhanced General Packet Radio Service
FAT	File Allocation Table
FPGA	Field Programmable Gate Array
FSVS	Future Secure Voice System
GCC	GNU Compiler Collection
GPL	GNU General Public License
GPRS	General Packet Radio Service

GPS	Global Position System
GSM	Global System for Mobile communications
HDK	Hardware Development Kit
HSDPA	High-Speed Downlink Packet Access
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IP	Internet Protocol
ISDN	Integrated Services Digital Network
IDE	Integrated Development Environment
JTAG	Joint Test Action Group
kbps	kilobit per second
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
LUT	Lookup Table
MIPS	Million Instructions Per Second
MMC	Multi Media Card
MS	Mobile Station
MSC	Mobile-services Switching Centre
NiMH	Nickel-Metal Hydride battery
NTNU	The Norwegian University of Science and Technology
NSA	National Security Agency
op-amp	operational amplifier
PC	Personal Computer
PCB	Printed Circuit Board
PCM	Pulse-Code Modulation
PCU	Packet Control Unit
PPC	PowerPC

PROM Programmable Read-Only Memory

PSU Power Supply Unit

RAM Random Access Memory

SD Secure Digital

SIM Subscriber Identity Module

SGPN Serving GPRS Support Node

SVN Subversion

SMS Short Message Service

SRAM Static Random Access Memory

SPI Serial Peripheral Interface Bus

STK500 Atmel AVR Starter Kit and Development System

STK501 Expansion Module for the Atmel STK500

STU-III Secure Telephone Unit, Third generation

TCP Transmission Control Protocol

TDMA Time Division Multiple Access

UART Universal Asynchronous Receiver/Transmitter

UMTS Universal Mobile Telecommunications System

USB Universal Serial Bus

VHSIC Very-High-Speed Integrated Circuits

VHDL VHSIC Hardware Description Language

WLAN Wireless Local Area Network

Bibliography

[1] Acte AS. http://www.acte.no.

[2] Speech compression. *http://www.data-compression.com/speech.html*.

[3] Siemens AG. MC75 product brief. *http://pia.khe.siemens.com/efiles/wireless/datasheets/MC75_Datasheet.pdf*, 2006.

[4] Digital Voice Systems, inc. DVSINC VC-55. http://dvsinc.com/products/vc-55-pr.htm.

[5] GSMK. Encryption. http://www.cryptophone.com/background/encryption/index.html.

[6] Jan Audestad, Teacher in TTM4105 Access and transport networks. Land mobile systems. *http://www.item.ntnu.no/fag/ttm4105/2007/8_Celluar.pdf*, 2007.

[7] Atmel, Inc. ATmega128L Datasheet. *http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf*, 2007.

[8] Brilliance Semiconductor, inc. BS62LV1027 - very low power/voltage CMOS SRAM 128k x 8-bit. *http://www.brilliancesemi.com/product/BS62LV1027.pdf*, 2004.

[9] Digital Voice Systems, Inc. Voice coding overview. *http://dvsinc.com/papers/vc_over.htm*.

[10] Digital Voice Systems, Inc. AMBE-2000™ Vocoder Chip User's Manual. *http://www.dvsinc.com/manuals/AMBE-2000_manual.pdf*, 2007.

[11] Ben Johnsen. *Cryptography - the secret writing*. Tapir Akademisk Forlag, Trondheim.

[12] Gentre Graham, David Leifker. VHDL AES128 Encryption/Decryption, 2005. http://cegt201.bradley.edu/projects/proj2005/aes128/.

[13] Coretex Systems, LLC. 3DES (Triple DES) / DES (VHDL), 2006. http://www.opencores.org/projects.cgi/web/3des_vhdl/overview.

[14] Spyros Ninos. twofish 128/192/256, 2006. http://www.opencores.org/projects.cgi/web/twofish/overview.

[15] National Institute of Standards and Technology. Data encryption standard (des). *http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf*, 1999.

[16] National Institute of Standards and Technology. Announcing the advanced encryption standard (aes). *http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf*, 2001.

[17] Maxim Integrated Products. DS1233 - 3.3v EconoReset. *http://datasheets.maxim-ic.com/en/ds/DS1233.pdf*, 2001.

[18] Maxim Integrated Products. MAX712 - NiMH Battery Fast-Charge Controller. *http://datasheets.maxim-ic.com/en/ds/MAX712-MAX713.pdf*, 2002.

[19] Roland Riegel. MMC/SD card reader example application. *http://www.roland-riegel.de/sd-reader/*.

[20] Hemanth Satyanarayana. AES128, 2004. http://www.opencores.org/projects.cgi/web/aes_crypto_core/overview.

[21] ASICS / Rudolf Usselmann. AES (Rijndael) IP Core, 2002. http://www.opencores.org/projects.cgi/web/aes_core/overview.

[22] Wikipedia. Advanced Encryption Standard - Wikipedia, The Free Encyclopedia, 2007. http://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard\&oldid=172668376.

[23] Wikipedia. Data Encryption Standard - Wikipedia, The Free Encyclopedia, 2007. http://en.wikipedia.org/w/index.php?title=Data_Encryption_Standard&oldid=172344919.

[24] Wikipedia. STU-III - Wikipedia, The Free Encyclopedia, 2007. http://en.wikipedia.org/w/index.php?title=STU-III&oldid=153712820.

[25] Wikipedia. VHDL - Wikipedia, The Free Encyclopedia, 2007. http://en.wikipedia.org/w/index.php?title=VHDL&oldid=171356985.

[26] Thomas Wilburn. The audiofile: Analog-to-digital conversion. *http://arstechnica.com/articles/paedia/audiofile-analog-to-digital-conversion.ars*, 2007.

[27] The xiph open source community. Speex: A Free Codec For Free Speech. *http://www.speex.org*.

Appendix A

Datasheets/Specifications

A.1 GSM

A.1.1 Siemens MC75

- Frequency bands Quad band: GSM 850/900/1800/1900MHz GSM class Small MS
- Power supply 3.2V to 4.3V
- Physical Dimensions:
  - 33.9mm x 44.6mm x max. 3.5mm
  - Weight: approx. 7.5g
  - RoHS compliant
- EGPRS (EDGE) class 10 and GPRS class 12
- PPP-stack for GPRS data transfer

A.1.1.1 Feature Implementation

- SMS
- Fax Group 3; Class 1

A.1.1.2 Audio Speech codecs

- Support for Half rate HR, Full rate FR, Enhanced full rate EFR, Adaptive Multi Rate AMR
- speakerphone operation, eho cancellation, noise suppression, DTMF, 7 ringing tones

A.1.1.3 Software

- AT commands AT-Hayes GSM 07.05 and 07.07, Siemens
- AT commands for RIL compatibility (NDIS/RIL)

- AT command compatible for module management
- IPv4 and IPv6 support

Remote SIM Access
Firmware update via UART/USB for Windows

A.1.1.4 Interfaces

2 serial interfaces
ASC0

- 8-wire modem interface with status and control lines, unbalanced, asynchronous
- Fixed bit rates: 300bps to 460,800bps
- Autobauding: 1,200bps to 460,800bps
- Supports RTS0/CTS0 hardware handshake and software
- ON/XOFF flow control.
- Multiplex ability according to GSM 07.10 Multiplexer Protocol.
- MC75 Hardware Interface Description
- ASC1
- 4-wire, unbalanced asynchronous interface
- Fixed bit rates: 300bps to 460,800bps
- Supports RTS1/CTS1 hardware handshake and software XON/XOFF flow control

USB 2.0 support
I2C
I2C bus for 7-bit addressing and transmission rates up to 400kbps. Programmable with AT$\hat{S}$SPI command.
SD card interface
Interface for SD memory card or multimedia card

A.1.1.5 Audio

- 2 analog interfaces
- 1 digital interface (PCM)

A.1.1.6 SIM interface

Supported SIM cards: 3V, 1.8V

A.1.1.7 Antenna

50Ohms. External antenna can be connected via antenna connector or solderable pad.

A.1.1.8 Module interface

80-pin board-to-board connector Power on/off, Reset

A.1.1.9 Power on/off

- Switch-on by hardware pin IGT
- Switch-off by AT command
- Automatic switch-off in case of critical temperature and voltage conditions.

A.1.1.10 Reset

- Orderly shutdown and reset by AT command
- Emergency reset by hardware pins EMERG_RST and IGT.

A.1.1.11 Special features

- Charging
  Supports management of rechargeable Lithium Ion and Lithium Polymer batteries
- Real time clock
  Timer functions via AT commands
- Phonebook
  SIM and phone

A.1.1.12 Evaluation kit

- DSB75 Evaluation Board

A.1.2 Round Solutions GM862

A.1.2.1 Product Features

- Dual-band 900/1800 MHz
- Management by AT commands
- Supply Voltage: 3.4V - 4.2V, nominal: 3.8V
- Power Consumption: idle mode <3.5 mA, dedicated mode 250 mA
- Dimensions (mm): 6 x 43.9 x 43.9
- Temp range: -20 to +70 (operational)

A.1.2.2 Interfaces

- 50 pin industrial connector: PSU, serial bidirectional bus, 7 general purpose I/O ports, Internal / External SIM 3V, analog audio
- V.24 serial link, baud rate from 300 - 115200 bps, autobauding from 1200-57600 bps
- 50 Ohm Antenna connector

A.1.2.3 Audio

- Telephony, Emergency calls, Half/Full/Enhanced Full rates
- Superior echo cancellation & Noise reduction
- DTMF
- Handset and handsfree operations

A.1.2.4 Misc

- SMS-support
- Circuit-Switched Data up to 14.4 Kbps
- GPRS class 8/10
- EASY-GPRS AT command extension

A.1.3 Multitech SocketModem EDGE (based on Siemens MC75)

- EDGE (E-GPRS) Class 10
- GPRS Class 12
- Quad-band GSM 850/900/1800/1900 MHz
- Packet data rates up to 240K bps (coding scheme, MCS-9, LLC layer, 4 time slots)
- Embedded TCP/IP stack supports TCP, UDP, DNS, FTP, SMTP, POP3, HTTP
- Circuit-switched data up to 14.4K bps non-transparent mode
- Supports Short Message Service such as text and PDU mode, point-to-point (MT/MO) and cell broadcast
- MMCX antenna connector
- SIM card holder
- Serial interface supporting DTE speeds to 460K bps
- AT command compatible
- FCC, PTCRB and R&TTE certified

- Voice features include Half rate (HR), Full rate (FR), Enhanced full rate (EFR), Adaptive multi rate (AMR), as well as hands free echo cancellation, and noise reduction

A.1.3.1 CAP-XX GS203F specifications

This is the corresponding capacitor that we use together with the Multitech SocketModem EDGE.

- 0.2F capacitance
- 50 mΩESR
- Dimensions (mm): 2.15 x 39.5 x 17.5
- Operating temperature: -40°C to +75°C, nominal +25°C

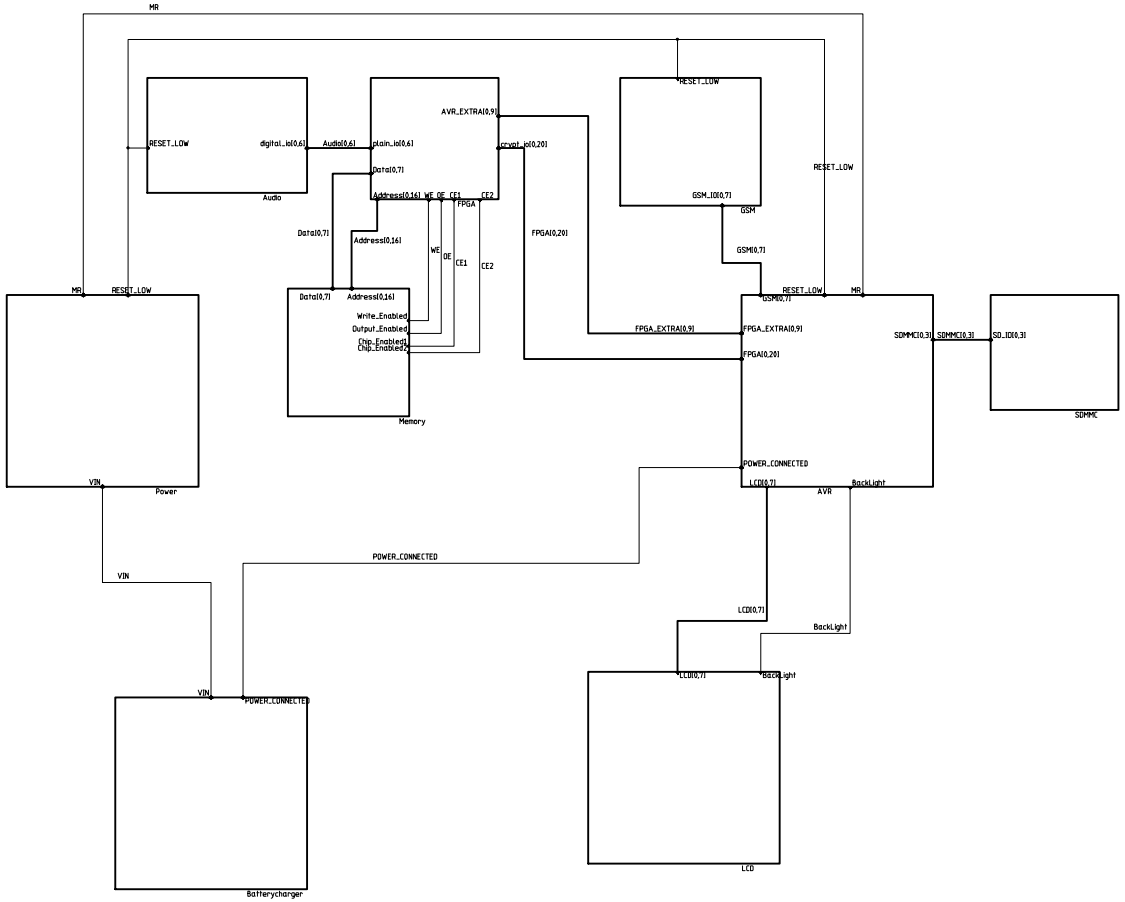
Appendix B

Schematics

DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED



CONTRACT NO.		
APPROVALS	DATE	TOP LEVEL
DRAWN		
CHECKED		
ISSUED		

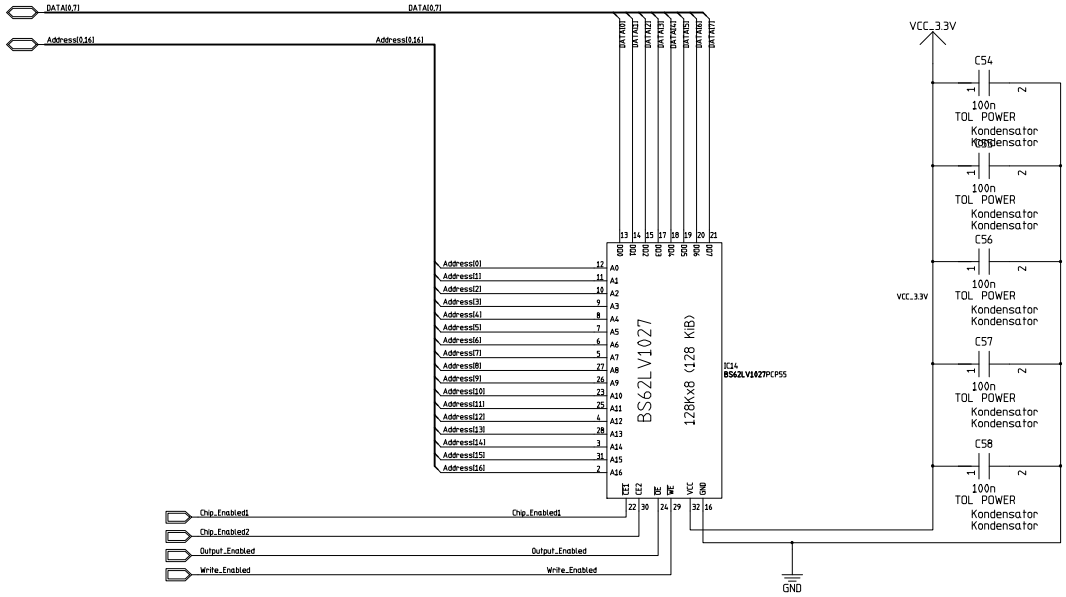
SIZE	FSCM NO.	DWG. NO.	REV
D			
SCALE		SHEET	

HxBND; Vx4LA

DWG. NO.	SH.	REV

REVISIONS

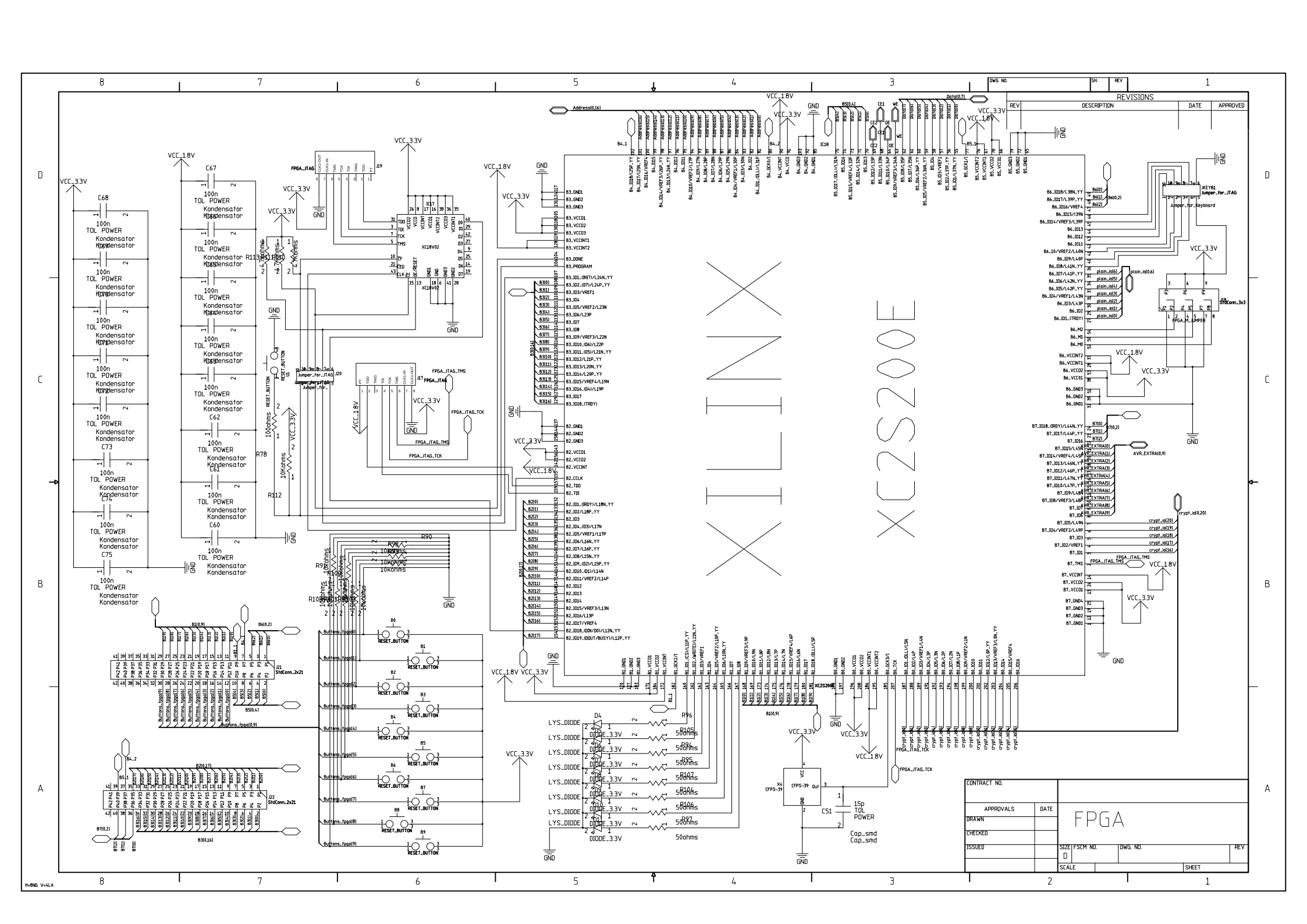
REV	DESCRIPTION	DATE	APPROVED

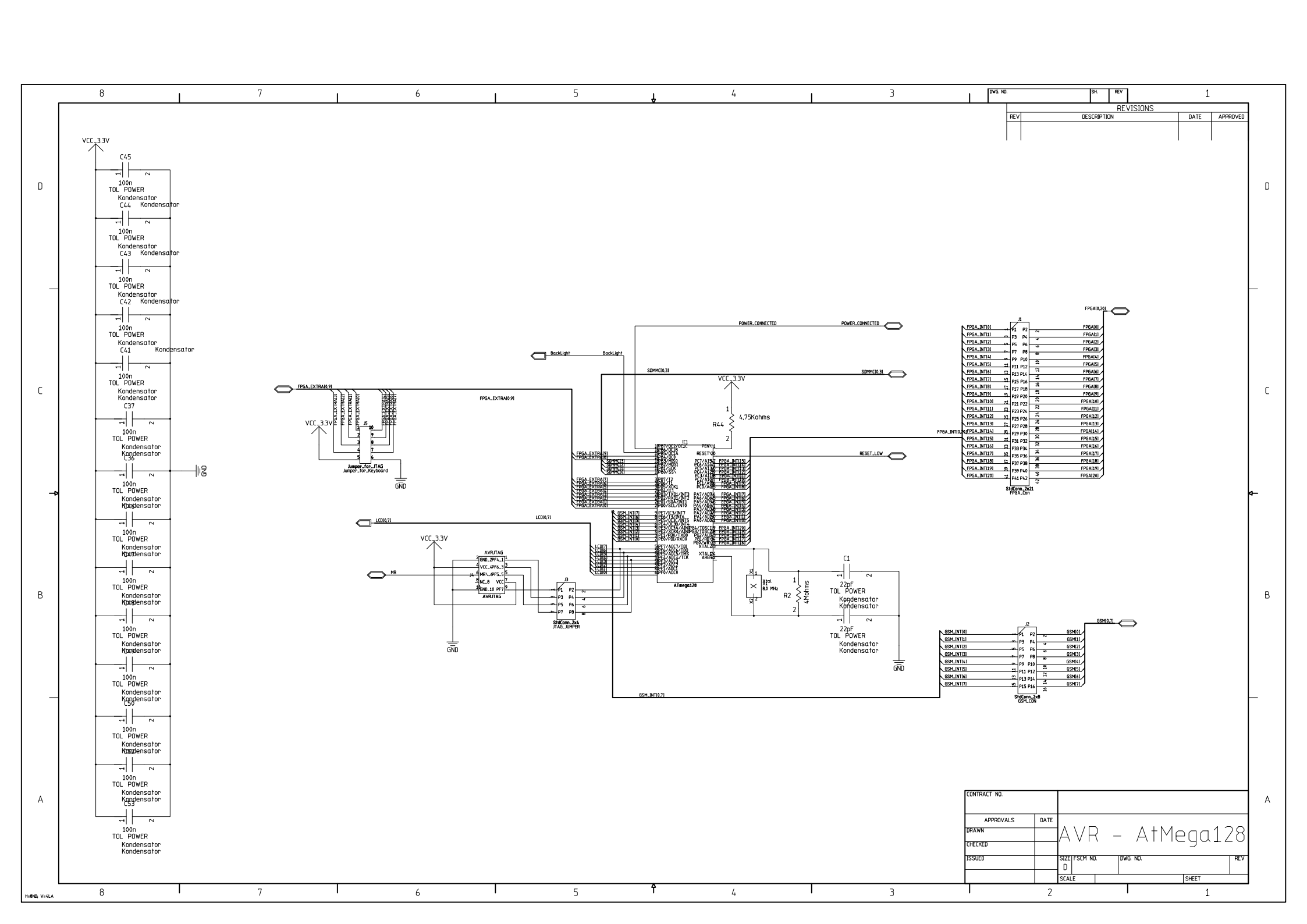


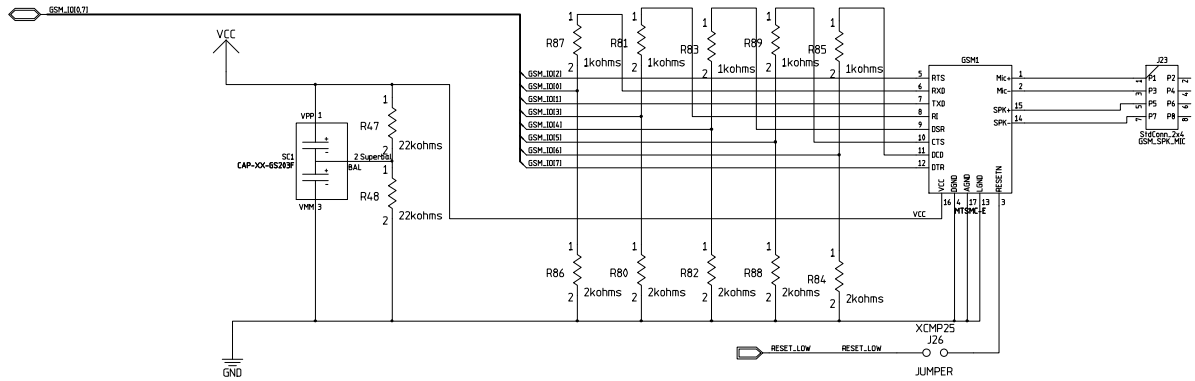
CONTRACT NO.		
APPROVALS	DATE	MEMORY
DRAWN		
CHECKED		
ISSUED		

SIZE	FSCM NO.	DWG. NO.	REV
D			
SCALE		SHEET	

H:BND; V:4LA


XC2S200E
XILINX
REVISIONS
REV
DESCRIPTION
DATE
APPROVED
DWG NO.
SH.
REV
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
FPGA
SIZE
FSCM NO.
DWG. NO.
REV
D
SCALE
SHEET
Address[0.16]
Data[0.7]
FPGA_JTAG
FPGA_JTAG_TMS
FPGA_JTAG_TCK
Jumper_for_JTAG
Jumper_for_Keyboard
FPGA_M_JUMPER
AVR_EXTRA[0.9]
Buttons_fpga[0.9]
LYS_DIODE
DIODE_3.3V
RESET_BUTTON
XC18V02
CFPS-39
StdConn_2x21
StdConn_3x3
Cap_smd
100n TOL POWER Kondensator
10kohms
4.7kohms
50ohms
15p TOL POWER


DWG. NO.
SH.
REV
REVISIONS
REV
DESCRIPTION
DATE
APPROVED
VCC_3.3V
C45
100n
TOL POWER
Kondensator
C44 Kondensator
C43 Kondensator
C42 Kondensator
C41 Kondensator
C37
C36
C35
C34
C33
GND
FPGA_EXTRA[0,9]
Jumper_for_JTAG
Jumper_for_Keyboard
BackLight
SDMMC[0,3]
POWER_CONNECTED
RESET_LOW
R44
4,75Kohms
LCD[0,7]
AVRJTAG
MR
J3
StdConn_2x4
JTAG_JUMPER
ATmega128
X1
Xtal
8.0 MHz
R2
4Mohms
C1
22pF
TOL POWER
Kondensator
J1
FPGA_INT[0,20]
StdConn_2x21
FPGA_CON
J2
GSM_INT[0,7]
StdConn_2x8
GSM_CON
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
AVR - AtMega128
SIZE
FSCM NO.
DWG. NO.
REV
D
SCALE
SHEET



DWG. NO.	SH.	REV

REVISIONS

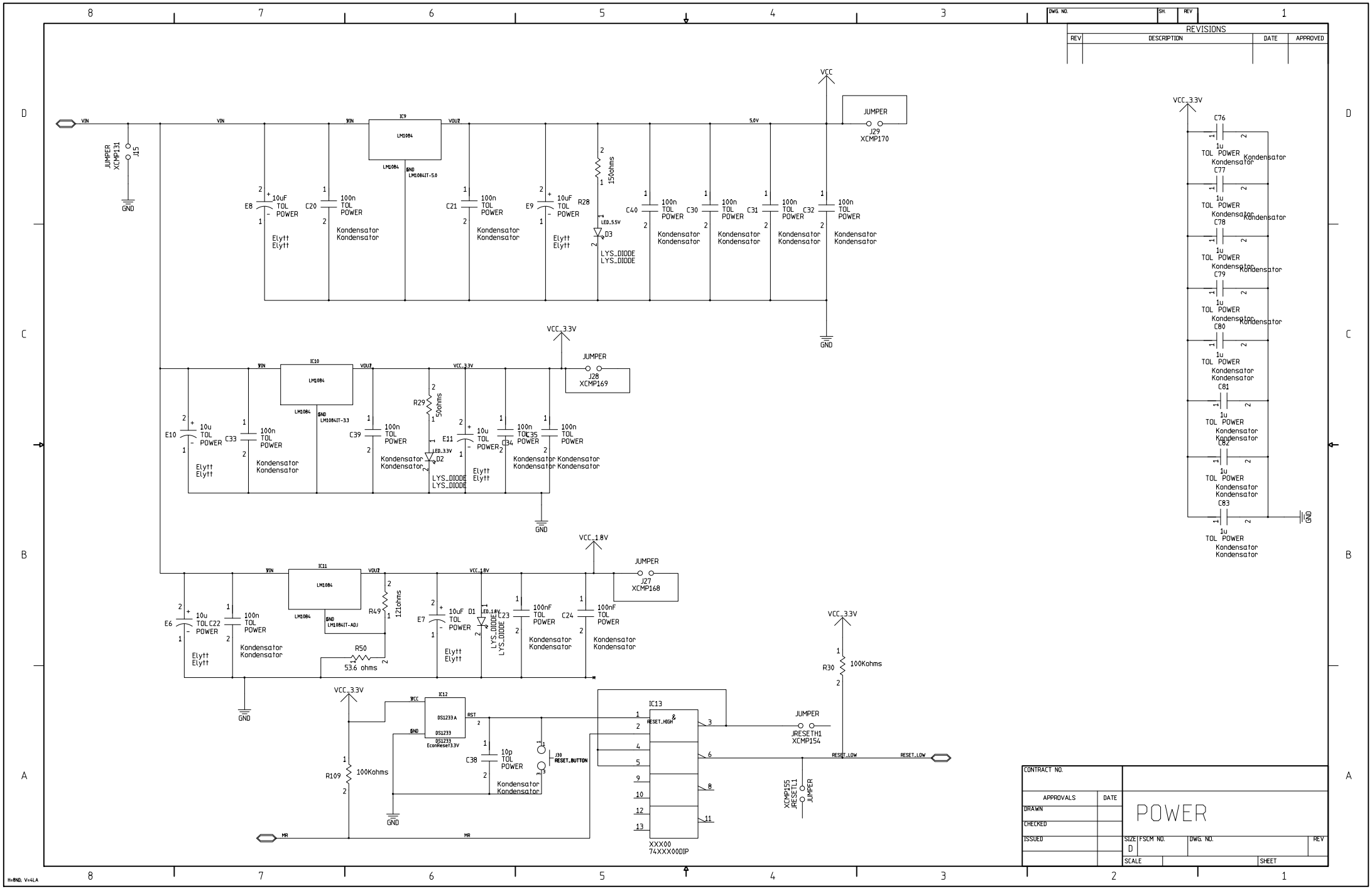
REV	DESCRIPTION	DATE	APPROVED

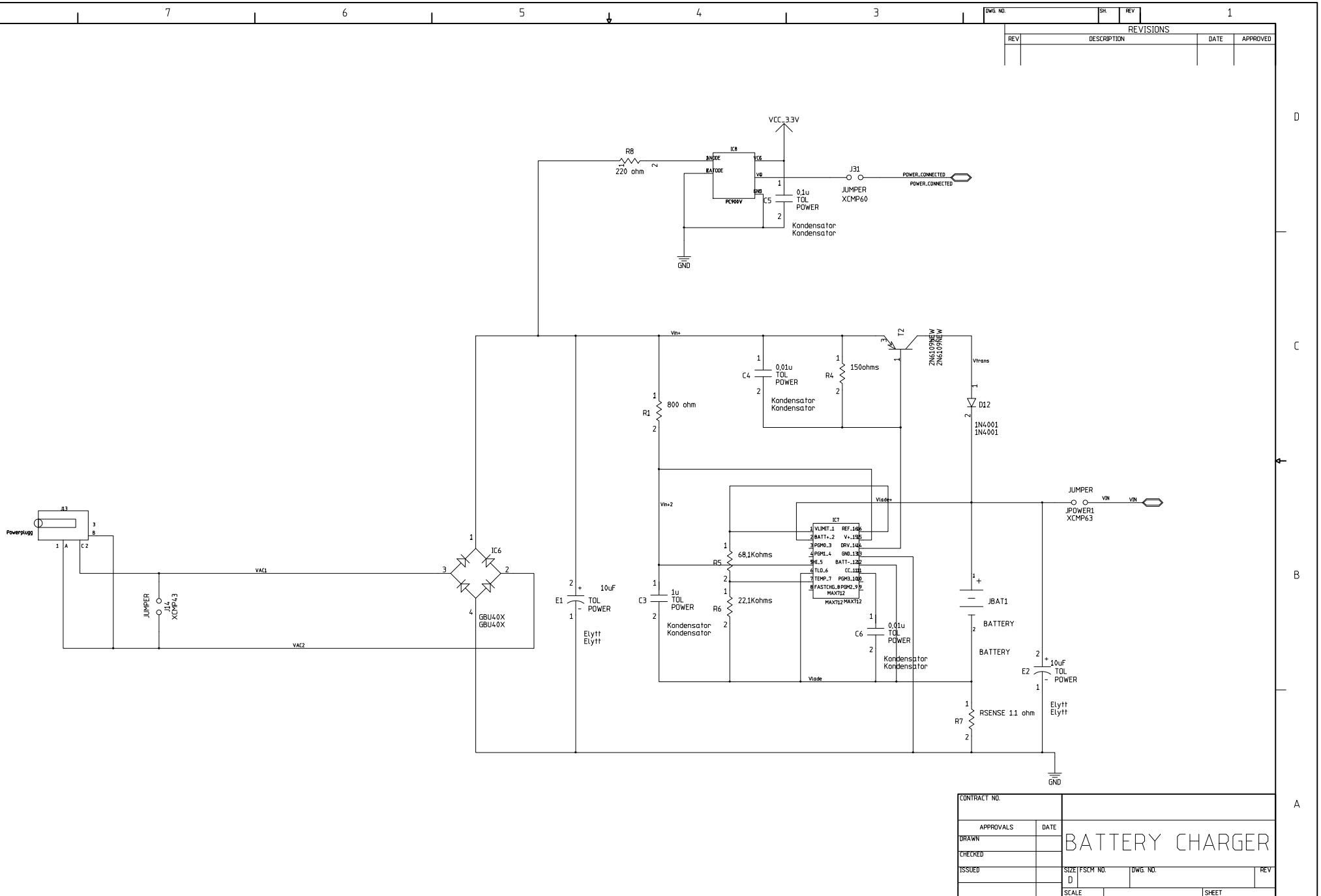
CONTRACT NO.	
APPROVALS	DATE
DRAWN	
CHECKED	
ISSUED	

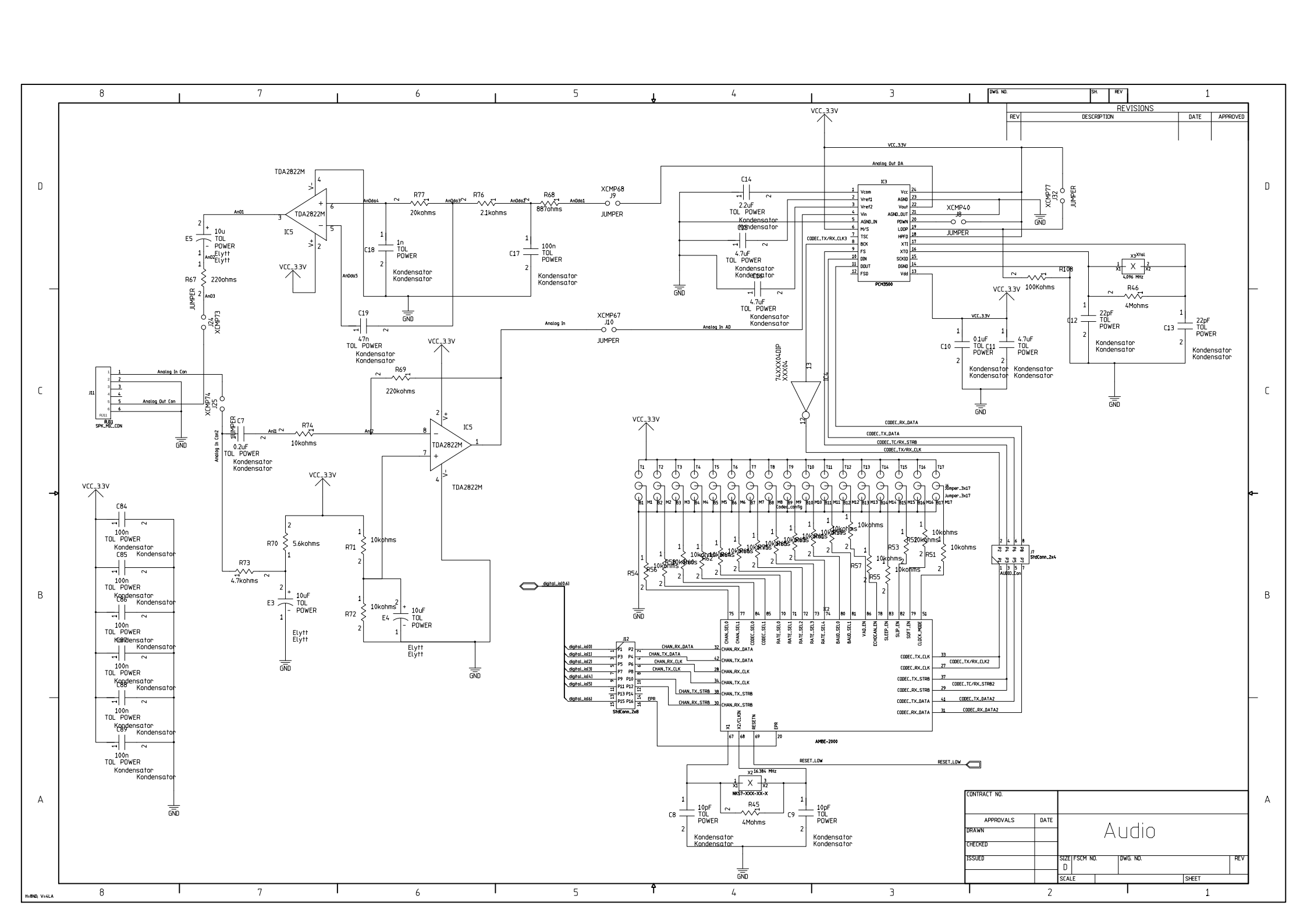
GSM

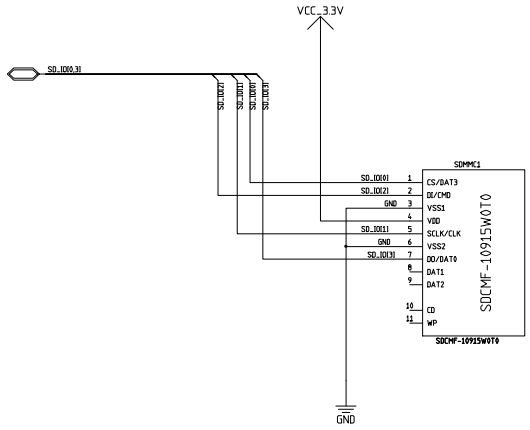
SIZE	FSCM NO.	DWG. NO.	REV
D			
SCALE		SHEET	

H:\BND; V:\4LA


DWG. NO.
SH.
REV
REVISIONS
REV
DESCRIPTION
DATE
APPROVED
VIN
JUMPER
XCMP131
J15
GND
IC9
LM1084
LM1084IT-5.0
10uF
TOL
POWER
E8
C20
100n
TOL
POWER
Kondensator
Kondensator
C21
E9
R28
150ohms
LED_5.5V
D3
LYS_DIODE
C40
C30
C31
C32
5.0V
JUMPER
J29
XCMP170
VCC
IC10
LM1084IT-3.3
E10
C33
C39
R29
50ohms
LED_3.3V
D2
E11
C34
C35
VCC_3.3V
J28
XCMP169
IC11
LM1084IT-ADJ
E6
C22
R49
121ohms
R50
53.6 ohms
E7
D1
LED_1.8V
C23
100nF
C24
VCC_1.8V
J27
XCMP168
IC12
DS1233 A
DS1233
DS1233
EconReset3.3V
RST
VCC
C38
10p
J30
RESET_BUTTON
R109
100Kohms
MR
IC13
RESET_HIGH
XXX00
74XXX00DIP
JRESETH1
XCMP154
R30
100Kohms
RESET_LOW
XCMP155
JRESETL1
C76
C77
C78
C79
C80
C81
C82
C83
1u
TOL POWER
Kondensator
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
POWER
SIZE
D
FSCM NO.
DWG. NO.
REV
SCALE
SHEET
H=BND; V=4LA


DWG. NO.
SH.
REV
REVISIONS
REV
DESCRIPTION
DATE
APPROVED
VCC_3.3V
R8
220 ohm
IC8
ANODE
CATODE
VCC
VO
GND
PC900V
J31
JUMPER
XCMP60
POWER_CONNECTED
POWER_CONNECTED
C5
0,1u
TOL
POWER
Kondensator
Kondensator
GND
Vin+
C4
0,01u
TOL
POWER
Kondensator
Kondensator
R4
150ohms
T2
2N6109NEW
2N6109NEW
Vtrans
D12
1N4001
1N4001
R1
800 ohm
Vin+2
Vlade+
JUMPER
VIN
VIN
JPOWER1
XCMP63
IC7
VLIMIT_1
BATT+_2
PGM0_3
PGM1_4
SHI_5
TLO_6
TEMP_7
FASTCHG_8
REF_16
V+_15
DRV_14
GND_13
BATT-_12
CC_11
PGM3_10
PGM2_9
MAX712
MAX712
R5
68,1Kohms
R6
22,1Kohms
C3
1u
TOL
POWER
Kondensator
Kondensator
C6
0,01u
TOL
POWER
Kondensator
Kondensator
Vlade
JBAT1
BATTERY
BATTERY
E2
10uF
TOL
POWER
Elytt
Elytt
R7
RSENSE 11 ohm
E1
10uF
TOL
POWER
Elytt
Elytt
J13
Powerplugg
VAC1
VAC2
JUMPER
J14
XCMP43
IC6
GBU40X
GBU40X
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
BATTERY CHARGER
SIZE
FSCM NO.
DWG. NO.
REV
D
SCALE
SHEET
H:\BND; V:\4LA


REVISIONS
REV
DESCRIPTION
DATE
APPROVED
DWG. NO.
SH.
REV
VCC_3.3V
TDA2822M
IC5
R77
20kohms
R76
2.1kohms
R68
887ohms
C18
1n
TOL POWER
Kondensator
C17
100n
C19
47n
E5
10u
R67
220ohms
XCMP68
J9
JUMPER
XCMP73
J24
XCMP74
J25
XCMP67
J10
Analog In
Analog In AD
Analog Out DA
XCMP40
J8
XCMP77
J32
C14
2.2uF
C15
4.7uF
C16
4.7uF
IC3
PCM3500
Vcom
Vref1
Vref2
Vin
AGND_IN
M/S
TSC
BCK
FS
DIN
DOUT
FSO
AGND
Vout
AGND_OUT
PDWN
LOOP
HPFD
XTI
XTO
SCKIO
DGND
Vdd
CODEC_TX/RX_CLK3
R108
100Kohms
X3 Xtal
4.096 MHz
R46
4Mohms
C12
22pF
C13
22pF
C10
0.1uF
C11
4.7uF
GND
J11
RJ11
SPK_MIC_CON
Analog In Con
Analog Out Con
C7
0.2uF
R74
10kohms
R69
220kohms
74LVXXX04DIP
XXX04
IC4
CODEC_RX_DATA
CODEC_TX_DATA
CODEC_TC/RX_STRB
CODEC_TX/RX_CLK
R70
5.6kohms
R71
10kohms
R72
10kohms
R73
4.7kohms
E3
10uF
E4
10uF
C84
C85
C86
C87
C88
C89
100n
R54
R56
R58
R60
R62
R64
R66
R68
R57
R55
R53
R52
R51
Codec_config
Jumper_3x17
J7
StdConn_2x4
AUDIO_Con
digital_io[0..6]
J12
StdConn_2x8
CHAN_RX_DATA
CHAN_TX_DATA
CHAN_RX_CLK
CHAN_TX_CLK
CHAN_TX_STRB
CHAN_RX_STRB
EPR
X2
AMBE-2000
CHAN_SEL0
CHAN_SEL1
CODEC_SEL0
CODEC_SEL1
RATE_SEL0
RATE_SEL1
RATE_SEL2
RATE_SEL3
RATE_SEL4
BAUD_SEL0
BAUD_SEL1
VAD_EN
ECHOCAN_EN
SLEEP_EN
SLIP_EN
SOFT_EN
CLOCK_MODE
X1
X2/CLKIN
RESETN
RESET_LOW
X2 16.384 MHz
MKS7-XXX-XX-X
C8
C9
10pF
R45
4Mohms
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
Audio
SIZE
FSCM NO.
DWG. NO.
REV
D
SCALE
SHEET



DWG. NO.	SH.	REV

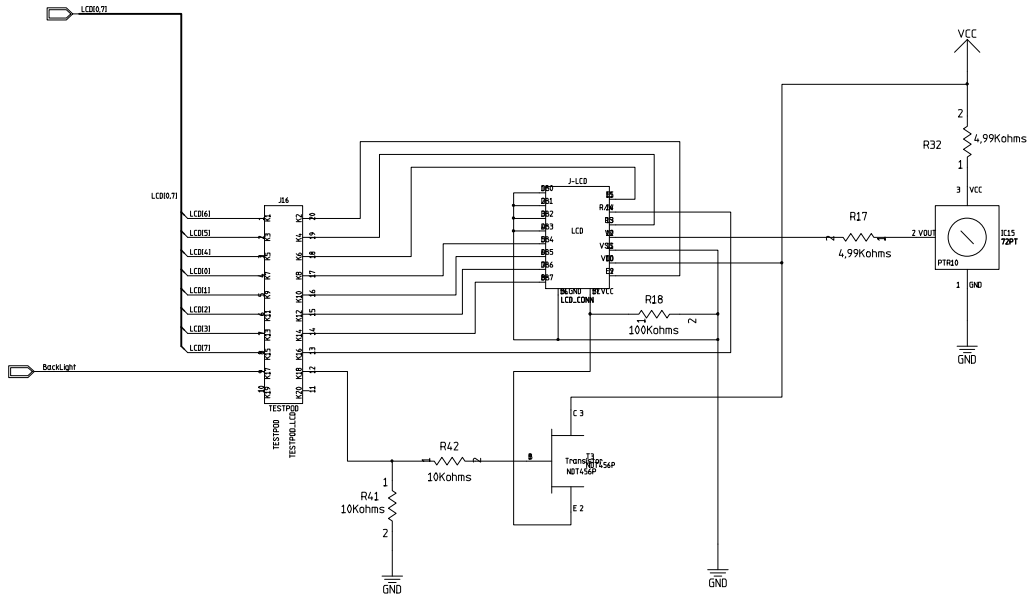
REVISIONS

REV	DESCRIPTION	DATE	APPROVED

CONTRACT NO.		
APPROVALS	DATE	SDMMC
DRAWN		
CHECKED		
ISSUED		

SIZE	FSCM NO.	DWG. NO.	REV
D			
SCALE		SHEET	

H=BND; V=4LA



DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED

CONTRACT NO.	
APPROVALS	DATE
DRAWN	
CHECKED	
ISSUED	

LCD

SIZE	FSCM NO.	DWG. NO.	REV
D			
SCALE		SHEET	

H=BND; V=4LA

Appendix C

Pin mappings

C.1 FPGA

Pin #	Net	VHDL port
3	crypt_io[16]	avr_write
4	crypt_io[17]	avr_read
5	crypt_io[18]	avr_address_latch_en
55	Data[0]	sram_data[0]
56	Data[1]	sram_data[1]
57	Data[2]	sram_data[2]
58	Data[3]	sram_data[3]
59	Data[4]	sram_data[4]
60	Data[5]	sram_data[5]
61	Data[6]	sram_data[6]
62	Data[7]	sram_data[7]
63	WE	sram_write_en
64	OE	sram_output_en
68	CE1	sram_en_1
69	CE2	sram_en_2
81	Address[0]	sram_address[0]
82	Address[1]	sram_address[1]
83	Address[2]	sram_address[2]
84	Address[3]	sram_address[3]
86	Address[4]	sram_address[4]
87	Address[5]	sram_address[5]
88	Address[6]	sram_address[6]
89	Address[7]	sram_address[7]
93	Address[8]	sram_address[8]
94	Address[9]	sram_address[9]
95	Address[10]	sram_address[10]
96	Address[11]	sram_address[11]
97	Address[12]	sram_address[12]
98	Address[13]	sram_address[13]
99	Address[14]	sram_address[14]
100	Address[15]	sram_address[15]
101	Address[16]	sram_address[16]

Pin #	**Net**	**VHDL port**
160		leds[0]
161		leds[1]
162		leds[2]
163		leds[3]
164		leds[4]
165		leds[5]
166		leds[6]
167		leds[7]
168	B1[0]	buttons[0]
169	B1[1]	buttons[1]
173	B1[2]	buttons[2]
174	B1[3]	buttons[3]
175	B1[4]	buttons[4]
176	B1[5]	buttons[5]
178	B1[6]	buttons[6]
179	B1[7]	buttons[7]
180	B1[8]	buttons[8]
181	B1[9]	buttons[9]
187	crypt_io[0]	avr_address_low_data[0]
188	crypt_io[1]	avr_address_low_data[1]
189	crypt_io[2]	avr_address_low_data[2]
191	crypt_io[3]	avr_address_low_data[3]
192	crypt_io[4]	avr_address_low_data[4]
193	crypt_io[5]	avr_address_low_data[5]
194	crypt_io[6]	avr_address_low_data[6]
198	crypt_io[7]	avr_address_low_data[7]
199	crypt_io[8]	avr_address_high[0]
200	crypt_io[9]	avr_address_high[1]
201	crypt_io[10]	avr_address_high[2]
202	crypt_io[11]	avr_address_high[3]
203	crypt_io[12]	avr_address_high[4]
204	crypt_io[13]	avr_address_high[5]
205	crypt_io[14]	avr_address_high[6]
206	crypt_io[15]	avr_address_high[7]
185		clock
46		keyboard_rows
36		keyboard_rows
45		keyboard_rows
40		keyboard_rows
44		keyboard_cols
41		keyboard_cols
43		keyboard_cols
42		keyboard_cols
21	avr_extra[1]	avr_interrupt

Appendix D

Budget

D.1 Budget table

Component	Name	Count	Price	Total
Headset		2	79	158
Casing		2	298	596
SD-card		2	119	238
AD/DA-chip	PCM3500EG4	4	56,29	225,16
SD-card socket	SDCMF-10915W0T0	5	11,25	56,2
Battery charger	MAX712CPE	4	84,56	338,23
Voltage regulators	LM1084IT-ADJ	5	14,12	70,6
Buttons	MCDTS6-1N	40	3,34	133,6
RAM	1 M (128k X 8) SRAM	5	58,76	293,8
Printed Circuit Board	Elprint-Cryptofon	10		10250
Audio compression codec	AMBE-2000	5		2000
GSM Modem	MC75	2		3000
Oscillators/crystalls		11		432,73
Resistors		110		70,24
Jumpers etc.		10		1056,88
Capacitators		230		332,9
Reset circuit	DS 1233	4	21,1	84,4
Potentiometer, 500 Ω		4	85,1	340,4
Swicth		4	18,9	75,6
Piezoelectric soundsgenerator		4	20,1	80,4
Sum	Total	453		19833,19

The limit was 23000 NOK in component expenses, and of that we have 3167 NOK left.

Appendix E

GSM AT commands

Listing over at commands used in out project.

E.1 Commands to test

In this section we will list up the tested AT commands and explain more thoroughly how these commands work, given in the table E.1.

AT Command	Description	Comment
AT+CPIN=<PIN Code>	Specifies the PIN-code	
AT^SICS=0,conType,GPRS0 AT^SICS=0,inactTO,"0" AT^SICS=0,dns1,"129.241.0.200" AT^SICS=0,authMode,"PAP" AT^SICS=0,apn,"vpn.gateway.no"	Sets up the GPRS connection profile (server)	The Access point vpn.gateway.no makes it possible for the GPRS unit to acquire a public IP address. We use one of NTNU's DNS servers.
AT^SISS=4,srvType,socket AT^SISS=4,conId,0 AT^SISS=4,address, "socktcp://listener:5434"	Sets up the GPRS service profile on profile number 4	Sets up a socket service, the modem acts as a socket server
AT^SISS=1,srvType,socket AT^SISS=1,conId,0 AT^SISS=1,address, "socktcp://129.241.76.18:4543"	Sets up the GPRS service profile on profile number 1	Sets up a socket service, the modem acts as a socket client
AT^SISO=4	Initiates the GPRS connection	
AT^SISO?	Check GPRS status	
AT^SISC=<service profile number, from 0-9>	Closes a GPRS connection on a given service profile	
AT^SISR=<service-profile>,<number of bytes>	Reads incoming data from the connection	
AT^SISR=<service-profile>,<number of bytes>	Writes outgoing data to the connection	
AT+CMGF=1	Sets message format	This makes it possible to read SMS in clear text
AT+CPMS="MT"	Sets SMS storage to modem + SIM card	SMS storage has now become a combination between SMS storage and module storage
AT+CMGS=*phone number* > text to be sent <CTRL-Z><ESC>	Sends an SMS	After text is written control signals ctrl-z and esc must be sent to the module
AT+CMGL=*category*	Lists SMS from a specified category	
AT+CMGL=?	Lists the different categories to be used above	
AT+CMGR=<index>	Reads an SMS	
AT+CMGD=<index>	Deletes an SMS	
AT+CSQ	Reports signal strength	
AT+CNMI=1,1,0,0,1	Makes the modem to give a signal when a new SMS is received	This behavior is by default turned off. We need to enter this command in order to get this notification.
ATD12345678	Calls number 12345678	
ATA	Answers a call	
ATH	Hangs up a call	
AT+CSNS=4	Sets the calling mode to "data"	

Table E.1: AT commands used

Appendix F

Memory mapped I/O addresses

Address	Direction	Function
0xE000	Both	Writing makes the LEDs display a pattern corresponding to the data bits. Reading returns the current pattern
0xE001	Read	The interrupt status register
0xE002 - 0xE003	Read	Returns the current status of the ten buttons
0xE004 - 0xE005	Read	Returns the current keyboard status
0xE006 - 0xE007	Read	Returns data from the sound codec (currently unused)
0xEC00 - 0xEC17	Write	Sets the corresponding byte of the encryption key
0xED00 - 0xED07	Write	Sets the corresponding byte of the current block to be encrypted or decrypted
0xEE00 - 0xEE07	Read	Returns the result of the most recently completed encryption or decryption
0xEF00	Read	Returns the status of the encryption module
0xEF01	Write	Writing anything to this address triggers an encryption of the current block
0xEF02	Write	Writing anything to this address triggers a decryption of the current block
0xEF03	Write	Writing 0 to this address disables cipher block chaining; anything else enables it
0xEF04	Write	Writing anything to this address will reset the cipher block chaining for the encryption
0xEF05	Write	Writing anything to this address will reset the cipher block chaining for the decryption

Appendix G

Code listings

G.1 VHDL code

All files that contain `-- CoreTex` in the first line originally contained the following copyright statement, but it has been moved here in order to conserve space.

```
--------------------------------------------------------------------------
--               (c) Copyright 2006, CoreTex Systems, LLC               --
--                        www.coretexsys.com                            --
--                                                                      --
--      This source file may be used and distributed without            --
--      restriction provided that this copyright statement is not       --
--      removed from the file and that any derivative work contains     --
--      the original copyright notice and the associated disclaimer.    --
--                                                                      --
--          THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY         --
--      EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED       --
--      TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       --
--      FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR          --
--      OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,             --
--      INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES        --
--      (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE       --
--      GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR            --
--      BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF      --
--      LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT      --
--      (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT      --
--      OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE             --
--      POSSIBILITY OF SUCH DAMAGE.                                     --
--                                                                      --
--------------------------------------------------------------------------


--------------------------------------------------------------------------

-- Poject structure:

--  |- tdes_top.vhd
--  |
--    |- des_cipher_top.vhd
--    |- des_top.vhd
```



```
--      |- block_top.vhd
--        |- add_key.vhd
--        |
--        |- add_left.vhd
--        |
--               |- e_expansion_function.vhd
--               |
--               |- p_box.vhd
--               |
--               |- s_box.vhd
--            |- s1_box.vhd
--            |- s2_box.vhd
--            |- s3_box.vhd
--            |- s4_box.vhd
--            |- s5_box.vhd
--            |- s6_box.vhd
--            |- s7_box.vhd
--            |- s8_box.vhd
--    |- key_schedule.vhd

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--
-- Title       : tdes_top
-- Company     : CoreTex Systems, LLC
--
--------------------------------------------------------------------------
```

Listing G.1: top_level.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top_level is
    port (
        avr_read : in std_logic;
        avr_write : in std_logic;
        avr_address_latch_en : in std_logic;
        --address_low : inout std_logic_vector(7 downto 0); -- also
           data
        --address_high : in std_logic_vector(15 downto 8)
        avr_address_high : in std_logic_vector(7 downto 0);
        avr_address_low_data : inout std_logic_vector(7 downto 0);  --
           overlaps the low bits of address

        sram_address : out std_logic_vector(16 downto 0);
        sram_data : inout std_logic_vector(7 downto 0);
        sram_en_1 : out std_logic;
        sram_en_2 : out std_logic;
        sram_write_en : out std_logic;
        sram_output_en : out std_logic;
```

```
        leds : out std_logic_vector(7 downto 0);
        buttons : in std_logic_vector(9 downto 0);

        clock : in std_logic;

        codec_rx_data : out std_logic;
        codec_tx_data : in std_logic;
        codec_rx_clk : out std_logic;
        codec_tx_clk : out std_logic;
        codec_tx_strb : inout std_logic;
        codec_rx_strb : out std_logic;
        codec_epr : in std_logic;

        keyboard_rows : out std_logic_vector(3 downto 0);
        keyboard_cols : inout std_logic_vector(3 downto 0);

        avr_interrupt : out std_logic
        --reset : in std_logic;
    );

    attribute clock_buffer: string;
    attribute clock_buffer of avr_address_latch_en: signal is "ibuf";
    attribute clock_buffer of avr_write: signal is "ibuf";
    attribute clock_buffer of avr_read: signal is "ibuf";

    attribute LOC : string;
    attribute LOC of avr_read : signal is "P4";
    attribute LOC of avr_write : signal is "P3";
    attribute LOC of avr_address_latch_en : signal is "P5";
    attribute LOC of avr_address_high : signal is "P206,P205,P204,P203,
       P202,P201,P200,P199";
    attribute LOC of avr_address_low_data : signal is "P198,P194,P193,
       P192,P191,P189,P188,P187";

    attribute LOC of leds : signal is "P167,P166,P165,P164,P163,P162,
       P161,P160";
    attribute LOC of buttons : signal is "P181,P180,P179,P178,P176,P175
       ,P174,P173,P169,P168";

    attribute LOC of sram_address : signal is "P101,P100,P99,P98,P97,
       P96,P95,P94,P93,P89,P88,P87,P86,P84,P83,P82,P81";
    attribute LOC of sram_data : signal is "P62,P61,P60,P59,P58,P57,P56
       ,P55";
    attribute LOC of sram_en_1 : signal is "P68";
    attribute LOC of sram_en_2 : signal is "P69";
    attribute LOC of sram_write_en : signal is "P63";
    attribute LOC of sram_output_en : signal is "P64";

    attribute LOC of codec_rx_data : signal is "P27";
    attribute LOC of codec_tx_data : signal is "P29";
    attribute LOC of codec_rx_clk : signal is "P30";
```



```
    attribute LOC of codec_tx_clk : signal is "P31";
    attribute LOC of codec_tx_strb : signal is "P33";
    attribute LOC of codec_rx_strb : signal is "P34";
    attribute LOC of codec_epr : signal is "P35";

    attribute LOC of keyboard_rows : signal is "P40,P45,P36,P46";
    attribute LOC of keyboard_cols : signal is "P42,P43,P41,P44";
    attribute LOC of clock : signal is "P185";

    attribute LOC of avr_interrupt : signal is "P21";
end top_level;

architecture Behavioral of top_level is

    component clock_divider is
        generic (width : integer);
        port (
            clock_in : in std_logic;
            value : out std_logic_vector(width - 1 downto 0)
        );
    end component;


    component keyboard_controller
      port (
         clock        : in std_logic;

         rows         : out std_logic_vector(3 downto 0);
         columns      : in std_logic_vector(3 downto 0);

         mem_address  : in std_logic_vector(15 downto 0);
         mem_read     : in std_logic;
         mem_data_out : out std_logic_vector(7 downto 0);

         interrupt    : out std_logic
      );
    end component;

    component pulldown
        port (O : out std_ulogic);
    end component;

    component codec_rx_control is
        port (
            clock : in std_logic;

            mem_address : in std_logic_vector(15 downto 0);
            mem_read : in std_logic;
            mem_data_out : out std_logic_vector(7 downto 0);

            chan_tx_clk : out std_logic;
            chan_tx_strb : out std_logic;
```

```
            chan_tx_data : in std_logic
        );
    end component;

    component sram_controller
        port (
            sram_address : out std_logic_vector(16 downto 0);
            sram_data : inout std_logic_vector(7 downto 0);
            sram_en_1 : out std_logic;
            sram_en_2 : out std_logic;
            sram_write_en : out std_logic;
            sram_output_en : out std_logic;

            mem_address  : in std_logic_vector(15 downto 0);
            mem_read     : in std_logic;
            mem_data_out : out std_logic_vector(7 downto 0);
            mem_write    : in std_logic;
            mem_data_in  : in std_logic_vector(7 downto 0)
        );
    end component;

    component tdes_control is
        port (
            clock_32MHz : std_logic;
            clock_8MHz : std_logic;

            mem_address  : in std_logic_vector(15 downto 0);
            mem_read     : in std_logic;
            mem_data_out : out std_logic_vector(7 downto 0);
            mem_write    : in std_logic;
            mem_data_in  : in std_logic_vector(7 downto 0);

            interrupt    : out std_logic
        );
    end component;

    component buttons_controller is
        port (
            clock        : in std_logic;

            buttons      : in std_logic_vector(9 downto 0);

            mem_address  : in std_logic_vector(15 downto 0);
            mem_read     : in std_logic;
            mem_data_out : out std_logic_vector(7 downto 0);

            interrupt    : out std_logic
        );
    end component;

    signal mem_address_latched : std_logic_vector(15 downto 0);
    signal leds_value : std_logic_vector(7 downto 0) := b"11111001";
```



```
    signal keyboard_read : std_logic := '0';
    signal keyboard_data_out : std_logic_vector(7 downto 0);
    signal keyboard_interrupt : std_logic := '1';

    signal interrupt_status : std_logic_vector(7 downto 0);

    signal clocks : std_logic_vector(4 downto 0);
    signal clock_8MHz : std_logic;
    signal clock_1MHz : std_logic;

    signal codec_rx_read : std_logic := '0';
    signal codec_rx_data_out : std_logic_vector(7 downto 0);

    signal sram_ctrl_read : std_logic;
    signal sram_ctrl_data_out  : std_logic_vector(7 downto 0);
    signal sram_ctrl_write : std_logic;
    signal sram_ctrl_data_in : std_logic_vector(7 downto 0);

    signal tdes_mem_read : std_logic;
    signal tdes_mem_data_out : std_logic_vector(7 downto 0);
    signal tdes_mem_write : std_logic;
    signal tdes_mem_data_in : std_logic_vector(7 downto 0);
    signal tdes_interrupt : std_logic;

    signal buttons_mem_read : std_logic;
    signal buttons_mem_data_out : std_logic_vector(7 downto 0);
    signal buttons_interrupt : std_logic;

begin

    --- Clock divider

    CLOCK_DIVIDER_INST: clock_divider
    generic map ( width => 5) port map (
        clock_in => clock,
        value => clocks
    );

    process(clocks)
    begin
        -- clock_16MHz <= clocks(0);
        clock_8MHz <= clocks(1);
        -- clock_4MHz <= clocks(2);
        -- clock_2MHz <= clocks(3);
        clock_1MHz <= clocks(4);
    end process;


    -- Keyboard

    KEYBOARD_INSTANCE: keyboard_controller
```

```
    port map (
        clock => clock,

        rows => keyboard_rows,
        columns => keyboard_cols,

        mem_address => mem_address_latched,
        mem_read => keyboard_read,
        mem_data_out => keyboard_data_out,

        interrupt => keyboard_interrupt
    );

    PULLDOWN_INSTANCE_0 : pulldown
    port map (O => keyboard_cols(0));

    PULLDOWN_INSTANCE_1 : pulldown
    port map (O => keyboard_cols(1));

    PULLDOWN_INSTANCE_2 : pulldown
    port map (O => keyboard_cols(2));

    PULLDOWN_INSTANCE_3 : pulldown
    port map (O => keyboard_cols(3));


    -- Codec RX channel (read from codec)
    CODEC_RX_CONTROL_INST: codec_rx_control
    port map (
        clock => clock_1MHz,

        mem_address => mem_address_latched,
        mem_read => codec_rx_read,
        mem_data_out => codec_rx_data_out,

        chan_tx_clk => codec_tx_clk,
        chan_tx_strb => codec_tx_strb,
        chan_tx_data => codec_tx_data
    );


    -- External memory
    SRAM_CONTROLLER_INST: sram_controller
    port map (
        sram_address => sram_address,
        sram_data => sram_data,
        sram_en_1 => sram_en_1,
        sram_en_2 => sram_en_2,
        sram_write_en => sram_write_en,
        sram_output_en => sram_output_en,

        mem_address => mem_address_latched,
```



```
        mem_read => sram_ctrl_read,
        mem_data_out => sram_ctrl_data_out,
        mem_write => sram_ctrl_write,
        mem_data_in => sram_ctrl_data_in
    );

    TDES_CONTROL_INST: tdes_control
    port map (
        clock_32MHz => clock,
        clock_8MHz => clock_8MHz,

        mem_address => mem_address_latched,
        mem_read => tdes_mem_read,
        mem_data_out => tdes_mem_data_out,
        mem_write => tdes_mem_write,
        mem_data_in => tdes_mem_data_in,

        interrupt => tdes_interrupt
    );

    BUTTONS_CONTROLLER_INST: buttons_controller
    port map (
        clock => clock,

        buttons => buttons,

        mem_address => mem_address_latched,
        mem_read => buttons_mem_read,
        mem_data_out => buttons_mem_data_out,

        interrupt => buttons_interrupt
    );


    -- A latch that stores all of the 16 address bits when ALE goes low
    process(avr_address_latch_en)
    begin
        if (falling_edge(avr_address_latch_en)) then
            mem_address_latched(15 downto 8) <= avr_address_high;
            mem_address_latched(7 downto 0) <= avr_address_low_data;
        end if;
    end process;

    process(avr_write, mem_address_latched, avr_address_low_data)
    begin
        sram_ctrl_write <= '0';
        tdes_mem_write <= '0';

        if (avr_write = '0') then
            if mem_address_latched(15 downto 13) = "111" then
                if (mem_address_latched = x"E000") then
                    leds_value <= avr_address_low_data;
```

```
            elsif (mem_address_latched(15 downto 8) = x"EC" or
                mem_address_latched(15 downto 8) = x"ED" or
                mem_address_latched(15 downto 8) = x"EE" or
                mem_address_latched(15 downto 8) = x"EF") then

                tdes_mem_data_in <= avr_address_low_data;
                tdes_mem_write <= '1';
            end if;
        else
            sram_ctrl_write <= '1';
            sram_ctrl_data_in <= avr_address_low_data;
        end if;
    end if;
end process;


process(avr_read, mem_address_latched, leds_value, interrupt_status
    , keyboard_data_out,
    codec_rx_data_out, tdes_mem_data_out, sram_ctrl_data_out)
begin
    avr_address_low_data <= "ZZZZZZZZ";
    buttons_mem_read <= '0';
    keyboard_read <= '0';
    codec_rx_read <= '0';
    sram_ctrl_read <= '0';
    tdes_mem_read <= '0';

    if (avr_read = '0') then
        if mem_address_latched(15 downto 13) = "111" then
            if (mem_address_latched = x"E000") then
                avr_address_low_data <= leds_value;
            elsif (mem_address_latched = x"E001") then
                avr_address_low_data <= interrupt_status;
            elsif (mem_address_latched = x"E002" or
                mem_address_latched = x"E003") then
                avr_address_low_data <= buttons_mem_data_out;
                buttons_mem_read <= '1';
            elsif (mem_address_latched = x"E004" or
                mem_address_latched = x"E005") then
                avr_address_low_data <= keyboard_data_out;
                keyboard_read <= '1';
            elsif (mem_address_latched = x"E006" or
                mem_address_latched = x"E007") then
                avr_address_low_data <= codec_rx_data_out;
                codec_rx_read <= '1';
            elsif (mem_address_latched(15 downto 8) = x"EC" or
                mem_address_latched(15 downto 8) = x"ED" or
                mem_address_latched(15 downto 8) = x"EE" or
                mem_address_latched(15 downto 8) = x"EF") then

                avr_address_low_data <= tdes_mem_data_out;
                tdes_mem_read <= '1';
```



```
                end if;
            else
                sram_ctrl_read <= '1';
                avr_address_low_data <= sram_ctrl_data_out;
            end if;
        end if;

    end process;

    -- Leds
    leds <= leds_value;


    -- Interrupt
    interrupt_status(7 downto 3) <= b"00000";
    interrupt_status(0) <= keyboard_interrupt;
    interrupt_status(1) <= tdes_interrupt;
    interrupt_status(2) <= buttons_interrupt;

    process(interrupt_status)
    begin
        -- Set interrupt line low as long as we have an interrupt.
        if interrupt_status /= b"00000000" then
            avr_interrupt <= '0';
        else
            avr_interrupt <= '1';
        end if;
    end process;

    codec_rx_clk <= '0';
    codec_rx_strb <= '0';
    codec_rx_data <= '0';

end Behavioral;
```

Listing G.2: sram_controller.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sram_controller is
    port (
        sram_address : out std_logic_vector(16 downto 0);
        sram_data : inout std_logic_vector(7 downto 0);
        sram_en_1 : out std_logic;
        sram_en_2 : out std_logic;
        sram_write_en : out std_logic;
        sram_output_en : out std_logic;

        mem_address  : in std_logic_vector(15 downto 0);
        mem_read     : in std_logic;
        mem_data_out : out std_logic_vector(7 downto 0);
        mem_write    : in std_logic;
```

```
        mem_data_in  : in std_logic_vector(7 downto 0)
    );
end sram_controller;

architecture sram_controller_impl of sram_controller is

begin
    sram_address(16) <= '0';
    sram_address(15 downto 0) <= mem_address;

    sram_en_1 <= '0';
    sram_en_2 <= '1';

    process(mem_read, sram_data)
    begin
        if mem_read = '1' then
            mem_data_out <= sram_data;
            sram_output_en <= '0';
        else
            mem_data_out <= "00000000";
            sram_output_en <= '1';
        end if;
    end process;

    process(mem_write, sram_data, mem_data_in)
    begin
        if mem_write = '1' then
            sram_data <= mem_data_in;
            sram_write_en <= '0';
        else
            sram_data <= "ZZZZZZZZ";
            sram_write_en <= '1';
        end if;
    end process;

end sram_controller_impl;
```

Listing G.3: tdes_control.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tdes_control is
    port (
        clock_32Mhz : std_logic;
        clock_8Mhz : std_logic;

        mem_address  : in std_logic_vector(15 downto 0);
        mem_read     : in std_logic;
        mem_data_out : out std_logic_vector(7 downto 0);
        mem_write    : in std_logic;
```



```
        mem_data_in  : in std_logic_vector(7 downto 0);

        interrupt    : out std_logic
    );
end tdes_control;

architecture tdes_control_impl of tdes_control is

    component tdes_top is
        port(
            --
            -- inputs for key expander
            --
            key1_in:        in std_logic_vector(0 to 63);
            key2_in:        in std_logic_vector(0 to 63);
            key3_in:        in std_logic_vector(0 to 63);

            --
            -- function select
            --
            function_select:    in  std_logic; -- active when
                encryption, inactive when decryption

            --
            -- input into des_cipher_top
            --
            data_in:        in std_logic_vector(0 to 63);

            --
            -- input into des_cipher_top
            --
            data_out:   out std_logic_vector(0 to 63);

            --
            -- data interface to MCU
            --
            lddata:         in  std_logic;  -- active when data for
                loading is ready
            ldkey:          in  std_logic;  -- active when key for
                loading is ready
            out_ready:  out std_logic;  -- active when encryption of
                data is done

            --
            -- General clock and reset
            --
            reset: in std_logic;
            clock: in std_logic
        );
    end component;
```

```
    signal crypt_key : std_logic_vector(0 to 191) := x"
       000000000000000000000000000000000000000000000000";
    signal crypt_data_in : std_logic_vector(0 to 63) := x"
       0000000000000000";
    signal actual_crypt_data_in : std_logic_vector(0 to 63) := x"
       0000000000000000";
    signal crypt_data_out : std_logic_vector(0 to 63);
    signal actual_crypt_data_out : std_logic_vector(0 to 63);
    signal previous_encryption_result : std_logic_vector(0 to 63) := x"
       0000000000000000";
    signal previous_decryption_input : std_logic_vector(0 to 63) := x"
       0000000000000000";
    signal use_cipher_block_chaining : std_logic := '1';

    signal crypt_mode : std_logic := '0';
    signal crypt_load_data : std_logic := '0';
    signal crypt_load_key : std_logic := '0';
    signal out_ready : std_logic;

    signal tdes_reset : std_logic := '0';

    constant HOLD_CYCLES : integer := 5; -- 32MHz / 8MHz = 4 (At least
       4 cycles needed.)
    signal hold_counter : integer := 0;

    type tdes_state_type is (waiting, start_operation, operating,
       finished);
    signal state : tdes_state_type := waiting;
begin

    TDES_INSTANCE: tdes_top
        port map (
            key1_in => crypt_key(0 to 63),
            key2_in => crypt_key(64 to 127),
            key3_in => crypt_key(128 to 191),

            function_select => crypt_mode,

            data_in => actual_crypt_data_in,
            data_out => actual_crypt_data_out,

            lddata => crypt_load_data,
            ldkey => crypt_load_key,

            out_ready => out_ready,

            reset => tdes_reset,

            clock => clock_8Mhz
        );

    process(clock_32MHz)
```



```
    begin
        if rising_edge(clock_32MHz) then
            crypt_load_data <= '0';
            crypt_load_key <= '0';
            tdes_reset <= '0';
            interrupt <= '0';

            case state is
                when waiting =>
                    if mem_write = '1' then
                        -- Encrypt
                        if mem_address = x"EF01" then
                            state <= start_operation;
                            crypt_mode <= '1';
                            tdes_reset <= '1';

                        -- Decrypt
                        elsif mem_address = x"EF02" then
                            state <= start_operation;
                            crypt_mode <= '0';
                            tdes_reset <= '1';

                        -- Reset CBC for encryption
                        elsif (mem_address = x"EF04") then
                            previous_encryption_result <= x"
                               0000000000000000";

                        -- Reset CBC for decryption
                        elsif (mem_address = x"EF05") then
                            previous_decryption_input <= x"
                                0000000000000000";
                        end if;
                    end if;

                when start_operation =>

                    crypt_load_data <= '1';
                    crypt_load_key <= '1';
                    if (crypt_mode = '1' and use_cipher_block_chaining
                       = '1') then
                        actual_crypt_data_in <= crypt_data_in xor
                           previous_encryption_result;
                    else
                        actual_crypt_data_in <= crypt_data_in;
                    end if;

                    tdes_reset <= '1';

                    hold_counter <= hold_counter + 1;
                    if hold_counter = HOLD_CYCLES then
                        hold_counter <= 0;
                        state <= operating;
```

```
                end if;

            when operating =>
                crypt_load_data <= '1';
                crypt_load_key <= '1';

                if out_ready = '1' then
                    if (crypt_mode = '1') then
                        previous_encryption_result <=
                           actual_crypt_data_out;
                    end if;
                    state <= finished;
                    if (crypt_mode = '0' and
                       use_cipher_block_chaining = '1') then
                        crypt_data_out <= actual_crypt_data_out xor
                            previous_decryption_input;
                    else
                        crypt_data_out <= actual_crypt_data_out;
                    end if;
                    if (crypt_mode = '0') then
                        previous_decryption_input <= crypt_data_in;
                    end if;
                end if;

            when finished =>
                interrupt <= '1';



                if mem_read = '1' and mem_address(15 downto 8) = x"
                   EE" then
                    state <= waiting;
                end if;
        end case;
    end if;
end process;

process(mem_address, mem_write, mem_data_in)
begin
    if mem_write = '1' then

        -- DES key
        if (mem_address = x"EC00") then
            crypt_key(0 to 7) <= mem_data_in;
        elsif (mem_address = x"EC01") then
            crypt_key(8 to 15) <= mem_data_in;
        elsif (mem_address = x"EC02") then
            crypt_key(16 to 23) <= mem_data_in;
        elsif (mem_address = x"EC03") then
            crypt_key(24 to 31) <= mem_data_in;
        elsif (mem_address = x"EC04") then
            crypt_key(32 to 39) <= mem_data_in;
```



```
        elsif (mem_address = x"EC05") then
            crypt_key(40 to 47) <= mem_data_in;
        elsif (mem_address = x"EC06") then
            crypt_key(48 to 55) <= mem_data_in;
        elsif (mem_address = x"EC07") then
            crypt_key(56 to 63) <= mem_data_in;
        elsif (mem_address = x"EC08") then
            crypt_key(64 to 71) <= mem_data_in;
        elsif (mem_address = x"EC09") then
            crypt_key(72 to 79) <= mem_data_in;
        elsif (mem_address = x"EC0A") then
            crypt_key(80 to 87) <= mem_data_in;
        elsif (mem_address = x"EC0B") then
            crypt_key(88 to 95) <= mem_data_in;
        elsif (mem_address = x"EC0C") then
            crypt_key(96 to 103) <= mem_data_in;
        elsif (mem_address = x"EC0D") then
            crypt_key(104 to 111) <= mem_data_in;
        elsif (mem_address = x"EC0E") then
            crypt_key(112 to 119) <= mem_data_in;
        elsif (mem_address = x"EC0F") then
            crypt_key(120 to 127) <= mem_data_in;
        elsif (mem_address = x"EC10") then
            crypt_key(128 to 135) <= mem_data_in;
        elsif (mem_address = x"EC11") then
            crypt_key(136 to 143) <= mem_data_in;
        elsif (mem_address = x"EC12") then
            crypt_key(144 to 151) <= mem_data_in;
        elsif (mem_address = x"EC13") then
            crypt_key(152 to 159) <= mem_data_in;
        elsif (mem_address = x"EC14") then
            crypt_key(160 to 167) <= mem_data_in;
        elsif (mem_address = x"EC15") then
            crypt_key(168 to 175) <= mem_data_in;
        elsif (mem_address = x"EC16") then
            crypt_key(176 to 183) <= mem_data_in;
        elsif (mem_address = x"EC17") then
            crypt_key(184 to 191) <= mem_data_in;

        -- DES data input
        elsif (mem_address = x"ED00") then
            crypt_data_in(0 to 7) <= mem_data_in;
        elsif (mem_address = x"ED01") then
            crypt_data_in(8 to 15) <= mem_data_in;
        elsif (mem_address = x"ED02") then
            crypt_data_in(16 to 23) <= mem_data_in;
        elsif (mem_address = x"ED03") then
            crypt_data_in(24 to 31) <= mem_data_in;
        elsif (mem_address = x"ED04") then
            crypt_data_in(32 to 39) <= mem_data_in;
        elsif (mem_address = x"ED05") then
            crypt_data_in(40 to 47) <= mem_data_in;
```

```
            elsif (mem_address = x"ED06") then
                crypt_data_in(48 to 55) <= mem_data_in;
            elsif (mem_address = x"ED07") then
                crypt_data_in(56 to 63) <= mem_data_in;

            -- Choose whether or not to use cipher block chaining (CBC)
            elsif (mem_address = x"EF03") then
                if (mem_data_in = x"00") then
                    use_cipher_block_chaining <= '0';
                else
                    use_cipher_block_chaining <= '1';
                end if;
            end if;
        end if;
    end process;

    process(mem_address, mem_read, crypt_data_out, state)
    begin
        mem_data_out <= b"00000000";
        if mem_read = '1' then
            if (mem_address = x"EF00") then
                case state is
                    when waiting => mem_data_out <= x"01";
                    when start_operation => mem_data_out <= x"02";
                    when operating => mem_data_out <= x"03";
                    when finished => mem_data_out <= x"04";
                    when others => mem_data_out <= x"cc";
                end case;
            elsif (mem_address = x"EE00") then
                mem_data_out <= crypt_data_out(0 to 7);
            elsif (mem_address = x"EE01") then
                mem_data_out <= crypt_data_out(8 to 15);
            elsif (mem_address = x"EE02") then
                mem_data_out <= crypt_data_out(16 to 23);
            elsif (mem_address = x"EE03") then
                mem_data_out <= crypt_data_out(24 to 31);
            elsif (mem_address = x"EE04") then
                mem_data_out <= crypt_data_out(32 to 39);
            elsif (mem_address = x"EE05") then
                mem_data_out <= crypt_data_out(40 to 47);
            elsif (mem_address = x"EE06") then
                mem_data_out <= crypt_data_out(48 to 55);
            elsif (mem_address = x"EE07") then
                mem_data_out <= crypt_data_out(56 to 63);
            end if;
        end if;
    end process;

end tdes_control_impl;
```

Listing G.4: buttons_controller.vhd


```
-- This component implements a keyboard controller. It connects to the
   keyboard pins,
-- and allows access to the keyboard state through the memory bus. An
   interrupt will be
-- delivered when the state of the keyboard changes and after a reset.

library IEEE;
use IEEE.std_logic_1164.all;

entity buttons_controller is
  port (
    clock        : in std_logic;

    buttons      : in std_logic_vector(9 downto 0);

    mem_address  : in std_logic_vector(15 downto 0);
    mem_read     : in std_logic;
    mem_data_out : out std_logic_vector(7 downto 0);

    interrupt    : out std_logic
  );
end buttons_controller;

architecture buttons_controller_impl of buttons_controller is
  component keyboard_debounce is
    port (
      clock        : in std_logic;

      input_state     : in std_logic_vector(15 downto 0);
      debounced_state : out std_logic_vector(15 downto 0)
    );
  end component;

  signal buttons_state : std_logic_vector(15 downto 0);
  signal debounced_state : std_logic_vector(15 downto 0);
  signal previous_state : std_logic_vector(15 downto 0) := b"
     0000000000000000";


  signal memory_low_byte : std_logic_vector(7 downto 0) := b"00000000";
  signal read_low_byte : std_logic := '0';

  signal memory_high_byte : std_logic_vector(7 downto 0) := b"00000000"
     ;
  signal read_high_byte : std_logic := '0';
begin

  buttons_state(15 downto 10) <= "000000";
  buttons_state(9 downto 0) <= buttons;

  DEBOUNCE_INSTANCE : keyboard_debounce port map (
    clock => clock,
```

```
    input_state => buttons_state,
    debounced_state => debounced_state
  );

  -- Raise an interrupt if the CPU hasn't read the latest keyboard
     state.
  interrupt <= (not read_low_byte) or (not read_high_byte);

  process(clock)
  begin
    if rising_edge(clock) then
      if debounced_state /= previous_state then
        -- The state has changed. Update it and reset the read flags.
        previous_state <= debounced_state;

        memory_low_byte <= debounced_state(15 downto 8);
        read_low_byte <= '0';

        memory_high_byte <= debounced_state(7 downto 0);
        read_high_byte <= '0';

      elsif mem_read = '1' then
        -- We only read data if the state hasn't changed because this
           makes the code much simpler.
        -- This should be OK since we can wait several cycles before
           returning data to a read
        -- operation, and the state will never change several cycles in
            a row (thanks to debouncing).

        -- We only care about the top bit in the memory address.
        if mem_address(0) = '0' then
          mem_data_out <= memory_low_byte;
          read_low_byte <= '1';
        else
          mem_data_out <= memory_high_byte;
          read_high_byte <= '1';
        end if;
      end if;
    end if;
  end process;

end buttons_controller_impl;
```

Listing G.5: clock_divider.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity clock_divider is
    generic (width : integer);
```



```
    port (
        clock_in : in std_logic;
        value : out std_logic_vector(width - 1 downto 0)
    );
end clock_divider;

architecture Behavioral of clock_divider is
    signal counter : std_logic_vector(width - 1 downto 0) :=
       std_logic_vector(to_unsigned(0, width));
begin

    value <= counter;

    process(clock_in)
    begin
        if (rising_edge(clock_in)) then
            counter <= counter + 1;
        end if;
    end process;
end Behavioral;
```

Listing G.6: codec_rx_channel.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity codec_rx_channel is
    port (
        clock : in std_logic;

        read_word : out std_logic_vector(15 downto 0);
        read_completed : out std_logic;

        chan_tx_clk : out std_logic;
        chan_tx_strb : out std_logic;
        chan_tx_data : in std_logic
    );
end codec_rx_channel;

architecture codec_rx_channel_impl of codec_rx_channel is
    type rx_state_type is (waiting, starting, reading);

    signal current_state : rx_state_type := waiting;
    signal current_word : std_logic_vector(15 downto 0) := "
       0000000000000000";
    signal current_index : integer := 0;

    signal completed_word : std_logic_vector(15 downto 0) := "
       0000000000000000";
begin

    read_word <= completed_word;
```

```
    chan_tx_clk <= clock;

    process(clock)
    begin
        if rising_edge(clock) then

            case current_state is
                when waiting =>
                    chan_tx_strb <= '1';
                    current_state <= starting;
                    read_completed <= '0';

                when starting =>
                    chan_tx_strb <= '0';
                    current_state <= reading;
                    read_completed <= '0';

                when reading =>
                    chan_tx_strb <= '0';
                    if current_index = 16 then
                        current_state <= waiting;
                        completed_word <= current_word;
                        read_completed <= '1';
                    else
                        current_state <= reading;
                        read_completed <= '0';
                    end if;

                when others =>
                    current_state <= waiting;
                    chan_tx_strb <= '0';
                    read_completed <= '0';
            end case;
        end if;
    end process;

    process(clock)
    begin
        if falling_edge(clock) then

            case current_state is
                when starting =>
                    current_index <= 0;

                when reading =>
                    current_index <= current_index + 1;
                    current_word(current_index) <= chan_tx_data;

                when others =>
                    current_index <= 0;
            end case;
        end if;
```



```
    end process;

end codec_rx_channel_impl;
```

Listing G.7: codec_rx_control.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity codec_rx_control is
    port (
        clock : in std_logic;

        mem_address : in std_logic_vector(15 downto 0);
        mem_read : in std_logic;
        mem_data_out : out std_logic_vector(7 downto 0);

        chan_tx_clk : out std_logic;
        chan_tx_strb : out std_logic;
        chan_tx_data : in std_logic
    );
end codec_rx_control;

architecture codec_rx_control_impl of codec_rx_control is
    component codec_rx_channel is
        port (
            clock : in std_logic;

            read_word : out std_logic_vector(15 downto 0);
            read_completed : out std_logic;

            chan_tx_clk : out std_logic;
            chan_tx_strb : out std_logic;
            chan_tx_data : in std_logic
        );
    end component;

    signal read_completed : std_logic;
    signal current_word : std_logic_vector(15 downto 0);
begin

    CODEC_RX_CHANNEL_INST: codec_rx_channel
    port map (
        clock => clock,

        read_word => current_word,
        read_completed => read_completed,

        chan_tx_clk => chan_tx_clk,
        chan_tx_strb => chan_tx_strb,
        chan_tx_data => chan_tx_data
    );
```

```
    process(mem_read, mem_address, current_word)
    begin
        if mem_read = '1' then
            if mem_address(0) = '0' then
                mem_data_out <= current_word(15 downto 8);
            else
                mem_data_out <= current_word(7 downto 0);
            end if;
        else
            mem_data_out <= "00000000";
        end if;
    end process;

end codec_rx_control_impl;
```

Listing G.8: keyboard.vhd

```
-- This component implements a keyboard decoder.
-- No debouncing is done.

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity keyboard is
  port (
    clock        : in std_logic;

    rows         : out std_logic_vector(3 downto 0);
    columns      : in std_logic_vector(3 downto 0);

    state        : out std_logic_vector(15 downto 0)
  );
end keyboard;

architecture keyboard_impl of keyboard is
  -- The number of clock cycles we should wait between setting the
     outputs and
  -- reading the inputs. This should be at least one.
  constant WAIT_CYCLES : integer := 100;

  -- The number of bits neccecary to store the WAIT_CYCLES number.
  constant WAIT_CYCLES_BITS : integer := 8;

  -- The counter for the delay before read.
  signal wait_counter : std_logic_vector(WAIT_CYCLES_BITS-1 downto 0)
        := std_logic_vector(to_unsigned(0, WAIT_CYCLES_BITS));

  -- The current row we are processing.
  signal current_row : std_logic_vector(1 downto 0) := "00";
```



```
  -- The current state we send.
  signal current_state : std_logic_vector(15 downto 0) := "
     0000000000000000";
begin

  state <= current_state;

  process(clock)
  begin
    if rising_edge(clock) then
      if wait_counter = std_logic_vector(to_unsigned(0,
         WAIT_CYCLES_BITS)) then
        -- We are on the first cycle of reading a row. Set the output
           to select the current row.
        case current_row is
          when "00" => rows <= "1ZZZ";
          when "01" => rows <= "Z1ZZ";
          when "10" => rows <= "ZZ1Z";
          when "11" => rows <= "ZZZ1";
          when others => rows <= "ZZZZ";
        end case;
      end if;

      if wait_counter = std_logic_vector(to_unsigned(WAIT_CYCLES,
         WAIT_CYCLES_BITS)) then
        -- We have waited the specified number of cycles, and are now
           going to
        -- read state of the keys in the current row.
        case current_row is
          when "00" => current_state(15 downto 12) <= columns;
          when "01" => current_state(11 downto 8) <= columns;
          when "10" => current_state(7 downto 4) <= columns;
          when "11" => current_state(3 downto 0) <= columns;
          when others => -- Do nothing.
        end case;

        -- Go to the next row. It is possible to wrap around.
        current_row <= current_row + 1;

        -- Reset the read counter.
        wait_counter <= std_logic_vector(to_unsigned(0,
           WAIT_CYCLES_BITS));
      else
        -- We have not yet reached the number of wait cycles we should
           have before we should
        -- read the inputs.

        -- Increase the wait counter.
        wait_counter <= wait_counter + 1;
      end if;
    end if;
```

```
  end process;

end keyboard_impl;
```

Listing G.9: keyboard_controller.vhd

```
-- This component implements a keyboard controller. It connects to the
   keyboard pins,
-- and allows access to the keyboard state through the memory bus. An
   interrupt will be
-- delivered when the state of the keyboard changes and after a reset.

library IEEE;
use IEEE.std_logic_1164.all;

entity keyboard_controller is
  port (
    clock        : in std_logic;

    rows         : out std_logic_vector(3 downto 0);
    columns      : in std_logic_vector(3 downto 0);

    mem_address  : in std_logic_vector(15 downto 0);
    mem_read     : in std_logic;
    mem_data_out : out std_logic_vector(7 downto 0);

    interrupt    : out std_logic
  );
end keyboard_controller;

architecture keyboard_controller_impl of keyboard_controller is
  component keyboard is
    port (
      clock        : in std_logic;

      rows         : out std_logic_vector(3 downto 0);
      columns      : in std_logic_vector(3 downto 0);

      state        : out std_logic_vector(15 downto 0)
    );
  end component;

  component keyboard_debounce is
    port (
      clock        : in std_logic;

      input_state     : in std_logic_vector(15 downto 0);
      debounced_state : out std_logic_vector(15 downto 0)
    );
  end component;

  signal keyboard_state : std_logic_vector(15 downto 0);
  signal debounced_state : std_logic_vector(15 downto 0);
```



```
  signal previous_state : std_logic_vector(15 downto 0) := b"
     0000000000000000";


  signal memory_low_byte : std_logic_vector(7 downto 0) := b"00000000";
  signal read_low_byte : std_logic := '0';

  signal memory_high_byte : std_logic_vector(7 downto 0) := b"00000000"
     ;
  signal read_high_byte : std_logic := '0';
begin

  KEYBOARD_INSTANCE: keyboard port map (
    clock => clock,
    rows => rows,
    columns => columns,
    state => keyboard_state
  );

  DEBOUNCE_INSTANCE : keyboard_debounce port map (
    clock => clock,
    input_state => keyboard_state,
    debounced_state => debounced_state
  );

  -- Raise an interrupt if the CPU hasn't read the latest keyboard
     state.
  interrupt <= (not read_low_byte) or (not read_high_byte);

  process(clock)
  begin
    if rising_edge(clock) then
      if debounced_state /= previous_state then
        -- The state has changed. Update it and reset the read flags.
        previous_state <= debounced_state;

        memory_low_byte <= debounced_state(15 downto 8);
        read_low_byte <= '0';

        memory_high_byte <= debounced_state(7 downto 0);
        read_high_byte <= '0';

      elsif mem_read = '1' then
        -- We only read data if the state hasn't changed because this
           makes the code much simpler.
        -- This should be OK since we can wait several cycles before
           returning data to a read
        -- operation, and the state will never change several cycles in
            a row (thanks to debouncing).

        -- We only care about the top bit in the memory address.
        if mem_address(0) = '0' then
```

```
          mem_data_out <= memory_low_byte;
          read_low_byte <= '1';
        else
          mem_data_out <= memory_high_byte;
          read_high_byte <= '1';
        end if;
      end if;
    end if;
  end process;

end keyboard_controller_impl;
```

Listing G.10: keyboard_debounce.vhd

```
-- This component implements a keyboard debouncer.

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity keyboard_debounce is
  port (
     clock        : in std_logic;

    input_state     : in std_logic_vector(15 downto 0);
    debounced_state : out std_logic_vector(15 downto 0)
  );
end keyboard_debounce;

architecture keyboard_debounce_impl of keyboard_debounce is
  -- The number of clock cycles we should wait between setting the
     outputs and
  -- reading the inputs. This should be at least one.
  constant WAIT_CYCLES : integer := 1000;

  -- The number of bits neccecary to store the WAIT_CYCLES number.
  constant WAIT_CYCLES_BITS : integer := 10;

  -- The counter for the delay before we consider a signal to be stable
     .
  signal wait_counter : std_logic_vector(WAIT_CYCLES_BITS-1 downto 0)
        := std_logic_vector(to_unsigned(0, WAIT_CYCLES_BITS));

  -- The previous state we read.
  signal previous_state : std_logic_vector(15 downto 0)
        := "0000000000000000";

  -- The current state we send.
  signal current_state : std_logic_vector(15 downto 0)
        := "0000000000000000";

begin
```



```
    debounced_state <= current_state;

  process(clock)
  begin
    if rising_edge(clock) then

      if previous_state = input_state then
        -- The signal hasn't changed.

        if wait_counter = std_logic_vector(to_unsigned(WAIT_CYCLES,
           WAIT_CYCLES_BITS)) then
          -- The signal has been stable for the specified number of
             cycles, and we can
          -- update the output signal.
          current_state <= input_state;
        else
          -- The signal has been stable for one more cycle.
          wait_counter <= wait_counter + 1;
        end if;

      else
        -- The signal has changed.

        -- Reset the wait counter.
        wait_counter <= std_logic_vector(to_unsigned(0,
           WAIT_CYCLES_BITS));

        -- Remember the current state.
        previous_state <= input_state;
      end if;
    end if;
  end process;

end keyboard_debounce_impl;
```

Listing G.11: testbench.vhd

```
-- Based on a file containing the following header:
--
--
   ********************************************************************

-- Project    : AES128                                              *
--

   *
-- Block Name : aes_fips_tester.vhd                                 *
--

   *
```

```
-- Author     : Hemanth Satyanarayana
--
-- Email      : hemanth@opencores.org
--
-- Description: Test bench module to test the aes implemntation
--              for KAT based tests.
--
-- Revision History
--
-- |   Name    |    Date     | Version |          Revision details
-- | Hemanth   | 15-Dec-2004 | 1.1.1.1 |            Uploaded
--
--
-- Refer FIPS-KAT Document for details
--
-- Copyright (C) 2004 Author
--
-- This source file may be used and distributed without
```



```
-- restriction provided that this copyright statement is not
-- removed from the file and that any derivative work contains
-- the original copyright notice and the associated disclaimer.
--
-- This source file is free software; you can redistribute it
-- and/or modify it under the terms of the GNU Lesser General
-- Public License as published by the Free Software Foundation;
-- either version 2.1 of the License, or (at your option) any
-- later version.
--
-- This source is distributed in the hope that it will be
-- useful, but WITHOUT ANY WARRANTY; without even the implied
-- warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
-- PURPOSE.  See the GNU Lesser General Public License for more
-- details.
--
-- You should have received a copy of the GNU Lesser General
-- Public License along with this source; if not, download it
-- from http://www.opencores.org/lgpl.shtml
--
--------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.math_real.all;
```

```
use std.textio.all;
use ieee.std_logic_textio.all;
use ieee.numeric_std;

library std_developerskit;
use std_developerskit.std_iopak.all;

entity testbench is end testbench;

architecture behavioral of testbench is
    component clock_divider is
        generic (width : integer);
        port (
            clock_in : in std_logic;
            value : out std_logic_vector(width - 1 downto 0)
        );
    end component;

    component tdes_control is
        port (
            clock_32MHz : std_logic;
            clock_8MHz : std_logic;

            mem_address  : in std_logic_vector(15 downto 0);
            mem_read     : in std_logic;
            mem_data_out : out std_logic_vector(7 downto 0);
            mem_write    : in std_logic;
            mem_data_in  : in std_logic_vector(7 downto 0);

            interrupt    : out std_logic
        );
    end component;

signal clock_tb: std_logic:='0';
signal reset_tb: std_logic:='0';
signal start_tb: std_logic:='0';
signal load_tb: std_logic:='0';
signal done_tb: std_logic;
--##############################
signal mode_tb: std_logic:='1'; -- 1-> encode; 0-> decode
--##############################
signal data_in_tb: std_logic_vector(63 downto 0):=X"0000000000000000";
signal data_out_tb: std_logic_vector(127 downto 0);
signal key_tb: std_logic_vector(63 downto 0):=X"0000000000000000";

signal mem_address  : std_logic_vector(15 downto 0);
signal mem_read     : std_logic := '0';
signal mem_data_out : std_logic_vector(7 downto 0);
signal mem_write    : std_logic := '0';
signal mem_data_in  : std_logic_vector(7 downto 0);
signal interrupt    : std_logic := '0';
```



```
signal key          : std_logic_vector(0 to 191);
signal data         : std_logic_vector(0 to 63);
signal result       : std_logic_vector(0 to 63);

signal clocks: std_logic_vector(1 downto 0);
begin

clock_tb <= not clock_tb after 50 ns;
reset_tb <= '1','0' after 150 ns;

    TDES_CONTROL_INST: tdes_control
    port map (
        clock_32MHz => clock_tb,
        clock_8MHz => clocks(1),

        mem_address => mem_address,
        mem_read => mem_read,
        mem_data_out => mem_data_out,
        mem_write => mem_write,
        mem_data_in => mem_data_in,

        interrupt => interrupt
    );

    CLOCK_DIVIDER_INST: clock_divider
    generic map ( width => 2) port map (
        clock_in => clock_tb,
        value => clocks
    );

process
    file infile1 : text open read_mode is "ecb_tbl.txt";
    file outfile1: text open write_mode is "ecb_tb_results.txt";
    file infile2 : text open read_mode is "ecb_vk.txt";
    file outfile2: text open write_mode is "ecb_vk_results.txt";
    file infile3 : text open read_mode is "ecb_vt.txt";
    file outfile3: text open write_mode is "ecb_vt_results.txt";
    variable inline       : line;
    variable outline      : line;
    variable itr_numline  : string(1 to 2);
    variable key_line     : string(1 to 4);
    variable pt_line      : string(1 to 3);
    variable ct_line      : string(1 to 3);
    variable iteration_num: integer;
    variable hex_key_str  : string(1 to 48);
    variable pt_str       : string(1 to 16);
    variable ct_str       : string(1 to 16);
    variable exp_cipher   : std_logic_vector(127 downto 0);
    variable i : integer;
    variable mem_address_var : std_logic_vector(15 downto 0);

    procedure load_key(key_value : in std_logic_vector) is
```

```
        variable addr : integer;
    begin
        addr := 16#EC00#;
        for i in 0 to 23 loop
            wait until rising_edge(clock_tb);
            mem_address <= conv_std_logic_vector(addr, 16);
            addr := addr + 1;
            mem_data_in <= key_value(8 * i to 8 * i + 7);
            mem_write <= '1';
            wait until rising_edge(clock_tb);
            mem_write <= '0';
        end loop;
    end load_key;

    procedure load_data(data_value : in std_logic_vector) is
        variable addr : integer;
    begin
        addr := 16#ED00#;
        for i in 0 to 7 loop
            wait until rising_edge(clock_tb);
            mem_address <= conv_std_logic_vector(addr, 16);
            addr := addr + 1;
            mem_data_in <= data_value(8 * i to 8 * i + 7);
            mem_write <= '1';
            wait until rising_edge(clock_tb);
            mem_write <= '0';
        end loop;
    end load_data;

    procedure get_result(signal result_value : out std_logic_vector) is
        variable addr : integer;
    begin
        addr := 16#EE00#;
        for i in 0 to 7 loop
            wait until rising_edge(clock_tb);
            mem_address <= conv_std_logic_vector(addr, 16);
            addr := addr + 1;
            mem_read <= '1';
            wait until rising_edge(clock_tb);
            wait until rising_edge(clock_tb);
            result_value(8 * i to 8 * i + 7) <= mem_data_out;
            mem_read <= '0';
        end loop;
        wait until rising_edge(clock_tb);
    end get_result;

    procedure encrypt is
    begin
        wait until rising_edge(clock_tb);
        mem_address <= x"EF01";
        mem_data_in <= x"00";
        mem_write <= '1';
```



```
        wait until rising_edge(clock_tb);
        mem_write <= '0';
        wait until interrupt = '1';
    end encrypt;

    procedure decrypt is
    begin
        wait until rising_edge(clock_tb);
        mem_address <= x"EF02";
        mem_data_in <= x"00";
        mem_write <= '1';
        wait until rising_edge(clock_tb);
        mem_write <= '0';
        wait until interrupt = '1';
    end decrypt;

begin
    hex_key_str := "33333333333333333333333333CCCCCCCCCCCCCCCCCCCCCCCC";
    pt_str := "8000000000000000";
    key <= to_StdLogicVector(From_HexString(hex_key_str(1 to 48)));
    data <= to_StdLogicVector(From_HexString(pt_str(1 to 16)));
    wait for 1 ns;
    wait until reset_tb = '0';

    load_key(key);

    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    writeline(output, outline);

    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    write(outline, lf);
    writeline(output, outline);


    data <= x"4000000000000000";
    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
```

```
        write(outline, ' ');
    end loop;
    writeline(output, outline);

    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    write(outline, lf);
    writeline(output, outline);


    data <= x"2000000000000000";
    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    writeline(output, outline);

    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    write(outline, lf);
    writeline(output, outline);


    data <= x"1000000000000000";
    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    writeline(output, outline);

    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
```



```
    end loop;
    write(outline, lf);
    writeline(output, outline);


    wait until rising_edge(clock_tb);
    mem_address <= x"EF04";
    mem_data_in <= x"00";
    mem_write <= '1';
    wait until rising_edge(clock_tb);
    mem_write <= '0';
    wait until rising_edge(clock_tb);

    wait until rising_edge(clock_tb);
    mem_address <= x"EF05";
    mem_data_in <= x"00";
    mem_write <= '1';
    wait until rising_edge(clock_tb);
    mem_write <= '0';
    wait until rising_edge(clock_tb);


    data <= x"0800000000000000";
    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    writeline(output, outline);

    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    write(outline, lf);
    writeline(output, outline);


    data <= x"0400000000000000";
    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    writeline(output, outline);
```

```
    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    write(outline, lf);
    writeline(output, outline);


    data <= x"0200000000000000";
    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    writeline(output, outline);

    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    write(outline, lf);
    writeline(output, outline);


    data <= x"0100000000000000";
    load_data(data);
    encrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    writeline(output, outline);

    load_data(result);
    decrypt;
    get_result(result);
    for i in 0 to 7 loop
        write(outline, result(8 * i to 8 * i + 7));
        write(outline, ' ');
    end loop;
    write(outline, lf);
    writeline(output, outline);
```



```
end process;


end behavioral;
```

Listing G.12: add_key.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity add_key is
port(
            x0_in: in std_logic_vector(0 to 5);
            x1_in: in std_logic_vector(0 to 5);
            x2_in: in std_logic_vector(0 to 5);
            x3_in: in std_logic_vector(0 to 5);
            x4_in: in std_logic_vector(0 to 5);
            x5_in: in std_logic_vector(0 to 5);
            x6_in: in std_logic_vector(0 to 5);
            x7_in: in std_logic_vector(0 to 5);
            key: in std_logic_vector(0 to 47);
            x0_out: out std_logic_vector(5 downto 0);
            x1_out: out std_logic_vector(5 downto 0);
            x2_out: out std_logic_vector(5 downto 0);
            x3_out: out std_logic_vector(5 downto 0);
            x4_out: out std_logic_vector(5 downto 0);
            x5_out: out std_logic_vector(5 downto 0);
            x6_out: out std_logic_vector(5 downto 0);
            x7_out: out std_logic_vector(5 downto 0)
    );
end add_key;


architecture Behavioral of add_key is
begin

    x0_out <= x0_in xor key(0 to 5);
    x1_out <= x1_in xor key(6 to 11);
    x2_out <= x2_in xor key(12 to 17);
    x3_out <= x3_in xor key(18 to 23);
    x4_out <= x4_in xor key(24 to 29);
    x5_out <= x5_in xor key(30 to 35);
    x6_out <= x6_in xor key(36 to 41);
    x7_out <= x7_in xor key(42 to 47);

end Behavioral;
```

Listing G.13: add_left.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity add_left is
port(
            x_in: in std_logic_vector(0 to 31);
            left_in: in std_logic_vector(0 to 31);
            x_out: out std_logic_vector(0 to 31)
    );
end add_left;

architecture Behavioral of add_left is

begin

    x_out <= x_in xor left_in;

end Behavioral;
```

Listing G.14: block_top.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity block_top is
port(
        --
        -- input into top level block
        --
        L_in: in std_logic_vector(0 to 31);
        R_in: in std_logic_vector(0 to 31);

        --
        -- output from top level block
       --
        L_out: out std_logic_vector(0 to 31);
        R_out: out std_logic_vector(0 to 31);

        --
        -- expanded key from key block
        --
        round_key_des: in std_logic_vector(0 to 47) -- current round
           key
```



```
    );
end block_top;

architecture Behavioral of block_top is
--
-- DECLARATION OF MODULES IN THE BLOCK_TOP
--

--
--  E _ E X P A N S I O N _ F U N C T I O N
--
component e_expansion_function
    port(
            x_in: in std_logic_vector(0 to 31);
            block0_out: out std_logic_vector(0 to 5);
            block1_out: out std_logic_vector(0 to 5);
            block2_out: out std_logic_vector(0 to 5);
            block3_out: out std_logic_vector(0 to 5);
            block4_out: out std_logic_vector(0 to 5);
            block5_out: out std_logic_vector(0 to 5);
            block6_out: out std_logic_vector(0 to 5);
            block7_out: out std_logic_vector(0 to 5)
    );
end component;

--
--  A D D _ K E Y
--
component add_key
    port(
            x0_in: in std_logic_vector(0 to 5);
            x1_in: in std_logic_vector(0 to 5);
            x2_in: in std_logic_vector(0 to 5);
            x3_in: in std_logic_vector(0 to 5);
            x4_in: in std_logic_vector(0 to 5);
            x5_in: in std_logic_vector(0 to 5);
            x6_in: in std_logic_vector(0 to 5);
            x7_in: in std_logic_vector(0 to 5);
            key: in std_logic_vector(0 to 47);
            x0_out: out std_logic_vector(5 downto 0);
            x1_out: out std_logic_vector(5 downto 0);
            x2_out: out std_logic_vector(5 downto 0);
            x3_out: out std_logic_vector(5 downto 0);
            x4_out: out std_logic_vector(5 downto 0);
            x5_out: out std_logic_vector(5 downto 0);
            x6_out: out std_logic_vector(5 downto 0);
            x7_out: out std_logic_vector(5 downto 0)
    );
end component;

--
--  S _ B O X
```

```
--
component s_box
    port(
            block0_in: in std_logic_vector(5 downto 0);
            block1_in: in std_logic_vector(5 downto 0);
            block2_in: in std_logic_vector(5 downto 0);
            block3_in: in std_logic_vector(5 downto 0);
            block4_in: in std_logic_vector(5 downto 0);
            block5_in: in std_logic_vector(5 downto 0);
            block6_in: in std_logic_vector(5 downto 0);
            block7_in: in std_logic_vector(5 downto 0);
            x0_out: out std_logic_vector(3 downto 0);
            x1_out: out std_logic_vector(3 downto 0);
            x2_out: out std_logic_vector(3 downto 0);
            x3_out: out std_logic_vector(3 downto 0);
            x4_out: out std_logic_vector(3 downto 0);
            x5_out: out std_logic_vector(3 downto 0);
            x6_out: out std_logic_vector(3 downto 0);
            x7_out: out std_logic_vector(3 downto 0)
    );
end component;

--
--  P _ B O X
--
component p_box
    port(
            x0_in: in std_logic_vector(3 downto 0);
            x1_in: in std_logic_vector(3 downto 0);
            x2_in: in std_logic_vector(3 downto 0);
            x3_in: in std_logic_vector(3 downto 0);
            x4_in: in std_logic_vector(3 downto 0);
            x5_in: in std_logic_vector(3 downto 0);
            x6_in: in std_logic_vector(3 downto 0);
            x7_in: in std_logic_vector(3 downto 0);
            x_out: out std_logic_vector(0 to 31)
    );
end component;

--
--  A D D _ L E F T
--
component add_left
    port(
            x_in: in std_logic_vector(0 to 31);
            left_in: in std_logic_vector(0 to 31);
            x_out: out std_logic_vector(0 to 31)
    );
end component;


--
```



```
-- Signals that connects modules within block_top
--
signal a0, a1, a2, a3, a4, a5, a6, a7: std_logic_vector(0 to 5);
signal b0, b1, b2, b3, b4, b5, b6, b7: std_logic_vector(5 downto 0);
signal c0, c1, c2, c3, c4, c5, c6, c7: std_logic_vector(3 downto 0);
signal d0: std_logic_vector(0 to 31);
signal R_out_internal: std_logic_vector(0 to 31);

begin

L_out <= R_in;
R_out <= R_out_internal;

--
-- INSTANTIATION OF E_EXPANSIONFUNCTION
--
E_EXPANSIONFUNCTION : e_expansion_function
port map (
            x_in => R_in,
            block0_out => a0,
            block1_out => a1,
            block2_out => a2,
            block3_out => a3,
            block4_out => a4,
            block5_out => a5,
            block6_out => a6,
            block7_out => a7
);

--
-- INSTANTIATION OF ADDKEY
--
ADDKEY : add_key
port map (
            x0_in => a0,
            x1_in => a1,
            x2_in => a2,
            x3_in => a3,
            x4_in => a4,
            x5_in => a5,
            x6_in => a6,
            x7_in => a7,
            key => round_key_des,
            x0_out => b0,
            x1_out => b1,
            x2_out => b2,
            x3_out => b3,
            x4_out => b4,
            x5_out => b5,
            x6_out => b6,
            x7_out => b7
);
```

```
--
-- INSTANTIATION OF SBOX
--
SBOX : s_box
port map (
            block0_in => b0,
            block1_in => b1,
            block2_in => b2,
            block3_in => b3,
            block4_in => b4,
            block5_in => b5,
            block6_in => b6,
            block7_in => b7,
            x0_out =>  c0,
            x1_out =>  c1,
            x2_out =>  c2,
            x3_out =>  c3,
            x4_out =>  c4,
            x5_out =>  c5,
            x6_out =>  c6,
            x7_out =>  c7
);

--
-- INSTANTIATION OF PBOX
--
PBOX : p_box
port map (
            x0_in => c0,
            x1_in => c1,
            x2_in => c2,
            x3_in => c3,
            x4_in => c4,
            x5_in => c5,
            x6_in => c6,
            x7_in => c7,
            x_out => d0
);


--
-- INSTANTIATION OF ADDLEFT
--
ADDLEFT : add_left
port map (
            x_in => d0,
            left_in => L_in,
            x_out => R_out_internal
);
```



```
end Behavioral;
```

Listing G.15: des_cipher_top.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity des_cipher_top is
port(
        --
        -- Core Interface
        --
        key_in:             in std_logic_vector(0 to 63);       --
           input for key
        --ldkey:                in std_logic;
              -- signal for loading key
        function_select:    in  std_logic;                          --
           function select: '1' = encryption, '0' = decryption

        data_in:                in std_logic_vector(0 to 63);       --
           input for data

        data_out:           out std_logic_vector(0 to 63);  -- output
           for data

        lddata:             in  std_logic;                      -- data
            strobe (active high)
        core_busy:          out std_logic;                      --
           active high when encrypting/decryption data
        des_out_rdy:        out std_logic;                      --
           active high when encryption/decryption of data is done

        reset:              in std_logic;                           --
           active high
        clock:              in std_logic                            --
           master clock

    );
end des_cipher_top;

architecture Behavioral of des_cipher_top is

--
--
--
component key_schedule is
port (
        -- Signals for loading key from external device
```

```
        key_in:         in std_logic_vector(0 to 63);       -- input
           for key

        -- signals for communication with des top
        KeySelect:      in std_logic_vector(3 downto 0);    -- selector
            for key
        key_out:        out std_logic_vector(0 to 47);  -- expaned key
           (depends on selector)
        key_ready:      out std_logic;                          --
           signal for the core that key has been expanded

        reset: in std_logic;                                    --
           active high
        clock: in std_logic                                     --
           master clock
        );
end component;

component des_top is
port (
        -- Main Data
        key_round_in:   in  std_logic_vector(0 to 47);
        data_in:            in  std_logic_vector(0 to 63);
        data_out:       out     std_logic_vector(0 to 63);

        -- Signals for communication with des
        KeySelect:      inout std_logic_vector(3 downto 0); -- selector
            for key
        key_ready:      in std_logic;                               --
           signal for aes that key has been expanded
        data_ready:     in std_logic;                               --
           signal for aes that key has been expanded
        func_select:    in std_logic;

        des_out_rdy:    out std_logic;
        core_busy:      out std_logic;

        reset:          in std_logic;
        clock:          in std_logic                            --
           master clock
        );
end component;

signal key_select_internal: std_logic_vector(3 downto 0);
signal key_round_internal: std_logic_vector(0 to 47);
signal key_ready_internal: std_logic;
signal data_in_internal: std_logic_vector(0 to 63);
signal data_ready_internal: std_logic;

begin

process (clock)
```



```
begin

if rising_edge(clock) then

        if lddata = '1' then

            -- capute data from the bus
            data_in_internal        <= data_in; -- register data from
               the bus
            data_ready_internal <= '1';     -- data has been loaded:
               continue with encryptio/decryption

        else

            data_ready_internal <= '0';     -- data is not loaded: wait
                for data

        end if;

end if;
end process;

--
-- KEY EXPANDER AND DES CORE instantiation
--
KEYSCHEDULE: key_schedule
port map (

        KeySelect   => key_select_internal,
        key_in      => key_in,

        key_out         => key_round_internal,
        key_ready   => key_ready_internal,

        reset       => reset,
        clock       => clock
);

DESTOP: des_top
port map (

        key_round_in    => key_round_internal,

        data_in         => data_in_internal,

        key_ready       => key_ready_internal,
        data_ready      => data_ready_internal,

        KeySelect       => key_select_internal,

        func_select     => function_select,
```

```
        data_out        => data_out,
        core_busy       => core_busy,
        des_out_rdy     => des_out_rdy,

        reset           => reset,
        clock           => clock
);

end Behavioral;
```

Listing G.16: des_top.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity des_top is
port (

        -- input/output core signals
        key_round_in:   in  std_logic_vector(0 to 47);
        data_in:            in  std_logic_vector(0 to 63);
        data_out:       out     std_logic_vector(0 to 63);

        -- signals for communication with key expander module
        KeySelect:      inout std_logic_vector(3 downto 0);     --
           selector for key
        key_ready:      in std_logic;                           -- active
           high when key is ready
        data_ready:     in std_logic;                           -- active
           high when data is ready
        func_select:    in std_logic;                           --
           encryption/decryption flag

        des_out_rdy: out std_logic;                     -- active high
           when decrypted/encrypted data are ready
        core_busy: out std_logic;                               -- active
           high when core is in process of encryption

        reset: in std_logic;                                    -- master
           reset
        clock: in std_logic                                     -- master
           clock
        );
end des_top;

architecture Behavioral of des_top is

--
```



```
-- BLOCK_TOP entity performs encryption/deccryption operation. It uses
   expaned key for that process
--
component block_top is
port(
        L_in: in std_logic_vector(0 to 31);             -- left
           permuted input
        R_in: in std_logic_vector(0 to 31);             -- right
           permuted input

        L_out: out std_logic_vector(0 to 31);           -- left
           permuted output
        R_out: out std_logic_vector(0 to 31);           -- right
           permuted output

        round_key_des: in std_logic_vector(0 to 47) -- current round
           key

    );
end component;

--
-- Internal DES_TOP signals
--
signal L_in_internal, R_in_internal:    std_logic_vector(0 to 31);
signal L_out_internal, R_out_internal: std_logic_vector(0 to 31);
type statetype is (WaitKey, WaitData, InitialRound, RepeatRound,
   FinalRound);
signal nextstate: statetype;
signal RoundCounter:                        std_logic_vector(3 downto
   0);

begin

--
-- Finite state machine
--
process (clock)
begin
    if rising_edge(clock)  then
        if reset = '1' then
            --
            -- Reset all signal to inital values
            --
            nextstate               <= WaitKey;
            RoundCounter            <= "0000";
            core_busy               <= '0';             -- core is in
               reset state: not busy
            des_out_rdy             <= '0';             -- output data
               is not ready

        else
```

```
        case nextstate is

            --
            -- WaitKey: wait for key to be expanded
            --
            when WaitKey =>

                -- wait until key has been expanded
                if key_ready = '0' then
                    nextstate   <= WaitKey;
                else
                    nextstate   <= WaitData;
                end if;

                core_busy               <= '0';             -- core
                    waits for the key: not busy
                des_out_rdy             <= '0';             --
                   output data is not ready

            --
            -- WaitData: waits for data until it is ready
            --
            when WaitData =>

                -- wait for data to be loaded in input registers
                if (data_ready = '0') then

                    nextstate               <= WaitData;

                else
                    core_busy               <= '1';             --
                       core is processing = busy

                    L_in_internal <=    data_in(57) & data_in(49) &
                        data_in(41) & data_in(33) & data_in(25) &
                       data_in(17) &
                                        data_in(9) & data_in(1)
                                            & data_in(59) &
                                           data_in(51) &
                                           data_in(43) &
                                           data_in(35) &
                                        data_in(27) & data_in
                                           (19) & data_in(11) &
                                            data_in(3) &
                                           data_in(61) &
                                           data_in(53) &
                                        data_in(45) & data_in
                                           (37) & data_in(29) &
                                            data_in(21) &
                                           data_in(13) &
                                           data_in(5) &
```



```
                                           data_in(63) & data_in
                                              (55) & data_in(47) &
                                               data_in(39) &
                                              data_in(31) &
                                              data_in(23) &
                                           data_in(15) & data_in
                                              (7);

                    R_in_internal <=    data_in(56) & data_in(48) &
                        data_in(40) & data_in(32) & data_in(24) &
                       data_in(16) &
                                           data_in(8) & data_in(0)
                                               & data_in(58) &
                                              data_in(50) &
                                              data_in(42) &
                                              data_in(34) &
                                           data_in(26) & data_in
                                              (18) & data_in(10) &
                                               data_in(2) &
                                              data_in(60) &
                                              data_in(52) &
                                           data_in(44) & data_in
                                              (36) & data_in(28) &
                                               data_in(20) &
                                              data_in(12) &
                                              data_in(4) &
                                           data_in(62) & data_in
                                              (54) & data_in(46) &
                                               data_in(38) &
                                              data_in(30) &
                                              data_in(22) &
                                           data_in(14) & data_in
                                              (6);

                    nextstate       <= InitialRound;

                    -- function select (decrypting/encrypting) will
                        determine key selection
                    if func_select = '1' then
                        KeySelect   <= "0000";
                    else
                        KeySelect   <= "1111";
                    end if;

                end if;

            --
            -- Initial State where input is equal to a block that
               we need to encode
            --
            when InitialRound =>
```

```
                L_in_internal   <= L_out_internal;
                R_in_internal   <= R_out_internal;

                -- fuction select determines direction of key
                   selection
                if func_select = '1' then
                    KeySelect   <= KeySelect + '1';
                else
                    KeySelect   <= KeySelect - '1';
                end if;

                nextstate       <= RepeatRound;

            --
            -- Repeat Section, where input is output from prevous
               state
            --
            when RepeatRound =>

                L_in_internal <= L_out_internal;
                R_in_internal <= R_out_internal;

                -- fuction select determines direction of key
                   selection
                if func_select = '1' then
                    KeySelect <= KeySelect + '1';
                else
                    KeySelect <= KeySelect - '1';
                end if;

                RoundCounter <= RoundCounter + '1';

                -- if finished with all rounds, go to the final
                    round
                if RoundCounter = x"E" then

                    -- perform inverse initial permutation
                    data_out    <=  L_out_internal(7) &
                       R_out_internal(7) & L_out_internal(15) &
                        R_out_internal(15) &
                                    L_out_internal(23) &
                                       R_out_internal(23) &
                                       L_out_internal(31) &
                                       R_out_internal(31) &
                                    L_out_internal(6) &
                                       R_out_internal(6) &
                                       L_out_internal(14) &
                                       R_out_internal(14) &
                                    L_out_internal(22) &
                                       R_out_internal(22) &
                                       L_out_internal(30) &
                                       R_out_internal(30) &
```



```
                                    L_out_internal(5) &
                                       R_out_internal(5) &
                                       L_out_internal(13) &
                                       R_out_internal(13) &
                                    L_out_internal(21) &
                                       R_out_internal(21) &
                                       L_out_internal(29) &
                                       R_out_internal(29) &
                                    L_out_internal(4) &
                                       R_out_internal(4) &
                                       L_out_internal(12) &
                                       R_out_internal(12) &
                                    L_out_internal(20) &
                                       R_out_internal(20) &
                                       L_out_internal(28) &
                                       R_out_internal(28) &
                                    L_out_internal(3) &
                                       R_out_internal(3) &
                                       L_out_internal(11) &
                                       R_out_internal(11) &
                                    L_out_internal(19) &
                                       R_out_internal(19) &
                                       L_out_internal(27) &
                                       R_out_internal(27) &
                                    L_out_internal(2) &
                                       R_out_internal(2) &
                                       L_out_internal(10) &
                                       R_out_internal(10) &
                                    L_out_internal(18) &
                                       R_out_internal(18) &
                                       L_out_internal(26) &
                                       R_out_internal(26) &
                                    L_out_internal(1) &
                                       R_out_internal(1) &
                                       L_out_internal(9) &
                                       R_out_internal(9) &
                                    L_out_internal(17) &
                                       R_out_internal(17) &
                                       L_out_internal(25) &
                                       R_out_internal(25) &
                                    L_out_internal(0) &
                                       R_out_internal(0) &
                                       L_out_internal(8) &
                                       R_out_internal(8) &
                                    L_out_internal(16) &
                                       R_out_internal(16) &
                                       L_out_internal(24) &
                                       R_out_internal(24);

                core_busy               <= '0';     -- core
                    is not busy
```

```
                        des_out_rdy             <= '1';     --
                            output data is ready
                        nextstate               <= FinalRound;

                    else

                        -- Continue with regular rounds
                        nextstate <= RepeatRound;

                    end if;

                --
                -- Last round
                --
                when FinalRound =>

                    RoundCounter            <= "0000";
                    nextstate               <= WaitKey;
                    des_out_rdy             <= '0';     -- deselect out
                        data ready signal

                when others =>
                    -- should never happen
            end case;
        end if;
    end if;
end process;

--
-- Instantations
--
BLOCKTOP: block_top
port map (
        L_in                => L_in_internal,
        R_in                => R_in_internal,

        round_key_des   => key_round_in,

        L_out           => L_out_internal,
        R_out           =>  R_out_internal

);

end Behavioral;
```

Listing G.17: e_expansion_function.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```



```
entity e_expansion_function is
port(
            x_in: in std_logic_vector(0 to 31);
            block0_out: out std_logic_vector(0 to 5);
            block1_out: out std_logic_vector(0 to 5);
            block2_out: out std_logic_vector(0 to 5);
            block3_out: out std_logic_vector(0 to 5);
            block4_out: out std_logic_vector(0 to 5);
            block5_out: out std_logic_vector(0 to 5);
            block6_out: out std_logic_vector(0 to 5);
            block7_out: out std_logic_vector(0 to 5)
    );
end e_expansion_function;

architecture Behavioral of e_expansion_function is

begin

        block0_out <= x_in(31) & x_in(0) & x_in(1) & x_in(2) & x_in(3)
            & x_in(4);
        block1_out <= x_in(3) & x_in(4) & x_in(5) & x_in(6) & x_in(7) &
            x_in(8);
        block2_out <= x_in(7) & x_in(8) & x_in(9) & x_in(10) & x_in(11)
            & x_in(12);
        block3_out <= x_in(11) & x_in(12) & x_in(13) & x_in(14) & x_in
            (15) & x_in(16);
        block4_out <= x_in(15) & x_in(16) & x_in(17) & x_in(18) & x_in
            (19) & x_in(20);
        block5_out <= x_in(19) & x_in(20) & x_in(21) & x_in(22) & x_in
            (23) & x_in(24);
        block6_out <= x_in(23) & x_in(24) & x_in(25) & x_in(26) & x_in
            (27) & x_in(28);
        block7_out <= x_in(27) & x_in(28) & x_in(29) & x_in(30) & x_in
            (31) & x_in(0);

end Behavioral;
```

Listing G.18: key_schedule.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity key_schedule is
port (

        key_in:         in std_logic_vector(0 to 63);       -- key to
            be expanded
```

```
        -- interface signals for communication with DES
        KeySelect:      in std_logic_vector(3 downto 0);    -- selector
            for key
        key_out:        out std_logic_vector(0 to 47);  -- expaned key
           output
        key_ready:      out std_logic;                          --
           signal for DES that key has been expanded

        reset:          in std_logic;                           --
           reset
        clock:          in std_logic                            --
           master clock
        );
end key_schedule;

architecture Behavioral of key_schedule is

--
-- Storage for expanded key
--
signal K16, K1:  std_logic_vector(0 to 47);
signal K2,  K3:  std_logic_vector(0 to 47);
signal K4,  K5:  std_logic_vector(0 to 47);
signal K6,  K7:  std_logic_vector(0 to 47);
signal K8,  K9:  std_logic_vector(0 to 47);
signal K10, K11: std_logic_vector(0 to 47);
signal K12, K13: std_logic_vector(0 to 47);
signal K14, K15: std_logic_vector(0 to 47);

begin


--
-- Selector for expaned key
--
key_out <=  K1  when    KeySelect = x"0" else
                K2  when    KeySelect = x"1" else
                K3  when    KeySelect = x"2" else
                K4  when    KeySelect = x"3" else
                K5  when    KeySelect = x"4" else
                K6  when    KeySelect = x"5" else
                K7  when    KeySelect = x"6" else
                K8  when    KeySelect = x"7" else
                K9  when    KeySelect = x"8" else
                K10     when    KeySelect = x"9" else
                K11     when    KeySelect = x"A" else
                K12     when    KeySelect = x"B" else
                K13     when    KeySelect = x"C" else
                K14     when    KeySelect = x"D" else
                K15     when    KeySelect = x"E" else
                K16;
```



```
process (clock)
begin

--
-- input key will expaned and stored after rising edge of the first
   clock after mater reset.
-- the keys are captured at a triple-des level
--
if rising_edge(clock) then

    if reset = '1' then
        key_ready <= '0';
    else

        --
        -- key expansion from the input key
        --
        K1 <= key_in(9) & key_in(50) & key_in(33) & key_in(59) & key_in
            (48) & key_in(16) & key_in(32) & key_in(56) &
                 key_in(1) & key_in(8) & key_in(18) & key_in(41) &
                    key_in(2) & key_in(34) & key_in(25) & key_in(24) &
                 key_in(43) & key_in(57) & key_in(58) & key_in(0) &
                    key_in(35) & key_in(26) & key_in(17) & key_in(40) &
                 key_in(21) & key_in(27) & key_in(38) & key_in(53) &
                    key_in(36) & key_in(3) & key_in(46) & key_in(29) &
                 key_in(4) & key_in(52) & key_in(22) & key_in(28) &
                    key_in(60) & key_in(20) & key_in(37) & key_in(62) &
                 key_in(14) & key_in(19) & key_in(44) & key_in(13) &
                    key_in(12) & key_in(61) & key_in(54) & key_in(30);

        K2 <= key_in(1) & key_in(42) & key_in(25) & key_in(51) & key_in
            (40) & key_in(8) & key_in(24) & key_in(48) &
                 key_in(58) & key_in(0) & key_in(10) & key_in(33) &
                    key_in(59) & key_in(26) & key_in(17) & key_in(16) &
                 key_in(35) & key_in(49) & key_in(50) & key_in(57) &
                    key_in(56) & key_in(18) & key_in(9) & key_in(32) &
                 key_in(13) & key_in(19) & key_in(30) & key_in(45) &
                    key_in(28) & key_in(62) & key_in(38) & key_in(21) &
                 key_in(27) & key_in(44) & key_in(14) & key_in(20) &
                    key_in(52) & key_in(12) & key_in(29) & key_in(54) &
                 key_in(6) & key_in(11) & key_in(36) & key_in(5) &
                    key_in(4) & key_in(53) & key_in(46) & key_in(22);

        K3 <= key_in(50) & key_in(26) & key_in(9) & key_in(35) & key_in
            (24) & key_in(57) & key_in(8) & key_in(32) &
                 key_in(42) & key_in(49) & key_in(59) & key_in(17) &
                    key_in(43) & key_in(10) & key_in(1) & key_in(0) &
                 key_in(48) & key_in(33) & key_in(34) & key_in(41) &
                    key_in(40) & key_in(2) & key_in(58) & key_in(16) &
                 key_in(60) & key_in(3) & key_in(14) & key_in(29) &
                    key_in(12) & key_in(46) & key_in(22) & key_in(5) &
```

```
            key_in(11) & key_in(28) & key_in(61) & key_in(4) &
               key_in(36) & key_in(27) & key_in(13) & key_in(38) &
            key_in(53) & key_in(62) & key_in(20) & key_in(52) &
               key_in(19) & key_in(37) & key_in(30) & key_in(6);

    K4 <= key_in(34) & key_in(10) & key_in(58) & key_in(48) &
        key_in(8) & key_in(41) & key_in(57) & key_in(16) &
            key_in(26) & key_in(33) & key_in(43) & key_in(1) &
               key_in(56) & key_in(59) & key_in(50) & key_in(49) &
            key_in(32) & key_in(17) & key_in(18) & key_in(25) &
               key_in(24) & key_in(51) & key_in(42) & key_in(0) &
            key_in(44) & key_in(54) & key_in(61) & key_in(13) &
               key_in(27) & key_in(30) & key_in(6) & key_in(52) &
            key_in(62) & key_in(12) & key_in(45) & key_in(19) &
               key_in(20) & key_in(11) & key_in(60) & key_in(22) &
            key_in(37) & key_in(46) & key_in(4) & key_in(36) &
               key_in(3) & key_in(21) & key_in(14) & key_in(53);

    K5 <= key_in(18) & key_in(59) & key_in(42) & key_in(32) &
        key_in(57) & key_in(25) & key_in(41) & key_in(0) &
            key_in(10) & key_in(17) & key_in(56) & key_in(50) &
               key_in(40) & key_in(43) & key_in(34) & key_in(33) &
            key_in(16) & key_in(1) & key_in(2) & key_in(9) & key_in
               (8) & key_in(35) & key_in(26) & key_in(49) &
            key_in(28) & key_in(38) & key_in(45) & key_in(60) &
               key_in(11) & key_in(14) & key_in(53) & key_in(36) &
            key_in(46) & key_in(27) & key_in(29) & key_in(3) &
               key_in(4) & key_in(62) & key_in(44) & key_in(6) &
            key_in(21) & key_in(30) & key_in(19) & key_in(20) &
               key_in(54) & key_in(5) & key_in(61) & key_in(37);

    K6 <= key_in(2) & key_in(43) & key_in(26) & key_in(16) & key_in
        (41) & key_in(9) & key_in(25) & key_in(49) &
            key_in(59) & key_in(1) & key_in(40) & key_in(34) &
               key_in(24) & key_in(56) & key_in(18) & key_in(17) &
            key_in(0) & key_in(50) & key_in(51) & key_in(58) &
               key_in(57) & key_in(48) & key_in(10) & key_in(33) &
            key_in(12) & key_in(22) & key_in(29) & key_in(44) &
               key_in(62) & key_in(61) & key_in(37) & key_in(20) &
            key_in(30) & key_in(11) & key_in(13) & key_in(54) &
               key_in(19) & key_in(46) & key_in(28) & key_in(53) &
            key_in(5) & key_in(14) & key_in(3) & key_in(4) & key_in
               (38) & key_in(52) & key_in(45) & key_in(21);

    K7 <= key_in(51) & key_in(56) & key_in(10) & key_in(0) & key_in
        (25) & key_in(58) & key_in(9) & key_in(33) &
            key_in(43) & key_in(50) & key_in(24) & key_in(18) &
               key_in(8) & key_in(40) & key_in(2) & key_in(1) &
            key_in(49) & key_in(34) & key_in(35) & key_in(42) &
               key_in(41) & key_in(32) & key_in(59) & key_in(17) &
            key_in(27) & key_in(6) & key_in(13) & key_in(28) &
               key_in(46) & key_in(45) & key_in(21) & key_in(4) &
```



```
            key_in(14) & key_in(62) & key_in(60) & key_in(38) &
               key_in(3) & key_in(30) & key_in(12) & key_in(37) &
            key_in(52) & key_in(61) & key_in(54) & key_in(19) &
               key_in(22) & key_in(36) & key_in(29) & key_in(5);

    K8 <= key_in(35) & key_in(40) & key_in(59) & key_in(49) &
        key_in(9) & key_in(42) & key_in(58) & key_in(17) &
            key_in(56) & key_in(34) & key_in(8) & key_in(2) &
               key_in(57) & key_in(24) & key_in(51) & key_in(50) &
            key_in(33) & key_in(18) & key_in(48) & key_in(26) &
               key_in(25) & key_in(16) & key_in(43) & key_in(1) &
            key_in(11) & key_in(53) & key_in(60) & key_in(12) &
               key_in(30) & key_in(29) & key_in(5) & key_in(19) &
            key_in(61) & key_in(46) & key_in(44) & key_in(22) &
               key_in(54) & key_in(14) & key_in(27) & key_in(21) &
            key_in(36) & key_in(45) & key_in(38) & key_in(3) &
               key_in(6) & key_in(20) & key_in(13) & key_in(52);

    K9 <= key_in(56) & key_in(32) & key_in(51) & key_in(41) &
        key_in(1) & key_in(34) & key_in(50) & key_in(9) &
            key_in(48) & key_in(26) & key_in(0) & key_in(59) &
               key_in(49) & key_in(16) & key_in(43) & key_in(42) &
            key_in(25) & key_in(10) & key_in(40) & key_in(18) &
               key_in(17) & key_in(8) & key_in(35) & key_in(58) &
            key_in(3) & key_in(45) & key_in(52) & key_in(4) &
               key_in(22) & key_in(21) & key_in(60) & key_in(11) &
            key_in(53) & key_in(38) & key_in(36) & key_in(14) &
               key_in(46) & key_in(6) & key_in(19) & key_in(13) &
            key_in(28) & key_in(37) & key_in(30) & key_in(62) &
               key_in(61) & key_in(12) & key_in(5) & key_in(44);

    K10 <= key_in(40) & key_in(16) & key_in(35) & key_in(25) &
        key_in(50) & key_in(18) & key_in(34) & key_in(58) &
            key_in(32) & key_in(10) & key_in(49) & key_in(43) &
               key_in(33) & key_in(0) & key_in(56) & key_in(26) &
            key_in(9) & key_in(59) & key_in(24) & key_in(2) &
               key_in(1) & key_in(57) & key_in(48) & key_in(42) &
            key_in(54) & key_in(29) & key_in(36) & key_in(19) &
               key_in(6) & key_in(5) & key_in(44) & key_in(62) &
            key_in(37) & key_in(22) & key_in(20) & key_in(61) &
               key_in(30) & key_in(53) & key_in(3) & key_in(60) &
            key_in(12) & key_in(21) & key_in(14) & key_in(46) &
               key_in(45) & key_in(27) & key_in(52) & key_in(28);

    K11 <= key_in(24) & key_in(0) & key_in(48) & key_in(9) & key_in
        (34) & key_in(2) & key_in(18) & key_in(42) &
            key_in(16) & key_in(59) & key_in(33) & key_in(56) &
               key_in(17) & key_in(49) & key_in(40) & key_in(10) &
            key_in(58) & key_in(43) & key_in(8) & key_in(51) &
               key_in(50) & key_in(41) & key_in(32) & key_in(26) &
            key_in(38) & key_in(13) & key_in(20) & key_in(3) &
               key_in(53) & key_in(52) & key_in(28) & key_in(46) &
```

```
            key_in(21) & key_in(6) & key_in(4) & key_in(45) &
               key_in(14) & key_in(37) & key_in(54) & key_in(44) &
            key_in(27) & key_in(5) & key_in(61) & key_in(30) &
               key_in(29) & key_in(11) & key_in(36) & key_in(12);

        K12 <= key_in(8) & key_in(49) & key_in(32) & key_in(58) &
           key_in(18) & key_in(51) & key_in(2) & key_in(26) &
                key_in(0) & key_in(43) & key_in(17) & key_in(40) &
                   key_in(1) & key_in(33) & key_in(24) & key_in(59) &
                key_in(42) & key_in(56) & key_in(57) & key_in(35) &
                   key_in(34) & key_in(25) & key_in(16) & key_in(10) &
                key_in(22) & key_in(60) & key_in(4) & key_in(54) &
                   key_in(37) & key_in(36) & key_in(12) & key_in(30) &
                key_in(5) & key_in(53) & key_in(19) & key_in(29) &
                   key_in(61) & key_in(21) & key_in(38) & key_in(28) &
                key_in(11) & key_in(52) & key_in(45) & key_in(14) &
                   key_in(13) & key_in(62) & key_in(20) & key_in(27);

        K13 <= key_in(57) & key_in(33) & key_in(16) & key_in(42) &
           key_in(2) & key_in(35) & key_in(51) & key_in(10) &
                key_in(49) & key_in(56) & key_in(1) & key_in(24) &
                   key_in(50) & key_in(17) & key_in(8) & key_in(43) &
                key_in(26) & key_in(40) & key_in(41) & key_in(48) &
                   key_in(18) & key_in(9) & key_in(0) & key_in(59) &
                key_in(6) & key_in(44) & key_in(19) & key_in(38) &
                   key_in(21) & key_in(20) & key_in(27) & key_in(14) &
                key_in(52) & key_in(37) & key_in(3) & key_in(13) &
                   key_in(45) & key_in(5) & key_in(22) & key_in(12) &
                key_in(62) & key_in(36) & key_in(29) & key_in(61) &
                   key_in(60) & key_in(46) & key_in(4) & key_in(11);

        K14 <= key_in(41) & key_in(17) & key_in(0) & key_in(26) &
           key_in(51) & key_in(48) & key_in(35) & key_in(59) &
                key_in(33) & key_in(40) & key_in(50) & key_in(8) &
                   key_in(34) & key_in(1) & key_in(57) & key_in(56) &
                key_in(10) & key_in(24) & key_in(25) & key_in(32) &
                   key_in(2) & key_in(58) & key_in(49) & key_in(43) &
                key_in(53) & key_in(28) & key_in(3) & key_in(22) &
                   key_in(5) & key_in(4) & key_in(11) & key_in(61) &
                key_in(36) & key_in(21) & key_in(54) & key_in(60) &
                   key_in(29) & key_in(52) & key_in(6) & key_in(27) &
                key_in(46) & key_in(20) & key_in(13) & key_in(45) &
                   key_in(44) & key_in(30) & key_in(19) & key_in(62);

        K15 <= key_in(25) & key_in(1) & key_in(49) & key_in(10) &
           key_in(35) & key_in(32) & key_in(48) & key_in(43) &
                key_in(17) & key_in(24) & key_in(34) & key_in(57) &
                   key_in(18) & key_in(50) & key_in(41) & key_in(40) &
                key_in(59) & key_in(8) & key_in(9) & key_in(16) &
                   key_in(51) & key_in(42) & key_in(33) & key_in(56) &
                key_in(37) & key_in(12) & key_in(54) & key_in(6) &
                   key_in(52) & key_in(19) & key_in(62) & key_in(45) &
```



```
                key_in(20) & key_in(5) & key_in(38) & key_in(44) &
                   key_in(13) & key_in(36) & key_in(53) & key_in(11) &
                key_in(30) & key_in(4) & key_in(60) & key_in(29) &
                   key_in(28) & key_in(14) & key_in(3) & key_in(46);

        K16 <= key_in(17) & key_in(58) & key_in(41) & key_in(2) &
           key_in(56) & key_in(24) & key_in(40) & key_in(35) &
                key_in(9) & key_in(16) & key_in(26) & key_in(49) &
                   key_in(10) & key_in(42) & key_in(33) & key_in(32) &
                key_in(51) & key_in(0) & key_in(1) & key_in(8) & key_in
                   (43) & key_in(34) & key_in(25) & key_in(48) &
                key_in(29) & key_in(4) & key_in(46) & key_in(61) &
                   key_in(44) & key_in(11) & key_in(54) & key_in(37) &
                key_in(12) & key_in(60) & key_in(30) & key_in(36) &
                   key_in(5) & key_in(28) & key_in(45) & key_in(3) &
                key_in(22) & key_in(27) & key_in(52) & key_in(21) &
                   key_in(20) & key_in(6) & key_in(62) & key_in(38);

        key_ready <= '1';

    end if;

end if;
end process;


end Behavioral;
```

Listing G.19: p_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity p_box is
port(
            x0_in: in std_logic_vector(3 downto 0);
            x1_in: in std_logic_vector(3 downto 0);
            x2_in: in std_logic_vector(3 downto 0);
            x3_in: in std_logic_vector(3 downto 0);
            x4_in: in std_logic_vector(3 downto 0);
            x5_in: in std_logic_vector(3 downto 0);
            x6_in: in std_logic_vector(3 downto 0);
            x7_in: in std_logic_vector(3 downto 0);
            x_out: out std_logic_vector(0 to 31)
    );
end p_box;

architecture Behavioral of p_box is
```

```
signal x_in: std_logic_vector(0 to 31);

begin

        x_in    <= x0_in & x1_in & x2_in & x3_in & x4_in & x5_in &
           x6_in & x7_in;
        x_out <= x_in(15) & x_in(6) & x_in(19) & x_in(20) & x_in(28) &
           x_in(11) &
                    x_in(27) & x_in(16) & x_in(0) & x_in(14) & x_in(22)
                        & x_in(25) &
                    x_in(4) & x_in(17) & x_in(30) & x_in(9) & x_in(1) &
                        x_in(7) &
                    x_in(23) & x_in(13) & x_in(31) & x_in(26) & x_in(2)
                        & x_in(8) &
                    x_in(18) & x_in(12) & x_in(29) & x_in(5) & x_in(21)
                        & x_in(10) &
                    x_in(3) & x_in(24);

end Behavioral;
```

Listing G.20: s1_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s1_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s1_box;

architecture Behavioral of s1_box is

begin

SPO     <= "1110" when A = x"0" else
            "0000" when A = x"1" else
            "0100" when A = x"2" else
            "1111" when A = x"3" else
            "1101" when A = x"4" else
            "0111" when A = x"5" else
            "0001" when A = x"6" else
            "0100" when A = x"7" else
            "0010" when A = x"8" else
            "1110" when A = x"9" else
            "1111" when A = x"A" else
            "0010" when A = x"B" else
            "1011" when A = x"C" else
            "1101" when A = x"D" else
```



```
            "1000" when A = x"E" else
            "0001" when A = x"F" else
            "0011" when A = x"10" else
            "1010" when A = x"11" else
            "1010" when A = x"12" else
            "0110" when A = x"13" else
            "0110" when A = x"14" else
            "1100" when A = x"15" else
            "1100" when A = x"16" else
            "1011" when A = x"17" else
            "0101" when A = x"18" else
            "1001" when A = x"19" else
            "1001" when A = x"1A" else
            "0101" when A = x"1B" else
            "0000" when A = x"1C" else
            "0011" when A = x"1D" else
            "0111" when A = x"1E" else
            "1000" when A = x"1F" else
            "0100" when A = x"20" else
            "1111" when A = x"21" else
            "0001" when A = x"22" else
            "1100" when A = x"23" else
            "1110" when A = x"24" else
            "1000" when A = x"25" else
            "1000" when A = x"26" else
            "0010" when A = x"27" else
            "1101" when A = x"28" else
            "0100" when A = x"29" else
            "0110" when A = x"2A" else
            "1001" when A = x"2B" else
            "0010" when A = x"2C" else
            "0001" when A = x"2D" else
            "1011" when A = x"2E" else
            "0111" when A = x"2F" else
            "1111" when A = x"30" else
            "0101" when A = x"31" else
            "1100" when A = x"32" else
            "1011" when A = x"33" else
            "1001" when A = x"34" else
            "0011" when A = x"35" else
            "0111" when A = x"36" else
            "1110" when A = x"37" else
            "0011" when A = x"38" else
            "1010" when A = x"39" else
            "1010" when A = x"3A" else
            "0000" when A = x"3B" else
            "0101" when A = x"3C" else
            "0110" when A = x"3D" else
            "0000" when A = x"3E" else
            "1101" when A = x"3F" else
            "1101";
```

```
END Behavioral;
```

Listing G.21: s2_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s2_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s2_box;

architecture Behavioral of s2_box is

begin

SPO     <= "1111" when A = x"0" else
           "0011" when A = x"1" else
           "0001" when A = x"2" else
           "1101" when A = x"3" else
           "1000" when A = x"4" else
           "0100" when A = x"5" else
           "1110" when A = x"6" else
           "0111" when A = x"7" else
           "0110" when A = x"8" else
           "1111" when A = x"9" else
           "1011" when A = x"A" else
           "0010" when A = x"B" else
           "0011" when A = x"C" else
           "1000" when A = x"D" else
           "0100" when A = x"E" else
           "1110" when A = x"F" else
           "1001" when A = x"10" else
           "1100" when A = x"11" else
           "0111" when A = x"12" else
           "0000" when A = x"13" else
           "0010" when A = x"14" else
           "0001" when A = x"15" else
           "1101" when A = x"16" else
           "1010" when A = x"17" else
           "1100" when A = x"18" else
           "0110" when A = x"19" else
           "0000" when A = x"1A" else
           "1001" when A = x"1B" else
           "0101" when A = x"1C" else
           "1011" when A = x"1D" else
           "1010" when A = x"1E" else
           "0101" when A = x"1F" else
```



```
           "0000" when A = x"20" else
           "1101" when A = x"21" else
           "1110" when A = x"22" else
           "1000" when A = x"23" else
           "0111" when A = x"24" else
           "1010" when A = x"25" else
           "1011" when A = x"26" else
           "0001" when A = x"27" else
           "1010" when A = x"28" else
           "0011" when A = x"29" else
           "0100" when A = x"2A" else
           "1111" when A = x"2B" else
           "1101" when A = x"2C" else
           "0100" when A = x"2D" else
           "0001" when A = x"2E" else
           "0010" when A = x"2F" else
           "0101" when A = x"30" else
           "1011" when A = x"31" else
           "1000" when A = x"32" else
           "0110" when A = x"33" else
           "1100" when A = x"34" else
           "0111" when A = x"35" else
           "0110" when A = x"36" else
           "1100" when A = x"37" else
           "1001" when A = x"38" else
           "0000" when A = x"39" else
           "0011" when A = x"3A" else
           "0101" when A = x"3B" else
           "0010" when A = x"3C" else
           "1110" when A = x"3D" else
           "1111" when A = x"3E" else
           "1001" when A = x"3F" else
           "1001";

END Behavioral;
```

Listing G.22: s3_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s3_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s3_box;

architecture Behavioral of s3_box is
```

```
begin

SPO      <= "1010" when A = x"0" else
            "1101" when A = x"1" else
            "0000" when A = x"2" else
            "0111" when A = x"3" else
            "1001" when A = x"4" else
            "0000" when A = x"5" else
            "1110" when A = x"6" else
            "1001" when A = x"7" else
            "0110" when A = x"8" else
            "0011" when A = x"9" else
            "0011" when A = x"A" else
            "0100" when A = x"B" else
            "1111" when A = x"C" else
            "0110" when A = x"D" else
            "0101" when A = x"E" else
            "1010" when A = x"F" else
            "0001" when A = x"10" else
            "0010" when A = x"11" else
            "1101" when A = x"12" else
            "1000" when A = x"13" else
            "1100" when A = x"14" else
            "0101" when A = x"15" else
            "0111" when A = x"16" else
            "1110" when A = x"17" else
            "1011" when A = x"18" else
            "1100" when A = x"19" else
            "0100" when A = x"1A" else
            "1011" when A = x"1B" else
            "0010" when A = x"1C" else
            "1111" when A = x"1D" else
            "1000" when A = x"1E" else
            "0001" when A = x"1F" else
            "1101" when A = x"20" else
            "0001" when A = x"21" else
            "0110" when A = x"22" else
            "1010" when A = x"23" else
            "0100" when A = x"24" else
            "1101" when A = x"25" else
            "1001" when A = x"26" else
            "0000" when A = x"27" else
            "1000" when A = x"28" else
            "0110" when A = x"29" else
            "1111" when A = x"2A" else
            "1001" when A = x"2B" else
            "0011" when A = x"2C" else
            "1000" when A = x"2D" else
            "0000" when A = x"2E" else
            "0111" when A = x"2F" else
            "1011" when A = x"30" else
            "0100" when A = x"31" else
```



```
            "0001" when A = x"32" else
            "1111" when A = x"33" else
            "0010" when A = x"34" else
            "1110" when A = x"35" else
            "1100" when A = x"36" else
            "0011" when A = x"37" else
            "0101" when A = x"38" else
            "1011" when A = x"39" else
            "1010" when A = x"3A" else
            "0101" when A = x"3B" else
            "1110" when A = x"3C" else
            "0010" when A = x"3D" else
            "0111" when A = x"3E" else
            "1100" when A = x"3F" else
            "1100";

END Behavioral;
```

Listing G.23: s4_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s4_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s4_box;

architecture Behavioral of s4_box is

begin

SPO      <= "0111" when A = x"0" else
            "1101" when A = x"1" else
            "1101" when A = x"2" else
            "1000" when A = x"3" else
            "1110" when A = x"4" else
            "1011" when A = x"5" else
            "0011" when A = x"6" else
            "0101" when A = x"7" else
            "0000" when A = x"8" else
            "0110" when A = x"9" else
            "0110" when A = x"A" else
            "1111" when A = x"B" else
            "1001" when A = x"C" else
            "0000" when A = x"D" else
            "1010" when A = x"E" else
            "0011" when A = x"F" else
```

```
            "0001" when A = x"10" else
            "0100" when A = x"11" else
            "0010" when A = x"12" else
            "0111" when A = x"13" else
            "1000" when A = x"14" else
            "0010" when A = x"15" else
            "0101" when A = x"16" else
            "1100" when A = x"17" else
            "1011" when A = x"18" else
            "0001" when A = x"19" else
            "1100" when A = x"1A" else
            "1010" when A = x"1B" else
            "0100" when A = x"1C" else
            "1110" when A = x"1D" else
            "1111" when A = x"1E" else
            "1001" when A = x"1F" else
            "1010" when A = x"20" else
            "0011" when A = x"21" else
            "0110" when A = x"22" else
            "1111" when A = x"23" else
            "1001" when A = x"24" else
            "0000" when A = x"25" else
            "0000" when A = x"26" else
            "0110" when A = x"27" else
            "1100" when A = x"28" else
            "1010" when A = x"29" else
            "1011" when A = x"2A" else
            "0001" when A = x"2B" else
            "0111" when A = x"2C" else
            "1101" when A = x"2D" else
            "1101" when A = x"2E" else
            "1000" when A = x"2F" else
            "1111" when A = x"30" else
            "1001" when A = x"31" else
            "0001" when A = x"32" else
            "0100" when A = x"33" else
            "0011" when A = x"34" else
            "0101" when A = x"35" else
            "1110" when A = x"36" else
            "1011" when A = x"37" else
            "0101" when A = x"38" else
            "1100" when A = x"39" else
            "0010" when A = x"3A" else
            "0111" when A = x"3B" else
            "1000" when A = x"3C" else
            "0010" when A = x"3D" else
            "0100" when A = x"3E" else
            "1110" when A = x"3F" else
            "1110";

END Behavioral;
```


Listing G.24: s5_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s5_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s5_box;

architecture Behavioral of s5_box is

begin

SPO     <= "0010" when A = x"0" else
           "1110" when A = x"1" else
           "1100" when A = x"2" else
           "1011" when A = x"3" else
           "0100" when A = x"4" else
           "0010" when A = x"5" else
           "0001" when A = x"6" else
           "1100" when A = x"7" else
           "0111" when A = x"8" else
           "0100" when A = x"9" else
           "1010" when A = x"A" else
           "0111" when A = x"B" else
           "1011" when A = x"C" else
           "1101" when A = x"D" else
           "0110" when A = x"E" else
           "0001" when A = x"F" else
           "1000" when A = x"10" else
           "0101" when A = x"11" else
           "0101" when A = x"12" else
           "0000" when A = x"13" else
           "0011" when A = x"14" else
           "1111" when A = x"15" else
           "1111" when A = x"16" else
           "1010" when A = x"17" else
           "1101" when A = x"18" else
           "0011" when A = x"19" else
           "0000" when A = x"1A" else
           "1001" when A = x"1B" else
           "1110" when A = x"1C" else
           "1000" when A = x"1D" else
           "1001" when A = x"1E" else
           "0110" when A = x"1F" else
           "0100" when A = x"20" else
           "1011" when A = x"21" else
```

```
            "0010" when A = x"22" else
            "1000" when A = x"23" else
            "0001" when A = x"24" else
            "1100" when A = x"25" else
            "1011" when A = x"26" else
            "0111" when A = x"27" else
            "1010" when A = x"28" else
            "0001" when A = x"29" else
            "1101" when A = x"2A" else
            "1110" when A = x"2B" else
            "0111" when A = x"2C" else
            "0010" when A = x"2D" else
            "1000" when A = x"2E" else
            "1101" when A = x"2F" else
            "1111" when A = x"30" else
            "0110" when A = x"31" else
            "1001" when A = x"32" else
            "1111" when A = x"33" else
            "1100" when A = x"34" else
            "0000" when A = x"35" else
            "0101" when A = x"36" else
            "1001" when A = x"37" else
            "0110" when A = x"38" else
            "1010" when A = x"39" else
            "0011" when A = x"3A" else
            "0100" when A = x"3B" else
            "0000" when A = x"3C" else
            "0101" when A = x"3D" else
            "1110" when A = x"3E" else
            "0011" when A = x"3F" else
            "0011";

END Behavioral;
```

Listing G.25: s6_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s6_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s6_box;

architecture Behavioral of s6_box is

begin
```



```
SPO     <= "1100" when A = x"0" else
           "1010" when A = x"1" else
           "0001" when A = x"2" else
           "1111" when A = x"3" else
           "1010" when A = x"4" else
           "0100" when A = x"5" else
           "1111" when A = x"6" else
           "0010" when A = x"7" else
           "1001" when A = x"8" else
           "0111" when A = x"9" else
           "0010" when A = x"A" else
           "1100" when A = x"B" else
           "0110" when A = x"C" else
           "1001" when A = x"D" else
           "1000" when A = x"E" else
           "0101" when A = x"F" else
           "0000" when A = x"10" else
           "0110" when A = x"11" else
           "1101" when A = x"12" else
           "0001" when A = x"13" else
           "0011" when A = x"14" else
           "1101" when A = x"15" else
           "0100" when A = x"16" else
           "1110" when A = x"17" else
           "1110" when A = x"18" else
           "0000" when A = x"19" else
           "0111" when A = x"1A" else
           "1011" when A = x"1B" else
           "0101" when A = x"1C" else
           "0011" when A = x"1D" else
           "1011" when A = x"1E" else
           "1000" when A = x"1F" else
           "1001" when A = x"20" else
           "0100" when A = x"21" else
           "1110" when A = x"22" else
           "0011" when A = x"23" else
           "1111" when A = x"24" else
           "0010" when A = x"25" else
           "0101" when A = x"26" else
           "1100" when A = x"27" else
           "0010" when A = x"28" else
           "1001" when A = x"29" else
           "1000" when A = x"2A" else
           "0101" when A = x"2B" else
           "1100" when A = x"2C" else
           "1111" when A = x"2D" else
           "0011" when A = x"2E" else
           "1010" when A = x"2F" else
           "0111" when A = x"30" else
           "1011" when A = x"31" else
           "0000" when A = x"32" else
           "1110" when A = x"33" else
```

```
            "0100" when A = x"34" else
            "0001" when A = x"35" else
            "1010" when A = x"36" else
            "0111" when A = x"37" else
            "0001" when A = x"38" else
            "0110" when A = x"39" else
            "1101" when A = x"3A" else
            "0000" when A = x"3B" else
            "1011" when A = x"3C" else
            "1000" when A = x"3D" else
            "0110" when A = x"3E" else
            "1101" when A = x"3F" else
            "1101";

END Behavioral;
```

Listing G.26: s7_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s7_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s7_box;

architecture Behavioral of s7_box is

begin

SPO     <= "0100" when A = x"0" else
           "1101" when A = x"1" else
           "1011" when A = x"2" else
           "0000" when A = x"3" else
           "0010" when A = x"4" else
           "1011" when A = x"5" else
           "1110" when A = x"6" else
           "0111" when A = x"7" else
           "1111" when A = x"8" else
           "0100" when A = x"9" else
           "0000" when A = x"A" else
           "1001" when A = x"B" else
           "1000" when A = x"C" else
           "0001" when A = x"D" else
           "1101" when A = x"E" else
           "1010" when A = x"F" else
           "0011" when A = x"10" else
           "1110" when A = x"11" else
```



```
           "1100" when A = x"12" else
           "0011" when A = x"13" else
           "1001" when A = x"14" else
           "0101" when A = x"15" else
           "0111" when A = x"16" else
           "1100" when A = x"17" else
           "0101" when A = x"18" else
           "0010" when A = x"19" else
           "1010" when A = x"1A" else
           "1111" when A = x"1B" else
           "0110" when A = x"1C" else
           "1000" when A = x"1D" else
           "0001" when A = x"1E" else
           "0110" when A = x"1F" else
           "0001" when A = x"20" else
           "0110" when A = x"21" else
           "0100" when A = x"22" else
           "1011" when A = x"23" else
           "1011" when A = x"24" else
           "1101" when A = x"25" else
           "1101" when A = x"26" else
           "1000" when A = x"27" else
           "1100" when A = x"28" else
           "0001" when A = x"29" else
           "0011" when A = x"2A" else
           "0100" when A = x"2B" else
           "0111" when A = x"2C" else
           "1010" when A = x"2D" else
           "1110" when A = x"2E" else
           "0111" when A = x"2F" else
           "1010" when A = x"30" else
           "1001" when A = x"31" else
           "1111" when A = x"32" else
           "0101" when A = x"33" else
           "0110" when A = x"34" else
           "0000" when A = x"35" else
           "1000" when A = x"36" else
           "1111" when A = x"37" else
           "0000" when A = x"38" else
           "1110" when A = x"39" else
           "0101" when A = x"3A" else
           "0010" when A = x"3B" else
           "1001" when A = x"3C" else
           "0011" when A = x"3D" else
           "0010" when A = x"3E" else
           "1100" when A = x"3F" else
           "1100";

END Behavioral;
```

Listing G.27: s8_box.vhd

```
-- CoreTex
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY s8_box IS
    port (
    A: IN std_logic_VECTOR(5 downto 0);
    SPO: OUT std_logic_VECTOR(3 downto 0));
END s8_box;

architecture Behavioral of s8_box is

begin

SPO     <= "1101" when A = x"0" else
           "0001" when A = x"1" else
           "0010" when A = x"2" else
           "1111" when A = x"3" else
           "1000" when A = x"4" else
           "1101" when A = x"5" else
           "0100" when A = x"6" else
           "1000" when A = x"7" else
           "0110" when A = x"8" else
           "1010" when A = x"9" else
           "1111" when A = x"A" else
           "0011" when A = x"B" else
           "1011" when A = x"C" else
           "0111" when A = x"D" else
           "0001" when A = x"E" else
           "0100" when A = x"F" else
           "1010" when A = x"10" else
           "1100" when A = x"11" else
           "1001" when A = x"12" else
           "0101" when A = x"13" else
           "0011" when A = x"14" else
           "0110" when A = x"15" else
           "1110" when A = x"16" else
           "1011" when A = x"17" else
           "0101" when A = x"18" else
           "0000" when A = x"19" else
           "0000" when A = x"1A" else
           "1110" when A = x"1B" else
           "1100" when A = x"1C" else
           "1001" when A = x"1D" else
           "0111" when A = x"1E" else
           "0010" when A = x"1F" else
           "0111" when A = x"20" else
           "0010" when A = x"21" else
           "1011" when A = x"22" else
           "0001" when A = x"23" else
```



```
           "0100" when A = x"24" else
           "1110" when A = x"25" else
           "0001" when A = x"26" else
           "0111" when A = x"27" else
           "1001" when A = x"28" else
           "0100" when A = x"29" else
           "1100" when A = x"2A" else
           "1010" when A = x"2B" else
           "1110" when A = x"2C" else
           "1000" when A = x"2D" else
           "0010" when A = x"2E" else
           "1101" when A = x"2F" else
           "0000" when A = x"30" else
           "1111" when A = x"31" else
           "0110" when A = x"32" else
           "1100" when A = x"33" else
           "1010" when A = x"34" else
           "1001" when A = x"35" else
           "1101" when A = x"36" else
           "0000" when A = x"37" else
           "1111" when A = x"38" else
           "0011" when A = x"39" else
           "0011" when A = x"3A" else
           "0101" when A = x"3B" else
           "0101" when A = x"3C" else
           "0110" when A = x"3D" else
           "1000" when A = x"3E" else
           "1011" when A = x"3F" else
           "1011";

END Behavioral;
```

Listing G.28: s_box.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity s_box is
port(
           block0_in: in std_logic_vector(5 downto 0);
           block1_in: in std_logic_vector(5 downto 0);
           block2_in: in std_logic_vector(5 downto 0);
           block3_in: in std_logic_vector(5 downto 0);
           block4_in: in std_logic_vector(5 downto 0);
           block5_in: in std_logic_vector(5 downto 0);
           block6_in: in std_logic_vector(5 downto 0);
           block7_in: in std_logic_vector(5 downto 0);
           x0_out: out std_logic_vector(3 downto 0);
           x1_out: out std_logic_vector(3 downto 0);
```

```
            x2_out: out std_logic_vector(3 downto 0);
            x3_out: out std_logic_vector(3 downto 0);
            x4_out: out std_logic_vector(3 downto 0);
            x5_out: out std_logic_vector(3 downto 0);
            x6_out: out std_logic_vector(3 downto 0);
            x7_out: out std_logic_vector(3 downto 0)
    );
end s_box;

architecture Behavioral of s_box is

component s1_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;

component s2_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;

component s3_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;

component s4_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;

component s5_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;

component s6_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;
```



```
component s7_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;

component s8_box
    port(
    a: in  std_logic_VECTOR(5 downto 0);
    spo: out std_logic_VECTOR(3 downto 0)
    );
end component;

begin

S1 : s1_box
    port map (
        a => block0_in,
        spo => x0_out
);

S2 : s2_box
    port map (
        a => block1_in,
        spo => x1_out
);

S3 : s3_box
    port map (
        a => block2_in,
        spo => x2_out
);

S4 : s4_box
    port map (
        a => block3_in,
        spo => x3_out
);

S5 : s5_box
    port map (
        a => block4_in,
        spo => x4_out
);

S6 : s6_box
    port map (
        a => block5_in,
        spo => x5_out
);
```

```
S7 : s7_box
    port map (
        a => block6_in,
        spo => x6_out
);

S8 : s8_box
    port map (
        a => block7_in,
        spo => x7_out
);

end Behavioral;
```

Listing G.29: tdes_top.vhd

```
-- CoreTex

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tdes_top is
port(
        --
        -- inputs for key expander
        --
        key1_in:        in std_logic_vector(0 to 63);
        key2_in:        in std_logic_vector(0 to 63);
        key3_in:        in std_logic_vector(0 to 63);

        --
        -- function select
        --
        function_select:    in  std_logic; -- active when encryption,
           inactive when decryption

        --
        -- input into des_cipher_top
        --
        data_in:        in std_logic_vector(0 to 63);

        --
        -- input into des_cipher_top
        --
        data_out:   out std_logic_vector(0 to 63);

        --
        -- data interface to MCU
        --
```



```
        lddata:         in  std_logic;  -- active when data for loading
            is ready
        ldkey:          in  std_logic;  -- active when key for loading
           is ready
        out_ready:  out std_logic;  -- active when encryption of data
           is done

        --
        -- General clock and reset
        --
        reset: in std_logic;
        clock: in std_logic
    );
end tdes_top;

architecture Behavioral of tdes_top is

component des_cipher_top is
port(
        --
        -- inputs for key expander
        --
        key_in:     in std_logic_vector(0 to 63);   -- interface to MCU
        --ldkey:        in std_logic;                        -- active
           signal for loading keys

        --
        -- function select
        --
        function_select:    in  std_logic; -- active when encryption,
           inactive when decryption

        --
        -- input into des_cipher_top
        --
        data_in:        in std_logic_vector(0 to 63);

        --
        -- input into des_cipher_top
        --
        data_out:   out std_logic_vector(0 to 63);

        --
        -- data interface to MCU
        --
        lddata:         in  std_logic;  -- active when data for loading
            is ready
        des_out_rdy:    out std_logic;  -- active when encryption of
           data is done

        --
        -- General clock and reset
```

```
        --
        reset: in std_logic;
        clock: in std_logic
    );
end component;

type statetype is (WaitKeyState, WaitDataState);
signal nextstate: statetype;
signal key1_in_internal: std_logic_vector(0 to 63);
signal key2_in_internal: std_logic_vector(0 to 63);
signal key3_in_internal: std_logic_vector(0 to 63);
signal memkey1: std_logic_vector(0 to 63);
signal memkey3: std_logic_vector(0 to 63);
signal fsel_internal: std_logic;
signal fsel_internal_inv: std_logic;
signal des_out_rdy_internal: std_logic;
signal des_out_rdy_internal1: std_logic;
signal des_out_rdy_internal2: std_logic;
signal data_in_internal: std_logic_vector(0 to 63);
signal data_out_internal: std_logic_vector(0 to 63);
signal data_out_internal1: std_logic_vector(0 to 63);
signal data_out_internal2: std_logic_vector(0 to 63);
signal lddata_internal: std_logic;

begin

process (clock)
begin
if rising_edge(clock) then
    if reset = '1' then
        nextstate       <= WaitKeyState;
        lddata_internal     <= '0';
        out_ready       <= '0';
        fsel_internal       <= function_select;
        fsel_internal_inv   <= not function_select;
    else
        data_out                <= data_out_internal;
        out_ready               <= des_out_rdy_internal;

        case nextstate is
            --
            -- wait key state
            --
            when WaitKeyState =>
                -- wait until key is ready (as well as the
                   function_select)
                if ldkey = '0' then
                    nextstate       <= WaitKeyState;
                else
                    key1_in_internal    <= key1_in;
                    key2_in_internal    <= key2_in;
                    key3_in_internal    <= key3_in;
```



```
                    memkey1         <= key1_in;
                    memkey3         <= key3_in;
                    nextstate       <= WaitDataState;
                end if;
            --
            -- wait data state
            --
            when WaitDataState =>
                -- wait until data is ready to be loaded
                if lddata = '0' then
                    nextstate   <= WaitDataState;
                    lddata_internal     <= '0';
                else
                    lddata_internal     <= '1';
                    if fsel_internal = '0' then
                        key1_in_internal <= memkey3;
                        key3_in_internal <= memkey1;
                    end if;
                    data_in_internal    <= data_in;
                    nextstate       <= WaitDataState;
                end if;

        end case;
    end if;
end if;
end process;


DESCIPHERTOP1: des_cipher_top
port map (
        key_in => key1_in_internal, -- interface to MCU

        function_select => fsel_internal, -- active when encryption,
           inactive when decryption

        data_in => data_in_internal,

        data_out => data_out_internal1,

        lddata => lddata_internal,  -- active when data for loading is
           ready
        des_out_rdy => des_out_rdy_internal1,   -- active when
           encryption of data is done

        reset => reset,
        clock =>    clock
);

DESCIPHERTOP2: des_cipher_top
port map (
        key_in => key2_in_internal, --subkey_in_internal,   --
           interface to MCU
```

```
        function_select => fsel_internal_inv, -- active when encryption
           , inactive when decryption

        data_in => data_out_internal1,

        data_out => data_out_internal2,

        lddata => des_out_rdy_internal1,    -- active when data for
           loading is ready
        des_out_rdy => des_out_rdy_internal2,   -- active when
           encryption of data is done

        reset => reset,
        clock =>    clock
);

DESCIPHERTOP3: des_cipher_top
port map (
        key_in => key3_in_internal, -- subkey_in_internal,  --
           interface to MCU

        function_select => fsel_internal, -- active when encryption,
           inactive when decryption

        data_in => data_out_internal2,

        data_out => data_out_internal,

        lddata => des_out_rdy_internal2,    -- active when data for
           loading is ready
        des_out_rdy => des_out_rdy_internal,    -- active when
           encryption of data is done

        reset => reset,
        clock =>    clock
);

end Behavioral;
```

G.2 C code

The next code listing is the software for our microcontroller that acted as a main controller in our system. Then there will come 3 files in the end which is the gsm test code, and lastly the last file is the program used for filling the SD card with name, numbers and crypto keys.

Listing G.30: src/init.c

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdio.h>
```



```
#include <string.h>
#include <stdlib.h>

#include "assert.h"
#include "delay.h"
#include "fpga.h"
#include "fs.h"
#include "jtag.h"
#include "memory.h"
#include "output.h"
#include "tests.h"
#include "time.h"
#include "thread.h"
#include "menu.h"
#include "usart.h"
#include "lcd.h"
#include "keyboard_int.h"
#include "sleep.h"
#include "sms.h"
#include "timeout.h"
#include "fat16.h"
#include "sd_raw.h"
#include "partition.h"
#include "ui.h"
#include "message.h"
#include "modem.h"

static thread_t modem_thread_data;

thread_t timeout_thread;
char screen_buf[160];

void init_intr(void)
{
  PORTD |= 0x01;            // enable pullups for interrupt 0
  DDRD &= ~0x01;          // interrupt pin is input
  EICRA = (EICRA & 0xfc) | 0x00; // set interrupts 0 to active low mode
  EIMSK |= 0x01;          // enable ext int 0
}


/* Entry point for this code.
 */
int main(void)
{
  init_xmem();

  if(FPGA_KEYBOARD & 0x0001) {
    FPGA_LEDS = 0xcc;
    for(;;);
  }
  FPGA_LEDS = 0;
```

```
  //cli();
  //ASSERT_INTERRUPTS_OFF();

  init_intr();
  //cli();
  //ASSERT_INTERRUPTS_OFF();

  delay(500000UL);

  //cli();
  //ASSERT_INTERRUPTS_OFF();

  jtag_disable();
  //cli();
  //ASSERT_INTERRUPTS_OFF();
  init_xmem();
  //cli();
  //ASSERT_INTERRUPTS_OFF();
  output_init();

  //cli();
  //ASSERT_INTERRUPTS_OFF();
  printf_P(PSTR("Initializing cryptophone...\n"));
  ASSERT_INTERRUPTS_OFF();

  ASSERT_INTERRUPTS_OFF();
  printf_P(PSTR("Initializing timer...\n"));
  time_init();

  ASSERT_INTERRUPTS_OFF();
  printf_P(PSTR("Resetting usarts.\n"));
  usart_init();

  DDRE = 0x80;
  PORTE = 0x80;

  DDRB = 0x40;

  // turn off LCD backlight:
  PORTB |= 1<<6;
  ASSERT_INTERRUPTS_OFF();

  fs_init();
  ASSERT_INTERRUPTS_OFF();

  printf_P(PSTR("Initializing threads...\n"));
  thread_init();

  ASSERT_INTERRUPTS_OFF();

  message_init();
```



```
  ASSERT_INTERRUPTS_OFF();

  tests_sleep();

  ASSERT_INTERRUPTS_OFF();

  DDRD |= 1<<7;

  /* Create worker threads. */

  thread_setup(&modem_thread_data, modem_thread, NULL);

  ASSERT_INTERRUPTS_OFF();

  timeout_init(&timeout_thread);

  ASSERT_INTERRUPTS_OFF();

  sms_init();
  menu_init();
  ASSERT_INTERRUPTS_OFF();
  menu_set_screen_buffer(screen_buf);
  ASSERT_INTERRUPTS_OFF();

  ui_init();
  ASSERT_INTERRUPTS_OFF();

  keyboard_int_init(menu_key_pressed);
  ASSERT_INTERRUPTS_OFF();

  /* Begin threading. */
  thread_begin();
}
```

Listing G.31: src/crypt.c

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>

#include "assert.h"
#include "fpga.h"

#define MODE_ENCRYPT  1
#define MODE_DECRYPT  2

static void crypt_block(uint8_t *dest, uint8_t *source, uint8_t mode)
{
  /* Set data. */
  memcpy((void *)FPGA_CRYPT_DATA_IN, source, 8);

  /* Start encryption/decryption. */
```

```
  switch(mode) {
  case MODE_ENCRYPT:
    FPGA_CRYPT_ENCRYPT = 1;
    break;
  case MODE_DECRYPT:
    FPGA_CRYPT_DECRYPT = 1;
    break;
  default:
    DIE();
  }

  /* Wait for encryption completed. */
  while(FPGA_CRYPT_STATUS != 0x04 && FPGA_CRYPT_STATUS != 0x01);

  /* Copy the encrypted data out. */
  memcpy(dest, (void *)FPGA_CRYPT_DATA_OUT, 8);
}


/*
 * Encrypt len bytes of data from src into dest. len MUST be a
 * multiple of 8.
 */
void crypt_encrypt(uint8_t *dest, uint8_t *src, uint8_t len, uint8_t *
    key)
{
  uint8_t i;

  ASSERT(len % 8 == 0);

  FPGA_CRYPT_RESET_CBC_ENCRYPTION = 1;

  memcpy((void *)FPGA_CRYPT_KEY, key, 3*8);

  for (i = 0; i < len; i+=8) {
    crypt_block(dest + i, src + i, MODE_ENCRYPT);
  }
}

/*
 * Decrypt len bytes of data from src into dest. len MUST be a
 * multiple of 8.
 */
void crypt_decrypt(uint8_t *dest, uint8_t *src, uint8_t len, uint8_t *
    key)
{
  uint8_t i;

  ASSERT(len % 8 == 0);

  FPGA_CRYPT_RESET_CBC_DECRYPTION = 1;
```



```
  memcpy((void *)FPGA_CRYPT_KEY, key, 3*8);

  for (i = 0; i < len; i+=8) {
    crypt_block(dest + i, src + i, MODE_DECRYPT);
  }
}
```

Listing G.32: src/delay.c

```
#define F_CPU 8000000UL  // 8 MHz

#include <util/delay.h>

#include "delay.h"


void delay(uint32_t us)
{
  while(us > 10) {
    _delay_us(10);
    us -= 10;
  }
  _delay_us(us);
}
```

Listing G.33: src/error.c

```
#include <stdlib.h>
#include <stdio.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>

#include "lcd.h"
#include "menu.h"

void fatal_error(const char *fmt, ...)
{
  cli();
  if (lcd_get_pos() == 255)
    lcd_goto(0);
  printf_P(PSTR("ERROR: "));
  va_list arg;
  va_start(arg, fmt);
  vprintf(fmt, arg);
  va_end(arg);
  while (1) ;
}

void fatal_error_P(const char *fmt, ...)
{
  cli();
  if (lcd_get_pos() == 255)
    lcd_goto(0);
```

```
  printf_P(PSTR("ERROR: "));
  va_list arg;
  va_start(arg, fmt);
  vfprintf_P(stdout, fmt, arg);
  va_end(arg);
  while (1);
}

void error(const char *fmt, ...)
{
  menu_t *m = menu_add(PSTR("Error"), 1);
  menu_setup_done(m);

  char *str = malloc(sizeof(char)*(MENU_LINE_WIDTH*3));
  va_list arg;
  va_start(arg, fmt);
  vsprintf(str, fmt, arg);
  va_end(arg);
  menu_setstring(0, str);
  menu_update_display();
  free(str);
}

void error_P(const char *fmt, ...)
{
  menu_t *m = menu_add(PSTR("Error"), 1);
  menu_setup_done(m);

  char *str = malloc(sizeof(char)*(MENU_LINE_WIDTH*3));
  va_list arg;
  va_start(arg, fmt);
  vsprintf_P(str, fmt, arg);
  va_end(arg);
  menu_setstring(0, str);
  menu_update_display();
  free(str);
}
```

Listing G.34: src/fat16.c

```
/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#include "partition.h"
#include "fat16.h"
#include "fat16_config.h"
#include "sd-reader_config.h"
```



```
#include <string.h>

#if USE_DYNAMIC_MEMORY
    #include <stdlib.h>
#endif

/**
 * \addtogroup fat16 FAT16 support
 *
 * This module implements FAT16 read and write access.
 *
 * The following features are supported:
 * - File names up to 31 characters long.
 * - Unlimited depth of subdirectories.
 * - Short 8.3 and long filenames.
 * - Creating and deleting files.
 * - Reading and writing from and to files.
 * - File resizing.
 * - File sizes of up to 4 gigabytes.
 *
 * @{
 */
/**
 * \file
 * FAT16 implementation.
 *
 * \author Roland Riegel
 */

/**
 * \addtogroup fat16_config FAT16 configuration
 * Preprocessor defines to configure the FAT16 implementation.
 */

/**
 * \addtogroup fat16_fs FAT16 access
 * Basic functions for handling a FAT16 filesystem.
 */

/**
 * \addtogroup fat16_file FAT16 file functions
 * Functions for managing files.
 */

/**
 * \addtogroup fat16_dir FAT16 directory functions
 * Functions for managing directories.
 */

/**
 * @}
```

```
 */

#define FAT16_CLUSTER_FREE 0x0000
#define FAT16_CLUSTER_RESERVED_MIN 0xfff0
#define FAT16_CLUSTER_RESERVED_MAX 0xfff6
#define FAT16_CLUSTER_BAD 0xfff7
#define FAT16_CLUSTER_LAST_MIN 0xfff8
#define FAT16_CLUSTER_LAST_MAX 0xffff

#define FAT16_DIRENTRY_DELETED 0xe5
#define FAT16_DIRENTRY_LFNLAST (1 << 6)
#define FAT16_DIRENTRY_LFNSEQMASK ((1 << 6) - 1)

/* Each entry within the directory table has a size of 32 bytes
 * and either contains a 8.3 DOS-style file name or a part of a
 * long file name, which may consist of several directory table
 * entries at once.
 *
 * multi-byte integer values are stored little-endian!
 *
 * 8.3 file name entry:
 * ====================
 * offset  length  description
 *      0       8  name (space padded)
 *      8       3  extension (space padded)
 *     11       1  attributes (FAT16_ATTRIB_*)
 *
 * long file name (lfn) entry ordering for a single file name:
 * ===========================================================
 * LFN entry n
 *     ...
 * LFN entry 2
 * LFN entry 1
 * 8.3 entry (see above)
 *
 * lfn entry:
 * ==========
 * offset  length  description
 *      0       1  ordinal field
 *      1       2  unicode character 1
 *      3       3  unicode character 2
 *      5       3  unicode character 3
 *      7       3  unicode character 4
 *      9       3  unicode character 5
 *     11       1  attribute (always 0x0f)
 *     12       1  type (reserved, always 0)
 *     13       1  checksum
 *     14       2  unicode character 6
 *     16       2  unicode character 7
 *     18       2  unicode character 8
 *     20       2  unicode character 9
 *     22       2  unicode character 10
```



```
 *     24       2  unicode character 11
 *     26       2  cluster (unused, always 0)
 *     28       2  unicode character 12
 *     30       2  unicode character 13
 *
 * The ordinal field contains a descending number, from n to 1.
 * For the n'th lfn entry the ordinal field is or'ed with 0x40.
 * For deleted lfn entries, the ordinal field is set to 0xe5.
 */

struct fat16_header_struct
{
    uint32_t size;

    uint32_t fat_offset;
    uint32_t fat_size;

    uint16_t sector_size;
    uint16_t cluster_size;

    uint32_t root_dir_offset;

    uint32_t cluster_zero_offset;
};

struct fat16_fs_struct
{
    struct partition_struct* partition;
    struct fat16_header_struct header;
};

struct fat16_file_struct
{
    struct fat16_fs_struct* fs;
    struct fat16_dir_entry_struct dir_entry;
    uint32_t pos;
    uint16_t pos_cluster;
};

struct fat16_dir_struct
{
    struct fat16_fs_struct* fs;
    struct fat16_dir_entry_struct dir_entry;
    uint16_t entry_next;
};

struct fat16_read_callback_arg
{
    uint16_t entry_cur;
    uint16_t entry_num;
    uint32_t entry_offset;
    uint8_t byte_count;
```

```
};

struct fat16_usage_count_callback_arg
{
    uint16_t cluster_count;
    uint8_t buffer_size;
};

#if !USE_DYNAMIC_MEMORY
static struct fat16_fs_struct fat16_fs_handlers[FAT16_FS_COUNT];
static struct fat16_file_struct fat16_file_handlers[FAT16_FILE_COUNT];
static struct fat16_dir_struct fat16_dir_handlers[FAT16_DIR_COUNT];
#endif

static uint8_t fat16_read_header(struct fat16_fs_struct* fs);
static uint8_t fat16_read_root_dir_entry(const struct fat16_fs_struct*
   fs, uint16_t entry_num, struct fat16_dir_entry_struct* dir_entry);
static uint8_t fat16_read_sub_dir_entry(const struct fat16_fs_struct*
   fs, uint16_t entry_num, const struct fat16_dir_entry_struct* parent,
    struct fat16_dir_entry_struct* dir_entry);
static uint8_t fat16_dir_entry_seek_callback(uint8_t* buffer, uint32_t
   offset, void* p);
static uint8_t fat16_dir_entry_read_callback(uint8_t* buffer, uint32_t
   offset, void* p);
static uint8_t fat16_interpret_dir_entry(struct fat16_dir_entry_struct*
    dir_entry, const uint8_t* raw_entry);
static uint16_t fat16_get_next_cluster(const struct fat16_fs_struct* fs
   , uint16_t cluster_num);
static uint16_t fat16_append_clusters(const struct fat16_fs_struct* fs,
    uint16_t cluster_num, uint16_t count);
static uint8_t fat16_free_clusters(const struct fat16_fs_struct* fs,
   uint16_t cluster_num);
static uint8_t fat16_terminate_clusters(const struct fat16_fs_struct*
   fs, uint16_t cluster_num);
static uint8_t fat16_clear_cluster(const struct fat16_fs_struct* fs,
   uint16_t cluster_num);
static uint16_t fat16_clear_cluster_callback(uint8_t* buffer, uint32_t
   offset, void* p);
static uint32_t fat16_find_offset_for_dir_entry(const struct
   fat16_fs_struct* fs, const struct fat16_dir_struct* parent, const
   struct fat16_dir_entry_struct* dir_entry);
static uint8_t fat16_write_dir_entry(const struct fat16_fs_struct* fs,
   struct fat16_dir_entry_struct* dir_entry);

static uint8_t fat16_get_fs_free_callback(uint8_t* buffer, uint32_t
   offset, void* p);

static void fat16_set_file_modification_date(struct
   fat16_dir_entry_struct* dir_entry, uint16_t year, uint8_t month,
   uint8_t day);
static void fat16_set_file_modification_time(struct
   fat16_dir_entry_struct* dir_entry, uint8_t hour, uint8_t min,
```



```
   uint8_t sec);

/**
 * \ingroup fat16_fs
 * Opens a FAT16 filesystem.
 *
 * \param[in] partition Discriptor of partition on which the filesystem
     resides.
 * \returns 0 on error, a FAT16 filesystem descriptor on success.
 * \see fat16_open
 */
struct fat16_fs_struct* fat16_open(struct partition_struct* partition)
{
    if(!partition ||
#if FAT16_WRITE_SUPPORT
       !partition->device_write ||
       !partition->device_write_interval
#else
       0
#endif
      )
        return 0;

#if USE_DYNAMIC_MEMORY
    struct fat16_fs_struct* fs = malloc(sizeof(*fs));
    if(!fs)
        return 0;
#else
    struct fat16_fs_struct* fs = fat16_fs_handlers;
    uint8_t i;
    for(i = 0; i < FAT16_FS_COUNT; ++i)
    {
        if(!fs->partition)
            break;

        ++fs;
    }
    if(i >= FAT16_FS_COUNT)
        return 0;
#endif

    memset(fs, 0, sizeof(*fs));

    fs->partition = partition;
    if(!fat16_read_header(fs))
    {
#if USE_DYNAMIC_MEMORY
        free(fs);
#else
        fs->partition = 0;
#endif
        return 0;
```

```
    }

    return fs;
}

/**
 * \ingroup fat16_fs
 * Closes a FAT16 filesystem.
 *
 * When this function returns, the given filesystem descriptor
 * will be invalid.
 *
 * \param[in] fs The filesystem to close.
 * \see fat16_open
 */
void fat16_close(struct fat16_fs_struct* fs)
{
    if(!fs)
        return;

#if USE_DYNAMIC_MEMORY
    free(fs);
#else
    fs->partition = 0;
#endif
}

/**
 * \ingroup fat16_fs
 * Reads and parses the header of a FAT16 filesystem.
 *
 * \param[inout] fs The filesystem for which to parse the header.
 * \returns 0 on failure, 1 on success.
 */
uint8_t fat16_read_header(struct fat16_fs_struct* fs)
{
    if(!fs)
        return 0;

    struct partition_struct* partition = fs->partition;
    if(!partition)
        return 0;

    /* read fat parameters */
    uint8_t buffer[25];
    uint32_t partition_offset = partition->offset * 512;
    if(!partition->device_read(partition_offset + 0x0b, buffer, sizeof(
       buffer)))
        return 0;

    uint16_t bytes_per_sector = ((uint16_t) buffer[0x00]) |
                                ((uint16_t) buffer[0x01] << 8);
```



```
    uint8_t sectors_per_cluster = buffer[0x02];
    uint16_t reserved_sectors = ((uint16_t) buffer[0x03]) |
                                ((uint16_t) buffer[0x04] << 8);
    uint8_t fat_copies = buffer[0x05];
    uint16_t max_root_entries = ((uint16_t) buffer[0x06]) |
                                ((uint16_t) buffer[0x07] << 8);
    uint16_t sector_count_16 = ((uint16_t) buffer[0x08]) |
                               ((uint16_t) buffer[0x09] << 8);
    uint16_t sectors_per_fat = ((uint16_t) buffer[0x0b]) |
                               ((uint16_t) buffer[0x0c] << 8);
    uint32_t sector_count = ((uint32_t) buffer[0x15]) |
                            ((uint32_t) buffer[0x16] << 8) |
                            ((uint32_t) buffer[0x17] << 16) |
                            ((uint32_t) buffer[0x18] << 24);

    if(sectors_per_fat == 0)
        /* this is not a FAT16 */
        return 0;

    if(sector_count == 0)
    {
        if(sector_count_16 == 0)
            /* illegal volume size */
            return 0;
        else
            sector_count = sector_count_16;
    }

    /* ensure we really have a FAT16 fs here */
    uint32_t data_sector_count = sector_count
                                 - reserved_sectors
                                 - (uint32_t) sectors_per_fat *
                                    fat_copies
                                 - ((max_root_entries * 32 +
                                    bytes_per_sector - 1) /
                                    bytes_per_sector);
    uint32_t data_cluster_count = data_sector_count /
       sectors_per_cluster;
    if(data_cluster_count < 4085 || data_cluster_count >= 65525)
        /* this is not a FAT16 */
        return 0;

    partition->type = PARTITION_TYPE_FAT16;

    /* fill header information */
    struct fat16_header_struct* header = &fs->header;
    memset(header, 0, sizeof(*header));

    header->size = sector_count * bytes_per_sector;

    header->fat_offset = /* jump to partition */
                         partition_offset +
```

```
                        /* jump to fat */
                        (uint32_t) reserved_sectors * bytes_per_sector
                            ;
    header->fat_size = (data_cluster_count + 2) * 2;

    header->sector_size = bytes_per_sector;
    header->cluster_size = (uint32_t) bytes_per_sector *
        sectors_per_cluster;

    header->root_dir_offset = /* jump to fats */
                              header->fat_offset +
                              /* jump to root directory entries */
                              (uint32_t) fat_copies * sectors_per_fat *
                                  bytes_per_sector;

    header->cluster_zero_offset = /* jump to root directory entries */
                                  header->root_dir_offset +
                                  /* skip root directory entries */
                                  (uint32_t) max_root_entries * 32;

    return 1;
}

/**
 * \ingroup fat16_fs
 * Reads a directory entry of the root directory.
 *
 * \param[in] fs Descriptor of file system to use.
 * \param[in] entry_num The index of the directory entry to read.
 * \param[out] dir_entry Directory entry descriptor which will get
    filled.
 * \returns 0 on failure, 1 on success
 * \see fat16_read_sub_dir_entry, fat16_read_dir_entry_by_path
 */
uint8_t fat16_read_root_dir_entry(const struct fat16_fs_struct* fs,
   uint16_t entry_num, struct fat16_dir_entry_struct* dir_entry)
{
    if(!fs || !dir_entry)
        return 0;

    /* we read from the root directory entry */
    const struct fat16_header_struct* header = &fs->header;
    device_read_interval_t device_read_interval = fs->partition->
       device_read_interval;
    uint8_t buffer[32];

    /* seek to the n-th entry */
    struct fat16_read_callback_arg arg;
    memset(&arg, 0, sizeof(arg));
    arg.entry_num = entry_num;
    if(!device_read_interval(header->root_dir_offset,
                             buffer,
```



```
                             sizeof(buffer),
                             header->cluster_zero_offset - header->
                                root_dir_offset,
                             fat16_dir_entry_seek_callback,
                             &arg) ||
       arg.entry_offset == 0
      )
        return 0;

    /* read entry */
    memset(dir_entry, 0, sizeof(*dir_entry));
    if(!device_read_interval(arg.entry_offset,
                             buffer,
                             sizeof(buffer),
                             arg.byte_count,
                             fat16_dir_entry_read_callback,
                             dir_entry))
        return 0;

    return dir_entry->long_name[0] != '\0' ? 1 : 0;
}

/**
 * \ingroup fat16_fs
 * Reads a directory entry of a given parent directory.
 *
 * \param[in] fs Descriptor of file system to use.
 * \param[in] entry_num The index of the directory entry to read.
 * \param[in] parent Directory entry descriptor in which to read
    directory entry.
 * \param[out] dir_entry Directory entry descriptor which will get
    filled.
 * \returns 0 on failure, 1 on success
 * \see fat16_read_root_dir_entry, fat16_read_dir_entry_by_path
 */
uint8_t fat16_read_sub_dir_entry(const struct fat16_fs_struct* fs,
   uint16_t entry_num, const struct fat16_dir_entry_struct* parent,
   struct fat16_dir_entry_struct* dir_entry)
{
    if(!fs || !parent || !dir_entry)
        return 0;

    /* we are in a parent directory and want to search within its
       directory entry table */
    if(!(parent->attributes & FAT16_ATTRIB_DIR))
        return 0;

    /* loop through all clusters of the directory */
    uint8_t buffer[32];
    uint32_t cluster_offset;
    uint16_t cluster_size = fs->header.cluster_size;
    uint16_t cluster_num = parent->cluster;
```

```
    struct fat16_read_callback_arg arg;

    while(1)
    {
        /* calculate new cluster offset */
        cluster_offset = fs->header.cluster_zero_offset + (uint32_t) (
           cluster_num - 2) * cluster_size;

        /* seek to the n-th entry */
        memset(&arg, 0, sizeof(arg));
        arg.entry_num = entry_num;
        if(!fs->partition->device_read_interval(cluster_offset,
                                                buffer,
                                                sizeof(buffer),
                                                cluster_size,
                                                fat16_dir_entry_seek_callback
                                                   ,
                                                &arg)
          )
            return 0;

        /* check if we found the entry */
        if(arg.entry_offset)
            break;

        /* get number of next cluster */
        if(!(cluster_num = fat16_get_next_cluster(fs, cluster_num)))
            return 0; /* directory entry not found */
    }

    memset(dir_entry, 0, sizeof(*dir_entry));

    /* read entry */
    if(!fs->partition->device_read_interval(arg.entry_offset,
                                           buffer,
                                           sizeof(buffer),
                                           arg.byte_count,
                                           fat16_dir_entry_read_callback
                                              ,
                                           dir_entry))
        return 0;

    return dir_entry->long_name[0] != '\0' ? 1 : 0;
}

/**
 * \ingroup fat16_fs
 * Callback function for seeking through subdirectory entries.
 */
uint8_t fat16_dir_entry_seek_callback(uint8_t* buffer, uint32_t offset,
    void* p)
{
```



```
    struct fat16_read_callback_arg* arg = p;

    /* skip deleted or empty entries */
    if(buffer[0] == FAT16_DIRENTRY_DELETED || !buffer[0])
        return 1;

    if(arg->entry_cur == arg->entry_num)
    {
        arg->entry_offset = offset;
        arg->byte_count = buffer[11] == 0x0f ?
                          ((buffer[0] & FAT16_DIRENTRY_LFNSEQMASK) + 1)
                              * 32 :
                          32;
        return 0;
    }

    /* if we read a 8.3 entry, we reached a new directory entry */
    if(buffer[11] != 0x0f)
        ++arg->entry_cur;

    return 1;
}

/**
 * \ingroup fat16_fs
 * Callback function for reading a directory entry.
 */
uint8_t fat16_dir_entry_read_callback(uint8_t* buffer, uint32_t offset,
    void* p)
{
    struct fat16_dir_entry_struct* dir_entry = p;

    /* there should not be any deleted or empty entries */
    if(buffer[0] == FAT16_DIRENTRY_DELETED || !buffer[0])
        return 0;

    if(!dir_entry->entry_offset)
        dir_entry->entry_offset = offset;

    switch(fat16_interpret_dir_entry(dir_entry, buffer))
    {
        case 0: /* failure */
            return 0;
        case 1: /* buffer successfully parsed, continue */
            return 1;
        case 2: /* directory entry complete, finish */
            return 0;
    }

    return 0;
}
```

```
/**
 * \ingroup fat16_fs
 * Interprets a raw directory entry and puts the contained
 * information into the directory entry.
 *
 * For a single file there may exist multiple directory
 * entries. All except the last one are lfn entries, which
 * contain parts of the long filename. The last directory
 * entry is a traditional 8.3 style one. It contains all
 * other information like size, cluster, date and time.
 *
 * \param[in,out] dir_entry The directory entry to fill.
 * \param[in] raw_entry A pointer to 32 bytes of raw data.
 * \returns 0 on failure, 1 on success and 2 if the
 *          directory entry is complete.
 */
uint8_t fat16_interpret_dir_entry(struct fat16_dir_entry_struct*
   dir_entry, const uint8_t* raw_entry)
{
    if(!dir_entry || !raw_entry || !raw_entry[0])
        return 0;

    char* long_name = dir_entry->long_name;
    if(raw_entry[11] == 0x0f)
    {
        uint16_t char_offset = ((raw_entry[0] & 0x3f) - 1) * 13;

        if(char_offset + 12 < sizeof(dir_entry->long_name))
        {
            /* Lfn supports unicode, but we do not, for now.
             * So we assume pure ascii and read only every
             * second byte.
             */
            long_name[char_offset + 0] = raw_entry[1];
            long_name[char_offset + 1] = raw_entry[3];
            long_name[char_offset + 2] = raw_entry[5];
            long_name[char_offset + 3] = raw_entry[7];
            long_name[char_offset + 4] = raw_entry[9];
            long_name[char_offset + 5] = raw_entry[14];
            long_name[char_offset + 6] = raw_entry[16];
            long_name[char_offset + 7] = raw_entry[18];
            long_name[char_offset + 8] = raw_entry[20];
            long_name[char_offset + 9] = raw_entry[22];
            long_name[char_offset + 10] = raw_entry[24];
            long_name[char_offset + 11] = raw_entry[28];
            long_name[char_offset + 12] = raw_entry[30];
        }

        return 1;
    }
    else
    {
```



```
        /* if we do not have a long name, take the short one */
        if(long_name[0] == '\0')
        {
            uint8_t i;
            for(i = 0; i < 8; ++i)
            {
                if(raw_entry[i] == ' ')
                    break;
                long_name[i] = raw_entry[i];
            }
            if(long_name[0] == 0x05)
                long_name[0] = (char) FAT16_DIRENTRY_DELETED;

            if(raw_entry[8] != ' ')
            {
                long_name[i++] = '.';

                uint8_t j = 8;
                for(; j < 11; ++j)
                {
                    if(raw_entry[j] != ' ')
                    {
                        long_name[i++] = raw_entry[j];
                    }
                    else
                    {
                        break;
                    }
                }
            }

            long_name[i] = '\0';
        }

        /* extract properties of file and store them within the
           structure */
        dir_entry->attributes = raw_entry[11];
        dir_entry->cluster = ((uint16_t) raw_entry[26]) |
                             ((uint16_t) raw_entry[27] << 8);
        dir_entry->file_size = ((uint32_t) raw_entry[28]) |
                               ((uint32_t) raw_entry[29] << 8) |
                               ((uint32_t) raw_entry[30] << 16) |
                               ((uint32_t) raw_entry[31] << 24);

#if FAT16_DATETIME_SUPPORT
        dir_entry->modification_time = ((uint16_t) raw_entry[22]) |
                                       ((uint16_t) raw_entry[23] << 8);
        dir_entry->modification_date = ((uint16_t) raw_entry[24]) |
                                       ((uint16_t) raw_entry[25] << 8);
#endif

        return 2;
```

```
    }
}

/**
 * \ingroup fat16_file
 * Retrieves the directory entry of a path.
 *
 * The given path may both describe a file or a directory.
 *
 * \param[in] fs The FAT16 filesystem on which to search.
 * \param[in] path The path of which to read the directory entry.
 * \param[out] dir_entry The directory entry to fill.
 * \returns 0 on failure, 1 on success.
 * \see fat16_read_dir
 */
uint8_t fat16_get_dir_entry_of_path(struct fat16_fs_struct* fs, const
   char* path, struct fat16_dir_entry_struct* dir_entry)
{
    if(!fs || !path || path[0] == '\0' || !dir_entry)
        return 0;

    if(path[0] == '/')
        ++path;

    /* begin with the root directory */
    memset(dir_entry, 0, sizeof(*dir_entry));
    dir_entry->attributes = FAT16_ATTRIB_DIR;

    if(path[0] == '\0')
        return 1;

    while(1)
    {
        struct fat16_dir_struct* dd = fat16_open_dir(fs, dir_entry);
        if(!dd)
            break;

        /* extract the next hierarchy we will search for */
        const char* sep_pos = strchr(path, '/');
        if(!sep_pos)
            sep_pos = path + strlen(path);
        uint8_t length_to_sep = sep_pos - path;

        /* read directory entries */
        while(fat16_read_dir(dd, dir_entry))
        {
            /* check if we have found the next hierarchy */
            if((strlen(dir_entry->long_name) != length_to_sep ||
                strncmp(path, dir_entry->long_name, length_to_sep) !=
                   0))
                continue;
```



```
            fat16_close_dir(dd);
            dd = 0;

            if(path[length_to_sep] == '\0')
                /* we iterated through the whole path and have found
                   the file */
                return 1;

            if(dir_entry->attributes & FAT16_ATTRIB_DIR)
            {
                /* we found a parent directory of the file we are
                   searching for */
                path = sep_pos + 1;
                break;
            }

            /* a parent of the file exists, but not the file itself */
            return 0;
        }

        fat16_close_dir(dd);
    }

    return 0;
}

/**
 * \ingroup fat16_fs
 * Retrieves the next following cluster of a given cluster.
 *
 * Using the filesystem file allocation table, this function returns
 * the number of the cluster containing the data directly following
 * the data within the cluster with the given number.
 *
 * \param[in] fs The filesystem for which to determine the next cluster
    .
 * \param[in] cluster_num The number of the cluster for which to
    determine its successor.
 * \returns The wanted cluster number, or 0 on error.
 */
uint16_t fat16_get_next_cluster(const struct fat16_fs_struct* fs,
   uint16_t cluster_num)
{
    if(!fs || cluster_num < 2)
        return 0;

    /* read appropriate fat entry */
    uint8_t fat_entry[2];
    if(!fs->partition->device_read(fs->header.fat_offset + 2 *
       cluster_num, fat_entry, 2))
        return 0;
```

```
    /* determine next cluster from fat */
    cluster_num = ((uint16_t) fat_entry[0]) |
                  ((uint16_t) fat_entry[1] << 8);

    if(cluster_num == FAT16_CLUSTER_FREE ||
       cluster_num == FAT16_CLUSTER_BAD ||
       (cluster_num >= FAT16_CLUSTER_RESERVED_MIN && cluster_num <=
          FAT16_CLUSTER_RESERVED_MAX) ||
       (cluster_num >= FAT16_CLUSTER_LAST_MIN && cluster_num <=
          FAT16_CLUSTER_LAST_MAX))
        return 0;

    return cluster_num;
}

/**
 * \ingroup fat16_fs
 * Appends a new cluster chain to an existing one.
 *
 * Set cluster_num to zero to create a completely new one.
 *
 * \param[in] fs The file system on which to operate.
 * \param[in] cluster_num The cluster to which to append the new chain.
 * \param[in] count The number of clusters to allocate.
 * \returns 0 on failure, the number of the first new cluster on
    success.
 */
uint16_t fat16_append_clusters(const struct fat16_fs_struct* fs,
   uint16_t cluster_num, uint16_t count)
{
#if FAT16_WRITE_SUPPORT
    if(!fs)
        return 0;

    device_read_t device_read = fs->partition->device_read;
    device_write_t device_write = fs->partition->device_write;
    uint32_t fat_offset = fs->header.fat_offset;
    uint16_t cluster_max = fs->header.fat_size / 2;
    uint16_t cluster_next = 0;
    uint16_t count_left = count;
    uint8_t buffer[2];

    for(uint16_t cluster_new = 0; cluster_new < cluster_max; ++
       cluster_new)
    {
        if(!device_read(fat_offset + 2 * cluster_new, buffer, sizeof(
           buffer)))
            return 0;

        /* check if this is a free cluster */
        if(buffer[0] == (FAT16_CLUSTER_FREE & 0xff) &&
           buffer[1] == ((FAT16_CLUSTER_FREE >> 8) & 0xff))
```



```
        {
            /* allocate cluster */
            if(count_left == count)
            {
                buffer[0] = FAT16_CLUSTER_LAST_MAX & 0xff;
                buffer[1] = (FAT16_CLUSTER_LAST_MAX >> 8) & 0xff;
            }
            else
            {
                buffer[0] = cluster_next & 0xff;
                buffer[1] = (cluster_next >> 8) & 0xff;
            }

            if(!device_write(fat_offset + 2 * cluster_new, buffer,
               sizeof(buffer)))
                break;

            cluster_next = cluster_new;
            if(--count_left == 0)
                break;
        }
    }

    do
    {
        if(count_left > 0)
            break;

        /* We allocated a new cluster chain. Now join
         * it with the existing one.
         */
        if(cluster_num >= 2)
        {
            buffer[0] = cluster_next & 0xff;
            buffer[1] = (cluster_next >> 8) & 0xff;
            if(!device_write(fat_offset + 2 * cluster_num, buffer,
               sizeof(buffer)))
                break;
        }

        return cluster_next;

    } while(0);

    /* No space left on device or writing error.
     * Free up all clusters already allocated.
     */
    fat16_free_clusters(fs, cluster_next);

    return 0;
#else
    return 0;
```

```
#endif
}

/**
 * \ingroup fat16_fs
 * Frees a cluster chain, or a part thereof.
 *
 * Marks the specified cluster and all clusters which are sequentially
 * referenced by it as free. They may then be used again for future
 * file allocations.
 *
 * \note If this function is used for freeing just a part of a cluster
 *       chain, the new end of the chain is not correctly terminated
 *       within the FAT. Use fat16_terminate_clusters() instead.
 *
 * \param[in] fs The filesystem on which to operate.
 * \param[in] cluster_num The starting cluster of the chain which to
    free.
 * \returns 0 on failure, 1 on success.
 * \see fat16_terminate_clusters
 */
uint8_t fat16_free_clusters(const struct fat16_fs_struct* fs, uint16_t
   cluster_num)
{
#if FAT16_WRITE_SUPPORT
    if(!fs || cluster_num < 2)
        return 0;

    uint32_t fat_offset = fs->header.fat_offset;
    uint8_t buffer[2];
    while(cluster_num)
    {
        if(!fs->partition->device_read(fat_offset + 2 * cluster_num,
           buffer, 2))
            return 0;

        /* get next cluster of current cluster before freeing current
           cluster */
        uint16_t cluster_num_next = ((uint16_t) buffer[0]) |
                                    ((uint16_t) buffer[1] << 8);

        if(cluster_num_next == FAT16_CLUSTER_FREE)
            return 1;
        if(cluster_num_next == FAT16_CLUSTER_BAD ||
           (cluster_num_next >= FAT16_CLUSTER_RESERVED_MIN &&
            cluster_num_next <= FAT16_CLUSTER_RESERVED_MAX
           )
          )
            return 0;
        if(cluster_num_next >= FAT16_CLUSTER_LAST_MIN &&
           cluster_num_next <= FAT16_CLUSTER_LAST_MAX)
            cluster_num_next = 0;
```



```
        /* free cluster */
        buffer[0] = FAT16_CLUSTER_FREE & 0xff;
        buffer[1] = (FAT16_CLUSTER_FREE >> 8) & 0xff;
        fs->partition->device_write(fat_offset + 2 * cluster_num,
           buffer, 2);

        /* We continue in any case here, even if freeing the cluster
           failed.
         * The cluster is lost, but maybe we can still free up some
            later ones.
         */

        cluster_num = cluster_num_next;
    }

    return 1;
#else
    return 0;
#endif
}

/**
 * \ingroup fat16_fs
 * Frees a part of a cluster chain and correctly terminates the rest.
 *
 * Marks the specified cluster as the new end of a cluster chain and
 * frees all following clusters.
 *
 * \param[in] fs The filesystem on which to operate.
 * \param[in] cluster_num The new end of the cluster chain.
 * \returns 0 on failure, 1 on success.
 * \see fat16_free_clusters
 */
uint8_t fat16_terminate_clusters(const struct fat16_fs_struct* fs,
   uint16_t cluster_num)
{
#if FAT16_WRITE_SUPPORT
    if(!fs || cluster_num < 2)
        return 0;

    /* fetch next cluster before overwriting the cluster entry */
    uint16_t cluster_num_next = fat16_get_next_cluster(fs, cluster_num)
       ;

    /* mark cluster as the last one */
    uint8_t buffer[2];
    buffer[0] = FAT16_CLUSTER_LAST_MAX & 0xff;
    buffer[1] = (FAT16_CLUSTER_LAST_MAX >> 8) & 0xff;
    if(!fs->partition->device_write(fs->header.fat_offset + 2 *
       cluster_num, buffer, 2))
        return 0;
```

```
    /* free remaining clusters */
    if(cluster_num_next)
        return fat16_free_clusters(fs, cluster_num_next);
    else
        return 1;
#else
    return 0;
#endif
}

/**
 * \ingroup fat16_fs
 * Clears a single cluster.
 *
 * The complete cluster is filled with zeros.
 *
 * \param[in] fs The filesystem on which to operate.
 * \param[in] cluster_num The cluster to clear.
 * \returns 0 on failure, 1 on success.
 */
uint8_t fat16_clear_cluster(const struct fat16_fs_struct* fs, uint16_t
   cluster_num)
{
#if FAT16_WRITE_SUPPORT
    if(cluster_num < 2)
        return 0;

    uint32_t cluster_offset = fs->header.cluster_zero_offset +
                              (uint32_t) (cluster_num - 2) * fs->header
                                  .cluster_size;
    uint8_t zero[16];
    return fs->partition->device_write_interval(cluster_offset,
                                                zero,
                                                fs->header.cluster_size
                                                    ,
                                                fat16_clear_cluster_callback
                                                    ,
                                                0
                                               );
#else
    return 0;
#endif
}

/**
 * \ingroup fat16_fs
 * Callback function for clearing a cluster.
 */
uint16_t fat16_clear_cluster_callback(uint8_t* buffer, uint32_t offset,
    void* p)
{
```



```
#if FAT16_WRITE_SUPPORT
    memset(buffer, 0, 16);
    return 16;
#else
    return 0;
#endif
}

/**
 * \ingroup fat16_file
 * Opens a file on a FAT16 filesystem.
 *
 * \param[in] fs The filesystem on which the file to open lies.
 * \param[in] dir_entry The directory entry of the file to open.
 * \returns The file handle, or 0 on failure.
 * \see fat16_close_file
 */
struct fat16_file_struct* fat16_open_file(struct fat16_fs_struct* fs,
   const struct fat16_dir_entry_struct* dir_entry)
{
    if(!fs || !dir_entry || (dir_entry->attributes & FAT16_ATTRIB_DIR))
        return 0;

#if USE_DYNAMIC_MEMORY
    struct fat16_file_struct* fd = malloc(sizeof(*fd));
    if(!fd)
        return 0;
#else
    struct fat16_file_struct* fd = fat16_file_handlers;
    uint8_t i;
    for(i = 0; i < FAT16_FILE_COUNT; ++i)
    {
        if(!fd->fs)
            break;

        ++fd;
    }
    if(i >= FAT16_FILE_COUNT)
        return 0;
#endif

    memcpy(&fd->dir_entry, dir_entry, sizeof(*dir_entry));
    fd->fs = fs;
    fd->pos = 0;
    fd->pos_cluster = dir_entry->cluster;

    return fd;
}

/**
 * \ingroup fat16_file
 * Closes a file.
```

```
 *
 * \param[in] fd The file handle of the file to close.
 * \see fat16_open_file
 */
void fat16_close_file(struct fat16_file_struct* fd)
{
    if(fd)
#if USE_DYNAMIC_MEMORY
        free(fd);
#else
        fd->fs = 0;
#endif
}

/**
 * \ingroup fat16_file
 * Reads data from a file.
 *
 * The data requested is read from the current file location.
 *
 * \param[in] fd The file handle of the file from which to read.
 * \param[out] buffer The buffer into which to write.
 * \param[in] buffer_len The amount of data to read.
 * \returns The number of bytes read, 0 on end of file, or -1 on
    failure.
 * \see fat16_write_file
 */
int16_t fat16_read_file(struct fat16_file_struct* fd, uint8_t* buffer,
   uint16_t buffer_len)
{
    /* check arguments */
    if(!fd || !buffer || buffer_len < 1)
        return -1;

    /* determine number of bytes to read */
    if(fd->pos + buffer_len > fd->dir_entry.file_size)
        buffer_len = fd->dir_entry.file_size - fd->pos;
    if(buffer_len == 0)
        return 0;

    uint16_t cluster_size = fd->fs->header.cluster_size;
    uint16_t cluster_num = fd->pos_cluster;
    uint16_t buffer_left = buffer_len;
    uint16_t first_cluster_offset = fd->pos % cluster_size;

    /* find cluster in which to start reading */
    if(!cluster_num)
    {
        cluster_num = fd->dir_entry.cluster;

        if(!cluster_num)
        {
```



```
            if(!fd->pos)
                return 0;
            else
                return -1;
        }

        if(fd->pos)
        {
            uint32_t pos = fd->pos;
            while(pos >= cluster_size)
            {
                pos -= cluster_size;
                cluster_num = fat16_get_next_cluster(fd->fs,
                   cluster_num);
                if(!cluster_num)
                    return -1;
            }
        }
    }

    /* read data */
    do
    {
        /* calculate data size to copy from cluster */
        uint32_t cluster_offset = fd->fs->header.cluster_zero_offset +
                                  (uint32_t) (cluster_num - 2) *
                                     cluster_size +
                                     first_cluster_offset;
        uint16_t copy_length = cluster_size - first_cluster_offset;
        if(copy_length > buffer_left)
            copy_length = buffer_left;

        /* read data */
        if(!fd->fs->partition->device_read(cluster_offset, buffer,
           copy_length))
            return buffer_len - buffer_left;

        /* calculate new file position */
        buffer += copy_length;
        buffer_left -= copy_length;
        fd->pos += copy_length;

        if(first_cluster_offset + copy_length >= cluster_size)
        {
            /* we are on a cluster boundary, so get the next cluster */
            if((cluster_num = fat16_get_next_cluster(fd->fs,
               cluster_num)))
            {
                first_cluster_offset = 0;
            }
            else
            {
```

```
                fd->pos_cluster = 0;
                return buffer_len - buffer_left;
            }
        }

        fd->pos_cluster = cluster_num;

    } while(buffer_left > 0); /* check if we are done */

    return buffer_len;
}

/**
 * \ingroup fat16_file
 * Writes data to a file.
 *
 * The data is written to the current file location.
 *
 * \param[in] fd The file handle of the file to which to write.
 * \param[in] buffer The buffer from which to read the data to be
    written.
 * \param[in] buffer_len The amount of data to write.
 * \returns The number of bytes written, 0 on disk full, or -1 on
    failure.
 * \see fat16_read_file
 */
int16_t fat16_write_file(struct fat16_file_struct* fd, const uint8_t*
   buffer, uint16_t buffer_len)
{
#if FAT16_WRITE_SUPPORT
    /* check arguments */
    if(!fd || !buffer || buffer_len < 1)
        return -1;
    if(fd->pos > fd->dir_entry.file_size)
        return -1;

    uint16_t cluster_size = fd->fs->header.cluster_size;
    uint16_t cluster_num = fd->pos_cluster;
    uint16_t buffer_left = buffer_len;
    uint16_t first_cluster_offset = fd->pos % cluster_size;

    /* find cluster in which to start writing */
    if(!cluster_num)
    {
        cluster_num = fd->dir_entry.cluster;

        if(!cluster_num)
        {
            if(!fd->pos)
            {
                /* empty file */
```



```
                fd->dir_entry.cluster = cluster_num =
                    fat16_append_clusters(fd->fs, 0, 1);
                if(!cluster_num)
                    return -1;
            }
            else
            {
                return -1;
            }
        }

        if(fd->pos)
        {
            uint32_t pos = fd->pos;
            uint16_t cluster_num_next;
            while(pos >= cluster_size)
            {
                pos -= cluster_size;
                cluster_num_next = fat16_get_next_cluster(fd->fs,
                    cluster_num);
                if(!cluster_num_next && pos == 0)
                    /* the file exactly ends on a cluster boundary, and
                        we append to it */
                    cluster_num_next = fat16_append_clusters(fd->fs,
                        cluster_num, 1);
                if(!cluster_num_next)
                    return -1;

                cluster_num = cluster_num_next;
            }
        }
    }

    /* write data */
    do
    {
        /* calculate data size to write to cluster */
        uint32_t cluster_offset = fd->fs->header.cluster_zero_offset +
                                  (uint32_t) (cluster_num - 2) *
                                     cluster_size +
                                     first_cluster_offset;
        uint16_t write_length = cluster_size - first_cluster_offset;
        if(write_length > buffer_left)
            write_length = buffer_left;

        /* write data which fits into the current cluster */
        if(!fd->fs->partition->device_write(cluster_offset, buffer,
            write_length))
            break;

        /* calculate new file position */
        buffer += write_length;
```

```
        buffer_left -= write_length;
        fd->pos += write_length;

        if(first_cluster_offset + write_length >= cluster_size)
        {
            /* we are on a cluster boundary, so get the next cluster */
            uint16_t cluster_num_next = fat16_get_next_cluster(fd->fs,
                cluster_num);
            if(!cluster_num_next && buffer_left > 0)
                /* we reached the last cluster, append a new one */
                cluster_num_next = fat16_append_clusters(fd->fs,
                    cluster_num, 1);
            if(!cluster_num_next)
            {
                fd->pos_cluster = 0;
                break;
            }

            cluster_num = cluster_num_next;
            first_cluster_offset = 0;
        }

        fd->pos_cluster = cluster_num;

    } while(buffer_left > 0); /* check if we are done */

    /* update directory entry */
    if(fd->pos > fd->dir_entry.file_size)
    {
        uint32_t size_old = fd->dir_entry.file_size;

        /* update file size */
        fd->dir_entry.file_size = fd->pos;
        /* write directory entry */
        if(!fat16_write_dir_entry(fd->fs, &fd->dir_entry))
        {
            /* We do not return an error here since we actually wrote
             * some data to disk. So we calculate the amount of data
             * we wrote to disk and which lies within the old file size
                .
             */
            buffer_left = fd->pos - size_old;
            fd->pos = size_old;
        }
    }

    return buffer_len - buffer_left;

#else
    return -1;
#endif
}
```



```
/**
 * \ingroup fat16_file
 * Repositions the read/write file offset.
 *
 * Changes the file offset where the next call to fat16_read_file()
 * or fat16_write_file() starts reading/writing.
 *
 * If the new offset is beyond the end of the file,
 * fat16_resize_file() is implicitly called, i.e. the file is
 * expanded. (removed this; we don't want to resize files)
 *
 * The new offset can be given in different ways determined by
 * the \c whence parameter:
 * - \b FAT16_SEEK_SET: \c *offset is relative to the beginning of the
    file.
 * - \b FAT16_SEEK_CUR: \c *offset is relative to the current file
    position.
 * - \b FAT16_SEEK_END: \c *offset is relative to the end of the file.
 *
 * The resulting absolute offset is written to the location the \c
    offset
 * parameter points to.
 *
 * \param[in] fd The file decriptor of the file on which to seek.
 * \param[in,out] offset A pointer to the new offset, as affected by
    the \c whence
 *                  parameter. The function writes the new absolute
    offset
 *                  to this location before it returns.
 * \param[in] whence Affects the way \c offset is interpreted, see
    above.
 * \returns 0 on failure, 1 on success.
 */
uint8_t fat16_seek_file(struct fat16_file_struct* fd, int32_t* offset,
    uint8_t whence)
{
    if(!fd || !offset)
        return 0;

    uint32_t new_pos = fd->pos;
    switch(whence)
    {
        case FAT16_SEEK_SET:
            new_pos = *offset;
            break;
        case FAT16_SEEK_CUR:
            new_pos += *offset;
            break;
        case FAT16_SEEK_END:
            new_pos = fd->dir_entry.file_size + *offset;
            break;
```

```
        default:
            return 0;
    }

    if(new_pos > fd->dir_entry.file_size)// && !fat16_resize_file(fd,
       new_pos))
        return 0;

    fd->pos = new_pos;
    fd->pos_cluster = 0;

    *offset = new_pos;
    return 1;
}

/**
 * \ingroup fat16_file
 * Resizes a file to have a specific size.
 *
 * Enlarges or shrinks the file pointed to by the file descriptor to
    have
 * exactly the specified size.
 *
 * If the file is truncated, all bytes having an equal or larger offset
 * than the given size are lost. If the file is expanded, the
    additional
 * bytes are allocated.
 *
 * \note Please be aware that this function just allocates or
    deallocates disk
 * space, it does not explicitely clear it. To avoid data leakage, this
 * must be done manually.
 *
 * \param[in] fd The file decriptor of the file which to resize.
 * \param[in] size The new size of the file.
 * \returns 0 on failure, 1 on success.
 */
uint8_t fat16_resize_file(struct fat16_file_struct* fd, uint32_t size)
{
#if FAT16_WRITE_SUPPORT
    if(!fd)
        return 0;

    uint16_t cluster_num = fd->dir_entry.cluster;
    uint16_t cluster_size = fd->fs->header.cluster_size;
    uint32_t size_new = size;

    do
    {
        if(cluster_num == 0 && size_new == 0)
            /* the file stays empty */
            break;
```



```
        /* seek to the next cluster as long as we need the space */
        while(size_new > cluster_size)
        {
            /* get next cluster of file */
            uint16_t cluster_num_next = fat16_get_next_cluster(fd->fs,
               cluster_num);
            if(cluster_num_next)
            {
                cluster_num = cluster_num_next;
                size_new -= cluster_size;
            }
            else
            {
                break;
            }
        }

        if(size_new > cluster_size || cluster_num == 0)
        {
            /* Allocate new cluster chain and append
             * it to the existing one, if available.
             */
            uint16_t cluster_count = size_new / cluster_size;
            if((uint32_t) cluster_count * cluster_size < size_new)
                ++cluster_count;
            uint16_t cluster_new_chain = fat16_append_clusters(fd->fs,
               cluster_num, cluster_count);
            if(!cluster_new_chain)
                return 0;

            if(!cluster_num)
            {
                cluster_num = cluster_new_chain;
                fd->dir_entry.cluster = cluster_num;
            }
        }

        /* write new directory entry */
        fd->dir_entry.file_size = size;
        if(size == 0)
            fd->dir_entry.cluster = 0;
        if(!fat16_write_dir_entry(fd->fs, &fd->dir_entry))
            return 0;

        if(size == 0)
        {
            /* free all clusters of file */
            fat16_free_clusters(fd->fs, cluster_num);
        }
        else if(size_new <= cluster_size)
        {
```

```
            /* free all clusters no longer needed */
            fat16_terminate_clusters(fd->fs, cluster_num);
        }

    } while(0);

    /* correct file position */
    if(size < fd->pos)
    {
        fd->pos = size;
        fd->pos_cluster = 0;
    }

    return 1;
#else
    return 0;
#endif
}

/**
 * \ingroup fat16_dir
 * Opens a directory.
 *
 * \param[in] fs The filesystem on which the directory to open resides.
 * \param[in] dir_entry The directory entry which stands for the
    directory to open.
 * \returns An opaque directory descriptor on success, 0 on failure.
 * \see fat16_close_dir
 */
struct fat16_dir_struct* fat16_open_dir(struct fat16_fs_struct* fs,
   const struct fat16_dir_entry_struct* dir_entry)
{
    if(!fs || !dir_entry || !(dir_entry->attributes & FAT16_ATTRIB_DIR)
       )
        return 0;

#if USE_DYNAMIC_MEMORY
    struct fat16_dir_struct* dd = malloc(sizeof(*dd));
    if(!dd)
        return 0;
#else
    struct fat16_dir_struct* dd = fat16_dir_handlers;
    uint8_t i;
    for(i = 0; i < FAT16_DIR_COUNT; ++i)
    {
        if(!dd->fs)
            break;

        ++dd;
    }
    if(i >= FAT16_DIR_COUNT)
        return 0;
```



```
#endif

    memcpy(&dd->dir_entry, dir_entry, sizeof(*dir_entry));
    dd->fs = fs;
    dd->entry_next = 0;

    return dd;
}

/**
 * \ingroup fat16_dir
 * Closes a directory descriptor.
 *
 * This function destroys a directory descriptor which was
 * previously obtained by calling fat16_open_dir(). When this
 * function returns, the given descriptor will be invalid.
 *
 * \param[in] dd The directory descriptor to close.
 * \see fat16_open_dir
 */
void fat16_close_dir(struct fat16_dir_struct* dd)
{
    if(dd)
#if USE_DYNAMIC_MEMORY
        free(dd);
#else
        dd->fs = 0;
#endif
}

/**
 * \ingroup fat16_dir
 * Reads the next directory entry contained within a parent directory.
 *
 * \param[in] dd The descriptor of the parent directory from which to
    read the entry.
 * \param[out] dir_entry Pointer to a buffer into which to write the
    directory entry information.
 * \returns 0 on failure, 1 on success.
 * \see fat16_reset_dir
 */
uint8_t fat16_read_dir(struct fat16_dir_struct* dd, struct
   fat16_dir_entry_struct* dir_entry)
{
    if(!dd || !dir_entry)
        return 0;

    if(dd->dir_entry.cluster == 0)
    {
        /* read entry from root directory */
        if(fat16_read_root_dir_entry(dd->fs, dd->entry_next, dir_entry)
           )
```

```
        {
            ++dd->entry_next;
            return 1;
        }
    }
    else
    {
        /* read entry from a subdirectory */
        if(fat16_read_sub_dir_entry(dd->fs, dd->entry_next, &dd->
           dir_entry, dir_entry))
        {
            ++dd->entry_next;
            return 1;
        }
    }

    /* restart reading */
    dd->entry_next = 0;

    return 0;
}

/**
 * \ingroup fat16_dir
 * Resets a directory handle.
 *
 * Resets the directory handle such that reading restarts
 * with the first directory entry.
 *
 * \param[in] dd The directory handle to reset.
 * \returns 0 on failure, 1 on success.
 * \see fat16_read_dir
 */
uint8_t fat16_reset_dir(struct fat16_dir_struct* dd)
{
    if(!dd)
        return 0;

    dd->entry_next = 0;
    return 1;
}

/**
 * \ingroup fat16_fs
 * Searches for space where to store a directory entry.
 *
 * \param[in] fs The filesystem on which to operate.
 * \param[in] dir_entry The directory entry for which to search space.
 * \returns 0 on failure, a device offset on success.
 */
uint32_t fat16_find_offset_for_dir_entry(const struct fat16_fs_struct*
   fs, const struct fat16_dir_struct* parent, const struct
```



```
   fat16_dir_entry_struct* dir_entry)
{
#if FAT16_WRITE_SUPPORT
    if(!fs || !dir_entry)
        return 0;

    /* search for a place where to write the directory entry to disk */
    uint8_t free_dir_entries_needed = (strlen(dir_entry->long_name) +
       12) / 13 + 1;
    uint8_t free_dir_entries_found = 0;
    uint16_t cluster_num = parent->dir_entry.cluster;
    uint32_t dir_entry_offset = 0;
    uint32_t offset = 0;
    uint32_t offset_to = 0;

    if(cluster_num == 0)
    {
        /* we read/write from the root directory entry */
        offset = fs->header.root_dir_offset;
        offset_to = fs->header.cluster_zero_offset;
        dir_entry_offset = offset;
    }

    while(1)
    {
        if(offset == offset_to)
        {
            if(cluster_num == 0)
                /* We iterated through the whole root directory entry
                 * and could not find enough space for the directory
                    entry.
                 */
                return 0;

            if(offset)
            {
                /* We reached a cluster boundary and have to
                 * switch to the next cluster.
                 */

                uint16_t cluster_next = fat16_get_next_cluster(fs,
                   cluster_num);
                if(!cluster_next)
                {
                    cluster_next = fat16_append_clusters(fs,
                       cluster_num, 1);
                    if(!cluster_next)
                        return 0;

                    /* we appended a new cluster and know it is free */
                    dir_entry_offset = fs->header.cluster_zero_offset +
```

```
                                    (uint32_t) (cluster_next - 2) *
                                       fs->header.cluster_size;

                    /* clear cluster to avoid garbage directory entries
                        */
                    fat16_clear_cluster(fs, cluster_next);

                    break;
                }
                cluster_num = cluster_next;
            }

            offset = fs->header.cluster_zero_offset +
                     (uint32_t) (cluster_num - 2) * fs->header.
                        cluster_size;
            offset_to = offset + fs->header.cluster_size;
            dir_entry_offset = offset;
            free_dir_entries_found = 0;
        }

        /* read next lfn or 8.3 entry */
        uint8_t first_char;
        if(!fs->partition->device_read(offset, &first_char, sizeof(
           first_char)))
            return 0;

        /* check if we found a free directory entry */
        if(first_char == FAT16_DIRENTRY_DELETED || !first_char)
        {
            /* check if we have the needed number of available entries
               */
            ++free_dir_entries_found;
            if(free_dir_entries_found >= free_dir_entries_needed)
                break;

            offset += 32;
        }
        else
        {
            offset += 32;
            dir_entry_offset = offset;
            free_dir_entries_found = 0;
        }
    }

    return dir_entry_offset;

#else
    return 0;
#endif
}
```



```
/**
 * \ingroup fat16_fs
 * Writes a directory entry to disk.
 *
 * \note The file name is not checked for invalid characters.
 *
 * \note The generation of the short 8.3 file name is quite
 * simple. The first eight characters are used for the filename.
 * The extension, if any, is made up of the first three characters
 * following the last dot within the long filename. If the
 * filename (without the extension) is longer than eight characters,
 * the lower byte of the cluster number replaces the last two
 * characters to avoid name clashes. In any other case, it is your
 * responsibility to avoid name clashes.
 *
 * \param[in] fs The filesystem on which to operate.
 * \param[in] dir_entry The directory entry to write.
 * \returns 0 on failure, 1 on success.
 */
uint8_t fat16_write_dir_entry(const struct fat16_fs_struct* fs, struct
   fat16_dir_entry_struct* dir_entry)
{
#if FAT16_WRITE_SUPPORT
    if(!fs || !dir_entry)
        return 0;

#if FAT16_DATETIME_SUPPORT
    {
        uint16_t year;
        uint8_t month;
        uint8_t day;
        uint8_t hour;
        uint8_t min;
        uint8_t sec;

        fat16_get_datetime(&year, &month, &day, &hour, &min, &sec);
        fat16_set_file_modification_date(dir_entry, year, month, day);
        fat16_set_file_modification_time(dir_entry, hour, min, sec);
    }
#endif

    device_write_t device_write = fs->partition->device_write;
    uint32_t offset = dir_entry->entry_offset;
    const char* name = dir_entry->long_name;
    uint8_t name_len = strlen(name);
    uint8_t lfn_entry_count = (name_len + 12) / 13;
    uint8_t buffer[32];

    /* write 8.3 entry */

    /* generate 8.3 file name */
    memset(&buffer[0], ' ', 11);
```

```
    char* name_ext = strrchr(name, '.');
    if(name_ext && *++name_ext)
    {
        uint8_t name_ext_len = strlen(name_ext);
        name_len -= name_ext_len + 1;

        if(name_ext_len > 3)
            name_ext_len = 3;

        memcpy(&buffer[8], name_ext, name_ext_len);
    }

    if(name_len <= 8)
    {
        memcpy(buffer, name, name_len);

        /* For now, we create lfn entries for all files,
         * except the "." and ".." directory references.
         * This is to avoid difficulties with capitalization,
         * as 8.3 filenames allow uppercase letters only.
         *
         * Theoretically it would be possible to leave
         * the 8.3 entry alone if the basename and the
         * extension have no mixed capitalization.
         */
        if(name[0] == '.' &&
           ((name[1] == '.' && name[2] == '\0') ||
            name[1] == '\0')
          )
            lfn_entry_count = 0;
    }
    else
    {
        memcpy(buffer, name, 8);

        /* Minimize 8.3 name clashes by appending
         * the lower byte of the cluster number.
         */
        uint8_t num = dir_entry->cluster & 0xff;

        buffer[6] = (num < 0xa0) ? ('0' + (num >> 4)) : ('a' + (num >>
           4));
        num &= 0x0f;
        buffer[7] = (num < 0x0a) ? ('0' + num) : ('a' + num);
    }
    if(buffer[0] == FAT16_DIRENTRY_DELETED)
        buffer[0] = 0x05;

    /* fill directory entry buffer */
    memset(&buffer[11], 0, sizeof(buffer) - 11);
    buffer[0x0b] = dir_entry->attributes;
#if FAT16_DATETIME_SUPPORT
```



```
    buffer[0x16] = (dir_entry->modification_time >> 0) & 0xff;
    buffer[0x17] = (dir_entry->modification_time >> 8) & 0xff;
    buffer[0x18] = (dir_entry->modification_date >> 0) & 0xff;
    buffer[0x19] = (dir_entry->modification_date >> 8) & 0xff;
#endif
    buffer[0x1a] = (dir_entry->cluster >> 0) & 0xff;
    buffer[0x1b] = (dir_entry->cluster >> 8) & 0xff;
    buffer[0x1c] = (dir_entry->file_size >> 0) & 0xff;
    buffer[0x1d] = (dir_entry->file_size >> 8) & 0xff;
    buffer[0x1e] = (dir_entry->file_size >> 16) & 0xff;
    buffer[0x1f] = (dir_entry->file_size >> 24) & 0xff;

    /* write to disk */
    if(!device_write(offset + (uint32_t) lfn_entry_count * 32, buffer,
       sizeof(buffer)))
        return 0;

    /* calculate checksum of 8.3 name */
    uint8_t checksum = buffer[0];
    for(uint8_t i = 1; i < 11; ++i)
        checksum = ((checksum >> 1) | (checksum << 7)) + buffer[i];

    /* write lfn entries */
    for(uint8_t lfn_entry = lfn_entry_count; lfn_entry > 0; --lfn_entry
       )
    {
        memset(buffer, 0xff, sizeof(buffer));

        /* set file name */
        const char* long_name_curr = name + (lfn_entry - 1) * 13;
        uint8_t i = 1;
        while(i < 0x1f)
        {
            buffer[i++] = *long_name_curr;
            buffer[i++] = 0;

            switch(i)
            {
                case 0x0b:
                    i = 0x0e;
                    break;
                case 0x1a:
                    i = 0x1c;
                    break;
            }

            if(!*long_name_curr++)
                break;
        }

        /* set index of lfn entry */
        buffer[0x00] = lfn_entry;
```

```
        if(lfn_entry == lfn_entry_count)
            buffer[0x00] |= FAT16_DIRENTRY_LFNLAST;

        /* mark as lfn entry */
        buffer[0x0b] = 0x0f;

        /* set 8.3 checksum */
        buffer[0x0d] = checksum;

        /* clear reserved bytes */
        buffer[0x0c] = 0;
        buffer[0x1a] = 0;
        buffer[0x1b] = 0;

        /* write entry */
        device_write(offset, buffer, sizeof(buffer));

        offset += sizeof(buffer);
    }

    return 1;

#else
    return 0;
#endif
}

/**
 * \ingroup fat16_file
 * Creates a file.
 *
 * Creates a file and obtains the directory entry of the
 * new file. If the file to create already exists, the
 * directory entry of the existing file will be returned
 * within the dir_entry parameter.
 *
 * \note The file name is not checked for invalid characters.
 *
 * \note The generation of the short 8.3 file name is quite
 * simple. The first eight characters are used for the filename.
 * The extension, if any, is made up of the first three characters
 * following the last dot within the long filename. If the
 * filename (without the extension) is longer than eight characters,
 * the lower byte of the cluster number replaces the last two
 * characters to avoid name clashes. In any other case, it is your
 * responsibility to avoid name clashes.
 *
 * \param[in] parent The handle of the directory in which to create the
     file.
 * \param[in] file The name of the file to create.
 * \param[out] dir_entry The directory entry to fill for the new file.
 * \returns 0 on failure, 1 on success.
```



```
 * \see fat16_delete_file
 */
uint8_t fat16_create_file(struct fat16_dir_struct* parent, const char*
   file, struct fat16_dir_entry_struct* dir_entry)
{
#if FAT16_WRITE_SUPPORT
    if(!parent || !file || !file[0] || !dir_entry)
        return 0;

    /* check if the file already exists */
    while(1)
    {
        if(!fat16_read_dir(parent, dir_entry))
            break;

        if(strcmp(file, dir_entry->long_name) == 0)
        {
            fat16_reset_dir(parent);
            return 0;
        }
    }

    struct fat16_fs_struct* fs = parent->fs;

    /* prepare directory entry with values already known */
    memset(dir_entry, 0, sizeof(*dir_entry));
    strncpy(dir_entry->long_name, file, sizeof(dir_entry->long_name) -
       1);

    /* find place where to store directory entry */
    if(!(dir_entry->entry_offset = fat16_find_offset_for_dir_entry(fs,
       parent, dir_entry)))
        return 0;

    /* write directory entry to disk */
    if(!fat16_write_dir_entry(fs, dir_entry))
        return 0;

    return 1;

#else
    return 0;
#endif
}

/**
 * \ingroup fat16_file
 * Deletes a file or directory.
 *
 * If a directory is deleted without first deleting its
 * subdirectories and files, disk space occupied by these
 * files will get wasted as there is no chance to release
```

```
 * it and mark it as free.
 *
 * \param[in] fs The filesystem on which to operate.
 * \param[in] dir_entry The directory entry of the file to delete.
 * \returns 0 on failure, 1 on success.
 * \see fat16_create_file
 */
uint8_t fat16_delete_file(struct fat16_fs_struct* fs, struct
   fat16_dir_entry_struct* dir_entry)
{
#if FAT16_WRITE_SUPPORT
    if(!fs || !dir_entry)
        return 0;

    /* get offset of the file's directory entry */
    uint32_t dir_entry_offset = dir_entry->entry_offset;
    if(!dir_entry_offset)
        return 0;

    uint8_t buffer[12];
    while(1)
    {
        /* read directory entry */
        if(!fs->partition->device_read(dir_entry_offset, buffer, sizeof
           (buffer)))
            return 0;

        /* mark the directory entry as deleted */
        buffer[0] = FAT16_DIRENTRY_DELETED;

        /* write back entry */
        if(!fs->partition->device_write(dir_entry_offset, buffer,
           sizeof(buffer)))
            return 0;

        /* check if we deleted the whole entry */
        if(buffer[11] != 0x0f)
            break;

        dir_entry_offset += 32;
    }

    /* We deleted the directory entry. The next thing to do is
     * marking all occupied clusters as free.
     */
    return (dir_entry->cluster == 0 || fat16_free_clusters(fs,
       dir_entry->cluster));
#else
    return 0;
#endif
}
```



```
/**
 * \ingroup fat16_dir
 * Creates a directory.
 *
 * Creates a directory and obtains its directory entry.
 * If the directory to create already exists, its
 * directory entry will be returned within the dir_entry
 * parameter.
 *
 * \note The notes which apply to fat16_create_file also
 * apply to this function.
 *
 * \param[in] parent The handle of the parent directory of the new
    directory.
 * \param[in] dir The name of the directory to create.
 * \param[out] dir_entry The directory entry to fill for the new
    directory.
 * \returns 0 on failure, 1 on success.
 * \see fat16_delete_dir
 */
uint8_t fat16_create_dir(struct fat16_dir_struct* parent, const char*
   dir, struct fat16_dir_entry_struct* dir_entry)
{
#if FAT16_WRITE_SUPPORT
    if(!parent || !dir || !dir[0] || !dir_entry)
        return 0;

    /* check if the file or directory already exists */
    while(1)
    {
        if(!fat16_read_dir(parent, dir_entry))
            break;

        if(strcmp(dir, dir_entry->long_name) == 0)
        {
            fat16_reset_dir(parent);
            return 0;
        }
    }

    struct fat16_fs_struct* fs = parent->fs;

    /* allocate cluster which will hold directory entries */
    uint16_t dir_cluster = fat16_append_clusters(fs, 0, 1);
    if(!dir_cluster)
        return 0;

    /* clear cluster to prevent bogus directory entries */
    fat16_clear_cluster(fs, dir_cluster);

    memset(dir_entry, 0, sizeof(*dir_entry));
    dir_entry->attributes = FAT16_ATTRIB_DIR;
```

```
    /* create "." directory self reference */
    dir_entry->entry_offset = fs->header.cluster_zero_offset +
                              (uint32_t) (dir_cluster - 2) * fs->header
                                  .cluster_size;
    dir_entry->long_name[0] = '.';
    dir_entry->cluster = dir_cluster;
    if(!fat16_write_dir_entry(fs, dir_entry))
    {
        fat16_free_clusters(fs, dir_cluster);
        return 0;
    }

    /* create ".." parent directory reference */
    dir_entry->entry_offset += 32;
    dir_entry->long_name[1] = '.';
    dir_entry->cluster = parent->dir_entry.cluster;
    if(!fat16_write_dir_entry(fs, dir_entry))
    {
        fat16_free_clusters(fs, dir_cluster);
        return 0;
    }

    /* fill directory entry */
    strncpy(dir_entry->long_name, dir, sizeof(dir_entry->long_name) -
       1);
    dir_entry->cluster = dir_cluster;

    /* find place where to store directory entry */
    if(!(dir_entry->entry_offset = fat16_find_offset_for_dir_entry(fs,
       parent, dir_entry)))
    {
        fat16_free_clusters(fs, dir_cluster);
        return 0;
    }

    /* write directory to disk */
    if(!fat16_write_dir_entry(fs, dir_entry))
    {
        fat16_free_clusters(fs, dir_cluster);
        return 0;
    }

    return 1;

#else
    return 0;
#endif
}

/**
 * \ingroup fat16_dir
```



```
 * Deletes a directory.
 *
 * This is just a synonym for fat16_delete_file().
 * If a directory is deleted without first deleting its
 * subdirectories and files, disk space occupied by these
 * files will get wasted as there is no chance to release
 * it and mark it as free.
 *
 * \param[in] fs The filesystem on which to operate.
 * \param[in] dir_entry The directory entry of the directory to delete.
 * \returns 0 on failure, 1 on success.
 * \see fat16_create_dir
 */
#ifdef DOXYGEN
uint8_t fat16_delete_dir(struct fat16_fs_struct* fs, struct
   fat16_dir_entry_struct* dir_entry);
#endif

/**
 * \ingroup fat16_file
 * Returns the modification date of a file.
 *
 * \param[in] dir_entry The directory entry of which to return the
    modification date.
 * \param[out] year The year the file was last modified.
 * \param[out] month The month the file was last modified.
 * \param[out] day The day the file was last modified.
 */
void fat16_get_file_modification_date(const struct
   fat16_dir_entry_struct* dir_entry, uint16_t* year, uint8_t* month,
   uint8_t* day)
{
#if FAT16_DATETIME_SUPPORT
    if(!dir_entry)
        return;

    *year = 1980 + ((dir_entry->modification_date >> 9) & 0x7f);
    *month = (dir_entry->modification_date >> 5) & 0x0f;
    *day = (dir_entry->modification_date >> 0) & 0x1f;
#endif
}

/**
 * \ingroup fat16_file
 * Returns the modification time of a file.
 *
 * \param[in] dir_entry The directory entry of which to return the
    modification time.
 * \param[out] hour The hour the file was last modified.
 * \param[out] min The min the file was last modified.
 * \param[out] sec The sec the file was last modified.
 */
```

```
void fat16_get_file_modification_time(const struct
    fat16_dir_entry_struct* dir_entry, uint8_t* hour, uint8_t* min,
    uint8_t* sec)
{
#if FAT16_DATETIME_SUPPORT
    if(!dir_entry)
        return;

    *hour = (dir_entry->modification_time >> 11) & 0x1f;
    *min = (dir_entry->modification_time >> 5) & 0x3f;
    *sec = ((dir_entry->modification_time >> 0) & 0x1f) * 2;
#endif
}

/**
 * \ingroup fat16_file
 * Sets the modification time of a date.
 *
 * \param[in] dir_entry The directory entry for which to set the
    modification date.
 * \param[in] year The year the file was last modified.
 * \param[in] month The month the file was last modified.
 * \param[in] day The day the file was last modified.
 */
void fat16_set_file_modification_date(struct fat16_dir_entry_struct*
    dir_entry, uint16_t year, uint8_t month, uint8_t day)
{
#if FAT16_WRITE_SUPPORT
#if FAT16_DATETIME_SUPPORT
    if(!dir_entry)
        return;

    dir_entry->modification_date =
        ((year - 1980) << 9) |
        ((uint16_t) month << 5) |
        ((uint16_t) day << 0);
#endif
#endif
}

/**
 * \ingroup fat16_file
 * Sets the modification time of a file.
 *
 * \param[in] dir_entry The directory entry for which to set the
    modification time.
 * \param[in] hour The year the file was last modified.
 * \param[in] min The month the file was last modified.
 * \param[in] sec The day the file was last modified.
 */
void fat16_set_file_modification_time(struct fat16_dir_entry_struct*
    dir_entry, uint8_t hour, uint8_t min, uint8_t sec)
```



```
{
#if FAT16_WRITE_SUPPORT
#if FAT16_DATETIME_SUPPORT
    if(!dir_entry)
        return;

    dir_entry->modification_time =
        ((uint16_t) hour << 11) |
        ((uint16_t) min << 5) |
        ((uint16_t) sec >> 1) ;
#endif
#endif
}

/**
 * \ingroup fat16_fs
 * Returns the amount of total storage capacity of the filesystem in
    bytes.
 *
 * \param[in] fs The filesystem on which to operate.
 * \returns 0 on failure, the filesystem size in bytes otherwise.
 */
uint32_t fat16_get_fs_size(const struct fat16_fs_struct* fs)
{
    if(!fs)
        return 0;

    return (fs->header.fat_size / 2 - 2) * fs->header.cluster_size;
}

/**
 * \ingroup fat16_fs
 * Returns the amount of free storage capacity on the filesystem in
    bytes.
 *
 * \note As the FAT16 filesystem is cluster based, this function does
    not
 *       return continuous values but multiples of the cluster size.
 *
 * \param[in] fs The filesystem on which to operate.
 * \returns 0 on failure, the free filesystem space in bytes otherwise.
 */
uint32_t fat16_get_fs_free(const struct fat16_fs_struct* fs)
{
    if(!fs)
        return 0;

    uint8_t fat[32];
    struct fat16_usage_count_callback_arg count_arg;
    count_arg.cluster_count = 0;
    count_arg.buffer_size = sizeof(fat);
```

```
    uint32_t fat_offset = fs->header.fat_offset;
    uint32_t fat_size = fs->header.fat_size;
    while(fat_size > 0)
    {
        uint16_t length = UINT16_MAX - 1;
        if(fat_size < length)
            length = fat_size;

        if(!fs->partition->device_read_interval(fat_offset,
                                                fat,
                                                sizeof(fat),
                                                length,
                                                fat16_get_fs_free_callback
                                                    ,
                                                &count_arg
                                               )
          )
            return 0;

        fat_offset += length;
        fat_size -= length;
    }

    return (uint32_t) count_arg.cluster_count * fs->header.cluster_size
       ;
}

/**
 * \ingroup fat16_fs
 * Callback function used for counting free clusters.
 */
uint8_t fat16_get_fs_free_callback(uint8_t* buffer, uint32_t offset,
   void* p)
{
    struct fat16_usage_count_callback_arg* count_arg = (struct
       fat16_usage_count_callback_arg*) p;
    uint8_t buffer_size = count_arg->buffer_size;

    for(uint8_t i = 0; i < buffer_size; i += 2)
    {
        if((((uint16_t) buffer[1] << 8) | ((uint16_t) buffer[0] << 0))
           == FAT16_CLUSTER_FREE)
            ++(count_arg->cluster_count);

        buffer += 2;
    }

    return 1;
}

uint32_t fat16_get_file_size(const struct fat16_file_struct *fd)
{
```



```
  return fd->dir_entry.file_size;
}
```

Listing G.35: src/fifo.c

```
#include <avr/interrupt.h>
#include <stdlib.h>
#include <string.h>

#include "assert.h"
#include "fifo.h"
#include "lcd.h"
#include "time.h"


/* This function allocates a fifo. The memory necessary for the fifo
 * is allocated with malloc.
 *
 * Parameters:
 *  size            The size of the buffer for the fifo.
 *
 * Returns:
 *  The fifo object.
 */
fifo_t *fifo_alloc(uint16_t size)
{
  fifo_t *fifo;

  fifo = malloc(sizeof(fifo_t) + size);

  fifo->size = size;
  fifo->available = 0;

  fifo->read_pos = 0;
  fifo->write_pos = 0;

  fifo->readable = 0;
  fifo->writeable = 1;

  fifo->readable_notify = NULL;
  fifo->readable_notify_data = NULL;
  fifo->readable_thread = NULL;

  fifo->writeable_notify = NULL;
  fifo->writeable_notify_data = NULL;
  fifo->writeable_thread = NULL;

  return fifo;
}


/* Set the notifier function which is called when the thread becomes
 * readable.
```

```
 *
 * Interrupts must be disabled if there is a possibility that the fifo
 * will become readable during this function call.
 *
 * Parameters:
 *  fifo            The fifo we should set the notifier on.
 *  notify          The notifier which will be called when the fifo
 *                  becomes readable.
 *  data            Argument which will be passed to the notifier
    function.
 */
void fifo_set_notify_readable(fifo_t *fifo, fifo_notify_t notify, void
   *data)
{
  fifo->readable_notify = notify;
  fifo->readable_notify_data = data;
}


/* Set the notifier function which is called when the thread becomes
 * writeable.
 *
 * Interrupts must be disabled if there is a possibility that the fifo
 * will become writeable during this function call.
 *
 * Parameters:
 *  fifo            The fifo we should set the notifier on.
 *  notify          The notifier which will be called when the fifo
 *                  becomes writeable.
 *  data            Argument which will be passed to the notifier
    function.
 */
void fifo_set_notify_writeable(fifo_t *fifo, fifo_notify_t notify, void
    *data)
{
  fifo->writeable_notify = notify;
  fifo->writeable_notify_data = data;
}


/* This function is used to freeze the current thread until the fifo
   becomes
 * readable/writeable, or until the specified time is reached.
 *
 * Interrupts must be disabled when calling this function.
 *
 * Parameters:
 *  thread_waker    Pointer to either readable_thread or
    writeable_thread in
 *                  the fifo we should wait on.
 *  wake_when       The time the thread should be unfrozen if the fifo
    doesn't
```



```
 *                  become readable/writeable.
 */
static void fifo_wait(thread_t **thread_waker, uint32_t wake_when)
{
  time_wake_element_t te;

  /* Mark that the current thread should be unfrozen when the fifo
     becomes
   * readable/writeable.
   */
  *thread_waker = thread_current;

  /* Mark that the current thread should be unfrozen at the specified
     time. */
  time_add_wake(&te, wake_when);

  /* Freeze the thread. */
  ASSERT_INTERRUPTS_OFF();
  thread_wait();
  ASSERT_INTERRUPTS_OFF();
  //lcd_putchar('S');

  /* Remove the timeout. */
  time_remove_wake(&te);

  /* Remove the unfreezing on the specified event. */
  *thread_waker = NULL;
}


/* This function updates the readable and writeable state of the fifo.
 *
 * If the readable flag is enabled, then it will call the readable
 * notifier, and wake the thread waiting for the fifo to become
    readable.
 *
 * If the writeable flag is enabled, then it will call the writeable
 * notifier, and wake the thread waiting for the fifo to become
    writeable.
 *
 * Interrupts must be disabled when calling this function.
 *
 * Parameters:
 *  fifo            The fifo where the readable and writeable flags
 *                  should be updated.
 */
static void fifo_update_state(fifo_t *fifo)
{
  uint8_t prev_readable;
  uint8_t prev_writeable;

  /* Store the previous readable and writeable state. */
```

```
  prev_readable = fifo->readable;
  prev_writeable = fifo->writeable;

  /* Check if the fifo is readable. */
  if(fifo->available) {
    fifo->readable = 1;
  } else {
    fifo->readable = 0;
  }

  /* Check if the fifo is writeable. */
  if(fifo->available < fifo->size) {
    fifo->writeable = 1;
  } else {
    fifo->writeable = 0;
  }


  /* Call the readable notifier if the fifo became readable. */
  if(fifo->readable && !prev_readable && fifo->readable_notify != NULL)
      {
    (*fifo->readable_notify)(fifo, fifo->readable_notify_data);
  }

  /* Wake the thread which is waiting for the fifo to become readable.
     */
  if(fifo->readable && fifo->readable_thread) {
    thread_wake(fifo->readable_thread);
    fifo->readable_thread = NULL;
    //lcd_putchar('W');
  }

  /* Call the writeable notifier if the fifo became writeable. */
  if(fifo->writeable && !prev_writeable && fifo->writeable_notify !=
     NULL) {
    (*fifo->writeable_notify)(fifo, fifo->writeable_notify_data);
  }

  /* Wake the thread which is waiting for the fifo to become writeable.
      */
  if(fifo->writeable && fifo->writeable_thread) {
    thread_wake(fifo->writeable_thread);
  }
}


/* Helper function to write to the fifo. The writing will wrap around
    to
 * the beginning if it is neccesary. The length must limited before
    calling
 * this function.
 *
```



```
 * This function must be called with interrupts disabled.
 *
 * Parameters:
 *   fifo             The fifo which should be written to.
 *   data             The data which should be written.
 *   length           The length of the data.
 */
static void fifo_write_helper(fifo_t *fifo, const uint8_t *data,
                              uint16_t length)
{
  uint16_t len1;
  uint16_t len2;

  if(fifo->write_pos + length > fifo->size) {
    len1 = fifo->size - fifo->write_pos;
    len2 = length - len1;
  } else {
    len1 = length;
    len2 = 0;
  }

  memcpy(&fifo->data[fifo->write_pos], data, len1);
  memcpy(fifo->data, &data[len1], len2);

  fifo->write_pos += length;
  if(fifo->write_pos >= fifo->size) {
    fifo->write_pos -= fifo->size;
  }
}


/* This function writes up to length bytes to the specified fifo. It
    will
 * always return immediately.
 *
 * Interrupts must be disabled when calling this function.
 *
 * Parameters:
 *   fifo             The fifo we should write to.
 *   data             The buffer with the data which should be written.
 *   length           The maximum number of bytes which should be written
     .
 *
 * Returns:
 *   The number of bytes written. This will return 0 if the fifo is full
     .
 */
uint16_t fifo_write_interrupt(fifo_t *fifo, const void *data, uint16_t
    length)
{
  uint16_t num_written;
```

```
  ASSERT_INTERRUPTS_OFF();

  /* Calculate the number of bytes which will be written to the fifo.
     This
   * may be less than length if the fifo is nearly full.
   */
  if(length <= fifo->size - fifo->available ) {
    /* We have space for length bytes. */
    num_written = length;
  } else {
    /* We don't have space for length bytes. */
    num_written = fifo->size - fifo->available;
  }

  /* Copy the input data to the fifo. */
  fifo_write_helper(fifo, (const uint8_t *)data, num_written);

  /* Increase the number of bytes available in the fifo. */
  fifo->available += num_written;

  /* Update the readable and writeable state of the fifo. */
  fifo_update_state(fifo);

  /* Return the number of bytes written. */
  return num_written;
}


/* This function writes up to length bytes to the specified fifo. It
   will
 * return as soon as some data is written, or when we have a timeout.
 * Pass FIFO_INFINITE as timeout to disable the timeout.
 *
 * Interrupts must be enabled when calling this function.
 *
 * Parameters:
 *  fifo            The fifo we should write to.
 *  data            The buffer with the data which should be written.
 *  length          The maximum number of bytes we should write.
 *  timeout         The maximum time we should attempt to write data,
    in
 *                  milliseconds. Pass FIFO_INFINITE to wait forever.
 *
 * Returns:
 *  The number of bytes written, or 0 if we have a timeout.
 */
uint16_t fifo_write(fifo_t *fifo, const void *data, uint16_t length,
                    uint16_t timeout)
{
  uint32_t wake_when;
  uint16_t num_written;
```



```
  ASSERT_INTERRUPTS_ON();

  /* Calculate the timeout. */
  if(timeout != FIFO_INFINITE) {
    wake_when = time_get() + timeout;
  } else {
    wake_when = TIME_INFINITE;
  }

  /* Disable interrupts. */
  cli();

  /* Wait for the fifo to be writeble or timeout. */
  while(!fifo->writeable && time_get() <= wake_when) {
    fifo_wait(&fifo->writeable_thread, wake_when);
  }

  /* Attempt to write data. */
  num_written = fifo_write_interrupt(fifo, data, length);

  /* Enable interrupts. */
  sei();

  /* Return the number of bytes which was written. */
  return num_written;
}


/* This function writes data to the specified fifo. It will return when
 * the specified number of bytes is written, or when we have a timeout.
 * Pass FIFO_INFINITE as timeout to disable the timeout.
 *
 * Interrupts must be enabled when calling this function.
 *
 * Parameters:
 *  fifo            The fifo we should write to.
 *  data            Pointer to the buffer with the data which should be
 *                  written.
 *  length          The number of bytes we should write.
 *  timeout         The maximum time we should attempt to write data,
    in
 *                  milliseconds. Pass FIFO_INFINITE to wait forever.
 *
 * Returns:
 *  Number of bytes written. This will always be equal to length unless
     we
 *  have a timeout.
 */
uint16_t fifo_write_full(fifo_t *fifo, const void *data, uint16_t
   length,
                         uint16_t timeout)
{
```

```
  uint32_t wake_when;
  const uint8_t *d;
  uint16_t nw;
  uint16_t pos;

  ASSERT_INTERRUPTS_ON();

  /* Calculate the timeout. */
  if(timeout != FIFO_INFINITE) {
    wake_when = time_get() + timeout;
  } else {
    wake_when = TIME_INFINITE;
  }

  /* We need to be able to address the buffer on byte boundaries. */
  d = (const uint8_t *)data;

  /* The current position in the output buffer (and also the number of
   * bytes written).
   */
  pos = 0;

  /* Disable interrupts. */
  cli();

  do {
    /* Wait for writeable or timeout. */
    while(!fifo->writeable && time_get() <= wake_when) {
      fifo_wait(&fifo->writeable_thread, wake_when);
    }

    /* Attempt to write data. */
    nw = fifo_write_interrupt(fifo, &d[pos], length - pos);

    pos += nw;

    /* Loop until we have written all the data, or we have a timeout.
       */
  } while(pos < length && time_get() <= wake_when);

  /* Enable interrupts. */
  sei();

  /* pos contains the current position in the input buffer, which is
     also
   * the number of bytes we have written.
   */
  return pos;
}
```



```
/* Helper function to read from the fifo. The reading will wrap around
   to
 * the beginning if it is neccesary. The length must limited before
    calling
 * this function.
 *
 * This function must be called with interrupts disabled.
 *
 * Parameters:
 *  fifo            The fifo which should be read from.
 *  data            The buffer for the data which will be read.
 *  length          The length of the buffer.
 */
static void fifo_read_helper(fifo_t *fifo, uint8_t *data, uint16_t
   length)
{
  uint16_t len1;
  uint16_t len2;

  if(fifo->read_pos + length > fifo->size) {
    len1 = fifo->size - fifo->read_pos;
    len2 = length - len1;
  } else {
    len1 = length;
    len2 = 0;
  }

  memcpy(data, &fifo->data[fifo->read_pos], len1);
  memcpy(&data[len1], fifo->data, len2);

  fifo->read_pos += length;
  if(fifo->read_pos >= fifo->size) {
    fifo->read_pos -= fifo->size;
  }
}


/* This function reads up to length bytes from the specified fifo. It
   will
 * always return immediately.
 *
 * Interrupts must be disabled when calling this function.
 *
 * Parameters:
 *  fifo            The fifo we should read from.
 *  data            The buffer where the data should be stored.
 *  length          The maximum number of bytes which should be read.
 *
 * Returns:
 *  The number of bytes read. This will return 0 if the fifo is empty.
 */
uint16_t fifo_read_interrupt(fifo_t *fifo, void *data, uint16_t length)
```

```
{
  uint16_t num_read;

  ASSERT_INTERRUPTS_OFF();

  /* Calculate the number of bytes we can read. This will be less than
   * length unless we have length bytes available in the buffer.
   */
  if(length <= fifo->available) {
    /* We have at least length bytes available in the buffer. */
    num_read = length;
  } else {
    /* We have less than length bytes available in the buffer. */
    num_read = fifo->available;
  }

  /* Copy the data to the output buffer. */
  fifo_read_helper(fifo, data, num_read);

  /* Decrease the number of bytes available in the buffer. */
  fifo->available -= num_read;

  /* Update the buffer state. */
  fifo_update_state(fifo);

  /* Return the number of bytes which was read. */
  return num_read;
}


/* This function reads up to length bytes from the specified fifo. It
   will
 * return as soon as some data is read, or when we have a timeout. Pass
 * FIFO_INFINITE as timeout to disable the timeout.
 *
 * Interrupts must be enabled when calling this function.
 *
 * Parameters:
 *  fifo            The fifo we should read from.
 *  data            The buffer where the data should be stored.
 *  length          The maximum number of bytes we should read.
 *  timeout         The maximum time we should attempt to read data, in
 *                  milliseconds. Pass FIFO_INFINITE to wait forever.
 *
 * Returns:
 *  The number of bytes read, or 0 if we have a timeout.
 */
uint16_t fifo_read(fifo_t *fifo, void *data, uint16_t length, uint16_t
   timeout)
{
  uint32_t wake_when;
  uint16_t num_read;
```



```
  ASSERT_INTERRUPTS_ON();

  /* Calculate the timeout. */
  if(timeout != FIFO_INFINITE) {
    wake_when = time_get() + timeout;
  } else {
    wake_when = TIME_INFINITE;
  }

  /* Disable interrupts. */
  cli();

  /* Wait for the fifo to be readable or timeout. */
  while(!fifo->readable && time_get() <= wake_when) {
    ASSERT_INTERRUPTS_OFF();
    fifo_wait(&fifo->readable_thread, wake_when);
    ASSERT_INTERRUPTS_OFF();
  }

  /* Attempt to read data. */
  num_read = fifo_read_interrupt(fifo, data, length);

  /* Enable interrupts. */
  sei();

  /* Return the number of bytes which was read. */
  return num_read;
}


/* This function reads data from the specified fifo. It will return
   when
 * the specified number of bytes is read, or when we have a timeout.
    Pass
 * FIFO_INFINITE as timeout to disable the timeout.
 *
 * Interrupts must be enabled when calling this function.
 *
 * Parameters:
 *  fifo            The fifo we should read from.
 *  data            Pointer to the buffer where the data should be
   stored.
 *  length          The number of bytes we should read.
 *  timeout         The maximum time we should attempt to read data, in
 *                  milliseconds. Pass FIFO_INFINITE to wait forever.
 *
 * Returns:
 *  Number of bytes read. This will always be equal to length unless we
    have
 *  a timeout.
 */
```

```
uint16_t fifo_read_full(fifo_t *fifo, void *data, uint16_t length,
                        uint16_t timeout)
{
  uint32_t wake_when;
  uint8_t *d;
  uint16_t nr;
  uint16_t pos;

  ASSERT_INTERRUPTS_ON();

  /* Calculate timeout time. */
  if(timeout != FIFO_INFINITE) {
    wake_when = time_get() + timeout;
  } else {
    wake_when = TIME_INFINITE;
  }

  /* We need to be able to address the buffer on byte boundaries. */
  d = (uint8_t *)data;

  /* The current position in the output buffer. */
  pos = 0;

  /* Disable interrupts. */
  cli();

  do {
    /* Wait for readable or timeout. */
    while(!fifo->readable && time_get() <= wake_when) {
      fifo_wait(&fifo->readable_thread, wake_when);
      ASSERT_INTERRUPTS_OFF();
    }

    /* Attempt to read data. */
    nr = fifo_read_interrupt(fifo, &d[pos], length - pos);

    pos += nr;

    /* Loop until we have read all the data, or we have a timeout. */
  } while(pos < length && time_get() <= wake_when);

  /* Enable interrupts. */
  sei();

  /* pos contains the current position in the output buffer, which is
     also
   * the number of bytes we have read.
   */
  return pos;
}
```

Listing G.36: src/fs.c


```
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <avr/pgmspace.h>

#include "sd_raw.h"
#include "fat16.h"
#include "partition.h"
#include "fs.h"
#include "error.h"

static struct fat16_dir_struct* root_dir;
static struct fat16_fs_struct* fs;

static uint8_t find_file_in_dir(struct fat16_fs_struct* fs, struct
    fat16_dir_struct* dd, const char* name, struct
    fat16_dir_entry_struct* dir_entry)
{
  while(fat16_read_dir(dd, dir_entry)) {
    if(strcmp(dir_entry->long_name, name) == 0) {
      fat16_reset_dir(dd);
      return 1;
    }
  }

  return 0;
}

static struct fat16_file_struct* open_file_in_dir(struct
    fat16_fs_struct* fs, struct fat16_dir_struct* dd, const char* name)
{
    struct fat16_dir_entry_struct file_entry;
    if(!find_file_in_dir(fs, dd, name, &file_entry))
        return 0;

    return fat16_open_file(fs, &file_entry);
}

/*
 * Open a file which resides in the root directory of the file
 * system. This is a simple helper function to avoid using the fat16.c
 * interface directly.
 */
struct fat16_file_struct *fs_open_file(const char *filename)
{
  struct fat16_file_struct* fd = open_file_in_dir(fs, root_dir,
     filename);
  if(!fd) {
    fatal_error_P(PSTR("open_file_in_dir failed\n"));
    return NULL;
  }
  return fd;
```

```
}

/*
 * Initialize file system.
 */
void fs_init(void)
{
  printf("fs init\n");
  if (!sd_raw_init()) {
    printf("sd_raw_init failed\n");
    return;
  }
  printf("sd_raw_init done\n");
  struct partition_struct *partition = partition_open(sd_raw_read,
                              sd_raw_read_interval,
                              sd_raw_write,
                              sd_raw_write_interval,
                              0);

  if(!partition) {
    partition = partition_open(sd_raw_read,
                  sd_raw_read_interval,
                  sd_raw_write,
                  sd_raw_write_interval,
                  -1);
    if(!partition) {
      printf("partition_open failed\n");
      return;
    }
  }

  fs = fat16_open(partition);
  if(!fs) {
    printf("fat16_open failed\n");
  }

  struct fat16_dir_entry_struct directory;
  fat16_get_dir_entry_of_path(fs, "/", &directory);

  root_dir = fat16_open_dir(fs, &directory);
  if(!root_dir) {
    printf("opening root directory failed\n");
    return;
  }

  printf("dir listing:\n");
  struct fat16_dir_entry_struct dir_entry;
  while(fat16_read_dir(root_dir, &dir_entry)) {
    printf("%s sz:%d\n", dir_entry.long_name, dir_entry.file_size);
  }

}
```


Listing G.37: src/fs.c

```
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <avr/pgmspace.h>

#include "sd_raw.h"
#include "fat16.h"
#include "partition.h"
#include "fs.h"
#include "error.h"

static struct fat16_dir_struct* root_dir;
static struct fat16_fs_struct* fs;

static uint8_t find_file_in_dir(struct fat16_fs_struct* fs, struct
    fat16_dir_struct* dd, const char* name, struct
    fat16_dir_entry_struct* dir_entry)
{
  while(fat16_read_dir(dd, dir_entry)) {
    if(strcmp(dir_entry->long_name, name) == 0) {
      fat16_reset_dir(dd);
      return 1;
    }
  }

  return 0;
}

static struct fat16_file_struct* open_file_in_dir(struct
    fat16_fs_struct* fs, struct fat16_dir_struct* dd, const char* name)
{
    struct fat16_dir_entry_struct file_entry;
    if(!find_file_in_dir(fs, dd, name, &file_entry))
        return 0;

    return fat16_open_file(fs, &file_entry);
}

/*
 * Open a file which resides in the root directory of the file
 * system. This is a simple helper function to avoid using the fat16.c
 * interface directly.
 */
struct fat16_file_struct *fs_open_file(const char *filename)
{
  struct fat16_file_struct* fd = open_file_in_dir(fs, root_dir,
     filename);
  if(!fd) {
    fatal_error_P(PSTR("open_file_in_dir failed\n"));
    return NULL;
  }
```

```
  return fd;
}

/*
 * Initialize file system.
 */
void fs_init(void)
{
  printf("fs init\n");
  if (!sd_raw_init()) {
    printf("sd_raw_init failed\n");
    return;
  }
  printf("sd_raw_init done\n");
  struct partition_struct *partition = partition_open(sd_raw_read,
                              sd_raw_read_interval,
                              sd_raw_write,
                              sd_raw_write_interval,
                              0);

  if(!partition) {
    partition = partition_open(sd_raw_read,
                  sd_raw_read_interval,
                  sd_raw_write,
                  sd_raw_write_interval,
                  -1);
    if(!partition) {
      printf("partition_open failed\n");
      return;
    }
  }

  fs = fat16_open(partition);
  if(!fs) {
    printf("fat16_open failed\n");
  }

  struct fat16_dir_entry_struct directory;
  fat16_get_dir_entry_of_path(fs, "/", &directory);

  root_dir = fat16_open_dir(fs, &directory);
  if(!root_dir) {
    printf("opening root directory failed\n");
    return;
  }

  printf("dir listing:\n");
  struct fat16_dir_entry_struct dir_entry;
  while(fat16_read_dir(root_dir, &dir_entry)) {
    printf("%s sz:%d\n", dir_entry.long_name, dir_entry.file_size);
  }
```



```
}
```

Listing G.38: src/jtag.c

```
#include <avr/io.h>

#include "jtag.h"

void jtag_disable(void)
{
  uint8_t tmp;
  tmp = MCUCSR | _BV(JTD);
  MCUCSR = tmp;
  MCUCSR = tmp;
}
```

Listing G.39: src/keyboard.c

```
#include <avr/io.h>
#include <stdint.h>

#include "keyboard.h"
#include "sleep.h"

#define KEY_PORT PORTD
#define KEY_PIN PIND
#define KEY_DDR DDRD

#define SLEEP_TIME 50
#define REPEAT_SLEEP_TIME 300

/* char keys[] = { '1', '2', '3', 'A', */
/*       '4', '5', '6', 'B', */
/*       '7', '8', '9', 'C', */
/*       '*', '0', '#', 'D' }; */

char keys[] = { 'A', 'B', 'C', 'D',
        '1', '4', '7', '*',
        '2', '5', '8', '0',
        '3', '6', '9', '#' };


void keyboard_init(void)
{
  KEY_DDR = 0x0F;
}

/*
 * Look for keyboard input infinitely, reporting typed characters to
 * kh, which should be a function of type keyboard_handler_t.
 */
void keyboard_scan_forever(void *kh)
{
```

```
  keyboard_handler_t keyhandler = (keyboard_handler_t)kh;
  keyboard_scan();
  sleep(REPEAT_SLEEP_TIME);
  while (1) {
    int8_t r = keyboard_scan();
    if (r != -1) {
      keyhandler(keyboard_translate(r));
      sleep(REPEAT_SLEEP_TIME);
    } else {
      sleep(SLEEP_TIME);
    }
  }
}

int8_t keyboard_scan(void)
{
  uint8_t i, j, pin;
  for (i = 0; i < 4; i++) {
    KEY_PORT = 0xFF & ~(1<<i);
    pin = ~(KEY_PIN>>4);
    for (j = 0; j < 4; j++)
      if (pin & (1<<j))
    return i*4+j;
  }
  return -1;
}

char keyboard_translate(int8_t keycode)
{
  return keys[keycode];
}
```

Listing G.40: src/keyboard_int.c

```
#include <stdint.h>
#include <stdlib.h>
#include <avr/interrupt.h>

#include "keyboard_int.h"
#include "fifo.h"
#include "fpga.h"
#include "thread.h"
#include "message.h"

static char keys[] = "B654A321D#0*C987";

static fifo_t *keyboard_int_fifo = NULL;
static fifo_t *fifo_translated = NULL;
static keyboard_int_handler_t key_handler;
static thread_t keyboard_int_thread_data;

static void keyboard_int_thread(void *ignored);
```



```
/*
 * Initialize reading of keyboard via interrupts from FPGA. For each
 * key press, the handler function will be called with the character
 * of the pressed key as argument.
 */
void keyboard_int_init(keyboard_int_handler_t handler)
{
  key_handler = handler;
  keyboard_int_fifo = fifo_alloc(10);
  fifo_translated = fifo_alloc(5);
  thread_setup(&keyboard_int_thread_data,
           keyboard_int_thread, NULL);
}

/*
 * Interrupt handler. Puts keyboard data from FPGA (two bytes) into
 * keyboard_int_fifo at each keyboard interrupt.
 */
SIGNAL (SIG_INTERRUPT0)
{
  uint8_t cause;
  uint16_t key_status;

  cause = FPGA_ISR;

  if(cause & _BV(FPGA_INTR_KEYBOARD)) {
    if (keyboard_int_fifo != NULL) {
      key_status = FPGA_KEYBOARD;
      fifo_write_interrupt(keyboard_int_fifo,
               &key_status, 2);
    }
    FPGA_LEDS = FPGA_KEYBOARD_L;
    FPGA_LEDS = FPGA_KEYBOARD_H;
  }

  if(cause & _BV(FPGA_INTR_DES)) {
    FPGA_LEDS = FPGA_CRYPT_DATA_OUT[0];
  }

  if(cause & _BV(FPGA_INTR_BUTTONS)) {
    FPGA_LEDS = FPGA_BUTTONS;
  }
}


void usart_kick(void);

/*
 * Reads data in chunks of two bytes from keyboard_int_fifo, translates
    to
 * chars and sends them in messages to ui
 */
```

```
static void keyboard_int_thread(void *ignored)
{
  uint16_t old_key_status;
  uint16_t new_key_status;
  uint16_t key_status = 0;
  uint8_t i;
  message_t *msg;
  while (1) {
    fifo_read_full(keyboard_int_fifo, &new_key_status, 2, FIFO_INFINITE
       );
    old_key_status = key_status;
    key_status = new_key_status;
    new_key_status &= ~old_key_status;
    for (i = 0; i < 16; i++) {
      if (new_key_status & (1<<i)) {
    msg = malloc(sizeof(message_t));
    msg->type = MSG_KEY_PRESS;
    msg->data.key = keys[i];
    message_put(msg_to_ui, msg);

    /*
    if (keys[i] == '*')
      usart_kick();
    */

    break;
      }
    }
  }
}
```

Listing G.41: src/lcd.c

```
#define F_CPU 8000000UL  // 8 MHz

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdint.h>
#include <string.h>

#include "lcd.h"

/* Which port we should use. */
#define DISP_PORT PORTF
#define DISP_PIN PINF
#define DISP_DDR DDRF

/* Pin configurations. */
#define DISP_RS_PIN 0
#define DISP_RW_PIN 1
#define DISP_E1_PIN 2
#define DISP_E2_PIN 3
```



```
#define DISP_FIRST_DATA_PIN 4


/* This variable holds the current direction the display port is
 * configured for.
 */
static uint8_t lcd_current_dir;

/* This array holds the current content of the display. */
static char display_buffer[160];

/* This variable holds the current cursor position. */
static uint8_t cursor_position;

static void lcd_delay(unsigned us)
{
  while(us > 10) {
    _delay_us(10);
    us -= 10;
  }
  _delay_us(us);
}


/* This function configures the direction of the port connected to the
 * LCD.
 *
 * Parameters:
 *   direction       0 for output, 1 for input
 */
static void lcd_set_port_dir(uint8_t direction)
{
  uint8_t mask;

  /* rs, rw, e1 & e2 are always output pins. */
  mask = _BV(DISP_RS_PIN) | _BV(DISP_RW_PIN)
    | _BV(DISP_E1_PIN) | _BV(DISP_E2_PIN);

  if(direction == 0) {
    /* Output */
    mask |= 0xf << DISP_FIRST_DATA_PIN;
  }

  DISP_DDR = mask;
  lcd_current_dir = direction;
}


/* This function calculates the mask which should be written to the
   display
 * port.
 *
```

```
 * Parameters:
 *  display         0 for the top half, 1 for the bottom half,
 *                  0xff for neither.
 *  rs              Which register should be read/written.
 *  rw              0 to write to the display, 1 to read from the
    display.
 *  data            The data which should be written. Not used if rw =
    1.
 */
static inline uint8_t lcd_calc_mask(uint8_t display, uint8_t rs,
   uint8_t rw,
                                    uint8_t data)
{
  uint8_t mask;

  switch(display) {
  case 0:
    mask = _BV(DISP_E1_PIN);
    break;
  case 1:
    mask = _BV(DISP_E2_PIN);
    break;
  default:
    mask = 0;
  }

  if(rs) {
    mask |= _BV(DISP_RS_PIN);
  }

  if(rw) {
    /* We should read a nibble from the display. */
    mask |= _BV(DISP_RW_PIN);
  } else {
    /* We should write a nibble to the display. */
    mask |= (data & 0xf) << DISP_FIRST_DATA_PIN;
  }

  return mask;
}

/* This function writes a nibble to the display.
 *
 * Parameters:
 *  display         0 for the top display, 1 for the bottom display.
 *  rs              Register which should be written.
 *  data            Data we should send if we should send data.
 */
static void lcd_write_nibble(uint8_t display, uint8_t rs, uint8_t data)
{
  /* Update the port direction if it has changed. */
  if(lcd_current_dir != 0) {
```



```
    lcd_set_port_dir(0);
  }

  /* Set make sure that rs & rw are configured correctly before setting
   * the enable signal. */
  DISP_PORT = lcd_calc_mask(0xff, rs, 0, data);

  /* Wait at least 40 ns.
   * We don't need this delay since each instruction is 125 ns.
   */

  /* Set enable for the correct display. */
  DISP_PORT = lcd_calc_mask(display, rs, 0, data);

  /* Hold enable high for at least 220 ns. */
  _delay_us(1);

  /* Turn the enable signal off. Hold the data valid for at least 20 ns
     . */
  DISP_PORT = lcd_calc_mask(0xff, rs, 0, data);
}


/* This functions writes a byte to the display.
 *
 * Parameters:
 *  display         Which display we should write to. 0 for the top,
 *                  1 for the bottom.
 *  rs              Which register we should write to.
 *  data            The byte we should write.
 */
static void lcd_write(uint8_t display, uint8_t rs, uint8_t data)
{
  uint8_t ie;

  /* Save interrupt enabled register. */
  ie = SREG & _BV(SREG_I);

  /* Disable interrupts. */
  cli();

  /* Send top half. */
  lcd_write_nibble(display, rs, (data >> 4) & 0xf);
  /* Send bottom half. */
  lcd_write_nibble(display, rs, data & 0xf);

  /* Enable interrupts if they were enabled before. */
  if(ie) {
    sei();
  }
}
```

```
/* This function reads a nibble from the display.
 *
 * Parameters:
 *  display         0 for the top display, 1 for the bottom display.
 *  rs              Register which should be read.
 *
 * Returns:
 *  The nibble which was read from the display.
 */
static uint8_t lcd_read_nibble(uint8_t display, uint8_t rs)
{
  uint8_t ret;

  /* Update the port direction if it has changed. */
  if(lcd_current_dir != 1) {
    lcd_set_port_dir(1);
  }

  /* Set make sure that rs & rw are configured correctly before setting
   * the enable signal. */
  DISP_PORT = lcd_calc_mask(0xff, rs, 1, 0);

  /* Wait at least 40 ns.
   * We don't need this delay since each instruction is 125 ns.
   */

  /* Set enable for the correct display. */
  DISP_PORT = lcd_calc_mask(display, rs, 1, 0);

  /* Wait for valid data. */
  _delay_us(1);

  ret = (DISP_PIN >> DISP_FIRST_DATA_PIN) & 0xf;

  /* Turn the enable signal off. Hold for a short while. */
  DISP_PORT = lcd_calc_mask(0xff, rs, 1, 0);
  _delay_us(1);

  return ret;
}


/* This function reads a byte from the display.
 *
 * Parameters:
 *  display         Which display we should read from.
 *  rs              Which register should be read.
 *
 * Returns:
 *  The byte which was read from the given register at the given
    display.
```



```
 */
static uint8_t lcd_read(uint8_t display, uint8_t rs)
{
  uint8_t ie;
  uint8_t ret;

  /* Save interrupt enabled register. */
  ie = SREG & _BV(SREG_I);

  /* Disable interrupts. */
  cli();

  /* Read byte. */
  ret = (lcd_read_nibble(display, rs) << 4) | lcd_read_nibble(display,
     rs);

  /* Enable interrupts if they were enabled before. */
  if(ie) {
    sei();
  }

  return ret;
}


/* This function waits until the display isn't busy.
 *
 * Parameters:
 *  display         Which display we should wait for. 0 for the top,
 *                  1 for the bottom.
 */
static void lcd_wait_busy(uint8_t display)
{
  while(lcd_read(display, 0) & 0x80);
}


/* This function writes a byte to the display, waiting for it
 * to become ready first.
 */
static void lcd_write_wait(uint8_t display, uint8_t rs, uint8_t data)
{
  lcd_wait_busy(display);
  lcd_write(display, rs, data);
}

/* This function selects an address in the character ram.
 */
static void lcd_disp_goto_cgram(uint8_t display, uint8_t address)
{
  lcd_write_wait(display, 0, 0x40 | (address & 0x3f));
}
```

```
/* This function initializes the character ram with our own custom
 * characters.
 */
static void lcd_disp_load_characters(uint8_t display)
{
  lcd_disp_goto_cgram(display, 0);

  /* Load character 1 */
  //lcd_write_wait(display, 1, 0x1f); /* ***** */
  //lcd_write_wait(display, 1, 0x1b); /* ** ** */
  //lcd_write_wait(display, 1, 0x15); /* * * * */
  //lcd_write_wait(display, 1, 0x11); /* *   * */
  //lcd_write_wait(display, 1, 0x11); /* *   * */
  //lcd_write_wait(display, 1, 0x11); /* *   * */
  //lcd_write_wait(display, 1, 0x11); /* *   * */
  //lcd_write_wait(display, 1, 0x1f); /* ***** */
  lcd_write_wait(display, 1, 0x1f); /* ***** */
  lcd_write_wait(display, 1, 0x11); /* *   * */
  lcd_write_wait(display, 1, 0x19); /* **  * */
  lcd_write_wait(display, 1, 0x15); /* * * * */
  lcd_write_wait(display, 1, 0x15); /* * * * */
  lcd_write_wait(display, 1, 0x19); /* **  * */
  lcd_write_wait(display, 1, 0x11); /* *   * */
  lcd_write_wait(display, 1, 0x1f); /* ***** */

  /* Move the cursor to (0,0) in the display ram. */
  lcd_write_wait(display, 0, 0x80);
}


/* This function resets the given lcd display.
 *
 * Parameters:
 *  display         0 for the top display, 1 for the bottom display.
 */
static void lcd_reset_display(uint8_t display)
{
  /* Make sure the display is in 8-bit mode. */
  lcd_write_nibble(display, 0, 0x3);
  lcd_delay(5000);
  lcd_write_nibble(display, 0, 0x3);
  lcd_delay(64);
  lcd_write_nibble(display, 0, 0x3);
  lcd_delay(64);

  /* Set 4-bit mode. */
  lcd_write_nibble(display, 0, 0x2);
  lcd_delay(64);

  /* Enable 2-line mode, display on. */
```



```
  lcd_write(display, 0, 0x2c);
  lcd_wait_busy(display);

  /* Turn display on. Enable blinking cursor */
  lcd_write(display, 0, 0x0F);
  lcd_wait_busy(display);

  /* Clear the display. */
  lcd_write(display, 0, 0x01);
  lcd_wait_busy(display);

  // Set entry to fill data from left to right. */
  lcd_write(display, 0, 0x06);
  lcd_wait_busy(display);

  /* Load our custom characters. */
  lcd_disp_load_characters(display);
}


/* This function writes a character to the specified display.
 *
 * Parameters:
 *  display         Which display we should write to.
 *  character       Which character we should write.
 */
static void lcd_disp_putchar(uint8_t display, char character)
{
  lcd_wait_busy(display);
  lcd_write(display, 1, (uint8_t)character);
}


/* This function moves the cursor of the display to the first position.
 *
 * Parameters:
 *  display         Which the cursor should be moved on.
 */
static void lcd_disp_go_home(uint8_t display)
{
  lcd_wait_busy(display);
  lcd_write(display, 0, 0x02);
}


/* This function moves the cursor on the given display to the specified
 * position. If the position is 255 (0xff), then the cursor will be
 * disabled.
 *
 * Parameters:
 *  display         Which display we shoud set the cursor on.
```

```
 *  pos             The position of the cursor or 255 if the cursor
    should
 *                  be disabled.
 */
static void lcd_disp_goto(uint8_t display, uint8_t pos)
{
  lcd_wait_busy(display);

  if(pos == 255) {
    /* Turn the cursor off. */
    lcd_write(display, 0, 0x0c);
  } else {
    /* Turn the cursor on. */
    lcd_write(display, 0, 0x0f);
  }

  lcd_wait_busy(display);

  /* The second line is from 0x40 to 0x67. */
  if(pos >= 40) {
    pos = pos - 40 + 0x40;
  }

  /* Move the cursor. */
  lcd_write(display, 0, 0x80 | (pos & 0x7f));
}


/* This function resets both displays.
 */
void lcd_init(void)
{
  /* Initialize the port. */
  lcd_set_port_dir(0);

  /* Reset both displays. */
  lcd_reset_display(0);
  lcd_reset_display(1);

  /* Set the cursor position. */
  lcd_goto(0);

  /* Set the display buffer to empty. */
  memset(display_buffer, ' ', 160);
}


/* This function retrieves the current cursor position. The position is
    the
 * index of the next character which will be written.
 *
 * Returns:
```



```
 *  The index of the next character which will be written, or 255 if
    the
 *  cursor is disabled.
 */
uint8_t lcd_get_pos(void)
{
  return cursor_position;
}


/* This function sets the cursor to the given position. The position is
    the
 * index of the next character which will be written.
 *
 * Parameters:
 *  position        The new position of the cursor.
 */
void lcd_goto(uint8_t position)
{
  if(position < 80) {
    lcd_disp_goto(0, position);
    lcd_disp_goto(1, 255);
  } else if(position < 160) {
    lcd_disp_goto(0, 255);
    lcd_disp_goto(1, position - 80);
  } else {
    lcd_disp_goto(0, 255);
    lcd_disp_goto(1, 255);
  }

  cursor_position = position;
}


/* This function scrolls the display down a single line. The cursor
    position
 * in updated afterwards (unless it is 255).
 */
void lcd_scrolldown(void)
{
  /* Copy the bottom three lines up one line. */
  memmove(display_buffer, &display_buffer[40], 120);

  /* Fill the bottom line with spaces. */
  memset(&display_buffer[120], ' ', 40);

  /* Move the cursor if it should be moved. */
  if(cursor_position < 40) {
    cursor_position = 0;
  } else if(cursor_position <= 160) {
    cursor_position -= 40;
  } else {
```

```
    cursor_position = 255;
  }

  lcd_set_data(display_buffer, cursor_position);
}


/* This function prints a character to the display.
 *
 * Parameters:
 *   character       The character we should print to the display.
 */
void lcd_putchar(char character)
{
  if(cursor_position >= 160) {
    /* We are outside of the display. Don't print anything. */
    return;
  }

  switch(character) {
  case '\n':
    /* New line. */
    cursor_position += 40;
    cursor_position -= cursor_position % 40;
    lcd_goto(cursor_position);
    break;
  case '\r':
    /* Return to the beginning of the line. */
    cursor_position -= cursor_position % 40;
    lcd_goto(cursor_position);
    break;

  default:
    /* Print a normal character. */

    /* Update the local buffer. */
    display_buffer[cursor_position] = character;

    if(cursor_position < 80) {
      lcd_disp_putchar(0, character);
      cursor_position++;
    } else if(cursor_position < 160) {
      lcd_disp_putchar(1, character);
      cursor_position++;
    }
  }

  if(cursor_position == 80) {
    /* Cursor just moved over to the second display. Update the cursors
       . */
    lcd_goto(cursor_position);
  } else if(cursor_position == 160) {
```



```
    /* Cursor just moved below the screen. Scroll down. */
    lcd_scrolldown();
  }
}


/* This function updates the display, and sets the cursor at the given
 * position afterwards.
 *
 * Parameters:
 *   data             160 (4x40) character array with the data which
    should
 *                    be shown on the display.
 *   cursor_pos       Current position (in characters) where the cursor
    should
 *                    be. Set to 255 (0xff) to hide the cursor.
 */
void lcd_set_data(const char *data, uint8_t cursor_pos)
{
  uint8_t i;

  /* Reset the cursor position. */
  lcd_disp_go_home(0);
  lcd_disp_go_home(1);

  /* We fill both the top and bottom display at the same time.
   * This should save time.
   */
  for(i = 0; i < 80; i++) {
    lcd_disp_putchar(0, data[i]);
    lcd_disp_putchar(1, data[i + 80]);

    display_buffer[i] = data[i];
    display_buffer[i + 80] = data[i + 80];
  }

  lcd_goto(cursor_pos);
}


/* This function writes a string to the display.
 *
 * Parameters:
 *   str              The string which should be written to the display.
 */
void lcd_putstring(const char *str)
{
  const char *c;
  for(c = str; *c; c++) {
    lcd_putchar(*c);
  }
}
```

Listing G.42: src/memory.c

```
#include <avr/io.h>

#include "memory.h"

/* This function initializes the external memory interface of the
 * atmega128.
 */
void init_xmem(void)
{
  /* We divide the memory into a upper and lower sector. This allows us
      to
   * configure different timings for the upper and lower sector.
   * The upper sector is for memory mapped IO and the lower sector is
      mapped
   * to the memory chip.
   */

  /* Set divide between lower and upper sector at 0xE000. Enable
     maximum wait
   * for both sectors.
   */
  XMCRA = _BV(SRL2) | _BV(SRL1) | _BV(SRL0)
    | _BV(SRW11) | _BV(SRW01) | _BV(SRW00);

  /* Enable bus-keeper flag and the full 60 KiB address space. */
  XMCRB = _BV(XMBK);

  /* Enable the external memory interface. */
  MCUCR |= _BV(SRE) | _BV(SRW10);
}
```

Listing G.43: src/menu.c

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#ifndef NOAVR
#include <avr/pgmspace.h>
#endif

#include "lcd.h"
#include "menu.h"

#define MENU_STACK_SZ 8
#define SF(menu) ((menu)->progmem_strings ? "%S" : "%s")

#ifdef NOAVR
#define printf lcd_printf
void lcd_printf(const char *fmt, ...);
#endif
```



```
static const char menu_str_cancel[] PROGMEM = "cancel";
static char *menu_str_cancel_dyn;

//menu_t menu_stack[MENU_STACK_SZ];
menu_t *menu_stack;
menu_t *current_menu;

char *menu_screen;
int8_t menu_cursor;

uint8_t unread_sms;

void menu_init_menu(menu_t *menu, const char *title, uint8_t
    progmem_strings);
void menu_item_init(menu_item_t *item, const char *title, menu_cb_t
    callback);
void menu_write_header(menu_t *menu);
void menu_write_items(menu_t *menu);
void menu_write_item(menu_t *menu, uint8_t place, menu_item_t item,
    uint8_t num_items);
uint8_t menu_strlen(menu_t *menu, const char *str);
void menu_strcpy(menu_t *menu, char *dest, const char *src);
uint8_t menu_to_lcd_pos(uint8_t pos);

/*
 * Initialize menu system.
 */
void menu_init(void)
{
  menu_str_cancel_dyn = malloc(strlen_P(menu_str_cancel)+1);
  strcpy_P(menu_str_cancel_dyn, menu_str_cancel);

  menu_stack = malloc(sizeof(menu_t)*MENU_STACK_SZ);
  current_menu = menu_stack-1;
}

void menu_set_screen_buffer(char *buf)
{
  menu_screen = buf;
}

void menu_clear_screen(void)
{
  uint8_t i;
  for (i = 0; i < DISPLAY_WIDTH*4; i++)
    menu_screen[i] = ' ';
}

/*
 * Add a new menu to be displayed. The first argument is the title for
 * the menu, the other is 1 is the strings (title and menu item names)
 * in this menu reside in program memory, 0 otherwise.
```

```
 *
 * After calling this function, menu_set_item, menu_add_item and
 * menu_set_abcd_item may be used to add menu items; then
 * menu_setup_done must be called to make the menu display itself.
 */
menu_t *menu_add(const char *title, uint8_t progmem_strings)
{
  if (current_menu+1 == menu_stack+MENU_STACK_SZ) {
    lcd_goto(0);
    printf_P(PSTR("Menu stack overflow\n"));
    while (1) ;
    return NULL;
  }
  current_menu++;
  menu_init_menu(current_menu, title, progmem_strings);
  return current_menu;
}

/*
 * Set the item at position i of a menu. callback is called when the
 * menu item is activated.
 */
void menu_set_item(menu_t *menu, uint8_t i, const char *title,
    menu_cb_t callback)
{
  menu_item_init(&menu->items[i], title, callback);
}

/*
 * Add a menu item (the first time this is called, the menu item for
 * key '1' is set; the second time, for '2', etc.)
 */
void menu_add_item(menu_t *menu, const char *title, menu_cb_t callback)
{
  uint8_t i;
  for (i = 0; i < 9 && menu->items[i].title != NULL; i++) ;
  if (i == 9) i = 0;
  menu_item_init(&menu->items[i], title, callback);
}

/*
 * Set the menu item for an alphabetic key. item is the character of
 * the key, i.e. 'A', 'B', 'C' or 'D'.
 */
void menu_set_abcd_item(menu_t *menu, char item, const char *title,
    menu_cb_t callback)
{
  uint8_t i = item-'A';
  menu_item_init(&menu->abcd_items[i], title, callback);
}

/*
```



```
 * Pop off the current menu, displaying the one below it.
 */
void menu_pop(void)
{
  menu_pop_update(1);
}

/*
 * Pop off the current menu, but update the display only if
 * update_display is true.
 */
void menu_pop_update(uint8_t update_display)
{
  if (current_menu == menu_stack-1) return;
  current_menu--;
  if (current_menu == menu_stack-1) return;
  if (update_display)
    menu_setup_done(current_menu);
  //menu_update_display();
}

/*
 * Initialize a menu.
 */
void menu_init_menu(menu_t *menu, const char *title, uint8_t
    progmem_strings)
{
  menu->title = title;
  menu->progmem_strings = progmem_strings;

  uint8_t i;
  for (i = 0; i < 9; i++) {
    menu->items[i].title = NULL;
    menu->items[i].callback = NULL;
  }
  for (i = 0; i < 4; i++) {
    menu->abcd_items[i].title = NULL;
    menu->abcd_items[i].callback = NULL;
  }
  if (menu != menu_stack)
    menu->abcd_items[2].title = (progmem_strings ?
                menu_str_cancel :
                menu_str_cancel_dyn);

  menu->key_cb = NULL;
}

/*
 * Initialize a menu item.
 */
void menu_item_init(menu_item_t *item, const char *title, menu_cb_t
    callback)
```

```
{
  item->title = title;
  item->callback = callback;
}

/*
 * Indicates that a menu is completely set up and ready to be
 * displayed. Code which wants to display arbitrary text instead of
 * menu items should call this before setting the text (with
 * menu_setstring or menu_setchar), then call menu_update_display.
 */
void menu_setup_done(menu_t *menu)
{
  menu_cursor = -1;
  menu_clear_screen();
  menu_write_header(menu);
  menu_write_items(menu);
  menu_update_display();
}

/*
 * Called when a key is pressed. Activates the item (if any) in the
 * current menu which is bound to this key and calls the menu's key
 * callback function if it is set.
 */
void menu_key_pressed(char key)
{
  uint8_t pos = lcd_get_pos();
  lcd_goto(158);
  printf("%c", key);
  lcd_goto(pos);

  if (current_menu == menu_stack-1) return;

  int8_t num_key = key-'1';
  int8_t alp_key = key-'A';
  if (num_key < 9 && num_key >= 0
      && current_menu->items[num_key].callback != NULL) {
    current_menu->items[num_key].callback(num_key);
    return;
  }
  if (alp_key < 4 && alp_key >= 0) {
    if (current_menu->abcd_items[alp_key].callback != NULL)
      current_menu->abcd_items[alp_key].callback(alp_key);
    if (alp_key == 2 && current_menu != menu_stack)
      menu_pop();
    return;
  }
  if (current_menu->key_cb != 0)
    current_menu->key_cb(key);
}
```



```
/*
 * Update the display from the screen buffer. Should be called after
 * characters are written with menu_setstring or menu_setchar.
 */
void menu_update_display(void)
{
  menu_screen[DISPLAY_WIDTH*2-1] = (unread_sms ? LCD_MESSAGE : ' ');
  lcd_set_data(menu_screen,
           menu_cursor==-1 ? 255 : menu_to_lcd_pos(menu_cursor));
}

/*
 * Translate from positions in the menu content area (first 38 columns
 * of last three rows) to positions in the complete screen.
 */
uint8_t menu_to_lcd_pos(uint8_t pos)
{
  uint8_t lcd_pos = DISPLAY_WIDTH + pos;
  if (pos >= MENU_LINE_WIDTH) lcd_pos += DISPLAY_WIDTH-MENU_LINE_WIDTH;
  if (pos >= 2*MENU_LINE_WIDTH) lcd_pos += DISPLAY_WIDTH-
     MENU_LINE_WIDTH;
  return lcd_pos;
}

/*
 * Write a character to a position in the menu content area. Call
 * menu_update_display after this to make the change affect the
 * display.
 */
void menu_setchar(uint8_t pos, char c)
{
  if (pos >= 3*MENU_LINE_WIDTH)
    return;
  menu_screen[menu_to_lcd_pos(pos)] = c;
}

/*
 * Write a string to the menu content area, starting at the given
 * position. Call menu_update_display after this to make the change
 * affect the display.
 */
void menu_setstring(uint8_t pos, const char *str)
{
  while (*str)
    menu_setchar(pos++, *(str++));
}

/*
 * Set the position of the blinking cursor (in menu content area
 * coordinates), or -1 to disable cursor.
 */
void menu_set_cursor(uint8_t pos)
```

```
{
  menu_cursor = pos;
}

/*
 * Write len characters of text from the menu content area into buf.
 */
void menu_get_text(uint8_t *buf, uint8_t len)
{
  uint8_t i;
  if (len == 0) len = 3*MENU_LINE_WIDTH;
  for (i = 0; i < 3; i++) {
    if (len > MENU_LINE_WIDTH) {
      memcpy(buf+MENU_LINE_WIDTH*i,
         menu_screen+DISPLAY_WIDTH*(i+1),
         MENU_LINE_WIDTH);
      len -= MENU_LINE_WIDTH;
    } else {
      memcpy(buf+MENU_LINE_WIDTH*i,
         menu_screen+DISPLAY_WIDTH*(i+1),
         len);
      break;
    }
  }
}

/*
 * Write the first line of the menu to the screen buffer.
 */
void menu_write_header(menu_t *menu)
{
  int8_t i, pos;
  char sep = '-';

  pos = MENU_LINE_WIDTH-1;

  for (i = 3; i >= 0; i--) {
    if (menu->abcd_items[i].title != NULL) {
      uint8_t len = menu_strlen(menu, menu->abcd_items[i].title);
      menu_screen[pos] = sep;
      pos -= len+3;
      menu_screen[pos+1] = 'A'+i;
      menu_screen[pos+2] = ':';
      menu_strcpy(menu, menu_screen+pos+3, menu->abcd_items[i].title);
    }
  }

  while (pos >= 0)
    menu_screen[pos--] = sep;

  menu_strcpy(menu, menu_screen+1, menu->title);
}
```



```
/*
 * Write the menu items to the screen buffer.
 */
void menu_write_items(menu_t *menu)
{
  uint8_t i, num_items;
  for (num_items = 0; num_items < 9; num_items++)
    if (menu->items[num_items].title == NULL)
      break;
  for (i = 0; i < num_items; i++) {
    menu_write_item(menu, i, menu->items[i], num_items);
  }
}

void menu_write_item(menu_t *menu, uint8_t place, menu_item_t item,
    uint8_t num_items)
{
  uint8_t cols;
  if (num_items <= 3) cols = 1;
  else if (num_items <= 6) cols = 2;
  else cols = 3;
  uint8_t pos = MENU_LINE_WIDTH*(place%3) + (place/3)*(MENU_LINE_WIDTH/
     cols);
  pos = menu_to_lcd_pos(pos);

  menu_screen[pos++] = '1'+place;
  menu_screen[pos++] = ':';
  menu_strcpy(menu, menu_screen+pos, item.title);
}

/*
 * Helper function which calls the approptiate version of strlen for
 * this menu.
 */
uint8_t menu_strlen(menu_t *menu, const char *str)
{
  if (menu->progmem_strings) return strlen_P(str);
  return strlen(str);
}

/*
 * Helper function which calls the approptiate version of strcpy for
 * this menu.
 */
void menu_strcpy(menu_t *menu, char *dest, const char *src)
{
  if (menu->progmem_strings) {
    memcpy_P(dest, src, strlen_P(src));
  } else {
    memcpy(dest, src, strlen(src));
  }
```

```
}

/*
 * Show or remove the unread SMS indicator.
 */
void menu_set_unread_sms(uint8_t status)
{
  unread_sms = status;
  menu_update_display();
}
```

Listing G.44: src/message.c

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/pgmspace.h>

#include "message.h"
#include "fifo.h"

/*
 * Message queues for messages to UI and to modem.
 */
fifo_t *msg_to_ui, *msg_to_modem;

/*
 * Initialize message queues.
 */
void message_init(void)
{
  msg_to_ui = fifo_alloc(10);
  msg_to_modem = fifo_alloc(10);
}

/*
 * Send a message to a message queue.
 */
void message_put(fifo_t *fifo, message_t *m)
{
  fifo_write_full(fifo, &m, sizeof(message_t*), FIFO_INFINITE);
}

/*
 * Send a message with the given type and no content to a queue.
 */
void message_put_empty(fifo_t *fifo, message_type_t mtyp)
{
  message_t *m = malloc(sizeof(message_t));
  m->type = mtyp;
  message_put(fifo, m);
}
```



```
/*
 * Get the first waiting message in the queue. Blocks until a message
 * is available.
 */
message_t *message_get(fifo_t *fifo)
{
  message_t *m;
  fifo_read_full(fifo, &m, sizeof(message_t*), FIFO_INFINITE);
  return m;
}

/*
 * Get the first waiting message in the queue or return NULL after
 * timeout milliseconds.
 */
message_t *message_get_timeout(fifo_t *fifo, uint16_t timeout)
{
  message_t *m;
  uint8_t len = fifo_read_full(fifo, &m, sizeof(message_t*), timeout);
  while (len > 0 && len < sizeof(message_t*)) {
    len += fifo_read_full(fifo, ((uint8_t*)&m)+len, sizeof(message_t*)-
        len, 50);
  }
  if (len == 0)
    return NULL;
  return m;
}

/*
 * Discard all messages until a message of the given type is
 * encountered, then return it.
 */
message_t *message_wait_for(fifo_t *fifo, message_type_t mtyp)
{
  message_t *m;
  while (1) {
    m = message_get(fifo);
    //printf_P(PSTR("got message, type %d, wanted type %d\n"), m->type,
        mtyp);
    if (m->type == mtyp)
      return m;
    free(m);
  }
}
```

Listing G.45: src/modem.c

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```c
#include "delay.h"
#include "modem.h"
#include "sleep.h"
#include "usart.h"
#include "message.h"
#include "error.h"

#define BUFFER_SIZE 255
#define UNREAD_LIST_SZ 64

static char *response_buffer;
static uint8_t *unread_messages;
static uint8_t num_unread;

static uint8_t modem_busy = 0;

/*
 * Set (b=1) or unset (b=0) a flag indicating that the modem is
 * busy. Will wait until the flag is unset before setting it.
 *
 * Except for in the initialization phase (where only the modem thread
 * accesses the modem) all use of the modem should be surrounded by
 * modem_set_busy(1) and modem_set_busy(0).
 */
static void modem_set_busy(uint8_t b)
{
  if (b == 0) {
    modem_busy = 0;
  } else {
    while (modem_busy == 1) sleep(50);
    modem_busy = 1;
    sleep(100); // was 1000
  }
}

/*
 * Read one byte of data from the modem.
 */
static uint8_t modem_read_byte(void)
{
  uint16_t nr;
  uint8_t buffer;

  do {
    nr = usart_receive(0, &buffer, 1, 0xFFFF);
  } while(nr == 0);

  return buffer;
}

/*
 * Send an AT command to the modem.
```



```c
 */
static void modem_send_command(const char *fmt, ...)
{
  char buffer[67];
  uint16_t len;
  va_list arg;
  uint16_t i;
  uint8_t data;

  va_start(arg, fmt);
  len = vsnprintf(buffer, 64, fmt, arg);
  va_end(arg);
  strcpy(&buffer[len], "\r");
  len += 1;

  //printf_P(PSTR("> %s\n"), buffer);

  usart_transmit(0, (uint8_t *)buffer, len);

  //printf_P(PSTR("waiting for echo, length %d\n"), len);
  for(i = 0; i < len; i++) {
    data = modem_read_byte();
    //printf_P(PSTR("%c"), data);
    if(data != buffer[i]) {
      fatal_error_P(PSTR("Mismatch between sent data and received echo
          .\nSent: %s\nRecv: %c\n"), buffer, data);
    }
  }
}


/*
 * Read one line (CR LF-terminated) of response data from the modem.
 */
static char *modem_read_response(void)
{
  uint8_t len;
  //printf_P(PSTR("busy: %d\n"), modem_busy);
  len = 0;
  for(;;) {
    response_buffer[len] = modem_read_byte();

    if(response_buffer[len - 1] == '\r' &&
       response_buffer[len] == '\n') {
      break;
    }

    len++;
    if(len == BUFFER_SIZE) {
      fatal_error_P(PSTR("Response line to long...\n"));
    }
  }
```

```
  if(len == 1) {
    return modem_read_response();
  }

  response_buffer[len - 1] = '\0';

  return response_buffer;
}



/*
 * 0: RXD I
 * 1: TXD O
 * 2: RTS O
 * 3: RI  I
 * 4: DSR I
 * 5: CTS I
 * 6: DCD I
 * 7: DTR O
 */

/*
 * Initialize modem. Performes initialization up to, but not
 * including, entering of PIN code.
 */
void modem_init(void)
{
  response_buffer = (char *)malloc(sizeof(char) * BUFFER_SIZE);

  unread_messages = malloc(sizeof(uint8_t)*UNREAD_LIST_SZ);
  num_unread = 0;

  printf_P(PSTR("Initializing modem ..."));

  DDRE = _BV(7) | _BV(2);
  PORTE = _BV(7) | _BV(2);

  usart_setup(0);

  char *line;

  // TODO sette inn delay mellom kommandoer?
  printf_P(PSTR("init 1\n"));

  line = modem_read_response();
  if(strcmp(line, "^SYSSTART")) {
    fatal_error_P(PSTR("Expected startup message. Got: %s\n"), line);
  }

  printf_P(PSTR("init 2\n"));
```



```
  // code for turning off echoing of commands:
  /*
  modem_send_command("ATE0");
  line = modem_read_response();

  printf("init 2\n");

  if(!strcmp(line, "ATE0\r")) {
    // Ignore echo...
    line = modem_read_response();
  }

  if(strcmp(line, "OK")) {
    printf_P(PSTR("Unexpected response to ATE0 command: %s\n"), line);
    while(1);
  }
  */

  sleep(3000);

  printf_P(PSTR("init 3\n"));

  modem_send_command("AT");
  line = modem_read_response();
  if(strcmp(line, "OK")) {
    fatal_error_P(PSTR("Unexpected response to AT command: %s\n"), line
        );
  }

  sleep(3000);

  printf_P(PSTR("... modem up\n"));
}

/*
 * Send the pin code given as argument to the modem and perform the
 * rest of the initialization.
 */
uint8_t modem_enter_pin(const char *pin)
{
  char *line;

  modem_send_command("AT+CPIN=%s", pin);
  printf_P(PSTR("waiting for response\n"));
  line = modem_read_response();
  if(strcmp(line, "OK")) {
    printf_P(PSTR("Unexpected response to AT+CPIN command: %s\n"), line
        );
    return 0;
    //while(1);
  }
```

```
  modem_send_command("AT+CSNS=4");
  line = modem_read_response();
  if(strcmp(line, "OK")) {
    fatal_error_P(PSTR("Unexpected response to AT+CSNS=4 command: %s\n"
        ), line);
  }

  printf_P(PSTR("Config:\n"));

  modem_send_command("AT+CSNS?");
  line = modem_read_response();
  printf_P(PSTR("%s\n"), line);
  modem_read_response();

  modem_send_command("AT+CMGF=1");
  line = modem_read_response();
  if(strcmp(line, "OK")) {
    fatal_error_P(PSTR("Unexpected response to AT+CMGF=1 command: %s\n"
        ), line);
  }

  // (for å få modemet til å sende beskjeder ved nye meldinger:)
  // +CMTI (lager),(indeks) ved mottatt melding
  /*
  modem_send_command("AT+CNMI=1,1,0,0,1");
  line = modem_read_response();
  if (strcmp(line, "OK")) {
    printf_P(PSTR("Unexpected response to AT+CNMI command: %s\n"), line
        );
    while (1);
  }
  */

  return 1;
}

/*
 * Push msg_index onto the list of unread SMS messages.
 */
static void modem_push_unread(uint8_t msg_index)
{
  if (num_unread < UNREAD_LIST_SZ)
    unread_messages[num_unread++] = msg_index;
}
/*
 * Remove msg_index from the list of unread SMS messages.
 */
static void modem_pop_unread(uint8_t msg_index)
{
  uint8_t i;
  for (i = 0; i < num_unread; i++) {
```



```
    if (unread_messages[i] == msg_index) {
      num_unread--;
      for ( ; i < num_unread; i++) {
    unread_messages[i] = unread_messages[i+1];
      }
    }
  }
  if (num_unread == 0) {
    message_t *msg = malloc(sizeof(message_t));
    msg->type = MSG_NO_UNREAD_SMS;
    message_put(msg_to_ui, msg);
  }
}
/*
static uint8_t modem_unread(uint8_t msg_index)
{
  uint8_t i;
  for (i = 0; i < num_unread; i++)
    if (unread_messages[i] == msg_index)
      return 1;
  return 0;
}
*/

/*
 * Thread running initialization of modem and polling for new SMS
 * messages.
 */
void modem_thread(void *ignored)
{
  message_t *msg;
  modem_init();
  message_put_empty(msg_to_ui, MSG_READY_FOR_PIN);
  printf_P(PSTR("sent msg to ui...\n"));

  uint8_t pin_ok = 0;
  while (!pin_ok) {
    msg = message_wait_for(msg_to_modem, MSG_PIN);
    printf_P(PSTR("got pin code: %s\n"), msg->data.pin_code);

    if (modem_enter_pin(msg->data.pin_code))
      pin_ok = 1;
    free(msg);

    msg = malloc(sizeof(message_t));
    msg->type = MSG_PIN_RESULT;
    msg->data.pin_code_ok = pin_ok;
    message_put(msg_to_ui, msg);
  }

  /*
  printf_P(PSTR("start\n"));
```

```
  sleep(8000);
  printf_P(PSTR("start 2\n"));
  sleep(2000);
  */

  /*
  while (1) {
    char *ln = modem_read_response();
    printf_P(PSTR("res: %s\n"), ln);
  }
  */

  uint8_t sms_count = modem_get_sms_count();
  //sleep(8000);

  uint8_t foo = 0;
  while (1) {
    /*
    printf_P(PSTR("loop %d\n"), foo);
    sleep(6000);
    printf_P(PSTR("FOO!\n"));
    sleep(1000);
    */
    /*
    uint8_t sq = modem_signal_quality();
    msg = malloc(sizeof(message_t));
    msg->type = MSG_SIGNAL_QUALITY;
    msg->data.signal_quality = sq;
    message_put(msg_to_ui, msg);
    */

    uint8_t new_sms_count = modem_get_sms_count();
    if (new_sms_count > sms_count) {
      msg = malloc(sizeof(message_t));
      msg->type = MSG_RECV_SMS;
      message_put(msg_to_ui, msg);
    }
    while (sms_count < new_sms_count)
      modem_push_unread(++sms_count);
    sleep(4000);
  }
}

uint8_t modem_send_sms(sms_t *sms)
{
  char *line;
  uint8_t byte;

  printf_P(PSTR("send_sms, text='%s'\n"), sms->text);

  modem_set_busy(1);
```



```
  modem_send_command("AT+CMGS=\"%s\"", sms->number);
  if ((byte=modem_read_byte()) != '\r') {
    fatal_error_P(PSTR("read '%c', should be '>'\n"), byte);
  }
  if ((byte=modem_read_byte()) != '\n') {
    fatal_error_P(PSTR("read '%c', should be ' '\n"), byte);
  }
  if ((byte=modem_read_byte()) != '>') {
    fatal_error_P(PSTR("read '%c', should be '>'\n"), byte);
  }
  if ((byte=modem_read_byte()) != ' ') {
    fatal_error_P(PSTR("read '%c', should be ' '\n"), byte);
  }
  //if (!good_response) return 0;
  printf_P(PSTR("transmitting text...\n"));
  sleep(2000);
  usart_transmit(0, (uint8_t*)sms->text, strlen(sms->text));
  sleep(2000);
  uint8_t end = 0x1A;
  usart_transmit(0, &end, 1);
  //sleep(2000);
  //usart_transmit(0, (uint8_t*)modem_init_send, strlen(modem_init_send
    ));
  line = modem_read_response();
  printf_P(PSTR("response: %s\n"), line);
  //sleep(5000); // test
  if (strcmp(line, "ERROR")==0)
    return 0;
  line = modem_read_response();
  printf_P(PSTR("response: %s\n"), line);
  if (strcmp(line, "ERROR")==0)
    return 0;
  //sleep(5000); // test
  line = modem_read_response();
  if (strcmp(line, "OK")!=0)
    return 0;

  modem_set_busy(0);

  return 1;
}

int8_t modem_get_sms_count(void)
{
  char *line;
  modem_set_busy(1);
  modem_send_command("AT+CPMS=\"MT\"");
  line = modem_read_response();
  if (strncmp(line, "+CPMS", 5)!=0)
    return -1;
  int8_t n = atoi(line+7);
  modem_read_response();
```

```
  modem_set_busy(0);
  return n;
}


uint8_t modem_read_sms(sms_t *sms, uint8_t index, uint8_t mark_unread)
{
  char *line;
  //sleep(1000); // TODO trenger vi sleep her?
  modem_set_busy(1);
  modem_send_command("AT+CMGR=%d", index);
  line = modem_read_response();
  /*
  printf_P(PSTR("response: %s\n"), line);
  while (1) {
    line = modem_read_response();
    if (strcmp(line, "OK")==0)
      break;
  }
  */
  // example first response line:
  // +CMGR: "REC READ","+4712345678",,"07/11/08,22:30:57+04"
  uint8_t line_i, dest_i=0, commas, reading=0;
  for (line_i = 0, commas = 0; line[line_i] != '\0'; line_i++) {
    if (line[line_i]==',' && !reading) {
      commas++;
      dest_i = 0;
    } else if (line[line_i] == '"') {
      reading = 1-reading;
      if (commas == 1 && !reading) {
    sms->number[dest_i] = '\0';
      } else if (commas == 3 && !reading) {
    sms->date[dest_i] = '\0';
      }
    } else {
      if (commas == 1 && dest_i < SMS_NUMBER_LEN) {
    if (dest_i == SMS_NUMBER_LEN-1)
      sms->number[dest_i] = '\0';
    else
      sms->number[dest_i] = line[line_i];
    dest_i++;
      } else if (commas == 3 && dest_i < SMS_DATE_LEN) {
    if (line[line_i] == ',') {
      sms->date[dest_i] = ' ';
    } else if (line[line_i] == '+') {
      sms->date[dest_i] = '\0';
      break;
    } else {
      sms->date[dest_i] = line[line_i];
    }
    dest_i++;
      }
```



```
    }
  }

  dest_i = 0;
  while (1) {
    //printf_P(PSTR("reading response\n"));
    line = modem_read_response();
    printf_P(PSTR("response: %s\n"), line);
    if (strcmp(line, "OK") == 0) {
      sms->text[dest_i] = '\0';
      break;
    }
    for (line_i = 0; line[line_i] != '\0'; line_i++, dest_i++) {
      sms->text[dest_i] = line[line_i];
      if (dest_i == SMS_TEXT_LEN-1) break;
    }
    if (dest_i == SMS_TEXT_LEN-1) {
      sms->text[dest_i] = '\0';
      break;
    }
  }
  if (mark_unread)
    modem_pop_unread(index);
  modem_set_busy(0);
  return 0;
}

void modem_delete_sms(uint8_t index)
{
  char *line;
  modem_set_busy(1);
  printf_P(PSTR("Deleting message %d ..."), index);
  modem_send_command("AT+CMGD=%d", index);
  line = modem_read_response();
  printf_P(PSTR(" %s\n"), line);
  modem_set_busy(0);
}

void modem_delete_all_sms(void)
{
  uint8_t i, c;
  c = modem_get_sms_count();
  for (i = 1; i <= c; i++)
    modem_delete_sms(i);
}

uint8_t modem_signal_quality(void)
{
  char *line;
  modem_set_busy(1);
  modem_send_command("AT+CSQ");
  line = modem_read_response();
```

```
  modem_read_response();
  modem_set_busy(0);
  return atoi(line+6);
}
```

Listing G.46: src/output.c

```
#include <stdio.h>

#include "lcd.h"
#include "output.h"
#include "thread.h"

static FILE lcd_stdout;

static int lcd_putchar_wrapper(char character, FILE *stream)
{
  /* Check that we are within the stack. */
  thread_verify_stack();

  lcd_putchar(character);
  return 0;
}

void output_init(void)
{
  lcd_init();

  fdev_setup_stream(&lcd_stdout, lcd_putchar_wrapper, NULL,
      _FDEV_SETUP_WRITE);
  stdout = &lcd_stdout;
  stderr = &lcd_stdout;
}
```

Listing G.47: src/partition.c

```

/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#include "partition.h"
#include "partition_config.h"
#include "sd-reader_config.h"

#include <string.h>

#if USE_DYNAMIC_MEMORY
    #include <stdlib.h>
```



```
#endif

/**
 * \addtogroup partition Partition table support
 *
 * Support for reading partition tables and access to partitions.
 *
 * @{
 */
/**
 * \file
 * Partition table implementation.
 *
 * \author Roland Riegel
 */

/**
 * \addtogroup partition_config Configuration of partition table
    support
 * Preprocessor defines to configure the partition support.
 */

#if !USE_DYNAMIC_MEMORY
static struct partition_struct partition_handles[PARTITION_COUNT];
#endif

/**
 * Opens a partition.
 *
 * Opens a partition by its index number and returns a partition
 * handle which describes the opened partition.
 *
 * \note This function does not support extended partitions.
 *
 * \param[in] device_read A function pointer which is used to read from
     the disk.
 * \param[in] device_read_interval A function pointer which is used to
    read in constant intervals from the disk.
 * \param[in] device_write A function pointer which is used to write to
     the disk.
 * \param[in] device_write_interval A function pointer which is used to
     write a data stream to disk.
 * \param[in] index The index of the partition which should be opened,
    range 0 to 3.
 *                  A negative value is allowed as well. In this case,
    the partition opened is
 *                  not checked for existance, begins at offset zero,
    has a length of zero
 *                  and is of an unknown type.
 * \returns 0 on failure, a partition descriptor on success.
 * \see partition_close
 */
```

```
struct partition_struct* partition_open(device_read_t device_read,
    device_read_interval_t device_read_interval, device_write_t
    device_write, device_write_interval_t device_write_interval, int8_t
    index)
{
    struct partition_struct* new_partition = 0;
    uint8_t buffer[0x10];

    if(!device_read || !device_read_interval || index >= 4)
        return 0;

    if(index >= 0)
    {
        /* read specified partition table index */
        if(!device_read(0x01be + index * 0x10, buffer, sizeof(buffer)))
            return 0;

        /* abort on empty partition entry */
        if(buffer[4] == 0x00)
            return 0;
    }

    /* allocate partition descriptor */
#if USE_DYNAMIC_MEMORY
    new_partition = malloc(sizeof(*new_partition));
    if(!new_partition)
        return 0;
#else
    new_partition = partition_handles;
    uint8_t i;
    for(i = 0; i < PARTITION_COUNT; ++i)
    {
        if(new_partition->type == PARTITION_TYPE_FREE)
            break;

        ++new_partition;
    }
    if(i >= PARTITION_COUNT)
        return 0;
#endif

    memset(new_partition, 0, sizeof(*new_partition));

    /* fill partition descriptor */
    new_partition->device_read = device_read;
    new_partition->device_read_interval = device_read_interval;
    new_partition->device_write = device_write;
    new_partition->device_write_interval = device_write_interval;

    if(index >= 0)
    {
        new_partition->type = buffer[4];
```



```
        new_partition->offset = ((uint32_t) buffer[8]) |
                                ((uint32_t) buffer[9] << 8) |
                                ((uint32_t) buffer[10] << 16) |
                                ((uint32_t) buffer[11] << 24);
        new_partition->length = ((uint32_t) buffer[12]) |
                                ((uint32_t) buffer[13] << 8) |
                                ((uint32_t) buffer[14] << 16) |
                                ((uint32_t) buffer[15] << 24);
    }
    else
    {
        new_partition->type = 0xff;
    }

    return new_partition;
}

/**
 * Closes a partition.
 *
 * This function destroys a partition descriptor which was
 * previously obtained from a call to partition_open().
 * When this function returns, the given descriptor will be
 * invalid.
 *
 * \param[in] partition The partition descriptor to destroy.
 * \returns 0 on failure, 1 on success.
 * \see partition_open
 */
uint8_t partition_close(struct partition_struct* partition)
{
    if(!partition)
        return 0;

    /* destroy partition descriptor */
#if USE_DYNAMIC_MEMORY
    free(partition);
#else
    partition->type = PARTITION_TYPE_FREE;
#endif

    return 1;
}

/**
 * @}
 */
```

Listing G.48: src/phone_book.c

```
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <stdio.h>
#include <avr/pgmspace.h>

#include "fat16.h"
#include "fs.h"
#include "phone_book.h"
#include "menu.h"
#include "error.h"

#define PHONE_BOOK_FILE_NAME "phonebook"
struct fat16_file_struct *phone_book_file = NULL;

static uint8_t phone_book_offset;
//static uint8_t phone_book_i;
static phone_book_entry_t *entries[3] = { NULL, NULL, NULL };
static char *phone_book_menu_title;
static char *phone_book_menu_title_full;
static char *phone_book_menu_items[3];
static phone_book_callback_t phone_book_select_handler;

uint8_t phone_book_open(void)
{
  //phone_book_menu_title = malloc(sizeof(char)*DISPLAY_WIDTH);
  phone_book_menu_title_full = malloc(sizeof(char)*DISPLAY_WIDTH);
  uint8_t i;
  for (i = 0; i < 3; i++) {
    phone_book_menu_items[i] = malloc(sizeof(char)*DISPLAY_WIDTH);
    entries[i] = malloc(sizeof(phone_book_entry_t));
  }

  phone_book_file = fs_open_file(PHONE_BOOK_FILE_NAME);
  if (phone_book_file == NULL)
    return 0;
  return 1;
}

static uint8_t phone_book_seek(uint8_t pos)
{
  int32_t byte_offset = pos*sizeof(phone_book_entry_t);
  return fat16_seek_file(phone_book_file,
             &byte_offset,
             FAT16_SEEK_SET);
}

uint8_t phone_book_read(phone_book_entry_t *entry, uint8_t entry_index)
{
  if (!phone_book_seek(entry_index))
    return 0;
  int8_t res = fat16_read_file(phone_book_file, (uint8_t*)entry, sizeof
     (phone_book_entry_t));
  if (res <= 0) return res;
```



```
  if (res < sizeof(phone_book_entry_t)) error_P("Incomplete read of
     phone book entry");
  return 1;
  /*
  int8_t res;
  uint8_t bytes_read = 0;
  if (!phone_book_seek(entry_index))
    return 0;
  while (bytes_read < sizeof(phone_book_entry_t)) {
    res = fat16_read_file(phone_book_file,
              ((uint8_t*)entry)+bytes_read,
              sizeof(phone_book_entry_t)-bytes_read);
    if (res == -1)
      return -1;
    if (res == 0)
      return 0;
  }
  return 1;
  */
}

void phone_book_close(void)
{
  fat16_close_file(phone_book_file);
  phone_book_file = NULL;
}

uint8_t phone_book_size(void)
{
  if (phone_book_file == NULL) {
    error_P(PSTR("phone_book_size() called before phone book is opened"
       ));
    return 0;
  }
  return fat16_get_file_size(phone_book_file)/sizeof(phone_book_entry_t
     );
}


phone_book_entry_t *phone_book_search(char *number)
{
  if (phone_book_file == NULL && !phone_book_open()) {
    error_P(PSTR("Phone book search failed (could not open phone book)"
       ));
    return NULL;
  }
  uint8_t i;
  int8_t res;
  for (i = 0; i < phone_book_size(); i++) {
    res = phone_book_read(entries[0], i);
    if (res == -1) {
```

```
      error_P(PSTR("Phone book search failed (could not read phone book
          entry %d)"), i);
      return NULL;
    }
    if (res == 0) {
      error_P(PSTR("Phone book search failed (too early EOF in phone
         book file at entry %d)"), i);
      return NULL;
    }

    char prefix[] = "+47";
    if (strcmp((char*)(entries[0]->number), number)==0 ||
    (strncmp((char*)(entries[0]->number), prefix, 3)==0 && strcmp(&((
       char*)(entries[0]->number))[3], number)==0) ||
    (strncmp(number, prefix, 3)==0 && strcmp((char*)(entries[0]->number
       ), &number[3])==0))
      return entries[0];
  }
  return NULL;
}



// GUI:

static void phone_book_menu_select(uint8_t i)
{
  //phone_book_i = i;
  if (phone_book_select_handler == NULL)
    error_P(PSTR("No phone book handler set"));
  phone_book_select_handler(entries[i]);
}

static void phone_book_menu_scroll_up(uint8_t ignored);
static void phone_book_menu_scroll_down(uint8_t ignored);

static void phone_book_menu_show(void)
{
  menu_t *m;
  uint8_t pageno, pages;
  pages = (phone_book_size()+2)/3;
  pageno = phone_book_offset/3 + 1;
  snprintf_P(phone_book_menu_title_full, MENU_LINE_WIDTH, PSTR("%s[%d/%
     d]"), phone_book_menu_title, pageno, pages);
  m = menu_add(phone_book_menu_title_full, 0);
  menu_set_abcd_item(m, 'A', "up", phone_book_menu_scroll_up);
  menu_set_abcd_item(m, 'B', "down", phone_book_menu_scroll_down);

  uint8_t i;
  int8_t res;

  for (i = 0; i < 3; i++) {
```



```
    res = phone_book_read(entries[i], phone_book_offset+i);
    if (res == -1)
      sprintf_P(phone_book_menu_items[i], PSTR("(error)"));
    else if (res == 0)
      //sprintf_P(str, PSTR("(eof)"));
      break;
    else
      sprintf_P(phone_book_menu_items[i], PSTR("%s %s"), entries[i]->
         name, entries[i]->number);
    menu_add_item(m, phone_book_menu_items[i], phone_book_menu_select);
  }

  menu_setup_done(m);
  //menu_setstring(0, "(no more entries)");
  //menu_update_display();
}

static void phone_book_menu_scroll_up(uint8_t ignored)
{
  if (phone_book_offset-3 >= 0) {
    phone_book_offset -= 3;
    menu_pop_update(0);
    phone_book_menu_show();
  }
}
static void phone_book_menu_scroll_down(uint8_t ignored)
{
  if (phone_book_offset+3 < phone_book_size()) {
    phone_book_offset += 3;
    menu_pop_update(0);
    phone_book_menu_show();
  }
}

void phone_book_menu(char *title, phone_book_callback_t select_handler)
{
  phone_book_menu_title = title;
  phone_book_select_handler = select_handler;
  if (phone_book_file == NULL && !phone_book_open()) {
    error_P(PSTR("Could not open phone book"));
    return;
  }
  phone_book_offset = 0;
  phone_book_menu_show();
}
```

Listing G.49: src/scheduler.c

```
#include <avr/pgmspace.h>
#include <stdio.h>

#include "scheduler.h"
```

```
#define MAX_THREADS 16

typedef struct thread_queue_element_s {
  struct thread_queue_element_s *next;
  thread_t *thread;
} thread_queue_element_t;

static thread_queue_element_t thread_queue_elements[MAX_THREADS];

static thread_queue_element_t *next_thread;
static thread_queue_element_t **next_thread_slot;

static thread_queue_element_t *find_free_element(void)
{
  uint8_t i;

  for(i = 0; i < MAX_THREADS; i++) {
    if(thread_queue_elements[i].thread == NULL) {
      return &thread_queue_elements[i];
    }
  }

  printf_P(PSTR("No free scheduler slot.\n"));
  while(1);
}

void scheduler_init(void)
{
  uint8_t i;

  for(i = 0; i < MAX_THREADS; i++) {
    thread_queue_elements[i].thread = NULL;
  }

  next_thread = NULL;
  next_thread_slot = &next_thread;
}

void scheduler_add_thread(thread_t *thread)
{
  thread_queue_element_t *e;

  e = find_free_element();

  e->next = NULL;
  e->thread = thread;

  *next_thread_slot = e;
  next_thread_slot = &e->next;
}

thread_t *scheduler_get_next_thread(void)
```



```
{
  thread_queue_element_t *e;
  thread_t *t;

  e = next_thread;
  if(e == NULL) {
    return NULL;
  }

  next_thread = e->next;
  if(next_thread == NULL) {
    next_thread_slot = &next_thread;
  }

  t = e->thread;
  e->thread = NULL;
  e->next = NULL;

  return t;
}
```

Listing G.50: src/sd_raw.c

```
/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#include <string.h>
#include <avr/io.h>
#include "sd_raw.h"
#include <stdio.h>

/**
 * \addtogroup sd_raw MMC/SD card raw access
 *
 * This module implements read and write access to MMC and
 * SD cards. It serves as a low-level driver for the higher
 * level modules such as partition and file system access.
 *
 * @{
 */
/**
 * \file
 * MMC/SD raw access implementation.
 *
 * \author Roland Riegel
 */
```

```
/**
 * \addtogroup sd_raw_config MMC/SD configuration
 * Preprocessor defines to configure the MMC/SD support.
 */

/**
 * @}
 */

/* commands available in SPI mode */

/* CMD0: response R1 */
#define CMD_GO_IDLE_STATE 0x00
/* CMD1: response R1 */
#define CMD_SEND_OP_COND 0x01
/* CMD9: response R1 */
#define CMD_SEND_CSD 0x09
/* CMD10: response R1 */
#define CMD_SEND_CID 0x0a
/* CMD12: response R1 */
#define CMD_STOP_TRANSMISSION 0x0c
/* CMD13: response R2 */
#define CMD_SEND_STATUS 0x0d
/* CMD16: arg0[31:0]: block length, response R1 */
#define CMD_SET_BLOCKLEN 0x10
/* CMD17: arg0[31:0]: data address, response R1 */
#define CMD_READ_SINGLE_BLOCK 0x11
/* CMD18: arg0[31:0]: data address, response R1 */
#define CMD_READ_MULTIPLE_BLOCK 0x12
/* CMD24: arg0[31:0]: data address, response R1 */
#define CMD_WRITE_SINGLE_BLOCK 0x18
/* CMD25: arg0[31:0]: data address, response R1 */
#define CMD_WRITE_MULTIPLE_BLOCK 0x19
/* CMD27: response R1 */
#define CMD_PROGRAM_CSD 0x1b
/* CMD28: arg0[31:0]: data address, response R1b */
#define CMD_SET_WRITE_PROT 0x1c
/* CMD29: arg0[31:0]: data address, response R1b */
#define CMD_CLR_WRITE_PROT 0x1d
/* CMD30: arg0[31:0]: write protect data address, response R1 */
#define CMD_SEND_WRITE_PROT 0x1e
/* CMD32: arg0[31:0]: data address, response R1 */
#define CMD_TAG_SECTOR_START 0x20
/* CMD33: arg0[31:0]: data address, response R1 */
#define CMD_TAG_SECTOR_END 0x21
/* CMD34: arg0[31:0]: data address, response R1 */
#define CMD_UNTAG_SECTOR 0x22
/* CMD35: arg0[31:0]: data address, response R1 */
#define CMD_TAG_ERASE_GROUP_START 0x23
/* CMD36: arg0[31:0]: data address, response R1 */
#define CMD_TAG_ERASE_GROUP_END 0x24
/* CMD37: arg0[31:0]: data address, response R1 */
```



```
#define CMD_UNTAG_ERASE_GROUP 0x25
/* CMD38: arg0[31:0]: stuff bits, response R1b */
#define CMD_ERASE 0x26
/* CMD42: arg0[31:0]: stuff bits, response R1b */
#define CMD_LOCK_UNLOCK 0x2a
/* CMD58: response R3 */
#define CMD_READ_OCR 0x3a
/* CMD59: arg0[31:1]: stuff bits, arg0[0:0]: crc option, response R1 */
#define CMD_CRC_ON_OFF 0x3b

/* command responses */
/* R1: size 1 byte */
#define R1_IDLE_STATE 0
#define R1_ERASE_RESET 1
#define R1_ILL_COMMAND 2
#define R1_COM_CRC_ERR 3
#define R1_ERASE_SEQ_ERR 4
#define R1_ADDR_ERR 5
#define R1_PARAM_ERR 6
/* R1b: equals R1, additional busy bytes */
/* R2: size 2 bytes */
#define R2_CARD_LOCKED 0
#define R2_WP_ERASE_SKIP 1
#define R2_ERR 2
#define R2_CARD_ERR 3
#define R2_CARD_ECC_FAIL 4
#define R2_WP_VIOLATION 5
#define R2_INVAL_ERASE 6
#define R2_OUT_OF_RANGE 7
#define R2_CSD_OVERWRITE 7
#define R2_IDLE_STATE (R1_IDLE_STATE + 8)
#define R2_ERASE_RESET (R1_ERASE_RESET + 8)
#define R2_ILL_COMMAND (R1_ILL_COMMAND + 8)
#define R2_COM_CRC_ERR (R1_COM_CRC_ERR + 8)
#define R2_ERASE_SEQ_ERR (R1_ERASE_SEQ_ERR + 8)
#define R2_ADDR_ERR (R1_ADDR_ERR + 8)
#define R2_PARAM_ERR (R1_PARAM_ERR + 8)
/* R3: size 5 bytes */
#define R3_OCR_MASK (0xffffffffUL)
#define R3_IDLE_STATE (R1_IDLE_STATE + 32)
#define R3_ERASE_RESET (R1_ERASE_RESET + 32)
#define R3_ILL_COMMAND (R1_ILL_COMMAND + 32)
#define R3_COM_CRC_ERR (R1_COM_CRC_ERR + 32)
#define R3_ERASE_SEQ_ERR (R1_ERASE_SEQ_ERR + 32)
#define R3_ADDR_ERR (R1_ADDR_ERR + 32)
#define R3_PARAM_ERR (R1_PARAM_ERR + 32)
/* Data Response: size 1 byte */
#define DR_STATUS_MASK 0x0e
#define DR_STATUS_ACCEPTED 0x05
#define DR_STATUS_CRC_ERR 0x0a
#define DR_STATUS_WRITE_ERR 0x0c
```

```
#if !SD_RAW_SAVE_RAM

/* static data buffer for acceleration */
static uint8_t raw_block[512];
/* offset where the data within raw_block lies on the card */
static uint32_t raw_block_address;
#if SD_RAW_WRITE_BUFFERING
/* flag to remember if raw_block was written to the card */
static uint8_t raw_block_written;
#endif

#endif

/* private helper functions */
static void sd_raw_send_byte(uint8_t b);
static uint8_t sd_raw_rec_byte();
static uint8_t sd_raw_send_command_r1(uint8_t command, uint32_t arg);
static uint16_t sd_raw_send_command_r2(uint8_t command, uint32_t arg);

/**
 * \ingroup sd_raw
 * Initializes memory card communication.
 *
 * \returns 0 on failure, 1 on success.
 */
uint8_t sd_raw_init()
{
    /* enable inputs for reading card status */
    configure_pin_available();
    //configure_pin_locked();

    printf("init(1)\n");

    /* enable outputs for MOSI, SCK, SS, input for MISO */
    configure_pin_mosi();
    configure_pin_sck();
    configure_pin_ss();
    configure_pin_miso();

    printf("init(2)\n");

    unselect_card();

    printf("init(3)\n");

    /* initialize SPI with lowest frequency; max. 400kHz during
       identification mode of card */
    SPCR = (0 << SPIE) | /* SPI Interrupt Enable */
           (1 << SPE)  | /* SPI Enable */
           (0 << DORD) | /* Data Order: MSB first */
           (1 << MSTR) | /* Master mode */
           (0 << CPOL) | /* Clock Polarity: SCK low when idle */
```



```
           (0 << CPHA) | /* Clock Phase: sample on rising SCK edge */
           (1 << SPR1) | /* Clock Frequency: f_OSC / 128 */
           (1 << SPR0);
    SPSR &= ~(1 << SPI2X); /* No doubled clock frequency */

    /* initialization procedure */

    printf("init(4)\n");

    /*
    if(!sd_raw_available())
        return 0;
    */

    printf("init(5)\n");

    /* card needs 74 cycles minimum to start up */
    for(uint8_t i = 0; i < 10; ++i)
    {
        /* wait 8 clock cycles */
        sd_raw_rec_byte();
    }

    /* address card */
    select_card();

    /* reset card */
    uint8_t response;
    for(uint16_t i = 0; ; ++i)
    {
        response = sd_raw_send_command_r1(CMD_GO_IDLE_STATE, 0);
        if(response == (1 << R1_IDLE_STATE))
            break;

        if(i == 0x1ff)
        {
            unselect_card();
            return 0;
        }
    }

    /* wait for card to get ready */
    for(uint16_t i = 0; ; ++i)
    {
        response = sd_raw_send_command_r1(CMD_SEND_OP_COND, 0);
        if(!(response & (1 << R1_IDLE_STATE)))
            break;

        if(i == 0x7fff)
        {
            unselect_card();
            return 0;
```

```
        }
    }

    /* set block size to 512 bytes */
    if(sd_raw_send_command_r1(CMD_SET_BLOCKLEN, 512))
    {
        unselect_card();
        return 0;
    }

    /* deaddress card */
    unselect_card();

    /* switch to highest SPI frequency possible */
    SPCR &= ~((1 << SPR1) | (1 << SPR0)); /* Clock Frequency: f_OSC / 4
        */
    SPSR |= (1 << SPI2X); /* Doubled Clock Frequency: f_OSC / 2 */

#if !SD_RAW_SAVE_RAM
    /* the first block is likely to be accessed first, so precache it
       here */
    raw_block_address = 0xffffffff;
#if SD_RAW_WRITE_BUFFERING
    raw_block_written = 1;
#endif
    if(!sd_raw_read(0, raw_block, sizeof(raw_block)))
        return 0;
#endif

    return 1;
}

/**
 * \ingroup sd_raw
 * Checks wether a memory card is located in the slot.
 *
 * \returns 1 if the card is available, 0 if it is not.
 */
uint8_t sd_raw_available()
{
    return get_pin_available() == 0x00;
}

/**
 * \ingroup sd_raw
 * Checks wether the memory card is locked for write access.
 *
 * \returns 1 if the card is locked, 0 if it is not.
 */
uint8_t sd_raw_locked()
{
    return get_pin_locked() == 0x00;
```



```
}

/**
 * \ingroup sd_raw
 * Sends a raw byte to the memory card.
 *
 * \param[in] b The byte to sent.
 * \see sd_raw_rec_byte
 */
void sd_raw_send_byte(uint8_t b)
{
    SPDR = b;
    /* wait for byte to be shifted out */
    while(!(SPSR & (1 << SPIF)));
    SPSR &= ~(1 << SPIF);
}

/**
 * \ingroup sd_raw
 * Receives a raw byte from the memory card.
 *
 * \returns The byte which should be read.
 * \see sd_raw_send_byte
 */
uint8_t sd_raw_rec_byte()
{
    /* send dummy data for receiving some */
    SPDR = 0xff;
    while(!(SPSR & (1 << SPIF)));
    SPSR &= ~(1 << SPIF);

    return SPDR;
}

/**
 * \ingroup sd_raw
 * Send a command to the memory card which responses with a R1 response
    .
 *
 * \param[in] command The command to send.
 * \param[in] arg The argument for command.
 * \returns The command answer.
 */
uint8_t sd_raw_send_command_r1(uint8_t command, uint32_t arg)
{
    uint8_t response;

    /* wait some clock cycles */
    sd_raw_rec_byte();

    /* send command via SPI */
    sd_raw_send_byte(0x40 | command);
```

```
    sd_raw_send_byte((arg >> 24) & 0xff);
    sd_raw_send_byte((arg >> 16) & 0xff);
    sd_raw_send_byte((arg >> 8) & 0xff);
    sd_raw_send_byte((arg >> 0) & 0xff);
    sd_raw_send_byte(command == CMD_GO_IDLE_STATE ? 0x95 : 0xff);

    /* receive response */
    for(uint8_t i = 0; i < 10; ++i)
    {
        response = sd_raw_rec_byte();
        if(response != 0xff)
            break;
    }

    return response;
}

/**
 * \ingroup sd_raw
 * Send a command to the memory card which responses with a R2 response
    .
 *
 * \param[in] command The command to send.
 * \param[in] arg The argument for command.
 * \returns The command answer.
 */
uint16_t sd_raw_send_command_r2(uint8_t command, uint32_t arg)
{
    uint16_t response;

    /* wait some clock cycles */
    sd_raw_rec_byte();

    /* send command via SPI */
    sd_raw_send_byte(0x40 | command);
    sd_raw_send_byte((arg >> 24) & 0xff);
    sd_raw_send_byte((arg >> 16) & 0xff);
    sd_raw_send_byte((arg >> 8) & 0xff);
    sd_raw_send_byte((arg >> 0) & 0xff);
    sd_raw_send_byte(command == CMD_GO_IDLE_STATE ? 0x95 : 0xff);

    /* receive response */
    for(uint8_t i = 0; i < 10; ++i)
    {
        response = sd_raw_rec_byte();
        if(response != 0xff)
            break;
    }
    response <<= 8;
    response |= sd_raw_rec_byte();

    return response;
```



```
}

/**
 * \ingroup sd_raw
 * Reads raw data from the card.
 *
 * \param[in] offset The offset from which to read.
 * \param[out] buffer The buffer into which to write the data.
 * \param[in] length The number of bytes to read.
 * \returns 0 on failure, 1 on success.
 * \see sd_raw_read_interval, sd_raw_write, sd_raw_write_interval
 */
uint8_t sd_raw_read(uint32_t offset, uint8_t* buffer, uint16_t length)
{
    uint32_t block_address;
    uint16_t block_offset;
    uint16_t read_length;
    while(length > 0)
    {
        /* determine byte count to read at once */
        block_address = offset & 0xfffffe00;
        block_offset = offset & 0x01ff;
        read_length = 512 - block_offset; /* read up to block border */
        if(read_length > length)
            read_length = length;

#if !SD_RAW_SAVE_RAM
        /* check if the requested data is cached */
        if(block_address != raw_block_address)
#endif
        {
#if SD_RAW_WRITE_BUFFERING
            if(!raw_block_written)
            {
                if(!sd_raw_write(raw_block_address, raw_block, sizeof(
                    raw_block)))
                    return 0;
            }
#endif

            /* address card */
            select_card();

            /* send single block request */
            if(sd_raw_send_command_r1(CMD_READ_SINGLE_BLOCK,
                block_address))
            {
                unselect_card();
                return 0;
            }

            /* wait for data block (start byte 0xfe) */
```

```
            while(sd_raw_rec_byte() != 0xfe);

#if SD_RAW_SAVE_RAM
            /* read byte block */
            uint16_t read_to = block_offset + read_length;
            for(uint16_t i = 0; i < 512; ++i)
            {
                uint8_t b = sd_raw_rec_byte();
                if(i >= block_offset && i < read_to)
                    *buffer++ = b;
            }
#else
            /* read byte block */
            uint8_t* cache = raw_block;
            for(uint16_t i = 0; i < 512; ++i)
                *cache++ = sd_raw_rec_byte();
            raw_block_address = block_address;

            memcpy(buffer, raw_block + block_offset, read_length);
            buffer += read_length;
#endif

            /* read crc16 */
            sd_raw_rec_byte();
            sd_raw_rec_byte();

            /* deaddress card */
            unselect_card();

            /* let card some time to finish */
            sd_raw_rec_byte();
        }
#if !SD_RAW_SAVE_RAM
        else
        {
            /* use cached data */
            memcpy(buffer, raw_block + block_offset, read_length);
            buffer += read_length;
        }
#endif

        length -= read_length;
        offset += read_length;
    }

    return 1;
}

/**
 * \ingroup sd_raw
 * Continuously reads units of \c interval bytes and calls a callback
    function.
```



```
 *
 * This function starts reading at the specified offset. Every \c
    interval bytes,
 * it calls the callback function with the associated data buffer.
 *
 * By returning zero, the callback may stop reading.
 *
 * \note Within the callback function, you can not start another read
    or
 *       write operation.
 * \note This function only works if the following conditions are met:
 *       - (offset - (offset % 512)) % interval == 0
 *       - length % interval == 0
 *
 * \param[in] offset Offset from which to start reading.
 * \param[in] buffer Pointer to a buffer which is at least interval
    bytes in size.
 * \param[in] interval Number of bytes to read before calling the
    callback function.
 * \param[in] length Number of bytes to read altogether.
 * \param[in] callback The function to call every interval bytes.
 * \param[in] p An opaque pointer directly passed to the callback
    function.
 * \returns 0 on failure, 1 on success
 * \see sd_raw_write_interval, sd_raw_read, sd_raw_write
 */
uint8_t sd_raw_read_interval(uint32_t offset, uint8_t* buffer, uint16_t
    interval, uint16_t length, sd_raw_read_interval_handler_t callback,
    void* p)
{
    if(!buffer || interval == 0 || length < interval || !callback)
        return 0;

#if !SD_RAW_SAVE_RAM
    while(length >= interval)
    {
        /* as reading is now buffered, we directly
         * hand over the request to sd_raw_read()
         */
        if(!sd_raw_read(offset, buffer, interval))
            return 0;
        if(!callback(buffer, offset, p))
            break;
        offset += interval;
        length -= interval;
    }

    return 1;
#else
    /* address card */
    select_card();
```

```
    uint16_t block_offset;
    uint16_t read_length;
    uint8_t* buffer_cur;
    uint8_t finished = 0;
    do
    {
        /* determine byte count to read at once */
        block_offset = offset & 0x01ff;
        read_length = 512 - block_offset;

        /* send single block request */
        if(sd_raw_send_command_r1(CMD_READ_SINGLE_BLOCK, offset & 0
           xfffffe00))
        {
            unselect_card();
            return 0;
        }

        /* wait for data block (start byte 0xfe) */
        while(sd_raw_rec_byte() != 0xfe);

        /* read up to the data of interest */
        for(uint16_t i = 0; i < block_offset; ++i)
            sd_raw_rec_byte();

        /* read interval bytes of data and execute the callback */
        do
        {
            if(read_length < interval || length < interval)
                break;

            buffer_cur = buffer;
            for(uint16_t i = 0; i < interval; ++i)
                *buffer_cur++ = sd_raw_rec_byte();

            if(!callback(buffer, offset + (512 - read_length), p))
            {
                finished = 1;
                break;
            }

            read_length -= interval;
            length -= interval;

        } while(read_length > 0 && length > 0);

        /* read rest of data block */
        while(read_length-- > 0)
            sd_raw_rec_byte();

        /* read crc16 */
        sd_raw_rec_byte();
```



```
        sd_raw_rec_byte();

        if(length < interval)
            break;

        offset = (offset & 0xfffffe00) + 512;

    } while(!finished);

    /* deaddress card */
    unselect_card();

    /* let card some time to finish */
    sd_raw_rec_byte();

    return 1;
#endif
}

/**
 * \ingroup sd_raw
 * Writes raw data to the card.
 *
 * \note If write buffering is enabled, you might have to
 *       call sd_raw_sync() before disconnecting the card
 *       to ensure all remaining data has been written.
 *
 * \param[in] offset The offset where to start writing.
 * \param[in] buffer The buffer containing the data to be written.
 * \param[in] length The number of bytes to write.
 * \returns 0 on failure, 1 on success.
 * \see sd_raw_write_interval, sd_raw_read, sd_raw_read_interval
 */
uint8_t sd_raw_write(uint32_t offset, const uint8_t* buffer, uint16_t
   length)
{
#if SD_RAW_WRITE_SUPPORT

    if(get_pin_locked())
        return 0;

    uint32_t block_address;
    uint16_t block_offset;
    uint16_t write_length;
    while(length > 0)
    {
        /* determine byte count to write at once */
        block_address = offset & 0xfffffe00;
        block_offset = offset & 0x01ff;
        write_length = 512 - block_offset; /* write up to block border
           */
        if(write_length > length)
```

```
            write_length = length;

        /* Merge the data to write with the content of the block.
         * Use the cached block if available.
         */
        if(block_address != raw_block_address)
        {
#if SD_RAW_WRITE_BUFFERING
            if(!raw_block_written)
            {
                if(!sd_raw_write(raw_block_address, raw_block, sizeof(
                    raw_block)))
                    return 0;
            }
#endif

            if(block_offset || write_length < 512)
            {
                if(!sd_raw_read(block_address, raw_block, sizeof(
                    raw_block)))
                    return 0;
            }
            raw_block_address = block_address;
        }

        if(buffer != raw_block)
        {
            memcpy(raw_block + block_offset, buffer, write_length);

#if SD_RAW_WRITE_BUFFERING
            raw_block_written = 0;

            if(length == write_length)
                return 1;
#endif
        }

        buffer += write_length;

        /* address card */
        select_card();

        /* send single block request */
        if(sd_raw_send_command_r1(CMD_WRITE_SINGLE_BLOCK, block_address
            ))
        {
            unselect_card();
            return 0;
        }

        /* send start byte */
        sd_raw_send_byte(0xfe);
```



```
        /* write byte block */
        uint8_t* cache = raw_block;
        for(uint16_t i = 0; i < 512; ++i)
            sd_raw_send_byte(*cache++);

        /* write dummy crc16 */
        sd_raw_send_byte(0xff);
        sd_raw_send_byte(0xff);

        /* wait while card is busy */
        while(sd_raw_rec_byte() != 0xff);
        sd_raw_rec_byte();

        /* deaddress card */
        unselect_card();

        length -= write_length;
        offset += write_length;

#if SD_RAW_WRITE_BUFFERING
        raw_block_written = 1;
#endif
    }

    return 1;
#else
    return 0;
#endif
}

/**
 * \ingroup sd_raw
 * Writes a continuous data stream obtained from a callback function.
 *
 * This function starts writing at the specified offset. To obtain the
 * next bytes to write, it calls the callback function. The callback
    fills the
 * provided data buffer and returns the number of bytes it has put into
    the buffer.
 *
 * By returning zero, the callback may stop writing.
 *
 * \param[in] offset Offset where to start writing.
 * \param[in] buffer Pointer to a buffer which is used for the callback
    function.
 * \param[in] length Number of bytes to write in total. May be zero for
    endless writes.
 * \param[in] callback The function used to obtain the bytes to write.
 * \param[in] p An opaque pointer directly passed to the callback
    function.
 * \returns 0 on failure, 1 on success
```

```
 * \see sd_raw_read_interval, sd_raw_write, sd_raw_read
 */
uint8_t sd_raw_write_interval(uint32_t offset, uint8_t* buffer,
    uint16_t length, sd_raw_write_interval_handler_t callback, void* p)
{
#if SD_RAW_WRITE_SUPPORT

#if SD_RAW_SAVE_RAM
    #error "SD_RAW_WRITE_SUPPORT is not supported together with
        SD_RAW_SAVE_RAM"
#endif

    if(!buffer || !callback)
        return 0;

    uint8_t endless = (length == 0);
    while(endless || length > 0)
    {
        uint16_t bytes_to_write = callback(buffer, offset, p);
        if(!bytes_to_write)
            break;
        if(!endless && bytes_to_write > length)
            return 0;

        /* as writing is always buffered, we directly
         * hand over the request to sd_raw_write()
         */
        if(!sd_raw_write(offset, buffer, bytes_to_write))
            return 0;

        offset += bytes_to_write;
        length -= bytes_to_write;
    }

    return 1;

#else
    return 0;
#endif
}

/**
 * \ingroup sd_raw
 * Writes the write buffer's content to the card.
 *
 * \note When write buffering is enabled, you should
 *       call this function before disconnecting the
 *       card to ensure all remaining data has been
 *       written.
 *
 * \returns 0 on failure, 1 on success.
 * \see sd_raw_write
```



```
 */
uint8_t sd_raw_sync()
{
#if SD_RAW_WRITE_SUPPORT
#if SD_RAW_WRITE_BUFFERING
    if(raw_block_written)
        return 1;
    if(!sd_raw_write(raw_block_address, raw_block, sizeof(raw_block)))
        return 0;
#endif
    return 1;
#else
    return 0;
#endif
}

/**
 * \ingroup sd_raw
 * Reads informational data from the card.
 *
 * This function reads and returns the card's registers
 * containing manufacturing and status information.
 *
 * \note: The information retrieved by this function is
 *        not required in any way to operate on the card,
 *        but it might be nice to display some of the data
 *        to the user.
 *
 * \param[in] info A pointer to the structure into which to save the
    information.
 * \returns 0 on failure, 1 on success.
 */
uint8_t sd_raw_get_info(struct sd_raw_info* info)
{
    if(!info || !sd_raw_available())
        return 0;

    memset(info, 0, sizeof(*info));

    select_card();

    /* read cid register */
    if(sd_raw_send_command_r1(CMD_SEND_CID, 0))
    {
        unselect_card();
        return 0;
    }
    while(sd_raw_rec_byte() != 0xfe);
    for(uint8_t i = 0; i < 18; ++i)
    {
        uint8_t b = sd_raw_rec_byte();
```

```
        switch(i)
        {
            case 0:
                info->manufacturer = b;
                break;
            case 1:
            case 2:
                info->oem[i - 1] = b;
                break;
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
                info->product[i - 3] = b;
                break;
            case 8:
                info->revision = b;
                break;
            case 9:
            case 10:
            case 11:
            case 12:
                info->serial |= (uint32_t) b << ((12 - i) * 8);
                break;
            case 13:
                info->manufacturing_year = b << 4;
                break;
            case 14:
                info->manufacturing_year |= b >> 4;
                info->manufacturing_month = b & 0x0f;
                break;
        }
    }

    /* read csd register */
    uint8_t csd_read_bl_len = 0;
    uint8_t csd_c_size_mult = 0;
    uint16_t csd_c_size = 0;
    if(sd_raw_send_command_r1(CMD_SEND_CSD, 0))
    {
        unselect_card();
        return 0;
    }
    while(sd_raw_rec_byte() != 0xfe);
    for(uint8_t i = 0; i < 18; ++i)
    {
        uint8_t b = sd_raw_rec_byte();

        switch(i)
        {
            case 5:
```



```
                csd_read_bl_len = b & 0x0f;
                break;
            case 6:
                csd_c_size = (uint16_t) (b & 0x03) << 8;
                break;
            case 7:
                csd_c_size |= b;
                csd_c_size <<= 2;
                break;
            case 8:
                csd_c_size |= b >> 6;
                ++csd_c_size;
                break;
            case 9:
                csd_c_size_mult = (b & 0x03) << 1;
                break;
            case 10:
                csd_c_size_mult |= b >> 7;

                info->capacity = (uint32_t) csd_c_size << (
                    csd_c_size_mult + csd_read_bl_len + 2);

                break;
            case 14:
                if(b & 0x40)
                    info->flag_copy = 1;
                if(b & 0x20)
                    info->flag_write_protect = 1;
                if(b & 0x10)
                    info->flag_write_protect_temp = 1;
                info->format = (b & 0x0c) >> 2;
                break;
        }
    }

    unselect_card();

    return 1;
}
```

Listing G.51: src/sleep.c

```
#include <avr/interrupt.h>
#include "sleep.h"
#include "thread.h"
#include "time.h"

/* This function freezes the current thread for at least timeout
 * milliseconds.
 *
 * Interrupts must be enabled when calling this function.
 *
 * Parameters:
```

```
 *  timeout         The number of milliseconds this thread should sleep
    .
 */
void sleep(uint16_t timeout)
{
  uint32_t wake_when;
  time_wake_element_t wake_event;

  /* Calulcate when the thread should wake. */
  wake_when = time_get() + timeout;

  /* Disable interrupts. */
  cli();

  /* Add a wake event. */
  time_add_wake(&wake_event, wake_when);

  /* Freeze the current thread. */
  thread_wait();

  /* Remove the wake event. */
  time_remove_wake(&wake_event);

  /* Enable interrupts again. */
  sei();
}
```

Listing G.52: src/sms.c

```
#ifndef NOAVR
#include <avr/pgmspace.h>
#endif
#include <stdlib.h>
#include <stdint.h>
#include <ctype.h>
#include <stdio.h>
#include <string.h>

#include "sms.h"
#include "menu.h"
#include "lcd.h"
#include "time.h"
#include "timeout.h"
#include "modem.h"
#include "sleep.h"
#include "crypt.h"
#include "phone_book.h"

#define WAIT_TIME 2000

static uint8_t sms_inbox_offset;
static uint8_t sms_inbox_i;
static uint8_t sms_inbox_sz;
```



```
static char *sms_inbox_menu_title = NULL;
static char *sms_inbox_titles[3] = { NULL, NULL, NULL };
static uint8_t sms_inbox_read_offset;

static void sms_inbox_read_show_text(void);
static void sms_inbox_read_scroll_up(uint8_t ignored);
static void sms_inbox_read_scroll_down(uint8_t ignored);
static void sms_inbox_read(uint8_t ignored);
static void sms_inbox_show_menu(void);
static void sms_inbox_scroll_down(uint8_t ignored);
static void sms_inbox_scroll_up(uint8_t ignored);
static void sms_inbox(uint8_t ignored);
static void sms_write(uint8_t ignored);

char sms_dispatch_0[] PROGMEM = " ";
char sms_dispatch_1[] PROGMEM = ".,-!?";
char sms_dispatch_2[] PROGMEM = "abc";
char sms_dispatch_3[] PROGMEM = "def";
char sms_dispatch_4[] PROGMEM = "ghi";
char sms_dispatch_5[] PROGMEM = "jkl";
char sms_dispatch_6[] PROGMEM = "mno";
char sms_dispatch_7[] PROGMEM = "pqrs";
char sms_dispatch_8[] PROGMEM = "tuv";
char sms_dispatch_9[] PROGMEM = "wxyz";
PGM_P sms_dispatch[] PROGMEM = {
  sms_dispatch_0,
  sms_dispatch_1, sms_dispatch_2, sms_dispatch_3,
  sms_dispatch_4, sms_dispatch_5, sms_dispatch_6,
  sms_dispatch_7, sms_dispatch_8, sms_dispatch_9
};

uint8_t sms_pos;
char sms_last;
uint8_t sms_last_rep;

typedef enum {
  MAJ, MIN, NUM, DST
} sms_mode_t;
#define MODES 4
sms_mode_t sms_mode;

timeout_t sms_timeout;

static sms_t *sms_msg = NULL;

char sms_test_key[] = "ABCDEFGHIJKLMNOPQRSTUVWX";

void sms_init(void)
{
  uint8_t i;
  sms_msg = malloc(sizeof(sms_t));
  sms_inbox_menu_title = malloc(sizeof(char)*MENU_LINE_WIDTH);
```

```
  for (i = 0; i < 3; i++)
    sms_inbox_titles[i] = malloc(sizeof(char)*(MENU_LINE_WIDTH+1));
}

static void sms_encode(uint8_t *dst, uint8_t *src, uint8_t len)
{
  uint8_t i;
  for (i = 0; i < len; i++) {
    dst[i*2] = ('0' | (src[i] & 0x0F));
    dst[i*2+1] = ('0' | (src[i] >> 4));
  }
}

static void sms_decode(uint8_t *dst, uint8_t *src, uint8_t len)
{
  uint8_t i;
  for (i = 0; i < len; i++) {
    dst[i] = ((src[i*2+1] & 0x0F) << 4) | (src[i*2] & 0x0F);
  }
}

static void sms_encrypt(uint8_t *key)
{
  uint8_t len = strlen(sms_msg->text);
  if (len > 80) {
    sms_msg->text[79] = '\0';
    len = 80;
  }
  while (len % 8 != 0)
    sms_msg->text[len++] = '\0';
  printf_P(PSTR("encrypting, length %d\n"), len);
  crypt_encrypt((uint8_t*)&sms_msg->text[80], (uint8_t*)sms_msg->text,
     len, key);
  printf_P(PSTR("encoding ...\n"));
  sms_encode((uint8_t*)sms_msg->text, (uint8_t*)&sms_msg->text[80], len
     );
  sms_msg->text[len*2] = '\0';
}

static void sms_decrypt(uint8_t *key)
{
  uint8_t len = strlen(sms_msg->text)/2;
  sms_decode((uint8_t*)sms_msg->text, (uint8_t*)sms_msg->text, len);
  sms_msg->text[len] = '\0';
  crypt_decrypt((uint8_t*)sms_msg->text, (uint8_t*)sms_msg->text, len,
     key);
}

static uint8_t sms_looks_encrypted(void)
{
  uint8_t len = strlen(sms_msg->text);
  if (len % 16 != 0) return 0;
```



```
  uint8_t i;
  for (i = 0; i < len; i++)
    if (sms_msg->text[i] < '0' || sms_msg->text[i] > '?')
      return 0;
  return 1;
}

static void sms_goto(int8_t pos)
{
  if (pos < 0) pos = 0;
  sms_pos = pos;
  sms_last = 0;
  sms_last_rep = 0;
  menu_set_cursor(pos);
}

static void sms_putchar(char c)
{
  menu_setchar(sms_pos, (c=='\0'?' ':c));
  if (sms_pos < SMS_TEXT_LEN-1 && sms_mode != DST) {
    sms_msg->text[sms_pos] = c;
    sms_msg->text[sms_pos+1] = '\0';
  }
  menu_update_display();
}

static void sms_go_forward(void *data)
{
  sms_goto(sms_pos+1);
  menu_update_display();
}

static void sms_key_pressed(uint8_t c)
{
  /*
    if (c == '*') {
    sms_goto(sms_pos+1);
    menu_update_display();
    return;
    }
  */

  if (c == '#') {
    sms_mode++;
    if (sms_mode == MODES)
      sms_mode = 0;
    return;
  }

  if (c-'0' > 9 || c-'0' < 0) return;

  timeout_cancel(&sms_timeout);
```

```
  if (sms_last == 0) {
    sms_last = c;
  } else if (sms_last == c) {
    sms_last_rep++;
  } else {
    sms_goto(sms_pos+1);
    sms_last = c;
  }

  if (sms_mode == NUM || sms_mode == DST) {
    sms_putchar(c);
    sms_goto(sms_pos+1);
    menu_update_display();
  } else {
    PGM_P dispatch_entry = (PGM_P)pgm_read_word(&sms_dispatch[c-'0']);
    char ch = pgm_read_byte(&dispatch_entry[sms_last_rep]);
    if (ch == '\0')
      ch = pgm_read_byte(&dispatch_entry[sms_last_rep=0]);
    /*
      char ch = sms_dispatch[c-'0'][sms_last_rep];
      if (ch == '\0')
      ch = sms_dispatch[c-'0'][sms_last_rep=0];
    */
    if (sms_mode == MAJ)
      ch = toupper(ch);
    sms_putchar(ch);

    timeout_set(&sms_timeout, sms_go_forward, time_get()+WAIT_TIME,
       NULL);
  }
}

/*
static void sms_back(uint8_t ignored)
{
  sms_goto(sms_pos-1);
}
*/

static void sms_del(uint8_t ignored)
{
  timeout_cancel(&sms_timeout);
  if (sms_last == 0)
    sms_goto(sms_pos-1);
  sms_putchar('\0');
  sms_goto(sms_pos);
}

static void sms_send(uint8_t *crypt_key)
{
  menu_t *m;
```



```
  m = menu_add(PSTR("Sending SMS ..."), 1);
  menu_setup_done(m);

  // encrypt the message:
  //menu_set_cursor(0);
  //menu_update_display();
  if (crypt_key != NULL)
    sms_encrypt(crypt_key);

  uint8_t i;
  for (i = 0; i < strlen(sms_msg->number); i++)
    menu_setchar(i, sms_msg->number[i]);
  for (i = 0; i < strlen(sms_msg->text); i++)
    menu_setchar(i+MENU_LINE_WIDTH, sms_msg->text[i]);
  menu_set_cursor(MENU_LINE_WIDTH*2);
  menu_update_display();

  //lcd_goto(120);
  if (modem_send_sms(sms_msg)) {
    menu_pop();
    m = menu_add(PSTR("SMS sent"), 1);
    menu_setup_done(m);
  } else {
    menu_pop();
    m = menu_add(PSTR("SMS not sent"), 1);
    menu_setup_done(m);
  }
}

static void sms_send_uncrypted(uint8_t ignored)
{
  timeout_cancel(&sms_timeout);
  menu_get_text((uint8_t*)sms_msg->number, 8);
  sms_msg->number[8] = '\0';
  menu_pop();
  sms_send(NULL);
}

static void sms_enter_number(uint8_t ignored)
{
  timeout_cancel(&sms_timeout);

  /*
  if (sms_msg == NULL)
    sms_msg = malloc(sizeof(sms_t));
  menu_get_text((uint8_t*)sms_msg->text, 0);
  sms_msg->text[MENU_LINE_WIDTH*3] = '\0';
  uint8_t i;
  for (i = MENU_LINE_WIDTH*3-1; i > 0; i--) {
    if (sms_msg->text[i] == ' ')
      sms_msg->text[i] = '\0';
    else
```

```
      break;
  }
  */

  menu_pop();
  menu_t *m = menu_add(PSTR("Enter number"), 1);
  menu_set_abcd_item(m, 'A', PSTR("send"), sms_send_uncrypted);
  menu_set_abcd_item(m, 'D', PSTR("del"), sms_del);
  m->key_cb = sms_key_pressed;
  sms_mode = DST;
  menu_setup_done(m);
  sms_goto(0);
  menu_update_display();
}

static void sms_send_from_phone_book(phone_book_entry_t *entry)
{
  strncpy(sms_msg->number, (char*)entry->number, 12);
  sms_msg->number[11] = '\0';
  menu_pop();
  sms_send(entry->key);
}

static void sms_phone_book(uint8_t ignored)
{
  phone_book_menu("Send to", sms_send_from_phone_book);
}

static void sms_choose_number_method(uint8_t ignored)
{
  timeout_cancel(&sms_timeout);
  menu_pop();
  menu_t *m = menu_add(PSTR("Send SMS to ..."), 1);
  menu_add_item(m, PSTR("Number from phone book"), sms_phone_book);
  menu_add_item(m, PSTR("Enter number"), sms_enter_number);
  menu_setup_done(m);
}

static void sms_write(uint8_t ignored)
{
  menu_t *m = menu_add(PSTR("SMS"), 1);
  menu_set_abcd_item(m, 'A', PSTR("send"), sms_choose_number_method);
  //menu_set_abcd_item(m, 'B', PSTR("back"), sms_back);
  menu_set_abcd_item(m, 'D', PSTR("del"), sms_del);
  m->key_cb = sms_key_pressed;
  menu_setup_done(m);
  sms_goto(0);
  menu_update_display();
  sms_mode = MAJ;
  sms_msg->text[0] = '\0';
}
```



```
static void sms_inbox_read_show_text(void)
{
  uint8_t i = 0, j, text_i;
  uint8_t textlen = strlen(sms_msg->text);
  if (sms_inbox_read_offset == 0) {
    menu_setstring(0, sms_inbox_titles[sms_inbox_i]);
    /*
    menu_setstring(0, sms_msg->number);
    menu_setstring(strlen(sms_msg->number)+1, sms_msg->date);
    */
    i++;
  }
  for ( ; i < 3; i++) {
    for (j = 0; j < MENU_LINE_WIDTH; j++) {
      text_i = MENU_LINE_WIDTH*(i+sms_inbox_read_offset)+j -
    (sms_inbox_read_offset == 0 ? MENU_LINE_WIDTH : 0);
      if (text_i >= textlen) break;
      menu_setchar(MENU_LINE_WIDTH*i+j, sms_msg->text[text_i]);
    }
  }
  menu_update_display();
}

static void sms_inbox_read_scroll_up(uint8_t ignored)
{
  if (sms_inbox_read_offset > 0) {
    sms_inbox_read_offset -= 3;
    sms_inbox_read_show_text();
  }
}
static void sms_inbox_read_scroll_down(uint8_t ignored)
{
  if (sms_inbox_read_offset+3 < sms_inbox_sz) {
    sms_inbox_read_offset += 3;
    sms_inbox_read_show_text();
  }
}

extern uint8_t config_decrypt;

static void sms_inbox_read(uint8_t i)
{
  sms_inbox_i = i;

  modem_read_sms(sms_msg, sms_inbox_offset+i, 1);

  phone_book_entry_t *entry;
  switch (config_decrypt) {
  case 1:
    if (!sms_looks_encrypted()) break;
  case 0:
    entry = phone_book_search(sms_msg->number);
```

```
    if (entry != NULL)
      sms_decrypt(entry->key);
  }

  menu_t *m = menu_add(PSTR("Message"), 1);
  menu_set_abcd_item(m, 'A', PSTR("up"), sms_inbox_read_scroll_up);
  menu_set_abcd_item(m, 'B', PSTR("down"), sms_inbox_read_scroll_down);
  menu_setup_done(m);
  sms_inbox_read_offset = 0;
  sms_inbox_read_show_text();
}

static void sms_inbox_show_menu(void)
{
  uint8_t i;

  uint8_t pageno, pages;
  pages = sms_inbox_sz/3 + (sms_inbox_sz%3==0 ? 0 : 1);
  pageno = sms_inbox_offset/3 + 1;
  sprintf_P(sms_inbox_menu_title, PSTR("Inbox[%d/%d]"), pageno, pages);

  menu_t *m = menu_add(sms_inbox_menu_title, 0);
  for (i = 0; i < 3; i++) {
    if (i+sms_inbox_offset > sms_inbox_sz) {
      if (sms_inbox_sz == 0)
    menu_add_item(m, "(no messages)", NULL);
      break;
    }
    modem_read_sms(sms_msg, i+sms_inbox_offset, 0);
    phone_book_entry_t *sender = phone_book_search(sms_msg->number);
    snprintf(sms_inbox_titles[i], MENU_LINE_WIDTH, "%s %s",
         (sender != NULL ? sender->name : sms_msg->number),
         sms_msg->date);
    menu_add_item(m, sms_inbox_titles[i], sms_inbox_read);
  }
  menu_set_abcd_item(m, 'A', "up", sms_inbox_scroll_up);
  menu_set_abcd_item(m, 'B', "down", sms_inbox_scroll_down);
  menu_setup_done(m);
}

static void sms_inbox_scroll_down(uint8_t ignored)
{
  if (sms_inbox_offset+3 <= sms_inbox_sz) {
    menu_pop_update(0);
    sms_inbox_offset += 3;
    sms_inbox_show_menu();
  }
}
static void sms_inbox_scroll_up(uint8_t ignored)
{
  if (sms_inbox_offset-3 >= 1) {
    menu_pop_update(0);
```



```
    sms_inbox_offset -= 3;
    sms_inbox_show_menu();
  }
}

static void sms_inbox(uint8_t ignored)
{
  menu_t *m;
  int8_t messages_n = modem_get_sms_count();
  if (messages_n == -1) {
    m = menu_add(PSTR("Error reading inbox"), 1);
    menu_setup_done(m);
    return;
  }
  sms_inbox_sz = messages_n;
  sms_inbox_offset = 1;
  sms_inbox_show_menu();
}

void sms_menu(uint8_t ignored)
{
  menu_t *m = menu_add(PSTR("SMS"), 1);
  menu_add_item(m, PSTR("Inbox"), sms_inbox);
  menu_add_item(m, PSTR("New SMS"), sms_write);
  menu_setup_done(m);
}
```

Listing G.53: src/spi.c

```
/*! \file spi.c \brief SPI interface driver. */
//
   ***************************************************************************

//
// File Name    : 'spi.c'
// Title        : SPI interface driver
// Author       : Pascal Stang - Copyright (C) 2000-2002
// Created      : 11/22/2000
// Revised      : 06/06/2002
// Version      : 0.6
// Target MCU   : Atmel AVR series
// Editor Tabs  : 4
//
// NOTE: This code is currently below version 1.0, and therefore is
   considered
// to be lacking in some functionality or documentation, or may not be
   fully
// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
```

```
//
    ******************************************************************************


#include <avr/io.h>
#include <avr/interrupt.h>

#include "spi.h"

// Define the SPI_USEINT key if you want SPI bus operation to be
// interrupt-driven.  The primary reason for not using SPI in
// interrupt-driven mode is if the SPI send/transfer commands
// will be used from within some other interrupt service routine
// or if interrupts might be globally turned off due to of other
// aspects of your program
//
// Comment-out or uncomment this line as necessary
//#define SPI_USEINT

// global variables
volatile u08 spiTransferComplete;

// SPI interrupt service handler
#ifdef SPI_USEINT
SIGNAL(SIG_SPI)
{
    spiTransferComplete = TRUE;
}
#endif

// access routines
void spiInit()
{
#ifdef __AVR_ATmega128__
    // setup SPI I/O pins
    sbi(PORTB, 1);  // set SCK hi
    sbi(DDRB, 1);   // set SCK as output
    cbi(DDRB, 3);   // set MISO as input
    sbi(DDRB, 2);   // set MOSI as output
    sbi(DDRB, 0);   // SS must be output for Master mode to work
#elif __AVR_ATmega8__
    // setup SPI I/O pins
    sbi(PORTB, 5);  // set SCK hi
    sbi(DDRB, 5);   // set SCK as output
    cbi(DDRB, 4);   // set MISO as input
    sbi(DDRB, 3);   // set MOSI as output
    sbi(DDRB, 2);   // SS must be output for Master mode to work
#else
    // setup SPI I/O pins
    sbi(PORTB, 7);  // set SCK hi
    sbi(DDRB, 7);   // set SCK as output
    cbi(DDRB, 6);   // set MISO as input
```



```
    sbi(DDRB, 5);   // set MOSI as output
    sbi(DDRB, 4);   // SS must be output for Master mode to work
#endif

    // setup SPI interface :
    // master mode
    sbi(SPCR, MSTR);
    // clock = f/4
//  cbi(SPCR, SPR0);
//  cbi(SPCR, SPR1);
    // clock = f/16
    cbi(SPCR, SPR0);
    sbi(SPCR, SPR1);
    // select clock phase positive-going in middle of data
    cbi(SPCR, CPOL);
    // Data order MSB first
    cbi(SPCR,DORD);
    // enable SPI
    sbi(SPCR, SPE);


    // some other possible configs
    //outp((1<<MSTR)|(1<<SPE)|(1<<SPR0), SPCR );
    //outp((1<<CPHA)|(1<<CPOL)|(1<<MSTR)|(1<<SPE)|(1<<SPR0)|(1<<SPR1),
        SPCR );
    //outp((1<<CPHA)|(1<<MSTR)|(1<<SPE)|(1<<SPR0), SPCR );

    // clear status
    inb(SPSR);
    spiTransferComplete = TRUE;

    // enable SPI interrupt
    #ifdef SPI_USEINT
    sbi(SPCR, SPIE);
    #endif
}
/*
void spiSetBitrate(u08 spr)
{
    outb(SPCR, (inb(SPCR) & ((1<<SPR0)|(1<<SPR1))) | (spr&((1<<SPR0)
        |(1<<SPR1))));
}
*/
void spiSendByte(u08 data)
{
    // send a byte over SPI and ignore reply
    #ifdef SPI_USEINT
        while(!spiTransferComplete);
        spiTransferComplete = FALSE;
    #else
        while(!(inb(SPSR) & (1<<SPIF)));
    #endif
```

```
    outb(SPDR, data);
}

u08 spiTransferByte(u08 data)
{
    #ifdef SPI_USEINT
    // send the given data
    spiTransferComplete = FALSE;
    outb(SPDR, data);
    // wait for transfer to complete
    while(!spiTransferComplete);
    #else
    // send the given data
    outb(SPDR, data);
    // wait for transfer to complete
    while(!(inb(SPSR) & (1<<SPIF)));
    #endif
    // return the received data
    return inb(SPDR);
}

u16 spiTransferWord(u16 data)
{
    u16 rxData = 0;

    // send MS byte of given data
    rxData = (spiTransferByte((data>>8) & 0x00FF))<<8;
    // send LS byte of given data
    rxData |= (spiTransferByte(data & 0x00FF));

    // return the received data
    return rxData;
}
```

Listing G.54: src/tests.c

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdlib.h>

#include "fifo.h"
#include "lcd.h"
#include "modem.h"
#include "sleep.h"
#include "tests.h"
#include "thread.h"

//static thread_t modem_thread_data;

static thread_t sleep_thread_data;
```



```
//static fifo_t *fifo_test;
//static thread_t read_thread_data;
//static thread_t write_thread_data;

/*
static void modem_test_read_sms(void)
{
  sms_t *test_sms = malloc(sizeof(sms_t));
  sleep(2000);

  int8_t num_sms = modem_get_sms_count();
  printf_P(PSTR("Number of SMS messages: %d\n"), num_sms);

  sleep(2000);

  uint8_t test_i;
  for (test_i = 1; test_i <= num_sms; test_i++) {
    modem_read_sms(test_sms, test_i);
    printf_P(PSTR("%s\n"), test_sms->text);
    sleep(2000);
    printf_P(PSTR("From %s, %s\n"), test_sms->number, test_sms->date);
    sleep(2000);
  }
}

static void modem_thread(void *d)
{
  modem_init();
  //modem_enter_pin("4300");
  modem_enter_pin("7060");

  modem_test_read_sms();

  while (1) {
    //sleep(50);
    sleep(500);

    //printf("foo\n");

    //PORTD |= 1<<7;
    //sleep(60);
    //PORTD &= ~(1<<7);
  }
}
*/


static void sleep_thread(void *ignored)
{
  uint8_t i;
  const char *s = "|/-/";
  uint8_t pos;
```

```
  i = 0;
  while(1) {
    pos = lcd_get_pos();

    lcd_goto(39);
    putchar(s[i]);
    lcd_goto(pos);

    i++;
    if(s[i] == '\0') {
      i = 0;
    }

    sleep(200);
  }
}


/*
static void read_thread(void *ignored)
{
  uint8_t byte;
  while(1) {
    fifo_read_full(fifo_test, &byte, 1, FIFO_INFINITE);
    putchar(byte);
  }
}


static void write_thread(void *ignored)
{
  uint8_t byte;
  while(1) {
    for(byte = 'a'; byte <= 'z'; byte++) {
      fifo_write_full(fifo_test, &byte, 1, FIFO_INFINITE);
    }
  }
}
*/

/*
void tests_modem(void)
{
  thread_setup(&modem_thread_data, modem_thread, NULL);
}
*/

void tests_sleep(void)
{
  thread_setup(&sleep_thread_data, sleep_thread, NULL);
}
```



```
/*
void tests_fifo(void)
{
  fifo_test = fifo_alloc(10);

  thread_setup(&read_thread_data, read_thread, NULL);
  thread_setup(&write_thread_data, write_thread, NULL);
}
*/
```

Listing G.55: src/thread.c

```
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <stdio.h>
#include <string.h>

#include "assert.h"
#include "scheduler.h"
#include "thread.h"
#include "thread_switch.h"

/* Thread states: */
#define THREAD_STATE_RUNNABLE 0
#define THREAD_STATE_WAITING 1


/* This variable contains the current thread. */
thread_t *thread_current;


/* This function calculates the buffer pattern of a given index.
 *
 * Parameters:
 *  index           The index the pattern should be calculated for.
 *
 * Returns:
 *  The byte which should be located at the given index.
 */
static uint8_t thread_buffer_pattern(uint8_t index)
{
  return ((index+1) << 4) | index;
}


/* This function verifies the stack buffer of the current thread. An
 * error will be printed if the stack buffer doesn't match the buffer
 * pattern.
 */
static void thread_verify_stack_buffer(void)
{
```

```
  uint8_t i;
  uint8_t p;

  if(thread_current == NULL) {
    /* We aren't currently running any thread. */
    return;
  }

  for(i = 0; i < THREAD_STACK_BUFFER_SIZE; i++) {
    p = thread_buffer_pattern(i);
    if(p != thread_current->stack_buffer[i]) {
      printf_P(PSTR("Thread stack buffer mismatch. Index:%u, %02x !=
          %02x.\n"),
               i, thread_current->stack_buffer[i], p);
      for(;;);
    }
  }
}


/* This function transfers control to another thread without adding
   this
 * thread to the runqueue.
 *
 * Interrupts must be disabled when calling this function.
 */
static void thread_next(void)
{
  thread_t *prev_thread;

  /* Verify the stack. */
  thread_verify_stack_buffer();

  prev_thread = thread_current;

  thread_current = scheduler_get_next_thread();
  while(thread_current == NULL) {
    /* Enter sleep mode, waiting for an interrupt. */
    sleep_enable();
    sei();
    sleep_cpu();
    sleep_disable();
    cli();
    ASSERT_INTERRUPTS_OFF();

    /* Check if the interrupt(s) woke a thread. */
    thread_current = scheduler_get_next_thread();
  }

  if(thread_current != prev_thread) {
    thread_switch(thread_current->sp, &prev_thread->sp);
  }
```



```
}


/* This function initializes threading. */
void thread_init(void)
{
  thread_current = NULL;
  scheduler_init();
}


/* Contents of stack on thread startup:
 *
 * Pos  What                      Value
 * -36  <------------------------------------------------- Stack
    pointer
 * -35  SREG                      [Interrupts enabled]
 * -34  r31                       0
 * -33  r30                       0
 * -32  r29                       0
 * -31  r28                       0
 * -30  r27                       0
 * -29  r26                       0
 * -28  r25                       (((uint16_t)(data)) >> 8) & 0xff
 * -27  r24                       ((uint16_t)(data)) & 0xff
 * -26  r23                       0
 * -25  r22                       0
 * -24  r21                       0
 * -23  r20                       0
 * -22  r19                       0
 * -21  r18                       0
 * -20  r17                       0
 * -19  r16                       0
 * -18  r15                       0
 * -17  r14                       0
 * -16  r13                       0
 * -15  r12                       0
 * -14  r11                       0
 * -13  r10                       0
 * -12  r9                        0
 * -11  r8                        0
 * -10  r7                        0
 * -9   r6                        0
 * -8   r5                        0
 * -7   r4                        0
 * -6   r3                        0
 * -5   r2                        0
 * -4   r1                        0
 * -3   r0                        0
 * -2   Function address high.    (((uint16_t)(function)) >> 8) & 0xff
 * -1   Function address low.     ((uint16_t)(function)) & 0xff
 *
```

```
 * This must match the content of thread_switch.S.
 */

/* This function sets up a new thread and adds it to the run queue. The
 * thread function MUST never return. The thread will be started with
 * interrupts enabled.
 *
 * Parameters:
 *  TODO
 */
void thread_setup(thread_t *thread, thread_func_t function, void *data)
{
  uint8_t i;

  /* Clear the stack. */
  memset(thread->stack, 0, THREAD_STACK_SIZE);

  /* Set the return address to the function. */
  thread->stack[THREAD_STACK_SIZE - 1] = ((uint16_t)(function)) & 0xff;
  thread->stack[THREAD_STACK_SIZE - 2] = (((uint16_t)(function)) >> 8)
      & 0xff;

  /* Set the argument to the function. */
  thread->stack[THREAD_STACK_SIZE - 27] = ((uint16_t)(data)) & 0xff;
  thread->stack[THREAD_STACK_SIZE - 28] = (((uint16_t)(data)) >> 8) & 0
      xff;

  /* Set the status register.
   * SREG_I is the interrupt enable flag.
   */
  thread->stack[THREAD_STACK_SIZE - 35] = _BV(SREG_I);

  /* Set the stack pointer */
  thread->sp = &thread->stack[THREAD_STACK_SIZE - 36];

  /* Fill the thread stack buffer. */
  for(i = 0; i < THREAD_STACK_BUFFER_SIZE; i++) {
    thread->stack_buffer[i] = thread_buffer_pattern(i);
  }

  /* Wake the thread. */
  thread->state = THREAD_STATE_RUNNABLE;
  scheduler_add_thread(thread);
}


/* This function begins thread processing. It will never return.
 * At least one thread must be created before calling this function.
 *
 * Interrupts must be disabled when calling this function.
 */
void thread_begin(void)
```



```
{
  void *tmp;

  ASSERT_INTERRUPTS_OFF();

  thread_current = scheduler_get_next_thread();

  thread_switch(thread_current->sp, &tmp);
}


/* This function transfers control to another thread. The current
   thread
 * will be added to the runqueue.
 *
 * Interrupts must be disabled when calling this function.
 */
void thread_yield(void)
{
  ASSERT_INTERRUPTS_OFF();

  scheduler_add_thread(thread_current);

  thread_next();
}


/* This function marks the specified thread as runnable.
 *
 * Interrupts must be disabled when calling this function.
 *
 * Parameters:
 *  thread          The thread which should be marked as runnable.
 */
void thread_wake(thread_t *thread)
{
  ASSERT_INTERRUPTS_OFF();

  if(thread->state == THREAD_STATE_RUNNABLE) {
    /* This thread is already marked as runable. */
    return;
  }

  thread->state = THREAD_STATE_RUNNABLE;
  scheduler_add_thread(thread);
}


/* This function marks the current thread as waiting, and
 * transfers control to another thread.
 *
 * Interrupts must be disabled when calling this function.
```

```
 */
void thread_wait(void)
{
  ASSERT_INTERRUPTS_OFF();

  thread_current->state = THREAD_STATE_WAITING;
  thread_next();
}


/* This function verifies that the current stack pointer is
 * within the stack. An error will be printed and the program
 * will freeze if it is outside of the stack.
 */
void thread_verify_stack(void)
{
  if(thread_current == NULL) {
    /* We aren't currently running any thread. */
    return;
  }

  if(SP >= (uint16_t)&thread_current->stack[THREAD_STACK_SIZE]) {
    printf_P(PSTR("Stack pointer is above the stack.\n"));
    for(;;);
  }

  if(SP < (uint16_t)thread_current->stack) {
    printf_P(PSTR("Stack pointer is below the stack.\n"));
    for(;;);
  }
}
```

Listing G.56: src/thread_switch.S

```
#include <avr/io.h>

    .text

    ;; thread_switch is a function which stores all registers and the
    ;; status flag on the stack. Afterwards it replaces the stack
       pointer
    ;; with a new stack pointer and restores registers from that stack
    ;; pointer.
        ;;
        ;; This function can't be used to switch back to the currently
        ;; running thread.
    ;;
    ;; The C prototype for this function is:
    ;; void thread_switch(void *new_stack, void **old_stack);
    ;;
    ;; The old stack pointer is returned.
    ;;
```



```
    ;; r25:r24 contains the new stack pointer when the function is
       called.
        ;; r23:r22 contains the destination address for the old stack
        ;; pointer.
    ;; THIS FUNCTION MUST BE CALLED WITH INTERRUPTS DISABLED. FAILURE
       TO
    ;; DO THIS MAY RESULT IN AN INTERRUPT BEING PROCESSED WHEN WE HAVE
       AN
    ;; INVALID STACK POINTER.
.global thread_switch
    .type   thread_switch, @function
thread_switch:

    ;; First we save the current thread's state on the stack.

    ;; Push all registers except r24 & r25 to the stack.
    push r0
    push r1
    push r2
    push r3
    push r4
    push r5
    push r6
    push r7
    push r8
    push r9
    push r10
    push r11
    push r12
    push r13
    push r14
    push r15
    push r16
    push r17
    push r18
    push r19
    push r20
    push r21
    push r22
    push r23
        push r24
        push r25
    push r26
    push r27
    push r28
    push r29
    push r30
    push r31

    ;; We need to store the status register.
    in r0, _SFR_IO_ADDR(SREG)
    push r0
```

```
    ;; Time to replace the stack pointer.

    ;; First we load the current stack pointer into r1:r0.
    in r1, _SFR_IO_ADDR(SPH)
    in r0, _SFR_IO_ADDR(SPL)

        ;; Copy destination address to Z register.
        movw r30, r22

        ;; Then we store it in *old_stack (pointer r31:r30 - Z-register
           ).
        st Z, r0
        std Z+1, r1

    ;; Then we replace the stack pointer with the new stack pointer.
    out _SFR_IO_ADDR(SPH), r25
    out _SFR_IO_ADDR(SPL), r24

    ;; Now we have changed to the other thread's stack. Time to restore
    ;; the state of the thread.

    ;; Restore the status register.
    pop r0
    out _SFR_IO_ADDR(SREG), r0

    ;; Restore all registers except r24 and r25 from the new stack.
    pop r31
    pop r30
    pop r29
    pop r28
    pop r27
    pop r26
        pop r25
        pop r24
        pop r23
    pop r22
    pop r21
    pop r20
    pop r19
    pop r18
    pop r17
    pop r16
    pop r15
    pop r14
    pop r13
    pop r12
    pop r11
    pop r10
    pop r9
    pop r8
    pop r7
```



```
    pop r6
    pop r5
    pop r4
    pop r3
    pop r2
    pop r1
    pop r0

    ;; OK, we are done. Return.

    ret
```

Listing G.57: src/time.c

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>

#include "time.h"

static uint32_t current_time;

static time_wake_element_t *next_wake;

void time_init(void)
{
  /* Load oscilattor calibration value. This value should be read by
   * the avr programmmer, and set for each individual microcontroller.
   * To get the four different calibration bytes, you can run:
   *  make readcal
   */
  /* TODO: Load this value from the EEPROM. */
  OSCCAL = 0x9e;

  /* Reset the time. */
  current_time = 0;

  /* We want to wrap around and get an interrupt every 50000 us (50 ms)
     . */
  OCR1A = 50000;

  /* Timer 1 setup:
   * We want to increase the counter every 1 us (f_cpu / 8), and wrap
      around
   * when we reach OCR1A.
   *
   * (WGM13, WGM12, WGM11, WGM10) = 0100 (CTC).
   * (CS12, CS11, CS10) = 010 (clk_io/8).
   */
  TCCR1A = 0;
  TCCR1B = _BV(WGM12) | _BV(CS11);
  TCCR1C = 0;
```

```
  /* We want to generate an interrupt when we reach OCR0A. */
  TIMSK = _BV(OCIE1A);
}


uint32_t time_get(void)
{
  return current_time + TCNT1 / 1000;
}


/* Adds a wake event for the current thread at the specified time.
 *
 * This function must be called with interrupts disabled.
 */
void time_add_wake(time_wake_element_t *e, uint32_t wake_when)
{
  time_wake_element_t **i;

  e->thread = thread_current;
  e->wake_when = wake_when;

  /* Don't bother adding this wake event to the queue if it is infinite
     . */
  if(wake_when == TIME_INFINITE) {
    e->next = NULL;
    return;
  }

  if(next_wake == NULL) {
    next_wake = e;
    e->next = NULL;
    return;
  }

  for(i = &next_wake; *i != NULL; i = &(*i)->next) {
    if(wake_when < (*i)->wake_when) {
      e->next = *i;
      *i = e;
      return;
    }
  }

  *i = e;
  e->next = NULL;
}


/* Removes the specified wake event.
 *
 * This function must be called with interrupts disabled.
 */
```



```
void time_remove_wake(time_wake_element_t *e)
{
  time_wake_element_t **i;

  /* We won't find this wake event on the queue if it is infinite. */
  if(e->wake_when == TIME_INFINITE) {
    return;
  }

  for(i = &next_wake; *i != NULL; i = &(*i)->next) {
    if(*i == e) {
      *i = e->next;
      e->next = NULL;
      return;
    }
  }

  e->next = NULL;
}


ISR(TIMER1_COMPA_vect)
{
  time_wake_element_t *e;

  current_time += 50;

  while(next_wake && next_wake->wake_when <= current_time) {
    e = next_wake;
    next_wake = e->next;

    e->next = NULL;

    thread_wake(e->thread);
  }
}
```

Listing G.58: src/timeout.c

```
#include <stdint.h>
#include <stdlib.h>

#include "thread.h"
#include "timeout.h"
#include "time.h"
#include "sleep.h"

#define TIMEOUT_LIST_SZ 2
timeout_t *timeouts[TIMEOUT_LIST_SZ];
uint8_t n_timeouts = 0;

//thread_t *timeout_thread;
```

```
void timeout_run(void *data)
{
  uint8_t i;
  uint32_t time;
  while (1) {
    time = time_get();
    for (i = 0; i < n_timeouts; i++) {
      if (timeouts[i]->time < time) {
    timeouts[i]->fun(timeouts[i]->data);
    timeout_cancel(timeouts[i]);
      }
    }
    sleep(50);
  }
}

void timeout_init(thread_t *timeout_thread)
{
  thread_setup(timeout_thread, timeout_run, NULL);
}

void timeout_set(timeout_t *t, timeout_cb_t fun, uint32_t time, void *
    data)
{
  t->fun = fun;
  t->time = time;
  t->data = data;
  if (n_timeouts >= TIMEOUT_LIST_SZ) return;
  timeouts[n_timeouts++] = t;
}

void timeout_cancel(timeout_t *t)
{
  uint8_t i;
  for (i = 0; i < n_timeouts; i++)
    if (timeouts[i] == t)
      break;
  if (i == n_timeouts) return;
  n_timeouts--;
  for ( ; i < n_timeouts; i++)
    timeouts[i] = timeouts[i+1];
}
```

Listing G.59: src/ui.c

```
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <avr/pgmspace.h>
#include <avr/io.h>

#include "ui.h"
```



```
#include "thread.h"
#include "fifo.h"
#include "message.h"
#include "menu.h"
#include "sms.h"
#include "sleep.h"
#include "modem.h"
#include "timeout.h"
#include "phone_book.h"

static void ui_thread(void *ignored);
static void pin_menu(void);
static void main_menu(void);
static void recv_call_menu(void);
static void recv_sms(void);
static void ringtone(uint8_t on);

static thread_t ui_thread_data;
static timeout_t ui_ringtone_timeout;

void ui_init(void)
{
  thread_setup(&ui_thread_data, ui_thread, NULL);
}

static void ui_thread(void *ignored)
{
  message_t *msg;
  msg = message_wait_for(msg_to_ui, MSG_READY_FOR_PIN);
  free(msg);
  while (1) {
    pin_menu();
    msg = message_wait_for(msg_to_ui, MSG_PIN_RESULT);
    if (msg->data.pin_code_ok)
      break;
    free(msg);
    sleep(8000);
    menu_t *m = menu_add(PSTR("Wrong PIN"), 1);
    menu_setup_done(m);
    sleep(6000);
    menu_pop();
  }
  free(msg);

  main_menu();

  char key;
  while (1) {
    msg = message_get(msg_to_ui);
    switch (msg->type) {
    case MSG_KEY_PRESS:
      key = msg->data.key;
```

```
      menu_key_pressed(key);
      break;
    case MSG_RECV_SMS:
      recv_sms();
      menu_set_unread_sms(1);
      break;
    case MSG_NO_UNREAD_SMS:
      menu_set_unread_sms(0);
      break;
    case MSG_RECV_CALL:
      recv_call_menu();
      break;
    default:
      break;
    }
    free(msg);
  }
}

static void pin_menu(void)
{
  menu_t *m = menu_add(PSTR("Enter PIN code"), 1);
  menu_set_abcd_item(m, 'A', PSTR("ok"), NULL);
  menu_set_abcd_item(m, 'D', PSTR("del"), NULL);
  menu_setup_done(m);
  menu_set_cursor(0);
  menu_update_display();
  uint8_t pin_i = 0;
  char pin[5];
  while (1) {
    char key;
    message_t *msg = message_wait_for(msg_to_ui, MSG_KEY_PRESS);
    key = msg->data.key;
    free(msg);

    //printf_P(PSTR("pin, got key %c, pin_i=%d\n"), key, pin_i);

    if (key >= '0' && key <= '9' && pin_i < 4) {
      menu_setchar(pin_i, '*');
      menu_set_cursor(pin_i+1);
      pin[pin_i++] = key;
      menu_update_display();
    } else if (key == 'A' && pin_i == 4) {
      pin[pin_i++] = '\0';
      break;
    } else if (key == 'D' && pin_i > 0) {
      pin[pin_i--] = '\0';
      menu_setchar(pin_i, ' ');
      menu_update_display();
    }
  }
```



```
  menu_set_cursor(-1);
  menu_update_display();

  message_t *msg = malloc(sizeof(message_t));
  msg->type = MSG_PIN;
  strcpy(msg->data.pin_code, pin);
  message_put(msg_to_modem, msg);

  menu_pop();
}

static void signal_quality_menu(uint8_t ignored)
{
  menu_t *m = menu_add(PSTR("Signal quality"), 1);
  menu_setup_done(m);
  uint8_t sq = modem_signal_quality();
  char *str = malloc(8);
  sprintf(str, "%d", sq);
  menu_setstring(0, str);
  free(str);
  menu_update_display();
}

uint8_t config_decrypt = 1;

static void config_decrypt_set(uint8_t val)
{
  config_decrypt = val;
}

static void config_decrypt_menu(uint8_t ignored)
{
  menu_t *m = menu_add(PSTR("Decrypt SMS messages"), 1);
  menu_add_item(m, PSTR("Always"), config_decrypt_set);
  menu_add_item(m, PSTR("Guess based on content"), config_decrypt_set);
  menu_add_item(m, PSTR("Never"), config_decrypt_set);
  menu_setup_done(m);
}

static void config_menu(uint8_t ignored)
{
  menu_t *m = menu_add(PSTR("Config"), 1);
  menu_add_item(m, PSTR("Signal quality"), signal_quality_menu);
  menu_add_item(m, PSTR("SMS decryption"), config_decrypt_menu);
  menu_setup_done(m);
}

static void call(phone_book_entry_t *e)
{
  /* not implemented */
  menu_t *m = menu_add(PSTR("Call ..."), 1);
  menu_setup_done(m);
```

```
  menu_setstring(0, e->name);
  menu_setstring(MENU_LINE_WIDTH, e->number);
  menu_update_display();
}

static void phone_book_call_menu(uint8_t ignored)
{
  phone_book_menu("Phone book", call);
}

static void self_destruct_menu(uint8_t ignored)
{
  menu_t *m = menu_add(PSTR("SELF DESTRUCT"), 1);
  menu_setup_done(m);

  char str[8];
  int8_t i;
  for (i = 9; i >= 0; i--) {
    sleep(1000);
    snprintf_P(str, 8, PSTR("%d ..."), i);
    menu_setstring((9-i)*6, str);
    menu_update_display();
    ringtone(1);
    sleep(i==0?1000:100);
    ringtone(0);
  }

  menu_set_cursor(MENU_LINE_WIDTH*2);
  menu_update_display();
  modem_delete_all_sms();

  menu_pop();
}

static void main_menu(void)
{
  menu_t *m = menu_add(PSTR("Main menu"), 1);
  menu_add_item(m, PSTR("Phone book"), phone_book_call_menu);
  menu_add_item(m, PSTR("SMS"), sms_menu);
  menu_add_item(m, PSTR("Config"), config_menu);
  menu_add_item(m, PSTR("Self destruct"), self_destruct_menu);
  menu_setup_done(m);
  // test:
  /*
  menu_set_cursor(0);
  menu_update_display();
  */
}

static void recv_call_menu(void)
{
  menu_t *m = menu_add(PSTR("Receiving call"), 1);
```



```
  menu_setup_done(m);
  ringtone(1);
  sleep(500);
  ringtone(0);
}

static void recv_sms(void)
{
  //menu_t *m = menu_add(PSTR("New SMS received"), 1);
  //menu_setup_done(m);
  ringtone(1);
  sleep(1000);
  ringtone(0);
}

static void ringtone(uint8_t on)
{
  if (on)
    PORTD |= 1<<7;
  else
    PORTD &= ~(1<<7);
}
```

Listing G.60: src/usart.c

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <string.h>

#include "assert.h"
#include "fifo.h"
#include "usart.h"
#include "error.h"

#define BUFFER_SIZE 150

typedef struct {
  fifo_t *read_fifo;
  fifo_t *write_fifo;
} usart_port_t;

static usart_port_t usart[2];

/* This function resets the usart interface. */
void usart_init(void)
{
  uint8_t i;
  for(i = 0; i < 2; i++) {
    usart[0].read_fifo = NULL;
    usart[0].write_fifo = NULL;
  }
```

```
  UCSR0B = 0;
  UCSR1B = 0;
}

/* This function enables the data buffer empty interrupt for the
 * specified port.
 *
 * Parameters:
 *  port            Port number we should enable the interrupt on.
 */
static void usart_enable_udrie(uint8_t port)
{
  switch(port) {
  case 0:
    UCSR0B |= _BV(UDRIE0);
    break;
  case 1:
    UCSR1B |= _BV(UDRIE1);
    break;
  default:
    /* This case should never occur. */
    for(;;);
  }
}


/* This function disablesles the data buffer empty interrupt for the
 * specified port.
 *
 * Parameters:
 *  port            Port number we should disable the interrupt on.
 */
static void usart_disable_udrie(uint8_t port)
{
  switch(port) {
  case 0:
    UCSR0B &= ~_BV(UDRIE0);
    break;
  case 1:
    UCSR1B &= ~_BV(UDRIE1);
    break;
  default:
    /* This case should never occur. */
    for(;;);
  }
}


/* This function handles notifications we receive when a port is
   writeable.
 *
```



```
 * Parameters:
 *  port_ptr        The port number, casted to a pointer.
 */
static void usart_on_readable(fifo_t *fifo, void *port_ptr)
{
  uint16_t port;

  /* Convert the pointer to a port number. */
  port = (uint16_t)port_ptr;

  /* Enable the transmit interrupt. */
  usart_enable_udrie(port);
}

void usart_setup(uint8_t port)
{
  if(usart[port].read_fifo == NULL) {
    usart[port].read_fifo = fifo_alloc(BUFFER_SIZE);
  }

  if(usart[port].write_fifo == NULL) {
    usart[port].write_fifo = fifo_alloc(BUFFER_SIZE);
    fifo_set_notify_readable(usart[port].write_fifo, usart_on_readable,
                             (void*)((uint16_t)port));
  }

  if(port == 0) {
    /* Configures port 0 for 115200 baud rate, 8 data bits, 1 stop bit
     * and no parity.
     */
    UCSR0A = 0;
    UCSR0B = _BV(RXCIE0) | _BV(RXEN0) | _BV(TXEN0);
    UCSR0C = _BV(UCSZ01) | _BV(UCSZ00);

    UBRR0H = 0;
    UBRR0L = 3;
    /*
    UBRR0H = 0;
    UBRR0L = 51;
    */
  }
}

void usart_transmit(uint8_t port, const uint8_t *data, uint16_t length)
{
  fifo_write_full(usart[port].write_fifo, data, length, FIFO_INFINITE);
}


uint16_t usart_receive(uint8_t port, uint8_t *data, uint16_t length,
                       uint16_t timeout)
{
```

```
  usart_port_t *u = &usart[port];

  return fifo_read(u->read_fifo, data, length, timeout);
}


/* This is the interrupt handler for the transmit interrupt for usart
 * port 0.
 *
 * It is called every time there is space for one more byte in the
 * transmit buffer.
 */
ISR(USART0_UDRE_vect)
{
  uint8_t byte;
  uint16_t num_read;

  /* Make sure that there is space for one more byte. */
  if(!(UCSR0A & _BV(UDRE0))) {
    return;
  }

  /* Read the next byte we should send. */
  ASSERT_INTERRUPTS_OFF();
  num_read = fifo_read_interrupt(usart[0].write_fifo, &byte, 1);

  /* Disable the transmit interrupt if no data was available. */
  if(num_read == 0) {
    usart_disable_udrie(0);
    return;
  }

  /* Transmit the byte. */
  UDR0 = byte;
}

ISR(USART0_RX_vect){
  uint8_t status = UCSR0A;
  uint16_t nw;

  if(status & _BV(FE0)) {
    fatal_error_P(PSTR("Frame error...\n"));
  }

  if(status & _BV(DOR0)) {
    fatal_error_P(PSTR("Data overrun...\n"));
  }

  if(status & _BV(UPE0)) {
    fatal_error_P(PSTR("Parity error...\n"));
  }
```



```
  uint8_t data = UDR0;
  nw = fifo_write_interrupt(usart[0].read_fifo, &data, 1);
  if(nw == 0) {
    fatal_error_P(PSTR("USART receive buffer full...\n"));
  }
}

/*
 * Ugly hack, for use when we are waiting for response from modem but
 * it isn't responding. Doesn't really work.
 */
void usart_kick(void)
{
  char kick_msg[] = "OK\r\n";
  fifo_write_full(usart[0].read_fifo, kick_msg, strlen(kick_msg),
     FIFO_INFINITE);
}
```

Listing G.61: src/wait_queue.c

```
#include <stdlib.h>

#include "wait_queue.h"


/* Initializes a wait queue. */
void wait_queue_init(wait_queue_t *queue)
{
  queue->first = NULL;
  queue->last = NULL;
}

/* Wake all threads which are waiting on the specified wait queue.
 *
 * This function must be called with interrupts disabled.
 */
void wait_queue_wake_all(wait_queue_t *queue)
{
  wait_queue_element_t *e;
  for(e = queue->first; e != NULL; e = e->next) {
    thread_wake(e->thread);
  }
}


/* Adds the current thread to the specified wait queue.
 *
 * This function must be called with interrupts disabled.
 */
void wait_queue_add_thread(wait_queue_t *queue,
                           wait_queue_element_t *element)
{
  element->thread = thread_current;
```

```
  element->next = NULL;
  element->prev = queue->last;

  if(queue->last != NULL) {
    queue->last->next = element;
  }

  queue->last = element;

  if(queue->first == NULL) {
    queue->first = NULL;
  }
}


/* Removes the current thread (with the specified wait queue element)
 * from the specified wait queue.
 *
 * This function must be called with interrupts disabled.
 */
void wait_queue_remove_thread(wait_queue_t *queue,
                              wait_queue_element_t *element)
{
  if(element->prev == NULL && element->next == NULL) {
    if(queue->first == element) {
      queue->first = NULL;
      queue->last = NULL;
    }
    return;
  }

  if(element->prev != NULL) {
    element->prev->next = element->next;
  } else {
    queue->first = element->next;
  }

  if(element->next != NULL) {
    element->next->prev = element->prev;
  } else {
    queue->last = element->prev;
  }

  element->prev = NULL;
  element->next = NULL;
}
```

Listing G.62: inc/assert.h

```
#ifndef ASSERT_H
#define ASSERT_H
```



```
#include <avr/io.h>
#include <avr/pgmspace.h>

#include "error.h"

#define DIE() \
  do { \
    fatal_error_P(PSTR("D %s:%i"), __FILE__, __LINE__); \
  } while(0)

#define ASSERT(x) \
  do { \
    if(!(x)) { \
      fatal_error_P(PSTR("AF %s:%i"), __FILE__, __LINE__); \
    } \
  } while(0)

#define ASSERT_INTERRUPTS_OFF() \
  do { \
    if(SREG & _BV(SREG_I)) { \
      fatal_error_P(PSTR("ANI %02x %s:%i\n"), SREG, __FILE__, __LINE__)
          ; \
    } \
  } while(0)

#define ASSERT_INTERRUPTS_ON() \
  do { \
    if(!(SREG & _BV(SREG_I))) {                      \
      fatal_error_P(PSTR("AI %02x %s:%i\n"), SREG, __FILE__, __LINE__);
          \
    } \
  } while(0)



#endif /* ASSERT_H */
```

Listing G.63: inc/avrlibdefs.h

```
/*! \file avrlibdefs.h \brief AVRlib global defines and macros. */
//
   ************************************************************************

//
// File Name    : 'avrlibdefs.h'
// Title        : AVRlib global defines and macros include file
// Author       : Pascal Stang
// Created      : 7/12/2001
// Revised      : 9/30/2002
// Version      : 1.1
// Target MCU   : Atmel AVR series
// Editor Tabs  : 4
//
```

```
//  Description : This include file is designed to contain items useful
    to all
//                  code files and projects, regardless of specific
   implementation.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
   ************************************************************************



#ifndef AVRLIBDEFS_H
#define AVRLIBDEFS_H

// Code compatibility to new AVR-libc
// outb(), inb(), inw(), outw(), BV(), sbi(), cbi(), sei(), cli()
#ifndef outb
    #define outb(addr, data)    addr = (data)
#endif
#ifndef inb
    #define inb(addr)           (addr)
#endif
#ifndef outw
    #define outw(addr, data)    addr = (data)
#endif
#ifndef inw
    #define inw(addr)           (addr)
#endif
#ifndef BV
    #define BV(bit)         (1<<(bit))
#endif
#ifndef cbi
    #define cbi(reg,bit)    reg &= ~(BV(bit))
#endif
#ifndef sbi
    #define sbi(reg,bit)    reg |= (BV(bit))
#endif
#ifndef cli
    #define cli()           __asm__ __volatile__ ("cli" ::)
#endif
#ifndef sei
    #define sei()           __asm__ __volatile__ ("sei" ::)
#endif

// support for individual port pin naming in the mega128
// see port128.h for details
#ifdef __AVR_ATmega128__
// not currently necessary due to inclusion
// of these defines in newest AVR-GCC
// do a quick test to see if include is needed
```



```
#ifndef PD0
    #include "port128.h"
#endif
#endif

// use this for packed structures
// (this is seldom necessary on an 8-bit architecture like AVR,
//  but can assist in code portability to AVR)
#define GNUC_PACKED __attribute__((packed))

// port address helpers
#define DDR(x) ((x)-1)    // address of data direction register of port
    x
#define PIN(x) ((x)-2)    // address of input register of port x

// MIN/MAX/ABS macros
#define MIN(a,b)            ((a<b)?(a):(b))
#define MAX(a,b)            ((a>b)?(a):(b))
#define ABS(x)              ((x>0)?(x):(-x))

// constants
#define PI      3.14159265359

#endif
```

Listing G.64: inc/avrlibtypes.h

```
/*! \file avrlibtypes.h \brief AVRlib global types and typedefines. */
//
   ************************************************************************

//
// File Name    : 'avrlibtypes.h'
// Title        : AVRlib global types and typedefines include file
// Author       : Pascal Stang
// Created      : 7/12/2001
// Revised      : 9/30/2002
// Version      : 1.0
// Target MCU   : Atmel AVR series
// Editor Tabs  : 4
//
//  Description : Type-defines required and used by AVRlib.  Most types
    are also
//                      generally useful.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
   ************************************************************************
```

```
#ifndef AVRLIBTYPES_H
#define AVRLIBTYPES_H

#ifndef WIN32
    // true/false defines
    #define FALSE   0
    #define TRUE    -1
#endif

// datatype definitions macros
typedef unsigned char  u08;
typedef   signed char  s08;
typedef unsigned short u16;
typedef   signed short s16;
typedef unsigned long  u32;
typedef   signed long  s32;
typedef unsigned long long u64;
typedef   signed long long s64;

/* use inttypes.h instead
// C99 standard integer type definitions
typedef unsigned char   uint8_t;
typedef   signed char   int8_t;
typedef unsigned short  uint16_t;
typedef   signed short  int16_t;
typedef unsigned long   uint32_t;
typedef   signed long   int32_t;
typedef unsigned long   uint64_t;
typedef   signed long   int64_t;
*/
// maximum value that can be held
// by unsigned data types (8,16,32bits)
#define MAX_U08 255
#define MAX_U16 65535
#define MAX_U32 4294967295

// maximum values that can be held
// by signed data types (8,16,32bits)
#define MIN_S08 -128
#define MAX_S08 127
#define MIN_S16 -32768
#define MAX_S16 32767
#define MIN_S32 -2147483648
#define MAX_S32 2147483647

#ifndef WIN32
    // more type redefinitions
    typedef unsigned char   BOOL;
    typedef unsigned char   BYTE;
    typedef unsigned int    WORD;
    typedef unsigned long   DWORD;
```



```
    typedef unsigned char   UCHAR;
    typedef unsigned int    UINT;
    typedef unsigned short  USHORT;
    typedef unsigned long   ULONG;

    typedef char            CHAR;
    typedef int             INT;
    typedef long            LONG;
#endif

#endif
```

Listing G.65: inc/config.h

```
#ifndef __EFSL_CONFIG_H__
#define __EFSL_CONFIG_H__

/* Hardware target
   ---------------

 * Here you will define for what hardware-endpoint EFSL should be
    compiled.
 * Look in interfaces.h to see what systems are supported, and add your
    own
 * there if you need to write your own driver. Then, define the name
    you
 * selected for your hardware there here. Make sure that you only
    select one
 * device!
*/

    /*#define HW_ENDPOINT_LINUX*/
    #define HW_ENDPOINT_ATMEGA128_SD
    /*#define HW_ENDPOINT_DSP_TI6713_SD*/


/* Memory configuration
   --------------------

 * Here you must configure wheter your processor can access memory byte
 * oriented. All x86 processors can do it, AVR's can do it to. Some DSP
 * or other microcontrollers can't. If you have an 8 bit system you're
    safe.
 * If you are really unsure, leave the setting commented out, it will
    be slower
 * but it will work for sure.
*/

    #define BYTE_ALIGNMENT

/* Cache configuration
   -------------------
```

```
 * Here you must configure how much memory of cache you can/want to use
    .
 * The number you put at IOMAN_NUMBUFFER is multiplied by 512. So 1
    means
 * 512 bytes cache, 4 means 2048 bytes cache. More is better.
 * The number after IOMAN_NUMITERATIONS should be untouched.
 * The last field (IOMAN_DO_MEMALLOC) is to tell ioman to allocate it's
 * own memory in it's structure, or not. If you choose to do it
    yourself
 * you will have to pass a pointer to the memory as the last argument
    of
 * ioman_init.
*/
    #define IOMAN_NUMBUFFER 1
    #define IOMAN_NUMITERATIONS 3
    #define IOMAN_DO_MEMALLOC

/* Cluster pre-allocation
   ----------------------

 * When writing files, the function that performs the actual write has
    to
 * calculate how many clusters it will need for that request. It then
    allocates
 * that number of new clusters to the file. Since this involves some
 * calculations and writing of the FAT, you might find it beneficial to
     limit
 * the number of allocations, and allow fwrite to pre-allocate a number
     of
 * clusters extra. This setting determines how many clusters will be
    extra
 * allocated whenever this is required.
 * Take in carefull consideration how large your clustersize is,
    putting 10 here
 * with a clustersize of 32kb means you might waste 320 kb.
 * The first option is for preallocating files, the other is used when
    enlarging
 * a directory to accomodate more files
*/
      #define CLUSTER_PREALLOC_FILE 0
      #define CLUSTER_PREALLOC_DIRECTORY 0


/* Endianess configuration
   -----------------------

 * Here you can configure wheter your architecture is little or big
    endian. This
 * is important since all FAT structures are stored in intel little
    endian
 * order. So if you have a big endian system the library has to convert
     all
```



```
 * figures to big endian in order to work.
 */
    #define LITTLE_ENDIAN


/* Date and Time support
   ---------------------

 * Here you can enable or disable date and time support. If you enable
 * it you will have to create 6 functions, that are described in the
 * EFSL manual. If the functions are not present when linking your
 * program with the library you will get unresolved dependencies.
 */
    /* #define DATE_TIME_SUPPORT */

/* Error reporting support
   -----------------------

 * When you receive an error in userland, it usually only gives limited
 * information (most likely, fail or success). If error detection and
 * reporting is important for you, you can enable more detailed error
 * reporting here. This is optional, the costs are 1 byte per object,
 * and a small increase in code size.
 * You can enable error recording for all object, or you can select the
 * object manually.
 * For full error reporting use FULL_ERROR_SUPPORT
 * For only the base-core of the library use BASE_ERROR_SUPPORT
 * For IO/Man use ERRSUP_IOMAN
 * For Disc   use ERRSUP_IOMAN
 * For Part   use ERRSUP_PARTITION
 * For Fs     use ERRSUP_FILESYSTEM
 * For File   use ERRSUP_FILE
*/

    #define FULL_ERROR_SUPPORT
    /*#define BASE_ERROR_SUPPORT*/

/* List options
   ------------

 * In this section you can configure what kind of data you will get
    from
 * directory listing requests. Please refer to the documentation for
 * more information
*/

#define LIST_MAXLENFILENAME 12


/* Debugging configuration
   -----------------------
```

```
 * Here you can configure the debugging behaviour. Debugging is
    different
 * on every platform (see debug.h for more information).
 * If your hardware has no means of output (printf) dont define any
    anything,
 * and nothing will happen. For real world use debugging should be
    turned off.
*/

    #define DEBUG


/* Debugging configuration - AVR Specific: PORT
   ---------------------------------------------

 * Here you can select which UART you want to use for debugging.
 * If you did not define DEBUG, this setting has no effect.
 * Note that it is not a good idea to use a port that you use in
    userspace.
*/

    #define DEBUG_PORT 0        /* Select UART0 */
    /*#define DEBUG_PORT 1*/    /* Select UART1 */


/* Debugging configuration - AVR Specific: UBRR
   ---------------------------------------------

 * Here you can set UBRR, this value will select the serial clock speed
    .
 * This value depends on your baudrate and clockrate. U2X is by
    standard 0,
 * if you would want this 1 for some reason, this can be done in debug.
    c.
*/

    #define DEBUG_UBRR 51       /* 9600bps on 8Mhz */
    /*#define DEBUG_UBRR 95*/   /* 9600bps on 14.7456Mhz */
    /*#define DEBUG_UBRR 103*/  /* 9600bps on 16Mhz */


#endif
```

Listing G.66: inc/crypt.h

```
#ifndef CRYPT_H
#define CRYPT_H

void crypt_encrypt(uint8_t *dest, uint8_t *src, uint8_t len, uint8_t *
   key);
void crypt_decrypt(uint8_t *dest, uint8_t *src, uint8_t len, uint8_t *
   key);
```



```
#endif
```

Listing G.67: inc/delay.h

```
#ifndef DELAY_H
#define DELAY_H

#include <stdint.h>

void delay(uint32_t us);

#endif /* DELAY_H */
```

Listing G.68: inc/error.h

```
#ifndef ERROR_H
#define ERROR_H

void fatal_error(const char *fmt, ...);
void fatal_error_P(const char *fmt, ...);
void error(const char *fmt, ...);
void error_P(const char *fmt, ...);

#endif
```

Listing G.69: inc/fat.h

```
/*! \file fat.h \brief FAT16/32 file system driver. */
//
   *****************************************************************************

//
// File Name    : 'fat.h'
// Title        : FAT16/32 file system driver
// Author       : Pascal Stang
// Date         : 11/07/2000
// Revised      : 12/12/2000
// Version      : 0.3
// Target MCU   : ATmega103 (should work for Atmel AVR Series)
// Editor Tabs  : 4
//
// NOTE: This code is currently below version 1.0, and therefore is
   considered
// to be lacking in some functionality or documentation, or may not be
   fully
// tested.  Nonetheless, you can expect most functions to work.
//
/// \ingroup general
/// \defgroup fat FAT16/32 File System Interface (fat.c)
/// \code #include "fat.h" \endcode
/// \par Overview
///     This FAT16/32 interface allows you to detect and mount FAT16/32
///     partitions, browse directories and files, and read file data.
```

```
///     The interface is designed to operate with the avrlib IDE/ATA
    driver.
///     Reading FAT efficiently requires at least 512+ bytes of RAM so
    this
///     interface may not be suitable for processors with less than 1K
    of RAM.
///     This interface will properly follow a file's cluster chain so
    files
///     need not be defragmented.
///
/// \note This code is based in part on work done by Jesper Hansen for
    his
///     excellent YAMPP MP3 player project.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
    ***************************************************************************


#ifndef FAT_H
#define FAT_H

#include "global.h"


// Some useful cluster numbers
#define MSDOSFSROOT     0               // cluster 0 means the root dir
#define CLUST_FREE      0               // cluster 0 also means a free
    cluster
#define MSDOSFSFREE     CLUST_FREE
#define CLUST_FIRST     2               // first legal cluster number
#define CLUST_RSRVD     0xfffffff6      // reserved cluster range
#define CLUST_BAD       0xfffffff7      // a cluster with a defect
#define CLUST_EOFS      0xfffffff8      // start of eof cluster range
#define CLUST_EOFE      0xffffffff      // end of eof cluster range

#define FAT12_MASK      0x00000fff      // mask for 12 bit cluster
    numbers
#define FAT16_MASK      0x0000ffff      // mask for 16 bit cluster
    numbers
#define FAT32_MASK      0x0fffffff      // mask for FAT32 cluster
    numbers


// Partition Type used in the partition record
#define PART_TYPE_UNKNOWN       0x00
#define PART_TYPE_FAT12         0x01
#define PART_TYPE_XENIX         0x02
#define PART_TYPE_DOSFAT16      0x04
#define PART_TYPE_EXTDOS        0x05
```



```
#define PART_TYPE_FAT16         0x06
#define PART_TYPE_NTFS          0x07
#define PART_TYPE_FAT32         0x0B
#define PART_TYPE_FAT32LBA      0x0C
#define PART_TYPE_FAT16LBA      0x0E
#define PART_TYPE_EXTDOSLBA     0x0F
#define PART_TYPE_ONTRACK       0x33
#define PART_TYPE_NOVELL        0x40
#define PART_TYPE_PCIX          0x4B
#define PART_TYPE_PHOENIXSAVE   0xA0
#define PART_TYPE_CPM           0xDB
#define PART_TYPE_DBFS          0xE0
#define PART_TYPE_BBT           0xFF

struct partrecord // length 16 bytes
{
    BYTE    prIsActive;                 // 0x80 indicates active
       partition
    BYTE    prStartHead;                // starting head for partition
    WORD    prStartCylSect;             // starting cylinder and sector
    BYTE    prPartType;                 // partition type (see above)
    BYTE    prEndHead;                  // ending head for this
       partition
    WORD    prEndCylSect;               // ending cylinder and sector
    DWORD   prStartLBA;                 // first LBA sector for this
       partition
    DWORD   prSize;                     // size of this partition (
       bytes or sectors ?)
};


struct partsector
{
    CHAR    psPartCode[512-64-2];       // pad so struct is 512b
    BYTE    psPart[64];                 // four partition records (64
       bytes)
    BYTE    psBootSectSig0;             // two signature bytes (2 bytes
       )
    BYTE    psBootSectSig1;
#define BOOTSIG0        0x55
#define BOOTSIG1        0xaa
};



// Format of a boot sector.  This is the first sector on a DOS floppy
    disk
// or the first sector of a partition on a hard disk.  But, it is not
    the
// first sector of a partitioned hard disk.
struct bootsector33 {
    BYTE    bsJump[3];                  // jump inst E9xxxx or EBxx90
```

```
    CHAR    bsOemName[8];           // OEM name and version
    CHAR    bsBPB[19];              // BIOS parameter block
    CHAR    bsDriveNumber;          // drive number (0x80)
    CHAR    bsBootCode[479];        // pad so struct is 512b
    BYTE    bsBootSectSig0;         // boot sector signature byte 0
       x55
    BYTE    bsBootSectSig1;         // boot sector signature byte 0
       xAA
#define BOOTSIG0        0x55
#define BOOTSIG1        0xaa
};

struct extboot {
    CHAR    exDriveNumber;          // drive number (0x80)
    CHAR    exReserved1;            // reserved
    CHAR    exBootSignature;        // ext. boot signature (0x29)
#define EXBOOTSIG       0x29
    CHAR    exVolumeID[4];          // volume ID number
    CHAR    exVolumeLabel[11];      // volume label
    CHAR    exFileSysType[8];       // fs type (FAT12 or FAT16)
};

struct bootsector50 {
    BYTE    bsJump[3];              // jump inst E9xxxx or EBxx90
    CHAR    bsOemName[8];           // OEM name and version
    CHAR    bsBPB[25];              // BIOS parameter block
    CHAR    bsExt[26];              // Bootsector Extension
    CHAR    bsBootCode[448];        // pad so structure is 512b
    BYTE    bsBootSectSig0;         // boot sector signature byte 0
       x55
    BYTE    bsBootSectSig1;         // boot sector signature byte 0
       xAA
#define BOOTSIG0        0x55
#define BOOTSIG1        0xaa
};

struct bootsector710 {
    BYTE    bsJump[3];              // jump inst E9xxxx or EBxx90
    CHAR    bsOEMName[8];           // OEM name and version
    CHAR    bsBPB[53];              // BIOS parameter block
    CHAR    bsExt[26];              // Bootsector Extension
    CHAR    bsBootCode[418];        // pad so structure is 512b
    BYTE    bsBootSectSig2;         // 2 & 3 are only defined for
       FAT32?
    BYTE    bsBootSectSig3;
    BYTE    bsBootSectSig0;         // boot sector signature byte 0
       x55
    BYTE    bsBootSectSig1;         // boot sector signature byte 0
       xAA
#define BOOTSIG0        0x55
#define BOOTSIG1        0xaa
#define BOOTSIG2        0
```



```
#define BOOTSIG3        0
};


/*************************************************************/
/*************************************************************/

// BIOS Parameter Block (BPB) for DOS 3.3
struct bpb33 {
        WORD    bpbBytesPerSec; // bytes per sector
        BYTE    bpbSecPerClust; // sectors per cluster
        WORD    bpbResSectors;  // number of reserved sectors
        BYTE    bpbFATs;        // number of FATs
        WORD    bpbRootDirEnts; // number of root directory entries
        WORD    bpbSectors;     // total number of sectors
        BYTE    bpbMedia;       // media descriptor
        WORD    bpbFATsecs;     // number of sectors per FAT
        WORD    bpbSecPerTrack; // sectors per track
        WORD    bpbHeads;       // number of heads
        WORD    bpbHiddenSecs;  // number of hidden sectors
};

// BPB for DOS 5.0
// The difference is bpbHiddenSecs is a short for DOS 3.3,
// and bpbHugeSectors is not present in the DOS 3.3 bpb.
struct bpb50 {
        WORD    bpbBytesPerSec; // bytes per sector
        BYTE    bpbSecPerClust; // sectors per cluster
        WORD    bpbResSectors;  // number of reserved sectors
        BYTE    bpbFATs;        // number of FATs
        WORD    bpbRootDirEnts; // number of root directory entries
        WORD    bpbSectors;     // total number of sectors
        BYTE    bpbMedia;       // media descriptor
        WORD    bpbFATsecs;     // number of sectors per FAT
        WORD    bpbSecPerTrack; // sectors per track
        WORD    bpbHeads;       // number of heads
        DWORD   bpbHiddenSecs;  // # of hidden sectors
// 3.3 compat ends here
        DWORD   bpbHugeSectors; // # of sectors if bpbSectors == 0
};

// BPB for DOS 7.10 (FAT32)
// This one has a few extensions to bpb50.
struct bpb710 {
        WORD    bpbBytesPerSec; // bytes per sector
        BYTE    bpbSecPerClust; // sectors per cluster
        WORD    bpbResSectors;  // number of reserved sectors
        BYTE    bpbFATs;        // number of FATs
        WORD    bpbRootDirEnts; // number of root directory entries
        WORD    bpbSectors;     // total number of sectors
        BYTE    bpbMedia;       // media descriptor
        WORD    bpbFATsecs;     // number of sectors per FAT
```

```
        WORD    bpbSecPerTrack; // sectors per track
        WORD    bpbHeads;       // number of heads
        DWORD   bpbHiddenSecs;  // # of hidden sectors
// 3.3 compat ends here
        DWORD   bpbHugeSectors; // # of sectors if bpbSectors == 0
// 5.0 compat ends here
        DWORD     bpbBigFATsecs;// like bpbFATsecs for FAT32
        WORD      bpbExtFlags;  // extended flags:
#define FATNUM    0xf           // mask for numbering active FAT
#define FATMIRROR 0x80          // FAT is mirrored (like it always was)
        WORD      bpbFSVers;    // filesystem version
#define FSVERS    0             // currently only 0 is understood
        DWORD     bpbRootClust; // start cluster for root directory
        WORD      bpbFSInfo;    // filesystem info structure sector
        WORD      bpbBackup;    // backup boot sector
        // There is a 12 byte filler here, but we ignore it
};




// ************************************************************
// * byte versions of the above structs                       *
// ************************************************************


// BIOS Parameter Block (BPB) for DOS 3.3
struct byte_bpb33 {
        CHAR bpbBytesPerSec[2];     // bytes per sector
        CHAR bpbSecPerClust;        // sectors per cluster
        CHAR bpbResSectors[2];      // number of reserved sectors
        CHAR bpbFATs;               // number of FATs
        CHAR bpbRootDirEnts[2];     // number of root directory entries
        CHAR bpbSectors[2];         // total number of sectors
        CHAR bpbMedia;              // media descriptor
        CHAR bpbFATsecs[2];         // number of sectors per FAT
        CHAR bpbSecPerTrack[2];     // sectors per track
        CHAR bpbHeads[2];           // number of heads
        CHAR bpbHiddenSecs[2];      // number of hidden sectors
};

// BPB for DOS 5.0
// The difference is bpbHiddenSecs is a short for DOS 3.3,
// and bpbHugeSectors is not in the 3.3 bpb.
struct byte_bpb50 {
        CHAR bpbBytesPerSec[2];     // bytes per sector
        CHAR bpbSecPerClust;        // sectors per cluster
        CHAR bpbResSectors[2];      // number of reserved sectors
        CHAR bpbFATs;               // number of FATs
        CHAR bpbRootDirEnts[2];     // number of root directory entries
        CHAR bpbSectors[2];         // total number of sectors
        CHAR bpbMedia;              // media descriptor
```



```
        CHAR bpbFATsecs[2];         // number of sectors per FAT
        CHAR bpbSecPerTrack[2];     // sectors per track
        CHAR bpbHeads[2];           // number of heads
        CHAR bpbHiddenSecs[4];      // number of hidden sectors
        CHAR bpbHugeSectors[4];     // # of sectors if bpbSectors == 0
};

// BPB for DOS 7.10 (FAT32).
// This one has a few extensions to bpb50.
struct byte_bpb710 {
        BYTE bpbBytesPerSec[2];     // bytes per sector
        BYTE bpbSecPerClust;        // sectors per cluster
        BYTE bpbResSectors[2];      // number of reserved sectors
        BYTE bpbFATs;               // number of FATs
        BYTE bpbRootDirEnts[2];     // number of root directory entries
        BYTE bpbSectors[2];         // total number of sectors
        BYTE bpbMedia;              // media descriptor
        BYTE bpbFATsecs[2];         // number of sectors per FAT
        BYTE bpbSecPerTrack[2];     // sectors per track
        BYTE bpbHeads[2];           // number of heads
        BYTE bpbHiddenSecs[4];      // # of hidden sectors
        BYTE bpbHugeSectors[4];     // # of sectors if bpbSectors == 0
        BYTE bpbBigFATsecs[4];      // like bpbFATsecs for FAT32
        BYTE bpbExtFlags[2];        // extended flags:
        BYTE bpbFSVers[2];          // filesystem version
        BYTE bpbRootClust[4];       // start cluster for root directory
        BYTE bpbFSInfo[2];          // filesystem info structure sector
        BYTE bpbBackup[2];          // backup boot sector
        // There is a 12 byte filler here, but we ignore it
};

// FAT32 FSInfo block.
struct fsinfo {
        BYTE fsisig1[4];
        BYTE fsifill1[480];
        BYTE fsisig2[4];
        BYTE fsinfree[4];
        BYTE fsinxtfree[4];
        BYTE fsifill2[12];
        BYTE fsisig3[4];
        BYTE fsifill3[508];
        BYTE fsisig4[4];
};


/*************************************************************/
/*************************************************************/


// Structure of a dos directory entry.
struct direntry {
        BYTE        deName[8];      // filename, blank filled
```

```
#define SLOT_EMPTY      0x00            // slot has never been used
#define SLOT_E5         0x05            // the real value is 0xE5
#define SLOT_DELETED    0xE5            // file in this slot deleted
        BYTE        deExtension[3]; // extension, blank filled
        BYTE        deAttributes;   // file attributes
#define ATTR_NORMAL     0x00            // normal file
#define ATTR_READONLY   0x01            // file is readonly
#define ATTR_HIDDEN     0x02            // file is hidden
#define ATTR_SYSTEM     0x04            // file is a system file
#define ATTR_VOLUME     0x08            // entry is a volume label
#define ATTR_LONG_FILENAME  0x0F        // this is a long filename
    entry
#define ATTR_DIRECTORY  0x10            // entry is a directory name
#define ATTR_ARCHIVE    0x20            // file is new or modified
        BYTE        deLowerCase;    // NT VFAT lower case flags  (set
           to zero)
#define LCASE_BASE      0x08            // filename base in lower case
#define LCASE_EXT       0x10            // filename extension in lower
    case
        BYTE        deCHundredth;   // hundredth of seconds in CTime
        BYTE        deCTime[2];     // create time
        BYTE        deCDate[2];     // create date
        BYTE        deADate[2];     // access date
        WORD        deHighClust;    // high bytes of cluster number
        BYTE        deMTime[2];     // last update time
        BYTE        deMDate[2];     // last update date
        WORD        deStartCluster; // starting cluster of file
        DWORD       deFileSize;     // size of file in bytes
};

// number of directory entries in one sector
#define DIRENTRIES_PER_SECTOR   0x10

// Structure of a Win95 long name directory entry
struct winentry {
        BYTE        weCnt;          //
#define WIN_LAST        0x40
#define WIN_CNT         0x3f
        BYTE        wePart1[10];
        BYTE        weAttributes;
#define ATTR_WIN95      0x0f
        BYTE        weReserved1;
        BYTE        weChksum;
        BYTE        wePart2[12];
        WORD        weReserved2;
        BYTE        wePart3[4];
};

#define WIN_ENTRY_CHARS 13      // Number of chars per winentry

// Maximum filename length in Win95
// Note: Must be < sizeof(dirent.d_name)
```



```
#define WIN_MAXLEN      255

// This is the format of the contents of the deTime field in the
    direntry
// structure.
// We don't use bitfields because we don't know how compilers for
// arbitrary machines will lay them out.
#define DT_2SECONDS_MASK        0x1F    // seconds divided by 2
#define DT_2SECONDS_SHIFT       0
#define DT_MINUTES_MASK         0x7E0   // minutes
#define DT_MINUTES_SHIFT        5
#define DT_HOURS_MASK           0xF800  // hours
#define DT_HOURS_SHIFT          11

// This is the format of the contents of the deDate field in the
    direntry
// structure.
#define DD_DAY_MASK             0x1F    // day of month
#define DD_DAY_SHIFT            0
#define DD_MONTH_MASK           0x1E0   // month
#define DD_MONTH_SHIFT          5
#define DD_YEAR_MASK            0xFE00  // year - 1980
#define DD_YEAR_SHIFT           9

// Stuctures
struct FileInfoStruct
{
    unsigned long StartCluster;         //< file starting cluster for
        last file accessed
    unsigned long Size;                 //< file size for last file
        accessed
    unsigned char Attr;                 //< file attr for last file
        accessed
    unsigned short CreateTime;          //< file creation time for last
         file accessed
    unsigned short CreateDate;          //< file creation date for last
         file accessed
};

// Prototypes
unsigned char fatInit( unsigned char device);
unsigned int fatClusterSize(void);
unsigned char fatGetDirEntry(unsigned short entry);
unsigned char fatChangeDirectory(unsigned short entry);
void fatPrintDirEntry(void);
void fatDumpDirSlot(unsigned short entry);
struct FileInfoStruct* fatGetFileInfo(void);
unsigned long fatGetFilesize(void);
char* fatGetFilename(void);
char* fatGetDirname(void);
void fatLoadCluster(unsigned long cluster, unsigned char *buffer);
unsigned long fatNextCluster(unsigned long cluster);
```

```
#endif
```

Listing G.70: inc/fat16.h

```
/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef FAT16_H
#define FAT16_H

#include "fat16_config.h"

#include <stdint.h>

/**
 * \addtogroup fat16
 *
 * @{
 */
/**
 * \file
 * FAT16 header.
 *
 * \author Roland Riegel
 */

/**
 * \addtogroup fat16_file
 * @{
 */

/** The file is read-only. */
#define FAT16_ATTRIB_READONLY (1 << 0)
/** The file is hidden. */
#define FAT16_ATTRIB_HIDDEN (1 << 1)
/** The file is a system file. */
#define FAT16_ATTRIB_SYSTEM (1 << 2)
/** The file is empty and has the volume label as its name. */
#define FAT16_ATTRIB_VOLUME (1 << 3)
/** The file is a directory. */
#define FAT16_ATTRIB_DIR (1 << 4)
/** The file has to be archived. */
#define FAT16_ATTRIB_ARCHIVE (1 << 5)

/** The given offset is relative to the beginning of the file. */
#define FAT16_SEEK_SET 0
```



```
/** The given offset is relative to the current read/write position. */
#define FAT16_SEEK_CUR 1
/** The given offset is relative to the end of the file. */
#define FAT16_SEEK_END 2

/**
 * @}
 */

struct partition_struct;
struct fat16_fs_struct;
struct fat16_file_struct;
struct fat16_dir_struct;

/**
 * \ingroup fat16_file
 * Describes a directory entry.
 */
struct fat16_dir_entry_struct
{
    /** The file's name, truncated to 31 characters. */
    char long_name[32];
    /** The file's attributes. Mask of the FAT16_ATTRIB_* constants. */
    uint8_t attributes;
#if FAT16_DATETIME_SUPPORT
    /** Compressed representation of modification time. */
    uint16_t modification_time;
    /** Compressed representation of modification date. */
    uint16_t modification_date;
#endif
    /** The cluster in which the file's first byte resides. */
    uint16_t cluster;
    /** The file's size. */
    uint32_t file_size;
    /** The total disk offset of this directory entry. */
    uint32_t entry_offset;
};

struct fat16_fs_struct* fat16_open(struct partition_struct* partition);
void fat16_close(struct fat16_fs_struct* fs);

struct fat16_file_struct* fat16_open_file(struct fat16_fs_struct* fs,
    const struct fat16_dir_entry_struct* dir_entry);
void fat16_close_file(struct fat16_file_struct* fd);
int16_t fat16_read_file(struct fat16_file_struct* fd, uint8_t* buffer,
    uint16_t buffer_len);
int16_t fat16_write_file(struct fat16_file_struct* fd, const uint8_t*
    buffer, uint16_t buffer_len);
uint8_t fat16_seek_file(struct fat16_file_struct* fd, int32_t* offset,
    uint8_t whence);
uint8_t fat16_resize_file(struct fat16_file_struct* fd, uint32_t size);
```

```
struct fat16_dir_struct* fat16_open_dir(struct fat16_fs_struct* fs,
    const struct fat16_dir_entry_struct* dir_entry);
void fat16_close_dir(struct fat16_dir_struct* dd);
uint8_t fat16_read_dir(struct fat16_dir_struct* dd, struct
    fat16_dir_entry_struct* dir_entry);
uint8_t fat16_reset_dir(struct fat16_dir_struct* dd);

uint8_t fat16_create_file(struct fat16_dir_struct* parent, const char*
    file, struct fat16_dir_entry_struct* dir_entry);
uint8_t fat16_delete_file(struct fat16_fs_struct* fs, struct
    fat16_dir_entry_struct* dir_entry);
uint8_t fat16_create_dir(struct fat16_dir_struct* parent, const char*
    dir, struct fat16_dir_entry_struct* dir_entry);
#define fat16_delete_dir fat16_delete_file

void fat16_get_file_modification_date(const struct
    fat16_dir_entry_struct* dir_entry, uint16_t* year, uint8_t* month,
    uint8_t* day);
void fat16_get_file_modification_time(const struct
    fat16_dir_entry_struct* dir_entry, uint8_t* hour, uint8_t* min,
    uint8_t* sec);

uint8_t fat16_get_dir_entry_of_path(struct fat16_fs_struct* fs, const
    char* path, struct fat16_dir_entry_struct* dir_entry);

uint32_t fat16_get_fs_size(const struct fat16_fs_struct* fs);
uint32_t fat16_get_fs_free(const struct fat16_fs_struct* fs);

uint32_t fat16_get_file_size(const struct fat16_file_struct *fd);

/**
 * @}
 */

#endif
```

Listing G.71: inc/fat16_config.h

```
/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef FAT16_CONFIG_H
#define FAT16_CONFIG_G

/**
 * \addtogroup fat16
 *
```



```
 * @{
 */
/**
 * \file
 * FAT16 configuration.
 */

/**
 * \ingroup fat16_config
 * Controls FAT16 write support.
 *
 * Set to 1 to enable FAT16 write support, set to 0 to disable it.
 */
#define FAT16_WRITE_SUPPORT 1

/**
 * \ingroup fat16_config
 * Controls FAT16 date and time support.
 *
 * Set to 1 to enable FAT16 date and time stamping support.
 */
#define FAT16_DATETIME_SUPPORT 0

/**
 * \ingroup fat16_config
 * Determines the function used for retrieving current date and time.
 *
 * Define this to the function call which shall be used to retrieve
 * current date and time.
 *
 * \note Used only when FAT16_DATETIME_SUPPORT is 1.
 *
 * \param[out] year Pointer to a \c uint16_t which receives the current
     year.
 * \param[out] month Pointer to a \c uint8_t which receives the current
     month.
 * \param[out] day Pointer to a \c uint8_t which receives the current
    day.
 * \param[out] hour Pointer to a \c uint8_t which receives the current
    hour.
 * \param[out] min Pointer to a \c uint8_t which receives the current
    minute.
 * \param[out] sec Pointer to a \c uint8_t which receives the current
    sec.
 */
#define fat16_get_datetime(year, month, day, hour, min, sec) \
    get_datetime(year, month, day, hour, min, sec)
/* forward declaration for the above */
void get_datetime(uint16_t* year, uint8_t* month, uint8_t* day, uint8_t
   * hour, uint8_t* min, uint8_t* sec);

/**
```

```
 * \ingroup fat16_config
 * Maximum number of filesystem handles.
 */
#define FAT16_FS_COUNT 1

/**
 * \ingroup fat16_config
 * Maximum number of file handles.
 */
#define FAT16_FILE_COUNT 1

/**
 * \ingroup fat16_config
 * Maximum number of directory handles.
 */
#define FAT16_DIR_COUNT 2

/**
 * @}
 */

#endif
```

Listing G.72: inc/fatconf.h

```
/*! \file fatconf.h \brief FAT16/32 file system driver configuration.
    */
//
    ***************************************************************************

//
// File Name    : 'fatconf.h'
// Title        : FAT16/32 file system driver configuration
// Author       : Pascal Stang
// Date         : 4/19/2003
// Revised      : 4/19/2003
// Version      : 0.3
// Target MCU   : Atmel AVR Series
// Editor Tabs  : 4
//
// NOTE: This code is currently below version 1.0, and therefore is
    considered
// to be lacking in some functionality or documentation, or may not be
    fully
// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
    ***************************************************************************
```



```
#ifndef FATCONF_H
#define FATCONF_H

// debug on/off
//#define DEBUG_FAT

#define SECTOR_BUFFER1_ADDR          0x0600+0x0600
#define SECTOR_BUFFER1_SIZE          0x0200

#define LONGNAME_BUFFER_ADDR         0x0200+0x0600
#define LONGNAME_BUFFER_SIZE         0x0100

#define DIRNAME_BUFFER_ADDR          0x0300+0x0600
#define DIRNAME_BUFFER_SIZE          0x0100

#define FAT_CACHE_ADDR               0x0400+0x0600
#define FAT_CACHE_SIZE               0x0200

#endif
```

Listing G.73: inc/fifo.h

```
#ifndef FIFO_H
#define FIFO_H

#include <stdint.h>

#include "thread.h"

#define FIFO_INFINITE  ((uint16_t)0xffff)

struct fifo_s;

typedef void (*fifo_notify_t)(struct fifo_s *fifo, void *data);

typedef struct fifo_s {
  uint16_t size;
  uint16_t available;

  uint16_t read_pos;
  uint16_t write_pos;

  uint8_t readable;
  uint8_t writeable;

  fifo_notify_t readable_notify;
  void *readable_notify_data;
  thread_t *readable_thread;

  fifo_notify_t writeable_notify;
  void *writeable_notify_data;
  thread_t *writeable_thread;
```

```
  uint8_t data[];
} fifo_t;

fifo_t *fifo_alloc(uint16_t size);

void fifo_set_notify_readable(fifo_t *fifo, fifo_notify_t notify, void
   *data);
void fifo_set_notify_writeable(fifo_t *fifo, fifo_notify_t notify, void
    *data);

uint16_t fifo_write_interrupt(fifo_t *fifo, const void *data, uint16_t
   count);
uint16_t fifo_write(fifo_t *fifo, const void *data, uint16_t count,
                    uint16_t timeout);
uint16_t fifo_write_full(fifo_t *fifo, const void *data, uint16_t count
   ,
                         uint16_t timeout);

uint16_t fifo_read_interrupt(fifo_t *fifo, void *data, uint16_t count);
uint16_t fifo_read(fifo_t *fifo, void *data, uint16_t length, uint16_t
   timeout);
uint16_t fifo_read_full(fifo_t *fifo, void *data, uint16_t length,
   uint16_t timeout);

#endif /* FIFO_H */
```

Listing G.74: inc/fpga.h

```
#ifndef FPGA_H
#define FPGA_H

#include <stdint.h>


#define FPGA(x) (*((volatile uint8_t*)(x)))
#define FPGA16(x) (*((volatile uint16_t*)(x)))
#define FPGA_PTR(x) ((volatile uint8_t*)(x))

/* Interrupt flags. */
#define FPGA_INTR_KEYBOARD 0
#define FPGA_INTR_DES 1
#define FPGA_INTR_BUTTONS 2

/* LED interface. */
#define FPGA_LEDS FPGA(0xe000)

/* FPGA interrupt status register. */
#define FPGA_ISR FPGA(0xe001)

/* Buttons. */
#define FPGA_BUTTONS_L FPGA(0xe002)
#define FPGA_BUTTONS_H FPGA(0xe003)
#define FPGA_BUTTONS FPGA16(0xe002)
```



```
/* Keyboard - L is the two top rows, H is the two bottom rows. */
#define FPGA_KEYBOARD_L FPGA(0xe004)
#define FPGA_KEYBOARD_H FPGA(0xe005)
#define FPGA_KEYBOARD FPGA16(0xe004)

/* Codec receive channel test. */
#define FPGA_CODEC_RX_L FPGA(0xe006)
#define FPGA_CODEC_RX_H FPGA(0xe007)
#define FPGA_CODEC_RX FPGA16(0xe006)


/* FPGA SRAM. */
#define FPGA_SRAM(offset) FPGA(0x1100 + (offset))

/* DES interface. To be completed. */
#define FPGA_CRYPT_KEY  FPGA_PTR(0xEC00)
#define FPGA_CRYPT_DATA_IN  FPGA_PTR(0xED00)
#define FPGA_CRYPT_DATA_OUT  FPGA_PTR(0xEE00)
#define FPGA_CRYPT_STATUS FPGA(0xEF00)
#define FPGA_CRYPT_ENCRYPT  FPGA(0xEF01)
#define FPGA_CRYPT_DECRYPT FPGA(0xEF02)
#define FPGA_CRYPT_USE_CBC FPGA(0xEF03)
#define FPGA_CRYPT_RESET_CBC_ENCRYPTION FPGA(0xEF04)
#define FPGA_CRYPT_RESET_CBC_DECRYPTION FPGA(0xEF05)


#endif /* FPGA_H */
```

Listing G.75: inc/fs.h

```
#ifndef FS_H
#define FS_H

#include "fat16.h"

void fs_init(void);
struct fat16_file_struct *fs_open_file(const char *filename);


#endif
```

Listing G.76: inc/global.h

```
/*! \file global.h \brief AVRlib project global include. */
//
   ***************************************************************************

//
// File Name    : 'global.h'
// Title        : AVRlib project global include
// Author       : Pascal Stang - Copyright (C) 2001-2002
// Created      : 7/12/2001
```

```
// Revised      : 9/30/2002
// Version      : 1.1
// Target MCU   : Atmel AVR series
// Editor Tabs  : 4
//
//  Description : This include file is designed to contain items useful
    to all
//                  code files and projects.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
   ***************************************************************************


#ifndef GLOBAL_H
#define GLOBAL_H

// global AVRLIB defines
#include "avrlibdefs.h"
// global AVRLIB types definitions
#include "avrlibtypes.h"

// project/system dependent defines

// CPU clock speed
//#define F_CPU        16000000                    // 16MHz processor
//#define F_CPU        14745000                    // 14.745MHz
   processor
//#define F_CPU        8000000                     // 8MHz processor
#define F_CPU        7372800                   // 7.37MHz processor
//#define F_CPU        4000000                     // 4MHz processor
//#define F_CPU        3686400                     // 3.69MHz
   processor
#define CYCLES_PER_US ((F_CPU+500000)/1000000)  // cpu cycles per
   microsecond

#endif
```

Listing G.77: inc/jtag.h

```
#ifndef JTAG_H
#define JTAG_H

void jtag_disable(void);

#endif /* JTAG_H */
```

Listing G.78: inc/keyboard.h

```
#ifndef KEYBOARD_H
#define KEYBOARD_H
```



```
typedef void (*keyboard_handler_t)(char);

void keyboard_init(void);
void keyboard_scan_forever(void *keyhandler);
int8_t keyboard_scan(void);
char keyboard_translate(int8_t keycode);

#endif
```

Listing G.79: inc/keyboard_int.h

```
#ifndef KEYBOARD_INT_H
#define KEYBOARD_INT_H

typedef void (*keyboard_int_handler_t)(char);

void keyboard_int_init(keyboard_int_handler_t handler);

#endif
```

Listing G.80: inc/lcd.h

```
#ifndef LCD_H
#define LCD_H

#define LCD_MESSAGE   ((char)0x08)

void lcd_init(void);
uint8_t lcd_get_pos(void);
void lcd_goto(uint8_t position);
void lcd_scrolldown(void);
void lcd_putchar(char character);
void lcd_set_data(const char *data, uint8_t cursor_pos);
void lcd_putstring(const char *s);

#endif
```

Listing G.81: inc/memory.h

```
#ifndef MEMORY_H
#define MEMORY_H

void init_xmem(void);

#endif /* MEMORY_H */
```

Listing G.82: inc/menu.h

```
#ifndef MENU_H
#define MENU_H

#define MAX_MENU_ITEM_LEN 10
```

```
#define MAX_MENU_TITLE_LEN 26

#define DISPLAY_WIDTH 40
#define MENU_LINE_WIDTH 38
#define MENU_CURSOR_MAX 3*MENU_LINE_WIDTH-1

typedef void (*menu_cb_t)(uint8_t);

typedef struct {
  const char *title;
  menu_cb_t callback;
} menu_item_t;

typedef struct menu_t {
  const char *title;

  // menu items for the 9 numeric keys:
  menu_item_t items[9];
  // menu items for the keys A, B, C, D:
  menu_item_t abcd_items[4];

  // callback function to be called whenever a key is pressed while in
  // this menu:
  menu_cb_t key_cb;

  // tells if the strings in this menu are in program memory (1) or
  // normal memory (0):
  uint8_t progmem_strings;
} menu_t;


void menu_init(void);
menu_t *menu_add(const char *title, uint8_t progmem_strings);
void menu_pop(void);
void menu_pop_update(uint8_t update_display);
void menu_set_item(menu_t *menu, uint8_t i, const char *title,
    menu_cb_t callback);
void menu_add_item(menu_t *menu, const char *title, menu_cb_t callback)
    ;
void menu_set_abcd_item(menu_t *menu, char item, const char *title,
    menu_cb_t callback);
void menu_setup_done(menu_t *menu);
void menu_update_display(void);

void menu_setchar(uint8_t pos, char c);
void menu_setstring(uint8_t pos, const char *str);
void menu_set_cursor(uint8_t pos);
void menu_get_text(uint8_t *buf, uint8_t len);

void menu_key_pressed(char key);

void menu_set_screen_buffer(char *buf);
```



```
void menu_set_unread_sms(uint8_t status);

#endif
```

Listing G.83: inc/message.h

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <stdint.h>
#include "fifo.h"

extern fifo_t *msg_to_ui, *msg_to_modem;

typedef enum {
  MSG_KEY_PRESS,
  MSG_READY_FOR_PIN, MSG_PIN, MSG_PIN_RESULT,
  MSG_RECV_CALL, MSG_RECV_SMS,
  MSG_NO_UNREAD_SMS,
  MSG_SIGNAL_QUALITY
} message_type_t;

typedef struct {
  message_type_t type;
  union {
    char key;
    char pin_code[5];
    uint8_t pin_code_ok;
    uint8_t signal_quality;
  } data;
} message_t;

void message_init(void);
void message_put(fifo_t *fifo, message_t *m);
void message_put_empty(fifo_t *fifo, message_type_t mtyp);
message_t *message_get(fifo_t *fifo);
message_t *message_get_timeout(fifo_t *fifo, uint16_t timeout);
message_t *message_wait_for(fifo_t *fifo, message_type_t mtyp);

#endif
```

Listing G.84: inc/mmc.h

```
/*! \file mmc.h \brief MultiMedia and SD Flash Card Interface. */
//
   ***************************************************************************

//
// File Name    : 'mmc.h'
// Title        : MultiMedia and SD Flash Card Interface
// Author       : Pascal Stang - Copyright (C) 2004
// Created      : 2004.09.22
```

```
// Revised      : 2004.09.22
// Version      : 0.1
// Target MCU   : Atmel AVR Series
// Editor Tabs  : 4
//
/// \ingroup driver_hw
/// \defgroup mmc MultiMedia and SD Flash Card Interface (mmc.c)
/// \code #include "mmc.h" \endcode
/// \par Description
///     This library offers some simple functions which can be used
///     to read and write data on a MultiMedia or SecureDigital (SD)
   Flash
///     Card.  Although MM and SD Cards are designed to operate with
   their own
///     special bus wiring and protocols, both types of cards also
   provide a
///     simple SPI-like interface mode which is exceptionally useful
   when
///     attempting to use the cards in embedded systems.
///
/// \par Wiring
///     To work with this library, the card must be wired to the SPI
   port of
///     the Atmel microcontroller as described below.  Typical cards
   can
///     operate at up to 25MHz maximum SPI clock rate (thus faster than
    most
///     AVR's maximum SPI clock rate).
/// <pre>
///       ------------------
///      /  1 2 3 4 5 6 78 |    <- view of MMC/SD card looking at
   contacts
///     / 9                |    Pins 8 and 9 are present only on SD
   cards
///     |    MMC/SD Card   |
///     |                  |
///     /\/\/\/\/\/\/\/\/\/
///     1 - CS   (chip select)          - wire to any available I/O pin
   (*)
///     2 - DIN  (data in, card<-host)  - wire to SPI MOSI pin
///     3 - VSS  (ground)               - wire to ground
///     4 - VDD  (power, 3.3V only?)    - wire to power (MIGHT BE 3.3V
   ONLY!)
///     5 - SCLK (data clock)           - wire to SPI SCK pin
///     6 - VSS  (ground)               - wire to ground
///     7 - DOUT (data out, card->host) - wire to SPI MISO pin
///
///     (*) you must define this chip select I/O pin in mmcconf.h
/// </pre>
/// \note This code is currently below version 1.0, and therefore is
   considered
```



```
/// to be lacking in some functionality or documentation, or may not be
    fully
/// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
   ***************************************************************************

//@{

#ifndef MMC_H
#define MMC_H

#include "global.h"

// constants/macros/typdefs
// MMC commands (taken from sandisk MMC reference)
#define MMC_GO_IDLE_STATE           0       ///< initialize card to SPI
   -type access
#define MMC_SEND_OP_COND            1       ///< set card operational
   mode
#define MMC_SEND_CSD                9       ///< get card's CSD
#define MMC_SEND_CID                10      ///< get card's CID
#define MMC_SEND_STATUS             13
#define MMC_SET_BLOCKLEN            16      ///< Set number of bytes to
    transfer per block
#define MMC_READ_SINGLE_BLOCK       17      ///< read a block
#define MMC_WRITE_BLOCK             24      ///< write a block
#define MMC_PROGRAM_CSD             27
#define MMC_SET_WRITE_PROT          28
#define MMC_CLR_WRITE_PROT          29
#define MMC_SEND_WRITE_PROT         30
#define MMC_TAG_SECTOR_START        32
#define MMC_TAG_SECTOR_END          33
#define MMC_UNTAG_SECTOR            34
#define MMC_TAG_ERASE_GROUP_START   35      ///< Sets beginning of
   erase group (mass erase)
#define MMC_TAG_ERARE_GROUP_END     36      ///< Sets end of erase
   group (mass erase)
#define MMC_UNTAG_ERASE_GROUP       37      ///< Untag (unset) erase
   group (mass erase)
#define MMC_ERASE                   38      ///< Perform block/mass
   erase
#define MMC_CRC_ON_OFF              59      ///< Turns CRC check on/off
// R1 Response bit-defines
#define MMC_R1_BUSY                 0x80    ///< R1 response: bit
   indicates card is busy
#define MMC_R1_PARAMETER            0x40
#define MMC_R1_ADDRESS              0x20
#define MMC_R1_ERASE_SEQ            0x10
```

```
#define MMC_R1_COM_CRC              0x08
#define MMC_R1_ILLEGAL_COM          0x04
#define MMC_R1_ERASE_RESET          0x02
#define MMC_R1_IDLE_STATE           0x01
// Data Start tokens
#define MMC_STARTBLOCK_READ         0xFE    ///< when received from
   card, indicates that a block of data will follow
#define MMC_STARTBLOCK_WRITE        0xFE    ///< when sent to card,
   indicates that a block of data will follow
#define MMC_STARTBLOCK_MWRITE       0xFC
// Data Stop tokens
#define MMC_STOPTRAN_WRITE          0xFD
// Data Error Token values
#define MMC_DE_MASK                 0x1F
#define MMC_DE_ERROR                0x01
#define MMC_DE_CC_ERROR             0x02
#define MMC_DE_ECC_FAIL             0x04
#define MMC_DE_OUT_OF_RANGE         0x04
#define MMC_DE_CARD_LOCKED          0x04
// Data Response Token values
#define MMC_DR_MASK                 0x1F
#define MMC_DR_ACCEPT               0x05
#define MMC_DR_REJECT_CRC           0x0B
#define MMC_DR_REJECT_WRITE_ERROR   0x0D

// functions

//! Initialize AVR<->MMC hardware interface.
/// Prepares hardware for MMC access.
void mmcInit(void);

//! Initialize the card and prepare it for use.
/// Returns zero if successful.
u08 mmcReset(void);

//! Send card an MMC command.
/// Returns R1 result code.
u08 mmcSendCommand(u08 cmd, u32 arg);

//! Read 512-byte sector from card to buffer
/// Returns zero if successful.
u08 mmcRead(u32 sector, u08* buffer);

//! Write 512-byte sector from buffer to card
/// Returns zero if successful.
u08 mmcWrite(u32 sector, u08* buffer);

//! Internal command function.
/// Issues a generic MMC command as specified by cmd and arg.
u08 mmcCommand(u08 cmd, u32 arg);

#endif
```



```
//@}
```

Listing G.85: inc/mmcconf.h

```
/*! \file mmcconf.h \brief MultiMedia and SD Flash Card Interface
   Configuration. */
//
   *****************************************************************************

//
// File Name    : 'mmc.h'
// Title        : MultiMedia and SD Flash Card Interface Configuration
// Author       : Pascal Stang - Copyright (C) 2004
// Created      : 2004.09.22
// Revised      : 2004.09.22
// Version      : 0.1
// Target MCU   : Atmel AVR Series
// Editor Tabs  : 4
//
// NOTE: This code is currently below version 1.0, and therefore is
   considered
// to be lacking in some functionality or documentation, or may not be
   fully
// tested.  Nonetheless, you can expect most functions to work.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
   *****************************************************************************


#ifndef MMCCONF_H
#define MMCCONF_H

// define to enable debugging print statements
//#define MMC_DEBUG

// MMC card chip select pin defines
#define MMC_CS_PORT          PORTB
#define MMC_CS_DDR           DDRB
#define MMC_CS_PIN           0

#endif
```

Listing G.86: inc/modem.h

```
#ifndef MODEM_H
#define MODEM_H

#define SMS_NUMBER_LEN 12
#define SMS_TEXT_LEN 161
#define SMS_DATE_LEN 21
```

```
void modem_init(void);

typedef struct {
  char number[SMS_NUMBER_LEN];
  char text[SMS_TEXT_LEN];
  char date[SMS_DATE_LEN];
} sms_t;

uint8_t modem_enter_pin(const char *pin);

uint8_t modem_send_sms(sms_t *sms);
int8_t modem_get_sms_count(void);
uint8_t modem_read_sms(sms_t *sms, uint8_t index, uint8_t mark_unread);
uint8_t modem_signal_quality(void);
void modem_delete_sms(uint8_t index);
void modem_delete_all_sms(void);

void modem_thread(void *ignored);

#endif /* MODEM_H */
```

Listing G.87: inc/output.h

```
#ifndef OUTPUT_H
#define OUTPUT_H

void output_init(void);

#endif /* OUTPUT_H */
```

Listing G.88: inc/partition.h

```
/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef PARTITION_H
#define PARTITION_H

#include <stdint.h>

/**
 * \addtogroup partition
 *
 * @{
 */
/**
```



```
 * \file
 * Partition table header.
 *
 * \author Roland Riegel
 */

/**
 * The partition table entry is not used.
 */
#define PARTITION_TYPE_FREE 0x00
/**
 * The partition contains a FAT12 filesystem.
 */
#define PARTITION_TYPE_FAT12 0x01
/**
 * The partition contains a FAT16 filesystem with 32MB maximum.
 */
#define PARTITION_TYPE_FAT16_32MB 0x04
/**
 * The partition is an extended partition with its own partition table.
 */
#define PARTITION_TYPE_EXTENDED 0x05
/**
 * The partition contains a FAT16 filesystem.
 */
#define PARTITION_TYPE_FAT16 0x06
/**
 * The partition contains a FAT32 filesystem.
 */
#define PARTITION_TYPE_FAT32 0x0b
/**
 * The partition contains a FAT32 filesystem with LBA.
 */
#define PARTITION_TYPE_FAT32_LBA 0x0c
/**
 * The partition contains a FAT16 filesystem with LBA.
 */
#define PARTITION_TYPE_FAT16_LBA 0x0e
/**
 * The partition is an extended partition with LBA.
 */
#define PARTITION_TYPE_EXTENDED_LBA 0x0f
/**
 * The partition has an unknown type.
 */
#define PARTITION_TYPE_UNKNOWN 0xff

/**
 * A function pointer used to read from the partition.
 *
 * \param[in] offset The offset on the device where to start reading.
 * \param[out] buffer The buffer into which to place the data.
```

```
 * \param[in] length The count of bytes to read.
 */
typedef uint8_t (*device_read_t)(uint32_t offset, uint8_t* buffer,
    uint16_t length);
/**
 * A function pointer passed to a \c device_read_interval_t.
 *
 * \param[in] buffer The buffer which contains the data just read.
 * \param[in] offset The offset from which the data in \c buffer was
     read.
 * \param[in] p An opaque pointer.
 * \see device_read_interval_t
 */
typedef uint8_t (*device_read_callback_t)(uint8_t* buffer, uint32_t
    offset, void* p);
/**
 * A function pointer used to continuously read units of \c interval
     bytes
 * and call a callback function.
 *
 * This function starts reading at the specified offset. Every \c
     interval bytes,
 * it calls the callback function with the associated data buffer.
 *
 * By returning zero, the callback may stop reading.
 *
 * \param[in] offset Offset from which to start reading.
 * \param[in] buffer Pointer to a buffer which is at least interval
     bytes in size.
 * \param[in] interval Number of bytes to read before calling the
     callback function.
 * \param[in] length Number of bytes to read altogether.
 * \param[in] callback The function to call every interval bytes.
 * \param[in] p An opaque pointer directly passed to the callback
     function.
 * \returns 0 on failure, 1 on success
 * \see device_read_t
 */
typedef uint8_t (*device_read_interval_t)(uint32_t offset, uint8_t*
    buffer, uint16_t interval, uint16_t length, device_read_callback_t
    callback, void* p);
/**
 * A function pointer used to write to the partition.
 *
 * \param[in] offset The offset on the device where to start writing.
 * \param[in] buffer The buffer which to write.
 * \param[in] length The count of bytes to write.
 */
typedef uint8_t (*device_write_t)(uint32_t offset, const uint8_t*
    buffer, uint16_t length);
/**
 * A function pointer passed to a \c device_write_interval_t.
```



```
 *
 * \param[in] buffer The buffer which receives the data to write.
 * \param[in] offset The offset to which the data in \c buffer will be
     written.
 * \param[in] p An opaque pointer.
 * \returns The number of bytes put into \c buffer
 * \see device_write_interval_t
 */
typedef uint16_t (*device_write_callback_t)(uint8_t* buffer, uint32_t
    offset, void* p);
/**
 * A function pointer used to continuously write a data stream obtained
     from
 * a callback function.
 *
 * This function starts writing at the specified offset. To obtain the
 * next bytes to write, it calls the callback function. The callback
     fills the
 * provided data buffer and returns the number of bytes it has put into
     the buffer.
 *
 * By returning zero, the callback may stop writing.
 *
 * \param[in] offset Offset where to start writing.
 * \param[in] buffer Pointer to a buffer which is used for the callback
     function.
 * \param[in] length Number of bytes to write in total. May be zero for
     endless writes.
 * \param[in] callback The function used to obtain the bytes to write.
 * \param[in] p An opaque pointer directly passed to the callback
     function.
 * \returns 0 on failure, 1 on success
 * \see device_write_t
 */
typedef uint8_t (*device_write_interval_t)(uint32_t offset, uint8_t*
    buffer, uint16_t length, device_write_callback_t callback, void* p);

/**
 * Describes a partition.
 */
struct partition_struct
{
    /**
     * The function which reads data from the partition.
     *
     * \note The offset given to this function is relative to the whole
         disk,
     *       not to the start of the partition.
     */
    device_read_t device_read;
    /**
```

```
     * The function which repeatedly reads a constant amount of data
        from the partition.
     *
     * \note The offset given to this function is relative to the whole
        disk,
     *       not to the start of the partition.
     */
    device_read_interval_t device_read_interval;
    /**
     * The function which writes data to the partition.
     *
     * \note The offset given to this function is relative to the whole
        disk,
     *       not to the start of the partition.
     */
    device_write_t device_write;
    /**
     * The function which repeatedly writes data to the partition.
     *
     * \note The offset given to this function is relative to the whole
        disk,
     *       not to the start of the partition.
     */
    device_write_interval_t device_write_interval;

    /**
     * The type of the partition.
     *
     * Compare this value to the PARTITION_TYPE_* constants.
     */
    uint8_t type;
    /**
     * The offset in bytes on the disk where this partition starts.
     */
    uint32_t offset;
    /**
     * The length in bytes of this partition.
     */
    uint32_t length;
};

struct partition_struct* partition_open(device_read_t device_read,
   device_read_interval_t device_read_interval, device_write_t
   device_write, device_write_interval_t device_write_interval, int8_t
   index);
uint8_t partition_close(struct partition_struct* partition);

/**
 * @}
 */

#endif
```


Listing G.89: inc/partition_config.h

```
/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef PARTITION_CONFIG_H
#define PARTITION_CONFIG_G

/**
 * \addtogroup partition
 *
 * @{
 */
/**
 * \file
 * Partition configuration.
 */

/**
 * \ingroup partition_config
 * Maximum number of partition handles.
 */
#define PARTITION_COUNT 1

/**
 * @}
 */

#endif
```

Listing G.90: inc/phone_book.h

```
#ifndef PHONE_BOOK_H
#define PHONE_BOOK_H

#include <stdint.h>

typedef struct {
  uint8_t name[20];
  uint8_t number[12];
  uint8_t key[24];
  uint8_t unused[8];
} phone_book_entry_t;

typedef void (*phone_book_callback_t)(phone_book_entry_t *entry);

uint8_t phone_book_open(void);
```

```
uint8_t phone_book_read(phone_book_entry_t *entry, uint8_t entry_index)
    ;
void phone_book_close(void);

void phone_book_menu(char *title, phone_book_callback_t select_handler)
    ;

phone_book_entry_t *phone_book_search(char *number);

#endif
```

Listing G.91: inc/scheduler.h

```
#ifndef SCHEDULER_H
#define SCHEDULER_H

#include "thread.h"

void scheduler_init(void);
void scheduler_add_thread(thread_t *thread);
thread_t *scheduler_get_next_thread(void);


#endif /* SCHEDULER_H */
```

Listing G.92: inc/sd-reader_config.h

```

/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef SD_READER_CONFIG_H
#define SD_READER_CONFIG_H

/**
 * \addtogroup config Sd-reader configuration
 *
 * @{
 */

/**
 * \file
 * Common sd-reader configuration used by all modules.
 *
 * \note This file contains only configuration items relevant to
 * all sd-reader implementation files. For module specific
    configuration
 * options, please see the files fat16_config.h, partition_config.h
```



```
 * and sd_raw_config.h.
 */

/**
 * Controls allocation of memory.
 *
 * Set to 1 to use malloc()/free() for allocation of structures
 * like file and directory handles, set to 0 to use pre-allocated
 * fixed-size handle arrays.
 */
#define USE_DYNAMIC_MEMORY 0

/**
 * @}
 */

#endif
```

Listing G.93: inc/sd_raw.h

```

/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef SD_RAW_H
#define SD_RAW_H

#include <stdint.h>
#include "sd_raw_config.h"

/**
 * \addtogroup sd_raw
 *
 * @{
 */
/**
 * \file
 * MMC/SD raw access header.
 *
 * \author Roland Riegel
 */

/**
 * The card's layout is harddisk-like, which means it contains
 * a master boot record with a partition table.
 */
#define SD_RAW_FORMAT_HARDDISK 0
/**
```

```
 * The card contains a single filesystem and no partition table.
 */
#define SD_RAW_FORMAT_SUPERFLOPPY 1
/**
 * The card's layout follows the Universal File Format.
 */
#define SD_RAW_FORMAT_UNIVERSAL 2
/**
 * The card's layout is unknown.
 */
#define SD_RAW_FORMAT_UNKNOWN 3

/**
 * This struct is used by sd_raw_get_info() to return
 * manufacturing and status information of the card.
 */
struct sd_raw_info
{
    /**
     * A manufacturer code globally assigned by the SD card
         organization.
     */
    uint8_t manufacturer;
    /**
     * A string describing the card's OEM or content, globally assigned
         by the SD card organization.
     */
    uint8_t oem[3];
    /**
     * A product name.
     */
    uint8_t product[6];
    /**
     * The card's revision, coded in packed BCD.
     *
     * For example, the revision value \c 0x32 means "3.2".
     */
    uint8_t revision;
    /**
     * A serial number assigned by the manufacturer.
     */
    uint32_t serial;
    /**
     * The year of manufacturing.
     *
     * A value of zero means year 2000.
     */
    uint8_t manufacturing_year;
    /**
     * The month of manufacturing.
     */
    uint8_t manufacturing_month;
```



```
    /**
     * The card's total capacity in bytes.
     */
    uint32_t capacity;
    /**
     * Defines wether the card's content is original or copied.
     *
     * A value of \c 0 means original, \c 1 means copied.
     */
    uint8_t flag_copy;
    /**
     * Defines wether the card's content is write-protected.
     *
     * \note This is an internal flag and does not represent the
     *       state of the card's mechanical write-protect switch.
     */
    uint8_t flag_write_protect;
    /**
     * Defines wether the card's content is temporarily write-protected
         .
     *
     * \note This is an internal flag and does not represent the
     *       state of the card's mechanical write-protect switch.
     */
    uint8_t flag_write_protect_temp;
    /**
     * The card's data layout.
     *
     * See the \c SD_RAW_FORMAT_* constants for details.
     *
     * \note This value is not guaranteed to match reality.
     */
    uint8_t format;
};

typedef uint8_t (*sd_raw_read_interval_handler_t)(uint8_t* buffer,
    uint32_t offset, void* p);
typedef uint16_t (*sd_raw_write_interval_handler_t)(uint8_t* buffer,
    uint32_t offset, void* p);

uint8_t sd_raw_init();
uint8_t sd_raw_available();
uint8_t sd_raw_locked();

uint8_t sd_raw_read(uint32_t offset, uint8_t* buffer, uint16_t length);
uint8_t sd_raw_read_interval(uint32_t offset, uint8_t* buffer, uint16_t
     interval, uint16_t length, sd_raw_read_interval_handler_t callback,
     void* p);
uint8_t sd_raw_write(uint32_t offset, const uint8_t* buffer, uint16_t
    length);
uint8_t sd_raw_write_interval(uint32_t offset, uint8_t* buffer,
    uint16_t length, sd_raw_write_interval_handler_t callback, void* p);
```

```
uint8_t sd_raw_sync();

uint8_t sd_raw_get_info(struct sd_raw_info* info);

/**
 * @}
 */

#endif
```

Listing G.94: inc/sd_raw_config.h

```
/*
 * Copyright (c) 2006-2007 by Roland Riegel <feedback@roland-riegel.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#ifndef SD_RAW_CONFIG_H
#define SD_RAW_CONFIG_H

/**
 * \addtogroup sd_raw
 *
 * @{
 */
/**
 * \file
 * MMC/SD support configuration.
 */

/**
 * \ingroup sd_raw_config
 * Controls MMC/SD write support.
 *
 * Set to 1 to enable MMC/SD write support, set to 0 to disable it.
 */
#define SD_RAW_WRITE_SUPPORT 1

/**
 * \ingroup sd_raw_config
 * Controls MMC/SD write buffering.
 *
 * Set to 1 to buffer write accesses, set to 0 to disable it.
 *
 * \note This option has no effect when SD_RAW_WRITE_SUPPORT is 0.
 */
#define SD_RAW_WRITE_BUFFERING 1

/**
```



```
 * \ingroup sd_raw_config
 * Controls MMC/SD access buffering.
 *
 * Set to 1 to save static RAM, but be aware that you will
 * lose performance.
 *
 * \note When SD_RAW_WRITE_SUPPORT is 1, SD_RAW_SAVE_RAM will
 *       be reset to 0.
 */
#define SD_RAW_SAVE_RAM 1

/* defines for customisation of sd/mmc port access */
#if defined(__AVR_ATmega8__) || \
    defined(__AVR_ATmega48__) || \
    defined(__AVR_ATmega88__) || \
    defined(__AVR_ATmega168__)
    #define configure_pin_mosi() DDRB |= (1 << DDB3)
    #define configure_pin_sck() DDRB |= (1 << DDB5)
    #define configure_pin_ss() DDRB |= (1 << DDB2)
    #define configure_pin_miso() DDRB &= ~(1 << DDB4)

    #define select_card() PORTB &= ~(1 << PB2)
    #define unselect_card() PORTB |= (1 << PB2)
#elif defined(__AVR_ATmega16__) || \
      defined(__AVR_ATmega32__)
    #define configure_pin_mosi() DDRB |= (1 << DDB5)
    #define configure_pin_sck() DDRB |= (1 << DDB7)
    #define configure_pin_ss() DDRB |= (1 << DDB4)
    #define configure_pin_miso() DDRB &= ~(1 << DDB6)

    #define select_card() PORTB &= ~(1 << PB4)
    #define unselect_card() PORTB |= (1 << PB4)
#elif defined(__AVR_ATmega64__) || \
      defined(__AVR_ATmega128__) || \
      defined(__AVR_ATmega169__)
    #define configure_pin_mosi() DDRB |= (1 << DDB2)
    #define configure_pin_sck() DDRB |= (1 << DDB1)
    #define configure_pin_ss() DDRB |= (1 << DDB0)
    #define configure_pin_miso() DDRB &= ~(1 << DDB3)

    #define select_card() PORTB &= ~(1 << PB0)
    #define unselect_card() PORTB |= (1 << PB0)
#else
    #error "no sd/mmc pin mapping available!"
#endif

#define configure_pin_available() DDRC &= ~(1 << DDC4)
#define configure_pin_locked() DDRC &= ~(1 << DDC5)

#define get_pin_available() ((PINC >> PC4) & 0x01)
#define get_pin_locked() ((PINC >> PC5) & 0x01)
```

```
/**
 * @}
 */

/* configuration checks */
#if SD_RAW_WRITE_SUPPORT
#undef SD_RAW_SAVE_RAM
#define SD_RAW_SAVE_RAM 0
#else
#undef SD_RAW_WRITE_BUFFERING
#define SD_RAW_WRITE_BUFFERING 0
#endif

#endif
```

Listing G.95: inc/sleep.h

```
#ifndef SLEEP_H
#define SLEEP_H

#include <stdint.h>

void sleep(uint16_t timeout);

#endif /* SLEEP_H */
```

Listing G.96: inc/sms.h

```
#ifndef SMS_H
#define SMS_H

void sms_init(void);
void sms_menu(uint8_t ii);

#endif
```

Listing G.97: inc/spi.h

```
/*! \file spi.h \brief SPI interface driver. */
//
   ***************************************************************************

//
// File Name    : 'spi.h'
// Title        : SPI interface driver
// Author       : Pascal Stang - Copyright (C) 2000-2002
// Created      : 11/22/2000
// Revised      : 06/06/2002
// Version      : 0.6
// Target MCU   : Atmel AVR series
// Editor Tabs  : 4
//
```



```
// NOTE: This code is currently below version 1.0, and therefore is
   considered
// to be lacking in some functionality or documentation, or may not be
   fully
// tested.  Nonetheless, you can expect most functions to work.
//
/// \ingroup driver_avr
/// \defgroup spi SPI (Serial Peripheral Interface) Function Library (
   spi.c)
/// \code #include "spi.h" \endcode
/// \par Overview
///     Provides basic byte and word transmitting and receiving via the
    AVR
/// SPI interface.  Due to the nature of SPI, every SPI communication
   operation
/// is both a transmit and simultaneous receive.
///
/// \note Currently, only MASTER mode is supported.
//
// This code is distributed under the GNU Public License
//      which can be found at http://www.gnu.org/licenses/gpl.txt
//
//
   ***************************************************************************


#ifndef SPI_H
#define SPI_H

#include "global.h"

// function prototypes

// SPI interface initializer
void spiInit(void);

// spiSendByte(u08 data) waits until the SPI interface is ready
// and then sends a single byte over the SPI port.  This command
// does not receive anything.
void spiSendByte(u08 data);

// spiTransferByte(u08 data) waits until the SPI interface is ready
// and then sends a single byte over the SPI port.  The function also
// returns the byte that was received during transmission.
u08 spiTransferByte(u08 data);

// spiTransferWord(u08 data) works just like spiTransferByte but
// operates on a whole word (16-bits of data).
u16 spiTransferWord(u16 data);

#endif
```

Listing G.98: inc/tests.h

```
#ifndef TESTS_H
#define TESTS_H

void tests_modem(void);
void tests_sleep(void);
void tests_fifo(void);

#endif /* TESTS_H */
```

Listing G.99: inc/thread.h

```
#ifndef THREAD_H
#define THREAD_H

#include <stdint.h>

#define THREAD_STACK_SIZE 256
#define THREAD_STACK_BUFFER_SIZE 5

typedef void (*thread_func_t)(void *data);

typedef struct {
  uint8_t state;
  void *sp;
  uint8_t stack_buffer[THREAD_STACK_BUFFER_SIZE];
  uint8_t stack[THREAD_STACK_SIZE];
} thread_t;

extern thread_t *thread_current;

void thread_init(void);
void thread_setup(thread_t *thread, thread_func_t function, void *data)
    ;
void thread_begin(void);
void thread_yield(void);
void thread_wake(thread_t *thread);
void thread_wait(void);
void thread_verify_stack(void);

#endif /* THREAD_H */
```

Listing G.100: inc/thread_switch.h

```
#ifndef THREAD_SWITCH_H
#define THREAD_SWITCH_H

void thread_switch(void *new_sp, void **old_sp);

#endif /* THREAD_SWITCH_H */
```

Listing G.101: inc/time.h


```
#ifndef TIME_H
#define TIME_H

#include <stdint.h>

#include "wait_queue.h"

#define TIME_INFINITE  ((uint32_t)0xffffffff)

typedef struct time_wake_element_s {
  struct time_wake_element_s *next;

  thread_t *thread;
  uint32_t wake_when;
} time_wake_element_t;

void time_init(void);
uint32_t time_get(void);
void time_add_wake(time_wake_element_t *e, uint32_t wake_when);
void time_remove_wake(time_wake_element_t *e);

#endif /* TIME_H */
```

Listing G.102: inc/timeout.h

```
#ifndef TIMEOUT_H
#define TIMEOUT_H

typedef void (*timeout_cb_t)(void *data);

typedef struct {
  timeout_cb_t fun;
  uint32_t time;
  void *data;
} timeout_t;

void timeout_init(thread_t *thread);
void timeout_set(timeout_t *t, timeout_cb_t fun, uint32_t time, void *
    data);
void timeout_cancel(timeout_t *t);

#endif
```

Listing G.103: inc/ui.h

```
#ifndef UI_H
#define UI_H

void ui_init(void);

#endif
```

Listing G.104: inc/usart.h

```
#ifndef USART_H
#define USART_H

#include <stdint.h>

void usart_init(void);

void usart_setup(uint8_t port);

void usart_transmit(uint8_t port, const uint8_t *data, uint16_t length)
    ;
uint16_t usart_receive(uint8_t port, uint8_t *data, uint16_t length,
                       uint16_t timeout);


#endif /* USART_H */
```

Listing G.105: inc/wait_queue.h

```
#ifndef WAIT_QUEUE_H
#define WAIT_QUEUE_H

#include "thread.h"

typedef struct wait_queue_element_s {
  struct wait_queue_element_s *prev;
  struct wait_queue_element_s *next;
  thread_t *thread;
} wait_queue_element_t;

typedef struct {
  wait_queue_element_t *first;
  wait_queue_element_t *last;
} wait_queue_t;

void wait_queue_init(wait_queue_t *queue);
void wait_queue_wake_all(wait_queue_t *queue);
void wait_queue_add_thread(wait_queue_t *queue,
                           wait_queue_element_t *element);
void wait_queue_remove_thread(wait_queue_t *queue,
                              wait_queue_element_t *element);

#endif /* WAIT_QUEUE_H */
```

Listing G.106: gsmsandbox.c

```
#define F_CPU 8000000UL  // 8 MHz


#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```



```
#include <stdint.h>
#include <stdio.h>
#include "gsmsandbox.h"


uint8_t data_ready = 0;

char* message = "hei på deg";

void msg_send(){
    gsm_putstring(send_msg);
    gsmWait(50);
    gsm_putstring((const char*)uartBuffer);
    gsmWait(50);
    gsm_putstring(init_send);
}

void initSerial(void){

    /*
    Specs for our socketmodem:
        Baud Rate = 115200 bps
        Data Bits = 8 bits
        Parity = None
        Stop bits = 1 bit
        Hardware Flow Control RTS/CTS = Disabled
    */

    //SERIAL PORT TO SOCKETMODEM
    //2X speed
    //UCSR0A = (1<<U2X0);
    //rx+tx enable, tx+rx interrupts enable
    UCSR0B = (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);
    //1 stop bits, no parity, asynchronous operation, 8 data bits
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
    //set baudrate (see datasheet table p.196)
    UBRR0L = 3;



    //SERIAL PORT DEBUG TO PC
    //2X speed
    //UCSR0A = (1<<U2X1);
    //rx+tx enable, rx interrupts enable
    UCSR1B = (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1);
    //1 stop bits, no parity, asynchronous operation, 8 data bits
    UCSR1C = (1<<UCSZ11)|(1<<UCSZ10);
    //set baudrate (see datasheet table p.196)
    UBRR1L = 3; //115200

    sei();
```

```
}

void gsmWait(){
    ///while(PINE&0x80){
        //if(!(PINE&0x80)){
        //  break;
    //  }
    //}
    //kaller bare funksjonen delay inntil videre i påvente av en bedre
       måte å løse problemet på.
    delay(75);
}

void pincode(){
    gsm_putstring(pin);
    //while(!data_ready);
    //parse_response();
}

void smsInit(){
    gsm_putstring(msg_format);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(msg_storage);
    //while(!data_ready);
    //parse_response();
    gsmWait();
}

void gprs_conpro_init(void){
    gsm_putstring(con_pro_init_1);
    gsmWait();
    //while(!data_ready);
    //parse_response();
    gsm_putstring(con_pro_init_2);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(con_pro_init_3);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(con_pro_init_4);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(con_pro_init_5);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gprs_client_init();
```



```
}

void gprs_service_init(void){
    gsm_putstring(serv_pro_init_1);
    gsmWait();
    gsm_putstring(serv_pro_init_2);
    gsmWait();
    gsm_putstring(serv_pro_init_3);
    gsmWait();
    //pc_putstring("pr0ver init service on gprs\n");
}

void gprs_client_init(void){
    gsm_putstring(cli_pro_init_1);
    gsmWait();
    gsm_putstring(cli_pro_init_2);
    gsmWait();
    gsm_putstring(cli_pro_init_3);
    gsmWait();
}

void gsm_putstring(const char* s)
{
    uint8_t iterator = 0;

    while(s[iterator]){

        UDR0 = s[iterator++];

        /* Wait for empty transmit buffer */
        while ( !( UCSR0A & (1<<UDRE0)) )
        ;
    }
}


void parse_response(void){

    //data is currently discarded
    data_ready = 0;
}

void pc_putstring(const char* s)
{
    uint8_t iterator = 0;

    while(s[iterator]){

        UDR1 = s[iterator++];

        /* Wait for empty transmit buffer */
```

```
        while ( !( UCSR1A & (1<<UDRE1)) )
        ;
    }
}


//transmission to SocketModem complete
//ISR(USART0_TX_vect){
//
//  ;   //foobar
//
//}

//reception from SocketModem complete
ISR(USART0_RX_vect){

    unsigned char modemdata;
    modemdata = UDR0;
    //uartBuffer[i] = UDR0;
    //forward data from PC to SocketModem
    UDR1 = modemdata;
    //pc_putstring((const char*)uartBufferPtr);
    //while(uartBuffer[i] != 0 && i <= 20 ) {
        //UDR1 = uartBuffer[i];
    if(!(modemdata == '\0')){
        uartBuffer[uartBufferPtr] = modemdata;
        uartBufferPtr++;
    }
    //}

/*
    if (modemdata != "\n"){
        uartBuffer[(uint8_t)uartBufferPtr] = modemdata;
        uartBufferPtr += 1;
    }
*/
}


//transmission to PC complete
//ISR(USART1_TX_vect){
//
//  ;   //foobar
//
//}


//reception from PC complete
ISR(USART1_RX_vect){

    //forward data from PC to SocketModem
    //UDR0 = UDR1;
```



```
    /* Wait for empty transmit buffer */
    while ( !( UCSR0A & (1<<UDRE1)) )
        ;
    UDR0 = UDR1;

}

int main(void){

    initSerial();

    DDRB = 0;
    DDRC = 0xFF;

    PORTC = 0xFF;

    char* startupmsg = "Starting up AVR!\n";
    pc_putstring(startupmsg);

    while(1){

        if(!(PINB&0x01)){
            gprs_conpro_init();
            while(!(PINB&0x01))
            ;
            PORTC=0xFF-0x01;
        }

        if(!(PINB&0x02)){
            smsInit();
            while(!(PINB&0x02))
            ;
            PORTC=0xFF-0x02;
        }

        if(!(PINB&0x04)){
            pincode();
            while(!(PINB&0x04))
            ;
            PORTC=0xFF-0x04;
        }

        if(!(PINB&0x08)){
            msg_send();
            while(!(PINB&0x08))
            ;
            PORTC=0xFF-0x08;
        }

        if(!(PINB&0x10)){
            gsm_putstring(init_send);
```

```
            while(!(PINB&0x10))
            ;
            PORTC=0xFF-0x10;
        }

        if(!(PINB&0x20)){
            gsm_putstring(recv_msg);
            while(!(PINB&0x20))
            pc_putstring("\n");
            pc_putstring("The buffer is ");
            pc_putstring((const char*)uartBufferPtr);
            ;
            PORTC=0xFF-0x20;
        }

        if(!(PINB&0x40)){
            pc_putstring((const char*)uartBuffer);
        //  pc_putstring("\n");
        //  pc_putstring((const char*)uartBufferPtr);
            clear();
            pc_putstring("The buffer is ");
            pc_putstring(uartBufferPtr);
            while(!(PINB&0x40))
            ;
            PORTC=0xFF-0x40;
        }

        //PORTC++;
        //delay(200);

    }

}
```

Listing G.107: gsmsandbox.h

```
// Pin Code
const char* pin = "AT+CPIN=7060\r\n";

// SMS related
const char* msg_format = "AT+CMGF=1\r";
const char* msg_storage = "AT+CPMS=\"MT\"\r\n";
const char* init_send = "\x1a \x1b";
const char* recv_msg = "AT+CMGL=\"ALL\"\r\n";
const char* send_msg = "AT+CMGS=41443840\r\n";

// Internet Connection Setup Profile

const char* con_pro_init_1 = "AT^SICS=0,conType,GPRS0\r\n";
const char* con_pro_init_2 = "AT^SICS=0,inactTO,\"0\"\r\n";
const char* con_pro_init_3 = "AT^SICS=0,dns1,\"129.241.0.200\"\r\n";
const char* con_pro_init_4 = "AT^SICS=0,authMode,\"PAP\"\r\n";
const char* con_pro_init_5 = "AT^SICS=0,apn,\"vpn.netcom.no\"\r\n";
```



```
// Internet Service Setup Profile (server)

const char* serv_pro_init_1 = "AT^SISS=4,srvType,socket\r\n";
const char* serv_pro_init_2 = "AT^SISS=4,conId,0\r\n";
const char* serv_pro_init_3 = "AT^SISS=4,address,\"socktcp://listener
    :5434\"\r\n";

// Internet Service Setup Profile (client)

const char* cli_pro_init_1 = "AT^SISS=1,srvType,socket\r\n";
const char* cli_pro_init_2 = "AT^SISS=1,conId,0\r\n";
const char* cli_pro_init_3 = "AT^SISS=1,address,\"socktcp
    ://129.241.157.199:22022\"\r\n";

volatile unsigned char uartBuffer[1024];
unsigned int uartBufferPtr = 0;
volatile unsigned char* uartBufferSize;
volatile unsigned char connected;


// Prototypes of several methods we need

void gsm_putstring(const char *s);
void pc_putstring(const char *s);
void gprs_service_init(void);
void gprs_client_init(void);
void initSerial(void);
void parse_response(void);
void gsmWait();


void delay(unsigned int d){

    while(d > 25) {
        _delay_ms(25);
        d -= 25;
    }

    _delay_ms(d);
}

void clear(){
    int j = 0;
    uartBufferPtr = 0;
    for(j = 0; j < 1024;j++){
        uartBuffer[j] = '\0';
    }
}
```

Listing G.108: gsmsandbox_tryout.c

```
#define F_CPU 8000000UL  // 8 MHz


#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdint.h>
#include <stdio.h>


uint8_t data_ready = 0;
volatile unsigned char uartBuffer[255];
volatile unsigned char* uartBufferPtr = 0x00;
volatile unsigned char* uartBufferSize;
volatile unsigned char connected;

void gsm_putstring(const char *s);
void pc_putstring(const char *s);
void gprs_service_init(void);
void initSerial(void);
void parse_response(void);
void gsmWait();

char message[255];
int uart0BufferCount = 0;
// Pin Code
const char* pin = "AT+CPIN=7060\r\n";

// SMS related
const char* msg_format = "AT+CMGF=1\r";
const char* msg_storage = "AT+CPMS=\"MT\"\r\n";
const char* init_send = "\x1a \x1b";
const char* recv_msg = "AT+CMGL=\"ALL\"\r\n";
const char* send_msg = "AT+CMGS=41443840\r\n";

// Internet Connection Setup Profile

const char* con_pro_init_1 = "AT^SICS=0,conType,GPRS0\r\n";
const char* con_pro_init_2 = "AT^SICS=0,inactTO,\"0\"\r\n";
const char* con_pro_init_3 = "AT^SICS=0,dns1,\"129.241.0.200\"\r\n";
const char* con_pro_init_4 = "AT^SICS=0,authMode,\"PAP\"\r\n";
const char* con_pro_init_5 = "AT^SICS=0,apn,\"vpn.netcom.no\"\r\n";

// Internet Service Setup Profile (server)

const char* serv_pro_init_1 = "AT^SISS=4,srvType,socket\r\n";
const char* serv_pro_init_2 = "AT^SISS=4,conId,0\r\n";
const char* serv_pro_init_3 = "AT^SISS=4,address,\"socktcp://listener
   :5434\"\r\n";


void delay(unsigned int d){
```



```
    while(d > 25) {
        _delay_ms(25);
        d -= 25;
    }

    _delay_ms(d);
}

void msg_send(){
    gsm_putstring(send_msg);
    gsmWait(50);
    gsm_putstring(uartBuffer);
    gsmWait(50);
    gsm_putstring(init_send);
}

void initSerial(void){

    /*
    Specs for our socketmodem:
        Baud Rate = 115200 bps
        Data Bits = 8 bits
        Parity = None
        Stop bits = 1 bit
        Hardware Flow Control RTS/CTS = Disabled
    */

    //SERIAL PORT TO SOCKETMODEM
    //2X speed
    //UCSR0A = (1<<U2X0);
    //rx+tx enable, tx+rx interrupts enable
    UCSR0B = (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);
    //1 stop bits, no parity, asynchronous operation, 8 data bits
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
    //set baudrate (see datasheet table p.196)
    UBRR0L = 3;



    //SERIAL PORT DEBUG TO PC
    //2X speed
    //UCSR0A = (1<<U2X1);
    //rx+tx enable, rx interrupts enable
    UCSR1B = (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1);
    //1 stop bits, no parity, asynchronous operation, 8 data bits
    UCSR1C = (1<<UCSZ11)|(1<<UCSZ10);
    //set baudrate (see datasheet table p.196)
    UBRR1L = 3; //115200

    sei();
```

```
}

void gsmWait(){
    ///while(PINE&0x80){
        //if(!(PINE&0x80)){
        //  break;
    //  }
    //}
    //kaller bare funksjonen delay inntil videre i påvente av en bedre
       måte å løse problemet på.
    delay(75);
}

void pincode(){
    gsm_putstring(pin);
    //while(!data_ready);
    //parse_response();
}

void smsInit(){
    gsm_putstring(msg_format);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(msg_storage);
    //while(!data_ready);
    //parse_response();
    gsmWait();
}

void gprs_conpro_init(void){
    gsm_putstring(con_pro_init_1);
    gsmWait();
    //while(!data_ready);
    //parse_response();
    gsm_putstring(con_pro_init_2);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(con_pro_init_3);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(con_pro_init_4);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gsm_putstring(con_pro_init_5);
    //while(!data_ready);
    //parse_response();
    gsmWait();
    gprs_service_init();
```



```
}

void gprs_service_init(void){
    gsm_putstring(serv_pro_init_1);
    gsmWait();
    gsm_putstring(serv_pro_init_2);
    gsmWait();
    gsm_putstring(serv_pro_init_3);
}

void gsm_putstring(const char* s)
{
    uint8_t iterator = 0;

    while(s[iterator]){

        UDR0 = s[iterator++];

        /* Wait for empty transmit buffer */
        while ( !( UCSR0A & (1<<UDRE0)) )
        ;
    }
}


void parse_response(void){

    //data is currently discarded
    data_ready = 0;
}


void pc_putstring(const char* s)
{
    uint8_t iterator = 0;

    while(s[iterator]){

        UDR1 = s[iterator++];

        /* Wait for empty transmit buffer */
        while ( !( UCSR1A & (1<<UDRE1)) )
        ;
    }
}



//transmission to SocketModem complete
//ISR(USART0_TX_vect){
//
```

```
//  ;   //foobar
//
//}


//reception from SocketModem complete
ISR(USART0_RX_vect){
    //forward data from PC to SocketModem
    //UDR1 = modemdata;


    uartBuffer[uart0BufferCount++] = UDR0;
    if (UDR0 == '\0')
    {
        message = strdup(uartBuffer);
        uart0BufferCount = 0;
    }
    while ( !( UCSR0A & (1<<UDRE0)) )
        ;
/*
    if (modemdata != "\n"){
        uartBuffer[(uint8_t)uartBufferPtr] = modemdata;
        uartBufferPtr += 1;
    }
*/
}


//transmission to PC complete
//ISR(USART1_TX_vect){
//
//  ;   //foobar
//
//}


//reception from PC complete
ISR(USART1_RX_vect){

    //forward data from PC to SocketModem
    //UDR0 = UDR1;

    /* Wait for empty transmit buffer */
    while ( !( UCSR0A & (1<<UDRE1)) )
        ;
    UDR0 = UDR1;

}

int main(void){

    initSerial();
```



```
    DDRB = 0;
    DDRC = 0xFF;

    PORTC = 0xFF;

    char* startupmsg = "Starting up AVR!\n";
    pc_putstring(startupmsg);

    while(1){

        if(!(PINB&0x01)){
            gprs_conpro_init();
            while(!(PINB&0x01))
            ;
            PORTC=0xFF-0x01;
        }

        if(!(PINB&0x02)){
            smsInit();
            while(!(PINB&0x02))
            ;
            PORTC=0xFF-0x02;
        }

        if(!(PINB&0x04)){
            pincode();
            while(!(PINB&0x04))
            ;
            PORTC=0xFF-0x04;
        }

        if(!(PINB&0x08)){
            msg_send();
            while(!(PINB&0x08))
            ;
            PORTC=0xFF-0x08;
        }

        if(!(PINB&0x10)){
            gsm_putstring(init_send);
            while(!(PINB&0x10))
            ;
            PORTC=0xFF-0x10;
        }

        if(!(PINB&0x20)){
            gsm_putstring(recv_msg);
            while(!(PINB&0x20))
            ;
            PORTC=0xFF-0x20;
        }
```

```
        if(!(PINB&0x40)){
            pc_putstring(uartBuffer);
            while(!(PINB&0x40))
            ;
            PORTC=0xFF-0x40;
        }

        //PORTC++;
        //delay(200);

    }

}
```

Listing G.109: sdfiller.c

```
#include <stdio.h>
#include <stdint.h>



typedef struct {
    uint8_t name[20];
    uint8_t number[12];
    uint8_t key[24];
    uint8_t unused[8];
} phone_book_entry;

char *fgets_custom(char *s, int n, FILE *iop)
{
    register int *c;
    register uint8_t *cs;

    cs = s;
    while (--n > 0 && (c = getc(iop)) != EOF)
        if (c == '\n') break;
        else *cs++ = (uint8_t)c;
    *cs = '\0';
    return (c == EOF && cs == s) ? NULL : s;
}

void generateKey( uint8_t *key)
{
    FILE *random = fopen("/dev/urandom", "r");
    key = fread( key, 24, 1, random);
    fclose( random );
    printf("Key generated from /dev/random ...\n");
}

void sd_append( phone_book_entry *entry, char *file)
{
    FILE *sd_file = fopen(file, "ab+"); // append binary
```



```
    fwrite (entry, sizeof(uint8_t), 64, sd_file);
    fclose(sd_file);
}

int main(int argc, char **argv)
{
    phone_book_entry *entry = calloc(1, sizeof(phone_book_entry) );
    printf("phonebook file: %s\n", argv[1]);

    printf("Enter name of contact (max 20 chars): ");
    //uint8_t *name = calloc(1, sizeof(uint8_t)*20);
    fgets_custom( entry->name, 19, stdin);
    printf("name: %s\n", entry->name);
    printf("Enter number of the contact (max 11 numbers): ");
    //uint8_t *number = calloc(1, sizeof(uint8_t)*12);
    fgets_custom( entry->number, 10, stdin);
    printf("number: %s\n", entry->number );
    //uint8_t *key = calloc(1, sizeof(uint8_t)*24);
    generateKey( entry->key );
    sd_append(entry, argv[1]);
    //free(key);
    //free(number);
    free(entry);
    return 0;
}
```

G.3 C#

This section contains testing code for generating testvectors for the encryption.

Listing G.110: test_vector_generator.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;

namespace Cryptofon
{
    class TestVectorGenerator
    {
        // Encrypts the following plaintexts with the key 0
            x33333333333333333333333333CCCCCCCCCCCCCCCCCCCCCCCC
        // and prints the resulting ciphertexts: 0x80000000, 0x40000000
            , 0x20000000, 0x10000000...
        static void Main(string[] args)
        {
            char[] hex = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
                '9', 'A', 'B', 'C', 'D', 'E', 'F' };

            TripleDES des = TripleDES.Create();
```

```
            des.Key = new byte[] { 0x33, 0x33, 0x33, 0x33, 0x33, 0x33,
                0x33, 0x33, 0x33, 0x33, 0x33, 0x33,
                0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0
                    xCC, 0xCC, 0xCC };
            des.IV = new byte[] { 0, 0, 0, 0, 0, 0, 0, 0 };
            des.Mode = CipherMode.CBC; // Use cipher block chaining
            des.Padding = PaddingMode.None;
            ICryptoTransform crypto = des.CreateEncryptor();

            byte[] plaintext = new byte[] { 0x80, 0, 0, 0, 0, 0, 0, 0
                };
            byte[] cipherText = new byte[8];

            for (int i = 0; i < 8; ++i)
            {
                if (i == 4) // Reset the chaining halfway through
                    crypto = des.CreateEncryptor();
                if (i > 0)
                    plaintext[i - 1] = 0;
                for (int j = 0; j < 8; ++j)
                {
                    plaintext[i] = (byte)(0x80 >> j);
                    crypto.TransformBlock(plaintext, 0, plaintext.
                        Length, cipherText, 0);
                    for (int k = 0; k < cipherText.Length; ++k)
                    {
                        Console.Write(hex[cipherText[k] / 16]);
                        Console.Write(hex[cipherText[k] % 16]);
                    }
                    Console.WriteLine();
                }
            }
            Console.ReadKey();
        }
    }
}
```

Changelog: 23.11.07

All section numbers refer to the previous revision.

Text changes

- Some layout fixes regarding blank page between appendix chapters and italic text
- Many general language corrections
- Introduction - 1 - Changed some text.
- Introduction - 1.2 - Fixed reference to encryption.
- Background - 2.2 - Added section on modes of operation and several illustrations.
- Background - 2.5 - Remove claim that the separate program and data memory is quite unique for microcontrollers.
- Background - 2.6 - Clarifications of the text.
- Background - 2.7 - Added introduction to section.
- Background - 2.7.2 - Grammatical changes and rephrasing.
- Background - 2.7.3 - Rephrasing.
- Design - 3 - Expanded the introduction to the chapter.
- Design - 3.1 - Clarifications.
- Design - 3.2 - Replaced a sentence.
- Design - 3.3.2.1 - Minor clarifying rephrasings.
- Design - 3.7 - Expanded section.
- Implementation - 4 - Added section "FPGA implementation". 4.6.5 and 4.6.6 were moved into this section.
- Implementation - 4.5 - Added 5V workaround.
- Implementation - 4.6 - Added new subsection, "SMS".
- Implementation - 4.6.2 (AVR software, interrupts) - deleted last paragraph and figure 4.32.
- Testing - 5.1.1.2 - Expanded section.
- Testing - 5.2 - Added new section about main program testing.
- Discussion - 7 - Expanded introduction text.
- Discussion - 7 - Added a new section called "Audio codec chip".

- Discussion - 7.3 - Changed the introduction text and added some text in the “Leadership” section. Also added a new section called “Group”.
- Conclusion - 8 - Removed a line that replicated another, and changed “FPGA programming” to “writing VHDL”.
- Acknowledgements - 9 - Removed AVR and PROM from Atmel and changed “Netcom as” to “Netcom AS”.

Figure changes

- Figure 2.1 - Resized the figure.
- Figure 2.5 - Resized the figure.
- Figure 2.6 - Resized the figure.
- Figure 3.3 - Improved the figure text.
- Figure 4.7 - Changed name from external memory to FPGA.
- Figure 4.8 - Changed the figure to a more correct one by adding PROM and codec.
- Figure 4.8 - Resized the figure.
- Figure 4.12 - Resized the figure.
- Figure 4.14 - Resized the figure.
- Figure 4.32 - Deleted.