VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DEVELOPMENT BOARD FOR 32-BIT MICROCONTROLLER ATMEL AT91SAM9261

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MARTIN DEMIN
AUTHOR

VEDOUCÍ PRÁCE Ing. SIMEK VACLAV,
SUPERVISOR

BRNO 2009

Abstrakt

Vestaveny hardware je velice popularni v teto dobe. Proto jsme se rozhodli vytvorit desku s mikrokontrolerem AT91SAM9261 spolu so standartnim a nestandartnim hardwarem. Standartnim, beznym by se dal nazvat port LAN alebo audio vstup-vystup. Nestandartnim, specialnim by mohl byt obvod FPGA firmy Xilinx o velikosti 200k. Toto dovoluje vyuzit zarizeni v oblastech, kde vypocetni sila obycejniho CPU jiz neni dostacujici.

Abstract

Embedded hardware is very popular nowadays; we chose to design a board with AT91SAM9261 microcontroller with some standard and non-standard peripherals attached. As for the standard, common we have included audio port or a LAN controller. The non-standard, special is a 200k Xilinx FPGA. Using the FPGA, we may be able to achive higher throughput in some applications that are not very suitable for plain CPUs.

Klíčová slova

Keywords

embedded, FPGA, xilinx, atmel, AT91SAM9261, development board

Citace

Martin Demin: Development Board for 32-bit Microcontroller Atmel AT91SAM9261, diplomová práce, Brno, FIT VUT v Brně, 2009

Development Board for 32-bit Microcontroller Atmel AT91SAM9261

Prohlášení

Prohlasuji, ze jsem tuto diplomovou praci vypracoval samostatne pod vedenim pana Simeka Vaclava.

. . . .
Martin Demin
June 1, 2009

Contents

1 Introduction **3**
1.1 AT91SAM family . . . 3
1.2 Description of MCU . . . 3

2 Hardware description **5**
2.1 Selection of Parts . . . 6
2.1.1 Parallel FLASH Memory . . . 6
2.1.2 spiFLASH Memory . . . 7
2.1.3 Main Memory (SDRAM) . . . 8
2.1.4 FPGA . . . 8
2.1.5 DRAM for FPGA . . . 8
2.1.6 LAN PHY & MAC . . . 9
2.1.7 Audio Codec . . . 10
2.1.8 UART . . . 10

3 Proposal of Hardware **11**
3.1 Power Supply . . . 11
3.1.1 Low Power Modes . . . 12
3.2 Clock Generators . . . 12
3.2.1 MCU . . . 12
3.2.2 FPGA . . . 13
3.2.3 LAN Controller . . . 13
3.3 Signal Connections . . . 13
3.3.1 System Bus . . . 13
3.3.2 FPGA . . . 13
3.3.3 SPI bus . . . 13
3.3.4 I2S . . . 14
3.3.5 LAN interface . . . 15
3.3.6 Interrupts . . . 15

4 Printed Circuit Board **16**
4.1 Schematic . . . 16
4.1.1 Power Supplies . . . 16
4.1.2 Oscillators . . . 20
4.1.3 System Bus . . . 22
4.1.4 SPI . . . 24
4.1.5 I2S . . . 27
4.1.6 LAN interface . . . 27

4.1.7 USB Host Interface . . . 27
4.1.8 RS232 Converter . . . 27
4.2 Board . . . 29
4.2.1 Creating New Parts in Eagle . . . 29
4.2.2 Components' Placement . . . 30
4.2.3 Routing . . . 32
4.2.4 Production Output from Eagle . . . 35
4.3 Populating the board . . . 37

5 Software **38**
5.1 Applications and Develompent Tools . . . 38
5.1.1 Compilers . . . 38
5.1.2 Debugging . . . 38
5.1.3 Boot Loader . . . 38
5.1.4 Operating System . . . 38
5.2 Theory of Operation . . . 38
5.3 Building and using Tools . . . 39
5.3.1 OpenEmbedded . . . 39
5.3.2 SAM Boot Assistant . . . 40
5.3.3 AT91bootstrap . . . 41
5.3.4 Das U-Boot . . . 42
5.3.5 Linux Kernel . . . 42

6 Conclusion **44**

A List of Appendicies **46**

B Errata **47**

C Schematic **48**

D Board **50**

Chapter 1

Introduction

With the rapid development of hardware and rising complexity of chips is getting design of embedded systems simpler, faster and not least cheaper. For our work we have chosen a low-power microcontroller based on ARM926EJ-S processor. The reason for this was a relatively low pin count, availability of a wide variety of peripherals and a presence of a Memory Managament unit, which will allow running a full-scale GNU/Linux.

The work is divided into several chapters. At first we will introduce the AT91SAM microcontroller family, specially AT91SAM9261, which will be used in our work. We will further proceed into description of choses parts for the board and how they will be connected. Afterwards we will focus onto design of the board in CAD software, configuring the autorouter and preparing data for fabrication. We will finish with a little software overlook.

1.1 AT91SAM family

Atmel's AT91SAM 32-bit ARM Flash MCUs and Embedded MPUs are designed for system control, wired & wireless connectivity, user interface management, low power and ease of use.

The whole family is divided into two bigger groups:

- AT91SAM Flash MCUs feature Flash memories with densities of 16k Bytes, migrating up to 512k Bytes and operating frequencies up to 200MHz. A coherent set of peripherals allow reuse of software over the whole family. Usage of DMA channels provides optimization of bus utilization, freeing the processor for application. .
- a AT91SAM Embedded MPU are ARM926-based with operating frequencies up to 400 MHz.

1.2 Description of MCU

Note: Following section may contain information taken from AT91SAM9261 datasheet [2]. The hearth of MCU is a ARM926EJ-S ARM Thumb Processing unit which features a DSP instruction extension, JAVA Acceleration Technology and memory management unit. For faster execution are 16Kbyte data cache and 16Kbyte instruction cache included. When it comes to debugging the core offers us EmbeddedICE debug communication channel support.

The die also has in 32 Kbytes of internal ROM and 160 Kbytes of static RAM built in. Internal ROM allows for various boot strategies, while internal RAM can be used for frame buffer or for other DMA transfers.

The following peripherals are present for usage:

- LCD controller
- USB 2.0 full speed dual port host controller with in-chip transceiver
- USB 2.0 full speed device port with transceiver
- Reset controller, shutdown controller, clock generator and power management unit
- Advanced interrupt controller
- Debug unit
- Real-time clock
- Multimedia Card interface
- Three Synchronous Serial controllers
- Three UART ports
- Two SPI ports
- Two Wire interface
- JTAG port

A block diagram is present in the figure 1.1.

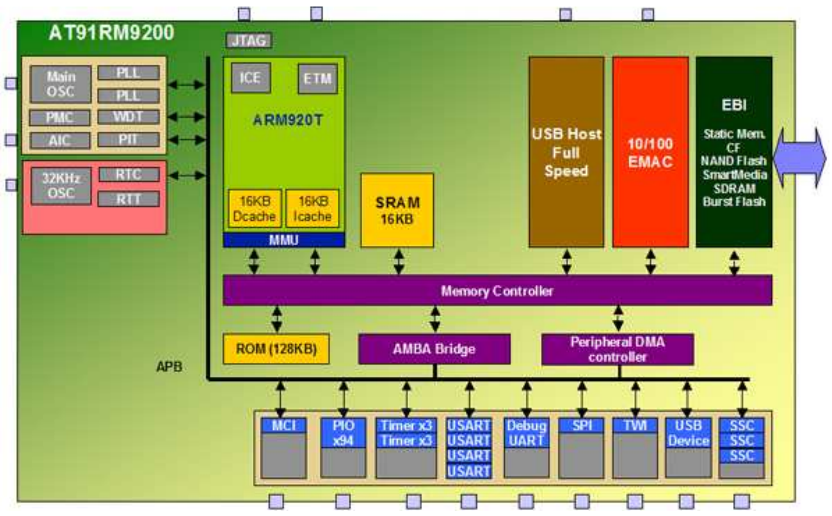


Figure 1.1: Block diagram

Chapter 2

Hardware description

As mentioned, the core of the board will be an AT91SAM9261 microcontroller with peripherals attached. The board should have following proposed features:

- 8 Gbit Flash memory
- 128 Mbytes of SDRAM
- 64 Mbit spiFLASH boot ROM
- FPGA with 16 Mbytes of DRAM attached
- LAN controller with Ethernet port
- Audio IO

After specification, we have created a simplistic block diagram which is shown in the figure 2.1.

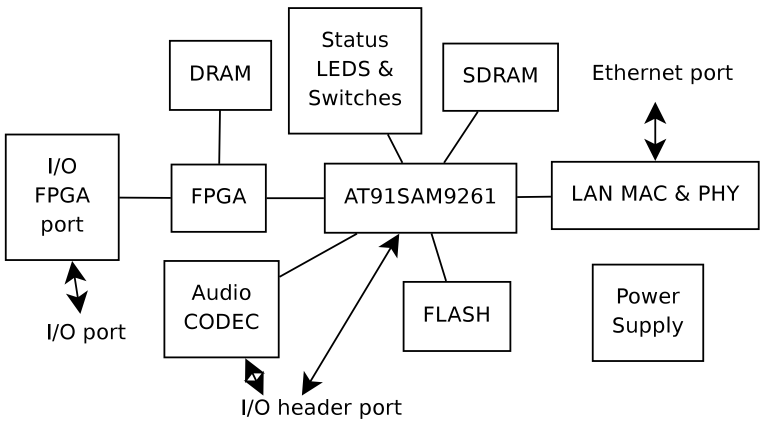


Figure 2.1: Block diagram

2.1 Selection of Parts

2.1.1 Parallel FLASH Memory

For parallel FLASH memory we have chosen K9K8G08U0A 8Gb in a 48-TSOP surface mounted package[1].

Power supply requirements:

- 3.3 V

Offered in 1G x 8bit, the K9K8G08U0A is a 8G-bit NAND Flash Memory with spare 256M-bit. Its NAND cell provides the most costeffective solution for the solid state application market. A program operation can be performed in typical 200Îźs on the (2K+64)Byte page and an erase operation can be performed in typical 1.5ms on a (128K+4K)Byte block. Data in the data register can be read out at 25ns cycle time per Byte. The I/O pins serve as the ports for address and data input/output as well as command input. The on-chip write controller automates all program and erase functions including pulse repetition, where required, and internal verification and margining of data. Even the write-intensive systems can take advantage of the K9K8G08U0A's extended reliability of 100K program/ erase cycles by providing ECC(Error Correcting Code) with real time mapping-out algorithm. The K9K8G08U0A is an optimum solution for large nonvolatile storage applications such as solid state file storage and other portable applications requiring non-volatility. An ultra high density solution having two 8Gb stacked with two chip selects is also available in standard TSOPI package.

When it comes to theory, there are two types of flash memories: NAND and NOR.

NOR vs. NAND

Reading from NOR memories is very much similar to reading from RAM or ROM because NOR memories use the very same address and data interface along with control signals like RD, WR and CE. Single bytes may be programmed individually but only block erasure is possible. Typical block sizes are 64, 128 or 256 KB. Most older NOR flashes do not feature any bad block management and this is left onto the device driver. Special commands are necessary for programming, erasure or locking. These are issued using Common Flash Memory Interface (CFI) which also supports identification of the chip. These chips are used mostly as boot memories on embedded devices because no internal ROM or flash memory are needed.

As for the NAND memories, these feature much higher densities and, on the contrary, much more complicated interface. The chips function as block devices much like hard drives or memory cards. Each block contains some number of pages. Typical page sizes are 512, 2048 or 4096 bytes. Each page has its own ECC bits for correction/error detection.

While reading and programming can be performed on page basis, erasure can only be performed on a block basis. Another obstacle is that a block can be written only sequentially.

NAND devices require bad block management in the device driver or controller. In linux, this is done by the JFFS2 filesystem. This filesystem also provides wear leveling since every block can be erased only limited number of times. Most chips provide extra free space to replace bad blocks.

ECC is provided to correct errors during reading. Depending on the number of bits used for ECC, it may correct 1 bit on each 2048bits or up to 22 bits in 2048 bits for MLC NAND devices. The bad block may be marked bad and avoided from usage.

NAND memories use less complicated communication interface than in NOR flashes. It consist of 8 or 16 data lines and 6 other control signals: CE, R/B, ALE, CLE, OE and WE. Usage of these flashes require specialized hardware or software controller.

SLC vs. MLC

As the requirement for memory sizes increases, companies are pushed to research for more dense memories. Currently a trend is to use multi-level cell in flash memories. These memories allow to store more than 1 bit per cell - typically up to 4 bits. This increases storage capacity but also increases error-proneness. These device have only 10k write life durability. To compensate for the errors, stronger ECC codes are implemented. The size of the chips are as high as 64Gbits.

2.1.2 spiFLASH Memory

AT45DB642D-TU 64Mbit boot flash was selected for this purpose as it is widely available and supported as boot memory. The part is available with 2 interfaces: parallel or serial. As a boot ROM we required and have chosen the serial version in CASON package.

Power supply requirements:

- 2.7 - 3.3 V

The AT45DB642D is a 2.7-volt, dual-interface sequential access Flash memory ideally suited for a wide variety of digital voice-, image-, program code- and data-storage applications. The AT45DB642D supports RapidS serial interface and Rapid8 8-bit interface. RapidS serial interface is SPI compatible for frequencies up to 66 MHz. The dual-interface allows a dedicated serial interface to be connected to a DSP and a dedicated 8-bit interface to be connected to a microcontroller or vice versa. However, the use of either interface is purely optional. Its 69,206,016 bits of memory are organized as 8,192 pages of 1,024 bytes (binary page size) or 1,056 bytes (standard DataFlash page size) each. In addition to the main memory, the AT45DB642D also contains two SRAM buffers of 1,024 (binary buffer size) bytes/1,056 bytes (standard DataFlash buffer size) each. The buffers allow receiving of data while a page in the main Memory is being reprogrammed, as well as writing a continuous data stream. EEPROM emulation (bit or byte alterability) is easily handled with a self-contained three step read-modifywrite operation

2.1.3 Main Memory (SDRAM)

The processor offers us 2 signal voltages 1.8V for low-power DRAM or 3.3V for standard SDRAM. To avoid using BGA packages we have selected standard 3.3V SDRAM MT48LC32M16A2TG capable of running up to 133MHz clock. The RAM's width is 16 bit and MCU supports 32 bit access. To gain speed and capacity there will be 2 parts connected in parallel.

Power supply requirements:

- 3.3 V

The K4S511632D is 536,870,912 bits synchronous high data rate Dynamic RAM organized as 4 x 8,392,608words by 16bits, fabricated with SAMSUNG's high performance CMOS technology

2.1.4 FPGA

We have decided to include FPGA in the design to allow high-speed processing for specific applications. FPGA will have a DRAM attached. It can also be used to generate VGA signal along with the DRAM to store frame buffer. Accessing FPGA's DRAM through main bus for displaying graphics will significantly offload main memory and bus. The FPGA will further provide IO pins for general usage. The chip we have chosen to use is an XC3S200.

Power supply requirements:

- 3.3 V - Output driver
- 1.200 - Internal supply voltage
- 2.500 - Auxilary supply voltage

The selected FPGA contains:

- 200K of System Gates
- 30K of Distributed RAM Bits
- 216K of Block RAM Bits
- 12 Dedicated Multipliers
- 4 DCMs

2.1.5 DRAM for FPGA

Since the requirement for capacity of RAM for FPGA may not be high, we have selected to use a 64Mbit SDRAM with 8 bit wide bus. Part number: MT48LC8M8A2P-75.

Power supply requirements:

- 3.3 V

2.1.6 LAN PHY & MAC

The MCU does not feature any MAC interface. We have found SiLabs' CP2200/1 MAC and PHY interesting and included it in our design. The PHY will furter be connected to a LAN port with integrated balun.

Power supply requirements:

- 3.3 V

The CP2200/1[10] is a single-chip Ethernet controller containing an integrated IEEE 802.3 Ethernet Media Access Controller (MAC), 10BASE-T Physical Layer (PHY), and 8 kB Non-Volatile Flash Memory available in a compact 48-pin TQFP package. The CP2200/1 can add Ethernet connectivity to any microcontroller or host processor with 11 or more Port I/O pins. The 8-bit parallel interface bus supports both Intel and Motorola bus formats in multiplexed and non-multiplexed mode. The data transfer rate in non-multiplexed mode can exceed 30 Mbps. The on-chip Flash memory may be used to store user constants, web server content, or as general purpose nonvolatile memory. The Flash is factory preprogrammed with a unique 48-bit MAC address stored in the last six memory locations. Having a unique MAC address stored in the CP2200/1 often removes the serialization step from the product manufacturing process of most embedded systems. The CP2200/1 has four power modes with varying levels of functionality that allow the host processor to manage the overall system power consumption. The optional interrupt pin also allows the host to enter a sleep mode and awaken when a packet is received or when the CP2200/1 is plugged into a network. Auto-negotiation allows the device to automatically detect the most efficient duplex mode (half/full duplex) supported by the network. A block diagram of a LAN subsystem is shown in the figure 2.2.

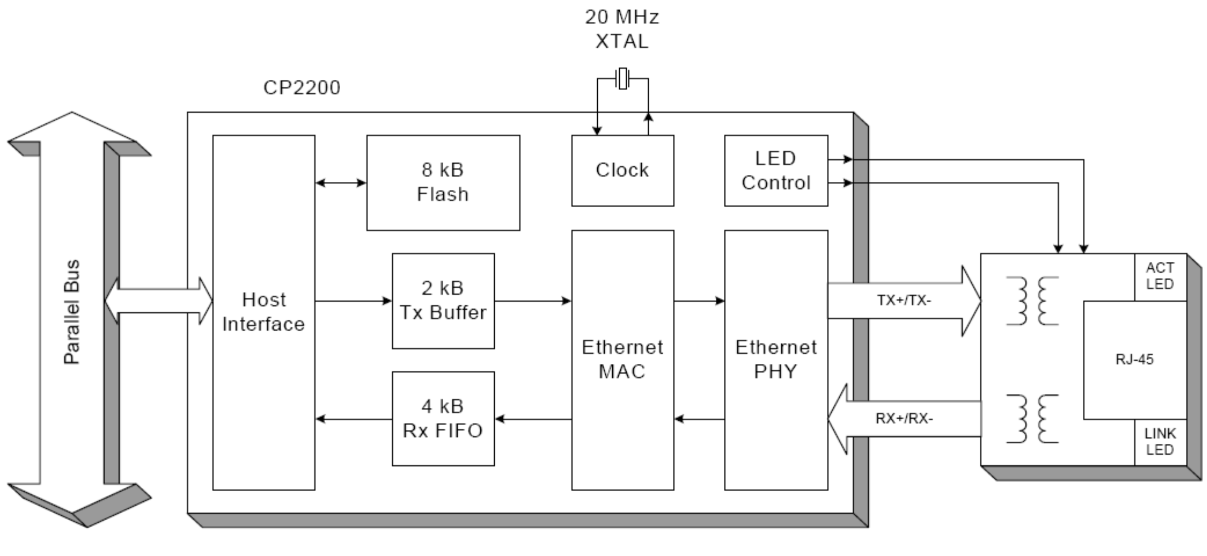


Figure 2.2: Block Diagram of CP2200[10]

2.1.7 Audio Codec

As an audio codec, we have chosen to use a TLV320AIC23B[5], which has the following power supply requirements:

- 3.3 V - Digital power supply
- 5 V - Analog power supply

The TLV320AIC23B is a high-performance stereo audio codec with highly integrated analog functionality. The analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) within the TLV320AIC23B use multibit sigma-delta technology with integrated oversampling digital interpolation filters. Data-transfer word lengths of 16, 20, 24, and 32 bits, with sample rates from 8 kHz to 96 kHz, are supported. The ADC features an architecture with up to 90-dBA signal-to-noise ratio (SNR) at audio sampling rates up to 96 kHz, enabling high-fidelity audio recording in a compact, power-saving design. The DAC modulator features an architecture with up to 100-dBA SNR at audio sampling rates up to 96 kHz, enabling high-quality digital audio-playback capability, while consuming less than 23 mW during playback only. The TLV320AIC23B is the ideal analog input/output (I/O) choice for our application. Integrated analog features consist of stereo-line inputs with an analog bypass path, a stereo headphone amplifier, with analog volume control and mute, and a complete electret-microphone-capsule biasing and buffering solution. The headphone amplifier is capable of delivering 30 mW per channel into 32Ω. The analog bypass path allows use of the stereo-line inputs and the headphone amplifier with analog volume control, while completely bypassing the codec, thus enabling further design flexibility. A microphone bias-voltage output provides a low-noise current source for electret-capsule biasing. The AIC23B has an integrated adjustable microphone amplifier (gain adjustable from 1 to 5) and a programmable gain microphone amplifier (0 dB or 20 dB). The microphone signal can be mixed with the output signals if a sidetone is required.

2.1.8 UART

For debugging and development purposes, a serial connection will be provided that allows direct connection of a PC via a null-modem cable. The MCU itself has an UART controller onboard. These pins will be connected to a MAX3232 chip that will translate the signal levels to the RS232 standard.

Power supply requirements:

- 3.3 V for MAX3232

Chapter 3

Proposal of Hardware

3.1 Power Supply

As was seen during initial parts consideration, there are 4 different power supply voltages reguired: 5, 3.3, 2.5, 1.2 . All voltages will be generated by using bulk voltage regulators, except for 5V branch.

5 V - The branch is needed for USB host port. The linear regulator selected for the design is LF50CV. This regulator has only 0.2V drop so a power supply with 5.2V or more output will be useable to power the system.

The LF00 series are very low drop regulator available in various packages and also in wide range of output voltages. The low drop voltage makes them suitable for low noise, low power applications. Some packages are pin compatible with famous LM7805 regulators. Only input and output capacitors are required. This regulator can provide up to 1A this should be fully compliant with two 500mA USB devices connected to the board.

2.5 V and 1.2 V - The branches both utilize LM2574M-ADJ adjustable bulk voltage regulator.

LM2574 is a monolithic integrated circuit that provide all the active functions for a step-down switching regulator, capable of driving a 0.5A load with sufficient line and load regulation. It requires only a minimum of external components and that makes it suitable for out application. This IC has an integrated fixed-frequency oscillator and transistor. The regulator's output can be inhibited using an ON/OFF pin which is a required feature for suspend.

3.3 V The branch is built on LM2738 adjustable bulk voltage regulator capable of delivering up to 1.5A current using the integrated transistor.

LM2738 regulator is a monolithic, high frequency, PWM step-down DC/DC converter. It provides all active functions for local DC/DC conversion with fast transient response and accurate regulation. Thanks to the packages, it is provided in, the whole regulating system can be implemented in a small area. The switching frequency is 1.6MHz which allows to use only a very small inductor and capacitor. Also an inhibit pin is provided and the regulator may be shut down to save power during suspend. Stand-by current is only 400nA.

1.2 V back-up - The branch will be used to power the VDDBU branch of the MCU. This will provide supply to the shutdown controller to allow wake-up using a button and for the RTC. This branch will have very low pover consumption of 3 uA at room temperature and can reach up to 20 uA when MCU heated to 85 degrees centigrade. The voltage will be regulated using LM317 linear regulator.

The LM317 regulator is capable of providing up to 1.5A in a wide input range. It features adjustable output voltage which can be programmed using a resistor voltage divider. The output voltage can be as low as 1.2V and this makes it suitable for this application. It needs only input and output bypass capacitor.

3.1.1 Low Power Modes

As with all devices today, there is a need for low power modes, e.g. shutdown or sleep. Both of these modes will be supported. To support shutdown, there are 2 dedicated pins provided by MCU. One separate part of MCU, shutdown controller, provides control signal to voltage controller. If we want to shutdown the board, we only have to modify corresponding registers in the MCU and the voltage regulator will be disabled. This cuts off power for the whole board and peripherals, leaving only the shutdown controller powered.

As for suspend, the situation gets more complicated, the devices will have to be individually forced into low power modes by software and only then the MCU can be switched to sleep mode, leaving only necessary parts of MCU powered to maintain program control. Omitting control of peripherals would have negative effect on power consumption.

Since there is a separate master-capable device present, the FPGA, this will have to be disabled too. The FPGA itself does not provide any direct control for that, however, the datasheet describes that 2 supply voltages may be taken down, leaving only IO logic powered. The IO logic contains protective diodes to the internal supply lanes and these would provide excessive load on the bus, when supply not provided. To accomplish this, the two FPGA dedicated power supplies 2.5V and 1.2V will be connected through a P channel MOSFET and may be enabled/disabled as needed.

When it comes to other peripherals: LAN and audio codec, these provide direct interfaces for low power modes. The audio codec is equipped with a PDWN pin, which will force ADC and DAC to turn off. The same comes for the headphone amplifier used. With the LAN controller, the situation is a little different. This will have to be accomplished from software through a control register.

3.2 Clock Generators

3.2.1 MCU

The MCU requires 2 clock oscilators: slow and main. The slow oscilator is used for powerup and RTC. Slow oscilator uses 32.768 kHz reference crystal. For main crystal, the frequency can be up to 50 MHz, but we have chosen to use a 18.432 MHz crystal. It is important that this crystal is supported for autodetection in the AT91's ROM, if we want to use the serial debug port[2].

3.2.2 FPGA

As for FPGA, it requires a clock signal connected to one of the clock generators. For simplicity and interoperability with existing platform, we have chosen to use the same resonator as used by FITkit - 7.3728 MHz [15].

3.2.3 LAN Controller

The LAN Controller uses a 20 MHz reference crystal.[10]

3.3 Signal Connections

3.3.1 System Bus

The MCU provides 32-bit-wide System Bus called External Bus Interface. It features 32 data lines and 26 address lines, along with 8 chip select lines.

Note: We originally proposed to use all 32 data lines, but later this turned out to be an issue and was reduced to 16 bits. See Appendix B.

The MCU has a direct support for NAND Flash memories and it is therefore not necessary to provide any auxiliary logic to support these memories. The memory will be connected directly to the MCU.

The SDRAM will be controlled by integrated SDRAM controller. The two SDRAM chips will share all of its address lines and be only dividing 32 bit data bus into two 16-bit lanes.

As for the other devices connected to the system bus, the connection is simple cross connection of RD, WR, data & address lines and using a coresponding CS line.

3.3.2 FPGA

The bus of the FPGA is very flexible and depends on the internal configuration. The SDRAM will be connected to the FPGA the same way as it is connected in the FITkit [15] to allow usage of the same software. The FPGA will further provide its spare GPIOs to a header to allow connection of peripherals, e.g. ADCs, DACs.

The FPGA will be configured through SPI interface and some further signals must be provided from the GPIOs of MCU for control.

3.3.3 SPI bus

The MCU provides SPI bus lines along with 4 Chip Selects, allowing connection of 4 different devices. Two of these will be occupied by SPI flash memory and FPGA as configuration interface. Third will be connected to the audio codec to allow control of volume, sampling rate etc.

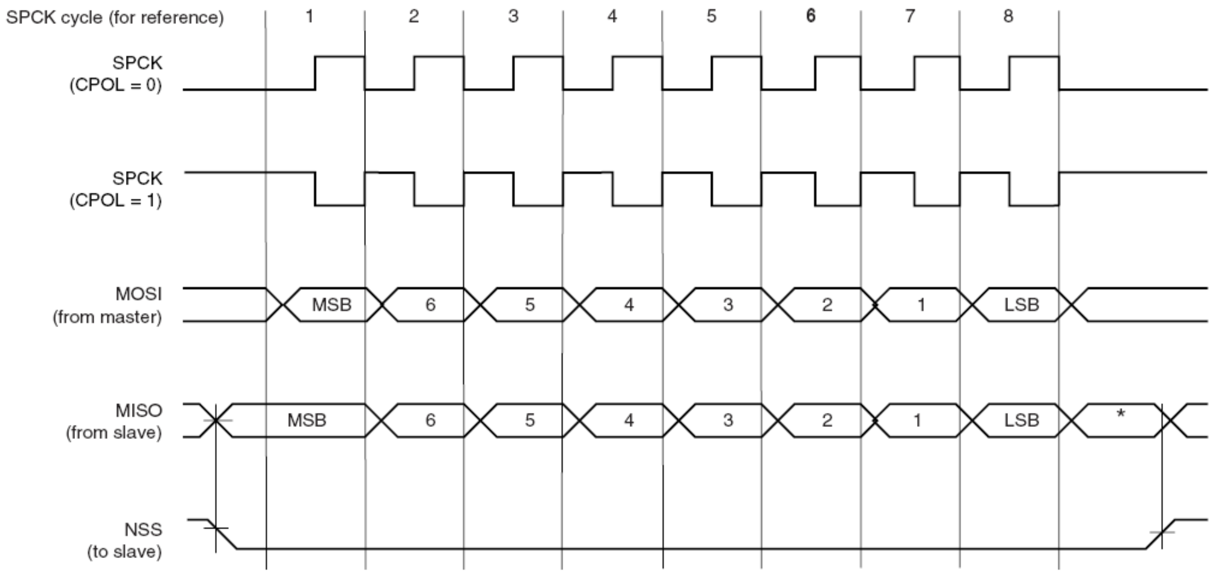


Figure 3.1: Typical timing of an 8 bit transmittion on SPI[2]

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices may be connected using separate chip select signals. The SPI bus specifies following signals:

- SCLK - Serial Clock provided by the master
- MOSI/SIMO - Master Output, Slave Input provided by the master
- MISO/SOMI - Master Input, Slave Output provided by the slave
- SS - Slave Select (Chip Select, active low)

An example of a data transmittion is shown in the figure 3.1. Many devices allow clock frequencies to be as high as 20MHz and even higher. Word length does not have to be limited to 8 bits and may have arbitrary size.

3.3.4 I2S

The I2S will be used for audio codec. The I2S will provide IO interface for digital audio transfer. Both audio input and output will be supported.

Integrated Interchip Sound, or I2S for short, is an electrical serial bus interface specially designed for transmitting digital audio signals. The bus features separate data and clock lines which results in lower jitter. Normally the bus has the following signals:

- Bit clock
- Word clock
- One or more data lines

One master is present on the bus and is responsible for generation of clock signals. The data is transmitted in MSB to LSB order. Various word lengths are supported.

3.3.5 LAN interface

The analog IO interface from the LAN PHY will be connected to an embedded connector with a balancing balun and LEDs.

3.3.6 Interrupts

As with all flexible hardware, the board should allow interrupt based communication with peripherals. Most of the devices provide interface for such communication, and these lines will be routed to the MCU. FPGA will share some GPIOs with MCU and these may be dedicated for use as interrupt requests.

Chapter 4

Printed Circuit Board

For design of the board, application EAGLE Layout Editor will be used. The PCBs dimensions will be determined uppon design, since no size constrains are given. The PCB itself will be a 4-layer-board, because number of the signal traces for the data bus is very restraining along with some more than 16 address lines.

The relatively high operation frequency of peripherals (100MHz) may generate a lot of noise that may be transmitted onto power lanes and disrupt error-free operation. For these purposes various capacitors will be placed along the board and especially MCU and FPGA. The FPGA manufacturer also provides very detailed guide for calculation of power supply requirements and capacitor placement, but this will not be critical for our board. Special care must also be taken not to produce a much longer trace for some single signal traces. This may especially happen with a misconfigured autorouter. These longer traces may provide for some longer delay.

4.1 Schematic

We have decided to draw the schematic onto a single sheet. In this section we shall describe the most important pieces of the design. For reference, the schematic is available in the Appendix A. We begin with the power supply and go on with auxiliary circuits for MCU as shutdown, PLL filter, serial module or boot mode select. Further we will go into details on connecting the system bus to the memories, FPGA and LAN. Finally we will show how the audio and LAN circuits are connected.

4.1.1 Power Supplies

While designing the power supplies, we used the original LM2738 and LM2574 datasheets as reference. These provide us with typical schematic diagram and with necessary calculations for inductors and capacitors, respectively.

LM2738

The typical connection of LM2738 is shown in the figure 4.1. Now we have to focus onto the selection of the necessary inductor, the output capacitor and the setting resistors. The datasheet[8] provides us with a detailed guide that we have followed.

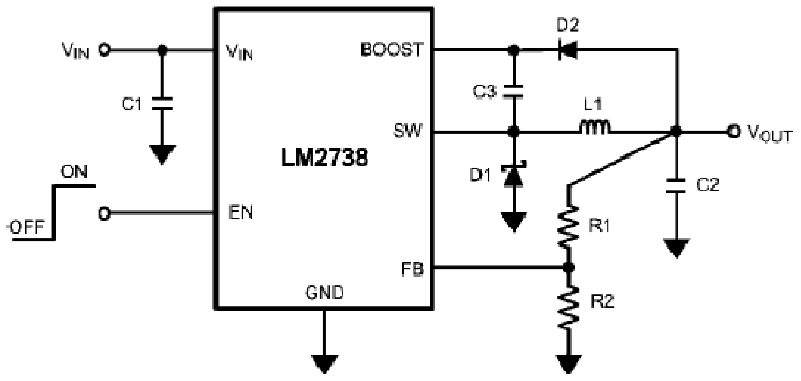


Figure 4.1: Typical schematic of LM2738

1. Voltage setting is done using a voltage divider connected to the output and the feedback pin of the IC. For calculation, the following formula is used:

$$R_1 = R_2\left(\frac{V_{OUT}}{V_{REF}} - 1\right)$$

In our case, for the 3.3V, we have chosen the R_2 to be 10k and we get:

$$R_1 = 10k\left(\frac{3.3V}{0.8V} - 1\right) = 31.25k$$

2. As for a correct inductor L_1, we have to perform some calculations according to the datasheet:

$$L = \left(\frac{DT_S}{2\Delta i_L}\right) \times (V_{IN} - V_{OUT}),$$

where D is duty cycle calculated as:

$$D = \frac{V_O + V_D}{V_{IN} + V_D + V_{SW}},$$

where V_D is voltage drop of the diode in the range from 0.3V to 0.7V. Specification for the chosen shottky diode MBRA210LT3G says 0.35V. V_O is output voltage, V_{IN} is input voltage and finally, V_{SW} is voltage drop on the switching transistor calculated as:

$$V_SW = I_{OUT} \times R_{DSON},$$

where I_{OUT} is output current and R_{DSON} is resistance of the internal switching transistor.

Last we need to know the Δi_L given by a general rule:

$$\Delta i_L = 0.1 \times I_{OUT} \rightarrow 0.2 \times I_{OUT}$$

Now we may proceed with calculations with given constants:

$$\begin{aligned} I_{OUT} &= 1A \\ \Delta i_L &= 0.15 \times 1A = 0.15A \\ V_{SW} &= 1A \times 250mohm = 0.25V \\ T_S &= 0.000000625s \\ D &= \frac{3.3V + 0.35V}{6V + 0.35V + 0.25V} = 0.553 \end{aligned}$$

Finally, as for the inductor:

$$L = \left(\frac{0.5530.000000625s}{2 \times 0.15A}\right) \times (6V - 3.3V) = 3.1\mu H$$

We have chosen to use an inductor NR6045T1R3N from Digi-Key Corporation.

3. When it comes to the output capacitor, the datasheet recommends at least $22\mu F$ electrolyt better with lower ESR. Usage of parallel multilayer ceramic capacitors of X7R or X5R type is also recommended since they provide high frequency noise filtering.

4. We chose a Schottky catch diode because of lower forward voltage drop and faster switching times. The Digi-Key part number is MBRA210LT3G.

LM2574

The typical connection of LM2574 is shown in the figure 4.2. Now we have to focus onto the selection of the necessary inductor, the output capacitor and the setting resistors. The datasheet[7] provides us with a detailed guide that we have followed.

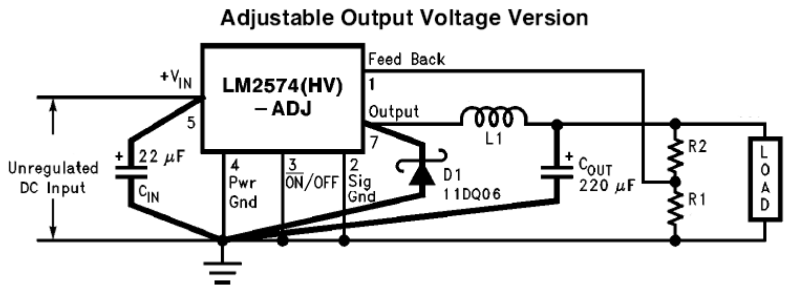


Figure 4.2: Typical schematic of LM2574

1. Voltage setting is done using a voltage divider connected to the output and the feedback pin of the IC. For calculation, the following formula is used:

$$V_{OUT} = V_{REF}\left(1 + \frac{R_2}{R_1}\right)$$

where V_{OUT} is the requested output voltage, V_{REF} is 1.23V and R_1 is between 1k and 5k. To calculate the R_2, we get:

$$R_2 = R_1 \left(\frac{V_OUT}{V_REF} - 1 \right)$$

In our case, for the 2.5V we have chosen the R_1 to be 4.75k and we get:

$$R_2 = 4.75k \left(\frac{2.5V}{1.23V} - 1 \right) = 4.90k$$

In case of the 1.23V supply, the FB pin has to be directly connected to the output. This way the regulator will maintain a voltage of 1.23V.

2. Inductor value is chosen from the graph 4.3, using two reference values, E . T and the maximum load current. The E . T value is calculated using the following formula:

   $$E.T = (V_IN - V_OUT) \frac{V_OUT}{V_IN} \bullet \frac{1000}{f(inkHz)} (V \bullet \mu s)$$

   For the 2.5V power source we get:

   $$E.T = (6V - 2.5V) \frac{2.5V}{6V} \bullet \frac{1000}{52kHz} (V \bullet \mu s) = 28V \bullet \mu s$$

   For the 1.2V power source we get:

   $$E.T = (6V - 1.2V) \frac{1.2V}{6V} \bullet \frac{1000}{52kHz} (V \bullet \mu s) = 18V \bullet \mu s$$

   Picking the maximum current to be about 300mA for each supply and using the provided graph 4.3, we get the the inductors to be $150\mu H$ and $100\mu H$, respectively. The inductors have to be suitable for 52kHz, as this is the operating frequency of the buck regulator and widthstand a minimum of 1.5 times the maximum load current. The parts TSL1112RA-151K1R1-PF and TSL1112RA-101K1R4-PF were chosen from Digi-key Corporation[3].

3. As for the output capacitors, also a formula is specified:

   $$C_OUT >= 13,300 \frac{V_IN(max)}{V_OUT \bullet L(\mu H)} (\mu F)$$

   For our two cases, 2.5V and 1.2V, we get:

   $$C_OUT >= 13,300 \frac{6V}{2.5V \bullet 150\mu H} = 212\mu F$$
   and
   $$C_OUT >= 13,300 \frac{6V}{1.2V \bullet 100\mu H} = 665\mu F$$

   However, for both cases we have chosen to use a $1000\mu F$ capacitor rated for the required voltage.

LM317

The LM317 is connected as shown in the typical connection diagram 4.4. The input capacitor was ommited because of the presence of a global input capacitor and output capacitor was set to be $100\mu F$. The ADJ pin is connected directly to the ground thus the output voltage is 1.2V.

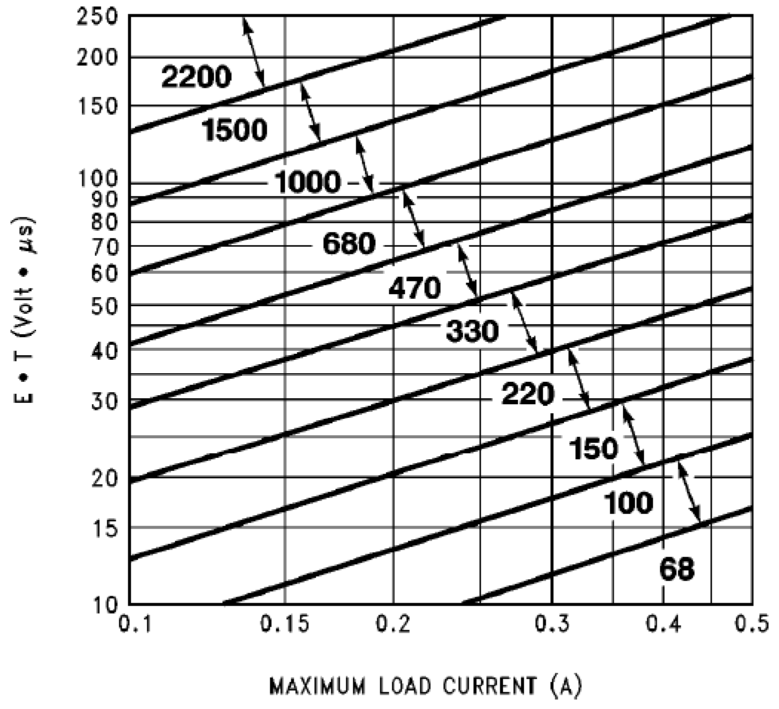


Figure 4.3: Helper Graph for Inductor Selection

LF50CV

The LF50CV is connected as shown in the figure 4.5. Input capacitor was not used because a global input capacitor is present. The output capacitor is a $100\mu F$ electrolytic one.

Sleep Mode

Two dedicated pins are present on the MCU for power management - WKUP and SHDWN. The WKUP pin is input only and is provided to wake-up the MCU from a sleep. The whole shutdown controller to which these pins are connected is powered using the backup 1.2V power supply that is always present. This supply is also provided for reset controller to allow a proper reset uppon wake-up. The WKUP is configurable to be sensitive to rising, falling or both edges. We have connected this pin to a switch and a pull-up resistor. The resistor connects the net to 1.2V backup voltage and switch connects the net to the ground. This way a falling edge is generated.

The shutdown controller has an output pin SHDWN that is used to control the power for the rest of the system. This pin is connected to enable pin of the DC/DC controllers. One input had to be inverted using a MOS-FET transitor and a pull-up resistor.

4.1.2 Oscillators

The MCU requires 2 oscillators, though only 1, slow, is needed. The slow clock oscillator is used to provide clock reference for the shutdown controller, reset controller and also may be used to clock the ARM core. The main oscillator is used to provide clock for PLL synthesis.

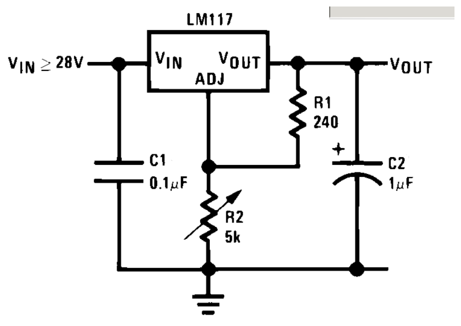


Figure 4.4: Typical connection of LM317[9]

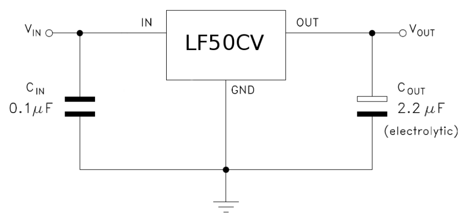


Figure 4.5: Typical connection of LF50[14]

The slow clock oscillator uses a 32768Hz crystal connected as shown in the figure 4.6. No additional component is necessary.

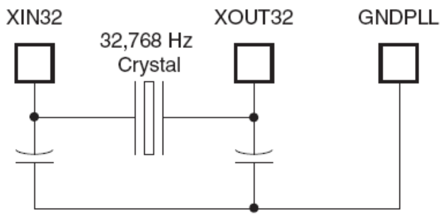


Figure 4.6: Connection of slow clock oscillator[2]

The main clock oscillator uses a 18.432MHz crystal connected as shown in the figure 4.6. An additional resistor may be connected, but is required only for crystals with frequency less than 8MHz.

Both PLLs require a RC filter. Such a filter is shown in the figure 4.8. These filter have to be picked to provide sufficent filtering and allow fast oscillator startup. We have chosen the values to be as followed:

$$R_1 = 4.87k$$
$$C_1 = 4.7n$$
$$C_2 = 470p$$

Both PLL filter us the same value of components. The same values are used widely in the

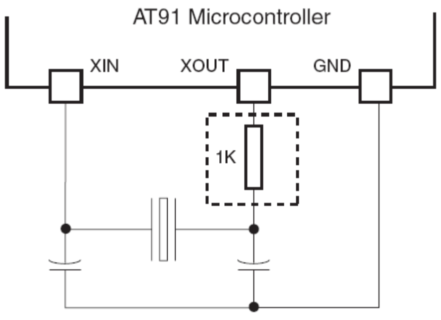


Figure 4.7: Connection of main clock oscillator[2]

designs for such processors and therefore no calculations were done.

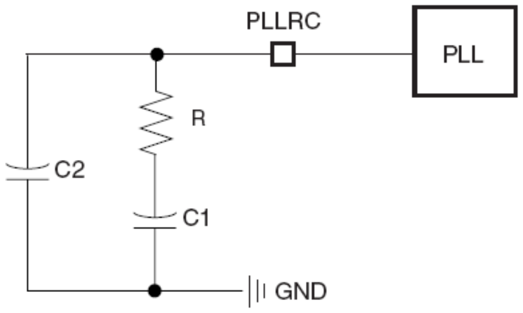


Figure 4.8: Connection of the Bypassing Filter[2]

The FPGA uses a dedicated oscillator ASFL1-7.3728MHZ-EK-T that is connected directly to the power supply and FPGA DCM input pin.

4.1.3 System Bus

SDRAM

The SDRAM is connected to the MCU as is shown in the figure 4.9. Explanation of single signals is provided in the table 4.1.

An SDRAM uses multiplexed address lines to allow selecting a row and a column that we are working with. Most SDRAMs have more banks that are chosen using the BA0-BA1 pins. SDRAMs are provided with 8, 16 and 32-bit width of data bus. For the purpose of this work we use a 16-bit-wide data bus. A typical block diagram of SDRAM is shown in the figure 4.10 to help understand the way addressing in SDRAM works.

We originally planned to use 32-bit-wide system bus, but the complexity of the board rose significantly and we were forced to omit 1 SDRAM chip and reduce data bus to 16 bits.

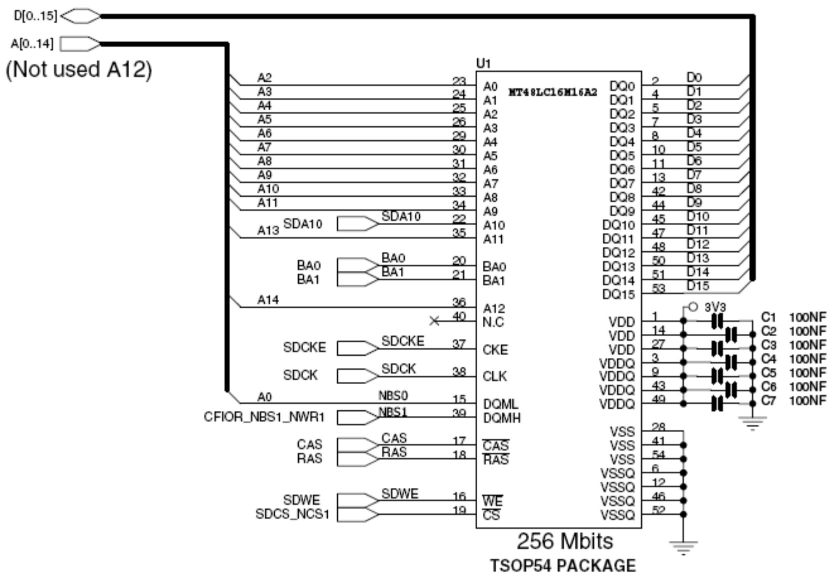


Figure 4.9: Connection of SDRAM to MCU[2]

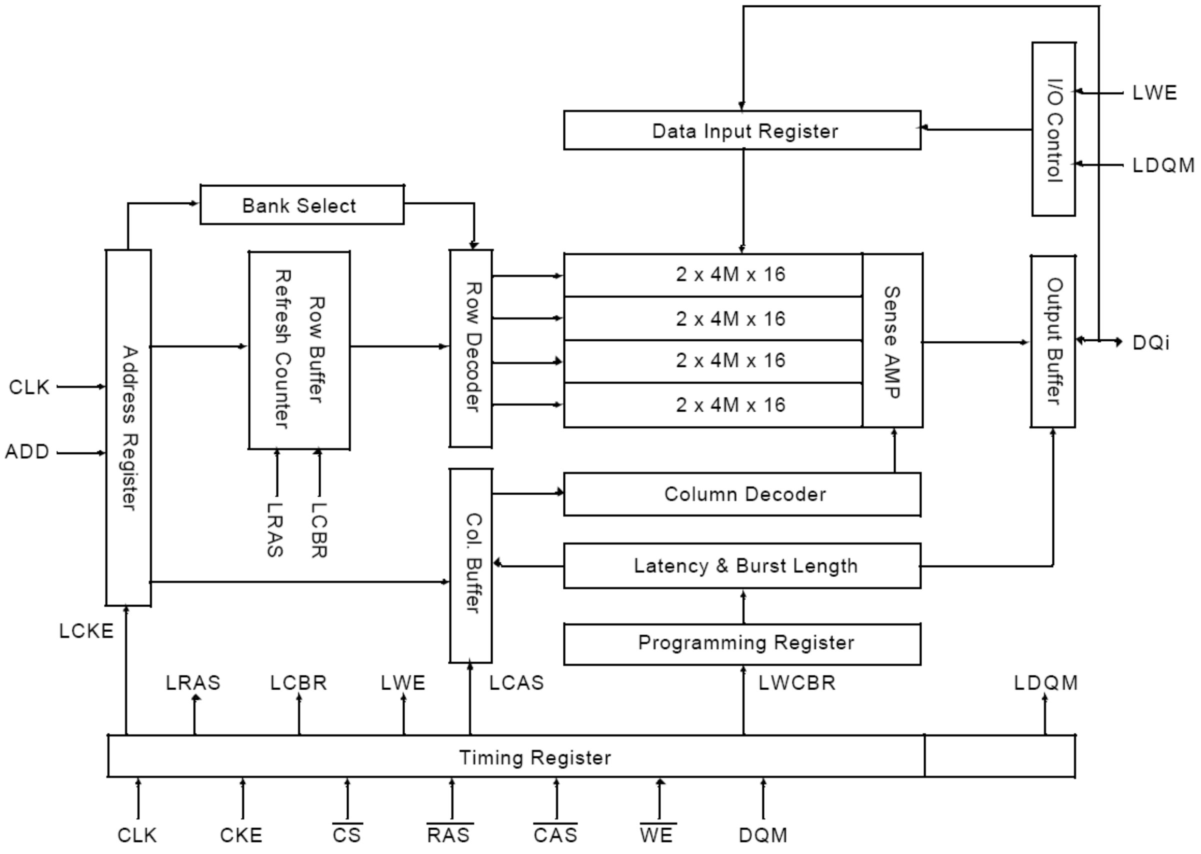


Figure 4.10: SDRAM block diagram[6]

Pin	Name	Function
CLK	System Clock	System clock
CS	Chip Select	Disables or enables device operation by masking or enabling all inputs
CKE	Clock enable	Mask system clock to freeze operation for the next clock cycle
A0-A12	Address	4 Row/column addresses are multiplexed
BA0-BA1	Bank select address	Selects activated bank during address latch time
RAS	Row address strobe	Latches row address
CAS	Column address strobe	Latches column address
WE	Write enable	Enables write operation and row precharge. Latches data in starting from CAS, WE active
DQM	Data I/O mask	Makes data output Z after clock and masks the output
DQ0-DQ15	Data I/O	Data I/O pins
VDD/VSS	Power supply/ground	Power and ground for input buffers and core logic
VDDQ/VSSQ	Data output power/ground	Power and ground for output drivers

Table 4.1: Typical SDRAM Pinout[6]

LAN Controller

The LAN Controller is connected to the system bus as shown in next sections in the figure 4.13. Since it will be necessary to receive interrupt request from the controller, we have connected the INT pin to PB30 pin of the MCU. This way the MCU may be notified that the controller needs attention.

NAND Flash Memory

As for the NAND Flash, we had a choice to use a 16-bit or an 8-bit wide data bus. Since the 8-bit chips are more common, we chose that way. The final connection is show in the figure 4.11.

FPGA

The FPGA is connected to the system bus according to the table 4.2

The FPGA also features its own SDRAM. The interconnection table is shown 4.3.

4.1.4 SPI

The typical connection of devices to the SPI bus is show in the figure 4.12. We have followed this recomendation for every device we used, only FPGA needed special attention.

MCU	FPGA		MCU	FPGA
A0	L01_N4		A20	L21_P6
A1	L01_P4		A21	L22_N6
A2	L27_P4		A22	L22_P6
A3	L30_N4		NRD	L23_N6
A4	L30_P4		NWR	L23_P6
A5	L32_N4		D0	L24_N6
A6	L40_P3		D1	L24_P6
A7	L01_N5		D2	L40_N6
A8	L01_P5		D3	L40_P6
A9	L28_N5		D4	L01_N7
A10	L28_P5		D5	L01_P7
A11	L31_N5		D6	L20_N7
A12	L31_P5		D7	L20_P7
A13	L32_N5		D8	L21_N7
A14	L32_P5		D9	L21_P7
A15	L01_N6		D10	L22_N7
A16	L01_P6		D11	L22_P7
A17	L20_N6		D12	L23_N7
A18	L20_P6		D13	L23_P7
A19	L21_N 6		D14	L24_N7
NCS0	L40_P7		D15	L24_P7

Table 4.2: Connection of FPGA to the MCU

SDRAM	FPGA		SDRAM	FPGA
A0	L01_N1		A20	L21_P6
A1	L01_P1		A21	L22_N6
A2	L28_N1		A22	L22_P6
A3	L28_P1		NRD	L23_N6
A4	L31_N1		CLKE	L23_P6
A5	L31_P1		D0	L27_N0
A6	L32_N1		D1	L27_P0
A7	L32_P1		D2	L30_N0
A8	L01_N2		D3	L30_P0
A9	L01_P2		D4	L31_N0
A10	L20_N2		D5	L31_P0
A11	L20_P2		D6	L32_N0
A12	L21_N2		D7	L32_P0
BA0	L21_P2		CLK	L40_N2
BA1	L22_N2		WE	L24_N2
DQM	L22_P2		CAS	L24_N2
CS	L23_N26		RAS	L23_P2

Table 4.3: Connection of FPGA to its SDRAM

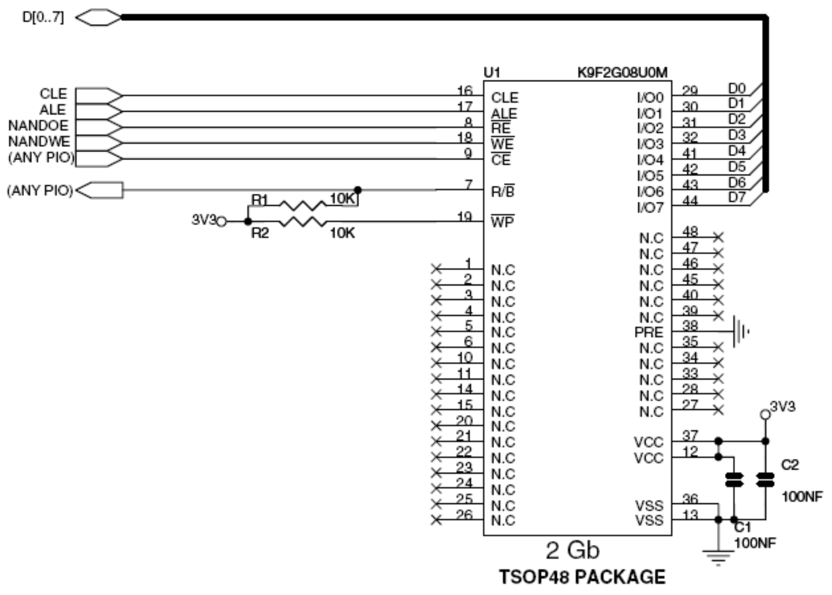


Figure 4.11: Connection of NAND Flash Memory to MCU[2]

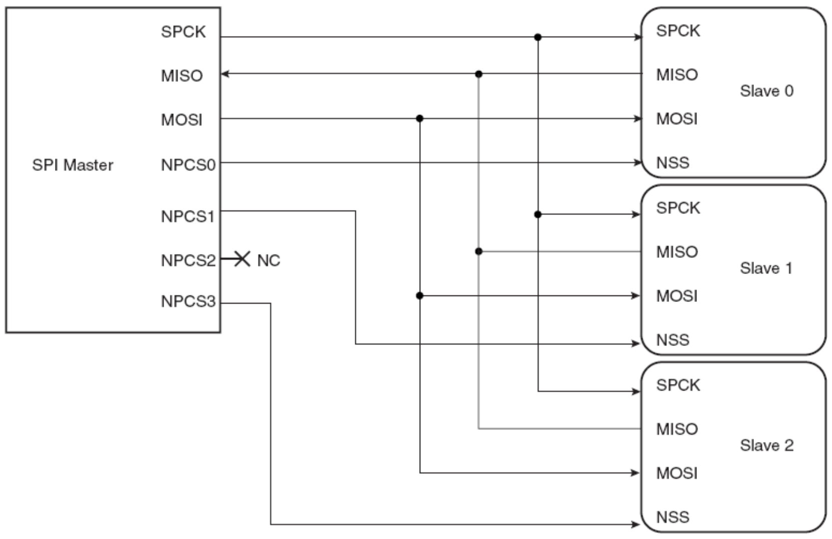


Figure 4.12: Connection of the SPI bus[2]

SPI DataFlash Memory

The SPI DataFlash memory is connected to the CS0 pin of the SPI0 interface on MCU. This further allows us to boot from this memory using the internal SAM bootloader.

FPGA Programming

The FPGA supports various programming interface as shown in the table 4.4. We have chosen to use the slave serial interface. This interface is compatible with SPI. We have to ommit the CS signal as this is not supported by the FPGA. Instead a dedicated GPIOs pins

configuration Mode	**M0**	**M1**	**M2**	**Data Width**
Master Serial	0	0	0	1
Slave Serial	1	1	1	1
Master Parallel	1	1	0	8
Slave Parallel	0	1	1	8
JTAG	1	0	1	1

Table 4.4: FPGA Configuration

has to be connected the the INIT_B, PROG_B and DONE pins. The bit clock is provide by the SPI subsystem.

To program the FPGA, a driver has to be written that will take over the SPI bus, since no other transitions on the bus are allowed while programming the FPGA. We have not found this to be very limiting, since, mostly, only one design has to be present in FPGA throughout the lifetime of an application.

Audio Codec

The audio CODEC is connected to the CS2 of the SPI0 bus of the MCU. This bus is used only for setting of the filters, volumes, sampling rate and etc. It is not used to transfer audio. To transfer audio, a seperate I2S link is connected to this CODEC. The CODEC features a standard SPI interface.

4.1.5 I2S

The CODEC supports more than one audio interface. The correct interface has to be chosen using the SPI bus prior to operation of the audio I/O.

4.1.6 LAN interface

The LAN PHY is built into the CP2200 integrated circuit. The LAN PHY requires two balun transformers for each direction of communication. These baluns are built into the connectors and greatly simplify the design of the board and schematic. This connector also provides an activity and link LED to help identify the status of the device. A recommended schematic diagram is shown in the diagram 4.13.

4.1.7 USB Host Interface

The board features two USB full speed 2.0 host ports. No additional USB PHY transceiver is necessary, since it is already build-in to help simplify the design. There is however need for some external pull-up resistors to help the host distinguish if a device is connected. The host has to provide power for the USB devices and this power is taken from a dedicated LF50CV linear regulator. The way the ports are connected is shown in the figure 4.14.

4.1.8 RS232 Converter

The RS232 converter is built on a MAX3232 integrated circuit. The IC can operate on 3.3V supply and supports 3.3V logic that is used by the MCU. As for the charge pump

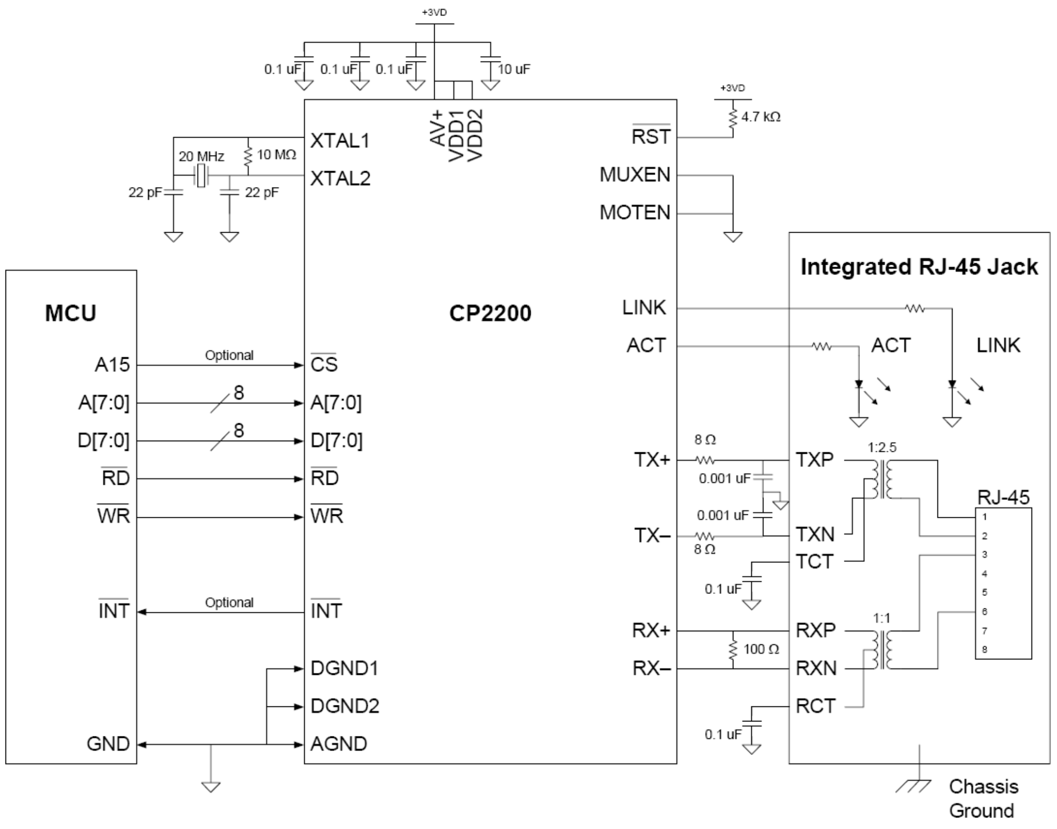


Figure 4.13: Typical connection of the CP2200[10]

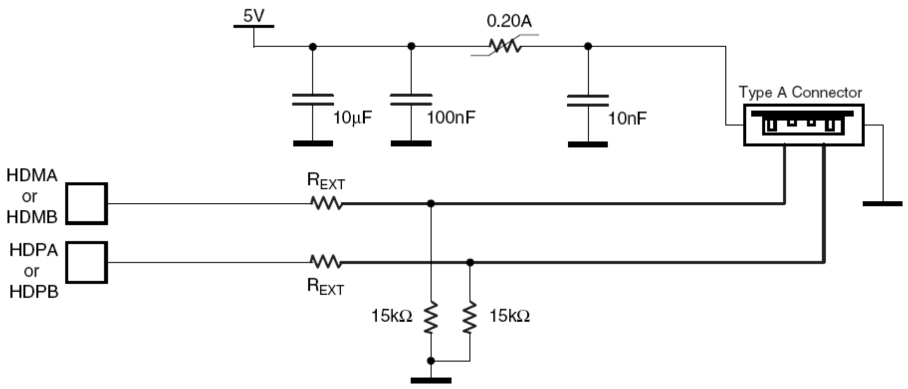


Figure 4.14: Connecting USB Host Ports[2]

capacitors, the values are provide by the datasheet [4]:

$$C_1 = 0.1\mu F$$
$$C_2 = C_3 = C_4 = 0.1\mu F$$

A typical connection is shown in the figure 4.15.

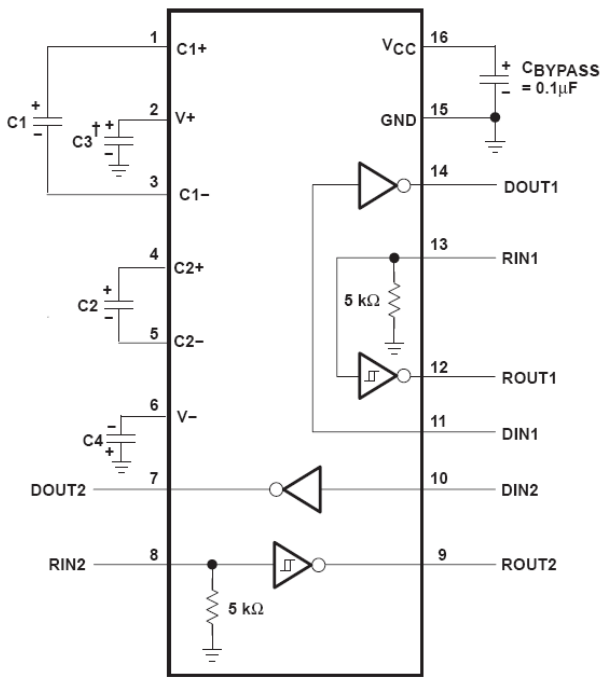


Figure 4.15: Connecting MAX3232[4]

4.2 Board

The board was autogenerated from the schematic that was draw in the Eagle. We chose to use a 4-layer PCB to be able to route the design.

4.2.1 Creating New Parts in Eagle

Before we were able to place the parts onto a board, we had to draw them and put them into the library. Many parts that we use can be found finished on the Internet using a search engine, so we could focus onto the parts that are not so frequently used or whose packages are not published in libraries.

Typically, we begin with drawing a symbol using an interface that is shown in the figure 4.16. Onto this symbol we place pins. Optionally we may specify direction, type and length of the pins. The direction may come handy later, when we want to do an ERC, however this is not entirelly bullet-proof because the Eagle does not allow to specify very detailed constrains onto the pin. For example, the pin may be input, but must not be connected, because it features an internal pullup. In this case, it will be shown as a warning or even as an error. We should assign correct names to the pins to speed up connection later and to make the schematic more clear.

To be able to see a name and a value of a component in the schematic, we have do write a text $>NAME$ onto the Name Layer and $>VALUE$ onto the Value Layer, respectively. The Eagle will interpret these substitutions and replace the with component name and value.

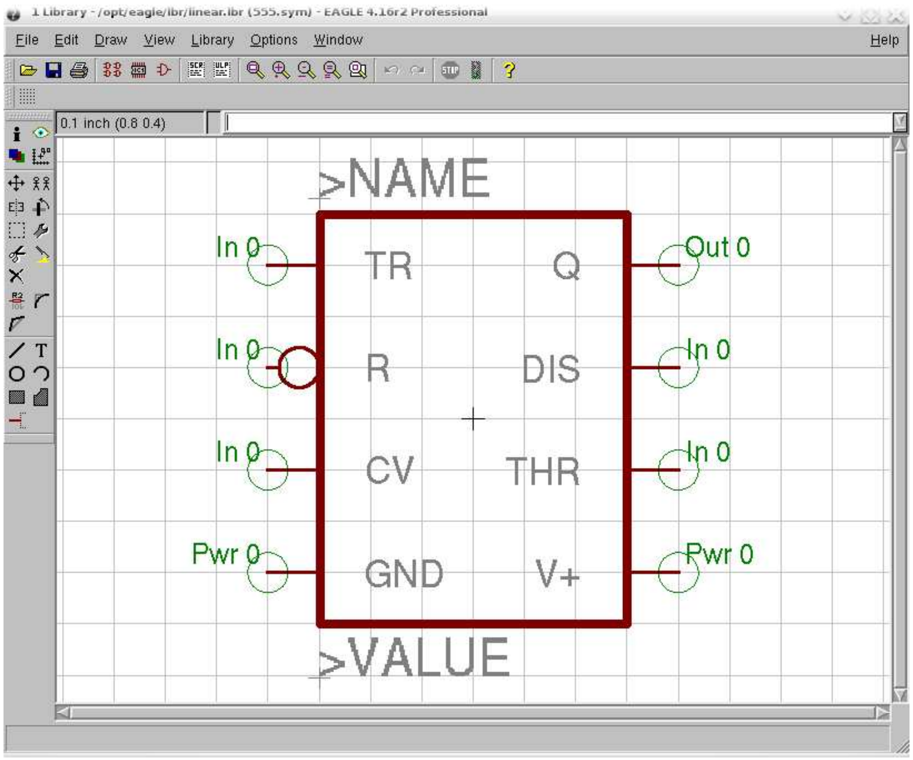

1 Library - /opt/eagle/lbr/linear.lbr (555.sym) - EAGLE 4.16r2 Professional
File Edit Draw View Library Options Window Help
0.1 inch (0.8 0.4)
>NAME
In 0
TR
Q
Out 0
In 0
R
DIS
In 0
In 0
CV
THR
In 0
Pwr 0
GND
V+
Pwr 0
>VALUE

Figure 4.16: Eagle's Interface To Draw a Symbol

Once we are finished with a symbol, we may proceed to drawing a package. The package is drawn in a separate interface as shown in the figure 4.17. We may start with drawing the outline and proceeding to the pads. Every datasheet of a component contains precise footprint of every package the device is shipped in. To be able to see the component's name and value on a board, we should write $> NAME$ and $> VALUE$ here as we also have for the symbol. We also should note the name of the pins or, prefferably, change them to pin number to greatly simplify connection later.

When the package and symbol are ready, we may create a new device. We start with specifying the device's name. Now we have to tell which symbol to use for that particular device by adding a symbol that we have drawn previously. Once this is done, we have to choose a package for that device. Once device may have more packages, we will be able to choose between them later during design.

For Eagle to know which symbol's pin to connect with which package's pad, we have a connect tool available. Example is shown in the figure 4.18. Every pad should be connected to some pin, otherwise we may not be able to use the device in a design.

Once we are finished and everything is correct, we will be present with a green tick mark next to a package as is shown in the figure 4.19.

4.2.2 Components' Placement

Once we are ready with the schematic, we may let Eagle create a board for us as shown in the figure 4.22. Eagle does not contain an autoplacement tool, so we are left to do this

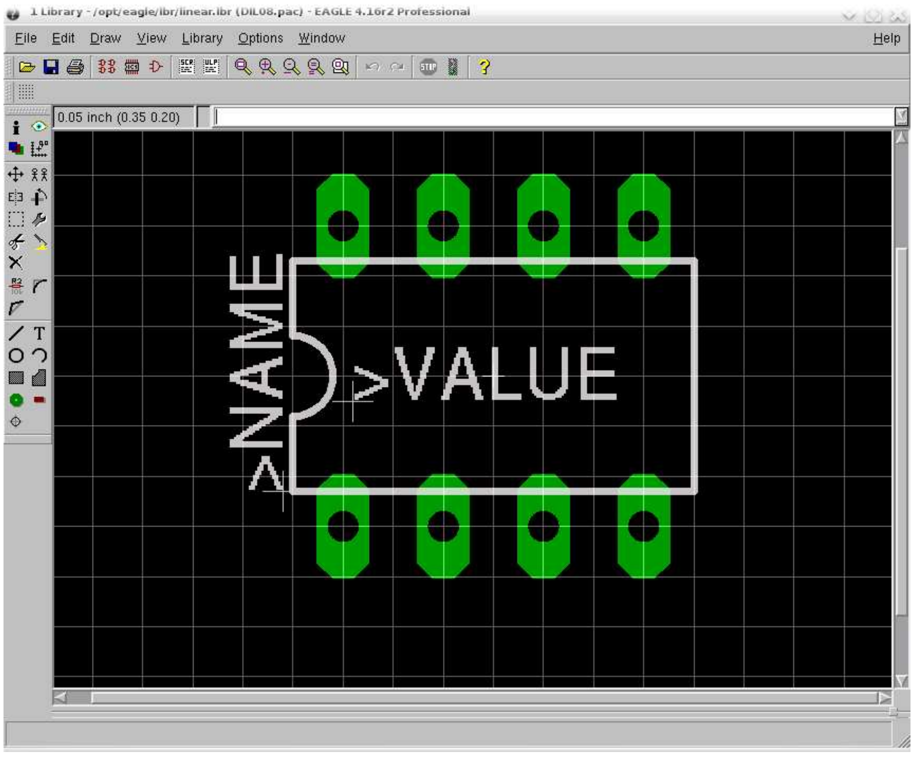
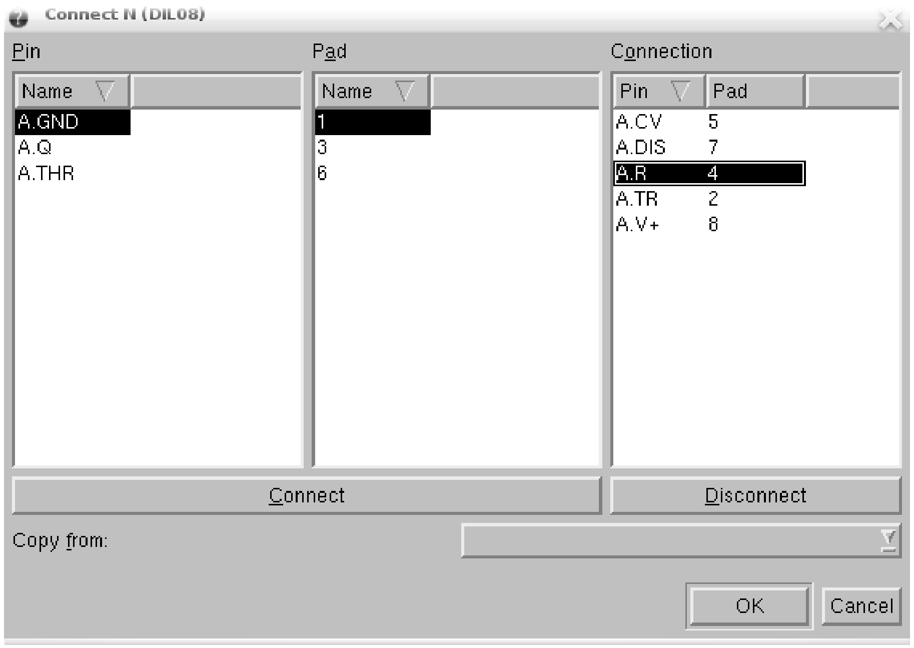

1 Library - /opt/eagle/lbr/linear.lbr (DIL08.pac) - EAGLE 4.16r2 Professional
File Edit Draw View Library Options Window Help
0.05 inch (0.35 0.20)
>NAME
>VALUE

Figure 4.17: Package of *555



Pin Name	Pad Name	Connection Pin	Connection Pad
A.GND	1	A.CV	5
A.Q	3	A.DIS	7
A.THR	6	A.R	4
		A.TR	2
		A.V+	8

Figure 4.18: Connecting the pins with pads

manually. Although, this is a very tedious work, because we have a lot of small components,

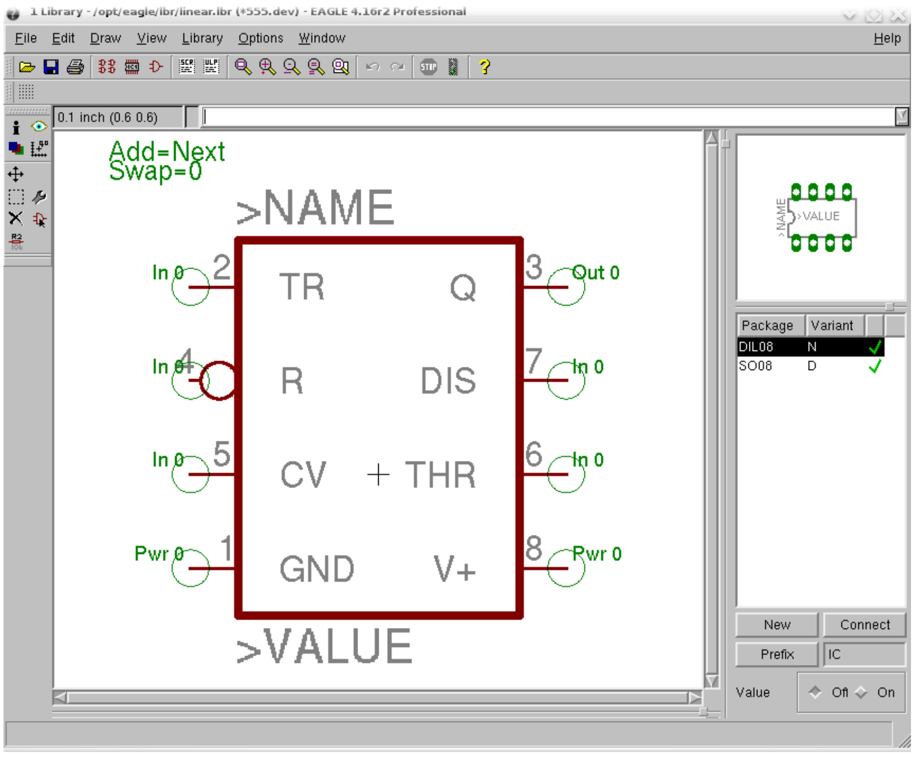


Figure 4.19: Finished *555 device

this work is important because it has a direct influence onto routability of the board and price - bigger boards cost more. We decided to place all important big parts onto top side of the board and left small discrete components to be placed onto the bottom side.

4.2.3 Routing

Once we are finished with the placement of the components, we start with routing. The Eagle's autorouter is not one of the best available in the market, so we were forced to do manual routing for the BGA. During the routing, we had to keep some constrains that are specific for every PCB manufacturer. We have chosen to use PragoBoard s.r.o in Prague. The constrains were:

- Minimum drill 0.3 mm
- Wire/Space distance 150μm
- Wire width 150μm

We have found out that using these constrains, we are not able to place anything between the balls of the BGA package, neither a via nor a wire. After a little negotiation with the PCB manufacturer, we have found out that they are able to produce wire thickness down to 100μm and with wire/space distance down to 100μm. As it turned out 125μm for wire thickness and wire/space distance was enough to place a wire in between the pads. We did not want to do the whole bga using this wire thickness, so we did the manual routing using

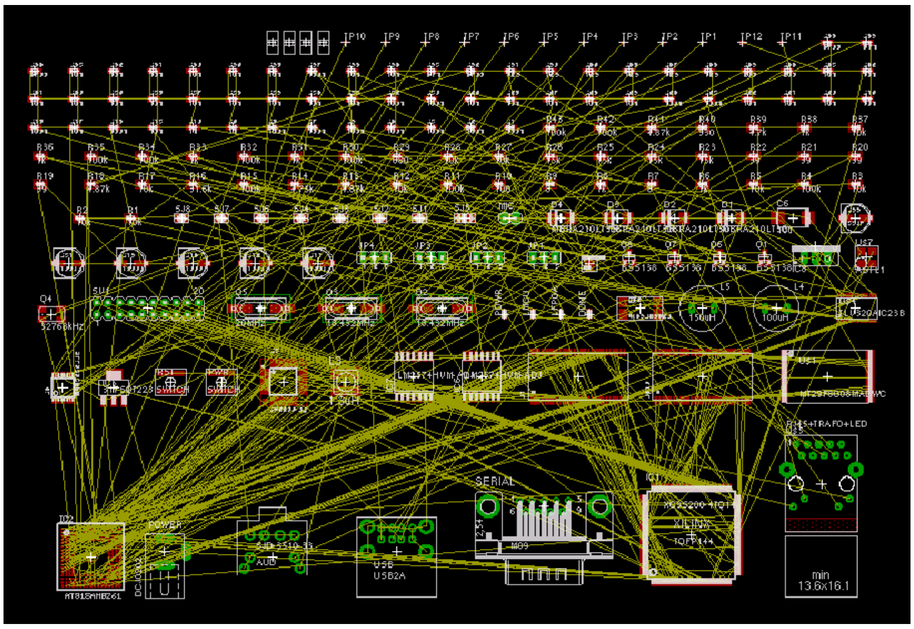
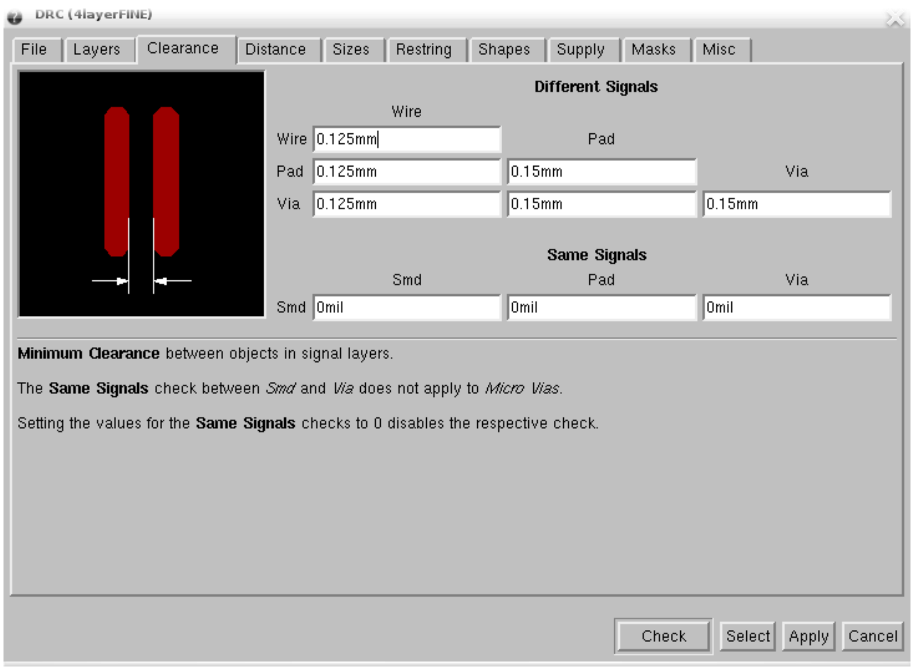


Figure 4.20: Board with Components not in place



Different Signals

	Wire	Pad	Via
Wire	0.125mm		
Pad	0.125mm	0.15mm	
Via	0.125mm	0.15mm	0.15mm

Same Signals

	Smd	Pad	Via
Smd	0mil	0mil	0mil

Minimum Clearance between objects in signal layers.

The **Same Signals** check between *Smd* and *Via* does not apply to *Micro Vias*.

Setting the values for the **Same Signals** checks to 0 disables the respective check.

Figure 4.21: Configuring the DRC

this thickness and let autorouter connect to these thin wires with normal thickness. The manually routed BGA may be seen in the figure 4.23.

A crucial tool to do routing is Design Rule Check (DRC), this tool check all the constrains that are necessary to keep to produce a good quality PCB (figure 4.21). These

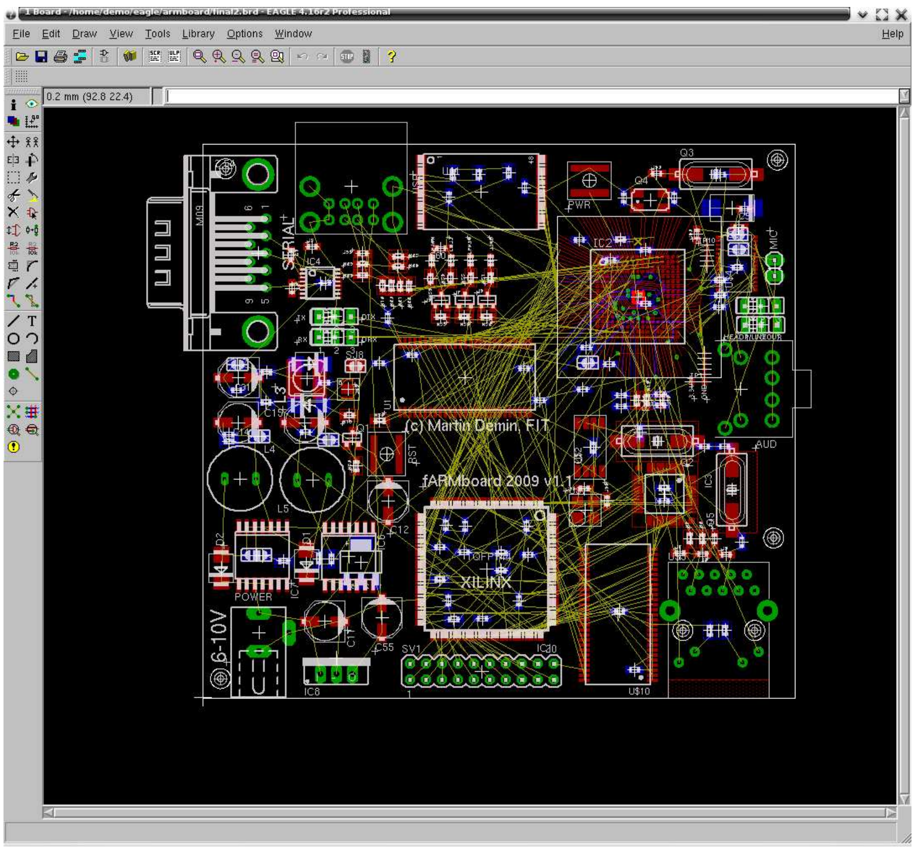

1 Board - /home/demo/eagle/armboard/final2.brd - EAGLE 4.16r2 Professional
File Edit Draw View Tools Library Options Window Help
0.2 mm (92.8 22.4)
(c) Martin Demin, FIT
fARMboard 2009 v1.1
XILINX
SERIAL
POWER
6-10V
PWR
AUD
MIC

Figure 4.22: Components are connected only using air wires

constrains are also kept by the autorouter and during manual routing we may run a DRC to see the problems. These checks include:

- Minimum drill
- Wire width
- Pad-pad, pad-via, pad-wire, wire-via, wire-wire and via-via distance
- Minimal distance from board side
- And more

To utilize this DRC properly, we have created 3 sets of the rules

1. Fine - used under the BGA
2. Normal - used globally on the board
3. Fill - used to spill copper

The fine setting was set to keep distance wire-wire 0.125mm and wire-pad and wire-via 0.15mm. The normal setting was 0.15mm for every clearence. We chose the fill setting to avoid wires, pads, vias by 0.5mm and board edge by 3mm.

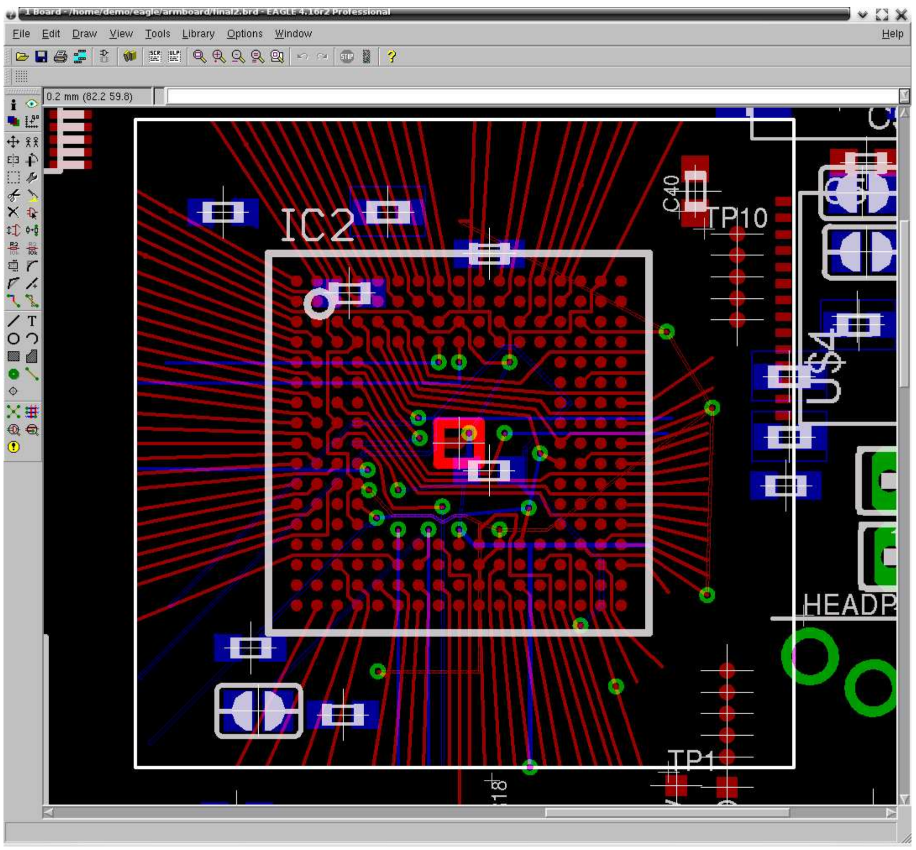


Figure 4.23: We were forced to manually route the BGA

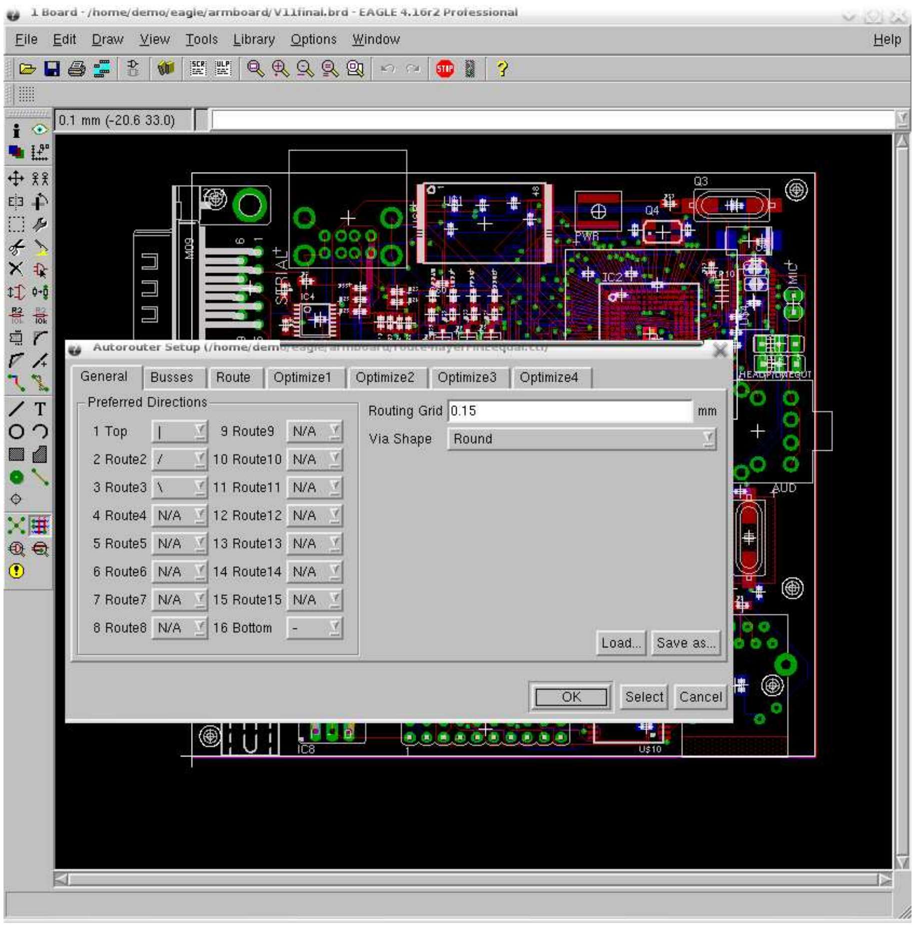
As for the autorouter, before we are able to use it, we have to specify the the way it should route. The setting for the router was:

- Layer Top - direction |
- Layer Route 2 - direction /
- Layer Route 3 - direction \
- Layer Bottom - direction -
- Routing Grid - 0.15mm
- Via shape - round

Next we set all layers to have an equal cost for all optimizations. The whole process of routing took some considerate amount because of the board complexity. The setting of the autorouter is shown in the figure 4.24.

4.2.4 Production Output from Eagle

According to the PCB manufacturer's website the following files are required


1 Board - /home/demo/eagle/armboard/V11final.brd - EAGLE 4.16r2 Professional
File Edit Draw View Tools Library Options Window Help
0.1 mm (-20.6 33.0)
Autorouter Setup
General Busses Route Optimize1 Optimize2 Optimize3 Optimize4
Load... Save as...
OK Select Cancel

Preferred Directions

1 Top	|	9 Route9	N/A
2 Route2	/	10 Route10	N/A
3 Route3	\	11 Route11	N/A
4 Route4	N/A	12 Route12	N/A
5 Route5	N/A	13 Route13	N/A
6 Route6	N/A	14 Route14	N/A
7 Route7	N/A	15 Route15	N/A
8 Route8	N/A	16 Bottom	-

Routing Grid 0.15 mm

Via Shape Round

Figure 4.24: Setting of the Autorouter

FileName	description	type
top.gbr	top layer copper	gerber
bot.grb	bottom layer copper	gerber
smt.gbr	solder mask top	gerber
smb.gbr	solder mask bottom	gerber
plt.gbr	print top	gerber
plb.gbr	print bottom	gerber
in2.gbr	internal copper layer to top	gerber
in3.gbr	internal copper layer to bottom	gerber
mil.gbr	Data to cut the board edge	gerber
pth.exc	metalic drill	exceltron
npth.exc	non-metalic drill	exceltron

The manufacturer also provides a CAM script file to produce these files, but only for 2-layer board. The in2.gbr and in3.gbr files had to be added into the script file to produce complete output. These files were packed and sent via e-mail for production. Estimated production time is 10 days.

4.3 Populating the board

The whole board was manually populated. For SMD parts we have used Pb-free solder in paste. For through-hole mounted parts, like connectors, inductors etc. we have used Pb wire solder.

We started to populate the board with the MCU as this was the most complex package. We have applied enough flux onto all BGA pads, placed the chip very precisely into place and carefully put it into a grill. To know the temperature, a drop of Pb-free solder paste was placed into a place where no vias or pads were present. This drop was closely monitored throughout the heating process and once it has melted, the grill was opened and shutdown. The board was let to cool for a few minutes.

As a next step, we started to place all the parts necessary for the DC/DC converter to place, this included the IC, capacitors, resistors, power connector and transistors. This step was important, because we need to check if the provided voltage is correct before powering the MCU. This MCU is not powered as long as the solder bridges for every voltage are not joined. In this step we have found out that some DC/DC convertors needed some load for correct regulation, especially both 1.2V regulators. Once this was fixed the power supplies were operating correctly and the solder bridges could be connect.

Next we placed the MAX3232 in place along with the capacitors and D-BUS connector for serial link. We selected to use the DBGU port.

Before we could powerup the system, we had to place components for PLL filter onto the board, because also the internal boot program needs to start the PLLB. Afterwards we place the crystals for the MCU and R42 to select boot mode. We powered up the system and tried to communicate with it via serial port set to 115200,n,8.

Once we had enough software ready, we had to put SDRAM in place, this turned out to be a picky work, because it was hard to make all pins connected properly. We ended up debugging bits written and read from SDRAM and determining the unconnected pins.

Next had to modify some software and put the SPI dataflash in place. The communication was flawless and we could start flashing some bootloader into it.

Next steps were not performed and are provided only for reference

Afterwards we put the NAND in place and flash some meaningful filesystem in it. We would boot the system and start to modify the drivers to support our board. First we would start with the LEDs, we would solder them in place and modify the LED layer in Linux kernel. Next we would proceed with LAN controller and audio controller. Once we had those wourking, we would solder the FPGA on the board and try programming something into it. As a last step we would put the SDRAM for FPGA in place.

Chapter 5

Software

5.1 Applications and Develompent Tools

There are variety of software tools and application available already designed for AT91SAM9261.

5.1.1 Compilers

For compiling operating systems and applications there is a GNU toolchain available built on GCC [11] compiler. This toolchain can be downloaded pre-built also for win32 platform[13].

5.1.2 Debugging

As this an ARM processor core, there are a lot of debugger tools available. The MCU supports EmbeddedICE which allows in circuit emulation/debugging of embedded applications. The EmbeddedICE is accessible on the JTAG interface.

5.1.3 Boot Loader

As a first stage bootloader, we have chosen U-Boot[12]. This MCU is already supported and a bootloader will provide a more flexible and robust interface to allow faster development.

5.1.4 Operating System

As for the available operating systems, the well know and developed Linux is already ported to this MCU. This makes porting for this platform simpler. Most of the integrated peripherals should already be supported, the problems may arrise only from the externally connected devices like LAN controller or codec.

When it comes to different operating systems, also Windows CE (Windows Mobile) are available. These were, however, not tested, but test images for various development board are available from the Atmel website.

5.2 Theory of Operation

We will now try to draft theory of operation, from powerup to booting of custom OS.

Power Up, Stage One After powerup, the boot ROM is executed. This ROM is mapped onto the 0x00000000 address. In this ROM is a simple SAM bootloader that allows us to upload some code into memory and execute it. This feature is used by the SAM Boot Assistant(SAM-BA) refered to in section 5.3.2.

This initial bootloader also allows to download some bigger program from an SPI dataFlash into SRAM (the AT91SAM9261 has up to 160kbytes of SRAM) and execute it. This feature is used in production.

Stage Two - AT91bootstrap In this stage an at91bootstrap is loaded from dataFlash by the initial ROM bootloader. This bootstrap is further analyzed in section 5.3.3. This tool is responsible for setting up SDRAM and finding next boot program.

Stage Three - U-Boot U-Boot is a fairly complex tool and therefore can not fit into SRAM directly and has to be loaded into SDRAM. This tool allows to access jffs2 or vfat filesystems on USB mass storage, NAND memories or even on dataFlash. Further can access network and provide network boot. We use it to load Linux kernel from dataFlash to SDRAM and boot it. This tool is further analyzed in section 5.3.4.

Stage Four - OS The OS is starting in this stage. Modules are loaded for the hardware by the kernel and finally an user application is loaded. For details in building the OS see section 5.3.5.

5.3 Building and using Tools

5.3.1 OpenEmbedded

As a build system, we have used Gentoo linux. First we have to obtail Bitbake. We have decided to use a stable release as opposed to a git release.

```
$ emerge =bitbake-1.8.12
```

Once we have Bitbake, we create a basic directory structure for our OE environment.

```
$ mkdir -p /OE/build/conf
$ cd /OE/
```

Next we need to optain local copy of openembedded using git:

```
$ git clone git://git.openembedded.org/openembedded
```

Once we have OE pulled from the server, we may create our configuration file for example.

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

Some lines have to be modified using an editor:

```
BBFILES = ,,/stuff/openembedded/recipes/*/*.bb‘‘
DISTRO = ,,angstrom-2008.1‘‘
MACHINE = ,,farmboard‘‘
```

In order for OE to support our machine, we had to create a config from an existing one:

```
cp /OE/openembedded/conf/machine/at91sam9261ek.conf
   /OE/openembedded/conf/machine/farmboard.conf
```

So far we don't need to do any modifications in it. Setting the environment:

```
$ export BBPATH=/OE/build:/OE/openembedded
```

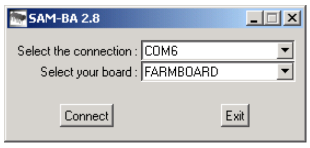
Now we may start building some package because this process will take long time, this build system has to compile toolchain for our processor, including gcc and libraries.

```
$ bitbake at91bootstrap
```

5.3.2 SAM Boot Assistant

SAM Boot Assistant (SAM-BA) is a specialized tool for working with SAM series of AT91 microcontrollers. This tool allows working with all memories that are connected to such an MCU. To accomplish this, applets are used. These applets are pieces of code that are uploaded into SRAM or SDRAM and executed.

After we start the application, we have to select a communication and type of board as can be seen in the figure 5.1. To be able to select our custom board, we had to copy



Select the connection : COM6
Select your board : FARMBOARD

Figure 5.1: select

directory lib/AT91SAM9261EK to lib/FARMBOARD and compile an applet that needed modification. Since our board has only 16-bit wide data bus, we had to create board directory by copying applets/at91lib/boards/at91sam9261-ek to farmboard. The file that has to be modified is board_memories.c as follows on the marked line:

```
void BOARD_ConfigureSdram()
{
        WRITE(AT91C_BASE_SDRAMC, SDRAMC_CR, AT91C_SDRAMC_NC_10
                           | AT91C_SDRAMC_NR_13
                           | AT91C_SDRAMC_CAS_2
                           | AT91C_SDRAMC_NB_4_BANKS
---->                      | AT91C_SDRAMC_DBW_16_BITS
                           | AT91C_SDRAMC_TWR_2
                           | AT91C_SDRAMC_TRC_7
                           | AT91C_SDRAMC_TRP_2
                           | AT91C_SDRAMC_TRCD_2
                           | AT91C_SDRAMC_TRAS_5
                           | AT91C_SDRAMC_TXSR_8);
...
}
```

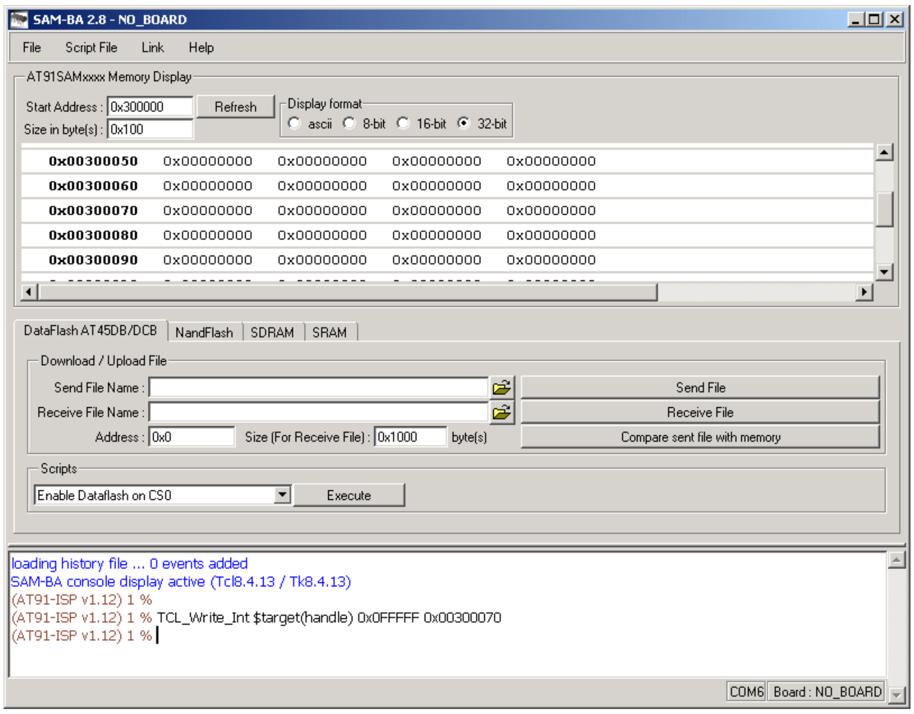
to compile the correct applet, we do (provided we have the correct toolchain in path):

```
$ cd applets/isp-applets/extram/
$ make CHIP=at91sam9261 BOARD=farmboard MEMORY=sram_samba DYN_TRACES=1 clean all
```

Next we have to copy the applet into correct directory:

```
$ cp bin/isp-extram-at91sam9261.bin ../../../lib/FARMBOARD/
```

Now we may start to use the SAM Boot Assistant. The interface will look as shown in the figure 5.2. Using this tool we may flash AT91bootstrap and U-Boot into the dataFlash



Start Address : 0x300000
Size in byte(s) : 0x100
Display format: ascii 8-bit 16-bit [✓ 32-bit]

```
0x00300050   0x00000000   0x00000000   0x00000000   0x00000000
0x00300060   0x00000000   0x00000000   0x00000000   0x00000000
0x00300070   0x00000000   0x00000000   0x00000000   0x00000000
0x00300080   0x00000000   0x00000000   0x00000000   0x00000000
0x00300090   0x00000000   0x00000000   0x00000000   0x00000000
```

Send File Name :
Receive File Name :
Address : 0x0
Size (For Receive File) : 0x1000 byte(s)
Scripts: Enable Dataflash on CS0

```
loading history file ... 0 events added
SAM-BA console display active (Tcl8.4.13 / Tk8.4.13)
(AT91-ISP v1.12) 1 %
(AT91-ISP v1.12) 1 % TCL_Write_Int $target(handle) 0x0FFFFF 0x00300070
(AT91-ISP v1.12) 1 %
```

Figure 5.2: samba

memory.

5.3.3 AT91bootstrap

We found best to use OE to build this tool by running:

```
bitbake -b /OE/openembedded/recipes/at91bootstrap/at91bootstrap_2.11.bb
 -DDDD -c configure
```

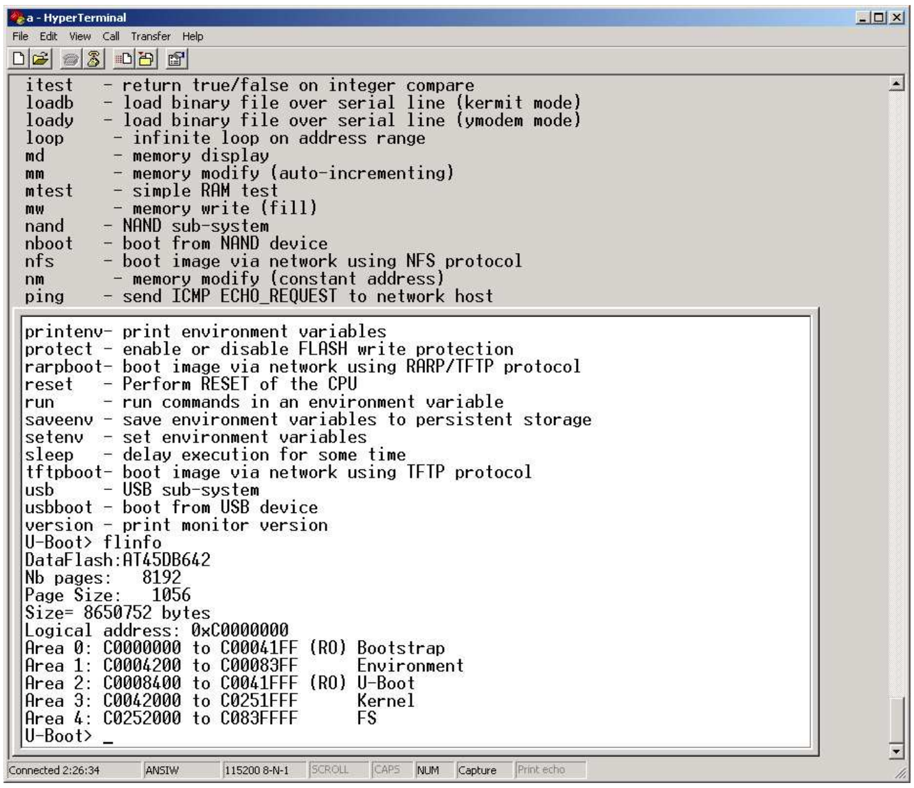
This way we get a source in the /OE/tmp/work/farmboard-angstrom-linux-gnueabi/at91bootstrap-2.11-r0/ directory and we have to modify file board/at91sam9261ek/at91sam9261ek.c to initialize only 16 bits of the SDRAM bus. Once we were finished we may compile the tool:

```
bitbake -b /OE/openembedded/recipes/at91bootstrap/at91bootstrap_2.11.bb
 -DDDD -c compile
```

The binary file may be found in the binaries directory. The at91sam9261ek-dataflashboot-2.11-r0.bin.fixboot file si to be flashed into dataflash at address 0x0.

5.3.4 Das U-Boot

Das U-Boot is a very portable bootloader that has many drivers taken from linux kernel, thus supports a lot of devices. We did not compile our own U-Boot, but have used one provide by a demonstration package intended for use with AT91SAM9261 Evaluation Kit. The U-Boot should be flashed at address 0x8400 into dataFlash and will be loaded into SDRAM uppon startup. Once U-Boot is running, it will communicate with us using DBGU serial port. On the host we may use for example a Hyperterminal configured for 115200,n,8 5.3. The U-Boot is presented with a command prompt where the user may issue various



```
itest    - return true/false on integer compare
loadb    - load binary file over serial line (kermit mode)
loady    - load binary file over serial line (ymodem mode)
loop      - infinite loop on address range
md        - memory display
mm        - memory modify (auto-incrementing)
mtest     - simple RAM test
mw        - memory write (fill)
nand     - NAND sub-system
nboot    - boot from NAND device
nfs      - boot image via network using NFS protocol
nm        - memory modify (constant address)
ping     - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset    - Perform RESET of the CPU
run      - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sleep   - delay execution for some time
tftpboot- boot image via network using TFTP protocol
usb      - USB sub-system
usbboot - boot from USB device
version - print monitor version
U-Boot> flinfo
DataFlash:AT45DB642
Nb pages:   8192
Page Size:   1056
Size= 8650752 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C00041FF (RO) Bootstrap
Area 1: C0004200 to C00083FF      Environment
Area 2: C0008400 to C0041FFF (RO) U-Boot
Area 3: C0042000 to C0251FFF      Kernel
Area 4: C0252000 to C083FFFF      FS
U-Boot> _
```

Figure 5.3: hyperterm

commands. Once we have U-Boot running, we may start obtaining an image for it to load.

5.3.5 Linux Kernel

A working Linux Kernel image was obtained from demonstration package intended for use with AT91SAM9261 Evaluation Kit. This was intended for rapid development, but will have to be customly compiled and modified, because of the differences that exist between this board and the evaluation kit.

To flash the kernel, we have to first load it into SDRAM by calling:

```
loady 22000000
```

After calling this command, u-boot is accepting Ymodem upload on the serial port. Once this is finished, we have a kernel image on the address 0x22000000 in the SDRAM. Now we have to flash this image into dataFlash at address 0xC0042000 by running:

```
cp.b 22000000 C0042000 <size_of_image>
```

Next modify the bootcmd environment variable by running:

```
setenv bootcmd 'cp.b C0042000 22200000 210000; bootm 22200000'
```

The boot arguments for the kernel may be set using:

```
setenv bootargs 'console=ttyS0 root=/dev/mtdblock0
 mtdparts=AT45DB642x-nand:6072k@252000 rootfstype=jffs2'
```

Finally we may save the environment:

```
saveenv
```

And reboot using the button.

Chapter 6

Conclusion

We have found the PCB design to be a very complex task. It is almost impossible for one person to avoid doing a single mistake. In the first version of the board, we have forgotten to connect Boot Mode Select pin which resulted in system being unbootable, although we tried to repair the board and also to reball the BGA, we ended up replacing both in a very limited time. In the end not leaving us enough time to finish porting an OS and drivers onto the board.

Even in the second version of hardware we have found many flaws that were left unseen in the first version. All of them are listed in the chapter Errata. Some were easily fixable, some harder and some impossible. However, we have learned a lot about the soldering of BGAs and routing of complex 4-layer boards.

As for the chosen CAD software Eagle, we would say that it is barely sufficient for boards with such a complexity. Many times be were left with few unconnected signals and we were forced to do some manual routing and moving of the components around. Modern CAD softwares specifically designed for PCB further allow component autoplacement and we may set constrains for individual signals. These constrains may include length of wires - keeping all signals of a bus within the same length or keeping differential signals next to each other to minimize EMI. Also, for Eagle, it would be a big improvement if the autorouter could utilize multiple processors by spawning more threads with the autorouter. However, this was all designed in the 4.16r2 version of the Eagle and there are many never releases available in which some of the features could already be present.

For future, we have found out that a faster 400MHz ARM processor is available from Atmel, it may have very similar package, so this design may be modified and corrected to support a faster processor. Also more precise fabrication process may allow to route all BGA pads outside and make the board more universal as more GPIOs would be available.

Bibliography

[1] Atmel Corporation. At45db642 datasheet.
`http://www.gaw.ru/pdf/Atmel/AT45/at45db642.pdf` .

[2] Atmel Corporation. At91 arm thumb-based microcontrollers at91sam9261.
`http://www.atmel.com/dyn/resources/prod_documents/doc6062.pdf` .

[3] Digikey Corporation. Digikey corporation. `http://www.digikey.com` .

[4] Texas Instruments Incorporated. Max3232 datasheet.
`http://focus.ti.com/lit/ds/symlink/max3232.pdf` .

[5] Texas Instruments Incorporated. Tlv320aic23b. `http://ti.com/` .

[6] Samsung. K4s511632d-uc75. `http://www.samsung.com/` .

[7] National Semiconductors. Lm2574 - simple switcher 0.5a.
`http://www.national.com/pf/LM/LM2574.html` .

[8] National Semiconductors. Lm2738 - 550khz/1.6mhz 1.5a step-down regulator.
`http://www.national.com/pf/LM/LM738.html` .

[9] National Semiconductors. Lm317 - 3-terminal adjustable regulator.
`http://www.national.com/pf/LM/LM317.html` .

[10] SiLabs. Cp2200/1. `https://www.silabs.com/Support Documents/TechnicalDocs/CP2200.pdf` .

[11] Open Source. Gcc, the gnu compiler collection. `http://gcc.gnu.org/` .

[12] Sourceforge. Das u-boot - universal bootloader.
`http://sourceforge.net/projects/u-boot` .

[13] Martin THOMAS. Arm-projects.
`http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/` .

[14] Unknown. Lf00 series. `http://` .

[15] FIT VUT v Brne. Fitkit. `http://merlin.fit.vutbr.cz/FITkit/` .

Appendix A

List of Appendicies

The followint Appendicies are part of this work:

1. This List
2. Errata
3. Schematic Diagram
4. Board Drawing

Appendix B

Errata

This page is provided to warn the users about the flaws that are present in the current design (both, schematic and board)

1. The IC U$8 LM2738 has an incorrect footprint on the board, the footprint is mirrored.
2. The address and data bus that are drawn separately on the schematic for the dedicated SDRAM for the FPGA are in fact, thanks to the unique feature of the Cadsoft Eagle, joined with the main bus. It is therefore impossible to place this SDRAM onto board and use it !
3. Footprint for 32768Hz crystal is incorrect, see datasheet for correct footprint.
4. CE pin on the NAND memory is not connected, should be connected to PC14 of MCU.
5. ALE and CLE pins are interchanged, should be swapped.

Appendix C

Schematic

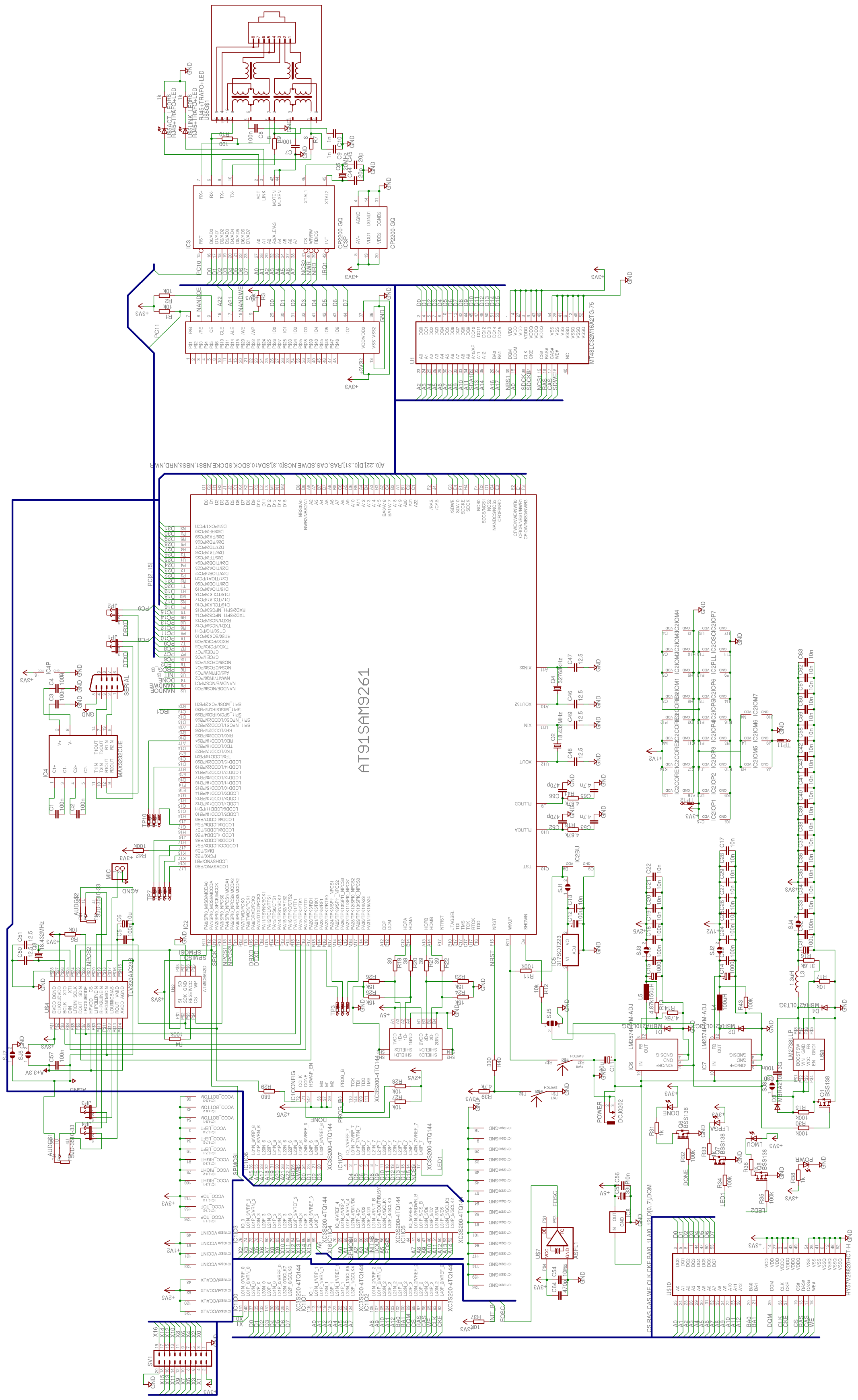


Figure C.1: Schematic Diagram

Appendix D

Board

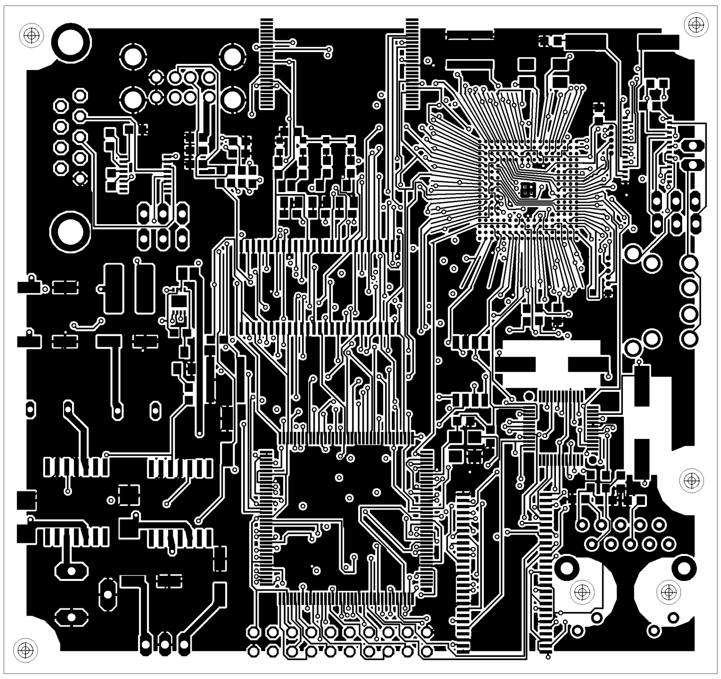


Figure D.1: Copper layer - TOP

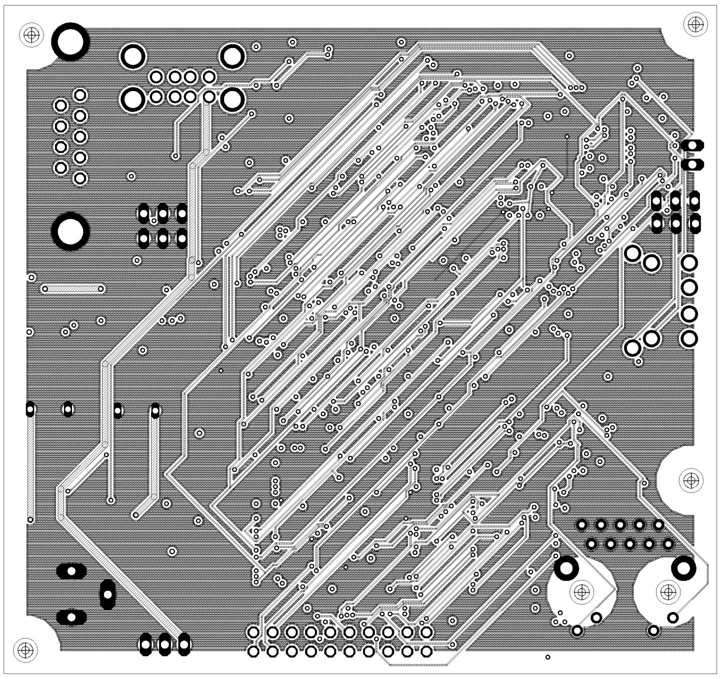


Figure D.2: Copper layer - IN2

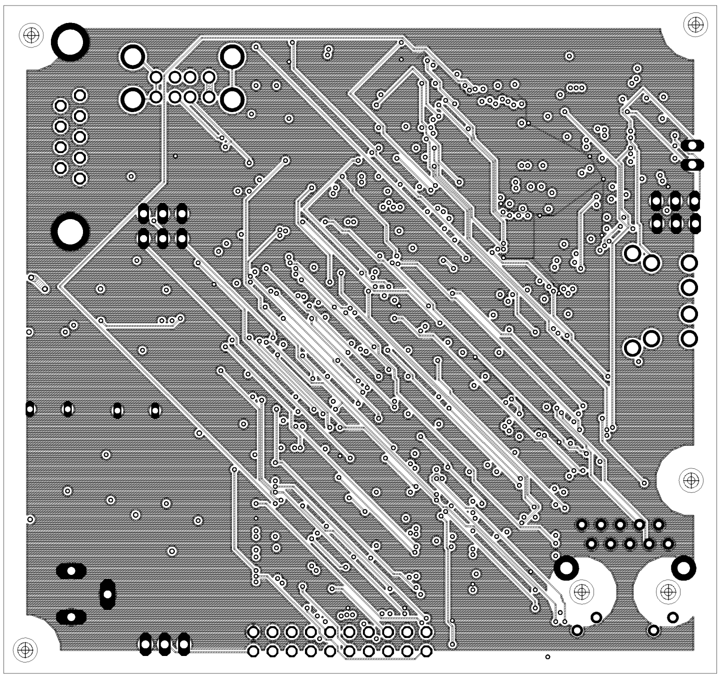


Figure D.3: Copper layer - IN3

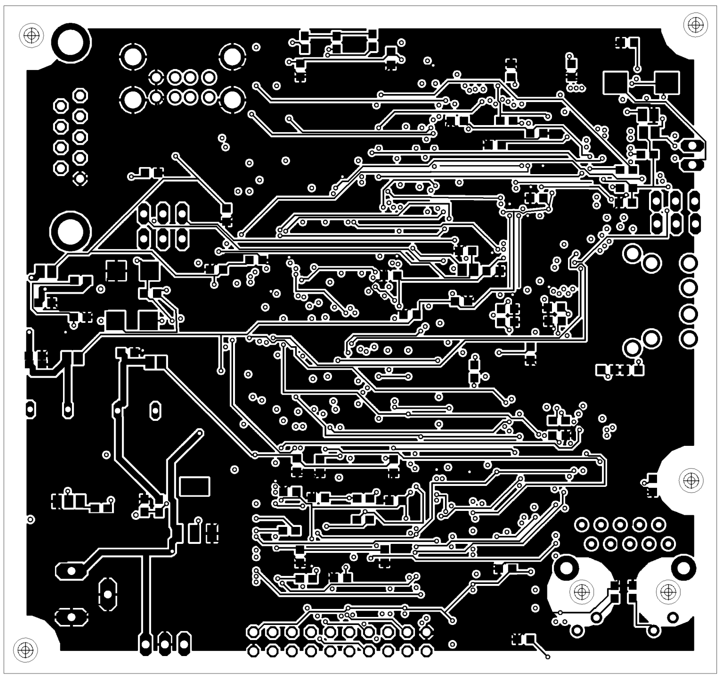


Figure D.4: Copper layer - BOTTOM

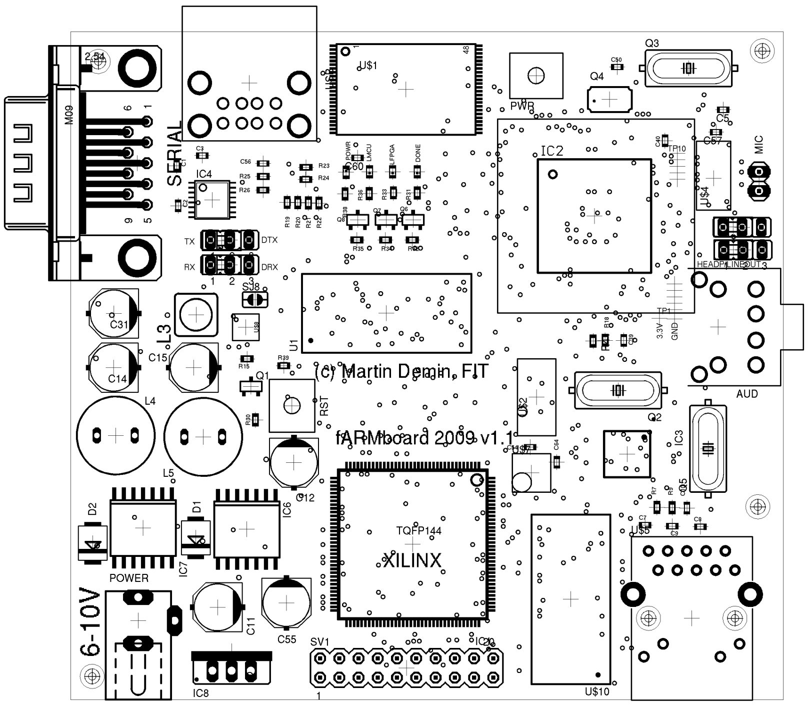


Figure D.5: Component placement - TOP

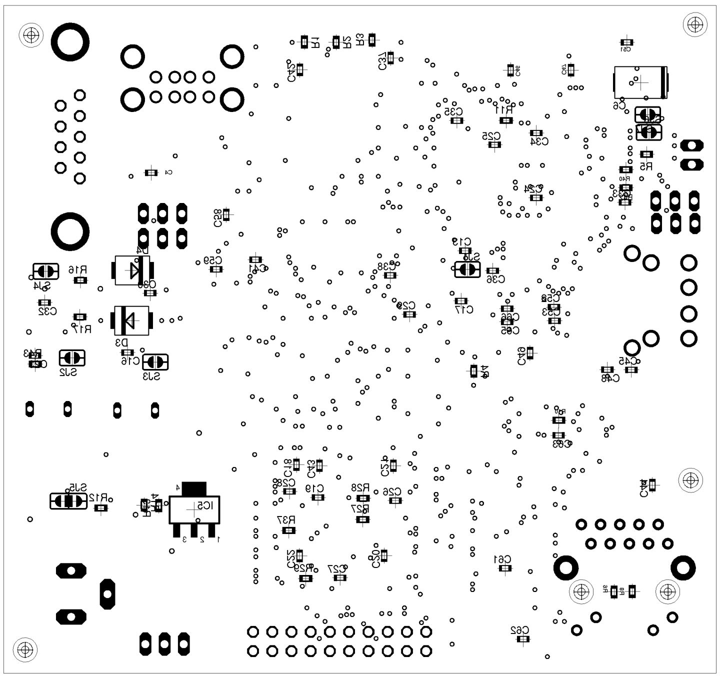


Figure D.6: Component placement - BOTTOM