Innovate Nordic is a multi-discipline engineering design contest open to all undergraduate and graduate engineering students in the Nordic region. Innovate brings together the smartest engineering students in Nordic region and the programmable logic leadership of Altera Corporation to create an environment of learning through innovation. Innovate is not only about smart design projects, it is about designing complete products that target specific markets. Students participating in Innovate are judged equally on the market value of their design as well as on the quality of execution.

2007

Many entries were received but the best projects were invited to present and demonstrate their projects at the FPGA world event in Stockholm in Sweden. This Volume contains the submitted papers from the seven finalists.

The final results were:

1st Place: ***Team Digital Stereo*** from Engineering College of Aarhus, Denmark
This project used an fpga to stream and process audio from a commercial CD drive over the IDE interface – their objective was to drive a class D amplifier and deliver a 100% digital audio system

2nd Place: ***Team Aliasing*** from Aalborg University, Denmark
The objective here was to cost reduce Software Defined Radio systems by designing a novel architecture for SSB demodulation, using multirate signal processing and FPGA.

3rd Place: ***Team Panda*** from KTH in Stockholm, Sweden
This team designed an implemented and FPGA based Voiceprint authentication System

Runners Up:

Team SO-SIG from Helsinki University of Technology, Finland
An FPGA based Biosignal measurement system
Team Kimmo Järvinen from Helsinki University of Technology, Finland
A Cryptoprocessor for Elliptic Curve Digital Signature Algorithm
Team Min Myra from Linköping University, Sweden
An autonomous robot design
Team Hervanta from Tampere University of Technology, Finland
A Leon3 Multi-processor system on an Altera FPGA

LEARNING THROUGH INNOVATION

INNOVATE NORDIC

Realize your design dreams using Altera FPGAs

Project documentation of

Digital Stereo

Peter Stensgaard Karlsen
psk@mp-tech.dk

Jens Egeløkke Frederiksen
jef@jefec.dk

Team: DigitalStereo ID: IN00000014
Associated lecturer: Stig Kalmo, Engineering College of Aarhus

2007-08-21

Abstract

This is project documentation. In here is developed a fully digital HIFI stereo set. Audio data is read from a CD utilizing a standard CD-ROM drive and routed to a custom made class D amplifier. The amplifier is based on Texas Instruments TAS5504 and TAS5142. All routing and configuration are implemented on the terasic DE1 demo board holding a Cyclone II FPGA. All configurations are using the I^2C bus, and all audio is routed using the I^2S bus. The NIOS II embedded processor is a key part of the system.

All software implemented are working, but the class D power amplifier is failing.

There are no source codes in this project report. Electrical drawings are provided.

1. Contents

2. Introduction

This project proposes a HIFI audio system where the path from the CD to the loudspeaker is fully digitized. It is submitted as a contest proposal for the Altera Innovate Nordic.

2.1 Background

Through the past 20 years storage of music has gone through a revolutionizing digitalization. Phillips and Sony started this development in 1982 when they introduced the Compact Disk (CD) media. Since then several other digital Medias has been proposed, but the most successful is the MP3 format (Fraunhofer, Phillips, CCETT and IRT) introduced around 1994. Today MP3 is used by almost everyone, and it has been a very big part of computerizing music handling for the end user.

This digitalization of the media hasn't had a great impact on HI-FI music systems. Some systems have MP3 decoders build-in, but most systems translate the digital audio into an analog signal fed to an analog amplifier. The signal path from the digital media to the loudspeaker is still mostly analog, which adds "colorization" and uncorrectable noise to the signal.

2.2 Project overview

The DE1 development (Cyclone II FPGA) board is the center of the system.

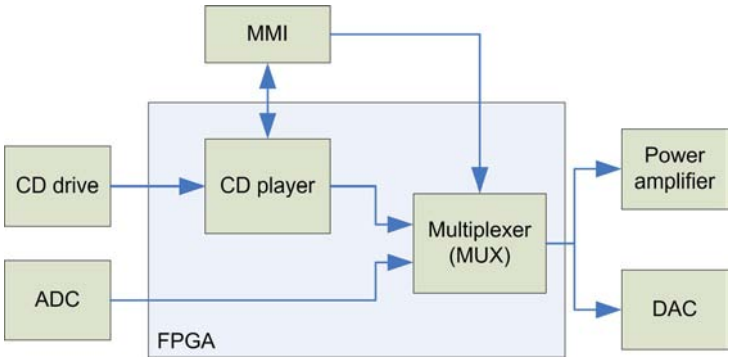


Figure 1: Simplified block diagram

Figure 1 shows a simplified block diagram over the construction developed in this project. Only the CD drive, the power amplifier and the loudspeakers are off-board modules. As CD drive is used an A-open CD-ROM drive. The CD drive is controlled by the CD player; witch retrieves musical data through the IDE interface. Audio stream from either the CD player or the ADC is routed to the output by the multiplexer. The ADC and DAC is The Wolfson WM8731 CODEC on the DE1 board. MMI is push buttons, switches, LEDs and 7-segment displays also on the DE1 board. Amplification is using class D amplifiers and is a custom made external PCB module.

A photograph of the project hard ware can be seen on Figure 2.

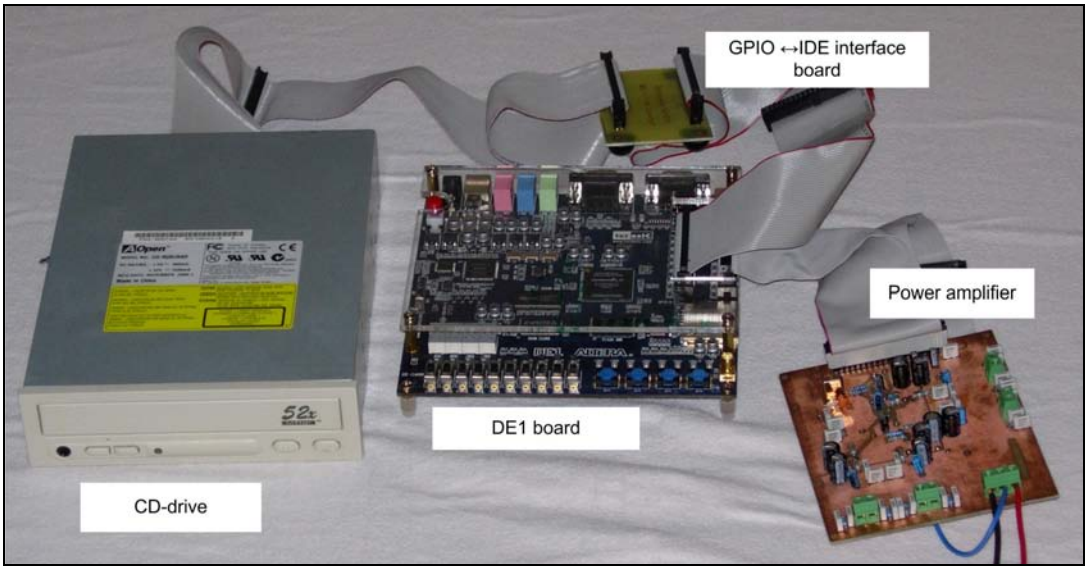


Figure 2: Photograph of prototype

In the photograph isn't shown the SMD side of the PCB. It can be seen in the chapter about the power amplifier.

The external modules are connected to the DE1 board through ribbon cables. The CD-ROM IDE interface pins have to be rerouted to fit the GPIO connector (JP1) on the DE1 board. This is why the "GPIO↔IDE interface board" is necessary.

2.3 Development responsibilities

The development work has been divided in the group. Jens has been working on the Class D amplifier hardware and the design of the audio processing system. Peter has been working on the CD-Rom interface and on retrieving the audio data from the CD.

3. CD drive interface

The purpose of the CD-Rom interface is to access the CD in the drive, read the audio data and send it to the audio processing/amplifier block in an acceptable format. It is agreed to use the I^2S format since it is widely used, and can be feed directly into the audio processor that controls the amplifiers as well as the audio codec placed on the DE1 board.

The MMI is the keys and switches on the DE1 board as a simple interface for controlling playback of the CD, and the 7 segment display to show the current playing position of the CD. For controlling and reading data from the CD-Rom drive it is decided to use the Nios II processor. The scope of this sub part of the project is to provide CD player functionality and sending digital audio data to the audio processing/amplifier block, which is another subpart of the project.

3.1 Interfacing the CD-Rom drive to the DE1 board

The IDE cable from the CD-Rom drive doesn't just plug into one of the GPIO ports on the DE1 board since the DE1 board has +3,3V and Gnd on pins that are used for signals in the CD-Rom drive. In order to solve this problem an IDE to DE1 converter was build. It is simply a small PCB with two 2x20 pin connectors on it. The connectors are interconnected in a way that makes it possible for the FPGA on the DE1 board to connect to the CD-Rom drive. The schematic of the converter can be seen on Figure 4.

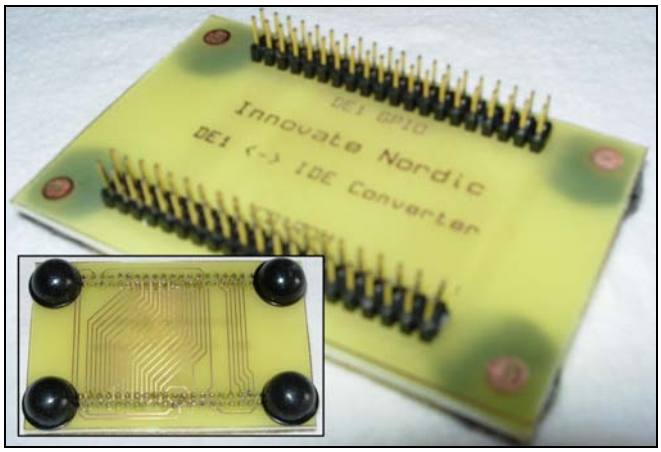


Figure 3: Photograph of IDE to DE1 converter

As it is seen in the schematic not all signal pins from the IDE interface are connected to the DE1 board. They are not needed for the type of communication we plan to have between the Nios II processor and the CD-Rom drive. The extra pins are used for DMA transfer, which we are not going to use. The interface consisting of 16 data pins, 5 address pins and 3 control pins is sufficient to communicate with the CD-Rom drive. Earlier we have had success using the same interface communicating with a Compact Flash memory card,

and since the interface also supports Hard Disc and CD-Rom access we are confident that it will work.

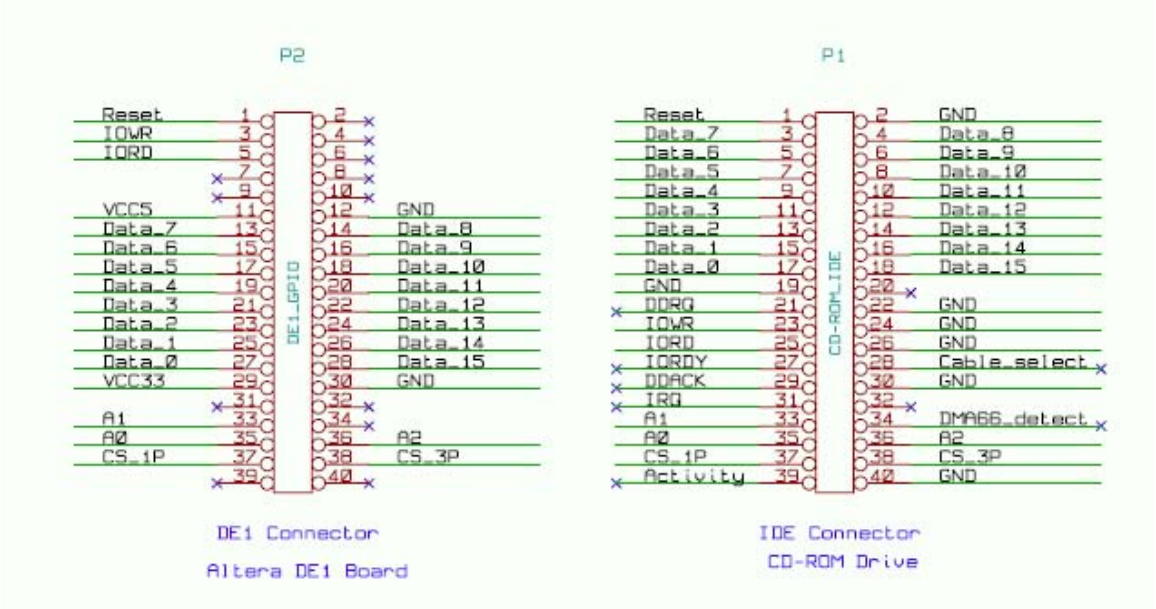


Figure 4: Schematic of IDE to DE1 converter

3.2 Nios II processor design

The design and configuration of the Nios II processor is planned to evolve as the work and design progresses. To begin with the Nios processor was designed to have a jtag uart, some sram, some sdram, a 16 bit bidirectional PIO for the data bus, a 5 bit output PIO for the address bus and a 3 bit output PIO for the control signals for CD-Rom drive. To test the Nios processor one of the software examples were used. This gave quite a bit of problems. Neither the Nios IDE nor the Altera Debug Client was able to connect to the Nios processor and download software. Over a week was spent trying to figure out what was wrong. Different configurations of the Nios system in the SOPC builder were tried without success. Finally a user forum was found on the internet, where several people had the same problem. It seems it is a bug in the SOPC builder version 7.0/7.1 when the Altera Avalon tri-state bus is used. There was no obvious solution to the problem.

As a work-around a pre-defined Nios II system found on the CD-Rom that came with the DE1 kit under the DE1 demonstrations. This system was defined in a previous version of SOPC builder (probably 6.x) so it needed an upgrade when it was loaded in SOPC builder version 7.1. This system had no problems with downloading software. An identical system was designed from scratch in SOPC builder 7.1, but the Nios IDE wasn't able to connect to this system either. The effort to build a Nios system from nothing was abandoned with the conclusion that there might be a bug in the SOPC builder. Instead the demonstration system was modified to fit to our purpose.

The modifications consist of removing the 7segment interface and the PIO port for the red LEDs and adding the PIO ports mentioned earlier. The ports for the green LEDs, switches and keys were kept. The keys were planned to be used anyway, and the LEDs will be nice to have for debugging purposes.

As the development progressed, the Nios II processor was expanded. An interval timer was added to provide interrupts for polling the keys that became the control interface for the CD player. Additional PIO ports were added for sending audio data. These PIO ports include two 24bit output busses, one for each audio channel. The CD audio is only 16 bit, but some CD's are recorded in 20 bit, and DVD's are 24 bit, so it seemed reasonable to make the system capable to handle 24 bit data. The only changes needed to support this are software. The parallel audio data is converted into I^2S format by a VHDL design outside the Nios processor. This design has a reset and an enable input as well as a ready output that toggles when the converter is ready for new data. These signals are also connected to the Nios processor through PIO ports. The ready PIO port has an edge triggered interrupt. This is used to make sure that audio data will be present at the audio output busses.

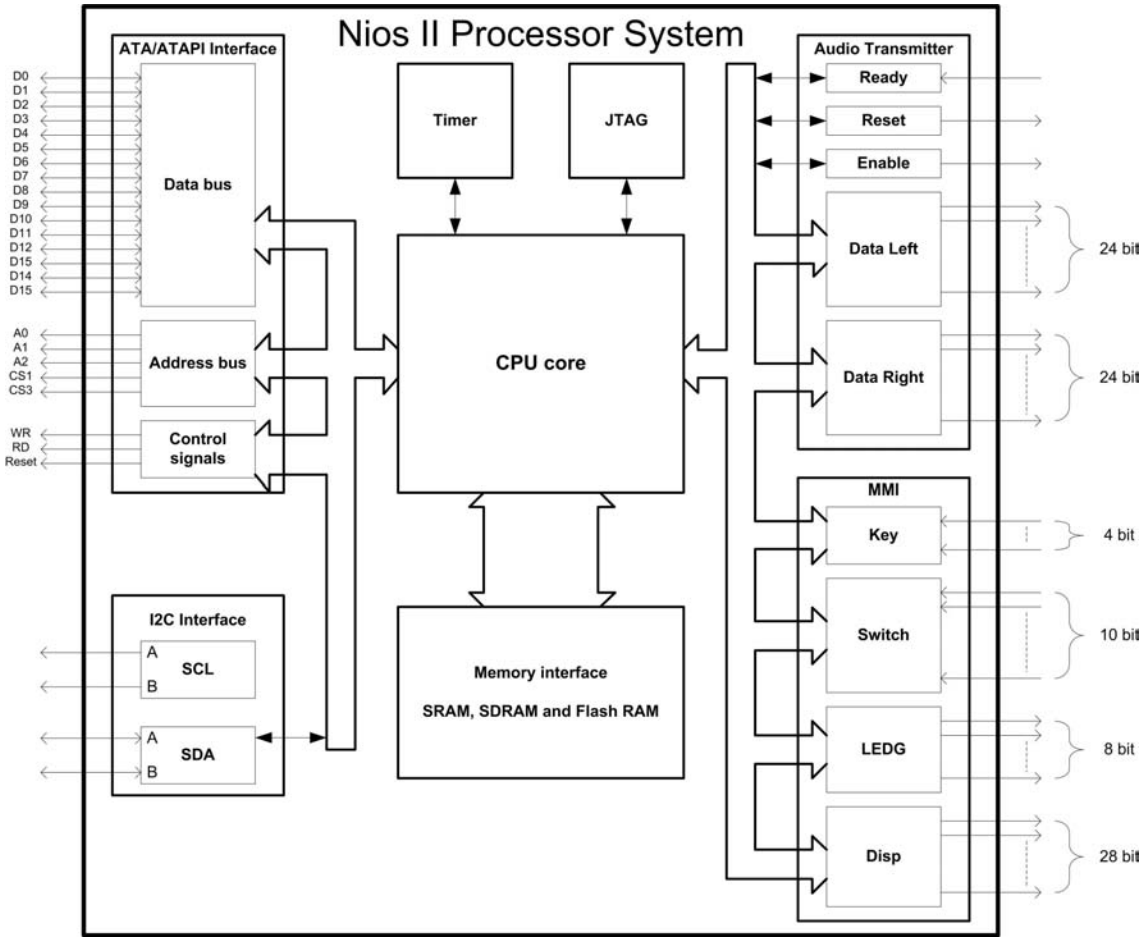
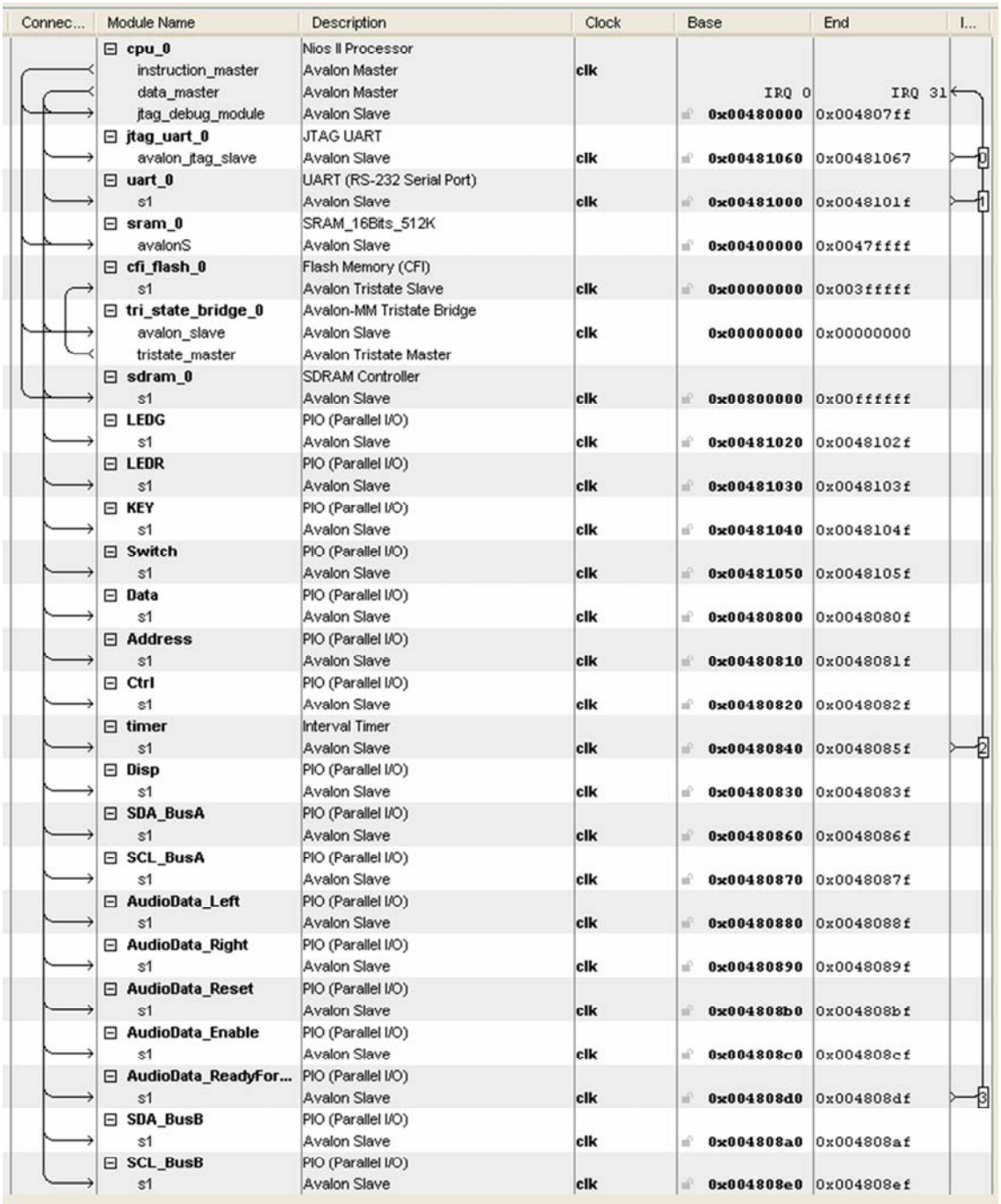


Figure 5: Nios II Processor System block diagram

A 28 bit PIO output port for controlling the 7 segments on the board was added. By controlling the 7 segments by software alone any of the LEDs in the segments can be switched on or off individually (If one wishes to).

For testing of the CD player block the audio codec on the DE1 board will be perfect. The codec is controlled by an I^2C interface. For easy control of the codec the clock and data signals are implemented as PIO ports in the Nios

processor. The clock line as output and the data line as a bidirectional port. The complete Nios II processor system can be seen on Figure 6 and as block diagram on Figure 5.



Connec...	Module Name	Description	Clock	Base	End	I...
	cpu_0	Nios II Processor				
	instruction_master	Avalon Master	clk			
	data_master	Avalon Master		IRQ 0	IRQ 31	
	jtag_debug_module	Avalon Slave		0x00480000	0x004807ff	
	jtag_uart_0	JTAG UART				
	avalon_jtag_slave	Avalon Slave	clk	0x00481060	0x00481067	0
	uart_0	UART (RS-232 Serial Port)				
	s1	Avalon Slave	clk	0x00481000	0x0048101f	1
	sram_0	SRAM_16Bits_512K				
	avalonS	Avalon Slave		0x00400000	0x0047ffff	
	cfi_flash_0	Flash Memory (CFI)				
	s1	Avalon Tristate Slave	clk	0x00000000	0x003fffff	
	tri_state_bridge_0	Avalon-MM Tristate Bridge				
	avalon_slave	Avalon Slave	clk	0x00000000	0x00000000	
	tristate_master	Avalon Tristate Master				
	sdram_0	SDRAM Controller				
	s1	Avalon Slave	clk	0x00800000	0x00ffffff	
	LEDG	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00481020	0x0048102f	
	LEDR	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00481030	0x0048103f	
	KEY	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00481040	0x0048104f	
	Switch	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00481050	0x0048105f	
	Data	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480800	0x0048080f	
	Address	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480810	0x0048081f	
	Ctrl	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480820	0x0048082f	
	timer	Interval Timer				
	s1	Avalon Slave	clk	0x00480840	0x0048085f	2
	Disp	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480830	0x0048083f	
	SDA_BusA	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480860	0x0048086f	
	SCL_BusA	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480870	0x0048087f	
	AudioData_Left	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480880	0x0048088f	
	AudioData_Right	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x00480890	0x0048089f	
	AudioData_Reset	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x004808b0	0x004808bf	
	AudioData_Enable	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x004808c0	0x004808cf	
	AudioData_ReadyFor...	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x004808d0	0x004808df	3
	SDA_BusB	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x004808a0	0x004808af	
	SCL_BusB	PIO (Parallel I/O)				
	s1	Avalon Slave	clk	0x004808e0	0x004808ef	

Figure 6: Nios II configuration

3.3 ATA/ATAPI interface

The communication protocol for the ATA/ATAPI interface is described in the standards published by the T13 group at www.t13.org . Here ATA/ATAPI-4 (d1153r18) was used. For communicating with the CD-Rom drive a different standard was used. At first the obsolete SSF-8020i was used because it has a very good description of the inner workings of the CD-Rom drive. Afterwards the Multi Media Command Set standard (mmc2r11a) from the T10 group at www.t10.org was used. It is not necessary to use a newer version of the standards, because the changes made in the standards only affects newer technologies. The newer versions of the MMC standards include support for CD writers, Blue Ray etc. This is not needed for this project and the MMC2 document is significantly smaller than the later versions.

First step in the development of the interface was hardware interface at the lowest level. This is what is described in the ATA/ATAPI-4 standard. The communication through the ATA interface is performed through 8 bit registers addressed with the 3 address and 2 chip select signals. These signals are the 5 bit address PIO on the Nios processor. Only one of the chip select signals can be active at the time. This makes 16 possible register addresses where only 9 are used. The reading from and writing to the registers are done with a read and a write signal. These signals are in the control PIO on the Nios processor who also contains a reset signal to the CD-Rom drive. The 8 bit register is bit 7 to 0 on the 16 bit data bus.

The 9 registers are called the task file. These are used to control the connected device. The task file of a data read/write register, error, status, feature, command register and a few other registers. The short version of the way it works is that commands on the device are performed by writing a byte to the command register. The command the device performs depends on the byte written to the command register. If the command need parameters to run these are written to the rest of the task file registers. The commands available and the parameters they take can be found in the ATA/ATAPI-4 standard. It is very well documented.

The most basic functionality for the interface is programmed in the hw_interface.c and hw_interface.h files (See the source code). They contain functions for reading from and writing to the registers in the task file and functions to initialize and reset the interface. For convenience there is also made a function that reads the value of the busy bit in the status register. It is quite easy to get the busy bit from the status register. The only reason that this has its own function is that it is used very often. On top of the simple basic functions are the real ATA and ATAPI commands.

As described in the MMC-2 ATAPI (or ATA Packet Interface) is an extension of ATA that allows SCSI devices to be connected to the ATA interface. The standard ATA interface is usually used by disk drives but Compact Flash cards supports the ATA interface as well. ATAPI is normally used by CD/DVD Rom drives.

The ATAPI protocol works by first sending an ATA command called ”ATAPI packet command” having the command byte A0h. Following this a packet of 12 bytes are send as data containing the packet command and the parameters for

the command. The device will then try to perform the command. If it fails an error is indicated in the status register. If the command is successful no error is indicated and if the command returns data the number of bytes to read from the device is available in two of the task file registers. The commands available and their parameters can be found in the Multi Media Command Set (MMC) standard. The standard is very well written, and the protocols for different ATAPI transfers are shown in detail.

The programming of the ATAPI software was unproblematic. The biggest challenge was getting an overview of the ATA/ATAPI-4 and the MMC-2 standards and sorting out which of the many commands are relevant for this project.

ATA and ATAPI commands used in this project can be found in ATAPI.h and ATAPI.c.

Not all of the ATA and ATAPI commands in the files are used in the final project, but they have been necessary during the development of the ATAPI interface software.

The CD-Rom drives used in this project are taken from an old computer. It had an old AOPEN x52 speed CD-Rom drive and a Creative CD writer. Both drives have been used during the development process.

Figure 7: CD-ROM drives used in the project

The capabilities of the drives were not know. Therefore some of the first commands written were the “Identify Device” and “Inquiry” commands. The first command returns information on the serial number, firmware revision, model number and electrical timing. The last command returns information on which standards the device conforms to, and various vendor and product information.

In order to get an idea of which data formats are supported and the hardware capabilities of the drive another ATAPI command was needed. The command “Mode Sense” has the ability to return a data packet (Called a page) containing CD-Rom capabilities and mechanical status. By reading this page it was verified that both drives support CDDA read operations. This means that the digital

audio data can be read directly from the CD. This wasn't the plan to begin with. At first the intention was to make the CD-Rom drive play the CD, and take the digital output from the connector on the rear side of the drive and feed it to the audio processing/amplifier stage. This was abandoned but more on that later.

The ATAPI commands to read status and capabilities information from the drive were a good way to test if the ATAPI interface software was working, and to build a good skeleton for implementing more commands.

After successful test of the first ATAPI commands several others were implemented in order to be able to play, pause and stop the CD. These functions were implemented and tested with success. To read the number of audio tracks, their length and the total length of the CD the Read TOC (Table of content) ATAPI command was implemented. The data received are processed by the functions in the TOChandler.h and TOChandler.c. These functions are described in the TOC handler section of this document. The ATAPI functions implemented at this time were sufficient to begin making the basic CD player functionality. Consult the CD player functionality section in this document for further details on that subject. Several other ATAPI commands are implemented. They are not that interesting. The way the ATAPI commands are send comes in two variations. One that returns data and one that doesn't. The only difference in the different commands are the setup of the packet before it is send to the CD-Rom drive.

3.4 CD player functionality

The basic CD player must contain a minimum of functions. This includes play, pause, stop, eject and skip to next and previous audio track. A lot of other features could be implemented, but since only four keys are available for control it is better to keep the interface simple. The function of the keys can be seen in the table below.

Key #	Functionality
1	Skip to next track
2	Skip to previous track
3	Play, Pause
4	Stop, Open/close CD tray

The functionality of the CD player is implemented as a state machine with six states. The states are: Check for CD, Ready, Not Ready, Tray Open, Playing and Paused. The sequence of the states and which events cause which transition can be seen on Figure 8.

After Power up general initialization is performed, and the state machine loop is entered. The relevant code for the current state are executed, and if an event that can cause a change in state it is changed accordingly.

In the Check for CD state it is attempted to read the TOC from the CD. If a TOC is successfully read it is written to the console via the jtag uart and the state changes to Ready. If it after several attempts fails to read the TOC the Not Ready state is entered.

In the Not Ready state the only allowed action is to eject the CD tray and load a new disk. When the CD tray is ejected the state changes to Tray Open.

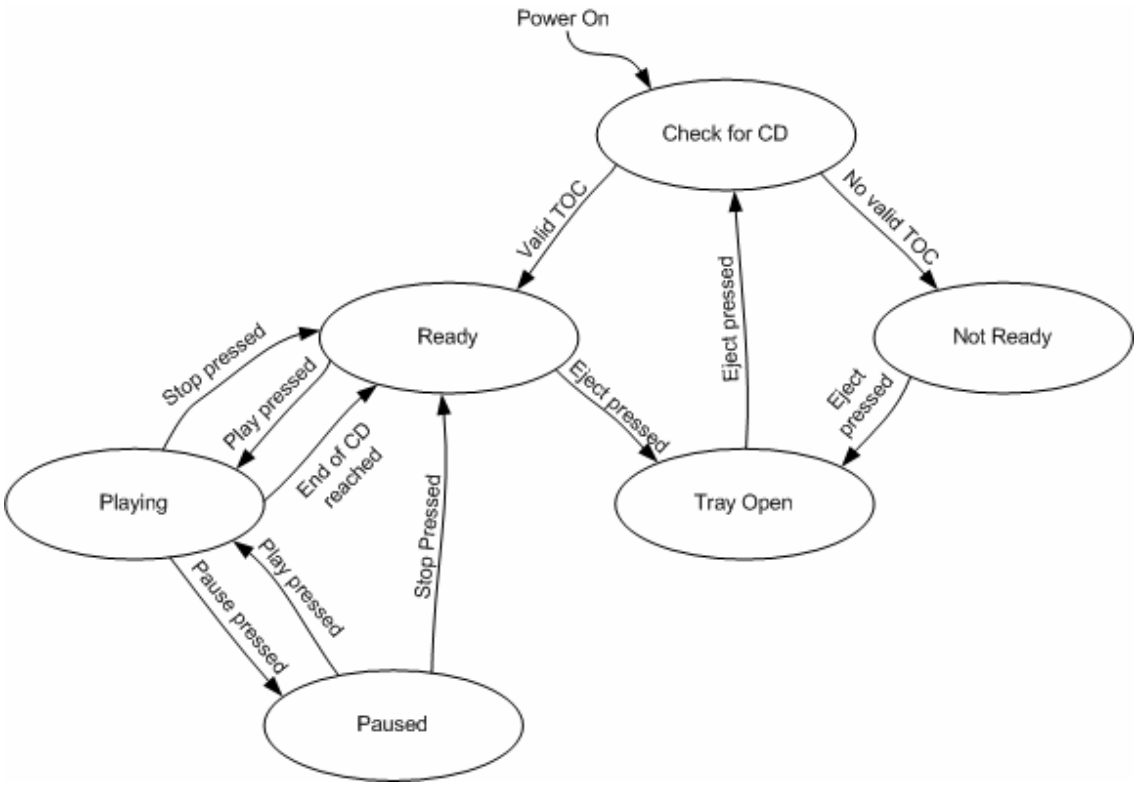


Figure 8: CD player State Machine

The Tray Open state does nothing. It only waits until the eject button is pressed, which initiates the closing of the CD tray, and changes the state to Check for CD.

If the current state is Ready it indicates that the CD is ready to be played, and the current position of the CD is at the beginning of the first track. Nothing is performed in this state until the play button is pressed starting playback of the CD and changing the state to Playing, or the eject button is pressed causing the tray to open and changing the state to Tray Open.

If the state is Playing all the keys have a function, but only two of them will change the state of the state machine. If the stop button is pressed the state changes to Ready, The audio playback is stopped and the current position on the CD is set to the beginning of the first track. If the pause button is pressed the audio playback will be stopped, but the current position on the CD when paused is kept in order to be able to resume playing from the same position. If the skip to next track button if pressed the current position on the CD is set to the beginning of the next track. If the current track is the last one on the CD the current position is set to the beginning of the first track. If the skip to previous track button is pressed the relative position of the current track is checked. If the relative position is more than three seconds into the current track the current position in the CD is set to the beginning of the current track. If the relative position is less the three seconds into the current track the current position on

the CD is set to the beginning of the track before the current track. If the current track is the first one on the CD then the current position is set to the beginning of the last track on the CD.

When the state machine is in the Paused state the buttons have almost the same function as in the Playing state. The only exceptions are that the play button resumes audio playback and changes the state to Playing. The two skip buttons and the Stop button have the exact same function as in the Playing state.

Every time the state machine loop has been executed the 7-segment displays are updated depending of the current state of the state machine. If the state is either Playing or Paused the current (absolute) position on the CD, the relative position of the current track or the number of the current track can be displayed. What information to display is chosen by the position of some of the switches on the DE1 board.

If the state machine is in the Playing state a bit of code is executed every time the main loop is run. This has to do with the audio playback, and updating the content of some data buffers. This will be described in detail in the section about the audio transmitter.

The CD player main loop seems to function alright. It hasn't been subject to extensive testing, but during the development process of the project it has been used quite a lot, an a few bugs has been found and corrected. The current CD player interface is very limited, and is only intended to use as a proof of concept interface. The final version should contain more keys for the control interface, and a different kind of display. Maybe a character or graphic LCD display. The extended control interface should make it possible to implement more advanced features. Some of the features could be repeat track, repeat disc, random track play, program track sequence, etc. This must wait until version 2 of the project.

3.5 Audio transmitter

The initial idea in the development of the CD player part of the project was to start the audio playback on the CD-Rom drive and route the digital audio output from the rear side of the drive to the audio processing/amplifier block. Once the audio playback is started the CD-Rom drive pretty much takes care of it self. That was one of the very appealing properties of this approach. This idea proved to be a bit harder to implement than first estimated. The format of the digital output is SPDIF. It is significantly different from the I^2S format that is needed by the audio processing/amplifier block. A SPDIF to I^2S converter could be constructed, but only one of the two CD-Rom drives used for this project had a SPDIF output. Because of that the design was reconsidered.

The new design idea is to read the audio data from the CD and sending them to a parallel to I^2S converter implemented in VHDL outside the Nios II processor.

The I^2S bus transmits data at 64 times the sampling frequency of the audio data. This creates a requirement for data being available in a steady stream. The I^2S converter is constructed in such a way that it toggles a signal when the parallel data has been read and the converter is ready for a new set of audio samples. To insure that data is always present for the I^2S converter an interrupt

function is set up to trigger on every edge received on the Ready pin in the Audio Transmitter block in the Nios processor. The interrupt function has access to two large buffers. They will be full of audio data. While one buffer is being transmitted to the I^2S converter the other buffer is being filled with data from the CD-Rom drive. Audio data read from the CD is read in frames of 2352 bytes. Since the audio data is stereo and has a resolution of 16 bits each block contains 588 samples, with a sampling frequency of 44100 Hz a frame of data will take $1/75^{th}$ of a second. This leads to the MSF structure/format used to address data on the CD. The MSF structure consist of three bytes named M, S and F. M indicated minutes, S is seconds and can hold the values zero to 59, F is frames and can hold the values from zero to 74.

The two buffers containing audio data can hold 30 frames each. Each buffers length in time is 400 ms. The time needed for the CD-Rom drive to fill the buffer is approximately 150 ms. This leaves plenty of time for the processor to perform other tasks. The length of the two buffers can be set to other values, but a length of 30 frames seemed to work fine, and there was no need to change this setting. Two state variables are associated with each of the buffers. They are used to tell the interrupt function if the buffers are ready to be played, and the section of code in the main loop (mentioned earlier) which buffer is ready to be filled with new audio data. The section of code in the main loop checks the state of the two buffers. If one is empty it is loaded with data, and its state is changed to ready.

The interrupt function has an index pointer telling how far along in the buffer the playback is. When the index pointer reaches the end of the buffer, the state of the buffer is changed to empty making it possible for the main loop to fill it with new data. If the other buffer is ready the interrupt function switches to this buffer, resets the index pointer and sets the state of the now selected buffer to playing. If the second buffer isn't ready when the first buffer has been processed, it is assumed that the end of the CD has been reached. In this case both buffer states are set to empty.

Along with the interrupt function there are a few other functions in the AudioTransmit.h and AudioTransmit.c. These functions are primarily used to read and write to the private variables of the AudioTransmit functions group. There are also functions for initialization and control functions for the I^2S converter. They can reset, enable/disable the vhdl module. Reset returns the I^2S converter to the initial state. If the enable signal is active data is read form the left and right data bus and transmitted on the I^2S bus. If enable is inactive the audio data is disregarded and the converter transmits zero values.

3.6 Table of Contents (TOC) handler

To process the TOC data read from the CD a set of functions has been written in a grouping called TOChandler. These are found in TOChandler.c and TOChandler.h. Since the length of the TOC varies with the number of tracks on the CD the memory for storing the TOC is dynamically allocated. The setTOC function gets a pointer to where the TOC data from the CD is located, and an unsigned short telling the number of bytes in the TOC data. The function then

allocated the needed memory, and organizes the TOC information in a way that is easily accessible for further use.

A data structure called MSF_type is defined in the TOChandler function set. It is used to contain position indexes on the CD in the MSF format mentioned earlier. The TOChandler group also includes functions to reset the TOC and free the allocated memory, to tell if a valid TOC is present and to retrieve various information from the TOC. This includes the number of tracks and length of the CD, the position of a given tracks beginning, and the current track number calculated from the current position on the CD. Further more there are a few functions to perform basic math and comparison operations on the MSF_type data structure.

3.7 Keypad

The control interface for the CD player is based on the keys on the DE1 board. The type of keys used has in previous projects proved to generate a lot of contact bouncing. Although debounced on the DE1 board it is further eliminated by combining polling with counters with hysteretic thresholds for each of the keys. It works by polling the keys and updating counters depending on the state of the keys. If a key is pressed the counter is incremented otherwise it is decremented. The counters are limited to a certain value range. The range can vary with the type of key used and the polling interval of keys. Upper and lower thresholds are set to suitable values within the counter range. If the upper threshold is exceeded the state of the key is considered active, if the counter value goes below the lower threshold the key state is considered inactive.

This method has proven successful in previous implementations on embedded systems. And it worked quite well here too. The polling function is triggered by a 1 ms interval timer interrupt. To check if a key has been pressed a function called pressedKeypad is used. This function returns a bit pattern corresponding to the pressed keys. Once read the bit pattern is reset until the key has been released and is pressed again.

The polling function is triggered by the timer interrupt repeatedly during program execution. The pressedKeypad function is called in the beginning of every main loop cycle.

3.8 Display

To control the four 7-segments a setDisplay function is used. It takes a string as parameter, and writes the supported characters to the 7-segments. If a character is not supported the 7-segment is left blank. The characters that are supported are all the numbers and the “-“ sign. The display is quite easy to use by first forming a string with the spritf function from the stdio.h library.

3.9 CD-Text

CD-text is text information stored in the CD. This can contain the name of the CD, the recording artist, track names etc. Not all CD’s contain this information, and not all CD-Rom drives support reading this information. Only one of the

drives used in this project had CD-text support. Just to se if it was possible CD-text was support was implemented.

It functions almost in the same way as the TOChandler. An initCDTEXT function is called with a pointer to the CD-text data from the CD, the size of the CD-text data in bytes and the number of tracks on the CD. The needed amount of memory is allocated, and the track titles and artist names are sorted out, and placed in memory for later retrieval. The getCDTEXTtitle and getCDTEXTartist functions takes a tracknumber as parameter and returns a pointer to the appropriate string. The CD-Text support has no crucial function, but it is a neat little feature.

3.10 I^2S converter

In order to be able to send data in the right format to the audio processing/amplifier block it was necessary to design a parallel to I^2S converter. The design is made in VHDL. It takes parallel data form to 24 bit busses, and sends it out on an I^2S bus. The design creates all the necessary clock signals for the I2S bus, but needs a master clock signal that is a multiple of 64 times the audio sampling frequency. Here 256 times FS = 11,2896 MHz is used. The clock is generated by a PLL with the 24 MHz clock on the DE1 board as input. The I^2S converter schematic symbol can be seen on Figure 9. The connections on the left are inputs and those on the right are outputs. The reset is an asynchronous reset meaning that it is independent clk signal. The clk signal if divided by four internally in the converter. The divided clock signal is now equal to 64 times FS. It is routed directly SCK output. The On the falling edge of the divided clock signal data from an internal 64 bit register is shifted out on the SD output while the index of the register is less than 32 the WS output is set to zero otherwise it is set to one. When the index reaches the end of the register values from the DataL and DataR busses are latched into the register if the enable input is one. If enable is zero the register is loaded with zeros. After the register is loaded the Ready output is toggled indicating that the converter is ready for new data on the DataL and DataR busses. The SCK, WS and SD signals forms the I^2S bus. SCK is the clock signal, SD data and WS the word select or left/right clock. It might seem a bit strange that two 24 bit busses are latched into a 64 bit register indexed bit 63 to 0 with 63 as the msb. The DataL bus is latched into bit 62 to 39 of the register and DataR into bit 30 to 7. This is to conform to the I^2S protocol.

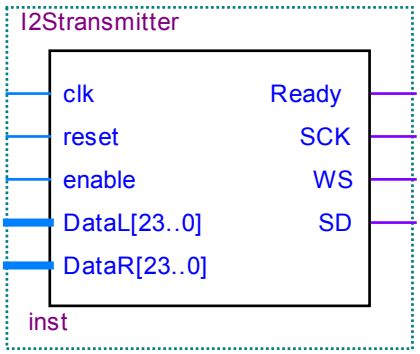


Figure 9: I^2S converter symbol

The I^2S converter design has been tested with simulated waveforms. The results of the simulation looked correct. To test even further the I^2S converter was connected to the audio codec on the DE1 board.

The codec is able to play the audio converted to I^2S format by the converter and read from the CD. It sounded quite good actually.

3.11 Phase Locked Loops

The system uses Phase Locked Loops to generate the needed clock signals. One PLL doubles the 50 MHz to a 100 MHz clock used by the Nios processor and a slightly delayed version of the 100 MHz clock used by the sram. The audio clock must be a multiple of 44100 Hz. The 24 MHz clock can be made into 22.5792 MHz which is 512 times FS.

4. The Wolfson WM8731 CODEC

The DE1 board is equipped with a WM8731 CODEC. In this project the CODEC is used as both test equipment for other modules and as secondary input/output for the audio system.

4.1 CODEC Hardware in this project

The CODEC is mounted on the DE1 board and has fixed connections to the FPGA (hence fixed pins; refer to the FPGA pintable in the overall implementation section). Initialization of the CODEC is through a standard I^2C (Inter-Integrated Circuit) bus, and sound transfer is through a 3-wire bus, witch in this project is defined as a standard I^2S (Inter-IC Sound) bus. Both the I2C [1] and the I2S [2] bus is originally defined by Philips, but is now de facto standards.

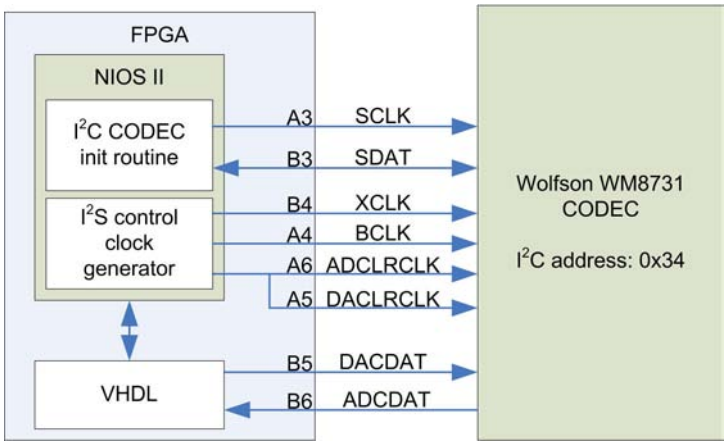


Figure 10: Connection diagram of CODEC part of the DE1 board

The figure above shows the structure of the CODEC connection to the project. Pin numbers on the FPGA are marked just inside the FPGA frame. All control signals for the CODEC are generated using the NIOS II processor. Routing of the sound signals is done using VHDL modules.

NOTE: Errors are encountered in the DE1 manual. On page 38, figure 4.15 the write (and read) addresses for the I^2C bus is wrong. True write address is 34h. On page 39 in table 4.9 the description for the I^2C bus is wrong. I2C_DATA is connected to PIN_B3 and I2C_CLOCK is connected to PIN_A3.

4.2 CODEC setup

Registers of the CODEC is initialized as shown in the table below. All registers are written using slave device address 34h on the I^2C bus. The sub address of this device operates with only 7bit. This deviates from the standard, witch

suggest an 8bit sub address. Because of this the sub address will not be uniquely identifiable in the hexadecimal value to send.

Register	Sub address 7bit	Binary setting 9bit	Hex to send	Description
R0	0000000b	000010111b	0017h	Line input amplification is 0dB and mute is disabled. Left (R0) and right (R1) input registers do not load simultaneously.
R1	0000001b	000010111b	0217h	
R2	0000010b	011100101b	04E5h	Output amplification is -20dB and mute is disabled. Left (R2) and right (R3) output registers do not load simultaneously.
R3	0000011b	011100101b	06E5h	
R4	0000100b	000010010b	0812h	Connect input to ADC. No mic or bypass from input to output.
R5	0000101b	000000000b	0A00h	Enable ADC high pass filter. Disregard DC level. No mute or other filtering.
R6	0000110b	001100010b	0C62h	Power down: mic. ampl., oscillator and CLKOUT.
R7	0000111b	000001010b	0E0Ah	Use I^2S, 24bit, lrclk as in standard and CODEC is slave.
R8	0001000b	000101000b	1028h	Use normal mode with f_s=44.1kHz and XCLK=256f_s
R9	0001001b	000000001b	1201h	Activate device.
R10	0001111b	Reset register is not used in this application		

The CODEC output volume is corrected by user intervention at runtime, otherwise all CODEC registers is only set at power on.

4.3 I^2C

The standard defines a master/slave based bus system. In the CODEC case the slave is the WM8731 and the master the FPGA.

Write of one byte

The protocol is shown in Figure 11. From last chapter is recognized 8bit data, 8bit sub address and a 7bit address followed by a R/W bit. In the CODEC case only writing is interesting, and these 8bits give the 34h mentioned earlier.

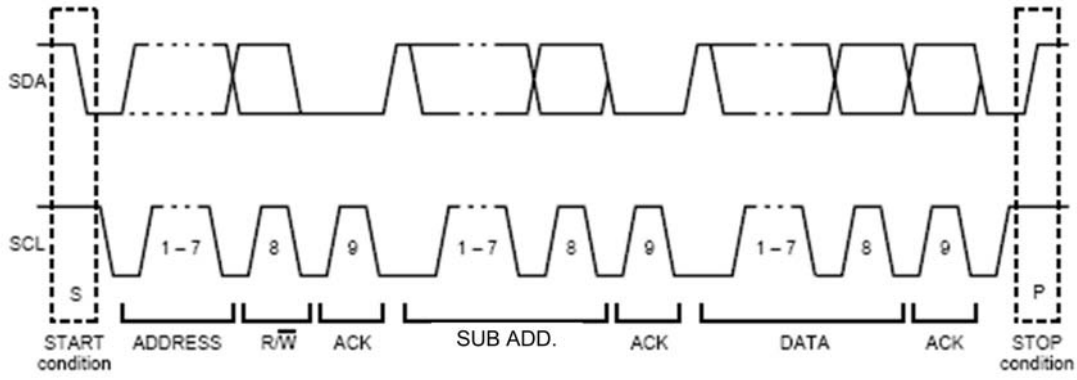


Figure 11: I2C protocol (from Philips I^2C specification)

ACK is a low response from the slave.
The protocol defines that idle of the bus is logic high. This is achieved by using pull-up resistors on the physical bus (R=2kΩ on the DE1 board), and putting outputs of slaves and masters on the bus in tristate when idle. This also means that when a device is transmitting logic high, it should tristate the output, and when low it must pull the wire to ground. In this project however logic high and low is used. This will not pose a problem.

Write of multiple bytes

Some I^2C devices require the writing of several bytes for each sub address. The power amplifier circuits used I the project requires a four byte write for some of the sub addresses. There don't seem to be basis for this kind of operation in the I^2C standard but never the less it is used. Below is the impulse diagram of the SDA, the start and stop is similar to Figure 11.

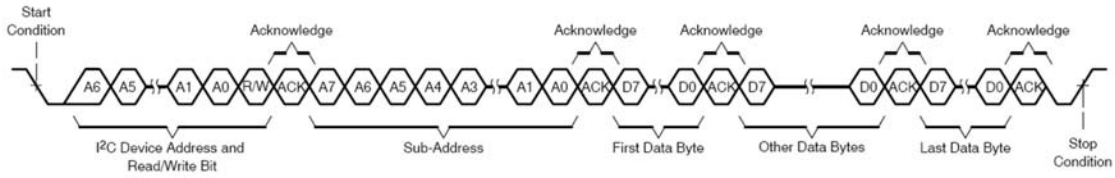


Figure 12: Multiple byte write transfer (from TI datasheet TAS5504)

Read

First part of the read process is similar to the write. The master device transmit START condition, address of the slave, '0' in the R/W bit. After ACK the sub address is transmitted. Here the similarities end, because a new START condition is transmitted followed by the slave address but a '1' as R/W bit. Now the slave takes control of the SDA, and the master replies with ACK.

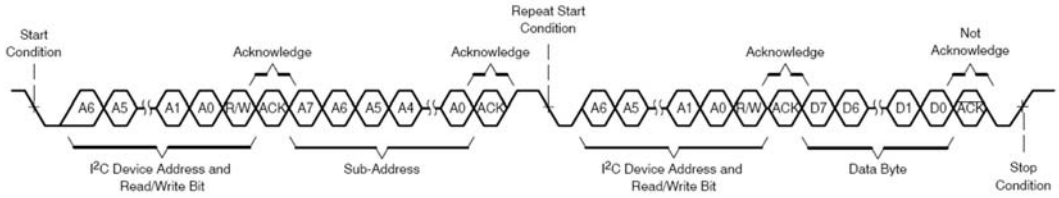


Figure 13: Read of a single byte (from TI datasheet TAS5504)

I^2C signal routing in present project

Due to the DE1 hardware there is only two devices on the bus (I2C_bus_A), however the need to attach further devices have arisen in this project (the power amplifier also needs I^2C bus connectivity). To overcome this problem a second pair of bus pins will be assigned the exact same signals as the I2C_bus_a. This second bus is named I2C_bus_b. This approach is chosen

because the FPGA software will not allow two bidirectional wires to be connected directly.

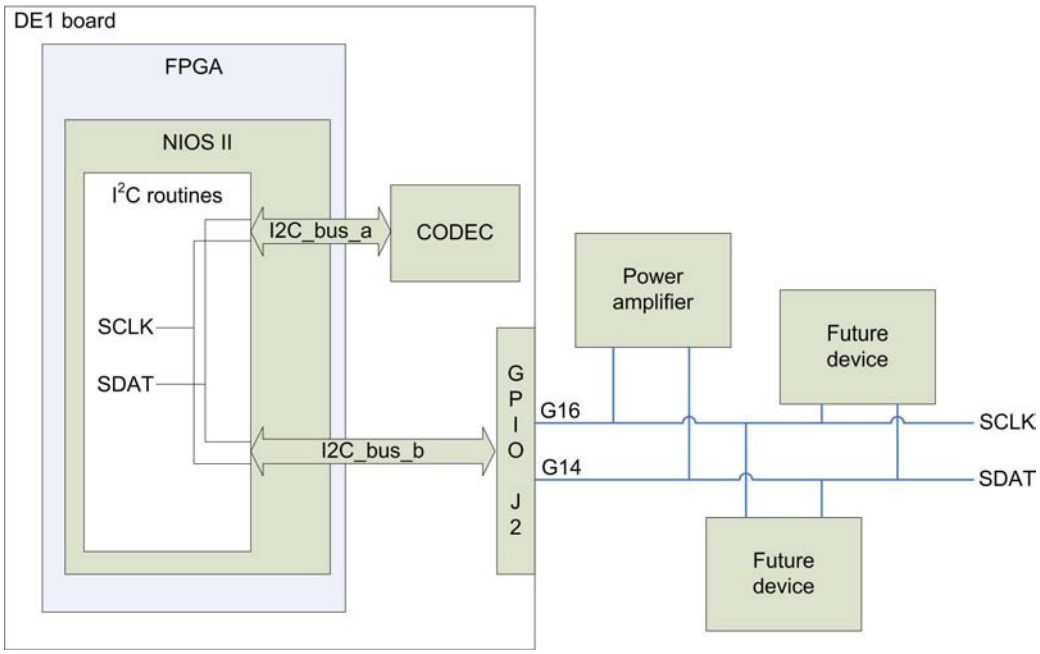


Figure 14: Routing of I²C signals

Because of the chosen routing, the devices on either I2C_bus_a or I2C_bus_b will appear to be on the same bus. When using the send, long send and receive functions (I²C implementation in next section) the programmer will not have to distinguish between two hardware busses. A second master device on I2C_bus_b will not be able to see the I2C_bus_a though.

Figure 14 shows the setup of the I²C routing. The off-board I2C bus uses GPIO socket 1 (JP2) pin 5 for SDAT and pin 8 for SCLK (pin G14 and G16 on the FPGA). Future devices on the figure are not a part of the project, but merely to show the expandability of the bus.

I²C implementation

The I2C_bus_a and I2C_bus_b hardware are generated by adding PIOs to the Avalon bus using the SOPC builder. All address configurations is also managed by the SOPC builder.

s1	Avalon Slave	clk	0x00480830	0x0048083f
SDA_BusA	PIO (Parallel I/O)			
s1	Avalon Slave	clk	0x00480860	0x0048086f
SCL_BusA	PIO (Parallel I/O)			
s1	Avalon Slave	clk	0x00480870	0x0048087f
SDA_BusB	PIO (Parallel I/O)			
s1	Avalon Slave	clk	0x004808a0	0x004808af
SCL_BusB	PIO (Parallel I/O)			
s1	Avalon Slave	clk	0x004808e0	0x004808ef
AudioData_Left	PIO (Parallel I/O)			

Figure 15: Snapshot of the SOPC builder of I²C hardware

Addresses of the I2C hardware is shown in
Figure 15, but are automatically copied to the system.h file on “make clean” command in the Nios II environment.

Protocols are implemented by driving the bus to hard low or high according to the previously described. When the ACK is expected the direction of the SDA pin is reversed, if the slave pulls SDA low no error is returned, else error is indicated. The error indication is displayed in the IDE console (via JTAG), but do not affect the program in present implementation.

4.4 Test of CODEC

To ensure the CODEC doesn’t corrupt test data (and audio data) the transfer function of the codec is measured. During test output amplification is set to 0dB (R2: 04F9h and R3: 06F9). In the FPGA the DACDAT is connected ADCDAT (Figure 17). Test setup is shown below.

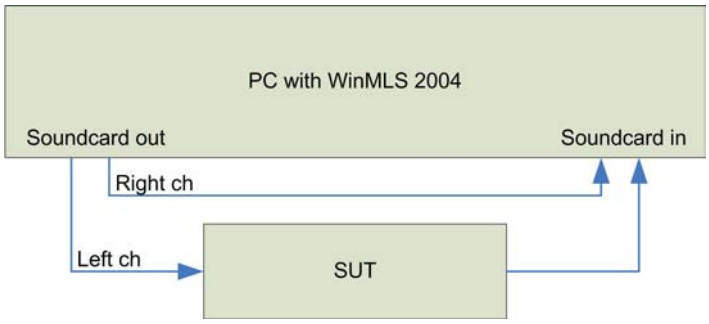


Figure 16: Test setup

To compensate for the standard soundcard in the PC, the right output channel is patched directly to the right input channel, and the computer software (WinMLS 2004) automatically subtracts this transfer function from the one measured by the left channel. The left channel is used to measure the System under Test (SUT), namely the DE1 board. The DE1 board loaded with NIOS II code witch produces I^2C, I^2S clock signals and init registers in the CODEC. Part of FPGA schematics specific for the test is shown in the figure below.

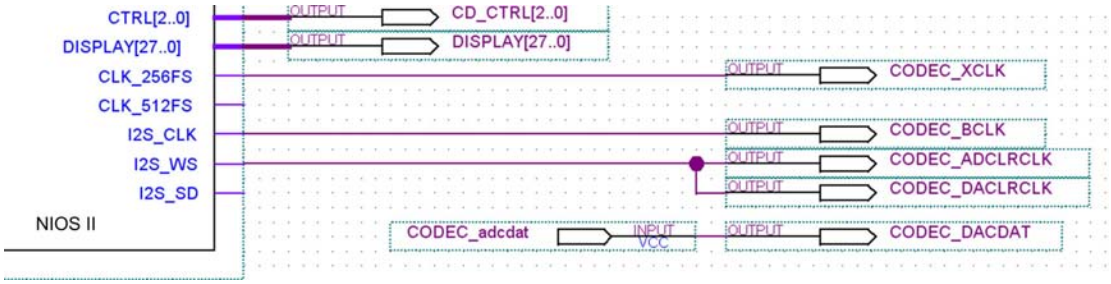


Figure 17: Software for test of CODEC transfer function.

The WinMLS 2004 uses sinus sweeps to measure the transfer function.

Results

Magnitude plot from the measurement is shown in Figure 18.

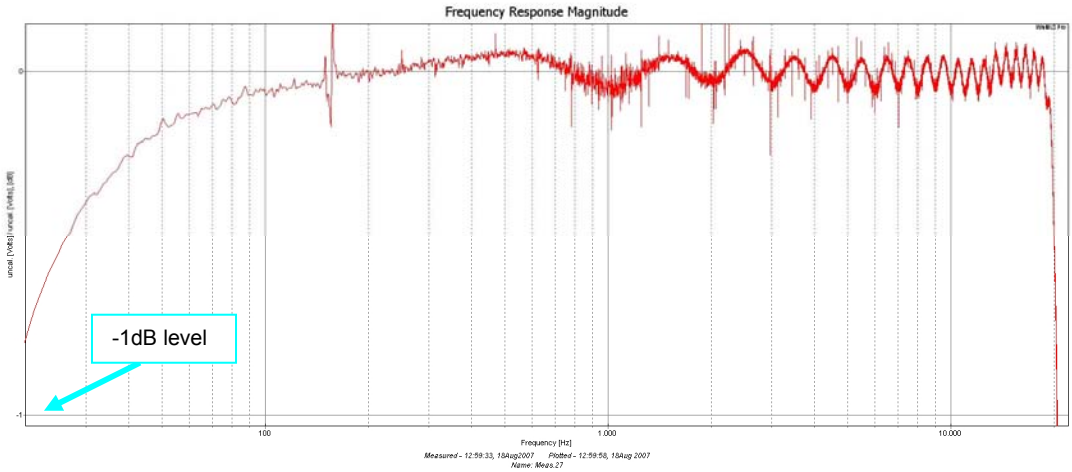


Figure 18: Measurement of CODEC

From the figure it is evident that the CODEC not will pose a problem. The ripple is much less than 1dB, the upper cut-off frequency is about f_U=20kHz and the lower below f_L=10Hz.

5. Power amplifier

The output part of the project is utilizing class D amplifiers.

5.1 Class D functional description

Figure 19 shows the basic setup for a class D amplifier. The desire is to amplify a signal (marked audio in the figure).

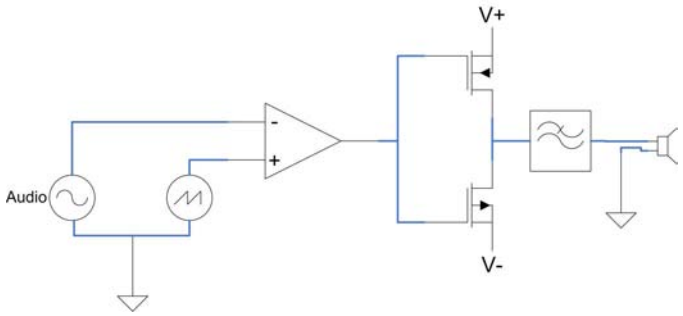


Figure 19: Basic Class D amplifier

This signal is fed to a comparator, where it is compared with a triangle wave (or a saw tooth wave). The output of the comparator is a modulated square wave, or a Pulse-with Modulated (PWM) signal (Figure 20). The duty cycle of the PWM reflects information about the amplitude. The frequency of the signal is represented by the PWM's change over time.

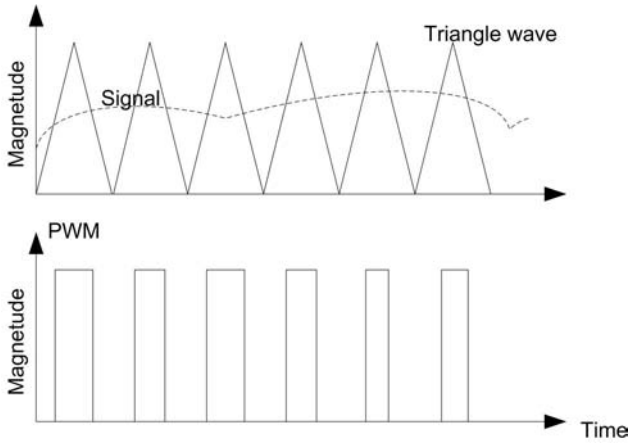


Figure 20: Generating a PWM signal.

The PWM signal drives the power stage. The power stage is in this example a pair of FET's, but often the speaker is the center of an H-bridge. The FET's reproduce the PWM at power supply levels, so demodulation must be conducted. This is done by low-pass filtering the signal. After filtering the signal is analog and ready for the speaker.

5.2 The triangle wave

The frequency of the triangle wave must be way outside the audio signal frequency range, and always higher. If the triangle wave were equal to or had a lower frequency, the harmonics would corrupt the audio signal. The normal audible frequency range is in the interval f_h=[18; $20 \cdot 10^3$] Hz, so the switch frequency should be about f_{sw}=30kHz.

The waveform can easily be generated with a up/down counter, but to avoid degrading the audio signal by the amplifier, the solution of the triangle should be as good as the audio signal. In this project that is 24bit.

This means that we need a 24bit counter that can count an up/down cycle in $t=1/f_{sw}$=33.3µs or by other words, the clock supplying the counter operates at

$$f_{clk} = f_{sw} \cdot 2^B = 30 \cdot 10^3 \cdot 2^{24} = 503.3GHz$$

witch is simply not doable in a FPGA. Even if the demands are loosen there is a long way down to the maximum clock frequency of the FPGA at f_{max}=300MHz.

This forces the PWM modulation outside the FPGA.

5.3 Texas Instruments 4 channel digital audio processor TAS5504

The TAS5504 is chosen to do the PWM modulation. Besides being able to PWM modulate 24bit at a switching frequency in the range from 32k-48k Hz it has an enormous amount of features. For this project the important features are: I^2C interface for programming. I^2S interface for transfer of audio signals. Four PWM channels, capable of driving four H-bridge power stages.

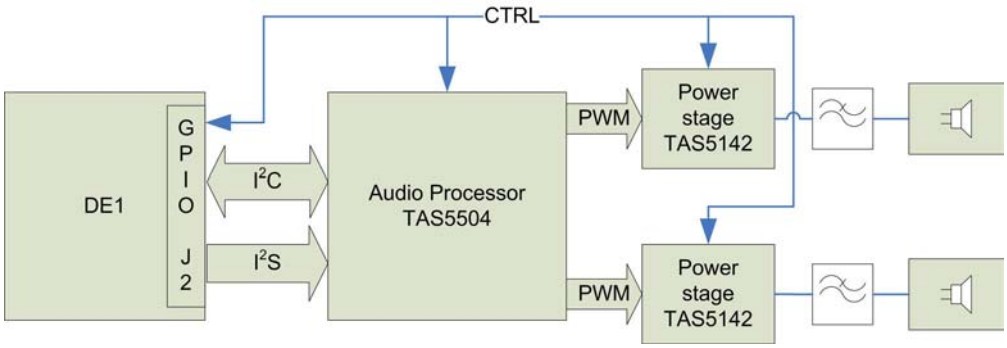


Figure 21: Block diagram of power amplifier

Figure 21 show the power amplifier and the connection to the DE1 board. For each power stage shown, there are two channels. This offer the possibility to move the crossover filter from the analog domain (in the speakers) to the digital domain (in the FPGA). Such configurations require an amplifier for each speaker driver (Tweeder and bass for both left and right).

TAS5504 setup

There are more than 50 registers to setup in the TAS5504, so only registers that differ from the default setting is described here.

Sub address 8bit	Binary setting	Hex to send	Description
00h	01000001b	0041h	44.1kHz sampling and master clk is 64f_s
D1h	00000000b 00000000b 00000000b 01000100b	D1h 00000044h	Volume of ch1 to 0dB. This register requires four data bytes.
D2h	00000000b 00000000b 00000000b 01000100b	D2h 00000044h	Volume of ch2 to 0dB. This register requires four data bytes.
D7h	00000000b 00000000b 00000000b 01000100b	D7h 00000044h	Volume of ch3 to 0dB. This register requires four data bytes.
D8h	00000000b 00000000b 00000000b 01000100b	D8h 00000044h	Volume of ch4 to 0dB. This register requires four data bytes.
D9h	00000000b 00000000b 00000000b 01000100b	D9h 00000044h	Master volume to 0dB. This register requires four data bytes.

The clock control register is automatically updated by TAS5504 auto detection, so it doesn't really need to be initialized.

5.4 Texas Instruments Stereo Digital Amplifier Stage TAS5142

The power stage, or the backend, of the system is based on semiconductors. In the example in the beginning of this section both a P channel and an N channel MOSFET is used. Both the on and off switching time will probably be different. When both FETs are on, the current through the components will be destructive. Therefore some kind of protection is required. This project utilizes the TAS5142 Power stage. Here the protection is embedded. Further more the FETs are produced on the same piece of silicon witch reduces time differences.

TAS5142 also offer over temperature detection capabilities and automatic shutdown when the temperature level is damaging.

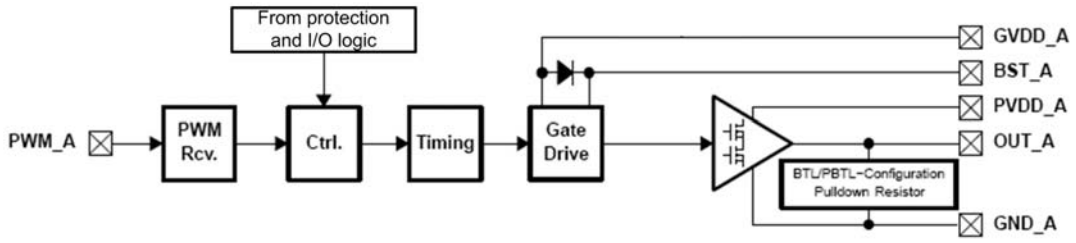


Figure 22: TAS5142 channel A (from TI datasheet TAS5142)

There are two full H-bridges in a TAS5142. Figure 21 show channel A of the power stage. TAS5142 has four of these channels. The PWM signal produced by the TAS5504 is received by the PWM receiver. Here levels and waveform are reconstructed, before latched through the control block. If the power stage is experiencing an error (power or temperature) or external reset is at logic low, the output of the block is low. Otherwise the PWM signal is latchet to the timing block. Here the PWM is split for the two FET's and are manipulated to deal with the FET's different switching time. Last the PWM is converted to drive signals with the high-side drivers (gate drive block). The gate drive needs a boot-strap capacitor, from GVDD to BST, to build up charge to drive the high-side of the half bridge.

Output filters

The filters needed to reconstruct the analog audio signal are suggested in the data sheet for the TAS5142.

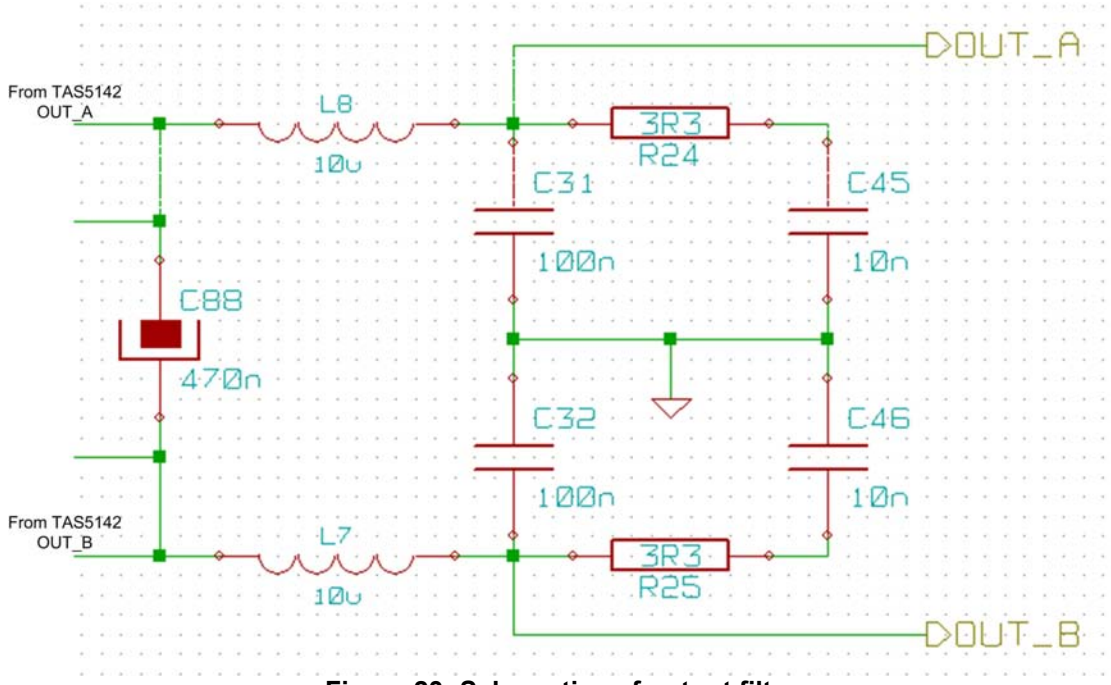


Figure 23: Schematics of output filter

Figure 23 shows the schematics of the output filter.

5.5 Hardware for the power amplifier

The overall schematics and PCB layout can be found in appendix A. This chapter discusses details in the HW design.

Error interaction

Figure 24 show how the audio processor and the power stage exchange error signals. Numbers in parentheses are pin numbers on actual components. The

audio processor produces a high valid signal when the PWM is a valid representation of the incoming I^2S signal, according to register setup. The valid pin controls the reset pin on the power stage. The reset pin is active low, so when the PWM is valid the power stage is not in reset.

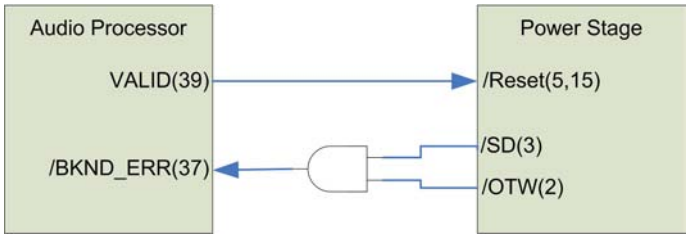


Figure 24: Error interaction between Audio Processor and Power Stage

The power stage has two error signaling pins. The Over Temperature Warning (OTW) warns that the component is above T=125°C. The OTW signal does not effect the operation of the TAS5142. The other error signaling pin is the Shutdown (SD). When this is low the device switches all FET's off. This signal is active on higher temperature, voltage errors or overload. These signals are combined in and AND gate and connected to the audio processors Backend Error (BKND_ERR) input. If the BKND_ERR is low the PWM is put in 50% duty cycle and the valid signal is pulled low. The AND gate is implemented in the FPGA.

Audio processor control signals

The control pins of the audio processor are routed to the FPGA. The pins in question are reset and mute. In the FPGA the reset is fixed to logic high, but the mute pin is controlled by DE1 switch 9 (SW9).

Power stage mode select

The power stage mode select pins (11, 12 and 13) are fixed to gnd. This means that the power stage expects two PWM signals that are inverts of each other, and also uses two channels for each output (using full H-bridges).

PCB layout

In the layout there are numerous decoupling capacitors and load capacitors. Capacitors are placed close to the components and on all connections to power supplies.

Ground is routed so digital and analog ground meet in a star architecture or at high cobber cross-sections. The ground is also filling on the board. This helps to shield signal connection and always provide large cobber cross-sections.

Inductors are placed in an angle of 90° to its neighbor inductors to avoid mutual induction and cross talk.

Lines for output of power stages are also chosen with larger cobber cross-sections to meet the requirements of currents up to I=10A.

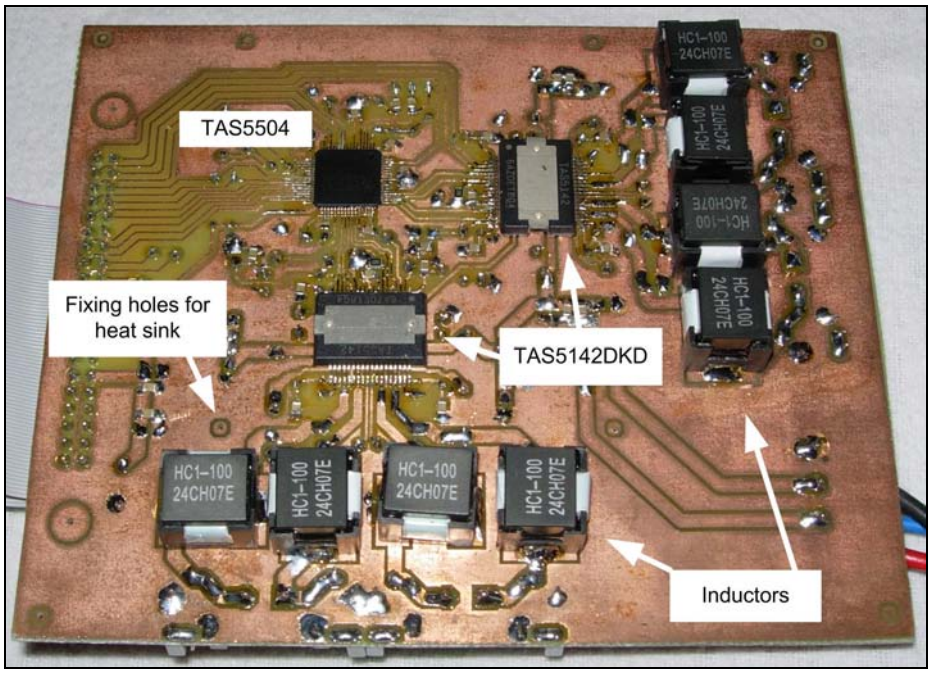


Figure 25: SMD components on power amplifier PCB

Cooling

The TAS5142 must be equipped with an appropriate heat sink to avoid overheating. At present no calculations is made on temperature, but holes are prepared for heat sink mounting. Figure 25 only marks one of these holes, but there are four placed quadratic.

Connection to the DE1 module

The power amplifier PCB is connected to the DE1 module GPIO port 1 (JP2). While the CD drive is connected trough a standard IDE ribbon cable, the power amplifier has to be connected through an ATA ribbon cable. The difference between these two cables is that every second line in the ATA cable is a ground connection. This shields the signal lines from ambient noise and more important from cross talk.

Cross talk was found to be a problem when the ordinary IDE connection was used.

I^2C pull up resistors

Pull up resistors on the I^2C bus is placed on the power amplifier PCB. To sink as little current as possible, the pull up resistors is chosen to R=200kΩ (16.5µA @ V_{cc}=3.3V). This posed a problem though. The resistors could not raise the I^2C bus to logic high sufficiently fast, when the bus is tri-stated.

In the present project this isn't a problem, because the I^2C masters violates the standard by pulling the bus to hard high (and do not tri-states the bus) when logic high is needed.

PCB power supply

The logic power V_{cc}=3.3V for the PCB is supplied from the DE1 board through the GPIO connector (JP2, pin 29). Power to drive the speakers PV_{DD}=30V is supplied from a LAB power supply directly to the power amplifier board (screw terminals).

5.6 Test of power amplifier module

There are two paths there should be testes. The first is the I^2C bus for setting up the audio processor. The second is the audio path.

Test of control structure

First test is to ensure the ability to communicate with the audio processor via I^2C. The I^2C routine confirms that the device answers the transmission with ACK (logic low). Second register 01h (General Status Register) is read and the chip identification code is confirmed with the datasheet. Last 00h is first written to register 02h (Error Status Register). This reset sticky errors that might be obsolete, and then register 02h is read to ensure no errors occur.

Unfortunately there is an error: "PLL auto lock error". This error is fatal, and prevents the processor from producing PWM. The course of the problem is at deadline not found, so the power amplifier does not work.

Test of audio path

This is obviously not possible due to the faults found in the audio processor PLL. But a transfer function would have been obtained similar to that of the CODEC.

6. Multiplexing and miscellaneous

Figure 26 show VHDL modules described in this chapter.

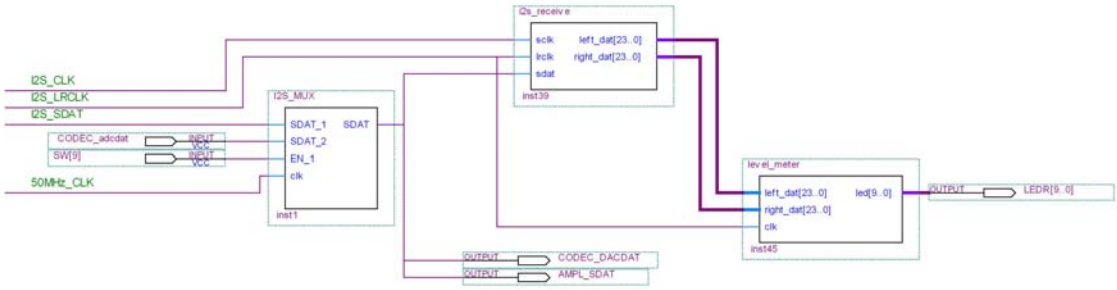


Figure 26: Routing of audio signal and level meter

The important part of the figure is the I2S_mux. This is the audio multiplexer. To the left is the I2S_receive there converts the I^2S to parallel data. The level_meter is indicating level of action on the parallel bus.

6.1 Audio multiplexer

Details of the audio multiplexer are shown on Figure 27.

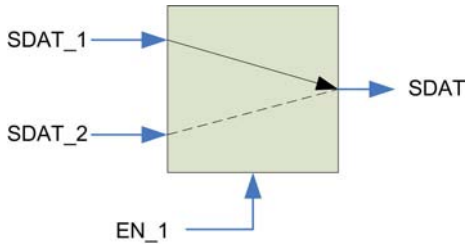


Figure 27: Details of the I2S_MUX

On a positive clock edge a low EN_1 will select SDAT_2 otherwise SDAT_1 will be selected. EN_1 is directly controlled by SW9 on the DE1 board.

The output of the multiplexer is always routed to the CODEC, the power amplifier and to the I^2S/par converter.

6.2 I^2S to parallel converter

Figure 28 show the functional simulation of the I^2S to parallel converter (i2s_recieve). The lower three sequences are internal signals. The signal LD is generated to mark a level shift in the LRCLK. The generation of the signal is generated as follows:

$$LD = LRCLK \oplus LRCLK \cdot z^{-1}$$

Where $\oplus$ is the “exclusive or” operator and the z^{-1} is a unity delay.

The operation is triggered by the SCLK. This means that difference in the LRCLK from one positive SCLK edge to the next is detected.

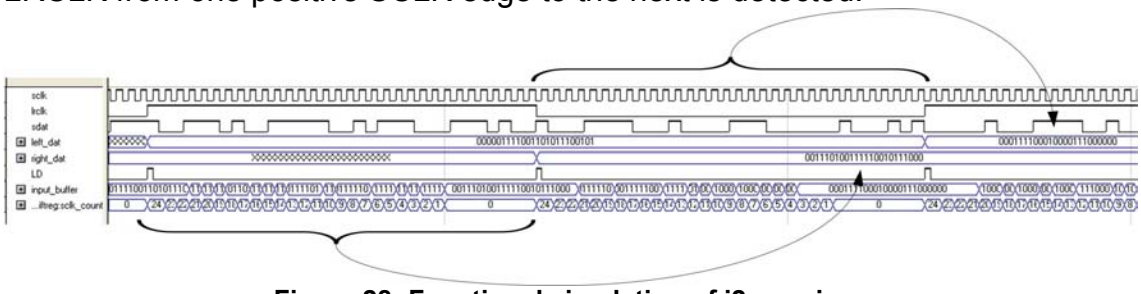


Figure 28: Functional simulation of i2s_recieve

LRCLK difference is indicated by a high LD. When LD is high the input_buffer is copied to the respective output. This is indicated by arrows in the figure. When LRCLK is low the input_buffer contains left data, else the data is from the right channel.

Filling of the input_buffer is done from the first positive SCLK edge after the LD goes low, and 24bits ahead. The input_buffer is indexed by the sclk_count signal.

In the simulation the SDAT is random data. It is shown that the output data is indeed equal to SDAT in the described interval.

6.3 Level meter

The level meter is producing a visual effect and can be used as a diagnostics tool. The meter is comparing positive audio values (MSB is low) to a fixed number and then turning red LED's on according to the value. The concept is illustrated in Figure 29.

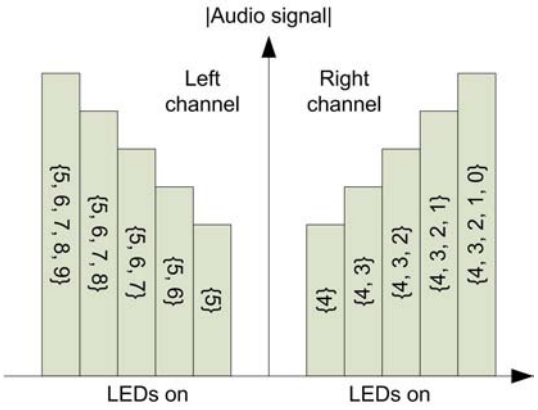


Figure 29: Level meter concept

Numbers in the columns refer to the reds LED on the DE1 board. The values compared to are chosen only on the basis of "what looks good", and are not necessarily linear. Only comparison on every 10th sample is performed.

7. Conclusions

Development of the digital stereo is complete. Unfortunately the class D power amplifier is failing. The exact reasons for this are still uncovered. PCB, hardware setup and soft setup are examined and no apparent reasons are found.

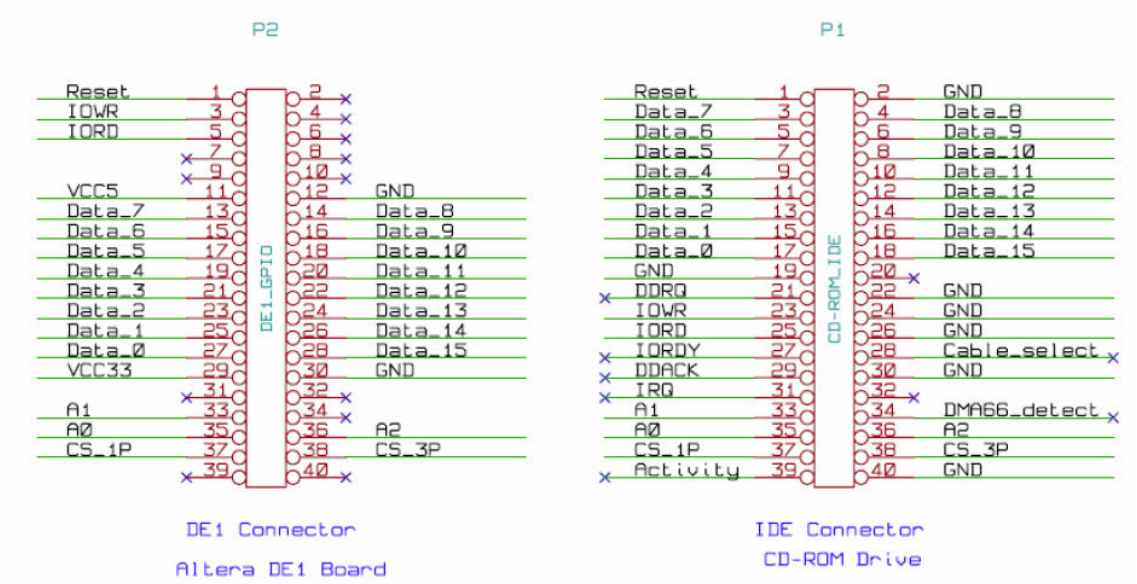
Århus, Denmark the 21. of August 20, 2007 and the project is ended.

8. Further development

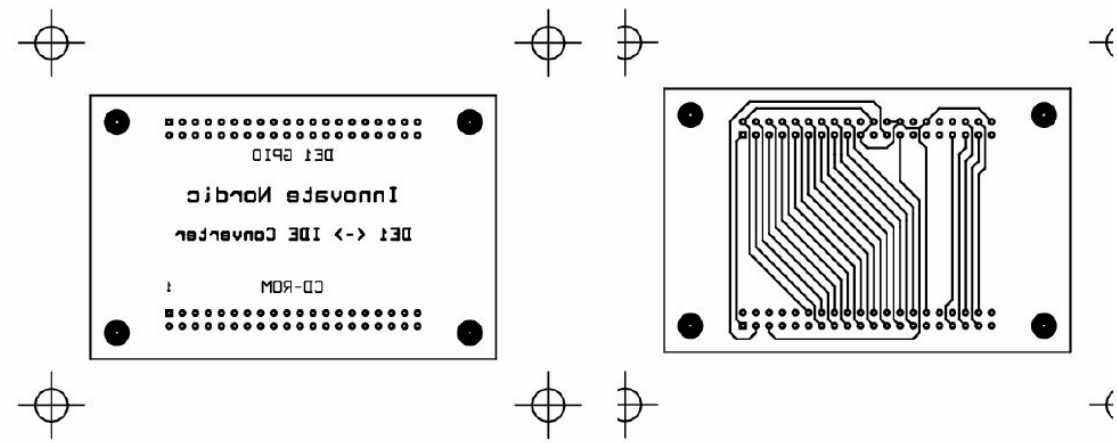
The project continues. Next step (aside from fixing the class D problem) is to digitize the crossover filters. Also the amplifiers are considered moved to the loudspeakers. If this is done, the possibility to network the speakers with each other and other devices comes naturally. Also other input devices are a natural extension to the project.

A Electrical drawings

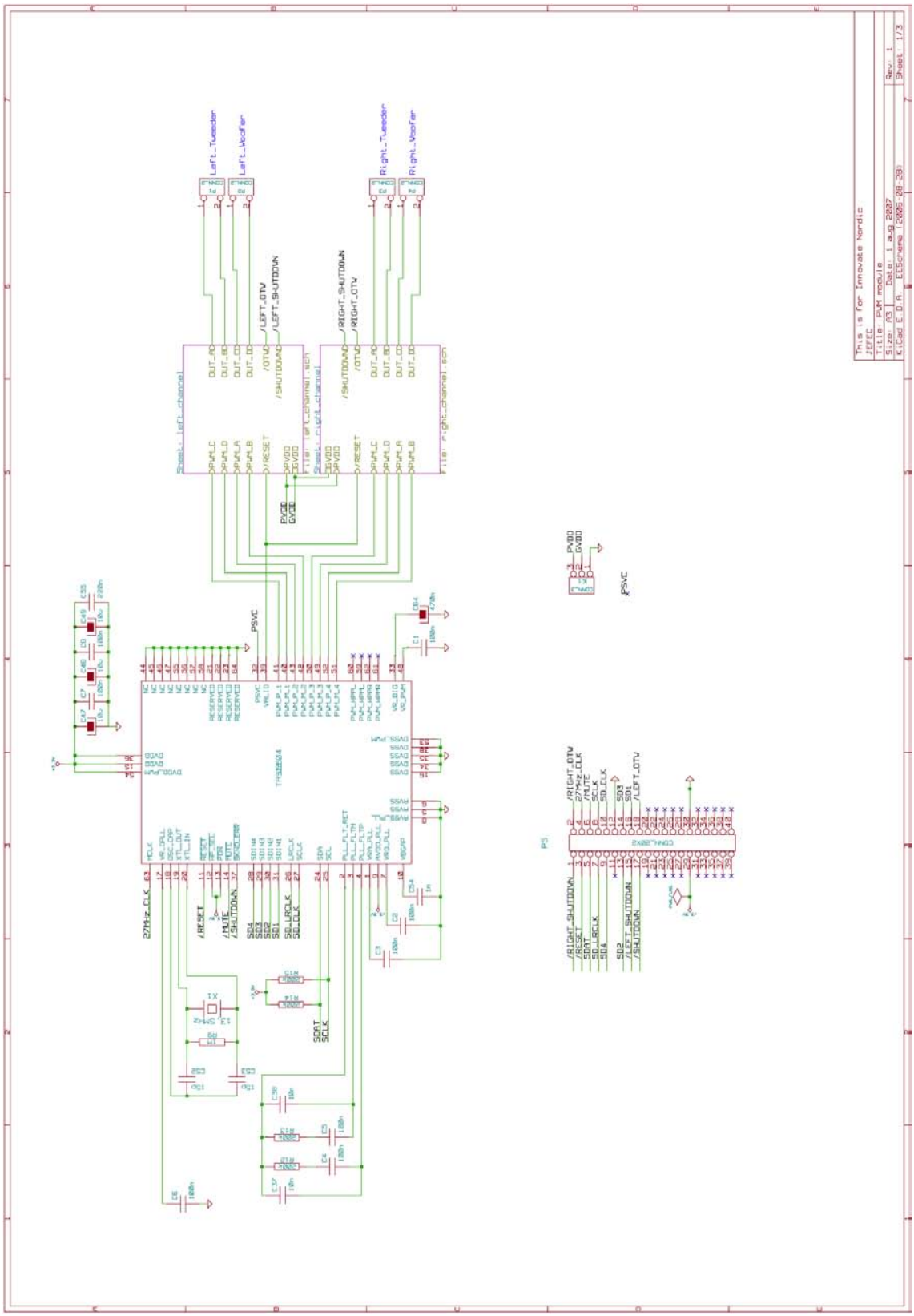
IDE ↔ DE1 interface schematics

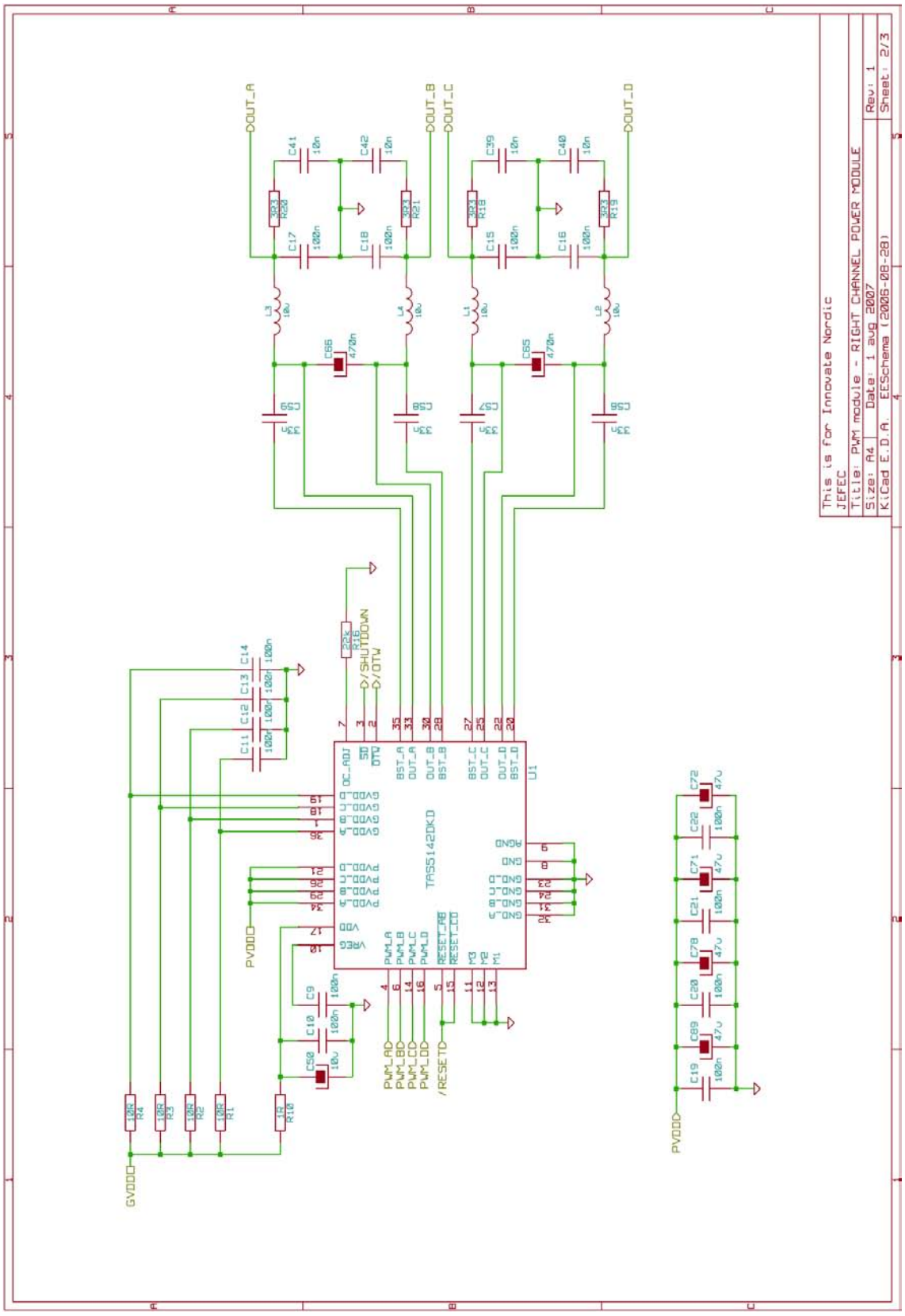


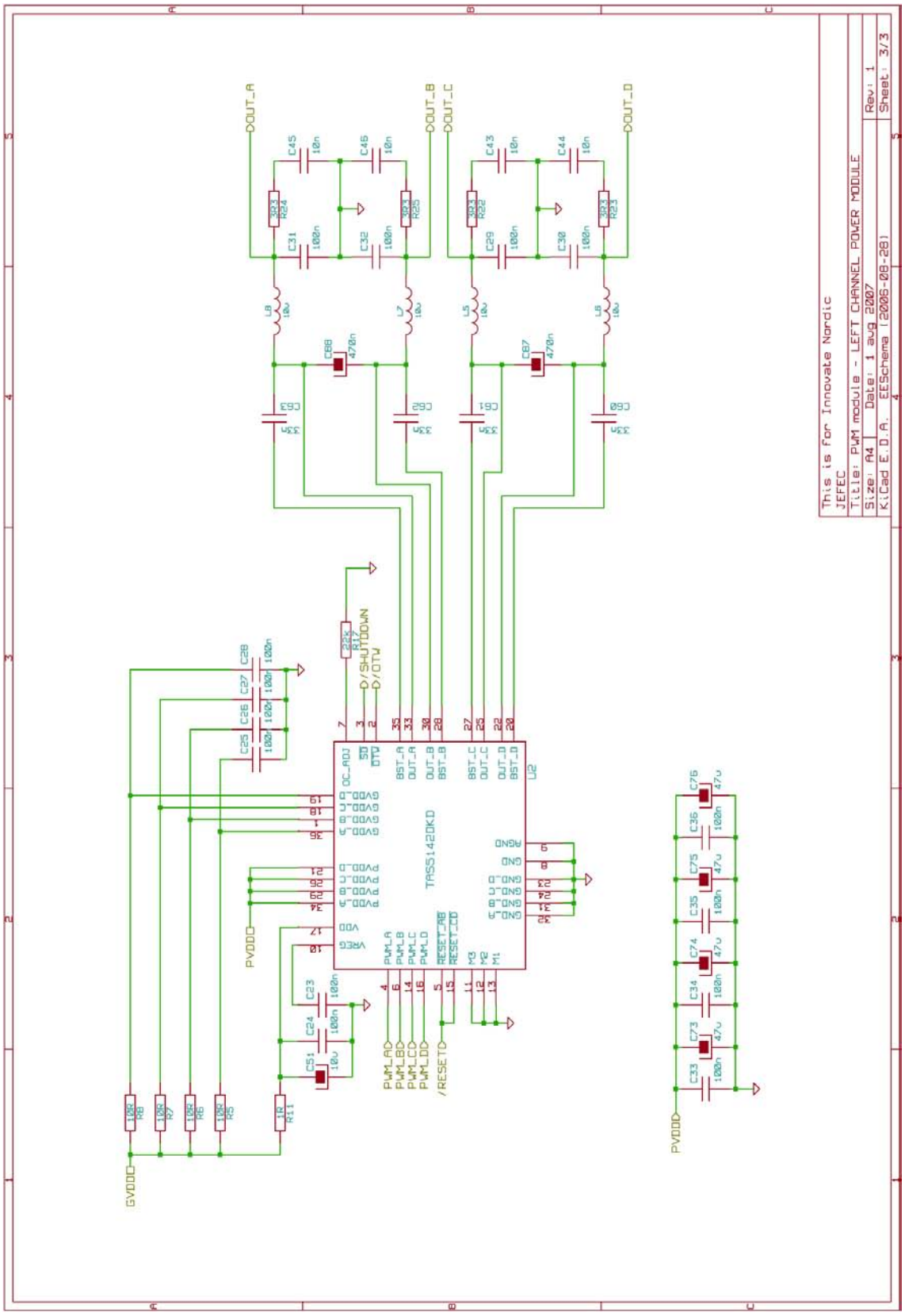
IDE ↔ DE1 interface PCB layout



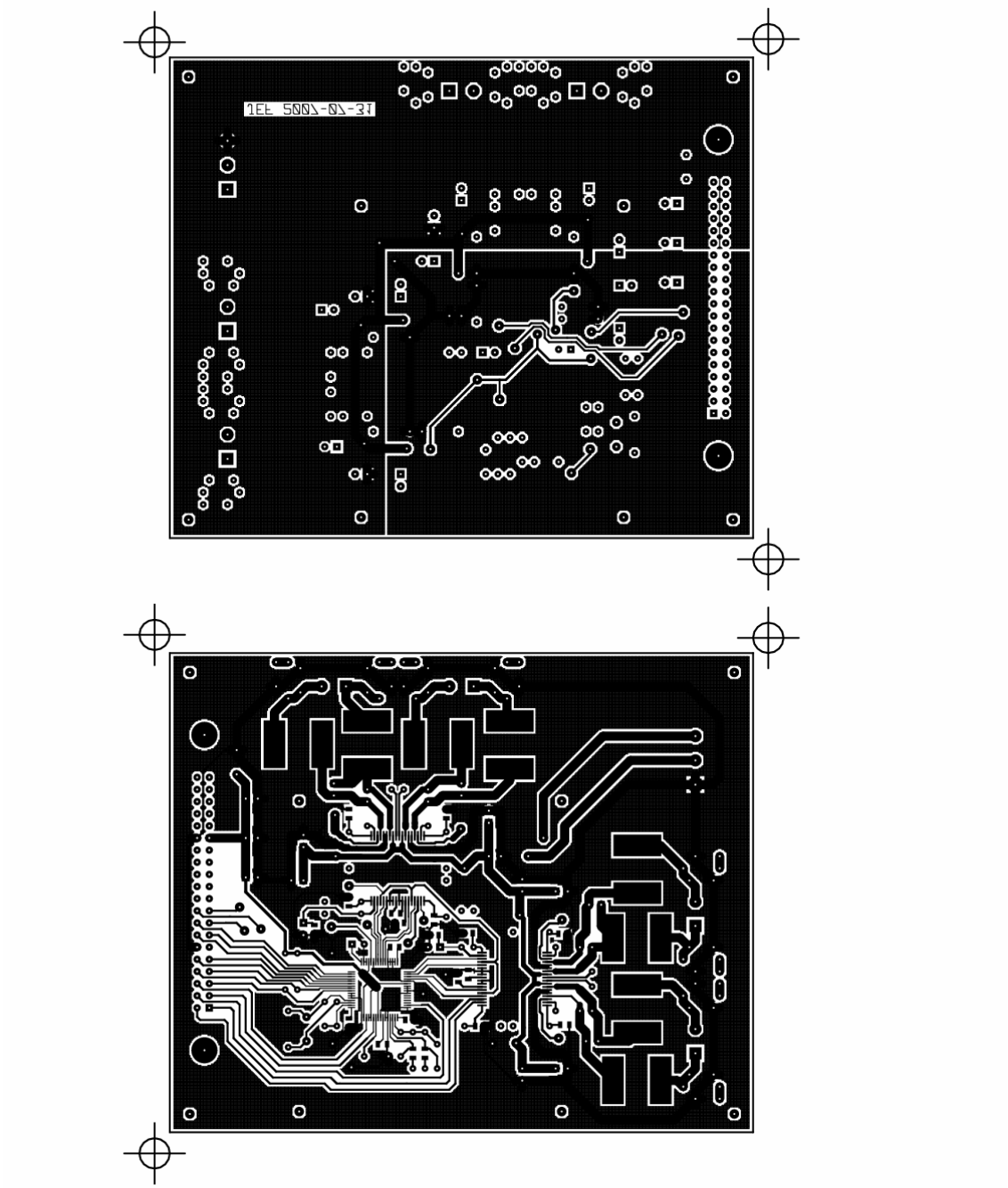
Power amplifier schematics



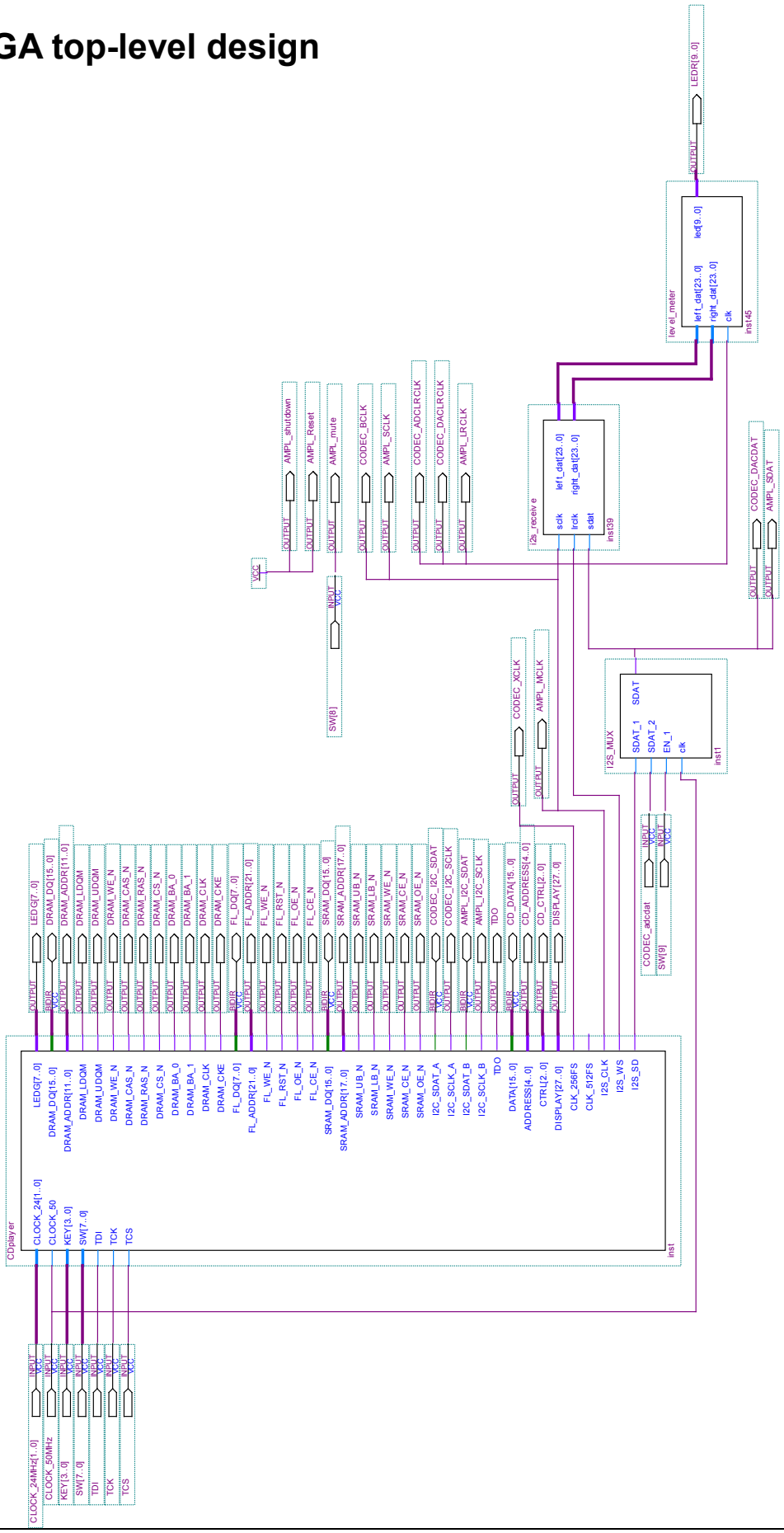

This is for Innovate Nordic
JEFEC
Title: PWM module - RIGHT CHANNEL POWER MODULE
Size: A4 Date: 1 aug 2007 Rev: 1
KiCad E.D.A. EESchema (2006-08-28) Sheet: 2/3
TAS5142DKD
GVDD
PVDD
/SHUTDOWN
/OTW
PWM_AD
PWM_BD
PWM_CD
/RESETD


This is for Innovate Nordic
JEFEC
Title: PWM module - LEFT CHANNEL POWER MODULE
Size: A4 Date: 1 aug 2007 Rev: 1
KiCad E.D.A. EESchema (2006-08-28) Sheet: 3/3
TAS5142DKD
U2
DOUT_A
DOUT_B
DOUT_C
DOUT_D
PVDDD
GVDDD
/SHUTDOWN
/OTW
/RESETD
PWMLAD
PWMLBD
PWMLCD
PWMLDD

Power amplifier PCB layout



B FPGA top-level design



C References

1. ANSI standard: AT Attachment with Packet Interface Extension (ATA/ATAPI), Rev.18, 1998.

2. IEC standard: Compact disk digital audio system (IEC60809), “The Red book”, 2nd edition, 1999-02.

3. Datasheet: Texas Instruments TAS5142 Stereo Digital Amplifier Power Stage, MAY 2005.

4. Datasheet: Texas Instruments TAS5504 4 Channel Digital Audio PWM Processor, October 2004.

5. W. Marshall Leach, Jr.: Introduction to Electroacoustics & Audio Amplifier Design, 3rd edition, Kendall/Hunt Publishing Company, 2003.

6. Maxim Application note: Class D Amplifiers: Fundamentals of Operation and Recent Developments, Dec 2006, Note # 3977.

7. Philips manual: CDD3610 Command Specification, Version 1.2 (final), APR 2007, Ref # AHR-66-MJS-0001.

8. Small Form Factor Committee Specification: ATA Packet Interface for CD-ROMs (SFF-8020i), Rev. 2.6, JAN 1996.

Interim Project Report

Project Name: Efficient Implementation of SSB demodulation, using multirate signal processing

Team Name: Tema Aliasing

Team Members: Martin Lindberg

Email Adress: mlch03@kom.aau.dk

Contact No: +45 24 45 17 19

Instructor: Peter Koch - pk@es.aau.dk

Innovative nordic

0.1 Introduction

The main goal for introducing Software Defined Radio (SDR) is the reduction of the analog hardware in the system. Software radios define an emerging technology, thought to provide a flexible radio system, reconfigurable and reprogrammable by software [1]. The advantages of SDR can be stated as follows.

- Flexibility: Changing the functionallity in the receiver can easily be done by a software updating, instead of replacing or modifing existing physical hardware like analog electronics.
- Cost: The total cost are together with power one of the biggest technical issues facing developers of SDR.
- Time-to-market: The prototype development of a SDR can be done with implementation of FGPA's, which makes the time-to-marked fast.
- Size: Analog components will be replaced by digital hardware, i.e. the physical size of the receiver will be reduced.
- ADC: The ADC must have a high dynamic range and sampling rate, this are limiting factors that determining the maximum achievable data rate of the receiver.

In software radio it is desirable to move the analog-to-digital conversion as close to the antenna as possible, as in the ideal software radio [6]. This is shown in figure 1, but is unfortunately not possible yet due to limitations in todays hardware technology [4].

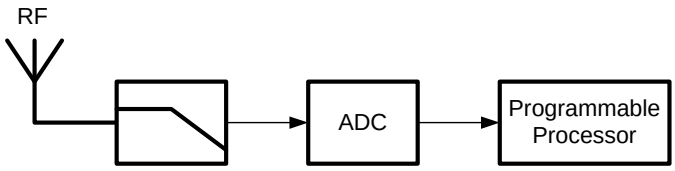


Figure 1: The ideal software radio architecture, where the digital processing starts at the RF range [4].

A part digitalization of a radio receiver will cause incorporation of software in the system and new hardware as well. The rapid development of high-speed analog to digital (ADC) converters

and large field programmable gate arrays (FPGA) allows designers to design compact solutions that were unthinkable a few years ago. Digital signal processing (DSP) with affordable high performance makes the possibility for mapping of the analogue part of the receiver in to the digital domain reachable.

The basis application point in this report is to redesign the receiver part of a current transceiver, which is depicted in the simplified block diagram in figure 1.2.

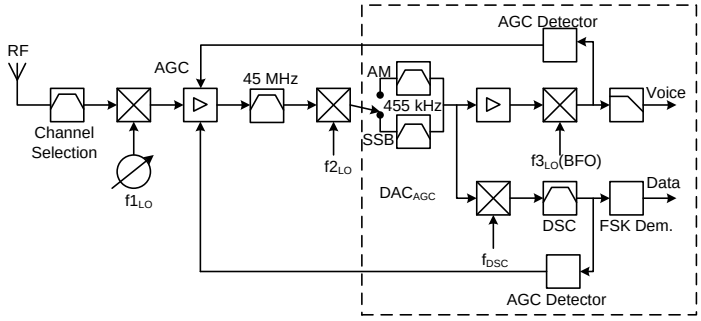


Figure 2: Simplification of a receiver system. The blocks inside the square are the parts of focus in the thesis.

0.2 Problem description

In the preceding, the implementation of software radio into an existing system has outlined that there is a need for investigation. Since the tasks of transforming a part of the receiver from the analog domain into the digital hardware is not a trivial task, there is a need for analysing these tasks. These are two coherent tasks, the first is the mapping of the analog circuitry into the digital domain, and the second is the design of a system architecture where the algorithm can be implemented.

Since this work is an existent system, the function basis is a priori information from the analog domain. This implies that there is a need for functions to be synthesized into the digital domain. Knowing about these existing approaches, considerations can be done in order to optimize the algorithms instead of developing new ones.

The receiver should perform a real-time SSB demodulation using the traditional heterodyne approach, and still be able to keep the specification within the limitations set by ETSI. A software radio for a Super-Heterodyne receiver should be able to handle a large dynamic range, for which reason an AGC is needed. Since the aim of this work is to consider a receiver design with the ADC placed after the second mixer, the second IF must be chosen carefully. Before the final demodulation re-produces the baseband output signal, the modulated signal is filtered to make the last selection in the chain of the receivers. This is done in several steps through the chain to avoid adjacent channel, and image response.

0.3 Design considerations

This section will state the design considerations as an intro to the algorithm chapter. We have the possibility to lower the second intermediate frequency, from 455 kHz to a new wanted frequency. There are mainly two things to be aware of when lowering this frequency, the bandwidth of the spectrum which is assumed to be band limited (attenuated by 70 dB) to ± 40 kHz with fc as the center frequency of the spectrum. The other important consideration is the down-sampling factor which is wanted, to be an integer multiplum of the samplings frequency. In the design considerations, we consider the block diagram shown in figure 3.

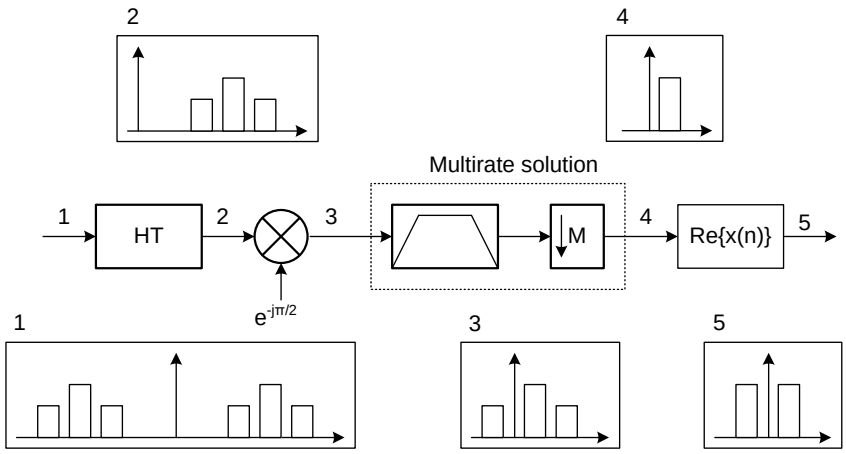


Figure 3: Block diagram showing the simulated scenario. The solution space is surrounded by the (a)dotted box, indicating that there are several possibilities to decrease the computation requirements, within this box.

Hilbert Transform

The Hilbert transform is applied to the sampled incomming signal, by using a two path half-band filter that we modulate to the quarter-sample rate, as shown in figure 4. When applying the Hilbert transform we zero out all negative frequency. Since the half of coefficients in a half band filter are zero, we can cost free utilize the inherent properties of down-sample by a factor of two by skipping the zero multiplications. This results in a throughput of four times the intermediate frequency.

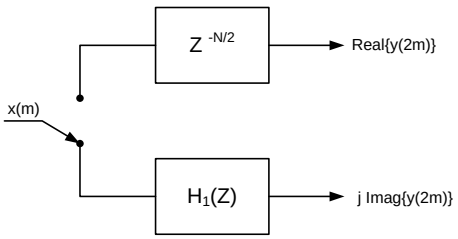


Figure 4: Two-path Hilbert transform filter.

Figure 5 shows the impulse response of the two-path Hilbert transform filter, where the upper figure is the real part of the filter. Since the upper part is a delay of $N/2$, where N is the length of the filter, the real path impulse response is simply a delayed version of the impuls.
The impulse response in the lower part of figure 5, will shift all the negative frequency components by $+90^o$ and the positive by -90^o.

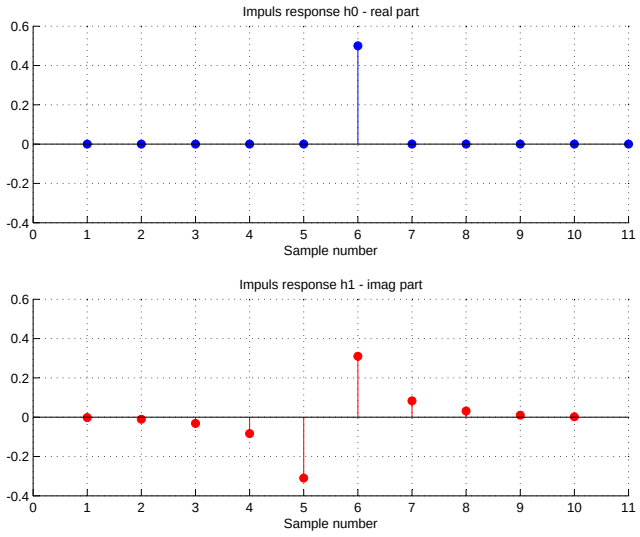


Figure 5: Impulse response of the upper and lower path of the HT. We see that the lower impulse response correspond to the properties of the HT.

Another recursive method to perform the Hilbert transform, is the a recursive method based on the two-path all-pass filters [3]. This recursive structure can be performed with a side band attenuation of 80 dB using only four coefficients.

Down-conversion

Moving the wanted channel to base-band with the effect of the quadrature is cost-free in a hardware solution, and makes is less computational demanding . The output of the quadrature is shown in equation 1 to 4, where n is the sample number.

$$I \quad \text{for} \quad n = 0, 4.... \tag{1}$$
$$Q \quad \text{for} \quad n = 1, 5.... \tag{2}$$
$$-I \quad \text{for} \quad n = 2, 6.... \tag{3}$$
$$-Q \quad \text{for} \quad n = 3, 7.... \tag{4}$$

In hardware this corresponds to only choose the right path and change sign this solution should be compared to a CORDIC solution, which is computational expensive.

FIR Filter Design

The complex band-pass filter requirements to obtain the wanted selectivity are stated in table 1, where the transition band (Δf) is the guard band that separates the channels.

Fs	=	4 fc
Δ f	=	600 Hz
Stopband Attenuation	=	80 dB

Table 1: Filter requirements in order to obtain the wanted selectivity.

The required order of the filter is highly depended on the samplings frequency, and since we have chosen to use the properties of quadrature sampling the sampling frequency is dependent of the second intermediate frequency. Figure 6 shows an estimate of the filter order, when using the window and the Parks-McClellan (PM) method for designing the filters.

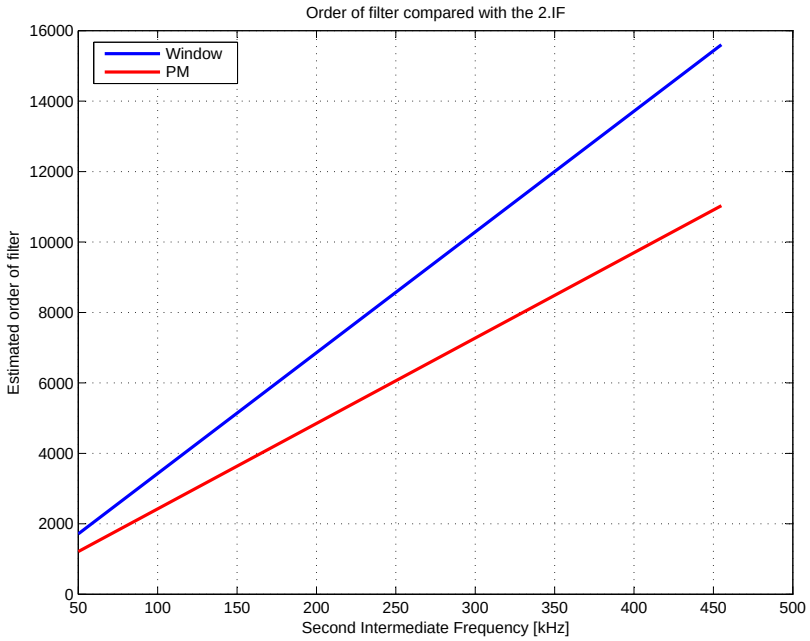


Figure 6: The estimated order of the filter, in order to obtain the wanted filter characteristic compared to the intermediate frequency, where the red line is based on Parks-McClellan filter design, and the blue line is base on the window method.

FIR Polyphase decomposition

We can take advantage of decomposition the FIR filter, when using down-sampling, by utilizing the properties of the noble identity. Before decomposition the filter, we need to known the down-sampling factor M, since the output rate need to be 12 kHZ, we use equation 5 to calculate M.

$$M = \frac{4fc}{12\text{ k}} \tag{5}$$

Figure 7 shows how the down-sampling factor increases when increasing the second intermediate frequency. The required number of sub-filters in a poly-phase decomposition of a FIR filter are equal to the down-sampling factor M.

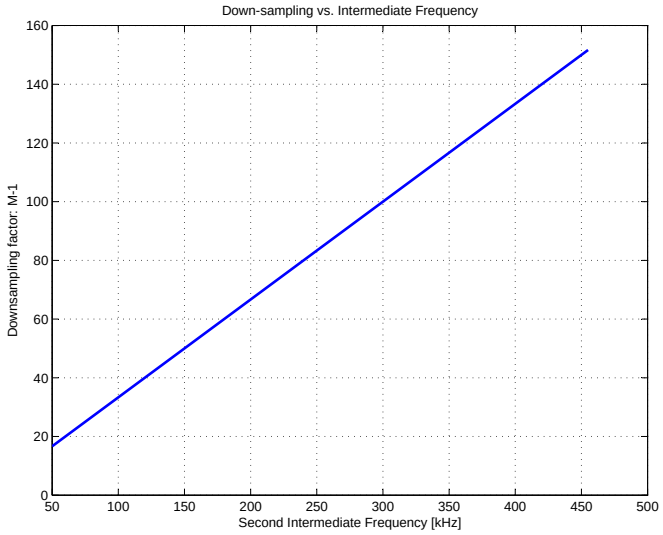


Figure 7: The down-sampling factor is linear dependent of the second intermediate frequency, which means that when lowering the frequency a lower down-sampling factor is required.

If we choose to lower the second intermediate frequency the down-sampling factor will decrease as well, like the sampling frequency and the order of the filter. Since we only need to calculate each sub-filter at each incoming samples, we can calculate the required number of operations in each sub-filter independent of the sampling frequency.

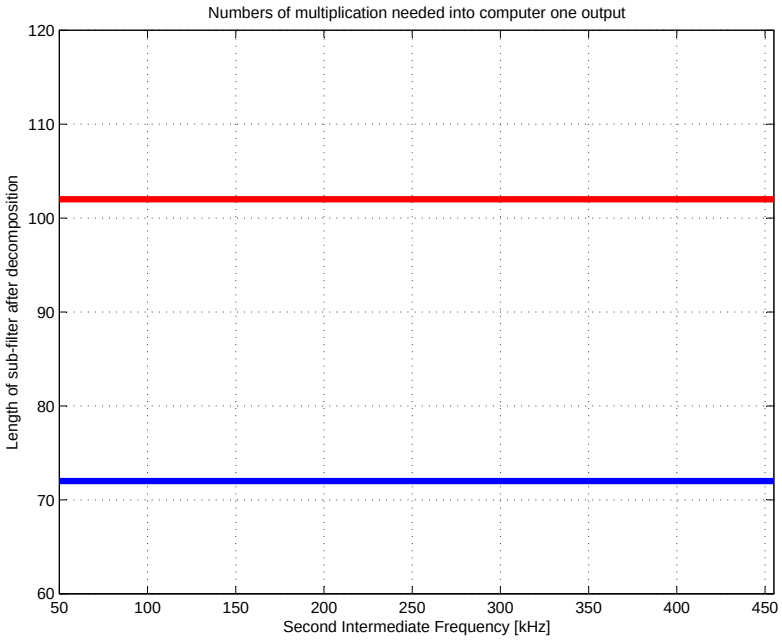


Figure 8: The red line is the window function that would require 102 operations in each sub-filter. The blue line is the PM, which only requires 72 operations in each sub-filter.

In figure 8 we see that when using the window function we need 102 computations in each sub-filter is needed, but only 72 when using the PM filter design method. It can be argued why this is the case, and that PM has the disadvantage of having more pass-band ripple, but we will not look into this now. Since the poly-phase filter still run at full-rate, we want to see if it is possible to make several down-sampling filter that runs at a lower rate, and still obtaining the wanted selectivity.

Two-path recursive All-pass half-band filter

The two-path recursive all-pass filter is able to down-sample by a factor of two in each section [2]. This will cause the down-samplings factor to be a power of two. When down-sampling by a factor of two in each section, the down-sampling factor need to be a power of two as shown in equation 6. We know that the spectrum is band limited within ± 40 kHz. The output at audio has as sample frequency of 12 kHz, which corresponds to that the second intermediate frequency is not allowed to be lower than 96 kHz, and five section of down-sampling.

$$\frac{4 \cdot fc}{2^N} = Fs_{out} \tag{6}$$

$$\frac{4 \cdot 96 \text{ kHz}}{2^N} = 12 \text{ kHz} \tag{7}$$

However, we can not just cascade couple all five sections, this would cause aliasing in the wanted spectrum. After two down-samplings sections the spectrum need to be band limited. This is done by a recursive all-pass filter, which is a more efficient filter than the regular IIR filters [3]. The spectrum of the first half-band filter can be seen in figure 9, while the coefficients are shown table 2.

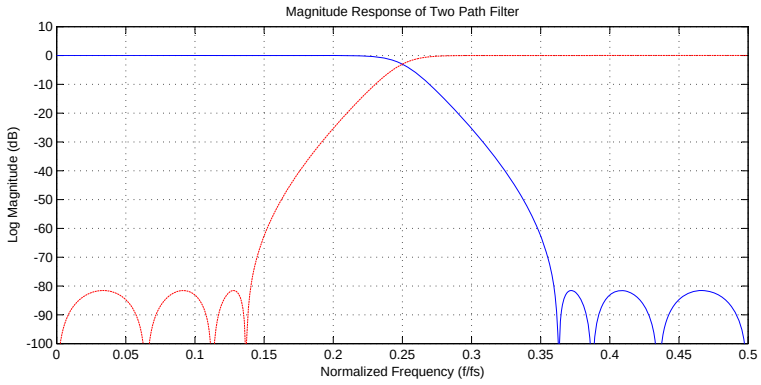


Figure 9: The first and second half-band filter, when down-sampling the signal by a factor two. The red line is the filter plotted as high pass filter, and is only a matter of sign changing [3]. This is included in order to show the possibility in this filter types.

In table 2 we see that only three multiplications needed to calculated the filter, two multipliers in the first path and one in the second path.

Path-0	Polynomial Coefficient
Filter-0	[1 0 0.07638269328882]
Filter-2	[1 0 0.69878248805530]

Path-1	Polynomial Coefficient
Filter-1	[1 0 0.30208200300176]

Table 2: Coefficients for the first half-band filter.

The last filter in the chain, need to be a complex filter, since the spectrum is complex and we want to obtain the final selectivity. This is done by up-converting a FIR filter. To obtain the wanted selectivity we need 201 taps.

The last recursive half-band filters are not plotted in this section, but will be pressent in the simulation results. The required number of coefficients is shown in table 3. Notice that we use five coefficient in block-3 instead of three, to shown how the number of coefficient change the steepness of the filter. This is further elaborated in the next chapter.

Block-1	Block-2	Block-All	Filter-3	Bllock-4	Block-5	Block-FIR	Total
3	3	13	5	3	3	201	231
48	24	104	20	6	3	201	406

Table 3: The second row shows the number of coefficients used in each section. While the third row shows the required number of multiplications needed in order to produce one output sample.

In table 3 the required number of coefficients is stated. To produce one output a total sum of 231 coefficients and 406 multiplications are required, as stated in table 3. This should be compared to the poly-phase implementation, where the entire filter length N need to run in order to deliver one output sample. Figure 6 indicates that the filter length is about 2000.

0.4 Simulation results

With the design consideration in mind we are able to design and simulate the system function to see if the requirements stated can be fulfilled. The calculation of the filter coefficient needed in the All-pass sections, is based on [7]. In this algorithm several different filter types has been used:

- Half-band Hilbert Transform
- Two-path Half-band filter
- Recursive All-pass filter
- Complex FIR filter

There are several possible constellations for combining the filters into this application, but in this algorithm most of the multirate techniques are used, in order to see the advantage in each filter design procedure, as well as the reduction in the computational complexity.

0.5 Simulation setup

To get a quick overview of the algorithm, figure 10 shows the different blocks, as designed in section 0.3. Since we are working on a multirate system the sampling frequency is varying from block to block. At the block diagram each incoming and outgoing frequency is stated to give an overview of the systems computational load.

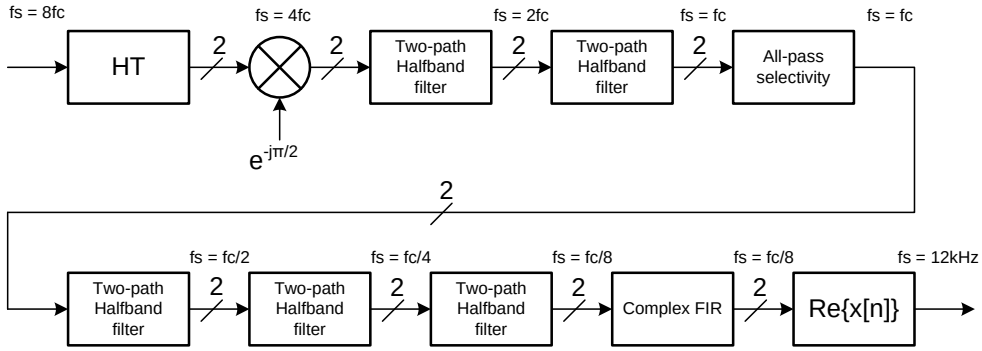


Figure 10: Block diagram of the algorithm showing the different filter blocks use. The number two on the arrow connections indicates that the signal is complex.

In order to fulfil the specification to the system specified by ETSI, we use the frequency component located relative to the carrier as the specification stats. Another important property to be aware of, when using multirate system is unwanted aliasing components that could rise in the wanted spectrum. In order to take this problem into account when simulating the system, we locate two frequency components at fc$\pm$13 kHz. To see whether the wanted channel is maintained and leaved unchanged though the system, we simulate the wanted channel with four modulated frequencies from the boundary of 300 to 2700 Hz.

Fs in:	8 fc
Fs out:	12 kHZ
Test signal:	fc(0.3,2.7) kHz
Test speech:	BW: 300-2700 Hz
Upper adj. signal:	fc(+4,+5,+8) kHz
Lower adj. signal:	fc(-1,-2,-5) kHz
Aliasing signal:	fc+13 kHz
Aliasing signal:	fc-13 kHz

Table 4: Test signals used in the simulation.

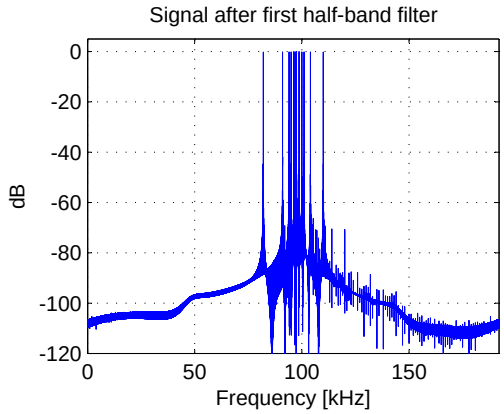


Figure 11: The SSB spectrum where the wanted signal is modulated with a carrier of 96 kHz, the other frequency components are unwanted

Hilbert transform

The Hilbert transform is designed by quadrature modulation of the low-pass coefficient, and then implemented as a two-path half-band filter. The implementation of a two-path Hilbert transform makes it possible to obtain a cost free down-conversion by 2-1, since half of the coefficient is zero. The upper path of the Hilbert transform will be a delay of $-N/2$, and the lower path will be the coefficients of the Hilbert transform, by multiplying the following by j we shift the signal by 90^o, doing so will present the signal analytic, given by $x_a = x_{upper} + jx_{lower}$. This can be seen as a one side spectrum, where the negative frequencies are zeroed out, as the right figure in 12. It should be noticed that the spectrum now is complex and asymmetric.

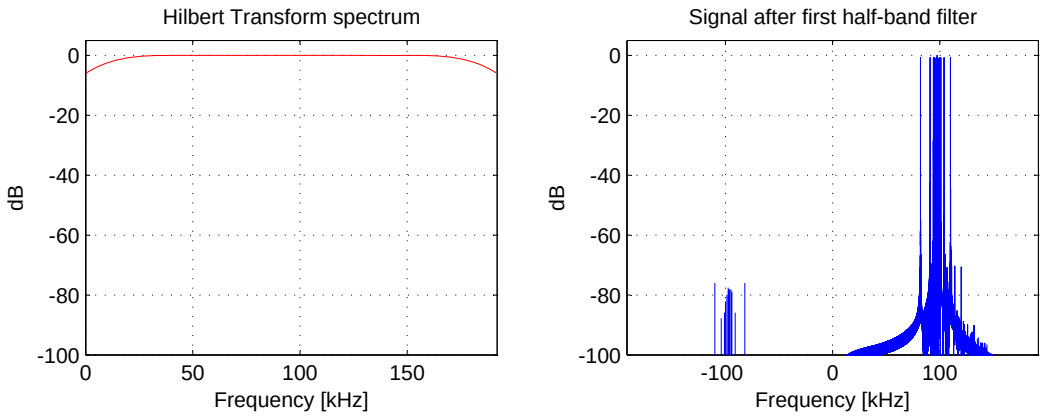


Figure 12: Left figure show the spectrum of the Hilbert transform, and the rigth the output since the Hilbert transform, where the negative frequency are not completely zeroed out but attenuated 80 dB.

Down-conversion

To avoid an expensive solution, we take advantage of using the quadrature effect when doing the down-conversion. This will move the wanted channel down to base-band, as shown in figure 13.

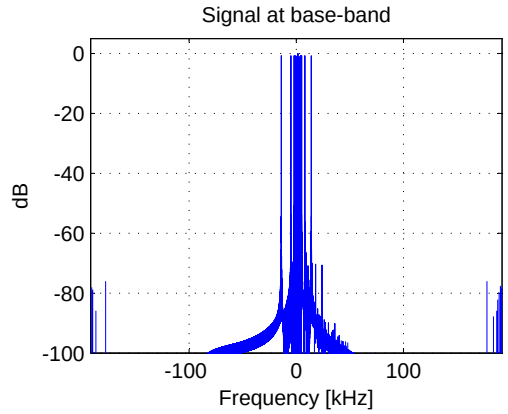


Figure 13: The quadrature down-conversion locate the wanted channel at baseband

Down-sampling

From section 0.3 we use the property of the half-band filter by down-sampling by a factor of two in each section. Since the incoming sample rate is four times the intermediate frequency, we need to down-sample by a factor of 32 to obtain an output of 12 kHz. This can be obtained by using a cascade of five half-band filters. When designing these half-band filters we need to pay attention to the fact that aliasing can occur if the signal not are band limited when doing the down-sampling. It is possible to down-sample by a factor of four before problem with aliasing occurs. A down-sampling by a factor of four gives an output frequency of 96 kHz which will force a band limitation before doing another 2-1 down-sampling, otherwise aliasing will occur. The recursive all-pass filter is applied to band limit the signal, so the following half-band filters can down-sample by a factor of 8.

First Half-band filter - (Fs_{in}:384 kHz Fs_{out}:192 kHz)

After the second IF the spectrum is band-limited from the analog domain, and since the sampling frequency is equal to the quadrature, no aliasing problem occurs when doing a 2-1 down-sampling with a half-band filter. The spikes seen in the positive frequency spectrum in the right figure 14 are aliasing from negative frequency spectrum that the Hilbert transform did not remove entirely.

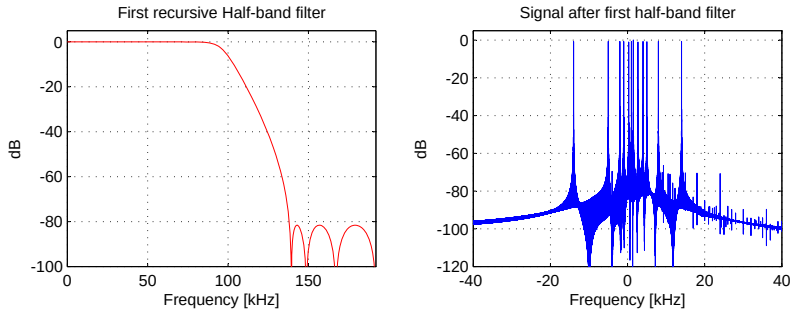


Figure 14: First half-band filter spectrum, and the output from the filter.

Second Half-band filter - (Fs_{in}:192 kHz Fs_{out}:96 kHz)

We can simply apply another half-band filter without care about aliasing since the half incoming frequency is still higher that the spectrum. Figure 15 show the output spectrum from the filter as well as the filter spectrum.

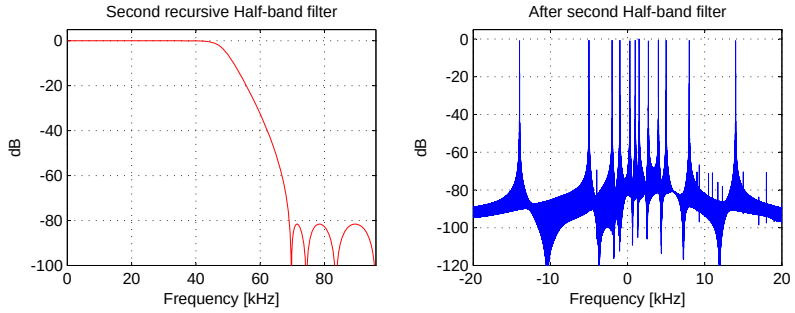


Figure 15: The left figure shows the spectrum of the half-band filter, and the right the output.

Recursive All-pass filter - (Fs_{in}:96 kHz Fs_{out}:96 kHz)

To lowering the sampling frequency by another factor of 2, we need to band-limit the spectrum to avoid aliasing problems. This is done with the all-pass filter that has a low complexity and is still able to have a high out of band attenuation. In figure 16, the spectrum shows the low pass-band and large stop band attenuation for the recursive all-pass filter. We also see that the channels located above the wanted is attenuated, and the only frequency components present are located at fc-1 kHz and fc-2 kHz. The small spikes shown are our unwanted frequency components that are attenuated below 80 dB.

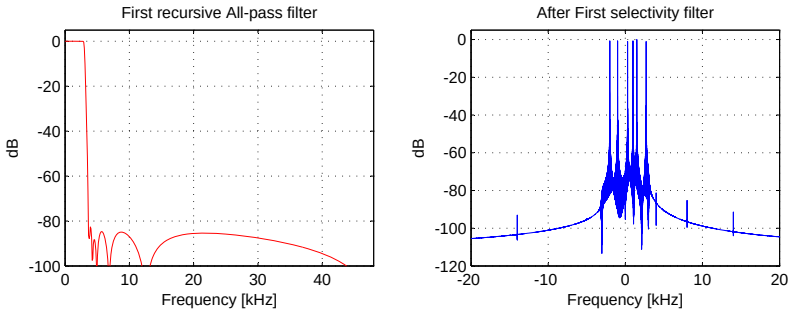


Figure 16: The left figure show the spectrum of the recursice All-pass selectivity filter, and the right the output.

Third Half-band filter - (Fs_{in}:96 kHz Fs_{out}:48 kHz)

The spectrum is now band-limited, so the requirements to the half-band filters are less demanding. The filter spectrum and output from the third half-band filter can be seen in figure 17.

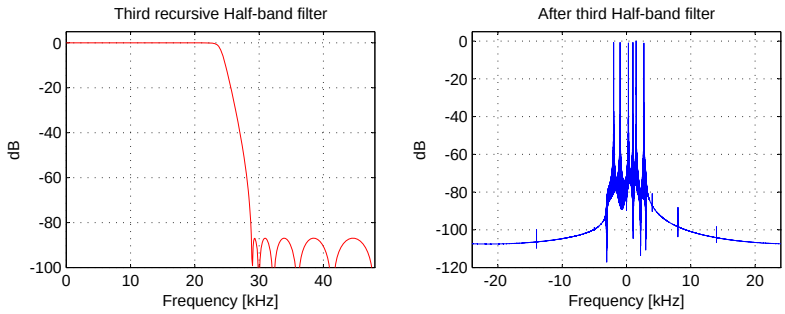


Figure 17: The left figure show the spectrum of the recursice All-pass selectivity filter, and the right the output.

Fourth Half-band filter - (Fs_{in}:48 kHz Fs_{out}:24 kHz)

Figure 18 shows the spectrum and the output of the filter. We see that the unwanted frequency components are almost invisible at this point in the chain, but the two frequency components below the wanted channel are still present.

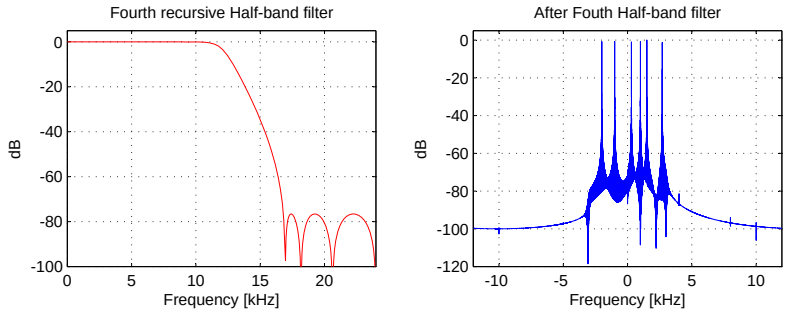


Figure 18: The left figure show the spectrum of the half-band filter, and the right the output.

Fifth Half-band filter - (Fs_{in}:24 kHz Fs_{out}:12 kHz)

The fifth filter is doing the last down-sampling, the output frequency is at this stage 12 kHz, which was the desired. However looking at the spectrum in the right figure 19 we see that two unwanted frequency component is still present, and need to be removed.

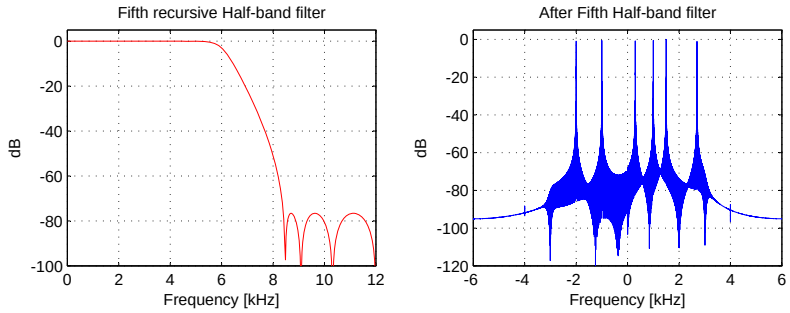


Figure 19: The left figure show the spectrum of the half-band filter, and the right the output.

Selectivity filter - (Fs_{in}:12 kHz Fs_{out}:12 kHz)

The last filter is applied to obtain the final selectivity in the chain, we have complex down-converted the signal, follow by down-sampling. The complex down-conversion leaves us with an asymmetric frequency spectrum, where the wanted channel is located from 300 to 2700 Hz, all that is located below and above this band is unwanted and needs to be attenuated. We need to design a filter that has an asymmetric frequency spectrum, the easiest way to do so is to design a low-pass filter with the cut-off frequency at half the bandwidth of the wanted signal. The low-pass filter is real, but we simply modulate the filter coefficients by a complex phase rotation so the center is located (f1-f2)/2, this leaves us with a complex band-pass filter as shown in figure 20. It can be argued that a complex IIR filter is more efficient. However, the sample rate is reduced to only 12 kHz, we don't consider time as a time constraint at this stage.

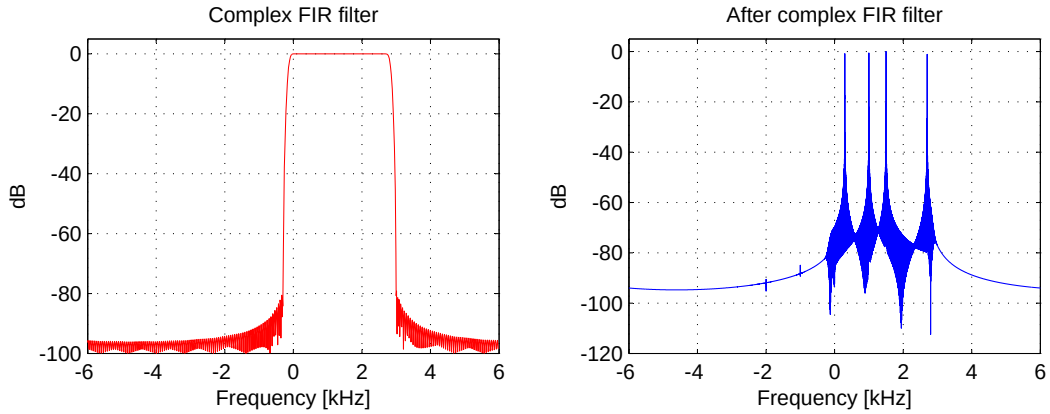


Figure 20: The left show the asymmetric spectrum of the complex band-pass filter, at the right only the wanted frequency component is present now.

0.6 Output result

The real output from the algorithm is shown in figure 21, where all the four frequency components still are present and the out of band attenuation below 80 dB. The final performance test is done by the company xxxx in order to verify the quality of the speech signal though the chain.

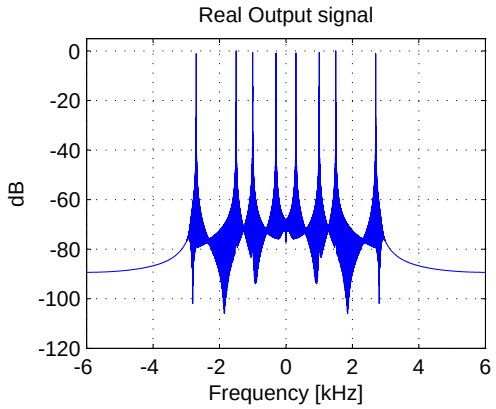


Figure 21: Real output spectrum.

Quality Test

A quality test on the system has been made at algorithm stage. The three SINAD measurement was performed in order to verify the specification from ETSI. A 1 kHz sinus was modulated with the carrier (fc) as the wanted channel, the adjacent channel was located as stated in table 5. The adjacent channel was applied with an amplitude 50 dB above the wanted channel.

Test nr.	Test signal	SINAD	Distortion
Test-1	((fc-1)+(fc+4)) kHz	40 dB	N/A
Test-2	fc-2 kHz	40.8 dB	0.9%
Test-3	fc+4 kHz	45.3 dB	0.54%

Table 5: Results from quality test performed by xxxx

The specification stated that the SINAD measurement should be above 20 dB, and with a distortion of maximum 1%. The performance test shows that the adjacent channel located above the wanted in the frequency spectrum performed best with 45.3 dB SINAD. This was expected as shown in figure 19 and 20. The test verified the algorithm, and we can now consider the implementations possibilities in the design space exploration.

0.7 Sub-conclusion

The simulation shows that the algorithm is able to demodulate a SSB signal and still fulfil the requirements from ETSI. There are several solutions to demodulate the signal with multirate signal processing, this solution has tried to combine different filter type and obtaining a low complexity compare to the known legacy solution. In this solution the company is able to adapt the algorithm to fulfill their requirements, if it turns out that the sampling frequency is to high it can be lowered by a factor of two, without any lost performance since a factor 2 down-sampling is inherent in the Hilbert transformation, if it turns out the Hilbert transform not is efficient enough and the down-sampling is not needed, the company should look into the design of a recursive Hilbert transform where an out of band attenuation of 80 dB, can be achieve canwith only 4 coefficients, which will lower the complexity sufficiently. The quadrature solution for down-conversion was the obvious choice for down-conversion, since this is a matter of shift between the I-channel and Q-channel, and changing the sign of the channels. The recursive two-path half band filter makes it possible to lower the complexity since half the coefficients is zero, however the filter should be able to obtain a high out of band attenuation, and by utilizing the noble identity we split it into a poly-phase decomposition that makes the 2-1 down-sampling cost free. If these recursive half band filter is unwanted they can be replaced by a FIR filter decomposed into a poly-phase structure, this increase the complexity but avoid the feed back loop inherent in the recursive structure. One of the interesting filters in this algorithm is the recursive all-pass filter. This filter should be compared to IIR filter like the Elliptic, Chebyshev e.g., the complexity in this filter is lower than in the traditional IIR filter which makes it suitable when we having a time constraint.
In the current algorithm we have purely chosen to do the down-sampling with half band filters, which would require a final filtering to obtain the wanted selectivity, the filter needs to be complex since the spectrum is asymmetric. We have done the complex filtering with at modulated low-pass filter centred at half the band-width of the wanted spectrum.
The SINAD quality test of the algorithm was preformed and the result is stated in table 5. The results showed that the overhead was above 20 dB SINAD, which indicate how effective the multirate rate filters can be.

0.8 Wordlength analysis

The disadvantage by using a DSP based implementation in a multirate base system is due to the need for FIFO buffers within between each block. For this reason an FPGA will be an obvious choice for prototyping development, as stated below.

- Variable clock rate
- Different word length
- No, control calculations

The specification were fulfilled, and we will start to optimize the algorithm to fit into a suitable architecture. In this algorithm the selectivity and sensitivity are the two important parameters, which have the highest priority when we compare it to the wanted performance, this mean that we accept a larger area in cost of performance.
There are two parameters that influence the area: the number of functional units and the word-length of the calculations used in each intermediate operation. Since the system is based on several recursive blocks a low number of coefficient, it would be an interesting parameter to see the performance of the blocks at different word length.
In an FPGA implementation, each variable can be customised to produce the best tradeoffs in numerical accuracy, design size, speed and power consumption, which is useful in a multirate system.
By using the word length analysis we will express the cost function purely dependent of the number of bits used in the total system. The new cost function can be seen in equation (8), and are the sum of bits used to present each filter block.

$$\mathrm{Cost}_{\mathrm{bit}} = \sum_{i=1}^{M} b_i \tag{8}$$

To simulate a fixed point hardware environment with different word-length in each block we need a high level league like SystemC.

0.9 SystemC

When simulating the algorithm in Matlab, the precision is approximate infinite. Unfortunately the hardware cost follow the precision trend toward infinity. In order to minimize cost in a FPGA based system, we want to see the performance of the algorithm, in a fixed point hardware enviroment. To model the behaviour of fixed point hardware, the tool SystemC[5] is used, which is a hardware description language like VHDL and Verilog, but is more aptly described as a system description language.

In SystemC it is possible to model the fixed point bit accurately for each intermediate variable in the functions used within the algorithm. Furthermore, SystemC supports features like modelling difference quantization mode, and overflow behaviour at a high level. This tool makes it possible to simulate a hardware environment, that enable us to see the performance of the algorithm at different word length.

Two data types are used to model the fixed point hardware architecture, one signed and one unsigned. The presentation of the fixed point word length is defined as expressed in equation (9) for signed and (10) for unsigned. These arguments are static and must be known at compile time.

$$\mathrm{wl}_{\mathrm{signed}} = [-2^{(iwl-1)}, 2^{(iwl-1)} - 2^{-(wl-iwl)}] \quad (9)$$

$$\mathrm{wl}_{\mathrm{unsigned}} = [0, 2^{(iwl)} - 2^{-(wl-iwl)}] \quad (10)$$

There are four parameters associated with these data class types, these are listed below:

- wl - Total word length, used for fixed point representation. Equivalent to the total number of bits used in the type.
- iwl - Integer word length - specifies the number of bits that are to the left of the binary point(.) in a fixed point number.
- q_mode - quantization mode, this parameter determines the behavior of the fixed point type when the result of an operation generates more precision in the least significant bits than is available as specified by the word length and integer word length paramenters.
- o_mode - overflow mode, this parameter determines the behavior of the fixed point most significant bits when an operation generates more precision thah the most significant bits available.

0.10 System setup

In order to measure performance in the filter blocks, with different word length, we have to come up with a performance estimate, where comparing the high level Matlab output with the word length reduced output signal. Since we are working at a high block level, we choose to express the performance as the error or total sum of squares as expressed in equation (11). We want to minimize the error, but to set value on an acceptable error is difficult, but we would take a look at the error influence of the filter characteristic, specially with focus on the slope of the filter and attenuation in the stop band.

$$e = \sum_{i=0}^{n} (q(i) - h(i))^2 \quad (11)$$

The integer word length (iwl) is determined by applying a steep response to each block, and to avoid overflow we set a watch flag in SystemC to observe if this occurs. The number of iwl bits for each section can be seen in table 6. This parameter is fixed and we vary the wl from thirty-two to ten bits, with a interval of two.

The hardware accumulator used in the simulation is set to infinity, but this can of cause be simulated with a fixed point bit number if wanted.

0.11 Simulation results

The total sum of squared error depend on the different word length as shown in figure 22. As expected the FIR filter blocks (HT & Complex filter) error are low compared to the recursive

Value	HT	Block1-2	ALL-pass	Block3	Block4-5	Complex FIR
wl	32:-2:10	32:-2:10	32:-2:10	32:-2:10	32:-2:10	32:-2:10
iwl	2	2	3	2	2	2

Table 6: The parameter used for each block simulation in SystemC.

filters. The half-band filters 1 and 2 using only three coefficients, and have equal coefficient which implies the same squared error, as well as 4 and 5. Comparing half-band filter 1,2 with 4,5 the error is almost equal, even when the coefficient are different, but the number of coefficient used are equal, which implies almost an equal error. The longest half band filter (3) have five coefficients, here we seen and increased error compared three coefficient filters. The last filter tested is the recursive all-pass filter, in this filter block the signal is band limited, and some of the selectivity is obtain here. The filter has thirteen coefficients and the error variation is the biggest in the filter chain.

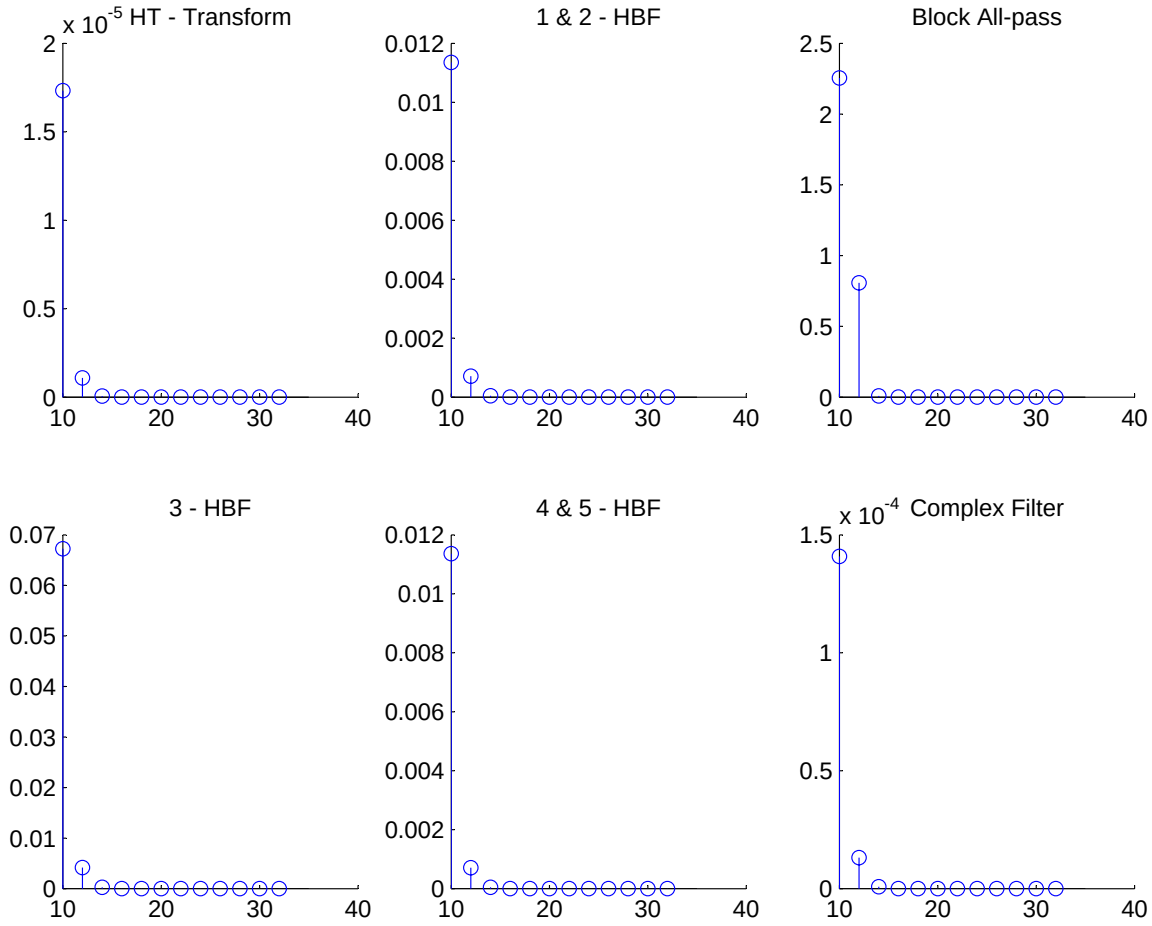


Figure 22: The graphs show the total sum off squared error, for each block. The x-axis represents the number of bits and the y-axis the error. It should be noticed that the filter coefficient in block 1 and 2 are the same, as well as in block 4 and 5.

To see a correlation between the error and the filter characteristic, the response has been plotted for each section. Figure 23 show that the transfer function is presented as wanted from 32 to 10 bit, which was expected since there is only ten coefficient, and the total sum of squared errors was small.

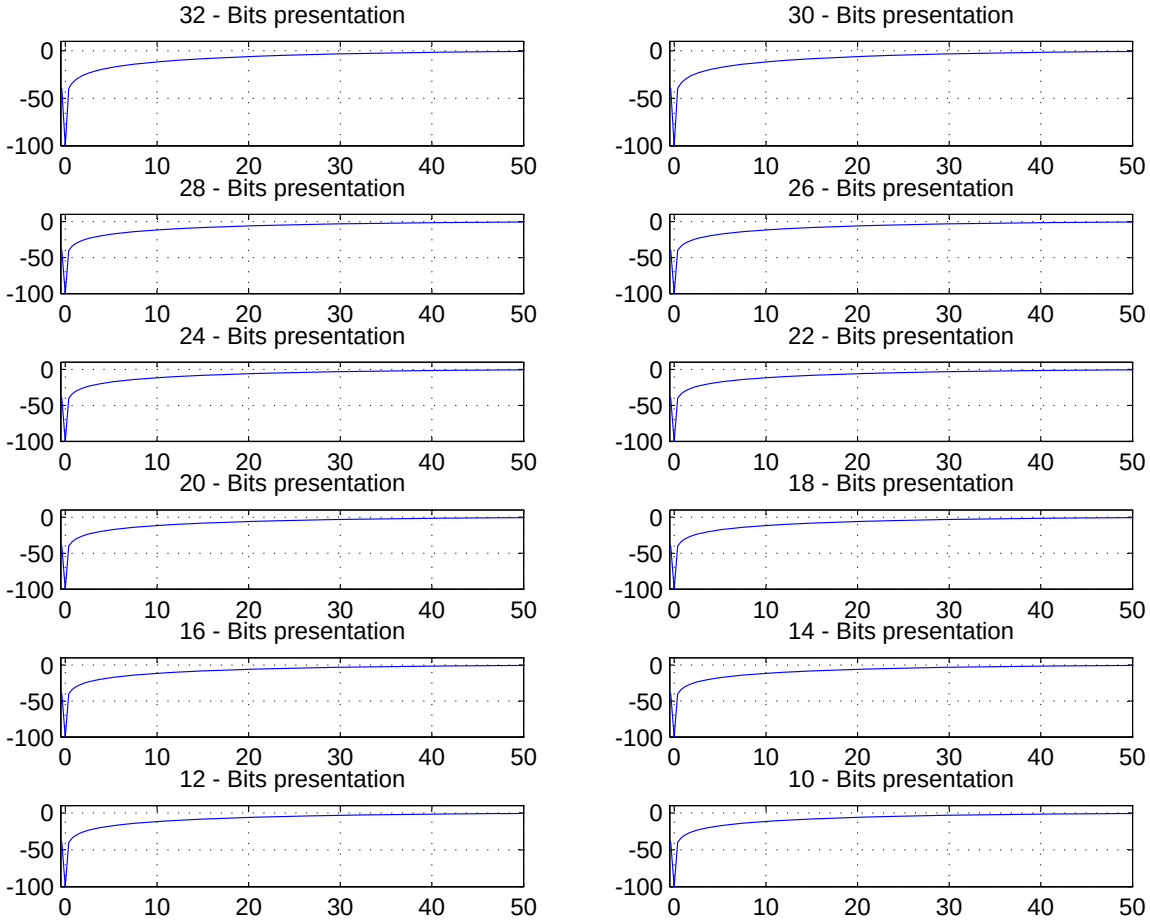


Figure 23: Filter characteristic Hilbert Transform for varying bit representation. The y-axis is in dB and the x-axis in kHz.

Figure 24 show the transfer characteristics for half-band blocks 1 and 2. The red line show the 80 dB criteria, and at least 20 bits is needed to maintain the stopband attenuation.

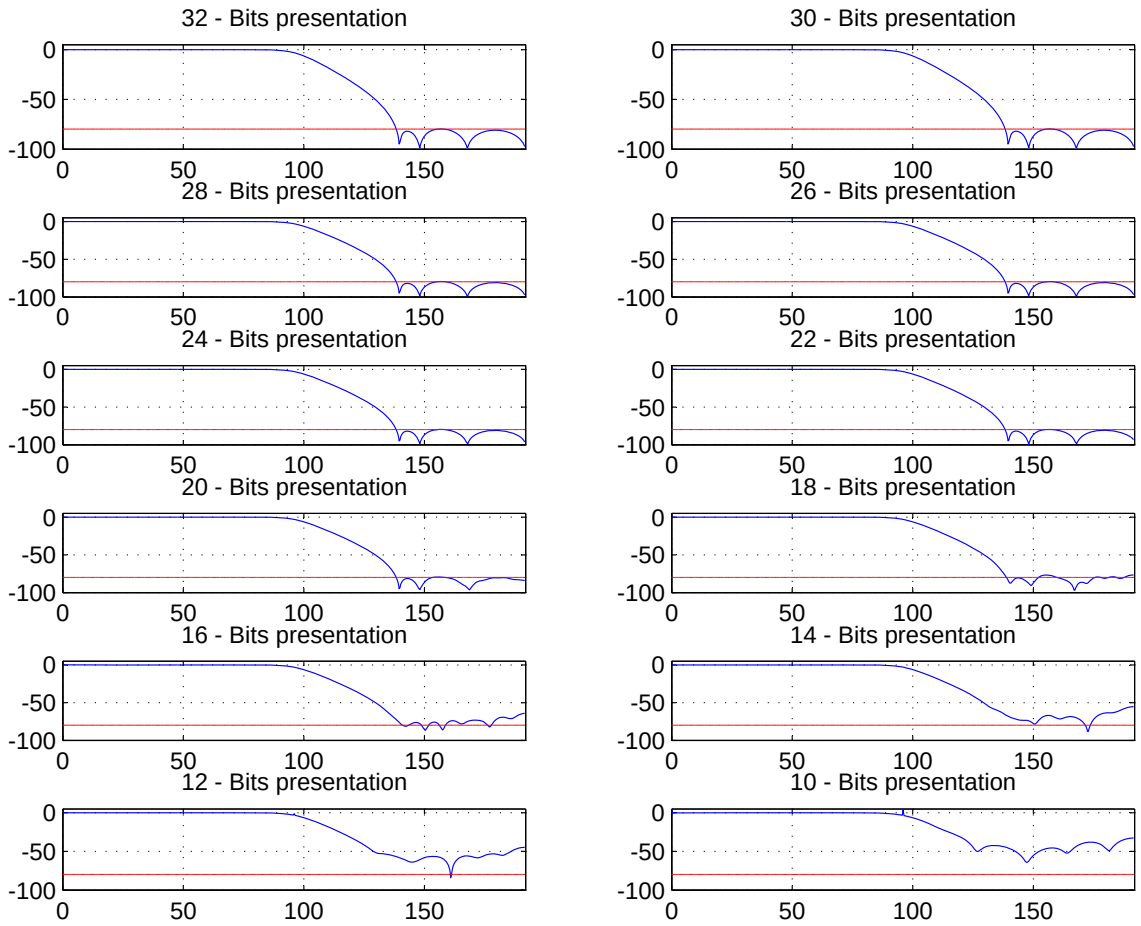


Figure 24: Filter characteristic for half pass filter block 1 and 2 for varying bit representation. The y-axis is in dB and the x-axis in kHz.

The recursive all-pass filter is the first selectivity filter used in the filter chain, and from figure 25, we see that due to the length of the filter it is more sensitive to quantization noise, than the short half-band filters. To maintain the 80 dB limit, at least 26 dB is needed.

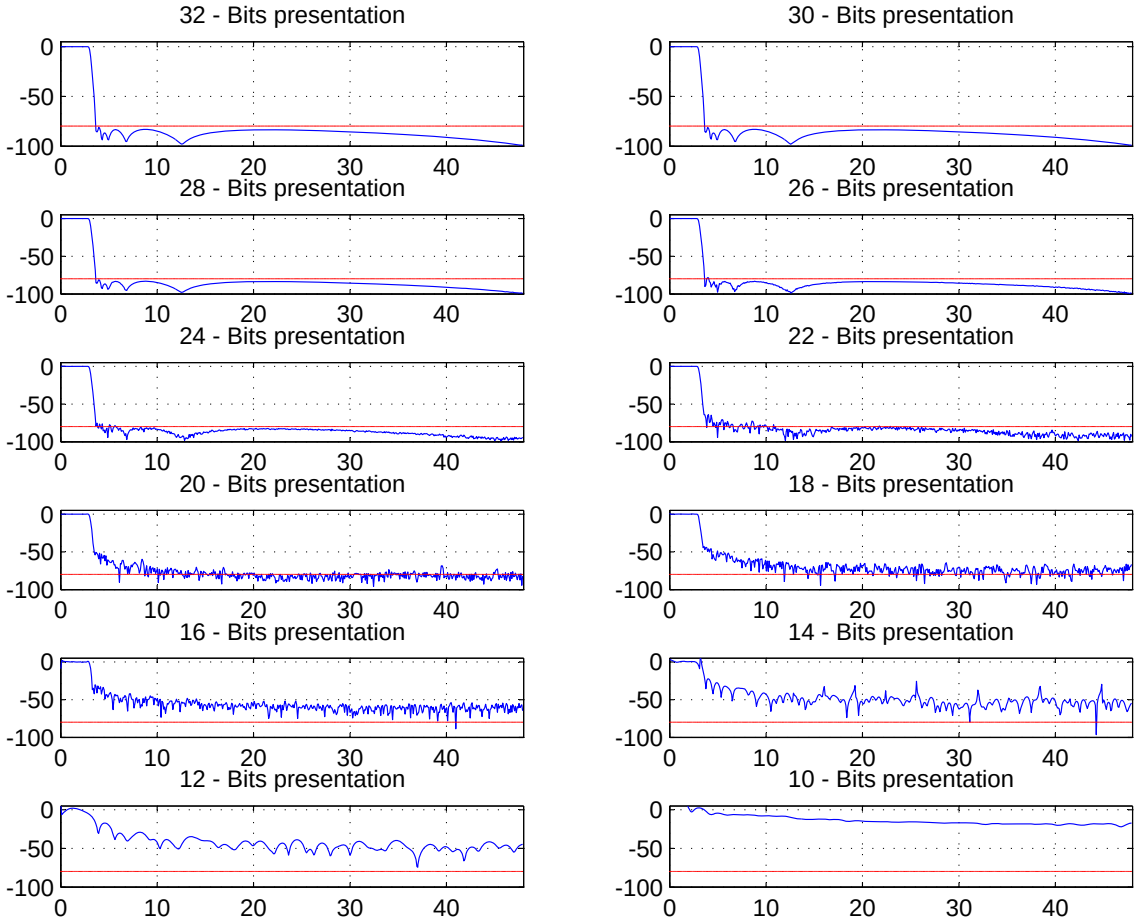


Figure 25: Filter characteristic for the recursive all-pass filter for varying bit representation. The y-axis is in dB and the x-axis in kHz.

The five coefficient half band filter need at least 20 bit to maintain the stop-band attenuation as shown in figure 26. This is equal to the representation of three coefficient half-band filters, and since the total sum of squared error only differed by a small factor, this was expected.

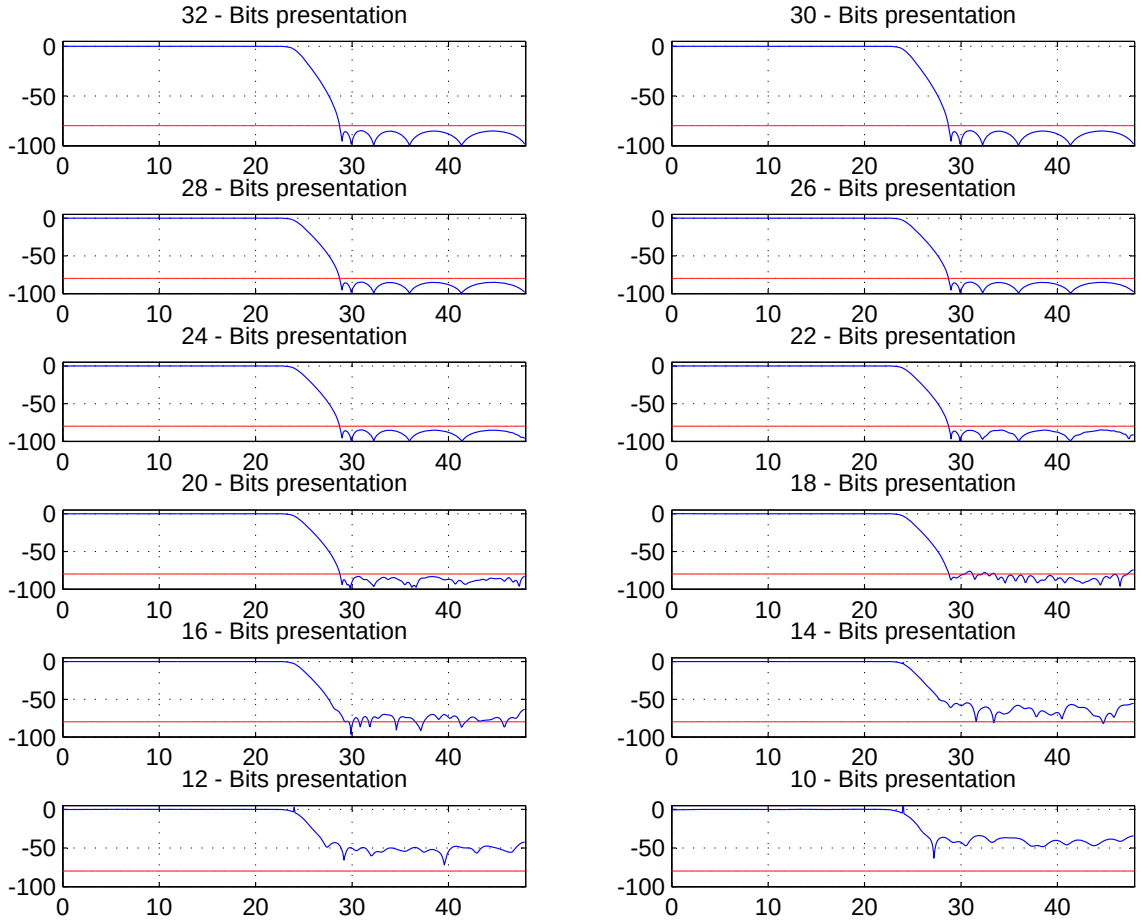


Figure 26: Filter characteristic for half pass filter block 3 for varying bit representation. The y-axis is in dB and the x-axis in kHz.

The last filter in the chain is the complex FIR, here the last selectivity is obtained. The slope on the filter has to maintain the steepness, in order to maintain the wanted selectivity. In figure 27, the simulation of the complex FIR filter shown, a word length would require 20 bit to maintain the slope of the filter, as well as the out of band attenuation.

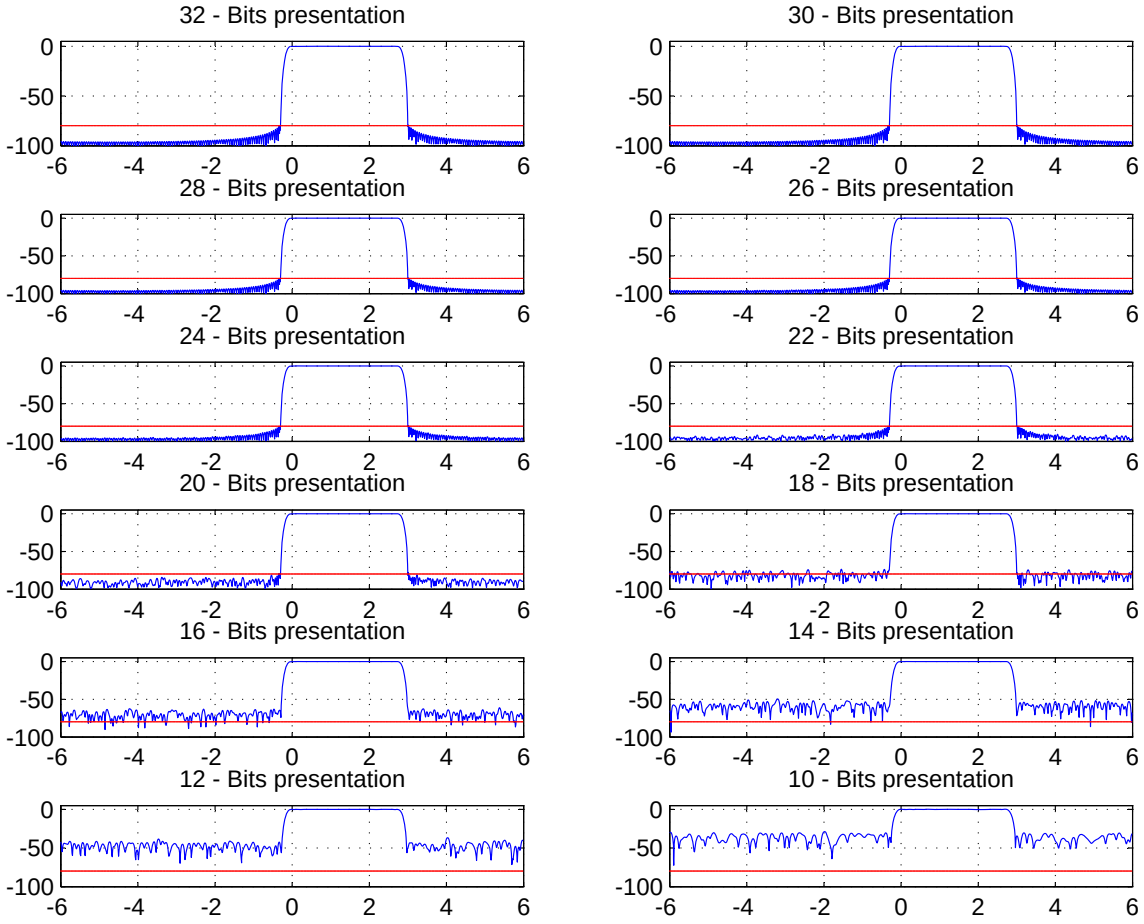


Figure 27: Filter characteristic for complex FIR filter block for varying bit representation. The y-axis is in dB and the x-axis in kHz.

0.12 Sub-conclusion

The simulation of the word length shows the performance of the filters at different bit representations. The recursive all-pass filter is most sensitive to quantization, which was expected since it was the longest of the recursive filters. The complex FIR filter is the most critical filter in order to obtain a high selectivity, as well as sensitivity due to the complex structure of the filter. It has been show that the number of bits can be reduced in each block, and this will reduce the cost in a hardware solution. However it should be notice that the simulation do not take care of scaling within each block, and in a final implementation of this should be done to avoid overflow. A final solution with reduced word length could be a Hilbert Transform presented with 10 bit, all the half-band section presented with 20 bit, the all-pass section with 24 bit and the complex FIR filter with 20 bit. With this combination the algorithm should still be able to fulfil the criteria specified, but this should be tested and verified before a final implementation.

Implementation

The implementation in this work was done using the DSP Builder, which allows implementing of the system modelled in simulink. The SignalCompiler block generates the VHDL files for synthesis, hardware implementation, and simulation. To verify the fixed-point simulink simulation the output is compared with the Matlab result.

0.13 Block implementation

The implemented blocks design by as listed below.

- Hilbert transform: Two-path half band filter
- Down-convert: NCO MegaCore Function
- Recursive HBF1: Two-path half band filter
- Recursive HBF2: Two-path half band filter
- Recursive All-Pass: Two-path filter
- Recursive HBF3: Two-path half band filter
- Recursive HBF4: Two-path half band filter
- Recursive HBF5: Two-path half band filter
- Complex Filter: FIR filter

Each block was designed as a sub-system in Simulink and Mask in order to generate the VHDL code with the SignalCompiler. Each block was test independenly, as shown in figure 28.

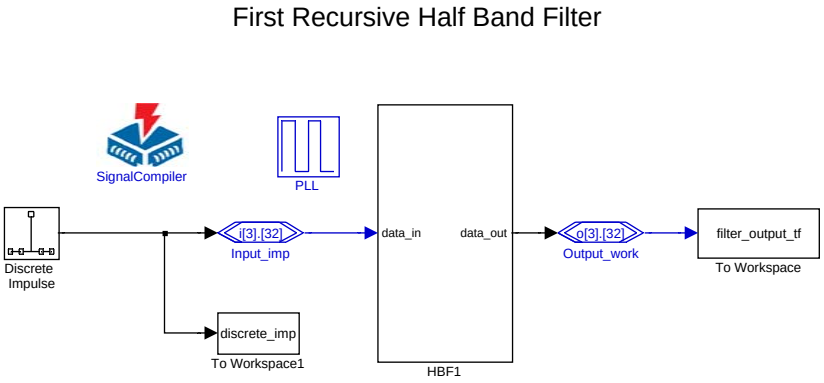


Figure 28: Black box test of the first half band filter.

Figure 29, show the sub-system for the first half band filter, and within the sub-system three All-pass sub-system is implemented. The output from each intermediate variable was save in MatlabŠs workspace in order to verify the result.

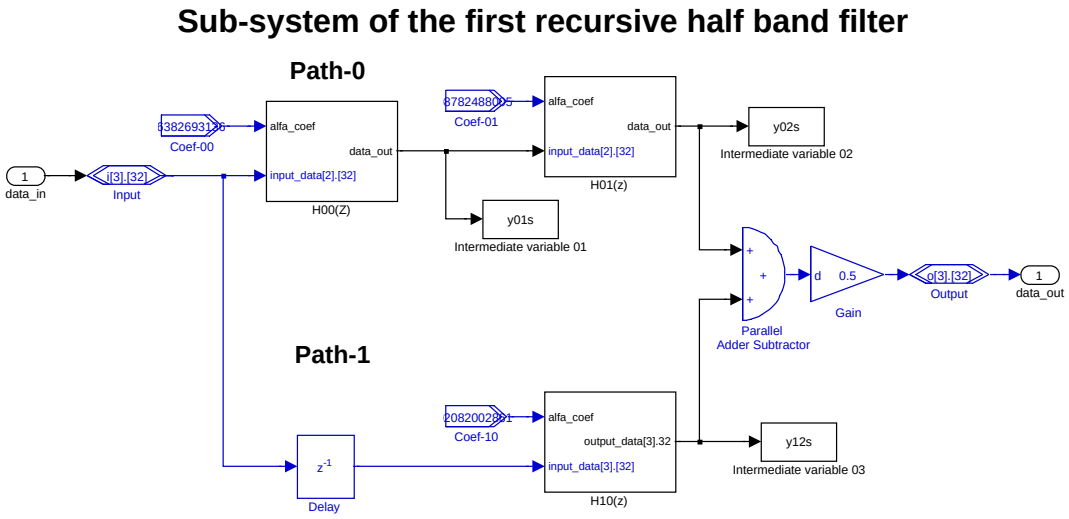


Figure 29: The sub-system for the first half band filter, notice that the down-samplings part is omitted in this test implementation.

The entire block system is implemented in simulink/DSPBuilder as shown in figure 10 at different word length, the result has then been compared with the SystemC simulation as shown in section 0.11.

0.14 Sub-conclusion

The DSPBuilder has shown to be a quick development tools that in co-operation with simulink is capable of developing fast prototypes for a communication. The interaction with Matlab makes it easy to verify the result obtained from DSPBuilder and simulink with the simulated results in Matlab.

The next step in this work is to test the system in a real-time communication system in co-operation with the company xxxx, and give them an introduction to multirate filtering system together with the implementing tools used within this project. The outcome from this should be, to show the possibility to develop software defined radios in less time than earlier with the efficient hardware component and software tools available today.

Bibliography

[1] M.W. Chamberlain.
A software defined hf radio.
In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 2448–2453Vol.4, 17-20 Oct. 2005.

[2] fred harris.
An Efficient Constant-Q Spectral Analyzer Architecture Using All-pass Recursive Filters.
PhD thesis, Electrical and Computer Engineering Department San Diego State University, 2000.

[3] fredric j harris.
Multirate Signal Processing For Communication Systems.
Prentice Hall, 1st edition, 2004.

[4] Jung Ko, V.C. Gaudet, and R. Hang.
Tier 3 software defined am radio.
In *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*, pages 257–261, 20-24 July 2005.

[5] C++ open source language.
Systemc community.
www.systemc.org, 2007.

[6] Walter HW Tuttlebee.
Advances in software-defined radio.
Electronics Systems and Software, pages 26–31, 2003.

[7] R.A. Valenzuela and A.G. Constantinides.
Digital signal processing schemes for efficient interpolation and decimation.
IEE Proc. Part G, 130(6):225–235, Dec. 1983.

Final Project Report

Table of Contents

Project Name : SOPC-based Voiceprint Identification System		
Team Name : Panda		
Team ID : IN00000004		
Tem Members :	1.	Huan Fang
	2.	Xin Liu
Email Address :	1.	huanf@kth.se
	2.	xinliu@kth.se
Contact No :	1.	0762321822
	2.	0707574312
Instructor :		Prof. Ingo Sander

Design Introduction

As globalization, networking, information and digital era's coming, the demand of high reliability of our identity verification is growing.An efficient mean to this is by authenticating users through biometric methods. Among the existing biometric methods, voice biometrics can be an affordable and accurate authentication technology that has been already successfully and widely employed. Voiceprint, as a basic human physiological characteristics, possess a unique role which is difficult to counterfeit, imitate and replace.As a non-contact identification technology, Voice Recognition Technology is being accepted by the users.

Voice authentication refers to the process of accepting or rejecting the identity claim of a speaker on the basis of individual information present in the speech waveform . It has received increasing attention over the past two decades, as a convenient, user-friendly way of replacing (or supplementing) standard password-type matching.

The authentication procedure requests from the user to pronounce a random sequence of digits. After capturing speech and extracting voice features, individual voice characteritics are generated by registration algorithm. The central process unit decides whether the received features match the stored voiceprint of the customer who claims to be, and accordingly grants authentication.

In this work, the architecture of an sopc-based voiceprint identification system is presented.

1. Voice Recognition Technology Principle

Voice Recognition, also known as the Speaker Recognition, has two categories: speaker identification and speaker verification. Speaker identification is used to determine which one of the people speaks, i.e. "one out of more election" ; and speaker verification is used to determine whether a person specified speaks, i.e. "one-on-one recognition".

According to the voice of different materials, voice recognition can be divided into the text-dependent, and text-independent technology. The text-dependent voice recognition system requires speaker pronounce in accordance with the contents of the text. Each person's individual sound profile model is established accurately. People must also be identified by the contents of the text during recognition to achieve better effect. Text-independent recognition system does not require fixed contents of words, which is relatively difficult to model, but is convenient for user and can be applied to a wide range.

Voiceprint recognition is an application based on physiological and behavioral characteristics of the speaker's voice and linguistic patterns. Different from speech recognition, voiceprint recognition is regardless of contents of speech.Rather, the unique features of voice are analyzed to identify the speaker. With voice samples, the unique features will be extracted and converted to digital symbols, and then these symbols are stored as that person's character template. This template is stored in a computer database, a smart card or bar-coded cards. User authentication is processed inside the recognition system to identify matching or not. The system architecture

block diagram is shown in Figure 1.

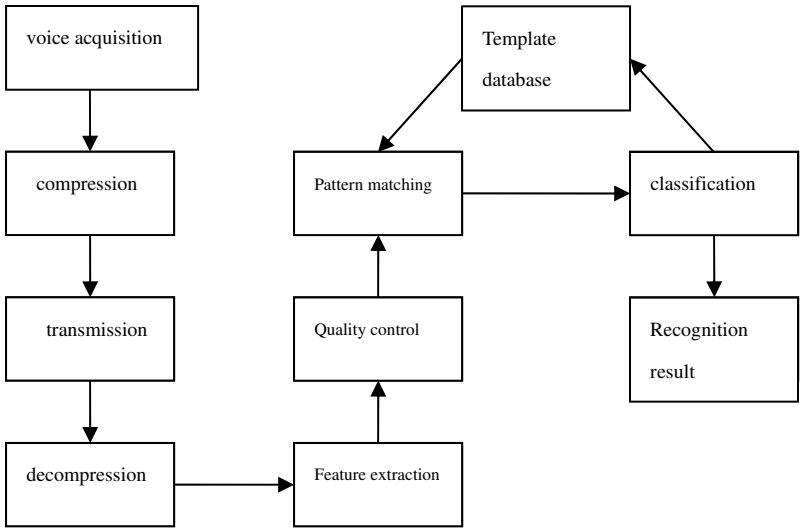


Figure 1 voiceprint recognition system architecture block diagram

2. Hardware Implementation

The Altera DE1 development board features a state-of-the-art Cyclone® II 2C20 FPGA in a 484-pin package. All important components on the board are connected to pins of this chip, allowing the user to control all aspects of the board's operation.
This design is implemented by a 32bit NiosII softcore processor.All IPs are connected on the avalon bus in SOPC builder, including custom peripherals

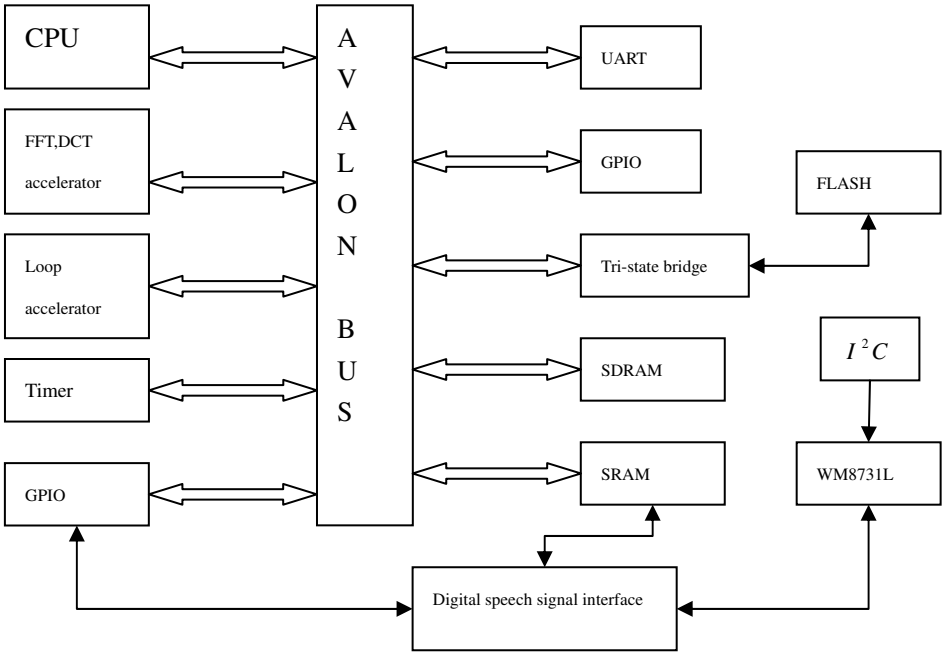


Figure 2 hardware architecture

The system hardware architecture is shown in figure 2,including CPU,uart,tri-state bridge,ram and I/O controls,which are all reusable.Such a design method not only

make it modulization,but also greatly reduce the design cycle of the system.FFT module can not only access IP directly, but also use C2H accelerator tool to improve system performance.In this design ,performance-critical sections such as FFT,DCT and iterative computations will be implemented via C2H hardware accelerator.

Nios II softcore processor

Nios II is a high performance 32-bit sofcore processor. The processor is configured on an Altera Cyclone II FPGA. Custom instructions are added to improve system performance, furthermore, more on-chip rams can be added to improve data processing capacity.

Voice acquisition and verification report

The DE1 board provides high-quality 24-bit audio via the Wolfson WM8731 audio CODEC(enCOder/DECoder). This chip supports microphone-in, line-in, and line-out ports, with a sample rate adjustable from 8 kHz to 96 kHz. The WM8731 is controlled by a serial I2C bus interface, which is connected to pins on the Cyclone II FPGA. A schematic diagram of the audio circuitry is shown in Figure 3.

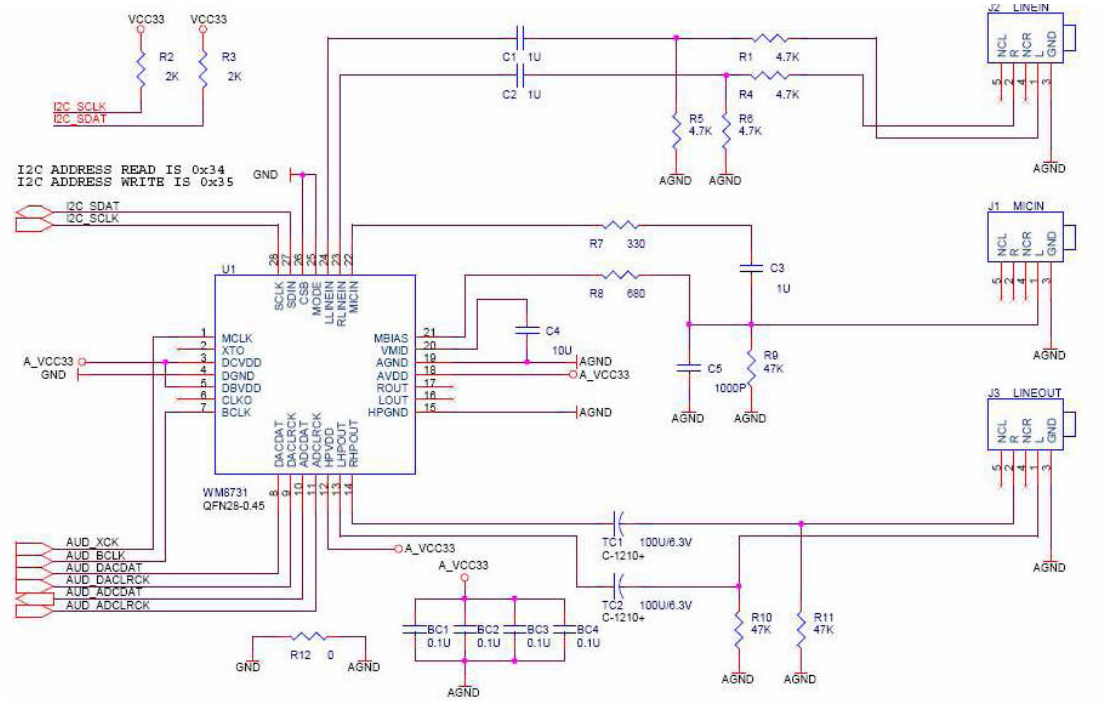


Figure 3 audio schematic diagram

WM8731 contains A/D,D/A modules with a high sample rate and quantization precision.We will use 8kHz sample rate and 16bit quantization precision in this design.

In voice acquisition part, since A / D is the serial data output, a serial to parallel data conversion and control of the SRAM Verilog module is needed.Voice report is communicated with CPU via GPIO,different voice is played according to different verification result.GPIO control is done in Nios IDE. Similarly, since the voice broadcast from FLASH are read out in parallel, thus a parallel to serial data conversion verilog module is needed.

C2H Hardware Acceleration

The Nios® II C-to-Hardware Acceleration (C2H) Compiler is a tool that allows you to create custom hardware accelerators directly from ANSI C source code. A hardware accelerator is a block of logic that implements a C function in hardware, which often improves the execution performance by an order of magnitude. Using the C2H Compiler, you can develop and debug an algorithm in C targeting an Altera® Nios II processor, and then quickly convert the C code to a hardware accelerator implemented in a field programmable gate array (FPGA).

The C2H Compiler improves the performance of Nios II programs by implementing specific C functions as hardware accelerators. The C2H Compiler is not a tool for creating arbitrary hardware systems using C as a design language. What the C2H Compiler does do is to generate an accelerator which is functionally identical to the original C function.

Based on these premises, the C2H Compiler's design methodology provides the following features:

■ ANSI C compliance – The C2H Compiler operates on plain ANSI C code, and supports most C constructs, including pointers, arrays, structures, global and local variables, loops, and subfunction calls. The C2H Compiler does not require special syntax or library functions to specify the structure of the hardware. Unsupported ANSI C constructs are documented.

■ Straightforward C-to-hardware mapping – The C2H Compiler maps each element of C syntax to a defined hardware structure, giving you control over the structure of your hardware accelerator.

■ Integration with C language development environments for the Nios II processor, including the Nios II integrated development environment (IDE), and the BSP generator and related tools. You control the C2H Compiler with the Nios II C development tools. You do not need to learn a new environment to use the C2H Compiler.

■ Based on SOPC Builder and Avalon system interconnect fabric – The C2H Compiler uses SOPC Builder as the infrastructure to connect hardware accelerators into Nios II systems. A C2H accelerator becomes a component within an existing Nios II system. SOPC Builder automatically generates system interconnect fabric to connect the accelerator to the system, saving you the time of manually integrating the hardware accelerator.

■ Reporting of generated results – The C2H Compiler produces a detailed report of hardware structure, resource usage, and throughput.

3. Software Algorithm

MFCC is currently the most popular feature coefficient used in the speech recognition, and it can obtain the more accurate results of speech recognition under a non-noise condition. In the MFCC algorithm, we first use the FFT to calculate the signal frequency spectrum , then we use DCT to further reduce the speech signal’s redundant information, and reach the aim of regulating the speech signal into feature coefficients

with small dimensions. The FFT and DCT algorithm can be used for any speech segment whose time-frequency resolution is fixed.

Feature extraction

MFCC feature coefficient extraction flow chart is shown as Fig. 1. The working process is:

1. Pre-emphasis of the speech signal, frame, adding window, then make the FFT to obtain the frequency information.
2. Pass the signal through the Mel frequency coordinate triangle filter groups to mimic the human hearing mechanism and the human hearing sensibility to different speech spectrum.
3. Calculate the logarithm value of the signal after the Mel filters to obtain the logarithmic spectrum.
4. Make the discrete cosine transform to the signal and obtain the MFCC coefficients.

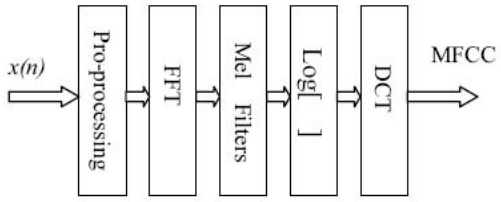
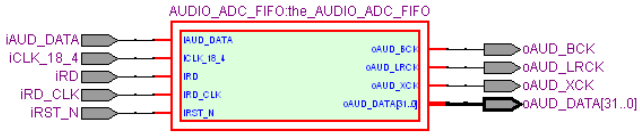


Fig.1 The feature extraction of the
Mel frequency cepstrum coefficients

4. Design Architecture

As seen in SOPC builder, we add an AUDIO_ADC_FIFO_0 module



to convert serial data to parallel audio samples(16 bit). On-chip rams (BufferRAM, CosRAM, SinRAM) are used to store data needed by FFT module which is C2H accelerated.

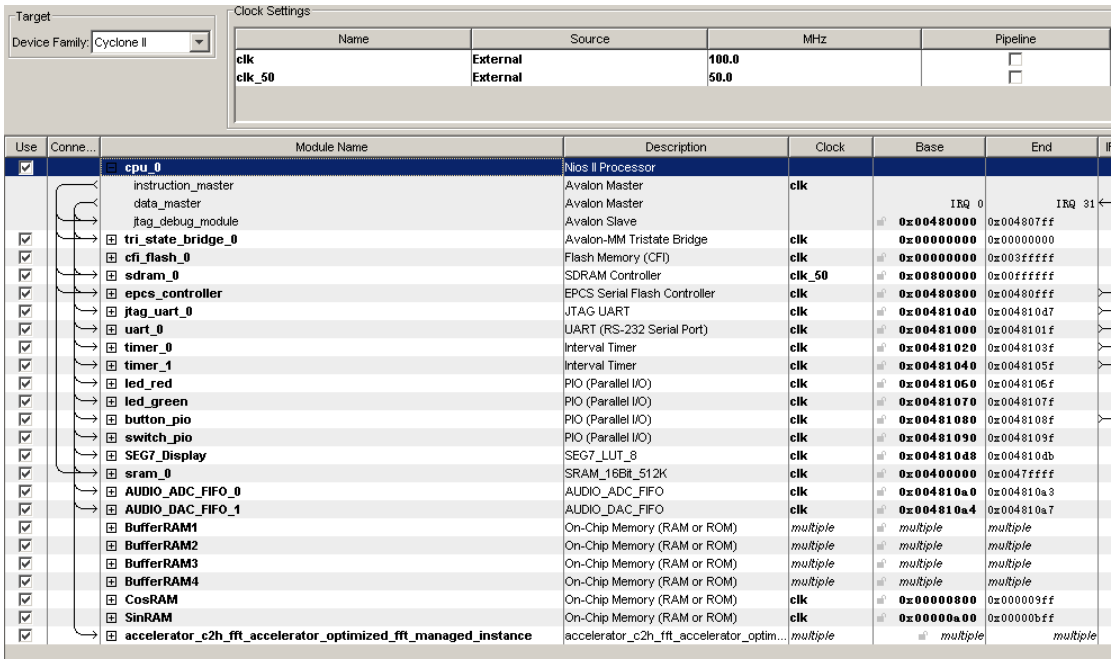


Figure 4 . SOPC builder system

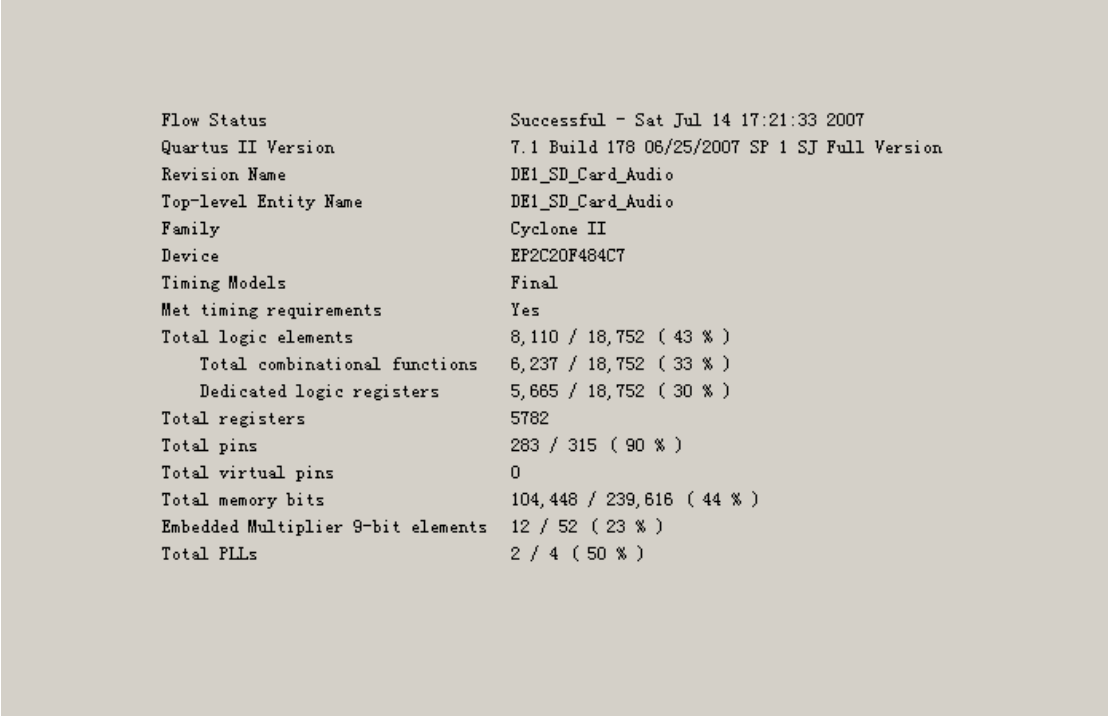


Figure 5 . Final Compilation report

Speech Acquisition

Speech acquisition requires a microphone coupled with an amplified ADC to receive the voice speech signal, sample it, and convert it into digital speech for input to the FPGA. The DE1 development board has a WM8731 audio codec chip connected both to the microphone input pins and the Altera Cyclone II FPGA pins through an I2C serial controller interface. The WM8731 is a versatile audio codec that provides up to 24-bit encoding/decoding of audio signals in various sampling rates ranging from 8 to

96 KHz. The codec clock input is generated by dividing the system clock by four using a custom hardware block. The block combines the clock divider logic and the I2S digital audio interface logic as well as options for programming the control registers. The DE1 board's microphone input port connects the microphone and headset for speech acquisition. The following table shows the codec's control register settings:

Codec Register Settings

Register	Setting
ADCDAT	16 bit
USB/normal mode	Normal mode
Master/slave mode	Slave
Digital audio interface	I2S interface
MCLK input	50 MHz divided by 4
ADC sampling rate	8 KHz

These settings are programmed by setting or resetting of various bits in the control registers as shown in the following table:

Register	Address	Register Name	Value
R0	0000000	Left line in	001A
R1	0000001	Right line in	021A
R2	0000010	Left headphone Out	047B
R3	0000011	Right headphone Out	067B
R4	0000100	Analog audio path control	0815
R6	0000110	Power down control	0C00
R7	0000111	Digital audio interface format	0A06
R8	0001000	Sampling control	100E
R9	0001001	Active control	1201

Original sound wave and frequency spectrum after FFT in MATLAB:

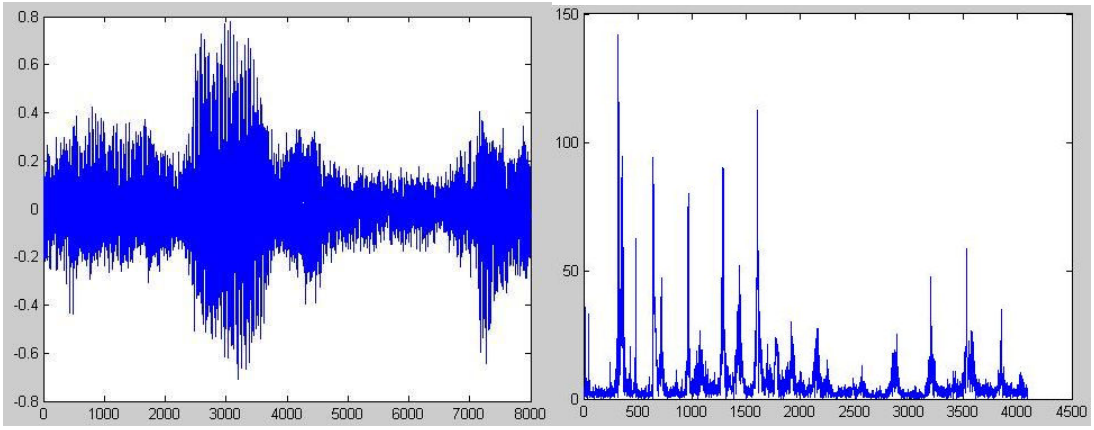


Figure 6: sound data in time domain and frequency domain

Mel Frequency Cepstral Coefficients (MFCCs)

MFCC are coefficients that represent audio. They are derived from a type of cepstral representation of the audio clip (a "spectrum-of-a-spectrum"). The difference between

the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are positioned logarithmically (on the mel scale) which approximates the human auditory system's response more closely than the linearly-spaced frequency bands obtained directly from the FFT or DCT. This can allow for better processing of data, for example, in audio compression. However, unlike the sonogram, MFCCs lack an outer ear model and, hence, cannot represent perceived loudness accurately.

MFCCs are commonly derived as follows:

1. Take the Fourier transform of (a windowed excerpt of) a signal
2. Map the log amplitudes of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
3. Take the Discrete Cosine Transform of the list of mel log-amplitudes, as if it were a signal.
4. The MFCCs are the amplitudes of the resulting spectrum.

Dynamic Time Warping (DTW)

A distance measurement between time series is needed to determine similarity between time series and for time series classification. Euclidean distance is an efficient distance measurement that can be used. The Euclidian distance between two time series is simply the sum of the squared distances from each nth point in one time series to the nth point in the other. The main disadvantage of using Euclidean distance for time series data is that its results are very unintuitive. If two time series are identical, but one is shifted slightly along the time axis, then Euclidean distance may consider them to be very different from each other. Dynamic time warping (DTW) was introduced to overcome this limitation and give intuitive distance measurements between time series by ignoring both global and local shifts in the time dimension.

Problem Formulation. The dynamic time warping problem is stated as follows: Given two time series X, and Y, of lengths |X| and |Y|,

$$X = x_1, x_2, \ldots, x_i, \ldots, x_{|X|}$$
$$Y = y_1, y_2, \ldots, y_j, \ldots, y_{|Y|}$$

construct a warp path W

$$W = w_1, w_2, \ldots, w_K \qquad \max(|X|, |Y|) \le K < |X| + |Y|$$

where K is the length of the warp path and the kth element of the warp path is

$$w_k = (i, j)$$

where i is an index from time series X, and j is an index from time series Y. The warp path must start at the beginning of each time series at w1 = (1, 1) and finish at the end of both time series at wK= (|X|, |Y|). This ensures that every index of both time series is used in the warp path. There is also a constraint on the warp path that forces i and j to be monotonically increasing in the warp path, which is why the lines representing the warp path in Figure 1 do not overlap. Every index of each time series must be used. Stated more formally:

$$w_k = (i, j), w_{k+1} = (i', j') \qquad i \le i' \le i+1, \;\; j \le j' \le j+1$$

The optimal warp path is the warp path is the minimum-distance warp path, where the distance of a warp path W is

$$Dist(W) = \sum_{k=1}^{k=K} Dist(w_{ki}, w_{kj})$$

Dist(W) is the distance (typically Euclidean distance) of warp path W, and Dist(wki, wkj) is the distance between the two data point indexes (one from X and one from Y) in the kth element of the warp path.

5. Performance Parameters

Recognition Accuracy

We record the word “hello” of user1 for our test voice and store it in the database.

Test case1: user1 speaks “hello” recognition accuracy: 80%

Test case2: user1 speaks other words rejection accuracy: 92%

Recognition Speed

FFT calculation: 100 ms

MFCC feature extraction: 8~12s

DTW algorithm: 1~2s

Total recognition time: 9~14s

6. Conclusion

Our project implements a voiceprint identification system in which a Nios II processor performs the recognition process. We implemented computationally intensive tasks with C2H accelerator and the SOPC Builder helped us integrate these blocks into the Nios II system. However, some tasks can not be C2H accelerated because C2H does not support float and double data type. These processes consumes a large amount of time. Recognition time can be reduced if a faster processor is used.

FINAL REPORT 22.8.2007
Team: SO-SIG

Forewords:

When I sent my interim report to Lena Engdahl only few days before the mid-summer eve and my summer vacation, I mentioned that unfortunately the project is not going any further during my vacation, which lasted until 1.8.2007. After returning to work I have had about two weeks time until I received, somewhat surprisingly, a query concerning this final report. During this two weeks time, there has been small advances in the project, which are described in the end of this document (cf. Chapter: Situation Today). Otherwise, this document is the same, which was sent to you as an interim report.

Although the Innovate Nordic competition is over, the work described in this document continues in the form of Licenciate's Thesis. In addition, there is also one student starting to work towards his Master's Thesis (under my guidance) in the middle of September. His work will be based mostly on the DE1 board and the results described in this document.

Introduction:

The initial idea here, was to implement an FPGA design, which would estimate the human posture by measuring the acceleration of his/her arm. The acceleration would be measused with an accelerometer attached to his/her hand.

It was expected, that because the subsequent signal processing and analyzation differs from an application to another, the FPGA should be used for implementation as this allows the basic design to be flexible, which can be easily extended into several application areas.

The Current Research Object:

During the course of time, the whole concept of the project has changed to a biosignal measurement system. In practice this means, that in addition to the accelerometer, a temperature sensor and a self-made pulse oximeter have been attached to the FPGA.

The system comprises of three sensors, namely accelerometer, temperature sensor and pulse oximeter. The two former are located on the same circuit board, which is connected via the pulse oximeter's circuit board to the FPGA (DE1 board). As the pulse oximeter and the temperature sensor are analog sensors, there is a need for an external A/D-converters. The block diagram of the whole system is depicted in Figure below

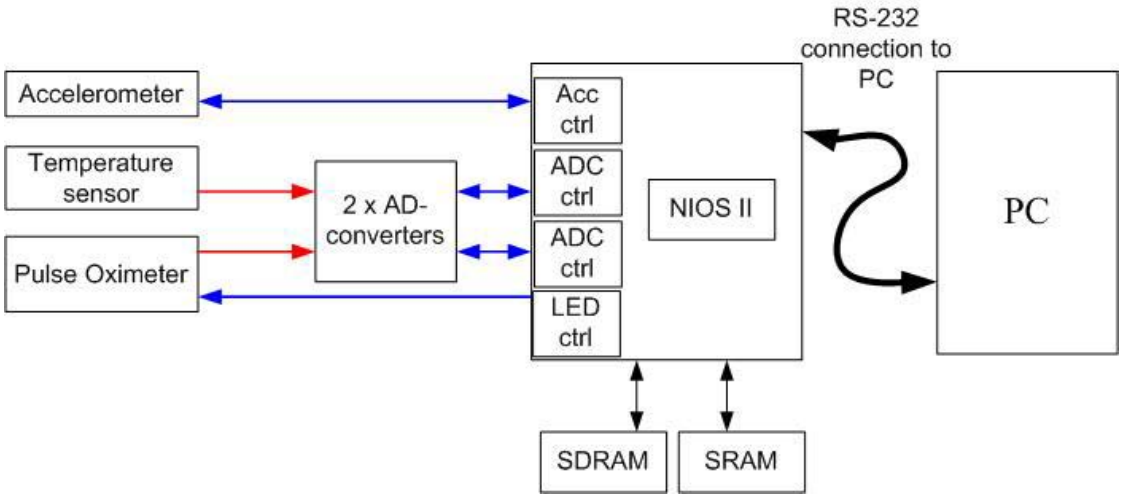


Figure 1. Block diagram of the measurement system

The Role of the FPGA:

Controllers

As the accelerometer is a digital device, communicating via 4-wire SPI bus, there is a need for a controller (Acc_crtl), which uses a special protocol, depicted in Figure 2 (see VTI3000-E04 datasheet for more details). The controller reads all the three axes of the accelerometer data from the specific addresses with a single command. In addition, as the data is received in serial form, the controller converts the data into parallel form. This data is sent to the NIOS II-processor.

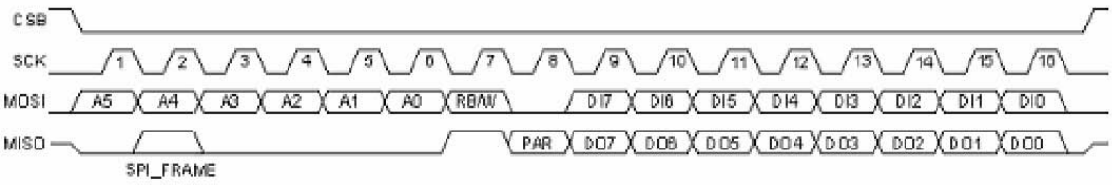


Figure 2. 4-wire SPI protocol in VTI3000-E04 accelerometer

The ADCs require somewhat similar controllers. The controllers differ somewhat from each other in detailed level, as the ADCs are different from each other. However, common to both of the controllers is to initiate the AD-conversion, read the obtained result and finally put the ADC to sleep mode, in order to conserve power. The obtained results are converted into parallel form if necessary and then sent to the NIOS II-processor.

The operation of the pulse oximeter is based on the different absorption spectra in red light wavelength and in infrared wavelength (IR). Therefore, a LED controller is needed here. The purpose of the controller is to alternatively light one of the two LEDs (red or IR) and to keep a small pause between the light periods. This scheme is depicted in Figure 3

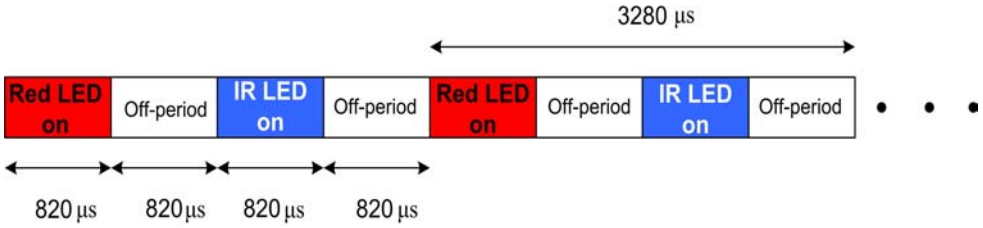


Figure 3. LED controlling scheme

NIOS II – processor

The data sent to the NIOS II –processor from the controllers cause an interrupt event. If an interrupt is detected, the measurement results from all the different channels connected to the NIOS II- processor are saved into the memory. The different channels are as follows: Acceleration (in x,-y- and z-directions), temperature, received light from pulse oximeter (red light, IR-light and the light when both LEDs are off).

When the installed SRAM (512kB) and SDRAM (8 MB) are full of measurement data or alternatively the conducted test is over, the NIOS-II processor opens the serial port and the data is sent to the PC via RS-232. Figures 4a and 4b show an example of measured data.

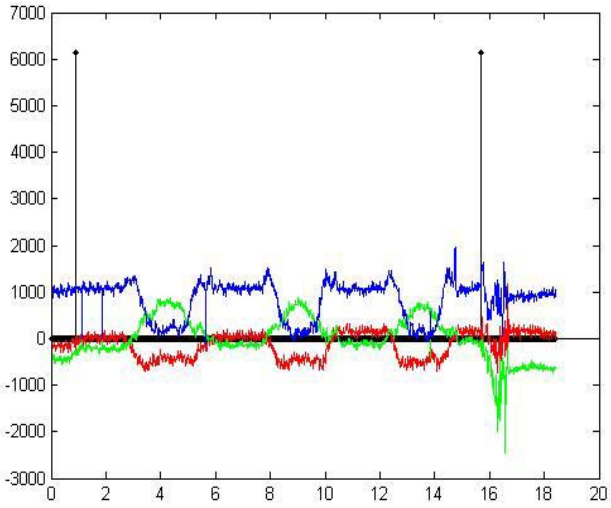


Figure 4a. Measurements from the accelerometer (x-axis = red, y-axis = green, z-axis = blue)

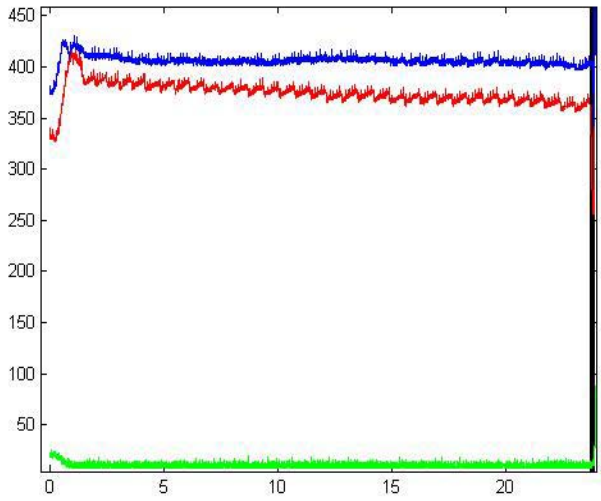


Figure 4b. Measurements from the pulse oximeter (red LED = red, IR-LED = blue, both LEDs off = green)

Conclusion:

Initially the project was supposed to have an accelerometer and an algorithm, which would compute the posture of a person. However, during the course of time, the project focus was shifted towards a **portable biosignal recorder,** which includes more sensors than just the mentioned accelerometer. Due to this reason, the required algorithms were never implemented but the main focus has been in developing an the hardware (electronics and VHDL code) for the biosignal recorder.

In the future, there is an intention to improve the system in many areas, including the digital signal on FPGA and connecting the DE1 board to PC using a wireless ZigBee technology. However, this won't be ready due to the deadline of the competition in August.

Situation Today:

In this two weeks time, there has been quite a lot misfortune in the development of measurement electronics. The output from the temperature sensor oscillated for some unknown reason, which caused the breaking of the operational amplifier, the A/D-converter's voltage reference and one of the A/D-converters. These parts were changed. In addition, there was a second order filter added into the analog temperature channel to filter out the 50 Hz power line hum.

In the NIOS-II development, it was discovered that the original idea, where the incoming data from each sensor caused an interrupt to the processor, was unsuitable for the data collection. This data collection scheme led to a situation, where the time between subsequent interrupts (and thus samples from the sensors) was forever changing instead of being constant. This of course causes problems in the further development of the algorithms as the sample times were not absolute but relative to each other. As an improvement, a controller keeps track of the time and after three consecutive samples from the pulse oximeter, which has the highest sampling rate, are received, the NIOS-II processor stores the data from **all** the sensors. In this way, we know the absolute time interval between different samples.

There are many advances planned in the future. These include the following:

1) One additional sensor, namely a capacitive touch sensor, will be added to the biosignal recorder. The information from this sensor can be used to deduce, whether the biosignal recorder is being worn by the user and whether the positioning of the device is correct. The development begins as soon as the electronic components are received

2) The wireless link will be developed between the biosignal recorder and a PC. This link will be based on a wireless personal area network (WPAN) technique e..g Bluetooth or ZigBee. The development begins in the middle of September.

3) The algorithm development begins with collecting data. The aim is to develop such algorithms, that can extract important medical data from the raw data. This data can be used for e.g. alarming the nursing staff or it can be used for measuring physiologic feedback for some stimuli e.g. a stress in physiological experiments. The developed algorithms should utilize the parallel processing capability offered by the Cyclone II FPGA.

Final Project Report: Cryptoprocessor for Elliptic Curve Digital Signature Algorithm (ECDSA)

Team ID: IN00000026
Team member: Kimmo Järvinen
tel. +358-9-4512429, email. kimmo.jarvinen@tkk.fi
Instructor: Prof. Jorma Skyttä
tel. +358-9-4512450, email. jorma.skytta@tkk.fi

Helsinki University of Technology, Signal Processing Laboratory
Otakaari 5A, FIN-02150, Espoo, Finland

August 7, 2007

Abstract

Elliptic Curve Digital Signature Algorithm (ECDSA) is implemented on an Altera Cyclone II EP2C20F484C7 FPGA using a DE1 development and education board. Digital signatures are digital counterparts of handwritten signatures. They provide proof of authorship and authenticity and they are unforgeable. They also provide proof that the document has not been altered after signing. The design includes a Nios II processor together with custom-designed modules for elliptic curve cryptography, SHA-1 hash function and modular arithmetic. A pseudo-random number generator is also included for rapid and secure generation of pseudo-random numbers. A user interface is designed with Nios II Integrated Development Environment (IDE) for demonstrating the use of the design. The design requires approximately 85 % of the device resources. Signature generation is computed in 0.94 ms and signature verification requires 1.61 ms.

1 Preliminaries

Research on hardware implementation of cryptographic algorithms has been intensive during the recent years. Field-programmable gate arrays (FPGAs) are very attractive platforms for implementing cryptographic algorithms for various reasons including performance, flexibility and cost efficiency [15]. This report presents an efficient implementation of Elliptic Curve Digital Signature Algorithm (ECDSA) by using a standardized curve B-163 which is listed in [11].

Digital signatures play a central role in modern cryptosystems. They can be viewed as digital counterparts for handwritten signatures and they are authentic,

unforgeable and non-reusable. A signed document is also unalterable and the signature cannot be repudiated meaning that the signer cannot afterwards claim that (s)he did not sign the document. [13]

Digital signature algorithms are public-key cryptographic algorithms and thus they involve two keys; one which is private and one which is public. A document is signed with the private key and the signature is verified with the public key. Only the private key needs to be kept in secret in order to prevent other people from forging one's signature.

Public-key cryptographic algorithms are based on mathematics (or number theory to be more precise) and it is impossible to discuss these algorithms without any math. The focus of this report is in implementing ECDSA on an FPGA and details of the algorithms are consider only to the point which is necessary for understanding the implementation. If more detailed descriptions of algorithms are wanted, then the reader should consult references which are listed in the end of the document.

The implementation is optimized especially for Altera FPGAs and it is designed to take advantage of embedded memory blocks inside Cyclone II. Most of the design efforts have been dedicated to elliptic curve operations which are the most time consuming operations in ECDSA, by far. The results show that even a low-cost FPGA such as Cyclone II can be efficiently used for implementing complex high-security public-key cryptographic algorithms.

Sec. 1.1 presents the basics of elliptic curve cryptography and Sec. 1.2 describes ECDSA. Hardware architecture is presented in Sec. 2 and a user interface implemented by using Nios II IDE is considered in Sec. 3. Results are presented in Sec. 4 together with discussion on them. Finally, the report ends with a list of possible improvements.

1.1 Elliptic Curve Cryptography

The theory of elliptic curves is deep and an enormous amount of research has been done on elliptic curve cryptography during the past twenty years or so. Therefore, it is impossible to present an extensive review of the field here and only subjects which are the most relevant are discussed in the following. Interested readers are referred to [3], for example, for further information.

All elliptic curve cryptosystems are based on an operation called elliptic curve point multiplication which is defined as

$$Q = kP \tag{1}$$

where k is an integer and Q and P are points on an elliptic curve. A point is represented with two coordinates as (x, y).

The reason why elliptic curve point multiplication is used in cryptosystem is that it is relatively easy to compute but its inverse operation called elliptic curve discrete logarithm problem, that is finding k if P and Q are known, is considered

impossible to solve with present computational resources if parameters are chosen correctly. Thus, elliptic curve discrete logarithm problem can be compared, for example, to integer factorization problem which is used in the popular RSA cryptosystems. There is, however, a notable difference because sub-exponential algorithms for solving elliptic curve discrete logarithm problem are not known and, therefore, key lengths can be shorter than in RSA.

Elliptic curve point multiplication is computed by using two principal operations; namely, point addition and point doubling. Point addition is the operation $P_3 = P_1 + P_2$ where P_i are points on an elliptic curve. Point doubling is the operation $P_3 = 2P_1$. In this design, point multiplication is computed with the so-called Montgomery's ladder [10] which operates as shown in Alg. 1 of the Appendix.

Elliptic curves used in cryptosystems are defined over finite fields denoted by $GF(q)$ where q is the number of elements in the field. It is commonly preferred especially in hardware implementations to use binary fields $GF(2^m)$ where an element of the field is presented with m bits. In this design, the field $GF(2^{163})$ is used and it is constructed by using normal basis. Arithmetic operations are computed as follows:

- Addition $a + b$ is computed with a bitwise exclusive-or (XOR).
- Multiplication $a \times b$ is computed as presented by Wang et al. in [14]. This multiplier structure is referred to as Massey-Omura multiplier and it is discussed in Sec. 2.1.1.
- Squaring a^2 is simply a cyclical rotation of the bit vector representing a.
- Finding an inverse element a^{-1} such that $a^{-1} \times a = 1$ is performed as suggested by Itoh and Tsujii in [4] and it is called henceforth Itoh-Tsujii inversion. One Itoh-Tsujii inversion requires 9 multiplications and 162 squarings if $m = 163$ [4].

Point representation with two coordinates as (x, y) is referred to as the affine coordinate representation. When points are represented in affine coordinates, both point addition and point doubling require inversion in $GF(2^m)$. Inversion is by far the most expensive operation and, thus, it is advantageous to trade inversions for multiplications. This can be done by representing points with projective coordinates as (X, Y, Z); that is, with three coordinates. Mappings between these two representations are performed as $(x, y, 1)$ and $(X/Z, Y/Z)$. As can be seen, the mapping from affine to projective coordinates does not require any operations but the mapping from projective to affine coordinates requires two multiplications and one inversion. Using projective coordinates is very advantageous because point additions and point doublings can be performed without inversions and the total number of inversions in elliptic curve point multiplication is therefore one.

A very efficient algorithm for computing (1) on elliptic curves over $GF(2^m)$ was presented by Julio López and Ricardo Dahab in [9]. They showed that, when Alg. 1 is used, it suffices to consider only the x-coordinate and the y-coordinate can

be recovered in the end [9]. This leads to a very efficient algorithm with projective coordinates. Point addition $(X_3, Z_3) = (X_1, Z_1) + (X_2, Z_2)$ can be computed as follows: [9]

$$Z_3 = (X_1 Z_2 + X_2 Z_1)^2, \quad X_3 = x Z_3 + X_1 Z_2 X_2 Z_1 \tag{2}$$

where x is the x-coordinate of the base point P in Alg. 1. The cost of point addition is four multiplications, two additions and one squaring. Point doubling $(X_3, Z_3) = 2(X_1, Z_1)$ is even simpler [9]

$$X_3 = X_1^4 + a_6 Z_1^4, \quad Z_3 = X_1^2 Z_1^2 \tag{3}$$

where a_6 is a fixed curve parameter. Thus, point doubling costs two multiplications, four squarings and one addition. The y-coordinate is recovered in the end by computing $x_1 = X_1/Z_1$ and $x_2 = X_2/Z_2$ and then by using the formula [9]:

$$y_1 = \frac{(x_1 + x)\left((x_1 + x)(x_2 + x) + x^2 + y\right)}{x} + y \tag{4}$$

where (x, y) is the base point P. This can be computed with one inversion, ten multiplications, six additions and one squaring.

1.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is a standard of ANSI, IEEE, and NIST, among others. The following description is based on Johnson and others' presentation in [5].

The algorithm operates so that first the user, who is commonly called Alice or A for short, generates two keys, private and public, by performing a key pair generation procedure. Then, she publishes her public key. Alice signs a message by performing a signature generation procedure after which she sends both the message and the attached signature to the receiver who is called Bob, or B for short. Bob can verify the signature on the message by first getting Alice's public key and then by performing the signature verification procedure.

Key pair generation, signature generation and signature verification are consider in the following sections.

1.2.1 Key Pair Generation

Private and public key for an identity A is generated as follows:

$$\begin{aligned} d &\in_R [1, n-1] \\ Q &= dG \end{aligned} \tag{5}$$

where $d \in_R [1, n-1]$ means that d is an integer selected at random from the interval $[1, n-1]$. The integer d is A's private key and Q is A's public key. The computation of (5) requires generation of one random integer and computation of one elliptic curve point multiplication.

1.2.2 Signature Generation

In order to generate a signature for a message $\mathcal{M}$ the identity A computes

$$\begin{aligned} k &\in_R [1, n-1] \\ r &= [kG]_x \pmod{n} \\ e &= \text{SHA-1}(\mathcal{M}) \\ s &= k^{-1}(e + dr) \pmod{n} \ . \end{aligned} \tag{6}$$

A's signature on $\mathcal{M}$ is (r, s). The notation $[kG]_x$ denotes the x-coordinate of the result point of kG. Notice that A uses his/her private key d in the signature generation. Thus, other identities cannot produce the same signature without knowing d. Signing a message requires generation of one random integer, computation of one elliptic curve point multiplication and one hashing. In addition, modular inversion, addition and multiplication are required.

1.2.3 Signature Verification

Identity B verifies A's signature (r, s) on the message $\mathcal{M}$ by computing

$$\begin{aligned} e &= \text{SHA-1}(\mathcal{M}) \\ w &= s^{-1} \pmod{n} \\ u_1 &= ew \pmod{n} \\ u_2 &= rw \pmod{n} \\ v &= [u_1 G + u_2 Q]_x \pmod{n} \end{aligned} \tag{7}$$

where Q is A's public key and thus known by B. If $v = r$, B accepts the signature, otherwise (s)he rejects it. Verification requires one hashing and two elliptic curve point multiplications which are combined with a single elliptic curve point addition. Modular inversion and two multiplications are needed, as well.

2 Hardware Architecture

Based on the description of ECDSA given in Sec. 1.2, it is clear that the implementation of ECDSA must be capable of performing the following operations:

- Elliptic curve point multiplication
- SHA-1 hash function
- Modular addition, multiplication and inversion
- (Pseudo-)random number generation

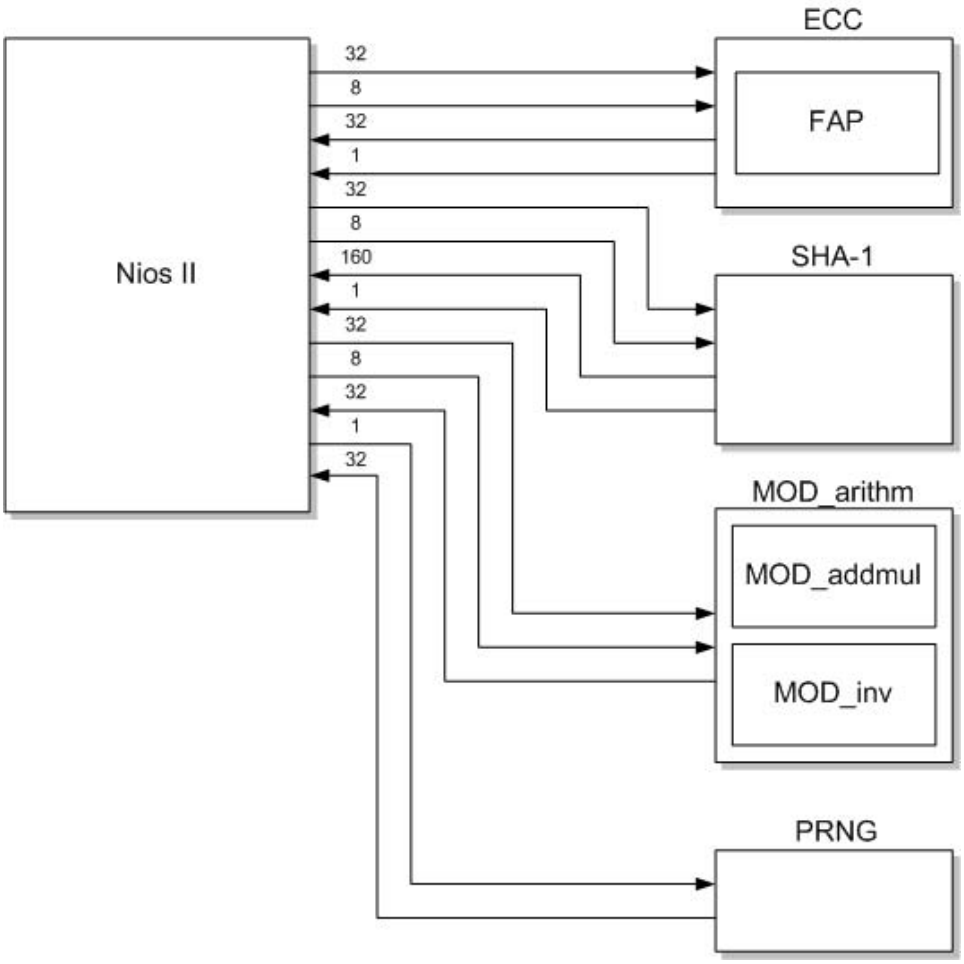


Figure 1: Block diagram of the system

The implementation includes custom-build blocks for each of the above operations in order to ensure fast performance. The most time and resource consuming operation is elliptic curve point multiplication and thus most of the effort was devoted in optimizing it.

Fig. 1 shows the block diagram of the ECDSA system. The Nios II processor is used for user interface and control. Four peripheral components are attached to Nios II and the actual ECDSA computations are performed with them. The peripheral components are elliptic curve module (ECC in Fig. 1), hash module (SHA-1), modular arithmetic module (MOD_arithm), and pseudo-random number generator (PRNG) and they are considered in the following sections. The user interface was realized on Nios II by using Nios II IDE 6.0, and it is considered in Sec. 3.

In order to enhance performance a phase-locked loop (PLL) is used for generating different clocks for different parts of the design. Nios II runs at 50 MHz,

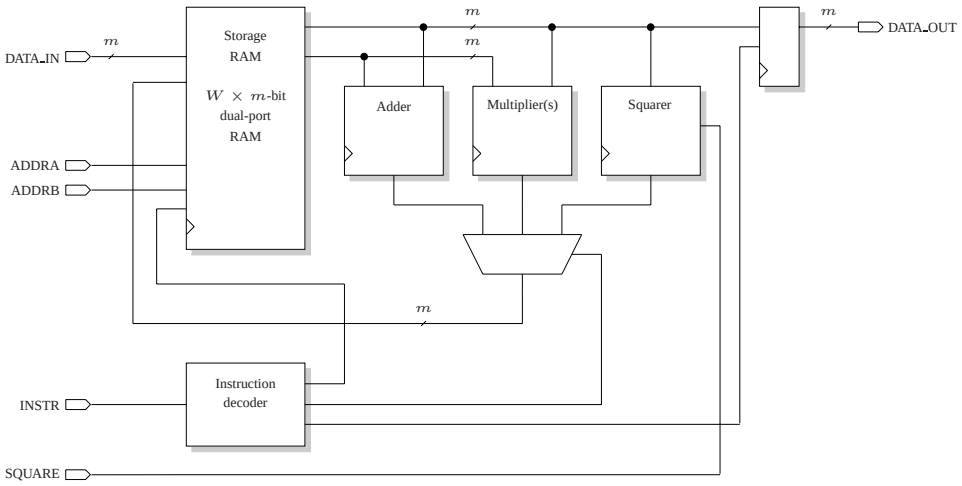


Figure 2: Block diagram of the FAP

ECC, PRNG and SHA-1 blocks at 75 MHz and modular arithmetic blocks at 20 MHz.

2.1 Elliptic Curve Module

The elliptic curve module consists of a field arithmetic processor (FAP) and logic controlling it. That is, the FAP performs operations in $GF(2^{163})$ and the control logic implements elliptic curve operations by using the FAP for field operations. The architecture is based on an elliptic curve processor which has been used in [2, 7, 8] which are scientific publications (co-)authored by the author. The author is alone responsible for the development of the architecture and all VHDL coding.

2.1.1 Field Arithmetic Processor

The FAP consists of adder, squarer, multiplier, storage RAM and instruction decoder. Block diagram of the FAP is presented in Fig. 2.

Adder and Squarer The adder computes a bitwise XOR of two m-bit operands, and it has a latency of one clock cycle. The squarer supports computation of multiple successive squarings, i.e. x^{2^d} where x is an element of $GF(2^{163})$ and d is an integer in the interval $[0, d_{\max}]$ with $d_{\max} = 2^5 - 1$. In normal basis squaring is a rotation of the bit vector as mentioned in Sec. 1.1, and the squarer is a shifter which computes x^{2^d} in one clock cycle.

Multiplier Field multiplication is critical for the overall performance. Multiplication in normal basis is performed with a multiplier which is a digit-serial implementation of the Massey-Omura multiplier [14]. In a bit-serial Massey-Omura

multiplier, one bit of the output is calculated in one clock cycle and, hence, m cycles are required in total. One bit z_i of the result $z = x \times y$, where x, y, z are elements of $GF(2^{163})$, is computed from x and y by using an F-function. The F-function is field specific, and the same F is used for all output bits z_i as follows: $z_i = F(x_{\lll i}, y_{\lll i})$, where $\lll i$ denotes cyclical left shift by i bits. Hence, a bit-serial implementation of the Massey-Omura multiplier requires three m-bit shift registers and one F-function block. A bit-parallel implementation, where all bits z_i are computed in parallel in one clock cycle, requires m F-function blocks and an m-bit register for storing the result. [11, 14]

In practice, the bit-serial implementation requiring at least $m + 1$ clock cycles is too slow and the bit-parallel implementation requires too much area. A good tradeoff is a digit-serial multiplier, where p bits are computed in parallel with p F-function blocks. The F-function blocks can be pipelined in order to increase the maximum clock frequency. As one clock cycle is required in loading the operands into the shift registers and each pipeline stage increases latency by one clock cycle, the latency becomes

$$\left\lceil \frac{m}{p} \right\rceil + c + 1 \tag{8}$$

where $\lceil \cdot \rceil$ denotes rounding up to the nearest larger integer and c is the number of pipeline stages inside the F-function blocks, i.e. $c \geq 0$. In this design, the parameters were selected to be $c = 1$ and $p = 12$.

Others The storage RAM is used for storing elements of $GF(2^{163})$ and it is implemented as a dual-port RAM by using embedded memory blocks in the FPGA, i.e. M4K blocks. The storage RAM is capable of storing W elements. When the architecture is implemented in a Cyclone II FPGA, a logical choice is $W = 256$ because, while in true dual-port mode, the widest mode that an M4K block can be configured to is 256×18-bits. Thus, the storage RAM requires $\lceil 163/18 \rceil = 10$ M4Ks resulting in a storage capacity of 256×163-bits. This much storage space is rarely needed, but it can be used for example for storing pre-computed points, and selecting a smaller depth than 256 would not reduce the number of required M4Ks. Both writing and reading to and from the storage RAM require one clock cycle. However, the dual-port RAM can be configured into the read-during-write mode [1] which saves certain clock cycles; see Sec. 2.1.2.

The instruction decoder decodes instructions to signals controlling the FAP blocks.

2.1.2 Control Logic

The logic controlling the FAP consists of finite state machine (FSM) and ROM containing instruction sequences.

The instruction sequences are carefully hand-optimized, and certain tricks are used in order to minimize latencies of point operations. As mentioned in Sec 2.1.1,

the read-during-write mode can be used for reducing latencies. In order to maximize the advantages in this case, operations are ordered so that the result of the previous operation is used as the operand of the next operation whenever possible. This saves one clock cycle because the operands of the next operation can be read simultaneously while the result of the previous operation is being written.

Inversions in $GF(2^{163})$ are computed with successive multiplications and squarings as suggested by Itoh and Tsujii in [4]. An Itoh-Tsujii inversion has the constant cost of 9 multiplications and 162 squarings when $m = 163$ [4]. Although the number of squarings is high, the successive squaring feature of the squarer (see Sec. 2.1.1) ensures that the computational cost of the squarings remains reasonable.

The elliptic curve module computes point addition and point doubling (one step in Alg. 1) in 125 clock cycles. Interfacing and mapping back to affine coordinates requires 404 clock cycles. In total 162 point additions and point doublings are needed because $\ell = 163$ in Alg. 1. Thus, one elliptic curve point multiplication requires 20,654 clock cycles. Signature verification requires computation of two elliptic curve point multiplications whose results are added with a point addition. This point addition is performed in affine coordinates and it requires 247 clock cycles and the elliptic curve operations computed in verification thus require 41,555 clock cycles.

2.2 Hash Module

The hash module implements SHA-1 hash algorithm according to the standard [12]. The architecture is described in detail in [6] but a short review is given here.

First of all, SHA-1 handles messages in blocks of 512 bits each of which requires computation of 80 steps. One step handles five 32-bit variables by computing four 32-bit modular additions ($a + b \bmod 2^{32}$) and certain 32-bit logical functions which depend on the step index. When all blocks have been processed, the hash of the message is in the five variables and thus SHA-1 outputs a 160-bit hash. [12]

The hash module implements SHA-1 in a straightforward manner and the design utilizes only logic resources. The VHDL describing the design is portable and device independent[1].

The implementation consists of four main components; namely, step function, message schedule, constants block and control logic. The step function block determines the maximum clock cycle of the implementation and it was carefully optimized. The four 32-bit additions form the critical path and all other operations (logic operations and rotations) are computed in parallel with these additions. The message schedule stores 512-bits message bits and derives a 32-bit word for each step from these 512 bits by using three 32-bit bitwise XORs and a rotation. The constants block includes step constants. The word from the message schedule and

[1] Actually, the code was originally written for Xilinx Virtex-II FPGA but it synthesized for Cyclone II without any modifications.

the constants are then used in the step function. The control logic is used for controlling the computation and it consists of a counter, multiplexors and coders. [6]

2.3 Modular Arithmetic Modules

Modular arithmetic modules implement the following operations:

- $c = a + b \pmod{n}$
- $c = a \times b \pmod{n}$
- Computation of a^{-1} such that $a^{-1} \times a = 1 \pmod{n}$

where a, b, c are integers in the interval $[0, n-1]$ and n is a fixed prime number which is hardwired into the design. The length of all integers is 163 bits.

Addition and multiplication are implemented in the same block. Modular addition is trivial. Addition is first carried out as a traditional 163-bit integer addition. If the result is larger or equal to n, then n is subtracted from the result. An addition is computed in two clock cycles.

Modular multiplication is more involved. In this design, it is carried out as shown in Alg. 2 of the Appendix. Computation of $2^i b$ is a shift to the left. The register holding a is shifted to the right so that it suffices to observe only the lsb of the register. One step in the for-loop in Alg. 2 requires three clock cycles and thus multiplication requires 489 clock cycles.

Inversion is the most complex operation of the three. It is performed with a binary algorithm as presented in [3], for example. The algorithm is presented in Alg. 3 of the Appendix. The binary algorithm was chosen because it does not require any integer divisions and is therefore efficient to implement. The latency of inversion is not constant but on average it is about the same as the latency of multiplication. Inversion, however, requires significantly more resources than multiplication.

2.4 Pseudo-Random Number Generator

A pseudo-random number generator (PRNG) is used in generating random integers in key pair generation and signing. The PRNG was implemented as a Linear Feedback Shift Register (LFSR). The length of the shift register was chosen to be 128 bits, and an irreducible polynomial [13]

$$p(x) = x^{128} + x^7 + x^2 + x + 1 \tag{9}$$

was used as a feedback function. Thus, the LFSR has a period of $2^{128} - 1$ bits [13]. The output bits from the LFSR are stored in a 32-bit shift register whose value is shown as the output of the PRNG every 32nd clock cycle. At the beginning, the PRNG is set to an initial state (all ones) with the reset signal.

```
Elliptic Curve Digital Signature Algorithm (ECDSA)
USER INTERFACE; (c) Kimmo Järvinen, 2007

MAIN MENU:
(g) Generate a new identity.
(s) Sign a message.
(v) Verify a signature.
(p) Print timings.
Enter your selection:
```

Figure 3: Main menu of the user interface

3 User Interface in Nios II IDE

A user interface was created in order to be able to demonstrate the operation of the ECDSA blocks. It should be noticed that the user interface was designed for demonstration purpose only, and it was not optimized for performance. In order to use the blocks in an application requiring fast performance, the user interface (and probably the Nios II processor altogether) should be replaced with custom-written logic.

The user interface was written in C language and it is used with the Nios II IDE. It supports four operations:

- Generation of new identities
- Signing of messages
- Verification of signatures
- Performance evaluation

The above list also forms the main menu of the user interface which is shown in Fig. 3. Three first ones are ECDSA operations and they implement equations given in Sec. 1.2. The last operation is used for measuring the performance of the implementation. A performance counter was attached to the Nios II processor and it can be used for measuring different parts of the code.

The user interface uses host file system supported by the IDE and it stores and handles identities, messages and signatures which are located on the harddisk of the host computer. This slows down the performance of the user interface but this approach was chosen because of the ease of implementation and use.

The C code is structured so that `ecdsa_interface.c` describes the user interface and ECDSA functions are given in `ecdsa.h/c`. The ECDSA functions directly control the peripheral components attached to Nios II. The functions include both top level functions for performing high-level tasks such as signature generation and verification and low-level handles for controlling each component

Table 1: Area consumption

Component	LEs	Regs.	M4Ks
Nios II	2,879	1,715	14
ECC	6,441	3,696	22
SHA-1	1,855	1,228	0
MOD_addmul	1,547	542	0
MOD_inv	2,871	1,026	0
PRNG	199	197	0
Total	15,879	8,472	36

individually. There are also handles for starting and stoping operations so that computations can be easily parallelized. For example, one can first begin computation of elliptic curve point multiplication which is the most time consuming operation, then begin hash function, compute some modular arithmetic operations after which different handles can be used for collecting the results of hashing and point multiplication. Considerable performance increases can be achieved with this approach compared to computing all operations sequentially.

The ECDSA functions are fast enough to be used also in real applications. However, custom-written control logic is probably needed for applications requiring very fast performance because Nios II will become the bottleneck as it already slows down the performance considerably as will be shown in Sec. 4.

4 Results and Discussion

The architecture described in Sec. 2 was written in VHDL and synthesized for Cyclone II EP2C20F484C7 with Quartus II 6.0 SP1. ModelSim SE 6.1b was used for simulating the code. Table 1 presents the area consumption of the design components. The design occupies 85 % of the logic elements (LEs) available on the device. Memory block (M4K) usage is 69 %.

Timing evaluations can be computed based on theoretical values and by measuring them with the performance counter. The theoretical values represent the time that the hardware module computes an operation whereas values given by the performance counter always include some overhead caused by Nios II. Thus, both of these values are provided in Table 2. Measured timings in Table 2 are averages from five runs. End-to-end time includes printings to the console and communication with the host computer using host file system. Computation time is the time consumed for computation of the ECDSA operations. ECC only time denotes the time that is taken by elliptic curve operations. Theoretical computation time assumes that parallel computation is used for all operations whenever possible. The time required in interfacing is neglected in theoretical times but included in measured times, and theoretical times equal with the actual computation times without

Table 2: Timings in milliseconds

Operation	Theoretical	Measured
Key pair generation		
End-to-end	n/a	1143.32
Computation	0.28	**0.60**
ECC only	0.28	0.50
Signature generation		
End-to-end	n/a	2073.84
Computation	0.35	**0.94**
ECC only	0.28	0.54
Signature verification		
End-to-end	n/a	2296.73
Computation	0.67	**1.61**
ECC only	0.55	1.00

time spent in interfacing. Notice that the bolded values in Table 2 should be used for comparisons to other designs because the user interface was designed only for demonstration purposes.

Key pair generation is expectedly the fastest operation because it requires only one point multiplication and generation of a random integer. Verification which requires computation of two point multiplications is, again, expectedly the slowest operation. Elliptic curve point multiplication dominates in computation of all three operations.

5 List of Possible Improvements

- Modular arithmetic components are currently straightforward implementations of simple algorithms and considerable increases in performance and reductions in area would probably apply if more efficient algorithms were implemented.
- The two elliptic curve point multiplications in verification could be accelerated by using multiple point multiplication techniques. In these techniques, both point multiplications are computed simultaneously resulting in considerable increase in speed.
- Koblitz curves could be used instead of general elliptic curves. This would speed up elliptic curve operations by approximately 50 % as shown in [8].

References

[1] Altera Corporation. Cyclone II device handbook, February 2007.

[2] V.S. Dimitrov, K.U. Järvinen, M.J. Jacobson, W.F. Chan, and Z. Huang. FPGA implementation of point multiplication on Koblitz curves using Kleinian integers. In *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 445–459, Yokohama, Japan, October 10–13, 2006.

[3] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[4] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78(3):171–177, September 1988.

[5] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, August 2001.

[6] K. Järvinen. Design and implementation of a SHA-1 hash module on FPGAs. Technical report, Helsinki University of Technology, Signal Processing Laboratory, November 2004. `http://wooster.hut.fi/~kjarvine/documents/sha.pdf`.

[7] K. Järvinen, J. Forsten, and J. Skyttä. FPGA design of self-certified signature verification on Koblitz curves. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 2007*, Vienna, Austria, September 10-13, 2007. To appear.

[8] K. Järvinen and J. Skyttä. On parallelization of high-speed processors for elliptic curve cryptography. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2007. Submitted.

[9] J. López and R. Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 316–317, Worcester, Massachusetts, USA, August 12–13, 1999.

[10] P. L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, January 1987.

[11] National Institute of Standards and Technology (NIST). Digital signature standard (DSS). *Federal Information Processing Standard, FIPS PUB 186-2*, January 27, 2000.

[12] National Institute of Standards and Technology (NIST). Secure hash standard (SHS). *Federal Information Processing Standard, FIPS PUB 180-2*, August 1, 2002.

[13] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 2nd edition, 1996.

[14] C. C. Wang, T. K. Troung, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed. VLSI architectures for computing multiplications and inverses in $GF(2^m)$. *IEEE Transactions on Computers*, 34(8):709–717, August 1985.

[15] T. Wollinger, J. Guajardo, and C. Paar. Security on FPGAs: State-of-the-art implementations and attacks. *ACM Transactions on Embedded Computing Systems*, 3(3):534–574, August 2004.

Appendix: Algorithms

Algorithm 1 Point multiplication using Montgomery's ladder

Require: Point P, integer $k = \sum_{i=0}^{\ell-1} k_i 2^i$ where $k_i \in \{0, 1\}$ and $k_{\ell-1} = 1$
Ensure: Point $Q = kP$
  $P_1 \leftarrow P$ and $P_2 \leftarrow 2P$
  for $\ell - 2$ **downto** 0 **do**
    if $k_i = 0$ **then**
      $P_1 \leftarrow 2P_1$ and $P_2 \leftarrow P_1 + P_2$
    else
      $P_1 \leftarrow P_1 + P_2$ and $P_2 \leftarrow 2P_2$
    end if
  end for
  $Q \leftarrow P_1$

Algorithm 2 Modular addition, $c = a \times b \bmod n$

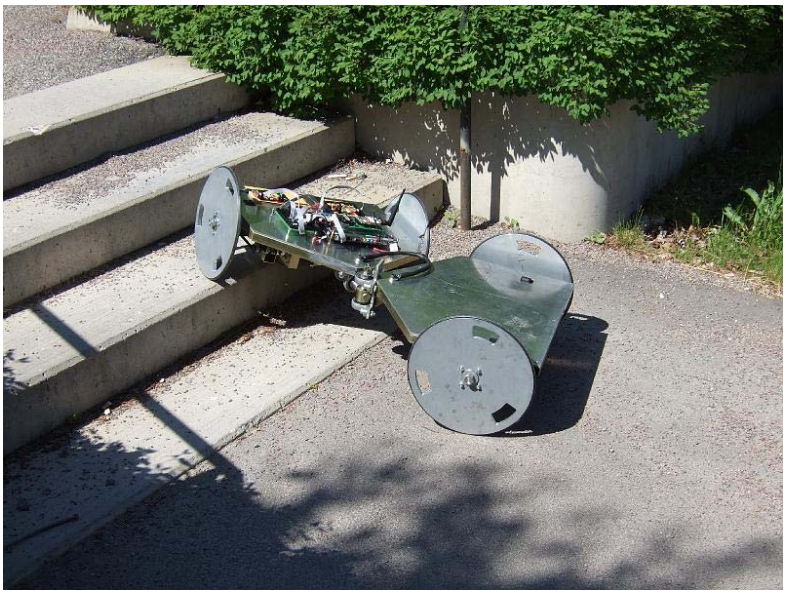
Require: Two ℓ-bit integers a and b in the interval $[0, n-1]$ and a prime n
Ensure: $c = a \times b \pmod{n}$
  $c \leftarrow 0$
  for $i = 0$ **to** $\ell - 1$ **do**
    if a is odd **then**
      $c \leftarrow c + b$
    end if
    $a \leftarrow \lfloor a/2 \rfloor$ $\{a \ggg 1\}$
    $b \leftarrow 2b$ $\{b \lll 1\}$
    if $b \geq n$ **then**
      $b \leftarrow b - n$
    end if
    if $c \geq n$ **then**
      $c \leftarrow c - n$
    end if
  end for

Algorithm 3 Modular inversion, $c = a^{-1} \bmod n$

Require: An integer a in the interval $[1, n-1]$ and a prime n
Ensure: $c = a^{-1} \pmod{n}$

```
u ← a, v ← n
x1 ← 1, x2 ← 0
while u ≠ 1 and v ≠ 1 do
  while u is even do
    u ← u/2
    if x1 is even then
      x1 ← x1/2
    else
      x1 ← (x1 + n)/2
    end if
  end while
  while v is even do
    v ← v/2
    if x2 is even then
      x2 ← x2/2
    else
      x2 ← (x2 + n)/2
    end if
  end while
  if u ≥ v then
    u ← u − v, x1 ← x1 − x2
  else
    v ← v − u, x2 ← x2 − x1
  end if
end while
if u = 1 then
  if x1 ≥ 0 then
    c ← x1
  else
    c ← x1 + n
  end if
else
  if x2 ≥ 0 then
    c ← x2
  else
    c ← x2 + n
  end if
end if
```

Team MinMyra



Introduction

The MinMyra project has been developed by 4 students as a final project during the spring before doing their master thesis. The project is at the division of Fluid and Mechanical Engineering Systems (FluMeS) at Linköpings University, Linköping Sweden.

The MinMyra is a prototype platform that focuses on mechanical stability to ensure the safety of the platform and its cargo. The platform is made as a welded frame, powered by step engines and controlled by a FPGA (Field Programmable Gate Array). The parts are manufactured from standard parts, easy obtainable and low in cost. The platform shall in future studies carry sensor systems to learn to navigate and detect the surrounding environment.

The purpose of the robot is to build a stable autonomic platform that can safely travel from A to B. It shall be stable enough to carry expensive sensors and electronics. The usage of this sort of robot is in environments where it's dangerous for humans to be, i.e. minefields, mines and contaminated environments as accidents and nuclear facilities. With the right sensors it's a cheap way to solve problems in those areas.

Mechanical design

A mechanical model was developed to describe the movement of the robot. From this model was a robot built. Al the mechanics was drawn in pro/ENGINEER wildfire and produced locally at the university. The mechanics is described in teknisk_dokumentation.pdf. Much care was focused on getting a stable and durable platform.

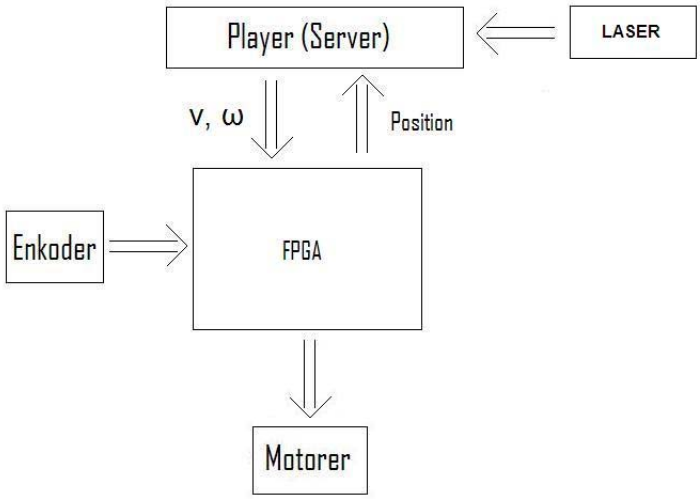
Electronic and software design

The FPGA have a number of support systems, i.e. amplifying the motor control signals or transforming the signals from the wheel encoders. The circuit boards were designed by us.

The FPGA had multiple responsibilities.

- Sending and receiving information with a computer according to player standard interface.
- Calculate wheel velocities according to the received information.
- Transform the velocities to control signals for the stepper motors. This is made by a VHDL block.
- Calculating the distance travelled by each wheel according to encoder data. This is also made by a VHDL block.
- Calculating the position of the robot depending on how long the wheels have travelled and sends it to the player computer.

A schematic picture of the system is shown below.



LASER

A LASER sensor was mounted on top of the robot which gave the robot a possibility to detect and avoid the environment. The LASER was connected to the player computer and the host program sent information to the FPGA to control the robot.

Result

The robot was made and it works very well, it is able to detect and avoid obstacles in a non static environment. Today the robot can not detect mines but with right added sensors it can help with the detection of those. The robot shows that it's possible to build a platform with limited budget, small means and an ordinary workshop.

The robot is 1000x600x300 mm and weighs approximately 25 kg. The travelling speed of the robot is 4 km per hour. The work was presented and a demonstration was made of the robot at "hydraulikdagarna", a conference at the university held by FluMeS.

Technical Documentation

Johan Vestman

Version 1.0

Status

Granskad		
Godkänd		

PROJEKTIDENTITET

MinMyra 07
Linköpings tekniska högskola, IEI

Namn	Ansvar	Telefon	E-post
Patrik Stener	Projektanansvarig	0702-268039	patst577@student.liu.se
Johan Vestman	dokumentansvarig	0707-518125	Johve944@student.liu.se
Patrik Sjöberg	designansvarig	0733-263676	patsj733@student.liu.se
Erik Moqvist	testansvarig	0702-775832	erimo144@student.liu.se

E-postlista för hela gruppen tmms06_minmyra@listserv.ikp.liu.se

Kund: FOI, Linköping
Kontaktperson hos kund: Jonas Nygårds
Kursansvarig: Karl-Erik Rydberg, 013-281187, karry@ikp.liu.se
Handledare: : Magnus Sethson, 013-282733, magnus.sethson@liu.se

Contents

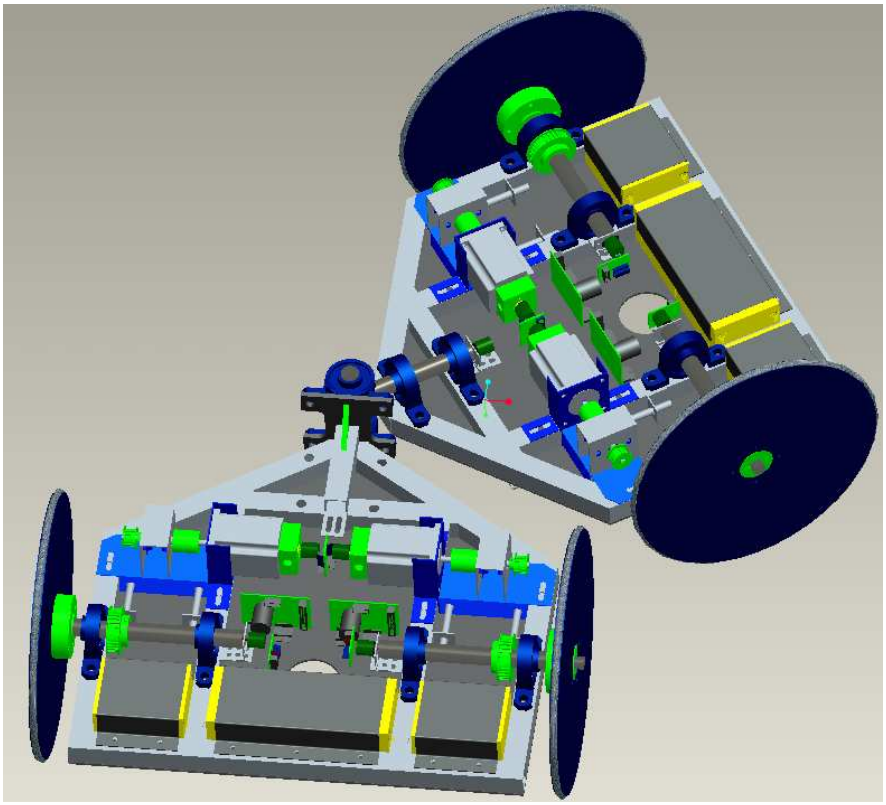
Document history

version	date	Changes	made by	reviewed
1.0	2004-05-07	First version	JV	

1. System overview

1.1. Mechanics

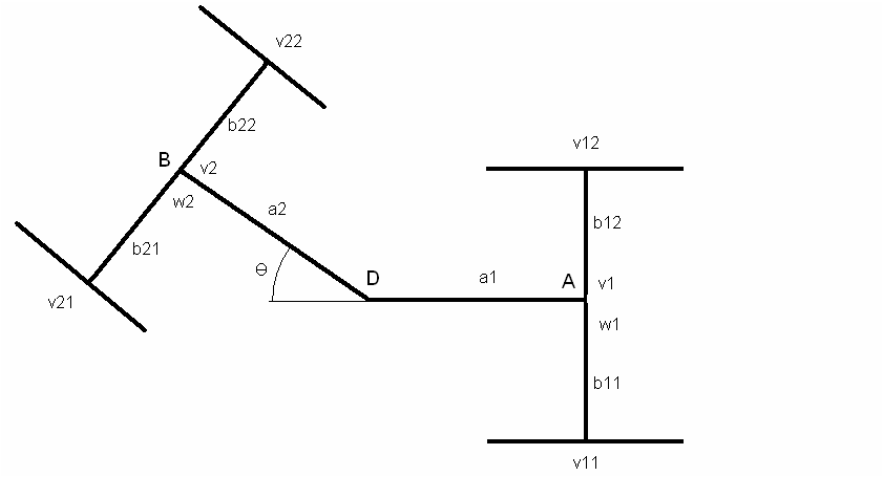
In figure 1 below you can se an overview of the underside of the robot. It shows the cargo bay for the batteries, the wheel axis, the motor block and the mid joint. You can also se the aluminium surface that will be used to attach electronics and our high precision wheels.



Figur 1, overview of the robot

1.2. Mechanical model

We have assumed that the robot is moving on a non slippery surface. The control parameters are V_1 and ω_1 and they will be set by the main program. With this we calculate the speed of the front wheels, v_{11} and $v_{12,}$ see figure 2. The mid joint centre of rotation D will decide what speed the rear wheels, v_{21} and $v_{22,}$ have to be. The v_{21} and v_{22} is a slave system to v_{11} and v_{12}.



Figur 2, Geometric description

In figure 2 above:

Name	Description	SI-Unit
V_1, V_2	Speed of the axis centers.	m/s
ω_1, ω_2	Angle velocity of the axis mid points.	rad/s
b_{11}, b_{12}, b_{21}, b_{22}	Distance from the wheels to the centre of the axis.	m
a_1, a_2	Distance from the mid joint centre of rotation to the front and rear axis.	m
v_{11}, v_{12}, v_{21}, v_{22}	Speeds on the wheels.	m/s
θ	Angle of the mid joint.	rad

The mean speed and the angular velocity at point A (Figure 2) is calculated as:

$$v_1 = \frac{v_{11}b_{12} + v_{12}b_{11}}{b_{11} + b_{12}} \quad \omega_1 = \frac{v_{11} - v_{12}}{b_{11} + b_{12}}$$

$$\begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix} = \frac{1}{b_{12} + b_{11}} \begin{bmatrix} b_{12} & b_{11} \\ 1 & -1 \end{bmatrix} \begin{bmatrix} v_{11}) \\ v_{12} \end{bmatrix} \Rightarrow \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix} \begin{bmatrix} 1 & b_{11} \\ 1 & -b_{12} \end{bmatrix}$$

The velocity of the midpoint D must be the same for both the front part and the rear.

$$V_{D1} = R(\theta) \begin{bmatrix} 1 & 0 \\ 0 & -a_1 \end{bmatrix} \begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix} \qquad V_{D2} = \begin{bmatrix} 1 & 0 \\ 0 & a_2 \end{bmatrix} \begin{bmatrix} v_2 \\ \omega_2 \end{bmatrix}$$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \cos(90 - \theta) \\ -\cos(90 - \theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$V_{D2} = V_{D1}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & a_2 \end{bmatrix} \begin{bmatrix} v_2 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -a_1 \end{bmatrix} \begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix}$$

$$\begin{bmatrix} v_2 \\ \omega_2 \end{bmatrix} = \frac{1}{a_2}\begin{bmatrix} a_2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & -a_1 \end{bmatrix}\begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix}$$

In this case v_{11} and v_{12} is the measured velocities on the front wheel axis, that gives a master slave system.

$$v_2 = \frac{v_{21}b_{22} + v_{22}b_{21}}{b_{21} + b_{22}} \;,\; \omega_2 = \frac{v_{21} - v_{22}}{b_{21} + b_{22}} \;,\; v_1 = \frac{v_{11}b_{12} + v_{12}b_{11}}{b_{11} + b_{12}} \;,\; \omega_1 = \frac{v_{11} - v_{12}}{b_{11} + b_{12}}$$

$$\frac{1}{b_{21} + b_{22}}\begin{bmatrix} b_{22} & b_{21} \\ 1 & -1 \end{bmatrix}\begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \frac{1}{a_2}\begin{bmatrix} a_2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & -a_1 \end{bmatrix}\frac{1}{b_{11} + b_{12}}\begin{bmatrix} b_{12} & b_{11} \\ 1 & -1 \end{bmatrix}\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix}$$

$$\begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \frac{b_{21} + b_{22}}{a_2(b_{11} + b_{12})(-b_{22} - b_{21})}\begin{bmatrix} -1 & -b_{21} \\ -1 & b_{22} \end{bmatrix}\begin{bmatrix} a_2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & -a_1 \end{bmatrix}\begin{bmatrix} b_{12} & b_{11} \\ 1 & -1 \end{bmatrix}\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix}$$

$$\begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \frac{1}{a_2(b_{11} + b_{12})}\begin{bmatrix} a_2 & b_{21} \\ a_2 & -b_{22} \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} b_{12} & b_{11} \\ -a_1 & a_1 \end{bmatrix}\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix}$$

Summary:

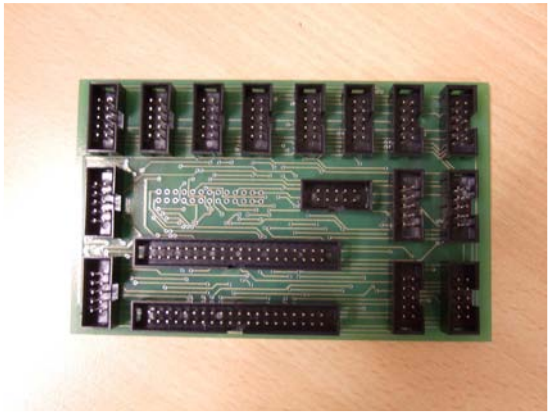
$$\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} 1 & b_{11} \\ 1 & -b_{12} \end{bmatrix}\begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix}$$

$$\begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \frac{1}{a_2(b_{11} + b_{12})}\begin{bmatrix} a_2 & b_{21} \\ a_2 & -b_{22} \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}\begin{bmatrix} b_{12} & b_{11} \\ -a_1 & a_1 \end{bmatrix}\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix}$$

1.3. Electronics

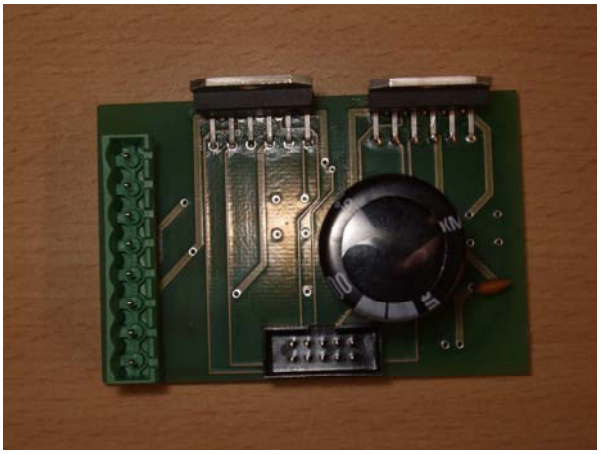
All the circuit boards are designed in a freeware called Eagle. The robot has three basic circuit boards. All circuit diagrams can be found in the Board map in the document files.

The IO board is for the signal routing between the FPGA and the motors and the sensors.



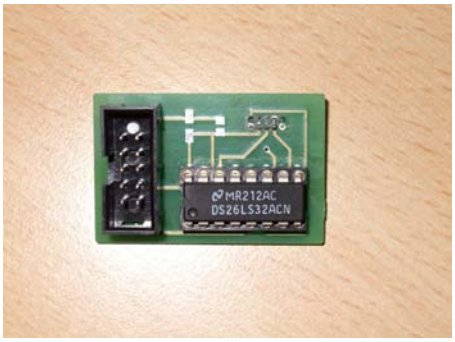
Figur 3, IO board

The stepper motor board is based on a LMD 18200 circuit. It's a circuit that can be used for direct control and power drive for a stepper motor. The circuit can deliver 3 Ampere at continues drive and 6 ampere at peak power and that's enough for our motors. The voltage span is +12 to + 55 volt and we have chosen to run at 36 volt to the motors.

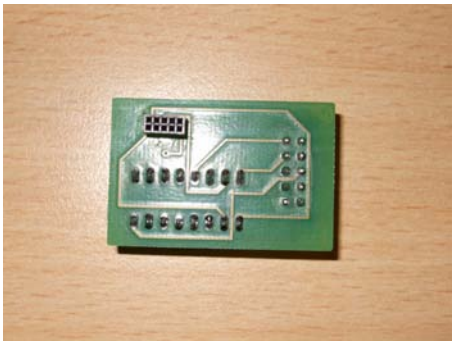


Figur 4, Stepper motor board

The sensor board is based on a DS26LS32. The DS26LS32 is a quad differential line receiver for data transmission. This circuit was recommended by Avago for the encoders. It transforms a differential signal to an absolute signal.



Figur 5, Sensor board circuit



Figur 6, sensor connection

Optocoupler is may be needed between the FPGA and the motor board. This is to ensure that the signals get a correct ground. We have used the Optocoupler bought from ELFA. The circuit board is handmade.

A 9 voltage DC supply is made to power the optocouplers and the wireless RS232.

1.4. Field Programmable Gate Array

A Field Programmable Gate Array or FPGA is a board with programmable connections and logic components. We have an Altera DE2 development and education board that includes:

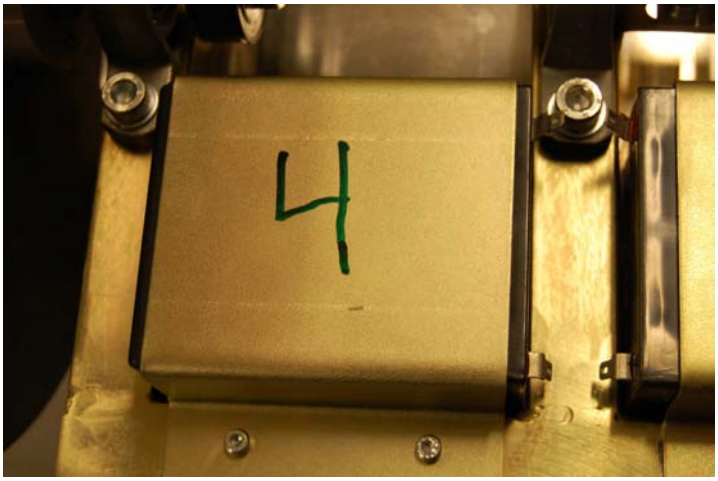
- Altera Cyclone II (2C35).
- 16Mbit Serial configuration device for AS mode.
- Built-in USB blaster with enhanced API link IP
- TV decoder for NTSC/PAL
- 24-bit CD-quality audio codec
- VGA DAC
- USB host and device
- Ethernet 100/10Mbps
- SRAM, SDRAM Flash SD card connector.
- RS232, IrDA, PS/2

1.5. Motors

The motors that are used are stepper motors called KH56QM-961, sold by Aratron. It's a two phase stepper motor, bipolar with a double axis. The motor uses 200 steps per round. The motors are magnetic balanced and a stepper motor is a good choice if you want to have good position accuracy. The stepper motor also has high torque at low speed. The double axis is needed to attach the sensor. The main issue with these motors is vibrations and noise at low speed. The solution to this is to use micro steps but we have used half steps in our control of the motor so we get vibrations. We also had to make a sensor holder and used epoxy glue to attach it to the motor because there was no drilled holed on the back of the motor, see Figure 11, number 7.

1.6. Batteries

The batteries are a 12 voltage 2,9Ampere lead-accumulator bought at Biltema. We use six batteries to create two 36 voltage batteries for the motors and a single battery for the FPGA. There are a total of eight batteries on the robot, but only seven is used so there is one extra battery for future use. The batteries are mounted under the robot in a cargo bay.
The battery holder is handmade, a basic sketch is found in the solid models in pro engineer.



Figur 7, single battery.

1.7. Encoder

The encoder AEDA-3300-TAQ from Avago is a miniature high resolution encoder. The resolution is spanning from 600 to 20000 CPR depending on the model. It's one of the smallest high performance encoders on the market. The one we choose have 4000 CPR. We have a total of 10 encoders possible to use on the robot.

1.8. Programming

The processor on the FPGA that we have defined does the calculations for the speed for the different wheels. The programming language is C. All the calculations about the position is made on the FPGA. All the related files are found on the presentation CD.

We use VHDL for programming the motor control and to translate information from the encoders.

We use serial transmission of data that we transfer with a wireless connection

A limitation is the minimum turning radius gives the constraints for the velocity V and ω. If we assume that the distance between the mid joint and the both wheel axis is the same and the max angle is 60 degrees then:

$$\tan\left(\frac{180-\theta}{2}\right)=\frac{r}{a_1}$$

$$r = a_1 \tan\left(\frac{180-\theta}{2}\right)=\left|\begin{matrix}\theta = 60\\ a_1 = 0.352\end{matrix}\right| = 0.61 \text{ Meter.}$$

We round that number upwards so the limitation in turning radius gives that the quote $\frac{V}{\omega} \geq 0.65$, where V and ω are the control parameters.

1.9. Player driver

The control on the PC side is implemented as a player driver. Player is an interface to communicate with other robots and it's easy to build more modules to the robot such as a laser module.

2. System modules

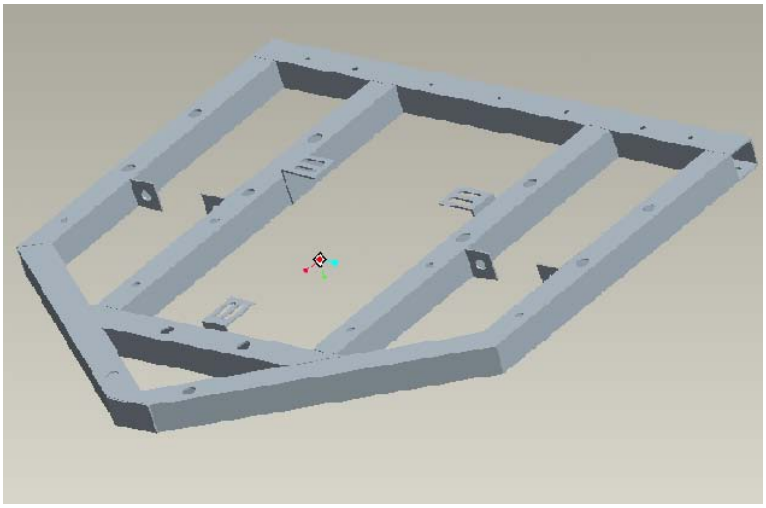
2.1. Framework

The robot two frames are made of welded square pipes. The framework ensures a stiff and lightweight construction. An overview of one of them is shown in figure 8.
The drawings for the pipes are found in drawing_23, drawing_24, drawing_25, drawing_26, drawing_28, drawing_29 and drawing_30.

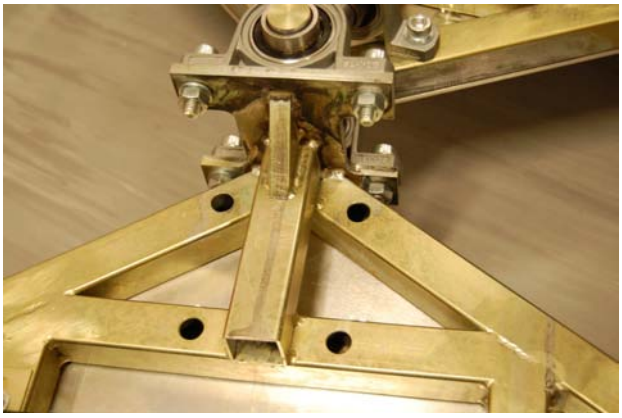
In figure 9 the mid joint welding is shown, and in figure 10 a cover plate is welded instead of the vertical bearing holder. The plate in figure 10 is found in drawing_20.

The L-brackets for the encoders and the belt drive is found in drawing_14 and drawing_15 and welded to the frame.

All the drilling is made when the frame is manufactured to ensure that all the holes have the correct distances between each other. The drilling and the placement of the L-brackets are found in drawing_31. To mount the motor block and the bearing housings we use flat head rivet nut in the 11.1mm and 7mm drilled holes. These shall be mounted on the opposite side of the bearings and motor block.



Figur 8, A welded frame



Figur 9, Midjoint welding

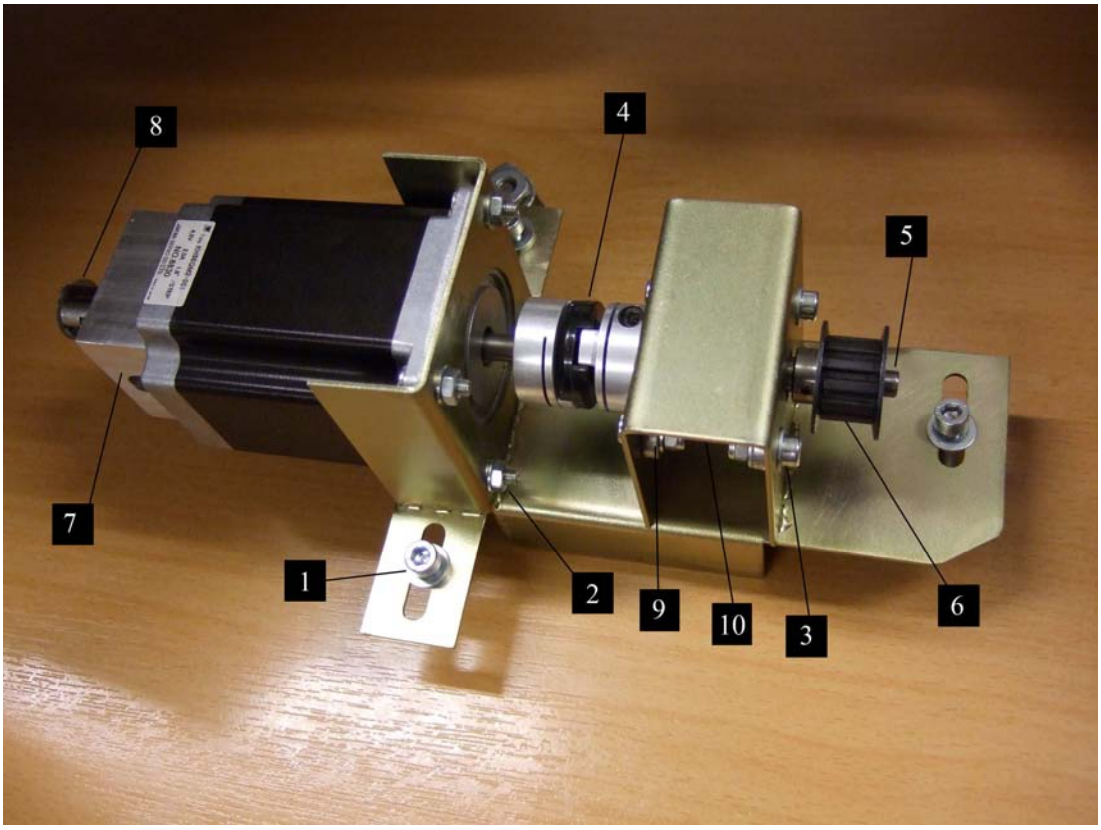
Figur 10, Mid joint welding

2.2. Aluminum plate

The top of the robot is a 3 mm plate that is screwed to the frame with slotted countersunk screw M4-6. On its surface the FPGA and the IO-board are mounted and under the plate the stepper motor boards are mounted. The plate is found in drawing_1. The two parts of the robot shall be connected with a grounding cable between the plates. This plate is used as a grounding plane for the electronics.

2.3. Motor block

The motor block is welded together as bent steel plates. The individual parts are found in drawing_17, drawing_32 and drawing_18/drawing_19 and the screw dimensions and other parts are listed below. There is a difference between the left and the right motor block see drawing_18 and drawing_19.



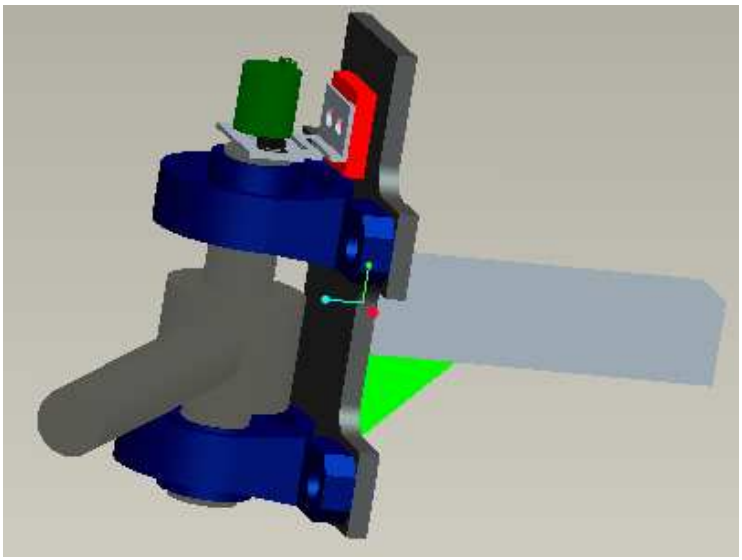
Figur 11, the motor block and numbering of the included parts

Listings of the parts in figure 11 above.

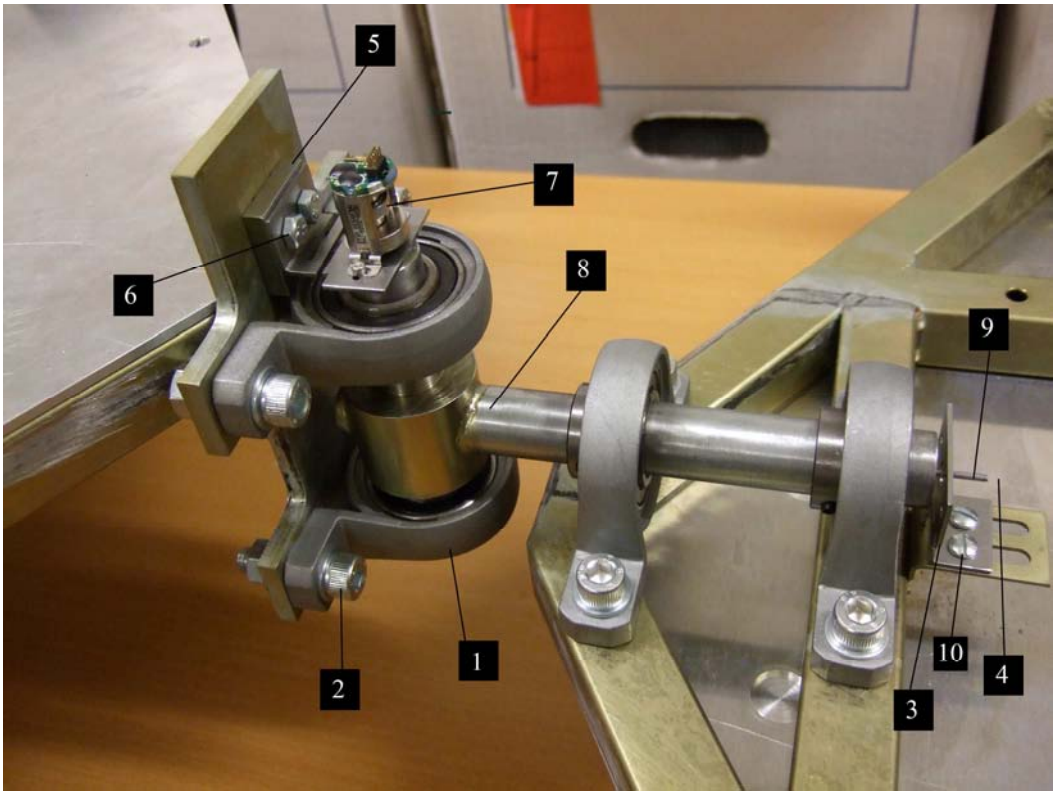
Number	Name and description	Quantity
1	Hex cap pan head screw M4-30, 3 washers	3
2	Slotted pan head screw M3-14, 8 washers	4
3	Hex cap pan head screw M4-12, 8 washers and 4 locknuts	4
4	Oldham coupling, 5mm/6.35mm	1
5	Axis 5mm, drawing_7	1
6	Pulley 12mm	1
7	Aluminium encoder holder, drawing_8	1
8	Encoder	1
9	Flanged bearing housing	1
10	Locking mechanism for the bearings, drawing _16	1

2.4. Mid joint

The mid joint is made by two welded parts that holds together with two bearing housings. One part is welded with parts from drawing_21 drawing_22 and drawing_27 on to the frame. The other half is welded with parts from drawing_5 and drawing_6.



Figur 12, A solid model that shows the mid joint



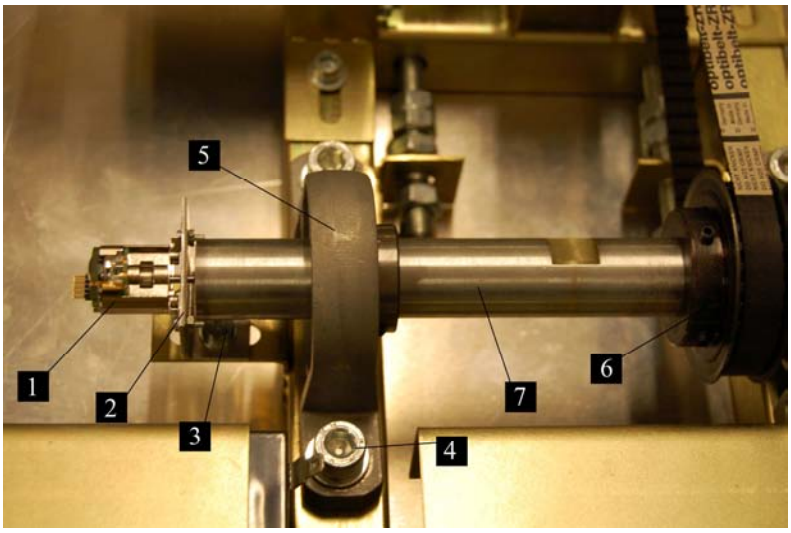
Figur 13, the mid joint and numbering of the included parts.

Listings of the parts in figure 13 above.

Number	Name and description	Quantity
1	Bearing house	4
2	Hex cap pan head screw M8-30, 12 washers 4 nuts	8
3	Encoder holder, drawing_9	1
4	Future location of the roll encoder.	
5	Washer, drawing _13	1
6	Bolt M4-20,2 washers	2
7	Encoder	1
8	Welded mid joint, drawing_5 and drawing_6	1
9	Encoder axis, drawing_2	1
10	Sloted pan head screw M3-10, 2 washers and 2 nuts	2

2.5. Wheel axis

The wheel axis is based on a 20 mm axis. The end of the axis is lathed to fit the wheels, see drawing_4.



Figur 14, wheel axis

Listings of the parts in figure 14 above.

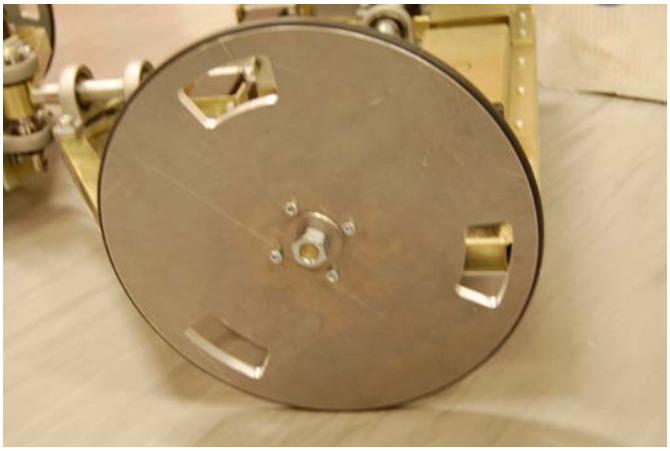
Number	Name and description	Quantity
1	Encoder	1
2	Encoder holder, drawing_9	1
3	Slotted pan head screw M3-10, 2 washers and 2 nuts	2
4	Hex cap pan head screw M8-30, 4 washers	4
5	Bearing house	2
6	Pulley 12mm	1
7	Wheel axis, 20 mm. see drawing_4	1

2.6. Wheels

Two types of wheels are manufactured, a high precision wheel for testing and indoor driving, and an outdoor terrain wheel with lower precision.

Figur 15, Low precision terrain wheel

The low precision wheel above in figure 10 is an aluminum rim manufactured by our workshop. The drawing for the rim is found in drawing_12. The tire we bought is a 12 inch tire for a bicycle at Biltema.



Figur 16, High precision indoor wheel

The high precision wheel above in figure 11 is an aluminium rim made by our workshop. The rim is assembled by two parts. The centre hub is found in drawing_33, and the aluminium pate is found in drawing_34. The two parts is held together by four M5 bolts.

The rubber is made from an o-ring that is glued to the right diameter.

2.7. IO-board

One board is needed. The Circuit diagram is found in Interface_Board1_1.

Description	Supplier and art. Nr.	Quantity
10 pin low profile connector	ELFA 43-155-03	15
26 pin low profile connector	ELFA 43-151-56	1
40 pin low profile connector	ELFA 43-155-60	2

2.8. Step motor board

One board is needed for each motor. The circuit diagram is found in Motor_Board1_1.

Description	Supplier and art. Nr.	Quantity
10 pin low profile connector	ELFA 43-155-03	1
8 pin connector	ELFA 48-452-28	1
Capacitor	ELFA 67-544-93	1
Control circuit for step motors	ELFA 73-288-34	2
Capacitor 47 µF 50V	ELFA 67-013-87	4

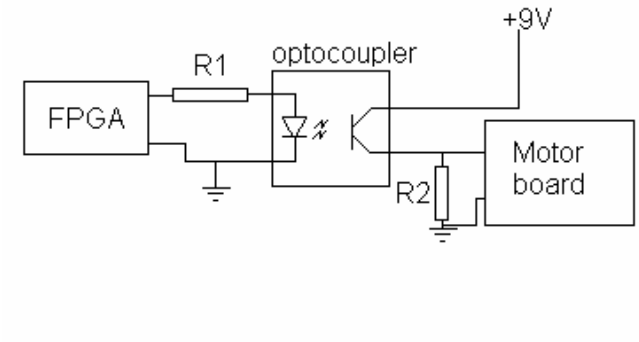
2.9. Sensor board

One board is needed for each sensor. The circuit diagram is found in Enchoder_Board1_1.

Description	Supplier and art. Nr.	Quantity
10 pin low profile connector	ELFA 43-155-03	1
IC Holder	ELFA 48-135-80	1
10 pin dual row vertical socket	Farnell 102-2299	1
IC DS 26LS32 quad differential Line Receiver.	ELFA 73-143-62	1
Capacitor 47pF	ELFA 65-778-52	1

2.10. Optocoupler board

One Optocoupler is needed for each signal to the step motor board. A basic drawing is found in figure 17. The R1=383Ω and the R2=560Ω. Six optocouplers is needed for each motor board.



Figur 17. Basic drawing for one optocoupler.

The table below is the components for six optocouplers for the signals to one motor board.

Description	Supplier and art. Nr.	Quantity
Resistance 383Ω	ELFA 60-717-99	6
IC Holder	ELFA 48-135-80	1
IC Holder	ELFA 48-135-49	1
Resistance 560Ω	ELFA 60-719-97	6
Optocoupler	ELFA 75-362-46	1
Optocoupler	ELFA 75-362-20	1

2.11. 9V DC supply

One is needed for each optocoupler board and one is needed for the wireless RS232, the material for one is found in the table below.

Description	Supplier and art. Nr.	Quantity
L7809 9V 1,5A	ELFA 73-091-23	1
Capacitor 0,1µF	ELFA 65-183-69	1
Capacitor 0,33µF	ELFA 65-184-27	1

2.12. Cable specification

The cables connected from the IO-board and the sensors and motors are listed below.

	Header and pin		Header and pin	Description
IO_Board	J_1-1	Encoder Board	J_2-1	Output A
	J_1-2		J_2-2	Output B
	J_1-3		J_2-3	Out I
	J_1-4		J_2-4	Vcc
	J_1-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_2-1	Encoder Board	J_2-1	Output A
	J_2-2		J_2-2	Output B
	J_2-3		J_2-3	Out I
	J_2-4		J_2-4	Vcc
	J_2-5		J_2-5	GND
	J_3-1		J_2-1	Output A
	J_3-2		J_2-2	Output B
	J_3-3		J_2-3	Out I
	J_3-4		J_2-4	Vcc
	J_3-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_4-1	Motor board	J_1-1	GND
	J_4-2		J_1-2	GND
	J_4-3		J_1-3	PWM-B
	J_4-4		J_1-4	PWM-A
	J_4-5		J_1-5	BREAK-B
	J_4-6		J_1-6	BREAK-A
	J_4-7		J_1-7	DIR-B
	J_4-8		J_1-8	DIR-A
	J_4-9		J_1-9	I-SENSE-B
	J_4-10		J_1-10	I-SENSE-A

	Header and pin		Header and pin	Description
IO_Board	J_5-1	Motor board	J_1-1	GND
	J_5-2		J_1-2	GND
	J_5-3		J_1-3	PWM-B
	J_5-4		J_1-4	PWM-A
	J_5-5		J_1-5	BREAK-B
	J_5-6		J_1-6	BREAK-A
	J_5-7		J_1-7	DIR-B
	J_5-8		J_1-8	DIR-A
	J_5-9		J_1-9	I-SENSE-B
	J_5-10		J_1-10	I-SENSE-A

	Header and pin		Header and pin	Description
IO_Board	J_6-1	Encoder Board	J_2-1	Output A
	J_6-2		J_2-2	Output B
	J_6-3		J_2-3	Out I
	J_6-4		J_2-4	Vcc
	J_6-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_7-1	Encoder Board	J_2-1	Output A
	J_7-2		J_2-2	Output B
	J_7-3		J_2-3	Out I
	J_7-4		J_2-4	Vcc
	J_7-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_8-1	Encoder Board	J_2-1	Output A
	J_8-2		J_2-2	Output B
	J_8-3		J_2-3	Out I
	J_8-4		J_2-4	Vcc
	J_8-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_9-1	Encoder Board	J_2-1	Output A
	J_9-2		J_2-2	Output B
	J_9-3		J_2-3	Out I
	J_9-4		J_2-4	Vcc
	J_9-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_10-1	Encoder Board	J_2-1	Output A
	J_10-2		J_2-2	Output B
	J_10-3		J_2-3	Out I
	J_10-4		J_2-4	Vcc
	J_10-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_11-1	Encoder Board	J_2-1	Output A
	J_11-2		J_2-2	Output B
	J_11-3		J_2-3	Out I
	J_11-4		J_2-4	Vcc
	J_11-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_12-1	Motorboard	J_1-1	GND
	J_12-2		J_1-2	GND
	J_12-3		J_1-3	PWM-B
	J_12-4		J_1-4	PWM-A
	J_12-5		J_1-5	BREAK-B
	J_12-6		J_1-6	BREAK-A
	J_12-7		J_1-7	DIR-B
	J_12-8		J_1-8	DIR-A
	J_12-9		J_1-9	I-SENSE-B
	J_12-10		J_1-10	I-SENSE-A

	Header and pin		Header and pin	Description
IO_Board	J_13-1	Encoder Board	J_2-1	Output A
	J_13-2		J_2-2	Output B
	J_13-3		J_2-3	Out I
	J_13-4		J_2-4	Vcc
	J_13-5		J_2-5	GND

	Header and pin		Header and pin	Description
IO_Board	J_14-1	Motor board	J_1-1	GND
	J_14-2		J_1-2	GND
	J_14-3		J_1-3	PWM-B
	J_14-4		J_1-4	PWM-A
	J_14-5		J_1-5	BREAK-B
	J_14-6		J_1-6	BREAK-A
	J_14-7		J_1-7	DIR-B
	J_14-8		J_1-8	DIR-A
	J_14-9		J_1-9	I-SENSE-B
	J_14-10		J_1-10	I-SENSE-A

The connections between the motor board and the motor are listed below.

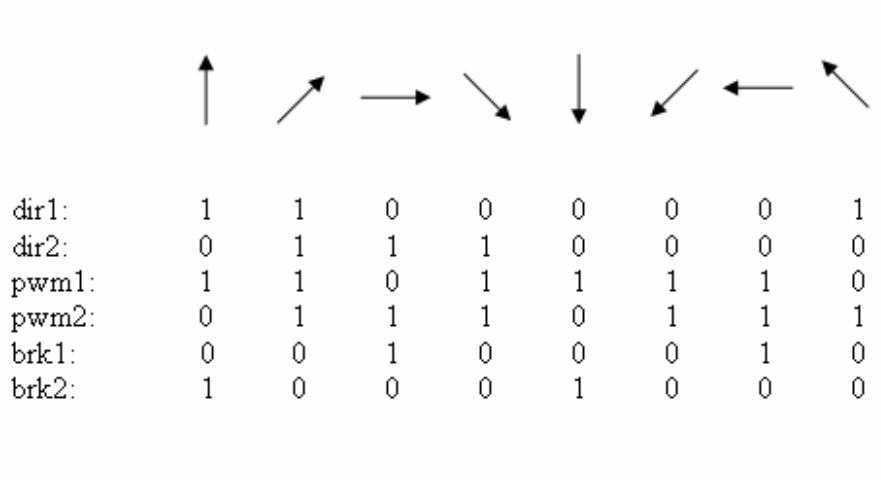
	header and pin		Description
Motor Board	X1-1	Input GND	
	X1-2	Input +36V	
	X1-3		Output 1 winding A
	X1-4		Output 2 winding A
	X1-5		Output 1 winding B
	X1-6		Output 2 winding B

The connection between the encoder board and the encoder is listed below.

	Header and pin		
Encoder board	J_1-1	Encoder	Pin 1
	J_1-2		Pin 2
	J_1-3		Pin 3
	J_1-4		Pin 4
	J_1-5		Pin 5
	J_1-6		Pin 6
	J_1-7		Pin 7
	J_1-8		Pin 8
	J_1-9		Pin 9
	J_1-10		Pin 10

2.13. Stepper Motor control

Related files are stepper_motor_ctrl.VHDL
A counter counts up on a clock signal, and then it's compared with period. If it's equal the current state is updated to next state and the counter is set to zero.



Figur 18, Bit description for the motor control

The figure 18 above shows the output to turn the motor axis.
The table below is listing all the signals connected to the stepper motor control.

Signal in	Clock	Clock frequency
	Dir	Rotation direction
	Free	Must be equal to 1 if the motor axis is going to turn free.
	Brk	If Brk equals to 1 then the motor will break.
	Period[31..0]	The number of clock pulses between each step

Signal out	Dir1	Direction for winding 1
	Dir2	Direction for winding 2
	PWM1	PWM signal for winding 1
	PWM2	PWM signal for winding 2
	Brk1	Break signal for winding 2
	Brk2	Break signal for winding 2
	Pulse_in	Control variable

2.14. Quadature encoder

Related files are quadature_encoder.VHDL
Takes in a and b then returns position and period. Index and velocity is not implemented or used yet. The table below is listing all the signals connected to the Quadature encoder.

Signal in	a	Input a from the encoder
	b	Input b from the encoder
	index	Input index from the encoder
	timeclock	Clock pulse that is equal to 1 every 2^{22} main clock pulse.

Signal out	position	The position of the encoder
	period	The difference in position between two timeclock
	velocity	Not implemented, gives the same value as period.

3. Supliers

	Supplier	Quantity
Oldham coupling	ELFA Hub 6.35mm art.nr: 54-651-17 ELFA Hub 8mm art.nr: 54-651-25 ELFA Disk 25mm art.nr:54-653-15	4 4 4

	Supplier	Quantity
Battery	Biltema Lead accumulator 2.9A. art.nr:80-407 Biltema Lead accumulator charger art.nr:37-711 ELFA, Power switch C1550ATBB, art.nr:35-016-08	16 4 2

	Supplier	Quantity
Motor	Aratron step motor. art.nr: 1007737	4

	Supplier	Quantity
Belt drive	Kedjeteknik Pulley 12mm, art.nr: 12XL 037F Kedjeteknik Pulley 30mm, art.nr: 30XL 037F Kedjeteknik Belt 355mm, art.nr: 140XL 037	4 4 4

	Manufacturer	Quantity
Encoder	Avago, high resolution encoder AEDA-3300_TAQ	10

	Supplier	Quantity
Bearings	Nomo, Flanged bearing housing art.nr: UFL 08 Clas Ohlsson, bearing house, art.nr: 30.4684	8 12

	Supplier	Quantity
Wheels	Momentum, 8mm o-ring. Biltema, 12 inch tire Biltema , 12 inch inner tire	4 meters 4 4

	Supplier	Quantity
FPGA	Altera Cyclone II (2C35)	1

	Supplier	Quantity
Electronics	ELFA, 10 pin low profile header, art.nr: 43-155-03 ELFA, 26 pin low profile header, art.nr: 43-155-45 ELFA, 40 pin low profile header, art.nr: 43-155-60	29 1 2

	ELFA, 8 pin header, art.nr: 48-452-28	4
	ELFA IC-holder, art.nr: 48-135-80	10
	ELFA IC-holder, art.nr: 48-135-49	3
	Farnell 10 pin dual row vertical socket, art.nr:102-2299	10
	ELFA, DS 26LS32 quad differential Line Receiver, art.nr: 73-143-62	10
	ELFA, Control circuit for step motors art.nr: 73-288-34	8
	ELFA, Capacitor 1000µF 100V, art.nr: 67-544-93	4
	ELFA, Capacitor 0,1 µF 63V art.nr: 65-183-69	3
	ELFA, Capacitor 0,33 µF 50V art.nr: 65-184-27	3
	ELFA, Capacitor 47 µF 50V art.nr: 67-013-87	16
	ELFA, insulation, art.nr: 75-646-85	10
	ELFA, insulation, art.nr: 75-646-77	10
	ELFA TLP620-4 quad optocoupler, art.nr: 75-362-46	2
	ELFA TLP620-2 dual optocoupler, art.nr: 75-362-20	2
	ELFA Resistance 383Ω art.nr: 60-717-99	12
	ELFA Resistance 560Ω art.nr: 60-719-97	12
	ELFA Experiment board art.nr: 48-326-48	3
	ELFA L7809CV 9V 1,5A art.nr: 73-091-23	3

	Supplier	Quantity
Cables	ELFA, Female connector 10 pos. art.nr: 43-150-16	29
	ELFA, Female connector 40 pos. art.nr: 43-150-73	4
	ELFA, Female connector 8 pos. art.nr: 48-451-45	4
	ELFA, D-sub 9 pos. art.nr: 44-117-08	4
	ELFA, Flat cable 10 pos. art.nr: 55-660-13	15 meter
	ELFA, Flat cable 26 pos. art.nr: 55-660-96	2 meter
	ELFA, Flat cable 40 pos. art.nr: 55-661-38	2 meter

	Supplier	Quantity
Screws	Järnia, locking nut M4	16
	Järnia, nut M4	40
	Järnia, nut M8	16
	Järnia, nut M14	4
	Järnia, washer M4	88
	Järnia, washer M5	12
	Järnia, washer M8	28
	Järnia, hex cap pan head screw M2-4	20
	Järnia, hex cap pan head screw M4-8	24
	Järnia, hex cap pan head screw M4-10	16
	Järnia, hex cap pan head screw M4-20	2
	Järnia, hex cap pan head screw M5-30	28
	Järnia, hex cap pan head screw M8-30	24
	Järnia, slotted, countersunk screw M4-6	14
	Järnia, flat head rivet nut M8	20
	Järnia, flat head rivet nut M5	12

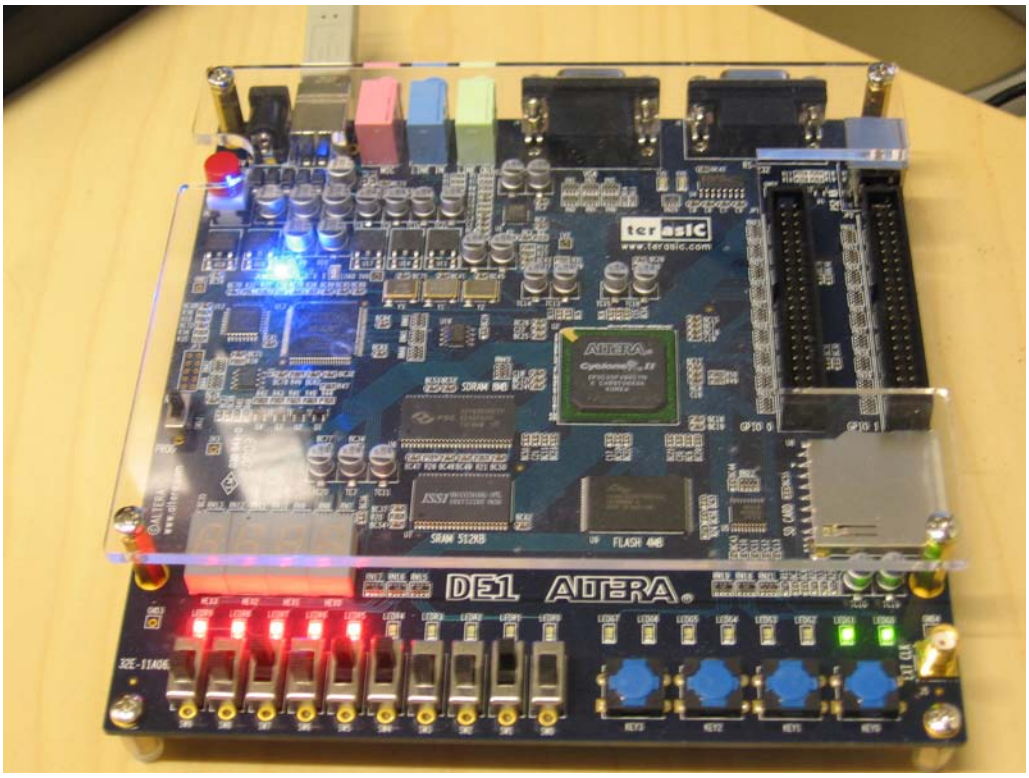
References

Ulf Larsson, Caj Zell, Kalevi Hyyppä and Åke Wernersson. Navigating an articulated vehicle and reversing with a trailer. Computer science and electrical engineering, University of Luleå.

Altera Innovate Nordic 2007

Final Project Report

8/28/2007



Project name:	Leon3 MP on Altera FPGA
Team name:	Team Hervanta
Team ID:	IN00000033
Team members:	Hannu Penttinen
	Tapio Koskinen
Email Addresses:	hannu.penttinen@tut.fi
	tapio.koskinen@tut.fi
Contact No:	00358503708348
Instructor:	Marko Hännikäinen

INTRODUCTION

Growing need for increasing computational power while keeping power consumption low in handheld devices, such as mobile phones and PDAs, has lead to increased usage of multiprocessor systems-on-chip. By using multi-processor systems the required performance can be achieved at lower power consumption than in single-processor systems. The most power efficient solution would be to use application-specific hardware, but it takes a lot of effort to design them. Multiprocessor systems provide a reusable and versatile yet energy efficient solution with much less design effort.

The goal of our project was to implement a configurable SparcLeon3 based multiprocessor system on Altera Cyclone FPGA with arbitrary number of SparLeon3-processors. Ecos was chosen as the operating system since it's been ported to SparcLeon3 by Gaisler research and it's royalty-free. Another goal was to implement an AMBA-to-HIBI-interface to provide a DMA access from AMBA to HIBI-bus [1]. The HIBI-bus could be used for inter-processor communication and therefore the AMBA-bus could be used as processor's internal bus. Our system can be used for prototyping various multi-processor systems with arbitrary number of processors on FPGA. This enables concurrent software and hardware development which in turn decreases the development time.

The starting point for our work was a project we carried out at Tampere University of Technology's Project course last spring. In the course we implemented a similar system on Nios Development Board, Stratix Professional Edition.

In this project, we achieved the following results:

- Easily configurable symmetric multiprocessor system of 2 SparcLeon3-processors with eCos RTOS
- Implemented AMBA-to-HIBI DMA interface
- Implemented AMBA-to-HIBI DMA driver for eCos

Each of the results is discussed in more detail in the following chapters.

SYMMETRIC MULTIPROCESSOR SYSTEM

Our goal was to implement a similar system to a system described in [3]. Our planned system is also depicted in figure 1. The idea was that both processors had their own instruction and data memories and that they both were totally independent of each other. This way the software would be easy to compile and link, since there would be no issues with linking addresses etc. as it would be with shared memories. However, due to the memory requirements of eCos, we noticed that there was not enough on-chip memory for implementing the system.

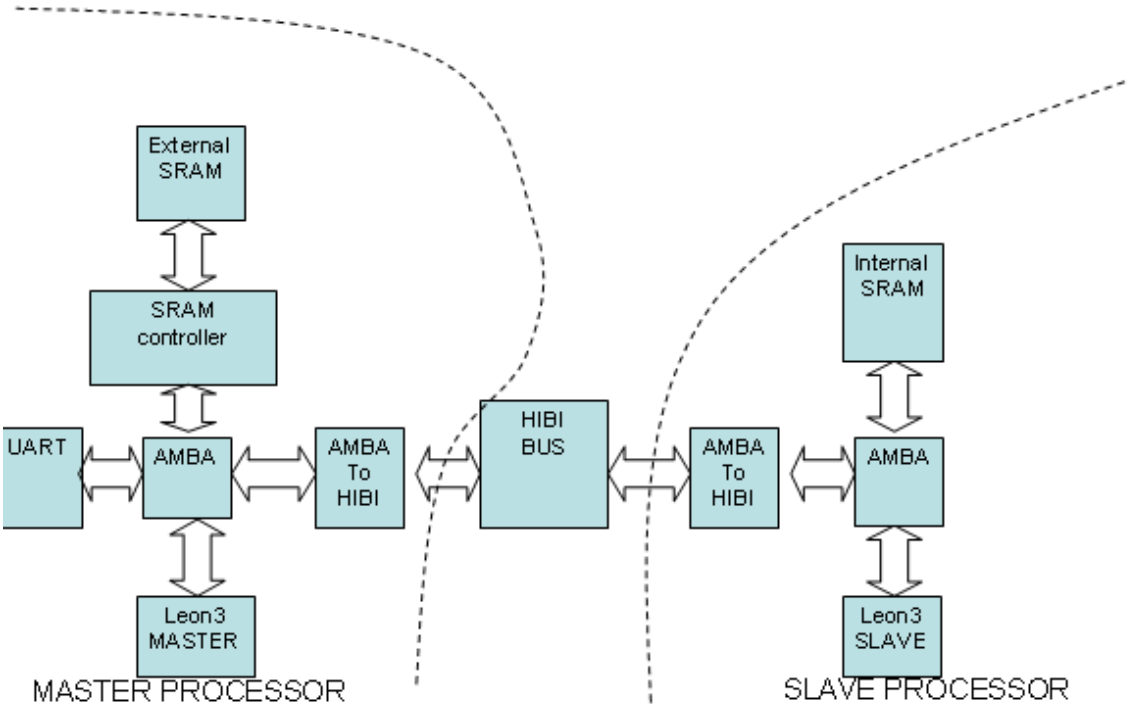


Figure 1. Planned multiprocessor system

Our next proposal was to use external SRAM as shared program memory for the processors. Data memories would still be non-shared. In order to do this we planned to connect the slave processor's AMBA bus to master's AMBA bus with uni-directional AMBA-bridge to provide slave access to external SRAM program memory. This attempt failed, since the AMBA-bridge was available only in the commercial version of Gaisler Reasearch's GRLIB.

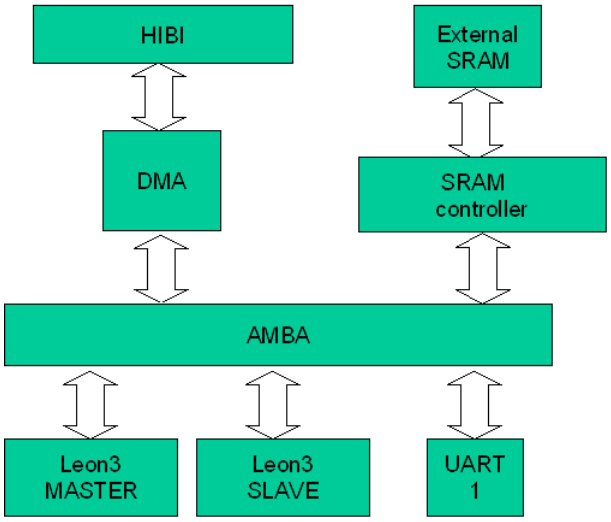


Figure 2. Realized multiprocessor system

Finally we decided to implement a symmetric multiprocessor system depicted in figure 2. The number of Leon3's can be selected by changing one VHDL-generic. Due to limited resources of Cyclone FPGA, only two processors were implemented in our system. The system has two AMBA-to-HIBI DMA units for testing the AMBA-to-HIBI interface. In a real system only one DMA would be used.

The originally planned system could be derived from this system with small effort if an AMBA-bridge was available: first implementing one instance with one Leon3, AMBA-to-HIBI DMA, external SRAM and an AMBA-bridge and then implementing second instance with one Leon3, AMBA-to-HIBI DMA and an AMBA-bridge. Finally these two instances could be connected with AMBA-bridge to provide program memory access to slave processor. This system would be the same kind of architecture that is used in [3] and it's depicted in figure 3.

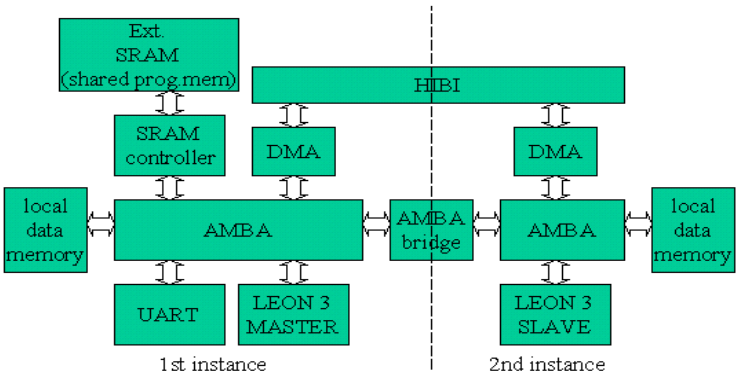


Figure 3. Proposed system

ECOS

INSTALLATION

Five things are needed to get eCos running on FPGA:

1. Cygwin
2. eCos configuration tool
3. eCos repository
4. Cross-compiler for Sparc architecture
5. Grmon for downloading software to FPGA.

These can be found in following addresses: http://www.cygwin.com/setup.exe,
http://www.ecoscentric.com/snapshots/configtool-060710.exe.bz2
ftp://gaisler.com/gaisler.com/ecos/src/ecos-rep-1.0.7.tar.gz,
ftp://gaisler.com/gaisler.com/bcc/bin/windows/sparc-elf-3.4.4-1.0.29d-cygwin.tar.bz2,
ftp://gaisler.com/gaisler.com/grmon/GrmonRCP-eval-0.8.zip.

It is also possible to configure eCos on Linux, in which case other versions of these programs are needed from Gaisler's website (and naturally Cygwin is not needed at all).

Installation is pretty straightforward. Cygwin is fine with default settings. Others can be unpacked basically anywhere ("tar xvf <package name>", configtool: "bunzip2 <file>"). eCos configuration tools needs the location of eCos repository (packages-directory) and the location of cross-compiler (sparc-elf-3.4.4/bin). Repository location can also be specified with ECOS_REPOSITORY environment variable. GrmonRCP needs Java, but if that is a problem there is also a command line tool for downloading software to FPGA (ftp://gaisler.com/gaisler.com/grmon/grmon-eval-1.1.21.tar.gz).

Before starting with the configuration tool, we install our own modifications to eCos repository. There is Dmahibi-package/driver, which should be unpacked to ecos-repository/packages/io/dmahibi. Then there is a modified ecos.db, which has definitions for dmahibi-package and should be copied to ecos-repository/packages-directory. Now we are ready to start the eCos configuration tool.

CONFIGURING AND COMPILING ECOS

For a quick start there is leon3.ecc file enclosed, which has ready-made configuration. These are the steps to create it: Open configuration tool and choose Build-> Templates. Choose Leon3 processor. Then choose Build->Packages, Dma for Hibi, Add, OK. Now let's enable SMP support. Go to eCos HAL / Sparc architecture and check SMP support. Then go to eCos kernel and check SMP support. Now save the configuration with File->Save as.

To compile eCos choose Build->Generate Build Tree and Build-> Library. In the directory where configuration was saved (ecc-file) there will be created directories xyz_build and xyz_install. Xyz_build contains object files which are not necessarily needed anymore (naturally if left alone they speed up the following compiling processes). In the install directory there are many header files (include-directory) and in the lib directory there is a library libtarget.a which is linked into applications and target.ld, a linker script.

Applications can be compiled and linked with eCos on Cygwin command line with the following command:

sparc-elf-gcc -g -Ixyz_install/include -Lxyz_install/lib -Ttarget.ld -nostdlib hello.c -o hello

-I tells where headers are located. -L tells where library is. -Ttarget.ld specifies the linker script which tells the memory addresses to linker. -nostdlib tells the compiler not to use standard libraries (everything should be in libtarget.a). -g just tells to add debugging information. Hello.c is the code file and -o hello tells linker to name binary file as "hello".

TEST SOFTWARE

Test software starts three threads which sleep somewhat random (but short) time. First thread sends every now and then data over HIBI to thread number three. Second thread does basically nothing. eCos schedules these threads automatically on all CPUs. Source code and binaries are attached.

Downloading compiled software to FPGA is done with Grmon. After the FPGA board is programmed, reseted (key0) and debugger enabled (key1), GrmonRCP can be started. Choose Initialize target, Serial, com1 (this can vary) and 115200. Also enable UART loopback. Load program file and click Run.

OTHER INFORMATION

There are good eCos manuals Sourceware's website. There is a user guide, a reference manual and a component writer's guide. These can be found at http://ecos.sourceware.org/docs-2.0/. Then there is also a mailing list for questions: http://ecos.sourceware.org/ml/ecos-discuss/.

AMBA-TO-HIBI INTERFACE

This block provides a DMA connection from AMBA-bus to HIBI-bus. It is divided into three sub-blocks: dma_ctrl, which functions as a configuration unit, dma_tx, which implements one transmit (tx) channel and dma_rx, which implements an arbitrary number of receive (rx) channels.

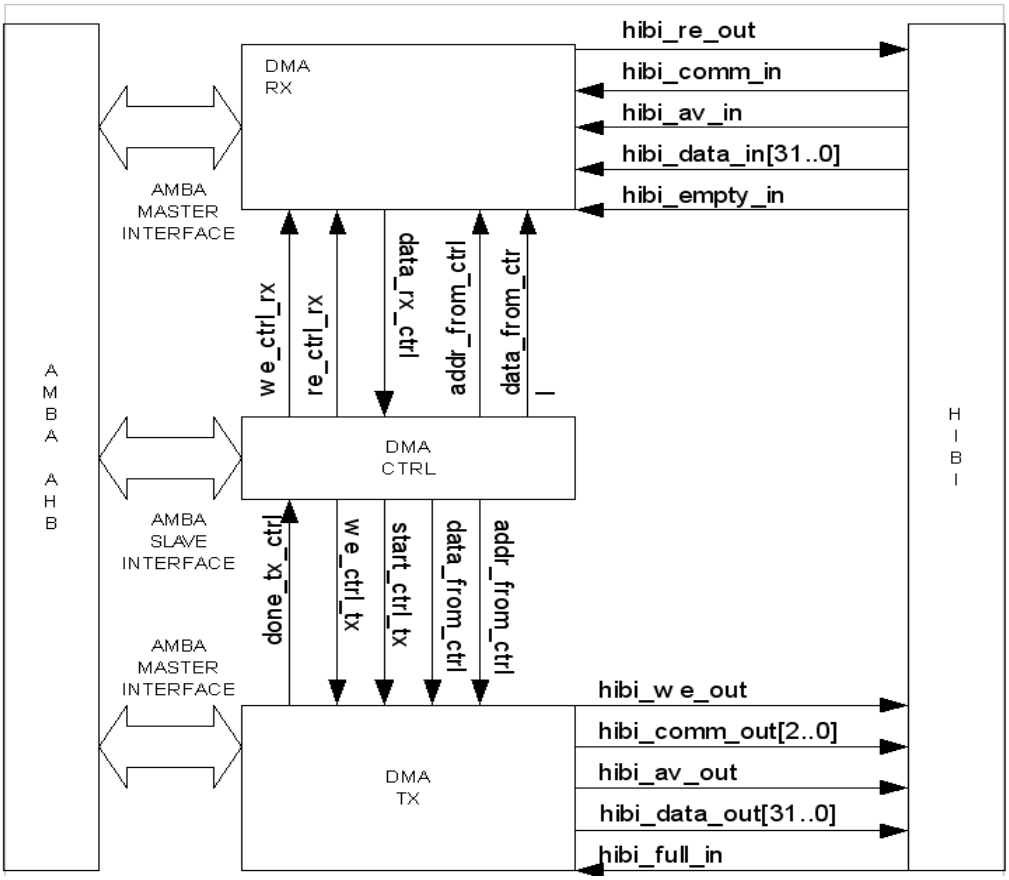


Figure 4. Block diagram of AMBA-to-HIBI interface

DMA_CTRL

This block is connected to AMBA AHB-bus with slave interface. Its interface ports are described in table 1. Dma_ctrl functions as a configuration unit and the processor can read and write the tx- and rx-channels' configuration registers through this block. Dma_ctrl includes one state-machine which is depicted in figure 5.

Port	Direction	Function
clk	in	clock
rst_n	in	reset
amba_slv_in	in	AMBA slave interface inputs
amba_slv_out	out	AMBA slave inteface outputs
rx_data_in	in	Data from dma_rx
tx_done_in	in	Tx-done from dma_tx
rx_re_out	out	Read enable to dma_rx
rx_we_out	out	Write enable to dma_rx
tx_we_out	out	Write enable to dma_tx
rx_addr_out	out	Channel- and register address to dma_tx
tx_addr_out	out	Register address to dma_tx
chan_data_out	out	Write data to both dma_rx and dma_tx
tx_start_out	out	TX-start for dma_tx

Table 1. Dma_ctrl interface

The function of the block is following. The block waits for read/write requests from AMBA and propagates them to either dma_tx or dma_rx depending on the request address. If request was a read request, dma_tx/dma_rx block returns the value of requested register to dma_ctrl which in turn sends it to requester.

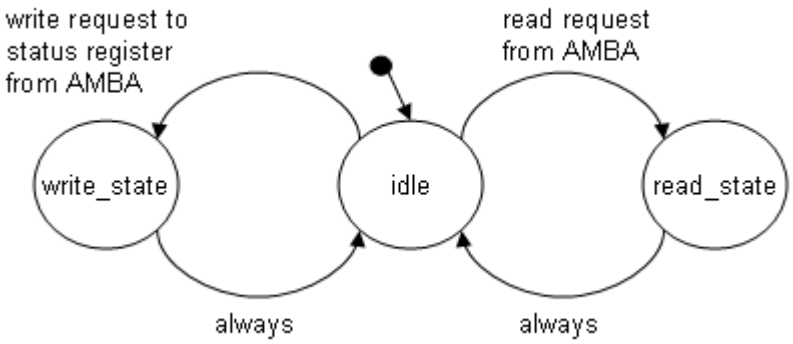


Figure 5. Dma_ctrl state-transitions

DMA_TX

Dma_tx implements one transmit channel. The interface of this block is described in table 2. The block contains tx-configuration registers (table 3) and two state-machines. One state machine handles the AMBA interface and the other the HIBI interface. The state-transitions of state machines are depicted in figures 6 and 7. The configuration registers are set according to write commands from dma_ctrl.

Port	Direction	Function
clk	in	clock
rst_n	in	reset
amba_mst_in	in	AMBA master interface inputs
amba_mst_out	out	AMBA master interface outputs
ctrl_data_in	in	write data from dma_ctrl
ctrl_addr_in	in	register address from dma_ctrl
ctrl_we_in	in	write enable from dma_ctrl
tx_done_out	out	tx-done to dma_ctrl
tx_start_in	in	tx-start from dma_ctrl
hibi_we_out	out	HIBI write enable
hibi_comm_out	out	HIBI command
hibi_full_in	in	HIBI full
hibi_data_out	out	write data to HIBI
hibi_av_out	out	HIBI address valid

Table 2. Dma_tx interface

The transmission begins when dma_ctrl sets tx_start_in signal high. Next the AMBA-state machine starts requesting tx-data from address stored in mem_addr-register. When data is available from AMBA, HIBI-state machine sends data to HIBI and mem_addr is incremented. When the configured amount of data is successfully sent to HIBI, transmission ends and tx-start signal goes to value '1'. The value of tx-start can be read from status register bit 16.

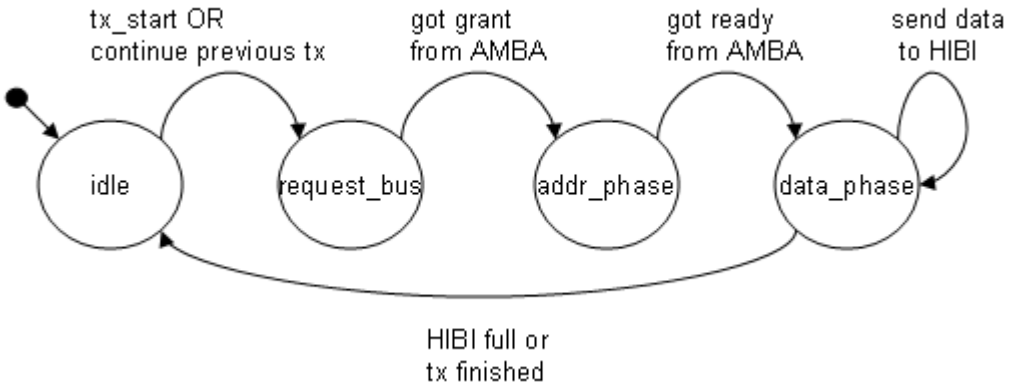


Figure 6. Dma_tx AMBA state machine

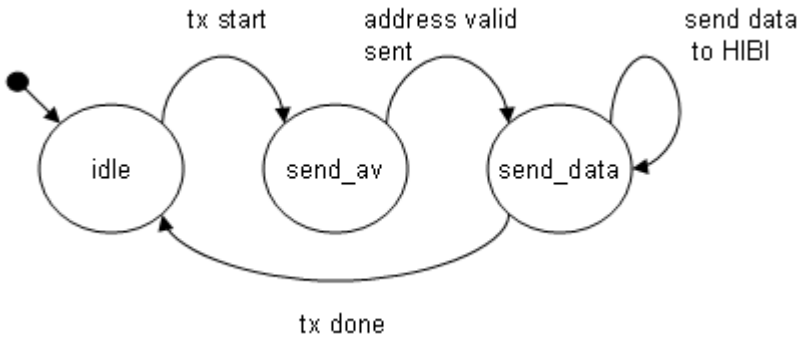


Figure 7. Dma_tx HIBI state machine

Address	Register	Purpose
0x20	mem_addr	Source address of data
0x24	amount	Amount to send
0x28	comm	HIBI command
0x2C	HIBI_addr	HIBI destination address

Table 2. Dma_tx registers

DMA_RX

This block implements the receive channels of AMBA-to-HIBI interface. The interface of this block is depicted in table 4. The block contains an arbitrary number of rx-channels (number is chosen by VHDL-generic) and their configuration registers (table 5). The block also contains one state-machine which handles the AMBA interface. State machine is depicted in figure 8.

Port	Direction	Function
clk	in	clock
rst_n	in	reset
amba_mst_in	in	AMBA master interface inputs
amba_mst_out	out	AMBA master interface outputs
ctrl_data_in	in	write data in from dma_ctrl to register
ctrl_addr_in	in	channel and register address from dma_ctrl
ctrl_we_in	in	write enable from dma_ctrl
ctrl_re_in	in	read enable from dma_ctrl
ctrl_data_out	out	register value output to dma_ctrl
hibi_re_out	out	HIBI read enable
hibi_empty_in	in	HIBI write enable
hibi_data_in	in	HIBI data input
hibi_av_in	in	HIBI address valid input

Table 4. Dma_rx interface

The configuration registers function the same way as in dma_tx except the request address includes the channel number in addition to register address. The bits 4..0 contain the register address and bits [4 + n_chan_bits_g .. 4] contain the channel number. N_chan_bits_g is a VHDL-generic, which tells the number of bits used to index rx-channels. It's value should be set to ceil(log2(number_of_channels)).

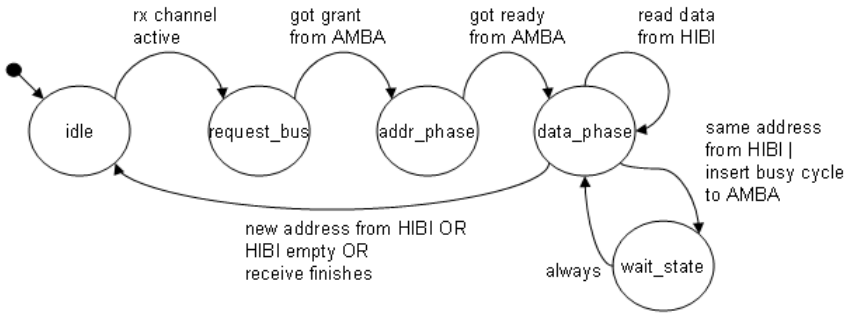


Figure 8. Dma_rx state machine

Receive channel is activated by writing '1' to a bit corresponding channel number in init_chan-register. After this, the activated channel waits until an address matching to receive_addr-register arrives from HIBI. When the address arrives, state-machine requests AMBA-bus and, after getting grant from AMBA, starts sending data from HIBI to AMBA. The destination address is in mem_addr-register. If AMBA is not ready to receive data, HIBI read is stalled until AMBA is ready. Data receiving continues until a new address comes from HIBI, HIBI gets empty or all expected data is received.

When receive finishes, the bit corresponding to channel number goes to '1' in IRQ source-register. If interrupts are allowed in config-register(table 6), an interrupt is given to AMBA. The interrupt is acknowledged by writing '1' to bit corresponding the channel number in IRQ source-register

Address	Register	Purpose
0x00	mem_addr	Destination start address for received data
0x04	receive_addr	HIBI-address where to receive data
0x08	amount	Amount of data to be received
0x0C	curr_addr	Current destination address for received data
0x10	config/status	Lower 16-bits are status register, upper configuration register(table 6)
0x14	init_chan	Initialize RX-channel (one-hot encoded, common for all RX-channels)
0x18	(RESERVED)	-
0x1C	IRQ source	RX-channel IRQ register(one-hot encoded, common for all RX-channels)

Table 5. Dma_rx registers

Bit	31..18	17	16	15..2	1	0
Purpose	(RESERVED)	RX-busy	TX-done	(RESERVED)	RX IRQ enable	TX-start

Table 6. Status/Config register

TIMETABLE

What has been done for this competition relates to a course done at the TUT. The work for that project was started at the end of 2006, and some 600 hours have been used in total for that TUT-project and this competition. Naturally these projects differed and not all work used on TUT-project was useful on this competition, and also some extra work was needed for the competition. Implementing AMBA-to-HIBI interface took roughly 200 hours, eCos 100 hours and multiprocessor system 100 hours.

DISTRIBUTION OF WORK

Hannu Penttinen worked on implementing the Leon3 Sparc processor and other hardware elements on the FPGA. Tapio Koskinen worked on eCos and test software. Aarno Tenhunen didn't participate in the project after all.

CONCLUSIONS

The goal of the project was to create an asymmetric multiprocessor system using SparcLeon3-processors and Altera Cyclone FPGA. However, due to the problems encountered, a symmetric multiprocessor (SMP) system was implemented. SMPs have also many applications and they are used in growing numbers nowadays. Our final system is an excellent platform for prototyping such systems on FPGA and it enables concurrent software and hardware designing on SMPs which can reduce design time considerably. Our system is also a good starting point for implementing a prototyping platform for asymmetric processor systems.

REFERENCES

[1] Erno Salminen, Vesa Lahtinen, Tero Kangas, Jouni Riihimäki, Kimmo Kuusilinna, Timo D. Hämäläinen, **"HIBI v.2 Communication Network for System-on-Chip"**, Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS IV), Samos, Greece, July 18-20, 2004, Vol.LNCS 3133 Computer Systems: Architectures, Modeling, and Simulation, A.D. Pimentel, S. Vassiliadis, (eds.), pp. 412-422, Springer-Verlag, Berlin, Germany.

[2] Ari Kulmala, **"Multiprocessor System with General-Purpose Interconnection Architecture on FPGA"**, Tampere, Finland, 2005, 73 pages, Tampere University of Technology

[3] Olli Lehtoranta, Erno Salminen, Ari Kulmala, Marko Hännikäinen, Timo D. Hämäläinen, **"A Parallel MPEG-4 Encoder for FPGA Based Multiprocessor SoC"**, 15th International Conference on Field Programmable Logic and Applications (FPL 2005), Tampere, Finland, August 24-26, 2005, pp. 380--385, Springer LNCS.