NTNU

Norwegian University of
Science and Technology

Embedded demonstrator for audio manipulation

Jarle Larsen

Master of Science in Electronics
Submission date: June 2010
Supervisor: Per Gunnar Kjeldsberg, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

In many situations, like school visits at NTNU, Forskningstorget and Elektronikk- & telekommunikasjonsdagen, it is desirable for Department of Electronics and Telecommunication to demonstrate good examples of electronic systems. Embedded systems are well suited as demonstrators since the combination of hardware and software gives both flexibility and wide possibilities for optimization.

In this task, the student will implement a specified system for demonstration of topics related to courses provided by the department. In order to make a good demonstration, necessary presentation material is also to be made, together with a plan on how to demonstrate the embedded system.

Assignment given: 15. January 2010
Supervisor: Per Gunnar Kjeldsberg, IET

Abstract

Demonstration of embedded systems is a good way to motivate and recruit students to a future career in electronics. For Department of Electronics and Telecommunication at the Norwegian University of Science and Technology (NTNU), it is thus desirable to have an embedded demonstrator that gives the pupils an insight in what is actually possible when studying electronics at the university, a system that the department may present at different occasions. A good embedded demonstrator provides an interesting presentation of one or more topics related to electronics, and should be presented together with relevant theory in order to provide a level of education to the user.

This report covers the implementation of an embedded demonstrator for audio manipulation on Altera's DE2 development and education board. The system is specified to demonstrate signal processing subjects like sampling and filtering through manipulation of analog audio signals. The main modules in the system are the Cyclone II 2C35 FPGA from Altera, running a Nios II soft-CPU, and a Wolfson WM8731 audio-codec. The specification of their operation is made with background in pedagogics theory in order to make the most interesting demonstration. To realize this specification, the system incorporates several design features for both activation and motivation of the user.

The audio manipulator provides possibilities for comparison between different sample rates and filter characteristics in real-time operation. This makes the system well suited for practical demonstration of signal processing theory. Due to the presentation of perceivable results, in addition to the implementation of a user interface for interaction, the implemented audio demonstrator is considered to be a well suited platform for demonstration of topics related to electronics.

Preface

I started my electronics career in lower secondary school about a decade ago, based on a rather genuine interest in hi-fi and audio systems. Since then, I have been following this path of electronics through a number of different courses at the upper secondary school, and now also at the university. This master thesis represents the finish line of my five years as a student at NTNU in Trondheim. These years have without doubt been the best and most exciting years of my life, and although the time has brought me through an unknown number topics in both the analog and digital domain, I am pleased to close this era with an in-depth study of an audio manipulator and its characteristics, the subjects that once started it all.

The goal with this project has been to implement a demonstrator for recruitment of future students to Department of Electronics and Telecommunication at NTNU. To design a system in order to motivate younger pupils to a future career in electronics seemed like a meaningful and important task. This, together with the possibility to work with and study the functionality of audio systems, was my main motivation-factor through out the process.

The last six months have been both exciting and challenging, providing a great deal of practical work through system implementation on Altera's DE2 board. It has been motivating to get some hands-on with electronics, using knowledge from five years at the university to implement a system based on my own specifications. Due to time constraints, the system never got any filter functionality in software as stated in the specification. However, due to the different system functionalities implemented in hardware, the final system was considered to present important signal processing subjects in a good way, also without software filters.

Several people contributed to the work presented in this report. I want to thank my supervisor Per Gunnar Kjeldsberg at the department for his advices and guidelines through the whole process. I am also thankful to my good friends Cato M. Jonassen and Kai André Venjum for their contribution and support, and to Cato for helping me with LaTeX related problems and the implementation of an LCD-display in the system. I would also like to thank Ingunn Amdal at the department for her contribution to theory regarding listening tests, and Kai André, Eivind Tjelde, Kjell Tutvedt and Ingrid Tøgersen for their time and effort as a test panel during an informal listening test performed on the system. Ingulf Helland should also be mentioned for his time and help on practical issues related to Altera's DE2 board.

Jarle Larsen
NTNU, Trondheim
June 2010

Contents

Definitions

ADC	:	Analog-to-Digital Converter - Converts analog signals to the digital domain
ASIC	:	Application-Specific Integrated Circuit - A hardware circuit of the application in silicone
BSP	:	Board Support Packages - A Nios II BSP project is a specialized library containing system-specific support code
CPU	:	Central Processing Unit - Primary element for performing operations in a computer
DAC	:	Digital-to-Analog Converter - Converts digital signals to the analog domain
DSP	:	Digital Signal Processor - Processor specialized for signal processing tasks
FDATOOL	:	Filter Design and Analysis Tool - User interface in MATLAB for specification of digital filters and generation of filter coefficients
FIFO	:	First-In First-Out - Refers to a way of queuing and organizing data
FIR	:	Finite Impulse Repsonse - Digital filter classification
FPGA	:	Field-Programmable Gate Array - Re-programmable logic device for implementation of hardware
HAL	:	Hardware Abstraction Layer - A software abstraction layer between the physical hardware and the software on a computer

HDL	:	Hardware Descriptive Language - Language for description of hardware
I^2C	:	Inter-Integrated Circuit - Communication standard with a two-wired, bidirectional bus
IIR	:	Infinte Impulse Repsonse - Digital filter classification
LE	:	Logic Element - Logic building block in an FPGA
LUT	:	Look-up Table - Function generator that implements logical functions in an LE
MATLAB	:	Matrix Laboratory - An advanced numerical programming environment with its own language based on C
OSR	:	Oversampling Rate - Defines the oversampling rate in an ADC or DAC
PLL	:	Phase-Locked Loop - Generates an output signal with equal or different phase and frequency based on a reference signal
RAM	:	Random Access Memory - Volatile computer memory. Looses information when the power is turned off
RISC	:	Reduced Instruction-Set Computer - A processor architecture with a reduced number of instructions
RMS	:	Root Mean Square - Defines the effective voltage or current in an AC wave
ROM	:	Read Only Memory - Computer memory that can not be modified
SQNR	:	Signal-to-Quantization Noise Ratio - The ratio between the preferred signal and the quantization noise in an ADC

Chapter 1

Introduction

In many situations, like education exhibitions and school visits at the NTNU, it is important for the electronics department to promote for its courses to get the attention from future students. To achieve this, it is desirable to have a good demonstration of what is actually possible when you study electronics at NTNU. A demonstration might consists of just talking to pupils and presenting posters of relevant topics, but in this project it is desirable to make a demonstrator that presents concrete examples of practical use of electronics.

The main goal with a demonstrator is to get people's attention, and to demonstrate a system that makes them interested in the presented topics. The demonstration should thus include an interesting application, and also provide some education on the subjects that the demonstrator presents.

This report describes the implementation of an embedded demonstrator based on the specification specified in [29]. This specification is made with background in theory related to pedagogics, and specifies an audio system for demonstration of subjects related to digital signal processing and embedded system design.

In order to present the implementation of the specified system, this report covers the following main topics:

- Presentation of the system specification used for implementation.
- Discussion of pedagogical aspects related to demonstrator implementation.
- Discussions related to the implementation of the system on Altera's DE2.
- Description of how the system should be presented to obtain a good demonstration.

1.1 Preliminary Work

The motivation for the mentioned specification was to specify a system, that through demonstration of subjects related to cources provided by Department of Electronics and Telecommunication at NTNU, could be used to recruit future students to the university. In order to

achieve this, the specification presents a system for implementation on Altera's DE2 development and education board, called an *embedded audio demonstrator*. This system performs domain conversion on the audio signal between the analog and digital domain, and filters the digital representation in both hardware- and software-implemented FIR filters as seen in figure 1.1.

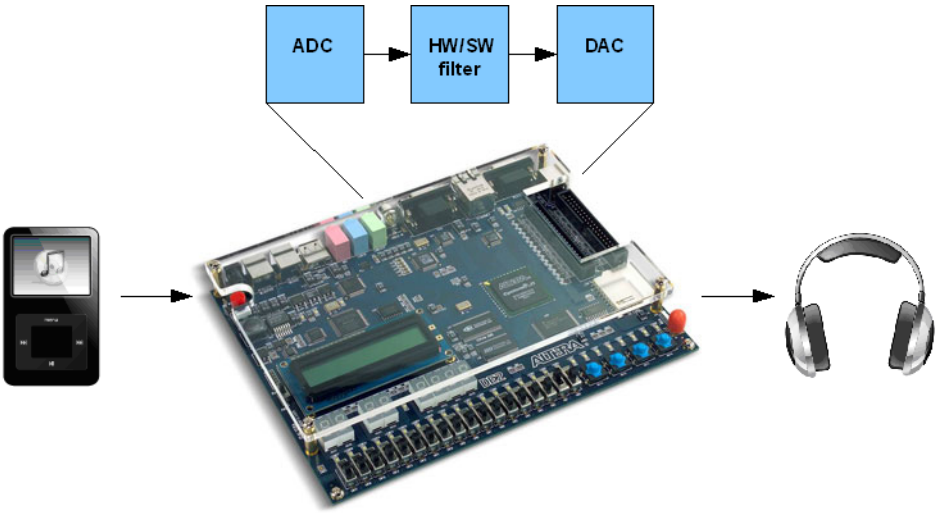

Figure 1.1: Embedded audio demonstrator - basic block diagram.

To make a good demonstration of sampling in both analog-to-digital- and digital-to-analog converters, ADCs and DACs, the system is specified with an adjustable sample rate. This makes it possible to present perceivable results of topics related to sampling to the listeners. Here they will get a real-time demonstration of how sampling at different rates, both above and below the Nyquist rate for audieble signals, affects the perceived sound quality. In addition, it is possible to present the relation between sample frequency and audio bit rate, and demonstrate how an increase or decrease in bit rate affects the perception of the sound.

For signal manipulation in the digital domain, the embedded audio demonstrator is specified with a high-pass and a low-pass filter for implementation in both hardware and software. Filtering is a well known application, and might be common to most people through the use of sound equalizers in home stereo systems and TVs. Digital filtering demonstrates the implementation of this operation in the digital domain, and provides knowledge to the listener about the effect of high-pass and low-pass filtering. In addition, since the system is specified with FIR filters implemented in both hardware and software, an interesting comparison between the filter performance in these two domains could be included as a part of the demonstration.

The specified system is considered to be an interesting and usefull demonstrator for demonstration of topics related to digital signal processing. However, the presentation of such an embedded system should not be done without considering some pedagogic aspects related to demonstration. The next two sections highlights some of the advantages related to the use of embedded systems as demonstrators, and presents a level of pedagogics that should be considered in order to provide a good demonstration of such embedded systems.

1.2 Embedded Systems

Embedded systems can be found everywhere - in consumer electronics, home appliances, business equipments, automobiles and so on [16]. The design is based on computer technology, but focuses on specific applications instead of general processing as central processing units (CPUs). These systems are well suited as demonstrators because the combination of hardware and software gives both flexibility and wide possibilities for optimization. This design methodology, called hardware/software co-design, means meeting system level objectives by exploiting the synergism of hardware and software through their concurrent design [13]. While hardware circuits like application-specific integrated circuit (ASICs) are configured at manufacturing time, the introduction of field-programmable gate arrays (FPGAs) increases the flexibility of an embedded system through its ability to handle configuration after the manufacturing process. This reconfigurability makes the system very versatile, and thus well suited for implementation of embedded demonstrators.

The flexibility of the FPGA enables the implementation of a soft-core embedded processor in the reconfigurable logic. A soft-core processor is a hardware description language (HDL) model of a specific processor that can be customized for a given application and synthesized for an ASIC or FPGA target [26]. The use of soft-core processors holds many advantages for the designer of an embedded system. First, soft-core processors are flexible and can be customized for a specific application with relative ease. Second, since soft-core processors are technology-independent and can be synthesized for any given target ASIC or FPGA technology, they are more immune to becoming old-fashioned when compared with circuit- or logic level descriptions of a processor [26].

For embedded demonstrators, the use of soft-core processors provides the designer with the opportunity to allocate the different modules in the system to either hardware or software, based on an understanding of which of the two domains that provides the best performance for the actual module. In addition, the versatility of the FPGA makes it possible to replace multiple components with one single chip. The reduction in the number of components will again reduce the board size, both of which will save development time and costs. This will also make the demonstrator more flexible, due to the increased portability.

The use of embedded processors has many advantages as presented above. However, embedding a processor inside an FPGA is not without disadvantages [20]. First of all, and unlike an of-the-shelf processor, the hardware platform for the embedded processor must be designed. Thus, the embedded designer becomes the hardware processor system designer when an FPGA solution is selected. Secondly, the device cost must be considered. If a standard, of-the-shelf processor can do the job, it will be more sensible and less expensive to use this than to implement an FPGA embedded processor. However, if there already exists a large FPGA in the system with unused gates, it might be economical, both in terms of development time and area cost, to implement a soft CPU on this one rather than using a separate processor [20].

1.2.1 Soft vs. Hard Processors

Originally, the embedded processor cores are soft core IPs, for example the Xilinx 32-bit MicroBlaze and Altera's 32-bit Nios II [16]. However, both Xilinx and Altera produce FPGA families that embed a physical processor core into the FPGA silicon [20]. A processor built from dedicated silicon is referred to as a hard processor, in contrast to soft processors that runs on the FPGA's general logic. For comparision, Table 1.1 and 1.2 presents the difference in performance for hard-core and soft-core processors on Altera and Xilinx FPGAs respectively.

Table 1.1: Altera Embedded Processors and Performance, [20].

Processor	Processor Type	Device Family	Speed(MHz)	DMIPSs
ARM922T	Hard	Excalibur	200	210
Nios II	Soft	Cyclone II	Not reported	100

Table 1.2: Xilinx Embedded Processors and Performance, [20].

Processor	Processor Type	Device Family	Speed(MHz)	DMIPSs
PowerPC 405	Hard	Virtex-4	450	680
Microblaze	Soft	Virtex-II Pro	150	123

As shown in these tables, hard processors have a clear performance advantage over soft-core processors. However, the key issue is whether a need for a hard structure appears often enough in the set of target applications, and, in the case where the architect seeks enhanced speed, if that structure appears on the critical path of designs when implemented as part of the soft fabric [35]. In addition, some of the performance differences between hard-core and soft-core processors can be made up due to the reconfigurability of the soft core processors, allowing a trade-off between performance and area by changing the architecture. Also, a programmable quantity of processors can be instantiated as needed, each tuned to the required area and performance specifications [35].

So, an alternative to add hard structures to an FPGA is to find ways to improve the performance of the soft logic fabric. If this can be done, it more easily makes all systems and applications both faster and cheaper, in addition to less power consuming [35].

1.3 Presentation of Embedded Systems

Demonstration of embedded systems is a good way to give the pupils or students who attends the presentation an insight in what is actually possible when you study electronics at NTNU. When presenting an embedded system, different topics related to pedagogics should be considered, in order to make the best and most interesting demonstration.

Learning is based on different pedagogical aspects that the teacher, or the person who demonstrates a subject, will have to consider. Motivation is one of them [24]. Motivation helps the pupils to understand the usefulness of the theory presented, and also to relate the presentation to something that they are familiar with. The person who demonstrates the audio demonstrator should thus motivate the pupils before the demonstration starts, in order to increase

the quality of the presentation. When the person or group is motivated, the best educational setting is achieved if the presentation contains a level of activation [23]. This should motivate an adjustment of the subjects presented so that people who attends the demonstration are able to take part in the presentation. In an embedded demonstrator, an example of activation is the implementation of a user interface that the user can operate, in addition to the use of multimedia content like audio and video in the demonstration.

In addition to motivation and activation of the pupils or students, the person who presents the demonstrator should also consider the age level of the group that is involved in the presentation. Differentiation is the process of providing different levels of education to different groups of people [23]. To divide people into groups based on their age is the most common example of differentiation in the school system today, and this assumes that people at the same age level are more or less able to work with the same subjects and challenges [24]. In the demonstration situation this demonstrator is intended for, the differentiation into groups is most likely done beforehand. Thus, the person who demonstrates the system will have to adjust the presentation according to the person or group that attends the demonstration. A group of high school students that study electronics may be both interested in and capable of understanding an in-depth description of both sample rates and filtering. Here, the person who demonstrates the system should be able to provide the group with a detailed description of how the system works. However, for younger persons, for instance a group of primary school pupils, the demonstration should not focus on the system level specifications, but rather on the aspects of the demonstration that are easy to perceive.

1.4 Structure of Report

In order to cover the presented topics, this report is organized in the following way: Chapter 2 presents theory on subjective quality measures related to sound. In Chapter 3, 4 and 5 the theory needed to understand the functionality of the system is described, including Inter-Integrated Circuit I^2C-communication, A/D- and D/A-conversion and digital filtering. Chapter 6 introduces the main modules on the Altera DE2 platform, while Chapter 7 describes the development softwares used for implementation of the demonstrator. Chapter 8 provides an in-depth description of the system implementation and discussions related to this implementation, followed by a description of how the implemented system should be demonstrated in Chapter 9. Chapter 10 presents the main conclusions related to the system implementation.

Chapter 2

Subjective Quality Measures

When designing an embedded system intended for manipulation of analog audio signals, subjective quality measures of the sound could be used to evaluate the performance of the design. This chapter presents some theory on the frequency range of the human ear, as well as an introduction to the term sound quality, and a presentation of how a subjective evaluation of an audio system may be performed.

2.1 Frequency Range of the Human Ear

The human hearing system is usually quoted as having an average frequency range of 20 - 20.000 Hz [21]. However, there can be a considerable variation between individuals. The frequency range changes as a part of the human ageing process, particularly in terms of the upper limit which tends to reduce. Healthy young children may have a full hearing range up to 20 kHz, but by the age of 20, the upper limit may have dropped to about 16 kHz. From this age, the frequency range continues to reduce gradually. The reduction in the upper frequency limit of the hearing range is accompanied by a reduction in hearing sensitivity at all frequencies, although this sensitivity loss is less present for lower frequencies [21].

2.2 What is Sound Quality?

It is possible to talk about sound quality in both physical or technical and perceptual terms [36]. In physical terms it generally relates to certain desirable measured characteristics of audio devices, transmission channels or signals. Bitrate on a digital audio stream and the frequency response of a speaker are two examples of measurable characteristics used to determine the quality of an audio system. In perceptual terms, the sound quality relates to what is heard and how it is judged by the human listeners. Listening tests are used to determine the perceptual quality, where the listeners evaluates the quality of the perceived sound signal based on a predefined scale. The scale consists of either words or numbers, and describes the sound quality in terms that are meaningful to the listeners, like "excellent", "good", "poor" or "bad". In an ideal world, the technical and the perceptual terms could be related or mapped

directly to each other. However, there may be aspects of sound quality that can be perceived, even though they can not be measured, and some that can be measured and not perceived [36].

2.3 Evaluation of Sound Quality

To be able to decide the perceptual quality of an audio system, a formal listening test could be employed [36]. A number of different tests exist, and common to nearly all of them is the need to define exactly what is to be evaluated, whether it is an overall quality judgement supposed to include all aspects of sound quality, or if one or more specific attributes of the sound quality, like brightness or spaciousness, are to be tested. It is also very common to employ a reference of some sort. This might be a quality level to which others are compared, since human listeners tend to be quite poor at judging sound quality when they have nothing to compare with. When a reference quality is defined, the judgement can be quite reliable [36].

An example of a scenario where subjective testing of the audio quality could be used, is to evaluate whether the sound from a particular coder at a certain bitrate is perceptually transparent [19]. One way to perform this evaluation is through a forced-choice discrimination test. Here the listeners are presented with the reference signal and a lower bitrate representation in a random order. Then they will have to explain in which order the signals where presented. The idea is that if the listeners cannot distinguish the difference between a sound signal from an accepted quality standard such as a CD, compared to the same audio signal at a lower bitrate, then the two signals are perceptually equivalent, and the lower bitrate signal is defined as perceptually transparent [19].

Chapter 3

I^2C Communication

Many solutions for communication between modules in an embedded system exists. In this chapter a serial interface for peripheral connection, namely the Inter-Integrated Circuit (I^2C) communication standard, will be presented, due to its presence as a communication protocol for register configuration in the audio-codec on the DE2 board.

The I^2C bus is a cheap, but effective network used in small-scale embedded systems to interconnect peripheral devices [12]. The two-wired multi-master bus is bidirectional, low-speed and synchronous to a common clock. The two wires are named SDIN and SCLK, as seen in Figure 3.1, and are connected to a positive supply via a pull-up resistor. This sets both wires to logic high when not in use. Each device connected to the I^2C bus has a unique address and may work as either a transmitter (bus master), a receiver (bus slave) or both. The transmission begins with the address bits, followed by the data. The address byte consists of seven address bits and one direction bit. If the direction bit is *0*, the master will send data to a slave. If instead the direction bit is *1*, the master requests data from the slave.

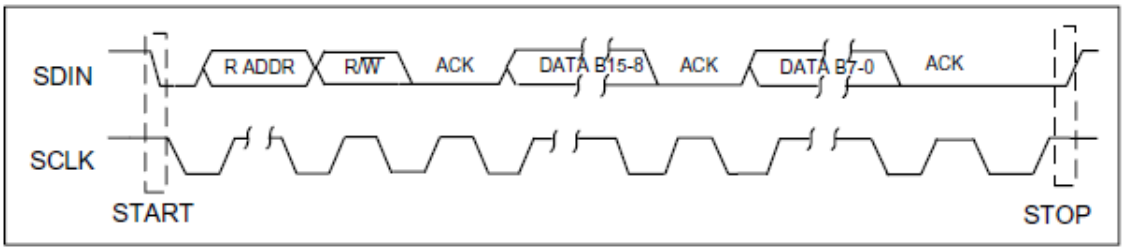

Figure 3.1: I^2C communication, [41].

A device using the I^2C bus to communicate drives the lines low or leaves them high as appropriate [12]. When idle, as seen in Figure 3.1, both SDIN and SCLK are high. A transmission starts with SDIN going low, followed by SCLK. This is a signal to the receivers on the bus that a packet is on its way. While SCLK is low, SDIN puts the first valid data bit on the line. The data bit is sampled on the rising edge of SCLK and must remain valid until SCLK goes low again. Then SDIN puts another bit on the line to be sampled by SCLK. A data transaction ends and the communication enters "stop condition" when SCLK is set to high followed by SDIN. Data is transferred with the most significant bit first, and if the

receiver is unable to receive more bytes at a given moment, it can abort the transmission by holding SCLK low. This forces the transmitter to wait for SCLK to be released.

When a byte is transmitted, the receiver will have to acknowledge the transmission. This happens when the transmitter releases the SDIN line, and then generates an additional clock pulse on SCLK. This tells the receiver to acknowledge the byte by pulling SDIN low. If the receiver fails to do this, the transmitter must start an error handling operation [12].

The process of error handling is described in the I^2C-standard [32]. If the receiver fails to acknowledge the address- or data byte, it will leave the SDIN line high, telling the transmitter that the data transfer failed. The transmitter will then generate either a stop condition to abort the transfer, or repeat the start condition to initialize a new transmission. In addition, if the receiver is unable to receive another byte from the transmitter due to other operations like internal interrupts, it can hold SCLK low and force the transmitter to wait with the next transmission. The transfer continues when the receiver is ready and thus releases the SCLK line [32].

Chapter 4

Domain Conversion

Most signals of practical interest, such as speech, biological signals, seismic signals, radar signals, and various communication signals used for audio and video, are analog [33]. In order for the audio demonstrator to process analog signals in the digital domain, it is first necessary to convert them into digital form, that is, to convert them to a sequence of numbers with a finite precision. This procedure is called A/D-conversion, and the process is performed by an analog-to-digital converter (ADC).

After manipulation in the digital domain, it desirable to convert the processed digital audio signals back to the analog domain, thus perform a D/A-conversion. In this chapter, topics related to conversion between these two domains are presented. Some theory on sample rates and quantization will be given, and also an introduction to the sigma-delta ADC and the Sigma-Delta digital-to-analog converter (DAC).

4.1 A/D Conversion

The process of converting an analog signal to the digital domain is known as sampling and quantization [12]. Sample rate is expressed as samples per second, and defines the frequency at which the analog signal is converted to a digital code. The resolution of an ADC, expressed in bits, determines the accuracy of each sample. As an example, an 8 bit ADC will be able to quantize the analog input signal to $2^8 = 256$ discrete values. More bits per sample gives a more accurate signal representation, but this will again make the ADC more expensive in terms of area use and power consumption. The designer must thus decide how many bits that is necessary for his or her application, and a high resolution ADC might not always be required.

Bit rate, expressed as thousands of bits per second, or kbps., is often used as a quality measure for digital audio systems [15]. To calculate the bit-rate of the sampled audio signal, the sample rate of the ADC is multiplied by the ADC resolution, as seen in Equation (4.1).

$$Bitrate = Sample\,rate \cdot resolution. \tag{4.1}$$

Lower bit rates results in less required storage space for the audio data, but at the same time a poorer sound quality. For higher bit rates the sound quality is increased, but this again increases the storage space needed for the digital audio representation. Bit rate is thus a trade-off between system complexity and sound quality [15].

Different approaches to the conversion between the analog and digital domain exists, and a popular A/D-converter is the oversampled Sigma-Delta ADC. Oversampling means using a sample rate which is greater, often substantially greater, than the Nyquist rate [38]. The Nyquist rate is defined as two times the maximum signal frequency. A measure of this oversampling is the oversampling ratio, OSR, defined in Equation (4.2), [11].

$$OSR = \frac{F_s}{F_N} \tag{4.2}$$

In this equation, F_s is the sample rate and F_N is the Nyquist rate for the input signal. Usually, the value of OSR is taken to be a power of 2. If the OSR is between 2 and 16, it is characterized as mild oversampling, whereas heavy oversampling occurs if the OSR is between 16 and 256.

Whether the sampling is done at or above the Nyquist frequency of the input signal, it is possible to recover the original analog signal exactly. Sampling below the Nyquist rate gives an overlap in the frequency domain called aliasing [33], seen in Figure 4.1. This occurs because the frequency spectrum is modulated around multiples of the sampling frequency [25], thus creating an overlap in the frequency domain if the sample rate is too low. Aliasing leads to a degraded representation of the analog signal when the digital representation is converted in a DAC. Sampling at rates high above the Nyquist rate is thus not required to obtain a given signal quality, but sampling at two times the signal frequency puts extreme high demands on component accuracy when a converter is implemented. The use of sample rates high above the Nyquist rate allows the anti-aliasing and reconstruction filters to be constructed with much more gentle cut-off slopes, as seen in Figure 4.2. This makes it possible to use components without very close tolerances, making the component cost less expensive [33].

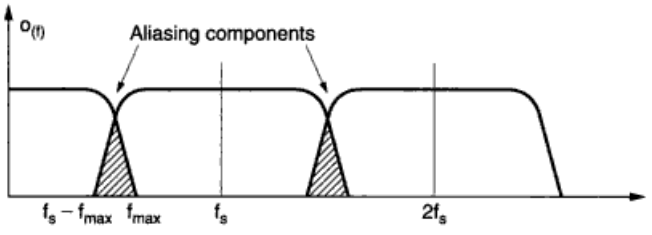

Figure 4.1: Aliasing due to undersampling, [25].

Another advantage with oversampling is the fact that the quantization noise energy is spread over a much wider frequency range, thus reducing the level of noise in the frequency band of interest [22]. Equation (4.3) describes the theoretical maximum signal-to-quantization noise ratio (SQNR) for a linear ADC, expressed in decibels (dB).

$$SQNR = 6.02B + 1.7dB \tag{4.3}$$

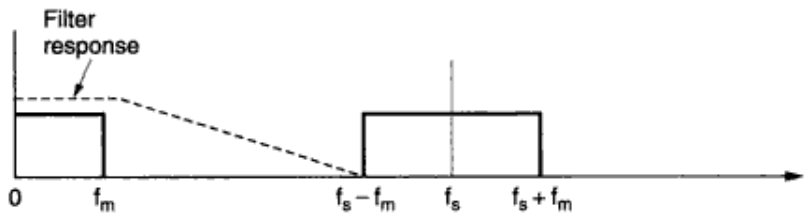

Figure 4.2: Oversampling gives simpler aliasing filter characteristics, [25].

In this equation, B states the number of bits per sample. The SQNR is thus only dependent on the resolution of the sampled signal, so an increase in the sample rate with a fixed resolution will not change the SQNR, only spread the noise energy over a wider frequency spectrum.

After the input signal is sampled and quantized to a finite number of discrete values, a decimator is used to convert the oversampled output words of the quantizer at a sample rate of F_s to a data word of lower rate [17]. If the oversampled input signal is reduced by a downsampling factor D, without any other operation, the result will be an aliased version of this signal [33]. Aliasing occurs because the frequency spectrum is repeated for every $f = \frac{f_s}{2D}$. It is thus possible to get an overlap between the repetitive frequency spectrums if the signal is downsampled too much. To avoid this effect, the bandwidth of the signal is reduced to $f = \frac{f_s}{2D}$ by a digital filter before the decimation occurs. The signal is then, without the risk of aliasing, downsampled to the desired sample frequency using decimation. The downsampled representation from the decimator defines the output signal from the ADC. It should be noted that the decimation process does not introduce any loss of information, since the digital filter removed all components that could be aliased into the frequency band of interest [17].

4.2 D/A Conversion

D/A-conversion is a process used to convert the digital signal into an analog form after it has been digitally processed, transmitted or stored [22]. In general, a DAC takes the digital samples at the input and returns an analog signal at the output. A popular approach is to perform D/A-conversion at a much higher rate than specified by the Nyquist theorem. The motivation for the use of oversampled Sigma-Delta DACs is similar to that for ADCs. An oversampling DAC uses interpolation and inserts zeros at a high rate between the already existing "low rate" samples in the data stream [37]. The oversampling makes it possible to implement analog filters with less strict constraints than necessary without interpolation. Another advantage is the fact that the quantization noise is spread over the total frequency band, and thus reduced due to the increased width of the spectrum [22]. This makes it possible to achieve high quality D/A conversion with a low resolution DAC.

Oversampling on its own is not enough to receive the desired DAC resolution [22]. Because of this, a practical oversampling DAC typically consist of four main parts: An oversampling digital filter, a noise shaper, for instance a Sigma-Delta modulator, a low resolution DAC and a simple analog anti-aliasing filter as seen in Figure 4.3. The oversampling filter is used to increase the sample rate and to reduce the aliasing components. The noise shaper serves to

push the quantization noise towards the high frequency end, thus making it easy to remove these components. After oversampling and noise shaping, the low resolution DAC converts the digital samples to a "staircase" shaped representation of the analog output signal due to a given hold time for each sample. At the end, a simple anti-aliasing filter is used to smooth out the "stairs" and recover the analog audio signal [22].

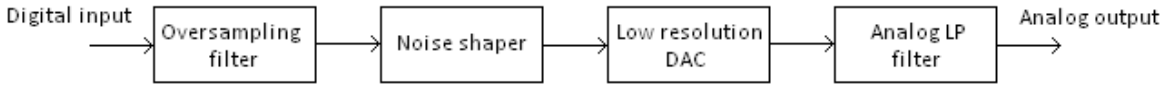

Figure 4.3: DAC blockdiagram, [22].

Chapter 5

Digital Filters

An electrical filter, analog or digital, is a device or network that separates waves on the basis of their frequency [10]. While the analog filters are defined by their respons in the frequency domain, digital filters operates on digital data in the time domain, performing numerical calculations on discrete samples to implement the filter's equation. In this chapter, theory regarding specification and implementation of digital filters are presented. There will also be given an introduction to how this operation is performed on an FPGA, and also on a CPU or digital signal processor (DSP). In addition, an introduction to fixed-point and floating-point representation of numbers will be provided.

5.1 Filter Specification

Digital filters are becoming more and more widespread, and are replacing analog filters in many systems today [39]. These digital filters are classified according to their impulse responses, and two categories exists: The finite impulse respons (FIR) filters, which are also known as non-recursive, and the infinite impulse respons (IIR) filters that are considered recursive due to their feedback functionality. There are some important differences between FIR and IIR filters that must be considered when designing a filter system. FIR filters may have an exact linear phase respons [33]. This implies that no phase distortion is introduced into the signal that is filtered due to a constant delay for each frequency component, making FIR filters well suited for audio applications. Phase distortion occurs on the other hand in IIR filters, due to a non-linear phase respons. FIR filters are also stable because of their non-feedback design. This stability can not be guaranteed in IIR filters due to the use of feedback. On the other hand, FIR filters require more coefficients for sharp cut-off slopes than the same filter type implemented as an IIR. An FIR filter will thus require more storage space and processing time for a given amplitude respons specification than the same specification in an IIR filter implementation.

In the design process of the filter, the designer must determine the cut-off frequency (or frequencies) and the stopband attenuation [39]. It must also be decided whether a low-pass, high-pass, band-pass or band-stop filter is required. The passband of a filter defines the range of frequencies that are allowed to pass through, with little or no change in signal level.

The stopband is defined as the range of signal frequencies that are reduced in amplitude by an amount specified in the design, and that effectively is prevented from passing. As an example, Figure 5.1 shows a filter characteristic with passband and stopband for an ideal low-pass filter. The frequency range between the passband and the stopband changes rapidly in signal amplitude due to attenuation performed by the filter [39]. There is a trade-off between the level of stopband attenuation and the width of the frequency range between the passband and the stopband. Also, high levels of attenuation generally require more filter taps, which again increases both time delay and filter complexity. This should thus be considered by the designer.

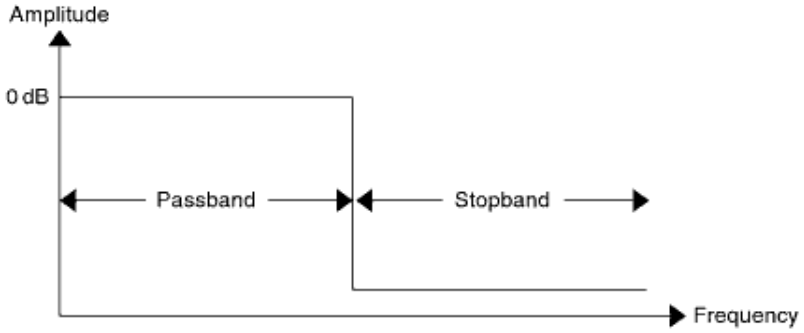

Figure 5.1: Ideal low-pass filter characteristic, [18].

The passband cut-off frequency is the passband edge where there is a 3 dB reduction in the signal amplitude. In a digital FIR filter, this cut-off frequency is directly proportional to the data sampling clock frequency. Using a single set of filter coefficients, described in more detail in Section 5.2, this cut-off frequency can thus be doubled by doubling the sampling clock frequency [39].

5.2 Filter Implementation

An FIR filter of length M with input signal x(n) and output signal y(n) is described in Equation (5.1). Here, h(k) is the set of filter coefficients [33].

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \tag{5.1}$$

To realize an FIR filter function, an array of delay elements, usually implemented as D flip-flops clocked by a master clock, is connected in series. The number of delay elements represents the filter order. The longer the delay, the closer the filter gets to the ideal frequency respons. Non-ideal filters, limited by the number of delays, may thus lead to rounding of the passband edges and ripple in the stopband. This motivates the use of many delay elements when designing a filter, but this again increases the filter's complexity.

After each delay element, a tap is taken and the value is multiplied by a filter coefficient. Since digital filters processes signals in the time domain [39], the frequency response of the desired

filter characteristic must be converted to a time domain impulse response representation using Inverse Fourier Transform (IFT) before implementation. The multiplied coefficient represents the impulse respons at a given moment in the time domain. It is thus the coefficients in the filter implementation that defines the filter characteristic. As an example, an ideal low-pass filter has a brick wall frequency respons, providing a flat passband with unity gain, and zero gain beyond cut-off. A time domain representation of this filter is a sinc(x) impulse respons [39], illustrated in Figure 5.2. In a sampled data system, such as a digital filter, this time domain respons is represented with discrete values. To obtain the filter coefficients that represents the characteristics of the filter, these discrete time domain samples must be multiplied with a selected window function. Windows are designed to truncate the time domain function to a certain number of taps, where the number of taps defines the length of the window. The simplest window function is the rectangular window seen in Figure 5.3. Here, the value of the window is unity over its whole length. For the ideal low-pass filter, the sinc(x) function is thus used as filter coefficients inside the window, while the sinc(x) samples outside are set to zero. By using the rectangular window, the first side lobe in the stop band of the frequency respons, as seen to the left in Figure 5.4, is limited to 13.2 dB, increasing with 6 dB per octave as the frequency increases [39].

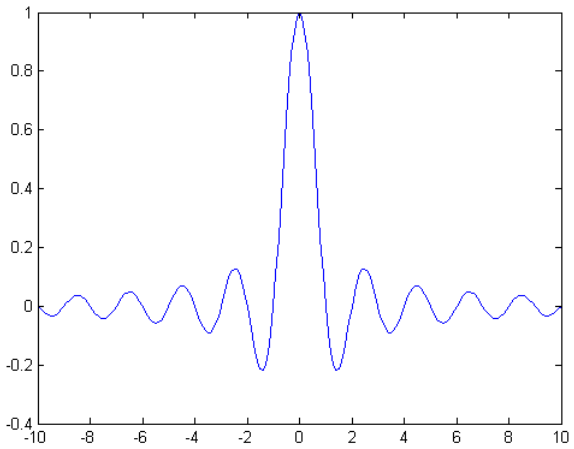

Figure 5.2: Sinc(x)-function.

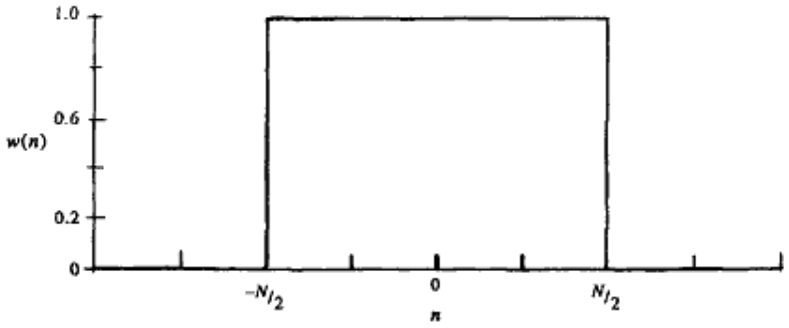

Figure 5.3: Rectangular window, [14].

If more attenuation is needed, other windows, like the Hamming window seen in Figure 5.5, could be used to achieve the given specification [39]. This window has a more complex representation, but provides a significant improvement in stop-band attenuation with up to 43 dB for the first side lobe in the frequency respons, seen to the right in figure 5.4. At higher frequencies this attenuation increases with 6 dB per octave. In the frequency domain, the amplitude of the main lobe is about twice as wide as that of the rectangular window [22] as seen to the right in Figure 5.4, but again, since the side lobes are smaller relative to the main lobe, the attenuation is more present. The result of this is that the Hamming window, compared with the rectangular window, will lead to a filter with a wider transition width, because of a wider main lobe, but also a higher attenuation of unwanted frequencies due to the smaller side lobes.

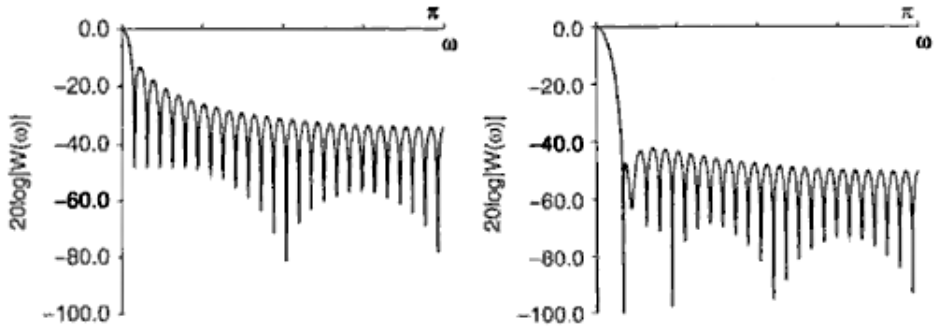

Figure 5.4: Frequency respons of rectangular window (left), and Hamming window (right), [27].

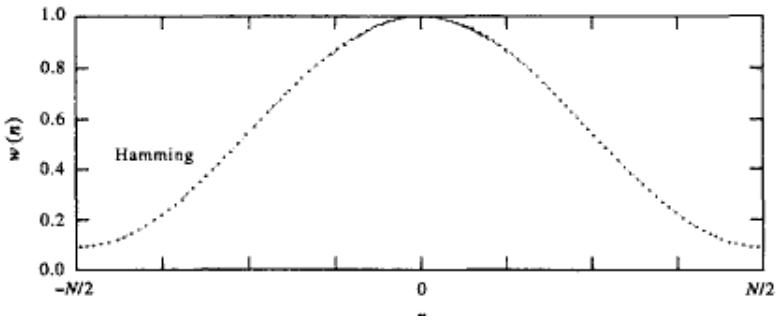

Figure 5.5: Hamming window, [14].

As earlier noted, the output from each delay element is multiplied by a filter coefficient. Since this multiplication is performed in the digital domain, results from binary multiplication must be taken into consideration. As an example, multiplying two 16-bit numbers produces a 32-bit result, so truncation may be required to remove the least significant bits [39]. This is necessary if the output resolution is equal to the input resolution of a system.

When the input signal has passed through the delay chain, and each output from the delay elements are multiplied by a given filter coefficient, the results of these multiplications are added together to form the output signal of the filter, thus realizing the filter's functionality.

5.3 Filtering in Hardware

FPGA architectures includes resources capable of more advanced, higher-performance signal processing operations with each new FPGA device family on the market [34], and is thus getting more and more suited for computation of complex mathematics and signal processing implementations such as FIR filters and Fast Fourier Transform (FFT). Figure 5.6 illustrates an example of a parallel implementation of an FIR filter within an FPGA. As seen, the function includes several multiplication and addition operations, also referred to as multiplication and accumulation (MAC) blocks [34]. Three popular implementations for the MAC operational groups within an FPGA are listed below:

- Both the multiplier and the accumulator may be implemented within the logic fabric of the FPGA, taking advantage of FPGA structures such as dedicated high-speed carry chains.
- The multiplier may be implemented in an optimized multiplier block, avoiding use of FPGA fabric for this operation. The accumulator is implemented in the logic fabric of the FPGA.
- Both the multiplier and the accumulator may be implemented within an advanced multiplier block requiring no use of FPGA logic.

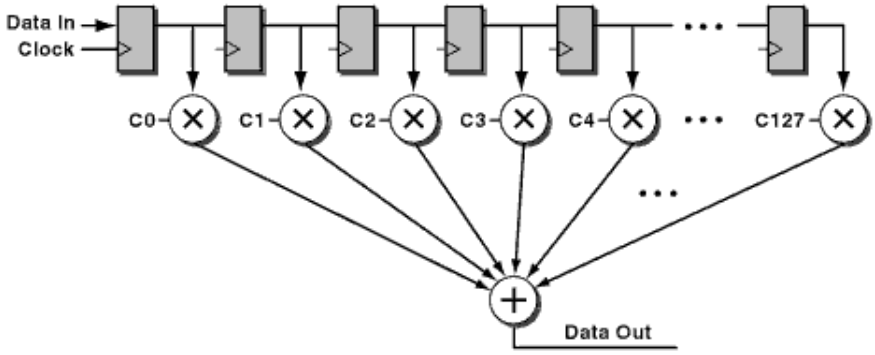

Figure 5.6: Parallel FPGA FIR filter implementation, [34].

The level of different digital signal processing blocks within an FPGA varies with the device families. As an example, the Altera Cyclone II family includes a set of embedded multipliers [3], dedicated for multiplication-intensive applications. The multiplication part of the functions may thus be implemented in this optimized multiplier block to prevent the use of the FPGA's logic for this operation [34]. The use of dedicated modules for specific operations should be considered in the design process due to their ability to increase the system performance.

5.4 Filtering in Software

The implementation of FIR filters in hardware, discussed in Section 5.3, may also be performed in software. Based on Equation (5.1) on page 16 it is possible to program and implement these

filters using a software language like C and execute the code on a processor. Most general purpose processors available today are based on the von Neumann concepts, where operations are performed sequentially [22]. When an instruction is processed in such a processor, the parts of the CPU not involved in the execution of the instructions waits in idle state until the control is passed on to them. Most digital signal processing algorithms, such as filtering and FFT, involves repetitive arithmetic operations such as multiply, add, memory access and heavy data flow through the CPU. The architecture of a general purpose microprocessor is thus not suited to perform this kind of activity. However, an advantage of using a CPU for digital signal processing is the fact that the C language is an efficient high level language, providing a compact code which reduces the need of space in memory. In addition, general purpose processors may also be optimized for digital signal processing through different techniques. The Harvard architecture uses the concept of parallelism and provides a full overlap in instruction fetch and execution [22]. Parallelism is also present in pipelining, where two or more operations are able to be executed at the same time. Both of these features helps to optimize a general purpose processor for digital signal processing.

As seen in Equation (5.1) on page 16, the numerical operations in an FIR filter realization are multiplication, addition and subtraction. To realize this function in software it is thus necessary to implement a number of basic components [22]. First of all, Random Access Memory (RAM) to store the present and past input samples, x(n) and x(n - k) respectively, is needed. It is also necessary to implement either RAM or Read Only Memory (ROM) for storing of the filter coefficients h(k). Finally the system needs an Arithmetic Logic Unit (ALU) to perform the mathematical operations.

5.5 Representation of Numbers

In the realization of FIR filters in hardware or in software on a general purpose computer, the accuracy with which filter coefficients can be specified is limited to the word length of the computer or the word length of the register provided to store the coefficients [33]. Since the coefficients used to implement a given filter are not exact due to quantization, the frequency respons of the system function will, in general, be different from the desired frequency respons.

In this section, fixed-point- and floating-point representation of numbers are presented, providing two different ways to represent filter coefficients in FIR filter implementation.

5.5.1 Fixed-point Representation of Numbers

Fixed-point arithmetic is most used in digital signal processing work because it leads to fast a implementation, but it is limited in the range of numbers that can be represented, and is sensitive to problems of overflow which may occur when the result of an addition exceeds the permissible number range [22]. To prevent overflow, the operands are scaled. Such scaling degrades the performance of the digital signal processing system, due to a reduced achievable signal-to-noise ratio.

The representation of numbers in a fixed point format is a generalization of the familiar decimal representation of a number as a string of digits with a decimal point [33]. In this

notation, the digits to the left of the decimal point represents the integer part of the number, while the digits on the right side of the decimal point represents the fractional part. The decimal point is not stored in any register, but is understood to be in a fixed position between the k most significant digits and the m least sigificant digits [28]. For this reason we call such representations fixed-point representations.

5.5.2 Floating-point Representation of Numbers

Floating point arithmetic is preferred where magnitudes of variables or system coefficients vary widely [22]. It allows a much wider dynamic range, and virtually eliminates overflow problems. However, floating-point arithmetic is often slower, although high speed digital signal processors (DSPs) with a build-in floating-point processor are becomming widely avaliable.

Floating-point is often used to represent very large or very small numbers [31]. When writing numbers of such magnitude, it is often convenient to use a notation called an e- or floating-point notation, seen in Equation (5.2).

$$X = M \cdot 2^E \tag{5.2}$$

Here, M is the mantissa or base value, while E is the exponent or scaling factor that moves the decimal point of the base value to the right or the left, thus making the decimal point "float" [31].

In comparing a fixed-point representation with a floating-point representation, each with the same number of total bits, it is apparent that the floating-point representation allows us to cover a larger dynamic range by varying the resolution across the range [33]. The resolution decreases with an increase in the size of successive numbers. In other words, the distance between two successive floating-point numbers increases as the numbers increase in size. It is this variable resolution that gives the floating point results a larger dynamic range. Alternatively, if it is desirable to cover the same dynamic range with both fixed-point and floating-point representations, the floating-point representation provides finer resolution for small numbers but coarser resolution for larger numbers. In contrast, the fixed-point representation provides a uniform resolution throughout the complete range of numbers [33].

Chapter 6

Altera DE2

In this chapter, the platform of the audio demonstrator, namely the Altera DE2 development board, is introduced. First, an overall introduction to the board's main features will be given, before a more detailed description of the Altera Cyclone II FPGA, the Nios II and the on-board audio-codec are presented. Further, an introduction to polling and interrupt is provided, in addition to an overall description of the WM8731 audio-codec and its on-board modules.

6.1 Features

The Altera DE2 development and education board is seen in Figure 6.1. This board includes an Altera Cyclone II 2C35 FPGA which is connected to all the important components on the board, and makes it possible to control every aspect of the board's operation [1]. The board offers a set of switches, 27 LEDs and a 7-segment display. There is also implemented SRAM, SDRAM, flash memory and a 16 x 2 LCD-display. The board is ready for audio- and video-application development, and includes both line-in, line-out and a microphone input, all connected to a 24-bits Wolfson WM8731 audio-codec. For picture, the DE2 includes a TV decoder connected to the board's video input and a 10-bits DAC for VGA conversion. The board also offers USB 2.0 connection, 10/100 Ethernet, infrared (Ir-DA)-port and memory slot with SD-card support.

6.2 Altera Cyclone II 2C35 FPGA

The Altera Cyclone II 2C35 is the FPGA implemented on the DE2 board. This FPGA features 33216 logical elements (LEs), 105 M4K embedded memory blocks, 35 multipliers and a total of 475 I/O pins [3]. A logical element, as seen in Figure 6.2, is a small unit of logic, providing an efficient way to implement logical functions. The LEs in Altera Cyclone II features a four-input, 16 bit "look-up table" (LUT) which is a function generator that may implement any logical function of four variables. The logical element also includes a programmable register that can be configured as either a D, T, JK or SR flip-flop. Each of these registers includes

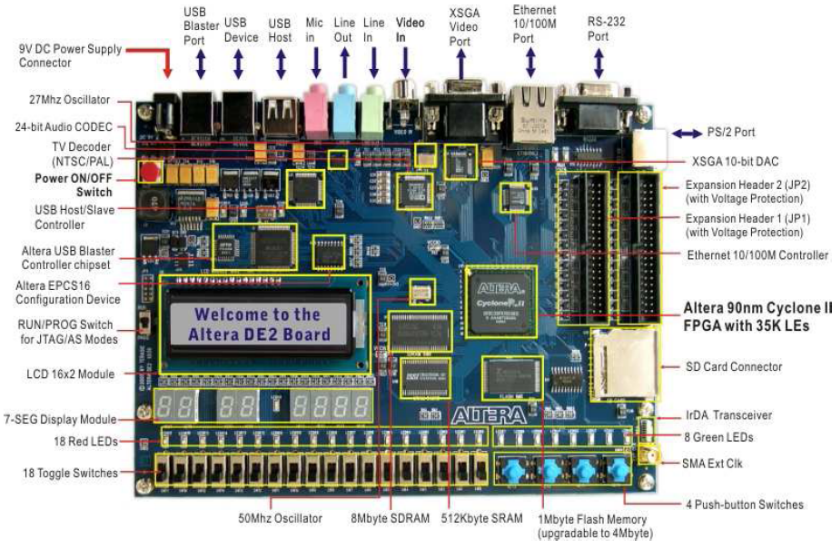

Figure 6.1: Altera DE2 development board, [1].

data-, clock-, clock enable- and clear inputs. If the LE is to implement a pure combinational function, this register will be bypassed, and the output of the LUT drives directly to the output of the logical element.

The embedded dual-port memory blocks of the device consist of 4Kbits each. These blocks can implement various types of memory depending on the system's requirements, including RAM, ROM and first-in-first-out (FIFO) buffers, and supports a maximum speed performance of 250 MHz. The device also includes a set of embedded multipliers, as seen in Figure 6.3, optimized for multiplier-intensive digital signal processing functions such as FFT, FIR filters and discrete cosine transform. Each embedded multiplier may operate as either one 18-bit multiplier or up to two independent 9-bit multipliers [3].

The Cyclone II LE can operate in either normal or arithmetic mode. The normal mode is suitable for general logic applications and combinational functions, while the arithmetic mode is ideal for implementing adders, counters, accumulators and comparators [3]. The operation mode for the logical elements should thus be chosen based on the functionality of the specified application.

6.3 Nios II

The Nios II is Alteras version of a configurable, soft-core, general purpose processor [6]. Configurable means that you can add or remove features to adjust the processor to your specified system and meet performance or price goals. Soft-core refers to the fact that the processor core is not produced as a final CPU-chip. Instead, it is written in a hardware description language like VHDL for implementation on a reconfigurable device. This makes it very versatile, and the Nios II can be implemented on any of Altera's FPGA families.

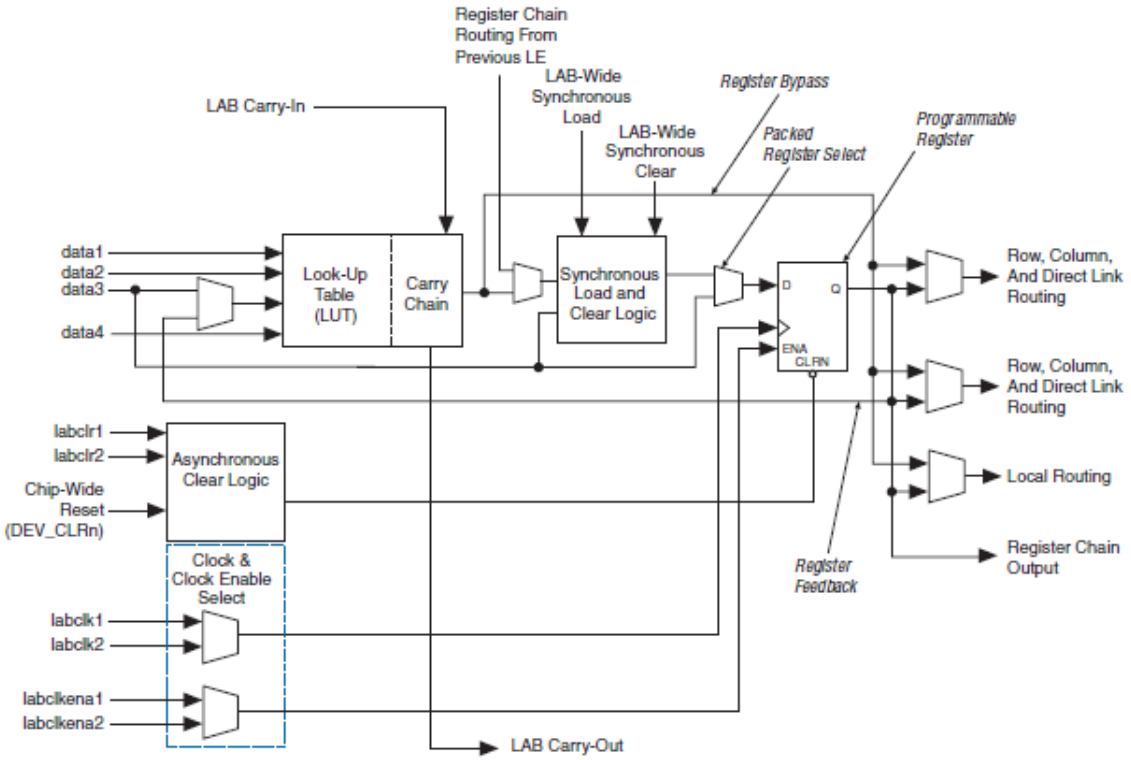

Figure 6.2: Cyclone II logical element, [3].

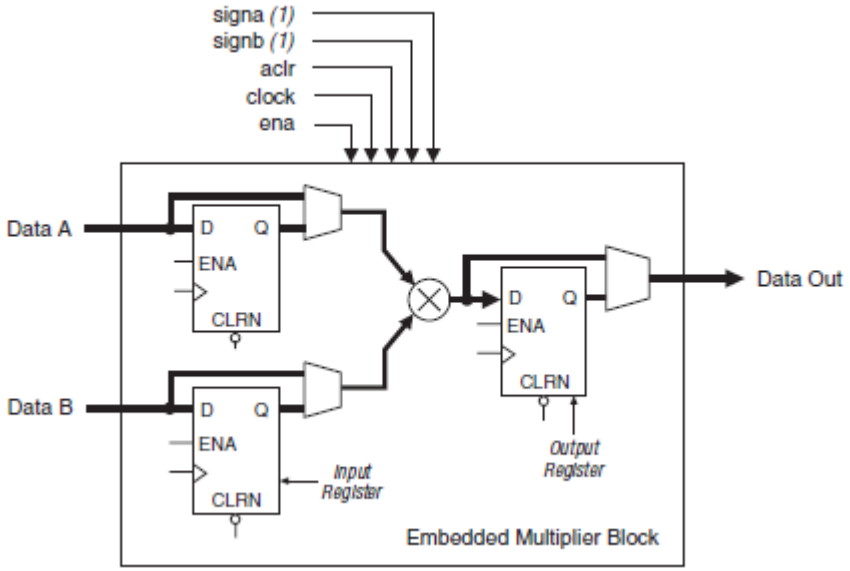

Figure 6.3: Multiplier block architecture, [3].

The Nios II core is available in three different versions, namely the "economy", the "standard" and the "fast" soft processor [6]. These differs in performance, area and the number of pipeline stages, as seen in Table 6.1. The CPU core clock frequency varies from 165 to 200 MHz between the three models. 1, 5 or 6 pipeline stages are available to support different design requirements, and the use of logical elements are less than 3000 for the largest implementation. The "standard" and the "fast" CPU version also offers hardware multipliers and division options, and is thus optimized for digital signal processing applications.

Table 6.1: Nios II Processor Cores, [6].

Feature	**Nios II/e**	**Nios II/s**	**Nios II/f**
DMIPS/MHz:	0.15	0.74	1.16
Max. DMIPS:	31	127	218
Max. f_{max}:	200 MHz	165 MHz	185 MHz
Area:	$<$ 700 LEs	$<$ 1400 LEs	$<$ 3000 LEs
Pipeline:	1 stage	5 stages	6 stages

6.3.1 Interrupt and Polling

Interrupt is a technique of diverting the processor from the execution of the current program so that it may deal with some event that has occurred [12]. Such an event may be an error from a peripheral, or simply that an I/O device has finished the last task it was given, and is now ready for another. Interrupts free the processor from having to continuously check the I/O devices to determine whether they require service or not. Instead, the processor may continue with other tasks. The I/O devices will notify it when they require any attention, through an interrupt signal on one of the processor's interrupt inputs.

Another technique used for this purpose is busy waiting or polling [12]. Here, the processor continuously checks the status of the device until the device requires attention. This wastes the processor's processing time, but is also the easiest technique to implement. In addition, for some time-critical applications, polling can reduce the time it takes for the processor to respond to a change of state in the peripheral [12].

6.4 Wolfson WM8731 Audio-Codec

The DE2 board provides high-quality 24-bit audio through the Wolfson WM8731 audio-codec [1]. Figure 6.4 shows a block diagram of this codec, with I/O-ports and internal modules. The chip supports microphone-in, line-in and line-out connections, with an adjustable sample rate from 8 to 96 kHz through oversampled Sigma-Delta ADCs and DACs, and supports I^2C communication for register configuration.

6.4.1 Line-in Circuit

To control the amplitude and frequency range of the analog input, a line-in circuit, seen in Figure 6.5, is implemented in the audio-codec [41]. This circuit provides a passive low

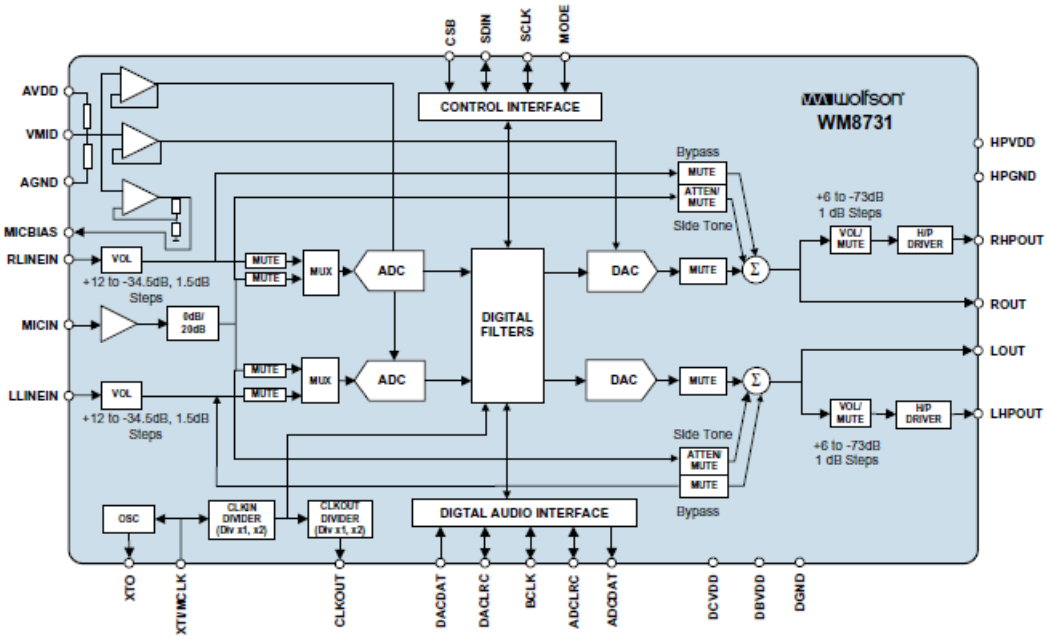

Figure 6.4: Wolfson WM8731 audio-codec, [41].

pass RC-filter to prevent high frequencies from aliasing into the audio band. In addition, a programmable volume controller for each channel is provided, adjusting the gain of the input signal between -34.5dB and +12dB before domain conversion to the digital domain.

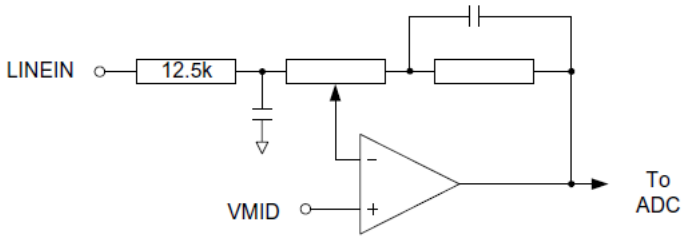

Figure 6.5: Line-in circuit, [41].

The ADC in the audio-codec supports analog input signals of maximum 1V RMS when VDD = 3.3V without causing distortion. Due to this, the DE2 board includes a 50/50 voltage divider at the input of the line-in circuit, dividing the analog input signals by a factor of 2. This provides the system with support for standard 2V RMS line-out signals from audio sources like CD- and MP3-players.

6.4.2 Digital Audio Interface

To be able to transmit and receive digital audio to and from other modules in the system, the audio-codec contains a digital audio interface, as seen in Figure 6.6. The WM8731 supports both slave mode and master mode for audio interface communication. In master mode the

audio-codec provides all signals for synchronization of audio data with the FPGA in this system, including BCLK, ADCLRC and DACLRC. In slave mode, the codec is depended on both master clock, BCLK, ADCLRC and DACLRC from an external module, in this case the FPGA.

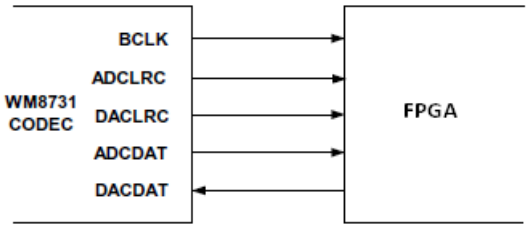

Figure 6.6: Communication through the audio-interface, [41].

BCLK synchronizes the data flow, where one bit is transmitted for each clock cycle as seen in Figure 6.7. LRCLK in this figure is used as a common description of both ADCLRC and DACLRC, because their relation to BCLK and the data lines are the same. LRCLK is an alignment clock that controls whether the left or the right channel is presented on the DACDAT or ADCDAT line, which defines the data lines to and from the audio-codec respectively.

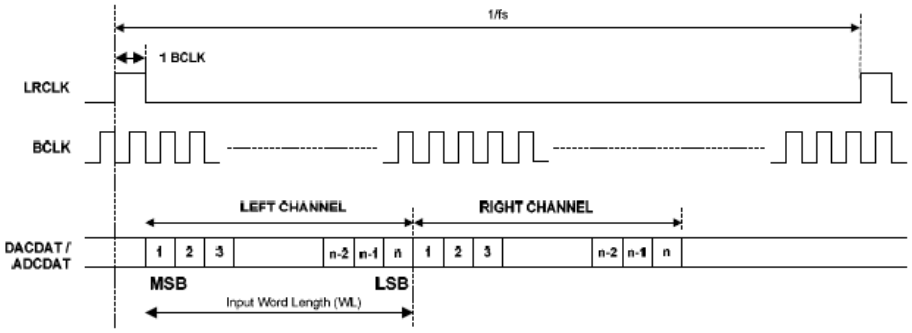

Figure 6.7: DSP mode audio interface synchronization, [41].

6.4.3 ADC and DAC

The variable sample rates in the audio demonstrator are generated on-chip from the master clock (MCLK), according to the division ratio seen in Equation (6.1), [40].

$$Divisionratio = \frac{MCLK}{Target\ sample\ rate} \tag{6.1}$$

These sample rates are generated in two different modes, namely the "Normal mode" and the "USB mode" [41], and the selected mode is valid for both the ADC and DAC. In "Normal mode", the user controls the sample rate by using an appropriate MCLK or crystal frequency together with the settings in the sample rate register. "Normal mode" provides six different

sample rates at 8, 32, 44.1, 48, 88.2 and 96 kHz, and requires different frequencies on MCLK to obtain all the different rates. Some of these are presented in Table 6.8. Here, the BOSR-bit defines the "Base Oversample Rate". This is the rate at which the digital signal processing in the audio-codec is carried out at, and the output sample rate will always be a sub-multiple of this, making it possible to change the sample rate by changing the decimation factor. The decimation factor is determined by Equation (6.1), and is set by the control bits BOSR and SR0-SR seen in Table 6.8.

SAMPLING RATE		MCLK FREQUENCY	SAMPLE RATE REGISTER SETTINGS					DIGITAL FILTER TYPE
ADC	DAC							
kHz	kHz	MHz	BOSR	SR3	SR2	SR1	SR0	
48	48	12.288	0 (256fs)	0	0	0	0	1
		18.432	1 (384fs)	0	0	0	0	
48	8	12.288	0 (256fs)	0	0	0	1	1
		18.432	1 (384fs)	0	0	0	1	
8	48	12.288	0 (256fs)	0	0	1	0	1
		18.432	1 (384fs)	0	0	1	0	
8	8	12.288	0 (256fs)	0	0	1	1	1
		18.432	1 (384fs)	0	0	1	1	
32	32	12.288	0 (256fs)	0	1	1	0	1
		18.432	1 (384fs)	0	1	1	0	
96	96	12.288	0 (128fs)	0	1	1	1	2
		18.432	1 (192fs)	0	1	1	1	

Figure 6.8: Normal mode sample rate table, [41].

The other operation mode, called "USB mode", provides the same selection of sample rates between 8 and 96 kHz as the "Normal mode". However, in this case, only one MCLK at 12 MHz is needed to obtain all the different rates as seen in Table 6.9, removing the need of more than one master clock or Phase Locked Loop (PLL) circuit for clock generation [41].

The length of the digital audio data is programmable at 16/20/24 or 32 bits [41]. Both the ADC and DAC are fixed at the same data length, and the ADC and DAC digital filters process data using 24 bits. If the ADC is programmed to output 16 or 20 bits of data, it strips the LSBs from the 24 bits input. If the ADC however is programmed to output 32 bits, then it packs the 24 LSBs with zeros. The same operation is performed in the DAC, where bits are added or removed in order to support 24 bits signal processing.

6.4.4 Headphone Amplifier

The WM8731 includes a stereo headphone amplifier [41], as seen in Figure 6.10. The output is designed specifically for driving 16 to 32 ohm headphones with maximum efficiency and low power consumption. The headphone amplifier includes a high quality volume level adjustment and an integrated mute function, and is configured through the configuration registers in the audio-codec.

SAMPLING RATE		MCLK FREQUENCY	SAMPLE RATE REGISTER SETTINGS					DIGITAL FILTER TYPE
ADC	DAC							
kHz	kHz	MHz	BOSR	SR3	SR2	SR1	SR0	
48	48	12.000	0	0	0	0	0	0
44.1 (Note 2)	44.1 (Note 2)	12.000	1	1	0	0	0	1
48	8	12.000	0	0	0	0	1	0
44.1 (Note 2)	8 (Note 1)	12.000	1	1	0	0	1	1
8	48	12.000	0	0	0	1	0	0
8 (Note 1)	44.1 (Note 2)	12.000	1	1	0	1	0	1
8	8	12.000	0	0	0	1	1	0
8 (Note 1)	8 (Note 1)	12.000	1	1	0	1	1	1
32	32	12.000	0	0	1	1	0	0
96	96	12.000	0	0	1	1	1	3
88.2 (Note 3)	88.2 (Note 3)	12.000	1	1	1	1	1	2

Figure 6.9: USB mode sample rate table, [41].

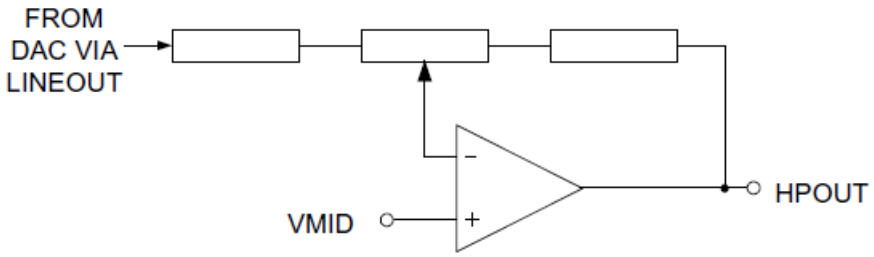

Figure 6.10: Headphone amplifier schematic, [41].

Chapter 7

Development Software

When designing an embedded system, one or more development tools may be used to increase the design speed and flexibility of the system design. This chapter presents a number of design tools provided by Altera, designed to help FPGA designers to create their systems in an effective way.

7.1 Quartus II and SOPC Builder

The Altera Quartus II is a design software that provides a complete, multiplatform design environment for designers to implement their specific system-on-a-programmable-chip (SOPC), and includes solutions for all phases of FPGA circuit design [5]. Quartus II enables compiling, timing analysis, RTL diagrams and target device configuration among other features, and supports Verilog, VHDL or Alteras own AHDL as hardware descriptive languages. A Block Editor is implemented to support the creation of symbols that represents the hardware descriptive design files and create circuit diagrams based on these blocks.

To help designers create systems based on the Nios II processor, SOPC Builder is included as a part of the Quartus II software [8]. The SOPC Builder automates the work of integrating hardware components in the design, and may also be used to create systems without the Nios II. Traditionally the designer manually writes HDL code to describe the modules in a system. With SOPC Builder one can specify the system components through a graphical interface, and the program automatically generates the interconnected logic. SOPC Builder generates HDL files for all modules in the system, including a top-level HDL file that connects the components together. This dramatically simplifies the designers work in creating a high-performance SOPC design [8].

7.2 MegaWizard Plug-In Manager

In addition to the SOPC builder, Altera also provides parameterizable megafunctions that are optimized for Altera device architectures [9]. Using megafunctions instead of coding your own logic saves valuable design time. Additionally, the Altera-provided megafunctions may offer

more efficient logic synthesis and device implementation. The MegaWizard Plug-In Manager provides a GUI to customize and parameterize megafunctions, and ensure that you set all megafunction parameters properly. When the designer has finished the parameterization, it is possible to select which files to be generated, and the MegaWizard automatically generates all the files needed to use the module in the design [9].

An example of an IP Core available through the MegaWizard Plug-In Manager is the FIR Compiler MegaCore function [4]. This core provides a fully integrated FIR filter function optimized for use with Altera FPGA devices, including a coefficient generator. The designer can specify the filter settings and coefficient options in the MegaWizard interface. When the sample rate, the cut-off frequency and the number of filter coefficients among other settings are specified, the MegaWizard creates a hardware FIR filter based on these specifications. In addition, the FIR Compiler enables the designer to load and use predefined coefficients, integers or floating-point number, from a file. If floating point coefficients are used, these coefficients will be quantized by the tool since only integer-coefficient filters can be generated with the FIR Compiler [30].

7.3 Nios II Embedded Design Suite

The Nios II Embedded Design Suite (EDS) provides a consistent software development environment that works for all Nios II processor systems [7]. With a PC, an Altera FPGA, and a JTAG download cable, you can write for, and communicate with, any Nios II processor system. The Nios II EDS includes many proprietary and open-source tools for creating Nios II programs, and automates the board support package (BSP) creation for Nios II processor-based systems, eliminating the need to spend time manually creating BSPs. A Nios II BSP project is a specialized library containing system-specific support code, and the BSP provides a C/C++ runtime environment, insulating the designer from the hardware in the embedded system.

Altera BSPs contain the Altera hardware abstraction layer (HAL) [7]. The HAL is a lightweight runtime environment that provides a simple device driver interface for programs to communicate with the underlying hardware. The HAL serves as a device driver package for Nios II processor systems, providing a consistent interface to the peripherals in the system.

Chapter 8

System Implementation and Discussion

The embedded audio demonstrator is based on the idea of demonstrating how the quality of an audio signal is reduced when the sample rates in data converters are set below the Nyquist rate, compared with sample rates above. In addition, to provide a more interesting demonstration of audio manipulation, two FIR filters are implemented to cover a wider range of signal processing subjects. This chapter describes the development process of the system, and discusses the different choices that were made during the implementation. In this case, the main goal with each choice was always to achieve the best demonstration of signal processing subjects. There were no other constraints to system performance beyond the limits of the devices, although some studies of the trade-off between resource use and the quality of the demonstration were made, in order to present some data on how the level of demonstration effects relates to the use of resources in the system.

8.1 Overall System Functionality

The system is implemented on Altera's DE2 development and education board, presented in Chapter 6. The choice of platform was mainly based on this board's availability at Department of Electronics and Telecommunication at NTNU, and an evaluation of other platforms would have been sensible if this board had not been available. However, due to the fact that this platform includes both an audio-codec and an FPGA, in addition to different I/O ports and an LCD-display, the DE2 board was considered to be a well suited solution for implementation of an embedded system for audio manipulation. The main system specifications is presented in Table 8.1.

A block diagram of the system is seen in Figure 8.1. Here, all the main modules in the audio demonstrator and their interconnection are presented, making it easier to understand the signal flow that will be presented. The system's main functionality is provided by the Wolfson WM8731 audio-codec, seen in more detail in Figure 6.4, and the Altera Cyclone II FPGA presented in Chapter 6.2. A user interface with buttons is implemented for user

Table 8.1: Main system specifications.

System platform:	Altera DE2 education and development board.
Main modules:	Altera Cyclone II FPGA and Wolfson WM8731 audio-codec.
Signal source:	Any audio source with analog output of max. 2V RMS.
Sample rates:	8, 32, 48 and 96 kHz in both ADC and DAC.
Digital filters:	High-pass and low-pass FIR-filters in hardware and software.
User interface:	Control panel with buttons and an 16x2 LCD-display.

interaction, and an LCD-display provides information to the user about sample rates and filters during the system's operation.

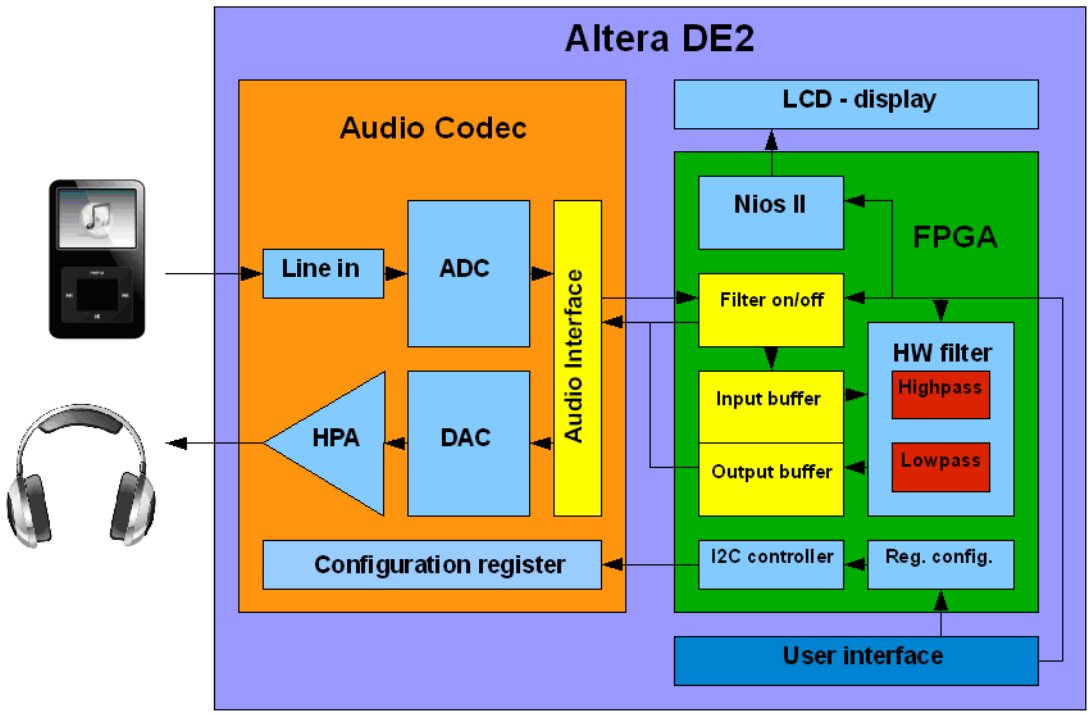

Figure 8.1: Embedded audio demonstrator - block diagram.

The ADC in the system provides an adjustable sample rate, and samples the analog input signal with 8, 32, 48 or 96 kHz. The operation of the ADC is configured by a module on the FPGA, setting different configurations based on inputs from the user interface. After conversion, the digital datastream is sent through the digital audio interface to the FPGA for further manipulation. A module at the input of the FPGA selects whether the signal is to be filtered or not, based on inputs from the user interface on the DE2 board. If the digital filters are deactivated, the datastream is sent directly back to the audio-codec. If not, the datastream is converted to 24-bit packages in the input buffer and sent to the filter module for digital filtering. The filter module includes both high-pass and low-pass possibilities, controlled by inputs from the implemented user interface. When filtered, the data packages are converted to a serial datastream in the output buffer. From here, the datastream enters the digital audio interface in the audio-codec, ready for conversion back to the analog domain.

This operation is performed by the DAC, providing the same set of sample rates as the ADC at the input. When converted, the analog signal is amplified in the headphone amplifier and then sent to a stereo 3,5mm jack-connection at the output. Here, the listener can listen to the results of various sample rates and filtering, performed on the original, analog audio signal.

In addition to the implemented modules in the signal path for the audio signal, a Nios II soft-CPU and an LCD-display are used in order to increase the usability of the system. The Nios II implements a driver for the LCD-display, making it possible to print out customizable text strings of up to 2 x 16 characters. This possibility is used to provide the user with information about sample rates and filters during the system's operation.

8.2 I^2C-Controller

The audio manipulator supports a set of configurations that changes the operation of the system based on inputs from the user interface. The choice of communication protocol for communication between the audio-codec and the FPGA was based on the protocol supported by the audio-codec. For the WM8731, the designers have implemented the I^2C communication standard for configuration, a well suited protocol for interconnection of peripheral devices in small-scale embedded system. This communication protocol is described in more detail in Chapter 3.

In the beginning of the design process, three different approaches were evaluated in order to implement an I^2C controller on the FPGA. The controller could be...:

1. ...implemented from scratch using VHDL.
2. ...based on a laboratory exercise provided by the department.
3. ...based on an example design developed by Altera.

The first idea was based on creating a VHDL module of the controller that adapts to the I^2C standard. A basic module was designed, supporting the fundamental communication properties of the I^2C protocol, although without any error handling at this stage in the design process. After verifying the basic functionality through simulation, the I^2C standard was studied in more detail to complete the implementation. However, since the implementation of the rest of the protocol seemed to be a time consuming process, it was decided to take a closer look at the two existing designs, in order to get the communication up and running.

The second idea was based on a functional audio system used in a laboratory exercise at the department, since this design included a functional I^2C controller. The design was rather complex, and there was no easy way to extract the I^2C controller from the system in order to use its functionality in the audio demonstrator. It was thus decided to take a close look at the example design from Altera, since this system was less complex and thus easier to understand than the laboratory exercise design.

The example design from Altera includes two modules, providing both generation of configuration packages and a controller for I^2C communication, described in more detail in Appendix A. In these modules, the configuration data for the audio-codec was described directly in hardware. This means that the FPGA will have to be re-programmed if the system for

some reason should implement a different set of configurations, beyond the once specified for the system. A solution to this could be to implement the I^2C controller on a soft-CPU like the Nios II. Here, different configurations for the audio-codec could be written in software and executed on the CPU, without changing the hardware description on the FPGA. However, since this embedded system is designed to perform a dedicated task, this is not considered to be an important feature, although configurability could increase the flexibility of the system. Thus, since the I^2C modules available through the example design from Altera provided the desired functionality, it was decided to base the I^2C controller on Altera's example design. This implementation was also easy to configure, making it possible to adapt the design to this specific implementation, saving time and effort compared with designing a completely new controller in VHDL.

8.3 Audio Signal Path

The audio signal path defines the path that the audio signal travels from the input jack-connection on the DE2 board to the headphone output that provides the result. The block diagram in Figure 8.1 on page 34 shows the main blocks in the audio manipulator and the connection between them, and this section follows the signal from input to output through the system. The path includes both A/D- and D/A-conversion, FPGA data-manipulation and signal amplification for the headphone output.

8.3.1 Line-in

As described in Section 6.4.1, the system supports standard line-out voltages for audio sources like CD- and MP3-players of 2V RMS. The line-in circuit contains a programmable volume controller with an adjustable gain between -34.5dB and +12dB. For the audio demonstrator, this gain was configured to be 0dB, making it possible to feed the system with a standard line-out signal without causing any distortion.

8.3.2 Analog-to-Digital Converter

Operation Modes

The audio-codec contains an oversampled Sigma-Delta ADC for domain conversion of the analog input signals. The resolution of the ADC can be set to either 16, 20, 24 or 32 bits as described in Section 6.4.3. The ADC process data using 24 bits, thus, if 16, 20 or 32 bits resolution at the output is selected, the sample width is either reduced or increased to obtain a 24 bits data width. For the functionality of the audio demonstrator, the ADC is programmed to output 24 bits data samples. This choice was made mainly because this is the resolution that the ADC, and also the DAC, uses for signal processing, in addition to the increased resolution obtained compared to the 16 bits representation.

The audio-codec provides two different modes of operation, as discussed in Section 6.4.3. For this design, the "USB mode" was selected, based on the flexibility of changing the output

sample rate without any change in the MCLK. First of all, this simplifies the implementation of the circuit, because it is only necessary to use one PLL, implemented on the FPGA, to realize the functionality of the ADC. In addition, in a system like this, where one of the main functions is to change the sample rate, it was decided that a constant MCLK would ease the implementation of more sample rates in the future, without considering the clock frequency of the MCLK.

Sample- and Bit-rates

For the audio demonstrator, four different sample rates at 8, 32, 48 and 96 kHz were implemented as a part of the system's functionality, providing two sample rates above and two sample rates below the Nyquist rate for audible signals. The oversample rate, as discussed in Section 4.1 and 6.4.3, was configured to be 250fs, in order to provide the system with the desired sample rates, using a single MCLK at 12MHz. Since the audio-codec only provides two sample rates below 40 kHz, which is the Nyquist rate for audible signals according to the Nyquist theorem presented in Section 4.1, both of them were implemented in order to give a good demonstration. Sampling at 8 kHz limits the frequency band to 4 kHz according to the Nyquist theorem, and removes much of the content of the music. The bitrate at this sample rate, considering a resolution of 24 bits per sample, is 384 kbps. of un-coded audio data. The bitrates for the other sample frequencies are shown in Table 8.2, calculated using Equation (4.1) on page 11.

Table 8.2: Bitrate of the output signal from the ADC.

Sample rate (kHz)	**Bitrate (kbps)**
8	384
32	1536
48	2304
96	4608

The two upper sample frequencies, 48 and 96 kHz respectively, are chosen in order to demonstrate the difference between sample rates above the Nyquist limit and those below, but also to compare them to each other. The analog output signal from a CD-player contains 1411 kbps. of audio data, where the digital data on the CD is originally sampled at 44.1 kHz. In the audio demonstrator, this analog audio signal is sampled again, also with sample rates that exceeds the sample rate of the original input signal at 44.1 kHz. Although it is not possible to create a representation that is better than the source signal provided by the CD-player by re-sampling it, a higher sample rate will produce a better reproduction of this signal compared with a lower sample rate. Thus, a higher sample rate in the ADC takes us closer to the source signal from the CD-player when the signal once again is converted to the analog domain. Because of this, the sample rates at 48 and 96 kHz in the audio demonstrator demonstrates how sampling above the Nyquist level retains the frequency content of the audio signal, in addition to a demonstration of how these two differs in sound quality due to the difference in representation of the original source signal.

Listening Test

An informal forced-choice discrimination test, described in Section 2.3, was arranged to see if it was possible to determine the difference between the two sample rates above the Nyquist frequency for audible. Four subjects participated, listening to Mark Knopfler's "Our Shangri-la" sampled at either 48 or 96 kHz through the audio demonstrator. It should be mentioned that the ADC and the DAC uses the same sample rate during operation, thus the DAC sample rate in this case is also 48 or 96 kHz.

Two of the subjects reported that they preferred the 96 kHz sample rate above the one at 48 kHz, claiming that Mark's voice became clearer and that the general detail level in the music increased. The two other subjects claimed the opposite, reporting that the sample rate at 48 kHz gave the most preferable sound. Although this test was not performed under the right conditions, the results are interesting. Three of the subjects are students at Department of Electronics and Telecommunication, while the last studies at Department of Electric Power Engineering. As a result of this, all subjects knew, although at different levels, something about sampling and sample rates. This may have affected the results, due to the subjects understanding on how this may influence the quality of the result. In addition, the listening test was not performed under ideal conditions, with some level of background noise interfering the subjects during testing. However, although the subjects may have been affected by the fact that they expected a variation, the test results show that all subjects reported that they heard some level of difference between 48 and 96 kHz sampling on the analog input signal. The implementation of two sample rates above the Nyquist rate was thus considered to add an interesting demonstration to the sample rate functionality, making it possible to test the difference on the group or person who attends the demonstration.

8.3.3 Digital Audio Interface

As described in Section 6.4.2, the digital audio interface is used to transmit audio data to and from the audio-codec. This interface supports both master and slave mode, and is used to synchronize the data transmission between the audio-codec and the FPGA, as seen in Figure 6.6 on page 28. For the audio demonstrator, master mode was selected. When the audio-codec is in master mode, the FPGA only needs to provide the audio-codec with a master clock, and the codec returns the other signals necessary for audio data synchronization. This simplifies the implementation, and insures that the transmission is correct.

The signal flow diagram in Figure 6.7 on page 28 presents the timing for data transmission between the audio-codec and the FPGA. In addition to control the presence of either the left or the right channel on the ADCDAT line, ADCLRC is also used in the audio demonstrator as an enable signal, telling the FPGA input buffer presented in the next section that a data package with samples is on its way. In the same way, for data transmission from the FPGA output buffer to the DAC through the DACDAT line, DACLRC is used to initialize the transmission. ADCLRC and DACLRC may thus be used as synchronization signals in the design of the audio input- and output buffers, since these signals initializes the data transmission both to and from the audio-codec.

8.3.4 Audio Input Buffer

The digital audio interface implemented in the audio-codec controls the data flow between the codec and the FPGA. To support the synchronization of the data transmission, an input buffer was implemented on the FPGA, providing the digital filters with the right audio data at the right time.

This module receives serial audio data in packages of 48 bits from the digital audio interface in the audio-codec. Each packet contains 2 x 24 bits of audio samples, where the first 24 bits represents the left channel sample, and the last 24 bits represents the right channel sample. The main functionality of the input audio buffer is to receive this data package, convert it to a parallel representation, and transmit the audio data in two packages of 24 bits each in parallel to the digital filter module presented in the next section.

The implementation of the input audio buffer was performed in VHDL using Quartus II, described in Section 7.1. A more detailed description is presented in Appendix B. To make it possible for the user to select whether the audio signal should be filtered or not, some logic were implemented at the input, making it possible to connect the ADCDAT directly to the DACDAT output, by-passing the digital signal processing functionality of the FPGA. This makes it easier to demonstrate how the filtering affects the input signal, by turning the filters on and off while the system operates.

8.3.5 Hardware FIR Filters

One of the main features of the embedded audio demonstrator is digital filtering, described in Chapter 5. The goal is to demonstrate how different filter characteristics affects the perception of the sound, and introduce the possibility of filtering in the digital domain. For the filters implemented in this system, the most important feature is the demonstration effect. The trade-off between filter performance and resource use was thus not considered to be an important factor, although a test was performed to study the correlation between the number of filter coefficients and the use of resources on the FPGA.

Overall Design

The low-pass filter in the system is seen in Figure 8.2. This module, and the corresponding high-pass filter, were designed using the Altera MegaWizard Plug-In Manager FIR Compiler described in Chapter 7.2. Compared to designing the filters in VHDL or Verilog, this graphical interface simplified the filter design considerably, by providing different configuration options and generation of the specified filter to a functional VHDL module ready for implementation. As seen in the figure, two tri-state buffers are connected to the "ast_source_ready" and the "ast_source_valid" outputs, due to the use of two different modules for high-pass and low-pass filtering, since these outputs will drive the same output buffer. The tri-state buffers enable signals are connected to the reset signal of the filter module, and the filter module that is not used when the filters are active will be held in a reset state, setting the tri-state buffers to a high impedance.

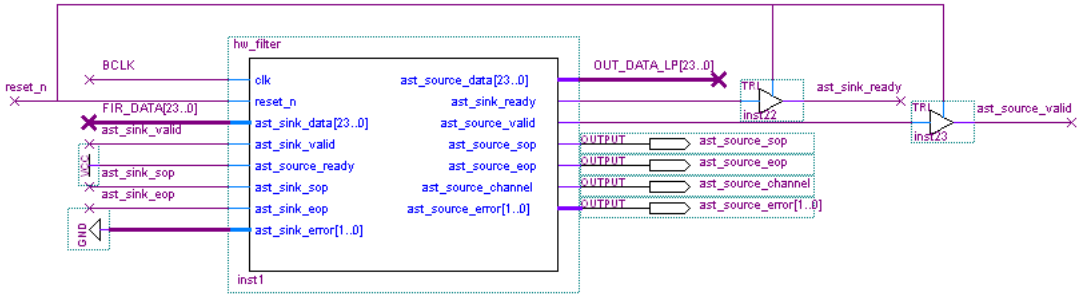

Figure 8.2: Hardware low-pass filter on the FPGA.

Coefficient Generation

Figure 8.3 shows a screenshot of the coefficient generator in the FIR Compiler. This generator contains a set of configuration possibilities that specifies the filter's operation, and creates a set of coefficients that implements this functionality. Filter type makes it possible for the designer to select the desired filter characteristic for his or her design. For the audio demonstrator, two different characteristics were implemented, namely a low-pass and a high-pass filter. These two characteristics were considered to give a good and effective demonstration of filters, with results that should be easy to perceive. However, the effect of the filtering is not only dependent on whether the characteristic is high-pass or low-pass, but also on the filter's specification. For the low pass filter, the cut-off frequency was set to 100 Hz, ideally removing all but the lowest frequencies from the input signal. It was decided through listening tests, discussed in Chapter 2, that the cut-off frequency should be as low as possible, due to the fact that this filter-implementation is far from ideal, including frequencies above the cut-off point in the resulting output signal. For the other filter, the goal was to find a cut-off frequency that removed most of the low frequencies, but at the same time preserved the sound level and the characteristics in the music. This is necessary because it is desirable that the listener recognizes the song, but at the same time perceives the results of the filtering. Through listening tests, a cut-off frequency of 2 kHz was selected. In the same way as for the low-pass filter, the output signal from the high-pass filter will include frequencies below the given cut-off point due to the non-ideal filter characteristics. However, a cut-off frequency at 2 kHz seemed to provide a good demonstration of low frequency removal, and at the same time maintain a decent level of the signal strength. Higher cut-off frequencies were considered, but discarded through listening tests because of the gradually decrease in signal level at the output as the cut-off frequency increased, due to the energy removal of the input signal.

The coefficient generator also provides a possibility of selecting the number of coefficients used in the filter to be implemented. The number of coefficients determines the performance of the filter, due to the filter order specified as the ((number of filter coefficients) - 1). An increase in the number of coefficients takes the frequency respons of the filter closer to the ideal case, but increases at the same time the filter's complexity. For the audio demonstrator, the system's complexity is restricted by the number of LEs available in the FPGA. In this case, before the implementation of the FIR filters, the system used about 6% of a total of 33216 LEs on the FPGA. The FIR filters could thus occupy up to 94% of the FPGA's resources in order to implement a filter demonstration that seemed good enough. The complexity of the filter

is more or less proportional with the number of filter coefficients as seen in Table 8.3, which shows the relation between the number of coefficients in the low-pass filter and use of logical elements on the FPGA.

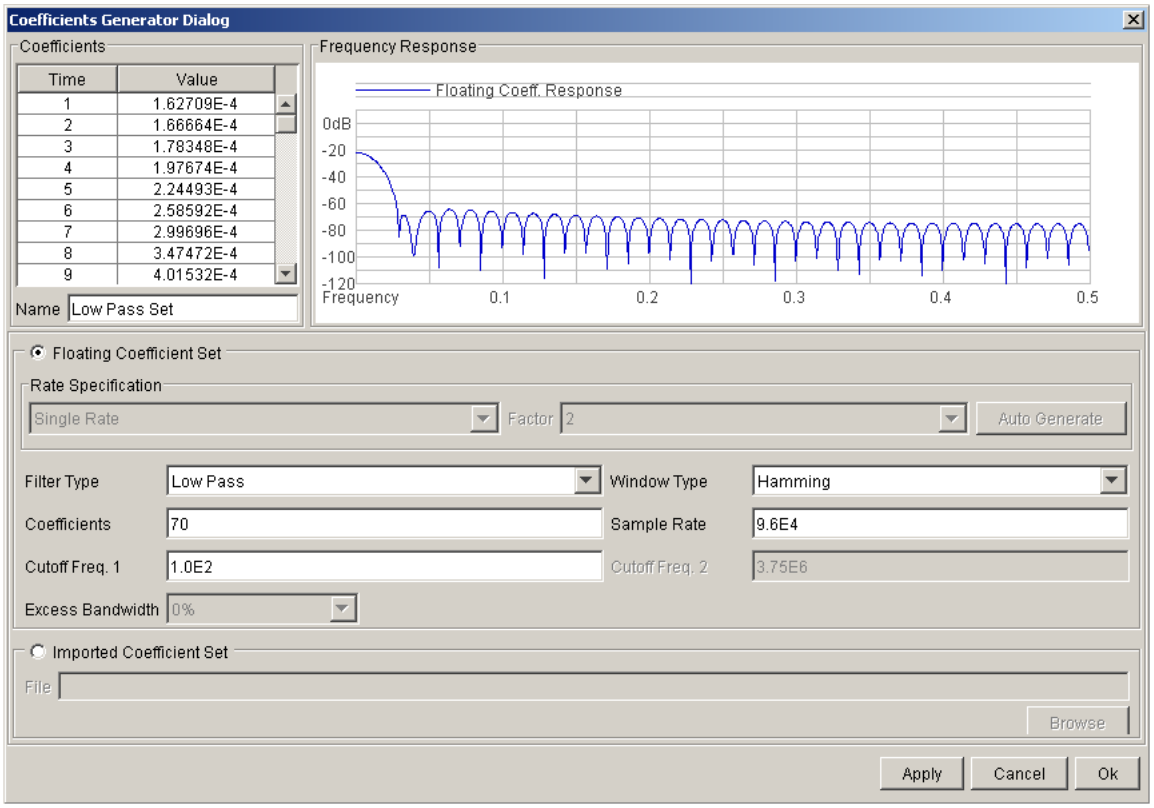

Figure 8.3: Screenshot of coefficient generator in Altera FIR Compiler.

Table 8.3: Filter coefficients for low-pass filter and the usage of logical elements.

Number of coefficients	Number of logical elements
10	1034 (3%)
20	1571 (5%)
30	2070 (6%)
40	2627 (8%)
50	3144 (9%)
60	3643 (11%)
70	4282 (13%)

The number of coefficients were increased in steps of ten in order to find a reasonable trade-off between resource use and the effect of the filter demonstration. The goal for the low-pass filter was to reach a sufficient level of attenuation of high frequencies in order to obtain a good demonstration. Figure 8.4 shows the frequency response of the low-pass FIR filter with 10 filter coefficients. This frequency responses is normalized for a sample rate of 96 kHz, and shows a considerable level of attenuation of all frequencies, although the attenuation increases, as expected, with the frequency. This respons was not considered to give a sufficient demonstration effect through listening tests, and because of the moderate resource use of only 1034 (3%) of the LEs, it was decided to increase the number of filter coefficients to a higher level.

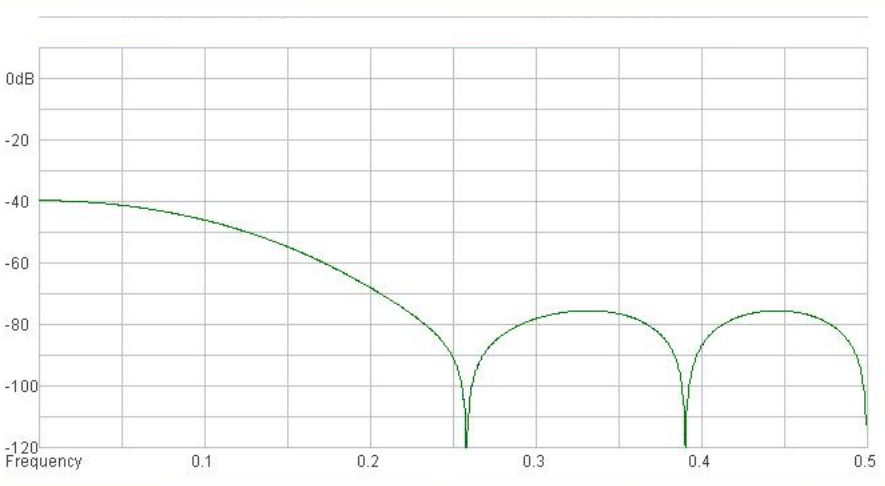

Figure 8.4: Low-pass frequency respons using 10 coefficients and rectangular window.

To get a better representation of the low-pass filter with a cut-off at 100 Hz, 70 filter coefficients were selected, providing the FIR filter with a filter order of 69 at 13% of the LEs on the FPGA according to Table 8.3. The frequency respons of this filter is seen in Figure 8.5. Here, the main lobe of the respons is more narrow and less attenuated than the same filter characteristic with 10 coefficients, seen in Figure 8.4.

Although the FIR filters could use up to over 90% of the resources on the FPGA, there was no need to do this in order to get a demonstration of the FIR filter that seemed good enough. It was also considered to be practical to allow some level of free space for future work and implementations on the FPGA. In addition, the system seemed to get some hold time violations during compilation when the number of filter coefficients reached a certain level. These violations were often related to the BCLK provided by the audio-codec, clocking the different modules in the path of the audio signal. As a result of this, the sound through one or both of the filter modules could be distorted during operation, thus removing their functionality completely. The hold time violations were improved through configurations in the Quartus II software, optimizing the synthesis operation for speed, and removing the optimization for power consumption. However, a certain level of complexity in the filter resulted in hold time violations for the system, and although further work could have resolved the problem, 70 coefficients were selected, making a good trade-off between system functionality, demonstration effects and system resources.

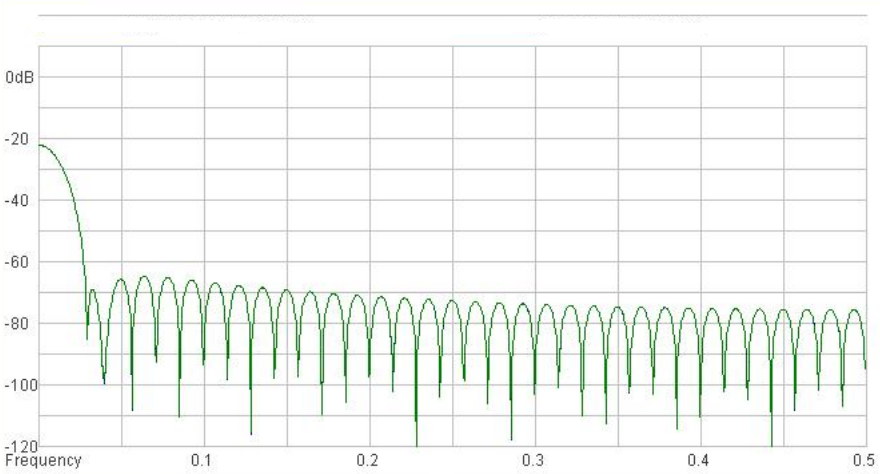

Figure 8.5: Low-pass frequency respons using 70 coefficients and rectangular window.

In the case of the high-pass filter, it was noticed that the number of coefficients needed to make a good demonstration of the filter effect was less than for the low-pass filter. Figure 8.6 shows the frequency respons of the high-pass filter with 40 coefficients and a cut-off at 2 kHz, normalized for a sample frequency of 96 kHz. As seen in the figure, the high-pass filter attenuates some of the lowest frequencies to below -40dB, and reaches the -3dB level at approximately 2.5 kHz. The effect of this respons was easy to perceive, and gave a good demonstration of high-pass filtering. Although higher cut-off frequencies were considered, the signal strength became weaker as more of the total input energy was removed, making it more difficult to perceive the effects through a pair of stereo headphones. This could have been solved by an increase in amplification in the headphone amplifier at the output when the high-pass filter is active, but since the demonstration effect seemed to be good enough at the given specifications, it was decided to keep the configuration and save the system's development time. However, for future work, the implementation of additional FIR filters with different coefficient lengths at the same characteristics could provide an interesting demonstration on how the number of coefficients affects the performance of the filter.

It should be noted that the cut-off frequency of the FIR filters are proportional to the sample rate used in the audio-codec, as described in Section 5.1. The coefficients specified for the two filters in this system are generated using a sample rate of 96 kHz in the FIR Compiler interface. However, since the cut-off frequency is reduced when the sample rate decreases, it is possible to achieve an even better demonstration of the filter effects, at least for the low-pass filter, since the reduced sample rate reduces the frequency spectrum of the audio signal. As a demonstration effect, this could thus be used to increase the perception of the filter's characteristic, without changing the filter coefficients. However, since the variations in sample rates also increases the number of parameters that the listener will have to relate to, it is recommended that the sample rate is kept constant at 96 kHz when the demonstrations of the filters are performed, at least for the youngest audience.

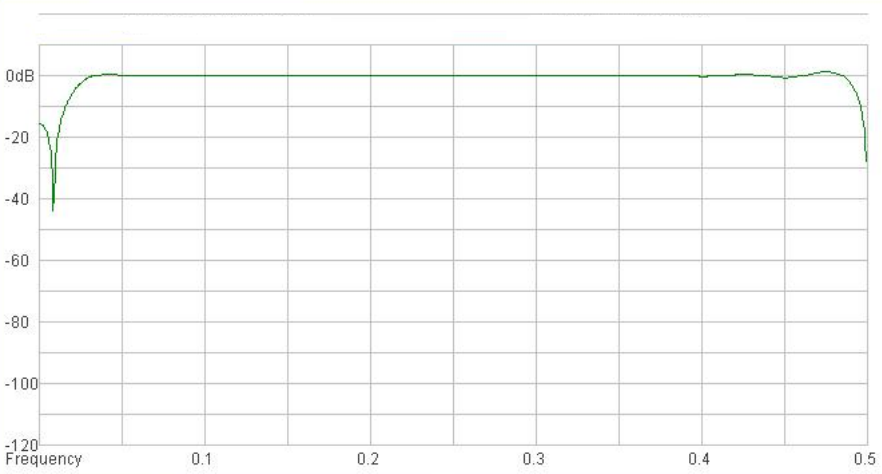

Figure 8.6: High-pass frequency respons using 40 coefficients and rectangular window.

Window Functions

The coefficient generator provides the designer with different window functions used to determine the filter coefficients as described in Section 5.1. Four different windows are provided, namely the rectangular, Hanning, Hamming and Blackman windows, which are used to adjust the filter respons to different applications. For the high-pass filter, a rectangular window was selected. This window is the least complex of the four windows provided, and gives a stop band attenuation of 13.2dB for the first side lobe in the in the frequency response. However, although the Hamming window provides more attenuation in the stop band, the main lobe of the rectangular window is more narrow, providing a faster attenuation of unwanted frequencies for a given number of coefficients. In this case, with 40 coefficients, the main lobe of the frequency respons using the Hamming window got to wide, pushing the first side lobe below the 0 Hz limit, as seen in Figure 8.7. With an increase in the number of filter coefficients, the main lobe got steeper, increasing the performance of the attenuation. However, since the rectangular window was considered to work good enough with 40 coefficients and a cut-off frequency at 2 kHz, attenuating frequencies at about 1.2 kHz with more than 40dB as seen in Figure 8.6 , this solution was selected for implementation.

For the low-pass filter, where the main part of the signal spectrum is to be attenuated, a Hamming window was chosen. This window gives a stop band attenuation of 43dB for the first side lobe in the frequency respons, with an increase in attenuation of 6dB per octave. For the high-pass filter, where the main part of the frequency spectrum passes without attenuation, it seemed important to get a fast attenuation of the frequencies that should be suppressed. For the low-pass filter however, where most of the energy is removed in the filter process, an overall attenuation at a higher level was considered to be more important than a narrow main lobe. Because of this, the Hamming window was selected, providing a good frequency respons for a low-pass filter with a cut-off point at 100 Hz, as seen in Figure 8.5.

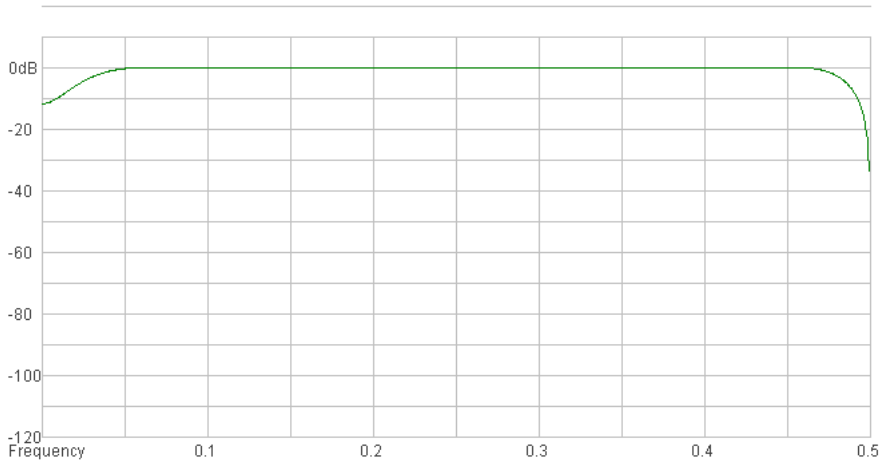

Figure 8.7: High-pass frequency respons using 40 coefficients and Hamming window.

Bit Depth of Coefficients

Another important factor for the performance of the FIR filters, in addition to the number of coefficients, is the coefficient resolution. The filter coefficients for these filters are decimal numbers between -1 and 1, represented by a selectable number of bits. Figure 8.8 shows a screenshot of Altera's Fir Compiler, presented in Section 7.2. This interface provides a number of specifications, making it possible to control many aspects of the filter's operation.

The coefficients in the coefficient generator described in Section 8.3.5 are generated as a floating-point set. However, due to the fact that the FIR Compiler only generates integer-coefficient FIR filters, the coefficient set will have to be transformed in an float-to-fixed-point operation. The FIR Compiler provides a selectable bit width of the filter coefficients from 2 to 32 bits. The resolution of these coefficients determines the precision of the transformation, due to an increased range of integer-coefficients as the bit width increases. As an example, 4 bit coefficients provides an integer range from -8 to 7, thus a maximum of 16 different values to represent the integer-coefficients. As a demonstration, Figure 8.9 shows the frequency respons of the filter for both the original floating-point coefficients, and the fixed-point integer coefficients calculated using 4 bits resolution. As the resolution increases, the fixed-point representation of the frequency response gets closer to the original floating-point respons, although this again increases the filter's complexity. Due to this, the number of bits used in the coefficients for internal signal processing were increased until there was little or no difference in the two frequency responses presented in the FIR Compiler. In this case, for both the high-pass and the low-pass filter, a sufficient fixed-point integer representation was found when the bit width was 16 bits. An additional listening test could have been performed in order to support the selection of this bit width. However, since the increase in bit width made little change in the area use on the FPGA, in addition to the fact that a listening test would have been a time consuming process, it was decided to base the selection of the bit-width on the data provided by the FIR Compiler.

The FIR filter uses 45 bits for internal calculations when the audio samples at the input contains 24 bits, and the resolution of the coefficients are set to 16 bits. Since the audio-codec

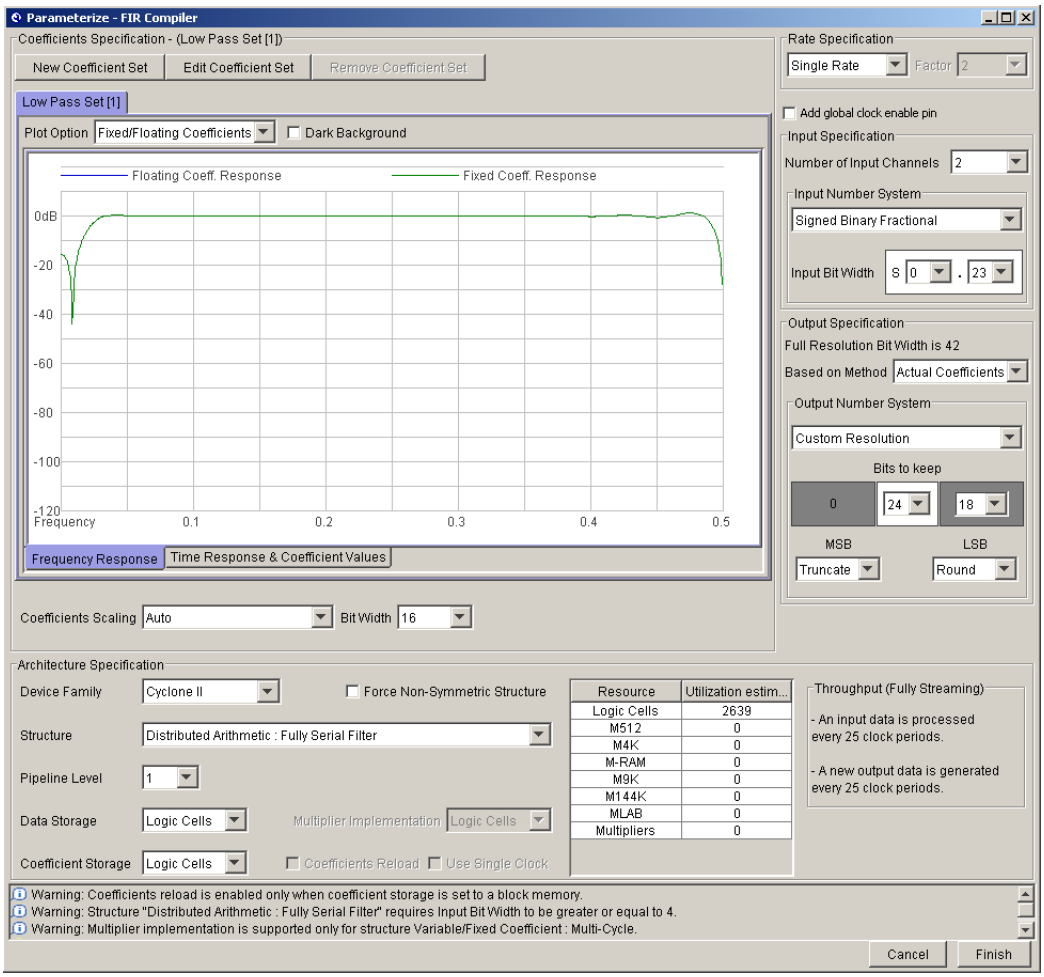

Figure 8.8: Screenshot of Altera's FIR Compiler

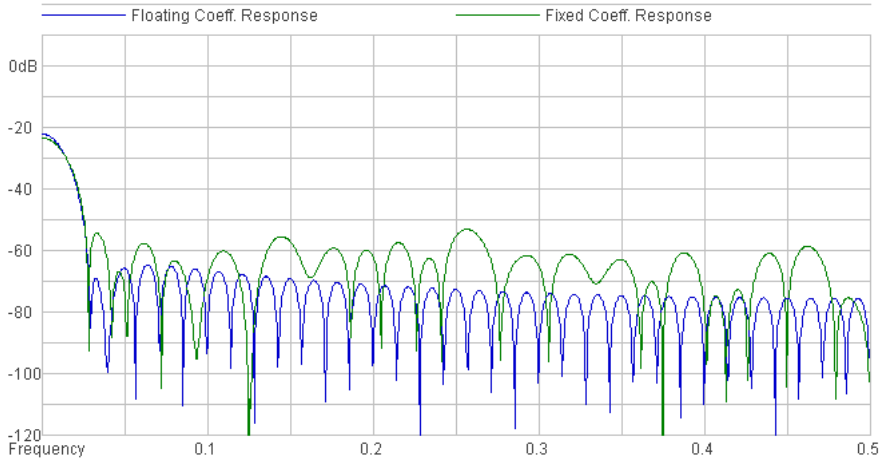

Figure 8.9: Frequency respons of the low-pass filter with float-coefficients(blue) and fixed-coefficients(green).

only supports an input of 24 bits to the digital audio interface and the DAC, the output from the two FIR filters must be truncated and rounded in order to support this specification. The FIR Compiler provides a custom resolution of the output data, making it possible to decide which part of the total data width that should be put on the FIR filter's output, and if the rest of the bits should be rounded or truncated. In this case, it was decided to keep the 24 MSBs of the audio data, since this is where the most decisive parts of each sample is kept. The 21 LSBs were truncated by the FIR Compiler, creating a 24 bits representation of the filtered audio data at the output, ready for transmission back to the audio-codec and domain conversion to the analog domain.

As an additional demonstration, it was considered to implement a possibility to select between different bit widths in the system. The audio-codec supports 16, 20, 24 and 32 bits of audio data through register configuration, and a selection between these could add an interesting demonstration of bit depth versus sound quality. However, the FIR filters implemented through the MegaWizard FIR Compiler must be generated with a defined bit width. Due to this, it is not possible to get a functional selection between different bit-depths in the FIR filters without implementing one filter module for each bit-width. This could have been performed, but since the lowest available bit-rate in the audio-codec is 16 bits, the same resolution as used in audio CDs, it was assumed that the demonstration effect would be difficult to perceive, due to the high resolution. This functionality was thus not implemented.

Filter Architecture

The FIR Compiler provides four different options of filter structures that can be selected for implementation. These are fully serial, multi-bit serial, fully parallel or multi-cycle filter structures, each with different specifications related to area and speed. In the case of the audio demonstrator, area use on the FPGA was not an initial consideration, although it was not considered to be unessential. When it comes to speed, the minimum requirements of the FIR filters are related to the maximum sample rate provided by the audio-codec. This is thus an important factor for the system to operate properly.

With the highest sample rate of 96 kHz, the audio-codec feeds the input buffer and the FIR filter with 192.000 audio samples per second, due to the sampling of both left and right channel. The FIR filters in the system are clocked with a clock frequency of 12 MHz, the same clock that drives the other modules in the signal path of the audio data. In the fully serial case, these filters requires 25 clock cycles to process one audio sample from input to output. This gives the filter a throughput of 480.000 samples per second, more than twice as much as required for this application, calculated as $\frac{clockfrequency}{processtime}$. Throughput calculations and use of area for all architectures for the low-pass filter with 70 coefficients are seen in TSable 8.4.

The initial idea for the filter implementation was to use the embedded multipliers implemented in the FPGA to increase the system's performance. However, it turned out that only one of the four architectures provided by the FIR Compiler, namely the multi-cycle structure, uses these multipliers for the filter implementation. The three other structures, including the fully serial filter, uses the logical elements in the FPGA to realize the multiplier blocks. The multi-cycle structure has a maximum throughput of one audio sample per clock cycle, and is

Table 8.4: Filter architectures, throughput and use of resources for a LP-filter with 70 coefficients.

Filter architecture	**Throughput**	**Number of logical elements**
Fully serial	480000 samples/sek	4221 (13%)
Multi-bit serial	923076 samples/sek	4634 (14%)
Fully parallel	12000000 samples/sek	9588 (29%)
Multi-cycle	12000000 samples/sek	7867 (24%)

thus 25 times faster than the fully serial version. Due to this, the multi-cycle filter gives the best processing performance at the lowest possible use of logical elements. However, since the performance of the fully serial filter provided a sufficient level of performance at a small use of resources, the embedded multipliers in the FPGA remained unused in this system.

8.3.6 Software FIR Filters

The initial idea with the filter implementation was to perform the operation in both hardware and software in order to provide a demonstration of the performance in both of these domains. Hardware/software co-design can be used to meet the system-level objectives in a system, through allocation of modules to the domain that provides the best performance, as discussed in Section 1.2. For the audio demonstrator, an implementation of FIR filters in both hardware and software could make it possible to study the performance difference between these domains, thus increase the system's usability as a demonstrator.

A Nios II soft CPU was implemented on the FPGA as a platform for the software filters in the system. This core is available in three different versions, each specified with different values related to area use and speed performance as described in Section 6.3. For the audio demonstrator, implementing two FIR filters in hardware with 40 and 70 filter coefficients, it was decided to use the fastest Nios core available using a maximum of 3000 logical elements and a maximum clock frequency of 185 MHz, in order to keep up with the hardware filters.

The SOPC builder in Quartus II, described in Section 7.1, provides several IP modules ready for implementation with the Nios II soft CPU. The audio core seen in Figure 8.10 is one of them, providing an interface for audio data to and from the audio-codec. This module was implemented in order to ease the audio data synchronization in the Nios II. The audio input and audio output on the audio-core are connected directly to the serial data lines on the audio-codec, thus by-passing the digital audio buffers implemented on the FPGA to support the hardware filters discussed in Section 8.3.5. The FIFO buffers in the module have adjustable widths, and are used as containers for the digital audio samples.

To see if it was possible to get the digital signal through the Nios II without any errors, a read/write function was implemented as a test through the Altera IDE, presented in Section 7.3. This function reads the audio samples from the input FIFOs and sends them back through the output FIFOs without performing any kind of signal manipulation. This worked without any problems, thus providing a good basis for further development.

The next step was to implement a direct form FIR filter to be executed on the Nios II, reading audio samples from the FIFO buffers and performing filter manipulations before the samples are returned. In the beginning, a first order low-pass FIR filter with only two coefficients

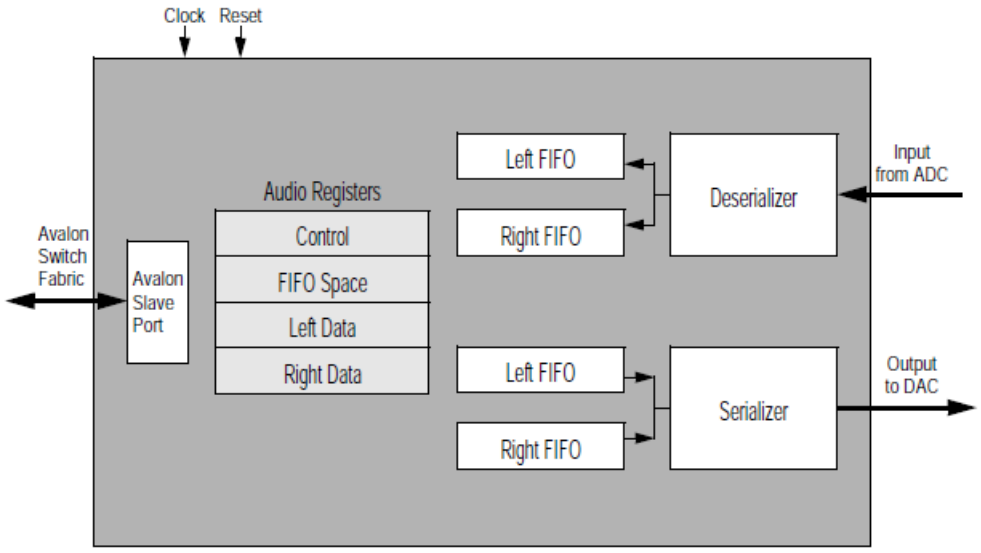

Figure 8.10: Block diagram for audio core in the Nios II, [2].

was implemented in C, in order to test the functionality of the filter function. This function performs multiplication, addition, subtraction and delay operations on the input signal, thus putting demands on the CPU when it comes to arithmetic functionality.

The floating point filter coefficients for the software filter were generated using MatLab's fdatool, providing a graphical interface for specification of FIR filters and coefficient generation for further implementation. Due to the fact that Altera's FIR Compiler only generates fixed-point integer coefficients, the hardware filter performs its operations with coefficients that are scaled with a scaling factor and rounded. In the software filter, a function for coefficient scaling and rounding was not implemented, thus the software function of the FIR filter had to perform floating point arithmetic in order to filter the input samples from the audio-codec.

To test the first order FIR filter, an audio signal was connected to the system. Although the FIR function did its mathematical operations correctly in tests with single numbers as input, the sound signal from the source was completely distorted after filtering. At first, the Nios II processor's performance was considered to be the problem. Thus, a performance counter provided by the SOPC Builder was added to the design, in order to count the number of clock cycles required to perform any function in the C code. However, based on a maximum required throughput rate of 192.000 samples per second, the tests with the performance counter did not reveal any errors related to speed performance. Thus, the problem was considered to be related to the arithmetic operations performed by the FIR filter.

The arithmetic operations consists of performing mathematical operations on the input samples and filter coefficients. As mentioned above, the filter coefficients generated for these filters are floating point coefficients. Floating point multiplication with the signed 2's complement samples delivered from the ADC in the audio-codec might have caused the output errors, and a further study on this topic may have resolved the problem. One solution to this would have been to perform the same float-to-fixed operation as the hardware filters perform, where the

floating point coefficients are scaled and rounded to a set of fixed integer coefficients before the multiplication operation. However, because of the limited time available on this project, this function was not implemented in this system.

So, due to challenges and time constraints during the process, the implementation of software filters was not completed. Unfortunately, this removed the systems ability to demonstrate the performance differences between hardware and software FIR filters, and reduced the co-design aspect in the design. However, due to the fact that the system performed the desired filter operations, in addition to a functionality for adjusting the sample rates in both the ADC and DAC, the value of the demonstrator without software filters was considered to be good enough.

8.3.7 Audio Output Buffer

This output buffer works as an interface between the FIR filters and the audio-codec. When the FIR filter has performed its operation, the output buffer receives the filtered audio samples in parallel as 2 x 24 bit packages. The output buffer transforms these packages to a 48 bits serial data stream, and sends the audio data to the audio-codec.

As for the input buffer described in Section 8.3.4, the output audio buffer was described in VHDL using the Quartus II software. Its operation is specified for this system, and provides support for both the FIR filters and the audio-codec's communication protocols, at the input and output respectively. In order to select which of the two hardware filters that should feed the output buffer with data, a multiplexer is implemented as a selector between the FIR filters and the buffer. This multiplexer is seen in Figure 8.11. A "filter_select" signal from the user interface makes it possible to decide if the system should provide low-pass or high-pass filtering. This signal is also connected to the filter modules, keeping the filter that is not active in a reset state. More details on the audio-buffer are presented in Appendix C.

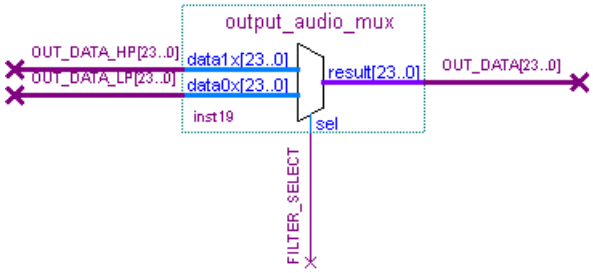

Figure 8.11: Module for selection between highpass and low-pass filtering.

8.3.8 Digital-to-Analog Converter

When the audio samples returns from the FPGA to the audio-codec, they are received by the digital audio interface discussed in Section 8.3.3. This interface, together with the output buffer on the FPGA, synchronizes the data transmission between the two modules. The audio interface is connected to the oversampled Sigma-Delta DAC implemented in the audio-codec,

making it possible to convert the manipulated audio data back to the analog domain, as described in Section 4.2.

The DAC's operation is controlled by the same registers as the ADC, and the two converters shares the same selection of modes and oversampling ratios as discussed in Section 6.4. However, it is possible to achieve different output sample rates from the ADC and DAC, although it was decided to use the same rates in both converters for this system. This was done in order to decrease the number of variables in the demonstration, thus making it easier to describe the effects that the listeners perceive. In addition, there were some limits in the audio-codec regarding the possibility of combining different rates in the ADC and DAC. Thus, whether the ADC has an output sample rate of 8, 32, 48 or 96 kHz, the DAC will have the same, converting the digital signals to the analog domain through oversampling.

As described in Section 8.3.2, a listening test was performed in order to test the difference between sampling at 48 and 96kHz. Since the ADC and DAC uses the same samples rates in this system, this test was actually a comparison of sound quality as a result of both A/D- and D/A-conversion. Due to the acceptable sound quality, and the fact that the number of combinations of different sample rates in the ADC and DAC are limited, it was not considered to perform any other evaluation of the DAC. However, to increase the possibilities for demonstration of A/D-conversion and D/A-conversion, an independent selection of sample rates between the ADC and DAC could be implemented in the future.

8.3.9 Headphone Amplifier

The headphone amplifier seen in Figure 6.10 on page 30 defines the output module of the system, and is used in the audio demonstrator to drive headphones that provides the results of the audio manipulation. Although this module provides an adjustable volume level through register configuration, as described in Section 6.4.4, it was decided to not use development time to implement a function for this. The volume level was thus configured to a fixed level, mainly because most sources that will be used with the audio demonstrator is considered to have an integrated volume controller. However, as a possibility for future work, a function for volume adjustment could increase the flexibility of the system.

8.4 User Interface

Since the system is intended for demonstration, a user interface was implemented in order to let the user interact with the system's functionality. The user interface consists of a control panel with switches and an LCD-display for system feedback, all implemented on the Altera DE2 development board.

8.4.1 Control Panel

The control panel is seen in figure 8.12. For the audio demonstrator, two of the pushbuttons and four of the switches available on the DE2 board were used to control the functionality of the system. During demonstration, the persons who listens to the results of the audio

manipulation should be given the possibility to operate the control panel on their own. This will increase the level of interactivity between the users and the system, and is considered to increase the overall quality of the presentation.

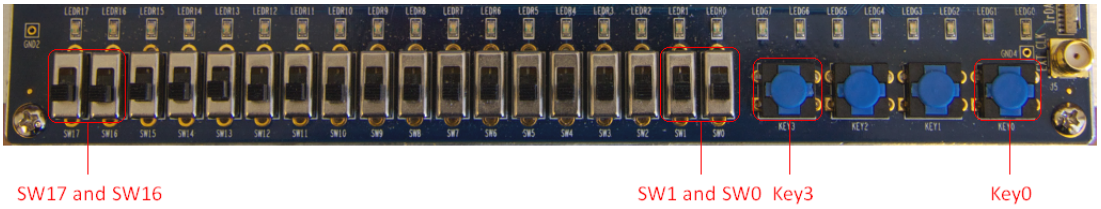

Figure 8.12: Control panel with buttons and switches on the DE2 board.

In Table 8.5, the different switches and push buttons are listed together with their functionality in the audio demonstrator. Table 8.6 and 8.7 includes the different combinations for sample rate and filter selections respectively, and shows how the system should be configured in order to get the desired functionality. It should be noted that the sample rate is updated first when the update-button, Key0, is pushed. Thus, the user will first have to select the desired sample rate with SW0 and SW1, then push Key0 in order for the system to change the configuration.

Table 8.5: Functionality of control panel.

Key0	Configuration update
Key3	System reset
SW0	Select sample rate
SW1	Select sample rate
SW16	High-pass/low-pass filter
SW17	Hardware filter on/off

Table 8.6: Switch combinations for selection of sample rate.

SW1	**SW0**	**Functionality**
0	0	8 kHz sample rate
0	1	32 kHz sample rate
1	0	48 kHz sample rate
1	1	96 kHz sample rate

Table 8.7: Switch combinations for selection of filter operation.

SW17	**SW16**	**Functionality**
0	0	Filter off
0	1	Filter off
1	0	Low-pass filter
1	1	High-pass filter

8.4.2 LCD-Display

As mentioned in Chapter 6.1, the DE2 board includes a 16 x 2 LCD display as seen in figure 8.13. This LCD-display was implemented in order to give the user some feedback from the system regarding sample rates and filter operation. This is considered to increase the demonstration effect, because it will make it easier for the listener to relate what he or she hears to the actual configuration on the system. The LCD-display is updated when a change is made on the control panel, and presents the actual sample rate in line one and the status of the filters in line two.

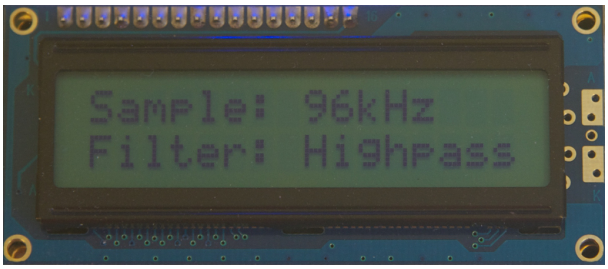

Figure 8.13: LCD-display on the DE2 board.

The LCD display is controlled by an LCD-controller implemented on the Nios II soft-CPU through the SOPC Builder. LCD-control is thus the only operation that the Nios II performs in the system, due to the absence of software FIR filters. However, due to the fast implementation of an LCD-controller through Altera's SOPC builder, it was decided to keep the Nios II in the system for this operation, although an LCD-driver could also be written directly in VHDL, possible saving area cost for the system. As mentioned in Chapter 6.3, the Nios II is available in three different sizes regarding area and performance. The initial system used the fastest and most area consuming soft-CPU version due to the performance requirements of the FIR filters. However, since these filters were not implemented in the audio demonstrator, the Nios II is only used for LCD-display configuration. Thus, the slowest and least area consuming Nios II version was implemented, providing a speed performance of 31 DMIPS at less than 700 logical elements.

To implement the functionality of the LCD-display, functions for writing to the two LCD-lines were written in the Nios II EDS development software, described in Chapter 7.3. The Nios II is connected directly to the control panel presented in the Section 8.4.1, and changes the content of the display based on the inputs from the push buttons and switches. Since the LCD-functionality is the only operation that the Nios II will perform, it was decided to implement this operation through polling. Polling, as discussed in Section 6.3.1, means that the soft-CPU will have to monitor the operation of the push-buttons and switches continuously, updating the LCD-display whenever a change occurs. Although polling wastes CPU-time, this was not considered to be a problem for the audio demonstrator, due to the fact that the soft-CPU only performs operations related to the LCD display. However, for future development, if the software FIR filters are to be implemented, the system should support interrupt handling in order to increase the system's performance.

8.5 Summary - System Implementation

The system was implemented on Altera's DE2 development and education board, based on the specification presented in Section 1.1. Although the final system did not provide FIR filter operation in software, the overall system functionality with variable sample rates and filtering in hardware were implemented and verified, providing the audio demonstrator with several functions for demonstration of signal processing topics. To illustrate the specifications of the final implementation, Table 8.8 presents the different operation modes that the audio demonstrator provides, together with information about the features available in each mode.

As described in Section 8.3.5, the cut off frequency is proportional to the sample rate. Both filters were specified with a sample rate of 96 kHz, and although the cut-off frequencies where specified to be 100 and 2000 Hz for low-pass and high-pass filters respectively, the cut-off frequency will change when the sample rate is configured to something else than 96 kHz. Due to this, the cut-off frequencies in Table 8.8 are calculated from a base sample rate of 96 kHz.

Table 8.8: Specification of the implemented system.

Mode / **Features**	**8 kHz**	**32 kHz**	**48 kHz**	**96 kHz**
Audio bandwidth	4 kHz	16 kHz	24 kHz	48 kHz
Audio bitrate	24 bits	24 bits	24 bits	24 bits
Filter type	High-pass/low-pass/off	High-pass/low-pass/off	High-pass/low-pass/off	High-pass/low-pass/off
Filter domain	Hardware	Hardware	Hardware	Hardware
Cut-off frequency low-pass	8.33 Hz	33.3 Hz	50 Hz	100 Hz
Cut-off frequency high-pass	166.7 Hz	666.7 Hz	1000 Hz	2000 Hz

Chapter 9

Demonstration

As described in Chapter 1.3, it is important to consider different topics related to pedagogics when demonstrating an embedded system. This chapter presents a guideline to how the system should be introduced in order to make the listeners motivated, and provides knowledge to the demonstrators on how the system's functionality should be used in the presentation.

The demonstration consists of the embedded audio demonstrator implemented on Altera's DE2 board, an audio source with an analog output and a set of headphones or active speakers. In most cases, if more than one person attends the demonstration at once, a pair of active speakers should be used in order to demonstrate the effects to more than one person at the time. It is important that these speakers are active, thus contains an internal amplifier, due to the low output power from the headphone output on the DE2 board. In addition, whether active speakers or headphones are used, they should provide a good representation of all frequencies in the audible spectrum in order to present the effects of different sample rates and filter characteristics in a perceivable way. Headphones limits the perception of the demonstrated effects to one person, but reduces at the same time the level of background noise for the listener. This makes it easier to perceive the results, and headphones should thus be considered if the group contains few people.

As mentioned earlier, the system supports any audio source with an analog line-out level of maximum 2V RMS. If one of the persons who attends the demonstration has a portable music device like an MP3-player or mobile phone with MP3 possibilities and jack-connection available, this devices should be used as source, if possible, in order to demonstrate that the audio manipulation might be performed on any audio source with an analog output.

The demonstration is based on an oral presentation of the topics that the demonstrator presents. In addition, a poster with a block diagram and an illustration of the signal flow, together with some motivation for use of the demonstrator, is included to complete the demonstration. This poster is presented in Appendix D on page 73. However, the main focus in the demonstration should be on the presentation provided by the person who demonstrates the system, and the effects presented by the demonstrator. Table 9.1 describes how different motivation- and demonstration-techniques could be used in order to adjust the presentation to different age- and education-levels. It is important that the demonstrators who demonstrates the system takes these factors into consideration before the presentation starts.

Table 9.1: Motivation and Presentation of the Audio Demonstrator.

Target group	**Motivation**	**Demonstration**
Primary school pupils	The possibility to change the sound through the DE2 board.	Give a short demonstration on how the different configurations changes appearance of the sound, but focus on providing the pupils with a chance to configure the system on their own. Answer questions.
Secondary school pupils	Present trade-off between sound quality and sample rate(bit rate), and relate this to portable sound systems. Describe equalizers in home stereo systems and MP3-players in relation to filters.	Demonstrate the system with different sample rates without filtering. Then, set the sample rate to 96 kHz and change between the low-pass and high-pass filters. At the end, allow the listeners to change the configuration of the system on their own. Answer questions.
High school/university students	If the group study electronics, the motivation could be related to theoretical courses that they have had. Since both FIR filters and sampling might be familiar, the motivation should focus on the fact that this system is a practical example of how to implement this theory in a design. For students without any background in electronics, the motivation should be the same as for the secondary school pupils.	Demonstrate the system with different sample rates without filtering. Then, set the sample rate to 96 kHz and change between the low-pass and high-pass filters. In addition, present the fact that the cut-off frequency changes with the sample rate, and demonstrate how the high-pass and low-pass filters sound differently with a lower sample rate than 96 kHz. At the end, allow the target group to change the configuration of the system on their own. Answer questions.

Chapter 10

Conclusions

Embedded systems provides a wide range of possibilities for the system designer when designing an embedded demonstrator for demonstration of topics related to electronics. The use of configurable hardware like FPGAs leads to shorter development time as well as an increased flexibility in the design process of such systems.

In this master thesis, and embedded system for audio manipulation was implemented in order to demonstrate topics related to digital signal processing. Through the use of a Cyclone II FPGA and a Wolfson WM8731 audio-codec, both implemented on Altera's DE2 board, the system performs real-time demonstration of adjustable sample rates and different filter characteristics.

A good demonstration of embedded systems requires both motivation and activation of the attending audience. For motivation, the person who presents the design should relate the systems functionality to something that the people who attends the demonstration are familiar with. In this case, sampling and filtering could be related to bitrates of MP3-files and sound equalizers in CD- and MP3-players in order to provide a better understanding of the system's functionality. To activate, interactive design features should be added to the system in order to let the user be a part of the demonstration. For the embedded audio demonstrator, a user friendly control panel and an LCD-display for system feedback are implemented in order to provide the listener with control of the system's operation. These factors, together with the multimedia content that the system provides through manipulation of audio, creates a good basis for a successful demonstration.

Further development could increase the demonstration quality of the implemented system, and extend the number of demonstration effects in the audio demonstrator. In addition to adjustable sample rates and high-pass and low-pass FIR filtering in hardware, the implementation of software FIR filters could provide the system with an interesting demonstration of the performance difference between filtering in these two domains. There could also have been performed a more in-depth study of which filter characteristics that are best suited for demonstration of filters in general. Additional listening tests could be used as a tool in this research. In addition, due to the wide possibilities of the Altera DE2 system platform, several new features could be implemented in the future to increase the number of topics covered by the audio demonstrator.

Bibliography

[1] Altera Corporation. *DE2 Development and Education Board User Manual, Version 1.4*, 2006.

[2] Altera Corporation. *Audio Core for Altera DE2/DE1 Boards, Version 1.0*, May 2007.

[3] Altera Corporation. *Cyclone II Device Handbook, Volume 1*, February 2007.

[4] Altera Corporation. *FIR Compiler - User Guide*, November 2009.

[5] Altera Corporation. *Introduction to the Quartus II Software, Version 9.0*, 2009.

[6] Altera Corporation. *Nios II Processor Reference Handbook, Version 9.0*, March 2009.

[7] Altera Corporation. *Nios II Software Developer's Handbook*, March 2009.

[8] Altera Corporation. *Quartus II Handbook Version 9.0 Volume 4: SOPC Builder*, March 2009.

[9] Altera Corporation. *Quartus II Handbook Version 9.1 Volume 1: Design and Synthesis*, November 2009.

[10] Ballou, G. M., Ed. *Handbook for Sound Engineers*, third ed. Focal Press, Jordan Hill, Oxford, UK, 2002.

[11] Bourdopoulos, G., Pnevmatikakis, A., Anastassopoulos, V., and Deliyannis, T. L. *Delta-Sigma Modulators*. Imperial College Press, Covent Garden, London, UK, 2003.

[12] Catsoulis, J. *Designing Embedded Hardware*, second ed. O'Reilly Media, Inc., Sebastopol, California, USA, 2005.

[13] de Michelli, G., and Gupta, R. K. Hardware/Software Co-Design. In *Proceedings of The IEEE, Vol.85, NO.3, March 1997* (March 1997), IEEE.

[14] Emery, W. J., and Thomson, R. E. *Data Analysis Methods in Physical Oceanography*, second ed. Gulf Professional Publishing, Oxford, UK, 2001.

[15] Fries, B., and Fries, M. *Digital Audio Essentials*. O'Reilly Media, Inc., Sebastopol, California, USA, 2005.

[16] Fuming Sun, Xiaoying Li, Q. W., and Tang, C. Fpga-based embedded system design. In *APCCAS - IEEE Asia Pacific Conference on Circuits and Systems* (2008), IEEE, pp. 733 – 736.

[17] Geerts, Y., Steyaert, M., and Sansen, W. *Design of Multi-bit Delta-Sigma A/D Converters*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2002.

[18] Grout, I. *Digital Systems Sesign with FPGAs and CPLDs*, fourth ed. Newnes, Burlingston, Massachusetts, USA, 2008.

[19] Havelock, D., Kuwano, S., and Vorländer, M., Eds. *Handbook of Signal Processing in Acoustics, Volume 1*. Springer Science + Business Media, LLC, Spring Street, New York, USA, 2008.

[20] H.Fletcher, B. Fpga Embedded Processors - Revealing True System Performance. Memec. Embedded Systems Conference San Francisco 2005.

[21] Howard, D. M., and Angus, J. A. S. *Acoustics and Psychoacoustics*, third ed. Focal Press, Jordan Hill, Oxford, UK, 2006.

[22] Ifeachor, E. C., and Jervis, B. W. *Digital Signal Processing - A Practical Approach*, second ed. Pearson Education Limited, Harlow, UK, 2002.

[23] Imsen, G. *Elevenes Verden - Innføring i Pedagogisk Psykologi*, fourth ed. Universitetsforlaget, Oslo, Norway, 2005.

[24] Imsen, G. *Lærerens Verden - Innføring i Generell Didaktikk*, third ed. Universitetsforlaget, Oslo, Norway, 2006.

[25] Jan Maes, M. V., and Baert, L. *Digital Audio Technology: a Guide to CD, MiniDisc, SACD, DVD(A), MP3 and DAT*, fourth ed. Focal Press, Jordan Hill, Oxford, UK, 2001.

[26] Jason G. Tong, I. D. L. A., and Khalid, M. A. S. Soft-core processors for embedded systems. In *The 18th International Conference on Microelectronics (ICM)* (2006), IEEE, pp. 170–173.

[27] Kondoz, A. M. *Digital Speech - Coding for Low Bit Rate Communication Systems*, second ed. John Wiley and Sons Ltd, Chichester, West Sussex, UK, 2004.

[28] Koren, I. *Computer Arithmetic Algorithm*, second ed. A. K. Peters, Natick, Massachusetts, USA, 2002.

[29] Larsen, J. Embedded demonstrator for audio manipulation. December 2009.

[30] Meyer-Baese, U. *Digital Signal Processing with Field Programmable Gate Arrays*, third ed. Springer-Verlag, Heidelberg, Germany, 2007.

[31] Michelsen, K. *C ♯ Primer Plus*. Sams Publishing, USA, 2001.

[32] Philips Semiconductors. *The I^2C-bus Specification, Version 2.1*, January 2000.

[33] Proakis, J. G., and Manolakis, D. G. *Digital Signal Processing*, fourth ed. Pearson Education, Inc, New Jersey, USA, 2007.

[34] R.C.Cofer, and Harding, B. *Rapid System Prototyping with FPGAs - Accelerating the Design Process*. Elsevier Inc., Burlington, Vermont, USA, 2006.

[35] Rose, J. Hard vs. soft: The central question of pre-fabricated silicon. In *34th International Symposium on Multiple-Valued Logic* (2004), IEEE, pp. 2 – 5.

[36] Rumsey, F., and McCormick, T. *Sound and Recording*, sixth ed. Focal Press, Jordan Hill, Oxford, UK, 2009.

[37] Walt Kester, A. D. I. *The Data Conversion Handbook*. Newnes, Burlington, USA, 2005.

[38] Watkinson, J. *The Art of Digital Audio*, third ed. Focal Press, Burlington, USA, 2001.

[39] Winder, S. *Analog and Digital Filter Design*, second ed. Elsevier Science, Woburn, Massachusetts, USA, 2002.

[40] Wolfson Microelectronics. *Master Clocks and Audio Sample Rates Supported by the Wolfson's Portable Audio ICs*, May 2002.

[41] Wolfson Microelectronics. *Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates, rev 4.0*, February 2005.

Appendix A

I^2C-controller - Details

The Verilog modules for configuration based on Altera's example design, as described in Section 8.2, are seen in Figure A.1. These modules are modified versions of those provided by Altera's example design, adjusted to support the functionality of the audio demonstrator. The "codec_config" module in the figure contains a set of configurations specified for the WM8731 audio-codec. This codec includes a number of registers, providing the designer with a wide range of possibilities for customization of the audio-codec for his or her design. For this implementation, four different configuration sets were made. These configurations relates to the operation of the ADC and DAC, and are described in more detail in Section 8.3.2. A configuration set consists of ten packages of 24 bits, where each packet contains a device address, a register address and the configuration data. The 8 MSBs are the address byte to the audio-codec. Each device connected to the I^2C bus has a unique address, making it possible for the transmitter to send the data packages to one specified receiver, in this case the audio-codec. The address bits for the device must be the MSBs of each packet transmission. The next 8 bits are the register address bits. These bits specifies which register in the audio-codec that is to be configured, making it possible to re-configure only parts of the codec's functionality. The 8 LSBs contains the data bits. It is these 8 bits that configures the audio-codec and determines the functionality of the device.

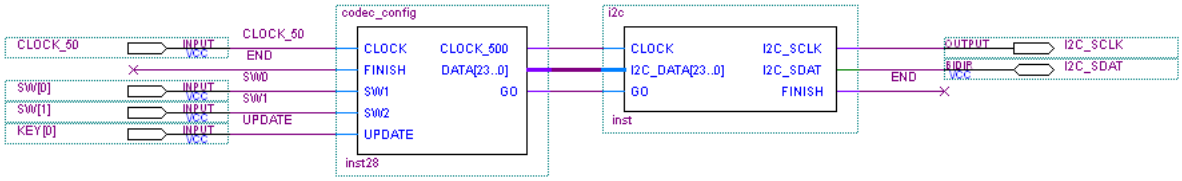

Figure A.1: Verilog modules for audio-codec configuration.

The "codec_config" module seen in Figure A.1 is connected to the user interface, and selects one of the four configuration sets that applies to the user's specification. This configuration set contains 10 x 24 bits, and provides information to the audio-codec about sample rates, volume level and input source. When selected, the configuration set that supports the user's selection is transmitted to the I^2C-module seen to the right in Figure A.1, 24 bits at the time. The I^2C module provides the functionality described in the I^2C standard, and transmits the data

package as 3 x 8 bits to the audio-codec. The transmission of these three bytes are separated by an acknowledge state controlled by the transmitter, where the receiver confirms that the packet is received. In the design provided by Altera, the transmitter holds its operation for one clock cycle after each packet of 8 bits is transmitted, waiting for the receiver to acknowledge the transmission. However, there is not implemented any functionality in the transmitter to handle the situation where the receiver fails to acknowledge the address or data byte. The I^2C module will just continue the transmission of the next byte after the mentioned acknowledge state, until both addresses and the data byte are transferred. The I^2C standard specifies that if an error occurs in the data transmission, the transmitter will either generate a stop condition to abort the transfer, or repeat the start condition to initialize a new transmission. In this case, if the audio-codec fails to acknowledge the transmission, it returns to an idle state, waiting for the I^2C controller on the FPGA to initialize a new data transfer. Since this controller not includes any functionality to support errors in the data transmission, it will just continue its transfer until all of the 24 bits are transferred, causing loss of configuration data. In the meantime, the audio-codec waits in its idle state, ready for transmission of a new data package.

When configuring the I^2C modules provided by Altera, adjusting them to support the functionality of the audio demonstrator, it was decided to accept the fact that the I^2C controller does not support the error handling stated in the I^2C standard. First of all, the Altera provided modules worked as expected in the demonstration design, configuring the audio-codec with different configurations without errors during testing. Secondly, although the I^2C controller does not support automatic initialization of the transmission after an error, manual re-configuration is supported, an operation which is considered to work well for the current system. Thus, although the I^2C controller could have been better at error handling, it was considered to use it as it was designed, saving time and effort by not changing a module that already worked.

Appendix B

Audio Input Buffer - Details

This appendix describes the functionality of the audio input buffer presented in Section 8.3.4. Here, a presentation of the signals and the data transmission is provided, together with the VHDL code written to implement the functionality.

As seen in Figure B.1, the input audio buffer receives BCLK, ADCDAT and ADCLRC from the digital audio interface. In addition, a signal from the hardware filter, called "ast_sink_ready", provides information to the input buffer on whether the FIR filter is ready to receive new audio data or not. Each time the audio-codec transmits a pulse on the ADCLRC line, as seen in Figure 6.7 on page 28, the input audio buffer prepares to receive the first data bit in the next clock cycle on BCLK. The input buffer is now in receive mode, receiving all the 48 bits of audio data and saving them internally. When the package is received, the data is sent to the FIR filter as 24 bit packages through the FIR_DATA[23..0] bus, whenever the filter is ready to receive.

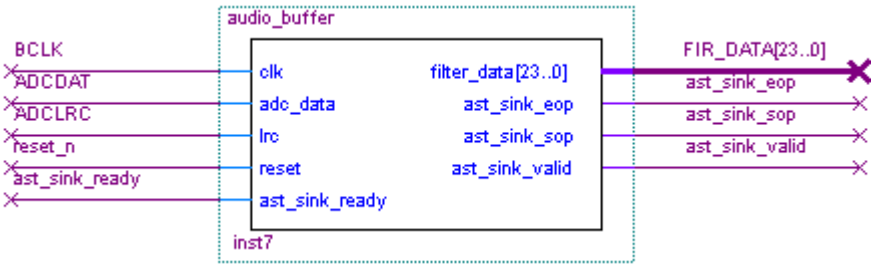

Figure B.1: Input audio buffer on FPGA.

The input buffer follows a communication protocol specified for the FIR filter. When data for the left channel is ready at the output of the input buffer, "ast_sink_valid" and "ast_sink_sop" are set to '1', telling the FIR filter that the data on the input is valid, and that these 24 bits represents the start of the package. In the next clock cycle on BCLK, "ast_sink_valid" is unchanged, while "ast_sink_sop" is set to '0'. Since the data package in the transmission only contains two parts, namely the left and the right channel sample, the next 24 bits represents the last part of the package. The input buffer indicates this by setting "ast_sink_eop" to '1', telling the FIR filter that this is the end of the data packet. The FIR filter reads the valid

data on the input, before "ast_sink_valid" and "ast_sink_eop" returns to '0'. The input audio buffer is now ready to receive a new data package from the ADC, and waits in an idle state until a new transmission is initialized.

B.1 VHDL Implementation of Input Buffer

```
1  ----------------------------------------------------------------------------
2  -- Audio-buffer. Receives seriel data from ADC, 48 bits including both --
3  -- channels, and transmitts 2x24 bit packages, left channel            --
4  -- first, in parallel at the output.                                   --
5  ----------------------------------------------------------------------------
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9
10 entity audio_buffer is
11 port(
12         clk,adc_data,lrc,reset,ast_sink_ready : in std_logic;
13         filter_data : out std_logic_vector (23 downto 0);
14         ast_sink_eop,ast_sink_sop,ast_sink_valid : out std_logic
15 );
16 end entity audio_buffer;
17
18 architecture structure of audio_buffer is
19 type state is (IDLE,RECEIVE,TRANSFER_LEFT,TRANSFER_RIGHT);
20 signal next_state : state;
21 signal audio_data : std_logic_vector(47 downto 0); --Audio-data storage
22 signal counter : integer range 47 downto 0 := 47;
23 begin
24         FSM : process(clk,reset)
25         begin
26         --Initial configuration of buffer
27           if(reset = '0') then
28             next_state <= IDLE;
29             filter_data <= (OTHERS => '0');
30             audio_data <= (OTHERS => '0');
31             ast_sink_valid <= '0';
32             ast_sink_eop <= '0';
33             ast_sink_sop <= '0';
34
35                 elsif(rising_edge(clk)) then
36                 case next_state is
37                 --IDLE state
38                   when IDLE =>
39                     if(lrc = '1') then --Codec transfers data from ADC
40                         ast_sink_eop <= '0';
41                         ast_sink_sop <= '0';
42                         ast_sink_valid <= '0';
43                         next_state <= RECEIVE;
44                     elsif(lrc = '0') then
45                         ast_sink_eop <= '0';
46                         ast_sink_sop <= '0';
47                         ast_sink_valid <= '0';
48                         next_state <= IDLE;
49                     end if;
```

```
50
51                  --Receive data from both channels, left channel first
52                    when RECEIVE =>
53                      if(counter > 0) then
54                        audio_data(counter) <= adc_data; --Saves audio data
55                        counter <= counter - 1;
56                        next_state <= RECEIVE;
57                      else
58                        audio_data(counter) <= adc_data;
59                        counter <= 47;
60                        next_state <= TRANSFER_LEFT;
61                      end if;
62
63                  --Puts left channel data on the parallel output
64                    when TRANSFER_LEFT =>
65                      if(ast_sink_ready = '1') then --FIR filter ready for data
66                        ast_sink_sop <= '1';        --Start of data package
67                        filter_data(23 downto 0) <= audio_data(47 downto 24);
68                        ast_sink_valid <= '1';
69                        next_state <= TRANSFER_RIGHT;
70                      else --Wait until FIR filter is ready to accept data
71                        next_state <= TRANSFER_LEFT;
72                      end if;
73
74                  --Puts right channel data on the parallel output
75                    when TRANSFER_RIGHT =>
76                      ast_sink_sop <= '0';
77                      ast_sink_eop <= '1'; --End of data package
78                      filter_data(23 downto 0) <= audio_data(23 downto 0);
79                      next_state <= IDLE;
80                  end case;
81                end if;
82          end process;
83  end architecture;
```

Listing B.1: Audio input buffer

Appendix C

Audio Output Buffer - Details

In this appendix, the audio output buffer is presented. A description of its functionality is provided, together with the VHDL code that describes the buffers functionality on the FPGA.

The output buffer is seen in Figure C.1. When the filter that is active is ready with its signal processing on a data package, the filter module sets the "ast_source_valid" input on the audio buffer high. This tells the audio buffer to save the current and the next data package at the input, containing a filtered sample for each of the two audio channels. When this is done, the audio buffer waits for a pulse on the "DACLRC" input before it transforms the two stored audio samples to a serial bit stream of 48 bits. This data stream is then sent to the DAC in the audio-codec through the "DACDAT" channel, ready for conversion to the analog domain.

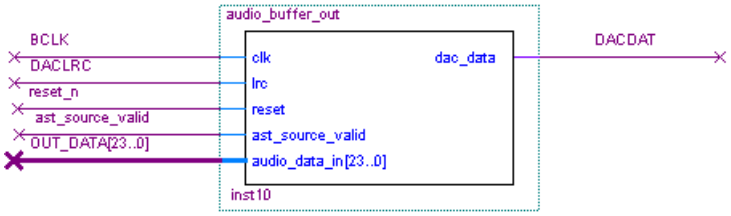

Figure C.1: Output audio buffer on the FPGA.

C.1 VHDL Implementation of Output Buffer

```
1  ------------------------------------------------------------------------
2  -- Audio output buffer. Receives 2 x 24 bit parallel data packages --
3  -- from filter, and puts 48 bits of serial data to the output when --
4  -- audio-codec is ready to receive.                                  --
5  ------------------------------------------------------------------------
6
7  library ieee;
8  use ieee.std_logic_1164.all;
9
10 entity audio_buffer_out is
11 port(
```

```
12          clk,lrc,reset,ast_source_valid : in std_logic;
13          audio_data_in : in std_logic_vector(23 downto 0);
14          dac_data : out std_logic --Output to the audio-codec
15  );
16  end entity audio_buffer_out;
17
18  architecture structure of audio_buffer_out is
19  type state is (IDLE,RECEIVE_RIGHT,START_TRANSFER,CONT_TRANSFER);
20  signal next_state : state;
21  signal audio_data : std_logic_vector(47 downto 0);
22  signal counter : integer range 47 downto 0 := 47;
23  begin
24
25          FSM : process(clk,reset)
26          begin
27          --Initial configuration of output buffer
28            if(reset = '0') then
29              next_state <= IDLE;
30              dac_data <= '0';
31              audio_data(47 downto 0) <= (OTHERS => '0');
32
33            elsif(rising_edge(clk)) then
34              case next_state is
35              --IDLE state
36            when IDLE =>
37          --Save data for left channel on the 24 MSBs
38              audio_data(47 downto 24) <= audio_data_in(23 downto 0);
39              if(ast_source_valid = '1') then --If data from FIR is ready
40                dac_data <= '0';
41                next_state <= RECEIVE_RIGHT;
42              else
43                dac_data <= '0';
44                next_state <= IDLE;
45              end if;
46
47          --Receive and save data (24 bit) for right channel
48            when RECEIVE_RIGHT =>
49              audio_data(23 downto 0) <= audio_data_in(23 downto 0);
50              next_state <= START_TRANSFER;
51
52          --Transmits the MSB to audio-codec when codec is ready
53            when START_TRANSFER =>
54              if(lrc = '1') then --When DAC is ready
55                dac_data <= audio_data(counter); --MSB left channel
56                counter <= counter - 1;
57                next_state <= CONT_TRANSFER;
58              elsif(lrc = '0') then
59                next_state <= START_TRANSFER;
60              end if;
61
62          --Sends the rest of the audio data to the codec in serial
63            when CONT_TRANSFER =>
64              if(counter > 0) then
65                dac_data <= audio_data(counter);
66                counter <= counter - 1;
67                next_state <= CONT_TRANSFER;
68              else
```

```
69                dac_data <= audio_data(counter);
70                counter <= 47;
71                next_state <= IDLE;
72              end if;
73              end case;
74            end if;
75          end process;
76  end architecture;
```

Listing C.1: Audio output buffer

Embedded Demonstrator for Audio Manipulation

Jarle Larsen
Dep. of Electronics and Telecom.
NTNU, Trondheim

Per Gunnar Kjeldsberg (supervisor)
Dep. of Electronics and Telecom.
NTNU, Trondheim

The embedded audio demonstrator, seen in Figure 1, demonstrates the effects of sampling and filtering through manipulation of analog audio signals on Altera's DE2 board.

MOTIVATION

Sampling of audio signals is the process of converting the analog music to a digital representation for manipulation, transmission or storage in the digital domain [1]. On standard CD discs, the sample rate is 44,1 kHz, meaning that the analog audio signal is sampled every 1/44100 second. The Nyquist theorem states that the sample rate must be at least twice as high as the highest frequency in the sampled signal in order recover the analog signal completely from the digital domain. The audio demonstrator provides an adjustable sample rate with rates both above and below the Nyquist frequency, making it possible to study how the quality of the perceived sound changes with the sample rate.

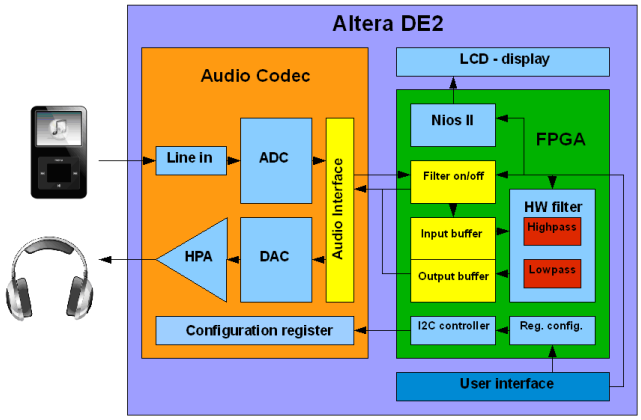

Figure 1: Altera DE2 development board

Digital filters are present in many digital audio systems today, and provides equalizers and sound manipulation in MP3-players, amplifiers and TVs [1]. In order to demonstrate how filtering affects the sound signal, the audio demonstrator is designed with both high-pass- and low-pass filters, in addition to a user interface for filter characteristic control.

DEMONSTRATION

Figure 2 presents the signal flow in the audio demonstrator, and illustrates how the signal changes through the system.

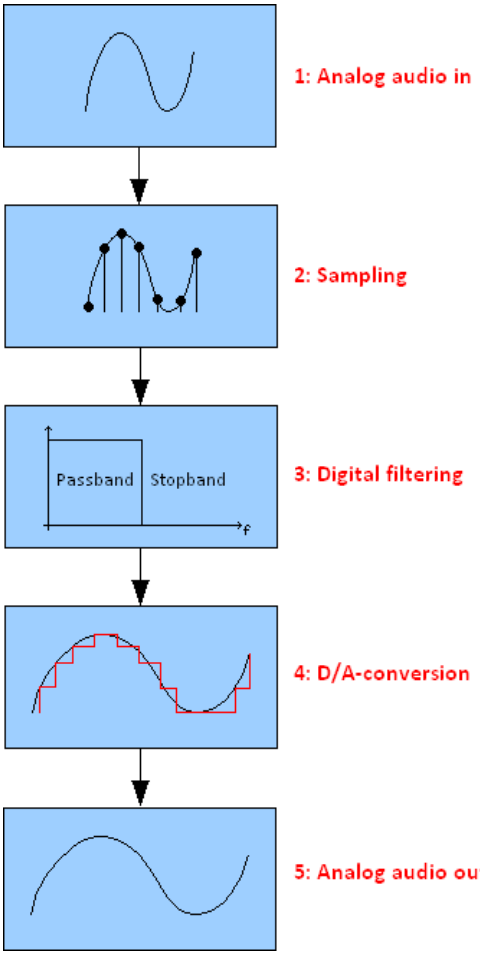

Figure 2: Signal flow in the audio demonstrator

REFERENCES

[1] – Larsen, J. *Embedded Demonstrator for Audio Manipulation,* 2010. Unpublished.

Demonstration poster, spring 2010
NTNU, Department of Electronics and Telecommunication