Home Institution (Greece):

Technological Educational Institute of Messolonghi
Faculty of Applied Technology
Department of
Telecommunication Systems and Networks

“Design and development of a low-cost implementation in the area ‘Smart Home’ using speech recognition.”

FUSCHELBERGER David
PYROUNAKIS Ioannis

Final report submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science
in Telecommunication and Network Engineering

July 2010

Supervisor
Carlos Ribeiro, Ph.D (IPL, Portugal)

Host Institution (Portugal):

IPL
instituto politécnico de leiria

Polytechnic Institute of Leiria
Faculty of Technology and Management
Department of
Electrical Engineering

Acknowledgements

First of all we would like to express our thankfulness to our supervisor Mr. Carlos Ribeiro, who assisted us throughout our project with his overall knowledge and his encouragement. His interest in this project was of great significance.

We would also like to thank our professor Mr. Evangelinos Mariatos, who gave us the initial idea for our project and, due to his course (Real Time Systems) which we had attended, had given us some basic understanding of several issues regarding the project.

Special thanks to Mr. Nuno Fonseca, whose course (Voice Coding and Transmission), which we were attending at IPL during the period of our project, perfectly furthered our knowledge regarding audio issues. We took full advantage of his great experience in audio/voice technologies.

We also need to thank our coordinators Mr. Nikolaos Voros and Mr. Anastasios Ntagiuklas for the realization of this Socrates/Erasmus-programme exchange.

In addition to these personal acknowledgements, we feel the need to show our gratitude to the Polytechnic Institute of Leiria, and specifically to the School of Technology and Management and the department of Electrical Engineering, for the great hospitality and the full support throughout our stay in Portugal.

Last but not least, we kindly dedicate this project to our parents, who are supporting our education in every way throughout all these years.

Leiria, July 2010

Giannis Pyrounakis
David Fuschelberger

Abstract

The ultimate goal of this project is the implementation of a device which will properly capture, process and transmit several voice commands to a central PC over a local Ethernet network. The PC, receiving these commands, will be able to control several devices using a purchased relay board and appropriately developed software including speech recognition.

In summary, this project consists of three main parts. The first part mostly deals with analog electronic, including the research on appropriate components required for the implementation of the analog circuit, such as a microphone, a pre-amplifier, an Analog-to-Digital Converter (ADC), etc. and their proper interconnection. This part also involves several techniques regarding noise cancelling, as this is of great significance in our system, and the design of the Printed Circuit Board (PCB) daughterboard.

The second part, mostly digital electronic, has as its main component a Field Programmable Gate Array (FPGA) board with a Cyclone chip of Altera. The PCB daughterboard will be interfaced to the FPGA board, whose chip will be programmed using Very-high-speed integrated circuit Hardware Description Language (VHDL). The voice samples, serially received from the analog circuit, will be digitally filtered, encapsulated into strictly structured UDP/IP packets and sent to the PC over a 10Base-T Ethernet connection.

The third part involves the development of the appropriate software to receive and recognize the commands, and finally drive the relay board as a result of this end-to-end communication. We are using Microsoft's C# Sharp, .NET Framework and the Speech Application Programming Interface (SAPI).

Table of Contents

Abbreviations/Acronyms

ADC	Analog-to-Digital Converter
ARP	Address Resolution Protocol
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DAC	Digital-to-Analog Converter
DHCP	Dynamic Host Configuration Protocol
DLL	Data Link Layer
DSP	Digital Signal Processing/or
EIB	European Installation Bus
ECM	Electret Condenser Microphone
ENOB	Effective Number of Bits
FCS	Frame Check Sequence
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
GND	Ground
GUI	Graphical User Interface
HDL	Hardware Description Language
HVAC	Heating – Ventilation – Air Conditioning
I2C	Inter Integrated Circuit
IC	Integrated Circuit
IIR	Infinite Impulse Response
IFG	Inter-Frame Gap
I/O	Input/Output
LAN	Local Area Network
LLC	Logical Link Control
LSB	Least/Less Significant Bit(s)
LTP	Link Test Pulse
LVTTL	Low Voltage Transistor-Transistor Level
MAC	Medium Access Control
MSB	Most Significant Bit(s)
MSPS	Mega Samples per Second
NIC	Network Interface Controller

NLP	Normal Link Pulse
OP-AMP	Operational Amplifier
OSI	Open Systems Interconnection
OSR	Over-Sampling Rate
PCB	Printed Circuit Board
PDU	Protocol Data Unit
PGA	Programmable Gain Amplifiers
PHY	Physical (Layer)
PLL	Phase-Locked Loop
PoE	Power over Ethernet
PPI	Parallel Peripheral Interface
PPS	Packets per Second
PROM	Programmable Read-Only Memory
RTL	Register Transfer Level
SAPI	Speech Application Programming Interface
SAR	Successive Approximation Register
SMD	Surface Mounded Devices
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
TOS	Types of Services
TRX	Transceiver
TTL	Time to Live
UDP	User Datagram Protocol
UTP	Unshielded Twisted Pair
VAD	Voice Activity Detector
VHDL	Very-high-speed integrated circuit Hardware Description Language
VLAN	Virtual Local Area Network

List of Figures

List of Tables

1. Introduction

This project and therefore its report refer to a wide range of technical issues. In order the introduction of the report to be comprehensive, but also to be providing meaningful information about the project in overall, it is divided into three sections. In the first section, we will present a small introduction of the area this project belongs to, home automation. In Section 1.2, we will refer to the concept of the project and our basic initial ideas. Finally, we will describe the practical flow of the design that we followed, as well as several technical issues we were concerned about.

1.1 Introduction to the term 'Smart Home'

'Smart Home' is a term mainly used to describe the area of home automation, an attempt of reliable automated control systems for household appliances. These systems may be divided into several directions as you will notice later in this report. Some are full automated systems not requiring any human interruption; others need human control, which many different technologies have been used for. Home automation, also called 'domotics', became popular in the past few years, as the price of digital electronics fell dramatically and implementations became actually worth it to be designed and put on the market, although the research in this area had began much earlier. There have been many products on the market so far, and standards of hardware being used for these designs have been defined throughout the years. Some of these standards, such as *Insteon* and the previous *x10*, have been successfully used and become popular.

Three architectures have been defined to describe home automation systems:

1) ***Centralized***: A central controller, such as a PC, is aggregating data from the sensors to control the actuators.
2) ***Distributed*** (decentralized): Each device, sensor or actuator, has its own intelligence, having the ability of processing data and controlling other devices.
3) ***Mixed***: A decentralized architecture, in which some devices have further abilities than others.

Home automation has several tasks which have been or are about to be covered. The most important of these tasks in general are listed below:

1) Heating, Ventilation and Air-Conditioning (HVAC)
2) Lighting
3) Audio/Video
4) Security

Several technologies are used for designs in home automation, starting with the physical medium for the communication of the devices through to protocols developed for devices to interact with each other. Considering the medium, a wired network would require initial design during the construction of a building. Along with traditional wiring of a house, such as electrical power and telephony/TV, a Local Area Network (LAN) is nowadays common to be designed though. However, due to the evolution of technology, wireless solutions (e.g. Wi-Fi or Bluetooth) become more popular for digital systems, although the use of such solutions is always a price issue, as newer technology has higher costs of implementation. Nevertheless, wireless solutions are not replacing the regular wiring for electricity anyway. In Table 1.1 you can see some technologies used in this area.

Technology	**Physical Medium**	**Speed**	**Distance**
Ethernet (IEEE 802.3)	Unshielded Twisted Pair Optical Fibre	10 Mbps - 1 Gbps	100 m
Wi-Fi (IEEE 802.11)	Radio Frequency	11 Mbps – 248 Mbps	30 m – 100 m
Bluetooth (IEEE 802.15.1)	Radio Frequency	1 Mbps – 10 Mbps	10 m – 100 m
ZigBee (IEEE 802.15.4)	Radio Frequency	20 kbps – 250 kbps	10 m – 1500 m

Table 1.1 - Technologies used in home automation

Figure 1.1 shows a typical home automation system, not too complex in its nature, involving several tasks and technologies.

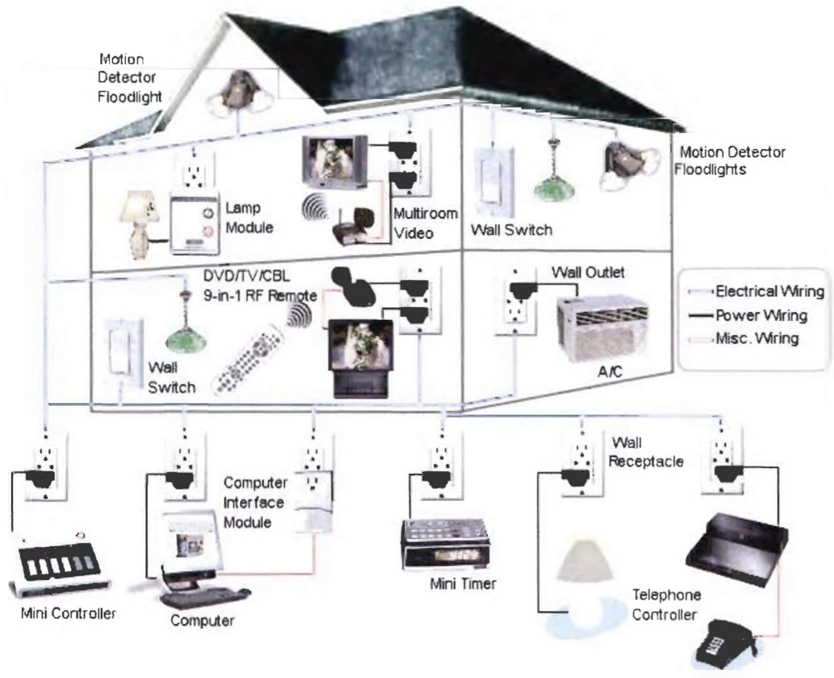


Figure 1.1 – Example of a home automation system

This example shows the wiring of the house, and several tasks executed with different technologies (e.g. remote control for the home entertainment system or motion detection for the outside floodlight). Regarding the functionality of our project, some of these could be combined and easily done, such as the lighting (no need to operate the switch). More details regarding such tasks are mentioned in section 1.2.

Regarding home automation systems, there is one main difference to be considered. There are many ways to control a house and its devices – automatically, remotely, etc. For example, security tasks are commonly controlled automatically: motion detection or other kinds of sensors are used to identify intrusion or similar security threats and usually some kind of alarm message is sent to the owner in several ways, such as a mail or short message. The lighting could be controlled automatically, using a light sensor to identify whether daylight is sufficient or not, or also (as widely used in the past few years), a motion detector for the occasional lighting of specific areas. Also, due to the ease of today's internet, some tasks could be easily done remotely using an internet connection and a server at home. However, this refers to a different area of interest. Generally, some tasks have been covered because of the need for reliability of some specific issues (e.g. security), and others because of the need for a more comfortable way of living in our own environment. This last category of living comfort is the area of interest of this project, as it is detailed later in section 1.2.

We should outline one special category regarding the above mentioned difference. Both necessary and desired tasks may be combined into one helping theme for elders and people with disabilities. Specifically, the way of operating several devices without the need of movement makes this area significantly interesting.

Today, one remarkable system to be mentioned is *Instabus* (also known as EIB – European Installation Bus), a decentralized system developed by *Siemens AG* which is used and provided by many companies around the world. *Instabus* is an example of using the same medium for both digital control and power supply of household devices. The term 'Smart Home' itself is also used by a company, providing home automation solutions, as a title (*www.smarthome.com*) - a company of *SmartLabs, Inc.*, the creator of *Insteon*. We used the term 'Smart Home' rather than home automation itself because of its clever way of indirectly expressing the whole meaning in two words. For further details regarding home automation please refer to [1].

1.2 Concept of the project

This project is considered to be categorized as an implementation in the area of home automation. Its architecture is *centralized*, as the main intelligence is located at a central PC. The idea is to design a stand-alone device which will act as a sensor, capturing voice commands of the house owner and sending these commands to the PC (actuator) to operate the central switchboard of the house to control several home devices. Other tasks may be considered to be done straight on the PC, regarding operations related to the PC itself. The PC has the great task of recognizing what it is told to do, but fortunately Microsoft gives great solutions in this area. We will present an overview of this part in more detail later in chapter 4 – the software implementation. Considering the wiring mentioned in section 1.1, we had made our thoughts about this and came up with the idea to at least supply our device with power over Ethernet (since we use a local network rather than a wireless solution), to discard the use of an annoying adapter. However, this does not replace the idea of a regular electricity network with a central switchboard, from which the home devices will be operated. Regarding the cost of such an implementation, our goal is to show that a simple design could be as efficient as an expensive solution on today's market.

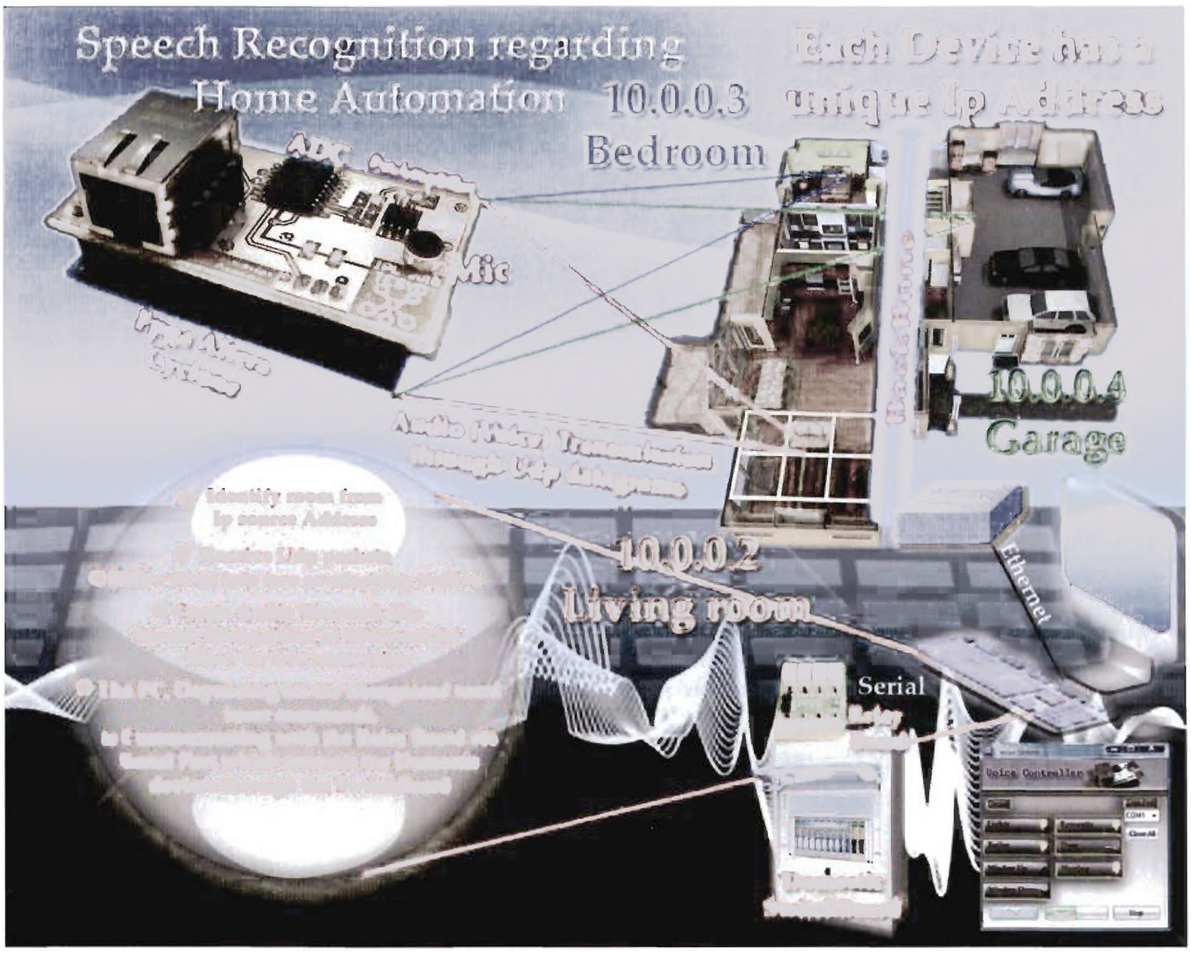


Figure 1.2 – General view of the system and its functionality

Beside the fact that in our days many household appliances are controlled automatically, there has always been the need to give people the feeling of control and comfort. It would be easier not to operate each single switch to switch on or off the lights, to open or close the shutters or to go and check if the boiler is on. One nice way of having the control over your house is to communicate with it, such as with another person – talking to it! You may have an automated air-condition or heater to provide you with an average temperature, but some times you might get colder or hotter than normally and you would wish to turn the heating/cooling higher or lower – why not doing that from the couch or the bed? It would be comfortable to turn on your stereo by giving a simple speech command, or to get sure that your doors are locked by just saying it out loud. However, there are many things which could be done in this way; a person's creativity is the key to find interesting tasks for this kind of systems. Not to forget, as also mentioned before in section 1.1, beside the factor of living comfort generally, there is the need to make elders' and disabled peoples' lives much easier by helping them in preventing aimless movement.

One simple question regarding this project might be, why not netting the house with directly connected microphones? The answer is that using a digital system, like the one described in this report, has many advantages. Specifically in our case, the identification of which room the command came from, is a great benefit. Using an OSI layer-based network with IP intelligence makes that easy to be achieved. Also, a digital network provides reliable transmission as well as higher immunity to noise, which improves the communication and results in good functionality. But there are many reasons to use an organized digital system rather than an amateur solution.

Our system has some initial requirements to be considered, regarding some already existing equipment. This includes an Ethernet LAN, a PC running MS Windows (XP or later), a well-organized electricity switchboard including the control of individual devices such as the lights or windows of a single room of the house, and user voice profiles created on the PC.

Some steps of this project have been simplified for several reasons, mostly because of the needs of actually being an educational project. First thing to mention is that some parts that we used were purchased. In Table 1.4 you can see the purchased equipment used for the realization of our project.

1) *Pluto-II* FPGA Board from *KNJN LLC* with an Altera Cyclone EP1C3T100 chip.
2) *8 relay board* from *KMtronic*

We chose this FPGA board for several reasons. It has an implemented serial programmer to easily program the chip's Boot-PROM. The board itself is small and fits perfectly the size we had thought our device to be and in addition it was not too expensive. Concerning a final product of this project, and considering an actual low-cost implementation, it would be achievable to add only the Cyclone chip to our circuit, implementing a self-created JTAG programmer, to significantly reduce the cost of the design.

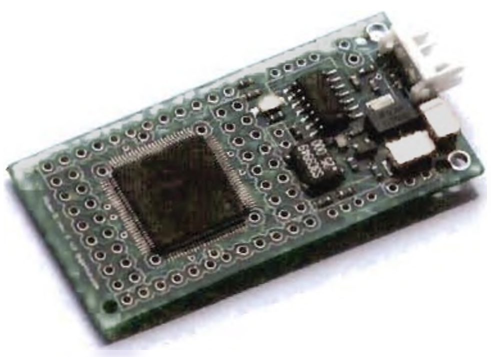


Figure 1.3 – Pluto-II FPGA Board

The reason for buying the relay board is mostly to show the results of our system in practice. The thought is this board to be placed at the switchboard, connecting the relays to several devices of the house. This board might eventually consist of more relays. Although the results could be shown only on the PC, or several tasks may not need this additional board (such as sending an alert message via mail or starting some software on the PC), we thought that this is would be a nice way to show the functionality of our project.

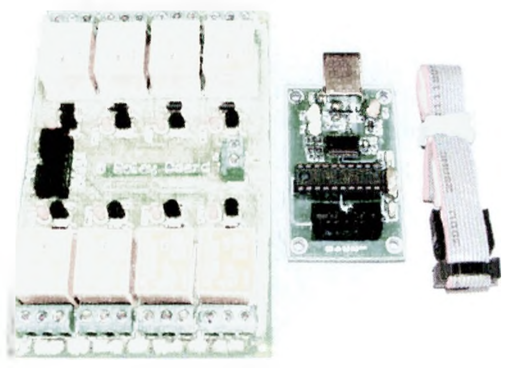


Figure 1.4 – Relay Board

The next issue needed to be mentioned is that we are actually implementing a one-way communication. We are using a 10Base-T Ethernet transmitter-only instead of a full Transceiver (TRX), implemented on the FPGA, which means simplex-communication. This issue results in a simplified network implementation, with loss of some basic operations normally used in Ethernet standard. We will refer to this issue in more detail in chapter 3, the digital part of our project (FPGA implementation). In addition to this implementation, we have to mention that we did not use an external PHY chip for the design of the transmitter. The most important reason for this decision was the price issue, as the purchase of such a chip would significantly raise the cost of our application.

One note to be made at this point is that we thought of having the ability for further expansion of our project, so that we also implemented a receiving part, which could be used for other operations. We will concentrate on this issue later in this report, as it has also to do with the lack of Ethernet operations mentioned above.

Regarding the tasks executed by our system we thought of the following:

1) Switching on/off the lights of an individual room.
2) Switching on/off the dimmer for romantic lights in an individual room, turning on/off the normal lights if required.
3) Opening/closing the shutter of an individual room, having the ability to stop it at the required height.
4) Locking/unlocking the door of an individual room
5) Turning on/off the boiler.
6) Turning on/off the heating.

We have the ability to combine these commands (e.g. switching on/off all of the lights of the house or locking/unlocking all the doors) if necessary. Many tasks could be thought to be executed straight on the PC (e.g. turning on the music or some other software). Additionally, we implemented a response to each command in form of spoken words produced and played at the PC. As a result of this the user gets an answer as a confirmation that his task was successfully executed. This is obviously an important issue, as it makes the interaction of the owner and the house a two-way communication.

Regarding this report there are some notes to be made. Besides the three main parts of implementation, we included one more chapter where we present tests and simulations which were done and achieved throughout the project. That means that this part is not described individually for every step in each of the next three chapters which follow. Also, due to the project's huge variety in technologies, this report may at some points not be completely detailed. In order not to exceed the size of a normal report, we tried to make a high-level description of each part.

1.3 Practical overview of the project

Due to the complexity of this project, as it covers an end-to-end system including low and high level design, a step-by-step progress was the only way to achieve a successful final result. It could generally be separated into three main parts; the analog, digital and software part. Figure 1.5 shows a general Block diagram of the project's 3 main parts.

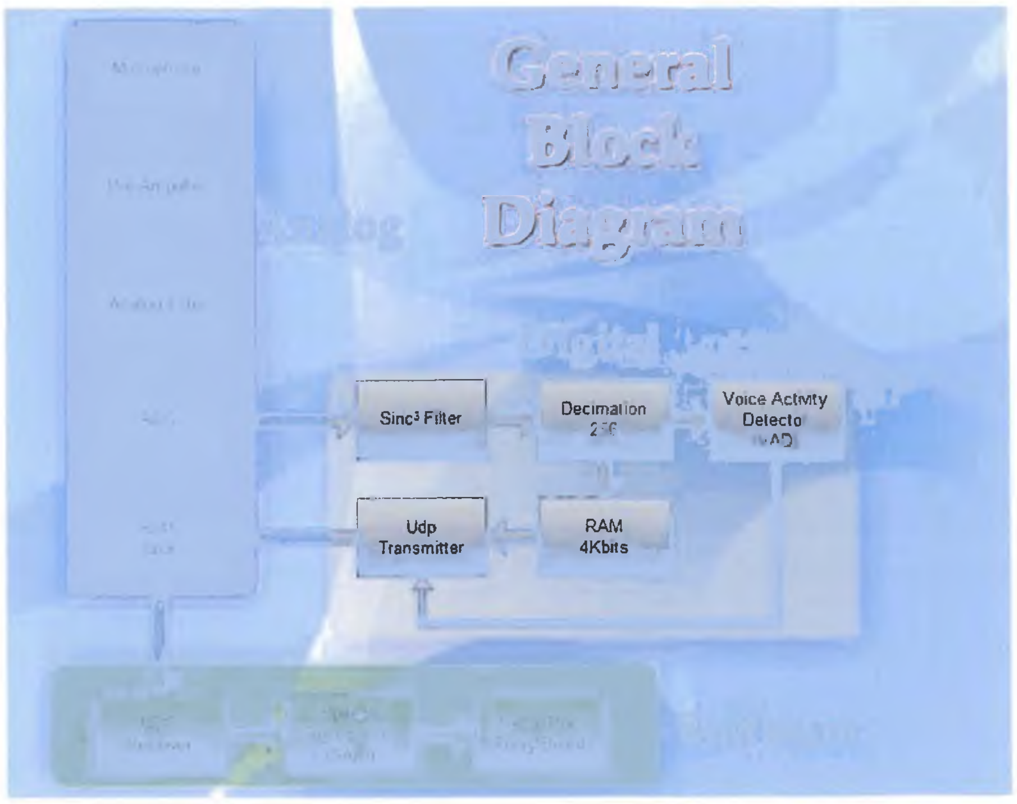


Figure 1.5 – Block diagram of the project

The three main blocks can be clearly seen. The analog part mainly consists of several components connected with each other, such as a microphone, a pre-amplifier and an Analog-to-Digital Converter (ADC), plus an analog filter circuitry and a RJ45 Ethernet jack. The digital part includes a digital $Sinc^3$ filter plus decimation process and the Ethernet transmitter, which gets the voice samples from the RAM block and is controlled by the Voice Activity Detector (VAD). Finally, the software part consists of a UDP receiver, the Speech Recognition block and the serial port communication, which the relay board is driven through.

We followed a specific step-by-step design flow, which is shown below in Figure 1.6.

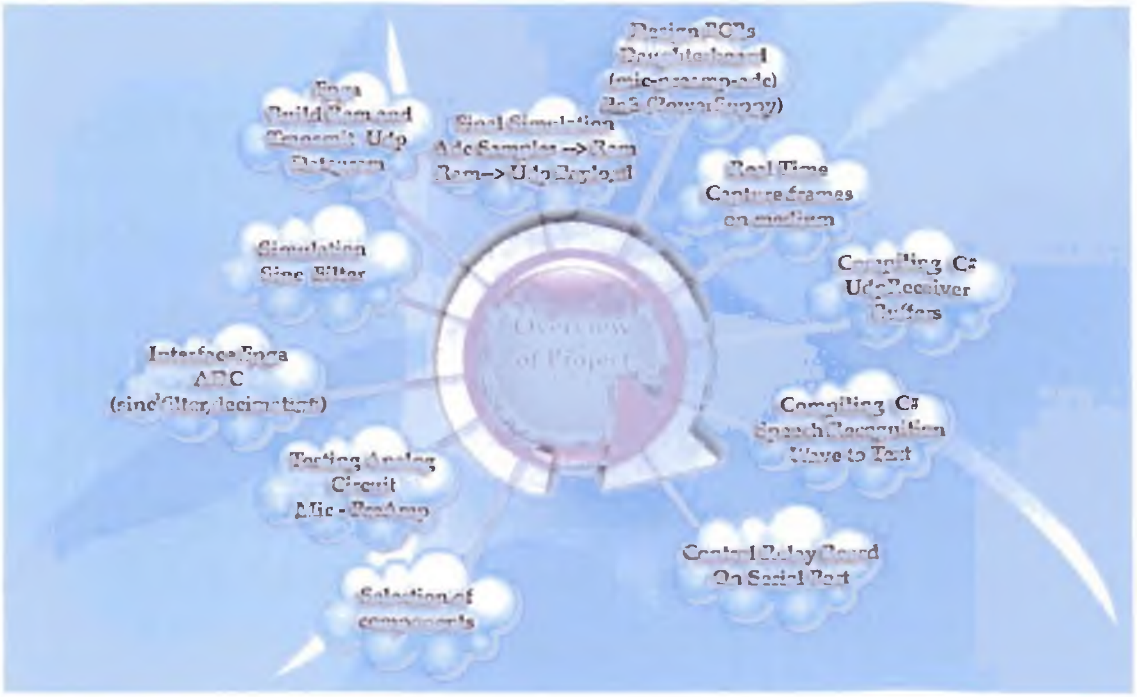


Figure 1.6 – Design flow overview

The first step was to design and simulate a circuit to proper capture and digitize the analog signal. This involved the choice of appropriate components used for the circuit, as well as their interconnection and fulfilment of their specifications and requirements. An analog low-pass filter had to be designed and tested, and several issues regarding analog electronic noise had to be considered. All these are combined and result in one final Printed Circuit Board (PCB), which will be connected to the FPGA board. In detail, after choosing the components, we had to deal with the amplifying and filtering of the signal. We tried many filter topologies, combined with the gain of amplification as well as DC filtering and the background noise level.

After achieving a well-working circuit, due the use of a Sigma-Delta modulator (described in chapter 2), we had to interface the digitized serial bit stream to the FPGA and to build a digital $Sinc^3$ plus decimation filter. The filter was tested and simulated in several ways. At this point we got the final voice samples at their correct bit rate and sampling frequency. After that, the final samples had to be buffered in a Ram block, where they are taken from by the Finite-State Machine (FSM) to be encapsulated into the packets. The UDP packets, their framing and the operation of several functions (e.g. Cyclic Redundancy Check – CRC) were also done at the transmitter block. We simulated the proper writing of the audio samples into the Ram, as well as their encapsulation into the payload of the structured packet.

The next step was to design the PCB, including considerations of several details such as a separated analog and digital ground plane regarding electronic noise, the connectors to interface the FPGA board, voltage regulators and other components, and additional circuitry required, such as for the Ethernet jack and the ADC. We also thought of Power over Ethernet (PoE) to be a nice solution as the power supply of the device, so that we also designed and built a separate circuit for this purpose. At this point, after connecting the PCB to the FPGA and supply it with PoE, we were able to capture the transmitted packets on the medium, with a digital oscilloscope, which helped us to confirm the proper Ethernet transmission.

The final step was done at the PC, having much more flexibility due to the object-oriented programming language that we used and the general conveniences provided by Microsoft and its solutions. After receiving the packets at the Ethernet NIC of the PC, the first thing to be done was to get access to them through a UDP receiver. After extracting the payload, namely the raw voice samples, a memory stream is created out of a buffer and inputted into the Speech Recognition Engine. This is where the commands are recognized and mapped to the pre-programmed tasks, which are then finally executed. As already mentioned, this final step was of great significance. The success of our project notably depended on this high-level design part, as it involved the highest complexity and intelligence of our system.

2. Analog Part of the Design

The analog part of our project consisted of several steps to be fulfilled. Most of them required detailed research plus some tests and simulations (which are described among the rest of simulations in chapter 5) in order to proceed to the next part of our project. After choosing the components we did a research on analog active low-pass filters, general Printed Circuit Board (PCB) design techniques and noise issues regarding the circuit. We have also included a self-created Power over Ethernet (PoE) device and an Rx board for eventual duplex communication of the device in the future. In this chapter we present a detailed description of each individual step that we followed regarding these above mentioned analog issues of our design.

2.1 Research on appropriate components

The fact that the signal is captured in a noisy environment and that an electronic design itself has normally many noise issues too, makes the choice of appropriate components one of several critical issues. In the following sub-chapters we will describe the research we made and the reasons for choosing the specific components that we finally used for our design.

2.1.1 Microphone

Regarding the choice of an appropriate microphone, we first had to decide what kind to choose. Our application requires capturing of voice coming from many directions, so that we needed an omni-directional microphone. In implementations similar to ours, the most widely used type of microphone is ECM, Electret Condenser Microphone. They can be found in many common products known on today's market, such as laptops, cell phones and more. ECMs have a good response in both high and low frequencies as well as low input noise levels. Another kind of microphone is called *piezoelectric*, which is limited in frequency response and has high output impedance. [2]

Figure 2.1a – ECM

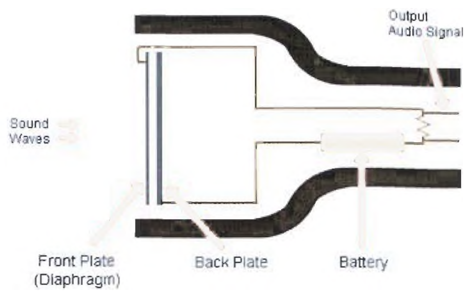


Figure 2.1b – ECM Schematic

2.1.2 Pre-Amplifier

In order to get a usable waveform we needed to amplify the captured signal, as its amplitude after being captured is minimal. After taking a closer look at the amplifiers available on the market, we came up with some and compared them. This comparison is shown in Table 2.1. Several features characterize the types of amplifiers: for instance, operational amplifiers have high DC-coupled gain while instrumentation amplifiers a low DC offset. Both types are very similar to each other, though. However, the most important factor for us to consider was the noise level of each of them.

	Type	Input Noise
NE5532	Op-Amp	8nV
AD622	Instrumentation	12nV
TL072	Op-Amp	18nV

Table 2.1 – Pre-amplifiers

Comparing the noise levels of these specific amplifiers it is clear that our choice needed to be the NE5532 Op-Amp. Besides the minimal input noise, NE5532 is widely used in similar applications [3]. Figure 2.2a shows the NE5532 as Surface Mounted Device (SMD) and Figure 2.2b shows its schematic, where it can be clearly seen that NE5532 is a dual Op-Amp (two amplifiers in one package). However, we will not need the second amplifier.

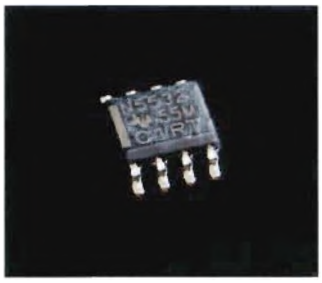


Figure 2.2a – NE5532 Op-Amp

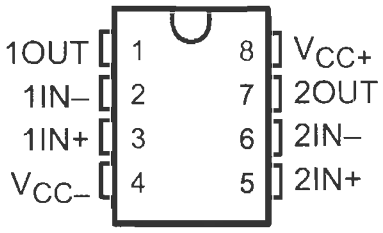


Figure 2.2b – NE5532 Schematic

2.1.3 ADC

The choice of an appropriate ADC demanded a lot of research, especially on the architectures of ADCs and their features. There are three main architectures defined:

1) SAR – Successive Approximation Register
2) Σ-Δ (Sigma-Delta)
3) Pipelined

Successive Approximation Register ADCs are mostly used in data acquisition applications, and generally in a wide range of common applications. They got popular due to their general ease of use and easy data input multiplexing. Their resolution varies up to 18 bit and their sampling rates up to 3 MSPS (Mega Samples per Second).

Pipelined ADCs are high-speed ADCs used in applications which commonly require a sampling rate approximately greater than 5 MSPS. They have a resolution up to 14 bits and offer very high Signal-to-Noise Ratios (SNR). The application set, which pipelined ADCs are used in, is commonly divided into instrumentation applications (digital oscilloscopes, spectrum analyzers, and medical imaging), video/radar communication applications (e.g. IF sampling) and consumer electronics (digital cameras, DVD etc).

Sigma - Delta or Delta - Sigma ADCs have currently two wide areas of usage: industrial measurement applications and voice-band applications. Their high resolution, 16 to 24 bits, often allows, in combination with on-chip PGAs (Programmable Gain Amplifiers), the direct connection of a sensor to the ADC without the need of an instrumentation amplifier. However, we used a traditional Op-Amp – ADC approach for our application. Sigma-Delta ADCs have some additional features such as *oversampling* and *noise shaping*, which help achieving high SNR levels. Figure 2.3 shows today's ADC architectures as well as their usage area.

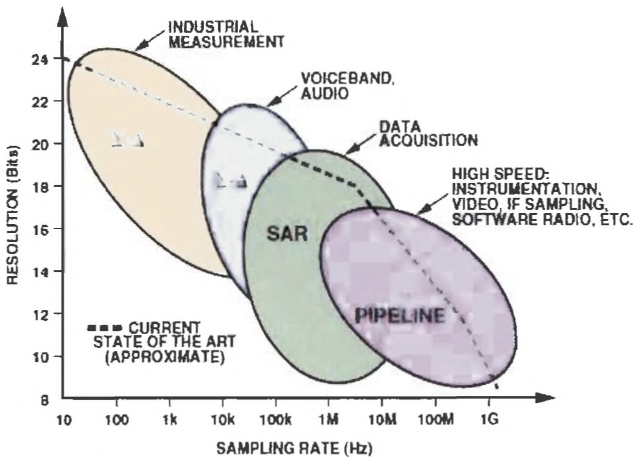


Figure 2.3 – ADC architectures [4]

After understanding these basic differences between the architectures of ADCs, the decision of using a Sigma-Delta ADC was clear. We will now focus on the specific features of Sigma-Delta ADCs.

It is common the Sigma-Delta ADC to be separated into the modulator and the digital filter. A basic first-order Sigma-Delta ADC is shown in Figure 2.4, with the modulator in some detail.

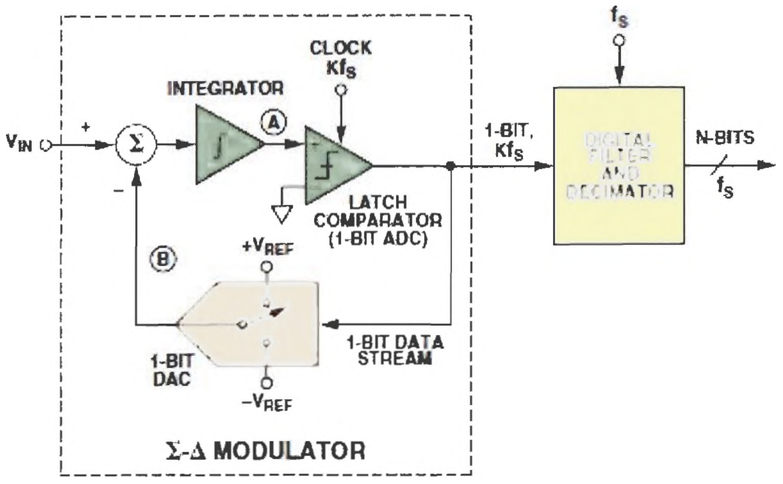


Figure 2.4 – First-order Sigma-Delta ADC [4]

The 'heart' of this basic Sigma-Delta modulator is a 1-bit ADC (comparator) and a 1-bit DAC (switch). The output of the modulator is a 1-bit stream of data. The negative feedback around the integrator requires the signal at B to be equal to the input signal V_{IN}. If V_{IN} is zero, the output will be an equal number of ones and zeros. As V_{IN} increases, the number of ones increases as well while the number of zeros decreases and vice versa.

Sigma-Delta ADCs use several techniques in order to achieve high SNRs. Oversampling, digital filtering, decimation and noise shaping are the basic ones. These techniques are combined to result in less quantization noise. Figures 2.5a and 2.5b show these basic operations of Sigma-Delta ADCs.

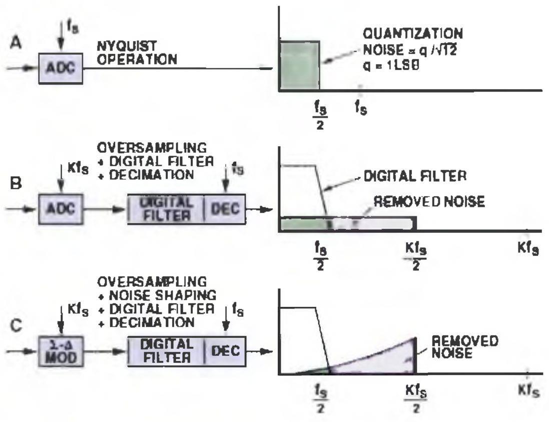

A
NYQUIST OPERATION
ADC
QUANTIZATION NOISE = q/√12
q = 1LSB
OVERSAMPLING + DIGITAL FILTER + DECIMATION
B
DIGITAL FILTER
DEC
DIGITAL FILTER
REMOVED NOISE
OVERSAMPLING + NOISE SHAPING + DIGITAL FILTER + DECIMATION
C
Σ-Δ MOD
REMOVED NOISE
fs/2
fs
Kfs/2
Kfs

Figure 2.5a – Σ-Δ operations [4]

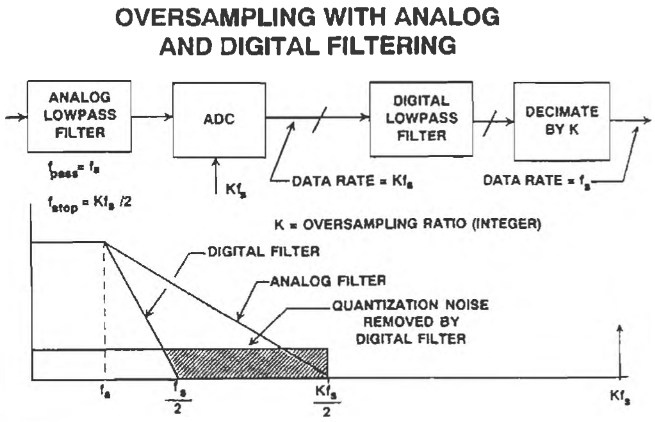


Figure 2.5b – Removed Noise [4]

In Figure 2.5a, *A* shows the noise spectrum for traditional *Nyquist* operation. The quantization noise is spread over the same bandwidth as the input signal. In *B*, the sampling frequency has been increased by *K*, which is called *Oversampling Rate* (OSR). The input signal bandwidth is unchanged, but the quantization noise outside of the signal bandwidth is removed with a digital filter (which needs to be by 1 order higher than the ADC). After this operation, the data rate can be reduced to its original sampling frequency, which is called *decimation. C* shows the Sigma-Delta architecture, where the ADC is replaced by a modulator and the rest of operations are usually done on some *Digital Signal Processor* (DSP). The modulator shapes the quantization noise so that most of it occurs outside the bandwidth of interest, which is called *noise shaping.* Figure 2.5b shows the removal of quantization noise comparing analog and digital filtering.

Regarding the order of a Sigma-Delta modulator, this plays an important role in the effectiveness of the conversion. The SNR level strongly depends on what order the modulator is (0-5) and on what OSR is used, as shown in Figure 2.6.

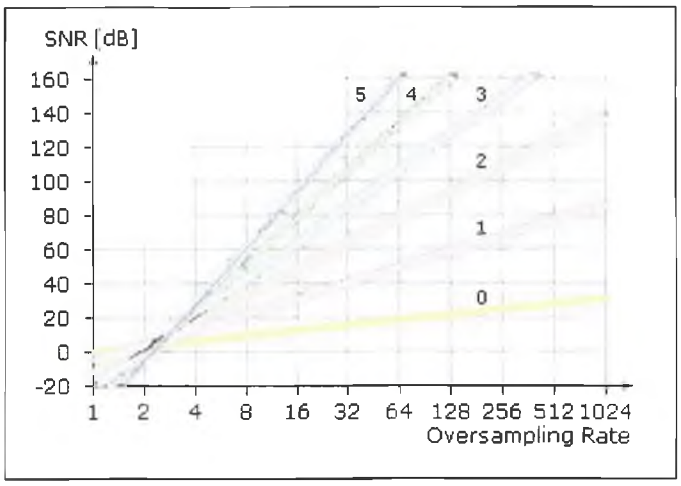


Figure 2.6 – SNR levels [4]

The final choice of a Sigma-Delta Modulator brought us to two specific ADCs, the ADS1202 from *Texas Instruments* and the AD7401 from *Analog Devices*. Both have great characteristics, but one main difference made us to finally choose the AD7401. In Table 2.3 you can see some of their characteristics.

	Resolution	Input Voltage	Clock	Output voltage
ADS1202	16-bit	± 250nV	<20MHz	5V
AD7401	16-bit	± 200nV	<20MHz	3V

Table 2.2 – ADC comparison

We needed the lowest input voltage range possible, so that we could amplify the signal as less as possible to lower the noise levels due to amplification. Additionally, the 3V output voltage fits perfectly because of the LVTTL tolerance of the FPGA input pins (3.3 Volts). ADS1202 has 4 modes of operation and may be used with an external oscillator only in mode3. The mode is chosen by a standard circuitry. AD7401 has no modes of operation, is operating with a clock up to 20MHz. Its analog and digital sides are isolated, a good fact regarding noise issues. Finally, we took big advantage of the datasheet of AD7401 provided by *Analog Devices*, which, unlike to the datasheet of ADS1202, is well organized and detailed. It also provides a sample code in Verilog HDL for a *Sinc*^{3} digital filter and the decimation of the data rate. This will be further described in chapter 3. Although we had found an Application note for a FPGA implementation of ADS1202 (*SBAA094*), the information of AD7401 was in any way better.

In Figure 2.7 you can see the schematic of AD7401, in which the isolated separation of analog and digital sides can be clearly seen.

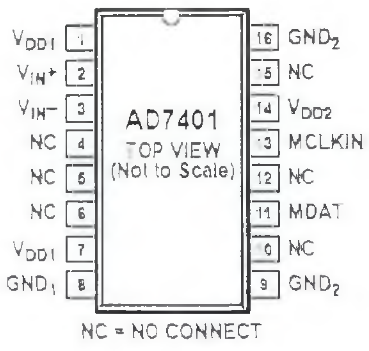


Figure 2.7 – AD7401 Schematic [6]

For any further details regarding ADC architectures, Sigma-Delta modulation and specific details of AD7401, please refer to [4], [5], [6].

2.1.4 Transmission requirements – Transformer

As you will see described in chapter 3, we did not use an integrated PHY chip. This means that we had to fulfil some standard requirements of an Ethernet transmission, regarding physical equipment. The most important component to consider was a transformer, which is necessary for the differential output of the generated bit stream. A transformer itself requires a lot of space regarding our PCB, what made us search for a more convenient solution. We finally came up with an RJ45 Ethernet jack from *Pulse*, which has an integrated transformer. This came out to be a nice as well as flexible solution for our purposes.

Figure 2.8 – RJ45 with integrated transformer

2.2 Analog low-pass filter

Regarding the analog filter that we implemented, we need to refer to its purpose before describing it. Due to the oversampling operation of the Sigma-Delta modulator, there is actually no chance for aliasing to take place. Aliasing in short terms is the condition where the Nyquist theorem is not respected. Normally this is achieved with an analog low-pass filter, but in our case this would be considered as useless. However, the use of an analog filter is still recommended, as it has other advantages, such as the reduction of the captured background noise along with the desired signal.

The human voice frequency band ranges from approximately 300 Hz to 3.4 kHz. Many active filter topologies may be found to implement such a low-pass cut-off frequency. One of them is the common Sallen-Key topology [7], which is combined with a pre-amplifier to define the gain of amplification and the cut-off filter. Sallen-Key filters may be more than 1-order, often using two-staged pre-amplifiers such as the NE5532 that we used. We tried many topologies and finally ended up with a 1-order active feedback filter, with a cut-off frequency of 4 kHz. The gain is regulated by the ratio of the feedback resistor and the output impedance of the ECM, which is approximately 2 kOhm. The capacitor at the output of the filter is used to eliminate the DC offset. Figure 2.9 shows the topology of the filter that we used.

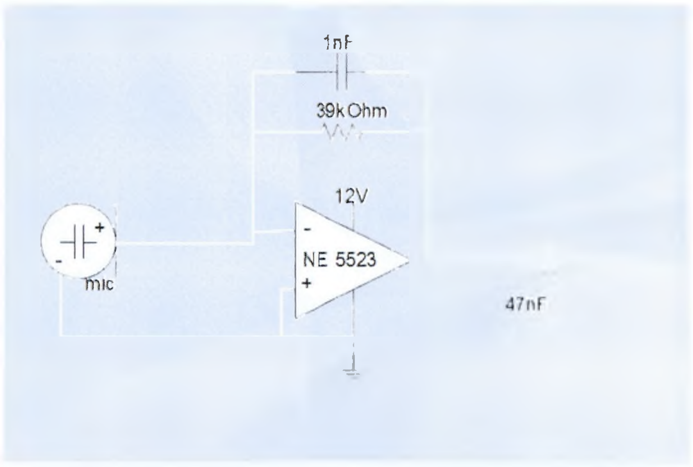


Figure 2.9 – Filter topology

2.3 PCB design

The final design and construction of the PCB daughterboard took place after considerations regarding the FPGA design, as these two boards are directly connected. The PCB daughterboard had to be designed to fit the size of the FPGA motherboard, the connectors used to communicate with the FPGA board had to be correctly placed on the PCB daughterboard and generally other details had to be considered before designing and building the PCB.

2.3.1 Circuit design

The design of the basic circuit included many details. We will start with a list of the components the PCB consists of.

1) ECM
2) Op-Amp NE5532
3) Σ-Δ modulator AD7401
4) 20 MHz oscillator
5) Voltage regulators 7805, 7812
6) Ethernet RJ45 with integrated transformer shielding
7) Several connectors, resistors and capacitors

All components are Surface Mounted Devices (SMD). The PCB itself is double layer. The *Altium designer v6.*0 was used for the design of the circuit.

The flow of the signal is the following: after being captured by the ECM, it is passed through the pre-amplifier and analog filter circuit. It is then inputted to the ADC after passing a small RC circuit recommended by AD7401 datasheet in order to cut off eventual spikes created by the pre-amplifier. From the ADC it is directly passed to the FPGA board. After creating, at the FPGA, the bit stream representing the frame which must be sent over the medium, it returns to the PCB daughterboard from where it is finally transmitted (RJ45) as a differential output. Power is provided over Ethernet (14V), as described in section 2.4. We used two voltage regulators combined with some required capacitors and one inductor (detailed in section 2.3.2), one to regulate 12 out of 14V, and the other for 5 out of 14V. An extra signal controlled by a push button is netted to the FPGA board working as an asynchronous reset. The control signals for the leds which are integrated in the RJ45 connector are driven by the FPGA. Finally, the 20

MHz clock is sent to the FPGA board, although it is returned to the PCB daughterboard, divided by four for the needs of the ADC (5 MHz sampling). We used an external 20 MHz oscillator, although the FPGA board has an integrated 25 MHz oscillator, because of the needs of 10Base-T Ethernet transmission which is Manchester encoded. Further details regarding this issue are found in chapter 3.

The schematic of the design is shown in Figure 2.10.

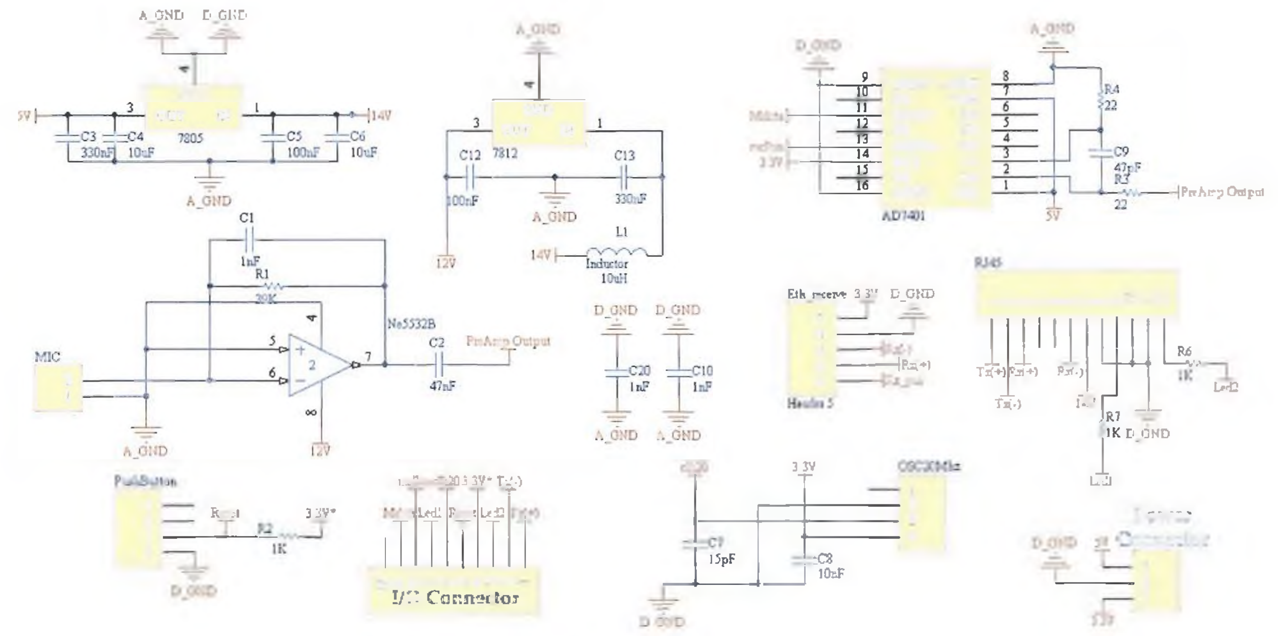


Figure 2.10 – PCB daughterboard Schematic

In detail: there are two connectors used for the interconnection of the PCB and the FPGA board. The I/O Connector consists of ten pins. Seven of these pins lead to general purpose I/O pins of the FPGA, one is the dedicated clock input, one is 3.3 Volts and the 10th is ground. The assignment of these pins is shown in Table 2.3.

Pin	Function
Pin 1	Tx +
Pin 2	Tx -
Pin 3	Led 1
Pin 4	3.3V
Pin 5	Push Button

Pin 6	Clock
Pin 7	Led2
Pin 8	Clock output to ADC
Pin 9	Data from ADC
Pin 10	Ground

Table 2.3 – Main connector pin assignment

The Power Connector consists of three pins, one to supply the FPGA with 5V, one to get back 3.3V from the integrated regulator at the FPGA board and one for the ground.

Table 2.4 shows the pin assignment of the RJ45 connector.

Pin	**Function**
Pin 1	Tx +
Pin 2	Tx -
Pin 3	Rx +
Pin 4	Not connected
Pin 5	Not connected
Pin 6	Rx -
Pin 7	DC +
Pin 8	DC -
Pin 9	Led 1 +
Pin 10	Led 2 -
Pin 11	Led 1 -
Pin 12	Led 2 +

Table 2.4 – RJ45 pin assignment

Figure 2.11 shows the PCB layout from bottom (blue) and top (red) view. One detail to be mentioned is the design of isolated ground planes (described in section 2.3.2), where the analog components are separately grouped in an isolated area (right top corner).

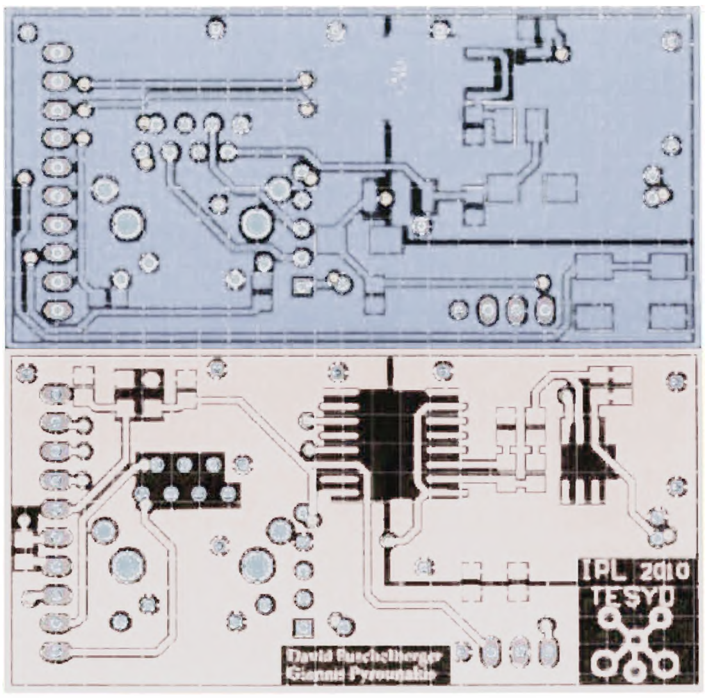


Figure 2.11 – PCB daughterboard Layout

We wanted the PCB daughterboard to exactly fit the FPGA board; therefore its size was measured to be 60mm x 29mm. Because of the minimal available space, routing was done manually. A general rule in routing of nets in a PCB is the curves of the nets to be less than 90 degrees. GND nets were designed to be 0.6mm, power nets 0.5mm and the rest of the nets 0.4mm thick.

The final result of the PCB daughterboard is shown in Figure 2.12, after having built, tested and changed several boards.

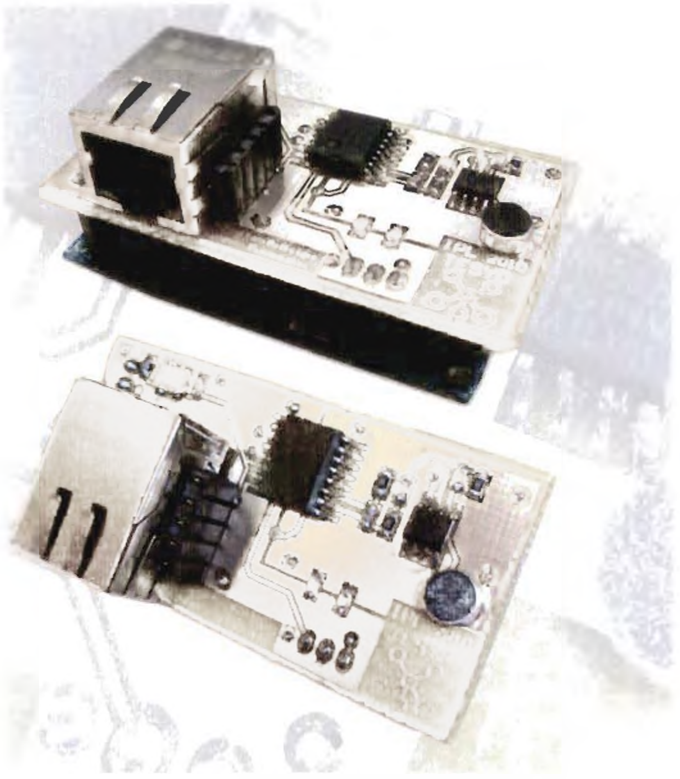


Figure 2.12 – Final PCB daughterboard

2.3.2 Noise issues

The fact that electronics are not perfect leads to several issues regarding the distortion of signals, also commonly known as *noise*. Noise may have many different origins and may lead to a complete failure of any design. It is common that operations done in a digital environment distort the signal while it is processed in the analog environment, as analog components and circuitry are more vulnerable to noise. The reduction of noise is therefore vital for any system, and requires several methods to be used.

First thing to consider was to design a separated ground plane for the analog and digital components of the board. This results in a much smoother electrical signal flow, and, of course, in lower noise which interferes the signal. Top and bottom layer of each ground plane had to be connected by vias and the ground planes with two capacitors, which help to filter out some noise. The planes are also connected by the ADC, as AD7401 has isolated analog and digital sides. The microphone, the pre-amplifier plus analog filter, the 12V regulator and the analog side of the ADC all belong to the analog plane. The oscillator, the two connectors for the FPGA, the RJ45, the push button and the digital side of the ADC are included in the digital plane. Also, the 5V regulator is fed with 12V from the analog side and outputs 5V in the digital side.

The second issue was the power supply. As being inexperienced, we first tried to feed the pre-amplifier directly with 12V over PoE, but this resulted in high electronic noise distorting the amplification and filtering of the signal and therefore the signal itself. We did two things to solve this problem: we used one more voltage regulator to get 12V; we also studied about power supply noise reduction methods, such as bypassing and decoupling, which led us to use a LC circuit to decouple the 12V power supply. This design showed great final results. For more details regarding bypassing and decoupling methods please refer to [8].

2.4 Power Supply - PoE

As you should have already noticed, we designed a PoE device for the power supply of our application. Our design was based on the IEEE 802.3 PoE standard, which defines safe current supply through a standard Ethernet UTP cable along with the transmission of data. In this way, we avoid the use of an adapter to supply our device with power, which makes it flexible. Our PoE device may be plugged in to the network in any location, using PoE compatible[1] network equipment. That means that using a PoE compatible switch for example, several of our devices could be supplied with power by using only one of our PoE devices. However, we designed this device in a way in which it may be also used in a simple topology, such as the direct connection of the device to the PC.

The characteristics defined by IEEE for the 802.3 Power over Ethernet standard are shown in Table 2.5.

PINS on Switch	10/100 DC on Spares	10/100 Mixed DC & Data	1000 Gigabit DC & Bi-Data
Pin 1	Rx +	Rx + DC -	TxRx A + DC +
Pin 2	Rx -	Rx - DC +	TxRx A - DC +
Pin 3	Tx +	Tx + DC -	TxRx B + DC -
Pin 4	DC +	unused	TxRx C +
Pin 5	DC +	unused	TxRx C -
Pin 6	Tx -	Tx - DC +	TxRx B - DC -

[1] PoE compatible does not necessarily mean that the equipment *supplies* power, but that it also allows current to *pass through*. However, equipment which provides PoE power supply of connected devices does exist on the market.

Pin 7	DC -	unused	TxRx D +
Pin 8	DC -	unused	TxRx D -

Table 2.5 – IEEE PoE specifications

As we are using 10Base-T Ethernet transmission, we preferred the *DC on Spares* category for our PoE design.

The design is very simple, using two RJ45 connectors (Ethernet jacks) and one power jack. We use pins 4 and 5 for the positive DC (+) and pins 7 and 8 for the negative DC (-), although only the Input RJ45 connector is supplied with DC voltage. That means that there is no current flow in direction of the PC. The rest of the pins are regularly used for data communication. Regarding the appropriate cabling of our device, if no switch is used, a crossover cable is required to connect the PoE with the PC, while the device should be connected to the PoE with a straight-through cable. If a switch is used, there is no trouble caused by using any type of cable.

The layout of the PCB of our PoE device is shown in Figure 2.13. There is no need to show the schematic, as it is simple and described above.

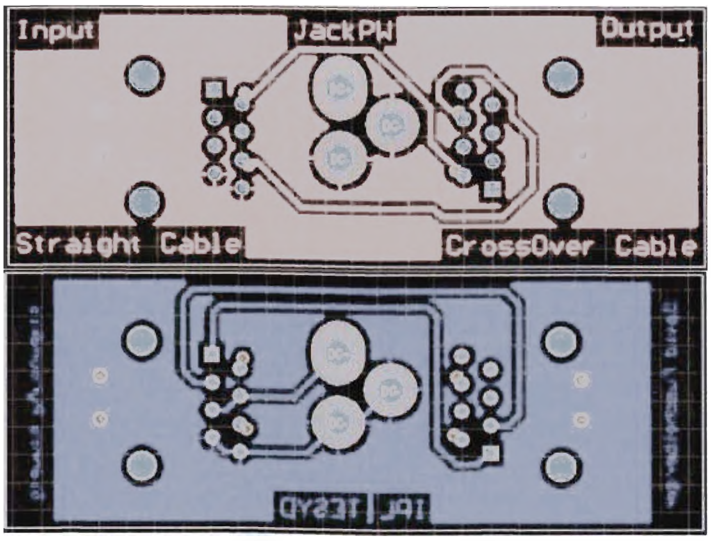


Figure 2.13 – PoE Layout

Figure 2.14 shows the final PCB of our PoE.

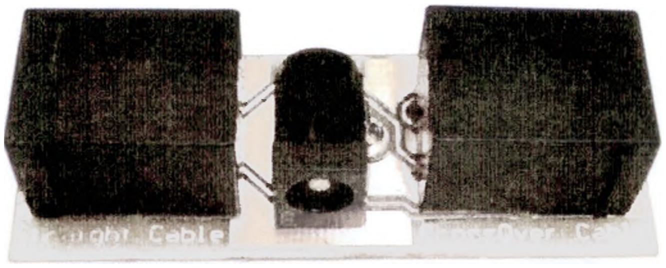


Figure 2.14 – PoE device

For further information over PoE please refer to [9].

2.5 Receiver circuit

Our device uses a one-way communication, also referred to as *simplex*. In order the device to be flexible for further implementation and eventual expansion of the application, we additionally designed the circuitry required for Ethernet reception. This offers the ability to add several functions which require a two-way communication.

Figures 2.15a and 2.15b show the schematic and the layout of the PCB, from where you can also distinguish the combination of resistors, capacitors and transistors used to achieve this circuit's functionality.

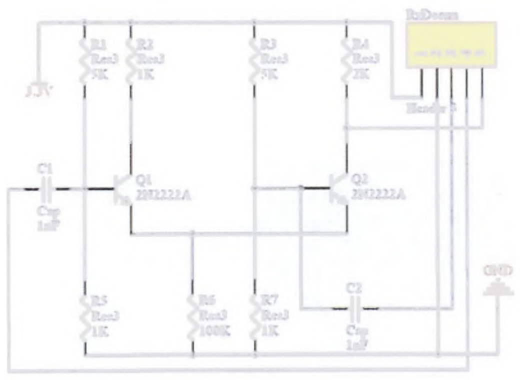


Figure 2.15a – Rx Schematic

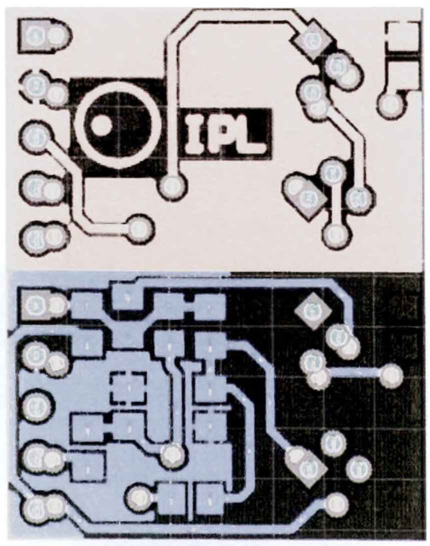


Figure 2.15b – Rx Layout

One note which has do be made is that this circuit is recommended for Ethernet reception without using a PHY chip by *www.fpga4fun.com*, which we will refer to later in this report.

3. Digital part of the Design

The digital part of our project was implemented on the FPGA chip of *Altera*, Cyclone EP1C3T100, with 2910 logic cells and 51 I/O pins. The board, which the chip is on, has an integrated serial programmer, which facilitated our tests and therefore our design in general. The software used to compile, debug and simulate the design was Quartus II 9.1 sp2 Web Edition by *Altera*. Both Verilog HDL and VHDL were used throughout the design and the final implementation of the FPGA design exists in both languages. For any eventual question regarding the Cyclone chip of Altera please refer to [23].

3.1 Main description of FPGA design

The design consists of five blocks as shown in Figure 3.1 and communication is achieved through eight I/O pins, excluding the pins used for power supply and GND.

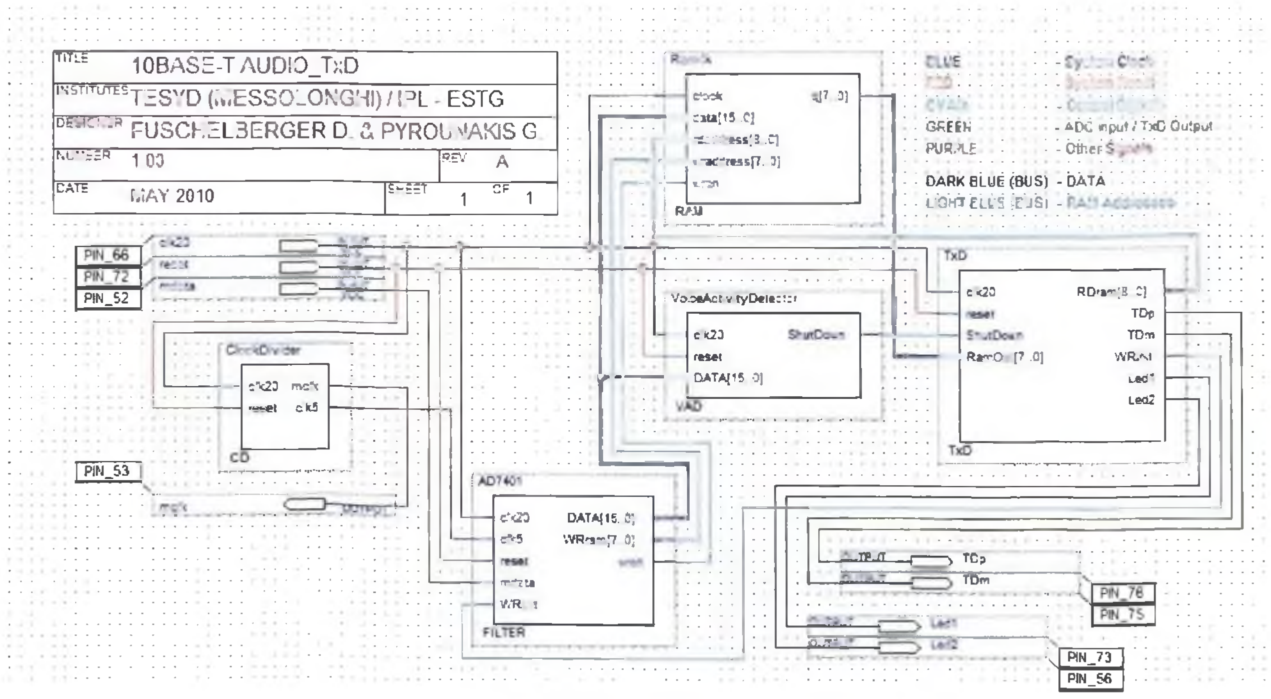


Figure 3.1 – FPGA design block diagram

Seven general purpose I/O pins plus the dedicated clock input pin (pin 66), at which the 20 MHz clock is received from the PCB, are used to exchange signals with the PCB daughterboard. The active low reset signal generated by the push button is received at pin 72. The digitized signal output from the ADC is assigned to pin 52. The signals driving the two leds are outputted from pins 73 and 56. Pins 76 and 75 are outputting the generated bit stream (data frame) which is turned into a differential output at the RJ45. Finally, pin 53 is used to send the divided 5 MHz clock to the ADC.

Pin	Function
Pin 52	mdata
Pin 53	mclk
Pin 56	Led2
Pin 66	clk20
Pin 72	reset
Pin 73	Led1
Pin 75	TDm
Pin 76	TDp

Table 3.1 – FPGA pin assignment

The serial bit stream representing the digitized data from the ADC (mdata) is inputted to the Filter block, where the interface, filter and decimation process is executed. The writing process of the RAM block is also achieved at this block (wren/WRram), receiving a control signal (WRinit) from the transmitter (TX) for the initialization of the RAM addresses. Data (16-bit samples) is then buffered into the RAM block (DATA). The Transmitter block is generating the UDP packets (TDm/TDp) with the stored data encapsulated into its payload (RamOut) by reading it from the RAM block (RDram), but is also controlled by the Voice Activity Detection (VAD) block, which, if it is detecting silence, is shutting down the transmission to prevent unnecessary network flooding (ShutDown). The two leds are also driven at this block (Led1/Led2). There is one more block acting as a clock divider, which is dividing the 20 MHz clock (clk20) by four, outputting a 5 MHz clock signal (clk5) for the needs of the ADC (oversampling). Finally, an asynchronous reset signal is netted to all blocks except the RAM block.

Blocks, pins and signals were connected through the schematic design shown in Figure 3.1 rather than manual port mapping, which creates a top-entity VHDL file for the compilation of the project.

The topology of the I/O pins of the FPGA board is shown in Figure 3.2. The two connectors described in chapter 2 for the communication of the FPGA board and the PCB daughterboard may be clearly noticed.

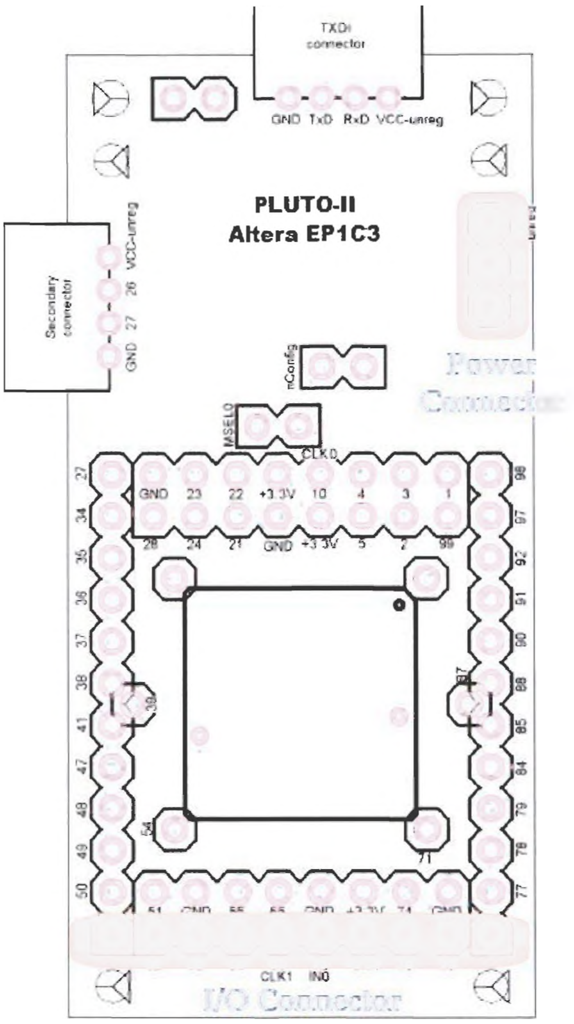


Figure 3.2 – FPGA board pin schematic

3.2 Interface, filter and decimation block

Several operations are done at this block, as already mentioned. The first thing to be done was to interface the serial bit stream from the ADC to the FPGA. Regarding parallel and serial interfacing, the main difference is that using a Parallel Peripheral Interface (PPI) you must suffer an enormous waste of I/O pins because of the need to use as many pins as your data bus width. For example, in our case we would need 16 pins just to interface the ADC (16-bit resolution); of course this is just a hypothetical example, as AD7401 demands a serial interface. Regarding serial interfacing, there are several different ways to do this, such as the Serial Peripheral Interface (SPI) bus by *Motorola* or the Inter Integrated Circuit (I2C) bus from *Philips*. These interfaces use a master/slave operation and may interface more than one external device to a DSP or a FPGA. However, it is much simpler in our case. A simple serial interface based on Delta modulation (because of the Σ-Δ architecture of AD7401) was enough to interface the ADC and serially receive the bit stream (mdata, *ipdata1* in Figure 3.3). Mdata is either a logical zero '0' or a logical one '1', and is received at every positive edge of clk5 (5 MHz).

After interfacing the ADC, the bit stream had to be filtered and decimated to its original data rate. AD7401 is a 2nd order modulator, which means that the filter must be at least 3-order staged. A $Sinc^3$ filter is the most common digital filter used in such cases. The $Sinc^3$ filter which we used consists of three accumulators and three differentiators. The accumulation (IIR) is done at the speed of the modulator (5MHz, mclk *–mclkin* in Figure 3.3) and the differentiation (FIR) at the decimated speed (5 MHz /256, word_clk), depending on the decimation ratio used (further described below). Regarding Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters, also known as *non-recursive* and *recursive*, it is to mention that they have one main difference: the FIR filter output depends on the previous and present input data, while the output of an IIR filter depends on the previous and present input data as well as the previous output data. The word *recursive* may be referred to as *running back*. This explains the 3-order topology of the filter that we used. The filter's output is depending on the 3 previous samples, resulting in a smooth response of the signal in case of eventual quantization errors occurring during the A-D conversion. Figure 3.3 shows the schematic of the filter topology.

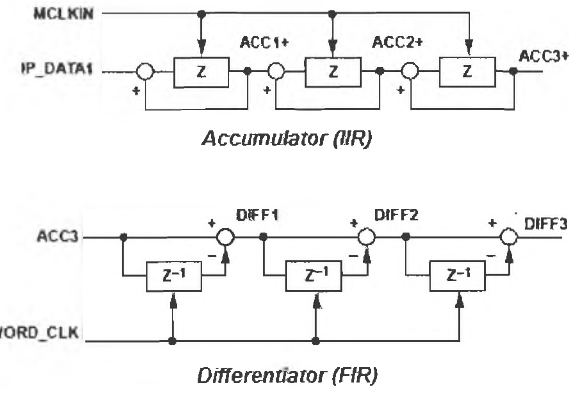


Figure 3.3 – Accumulator and Differentiator schematic [6]

Regarding the decimation process, our signal is oversampled at 5 MHz and we use a decimation ratio of 256, which gives a final frequency of 19.531Hz, or approximately 19.5 kHz. As shown in Figure 3.4, the 16 Most Significant Bits (MSB) of the oversampled data vector are extracted at the decimated speed, which outputs the Effective Number of Bits (ENOB) defined by the datasheet of AD7401 due to its 16-bit resolution. This means that after the decimation operation we get a final 16-bit wide audio sample every 256th cycle of the 5 MHz clock. The decimation speed is achieved through a clock enable signal every 256 cycles (word_clk), avoiding in this way the use of a ripple clock.

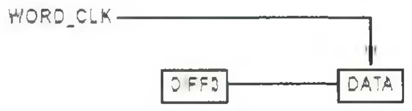


Figure 3.4 – Decimation process [6]

The SNR achieved by the 3rd order filter and the decimation ratio of 256 is higher than 80dB, as shown in Figure 3.5.

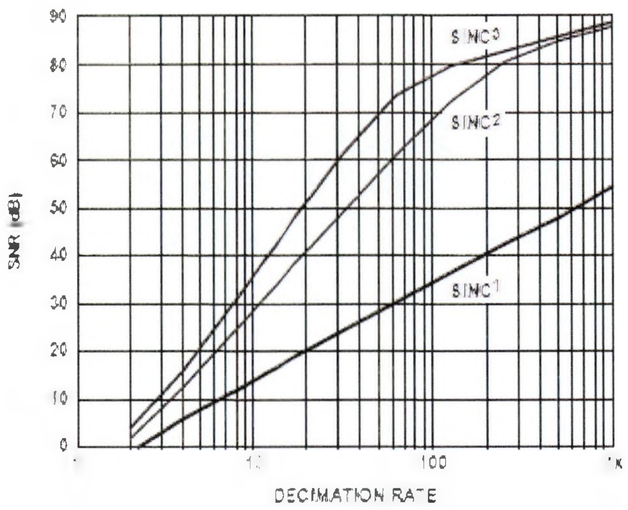


Figure 3.5 – SNR levels [6]

The code used to create this $Sinc^3$ filter and decimation operation was the sample code provided by the datasheet of AD7401. The code was initially written in Verilog, but we translated it to VHDL and made a few small changes, such as the replacement of the ripple clock (used for decimation) with a clock enable signal. For more details please refer to [6].

One more thing done at this block is the operation which writes and addresses the final samples into the RAM block. This is achieved by creating another clock enable signal (wren) raised at every 256th cycle of clk5 when the sample is ready to be transferred to the RAM. The RAM address used for the writing process is increased at every enable signal and overflows when enough data samples required to fill the payload of one packet have been written into the RAM. This operation is also further described in section 3.3. There is a problem in the synchronization of the addressing scheme at the beginning of the transmission, which requires a control signal (WRinit) sent from the transmitter block to initiate the writing process whenever a new transmission is done. This is normally required only during the first packet which is sent when the connection is established, as you will notice later in section 3.3. After the initial synchronization of the process, the write address vector overflows at the end of the transmitted packet to start writing the data for the next packet.

One last thing to be mentioned is that DATA is turned into twos complement for the demands of the software part. This is achieved by subtracting 0x8000 from the ones complement data sample. Also, the RAM block that we created outputs the 16-bit samples divided into two 8-bit vectors (due to the needs of the transmitter), and does this inversely. This means that the initial 16-bit sample written to the RAM is outputted with its 8 Less Significant Bits (LSB) as MSB and vice versa. That could easily be corrected by extracting the MSB in exactly the same way, but essentially the wave file format (used at the PC, software part) requires the 16-bit samples in this order, with the LSB first, so that this issue is actually solved by itself.

The whole RTL source code of the Interface/Filter/Decimation block, written in VHDL, can be found in Appendix A.

3.3 Transmission

The transmission was a vital part of our project, as its successful achievement let us experiment with final results received at the PC, making any eventual debug steps necessary much easier. The transmission involves many details and characteristics, general as well as practical, which we will try to describe step by step in the following subchapters.

3.3.1 Ethernet requirements

In order to proceed to the practical analysis of the implementation, there are several issues which must be cleared out. In our application we used a simplified version of an Ethernet 10Base-T transmitter, avoiding the use of a two-way communication and a specific PHY chip. We create and transmit UDP packets, without the need of establishing a TCP session (3-way handshake, requiring duplex operation). UDP is a connectionless protocol, which gives us the ability to implement a transmission-only operation with low cost characteristics. Regarding the protocol stack used to describe the operation of our design, we will analyze some basic functions of Physical layer 1, Ethernet MAC layer 2 standards and layer 3 and 4 UDP/IP standards.

Regarding layer 1 PHY, there may be found several solutions on the market offering an integrated PHY chip which executes all the needed operations for the data to be placed onto the medium. The use of a purchased PHY chip for the required Layer-1 operations would be an expensive choice regarding a low-cost implementation. Below you can see a list of these basic PHY operations, which we finally implemented on the FPGA and the PCB daughterboard.

1) Data has to be shifted bit-by-bit onto the physical medium (bit stream), which requires a shift register.
2) Data must be Manchester encoded.
3) Two continuous high pulses, called Normal Link Pulse (NLP), are used for the establishment of the connection and have also to be sent every 16 ± 8 ms, if no packets are transmitted, in order the devices to indicate their presence to each other.
4) Six continuous high pulses (TP_IDL) must be added at the end of each frame to indicate the Interframe Gap (IFG).
5) The nominal output voltage level of each differential output for 10Base-T Ethernet has to be between 2.2 – 2.8 Volts [15].
6) A transformer has to be placed before the output of data onto the medium for electrical isolation and differential output.

Points 1-4 are done on the FPGA, as described later in this chapter. The FPGA itself has an output voltage of 3.3 Volts (LVTTL), which is acceptable regarding point 5. Point 6 is done by the RJ45 Ethernet Jack with the integrated transformer.

There is a part of one more operation done at Layer-1 called Carrier Sense Multiple Access with Collision Detection (CSMA/CD), which is used in network topologies with only one collision domain in order to allow multiple devices to communicate without collisions. CSMA/CD requires a two-way communication and was not implemented in our design. For further information on CSMA/CD please refer to [10].

The Manchester code is a simple way of encoding data which is transmitted over some kind of medium. It is the code used for 10Base-T Ethernet encoding. Regarding its operation, every bit is represented by actually two bits, in each clock cycle. The theory by *G.E. Thomas* [11] initially defined a '0' to be represented with a low-to-high transaction and a '1' with a high-to-low transaction. IEEE standard nowadays defines it inversely, but both ways are recognized by today's hardware. The Manchester code has no DC component, and permits clock recovery out of the transmitted data.

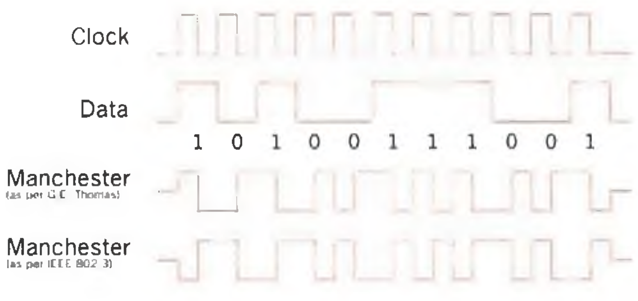


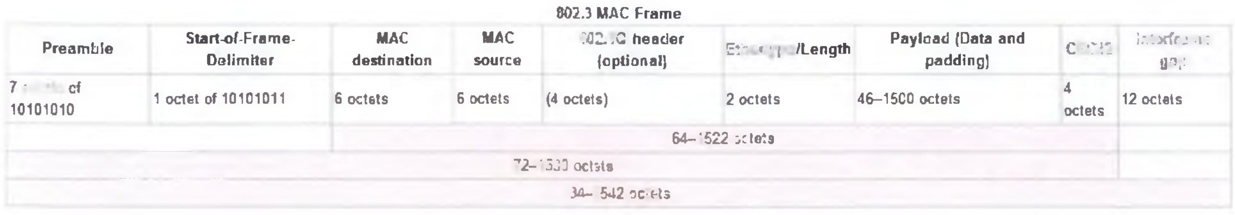
Figure 3.6 – Manchester code [11]

We achieve Manchester encoding by using the 20 MHz clock signal divided by two, to get 10 Mbps of the transmission-only Ethernet link.

The NLP, also known as *Link Test Pulse* (LTP), has several functions. It initializes an Ethernet link indicating an active transmitter and establishing the link. It is also used for the operation known as *Autonegotiaton*, in which two connected devices choose their transmission characteristics, such as half- or full-duplex and/or speed. Finally, it is also used by the communicating devices to indicate their presence to each other, when no packets are exchanged. This signal is sent at intervals of 16 ± 8 ms. [12]

The IFG stands for a minimum idle period between transmissions of Ethernet frames. This is 9.6 μs for 10Base-T, by far less than we need. The Interframe Gap is indicated by a signal known as TP_IDL, which is transmitted immediately after each Ethernet frame. This is a signal of a continuous 6-cycle high pulse [13] [14].

Considering layer 2 Ethernet operation (only MAC, LLC sublayer is not described by the Ethernet standard), there are several details to be mentioned. In Figure 3.7 you can see the structure of an Ethernet PDU – *frame*.



802.3 MAC Frame

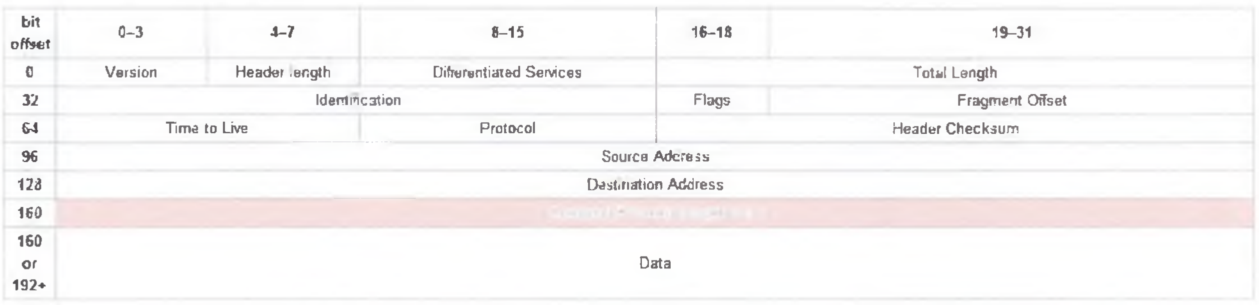
Preamble	Start-of-Frame-Delimiter	MAC destination	MAC source	802.1Q header (optional)	Ethertype/Length	Payload (Data and padding)	CRC32	Interframe gap
7 octets of 10101010	1 octet of 10101011	6 octets	6 octets	(4 octets)	2 octets	46–1500 octets	4 octets	12 octets
		64–1522 octets						
72–1530 octets								
84–1542 octets								

Figure 3.7 – Ethernet frame structure [17]

Each field of the frame is briefly described below, and almost all of them have to be designed in order the Ethernet transmission to be achieved successfully.

1) Preamble – by default 7 bytes of 0xFF
2) Start-of-Frame Delimiter – by default 0xD5
3) MAC Destination Address (6 bytes)
4) MAC Source Address (6 bytes)
5) 802.1q (VLAN Tagging) – optional field (4 bytes)
6) Ethertype/Length – 2 bytes, 0x0800 for IPv4
7) Payload – max 1500 bytes per frame
8) FCS/Ethernet Trailer – 4 bytes of CRC'32 calculated checksum, Preamble is NOT CRC calculated [18]
9) Interframe Gap – depends on PPS being transmitted

Layer 3 defines the IP packet. In Figure 3.8 you can see the structure of an IP PDU – *packet*.



bit offset	0–3	4–7	8–15	16–18	19–31
0	Version	Header length	Differentiated Services	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	[illegible]				
160 or 192+	Data				

Figure 3.8 – IP packet structure [16]

As above in layer 2, also the fields of the IP header are briefly described below. Some of them do not need to be used.

1) Version – IP version used, 4 bits, 0x4 for IPv4
2) Header length – length of IP header, 4 bits
3) Differentiated Services – originally defined as TOS, specifies preferences of how the datagram should be handled, 1 byte
4) Total Length – defines the size of the datagram, including header and payload, min is 20 bytes - max is 2^{16}, 2 bytes
5) Identification – primarily used to uniquely identify fragments of an original datagram, 2 bytes
6) Flags – 3-bit field used to control fragments, bit0: must be zero, bit1: DF (Don't Fragment), bit2: MF (More Fragments)
7) Fragment Offset – controls the offset of a particular fragment, 13 bits, allows a max offset of $(2^{13}-1)*8=65.528$ bytes which would exceed the max packet length (Total Length)
8) TTL – defines a max amount of time a packet may stay alive (in sec), prevents loops, 1 byte
9) Protocol – defines which Layer-4 protocol is encapsulated into the IP datagram, 1 byte, 17 for UDP
10) Header Checksum – used for error-checking of the header, 2 bytes, [20]
11) Source Address – 4 bytes logical source address
12) Destination Address – 4 bytes logical destination address
13) Payload

Finally, the typical layer 4 UDP PDU, *datagram*, consists of the following:

1) Source port number – normally used if required due to eventual response Destination port number – indicates the receiver's port, required
2) Length – length of the entire datagram, in bytes, with a max of 65.527 bytes data (IP packet) + 8 bytes UDP header
3) Checksum – used for error checking of header and data, if omitted in IPv4 the field has a value of all-zeros, not optional for IPv6

The above mentioned layer 2, 3 and 4 headers, along with the encapsulated data in the UDP payload, result in one final Ethernet frame which is turned into a bit stream at layer 1 and transmitted over the medium.

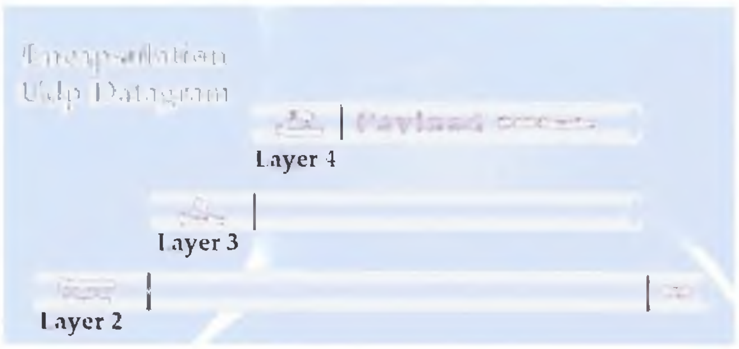


Figure 3.9 – Encapsulation process

For any further detailed information regarding the IEEE 802.3 Ethernet standard requirements and specifications, as well as the OSI model, please refer to [15] [17] [18].

3.3.2 Transmission characteristics

As already mentioned, the final sampling frequency of our signal is approximately 19.5 kHz. This sampling frequency, along with the 16-bit resolution and Mono audio (1 channel), results in a data rate of 312 kbps ($DR = F_S * \mathrm{Re}s * Ch$). We decided to encapsulate 512 bytes payload into each packet, which represents about 13.1 ms of audio data. The total overhead of our transmission has a size of 54 bytes, which, together with the payload, makes a total packet of 566 bytes. The minimum transmission requirement refers to 19.500 transmitted samples per second (sampling frequency) in order to avoid loss of information. Considering 256 samples encapsulated in each packet, this results in 76 packets per second (19500/256 □ 76 pps). Finally, the total size of the packet and the packets transmitted per second result in approximately 344 kbps of consumed bandwidth. Table 3.2 shows the summary of these characteristics.

Sampling Frequency	19.5 kHz
Data Rate	312 kbps
Size of Packet	566 bytes (54 + 512)
Packets per Second	76
Total Bandwidth	344 kbps

Table 3.2 – Transmission characteristics

3.3.3 Ram block

We placed the RAM block in this subchapter of transmission, because it is closely related with the transmitter design. The RAM block is Wizard-generated, using the *altsyncram* Megafunction available in Quartus-II software. It is a dual-port Ram with an input word width of 16 bits and output width of 8 bits, due to the needs of the encapsulation progress at the transmitter. The block's size is 4 kbits, which is exactly the size of the encapsulated payload. As already described, a write enable signal is driving the addresses to which data is written in the Ram. As you will see in section 3.3.4, no read enable signal is used to read data out of the Ram. This is done by simply increasing the read address every time data has to be read during the encapsulation progress.

3.3.4 Transmitter block

The main code of the 10Base-T transmitter block is based on the source code of an example project (10Base-T) located at www.fpga4fun.com, which is an additional webpage of KNJN LLC, from which we purchased the Pluto-II board. The initial code was written in Verilog, but the concept of the example project is totally different than ours. Additionally, we have written our entire FPGA design in both Verilog and VHDL.

Before beginning the description of each part, we have to mention that the clock used in this block is the system clock *clk20* (20 MHz), required by the implemented one-way communication (transmission-only) to achieve 10 Mbps. This is needed because of the Manchester code used to encode data transmitted over a 10Base-T Ethernet link.

Regarding the flow diagrams used to describe several operations of the TX block as well as the explanation itself, note that this is a low-level design and may at some points be hard to understand. A closer look at the RTL source code (VHDL) in Appendix B may result in better understanding.

The transmitter block executes several operations which are listed below.

1) Ram reading process to get access to data.
2) Transmission initialization process to control the transmission flow.

3) Encapsulation process to create the packet and encapsulate data accessed from the Ram.
4) CRC process for the checksum calculation of the entire Ethernet frame, excluding the preamble[2].
5) NLP process to create the necessary NLPs whenever no transmission of packets is occurring.
6) TP_IDL process to achieve the indication of the Interframe Gap (IFG).
7) Manchester encoder for the encoding of transmitted data.
8) Led process to control the way the two leds of the RJ45 behave in idle mode of the transmitter and during the transmission.

Figure 3.10 shows the flow diagram of the transmitter.

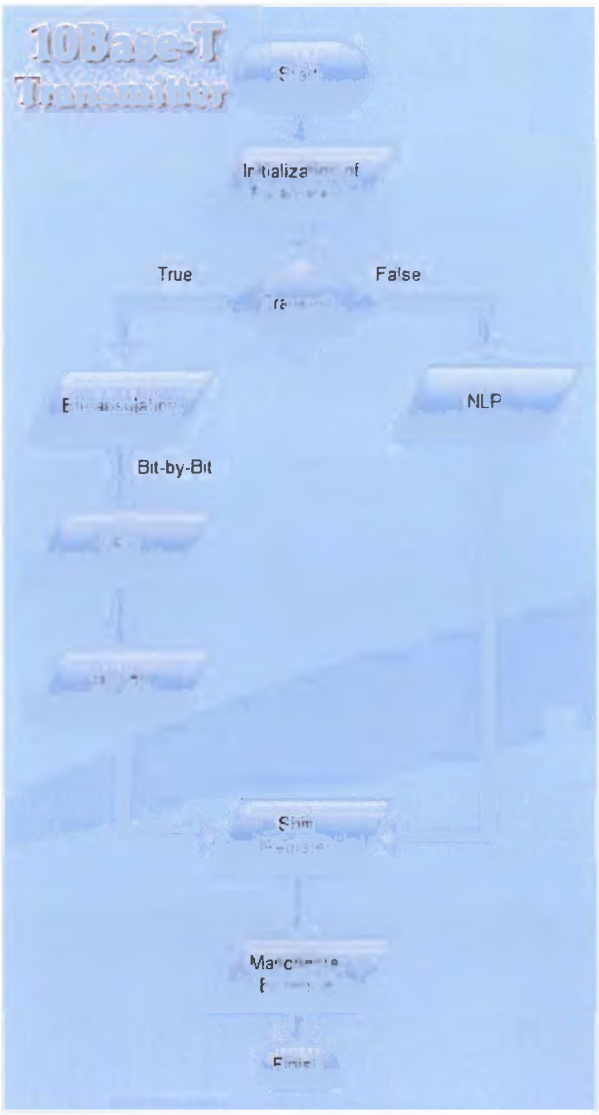


Figure 3.10 – Flow diagram of 10Base-T transmitter

[2] Additionally to the CRC checksum, the IP checksum is calculated and added to its field in the IP header.

Starting with the initialization of the parameters, this is mainly where the number of packets sent per second is defined. This is done with an 18-bit counter initializing the transmission of the packet (*StartSending*) every 13 ms (2^{18} * 50 ns). In order the Ethernet link to be established it is required to transmit several NLP signals first. This is also done in this process, turning the decision block into *false* for approximately one second in order to establish the link. This is done at power-up and when the device is reset. During regular operation, the decision block is controlled by the *ShutDown* signal sent by the VAD block. If voice activity is detected (*true*), the transmission occurs normally, otherwise the state goes into *false* and only the transmission of NLPs is done in order to indicate the presence of the device and not to lose the connection to the PC. When transmitting, the TX block executes the Encapsulation, CRC and TP_IDL processes to create and forward the packets. Either packets or NLPs, both are aggregated bit-by-bit at the Shift Register and put into the correct sequence, before being forwarded to the Manchester Encoder and finally transmitted.

Figure 3.11 shows the flow diagram of the encapsulation process.

Stat: nformat:on
Read RAM

Figure 3.11 – Encapsulation process flow diagram

The encapsulation process is where the structure of the packet is created. This process includes one huge *case* block, whose control register is *RDaddress*. *RDaddress* is controlled by *ReadRam* which turns high every 16th clock cycle. This operation is done every 16th cycle due to the needs of the Manchester code, which requires twice the time in order to encode the transmitted data (two bits representing one bit of data). The encapsulation case is sequentially getting access of the constant values of the headers and then the audio samples located at the RAM block, one byte per case. The read address vector (*RDram*) used to access data from Ram in order to be placed into the payload, is increased and handled along with *RDaddress*, starting after the encapsulation of the last header (UDP). These bytes (*pkt-data*), which represent the packet, are then assigned to another vector in one cycle (when *ReadRam* is high), *ShiftData*, which is shifting the data every 2nd cycle bit-by-bit into the CRC process. Additionally, the *WRinit* signal is created in this process, which is sent back to the Filter block to initialize the Ram address assignment (Ram writing process) because of initial synchronization mismatch. This signal is included into the reset operation to achieve again synchronization in case of an error.

Figure 3.13 shows the flow diagram of the CRC calculation process.

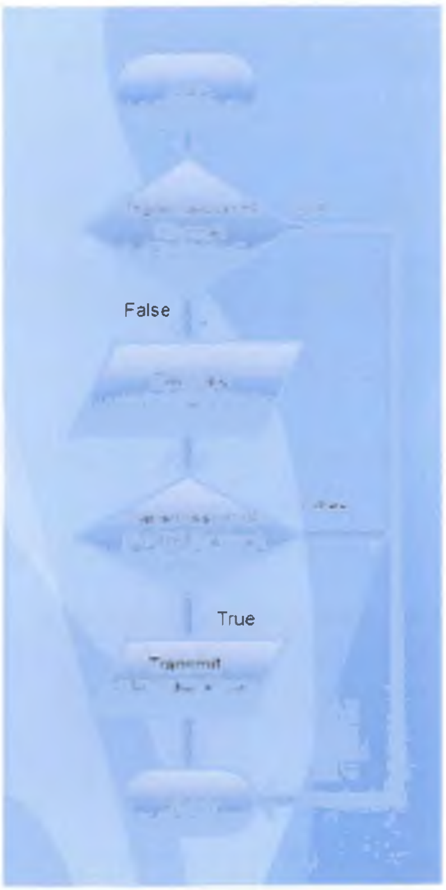


Figure 3.12 – CRC calculation flow diagram

The CRC process is calculating the CRC'32 4-byte vector which is placed at the end of the frame, in the FCS field of the Ethernet trailer. The Ethernet preamble is not required to be calculated. The process uses several control signals. One of these is *CRCflush*, which turns high for 64 continuous clock cycles after the encapsulation process, or in other words for 4 *RDaddresses* or 32 bits (size of CRC vector). *CRCflush* is used at the Shift Register to add the CRC calculated vector to the end of the packet. Another signal, *CRCinit*, indicates the start of the CRC calculation after the preamble has been retrieved. The CRC calculation itself is achieved by several operations, computing the bit-by-bit passing data with the CRC'32 calculation polynomial (0x04C11DB7) [19].

Figure 3.13 shows the flow diagram of the TP_IDL process.

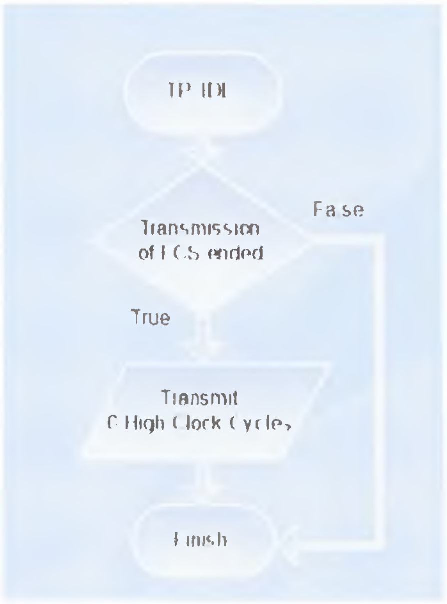


Figure 3.13 – TP_IDL process flow diagram

The TP_IDL process is creating a signal which is high for 6 continuous cycles along with every packet, to be added to each packet at the end of the transmission.

The NLP process is simply creating a pulse which is high for two continuous cycles every 16 ms, to be sent whenever needed (no transmission/link establishment).

Finally, the Shift Register is aggregating the bit stream (data), the 4-byte CRC calculated vector, the 6 cycles high TP_IDL signal and eventually (when needed) the NLP signal, in order to forward this information *correctly sequenced* [3] to the Manchester Encoder where it is encoded and finally transmitted.

The Manchester Encoder is using the *ShiftCount* vector, used by the Encapsulation process to control *ShiftData* which shifts the data bit-by-bit, in order to achieve the Manchester code. The data bit (*DataOut*) is available for 2 cycles and is encoded by XORing it with the LSB of *ShiftCount* (which changes every cycle). Specifically, if *DataOut* is a '1' it is XORed with *ShiftCount(0)* (which is '1' during the first cycle) giving a '0'. In the second cycle *DataOut* remains the same, as it is not affected by being XORed with '0' (*ShiftCount* during the second cycle). This results in *DataOut*, initially a '1', being finally represented as '01'. Likewise, if *DataOut* is a '0', a '1' is added in front of it making it '10'. In this way we achieve the Manchester Code defined by IEEE 802.3.

Regarding the two leds, they are programmed to blink quickly during the transmission, while in idle mode of the transmitter one of them is static and the other one is blinking once a second.

3.3.5 Further considerations

Some further thoughts had been done regarding the transmitter and the device in general. As described in chapter 2, we built a small extra PCB which gives us the ability, besides transmitting, to also receive packets at our device. Some extra functions could be added using a two-way communication. Specifically, the main ideas for expansion are two protocols known as Address Resolution Protocol (ARP) and Dynamic Host Configuration Protocol (DHCP). ARP is responsible for MAC addresses to be learned throughout the network, while DHCP (some server, either a Router or even the PC) automatically assigns an available IP address to several network devices. These issues were bypassed in our project by simply configuring both MAC and IP addresses manually at the FPGA. When configuring the MAC destination address we have to use the next-hop MAC address, such as the switch. When configuring the IP addresses, we have to consider the device to be in the same logical network as the PC. ARP and DHCP would solve out these issues, reaching the concept of the device to a higher level. One critical question would be if the logical elements available on our chip would be enough for such an extra implementation. However, these

[3] The correct sequence is 562 bytes of headers plus payload, 4 bytes of the CRC vector and finally the 6-cycle TP_IDL signal. The NLP is forwarded independently of the above information, as it is not part of the packet.

considerations are nothing but ideas and their accomplishment is beyond the scope of this project, mostly due to limited time available for the realization of the project [21] [22].

3.4 VAD block

The Voice Activity Detector is used to identify whether voice is actually being captured or not. This is done in order to prevent the network from being unnecessarily flooded by useless packets. This is achieved by controlling the amplitude of the samples received at the Filter block and their assignment to a specific threshold, which has been measured as noise during silence. Specifically, when silence occurs, the samples received from the ADC do not have an absolute amplitude of zero. In fact, this silence is considered to be noisy and higher noise spikes may eventually occur. This is why a threshold is assigned to define the levels between silence and voice activity.

Regarding the design of the VAD, it is sending a *ShutDown* signal to the transmitter after detecting 200 ms of continuous silence (samples inside of the threshold). In order to avoid unnecessary enabling of the transmitter due to eventual spikes (NO voice activity), a continuous out-of-threshold activity of approximately 10 ms has to be detected to re-enable the transmitter after a period of silence.

Only NLPs are transmitted during the period of detected silence. DATA is still written into the Ram, but is never retrieved and does not affect in any way the rest of the operation.

Figure 3.14 shows the general concept of the VAD block.

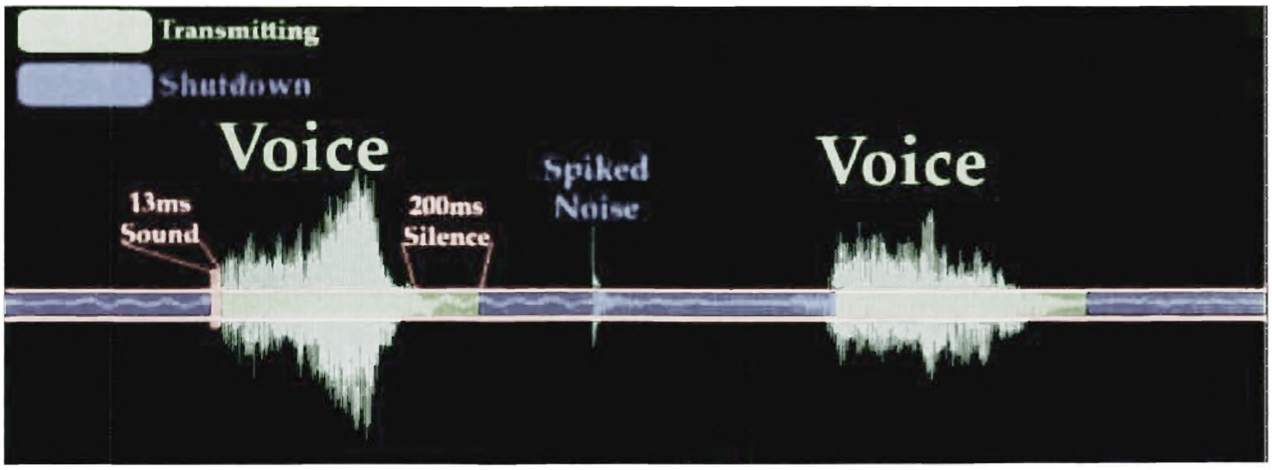


Figure 3.14 – VAD concept

The RTL source code of the VAD block, written in VHDL, can be found in Appendix C.

3.5 Clock Divider block

The Clock Divider is generating a ripple clock of system clock *clk20*, dividing it by four in order to achieve a 5 MHz clock signal (*clk5*) for the needs of the ADC. The Cyclone EP1C3T100 FPGA chip has one internal Phase-Locked Loop (PPL), which is used to divide or multiply clock signals. Unfortunately it cannot produce clock frequencies lower than approximately 16 MHz, so that we could not use it to achieve the 5 MHz frequency. One idea would have been to use a 5 MHz oscillator and multiply it by four in order to achieve the 20 MHz system clock, but the PLL is not compatible with input frequencies lower than 16 MHz. However, although the use of a ripple clock is not recommended, it does in fact not affect our digital design, since it is not of notable significance in systems using low clock frequencies such as 20 MHz. Further information regarding PLL can be found in [23].

4. Software Part

The software part of our project consisted of the development of the appropriate high-level design software application to handle the UDP packets (and therefore the audio information) which are being received at the NIC of the PC. We used *Microsoft's* .NET development framework and C# Sharp along with several libraries, including SAPI. C# is a state-of-the-art object-oriented programming language, recently being preferred by many software engineers. The software used to compile and debug the source code was MS Visual Studio 2008.

Regarding our application, the goal was to properly receive the UDP packets and extract the payload, buffer the data and finally forward it to the speech recognition engine. The words which are recognized and converted to text are then checked by an algorithm and eventually mapped to a command. If a command occurs, the relay board is operated via the serial port. Last but not least, the Graphical User Interface (GUI) of our application had to provide some basic information about the states of the devices and also basic options such as the choice of the serial COM port used to connect the relay board.

In order for the application to be efficient, we used four different threads (multithreading) to handle the different operations which are executed. In this way we achieved not to have any latency and/or loss of packets due to processing. Therefore our application may be divided into four different parts, listed in Table 4.1.

1) UDP receiver
2) Speech recognition
3) Background worker
4) Main thread

We will first present a small overview of speech recognition, and then proceed to the description of the four threads. For any further information regarding Microsoft Speech Recognition please refer to [24].

4.1 Speech Recognition

Although speech recognition has been an area of research and development for many years, it was never really appreciated. This was because of its unreliability regarding the recognition success. In the year of 1993 recognition rate was disappointing, about 90% failure. Two years later it was improved to approximately 50% of recognition success, and reached an 80% success rate in 2001. 2001 was also the year when speech recognition used in home automation peaked significantly. Since then several technologies and products using them have been developed, but still this area is concerned to be emerging. Microsoft made speech recognition a part of the mainstream culture when including it in MS Windows Vista in 2007.

The initial failure of success regarding speech recognition can be explained by its complexity. As a comparison, you may consider that even humans can sometimes hardly understand spoken words of another human. For example, if a conversation is taking place in a noisy environment, it is often difficult to clearly hear and understand what the other person is trying to tell you. Many other examples like the previous one could be found to confirm this concept. Now imagine how difficult this operation can get for a system in a digital environment. [25]

One more issue to be mentioned is that in order a speech recognition application to have accuracy and effective performance, it is vital to carefully choose the words used for recognition, since many words may be similar and hard to be separately recognized by the engine. To avoid such problems we have to come up with words/commands that have the same meaning but different phonemes. A good example to describe this issue is the use of the words *on* and *off*, which are totally different but easily confusable by the engine. Instead of *on* we could use words like *open*, *enable* or *activate*, and instead of *off* the words *close*, *disable* or *deactivate*.

4.2 Main thread – Background worker

The main thread may be considered the master thread, which initiates and terminates the sub-threads, creates the GUI and controls the application in general. The background worker helps to update the GUI to avoid the 'freezing' problem[4], which would appear if only one thread had been used. The main job of the background worker is to check for flags which are calling it to update information such as the state of

[4] The freezing problem refers to the inability of moving the application window.

the relays. An issue that we encountered was when we needed to share objects through threads, but we used the invoke method to solve this problem. Regarding the GUI of our application, besides the state of each relay, we also added a Start/Stop button, a Close All button and a COM port list for the connection of the relay board. Finally, a progress bar informs the user whether the threads are running and the application is functional or not. Figure 4.1 shows the GUI of our application.

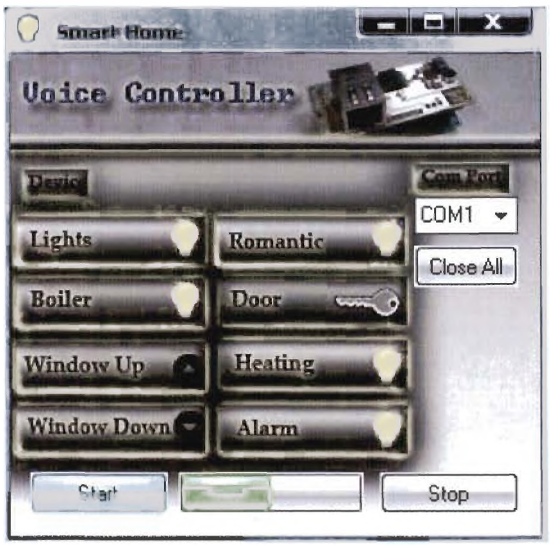


Figure 4.1 – Application GUI

4.3 UDP receiver

In this part each packet had to be received and the payload (512 bytes) had to be extracted and continuously buffered. By using only one buffer to do this, it would be likely the received command to be placed in the middle of the buffer exceeding its size and being cut into two halves, with a result the command not to be finally recognized. In order to avoid this problem, we finally used two buffers with a length of one second each, which overlap each other. Figure 4.2 shows this method.

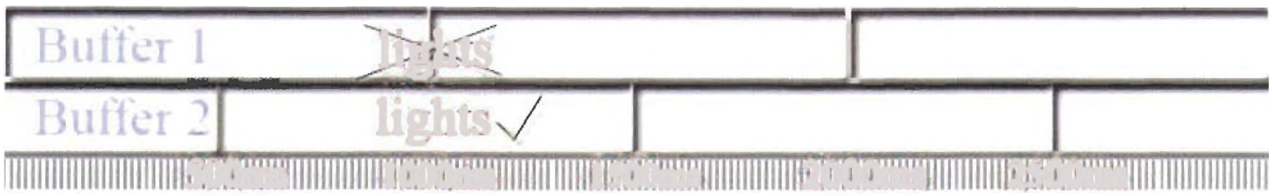


Figure 4.2 – Overlapping buffers

Figure 4.3 shows the flow diagram of the UDP receiver thread.

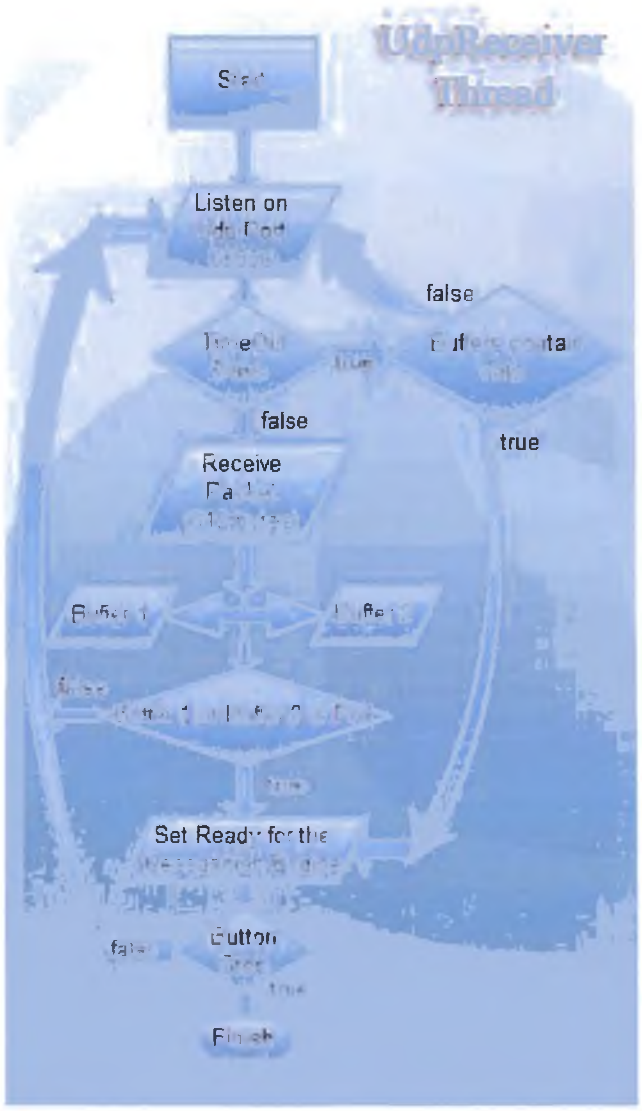


Figure 4.3 – UDP receiver thread

The UDP receiver thread is continuously listening for packets on UDP port 65000. If no packets are received, the timeout of the network socket property is used to catch a timeout exception after 50 ms and to check if the buffers contain usable data. If so, the recognition thread is handling it and if not, the thread is looped and turned into an idle state of waiting for data. When packets are received continuously, the buffers are filled normally and the handling is done regularly by the recognition thread.

4.4 Speech recognition thread

This thread is handling the data whenever the buffers are full, and creates a Memory Stream which is required by the Speech Recognition Engine. When data is being accessed from the buffers, our commands are chosen to build a self-created grammar instead of loading the default grammar, in order to reduce the processing time to less than 1%. After our custom grammar is loaded, the audio stream is created with the characteristics of our samples (F_S 19.500 Hz, Resolution 16 and Channels 1). Last step is to call the Synchronous Recognition method, and when a word is recognized, the *SpeechRecognized* method is called to handle it.

Figure 4.4 shows the flow diagram of the Speech Recognition Engine.

Figure 4.4 – Flow diagram of Speech Recognition Engine

In the *SpeechRecognized* method the mapping of the recognized word (command) with the task assigned to it is done. Some commands need two words while others only one, which makes the decision simpler. Additionally, some commands may be used to both enable and disable the device assigned to it, such as *lights*. For example, by giving the command *lights* once, the task is executed (e.g. lights are turned on) and an internal state of the command is kept. By giving the command again, the task will be executed inversely, due to the state of the device (lights will be turned off). This method uses a case-based algorithm. In order the tasks to be executed we have to communicate with the serial port (user's choice) by sending 3 bytes (255, the number of the relay 1-8 and the state of the relay 0-1). A flag is also set to keep the state of each relay and to update the GUI when any of the relays changes state.

Figure 4.5 shows the flow diagram of the method *SpeechRecognized*.

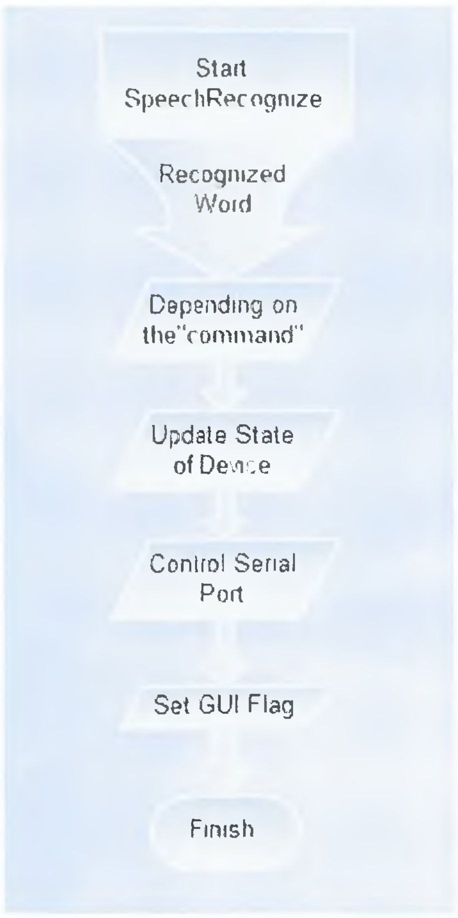


Figure 4.5 – Flow diagram of *SpeechRecognized*

5. Tests and Simulations

The main tests and simulations done throughout the process of our project are briefly presented and described in this chapter. They are divided into two groups, the tests regarding the analog part and those done at the digital part. At the analog part we verified the correct frequency response of the analog filter circuit along with the desired voltage range of the output signal. At the digital part we first verified the correct response of the digital filter. Then we had to simulate and confirm the correct process of accessing the data from the Ram and encapsulating it into the payload. Finally, we captured several packets at the output of the device in order to verify the correct structure of the produced frame.

5.1 Analog circuit

The first simulation regarding the analog filter circuit was to verify the correct frequency response of the output signal, after being amplified and filtered.

Figure 5.1 shows the frequency response of the analog microphone - pre-amplifier - filter circuit. The software used for the creation of this figure was Audition from *Adobe*, although we had done this specific simulation also in Matlab's *dspstfft* tool.

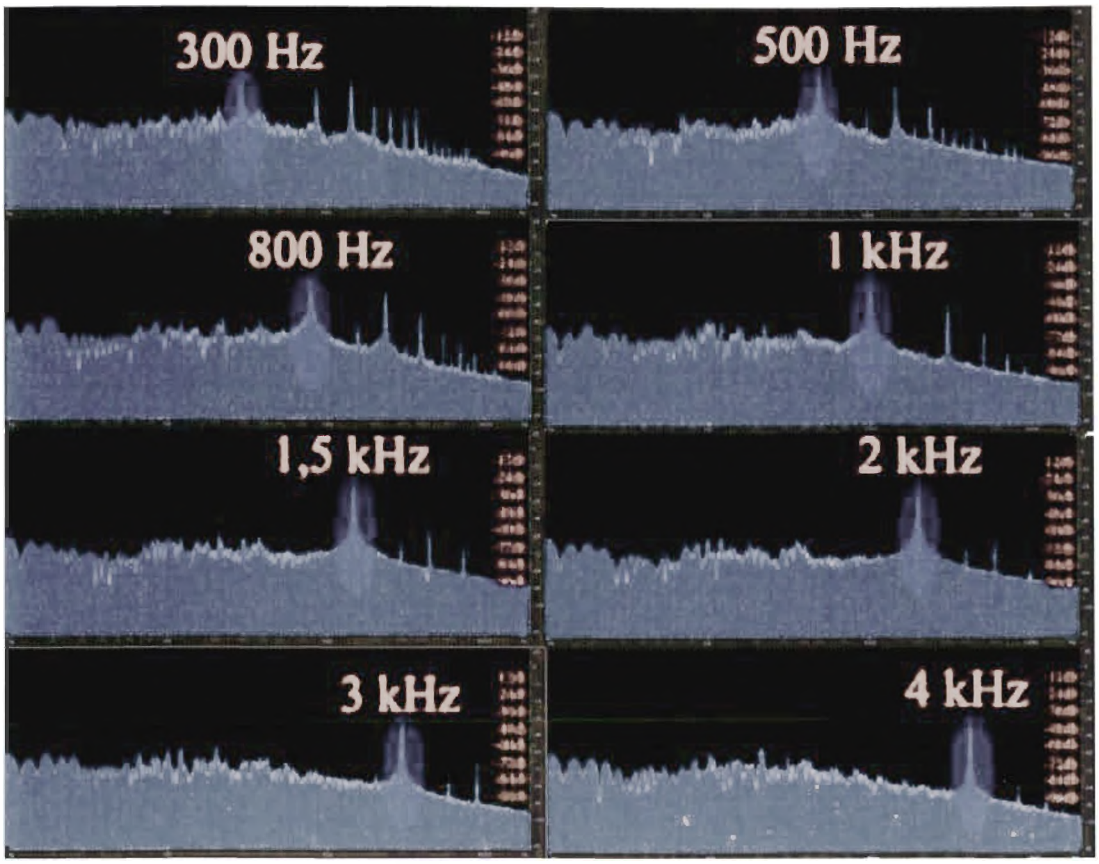


Figure 5.1 – Frequency response

This test was done in order to evaluate the amplified and filtered signal. We recorded the output signal of the circuit using the line-in of the PC, at various frequencies in a range of 300 Hz – 4 kHz. The highlighted spikes in Figure 5.1 show the response of each created frequency out of the average noise level. It can be noticed that the spikes represent the power of each produced signal, beginning with 25 dB SNR at 300 Hz until 50 dB SNR at 4 kHz. The *harmonics*, the spikes seen after the original signal, are referred to as "integer multiples of the fundamental frequency". Harmonics may distort the signal in some cases, but is not of significance in our design. More information about harmonics can be found in [26] [27]. This test showed and confirmed us that the chosen microphone and pre-amplifier plus the analog filter topology that we used were responding properly.

The second test was done to check the voltage range of the signal at the output of the pre-amplifier. This was done in order to regulate the gain of amplification to achieve an output waveform of maximum $\pm$ 200 mV, the recommended input signal voltage range defined by the datasheet of AD7401 [6]. Figure 5.2 shows a real time capture of the data output of the analog circuit (using a digital oscilloscope of *Tektronix*).

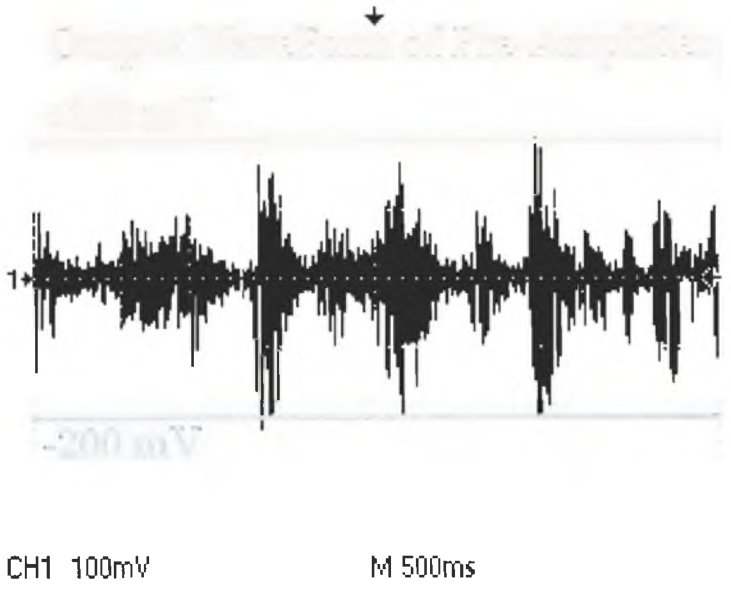


Figure 5.2 – Output waveform of analog circuit

Figure 5.2 shows that the waveform at the output of the pre-amplifier is within the desired voltage range. Each square stands for 100 mV, and the signal hardly reaches four squares, which means that it does not exceed the recommended input voltage range.

5.2 Digital operations

All the simulations regarding the FPGA were done in Quartus-II software, by *Altera.*

The first simulation at the FPGA was to verify the functionality of the digital *Sinc*^{3} filter. In order to achieve this, we referred to the information provided by the datasheet of AD7401, which defines that "a differential input of 200 mV produces a stream of 1s and 0s that are high 81.25% of the time, while a differential input of -200 mV produces a stream of 1s and 0s that are high 18.75% of the time" [6]. The right part of Figure 5.3 was taken from the datasheet and shows these input ranges and the digital (quantization) code produced by the ADC. The left part of the figure shows the simulation which we did in order to compare our results with the numbers given by the datasheet. We simulated one input 81.25% high and another one 18.75% high, and the *Sinc*^{3} filter correctly responded to both (DATA = numbers given by datasheet), as seen in the figure.

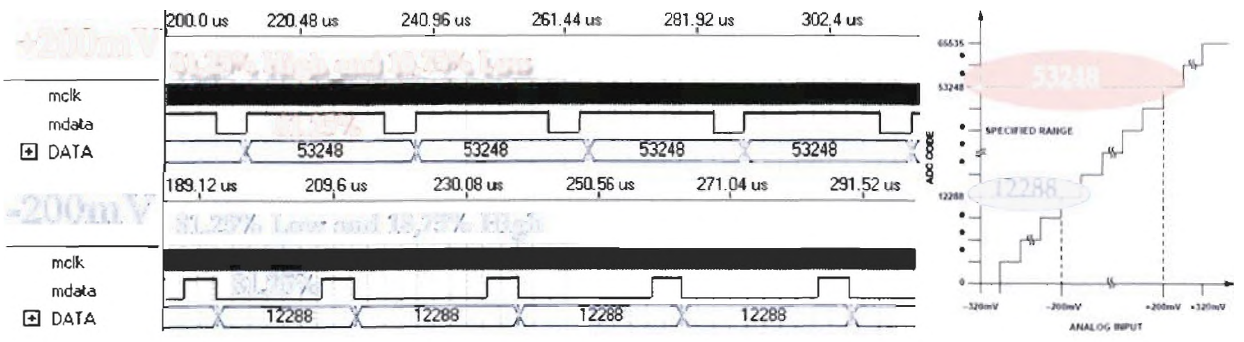


Figure 5.3 – Digital filter response

The next simulation was to check if data is correctly read from the Ram and encapsulated into the payload of the packet. In order to verify the functionality of this operation, we simulated the creation of a packet and checked if the data which was written into the Ram was correctly encapsulated into the payload of the packet. The upper part of Figure 5.4 shows the first two addresses which data (two samples, 4 bytes) is written to (*WRcount*). The bottom of the figure shows these two samples being encapsulated into the first 4 bytes of payload (*pkt_data*). Specifically, Samples 0 and 1 with values 0x82AA and 0xC411 can be seen encapsulated at the first 4 bytes of the payload, but in a different order (*inversely*, Least Significant Byte first), as already explained in chapter 3.

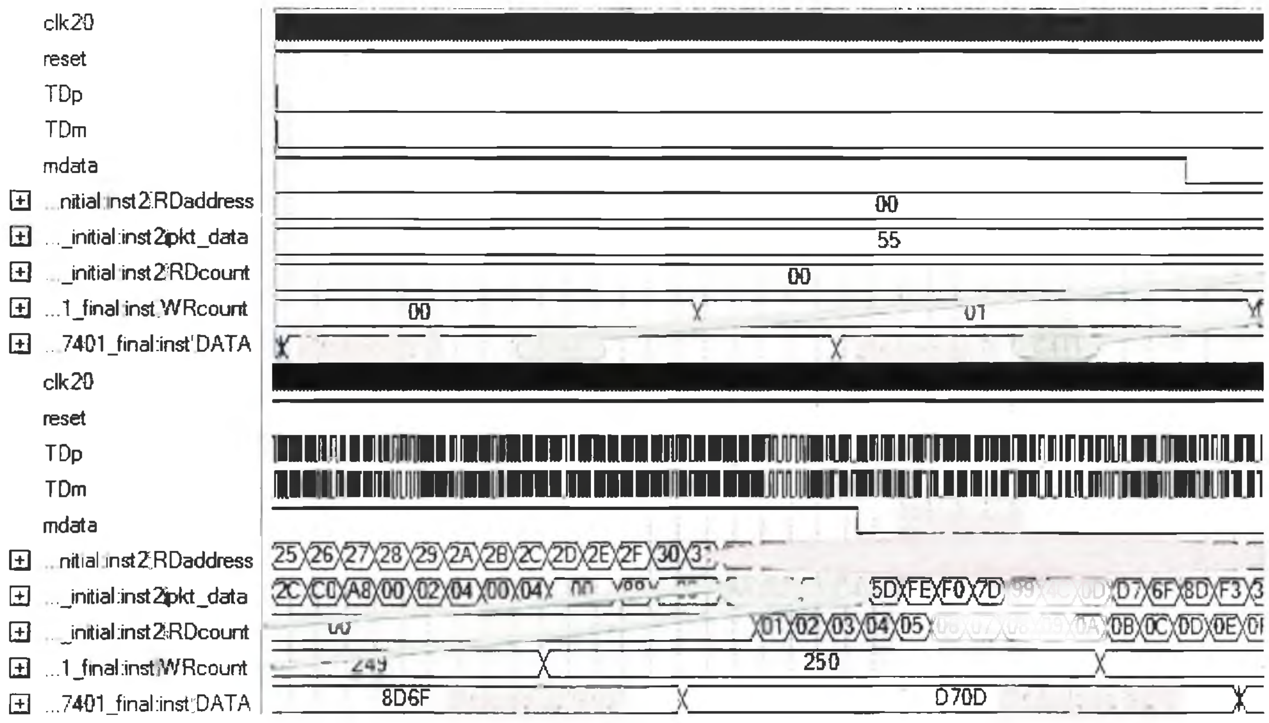


Figure 5.4 – Data access and Encapsulation

The last test was to capture a frame on the medium, in order to verify several characteristics of the packet and its structure. Figure 5.5 shows a frame captured at the output of the FPGA (left) and another one captured at the output of the RJ45, after passing through the transformer. The figure shows that the transformer is turning the bit stream into a differential output, as desired. The figure also shows that the differential output at TX (+) and TX (-) is about ±2.5 Volts, as required by the 10Base-T Ethernet standard [15].

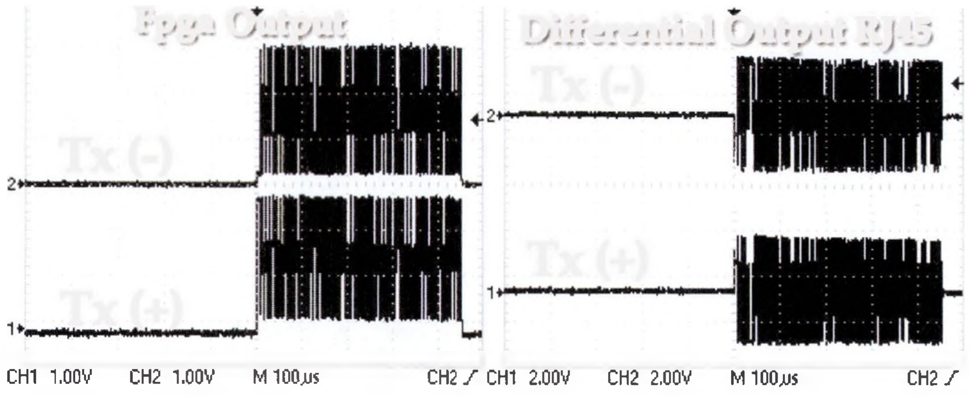


Figure 5.5 – Captured frame

By zooming into this figure, we could also identify several parts of the packet, such as the Ethernet preamble. This helped us to confirm that the produced frame was correctly structured.

Several other simulations were done throughout the design, such as the verification of functionality of the VAD (shut down and re-enable the transmitter) and the correct transmission of NLPs. At the time when the device was transmitting properly, we used *Wireshark* at the PC in order to identify and correct several network-related details, especially regarding the UDP packet-reception at the software part.

6. Conclusion

This project included several parts in a wide area of interest. We talked about home automation in general, several technologies used and tasks being executed in this area of development. We referred to analog electronic issues, such as the design of the PCB and noise cancelling, as well as several components used for the design of the analog part of our device. Regarding the components, we presented an overview of microphones, pre-amplifiers, ADCs and their specifications and the transformer required for Ethernet transmission. We also talked about power supply over Ethernet, know as PoE. Continuing, we talked about digital electronic design on a FPGA, and specifically about the implementation of a digital $Sinc^3$ decimation filter due to the needs of the Sigma-Delta modulator which we used, a 10Base-T Ethernet transmitter as well as several other block designs such as a Voice Activity Detection block. We also briefly presented the requirements of 10Base-T Ethernet transmission. We referred to Speech Recognition and its usage, and talked about our high-level application written in C# Sharp. Finally, we showed some tests and simulations which were done during the process of the project.

The functionality of our project may be considered as complete, since the initial goal was entirely achieved. The end-to-end communication between our device and the relay board controlled by the PC was successfully done. One note to be made is that the realization of the practical part of this project was achieved in a time period of four months.

Regarding the total cost of this implementation, excluding the relay board and considering the FPGA as only the chip, the indicative prices of the main components are the following:

• ECM, Pre-amplifier	– 1 € each
• Oscillator	– 3 €
• ADC, Cyclone EP1C3T100	– 10 € each
• RJ45	– 8 €
• Other PCB costs	– approx. 2 € per PCB

The above price list results in an approximate total cost of 35 € for the realization of this project.

References

References of this report and hyperlinks provided in it – as of July 2010.

[1] Home automation, *Wikipedia*, http://en.wikipedia.org/wiki/Home_automation

[2] Microphone Handbook - Test and Measurement Microphones, *PCB Piezotronics, Inc.*

[3] Preamplifier, *Wikipedia*, http://en.wikipedia.org/wiki/Preamplifier

[4] Walt Kester, *Which ADC Architecture Is Right for Your Application*, Analog Dialogue 39-06, June 2005

[5] Delta-sigma modulation, *Wikipedia*, http://en.wikipedia.org/wiki/Delta-sigma_modulation

[6] Isolated Sigma-Delta Modulator AD7401, *Analog Devices*

[7] Active filter, *Wikipedia*, http://en.wikipedia.org/wiki/Active_filter

[8] Ken Kundert, *Power Supply Noise Reduction*, Designer's Guide Consulting, Inc., Version 4, January 2004

[9] Power over Ethernet, *Wikipedia*, http://en.wikipedia.org/wiki/Power_over_Ethernet

[10] Carrier sense multiple access with collision detection, *Wikipedia*, http://en.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_detection

[11] Manchester code, *Wikipedia*, http://en.wikipedia.org/wiki/Manchester_code

[12] Autonegotiaton, *Wikipedia*, http://en.wikipedia.org/wiki/Autonegotiation

[13] Interframe gap, *Wikipedia*, http://en.wikipedia.org/wiki/Interframe_gap

[14] An Overview of the Electrical Validation of 10BASE-T, 100BASE-TX, and 1000BASE-T Devices, *Agilent Technologies*, Application Note 1600, August 2008

[15] Ethernet over twisted pair, *Wikipedia*, http://en.wikipedia.org/wiki/Ethernet_over_twisted_pair

[16] IPv4, *Wikipedia*, http://en.wikipedia.org/wiki/IPv4

[17] Ethernet, *Wikipedia*, http://en.wikipedia.org/wiki/Ethernet

[18] OSI model, *Wikipedia*, http://en.wikipedia.org/wiki/OSI_model

[19] Cyclic redundancy check, *Wikipedia*, http://en.wikipedia.org/wiki/Cyclic_redundancy_check

[20] *Alex* Urich, Short description of the Internet checksum, http://www.netfor2.com/checksum.html

[21] Dynamic Host Configuration Protocol, *Wikipedia*, http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

[22] Address Resolution Protocol, *Wikipedia*, http://en.wikipedia.org/wiki/Address_Resolution_Protocol

[23] Cyclone Device Handbook, *Altera*, volume 1, May 2008

[24] Microsoft Speech API, *Wikipedia*, http://en.wikipedia.org/wiki/Microsoft_Speech_API

[25] Speech recognition, *Wikipedia*, http://en.wikipedia.org/wiki/Speech_recognition

[26] Harmonic, *Wikipedia*, http://en.wikipedia.org/wiki/Harmonic

[27] Distortion, *Wikipedia*, http://en.wikipedia.org/wiki/Distortion

Appendix A	-	*Filter_Block.pdf*
Appendix B	-	*TxD_Block.pdf*
Appendix C	-	*VAD_Block.pdf*