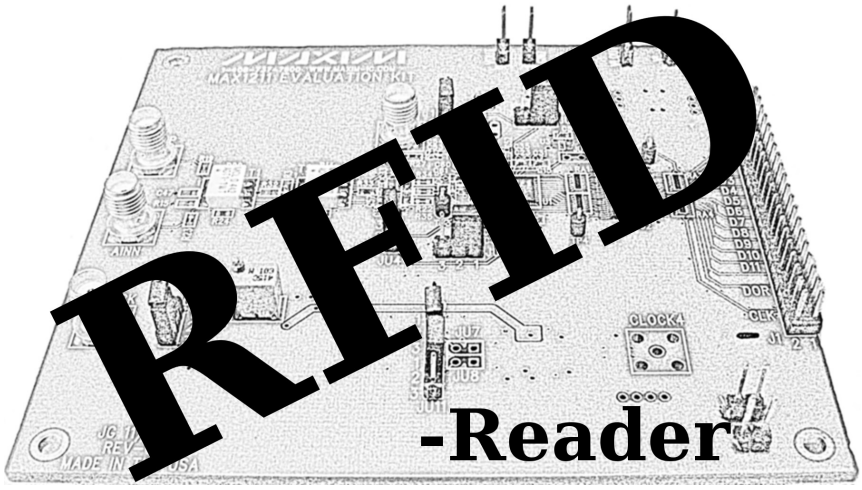
CHALMERS



Implementation of an RFID-reader based on the EGON protocol

Master of Science Thesis in Integrated Electronic System Design

Björn Felber
Shine Vallath Sadhanandan

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, October 2009

Implementation of an RFID-reader based on the Egon protocol

Björn. Felber,
Shine Vallath Sadhanandan

Examiner: Lars Bengtsson

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2009

Sammanfattning

Rapporten fokuserar på implementering av ett nyutvecklat protokoll för energieffektiv RFID-kommunikation. Examensarbetet är en del av EGON projektet, där man har utvecklat detta protokoll. Implementeringen är gjord på ett Virtex2 Pro utvecklingskit med en AD-omvandlare från *MAXIM (MAX1211)* ansluten till data ingången. Projektet innefattar bland annat detektering av frekvenser med hjälp av *Xilinx LogiCORE Fast Fourier Transformer v6.0* och beräkning av tillhörande energispektrum. En tillståndsmaskin(state machine) bestående av sex tillstånd där varje tillstånd består av en sändande och en mottagande fas, används vid implementationen av algoritm för att erhålla rfid-tagens(transponderns) identifikationsnummer. En Microblaze processor implementerad på FPGAN används för kommunikation med dator genom RS232 porten. Ett grafiskt gränssnittet skrivet i C och körs på Microblaze processorn, genom det kan de detekterade taggarnas identifikationsnummer visas i binärt format och man kan också ändra olika parametrar i designen. Det hardvarubeskrivande språket VHDL har använts vid skapandet av hårdvarublocken och EDA (Electronic Design Automation) verktyg från Xilinx (I*SE 10.1* och *EDK 10.1*) har använts vid utvecklingen av dem. Vid simulering av blocken har *ModelSim* från Mentor Graphics använts.

Abstract

This report focuses on the implementation of a newly developed protocol for energy efficient active RFID communication. The master thesis is part of a project called EGON, which is a project where an energy efficient protocol is needed. Implementation is performed on a *Virtex2 Pro* Development board connected to an AD-converter from *MAXIM (MAX1211)*. The project involves detecting frequencies by using the *Xilinx LogiCORE Fast Fourier Transformer v6.0* and calculating the corresponding power spectrum. A state machine is used when implementing the algorithm for receiving the tag (transponder) identification number consisting of six states, where each state has a sending and a receiving part. For the graphical interface a soft-core processor (MicroBlaze) is employed to handle the communication to the computer, which is accomplished through the RS232 port. The graphical interface is written in C and is running on the soft-core processor. It can display tags found as a binary number on screen and it can also be used to change different parameters of the design. The hardware description language *VHDL* is used when creating the hardware blocks needed. In the project EDA (Electronic Design Automation) tools from Xilinx (*ISE 10.1* and *EDK 10.1*) were used for development and ModelSim from Mentor Graphics was used for simulation.

Abbreviation

ADC/ AD-converter	Analog to Digital Converter
EDA	Electronic Design Automation
FFT	Fast Fourier Transform
FIFO	First in First out
FPGA	Field Programmable Gate Array
IP	Intellectual Property
LSB	Least Significant Bit
LUT	Lookup Table
MSB	Most Significant Bit
PLB	Peripheral Local Bus
POSIX	Portable Operating System Interface for Unix
PSD	Power Spectral Density
RFID	Radio Frequency Identification
RISC	Reduced Instruction Set Computer
VHDL	Very high speed integrated circuit Hardware Description Language

Preface

This project is performed by two students and the workload has been distributed equally among them.

Björn Felber and Shine Sadhanandan have a bachelor degree in electronics and this master thesis is the final work of the Integrated electronics and system design master program at Chalmers University.

As both people involved in the project has about the same knowledge in VHDL programming, the parts concerning the hardware blocks were divided equally. When one part was considered finished it was tested and debugged by both parts. This way both students have a good knowledge of the code. Below follows a list of the main author of the different parts of this report.

Topic	Author
Introduction	Björn
Task description	Björn
Hardware	Björn
RFID	Björn
EGON protocol	Björn
FFT-core	Björn
MicroBlaze	Shine
Xilinx EDK	Shine
Xilinx ISE	Shine
ModelSim	Shine
Implementation of the RFID-reader	Björn
FFT	Björn
Power Spectrum	Shine
Frequency Finder	Shine
Clock divider	Björn
Algorithm	Björn
Software Application	Shine
Project Phases	Shine
Tag emulator	Shine
Simulation and debugging	Shine
Simulink model	Björn
Set-up	Björn
Results	Björn Shine
Improvements	Shine

Table 1: Documentation distribution

The list of tasks and the person responsible for them can be seen in Table 2.

Task Description	**Responsible**
AD-sampling	Björn
Algorithm implementation	Björn
Down clocking block	Björn
FFT block implementation	Björn
Frequency detection	Shine
Graphical interface	Shine
MicroBlaze integration	Shine
Power spectrum block	Shine

Table 2: Task distribution

Table of Contents

List of Figures

List of Tables

1 Introduction

In RFID communication a small transponder called *RFID-tag* and a *RFID-reader* is used. It is used in many places today and has a lot of opportunities for new innovative areas; attaching a tag to all articles in a warehouse and easily tracking the inventory is one [1]. The most common version of RFID-tags are the passive ones; meaning that they do not have a power source of their own. This also means they can be kept small, and there is no battery to be worn out. Those tags have the drawback of only operating in really short range. For longer ranges an active RFID tag is needed. Active tags contain a battery, this provides the tag with necessary power to operate at much longer ranges. An obvious drawback here is that the life of the tag ends when its battery is empty; this makes the use of a power efficient algorithm necessary [2]. This project implements a power efficient RFID algorithm on a Xilinx Virtex2 FPGA development kit using an evaluation kit from Maxim for AD-Conversion.

The project started as an idea between Lars Bengtsson at Chalmers University and Björn Nilsson at Halmstad University, the purpose was to create a Master Thesis based on a project carried out in Halmstad in developing a RFID algorithm. The algorithm was already developed, but no real implementation had been performed. Since work already had been started with construction of a tag, the idea was to construct a reader for testing purposes. As the authors of the project are educated in digital circuit design, the main interest was to carry out the project as a FPGA implementation. A meeting was held with Björn Nilsson and Lars Bengtsson to discuss structure and content, giving an overview to the project and what product to expect. Later another meeting was held where the RFID algorithm was explained in more details. As the algorithm is rather simple and relies on detections of different signal frequencies and since FPGA implementation is commonly used in signal processing this seems feasible.

The project of constructing the *RFID-reader* was divided into five phases; the first part involved setting up a basic structure and be able to detect frequencies. As no AD-converter was available in that phase, a file containing signals was used for testing. In the next phase the algorithm was implemented to make use of the detected signals and to be able to retrieve the identification number of the tag. The third and fourth phases are almost like the two first, but instead of using test vectors as input, an AD-converter was used. Each phase involved testing and verifying to make sure that every phase was working probably before moving on to the next. The final testing of the *RFID-reader* involved connecting to a physical tag and receiving the corresponding tag identification number. Because of many uncertainties in the specification, as for example which frequencies to be used and time of the sending and receiving cycles, the reader was constructed with flexibility in mind, making it easy to change parameters in the code through a graphical interface.

2 Task description

The main purpose of this project is to design a prototype *RFID-reader* for the company Lepton Radio AB. The prototype *RFID-reader* should be constructed for an active prototype *RFID-tag* using the EGON protocol developed by the same company. It will not have the final manufacturing specifications as its purpose is to be used for proof of concept. The EGON protocol is a low power dissipation protocol especially developed for active RFID tags and with this protocol the tag should be able to use a printed battery and therefore be able to be made really slim and compact. The implementation is to be performed by using a standard FPGA development kit. Research in signal processing is carried out to be able to make valid decisions on the type of hardware to be used and their features. The main criterias for the project are listed below:

- Be able to detect frequencies up to 30MHz.
- Have a resolution of less than 600 kHz between detectable frequencies
- Work at clock frequency of at least 100Mhz.
- Resistance to noise
- Display found tags on computer monitor
- Be able to handle 48bit identification numbers

3 Hardware

As this project is going to be performed at Chalmers University, it is preferable to use the existing hardware available and only purchase what is not available.

3.1 FPGA Development board

The choice was between a Spartan 3 development board and a Virtex2 Development board. Because of the ability to run the MicroBlaze processor and also better support for the FFT the decision fell on the Virtex-2 Pro Development System from Digilent. It has a built-in clock of 100MHz and 13969 slices on the FPGA [3]. Data taken from the Digilent Hardware reference manual gives the following data (Table 3) for the on board FPGA. For additional memory a 512MByte random access memory was also installed. The board can be seen in Figure 1.

Features	XC2VP30
Slices	13969
Array Size	80 x 46
Distributed RAM	428 Kb
Multiplier Blocks	136
Block RAMs	2448 Kb
DCMs	8
PowerPC RISC Cores	2
Multi-Gigabit Transceivers	8

Table 3: Data of the XC2VP30

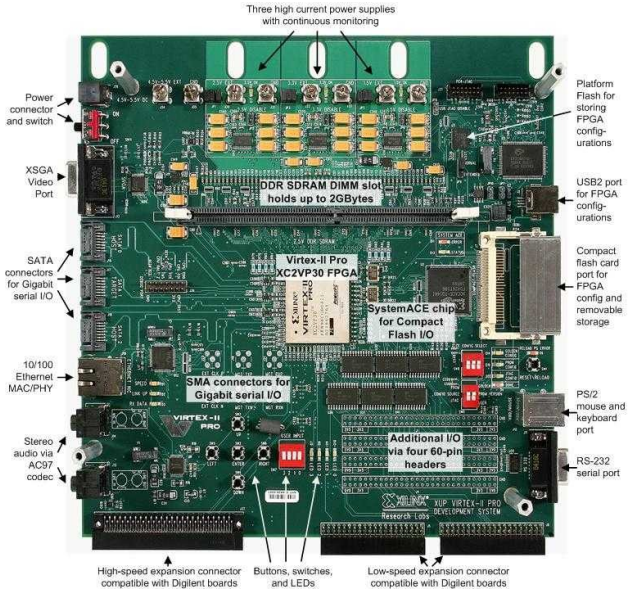


Figure 1: Digilent Virtex-II Pro Development System

3.2 AD-converter

For the analog to digital conversion the AD-converter MAX1211 from MAXIM is used. The AD works at a maximum of 65MHz and has an output bit length of 12 bits. The chip comes mounted on an evaluation board from Maxim. The board has SMC connectors for the input and clock signal, an on-board clock shaping circuit is also available on the board. The output is a 40 pin connector of which 14 pins are used, 12 for data, one for data available (dav) and one for data out of range (dor). A small change had to be done to the board for a single-ended input configuration, how to perform this can be studied in the data sheet [4]. The board can be studied in Figure 2.

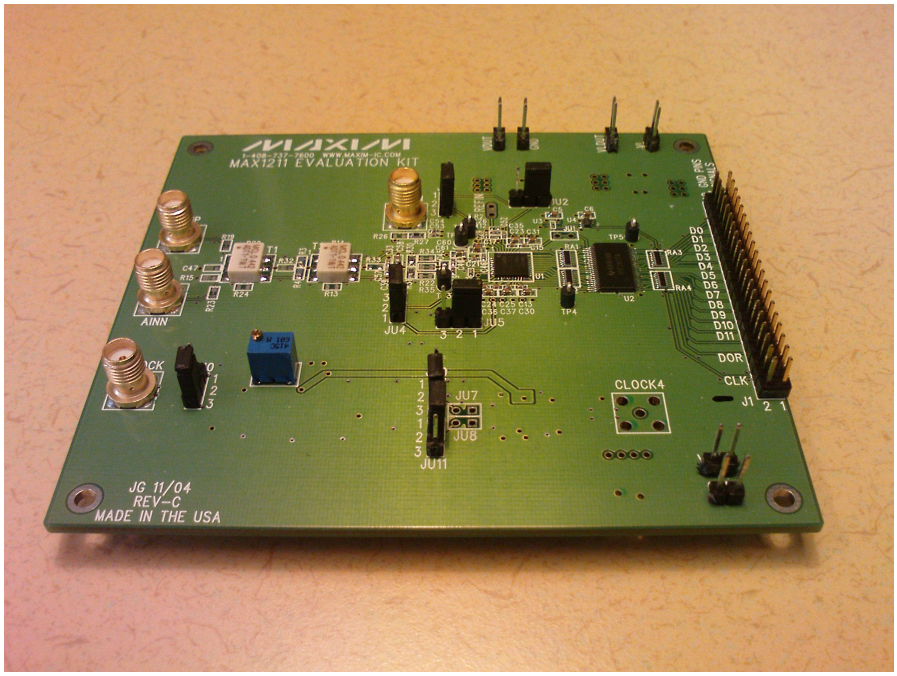


Figure 2: MAX1211 Evaluation kit

3.3 Filter

An analog filter before the AD-converter is needed to avoid aliasing from unwanted signals above 30MHz [5]. In a closed environment like the lab a filter is not needed but almost everywhere else disturbances can interfere with the signal. The construction of the filter was not performed during the time span of this project it was instead left as a later improvement.

4 Theory

In this chapter theory of different parts of the design is explained.

4.1 RFID

RFID is a common name for a variety of different ways of wireless communication between a tag and reader. There are three types of *RFID-tags*, active, semi-active and passive. Active *RFID-tags* have their own power source and usually they have a longer working range a drawback is of course that the size of the tag increases when using a battery. Passive tags do not have a battery and their power is based on induction which charges the tag for a short while. Without a battery the *RFID-tag* can be made really small, the drawback is that the working range is small, only up to a few meters. Semi-active is as the name indicates, a mixture of the two above methods. The *RFID-tag* has a battery but also makes use of induction when operating [2]. This report focuses on the active RFID-tag; the protocol is called EGON and differs from the conventional implementation where a tree structure is used.

4.2 EGON protocol

The algorithm is basically using five frequencies of which one is a beacon. The protocol does not involve modulation of the signals. This excludes the need for modulator/demodulator in the *RFID-tag* and *RFID-reader* which makes the logic much simpler to build.

The beacon signal is used to activate the tags and begin the retrieving of the tag identification number. The other frequencies each correspond to a certain bit number, which can be seen in Table 4.

Frequency	**Bit number**
f1	00
f2	01
f3	10
f4	11

Table 4: Frequency bit

The process starts by sending the beacon. The tag(s) then responds by sending the beacon back. The receiver now knows that there are tags nearby. In the next step the receiver sends on all frequencies, f1, f2, f3 and f4. The tag(s) with the corresponding first two bit-numbers in its/their id sends back on the same frequency.

The *RFID-reader* detects the frequency with the highest energy and sends back on that frequency again. If more than one tag is answering it will only reply the tag with the highest amount of energy in its signal. The tag with the corresponding first two bits will respond with the frequency received.

When the *RFID-reader* receives the frequency, it saves the corresponding first bit of the tag identification number. Then it will send on those frequencies which correspond to a bit-number starting with the second bit of the received signal. The tags with the second and third bit corresponding to the sent signal will answer by sending back at the corresponding frequency and then the second address bit of the tag is received and saved.
This will go on until only the last two bits of the tag identification number is left, the receiver will

then send on all frequencies (f1, f2, f3, f4) and the tags with the corresponding last two bit numbers will answer and in this way at most four different tags will be completed simultaneously. The detected tag(s) will then go to sleep and the next time the beacon is sent they will not wake up and because of that the algorithm will not find the same tag twice. The RFID-reader will keep running the algorithm and sending the beacon until all tags are sleeping and no more tags are detected [1].

The protocol is developed by Lepton Radio AB and is still under development, therefore a more detailed description of the algorithm cannot be given.

4.3 FFT-core

As this project is planning on detecting frequencies using the Fourier transform a digital implementation is needed.

Fast Fourier transform (FFT) is a fast and efficient algorithm for doing digital Fourier transforms (DFT). The FFT can be applied to sample sizes that are a positive integer of the power of two. In equation 1 the definition of the DFT can be observed, where n = 0,...N-1, k=0,...,N-1, N is the transform size and $i = \sqrt{-1}$.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{(-i2\pi k \frac{n}{N})} \quad k = 0, \ldots, N-1 \quad (1).$$

There are different algorithms implementing the FFT, for example, when using the Xilinx FFT core there are two available; Radix-4 and Radix-2. The algorithms can then be implemented using different architectures. When using the Xilinx IP core there are four architectures available; Pipelined Streaming I/O, Radix-4 Burst I/O, Radix-2 Burst I/O and Radix 2 Lite Burst I/FO. The main considerations when choosing one of the architectures are size (resources used on the FPGA) and transform time, where the Pipelined Streaming I/O is the fastest but also the largest, Radix2 Lite Burst I/O the slowest and smallest and the other two are in between. The architecture chosen in this project is the Radix-4 burst I/O because it had a suitable trade of between size and transform time. When using the Burst I/O architecture the data is processed in different stages and grows with each stage, for the Radix-4 the growth is up to 3bits in each stage. To avoid overflow at the output a scaling factor can be set in each stage, which means that the final output will be modified by a factor $\frac{1}{S}$, as seen in equation 2.

$$X(k) = \frac{1}{S} \sum_{n=0}^{N-1} x(n) e^{(-i2\pi k \frac{n}{N})} \quad k = 0, \ldots, N-1 \quad (2)$$

A conservative scaling vector proposed by Xilinx is [01 10 10 10 11], for a FFT of length 512, this means that the scaling in the first stage is 3, 2 in the second, 2 in the third, 2 in the fourth and 1 in the fifth. This scaling guarantees that no overflow occurs at the output [6].

4.4 MicroBlaze

To incorporate the various peripherals with the design and interface it with a computer a processor core needed to be implemented on the FPGA. The major goal of this was to successfully be able to communicate with the co-processor (*RFID-reader* core), set/change parameters and extract the required results. The selected processor architecture should also allow use of peripherals like SDRAM, on-board switches and LEDs. The processor was also to be used extensively for debugging, since it was the only available, efficient option.

There are two processor architectures that can be implemented on the XUP Virtex-II Pro Development Board; MicroBlaze and PowerPC. Since MicroBlaze is the processor that can be

implemented in a wide range of Xilinx FPGA family, it was chosen.

The MicroBlaze(TM) 32-bit soft processor is a RISC-based engine with a 32 register by 32 bit LUT RAM-based Register File. This embedded processor supports both on-chip Block RAM and/or external memory, of which both were used in this project. The MicroBlaze can be connected to a co-processors implemented on the FPGA. The co-processors can be interfaced with it using FSL (Fast Simplex Link) or PLB (Processor Local Bus). PLB v4.6 was used in the thesis project to interface the *RFID-reader* co-processor with the MicroBlaze. This facilitates data streaming between the embedded processor and the required design in hardware.

The MicroBlaze supports a vast array of peripherals to be interfaced with it. In this thesis project, the major peripherals added were RS232 and DDR SDRAM. At various stages of the project development, peripherals such as on-board dip switches, LEDs and push buttons were also added to the MicroBlaze. The processor can also be configured with an on-chip hardware debug module which helps to debug the software on the MicroBlaze [7].

The processor is also capable of running operating systems like μC/OS-II [8] and μClinux [9]. The Xilinx EDK environment facilitates custom C/C++ applications that can be run on the MicroBlaze instead of any operating system.

4.5 Xilinx EDK

The Xilinx® Embedded Development Kit (EDK) is an environment of tools and Intellectual Property (IP) that enables the implementation of a complete embedded processor system on a Xilinx FPGA device. EDK is then further divided into two: XPS and SDK [10].

XPS (Xilinx Platform Studio) deals with the hardware aspect of the project. It supports two processors to be implemented on the FPGA: MicroBlaze and PowerPC, of which the MicroBlaze was used in this project. Through XPS, the MicroBlaze can be configured (clock frequency, peripherals etc) according to the designers need and the hardware specification of the processor can be created. XPS also helps in interfacing co-processors, the custom *RFID-reader* in this case, to the MicroBlaze. For the purpose of interfacing the co-processor, PLB (Peripheral Local Bus) or FSL (Fast Simplex Link) is provided by XPS, of which PLB is used in the project design. After all the hardware configuration is completed in XPS, the specification is converted by XPS to bit file, which can be downloaded into the FPGA board. For the conversion of specification to bit file, XPS makes use of the Xilinx tools for synthesis, translate, map, place and route, and bit stream generation in the background. For the project, the co-processor was designed in Xilinx ISE which also includes all the Xilinx tools. It facilitates as easier design and debugging process. After it was verified in ISE, the design files can be used in the XPS. To successfully complete these processors, various design files has to be modified, details of which is included in the appendix.

Processor	MicroBlaze
Clock frequency	100 MHz
Debug I/F	On-chip H/W debug module
Peripherals	RS232, DDR SDRAM
Standard I/O device	RS232
Boot memory	Local Memory(BRAM)
Co-processor Bus interface	Processor Local Bus(PLBv4.6)
Co-processor IP interface Services	Read/write FIFO

Table 5: Processor, co-processor settings

The SDK (Software Development Kit) provides an environment in which the software to be run on the embedded processor can be designed. It is based on the Eclipse open source suite and enables writing, compiling and debugging C/C++ applications targeted at the embedded processor. SDK includes Standalone and Xilkernel software platform. Standalone corresponds to simple environment with basic features while Xilkernel provides POSIX-style services. A simple Standalone software platform was used for the project which provides input/output operations, access to the processor features and parameters, and basic C/C++ operations [11].

The table, Table 5, gives a brief overview of the processor and co-processor settings.

4.6 Xilinx ISE

Xilinx Integrated Software Environment (ISE) is a software suit which above all provides tools for compiling, synthesizing and floorplanning. It also has various tools for verification such as simulation abilities. Working in ISE usually consist of a number of steps. First of all the design entry is made; here the source files are entered and the structure of the files are viewed. The source files can be written in any hardware description language, such as VHDL, Verilog or ABEL. In the next step synthesis is performed where a netlist is generated from the hardware description language code. The netlist is then used in the implementation process where the netlist is translated into physical file which can be downloaded to the FPGA device [12]. In this project version 10.1 of Xilinx Integrated Software Environment (ISE) was used.

4.7 ModelSim

ModelSim is a verification environment by Mentor Graphics. It supports simulation and verification of multiple languages like VHDL, Verilog and System C. When a design has to be simulated, ModelSim compiles the code using in-built compilers. The graphical user interface (GUI) provided by it allows access to all the inputs, outputs and signals in the design on which numerous debugging options can be applied [13].

This verification environment was used in the project to verify the properties. Test benches, which can be written to automate simulation, were extensively used to verify the functionalities of the design. The version used in the project was ModelSim SE PLUS 6.5.

4.8 VHDL

VHDL is a popular language for describing digital circuits. Many EDA (Electronic Design Automation) tools like Xilinx ISE and Quartus supports VHDL design entry [14]. It supports structural and behavioral descriptions of circuits [15]. In this project, VHDL was used to design all the user-defined hardware blocks.

5 Implementation of the RFID-reader

This chapter will begin with a brief overview of the design used when implementing the *RFID-reader,* explaining the data flow and how all blocks are connected. Later on in the report each block will be explained in more detail.

The EGON protocol is based on a set of five frequencies of which the *RFID-reader* and the *RFID-tags* uses to communicate with each other (as explained in 4.1). To detect tags and proceed through the steps in the protocol, the *RFID-reader* needs to detect the frequency of the sinusoidal wave input from the tags. Since most of the *RFID-reader* design is implemented on the FPGA, signals from the tags are to be sampled through the AD-converter. To detect the interested frequencies in the digitized sine wave, the power spectral density (PSD) was considered as a solution. To compute the PSD, the time domain signal has to be converted into frequency domain, which was done by performing FFT (Fast Fourier Transform). Even though it is possible to conduct DFT (Discrete Fourier transform) such as Goertzel Algorithm for five separate frequencies, FFT allows a simpler IP implementation and more freedom in varying the frequencies in the design [16].

The final implementation of the *RFID-reader* consists of four blocks, FFT, Power spectrum, Frequency finder, Algorithm and Clock divider, as seen in Figure 3. The data from the ADC gets stored and processed in the FFT block and then passed on to the power spectrum block, where the power spectrum is calculated. The frequency finder finds the frequencies and passes them together with the frequencies to be sent to the algorithm block.

The algorithm block is divided into different states. Depending on the current state and the data from the frequency finder block, the block adds bits to the tag identification number and handles the sending frequencies. When a complete identification number is obtained it sends the number to the MicroBlaze which prints it out through RS232. The tag identification number can then be viewed by connecting a computer to the RS232 port and using a terminal application.

To be able to set the timing for the sending and receiving cycles a separate block for the time was created, this block sends the time to the Algorithm and Frequency finder blocks.

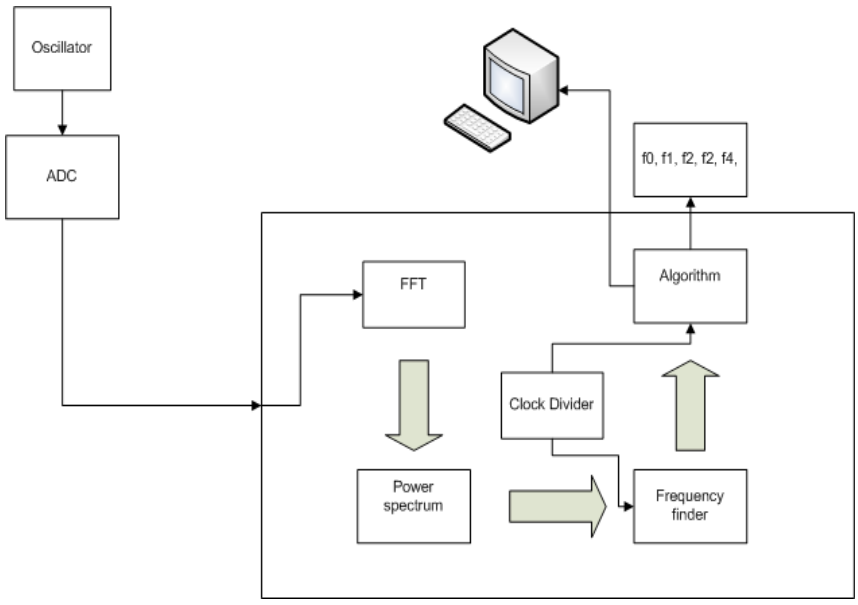


Figure 3: System overview.

5.1 Block descriptions

The VHDL code for the *RFID-reader* is divided into five blocks; the structure can be seen in Figure 4 and the function of each block is described below. The main block acts as a hub in the middle, simply connecting the ports coming from each block to each other.

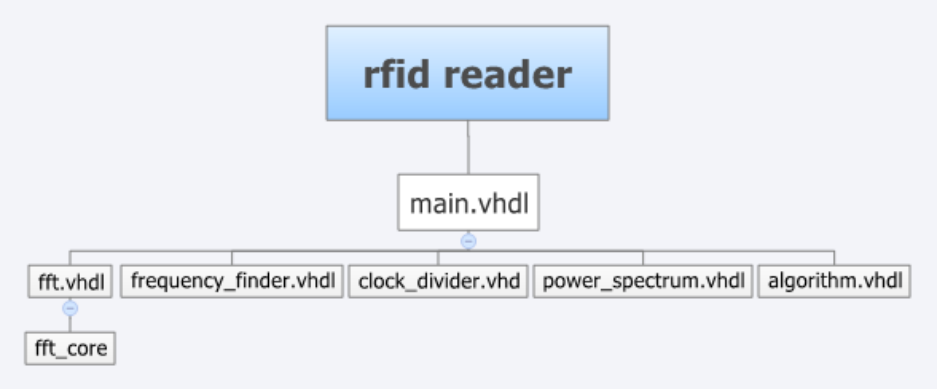


Figure 4: VHDL Structure

5.1.1 FFT

A separate block for the FFT calculation is used with a start signal as input and the calculated FFT data with its corresponding index as output.

The block stores values from the ADC in a buffer of size 512, the same length as the FFT being calculated. The buffer has a pointer which moves one step whenever a data available (dav) signal is received from the ADC, when the pointer reaches the end of the buffer it wraps around and starts from the beginning of the buffer again. Data are then read from the buffer and passed to the real input (xn_re) of the FFT-core.

Port	Type	Description
clk	in	System clock
rst	in	Reset
Start	in	Start calculating FFT
xk_im	out	Imaginary output
xk_re	out	Real output
xk_index	out	Index of output

Table 6: Port description - FFT

The FFT block makes use of a generated FFT-core from Xilinx core library, it uses a length of 512, and because of the data from the ADC being 12bit the input and output bit length of the FFT is set to 12bits. A pointer is used, pointing at the index of the input word; it is set to point at the buffer containing the saved values from the ADC. The FFT-block takes about 5µs (derived by simulation) to complete one transformation, this means that during one listening cycle of 250us, 50 FFT calculations can be performed. The IP core was generated with the settings depicted in Table 7. As mentioned before a scaling vector is used, the vector is directly attached to the FFT-core. The in and outputs of the block can be studied in Table 6.

Function	Setting
Implementation	Radix-4, Burst I/O
Transform length	512
Channels	1
Data Format	Fixed Point
Input Data Width	12
Phase Factor Width	12
Scaling	Scaled
Rounding Modes	Truncation
Output Ordering	Bit/Digit Reversed Order
Data	Block RAM
Phase Factors	Block RAM

Table 7: Configurations of FFT core

5.1.2 Power Spectrum

The primary function of the *Power Spectrum* block, is to compute the power spectral density of the FFT output. Table 8 depicts the port description of the *Power Spectrum* block. The power spectral density (PSD) can be calculated using the following relations:

$$X = FFT(Y)$$

$$PSD = \frac{X.\bar{X}}{number\ of\ FFT\ points}$$

Port	Type	Description
clk	in	System clock
re	in	Real output from FFT
im	in	Imaginary output from FFT
valid	in	Valid flag from FFT
index	in	FFT output index
index_out	out	FFT output index
valid_out	out	Valid flag from FFT
com_conj	out	$re^2 + im^2$

Table 8: Port description - Power Spectrum

To simplify the process and to minimize the hardware utilization, the division by number of FFT points in the PSD equation was omitted. This means that the results will be amplified by number of FFT points, and since this signal is dealt internally between two blocks the extra logic would seem unnecessary. The inputs im and re from FFT block is manipulated in the following way to obtain the com_conj:

$$com\,conj = real + i.imaginary \cdot (real - i.imaginary)$$
$$= real^2 - i^2 \cdot imaginary^2$$
$$= real^2 + imaginary^2$$

5.1.3 Frequency Finder

Frequency Finder is the block succeeding *Power Spectrum*. As the name suggests, it is designed to detect the required frequencies from the FFT output. A threshold (*threshold*) is set for the frequency detection. To detect the frequencies, the position of the required frequency in the spectrum has to be determined. These positions can be calculated using the following formula:

$$Frequency\ position = \frac{length\ of\ FFT}{Sampling\ frequency} \cdot frequency$$

Port	Type	Description
clk	in	System clock
toggle	in	Toggle from clock divider
valid	in	Valid from FFT
threshold	in	Threshold for freq detection
time_us	in	Time in us from clock divider
com_conj	in	com_conj from *Power Spectrum*
index	in	FFT output index
f0_position	in	Index of f0 in FFT output
f1_position	in	Index of f1in FFT output
f2_position	in	Index of f2 in FFT output
f3_position	in	Index of f3 in FFT output
f4_position	in	Index of f4 in FFT output
toggle_out	out	Toggle from clock divider
time_us_out	out	Time in us from clock divider
tag_bit	out	Detected tag bit
frequency	out	Detected frequencies from tags
send_freq	out	Reader output freq for next state(except state 3)
state_3_frequency	out	Reader output freq for state 3

Table 9: Port description-Frequency Finder

Using the above relation, the expected position of all the five frequencies involved in the design (beacon and four frequencies) were calculated and in the C application and relayed to the *frequency finder* (f0_position, f1_position, f2_position, f3_position and f4_position). If the computed position matches the index, and if the com_conj value for the index is greater than threshold when toggle is high (receiving stage), the corresponding frequency bit is set high.

The output *send_freq* is based on the frequency with the highest power among the detected frequencies, except when the beacon is encountered. An internal signal keeps track of the highest power among the detected frequencies and based on that *send_freq* is calculated according to the Table 10.

Frequency(highest power)	send_freq
f0	f1,f2,f3,f4
f1	f1,f2
f2	f3,f4
f3	f1,f2
f4	f3,f4

Table 10: send_freq logic

state_3_frequency is an output closely related to *send_freq*, which is specifically designed for use in state 3 in the state machine in the algorithm block. The logic associated with the signal is the same as *send_freq* except that it will only give one frequency to be sent in state 3. The *state_3_frequency* is assigned according to the table below, based on the highest power frequency detected. It should also be noted that this output is not related to f0, the beacon.

Frequency(highest power)	state_3_frequency
f1	f1
f2	f4
f3	f3
f4	f4

Table 11: state_3_frequency logic

tag_bit is a two bit std_logic_vector. The LSB is the valid bit and MSB being the detected tag bit. This signal is also based on the detected frequency with highest power. If a valid bit is found based on the detected frequencies, the LSB of tag_bit will be set high and the MSB will be set as per Table 12.

Frequency(highest power)	tag_bit<1>
f1	0
f2	0
f3	1
f4	1

Table 12: tag_bit logic

If a valid bit could not be found, the LSB will be set to '0', pronouncing the MSB to be invalid. This signal is further used in algorithm, to detect the tags and also to detect if the tag is set to be valid.

5.1.4 Clock divider

The EGON algorithm is divided into two parts, sending and receiving, of which, using standard configuration, the first occurs for 200µs and the second for 250µs. For that reason an implementation of a separate block handling the sending and receiving time is a solution. The *clock divider* creates a clock with a period of one microsecond and from that an asymmetrical clock pulse with a period of 450µs is made (*toggle*). The pulse is low for 200µs and high for 250µs. An output for the time in microseconds will also be available, which counts from 1µs to 200µs or 1µs to 250µs depending on if toggle is low or high. The block is made in such a way that changing of sending or receiving times requires little effort as those are implemented as inputs which can be modified through the user interface. A description of the ports can be found in Table 13.

Port	Type	Description
clk	in	System clock
rst	in	Reset
out_time	out	Output time
toggle	out	Pulse of send and receive time
time_send	in	Input for setting the sending time
time_receive	in	Input for setting the receiving time

Table 13: Port description - Clock divider

5.1.5 Algorithm

To have good overview of the code, different processes were used for different parts of the implementation of the algorithm block. A description of each process can be studied below and a descriptions of the ports can be viewed in Table 14.

Port	Type	Description
clk	in	System clock
rst	in	Reset
frequency	in	The detected frequency
stage_3_frequency	in	Frequency to be sent in stage 3
tag_bit	in	The bit number for the found frequency
send_frq_in	in	The frequencies to be sent
time_us	in	Current time
toggle	in	Pulse with send and receive cycles
time_receive	in	Receiving time
send_f	out	Frequencies which are sent
tag_output	out	Found tag output

Table 14: Port description - Algorithm

State machine

The *algorithm* block consists of a state machine with six states. In Figure 5 all of its states can be observed. Depending on which states the state machine is in, different operation will be performed. Each state lasts for 450µs of which 200µs is the sending part and 250µs the receiving part. On reset the state machine will go to state1 and then start looping through the states. It will continue looping forever as long as it gets the beacon. If no beacon is detected, it will stay in state1 and continue to send the beacon. The beacon is sending for 200µs, then it waits for 250µs, if it gets the beacon back it will continue to the next state, else it will send the beacon again. When in the second state all frequencies except the beacon are sent. The frequency with the highest amount of energy received is then sent in state 3. This is also where the first bit of the tag is received. In state 4 and 5, two frequencies will be sent based on the received frequencies of the previous state. In each of these states one bit of the tag identification number will be added. When there is only 2 bits left it will go to state 6 and send on all frequencies. It will then concatenate the already received tag identification bits with all combinations of the two last bits, and also a valid bit. The valid bit will be put to zero if at some point no frequency is detected.

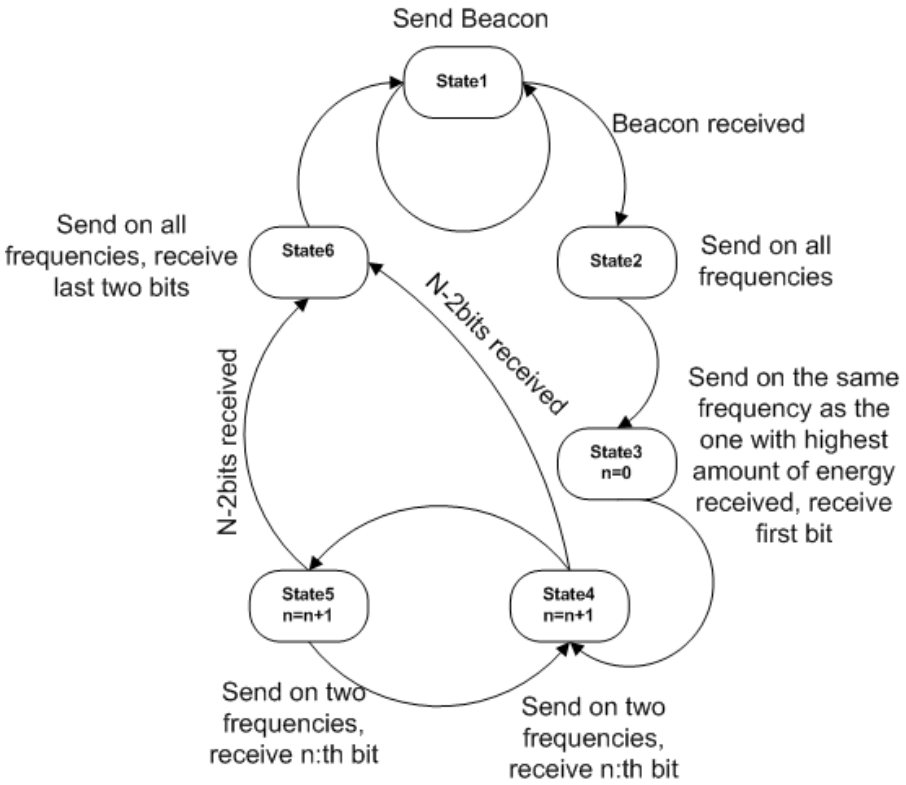


Figure 5: State machine where n is the bit being received and N the total length of the tag identification number.

Sending

A single process handles the sending part. Based on the current state and toggle, it decides when to send. In state 2 and 6 the sending frequencies are constant. In state 3 to 5 the sending frequencies are based on the frequencies detected and passed from the *frequency finder* block

Counter

To be able to know how many bits have already been processed a counter process is implemented, the process counts each time the state machine visits state 3, 4, 5 and 6.

Tag output to MicroBlaze (tag_output_pro)

To be able to read the tag in the MicroBlaze, the tag identification number has to be available at the output. This is done by putting out the tag id vector on a bus available for the MicroBlaze. Because of limitations in the MicroBlaze only 32 bit numbers can be transferred, this is solved by dividing the 48 bit tag into two parts and sending them one by one. The tags are then converted into binary format by a program running on the MicroBlaze and displayed on the screen.

Main

The main process handles the state transitions of the state machine. Each transition occurs at the positive clock edge.

Write bits (tag_temp_pro)

Based on the value of the counter, this process adds the corresponding bit from the received signal to the tag's identification vector. The vector will be updated during the first microsecond in the sending sequence after the listening sequence. This vector contains all the tag's identification number bits except the two most significant, which will be appended in the *tag_app_pro* process in the end.

Appending last two bits and valid bit (tag_app_pro)

When the tag id vector is completed, this process appends all combinations of the two most significant bits and also an extra bit at the end (lsb) for validation. This creates four different tag identification numbers, with all of the bits being the same except for the two first (msb) bits. The value of the validation bit is passed from the frequency finder block. An example of how 8bit tag identification vectors can look like can be found in Table 15. In the example only one tag is valid, 00101010, which has the valid bit set to 1.

Complete tag id with valid bit		
Tag id last 2 msb	**Tag id**	**Valid bit**
00	101010	1
01	101010	0
10	101010	0
11	101010	0

Table 15: Tag id example

5.2 Software Application

A software application is necessary to be executed on the MicroBlaze processor to provide various functionalities. The application must be given access to the required co-processor parameters which has to be modified or read. This is done indirectly through the read/write FIFO of the PLB interface, which is connected to the required input/output ports of the co-processor.

In the C/C++ application for the thesis project, the software is given access to the read/write FIFO. Through this the necessary parameters of the system designs can be changed easily. During the testing and verification stages of the design this was extensively used. Various parameters of the *RFID-reader* design including threshold, reset and start were connected to the write FIFO, so that these values could be changed from the software. To check the various components on hardware, any of the required outputs to be checked was connected to read-FIFO and was accessed through the C application. The standard input/output device of the MicroBlaze was configured as the

RS232. Using the serial port, the outputs from the software application was displayed in *Tera Term*, a terminal emulator [17].

In the earlier stages, the *RFID-reader* design included an tag emulator embedded within, which is further explained later in the report. This tag emulator was effectively used for testing, using the C/C++ application on the MicroBlaze. The tag emulator emulates a tag whose identification number is given as an input through the C/C++ application. The C/C++ application generates the tag identification number exhaustively, based on the frequency vectors in memory supplied as input to the FFT. To test the correctness of the system, the detected tag id from the co-processor is then compared to the application generated tag id.

The final version of the project software manages various inputs and output of the co-processor as specified in Table 17.

Co-processor parameter	Type
threshold	input
f0_position	input
f1_position	input
f2_position	input
f3_position	input
f4_position	input
time_receive	input
time_send	input
tag_output	output

Table 16: Software accessible parameters

As threshold is an input, this gives the possibility to change the threshold when the experimental setup is changed. The frequencies associated with the design (f0, f1, f2, f3 and f4) can be changed through the software application. The corresponding index of the frequencies (f0_position etc) can be calculated and passed through the PLB. The sending and receiving cycle duration of the protocol can also be altered through the C application.

The detected tags (tag_output) are read through the read FIFO. This data is initially represented in decimal form, and a separate coding is required to represent in binary format, since the print function in the C/C++ library does not support binary representation.

Limitations: The major limitation of the software is its inability to keep up with the speed of the hardware. For longer sending and receiving duration between *RFID-reader* and *RFID-tag,* the software will have enough time to process different computations needed, such as binary conversion and printing of results. But as the sending and receiving times become shorter, the software might not be able to complete the required processes in the given time. This leads to loss of data. The above stated limitation does not occur for the specification of this project.

The other limitation is related to higher tag identification number length than the lower one. The macros and functions defined in the C/C++ libraries in EDK are targeted at 32 bits and lower. When the tag address is more than 31 bits (+ 1 valid bit), these functions are unable to relay the output from co-processor to Tera Term. To solve this the tag id was separated as follows: If the tag id is 48 bit long (49 bits including the valid bit), the tag address is separated as 32 bits and 17 bits. This data is sent to the MicroBlaze and the software prints it as a single address.

5.3 Project Phases

The project was divided into different phases, each having a definite objective, to systematically fulfill the specifications. Since the A/D converter was not available at the initial stages of the thesis, the first two phases dealt only with the digital realm. The two final phases introduces ADC into the design, and can be considered as modified version of first two phases.

5.3.1 Phase I

The objective of this phase was to implement an FPGA design which could detect five specific frequencies. This was necessary for the later stages, in which the implemented protocol works based on the detected frequencies.

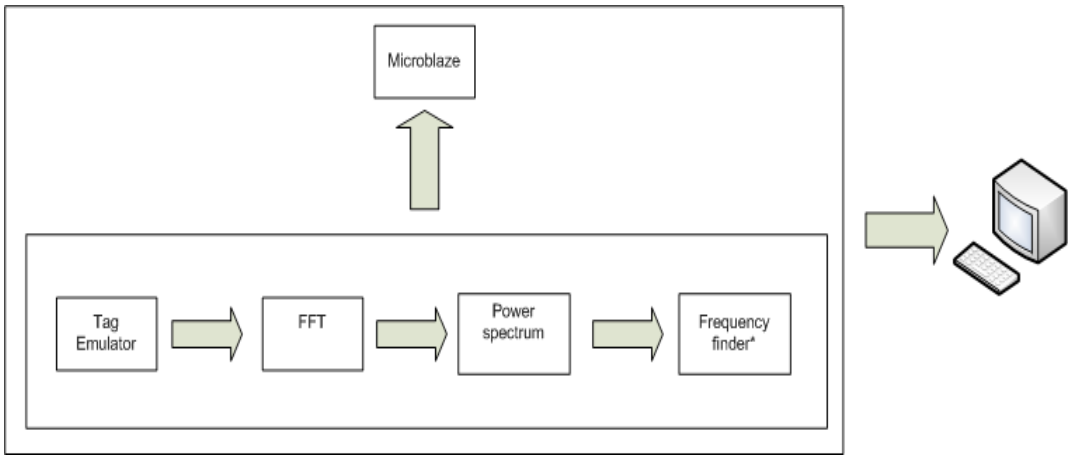


Figure 6: Phase I block diagram

The design flow can be seen in Figure 6. The 512-point FFT block receives the sampled input data on which the FFT is calculated. This data is sent to the *power spectrum* block to deduct the power spectrum, which is followed by *frequency finder*. The *frequency finder* block used for this design is a simpler version than that in final design. This version was designed to detect the presence of any of the five frequencies specified. Based on the power spectrum data, the *frequency finder* block detects if the frequency is present using power spectrum value corresponding to the respective position. The design was implemented as a co-processor with the MicroBlaze processor, interfaced through PLB. *Threshold* and the *frequency finder* output were connected to the PLB. A C-software application was designed to modify the *threshold* through the terminal. The software application was also designed to access the *frequency finder* output and notify the user if any of the frequencies were detected.

The tag emulator was included in the project to provide the FFT block with a sampled input, which could also be controlled through the terminal.

5.3.2 Phase II

In phase II, the complete digital aspect of the thesis was covered, as seen in Figure 7. The tag emulator was used to feed input to the system and test the functionality. The FFT of the input data is calculated and as in Phase I, this is further processed in *power spectrum* block. The *frequency finder* detects the frequencies and computes various signals required in the algorithm block. The *frequency finder* used in this phase is the final version explained in the earlier sections. The *clock divider* unit

controls the timing of the sending and receiving cycles of the *RFID-reade*r. Based on the signals from the *frequency finder*, the algorithm block determines the tag address and frequencies to be transmitted in the next cycle.

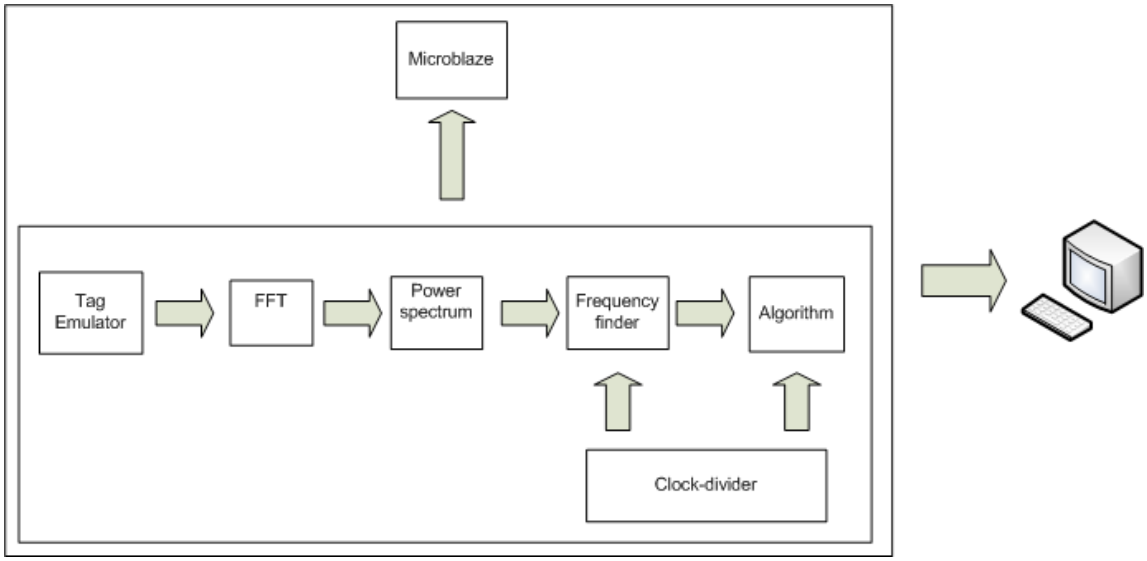


Figure 7: Phase II block diagram

As seen in Figure 7, MicroBlaze processor was implemented in FPGA and the reader design was connected using PLB, with access to various design parameters. This is to enable easier system modification by the user, which was enabled through the C application run on the MicroBlaze.

5.3.3 Phase III

This phase introduces the ADC to the project scope. The objective of this phase was to implement the ADC in the phase I design. As a result, the tag emulator was removed and the ADC was interfaced to the design.

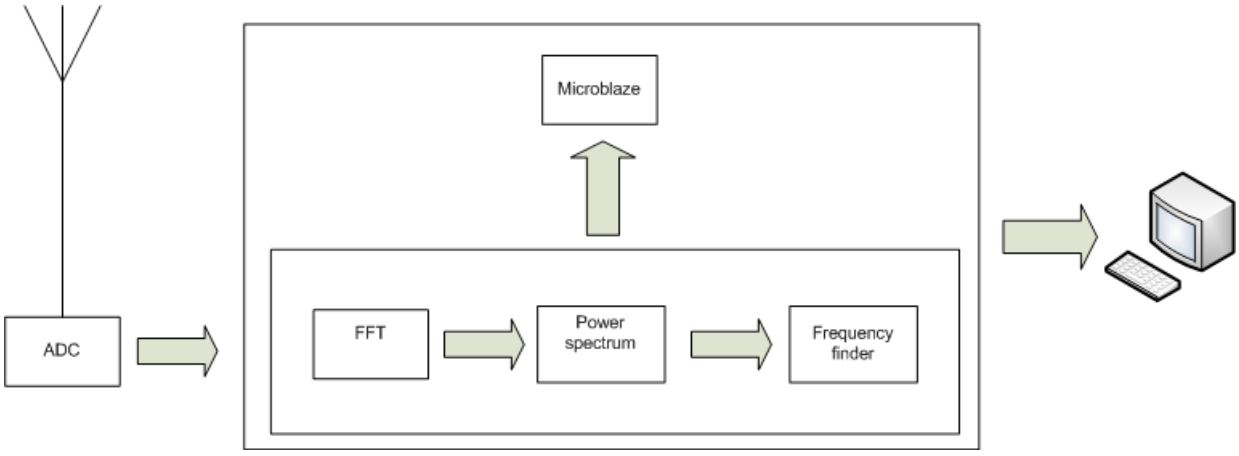


Figure 8: Phase III block diagram

Since the tag emulator was designed to represent the ADC, with similar 12-bit output, the processes involved were similar to that in phase I. Figure 8 depicts the block diagram for this phase. The design was tested using frequency generator and the prototype *RFID-tag*.

5.3.4 Phase IV

The fourth phase of the project was intended to fulfill all the design specifications. In this phase, the design in phase II was upgraded to include the ADC. The digital part of the design is similar to that in phase II except the removal of tag emulator. The 12-bit ADC feeds the input to the FFT block. The design flow can be seen in Figure 9.

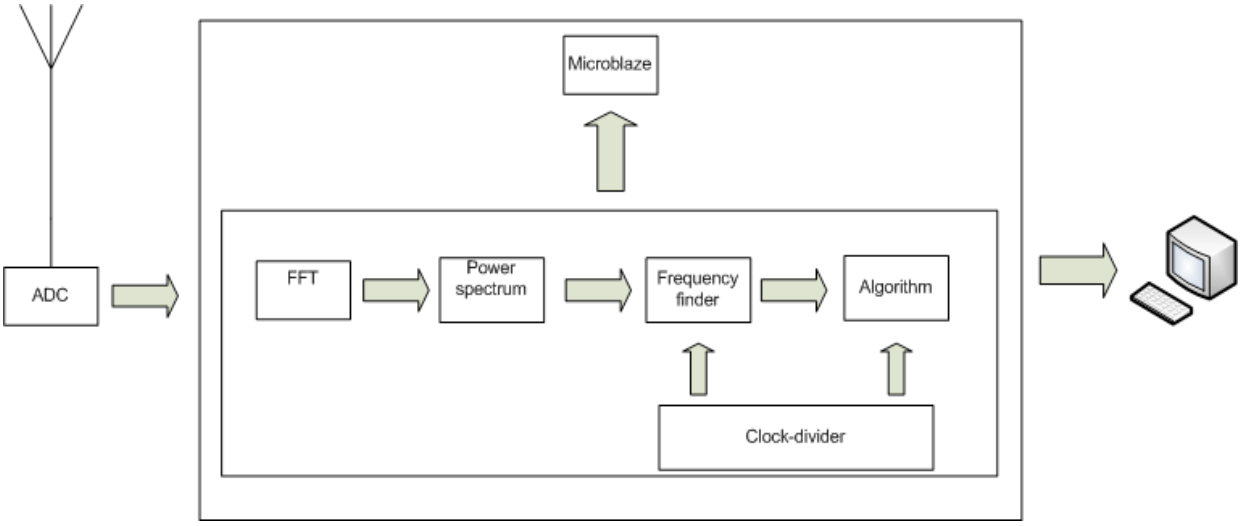


Figure 9: Phase IV block diagram

The *RFID-reader* design was connected to the MicroBlaze through PLB. For the purpose of easy and quick system modification, the MicroBlaze was given access to the following parameters: threshold, sending and receiving duration and the frequencies associated. The C application was modified to access these parameters, giving the user options to modify it. The design was tested using frequency generator and the prototype *RFID-tag*

5.4 Tag emulator

To verify the *RFID-reader* design, a tag based on the EGON protocol was necessary. Since it was not available, a tag emulator was required to test and debug the design. This was incorporated within the whole system. The emulator was not implemented as a separate functional unit, but as a design that spreads across different components of the system. It was implemented mainly within the *algorithm* block and *FFT* block.

The *tag emulator* was used in the earlier designs which did not include the A/D interface. It was completely within the digital realm. A file was included to the design which contained sampled values of sine waves of the following frequencies: 20MHz, 22MHz, 24MHz, 26MHz, 28MHz and noise. The frequencies involved with the *RFID-reader* were set as in Table 17.

Frequency variable	Frequency(MHz)
f0	20
f1	22
f2	24
f3	26
f4	28

Table 17: Tag emulator frequencies

To simulate the presence of tags, the tag addresses to be detected was embedded to the C application on the MicroBlaze. A function was created in the application which was to generate the tag addresses systematically, and to break down this tag addresses to the frequencies involved in it. This frequency information was sent to the *RFID-reader* co-processor. Based on this information, the tag emulator sends the appropriate sampled sine wave values to the *FFT* block at various algorithm stages. After the whole cycle to detect a single tag was completed, the tag address detected by the design, based on the frequency information from MicroBlaze, is sent to the MicroBlaze C application and cross checked with the expected result.

Limitations

Since the *tag emulator* was intended for preliminary testing only, it was not designed with general use in mind. As a result it was spread across various blocks in the design, which makes it harder to modify without detailed knowledge of the *tag emulator* implementation.

6 Simulation and debugging

In this project, testing and verification were mainly conducted through the MicroBlaze. In the early stage of the design, to simulate and verify the functionalities of the FFT IP block from Xilinx, a Simulink model was used. Sampled frequency data was generated using MATLAB and sent to the FFT block. The output was captured and studied. Based on these tests, the FFT block was calibrated in the VHDL design.

The Phase I of the project, which involved detecting a frequency based on the sampled frequency-data in file, was simulated using test benches in ModelSim. The input data file consisted of seven sets of data; combination of four frequencies and noise. To test this on the board, initially, the ISE project bit file was downloaded to the FPGA board. Switches were assigned to select the input data corresponding to a frequency or set of frequencies. LEDs were mapped to light up if a frequency corresponding to one of the four LEDs were detected. To ease the testing and debugging process a MicroBlaze processor was implemented on the FPGA with Phase I design being the co-processor. This allowed the selection of data sets through the terminal. This was also useful in debugging since any signal in the design could be displayed through the terminal.

The second phase of the project involved the implementation of the whole *RFID-reader* without the A/D interface. The design also included the tag emulator. This design was simulated in ModelSim using various test benches and verified to a certain extent. Due to the sheer number of signals involved in the design, simulation and waveform study proved difficult, but manageable. Similar to Phase I, MicroBlaze was used to debug the design on the FPGA board. Initial testing showed that the simulation results varied from the on-board results. To rectify the unexpected behaviors, properties of each block were scrutinized. For this purpose, various signals associated with the blocks were connected to the MicroBlaze and made available through the terminal. The majority of the problems were detected to be related to the algorithm design file. Timing issues with the state machine and design interpretation by the tools contributed to abnormalities. The functions involved in the algorithm were all treated as separate processes, which rectified most problems. The operating frequency of the design was lowered, since the C applications had limited processing power, to observe the changes more clearly. Based on these tests, the algorithm was fixed. To verify this design phase, the tag emulator was designed to produce 2401 tag instances and verify the design outputs, which was successfully completed.

The third phase of the project was the A/D interfaced version of Phase I design. The design did not include the sampled data file for different frequencies; but used the input as the data from the A/D converter. To test the design, a signal generator was used, which was connected to the ADC input. The frequency of the signal generator was varied and the user was notified if any of the interested frequency in the design was detected, through the MicroBlaze. Since the FPGA design was very close to that in Phase I, most of the issues were related to the ADC and interfacing it with the FPGA. Signal integrity was a major problem, which was improved by using a 40-pin flat cable instead of 20-pin flat cable.

In the fourth phase, the second phase of the project was modified to include the ADC. Testing of this modified design proved difficult due to that no physical tag was available at this time. As a result initial testing revolved around the beacon signal. Since the design proceeds through all the stages if a beacon signal is detected, producing a valid/invalid tag address, it could be tested if the system detects a beacon. The frequency associated with the beacon was varied, and a signal generator was used to test the expected functionality. While sweeping the frequencies, it was noticed that beacon was detected at unexpected frequencies. This was rectified by determining the optimal threshold for the experimental settings.

When connecting different kinds of hardware, it is important to know how each device is functioning, including the ADC used in this project. To test the AD-converter a code was made for printing out the sampled values. Because a set-up of the MicroBlaze processor had already been done, implementation of such code was rather easy. With the AD-converter connected to a signal generator producing a sine wave at 10MHz, with a peak to peak value of 300mV, the values from the 12bit output of the AD-converter was captured and stored by the Virtex2 board. The values were then sent through the RS232 port to a computer and stored to a file. MATLAB was later used to plot the values (Figure 10). The data coming from the AD is in 12 bit two's complement form. This means that the data range is -2048 to 2047. As seen in Figure 10 the data range of the captured values is much smaller, since the number of bits used is only 10. This is as expected, as the full input range of the ADC is 1 V_{P-P} and only 300mV was used.

In MATLAB, the built-in FFT function was used to calculate the FFT of the captured signal. Using the known sample rate of 64MHz, the power of the FFT was plotted with a correct scaling on the frequency axes. As expected, most of the energy was found around 10MHz as seen in Figure 11.

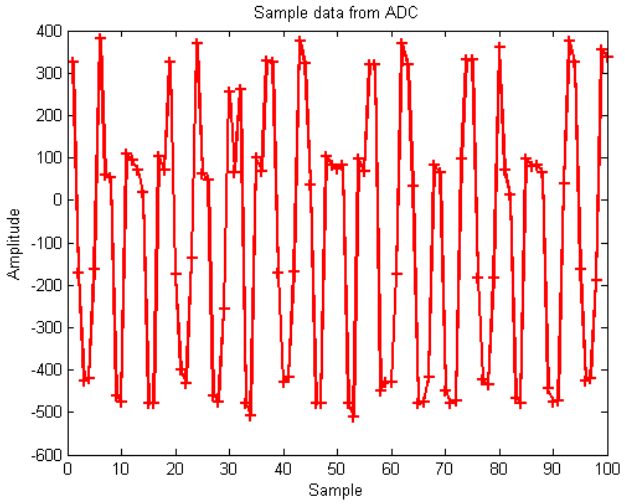


Figure 10: Samples captured from the ADC at an input signal of 10MHz.

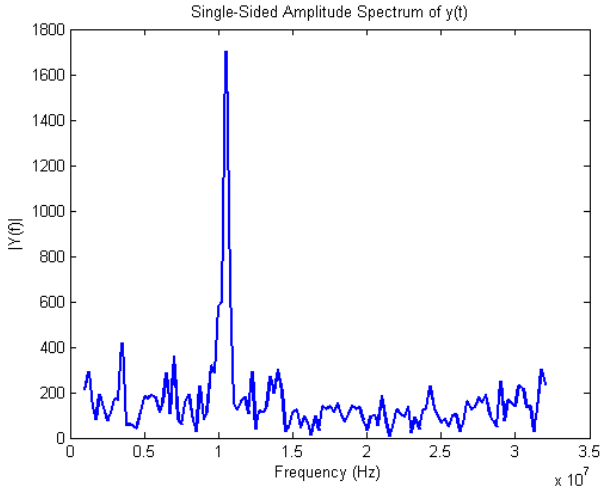


Figure 11: FFT of samples from the AD converter, with a peak at 10MHz.

This test helped in verification of the ADC and also confirming that the connections between the boards were done correctly.

6.1 Simulink model

Simulink is a tool from MathWorks the developer of MATLAB. It is a graphical environment where you can do simulation of different systems with the help of blocks. It is especially developed for communications, controls, signal processing, video processing, and image processing.

As project plans for this project was to use a Xilinx FFT core for the FFT calculations, it was fortunate that Xilinx had provided a block library for Simulink, with their own cores. By adding the Xilinx FFT-core V6 in Simulink, tests could be performed too learn more about the core. The block has two inputs, one for real numbers and one for imaginary. In the tests the imaginary input was set to zero and a sine wave was applied to the real input. An input called *start* needs to be high for the block to process the data at the input. The inputs for choosing whether the FFT core should use forward or backward transformation (*fwd_inv*) is set to forward.
There are different types of implementation of the algorithm. In this project the Radix-4 is used, which divides the calculations into a number of stages depending of FFT length. The scaling in each

stage is set by a vector connected to the *scale_sch* port. In the model this is set to 0110101011 or 427 in decimal.
Further information about the Xilinx FFT core can be found in the Xilinx LogiCORE product specification sheet. ("Xilinx LogiCORE Fast Fourier Transformer v6.0 DS260 September 2008").

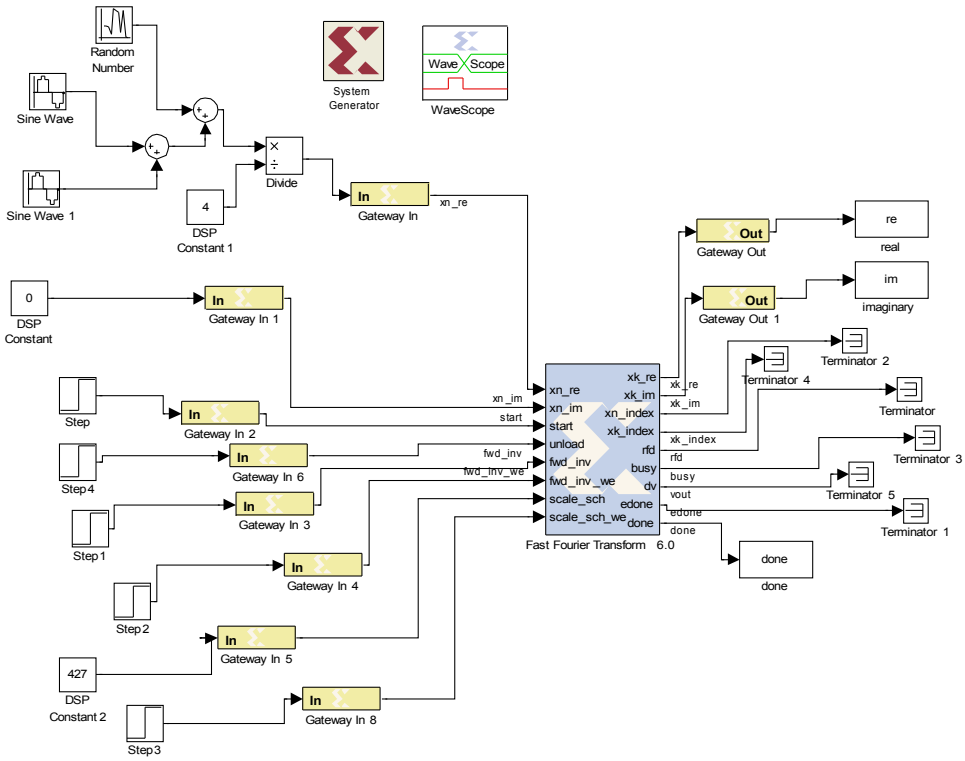


Figure 12: Simulink model with FFT-block in the middle.

The Simulink model created, including the FFT block, is depicted in Figure 12. The model was constructed to get a better view of the expected output and help in deciding the length of the FFT.

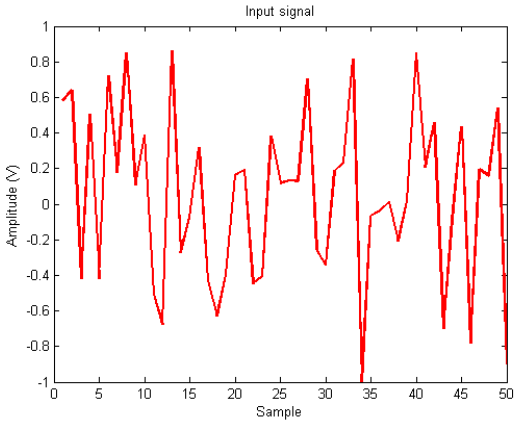


Figure 13: Input sine wave containing 21MHz and 25MHz frequencies

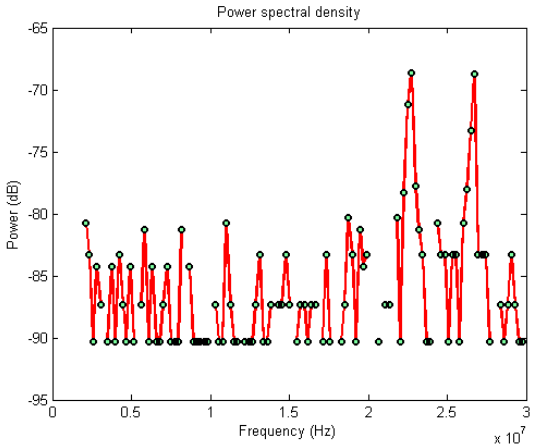


Figure 14: Output, 256 FFT

The output of the FFT block was sampled at 60MHz and plotted for different lengths, utilizing an input signal containing noise and two frequencies at 21MHz and 25MHz (Figure 13).

The noise added was generated using uniform random number, ranging from -1 to 1. Three test cases were studied; FFTs of length 256, 512 and 1024, which can be observed in Figure 14, Figure 15, Figure 16 respectively. As seen, higher the resolution, the more precise the peak detection is and lesser the effect by noise. By studying the output and considering criteria for the bandwidth between frequencies, the decision fell on a FFT of length 256 which has a band width of 234kHz between the detectable frequencies. This was no crucial decision because changing the FFT length is not a complicated process. The decided length should be seen as a initial value which later can be changed after testes if necessary.

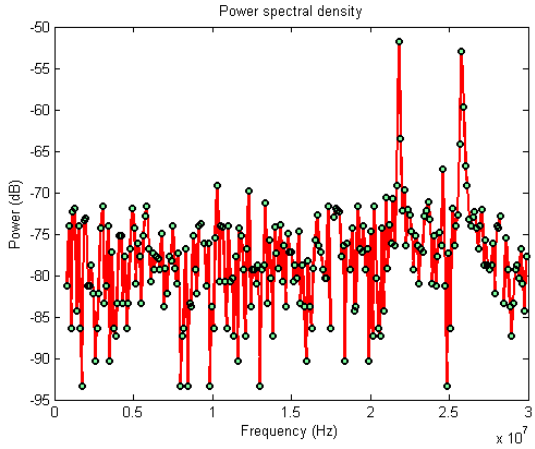


Figure 15: Output, 512 FFT

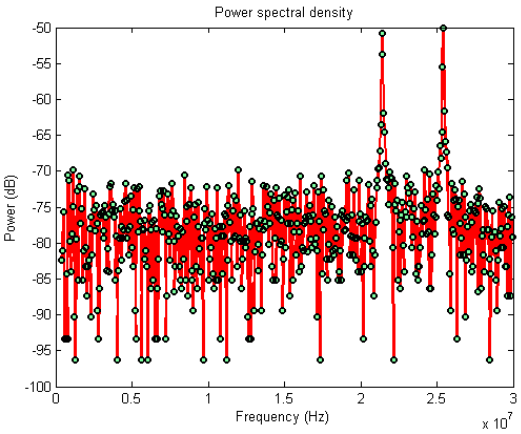


Figure 16: Output, 1024 FFT

7 Set-up

The AD-board is connected to the Virtex2 board with a standard 40 pin flat cable, with every other pin connected to ground for shielding purpose. An oscillator with a frequency of 64 MHz is used as a clock for the AD-board. The Virtex2 board has built in 2.5, 3.3 and 5 V connectors which can be used when connecting external peripherals. In Table 18 the connection between the AD-board and the Virtex2 board is shown, where J5 is the leftmost of the two 40 pin connectors on the board.

Pin on AD-board	**Pin on Virtex2 development board (FPGA pins in brackets)**	**Description**
VDUT	3.3V	Analog power
VLDUT	2.5V	Digital power
VL	2.5V	Logic power
GND	GND	ground
CLOCK	taken from oscillator	input clock
D0	J5 pin 8 (M2)	bit 1
D1	J5 pin 10 (P9)	bit 2
D2	J5 pin 12 (M4)	bit 3
D3	J5 pin 14 (N1)	bit 4
D4	J5 pin 16 (P8)	bit 5
D5	J5 pin 18 (N4)	bit 6
D6	J5 pin 20 (P3)	bit 7
D7	J5 pin 22 (R8)	bit 8
D8	J5 pin 24 (P5)	bit 9
D9	J5 pin 26 (R2)	bit 10
D10	J5 pin 28 (R6)	bit 11
D11	J5 pin 30 (R4)	bit 12
DOR	J5 pin 34 (T5)	data out of range
CLK	J5 pin 38 (U2)	data available clock

Table 18: Connections between AD-board and Virtex2 development board

In Table 19, the pin configuration of the board can be viewed, where J6 is the rightmost 40 pin connector. Sw3 is the leftmost switch of the blue dip-switches and PB ENTER is the center of the five pushbuttons.

Five outputs, active low, are used to set the sending frequencies. Connected to these port there will later be a signal generator, generating the appropriate frequencies.

A LED on the board is indicating if the input is out of range, making it easier to debug for possible errors concerning input amplitude.

The *fft start* input is connected to a switch on the board, this switch can then be used to turn on and off the output of the *FFT-block* and this switch should always be in the on position during standard operation.
The reset button is used to clear all variables and put the state machine back into state one. During normal operation reset is done automatically.

To be able to use the graphical interface, a RS232 serial cable has to be connected between the Virtex2 development board and a computer. The graphical interface can then be viewed with a terminal emulator software such as *Tera Term [17]*.

Pin on Virtex2 development board (FPGA pin in brackets)	**Description**	**Direction**
J6 pin 5 (T8)	beacon	out
J6 pin 7 (U5)	frequency 1	out
J6 pin 9 (W2)	frequency 2	out
J6 pin 11 (U9)	frequency 3	out
J6 pin 13 (V4)	frequency 4	out
(AJ15)	clock	in
LED 2 (AA6)	dor (data out of range)	out
SW 3 (AF9)	FFT start	in
PB ENTER (AG5)	reset	In
RS232	serial communication port	in/out

Table 19: Connection pins used on the Virtex2 development board

8 Results

Test of the implementation was performed during a trip to Halmstad University (Högskolan Halmstad). Parts of the design was tested against a demo-tag done by the Björn Nilsson and Emil Nilsson. The tests showed that detection of signals sent by the tags were possible. During the tests, it was discovered that the oscillator being used for the AD-converter was interfering with the tag at a close distance (1-2m). That problem was later solved by connecting 100nF coupling-capacitor between V_{dd} and ground of the oscillator. As help when choosing the threshold value for detection of signals with different amplitude a test was performed by feeding the AD-converter with a signal of 10MHz and writing out the calculated power value for the detected frequency. As input, a high precision signal generator was used. The measured values along with a trend line can be viewed in Figure 17. The FFT function is linear and power calculation is a second order operation this leads to the expected behavioral of a second order trend line.

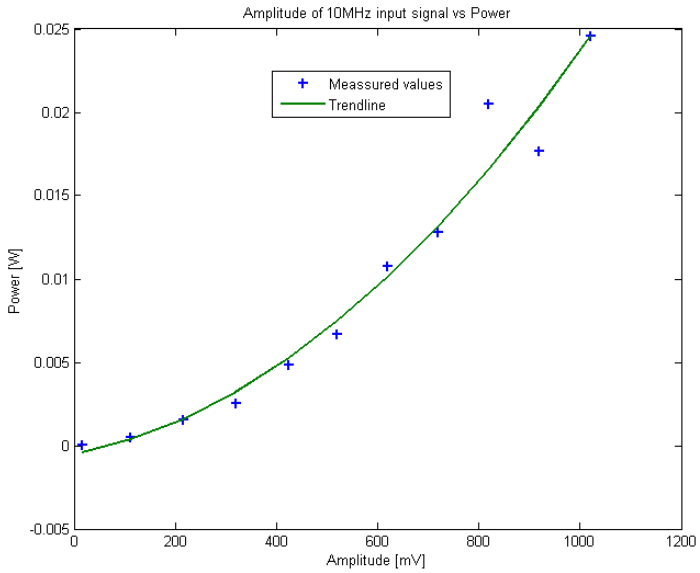


Figure 17: Input amplitude versus Power

Later on further tests were performed at Chalmers using the prototype *RFID-tag*. Frequencies sent by the tag could be detected and after some calibration the tag was able to detect the beacon sent by the *RFID-reader*. The final interface can be viewed in Figure 18. In the GUI you have the choice to set the value of the threshold, frequencies, duration (send and receive time) and tag id bit length. The threshold has to be set according to the input signal amplitude. A value of 300 was used as threshold for the project, which corresponded to $300mV_{P-P}$. All five of the frequencies should be entered in kilo Hertz starting with f0. It is also possible to set the send and receive time separately under the duration option, with each unit representing 10µs. An option for changing tag identification number bit length is also available. This change though has to be changed in combination with change in the hardware. The input and output bus width of MicroBlaze processor is 32, which means all inputs have a range of 0 to $2^{27,}$ since 5 bits are used as control signals. In Figure 18 the design can be seen running and three detected tags are showing.

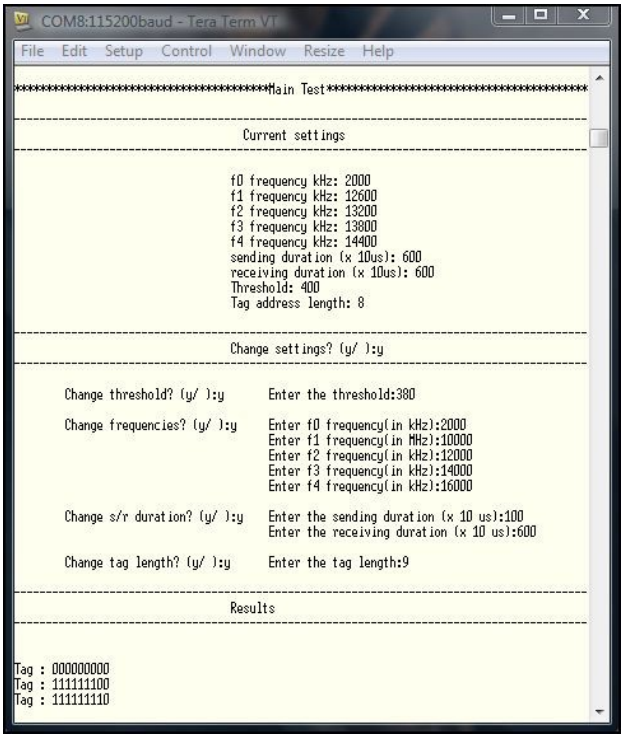


Figure 18: A screenshot of the user interface, three tags are found so far.

An utilization summary for the final design is depicted in Table 21 and Table 20.

Table 20 describes the device utilization of the co-processor, which is the *RFID-reader* design along with the PLB interface. Among the 25% of flip flops and 18% of input LUTs, around 70% was utilized solely for the implementation of the FFT.

Number of Slices:	6140 out of 13696	44,00%
Number of Slice Flip Flops:	7023 out of 27392	25,00%
Number of 4 input LUTs:	5171 out of 27392	18,00%
Number of IOs:	291	
Number of bonded IOBs:	0 out of 556	0,00%
Number of BRAMs:	2 out of 136	1,00%
Number of MULT18X18s:	2 out of 136	1,00%
Number of GCLKs:	1 out of 16	6,00%

Table 20: Device utilization summary- Co-processor

From Table 21, device utilization for the entire project design, it can be seen that 51% of the total available flip flops on the FPGA was used along with 40% of the 4-input LUTs. It should be noted that the *FFT* block along with associated logic utilizes about a third of the total used resources. This was expected from an implementation of 512-point FFT with 512 12-bit registers. The MicroBlaze along with its various device wrappers also used about a third of resources.

Number of Slices:	11176 out of 13696	81,00%
Number of Slice Flip Flops:	14127 out of 27392	51,00%
Number of 4 input LUTs:	10978 out of 27392	40,00%
Number of IOs:	139	
Number of bonded IOBs: IOB Flip Flops:	53 out of 556 84	9,00%
Number of BRAMs:	22 out of 136	16,00%
Number of MULT18X18s:	14 out of 136	10,00%
Number of GCLKs:	8 out of 16	50,00%
Number of DCMs:	2 out of 8	25,00%

Table 21: Project Device utilization summary

9 Improvements

To better the performance and user friendliness of the thesis project, a number of improvements can be considered, as recommended below.

FFT

The major improvement to be considered is the resolution of the FFT results. With the design having a 512-point FFT at a sampling frequency of 64 MHz, the resolution between FFT output data points is 125 kHz. Even though this is sufficient for the current requirements, to take complete advantage of the design, it is necessary to improve the resolution. This can be achieved by down converting the A/D converter output, and then decimating it with an appropriate value. For a design concentrating on the frequency range 20-30 MHz, with the same setup as described above, it is possible to achieve a resolution higher than 50 kHz, as per preliminary MATLAB simulation [18].

It is also possible to further improve the resolution by using multiple FFTs in the same design. In this method, each frequency associated with the RFID tag (f0, f1, f2, f3, f4) will have a dedicated FFT block with down-converter and decimator. As a result the frequency range for each FFT can be narrowed down to attain a much higher resolution [19].

Tag emulator

As a future improvement, it is possible to design a tag emulator as an independent block. The block can also me made less dependent on the MicroBlaze, by moving the tag address generation from C application to the FPGA design. An exhaustive tag address generation and verification can also be made possible.

Debugging

Due to the lack of availability of required tools and instruments, debugging was a complex aspect of the thesis project. The debugging of hardware and software design relied solely on the MicroBlaze debug module. This means that detecting a bug involves lot of time and changes in design. It would be more efficient to use a better environment for debugging. Even though tools like ChipScope Pro, which helps in debugging, were available in market, it required an expensive upgrade of hardware.

The debugging of the analog aspect of the design can also be improved in presence of more instruments. It was carried out mostly with a signal generator and an oscilloscope. This limited the number of frequencies to be produced at a given moment to one. A device such as Programmable Frequency Sweep and Output Burst Waveform Generator would have helped to generate multiple frequencies and simulate a tag functionality to some extent.

Software Application

The C/C++ software application running on the MicroBlaze was used not only as an interface, but also for testing, verification and debugging. But this application is limited due to its operational speed. For the current settings and data rate of the FPGA, the C application performs without a fault. When the data rate was increased (as a result of lesser number of tag address bits and shorter interval between sending/receiving stages of algorithm) in such a way as to detect over 1000 tags per second, operations like decimal to binary conventions was impossible to include in the C application. Though an absolute alternative solution cannot be given to this problem at this point, a faster processor can be considered.

Embedded test module

Throughout the thesis stages, the project design included test modules and tag emulators. But the final design does not involve anything to check the correctness of the design, if any changes were to be made. To address this problem a test module can be embedded in the system, which has a similar functionality of the tag emulator. This module will generate frequency data samples based on the predefined tag addresses. The module then cross checks the results of the *RFID-reader* to the expected results, and notifies the user if an error was detected.

Threshold advisor

The detection of a proper threshold to detect for the frequencies was a very important aspect. It is important to have an appropriate value for threshold, since the whole tag detection has a great impact on this. During the test stages, due to the difference in setup and equipments, the threshold differed. A separate project was created to detect the power spectrum values so that a decision on the threshold could be made. This feature can be further improved and incorporated into the *RFID-reader* design. This can be made available through the user interface, where the user selects the option. The design replies by asking the user to feed an input frequency of specific value. Then the highest power spectrum value for that frequency for a limited interval will be extracted, and the user will be given an advice, to set the threshold, based on this.

Analog filter

An analog filter before the AD-converter is needed to avoid aliasing from unwanted signals above 30MHz. In a closed environment like the lab a filter is not needed but almost everywhere else disturbances can interfere with the signal. Adding a filter between antenna and ADC will make the reader less affected by disturbances [5].

Bibliography

[1] B. Nilsson, L. Bengtsson, PA Wiberg, and B. Svensson, "Protocols for Active RFID - The Energy Consumption Aspect", IEEE Symposium on Industrial Embedded Systems - SIES'2007, Lisbon, Portugal, 4-6 July, 2007. pp. 41-48.

[2] Roberts. Chris M, "Radio frequency identification (RFID)", Computers & Security 2006, 25 (1), pp. 18-26. ISSN 0167-4048.

[3] Xilinx Inc, "Xilinx University Program Virtex-II Pro Development System – Hardware Reference Manual", 8 March 2005, [Online]. Available: http://www.xilinx.com/univ/XUPV2P/Documentation/XUPV2P_User_Guide.pdf [Accessed: 5 Sept. 2009].

[4] Maxim Integrated Products Inc, "MAX1211 Evaluation Kit", June 2005, [Online]. Available: http://datasheets.maxim-ic.com/en/ds/MAX1211EVKIT.pdf [Accessed: 5 Sept. 2009].

[5] C. E. Shannon, "Communication in the presence of noise," Proc. IEEE, vol. 72, no. 9, Sept. 1984, pp. 1192–1201.

[6] Xilinx Inc, "LogiCORE Fast Fourier Transform v7.0", 24 June 2009, [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf [Accessed: 5 Sept. 2009].

[7] Xilinx Inc, "MicroBlaze Soft Processor Core", 2009, [Online]. Available: http://www.xilinx.com/tools/microblaze.htm [Accessed: 5 Sept. 2009].

[8] Jean J. Labrosse, "Use an RTOS on Your NextMicroBlaze-Based Product", Spring 2004, [Online]. Available: http://www.xilinx.com/publications/xcellonline/xcell_48/xc_pdf/xc_micrium48.pdf [Accessed: 5 Sept. 2009].

[9] John Williams, "Microblaze uClinux Project Home Page", 12 May 2006, [Online]. Available: http://www.itee.uq.edu.au/~jwilliams/mblaze-uclinux/ [Accessed: 25 May 2009].

[10] Xilinx Inc, "EDK Concepts and Techniques - A Hands-On Guide to Effective Embedded System Design", 2009, [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/edk_ctt.pdf [Accessed: 5 Sept. 2009].

[11] Xilinx Inc, "Software platforms" , 2009, [Online] . Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/SDK_doc/concepts/sdk_c_platforms.htm [Accessed: 5 Sept. 2009].

[12] Xilinx Inc, "ISE Design Suite Software Manuals and Help", June 24 2009, [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf [Accessed: 5 Sep. 2009].

[13] Mentor Graphics, "ModelSim - Datasheet", 2009, [Online]. Available: http://www.mentor.com/products/fv/modelsim/upload/datasheet.pdf [Accessed: 5 Sept. 2009].

[14] Altera Corporation, "Quartus II Handbook Version 9.0", 2009. [Online]. Available: http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf. [Accessed: 5 Sept. 2009].

[15] Peter J. Ashenden, "The VHDL Cookbook", July, 1990. [Online]. Available: http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf. [Accessed: 5 Sept. 2009].

[16] M. C. Lin, G R. Tsai, Y. C. Tu, T. H. Chang, and C. H. Lin, "FPGA-based Spectrum Analyzer with High Area Efficiency by Goertzel Algorithm", Image and Signal Processing, 2008. CISP '08. Congress on, 27-30 May, 2008, vol.2, pp. 157-159.

[17] Ayera Technologies Inc, "TeraTerm Pro Web 3.1.3 - Enhanced Telnet/SSH2 Client". [Online]. Available: http://www.ayera.com/teraterm/. [Accessed: 5 Sept. 2009].

[18] The MathWorks, Inc., ”Implementing the Filter Chain of a Digital Down-Converter in HDL”. [Online]. Available: http://www.mathworks.com/applications/dsp_comm/demos.html?file=/products/demos/shipping/filterdesign/ddcfilterchaindemo.html#1. [Accessed: 5 Sept. 2009].

[19] Y. Nakagawa, M. Muraguchi, H. Kawamura, K. Ohashi, K. Sakaguchi, and K. Araki, ”Novel Multi-Stage Transmultiplexing Digital Down Converter for Implementation of RFID (ISO18000-3 MODE 2) Reader/Writer”, Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th, 22-25 April, 2007, pp. 2300-2304.