LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Faculty of Technology
Master's Degree Program in Technomathematics and Technical Physics

Evgenia Bogdanova

FIELD-PROGRAMMABLE GATE ARRAY CARD FOR READOUT AND CONTROL OF CRYOELECTRONICS

Examiners: Professor Erkki Lähderanta
Docent Mikko Möttönen

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology
Master's Degree Program in Technomathematics and Technical Physics

Evgenia Bogdanova

Field-programmable gate array card for readout and control of cryoelectronics

Master's Thesis

2012

51 pages, 30 figures, 3 tables, 2 appendices

Examiners: Professor Erkki Lähderanta
Docent Mikko Möttönen

Keywords: FPGA, FFT, DFT, VHDL, ADC, DAC, filter, cryoelectronics, single-electron transistor

The main goal of the present Master's Thesis project was to create a field-programmable gate array (FPGA) based system for the control of single-electron transistors or other cryoelectronic devices. The FPGA and similar technologies are studied in the present work. The fixed and programmable logic are compared with each other. The main features and limitations of the hardware used in the project are investigated. The hardware and software connections of the device to the computer are shown in detail.

The software development techniques for FPGA-based design are described. The steps of design for programmable logic are considered. Furthermore, the results of filters implemented in the software are illustrated.

Acknowledgements

This Master's Thesis work was carried out in Quantum Computing and Devices (QCD) group, Aalto University School of Science, during the Spring semester 2012. First of all, I would like to thank my group leader Adj. Prof. Mikko Möttönen for giving me the opportunity to work in his group and for instructing me at all steps of my work.

I am happy to express my gratitude to Prof. Erkki Lähderanta who provided me the possibility to study at Lappeenranta University of Technology for the good organization of the Double-Degree Program at the Department of Technical Physics in Lappeenranta, and for his useful comments to my Master's Thesis helping me to complete it successfully.

I would like to thank my supervisor in Russia, Igor Klimov, for his advice to take part in the double-degree program between Petrozavodsk State University and Lappeenranta University of Technologies. Furthermore, I am very grateful to Prof. Valeriy Gurtov who coordinates the program in Petrozavodsk for his support in organization questions and active work with students.

I am very grateful to a postdoctoral researcher in the QCD group, Kuan Yen Tan, and to a PhD student, Joonas Govenius, for their useful advice and help in many questions, especially in the beginning of my work.

Special thank goes to Maria Berdova who informed me in time about the vacant position in the QCD group, for her patient answers to my many questions about moving to Helsinki and starting to work in Micronova.

I wish to thank Igor Shavrin and Andrey Serkov for the nice company and interesting talks during the lunch times we shared.

I give special thanks to Carl Jahn for supporting me during the whole time of my studies, for his patience and sense of humor which always improved my mood.

And of course, I am very grateful to my mother for the psychological help during the most intensive work time of my life.

Contents

List of abbreviations

A/D	Analog-to-Digital
ADC	Analog-to-Digital Converter
ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuit
CPLD	Complex Programmable Logic Device
D/A	Digital-to-Analog
DAC	Digital-to-Analog Converter
DFT	Discrete Fourier Transform
DIP	Dual In-line Package
DSP	Digital Signal Processing
EEPROM	Electrically Erasable Programmable Read-Only Memory
FFT	Fast Fourier Transform
FIFO	First In First Out (buffer or memory)
FIR	Finite Impulse Response
FMC	FPGA Mezzanine Card (Standard)
FPGA	Field-Programmable Gate Array
GAL	Generic Array Logic
GPIO	General Purpose Input/Output
HDL	Hardware Description Language
HPC	High Pin Count
IBIS	Input-Output Buffer Information Specification
IDE	Integrated Development Environment
IFFT	Inverse Fast Fourier Transform
IEEE	Institute of Electrical and Electronic Engineers
I/O	Input/Output
IP	Intellectual Property
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
LPC	Low Pin Count
LVDS	Low-Voltage Differential Signaling
MMCX	Micro-Miniature Coaxial
MMI	Monolithic Memories Inc.
PAL	Programmable Array Logic
PLA	Programmable Logical Array

PLD	Programmable Logic Device
PROM	Programmable Read-Only Memory
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
RTL	Register Transfer Level
SEM	Scanning Electron Microscope
SPLD	Simple Programmable Logic Devices
SPS	Samples Per Second
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits

Introduction

The conception of field programmable gate arrays (FPGAs) was developed in Xilinx in the beginning of the 1980s. The first commercial version of an FPGA, which was presented to the world by Ross Freeman, attracted a great interest among scientists and engineers [1]. Today FPGA technology is widely used in many commercial applications such as digital signal processing, communication encoding and filtering, design prototyping and device controllers [2].

Development boards based on FPGAs are interesting for research laboratories because of their re-configurability and performance. In this thesis, the possibility to use an FPGA for controlling single-electron transistors is investigated.

Single-electron transistors are nanometer-scale metallic or semiconducting devices which are operating as transistors [3]. A single-electron transistor consists of two tunnel junctions which are connected to a small island which holds an integer number of electrons. There are three electrodes called drain (positive), source (negative), and gate. A gate electrode is coupled to the metallic island only capacitevely, but electrons can tunnel one by one between the island and the source and drain reservoirs through the tunnel junction [4].

At the current operation point, the current through a single-electron transistor is very sensitive to the changes of the electric fields near the island. Thus it can be utilized as a charge sensor of other devices or even single electron atoms [5]. However, large-amplitude charge transistors can take the sensors out of its optimal working point, which calls for a feedback control for the gate voltage.

A schematic structure of the system is shown in Figure 1.

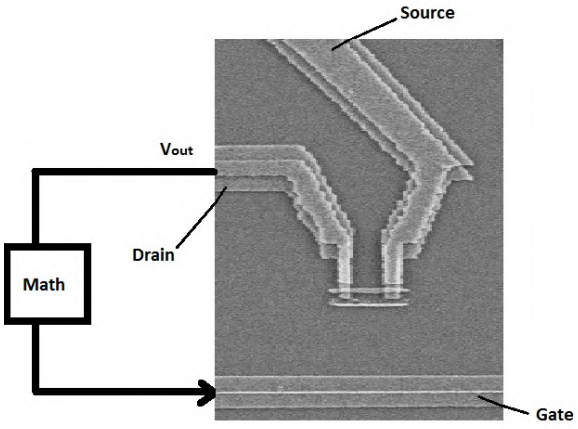

Fig. 1. Single-electron transistor view by scanning electron microscope and a schematic route of the signal in the feedback control system [4].

The present Master's Thesis consists of three chapters. The first chapter introduces the FPGA technology and VHDL programming language for programming of FPGAs. The second chapter describes the steps for implementing a system for the control of single-electron transistors, the hardware, the software, and the programming techniques used for this. In the third Chapter the results are presented and discussed, conclusions are made and the plans for the future are proposed.

Programmable logic devices

In digital electronic systems, there are three basic kinds of devices: memory, microprocessors, and logic devices. Memory devices store information such as the contents of a spreadsheet or database. Microprocessors execute software instructions to perform a wide variety of tasks such as running a word processing program or video game. Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform [6].

Usually logic devices are classified into two categories: fixed logic and programmable logic.

The circuits in a fixed logic device are permanent, they perform one function or set of functions – once manufactured, they cannot be changed [6]. A programmable logic device (PLD) is a general name for a digital integrated circuit capable of being programmed to provide a variety of different logic functions [7]. Examples of PLD are gates, multiplexers, demultiplexers, flip-flops, counters, and registers.

Both fixed and programmable logic devices have their own advantages and disadvantages (see Table 1) which define the possibility of using the device depending on the application. The usage of fixed logic devices is reasonable for mass production systems which require the highest performance level at low cost. Fixed logic devices can also be used in the systems in which changes made by users can be critical.

As PLD technology allows easy modification by the user, programmable logic devices can be used in development of systems which require fast reconfiguration. The PLD technology is becoming more popular because it provides a possibility of fast and cheap development and usage of the same hardware for several purposes. Different companies are developing their own solutions based on PLD.

Table 1. Comparison of Fixed and Programmable Logic.

Fixed Logic	Programmable Logic
Modification in the factory	Modification by user
Long time of development	Decreased time of development
Expensive validation and testing	Validation and testing with inexpensive software tools
Suitable for systems of mass production with high performance level	Suitable for systems required fast reconfiguration

The history of programmable logic starts from the time when random access memory was used for hardware implementation and for arbitrary combinational functions of a given number of inputs. Programmable logical arrays (PLA) were developed by Texas Instruments in the 1970s. In 1978, Monolithic Memories Inc. (MMI) introduced a similar technology – programmable array logic (PAL). Generic array logic (GAL) was invented by Lattice Semiconductors in 1985.

Devices within PLA, PAL, GAL are often grouped into a single category called simple programmable logic devices (SPLD) to distinguish them from complex programmable logic devices (CPLD). In fact, CPLD may contain circuitry equivalent to that of several PAL devices which are linked to each other by programmable interconnections [8].

The simplified structure of CPLD architecture is shown in Figure 2.

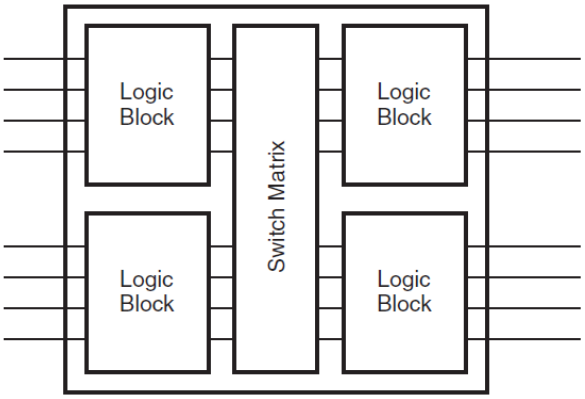

Fig. 2. Simplified complex programmable logic devices architecture, consisting of interconnected logic blocks. The route of the signal is defined by a switch matrix [8].

1.1 Field programmable gate array technology

Field programmable gate array (FPGA) technology is considered separately from the architectures listed above because it applies a different model of operation. Namely, FPGA uses an array of blocks, which can be configured by the user. The internal architecture of an FPGA device has three main parts: the array of logic blocks, the programmable interconnects, and the I/O blocks [8]. The architecture of a typical FPGA is shown in Figure 3.

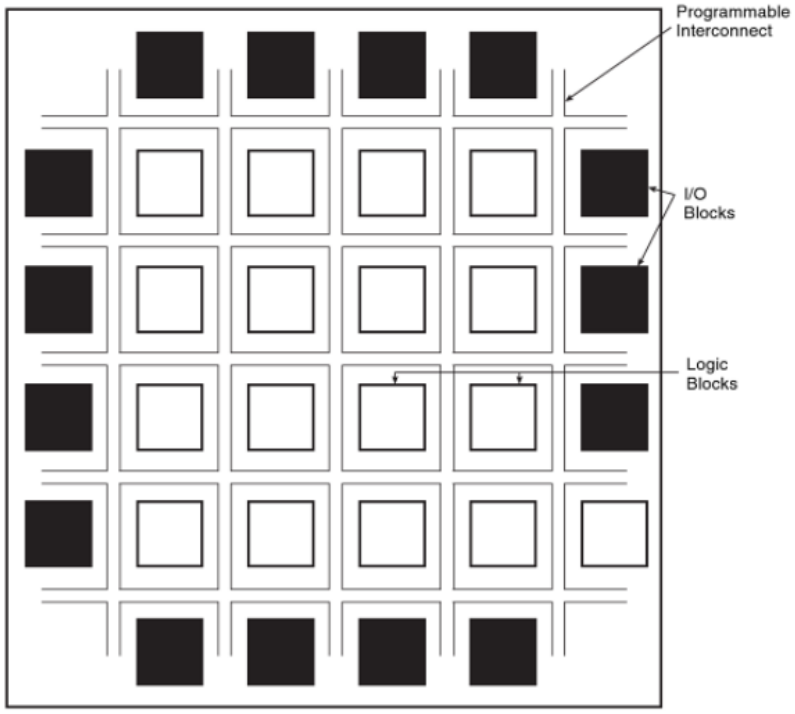

Fig. 3. Field-programmable gate array architecture. White squares indicate logic blocks, black squares indicate input/output blocks, black lines denote programmable interconnections between the blocks [8].

Although FPGAs, as well as CPLDs, are programmable logic, these two technologies are very different from each other. The properties of two technologies are compared in Table 2.

Table 2: Comparison FPGA and CPLD technologies.

FPGA	CPLD
contains many tiny blocks (~100000)	contains relatively few (~100) large blocks
RAM based; the configuration is erased after the power is switched off	EEPROM based; the configuration remains after the power is switched off
can contain large designs	can contain relatively small designs

1.2 VHDL – hardware description language

The acronym VHDL stands for VHSIC Hardware Description Language, and VHSIC is an abbreviation for Very High Speed Integrated Circuits. VHDL is a hardware description language for controlling the behavior of electronic circuits [9]. Hardware description languages (HDLs) are used for simulation, modeling, testing, design, and documentation of projects for hardware [10].

The initiative of creating VHDL belongs to United States Department of Defense (DoD). In the summer of 1981, DoD sponsored a workshop on HDLs at Woods Hole, Massachusetts. This workshop was arranged by the Institute for Defense Analysis to study different hardware description methods, the need for a standard language, and the features for this standard [10].

In 1983, DoD established the requirements for VHDL and the contract for the development of VHDL, its environment, and software was signed with IBM, Texas Instrument, and Intermetrics [10]. The first version of the language was VHDL 87 that was later upgraded to VHDL 93 [9]. Nowadays, the second version of the language is widely used for the development of many projects for hardware, including the project of this thesis.

VHDL is the first hardware description language which was standardized by the Institute of Electrical and Electronics Engineers (IEEE). The motivation to use it for the description of hardware is that VHDL is vendor and technology independent, and therefore portable and reusable. This programming language is generally used for programming CPLD and FPGA devices and is also applicable to Application Specific Integrated Schemes (ASIC) [9].

The language was invented for describing hardware and in fact, it is a concurrent language. This means that all instructions are executed at the same time. This is the main difference of VHDL compared to high-level computer languages such as C or Java [11].

VHDL can be used to model digital circuits. The word “model” in this context means a description of something that presents a certain level of detail. Due to its rich syntax VHDL allows to describe any digital circuit at any level of detail.

There are two primarily purposes of VHDL usage: circuit synthesis and simulation [9].

Synthesis is the process of interpreting the VHDL code and output a definition of the physical implementation of the circuit to be programmed in a device such as a FPGA. Simulation is useful for debugging process, for testing and verification. Any VHDL model can be simulated, but not every model can be successfully synthesized.

Although modeling of digital circuits is often based on the schematic approach, the advantages of VHDL to schematic-based methods are becoming obvious when there is the need to design large systems. The direct connection of elements in the large schemes requires high accuracy and hence it is prone to error.

Implementation of the feedback control system

For the implementation of the feedback control system Virtex-6 FPGA development kit for digital signal processing (DSP) was used.

2.1 Virtex-6 development kit for digital signal processing

Virtex-6 DSP development kit includes:

- Xilinx ML605 Development Board including Virtex-6 FPGA (LX240T model);
- daughter card for analog-to-digital and digital-to-analog conversion (FMC150 model);
- the software ISE® Design Suite: System Edition

2.2 ML605 field programmable array board

The ML605 board enables hardware and software developers to create or evaluate designs targeting the Virtex®-6 XC6VLX240T-1FFG1156 FPGA. The general view of the board is shown in Figure 4. The main features of the board are listed below.

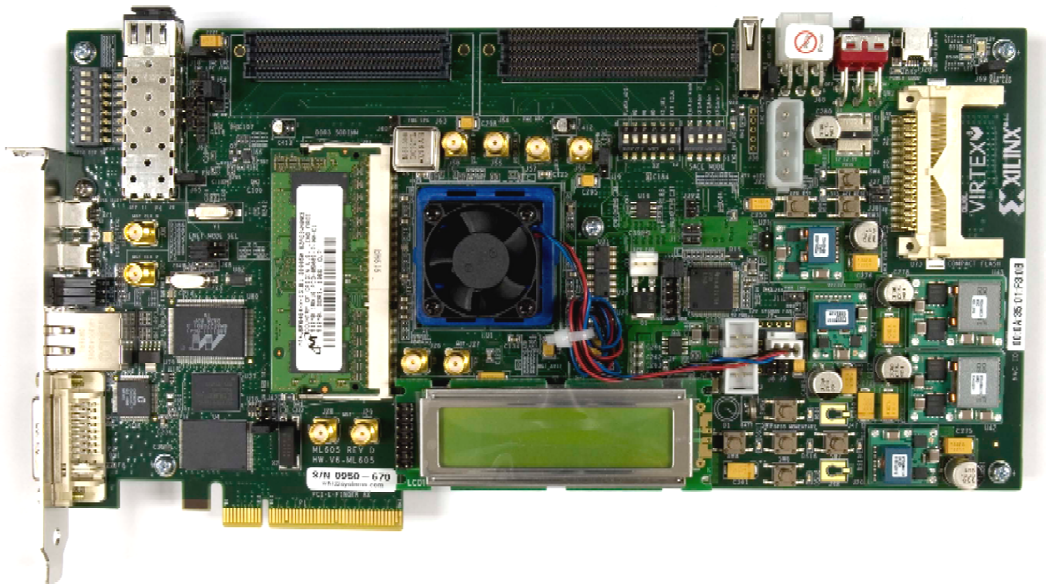

Fig. 4. ML605 board. The description of some elements is given in the text below [12].

FPGA Virtex-6 LX240T is the main component of the board. FPGA is processing the signals in the system.

There are several types of memory devices on the board. **DDR3 Memory SODIMM** (512 MB) is used for storing large amounts of data. **Linear BPI Flash** (32 MB) is used for the storage of the bit stream file of the program for FPGA.

USB JTAG is used for downloading the bit stream file from computer to the FPGA and vice versa.

USB-UART Bridge is used for data transferring from a computer to the FPGA in the present project. **10/100/1000 Tri-Speed Ethernet PHY** can be used for transferring data to or from a computer. **System Advanced Configuration Environment (System ACE)** is an interface for connecting CompactFlash card to the board.

Clock generation – the board contains a built-in differential 200 MHz oscillator. Also there is a possibility to use external clocks.

FMC HPC and **LPC connectors** are used for adding new FMC compatible devices to the board (e.g. ADC/DAC board FMC150).

User I/Os are used for connecting external signals to the board. For example, **status light-emitting diodes** (LEDs) indicate the configuration parameters of the board. **Switches** can be programmed by the user, e.g. General Purpose Input/Output (GPIO) switch. Some switches have special meaning for the board configuration, e.g. the configuration mode switch.

Figure 5 shows a high-level block diagram of the ML605 and its peripherals [12].

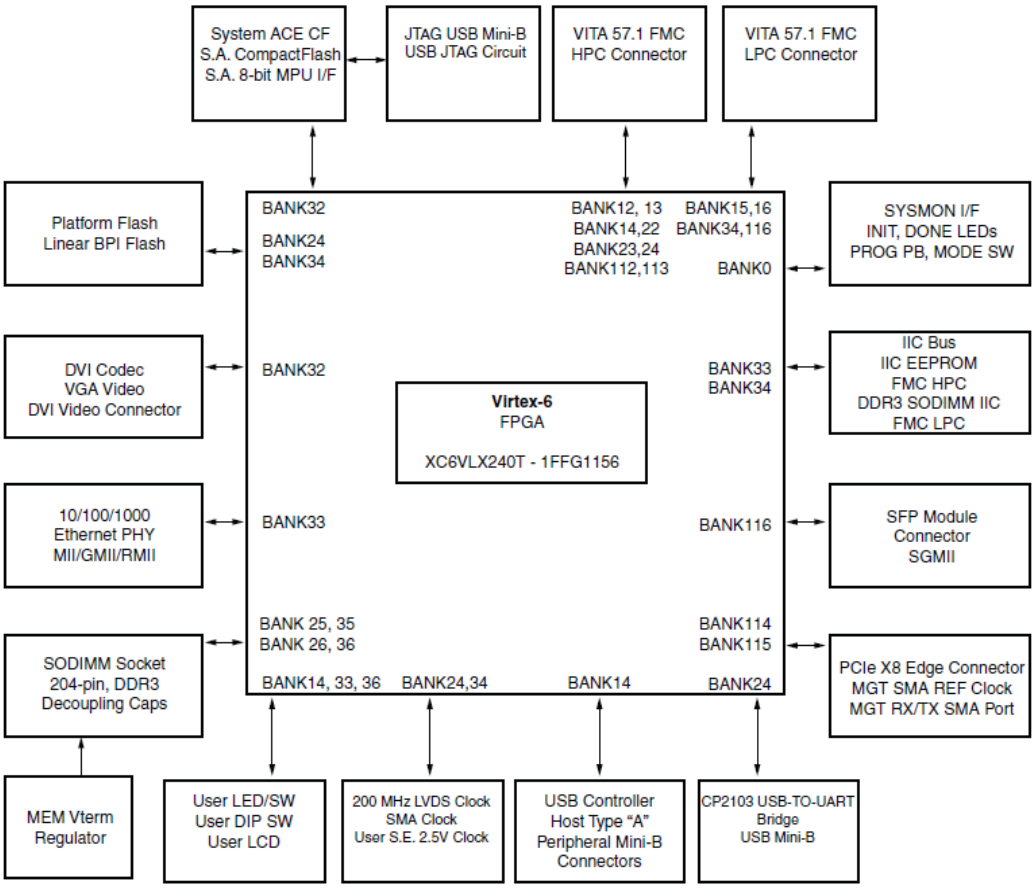

Fig. 5. ML605 high-level block diagram. The FPGA is placed in the center of the diagram and is connected to peripherals. The BANK indication describes the index of the pin bank used for accessing every peripheral [12].

2.3 Analog-to-digital and digital-to-analog conversion card

The FMC150 is a four channel FPGA Mezzanine daughter card (FMC) with embedded analog-to-digital and digital-to-analog converters. The card provides two 14-bit analog-to-digital (A/D) channels and two 16-bit digital-to-analog (D/A) channels which can be clocked by an internal clock source or an externally supplied sample clock. In addition there is one trigger input for customized sampling control [13].

The card contains:

- Analog-to-Digital Converter (Texas Instruments ADS62P49);
- Digital-to-Analog Converter (Texas Instruments DAC3283);
- Clock tree (Texas Instruments CDCE72010);
- Power supply and temperature monitoring block (Texas Instruments AMC7823);

- write-protected electrically erasable programmable read-only memory (EEPROM). The protection can be removed by switching on SW1 of the dual in-line package (DIP) switch on the ML605 board.

The FMC150 card is compliant to FMC standard (ANSI/VITA 57.1). It has a low-pin count (LPC) connector, and supports the JTAG interface. The block diagram of the FMC150 card is shown in Figure 6.

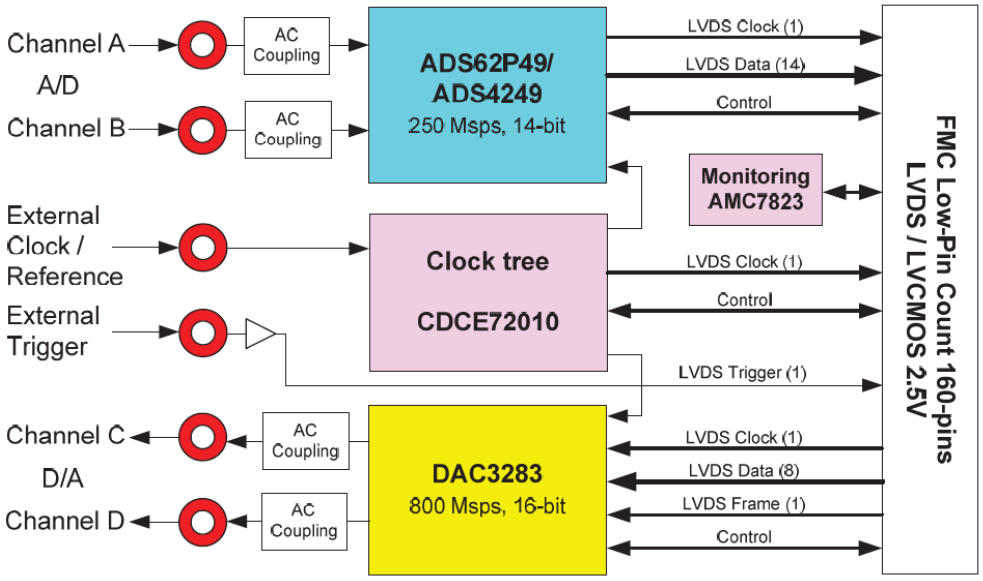

Fig. 6. FMC 150 block diagram. The daughter card contains two input and two output channels. The channels are AC-coupled. There are also input channels for an external clock and a trigger. The blue block indicated ADC, the yellow block indicates DAC. The card is connected to the FPGA board with the FMC LPC connector. The interface of data transfering is 2.5V LVDS [13].

2.3.1 Development software kit

The software ISE® Design Suite is used for control of all aspects of the design flow [14].

The following parts of the software kit are described below in detail: Project Navigator, System Generator, Core Generator, ChipScope Pro, PlanAhead and iMPACT.

Implementing a design for an FPGA-based hardware is an iterative process and includes many steps. Every part of the development requires work with a specialized software tool. **Project Navigator** allows access to all the design entries and design implementation tools [15].

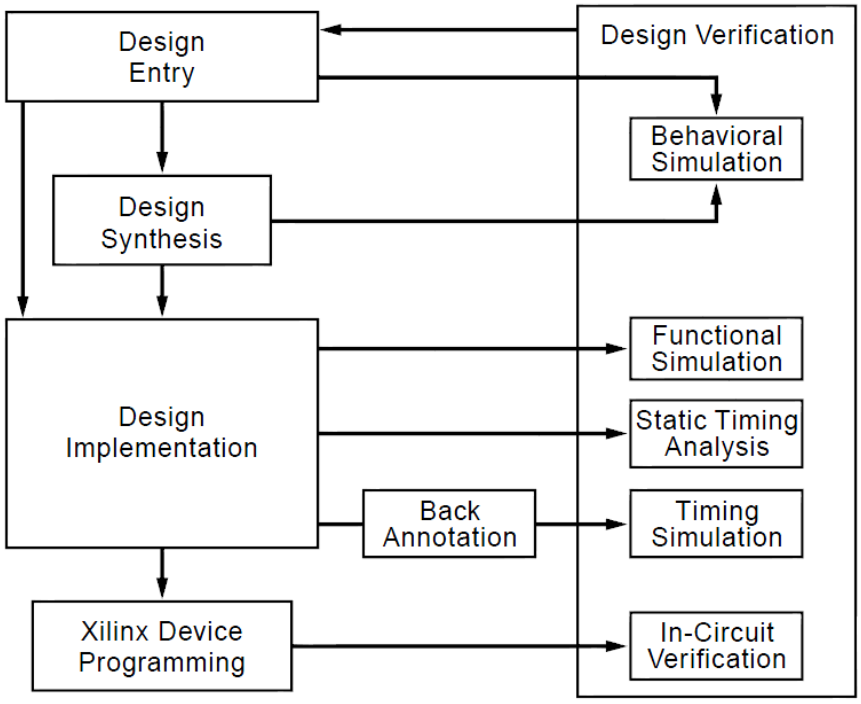

Fig. 7. Block diagram of the process of design implementation. The left part of the diagram represents the development flow, the right part represents the verification flow. The process of the design implementation is iterative: errors or disparities can be found at any step [15].

System Generator is a DSP design tool from Xilinx that enables the use of the Mathworks model-based design environment Simulink for FPGA design [16]. The design of the project for the present thesis is implemented in the Simulink environment with the help of blocks from Xilinx Blockset Library. For the automated compilation of the Simulink model and integrating this to a VHDL project, it is necessary to include the block System Generator to the model.

System Generator allows common usage of MatLab code and HDL code, hardware co-simulation and simulation with the tools of ChipScope. System Generator allows to create both whole models for chosen hardware and modules for importing into a larger system.

Xilinx **CORE Generator**™ System accelerates design time by providing access to highly parameterized Intellectual Property (IP) cores for Xilinx FPGAs [17]. There are plenty of IP cores for different purposes and areas of usage. Widely used libraries consist of building blocks, FPGA architecture features, DSP-functions, video/image processing, wired and wireless telecommunications, and cores for debug and verifications – ChipScope cores.

The **ChipScope Pro** Serial I/O Toolkit provides features and capabilities specific to the exploration and debug of designs that use the high-speed serial transceiver I/O capability of Xilinx FPGAs. Cores can be included into HDL source code or directly into synthesized design netlist.

There is a large amount of cores which can be used for different devices. These cores do not affect the final design in the hardware level, but they allow analyzing the design without loading the design to the FPGA with the help of ChipScope Pro Analyzer tool.

The **PlanAhead** software is a tool that provides a user friendly environment for the entire FPGA design, analysis and implementation process [18]. PlanAhead is used for managing the design data flow from Register Transfer Level (RTL), performing I/O pin planning, managing constraints and performing floorplanning.

2.3.2 Device configuration

The software kit ISE Design Suite is a good tool for all the steps of program development: writing the code, simulation, and verification. The big advantage of this software is the ability to program the FPGA without extra device programmers. The FPGA configuration can be made with the help of the USB JTAG interface or standard CompactFlash cards.

The usage of the CompactFlash card is more convenient when the project is ready and there are no changes in the code which require recompilation. The card should be configured in such a way that after switching power on, the data from the flash card is downloaded to the FPGA automatically. This takes more time compared with JTAG downloading. However, after configuring the CompactFlash card, users do not need to control downloading data to the FPGA. For reconfiguration they need only to change the flash card.

During the development of this project the program data was sent to FPGA by USB cable through the JTAG interface. The disadvantage of this method is that after the device is switched off the information from the FPGA is erased. However, this method is more convenient in the development stage of the project since downloading the program bit stream to the FPGA is much faster in this case.

Before configuring the target device, it is necessary to create all the needed files. This can be done from Project Navigator IDE. Project Navigator creates these system files

from the VHDL (or Verilog) files, IP-cores and MatLab files which are made by the user. The sequence of procedures to obtain all needed files is shown in Figure 8.

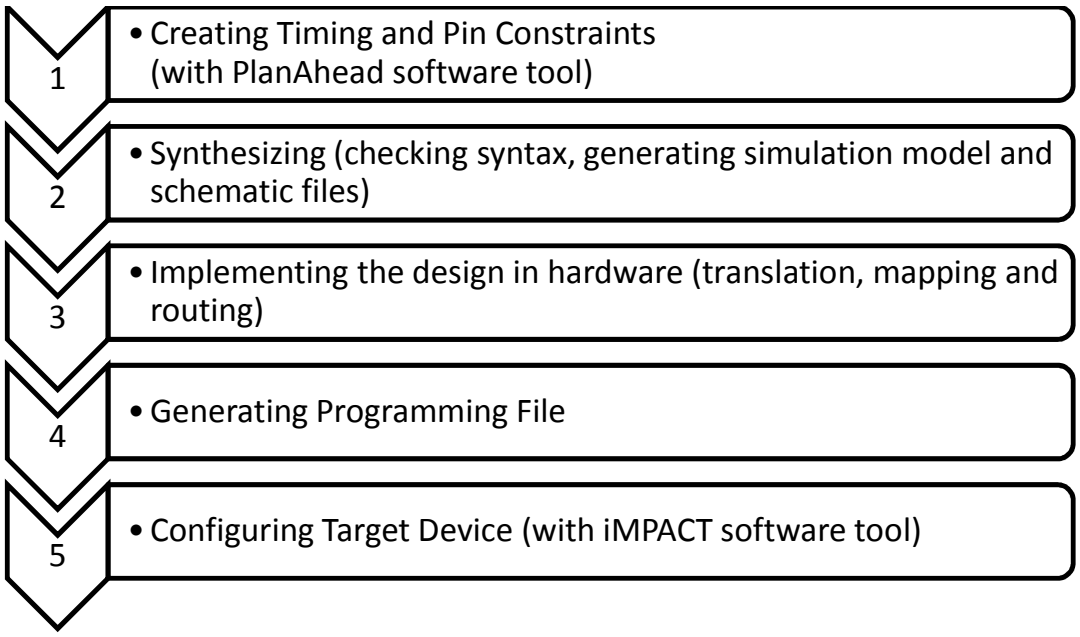

Fig. 8. Flow of the procedures for configuring FPGA.

The user constraints file is a text file with .ucf extension which describes the connections between the I/O ports in the code and pins in the hardware. It also includes the timing constraints. This file can be created with any text editor or in Project Navigator IDE or with the help of PlanAhead software tool.

The advantage of using PlanAhead is its clear layout for different pieces of information. It shows in a table which ports are declared in the code, which pins are already used, and what is the type of each pin. Furthermore, PlanAhead presents the available and occupied pins in a graphical way as shown in Figure 9.

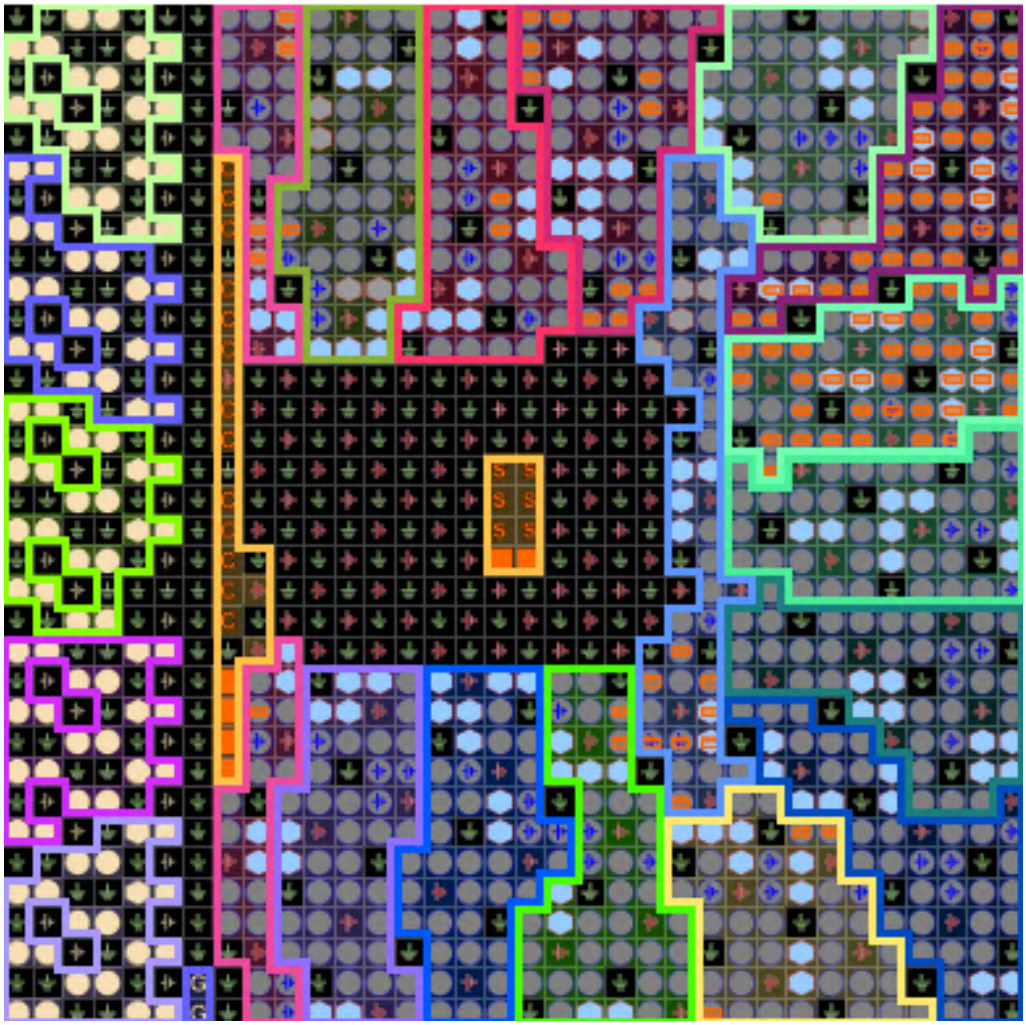

Fig. 9. Pin-view in the PlanAhead program. The colored lines present boundaries between different pin banks of the FPGA board. Grey circles and blue hexagons indicate user input/output pins. Pins marked with 'S' are System Monitor pins. Pins with a 'C' mark are configuration pins. Pins marked with a red rectangle are occupied.

After all pins are assigned, the project can be synthesized. It means that Project Navigator IDE compiles the code, creates RTL and technology schematic files, checks the syntax of all code files, and generates the post-synthesis simulation model. The ability to see the schematic files after the compilation is important because in the case of huge projects, the probability to make mistakes in the code by sending the signal to a wrong pin is very high. The schematic view allows finding these mistakes easily.

Design implementation consists of three steps: translation, mapping, and routing. During the implementation step, different software tools are involved in the calculations of the shortest routes for the signals in the hardware, in analyzing the power and space distribution and in generating of I/O Buffer Information Specification (IBIS) model.

These steps are usually time consuming. In this project, they took from 20 minutes to 2.5 hours. If there were mistakes at any of the above mentioned steps, the IDE would stop the compilation and show error messages.

If the compilation is completed successfully (without errors but maybe with some warnings), the next step, generation of a programming file is allowed to be run. At this step, the software creates a .bit file which is the bit stream of the created project and which can be downloaded to the FPGA.

This bit stream will be stored in the internal FPGA memory blocks. A memory block is to be chosen from the list, because for every device there are specified memory blocks and it is necessary to choose a proper block with the size larger or equal to the size of the file downloaded to the FPGA. For example, for Virtex-6 devices blocks of size up to 16 Mb are available.

Subsequently, the .bit file can be added to this memory. The software tool iMPACT will show in the diagram how much memory is taken by the bit stream file and will also indicate if the chosen memory block is too small for the file. After this step, the memory file with the extension .mcs can be generated.

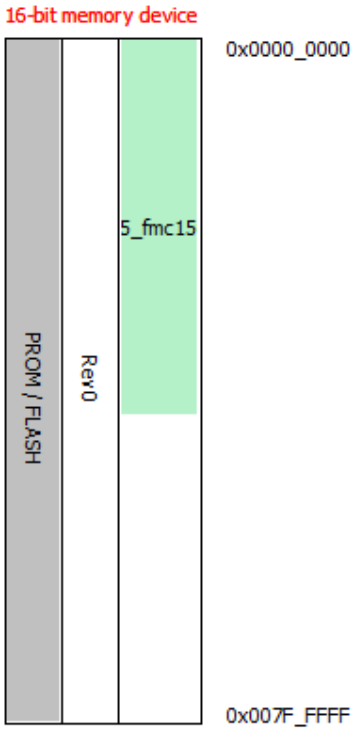

Fig. 10. Memory block presenting in the iMPACT program. The diagram shows the memory width (16 bit), the type of memory (PROM/Flash), the start and end addresses. The green area indicates the amount of the occupied memory.

Before configuring the target device, it should be connected to computer through the JTAG port and be powered on. After adding the configuration files (bit stream and memory) the FPGA can be programmed.

2.4 Project diagram

The project consists of the top module and several sub-modules. The top module describes the system and its behavior at the highest level of abstraction. The sub-modules and connections between them are defined in the top module. Four large groups of sub-modules can be found in the project.

The modules of the ADC/DAC daughter card processing include the description of the ADC, DAC, and clock operation, and the signal interconnections between them.

The mathematical operations modules are made in the Simulink environment using the System Generator tool. The mathematical modules implement the Fast Fourier transform, the finite impulse response filter, and the ensemble average filter.

The modules for debugging and signal control are the following ChipScope cores: Virtual Input/Output (VIO) for representing virtual buttons, LEDs and switches, Integrated Logic Analyzer (ILA) for viewing the signal in the virtual scope, Mixed Mode Clock Manager (MMCM) for generation different clocks based on 200 MHz diffrerntial clock, and Integrated Controller (ICON) for providing an interface between JTAG interface and other cores.

The modules for the PicoBlaze and UART usage are VHDL files describing the PicoBlaze processor, UART receiver and transmitter, and also the PicoBlaze assembler file with the instructions for the processor.

A simplified diagram of the project used in this thesis is shown in Figure 11.

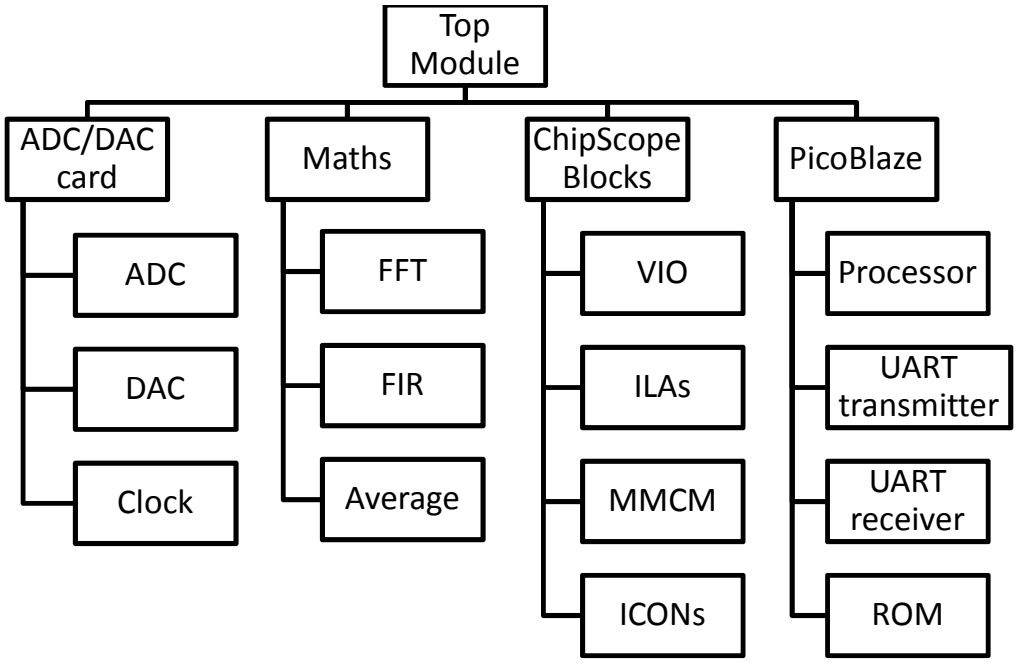

Fig. 11. Simplified diagram of the project. The explanation of the diagram is presented in the text above.

2.5 Fourier transform

The idea about decomposing of a periodic function into a sum of simpler periodic functions has already been discussed in 1822 when Fourier showed how to use this as a powerful tool for understanding nature [19]. At present, the Fourier transform is one

of the most important tools to study the transfer of signals in control and communication systems [20]. The Fourier transform is widely used for calculating the impulse response and the frequency response of a system [20].

The most common applications of the Fourier transform are situations when there is a problem that can be solved only with difficultly, but the transformed problem can be calculated relatively easy. The strategy is to formulate the problem in the transform space and then to apply the inverse transform to the found solution [21]. See Figure 12 for a schematic illustration.

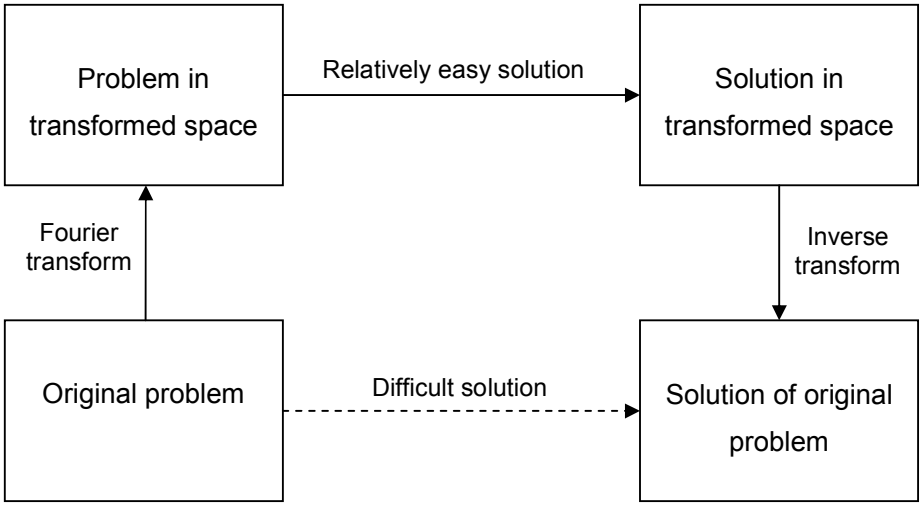

Fig. 12. Possible case for the use of Fourier transforms. When the original problem has a difficult solution, the same problem can have an easier solution in the transformed space.

The Fourier transform is applicable in many engineering tasks such as bandwidth compression, channel separation and combination, decomposition of convolved signals, electrocardiography and electroencephalography signal processing, filter simulation, Fourier spectroscopy, video and image compression, motion estimation, noise filtering, optical signal processing, pattern recognition, image rotation [22].

2.5.1 Definitions

The classical Fourier transform of a time dependent function $f(t)$ that is defined for all values of a real argument t is defined by [23]

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{i\omega t}\, dt. \qquad (1)$$

The Inverse Fourier transform (or the Fourier integral) is given by [24]

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{-i\omega t}\, d\omega. \qquad (2)$$

Here, $F(\omega)$ represents a function depending on frequency ω, while $f(t)$ depends on time t. Hence, it is said that $F(\omega)$ is defined in the frequency domain and that $f(t)$ is defined in the time domain [20].

Practically Fourier transform is used to see a signal not only in the time domain but also in the frequency domain. Usually a signal is described in a graph with time as the abscissa and the magnitude of the wave as the ordinate. After converting the function with the help of the Fourier transform the wave can be described in a second graph with a frequency as the abscissa and its amplitude as the ordinate [25].

For example, if it is needed to televise a picture, the full signal is transformed with the Fourier transform in order to know the color spectrum of the picture [25].

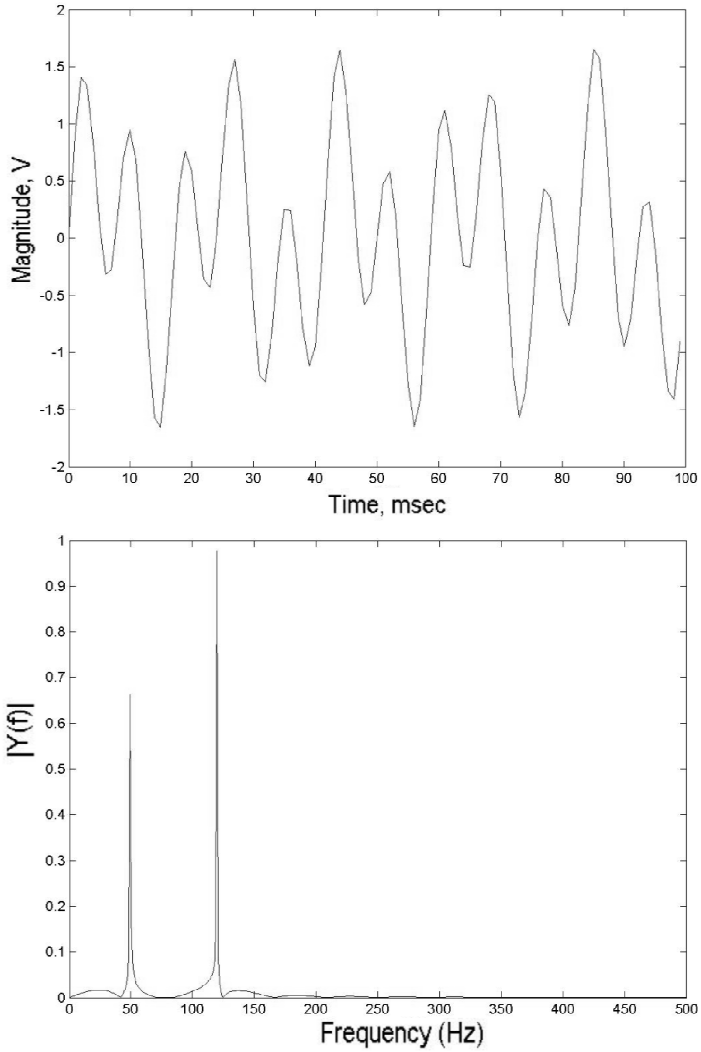

Fig. 13. Fourier transform of the signal of two frequencies. MatLab modeling.

The second definition of Fourier integrals is in terms of frequency ν as shown below [24]. Fourier transform as

$$G(\nu) = \int_{-\infty}^{\infty} g(t) e^{-i2\pi\nu t}\, dt, \tag{3}$$

and the inverse Fourier transform as

$$g(t) = \int_{-\infty}^{\infty} G(v) e^{-i2\pi vt}\, dt. \tag{4}$$

The Fourier transform of a function *f* and its inverse exists when the following three conditions are met [23]:

$$\begin{array}{ll} 1. & \exists \int_{-\infty}^{\infty} |f(t)|\, dt \\ 2. & \text{Any discontinuities in } f \text{ are finite} \\ 3. & f(t) \text{ has bounded variation} \end{array} \tag{5}$$

Often $F(\omega)$ is split up into a real part and an imaginary part. Here,$|F(\omega)|$ is called the amplitude spectrum or the spectral amplitude density of f and $Arg[F(\omega)]$ is called the phase spectrum of f. Furthermore, $|F(\omega)|^2$ is the energy spectrum or spectral energy density of $f(t)$ [20].

Discrete Fourier transform (DFT) is based on the classical Fourier transform, but it requires an input function to be discrete. A discrete function can be obtained, for example, by sampling a continuous function. As digital signals which are used in computers are discrete signals, DFT is an efficient tool for signal processing.

The DFT of a sequence is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-ikn/2\pi}\,, \tag{6}$$

where N is the transform length.

The inverse DFT is given by

$$x(n) = \sum_{k=0}^{N-1} X(k) e^{ikn/2\pi}\,. \tag{7}$$

2.5.2 Fast Fourier transform

Fast Fourier transform (FFT) is an algorithm for the discrete Fourier transform calculation. The history of FFT starts in 1805 when C. F. Gauss developed an

algorithm for the calculation of the discrete Fourier transform even before Fourier published his results in 1822. However, Gauss did not publish his idea. Only after 160 years Cooley and Tukey reinvented the algorithm. Later other scientists developed efficient FFT methods, but the Gauss, and Cooley and Tukey algorithms are the most general, and still widely used [26].

2.5.3 Implementation scheme

Most of the software libraries offer a variety of ready FFT routines, and hence it is not necessary to write the FFT code from scratch. There are built-in FFT commands in MatLab.In addition Xilinx Blockset library for Simulink contains FFT/IFFT block, the block is shown in Figure 14.

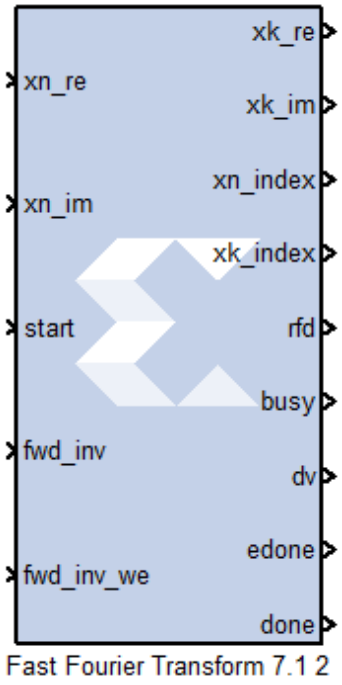

Fig. 14. Xilinx Fast Fourier transform block interface. The description of the ports is given in the text below.

The block contains several input and output signals which are described below [27].

xn_re, **xn_im** are the real and imaginary parts of the input signal.

start marks the beginning of each data frame. The input signal can be asserted as a pulse to start processing. The type of the signal must be Boolean (Bool).

fwd_inv must be assigned to 0 for inverse transform, 1 for forward transform. Default configuration is forward transform. The type of the signal must be Bool. **fwd_inv_we** loads the transform type to the input port forward for the next input data frame. The type of the signal must be Bool.

xk_re, **xk_im** are the real and imaginary components of the output stream. Both signals must have the same data type. **xn_index, xk_index** mark the index of the input and output data.

rfd is a boolean signal which is active high after the **start** signal is asserted till the **xn_index** count reaches N-1 (N is the maximum point size).

busy is a boolean signal which is active high when the block is processing the current input data frame. **dv** is a boolean signal which indicates that the output data is valid when dv equals to 1.

edone is a boolean signal which is active high one sample period before the block is ready to output the frame. **done** is a boolean signal which is active when the block is ready to output the frame.

The FFT block parameters can be viewed and changed in the parameters dialog box which can be invoked by double-clicking the icon of the block.

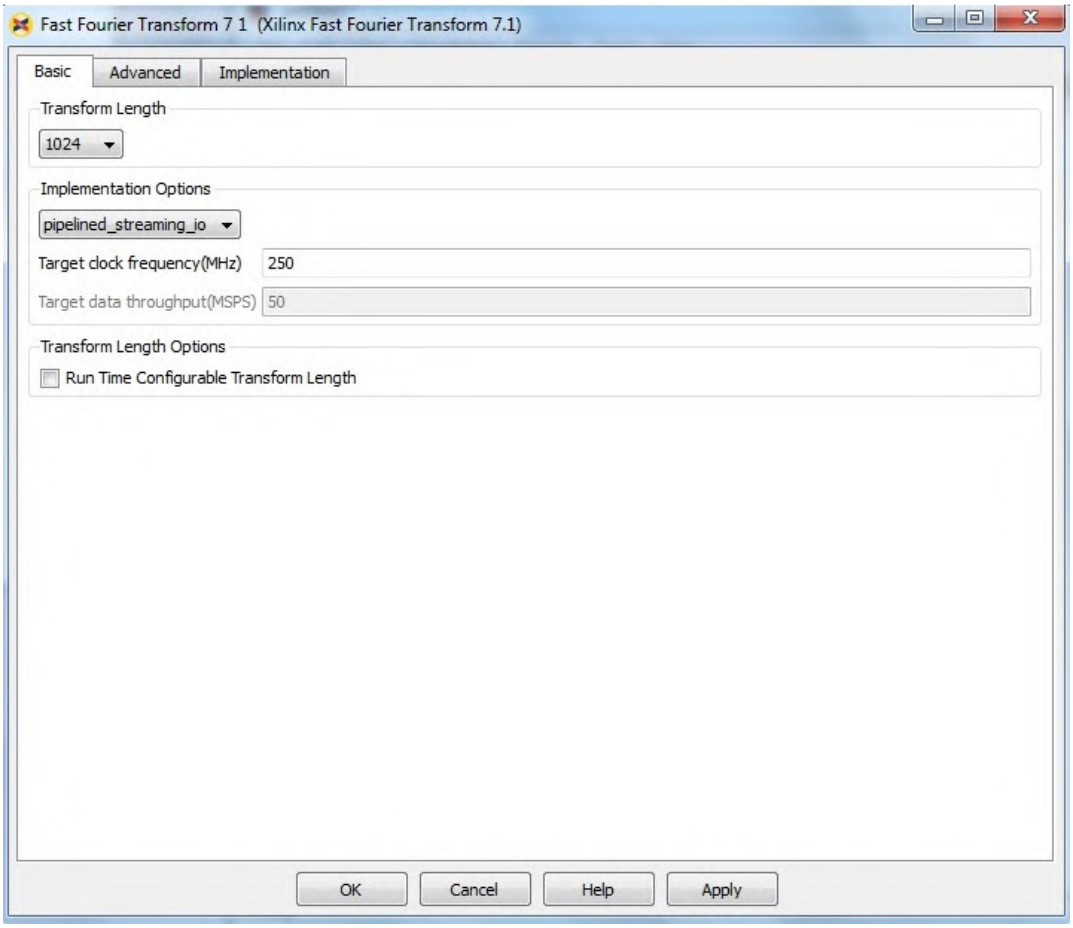

Fig. 15. Xilinx FFT block parameters. The details are given in the text below.

The basic parameters tab which is shown in Figure 15 allows to choose the transform length, type of implementation, target clock frequency (MHz) or target data throughput (MSPS).

The transform length can be equal to $2^3 – 2^{16}$ (8 – 65536) or it can be set through the extra input port **nfft** which provides the point size for the next input frame [27].

As type of implementation, one of the following algorithms can be chosen: *pipelined_streaming_io, radix_4_burst_io, radix_2_burst_io or radix_2_lite_burst_io.* These are different algorithms of FFT implementation. *Pipelined_streaming_io* option allows continuous data processing. Therefore this was used in the present project.

The Advanced parameters tab allows to choose the phase factor width (between 8 and 34), scaling options (unscaled, scaled or block floating point data types), rounding modes (truncation or convergent rounding) and output ordering (bit/digit reversed or natural order).

The Implementation tab parameters were left as default. This tab allows to change the memory configuration and to choose optimization options.

The scheme of the Fourier transform module is shown in Figure 16.

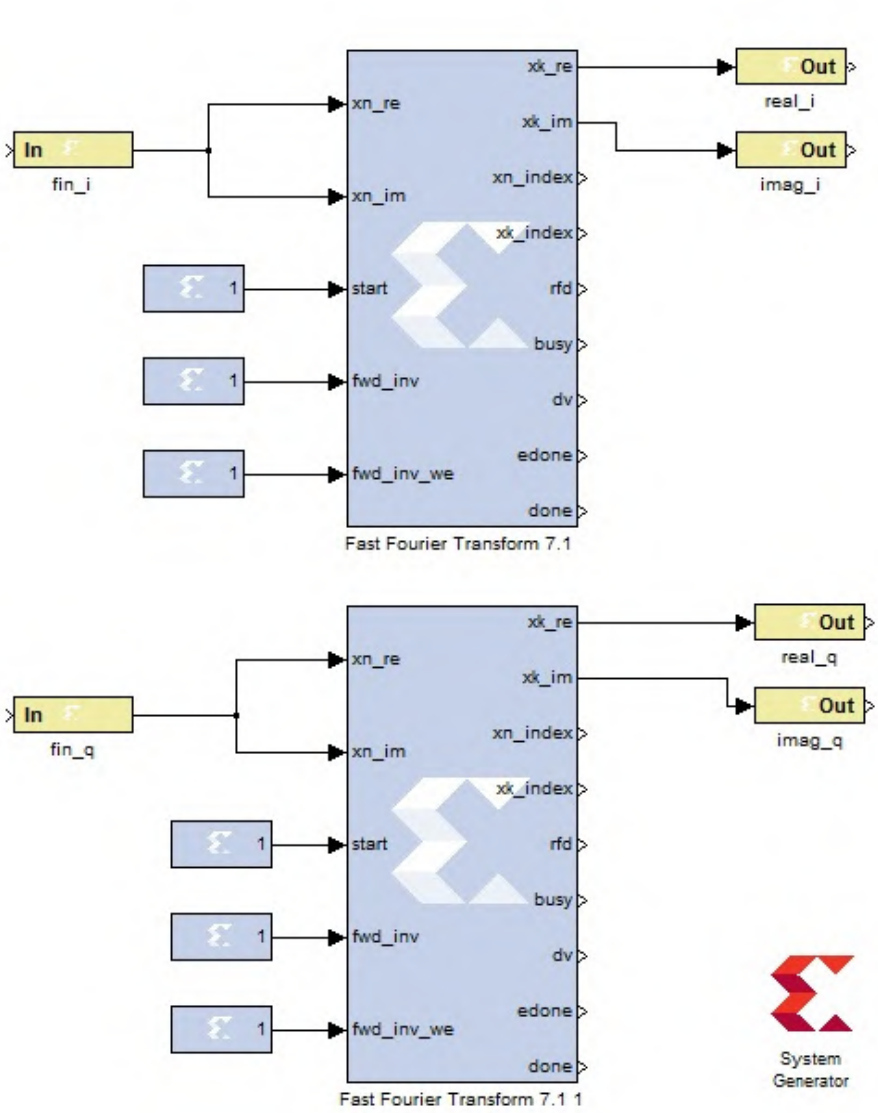

Fig. 16. FFT module of the project. Yellow blocks are inputs and outputs and are needed for the integration of the module to the main project. The System Generator block implements the compilation of the module.

2.6 Filters

The input signal is filtered with three filters: the **high-pass filter** which is built in the ADC/DAC card (AC coupling), **finite impulse response (FIR) filter**, and **ensemble average filter**.

The first filter is realized in hardware. Two last filters are developed in the software and are described below.

2.6.1 Finite impulse response filter

The equation of any FIR filter is given by

$$y[n] = \sum_{k=0}^{T-1} b_k x[n-k]\,, \tag{8}$$

where T is the filter order (number of filter coeffecients), b_k is the k-th coefficient of the filter, n is a sample number.

The design of a FIR filter or the calculation of the filter coefficients can be easily made with MatLab FDATool or with Xilinx FDATool for Simulink.

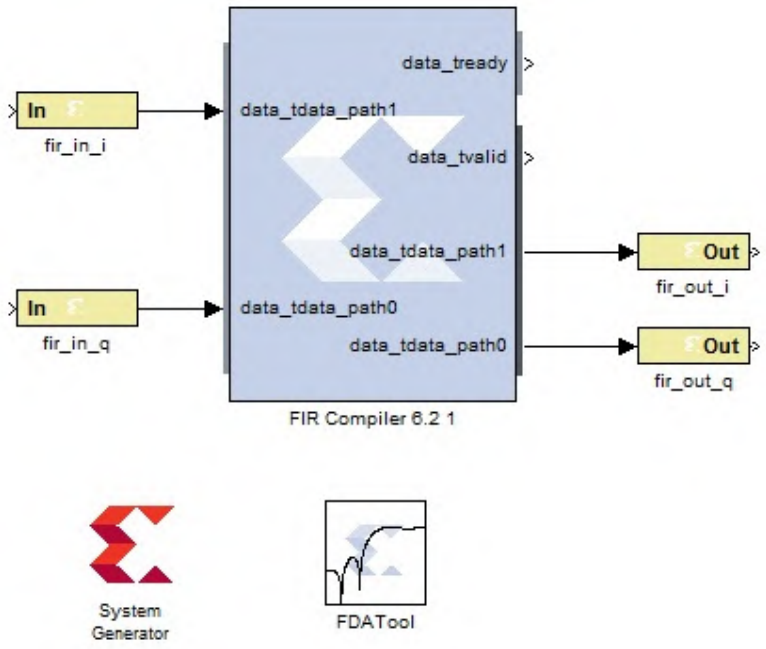

Fig. 17. Finite impulse response filter implementation module. The FDA Tool is a block for choosing the filter type and the calculation of the filter coefficients.

The implementation scheme of a FIR filter module is shown in Figure 17. The FIR Compiler block requires some parameters from the user such as number of the coefficients sets, the coefficient vector, the coefficients type (signed or unsigned), a filter type (single rate, interpolation, decimation), and the rounding mode.

For usage of Xilinx FDATool it is necessary to use the function **xlfda_numerator('FDATool')** in the FIR compiler parameters in the line of coefficients vector.

The block has a possibility to create several inputs for the parallel processing of the data. The input signal in the present system is divided to two parts, therefore the block has two inputs. Every input has its own output port.

The parameters used in the program are shown in Table 3.

Table 3. Parameters of the FIR Compiler block.

Name of parameter	**Value**
Coefficient vector	xlfda_numerator('FDATool')
Number of coefficients sets	1
Filter type	Single rate
Hardware oversampling format	Maximum possible
Filter architecture	Systolic_Multiply_Accumulate
Quantization	Quantize only
Coefficient width	16
Coefficient fractional bits	0
Coefficient structure	Inferred
Number of paths	2
Output rounding mode	Full precision

2.6.2 Ensemble average filter

Another filter used in the system is an ensemble average filter. This filter was implemented in the Simulink environment using Xilinx blocks. The model of the filter is shown in Appendix B.

The idea of the filter is simple. Every 1024 samples of the input signal are stored in a memory and are summed to another 1024 samples. After every cycle (up to 512) of summing every sample of the signal is divided by the cycle index and is output.

2.7 Signal readout

After the calculations the signal can be read out with the help of an oscilloscope or by sending data to a computer.

2.7.1 PicoBaze™ 8-bit embedded microcontroller

The PicoBlaze™ microcontroller is a compact, capable, and cost-effective 8-bit Reduced Instruction Set Computing (RISC) microcontroller core optimized for the Xilinx FPGA families. It is provided as a free, source-level VHDL file with royalty-free reuse within Xilinx FPGA [28]. The KCPSM6 version of PicoBlaze microcontroller suits for Virtex-6 FPGA devices and requires only 26 logic slices and one block memory which equates to less than 4% of all available ones [29].

One block RAM can store up to 1024 processor instructions, which are automatically loaded during the FPGA configuration. This usually is enough for typical implementations [28].

The principal block diagram of PicoBlaze is shown in Figure 18.

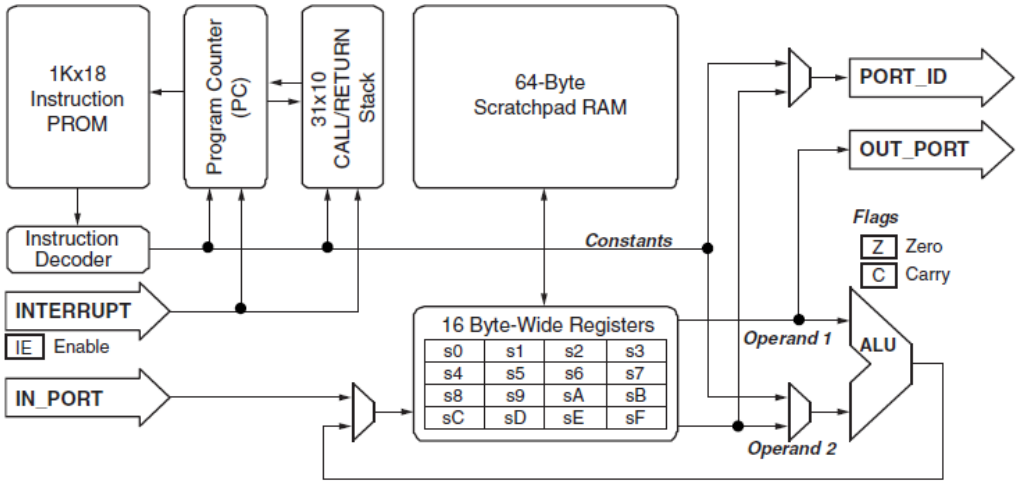

Fig. 18. Picoblaze Embedded Microcontroller block diagram. It consists of PROM, counter, instruction decoder, scratchpad RAM, call/return stack, arithmetic logic unit (ALU) and registers. The registers which can be used are named s0 – sF. External signals of the microcontroller are input and output signals, interrupt and port id signals [28].

The use of PicoBlaze microcontroller is helpful for implementing e.g. state machines or for UART communication [29].

The top-level interface signals to PicoBlaze microcontroller are shown in Figure 19.

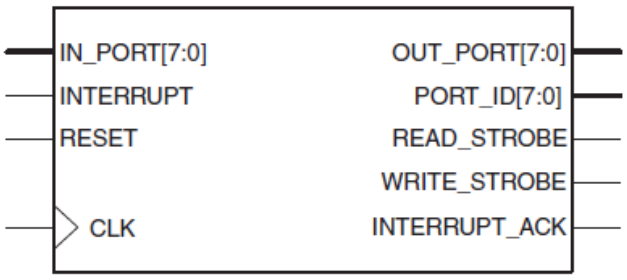

Fig. 19. PicoBlaze interface connections. The port description is in the text below [28].

The signals description [28]:

IN_PORT[7:0] is an input data port which captures data during an INPUT assembler instruction. There can be several IN_PORT[7:0] if it is necessary to process the data of width more than 8 bits.

INTERRUPT input is needed for processing interrupt events. **RESET** input is needed for generation of resent events.

CLK is a clock input. All synchronous elements in the PicoBlaze design should be connected to one clock signal. The frequency may range from DC to the maximum operating frequency**.**

OUT_PORT[7:0] is an output port. Output data appears on this port during an OUTPUT assembler instruction.

PORT_ID is a port which outputs I/O port addresses during INPUT and OUTPUT assembler instructions.

READ_STROBE is asserted active high when input data on the IN_PORT[7:0] was captured to the specified data register during an INPUT instruction. **WRITE_STROBE** is asserted active high when output data on the OUT_PORT[7:0] was captured during an OUTPUT instruction.

INTERRUPT_ACK is asserted High for acknowledging of interrupt event occurring.

For using KCPSM6 PicoBlaze processor this block should be included in the main project and necessary signals should be declared. Since all processor instructions are stored in the memory, a block memory should be added to the project as well. To be compatible with the normal hardware design flow, the program memory is defined by a standard HDL-file (VHDL in the case of the present project) [29].

The memory HDL-file is obtained from the PSM-file with assembler instructions. The KCPSM6 assembler is used for compiling PSM-files. The assembler is provided with the KCPSM6 processor and it should be copied and pasted to the same directory where PSM-file is situated [29].

Each line of PSM-file should follow the basic syntax:

LABEL: INSTRUCTION OPERAND, OPERAND2 ;COMMENT

Labels are case sensitive and can use number and standard characters. Labels can be used at any place of the code to define the target address for a JUMP or CALL instruction [29].

Instructions are not case sensitive. There is wide set of KCPSM6 instructions, the most often used are INPUT, OUTPUT, JUMP, LOAD, ADD/SUB, AND/OR/XOR, CALL/RETURN, and STORE [29]. All instructions except RETURN have at least one operand. If an instruction requires a second operand, it should be separated from the first operand by a comma. Anything following a semicolon is treated as a comment and is ignored by the assembler [29].

2.7.2 Universal asynchronous receiver and transmitter

Universal asynchronous receiver and transmitter (UART) is a circuit which sends parallel data through a serial line [30]. UART is commonly used with RS-232 standard.

A UART includes a transmitter and a receiver. The transmitter is a special shift register that loads data in parallel format and then shifts it out bit by bit. The receiver shifts in data bit by bit and then reassembles it. The serial line is ‘1’ when it is idle. The transmission starts with a start bit (‘0’) and ends with a stop bit (‘1’). The number of data bits is to be up to 8, the number of stop bits can be 1, 1.5 or 2. There is an optional parity bit which can be used for error detection [30]. The time diagram for a byte transmission without parity and with one stop bit is shown in Figure 20.

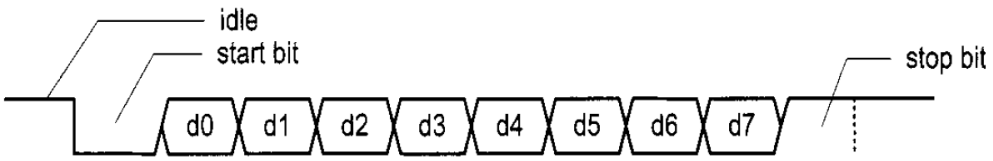

Fig. 20. Transmission of a byte with UART. The Idle signal means system inactivity. Start and stop bits indicate the start and the end of the transmission. Bits d0 – d7 are transmitted bits [30].

The ML605 board contains an USB-to-UART bridge device which allows connection to a host computer with a USB cable. Xilinx UART IP or UART macros should be implemented for use the USB-to-UART bridge device [12].

The macros consist of UART transmitter and UART receiver which include 16-byte FIFO buffer. The macros occupy 10 slices of a Virtex-6 device that is less than 1% of the total amount. The UART6 macros can be used standalone in the design but they are ideally matched with the PicoBlaze processor [31].

The UART macros have the pre-defined configuration of transmitter and receiver: 8-bit data, 1 stop bit, and no parity [31].

For using UART macros in the design components uart_tx and uart_rx should be declared and instantiated in the code. The schematic interfaces of these components are shown in Figures 21 and 22. The predefined configuration is recommended for usage but the baud rate must be defined by user. The baud rate is the number of bits per second at which information is transmitted and received. The input port **en_16_x_baud** is used for defining the desired baud rate. Usually the baud rate generation takes the form of a simple counter [31].

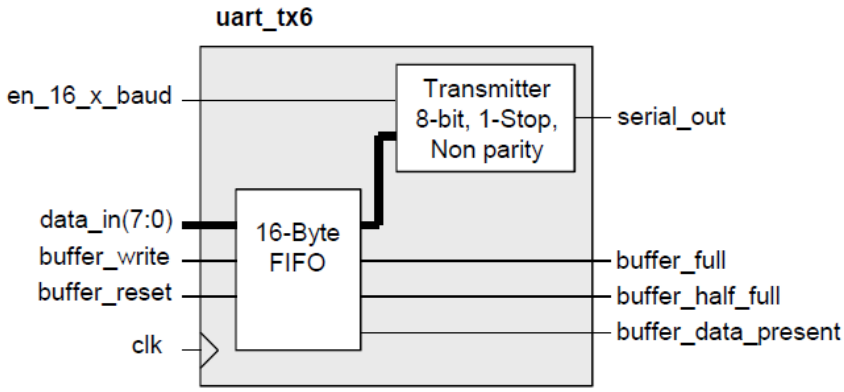

Fig. 21. Interface of UART transmitter. 16-bit FIFO buffer keeps the data up to 16 byte before transmitting. The ports description is in the text below [31].

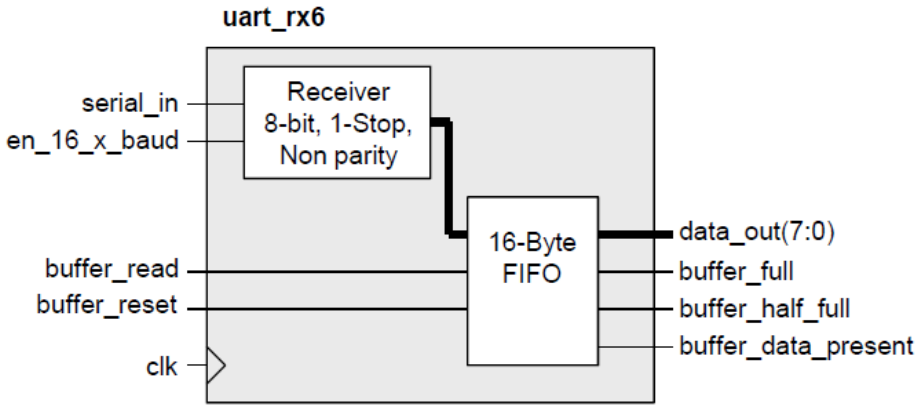

Fig. 22. Interface of UART receiver. Receiver is reading data from the computer and saves it into 16-byte FIFO buffer before output [31].

The data for transmitting should be connected to **data_in(7:0)** port of uart_tx6 component. Received data is to be connected to **data_out(7:0)** port of uart_rx6. If it is needed to transfer data of width bigger than 8 bits there should be more **data_in** and **data_out** ports in the design [31].

Buffer_full, **buffer_half_full**, **buffer_data_present** are supporting ports which indicate information about the embedded FIFO buffer state.

Clk is a port for a clock signal. It is recommended to connect clk ports of uart_rx, uart_tx and PicoBlaze processor block to the same clock signal.

When all the signals are connected in a proper way there should be four external conditions met to get data to the computer with the help of UART:

1. Virtual COM Port driver from Silicon Labs should be installed to a host computer to permit the USB-to-UART bridge to appear as a COM port in a host computer;
2. USB cable should be connected to UART port on ML605 board;
3. any type of terminal should be installed and configured on the computer (e.g. PuTTY, HyperTerminal);
4. FPGA must be configured.

The scheme of connection of UART transmitter, UART receiver and PicoBlaze processor is shown in Figure 23.

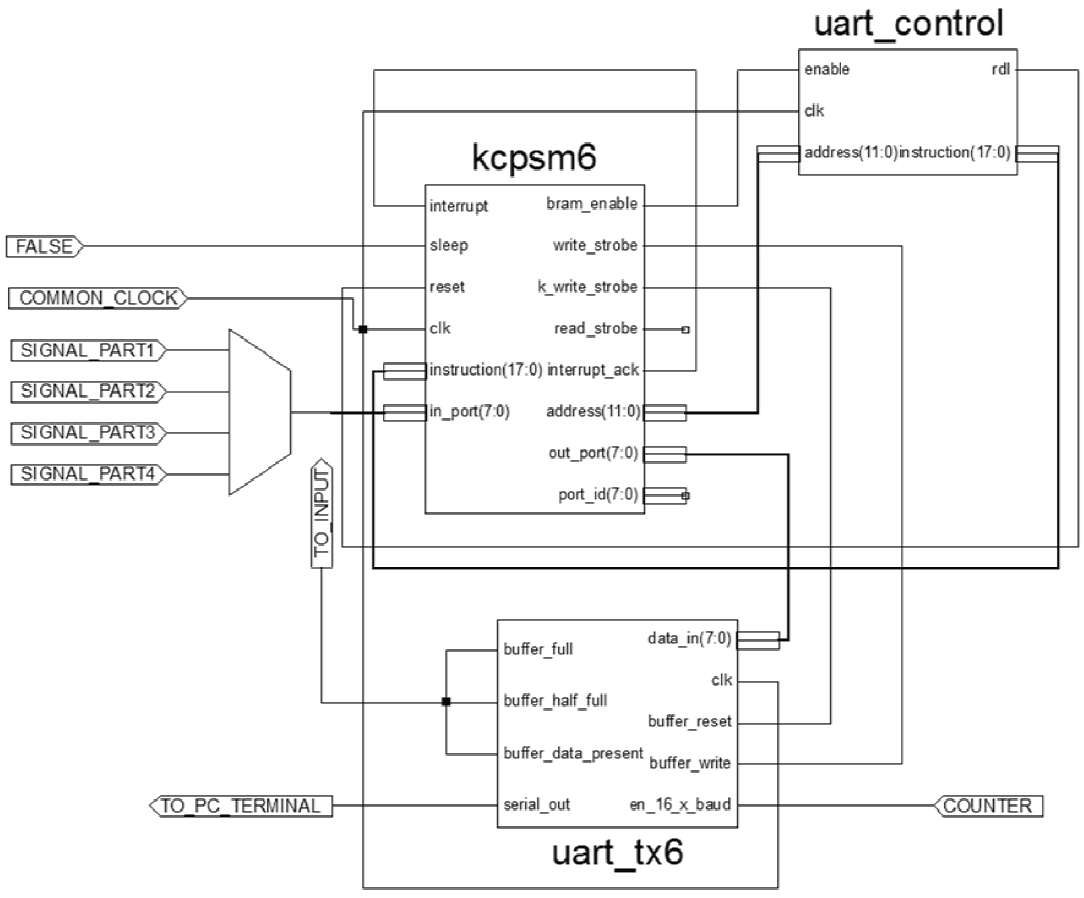

Fig. 23. Schematic view of the PicoBlaze UART control module. The block kcpsm6 represents PicoBlaze processor. The uart_tx and uart_rx blocks are blocks for UART macros usage. The input signal is divided to 4 parts.

Results and conclusions

The results obtained during the testing of system are described in this chapter. The results of the simulations will be shown. The problems and limitations observed during development of the project are discussed.

3.1 Analog-to-Digital/Digital-to-Analog card limitations

During the testing of the hardware, we found that the ADC/DAC card cannot be used at low frequencies. The route of the signal in the test model is shown in Figure 24.

Fig. 24. Signal route in the test program.

The signal was generated by the waveform generator AWG7122B (Tektronix) and was measured with the oscilloscope MSO9404A (Agilent Technologies). As it is shown in Figure 25, the output signal is cut at frequency 200 kHz of the input signal, but the power of the signal is more than half of the input signal. The shape of the signals does not change at frequencies more than 1 MHz.

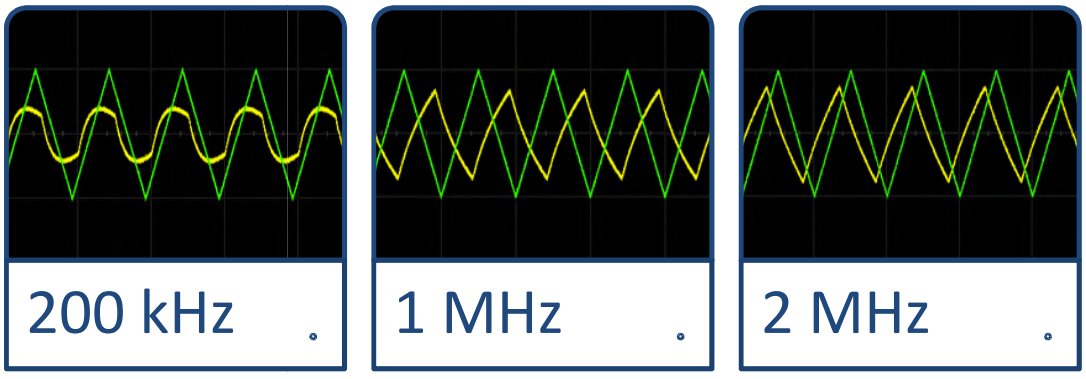

Fig. 25. Comparison of the input and output signals of ADC/DAC card measured at differenct frequencies. The green line represents the input signal. The yellow line indicates the output signal.

Moreover, the frequency response of the system was measured. The frequency range is 20 kHz – 12 MHz. The results of the measurements are shown in Figure 26.

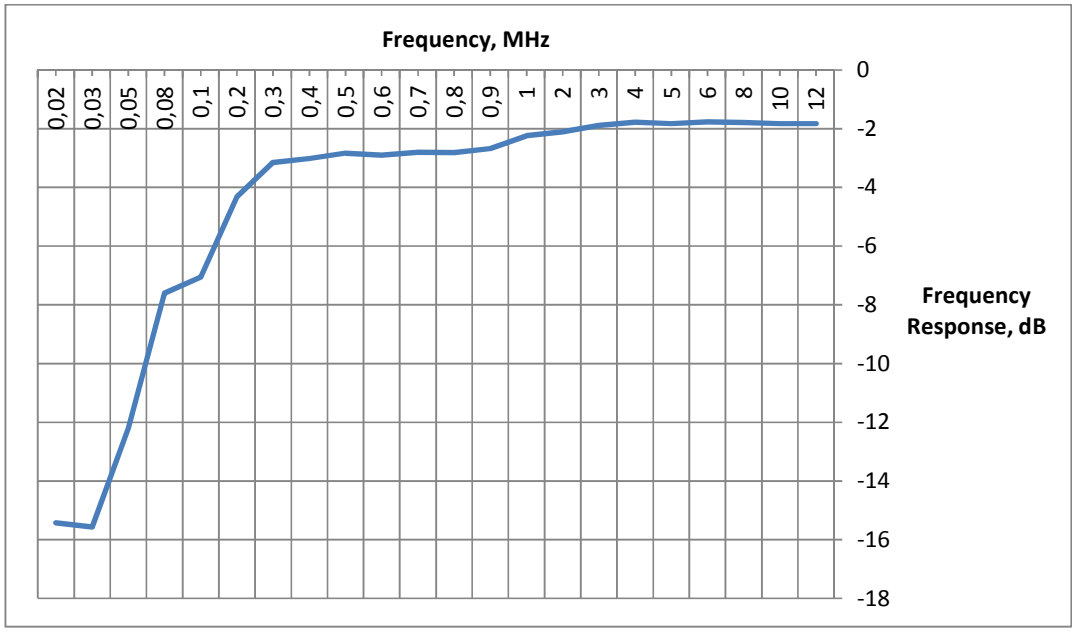

Fig. 26. Frequency response.

Figure 26 shows that at the frequency range from 1 MHz to 12 MHz the frequency response is rather flat. Therefore, we can accept the point -2 dB as a zero. It is known that the signal is accepted as good if its frequency response is more than -3 dB from zero. In this case it converts into -5 dB in the graph which corresponds to the frequency ~200 kHz as in the first experiment.

Moreover, the maximum frequency of the input signal can be ~122 MHz according to Nyquist-Shannon sampling theorem which states that the digitizing frequency should be at least two times higher than the frequency of the digitized signal.

3.2 Fast Fourier transform

The Fast Fourier Transform implementation scheme demonstrates all advantages of the model-based design. Xilinx FFT block is a simple block with input and output ports which should be connected to the proper signals.

The imaginary input port was connected to a zero signal. The results of simulation are shown in Figure 27.

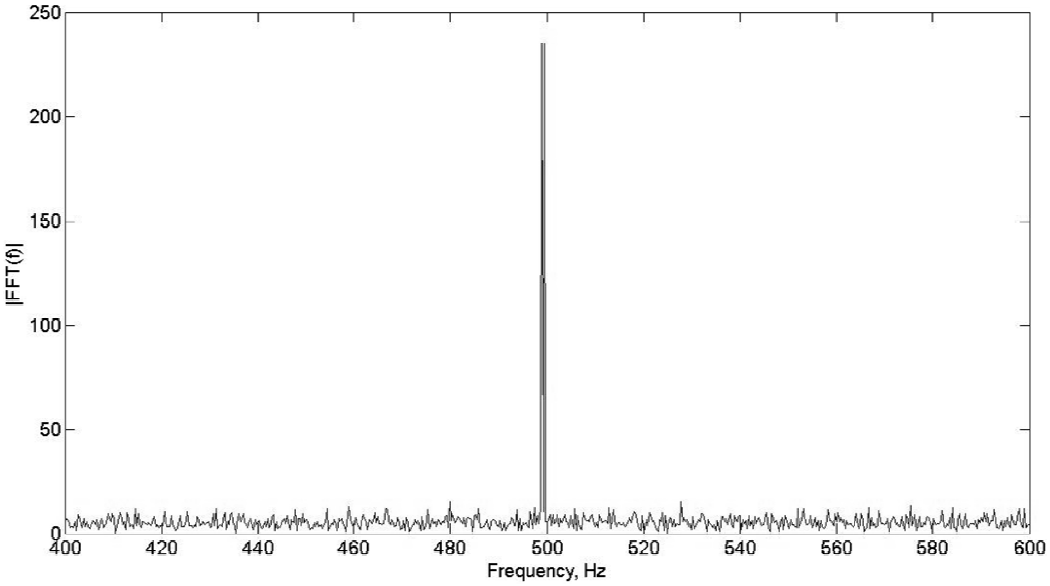

Fig. 27. Fast Fourier Transform simulation results for the sine with the frequency 500 Hz.

3.3 Filter implementation

Two filters were implemented in the system: FIR high-pass filter and ensemble average filter.

The FIR filter is implemented as a standard block of MatLab. The ensemble average filter is built of MatLab Xilinx blocks. The simulation results for both filters are shown in Figure 27. It is observed that the filters give the best filtration when used together. Without the FIR filter, the ensemble averaging filter calculations take too much time. The ensemble average filter increases the signal-to-noise ratio of the filtered signal.

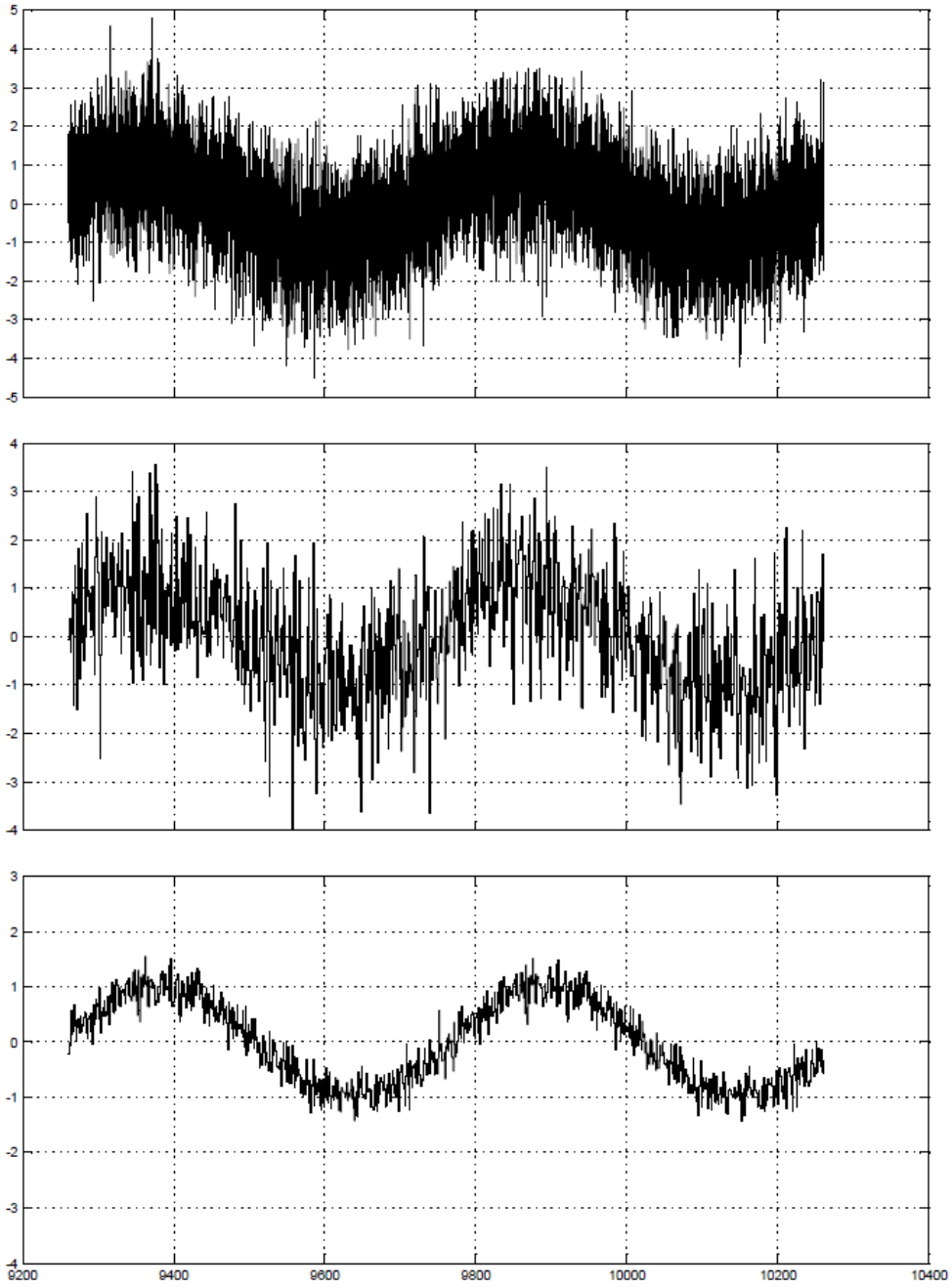

Fig. 28. Signal filtration. The graphs are made in Simulink during simulation. The first graph presents the input simulated signal with a noise. The second graph shows the same signal after the FIR filter. The third graph presents the same signal after the ensemble average filtration.

3.4 USB data transferring

During the implementation of the data transferring to the computer, one limitation of the system was detected. The PicoBlaze processor can only work at the frequency up to 160 MHz while ADC digitized a signal at frequency ~247 MHz.

3.5 Conclusions

The following conclusions were made:

1. The low-frequency limit for the input of ADC/DAC card is 200 kHz because of high-pass filters built in the card. The high-frequency limit is 122 MHz because the sampling rate of ADC is ~245 samples per second.
2. The maximum frequency of transferring data from the FPGA to a computer with the USB UART interface is 160 MHz while the digitizing frequency of ADC can reach 247 MHz. A possible solution is to decrease the digitizing frequency or to round the transferring data or to use another way of transferring (e.g. Ethernet).
3. The ensemble average filter decreases noise very well but it works slowly without pre-filtering with FIR filter.
4. The averaging filter works properly during simulation but not yet in the hardware. Therefore, the plan for future work is to solve this problem. A possible solution is to check the compatibility of the clock frequencies in the system, or to investigate if the amount of memory for the filtering is enough, or to check if the triggering is operating in a proper way.

Bibliography

[1] National Instruments, "FPGA Fundamentals," National Instruments, 25 April 2012. [Online]. Available: http://zone.ni.com/devzone/cda/tut/p/id/6983. [Accessed 27 April 2012].

[2] S. Brown and J. Rose, "Architecture of FPGAs and CPLDs: A Tutorial," Department of Electrical and Computer Engineering, University of Toronto, Toronto.

[3] R. Hanson, L. P. Kowenhoven and e. al., "Spins in few-electron quantum dots," *Reviews of Modern Physics,* vol. 79, no. 4, pp. 1217 - 1265, 2007.

[4] A. Feshchenko, "Improving the Quality of Aluminium Oxide Tunnel Junctions by Thermal Annealing," Lappeenranta University of Technology, Lappeenranta, 2011.

[5] A. Morello, "Single-shot readout of an electron spin in silicon," *Nature,* vol. 467, p. 687–691, 2010.

[6] Xilinx, "What is programmable logic?," [Online]. Available: http://www.xilinx.com/company/about/programmable.html. [Accessed 15 March 2012].

[7] C. H. Roth and L. L. Kinney, Fundamentals of Logic Design, Stanford: Cengage Lerning, 2010.

[8] A. K. Maini, Digital Electronics: Principles, Devices and Applications, Chichester: J. Willey, 2007.

[9] V. A. Pedroni, Circuit Design with VHDL, Massachusetts: Massachusetts Institute of Technology, 2004.

[10] Z. Navabi, VHDL: Analysis And Modelling of Digital Systems, New York: McGraw-Hill, 1998.

[11] B. Mealy and F. Tappero, "Free Range VHDL," 31 March 2012. [Online].

Available: http://www.freerangefactory.org. [Accessed 12 April 2012].

[12] "ML605 Hardware User Guide," 18 July 2011. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf. [Accessed 1 April 2012].

[13] 4DSP, "FMC150 User Manual," 4DSP LLC, Austin, 2010.

[14] Xilinx, "ISE In-Depth Tutorial," 1 March 2011. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/ise_tutorial_ug695.pdf. [Accessed 20 February 2012].

[15] Xilinx, "ISE Design Suite Software Manuals and Help - PDF Collection," 2 December 2009. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf. [Accessed 25 February 2012].

[16] "System Generator for DSP. Getting Started Guide," March 2008. [Online]. Available: http://www.xilinx.com/support/sw_manuals/sysgen_gs.pdf. [Accessed 1 March 2012].

[17] Xilinx, "Xilinx Core Generator System," [Online]. Available: http://www.xilinx.com/tools/coregen.htm. [Accessed 29 January 2012].

[18] Xilinx, "PlanAhead User Guide," 6 July 2011. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/PlanAhead_UserGuide.pdf. [Accessed 15 February 2012].

[19] T. W. Körner, Fourier Analysis, Cambridge: Cambridge University Press, 1988.

[20] R. J. Beerends, H. G. ter Morsche, J. C. van den Berg and E. M. van de Vrie, Fourier and Laplace Transforms, Cambridge: Cambridge University Press, 2003.

[21] G. B. Arfken, H. J. Weber and F. E. Harris, Mathematical Methods for Physicists: a Comprehensive Guide, Waltham: Elsevier, 2012.

[22] K. R. Rao, D. N. Kim and J. J. Hwang, Fast Fourier Transform: Algorithms and

Applications, London: Springer Science + Buisness Media B. V., 2010.

[23] J. K. Beard, The FFT in the 21st Century: Eigenspace Processing, Norwell: Kluwer Academic Publishers, 2004.

[24] A. H. Kaiser, Digital Signal Processing using the Fast Fourier Transform (FFT), Norderstedt: GRIN Verlag, 1997.

[25] K. Morita, Applied Fourier Transform, Tokyo: Ohmsha, 1995.

[26] S. Wörner, "Fast Fourier Transform," Swiss Federal Institute of Technology Zurich, Zurich.

[27] Xilinx, "Fast Fourier Transform 7.1 [HELP]".

[28] Xilinx, "PicoBlaze 8-bit Embedded Microcontroller User Guide," 22 June 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf. [Accessed 18 April 2012].

[29] K. Chapman, "PicoBlaze for Spartan-6, Virtex-6 and 7-Series (KCPSM6)," Xilinx, 2011.

[30] P. P. Chu, FPGA Prototyping by VHDL Examples, Hoboken: John Wiley & Sons, 2008.

[31] K. Chapman, "Ultra-Compact UART Macros for Spartan-6, Virtex-6 and 7-Series," Xlinx, 2011.

[32] J. Claebort, "Superposition of sinusoids," 13 July 1997. [Online]. Available: http://sepwww.stanford.edu/sep/prof/waves/cs/paper_html/node8.html. [Accessed 15 April 2012].

Appendices

Appendix A. The equipment setup

The equipment setup includes an FPGA-based development board, an ADC/DAC card, an oscilloscope, a waveform generator, and a computer. The equipment setup is shown in Figure 29.

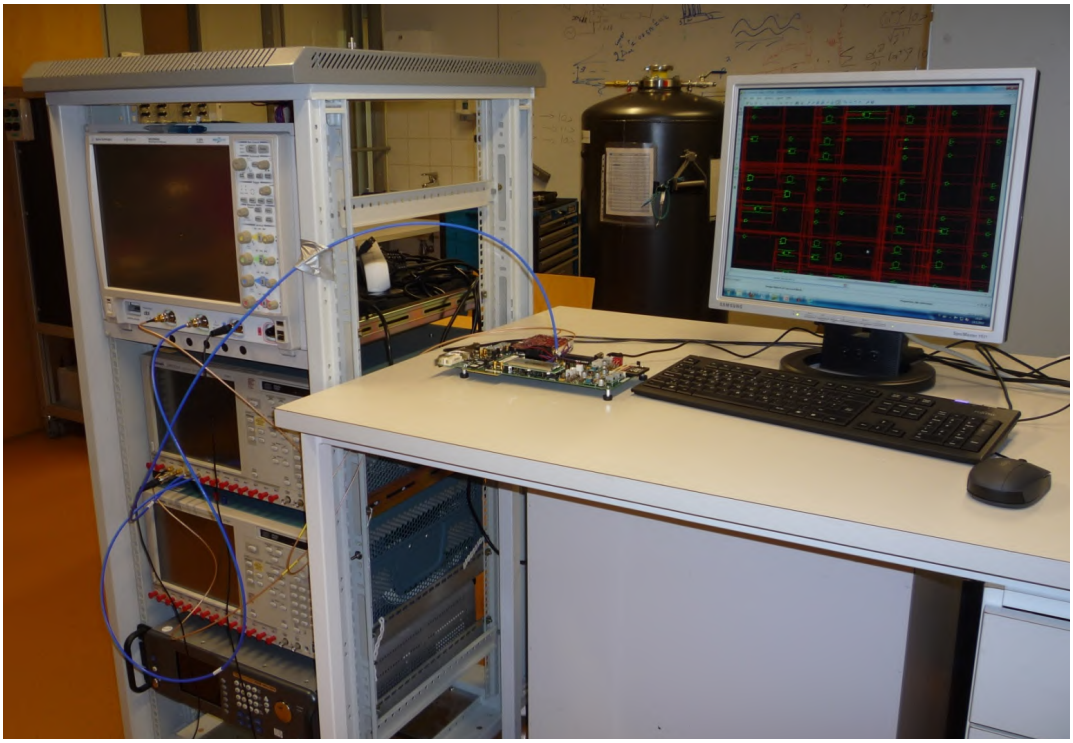

Fig. 29. Equipment setup.

Appendix B. The model of the ensemble average filter

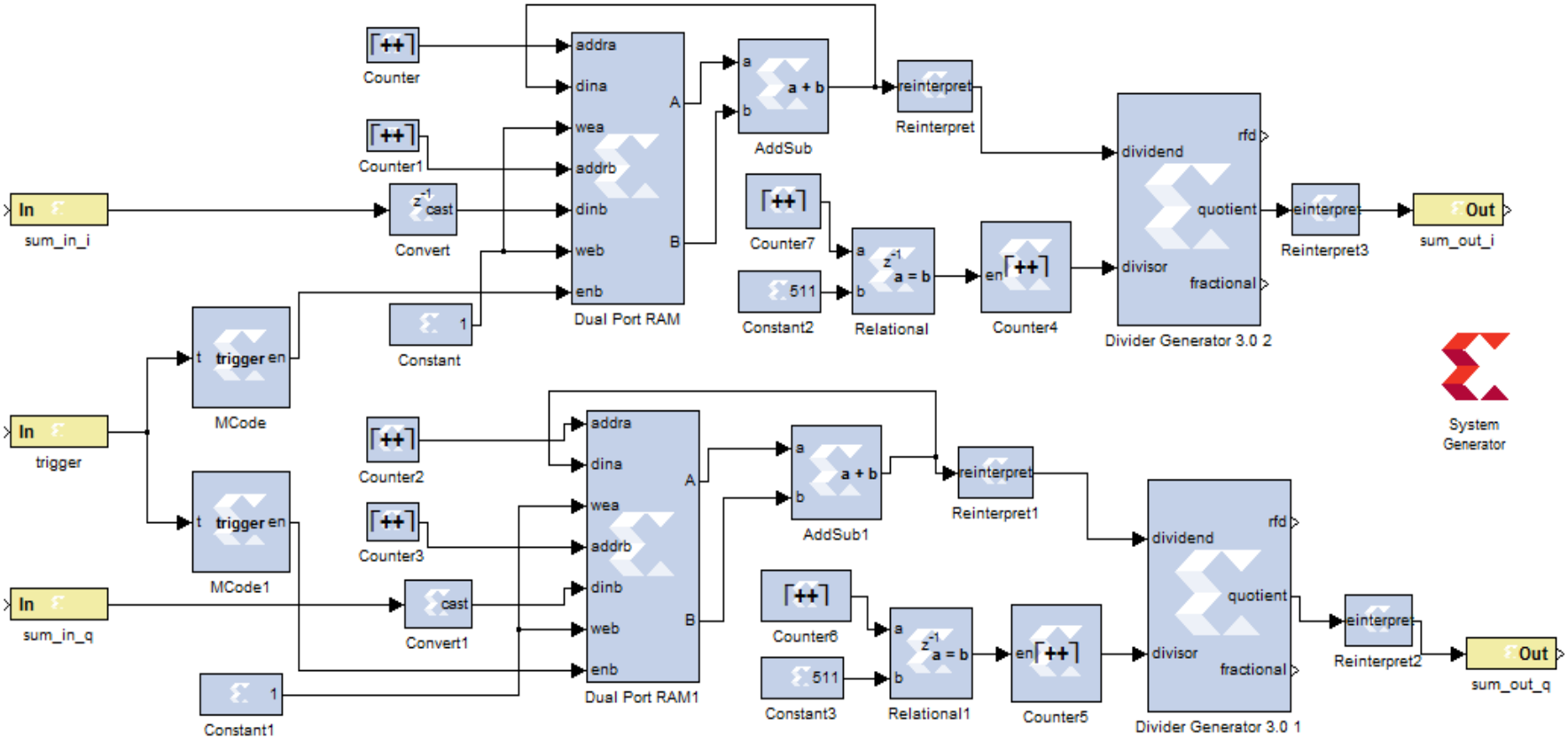

Fig. 30. Ensemble average filter Simulink model.