UNIVERSITY OF TWENTE.

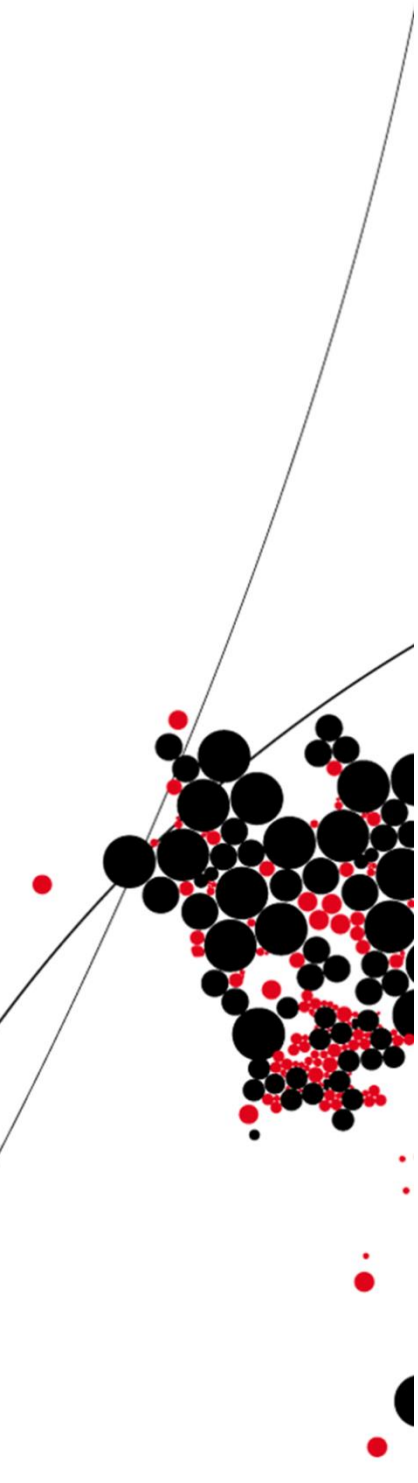
Faculty of Electrical Engineering, Mathematics & Computer Science

Implementation of a Distributed Correlator for Astronomical applications

Dominicus J.G. Moonen
M.Sc. Thesis assignment
November 2014

Supervisor:
dr. ir. M.J. Bentum

Telecommunication Engineering Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands



Index

1 INTRODUCTION

This report is part of the master Electrical Engineering with the specialization Telecommunication Engineering. The goal of the project is to build a working prototype for distributed correlation. The prototype is specifically designed to be used in an aperture synthesis array. The array examines signals in a relatively low frequency band which can't be received on earth due to the ionospheric cutoff.

Until the recent development of the CubeSat standard it wasn't financially feasible to send a constellation of satellites into space. Compared to the billion dollar satellite programs, the tens of thousands of dollar CubeSat projects are cheap. This creates new opportunities for scientific discoveries. The universe has not yet been mapped in these low frequency regimes.

A proposal for orbiting low frequency antennas for radio astronomy (OLFAR) suggests the creation of an aperture synthesis array (for example) for exploring the early cosmos at high hydrogen redshifts, for the discovery of planetary and solar bursts in other solar systems, for a tomographic view of space weather and perhaps the most interesting part 'the unknown'. The low frequency band is one of the last unexplored frequency bands.

A previous study showed that an FX-type correlator should be developed on FPGA's with a distribution system for computational efficiency and scalability. This report shows the design and implementation of a distributed correlator for the astronomical application designated OLFAR.

The following part will show the structure of the report and will shortly describe the contents of the chapters.

Chapter 4 will describe the system in which the correlator is used and describe why an FX-type correlator is implemented on FPGA's. This chapter can be seen as a short recapitulation of the internship report.[1]

An FX-type correlator basically consists of a Fourier transform and a complex multiplier. Chapter 5 will show which tools were used in developing the correlator. It will also show that the various parts of the correlator were tested individually before integrating them into the system. The distribution system is also designed and tested.

The next step in designing a prototype is connecting the parts. Therefore, Chapter 6 describes the total design of the prototype with the emphasis on signal flow. Some practical problems arose, like clock synchronization and control signals. Also a test bench was developed to verify the operation of the prototype. The test bench consists of a defined input signal that has a known output signal. This is first tested in simulation, followed by a hardware co-simulation. The hardware co-simulation runs on the connected FPGA's.

The results are then discussed in Chapter 7. Here is concluded that the prototype is indeed a working distributed correlator, however the output signals are not yet formatted correctly. There are some bugs that need to be fixed. These are discussed in Chapter 8. Also more general recommendations for future work are made.

2 SYSTEM OVERVIEW

This study is based on the proposal for Orbiting Low Frequency Antenna's for Radio astronomy (OLFAR)[2], which is a space based interferometer with an autonomous distributed sensor system. It will be used to explore a new low-frequency band that is located below the ionospheric cutoff frequency of approximately 10 MHz. The following chapter describes OLFAR and why it's being developed. First it's explained why aperture synthesis is used, followed by the implementation of the aperture synthesis array.

2.1 APERTURE SYNTHESIS

The angular resolution of a telescope is ultimately diffraction limited and is given by:

$$R = \lambda/D$$

With D as the measure of the aperture size in meters, λ is the wavelength in meters and R is the resolution given in radians. While going down in frequency (i.e. up in wavelength), one needs bigger aperture sizes to achieve the same resolution. In the OLFAR the longest baseline is 100km and the shortest wavelength is approximately 10m. So the max resolution will be around 20 arcseconds.

Deploying a 100 km wide dish in space is not feasible, if at all technologically possible. A technique called aperture synthesis is a specific form of interferometry and as the name suggests, it synthesizes an aperture with the diameter of the longest baseline.

Aperture synthesis has a trade-off with an actual aperture of the same size. The required computational power increases with the size of the constellation. i.e. the amount of nodes in the swarm. The received signals need to be cross-correlated to extract an image. This could be done on earth, however this would limit the capabilities of the constellation. The link budget puts a limit on the data rate back to earth. This would restrict the incoming data rate, if the constellation is used in real time without a buffer. The recent developments in the area of digital signal processing suggests it should be possible to do the correlation in space. The paper '*Distributed Correlators for Interferometry in Space*'[3] considers multiple scenarios, and suggests a Frequency Distributed Correlation with distributing the downlink and processing evenly among all the satellite nodes to be the most optimal.

The correlator is an integral part of the digital signal processing done in OLFAR. In this section the details of the OLFAR project will be briefly described, so the reader will gain an understanding of the astronomical application for which the correlator is developed.

2.2 OLFAR

In this section, the OLFAR concept is presented and its operation discussed. It describes OLFAR as a swarm of satellites orbiting the moon with multiple modes of operation depending on location.

The proposal for OLFAR describes a constellation of approximately 50 satellites that act as a space based interferometer with an autonomous distributed sensor system. Its suggested operating frequencies are between 0.3 to 30 MHz. All CubeSat's are equipped with dipole or tripole antennas for gathering astronomical data and patch antennas for the inter satellite communication link.

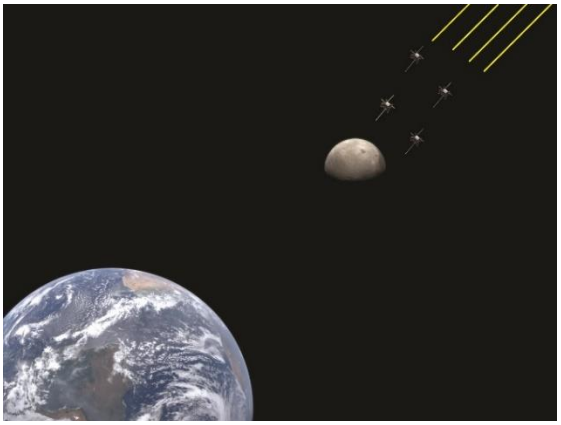


Figure 1: far side of the moon, gathering astronomical data

The proposal suggests a the moon's orbit as a deployment location. The following sketches demonstrate the three main phases of operation in this situation. Figure 1 shows that at the far side of the moon the constellation will be gathering astronomical data, mainly due to the reduced RFI from earth. Here the signal received can be directly converted with a Fourier transform and stored in memory until phase 2.

Phase 2 is displayed in Figure 2. The data is distributed between every node and the desired digital signal processing is done. Data distribution will be done via the Inter satellite communications link, which is being developed in another project. Therefore the prototype will not implement any known protocols for transmitting and receiving data. It will be shown in section 5.4 how data is distributed between the nodes.

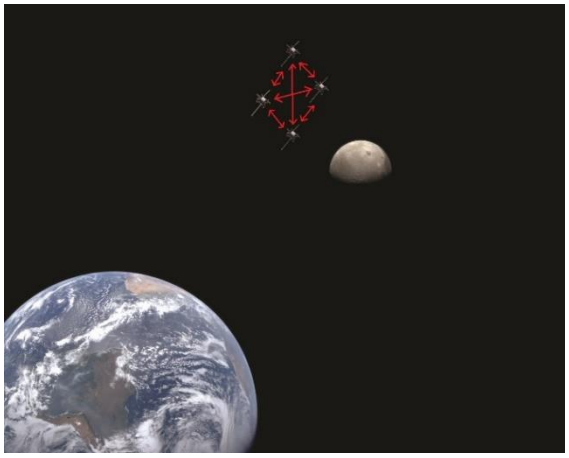


Figure 2: Lunar side, inter satellite communication and DSP

Next the multiplication part of the correlator is activated and starts reducing the datastreams in orders of magnitude. The data is now ready to be send back to earth, which is shown in Figure 3. At this time the constellation is closest to the earth, which is best for transmitting the processed data back to earth. This is because the link budget will be the most lenient (due to lowest free-space path loss) and therefore the highest data rate can be achieved.

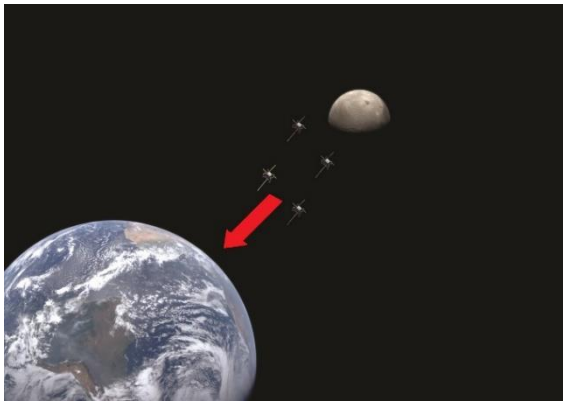


Figure 3: Earth side, Transmission back to earth

Figure 4 shows a flowchart that describes the distributed correlator. First the astronomical data is acquired by means of sampling the voltages across the antenna's. These signals are then Fourier transformed by a Fast Fourier transformation, which divides the signal into frequency bins. The frequency divided data can then be distributed across all nodes, so the computational load on every node is equal. The computation that needs to be done, is a complex conjugate multiplication of every signal with every other signal. Technically only half of all combinations, because the cross-correlation matrix is Hermitian.

2.3 PROCESS DIVISION

The main concept to be demonstrated is the distributed correlation. Distributed in the spatial sense. i.e. the processor at location A computes part of all the multiplications, while the processor at location B computes another part etc.

To explain this concept one could use the following simple analogy. Consider the complex conjugate multiplication as a huge number of simple multiplications that need to be done.

One could use a extremely fast Central Processing Unit (CPU) that does every multiplication, one by one. Or maybe if it's a multicore processor maybe four or eight multiplications simultaneous. A Graphical Processing Unit (GPU) for instance consists of a huge number of cores (assume for simplicity for instance 100 cores), all capable of doing these relatively simple calculations. The CPU needs to have an internal clock that is at least a 100 times faster than the GPUs to actually be the faster at calculating the multiplications.

Now this analogy demonstrates distribution at a single location, which in space would mean a single point of failure. Now consider for instance 10 GPUs, all doing a portion of the calculations in different locations. Distribution is now done spatially as well as internally. Assuming the size of the data that is being processed is so large that the data distribution time can be neglected. The spatially divided load will be processed much faster and now without a single point of failure.

Therefore it is emphasized, that the concept to be demonstrated is a spatially distributed correlation. The next section will provide a basic explanation of correlation, why correlation is required in aperture synthesis and which types of correlators are most commonly used. This is followed by a description of frequency division distributed correlation.

Figure 4 flowchart of distributed correlator as implemented in OLFAR

2.4 CORRELATION

The goal of OLFAR is to create an image of the 'sky' at low frequencies. This image is called the source brightness distribution. One can calculate the image by using the "*Van Cittert Zernike Theorem*", which states that the spatial correlation of the electric field in the uv-plane is related to the source brightness distribution by a two dimensional spatial Fourier transform [4]. The uv-plane is defined in a coordinate system in which baselines are represented by points, which can be determined from the position vectors of the antennas.

$$\Gamma_{12}(u,v,0) = \iint_{source} I(l,m) \cdot \ e^{-j2\pi(ul+vm)} \cdot dl \cdot dm$$

With I(l,m) being the brightness distribution and Γ being the mutual coherence function. The latter is related to the electric fields at two positions in the uv-plane by:

$$\Gamma_{12}(u,v,\tau) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} E_1(t) \cdot E_2^*(t-\tau) \cdot dt$$

The electric fields originate from a cosmic source, which travel the universe in every direction, some of which will be directed towards earth, or in our case towards the array of antennas. Antennas are nothing more than devices that convert (cosmic) EM-waves into voltages. In case of cosmic sources the signal can be considered a continuous random process. To simplify the explanation of correlation, one can consider it to be the comparison of two signals

As one can see from the expression, one can calculate the mutual coherence function by means of a special kind of convolution. Which is correlation and this is calculated for every baseline, with the values maped to uv-coordinates. The projected baselines are represented in a different coordinate system, that is assumed "parallel" to part of the sky under observation.

From the cross-correlation theorem (which is analogous to the convolution theorem) it follows that a cross-correlation of complex signals in the time-domain, simplifies to a multiplication of complex conjugates in the frequency domain. In case of digital signal processing, this reduces to:

$$S_{XY}[\omega] = X[\omega] \cdot Y^*[\omega]$$

The next section will describe two different types of correlators and will show why the FX-type was recommended for OLFAR.

2.4.1 Types of correlation

Both the FX and XF correlator consist of a Fourier transform('F') and a cross-multiplication('X'). The order of implementation is irrelevant, since these are linear operations.

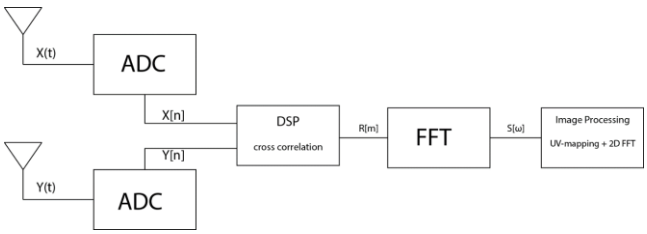


Figure 5: Signal Flow of a XF correlator

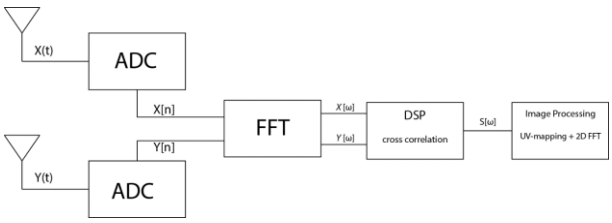


Figure 6 Signal Flow of a FX correlator

Notice that from the signals a spectral density is calculated. The cross-correlation can be seen as the amplitude visibility, which in this case of the XF-correlator is a function of lags. For the '*van Cittert-Zernike theorem'*[4] to hold, the visibilities need to be a function of frequency. The cross-correlation

is related to the spectral density (i.e. visibility as function of frequency, or flux density given in $W \cdot m^{-2} Hz^{-1}$) by means of a one dimensional time to frequency Fourier transform according to the Wiener-Khinchin theorem[5]. The type of Fourier transform is emphasized, since the brightness distribution of the sky is related to the mutual coherence function by means of a two dimensional spatial Fourier transform. This is often used in the post-processing to create a brightness map.[4]

In OLFAR the number of calculations (and thus the calculative power needed) increases with the size of the system (i.e. amount of nodes). A comparison of resources needed was made between a XF and FX type of correlator in [3].

N_x	Number of real multipliers
N_{sig}	Number of signals
N_{lags}	Number of lags
Δf_i	Instantaneous bandwidth
f_{sys}	Processing frequency

$$N_X^{xf} = 2{N_{sig}}^2 \cdot N_{lags} \cdot \frac{\Delta f_i}{f_{sys}}$$

N_x	Number of real multipliers
N_{sig}	Number of signals
N_{bins}	Number of frequency bins
Δf_i	Instantaneous bandwidth
f_{sys}	Processing frequency

$$N_X^{fx} = 2N_{sig} \cdot \frac{\Delta f_i}{f_{sys}} \cdot \left[N_{sig} + \log_2 N_{bins}\right]$$

Comparing this to the amount of multipliers needed for an XF correlator with the same amount of lags/number of frequency bins, one can conclude the FX correlator is favoured for scalability. As the spectral resolution goes up the amount of multipliers needed in an XF correlator scales linearly, while this is logarithmic for an FX type.

As was shown in Figure 6, the FX correlator converts the signal from time domain to frequency domain by means of a FFT. Figure 7 shows the resulting digital signal is divided into frequency bins and how this can be distributed over multiple nodes. To explain the basic concept of frequency division distributed correlation, Figure 8 shows a possible implementation for a three node system.

Figure 7 shows that every incoming signal is directly fast Fourier transformed into a signal with different frequency bins. Every node is then tasked with the cross multiplications between the different signals of a single frequency bin.

This would result in the following multiplications, this is only shown for node 1, since the same will hold for all other nodes:

$$\begin{aligned} Node\ 1 \rightarrow\ & (f_{1,1} \cdot f_{1,2}^*), \quad (f_{1,1} \cdot f_{1,3}^*), \\ & (f_{1,1} \cdot f_{1,4}^*), \quad (f_{1,1} \cdot f_{1,j}^*), \\ & (f_{1,2} \cdot f_{1,3}^*), \quad (f_{1,2} \cdot f_{1,4}^*), \\ & (f_{1,2} \cdot f_{1,j}^*), \quad (f_{1,3} \cdot f_{1,4}^*), \\ & (f_{1,3} \cdot f_{1,j}^*), \quad (f_{1,4} \cdot f_{1,j}^*) \end{aligned}$$

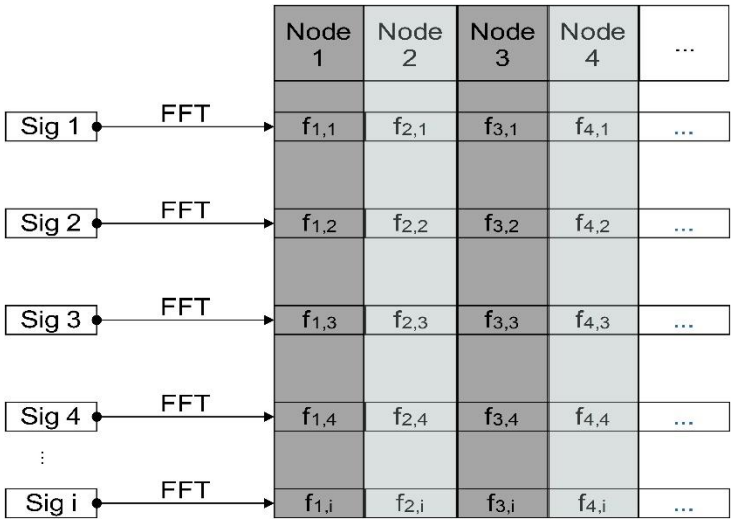


Figure 7: Division and distribution by frequency

Or in a expression:

$$S_{i,j,k}(f_i) = f_{i,j} \cdot f_{i,k}^* \qquad with\ (j < k)$$

The frequency is (or node that is processing it) designated by i. While j and k designate the signals of which nodes are being multiplied. By stating that j has to be smaller than k, one assures that only

half of every combination is calculated. Which suffices, since the cross-correlation matrix is Hermitian.

This example shows that this process is ideal for parallelization of the calculations, and thus for distributing the processing power. In the example shown in Figure 7, one can see that every node is responsible for the signal processing of a certain frequency, which in reality will be a frequency band. Figure 8 shows how this distribution could be implemented inside a three node system of satellites. This figure shows the three most important parts of a distributed correlator system. A Fast Fourier Transform, a data distribution system and the correlator. The correlator is basically an algorithm that implements complex conjugate multipliers.

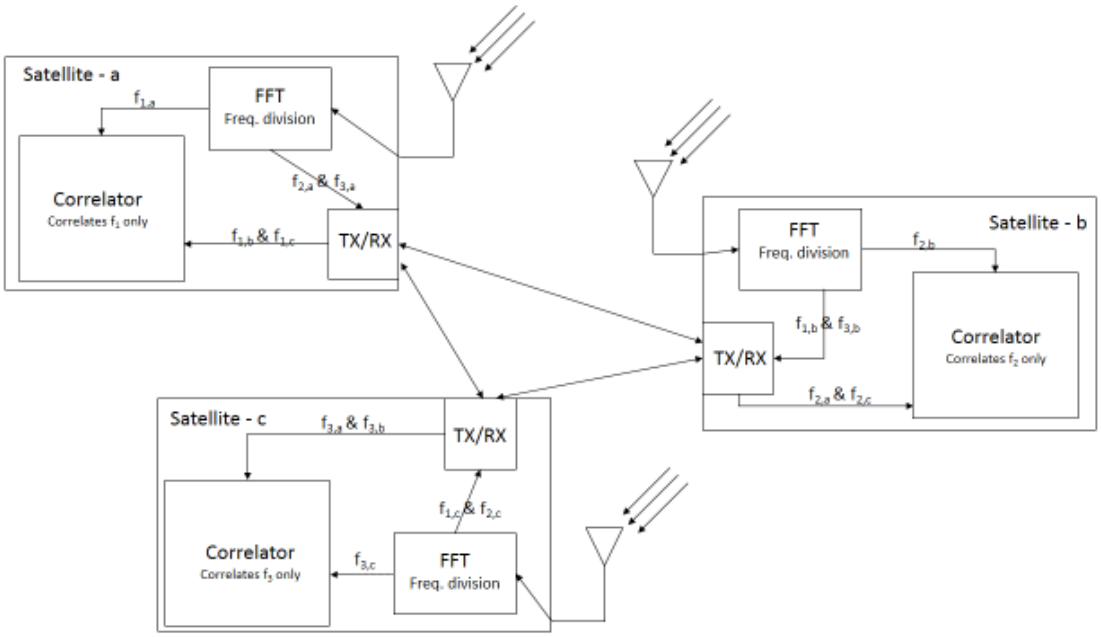


Figure 8: A sketch of the basic concept of (frequency) distributed correlation

Next a comparison of different “processors” will be given, to explain the choice of FPGAs (Field Programmable Gate Array).

2.5 FPGA BASED CORRELATOR

Now that it's clear "*what*" is being developed and "*why*". The next important aspect will be the "*how*". It was already mentioned in the preface that the correlator will be implemented on a Field Programmable Gate Array (FPGA). Figure 9 illustrates a comparison of the availability of specialized programmers versus the resource efficiency of alternative processing units. The alternative processing units considered are multicore CPUs, GPUs and Assignment Specific Integrated Circuits (ASIC). It's an interesting challenge to see if FPGA programming is accessible for researchers not familiar with the necessary programming skills needed

CPU's are primarily used for serial computation power and thus are fundamentally unsuited for parallel computing. They also have a huge overhead problem, since the CPU is the most general processor available. Thus CPU's are resource inefficient compared to a FPGA. The FPGA programmer decides how many recourse are dedicated to the task at hand.

The FPGA can be programmed to focus all its resources on multiplying and therefore work more efficiently. An ASIC can only be used for the one task that its designed for. It's not programmable and thus inherently inflexible. Also due to the development time an ASIC is not used. The timespan of this project is 6 months and thus it wouldn't be feasible to develop an ASIC.

According to [6], GPU's are on the other end of the spectrum as a possible accelerator for computational intensive applications. GPU's provide massive parallel execution resources and high memory bandwidth. In general, FPGA's provide the best expectation of performance, flexibility and low overhead, while GPU's tend to be easier to program and require less hardware resources. However, commercial off the shelf GPU's tend to be graphical cards with a PCI-express interface designed to be used in desktops in combination with a motherboard and CPU, which is suited for on earth correlation. However, considering the CubeSat system (with limited available power and physical dimensions) for which the correlator is developed, the FPGA's seems to be the more logical choice. Also a FPGA can be programmed to act as a software defined radio[7]. The following chapter will describe which methods were used to program the FPGA's. Also the three main components will be described and individually tested.

Figure 9: resource efficiency of different processing circuits vs the amount of specialized programmers

3 FPGA PROGRAMMING

From Chapter 2 it should be clear that a distributed correlator consists of three main components. The Fast Fourier Transformation, the complex conjugate multiplier and a signal distribution system. These components are designed and implemented on FPGAs. In the following chapter it's explained how these components are designed. First is explained which tools are used on which FPGA development board, next the tool setup is discussed, followed by the individual simulation and verification of the components.

3.1 DIGILENT NEXYS3

A decision needed to be made on which FPGA to use to develop the correlator. Two major FPGA development corporations have development boards available. The choice for Xilinx was made, due to the availability of the software licence. The Nexys3 development board from Digilent was chosen. Since this project is a prototype and used as a proof of concept, the finances available for development are limited. A single Nexys3 board costs approximately $140,- when academically used. Thus the board was chosen since it's cheap and meets all the requirements

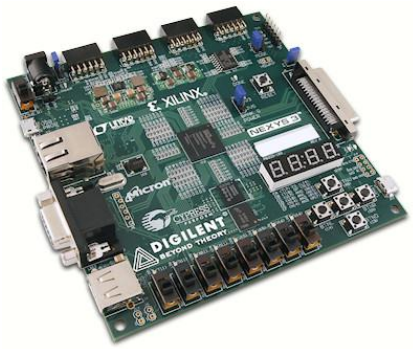


Figure 10: Nexys3 development board

The Nexys3 board is based on the Xilinx Spartan-6 LX16 FPGA. It has an internal clockspeed of 100 MHz, it contains 2278 slices each with four 6-input LookUp-Tables(LUTs) and eight flipflops, 576kbits of RAM and 32 Digital Signal Processing(DSP) slices.[8] This chip is known to be able to do FFT's and Complex Multiplications. More information can be found in the reference manuals[9], [10].

The development board also has a few interface capabilities. Available are VHDCI[1], USB-UART, PMOD[2], Ethernet, VGA. In section 3.4 a decision is made for using which connector for distributing the signal among multiple FPGA development boards.

Programming FPGA's can be done on multiple levels. Core design is done with the Xilinx embedded design kit (EDK) and/or Xilinx Vivado suite. One needs a comprehensive understanding of VHDL/Verilog to be able to design complex systems on this level.

A more accessible way of designing FPGA systems for researches is using the Xilinx system generator, which is interfaced with Mathworks MATLAB and Simulink. Hardware co-simultion enables a researcher to receive direct feedback and allows quick prototyping. The system can be exported as a processing core (pcore), which can in turn be imported in the EDK. Pcores can be considered as modules that can be combined to implement a complete system. This enables researches to develop separate cores simultaneously and implement them together into one design. An FFT pcore can be developed, which is used during phase 1 of the OLFAR system, next to a "cross-multiplication" pcore that would be used in phase 2.

This shows that when using high level design alternatives, the user does not sacrifice controllability. i.e. a researcher could contribute significantly to a FPGA programming project, without explaining in

[1] Very high density cable interconnect
[2] Peripheral Module interface, a standard defined by Digilent Inc.

great details to an embedded system programmer what is required. The correlator is thus developed through the Xilinx System generator.

3.2 Xilinx System Generator Setup

The Nexys3 board Hardware Co-simulation is not directly supported in Simulink, the following section describes a method of adding the support manually.

The System Generator Board Description Builder (SGBDB) shown in Figure 11, can be used to add hardware co-simulation support for arbitrary development boards. Using the settings as displayed, the Nexyx3 becomes programmable through the JTAG connection from Simulink.

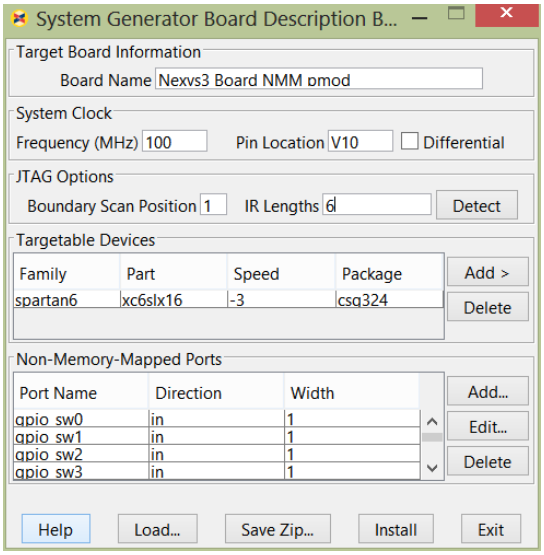


Figure 11: Board description Builder

The Non-Memory Mapped Ports (NMM ports) are used to create a library of in-/output ports which can be used in the Simulink design. In the reference guide of the Nexys3 board, one can find all the ports available [11]. In the example given here, the switches on the board are used as an input. They were connected to LED's, in Simulink displayed as output ports, as a verification that the mapping was done correctly.

After designing a component, one compiles the design and uploads it to the FPGA. The inputs and outputs of the compiled bitstream are represented in Figure 12. This allows Simulink to send and receive signals to and from the FPGA's

After adding the support for hardware co-simulation (hwcosim), parts of the correlator were designed and tested individually. First the Fourier transformation is created, followed by the complex multiplier and data transmission between the boards, which are the most crucial parts of the distributed correlator.

Data transmission is done between multiple Nexys3 development boards simultaneously. Therefore one needs to be able to program and run them simultaneously from Simulink. The programming over JTAG is done from Simulink using the Digilent plugin. One needs to specify the serial number of the board being programmed in the cable configuration. This is shown in Figure 13. It's emphasized here since this option is not mentioned in any tutorials, user guides or fora. There might be more even more parameters to be set

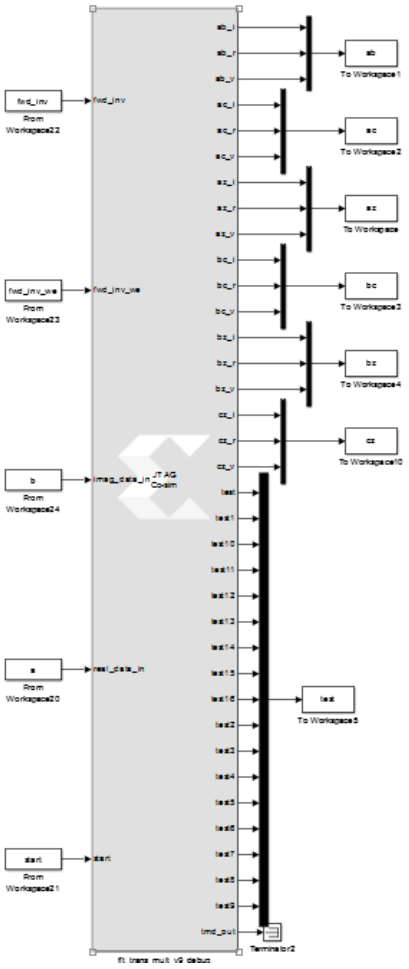


Figure 12: Block with inputs and outputs of the compiled bitstream, including debugging outputs

in this way, however due to lack of documentation these are unknown.

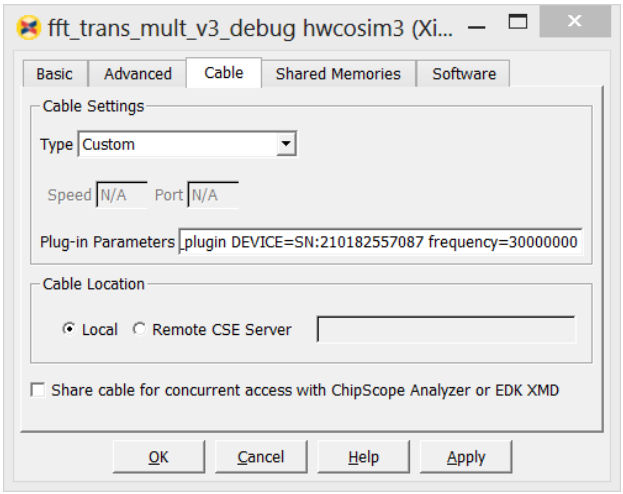


Figure 13: Cable settings for multi board hwcosim

3.3 Fast Fourier Transformation

Xilinx system generator toolbox allows the use of predefined Fourier transform modules as shown in Figure 14. One only needs to set parameters, such as the transform length, scaling options, precision, or implementation on FPGA. In the appendix one can see the selected parameters as implemented in the final design, displayed in Figure 23 through Figure 25. Note however the transform length varies across this report and will be mentioned where needed.

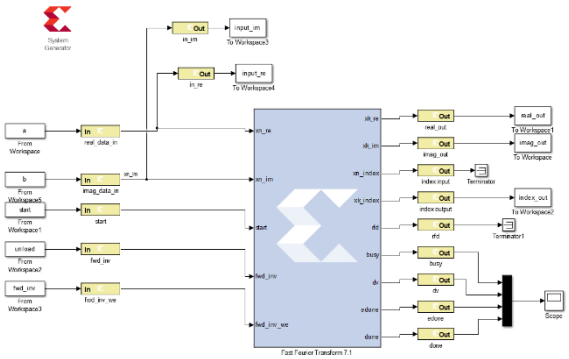


Figure 14: Xilinx FFT block as implemented in simulink

The input signal is quantized in a 10 bit signal with the binary point at 9, which implies the output of the FFT is a 16 bit number with a binary point at 9. Also it's a 32 point FFT, which is considered small for astronomical applications. It was used, because of the resource limitations of the Nexys3 board. One can easily increase the precision and length of the FFT when using more powerful and more expensive FPGA's. Since the prototype is only for demonstrating distributed correlation, the limited precision and transform length suffices.

The FFT block provides multiple operation settings displayed in Figure 15. The three main implementations of the FFT are, Radix 2, Radix 4 and pipelined. The pipelined implementation allows for continuous data processing, thus has the highest throughput. The high data throughput is at the cost of resources. Radix-2 and 4 are burst wise. This means it loads and processes data separately. It consumes less resources, however the transform time is longer. Resource vs throughput is shown in Figure 15. The streaming architecture was chosen, since this would be the most ideal one to be used during the data gathering phase.

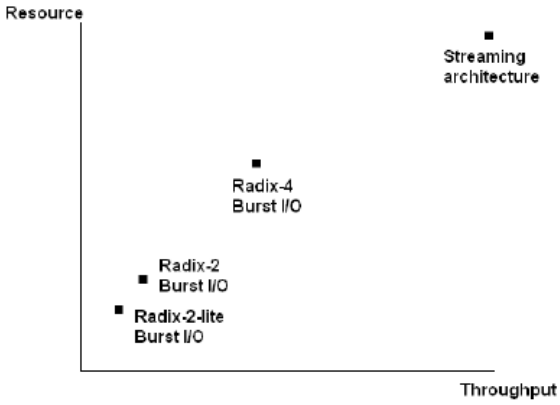


Figure 15: Resource versus throughput for architecture options

The method used to validate the FFT is a quantitative comparison between power density spectra resulting from Fourier transformations with different lengths of sine waves done on the FPGA, with the FFT done in Matlab. The sinewave signals where quantized in 32 bits, which is the same amount of bits used in Matlab's single precision FFT. The FFT lengths used were a 128 and a 256 point transformation.

The results are shown in the appendix in Figure 26 to Figure 32. As one can see, the errors are quite substantial. The errors most likely arise from truncation or rounding of values

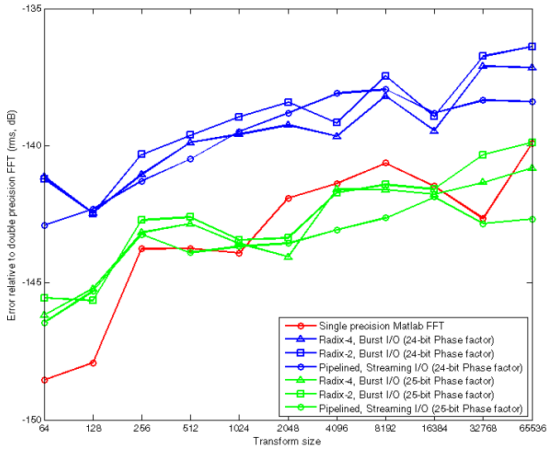


Figure 16: comparison of two levels of noise performance[9]

A better comparison would have been with the "Bit accurate C model FFT" that Xilinx provides. However, the script can only run under certain conditions (version of Matlab/Sysgen/Windows) and these were not met. Problems of not having the right dynamic link libraries arose. To solve these would have been too time consuming with little relevance to the design of the distributed

correlator. So from this point on, it's assumed the FFT works as expected with keeping in mind that the errors might be substantial in the final results.

An analysis on the accuracy of the FFT is done in [9], of which the results can be seen in Figure 16. Here the effect of increasing the length of the FFT and the phase factor width are shown. This proves that with a certain parameter set, one can calculate a relatively accurate Fourier transform.

3.4 Complex Multiplier

The complex multiplier is a mathematical operation done at the end of the correlator. Here the working of the block shown in Figure 17 is tested. It's possible the block uses some kind of buffer, however the results shown in Table 1 suggest differently.

The input vectors are defined as time series. i.e. at time interval 1, the first values are pushed. At time interval 2, the second values etc.

$$x(t) = [x(t_0);\ x(t_1);\ x(t_2);\ x(t_3)\ ]$$

So the output of the complex multiplier module will be an element wise multiplication of the vectors. The following vectors were defined:

$$a = [2 + 2i\ ;\ 2 + 1i;\ 1 + 2i;\ 3 + 3i\ ]$$

$$b = [5;\ 4 + 2i;\ 5 + 1i;\ 1 + 2i;\ 7 + 3i]$$

$$a_{valid} = [1;\ 1;\ 1;\ 1]$$

$$b_{valid} = [0;\ 1;\ 1;\ 1;\ 1\ ]$$

Notice that the length of vector *a* is 4 and the length of vector *b* is 5. The simulations assume zero inputs when no value is presented. So vectors *a* and a_{valid} get a trailing zero added in the simulations.

$$a = [2 + 2i\ ;\ 2 + 1i;\ 1 + 2i;\ 3 + 3i; 0]$$

$$a_{valid} = [1;\ 1;\ 1;\ 1; 0]$$

Now with these inputs defined, the first output value expect will be 5·(2+2i) = 10+10i. Which is indeed the first outputted value shown in Table 1. However, the validation signal output is a 0, since this signal is a result of a logical AND. a(0) = 1, and b(0) = 0, thus as expected from a logical AND, the output is a 0.

Table 1: results of complex multiplier, simulation

t	Valid_out	Imag	real
5	0	0	0
6	0	10	10
7	1	8	6
8	1	11	3
9	1	9	-3
10	0	0	0

Following the same steppes, one can confirm the results of Table 1. Notice however that it considers only three output values to be valid. This is due to the validation vectors of both signals only have three instances that overlap with a high signal. As was explained, the validation signal can be seen as the result of a logical AND. This shows the signals should be synchronized before routed into the complex multiplier.

Synchronization might be needed after distributing the resulting Fourier transformed signals. It depends on the method of data transmission. This is addressed in the next paragraph.

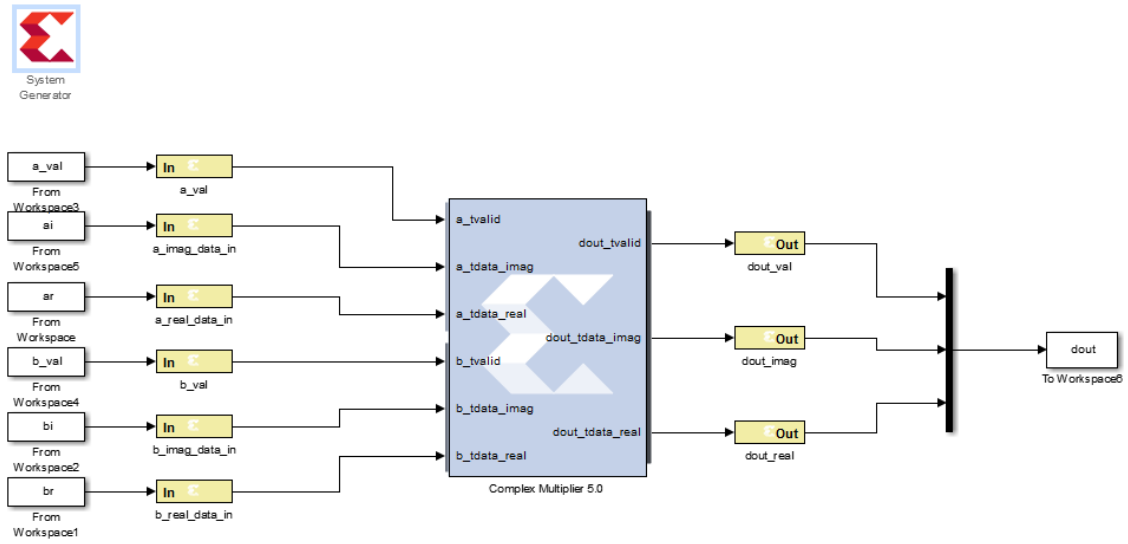


Figure 17: Xilinx Complex multiplier block (with AXI interface) as implemented in simulink

3.5 DATA TRANSMISSION

The transmission of data in this project is merely done for a proof of concept and thus doesn't need to be a wireless connection. The emphasis of this project is distributed correlation, in which it's assumed that the signal are correctly transported from node to node. However, it's essential to transport the signals between the boards, without using an intermediate storage device. In this case it would have been the notebook running Matlab/Simulink. It's essential, since the notebook would have represented a central processing node and the prototype is for an autonomous distributed sensor system.

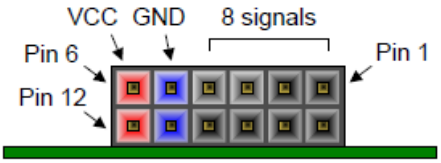


Figure 18: PMOD connector

The Nexys3 development boards have a number of possible data transmission lines. "Micro-USB UART, VHDCI, Ethernet, PMOD" where considered. Every connector needs a certain protocol. So next will be explained why the PMOD connector was chosen

Data has to be transmitted simultaneously to a specific location to maximize the efficiency of using distributed correlation. Thus Micro-USB UART connection was not used, since it's a serial port. It's not possible to receive signals from multiple nodes, while simultaneously sending different data to each node. The Ethernet ports needs a controller defined on the FPGA, which consumes resources. Also it's not yet clear if this is even possible to design via Simulink.

VHDCI has 68 pins with 20 matched pairs. This would have been the most ideal connector, however the bus-like cable with four connectors was not available and thus the PMOD connectors were chosen. Figure 18 shows that each 12-pin pmod connector has two 3.3V VCC signals, two ground signals and eight logic signals.

The eight logic signals have been divided into four receiving and four transmitting signals. Each development board has four pmod connectors, of which three are needed to interconnect all the boards.

Figure 19 shows the configuration of PMOD connections which was used to verify the transmission. The Bit-error-rate was found to be negligible for all connections without any encoding, decoding or error correction.

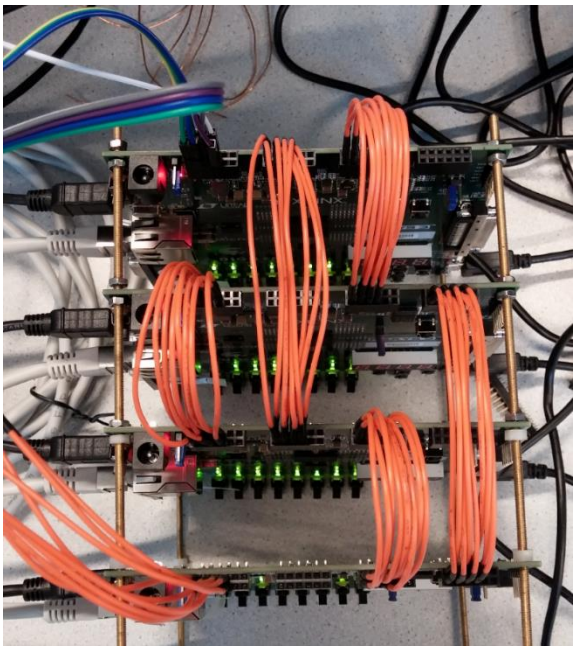


Figure 19: configuration without grounding between the PMOD connectors

4 SYSTEM INTEGRATION

The three main components of a FX type distributed correlator are the Fast Fourier transform, the complex multiplier and data distribution system. They have been designed and tested individually in Chapter 3. The following chapter describes the combining of the modules to implement a complete system. An overview of the total system can be seen in Figure 20.

As one can see, two new components have been added. The Time Divisional Multiplexer and the First-in-First-out memory blocks. These are needed to respectively transform and synchronise the signals. This is discussed in greater detail in section 4.1

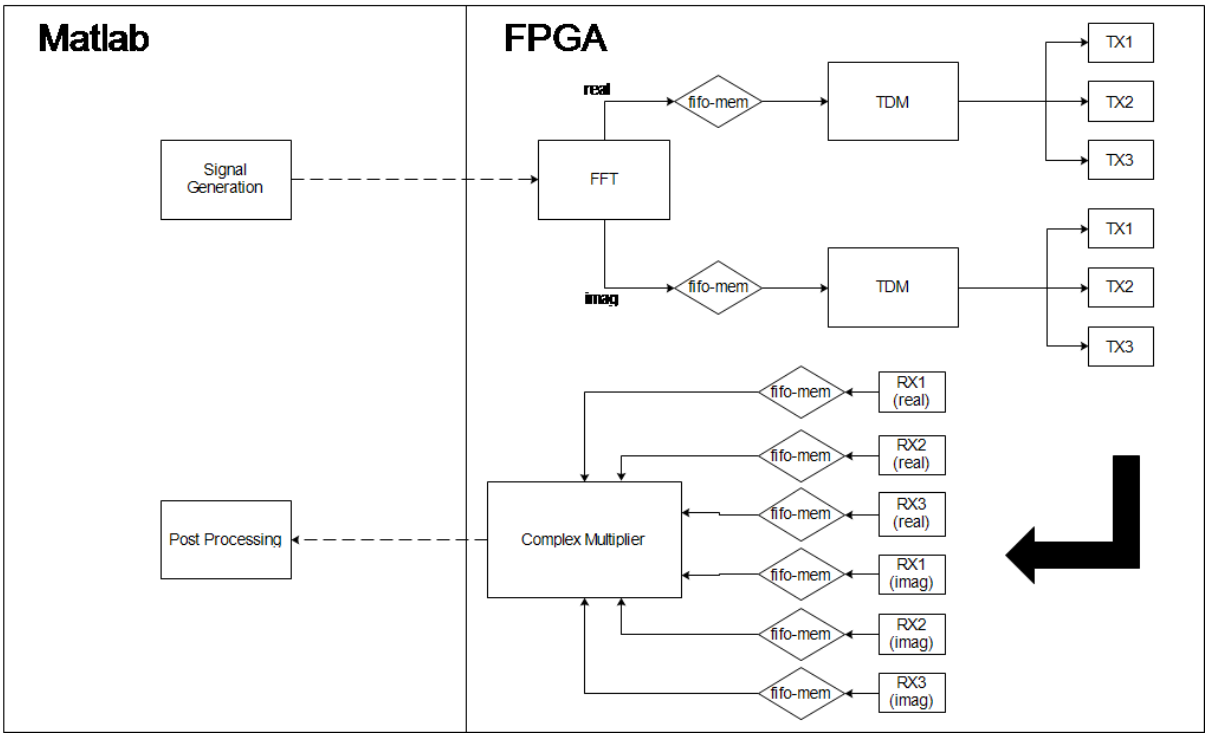


Figure 20: Total system overview, the TDD is considered part of the RX

4.1 SIGNAL FLOW

The previous chapters have explained that a FX-type correlator is being designed on multiple Nexys3 development boards by using Xilinx System Generator in combination with MATLAB/Simulink. As explained the correlator mainly consists of a FFT, complex multiplier and a distribution system. Both have been tested and verified individually. This chapter describes how these are combined and what extra components are added to assure it works and keep the design flexible.

Figure 20 has shown the total system overview, as it's implemented in Simulink. The Time Division Multiplexer(TDM), First in First out memory (FIFO-mem) and Time Division De-multiplexer(TDD) are needed to run the hwcosim. TDM is needed for converting the FFT output into a signal that can be transmitted through the PMOD connectors. The FIFO-memory banks are used as buffers and a way to synchronise the processing.

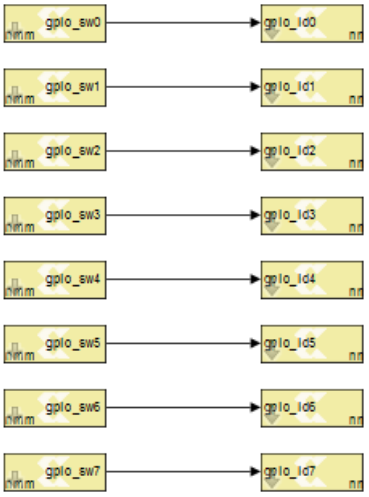


Figure 21: Box 1, GPIO switches and LEDS

Figure 33 shows the complete correlator system as implemented on one Nexys3 development board. Box 1, shown in Figure 21, consists of input signals controlled by switches and outputs that are connected to LED's. These are used for verification of Non Mapped memory ports.

Box 2 represents the Fast Fourier transform v7.1 of the Xilinx toolbox. Its options have been discussed in section 5.2. For the validation of the prototype a 32-point FFT has been chosen, due to the limit amount of resources available. The inputs consist of two's complement signed 10 bit words with the binarypoint located at 9. The outputs are 16 bit words with the binarypoint at 9.

Box 3 shows part of the FIFO memories used as buffers for the output signals of the FFT. The real and imaginary parts of the FFT have separate buffers. From this the signals are routed into a time division multiplexer. This can be seen in box 4. The 16 bit words at the output of the FFT need to be serialized. The PMOD connector in- and outputs only accept Boolean and Unsigned fixed point signals consisting of 1 bit(Ufix_1_0)[3]. The transferred signals are then stored in a FIFO memory after being parallelized. This means the Ufix_1_0 signal is translated back into a signed fixed point 16 bit signal with the binary point at bit number 9(Fix_16_9). One could see this as a Time Division De-multiplexer. This process is shown in box 6.

The transmission of the data is accompanied by the transmission of the validation signal. This validation signal is used as a control signal. One assumes that the transmission over the PMOD connectors are synced.

The FIFO's are then simultaneously read, so the signals are synchronized at the input of the complex multipliers. All combinations of signals (without the repetition of complex conjugates) are calculated in the prototype to compare results. This requires 6 complex multipliers, which all consist of three multiplications[10]. In the end product every node would multiply only a part of the signals. However in the prototype the entire signal is multiplied in every node, for comparison. Only six complex multipliers are needed, since the cross-correlation matrix is Hermitian.

[3] I^2C, SPI, H-Bridge and UART could be used. However this requires implementing the protocols in Simulink which is considered too time consuming.

It's now clear how a signal is being processed on one FPGA and distributed over the other nodes. However the system needs to be tested. First a description of the testbench is given. This explains how the system is tested. Next the system is simulated in Simulink for one development board. When the system passes this test, it's compiled and tested in a hardware co-simulation of four Nexys3 boards at one instance.

4.2 TESTBENCH

The input of the correlator was chosen to be a scaled complex step function. This signal is first processed by the FFT and then multiplied element wise with the complex conjugate of itself. The mathematical derivation of the expected result follows:

$$\mathcal{F}[0.1 \cdot [1+1i \quad 1+1i \quad 1+1i \cdots 1+1i \quad 1+1i \quad 1+1i]] = [\,3.2+3.2i \quad 0 \quad 0 \cdots 0 \quad 0 \quad 0]$$

$$\begin{bmatrix} 3.2+3.2i \\ 0 \\ \vdots \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 3.2-3.2i \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 20.48+0i \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

A few assumptions that were implicitly made need to be explained. The data transfer is perfect and there is no quantization error. Also the accuracy of the FFT is assumed to be infinite. (i.e. it's an analytical derivation). The next step is simulating the results in Simulink when the processing is done on only one FPGA.

4.3 SIMULATION

In Box 5 one can see the transmission from and to PMOD connectors. In the simulation these are replaced by direct lines from the TDM to the TDD blocks. This assumes that the signals are all transmitted and received correctly by every node, with synchronised clocks.

The simulation is run with an unity input vector scaled with 0.1, as was explained in section 4.2. The scaling was introduced to simplify the notation of the results. The results are shown in Table 2, here only integer time intervals are displayed. The system actually runs on 1/16th the time interval, which results in an output that is sampled 16 times in one integer time interval. The table only shows the results from three complex multipliers. The signals a, b (and c which is not displayed) represent the received signals from the other nodes. While the signal 'z' is the internally stored signal. Compared with the analytical result, one could conclude the correlator is working in simulation. The deviation between the expected values and the simulation arise from the errors made in the FFT and quantization of the signals.

Table 2: first 6 results of the correlator

t	a·b*	a·z*	b·z*	*Expected*
1	19.0701370239258	19.0701370239258	19.0701370239258	*20.48*
2	0.0201454162597656	0.0201454162597656	0.0201454162597656	*0*
3	0.0201148986816406	0.0201148986816406	0.0201148986816406	*0*
4	0.0200881958007813	0.0200881958007813	0.0200881958007813	*0*
5	0.0203285217285156	0.0203285217285156	0.0203285217285156	*0*
6	0.0204200744628906	0.0204200744628906	0.0204200744628906	*0*

After the verification, one compiles the design which is uploaded to each FPGA. Note that uploading the same design to each node improves the scalability of the project. The next section shows the results of the hardware co-simulation.

4.4 HARDWARE CO-SIMULATION

The FPGA's are now used as the platform on which the calculations are done. Therefore the assumptions of synchronous transmission do not hold anymore, however the FIFO-memory blocks should compensate for these potential problems.

Table 3 shows the result of the same complex multipliers used for the results in Table 2. The values need to be extracted by hand. The timescale of the results are still in 1/16th's, and therefore one expects 16 times the same result. The table displays only the extracted values of integer time values.

The multiplication between internal (*z*) and external(*a,b or c*) signal result in values that deviate from simulation and expected theoretical values. This is due to a problem with the FIFO memory containing the internal signal. The output is not synchronised and it is oversampled.

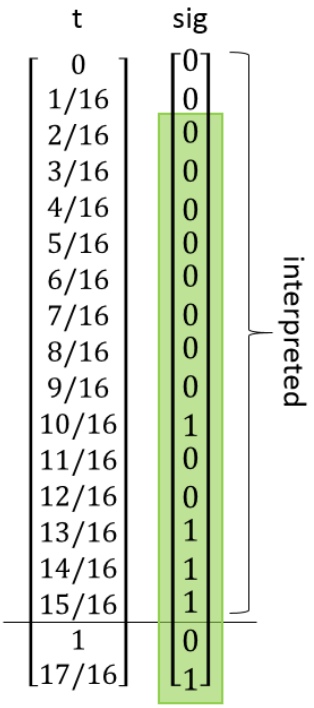


Figure 22: synchronization error in interpretation

However the working of the correlator is verified by the complex multiplication of two received signal. The deviation of the results from the expected values are large, the relative error made is approximately 60% for all values except the first value, however it arises from a synchronization error. For example, the outputs of the FFT are represented in 16 bits with a binary point at 9. At every integer time value the first bit of the 16 bit word is represented. After transmission the first bit of every word is shifted 2/16th time interval. The Time Division De-Multiplexer starts interpreting input values from integer time samples. Figure 22 shows a visualization example. The green box shows what should be interpreted as the Fix_16_9, however the interpretation is done starting at inter time values.

The goal of the project was to build a prototype for distributed correlation. Even though the results deviate significantly from the expected values, they show a working FFT, complex multiplier and distribution system. Which leads to the conclusion the distributed correlator works, however with a synchronisation problem.

Table 3: first 6 results of the correlator.

t	a·b*	a·z*	b·z*	Simulation	Exp.
1	18.9497070312500	-0.607208251953125 + 0.114227294921875i	-0.607208251953125 + 0.114227294921875i	19.0701370239258	*20.48*
2	1.24652862548828	0.155178070068359 - 0.0315208435058594i	0.155178070068359 - 0.0315208435058594i	0.0201454162597656	*0*
3	1.24713897705078	0.155693054199219 - 0.0285034179687500i	0.155693054199219 - 0.0285034179687500i	0.0201148986816406	*0*
4	1.24927520751953	0.156223297119141 - 0.0314674377441406i	0.156223297119141 - 0.0314674377441406i	0.0200881958007813	*0*
5	1.27002334594727	0.158477783203125 - 0.0286140441894531i	0.158477783203125 - 0.0286140441894531i	0.0203285217285156	*0*
6	1.27630996704102	0.158309936523438 - 0.0316276550292969i	0.158309936523438 - 0.0316276550292969i	0.0204200744628906	*0*

5 CONCLUSION

The goal of this project was to build a working distributed correlator, which would be used as a proof of concept for distributed correlation in astronomical applications. The goals of the research were to develop a Fast Fourier Transformation, a data distribution channel and an 'algorithm' for complex conjugate multiplication of the discussed combinations. These had to be combined in one big design to actually implement a working frequency divided and distributed correlator.

The FFT that was developed worked, however with some errors due to rounding/truncation and quantization. The data distribution was done through the PMOD connectors without the use of any specific protocols. It was shown that the data transmission was reliable and without any bit errors. The complex multiplier was shown to work only correctly if the inputs where synchronised, which was the reason for using first-in-first-out memory(FIFO-mem) in the complete design.

The total designed implemented the FFT, data distribution and complex multiplications with the addition of a few stages. A Time Division Multiplexer and Demultiplexer were added to convert the output for the FFT to a format that could be transferred over the PMOD connecters and interpreted back to the original format. To synchronise the whole process memory buffers where made by FIFO-memory blocks

Using a scaled complex unity signal the correlator was tested. The simulation showed a correctly working correlator. The hardware co-simulation showed that, using the same test, the prototype works with some bugs. From the results shown in Chapter 6, one can conclude that the FFT and complex multiplier are working as expected and thus suggest problems in the data transmission.

The goal of the project was to build a prototype for distributed correlation. As expected of prototypes it has its flaws and bugs. Nonetheless the concept of distributed correlation is proven. Since this has been proven, the conclusion may be drawn that the project was a success and a prototype of a distributed correlator now exists.

6 FUTURE WORK

The prototype of the distributed correlator works with the exception of some bugs. First a couple of recommendations for solving these problems are given, followed by more general recommendations that would contribute towards a fully functional wireless distributed correlator.

The results in Table 3 suggest there is a sampling problem in the received signals. One result of the multiplier is displayed 256 times, instead of the expected 16 times. To check where the problem occurs, one could export all the values at every point. These include before and after transmission, after de-multiplexing and interpretation, FIFO inputs and outputs and at last the in- and outputs of the complex multipliers.

One expects synchronised transmission, interpretation and multiplying of the real and imaginary parts of the signal. It is possible due to the lack of insight on the implementation the validation and control signals are delayed compared to the data signals in a (more or less) random manor. It's hard to debug these errors and one could blame this on the high level of designing. Recommended for future work is exporting the design into a pcore and implement it into the Xilinx EDK. One needs to define a certain interface before this is possible.

Next the more general recommendations for future work are given. The design of the prototype needs to be verified, therefore the signals were all completely cross-correlated in every node. The distributed correlator is actually created to use the combined processing power and thus distribute the load. This selectivity needs to be implemented.(i.e. node 1 only calculates the cross-correlations of frequency band 1, node 2 of freq. band 2 etc. etc.)

In this version of the distributed correlator, wired signal transmission was used. Therefor its suggested to upgrade this to a wireless transmission system. One of the benefits of using the PMOD connectors, is that already commercially available of the shelf components exist to implement this in the design.

As was previously indicated, the FPGA's in the prototype are relatively cheap. Therefore only a limited amount of resources were available on the FPGA and the system that was designed needed to be relatively small. For instance, the FFT was done with 32 points, while in OLFAR one needs at least 29.700 point FFT's[4]. So the scale of the system influences the scientific significance of the results. Therefore it might be interesting to develop a new design on more expensive development boards. This could also be beneficial for the wireless transmission, since some boards have wireless transmitters build on board.

When more expensive FPGA's are used, the exact parameters required for certain scientific goals need to be analysed. A study to the minimal FFT length, amount of bits used and other system parameters needs to be done. Thus a lot more work needs to be done before OLFAR becomes a reality.

In short the main recommendation would be the following. Use a couple of more expensive development boards to design wireless frequency distributed correlator with a 32k point FFT capable of selectively complex multiplying. The Nexys 4 development board is a suitable candidate, since it meets the requirements stated in the abstract submission by Dillon Engineering Inc. [12]. The Nexys 4 also has an On-chip analog-to-digtal converter, which might be interesting for work in the data acquisition area.

[4] Frequency range is 0.3-30MHz with a spectral resolution of 1kHz.

7 BIBLIOGRAPHY

[1] D. J. G. Moonen, “A Distributed Correlator Designed For a Space Based Interferometry Array,” 2014.

[2] M. Bentum, C. Verhoeven, and A. Boonstra, “OLFAR-orbiting low frequency antennas for radio astronomy,” 2009.

[3] R. Rajan, M. Bentum, A. W. Gunst, and A. Boonstra, “Distributed correlators for interferometry in space,” *… Conf. 2013 IEEE*, 2013.

[4] A. Thompson, J. Moran, and G. S. Jr, *Interferometry and synthesis in radio astronomy*. 2008.

[5] W. C. van Etten, *Introduction to Random Signals and Noise*. Chichester, UK: John Wiley & Sons, Ltd, 2005.

[6] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, “Accelerating Compute-Intensive Applications with GPUs and FPGAs,” *2008 Symp. Appl. Specif. Process.*, pp. 101–107, Jun. 2008.

[7] S. J. Olivieri, J. Aarestad, L. H. Pollard, A. M. Wyglinski, C. Kief, and R. S. Erwin, “Modular FPGA-Based Software Defined Radio for CubeSats,” pp. 3229–3233, 2012.

[8] “Digilent Inc. - Digital Design Engineer’s Source.” [Online]. Available: http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,897&Prod=NEXYS3. [Accessed: 02-Jun-2014].

[9] Xilinx, “Fast Fourier Transform v7.1,” 2011.

[10] Xilinx, “LogiCORE IP Complex Multiplier v5.0,” pp. 1–26, 2011.

[11] Digilent, “Nexys3 Board Reference Manual,” vol. 99163, no. 509, pp. 1–22, 2013.

[12] T. Dillon, “An efficient architecture for ultra long FFTs in FPGAs and ASICs,” 2004.

8 APPENDIX

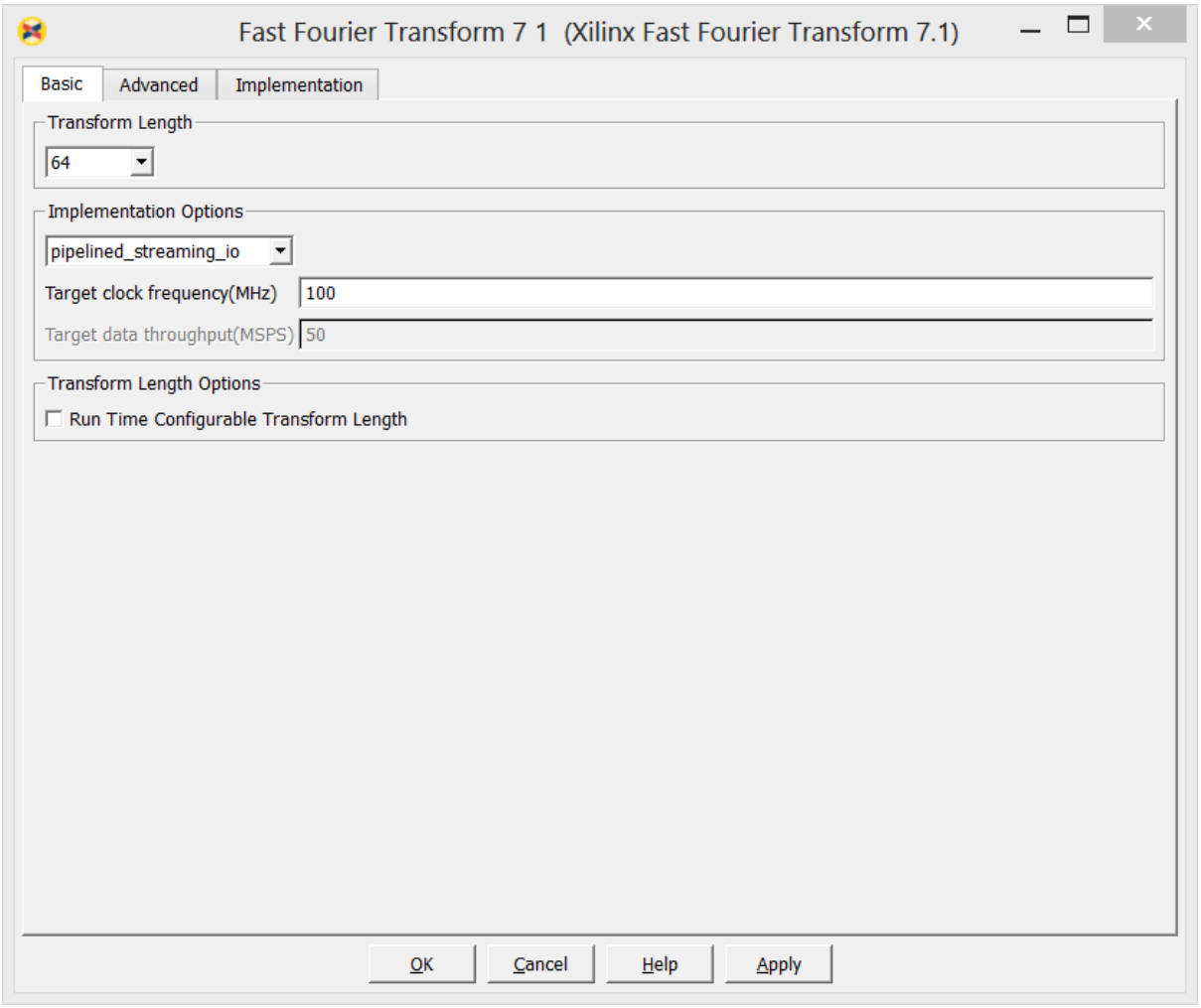


Figure 23:FFT options basic, transform length varies throughout the report

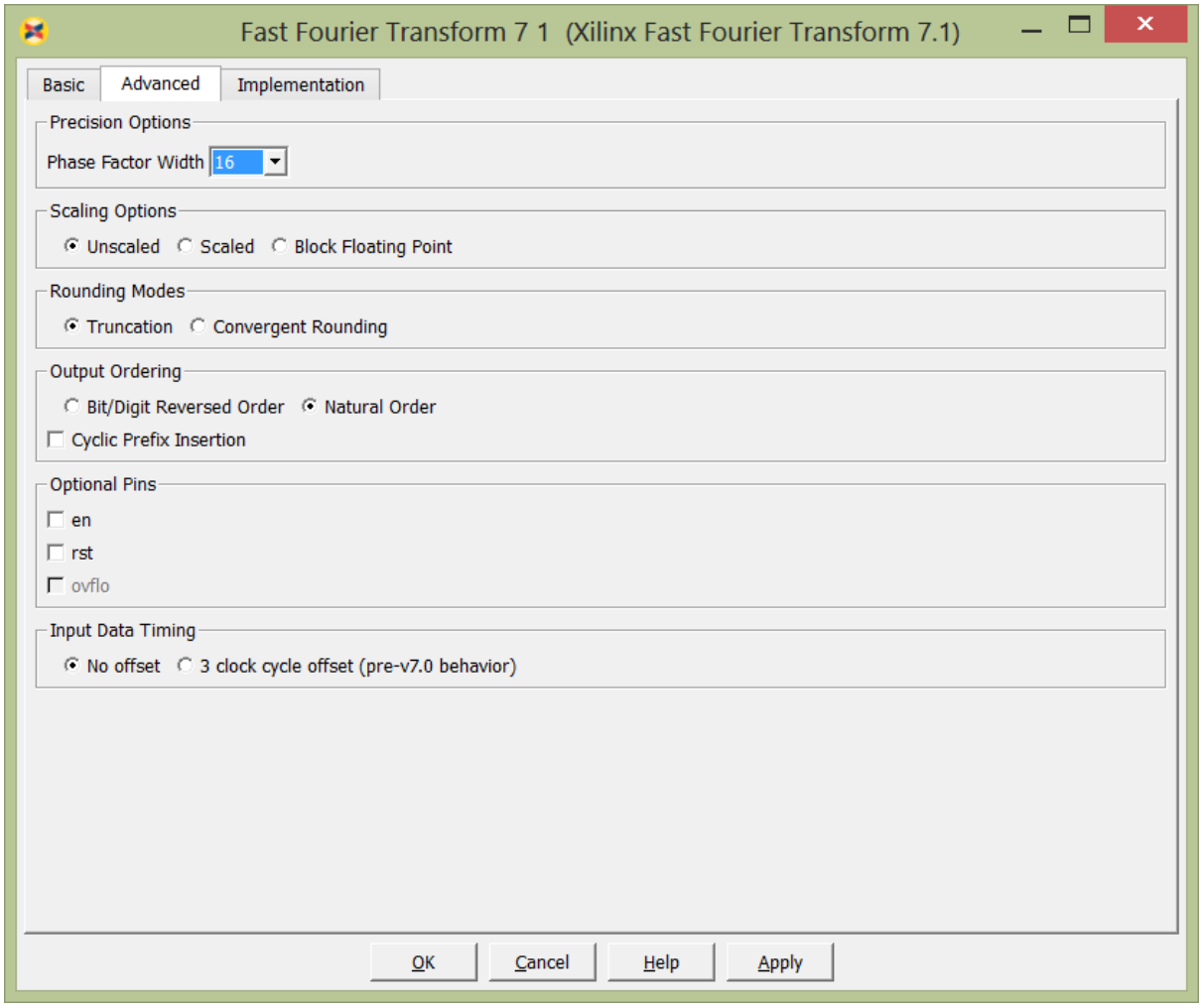


Figure 24: FFT options advanced

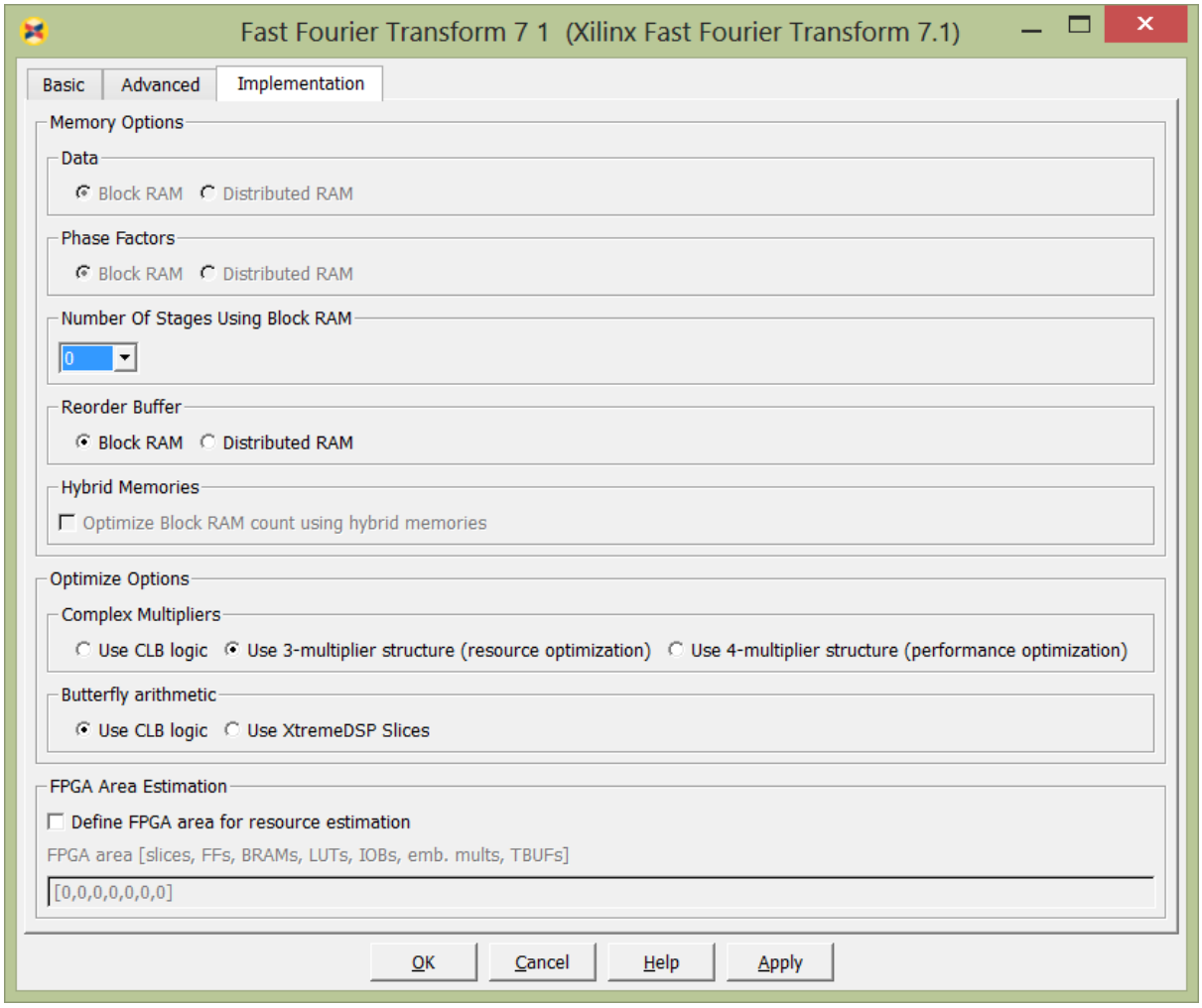


Figure 25: FFT options implementation

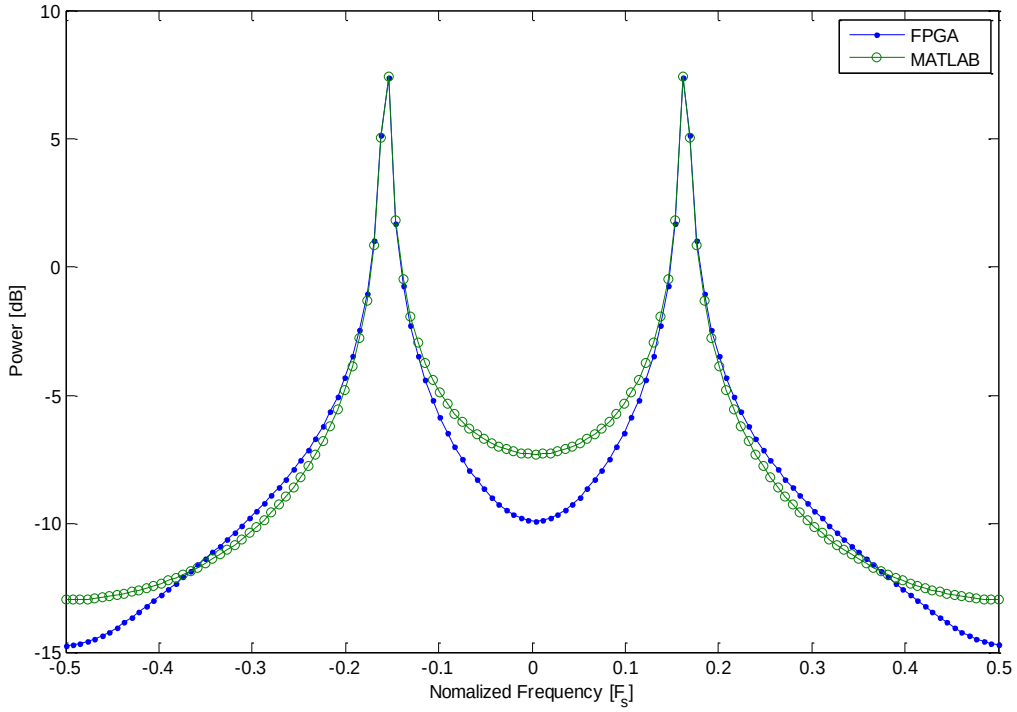


Figure 26: 128 point FFT, 1 freq. sinewave

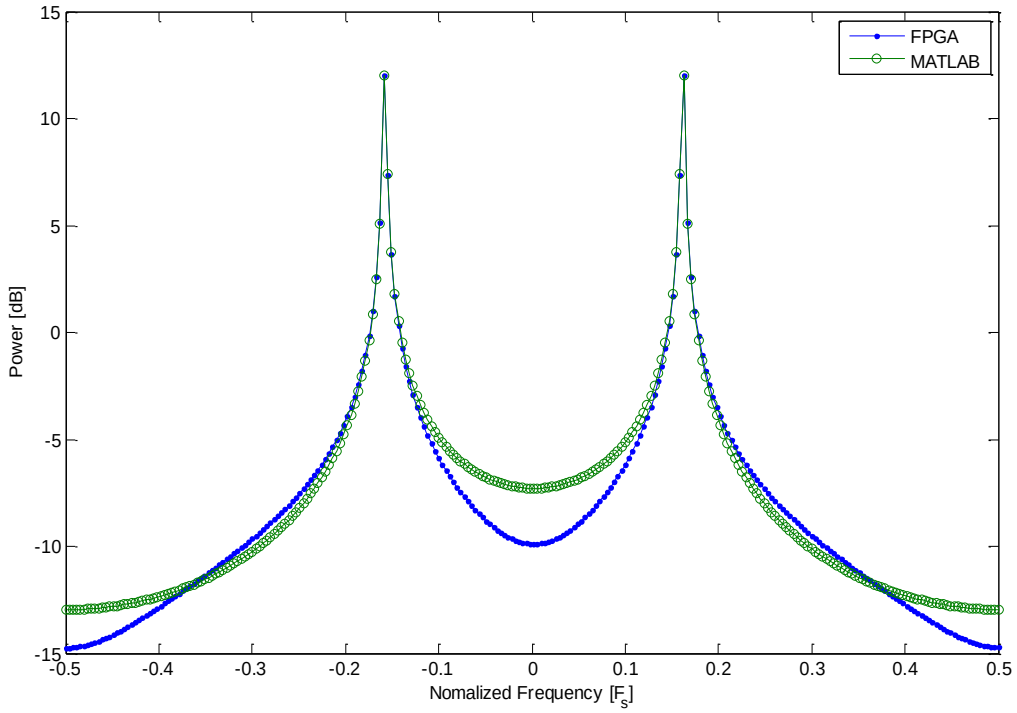


Figure 27: 256 point FFT, 1 freq. sinewave

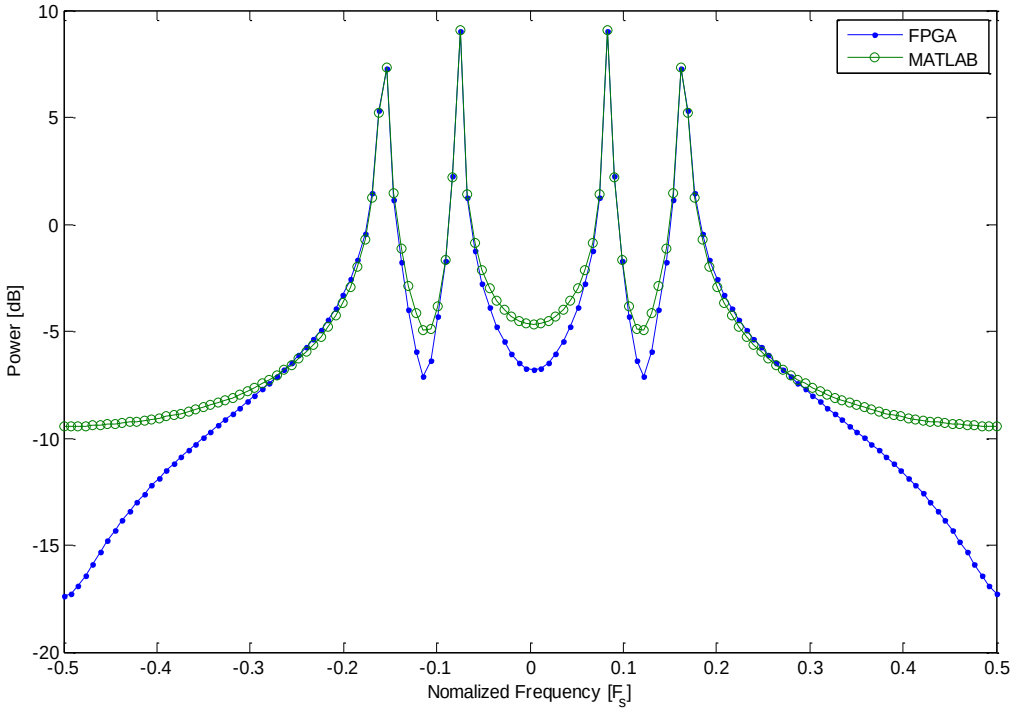


Figure 28: 128 point FFT, 2 freq. sinewaves

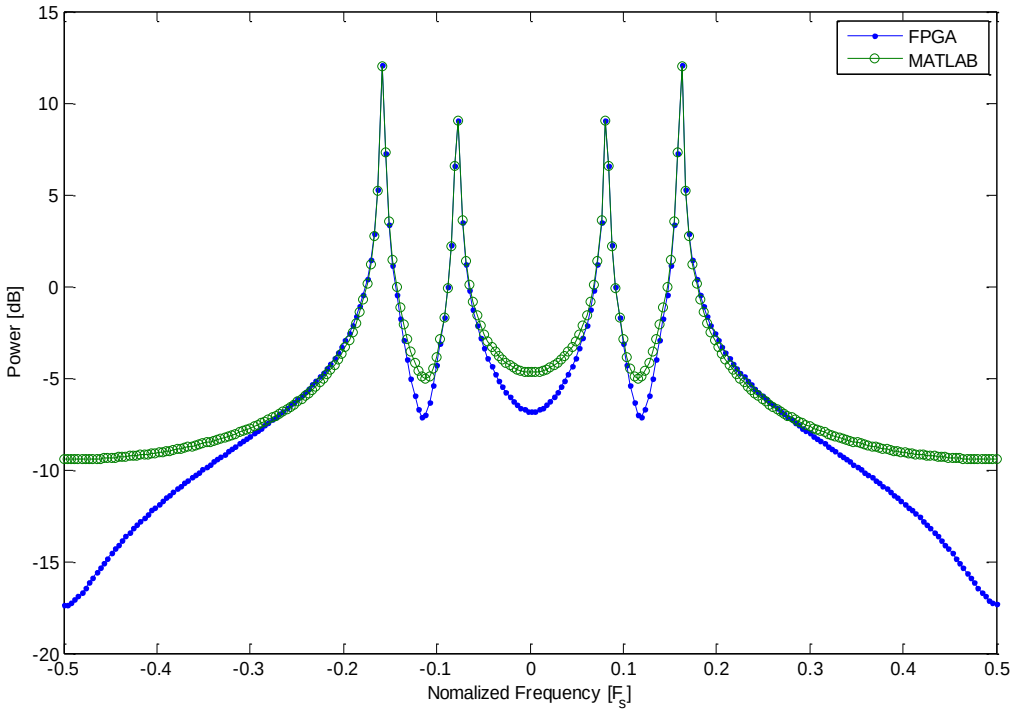


Figure 29: 256 point FFT, 2 freq. sinewaves

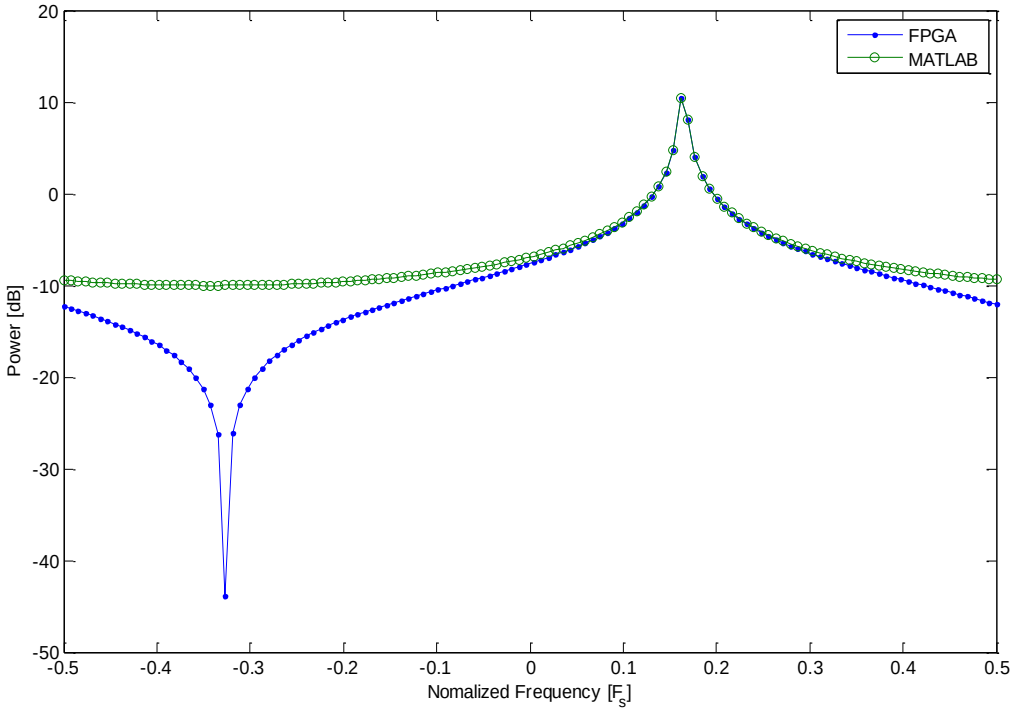


Figure 30: 128 point FFT, 1 freq. complex exp.

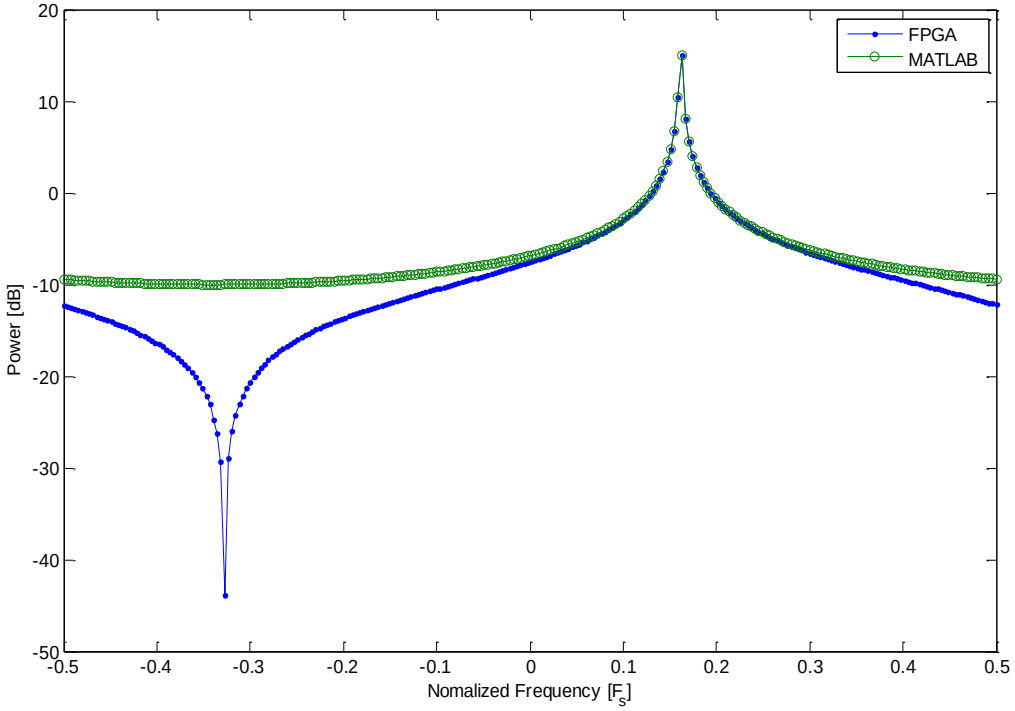


Figure 31: 256 point FFT, 1 freq. complex exp.

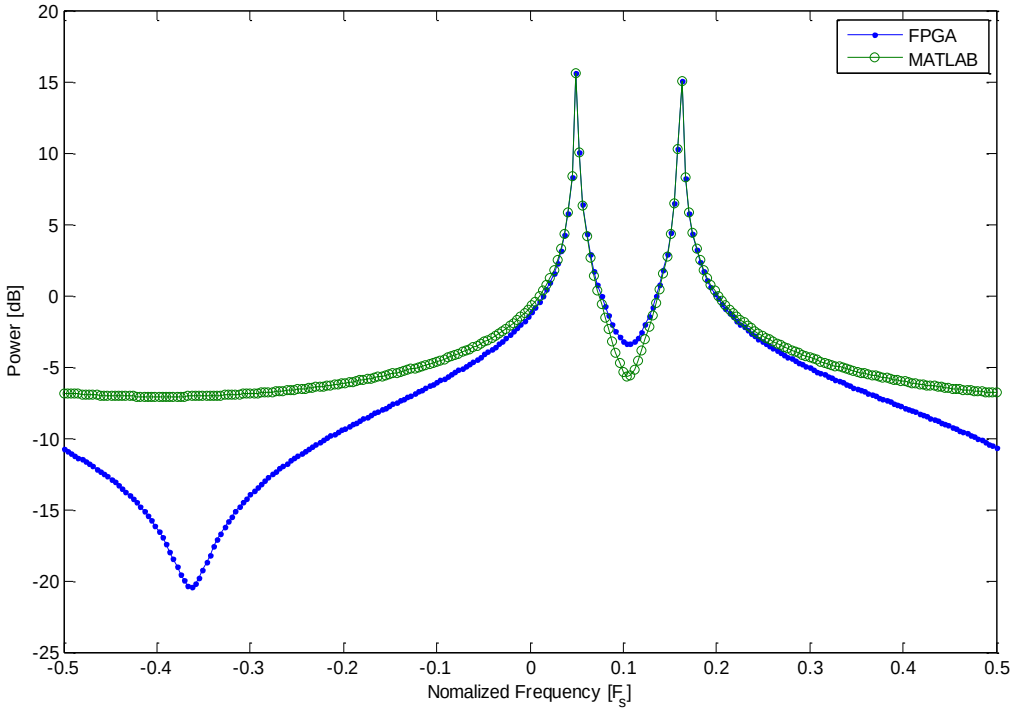


Figure 32: 256 point FFT, 2 freq. complex exp.

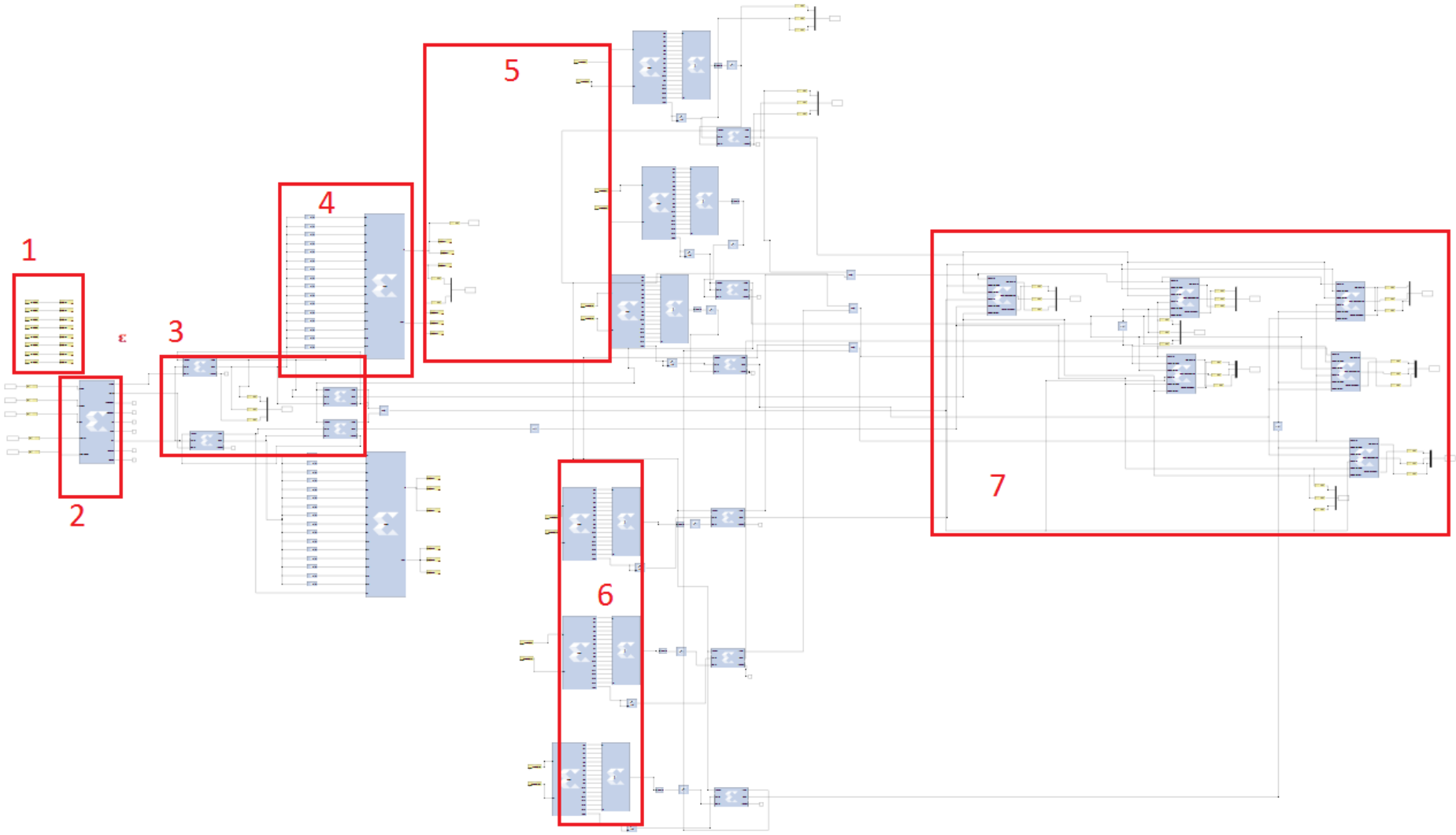


Figure 33: 1-GPIO switches connected to LEDS (verification). 2-FFT. 3-FIFO Mem. 4-TDM. 5-Data transfer over PMOD pins. 6-Serial to parallel with FIFO mem.. 7-Complex conjugate multiplications