MULTI-ELEMENT CORRELATOR AND BEAMFORMER USING OPENCL ON FPGA ACCELERATOR CARD

REPORT SUBMITTED TO
SAVITRIBAI PHULE PUNE UNIVERSITY
FORMERLY KNOWN AS UNIVERSITY OF PUNE

By

RAGHUTTAM SHREEPADRAJ HOMBAL

FOR THE DEGREE OF
MASTER OF SCIENCE (M.Sc.)
IN
ELECTRONIC SCIENCE

Under the Guidance of

Dr. Pranoti Bansode
Depratment of Electronics & Instrumentation Science

Mrs. Mekhala Muley
Giant Metrewave Radio Telescope,
NCRA - TIFR

Department of Electronics & Instrumentation Science
Savitribai Phule Pune University
Pune - 411 007
June, 2022

CERTIFICATE

Certified that the project work entitled "Multi-element Correlator and Beamformer using OpenCL on FPGA accelerator card" is work done by Mr. Raghuttam Hombal at Giant Metrewave Radio Telescope (GMRT), NCRA-TIFR in the partial fulfillment of requirements for award of M. Sc (Electronic Science) degree of Savitribai Phule Pune University during the year 2021-2022. The work has not formed the basis for the award of any other degree, diploma, in this or any other University or other institution of higher learning.

Head,
Department of Electronics
& Instrumentation Science,
S. P. Pune University

Mrs. Mekhala Muley,
Guide, Engineer E,
Giant Metrewave Radio Telescope,
NCRA - TIFR

Preface

As a part of my Curriculum Credit Course and also to gain more knowledge about the field of High Performance Computing and Astronomical Instrumentation, I present you this project report. The project is developed as a part of Student Trainee Program at Giant Metrewave Radio Telescope (GMRT), NCRA-TIFR which is one of the world's largest operational Radio Telescope.

Radio Interferometry refers to the process of combining signals from multiple antenna to form an image of the radio source in the sky. Radio-astronomical imaging is computationally challenging and poses strict performance and energy-efficiency requirement. The aim of this project is to design an energy-efficient multi-element correlator and beamformer on FPGA using High-Performance Computing. The digital signal processing for radio astronomy involves compute intensive operations such as FFT, correlation, beamformation, filtering etc. which can be implemented using parallel processing techniques. In this project, Open Computing Language (OpenCL) is used for designing the digital signal processing chain on the FPGA. We expect it to be high-performing and energy-efficient with rapid development cycle.

The project helped me in studying Digital Signal Processing with an approach that is more application-oriented. Through this project, I could obtain more insight on how the real-time systems work and how is the data flow designed and optimized to perform compute intensive tasks.

Acknowledgement

I would like to thank, my guide Mrs. Mekhala Muley, for the constant support, guidance and patience. I have benefited greatly from your knowledge and wisdom. I am extremely thankful for everything that I have gained from you during the course of this project. I would also like to thank Prof. Yashwanth Gupta (NCRA), Sri. Ajith Kumar (GMRT) and Dr. Jayanta Roy (NCRA) for providing me this opportunity to work on this project in such a prestigious institution. Thanks to Mr. Harshavardhan Reddy (GMRT), Mr. Sanjay Kudale (NCRA) & Mr. Kaushal Buch (GMRT) for their help in conducting multiple experiments. Special thanks to all the staff of Giant Metrewave Radio Telescope for their constant coordination and cooperation towards the whole project and make this project possible.

Thank you Dr. Pranoti Bansode for the support that you provided, remotely, during the course of the project. Special thanks to Prof. D. C. Gharpure for constantly supporting us in every way possible and helping us push our limits to the extreme. I would also like to mention the staff of Department of Electronic Science & Instrumentation Science for their patience and guidance.

List of Figures

Contents

Chapter 1

Introduction to Interferometry

1.1 An overview of GMRT

The Giant Metrewave Radio Telescope (**GMRT**)[3] is an array of 30 antenna, each being 45 m in diameter designed for study of Radio Astronomy. The Observatory has 14 antenna distributed randomly in a compact region of radius approximately 1 km, called the **Central Square**. Rest of the antennas are distributed in a roughly Y-shaped pattern with each arm being approximately 14 km in length as shown in Fig1.1. The central square provides shorter baselines, which is useful for imaging a large extended sources whose visibility is focused at the origin of the U-V Plane. The arms are very useful in imaging a smaller sources, where high angular resolution is essential, as they provide extended baseline. A single GMRT observation hence yields information on a variety of angular scales.[3]

The GMRT currently operates at 5 different frequency bands distributed within the range of 50 MHz to 1400 MHz. The feeds work at the ambient temperatures only. Band selection and preliminary amplification of RF signal is done in front-end system. After the first RF amplifier stage i.e. Low Noise Amplifier Stage (LNA), signals are fed to a common second stage amplifier.The RF signal is then brought to antenna base for further processing. Signal is then fed to lase-diode for converting them into optical signals and are transmitted to Central Electronics Building (CEB) by fiber optics cable for further digital signal processing. GMRT features two backends running simultaneously viz. :

- **Legacy system** - processing 32 MHz bandwidth using GMRT Software Backend (GSB)
- **uGMRT system** - processing 400MHz bandwidth using GMRT Wideband Backend (GWB)

FX correlator is used for interferometry mode, supporting both total intensity and full polar mode processing. A range of observing bandwidths are available with varied resolution.The correlator output / visibilities are stored in a format called LTA format which are further converted into FITS format. In parallel with the correlator, the GWB also has incoherent array (IA) and phased array (PA) beam-formers for the array mode, which are useful for observations of sources such as pulsars and fast transients.

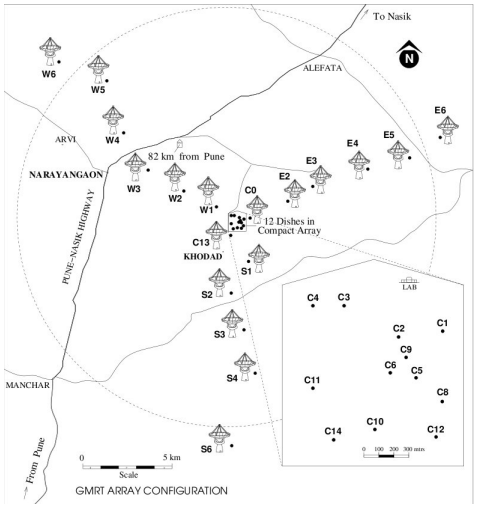


Figure 1.1: GMRT Array Configuration[3]

1.2 Radio Interferometry

The angular resolution of telescope i.e. the smallest angle between close objects that can be seen clearly to be separate, is given by Rayleigh's Criterion,

$$\theta \approx \frac{\lambda}{D} \tag{1.1}$$

where D is the size of Aperture, which implies that, for higher angular resolution, the Diameter of the instrument should be much larger. In order to achieve higher resolution in RF - Domain (metrewave lengths) it is not practically possible to increase the diameter of the telescope after a point (which would be in order of 10^5 m). This is why one uses a technique known as Interferometry. Interferometry is a technique that allows one to use an array of smaller antenna to observe the sky instead of a very large one (which is not practical to be built or used).

The physics of this imaging technique is built on the **Van Cittert - Zernike Theorem**. This theorem relates the spatial coherence function to the distribution of intensities of the incoming radiation. Ultimately, the Visibility function over an U-V plane is given by the equation[3]

$$\nu(u,v,w) = \int I(l,m) \exp^{-\imath 2\pi[lu+mv+nw]} \frac{dldm}{\sqrt{1-l^2-m^2}} \tag{1.2}$$

This is the fundamental relationship given by Van Cittert - Zernike Theorem.

This equation resembles Fourier Transform for a large extent when:

- The observations are confined to U-V plane, i.e. $w = 0$.
- The source brightness distribution is limited to a small region of sky.

1.2.1 Aperture Synthesis

The visibility function of a 2D plane is just a simple Fourier Transform of the source brightness distribution given in Equ 1.2, hence, a single source brightness distribution

can be derived by Fourier Transform of correlated voltages from multiple antenna. After many measurements, Inverse Fourier Transform of correlated data can be obtained to obtain what is referred to as a map. Since the Radio sky does not vary much with respect to time, one can measure all required Fourier Components at different instances by moving the telescope over an area to get more resolution. Radio Telescope Arrays take advantage of rotation of the earth, which apparently displaces the antenna after a while to a different coordinate in place (in km). This method of gradually building up all the required Fourier components and using them to image the source is called "aperture synthesis".This allows us to synthesis the data as if it is from a single Antenna.

1.2.2 Correlator

The mutual coherence function is measured by cross-correlating the voltages from each pair of antennas. The measured cross-correlation function is also called visibility. In general it is required to measure the visibility for different frequencies (spectral visibility) to get spectral information for the astronomical source.[3] This visibility data is used in imaging, continuum and many other astronomical observations. For an array of N antennas, at any given points one has to measure ${}^{N}C_2$ number of Fourier Components (samples on U-V Plane) which means that one needs to measure ${}^{N}C_2$ correlations at an instant of time. This operation is performed by what is known as a Correlator.

1.2.3 Beamformer

Pulsars are the weak radio sources, so their individual pulses often do not rise above the background noise level. Pulsar is a very highly magnetized rotating neutron star, which emits pulses with a very high precision period i.e small duty cycle.

Beamforming is the basic technique used for their studies. It is used in sensor arrays for directional signal transmission or reception. This is achieved by combining elements in the array in such a way that signals at particular angles experience constructive interference while others experience destructive interference. In beamformer, the antenna signals can be added coherently or incoherently. So for Pulsar observation, array mode with higher time resolution is used.

Chapter 2

Heterogenous Computing

In this chapter, we are going to see why do we need Heterogeneous Computing systems, What are the ways that one can achieve it and where does this finally take us. We shall see what are commonly used devices for this and why should we choose that particular device if we do so. What are the standards that let us achieve Heterogeneous Computing and how do we go about it.

2.1 Need for Heterogeneous Computing

Even though trend of Moore's law [10] is no more holding good, the need for faster processing has not depreciated, instead with advancements in software and complex systems it has increased a lot. Traditional CPUs can no longer perform the tasks as fast as we want them too. This has lead to development of **Application Specific Integrated Circuits** or ASICs. These systems are focused one performing few particular kinds of tasks with super-speed but lack general purpose task handling capabilities. **Graphics Processing Units** (GPUs) and **Field Programmable Gate Arrays** (FPGAs) and very commonly used types. These systems are particularly designed to achieve parallelism in order to complete the task as soon as possible but they are not capable of running independently.

During this era of internet, many tasks needs to be completed with almost real-time latency in order to keep up with the never ending ocean of tasks and queries coming in. Even many leading Cloud Service Providers offer options to run the servers using many such Heterogeneous Computing Systems that are specially designed to provide high throughputs for complex operations. This keeps the traffic smooth rather than using CPUs for compute intensive tasks and queue the rest. Computation can be done hundred folds faster when such systems are used efficiently.

2.2 Compute Platforms

As mentioned earlier, There are four major computing platforms - ASICs, CPUs, GPUs and FPGA. CPU or **Central Processing Unit** is mainly dedicated for General Purpose Computing. It is good with file handling, task management, mathematical operations, interfacing I/O to the system. It is flexible but with the cost of losing performance. The architecture of the CPU is designed in such a way that it should be able to do basically any task, but time is not a serious constraint here.

In recent times, GPU or **Graphical Processing Unit** emerged as a system that could allow user to crunch the numbers as fast as possible. With a number of cores, It could achieve parallelism in a much broader perspective than a CPU could ever achieve. Having multiple cores to work on, It could run a single instruction on multiple data within no time. This formed the basis of Single Instruction/Multiple Data or generally known as **SIMD**. In earlier days, even though they were not created for scientific computations, they had huge number of cores that were optimized only for integer or floating point arithmetic. They communicated with CPU via interconnect, traditionally using PCIe bus. Even though they are super-fast for transfers, they created a overhead problem, as transferring data over links itself could be much larger than gain in computation time. Even though there are solutions for this problem such as splitting the transfer or adding more number of GPUs, they are not a general ideal solution, especially when one wants a Real-time performance. The hard wired architecture of GPUs also restrict many functional possibilities that could have been inculcated in the design.

Field Programmable Gate Array or FPGA have gained popularity over a period of time in this domain. They have blocks of Gates, Flip-flops, DSPs built-in which can be used to carry out the job. Conventionally they are configured through Register-Transfer Level (RTL) descriptions using VHDL or Verilog. Synthesis tools (most of the times vendor specific) converts these HDL codes to RTL and then to bit file which configures the FPGA with the specific design. Nowadays, with advancements in technology, there are standards available through which we can program FPGAs using high-level language such as C/C++. This makes FPGAs more accessible to high-level programmers and provides smoother development environment.

2.3 OpenCL

OpenCL (Open Computing Language) is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs, FPGAs and other processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms.[9] It is actually designed to support a wide range of processing units for High Performance Computing solutions. This standard is derived from ISO C99 with extensions and also defines numerical requirements specified by IEEE 754. OpenCL on a much broader view, gives control for the programmer achieve parallelism in the form that he/she prefers having. May it be coarse-grained or fine-grained such as that for a single instruction.

2.3.1 Architecture

OpenCL API includes libraries that can be used to write portable yet efficient codes to obtain High performance from the machine. There are two basic units for a OpenCL Platform model.

Host: This can be any General Purpose compute unit such as a CPU. Even ARM devices can be used for this purpose. The primary job of the host is to identify devices, create context, program the attached device, dispatch and read the data from the device. A source code is usually written in C/C++ to generate an executable to perform this action.

Devices: These can be GPUs, FPGAs or maybe another CPU too. They are programmed during the execution, with a source file written using OpenCL language (with extension **.cl**) or through a generated binary file for vendor specific devices such as that of Intel FPGA.

The application that runs on the host is responsible for submitting commands from host to device in order to execute instructions/programs on work items (or compute units). A single host is capable of handling multiple Devices, they may not be of similar type too. It is discretion of the programmer to make the best use of the available resource.

2.3.2 Programming Models

There are two forms of programming models offered by OpenCL standard. Based on the nature of the task and requirement of the programmer, either of them or a combination of them can be implemented.

NDRange Programming Model

This model divides the task into multiple work items. For instance, if the task is to add two vectors of 100 elements each, then this model is capable of replicating an *Adder* (Basic unit of a Vector addition) into many work items and can have 100 operations in a single clock. This is a form of **SIMD** implementation, wherein a single instruction has to be executed on multiple sets of data. Note that *Every work-item is independent of each other and cannot communicate in whatsoever way.* It is possible to create a pipeline of such an instruction, and a initiation interval can be started for each work-item [1], but by doing this, we tend to achieve a coarse-grain parallelism which requires more logic and memory. Alternative option is to replicate more number of Compute Units and let assigning each Compute Unit the same task. Even though it increases the usage of arithmetic resource, It gives more control over the task.

Single Work-Item

In this form, an entire operation is performed by only a single work-item. High Performance is obtained by unrolling and pipelining the code wherever possible, such as Loops. Instead of executing a single instruction in sequential fashion, the block that is previous to the current stage can start the job with the next data. This reduces the time required to compute a set to data provided the data must be purely independent of each other. OpenCL can even merge multiple instructions, for instance, if there are 2 **Multiply and ACculmulate** operations adjecent to each other, then they can be combined into a single instruction to reduce the latency.

Not always can a task be divided into several units independent of each other. There has to be a combination of Pipelining and Sequencing of the instruction over a range. It is also important to know that, every work-item is independent of each other, which means that there is no direct way of synchronization of tasks. Synchronization is done by using some kinds of Events to indicate the status of the job, Profiling of a task or through a command, wherein it can force the work-items to delay until another event's occurrence takes place[13]. There can be Host notifications and callback functions to indicate the execution of a task. Wait-lists and command events are another way of achieving some form of synchronization among the Tasks enqueued. With events one can establish the

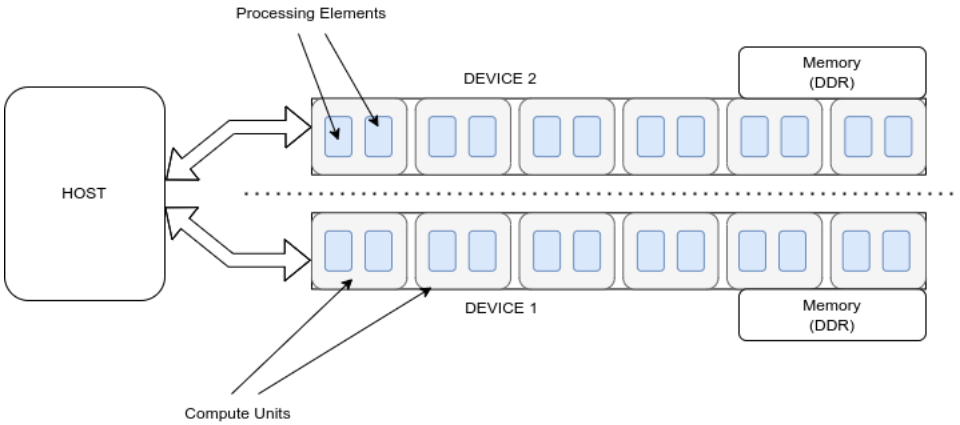


Figure 2.1: Represents OpenCL Architecture and Execution Model

order for the execution of commands. That is to force one or more commands to wait until a set of events have completed. These sets of delaying events are called wait lists.[13]

2.3.3 Execution and Memory Model

Even though we tend to focus on FPGAs here, Execution model for any other device is quite similar to this. There are two elements needed for OpenCL to be executed. The **Kernel Program**, which is programmed onto the device and the **Host Program** which is compiled by the native C/C++ compiler and executed locally. The Host is responsible for creation of required resource entities, allocation of any kind of Memory Units, Copying of the Buffers, Queuing the tasks for Execution, Reading back the Data from the Device, Managing Contexts, etcetera. All memory operations are explicit. Multiple levels of Memory are available. The user is responsible of making the best use of the memory types available. There is a Global Memory situated on some form of DDR memory that is available when we use FPGA or GPU, in order to eliminate the overhead of transferring over the Host-Device interconnect for data each time. There is also Local memory and Private memory based on the hierarchy of the stage.

A **Work-item** is an elementary unit of computing. It is accompanied by a **Private Memory** for faster data retrieval. There can be multiple Work-items doing the same job simultaneously which results in parallelism. Every work-item is independent of each other and have no synchronization whatsoever until explicitly told to synchronize. A group of work-items is called a work-group, which is native to the Device. Usually every work-item under a work-group performs same task or executes the same set of instruction. A Work-group also has a dedicated memory known as **Local Memory**. This is a kind of shared memory among the work-items of that group. This acts as a L2-cache for the work-items, much bigger but at the cost of latency.

Multiple Work-groups can be created in a device based on its capacity. A single or multiple devices can form what is called to be a Context. The devices selected work together, share a common memory known as **Global Memory**. Contexts make is possible to create **Command Queues**, the structures that allow hosts to send kernels to devices, and enqueue the tasks . [13] A Global Memory is like a pool of data that needs to be worked on. Every work-item fetches data from the Global Memory and performs operations as instructed. Host is capable of communicating to the Global Memory only. It is to be noted that, there can be multiple contexts on a platform using many devices,

but there cannot be any context using multiple devices from different platforms.

2.4 Intel FPGA SDK

Intel FPGA SDK is a work environment for designing digital circuits for Intel's FPGA using OpenCL in a 2-step process[6]. There is a OpenCL Offline Compiler that is responsible for compiling the OpenCL kernels and generate a bit file and there is a host-side regular GCC Compiler that compiles the host program written in C/C++ and links it to the OpenCL Kernels. There are three main parts of the programming model.

- A host application and Compiler
- The OpenCL Kernel and offline Compiler
- Custom Platform

Custom Platform comprises of OpenCL Reference Platform Design i.e. details about the target platform and Board Support Packages. It gives information about the device to the Offline compiler so that, the compiler can generate an hardware image of the target using the IP blocks. A series of files are generated by the Intel's AOC [13] Compiler that contains the HDL code, synthesis report, details about the timing constraints, etc. It takes time for the compiler to generate a bit file that programs the actual hardware. This might even be in hours, this makes it harder for the developer to test and debug the behavioural errors of the system. Instead, there is an **Emulator Mode** that can be used to analyse the functionality of the code. It takes very less time to get the job done.

The host application is then responsible for management of the process. It is supposed to read the bit file, use it to program the FPGA, Create buffers and communicate to target device. *It is to be noted that for FPGAs, onlny way to program them is by using the Binary file, unlike GPUs and CPUs where in we can program them using the* ***.cl*** *file too.*

2.4.1 Board Support Package

Board Support Package (BSP) contains file sets for generating OpenCL executables for running on the FPGA accelerator card. The BSP package also includes the required drivers and software libraries for creating and running the associated host executables. It supports communication of the target device with the Host, DDR memory, Networking ports and many more. In addition to the memory blocks and PCIe lanes, any HPC application would require ways to transport data efficiently to/from the target device using IO interface. This is taken care by the BSPs. There are a number of such packages specifically built for some purpose with some added advantages and usually the vendor of the board provides one. Even the on-chip memory of the FPGA, that is allocated as the local memory for the kernels, needs an interconnect to communicate and the BSP takes care of that. It does take up some amount of static on-board memory but is necessary for the process.

Chapter 3

Signal Processing

The main objective of the project is to implement **Multi-element Correlator and Beamformer** on FPGA using OpenCL. The word *Correlation* itself suggests that it is a measure of match/resemblance of two entities. In digital signal processing, It deals with the concept of how well a signal resembles another signal or itself. Computing the correlation between the two signals is to measure the degree to which the two signals are similar and thus to extract some information that depends to a large extent on the application. [8] There are two types of correlations -

Auto-Correlation This type of correlation involves only one signal. It is performing the operation on itself. This kind of operation is used to find out the faint signal that is buried under the noise.

Cross-Correlation This involves two signals. The operation is performed to find out the common frequency components between both of them, or to calculate the phase delay between two signals.

3.1 Cross-Correlation

As mentioned above, This type of Correlation involves more than one signal. Mathematically it can be represented as

$$\begin{aligned} r_{xy}(l) &= \sum_{n=-\infty}^{\infty} x(n+l)y(n) \\ &\qquad\qquad l = 0, \pm 1, \pm 2, \pm 3, \\ &= \sum_{n=-\infty}^{\infty} x(n)y(n-l), \end{aligned} \tag{3.1}$$

It is to be noted that,

$$r_{xy}(l) = r_{yx}(-l) \tag{3.2}$$

which implies that $r_{xy}(l)$ is simple the folded version of $r_{yx}(l)$, with respect to $l = 0$, but, they provide exactly the same information with respect to the similarity between $x(n)$ to $y(n)$. [8]

Similarity with Convolution

The operations Correlations of two signals and convolution of two signals are very similar to each other. In Convolution, one the signal is folded, time-shifted, multiplied and

summed up at every points. Correlation also does the same thing except for the folding of the signal about a point. Hence, Correlation operation is convolution of two signals with one of the folded about a point [8]. If $*$ is the operation for convolution, then,

$$r_{xy}(l) = x(l) * y(-l) \tag{3.3}$$

This will be helpful as we proceed further

3.2 Fourier Transform

Fourier Transform is one of the powerful tools of Engineering Mathematics that revolutionized the stream of Signal Processing. In simpler words, Fourier Transform **FT** converts a signal from Time-domain to its equivalent form in Frequency-domain. This gives the user capabilities of Filtering a signal, Compressing the Data, Convolving Signals faster and much more. Just like a prism that splits a white light into multiple components, Fourier Transform splits a signal into its different frequency components. The output of the Fourier transform is in format of a complex number. For Analog signals, FT is given by

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi\omega t} \tag{3.4}$$

For Discrete Signals that are causal in nature, it becomes,

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi k\frac{n}{N}} \tag{3.5}$$

This form is known as Discrete Fourier Transform **(DFT)**. More is the number of points or N, more will be the number of the distinct frequency components found. The process of computation requires atleast N^2 number of complex multipliers, even if we pre-calculate and form a Look-Up Table **(LUT)** for the twiddle factor coefficients. This was very time consuming as not at all feasible for applications that required FT to be computed immediately. In 1965, J. W. Cooley and John Tukey came up with an algorithm that would be a faster implementation of FT, popularly known as the **Fast Fourier Transform** or **FFT**.[5]

FFT uses 'Divide and Merge' theory to operate on the given data, and reduces the number of computations to $N\log(N)$. It is interesting to note that as the Number of points i.e. N increases, the effect of $\log(N)$ reduces and it becomes almost linear. Modern systems make use of FFT to a very large extent due to its reduced complexity, faster calculation and expandability. The working of FFT is shown in the Fig.3.1. It is just a FFT of 8-points but can be scaled up in similar fashion. It is very important to note that the Fourier transformation function is linear in nature, i.e. if F is function representing Fourier Transform, and $x(n)$ and $X(n)$ be the signal and FT of it, respectively, then,

$$\begin{aligned} if, x(n) &= \alpha_1 x_1(n) + \alpha_2 x_2(n) \\ then, X(n) &= F(x(n)) \\ &= F(\alpha_1 x_1(n)) + F(\alpha_2 x_2(n)) \\ &= \alpha_1 X_1(n) + \alpha_2 X_2(n) \end{aligned} \tag{3.6}$$

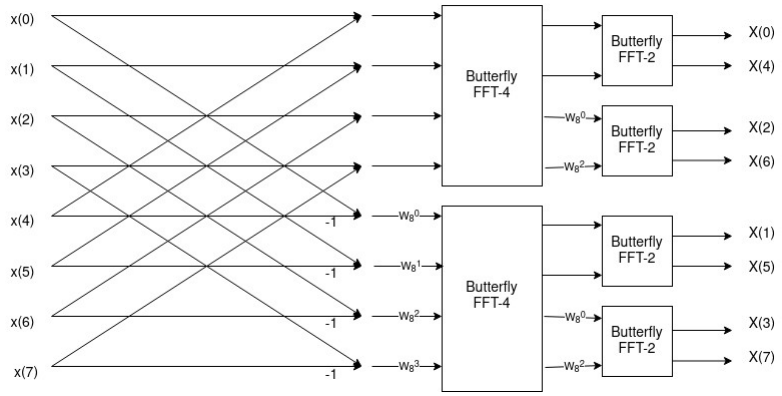


Figure 3.1: Butterfly Diagram of 8-point FFT

Advantage of Fourier Transform

One of the main advantages of converting any time-domain signal into fourier-domain is that, operations such as convolution becomes very simple. A convolution operation in Time-domain is just element-wise multiplication of Fourier-domain equivalents of those signals. if,

$$\begin{aligned} y(n) &= x(n) * h(n) \\ &= \sum_{n=-\infty}^{\infty} x(l-n)h(n) \end{aligned}$$

but if $Y(k)$, $X(k)$ and $H(k)$ are FT of $y(n)$, $x(n)$ and $h(n)$ respectively, then

$$Y(k) = X(k) \cdot H(k) \tag{3.7}$$

Since Convolution and Correlation are very similar to each other, we can extend the same to infer that, if R_{xy} is FT of r_{xy} by (Equ3.3)

Another feature of Fourier Transform is its Time-Shifting Property. A signal $x(n)$ can be time shifted in Fourier Domain by multiplying it with a factor. This can be used to make Fine Delay adjustments and Fringe Correction.

$$X(N - N_0) = X(N) \cdot e^{\frac{-N_0}{N}} \tag{3.8}$$

This property is used to delay the signal in fraction of samples and also for fringe stop. More about this is explained in sections below.

$$\begin{aligned} R_{xy}(n) &= FT\left[r_{xy}(n)\right] \\ &= FT\left[x(n) * y(-n)\right] \\ &= FT\left[x(n)\right] \cdot \left[y(-n)\right] \\ &= X(k) \cdot Y^{\star}(k) \end{aligned} \tag{3.9}$$

3.3 Correlators

A radio interferometer measures the mutual coherence function of the electric field due to a given source brightness distribution in the sky. The antennas of the interferometer

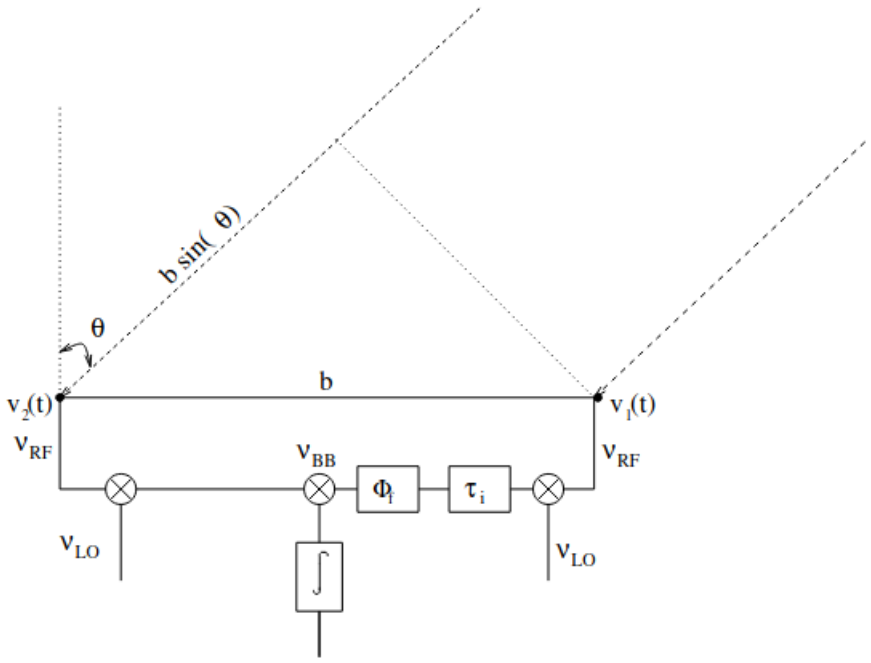


Figure 3.2: A two element interferometer with fringe stopping and delay tracking[3]

convert the electric field into voltages. The mutual coherence function is measured by cross correlating the voltages from each pair of antennas. Refer to the Figure 3.2. Since the radio waves from the source have to travel an extra distance $b \cdot sin(\theta)$ to reach antenna 2, the voltage there is delayed by the amount $b \cdot sin(\frac{\theta}{c})$. This is called the geometric delay. Also θ is function of angular frequency of earth's rotation causing the output to vary in quasi-sinusoidal form (called as fringes). Thus delay (geometric and instrumental) and fringe correction is done. So basically a correlator does the delay correction, FT, fine delay correction & fringe stop, multiplication and integration of the two signals.

3.4 Types of Correlators

As discussed before, There are two kinds of Correlators.[3] **FX** and **XF**. There is not much of a major difference between both of them but, in **FX** Correlator, FFT is performed on the signals before passing them through Multipliers, whereas in a **XF** Correlator a series of Complex Multipliers and Accumulators does their job before the FFT of the signal is obtained. Use of a particular method among the above mentioned is purely a choice of the programmer, but, GMRT uses FX Correlators for their operations[3]

The Block Diagrams show in Fig.3.3 and Fig.3.4 are of the entire system. For now, we are concerned only with a fewer blocks that include FFT and **M**ultiply and **AC**cumulate (MAC).

3.5 Beamformer

Beamforming is a signal processing technique used in sensor arrays for directional signal transmission or reception.In beamformer, the antennae signals can be added coherently

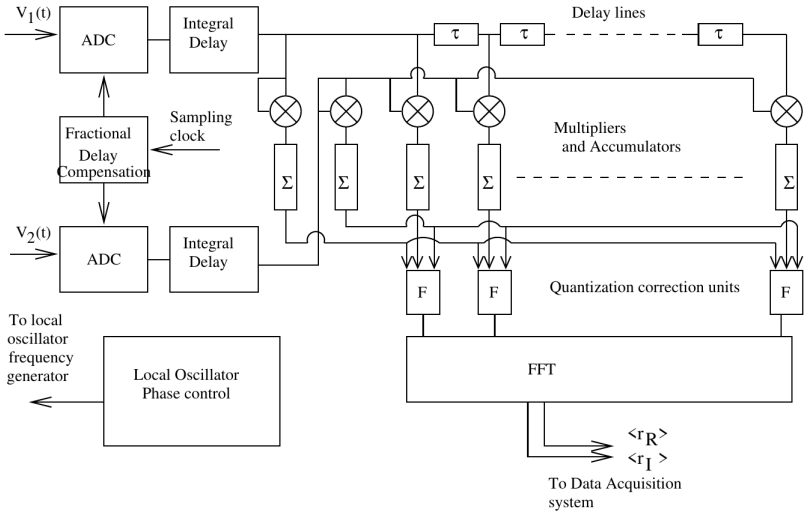


Figure 3.3: Block Diagram of XF Correlator[3]

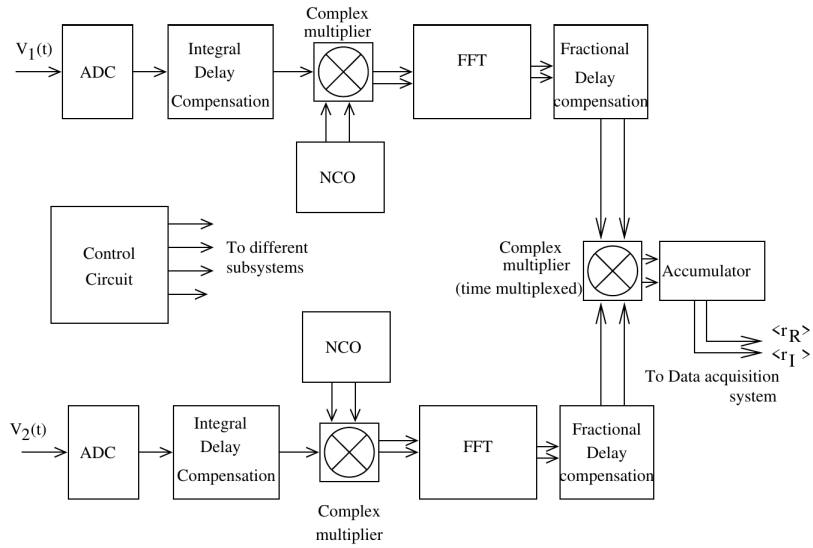


Figure 3.4: Block Diagram of FX Correlator[3]

or incoherently.

3.5.1 IA Beamformer

Incoherent Array (IA) beamformation and Phased Array (PA) beamformation are two most commonly used techniques for Antenna Arrays and are the ones used in GMRT back-ends.[12] The Incoherent Array beam is formed by summation of squared values of voltages by every antenna. Therefore, IA beam signal is given by,

$$P_{IA} = \sum_{i=0}^{N-1} |V_i|^2 \tag{3.10}$$

where V_i are the voltages from each antenna present in the array.

Advantages of IA Beamformer

- Improvement in sensitivity by $\sqrt{N}$
- Increased Beamwidth of single Antennae
- Application in large scale pulsar search and study of the source which are extended in size and covers a large fraction of the beam of the element pattern.

3.5.2 PA Beamformer

The Phased array (PA) or in other words, coherent Beam is formed by summing together of voltages that are phase corrected, and then squaring the resultant sum, i.e.

$$P_{PA} = \left[\sum_{i=0}^{N-1} V_i e^{-i\phi_i} \right]^2 \tag{3.11}$$

where, as we know, V_i are the voltages from each antenna present in the array and $e^{-i\phi_i}$ is the phase introduced in Antenna i.

Advantages of IA Beamformer

- Improvement in sensitivity by N
- Beamwidth becomes narrower than the single antennae by almost $\frac{1}{N}$ times
- Application in studies of individual known pulsars with its polarimetry studies.

The Incoherent beam is less sensitive than Phased Array Beam and also more vulnerable to gain fluctuations of the instruments and Radio-Frequency Interference.

Chapter 4

Implementation

A Correlator and Beamformer for 2 single polarized Antenna with baseband of bandwidth 100 MHz band *(can be varied)* is done. Long Term Accumulation (**LTA**) of correlator being 1.342s and Beamformer with Short Term Accumulation (**STA**) of 163.84μs has been implemented on a FPGA Accelerator Card using OpenCL for this design. The size of FFT can be varied and the designed can be compiled for range of 128 to 8k, but we have fixed our size for tests to 128 Point FFT and 4k point FFT for real-world signals. It is to be noted that we are doing this in offline mode, i.e. from pre-recorded Antenna raw voltages.

4.1 Specifications of FPGA Accelerator Card

The card that is being used for the project is 385A PCIe FPGA board that is powered by **Intel Arria 10 GX 1150** FPGA provided byb Nallatech limited. This is a low profile accelerator card equipped with powerful PCIe bus to enable fast data transfers between on-board Memory chips and FPGA itself. Having a 10G/40G I/O Platform makes development across a wide range of applications possible.

FPGA	Intel Arria 10 GX 1150
On-board Memory	2 Banks of 4GB SDRAM × 72-bit
Host Interface	x8 Gen3 interface to FPGA
Cooling	Single-width active heatsink
Operating Temperature	278K to 308K

The block diagram of Nallatech 385A is shown in the Figure 4.1 [7]. Some of the specifications of the Interal Arria 10 GX 1150 FPGA is mentioned below.

Resource	**Value**
Logic Elements	1,150
Adaptive Logic Modules	427,200
Registers	1,708,800
DSP Blocks	1,518
18 × 19 Multiplier	3,036
PLL (Fractional Synthesis)	32
PLL (I/O)	16
PCIe Hard IP Block	4
Hard Memory Controller	16

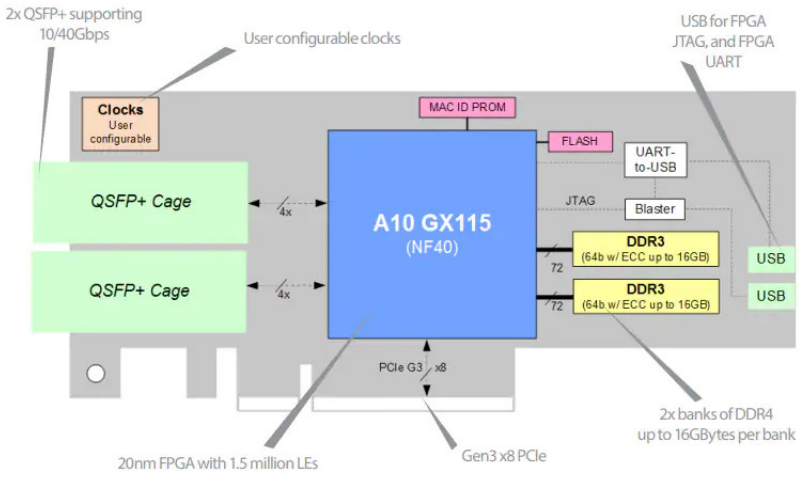


Figure 4.1: Block Diagram of Nallatech 385A

4.2 System Parameters

Some of the specifications of the system designed are given below.

Signals: Voltage values from 2 uni-polarized Antenna.

Input: Input format given to the system is a vector (here, structure) of two variables in **Character** format representing a real and imaginary part in a complex number. The format is decided to be 8-bit signed character as that is what we expect to obtain from the ADC used in the GWB *Note: Since we provide only real signals, we have the imaginary part equated to* 0 *all the time*

Output: Expected output format from the FPGA Accelerator card is also a vector of two variables in **Float** format representing a real and imaginary part respectively.

Other Parameters: The list of parameters that are needed to be mentioned during execution of program are listed below.

- **LOGN**: Logarithm value with base 2 of size of the FFT needed to be performed, like, for a 4K point FFT, LOGN should be 12. Acceptable Range: 6 - 14.
- **Iteration**: Number of FFT spectrum processed. Product of *Iteration* and *FFT Size* decides the number of points that will be processed in a batch, hence determining the integration time of the observation to be made. Acceptable Range: Multiples of LTA (here, 65536).
- **Inverse**: This is to inform the FFT Kernel if the operation needs to be a Forward FFT or Inverse. For Radio Interferometry, only Forward is used. Acceptable values: 0 (Forward FFT) or 1 (Inverse FFT)
- **Phasing parameters**: These parameters include Fractional Delay, Delay Rate, Fringe and Fringe Rate in radians. These four are followed by the Phasing values that needs to be compensated. All these paramters must be in *Radians* and Float-type.

4.3 Block Diagram and Signal Flow

The block diagram of the implemented system is shown in the Figure 4.2. Data from the Antenna is stored in the DDR Memory before we enqueue the task for the Device. Both the inputs are stored in different banks in order to speed up the process of memory transfer between Memory and FPGA. An array is of required size i.e. Number of Iterations $\times$ FFT size elements is allocated for each of the input signals and they are filled with the stored values. Coarse Delay compensation is done in the host itself. Once the task starts, 2 independent chains begin to process, with one input to each. They are fetched and processed in batches of 8 points and written on the channel respectively. Once a batch of 8 points is written on the channel, MAC and Beamformer kernel reads both the channels, and performs their function respectively and stores the output back to DDR Memory in order read it from the host.

The Multiply and Accumulate section, as the name suggestes, performs multiplication on the FFTs and goes on adding the correlated values on the previous one until the count of FFTs hits the LTA. Beamformer works in a little different way. The output are accumulated for a very small time (STA) and copied to the DDR memory. Next batch is not added to the previous one, but is concatenated with the previous output. This is why the size of correlator output is limited to the size of FFT being performed, but the size of Beamformer depends upon the integration time. More is the integration time, larger will the array of the Beamformer output. Signal flow of MAC and Beamformer section is shown in fig. 4.3

4.3.1 Fetch and FFT

These blocks are responsible for fetching the data in batches of 8 points in a single clock. *Fetch* is responsible for obtaining 8 points from the total data provided to the kernel. This particular kernel is a multiple work-item based kernel and the whole process is pipelined in order to fasten the process. Once a batch is fetched, it is written on to the respective channels. Since, we have multiple work items, burst filling of data takes place, which needs the channels to be of some depth, so that, none of the data is lost.

Followed by Fetch, the main *FFT* kernel is enqueued. This is made as a single work-item and the channels are read and FFT is performed. The output that we obtain from this process is in the bit-reversed form only. This is then written onto another set of channels which will continue the process further.

4.3.2 Fractional Delay Compensation and Fringe Stop

Correction is done by using Time-shifting property of Fourier Transform given in Equ 3.8.

$$A_{corrected}(N) = A(N) \cdot e^{\frac{\tau}{N} - \theta} \tag{4.1}$$

where,

$$\tau = Fractional\ Delay + (Time \cdot Delay\ Rate)$$

and

$$\theta = Fringe + (Time \cdot Fringe\ Rate)$$

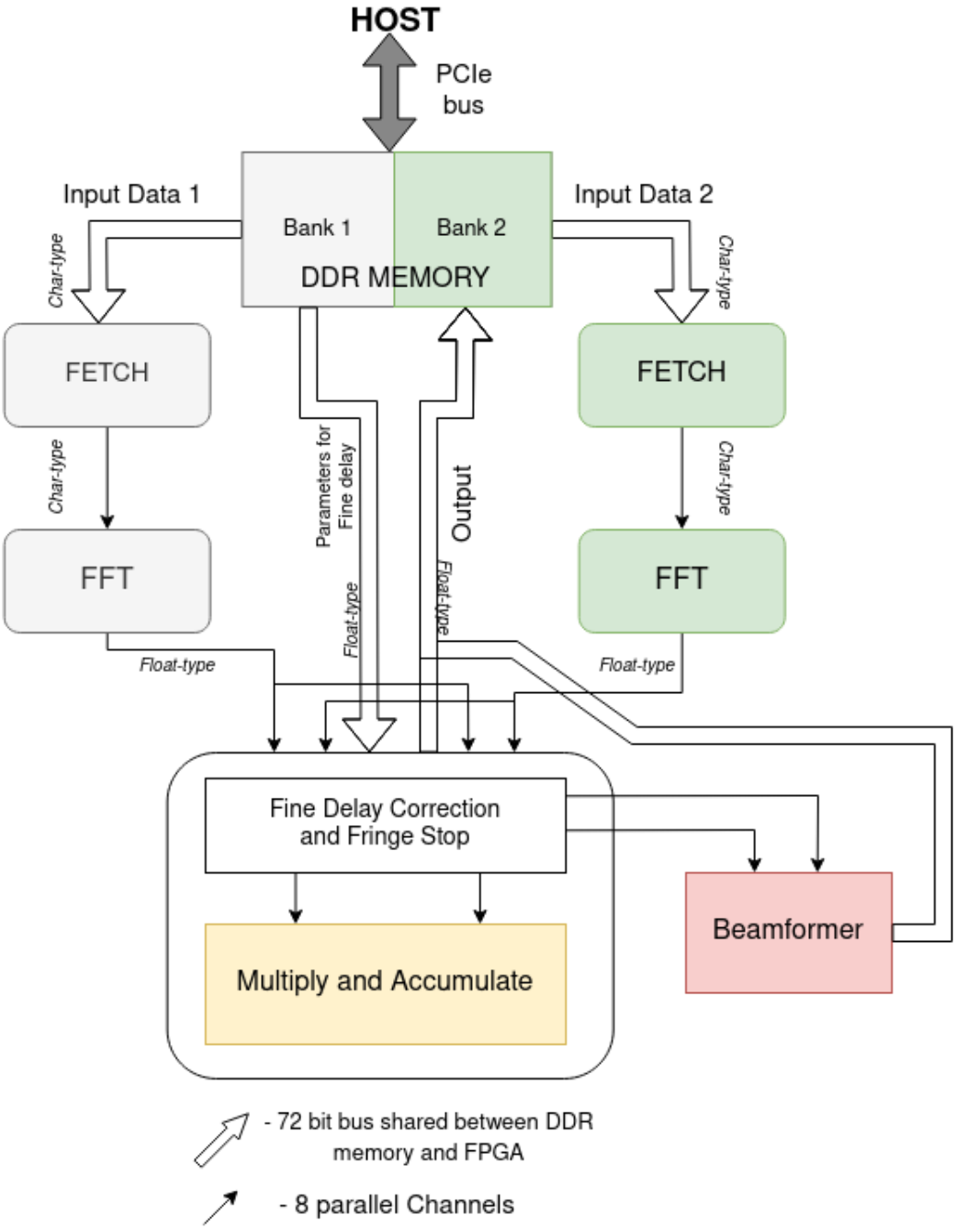


Figure 4.2: Block Diagram of the implemented system

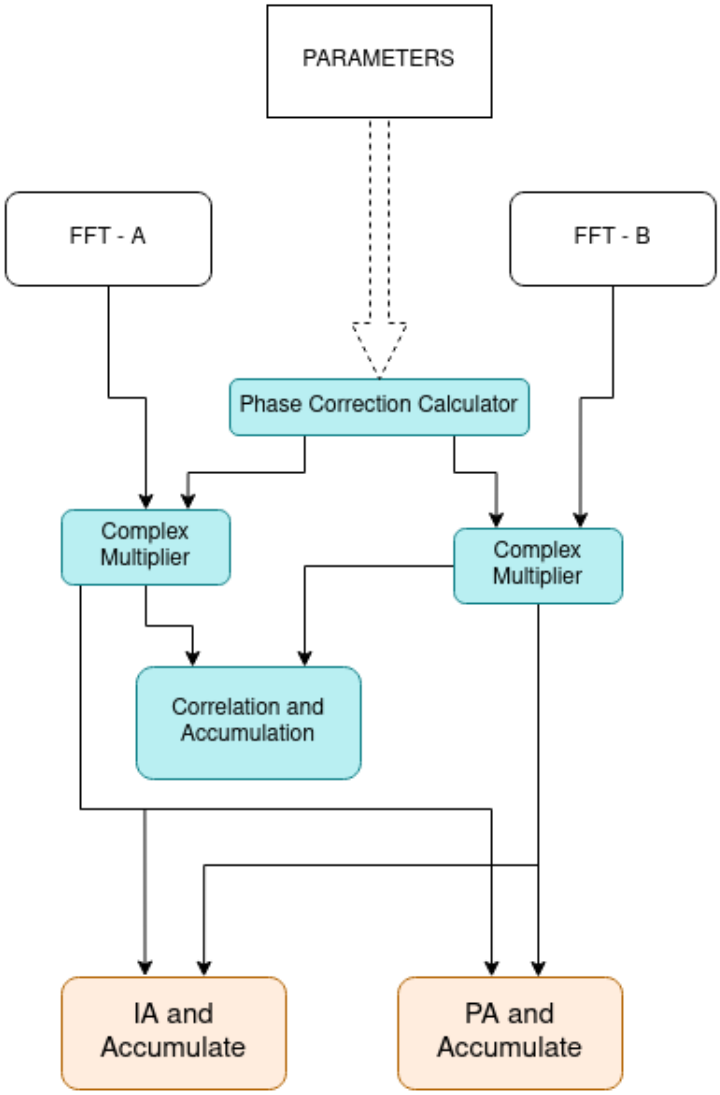


Figure 4.3: Signal flow of MAC and Beamformer section

Correlation operation is performed as shown in Equ.3.9.

Fractional Delay, Delay Rate, Fringe and Fringe Rate information is passed to the device from host during the execution as 2 buffers, one for each channel. With respective values of above mentioned parameters, τ (Delay) and θ (Fringe) are calculated and corrected with corresponding FFT Value. This operation is performed soon after the channels which contain FFT values are read. A single factor will be calculated which from these 4 inputs for each antenna and $cis()$ of this factor is multiplied to the corresponding sample.

If τ_0 (Fractional Delay), $\delta\tau$ (Delay Rate), θ_0 (Fringe) & $\delta\theta$ (Fringe Rate) be the values that we obtain for the respective antenna for corresponding set of data. Then,

$$\begin{aligned} \phi(N) &= \tau \cdot N - \theta \\ &= (\tau_0 + \delta\tau) \cdot N - (\theta_0 - \delta\theta) \end{aligned} \tag{4.2}$$

The quantity that will be multiplied to the sample that needs phase adjustment is given by $e^{i\phi}$.

4.3.3 Multiply and Accumulate

As the name suggests, this section is responsible for multiplying FFTs of 2 signals and produce Correlated output of the same. Now, even though the FFT values still belongs to the bit-reversed index, It would not change the result as every point in the FFT output is independent of each other. The correlated output is stored in a local array and then transferred to the DDR Memory so that Host could access the result. A local memory is used to accumulate because fetching the DDR memory every time would create a bigger overhead and could result in slower throughput. The extent of integration depends upon the number of iterations that are being processed, but finally the resultant would be an array of size same as FFT, with each point having two components - real and imaginary. This is passed on to the DDR Memory

4.3.4 Beamformer

There are two sections in Beamformers namely - IA and PA. In IA, the FFT values are squared and added with the corresponding index ones. These values are integrated for 8 FFT cycles and then concatenated with the previous ones. Integration happens in the local variable only, but after every 8 such integrations, the chunk is written on to the DDR Memory, and the pointer is incremented by number same as the size of FFT. Further the same process repeats.

PA Beamformer is also implemented in the same way, but, the phased signals in Fourier domain are added first and then the power is obtained by squaring them. STA is same here too (i.e. 8 FFT Cycles).

4.4 Resource Utilization

By the report generated by the *Intel Quartus Prime* software during the compilation of the code, the resources utilized by the design is shown in Table 4.1.

Component	Used	Available	Percentage used
ALUTs	253763	747080	33.96%
Flip-Flops	408940	1494160	27.36%
RAMs	1300	2367	54.92%
DSPs	1172	1446	81.05%

Table 4.1: This table shows the resource utilization of the design

4.5 Host Program

This program manages to create & destroy the resources, manage the data transfers, enabling the kernel to perform certain task, read/write the inputs/outputs and much more. This program is written in C++ language and bundles up all the libraries that are required to perform the set of desired jobs.

The Signal Flow inside Host program is shown in Fig. 4.4. It is to be noted that the LTA, STA, Iterations per cycle, Size of FFT and few such parameters are fixed and the Device program is also compiled for the same, but, number of batches of data to processed can be accordingly changed.

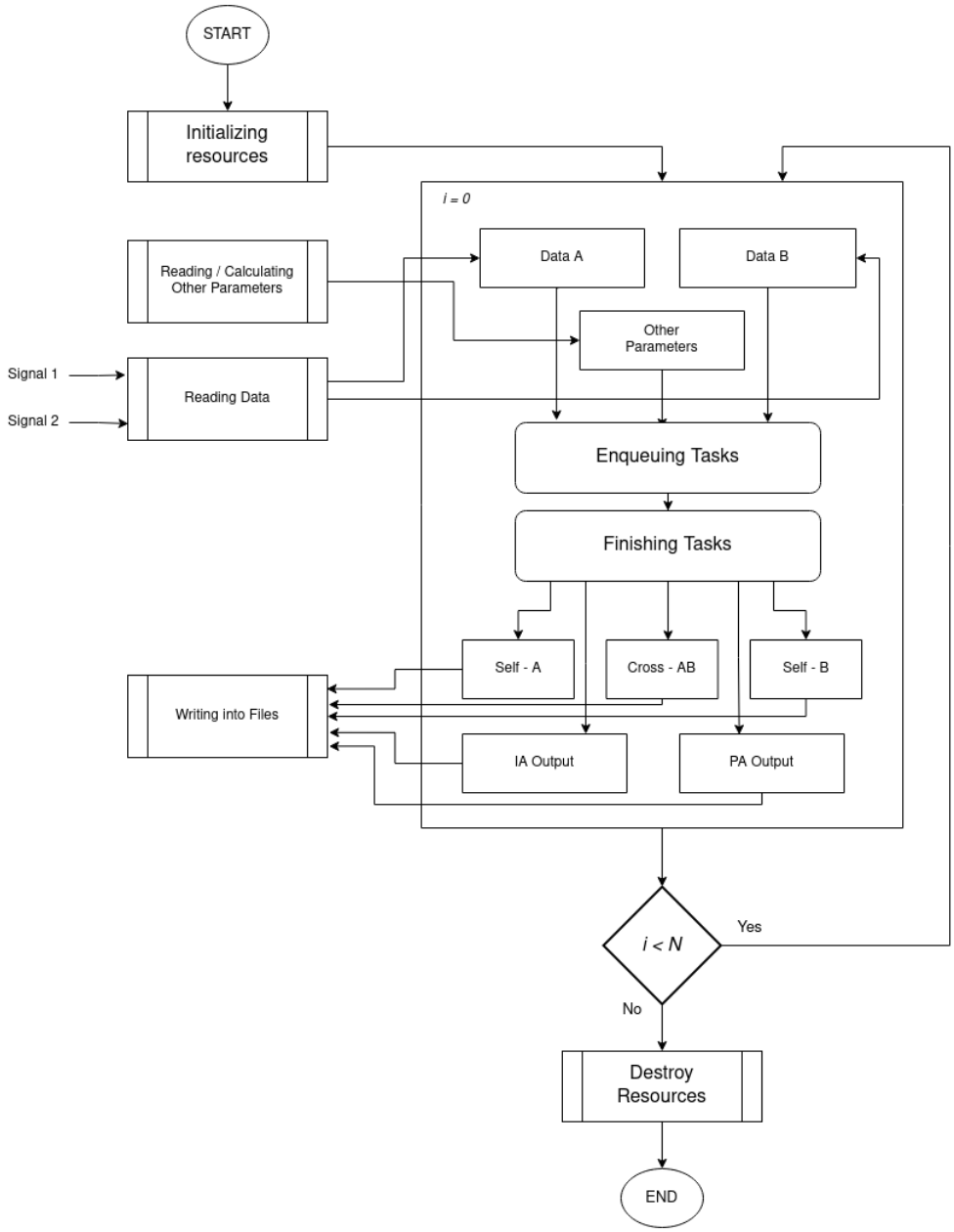


Figure 4.4: Signal flow chart of the Host Program

Chapter 5

Tests and Results

5.1 Two Single Tone Signals

Simplest way to verify the design was to run the design with 2 single tone signals. Since they had no noise, it was easy to recognise and verify the output of the Correlator and confirm its working. We are testing only FFT and correlation functionality here, without any delay & fringe corrections.

5.1.1 Inputs

Input to the system was 2 single tone signals separately at respective channels. Signals were sampled at Sampling frequency (**fS**) of 800 MHz. One of the signal was 50MHz and the other was 150 MHz. Both were generated from a code in C-language and stored in 2 different files with **.data** extension with 998,400 sample of floating point numbers present in each. With the FFT Size of 128, This number of samples was sufficient for the design to run for 7800 iterations. The Time-series plot of the signal is given in the figure 5.1.

5.1.2 Expectations

With the available amount of data and given the sampling frequency of the input signal, The processing that we can expect is

$$
\begin{aligned}
Total\ number\ of\ points &= 998,400 \\
Number\ of\ Points\ needed & \\
to\ be\ processed\ in\ 1\ second &= 800M \\
Throughput\ (Expected) &= \frac{Points\ to\ be\ processed}{Time\ to\ process} \\
&= 0.8\ Gpoints/sec \\
Time\ required\ to\ process\ our\ data &= \frac{Total\ number\ of\ points}{Throughput} \\
&= 1.248 \times 10^{-3} sec
\end{aligned}
$$

With the plots, we must observe a single spike at the desired frequency (Here, 150 Mhz and 50 MHz) for the magnitude values of self correlated power of signals 1 & 2, but no

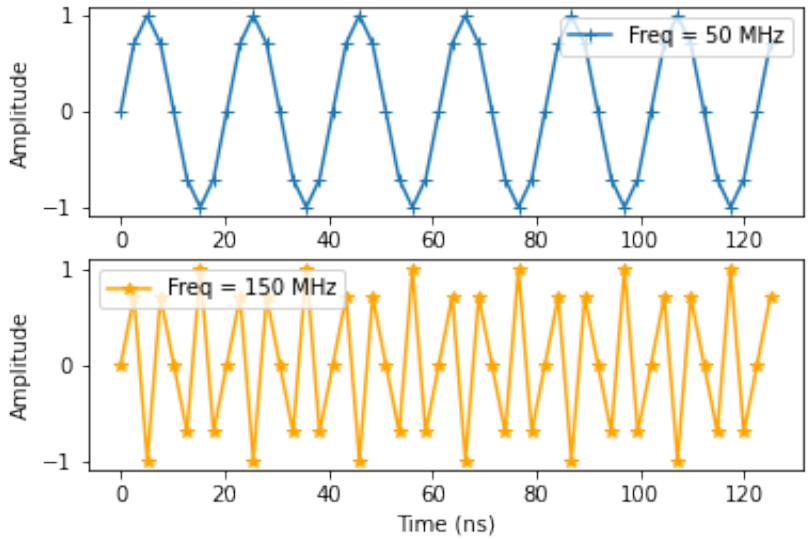


Figure 5.1: The plot shows the Amplitude of input signals w.r.t Time

power for the normalized correlated magnitude hence making it a null over the spectrum. There can be visible change in phase in the Cross-correlation but the phases of the self-correlated power of both signals.

5.1.3 Results

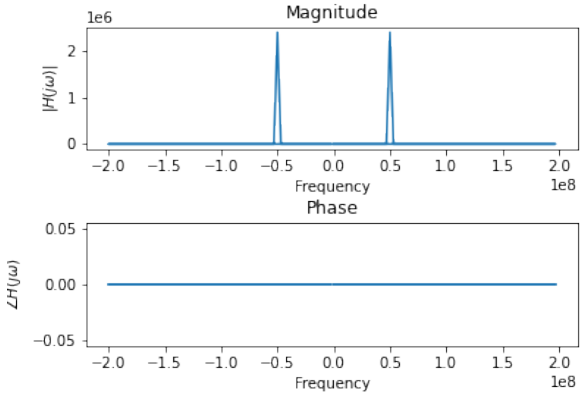
With the parameters mentioned above, the design was run on the board, and following is the results that were observed.

Processing

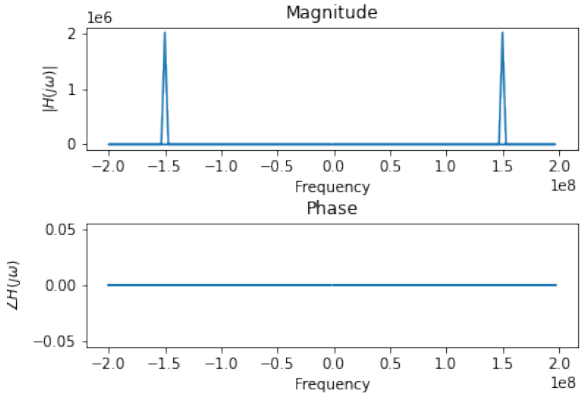
Total time taken to Process	1.1392 ms
Number of points processed	998400
Throughput (Observed)	0.8764 Gpoints/sec
Throughput (Expected)	0.8 Gpoints/sec
Time Taken (Expected)	1.248 ms
Performance	65.7315 Gflops

Plots

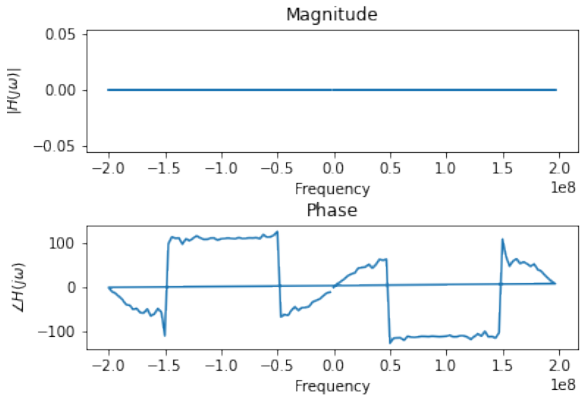
The plots are shown in Figure 5.2 and are as expected. One can observe sharp spikes at the corresponding frequency bins and also the cross-correlated power becomes *null* making the magnitude equivalent to zero.



(a) Self power of signal 1



(b) Self power of signal 2



(c) Normalized Correlation of Signal 1 & 2

Figure 5.2: Plots show Selfs & Cross Magnitude & Phase of Signals under test

5.2 Noise Source

5.2.1 Inputs and Expectations

The Inputs provided to the system were from two Noise Sources, 50% correlated. The values were generated Gaussian randomly and for 2.5μs i.e. 1000 iterations of 128 size FFT. Random amplitude and Phase is expected in the output spectrum which infers that system is responding to the Noise.

5.2.2 Plots

The output plots for Correlation of Noise Source 1, Noise Source 2 and Normalized plot of Cross-correlation of both these can be seen in Figure 5.3.

5.3 Varying Iterations

With the change in the integration time, the power of the correlated output changes with change in the integration time. This can be observed in the magnitude plots of the correlation data.

5.3.1 Inputs and Expectations

For simplicity, we provide single tone signal of 50 MHz to the board at both the inputs and change the number of iterations to be performed. We expect to obtain 2 sets of data and plot them separately in two plots. By doubling the number of iterations for one of the plot, we generally expect the magnitude of the output to be doubled.

5.3.2 Results

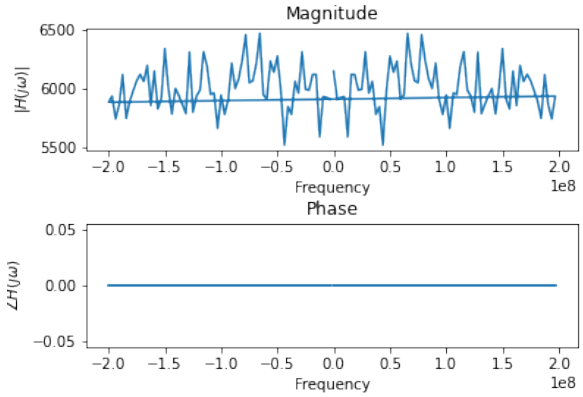
The plot of the magnitude can be seen in the Figure 5.4. It can be seen that the magnitude of the plot from the one with 1000 iterations has been doubled when we increase the number of iterations to 2000.

5.4 Correlated Noise

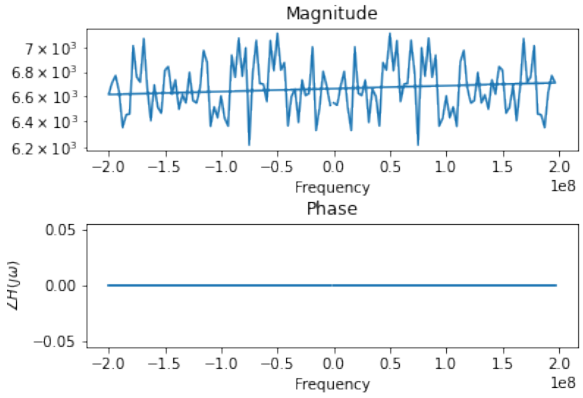
Since we are dealing with Correlator, we must be able to visually observe the power of the cross-correlated signal being reduced when we provide Noise sources with different amount of correlation to each other.

5.4.1 Inputs

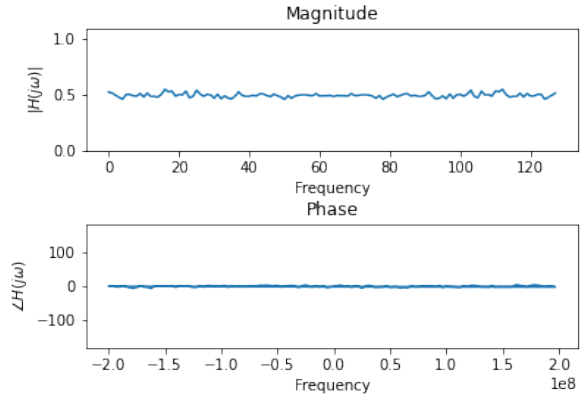
The inputs were 2 noise sources generated with different correlation levels. Generation of these data sets was done using Variable Correlation Digital Noise Source.[2] We provide 3 sets of signals, 2 in each which has 5%, 50% and 100% correlation with each other.



(a) Self power of Noise source 1

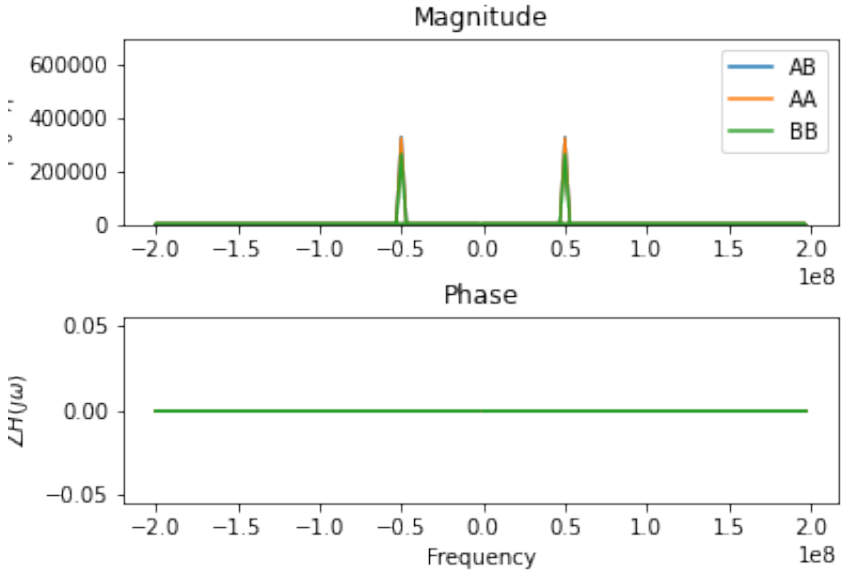


(b) Self power of Noise source 2

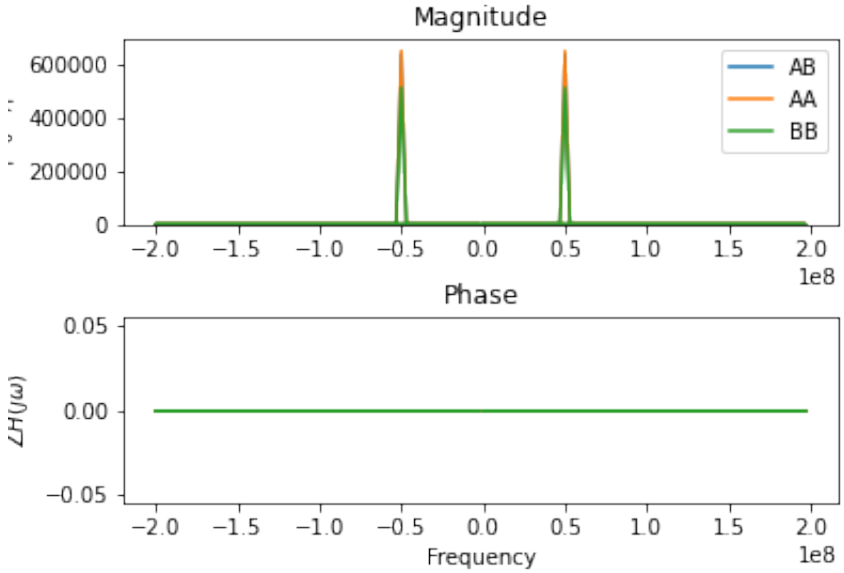


(c) Normalized Correlation of Noise sources 1 & 2

Figure 5.3: Plots show the Selfs & Cross Magnitude & Phase of 2 independent Noise sources

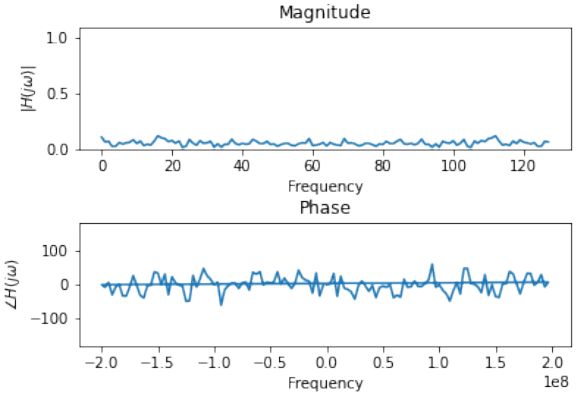


(a) 1000 Iterations

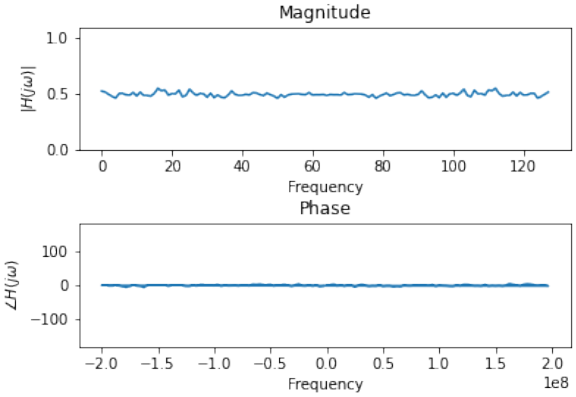


(b) 2000 Iterations

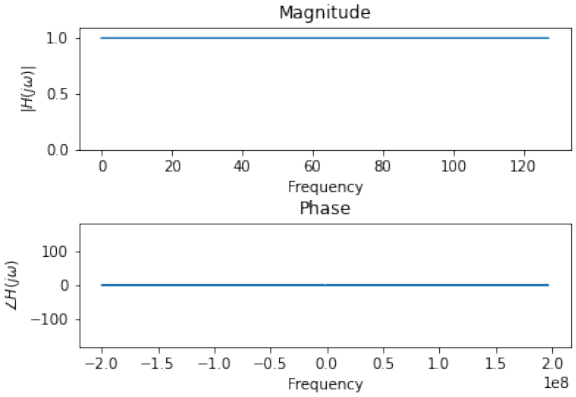
Figure 5.4: The plot shows the Correlated output with different number of iterations



(a) 5% Correlated Signals



(b) 50% Correlated Signals



(c) 100% Correlated Signals

Figure 5.5: Normalized Output of the Cross-correlated signals

5.4.2 Expectation

Since, we are trying to provide different levels of correlation, we expect the magnitude of the cross-correlation output to vary proportionally. When we obtain the normalized cross-correlation of both the inputs, we must observe the proportional increase in them.

5.4.3 Plots

The Figure 5.5 shows the plots of the outputs. Phase plots of these do not explain much of the hypothesis but, Magnitude plots clearly demonstrate the cross-correlation measure when noise sources with different correlation levels are used to test the system.

5.5 Antenna Signals

A set of raw data collect from C0 and C1 Antenna were used to test the design and output was stored and visualized by the help of a python script.

5.5.1 Inputs and Expectations

The data is a 8-bit value stored in a file with extension *.raw*. The host program was modified to read the binary file, convert it into float and transfer it to the device for further processing. The integration of 1 ms was taken with 400,000 samples which meant 3125 iterations of FFTs.

It is obvious to expect various values in multiple bands and also a non-singular cross-correlated spectrum. Since both the Antenna are geographically away, the signal reaching one of them is a little delayed than the other, which infers that there will be phase change in the cross-correlated spectrum and no phase change in the self-correlated powers.

5.5.2 Plots

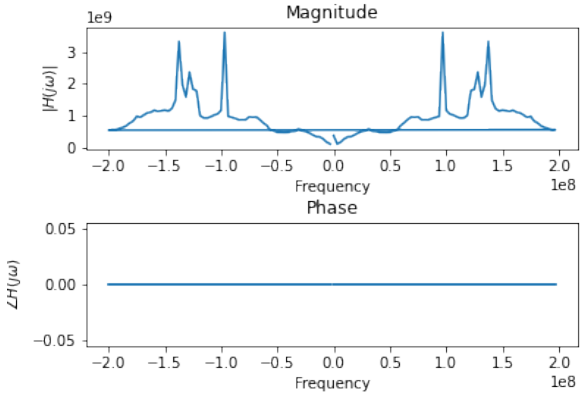
The output can be seen in Figure 5.6. Here, when we compare the magnitude plots of Self-correlated and Cross-correlated outputs, most of the unwanted components are filtered out and signals belonging to very few frequency bins are retained in the cross-correlated spectrum.

5.6 Beamformer

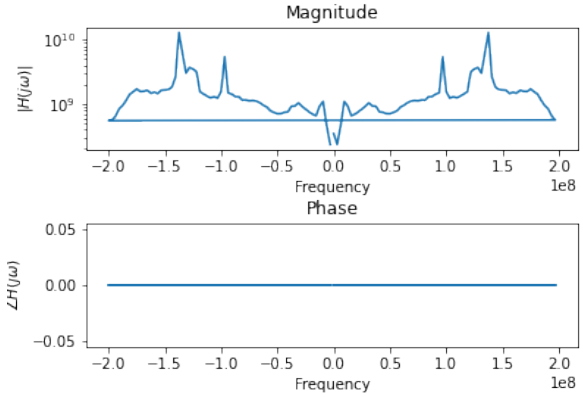
Currently in this test, we have included only Incoherent Array type of Beamformer. The way it is formed is explained in previous chapters.

5.6.1 Inputs and Expectations

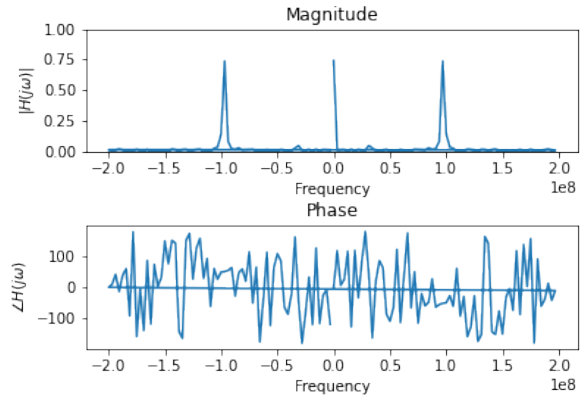
The single tone signals of 5 MHz were used to demonstrate the working of IA Beamformer. Therefore we expect to see a plot of FFT $\times$ Number of iterations / 10 as concatenation of multiple FFTs. Since we are using single tone signals, No noise is expected to show up in the Beamformed output. Number of iterations provided is 200 which should give us 20 concatenated FFT outputs of 5 Mhz signals.



(a) Self power of Signal from C0 Antenna



(b) Self power of Signal from C1 Antenna



(c) Normalized Cross - Correlated Power

Figure 5.6: Plots showing Correlated Signals from C0 and C1 Antenna

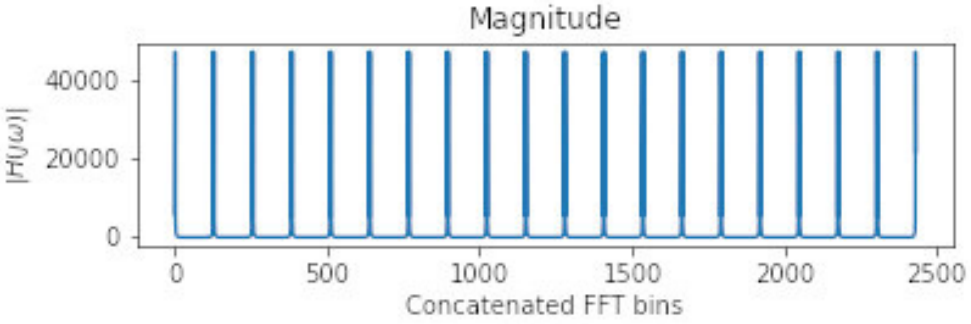


Figure 5.7: The plot of IA Beamformer output of 5 Mhz Signal

5.6.2 Plots

The Figure 5.7 shows the plots of Amplitude of the beamformation of the 5 MHz signal. The FFTs are concatenated to other and if we increase the number of iterations, the amplitude will remain the same, but the graph goes on continuing in X-axis. Each segment of FFT is integrated for 10 iterations just to make the output of Beamformer smaller and also avoid fetching DDR memory of *iteration* number of times.

5.7 Fractional Delay Correction and Fringe Stop

We used 100% Correlated noise to test Fractional Delay, Delay Rate, Fringe and Fringe Rate. When we have 100% correlated noise as inputs, there would be no change either in delay or in fringe. We can then add the above mentioned parameters to check if we are able to achieve the shift in phase and time. For instance, If we pass 90° to be Fractional delay, we must be able to observe 90° increase in phase over the spectrum (i.e. ramp from 0° to 90°). Corresponding plots can be seen in Fig. 5.8.

5.7.1 Varying Fractional Delay

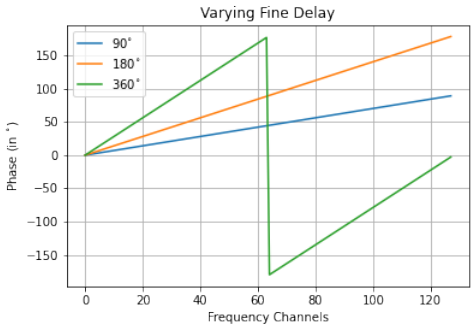
We try to vary the fractional delay only, keeping every other parameter constant and obtain the cross-phase between both the signals.The plot can be seen in Fig.5.8 .

5.7.2 Varying Fringe

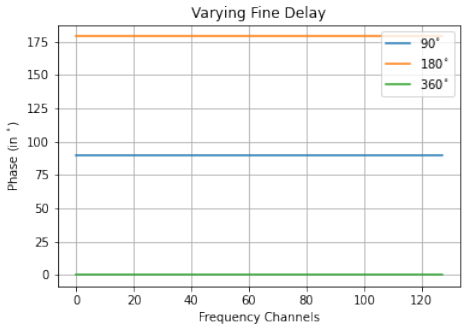
We try to vary the Fringe only, keeping every other parameter constant and obtain the cross-phase between both the signals.The plot can be seen in Fig.5.8 .

5.7.3 Varying Delay Rate

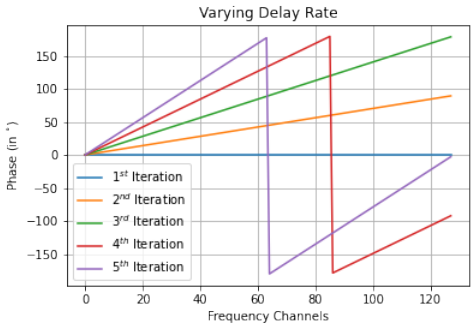
We try to vary the Delay Rate only, keeping every other parameter constant and obtain the cross-phase between both the signals. We expect to have no ramp at 1st iteration, but as we increase the number of iteration, we expect to see ramps increasing from 0° to θ° in steps. The plot can be seen in Fig.5.8 . The rate was fixed to $\frac{\pi}{2}$



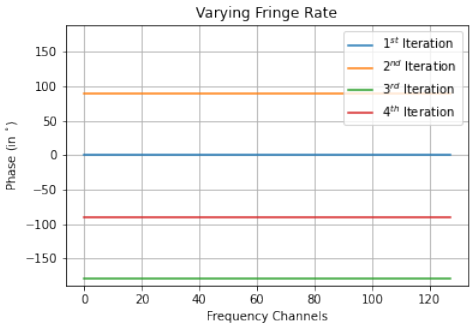
(a) Varying Fractional Delay



(b) Varying Fringe



(c) Varying Delay Rate



(d) Varing Fringe Rate

Figure 5.8: Phase of Cross Correlated Signals

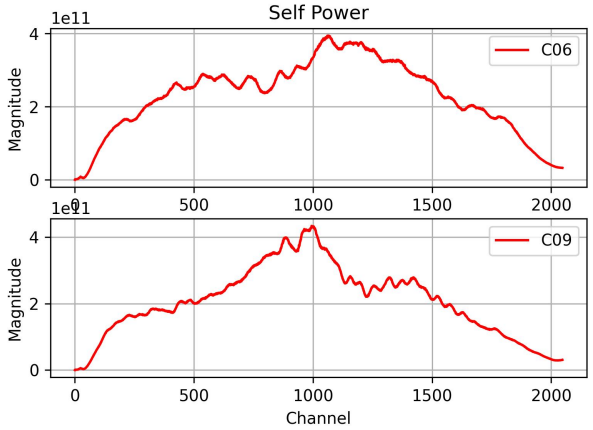


Figure 5.9: Self-power of both Antenna over a single spectrum

5.7.4 Fringe Rate

Similar to Delay Rate, even Fringe Rate was varied by $\frac{\pi}{2}$ and output was observed for every iteration.

5.8 Testing Correlator on 3C147 Source

5.8.1 Inputs

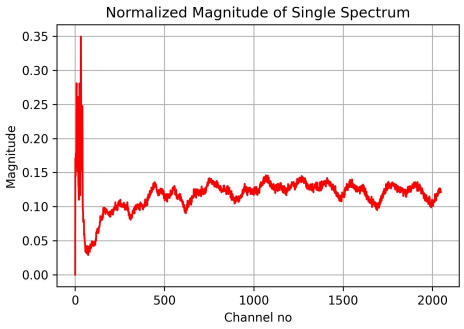
Observation was made on the source 3C147 using C06 and C09 antenna from the central square array and the raw voltages were recorded. In order to correct the delays and for fringe stop, the instantaneous values of F.S.T.C, Delay Rate, Fringe and Fringe Rate were also recorded for the corresponding Antenna. This set of data was used to process. In addition to above mentioned parameters, Phasing have to be done to make the output perfectly in phase. Using the original output obtained from correlator, phases were calculated to apply them for the same data set. One can even notice the difference in the plots given below.

5.8.2 Plots

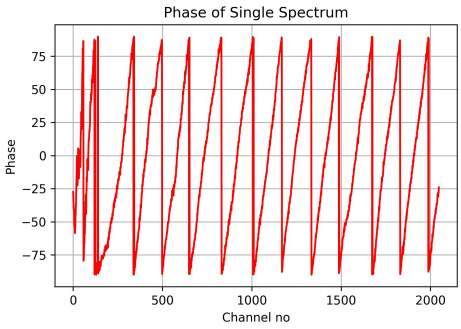
When analysing the outputs, one can have the self-power of the single antenna over the spectrum. This is shown in Fig. 5.9. The cross correlation can be observed in other plots. Normalized value of the cross-correlation in all the cross-plots. Fig. 5.10 represents Magnitude & Phase of Cross correlated signals over a spectrum followed by cross magnitude & phase of a single channel (Channel 1000) over time. Similarly, Fig. 5.11 shows the same plots but, after the phasing has been done.

5.8.3 Results

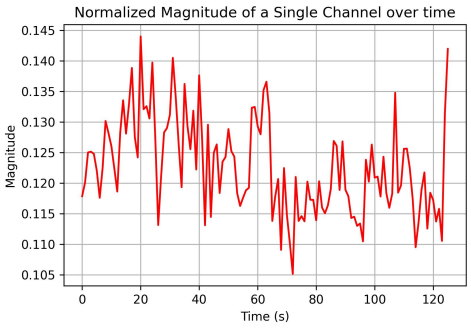
Even though the output is delay compensated and post fringe-stop, there will be some phase that needs to be additionally corrected. This is calculated by running the set of data, obtaining output, calculating phases and using those phases for correcting. This



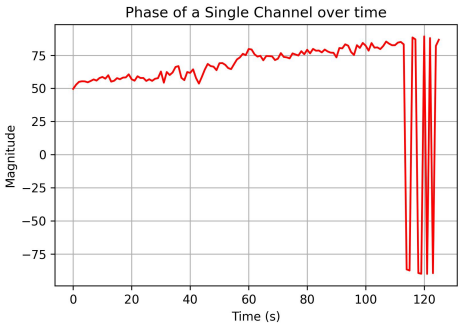
(a) Magnitude Plot



(b) Phase Plot

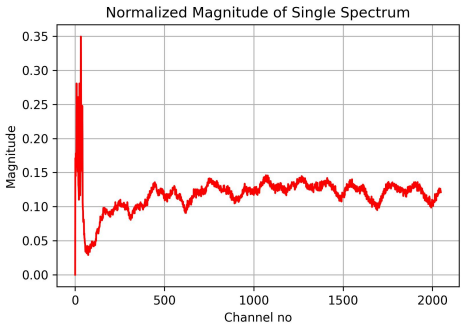


(c) Magnitude Plot

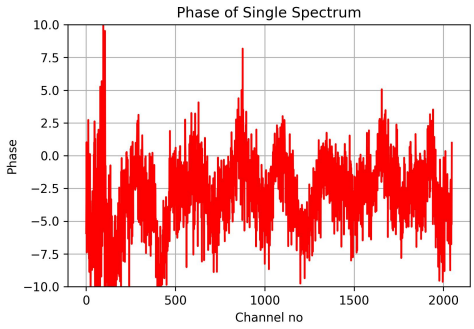


(d) Phase Plot

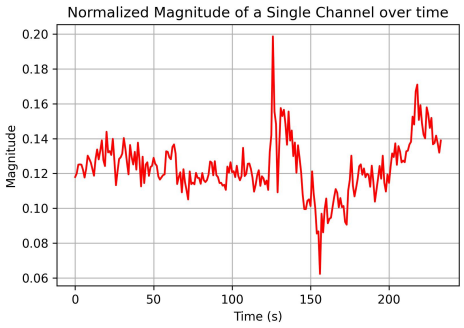
Figure 5.10: Cross of C06 & C09 before phasing



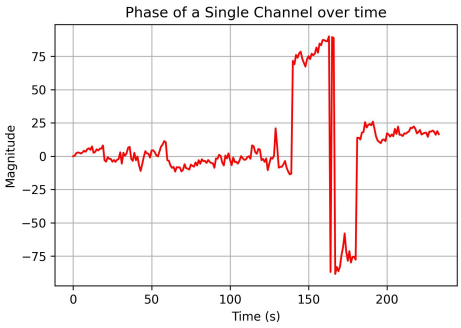
(a) Magnitude Plot



(b) Phase Plot



(c) Magnitude Plot



(d) Phase Plot

Figure 5.11: Cross of C06 & C09 after phasing

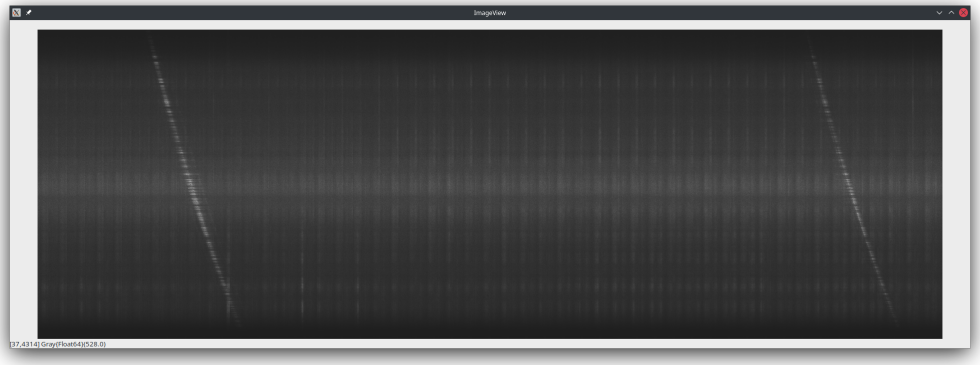


Figure 5.12: Monochrome Heatmap of Intensity vs FFT channel & Time

can be noticed when Fig. 5.10 and Fig. 5.11 are compared. Once the phasing is done, the cross phase gets nullified to great extent and this can also be observed over time.

Due to addition of so many parameters to the design and constraint of available system resources, the performance that could be achieved is given below.

Time taken to Process 1 segment	11.636 s
No. of points processed in 1 seg	268,435,456
Throughput (Observed)	0.0461 Gpoints/sec
Throughput (Expected)	0.2 Gpoints/sec
Time Taken (Expected)	1.342 s
Performance	3.5755 Gflops

5.9 Observing Pulsar B0329+54

5.9.1 Inputs

A Pulsar - B0329+54 was observed using 2 Antenna (C06 & C08). The ADC outputs were recorded into a file, which were further used for processing. The antennas were focused on a calibrator source (Here 3C147) and phases were calculated using cross-correlation output. These phases were again applied on the pulsar data to obtain phased signals from corresponding antenna.

5.9.2 Expectation

Both Incoherent Array and Phased Array were tested in this and the output was stored in binary format which was later visualized using self-written program and also using GPTool [4] . A signature of a pulsar is expected to appear when we look at heatmap of FFT's over a period of time. The pulsar that we are observing has a period of 0.71452 seconds which should be evident in our results.

5.9.3 Results

Signature of a pulsar can be seen in fig. 5.12. By the image we can obtain the difference between both the pulsars to be around 4360 pixels, making the time period between 2

pulsar sweep to be 0.71434 *sec.*

When the output data is processed using GPTool, the snaps of obtained outputs are shown in Fig. 5.13. Simultaneously, when the data was being recorded, GWB was processing. The results are shown in Fig 5.14.

5.9.4 Comparision with GWB

The outputs of IA Beamformer from the design was compared with that from GMRT Wideband Backend (GWB[11]) with the help of phase profile. In Fig. 5.15, IA Outputs of both - design and GWB are compared to each other. In Fig. 5.16, PA Outputs of both - design and GWB can be seen.

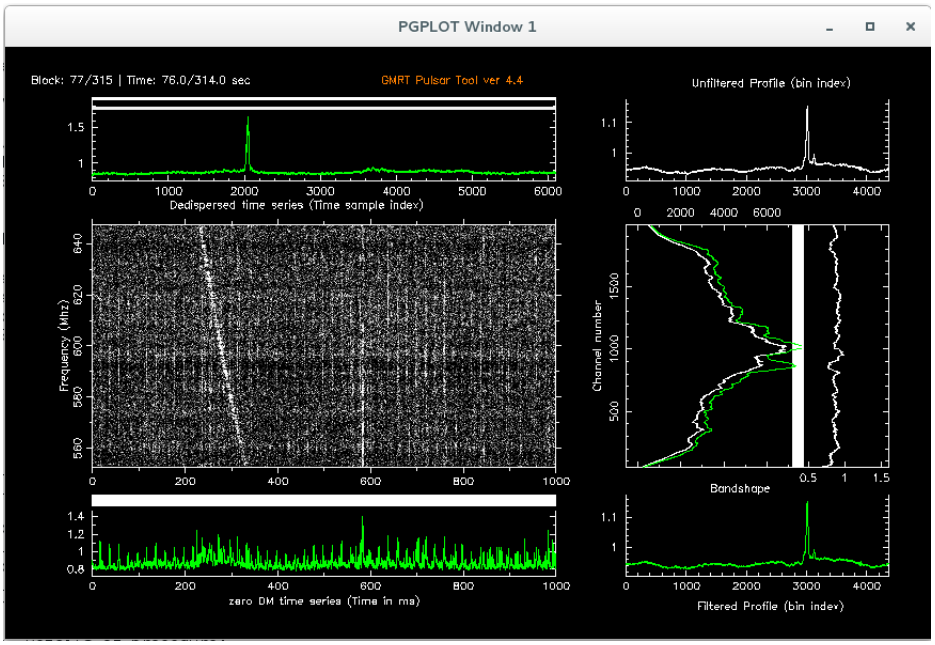
5.9.5 Inference

The pulsar can be seen in both IA as well as PA Beamformer. Phased Array Beamformer has sensitivity equal to $\sqrt{N}$ times that of Incoherent array, where N is number of Antenna (here $\sqrt{2}$). This is not reflected in the above shown Fig 5.13 and Fig 5.14. This might be because of multiple reasons. Firstly, The technique of calculating the phase for correction is different in the design when compared to that used in GWB, This might be a problem as these calculated phases are multiplied to FFT values which in turn are used to form the Beam. Another reason could be that, as we can see in Fig. 5.16, The peak of PA beam is also not at $\sqrt{2}$ times that of IA. This might be issue of RF-Interference that can be seen in the GPTool snaps 5.13 and 5.14. This might have caused our SNR to degrade to the value that we have obtained.

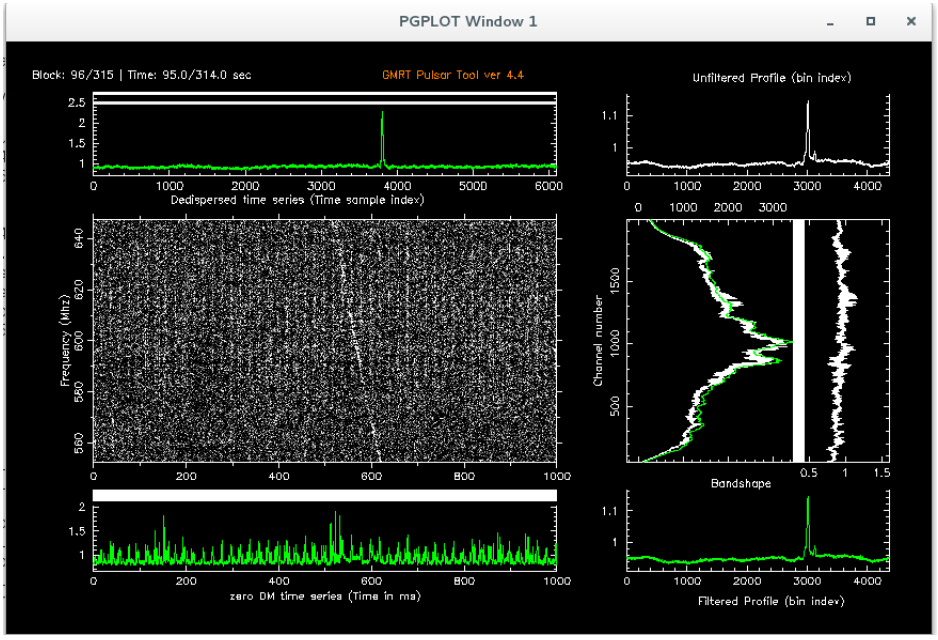
5.10 Static Power Consumption Report

Paramter	**Value (mW)**
Total Thermal Power Dissipation	32956.54
Transceiver Standby Thermal Power Dissipation	3047.21
Transceiver Dynamic Thermal Power Dissipation	6267.18
I/O Standby Thermal Power Dissipation	1358.67
I/O Dynamic Thermal Power Dissipation	4680.40
Core Dynamic Thermal Power Dissipation	10185.68
Device Static Thermal Power Dissipation	7417.82

The table shows us the static power report of the design with 50% I/O signals toggling, i.e. the maximum number of toggles that a signal can undergo in a single clock cycle of the fabric clock and 50% engagement of the clock signifies that the I/O signal cannot toggle twice in the same clock cycle. If this factor increases, it refers to the possibility of a glitch that can occur during the input of the signal.

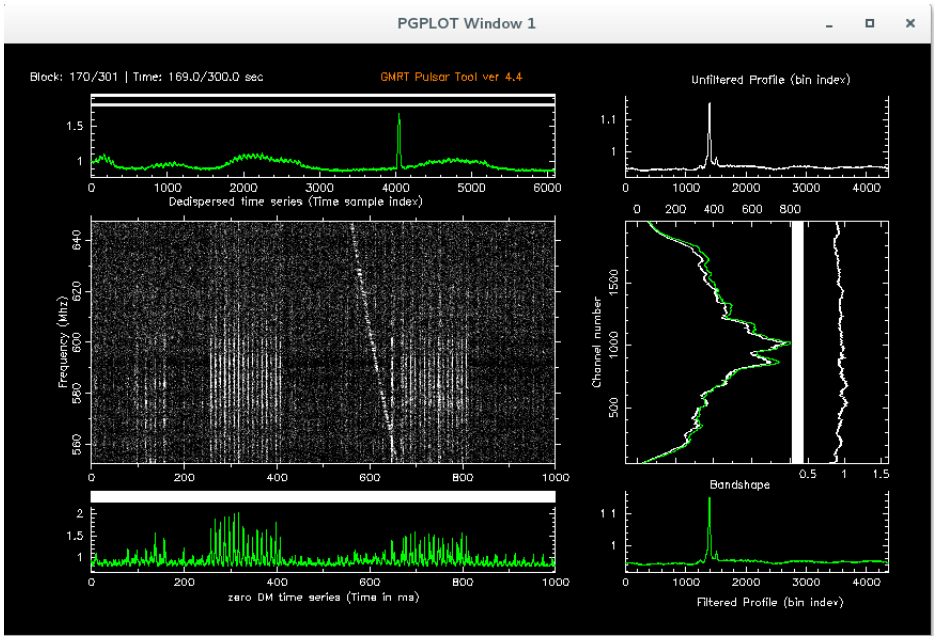


(a) Incoherent Array

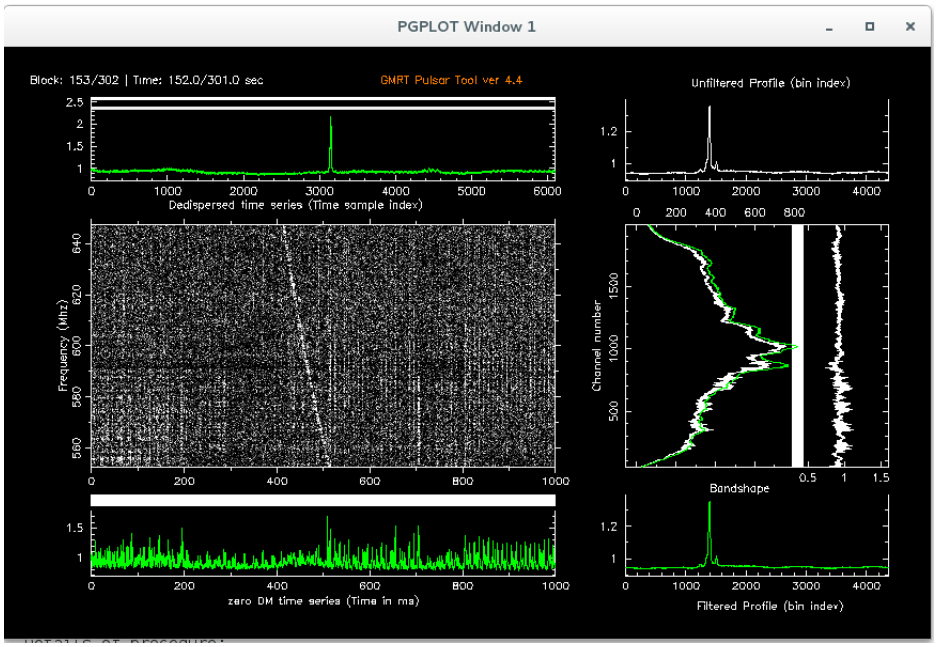


(b) Phased Array

Figure 5.13: Snapshots of GPTool processed by Test-Design

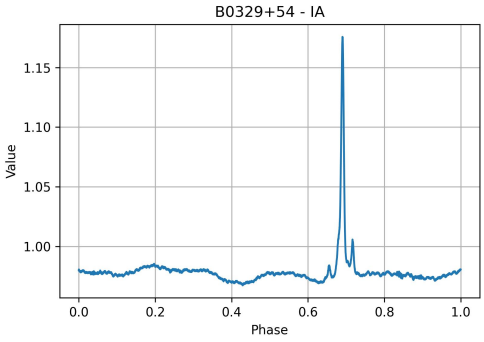


(a) Incoherent Array

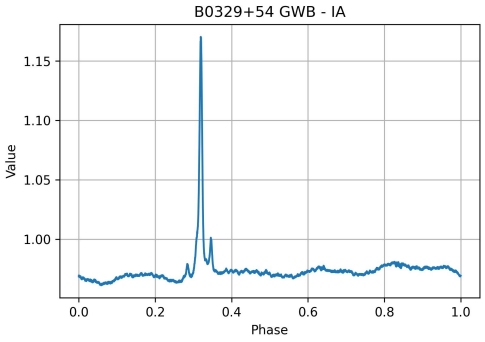


(b) Phased Array

Figure 5.14: Snapshots of GPTool processed by GWB

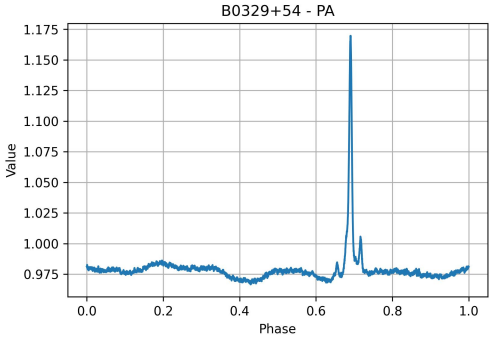


(a) Normalized output from design

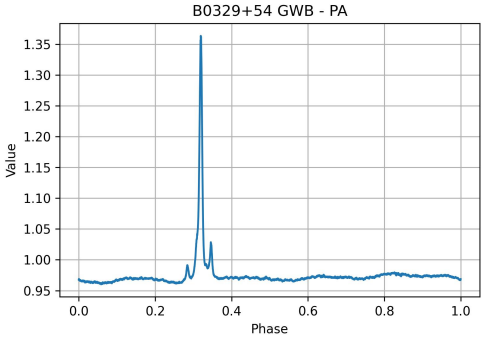


(b) Normalized output from GWB

Figure 5.15: Filtered Phase Profile - IA of B0329+54



(a) Normalized output from design



(b) Normalized output from GWB

Figure 5.16: Filtered Phase Profile - PA of B0329+54

Chapter 6

Conclusion

6.1 Summary

The design of 2 Antenna (single pol) Correlator and Beamformer was implemented on Nallatech 385A board with Intel Arria 10 FPGA using OpenCL standard. The size of FFT being 4k points with 100 MHz Bandwidth and has fixed LTA of $1.342s$ and STA of $163.84\mu s$. All these parameters have flexibility to vary according to the user's need.

Feasibility of running an off-line Correlator and Beamformer on FPGA Accelerator card is demonstrated using antenna raw voltages. Use of Intel's FPGA SDK for OpenCL have reduced the development cycle and made FPGAs more accessible to high level programmers. The outputs obtained from the design matches the ones from GWB to a great extent. In additiom to these, Parallel programming techniques were used to achieve real-time processing of data upto 400MHz bandwidth.

6.2 Future Scope

- Number of Antenna that this prototype is designed for is just a fraction of actual number of antenna used for observation. This implies that there needs to be thoughtful scaling of the same design and manage their data flow to obtain desired combinations of correlations from multiple antennas. For 2 element correlator, we would obtain 3 correlation values but for 30 elements, 465 correlation outputs are expected.

- To make the system real-time capable, one must also consider the overhead time that is being used up for data transfer from the host to device and vice-versa. There are multiple options to build this, either using the available I/O ports or smartly handling data transfers between two data processing sequences.

- When it comes to real-time operation of the system, apart from flexibility of parameters such as LTA & STA, even multi-threading of process plays important role, as the overhead introduced by Host computer itself will overpower the processing performance of the system.

- A Single device might not be able to implement for bigger set of Antenna, hence a multiple Accelerator cards can be set up and smartly managed to extract maximum efficiency both from the whole system as well as each Card.

- Bit reduction algorithms can be inculcated to reduce the amount of hardware used and can use the resources available in the device to its fullest extent.

Bibliography

[1] Martijin Berkers. Solving convex optimization problems on fpga using opencl. Master's thesis, Delft University of Technology, 2020.

[2] Kaushal D. Buch, Yashwant Gupta, and B. Ajith Kumar. Variable correlation digital noise source on fpga — a versatile tool for debugging radio telescope backends. *Journal of Astronomical Instrumentation*, 3, 2014.

[3] Jayaram Chengalur, Yashwant Gupta, and K Dwarkanath. *Low frequency Radio Astronomy*. National Centre for Radio Astrophysics, 01 2003.

[4] Aditya Chowdhury and Yashwant Gupta. Real-time rfi mitigation for the beamformer mode of the upgraded gmrt. In *General Assembly and Scientific Symposium*. URSI, August 2017.

[5] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, pages 297–301, 1965.

[6] Intel. *Intel FPGA SDK for OpenCL Programming Guide*, 2017.12.08 edition, 12 2017.

[7] Intel. *Intel Arria 10 Device Overview*, 2018.

[8] Dimitris G. Manolakis John G. Proakis. *Digital Signal Processing*. Pearson, 4 edition, 2007.

[9] Khoronos OpenCL Working Group. *The OpenCL Specification*, 1.0 edition, 6 2009.

[10] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, 2006.

[11] Suda Harshavardhan Reddy, Sanjay Kudale, Upendra Gokhale, Irappa Halagalli, Nilesh Raskar, Kishalay De, Shelton Gnanaraj, Ajith Kumar B, and Yashwant Gupta. A wideband digital back-end for the upgraded gmrt. *Journal of Astronomical Instrumentation*, 6(1):16, 2017.

[12] Jayanta Roy, Jayaram N. Chengalur, and Ue-Li Pen. A post-correlation beamformer for time-domain studies of pulsars and transients. *The Astrophysical Journal*, 864(2):160, 2018.

[13] Matthew Scarpino. *OpenCL in Action*. Manning Publications Co., 2012.