URSI - RCRS 2022, IIT (Indore), India, 1 - 4 December, 2022

Multi-element Correlator & Beamformer using OpenCL on FPGA Accelerator Card

Raghuttam Hombal*(1), Mekhala V. Muley(2) Harshvardhan S. Reddy(2), Sanjay S. Kudale(2), Jayanta Roy(2)
(1) Savitribai Phule Pune University, Pune, India; e-mail: raghuttamhombal@gmail.com
(2) GMRT Metrewave Radio Telescope, NCRA-TIFR, Pune, India.

1. Abstract

Radio Interferometry refers to the process of combining signals from multiple antennas to form an image of the radio source in the sky. Radio-astronomical signal processing using array telescopes is computationally challenging and poses strict performance and energy-efficiency requirements. The GMRT is one of the largest arrays with many antennas working in the metre wavelength. The ongoing developmental activities for expansion of the GMRT (called eGMRT) demand a many fold increase in the computational cost and power budget while providing an increased collecting area as well as field-of-view by building more antennas each equipped with phased array feed (PAF). Recent FPGAs provide higher Flops per Watt making it an energy-efficient hardware platform suitable for projects like the eGMRT requiring a high compute-to-power ratio. However, the traditional programming model for FPGAs is a primary drawback of using FPGAs for high-performance computing. Aided by the recent advancement of parallel programming on FPGAs using Open Computing Language (OpenCL), allows FPGAs to be used as general purpose accelerators like GPUs. The aim of this project is to design an energy-efficient multi-element correlator and beamformer on an FPGA Accelerator Card using OpenCL and to explore the possibilities of using such systems for real-time, number-crunching tasks.

2. Overview of GMRT

The Giant Metrewave Radio Telescope (**GMRT**)[1] is an array of 30 antennas, each being 45 m in diameter designed for study of Radio Astronomy. The Observatory has 14 antennas 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 Fig 1. The central square provides shorter baselines, which are useful for imaging large extended sources whose visibility is focused at the origin of the U-V Plane. The arms are very useful in imaging 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.

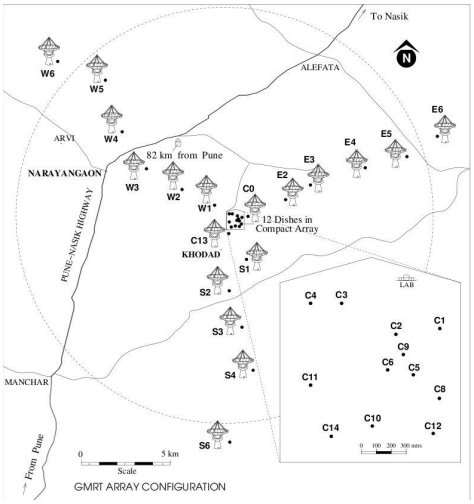

Figure 1. GMRT Array Configuration [1]

The GMRT currently operates at 4 different frequency bands distributed within the range of 120 MHz to 1400 MHz. The feeds work at the ambient temperatures only. Band selection and preliminary amplification of RF signals is done in the 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 the antenna base for further processing. Signals are then fed to laser-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, Roy et al. 2010)
- uGMRT system - processing 400MHz bandwidth using GMRT Wideband Backend (GWB, Reddy et al. 2017)

In GMRT, FX correlator is used for interferometry mode, supporting both total intensity and full polar mode. A range of observing bandwidths from 1.5625MHz up to 400MHz are available with varied resolution from 95.37Hz up to 0.1953MHz. The correlator output / visibilities are available at a rate of 0.674sec.

In parallel with the correlator, GWB also has incoherent array (IA) and phased array (PA) beam-formers for the array mode giving beam data at 0.08192ms which are useful for observations of sources such as pulsars and fast transients.

3. Heterogeneous Computing

High performance computing processes huge volumes of data at very high speed (above teraflops) using multiple computers and storage devices. There are four major computing platforms - ASICs, CPUs, GPUs and FPGAs. ASICs or **Application Specific Integrated Circuits** are custom-designed for application with optimum combination of performance and power consumption but have the longest development time with high cost. 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. In recent times, GPU or **Graphical Processing Unit** emerged as a system that could allow users 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. But the major drawbacks of GPUs include its hard wired architecture restricting implementation of many functional possibilities and high power consumption.

FPGA or **Field Programmable Gate Array** have gained popularity over a period of time and emerged as an alternative to GPUs in the field of HPC. The best thing about FPGAs is that they are reconfigurable and can be programmed many times as per the user's requirement. Recent FPGAs provide higher Flops per Watt making it an energy-efficient compute platform for HPC applications requiring high compute-to-power ratio. Conventionally they are configured through Register-Transfer Level (RTL) descriptions using VHDL or Verilog. Synthesis tools (most of the time vendor specific) convert these HDL codes to RTL and then to a 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 a smoother development environment.

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[2]. 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 to the programmer to 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.

3.1 OpenCL Architecture

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 viz. Kernel program and Host program. The **Kernel Program** is the main digital signal processing system to be implemented on the device and is written in C++. In case of FPGAs, the kernel program is further converted into bitfile by High-Level Synthesis (HLS) tools using Board Support Package (BSP). The **Host Program** is responsible for creation of required resource entities, allocation of different kinds of memory units, copying of the buffers, queuing the tasks for execution, reading back the data from the device, managing contexts, etcetera. The Host program is compiled by the native C/C++ compiler and executed on the host machine. Multiple levels of memory viz. Global, private and local memories are available. The user is responsible for making the best use of the memory types available. When we use FPGA or GPU, the Global Memory available in the form of DDR memory is used to eliminate the overhead of transferring data multiple times over the Host-Device interconnect. There is also Local memory and Private memory available in the device.

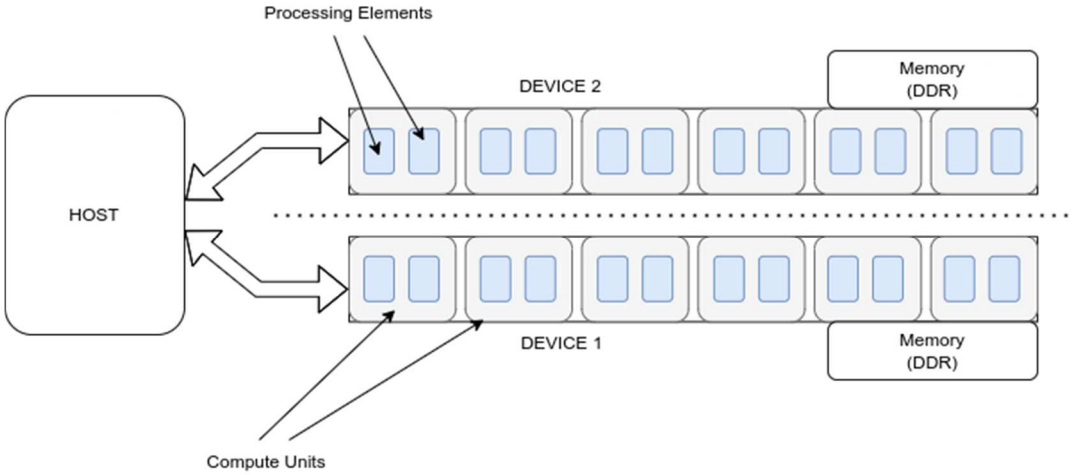

Figure 2. Represents OpenCL Architecture and Execution Model

A **Work-item** is an elementary unit of computing on the device. It is accompanied by **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 has 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 the same task or executes the same set of instructions. 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 resources. 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 it possible to create **Command Queues**, the structures that allow hosts to send kernels to devices, and enqueue the tasks[3]. 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.

4. Implementation

A 2-element correlator and beamformer is implemented on Intel's FPGA accelerator card using OpenCL. It processes 8-bit ADC raw voltages from antennas sampled at 200MHz in offline mode. Digital signal processing modules used in radio telescopes like Fast Fourier Transform, Delay Correction, Fringe Stop, Correlation and Beamforming (Incoherent Array, IA as well as Phased Array, PA) are implemented using parallel processing techniques with floating point precision. The correlator output i.e. visibilities and beam data (IA and PA) are available at the rate of 1.342s and 163.84 μs respectively. The design supports real-time processing of 400MHz bandwidth. The size of FFT can be varied and the design can be compiled for a range of 128 to 8K, but we have fixed FFT size to 128 Point FFT for lab tests and 4K point FFT for real-world signals. The card that is being used for the project is a 385A PCIe FPGA board that is powered by **Intel Arria 10 GX 1150** FPGA provided by Nallatech Limited (currently owned by Bittware[14]). This is a low profile accelerator card equipped with a powerful PCIe bus to enable fast data transfers between on-board Memory chips and FPGA itself.

4.1. Block Diagram and Signal Flow

The block diagram of the implemented system is shown in Fig 3. 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 DDR banks in order to speed up the process of data transfer between Memory and FPGA. In DDR memory an array of required size i.e. Number of Iterations * FFT size is allocated for storing pre-recorded raw voltages from each antenna. Coarse delay compensation is done in the host itself. Once the task starts, 2 independent chains begin to process the data from DDR memory. Data is fetched in batches of 8 samples, processed and written on the respective channel. Once a batch of 8 samples is written on the channel, MAC and Beamformer kernel reads both the channels, and performs their function simultaneously. The Multiply and Accumulate section, as the name suggests, performs multiplication on the spectral channels and does iterative addition of the correlated values for a long time (LTA-Long

Term Accumulation). Beamformer supports two modes - Incoherent Array and Phased Array. The outputs are accumulated for a very small time (STA-Small Time Accumulation). Outputs from MAC and Beamformer are transferred to the DDR memory, which is then read by the Host. Host program writes the processed visibilities and beam data on a file.

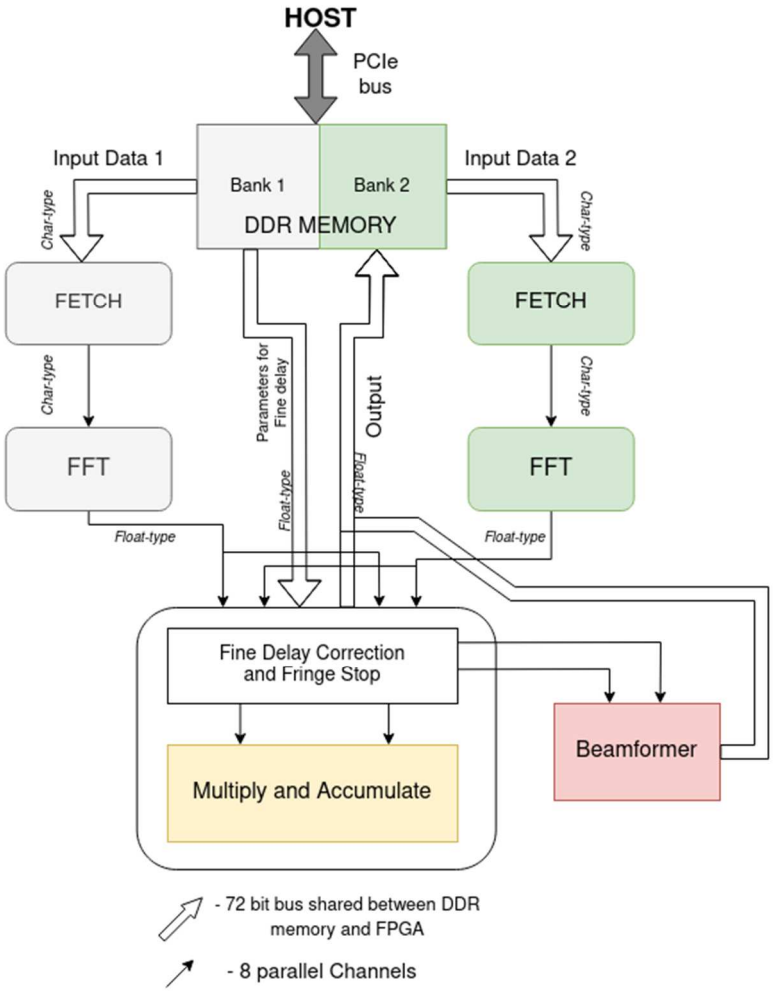

Figure 3. Block Diagram of the System

4.2. Fetch and FFT

Fetch kernel is responsible for fetching the data in batches of 8 samples in a single clock. 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 which reads data from the channels and performs 4096 points FFT with floating point precision. 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 are read by another kernel for further processing.

4.3. Fractional Delay Compensation and Fringe Stop

Residual delay (delay not corrected by sample shifting) and fringe stopping is done by using the Time-shifting property of Fourier Transform. The formula is given by Eq. 1.

$$A_{corrected}(n) = A(n) \cdot e^{\frac{\tau}{N} - \theta} \quad (1)$$

where, τ = Fractional Delay + (Delay Rate $\times$ Time)

and, θ = Fringe + (Fringe Rate $\times$ Time)

In the above equation Fractional Delay, Delay Rate, Fringe and Fringe Rate values are calculated by a standard program for given source, its position and frequency of observation, whereas Time is the time required to process one FFT spectrum. This information for both the inputs is passed during the execution to the device from the host. With respective values of above mentioned parameters, τ (Delay) and θ (Fringe) are calculated. A single phase angle will be calculated for each antenna and *cis()* of this factor is multiplied to the corresponding spectral channel of appropriate FFT spectrum. This operation is performed soon after the channels which contain output of the *FFT* kernel are read.

4.4. Multiply and Accumulate

As the name suggests, this block is responsible for multiplying spectral channels of two synchronized signals and producing correlated output of the same. The correlated output is integrated and stored in a local array and then transferred to the DDR Memory so that Host could access the result. The extent of integration depends upon the number of iterations that are being processed, in this design it is 1.34s. The resultant or visibilities would be an array of size the same as FFT, with each spectral channel having two components - real and imaginary which are stored in the DDR Memory.

4.5. Beamformer

There are two sections in the *Beamformer* kernel namely - IA and PA. In IA, spectral channels of two synchronized signals are first squared and then added. IA Beamformer is governed by the formula Eq. 2.

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

The squared voltages from both the antennas are integrated for 163.84 μs which corresponds to 8 FFT cycles. Integration happens in the local memory only, but after every 8 such integrations, the chunk is written on to the DDR Memory, and the DDR memory pointer is incremented by a 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 the same as IA mode (i.e. 8 FFT Cycles). PA Beamformer is governed by the formula Eq. 3.

$$P_{PA} = \left[\sum_{i=0}^{N-1} V_i\right]^2 \tag{3}$$

With the implemented design we consumed upto 81% of DSP Blocks of the Intel Arria 10 FPGA.

5. Tests and Results

The design implemented on the FPGA accelerator card is tested using real antenna signals. Tests include 2 components - Testing the correlator on a Radio Source and Testing the Beamformer on a Pulsar. The antenna voltages were recorded and processed in the FPGA accelerator card in an offline mode. GWB processed the same data in real time to get the visibilities at 1.34sec and IA-PA beam data at 163.84 μs.

5.1. Source: 3C147

Observation was made in Band4 on the source 3C147 using C06 and C09 antenna from the central square array and the raw voltages of 100MHz bandwidths were recorded for a few mins. 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 information was used while processing the data offline. In addition to above mentioned parameters, Phasing has to be done to make the output perfectly in phase before observing Pulsars. Phases were calculated from the visibilities itself and applied on the same data set. Cross amplitude and phase plot over the band are as shown in Fig. 5 & 6, whereas Fig. 7 & 8 are Cross amplitude and Phase of single channel over time.

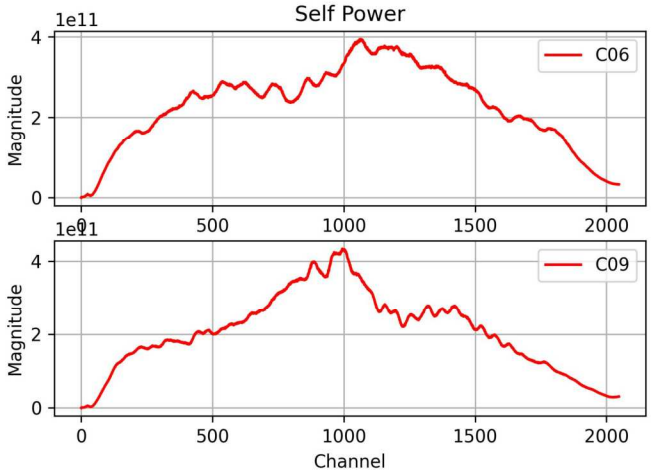

Figure 4. Self correlated power of both input signals over a spectrum

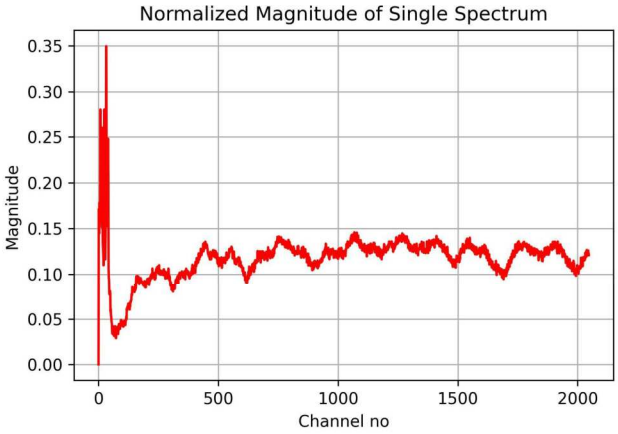

Figure 5. Normalized Cross amplitude Plot over a spectrum

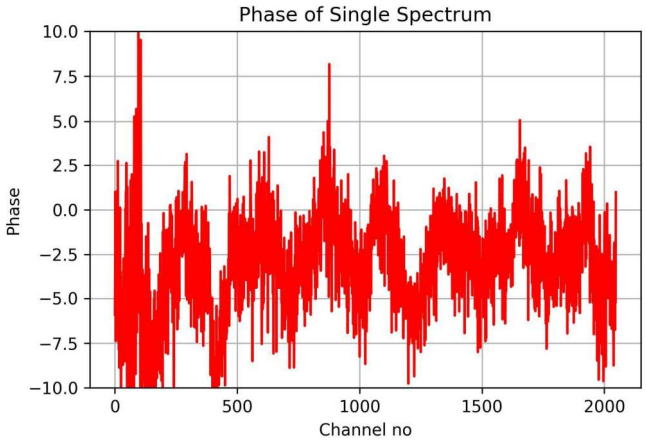

Figure 6. Phase Plot over a Spectrum

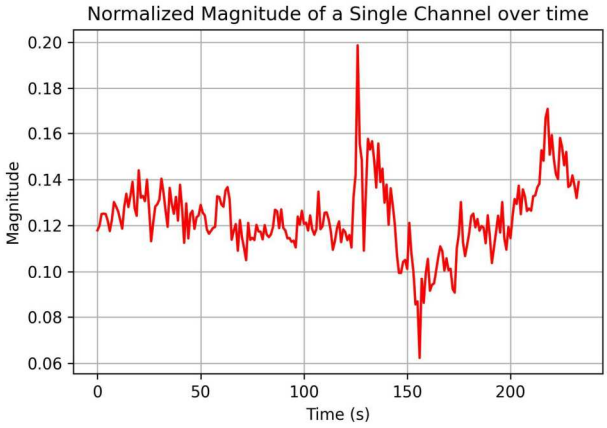

Figure 7. Normalized Cross amplitude of single channel over time

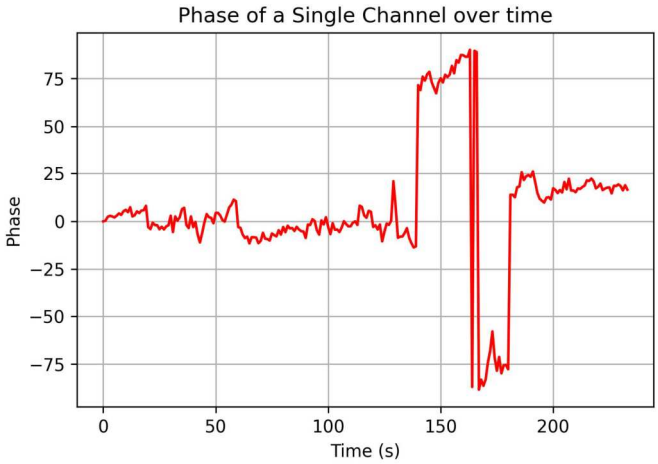

Figure 8. Phase of single channel over time

5.2. Pulsar: B0329+54

Figure 9. Monochrome Heatmap of Intensity vs FFT channel & Time

Pulsar B0329+54 was observed in Band 4 (550 - 650 MHz) using C06 and C09 antenna from the central square array and the raw voltages of 100 MHz bandwidths were recorded for a few mins. The raw data was processed in the FPGA accelerator card with proper delay, fringe stop and phasing applied. The preliminary analysis of processed beam data is done by plotting spectral channels on Y-axis and time on X-axis shows signatures of a pulsar as shown in Fig 9. From the figure we can obtain the period of the pulsar to be around 4360 pixels, making the time period between 2 pulses to be 0.71434 s. Beam output data is processed using GPTool, the snaps of obtained outputs are shown in Fig.10 & 11. When raw data was being recorded, GWB was processing it simultaneously in real time. The GWB processed beam data when analyzed using Gptool has a Pulsar profile plot as shown in Fig. 10 & 11.

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 $\sqrt{2}$). GWB processed PA beam data shows improvement by factor of $\sqrt{2}$ as compared to IA beam data. However this is not reflected in the data processed using FPGA accelerator card design. The reason could be that the algorithm used for phasing is different in our design as compared to that used in GWB which can be seen as a difference in Fig. 11.

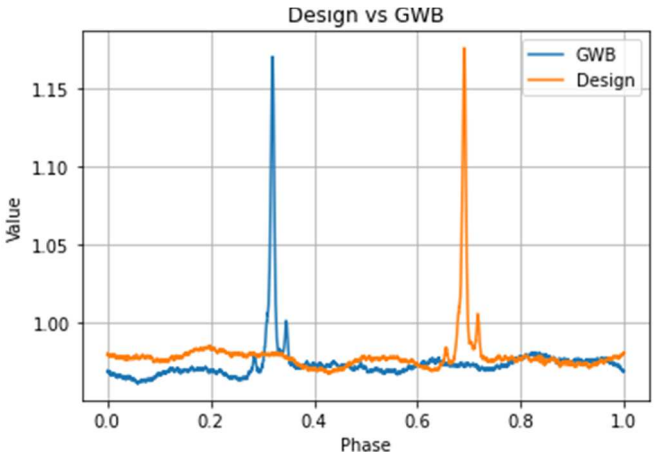

Figure 10. IA - Pulse Profile

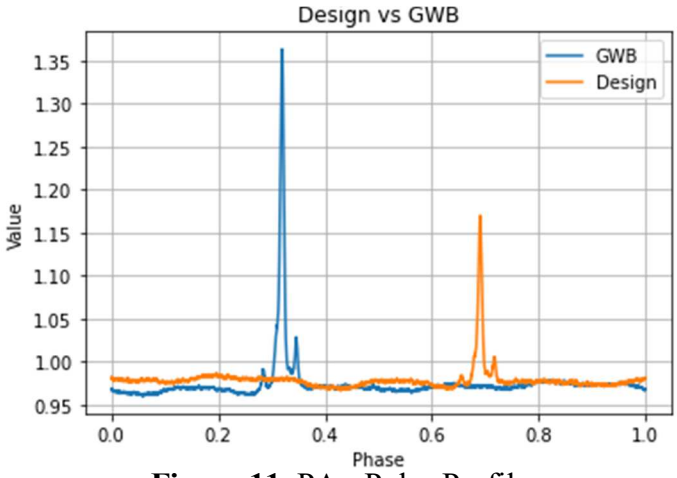

Figure 11. PA - Pulse Profile

5.3. Static Power Report

Power Consumption is estimated by the Power Analyzer tool in Quartus Prime Intel FPGA SDK. It was estimated to consume about 32.9 watts including all the statics and dynamics of the system at 50% IO Toggling rate (i.e. maximum numbers of toggles that a signal can undergo in a single clock cycle). Currently, our design of 100 MHz bandwidth runs at 0.152 Gflops/Watt but, the maximum we could achieve (performing FFT & MAC at 400 MHz) was 2.26 Gflops/Watt.

6. Conclusion

An offline 2-Antenna (single pol) Correlator and Beamformer processing 100MHz bandwidth with 4096 points FFT was implemented on Nallatech 385A board with Intel Arria 10 FPGA using OpenCL standard. It provides visibilities at 1.342s and beam data at 163.84 µs rate. All the design parameters have flexibility to vary according to the user's need. Feasibility of running an off-line Correlator and Beamformer on an FPGA Accelerator card is demonstrated using antenna raw voltages. Use of Intel's FPGA SDK for OpenCL has 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 addition to these, Parallel programming techniques were used to achieve real-time processing of data upto 400MHz bandwidth.

6.1. Future Scope

- The number of antennas that this prototype is designed for is just a fraction of the actual number of antennas 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 processes plays an important role, as the overhead introduced by the Host computer itself will overpower the processing performance of the system.
- A Single device might not be able to implement a bigger set of Antenna, hence 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.

7. Acknowledgements

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. Yashwant Gupta, Sri. Ajith Kumar and Dr. Jayanta Roy for providing me this opportunity to work on this project in such a prestigious institution. Thanks to Mr. Harshavardhan Reddy, Mr. Sanjay Kudale & Mr. Kaushal Buch 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 making 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 the Department of Electronic Science & Instrumentation Science for their patience and guidance.

8. References

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

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

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

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

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

[6] 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.

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

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

[9] Intel. Intel Arria 10 Device Overview, 2018.

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

[11] 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.

[12] 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.

[13] 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.

[14] PCIe FPGA Card 385A (BittWare, Concord, New Hampshire).