DESIGNING OF COHERENT POCKET BEAMFOREMR ON FPGA

Submitted in Partial Fulfillment of the Requirements of the Degree of

Bachelor of Technology

in

Avionics Engineering

by

ANKUR VERMA (SC08B113)

SUBHAJIT MAJUMDER (SC08B103)

Department of Avionics

Indian Institute of Space Science and Technology

Thiruvananthpuram

January-April, 2012

BONAFIDE CERTIFICATE

This is to certify that this project report entitled **"Designing of Coherent Pocket Beamformer on FPGA "** submitted to **Indian Institute of Space Science and Technology, Thiruvananthapuram,** is a bonafide record of work done by **Ankur Verma** and **Subhajit Majumder** under my supervision at the Giant Metrewave Radio Telescope, NCRA-TIFR from "**9th January 2012**" to "**27th April 2012**".

Mr. Ajith Kumar
Head, Digital Backend,
GMRT, NCRA-TIFR

Dr. Yashwant Gupta
Chief Scientist,
GMRT,NCRA-TIFR

Place: GMRT, Khodad

Date: 27/04/2012

DECLARATION BY AUTHORS

This is to declare that this report has been written by us. No part of the report is plagiarized from other sources. All information included from other sources has been duly acknowledged. We aver that if any part of the report is found to be plagiarized, we are shall take full responsibility for it.

Ankur Verma

Roll No.:SC08B113

Subhajit Majumder

Roll No.:SC08B103

Place: GMRT, Khodad

Date: 27/04/2012

ACKNOWLEDGEMENT

We would like to acknowledge the immense help received from all who made this project a real learning experience.

We thank Dr. **Yashwant Gupta**, Chief Scientist, GMRT, for sparing his valuable time to help us understand various aspects of this project and for guiding us to the successful completion of the project. We express our gratitude to Mr. **Ajith Kumar**, Head, Digital Backend Systems, GMRT, for facilitating this project and also for all help in support rendered. We wholeheartedly thank Shri. Sanjay Kudale (Scientific Officer C), Mekhala Muley(Engg C), Shri. Sandeep Choudhari(Engg. D), Upendra Gokhale (Graduate Trainee) and all other members from digital backend group of GMRT for spending their quality time and for helping us to complete this project.

We also place on record our thanks to **Dr. Swarna K. Ghosh,** Centre Director NCRA and **Dr. K.S. Dasgupta**, Director IIST for giving us such a great opportunity by arranging this internship program.

We thank Dr. **Anandmayee Tej**, and our project mentor Dr. **Sheeba Rani**, for recommending us to do project at GMRT and for helping before and during the Project.

Finally, we sincerely thank all people, who have helped us during the project, directly or indirectly.

April 2012 **Ankur Verma**

Subhajit Majumder

ABSTRACT

The coherent beamforming technique helps in observing a known pulsar with higher sensitivity to get the pulsar profile in time domain and the polarization information, i.e. full stokes parameters, about the pulsar. As part of the GMRT backend upgradation system, a coherent beamformer needs to be implemented on single FPGA platform. The goal of the project is set to design the coherent Pocket Beamformer (PoBe) for R circular and L circular polarizations of 2 antennas on single FPGA platform. The design approach was to use the pocket correlator (PoCo) design for 2 antenna as a basic one and then implement a 2 input coherent PoBe as an add-on to it. The designing part has three challenges such as (i) implementing the phase correction logic for coherence addition; (ii) designing the packetization and de-packetization logics of the 10GbE packet; (iii) modify the MAC with a complex adder. The individual logics were tested and verified with two sub-designs such as, (i) 2 antennas coherent PoCo design and (ii) the 1 antenna 2 polarizations PoBe design. Finally the 4 input, i.e., R and L polarizations of 2 antennas, coherent PoBe is implemented on single FPGA with some modifications for resource utilization. The testing of the design is done with noise source and real time radio source like Pulsars.

TABLE OF CONTENTS

Chapters

FIGURE LIST OF FIGURES PAGES

LIST OF ABBREVIATIONS

GMRT = Giant Metrewave Radio Telescope

PoCo= Pocket Correlator

PoBe= Pocket Beamformer

FPGA = Field Progammable Gate Array

FFT = Fast Fourier Transform

PA = Phased Array

IA = Incoherent Array

GSB = GMRT Software Backend

ROACH = (Reconfigurable Open Architecture Computing Hardware

ADC = Analog to Digital Converter

I, Q, U, V = 4 stokes parameters

R = Right circular polarization

L = Left circular polarization

PFB= Polyphase Filter Bank

MAC = Multiplier and Accumulator

10GbE = 10 Gigabyte Ethernet

BRAM = Block RAM (Random Access Memory)

LUT = look Up Table

PMON = Pulsar Monitoring Software

FIFO = First Input First Output

RF= Radio Frequency

C2 = central antenna 2 of the GMRT

1. Introduction:

1.1. Introduction to GMRT:

The Giant Metrewave Radio Telescope (GMRT), located near Pune in India, is the world's largest array of radio telescopes at meter wavelengths. It is operated by the National Centre for Radio Astrophysics, a part of the Tata Institute of Fundamental Research, Mumbai.

The GMRT contains 30 fully steerable telescopes, each 45 meters in diameter with the reflector made of wire rope stretched between metal struts in a parabolic configuration. This configuration works fine as the telescope operates at long wavelengths (21 cm and above). Every antenna has four different receivers mounted at the focus. Each individual receiver assembly can rotate, enabling the user to select any of them for the observation. GMRT antennas operate in five frequency bands, centered at 153, 233, 327, 610, and 1420 MHz. Out of the 30 telescopes at GMRT, fourteen telescopes are randomly arranged in the central square of 1 km by 1 km in size. Rest sixteen telescopes are arranged in three arms of a nearly "Y"-shaped array each having a length of 14 km from the array centre. Therefore GMRT can act as an interferometer which uses a technique known as aperture synthesis to make images of radio sources. The maximum baseline in the array gives the telescope an angular resolution (the smallest angular scale that can be distinguished) of about 1 arc-second, at the frequency of neutral hydrogen. provide seamless coverage from 100 Mhz to 1600MHz in addition to upgrades to the mechanical and servo control systems to the antenna and an improved high speed telemetry system for controlling the antennas remotely . This needs a major upgrade to the backend electronics, two possible solutions to the backend upgrade are currently being developed – one based on multiple FPGA boards, and second on GPU cluster. Currently, the GMRT is undergoing an upgrade. As part of the upgrade, the GMRT plans to increase the bandwidth of the GMRT from the present value of 32 MHz to about 400 MHz and also plans to upgrade the digital backend from GSB (GMRT software Backend) to FPGA and GPU based backend.

1.2. Introduction to digital backend:

The digital backend is responsible for digitisation of the analog signals, combining of the signals from different antennas and storage of the final data for offline processing and analysis. The RF (Radio Frequency) signal from the antenna after going through the analog backend is converted into baseband. This signal needs to be digitized so that it could easily be worked upon using digital signal processing. The signal after being converted into digital form is processed through FX Correlator (FX : FFT followed by Multiplier) to generate cross amplitude and phase information between each pair (baseline) among the 30 antennas to give the visibility information. This data is used in imaging, continuum and many other astronomical observations. The outputs from the FFT subsystem are given to the Array Combiner for generating the Incoherent Array (IA) and Phased Array (PA) outputs for pulsar observations. The data from IA and PA are used for pulsar search and studies. Incoherent Array is the addition of voltage signal coming from an array of antennas after passing it through square law detector, so it is a addition of power spectra. Phased Array is synthesizing process of a coherent resultant beam by adding the voltages from an array of antennas.

1.3. Introduction to the project:

The Project of implementing and testing coherent POBE (Pocket Beamformer) is a part of the upgradation process of GMRT Backend system from current GSB (GMRT Software Backend). In Radio astronomy, polarimeter is a technique which is used to get the full stokes polarization parameter and the pulsar profile. It can be of two types such as, Incoherent beamforming mode and coherent beamforming mode. The coherent beamformer adds voltage signals from different antennae in phase and gives computes the basic self and cross terms of voltage signals of the two polarizations, from which the full stoke parameters can be constructed. This coherent beaformer for 2 antennae and 2 orthogonal polarizations is implemented on a single Roach-board (FPGA platform) and tested with proper pulsar source. This design later can be increased for more than 2 antennae and implemented on more than one FPGA platforms.

1.4. Significance of the project:

Pulsars are weak radio sources, and their individual pulses often do not rise above the background noise, so even with long base line it appears as a point source. Beamforming is the standard signal processing approach for its study to get its profile in higher resolution. Coherent beamformer exhibits a higher sensitivity by N times (N= no of antennae). As the voltage signals of different antennae are added in phase, the coherence beamformer can also act as a polarimeter to provide vital information on the full stokes parameters of the pulsars. So as a part upgradation process of GMRT backend, coherent beamformer needs to be implemented on FPGA. FPGA is chosen as a hardware platform for its re-configurable features and better computing resources with lesser power conversation and higher bandwidth compared to the software based solution.

Within the scope of our project, we need to design the basic hardware and its interfacing utilities and test it with real time sources. So, the 2 antennae and 2 polarizations coherent beamformer is implemented on a single Virtex-5 pro FPGA (Roach-board) to verify the functioning of the coherent beamforming.

1.5. Aim and Objectives of the project:

The aim of this project is to design and implement coherent pocket beamformer on single Roach-board (FPGA platform) for 2 antennae 2 polarizations and test the design with Pulsars to get the pulsar profile and its full stokes parameters.

The objectives of the project are:

- Design and implement the coherent beamformer for 2 antennae and 2 polarizations on single FPGA platform (ROACH-board).
- Write the scripts for the necessary interfacings of the ROACH-board with host PC.

- Simulation and implementation of design on hardware for verifying design logic.
- Verify the design using sky-test, i.e. , testing with signals from radio sources(Pulsar).

1.6. Casper:

The Center for Astronomy Signal Processing and Electronics Research (CASPER) is a global collaboration dedicated to streamlining and simplifying the design flow of radio astronomy instrumentation by promoting design reuse through the development of platform-independent, open-source hardware and software.

The CASPER tool flow is better known as the MSSGE (Matlab/Simulink/System Generator/EDK) or bee xps tool flow. It is the platform for FPGA-based CASPER development and is the interface between several design and implementation environments.

Casper design environment in GMRT that is used during the course of this project use following version of different utility

- Matlab R2008a or R2008b (v7.7.0)
- Simulink R2008b (v7.2)
- Xilinx System Generator v10.1.3.1386
- Xilinx EDK v11.5
- Xilinx ISE v11.5
- MSSGE libraries

The aim is to couple the real-time streaming performance of application-specific hardware with the design simplicity of general-purpose software. By providing parameterized, platform-independent "gateware" libraries that run on reconfigurable, modular hardware building blocks, we abstract away low-level implementation details and allow astronomers to rapidly design and deploy new instruments.

CASPER instruments use reconfigurable open-source hardware built around Xilinx FPGAs. The GMRT uses Virtex 5 SXT95 based standalone FPGA processing board also called ROACH (Reconfigurable Open Architecture Computing Hardware). The ROACH board also has the following features: [see appendix K]

- A separate PowerPC runs Linux and is used to control the board
- CX4/XAUI/10GbE Networks Interfacing Cards
- ADC2x1000-8: Dual 8-bit, 1000Msps (or single 8-bit 2000Msps), Atmel/e2v AT84AD001B ADC

2. Theoretical concepts:

2.1. Interferometry and correlator:

Interferometry is a technique in which waves are superimposed in such a way that one can analyze wave property from residual phase and spectrum.

Interferometry makes use of the principle of superposition to combine waves in a way that will cause the result of their combination to have some meaningful pattern that is diagnostic of the original state of the waves. This works because when two waves with the same frequency combine, the resulting pattern is determined by the phase difference between the two waves—waves that are in phase will undergo constructive interference while waves that are out of phase will undergo destructive interference.

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 convert the electric field into voltages. 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.

The cross correlation between two signals $s_1(t)$ and $s_2(t)$

$$R_c(\tau) =< s_1(t)\ s_2(t+\tau) >$$

Where τ the time delay between the two signals and angle brackets is indicates averaging in time. According to Wiener-Khinchin theorem which says

The power spectral density (PSD) of a stationary stochastic process is defined to be the FT of its auto-correlation function that is if $R_c(\tau) =< s_1(t)s_2(t+\tau) >$ then power spectral density function $S_c(f)$ is

$$S_c(f) = \int_{-\infty}^{\infty} R_c(\tau)e^{-j2\pi f\tau}\ d\tau$$

From the property of Fourier transform we have

$$R_c(0) = < s_1(t)s_2(t) > = \int_{-\infty}^{\infty} S_c(f)\, df$$

This approach is used in the all the design in the course of this work.

2.2. Beamforming- coherent and incoherent;

Pulsars are the weak radio sources, so their individual pulses often do not rise above the background noise level. Beamforming is the basic technique used for their studies. Beamforming is a signal processing technique 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 antennae signals can be added coherently or incoherently.

Incoherent Beamforming:

- In incoherent beamfomrer, the voltage signals are firstly converted into power spectra. Then the power signals from the N dishes are combined to give the single incoherent beam. As the power spectra of the signals are added, the phase information is lost and no need of phase corrections.
- Root of N improvement in sensitivity.
- Beamwidth of single antenna.
- Application in large scale pulsar search
- The mathematical representation of the incohernt beamformer:

$$B_i = (V_1^2 + V_2^2)$$

Coherent beamformer:

- Voltage signals from the N dishes are combined to give the single coherent beam. As the voltages are added, it should be in phase with each other to get the resultant coherent signal referred as beam.
- N times improvement in sensitivity
- Beamwidth becomes narrower than the single antenna by nearly 1/N times.

- Application in studies individual known pulsars with its polarimetry studies.
- The mathematical representation of the coherent beamformer

$$B_i = (V_1 + V_2)^2$$

2.3. Pulsar observations requirements:

Pulsar is a very highly magnetized rotating neutron star, which emits pulses with a very high precision period. The compact nature of its emission makes it a point source even for largest baseline on the earth. So for Pulsar observation, the phase array mode is used with some requirements.

In phase array mode, higher sensitivity improves measurement. The coherence phase array mode helps to get the polarimetry outputs for full stokes parameters.

The higher bandwidth benefits observation. To avoid larger Smearing of the pulses produced by differential dispersion delay of frequencies across the band (due to propagation of the pulsar signal through the interstellar medium), the spectral resolution over the bandwidth should be higher.

Time resolution is a major factor. The pulses have very small duty cycle. So higher time resolution requires to get detail information about the pulsar profile.

Radiation from pulsars has been shown to be highly polarized. Significant amount of circular polarization and linear polarization are also seen frequently. The study of these polarization characteristics is very important for understanding the emission mechanism of pulsars.

2.4. Polarimetry:

In radio astronomy, the polarimetry technique is used for interpreting the polarization features of the radio sources. The polarization of the sources can be characterized by the full stoke parameters which calculate the linear polarization, circular polarization and intensity of the radio sources. Pulsar signal shows better polarized information. Generally for a circular polarized signal, the full stoke parameters are defined as,

$$I = |E_l|^2 + |E_r|^2$$

$$Q = 2Re(E_l{}^* E_r)$$

$$U = -2Im(E_l{}^* E_r)$$

$$V = |E_l|^2 - |E_r|^2$$

,where the E_l and E_r are the voltage signals from the Right circular and Left circular polarizations. The I and V terms provide information about the total intensity and circular polarization of the source. The Q, U gives information about the linear polarization. Here, in GMRT antennae, all the antennae have two circular polarizations R (right circular and polarization) and L (left circular polarization).

3. Description of the project work:

3.1. Pocket Correlato (PoCo) Design:

The ROACH based PoCo design has been tested at the GMRT and is the digital-backend for an interferometer. A general system design (PoCo and PoBe) is given below:

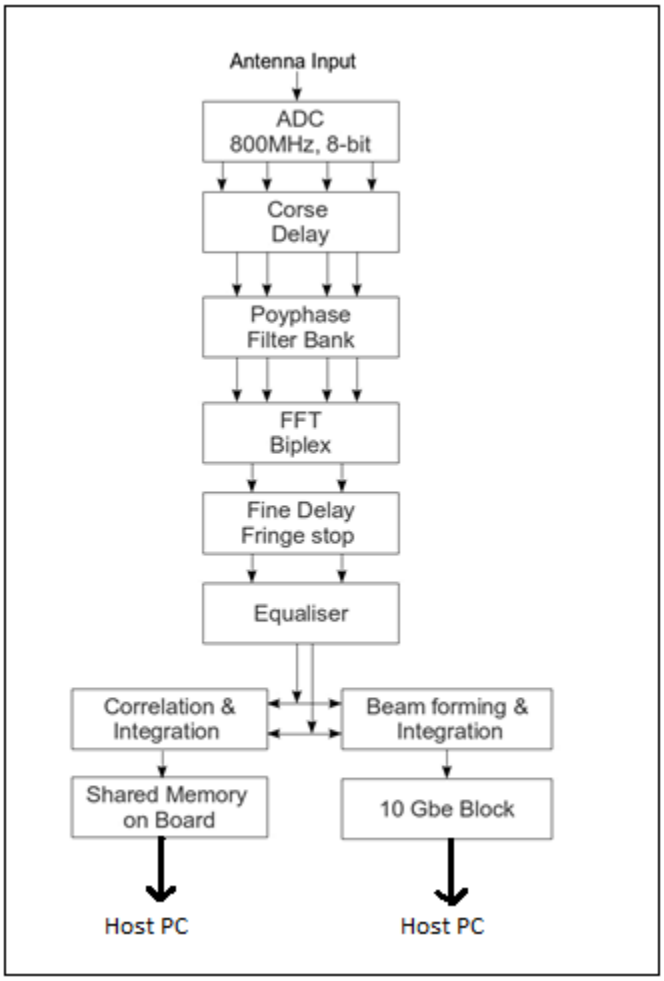


Figure 3.1.: PoCo design flow

Each block mentioned Figure (3.1) is explained in brief below:

1. ADC: The ADCs interfaced to the ROACH board are ADC2x1000-8. They normally operate at 800MHzand give an 8 bit output through 4 channels each operating at 200MHz. This is done as the FPGA operates at 200MHz. In our design, the ADC is running at 400MHz clock frequency.

2. Delay: The radio sources in the sky are in motion over the sky. This differential change in position of the radio sources with respect to the antennas gives some delays. Other than that, the propagation delays from the antennas to the receiver are also considered. The whole delay that need to corrected for proper phasing is divided into two part

    a. Integral multiple of clock is implemented in course delay block.

    b. Fractional delay is implemented in fine delay fringe stop block.

    The coarse delay is usually taken to be 0 for simulation purposes but is taken into consideration during sky tests. In case of the coherent PoBe, we account for delay for proper phasing the antennas. This course delay block can give delay that is integral multiple of clock. The data rate at the output will be 4 channels of 8 bits at 200MHz.

3. PFB (Polyphase Filter Block) block: The polyphase filter bank implements a hamming window. The PFB is used to reduce spectral leakage and to increase signal to noise ratio. The data rate at the output will be 4 channels each of 18 bits at 200MHz.

4. FFT (Fast Fourier Transform): The FFT block used is FFT Biplex Real 4x (real-sampled biplex FFT). This block computes the real-sampled Fast Fourier Transform using the biplex FFT algorithm to use a complex core to transform two real streams. The data rate of operation at FFT output is 36 bits each at 400MHz. One of the streams gives even channels while the other gives odd channels. Each channel consists of an 18 (fix 18_17 format)bit real part and an 18 bit imaginary part.

5. The FFT operation is performed during the period of a synchronization pulse which is of 0.89s at 200MHz bandwidth. This period is equal to 2^{19} FFT cycles. We calculate this by:

$$\frac{2^{27} \times 4}{2^{10}}$$

    Where 2^27 is the data transferred during a synchronization pulse, 2^10 is the 1K point FFT and 4 parallel inputs are the 4 input channels in to the FFT block. In addition to this, a 1K point FFT requires 256 clock cycle.

6. Fine delay fringe stop Block: Fringe delay appears due to the down conversion of the RF signal to the baseband signal. The delay values are compensated for baseband signal but this give a drift in phase for RF signal. To compensate this drift in phase fringe stop is used. Using fine delay fringe stop block we can apply maximum 1 clock delay.

   For the fringe correction Let the sync period is of 2^27 clks and number of FFT points be 2^10 then the maximum number of FFT cycles for incrementing the fringe phase by amount of resolution set for the Sine-Cos LUT = 2^27 / 2^10 = 2^17.

   It means that minimum rate of incrementing fringe phase by 0.02197 degrees is after 2^17 FFT cycles.

7. Equalizer block: This block scale down the amplitude of incoming from the channels by a given factor to avoid the over flow during correlation and integration. The scaling factor depends on the integration time and power level of the signal. This block casts the 36 bits input data into 8 bits data so that the bit growth during accumulation does not overflow 32 bits.

8. Correlation and Integration block: The correlator block in the design performs the product of voltages of two channels. The output of the equalizer is 8 bits. This is fed to an integrator which gives a 32 bit output. The typical integration time is in second. But it can vary in accordance of data rate and noise level. The output is all the self-correlated and cross-correlated pairs. This correlated data is transferred into shared BRAM so that host PC can read correlated data and store it on a disk for further use. The PoCo design normally uses 524288 FFT cycles of integration time.

9. Integration time = (No. of FFT cycle)*(No of FFT point)/(clock frequency)

10. Data rate= packet size/integration time.

3.2. Coherent PoBe as add-on to PoCo:

A coherent PoBe design can be built on the existing PoCo design as an add-on to it. There are observational requirements where both poco and pobe are simultaneously needed. The 3 basic modifications of PoCo required for the coherent PoBe design are:

(i) Add the phase correction logic to compensate the ionospheric and electronic phase mismatch between the two antennas.

(ii) Design the coherent adder and MAC (multiplier and accumulator) to generate the beamforming outputs and reduce the integration time for PoBe. The PoBe design needs to have a lower integration time to get hogher resolution over the pulse period and also to increase the data rate.

(iii) Implement logic for packetization of the 10GbE packets and depacketize the data on receiving end of the Ethernet connection.

3.3. PoBe design 1 antenna and R-L polarizations:

The 1 antennae 2 polarizations PoBe design is designed as an add-on of PoCo. So for designing the pobe a dedicated set of MACs are added in the PoCo design. The inputs of the PoBe are coming from the equalizer blocks which are exactly same as PoCo. There are few MACs in PoCo and PoBe design which perform the same functiona but we cannot use shared MAC for PoCo and PoBe because of the difference in the integration time. The MACs in the next stages are designed to serve the purpose of complex multiplication and integration. It takes two complex signals and multiplies and forward to accumulator. Accumulator holds the values until next data set is arrive. It adds new data set to old value and replaces old values by the sum in respective memory. This process of integration continues for the manually given integration time which can be change during run time to achieve proper data rate and is controlled by new_acc signal (see fig 3.2). When the integration gets over, accumulator will forward these data value to RAM which is accessed by packetized design.

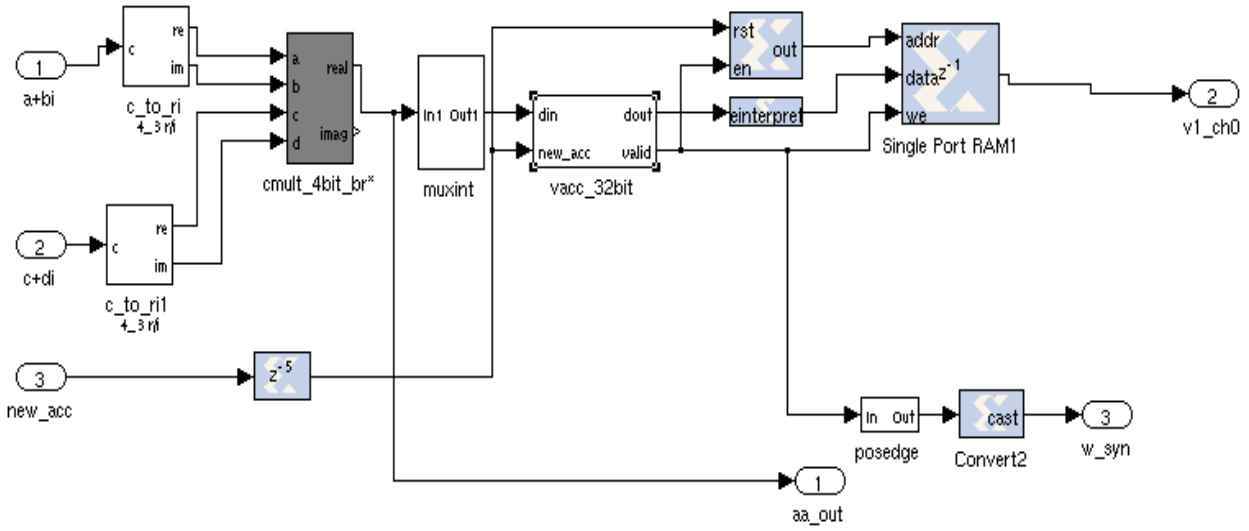


Figure 3.2.: designing of Multiplier and Accumulator (MAC) of PoBe

Since our design goal is to calculate beamformer output and few parameters which are used for calculating four stokes parameters of a source i.e. pulsar. For the two antenna two polarization design following are the four outputs (R1 + R2)(R1 + R2)*, (L1 + L2)(L1 + L2)*, Real part of (R1 + R2)(L1 + L2)*, Imaginary part of (R1 + R2)(L1 + L2)*, where RX is right polarized signal from antenna X and LX is left polarized signal from antenna X. Here for 1 antenna R and L polarization the inputs are only two signals from R and L of same antenna. So the two signals are always in-phased. The addition operation to form beam is also not required. So the output of the design becomes:

- RR*
- LL*
- Real {RL*}
- Imaginary {RL*}

In the design after FFT every signal is in two parallel channels namely EVEN and ODD. So for calculating above mentioned parameters we need four MACs which gives the real output of respective signals and two MACs which gives both real and imaginary output of respective channels.

Then these data sets are transferred to packetization logic. At the first stage of packetization logic, even and odd channels data is concatenated. Then, all the four data sets are concatenated and multiplexed to generate a 10GbE packet of 8192 Kbytes size. The UDP packet format for 256 channel design is as shown in following figure.

In the final design of 256 channels UDP packets are of 2K size with each data of 64 bits and depth of 256. Each line of 64 bits carries four data values as shown in figure. Corresponding depacketization logic is implemented in 'C' program [see appendix J]. After depacketization the data structure is in fig 3.3.

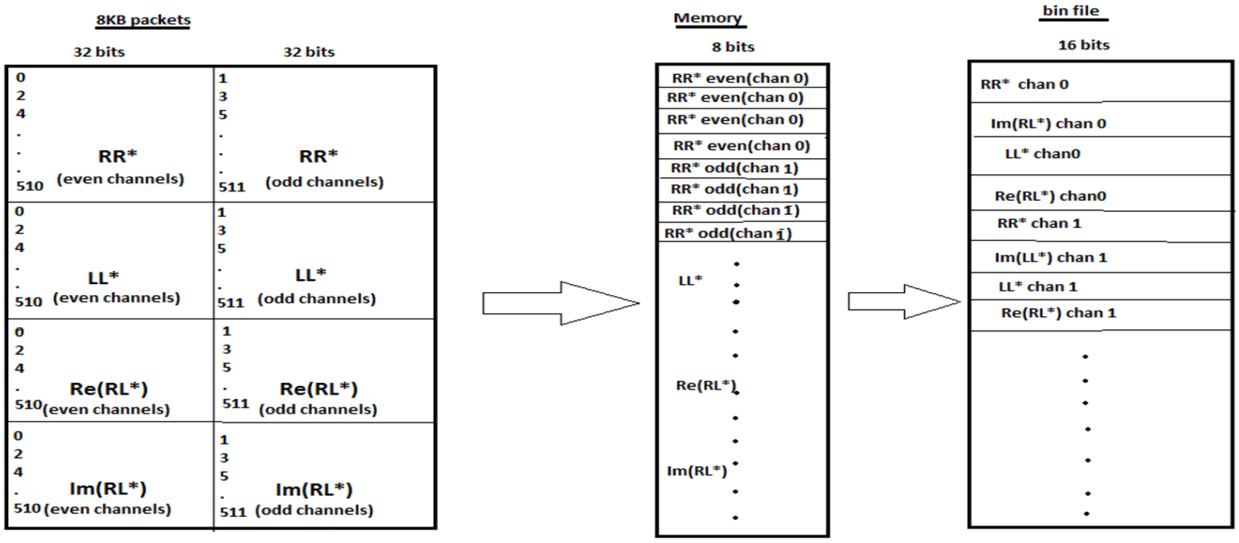


Figure 3.3.: The Packetization and Depacketization logic for 10GbE packets of PoBe

This data format is compatible for offline data Pulsar Monitoring Software (PMON). The packetization design is shown in appendix.

3.4. Coherent PoCo design for 2 antennae 1 polarization:

A phased array is an array of antennas in which the relative phases of the respective signals feeding the antennas are varied in such a way that the effective radiation pattern of the array is reinforced in a desired direction and suppressed in undesired directions.

In all the previous design of PoCo and Incoherent beam former, on-board run-time phase correction logic was not included. So this work aims to design and implement the phase correction logic in the beamformer design.

This module is designed by adding and modifying addition logic in fine delay fringe stop block. Fine delay fringe stop block performs the fine delay correction along with the fringe stop. It takes the simultaneous stream of data from the FFT module and has a run time programmable fine delay correction along with the fringe stopping. This block is specifically compatible with the "fft_wideband_real" module. For other FFT modules changes will be required in this block depending upon output of the FFT module used in the design.

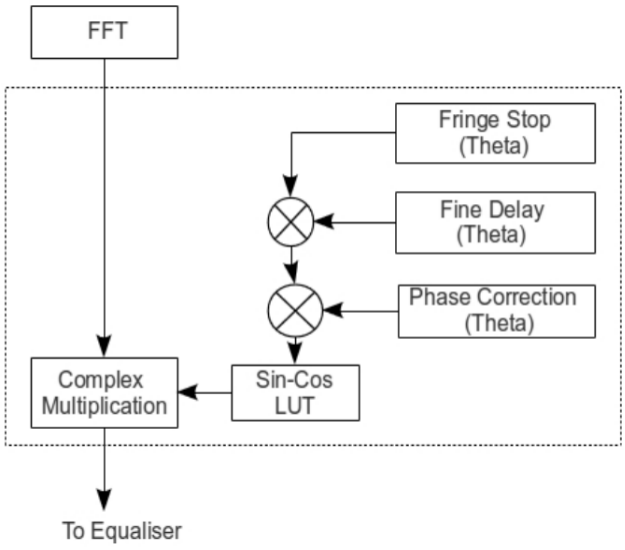


Figure 3.4.: Flow of phase correction logic

In the figure 3.4, data flow of fine delay fringe stop block is explained. Input of fine delay fringe stop block is coming from FFT block in previous stage of the design. The values for fringe stop and fine delay is calculated offline for the test setup depending upon the RF to baseband frequency and position of source and antenna. Then load these values using python script into the design during run time. So for the phase correction utility additional logic is added as shown in figure 3.4.

Implementation:

Into the fine delay fringe stop block, data is coming in two parallel fft channels i.e. Even channel and odd channel. So to store the phase correction values for even channels and odd channels, two

BRAMs are added, whose depth is equal to half of the number of fft channels. These BRAMs will store the phase values for even and odd channels separatelywhich are written in these memory during run time (Asynchronous with global syn) and read the phase value and multiply to each channel (synchronous to global sync). So for the switching this logic in two modes back and forth, a two input multiplexer used for selecting the address line and write_en signal is select control for this multiplexer. When write_en is high, the address line that drives both BRAMs is coming from external peripheral (python script running on host PC) and both the BRAMs are in writing mode. Now we start giving data to BRAM and incrementing the address values accordingly. When writing is over, write_en goes down and mux will switch the address lines. Now the address is coming from a counter of bits=fft_chan/2. This counter runs synchronously with the whole design that means the counter output (address) is always equal to channel numbers of data values coming for phase correction.

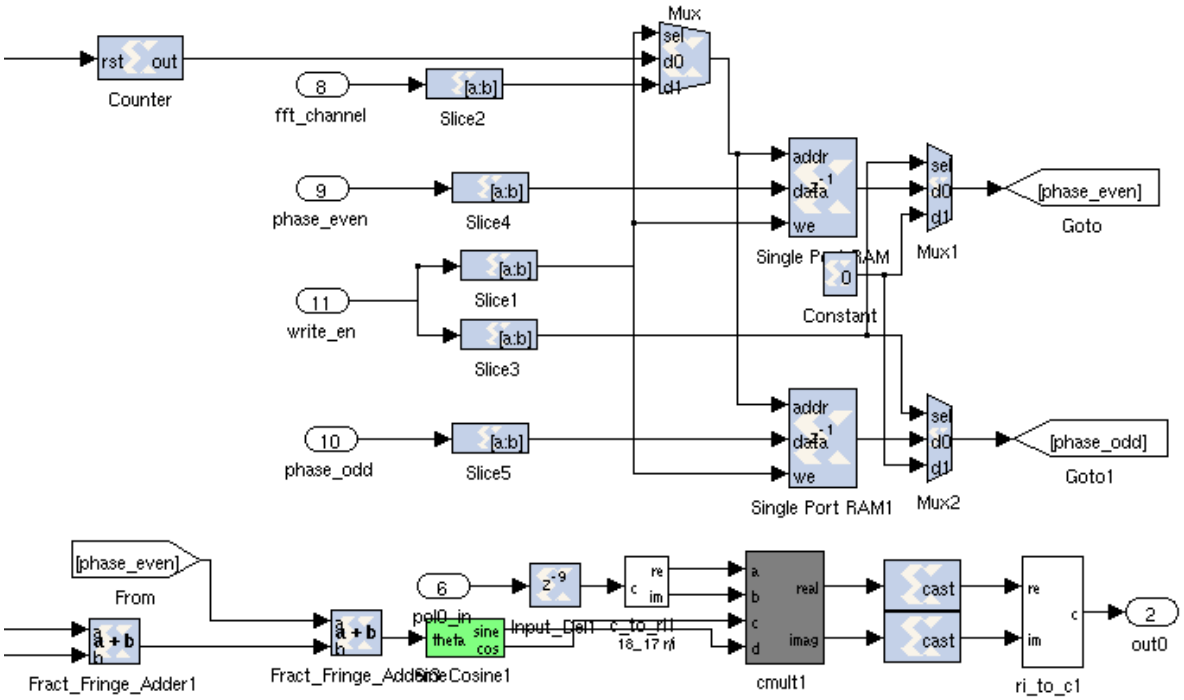


Figure 3.5.: Designing of phase correction logic using Xilinx blocksets

After reading the from the BRAM , the phase values will be added with addition theta values (sum of fine delay and fringe stop) . After this stage the sum of all three theta values fed to Sin-Cos LUT.

Corresponding sine and cosine values will be multiplied with particular FFT channel. This multiplication is a complex multiplication. Resolution of this phase correction logic is limited by depth of Sin-Cos LUT. During the course of this whole project, depth of Sin-Cos LUT is taken as 2^14 that is we can correct phase up to 0.0219 degrees.

All the inputs of this module are driven by python script running on host PC. The flow of the logic is as follows

- A python script will read the phase values from the shared memory holding correlated data at PoCo output for a certain period of time and write it on the disk for the later use.
- Another Python script is written for reading the correlated data from the disk. After averaging the all sets of phase data, upload these phase values with proper sign into BRAM. At the same time, save these average value in a separate file. So if after one iteration the phase offset is still there we can go for number of iteration.
- The BRAM which are written by external peripheral (python script) holding the phase correction values are driven in such a way that every FFT channel will be multiplied by the proper phase value. An addition counter synchronized with global *syn* signal. So if this logic will some enters into asynchronous mode, the next syn will reset the whole logic.

3.5. 2 antennae and 2 polarizations coherent PoBe Design approach:

3.5.1. Working principle:

The design approach is to combine the previous to design concept with a four input PoCo design and make a 2 antenna and 2 polarizations coherent PoBe. The Coherent Pocket Beamformer mainly adds the in-phased voltage signals from different antennae. The input to the design is R-L polarizations of 2 antennae.

- R polarization of the antenna 1(R1)
- L polarization of the antenna 1 (L1)
- R polarization of the antenna 2(R2)
- L polarization of the antenna 2 (L2)

Then as in PoBe the same polarization of each antenna needs to be added coherently to each other to form the coherent beam, the phase between the cross correlation of the same polarization of different antennae needs to be monitored and corrected to a zero value using the phase information got from PoCo. So the output of the PoCo part of the design is:

- Cross correlated amplitude of R1R2
- Cross correlated amplitude of L1L2
- Cross correlated phase of L1L2
- Cross correlated phase of L1L2

The output of the PoBe requires for the plotting the pulsar profile and the calculating the 4 stokes parameters (I, Q, U, V). So the required coherent outputs are:

- (R1+R2).(R1+R2)*
- (L1+L2).(L1+L2)*
- Real {(R1+R2).(L1+L2)*}
- Imaginary {(R1+R2).(L1+L2)*}

3.5.2. Design approach:

- Each Roach baord(FPGA) has two ADC cards for inputs. So the R and L polarizations of antenna 1 are given to the I and Q channel of the ADC0. The other two inputs from the antenna 2 are fed to the ADC1.
- The common path between PoCo and PoBe is modified from 2 inputs design to a 4 inputs design. So the coarse delay block, PFB (poly phase filter block), FFTs, Fine delay and phase correction blocks, Equalizer block are added to all the four paths from the inputs to the PoCo and PoBe.

- The PoBe part has an extra complex addition block, which will coherently add the voltages of same polarization of the two antennae.
- The packetization and depacketization remain same as the earlier design.

3.5.3. Modification of the design for single FPGA platform:

- The final design is not possible to be compiled within one FPGA (virtex5) platform because of short of resources (see appendix). The requirements for the Block RAMs and memories are overshooted by 240% compared to the resources available in one FPGA virtex5. As the project's final aim is to design the coherent PoBe for 2 antennae 2 polarizations on single Roach-board (FPGA), the following has been done to achieve the goal:[see appendix L]
- The max coarse delay, which can be compensated, is reduced from 128us to the 20.48 us, which will cover most of the central antennas.
- The coarse delay, fine delay and phase correction block is removed from the data path of the antenna 2. This restricts the design to be worked for only the reference antenna C2 as the 2^{nd} antenna. All the delay calculation and fringe and phase compensation is done with respect to the C2 antenna. This reduces a lot of resources in terms of memories (see appendix).
- The FFT size is reduced from 1024 points to the 512 points. This reduces the spectral channel from 512 to 256 and thus reduces the memory size requirement of all the consequent stages.
- The PoCo stage can be used to show all the cross and self correlation output of all the 2 polarizations of 2 antennas signals. But because of shortage of resource it is used to show only the cross correlation output between the same polarizations of different antennas.
- The corresponding changes are done in the interfacing utilities and python scripts.

3.5.4. Design specification the modified coherent PoBe on single FPGA:

- Bandwidth: 200 MHz
- Max coarse delay compensated: 20.48 us
- 2nd antenna should be the reference antenna C2.
- FFT size = 512 ponits
- Spectral channels = 256
- Packet size of the 10 GbE packet = 4096 bytes
- PoCo integration time: 524288 FFT cycle= 67.108864 sec
- 1 FFT cycle = 128 clock cycle
- PoBe integration time: 2048 FFT cycle= 2.61244 msec

3.6. Post processing and interfacing utilities:

- After compilation and downloading of the BOF (configurable file) file on the ROACH-board to configure it, some python scripts are required to act as interfaces between the roach-board and the external computer. These python script initialize the software registers for runtime configuration of the Roach-board and also can read the software registers and shared memories for reading data.
- A python is written to initialize all the software registers. These registers can set the parameter values of Integration time, power scaling factor, Ethernet IP address, IPport etc. [see appendix]
- The python script for plotting the PoCo output is written to read the shared memories at the PoCo and plot the amplitude and phase of the cross correlation between same polarizations of different antennae. It also dumps the phase values of channels into a file, which is used for phase correction.
- The phase correction python script is written. It reads the phase value from the above mentioned dumping file and compensates the phase values through software registers to get a zero phase value.

- Depacketization logic is also required to depacketize the binary file stored from the 10GbE packets to make the data compatible for post processing. Capturing of the 10GbE packets is done using Gulp utility. This depacketization is done using a C program.
- Finally to plot the pulsar profile, we need a utility called PMON (Pulsar Monitoring Program). The plotting of the full stokes parameter is also done by a separate utility developed by the GMRT.

The list of post processing utilities used is:

1. PYTHON Scripting to configure the ROACH board for the design
2. WIRESHARK: checking data transfer
3. GULP: data capturing utility
4. PSR_MON: for de-dispersion and folding to plot the pulsar profile
5. GMRT Pulsar polarization software for calculating the 4 stroke parameters.

The list of python scripts and C program required for interfacing and configuration:

1) Python script [see appendix F] to the configure roach-board after downloading the BOF to the board.
2) Python scripts [see appendix G] to read the shared memory and plot the cross correlation output of the PoCo and it also dumps the phase information to a file.
3) The python script [see appendix H] for updating the phase correction values.
4) The python scrip for updating the coarse and fine delay and fringe stop values.
5) The C program [see appendix I] to depacketize the 10GbE packets to make it compatible for PMON.

4. Testing of the designs and results:

4.1. Testing of the 2 antennae and 1 polarization coherent PoCo design:

Tha main aim of this design is to implement the phase correction logic in PoCo. So, simulation has been done for the memory writing part and phase value dumping from the PoCo outputs and then fed it back to the phase correction logic is verified using noise source.

Simulation result:

Specification:

- 512 FFT channels
- 512 clock cycle syn period
- BRAMs initialized with zero except first values that is 10
- Tested against the theta_fract(fine delay) that is Ramp with 1 for whole channel for simulation.

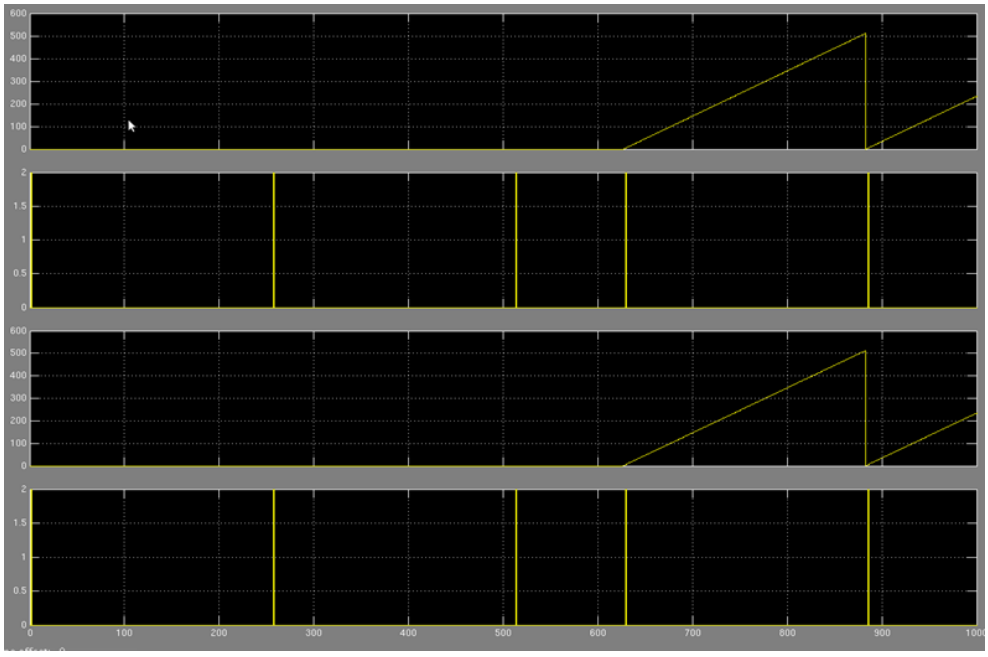


Figure 4.1.: Simulation result of synchronous memory writing

Inferences:

- After first sync pulse (at 512) , first value of both ramps and first value from memory location coincides and infer to first even and odd channel's fine delay and phase correction coming together that means reading BRAM for proper channel works fine.
- At 511 clock cycle memory pointer was at some location. When sync pulse arrives, memory pointer is set back to zero. That infers the proper working of reset logic.

Testing with Noise source:

Test setup:

- Noise power -90dbm/ Hz & 200MHz bandwidth
- RF filter 100 MHz
- 1-in-2-out RF power divider
- ADC clock 625mVpp @400 MHz
- Integration time for PoCo 524288 FFT cycle (1 FFT cycle = 256 clock cycle)
- Displaying Self- correlation for both input and cross correlation and last subplot shows cross phase.

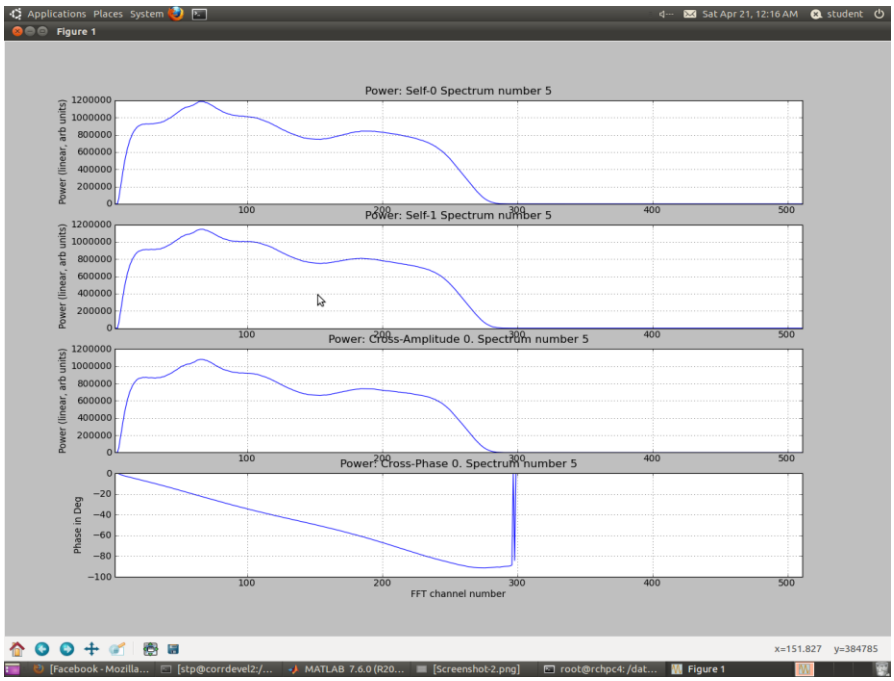


Figure 4.2: Plot of the self and cross correlation of PoCo without phase correction

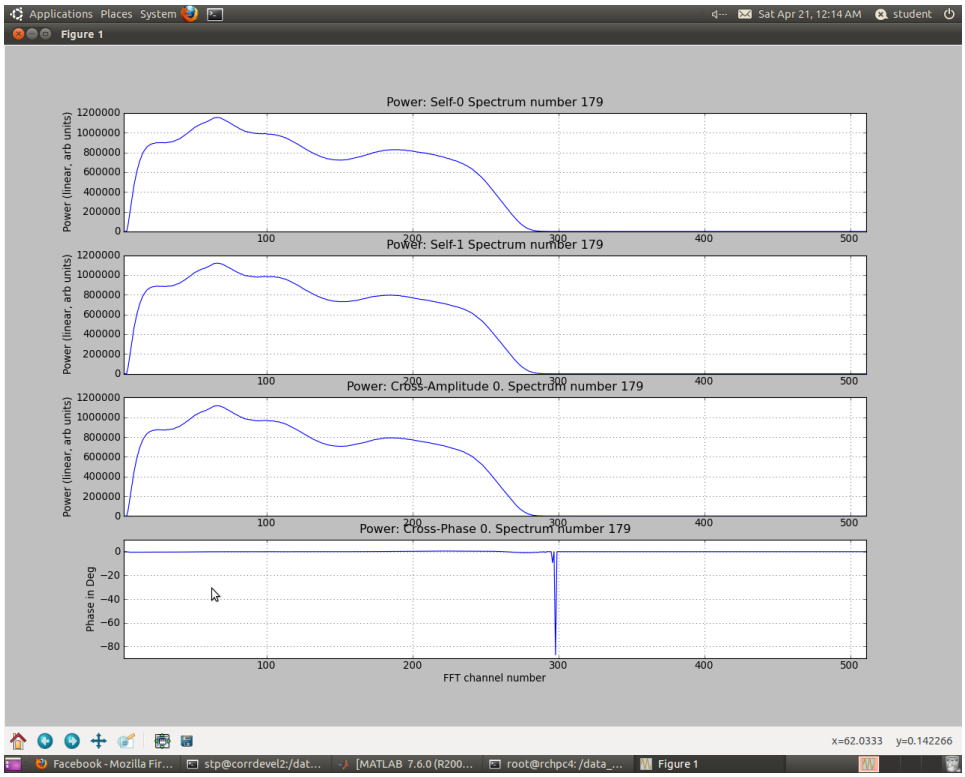


Figure 4.3: Plot of the self and cross correlation of PoCo with phase correction

Inferences:

- In first figure 4.2, Cross phase is ramp varying from 0 to -90 degrees over 0-300 FFT channels. After reading phase values in one shot and updating back to the PoCo design, cross phase values are Zero as shown in fig (4.3).
- Hardware implementation and verification of phase update in two antenna single polarization PoCo.
- Python Scripts for reading and uploading phase values works properly.

4.2. PoBe design for 1 antenna and R-L polarizations:

1 antenna and R-L polarizations PoBe is designed to check the working of the MAC blocks for PoBe and packetization of the 10GbE packets and depacketization of the final data (RR*, LL*, Real{RL*}, Imaginary{RL*}) to make it compatible for further processing. So, simulation, hardware testing with noise source and Pulsar signal are done.

Simulation result:

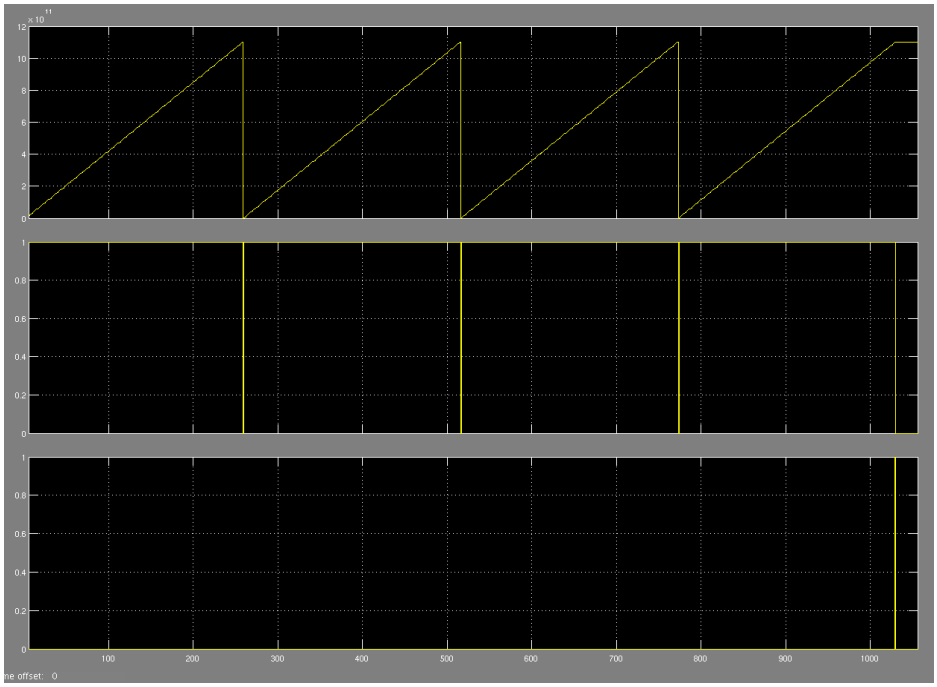


Figure 4.4.: Simulation result for packetization of the 10GbE packet:

Inference:

- This is a simulation result of the packetization of the 10GbE packet.
- The input the packetization logic is a free running counter.
- The 1st graph shows the the ouput data after multiplexing the 4 FIFOs. Each ram function is the individual output of each FIFO.
- The 2nd graph shows the data transmission valid signal which is fed to Tx_valid signal of the 10GbE NIC.
- The 3rd signal shows the end of frame signal. After this signal goes from 1 to 0, the 10GbE NIC acknowledge it as a end of one packet.

Hardware testing with Noise source:

Setup:

- Two the verify the proper bandshape at the PoBe output, a noise source signal filtered using 100 MHz lowpass filter is passed through a 2 way power divider and connected to the2 inputs of the ADC0. The following instruments are used.
- Noise source generator: -18.45dBm @each input of ADC
- Clock: sinusoidal wave @400MHz, 632mVpp
- 100 MHz RF filter
- 2 way Power Divider

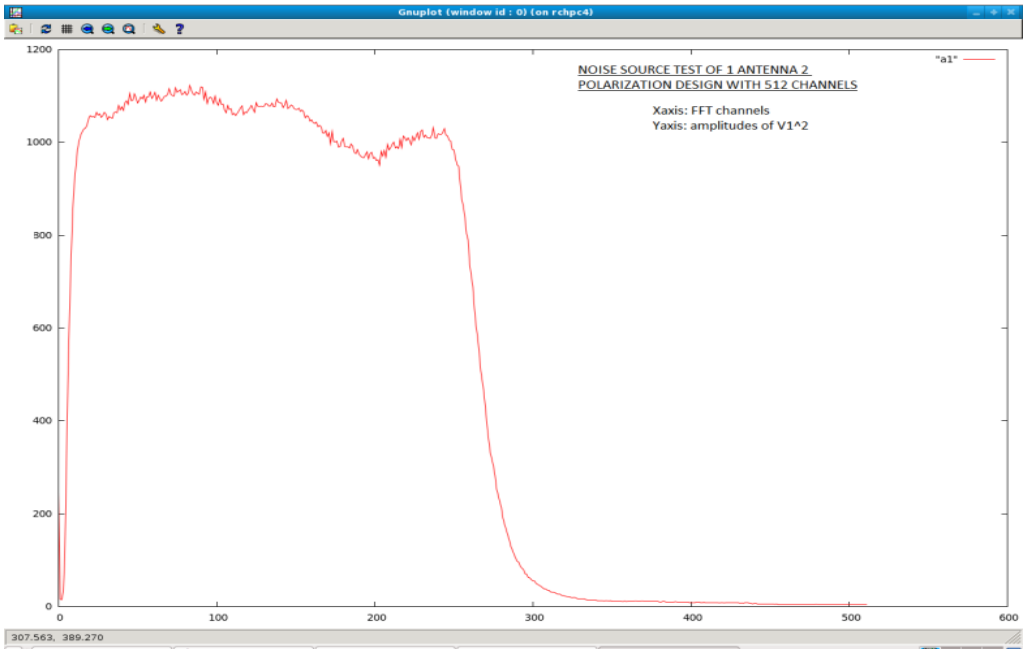


Figure 4.5.: Bandshape of noise source signal filtered by lowpass 100 MHz filter

Inference:

- shows Low pass filter Profile
- Cut off at 256 channel (corresponds to 100 MHz)
- Total channels: 512
- Scaling: 512

- Integration time: 20.97152 msec
- Data rate: 3.162 Mbps (Ref. Wireshark)

Hardware testing with Pulsar B0329+54:

Setup:

- Pulsar: B0329+54 (period: 714.578196 msec)
- Date: 11.03.2012
- Time: 15:21:00
- Data acquisition: 10 mins
- Antenna: C02
- Feed: 610 MHz
- 1st LO(local oscillator): 540MHz
- Baseband oscillator: 51MHz
- Power level at inputs: -14.32dBm @100MHz BW
- Bandwidth: 200MHz
- Max channels: 512
- Integration time: 20.97152msec

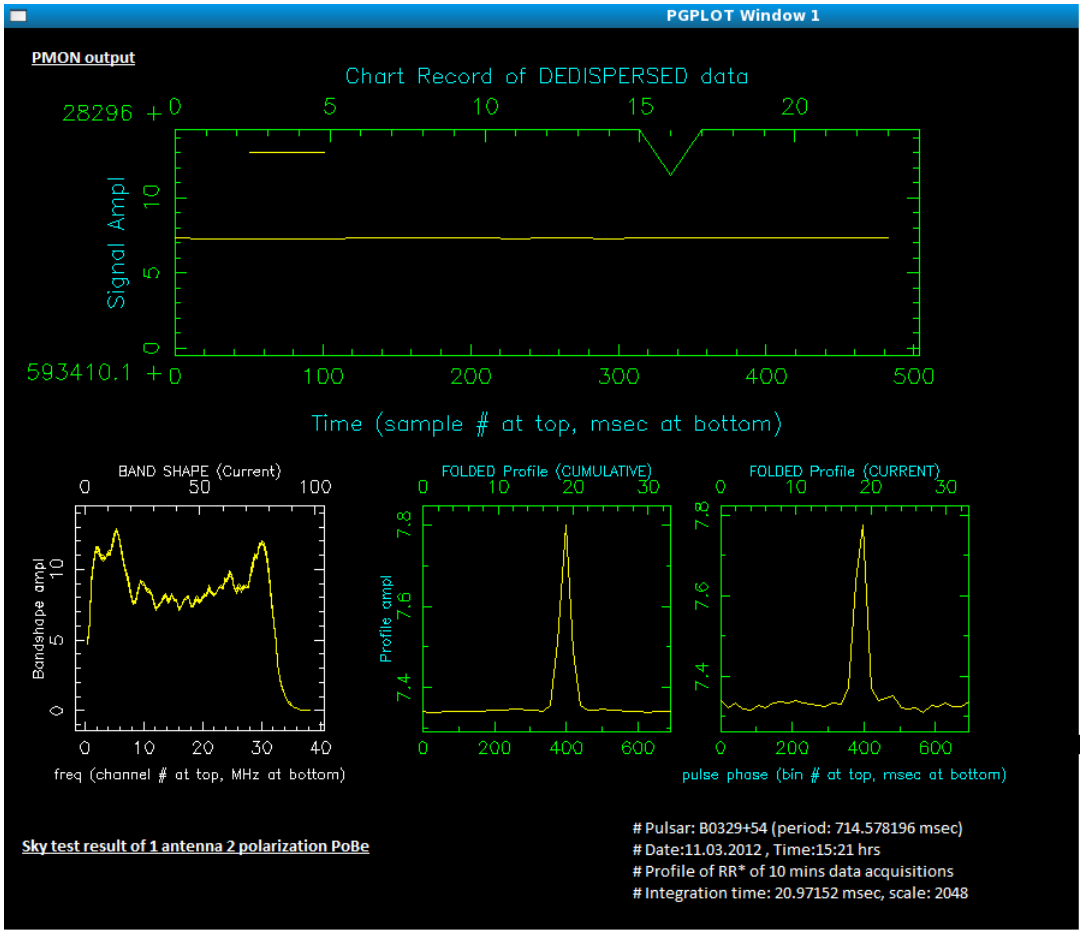


Figure 4.6.: PMON output of RR* profile of the pulsar

Inferences:

- The result is showing the profile of RR* parameter of pulsar B0329+54 for 10 minutes of data acquisitions.
- The bottom leftmost plot is showing the bandshape of the received signal over frequency.
- The bottom middle plot is the final pulsar profile in time domain after folding over 3 seconds.
- The bottom rightmost plot is the current de-dispersed profile of pulsar.
- The antenna feed used is 610 MHz of C2 GMRT antenna with baseband oscillator of 51 MHz.
- Bandwidth: 200MHz
- Maximum channels: 512
- The profile resolution achieved is 35 points, which needs to be improved.

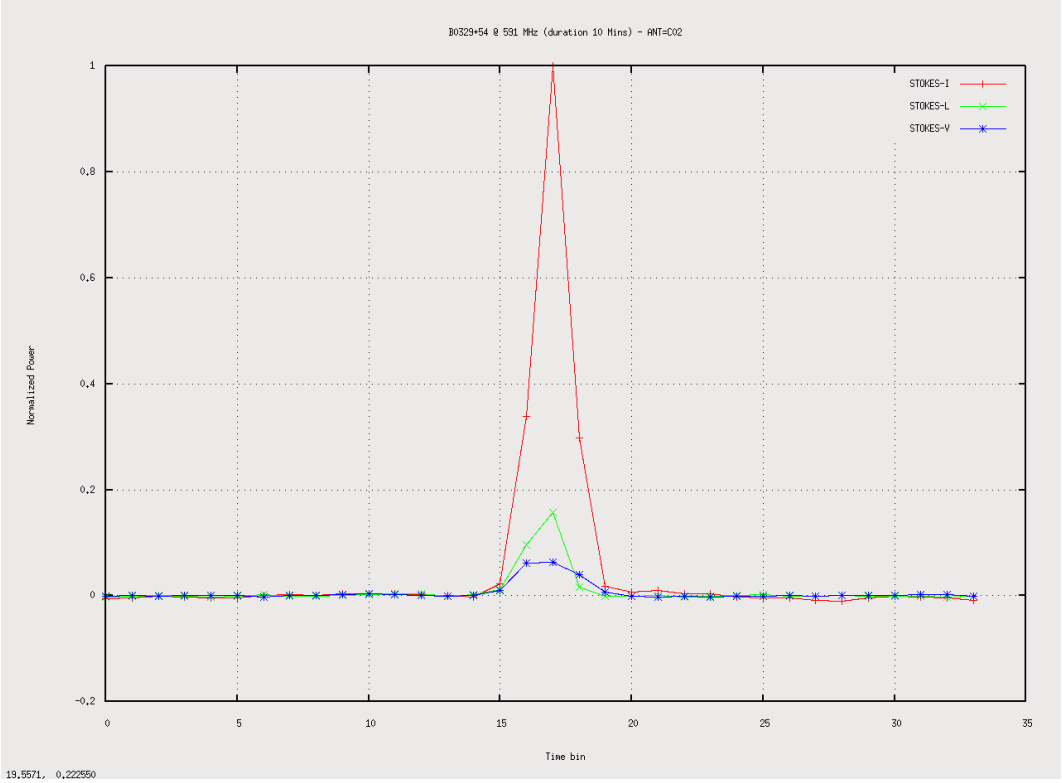


Figure 4.7.: Plot of the 4 stokes parameters for 1 antenna 2 polarizations PoBe

Inferences:

- The profile is verified.
- The red color plot is of the intensity of the Pulsar. The green one and the blue one shows the linear polarization and circular polarization respectively.

4.3. 2 antennas and 2 polarizations coherent PoBe :

Hardware Testing with Noise source:

Testing setup:

- Noise source generator: -95dBm/Hz@200MHz
- 100 MHz lowpass filter
- 1-in and 4-out power divider
- Clock: 400 MHz

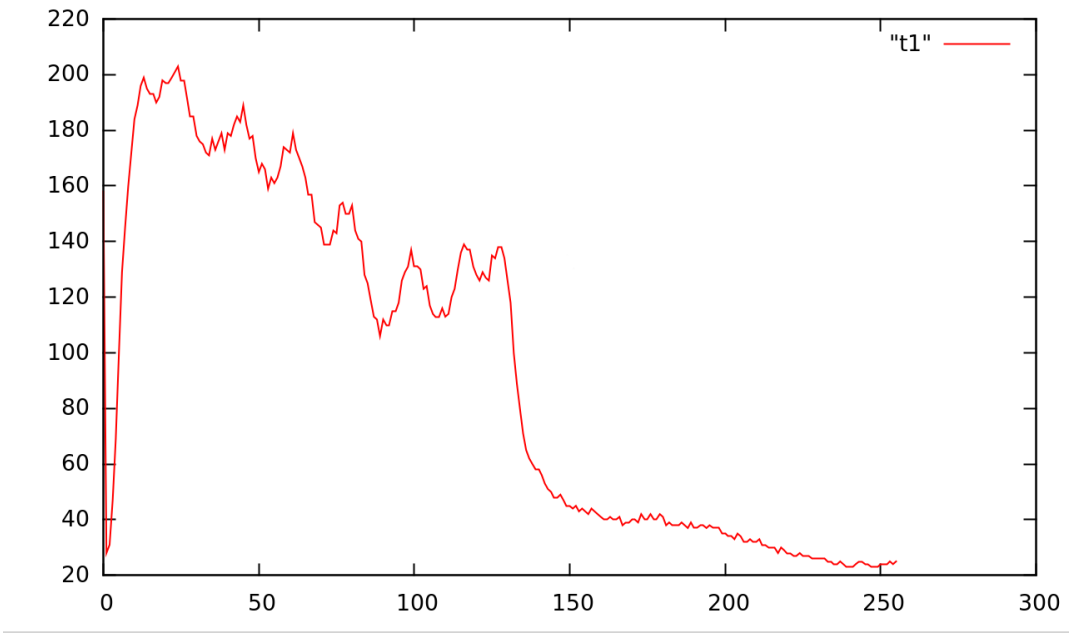


Figure 4.8.: Plot of the bandshape with noise source

Inference:

- This plot shows the frequency spectrum over the 256 channels with noise source signal filtered by low pass filter of 100MH.
- The X axis is the frequency channels and the Y axis is the magnitude with arbitrary units.

Hardware testing setup with signals from GMRT antennae:

Testing setup:

- Pulsar: B0329+54 (period: 714.578196 msec)
- Calibrator source for phase correction: 3C48
- Date: 26.04.2012
- Time: 16:00:00
- Data acquisition: 10 mins
- Antennas: C02,C03

- Feed: 610 MHz with bandwidth 32 MHz
- 1st LO(local oscillator): 680MHz
- Baseband oscillator: 51MHz
- Power level at inputs: -20dBm @200MHz BW
- Bandwidth: 200MHz
- Max channels: 256
- PoBe Integration time: 2.61244 msec
- PoCo Integration time: 67.10884 sec

Testing procedure:

- The BOF file generated from the simulink model [see appendix] is downloaded to the ROACH Borad.
- The python scripts [see appendix] is used to initialize the Board.
- A known strong radio source 3C48 is taken for calibrating the phase error between two antennas.
- The coarse delay, fine delay and fringe stop correction is done using a python script.
- The cross correlation output of POCO is plotted using the plotting python [see appendix] is used for plotting and dumping the phase value. The phase spectrum of cross correlation within the band should be stable with over time, otherwise change the parameters of the delay and fringe correction scripts accordingly.
- After stable phase spectrum is obtained, the phase correction python scripts is run for compensating the phase error of the cross correlation and the same polarizations of two antennas is brought down to nearly zero.
- After verifying the phase correction on the calibrator radio source, the source is changed to pulsar B0329+54 for observations.
- The data rate and packets traffic is checked using Wireshark software. Then the 10GbE packets of PoBe are captured using Gulp utility for 10 minutes of data acquisition.

- The captured data is depacketized using C program [see appendix] and stored in a binary file as per requirements of PMON software.
- The PMON is used then to plot the pulsar profile and to get the polarimetry output.

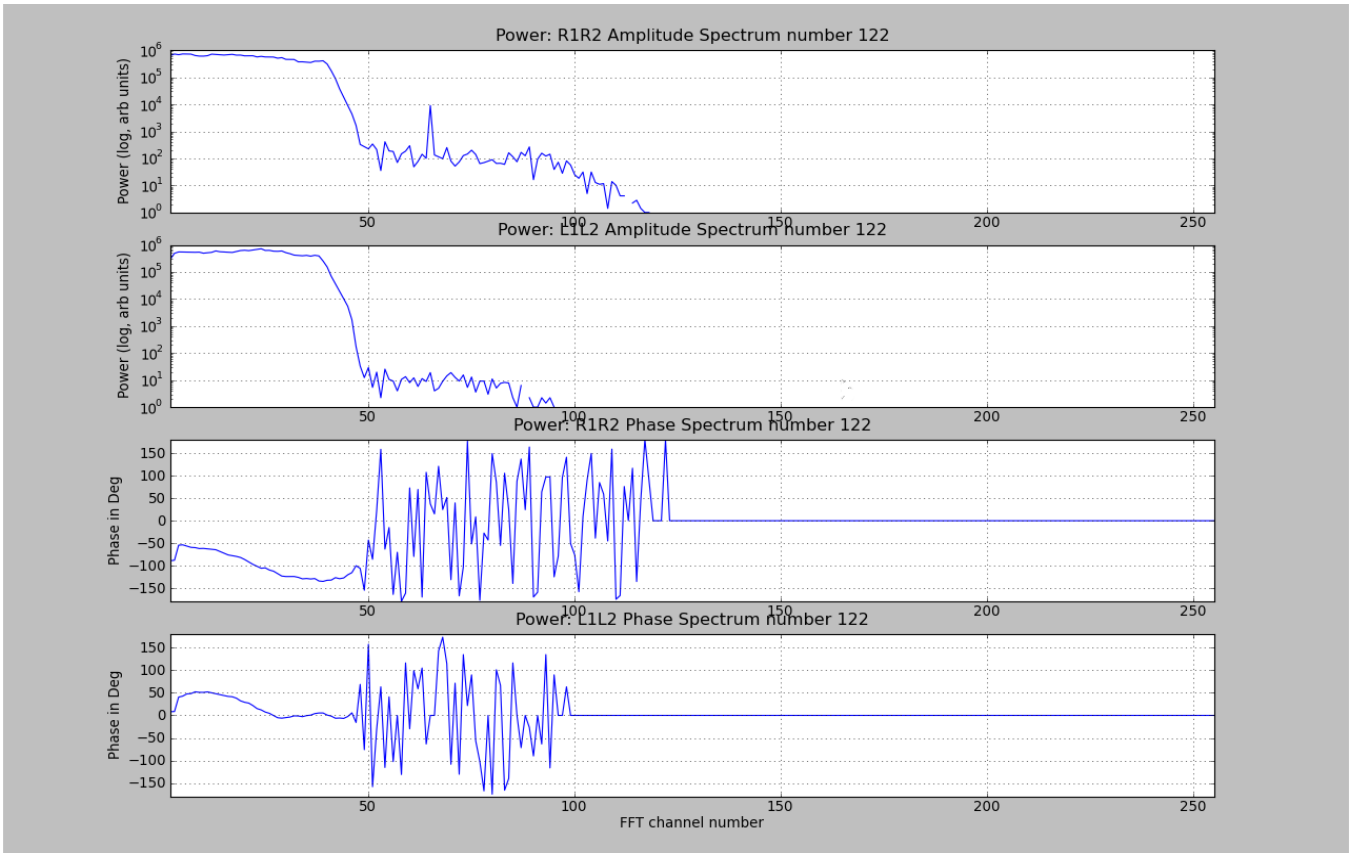


Figure 4.9.: Plot of the cross correlation of the PoCo output without phase correction

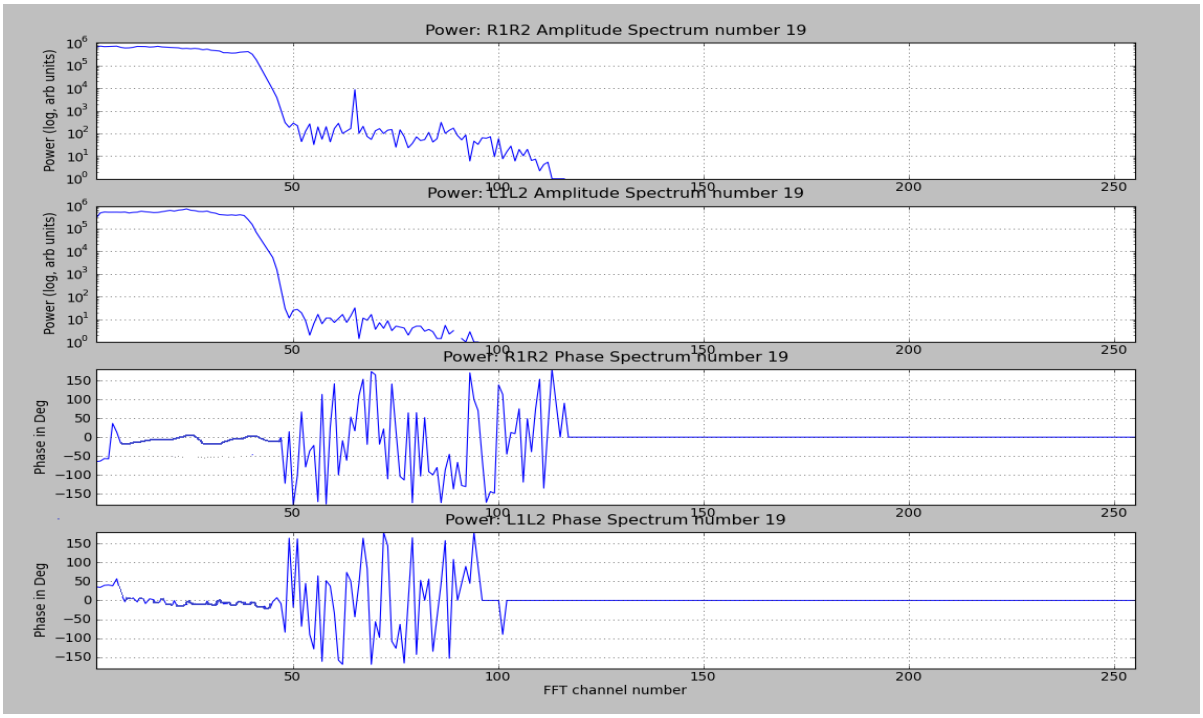


Figure 4.10.: Plot of the cross correlation of the PoCo output with phase correction

Inferences:

- The fig1 shows the stable phase difference between two antennas C2 and C3 for R and L polarizations over the 32MHz baseband after the coarse delay, fine delay and fringe stop corrections. Here the source is a known strong radio source, 3C48, which is used for phase calibration.
- The phase difference is nearly from -50 degree to -140 degree for R1R2 and nearly 50 degree to -5 dgree for the L1L2 polarizations.
- After the phase correction the stable phase difference over the 32 MHz baseband becomes close to zero degree. So the coherent PoBe output can be captured for the Pulsar observations.

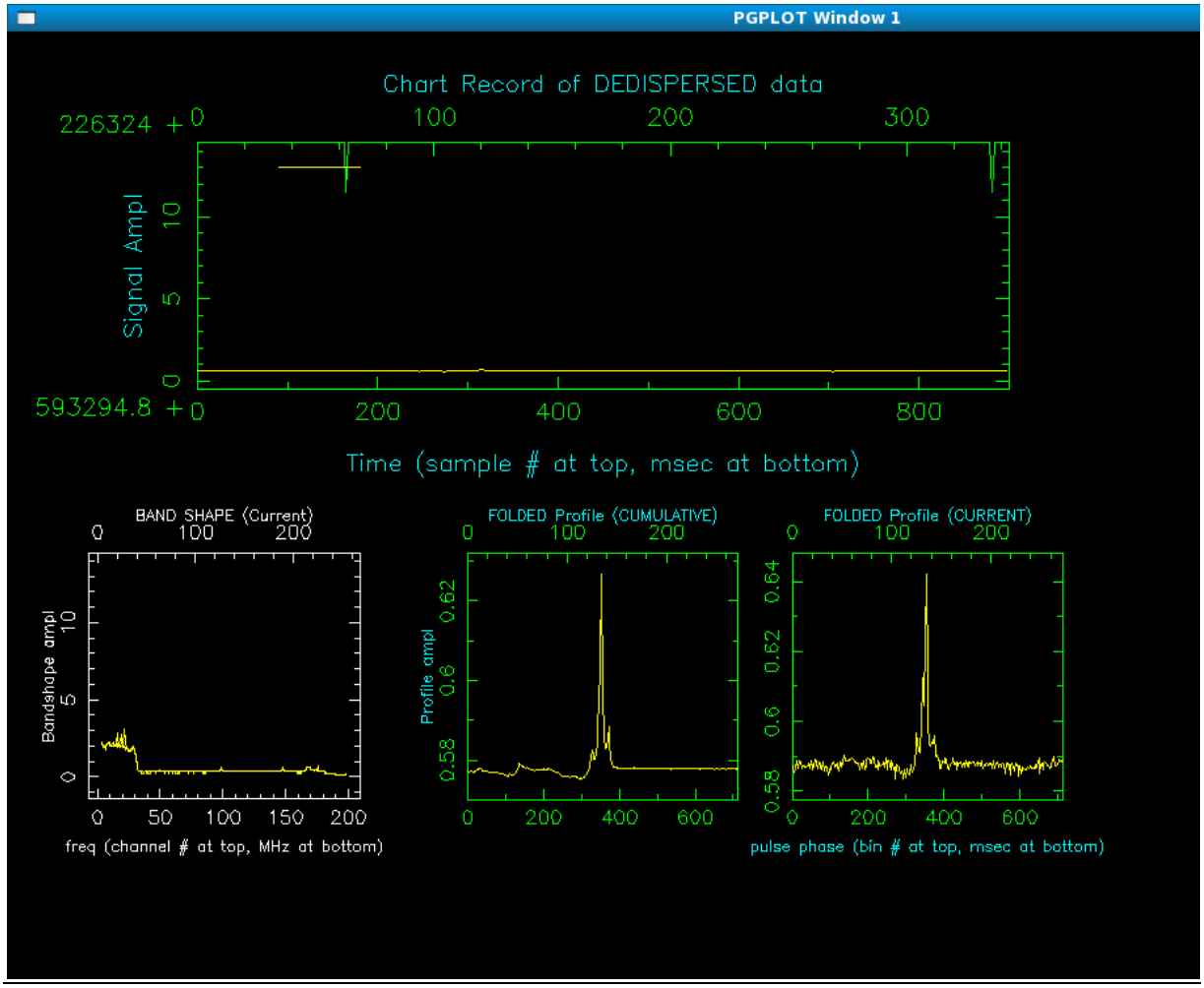


Figure 4.11.: Plot of the PoBe ouput: pulsar profile generated from PMON

Inferences:

- The final RR* profile of the pulsar B0329+54 is plotted in bottom middle plot after folding and de-dispersion.
- The bottom left plot shows the bandshape of the RR* over the complete 200 MHz band of the design.
- The bottom right shows the current RR* profile without folding.

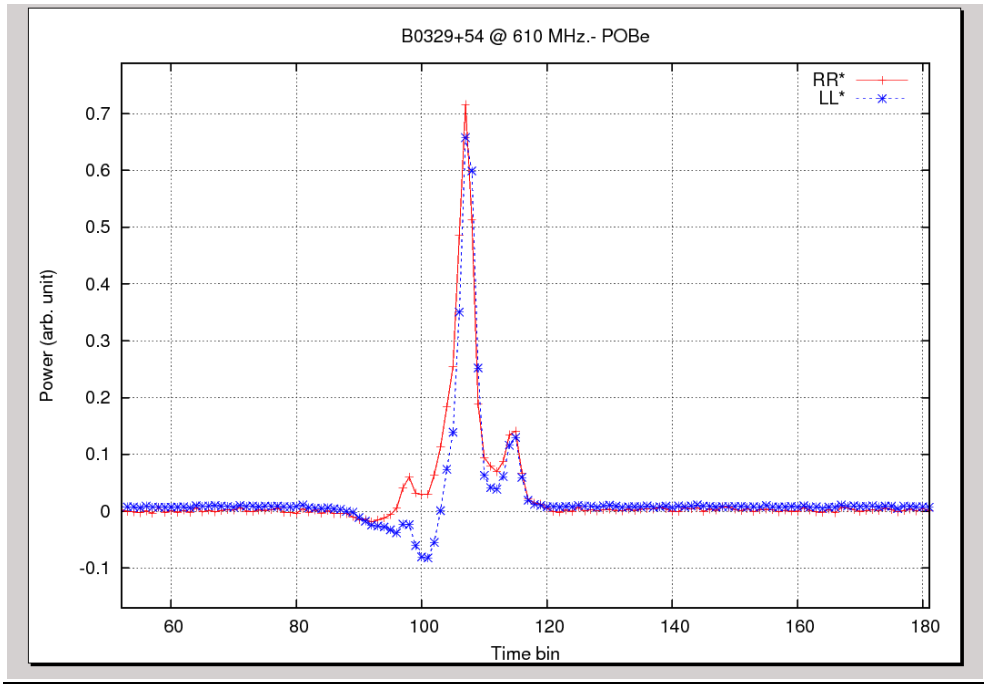


Figure 4.12.: Plot of the profiles of the (R1+R2).(R1+R2)* and (L1+L2).(L1+L2)* : coherent PoBe output

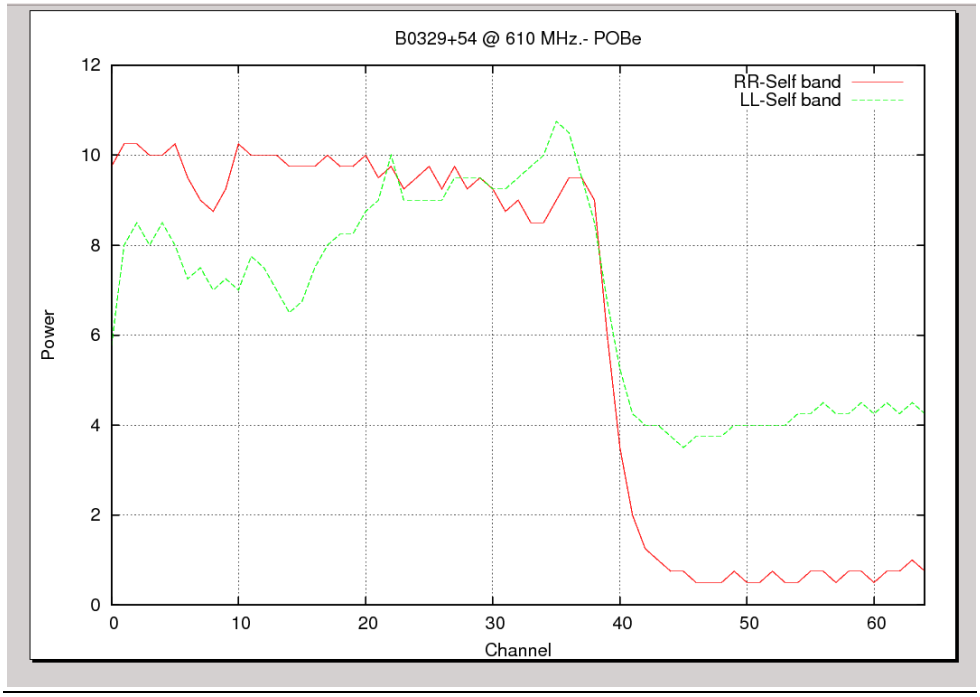


Figure 4.13.: The plot of the bandshape of the RR* and LL*

Inference:

- Ths fig 1 shows the RR* and LL* of the pulsar with time resolution of 272 points over the pulse period.
- The RR* plot seems to be properly obtained.
- But in the LL* plot the deep in the left side peak is not matched with the expected result.
- The 2nd fig is the bandshape of the RR* and LL* over 65 frequency channels.

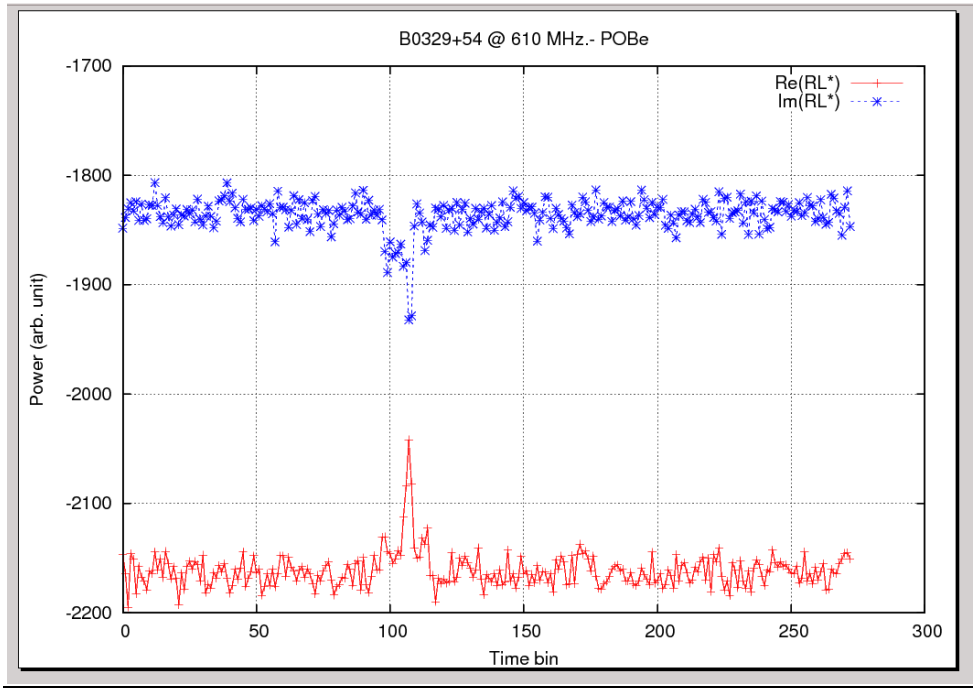


Figure 4.14.: The plot of profile of the Re[RL*] and Im[RL*]

Inferences:

- The Re[RL*] and Im [RL*] is plotted over the pulse period.
- The graph has the proper shape but it seems to have a magnitude offset of very large negative value which is not expected. This should be around the zero level.
- The higher negative value may be caused due to the higher power scaling factor or low input power from the antennas.

5. Conclusions:

The Coherent Pocket Beamformer(PoBe) for two antennas and two polarizations on single ROACH board (FPGA platform) is designed and tested with noise source and with Pulsar signals. As a part of the final design, the Pocket beamformer for 1 antenna 2 polarizations is tested and tested with noise sources and pulsar sources. The output of this design is verified properly and it is working perfectly. The pulsar profile for RR*, LL*, Re [RL*], Im [RL*] is properly plotted and matched the expected results. The full stokes parameters is also plotted with accuracy. This shows the proper working of the PoCo part of the design, MAC design for PoBe and packetization. The phase correction algorithm is also verified with 2 antenna PoCo design, which tested with noise source and known phase calibrating radio source. This shows the proper working of the phase correction algorithm for the PoCo.

The general coherent PoBe design with 1024 FFT points 2 antennae 2 polarizations is not possible to be fitted within single FPGA (Virtex 5 SXT95) because of scarcity of resources like BRAMs(Block RAM)[see appendix]. So, the 2 antennas 2 polarizations specification of the design is modified for resources available on single FPGA. The final design of Coherent Pocket beamformer for R-L polarizations of 2 antennas on single FPGA is now showing proper result with noise source. The skytest, i.e., testing with pulsar source, is done. The Pulsar profile for RR* in PMON is verified. The other three outputs (LL*, Re [RL*], Im [RL*]) in PMON is deviating from the expected result.

6. Future work and recommendations:

The designing of coherent pocket beamformer on single FPGA is a part of the new upgradation process. So, it has got lots scopes for future improvisations. The following future works are recommended:

- The design is checked with 610MHz feed of GMRT antennas. It can be tested over different antenna feeds of GMRT like L band, 325 MHz etc.
- The bandshape error for the Re (RL*) and Im(RL*) output of the coherent PoBe for R-L polarizations of 2 antennas can be improved. The modification can be required only for the power scaling for 512 FFTs.
- The data rate can be improved. So, lowering the integration time can be done to get higher time resolution in the pulsar profile.
- The number of FFT points can be improved for higher spectral resolution over the band.
- The coherent pocket beamformer for R-L polarizations of 2 antennas can be designed for two ROACH boards (FPGA platforms) or can be designed with one FPGA for higher version of FPGAs with more resources.
- Finally the coherent Pocket Beamformer can be designed for array of all the 30 antennas of the GMRT.

7. References

1. Jayaram N Chengalur, Yashwant Gupta, K S Dwarkanath, "*Low Noise Radio Astronomy"*, a note from school on radio astronomy held at NCRA, Pune , 1999.
2. Casper website, *https://casper.berkeley.edu/*
3. Casper tutorials on ROACH board,
    a. On Introduction to Simulink, *https://casper.berkeley.edu/wiki/Introduction_to_Simulink*
    b. On The 10GbE Interface, *https://casper.berkeley.edu/wiki/Tutorial_10GbE*
    c. On the Wideband Pocket Correlator, *https://github.com/casper-astro/tutorials_devel/tree/master/workshop_2010/roach_tut4_wideband_poco*
4. V. M. Tatke, "*A Digital Spectral Correlator for GMRT",* A Thesis submitted for the degree of Master of Science (Engg.), IISc, Bangalore, July, 1998
5. Cambodge bist, " *Pocket Beamformer on FPGA"* , student project report, NCRA,TIFR

Appendix A

Design summary for the one antenna two polarizations Beamformer design that tested with noise source and sky test.

Modal file name : <coh1a_2pol_1/>

```
Release 11.5 Map L.70 (lin64)
Xilinx Mapping Report File for Design 'system'

Design Information
------------------
Command Line   : map -ise ../__xps/ise/system.ise -timing -detail -ol high -
xe n
-register_duplication -o system_map.ncd -w -pr b system.ngdsystem.pcf
Target Device  : xc5vsx95t
Target Package : ff1136
Target Speed   : -1
Mapper Version : virtex5 -- $Revision: 1.51.18.1 $

Mapped Date    : Tue Mar  6 11:14:31 2012

Design Summary
--------------
Number of errors:      0
Number of warnings: 1974
Slice Logic Utilization:
  Number of Slice Registers:                25,148 out of  58,880   42%
    Number used as Flip Flops:              25,147
    Number used as Latch-thrus:                  1
  Number of Slice LUTs:                     26,057 out of  58,880   44%
    Number used as logic:                   16,915 out of  58,880   28%
      Number using O6 output only:          13,981
      Number using O5 output only:           1,658
      Number using O5 and O6:                1,276
    Number used as Memory:                   8,850 out of  24,320   36%
      Number used as Dual Port RAM:            316
        Number using O6 output only:           200
        Number using O5 output only:            36
        Number using O5 and O6:                 80
      Number used as Shift Register:         8,534
        Number using O6 output only:         8,534
    Number used as exclusive route-thru:       292
  Number of route-thrus:                     2,567
    Number using O6 output only:             1,924
    Number using O5 output only:               624
    Number using O5 and O6:                     19

Slice Logic Distribution:
```

```
  Number of occupied Slices:                 9,764 out of  14,720   66%
  Number of LUT Flip Flop pairs used:       31,643
    Number with an unused Flip Flop:         6,495 out of  31,643   20%
    Number with an unused LUT:               5,586 out of  31,643   17%
    Number of fully used LUT-FF pairs:      19,562 out of  31,643   61%
    Number of unique control sets:            430
    Number of slice register sites lost
to control set restrictions:              751 out of  58,880    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
one Flip Flop within a slice.  A control set is a unique combination of
clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
over-mapped for a non-slice resource or if Placement fails.
  OVERMAPPING of BRAM resources should be ignored if the design is
over-mapped for a non-BRAM resource or if placement fails.

IO Utilization:
  Number of bonded IOBs:                       131 out of     640   20%
    Number of LOCed IOBs:                      131 out of     131  100%
    IOB Flip Flops:                            102
    IOB Master Pads:                             1
    IOB Slave Pads:                              1
    Number of bonded IPADs:                     36 out of      50   72%
    Number of bonded OPADs:                     32 out of      32  100%

Specific Feature Utilization:
  Number of BlockRAM/FIFO:                     160 out of     244   65%
    Number using BlockRAM only:                160
    Total primitives used:
      Number of 36k BlockRAM used:              27
      Number of 18k BlockRAM used:             237
    Total Memory used (KB):                  5,238 out of   8,784   59%
  Number of BUFG/BUFGCTRLs:                      9 out of      32   28%
    Number used as BUFGs:                        9
  Number of BUFDSs:                              2 out of       8   25%
  Number of CRC64s:                              2 out of      16   12%
  Number of DCM_ADVs:                            2 out of      12   16%
  Number of DSP48Es:                           108 out of     640   16%
  Number of GTP_DUALs:                           8 out of       8  100%
  Number of PLL_ADVs:                            2 out of       6   33%

Average Fanout of Non-Clock Nets:                2.36

Peak Memory Usage:  1697 MB
Total REAL time to MAP completion:  17 mins 5 secs
Total CPU time to MAP completion:   16 mins 46 secs
```

Appendix B

Design summary for two antennas single polarization PoBo design with phase correction.

Modal file name : <two_antenna_poco_phase_corr>

```
Release 11.5 Map L.70 (lin64)
Xilinx Mapping Report File for Design 'system'

Design Information
------------------
Command Line   : map -ise ../__xps/ise/system.ise -timing -detail -ol high -
xe n
-register_duplication -o system_map.ncd -w -pr b system.ngdsystem.pcf
Target Device  : xc5vsx95t
Target Package : ff1136
Target Speed   : -1
Mapper Version : virtex5 -- $Revision: 1.51.18.1 $
Mapped Date    : Wed Apr  4 19:18:08 2012

Design Summary
--------------
Number of errors:      0
Number of warnings: 1402
Slice Logic Utilization:
  Number of Slice Registers:                22,335 out of  58,880   37%
    Number used as Flip Flops:              22,335
  Number of Slice LUTs:                     18,726 out of  58,880   31%
    Number used as logic:                   12,805 out of  58,880   21%
      Number using O6 output only:          10,307
      Number using O5 output only:           1,523
      Number using O5 and O6:                  975
    Number used as Memory:                   5,649 out of  24,320   23%
      Number used as Dual Port RAM:            100
        Number using O6 output only:            28
        Number using O5 output only:            36
        Number using O5 and O6:                 36
      Number used as Shift Register:         5,549
        Number using O6 output only:         5,549
    Number used as exclusive route-thru:       272
  Number of route-thrus:                     2,189
    Number using O6 output only:             1,776
    Number using O5 output only:               399
    Number using O5 and O6:                     14

Slice Logic Distribution:
  Number of occupied Slices:                 8,104 out of  14,720   55%
  Number of LUT Flip Flop pairs used:       26,460
    Number with an unused Flip Flop:         4,125 out of  26,460   15%
    Number with an unused LUT:               7,734 out of  26,460   29%
    Number of fully used LUT-FF pairs:      14,601 out of  26,460   55%
    Number of unique control sets:            195
    Number of slice register sites lost
```

```
to control set restrictions:             232 out of  58,880    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
one Flip Flop within a slice.  A control set is a unique combination of
clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
over-mapped for a non-slice resource or if Placement fails.
  OVERMAPPING of BRAM resources should be ignored if the design is
over-mapped for a non-BRAM resource or if placement fails.

IO Utilization:
  Number of bonded IOBs:                   131 out of     640   20%
    Number of LOCed IOBs:                  131 out of     131  100%
    IOB Flip Flops:                        102
    IOB Master Pads:                         1
    IOB Slave Pads:                          1

Specific Feature Utilization:
  Number of BlockRAM/FIFO:                 225 out of     244   92%
    Number using BlockRAM only:            225
    Total primitives used:
      Number of 36k BlockRAM used:         144
      Number of 18k BlockRAM used:         152
    Total Memory used (KB):              7,920 out of   8,784   90%
  Number of BUFG/BUFGCTRLs:                  3 out of      32    9%
    Number used as BUFGs:                    3
  Number of DCM_ADVs:                        2 out of      12   16%
  Number of DSP48Es:                       224 out of     640   35%

Average Fanout of Non-Clock Nets:            2.35

Peak Memory Usage:  1612 MB
Total REAL time to MAP completion:  13 mins 33 secs
Total CPU time to MAP completion:   13 mins 4 secs
```

Appendix C

Design summary for 2 antenna 2 polarizations 1K FFT points symmetric design with 240% memory over shoot.

```
Design Information
------------------
Command Line   : map -ise ../__xps/ise/system.ise -timing -detail -ol high -
xe n
-register_duplication -o system_map.ncd -w -pr b system.ngdsystem.pcf
Target Device  : xc5vsx95t
Target Package : ff1136
Target Speed   : -1
Mapper Version : virtex5 -- $Revision: 1.51.18.1 $
Mapped Date    : Wed Apr 11 04:09:16 2012

Interim Summary
---------------
Slice Logic Utilization:
  Number of Slice Registers:                43,192 out of  58,880   73%
    Number used as Flip Flops:              43,191
    Number used as Latch-thrus:                  1
  Number of Slice LUTs:                     44,905 out of  58,880   76%
    Number used as logic:                   27,781 out of  58,880   47%
      Number using O6 output only:          22,824
      Number using O5 output only:           2,917
      Number using O5 and O6:                2,040
    Number used as Memory:                  16,589 out of  24,320   68%
      Number used as Dual Port RAM:            394
        Number using O6 output only:           222
        Number using O5 output only:            50
        Number using O5 and O6:                122
      Number used as Shift Register:        16,195
        Number using O6 output only:        16,195
    Number used as exclusive route-thru:       535
  Number of route-thrus:                     3,452
    Number using O6 output only:             3,452

Slice Logic Distribution:
  Number of LUT Flip Flop pairs used:       56,427
    Number with an unused Flip Flop:        13,235 out of  56,427   23%
    Number with an unused LUT:              11,522 out of  56,427   20%
    Number of fully used LUT-FF pairs:      31,670 out of  56,427   56%
    Number of unique control sets:             576
    Number of slice register sites lost
to control set restrictions:             894 out of  58,880    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
one Flip Flop within a slice.  A control set is a unique combination of
clock, reset, set, and enable signals for a registered element.
```

The Slice Logic Distribution report is not meaningful if the design is over-mapped for a non-slice resource or if Placement fails.

OVERMAPPING of BRAM resources should be ignored if the design is over-mapped for a non-BRAM resource or if placement fails.

IO Utilization:				
Number of bonded IOBs:	197	out of	640	30%
Number of LOCed IOBs:	197	out of	197	100%
IOB Flip Flops:	131			
IOB Master Pads:	2			
IOB Slave Pads:	2			
Number of bonded IPADs:	36	out of	50	72%
Number of bonded OPADs:	32	out of	32	100%
Specific Feature Utilization:				
Number of BlockRAM/FIFO:	586	out of	244	240% (OVERMAPPED)
Number using BlockRAM only:	586			
Total primitives used:				
Number of 36k BlockRAM used:	153			
Number of 18k BlockRAM used:	433			
Total Memory used (KB):	13,302	out of	8,784	151% (OVERMAPPED)
Number of BUFG/BUFGCTRLs:	10	out of	32	31%
Number used as BUFGs:	10			
Number of IDELAYCTRLs:	22	out of	22	100%
Number of BUFDSs:	2	out of	8	25%
Number of CRC64s:	2	out of	16	12%
Number of DCM_ADVs:	3	out of	12	25%
Number of DSP48Es:	256	out of	640	40%
Number of GTP_DUALs:	8	out of	8	100%
Number of PLL_ADVs:	2	out of	6	33%

Appendix D

Design summary for 2 antennas 2 polarizations 512 points FFT and course delay is reduced to 256 clk and symmetric design with 163% overshoot of memory resources.

```
---------
Design Information
------------------
Command Line   : map -ise ../__xps/ise/system.ise -timing -detail -ol high -
xe n
-register_duplication -o system_map.ncd -w -pr b system.ngdsystem.pcf
Target Device  : xc5vsx95t
Target Package : ff1136
Target Speed   : -1
Mapper Version : virtex5 -- $Revision: 1.51.18.1 $
Mapped Date    : Wed Apr 11 16:30:57 2012

Interim Summary
---------------
Slice Logic Utilization:
  Number of Slice Registers:                43,128 out of  58,880   73%
    Number used as Flip Flops:              43,127
    Number used as Latch-thrus:                  1
  Number of Slice LUTs:                     44,833 out of  58,880   76%
    Number used as logic:                   27,741 out of  58,880   47%
      Number using O6 output only:          22,792
      Number using O5 output only:           2,909
      Number using O5 and O6:                2,040
    Number used as Memory:                  16,581 out of  24,320   68%
      Number used as Dual Port RAM:            394
        Number using O6 output only:           222
        Number using O5 output only:            50
        Number using O5 and O6:                122
      Number used as Shift Register:        16,187
        Number using O6 output only:        16,187
    Number used as exclusive route-thru:       511
  Number of route-thrus:                     3,420
    Number using O6 output only:             3,420

Slice Logic Distribution:
  Number of LUT Flip Flop pairs used:       56,339
    Number with an unused Flip Flop:        13,211 out of  56,339   23%
    Number with an unused LUT:              11,506 out of  56,339   20%
    Number of fully used LUT-FF pairs:      31,622 out of  56,339   56%
    Number of unique control sets:             576
    Number of slice register sites lost
to control set restrictions:             894 out of  58,880    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
one Flip Flop within a slice.  A control set is a unique combination of
clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
over-mapped for a non-slice resource or if Placement fails.
```

```
  OVERMAPPING of BRAM resources should be ignored if the design is
over-mapped for a non-BRAM resource or if placement fails.

IO Utilization:
  Number of bonded IOBs:                       197 out of     640   30%
    Number of LOCed IOBs:                      197 out of     197  100%
    IOB Flip Flops:                            131
    IOB Master Pads:                             2
    IOB Slave Pads:                              2
    Number of bonded IPADs:                     36 out of      50   72%
    Number of bonded OPADs:                     32 out of      32  100%

Specific Feature Utilization:
  Number of BlockRAM/FIFO:                     490 out of     244  200%
(OVERMAPPED)
    Number using BlockRAM only:                490
    Total primitives used:
      Number of 36k BlockRAM used:             153
      Number of 18k BlockRAM used:             337
    Total Memory used (KB):                 11,574 out of   8,784  131%
(OVERMAPPED)
  Number of BUFG/BUFGCTRLs:                     10 out of      32   31%
    Number used as BUFGs:                       10
  Number of IDELAYCTRLs:                        22 out of      22  100%
  Number of BUFDSs:                              2 out of       8   25%
  Number of CRC64s:                              2 out of      16   12%
  Number of DCM_ADVs:                            3 out of      12   25%
  Number of DSP48Es:                           256 out of     640   40%
  Number of GTP_DUALs:                           8 out of       8  100%
  Number of PLL_ADVs:                            2 out of       6   33%


Design Summary
--------------
Number of errors   :   2
Number of warnings :3216

Section 1 - Errors
------------------
ERROR:Place:836 - Not enough free sites available for the components of the
following type(s).
      BLOCKRAM    Number of Components 643      Number of Sites 488

ERROR:Pack:1654 - The timing-driven placement phase encountered an error.
--------------------------------------------------------------------------
--
```

Appendix D

Design summary for 2 antennas 2 polarization working design.

Modal name : <mod_en_pobe.mdl>

```
Release 11.5 Map L.70 (lin64)
Xilinx Mapping Report File for Design 'system'

Design Information
------------------
Command Line   : map -ise ../__xps/ise/system.ise -timing -detail -ol high -
xe n -register_duplication -o system_map.ncd
-w -pr b system.ngdsystem.pcf
Target Device  : xc5vsx95t
Target Package : ff1136
Target Speed   : -1
Mapper Version : virtex5 -- $Revision: 1.51.18.1 $
Mapped Date    : Thu Apr 26 03:06:52 2012

Design Summary
--------------
Number of errors:      0
Number of warnings: 3654
Slice Logic Utilization:
  Number of Slice Registers:                45,267 out of  58,880   76%
    Number used as Flip Flops:              45,266
    Number used as Latch-thrus:                  1
  Number of Slice LUTs:                     47,183 out of  58,880   80%
    Number used as logic:                   27,871 out of  58,880   47%
      Number using O6 output only:          22,929
      Number using O5 output only:           2,800
      Number using O5 and O6:                2,142
    Number used as Memory:                  18,888 out of  24,320   77%
      Number used as Dual Port RAM:            400
        Number using O6 output only:           224
        Number using O5 output only:            52
        Number using O5 and O6:                124
      Number used as Shift Register:        18,488
        Number using O6 output only:        18,488
    Number used as exclusive route-thru:        424
  Number of route-thrus:                     6,087
    Number using O6 output only:             3,070
    Number using O5 output only:             2,961
    Number using O5 and O6:                     56

Slice Logic Distribution:
  Number of occupied Slices:                14,163 out of  14,720   96%
  Number of LUT Flip Flop pairs used:       52,595
    Number with an unused Flip Flop:         7,328 out of  52,595   13%
```

```
    Number with an unused LUT:               5,412 out of  52,595   10%
    Number of fully used LUT-FF pairs:      39,855 out of  52,595   75%
    Number of unique control sets:             581
    Number of slice register sites lost
to control set restrictions:           1,038 out of  58,880    1%

  A LUT Flip Flop pair for this architecture represents one LUT paired with
one Flip Flop within a slice.  A control set is a unique combination of
clock, reset, set, and enable signals for a registered element.
  The Slice Logic Distribution report is not meaningful if the design is
over-mapped for a non-slice resource or if Placement fails.
  OVERMAPPING of BRAM resources should be ignored if the design is
over-mapped for a non-BRAM resource or if placement fails.

IO Utilization:
  Number of bonded IOBs:                       199 out of     640   31%
    Number of LOCed IOBs:                      199 out of     199  100%
    IOB Flip Flops:                            132
    IOB Master Pads:                             2
    IOB Slave Pads:                              2
    Number of bonded IPADs:                     36 out of      50   72%
    Number of bonded OPADs:                     32 out of      32  100%

Specific Feature Utilization:
  Number of BlockRAM/FIFO:                     240 out of     244   98%
    Number using BlockRAM only:                240
    Total primitives used:
      Number of 36k BlockRAM used:              91
      Number of 18k BlockRAM used:             285
    Total Memory used (KB):                  8,406 out of   8,784   95%
  Number of BUFG/BUFGCTRLs:                     11 out of      32   34%
    Number used as BUFGs:                       11
  Number of BUFDSs:                              2 out of       8   25%
  Number of CRC64s:                              2 out of      16   12%
  Number of DCM_ADVs:                            3 out of      12   25%
  Number of DSP48Es:                           246 out of     640   38%
  Number of GTP_DUALs:                           8 out of       8  100%
  Number of PLL_ADVs:                            2 out of       6   33%

Average Fanout of Non-Clock Nets:                2.24

Peak Memory Usage:  2370 MB
Total REAL time to MAP completion:  1 hrs 9 mins 45 secs
Total CPU time to MAP completion:   1 hrs 9 mins 18 secs
------------------------------------------------------------
```

Appendix F

Configuration script in python for final 2 antenna 2 polarization design.

Original name: `test1config.py`

```
-----------------------------------------------------------------
#!/usr/bin/python

#from katcp import BlockingClient, Message

importkatcp, numpy, pylab, time, corr


#device_host = "192.168.100.72"
#device_host = "192.168.100.80"
device_host = "192.168.100.70" #roacboard IP
device_port = 7147

my_corr=corr.katcp_wrapper.FpgaClient(device_host)

while not (my_corr.is_connected()):
     pass

my_corr.write_int("scale",2000)#for 256chan=2000 , for
512chan(coh1a_2pol_1)=500
print "scale = %i" %my_corr.read_int("scale")

my_corr.write_int("scale1",2000)#for 256chan=2000 , for
512chan(coh1a_2pol_1)=500
print "scale1 = %i" %my_corr.read_int("scale1")

my_corr.write_int("fft_shift",0x1ff00000)# 1024 fft = 0x03f00000, 512
fft= 0x1ff00000
print "fft_shift = %x" %my_corr.read_int("fft_shift")

my_corr.write_int("acc_len",524288)#524288
print "acc_len = %i" %my_corr.read_int("acc_len")

my_corr.write_int("acc_cntrl_int_time",2048)#integration_time in terms
of fft cycles
print "acc_cntrl_int_time = %i"
%my_corr.read_int("acc_cntrl_int_time")


my_corr.write_int("sys_rst",1)
print "sys_rst = %i" %my_corr.read_int("sys_rst")
my_corr.write_int("sys_rst",0)
```

```
print "sys_rst = %i" %my_corr.read_int("sys_rst")


my_corr.write_int("delay_a",0)
print "delay_a = %i" %my_corr.read_int("delay_a")
my_corr.write_int("delay_b",0)
print "delay_b = %i" %my_corr.read_int("delay_b")

my_corr.write_int("delaylatency",6) #delayblock latency= 0->15
print "delaylatency = %i" %my_corr.read_int("delaylatency")

my_corr.write_int("theta_fract",0)
print "theta_fract = %i" %my_corr.read_int("theta_fract")
my_corr.write_int("theta_fract1",0)
print "theta_fract1 = %i" %my_corr.read_int("theta_fract1")

my_corr.write_int("theta_fs",0)
print "theta_fs = %i" %my_corr.read_int("theta_fs")
my_corr.write_int("en_theta_fs",1)
print "en_theta_fs = %i" %my_corr.read_int("en_theta_fs")
my_corr.write_int("en_theta_fs",0)
print "en_theta_fs = %i" %my_corr.read_int("en_theta_fs")

my_corr.write_int("theta_fs1",0)
print "theta_fs1 = %i" %my_corr.read_int("theta_fs1")
my_corr.write_int("en_theta_fs1",1)
print "en_theta_fs1 = %i" %my_corr.read_int("en_theta_fs1")
my_corr.write_int("en_theta_fs1",0)
print "en_theta_fs1 = %i" %my_corr.read_int("en_theta_fs1")

my_corr.write_int("fft_fs",0)
print "fft_fs = %i" %my_corr.read_int("fft_fs")
my_corr.write_int("fft_fs1",0)
print "fft_fs1 = %i" %my_corr.read_int("fft_fs1")

my_corr.write_int("fft_fract",0)
print "fft_fract = %i" %my_corr.read_int("fft_fract")
my_corr.write_int("fft_fract1",0)
print "fft_fract1 = %i" %my_corr.read_int("fft_fract1")

my_corr.write_int("fblocklatency",7) #fringeblock latency= 16 + 0->15
print "fblocklatency = %i" %my_corr.read_int("fblocklatency")

my_corr.write_int("pobe_mode",2)
print "pobe_mode = %i" %my_corr.read_int("pobe_mode")

my_corr.write_int("dir_x2_v2ch1_muxint_muxint",1) #0-constant, 1-pobe
print "dir_x2_v2ch1_muxint_muxint= %i"
%my_corr.read_int("dir_x2_v2ch1_muxint_muxint")
my_corr.write_int("dir_x2_v2ch0_muxint_muxint",1) #0-constant, 1-pobe
print "dir_x2_v2ch0_muxint_muxint= %i"
%my_corr.read_int("dir_x2_v2ch0_muxint_muxint")
```

```
my_corr.write_int("dir_x2_v1ch1_muxint_muxint",1) #0-constant, 1-pobe
print "dir_x2_v1ch1_muxint_muxint= %i"
%my_corr.read_int("dir_x2_v1ch1_muxint_muxint")
my_corr.write_int("dir_x2_v1ch0_muxint_muxint",1) #0-constant, 1-pobe
print "dir_x2_v1ch0_muxint_muxint= %i"
%my_corr.read_int("dir_x2_v1ch0_muxint_muxint")

my_corr.write_int("dir_x2_v1v2ch0_muxint_muxint",1) #0-constant, 1-
pobe
print "dir_x2_v1v2ch0_muxint_muxint= %i"
%my_corr.read_int("dir_x2_v1v2ch0_muxint_muxint")
my_corr.write_int("dir_x2_v1v2ch0_muxint1_muxint",1) #0-constant, 1-
pobe
print "dir_x2_v1v2ch0_muxint1_muxint= %i"
%my_corr.read_int("dir_x2_v1v2ch0_muxint1_muxint")
my_corr.write_int("dir_x2_v1v2ch1_muxint_muxint",1) #0-constant, 1-
pobe
print "dir_x2_v1v2ch1_muxint_muxint= %i"
%my_corr.read_int("dir_x2_v1v2ch1_muxint_muxint")
my_corr.write_int("dir_x2_v1v2ch1_muxint1_muxint",1) #0-constant, 1-
pobe
print "dir_x2_v1v2ch1_muxint1_muxint= %i"
%my_corr.read_int("dir_x2_v1v2ch1_muxint1_muxint")

my_corr.write_int("dir_x2_muxdel1_muxdel1",1) #delay 0->15
print "dir_x2_muxdel1_muxdel1= %i"
%my_corr.read_int("dir_x2_muxdel1_muxdel1")

my_corr.write_int("dir_x2_10G_10gmux",1)#use 0 for counter-64bit and 1
for pobe output
print "dir_x2_10G_10gmux= %i" %my_corr.read_int("dir_x2_10G_10gmux")

my_corr.write_int("dir_x2_10G_gbe_rst0",1)
print "dir_x2_10G_gbe_rst0= %i"
%my_corr.read_int("dir_x2_10G_gbe_rst0")
my_corr.write_int("dir_x2_10G_gbe_rst0",0)
print "dir_x2_10G_gbe_rst0= %i"
%my_corr.read_int("dir_x2_10G_gbe_rst0")

my_corr.write_int("dir_x2_10G_dest_port0",10001)
print "dir_x2_10G_dest_port0= %i"
%my_corr.read_int("dir_x2_10G_dest_port0")

my_corr.write_int("dir_x2_10G_dest_ip0",0xC0A808C8) #eth=
192.168.8.200
print "dir_x2_10G_dest_ip0= %x"
%my_corr.read_int("dir_x2_10G_dest_ip0")

my_corr.stop()
----------------------------------------------------------------------
```

Appendix G

Python script for reading phase values from correlated output along with plotting utility.

Original file name: <stp_4pobe_plot_dump.py>

```
------------------------------------------------------------
#!/usr/bin/python

#from katcp import BlockingClient, Message
'''PLOTS CROSS AND AUTO CORRELATION FUNCTION OF WIDEBAND POCO n-CH
DESIGN ON ROACH BOARD
This program is a generalized program with few modifications to plot
and dump the data in
analysis program-tax native format into the given specified format.
'''
importmatplotlib
frommatplotlib import rcParams
matplotlib.use('GTKAgg')
importkatcp, numpy, pylab
importgtk, gobject, time, struct, sys, math

def interleave(*args):
    x=[]
foridx in range(0, max(len(arg) for arg in args)):
forarg in args:
try:
                #yield arg[idx]
x.append(arg[idx])
exceptIndexError:
continuie
return x


#device_host     = "192.168.100.72"    # For Lab Testing
#device_host     = "192.168.4.91"# For Receiver Room Testing
device_port     = 7147
acq_time   = 2**27/(100.0*(10**6))          # 0.89 is 1 sync period
time
n_chans         = 512   #512 fft
ants       = 2

if __name__ == '__main__':
fromoptparse import OptionParser
        p = OptionParser()
p.set_usage('tut4_poco_plot.py <ROACH_HOSTNAME_or_IP> [options] ')
p.set_description(__doc__)
```

```
p.add_option('-l', '--log', dest='log', action='store_true',
default=False,
help='''Plot the power in logarithmic scale (requires some non-zero
value signal).''')
p.add_option('--hold', dest='hold', action='store_true',
default=False,
help='''Turn on hold. This will plot subsequent specra on top of each
other.''')
p.add_option('-f', '--file', action='store', type='string',
dest='filename',
help='''file dump the data in native tax format''', metavar='FILE')
opts, args = p.parse_args(sys.argv[1:])

ifargs==[]:
print 'Please specify a ROACH board. \nExiting.'
exit()
else:
roach = args[0]

client =  katcp.BlockingClient(roach, device_port)
client.start()

bram_map=['dir_x1_ae_real','dir_x1_ae_imag','dir_x1_bf_real','dir_x1_b
f_imag','dir_x1_cg_real','dir_x1_cg_imag','dir_x1_dh_real','dir_x1_dh_
imag']

x=range(0,n_chans/2)
xlab=('FFT channel number')

fig = pylab.figure(1)
manager = pylab.get_current_fig_manager()
cnt = 0
#creating file to dump data
if (opts.filename != None):
      f_name = opts.filename #+ opts.filename
      print "dumping File is %s" %f_name
      fptr = open (f_name,'w')
else:
      print "dumping file is missing\n"

#continuous loop
defupdatefig(*args):
      globalcnt, ants
      cnt += 1
      print 'Grabbing spectrum number %i'%cnt

      rply       = [[],[],[],[],[],[],[],[],[]]
      cross           = [[],[]]
cx_intrleav     = []
      Rcomplex   = []
      Lcomplex   = []
      Rcross          = [[],[]]
```

```
      Lcross          = [[],[]]
      main_loop_t     = time.time()
# DATA GRABBING UTILITY
      for input in range(8):
            reply, inform   =
client.blocking_request(katcp.Message.request("wordread",
bram_map[input], "0", "128"))
            #print reply
            rply[input]     = map(eval, reply.arguments[1:n_chans/4+1])

            for i in range(128):                              # For 2's
Complement interpretation :)
                  if (rply[input][i] > 2147483647):
                        rply[input][i] = rply[input][i] - 4294967296
# DATA MANIPULATION------------#
      Rcross[0]  = interleave(rply[0],rply[2]) #real after interleaving
even odd
      Rcross[1]  = interleave(rply[1],rply[3]) #imag after interleaving
even odd
      Rcomplex=numpy.copy(numpy.array(Rcross[0]) +
numpy.array(Rcross[1])*1j)
      Lcross[0]  = interleave(rply[4],rply[6]) #real
      Lcross[1]  = interleave(rply[5],rply[7]) #imag
      Lcomplex=numpy.copy(numpy.array(Lcross[0]) +
numpy.array(Lcross[1])*1j)

      print
numpy.size(Rcross[0]),numpy.size(Rcross[1]),numpy.size(Lcross[0]),nump
y.size(Lcross[1])
#DATA DUMPING UTILITY-------------------#
      if (opts.filename != None):
            k=0
            for k in range (256):
            fptr.write('%f\n'%numpy.angle(Rcomplex[k],deg=True))
            k=0
            for k in range (256):
                  fptr.write('%f\n'%numpy.angle(Lcomplex[k],deg=True))

#DATA PLOTTING UTILITY-------------- #
      pylab.subplot(411) #plotting R1R2 Amplitude
      pylab.ioff()
      pylab.hold(False)
      print 'size of Rcomplex = %i'%numpy.size(Rcomplex)
      if opts.log:
            pylab.semilogy(x, numpy.abs(Rcomplex))
            pylab.ylabel('Power (log, arb units)')
      else:
            pylab.plot(x, numpy.abs(Rcomplex))
            pylab.ylabel('Power (linear, arb units)')
      pylab.xlim(x[2],x[n_chans/2-1])
      pylab.grid()
pylab.title('Power: R1R2 Amplitude Spectrum number %i'%cnt)
```

```
      pylab.subplot(412) #plotting L1L2 Amplitude
      pylab.ioff()
      pylab.hold(opts.hold)
      if opts.log:
      pylab.semilogy(x, numpy.abs(Lcomplex))
            pylab.ylabel('Power (log, arb units)')
      else:
            pylab.plot(x, numpy.abs(Lcomplex))
            pylab.ylabel('Power (linear, arb units)')
      pylab.xlim(x[2],x[n_chans/2-1])
      pylab.grid()
      pylab.title('Power: L1L2 Amplitude Spectrum number %i'%cnt)

      pylab.subplot(413) #plotting R1R2 phase
      pylab.ioff()
      pylab.hold(opts.hold)
      pylab.plot(numpy.angle(Rcomplex,deg=True))
      pylab.xlim(x[2],x[n_chans/2-1])
      pylab.ylim(-180,+180)
      pylab.ylabel('Phase in Deg')
      pylab.grid()
      pylab.title('Power: R1R2 Phase Spectrum number %i'%cnt)

      pylab.subplot(414) #plotting L1L2 phase
      pylab.ioff()
      pylab.hold(opts.hold)
      pylab.plot(numpy.angle(Lcomplex,deg=True))
      pylab.xlim(x[2],x[n_chans/2-1])
      pylab.ylim(-180,+180)
      pylab.ylabel('Phase in Deg')
      pylab.grid()
      pylab.title('Power: L1L2 Phase Spectrum number %i'%cnt)
      pylab.xlabel(xlab)
      pylab.draw()

      return True
gobject.idle_add(updatefig)
pylab.show()
client.stop()
client.join()
if(opts.filename != None):
fptr.close()
print 'Done with all'


------------------------------------------------------
```

Appendix H

Python script for averaging the phase data and uploading phase value to FPGA

Original file name: `phase_update_2ant_2pol.py`

```
------------------------------------------------------------
#!/usr/bin/python

#from katcp import BlockingClient, Message

importkatcp, numpy, pylab, time, corr, sys


device_host = "192.168.100.70"
device_port = 7147

if __name__ == '__main__':
fromoptparse import OptionParser
        p = OptionParser()
p.set_usage('phase_update_2ant_2pol.py <phase_filename> [options] ')
p.set_description(__doc__)
p.add_option('-t', '--flag', dest='flag', action='store_true',
default=False,
help='''for 1st runtime flag=0. to run previously phase updated bof,
flag=1.''')
opts, args = p.parse_args(sys.argv[1:])
     ifargs==[]:
          print 'phase read file is missing.\n Exiting'
          exit()
     else:
          f_name1=args[0]
my_corr=corr.katcp_wrapper.FpgaClient(device_host)

while not (my_corr.is_connected()):
     pass
#flag=1 # for 1st tym run=0,, for upgrade= 1
f_name="phase111"

print "phase read File is %s" %f_name1
fo = open (f_name1,'r')

ifopts.flag==True:
     print "dumping File is %s" %f_name
fi = open (f_name,'r+')
     list4=fi.readlines()
else:
     print "1st dumping File is %s" %f_name
     fi = open (f_name,'w')
```

```
list2 = []
list3 = []
list1= fo.readlines()
R_even = []
R_odd = []
L_even = []
L_odd = []
for i in range (512):
      list3.append(0)

printlen(list1)
printlen(list3)
for i in range(len(list1)/len(list3)):
      for j in range(512):
            list3[j]=list3[j]+float(list1[(i*512)+j])

for j in range(512):
      list3[j]=list3[j]/(len(list1)/len(list3))
#print '%s'%list3
#print '%s'%list4
ifopts.flag==True:
      for i in range (512):
            list3[i]=list3[i]+float(list4[i])
      for i in range(512):
            fi.write('%f\n'%list3[i])
else:
      for i in range(512):
fi.write('%f\n'%list3[i])

for i in range (512):
      if float(list3[i]) < 0:
            list2.append(float(list3[i])+360)
      else:
            list2.append(float(list3[i]))
channel = 0
while channel < 256:
      R_even.append(int(round(list2[channel] * 45.511111111))) #Degree
to Address Mapping Constant for 2^14 depth sin-cos LUT
      channel = channel + 1
      R_odd.append(int(round(list2[channel] * 45.511111111)))
      channel = channel + 1
while channel < 512:
      L_even.append(int(round(list2[channel] * 45.511111111))) #Degree
to Address Mapping Constant for 2^14 depth sin-cos LUT
      channel = channel + 1
      L_odd.append(int(round(list2[channel] * 45.511111111)))
      channel = channel + 1


print 'Start uploading phase value to Memory!!'
```

```
my_corr.write_int("write_en",1)
print "write_enable %i" %my_corr.read_int("write_en")

my_corr.write_int("write_en1",1)
print "write_enable1 %i" %my_corr.read_int("write_en1")

addr =1
while (addr< 128):                                  # Depth of block RAMs
storing the phase value i.e. 2*128=256 channels
     my_corr.write_int("count_soft", addr)
#    print "Channel Number for R %i" %my_corr.read_int("count_soft")

     my_corr.write_int("count_soft1", addr)
#    print "Channel Numberfor L %i" %my_corr.read_int("count_soft1")

     my_corr.write_int("phase_even", R_even[addr])
#    print "Channel Phase for R_even %i"
%my_corr.read_int("phase_even")

     my_corr.write_int("phase_even1", L_even[addr])
#    print "Channel Phase  for L_even%i"
%my_corr.read_int("phase_even1")

     my_corr.write_int("phase_odd", R_odd[addr])
#    print "Channel Phase for R_odd %i" %my_corr.read_int("phase_odd")

     my_corr.write_int("phase_odd1", L_odd[addr])
#    print "Channel Phase for L_odd %i"
%my_corr.read_int("phase_odd1")

     addr = addr +1

my_corr.write_int("write_en",0)
print "write_enable %i" %my_corr.read_int("write_en")

my_corr.write_int("write_en1",0)
print "write_enable1 %i" %my_corr.read_int("write_en1")

print "Phase Update done Successfully !!"

fo.close()
fi.close()
my_corr.stop()
------------------------------------------------------------
```

Appendix I

Depacketizing C program for 32 bit data size
Original file name: <depckt_4data_32bit.c>

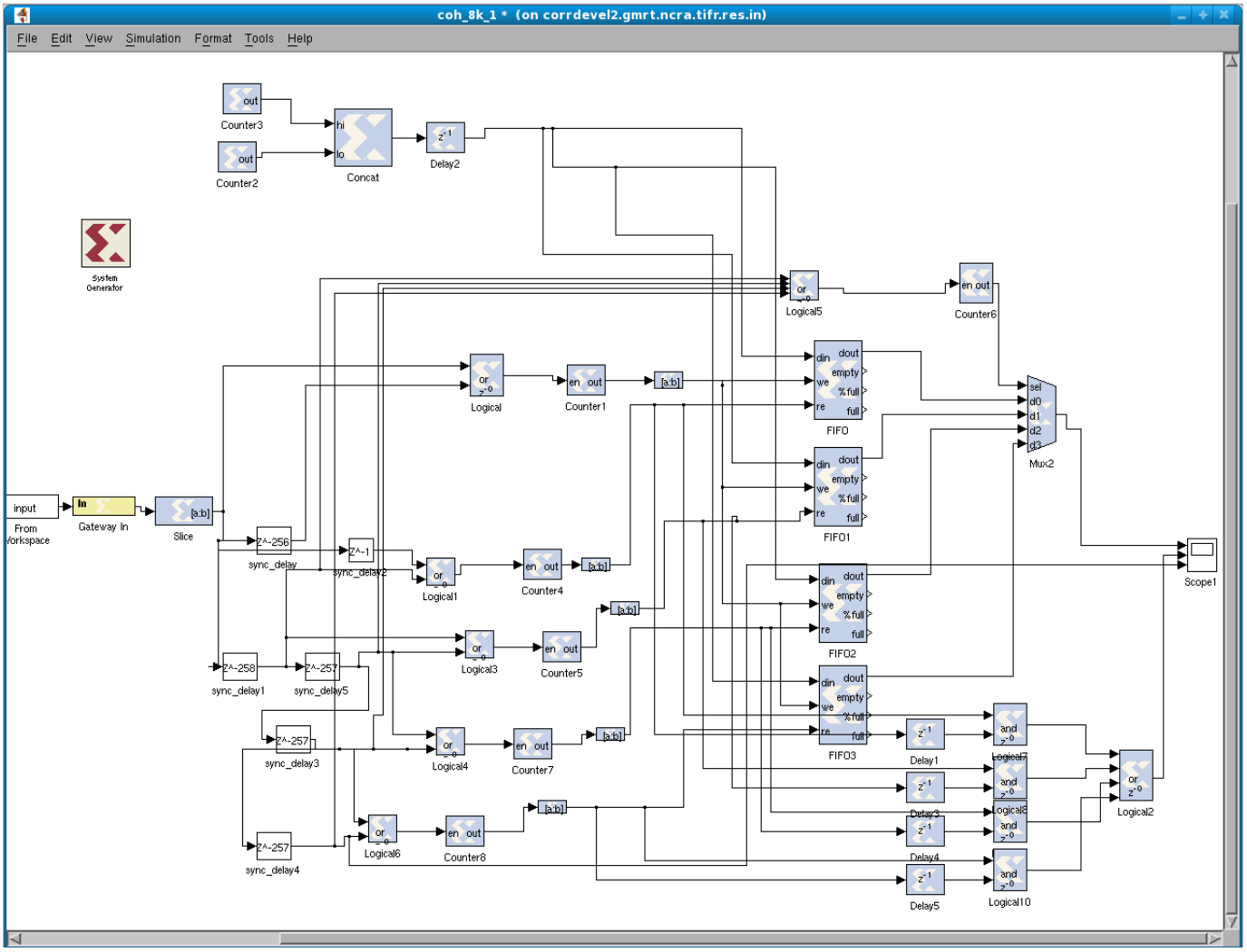
```
------------------------------------------------------------
// syntax:
<object><file_read><file1_write_data1><file2_write_data2><file3_write_data3><
file4_write_data4><file4_write_data4><pack_size><scale>
// the progwil work for 4 data to be written in 4 different files
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdint.h>
int main(intargc,char* argv[])
{
longlsize;
int i,i1,i2,i3,i4,i5,m=0;
i1=i2=i3=i4=i5=0;
int k;
int j;
unsigned short int r;
size_tresult,write;
size_t scale;
size_tpack_size;
pack_size = atoi(argv[7]);
printf("pack_size is %d\n",pack_size);
scale = atoi(argv[8]);
FILE *ha1 = fopen(argv[2],"w");
FILE *ha2 = fopen(argv[3],"w");
FILE *ha3 = fopen(argv[4],"w");
FILE *ha4 = fopen(argv[5],"w");
FILE *ha5 = fopen(argv[6],"w");
FILE *file = fopen(argv[1],"r");

        /* fopen returns 0, the NULL pointer, on failure */
if ( file == 0 )
      {
fputs ("File error",stderr);
exit (1);
      }
else
 {
fseek (file , 0 , SEEK_END);
lsize = ftell (file);
rewind (file);
printf("lsize= %d \n",lsize);

signed short int* buffout = (signed short int*)
malloc((lsize/4)*sizeof(signed short int));
unsigned char* buffer = (unsigned char*) malloc(lsize*sizeof(unsigned char));
signed short int* buffer1 = (signed short int*)
malloc((lsize/16)*sizeof(signed short int));
signed short int* buffer2 = (signed short int*)
malloc((lsize/16)*sizeof(signed short int));
```

```
signed short int* buffer3= (signed short int*)
malloc((lsize/16)*sizeof(signed short int));
signed short int* buffer4 = (signed short int*)
malloc((lsize/16)*sizeof(signed short int));
if (buffer == NULL)
{
fputs ("Memory error",stderr);
exit (2);
}
else
 {

result = fread(buffer,sizeof(unsigned char),lsize,file);
printf("fread result   %d\n",result);
if((lsize%pack_size) != 0)
printf("file is not in order of packet size\n");
else
printf("no of packet in file %i\n",lsize/pack_size);

for(i=0;i<lsize/pack_size;i++)
  {
unsigned long int temp=0;
for(k = 0;k<pack_size;k=k+4)

     {
unsigned long int u = 0;
signed short int v = 0;
temp = (unsigned long int) buffer[i*pack_size + k];
temp = temp << 24;
       u += (unsigned long int) temp;
temp = (unsigned long int) buffer[i*pack_size + k + 1];
temp = temp << 16;
       u += (unsigned long int) temp;
temp = (unsigned long int) buffer[i*pack_size + k + 2];
temp = temp << 8;
       u += (unsigned long int) temp;
temp = (unsigned long int) buffer[i*pack_size + k + 3];
       u += ( unsigned long int) temp;
if (u > 2147483647)
          u = u - 4294967296;

if (k>=0 && k<(pack_size/4))
           {v = (signed short int) (u/scale);
            //fprintf(ha1,"%u\n",v);
buffer1[i1]=v; i1++;}
else if (k>=(pack_size/4) && k<(1*pack_size/2))
           {v = (signed short int) (u/scale);
            //fprintf(ha2,"%u\n",v);
buffer2[i2]=v; i2++;}
else if (k>=(pack_size/2) && k<(3*pack_size/4))
           {v = (signed short int) (u/scale);
             //fprintf(ha3,"%u\n",v);
buffer3[i3]=v; i3++;}
else if (k>=(3*pack_size/4) && k<(1*pack_size/1))
           {v = (signed short int) (u/scale);
           //fprintf(ha4,"%u\n",v);
buffer4[i4]=v; i4++;}
```

```
     }
   }
for(j = 0;j<i1;j=j+1)
      {
buffout[m]=buffer1[j];m++;
        r=(unsigned short int)buffer1[j];
        //fprintf(ha5,"%u\n",r);i5++;

buffout[m]=buffer4[j];m++;
        r=(unsigned short int)buffer2[j];
        //fprintf(ha5,"%u\n",r);i5++;

buffout[m]=buffer2[j];m++;
        r=(unsigned short int)buffer3[j];
        //fprintf(ha5,"%u\n",r);i5++;

buffout[m]=buffer3[j];m++;
        r=(unsigned short int)buffer4[j];
        //fprintf(ha5,"%u\n",r);i5++;
      }

fwrite(buffer1,sizeof(signed short int),i1,ha1);
fwrite(buffer2,sizeof(signed short int),i2,ha2);
fwrite(buffer3,sizeof(signed short int),i3,ha3);
fwrite(buffer4,sizeof(signed short int),i4,ha4);
write=fwrite(buffout,sizeof(signed short int),m,ha5);
printf("fwrite result = %d\n",write);


free(buffer);free(buffer1);free(buffer2);free(buffer3);free(buffer4);free(buf
fout);

printf("filesize of
f1,f2,f3,f4=%d,%d,%d,%d\n",ftell(ha1),ftell(ha2),ftell(ha3),ftell(ha4));
printf("bin.file size=%d\n",ftell(ha5));
}
}
fclose(ha1);
fclose(ha2);
fclose(ha3);
fclose(ha4);
fclose(ha5);
fclose(file);
printf(" i1=%i , i2=%i , i3=%i  , i4=%i , m=%i\n",i1,i2,i3,i4,m);
return 0;

}
```

Appendix J

The packetization technique for making the 10GbE packets.



Appendix K

This is the roach board used for Experiment and implementation.

Appendix L

The final design of 2 antennas 2 polarization coherent PoBe model for single FPGA