PROJECT REPORT

ON

FRINGE TESTER FOR PARABOLIC ANTENNA BACKENDS

SUBMITTED BY

Mr. Shrikant Chaudhari.

Mr. Sanket Bhansali.

Under the Guidance of

Mr. Ajith Kumar.

Mrs. Kodgirwar.

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION

P.E.S'S MODERN COLLEGE OF ENGINEERING

PUNE – 411005

SAVITRIBAI PHULE PUNE UNIVERSITY

2014 – 15

SAVITRIBAI PHULE PUNE UNIVERSITY
2014 - 15

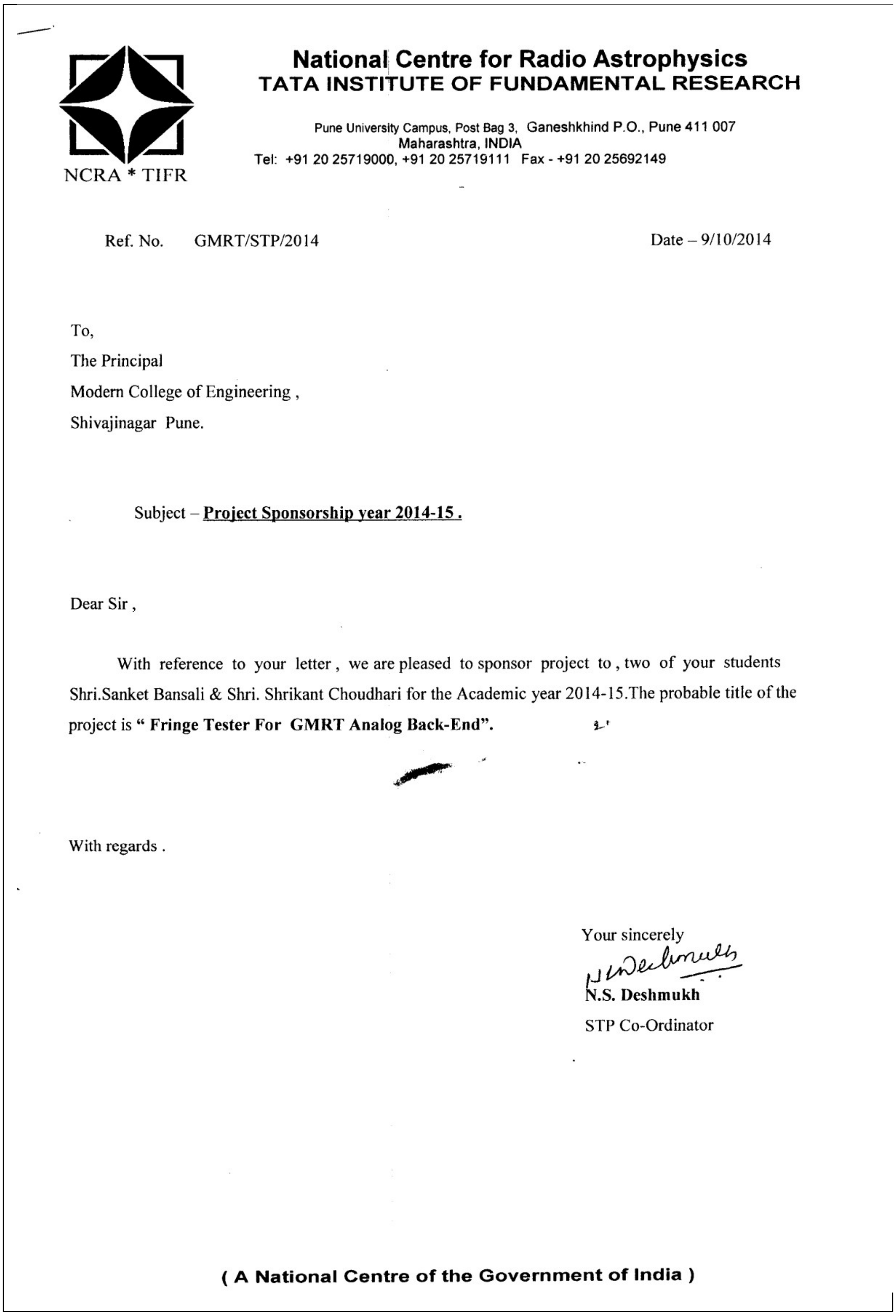
CERTIFICATE

This is to certify that

Mr.Shrikant Chaudhari	**Exam No. B80313023**
Mr.Sanket Bhansali	**Exam No. B80313013**

of B.E. (E&TC) have successfully completed the project titled '**FRINGE TESTER FOR PARABOLIC ANTENNA BACKENDS'** during the academic year 2014-15. This report is submitted as fulfillment of the BE project of degree in E&TC Engineering as prescribed by University of Pune.

Principal	**H.O.D.**	**Project Guide**	**Project Guide**
P.E.S's MCOE, Pune-5	**E&TC**	**Mr. Ajith Kumar**	**Mrs. Kodgirwar**
		(GMRT)	**(College)**



National Centre for Radio Astrophysics
TATA INSTITUTE OF FUNDAMENTAL RESEARCH

NCRA * TIFR

Pune University Campus, Post Bag 3, Ganeshkhind P.O., Pune 411 007
Maharashtra, INDIA
Tel: +91 20 25719000, +91 20 25719111 Fax - +91 20 25692149

Ref. No. GMRT/STP/2014 Date – 9/10/2014

To,
The Principal
Modern College of Engineering ,
Shivajinagar Pune.

Subject – **Project Sponsorship year 2014-15 .**

Dear Sir ,

With reference to your letter , we are pleased to sponsor project to , two of your students Shri.Sanket Bansali & Shri. Shrikant Choudhari for the Academic year 2014-15.The probable title of the project is **" Fringe Tester For GMRT Analog Back-End".**

With regards .

Your sincerely
[illegible]
N.S. Deshmukh
STP Co-Ordinator

(A National Centre of the Government of India)

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our guide teacher Mr. Ajith Kumar and also Mrs. Kodgirwar as well as our project coordinator Mrs. A.P. Laturkar and Mrs. Kanitkar who gave us wonderful opportunity to do this project "FRINGE TESTER", which also helped us in doing lot of research and we come to know about many new things we really thankful to them.

We are highly indebted to GMRT for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project. We would like to express my special gratitude and thanks to industry persons for giving us such attention and time.

Shrikant Chaudhari
Sanket Bhansali

ABSTRACT

As a part of the GMRT upgrade efforts, Frond-end and Back-end systems are undergoing major changes to achieve the upgrade system specifications, like increased bandwidth of 400MHz, direct processing of RF signals, increased dynamic range, improved channel resolution in the digital backend etc. In the process of up-gradation/development the testing and debugging of the system is an important aspect. Hence it is important to build a handy test equipment to quickly check the integrated system and debug the same at various interfacing levels. The handy test equipment in this case is called "Fringe Tester" which is basically a two antenna correlator system built using the CASPER (**C**ollaboration for **A**stronomy **S**ignal **P**rocessing and **E**ngineering **R**esearch) community technology. The aim of the CASPER community 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.

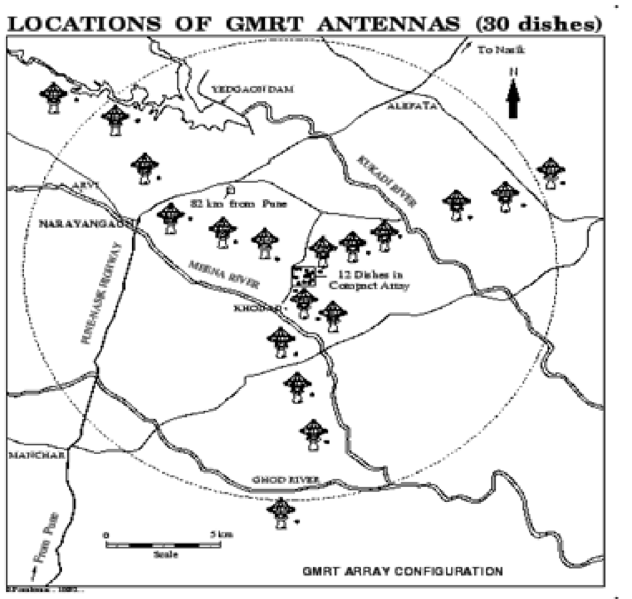
The Aim of the project is to build an analog backend fringe tester (a 2 Antenna Correlator) which is a handy and pluggable instrument using ROACH board (A custom board made by CASPER community and their custom made libraries (MSSGE [MATLAB-Simulink, System Generator & EDK) which can be used to quickly test the correlation between two antennas at the output of new GMRT Analog Backend (GAB) system.

The project involves design of 2-channel correlator viz. Fringe Tester using MSSGE tool flow with its existing DSP and I/O libraries, develop python control package for controlling the design and test the design with antenna signals. The design can be used as a debugging tool in GMRT for fringe (Antenna Correlation) related problems.

Table of Contents

Chapter 1. Introduction

1.1 Introduction to GMRT



Giant Meterwave Radio Telescope (GMRT), located near Pune in India, is an arrayof radio telescopes at meter wavelengths. It is operated by the **National Centre for Radio Astrophysics (NCRA)**, a part of the **Tata Institute of Fundamental Research**, Mumbai. Atthe time it was built, it was world's largest interferometric array. One of the important aims of the telescope is to search for the highly red shifted 21-cm line radiation from primordial neutral hydrogen clouds in order to determine the epoch of galaxy formation in the universe. Pulsar research is another major area for GMRT study. GMRT consists of 30 fully steerable gigantic parabolic dishes of 45m diameter each spread over distances of upto 25 km. The number and configuration of the dishes was optimized to meet the principal astrophysical objectives which require sensitivity at high angular resolution as well as ability to image radio emission from diffuse extended regions. Fourteen of the thirty dishes are located more or less randomly in a compact central array in a region of about 1 square km. Remaining sixteen dishes are spread out along the 3 arms of an approximately `Y'-shaped configuration over a much larger region, with the longest interferometric baseline of 25 km.

The multiplication or correlation of radio signals from all the 435 possible pairs of antennas or interferometers over several hours will thus enable radio images of celestial objects to be synthesized with a resolution equivalent to that obtainable with a single gigantic dish 25Km in diameter. The array will operate in six frequency bands centered around 50, 153, 233, 325, 610 and 1420MHz. All these feeds provide dual polarization outputs. In some configurations, dual-frequency observations are also possible.

1.2 Introduction to CASPER

CASPER (Collaboration for Astronomical Signal Processing and Electronics Research) is an international collaboration whose primary goal is to streamline and simplify the design flow of radio astronomy instrumentation by promoting design reuse through the development of platform-independent, open-source hardware and software. 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.

Collaborating institutes:

1. UC Berkeley Radio Astronomy Lab Berkeley, California
2. GMRT, India
3. MeerKAT / Karoo Array Telescope / SKA – SA, Cape Town, South Africa
4. .....
5. .....

The list goes on….!

Chapter 2. Fringe Tester: Specifications, Literature Survey and Hardware Selection

2.1 Fringe Tester Specifications:

- **Digital System:-**

✓ ADC No. of Bits :	8 bits
✓ Bandwidth :	400 MHz
✓ FFT No. of Channels :	1024

- ✓ Integer time delay correction
- ✓ Fringe rotation (Fractional Delay correction)
- ✓ Integration Time ~ 1 second

2.2 Literature Survey

The CASPER community has developed following processing Boards (Hardware):-

1. IBOB (2005 - present | Virtex-II Pro)
2. BEE2 (2005 - present | 5x Virtex-II Pro)
3. ROACH (2009 - present | Virtex 5 SXT95/LXT110/LXT155)
4. ROACH2 (Virtex 6)

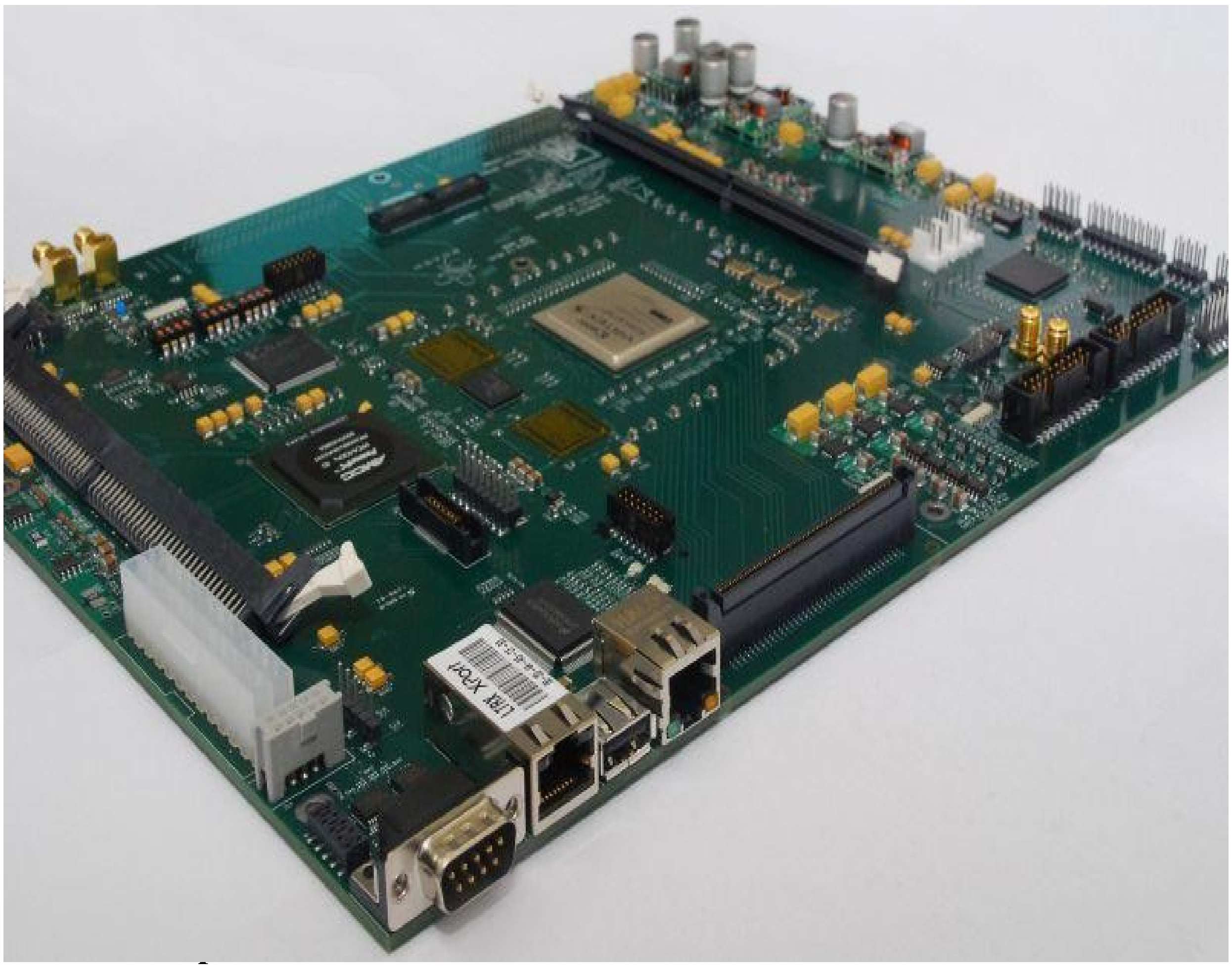
The CASPER community has also developed corresponding software/gateware libraries for the above boards.

- **IBOB:-**The IBOB (**I**nterconnect **B**reak-**o**ut **B**oard") is an FPGA-based processing board for DSP.
- ✓ FPGA:-1x Xilinx Virtex-II Pro XC2VP50-7FF1152 FPGA
- ✓ Interfaces:-
  - 2x Z-DOK+ 40 differential pair connectors
  - 2x CX4 10Gbps high-speed serial connectors
  - 1x 10/100 RJ45 Ethernet interface
  - 1x RS232 interface
  - 2x SMA IO
- ✓ Peripherals:- 2x 512k x 36-bit SRAMs

- **BEE2:-**The BEE2 (**B**erkeley **E**mulation Engine) is the current CASPER workhorse signal processing platform.
- ✓ FPGA:- 5x Xilinx Virtex-II Pro XC2VP70-7FF1704 FPGA
- ✓ Interfaces:-
  - 18x CX4 10Gbps high-speed serial connectors
  - 1x 10/100 RJ45 Ethernet interface
  - 1x RS232 interface

- ✓ Peripherals:-
  - o 20x DDR2 DIMMs
  - o 1x CompactFlash socket

- **ROACH(Reconfigurable Open Architecture Computing Hardware):-** ROACH is a Virtex5-based upgrade to current CASPER hardware. It merges aspects from the IBOB and BEE2 platforms into a single board.
- ✓ FPGA:-1x Xilinx Virtex-5 XC5VLX110T-1FF1136 or Virtex-5 XC5VSX9-1FF1136FPGA
- ✓ Interfaces:-
  - o 2x Z-DOK+ 40 differential pair connectors
  - o 4x CX4 10Gbps high-speed serial connectors
  - o 1x QSH 40 differential pair connector
  - o 16x GPIO
  - o 4x SMA IO (2x clock-capable)
  - o 1x USB2.0
  - o 1x RS232 DB9 serial port
  - o 1x 10/100/1000Mbit RJ45 Ethernet
- ✓ Peripherals:
  - o 2x 2M x 18-bit QDRII+ SRAMs
  - o 1x DDR2 DRAM DIMM
- ✓ CPU:- 1x AMCC PowerPC 440EPx Embedded Processor.
- ✓ 1x MMC/SD card socket



- Board:- ROACH stands for Reconfigurable Open Architecture Computing Hardware board is astandalone FPGA processing board build around Xilinx Virtex-5 (SX95T for DSP-slice-intensive applications).
- iADC board :- 1x Atmel/e2V AT84AD001B 8-bit dual 1Gsps, with clock 10MHz –1GHz 50Ω 0dBm.
- Control Computer.

2.3 Hardware Selection

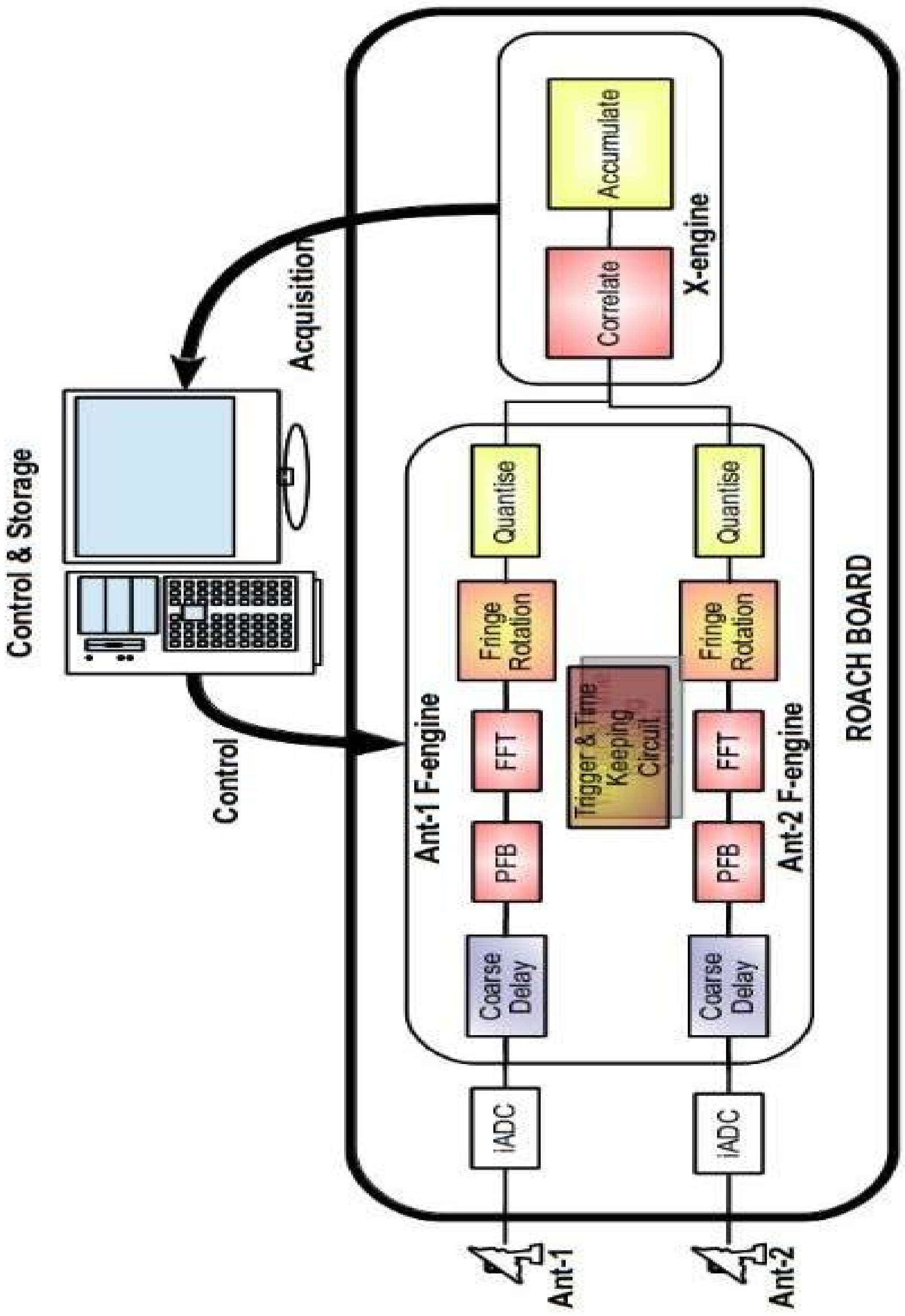
GMRT already started using ROACH board for the fringe tester project due to following reasons:-

- ROACH is Handy and compact board as compared to other board viz. iBOB& BEE-2.
- Virtex-5 FPGA of ROACH boards have sufficient BRAMs to cater GMRT’s max geometric delay correction need.
- Onboard self-bootable Power PC that acts as an interface between FPGA and user.
- The CASPER (Collaboration for Astronomy Signal Processing and Electronics Research) is the international community to streamline and simplify the design flow of radio astronomy instrumentation.
- They develop their custom made high speed FPGA boards and corresponding library based on MSSGE(MATLAB-Simulink-SystemGenerator & EDK).
- ROACH (**R**econfigurable **O**pen **A**rchitecture **C**omputing **H**ardware) board is one of the boards developed by CASPER community. It is a Vertex 5 SXT95 based board which has various peripherals including PPC440EPx as embedded processor that acts as an interface between FPGA design and programmer. iADC board is a 1x Atmel/e2V AT84AD001B 8-bit dual 1Gsps, with clock 10MHz – 1GHz 50Ω 0dBm.

Software/Tool flow:-

- **CASPER MSSGE Toolflow**: -
  1. MATLAB-Simulink
  2. System Generator
  3. Xilinx 11.5 - EDK
  4. Ubuntu Linux 11.10 (Kernel 3.0.0-17-server on x86_64)
  5. Python 2.7 or above & packages

2.4 Block Diagram



Chapter 3. CASPER Toolflow: Getting Started

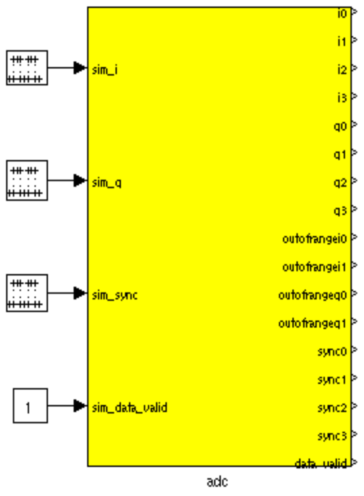
3.1 Tutorial-1: Capture iADC data using snap block

The design was made to check the output of ADC after digitization using a snap block. The design uses iADC and snap block from CASPER library.

3.1.1 iADC (AT84AD001C):-

- **Description:-**

  The ADC block converts the analog input to digital outputs. It is a functional block parameter. At every clock cycle, input are sampled and digitized to 8 bit binary points and are output by ADC.In this case, we have used adc0 and the clock rate in our case is 804MHz. We have not used the interleave mode.

  Two inputs are present at input terminals of ADC as ‘sim_i’ and ‘sim_q’ which are the pulse generators. These analog signals are used when interleave mode is not selected. Here the pulse types are of time based and sample based. Time based used when variable input step and sample based at fixed step input. The next input used is sim_sync which is used at output to measure the delay. The sim data valid input used to indicate whether the inputs are high or low. If we use interleave mode then only one input is present which is known as sim_in.

  For 2 inputs, the ADC samples both inputs and outputs them in parallel with (i0 - i3) corresponding to input ‘i’ and q0-q3 corresponding to input ‘q’. The ‘out of range’ output indicates whether the output samples are in the given range or out of it. The data valid indicates whether the output is in valid condition.

- **Specifications:-**
  - ✓ Dual ADC with 8-bit Resolution
  - ✓ 1 Gsps Sampling Rate per Channel
  - ✓ 500 mVppAnalog Input (Differential Only)
  - ✓ Power Supply: 3.3V (Analog), 3.3V (Digital), 2.25V (Output)
  - ✓ Interleaving Functions

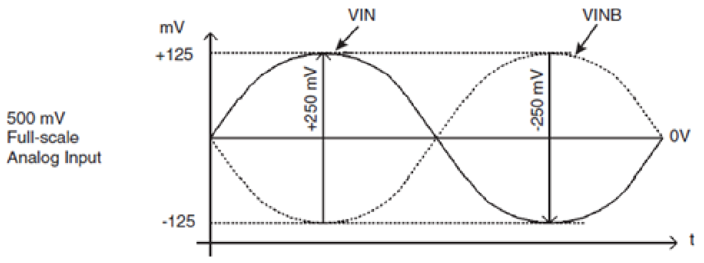
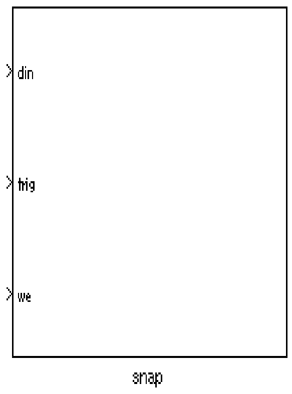


Figure 1: Differential Inputs Voltage Span (Full-scale)

- ✓ Low Power Consumption: 0.75W Per Channel
- ✓ 1.5 GHz Full Power Input Bandwidth (–3 dB)

The analog input full-scale range is 0.5V peak-to-peak (Vpp), or –2 dBm into the 50Ω (100Ω differential) termination resistor. In differential mode input configuration, this means 0.25V on each input, or ±125 mV around common mode voltage.

3.1.2 Snap Block:-



Description:-

The snap block provides a solution to capture data from FPGA and provide it to CPU. Snap block comes in two types as:

- *snap:* it captures 32 bit wide BRAM data.
- *snap64*: it captures 2*32 bit wide BRAM data, used when data is of 64 bit.

The snap block has 3 sub-devices: 'ctrl', 'bram' and 'addr'. 'ctrl' is an input register. Bit 0, when driven from low to high, enables a trigger/data capture to occur. Bit 1, when high, overrides 'trig' to trigger instantly. Bit 2, when high, overrides 'we' to always write data to 'bram'. 'addr' is an output register and records the last address of bram to which data was written. Bram is a 32-bit wide shared BRAM of the depth specified in parameters.

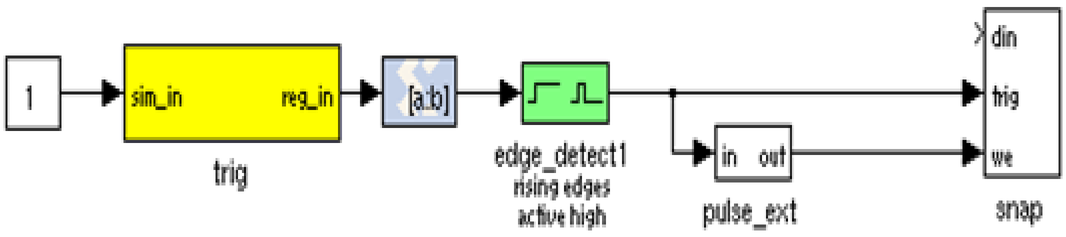


Figure 2: Snap Block Control Circuit

3.1.3 Hardware Design:-

The ADC samples the input data and converts it into 4 parallel streams of 1/4th of ADC sample frequency. The snap captures the data in its BRAM as the trigger pin transits to high state from low and captures the data till BRAM gets full in its high state. We write the i0 sample in BRAM and read it using python language and also tried to plot the waveform from the stored samples in BRAM.

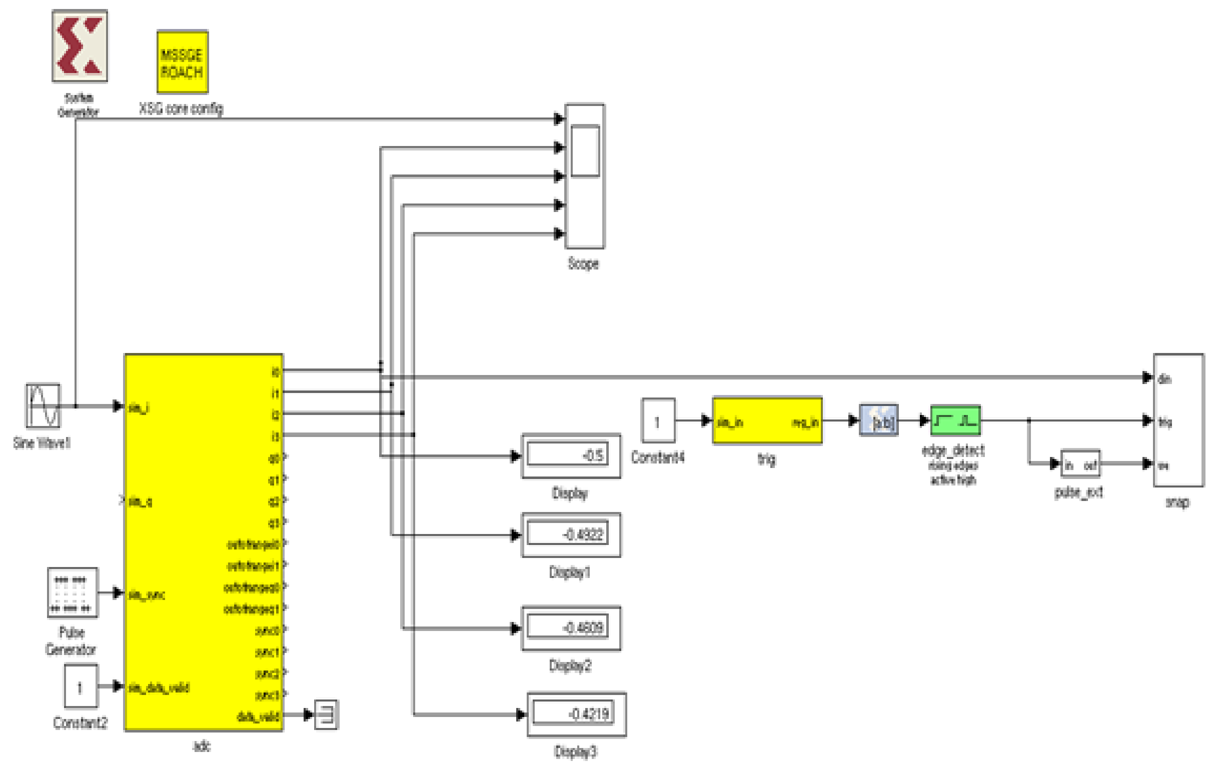
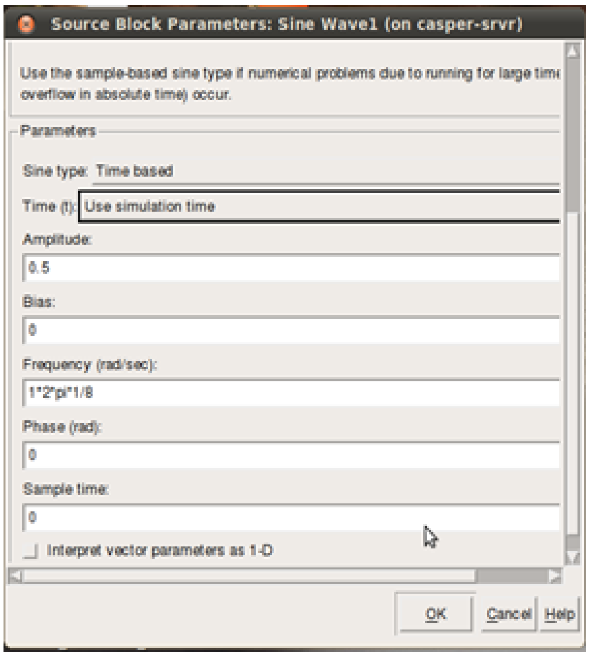


Figure 3: Capture iADC data using snap block design

3.1.4 Simulation:-

For simulation purpose we fed a sine wave to 'i' and 'q' channel with a frequency of 1/8th time period and simulated the result for a period of 2^{12} FPGA clocks. The simulation input sine wave parameters are as follows:-


Source Block Parameters: Sine Wave1 (on casper-srvr)

Use the sample-based sine type if numerical problems due to running for large time overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 0.5

Bias: 0

Frequency (rad/sec): 1*2*pi*1/8

Phase (rad): 0

Sample time: 0

Interpret vector parameters as 1-D

Figure 4: ADC Output Testing Using Snap Block

3.1.5 Hardware Testing:-

- ADC Clock = 400MHz.
- ADC input ‘i’ = 1MHz sine wave @ -3dBm power.

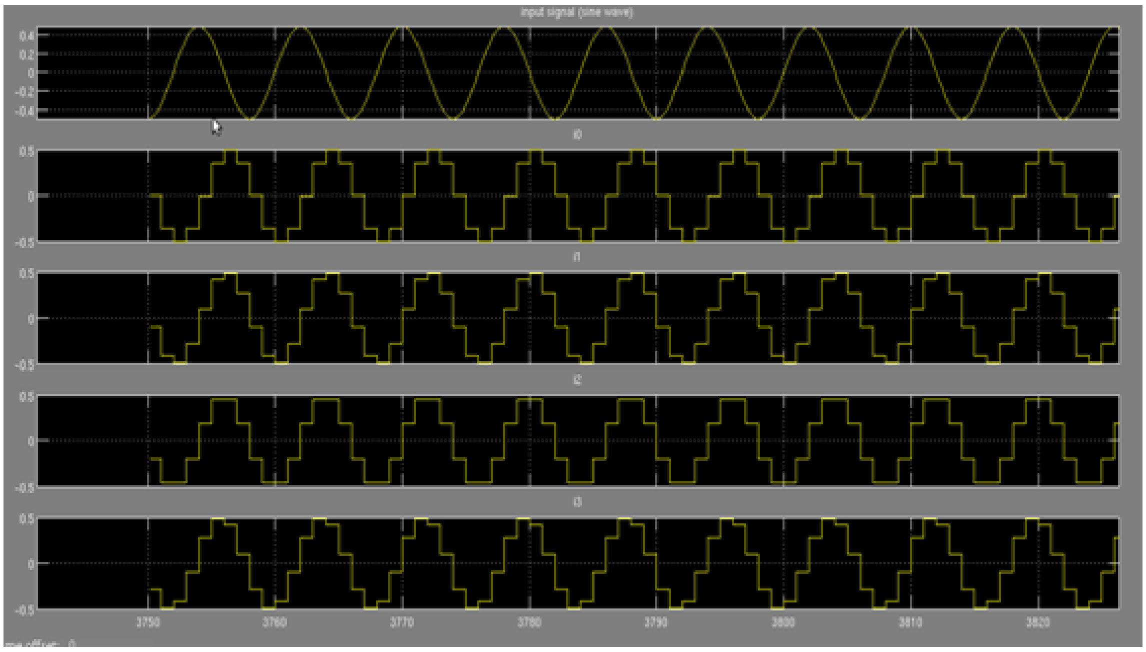


Figure 5: ADC Simulation Results with appropriate nomenclature

The input given to ADC is 1MHz sine wave at ‘i’ and practical clock of ADC was given to be 400 MHz due to unavailability of 800 MHz waveform generator. In design ‘i0’sampled stream was used to capture in ‘snap’ block.ADC date when initially captured with the python script somehow looked as shown below:-

The waveform was due to the ADC coding technique used by the manufacturer i.e. offset binary coding where the ADC input range of (-250mV to +250mV) was denoted by

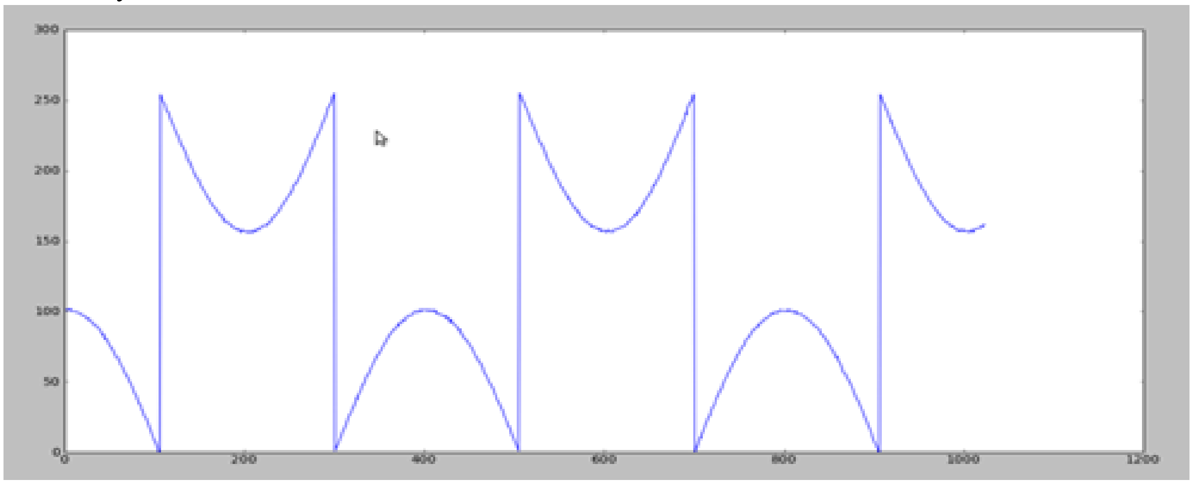


Figure 6: ADC snap block captured output

(-128 to +127). Hence the same was corrected in python script itself as shown below with bold letters.

After binary offset correction waveform looks like shown below:-

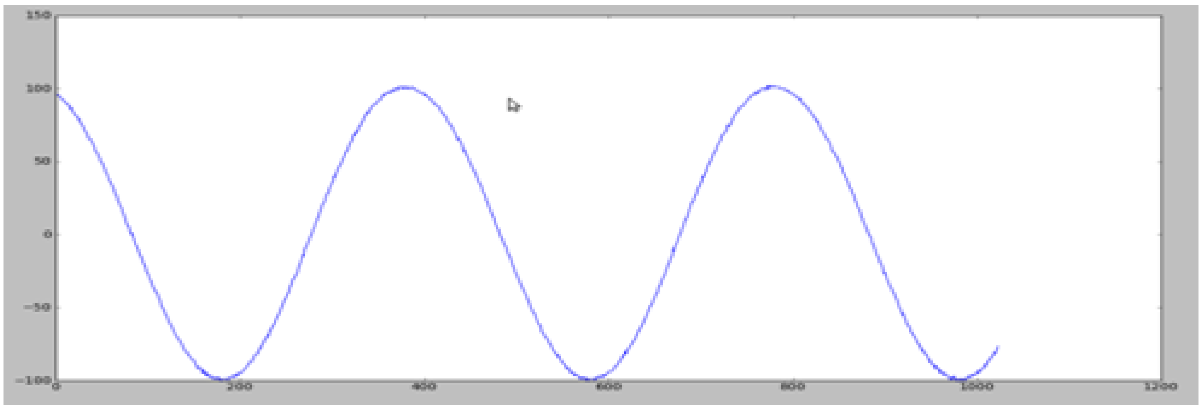


Figure 7: ADC output after binary offset correction

3.2 Tutorial-2: Design a Basic Pocket Correlator

Basic pocket correlator is a first step towards building a fringe tester. The design consists of following blocks:

1. iADC block
2. wideband delay block
3. PFB fir block
4. FFT block
5. Quantizer block
6. Accumulator block
7. Accumulation control block
8. Software registers to control correlator functionality.

Basic pocket correlator specifications are as follows:

1. ADC number of bits = 8-bit
2. Processing bandwidth = 400MHz
3. FFT = 1024 point real FFT (FFT block output only 512 channels as other 512 are mirror image and can be eliminated, thus saving the FPGA resources for FFT block)
4. Maximum integer delay correction = 1024 ADC clocks
5. Typical accumulation time = 1 second

3.2.1 Integer Delay Block:-

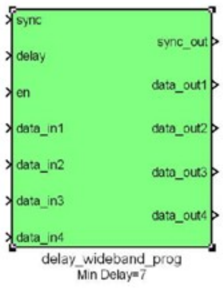


Figure 8: Coarse Delay Block

Coarse delay implies integer delay. we can correct the time delay which is integer multiple of clock pulse and it is corrected by this system hardware. It is configured & controlled by the software program.

Integer Delay Block:-

1. This block compensates the delays in multiples of sampling clock i.e. 800MHz.

2. It basically stores the data samples from ADC into memory which are then readout by giving an offset. This offset is equivalent to the delay in terms of sampling clock.
3. This block can give accuracy up to the sampling clock.

Mask Parameters:-

Parameter	**Variable**	**Description**
Max Delay	max_delay	The maximum length of delay which can be provided (in sample clock cycles).
Number of simultaneous inputs (2^?)	n_inputs_bits	Number of sequential time series inputs (specified in power of 2) required to the delay block.
BRAM Latency	bram_latency	The latency of the underlying storage BRAM.

Ports:-

Port	**Dir**	**Data Type**	**Description**
delay	in	Unsigned Integer	The runtime programmable delay value.
sync	in	Boolean	Sync pulse to synchronize this delay block with the other blocks in the design.
data_in	in	???	The simultaneous signals to be delayed.
sync_out	out	boolean	Synchronizing output pulse from the delay block.
data_out	out	inherited	The delayed simultaneous outputs.

Description:-

A delay block uses single port BRAM for its storage and has a run-time programmable delay for sequential time series inputs. Maximum delay should be in terms of powers of 2, if not, the block converts the maximum delay provided by user to the nearest power of 2.The minimum acceptable BRAM latency (Single and Dual Port) is 1, by default kept at 4. The functionality of the coarse delay block is to delay the simultaneous input data stream by specified number of clock cycles. This is achieved by writing the input data samples in memory and reading them with an offset equal to the amount of delay required.

Inside DelayBlock:-

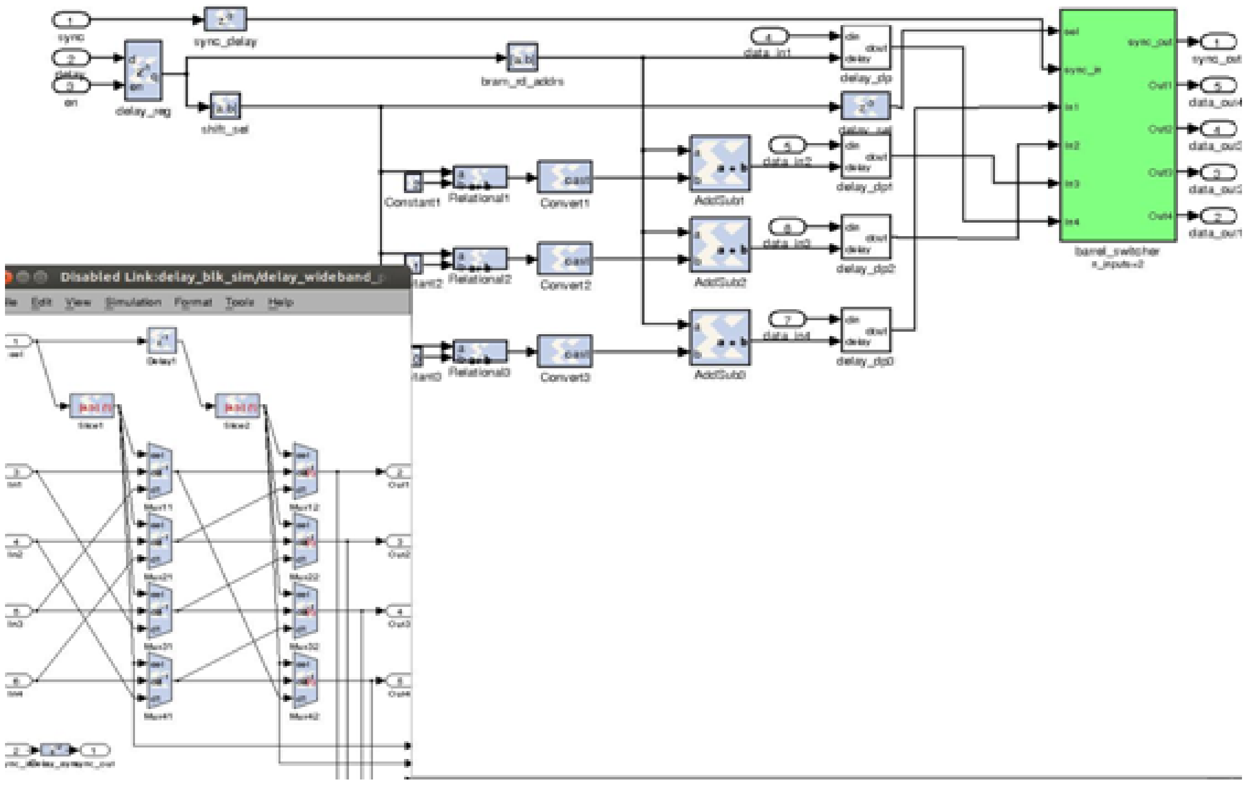


Figure 9: Integer Delay Block Inside

In the present case a delay block with 4 simultaneous input ports and can compensate maximum of 1024 clock cycles is simulated. As the simultaneous inputs are four, hence 4-BRAMs with 256 (i.e. 1024/4) locations are required to store the data. The internal logic is designed in such a way that the delays greater than 4 clock cycles are addressed by certain logic and delays less than 4 clock cycles are achieved via barrel shifter. A 'barrel_switcher' block which maps a number of inputs (in our case number of inputs = 4) to a number of outputs by rotating In(N) to Out(N+M) (where M is specified on the sel input), wrapping around to Out1 when necessary. A 'sync' pulse is provided to synchronize the delay block with rest of the logic in the design.

Integer Delay Block Simulation:-

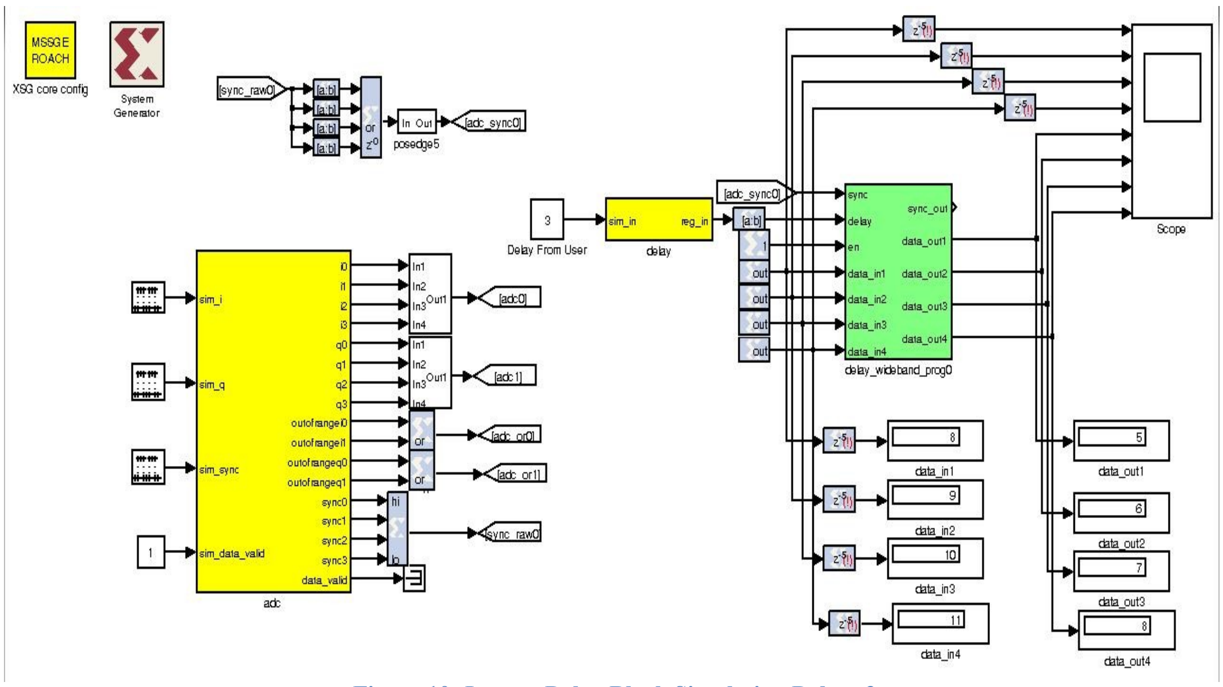


Figure 10: Integer Delay Block Simulation Delay=3

For simulation purpose we added four free running counter blocks (‘counter0’, ‘counter1’, ‘counter2’, ‘counter3’) with step size of 4 and initial value of 0,1,2,3 respectively. The output of each counter is given to four inputs (‘data_in1’, ‘data_in2’, ‘data_in3’, ‘data_in4’) of ‘delay_wideband_prog’ block respectively. The configurable delay was given from a software register block named ‘delay’. Then the design` was simulated for a period of 7 clock cycles. Inputs and corresponding outputs of ‘delay_wideband_prog’ block were shown on ‘numeric display’ blocks connected respectively. According to the value from ‘delay’ register, output was getting delayed w.r.t. input as seen.

3.2.2 Polyphase Filter Bank:-

In digital signal processing, an instrument or software that needs to do Fourier analysis of input signal performs a DFT. The straightforward application of the DFT on an input signal suffers from two significant drawbacks, namely, leakage and scalloping loss .DFT leakage is the phenomenon in which, depending on sampling frequency and the number of points in the transform, an input tone appears in more than one output frequency bin. If this tone is not strong enough, this effect can go unnoticed. But in the case of say a strong radio frequency interference (RFI) signal, the leakage can drown out astronomical signal of interest in the nearby bins. DFT scalloping loss is the loss in energy between frequency bin centers due to the non-flat nature of the single-bin frequency response. The polyphasefilter bank (PFB) technique is a mechanism for minimizing drawbacks of the straightforward DFT. The PFB – followed by the DFT – not only produces a flat response across the channel, but also provides excellent suppression of out of band signals as shown in below Fig. 11

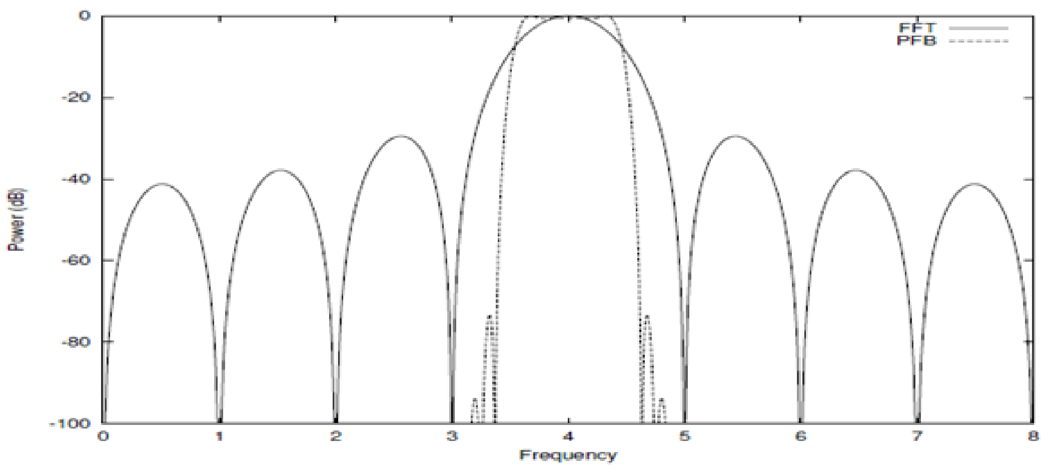


Figure 11: Comparison of single-bin frequency response of a PFB with a direct FFT.

Description:-

This block, combined with an FFT, implements a real Polyphase Filter Bank which uses longer windows of data to improve the shape of channels within a spectrum.

Usage:-

This block, combined with an FFT, implements a real Polyphase Filter Bank which uses longer windows of data to improve the shape of channels within a spectrum.

Mask Parameters:-

Parameter	Variable	Description
Size of PFB (2^{pnts})	PFBSize	The number of channels in the PFB (this should also be the size of the FFT which follows).
Total Number of Taps	Total Taps	The number of taps in the PFB FIR filter. Each tap uses 2 real multiplier cores and requires buffering the real and imaginary streams for $2^{PFBSize}$ samples.
Windowing Function	WindowType	Which windowing function to use (this allows trading passband ripple for steepness of rolloff, etc).
Number of Simultaneous Inputs ($2^?$)	n_inputs	The number of parallel time samples which are presented to the FFT core each clock. The number of output ports are set to this same value.
Make Biplex	MakeBiplex	Double up the inputs to match with a biplex FFT.
Input Bitwidth	BitWidthIn	The number of bits in each real and imaginary sample input to the PFB.
Output Bitwidth	BitWidthOut	The number of bits in each real and imaginary sample output from the PFB. This should match the bit width in the FFT that follows.
Coefficient Bitwidth	CoeffBitWidth	The number of bits in each coefficient. This is usually chosen to match the input bit width.

Ports:-

Port	Direction	Data Type	Description
Sync	IN	Boolean	Indicates the next clock cycle contains valid data
pol_in	IN	Inherited	The (real) time-domain stream(s).
sync_out	OUT	Boolean	Indicates that data out will be valid next clock cycle.
pol_out	OUT	Inherited	The (real) PFB FIR output, which is still a time-domain signal.

3.2.3 FFT_Wideband_Real(Real-sampled Wideband FFT):-

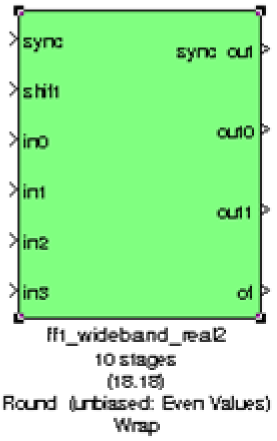


Figure 12: fft_wideband_real block

This is a standard reconfigurable block in CASPER library. It computes the real-sampled Fast Fourier Transform using the standard Hermitian conjugation trick to use a complex core to transform a single real stream using half the normal resources (this requires at least 4 time samples in parallel). Only positive frequencies are output (negative frequencies are the mirror images of their positive counterparts), so there the number of output ports is half the number of input ports. Uses a biplex FFT architecture under the hood which has been extended to handle time samples in parallel. Data is output in normal frequency order, meaning that channel 0 (corresponding to DC) is output first, followed by channel 1, on up to channel $2^{N-1} - 1$.

The four parallel time samples from iADC are pass through the pfb_fir_real and fft_wideband_real blocks, which together constitute a polyphase filter bank. We've selected 2^{10} = 1024 points, so we'll have a 2^{9} = 512 channel filter bank. The FFT_wideband_real block is the most important part of the design to understand. The cool green of the FFT block hides the complex and confusing FFT butterfly biplex algorithms that are under the hood.

Inputs/Outputs:-

Port	Description
Sync	Like many of the blocks, the FFT needs a heartbeat to keep it synced
Shift	Sets the shifting schedule through the FFT. Bit 0 specifies the behavior of stage 0, bit 1 of stage 1, and so on. If a stage is set to shift (with bit = 1), then every sample is divided by 2 at the output of that stage.
in0, in1, in2, in3	Four inputs for the parallel data streams coming from the ADC, through the pfb_fir_real filter block, and into here. Just connect them up.

out0 out1	This real FFT produces two simultaneous outputs. Because it's a real FFT, the spectrum's left and right halves are mirror images and so we don't bother to output the imaginary half (negative channel indices). Thus, for a 1024-point FFT, you get 512 useful channels. That's why there are half the number of parallel outputs (two complex output paths to four real input paths). Each of these parallel FFT outputs will produce sequential channels on every clock cycle. So, on the first clock cycle (after a sync pulse, which denotes the start), you'll get frequency channel zero and frequency channel one. Each of those are complex numbers. Then, on the second clock cycle, you'll get frequency channels 2 and 3. These are followed by 4 and 5 etc etc. So we chose to label these output paths "even" and "odd", to differentiate the path outputting channels 0,2,4,6,8...N-1 from the channel doing 1,3,5,7...N. As you can see, in order to recreate the full spectrum, we need to interleave these paths to produce 0,1,2,3,4,5...N. Following the lines you'll see that these two inputs end up in an "odd" and "even" shared BRAMs.

Mask Parameters:-

Parameter	**Variable**	**Description**	**Recommended Value**
Number simultaneous streams	n_streams	The number of input data streams to be processed in parallel. Each stream consists of a set of parallel inputs set by another parameter (see Number of Simultaneous Inputs)	
Size of FFT: (2^?)	FFTSize	The number of channels computed in the complex FFT core. The number of channels output for each real stream is half of this.	
Input Bit Width	input_bit_width	The number of bits in each real and imaginary sample as they are carried through the FFT. Each FFT stage will round numbers back down to this number of bits after performing a butterfly computation if bit growth is not enabled.	To make optimal use of BRAM => 18 For low FFT noise => 25
Coefficient Bit Width	coeff_bit_width	The number of bits used in the real and imaginary part of the twiddle factors at each stage.	18
Number of Simultaneous Inputs: (2^?)	n_inputs	The number of parallel time samples which are presented to the FFT core each clock. This must be at least 2^2. The number of output ports is half of this value.	

Fringe Rotation:-

It is used to compensate the effect of time delay correction on base band signal. As we are converting RF signal to base band signal time delay correction applied to RF signal cannot be applied to the baseband signal. It is fractional delay correction applied to the base band signal.

3.2.4 Quant Block:-

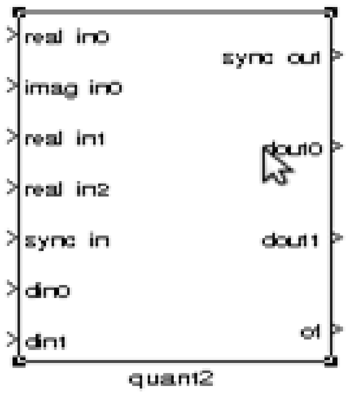


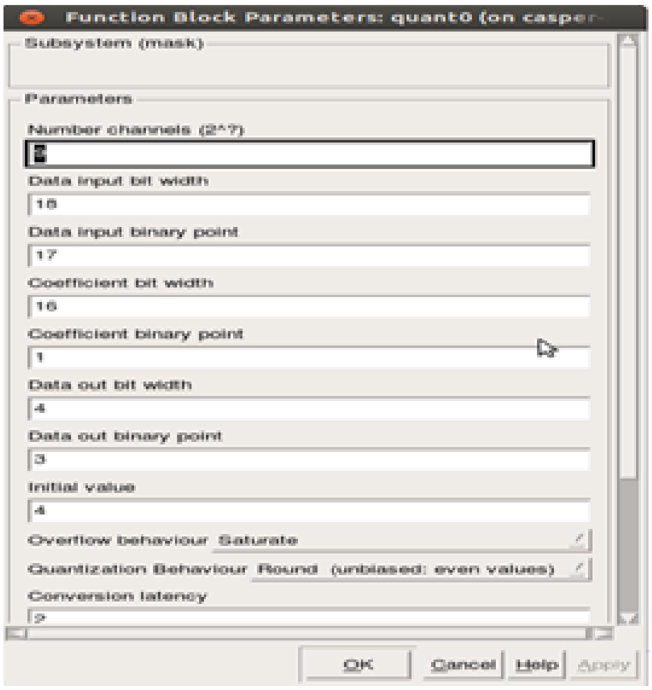
Figure 13: Quant Block

This block is used after FFT block to scale down the bit growth in earlier stages and enable the accumulator to store the accumulation result by minimally sacrificing dynamic range and/or resolution. This block re-quantizes FFT stage output from 18.17 bits to 4.3 bits. The bit growth in 'fft_wideband_real' block is due to the twiddle factor that gets multiplied with the input data at each stages of fft_wideband stage (In our case for 1024-point FFT, it will have 10 FFT stages). This block adjusts the scaling as per user requirement when the signal is too high or too low by minimally sacrificing dynamic range and/or resolution.

INPUTS/OUTPUTS:-

Port	Description
Sync	Input/output for the sync heartbeat pulse.
din0 din1	Data inputs – odd is connected to din0 and even is connected to din1. In our design, data in is 36.34 bits.
dout0 dout1	Data outputs. In this design, the quant0 block requantizes from the 36.34 input to 6.5 bits, so the output on both of these ports is 6.5 unsigned bits.

Functional Block parameters for quant:-



Function Block Parameters: quant0 (on casper-

Subsystem (mask)

Parameters

Number channels (2^?)
[illegible]

Data input bit width
18

Data input binary point
17

Coefficient bit width
16

Coefficient binary point
1

Data out bit width
4

Data out binary point
3

Initial value
4

Overflow behaviour Saturate

Quantization Behaviour Round (unbiased: even values)

Conversion latency
2

Figure 14: Quant Block Funtional Parameters

3.2.5 Accumulator Control Block:-

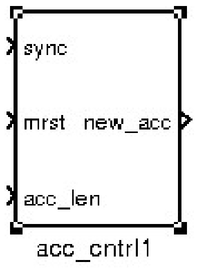


Figure 15: Accumulator Control Block

This block is custom made block that generates a ‘new_acc’ pulse on completion of specified numbers of fft cycles provided on input port ‘acc_len’ of the block. ‘sync’ is a marker signals whose description is given in section ----- and ‘mrst’ is a signal that resets the complete system.

Output of the ‘acc_ctrl’ block goes to the accumulator where self correlation and cross correlation is done and at the end of each accumulation i.e. on every ‘new_acc’ pulse the results of self and cross correlation gets stored in ‘Shared BRAMs’ of accumulator and then read through python to dump/plot in control computer.

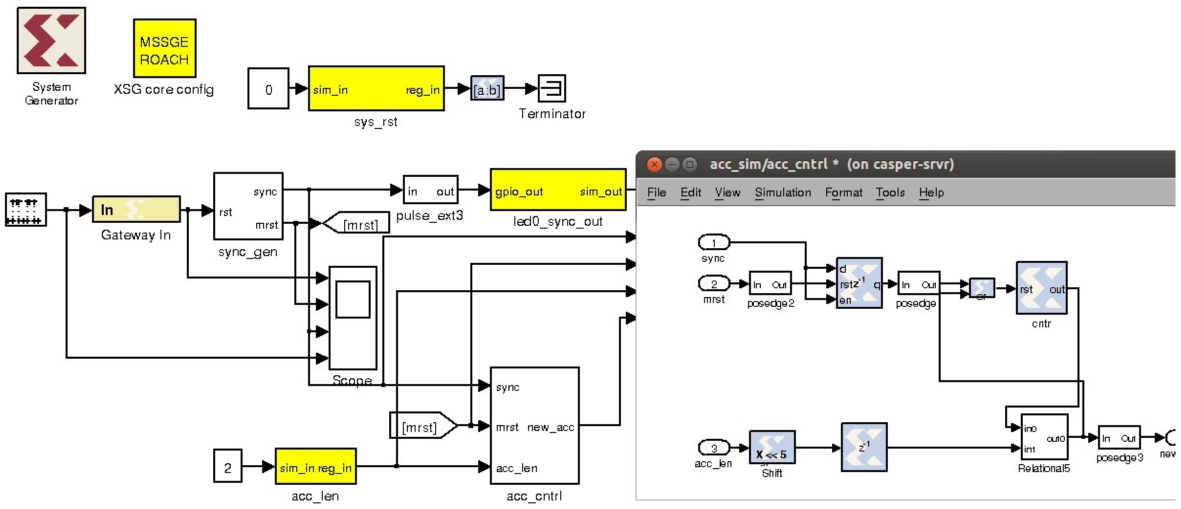
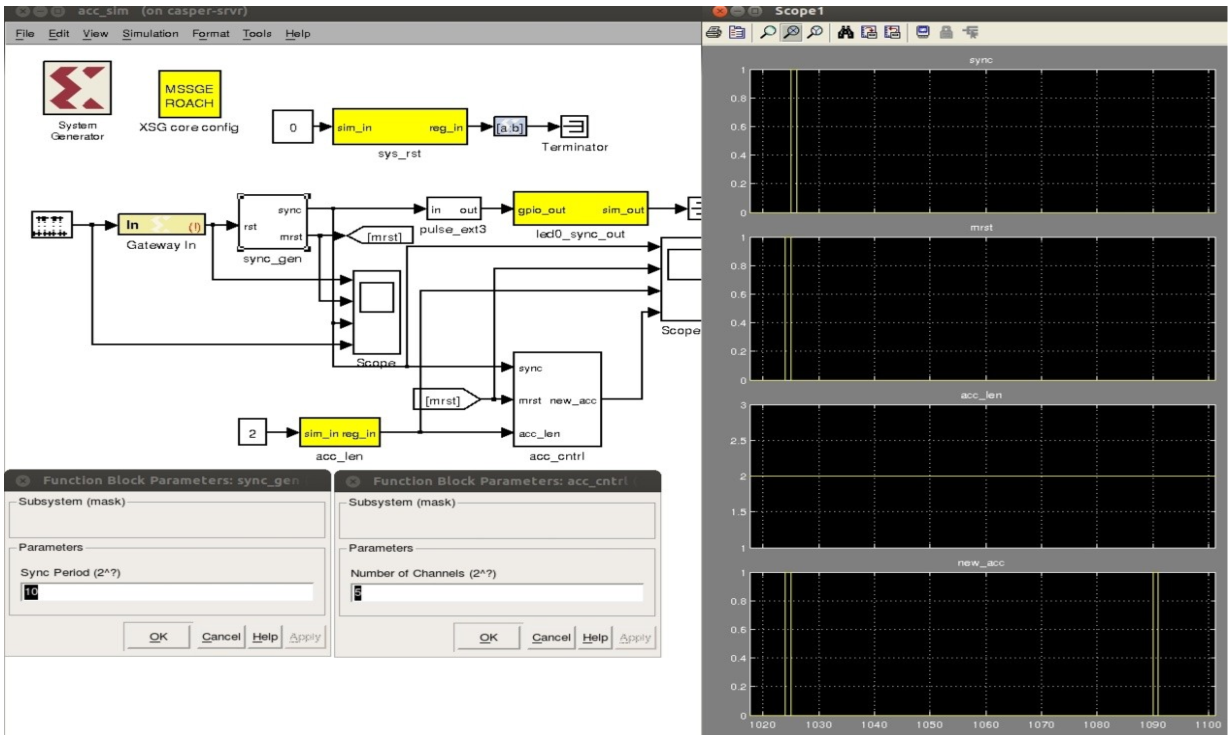


Figure 16: Accumulation Control Block Inside

Internally the block is designed to get reset on master reset and then onwards it counts specified number of fft cycles, here in above figure 'acc_len' = 2 and each fft cycle time for simulation purpose is kept as 32.



Function Block Parameters: sync_gen
Subsystem (mask)
Parameters
Sync Period (2^?)
10

Function Block Parameters: acc_cntrl
Subsystem (mask)
Parameters
Number of Channels (2^?)
5

Figure 17: Accumulation Control Block Simulation

For simulation purpose we scaled down the sync period to 2^{10} FPGA clocks and number of FFT channels were 2^5 i.e. 32 only. Hence 'new_acc' pulse will be generated after (2^4 *2 = 64) FPGA clocks.

3.2.6 Accumulator Block:-

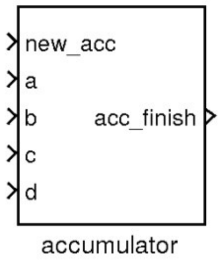


Figure 18: Accumulator Block

Accumulation block calculates self correlation and cross products of 2-Antenna single polarization and finally accumulates their result for specified accumulation time. Internally it consists of multiply and accumulate block for each baseline.
In our case of 2-Antenna singe polarization:
Baselines are 2x3/2 = 3 and each with 2 parallel streams hence 6 blocks
Self of CH-1 : 'aa' , 'bb' with only 'real' BRAM
Self of CH-2 : 'cc' , 'dd' with only 'real' BRAM
Cross of CH-1 & CH-2 : 'ac' , 'bd' with 'real' and 'imag' BRAMs.
Internally each consists of multiply ('cmult_4bit_sl' is a complex multiplier block) and accumulate ('vacc' is accumulate block).

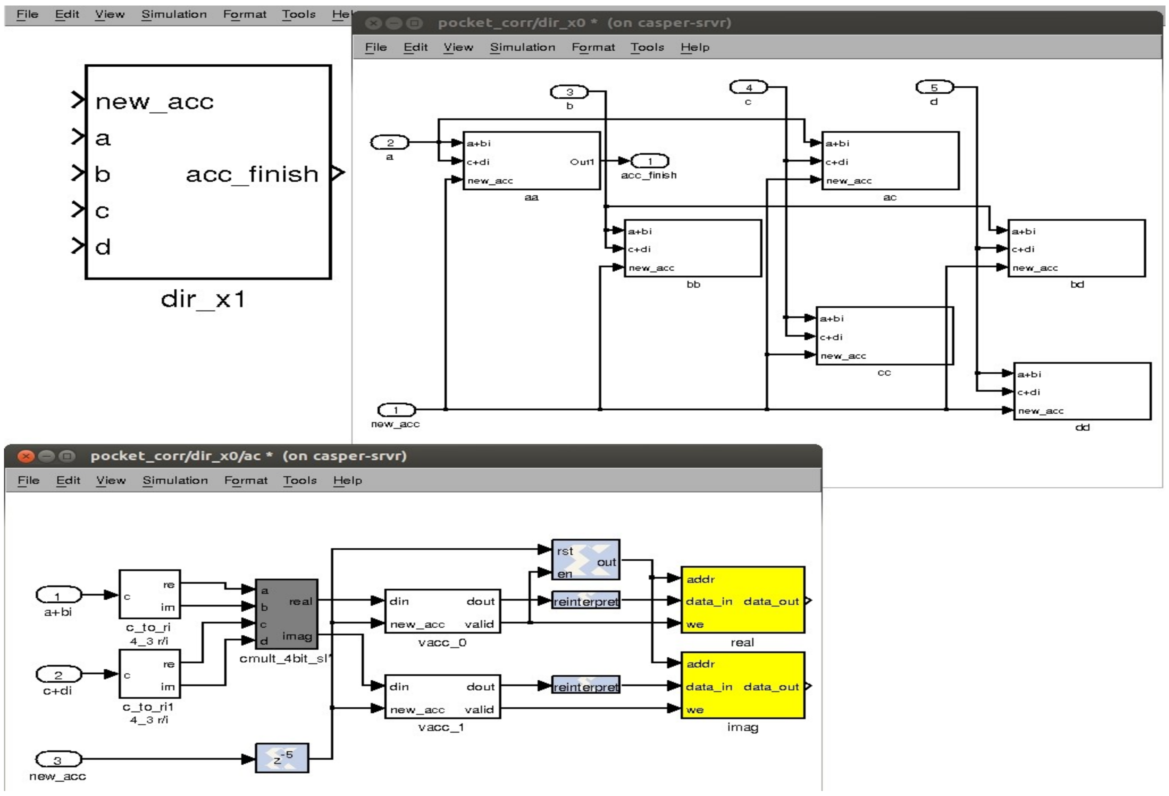


Figure 19: Accumulator Block Internal

At the end of each accumulation cycle the results are then transferred to shared BRAMs in yellow colours and can be read from python through PPC on ROACH board. For self correlations we have to read real BRAMs in each blocks of MAC.

Output of this block is an 'acc_finish' signal and is used to read shared BRAMs (yellow colored) only when asserted high. While readings from python program this signal is an indicator that we are reading only when shared BRAMs are not getting written.

3.2.7 Basic Sync Generator:-

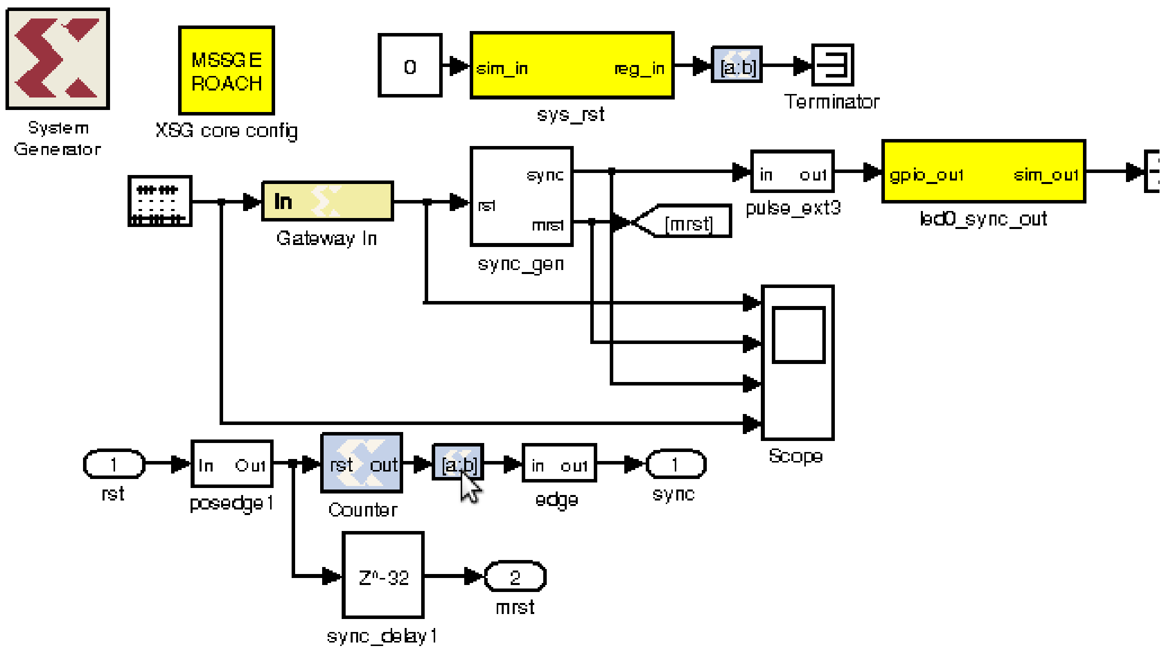


Figure 20: Basic Sync Generator Circuit

The sync pulse is used as a “vector warning signal”meaning that it precedes a frame of data 1 clock cycle prior to the first data sample of the frame. Hence it is used as data alignment signal in CASPER designs. It is also used to denote the propagation delay of each reconfigurable CASPER blocks through which it passes. More detailed discussion on sync generator circuit can be found in section 3.5

Putting all these blocks together we have designed a basic pocket correlator/fringe tester which is a starting point of our project. The design is shown in fig. 21.

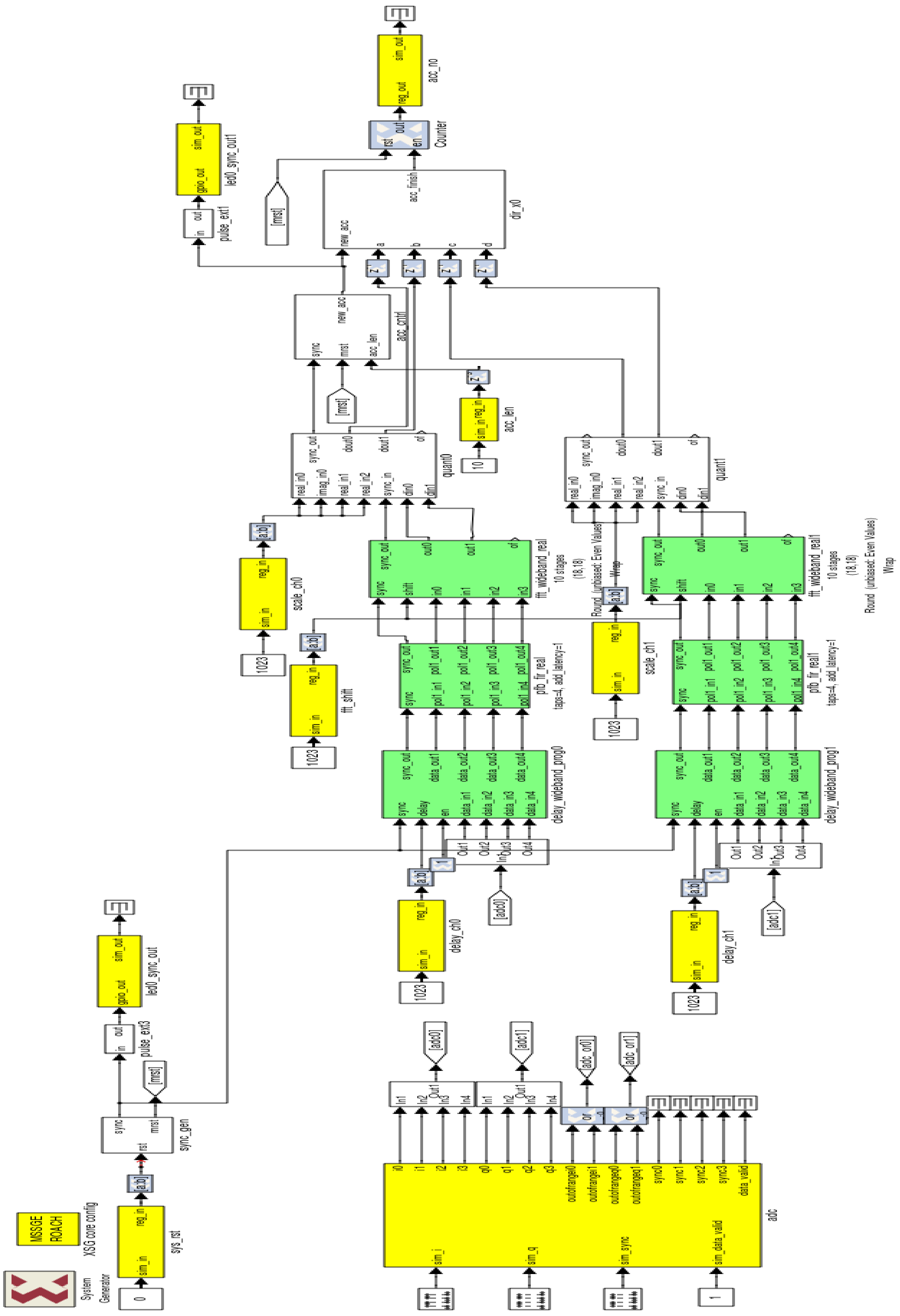
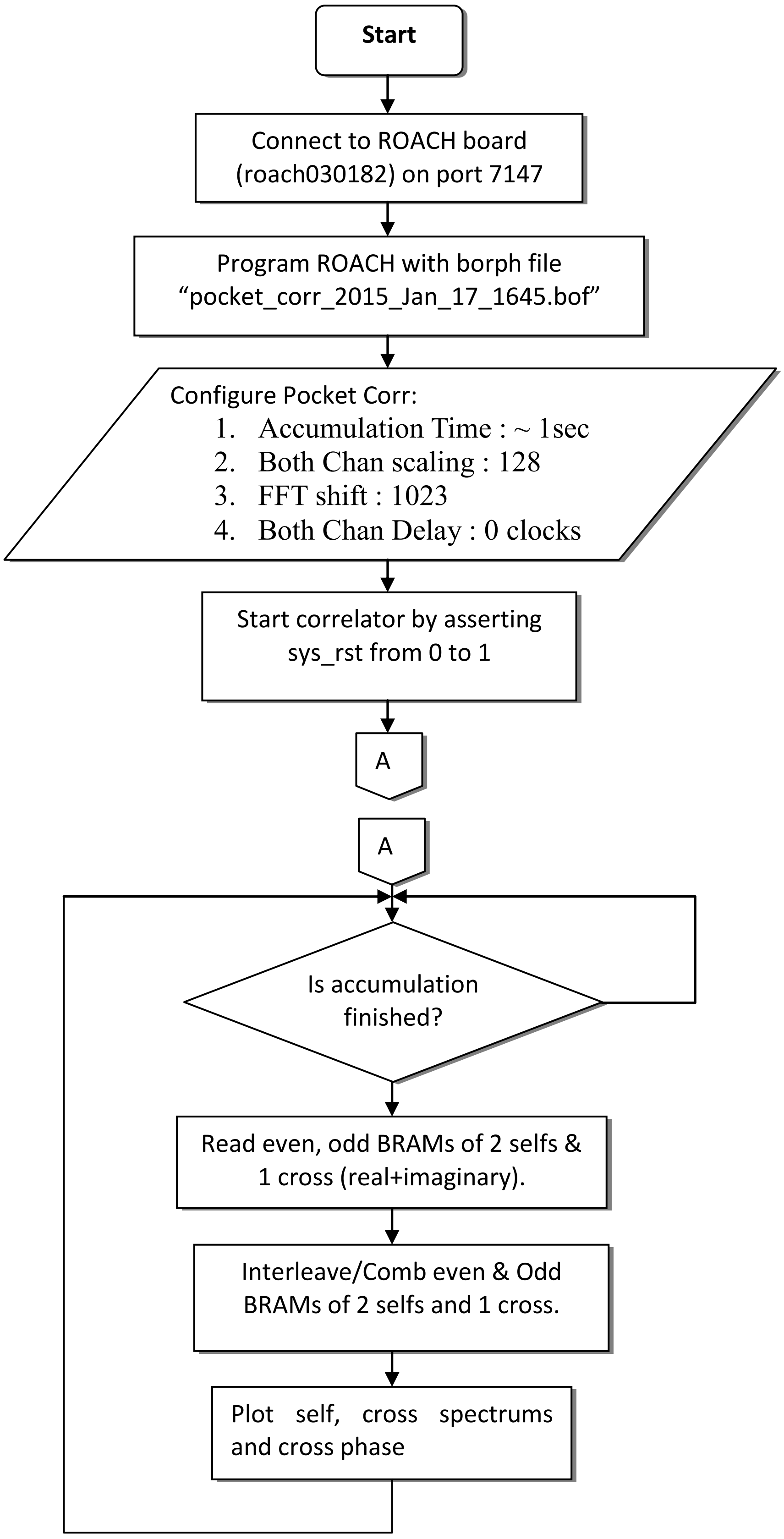


Figure 21: Basic Pocket Correlator

3.2.8 Hardware Testing:-

A program was written to program, configure correlator, then read final accumulation results and plot the correlation output using python programming. The flowchart is as follows:


Start
Connect to ROACH board (roach030182) on port 7147
Program ROACH with borph file "pocket_corr_2015_Jan_17_1645.bof"
Configure Pocket Corr:
1. Accumulation Time : ~ 1sec
2. Both Chan scaling : 128
3. FFT shift : 1023
4. Both Chan Delay : 0 clocks
Start correlator by asserting sys_rst from 0 to 1
A
A
Is accumulation finished?
Read even, odd BRAMs of 2 selfs & 1 cross (real+imaginary).
Interleave/Comb even & Odd BRAMs of 2 selfs and 1 cross.
Plot self, cross spectrums and cross phase

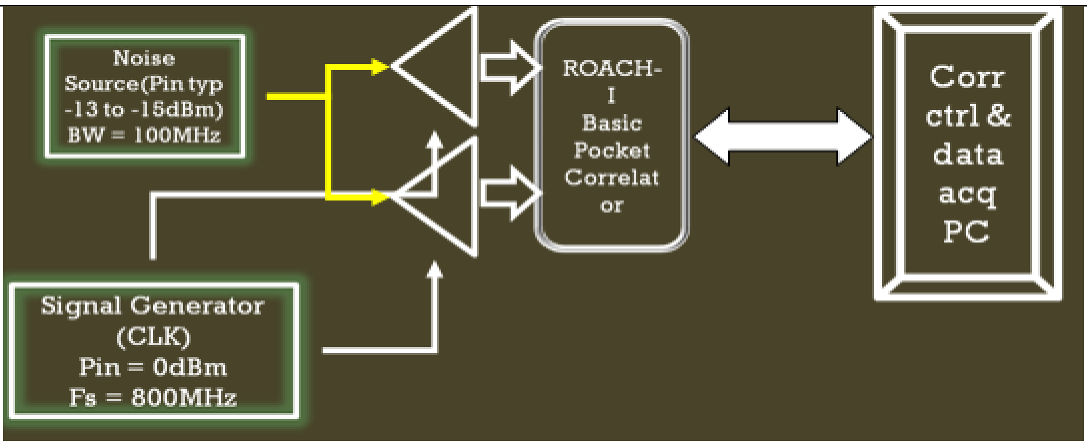
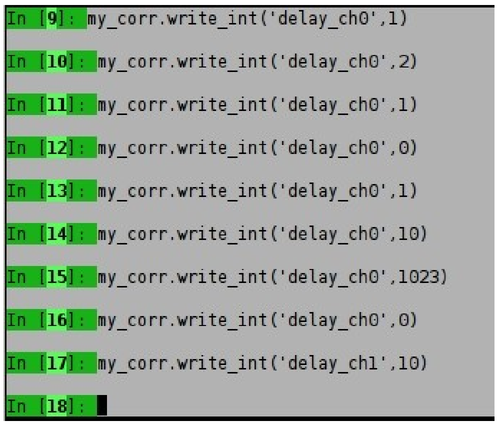

Noise Source(Pin typ -13 to -15dBm) BW = 100MHz
Signal Generator (CLK) Pin = 0dBm Fs = 800MHz
ROACH-I Basic Pocket Correlator
Corr ctrl & data acq PC

Figure 22: Basic Pocket Correlator Test Setup

As explained in the **"Integer Delay Block"** section with the help of this basic pocket correlator with data acquisition scrip we can verify the functionality of the 'delay_wideband_prog' block. The change in delay gets reflected in a cross correlation phase as seen in the plot. This integer delay block can be configured from a software register named 'delay_ch0' for CH-0 and 'delay_ch1' for CH-1.

- During initialization both channels were given zero delay, hence ideally cross phase should be flat but due to cables feeding to CH-1 & CH-2 length mismatch the phase will have some non-zero slope.
- Depending upon which channel you delay will get reflected in slope of cross correlation phase.
- CASE-I : If delay in CH-1 is more than CH-2 then phase will have ramp going in downward direction from 0th FFT channel.
- CASE-II : If delay in CH-2 is more than CH-1 then phase will have ramp going in upward direction from 0th FFT channel i.e. exactly opposite phase to that of fist case scenario.
- A delay of 1 clock means a ramp completing 360° phase cycle from 0th FFT channel to 1024th FFT channel.
- As the delay increases cross correlation amplitude drops compared to two self correlation amplitudes. When delay increases to FFT length then cross correlation nearly becomes very weak as seen in plot.



```
In [9]: my_corr.write_int('delay_ch0',1)

In [10]: my_corr.write_int('delay_ch0',2)

In [11]: my_corr.write_int('delay_ch0',1)

In [12]: my_corr.write_int('delay_ch0',0)

In [13]: my_corr.write_int('delay_ch0',1)

In [14]: my_corr.write_int('delay_ch0',10)

In [15]: my_corr.write_int('delay_ch0',1023)

In [16]: my_corr.write_int('delay_ch0',0)

In [17]: my_corr.write_int('delay_ch1',10)

In [18]: 
```

Figure 23: Delay setting through python console

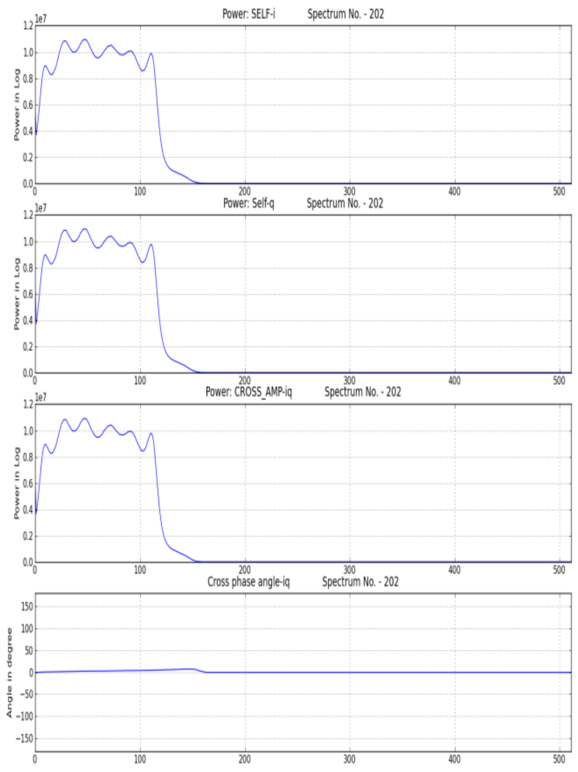


Figure 24: CH-1 delay of 1 clock cycle of ADC i.e. 800MHz

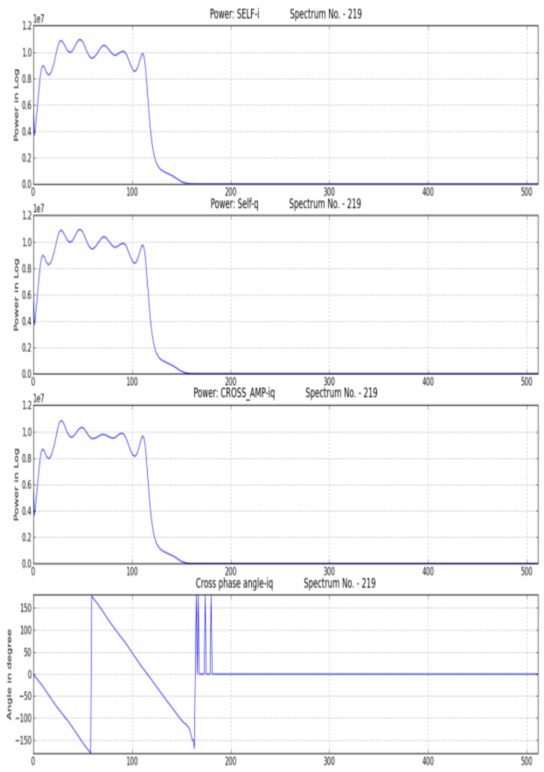


Figure 25: CH-1 delay of 10 clocks of ADC

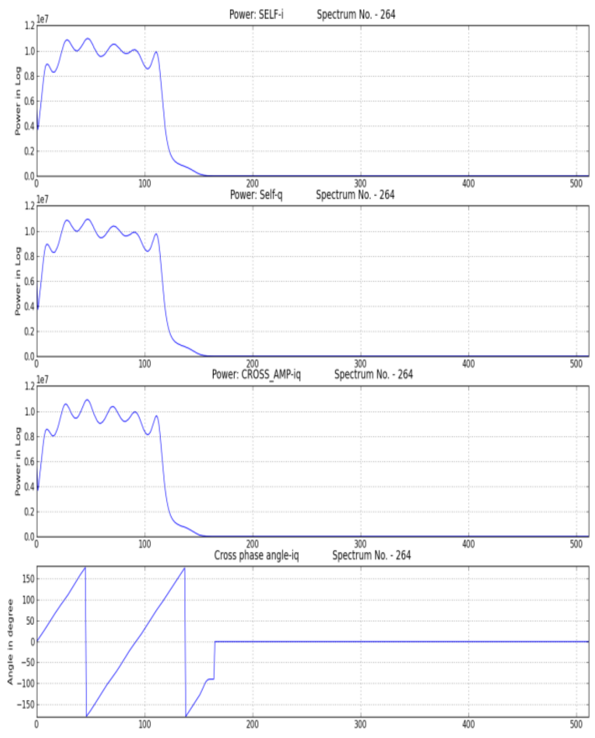


Figure 26: CH-2 delay of 10 clocks of ADC. Phase ramp is opposite to that of CH-1 delay of 10 clocks

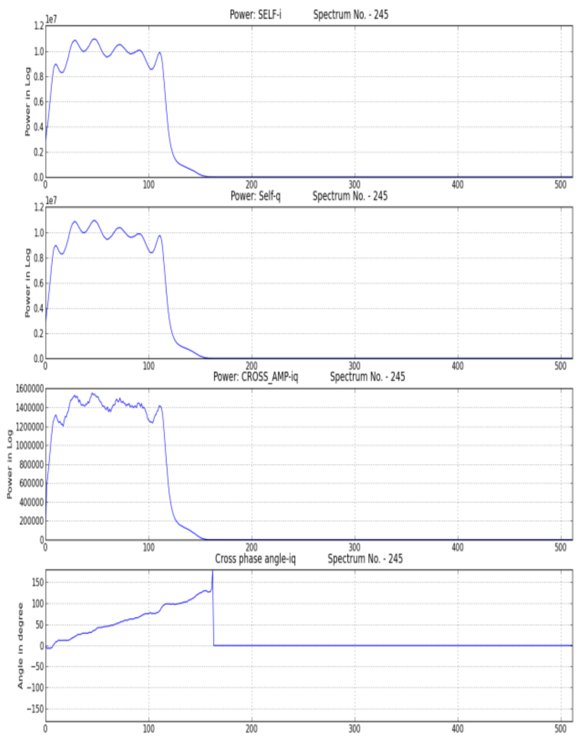


Figure 27: CH-1 delay of 1023 clocks of ADC. Drop in cross correlation amplitude

Chapter 4. Fringe Tester: Sync Gen, Trigger& Timing Circuit Design

4.1 Sync Gen:-

Most CASPER DSP blocks have a streaming architecture in which, after some initialization period, valid data is constantly being output. In addition, data tends to be defined in frames or windows, typically some spectrum or a signal of some known and repetitive vector length. To aid in managing the data stream, most major blocks in the library uses a sync pulse as timing fiducial, as well as a reset.

4.1.1 Sync Usage:-

1. Alignment:-

T

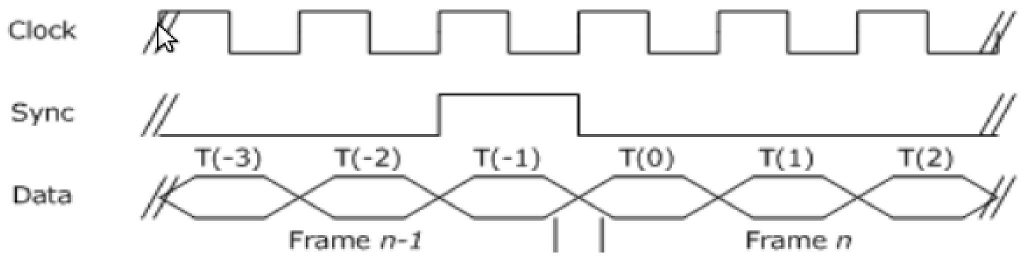


Figure 28: Sync pulse alignment to data

The sync pulse is used as a "vector warning signal," meaning that it precedes a frame of data 1 clock cycle prior to the first data sample of the frame. This allows the sync pulse to be used as an in-situ reset signal,counters and other state elements can be reset to their initialization conditions in line with the arrival of new data.

2. Propagation:-

Due to the reconfigurable, black-box nature of the CASPER DSP library, processing latency through a block varies depending on parameters, and can change with library revisions. Therefore, a latency-matched sync pulse is used to abstract data propagation delay in the data path. Blocks—particularly those with variable latency—are responsible for internally delaying a sync pulse input to match the data latency. This way, if a sync pulse is input to a block along with a frame of data, the results from that frame of data are aligned with the sync pulse output from that block, regardless of the block's internal datapath latency.

3. Timing:-

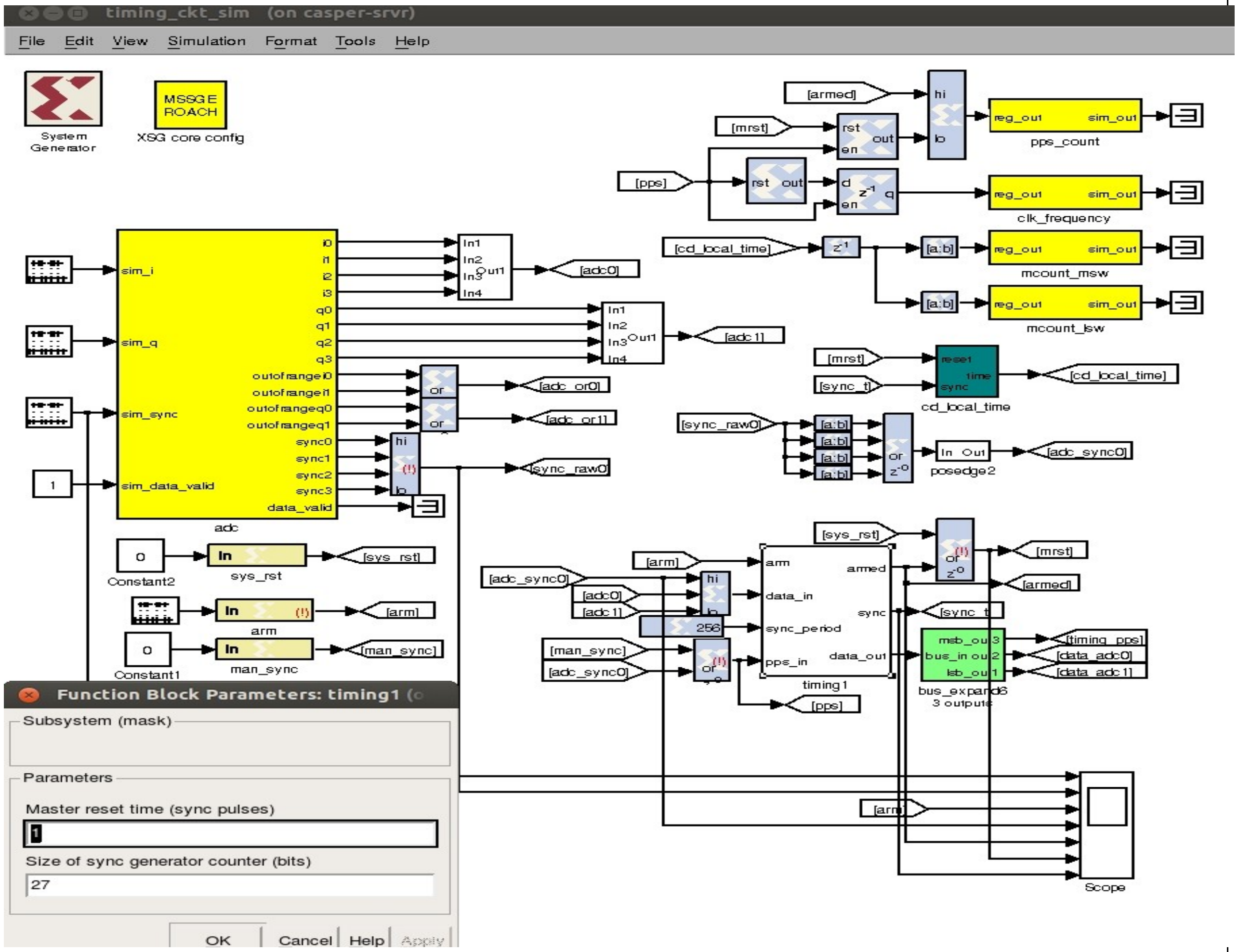
Sync pulses should be a periodic signal with a pulse width of 1 clock cycle. If a sync comes out of alignment, it will act as a reset for the data path. With the proper periodicity, the reset caused by the sync pulse should be coincident with the rollover to the initial conditions of a block's internal state. Period requirements vary with each particular design, depending on which processing elements are used. As channelization is the most common operation, its sync pulse timing is detailed in the memo (https://casper.berkeley.edu/wiki/Memos).

4.2 Trigger and Timing Circuit:

The basic pocket correlator/fringe tester at GMRT was missing the functionality of timing circuitry that can trigger the fringe tester at precise PPS (Pulse Per Second) Boundary and internally keep the record of local time which is required for loading of antenna delay values at exact precise time for which they are calculated on control PC.

The control PC is locked to NTP server at GMRT : a samay PC. The samay PC is locked using a GPS receiver by which very precise timing information is available to existing GMRT digital backend.

4.2.1 Circuit Diagram:-



Function Block Parameters: timing1

Subsystem (mask)

Parameters

Master reset time (sync pulses): 1

Size of sync generator counter (bits): 27

Figure 29: Sync and Timing Generator Circuit Design and Simulation

The sync and timing generator circuit is disciplined by manual arming and external 1 PPS signal. The design consists of ‘timing1’ block, ‘cd_local_time’ block and for monitoring local time through PPC, software registers – ‘clk_frequency’, ‘mcount_msw’, ‘mcount_lsw’

1. ‘timing1’ block description :- This block takes sync period as input and for simulation purpose we kept sync_period = 256, ‘pps_in’ port takes input from either 1PPS that is connected to ADC card externally or ‘man_sync’ that comes from software register. On arming, this block master reset the complete system on 2nd PPS and then from that point the system gets restarted. The

internal of this block consists of ‘reset_gen’ block and ‘sync_gen’ block.

a. ‘reset_gen’ block :- This block consists of a down counter equivalent to PPS signals which should skip before arming the system. In this design we skip 1 PPS pulse and triggers the system on 2nd PPS positive edge.

b. ‘sync_gen’ block : This block generates a vector warning signal i.e. sync signal which is aligned to the PPS signal edge at the time of arming only and then onwards gets repeated after a regular interval. Here in this case the sync interval is of 2^{26} FPGA clocks.

2. ‘cd_local_time’ block description:- The block consists of a 64-bit counter that gets reset on arming of ‘timing1’ block i.e. exactly on the edge of 2nd PPS from arming and runs at a rate of FPGA clock speed. Here in our case it is 200MHz (800MHz/4).
3. ‘mcount_lsw’ & ‘mcount_msw’ description :- These two software registers each of 32-bit, stores values of 64-bit ‘cd_local_time’ block counter value. Hence it shows the local time in FPGA clocks elapsed after arming the system.

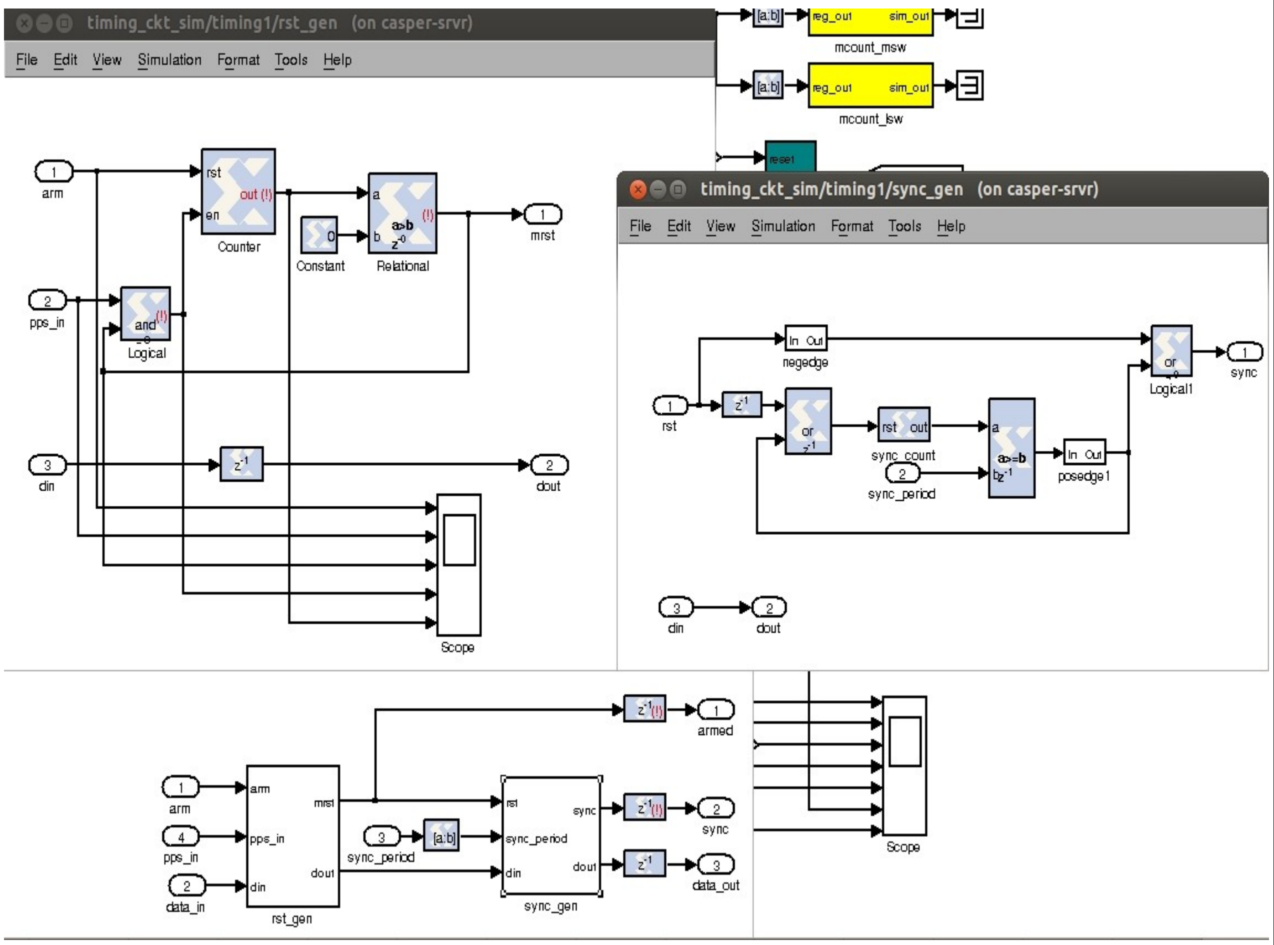
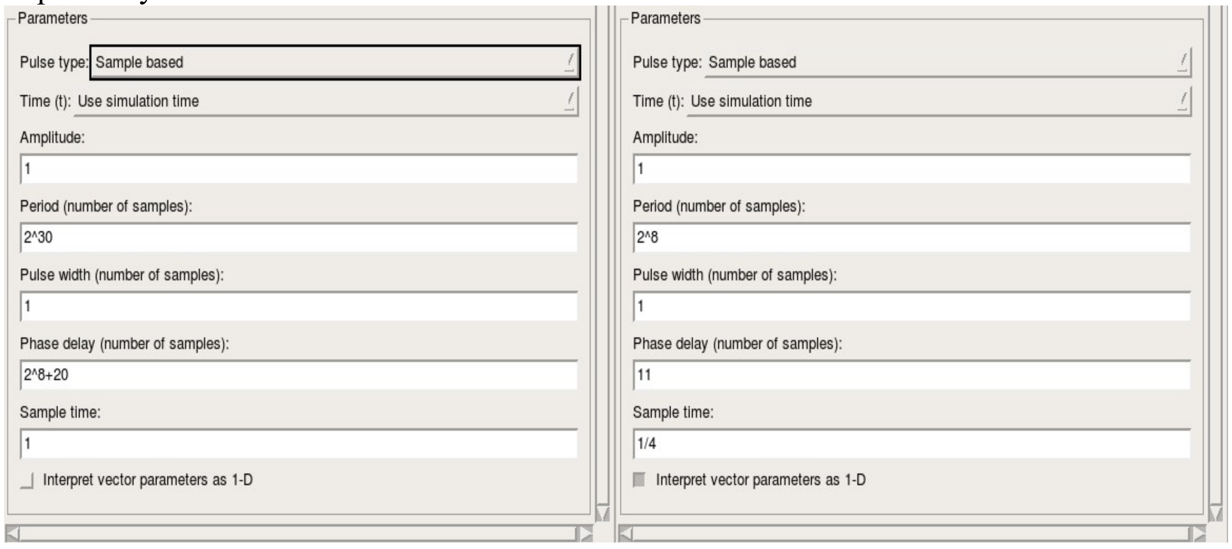


Figure 30: sync and Timing Circuit Internal Design Details

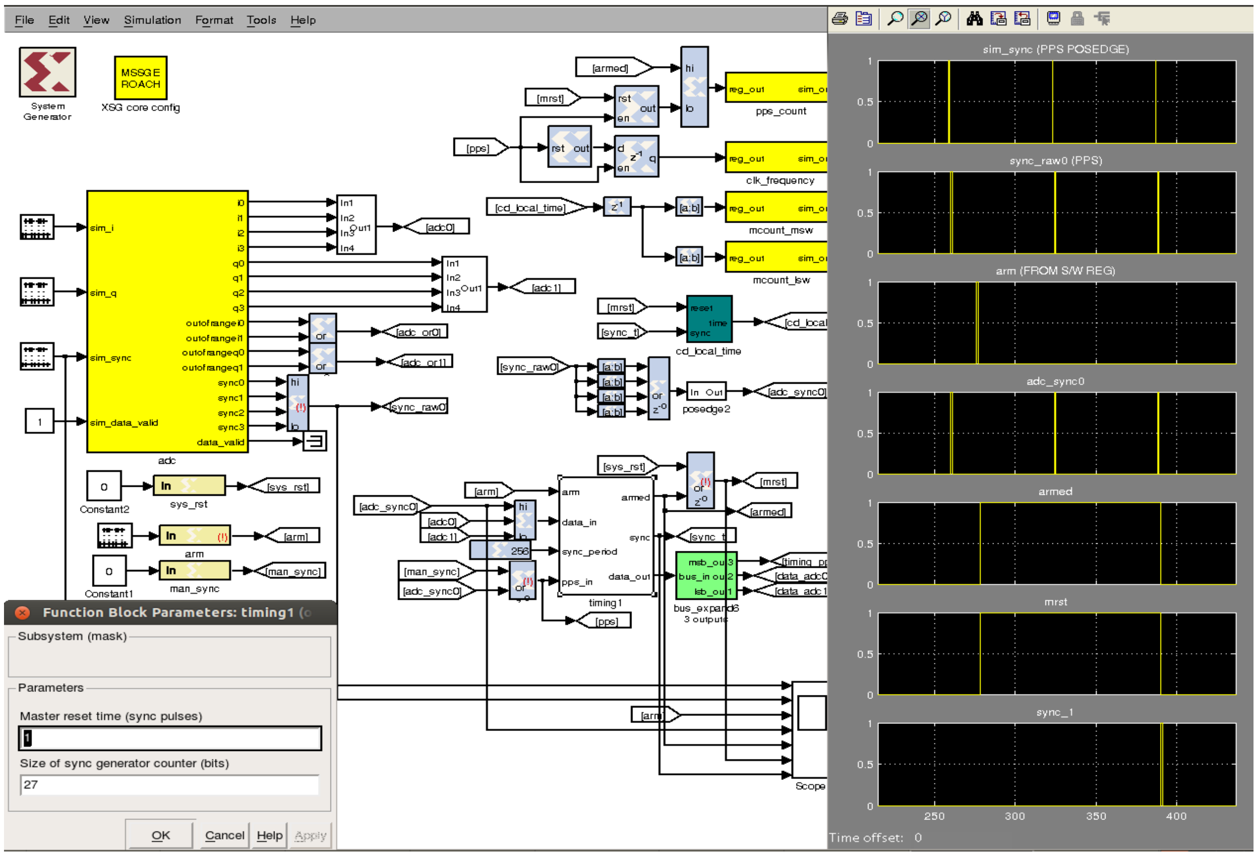
4.2.2 Simulation:-

The design was simulated with input sync_period = 256 FPGA clock (In practice it is 2^{26}).sim_sync input is a sample based pulse with period of 2^8 and sampling time of ¼ of simulation time.‘sys_rst,’man_sync’ and ‘arm’ signal was taken from software registers in practice but for simulation purpose only ‘arm’ signal was a pulse which will occur once at the start with a period of 2^{30} (Which will be out of simulation time).Below are the parameters for ‘arm’ and ‘sim_sync’ i.e. PPS from left to right in figure respectively.



	Parameters (left)	Parameters (right)
Pulse type:	Sample based	Sample based
Time (t):	Use simulation time	Use simulation time
Amplitude:	1	1
Period (number of samples):	2^30	2^8
Pulse width (number of samples):	1	1
Phase delay (number of samples):	2^8+20	11
Sample time:	1	1/4
Interpret vector parameters as 1-D	☐	☐

Figure 31: Simulated 'arm' pulse PPS pulse parameters to time & sync system resp.



Function Block Parameters: timing1

Subsystem (mask)

Parameters

Master reset time (sync pulses): 1

Size of sync generator counter (bits): 27

OK Cancel Help Apply

Figure 32: Complete time and sync system with simulation results.

In the above simulation the 1st signal is a PPS signal's positive edge which is an input to 'timing1' block. The output from yellow 'adc' block is sync_raw0 is a parallel sampled version of PPS signal. 'arm' signal is a user defined signal that comes from python program. 'adc_sync0' is a positive edge of sampled PPS signal.

The output signal 'armed' along with 'sys_rst' makes a 'mrst' signal and which in turn will be used to load delay values using 'delay_gen' block. The 'mrst' signal holds the subsequent system in reset until 2nd PPS arrives after arming. The sync then starts from 2nd PPS after arming with a period of 2^{26} FPGA clocks.

4.3 Fringe Tester with Timing Circuit:

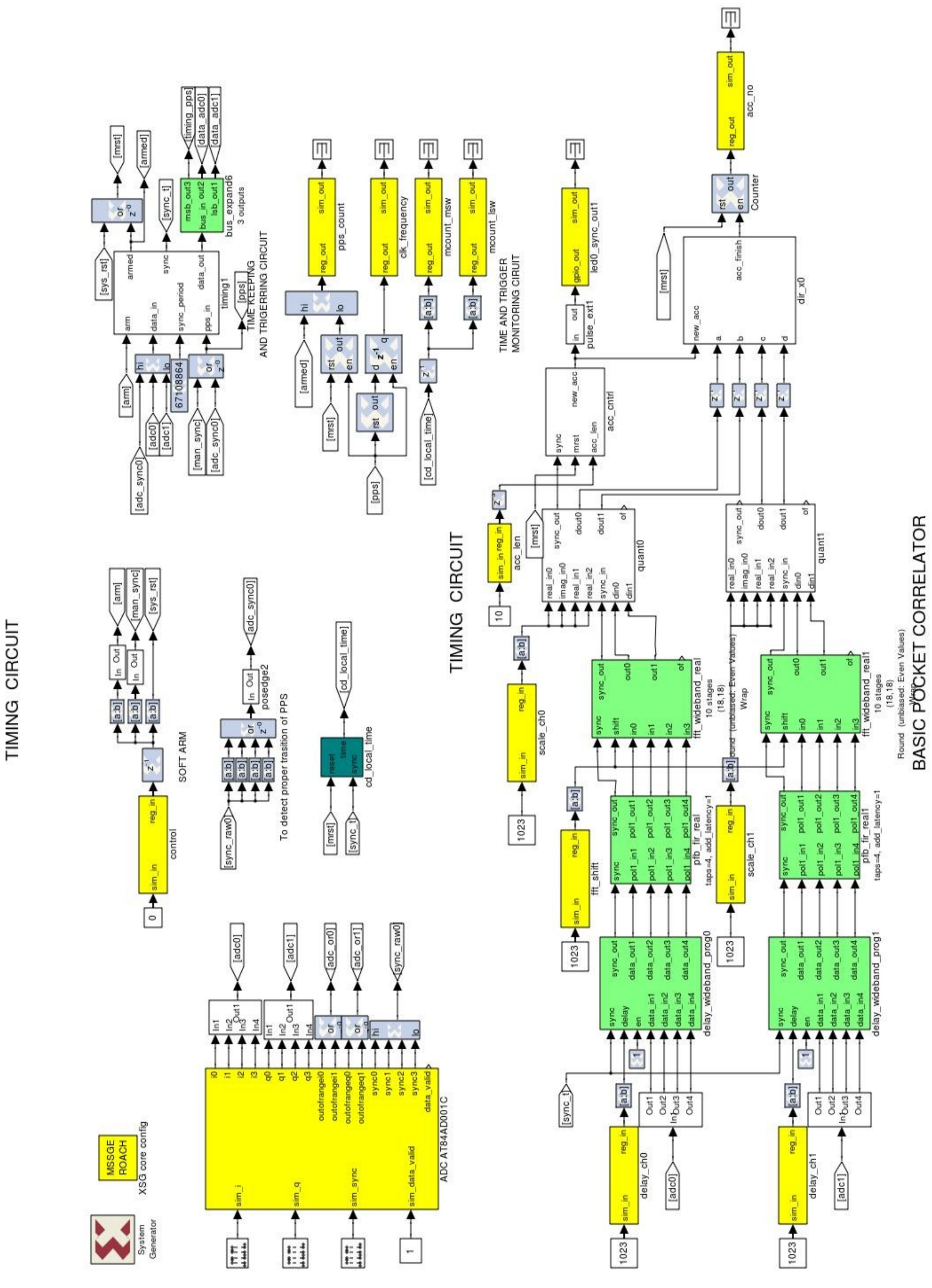


Figure 33: Fringe Tester with Trigger & Timing Circuit

Chapter 5. Fringe Tester: Software Design

5.1 Development of python package:

To configure fringe tester functionality by hiding lower level details from user, a python package needs to be developed. This python package can program/deprogram, initialize-trigger, configure. This python package design is divided into small pieces.

ROACH board PowerPC communicate with control PC over 1Gbps link using katcp (Karaoo Array Telescope Communication Protocol) – a custom protocol developed by CASPER community and is now available in python as a 'katcp_wrapper.py' program.

Using this program we can control fringe tester functionality at lower level. To hide details at lower level from user we have built fringe tester python module.

Python Package Functional Details:-

5.1.1 CONFIG FILE:-

Config file contains section wise configurable parameters details of correlator. This is required to have all configurable parameters of correlator to be at one place i.e. a config file.

- SECTION : katcp
  1. katcp_port : Port to communicate with FPGA design through ROACH PowerPC.
  2. servers : ROACH board IP/ name to connect.
  3. bitstream: Borph file to program ROACH board.

- SECTION: correlator
  1. n_chans : Number of frequency channels. (512)
  2. n_ants : Number of antennas in design. (2)
  3. fft_shift : fft shift to be applied to fft block. (1023)
  4. acc_len : Accumulation length in terms of fft time that decide accumulation time as fft_time= (n_chans/2) / FPGA_Clock (256/200MHz = 1.28µS)
     acc_len = accumulation time/fft_time (1second/1.28µS = 781250)
  5. adc_clk : External ADCclock fed to the system (800MHz)
  6. sync_time: The time since epoch when system triggered. (Needs to be very accurate)
  7. fft_len: Size of FFT inside design.
  8. max_int_dly : Maximum integer delay which can be compensated in design (2^{18} = 262144). For GMRT the delay requirement for farthest antenna w.r.t. C02 is +/-128µS and max_int_dly (2^{18} x 800MHz = 327µS) satisfies it.
  9. mcnt_bits: FPGA time keeping counter bits. (48-bits)

- SECTION: equalization
  1. scale0: Scaling factor for CHAN-0
  2. scale1: Scaling factor for CHAN-1

- SECTION: correlator
  1. ch0: Delay to be set for CHAN-0
  2. ch1: Delay to be set for CHAN-1

Config File Generation Script Details: -

Required Python Modules: ConfigObj from configobj.

1) Enter the ‘config file name’.
2) Create ‘katcp’ section/dictionary in the config file.
3) In that katcpsection,create variable katcp_port=7147,server=’roach030182’ and bitstream=’bof file name’.
4) Create another section/dictionary.
5) In that create following variables: n_chans=512, n_ants=2, fft_shift=0x3ff, acc_len=781250, adc_clk=800MHz, sync_time=2^{27} fft_len=2^{19} max_int_delay=2^{18},mcnt_bits=48.
6) Create section equalization.
7) In that add variables scale0=128,scale1=128.
8) Create delay section.
9) In that add variables ch0=0, ch1=0.
10) Call write function & write the data in ‘configfilename.cfg’.

Configuration File Reading Script Details :-

This script contains a class named ‘Corrcof’ that consists of all functions required to read configuration file of correlator and return the same to calling function.

Required Python Modules: iniparse,exceptions,struct,numpy

- __init__ : (Initialization function for Class Corrconf to do initial settings)
  1) Check configuration file existence
  2) Parse the configuration file using iniparse module.
  3) Create an empty dictionary ‘config’ to store section wise parameter values.
  4) Read all config file items values using function read_all() within the class ‘Corrconf

- __getitem__ (Return populated dictionary ‘config’ to calling function)
  1) Return populated dictionary ‘config’ to calling function

- __setitem__ (Set the given item in dictionary ‘config’ by the calling function)

1) Set the given item in dictionary ‘config’ by the calling function

- Function file_exists():
  1) Try to open the configuration file.
  2) Ifexists close the file and return exists flag = True
  3) Else not exists&return an IO error.

- Function: read_all() : (Reads all sections parameters within a configuration file and populate dictionary ‘config’
  1) Read ROACH server.
  2) Read bitstream to program server.
  3) Read katcp section: katcp port.
  4) Read correlator section:
     a. n_chans
     b. n_ants
     c. fft_shift
     d. acc_len
     e. adc_clk
     f. sync_time
     g. fft_len
     h. fft_per_sync
     i. max_int_dly
     j. mcnt_bits.
  5) Read equalization section:
     a. scale0
     b. scale1
  6) Derive feng_clk from adc_clk.
  7) Derive mcnt_scale_factor from derived feng_clk.

- Function: read_int()
  1) Read integer parameters of a variable inside a given sections.

- Function: read_str()
  1) Read string parameters of a variable inside a given sections.

- Function:write()
  1) Write given parameter into dictionary ‘config’
  2) Write given parameter especially sync_time into configuration file.

5.1.2 Fringe Tester Python Function Details:

This script contains a class named 'Correlator' that consists of all functions required to communicate, initialize and configure correlator functionality. This reads configuration file for setting correlator functionality.

➢ **CLASS CORRELATOR:-**

Required Python Modules: iniparse, exceptions, struct, numpy.

- __init__ : (Initialization function for Class Correlator to do initial settings)
  1) Print 'initialization'.
  2) Search for the config file contents.
  3) If content not found then read the default config file.
  4) Read the config file.

- Function: Check_katcp_connection()
  1) Set it initially TRUE.
  2) Try to ping the desired ROACH(roach030182).
  3) If pinged then print 'Katcp connection is OK'.
  4) Otherwise print'Katcpconnection FAIL'.
  5) Make status false.
  6) Return the status.

- Function: connect()
  8) Initialize my_corr class
  9) Connect the FPGA to desired roach using katcp protocol.
  10) Check the katcp connection.

- Function: disconnect()
  1) Stop the katcp connection.

- Function: deprog()
  1) Deprogram the FPGA.
  2) Print 'FPGA Deprogrammed'.

- Function: prog()
  1) Program the FPGA with desired bof file.
  2) Print'FPGA programmed'.

- Function: fpga_uptime()
  1) Read the software register pps_count.
  2) Extract the pps_cnt bit from pps count register.
  3) Extract arm_stat bit from pps_count register.
  4) Return pps_cnt&arm_stat.

- Function: arm()
  1) Wait till the mid of the 1stpps signal.
  2) Note the trig_time as it is 1 pps ahead of the current time.
  3) Write in control register ‘0’ then ‘4’ and then ‘0’.
  4) Print the ctime (dd/mm/yy)of the trig_time.
  5) Print the ctime of current time.
  6) Set arm TRUE.
  7) Set pps_cnt zero.
  8) Print arm_stat&pps_cnt.
  9) Wait till (trig_time-current time+0.3).
  10) Set arm TRUE.
  11) Set pps_cnt zero.
  12) Write sync_time in config file.
  13) Print arm_stat.
  14) Print pps_cnt.
  15) Print ctime of trig_time.
  16) Print ctime of current time.

5.2 Fringe Tester Configuration and Data Acquisition:

5.2.1 Correlator Initialization Script:-

1) Function poco_init(): (Initialize correlator and configure it according to configuration file parameters)
   a. Instantiate class Correlator as ‘corr1’.
   b. Connect to ROACH board specified in configuration file.
   c. Deprogram ROACH board.
   d. Program ROACH board with bitstream specified in configuration file.
   e. Read acc_len, scale0, scale1, fft_shift from configuration file
   f. Set register ‘acc_len’ in pocket correlator design with config file acc_len value.
   g. Set register ‘scale0’ in pocket correlator design with config file scale0 value.
   h. Set register ‘fft_shift’ with config file fft_shift value
   i. Set delay of channel-0 ‘delay_ch0’ to 0.
   j. Set delay of channel-1 ‘delay_ch1’ to 0.
   k. Arm the design i.e. trigger the design for accurate timing.
   l. Check uptime of design after arming.
   m. Return triggering time (in units of seconds since from epoch).
2) Disconnect the katcp link.
3) Exit

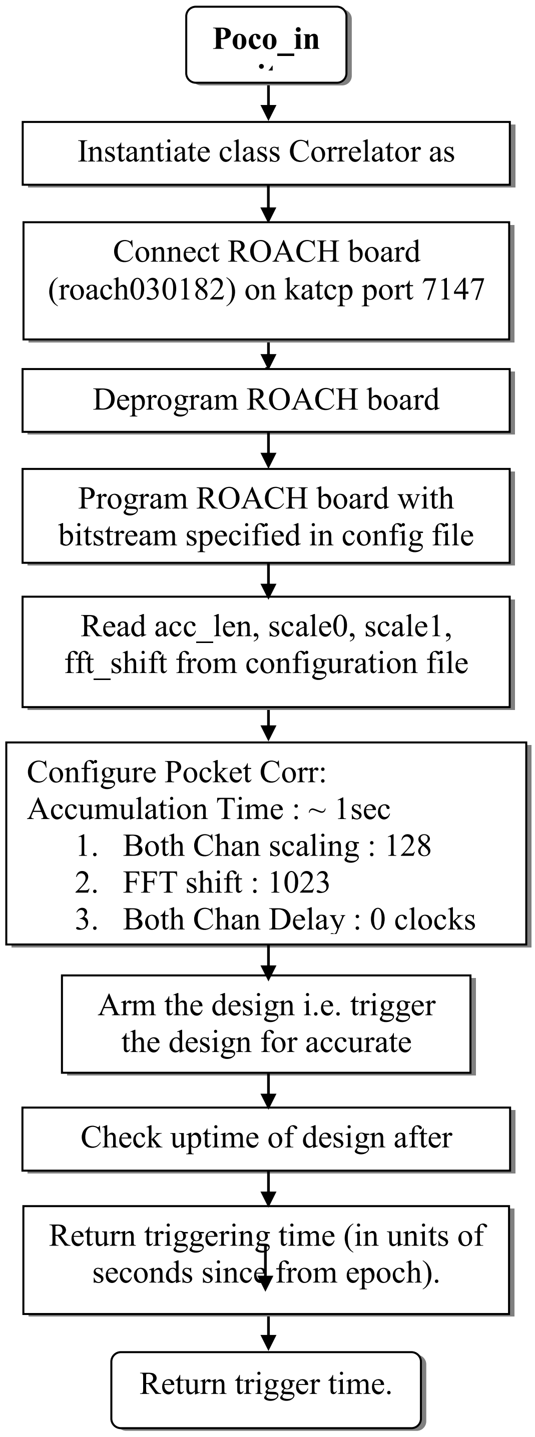
5.2.2 Correlator Data Plotting Script:-

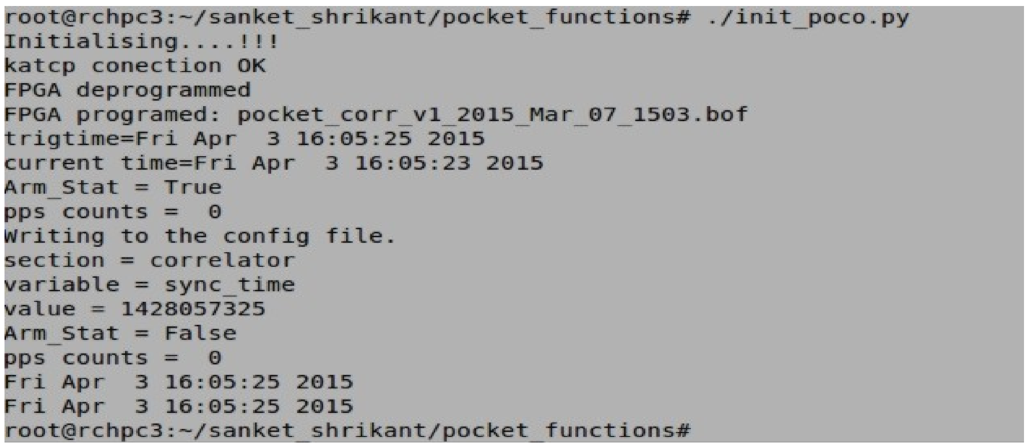
(This script expect you have configured correlator first with correlator initialization script)

1) Get command line arguments viz. multiplot/phase_plot/cross_plot and/or tax_dump options.
2) Initialize correlator instance.
3) Read configuration file (Here it is example.cfg).
4) Read accumulation number which reflects finished accumulation.
5) Continue reading accumulation number till next subsequent accumulation to be sure of readingnon-corrupted data.
6) Create a pylab figure object for plotting purpose.
7) Call function Updatefig()
8) Function Interleave () : (This functions takes any number of lists as arguments and interleave/comb their contents.)
    a. Iterate over the maximum contents inside lists and over total number of lists.
    b. Put the results in a local variable x.
    c. Return the interleaved/combed result x to calling function.
9) Function Updatefig():
    a. Read accumulation count as pre_read count.
    b. Read accumulator even and odd BRAMs viz. self CH-1(even & odd), self CH-2 (even & odd), Cross real (even & odd), Cross imag (even & odd).
    c. Read accumulation count as post_read count.
    d. If pre_read and post_read counts are same go to next step else go to (c).
    e. Create complex array as Cross = Cross_real + j*Cross_imag
    f. Interleave even and odd self CH-1 values.
    g. Interleave even and odd self CH-2 values.
    h. Interleave even and odd Cross values.
    i. If command line multiplot option is enabled plot 3 subplots as :
        i. Plot Selfs in subplot-1.
        ii. Plot Cross amplitude in subplot-2.
        iii. Plot Cross phase in subplot-3.
    j. If command line cross_plot option is enabled plot 2 subplots as :
        i. Plot Cross amplitude in subplot-1.
        ii. Plot Cross phase in subplot-2.
    k. If command line phase_plot option is enabled plot Cross phase.

5.3 Flow-Charts:-

5.3.1 Fringe Tester Initialization Flow Chart:-

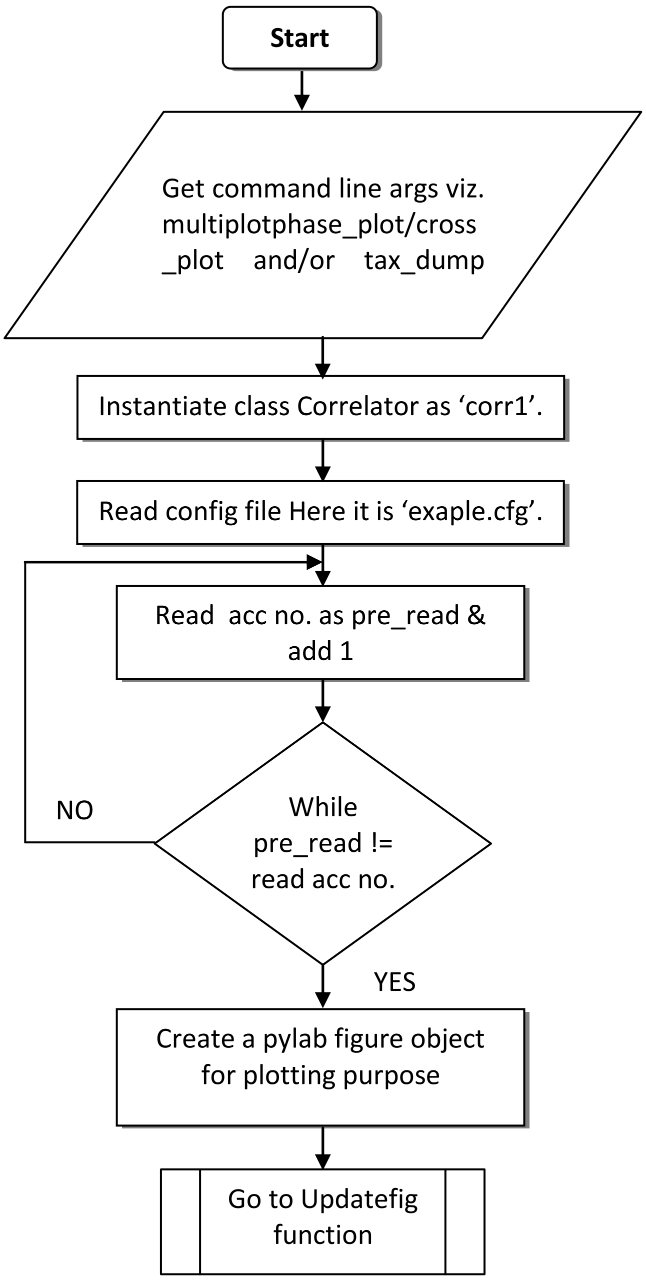

Poco_in
Instantiate class Correlator as
Connect ROACH board (roach030182) on katcp port 7147
Deprogram ROACH board
Program ROACH board with bitstream specified in config file
Read acc_len, scale0, scale1, fft_shift from configuration file
Configure Pocket Corr:
Accumulation Time : ~ 1sec
1. Both Chan scaling : 128
2. FFT shift : 1023
3. Both Chan Delay : 0 clocks
Arm the design i.e. trigger the design for accurate
Check uptime of design after
Return triggering time (in units of seconds since from epoch).
Return trigger time.

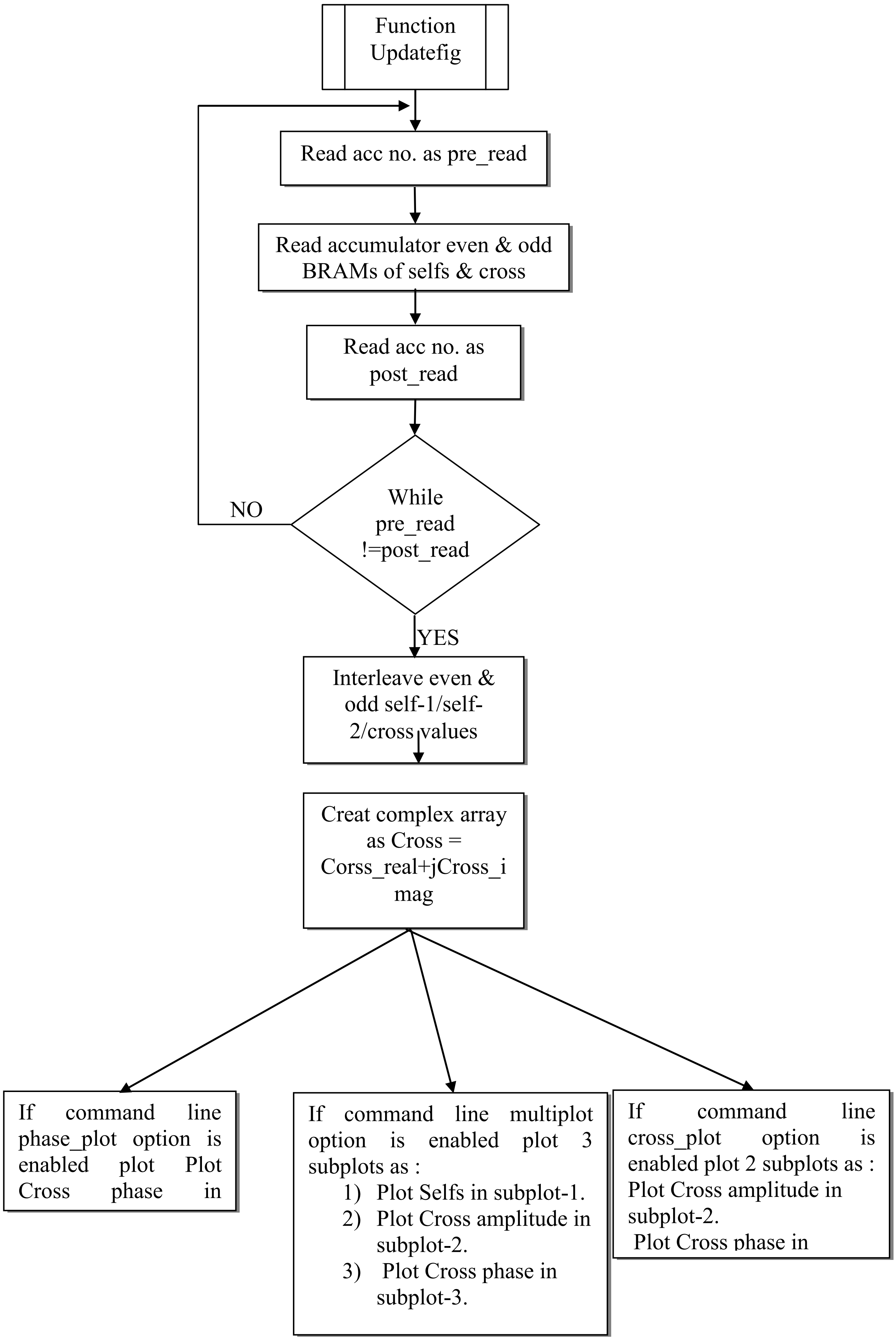


```
root@rchpc3:~/sanket_shrikant/pocket_functions# ./init_poco.py
Initialising....!!!
katcp conection OK
FPGA deprogrammed
FPGA programed: pocket_corr_v1_2015_Mar_07_1503.bof
trigtime=Fri Apr  3 16:05:25 2015
current time=Fri Apr  3 16:05:23 2015
Arm_Stat = True
pps counts =  0
Writing to the config file.
section = correlator
variable = sync_time
value = 1428057325
Arm_Stat = False
pps counts =  0
Fri Apr  3 16:05:25 2015
Fri Apr  3 16:05:25 2015
root@rchpc3:~/sanket_shrikant/pocket_functions#
```

Figure 34: Initialization script Testing

5.3.2 Fringe Tester output plotting:-


Start
Get command line args viz. multiplotphase_plot/cross_plot and/or tax_dump
Instantiate class Correlator as ‘corr1’.
Read config file Here it is ‘exaple.cfg’.
Read acc no. as pre_read & add 1
While pre_read != read acc no.
NO
YES
Create a pylab figure object for plotting purpose
Go to Updatefig function


Function Updatefig
Read acc no. as pre_read
Read accumulator even & odd BRAMs of selfs & cross
Read acc no. as post_read
While pre_read !=post_read
NO
YES
Interleave even & odd self-1/self-2/cross values
Creat complex array as Cross = Corss_real+jCross_imag
If command line phase_plot option is enabled plot Plot Cross phase in
If command line multiplot option is enabled plot 3 subplots as :
1) Plot Selfs in subplot-1.
2) Plot Cross amplitude in subplot-2.
3) Plot Cross phase in subplot-3.
If command line cross_plot option is enabled plot 2 subplots as : Plot Cross amplitude in subplot-2.
Plot Cross phase in

Chapter 6. Antenna Results and Applications

6.1 Antenna Results:-

We have expected that our design should trigger at very precise time using a designed triggering and time keeping circuit and python arm module. With the help of onboard timing circuitwe can keep precise time on FPGA board which is required for applying integer and fractional delay on that precise time.

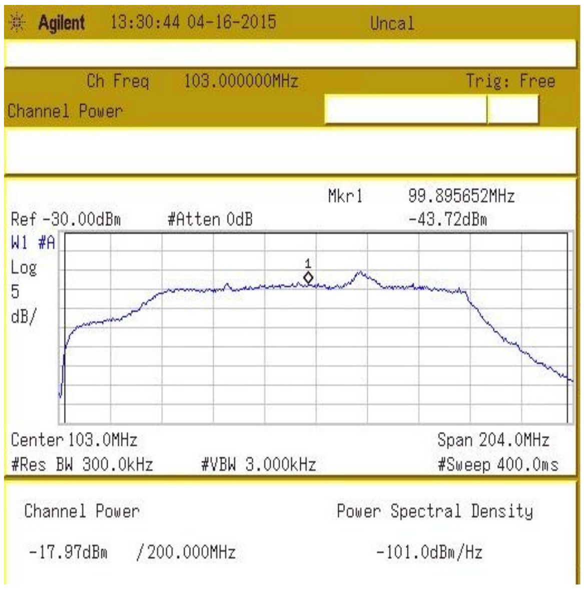
The design was tested with integer delay functionalityand as the delay increases the effect can be seen on cross phase and if delay is larger than FFT length then can be seen on cross amplitude.

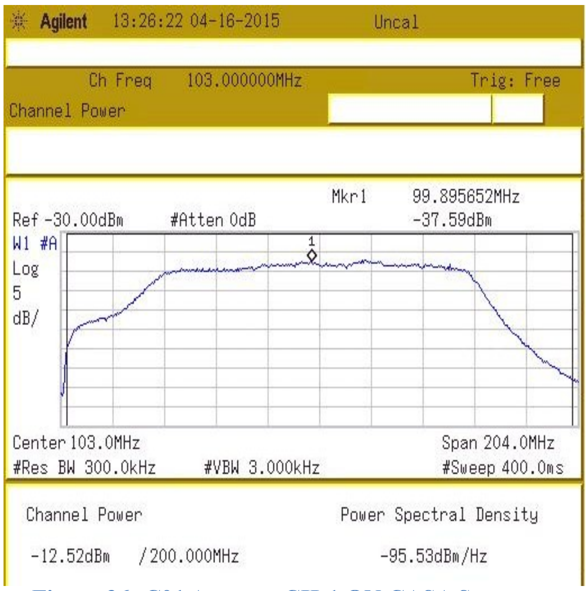
The product will test the two antennas signal and debug the problem in between the signals coming from these antennas. This equipment is handy and can debug the problem coming at the receiver chain. It should work properly giving us the fringes at the output. A fringe indicates that the cross-correlation is happening between the two signals which we are testing.If cross correlation is observed then there is no problem in receiver chain. If we don’t get cross-correlation then we will observe the auto-correlation. The antenna whose autocorrelation is zero that antenna has problem in receiving the signal and need the investigation.

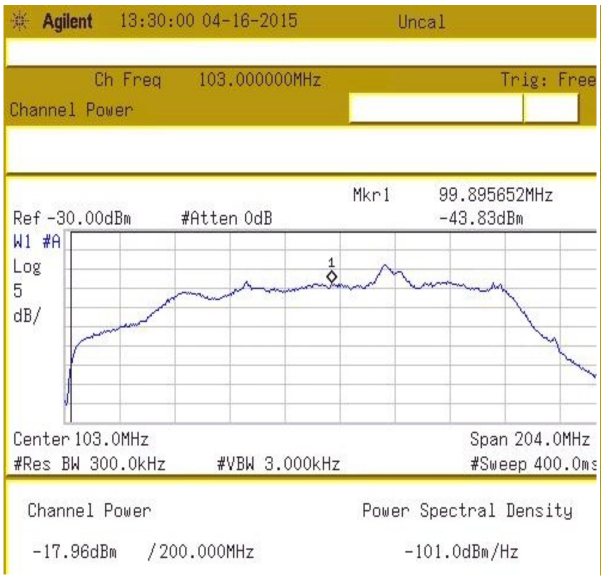
6.1.1 Antenna Test Setup:

Antennae used	: C01,C08
RF Frequency Band	: 1280MHz (RF Bandwidth : 120MHz)
GMRT Anlaog Backend LO (GAB LO)	: 1390MHz
GAB Filter	: 200MHz
Deflection Taken on	: CASA source (Expected ~ 8dB)(To check Front-End system of GMRT)
Cross Correlation seen on	: 3C147 source (Point source)

6.1.2 Deflection Test Results:

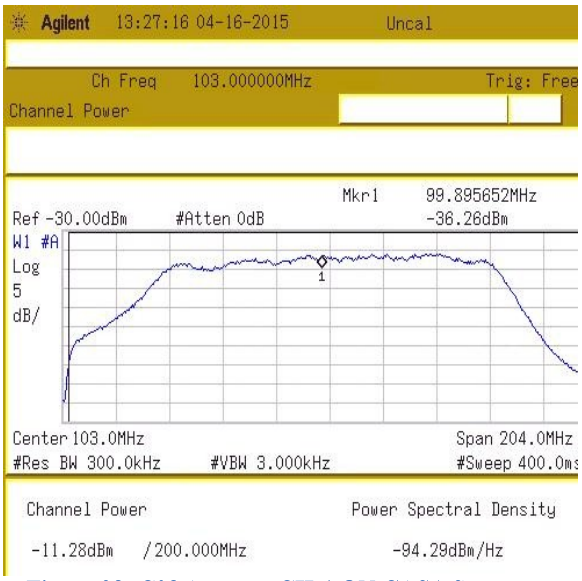

```
Agilent  13:30:44 04-16-2015          Uncal
    Ch Freq    103.000000MHz               Trig: Free
Channel Power
                                 Mkr1    99.895652MHz
Ref -30.00dBm     #Atten 0dB             -43.72dBm
W1 #A
Log
5
dB/
Center 103.0MHz                          Span 204.0MHz
#Res BW 300.0kHz     #VBW 3.000kHz       #Sweep 400.0ms
  Channel Power                 Power Spectral Density
  -17.97dBm   /200.000MHz         -101.0dBm/Hz
```
Figure 35: C01 Antenna CH-1 OFF CASA Source


```
Agilent  13:26:22 04-16-2015          Uncal
    Ch Freq    103.000000MHz               Trig: Free
Channel Power
                                 Mkr1    99.895652MHz
Ref -30.00dBm     #Atten 0dB             -37.59dBm
W1 #A
Log
5
dB/
Center 103.0MHz                          Span 204.0MHz
#Res BW 300.0kHz     #VBW 3.000kHz       #Sweep 400.0ms
  Channel Power                 Power Spectral Density
  -12.52dBm   /200.000MHz         -95.53dBm/Hz
```
Figure 36: C01 Antenna CH-1 ON CASA Source



```
Agilent   13:30:00 04-16-2015          Uncal

        Ch Freq    103.000000MHz          Trig: Free
Channel Power

                           Mkr1      99.895652MHz
Ref -30.00dBm     #Atten 0dB           -43.83dBm
W1 #A
Log
5
dB/

Center 103.0MHz                       Span 204.0MHz
#Res BW 300.0kHz      #VBW 3.000kHz    #Sweep 400.0ms

 Channel Power                  Power Spectral Density

 -17.96dBm    /200.000MHz          -101.0dBm/Hz
```

Figure 37: C08 Antenna CH-1 OFF CASA Source



```
Agilent   13:27:16 04-16-2015          Uncal

        Ch Freq    103.000000MHz          Trig: Free
Channel Power

                           Mkr1      99.895652MHz
Ref -30.00dBm     #Atten 0dB           -36.26dBm
W1 #A
Log
5
dB/

Center 103.0MHz                       Span 204.0MHz
#Res BW 300.0kHz      #VBW 3.000kHz    #Sweep 400.0ms

 Channel Power                  Power Spectral Density

 -11.28dBm    /200.000MHz          -94.29dBm/Hz
```

Figure 38: C08 Antenna CH-1 ON CASA Source

6.1.3 Fringe Testertest setup with same polarization:

ROACH board fringe tester iADC board was connected as follows:

1.I –port connected to	: C01 (CH-1/Pol-1 of Antenna)
2. Q-port connected to	: C08 (CH-1/Pol-1 of Antenna)
3. Sync port connected to	: 1 PPS signal
4. Clk port connected to	: 800MHz 0dBm clock source

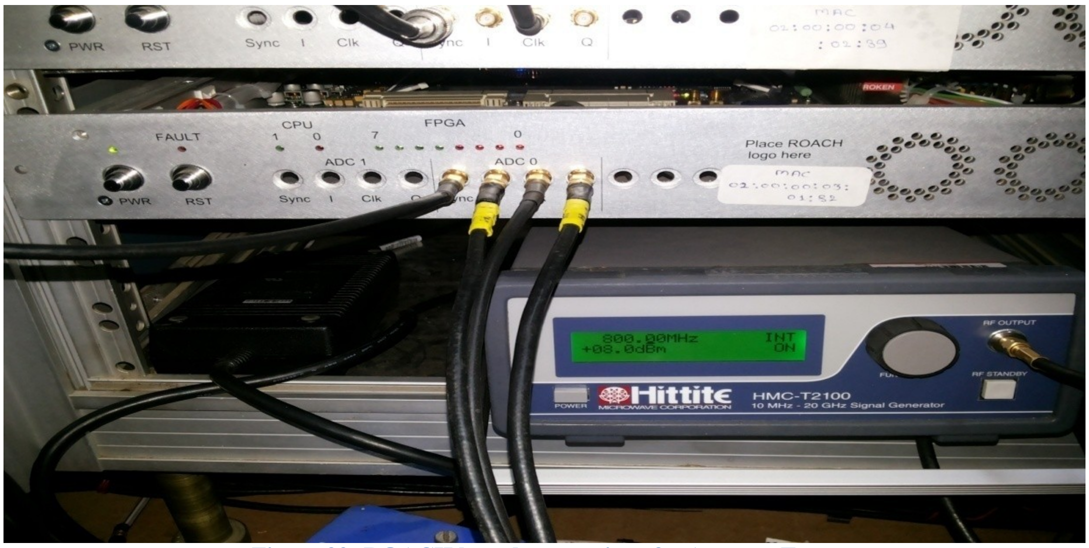
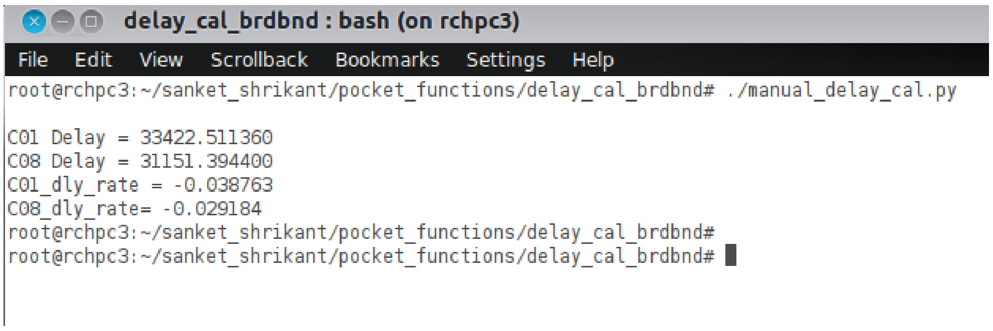


Figure 39: ROACH board connections for Antenna Tests

The instantaneous delay of each connected antenna i.e. C01 and C08 w.r.t. C02 was calculated using a GMRT delay_cal routine and manually loaded into delay register of Fringe Tester design.


delay_cal_brdbnd : bash (on rchpc3)
File Edit View Scrollback Bookmarks Settings Help

```
root@rchpc3:~/sanket_shrikant/pocket_functions/delay_cal_brdbnd# ./manual_delay_cal.py

C01 Delay = 33422.511360
C08 Delay = 31151.394400
C01_dly_rate = -0.038763
C08_dly_rate= -0.029184
root@rchpc3:~/sanket_shrikant/pocket_functions/delay_cal_brdbnd#
root@rchpc3:~/sanket_shrikant/pocket_functions/delay_cal_brdbnd#
```

Figure 40: Instantaneous delay calculation using delay_cal routine in terms of ADC clock

6.1.4 Fringe Tester Antenna Test Results:

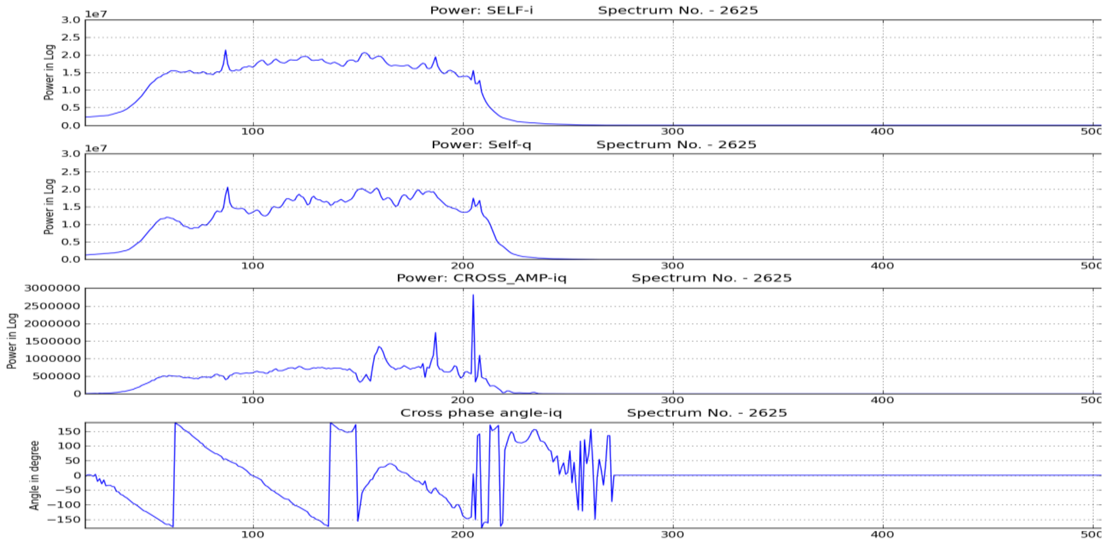


Figure 41: Fringe Tester output with same polarization. I=>CH-1,Q=>CH-1

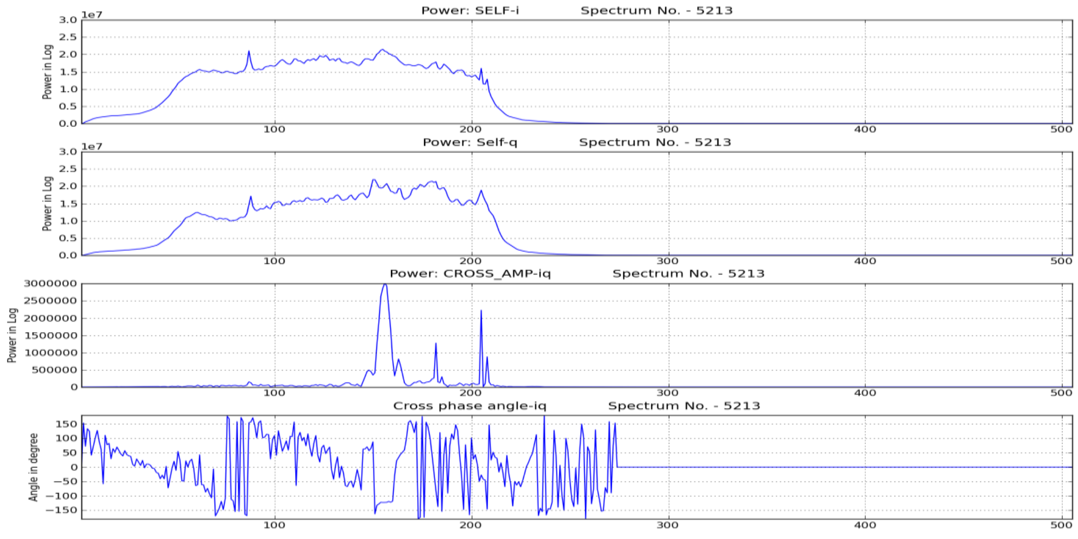


Figure 42: Fringe Tester output with different polarization. I=>CH-1,Q=>CH-2

6.2 Applications:-

1. The Fringe Tester design is fundamental broadband FPGA based correlator which can be expanded or developed for 30 antennas that can be used to synthesize 25 km diameter parabolic antenna using 30 GMRT antennas by a technique called interferometry.

2. The Fringe Tester system can be used as debugging tool to test new broadband GMRT Analog Backend (GAB), broadband front end system and hence broadband GMRT analog receiver chain.

3. The Fringe Tester design can be used to debug fringe related problems viz. low fringe/no fringe.

4. The precise triggering of this circuit ensures start of correlator at very precise time and keeps local time record on FPGA, thus is capable to apply delay values at very precise time and reduces timing related errors in delay and fringe correction.

5. The triggering and time keeping circuit along with python control package is a general purpose design that can be easily used in any future FPGA based design that requires precise triggering and timing information.

Chapter 7. Conclusion and Future Work

7.1 Conclusion

We have successfully designed, simulated and tested the handy fringe tester for 2 parabolic antennas. We also successfully designed a general purpose triggering and timing circuit for FPGA based design along with python control package.

We also written a python package to control Fringe Tester functionality and successfully tested it.

The complete design successfully tested in lab with noise source and then with antenna signals to demonstrate working of it's functionality. The fringes i.e. cross correlation count was observed and seen the effect on cross correlation due to

1. 2-Antenna same polarization.
2. 2-Antenna different polarization.

In case-I fringe i.e. cross correlation amplitude and undistorted phase can be seen.

In case-II fringe amplitude is very low and cross phase is distorted indicating problem in subsequent analog receiver chain.

Thus showing one of the debugging technique using the Fringe Tester.

7.2 Future Work

The design needs additional features to be called as the full-fledged Fringe Tester.

1. Fractional delay and fringe stop functionality can be added.
2. The subsequent python module to configure fractional delay and fringe stop block can be added.
3. Automatic delay calculation and update features then can be added later.

Chapter 8. References

[1] https://casper.berkeley.edu

[2] www.gmrt.ncra.tifr.res.in

[3] http://www.tutorialspoint.com

[4] www.xilinx.com

[5] http://www.mathworks.in

[6] Understanding Digital Signal Processing, Second Edition By Richard G. Lyons

[7] “Digital Signal Processing” by Proakis.

[8] “Learning Python” 2003 ”SECOND EDITION” by Mark Lutz

[9] “PPS Generation on Serial Port using Python and Processing It” By Tambade Dhiraj