The Pennsylvania State University

The Graduate School

INTERFEROMETRIC TECHNIQUES IN SOFTWARE DEFINED RADARS

A Thesis in

Electrical Engineering

by

Tejas Nagarmat

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2012

ABSTRACT

The incoherent scatter technique for ionospheric research has proven its ability to measure most of the parameters of interest that define the ionosphere. Penn State has been developing advanced cost effective instruments and technologies for future meteor radars to study the basic properties of the global meteor flux, such as average mass, velocity and chemical composition. Cross-correlative interferometric techniques not only allow us to accurately determine the trajectory and the speed of meteors, but also help in overcoming the geophysical clutter observed at these altitudes.

The Applied Signal Processing and Instrumentation Research Laboratory (ASPIRL) at Penn State has developed a state-of-the art radar instrumentation, the Penn State University Software Defined Radar (PSUSDR), by developing a generalized instrumentation core that can be customized using specialized output stage hardware using low cost field-programmable gate arrays (FPGAs). The use of open source software tools and a generalized object oriented software framework make this system a promising proposition for all future radar research.

In this work, I propose the implementation of general post processing interferometric techniques for PSUSDR and the specific modifications to suit the needs of the Poker Flat Incoherent Scatter Radar (PFISR) facility, Alaska. The instrument design concepts and some of the emerging technologies developed for this meteor radar are also discussed followed by simulations and analysis for the same.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would first and foremost like to thank my advisor, Dr. Julio Urbina for introducing me to this project and providing me with timely guidance. I would also like to thank Dr. John Mathews for being the second reader and for his efforts in providing me with the information and guidance on interferometry.

I thank my senior colleague Ryan Seal in a major way for guiding me with the technicalities of PSUSDR. I thank my lab mate Freddy Galindo for his support and information on interferometry. Also, this thesis would not have been possible without the support of my lab mates and colleagues, Alex Hackett, Burak Tuysuz and Hakan Arslan.

Lastly, I would like to thank my parents and dedicate this work to them for their support throughout my time at the Pennsylvania State University.

Chapter 1

Introduction

The field of meteor science is fascinating and requires wide array of knowledge from the fields of astronomy, signal processing, electromagnetics, plasma physics, radar science and detection theory. It is interesting that very small meteor bodies hundreds of kilometers above the earth's surface are detected with incredible accuracy.

High-Power Large-Aperture radars detect these incredibly fast moving meteor head echoes with a range-rate velocity which follows the meteoroid as it travels through the upper atmosphere [1]. The topic of meteor "head echoes" has become an area of interest recently as scientists have focused on the importance and usefulness of meteors [2][3][4][5][6][7]. Head echo measurements give accurate radial velocities which are along the radar line of sight and altitude ranges of deposition. Powerful narrow-beam radars can also measure head echoes and are able to detect many smaller meteors as compared to classical meteor radars [8]. The use of these radars allows us to study the population of meteors which probably contributes to the most extraterrestrial material to the Earth's upper atmosphere.

Cross-correlation interferometry is a popular technique used in radio astronomy to determine the position of point to point radio sources where random signals received at two spaced antennas are cross-correlated. The cross-correlation can be represented by a complex number whose amplitude depends on the size of the source and the phase on the source position with respect to the center plane perpendicular to the interferometer axis [9]. An extension to this technique has been applied at the Jicamarca Radio Observatory (JRO), Peru wherein the direction of the magnetic field is measured by delaying the signal of one antenna by relatively long time before it is correlated with the signal received at the other antenna. The technique in this work however differs in the way that the cross correlation methods are applied in the post processing stages which also have been applied at the JRO.

The PSUSDR currently uses software defined radio (SDR) methods to observe and study these meteor head echoes. The radar system supports four RF receive channels and operates at 49.92 MHz at the Rock Springs Radar Site, 15 miles from Penn State University. In order to provide an interferometric option to the system at PFISR, the PSUSDR has been used as a proof of concept. The data collection process for the receiver is exactly the same as PSUSDR with the difference being in the RF Front End responsible for the analog down conversion of the incoming Radio Frequency (RF) signal in the order of 400 MHz to an Intermediate Frequency (IF) which is suitable for the processing in the digital radar receiver. The RF Front End has been validated in the laboratory environment along with the testing of constant phase difference signals coming into the different channels of the radar receiver.

1.1 Motivation

The altitudes where most of the meteor head echoes are expected occur between 90 and 120 km. There exists a lot of geophysical clutter at these altitudes due to the presence of Equatorial Electro Jet (EEJ) and non-specular meteor trail echoes. In addition to improving the data acquisition systems, it is common practice to make the observations around sunrise so that there is an increase in the number of meteor observations since the EEJ echoes are expected to be weaker or more sporadic. Other special processing methods can be employed to overcome these clutter returns like using interferometry or pulse coding [1].

Intereferometry greatly increases the quality of the meteor head echo returns and will be useful if applied at the PFISR and the PSUSDR facilities. Another source of motivation for this work is the power of the ever growing open source community and that of software defined radio which are the building blocks of PSUSDR. The accuracy, cost effectiveness and the ease of setup and configuration of PSUSDR are the main attractions of this system and make a sound suggestion for future radar experiments.

1.2 Organization

Chapter 2 aims to review several key concepts and technologies associated with Software Defined Radars in order to update the reader with the latest developments in the field of Software Defined Radios. A significant part of this chapter explains the GNU Radio project and the general architecture of the Universal Software Radio Peripheral (USRP). The FPGA component of the USRP is also briefly discussed for a basic understanding of the signal processing inside the USRP.

The next two chapters are dedicated to the description of the setup and operation of the PSUSDR. Chapter 3 describes the setup of different components of the PSUSDR and their interconnections, with a view to represent the systemic implementation. The importance of RF chains is enormous when collecting high quality, noise free data and thus, the configurations of the present RF chains at the PSUSDR and the one suggested for PFISR are detailed in this chapter. After explaining the validation of the proposed RF Front End, the chapter shifts the focus towards other significant components of PSUSDR such as the Radar Controller, the Clock and the Radar Transmitter.

Chapter 4 digs into the nuances of the hardware adaptations for the building of GnuRadar, the open source software project responsible for the radar data collection. The key radar concepts and the tuning details of the USRP are explained to make the reader comfortable with configuring the radar as desired. It also talks about the key binaries of this software and thus attempts to explain the configuration and the working of GnuRadar. The final section of this chapter explains the data interpretation and analysis and describes the various tools available to have a better understanding and to help in better diagnosis of the received data.

Chapter 5 explores the theory of interferometry and explains the algorithms and routines which contribute to the post processing techniques applied to the radar data. The routines are tested for raw data obtained from JRO and the PSUSDR. Further, the simulations for constant phase difference signals coming into the different channels of the radar receiver are discussed along with their analysis. A brief explanation of the Graphical User Interface for interferometry is provided at the end of this chapter.

Final remarks and thoughts on how this work can be improved are provided in Chapter 6.

Chapter 2

Background for Software Defined Radars

This chapter provides a review of several key concepts and technologies involved in working with software defined radars. The PSUSDR finds its roots in Software Defined Radio technologies. It is thus imperative to have a basic understanding of concepts in software defined radio technology and signal processing.

2.1 Software Defined Radio

A software-defined radio, as the name implies, is a radio system that makes use of software in the processing of communication signals. The physical implementations of radio communications have changed significantly over the years although the fundamental theories behind these systems have remained relatively constant for the past fifty years [10]. This means that radio communications which were traditionally handled with the help of hardware are instead implemented by means of software on a personal computer or embedded computing devices [11]. Moreover, advances in the digital signal processing capabilities of the microprocessors have helped in the acceleration of these SDR systems to performance levels that exceed those of hardware radio systems.

SDR systems also bring flexibility, which allows us to change the operations with changes in software instead of changing the underlying hardware architecture. A practical SDR system consists of the following sections [12]:

RF Section: This block consists of antenna and RF front-end. The antenna covers the spectrum linked to entire range of operation for the purpose of both transmission and reception of the signal whereas RF front-end is responsible for tuning, detection, signal transmission and analog up conversion (for transmission from IF) or analog down conversion (to IF during reception). With increased bandwidth the radio becomes more vulnerable to noise and interference.

IF Section*:* This section plays most important role in digital radio because the analog-to-digital conversion (ADC) of the down-converted IF signal for reception and digital to analog conversion (DAC) for transmission take place here. This block also carries out digital up-conversion (DUC) or digital down-conversion (DDC) to make high frequency data compatible with installed computer resources. It also performs the most important function of modulation and demodulation.

Baseband section*:* The baseband section contains the processing of the actual information and deals with operations linked to security protocols, correlation etc.

Data section*:* This section acts as an interface to a Personal Computer (PC), for instance, to program and develop intelligent controls and different routines for the SDR system.

The above sections can be observed clearly in the approach shown below of a typical example shown in Figure 2-1 of an SDR where the RF to IF conversion is before the ADC [13]. The aim of any software defined radio system is to bring the ADC as close as possible to the receiving antenna such that most of the IF conversion and processing can be done digitally. This example cascades a superheterodyne receiver RF-to-IF down converter ahead of an ADC and I/Q transformation in digital mixers. The choice of IF values add speed and precision to ADC considerations. Once digitized, a mixer built from two digital multipliers performs signal-I/Q conversion with precision that is virtually independent of local-oscillator frequency. The local oscillator is a digital synthesizer that uses sine/cosine look-up tables and a phase accumulator to generate samples of two sine waves precisely offset by 90 degrees because the mixer runs at the ADC's native sample rate, the local oscillator varies the mix-down frequency by adjusting the phase advance between sine and cosine.

Digital mixers can down convert to zero frequency without dc-offset or significant image problems, allowing a low pass filter to extract signal content. This filter is normally a decimating FIR (finite-impulse-response) design, in which the decimation factor sets filter bandwidth and dictates the output-sample rate to the baseband processor. This composite local oscillator/ mixer/filter block is available as DDC.

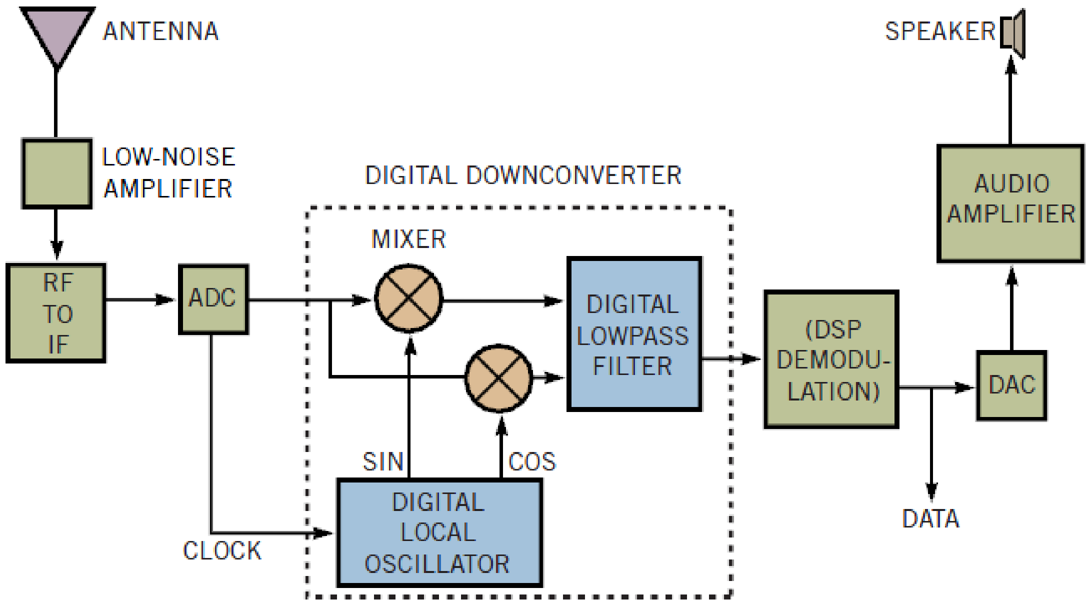


Figure 2-1. Software Defined Radio

In SDR systems, many designers prefer to include DDC functions within other logic, such as FPGAs or DSP arrays that can also serve baseband-processing duties. The shift from narrowband voice to wideband data-oriented services has a huge impact on processing power that conventional DSPs find hard to address [13]. Conventional processors are limited by fixed data paths and finite clock speeds, and their generalized nature often puts far too much processing power in the pathway for the task in hand. An FPGA allows us to break down the problem into multiple parts to perform relatively simple operations in parallel at a very high speed.

2.2 GNU Radio

GNU Radio is an open-source software development toolkit that provides the signal processing runtime and processing blocks to implement software radios using readily-available, low-cost RF hardware and processors. It is widely used in hobbyist, academic and commercial environments to support wireless communications research as well as to implement real-world radio systems [14].

Eric Blossom, Matt Ettus, et. al. have made this project possible which can turn an ordinary PC into a good quality radio receiver whereby the only additional hardware required are a low-cost RF tuner and an ADC to convert the received signal into digital samples.

The open-source software development toolkit in GNU Radio allows us to develop a custom, non commercial radio receiver by combining and interconnecting appropriate software modules, which are independent functional blocks. With the GNU Radio approach, the designer is a software developer who builds the radio by creating a graph (in a similar way to what happens in the graph theory) where the vertices are signal processing blocks and the edges represent the data flow between them [15]. The signal processing blocks are normally implemented in C++, whereas the graph structure is defined in Python.

GNU Radio applications are primarily written using the Python programming language, while the supplied, performance-critical signal processing path is implemented in C++ using processor floating point extensions where available. Thus, the developer is able to implement real-time, high-throughput radio systems in a simple-to-use, rapid-application-development environment [14]. GNU Radio is used either to implement real and working radio equipments, or just for research in the area of wireless communication and transmission.

2.3 The Universal Software Radio Peripheral

Universal Software Radio Peripheral (USRP: pronounced “U-Surp”) is a general purpose hardware module developed exclusively for use with GNU Radio, by Matt Ettus and his team at the Ettus Research LLC [16]. This board can support a wide variety of daughterboards which perform the transmit and receive functions. Since the microprocessors and the DSP’s are on board, this device becomes convenient and cost effective for researchers working with radio communication systems.

The USRP platform hosts the following major modules:

- Four 64-MS/s 12-bit, 85-dB SFDR analog-to-digital (ADC) converters (AD9862),
- Four 128-MS/s 14-bit, 83-dB SFDR digital-to-analog (DAC) converters (AD9862),
- An FPGA that can be reprogrammed (Altera Cyclone EP1C12Q240C8 FPGA),
- A high-speed USB 2.0 interface (Cypress EZ-USB FX2) that can send up to 16 MHz of RF bandwidth in both directions,
- 4 extension sockets (2 TX, 2 RX) in order to connect 2–4 daughterboards,
- 64 GPIO pins available through 4 BasicTX/BasicRX daughterboards (16 pins each)

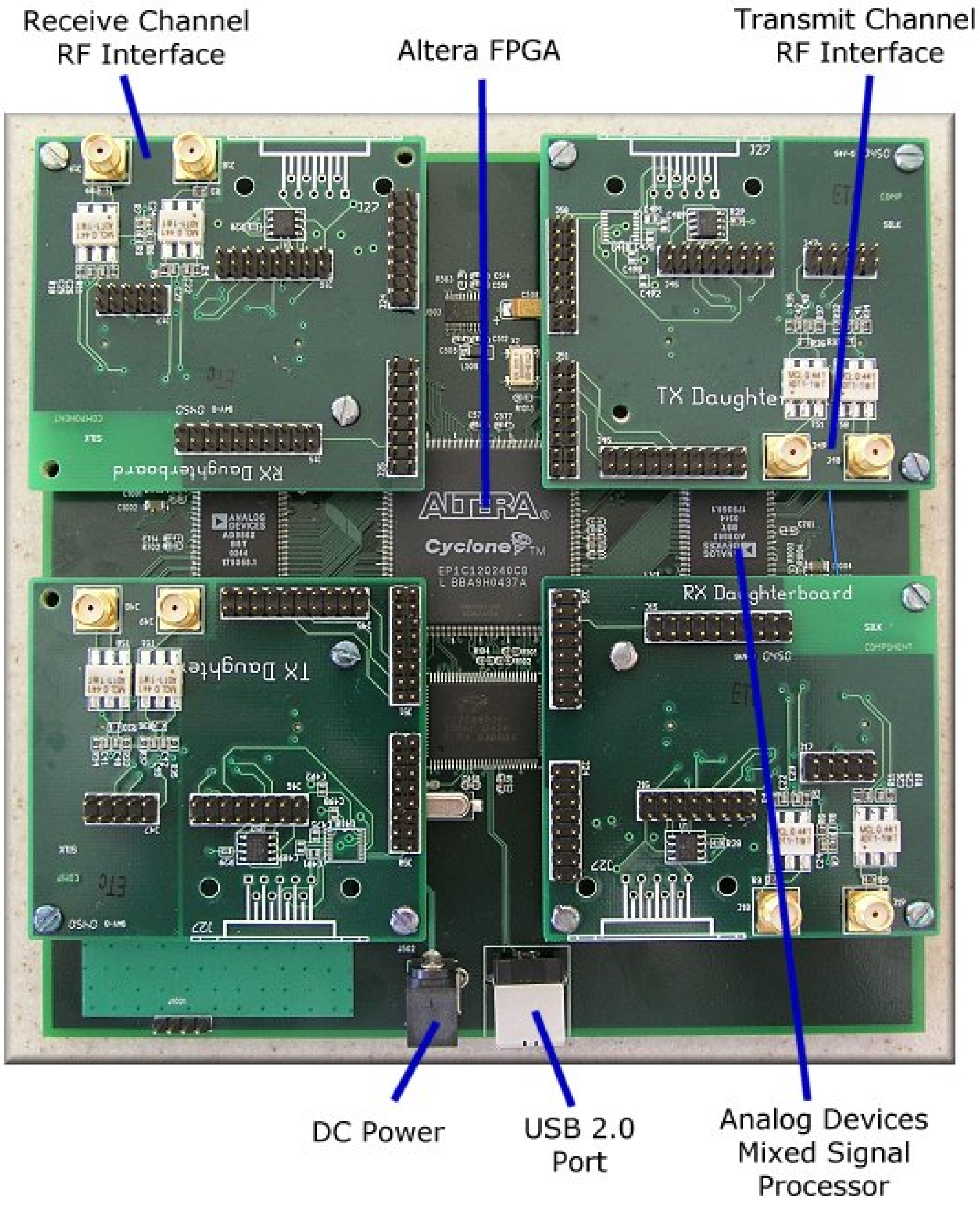

Receive Channel RF Interface
Altera FPGA
Transmit Channel RF Interface
DC Power
USB 2.0 Port
Analog Devices Mixed Signal Processor

Figure 2-2. Universal Software Radio Peripheral

2.3.1 Aliasing:

Aliasing is a major concern with the ADCs and DACs when using the USRP. When the sample rates after the ADC are down-converted to IF, it is important to note that the maximum sampling rate of ADC is, at most 64 MSamples/ sec. Therefore, the highest sampling frequency allowable to avoid aliasing is 32 MHz. Similarly, the DAC limits the transmitted frequency to a maximum of 64 MHz because of the 128 MS/s sampling rate of DAC.

2.3.2 Daughterboard:

The USRP is capable of using up to 4 daughter boards depending on the daughterboard needed for the specific job. Each of them can be used for different jobs and range for different frequency range, amplification, filtration and tuning capabilities. Since this document describes the radar receiver as an application, the daughterboard used for this purpose was the Basic Rx only.

The basic Rx contains no filtration or amplification and does not have mixer capabilities. An RF front end is not built-in with this daughterboard but can be attached to it. The operating range of these boards goes from 1MHz to 250MHz [17].

2.3.3 The FPGA:

The FPGA is the heart of the signal processing being carried out in the USRP. The ADC's and the DAC's are basically connected to the FPGA. The FPGA performs high bandwidth math to reduce the data rates to suit the speed of the USB2.0. The FPGA connects to a USB2 through the interface chip, the Cypress FX2.

The standard FPGA configuration includes digital DDC implemented with 4 stages Cascaded Integrator-Comb (CIC) filters. CIC filters are very high-performance filters which use only adds and delays. The standard FPGA configuration implements two complete DDCs. Each DDC has two inputs I and Q. Each of the 4 ADCs can be routed to either I or Q input of any of the 4 DDCs. This allows for having multiple channels selected out of the same ADC sample stream.

Figure 2-3 [18] below shows the block diagram of the USRP digital down converter.

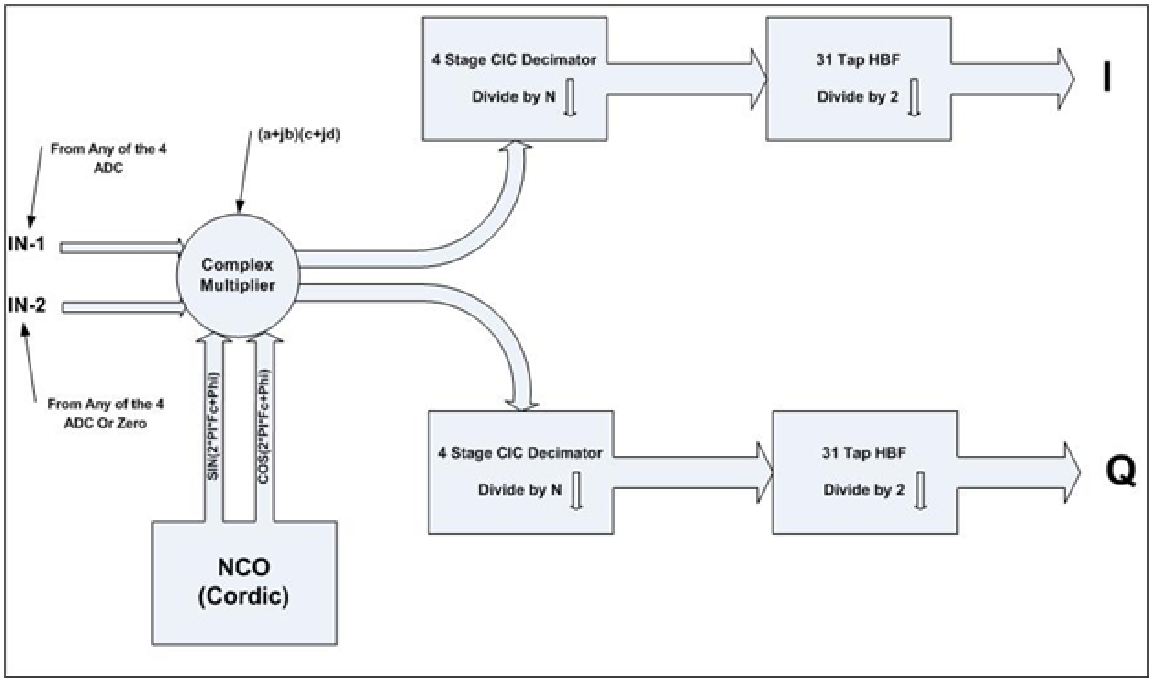


Figure 2-3. USRP Digital Down Converter

The DDC down converts the signal from the IF band to the base band and then decimates the signal so that the data rate can be adapted by the USB 2.0 and is reasonable for the computers' computing capability. The complex IF input signal is multiplied by the constant frequency exponential signal. The resulting signal is also complex and centered at 0 after which the signal is decimated with a factor of N.

The decimator is basically like a low pass filter followed by a down sampler. Given that the decimation factor is N, this means that one out of every N samples is selected. In the frequency domain, this means that a signal in the frequency range [-Fs, Fs] is reduced to [-Fs/N, Fs/N].

The total bandwidth is also an important consideration. All samples sent over the USB interface are 16-bit signed integers in IQ format, i.e. 16-bit I and 16-bit Q data (complex) which means 4 bytes per complex sample. Given that we can sustain 32MB/sec across the USB, 8Mega complex samples/sec can be sent across the USB giving us a maximum system bandwidth of 8MHz to work with.

Chapter 3

Systems Implementation

GnuRadar [19], the specialized radar receiver which is an open source adaptation of the GNU Radio project developed by ASPIRL is fully configurable and operable in software. The PSUSDR uses the USRP Version1 (Rev 4.5) and the Basic Rx to receive meteor head echoes. This radar currently supports four RF receive channels and operates at 49.92 MHz at the Rock Springs Radar Site, 15 miles from Pennsylvania State University.

In this section the key concepts in hardware that implement the radar system at PSUSDR are described. The radar system consists basically consists of a high voltage transmitter and a transmit antenna on the transmit chain and a receive antenna, an RF Front End chain and the USRP in the receive chain.

3.1 The Front End Chain

In any radio receiver, the RF front end is always the most critical aspect of the system which decides the quality of the information to be processed. The RF Front End is generally the circuitry between the antenna and the IF stage. This RF chain processes the incoming signal at the original RF and uses down converters and low noise amplifiers to accentuate the signal of interest and suppress the incoming noise.

3.1.1 The PSUSDR RF Front End:

PSUSDR currently uses the RF front end chain as shown in the Figure 3-1.

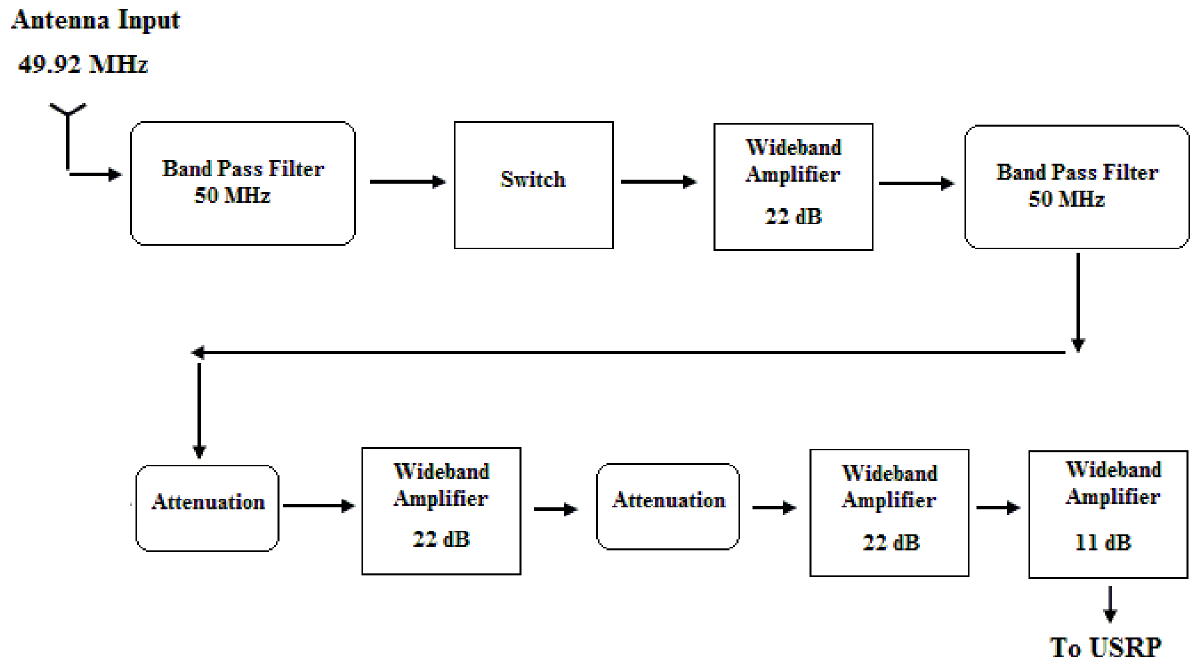


Figure 3-1. RF Front End at PSUSDR

The receive antenna is currently tuned to receive signals at 49.92 MHz and the incoming signal is first passed through a 50 MHz Band Pass Filter to filter out the noise in the other frequency ranges. This is followed by a Transmit/Receive (T/R) TTL switch which is programmed to operate in the time slots that the transmitter is not sending the transmit pulses. Currently, the T/R switch blanks the receiver chain with a pre-window of 10 microseconds and a post-window of 14 microseconds for a 77 microsecond long transmit pulse. This helps in the reduction of electrical noise and interference caused by the transmitter which is co-located with the receive chain.

The switch is followed by a series of wideband amplifiers and attenuators to achieve a theoretical gain of approximately 60 dB to achieve a good Signal to Noise Ratio (SNR). As observed in the tests, the sky noise coming in from the antenna has been determined to be around

– 60 dB and any gain provided to bring the signal to 0 dB or more is sufficient for the USRP to process.

The attenuators added in the configuration presently have 6 dB attenuation and they precede the amplifiers such that the amplifiers do not go into saturation and distort the incoming signal. The Noise Figure (NF) can be computed by Friis' Formula as follows [20]:

$$\mathrm{NF} = 10 * \log_{10}\left(F_1 + \frac{(F_2-1)}{G_1} + \frac{(F_3-1)}{G_1G_2} + \frac{(F_4-1)}{G_1G_2G_3} + \cdots\cdot + \frac{(F_N-1)}{G_1G_2G_3\cdots G_{N-1}}\right) \tag{3.1}$$

where F_N is the noise factor and G_N is the gain of the Nth device in the chain. Considering insertion losses in the filters of 1.8 dB at the given frequency, the calculations show the following:

- Total Gain = 60.3 dB
- Noise Figure = 3.54 dB

3.1.2 Front End Chain Design for PFISR:

In order to perform interferometric post processing for PFISR at Poker Flat, Alaska, a down conversion chain was designed to suit the IF requirements of 30 MHz from an incoming RF signal of 450 MHz. The RF front end chain in this case is depicted in Figure 3-2 shown below. This RF chain differs in the functionality such that it performs analog down conversion to IF such that it suits the processing power of the USRP. The antenna input comes into the chain at a frequency of 450 MHz which is passed through a High Pass filter centered at 400 MHz to reject the noise from lower frequencies. This signal is then amplified and mixed with a Local Oscillator (LO) frequency of 420 MHz to produce a difference component in the frequency of 30 MHz to be amplified and filtered again for better SNR.

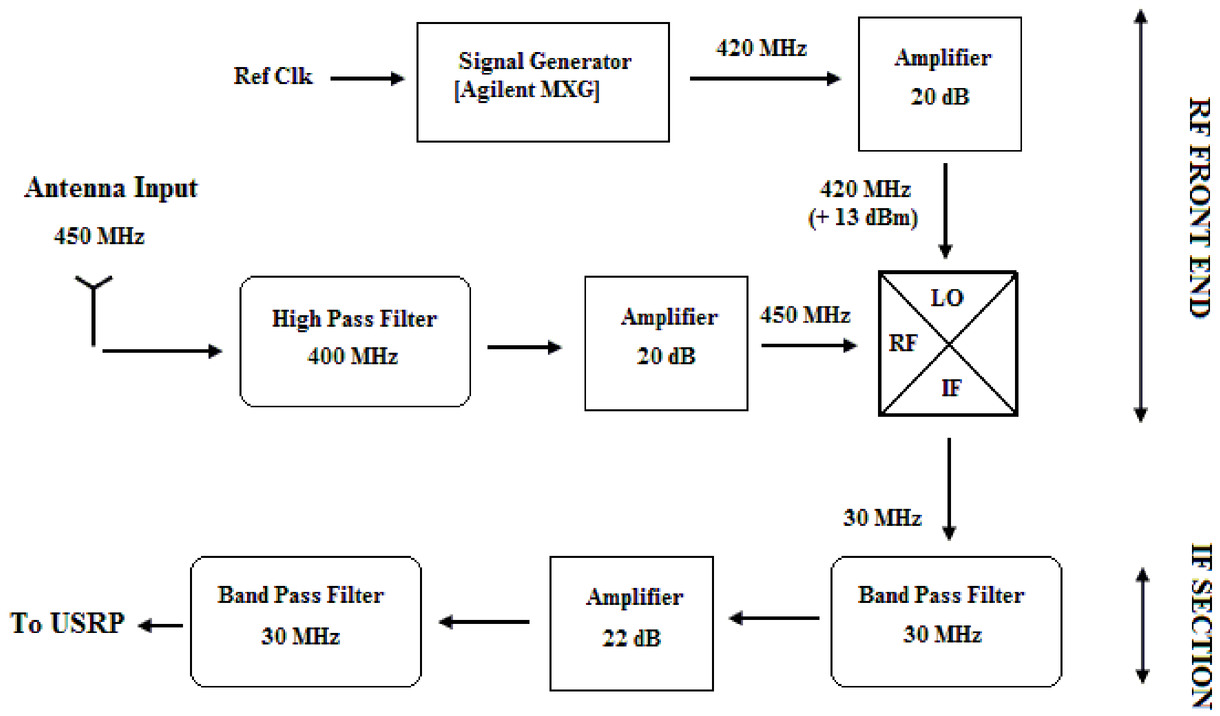


Figure 3-2. RF Mixer chain for PFISR, Alaska

The theoretical calculations for the chain show the following:

- Total Gain = 34.58 dB
- Noise Figure = 4.42 dB

3.1.3 Front End Design Validation:

In order to verify the accuracy of the design above, it was modified to suit the PSUSDR and produce an IF of 30 MHz from an RF input of 49.92 MHz. The incoming signal is trusted to be of good quality with good noise rejection since the input for this down conversion chain would be from the output of the present RF chain shown in Figure 3-1.

The modified design for the purpose of validation is shown as follows:

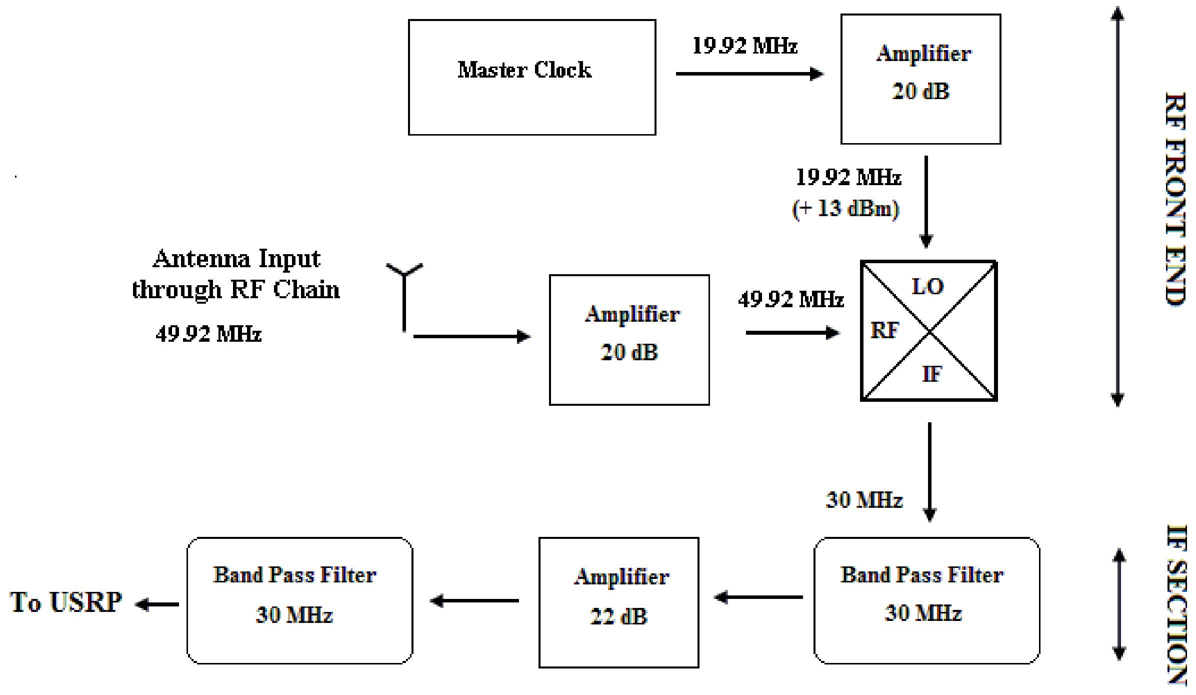


Figure 3-3. Modified Mixer chain to suit PSUSDR

The LO in this case is given by the Master Clock which provides the clock to the rest of the radar system. This helps to preserve the synchronization present in the radar system. The calculation results for this chain show:

- Total Gain = 35.34 dB
- Noise Figure = 3.81 dB

When cascaded with the pre existing RF chain, the calculations show a Noise Figure of 3.54 dB. As indicated in the figure above, the output was fed to the USRP which was tuned to receive signals at 30 MHz. Figure 3-4 displays the IQ diagram at 30 MHz obtained from this setup for two channels while Figure 3-5 shows the spectrum plot seen at the input of the USRP which shows a very good power response at 30 MHz. Both these plots obtained from the tests verify the RF design for PFISR.

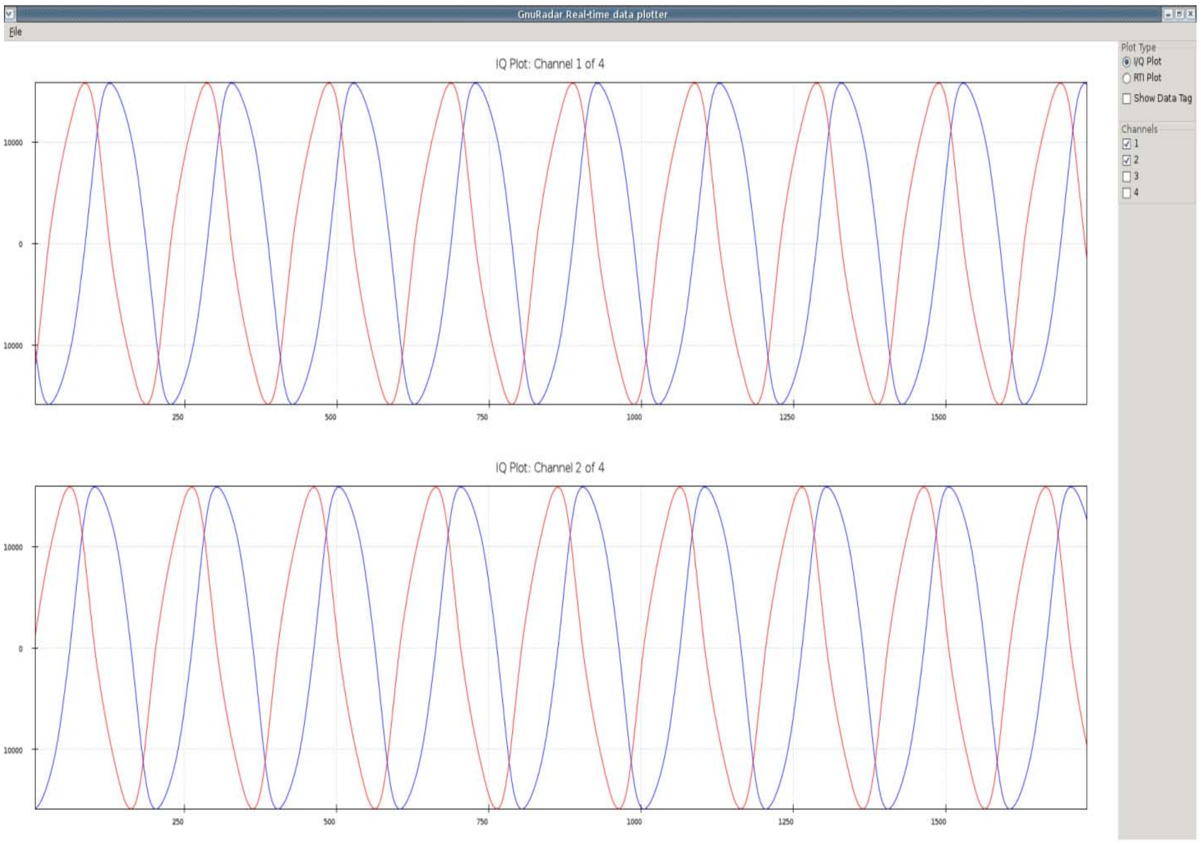
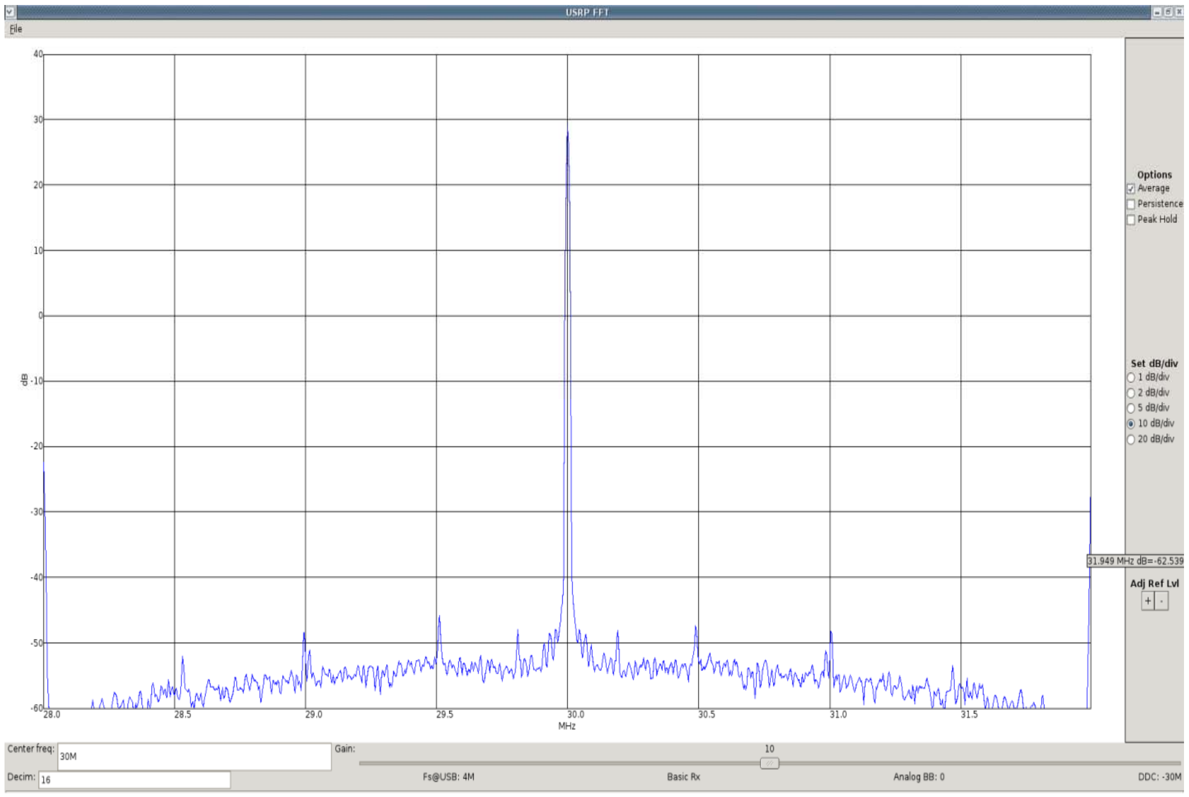


Figure 3-4. I-Q Plot obtained from the modified mixer chain at 30 MHz



[✓] Average
[ ] Persistence
[ ] Peak Hold

Set dB/div
1 dB/div
2 dB/div
5 dB/div
[✓ 10 dB/div]
20 dB/div

31.949 MHz dB=-62.539

Center freq: 30M
Decim: 16
Gain: 10
Fs@USB: 4M
Basic Rx
Analog BB: 0
DDC: -30M

Figure 3-5. USRP Spectrum Plot for modified mixer chain at 30 MHz

3.2 The Radar Controller

The Radar Controller, as the name suggests is the central control point and can be viewed as the heart of the radar system. It is responsible for the management of the receive and the transmit chains. To understand the full functionality for it is essential to understand the radar system as a whole.

The PSUSDR setup is depicted in the Figure 3-6, which shows the interconnections and relations between the various components of the radar system. It uses Gentoo Linux OS and the Opal Kelly Front Panel software.

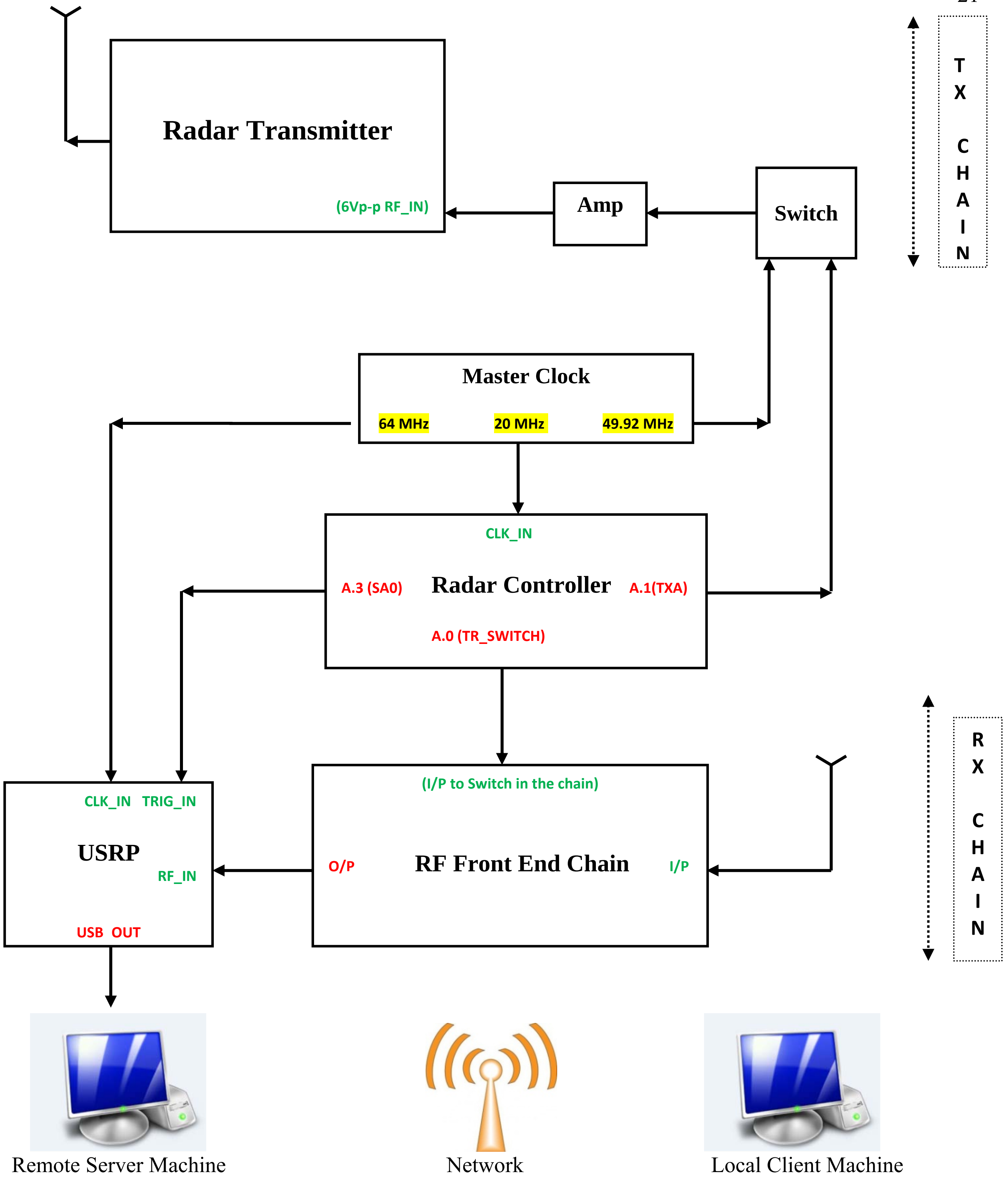


Figure 3-6. The PSU Radar System Block Diagram

On the hardware front it uses:

- Spartan 3A FPGA (XC3S400)
- Opal-Kelly FPGA Interface (XEM3001)
- Custom Designed FPGA carrier board
- 2 TTL Line driver boards
- signal translator board
- signal input and output connectors
- 80 GB HDD
- ATX PC power supply

The purpose of the controller is to control the output window of the transmitter signal via various user defined, high precision pulse signals while taking care not to drive the system beyond safe operating limits. The design presented utilizes a variety of open source tools and software, along with a consumer class FPGA to implement a highly reconfigurable robust, user friendly and cost effective radar controller system. Thus, it is a generalized Bit Pattern Generator [21] based on plain text files defining the output of the system (both digital and analog). The important definitions of the Inter Pulse Period (IPP), reference Clock Frequency and Baudwidth must be defined in these files.

3.2.1 Configuration:

For example, in the text file, a.11 = { +2 -5 +17 -9 +10 -6 } translates to "logic high for two bauds, low for 5 bauds, high for 17 bauds, low for 9 baud, high for 10 bauds and low for 6 bauds " on port A and channel 11. Before the bit pattern is downloaded to the FPGA, it must be translated in a friendly form for the FPGA to read. This is accomplished by the bpg-generate program which can be issued in the following steps:

```
➢ $ ./bpg-generate –o output.iif input.hif
➢ $ cp output.iif /usr/local/bpg/modes/
```

This program compiles the human defined controller parameters into machine readable format (iif) that can be downloaded to the FPGA. The /usr/local/bpg/modes/ is the location where all the configuration files are loaded from. An example of the configuration file used for the present setup at PSUSDR is explained below.

<Input.hif>

```
INSTRUMENT = RPG

#Sub Instrument Definition
RULES     = PSU1    # Several Sets of Transmitter rules can be defined here

#No Functionality has been implemented for this keyword yet.
MODE      =  Normal          # Select Normal or Multi modes

#Defines inter pulse period
IPP       =  4000 usec       # InterPulsePeriod

#Defines the RPGs master clock rate
REFCLOCK  =  20 MHz          # Reference clock rate

#Defines baud width for ports A and B
BAUDA     =  1 usec          # Baud width for port a
BAUDB     =  1 usec          # Baud width for port b

#TR SIGNAL : TR Switch Pulse
```

```
#Defines (pre,post) txa settings
TR         = a.0    (14 usec, 10 usec)

#TXA SIGNAL : Transmit Enable
#Defines trasmitted width
TXA        = a.1    ( 77 usec )
CODE       = a.2    { +21 -21 +7 -14 +7 -7 }

#sampling windows - one or more can be defined
#Defines (start,stop) sampling ranges
#Optional negate signal to invert logic
SA0        = a.3    ( 0 km, 130 km)
SA1        = a.4    (100 km, 110 km) , negate

#generic signal definitions
#Defines logic level in number of bauds
#a.5,a.6  =        {+320}
a.5,a.6   =        {+5 -2 +2 -1 +1 -100}
a.7,a.8   =        {+10 -20 +30}
```

3.2.2 Rules and Operation:

The rules are defined as per the application. In this case, the PSU rules define the operation for the transmitter. One has to be careful that:

- The code width should match the transmitted pulse width
- The duty cycle which is defined as the transmit pulse width divided by the IPP of the system should be less than two percent.

If these criteria are not met, the program does not compile the output file as directed. From the radar system's point of view, the other parameters that should be taken note from the displayed Input.hif are:

- The IPP is 4 milliseconds.
- The reference clock is 20 MHz which should come from the master clock.
- The T/R pulse mentioned before which goes to the switch in the receive chain should have a pre-window of 14 microseconds and post-window of 10 microseconds for blanking the receiver.
- The TXA which is the actual transmit pulse width lasts 77 microseconds long and the CODE should have an according width.
- The Sampling window, SA0 allows to sample from 0 km to 130 km to observe radar returns for that range.

In order to run the program, the following steps need to be implemented:

- ➢ `$ ./bpg-shell`
- ➢ `clock std        # Indicates that the clock should be taken from external input`
- ➢ `add output     # Loads the output.iif from /usr/local/bpg/modes/`
- ➢ `start a  # Starts the signal transmission through the indicated channels on port A`

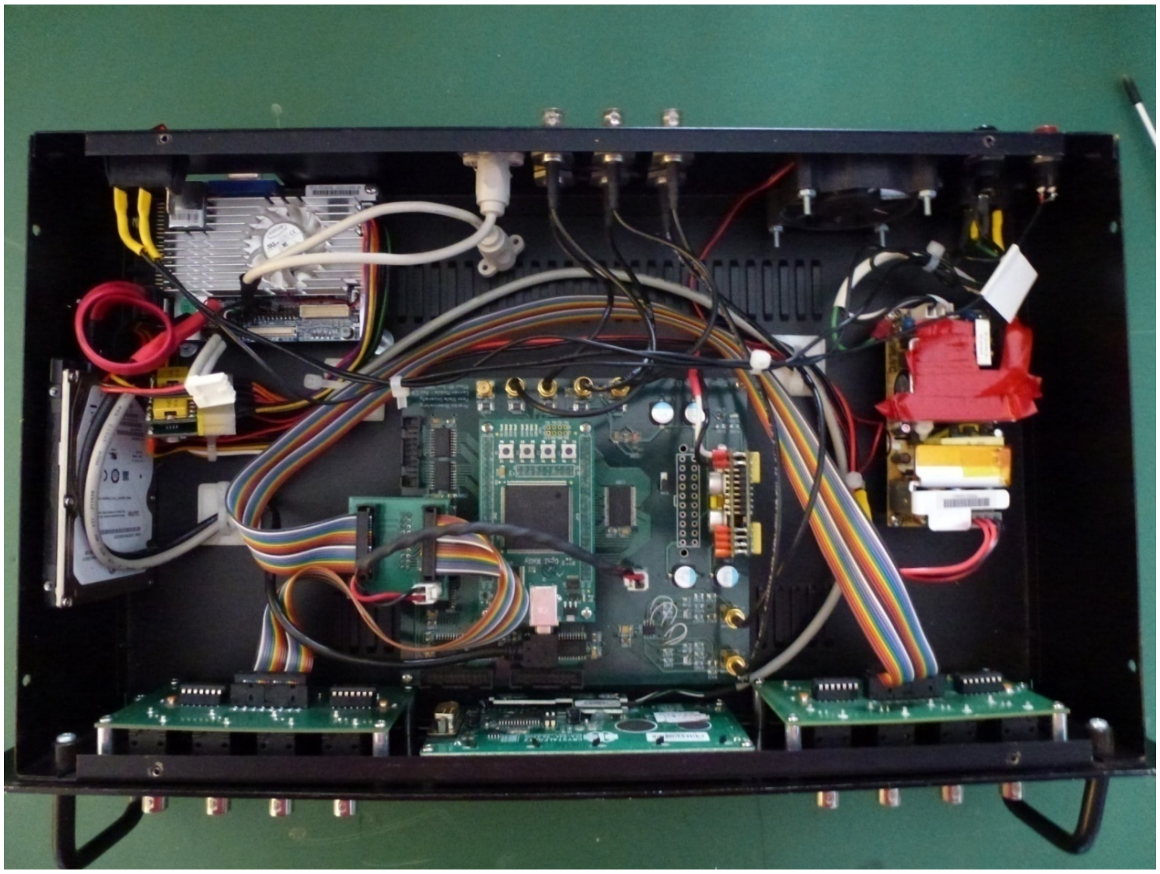


Figure 3-7. Top View of Radar Controller

3.3 Clocking

The most important component of any radar system is the clock in the system. Since the transmitter and the receiver are co-located it is very important for the transmit and receive sections of the radar system to be clocked by the same source. This helps in synchronization of the incoming data and also to gather intelligible data.

As shown in the Radar System Block Diagram, the Master Clock has three outputs, one going to the digital receiver (64 MHz to USRP), one going to the Radar Controller (20 MHz

Reference Clock) and another going to the T/R switch in the receive chain (frequency same as the operating frequency of the antenna).

The PSUSDR uses the Novatech 409B signal generator which generates four independent, phase synchronous sine wave output signals and is programmable in 0.1 Hz steps from 0.1 Hz to 171 MHz. Programming is via an RS232 serial interface [22]. The phase is 14-bit programmable and amplitude is 10-bit programmable. It requires 5 VDC external power.

Figure 3-8. Master Clock- Novatech 409B

The following describes the general how-to to configure the clock in the Gentoo Linux environment.

3.3.1 Setup:

If computer does not have a serial port, the kernel will need to be configured to allow Serial-to-USB adapters. The adapter type should be found in the following location in the kernel.

```
Device Drivers --->
        <*>USB support --->
                <*>USB Serial Converter support --->
                        [*] USB Serial Console device support
                        [*] USB Generic Serial Driver
                        <*> USB [BRAND OF ADAPTER] driver
```

Once correctly configured, one can type 'ls /dev/' to locate a ttyUSB0 device unless another USB device is plugged in. To be able to modify and operate as a user, one should look for file named '50-udev.rules' in etc/udev/rules.d and include

' KERNEL="ttyUSB[0-9]*",NAME="tts/USB%n", GROUP="tty", MODE="0666" '

in place of any other existing rule. If the file does not exist, it will have to be created.

3.3.2 Configuration:

3.3.2.1 File to write

```
$ nano [NAME OF FILE]
------------------------------------------------------------------------------------------
E D                | Disables serial echo for maximum interface speed <optional>
C I                | Enables Internal clock in case if external was previously enabled

                                     | Channel 0
F0 XXX.XXXXXXX   | Frequency in MHz (000.0000000-171.1276031)
P0 XXXXX         | Phase (00000-16383) N*360/16384 N*Pi/8192
V0 XXXX          | Amplitude (0000-1023) if N>=1024 set to full scale

F1 XXX.XXXXXXX           | Channel 1
P1 XXXXX
V1 XXXX

F2 XXX.XXXXXXX           | Channel 2
P2 XXXXX
```

```
V2 XXXX

F3 XXX.XXXXXXX          | Channel 3
P3 XXXXX
V3 XXXX

S                       | Saves the configuration to the EEPROM
------------------------------------------------------------------------------------------
```

3.3.2.2 Program the Clock

The speed should be firstly set to 19200 baud. This can be done by issuing ‘stty -F /dev/ttyUSB0 19200’ at the terminal and can be double checked by issuing ‘stty -F /dev/ttyUSB0’ to notice the baud rate of 19200 being displayed on the terminal.

Lastly, to download the file to the clock,

```
$ cat [LOCATION OF FILE]/[NAME OF FILE]  > /dev/ttyUSB0
```

3.3.3 External Clocking:

In many radar applications, it might be necessary to externally clock this signal generator and produce phase synchronous outputs. It should be remembered that the signal should be a sine wave or a square wave with a voltage level between 0.2 and 0.5 Vrms. A direct input of 1 MHz to 500 MHz can be provided.

3.3.3.1 Commands to clock the module externally:

```
C E      | To enable external clock
Kp 0a    | To set the PLL gain to be 10
#For example, setting Fout MHz output from reference Fin MHz for Kp value of 'a' in hex
#Fcommand = (Fout)*(15*28,633,115.306666667)/('10'*Fin*10,00,000)
Fn  Fcommand  | For 64 MHz output from reference 50 MHz
```

3.4 The Radar Transmitter

The present transmitter used at PSUSDR is configured to operate at 5.5 kV when input with an AC line of 208-240 Volts. The antenna is connected to the RF output of the transmitter. The various analog parameters can be determined with the help of the Monitor. It should be taken care that the operation is in 'Local' mode while operating. Table 3-1 shows the analog parameters which can be displayed on the panel.

Analog Ch No	**Parameter**	**Unit**
0	High Voltage 1 (8kV)	Volt
1	High Voltage 2 (4kV)	Volt
2	Cathode Bias Voltage, Final	Volt
3	Cathode Bias Voltage, Drive	Volt
4	Cathode Current, Final	Milliamp
5	Cathode Current, Driver	Milliamp
6	SSD Final, Supply Current	Amp
7	SSD Driver, Supply Current	Amp
8	SSD Final, Supply Voltage	Volt
9	T/R Duty Cycle	%
10	Microcontroller Supply Voltage (+5)	Volt
11	Real Supply Voltage (+28)	Volt
12	Filament Voltage, Final	Volt, RMS
13	Exhaust Air Temperature	Degrees C

14	RF Output Pulse Power, Forward	KiloWatt
15	RF Output Pulse Power, Reflected	KiloWatt
16	VSWR	

Table 3-1. Analog Parameters.

3.4.1 Machine States and Bringing the Transmitter up:

The transmitter has five machine states, four of which are called up by the operator on the keyboard. Machine state 0 is power applied to the controller and its support. Machine state 1 turns on the fan, cathode bias supplies, filament supplies and the solid state driver supplies. Machine state 2 is the intermediate step start position to high voltage. The step-start limits the inrush current when the high voltage supply is turned on. Machine state 3 is the high voltage supply turn on. Machine state 4 enables the input switch, allowing RF into the unit. At each stage, the controller will register a fault if the parameter values are out of limits. The controller will return to the preceding machine state or will not allow the operator to proceed without correcting or clearing the error.

The error codes are as shown in Table 3-2.

Error Code	**Error**
0	No Error
1	Analog Parameter error at state 0
2	Machine state 2 requested
3	Air Pressure Failure
4	Analog Parameter error at state 1
5	High Voltage fuse failure
6	Analog Parameter error at state 3

7	Analog Parameter error at state 4
8	High Voltage step start aborted for parameter error

Table 3-2. Error Codes.

The keying sequence for changing machine states is: F (for functions), machine state number, and E (for Execute). If the parameters in state 0 were correct, machine state 1 may be brought up by keying in F1E. The controller will respond with three bars in the first column of the display. In this case, the center bar represents state available, the lower bar represents state called and the upper bar represents the state active. This graph allows quick visualization and determination of current machine state.

It usually takes 300 seconds to get requested state 2 or 3 from state 1 and similarly it also takes similar time to bring it back down to state 1. If the machine state 1 parameters are acceptable, machine state 3 may be requested by keying in F3E. The display will show state 3 called, and the transmitter will proceed through state 2 to state 3. The analog parameters will show high voltage present, with the rest of the parameters unchanged from machine state 1.

The transmitter will not put out RF until it is in machine state 4. The value F4E will have to be keyed in to observe full transmission. The T/R pulse is active high. As shown in the Figure 3-9 it must lead the RF pulse by at least 2 usec (A), and lab the RF pulse by at least 1 usec (B).

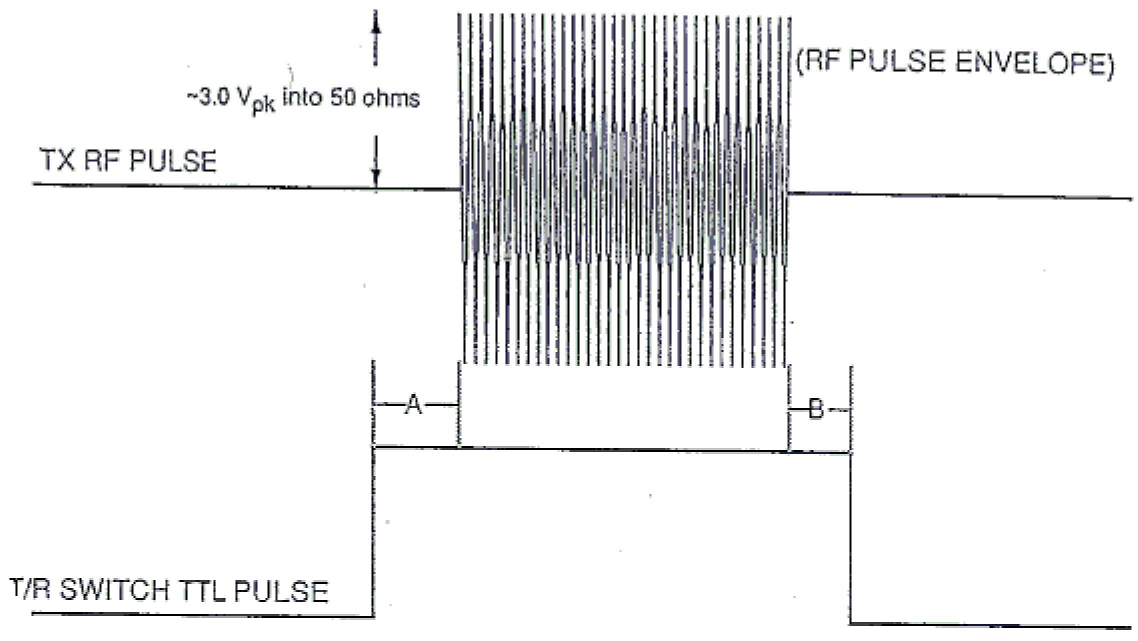


Figure 3-9. RF input for the transmitter

Chapter 4

GnuRadar

This section concentrates on the hardware and software aspects of the digital receiver component of the PSUSDR, the GnuRadar [19]. The basic hardware needed to build the receiver consists of the USRP and the Basic Rx daughterboard. The USRP needs to undergo minor changes in the hardware depending on the USRP motherboard version being used. The software uses a minimalistic version of the GNU Radio software and builds on the same using custom C++ codes to collect radar returns.

In the following sections, I aim to provide a brief manual for the setup and operation of the GnuRadar for the ease of those planning to learn and use this project for any radar application. The observations of this document have been carried out with the following system considerations:

- USRP Version1 (Rev 4.5)
- Basic Rx
- Operating System: Gentoo Linux 64 bit system

4.1 Hardware adaptations

The USRP hardware needs to undergo minor changes for the adaptation for GnuRadar. Firstly, external clocking needs to be provided to the USRP motherboard to synchronize the receiver with the rest of the radar system. Synchronization is very important in radar systems as the transmitter and the receiver need to be clocked from the same source to clearly comprehend the data received and also preserve any data alignments therein.

4.1.1 Changes to the motherboard:

The hardware changes for the motherboard are enumerated for the USRP v1 rev4.5 below. The adaptations for other versions or revisions can be found at [23]. The Rev 4 USRPs have introduced an enhancement to allow one board to be the master clock, and all other boards will become slave to it.

In this project, since we use an external master clock for the system, the USRP is used in the slave configuration. The changes needed for the motherboard are:

- Solder an SMA connector into J2001. This is the clock input. One has to be careful when soldering the SMA connector so that the delicate trace from J2001 to C927 does not break.
- Move R2029 to R2030. This disables the onboard clock. R2029/R2030 is a 0-ohm resistor.
- Move C925 to C926.
- Remove C924.
- In order to chain another USRP to of this one, one can use J2002 to provide a clock out.

4.1.2 Changes to the Basic Rx daughterboard:

The basic Rx daughterboard does not have any internal changes for the purpose of this project. However, it requires an external trigger from the system which is an indication of the receive window size it has to adapt to. It is advisable that another opening be made in the casing of the USRP for a SMA connector similar to one of the RF inputs for receiving the trigger for the basic Rx.

The J25 pin on the basic Rx daughterboard should be given a 50 ohm terminated trigger which comes from the radar controller. This can be done by slicing the coaxial cable to solder a 50 ohm resistor and kept intact with heat shrink tubing.

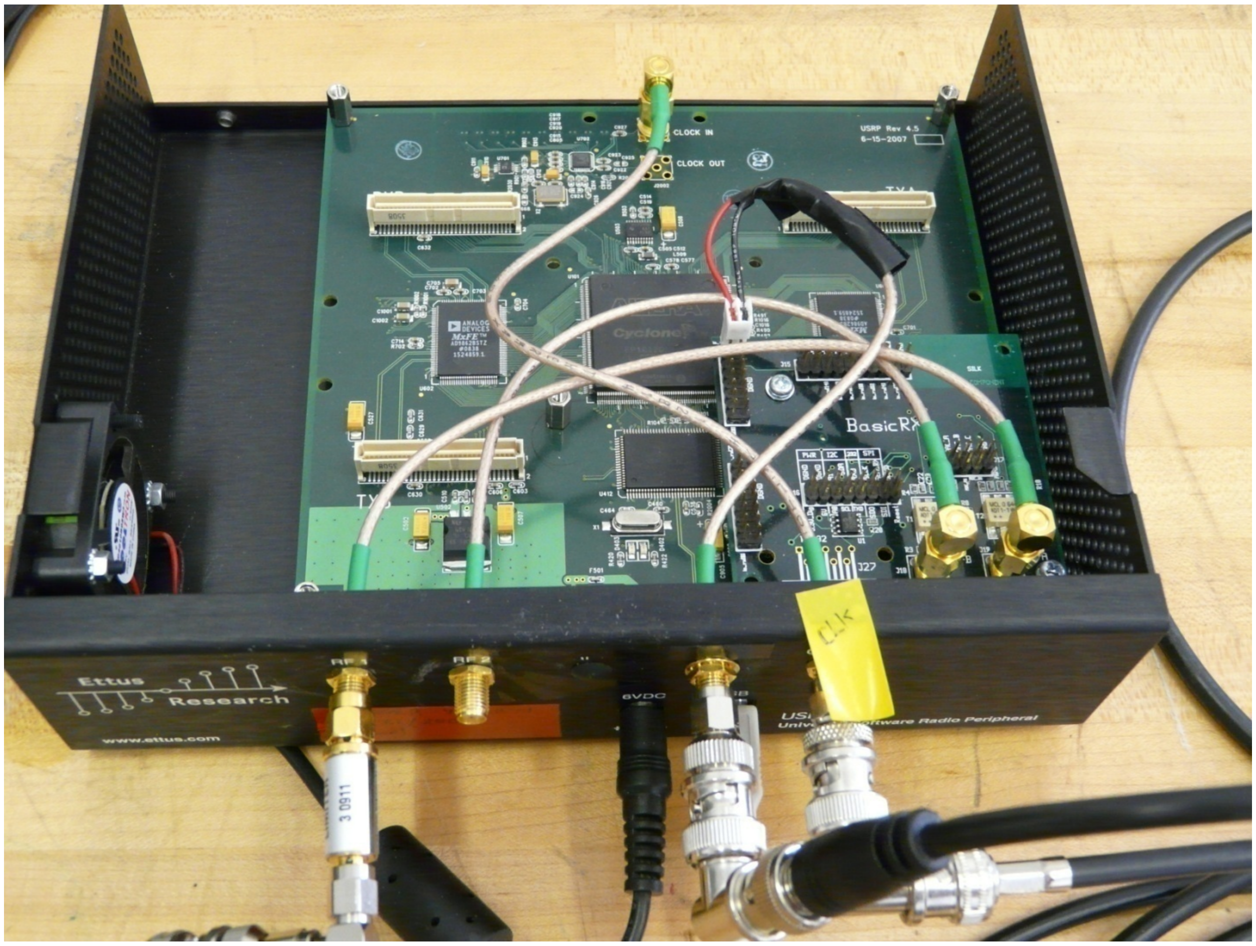


Figure 4-1. USRP for GnuRadar

It should be made sure that the USRP is identified by the computer and that the USRP is clocked well before starting the data collection. The lsusrp.py and test_usrp_benchmark.py from the GNU Radio distribution can be helpful here.

4.2 Software Configuration and Operation

This section explains the configuration and the operation of the GnuRadar software and the key theoretical concepts related therein. The build guide can be found on [19] which helps in installing the GnuRadar software. After having installed GnuRadar software successfully as directed in the readme, all the executable binaries will be installed in the 'bin' folder of the software and in '/usr/local/bin'. The software currently has three steps for operation as explained ahead.

4.2.1 Configuration – gradar-configure:

4.2.1.1 Key Concepts

1. **Aliasing: Signal Ambiguity in the frequency domain**

Periodic sampling, the process of representing a continuous signal with a sequence of discrete data values, pervades the field of digital signal processing.

In practice, sampling is generally performed by applying a continuous signal to an ADC which outputs a series of digital values. Sampling Theory plays a very important role in determining the accuracy and feasibility of any digital processing scheme [24].

Consider a sinusoidal wave with frequency f_0 which is represented in the discrete time domain where 'n' is the index on the time axis and t_s is the sampling time of the ADC. We can safely deduce the following given that 'm' and 'n' are integer values:

$$x(n) = \sin(2\pi f_0 n t_s) = \sin(2\pi f_0 n t_s + 2\pi m) = \sin(2\pi (f_0 + (m/ n t_s)) n t_s) \quad (4.1)$$

$$\therefore\ x(n) = \sin(2\pi(f_0 + k/t_s) n t_s) \quad (4.2)$$

where 'k' is an integer given that 'm' is an integer multiple of 'n'.

Further,

$$x(n) = \sin(2\pi(f_0 + k f_s) n t_s) \quad (4.3)$$

where f_s is the sampling frequency.

We can observe from equation 4.1 and 4.3 that the f_o and (f_o+kf_s) factors are equal. This means that an $x(n)$ sequence of digital sample values, representing a sine wave of f_o Hz, also exactly represents sine waves at other frequencies, namely, $f_o + kf_s$.

Thus, when sampling at a rate of f_s samples/s, if k is any positive or negative integer, we cannot distinguish between the sampled values of a sine wave of f_o Hz and a sine wave of (f_o+kf_s) Hz. This implies that no sequence of values stored in either an ADC or a computer can unambiguously represent one and only one sinusoid without additional information.

The spectrum of any discrete series of sampled values contains periodic replications of the original continuous spectrum. The period between these replicated spectra in the frequency domain will always be f_s. Thus aliasing occurs at multiples of the sampling frequency as shown in the figure alongside.

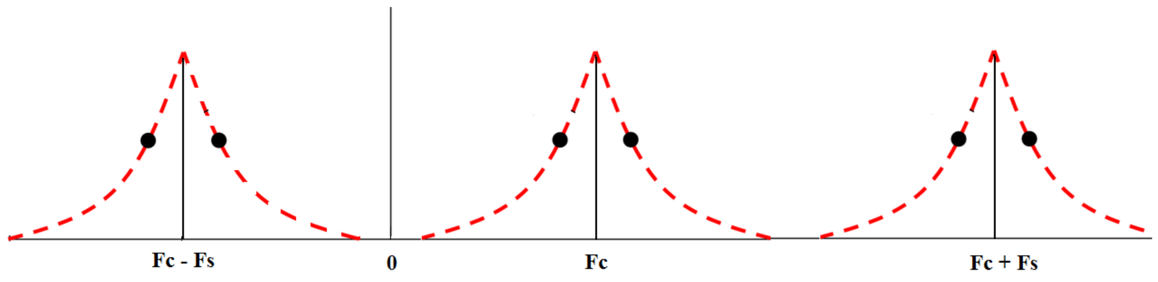


Figure 4-2. Spectral Replications

The care to be taken here is that the sampling frequency chosen should be more than twice the Fc according to Nyquist's sampling criterion such that these spectra do not overlap.

2. Tuning frequency of the USRP

As mentioned earlier, the clock sampling rate of the USRP ADC is 64 MHz and thus the sampling frequency here is preset as 64 MHz. Thus, any incoming signal with a signal Fc will have replications at (Fc $\pm$ nFs) where n is any integer.

As an example, a 30 MHz signal will have it's closest signal representations from the zero frequency at +30 MHz and -34 MHz. Thus going in accordance with the Nyquist Sampling Theorem, the USRP should be tuned to a frequency of 30 MHz. In the case of the PSUSDR, the 49.92 MHz signal is also found at -14.08 MHz which fits in the sampling criterion and the configuration should thus contain tuning frequency value of -14.08 MHz.

3. The Receiving Sample Window

As mentioned earlier, the radar controller specifies the sampling window (SA0) through channel 3 of port A to the daughterboard of the USRP. It is very important for the GnuRadar software which is totally independent of the Radar Controller software to be aware of this sampling window parameter.

For example, if the SA0 parameter as shown in the input.hif of the previous chapter extends from 0 km to 130 km, given that the signal takes twice the time to traverse the distance from the transmitter to the target and back, the time equivalent of this sampling window will be

$$\begin{aligned}\text{Time\_duration} &= 2 * (\text{Distance\_Traversed}/\text{Speed}) \qquad (4.4)\\ &= 2 * (130{,}000/\ 3\text{E}8)\\ &= 866.67 \text{ microseconds.}\end{aligned}$$

Thus the sampling window can be viewed as a signal with an on time of 866 microseconds and a total period of the IPP, i.e. 4 milliseconds.

Further, the bandwidth of the system is essentially dependent on the sampling rate of the USRP and the decimation set. As an example here, since the sample rate is 64 MHz and the decimation is 32 considering a four channel operation, the bandwidth of the system will be 2 MHz.

Thus, the number of digital samples received in this operation will be

$$\text{Samples} = \text{Time\_duration} * \text{Bandwidth} \qquad (4.5)$$

$= 866.67*10^{-6} * 2 * 10^{6}$

~ 1733

This means that approximately 1734 digital samples per channel will be received through the USRP during the GnuRadar operation.

4.2.1.2 Setting the Parameters

When the gradar-configure program is executed, Figure 4-3 appears on the screen. Some of the important parameters are explained as follows:

Sample Rate: This should be equal to the sampling rate of the USRP, which is the clock input coming from the master clock.

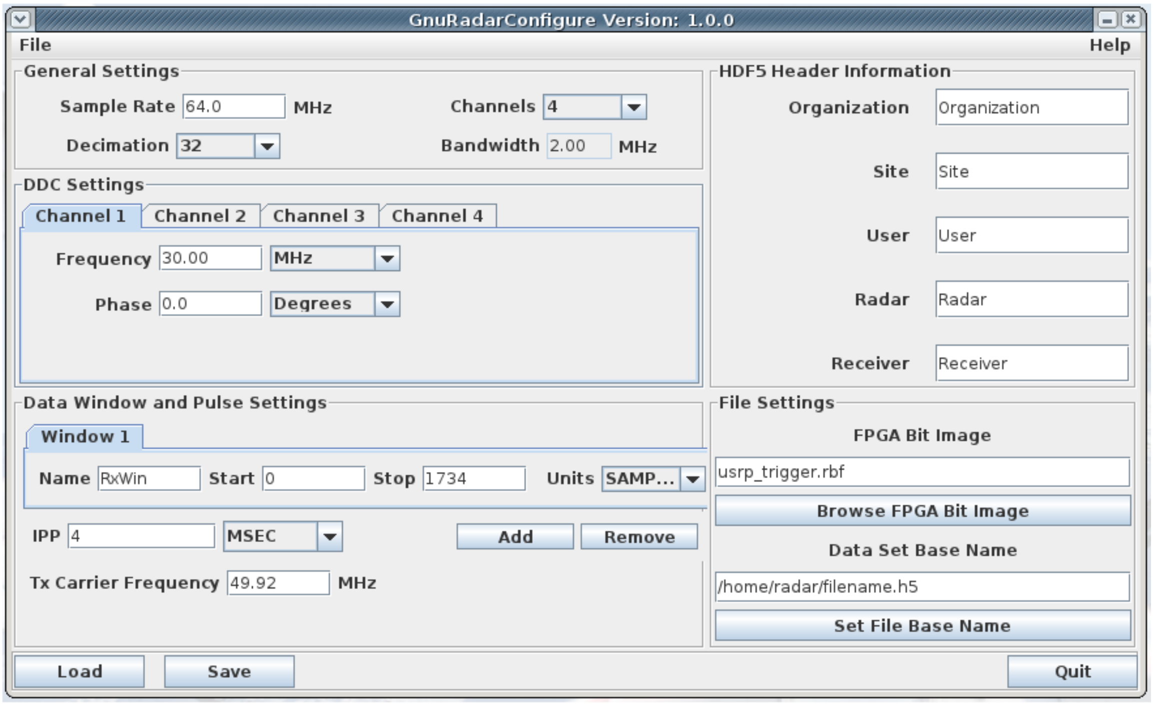
Decimation: This should be set to a minimum of 8 for one channel operation, minimum of 16 for a two channel operation and a minimum of 32 for a 4 channel operation.

Bandwidth: This value gets adjusted accordingly according to the settings of sample rate and decimation.

Frequency: The frequency tuning should be done for each channel according to the tuning concepts aforementioned.

Window Setup: The start and stop can be specified according to the number of digital samples derived as shown here or can be merely specified in terms of km as in the radar controller. The window setup currently expects a name of 'Rx_Win'

FPGA Bit Image : This is the FPGA raw binary file that will be loaded on the Altera Cyclone FPGA for the operation. This has to be a precompiled image which performs the signal processing inside the FPGA. To be able to load this file, the file should be located in /usr/local/share/usrp/rev4.



General Settings
Sample Rate 64.0 MHz
Channels 4
Decimation 32
Bandwidth 2.00 MHz

DDC Settings
Frequency 30.00 MHz
Phase 0.0 Degrees

Data Window and Pulse Settings
Name RxWin Start 0 Stop 1734 Units SAMP...
IPP 4 MSEC
Tx Carrier Frequency 49.92 MHz

HDF5 Header Information
Organization Organization
Site Site
User User
Radar Radar
Receiver Receiver

File Settings
FPGA Bit Image
usrp_trigger.rbf
Data Set Base Name
/home/radar/filename.h5

Figure 4-3. GnuRadar Configuration

Base_File_Name: This is the file name desired for the particular operation. It should be noted that this file name has to be changed for every unique receive operation. This is a precautionary measure so that the received data files are not over written.

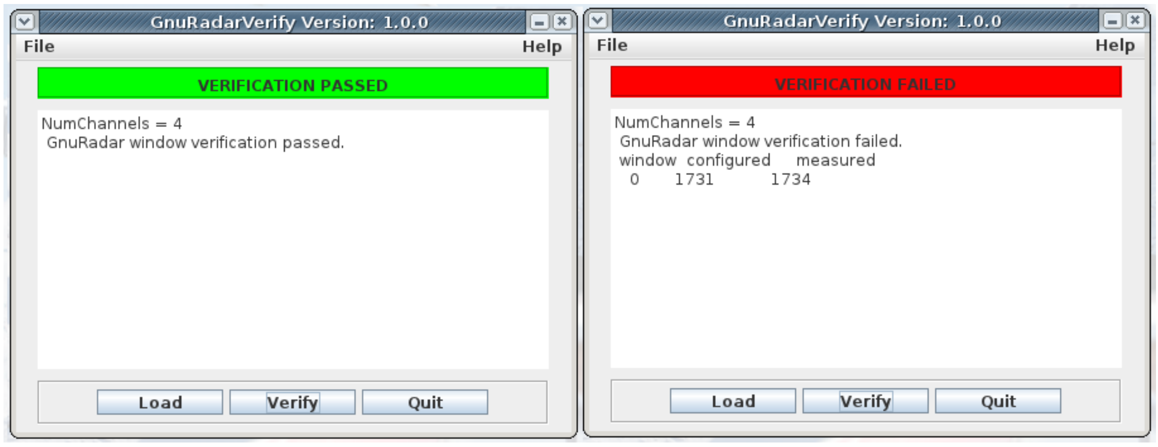
This configuration will be stored as a UCF file on the disk on a location as directed and can be reloaded any time in case editing is required.

4.2.2 Verification – gradar-verify:

As specified earlier, the sample window set between the Radar Controller and the USRP need to be in concurrence as both the softwares are independent in operation. The verification software is basically a method to make sure of the same.

When the gradar-verify program is executed, it actually takes some data in the background and verifies the number of samples taken per IPP between data tags and checks with the 'start' and 'stop' fields in the user specified configuration (UCF) file. Data tags are deliberately inserted values for every IPP at the start of a window to ensure synchronization.

In case of a mismatch, it reports that the 'Verification Failed' or 'Verification Passed' as shown in Figure 4-4. The operation can be carried on irrespective of a match or mismatch, but the verification serves to notify how far the configurations are off.



```
NumChannels = 4
 GnuRadar window verification passed.
```

```
NumChannels = 4
 GnuRadar window verification failed.
 window  configured     measured
   0      1731          1734
```

Figure 4-4. GnuRadar Verification User Interface Panel

4.2.3 Data Collection - gradar-run and gradar-run-server:

The GnuRadar software presently provides a remote server and local host configuration with networked ability such that the data collection can be started and stopped remotely by the user. From the networking point of view, it is important that the:

- /usr/local/gnuradar/gnuradar_server.xml has the broadcast_ip field set to the broadcast I.P address of the server
- /home/user/.gradarrc has the status_address field set to the broadcast IP address of the server
- /etc/services file has the port numbers assigned for GnuRadar as suggested in the readme.

When the gradar-run-server is executed, it accordingly displays the broadcast IP and the port being used. The gradar-run has to be loaded with the UCF file for the operation. The ‘Run’ and ‘Stop’ buttons can be used to begin and end the data collection.

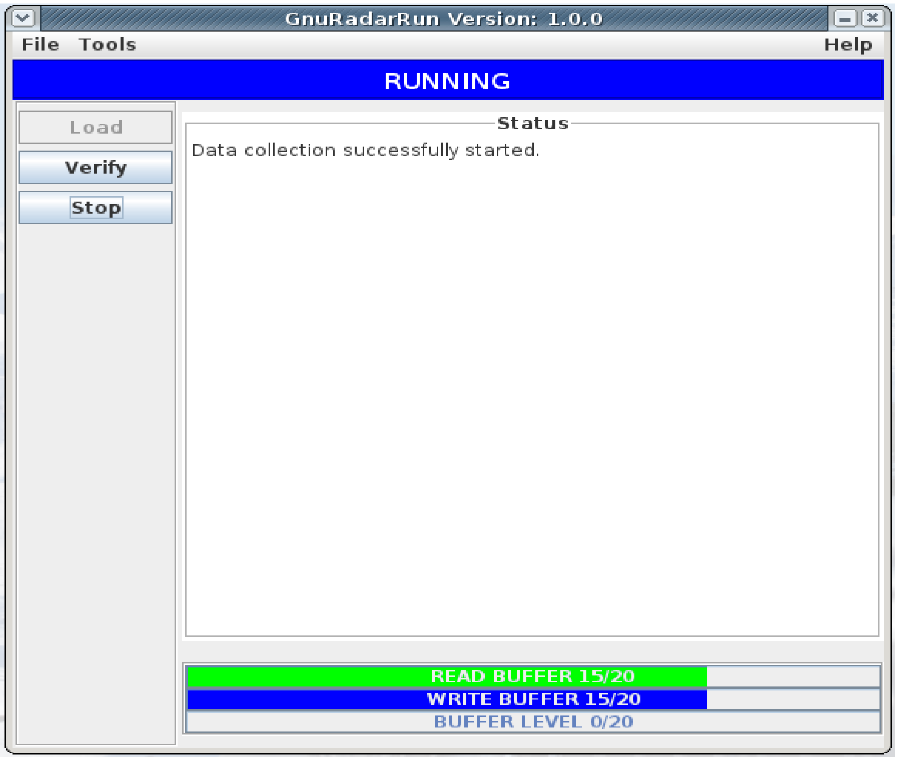
4.2.3.1 Key Concepts

1. **Producer Consumer problem**

The GnuRadarRun uses the Producer Consumer concept for data collection. The producer consumer problem (also known as the bounded-buffer problem) is a classical example of a multi-process synchronization problem [25]. In this process, two processes share a common, fixed size buffer as a queue. The producer does the work of filling up this buffer as the data arrives and the consumer process works to read this data piece by piece at the same time. The main issue in synchronizing this situation is to avoid the situation where the producer adds to the buffer when it is full (to prevent loss of incoming data) and also to avoid the consumer from reading from an empty buffer.

In the case of GnuRadar, the incoming raw data from the USRP is stored into the /dev/shm as a .buf file and the consumer process reads from this buffer to further process the data. The solution chosen here is that the producer goes to sleep if the buffer is full and the next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill this buffer again. Similarly, the consumer goes to sleep if it finds the buffer to be empty and the moment the producer puts data into the buffer, it wakes up the sleeping consumer.

On the server side during operation, a series of messages are displayed on the terminal which display the ongoing interactions with the producer and consumer. The GnuRadarRun GUI shows three bars which show the progress and the status of the data collection, one of which is an indication of 'dirty' i.e. unread buffers during the operation.

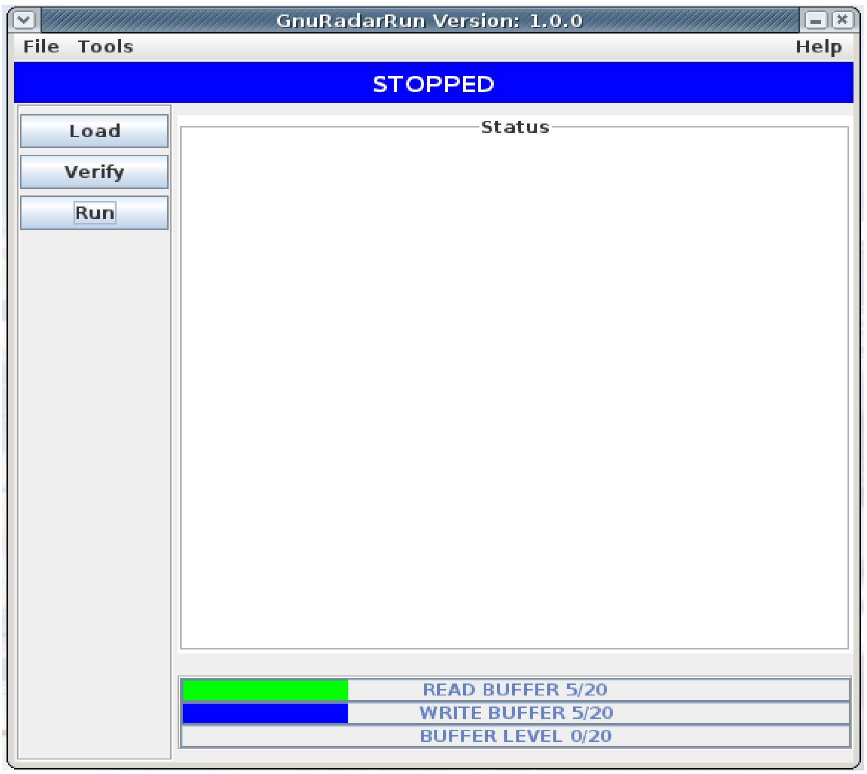


Data collection successfully started.

READ BUFFER 15/20
WRITE BUFFER 15/20
BUFFER LEVEL 0/20

Figure 4-5. GnuRadar Data Collection

It should be noted that the 'Stop' command is necessary for removing the buffers and cleanly quitting the data collection. This also ensures the accurate writing of the output files where the data is stored. Figure 4-6 is an illustration of issuing the Stop command.



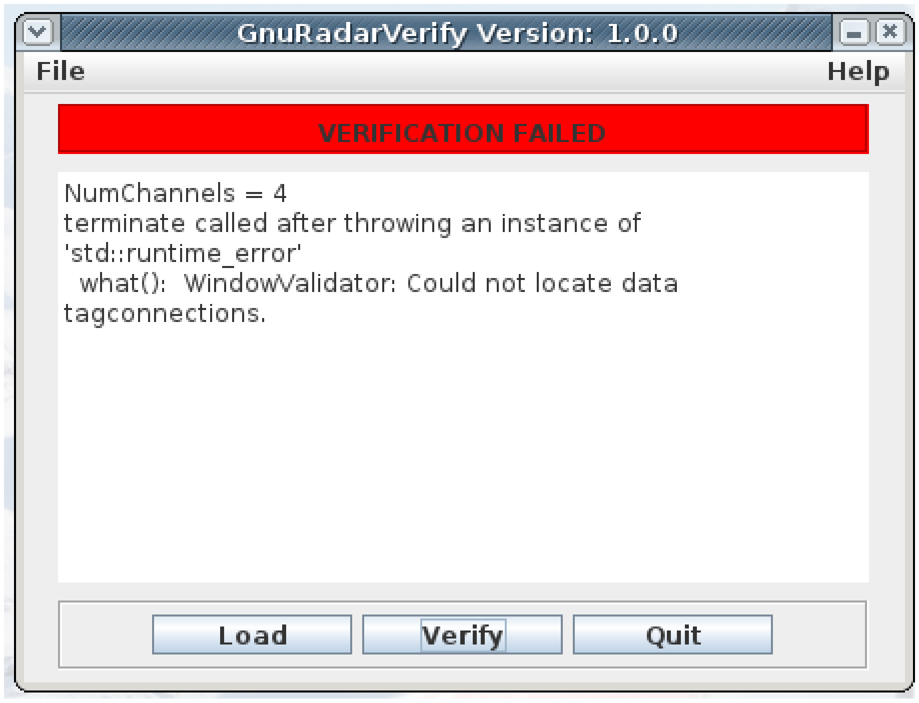
STOPPED

READ BUFFER 5/20
WRITE BUFFER 5/20
BUFFER LEVEL 0/20

Figure 4-6. Stopped Data Collection

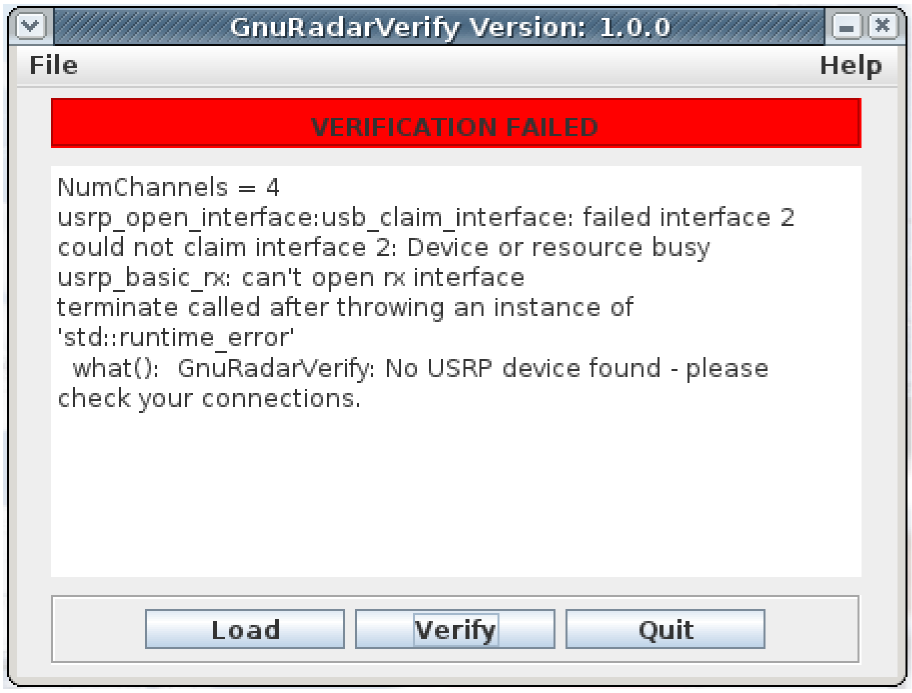
4.2.4 Troubleshooting:

The Verify utility is the best way to observe the health of the system. In case of any unexpected problems, the program produces the error messages accordingly. Some of the most common error messages are described here. Figure 4-7 shows the error message shown when the trigger signal is not found by the daughterboard as expected. It is good practice to always test the trigger on an oscilloscope before hooking it up on the daughterboard. Figure 4-8 shows the error message when the USRP is not detected by the software. Often, the way to overcome this error is to power cycle the USRP. It is also advisable to check or reconnect the USB cable in this situation.



```
NumChannels = 4
terminate called after throwing an instance of
'std::runtime_error'
  what():  WindowValidator: Could not locate data
tagconnections.
```

Figure 4-7. Trigger Error



```
NumChannels = 4
usrp_open_interface:usb_claim_interface: failed interface 2
could not claim interface 2: Device or resource busy
usrp_basic_rx: can't open rx interface
terminate called after throwing an instance of
'std::runtime_error'
  what():  GnuRadarVerify: No USRP device found - please
check your connections.
```

Figure 4-8. USRP detection error

4.3 Data Interpretation and Analysis

This section describes the tools and methods to perform data visualization and analysis.

4.3.1 The Real Time Plotter – gradar-plot:

The real time plotter is a diagnostic tool which helps to view the data as it comes into the system. The current plotter uses python plotting tools to read from the shared memory and produce real time IQ and Range-Time_Intensity (RTI) plots for four channels. It should be noted that the program is memory intensive and it will take a minimum of 8 GB memory to run smoothly.

The figures alongside show the IQ and RTI plot as seen with the real time plotter. The user has to merely 'connect' to the data stream from the 'File' menu and choose the type of plot desired for the number of channels. The data was fed into channels 1 and 2 of the USRP which is reflected in both these images.

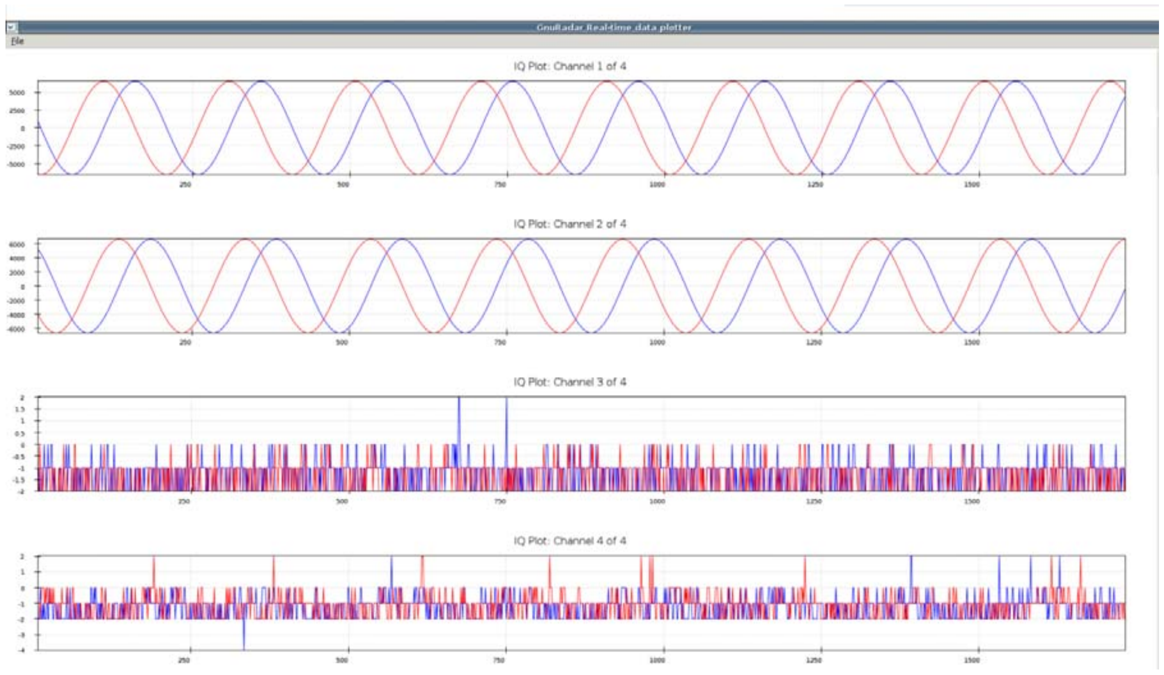


Figure 4-9. Real Time Plotter – IQ Plot of a 10.01 MHz signal

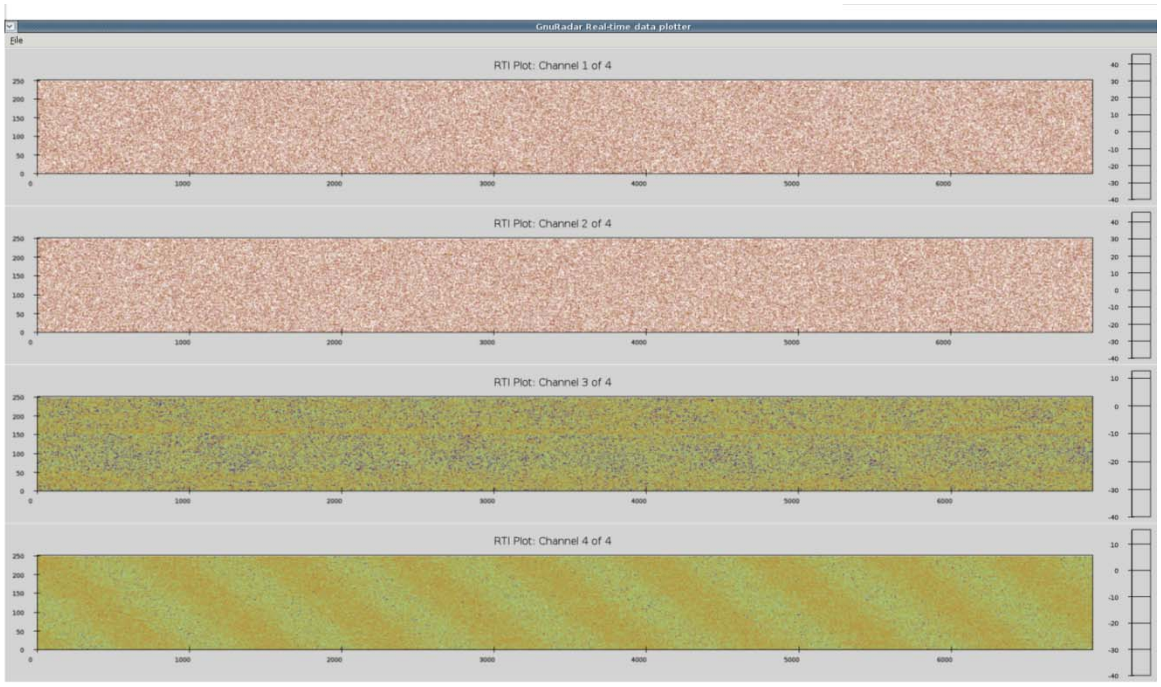


Figure 4-10. Real Time Plotter – RTI Plot

4.3.2 Post Processing:

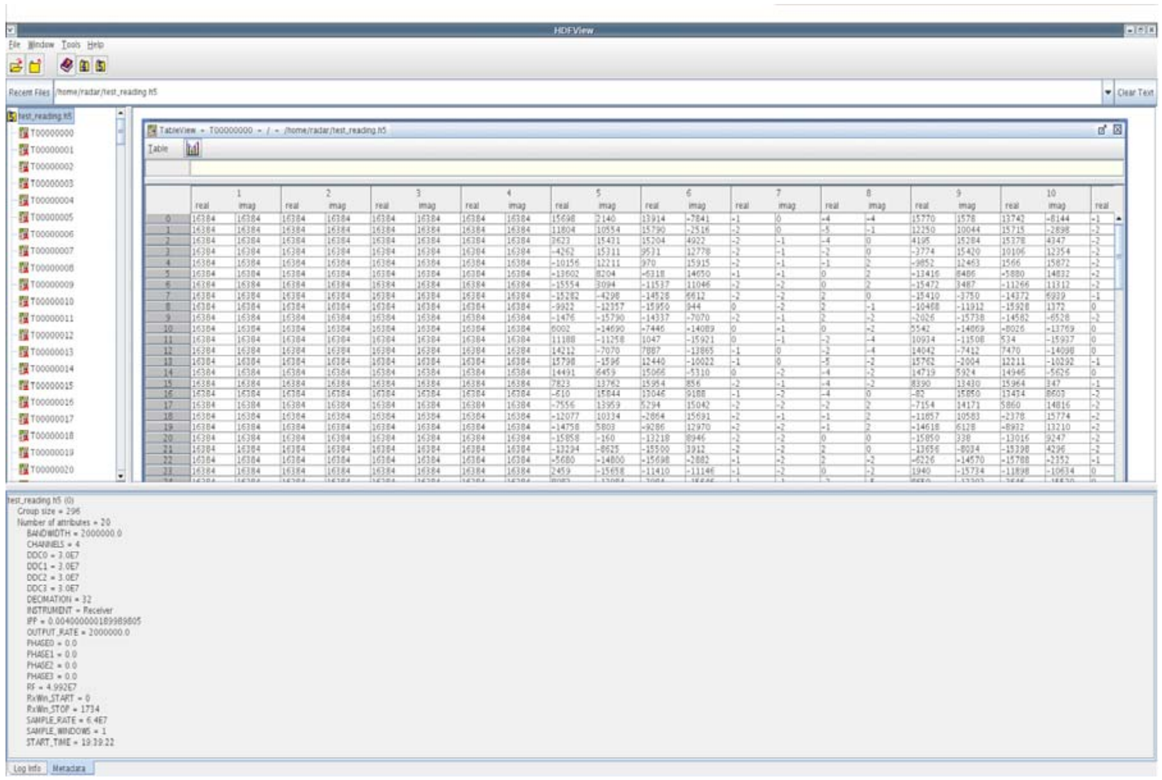
4.3.2.1 The Hierarchial Data Format (HDF5)

The raw binary data coming into the computer from the USRP is formatted into the Hierarchial Data Format (HDF5) and stored on disk. The HDF formats and libraries are designed to store and organize large amounts of numerical data. Originally developed at the National Center for Supercomputing Applications (NCSA), it is currently supported by the non-profit HDF Group, whose mission is to ensure continued development of HDF5 technologies, and the continued accessibility of data currently stored in HDF.

The HDF5 libraries and tools are available under a liberal, BSD-like license for general use and are supported by many commercial and non-commercial software platforms, including Java, MATLAB, IDL, and Python. The HDF5 file structure includes two major types of objects, i.e., Datasets, which are multidimensional arrays of a homogenous type, and Groups, which are container structures which can hold datasets and other groups [26]. This makes a file-system like structure which is hierarchical in nature. Metadata is stored in the form of user-defined, named attributes attached to groups and datasets.

HDFView is a Java based visual tool meant for browsing and editing the HDF5 files [27]. Using HDFView, it is possible to easily view a file hierarchy in a tree structure, create new file, add or delete groups and datasets, view and modify the content of a dataset, add, delete or modify attributes and view the metadata for each group or dataset.

Figure 4-11 shows the HDFView of a radar system collected data to illustrate the organization of the data collected in these files. As shown, the file test_reading.h5 displays a number of tables labeled T00000000 and so on. These tables are the two dimensional complex data collected per second. Each of these tables has rows as the IPP values and the columns as the ranges or height at which the data has been collected. The number of samples collected per IPP will be as indicated by the GnuRadarVerify program multiplied by the number of channels collecting data. The incoming complex data from each of the channels is interleaved; hence the data belonging to every channel is located at a stride of 4 columns from the first index. It should be noted that the first column for every channel of data will contain the data tags of value 16384 + 16384j. These tags help confirming the synchronization of the system and also help in processing the data. The data collection introduces a new file after every 2 GB of data collected to ensure reliability of the system because if the data collection is not cleanly quit, there are chances that the data will be corrupted due to incomplete HDF5 close operations. The metadata for the given HDF5 file or any given table displays the parameters under which the data was collected.



	1		2		3		4	
	real	imag	real	imag	real	imag	real	imag
0	16384	16384	16384	16384	16384	16384	16384	16384
1	16384	16384	16384	16384	16384	16384	16384	16384
2	16384	16384	16384	16384	16384	16384	16384	16384
3	16384	16384	16384	16384	16384	16384	16384	16384
4	16384	16384	16384	16384	16384	16384	16384	16384
5	16384	16384	16384	16384	16384	16384	16384	16384
6	16384	16384	16384	16384	16384	16384	16384	16384
7	16384	16384	16384	16384	16384	16384	16384	16384
8	16384	16384	16384	16384	16384	16384	16384	16384
9	16384	16384	16384	16384	16384	16384	16384	16384
10	16384	16384	16384	16384	16384	16384	16384	16384
11	16384	16384	16384	16384	16384	16384	16384	16384
12	16384	16384	16384	16384	16384	16384	16384	16384
13	16384	16384	16384	16384	16384	16384	16384	16384
14	16384	16384	16384	16384	16384	16384	16384	16384
15	16384	16384	16384	16384	16384	16384	16384	16384
16	16384	16384	16384	16384	16384	16384	16384	16384
17	16384	16384	16384	16384	16384	16384	16384	16384
18	16384	16384	16384	16384	16384	16384	16384	16384
19	16384	16384	16384	16384	16384	16384	16384	16384
20	16384	16384	16384	16384	16384	16384	16384	16384
21	16384	16384	16384	16384	16384	16384	16384	16384
22	16384	16384	16384	16384	16384	16384	16384	16384
23	16384	16384	16384	16384	16384	16384	16384	16384

	5		6		7		8		9		10		
	real	imag	real	imag	real	imag	real	imag	real	imag	real	imag	real
0	15698	2140	13914	-7841	-1	0	-4	-4	15770	1578	13742	-8144	-1
1	11804	10554	15790	-2516	-2	0	-5	-1	12250	10044	15715	-2898	-2
2	3623	15431	15204	4922	-2	-1	-4	0	4195	15284	15378	4347	-2
3	-4262	15311	9531	12778	-2	-1	-2	0	-3774	15420	10106	12354	-2
4	-10156	12211	970	15915	-2	-1	-1	2	-9852	12463	1566	15872	-2
5	-13602	8204	-6318	14650	-1	-1	0	2	-13416	8486	-5880	14832	-2
6	-15554	3094	-11537	11046	-2	-2	0	2	-15472	3487	-11266	11312	-2
7	-15282	-4298	-14528	6612	-2	-2	2	0	-15410	-3750	-14372	6939	-1
8	-9922	-12357	-15950	944	0	-2	2	-1	-10468	-11912	-15928	1372	0
9	-1476	-15790	-14337	-7070	-2	-1	2	-2	-2026	-15738	-14582	-6528	-2
10	6002	-14690	-7446	-14089	0	-1	0	-2	5542	-14869	-8026	-13769	0
11	11188	-11258	1047	-15921	0	-1	-2	-4	10934	-11508	534	-15937	0
12	14212	-7070	7887	-13865	-1	0	-2	-4	14042	-7412	7470	-14098	0
13	15798	-1596	12440	-10022	-1	0	-5	-2	15762	-2004	12211	-10292	-1
14	14491	6459	15066	-5310	0	-2	-4	-2	14719	5924	14946	-5626	0
15	7823	13762	15954	856	-2	-1	-4	-2	8390	13430	15964	347	-1
16	-610	15844	13046	9188	-1	-2	-4	0	-82	15850	13434	8603	-2
17	-7556	13959	5294	15042	-2	-2	-2	2	-7154	14171	5860	14816	-2
18	-12077	10334	-2864	15691	-2	-1	-1	2	-11857	10583	-2378	15774	-2
19	-14758	5803	-9286	12970	-2	-2	-1	2	-14618	6128	-8932	13210	-2
20	-15858	-160	-13218	8946	-2	-2	0	0	-15850	338	-13016	9247	-2
21	-13294	-8625	-15500	3912	-2	-2	2	0	-13656	-8034	-15398	4296	-2
22	-5680	-14800	-15698	-2882	-1	-2	2	-2	-6226	-14570	-15788	-2352	-1
23	2459	-15658	-11410	-11146	-1	-2	0	-2	1940	-15734	-11898	-10634	0

```
test_reading.h5 (0)
    Group size = 296
    Number of attributes = 20
        BANDWIDTH = 2000000.0
        CHANNELS = 4
        DDC0 = 3.0E7
        DDC1 = 3.0E7
        DDC2 = 3.0E7
        DDC3 = 3.0E7
        DECIMATION = 32
        INSTRUMENT = Receiver
        IPP = 0.004000000189989805
        OUTPUT_RATE = 2000000.0
        PHASE0 = 0.0
        PHASE1 = 0.0
        PHASE2 = 0.0
        PHASE3 = 0.0
        RF = 4.992E7
        RxWin_START = 0
        RxWin_STOP = 1734
        SAMPLE_RATE = 6.4E7
        SAMPLE_WINDOWS = 1
        START_TIME = 19:39:22
```

Figure 4-11. HDFView showing the data tags and metadata

4.3.2.2 RTI Post Processing

The figures shown alongside are the RTI plots for the data collected at the PSUSDR. We noticed some strong meteors from the data collection on November, 20 2011. The code used for processing is provided in the appendix and the images are shown below:

Since the transmitter pulse was 77 microseconds long, we can see that the meteor returns last approximately 11 km long on the RTI plots.

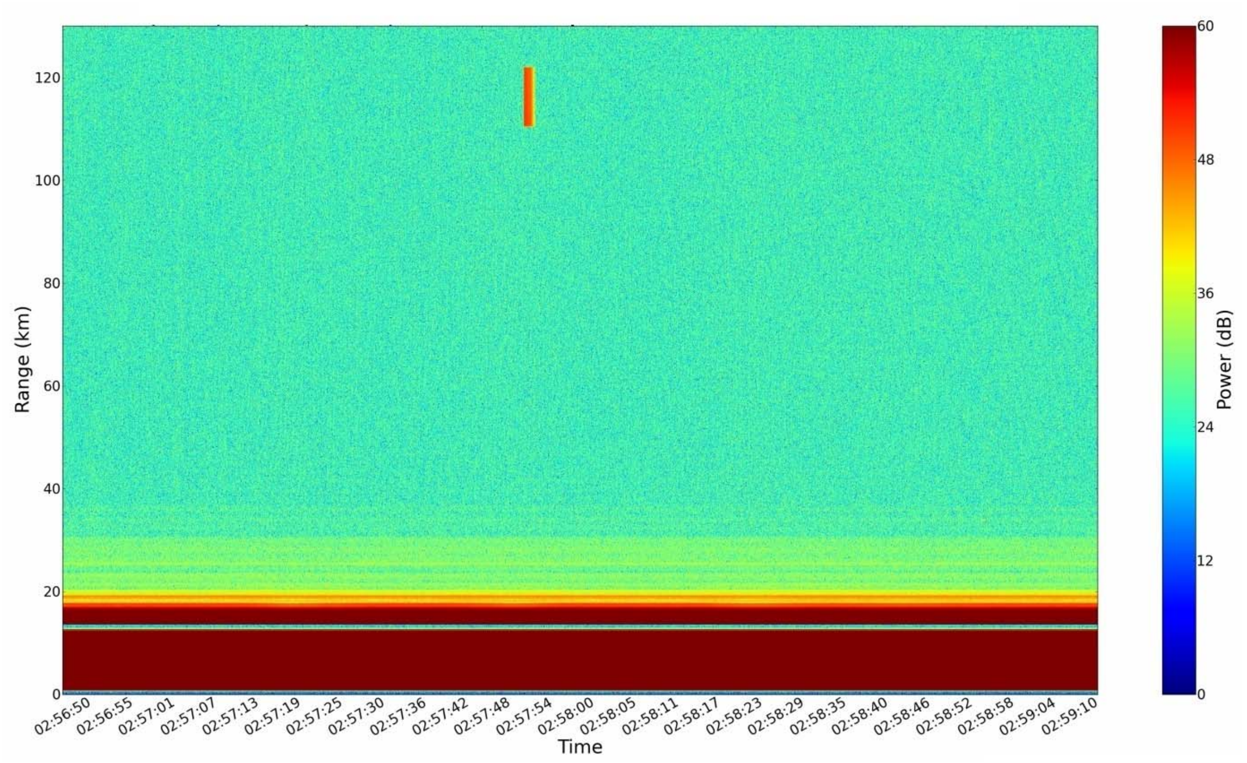


Figure 4-12. RTI Post Processing Plot – 11/20/2011 – 2:57 a.m. E.S.T.

Figure 4-12 shows strong meteor presence around 120 km on the morning of 20th November, 2011 with the PSUSDR setup. Figure 4-13 shows some strong and weak meteors later in the same morning around the 100 km altitude range.

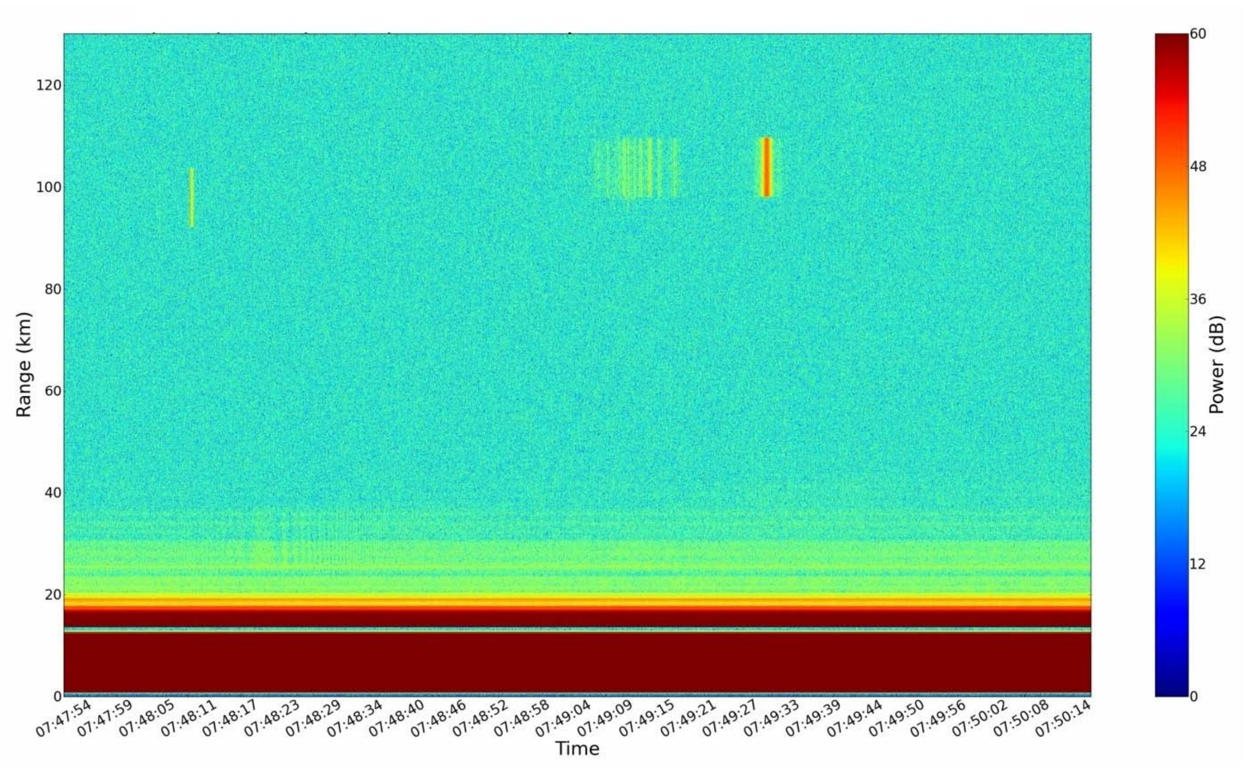


Figure 4-13. RTI Post Processing Plot – 11/20/2011 – 7:49 a.m. E.S.T.

4.3.3 Replaying Data – gradar-replay:

The GnuRadar distribution also provides a command line replay tool which helps the user to replay the data and view it on the real time plotter. The command line requires the arguments of the base file name to be replayed and the refresh rate desired.

Chapter 5

Interferometry

This section explains the principle of interferometry and details the algorithm applied along with the results obtained.

5.1 Principle of Interferometry

Intereferometric theory is found in the general fields of optics, astronomy and communications. The methods employed differ according to the applications these techniques are used for. The approach discussed here is phase interferometry, with an aim to have an accurate estimate of the Angle of Arrival (AOA) of the particular target signal. An interferometer generally refers to an array type antenna in which the large element spacing occurs. The tradeoff in this method is to have microwave circuitry which is complex and maintains a precise phase match over a wide frequency range under extreme environmental conditions.

In order to have a high accuracy in the order of 0.1 to 1 degrees, the baseline interferometers should have ambiguity resolving circuitry. The basic geometry is depicted in Figure 5-1 [28], whereby a plane wave seen arriving at an angle is received by one antenna earlier than the other due to the difference in path length.

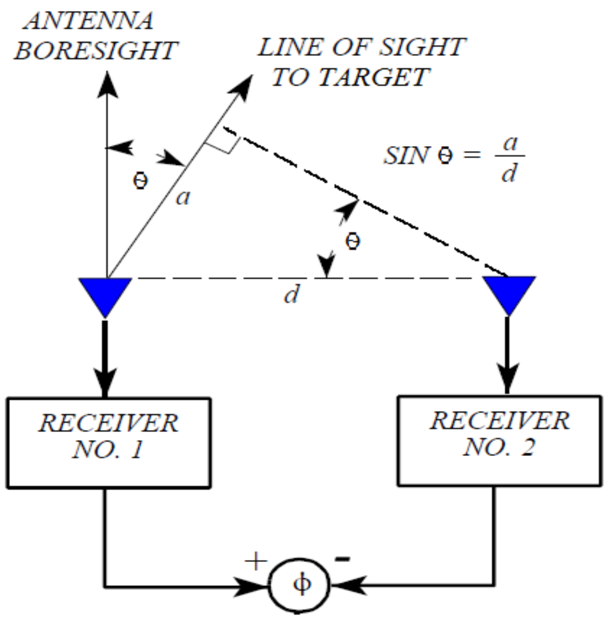


Figure 5-1. Phase Interferometer principle

The amplitude received at each of these antennas can be expressed as an amplitude which is given as $An = e^{j(\omega t+\Phi n)}$ where An is the voltage amplitude at the nth antenna. The time difference can be expressed as a phase difference Φ.

$$\Phi = \omega\Delta t = 2\pi a(f/c) = 2\pi(d*\sin\Theta)/\lambda \qquad (5.1)$$

where Φ is the phase difference

Θ is the angle of arrival

d is the antenna separation

λ is the wavelength.

The minimal distance between two antennas determines the maximal frequency of the input signal. The maximum distance between antennas for receiving the highest frequency component is λ/2 [29]. In a cross correlative measurement, the complex voltages at one antenna are multiplied by the conjugates of those at the other to obtain the angle of arrival.

Interferometer elements commonly use broad antenna beams with beam widths of the order of 90 degrees. This reduction of directivity limits system sensitivity due to the reduced antenna gain. It also makes the system open to interference signals from within the antenna's broad coverage. These signals often include multipath from strong signals which can affect the accuracy of the interferometer. In an interferometer, the locus of points that produce the same time or phase delay forms a cone.

Having said that the maximal frequency component can be achieved by keeping the maximum distance between the antennas as λ/2, when high accuracy is required, the separations employed are greater than λ/2. The increased separation sets up a multi-grating-lobe structure through the coverage angle which requires less SNR to achieve a specified accuracy.

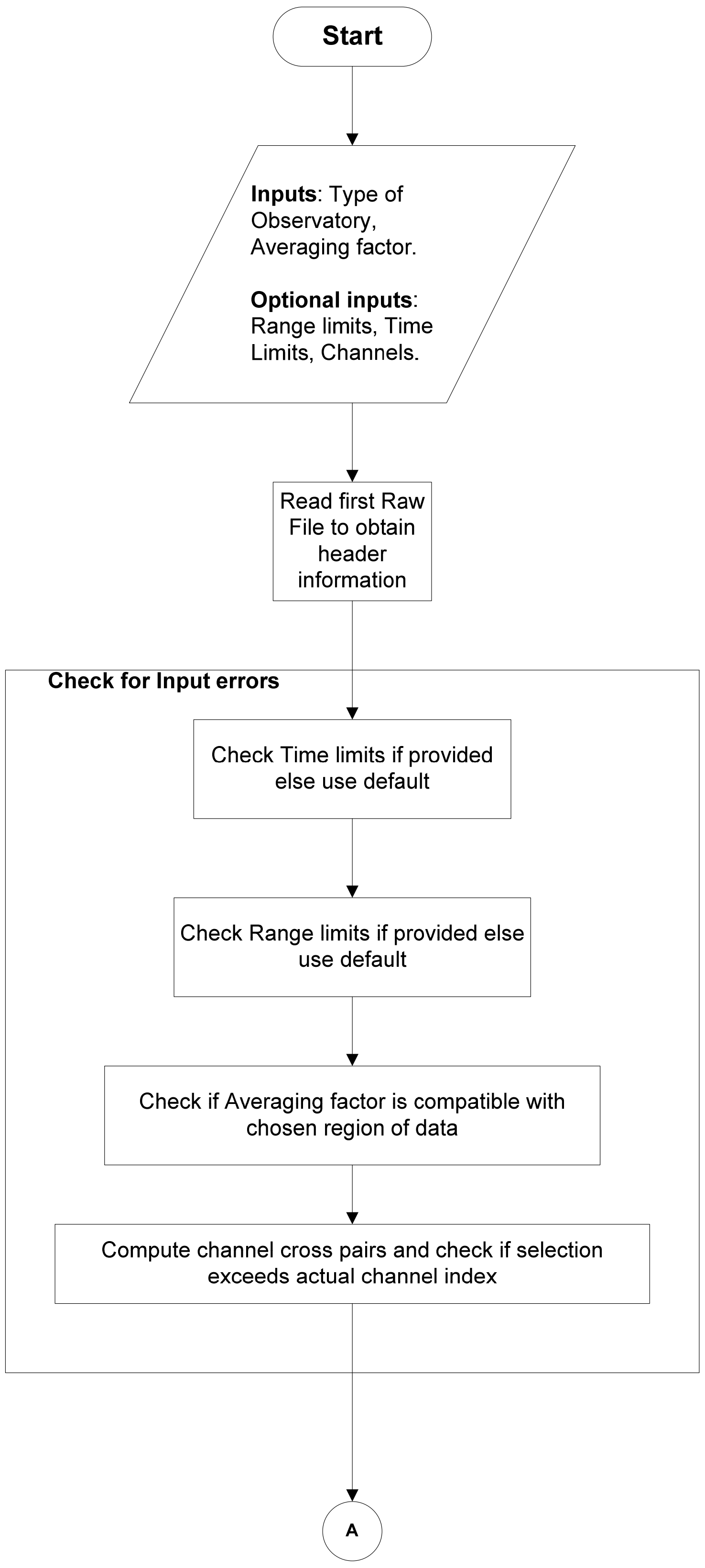
Sometimes, interferometers employ multiple antenna elements and are called multiple-baseline interferometers. The typical design has a receiver which consists of a reference antenna and a series of companion antennas. The spacing between the reference element and the first companion antenna is λ/2; other secondary elements are placed to form pairs separated by 1, 2, 4, and 8 wavelengths. The initial AOA is measured unambiguously by the shortest-spaced antenna pair. Every bit of the measurement from succeeding antenna pairs supplies a more accurate estimate of the AOA.

5.2 Interferometric Routine Algorithm

The algorithms applied in this work are the modified versions of those applied at the interferometer at JRO [1]. These observations of meteor-head echoes were made with the high-power large aperture Jicamarca radar in an interferometric mode. The large power-aperture of the system has helped record more than 3000 meteors per hour in the small volume subtended by the 1 degree antenna beam when the cluttering EEJ echoes were weak. The JRO radar has the lowest frequency of all the high-power large-aperture radars and is located under the magnetic equator (11.95 degrees S, 76.87 degrees W). It is able to measure the three dimensional vector of head echoes by operating in the interferometer mode.

In this case, the routine was modified to have a more generic, user friendly approach. The algorithm was also extended for the HDF5 file system obtained from the PSUSDR. Thus, the routine has the flexibility to obtain the AOA for any given observatory for any slice of range and time within a dataset. The choice of channels to be observed is also user defined.

One of the main characteristics of this routine, however, is the flexibility that is provided with the efficiency of the computations. The cross correlative math involved with the huge datasets often puts a huge load on the processing power of the machines. One way to make this more efficient is to perform the math across several operations and then coherently integrate them. This factor of averaging has also been made user configurable in this routine helping the user to tweak and optimize the operation to the limit desired. As this averaging factor increases, it also helps reduce the noise in the data and is particularly useful for observations of events in the ranges where EEJ is present.


Start
Inputs: Type of Observatory, Averaging factor.
Optional inputs: Range limits, Time Limits, Channels.
Read first Raw File to obtain header information
Check for Input errors
Check Time limits if provided else use default
Check Range limits if provided else use default
Check if Averaging factor is compatible with chosen region of data
Compute channel cross pairs and check if selection exceeds actual channel index
A

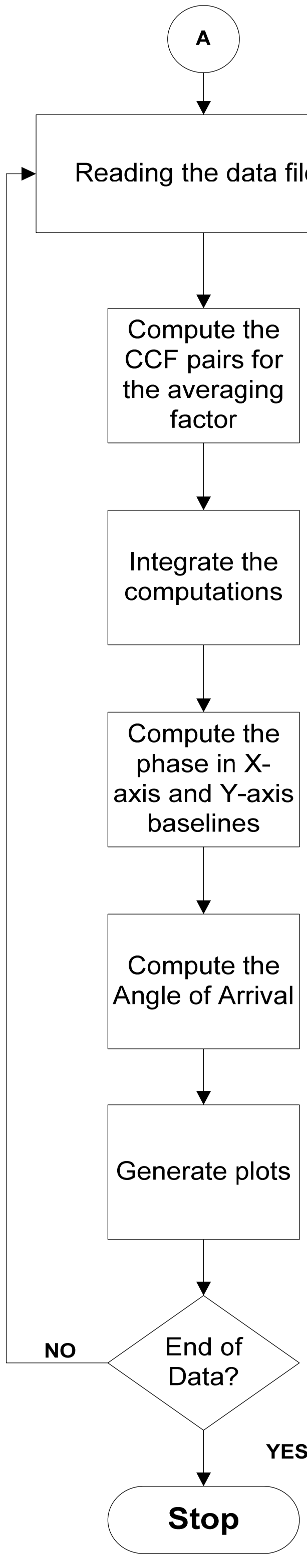

A
Reading the data files
Compute the CCF pairs for the averaging factor
Integrate the computations
Compute the phase in X-axis and Y-axis baselines
Compute the Angle of Arrival
Generate plots
End of Data?
NO
YES
Stop

Figure 5-2. Interferometric Routine Flowchart

5.3 Results

5.3.1 Generic Interferometric Routine:

The results obtained for the routines applied yielded accurate results. The primary approach was to obtain the magnitude and the phase plots with a generic routine which could produce the plots for all combinations of channels requested or present in the data.

Figures 5-3 and 5-4 show the verification of the algorithm for the data obtained from the JRO facility. Since the data is obtained from three antennas, we see the cross correlation obtained for all the three channels independently.

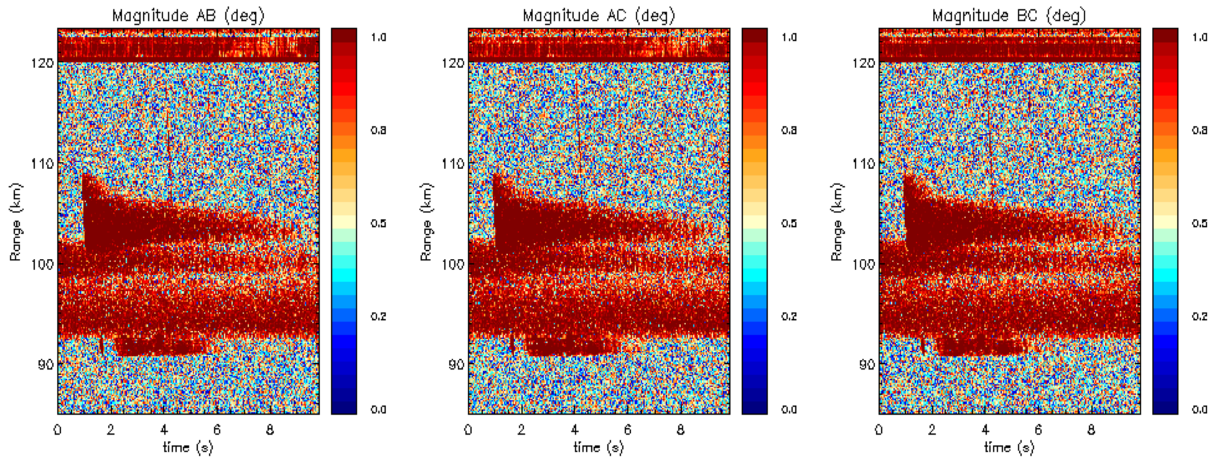


Figure 5-3. Magnitude plot for JRO data

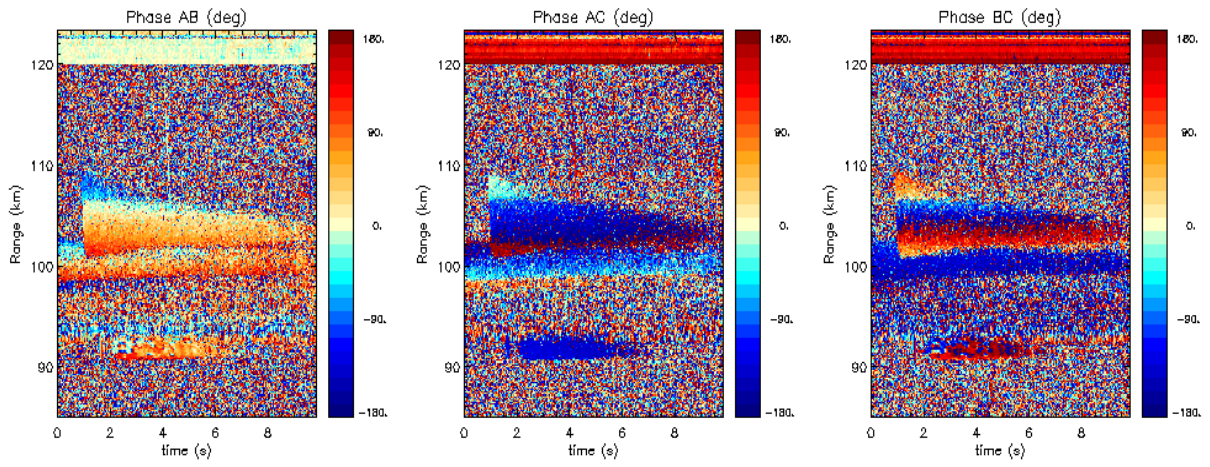


Figure 5-4. Phase plot for JRO data

When this scheme is applied to the four channel PSUSDR HDF5 system (Figure 5-5), we observe the cross correlation between all four channels. The raw data obtained here is the single channel data obtained from the PSUSDR on November 20, 2011. Although the plot obtained from this raw data does not provide much information about the interferometry, it assures the workability of the routine for the HDF5 system since it shows the traces of data with the subplots involving channel ‘A’ through which the data was collected.

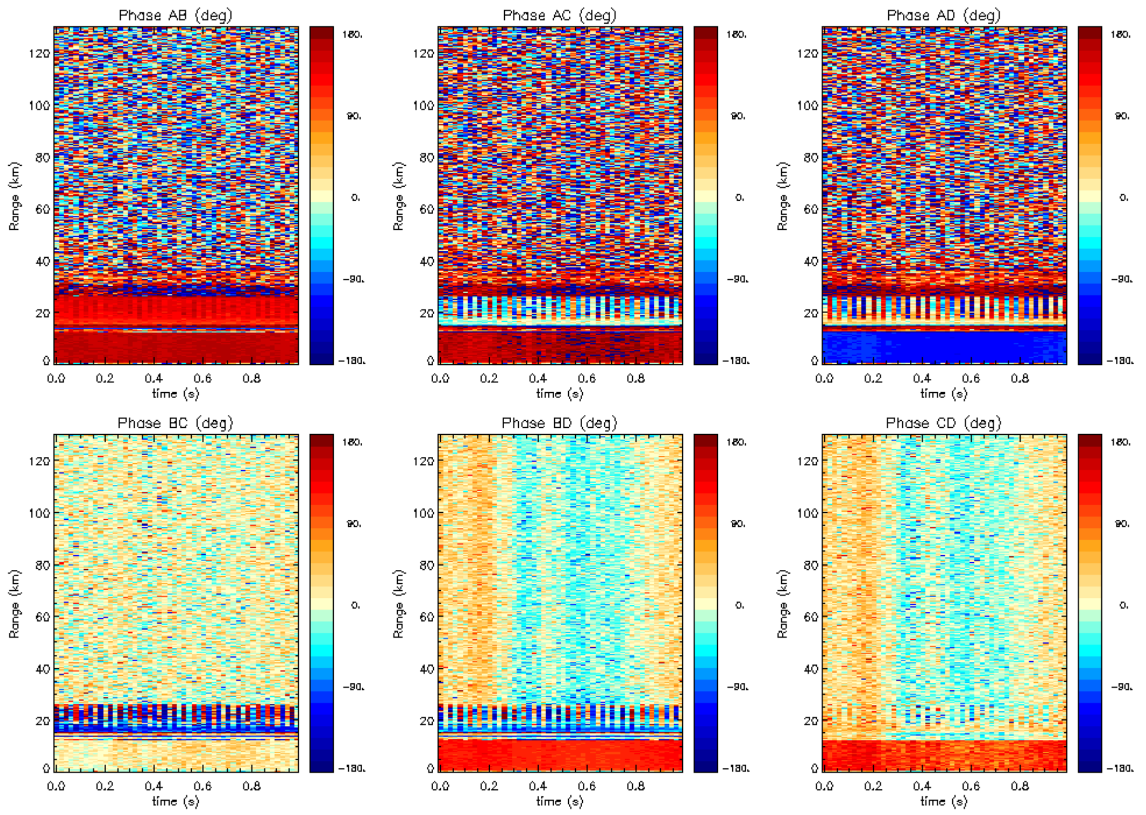


Figure 5-5. Phase plot for PSUSDR data

5.3.2 Angle of Arrival Estimation with baseline information:

At this point, the algorithms were modified to obtain the phase information for the baselines in the X and the Y direction and the exact angle of arrival was estimated. Keeping in mind that the ϴmax for the JRO configuration is 1 degree, the scaling of the phase was reduced to a range of -2 degrees to +2 degrees to obtain accurate information. Figure 5-6 shows the plot generated for the angle of arrival in the Y baseline.

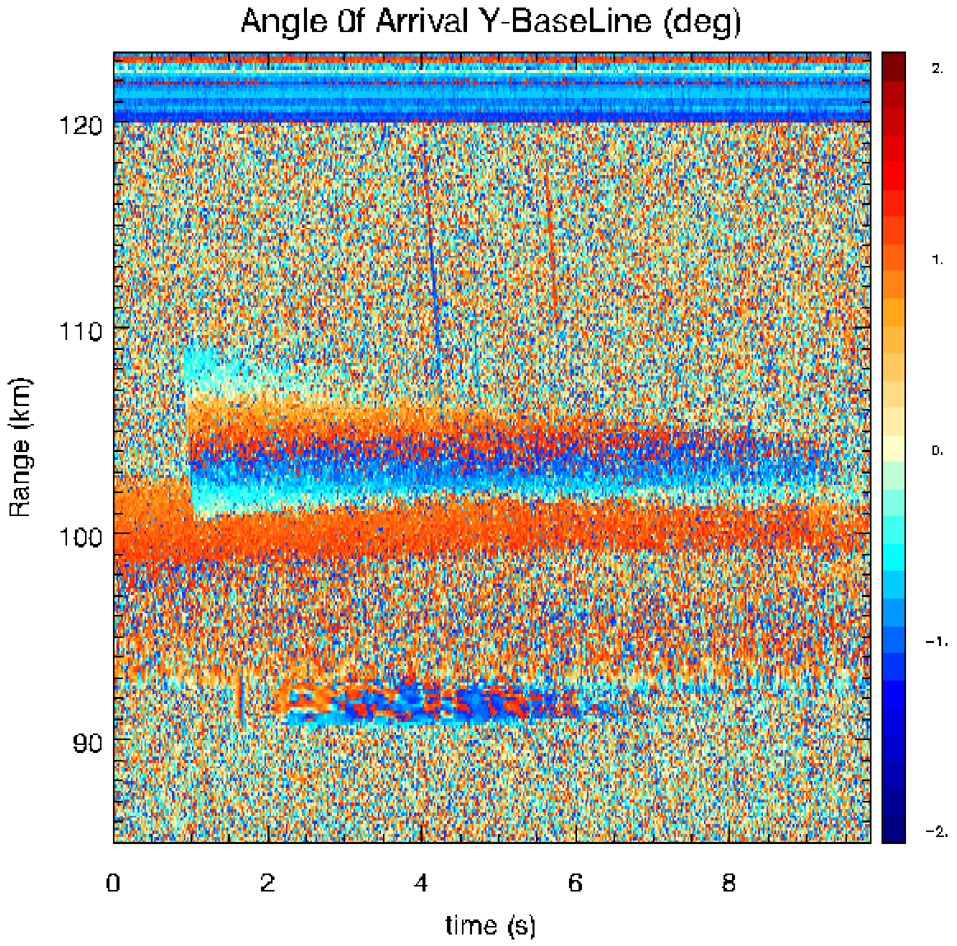


Figure 5-6. Plot for Y-Baseline at JRO (in degrees)

To further validate the efficacy of the algorithm, the data was collected through the RF Mixer chains shown in Figure 3-3 from signal sources emerging from the stable clock source Novatech 409B shown in Figure 3-8. The baseline configuration in this experiment was assumed to be the same as that at the JRO facility. The signals provided at both the channels were kept 0.5 degrees apart in phase and two sets of data were collected with a positive and a negative phase difference.

Figure 5-7 shows that the AOA was accurately estimated in the +0.5 degrees range for the first case. Since the signal source was constantly emitting at all ranges and IPPs, we see a solid color background indicating the accurate angle of arrival. Similarly, figure 5-8 shows the accurate estimation of -0.5 degrees for the second data set collected with the channels reversed.

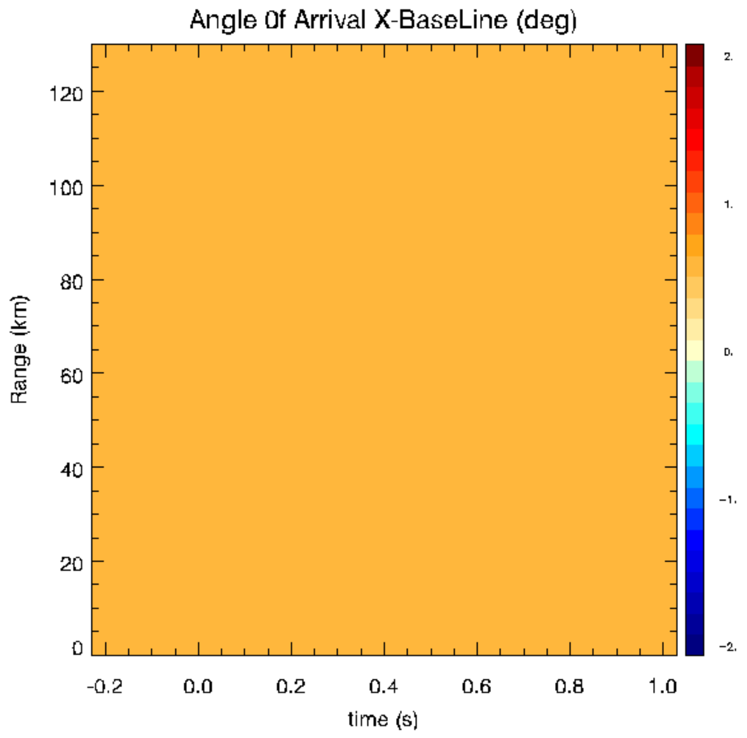


Figure 5-7. AOA plot for simulated X-Baseline obtained from USRP (+ 0.5 degrees)

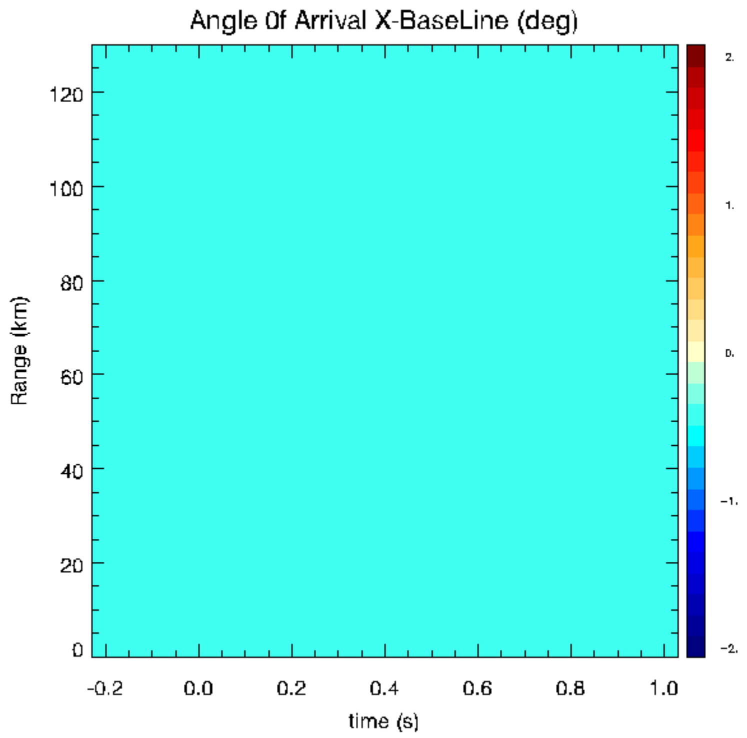
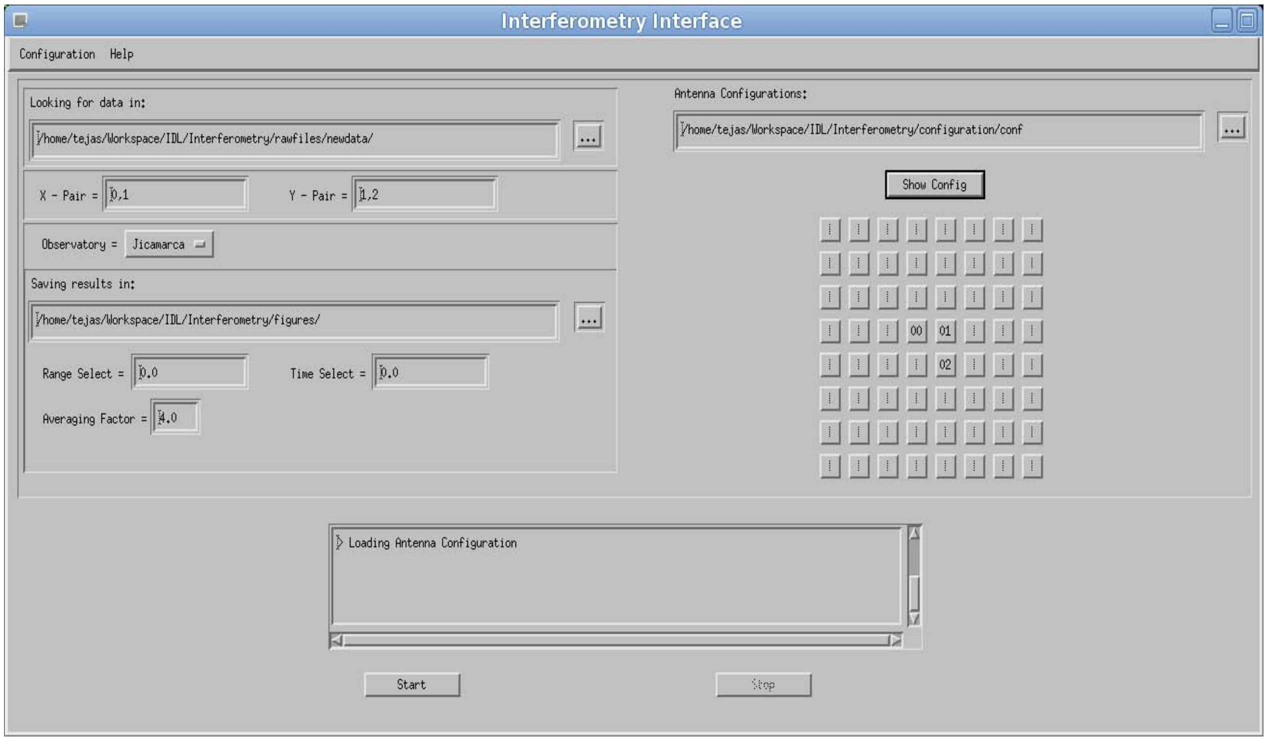


Figure 5-8. AOA plot for simulated X-Baseline obtained from USRP (- 0.5 degrees)

5.4 Graphical User Interface

A Graphical User Interface (GUI) was developed for performing the interferometric operations for the ease of the user. The Interferometry Interface V1.0 was made in IDL using event based widget programming. IDL GUI is constructed by combining widgets in a treelike hierarchy where each widget has one parent widget and zero or more child widgets which allow user interactions via simple graphical objects such as pushbuttons or sliders [30]. In an event driven system, the program creates an interface and then waits for messages or events to be sent to it from the window system. Events are generated in response to user manipulation, such as pressing a button or moving a slider. The program responds to events by carrying out the action or computation specified by the programmer, and then waiting for the next event. Actions occur in the order specified by the user at runtime, rather than in the order determined by the programmer.

As shown in figure 5-9, the Configuration tab provides the basic functionalities of saving and loading configurations. The configuration files are stored with an extension '.inter' on the disk which can be loaded at a later time using the 'Load Configuration' option.



Looking for data in: /home/tejas/Workspace/IDL/Interferometry/rawfiles/newdata/

X - Pair = 0,1
Y - Pair = 1,2

Observatory = Jicamarca

Saving results in: /home/tejas/Workspace/IDL/Interferometry/figures/

Range Select = 0.0
Time Select = 0.0

Averaging Factor = 4.0

Antenna Configurations: /home/tejas/Workspace/IDL/Interferometry/configuration/conf

```
Loading Antenna Configuration
```

Figure 5-9. Interferometry Interface GUI

The other inputs to the interface are as follows:

Observatory: The Observatory drop list helps the user to select the type of Observatory for performing the post processing. Currently the interface provides the options of 'Jicamarca' and 'Penn State'. In the background, the file extensions of these datasets are assumed to be '.r' and '.h5' respectively.

Path Entries: The long text boxes expect the path or the filename involved in that operation. The ‘Looking for data in’ option requires the path of the folder containing the raw data files. The ‘Saving results in’ option helps in choosing a folder path for saving the figures. The ‘Antenna Configurations’ option helps in picking the correct antenna configuration.

Antenna Configurations: The Antenna Configuration for a particular dataset can be stored in a plain text file which specifies the number of channels and the positions of the antennas. This file is to have prior knowledge about the antenna configurations so that the same could be shown graphically to the user during the operations. For example a three antenna configuration text file will contain the following:

```
ch = 3
rch = -73.5,73.5,73.5 ; 73.5,73.5,-73.5
```

The ‘ch’ should be equal to the number of channels and the ‘rch’ should provide the X positions and Y positions of the antennas separated by a colon. The ‘Show Config’ button then displays the configuration chosen by the user. The interface currently displays the 8 by 8 configuration present at Jicamarca and choosing the configuration highlights and numbers the antennas for the comfort of the user.

X-Pair and Y-Pair: The channel numbers or the antenna numbers for the X baseline and the Y baseline should be entered here. The displayed configuration by the ‘Show Config’ button should help the user in entering the right pair of antennas. Each text box expects the channel numbers to be separated by a comma as shown in the figure.

Range Select and Time Select: These inputs expect the subset of data that is preferred by the user for the operation. The range select helps to perform the interferometry at any height intervals. Similarly, the time select input can be fed with a time interval input. The default options observe the whole dataset and any erroneous input will revert to these default settings. The inputs here also require to be separated with a comma as shown.

Averaging Factor: This takes the averaging factor input which is the number of coherent integrations desired in the operation and also helps in noise reduction.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The RF Front End chain design for PFISR was modified to suit the requirements of PSUSDR and validated in the laboratory setup. The assumption here is that the signal coming into the mixer chain is from the output of the present RF Front End at PSUSDR and thus is relatively noise free. The IQ plots showed a perfect signal reproduced shown on the real time plotter. The spectrum plot further validated the chain by showing almost 80 dB difference between the signal and noise floor on the USRP spectrum plot. Particular care had to be taken such that the LO frequency was within the operating range of the amplifiers and also that the amplifiers were not saturating.

During the testing, it was observed that the frequency response of the filters is very critical while designing the RF chain. Often, while testing real radar data, the noise submerges the data and the filters with a high Q factor make a huge difference in the SNR measurements. Similarly, the use of wideband amplifiers in the front end chain should be avoided unless the receiver has a wide frequency range requirement. The RTI post processing plots for the data collected on November 20th, 2011 showed a few strong meteors. The transmitted pulse characteristics and the RF front end make a big impact on the number of the meteors observed.

The generalized interferometry algorithm was first tested for the JRO data in order to cross check with the existing codebase, to have some clarity on the subject. The extended HDF5 routine was also tested for the real radar data obtained from PSUSDR to observe that the plots provided sound information for the number of channels present in the system. With a view to obtain more accuracy in the routines, the phase of each baseline was computed and the exact angle of arrival was estimated. This approach was again tested and validated for the existing JRO data.

A final step was taken towards validating the whole system which encompassed the RF front end and the efficacy of the angle of arrival estimation routine. The results obtained from the signal generator sources fed through the chains into both the channels of the USRP provide accurate estimation of the angle of arrival of the signals. Overall, a sound proof of concept was presented for the purposes of introducing interferometry in the PFISR facility which has also paved the way for interferometric techniques at PSUSDR.

6.2 Future Work

The number of meteors observed in the data collected on November 20th, 2011 can be improved by further adding amplifiers to the existing RF Front end. The interferometric routines can be tested for actual data observed from two or more antennas at the radar site using the PSUSDR. Apart from interferometry, with a view to obtain more accurate and unambiguous measurements, pulse coding can be applied which can be achieved with the help of the radar controller and sending a coded pulse through the transmitter.

The idea of interferometry is fascinating and always suggests room for improvements. In keeping with the idea of a software defined radar, instead of employing correlators in hardware as practiced at the JRO facility, the cross correlation can also be done in software real time and the phase information can be directly displayed on the computer. An interesting tradeoff is whether to perform these complex operations in the FPGA of the USRP or purely in software. The USRP2, which is the next generation USRP, provides a much faster FPGA with the Xilinx Spartan3 family of FPGAs. Intuitively, the USRP2 suits this idea as an alternative to increasing the processing load in the computer.

Appendix

RTI Post Processing Code (Python)

```
#!/usr/bin/python

import numpy as np
import matplotlib as mp
mp.interactive(1)
import matplotlib.pyplot as pp
import h5py
import sys
import os
import pylab
import datetime
from datetime import datetime as dt


def ReadH5Files():

      # SET FILENAME HERE
      base_fileName = '/home/radar/RadarSiteData/radarsite’
      table_offset = 0

      os.system("ls -l "+base_fileName+"_* | wc -l > count")
      count_file  = open('count','r')
      count = count_file.readline()

      channel = 0

      # Parameters to plot
      xlabel = "Time"
```

```
ylabel = "Range (km)"
clabel = "Power (dB)"
aspect = 'auto'
vmin = 30
vmax = 60


for data in np.arange(int(count)):
      file_count = data + 1
      file_index = str(file_count).zfill(8)
      fileName = base_fileName + "_" + file_index + ".h5"


      fapl=h5py.h5p.create(h5py.h5p.FILE_ACCESS)
      file = h5py.File(fileName, mode='r', memb_size=1<<30)
      #print file.Group.

      if( data == 0 ):
            BW = float(file.attrs['BANDWIDTH'])
            Samples = int(file.attrs['RxWin_STOP'])
            extenty2 = Samples*1.5E8/(BW*1E3) # in KM
            ipp = float( file.attrs[ 'IPP' ] )


      iplots = 0
      iblock = 0
      label_count = 1
      NumberofTables = len(file.keys())

      nPlots = NumberofTables/2

      print "\n****Reading File "+fileName+"***\n"

      for iT in np.arange(NumberofTables):


            #table_number = iT + data*NumberofTables
```

```
                  Filename = "T"+str(iT + table_offset).zfill(8)
                  print "Reading Table "+Filename


                  #if not show data tag, channel = channel + 4

                  #read first data table
                  ds = file[Filename]
                  if iT == 0:
                        ArPower =
np.zeros((nPlots*ds.shape[0],ds.shape[1]/4 - channel)) #declare 2d
array of n-thousand by (sample size-offset)
                        #ArNoise =
np.zeros((nPlots*ds.shape[0],ds.shape[1]/4 - channel))
                        firstTableTimestampStr =
file[Filename].attrs['TIME']
                        print firstTableTimestampStr
                        firstTableTimestamp = dt.strptime(
firstTableTimestampStr, '%H:%M:%S' )
                        line = firstTableTimestampStr
                        line2 = line.split(":")
                        line3 = line2[0]
                        line4 = line2[1].split(":")
                        line5 = line4[0]
                        extentx1 = int(line3+line5)
                        print extentx1



            #Getting real and imaginary data.
                  dReal = ds['real'][:,channel::4]*1.0
                  dImag = ds['imag'][:,channel::4]*1.0

                  # Computing Power
                  Power = (dReal)**2. + (dImag)**2.

                  # Accumulating Power
```

```
      ArPower[iplots*ds.shape[0]:((iplots+1)*ds.shape[0]),:] =
Power[:,:]

                  iplots = iplots +  1
                  if iplots==nPlots:

                        tempo = np.sort( ArPower.flatten() )
                        noise = np.mean( tempo[0:tempo.size*.5] )
                        #ArNoise.append( noise )
                        print "Noise Level: ", noise

                        SNR = ( ArPower - noise ) / noise
                        dB_SNR = 10. * np.log10( SNR )

                        # pre plotting data handlings
                        #dB = np.log10( ArPower ) * 10
                        #db_trans = np.transpose( dB )
                        db_trans = np.transpose( dB_SNR )

                        # Plotting related stuff
                        fig = pp.figure( num=1, figsize=(40,20) )
                        pp.clf()
                        ax = fig.add_subplot(111)

                        #dims = dB.shape
                        dims = dB_SNR.shape
                        extent = ( 0, dims[0], 0, extenty2)#dims[1] )
                        im = pp.imshow( db_trans, origin="lower",
vmin=vmin, vmax=vmax, extent=extent, aspect=aspect )

                        cticks = np.int32( vmin + np.arange(6) * ( vmax
- vmin ) / 5. )

                        # calculate positions of x-tick marks
                        numTicks = 30
                                                # number of tickmarks
                        xtickLabels = [ firstTableTimestampStr ]
                              # vector of x tick labels
                        xtickLength = ipp * float( nPlots *
NumberofTables / numTicks )         # length of tick period (units)

                        newTimeStamp = firstTableTimestamp
                                    # datetime temporary object

                        for i in range ( 1, numTicks ):
                                          # iterate through each
tickmark
                              newTimeStamp += datetime.timedelta(
seconds=xtickLength )         # increment datetime object
                              xtickLabels.append(
newTimeStamp.strftime( '%H:%M:%S' ) )           # convert to string,
append to vector

                        ax.xaxis.set_major_locator(
mp.ticker.MaxNLocator( numTicks ) )

                        # calculate positions of y-tick marks
                        yticks = np.int32( np.array( ax.get_yticks() )
)

                        cb = pp.colorbar( im, ticks=cticks )

                        cticks = np.int32( np.array( vmin +
np.arange(6)*(vmax - vmin)/5. ) )
                        cb.ax.set_yticklabels( cticks, size=24 )
                        ax.set_xticklabels( xtickLabels, size=24,
rotation=30 )
                        ax.set_yticklabels( yticks, size=24 )

                        cb.ax.set_ylabel( clabel, size = 32 )
                        ax.set_xlabel( xlabel, size=32 )
                        ax.set_ylabel( ylabel, size=32 )
                        pp.title( fileName + " Part " + str(
label_count ), size=48 )

                        # save the file
                        NFile = "rti_usrp" + str( data * (
NumberofTables / nPlots ) + iblock ).zfill(6) + ".png"
                        print "\n Plotting "+NFile+" ...\n"

                        pp.savefig( NFile )

                        # data handling and cleanup
                        pylab.close()

                        iblock = iblock + 1
                        label_count = label_count + 1
                        iplots = 0

            table_offset = table_offset + NumberofTables

ReadH5Files()
pp.show()
```

Interferometric Routines (IDL)

```
      ;+
      ; NAME geometry
      ;   INTERF_MOD
      ;
      ; PURPOSE
      ;   Generic Interferometry technique to compute AOA
      ;
      ;
      ;
      ;WORKING
      ;   The program takes information about type of Observatory and
number of coherent integrations for efficient
      ;   computations. Time, Range, Channel information can be
provided but is not necessary. The program also needs data path,
      ;   file extension information and path to save the figures
      ;
      ;   After finding the data files, the program first gets the
header information required to generate/confirm error free Time,
      ;   Range and Channel information. It also computes the ccf pairs
automatically from this information.
      ;
      ;   The raw data files are read and the Magnitude,Phase and AOA
is accordingly computed, plotted and saved.
      ;

      PRO INTERF_MOD,NEvent=NEvent, NumAver=NumAver, LProf=LProf,
LRange=LRange, Chan_Index= Chan_Index



        IF N_ELEMENTS(NEvent) EQ 0 THEN BEGIN
          PRINT,"NEvent is not defined. Skip ..."
          RETURN
        ENDIF

        ; User definition of working path, filename/format and type of
data expected
        ;INTERFEROMETRIC_ROUTINE, NEVent=0, NumAver=4,
LProf=[0.0,9.83],LRange=[85.05,123.3],Chan_Index= [0,1,2]
        ;INTERFEROMETRIC_ROUTINE, NEVent=1, NumAver=5,
LProf=[0.0,0.996],LRange=[0.0,129.975],Chan_Index= [0,1,2,3]

        CASE NEvent OF
          00: BEGIN
                ; Defining working path. [from user]
                dpath = '/path/to/data '
                ; Defining the generic name of the files. [from user]
                dfile = 'D*.r'
                ; Observatory
                dtype = 0
                ; Defining path to save figures
                GPath = "/path/to/save"
                xpair = [0,1]  ; Pair of channels defining x-axis
                ypair = [1,2]  ; Pair of channels defining y-axis
                lambda = 6     ; Transmitted wavelength (6m)
                rch = [[-147/2.,147/2.,147/2.],[147/2.,147/2.,-147/2.]]
; Position of the recievers (in meters)

                PowTh = 55     ; Power Threshold (in dB)

              END
          01: BEGIN
               ; Defining working path. [from user]
               dpath = '/path/to/data '; Defining the generic name of
the files. [from user]
               dfile = '*h5'  ; Observatory
               dtype = 1      ; Defining path to save figures
               GPath = "/path/to/save"

               xpair = [0,1]  ; Pair of channels defining x-axis
               ypair = [1,2]  ; Pair of channels defining y-axis
               lambda = 10    ; 30 MHz signal
               rch =  [[-147/2.,147/2.,147/2.],[147/2.,147/2.,-147/2.]]
; Position of the recievers (in meters)

               PowTh = 55     ; Power Threshold (in dB)
              END
        ENDCASE

        ; Checking for input errors
        IF N_ELEMENTS(dpath) EQ 0 THEN BEGIN
          PRINT,"NEvent is not defined. Skip ..."
          RETURN
        ENDIF

        IF N_ELEMENTS(dfile) EQ 0 THEN BEGIN
          PRINT,"NEvent is not defined. Skip ..."
          RETURN
        ENDIF

        IF N_ELEMENTS(dtype) EQ 0 THEN BEGIN
          PRINT,"NEvent is not defined. Skip ..."
          RETURN
        ENDIF

        CASE dtype OF
          0: READ_PROC = "READ_JRO"
          1: READ_PROC = "READ_PSU"
          2: READ_PROC = "Other"
        ENDCASE


        ; Getting the first raw file.
        Files = FILE_SEARCH(dpath+dfile,count=NFiles)
        IF NFiles EQ 0 THEN BEGIN
          PRINT,'READ_RAW_FILES: Couldn''t find file ', dpath+dfile
          RETURN
        ENDIF
        Files = Files[SORT(Files)]

        ; Reading radar parameters
        Success =
CALL_FUNCTION(READ_PROC,Filename=Files[0],cur_ptr=0,header=header)


        ; Checking input errors in the Limits given by user
         IF N_ELEMENTS(LProf) LT 2 THEN BEGIN
          IF N_ELEMENTS(LProf) EQ 1 THEN PRINT, "Error: Please enter
both values"
          LProf = [MIN(header.time),MAX(header.time)]
         ENDIF ELSE BEGIN
          LProf = [LProf[0]>MIN(header.time),LProf[1]<MAX(header.time)]
         ENDELSE

         IF N_ELEMENTS(LRange) LT 2 THEN BEGIN
          IF N_ELEMENTS(LRange) EQ 1 THEN PRINT, "Error: Please enter
both values"
          LRange = [MIN(header.range),MAX(header.range)]
         ENDIF ELSE BEGIN
          LRange =
[LRange[0]>MIN(header.range),LRange[1]<MAX(header.range)]
         ENDELSE

          ; Compute the indexes for given limits
         IProf =  WHERE((header.time GE LProf[0]) and (header.time LE
LProf[1]))
         IRange = WHERE((header.range GE LRange[0]) and (header.range
LE LRange[1]))

         ;If selected LProf is not divisible by NumAver, quit. This is
because the first dimension is sliced
         ; into the number of coherent integrations
         IF N_ELEMENTS(IProf) MOD NumAver NE 0 THEN BEGIN
          PRINT,'Time/Profile Selection is not a multiple of Number of
Integrations  ', dpath+dfile
          PRINT, 'The current size of profile', N_ELEMENTS(IProf)
          RETURN
         ENDIF

         ; Computing possible cross pairs based on channels selected
         ;  cross_pairs=[[0,0,1],[1,2,2]] case of 3 chanels
         IF (KEYWORD_SET(Chan_Index) EQ 0)  THEN BEGIN
          cross_pairs = TRANSPOSE(pgcomb(header.Nchans,2))
          Chan_Index=indgen(header.Nchans)
         ENDIF ELSE BEGIN; this is to emulate the output of pgcomb for
the channels selected
          cross_pairs = TRANSPOSE(pgcomb(N_ELEMENTS(Chan_Index),2))
          temp_pairs = cross_pairs
          FOR k=0,N_ELEMENTS(Chan_Index)-1 DO BEGIN
           temp_pairs[(WHERE(cross_pairs EQ k))[*]] = Chan_Index[k]
          ENDFOR
          cross_pairs = temp_pairs
         ENDELSE

         ; This is to check if channels selected exceed dataset channel
values
         chan_check = WHERE(cross_pairs GT
MAX(TRANSPOSE(pgcomb(header.Nchans,2))))
        IF N_ELEMENTS(chan_check) GT 1 THEN BEGIN
          PRINT,'Selected Chanels exceed dataset channel values ',
dpath+dfile
          RETURN
        ENDIF

         IF (N_ELEMENTS(chan_check) EQ 1 AND chan_check NE -1) THEN
BEGIN
           PRINT,'Selected Chanels exceed dataset channel values ',
dpath+dfile
          RETURN
        ENDIF


         channels=Chan_Index
         chlabels = ['A','B','C','D','E','F','G','H']; labels for
figures


       ; Begin reading raw data
        iFile = 0
        block_count = 0

        REPEAT BEGIN
          FileName = Files[iFile]

          cur_ptr = 0
          REPEAT BEGIN
            Success =
CALL_FUNCTION(READ_PROC,Filename=Filename,cur_ptr=cur_ptr,volt=volt,hea
der=header)

            IF Success NE -1 THEN BEGIN

              block_count = block_count + 1
              ; Selection of data before plotting (IPPs, Range,
Channels)

volt=volt[MIN(IProf):MAX(IProf),MIN(IRange):MAX(IRange),Chan_Index]

              ;Compute CCF Pairs Begins /ccf

              dim=SIZE(VOLT,/DIM)
              volt=REFORM(volt,NumAver,dim[0]/NumAver,dim[1],dim[2])
              num_cross=N_ELEMENTS(cross_pairs[*,0])
              cross_0=INTARR(num_cross)
              cross_1=INTARR(num_cross)
              FOR icr=0,num_cross-1 DO BEGIN
```

```
                    cross_0[icr]=(WHERE(channels EQ cross_pairs[icr,0]))[0]
                    cross_1[icr]=(WHERE(channels EQ cross_pairs[icr,1]))[0]
                  ENDFOR


                  ; Sum of each component in first dimension, which is
integrating the data according to NumAver
                  ccf =
TOTAL(volt[*,*,*,cross_0]*CONJ(volt[*,*,*,cross_1]),1);avg the
correlations or avg 4 pulses

                  ;PHASE = ATAN(CCF,/PHASE)
                  ;P1 = TOTAL(ABS(volt[*,*,*,cross_0])^2,1)
                  ;P2 = TOTAL(ABS(volt[*,*,*,cross_1])^2,1)

                  ;MAG = ABS(CCF/SQRT(P1*P2))

                  mRange = Header.range[MIN(IRange):MAX(IRange)]
                  mTime = FINDGEN(dim[0]/NumAver)*(Header.Time[1]-
Header.Time[0])*NumAver
                  print,size(mTime)

                  ;Computing PHASE in the x-axis baseline
                  pair = WHERE(cross_pairs[*,0] EQ xpair[0] AND
cross_pairs[*,1] EQ xpair[1],cpair)
                  xPha = ATAN(ccf[*,*,pair[0]],/PHASE)

                  ;Computing PHASE in the y-axis baseline
                  pair = WHERE(cross_pairs[*,0] EQ ypair[0] AND
cross_pairs[*,1] EQ ypair[1],cpair)
                  yPha = ATAN(ccf[*,*,pair[0]],/PHASE)

                  ; Computing ambiguos angle of arrival
                  xdist = rch[WHERE(channels EQ xpair[0]),0]-
rch[WHERE(channels EQ xpair[1]),0]
                  ydist = rch[WHERE(channels EQ ypair[0]),1]-
rch[WHERE(channels EQ ypair[1]),1]
                  print, "Distance between receivers:", xdist, ydist ;
distance between any 2 receivers
                  radius = ASIN(lambda/(ABS(xdist)+ABS(ydist)))*!radeg
                  xAOA = ASIN(-lambda*xPha/(2.*!pi*xdist[0]))*180d/!pi
                  yAOA = ASIN(-lambda*yPha/(2.*!pi*ydist[0]))*180d/!pi

                  Pow = 10*ALOG10(MEAN(ABS(ccf)));,DIM=3))


                  ; Taking events greater than a threshold of intensity
                   noval = WHERE(Pow LT PowTh,cnoval)
                   IF cnoval GT 0 THEN BEGIN
                    xAOA[noval] = !VALUES.F_NAN
                    yAOA[noval] = !VALUES.F_NAN
                  ENDIF
```

```
        ; Plotting Cross Correlation function Phase.
                Graphic = 1
                IF Graphic NE 0 THEN BEGIN
                  GFile = "xAoA_"+STRING(block_count,FORMAT = '(I5.5)')
                  GSave = 1

                  ;xPlots = 3
                  ;yPlots = CEIL(NPairs/3.)
                  xPlots = 1
                  yPlots = 1

                  ;xsize = 990
                  ;ysize = 350*(yPlots<3)
                  xsize = 950
                  ysize = 950


CTRL_PLOT,wid=1,xsize=xsize,ysize=ysize,xpos=10,path=GPath,png=1,show=1
,title="Interferometry_Phase",$
                  overplot=0,file=GFile

                  !P.MULTI = [0,xPlots,yPlots]
                  CONTROL_PLOT,close=0

                  maxcol = 79
                  mincol = 50
                  maxlev = 2.
                  minlev = -2
                  makeformat = '(F6.0)'
                  novalid_col = !P.COLOR

                  title = 'Angle 0f Arrival X-BaseLine (deg)'
                  xtitle = 'time (s)'

                  charsize = 2

                  xrange = [MIN(mTime),MAX(mTime)]
                  yrange = [MIN(mRange),MAX(mRange)]

CONT_IMAGE,xAoA,mTime,mRange,TOP=maxcol,BOTTOM=mincol,MINA=minlev,XSTYL
E=1,MAXA=maxlev,$

XRANGE=xrange,YRANGE=yrange,YSTYLE=1,TITLE=title,XTITLE=xtitle,XTICKN=x
tickn,$
                    YTITLE='Range (km)',SUBTITLE=subtitle,
NOVALID_COL=novalid_col,XMARGIN=[8,12],$
                    CHARSIZE=charsize

                  nlev = 29
                  col = FINDGEN(nlev)/(nlev-1)*(maxcol-mincol)+mincol
                  lev = FINDGEN(nlev)/(nlev-1)*(maxlev-minlev)+minlev


PLACE_COLOR_BAR,barcharsize=charsize*0.4,col=col,nlev=nlev,lev=lev,$
```

```
bartitle=bartitle,makeformat = makeformat


                CTRL_PLOT, /close, save=GSave
                GFile = "yAoA_"+STRING(block_count,FORMAT = '(I5.5)')
                GSave = 1

                xPlots = 1
                yPlots = 1

                xsize = 950
                ysize = 950


CTRL_PLOT,wid=1,xsize=xsize,ysize=ysize,xpos=10,path=GPath,png=1,show=1
,title="Interferometry_Phase",$
                overplot=0,file=GFile

                !P.MULTI = [0,xPlots,yPlots]
                CONTROL_PLOT,close=0

                maxcol = 79
                mincol = 50
                maxlev = 2.
                minlev = -2
                makeformat = '(F6.0)'
                novalid_col = !P.COLOR

                title = 'Angle Of Arrival Y-BaseLine (deg)'
                xtitle = 'time (s)'

                charsize = 2

                xrange = [MIN(mTime),MAX(mTime)]
                yrange = [MIN(mRange),MAX(mRange)]

CONT_IMAGE,yAoA,mTime,mRange,TOP=maxcol,BOTTOM=mincol,MINA=minlev,XSTYL
E=1,MAXA=maxlev,$

XRANGE=xrange,YRANGE=yrange,YSTYLE=1,TITLE=title,XTITLE=xtitle,XTICKN=x
tickn,$
                  YTITLE='Range (km)',SUBTITLE=subtitle,
NOVALID_COL=novalid_col,XMARGIN=[8,12],$
                  CHARSIZE=charsize

                nlev = 29
                col = FINDGEN(nlev)/(nlev-1)*(maxcol-mincol)+mincol
                lev = FINDGEN(nlev)/(nlev-1)*(maxlev-minlev)+minlev


PLACE_COLOR_BAR,barcharsize=charsize*0.4,col=col,nlev=nlev,lev=lev,$

bartitle=bartitle,makeformat = makeformat
```

```
                CTRL_PLOT, /close, save=GSave

             ENDIF


          ENDIF

          ENDREP UNTIL Success NE 1
          iFile= iFile + 1
        ENDREP UNTIL iFile GE NFiles



        CLOSE,/ALL,/FORCE
        PRINT,'End of the processing...'

      END




      ;+
      ; NAME:
      ;   READ_JRO
      ;
      ; PURPOSE:
      ;   reads the JRO data and returns critical header information.
      ;
      ; INPUTS:
      ;   PATH = A string to define path of the folder which ccontains
the HDF5 files.
      ;   CUR_PTR = The dataset being accessed in the current file
      ;
      ; OUTPUTS:
      ;   RESULT = 1 if the files was opened successfully. If the file
does not exist it is -1 (It was not opened).
      ;   HEADER = A structure giving the information of the experiment
(e.g. time and range).
      ;   VOLT = Formatted data for post processing
      ;
      ; Authors:
      ;   Freddy Galindo, Penn State University
      ;

      FUNCTION
READ_JRO,Filename=Filename,cur_ptr=cur_ptr,volt=volt,header=header

        COMMON ReadInfo, fullheader, idltype, nfile

        IF N_ELEMENTS(cur_ptr) EQ 0 THEN cur_ptr = 0
        IF N_ELEMENTS(nfile) EQ 0 THEN nfile=1

        OPENR, nfile, FileName
```

```
  POINT_LUN,nfile,cur_ptr

  IF EOF(nfile) EQ 1 THEN BEGIN
    CLOSE,nfile
    RETURN,-1
  ENDIF

  ShortHeader = { SHEADER, HeaderLen:0UL $
          , HeaderVer:0US $
          , CurrentBlock:0UL $
          , time:0UL $
          , milliseconds:0US $
          , timezone:0US $
          , dstflag:0US $
          , ErrorCount:0UL }
  READU, nfile, ShortHeader


IF Shortheader.headerlen GT 24 THEN BEGIN

  SystemHeader = {SYSHEADER, SysHeaderLen:0UL $
                           , AcqSamples:0UL $
                          , AcqProfiles:0UL $
                          , AcqChannels:0UL $
                        , ADCResolution:0UL$
                       , PCIDIOBUSWidth:0UL }
  READU, nfile, SystemHeader


  DataWindow = { DWSTR, fH0:0E $
                      , fDH:0E $
                     , nNSA:0UL }

  RCHeader = { RC_HEADER, $
             RCHeaderLen:0UL, $
             EspType:0UL, $
             NTX:0UL, $
             Ipp: 0.0, $
             TxA: 0.0, $
             TxB: 0.0, $
             NumWindows: 0ul, $
             NumTaus: 0ul, $
             CodeType: 0ul, $
             Line6Function: 0ul, $
             Line5Function: 0ul, $
             Clock: 0.0, $
             PrePulseBefore: 0ul, $
             PrePulseAfter: 0ul, $
             RangeIpp: BYTARR(20), $
             RangeTxA: BYTARR(20), $
             RangeTxB: BYTARR(20)}
  READU, nfile, RCHeader

  ;Initial values
  RcSamWin=0UL
```

```
Taus=0UL
NumCodes = 0UL
NumBauds = 0UL
CODE = 0UL
FLIP1 = 0UL
FLIP2 = 0UL

IF RCHeader.NumWindows GT 0 then begin
  RcSamWin = REPLICATE(DataWindow,RCHeader.NumWindows)
  READU, nfile, RcSamWin
ENDif

IF RCHeader.NumTaus GT 0 then begin
  Taus = FLTARR(RCHeader.NumTaus)
  READU, nfile, Taus
ENDif

IF RCHeader.CodeType GT 0 then begin
  READU, nfile, NumCodes
  READU, nfile, NumBauds
  SplitParts=ceil(NumBauds/32.0)
  CODE=LONARR(SplitParts,NumCodes)
  READU, nfile, CODE
ENDif

IF RCHeader.Line5Function EQ 1 then READU, nfile, FLIP1
IF RCHeader.Line6Function EQ 1 then READU, nfile, FLIP2

RCDynHeader = { RCDYNHEADER_RAW, $
        RcSamWin:RcSamWin,$
        Taus:Taus, $
        NumCodes:NumCodes, $
        NumBauds:NumBauds, $
        CODE:CODE, $
        FLIP1:FLIP1, $
        FLIP2:FLIP2}

PHeader = { PHEADER $
        , PHeaderLen:0UL $
        , DataType:0UL $
        , SizeOfDataBlock:0UL $
        , BlockFFTProfiles:0UL $
        , FileBlocks:0UL $
        , DataWindows:0UL $
        , ProcessFlags:0UL $
        , nCohInt:0UL $
        , nIncohInt:0UL $
        , nTotalSpectra:0UL}

READU, nfile, PHeader

PDSamWin=0UL
TotalPSam=0UL

IF PHeader.DataWindows GT 0 then begin
```

```
          PDSamWin = REPLICATE(DataWindow,PHeader.DataWindows)
          READU, nfile, PDSamWin
          TotalPSam=FIX(TOTAL(PDSamWin.nNSA,/NAN))
        ENDif

        TotalSS=0UL;Number of SelfSpectra (mostly = number of channels)
        TotalCS=0UL;Number of CrossSpectra
        SpcComb=0UL
        if PHeader.nTotalSpectra gt 0 then begin
          SpcComb = BYTARR(2,PHeader.nTotalSpectra)
          READU, nfile, SpcComb;Number of Self and Cross - Spectra
combinations
          FOR i=0,PHeader.nTotalSpectra-1 DO BEGIN
              IF SpcComb(0,i) EQ  SpcComb(1,i) THEN TotalSS=TotalSS+1
          ENDFOR
          TotalCS=PHeader.nTotalSpectra-TotalSS;Number of CrossSpectra
        endif

        PDynHeader = { PDYNHEADER_RAW, $
                 PDSamWin:PDSamWin, $
                 TotalPSam:TotalPSam, $
                 SpcComb:SpcComb, $
                 TotalSS:TotalSS, $
                 TotalCS:TotalCS}


        fullheader={ FULL_HEADER_RAW, $
                  ShortHeader:ShortHeader,$
                  system_header:SystemHeader,$
                  radar_header:RCHeader,$
                  RCDynHeader:RCDynHeader,$
                  process_header:PHeader,$
                  PDynHeader:PDynHeader}

        ;Allocate memory and read the Block of data data structure
        datatype = ROUND(ALOG((pheader.ProcessFlags/2l^6) MOD
2l^6)/ALOG(2))
        idltype = datatype + (datatype LE 2) + 11*(datatype EQ 3)
      ENDIF

      ; Leyendo los datos crudos (rawdata).
      volt_iq =
MAKE_ARRAY(2,fullheader.system_header.AcqChannels,fullheader.PDynHeader
.TotalPSam,fullheader.process_header.BlockFFTProfiles,type=idltype,/noz
ero)
      READU, nfile, volt_iq

      fullheader.ShortHeader = ShortHeader
      POINT_LUN,-nfile,cur_ptr

      IF fullheader.process_header.DataType EQ 0 THEN BEGIN
          volt_iq = TEMPORARY(volt_iq) + 128b
          volt_real = volt_iq[0,*,*,*] - 128
          volt_imag = volt_iq[1,*,*,*] - 128
      ENDIF ELSE BEGIN
```

```
          volt_real = volt_iq[0,*,*,*]
          volt_imag = volt_iq[1,*,*,*]
      ENDELSE

        volt_real =
REFORM(volt_real,fullheader.system_header.AcqChannels,fullheader.PDynHe
ader.TotalPSam,fullheader.process_header.BlockFFTProfiles,/overwrite)
        volt_imag =
REFORM(volt_imag,fullheader.system_header.AcqChannels,fullheader.PDynHe
ader.TotalPSam,fullheader.process_header.BlockFFTProfiles,/overwrite)

        ; Raw data per block of data.
        volt = COMPLEX(volt_real,volt_imag)
        volt = TRANSPOSE(volt,[2,1,0])

        ; Creating an header
        range = fullheader.RCDynHeader.RCSamWin.fh0+$

FINDGEN(fullheader.PDynHeader.TotalPSam)*fullheader.RCDynHeader.RCSamWi
n.fDH
        time =
FINDGEN(fullheader.process_header.BlockFFTProfiles)*fullheader.radar_he
ader.ipp*0.001/150 ; in s

        NProfs = fullheader.process_header.BlockFFTProfiles
        NRans = fullheader.PDynHeader.TotalPSam
        NChans = fullheader.system_header.AcqChannels

        header = {range:range, time:time, NProfs:NProfs, NRans:NRans,
NChans:NChans}

        CLOSE, nfile
        RETURN,1

      END



      ;+
      ; NAME:
      ;   READ_PSU
      ;
      ; PURPOSE:
      ;   The READH_HDF5 reads information of the experiment from the
HDF5  file. If the reading-process  fails it
      ;   returns 1 or -1  if it  was successful. In addition this
function returns the experiment information in a
      ;   header (Like typical Jicamarca header).
      ;
      ; INPUTS:
      ;   PATH = A string to define path of the folder which contains
the HDF5 files.
      ;   CUR_PTR = The dataset being accessed in the current file
      ;
      ; OUTPUTS:
```

```
      ;   RESULT = 1 if the files was opened successfully. If the file
does not exist it is -1 (It was not opened).
      ;   HEADER = A structure giving the information of the experiment
(e.g. time and range).
      ;   VOLT = Formatted data for post processing
      ;
      ; Author:
      ;   Tejas Nagarmat, Penn State University
      ;

      FUNCTION
READ_PSU,Filename=Filename,cur_ptr=cur_ptr,volt=volt,header=header

        IF N_ELEMENTS(Filename) EQ 0 THEN Filename = 'path/to/data'

        IF N_ELEMENTS(set) EQ 0 THEN set = 42495L


        IF FILE_TEST(Filename) EQ 0 THEN RETURN, -1

        ; Getting informacion of the HDF file.
        ff = H5_PARSE(Filename)

        ; Opening the HDF5 to read the information.
        id = H5F_OPEN(Filename)

      ; GET NUMBER OF TABLES
        DNum = N_ELEMENTS(tag_names(ff)) - 26


        IF cur_ptr LE DNum-1 THEN BEGIN

        dataset = 'T0'+STRING(cur_ptr,FORMAT='(I7.7)')
        print,dataset


         ; Getting dimensions of the raw data.
        tmp_id = H5D_OPEN(id,dataset)
        tmp_id1 = H5D_GET_SPACE(tmp_id)
        dims = H5S_GET_SIMPLE_EXTENT_DIMS(tmp_id1)
        H5D_CLOSE,tmp_id

        ; Reading Raw data
        tmp_id = H5D_OPEN(id,dataset)
        id0 = H5D_GET_SPACE(tmp_id)
        memory_space = H5S_CREATE_SIMPLE(dims)
        raw_data =
H5D_READ(tmp_id,file_space=id0,memory_space=memory_space)

        ;READING TAGS
        NumChan=ff.CHANNELS._DATA
        start_time = ff.START_TIME._DATA
        sample_rate = ff.SAMPLE_RATE._DATA
        bandwidth = ff.BANDWIDTH._DATA
        decimation = ff.DECIMATION._DATA
        output_rate = ff.OUTPUT_RATE._DATA
        ipp = ff.IPP._DATA

        H5D_CLOSE,tmp_id

        ;create a volt variable

        volt = complexarr((dims[0]/NumChan),dims[1],NumChan)
;1734,250,4
        pre_volt = (complex(raw_data.real, raw_data.imag)); Struct to
array

        ref =
REFORM(pre_volt,NumChan,dims[1],dims[0]/NumChan);4,250,1734 Channel
data parsing

        FOR i=0, NumChan-1 DO BEGIN
        volt(*,*,i)=ref(i,*,*)
        ENDFOR

        volt=transpose(volt,[1,0,2]);250, 1734, 4

        NumChan=ff.CHANNELS._DATA
        start_time = ff.START_TIME._DATA
        sample_rate = ff.SAMPLE_RATE._DATA
        bandwidth = ff.BANDWIDTH._DATA
        decimation = ff.DECIMATION._DATA
        output_rate = ff.OUTPUT_RATE._DATA
        ipp = ff.IPP._DATA
        rx_start = ff.RXWIN_START._DATA
        rx_stop = ff.RXWIN_STOP._DATA

        height_lower = (rx_start/bandwidth)*1.5E8/1000
        height_upper = (rx_stop/bandwidth)*1.5E8/1000


time=FINDGEN((SIZE(volt))[1])*((SIZE(volt))[1]*ipp)/(SIZE(volt))[1]
        range=height_lower + FINDGEN(rx_stop -
rx_start)*(height_upper)/rx_stop

        header = {range:range, time:time,$
                  NProfs:(SIZE(volt))[1], NRans:(SIZE(volt))[2],
NChans:(SIZE(volt))[3]}

        cur_ptr = cur_ptr + 1
        RETURN, 1

        ENDIF ELSE BEGIN

         RETURN, -1

        ENDELSE
      END
```

Bibliography

[1] Chau, J. L. and Woodman, R. F.: Observations of meteor-head echoes using the Jicamarca 50MHz radar in interferometer mode, Atmos. Chem. Phys., 4, 511-521, doi:10.5194/acp-4-511-2004, 2004.

[2] Chapin, E. and Kudeki, E.: Radar interferometric imaging studies of long-duration meteor echoes observed at Jicamarca, J. Geophys. Res., 99, 8937-8949, 1994.

[3] Mathews, J. D., Meisel, D. D., Kunter, K. P., Getman, V. S., and Zhou, Q. H.: Very high resolution studies of micrometeors using the Arecibo 430MHz radar, Icarus, 126, 157-169, 1997.

[4] Zhou, Q. H., Perillat, P., Cho, J. Y. N., and Mathews, J. D.: Simultaneous meteor echo observations by large aperture VHF and UHF radars, Radio Sci., 33, 1641-1654, 1998.

[5] Close, S., Hunt, S. M., McKeen, F. M., and Minardi, M. J.: Characterization of Leonid meteor head echo data collected using the VHF-UHF advanced reserach projects agency long-range tracking and instrumentation radar (ALTAIR), Radio Sci., 37, doi:10.1029/2000RS002602, 2002.

[6] Close, S., Oppeheim, M., Hunt, S., and Dyrud, L.: Scattering characteristics of hig-resolution meteor head echoes detected at multiple frequencies, J. Geophys. Res., 107, doi:10.1029/2002JA009253, 2002.

[7] Janches, D., Nolan, M. C., Meisel, D. D., Mathews, J. D., Zhou, Q. H., and Moser, D. E.: On the geocentric micrometeor velocity distribution, J. Geophys. Res., 108, doi:10.1029/2002JA009789, 2003.

[8] Hocking, W. K., Fuller, B., and Vandepeer, B.: Real-time determinationof meteor-related parameters utilizing modern digital technology, J. Atmos. Sol. Terr. Phys., 63, 155-169, 2001.

[9] Woodman, R. F.: Inclination of the geomagnetic field measured by an incoherent scatter technique, J. Geophys. Res., 76, 178-184, 1971.

[10] Johnson C. R. and Sethares W. A.: Telecommunication Breakdown. Upper Saddle River, NJ: Prentice Hall, pp. 11–12, 111. , 2003.

[11] Dillinger M., Madani K., Alonistioti N.: Software Defined Radio: Architectures, Systems and Functions Page xxxiii, Wiley & Sons, 2003.

[12] Omer M: Intelligent Spectrum Sensor Radio: http://rave.ohiolink.edu/etdc/view?acc_num=wright1215360432

[13] Marsh D.; Software-defined Radio tunes, EDN Tech Trends, 2005.

[14] http://gnuradio.org/redmine/projects/gnuradio

[15] http://dev.emcelettronica.com/gnu-radio-open-source-software-defined-radio

[16] Shen, Dawei: Tutorial 4: the USRP Board.‖ Introduction. SDR Documentation. Notre Dame, IN: University of Notre Dame, 2005. Oct. 2007. http://www.nd.edu/~jnl/sdr/docs/

[17] Ettus, Matt. "USRP Datasheet." Ettus Research LLC. Oct. 2007 http://www.ettus.com/

[18] http://gnuradio.org/redmine/projects/gnuradio/wiki/UsrpFAQIntroFPGA

[19] https://github.com/rseal/GnuRadar

[20] http://www.rfcafe.com/references/electrical/noise-figure.htm

[21] https://github.com/rseal/BitPatternGenerator

[22] http://www.novatechsales.com/Bench-Signal-Generator.html

[23] http://gnuradio.org/redmine/projects/gnuradio/wiki/USRPClockingNotes

[24] Lyons R., Understanding Digital Signal Processing, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1996.

[25] Silberschatz A., Galvin P., Operating System Concepts, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1997.

[26] http://www.hdfgroup.org/HDF5

[27] http://www.hdfgroup.org/hdf-java-html/hdfview

[28] http://www.phys.hawaii.edu/~anita/new/papers/militaryHandbook/sig-sort.pdf

[29] Balogh L, Kollár I: Angle of Arrival Estimation Based on Interferometer Principle, Proceedings of the IEEE International Symposium on Intelligent Signal Processing. Budapest, Hungary, IEEE, pp. 219-224, 04/09/2003-06/09/2003.

[30] http://www.exelisvis.com/portals/0/tutorials/idl/Programming_in_IDL.pdf