SOFTWARE DEFINED RADAR FOR VHF METEOR
RESEARCH AT THE PENNSYLVANIA STATE UNIVERSITY

A Thesis in
Electrical Engineering
by
Ryan Seal

© 2014 Ryan Seal

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2014
The thesis of Ryan Seal was reviewed and approved* by the following:

Julio Urbina
Associate Professor of Electrical Engineering
Thesis Adviser

John Mathews
Professor of Electrical Engineering

Kultegin Aydin
Professor of Electrical Engineering
Head of the Department of Electrical Engineering

*Signatures are on file in the Graduate School.
Abstract

Very High Frequency (VHF) radars, defined by propagating radio frequencies between 30 and 300 MHz, were popularized during World War II as airborne intercept radars. Modern radar technology has long since relegated VHF radars obsolete for surveillance and communication. However, the VHF band, due to the composition of the Earth’s atmosphere, is capable of providing high-resolution mapping and remote sensing of the atmosphere, including the study of meteoroids. In this thesis, the design and construction of a VHF meteor radar will be provided, with an emphasis on the use of open-source tools, including elements of both hardware and software. The intention of this work is to provide researchers with a cost-effective, completely open platform that can be freely customized and expanded. Readers will find an in-depth analysis of signal processing techniques used, including modifications required for use in hardware. Additionally, the design of a networked client-server architecture, used to control and operate the system remotely, will be provided. The work will conclude with results from initial observations which will be compared with theoretical calculations.
# Table of Contents

List of Tables ........................................................................................................... vii

List of Figures ........................................................................................................... viii

Chapter 1. Introduction ........................................................................................... 1

Chapter 2. System Requirements ........................................................................... 3
  2.1 Meteor Science ............................................................................................... 3
  2.1.1 Underdense Echo ..................................................................................... 3
  2.1.2 Overdense Echo ....................................................................................... 7
  2.2 Antenna .......................................................................................................... 7
  2.3 Front End Gain Estimation ............................................................................ 10
  2.4 Front End Design .......................................................................................... 13
    2.4.1 Front-end Component Parameters ....................................................... 15

Chapter 3. Radar System Description .................................................................... 18
  3.1 Software Defined Radar ................................................................................ 22
    3.1.1 Sampling ............................................................................................... 23
    3.1.2 Downconverter ..................................................................................... 24
      3.1.2.1 CORDIC Algorithm .................................................................... 28
    3.1.3 Cascaded Integrator Comb (CIC) Filter .............................................. 37
      3.1.3.1 Implementation ............................................................................. 49
3.1.4 CIC Compensation Filter ............................... 53
3.1.5 Pulse Synchronization and Data Tagging ................. 56

Chapter 4. Software Architecture and Design Implementation ........ 59

4.1 Communication and Messaging Protocol ...................... 61
  4.1.1 MQTT ................................................... 62
  4.1.2 Google Protocol Buffers ............................... 64

4.2 Client Architecture ........................................ 64
  4.2.1 Configuration Tool .................................... 65
    4.2.1.1 General Settings ................................ 66
    4.2.1.2 DDC Settings .................................. 66
    4.2.1.3 Data Window and Pulse Settings ................. 67
    4.2.1.4 HDF5 Header Information ....................... 68
    4.2.1.5 File Settings .................................. 68
  4.2.2 Runtime Tool .......................................... 68

4.3 Server Architecture ....................................... 69
  4.3.1 Producer Thread ..................................... 71
    4.3.1.1 Device Interface ................................. 73
    4.3.1.2 USRP1 Library .................................. 74
    4.3.1.3 Data Synchronization ............................ 75
  4.3.2 Consumer Thread ..................................... 76
    4.3.2.1 Shared Header File .............................. 77
    4.3.2.2 Data Formatting and Storage .................... 78
List of Tables

2.1 Table defining parameters used in the 5-element Yagi-Uda antenna array used at the Rock Springs radar site. Element 0 represents the bottom-most element and spacing represents the distance from the current element to the next element, therefore no element spacing exists for the final element in the configuration. .................................................. 9

2.2 Table listing RF components used for radar signal conditioning and reception. ................................................................. 16

2.3 Overall Gain and NF of the RF front-end component chain. .......... 17

3.1 Illustration of resource usage when varying the number of stages, fixed vs. variable decimation values, and bit-pruning vs. fixed-width accumulator stages. ................................................................. 50
List of Figures

2.1 Illustration of *underdense* trail echo and equation for backscatter power resulting from a single electron. ................................. 5

2.2 Yagi-Uda array parameters ............................................. 8

2.3 5-element Yagi-Uda antenna simulation with $F_c = 49.80$ MHz. .......... 9

2.4 Antenna input level compared with compatible ADC level with no RF front end gain in present. ........................................ 12

2.5 Antenna input level compared with compatible ADC level with no RF front end gain in present. ........................................ 13

2.6 Block diagram describing the analog RF front-end signal conditioning and amplification. ............................................... 16

3.1 Block diagram of the Penn State radar system. .......................... 18

3.2 Frequency and time domain representation of an analytic bandpass signal. 19

3.3 Block diagram of the receiver path in the radar system. ................. 23

3.4 AD9862 Receive Path Block Diagram ................................... 24

3.5 Illustration of Digital I/Q downconverter implementations. ............ 25

3.6 Depiction of independently generated in-phase and quadrature signals used to illustrate the process of quadrature signal downconversion. . . . 26

3.7 Real bandpass signal generation using signal components in Figure 3.6 with a carrier signal frequency of 49.8 MHz. .......................... 27
3.8 Demodulation and low-pass filtering applied to the bandpass signal shown in Figure 3.7. .......................... 28

3.9 Illustration of a right-angle triangle with angle $\theta$ and unity-length hypotenuse. Applying the given constraints, $\cos \theta$ can be derived. ....... 31

3.10 Illustration of a 12-stage CORDIC algorithm iteratively converging on the given angle of $\pi/3$ radians. ............................. 34

3.11 Illustration of the phase wheel concept used to describe phase generation of the demodulating frequency used by the DDC in hardware. ......... 35

3.12 A hardware representation of an N-bit Phase Accumulator block. This output width is given by P, which is a subset of N, containing the right-most bits of the input width. ................................. 36

3.13 16-Tap Boxcar filter example illustrating the filter’s input response in both time and frequency domains. Note that the filter’s frequency response has been normalized for display purposes. ................... 38

3.14 Illustration of a single-stage, 4-bit CIC filter with $R = 1$ and ramp input applied. Although the filter’s internal state is unstable (i.e. overflow), the output of the system is correct. Using Two’s Complement arithmetic, the internal overflow has wrap-around behavior and the integrator’s addition is undone by the differentiator’s subtraction, thereby appearing stable when observing the system as a black box (i.e. observing only at the input and output of the system) .................. 41
3.15 Illustration of downsampling-by-R applied prior to the comb stages. Justification is provided by the Noble Identities [12]. The resulting structure at the bottom of this Figure is referred to as the Cascading Integrator Comb (CIC) filter. ................................................................. 41

3.16 Error statistics given J bits discarded from a filter stage when assuming uniformly distributed error. ................................................................. 44

3.17 4-stage CIC filter response ................................................................. 51

3.18 5-stage CIC filter response ................................................................. 52

3.19 Illustration of the compensation filter’s ability to flatten out the drooping response of the CIC filter. ................................................................. 55

3.20 Custom FIFO implementation in USRP1 firmware to provide pulsed mode operation ................................................................. 57

3.21 Signals used in system for pulsed mode operation ................................................................. 58

4.1 Block diagram illustrating client-server architecture of the system. ...... 61

4.2 Client-server architecture used in the GnuRadar system with 1 or more clients communicating with a single server using a request-reply message pattern. ................................................................. 63

4.3 Publish-subscribe message pattern used in the GnuRadar system with N subscribers and a single publisher. ................................................................. 63

4.4 GnuRadar Configuration Tool Interface ................................................................. 65

4.5 GnuRadar Run-time Tool Interface ................................................................. 69

4.6 Depiction of the producer-consumer pattern and usage of a circular buffer. 71
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7</td>
<td>Class diagram illustrating the <em>producer-consumer</em> model’s architecture in</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>the GnuRadar system.</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>Class diagram illustrating <em>ProducerThread</em> interaction in the GnuRadar</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>system.</td>
<td></td>
</tr>
<tr>
<td>4.9</td>
<td>Class diagram depicting <em>USRPI</em> receiver implementation into the <em>Gnu-Radar</em></td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>system.</td>
<td></td>
</tr>
<tr>
<td>4.10</td>
<td>Class diagram depicting <em>ConsumerThread</em> class and dependent classes</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>and libraries.</td>
<td></td>
</tr>
<tr>
<td>4.11</td>
<td>Illustration of text-based YAML file format used to provide header</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>information to external processes using the shared circular buffer.</td>
<td></td>
</tr>
<tr>
<td>4.12</td>
<td>Visualization of the format used in the <em>GnuRadar</em> system.</td>
<td>80</td>
</tr>
<tr>
<td>4.13</td>
<td>Visualization of a data table, which contains one second of I/Q data with</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>range cells in columns and PRIs in rows.</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td><em>USRPI</em> receiver test setup used for gain calibration.</td>
<td>83</td>
</tr>
<tr>
<td>5.2</td>
<td><em>USRPI</em> receiver test setup block diagram illustrating unknown loss in</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>the system’s matching circuitry.</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Image contaminated by interference.</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>Smoothed image using a 3x3 median filter.</td>
<td>90</td>
</tr>
<tr>
<td>6.3</td>
<td>Illustration of noise floor estimation using α-trimmed mean filter.</td>
<td>92</td>
</tr>
<tr>
<td>6.4</td>
<td>Illustration of strong interference in image.</td>
<td>94</td>
</tr>
<tr>
<td>6.5</td>
<td>Result of interference removal algorithm.</td>
<td>94</td>
</tr>
<tr>
<td>6.6</td>
<td>Illustration of clustering algorithm given a set of thresholded detections.</td>
<td>97</td>
</tr>
</tbody>
</table>
7.1 Autocorrelation function of 28-baud biphase code. . . . . . . . . . . . . 100
7.2 Number of events recorded per hour for the collection duration. . . . . 101
7.3 Histogram illustrating detected event durations. . . . . . . . . . . . . . 102
7.4 Histogram depicting the time between detected events. . . . . . . . . . 103
7.5 Histogram depicting range of detected events. . . . . . . . . . . . . . . 104
7.6 Histogram displaying SNR levels of detected events. . . . . . . . . . . 105
A.1 Meteor Event T00014326 . . . . . . . . . . . . . . . . . . . . . . . . . . 112
A.2 Meteor Event T00019008 . . . . . . . . . . . . . . . . . . . . . . . . . . 113
A.3 Meteor Event T00020777 . . . . . . . . . . . . . . . . . . . . . . . . . . 114
A.4 Meteor Event T00026107 . . . . . . . . . . . . . . . . . . . . . . . . . . 115
A.5 Meteor Event T00026270 . . . . . . . . . . . . . . . . . . . . . . . . . . 116
A.6 Meteor Event T00026494 . . . . . . . . . . . . . . . . . . . . . . . . . . 117
A.7 Meteor Event T00027301 . . . . . . . . . . . . . . . . . . . . . . . . . . 118
A.8 Meteor Event T00029887 . . . . . . . . . . . . . . . . . . . . . . . . . . 119
A.9 Meteor Event T00032898 . . . . . . . . . . . . . . . . . . . . . . . . . . 120
A.10 Meteor Event T00034926 . . . . . . . . . . . . . . . . . . . . . . . . . . 121
A.11 Meteor Event T00037650 . . . . . . . . . . . . . . . . . . . . . . . . . . 122
A.12 Meteor Event T00039549 . . . . . . . . . . . . . . . . . . . . . . . . . . 123
A.13 Meteor Event T00039550 . . . . . . . . . . . . . . . . . . . . . . . . . . 124
A.14 Meteor Event T00041871 . . . . . . . . . . . . . . . . . . . . . . . . . . 125
A.15 Meteor Event T00043312 . . . . . . . . . . . . . . . . . . . . . . . . . . 126
A.16 Meteor Event T00044059 . . . . . . . . . . . . . . . . . . . . . . . . . . 127
Chapter 1

Introduction

In a typical day, billions of meteoroids enter the Earth's atmosphere and provide an opportunity for researchers across several fields of study to better understand the properties of global meteor flux, which directly affect models used in fields such as: 1) solar system evolution, 2) gravity wave imaging, 3) noctilucent cloud formation, and 4) the study metal deposition into the mesosphere/lower thermosphere (MLT) and their resultant effects. Since the late 1920s optical, radio, and radar systems have been used to capture and record meteoroids. In more recent years, radar systems have been able to detect two new types of meteoroid echoes, rekindling interest in the field and paving a path for a new generation of meteor radars which use state-of-the-art technology and typically outperform older counterparts at a fraction of the cost. These new systems are capable of directly sampling RF signals, processing data primarily in the digital domain. Additionally, they are completely software configurable and are now commonly referred to as Software Defined Radars (SDR). SDR platforms have become increasingly popular [22, 24] as researchers, hobbyists, and military seek more efficient and cost-effective means for radar construction and operation. By definition, these systems utilize a software-based interface for configuration in contrast to traditional, hard-wired platforms. This thesis will begin with a discussion of basic system requirements necessary for system operation. Next, a complete analysis and design of components used to meet
requirements will be covered, along with calibration and testing of the digital receiver used. In the final chapters, a new technique for meteor discrimination and detection will be introduced, followed by an analysis from a 13-hour data collection that took place on June 14th, 2013.
Chapter 2

System Requirements

2.1 Meteor Science

The subject of meteor theory is beyond the scope of this thesis, but basic models have been introduced to estimate the target’s dynamic range (DR) which will determine the system’s gain requirements. Both models presented are specular, backscattering models (i.e. a monostatic radar configuration) and are used to determine the upper-bound of the target’s DR.

2.1.1 Underdense Echo

Micro-meteoroids enter the Earth’s atmosphere with kinetic energy sufficient to ionize the meteoroid’s atoms, creating a column of free electrons in its path. To develop a basic model, the following assumptions are made:

1. the column does not expand radially;

2. the electrons do not recombine, attach, or diffuse;

3. and each electron responds to the incident electromagnetic wave independently.
These assumptions define the *underdense* meteor echo and the scattering area is defined as that produced by a free electron [20]

\[
\sigma_e = 4\pi r_e^2 \sin^2 \gamma \left( m^2 \right),
\]

where \( r_e \) is classical electron radius and \( \gamma \) is the angle between the electric field vector of the incident wave and the line of sight to the receiver. For the backscatter case, \( \gamma = 90^\circ \) and \( \sigma_e \approx 1 \times 10^{-28}(m^2) \). Using the Radar Cross Section (RCS) of the electron in a backscatter configuration, the radar range equation is given by

\[
\Delta P_R = \frac{P_T G}{4\pi R^2} \frac{\sigma_e}{4\pi} \frac{\lambda^2 G}{4\pi} = \frac{P_T G^2 \lambda^2 \sigma_e}{(4\pi)^3 R^4} (W),
\]

where the first term represents the directed power radiating from the antenna, the second term describes the scattering properties of an electron, and the third term represents the effective area of the antenna. An illustration of an *underdense* trail echo is provided in Figure 2.1 and clearly shows that the target range varies with respect to time, with the minimum range (maximum power) depicted by \( R_0 \). The overall power response from this trail is computed by integrating along the meteor’s path. Assuming constant charge density along the path \( s \), the total echo from all electrons along the path is given by [20]

\[
A_R = q \sqrt{2z\Delta P_R} \int_s \sin \left( 2\pi ft - \frac{4\pi R}{\lambda} \right) ds,
\]

where \( q \) is the electron charge line density, \( z \) is the receiver’s input impedance, and \( f \) is the transmission frequency. The integral in Equation (2.3) is difficult to evaluate and a
Figure 2.1. Illustration of *underdense* trail echo and equation for backscatter power resulting from a single electron.

\[
\Delta P_R = \frac{P_T G \sigma_e}{4\pi R_0^2} \frac{G\lambda^2}{4\pi} = \frac{P_T G^2 \lambda^2 \sigma_e}{(4\pi)^3 R_0^4}
\]
series approximation around $R_0$ is used

$$A_R = q \frac{\sqrt{2z \Delta P_R}}{2} (C \sin \chi - S \cos \chi), \quad (2.4)$$

where $C$ and $S$ are the Fresnel integrals given by

$$C = \int_l \cos \frac{\pi \chi^2}{2} dl, \quad (2.5)$$

$$S = \int_l \sin \frac{\pi \chi^2}{2} dl. \quad (2.6)$$

Using Equation (2.4), power received at the receiver’s is defined by

$$P_R = \frac{A_R^2}{2z} = \frac{\Delta P_R R_0 \lambda}{2} \left[ \frac{C^2 + S^2}{2} \right]^2, \quad (2.8)$$

and, substituting Equation (2.2) results in

$$P_R = 2.5 \times 10^{-32} P_T G^2 \left( \frac{\lambda}{R_0} \right)^3 \left[ \frac{C^2 + S^2}{2} \right]^2 q^2. \quad (2.9)$$

Later work on the subject [20] showed that contributions further away from $R_0$ were negligible and a value of unity can be substituted in place of the Fresnel integrals when considering a few Fresnel zones on either side of $R_0$. Under this assumption, the final RMS power estimate of an underdense trail at the receiver’s input becomes

$$P_R = 2.5 \times 10^{-32} P_T G^2 \left( \frac{\lambda}{R_0} \right)^3 q^2 \langle W \rangle. \quad (2.10)$$
2.1.2 Overdense Echo

Once the electron density of a trail echo reaches a certain level, the scattered energy can no longer be modeled as a sum of independent scatters, and the interaction between electrons must be accounted for. In this model, the incident wave does not fully penetrate the column. The density threshold separating underdense and overdense echoes is $2 \times 10^{14}$ electrons per meter according to [20]. Scatter from overdense trails are considerably larger in magnitude than underdense echoes. Additionally, they are typically visible for several seconds, producing trails perturbed by atmospheric wind which modulate the returned echo. The overdense model is given by

$$P_R = \sqrt{\frac{r_c q P_T G^2}{\epsilon} \left( \frac{\lambda}{R_0} \right)^3} = 1.6 \times 10^{-11} P_T G^2 \sqrt{q} \left( \frac{\lambda}{R_0} \right)^3. \quad (2.11)$$

In Section 2.3, this model will be used to determine system gain required by the RF front-end.

2.2 Antenna

The Rock Springs radar site, located in rural Pennsylvania, is owned by the Pennsylvania State University and operated by the Applied Signal Processing and Instrumentation Laboratory (ASPIRL). The site is equipped with several different antenna structures to host a variety of research interests. The system designed in this thesis used two colocated, 5-element Yagi-Uda antennas operating in the 6-meter band. One antenna was used for transmission and the other for reception. The Yagi-Uda antenna is an array-based structure with carefully sized and spaced parasitic dipole elements used
to radiate energy [2], and is typically defined by one or two reflector elements, a single excitation element, and one or more director elements. A diagram illustrating parameters for a Yagi-Uda array are provided in Figure 2.2. Using the aforementioned configuration, simulations were carried out to characterize performance by entering the antenna’s physical dimensions, as shown in Table 2.1, into NEC2++ software [19]. Simulation results are summarized in Figure 2.3.
<table>
<thead>
<tr>
<th>Element Number</th>
<th>Length (m)</th>
<th>Radius (m)</th>
<th>Spacing (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.8956</td>
<td>$1.27 \times 10^{-2}$</td>
<td>1.21920</td>
</tr>
<tr>
<td>1</td>
<td>2.8734</td>
<td>$1.27 \times 10^{-2}$</td>
<td>2.41140</td>
</tr>
<tr>
<td>2</td>
<td>2.6400</td>
<td>$1.27 \times 10^{-2}$</td>
<td>3.62790</td>
</tr>
<tr>
<td>3</td>
<td>2.6400</td>
<td>$1.27 \times 10^{-2}$</td>
<td>4.82910</td>
</tr>
<tr>
<td>4</td>
<td>2.6160</td>
<td>$1.27 \times 10^{-2}$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1. Table defining parameters used in the 5-element Yagi-Uda antenna array used at the Rock Springs radar site. Element 0 represents the bottom-most element and spacing represents the distance from the current element to the next element, therefore no element spacing exists for the final element in the configuration.

$\Delta \phi_{3dB} \approx 48.0^\circ$

$G_{max} (\phi) = 10.95 \text{ dBi}$

$BW = 5.62 \text{ MHz}$

$\tilde{f}_{3dB} = [46.94, 52.56] \text{ MHz}$

Figure 2.3. 5-element Yagi-Uda antenna simulation with $F_c = 49.80 \text{ MHz}$. 
2.3 Front End Gain Estimation

Signal levels observed at the output of the receiving antenna typically require conditioning so that the receiver is presented with a signal level compatible with its DR. For a given operating frequency, this requires characterization of both target and noise sources. In this system, meteoroids are the target of interest and, unlike aircraft or other so-called hard targets, free electrons in the Earth’s Ionosphere are the dominant scattering mechanism, making the apparent target size variable over a large range as described in Section 2.1. The system’s minimum detectable signal is limited to noise observed at the input of the antenna, from which several sources can contribute. Having a general idea of both minimum and maximum signal levels, a target’s DR can be calculated and used to determine both RF front-end and receiver requirements. In this design, the receiver and antenna were provided, leaving only the design of the RF front-end. The primary noise contributor in the 6-meter band with a skyward-pointing antenna is cosmic noise, which is defined by various thermal sources found within the Milky Way galaxy. Cosmic noise is frequency dependent and specified by noise temperature in Kelvin. Using [16], a noise temperature of approximately 3000 Kelvin was chosen. Converting temperature to decibels is given by

$$S_{\text{min}} = 10 \log_{10} kT_a B$$  \hspace{1cm} (2.12)

where $k$ is Boltzmann’s constant, $T_a$ is the antenna noise temperature in Kelvin, and $B$ is the bandwidth of the signal. Using the antenna’s simulated bandwidth of 5.62 MHz
is used, the resulting noise floor in dBm is approximately

\[ S_{\text{min}} \approx -96.30 \text{ dBm}. \]

The maximum signal level is estimated using the \textit{overdense} meteoroid model in Equation (2.11). Using a transmit peak power of 30 kW, a simulated antenna gain of 10.95 dBi, maximum electron density of \(1 \times 10^{18}\) electrons per meter\(^3\), and a nominal height of 100 km, the estimated maximum becomes

\[ P_{\text{max}} = 1.6 \times 10^{-11} \times 30 \times 10^3 (10.95)^2 \sqrt{1 \times 10^{18}} \left( \frac{6}{100 \times 10^3} \right)^3 \text{ W}. \]

\[ S_{\text{max}} = 10 \log_{10}(P_{\text{max}}) + 30 \approx -49 \text{ dBm}. \]

The target’s dynamic range is given by

\[ DR_{\text{tgt}} = S_{\text{max}} - S_{\text{min}} = -49 - (-96.30) = 44.30 \text{ dB}. \]

Using the datasheet [1] for the receiver’s ADC, combined with the stated input impedance, the reported Signal-to-Noise Ratio (SNR) and full-scale input voltage can be used to determine the receiver’s full-scale input power and DR, given by

\[ P_{FS} = 10 \log_{10} \left( \frac{V_{pk}^2}{2z} \right) = -20 \log_{10} (10) + 30 = 10.00 \text{ dBm}, \quad (2.13) \]

where \(V_{pk}\) is the peak input voltage and \(z\) is the receiver’s input impedance, set to 50 ohms. The minimum signal in an ADC can be determined using a number of different
methods, each of which account for varying levels of noise produced internally. The ADC’s stated SNR is a value provided by the manufacturer, and is derived by testing a statistically significant number of components. Using the ADC’s stated SNR of 64.2 dB as the effective DR, it is more than adequate to accommodate the target’s calculated DR of 47.30 dB. Given the available headroom, the goal of the gain calculation is to raise the signal’s target level to a level compatible with the ADC’s input as illustrated in Figure 2.4.

![Diagram](image)

Figure 2.4. Antenna input level compared with compatible ADC level with no RF front end gain in present.

An optimal gain value can be computed by applying a guard buffer to the upper-end of the ADC’s input range and comparing with the target’s expected maximum level,
given by

\[ G_{\text{max}} \leq P_{\text{max}}^{ADC} - \alpha_{\text{max}} - S_{\text{max}}, \]  

(2.14)

where \( \alpha_{\text{max}} \) is the guard buffer level. Using \( \alpha_{\text{max}} = 1.0 \) results in

\[ G_{\text{max}} \leq 10.0 - 1.0 - (-52.0) = 61.0dB. \]

Applying Section 2.3 to Figure 2.4 is shown in Figure 2.5 and illustrates how the gain is now raised to a level that is now compatible with the ADC.

Figure 2.5. Antenna input level compared with compatible ADC level with no RF front end gain in present.

### 2.4 Front End Design

As determined in Section 2.3, a gain of 61 dB is required to raise the target’s DR to a level compatible with the receiver. In addition to gain, the system’s Noise Figure (NF) is another important parameter that can negatively affect the signal’s SNR as it
raises the apparent noise floor, potentially masking smaller targets. The receiver’s NF, like the antenna, is defined by its effective noise temperature in Kelvin. When working with digital receivers, the NF can be derived [18] from the ADC’s SNR, which is typically stated in the manufacturer’s datasheet. SNR is defined as

\[
SNR = 20 \log_{10} \left( \frac{S}{N} \right) = 20 \log_{10} \left( \frac{V_{FS}^{(rms)}}{V_{N}^{(rms)}} \right),
\]

(2.15)

where \( V_{FS} \) is the ADC’s full-scale input voltage (i.e. peak-to-peak) and \( V_{N} \) is the noise voltage. Solving Equation (2.15) for noise voltage

\[
V_{N}^{(rms)} = \left( \frac{V_{FS}^{(rms)}}{2} \right) \frac{\left( 10^{-SNR} / 10 \right)}{2} = 436 \times 10^{-6} \text{ (V)},
\]

(2.16)

results in a noise power level of

\[
P_{N} \bigg|_{r=50} = 10 \log_{10} \left( \frac{V_{N}^{(rms)}}{r} \right)^{2} + 30 = -54.2 \text{ (dBm)}.\]

(2.17)

From a component’s perspective, the NF is a ratio of the output noise power compared with the noise power present at the input. Noise power present at the input of the receiver is given by

\[
P_{\text{Therm}} = 10 \log_{10} (kT_{0}B) + 30 \text{ (dBm)},\]

(2.18)

where \( T_{0} \) is the receiver temperature in Kelvin. Using the typical value of \( T_{0} = 290 \text{K} \) and the anti-aliasing filter’s bandwidth of 9.00 MHz, the receiver’s input noise power is
approximated by

\[ P_{Therm} = -104.43 \ (dBm) \] (2.19)

The receiver’s ADC contributes noise defined by Equation (2.16), and the overall receiver NF can be defined by

\[ NF_{ADC} = P_N - P_{Therm} = -54.2 - (-104.43) \approx 50.2 \ (dB). \] (2.20)

In Chapter 5, the receiver’s gain is measured using a known input carrier tuned to the 6-m wavelength, resulting in a gain of

\[ G_{ADC} = 20 \times \log_{10} \left( \frac{271\ (steps)}{428\ (steps)} \right) = -3.9694 \ (dB). \] (2.21)

2.4.1 Front-end Component Parameters

A block diagram of the front-end instrumentation is provided in Figure 2.6 and a detailed description of each component is given in Table 2.2.
Figure 2.6. Block diagram describing the analog RF front-end signal conditioning and amplification.

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
<th>Gain (dB)</th>
<th>NF (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNA1</td>
<td>Advanced Receiver Research P50VDG</td>
<td>24.0</td>
<td>0.50</td>
</tr>
<tr>
<td>Trans. Line</td>
<td>100' RG-213 Coaxial Cable</td>
<td>-1.6</td>
<td>1.6</td>
</tr>
<tr>
<td>BiasTee</td>
<td>MiniCircuits ZFBT-282-1.5A DC Bias</td>
<td>-0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Limiter1</td>
<td>Custom Design RF Limiter</td>
<td>-0.23</td>
<td>0.23</td>
</tr>
<tr>
<td>RX Gate</td>
<td>MiniCircuits ZYSW-2-50DR RF Switch</td>
<td>-0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>BPF</td>
<td>KR Electronics KR-2867</td>
<td>-2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>ATTN</td>
<td>MiniCircuits VAT-6+ 6dB Attenuator</td>
<td>-6.0</td>
<td>6.0</td>
</tr>
<tr>
<td>LNA2</td>
<td>Advanced Receiver Research P30-1000/11VD</td>
<td>11.0</td>
<td>3.5</td>
</tr>
<tr>
<td>Limiter</td>
<td>MiniCircuits VLM-33+</td>
<td>-0.23</td>
<td>0.23</td>
</tr>
<tr>
<td>Receiver</td>
<td>USRP1</td>
<td>-3.9694</td>
<td>50.235</td>
</tr>
</tbody>
</table>

Table 2.2. Table listing RF components used for radar signal conditioning and reception.
Noise analysis of components in a cascade configuration is accomplished using the Friis Equation [24], given by

\[ F_T = F_1 + \frac{F_2 - 1}{g_1} + \frac{F_3 - 1}{g_1 g_2} + \cdots + \frac{F_L - 1}{g_1 g_2 \cdots g_L} = F_1 + \sum_{l=2}^{L} \left( \frac{F_l - 1}{\prod_{n=1}^{l-1} g_n} \right), \quad (2.22) \]

where \( F_l \) and \( g_l \) are the noise factor and gain for component \( l \) respectively, and \( F_T \) is the overall noise factor for the cascaded configuration. The system’s noise figure, in units of \( dB \), is found by taking the log of Equation (2.22), resulting in

\[ NF = 10 \log_{10} (F_T) \quad (dB). \quad (2.23) \]

When designing a cascaded component chain using Equation (2.22), the best noise performance is attained by placing high-gain, low-noise components first. In Figure 2.6, a Low-Noise amplifier (LNA) with low noise is used as a pre-amplifier to achieve a low overall NF. Applying Equation (2.22) to the RF front end in Figure 2.6, the resulting gain and NF are given in Table 2.3. Note that the total gain is relatively close to the value suggested in Section 2.3, while the overall noise figure degrades the signal by less than 1 dB. In Chapter 7, observational results will be used to compare with calculated values.

<table>
<thead>
<tr>
<th>Gain</th>
<th>Noise Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>59.57 (dB)</td>
<td>0.7188 (dB)</td>
</tr>
</tbody>
</table>

Table 2.3. Overall Gain and NF of the RF front-end component chain.
Chapter 3

Radar System Description

Meteor radars, like the one shown in Figure 3.1 operate in a pulsed mode, transmitting finite-width pulse trains at a rate referred to as the Pulse Repetition Interval (PRI). As mentioned in Section 2.1, passing meteoroids form ionized columns in the atmosphere, producing an electron density great enough such that backscattered energy from the radar’s transmitted pulse is detected by the receiver. This energy is collected by the receive antenna and represented as an analytic bandpass signal centered around the

Figure 3.1. Block diagram of the Penn State radar system.
transmitter’s carrier frequency as depicted in Figure 3.2. The bandpass signal’s instantaneous bandwidth is a combination of the transmitted waveform, antenna bandwidth, and the anti-aliasing filter in the receive path. In many systems, one or more mixer stages are used to shift the signal’s spectrum closer to baseband (i.e. origin), which has the effect of lessening requirements on subsequent components in the chain. Analytic signals are commonly used to conceptualize mathematically-tractable representations of the signal, but an observed bandpass signal is represented by the real component of the analytic representation, given by

\[
x_{bp}(t) = \mathcal{R}\{s(t)e^{j\Omega_c t}\},
\]

where

\[
s(t) = I(t) + jQ(t)
\]

Figure 3.2. Frequency and time domain representation of an analytic bandpass signal.
is the signal of interest. Combining Equations (3.1) and (3.2) produces

$$x_{bp}(t) = I(t) \cos \Omega_c t - Q(t) \sin \Omega_c t,$$

and the result is referred to as the real bandpass signal. Sampling the signal in Equation (3.3) at a uniform period $T_s$ results in

$$x_{bp}(nT_s) = x_{bp}[n] = I[n] \cos \omega_c n - Q[n] \sin \omega_c n,$$

where square brackets are used to denote the discrete form of the analog signal. Downconversion translates the signal from its center frequency $\omega_c$ to the origin, and the resulting signal is referred to as the baseband signal. Using the analytic signal from Figure 3.2, the baseband signal is a result of convolving the bandpass signal with $\delta[\omega + \omega_c]$, essentially undoing the shift introduced by the second term in the figure. An equivalent operation in the time domain is performed by multiplying the complex carrier signal by its conjugate, given by

$$e^{-j\omega_c n} = \cos \omega_c n - j \sin \omega_c n.$$

Refracting back Equation (3.3), both in-phase and quadrature channels are present, and by multiplying this signal by the sampled, complex time signal in Equation (3.5) results in

$$s'[n] = x[n]e^{-j\omega_c n} = I[n]\cos^2 \omega_c n + jQ[n] \sin^2 \omega_c n.$$
Inspection of Equation (3.6) shows that additional processing is required to retrieve the desired baseband components. Taking each of the squared cos and sin terms and placing them into their exponential form

\[
\cos^2 \omega_c n = \frac{e^{j\omega_c n} + e^{-j\omega_c n}}{2} = \frac{1}{2} + \frac{\cos 2\omega_c n}{2} \quad \text{and} \quad (3.7)
\]

\[
\sin^2 \omega_c n = \frac{e^{j\omega_c n} - e^{-j\omega_c n}}{2j} = \frac{1}{2} - \frac{\cos 2\omega_c n}{2}. \quad (3.8)
\]

one can see that multiplication by each of the squared \(\sin\) and \(\cos\) terms produces two components, given by

\[
s'[n] = \left(\frac{I[n]}{2} + \frac{I[n] \cos 2\omega_c n}{2}\right) + j \left(\frac{Q[n]}{2} - \frac{Q[n] \cos 2\omega_c n}{2}\right). \quad (3.9)
\]

In this form, both real and imaginary components contain an unwanted image at twice the carrier frequency. This unwanted image is removed with independent low-pass filters in both the in-phase and quadrature channels, resulting in

\[
I'[n] = \frac{I[n]}{2} \quad (3.10)
\]

\[
Q'[n] = \frac{Q[n]}{2} \quad (3.11)
\]

It’s important to note that each component’s power has been reduced by half. A literature search revealed that this artifact is commonly overlooked, some authors provided the correction in the mixing phase, multiplying the \(\cos\) and \(\sin\) components by 2, while others failed to mention this, likely assuming that use of relative values deemed it unnecessary.
In our system, the downconverter provides a partial correction in the implementation. This will be shown later in the chapter.

3.1 Software Defined Radar

Software Defined Radar (SDR), a term coined from the more well-known Software Defined Radio system, is a radar system with software-configurable components including but not limited to: carrier frequency, transmitted waveforms, and selectable sampling ranges. In the SDR system illustrated in Figure 3.1, the user interacts with the General Purpose Computer, programming the waveform generator, system clock, radar controller, and the receiver. Of these components, the digital receiver is most commonly associated with a software-configurable platform. Figure 3.3 depicts a typical receiver path utilizing a digital receiver. These receivers, at a minimum, perform 3 operations: 1) sampling, 2) downconversion, and 3) filtering. The Penn State radar system uses a Universal Software Radio Peripheral (USRP) digital receiver, more specifically, the USRP1. This device was chosen because of its open-source platform, as well as its popularity among hobbyists and Universities around the world. The USRP1 platform allows for complete customization of both firmware and interfacing software, making it suitable for prototyping, experimentation, and general research. In the sections that follow, core signal processing concepts related to the receiver will presented, followed by implementation details used in the USRP1.
Figure 3.3. Block diagram of the receiver path in the radar system.

3.1.1 Sampling

The *USRP1* uses the AD9862 [1] Mixed Signal Front-End Processor for both receiver sampling and signal transmission. Onboard samplers provide 1, 2, or 4 channels sampled at rates up to 64 MSPS with an SNR of 64.2 dB. Additional circuitry is provides programmable gain and decimate-by-2 low-pass filtering options. An illustration of the device’s receiver circuitry is shown in Figure 3.4. The AD9862’s front end contains a buffer for isolation and impedance control, followed by a Programmable Gain Amplifier (PGA) that can provide up to 20 dB of gain in 1 dB steps. After sampling, the device provides optional low-pass filters for additional out-of-band image rejection for signal bandwidths less than $\frac{3}{16}$ of the sampling rate. The Penn State radar system, operating at a carrier frequency 49.8 MHz and maximum instantaneous bandwidth of 5.62 MHz, is sampled directly at the maximum rate of 64 MSPS using a technique referred to as band-pass sampling [28], thereby eliminating the need for an Intermediate Frequency...
(IF) stage. In this configuration, the AD9862 processor was configured to sample the RF signal while bypassing both low-pass and hilbert filters.

![AD9862 Receiver Stage Diagram](image)

**Figure 3.4. AD9862 Receive Path Block Diagram**

### 3.1.2 Downconverter

Signal downconversion, as discussed earlier in the chapter, is the process of removing the received signal’s carrier frequency, converting the received bandpass signal into its quadrature components. In practice, the sampled bandpass signal is split into two separate channels, referred to as the in-phase and quadrature channels, which represent the real and imaginary components of the quadrature signal respectively. Using results derived in Equations (3.1) to (3.11), a baseband mixer, operating in the digital domain, is realized by multiplying the real bandpass signal by $2 \cos(\omega_c n)$ and $-2 \sin(\omega_c n)$, producing a baseband component and an unwanted higher-frequency image that is removed by subsequent low-pass filtering stages. An illustration of this operation is shown in Figure 3.5 and is referred to as a Digital Down-Converter (DDC). To better visualize
Figure 3.5. Illustration of Digital I/Q downconverter implementations.

this process, an example is given by defining two distinct in-phase and quadrature signal components, shown in Figure 3.6. Multiplying each signal component by a complex carrier signal and taking the real component produces a real bandpass signal as defined in Equation (3.1) and illustrated in Figure 3.7.
Figure 3.6. Depiction of independently generated in-phase and quadrature signals used to illustrate the process of quadrature signal downconversion.
Figure 3.7. Real bandpass signal generation using signal components in Figure 3.6 with a carrier signal frequency of 49.8 MHz.

After demodulation, the resulting unfiltered and filtered signal components are shown in Figure 3.8, respectively.
Figure 3.8. Demodulation and low-pass filtering applied to the bandpass signal shown in Figure 3.7.

3.1.2.1 CORDIC Algorithm

In practice, several hardware implementations exist [31], and, of these, the CO-ordinate Rotation Digital Computer (CORDIC) algorithm, considered the most efficient in terms of hardware resources, is implemented in the USRP1. This algorithm, typically used to calculate trigonometric and hyperbolic functions, can also approximate vector rotations, which can be exploited to perform digital demodulation. To understand the
process, begin with a complex unit vector with angle $u$

$$\mathbf{c}(u) = e^{ju} = \cos(u) + j\sin(u), \quad (3.12)$$

and rotate this vector by angle $v$, resulting in combined angle

$$\mathbf{c}(u + v) = e^{j(u+v)} = \cos(u + v) + j\sin(u + v). \quad (3.13)$$

Using the sum trig identities, $\cos(u + v)$ and $\sin(u + v)$ can be expanded into

$$\cos(u + v) = \cos u \cos v - \sin u \sin v \quad \text{and} \quad (3.14)$$

$$\sin(u + v) = \sin u \cos v + \cos u \sin v. \quad (3.15)$$

Using vector notation and labeling real and imaginary components with $x$ and $y$ respectively

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos v & -\sin v \\ \sin v & \cos v \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \quad (3.16)$$

where the initial vector is given by

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos u \\ \sin u \end{bmatrix}. \quad (3.17)$$
Using Equation (3.16), the counter-clockwise (CCW) form of a rotation matrix is defined by

\[
R = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta 
\end{bmatrix},
\]

(3.18)

where \( \theta \) represents the rotation angle in radians. The CORDIC algorithm is derived from this general definition for the rotation matrix, but several optimizations are introduced to make the algorithm suitable for use with digital-based hardware. The first optimization results in unity matrix coefficients along the rotation matrices’ diagonal, given by

\[
R = \cos \theta \begin{bmatrix}
1 & -\tan \theta \\
\tan \theta & 1
\end{bmatrix}.
\]

(3.19)

The second optimization eliminates multiplications at the expense of a multi-stage pipeline architecture. This optimization makes the algorithm suitable for hardware platforms devoid of multipliers. The algorithm, through the use of multiple stages, iteratively performs a series of subrotations that subsequently get smaller and smaller and begin to converge on the desired angle. First, the tangent functions in the rotation matrix defined in Equation (3.19) are replaced by

\[
\tan \theta[i] = 2^{-i} \text{ for } i=0,1,...,N-1
\]

(3.20)
where \( i \) represents the stage number. Making this substitution also requires some form of substitution for the \( \cos \theta \) in Equation (3.19). Applying the constraint given in Equation (3.20) in addition to unity vector length, \( \cos \theta[i] \) is given by

\[
\cos \theta[i] = \frac{1}{\sqrt{1 + 2^{-2i}}}. \tag{3.21}
\]

Figure 3.9 illustrates the origin of this term.

Figure 3.9. Illustration of a right-angle triangle with angle \( \theta \) and unity-length hypotenuse. Applying the given constraints, \( \cos \theta \) can be derived.

In this form, the modified rotation matrix

\[
R_i = \frac{1}{\sqrt{1 + 2^{-2i}}} \begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \tag{3.22}
\]
is unable to converge on a given rotation angle $\theta$ since it rotates in a single direction.

To overcome this limitation, a performance variable is introduced to track the angular error and its sign at each stage, given by

$$z[i + 1] = z[i] - \theta[i],$$  \hspace{1cm} (3.23)

where $\theta[i]$ is the angular rotation applied at stage $i$. Direction of rotation is controlled by the sign of the performance variable

$$\delta[i] = \text{sgn}(z[i]),$$  \hspace{1cm} (3.24)

and applied to the modified rotation matrix, resulting in

$$R[i] = \frac{1}{\sqrt{1 + 2^{-2i}}} \begin{bmatrix} 1 & -\delta[i]2^{-i} \\ \delta[i]2^{-i} & 1 \end{bmatrix},$$  \hspace{1cm} (3.25)

for each subrotation matrix. After $N$ rotations, the resultant vector will converge to the desired $\theta$, with an overall angular error given by

$$\epsilon = \theta - \sum_{i=0}^{N-1} \delta[i] \tan^{-1}(2^{-i}),$$  \hspace{1cm} (3.26)

stating that accuracy is directly proportional to the number of stages implemented. In addition to the resultant vector’s direction, the scaling factor in Equation (3.21) must be accounted for. The most common method combines all factors into a single value,
given by
\[ K = \prod_{i=0}^{N-1} K[i] = \prod_{i=0}^{N-1} \frac{1}{\sqrt{1 + 2^{-2i}}}, \]  
(3.27)

and applies the correction either pre- or post-process, with each approach posing different constraints on the hardware. Ignoring the scaling factor for a moment, each subrotation is produced using a transformation matrix given by
\[
\begin{bmatrix}
x[i + 1] \\
y[i + 1]
\end{bmatrix} =
\begin{bmatrix}
1 & -\delta[i]2^{-i} \\
\delta[i]2^{-i} & 1
\end{bmatrix}
\begin{bmatrix}
x[i] \\
y[i]
\end{bmatrix}.
\]  
(3.28)

To summarize, a stage is implemented using 3 equations, defined by
\[
\begin{align*}
z[i + 1] &= z[i] - \theta[i] \\
\theta[i] &= \delta[i]\tan^{-1}(2^{-i}) \\
\delta[i] &= \text{sgn}(z[i]),
\end{align*}
\]  
(3.29, 3.30, 3.31)

and pipelining multiple stages produces the desired result, with the number of stages determining the overall accuracy. Figure 3.10 illustrates a 12-stage CORDIC algorithm operating in vector mode, where a desired angle of \(\pi/3\) is given and the output of each stage is shown, illustrating the algorithm’s convergence behavior.
Figure 3.10. Illustration of a 12-stage CORDIC algorithm iteratively converging on the given angle of $\frac{\pi}{3}$ radians.

As defined in Equation (3.5), demodulation of a signal requires generation of a conjugate carrier frequency, and its instantaneous phase angle must be input into the CORDIC module at the system’s sample rate. In hardware, this downconverter’s tuning frequency is derived directly from the system clock using a programmable divisor to approximate the desired frequency. Applying this variable-rate clock concept to an N-bit accumulator, the overflow rate represents the conjugate carrier’s period, and the
angular rate can be visualized as a *phase wheel*, as shown in Figure 3.11. Using phasor terminology, the rate at which the phase vector traverses the wheel is given by the number of revolutions per second, with units of Hertz (Hz). Implementing the *phase wheel* in a digital system requires discrete steps, defined by a *step size*, to traverse the wheel. In a system with a fixed clock rate $f_{\text{clk}}$ applied to the phase traversal mechanism, the *step size* controls the output rate $f_{\text{out}}$ given by

$$f_{\text{out}} = f_{\text{clk}} \frac{M}{2^N}, \quad (3.32)$$

where $M$ is the *tuning word* and $N$ defines the accumulator’s bit width. A hardware representation of this component is provided in Figure 3.12.

![Figure 3.11. Illustration of the phase wheel concept used to describe phase generation of the demodulating frequency used by the DDC in hardware.](image)

$\Delta r = \text{step size}$

$M = \text{tuning word}$

$f_{\text{clk}} = \text{clock rate}$

$f_{\text{out}} = \text{output rate}$
In the original USRP1 implementation, a 12-stage pipelined CORDIC algorithm was used with 16-bit input and output precision and a fixed-width performance variable. To accommodate the scaling factor, 2 guard bits were added internally to prevent overflows. As an enhancement to the original application, a sizable reduction in hardware resources for the CORDIC module was achieved by:

1. Use of an explicit module to represent the shift adder stage, resulting in inferred 
   addsub primitives available in the FPGA fabric, and

2. 1-bit reduction in each subsequent stage’s performance variable (i.e. each stage 
   reduces the error by at least a factor of 2).

As in the original implementation, an output bit was trimmed from the left- and right-
most bits, effectively reducing the gain incurred by a factor of 2, resulting in an overall 
gain of \( A_v = 1.6468/2.0 = 0.82338 \), or a loss of

\[
L_{\text{CORDIC}} = -20 \log 10(A_v) = 1.6880 \text{ dB.} \quad (3.33)
\]
3.1.3 Cascaded Integrator Comb (CIC) Filter

If an incoming signal is sampled at a rate greater than the bandwidth of the information contained in the signal, the data rate can be reduced to match the signal’s bandwidth using a multirate signal processing technique known as decimation [12], which has the advantage of decreasing the system’s digital filter requirements. One type of low-pass filter, referred to as the Cascaded Integrated Comb Filter (CIC), can efficiently filter and decimate the signal rate simultaneously. In this chapter, a complete analysis of the CIC filter will be given, including optimizations to further reduce size requirements.

The most intuitive way to understand and derive the operation of the CIC filter begins by defining the basic Finite Impulse Response (FIR) filter. A standard low-pass FIR filter is described mathematically by

$$y[n] = \sum_{k=0}^{N-1} b[n]x[n-k], \quad (3.34)$$

where $b[n]$ are the filter’s coefficients, $x[n]$ is the input signal, and $y[n]$ is the output signal. The filter’s response, or spectral shape, is defined by the coefficients. Implementation in hardware requires storage registers for each of the coefficients, an equal number of multipliers, and an accumulator wide enough to prevent data overflow; alluding to the realization that this filter quickly becomes prohibitive in resource-limited systems.

By sacrificing control of the filter response, an immediate improvement in resources is gained by setting all coefficients to 1.0, effectively eliminating the need for coefficient storage and multipliers. In the time domain, this filter has a flat, rectangular shape, and...
is referred to as a boxcar filter, given by

\[ y[n] = \sum_{k=0}^{N-1} x[n - k], \quad (3.35) \]

with a time and frequency response illustrated in Figure 3.13. This filter’s response in

\[ Y(z) = \sum_{k=0}^{N-1} X(z) z^{-k} \implies H(z) = \sum_{k=0}^{N-1} z^{-k}. \quad (3.36) \]

Figure 3.13. 16-Tap Boxcar filter example illustrating the filter’s input response in both time and frequency domains. Note that the filter’s frequency response has been normalized for display purposes.
Examining the structure of the resulting transform reveals the converged form of a finite Geometric Series [23], given by

\[ a + ar + ar^2 + \ldots + ar^{n-1} = \sum_{k=0}^{N-1} ar^k = a \frac{1 - r^N}{1 - r}. \]  

(3.37)

Comparing Equations (3.36) and (3.37), equivalence can be made by setting \( a = 1 \) and letting \( r = z^{-1} \), resulting in

\[ H(z) = \frac{1 - z^{-N}}{1 - z^{-1}}. \]  

(3.38)

Rearranging Equation (3.38), the system’s zeros and poles are given by

\[ H(z) = \frac{1}{z^{N-1}} \frac{z^N - 1}{z - 1}. \]  

(3.39)

Equation (3.39) shows that stability requires pole cancellation by one of the zeros in the numerator. The precision required to achieve such cancellation forbids its use in floating or fixed-point implementations. Treating the numerator and denominator in Equation (3.38) independently, the system can be divided into an 1) integrator stage

\[ H_I(z) = \frac{1}{1 - z^{-1}} \]

\[ y[n] = x[n] + y[n - 1], \]  

(3.40)
and a 2) comb (i.e. differentiator) stage

\[ H_C(z) = 1 - z^{-N} \]

\[ y[n] = x[n] - x[n - N]. \]  

Linear Time-Invariant (LTI) systems, such as this, can be separated into sub-blocks given by Equations (3.40) and (3.41). Additionally, the commutative property applies, so block ordering is not important. However, in the case of the CIC filter, the system contains a zero that cancels the system’s only pole in order to maintain stability, and applying the commutative property fails. To work around this limitation, the Two’s complement number system, with its wrap-around overflow behavior, must be used.

Two’s Complement is a signed number representation defined by

\[ N = -a_{b-1}2^{b-1} + \sum_{i=0}^{i=b-2} a_i2^i, \]  

where \( b \) is the total number of bits used to define \( N \). In this system, overflows are recoverable due to the circular nature of the system. An illustration of this is depicted in Figure 3.14. Using Two’s Complement, the filter can be synthesized by cascading blocks together as shown in Figure 3.15; thereby justifying the filter’s alternate name, the Cascaded Integrator Comb (CIC) filter. In the author’s original paper [14], the comb filter’s delay is defined by letting \( N = RM \), where \( R \) is the decimation value and \( M \) is referred to as the differential delay. The differential delay is used to decouple the
Figure 3.14. Illustration of a single-stage, 4-bit CIC filter with $R = 1$ and ramp input applied. Although the filter’s internal state is unstable (i.e. overflow), the output of the system is correct. Using Two’s Complement arithmetic, the internal overflow has wrap-around behavior and the integrator’s addition is undone by the differentiator’s subtraction, thereby appearing stable when observing the system as a black box (i.e. observing only at the input and output of the system).

Figure 3.15. Illustration of downsampling-by-$R$ applied prior to the comb stages. Justification is provided by the Noble Identities [12]. The resulting structure at the bottom of this Figure is referred to as the Cascading Integrator Comb (CIC) filter.
decimation rate from the system’s frequency nulls. In this system, we assume $M = 1$ and use only $R$ to refer to the system’s decimation rate. As shown in Figure 3.13, a single-stage CIC filter’s response is a sinc function, and frequency nulls are a function of the system’s sampling rate. Starting with Equation (3.38) and substituting $R$ to represent decimation, the system’s frequency response can be examined using

$$P \left( e^{j\omega} \right) = H \left( e^{j\omega} \right) H^* \left( e^{j\omega} \right).$$

(3.43)

Analysis of Equation (3.43) becomes more intuitive if modified and placed in a trigonometric form by

$$H \left( e^{j\omega} \right) = \left( 1 - e^{-j\omega R} e^{j\omega \frac{R}{2}} \right) \left[ \frac{1}{(1 - e^{-j\omega}) e^{j\omega \frac{R}{2}}} \right] = \frac{\sin \omega \frac{R}{2}}{\sin \frac{\omega}{2} e^{j\omega \frac{1-R}{2}}}.$$  

(3.44)

The output response of the filter will operate at the decimation rate of $f/R$, providing an N-stage power response of

$$P \left( e^{j\omega} \right) = \left| \sin \frac{\omega}{\sin \frac{\omega}{R}} \right|^{2N}.$$  

(3.45)

The filter’s maximum gain is realized as the input frequency approaches the filter’s pole

$$G_{max} = \lim_{\omega \to 0} \left| P \left( e^{j\omega} \right) \right|^{\frac{1}{2}} \Rightarrow \left| \frac{R \cos \omega}{\cos \omega} \right|^N \bigg|_{\omega = 0} = R^N.$$  

(3.46)

Maintaining stability requires that the filter’s internal registers account for the maximum gain calculated in Equation (3.46). The system’s maximum bit width must accommodate
the data’s input width and the system’s maximum gain, resulting in a required register width of

\[ w_{\text{max}} = \lceil \log_2 G_{\text{max}} \rceil = \lceil N \log_2 R \rceil. \]  

(3.47)

If programmable decimation is used, Equation (3.47) varies with \( R \) and a Lookup Table (LUT) must be used to normalize the filter’s gain with decimation values. Also note that decimation values that are not a power of 2 will introduce an error given by

\[ \epsilon_w = w_{\text{max}} - N \log_2 R. \]  

(3.48)

In addition to gain error, bit truncation at the filter’s output is another source of error. As an example, consider a 5-stage CIC filter utilizing an input/output width of 16 bits and a decimation rate of 128. This system will require a register width of 51 bits across all internal stages. With an output width of 16 bits, an additional output stage is needed to select the upper 16 bits from the final comb stage to ensure that the proper output level is achieved. Truncation error is typically modeled under the following assumptions:

i) the error is white,

ii) all sources of error are independent, and

iii) the error is uniformly distributed.

Figure 3.16 illustrates the distribution of truncation error. Using these assumptions, it is feasible to take the overall truncation error and distribute it evenly across each of the system’s sub-blocks [14], using a method referred to as Hogenauer Bit Pruning (HBP). Although the overall error remains the same in both implementations, HBP
can greatly reduce the hardware implementation’s footprint by reducing the filter’s data path bit width prior to the final stage. This method is implemented by first determining the total error given by the number of bits discarded in the standard implementation’s output stage. Assuming a uniform distribution, the width of the error’s probability density function (pdf) is given by

\[ E_j = 2^{B_j} \quad \text{for} \ j = 0, 1, ..., 2N - 1 \quad , \quad (3.49) \]

where \( B_j \) represents the number of bits pruned at stage \( j \). Referring to Figure 3.16, both mean error and variance for each stage can be determined. The total number of

![Figure 3.16. Error statistics given J bits discarded from a filter stage when assuming uniformly distributed error.](image)
bits discarded by the filter is represented by

\[ b_d = b_{in} + w_{max} - b_{out}, \]  

(3.50)

where \( b_{in} \) is the input width, \( b_{out} \) is the output width, and \( b_d \) is the total number of bits discarded by the output stage. Mean error and variance from bit truncation in each stage is given by

\[ \mu_j = \frac{\epsilon_j}{2}, \quad \text{and} \]
\[ \sigma_j^2 = \frac{\epsilon_j^2}{12}, \]

(3.51)

respectively. Determination of the output error contributed by each sub-block in the filter can be modeled by injecting noise into each sub-block’s input and using the resulting system function to analyze the system’s output. Using the standard form of the z-transform

\[ H(z) = \sum_{k=0}^{\infty} h[k]z^{-k}, \]

(3.52)

combined with assumptions previously mentioned, each impulse response coefficient \( h[k] \) is considered an independent noise contributor, therefore finding the coefficient values \( h[k] \) will provide the error produced by each sub-block in the system. Resulting mean error and variance due to each sub-block \( j \) is given by

\[ D_j = \begin{cases} 
\sum_k h_j[k], & \text{for } j = 0, 1, ..., 2N - 2 \\
1, & \text{for } j = 2N - 1 
\end{cases} \]

(3.53)
and

\[ F_j = \begin{cases} 
\sum_k h_j^2[k], & \text{for } j = 0, 2, ..., 2N - 2 \\
1, & \text{for } j = 2N - 1 
\end{cases} \]  

(3.54)

respectively. Starting from the first integration stage and working towards the output, the system response for each source of error is given by

\[ H_j(z) = \begin{cases} 
H^1_j(z) = H^N_j(z)H^i_j(z), & \text{for } j = 0, 2, ..., N - 1 \\
H^2_j(z) = H^{2N-j}_j(z), & \text{for } j = N, N + 1, ..., 2N. 
\end{cases} \]  

(3.55)

\( H^1_j(z) \) from Equation (3.55) can be rewritten in a more intuitive form given by

\[ H^1_j(z) = \left( \frac{1 - z^{-R}}{1 - z^{-1}} \right)^N \equiv \left( \frac{1 - Z^{-R}}{1 - Z^{-1}} \right)^{N-j} \left( 1 - z^{-R} \right)^k, \]  

(3.56)

which can be viewed as \((N-j+1)\) boxcar system cascaded with \((k-1)\) comb filters. Since each impulse response coefficient is considered an independent contributor, a response for each error source \(k\) can be computed using

\[ H^1_j(z) = \left( \sum_{l=0}^{R-1} h_j(l)z^{-l} \right)^{N-j} * \left( \sum_{m=0}^{R} h_j(m)z^{-m} \right)^k. \]  

(3.57)
Treating each term in Equation (3.57) individually, the total number of coefficients $h_j$ required to represent the system’s error response for the $k^{th}$ stage can be determined by

$$H^a_j(z) = \left( \sum_{l=0}^{R-1} h_j(l)z^{-l} \right)^{N-j} \implies (R-1)(N-j) \quad \text{polynomial order}$$

$$H^b_j(z) = \left( \sum_{l=0}^{R} h_j(l)z^{-l} \right)^{k} \implies Rk \quad \text{polynomial order},$$

producing a combined response given by the form

$$H^l_j(z) = H^a_j(z) \ast H^b_j(z) = \sum_{k=0}^{N(R-1)+j} h_j[k]z^{-k}.$$  \hspace{1cm} (3.59)

Coefficient values for Equation (3.58) are computed using binomial expansions of the form

$$(a + b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-j} b^k, \quad \text{and}$$

$$\frac{1}{(a + b)^n} = \sum_{k=0}^{n} (-1)^k \binom{n+k}{k} a^{n-j} b^k.$$  \hspace{1cm} (3.60)

Applying expansions in Equation (3.60) to Equation (3.58) results in

$$H^l_j(z) = \frac{(1 - z^{-R})^N}{(1 - z^{-1})^{N-j}} \sum_{l=0}^{N} \binom{N}{l} (-1)^l z^{-Rl} \cdot \sum_{m=0}^{\infty} \binom{N-j+m-1}{m} z^{-m}$$

$$= \sum_{m=0}^{\infty} \sum_{l=0}^{N} (-1)^l \binom{N}{l} \binom{N-j+m-1}{m} z^{-m+Rl}.$$  \hspace{1cm} (3.61)
Making a variable substitution of $k = Rl + m$, Equation (3.61) can be rewritten as

$$H^1_j(z) = \sum_{k=0}^{N(R-1)+j} \sum_{l=0}^{\lfloor \frac{k}{R} \rfloor} (-1)^l \binom{N}{l} \binom{N - j + k - Rl - 1}{k - Rl} z^{-k}. \quad (3.62)$$

where the upper bound on $l$ derives from the binomial coefficient bound $k - Rl > 0$, while $k$ is bound by the polynomial size determined in Equation (3.59). Using Equation (3.59) as a reference, the coefficient values are directly calculable. Similarly, coefficients for $H^2_j(z)$ are computed using the first equation in Equation (3.60), resulting in

$$H^2_j(z) = \sum_{k=0}^{2N-j} (-1)^k \binom{2N-j}{k} z^{-kR}. \quad (3.63)$$

Given coefficients that are directly computable through the equations provided above, a method for distributing total truncation error, stated in Equation (3.51), equally across each stage is obtained by

$$\sigma^2_j F_j \leq \frac{\sigma^2_T}{2N}. \quad (3.64)$$

where $\sigma^2_T$ is the variance resulting bit truncation in the output stage. Substituting variables and using a common base results in

$$2B_j + 2 \log_2 F_j - \log_2 12 \leq 2 \log_2 \sigma_T - \log_2 2N. \quad (3.65)$$

Rearranging Equation (3.65) and solving for $B_j$

$$B_j = \left[ \log_2 \sigma_T + \frac{1}{2} \log_2 \frac{6}{N} - \log_2 F_j \right]. \quad (3.66)$$
provides the user with the number of bits to truncate at stage $j$.

### 3.1.3.1 Implementation

In the USRP1’s original implementation, a 4-stage design was used, utilizing a programmable decimation value ranging from 8 to 128 and an input/output bit width of 16. Bit pruning was not used, therefore a fixed internal register width was computed using Equation (3.47), resulting in

$$b_{max} = \lceil N \log_2(R_{max}) \rceil + b_{in} = 44 \text{ bits.}$$  \hspace{1cm} (3.67)

Although efficient when compared with a standard FIR filter, bit-pruned implementations will fare better in both resource utilization and speed given that the synthesis tools will have more freedom to optimize when given more space. Several variations were implemented and compared, including:

(a) fixed-width, 4-stage implementation,
(b) bit-pruned, 4-stage implementation,
(c) bit-pruned, 5-stage implementation,
(d) maximum fixed-decimation, bit-pruned, 5-stage implementation, and a
(e) minimum fixed-decimation, bit-pruned, 5-stage implementation.

Each of the above mentioned designs were synthesized on the USRP1’s Altera Cyclone I device using Quartus Version 10.1sp1 software. Bit-pruning algorithms were modified
to accommodate variable decimation in items (b) and (c), while items (d) and (e) utilized fixed decimation values to illustrate potential for minimizing the resource footprint when used in designs that do not require dynamic decimation control. Justification for the latter case can be made for the current design since the FPGA’s image is loaded at system startup, and the user is free to load an image containing the desired decimation.

Table 3.1 illustrates resource utilization of each design. Note that each result in Table 3.1 is calculated before executing the FPGA fitter phase, which optimizes the existing design based on chosen settings. For example, design (c), after fitting, takes up 96.7% of the devices available resources, whereas the preliminary design shows resource usage greater than 100 percent.

The USRP1 system used a 4-stage CIC filter with resource usage similar to (a), which provides reasonable image rejection and fits comfortably in a 4-channel setup. With bit pruning employed, the system can accommodate a 5-stage CIC filter while maintaining 4 receive channels. The additional stage increases image rejection by $\approx 8$ dB at a normalized output frequency of 0.80 when comparing 4-stage and 5-stage filters in Figures 3.17 and 3.18 respectively. Of the designs presented, option (c) was used in the final design.

<table>
<thead>
<tr>
<th>version</th>
<th>Logic Cells</th>
<th>LC Registers</th>
<th>Total Elements (pre-fitter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>749</td>
<td>556</td>
<td>11336/12060</td>
</tr>
<tr>
<td>(b)</td>
<td>662</td>
<td>484</td>
<td>10575/12060</td>
</tr>
<tr>
<td>(c)</td>
<td>861</td>
<td>677</td>
<td>12230*/12060</td>
</tr>
<tr>
<td>(d)</td>
<td>560</td>
<td>445</td>
<td>9826/12060</td>
</tr>
<tr>
<td>(e)</td>
<td>490</td>
<td>377</td>
<td>9266/12060</td>
</tr>
</tbody>
</table>
Figure 3.17. 4-stage CIC filter response
Figure 3.18. 5-stage CIC filter response
To eliminate gain approximation errors described in Equation (3.48), a bounded subset of programmable decimation values, given by

\[ \{d : d = 2^n \text{ for } n = 3, 4, \cdots, 7\}. \]

(3.68)

One disadvantage of increasing the filter order is the increased droop in the frequency response. This effect is minimal when a fraction of the normalized bandwidth is required, but quickly becomes unacceptable as bandwidth requirements increase. The standard method for correcting this response is to insert a FIR compensation filter into the system. However, given the lack of resources, this was not an option. For users requiring compensation, a compensating filter was designed for post-collection use as needed.

3.1.4 CIC Compensation Filter

As mentioned in Section 3.1.3, the CIC’s droop response becomes unacceptable when attempting to utilize a larger percentage of the available bandwidth. To improve the response, a secondary compensation filter can be added to the output of the CIC. Approximating the CIC filter’s rolloff as an \(N^{th}\)-order sinc function, a technique referred to as the frequency sampling method [21] can be used to design a compensating filter by taking the inverse of this response out to the desired bandwidth, which must be less than half the CIC filter’s output rate as dictated by the aliasing regions outlined in [14]. This design requires a piecewise function consisting of three distinct regions: the 1) passband region, the 2) transition region, and the 3) cutoff region. After selecting the samples in the frequency domain, and inverse FFT is used to produce filter coefficients in the time
domain. Using an application note provided by [7], a compensating filter design program was coded using Python and the final design was chosen using a symmetric, 31-tap filter with coefficients given by

$$c = [$$

105, −189, 263, −257, 35, 478, −1204,

1809, −1785, 616, 1884, −5284, 8195, −7820, −1632,

32767, −1632, −7820, 8195, −5284, 1884, 616, −1785,

1809, −1204, 478, 35, −257, 263, −189, 105

$$]$$

producing a normalized cutoff frequency of $\approx 0.35$. Figure 3.19 depicts the original cic response, the compensating filter’s response, and the resulting compensated output. Ideally, this filter would be implemented directly in hardware, but, given the resource limitations of the Cyclone I FPGA, this was not achievable. This filter can, however, be provided to the user as an optional post-processing stage. Ultimately, the use of the filter depends on the user’s bandwidth requirements. In our application, the system is used to observe residual meteor trails, where required bandwidth is derived from the trail’s induced doppler, whose primary component is high-altitude winds, which measure upwards of 100 m/s, corresponding to an approximated doppler shift of

$$f_d = \frac{2w_s f_c}{c},$$

(3.70)
Figure 3.19. Illustration of the compensation filter’s ability to flatten out the drooping response of the CIC filter.
where $w_s$ is the wind speed in m/s, $f_c$ is the radar’s transmitting frequency in Hz, and $c$ is the speed of light in m/s. Using a 6-m band with maximum wind speeds of 100 m/s, an expected maximum doppler shift of 33 Hz. Given a typical output bandwidth of 1 MHz, the normalized bandwidth is approximately $33 \times 10^{-6}$, providing negligible droop, leading to the conclusion that the filter will not provide any improvement for this application, but could prove useful for observations requiring utilizing a larger percentage of available bandwidth.

### 3.1.5 Pulse Synchronization and Data Tagging

The original USRP1 firmware design sampled data continuously, with no concept of synchronization at the firmware level. Our system operates in a pulsed mode, which requires synchronization with the transmitted pulse so that accurate ranging information can be provided. Additionally, a receive window was used, enabling the user to define the range at which the receiver begins sampling as well as the number of range samples to collect. Both features were provided by introducing a gate enable input signal into the receiver. When the gate enable signal is high, the receiver is enabled for sampling. During implementation, it was discovered that, even with the gating feature, synchronization problems were present in the output data stream. Further investigation revealed that the USRP1’s internal USB FIFOs on the dedicated USB controller chip were not flushed properly when issued a reset command. As a work-around, a data tag was inserted into the output stream, allowing pulse synchronization to take place in software when the system first starts collecting data. All of these features were added by replacing the original FIFO module in firmware with a custom version, as illustrated in Figure 3.20.
and related signals are shown in Figure 3.21 to illustrate range selection and the tagging process.

Figure 3.20. Custom FIFO implementation in *USRPI* firmware to provide pulsed mode operation
Figure 3.21. Signals used in system for pulsed mode operation
Chapter 4

Software Architecture and Design Implementation

The Penn State radar system was designed to provide a simple, configurable interface through which the system could be easily setup using commonly-used modes, while providing the flexibility to introduce new modes of operation as needed. Requirements for the design included:

1. the system must be capable of running several days without user intervention;
2. remote control and monitoring was required;
3. data storage must handle raw I/Q samples with up to 8-MHz instantaneous bandwidth per receiver unit;
4. the receiver interface architecture should be decoupled so that core software components are unaffected by new receiver implementations;
5. open source software and platform-independent tools should be used when possible to promote collaboration.

Being able to operate the system remotely requires a two-part software design: 1) an interface through which a user can interactively control the system and retrieve status, and 2) an interface that talks directly to the hardware; effectively collecting, processing, formatting, and storing the raw data samples. In practice such a design is referred to as a client-server pattern [6]. In this pattern, the Client and Server components each
represent an endpoint connected by a medium through which messages are transported (e.g. Ethernet), providing bidirectional messaging to both endpoints. Additionally, the endpoint relationship can be a many-to-one relationship, meaning that multiple clients all connected to a single server. The implemented solution is referred to as the GnuRadar software system [26] and utilizes a light-weight client to 1) configure, 2) control, and 3) monitor the system’s health, while the server component is responsible for communicating with the hardware and performs the following tasks: 1) synchronizing and streaming data from the device into the server’s memory, 2) formatting raw data into a standardized format, and 3) archiving data for off-line processing and historical record. Figure 4.1 illustrates a block diagram of the client-server architecture. Using this design pattern, along with a combination of tools and optimizations, all requirements were met.
Figure 4.1. Block diagram illustrating client-server architecture of the system.

4.1 Communication and Messaging Protocol

Both Client and Server objects communicate through a set of well-defined rules known as a communication protocol, which is generally provided by a library and/or language common to both objects; and this design permits each object to function without knowledge of the communication medium, effectively decoupling the Client and Server from the chosen communication network. In this design, the Server, due to the existing USRP1 software interface, was confined to the Linux Operating System and the C++ programming language. The Client, designed to control and monitor the Server, was not bound by OS platform or language, and, to make the system more readily available,
the Java programming language was chosen in an attempt to make the Client platform independent. Mixing programming languages between Client and Server requires that the communication protocol and messaging library chosen be compatible with both C++ and Java. After an exhaustive search for open source libraries capable of handling this task, a combination of two libraries were chosen: the 1) ØMQ [13, 15] library was used to communicate messages, and the 2) Google Protocol Buffers library provided the necessary messaging protocol between Client and Server. A brief overview of each library is provided in the sections that follow.

4.1.1 ØMQ

ØMQ is a small, efficient messaging library with implementations provided in several languages [15]. This library loosely resembles a basic TCP/IP socket but provides a number of features typically found in heavier-weight commercial packages [25]. In addition to its feature-rich capabilities, the library’s open source implementation, excellent documentation, and popularity were all factors resulting in its selection for use in this system. Client and Server messaging comprises both control and status messages, where control messages are sent using a request-reply [13] messaging pattern, while status messages use a publish-subscribe [13] pattern to periodically publish status to any listening subscribers. A diagram describing each pattern is illustrated in Figures 4.2 and 4.3.
Figure 4.2. *Client-server* architecture used in the *GnuRadar* system with 1 or more clients communicating with a single server using a *request-reply* message pattern.

Figure 4.3. *Publish-subscribe* message pattern used in the *GnuRadar* system with N subscribers and a single publisher.
4.1.2 Google Protocol Buffers

ProtoBufs [10] is an open source Google project that provides the designer with a tool for generating language-neutral message protocols that are much more efficient than older XML implementations (3-10 times smaller and 20-100 times faster according to Google testing) by providing serialize/deserialize methods used to transmit/receive binary-based packets. First, a messaging format is designed and coded using a simple set of rules encoded in a text-based message format file referred to as a proto buffer text format. After creating the rules, the protocol buffer file is passed to a code generating tool where the desired programming language is entered as a parameter. The output contains source code files in the specified language and are included in with the project’s source files. Expanding and/or redefining the existing rule set requires only an update to the protocol buffer file and regeneration of source files. The same principle also applies to newer version of the library, making updates and maintenance quite easy when compared with a custom implementation. In sections that follow, the rules used for the GnuRadar’s messaging format will be provided.

4.2 Client Architecture

The Client provides an interface for connecting to and communicating with the Server. Typically, this component is considered light-weight (i.e. provides little processing capabilities) and, ideally, platform independent. The GnuRadar client-side software two parts: 1) a configuration tool, and 2) a run-time control tool. Both programs were
developed with the Java programming language, primarily for its platform independence and integrated Graphical User Interface (GUI) libraries.

### 4.2.1 Configuration Tool

The Server, which controls operation of the radar’s receiver, requires a valid configuration to enable the system. A valid system configuration is created using the GnuRadar Configuration Tool, which is GUI-based Java program providing a simple interface with automatic rule checking and correction. A screen shot of this tool is shown in Figure 4.4. Each of the framed sections in the GnuRadar Configuration Tool is given as follows.

![GnuRadar Configuration Tool Interface](image)

**Figure 4.4. GnuRadar Configuration Tool Interface**
### 4.2.1.1 General Settings

The General Settings frame controls the sampling rate, bandwidth, and number of collection channels in the system. The following rules are applied to this frame:

1. **Sample Rate**: System sample rate entered with double-precision and bounded by $[1, 64]$ MHz.

2. **Channels**: Number of channels used for data collection. Values of 1, 2, or 4 are valid. Rule checking is applied to ensure that the total system bandwidth does not exceed 8 MHz total.

3. **Decimation**: Controls the system’s bandwidth. Valid values are $\{8, 16, 32, 64, 128\}$. Rule checking is applied to ensure that the total system bandwidth does not exceed 8 MHz total.

4. **Bandwidth**: This field is read-only and derived from the combination of items Sample Rate, Channels, and Decimation.

### 4.2.1.2 DDC Settings

The DDC Settings frame controls the downconverter tuning frequency and associated phase for each of the defined channels while undefined channels are disabled. The downconverter frequency is a double-precision variable and has selectable units of Hz, KHz, or MHz. Actual precision of the frequency is limited by the frequency step size, which is given by

$$\Delta f = \frac{f_{\text{clk}}}{2^N},$$  \hspace{1cm} (4.1)
where $f_{clk}$ is the system sample rate and $N = 32$ bits. Negative numbers can be entered to upconvert a signal. Tuning phase is adjustable using a double-precision variable, although actual resolution is limited to the step size of the phase accumulator defined in Section 3.1.2.1, which is $14.9 \times 10^{-3}$ degrees. Phase units are selectable in either $degress$ or $radians$ and resolution is dependent upon the selected step size.

4.2.1.3 Data Window and Pulse Settings

In a pulsed radar system, transmit pulses of a specified width are periodically emitted with repetition rate defined by the Inter-Pulse Period (IPP), which is more commonly referred to as a Pulse Repetition Interval (PRI) in military radar systems. Some time after the transmit pulse is sent, the receiver is enabled, sampling the scattered energy for a specified time, defined by the $Start$ and $Stop$ variables provided by each defined $Data Window$. Units are selectable and include time in microseconds, distance defined in kilometers, or the number of samples to collect. The IPP is double-precision and can be defined with time units of milliseconds or microseconds; or can be defined by the number of samples collected. Multiple $Data Windows$ can be added and/or removed as needed with the requirement that at least one window is defined for proper operation. The $TX Carrier Frequency$ variable does not affect system operation, but is recorded as metadata.
4.2.1.4 HDF5 Header Information

The HDF5 Header Information frame defines metadata that is recorded when the defined configuration is loaded by the radar system. These variables are not required, but are useful when archiving data.

4.2.1.5 File Settings

In the File Settings frame, the FPGA Bit Image is used to define the FPGA image loaded into the receiver at runtime. It should be noted that this field is currently required for the USRP1 configuration, and this variable may not apply to other receivers that may be added in future designs. The Data Set Base Name provides a base name to the file set used during collection. This base file name is created at runtime and an index number follows the base name to order the set (e.g. baseName_00000001.h5).

4.2.2 Runtime Tool

The GnuRadar Run tool communicates control commands and monitors status from the server. The program is also responsible for loading and validating a configuration file. This program, shown in Figure 4.5 uses a Wizard user interface pattern [29] to perform receiver setup and control in a linear fashion. After startup, the user’s only choice is to load a configuration file using the Load button. After a configuration file has been loaded, the receiver performs validation, which collects a fixed number of radar pulses and validates the number of data channels, data window settings, and selected decimation rate. If the system fails validation, the user is instructed to correct the problem and try again. Successful validation enables the Run button, which officially
starts data collection, writing data to a data set defined in the loaded configuration file. Once the system is in a running state, system health is reported using progress bars at the bottom of the GUI, each of which reports the current read, write, and depth buffers of the running server program. The Run button, when in a running state, acts as a Stop button to stop collection as needed. It’s important to note that recorded data sets cannot be overwritten, so a repeated system start and stop will fail with a write error, instructing the user to either rename the configuration’s data set, or explicitly remove the existing file set. This prevents the user from inadvertently deleting a valid data set. Both control commands and reported status data are communicated across the network using unique port numbers, which are configured and loaded at program startup using a file named .gradarrc, located in the user’s home directory (e.g. /home/<username>).

4.3 Server Architecture

The Server component in the GnuRadar system is considered the critical component in the system in terms of design, requiring a moderate rate of data streaming, real-time formatting, and demanding storage requirements due to the collection of raw
I/Q data samples. The Server is written in C++ and is started as a background daemon that listens for incoming commands from a connected client using a configurable network address and port. The receiver, streaming 32-bit, complex data samples, places continuous data rates of up to 32 MBPS that must be buffered in memory by the Server. To handle these rates, a producer-consumer design pattern is used, as shown in Figure 4.6. This pattern uses two threads simultaneously; a ProducerThread, responsible for buffering streaming data coming from the receiver; and a ConsumerThread, who reads the buffered data from memory, formats the data, and write the samples to disk. The success of the pattern relies on the ability of the ConsumerThread to read and process data from the buffered location at a rate much greater than the ProducerThread produces it. A circular buffer is used to manage the data, with ProducerThread writes marking buffers as dirty and advancing the head of the buffer, while the ConsumerThread marks the buffers as clean, and advances the tail of the circulating buffer. Each write increments a depth variable to track the number of dirty buffers, while each read decrements it. Overflows occur when the depth variable surpasses the total number of buffers in the system. Overflows can be prevented by adjusting the total number of buffers to absorb scheduling uncertainties and other latencies present in a Non-Real-Time Operating System (NRTOS). The consumer-producer architecture [17] is illustrated in Figure 4.7. In this diagram, both ProducerThread and ConsumerThread share access to the SynchronizedBufferManager, which provides a shared memory implementation with exclusive access through locking mechanisms common to both threads. The ProducerConsumer object initiates and manages both threads and the shared memory object.
4.3.1 Producer Thread

In the producer-consumer pattern, the ProducerThread is responsible for producing data, placing it into a shared memory region for access by the consumer. The GnuRadar architecture uses a ProducerThread object to transport a continuous stream of data from the USRP1 receiver, into a shared memory region managed by the ProducerConsumerModel shown in Figure 4.7. The ProducerThread communicates with the attached receiver through a generic, Device interface class. Additional receivers can be added by simply implementing this interface. Figure 4.8 depicts a class diagram of the ProducerThread and the primary objects that it interacts with.
Figure 4.7. Class diagram illustrating the producer-consumer model’s architecture in the GnuRadar system.
Figure 4.8. Class diagram illustrating *ProducerThread* interaction in the GnuRadar system.

### 4.3.1.1 Device Interface

As mentioned in the previous section, additional receivers can be added to the *GnuRadar* system by implementing the *Device* interface class, which dictates a minimum set of capabilities that a receiver must provide in order to be compatible with the system. Commercial Off-The-Shelf (COTS) receivers provide supporting software for the user interfaces with the system. Implementing a receiver software interface in the *GnuRadar* system requires that the user create a class that: 1) implements the *Device* interface class, and 2) links in the required software provided by the manufacturer. The receiver implementation provided for the *USRP1* device is illustrated with a class diagram in Figure 4.9.
4.3.1.2 USRP1 Library

The USRP1 device is backed by a community of users and comes prepackaged inside of a larger, open-source software processing platform called GnuRadio [4], but the core software required to interface with the USRP1 hardware is a small subset of the default installation. The GnuRadar system, targeted for use with hardware at the Rock Springs radar site, operates with a very specific set of parameters, thus eliminating the need for optional accessories available to the USRP1. Given this well-defined use case, along with a desire to keep the system simple and maintainable, a careful examination of the default USRP1 software installation was performed, and software components pertaining to our use case were isolated, extracted, and added to the baseline GnuRadar software using a subdirectory of the larger project and a separate build script for compilation of a static library that could be optionally linked into the build.
4.3.1.3 Data Synchronization

During the implementation of the USRP1 receiver component, a problem was discovered in the device’s USB buffering firmware, preventing the device from effectively flushing and resetting its internal buffers. This was not apparent in the device’s default operating mode, which collects data continuously, where users can safely ignore stale data in the USB chip’s buffer. For a pulsed radar system, synchronization is critical for determining range in the data stream, therefore each sample in the system must be known in relation to its corresponding transmit pulse. In addition to stale data in the buffer, the number of stale samples varied from one system start to the next, making synchronization impossible without injecting some form of identifying data into the existing sample stream to designate the first range cell for the $N^{th}$ pulse return. When considering this solution, two issues were of initial concern: 1) the identifying data would need to be sufficiently unique from the raw data to prevent false alignments, and 2) the solution would introduce data loss to some degree since it must replace valid samples in the receiver’s data stream. The first issue was remedied by using a single sample in each I/Q channel with maximum amplitude since a full-scale output was unachievable given signal losses in the processing chain, resulting in a less-than-unity output. The second issue was mitigated since the receiver’s processing chain uses a CIC filter, which introduces inherent filter delays in the hardware, producing uninitialized samples in the beginning of a radar return. These samples are typically discarded, but, in this case, we were able to insert the data framing samples into this region, thus maintaining the desired data rate while eliminating loss of data.
4.3.2 Consumer Thread

The ConsumerThread in the producer-consumer pattern must read and process (i.e. adding metadata and formatting for storage) data from the circular buffer at a rate greater than the ProducerThread supplies it in order to prevent data overflows, which, although acceptable in some systems, are not tolerated in a sampling radar system such as this. In this system, the worst-case data rate from the perspective of the ConsumerThread is determined by the receiver’s maximum, sustainable data rate, which is 32 MBPS for the USRP1 device. This rate is derived assuming continuous collection with the maximum, 8 MHz instantaneous bandwidth. Given that the GnuRadar system will use the USRP1 in a pulse mode of operation, only a fraction of the maximum data rate will be achievable, depending on the number of samples collected for each radar return. Initial testing of this model showed that almost 100 MBPS rates were achievable by the ConsumerThread thread, so even the worst-case scenario did not raise concerns. Additionally, a hardware-based RAID system provided responsive storage capabilities, with sustainable write speeds of up to 650 MBPS. In addition to reading, processing, and storing data, the ConsumerThread also manages a file in the shared memory region which provides real-time information about the current read buffer in memory. This file can be accessed and used by external processes for on-line data processing, diagnostics, analysis, et cetera. An overview of the ConsumerThread class and objects it interacts with are depicted in Figure 4.10.
Figure 4.10. Class diagram depicting *ConsumerThread* class and dependent classes and libraries.

### 4.3.2.1 Shared Header File

In a typical data collecting system, digitized data samples from a source are collected, possibly processed, formatted, and stored to disk in real time. Most systems provide some form of real-time data plotting tool that can be used as both a diagnostic tool to validate system health, and also a tool used for some form of real-time data analysis. Commonly, these processing and plotting tools operate on data that has been written to disk, which, depending on the data rates, can have a dramatic impact on disk storage performance due to scheduled reads and writes and the associated seek times required. In the *GnuRadar* system, the circular buffer was placed in a shared memory location, meaning that other processes (i.e. programs) can access the circular buffer as a read-only process without affecting *ConsumerThread* tasks. Providing access to the
data requires that the process know the head, tail, and depth of the circular buffer to prevent reading/processing stale data. Additionally, the reader must understand the format of the raw data in the buffer. All of this information is provided in a header file located in the shared memory region alongside the circular buffer. The ConsumerThread has access to the circular buffer’s state, and, each time it processes a buffer, it updates this header file. To make access to the file exclusive, a temporary lock file is created when the ConsumerThread updates the header file, and the lock file is removed when the update is complete. Readers accessing the header file can check for the existence of the lock file before attempting to read the header file. Although filesystems allow parallel access to files through copy mechanisms, the lock file ensures that the reader consistently gets the most current state of the circular buffer. The header file is formatted using an open format called YAML [8], which provides readers and writers in several languages, allowing readers the freedom to develop processing tools in the language of their choice. An example file is displayed in Figure 4.11.

4.3.2.2 Data Formatting and Storage

During the system configuration process, the size of buffers contained within the circular buffer are determined by calculating the number of bytes required to store one second of complex data samples. Note that a one-second buffer boundary only holds true when it’s an integer multiple of the chosen PRI, which has been the case for all used configurations thus far. If a non-integral PRI is chosen, a floor function is used to truncate the boundary to one PRI less than the one-second boundary. In either case, the time stamp applied to the metadata associated with the buffer is currently
Shared Buffer File Format

Figure 4.11. Illustration of text-based YAML file format used to provide header information to external processes using the shared circular buffer.

limited to one-second resolution, thus a search for a particular table in the data set may return multiple results, depending on the choice of the PRI. Data read by the ConsumerThread is formatted into a header containing metadata, and a block of raw data containing range cells from a collection of radar returns. The format used for storage in the GnuRadar system is referred to as the Hierarchical Data Format (HDF) [11], and more specifically HDF5, which refers to the most recent version of the format. HDF5 is a well-known, standardized format, both fast and efficient in terms of read/write speed and memory usage. From a top-level view, a file contains a set of hierarchical objects which use a reference system similar to that of the Unix filesystem, where “/” refers to the root object. Any number of objects can be stored under the root object, and objects can be nested with other objects; even linked if the design
requires it. In the *GnuRadar* system, a simple format was used, and is described as follows: the root object contains a global header file describing variables considered constant throughout the data collection and is followed by one or more data tables, which are essentially a header/data combination derived from each buffer read in the circular buffer. A visualization of this concept is provided in Figure 4.12 and an example of a *data table* is illustrating in Figure 4.13. By default, *HDF5* opens and writes to a single file, writing tables indefinitely until the user stops the data collection. This behavior was determined to be problematic since typical data collections could span several consecutive days, and any failure in the system corrupted the entire data set. In a first attempt to remedy the problem, the default file behavior was switched over to use something referred to as the *family* file set, which divided a data set into a group of fixed-size files, where the size was configurable by the user. Testing revealed that this storage variation also corrupted the entire data set during a single failure since all

![HDF5R Format](image_url)

Figure 4.12. Visualization of the format used in the *GnuRadar* system.
files were linked internally using metadata. Additionally, analysis tools provided by the 
HDF5 group required that files be recombined into the default SEC2 format. To fix the issue, a variation of the library was written [27], where a top-level object managed files used by the default HDF5 implementation. A user-definable file size was passed into this object during initialization, and, when files reached the defined size, the file was closed, and a new file was seamlessly opened in its place. To keep the file names unique, an 8-digit index was appended to the chosen base file name. This solution isolated collection failures to a single file at the expense of providing no connection between files in the set, other than the chosen base file name.
Accounting for unforeseen gain mismatches requires some form of receiver calibration. As discussed in an earlier section, all gains and losses in the digital stages of the receiver are well-defined and have been modeled. However, losses at the receiver’s input, before the signal is sampled, are not fully characterized due to unknown parameters in the receiver’s matching circuitry. Using a test setup shown in Figure 5.1, a test signal, slightly offset from the radar system’s carrier signal, was injected directly into the receiver for sampling and analysis. Both physical and logical components in the processing chain are illustrated in Figure 5.2 along with specified gains for each block in the chain. In this setup, an input signal with a frequency of 50.9 MHz a peak-to-peak amplitude of 428 mV was injected directly into the USRP1. The ADC’s programmable gain was set to 0 dB, and the digital mixer was tuned to 13 MHz, resulting in a down-converted output signal at 100 KHz. The USRP1 was configured for a single channel collection, sampled at 64 MSPS, and a CIC filter bandwidth of 1 MHz was used. Given the small, fractional bandwidth of the signal compared with the available receiver bandwidth, CIC filter droop loss was negligible (i.e. less than 0.1%) and purposely left out of the analysis. Given the ADC’s step size of

\[
\Delta = \frac{V_{FS}}{2^b} \bigg|_{FS=2, b=12} = \frac{2}{2^{12}} = 2^{-11} = 4.8828 \times 10^{-4} \left(\frac{V}{\text{step}}\right), \quad (5.1)
\]
**USRP Test Setup**

![Diagram of USRP Test Setup](image)

**Input Signal**
- freq = 50.9 MHz
- amp = 428 mV pk-pk

**Output Signal**
- freq = 100 KHz
- amp = 4440 ADC Count pk-pk

**Figure 5.1.** *USRP1* receiver test setup used for gain calibration.

**Figure 5.2.** *USRP1* receiver test setup block diagram illustrating unknown loss in the system’s matching circuitry.
where $V_{FS}$ is the full scale input voltage of the ADC, and $b$ is the ADC’s bit resolution. Although the ADC is a 12-bit sampler, samples are placed into 16-bit fields by shifting the 12-bit value up 3 bits and extending the sign bit. Given this information in conjunction with Equation (5.1), the sampled output peak from the test signal is given by

$$V_{pp}^{out} = y \times \Delta = (4440 \gg 3) \times 4.8828 \times 10^{-4} = 271.00 \times 10^{-3} \text{ (V)} ,$$  

where $y$ is the 16-bit, peak-to-peak ADC count. The total receiver loss is given by

$$RX_{loss} = -20 \log_{10} \left( \frac{271.00}{428.00} \right) = 3.9694 \text{ (dB)} .$$  

Using Figure 5.2, known receiver losses can be removed, leaving

$$M_{loss} = RX_{loss} - 0.38 - 1.6880 = 1.8896 \text{ (dB)}$$  

of unaccounted loss in the receiver. If assuming a simple mismatch loss as illustrated in Figure 5.2, the mismatch can be solved for load impedance using a reflection coefficient given by

$$\rho = \frac{|Z_l - Z_s|}{Z_s + Z_l} ,$$  

where $Z_s$ is the source impedance as seen by the ADC and, according to the schematics, assumed to be 50 Ω. The equation for mismatch loss using the reflection coefficient is given by

$$L_{match} = -10 \log_{10} \left( 1 - \rho^2 \right) .$$  

Solving the equation for $\rho$ results in

$$\rho = \sqrt{1 - 10^{-\frac{L_{\text{match}}}{10}}}. \quad (5.7)$$

Substituting $L_{\text{match}} = 1.8896$ with a source impedance of $Z_s = 50\Omega$, the apparent load impedance can be calculated, resulting in $Z_l = 196.29\Omega$. It’s interesting to note that the ADC’s datasheet specifies the ADC input impedance at $200\Omega$, which, if no compensating circuitry were added, would account for the loss completely. Validating this claim would require manual tracing of the ADC input signals on the main board as the schematics do not appear to show this area in any detail.
In this chapter, a discussion of meteor discrimination techniques will be discussed and an application of the provided algorithms will be shown in Chapter 7. Many techniques for meteor detection exist [5, 30], most of which use either time-series or frequency techniques. In this system, algorithms used are based on image processing techniques and provide more efficient means for quickly discriminating potential targets from noise.

6.1 Noise and Spurious Interference Reduction

Almost all target discriminators rely on some form of SNR-based algorithm that attempts to separate potential targets from noise and interference. Classical time-series techniques use Neyman-Pearson (NP) detectors that require a priori knowledge of both target and noise distributions, as well as an acceptable false alarm threshold which is used as the discriminator. Although these detectors perform well in well-behaved environments, spurious interference from sources of electromagnetic interference can greatly degrade detection performance, thereby increasing false alarm rates to unacceptable levels. Mitigating interference requires some level of signal preconditioning and standardization before discrimination is performed. In this chapter image-based processing techniques will be used to implement a multi-stage discriminator that is efficient and accurate. In this system, we start with a two-dimensional, \( m \times n \) raw data set denoted
by

\[ D = \begin{pmatrix} x_1 \ x_2 \ \cdots \ x_n \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix}, \quad (6.1) \]

where the \( m \)-dimension represents range in meters and the \( n \)-dimension represents PRIs in units of seconds. Data storage, as discussed in Section 4.3.2.2, partitions incoming data into an \( m \times n \) table containing fixed, one-second blocks of data. When visualizing a table of data, the resulting image will be referred to as a frame, illustrating a one-second window of time over some user-defined number of range cells. When visualizing unprocessed data frames containing power, various forms of interference are commonly present, most of which are unpredictable and short-lived in nature. When attempting to discriminate targets from noise, the most common approach is to integrate the image over some fixed amount of time, which increases the SNR by assuming that the noise present in the image is uncorrelated and Gaussian-distributed. This is true for thermal sources of noise, but doesn’t apply to Electromagnetic interference (EMI), which emanates from both man-made and natural sources. Additionally, rapid fluctuations in target power, which may provide key information for characterization, are lost during integration. Other approaches involve some form of low-pass filter to remove spurious noise, but these suffer similar problems and tend to blur the overall image. One widely-used image processing technique to remove spurious data is the median filter. This filter operates in a nonlinear fashion, removing short-lived interference without degrading the target of interest. Unlike a low-pass filter, which removes noise at the expense of blurring
the overall image, the median filter replaces a pixel with a median value determined by looking at a user-defined number of neighboring pixels. These pixels are sorted and the median value replaces the pixel being operated on. This has two primary effects on the resulting image: 1) Using median values will lower the overall intensity of the filtered image, and 2) objects large enough to span the region examined remain unaltered. EMI sources tend to produce intense bursts of energy that cover several range cells, but are constrained to a single PRI, meaning that a median filter operating with a relatively small window will remove interference with minimal effect to potential targets. As an example, Figure 6.1 illustrates a low-SNR target in a frame containing sporadic interference. Using a 3x3 pixel window, the median-filtered image is illustrated in Figure 6.2 and the resulting image shows that, although the overall target’s SNR has dropped by \( \approx 2 \) dB, the interference has been completely removed while increasing the effective target SNR by more than 6 dB.
Figure 6.1. Image contaminated by interference.
A 3x3 median filter, as shown in the example, is used as a first stage in the meteor detection algorithm.

6.2 Noise Estimation

Estimation of the noise present in the image is crucial for discriminating potential targets, especially for low-power targets. Several approaches are used in practice, many of which attempt to estimate noise using regions that are well-understood and considered target free. An alternate estimate, used in this system, obtains a noise estimate by applying a combination of two filters across the entire image. The first filter, operating
in the $m$-dimension, produces an average PRI power vector given by

$$\hat{X}_n = \left( E[x_1] \ E[x_2] \ \cdots \ E[x_n] \right)$$  

(6.2)

Next, $\hat{X}_n$ is filtered using an $\alpha$-trimmed mean filter [3], which belongs to a family of filters referred to as Order Statistic Filters (OSF) that rely on ranking of observations of a random variable by magnitude. This filter sorts $\hat{X}_n$ by magnitude in ascending order and the variable $\alpha$ is defined as a percentage of samples to trim from both the beginning and end of the selected data array. For example, setting $\alpha = 0.25$ results in removal of the lower and upper 25 percent of values. Remaining samples falling between the 25$^{th}$ and 75$^{th}$ percentiles will be used to calculate a mean. In this application, the first-stage filter averages across range cells for each return collected by the radar, averaging out variations across range at a particular instant in time. The second-stage filter, or $\alpha$-trimmed mean filter uses an $\alpha$ value of 0.25, effectively removing upper-bound outliers caused by strong interference and/or targets, while removing lower-bound values that might provide a lower-than-average estimate when considering the entire image. Using the median-filtered image in Figure 6.2 as an input, intermediate and final results of the two-stage filter noise estimate $\hat{p}_n$ are illustrated in Figure 6.3.
Figure 6.3. Illustration of noise floor estimation using $\alpha$-trimmed mean filter.

6.3 RFI Detection and Removal

In Section 6.2, an estimator for noise was provided with the assumption that the noise was Gaussian but contained spurious outliers. In this section, the noise estimate will be used to detect and remove larger, more prominent sources of interference that persist across several PRIs. The algorithm operates in two sequential phases each using a configurable threshold for each. In the first phase, each range cell’s SNR level is
compared against a threshold $\tau_{RFI}$ in a test defined by

$$
\gamma_{i,j} = \begin{cases} 
1 & : x_{i,j} \geq \tau_{RFI} \\
0 & : x_{i,j} < \tau_{RFI},
\end{cases}
$$

where a positive value indicates interference for sample value $x_{i,j}$ for $i \in M$ and $j \in N$.

After determining which range cells containing interference, a secondary calculation is performed to determine the percentage of interference contained within each PRI and the results is compared against a second threshold $\beta_{RFI}$, resulting in

$$
O = \left\{ j : \frac{\sum_{i \in M} \gamma_{i,j}}{|M|} \geq \beta_{RFI} \land j \in N \right\},
$$

where the set $O$ contains indices of all PRIs containing interference. Removing the interference from affected PRIs is accomplished by finding the maximum interference-to-noise ratio (INR) and adjusting the subset of contaminated PRIs by

$$
\mathbf{x}'_j = \left( x_{i,j} - \max_{i \in M} \left( x_{i,j} \right) \right)_j \quad \forall j \in O,
$$

where $\mathbf{x}'_j$ replaces affected vectors in Equation (6.1). In its current form, this technique is rather simplistic, and while greatly decreasing the false alarm rate, could potentially raise the miss rate if the source of interference overlaps a target falling with the same time slice since all range cells in a PRI are adjusted when detected. On the other hand, PRIs contaminated with this level of interference will typically mask any detections that may be present. An example of the removal algorithm is illustrated in Figures 6.4 and 6.5.
Figure 6.4. Illustration of strong interference in image.

Figure 6.5. Result of interference removal algorithm.
6.4 Clustering

After interference has been removed from the estimate, the noise floor is re-estimated and samples are thresholded against a minimum SNR level \( \tau_{SNR} \) using

\[
 f_{i,j} = \begin{cases} 
 1 : x_{i,j} \geq \tau_{SNR}, \\
 0 : x_{i,j} < \tau_{SNR} 
\end{cases} \quad (6.5)
\]

resulting in a set of tuples indexed by time and range

\[
 Q = \left\{ (i,j) : f_{i,j}, i \in M, j \in N \right\}. \quad (6.6)
\]

After thresholding with Equation (6.6), a form of Hierarchical Clustering Analysis (HCA) [9] is used to build a set of one or more clusters using metrics that take advantage of known meteor models. Typical algorithms require that the user build a similarity/dissimilarity matrix using one or more distance metrics, followed by a set of rules through which the samples are clustered, resulting in algorithmic complexity ranging from \( O(n^2) \) to \( O(2^n) \). Given that a general model of a meteor exists, combined with density constraints that can be applied to both time and range dimensions, algorithmic complexity can be greatly reduced over standard algorithms. The algorithm begins by creating clusters that encompass all detections for a given range cell spanning all PRIs in the image. This initial set of clusters is defined by

\[
 C = \left\{ j : (i,j) \in Q \land i = k \right\} : k \in M \right\}, \quad (6.7)
\]
where \( C \) represents a set of time indices \( j \) for each range \( i \) and empty sets result \( \forall i \neq k \).

Next, a first-order time difference is computed for each cluster in \( C \), along the time dimension (i.e. columns). Time gaps greater than \( \tau_{t_{\text{max}}} \) are marked as boundaries given by

\[
B = \left\{ j : x_{i,j} \geq \tau_{t_{\text{max}}} , \ i = k , \ j \in N \right\}_k \ k \in M
\]

which partition each initial range cluster into smaller clusters in the time dimension. After partitioning all non-empty range clusters are tested against a minimum time threshold, given by

\[
g_{i,j} = \begin{cases} 
1 : x_{i,j} \geq \tau_{t_{\text{min}}} \\
0 : x_{i,j} < \tau_{t_{\text{min}}}
\end{cases}
\]

and those clusters that fail to exceed the threshold are pruned from the list. Next, remaining clusters are compared against a maximum distance threshold in the range dimension, and those clusters that fall within the maximum limit are merged. Lastly, a cluster score is calculated based on the total number of detections contained within the cluster. If the cluster score passes a user-defined score threshold, the cluster is placed into the final output list and considered a meteor event. To better illustrate the algorithm, an illustrated example is provided in Figure 6.6. Although this algorithm is currently used as a detector, it may be better suited as a front-end discriminator, with a final detection stage that examines candidates using classical approaches as mentioned earlier in [5, 30]. Using this algorithm on a laptop with an Intel i7-3630qm processor and 8 GB of RAM, 13 hours of data were examined in approximately 20 minutes, providing an
Figure 6.6. Illustration of clustering algorithm given a set of thresholded detections.
average speed 39x faster than that required for real-time analysis, making algorithms of this nature suitable for real-time discrimination during the data collection process.
Chapter 7

Observational Results

On June 14th, 2013, data was collected for approximately 13 hours using two Yagi-Uda antennas, one for transmission, and one for reception, as described in Section 2.2. The RF front-end circuitry used in the experiment was exactly as described in Section 2.4. The transmitter was set to produce 30 KW peak power pulses using a 28-baud biphase coded pulse with a baud width of 5 microseconds. The pulse code is given by

\[
\text{code} = [ +1, +1, -1, +1, +1, -1, +1, -1, -1, -1, -1, +1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 ].
\] (7.1)

Taking the autocorrelation of the code, the sidelobe level (SLL) is approximately \(-22.92\) dB as shown in Figure 7.1.
In total, approximately 500 events were originally recorded and tagged as events. Further analysis reduced this number to approximately 330 to provide a conservative estimate by discarding weaker echoes that were detected. Figure 7.2 recorded the number of events detected per hour of observation. The peak number of events occurred between 05:30-06:30 AM EST.

Figure 7.1. Autocorrelation function of 28-baud biphase code.
Figure 7.2. Number of events recorded per hour for the collection duration.

Event durations were plotted in Figure 7.3 and illustrate an exponential distribution with the vast majority of events less than 1 second in duration. A handful of longer, non-specular events were recorded, with the longest approaching 12 seconds in duration.
Figure 7.3. Histogram illustrating detected event durations.

Figure 7.4 illustrates the time interval between events. This distribution is also exponential and it appears the bulk of events had intervals less than 5 minutes apart.
Since a single antenna captured events, only range information was available, and not the height of the event. Event ranges, as shown in Figure 7.5, depict Gaussian-distributed data with a mean close to 110 km, which would suggest that detected events were captured almost directly overhead given that they match typical height distributions quite well.
Figure 7.5. Histogram depicting range of detected events.

Figure 7.6 illustrates event SNR levels. Again this distribution appears Gaussian with a mean value around 20 dB. Target SNR levels match well with the theoretical DR calculation in Section 2.3.
Figure 7.6. Histogram displaying SNR levels of detected events.
Chapter 8

Conclusions

In general, the observed data set matched theoretical calculations quite well. It has been mentioned that an antenna noise temperature of 3000K may be underestimated and a more commonly accepted value might be closer to 20000K. I felt that the simulated antenna bandwidth of 5.62 MHz might have been overstated based on observations in the past. If both statements are true, it is likely that the overstated bandwidth may have compensated for the low noise temperature, and might explain why absolute power values matched so well. Since this initial data set was evaluated, students at the radar site have been able to continuously collect data for extended durations, further validating the work established in this thesis.

To conclude, the design of this open-source radar system is complete and available for customization as needed. With additional work, the core design can also be extended for use on a variety of platforms (e.g. ground, airborne, or space systems).
Bibliography


Appendix

Detected Events

The following images were recorded from the dataset described in Chapter 7 and were added to further validate system operation.
Figure A.1. Meteor Event T00014326
Figure A.2. Meteor Event T00019008
Figure A.3. Meteor Event T00020777
Figure A.4. Meteor Event T00026107
Figure A.5. Meteor Event T00026270
Figure A.6. Meteor Event T00026494
Figure A.7. Meteor Event T00027301
Figure A.8. Meteor Event T00029887
Figure A.9. Meteor Event T00032898
Figure A.10. Meteor Event T00034926
Figure A.11. Meteor Event T00037650
Figure A.12. Meteor Event T00039549
Figure A.13. Meteor Event T00039550
Figure A.14. Meteor Event T00041871
Figure A.15. Meteor Event T00043312
Figure A.16. Meteor Event T00044059
Figure A.17. Meteor Event T00045180
Figure A.18. Meteor Event T00045447
Figure A.19. Meteor Event T00046833
Figure A.20. Meteor Event T00049294
Figure A.21. Meteor Event T00049912