Spring 1-1-2017

Software Design and Implementation of a Next Generation Gnss Data Logger for Radio Frequency Interference Detection

Sean Rivera
University of Colorado at Boulder, sean.rivera@colorado.edu

Software Design and Implementation of a Next Generation GNSS Data Logger for Radio Frequency Interference Detection

by

Sean Rivera

B.S, University of Colorado Boulder, 2014

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Masters of Science

Aerospace Engineering

2017

This thesis entitled:
Software Design and Implementation of a Next Generation GNSS Data Logger for Radio Frequency Interference Detection
written by Sean Rivera
has been approved for the Aerospace Engineering

Prof. Dennis Akos

Dr. Nagaraj Channarayapatna Shivaramaiah

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Rivera, Sean (M.S Aerospace Engineering)

Software Design and Implementation of a Next Generation GNSS Data Logger for Radio Frequency Interference Detection

Thesis directed by Prof. Dennis Akos

The aim of this thesis is to implement a new GNSS interference detection system that extends the functionality of previous systems. This technology will be evaluated on the effectiveness of data collection in GPS L1, GPS L2C and all software changes required to implement it. Additionally there is discussion about extending the technology to log GLONASS L1 and GLONASS L2 data. This thesis is broken up into three sections. First, it describes the current state of GPS ASIC systems, using the SiGe receiver. It covers the reasons that this ASIC requires modernization as well as what behavior the system seeks to emulate. Next, it describes the new hardware that will make up the base of the new system. It covers what each component has been added for and its differences from the previous version. The focus is on the new ability of the receiver to receiver L2C simultaneously with L1 as well as the increased data rate which allows for better positioning. Finally, the thesis describes new software development effort centered around the new system. The thesis describes three test modes used, continuous logging, triggered logging and periodic logging. Extra attention is paid to the implementation of a real time networked version of the code. The final goal of the thesis is to build a low cost distributed system that is capable of detecting and localizing GPS and GLONASS interference on both the L1 and L2 bands.

Acknowledgements

I would like to thank Dr. Dennis Akos for all of his assistance with my master's degree. Without his help and patience, I would not be here. I would also like to thank Ryan Blay, Jorge Cervantes, and Alexander Ray for all of their help with this project. Finally, I would like to thank Luka Strizic and Guillem for their help with the development of essential software libraries. I would also like to thank Nagaraj Channarayapatna Shivaramaiah and Dallas Masters for serving on my committee and all of their help.

Contents

List of Tables

List of Figures

Definition of terms

GPS: Global Positioning System, US based

GNSS: Global Navigation Satellite System, Generic term

GLONASS: Global Navigation Satellite System, Russian based

L1: L1 Frequency, sits at 1575.42 MHz for GPS and spans 1602.0 MHz center frequency for GLONASS[1]

L2: L2 Frequency, sits at 1227.76 MHz for GPS and spans 1246.0 MHz center frequency for GLONASS

L2C: Civilian signal at GPS L2

TDOA: Time Difference Of Arrival

PDOA: Power Difference Of Arrival

AGC: Automatic Gain Control

ADC: Analog to Digital Converter

USB: Universal Serial Bus

RTOS: Real Time Operating System

[1] GLONASS used frequency division for their signals

Chapter 1

Introduction

1.1 Statement of purpose

GPS interference is a growing area of concern as more devices become dependent on GPS. A disruption to GPS has the potential to affect a wide variety of industries as not only is the GPS used for positioning of critical transportation infrastructures like aircraft and sea craft, it is also used as a synchronized timing device for the commercial, energy, and international sectors. There have been an increasing number of events in recent years where GPS systems have been compromised or otherwise interfered with[5]. These events are not simple stand alone problems but are instead indicative of a growing issue. As more and more components of infrastructure rely on GPS for accurate positioning and time, these components become more vulnerable to GPS interference which threatens to shut down. The goal of this project is to modernize previous research into GPS interference detection as well as extend techniques that have mostly been tested on GPS L1 onto GPS L2C providing a framework for further research efforts into GPS interference detection.

For this masters thesis, the goal is to build a multi-frequency GNSS data logging receiver for interference applications. These receivers would be physically distributed in over an area and not synchronized in time. Each receiver relies on its own GPS measurements to calculate a shared time and align all of the received data. The theory behind the interference detection is that once an interfering signal is activated, it propagates to all of the receiver stations, which can locate it with various algorithms. This receiver requires an AGC measurement to determine when interference is occurring as well as IF measurements in order to process and localize the interference. The old system measured the changes in AGC though the SiGE 4120 receiver and when those changes passed a predetermined threshold the system would begin logging the IF data as well. This IF data was used for the localization of a receiver using the TDOA algorithm. In order for this IF data to be useful for localization detection a record of past data has to be logged. This past data is used for acquiring a consistent position and time in order to sync the receivers and provide a base location for each station. Because of the requirement to log past data, all software for these receivers must include some sort of buffering system capable of logging enough data for GPS positioning. This buffer is traditionally paired with an AGC buffer which contains AGC data sampled over the same time steps as the IF data. This AGC data allows for easier post processing since it is time synchronized with the IF data, which allows for quick visualization of the signal quality, as well as determining exactly what caused a trigger. The design of this thesis is centered around the low level hardware and software components of the receiver with a focus on delivering the needed AGC and IF measurements, in a consistent reliable manor.

1.2 Assumptions

For this research paper, the following assumptions were designed into the software of the new system. First of all, it was assumed that GPS receivers start with enough interference free GPS data to get an accurate position solution. This means that data collection begins before GPS interference and that there are enough satellites in the sky visible to the receiver to get a GPS position solution. This is required because both the TDOA and PDOA algorithms require an initial position estimate of the receiver to make an estimate of the interference location. Next, it is assumed that sensors do not move once interference starts. The sensors have to remain stationary because once the interference starts, the software is unable to use GPS to get its current position and thus it continues to operate based on the last good measurement. If the sensors are moved during the meantime this propagation is disturbed and the results become inaccurate. The final assumption of the software is that there at least three sensors that must be distributed in distance. This assumption is simply required owning to the fundamentals of positioning. In order to position something in three dimensions, three signals are required assuming that there is time synchronization between the receivers which there is from GPS in this case.

Beyond the previous assumptions, there were a couple of assumptions that were made due to implementation constraints. With additional research, each of these assumptions could be removed, but they were useful for the purpose of this paper.

- Receiver is not dropping any samples.
- AGC data that is measured during periods of no interference will average out to a consistent value.

Chapter 2

Background

2.1 Interference in GPS signal spectrum

GPS has 3 frequency bands currently active, L1 (1575.42 MHz), L2 (1227.76 MHz), and L5 (1176.45 MHz). Originally the L1 band was for both civilian and military use while the L2 band was for military use only. This has since changed, and now L2 has a civil signal called L2C. Additionally, a new L5 signal was added in 2010 for better positioning. The L2 band has an explicit anti-jamming feature installed in the form the P code but this signal is for military use only.

GPS jamming is the act of broadcasting a signal on the GPS bands that interferes with normal GPS operation. This signal can take any form, as its power is the only concern. For this paper, the jammer used was a NEAT device[12], which has four operation modes, Continuous Wave(CW), Wide Band (WB) jamming, Narrow Band (NB) jamming, and Chirp jamming. Jamming is a growing risk as the number of systems that depend on GPS increases every year. GPS spoofing is the act of transmitting fake or altered GPS signals to a receiver to confuse its position or time measurement. Basic spoofing can be accomplished by retransmitting a position from another antenna. More advanced spoofing can attempt to change the position to keep up with changes in movement or other system changes. Spoofing and Jamming are similar in execution - poorly executed spoofing is just jamming. Spoofing is a greater risk because most systems are able to function without GPS for short periods anyway but without an ability to tell spoofed state from a safe state a large amount of damage can be done.

For every additional signal that a receiver tracks, spoofing becomes more difficult. That said, most receivers default to GPS L1 measurements over the less reliable L2C or L5[1] . This means that a spoofer can normally just target GPS L1 for spoofing and then jam GPS L2 for a similar effect. This does raise suspicion of interference, however, the L2 band isn't as protected as the L1 band from spurious signals.

2.2 Literature Review

The fundamental basis for this project was based on previous research by Jean-Paul Poncelet and Dennis Akos [15]. In their research paper "A low-cost monitoring station for detection and localization of interference in GPS L1 band" they demonstrated the use of AGC measurements to determine the existence of GPS interference. It also demonstrated the use of TDOA to localize interference and the distributed low-cost system model. The primary goal of this research project is to extend the research done in this paper to GPS L2, as well as modernize the low-cost receivers, as the receivers used have since fallen out of production. One important thing to note is that this paper also covered mitigation strategies for GPS interference.

The next Keystone from this project was a pair of articles from the InsideGNSS magazine, "Interference Localization from Space Theoretical Background"[6] and "Interference Localization from Space Part 2: Applications" [7]. These two articles covered the theoretical and practice components of GPS interference localization. While they were focused on the practical components of the receiver from space, they do an excellent job of covering the TDOA and PDOA[2] algorithms both in theory as well as in practical effect. The paper also covered the FDOA algorithm, but that also required the sensors to be placed in close proximity and thus it was not tested for this research. Beyond the formula and algorithmic theory the paper was an excellent source for the stability of the TDOA and PDOA algorithms, and how much data was needed before accuracy stopped significantly changing. Finally, it covered real world applications for jamming detection, and what exactly the parameters needed were.

Another critical paper that this research was based on is "Assessment of Camera Capture for GPS RFI Monitor"[8]. This paper built on the SiGe receiver adding a real time receiver and cameras to take

[1] The L2 band is less protected then L1. Additionally the L5 band isn't fully functional yet
[2] here called RSSD

pictures of potential RFI sources. While it didn't use TDOA to precisely position the interferences, it did use AGC measurements to detect them in real time and then the SDR code[3] to determine the effect of the passing jammer. This paper is still focusing on L1 only, and thus the challenges of the significantly noisier L2 band are not addressed.

The last of the critical papers is "Development and Demonstration of a TDOA-Based GNSS Interference Signal Localization system" [2]. This covers the practical considerations for the TDOA algorithm including expected energy to distance of a jammer as well as how to locate multiple active jammers with TDOA. This system leverages tightly coupled pairs of USRP receivers to perform the TDOA calculations allowing for a mobile platform of jammer detection.

The article "Is it possible to build a low-cost system to detect and locate a single GNSS jammer in near-real time?"[4] covers a very similar concept space to this masters thesis. While it is only L1 based again, it has a very similar design and test system. Instead of the NT1065 for sensing the jamming, it uses more general purpose USRP receivers which are more expensive than this system. That said this paper covers all of the theoretical backings for a real time TDOA positioning of GPS jamming using low-cost receivers, as well as the framework for passing data back to the server. The paper also covers the amount of noise to expect from multipath and other forms of interference from the TDOA algorithm.

In "Who's Afraid of the Spoofer?"[1] it is demonstrated that AGC responds to spoofing in the same way that it responds to jamming. Additionally, it demonstrated the fine tuning of AGC and the mitigation of the effects of temperature. As such this paper was a model for the triggering thresholds and for the effective distances that AGC can be used as a valid measurement. This paper did use the SiGe for GPS L1 detection, but the conclusions from it apply equally well to GPS L2.

The paper "Chirp-Style GNSS Jamming Signal Tracking and Geolocation"[13] is focused on the Chirp style interference. It focuses mostly on personal privacy devices that generate such as signal and is focused on tracking and location of such devices within 100 to 1000 meters. This paper approaches localization in a very different manner than the thesis, using a signature model and a Kalman filter for localization. While those differences do lead to very different results, the model this paper develops for the chirp signal is used for the detection and localization parts.

A very similar implementation of the system is created in "Implementation of a Jammer Localization System Based on TDOA/AOA Algorithm"[11]. It focuses on tracking broadband signals using the TDOA algorithm. While it also has a centralized station with remote monitoring system, it goes for Central station with remote tracking stations. Because of this, the results from this paper were used as a baseline to evaluate the results from the thesis.

In order to model the jammer spoofer combination, the paper "Characterization of Receiver Response to Spoofing Attacks"[17] was used. This paper analyzed how various civilian GPS receivers responded to spoofing attacks as well as the exact amount of power required to spoof each receiver. Additionally, it evaluated several forms of jamming detection and found that the jamming to noise measure was in effect as was most dynamic measurements. It did find good success in acceleration measurements for spoofing, in that most receivers correctly noted an issue after sufficient acceleration, but didn't notice an issue with velocity changes. The final result of this paper cautions the need for better spoofing detection.

The presentation "Developing Defenses Against Jamming & Spoofing of Civilian GNSS Receivers" [16] was an attempt to model exactly what strategies had been developed for defense against both jamming and spoofing. It then compared the difficulty of deployment and the effectiveness of each. For spoofing detection it found that the best option was in P code[3] correlation, but this required new infrastructure far away from the spoofing, making it more viable for cities and countries to deploy than individuals. That said, it did find that the P code defense was very secure for civilians if this infrastructure was created.

While not directly covering GPS, there was work done on using a multi-constellation receiver to defend against infrastructure under the PROGRESS project[9]. This project is focused on defending against a wide number of threats, including jamming and spoofing, but also the potential loss of GNSS signals in certain areas thanks to space disasters. While it doesn't directly work on hardening the receivers the project does work on augmenting existing receivers to be functional even during jamming. It does so by using the SMS system to keep GNSS systems functional during a loss of connection as well as a dis-

[3] The encrypted military signal

tributed ground station monitoring network to locate any issues.

While there are many solutions out there that target GPS L1, there is a gap in both multi-channel localization and L2C localization. The receiver designed by this thesis is an attempt to develop a receiver which could be used for research in both those areas, as well as any other research which requires multichannel GPS receivers.

2.3 AGC Theory

The AGC is a part of most modern radio systems. It's designed to adjust the gain of a receiver dynamically based on the amount of signal power received in order to extend the range of the ADC and to accommodate differing types of antennas. The ultimate goal of the functioning of the AGC is to create a Gaussian distribution of samples within the captured data[1]. Normally the AGC measurement is very stable, only changing sightly with changes to temperature of the system, and thus the thermal noise floor[4] but, it also can change with the receiver bandwidth for SDR applications These adjustments mean that when the AGC measurement decreases it is normally safe to assume that signal power in band has increased and that when the AGC measure increases signal power decreases. By tapping into this measurement it is possible to get a quick idea of the δ of the signal power with respect to time allowing detection of new signals entering the band. On the SiGe, this measurement is in volts, while on the NT1065 it is in dB of power.

The figure 2.1 is an example of the AGC results as additional power is introduced to the band. It starts out stable, but as signal power increases AGC decreases until it becomes saturated. At this point, the AGC can no longer be used to determine if the power in the band increases. For the new system, there are two measured forms of AGC interference RF AGC and IF AGC. RF AGC is the AGC of the system before mixing while IF AGC is the post mix AGC, and the gain needed directly to convert the results to samples.

[4] GPS sits below the thermal noise floor

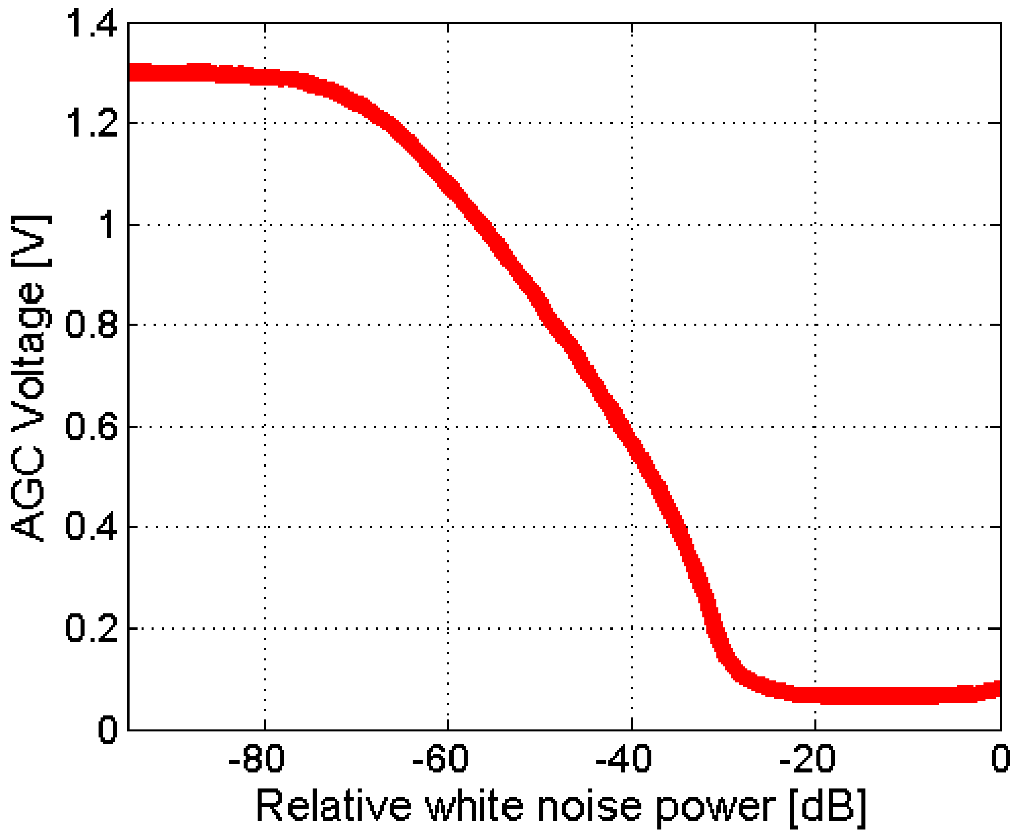


Figure 2.1: Theoretical explanation of AGC performance with respect to power [14]. Data taken from the SiGe receiver.

Chapter 3

Exploration of Existing SiGe Receiver

3.1 SiGe Receiver Hardware

The SiGe receiver is the SiGe GN3S[15] sampler developed by the University of Colorado to act as a distributed GPS receiver for the detection of GPS inference. It is built on the now discontinued SiGe SE4120 GPS L1 front-end chip and connected to the Cypress USB 2.0 controller. It can be configured for a wide variety of sampling rates and bit accuracies with the sampling rates most used for this paper is 8.1838 MHz and 16.3676 MHz at 2 bit complex GPS L1 data. This receiver was initially chosen because it had a way to directly monitor the AGC data as part of the data stream which allows for all the positioning functions described in this thesis. The receiver had a 2-bit ADC for a converter and a nominal IF of 4.273 MHz.

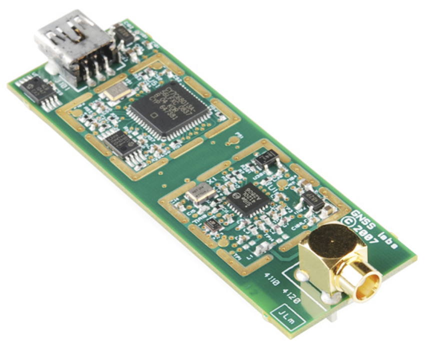


Figure 3.1: SiGe Receiver: released under the CC license[1]

The SiGe receiver firmware could only be programmed by the use of a single particular windows codebase. This codebase is only supported on up to Windows 7, and once the firmware is programmed into the system it remains there until it is programmed again. This dependence on a specialized software package creates an issue if anyone wants to change the firmware of the ASIC.

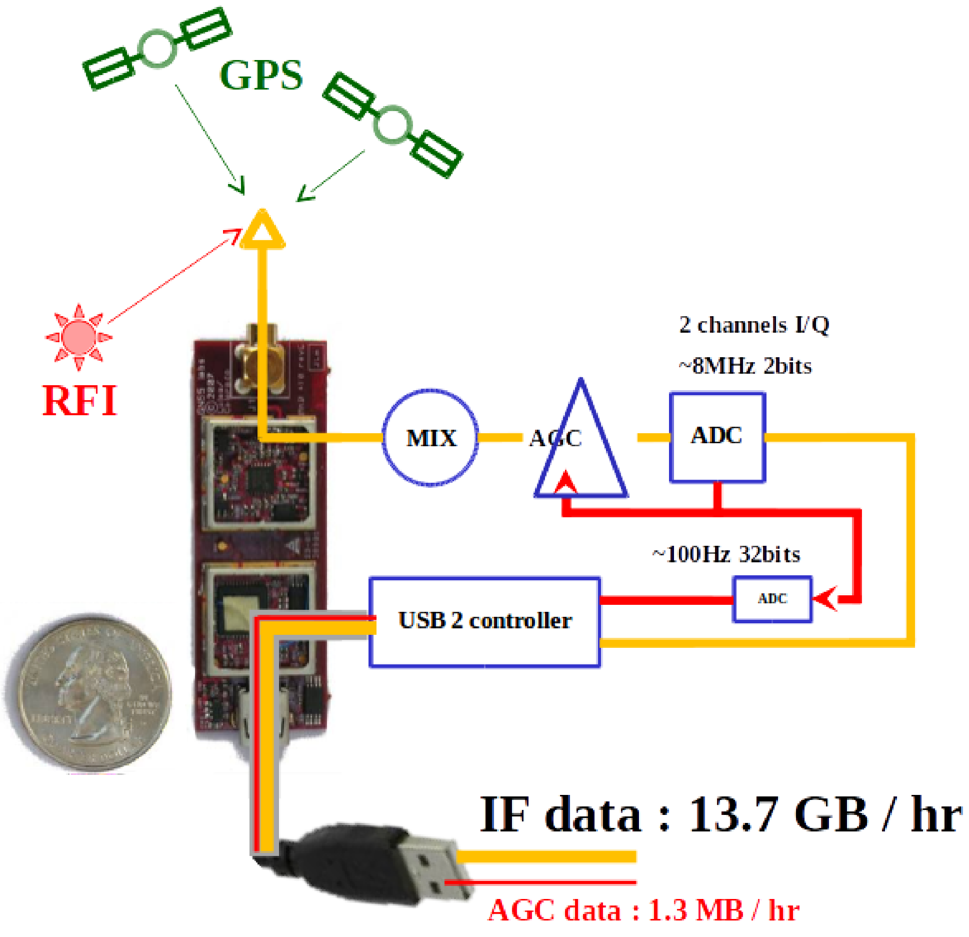


Figure 3.2: Block diagram of how the SiGe functions [14]

3.2 SiGe Receiver Software

A large part of the software research for this project was built on two software projects previously developed. The first of these was the code developed from the low-cost receiver project [15] and the second is the MATLAB SDR code developed based on the book "A Software-Defined GPS and Galileo

[1] Picture taken from `https://www.sparkfun.com/products/retired/10981`

Receiver"[3]. The previous codebase[15] was developed to target the SiGe receiver and collect data in a way that allowed for TDOA positioning during post processing and will be referred to as the previous codebase[15] for the rest of the paper. The MATLAB code is a standard GNSS receiver, with unique modules for GPS L1, GPS L2C, and a combined module for GLONASS L1 and L2 and will be referred to as the MATLAB SDR for the remainder of the document. At a high-level figure 3.3 gives an overview how the code would function as a client or a standalone receiver while figure 3.4 gives an overview of how the receiver functions as a server.

In order to understand the development work of this project, it is important to truly understand the previous work done by Poncelet[15]. This code was an object-oriented reader-writer threaded C++ project that would collect data from the SiGe with one of its threads, and then write out the data with the other thread. The writer thread also maintains a memory of past data in order to maintain history. This allows the system to acquire a GPS position before the interference stats which is essential for proper TDOA collection. The server code simply integrated with the writer thread, meaning that the data was directly logged to the server instead of the local disk. The code monitors the AGC voltage based on a set threshold, and once the measurements reach the threshold for a certain amount of time, the reader thread informs the writer thread to write. The writer thread will also log data on its own after a certain period of time has passed in order to get a nominal data set for comparison. The writer thread logs all of the data in one of the several different packed formats in order to save space, where each packed format stores data inside a byte. The packed format most used for testing was 2 sets of 2-bit real complex samples stored into a byte, however, 4 sets of 2 bit real samples was also an option. To visualize the individual network client see figure 3.3 where the IF and AGC data come into circular buffers from the external reader thread where they are monitored for triggers. Once a trigger is flagged, the system either writes the data to a file or over the network to a server. When running in network mode, there is one server and multiple clients who connect to the server. Each of the networked receivers collect data from their own SiGe module and monitor it for triggered AGC measures. Once one system detects a trigger it notifies the server, which requests that the clients send their data to it. The server logs all this data to a file for post processing.

The previous codebase[15] was a fully functional multi GPS receiver, with a focus on monitoring AGC for strange events. It was tested at White Sands New Mexico at a DHS event, and the results for it were found to be successful at locating sources of GPS RFI. This software was used for follow-up papers of GPS spoofing detection.

While the previous codebase[15] was fully functional, there were some extensions that were needed for this thesis. The largest issues came about because in the time since his work, a large part of the library that handled the interactions between the SiGe module and the computer, libusb, become deprecated and a completely new version was released. The changes to libusb also brought changes to the Linux kernel which caused errors when the previous codebase[15] was run on newer computers. Additionally, there were new settings that were needed help collect additional data, and enable the real time modes of operation. These setting were related to the nominal logging[2] , as well as the new client server code. Finally in order to move to a cleaner interface the buffer code that logged both the AGC and the IF data needed to be optimized.These changes allowed for writing data out in real time, as well as allowed for both a higher sampling rate of 16.384 MHz and the use of less buffering memory to communicate with the SiGe. The changes were also required to operate on unpacked data.

The libusb depreciation corresponded to a change in the way the Linux operation system was handling USB IO. In version 0.1, it was impossible to create a device with isochronous endpoint I/O, which is one of the four potential modes supported by USB. Isochronous transfers are designed for time-sensitive information where dropping frames are unacceptable for normal operation. This is very helpful for GPS logging since there is a bounded maximum latency, and guaranteed access to the USB pipe. Additionally, version 0.1 doesn't provide the ability to perform asynchronous callbacks, meaning that libusb function calls are blocking until the data returns. This necessitates a structure where the system is polling based, in that the SiGe code spent a portion of its time simply asking for data over the USB driver and then waiting for the results, and this portion of time was optimized by leveraging USB bulk transfers. The Windows version of the code did support both Isochronous Transfers and Asynchronous callbacks leading to a divergence between the code versions. When the Linux kernel began to move over

[2] Nominal logging is where the writer thread will log data on its own based on time

to support the full USB standard, libusb 0.1 was rendered non-functional as the kernel was providing a different USB device then the library could handle. Taken together the switch to libusb 1.0 was required, which necessitated a reworking of the Linux application, allowing it to be brought into line with the Windows application.

The Matlab SDR code is a modified version of open source GPS SDR code. The modifications are primarily focused on handling the packed data that both the SiGe and NT1065 default to. This SDR code is designed to post process GPS data and return position solution as well as all of the measurements necessary to find such a value. Of greatest interest to this project, the code returns time of the first sample and the clock error measurement. Thanks to the time of the first sample, the data sets can be synced up sufficiently well to perform the TDOA algorithm, and thanks to the clock error measurements, the TDOA can be run after the jammer cuts out GPS for some time.

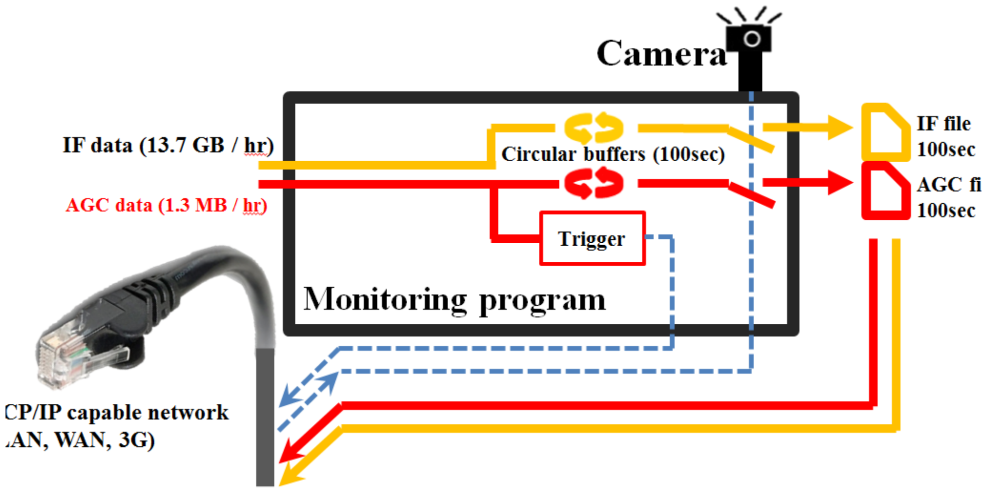


Figure 3.3: Block diagram of how the SiGe Software functions as a client [14]

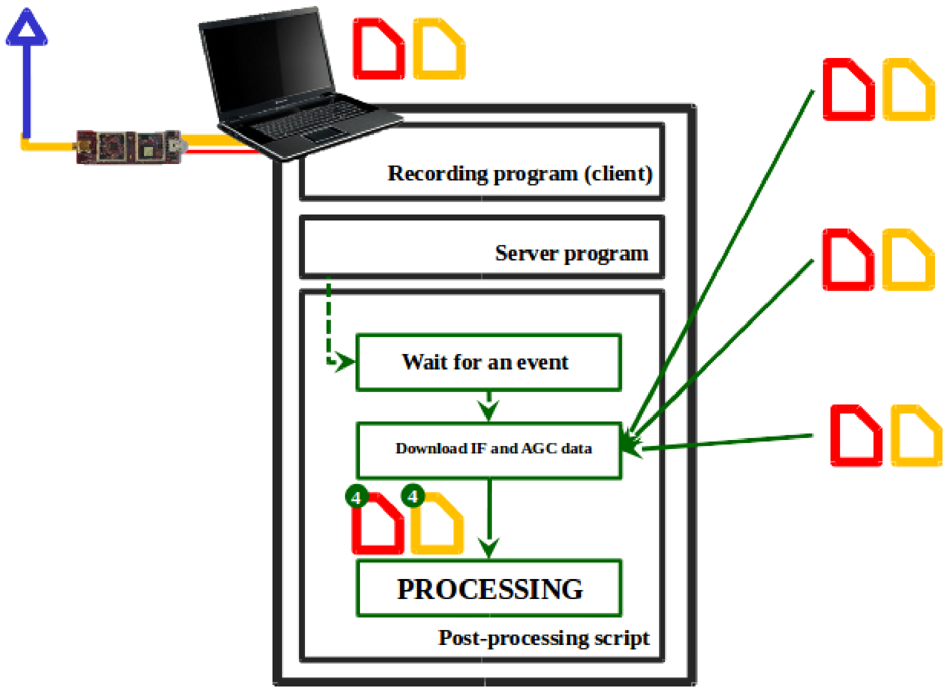


Figure 3.4: Block diagram of how the SiGe Software functions as a server [14]

Chapter 4

Methods

4.1 Hardware

One of the large center points of this project was to extend the existing SiGe research using a new multi channel NT1065 board. This board was produced by the company NTLab in order receive GNSS signals from various differing constellations. The system consists of three connected pieces of hardware that work together to pass data to the processing computer. First, there is the FX3 board which streams samples using the USB3 protocol. We used USB3 because the default data rate for the new NT1065 is 53 MB/s, which is only slightly below USB2's 76 MB/s theoretical maximum. USB3 gives a large factor of safety when one considers the need to take AGC data as well as the 53 MB/s IF data. Next, there is the central control system, the Zedboard which is an FPGA. This control system is currently just being used to pack the AGC data into control registers and stream IF data, however in the future this component will be used to resample IF data, and potentially handle some of the computationally intensive parts of the TDOA algorithm. Finally, there is the NT1065 board itself. This board is connected to a 4-way splitter and then a passive antenna which is powered by the Zedboard.

One key difference between the NT1065 and the SiGe is that the NT1065 AGC data is different from the SiGe because the NT1065 has a dual adaptable AGC system, where it logs both IF and RF AGC. While the new hardware works better than the SiGe hardware, it is still in an earlier stage of development, and thus requires more attention to be paid to the process of starting data collection. In order to begin the data collection process at present, the Zedboard needs to be brought up first, followed by the FX3 which must be brought up before the Zedboard finishes booting. Once the FX3 and Zedboard have been programmed correctly the NT1065 can be programmed by the developed software. One final thing to note is that once the FX3 board and NT1065 have been programmed with firmware for the first time, they have to be reset before they can be programmed again, running the software again will not change the firmware.

Overview.png

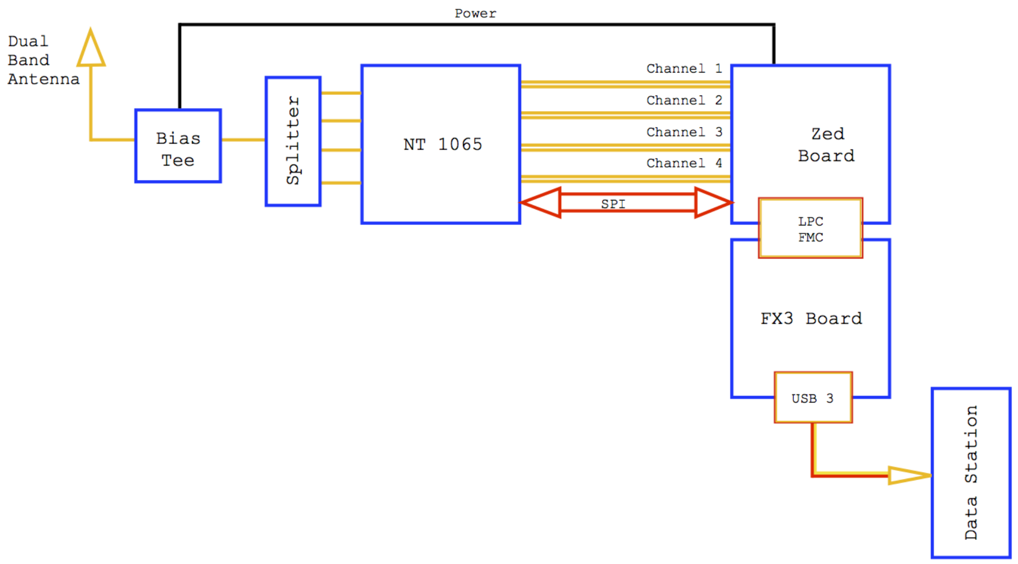


Figure 4.1: Block Diagram overview of the hardware

4.1.1 FX3

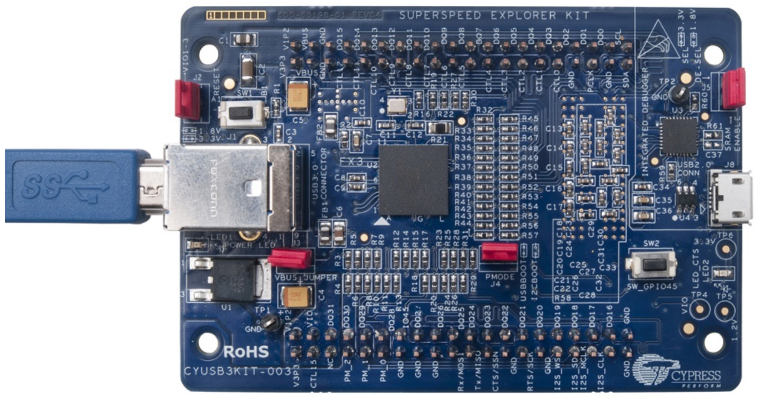


Figure 4.2: FX3 Explorer Kit Released under the CC license[1]

The FX3 SuperSpeed Explorer Kit[2] is a development platform aimed at allowing any system to use USB3 for data processing. It is shaped like an Arduino shield for easy integration and is connected to an LPC FMC[3] connector in order to interface with the Zedboard. The FX3 Board has a full ARM9 core inside of it, as well as 512 KB of RAM which allows it to be loaded with custom firmware. This firmware was designed to grant pulled values for the various registers as well as a continuous stream of data for the IF data.

The FX3 kit is not an extension of the FX2 board used in in the previous SiGe implementation[1], but rather a new architecture based on parallel processing for the higher speeds of USB3. It runs at a CPU speed of 200 MHz as opposed to the 48 MHz of the FX2 and has 512 KB of ram instead of just 16. Additionally, the FX3 board incorporates a more generic interface to stream data into it in the form of the GPIF II[10]. The GPIF II is a programmable data bus designed for high-speed capture which can process any data package sizes[4] from 8-bit to 32-bit. For this system, the FX3 board leveraged the GPIF II interface in 8-bit mode, though there had been some work to change over to 16-bit mode to support complex sampling. Additionally, unlike the FX2 board, the FX3 board is designed to boot from the GPIF

[1] Picture taken from `http://www.cypress.com/documentation/development-kitsboards/cyusb3kit-003-ez-usb-fx3-superspeed-explorer-kit`

[2] part number CYUSB3KIT-003

[3] Low Pin Count FPGA Mezzanine Card

[4] multiple of 8 bits only

II system using simple firmware on the Zedboard. From there the FX3 board is designed to get its current operating firmware from the USB bus from the collection program. Unlike the old FX2 system this allows for rapid firmware reconfiguration depending on needs, which means that the system can leverage the GPIF II bus's reconfigurability for processing. One last part of the system of interest to this project is that the FX3 firmware is designed to run a full threaded RTOS, allowing for finer control of concurrent data transfers and integrations.

4.1.2 NT1065

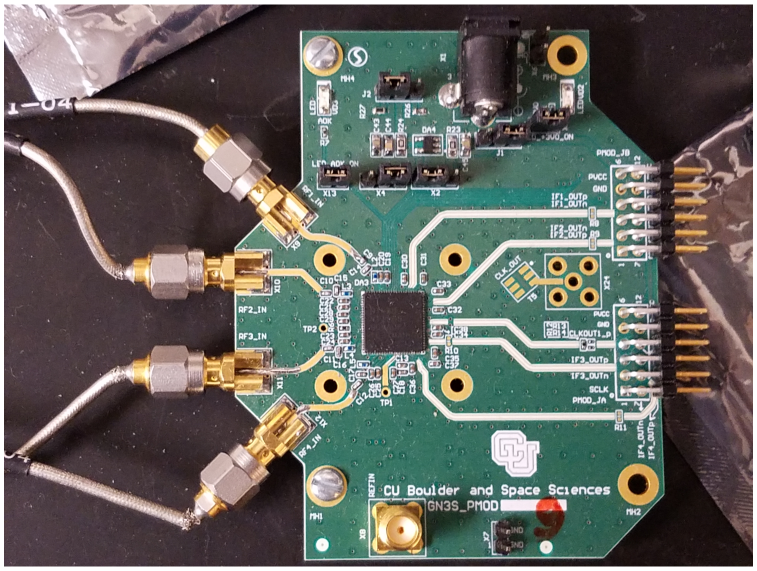


Figure 4.3: NT1065

The NT1065 is a 4 channel GNSS receiver that can be depending on configuration receive almost any combination of constellations and frequency bands. The only limiting factor is that two of the channels are on one of the VCOs, and two are on the other. This means that channels 1 and 3 share the same center frequency as to channels 2 and 4. Both of those sets of two channels have to share 25MHz of bandwidth for a total of 50MHz. Additionally, the configuration can change the bandwidth and the sampling rate. In order to connect the NT1065 to the Zedboard, there are 24 GPIO and power pins used. Of those 24 pins, 17 are actually used for data in some way while the other 7 are power, ground, or unused. From those 17 pins, there are two interesting sets. The first is a set of 8 pins that conveys all the 2-bit channel information. What's interesting about this is that each bit of the result has its own pin, likely to increase sample rate. Additionally, there is an SPI channel on the 24 pins connected which is how the firmware is loaded and the various registers are read. Finally, there is the clock output and other synchronization pins. For this project, the prebuilt configuration set 1 was used. This configuration set gives a sampling frequency of 53 MHz and is focused on recording GPS L1 and L2 data as well as GLONASS L1 and L2 data. It has center frequencies of 1590 MHz and 1235 MHz.

The NT1065 has a collection of 47 registers in a simple map that can all be modified by the SPI interface. These registers control everything from clocking information to system information to channel specific settings. These registers are what allows the settings files to control and configure the system as the settings file is just a list of registers and their byte values. These are passed into the NT1065 from the Zedboard from the host PC as a direct transfer with the first part of the 16-bit message being the register in question, and the second part being the new value. Once this configuration is complete the NT1065 starts collecting data across the specialized channel pins which are passed to the Zedboard where they are packed together into a single byte and sent to the host PC via the FX3 board.

Listing 4.1: Example config file for the NT1065

```
;NT1065
Reg2    03
Reg3    01
Reg4    03
Reg5    00
Reg6    1D
Reg11    0F
Reg12    1C
Reg13    03
Reg14    3E
```

General settings	
Reference Frequency	10MHz
LO Source	PLL «A» for ch#1, ch#3 and PLL «B» for ch#2, ch#4
Clock settings	
Clock Frequency Source	PLL «A»
Clock Frequency	53 MHz
Clock Type	LVDS
Clock amplitude	560/1130 mV
Channel settings	
Ch#1 GNSS	LSB
Ch#2 GNSS	USB
Ch#3 GNSS	USB
Ch#4 GNSS	LSB
Ch#1 IF passband	27.1 MHz
Ch#2 IF passband	20 MHz
Ch#3 IF passband	17.3 MHz
Ch#4 IF passband	15.1 MHz
Output data interface	2-bit ADC
GC mode	RF manual + IF auto
IF AGC Threshold	30 %
ADC output logic-level high	ext. (VCC)
ADC type	clocked by rising edge
Channel settings	
F_{LP} PLL «A»	1590 MHz
F_{LP} PLL «B»	1235 MHz

Table 4.1: Config Set 1 settings used for testing. Taken from `http://ntlab.com/IP/NT1065/NT1065_LE_DS_v2.04.pdf`

4.1.3 Zedboard

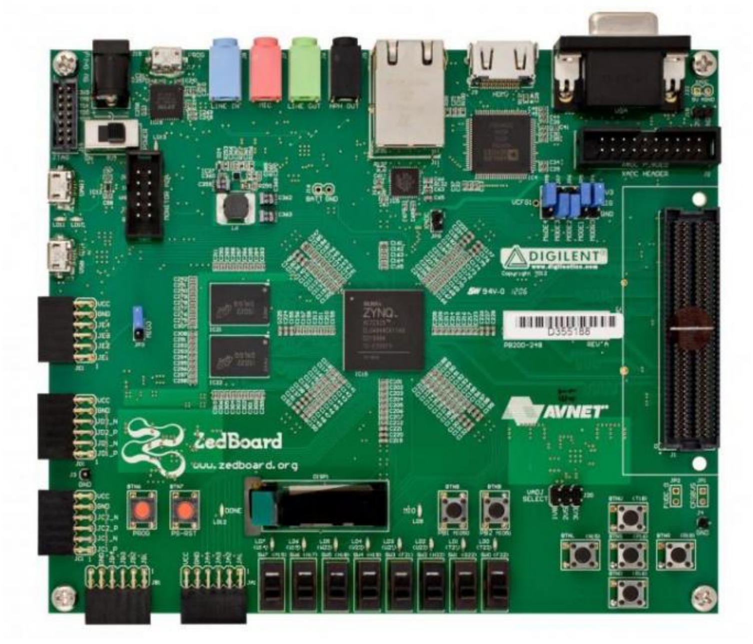


Figure 4.4: Zedboard Released under the CC license[5]

The Zedboard[6] is the central platform that holds the rest of the project together. For the initial develop the Zedboard is used as its modular hardware allows for different configurations to be tested easily. Additionally, it can be quickly programmed through the use of an SD card or through a server boot. The core of the Zedboard is the Xilinx Zynq-7000. This chip has an ARM processor and a full FPGA on board and tightly coupled for quick code integration and efficient processing. From the outside, the Zynq processor just resembles an FPGA, and it isn't until the internal logic is fully connected that the ARM core becomes functional. The Zedboard just breaks out all the pins on the Zynq and proved a host of convenient interfaces like the LPC FMC and the GPIO pins used to communicate with the FX3 and the NT1065 respectively.

The Zedboard is currently running a very basic firmware on it. This firmware simply takes the data from the NT1065 and wraps it into a single byte to write out. Additionally, this firmware handles

[5] Picture taken from `http://zedboard.org/sites/default/files/product_spec_images/ZedBoard_RevA_sideA_0_0%20%281%29_0.jpg`

[6] Zynq Evaluation and Development Board `https://reference.digilentinc.com/_media/zedboard:zedboard_ug.pdf`

the initialization of the FX3 and connects the NT1065 to the FX3 so that the NT1065 registers can be fully configured. Once all that is done, the Zedboard falls into a more passive mode, forwarding the NT1065 registers and IF data to the host PC. Note that the Zedboard is the main reason for the boot order limitations that currently exist, as it has to come online before the FX3 but needs to have not attempted to program the FX3 before the FX3 comes online.

4.1.4 Host PC

The host PC for this platform has very specific requirements for what it needs. The largest of these requirements is the need for a USB3 device port in order to transfer the data. Additionally, the PC needs an SSD capable of handling the sustained disk write of 53 MBs or a network card capable of transferring the same. The system also requires large amounts of RAM for the project, as the minimum of data required for a GPS acquisition and tracking is 2.5 GB.

4.2 Software

Overview.png

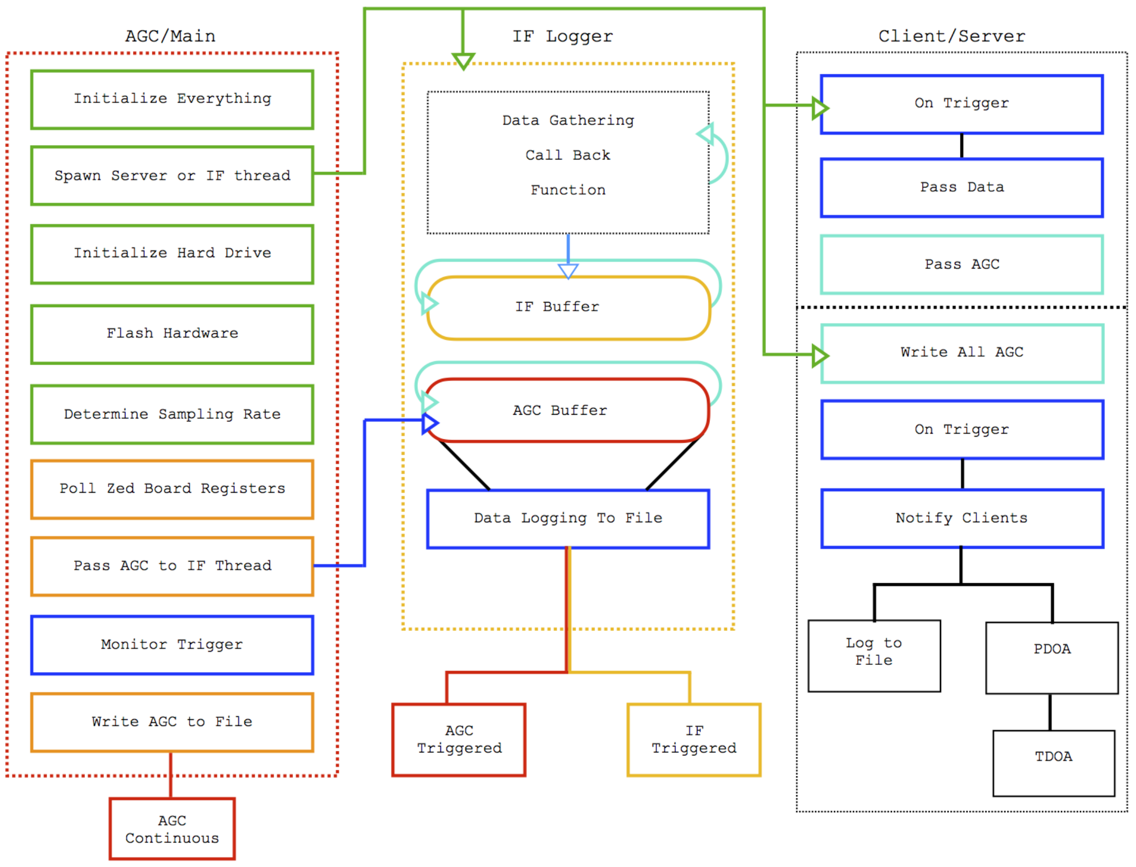


Figure 4.5: Flow chart overview of the Software. The Green indicates one-time events, while the red indicates repeating events. Blue is used to show cross thread communication and all of the threads are marked with dotted borders

4.2.1 Extending the Existing code

The first software goal of this thesis was to extend the previous codebase[15] to support the new features required. The goal here is to add a comparison point for the NT1065. The first step for the update was to improve the performance of the previous codebase[15]. The code initially was developed to support libusb 0.1 which had become deprecated since the code had been developed. Additionally, in order to support a higher sampling rate of 8 MB/s the software was reworked to a clear thread based architecture. This architecture has the two circular buffers which hold the AGC and IF data. These buffers are filled by a callback wrapper around the libusb USB event. Finally, the code was modified in order to better integrate with MATLAB post processing tools. This took the form of static file names, plus cleaner start stop integration from command line flags instead of user intervention.

4.2.2 Code Structure

In order to collect data on the NT1065, new software had to be developed with the goal of emulating the previous codebase[15]. This code was developed in C++ on top of the baseline logging code for the NT1065. The code is threaded by default, with data collection and logging implemented as an extended class. The NT1065 code is built on top of a multi threaded architecture similar to SiGe with three threads as part of the core code. The three threads running are the AGC thread, the IF thread, and the Client/Server thread. The key difference is that AGC collection and IF collection are running in separate threads whereas, in the SiGe, it is a more standard reader-writer architecture. This is to account for the huge data rate of the IF collection. As seen in figure 4.2 each of the three threads has very specific required functionality. As part of the core configuration of the system there are three modes that it can be run in (shown in figure 4.6).

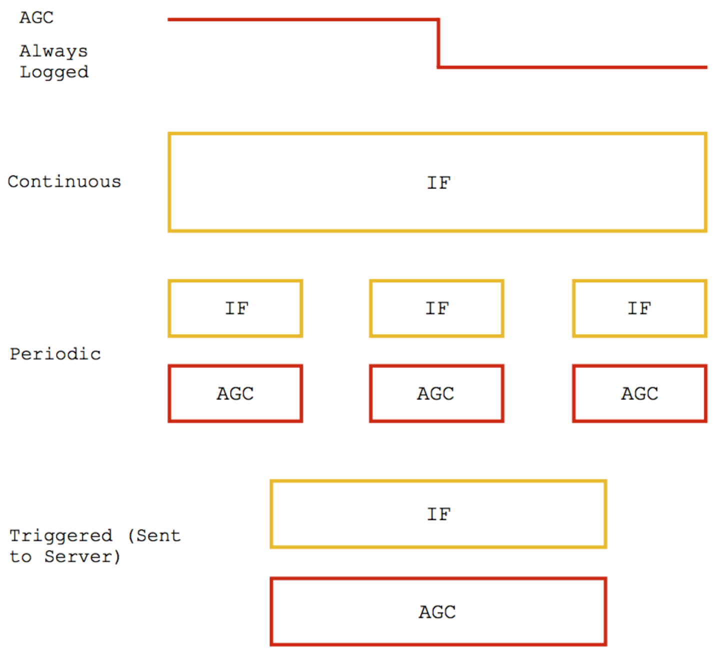
- Continuous mode: Always collect IF and AGC data and log it to a file. No monitoring for triggers or network communication.

- Periodic mode: The code logs data at set intervals, set based on the command line arguments. This data is both IF and AGC, but AGC data is also logged continuously. Can be activated concurrently with triggered mode in which trigger events trump periodic events for determining behavior in the event of a collision.

- Triggered Mode: Main research operation. Consists of a server and multiple clients, with each client connected to an NT1065. All of the clients are monitoring the AGC for a drop below a set threshold, and upon seeing such a drop each client will notify the server. Once notified the server will instruct all the clients to send synchronized data to the server for processing. Additionally AGC data is continuously logged to the server.

The AGC collection thread is the main thread of the program. It is responsible for collecting the AGC data, as well as monitoring for any changes in the AGC. This thread is the first thread spawned and also the only thread that can communicate with the other two. In order to collect AGC data, the thread pulls the receiver, using asynchronous callbacks to dump the results from the AGC registers at a rate of 33-60 Hz depending on other loads of the system with this rate being dependent on the polling of the USB device. These registers contain status information, local receiver temperature, RF AGC, and IF AGC. The most important registers there are the RF AGC and the IF AGC which when combined make up the current AGC measurement used by the receiver. While both AGC measures are important to determine the total system power, the IF AGC is the only one used for threshold detection. This is because the IF AGC is the more stable of the two, while the RF AGC is very noisy whenever it does change. In order to trigger the logger, a threshold is set by the user as a command line argument. Each channel has its own threshold in dB, below which the GNSS signal is considered worth logging. Once a channel records and IF AGC measurement below the threshold, all 4 channels are logged. This allows the reviver to try to get the timing and other location data from unaffected signals It is worth noting that the AGC data is continuously logged to a file, while the IF data defaults to logging only have an event has been recorded, though this can be changed.

The IF collection thread acts as both a reader and a writer for the IF data. It does so by having a registered callback function implemented on top of the USB3 stack. This call back function can be thought of as an independent thread. Additionally, the increased data rate means that there need to be a direct connection between firmware and code, which adds to the instability. Whenever this function receives new data from the USB device it checks to see if the data should be logged to disk or stored in a circular buffer. Once this is determined it loads the data into the appropriate buffer and returns control to the rest of the code. When a trigger is detected it sets a flag to notify the processor to start logging the data *n* seconds in the future, where n is set as a command line argument. Once all the needed data



Modes.png

Figure 4.6: Overview of all of the data collection modes for the software. Note that the continuous logging mode just leverages the constant AGC log instead of duplicating it. The other modes create a duplicate AGC log for ease of processing data

is collected the system writes out both the AGC and IF to a file, using non-blocking move constructors to ensure that it doesn't block the callback function. Once data is written, the code starts on the user specified cool down period, during this time any indication of a trigger is ignored, as there isn't enough data.

The client server thread is the largest alteration between the original paper and the new software. In order to implement real time processing, networking had to be added and while the previous codebase[15] just had each client transmit all their data to the central server for later processing. By treating the nodes as distributed collection points the new software has the ability to pass information about triggers between the clients which allows the triggered logging to work across the network. Similar to the non-networked mode, the AGC data is always logged, and then sent again during a trigger. Additionally, during a trigger, the IF data is sent to the server from all the clients, where the server can process the data for positioning. Both the rates of AGC data logging and the size of the IF/AGC buffers to send to the file can be configured by command line argument. One important thing to note is that the server cannot collect new data, however, it can process existing data with both the TDOA and PDOA algorithms.

The USB driver options available to the software are libusb and cypress with libusb being the default on Linux and cypress being the default on Windows. Both drivers offer asynchronous drivers as well as isochronous endpoints, however, they differ in their implementation. As an example the cypress library expects a buffer to fill for its asynchronous function. This buffer is filled automatically, with a flag being set to indicate when it is full. Contrasting this is the libusb callback function. This callback function is invoked every time there is new data and directly passes the data into the data logging code. In order to accommodate both versions of the USB drivers, the callback function is actually nested one layer above the actual code for handling the callback. This layer basically translates libusb and cypress into a consistent data structure for easy handling. Thanks to this interface the code functions identically regardless of which USB driver is used.

The data logging class supports both libusb callbacks and cypress driver auto filling buffers, though for this project most of the testing was done with libusb. The data logging class stores information in Boost circular buffers, whose size is specified by command line arguments. In order to optimize performance, the class preallocates the two buffers, while also using highly efficient move constructors when it logs data.

4.2.3 Command Line Arguments

Argument	Default	Effect
-h [–help]		Display this help message.
-f [–fximg] arg		File name for the FX3 Image (required)
-n [–ntcfg] arg		File name for the NT1065 CFG file (required)
-o [–outfile] arg		Raw IF output name - postpended by _gmtime_.bin (required)
-a [–agcfile] arg	AGC	Raw AGC output name - postpended by _gmtime_.bin (defaults to AGC)
-l [–library] arg		Library to use to communicate with the board (libusb | cypress) (required)
-d [–duration] arg	315360000	How long to run the code for. Defaults to 10 years.
-1 [–l1threshold] arg	30	Threshold in Db for the GPS L1 AGC monitor. This value is specific to an installation.
-2 [–g1threshold] arg	30	Threshold in Db for the GLO L1 AGC monitor. This value is specific to an installation.
-3 [–l2threshold] arg	30	Threshold in Db for the GPS L2 AGC monitor. This value is specific to an installation.
-4 [–g2threshold] arg	30	Threshold in Db for the GLO L2 AGC monitor. This value is specific to an installation.
–l1repeat arg	1	Number of times the GPS L1 AGC value is below the threshold before a trigger. This value is specific to an installation.
–g1repeat arg	1	Number of times the GLO L1 AGC value is below the threshold before a trigger. This value is specific to an installation.
–l2repeat arg	1	Number of times the GPS L2 AGC value is below the threshold before a trigger. This value is specific to an installation.
–g2repeat arg	1	Number of times the GLO L2 AGC value is below the threshold before a trigger. This value is specific to an installation.
–triggerdelay arg	1	Number of consecutive AGC samples under the threshold before trigger.
-r [–rearmtime] arg	120	Time in seconds before rearm the trigger.
-u [–unpacked]		Write out the data as unpacked samples
-c [–continuous]		Continuously log IF data to file
-s [–staticname]		Generate with a static file name instead of adding the date
–checkhigh		check if the AGC is above the threshold not below - Useful for testing
–nomlength arg	82800	Length in seconds between nominal saving of data (defaults to 23 hours)

-i [–serverip] arg		IP address of the server in the form xxx.xxx.xxx.xxx or ::1
-p [–serverport] arg	12345	Port to open up or connect to(defaults to 12345).
–server		Run in server mode, if this isn't set the port is assumed to be the server port, and thus a server IP is required
–client		Run in client mode overwridden by server mode
-b [–before] arg	60	Length in seconds of the data to be saved before the trigger (defaults to 60s)
-p [–post] arg	20	Length in seconds of the data to be saved after the trigger (defaults to 20s)

Table 4.2: Breakdown of all possible options to the collection code.

Chapter 5

Results

5.1 Verification

In order to confirm the functionality of the new NT1065 receiver and its use for interference localization there were several tests done. These goal of these tests was to get a model of the behavior of the NT1065, confirm what AGC measurements meant what level of interference as well as evaluate that the IF data could be used for interference localization. In order to do so the NT1065 system was placed in a moving vehicle and driven away from the receiver. At various points along the road the vehicle was stopped for 30 seconds in order to locate where in the drive particular data was taken. Once the system reached the end of the test, the NT1065 system was stopped and driven back to the start point for a different jamming mode to be tested.

Point	Distance	SiGe tested?	NT1065 tested?
P1	1m	X	X
P2	33m	X	X
P3	222m	X	X
P4	433m	X	X
P5	580m	X	
P6	700m	X	
P7	853m	X	

Table 5.1: Table of distances for SiGe and NT1065 calibration. For the calibration tests the NEAT[12] jammer was activated, and then the receiver was driven away from the jammer stopping at each of the listed points. This was done to determine the effect the jammer had on the receiver and at what range the SDR[3] could start tracking the GPS signal again.

For this thesis, the testing of the SiGe and NT1065 was done simultaneously. Both receivers were

Test #	NEAT Mode	NT1065 Collect Locations
NT1	L1 NB	P1-P4
NT2	L2 NB	P1-P4
NT3	L1/L2 NB	P1-P4
NT4	L1/L2 WB	P1-P4
NT5	L1/L2 Chirp	P1-P4
NT6	L1/L2 CW	P1-P4
NT7	Off	P1-P4

Table 5.2: Table of tests for NT1065 verification. This is a list of all of the different frequency patterns tested for the jamming test. The goal here was to differentiate the effects of each of the jamming modes.

directly compared to one another, with the SiGe being used as the reference master and the NT1065 being tested against it. This testing was done with the goal of confirming that both code bases behaved as closely as possible for all possible command line arguments. Additionally, both receivers had characterization tests in order to model their behavior. These tests were done both at Fort Huachuca in Arizona, USA as well as in the lab. For these test, the NEAT device was used to generate GPS jamming. The NEAT device is a small hand-held GPS jammer developed to train NATO forces to recognize GPS jamming. It has 100mw of output power and can independently generate jammer signals in both L1 and L2[12].

For the tests at Fort Huachuca, the systems were tested by placing them in the car that started driving away, stopping at all of the points indicated in table 5.1. This allows a comparison between the SiGe and the NT1065 for effectiveness, as well as giving a rough baseline of the amount of distance to the NEAT device that a certain AGC measurement corresponds to.

Outlined in the tables 5.2 and 5.1 there were seven tests of the system out at Fort Huachuca. All of these tests were characterization tests, whose results are shown below in figures 5.1, 5.2, 5.3, and 5.4. In order to understand these figures, it is important to note that they contain both the IF and RF AGC values. The markings on the figures of P1-4 correspond to the physical location [1] of the vehicle at that point in the data set while the green box covers the point that the MATLAB SDR code was able to acquire the signal again, and thus can be considered the point where the jamming is no longer fully interfering with the receiver and normal GPS operation can resume. P1 is the first point taken, and is recorded when the receiver is right next to the NEAT [12] device. At this point the larger variance in the plots is due to

[1] see table5.1

the RF AGC attempting to compensate for the NEAT while the IF AGC is fully saturated. Once the RF AGC reaches flatline saturation the IF AGC begins climbs in a predicable manner. The stepwise function visible is due to the NT1065 having a 3 dB resolution for AGC measurements leading to a smoother plot then the SiGe The NT1065 was tested over much less distance owing to it having less accuracy in its markings.

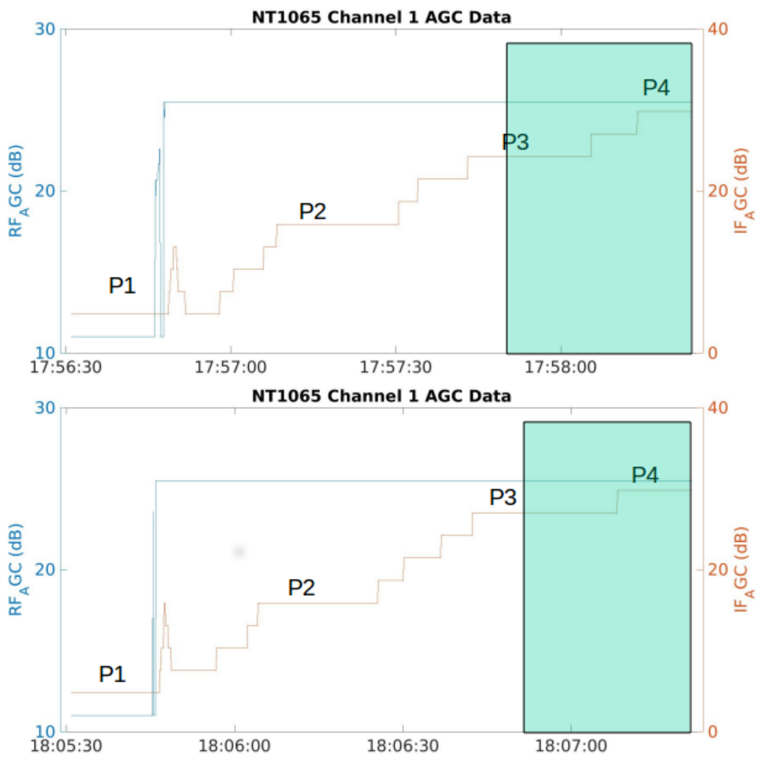


Figure 5.1: (Top to bottom) NT1065 tests NT1 and NT3 L1 Only. The green bar indicates where the SDR receiver is capable of processing the GPS data again, and thus the point where the system can be considered no longer interfered with. The long flat sections indicate the areas where the vehicle was stopped.

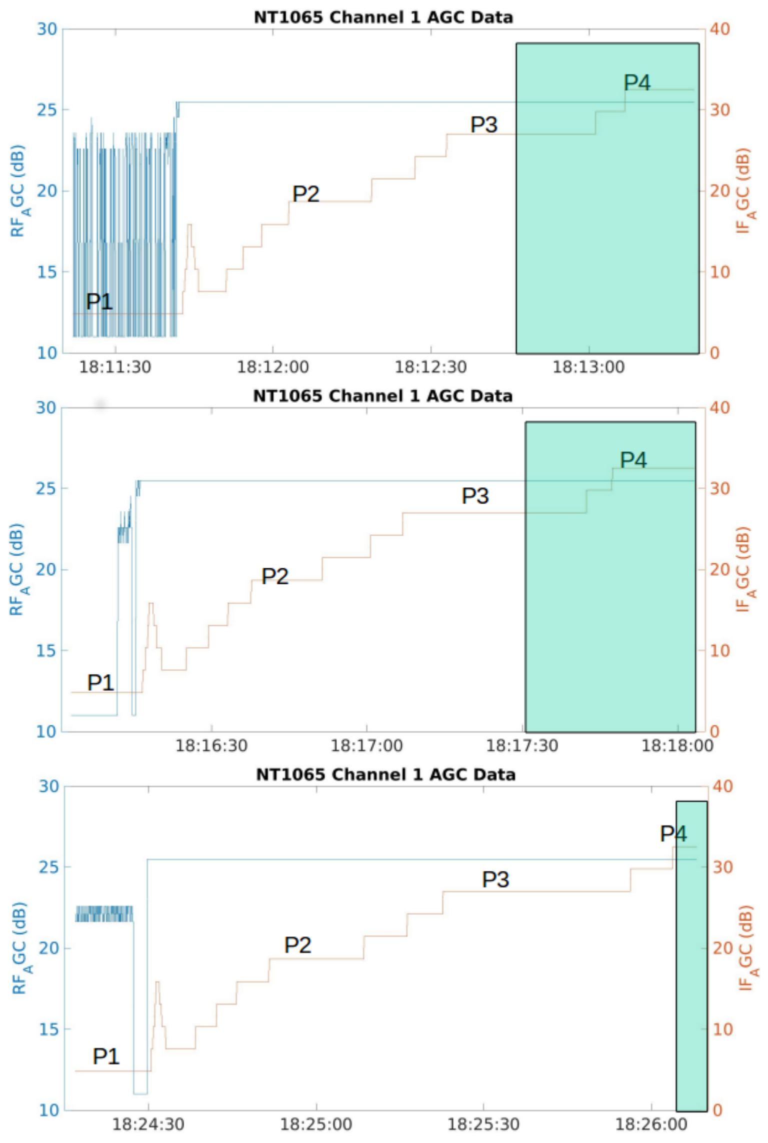


Figure 5.2: (Top to bottom) NT1065 tests NT4, NT5, and NT6 L1 Only The green bar indicates where the SDR receiver is capable of processing the GPS data again, and thus the point where the system can be considered no longer interfered with. The long flat sections indicate the areas where the vehicle was stopped.

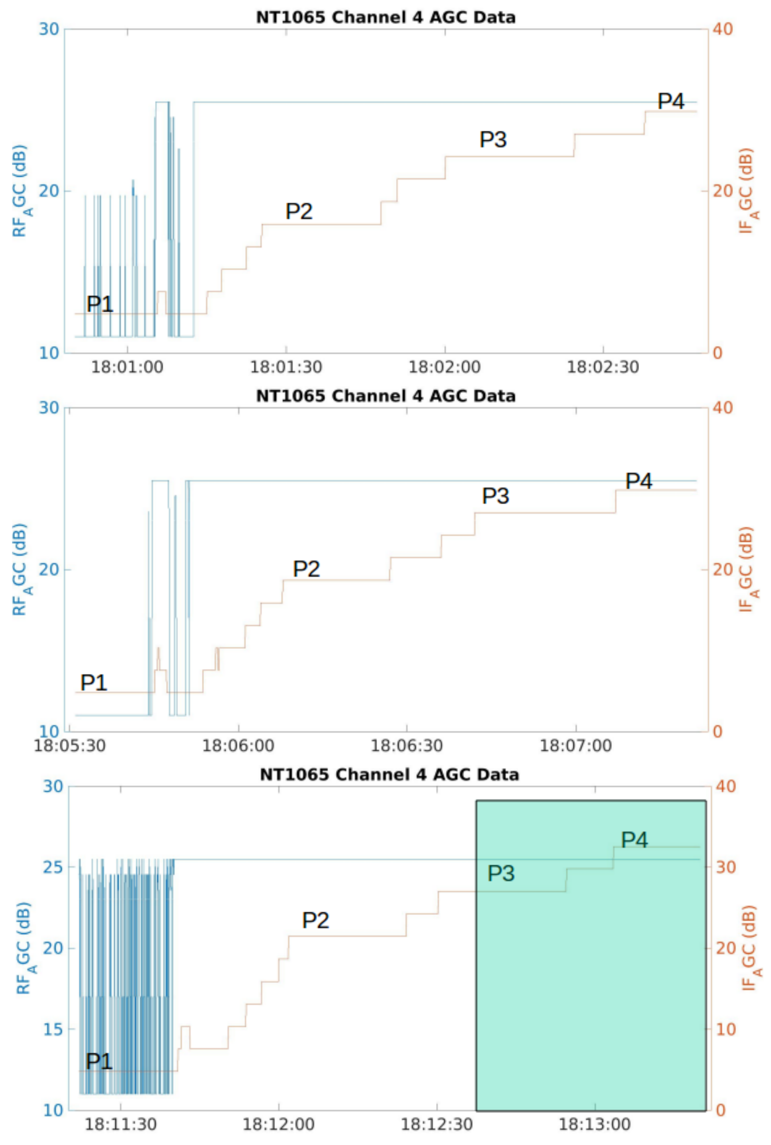


Figure 5.3: (Top to bottom) NT1065 tests NT2, NT3, and NT4 L2 Only The green bar indicates where the SDR receiver is capable of processing the GPS data again, and thus the point where the system can be considered no longer interfered with. The long flat sections indicate the areas where the vehicle was stopped.

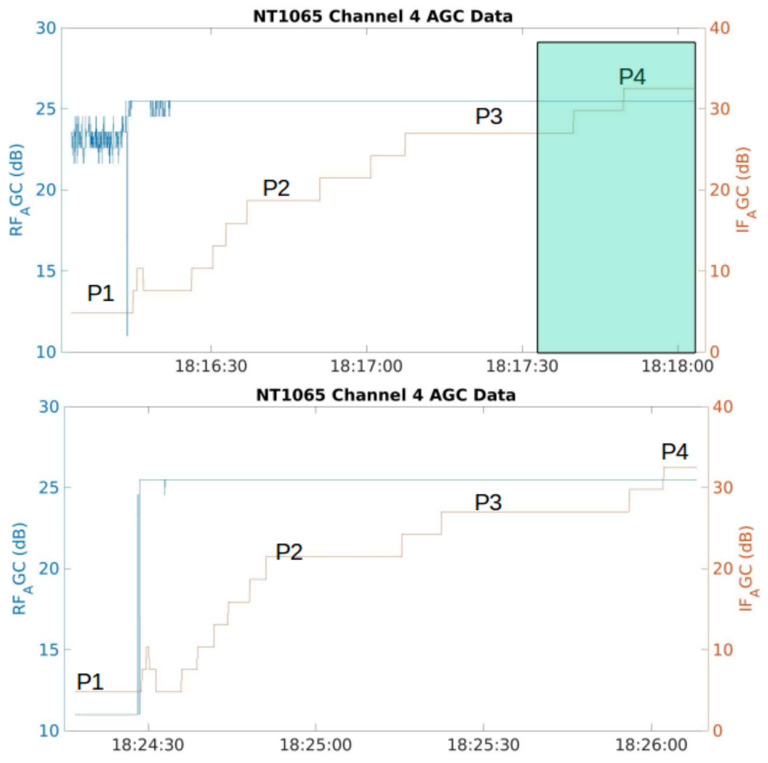


Figure 5.4: (Top to bottom) NT1065 tests NT5 and NT6 L2 Only The green bar indicates where the SDR receiver is capable of processing the GPS data again, and thus the point where the system can be considered no longer interfered with. The long flat sections indicate the areas where the vehicle was stopped.

5.2 Future work

While the system is functional at its current implementation there is a large amount of future work for the new NT1065 implementation, as well as a bit of testing for the SiGe. Initially, the NT1065 needs to be tested in more scenarios as it currently has only collected data from one GPS jammer. These scenarios should ideally do more with moving the jammer around after it has been turned on as well as attempt to jam on both L1 and L2 simultaneously and see what the positioning results for that are. Once the NT1065 has a better set of test results, the next step is to extend the low-cost receiver work properly. Some tests that need to be done include moving the receivers before the jamming starts as well as turning on multiple jammers inside the zone, both L1, and L2 as well as leveraging all of the different signal options. This testing should be done with both the SiGe and the NT1065 in order to provide more comparison points between the two. Once all of that is done, the NT1065 should be extended to looking at GLONASS effects as well as spoofing detection. This will require a bit of a redesign of the code base in order to be more sensitive to AGC effects, and for the TDOA when dealing with untested GPS signals.

For the next generation of both hardware and software, there is a large amount of potential future work that still needs to be done. The largest change is that the hardware package is being redesigned into a single board for ease of future work. This step is critical to creating a field ready version of the ASIC. This single board change would help address the boot issues as well as the programming load issues. The next known large change in the future is the addition of resampling on the Zedboard. This resampling will lower the volume of data passing through the FX3 in order to relieve some of the hard processing constraints on the system right now. The Zedboard has both an ARM processor and a full FPGA on board, both of which are capable of substantial processing. Augmenting the processing capabilities of the host PC is essential to extend the functionality of the ASIC for future work. The final area that needs major future development is the network code. While the system is able to handle the data loads in a controlled test environment, it is essential to build out a full four receiver system and test it in a field condition. After it has been confirmed function or had any lingering issues fixed, the system should be tested for both range and reliability over long tests. While the network will not be saturated with continuous data for very long, multiple triggers over the day will help verify that the system is fully functional.

- Redesign hardware into a single board
- Add resampling to the Zedboard
- Fully test network code
- Long duration tests
- GLONASS interference

Chapter 6

Conclusions

6.1 Learnings

There were several challenges encountered in the development of this project. Chief among them was setting up the NT1065 development environment and in successfully applying the TDOA algorithm to the NT1065 results. The NT1065 development environment was a very new system which had large amounts of instability still within it. While it wasn't that difficult to get the initial demo code built, running it created many problems. These problems included unreliable startup code, and that the system would not work if it wasn't started in exactly the correct configuration. Solving both of the problems required extensive testing and validation. One thing that took a long time to discover was that once the NT1065 was flashed with firmware for the first time, it wouldn't accept any new firmware changes until it was fully power-cycled. The code failed to give any indication of this, and upon further inspection, it was found to be difficult to check without reworking the firmware to include a version number that would have to be checked every startup. This is because the firmware would simply return success for a flashing operation, regardless of if anything had been changed.

The next larger challenge encountered in the development of the system was the NT1065 boot order requirements. Since the Zedboard is required to boot the initial firmware onto the FX3 board and then load the initial register connections for the NT1065, there is a strict timing requirement on the initialization. This timing requirement manifests in the requirement that the Zedboard is powered on before the FX3 board, but that the FX3 board has to be powered on before the Zedboard attempts to load the firmware. This timing difference is measured in tens of milliseconds and thus the best solution is to keep the FX3 in soft reset until the Zedboard is turned on, then quickly bring the FX3 board out of reset. if done correctly the software will load fully and the NT1065 will be able to load its firmware from the host PC.

6.2 Discussion

In conclusion, the changes to the SiGe and the NT1065 code worked as expected as seen in the results. The goal of taking the older code base and adapting it to newer hardware and software, as well as the implementation of the newer real time server client both performed better than expected. The code functions as a real time detector of GPS interference for both L1 and L2. Additionally, the NT1065 code behaves to similar levels to the SiGe giving a future platform to develop on top of for scenarios of testing.

The real time code as implemented supports large numbers of SiGe receivers working in concert, with the only upper limitations being set based on network capacity and the server's disk usage. It allows trigger events to propagate out to all of the clients as well as allowing the server to process the data using the TDOA and PDOA codes. Ideally, in the future, the real time code will be tested on a more dynamic network, and the live updating display will see practical use in the field.

As seen from the results chapter the NT1065 and SiGe were both capable of positioning the receiver correctly. While the NT1065 still has some issues to work out with its final accuracy, both results were very accurate, and the NT1065 was able to detect L2C jamming as well as L1. This shows great promise for future development and use of this tool.

Bibliography

[1] Dennis M Akos. Who's afraid of the spoofer? gps/gnss spoofing detection via automatic gain control (agc). Navigation, 59(4):281–290, 2012.

[2] Jahshan A Bhatti, Todd E Humphreys, and Brent M Ledvina. Development and demonstration of a tdoa-based gnss interference signal localization system. In Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION, pages 455–469. IEEE, 2012.

[3] Kai Borre, Dennis Akos, Nicolaj Bertelsen, Peter Rinder, and Soren Jensen. A Software-Defined GPS and Galileo Receiver. 2007.

[4] Alexis Bose. Is it possible to build a low-cost system to detect and locate a single gnss jammer in near-real time? insidegnss, page 32âĂŞ37, 2017.

[5] Guy Buesnel. Gps disruption is a growing problem for aviation, reports show. 2016.

[6] Luca Canzian, Samuele Fantinato, Stefano Ciccotosto, Andrea Chiara, Giovanni Gamba, Oscar Pozzobon, Rigas Ioannides, and Massimo Crisci. Interference localization from space theoretical background. Inside Gnss, page 59âĂŞ68, 2016.

[7] Luca Canzian, Samuele Fantinato, Stefano Ciccotosto, Andrea Chiara, Giovanni Gamba, Oscar Pozzobon, Rigas Ioannides, and Massimo Crisci. Interference localization from space applications. Inside Gnss, page 56âĂŞ67, 2017.

[8] Chun-Kai Feng, Shau-Shiun Jan, Thomas Johnson, and Dennis Akos. Assessment of camera capture for gps rfi monitor. In Position, Location and Navigation Symposium-PLANS 2014, 2014 IEEE/ION, pages 1272–1281. IEEE, 2014.

[9] Giovanni Gamb, Andrea Dalla Chiara, Oscar Pozzobon, and Damien Serant. Progress project: Jamming and spoofing detection and localization system for protection of gnss ground-based infrastructures. In Proceedings of the 29th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2016), pages 3133–3142, September 2016.

[10] Rama Sai Krishna. Differences in implementation of ez-usbÂő fx2lpâĎć and ez-usb fx3âĎć applications.

[11] D.W. Lim, H.W. Kang, M.B. Heo, H.H. Choi, and S.J. Lee. Implementation of a jammer localization system based on tdoa/aoa algorithm. Proceedings of the ION 2013 Pacific PNT Meeting, pages 406–410, Honolulu, Hawaii, 2013.

[12] Rod MacLeod. Novatel's gnss vulnerability mitigation. NovAtel Proprietary.

[13] Ryan Mitch, Mark Psiaki, and Tunc Ertan. Chirp-style gnss jamming signal tracking and geolocation. Navigation, 63(1):15–37, 2016.

[14] Jean-Paul Poncelet and Dennis M. Akos. Design of a low-cost monitoring station for l1 interference detection and localization.

[15] Jean-Paul Poncelet and Dennis M Akos. A low-cost monitoring station for detection & localization of interference in gps l1 band. In Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing,(NAVITEC), 2012 6th ESA Workshop on, pages 1–6. IEEE, 2012.

[16] Mark L Psiaki. Developing defenses against jamming & spoofing of civilian gnss receivers. In Proceedings of the ION GNSS Meeting, pages 3407–3417, 2011.

[17] Daniel Shepard. Characterization of receiver response to spoofing attacks. PhD thesis, 2011.

[18] Dong-Ho Shin, Seok-Bo Son, and Tae-Kyung Sung. Dop relationship between the toa and the tdoa positioning. In Proceedings of the IAIN World Congress and the 56th Annual Meeting of The Institute of Navigation, pages 436–442, 2000.

Appendix A

Using the tool

A.1 How to build

This code requires qmake, build-essentials, libusb, and boost to build. Additionally requires git and a connection to phabricator for the code.

(1) git clone <phabricator>/nt1065_code

(2) qmake AmungoFx3Dumper.pro

(3) make clean; make

(4) Check for the existance of AmungoFx3Dumper binary.

A.2 How to log data

(1) Ensure that the Zedboard has the correct firmware on the SD card

(2) Plug in the NT1065 and the FX3 board into the Zedboard.

(3) Plug the Zedboard into the wall and the FX3 board into the computer

(4) While holding down the two buttons on the FX3 board power on the Zedboard. After the green light comes on but before the blue or red lights come on remove your fingers from the FX3.

(5) If done correctly the Zedboard should have the green and blue lights solid with the red light blinking. There should be no lights blinking on the FX3. If there are repeat the process, paying close attention to the timing.

(6) Run the AmungoFX3Dumper binary with the chosen flags. Note that once the boards is programmed once it requires a full reboot to program again.

A.3 Troubleshooting

- If you get a board overflow: Full reboot + Possibly faster computer needed. Check connection is USB3
- If it fails with an error about the programmer reboot and attempt to reprogram. The timer was incorrect