EMBRY-RIDDLE
Aeronautical University™
SCHOLARLY COMMONS

Doctoral Dissertations and Master's Theses

Summer 8-2017

A GPS Signal Generator Using a ROACH FPGA Board

Kurt L. Pedrosa
Embry-Riddle Aeronautical University

A GPS Signal Generator Using a ROACH FPGA Board

A Thesis submitted in partial fulfillment of
the requirements for the award of the degree of

Master of Science
in
Electrical and Computer Engineering

Submitted by
Kurt L. Pedrosa

Under the guidance of
Dr. William C. Barott

EMBRY-RIDDLE
Aeronautical University
DAYTONA BEACH, FLORIDA

DEPARTMENT OF ELECTRICAL, COMPUTER, SOFTWARE, AND SYSTEMS ENGINEERING

RADAR & MICROWAVES LABORATORY

Daytona Beach, Florida - USA

August 2017

Committee Approval

"A GPS Signal Generator Using a ROACH FPGA Board"

Kurt L. Pedrosa

This thesis is prepared under the direction of the candidate's thesis committee chairman, Dr. William C. Barott, Department of Electrical, Computer, Software, and Systems Engineering, and has been approved by members of his thesis committee. It is submitted to the Electrical, Computer, Software, and Systems Engineering Department in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering.
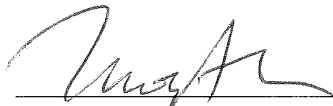
Dr. William C. Barott
Committee Chairman

Brian Butka Digitally signed by Brian Butka
Date: 2017.08.23 12:00:54
-04'00'

Dr. Brian Butka
Committee Member

Dr. Richard Stansbury
Committee Member
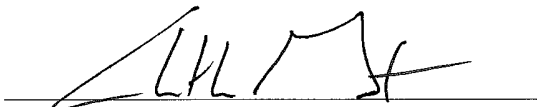
Dr. Tim Wilson
Department Chair

24 Aug 2017
Date

Dr. Maj Mirmirani
Dean of College of Engineering

8/24/2017
Date

Dr. Christopher Grant
Vice Chancellor for Academics

8/25/17
Date

Embry-Riddle Aeronautical University

Abstract

Master of Science in Electrical and Computer Engineering

A GPS Signal Generator Using a ROACH FPGA Board

by Kurt Pedrosa

Dr. William C. Barott, Dr. Richard Stansbury, Dr. Brian Butka
Department of Electrical, Computer, Software, and Systems Engineering

A Global Positioning System (GPS) signal simulator is a valuable testing tool. It allows for testing of GPS receivers, systems, and anti-spoofing algorithms. With the increased popularity of software defined radios (SDRs) merging GPS signal simulators and SDRs is a natural choice. A detailed review of the construction of a GPS signal generator using the ROACH processing board is presented herein. The ROACH, developed by the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) team, is a processing board that can be configure to function as a SDR. In this research, the ROACH was transformed to function as a GPS signal generator able to transmit the C/A L1 civilian GPS signal. Maximum manipulation of the GPS signal was built-in to the firmware allowing the user to change the signal for different applications. Its modular architecture and ease of reproduction makes this GPS signal simulator design a viable research tool in the field of GPS anti-spoofing, GPS system fabrication, and as a great GPS educational tool.

The GPS signal generator presented herein simulates the GPS C/A L1 signal at a frequency of 50.127 MHz. The generated signal contains no time delay or Doppler shift. A total of four independent GPS signals can be generated and transmitted as a single composite signal. The signal generator is capable of generating all of the current NAVSTAR defined PRN sequences allowing the simulation of any four satellite combination. It also uses the most up-to-date almanac data in the transmitted signal.

Future improvements to this GPS signal generator includes development of an up-converter to convert the transmitted signal frequency to the C/A L1 signal frequency of 1575.42 MHz, implementation of Doppler shift and time delay logic to the firmware, and software front end providing the user the ability to enter trajectory coordinates used to generate dynamic GPS signal along defined trajectory.

Acknowledgements

As this stressful but extremely rewarding period comes to an end the time has come to give acknowledgments to those who cheered me on every step of the way. I would first like to thank my thesis advisor, Dr. William C. Barott at Embry-Riddle Aeronautical University. He has been a beacon of strength, a well of patience. He has guided me every step of the way and set me up for a bright professional future. He has made me a life long student excited about discovery, learning, and curiosity.

To the Electrical, Computer, Software, and System (ECSSE) department at Embry-Riddle Aeronautical University I extend my deepest appreciation. With special mention of Dr. Brian Butka and Dr. Richard Stansbury for their support on this thesis research. To Dr. Timothy Wilson and Professor Farahzad Behi for all of their help and patience. As my days end as a student in the Department the memories and growth gained will forever remain. My time spent there will forever be in my heart.

Most importantly I would like to acknowledge the three women of my life. My Grandmother, Reny De Lima Torres, who at 89 years old continues to be the biggest trouble maker in the family. To my Mother, Sheila Tecchio; Your perseverance, strength, and determination inspires me. The man I am today is because of her. To my Wife, Michelle Dos Anjos Pedrosa, who is my biggest cheerleader. She endured countless lonely days and nights while I finished my thesis. All I am is hers, who I became will be because of her.

Contents

List of Figures

List of Tables

Abbreviations

ADC	Analog to Digital Converter
CASPER	Collaboration for Astronomy Signal Processing and Electronics Research
C/A	Coarse Acquisition Code
CTTC	Centre Tecnològic de Telecomunicacions de Cataluny
BPSK	Binary Phase Shift Keying
BRAM	Block Random Access Memory
DAC	Digital to Analog Converter
DGPS	Differential Global Positioning System
DOD	Department of Defense
DOT	Department of Transportation
DSSS	Direct Sequence Spread Spectrum
DTV	Digital Television
FAA	Federal Aviation Administration
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
GNSS	Global Navigation Satellite System
GRC	GNU Radio Companion
GUI	Graphical User Interface
HDL	Hardware Discriptive Language
HOW	Handover Word
ILS	Instrument Landing System
IMU	Inertial Measuring Unit
IP	Internet Protocol
ISF	Integrity Status Flag
LAN	Local Area Network
LO	Local Oscillator
LSB	Least Significant Bit
LUT	LookUp Tables
MEO	Medium Earth Orbit
MUX	Multiplexer
MSB	Most Significant Bit
NASA	National Aeronautics and Space Administration
NDGPS	Nationwide Differential Global Positioning System
NMEA	National Marine Electronics Association
NUDET	Nuclear Detonations
PC	Personal Computer
PPS	Precise Position Service

PRN	**P**seudo **R**andom **N**oise
PVT	**P**osition, **V**elocity, and **T**ime
P(Y)	**P**recision code
RF	**R**adio **F**requency
RINEX	**R**eceiver **I**ndependent **E**xchange **F**ormat
ROACH	**R**econfigurable **O**pen **A**rchitecture **C**omputing **H**ardware
SDR	**S**oftware **D**efined **R**ardio
SPS	**S**tandard **P**osition **S**ervice
SV	**S**atellite **V**ehicle
SIS	**S**ignal-**i**n-**s**pace
TCP	**T**ransmission **C**ontrol **P**rotocol
TLM	**T**elemetry
TOW	**T**ime **o**f the **W**eek
UHF	**U**ltra **H**igh **F**requency
UTC	**C**oordinated **U**niversal **T**ime
USCG	**U**nited **S**tates **C**oast **G**uard
USNDS	**U**nited **S**tates **N**uclear Detonation **D**etection **S**ystem
VOR	**V**ery High Frequency **O**mnidirectional **R**ange
WAAS	**W**ide **A**rea **A**ugmentation **S**ystem
WMS	**W**ide Area Augmentation System **M**aster Station
WRS	**W**ide Area Augmentation System **R**eference Station

Chapter 1

Introduction

This chapter discusses the motivations that led to the work of this thesis research on GPS (Global Positioning System) signal simulation. It further presents an outline of this thesis document.

1.1 Motivation

A GPS signal generator is a valuable educational, research, and development tool. As a research and development tool it can be used to test new anti-spoofing algorithms, new GPS navigation systems, GPS dependent systems, and many others. As an educational tool it can be used to teach many subjects including the GPS signal structure, provide hands-on experience with signal transmission and acquisition, and provide live demonstration of signal processing applications. GPS signal generators can cost upwards of $10,000, a price out of range for many research laboratories, universities, and enthusiasts. Naturally a demand exists for an inexpensive, modular system capable of generating a GPS signals for a wide range of applications.

GPS is the primary form of air, sea, and ground navigation used across all industries and virtually every part of the world. Problems like spoofing, signal

multipath, and interference must be analyzed. A well-designed GPS signal simulator with a modular architecture fulfills not only nearly every research area but it can also be used as an educational tool.

To assist with GPS signal research and to promote education in this area were not the only motivators for this thesis work. A driving curiosity for this thesis work was founded by a 2013 experiment by Todd Humphreys, professor and researched at University of Texas at Austin, where he and his team successfully spoofed a yacht's navigation system [1]. A quote from Humphreys where he said "The ship actually turned and we could all feel it, but the chart display and the crew saw only a straight line" really sparked a deep curiosity for GPS. This curiosity was the driving motivator for this research project.

Although a GPS signal simulator is not a spoofing device it can be used to test anti-spoofing algorithms. The device presented herein attempts to set a foundation for future research by providing a robust GPS signal simulator with a modular architecture that can be configured to nearly all applications. The Radar and Microwaves Laboratory focus on passive radar research and the tools used for this research were available as resources. Given the hardware and software tools available in the laboratory, a signal simulator architecture design that utilizes those tools is put forth.

1.2   Thesis Organization

The organization of this thesis document attempts to build up on the information presented. Chapter 1 conducts a background discussion of the GPS signal characteristics, satellite orbital description, ground and air support systems, and augmented GPS. The methodology, chapter 2, details the procedures used in the construction of the GPS signal simulator. It starts by first describing the firmware,

then the software design, followed by a GPS software decoder and signal acquisition procedures. Lastly, the results found during the construction of the GPS signal simulator are provided in chapter 4. A brief conclusion and discussion about possible future works closes out the document.

Chapter 2

Background and Theory

2.1   Global Positioning System

The NAVSTAR Global Positioning System, known as GPS, is a satellite navigation system owned and deployed by the United States of America. Its main purpose is to provide its users with navigation, positioning, and timing services. The need for such a system was proposed by several U.S. government organizations including the Department of Defense (DOD), the National Aeronautics and Space Administration (NASA), and the Department of Transportation (DOT) [2]. Although the current user base is both civilian and military, the initial objective for GPS was to provide the U.S. military with three-dimensional position determination. It was only after a civilian aircraft was shot down by the USSR Air Defense, killing 269 people, for deviating from its original route and flying into Soviet airspace [3] that President Ronald Reagan allowed GPS to be used in civilian applications.

The GPS program was launched in the early 1970s [4] with the first 11 GPS satellites launched between 1974 and 1985. These satellites were Block I satellites used to validate the concept and test various parts of the system. After the initial 11 Block I satellites the Air Force launched nine Block II satellites between 1989 and 1990, followed by 19 Block IIA satellites between 1990 and 1997, and 13 Block

IIR satellites launched between 1996 and 2004 [5]. Now, GPS is fully operational and meets all of the criteria put forth in the 1960s when it was first proposed. It currently is a dual-use system, providing the the Standard Position Service (SPS) for all civilian applications and Precise Position Service (PPS) for military use. The current GPS constellation consists of 31 active satellites, as of the publication of this thesis, in six orbital planes. Each plane has four to six satellites. The GPS constellation, shown in figure 2.1, is arranged in such a way that the user can expect between six to ten visible satellites at any point in time or position on earth.
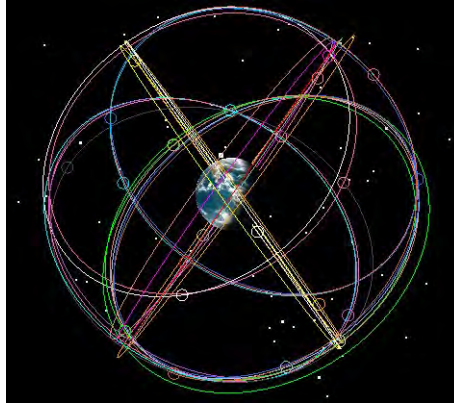
Figure 2.1: GPS Constellation Map [6]

There are three segments that make up GPS: the space segment, control segment, and the user segment. The space segment was initially composed of a 24-satellite constellation arranged into six equally-spaced orbital planes [7]. The constellation expended three of the 24 slots and repositioned six satellites to make room for the addition of three new satellites. This occurred in June of 2011 and was known as the "Expandable 24" expansion. The current active GPS constellation consists of 31 total satellites under full operational capability (FOC) [8]. The 31 satellites include 27 used for GPS, three reserve satellites, and one test satellite. The expansion from 24-slot to 27-slot constellation improves coverage in parts of the world and ensures users are able to receive signals from at least four satellites from virtually any point on earth [9].

The control segment is charged with tracking all 31 satellites, monitoring their transmissions, and transfers data to the entire constellation. Ground facilities spread around the world make up the control segment. The location of these facilities, seen on figure 2.2, were strategically chosen to monitor the satellites with minimal downtime. The Master Control Station, located in Colorado, is charged with conducting the primary control segment functions. The Monitor Stations track the satellites as they pass overhead. The 11 ground antennas position around the world are used to communicate with the satellites.

The user segment is made up of the GPS receivers able to use the GPS signal to determine their position, velocity, and time. The use of GPS has spread among virtually every industry. From agriculture to sports, surveying to emergency service, GPS has become an integral part of our daily lives. As such, the development of the GPS signal generator presented in this research helps test such applications.
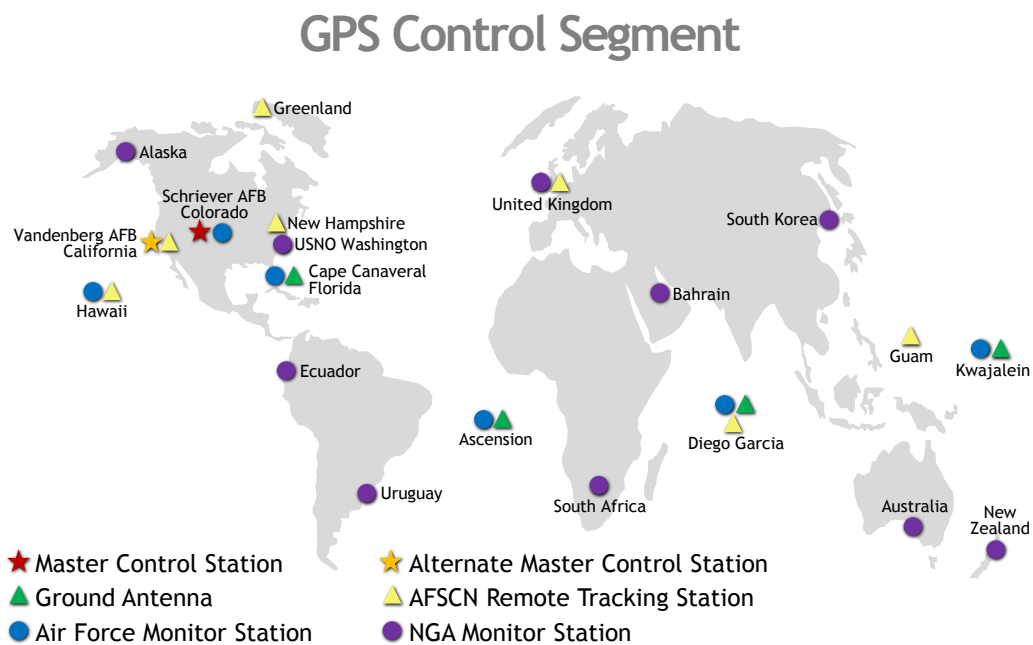
Figure 2.2: GPS Control Segment Map [7]

6

2.2 Orbital Description

GPS satellites operate in medium Earth orbit (MEO) at an altitude of about 20,000 km above the Earth's surface. Each satellite orbits around earth twice a day. MEO is characterize by an orbital altitude of 2000 km to 35,790 km, which is where most Global Navigation Satellite Systems (GNSS) operate. GPS satellites operate in six approximately-circular orbital planes and have an orbital period of 11 hours and 58 minutes. These orbital planes form an elliptical path focused around the center of the Earth. As part of the navigation message GPS provides the receivers with orbital description and satellite coordinates describing the position of the satellites within the orbital path. Accurate information about the satellites' position is needed for a GPS receiver to determine its position on Earth.

The forces acting on a satellite are the Earth's gravitational field, other bodies in space like the Sun, the Moon and solar radiation pressure [10]. The major force is Earth's gravitational field and it can be described using Newton's laws where the force acting on the satellite is described in equation 2.1. Force F is the product of the acceleration of the satellite a and its mass m. Then, the force is found by

$$F = ma = -G\frac{mM}{r^3}r \tag{2.1}$$

using the radial r, gravitational constant G, and Earth's mass M. The minus sign defines the attractive nature of the gravitational force [2]. Solving for the satellite's acceleration by taking the second derivative of the position solves to equation 2.2, where $\mu = G \cdot M$

$$\frac{d^2r}{dt^2} = -\frac{\mu}{r^3}r \tag{2.2}$$

This equation is known as two-body or Keplerian motion equation where the only force acting on the satellite is the Earth [2] and describes the acceleration of the satellites with respect to Earth. A satellite, an object in motion in three-dimensional space, has a position vector and a velocity vector each with x, y, and z component which dictates the use of six parameters to describe its trajectory. These parameters are known as Keplerian elements defined as semi-major axis, a, eccentricity, e, inclination, i, right ascension of the ascending node, Ω, argument of perigee, ω, and true anomaly, v. Figures 2.3a and 2.3b show four of the six orbital elements in reference to the orbital plane and the reference plane. The two orbital elements not shown in the figures due to graphical constraints are e and a.

The shape and size of the orbit are described by e and a. The e element is a number between 0 and 1 that describes the amount the orbit deviates from a perfect circle where $e = 0$ is a perfect circle. The a element is the mean between the apogee, a point on the orbital path furthest from the Earth's center, and the perigee, a point on the orbital path closest to the Earth's center. Equation 2.3 shows how e relates to the a and the semi-minor axis b. The orbital period T_p, which is time that it takes the satellite to complete one orbit around the Earth, of the satellite is calculated by equation 2.4. The orbital period has a direct relationship to the orbital altitude where the orbital altitude is the difference between the Earth's radius and the a.

$$e = \sqrt{1 - \frac{b^2}{a^2}} \tag{2.3}$$

$$T_p = 2\pi\sqrt{\frac{a^3}{\mu}} \tag{2.4}$$
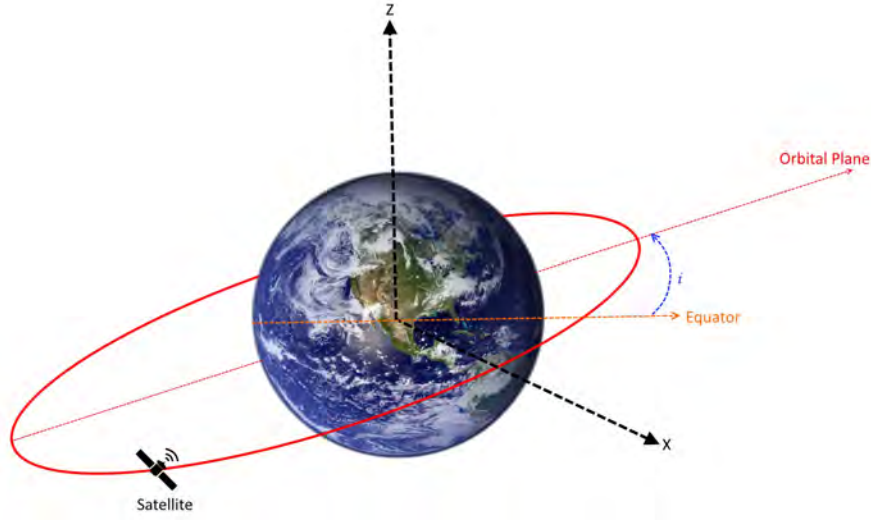
(a) Inclination between orbital plane and reference plane
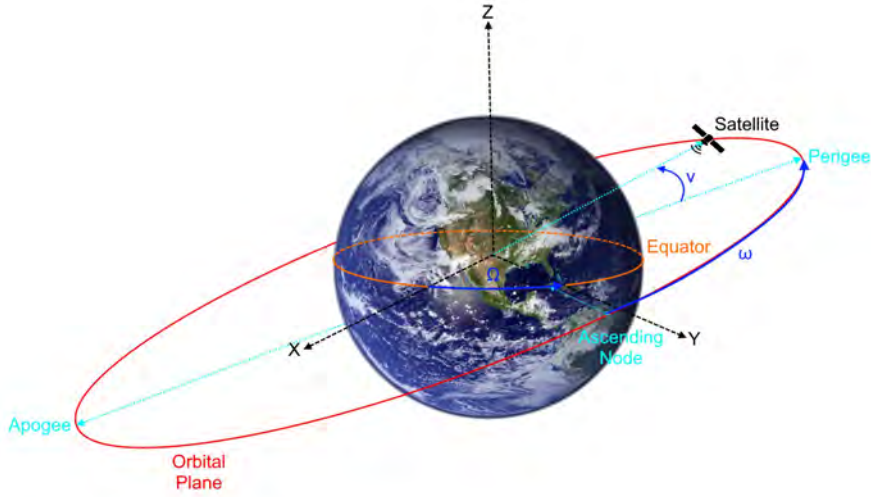
(b)

(c) Three orbital elements, ω, v, and Ω

Figure 2.3: Keplerian orbital elements

The orientation of the orbital plane is described by i, Ω, and ω. The position of the satellite is defined by v. The GPS receiver needs each of the six orbital planes for all of the satellites in the constellation. This information together with timing and angle corrections are contained in the ephemeris data encoded in the downlink message. A full list of the ephemeris data as defined by the GPS documentation is listed in table 2.1. Ephemeris data contain Keplerian elements containing periodic terms that correct gravitational perturbations.

t_{0_e}	Reference time of ephemeris
$\sqrt{a}$	Square root of the semi-major axis
e	Eccentricity
i_0	Inclination angle a reference time
Ω_0	Longitude of the ascending node
ω	Argument of perigee
M_0	Mean anomaly
$\dot{\Omega}$	Rate of change of longitude of the ascending node
Δn	Mean motion correction
C_{u_c}	Amplitude of cosine correction to argument of latitude
C_{u_s}	Amplitude of sine correction to argument of latitude
C_{r_c}	Amplitude of cosine correction to orbital radius
C_{r_s}	Amplitude of sine correction to orbital radius
C_{i_c}	Amplitude of cosine correction to inclination of angle
C_{i_s}	Amplitude of sine correction to inclination of angle

Table 2.1: Ephemeris data definitions

2.3 Almanac and Ephemeris

The GPS almanac contains low-resolution ephemeris data about every satellite in the GPS constellations. Each satellite transmits almanac data for all satellites but transmits only its own ephemeris data. The almanac data includes satellite states (e.g., health), coarse ephemeris, an ionospheric model, and timing correction information. Both the almanac data and ephemeris data are contained within the GPS message signal which will be furthered discussed later in this chapter. The difference between ephemeris and almanac is one of detail: whereas ephemeris data is a detailed description of the satellite's orbital and timing characteristics while almanac data is a general description. Ephemeris data are typically valid for 4 hours from the time of transmission while the almanac data can be valid for several weeks if no significant changes occur in the constellation [11]. It is important to acknowledge the separation between the validity of both ephemeris and almanac data and the transmission of that data by the satellites. Due to the characteristics of the message signal (further discussed in this section), each

satellite transmits ephemeris data every 30 seconds, but requires 12.5 minutes to transmit the complete almanac. This means that a GPS receiver can receive updated ephemeris data every 30 seconds but it would take roughly 12.5 minutes to attain all of the almanac data.

The control segment generates a new almanac once a day. The new almanac is transmitted to each satellite during its next communication with the control segment. The United States Coast Guard (USCG) logs almanacs in a public database [11]. The almanac data is saved in two formats called YUMA and SEM. Although both YUMA and SEM contain the same almanac data, the major difference is how the data are formatted. A YUMA almanac file separates the data in different lines, SEM clumps data together in the same line but different column. This formatting difference can be seen in figure 2.4, which shows a line-by-line snippet of the first satellite in the almanac.

```
******** Week 933 almanac for PRN-01 ********
ID:                        01
Health:                    000
Eccentricity:              0.6873130798E-002
Time of Applicability(s):  405504.0000
Orbital Inclination(rad):  0.9682738402
Rate of Right Ascen(r/s):  -0.7988904198E-008
SQRT(A)  (m 1/2):          5153.634277
Right Ascen at Week(rad):  0.3013838569E+001
Argument of Perigee(rad):  0.581413922
Mean Anom(rad):            0.2251446372E+000
Af0(s):                    0.5722045898E-004
Af1(s/s):                  0.0000000000E+000
week:                      933
```

```
31  CURRENT.ALM
 933 405504

1
63
0
 6.87313079833984E-03  8.21113586425781E-03  -2.54294718615711E-09
 5.15363427734375E+03  9.59334611892700E-01   1.85069799423218E-01
 7.16657638549805E-02  5.72204589843750E-05   0.00000000000000E+00
0
11
```

(a) YUMA almanac snippet   (b) SEM almanac snippet

Figure 2.4: Different Almanac Formatting of Satellite with PRN 1

For this project the YUMA almanac format was chosen for two reasons. First the YUMA almanac file is more "human readable," meaning that the description of each data provided is listed; the SEM does not define the data in the file itself. Second it was faster to parse each line individually instead of parsing lines and spaces, as would have required with the SEM almanac format. However an advantage of using the SEM is that the data provided is already in the units of

semi-circles or *semi-circles per seconds* while the YUMA almanac file kept all of its data in *radians* or *radians per second*. Using YUMA almanac required a conversion to the NAVSTAR standard, *semi-circles* an arbitrary choice that had no effect on the overall simulator quality.

2.4 GPS Signal Characteristics

The GPS satellites transmit navigation signals on two carrier frequencies called L1 and L2, both transmitted in the Ultra High Frequency (UHF) band that spans from 500 MHz to 3 GHz [12] and in particular in the L-band (1-2 GHz). These frequencies are derived from a single nominal reference frequency, f_0, at 10.23 MHz. Calculations for L1 and L2 are:

$$f_{L1} = 154 \times f_0 = 1575.42 \quad \text{MHz} \tag{2.5}$$

$$f_{L2} = 120 \times f_0 = 1227.60 \quad \text{MHz} \tag{2.6}$$

The carrier frequencies, f_{L1} and f_{L2}, contain GPS navigation data and uses three unique spreading sequences. The navigation data contains the information used by the GPS receivers to calculate position. The spreading sequence is a modulation method that widens the signal's bandwidth deliberately spreading it in the frequency domain. This type of modulation is typically done to ensure a secure communication, to decrease interference by other signals, or to prevent detection. The three spreading sequence used by GPS are the coarse acquisition code (C/A), precision (P) code, and the Y-code. The Y-code and P code are encrypted and referred to as P(Y) code, its encryption can only be decrypted by users with valid decryption key. The C/A code is not encrypted and for that reason it is used in civil applications making it the most widely used GPS signal. The C/A code

is also used for acquisition of the P(Y) code by users with access to the signal. The frequency f_{L1} is modulated by both C/A and P(Y) code because until 2000 Selective Availability, a GPS mode that denied full accuracy of GPS to SPS users, was still active and required the P code encoding to deny full access to the signal. Frequency f_{L2} is modulated only by the P(Y) code and used by the PPS users. Interplexing scheme combines the C/A, P(Y) and the message data on a common carrier while keeping the signal envelope the same [13]. The transmitted signal is composed of two orthogonal components and can be written as:

$$S_{L1} = A_{P(Y)} \cdot P(t) \cdot D(t) \cdot cos(2\pi \cdot f_{L1} \cdot t) + A_{C/A} \cdot C(t) \cdot D(t) \cdot sin(2\pi \cdot f_{L1} \cdot t) \quad (2.7)$$

where the signal transmitted by the L1 channel, S_{L1}, is the sum of two orthogonal signals where $A_{P(Y)}$ is the amplitude of the P(Y) code; $P(t) = \pm 1$, P(t) is the P(Y) code sequence, $A_{C/A}$ is the amplitude of the C/A code; $C(t) = \pm 1$, D(t) is the message data code; $D(t) = \pm 1$, and C(t) is the C/A code sequence. The minimum L1 signal power level measure at the receiver are -133 dBm for P(Y) code and -130 dBm for the C/A code [14]. Typically the GPS receiver expects a L1 signal power level about 16 dB below the noise.

User position accuracy is the measurement of a three-dimension position using GPS signal. The user position accuracy of these signals is based on the reference of the transmitted signal by the satellite, this reference is referred to as the signal-in-space (SIS). The performance standard for SPS states that for L1 C/A code, single-frequency position accuracy is ≤ 7.8 meters 95% of the time [15]. The performance standard for PPS states that that L2 P(Y), the dual-frequency (both f_{L1} and f_{L2} being used) position accuracy is ≤ 5.9 meters 95% of the time [16]. With the use of differential GPS (discussed further in section 2.5.1) accuracy can improve to ≤ 3 meters for SPS and ≤ 2 meters for PPS. User position accuracy

is heavily dependent on the quality of the receiver, atmospheric conditions, and blocking of the signal due to building, bridges, trees, and other objects.

Each GPS satellite has the same signal structure and transmit both f_{L1} and f_{L2}. A simplified diagram of this signal structure is show in figure 2.5 where the derivation of both transmitted signals is clearly demonstrated. The diagram must be read form left to right (same as direction of data flow) to accurately depict the signal construction. It is important to note that the f_0 is tuned to exactly 10.22999999543 MHz to adjust for relativistic effects but the observed frequency on by a GPS receiver is 10.23 MHz; therefore all depictions of the signal structure are from the receiver's perspective.

Current plans for GPS improvements are the addition of the L5 band, transmitted at 1176.45 MHz (or $115 \times f_0$). This band is being developed for civilian safety-of-life providing a secure and robust signal to life critical applications. Compared to L1 C/A the new L5 band will enhance performance of the signal, have a higher transmitted power, and other upgrades. Lastly, GPS transmits at 1381.05 MHz (or $135 \times f_0$) a signal used to detect nuclear detonations (NUDET) supported by the United States Nuclear Detonation Detection System (USNDS). This signal is referred to as the L3 band carrier signal and is part of the Detection System Payload.
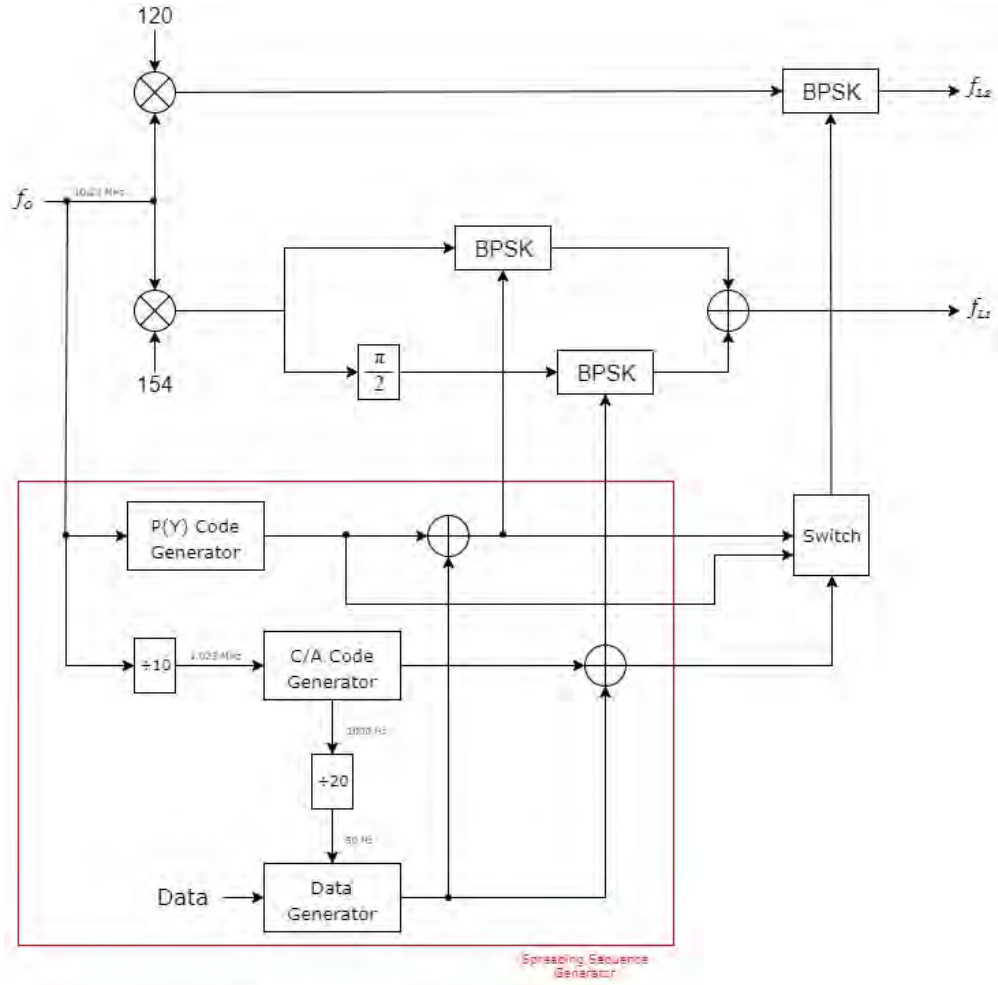
Figure 2.5: SPS and PPS signal structure

The signal structure diagram references equations 2.5 and 2.6 where the f_0 is multiplied by two constants, 120 for f_{L2} and 154 for f_{L1}. The bottom of diagram, encased by a red square, depicts the spreading sequence generators. Each of the two generator blocks, the C/A Code Generator and P(Y) Code Generator, produces a sequence of chips (bits that do not hold any information) which modulate the carriers. The C/A code generates a repeating sequence of 1023 chips at a rate of 1.023 MHz. The P(Y) code generates a sequence of about 2.35×10^6 chips with a chip rate of 10.23 MHz. Each generated spreading sequence is multiplied to the data sequence of bits and modulated on to the L1 and L2 carriers using Binary Phase Shift Keying (BPSK) modulation. The C/A code is modulated only

onto the L1 carrier frequency while the P(Y) code is modulated on both L1 and
L2 carrier frequencies. The GPS control segment selects the spreading sequence
modulated on to the L2 carrier. This is usually the P(Y) code but for unpublished
reasons the control segment still has the option to switch between them, hence
the switch block on the diagram.

2.4.1 Binary Phase Shift Keying

Using a sinusoidal signal as the carrier frequency and modulating it with a bipolar
stream of bits will reverse the polarity of the sinusoidal signal, this is illustrated
in figure 2.6.
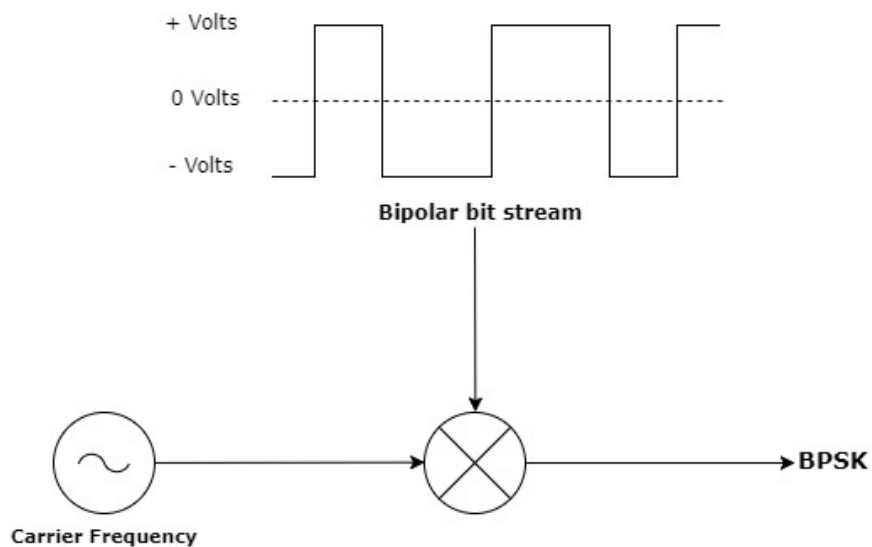
Figure 2.6: Generation of BPSK on a sinusoidal carrier frequency

BPSK uses two phases separated by 180° which means that the carrier is either
transmitted in its original form or with a phase shift. The change in phase occurs
at the time of the bit change. The top graph in figure 2.7 shows a sinusoidal carrier
signal at 2 Hz. This carrier signal is modulated by the square wave signal in the
middle graph. The modulated signal is the BPSK signal shown in the bottom
graph.
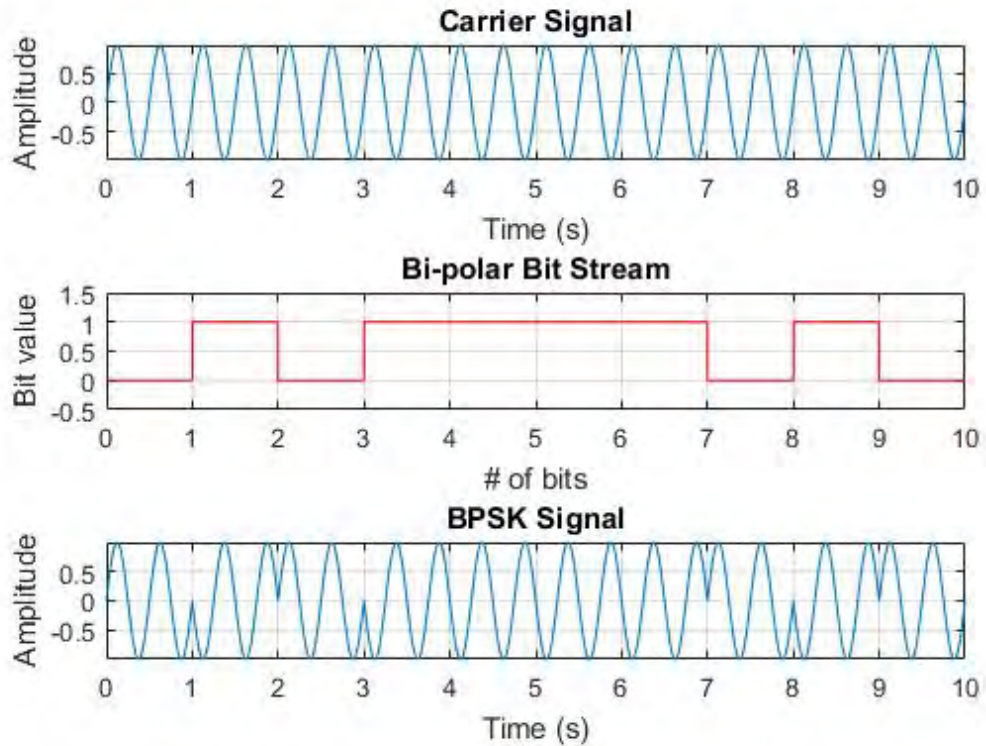
Figure 2.7: BPSK modulation

Observe the symmetrical shape of the BPSK signal in figure 2.7. This symmetry is caused by making the bit rate of the bipolar bit stream signal a sub-multiple of the carrier frequency. In GPS, this bipolar bit stream is the navigation data transmitted by the satellite. This navigation data is transmitted at a rate of 50 bits-per-second (bps) and modulated onto both the L1 and L2 carrier.

2.4.1.1 Direct Sequence Spread Spectrum

Direct Sequence Spread Spectrum (DSSS) extends BPSK by including a third signal called pseudo random noise (PRN) spreading code. PRN spreading code signal is similar to the data bit stream signal: it also contains a stream of -1 and 1 values, but at a significant higher chip rate. In GPS, the data bit stream is the navigational data and has a bit rate of 50 bps, while the PRN bit stream is the

17

C/A code at a chip rate of 1.023 MHz (20460 times more than the navigational data bit rate). The PRN sequence is periodic, finite, and is completely known to the receiver.

The modulation of the carrier is done by taking the output of an exclusive-or, defined as XOR, multiplication of the PRN sequence and the navigation data, this is shown in figure 2.8. Only when the value of one of the two signals is high will the output of the XOR multiplication be high, this input/output relationship, called a truth table, is shown in table 2.2.

Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.2: Truth table of the XOR
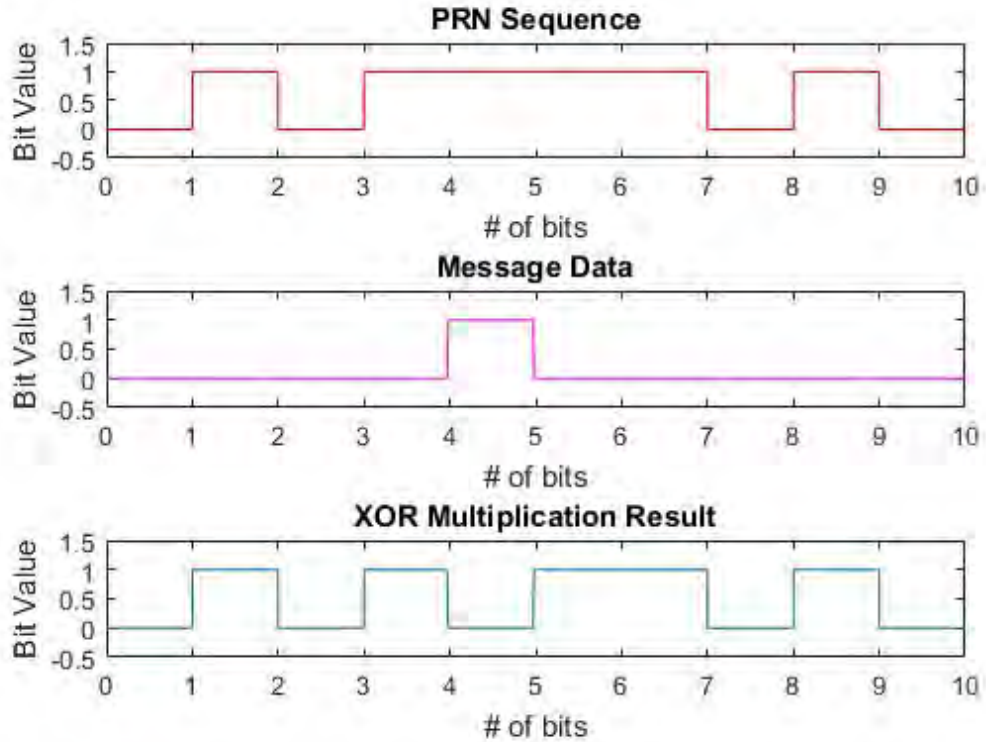
Figure 2.8: XOR multiplication of two signals

An illustration of how a DSSS modulation is applied can be seen in figure 2.9. The primary reason to use DSSS in GPS is to enable all of the satellites in the constellation to transmit on the same frequency. It is important to design proper PRN sequences so that cochannel interference can be reduced as small as possible [17]. The GPS C/A code uses the Gold code, a liner shift register sequence [18], with 1023 chip sequence.
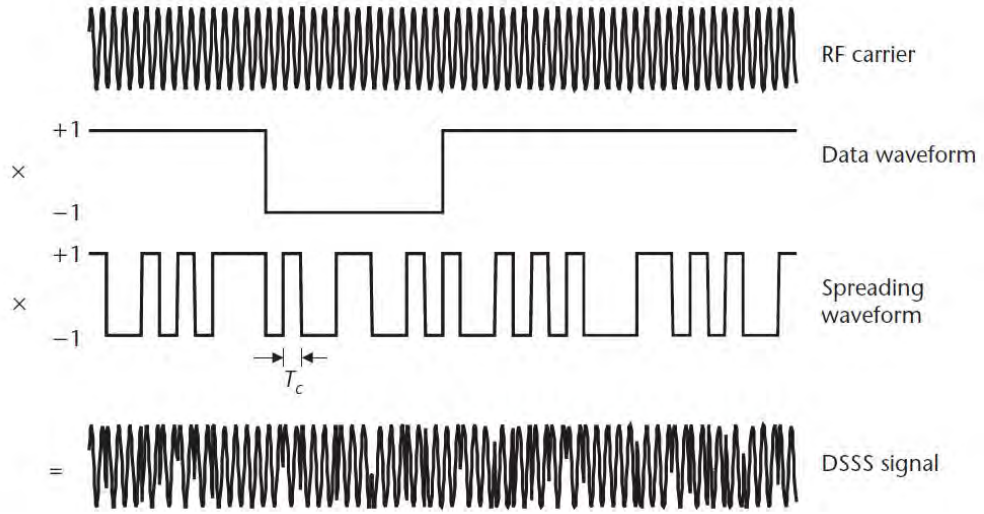
Figure 2.9: DSSS modulation [2]

2.4.2   C/A Code

The C/A code is the spreading sequence used in the L1 channel and consists of a binary sequence of 1 and 0 states. This binary sequence is called a PRN code, which indicates that the code may appear random while being generated by a deterministic process. It has a chip rate of 1.023 MHz generated using a 1023 length Gold code. Equation 2.8 shows that the C/A code sequence repeats every 1 ms. The PRN code allows the access to the carrier signal, by despreding a specific PRN sequence the user can then extract the data from the carrier signal.

$$C/A_{chiprate} = \frac{1023}{1.023 \times 10^6\,\text{Hz}} = 1\,\text{ms} \tag{2.8}$$

19

Each satellite in the GPS constellation is assigned a satellite vehicle (SV) number and a PRN number. These numbers are not the same, but are unique to the constellation: whereas the SV number identifies the satellite, the PRN number identifies the PRN signal. When a receiver is acquiring a satellite signal, a locally generated C/A code is cross-correlated with the acquired signal which in turn removes nearly all of the other signals from other satellites form the acquired signal. This only works when the locally generated C/A code has the correct code phase. Therefore, multiple SVs can transmit under the same frequency but with different PRN codes and a receiver can receive the signal from multiple SVs and be able to separate them.

The generation of a C/A code is published in [19] with the intent to be replicated in civilian GPS receivers. GPS only works when all segments are coherent, meaning that the user segment must be properly develop in accordance with the published specifications in order to work with the space segment. Figure 2.10 shows the design schema of the C/A code generator where two shift registers and a bit selector are color coded to represent the important sections of the generator.
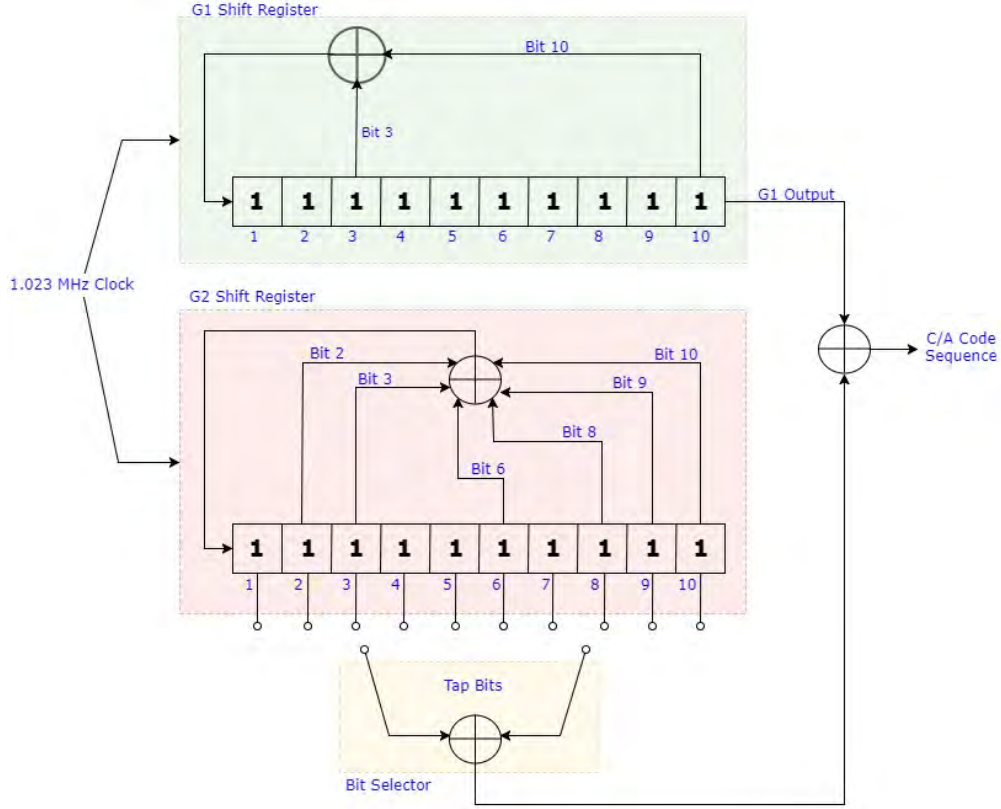
Figure 2.10: C/A code generator design schema. G_1 register in green, G_2 register in red, and the *Bit Selector* in yellow

The green section in the C/A code generator diagram shows the G_1 shift register. It is a 10-bit shift register with an input clock of 1.023 MHz. The 10 bits on the register are shifted right at every rising edge of the input clock. Bit 10, the least significant bit (LSB), is shifted out of the register and all other 9 bits are shifted right. The value of bits 2 through 10 is described as $G_1^*[n] = G_1[n-1]$ where n is the bit number. The left most bit depends on the XOR result of bits 3 and 10 and is described as $G_1^*[1] = G_1[3] \oplus G_1[10]$. The bit shifted out of the registered is used as one of the two input into the XOR that will result on the C/A code bit sequence. The red section in the diagram shows the G_2 shift register. Much like the G_1 register, G_2 holds 10-bits and also shifts each bit to the right at the rising edge of the clock input. Both shift registers are synchronized to the same input clock, this means that the shifting happens at the same time in both

registers. The value of the left most bit shifted into the register is described as $G_2^*[1] = G_1[2] \oplus G_1[3] \oplus G_1[6] \oplus G_1[8] \oplus G_1[9] \oplus G_1[10]$. The resulting value is 1 only when the compared bits have an odd number of bits equal to 1. The right most bit is shifted out of the register but not used. The feedback configuration of both registers can be represented using a polynomial equation, defined in equation 2.9 and 2.10, where the exponential value refer to the bit that is being fed back into the register. The output of register G_2, which is the bit used to be XOR compared with the output of the G_1 register, is a combination of two bits. The two bits are chosen by the *Bit Selector*, identified as the yellow section in the diagram. The *Bit Selector* selects two of the 10 bits of the G_2 register and the XOR comparison of those bits is then used to generate the C/A code sequence. Table 2.3 shows the bits that need to be selected by the *Bit Selector* to generate the desired C/A code sequence. Each bit pair is unique to a single SV except for PRN 34 and 37 which have the same bit pair and generating the same PRN sequence. PRN numbers 33 through 37 are never used for C/A code L1 signal, they have other uses.

$$f_{\text{G1}}(x) = 1 + x^3 + x^{10} \tag{2.9}$$

$$f_{\text{G2}}(x) = 1 + x^2 + x^3 + x^6 + x^8 + x^9 + x^{10} \tag{2.10}$$

At the start of the sequence both registers are initialized to all ones and neither of the shift registers will ever be a sequence of all zero bits.

PRN Number	Tap Bits	First 10 Chips (Octal)
1	$2 \oplus 6$	1440
2	$3 \oplus 7$	1620
3	$4 \oplus 8$	1710
4	$5 \oplus 9$	1744
5	$1 \oplus 9$	1133
6	$2 \oplus 6$	1455
7	$1 \oplus 8$	1131
8	$2 \oplus 9$	1454
9	$3 \oplus 10$	1626
10	$2 \oplus 3$	1504
11	$3 \oplus 4$	1642
12	$5 \oplus 6$	1750
13	$6 \oplus 7$	1764
14	$7 \oplus 8$	1772
15	$8 \oplus 9$	1775
16	$9 \oplus 10$	1776
17	$1 \oplus 4$	1156
18	$2 \oplus 5$	1467
19	$3 \oplus 6$	1633
20	$4 \oplus 7$	1615
21	$5 \oplus 8$	1746
22	$6 \oplus 9$	1763
23	$1 \oplus 3$	1063
24	$4 \oplus 6$	1706
25	$5 \oplus 7$	1743
26	$6 \oplus 8$	1761
27	$7 \oplus 9$	1770
28	$8 \oplus 10$	1774
29	$1 \oplus 6$	1127
30	$2 \oplus 7$	1453
31	$3 \oplus 8$	1625
32	$4 \oplus 9$	1712
33	$5 \oplus 10$	1745
34	$4 \oplus 10$	1713
35	$1 \oplus 7$	1134
36	$2 \oplus 8$	1456
37	$4 \oplus 10$	1713

Table 2.3: C/A code bit assignment and first 10-chip sequence

2.4.3 Message Signal Format

Navstar specifications define how the message data are modulated onto the carrier signal [19]. It contains ephemeris data, health data, almanac data, and other useful data used by the receivers for navigation purpose. The navigation data is a stream of binary values, zeros and ones, that, together with the C/A code, is modulated onto the carrier signal. The navigation data rate is at 50 bits per second, meaning that every 2 ms a bit is generated. When modulated using DSSS modulation, the message signal is the slower sequence of binary bits, while the C/A code is the much faster sequence (refer back to section 2.4.1.1). Together these create the transmitted signal called SPS, or L1C (short for L1 C/A).

The navigation data are divided into pages, subframes, and words. A page is a collection of 5 subframes (a subframe is made up of 10 words) and a word is 300 bits. There are a total of 5 subframes, each containing specific data about the satellite or the constellation. Figure 2.11 shows a typical subframe structure as well as the size of each word, size of the subframe, and the direction the data flows out.
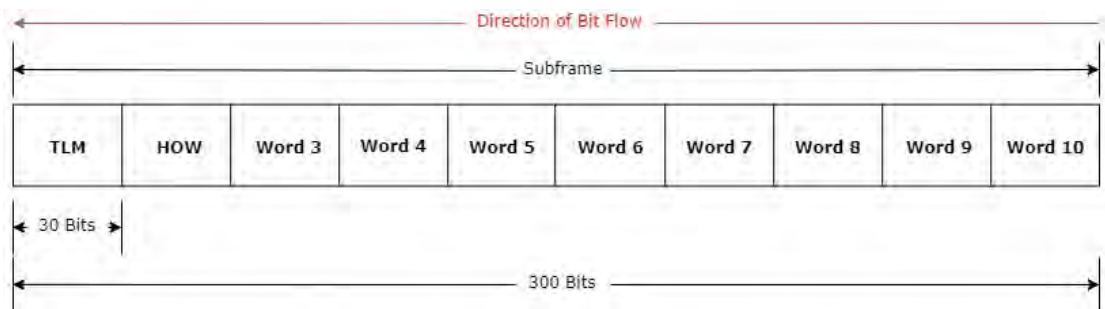
Figure 2.11: Subframe structure, size, and direction of data flow from SV

The data flows from left to right, meaning that the first bit transmitted is the left most bit of word 1. Word 1 is named TLM which is short for telemetry. The telemetry word is the first word of every subframe. It contains a preamble, an 8 bit known sequence used to identify the start of a new subframe. Figure 2.12 shows

the whole telemetry word starting with the 8-bit preamble, a telemetry message, one reserve bit, one status flag, and a 6 party bits. The telemetry message is a 14-bit-long message reserved to relay information about the PPS signal. The flag bit number 24 is an Integrity Status Flag (ISF). The last 6 bits are called the parity bits. Every word in every subframe reserves the last 6 bits for parity checking. These 6 bits are used by the receiver for error correction. The algorithm used for parity check is well-defined in [11] paragraph 20.3.5.2.
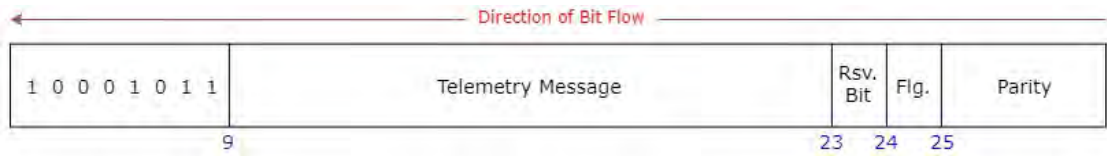
Figure 2.12: Telemetry word format

The HOW, short for handover word, is the second word of every subframe. The handover word starts with the time-of-the-week (TOW) count. The TOW (in the HOW word) is 17-bit truncated version of the 19-bit TOW count, and is an integer between 0 and 403,199, which equals to one full week. The full TOW count is the least significant bits of the Z-count, which is a time count that started in the night of January 5, 1980/ morning of January 6, 1980 [19]. The truncated TOW count maximum value is 100,799: once reached the count rolls over to zero and starts again. Figure 2.13 is the HOW word format including the TOW as the first 17 bits followed by two flags, bit 18 is the alert flag, and bit 19 is the anti-spoof flag. The 3-bit subframe ID represents the current subframe, with *000*, *110*, and *111* being invalid states. Bits 21 and 22 are used to solve for parity. In the HOW word, the last two parity bits, bits 29 and 30, must always be zeros. This is done by calculating the *'Solve Parity'* bits such that, when calculating the parity, bits 29 and 30 are zero. Like the TOW word, the HOW data also flows right to left with the first bit of the TOW count being the first transmitted bit. The HOW word is always the second word in every subfame and always follows the TOW.

25
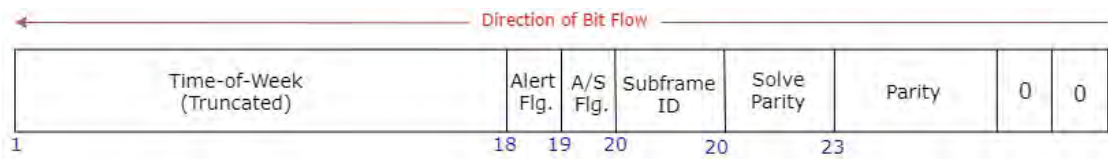
Figure 2.13: Handover word format

Words 3 through 10 vary depending on the subframe and page number. A short description of the eight words of data on each subframe is listed below. Refer to [19] for a full description of all the bits.

Subframe 1: Contains the GPS week number (relative to the epoch of midnight of January 6 1980), SV accuracy and health data (uses to determine if the SV is trustworthy), and clock correction data (to assist in computing the time the navigation message was transmitted from the SV).

Subframe 2 and 3: Contains the transmitting SV's ephemeris data which includes satellite orbits parameters, and correction terms used to calculate the transmitting SV's position.

Subframe 4: This subframe is subcommuted 25 times, once in each page. Each page of subframe 4 contains the almanac and health data for SVs 25 through 32. Subframe 4 can have 4 different formats depending on the page number. Those different formats contain special messages, satellite configuration flags, Coordinated Universal Time (UTC), and ionospheric data, in addition to almanac data.

Subframe 5: This subframe is also subcommuted into 25 different pages. Unlike the previous subframe, this frame only uses two formats. These two formats con-

tain almanac data for SVs 1 through 24, almanac reference time, week number, and health data.

Figure 2.14 illustrates a time graph showing the transmission timing of the navigation data. At the start of the transmission, **time** = 0s, the first bit of the first word in subframe 1 is transmitted. The message data is transmitted at 50 bps, therefore the transmission of 30 bits takes 0.6 seconds, 300 bits takes 6 seconds, 5 subframes takes 30 seconds, and 25 pages takes 750 seconds. It takes 12.5 minutes (750 seconds) to receive a full GPS navigation message. Receivers with augmented GPS connect to a database to acquire almanac data, so do not require receiving the full navigation message before their position can be calculated.
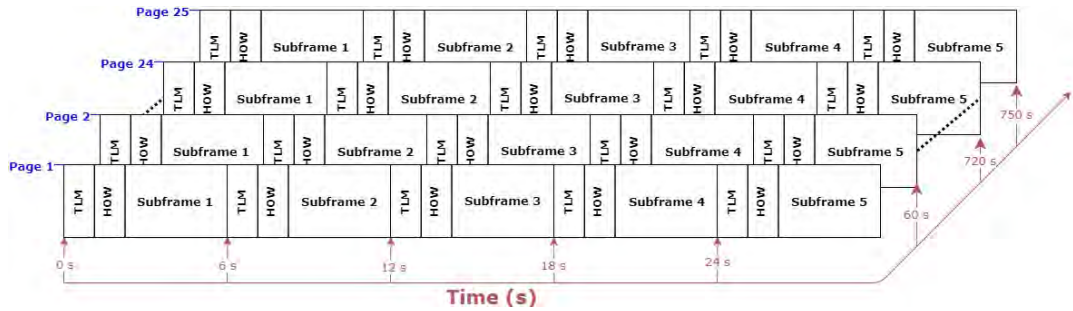
Figure 2.14: Timing graph of the GPS message data

2.5 Augmentation Systems

GPS augmentation system helps improve the accuracy, integrity, or availability of GPS. Augmentation systems can be space-based, like geostationary satellite, or ground-based, like a network to help accuracy in cellular telephones. A different type of GPS augmentation can include sensors and compasses that provide navigation in areas where GPS is not available. Some vehicles integrate the use of inertial measuring units (IMU), which measure force, angular rate, and magnetic field, and blend results with GPS to enable navigation in places like tunnels, inside

27

building. There are many augmentation systems worldwide, a few of which will be further discussed in this section.

2.5.1 Global Differential GPS

Sometimes called Nationwide Differential GPS (NDGPS), Differential GPS (DGPS) is a system that increases the accuracy and integrity of the GPS signal. These systems are ground-based and placed in areas where an increase in GPS accuracy is needed. The current DGPS Service is operated by the USCG and consists of one control center and forty-six broadcast sites [11]. These broadcast sites include weatherproof reference station antennas that broadcasting correction signals to improve position accuracy to better than 10 m to users who required better accuracy. The USCG transmits these correction signals in the long wave radio, frequencies between 285 kHz to 325 kHz, and typically near waterways and harbors. In 1993 the DOT calculated an estimated error growth in this system of 0.67 m per 100 km from the broadcasting site, further measurements in the Atlantic and Portugal suggested an error growth of just 0.22 m per 100 km [20]. The standard GPS accuracy of about 15 meters can be augmented to 10 m or better with DGPS [21].

2.5.2 Wide Area Augmentation System

On August of 1994 the Federal Aviation Administration (FAA) and the DOT announced that a new GPS augmentation service for civil aviation was going to be deployed to improve navigation. The goal of this new service was to increase accuracy, integrity, and availability of GPS and enable aircraft to use it throughout all phases of flight. The Wide Area Augmentation System (WAAS) was commissioned in 2003 and is an extremely accurate system with requirements enabling a horizontal accuracy of 16 meters or less at least 95% and vertical accuracy of 4

meters or less at least 95% of the time. The FAA WAAS Test Team measured actual accuracy performed at horizontal accuracy of 0.7 m and vertical accuracy of 1.2 m [22]. Figure 2.15 shows WAAS coverage of localized performance with vertical guidance (LPV), which are the highest precision GPS with WASS enabled instrument approach procedures into airports, in the whole Continental U.S. and Canada. Compared to the GPS horizontal accuracy requirements of 36 meters and vertical accuracy requirements of 77 meters it is clear why civil aviation requires the use of WAAS for flight navigation, and landing sequences.
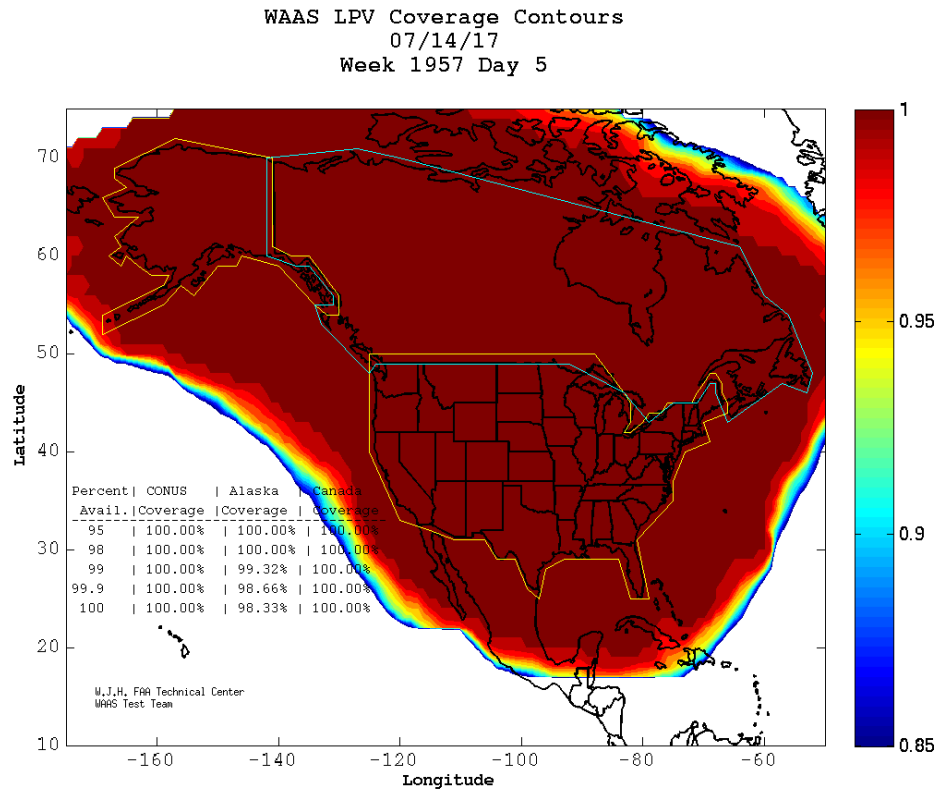
Figure 2.15: WASS LPV Coverage [23]

The current WAAS works by using 38 WAAS Reference Stations (WRS) spread around the US, Canada and Central America. These stations monitor GPS signals to determine position errors. The calculated position errors are sent to three WAAS Master Stations (WMS) via terrestrial communications. Augmentation

messages are generated at each WMS, which contain information that removes errors from the GPS signal. WAAS sends the augmentation messages to three geostationary satellites which then relay the GPS-like message to GPS/WAAS receivers on Earth.

The implementation of WAAS in civil aviation has enabled landing approaches with visibility as low as 200 feet, improving airport access to places where instrument landing system (ILS) are not available. It has provided the direct navigation between two airports without the need of ground base navigation instruments like VHF omnidirectional range (VOR) signals or distance measuring equipment (DME).

Chapter 3

Methodology

3.1   GPS Signal Reproduction

The reproduction of the C/A L1 signal consisted of generating the PRN and Message data on firmware and software, respectively. The process used to reconstruct the signal is outlined in this section.

The GPS signal reproduction was done by a combination of firmware and software. The firmware programmed the hardware and generated the signal to be transmitted. The software constructed the data to be transmitted over the signal and controlled the firmware during execution. This relationship between software and hardware is described in the diagram on figure 3.1. The firmware was developed using Simulink, a block based programming tool, and the software was written in MATLAB, a popular programming tool used in signal processing.
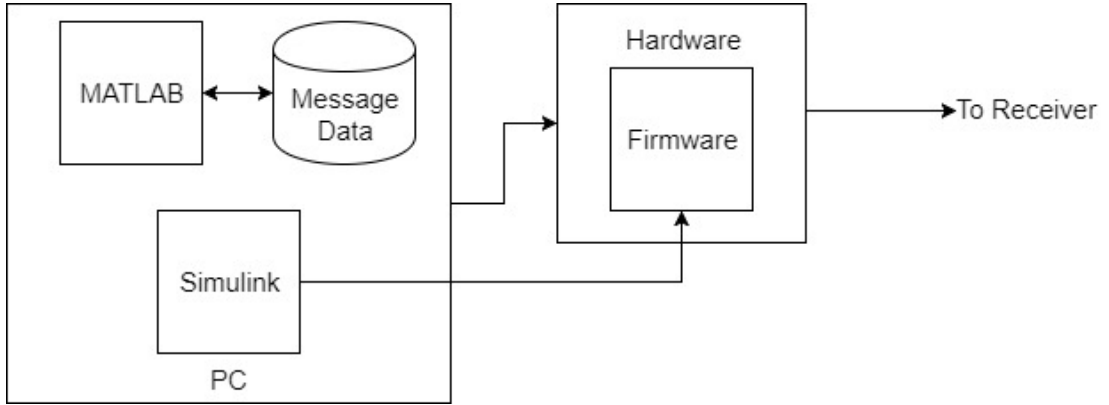
Figure 3.1: Diagram showing connection between the software and the firmware

The reproduced signal was the L1 C/A GPS signal transmitted for civilian use all over the world. It consists of the PRN signal and message signal, which are both modulated onto the carrier frequency L1. This project simulated transmission of the GPS signal from four concurrent SVs, therefore the same signal structure was replicated four times with a few data points being changed to distinguish each SV.

The reproduced signal is a BPSK modulated signal with a spreading sequence. The signal is transmitted at 50.127 MHz and is modulated by a message data signal at 50 Hz and a spreading signal at 1.023 MHz. Each SV has a unique message signal and spreading sequence. The resulting transmitted signal is the sum of four independently modulated BPSK signals.

3.1.1 ROACH FPGA

The Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) developed a serveral processing boards for radio astronomy. One of these boards is the Reconfigurable Open Architecture Computing Hardware (ROACH), a Field Programmable Gate Array (FPGA) based processing board that uses the CASPER and Xillinx, a FPGA manufacture, Simulink software blocks to provide signal analysis. The Radar and Microwaves Laboratory com-

missioned a set of ROACH processing boards to be used in various signal processing projects within the laboratory. The author of [24] provides a guide that details the setup of the ROACH and the software tools required for development of firmware. His application of the hardware and software environment was meant for real-time symbol recovery of digital television (DTV) signals for passive radar applications. Particular knowledge about the use of the software environment, the digital to analog converter (DACs), and access to the block random access memory (BRAM) was translated into this thesis project.

The ROACH runs on a Xilinx Virtex5 FPGA and an embedded AMCC PowerPC 440EPx processor [25]. The DAC used in this work is a Texas Instrument DAC5681 with 16 bit resolution and sample rate of 1 GS/s. The Simulink blocks used to develop the firmware for the thesis project was a combination between Xilinx and CASPER blocks. The Xilinx blocks performed logic targeting the FPGA and the CASPER blocks allowed control over the DAC and the BRAM.

3.1.2   Firmware

The firmware is software written to internal memory and used to program the ROACH in order to generate the desired signal. The Simulink software tool was used to design the firmware. Blocks used within the Simulink model are blocks from the Xilinx and CASPER toolbox. These blocks, when connected together in a block flow, produce logic that is compiled to FPGA hardware descriptive language (HDL) and uploaded onto the ROACH's internal read-only memory.

The firmware design requirement was to produce a sum of four GPS signals containing the PRN spreading sequence and the message data. The resulting firmware, shown in figure 3.2, meets the design requirements. In addition, it allows control over the number of signals being transmitted, configuration of the

PRN sequence to be generated, independent message data storage for each signal, clock synchronization, and other testing modes allowing the live testing of the generated signal.

For simplicity, the firmware will be described in parts. Descriptions of each individual block follow the data flow from left to right. Because each SV signal has an exact signal architecture, the firmware will be initially described as a single signal model. The block logic that connects the four SV signals together and the composite four signal transmission will be described as the multisignal model. This breaks traditional Simulink terminology where each SV signal model would be referred to as subsystems. The declaration of subsystems has been reserved to each of the four signal components, PRN Clock, Message Clock, and Message Signal, and PRN Generator.

An important distinction to make is the difference between the system clock, and the model clock. The system clock is running at 200.508 MHz and is derived from the FPGA input clock at four times this rate. Therefore each clock tick, or clock count, happens at the rate of the system clock, 200.508 MHz. A bit transition, signal delay, or other logic happens at the rising edge of the input clock. Keeping each clock domain separate is of paramount importance in order to understand the firmware model and data flow. In addition, it is important to emphasize that all of the logic in the firmware is referenced by bit-by-bit logic that transitions at the rate of the input clock. Simulink blocks manipulate data at bit level precision, so conventional programming thinking does not apply.
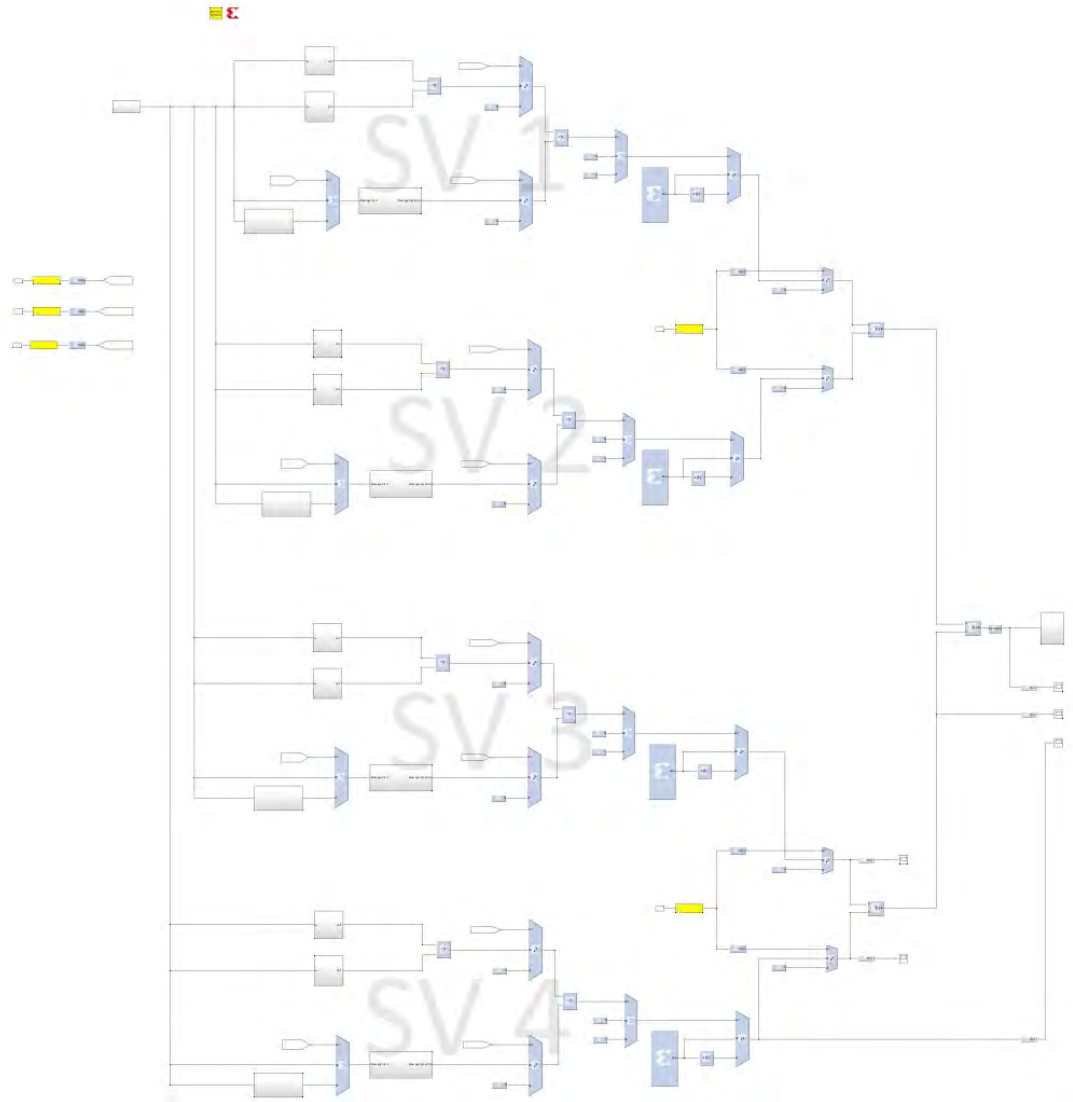
Figure 3.2: Complete firmware flow diagram with markings identifying the signals of the four SVs

3.1.2.1 Single Signal Model

The single signal model, shown in figure 3.3, generates a complete GPS signal from one SV. A complete GPS signal consists of the PRN signal, message signal, and the carrier frequency. Each of these subsystems are highlighted in the single signal model figure. The design called for the simulated GPS signal to be transmitted over a cable at the transmission frequency of 50.127 MHz. Future plans included an up-conversion of the simulated signal to the GPS L1 frequency, 1575.42 MHz.

Critical relationship among the ADC rate f_{adc}, which is 4 times the system clock f_{sys}. The FPGA operates best for $f_{\text{sys}} \approx 200\,\text{MHz}$. The 50.127 MHz clock is $\frac{f_{\text{sys}}}{4}$ (selected for ease of generating the subcarrier) and should have an integer number of code chips per cycle ($\frac{50.127\,\text{MHz}}{1.023\,\text{MHz}} = 49$).

Figure 3.3: Block flow diagram of the single signal model highlighting the four subsystems

The PRN Clock, at 1.023 MHz, is the internal model clock. All logic is synchronized to this clock by setting the clock pulse of the logic block to the rising edge of the model clock. To generate a clock at this frequency, a clock divider logic was constructed. This logic used the system clock as an input to a counter. The counter counts to a number determined by equation 3.1. A high output is produced if and only if the counter has reach its maximum count, any other time the logic yields a low output.

$$Count_{int} = \frac{System_{clk}}{Desired_{clk}} \tag{3.1}$$

The PRN Clock subsystem used the clock divider logic to generate 1.023 MHz, the desired clock frequency, from 200.508 MHz, the system clock. Using equation 3.1, the counter maximum count is $\frac{200.508MHz}{1.023MHz} = 196$. The counter defines the maximum count as a 8-bit number because the least amount of digits needed to represent 196 is 8 digits. Using the binary equivalent of 196, '11000100', bits 8, 7 and 3 can be used to check if the maximum count has been reached. In other words the counter starts the count at '00000001' then progresses up to '1100100', the only time during the count that bits 8, 7 and 3 are all one is when it reaches the full count. The choice of starting the count at '00000001' is to reduce the number of bits that needs to be checked for maximum counter count. By connecting those three bits to a logic AND block the output of the logic will only be high when the three bits are high. This logic, used to generate the PRN Clock, is shown in figure 3.4. The three middle blocks labeled Bit 8, Bit 7, and Bit 3 are splice blocks used to splice specific bits from the incoming bit stream. These splice blocks are used throughout the firmware flow diagram.
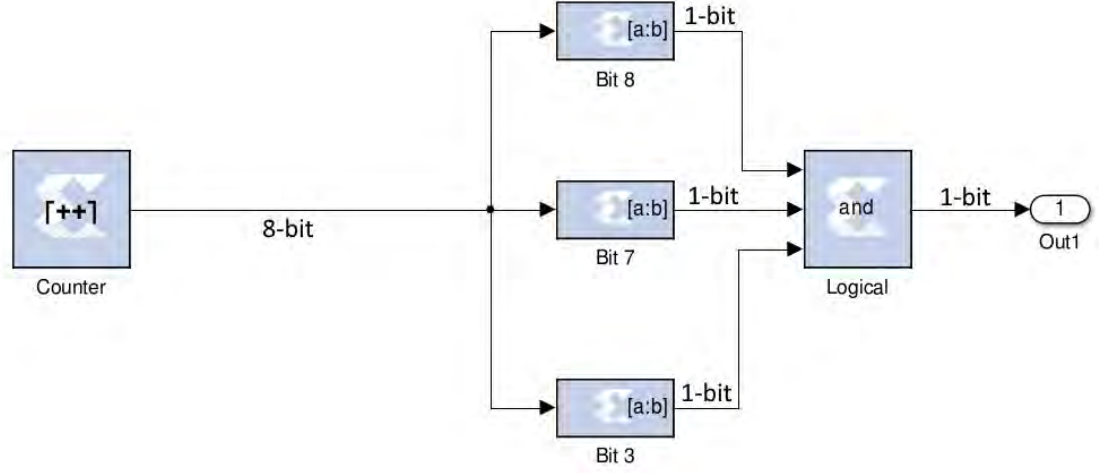
Figure 3.4: PRN Clock subsystem showing clock divider logic and bit stream size

The Message clock subsystem is of similar logic as the PRN sequence. This clock first uses two clock divider to take the input 1.023 MHz clock to 1000 Hz then to 50 Hz. Shown in figure 3.5 is the first clock divider logic. The multiplexer (MUX) block starts the logic flow. Its selector bit depends on the PRN clock output. The MUX will only output a high value when the PRN clock output is a high value otherwise outputs a low value. This logic synchronizes the counter to the rising edge of each PRN clock tick. The counter following the MUX is the first clock divider counter, *Message_Counter (1023)*. This counter's maximum count is to 1023. For simplicity the count starts at 2 and counts up to 1024, which is an 11-bit number. Splicing just the 11th bit serves as the clock pulse, as bit 11 will only be high every 1023 counts, which provides the 1000 Hz clock input to the second clock divider logic.
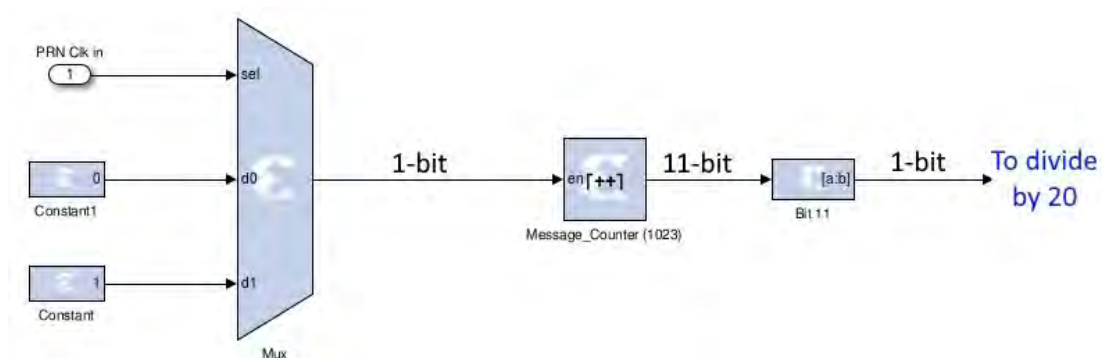
Figure 3.5: First sequence of clock divider reducing a 1.023 MHz clock to 1000 Hz

The second clock divider adds new logic used to check the rising edge of the input clock. Figure 3.6 shows the divide by 20 clock divider. The red rectangle surrounds the rising edge detector logic. The *Delay* block checks the input bit value one clock tick prior to the current clock tick, hence the z^{-1} label on the block face, then negates the value using the *Inverter* block. The logic AND result between the current value of the input clock and the negated previous value can only be high when the current clock input value is high and the previous value was low. The same logic follows on from there by dividing the 1000 Hz clock by 20, a counter with a maximum count of 20. The logical AND result between counter output bits 5 and 3 can only be high when the counter reaches maximum count, in turn resulting in a 50 Hz clock.
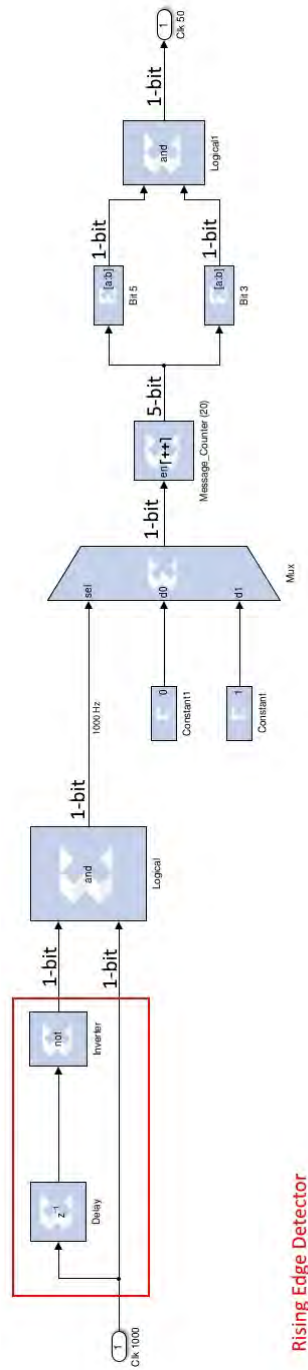
Figure 3.6: Second sequence of clock divider reducing a 1000 Hz clock to 50 Hz

The PRN Generator has two sets of logic, labeled as *G1* and *G2* registers, that mimic the PRN schema shown in figure 2.10. The *G1* register logic, shown in figure 3.7, is a 10-bit shift register logic that shifts the 9 MSB to the right and

calculates a new bit that is shifted into the MSB position. As defined in the GPS documentation, the new bit value is defined as $G_1^*[1] = G_1[3] \oplus G_1[10]$. Note, the MSB of the register is referred to as bit 1 in this thesis document as well as in GPS documents. The shifting of the register happens at the rate of the PRN clock. The output of the PRN clock controls the MUX selector, a high clock pulse selects the high MUX output and triggers a shift. The output of the register is always the 10th bit and at a rate of 1.023 MHz.
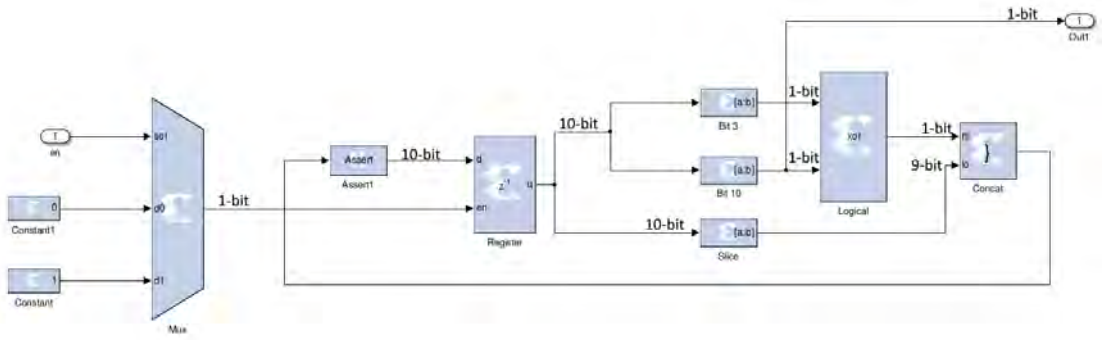
Figure 3.7: G1 10-bit shift register logic

The register *G2*, shown in figure 3.8, follows the same shift logic as *G1*; however, the calculation of the output depends on what SV is currently selected. Like the *G1*, register *G2* shifts to the right yet the new bit value is defined as $G_2^*[1] = G_2[2] \oplus G_1[3] \oplus G_1[6] \oplus G_1[8] \oplus G_1[9] \oplus G_1[10]$. This new calculated value is shifted into the register. The output logic, shown in figure 3.9, is determined by 2 bits, one selected by the upper MUX and the other from the bottom MUX. The selection of these bits is done through software: a command writes an integer value to a writable register in the firmware.
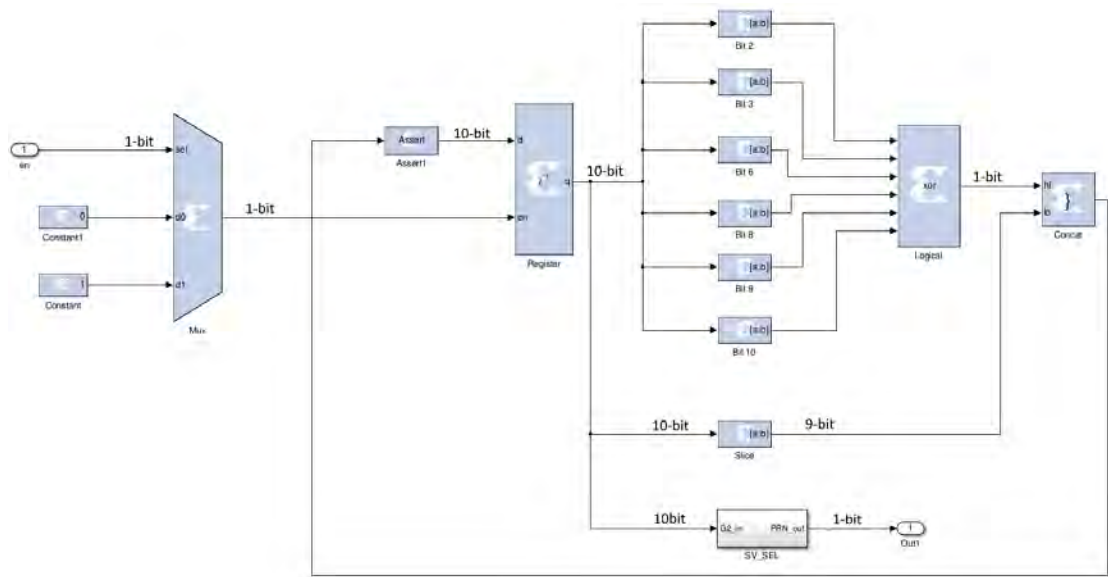
Figure 3.8: G2 10-bit shift register logic
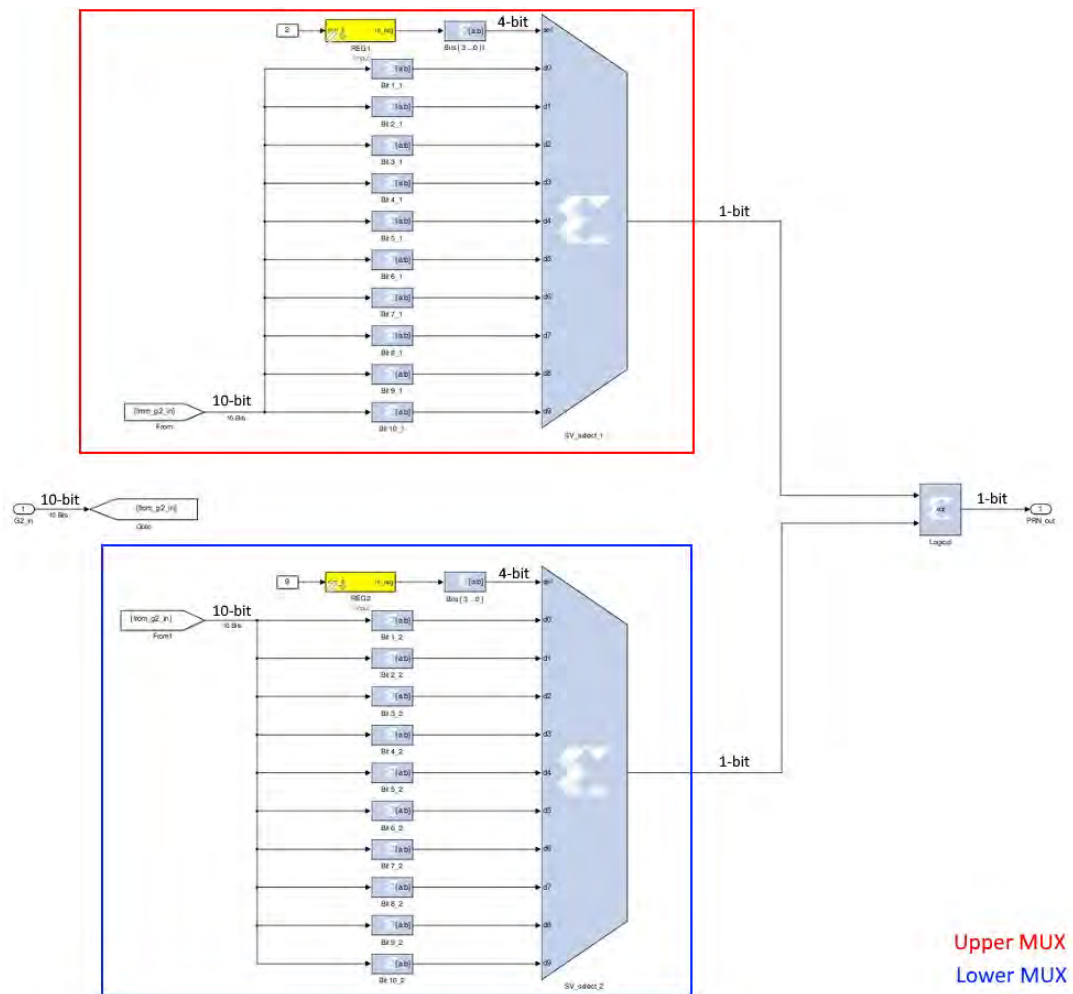
Figure 3.9: G2 register bit selector output logic

A close up of the upper MUX, figure 3.10, shows the 10 bit MUX input, controlled by a writable register labeled *REG1* and colored yellow. These yellow blocks are from the CASPER toolbox and are used to read data from, or write data to, the firmware. They are 32-bit registers by default so a splice block must be used to define which register bits are being used. The 10-bit MUX input comes from the *G2* register. Their selection depends on the PRN signal being generated. For example, if PRN 9 is selected then bit 2 of the upper MUX would be selected and bit 10 of the lower MUX would be selected. The XOR result of those two bits will yield the *G2* output.
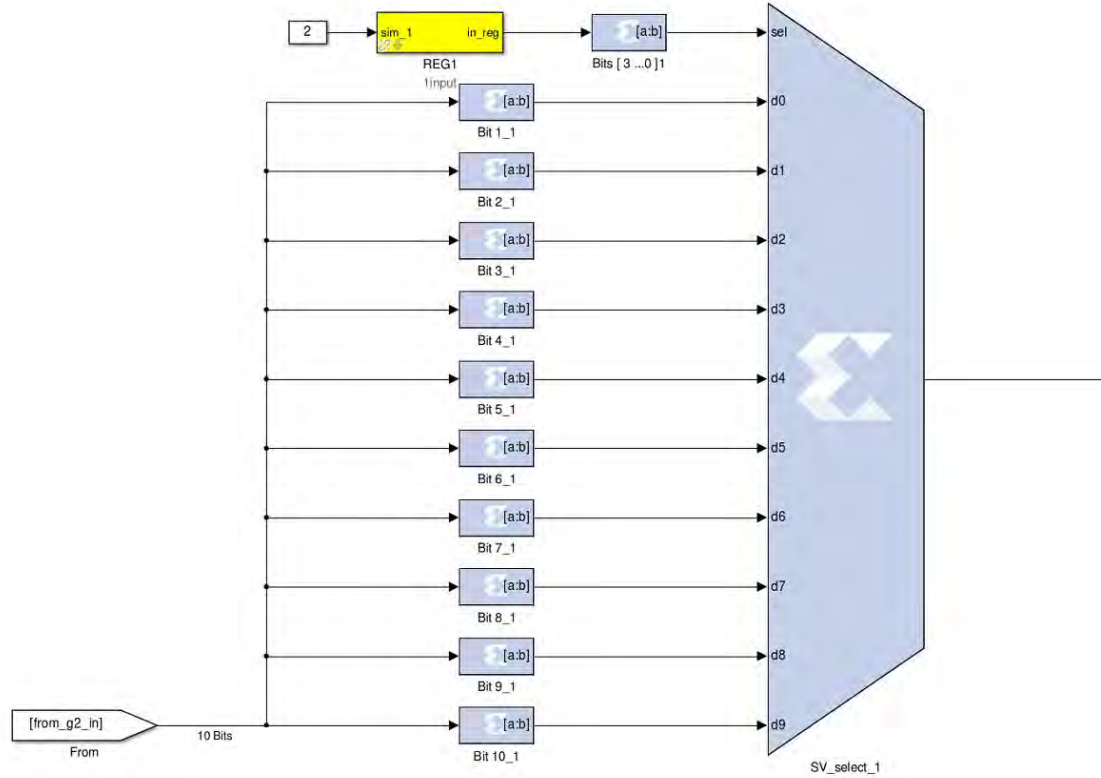
Figure 3.10: Close-up of the G2 register bit selector upper MUX

The Message Data subsystem generates the second part of the signal. The data modulated onto the carrier at 50 Hz is first created in software. Through write commands, the data are written to a BRAM block. The stored data are transmitted bit-by-bit at the rate of the Message Clock. Figure 3.11 shows the com-

plete logic flow of the Message Data subsystem. Most of the logic used here is also used throughout the rest of the firmware. Data flow starts with the address counter controlled by a 2-input MUX. The output of the Message Clock commands the MUX output to high at every rising clock edge. The counter, labeled *Bit_output_counter* and surrounded by a blue box, following the MUX counts is the bit counter. It counts starting at 0 and ends at 29, ignoring the last two bits, and each count selects the corresponding MUX input bit. Note, input MUX bits 31 and 32 are ignored. A second counter labeled *address*, and surrounded by a green box, counts through the BRAM address. The BRAM block is the yellow block in the center of the diagram flow and labeled as *bram1*. By default the BRAM has a address length of 1023 and a bit width of 32-bits meaning it can store up to 32,736 bits. The counter *address* counts up every time the *Bit_output_counter* reaches the maximum count of 32. An edge detector is triggered once the maximum count is reached, enabling the *address* counter to move up to the next address. This whole logic repeats indefinitely. In summary, this logic iterates to all bits of all of the BRAM addresses in a sequential order. Once the address counter reaches the value of 1023 it is reset back to zero and transmission of the first 30 bits begin once again.
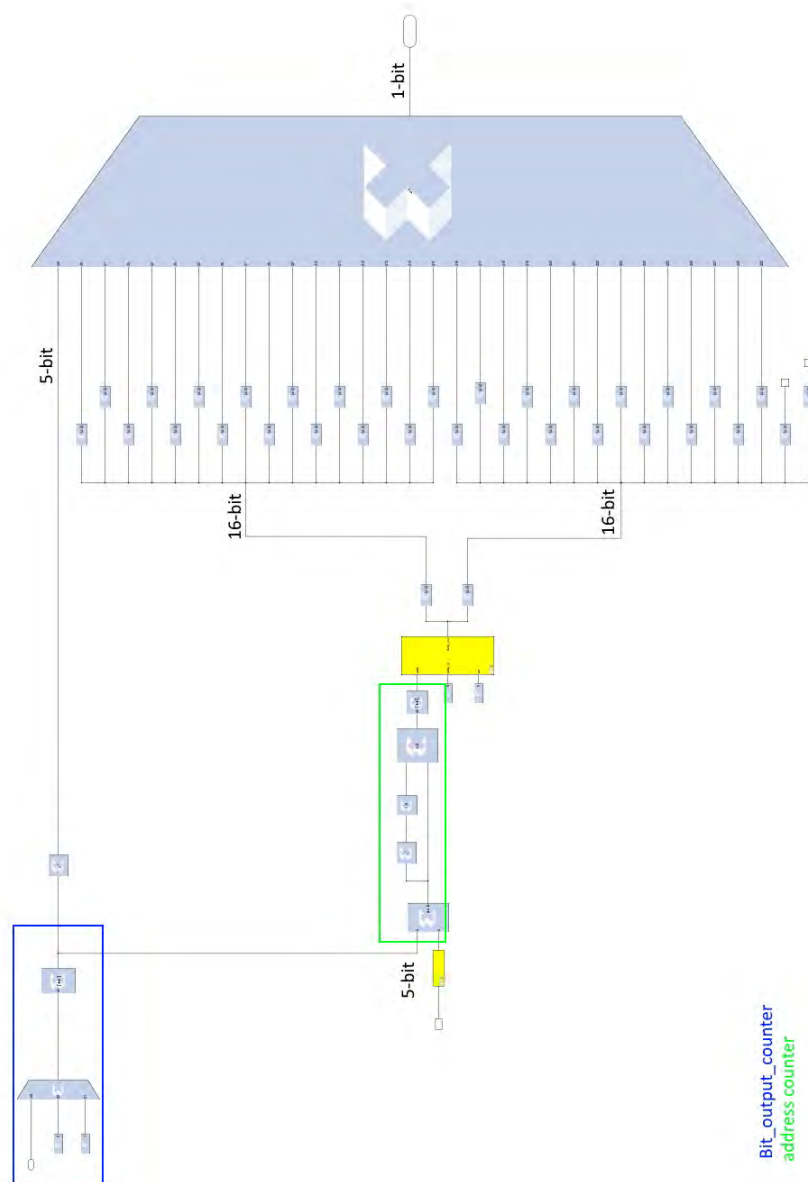
Figure 3.11: Message Data subsystem block logic

A cascade of MUX blocks create the last bit of logic before data are sent to the DAC to be transmitted. The output of the PRN Generator and the Message data subsystems are compared and the results of this comparison modulates the carrier frequency. The logic sequence is shown in figure 3.12 where the two MUX blocks received the output of the PRN Generator and Message Data subsystem and calculate the XOR result of both of those inputs. This is the first time in logic flow that two clock domains meet. The PRN Generator output changes at a rate of

45

1.023 MHz while the Message Data output changes at 50 Hz. The PRN bit input into the XOR changes at a much faster rate. This change translates to a typical DSSS modulation where a much faster bit change spreads the frequency while a much slower data bit modulates the frequency phase. In the spreading logic shown below the XOR block output controls a MUX block yielding a constant zero or one value that is fed into the selector bit of the carrier frequency MUX block. The *DSS Compiler 4.0* block generates a sinusoidal digital wave form at 50.127 MHz. This sinusoidal wave and its negated inverse can be selected via a MUX block. This selection depends on the XOR output result between the PRN sequence and the Message Data. The XOR comparator block acts as a multiplier. The resulting wave form is passed on to the DAC where it is staged for transmission. At this point in the logic flow a complete single SV GPS signal has been reconstructed.
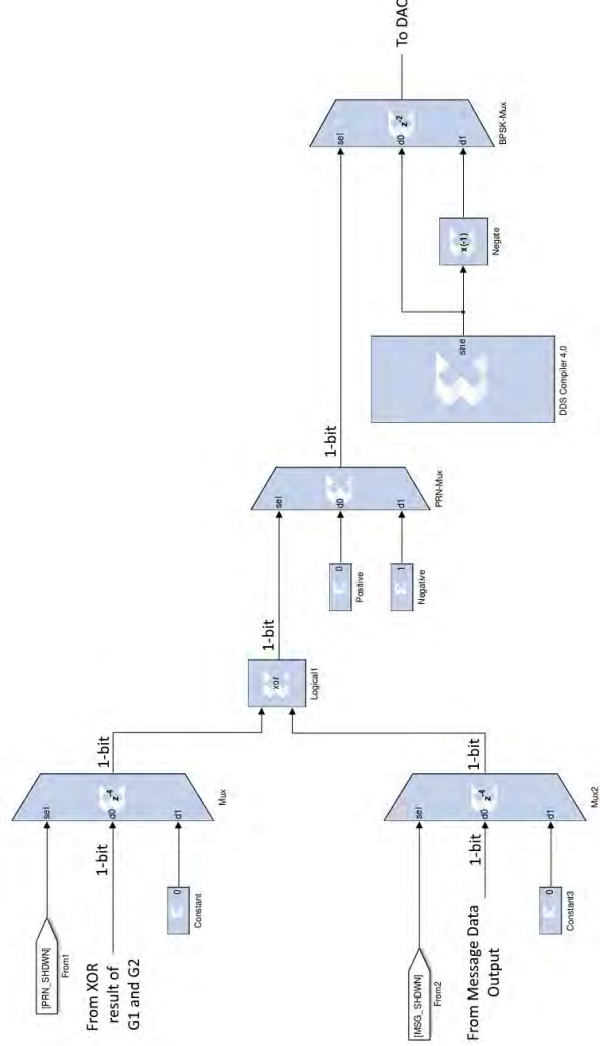
Figure 3.12: PRN and Message bit modulation with the carrier signal

The reconstructed signal, the output of the *DSS Compiler 4.0* block modulated by the PRN Sequence Generator and Message Data subsystems, is connected to the DAC as recommended in [24]. Other blocks in the model that were not mentioned were the configuration blocks. The CASPER toolbox requires the placement of the *XSG_core_config* block to configure the Xilinx *System Generator* block automatically, both blocks are shown in figure 3.13b, which defines a hardware platform, clock source, clock rate, and sample period. It must be present in the top level of the firmware logic flow. For additional of control registers were added in different

points to act as switches and allow live control of the signal structure. Some of the subsystems describe show some of these switches. Three registers, shown in figure 3.13a, control the message clock rate, the PRN sequence, and the message data. At any point while the firmware is running these registers can be written to via software commands. The message clock switch controls the rate of the message clock. This is implemented as a testing tool, allowing the increase of the message clock rate to be the same as the PRN clock rate. The PRN shutdown switch and the message shutdown switch were also implemented as testing tool allowing the on and off control of each individual signal. This made possible the individual testing of each subsystem.
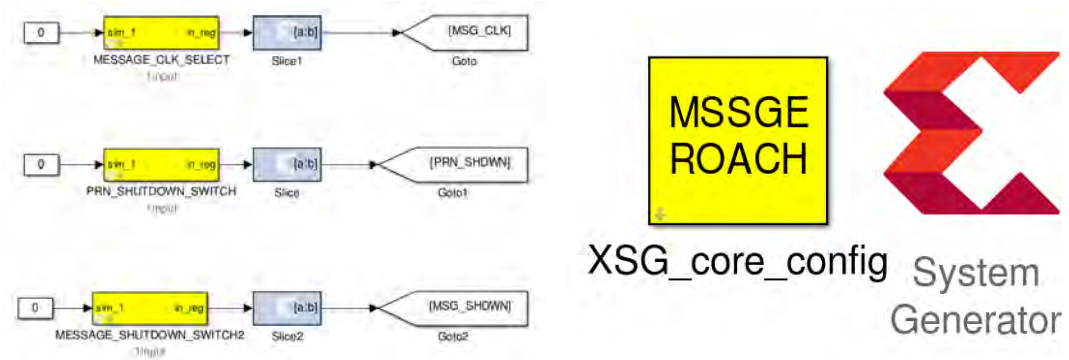
(a) Three CASPER register blocks used to control the transmitted signal structure

(b) CASPER and Xilinx configuration block

Figure 3.13: Additional blocks used in the single signal model

3.1.2.2 Multisignal Model

The multisignal model takes the single signal model and repeats it four times. Each repetition represents one of four SV signals. The multisignal model firmware generates four independent civilian GPS signals. The transmitted signal contains multiple copies of a single SV signal where each SV is independently configured. These copies are added together to generate the multi-SV signal. The full firmware flow diagram in figure 3.2 shows each SV copy connected by a addition logic. The signals are added together in steps. Signals SV 1 and SV 2 are together the same

48

time as signals SV 3 and SV 4. The two addition results are added once again to produce the multi-SV signal. A close up of the addition logic is shown in figure 3.14. The addition of signals is based on the amplitude of the signal at each clock tick. Signals can negate each other resulting on a zero amplitude or sum together to double the amplitude. This was important to consider in defining output levels.
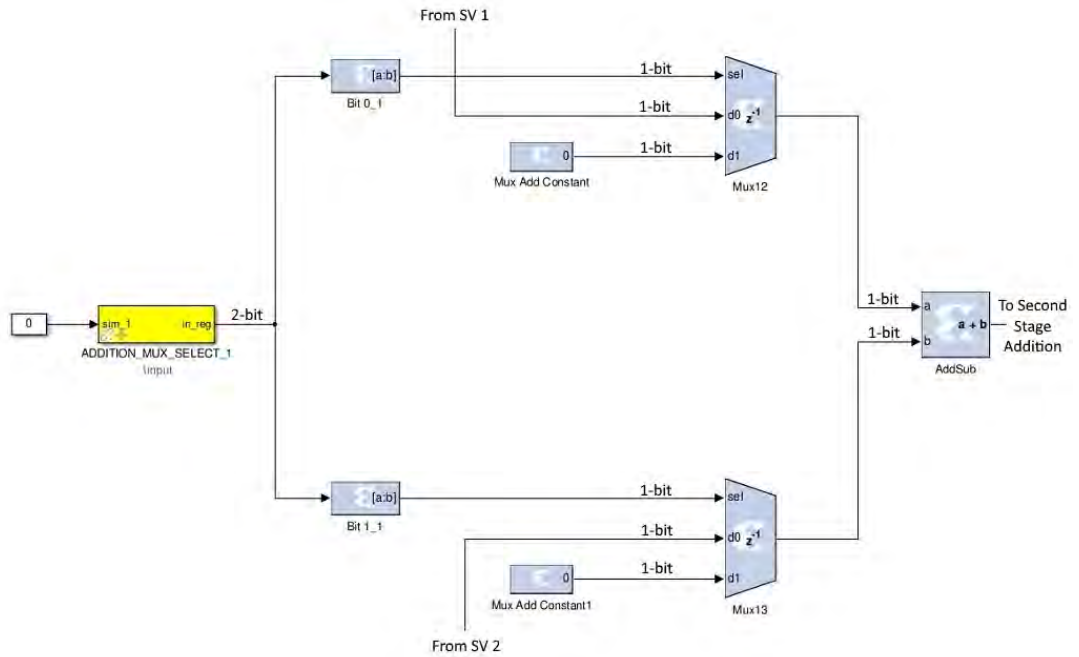
Figure 3.14: Close-up view of the first stage addition logic

For further control of the multi-SV, signal two register switches were added in the first addition stage. These two register switches allowed individual control over which signals are transmitted. Any of the four SV signals can be switched off during live execution of the firmware. This control was added as a testing tool.

3.1.2.3 Lessons Learned

It is easy to get confused with the different clock domains. Keeping track of which clock controls which logic is of paramount importance. Use of incorrect clocks

49

can produce undetectable errors. Because the different clock domains had vastly different frequencies, unwanted behavior was a common occurrence. Testing each logic set for proper clock rates was the most important test.

Bit delays had to be implemented throughout the block flow due to timing restrictions. During the compiling process of the firmware the compiler runs a timing check. This timing check analyzes the timing restriction of each block and determines, based on the complexity of the logic, if proper timing can be met. When timing is not met, a delay must be added on the bit flow into the block presenting timing restrictions. This process has been one of trial and error. The two ways to add a signal delay is either by the inclusion of *Delay* blocks or, in some block configuration parameters, setting an internal delay on the block logic. An example of this can be seen in figure 3.12 where both MUX blocks have a 4 tick delay represented by z^{-4} on the block face. Remember that each added delay is a system clock tick, meaning that a 4 tick delay at 200.508 MHz is much faster than any of the internal model clocks (PRN of Message clock).

3.2   Software Design

The software, written entirely in Matlab, provides control over the data stored in the firmware and control over different configurations parameters in the firmware. It stages the ROACH providing it with the necessary data to generate and transmit the desired signal.

The software flow diagram, shown in figure 3.15, starts with the attempt to connect the the ROACH using the KATCP tool. KATCP, the Karoo Array Telescope Control Protocol, is a tool created by CASPER to control ROACH devices over Transmission Control Protocol (TCP) or RS232 connection. Installation of this tool and steps on how to use it have been described in detail by the author of

[24]. In this project, it was found that to use the KATCP tool the directory where the KATCP library was stored had to be added to the software path. This was accomplished by the **addpath** Matlab command.

Once the software has established connection to the ROACH, using KATCP and the ROACH's internet protocol (IP) number, it will attempt to load the desired firmware. The firmware must be already saved in the ROACH's internal memory and the name of the firmware must be identical to the name defined in the software. KATCP will attempt to find the firmware defined in the function call in the */boffile* directory in the ROACH file system. Note, during the firmware load command the ROACH automatically executes the selected firmware therefore, if a oscilloscope is connected to the ROACH output, a garbage transmitted signal can be seen. Once the firmware has been loaded, which the only indication of a successfully loaded firmware is the absence of any errors, the software will define the two bit pair used for the PRN sequence generator. The selection of these bits depend on the SV selected. The software holds an internal list of bit pairs, each pair is assigned to a specific SV. Table 2.3 shows which SV gets which pair. The four selected bit pairs are later passed on to the firmware and translated into selector bits for the MUX controlling the PRN generator. For example, if SV 9 is selected then the input bits 3 and 10 of the MUX will be selected and used to generate the PRN sequence.

Following the SV selection process is the generation of all of the message data that is to be written to the BRAM in the firmware. The message data is generated part by hard coded values, most setting specific unused bits to zero, and by fetching an almanac database for the most up-to-date data file. A full description of this process is to follow. Finally, the software configures all of the configuration parameters in the firmware and writes the message data to the BRAM for each individual SV. Section 3.1.2 described a number of switches embedded in

the firmware to give live control of the generated signal. The software uses the KATCP **write** command to set these switches to different values depending on the desired signal output. By default, the software configures all switches to values that produce a true C/A L1 GPS signal.
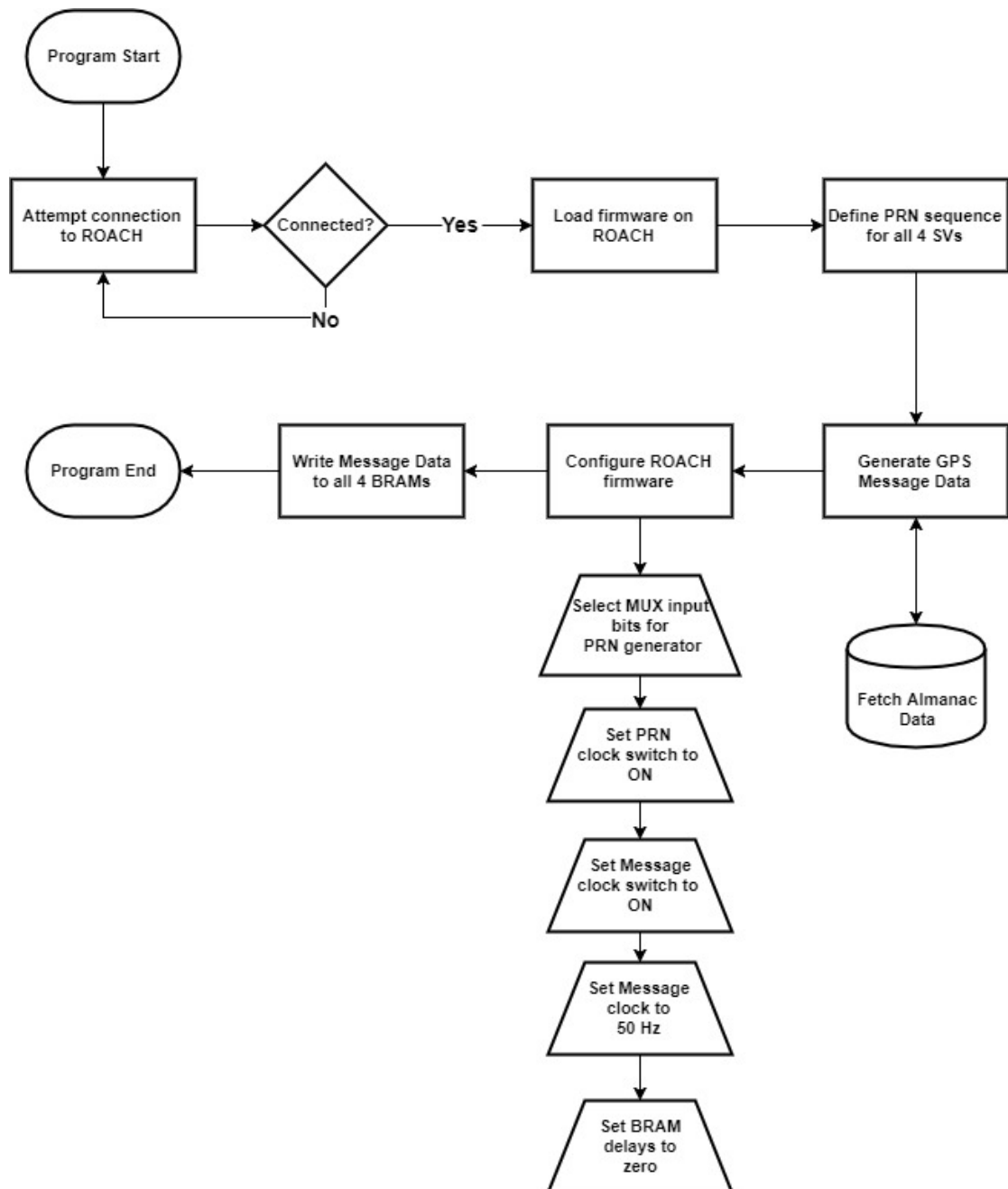

Figure 3.15: High-level software flow diagram

3.2.1 Subframe Generator Functions

A large part of the software functionality is to generate data for each of the five subframes and 25 pages. Those data are converted into bit arrays that are written onto the BRAM in the firmware. The data constructed in these functions are used to modulate the carrier frequency along with the PRN sequence, which is generated purely in firmware. The function *CreateMessageData(sv_selected)* is called when generation of message data is requested. It takes in vector composed of 4 integer values, with each relating to the selected SV. The function uses the selected SV number to fetch the proper data for each of the four SVs selected from the almanac database. Initially, the function defined the GPS epoch, a reference time used by the GPS, and calculates GPS week and the seconds of the week. This process, shown in the code snippet below, provides the system with accurate GPS time. These time values are later passed on to every subframe and are used by the GPS received to calculate PVT.

```
1  gps_epoch = [ 1980 1 6 00 00 00.000 ]; % Jan 6 1980 00:00:00.000
2  correction_EST = 4/24;  % Time correction for EST
3  gps_week = (( now() + correction_EST ) - datenum( gps_epoch ))/7;
4  true_gps_week = floor( gps_week ); % GPS week number
5      gps_seconds_of_week = floor((( gps_week - true_gps_week)...
6      *hours_in_day*minutes_in_hour*seconds_in_hour*days_in_week));
```

Once current GPS time has been calculated, the fetching of almanac data starts by calling the *fetchYumaData()* function. By determining the current day of the year and passing it to the end of the database URL, it downloads the almanac data for the current day. The algorithm used will always determine the current day of the year and fetch the most up-to-date almanac data available from the database. The almanac data is saved to the *.alm* file stored in the current directory, usually the directory containing all of the function files. Prior to returning the save file to the calling function, the recently generated almanac file is checked

to ensure that it contains data for 32 total satellites. Although at the time of this writing only 31 SVs were operational, the almanac data still transmits data for 32 different satellites. To fix this, the fetching function fills in dummy data for missing SV. It iterates through each data entry in the almanac file, when a missing SV is detected it fills in the missing data with the dummy data. This process follows recommended instructions in [19] and the dummy data is defined to be the recommended values.

With almanac data available the generation of 5 subframes for 25 pages starts. A design choice made was to have one function for each subframe. A total of 5 functions generate all of the data for the message signal. Two additional functions generate the HOW and TLM word for the subframes. These two functions are called by all 5 subframe functions to generate the first two words of data. Figure 3.16 describes the function calling process executed during the subframe data generation. A loop dictates how many times the 5 subframes need to be generated. Each time a subframe function is called, that function calls both the TLM and HOW functions to generate the first 2 words (60 bits) of the subframe. The following snippet of code shows how subframe 1 generates all of its 300 bits. It starts by defining the subframe number then calling a function to generate each of the 10 words, each requiring different parameters depending on what data needs to be generated for that word.

```matlab
function  subframe_1_300_bits  = GenerateSubframe1( ...
        GPS_week_number, TOW_truncated, sv_health,...
        sv_af0, sv_af1, D_star  )
    frame_id = [ 0 0 1 ];
    word_1  = GenerateTLMWord( D_star );
    word_2  = GenerateHOWWord( TOW_truncated,...
                              frame_id, word_1( 29:30 ));
    word_3  = GenerateWord3( GPS_week_number,...
                              sv_health, word_2( 29:30 ) );
    word_4  = GenerateWord4( word_3(29:30) );
    word_5  = GenerateWord5( word_4(29:30) );
    word_6  = GenerateWord6( word_5(29:30) );
    word_7  = GenerateWord7( word_6(29:30) );
    word_8  = GenerateWord8( word_7(29:30) );
    word_9  = GenerateWord9( sv_af1, word_8(29:30) );
    word_10 = GenerateWord10( sv_af0, word_9(29:30) );
    subframe_1_300_bits = [ word_1 ; word_2 ; word_3 ;
                            word_4 ; word_5 ; word_6 ;
                            word_7 ; word_8 ; word_9 ;
end
```
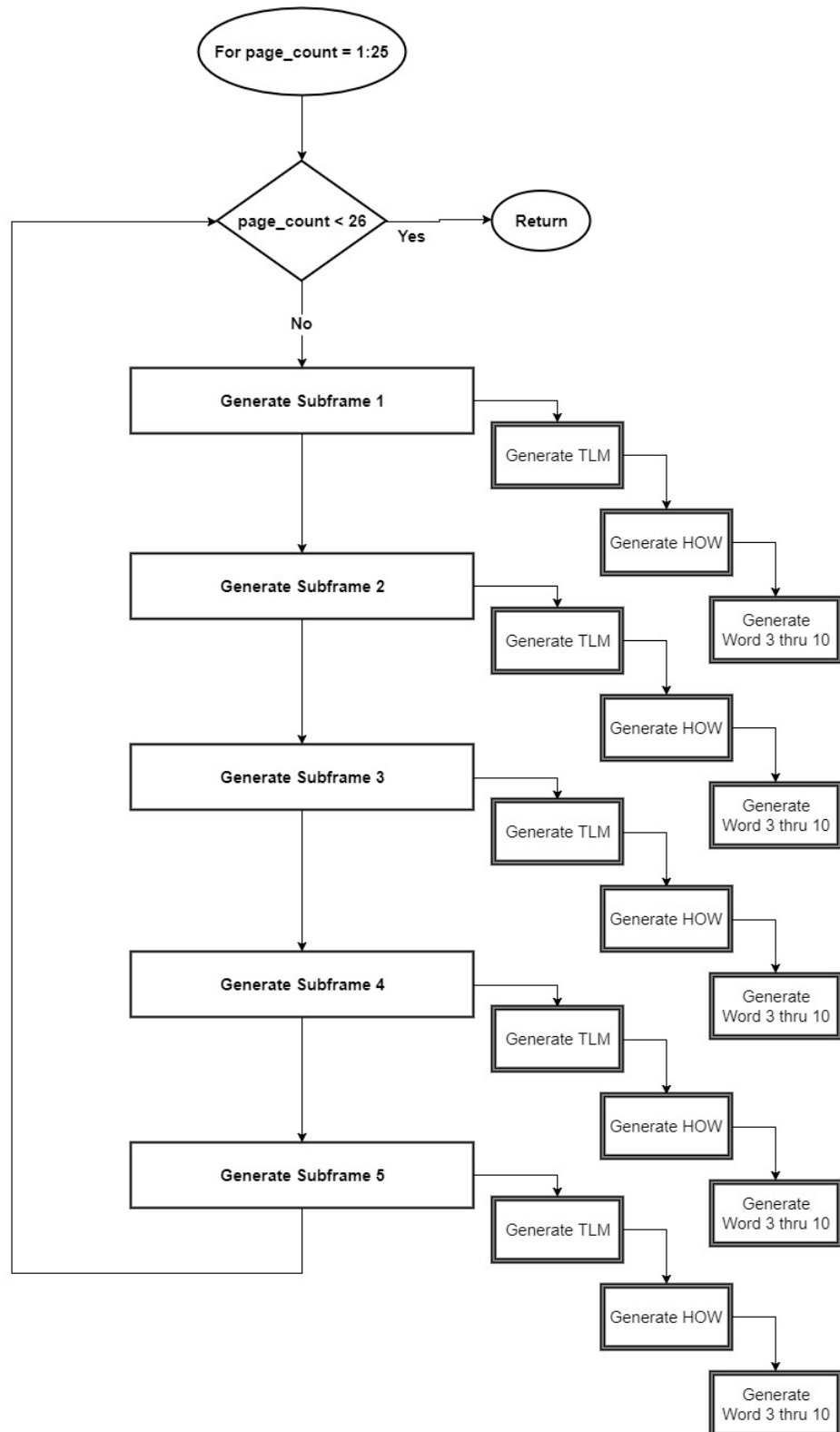
Figure 3.16: High-level function flow diagram for the subframe generation process

Functions generating subframes 2 through 5 use much of the same design as sub-

frame 1. The major changes occur in subframe 4 and 5 whose data depend on the current page number. During the function calls for subframe 4 and 5 each function call for word generation checks the current page number. Depending on the page number it will generate a set of data specific for that page number. As explained in section 2.4.3, subframe 4 and 5 are subcommuted 25 times. Subframe 5 transmits almanac data for SVs 1 through 24 and subframe 4, pages 2, 3, 4, 5, 7, 8, 9 and 10, transmitting almanac data for SVs 25 through 32 respectively. As such the word generator functions check the page number to determine the proper data to generate. Note that the word generator functions do not actually create the data, they read the data from the almanac file and converts it to bit arrays. Each word generator function returns a vector array containing 30 bits. A detailed summary of the various data contained in each of the pages of subframe 4 and 5 is shown in table 3.1.

Subframe	Pages	Data
4	2, 3, 4, 5, 7, 8, 9 and 10	Almanac data for SV 25 through 32
	1, 6, 11, 16, and 21	Reserved for future use
	12,19,20,22,23 and 24	Reserved for future use
	13	Navigation Message Correction Table
	14 and 15	Reserved for system use
	17	Special Messages
	18	Ionospheric and UTC data
	25	Anti-spoofing flag, SV health for SV 25 through 32
5	1 through 24	Almanac data for SV
	25	SV health for SV 1 through 24, Reference Time, Reference Week Number

Table 3.1: Summary of the various data contained in subframe 4 and 5

The subframe 4 function uses a polynomial equation to generate the SV index. The SV ID is a number between 25 and 32 that identifies the SV. This number is a 6 bit number that is store in word 3 of subframe 4 pages 2, 3, 4, 5, 7, 8, 9, 10, and subframe 4 pages 1 through 14. The SV ID in subframe 5 is the same as the page number. The use of this polynomial equation is only necessary for the pages mentioned above of subframe 4. This is because the SV ID in subframe 5 are the same as the page number, page number 1 in subframe 5 transmits almanac data for SV 1 therefore the SV ID is also 1. However, there is a discontinuity on

57

the relationship between page number and SV ID for subframe 4. In those pages almanac data is contained, the SV ID does not equal the page number. Instead of creating *if/else* logic for all 9 pages in subframe 4 and in turn increasing executing time and resources, a single polynomial equation is used to generate the SV ID. Below is the line of code that defines and calculates the polynomial equation. This polynomial is calculated once at the beginning of subframe 4 and depends only on the current page number.

```
sv_ID =  round( ( -1787/181440 ) * page_number^9 + ...
            ( 2161/4480 ) * page_number^8 - ...
            ( 152611/15120 ) *  page_number^7 + ...
            ( 339491/2880 ) * page_number^6 - ...
            ( 1455307/1728 ) * page_number^5 + ...
            ( 21846103/5760 ) * page_number^4 - ...
            ( 969047939/90720 ) * page_number^3 + ...
            ( 181025839/10080 ) * page_number^2 - ...
            ( 40634899/2520 ) * page_number + ...
            5789 );
```

The generation of this polynomial used a numerical analysis known as Newton polynomial, named after Isaac Newton. This numerical analysis calculates the coefficients of the polynomial using divided differences method. The author of [26] explains the process of divided differences in detail, which was the same process used to find the polynomial equation used.

To overall process of generating all of the message data ends with writing each data set to a specific BRAM in the firmware. While the firmware and software are separate entities, the software, through KATCP, can communicate with the firmware. This communication allows the software to command different registers that control the signal structure. It also allows access to all of the BRAM blocks defined in the firmware. The software uses the KATCP functions **wordwrite** and **write** to control the registers and write to the BRAM block. The following snippet of code shows the configuration of the PRN register bit selectors followed

by writing the message data sets to their respective BRAM blocks.

```matlab
% PRN Register bit selector
pause( global_pause );wordwrite( roach, 'G2_1_SV_SEL_REG1',...
    (selected_bit_sv1(1,1)-1));
pause( global_pause );wordwrite( roach, 'G2_1_SV_SEL_REG2',...
    (selected_bit_sv1(1,2)-1));
pause( global_pause );wordwrite( roach, 'G2_2_SV_SEL_REG1',...
    (selected_bit_sv2(1,1)-1));
pause( global_pause );wordwrite( roach, 'G2_2_SV_SEL_REG2',...
    (selected_bit_sv2(1,2)-1));
pause( global_pause );wordwrite( roach, 'G2_3_SV_SEL_REG1',...
    (selected_bit_sv3(1,1)-1));
pause( global_pause );wordwrite( roach, 'G2_3_SV_SEL_REG2',...
    (selected_bit_sv3(1,2)-1));
pause( global_pause );wordwrite( roach, 'G2_4_SV_SEL_REG1',...
    (selected_bit_sv4(1,1)-1));
pause( global_pause );wordwrite( roach, 'G2_4_SV_SEL_REG2',...
    (selected_bit_sv4(1,2)-1));
% Write the message signal data to BRAM
pause( global_pause );
write(roach, 'Message_Signal1_bram1',...
    repeated_message_signal_bytes_sv1' );
pause( global_pause );
write(roach, 'Message_Signal2_bram1',...
    repeated_message_signal_bytes_sv2' );
pause( global_pause );
write(roach, 'Message_Signal3_bram1',...
    repeated_message_signal_bytes_sv3' );
pause( global_pause );
write(roach, 'Message_Signal4_bram1',...
    repeated_message_signal_bytes_sv4' );
```

The **wordwrite** command connects to the object *'roach'*, which is the connected ROACH development board running the active firmware, and writes the values of *'selected_bits_sv'* to the G1 and G2 selector registers. The **write** command is used to write the data stored in the *'repeated_message_signal_bytes_sv'* to the BRAM blocks. Note the constant use of the **pause** command. It was found through the process of trial and error that the firmware needed time before receiving a new write command.

3.2.2 Parity Function

Every subframe used the party function, *GpsParityMaker()*, to calculate the 6 parity bits. The algorithm used for this function is defined in [19]. In summary, the function takes in a 24 or 22 bit message and calculates 6 or 8 parity bits, respectively. If a 22 bit message is passed to the function a forced parity sequence is used to force the parity calculation to set bits 23 and 24 to zero by generating some 6 bit sequence. In turn, if a 24 bit message is passed the parity function will calculate the 6 bit parity sequence. The parity bits depend on the message bits. The parity function looks at the last two bits of the previous message, which are passed to the function, and uses it to calculate the parity. If the last bit of the previous word is a '1' then all of the message bits must be negated, meaning '1' become '0' and vice-versa. The parity calculation uses a defined sequence for each parity bit. The calculations depend on the sequence and the last two bits of the previous word. The returned value from the parity function is a 30 bit number containing the message bits and the newly calculated 6 parity bits. In the case of the force parity, the return value is the message bits having bit 23 and 24 set to zero and the newly calculated parity bits.

3.2.3 Supporting Functions

A few supporting functions were implemented to assist with the generation of the message data. A brief explanation of each function and how it assisted the data generation is to follow. Some of these functions converted data types or format, others assisted in signal generation.

SvData2Binary(): This function takes in a decimal value and returns its binary equivalent. It differs from any Matlab built-in function because the return value is an array of bits and not a string of bits. Before converting to a bit array this

function also checks to see if the input decimal value is a negative number. If so, it calculates the two's complement and generates the negative binary representation of the value. This function is called when data are fetched from the almanac and written to the message data array stored in the BRAM.

cacode(): This function was written by Dan Boschen [27]. It generates a given PRN sequence. It takes in as a parameter the desired PRN number to be generated and the desired bit sample rate. The function is used during post-processing to generate a PRN signal to be correlated against the received GPS signal.

read_complex_binary(): This function is provided by GNU Radio project to extract the digital signal form the *.dat* file. During recording of the signal using the Ettus N210 a file is saved that contains a complex representation of the received signal. This function reads that file and returns an array of data points usable by Matlab for post-processing.

ConverToBytesAndPad(): This function returned a 10 by 30 bit array as a 4 by 10 byte array. Each 30 bits of the input array would be padded, adding two bits to the LSB, to include 32 total bids then converted into bytes, 8-bit chunks. The byte values would be stored in a column matrix. Each column representing a 32 bit word. This matrix is returned and used as the data set written to the BRAM.

3.2.4 Lessons Learned

The Matlab function **urlwrite** does not recognized self-signed web certificates. It only accepts trusted authorities to determine if a specific certificate should be trusted. The server used for fetching almanac data uses a HTTPS protocol and the certificate used is not accepted by Matlab; therefore the certificate must be manually declared as a trusted certificate. This is accomplished by first downloading the certificate file using a web browser. Running the code shown in source code

1 to add the certificate as a trusted MATLAB JRE key store. Execution must be performed with administrator (or root if on Linux) permissions and Matlab must be restarted after completion.

```matlab
1   function importcert(filename)
2       if (nargin == 0)
3           % Show open file dialog to select
4           [filename,path] = uigetfile({'*.cer;*.crt','Certificates
5               (*.cer,*.crt)'},'Select Certificate');
6           if (filename==0), return, end
7           filename = fullfile(path,filename);
8       end
9       % Determine Java keytool location and cacerts location
10      keytool = fullfile(matlabroot,'sys','java','jre',...
11              computer('arch'),'jre','bin','keytool');
12      cacerts = fullfile(matlabroot,'sys','java','jre',...
13              computer('arch'),'jre','lib','security',...
14              'cacerts');
15      % Create backup of cacerts
16      if (~exist([cacerts '.org'],'file'))
17          copyfile(cacerts,[cacerts '.org'])
18      end
19      % Construct and execute keytool
20      command = sprintf...
21      ('"%s" -import -file "%s" -keystore "%s" -storepass changeit',
22      keytool,filename,cacerts);
23      dos(command);
24  end
```

Source Code 1: Manual addition of trusted certificate to the Matlab key store

A new almanac is generated and available in the database every 23.9 hours. To fetch the almanac the software calculates the current day of the year. At times, the calculated day of the year may reference a day that a new almanac has not yet been generated for. This will result in a fetching error and the software will terminate. Simply changing the function **datenum** input date to January 1, 0000 will change the day of the year calculation to be one less than the previous calculation resulting in a successful fetch.

Lastly, at times when writing to the firmware registers the software will hang and not execute any of the write commands. The cause of this was not identified; however, arbitrarily removing the **pause** commands would change the behavior of the execution and fix the problem. Whenever this problem was encountered the current firmware would be manually terminated via SSH connection to the ROACH and the software executed after changes to delay command were made.

3.3   GPS Software Decoder

As an alternative to signal acquisition using a GPS receiver demodulation of the transmitted signal was accomplished in software. The transmitted signal, C/A L1 GPS signal, was recorded using the Ettus N210 and saved to file. The file was read by a Matlab scripts that extracts the received signal as complex values of each sample point recorded. These complex values were analyzed using signal processing methods so the transmitted bits, modulated into the signal, could be extracted.

3.3.1   Post-Processing

The post-processing demodulated the signal and extracted the message bit stream. Post-processing analysis accommodates recorded signal at a sample frequency f_s of 4 MHz and 8 MHz but explanation of the analysis assumes $f_s = 4\,\mathrm{MHz}$. Analysis differences dependent on the sample rate were highlighted.

The first step was to recover the carrier by estimating the frequency difference between the recorded signal and the Ettus N210 local oscillator (LO). The recorded signal was shifted by an initial guess of 98 kHz. The first shifted signals, called coarse tune signal, was squared and the resulting recovered carrier was checked for proper shifting of the frequency to DC. A more precise tuning of the frequency

shift was done using the coarse tune signal. The maximum frequency found in the recovered carrier from the coarse tune signal was indexed and the difference from DC measured. This difference was used as a more precise second guess for shifting the coarse tune signal to DC. The second shifted signal, called fine tune signal, was successfully shifted to DC. Figure 3.17 shows the power spectrum of the coarse tune (top graph) and fine tune (bottom graph) signals.
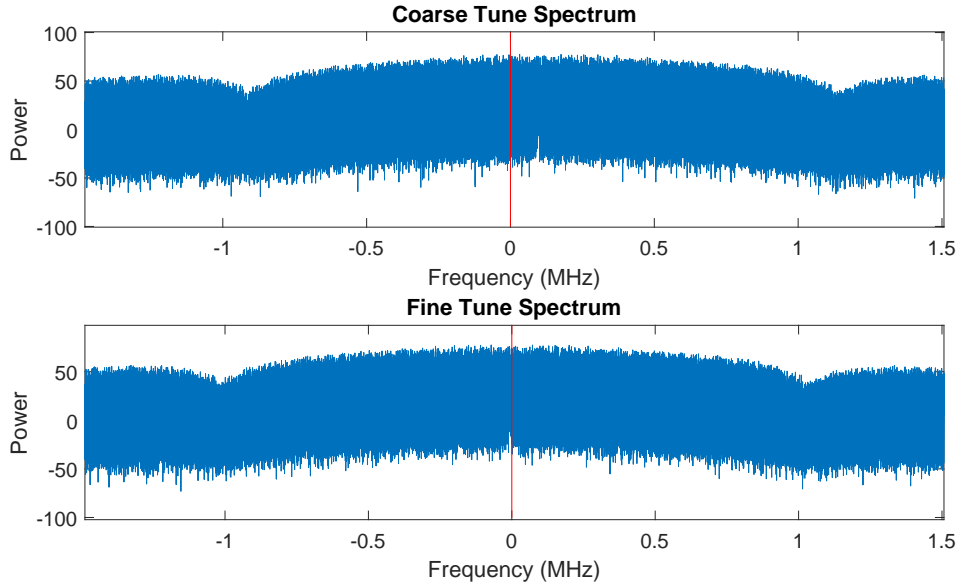

Figure 3.17: Power density spectrum of both shifted signals

The second step was determining the clock rate of the recorded signal. Knowing the clock rate would defined the rate at which the bits changed. The expected clock rate was the PRN clock frequency, $f_{PRN} = 1.023\,\mathrm{MHz}$ but due to differences in the recorded signal a new clock rate had to be estimated. The clock rate was estimated by comparing it to an initial guess calculated based on the clock edges of the recorded data. The clock edges were determined by setting a threshold as half of the median absolute value of amplitudes in the fine tune signal. The use of a threshold helped reduced error by excluding the first 25% of the recorded data avoiding the fragmented data that routinely happened at the beginning of a recorded file. Change in angle greater than the threshold were identified and

64

sorted. From the list of sorted angles a $\frac{\pi}{2}$ difference in angle was identified as a signal bit change. Expecting that one bit should persists for $\frac{f_s}{f_{\mathrm{PRN}}}$ samples, at $f_s = 4\,\mathrm{MHz}$ a single bit lasts about 4 to 5 samples. This was checked by counting the number of bits that lasted 3, 4, and 5 samples. The average of those bits lengths was used to guess the clock rate of the recorded signal. Figure 3.18 shows all of the bits found that lasted 3, 4, or 5 samples. The estimation of the clock rate was done a second time using the average bit length. Each estimation sets the recorded signal clock rate closer to the expected clock rate of 1.023 MHz
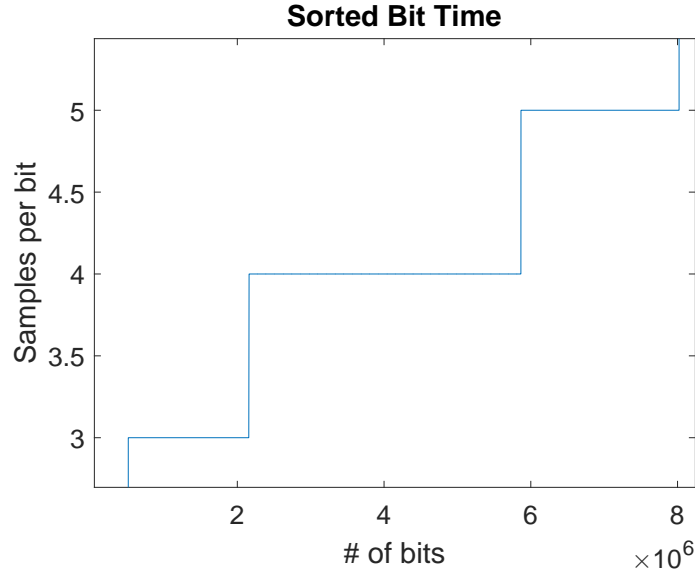

Figure 3.18: Bit sample length at 4 MHz sample rate

The third step was to generate a synthesized C/A code signal, a PRN code sequence generated in software. This was accomplished by using a Matlab script developed by [27]. The synthesized signal was resampled to the calculated clock rate of the recorded signal and generated for a PRN sequence known to be part of the recorded signal. The resulting signal was cross correlated against the fine tune signal. The result of the correlation showed the existence of the synthesized signal in the recorded signal in turn proving that a PRN sequence was correctly generated by the firmware and successfully transmitted to the Ettus N210. Figure

3.19 shows the cross correlation result between a synthesized C/A code signal for PRN 30 and a recorded signal containing the same C/A code sequence. Note the inclination of the correlation result. This inclination indicates a residual time difference between frames.
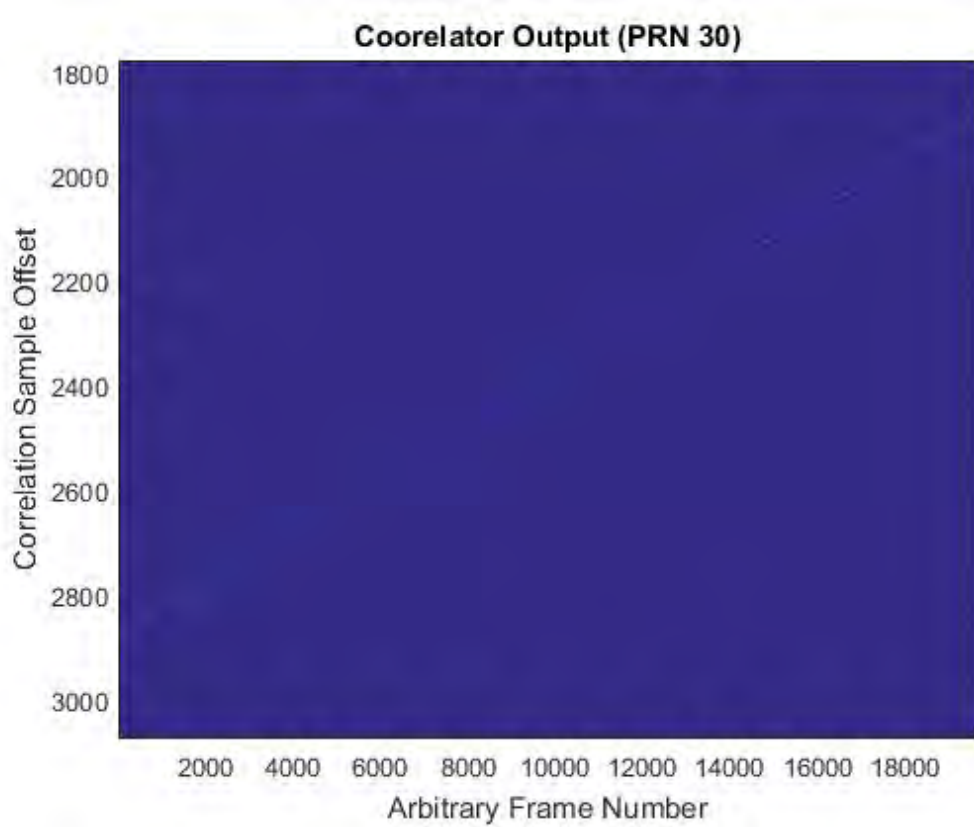

Figure 3.19: Cross correlation result for PRN 30

The recorded signal is demodulated using the synthesized C/A code, meaning that the PRN signal is removed and all that is left is the carrier modulated by the GPS message bits. Before extraction of the message bits was done the residual time difference was investigated to determine if it caused any bit loss. The message bits at a clock frequency f_{msg} of 50 Hz were expected to last for $\frac{f_{\mathrm{PRN}} \cdot f_s}{f_{\mathrm{msg}}} = 81840$ samples per bit. The loss rate due to the residual timing difference was calculated by the mean timing difference between each 2 ms correlation frames divided by the f_s. Therefore the calculated message bit length was determined by the integer result of the sum between the actual and expected bit lengths multiplied by the

loss rate.

The extraction of the message bit stream was done by determining how many bits could be contained in the recorded carrier signal given the calculated number of sampled a single message bit would take. Then, by iterating through each sample point in the recorded carrier signal and mapping the sample count to the phase change. Where a phase change of more than $\frac{\pi}{2}$ happened determined a new bit. Figure 3.20a shows the raw bit stream extracted from the signal containing the PRN 30 C/A sequence. After applying a threshold value to the raw bit stream values were normalized to 0 and 1. Figure 3.20b show the normalized bit stream with 0 or 1 bit values. Figures 3.21a and 3.21b shows a zoomed in view of the raw and normalized bit stream from bit number 580 to 780. Note how the bit values has changed but the bit change remained the same.
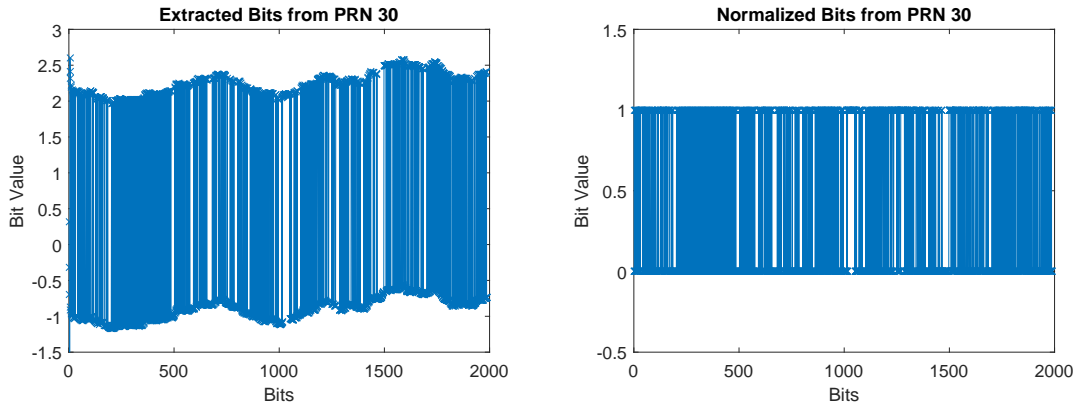
(a) Message data bit stream without normalization

(b) Message data bit stream after normalization

Figure 3.20: Message data extracted from PRN 30 before and after normalization
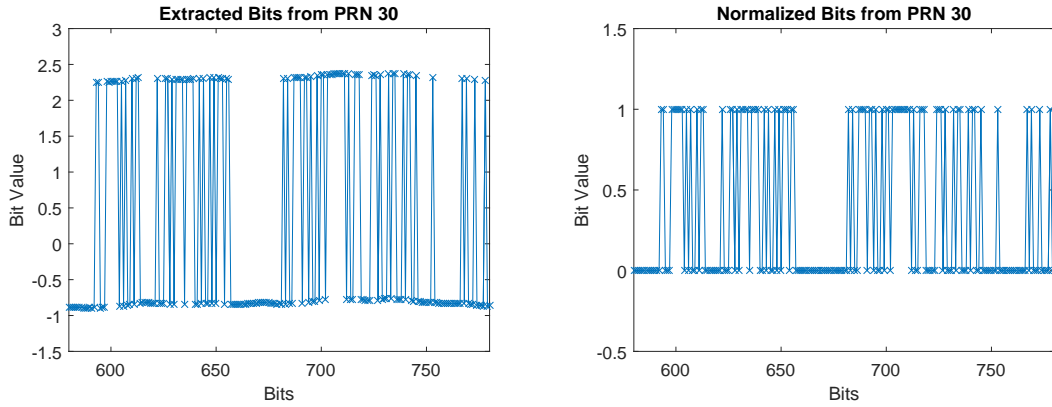
(a) A section of the message data bit stream without normalization

(b) A section of the message data bit stream after normalization

Figure 3.21: Message data bit stream from bits 580 to 780 before and after normalization

3.3.2 Decoder

The software decoder is made up of seven different functions. The decoder works by analyzing a stream of bits and identifying bit groupings that relate to specific terms in the message data. The development of the decoder was done by using the GPS NAVSTAR documentation [19] and knowledge of the signal expected to be parsed was not used. The diagram shown on figure 3.22 gives a high-level view of how the decoder parses the received bit stream.

The decoder uses the NAVSTAR standard to define each specific bit sequence contained in the message data. First, it searches the bit stream for the presence of a preamble. The preamble is an 8-bit sequence that defines the start of a subframe. When a start of a subframe is found, the decoder extracts 292 bits that follow the preamble. Those 292 bits plus the 8-bit preamble make up a single subframe.
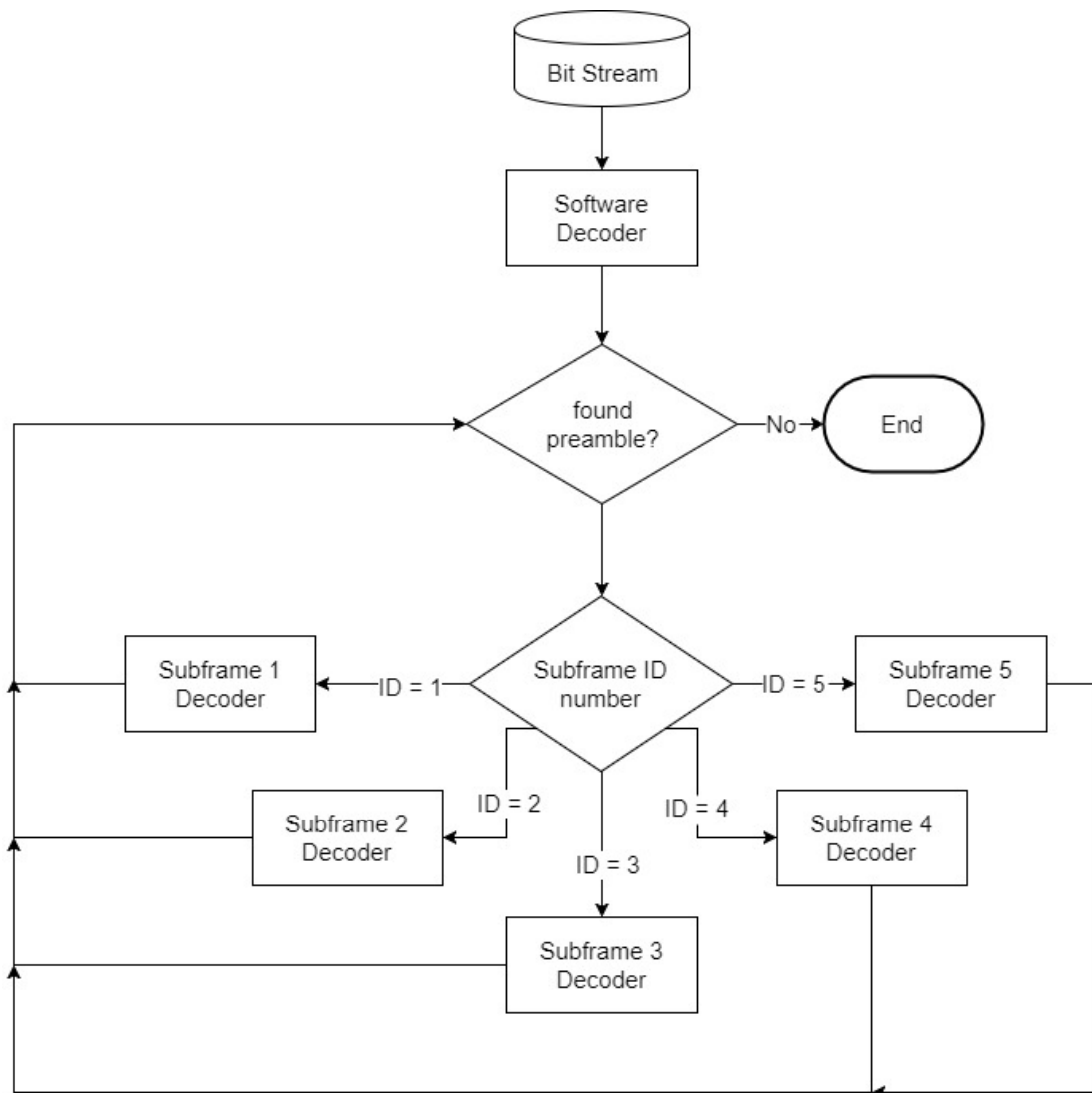
Figure 3.22: High-level software decoder design

With a full subframe extracted, the decoder follows the NAVSTAR specification to identify the subframe ID number. Depending on what subframe ID number is found a specific function parses the extracted subframe. A further analysis is done for subframes 4 and 5 that depend on the current page number. As such, the decoder finds the page ID value and parses it accordingly. If the page ID or the subframe ID are values not recognized by the decoder then that subframe is considered invalid and those 300 bits are ignored. This process repeats until all bits in the received bit stream have been analyzed.

3.4 Signal Acquisition

To acquire the generated GPS signal different hardware and software methods were used. Instead of testing the acquisition of the generated GPS signal using an off-the-shelf GPS receiver, a combination of software and hardware was used that would give better control over the acquisition process. GPS signal acquisition is the process where a receiver identifies a signal from a specific SV by using the received PRN signal and calculates the Doppler shift in the carrier frequency. General steps taken by a common receiver are to first determine all SVs currently in range of the receiver, calculate each SV signal Doppler shift, and detect the signal C/A code delay and carrier frequency.

3.4.1 Ettus N210 SRD

The Ettus N210 is an SDR that can be configured to either transmit or receive virtually any type of signals. It is powered by a Xilinx Spartan 3A-DSP 3400 FPGA [28]. It provides a dual 100 MS/s analog-to-digital converter (ADC), and a dual 400 MS/s DAC. For connectivity between a host PC, it uses a Gigabit Ethernet connection. Different daughterboards and radio frequency (RF) front ends can be purchased to provide the Ettus N210 with transmitter, receiver, or transceiver capabilities ranging from DC frequencies all the way up to 6 GHz. Figure 3.23 shows the Etttus N210 used in this project with its lid open where the internal board and the daughterboard can be seen.
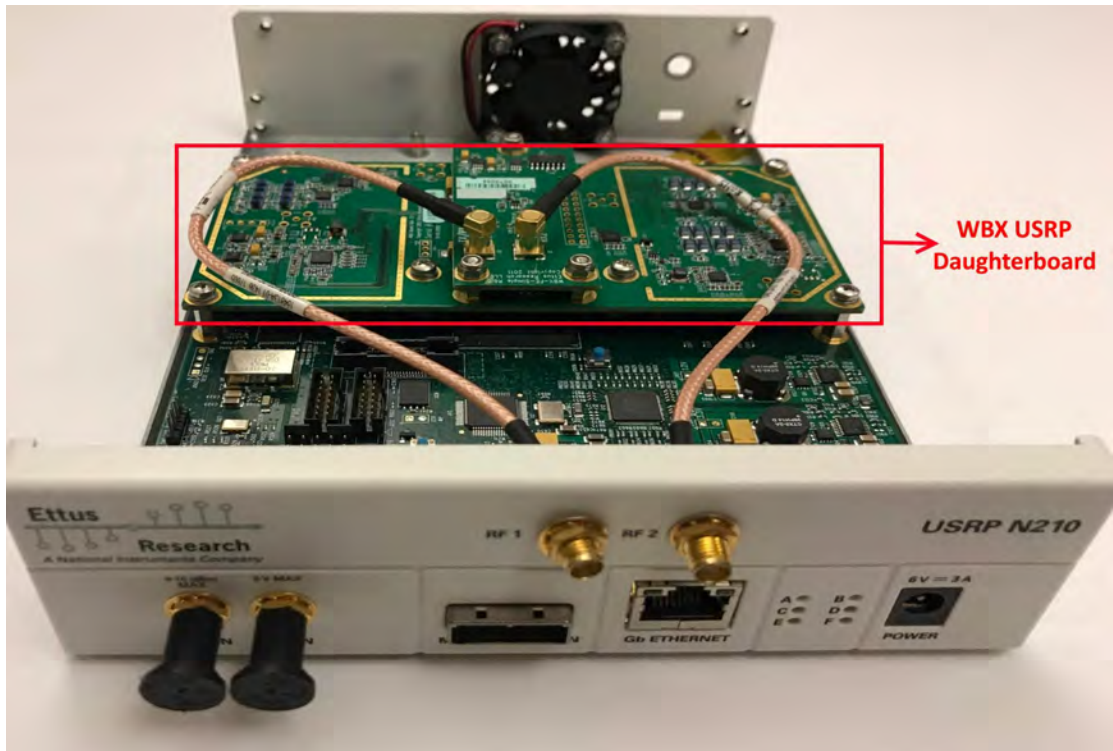
Figure 3.23: Ettus N210 SDR showing the internal board and the WBX daughterboard

Configuration of the Ettus N210 uses signal processing software to create flow graphs using pre-programmed blocks, these blocks include code written to execute a given task, which is then complied down to FGPA logic where it is represented by hundreds or thousands of small lookup tables (LUT) known as logic cells. These logic cells are D flip-flops and a 2 to 1 MUX interconnected together to create complex logical functions.

The Ettus N210 served as the GPS receivers used to test the generated signal. The generated signal, transmitted by the ROACH FPGA, was transmitted via a physical cable to the Ettus N210 *RF 1* input. The Ettus N210, with the help of software running on a target PC, processed the received signal and produced records later used to compare the generated signal to. As opposed to using an off-the-shelf GPS receiver, the Ettus N210 gave a level of modularity perfect for the project. While a standard off-the-shelf GPS receiver would have only been

able to receive signals at the GPS civilian frequencies, 1575.42 MHz, the Ettus N210, using the WBX daughterboard, allowed a frequency range from 50 MHz to 2.2 GHz. This was useful due to national restrictions on over-the-air transmission of GPS signals allowing the transmission of the generated GPS signal to be over-the-cable and at a different fequency, 50.127 MHz, than the GPS L1 standard. Due to limitations of the Ettus signal synthesizer its frequency range is reduced to 68.75 MHz to 2.2 GHz. This required experiments to target the third harmonic of the transmitted signal, 150.381 MHz, instead.

3.4.1.1 WBX USRP Daughterboard

A daughterboard is a circuit with RF components designed to transmit or receive signals at a set frequency range. It prepares a received signal to be processed by the FPGA or prepares a generated signal to be transmitted by the RF front-end. The DAC and ADC work between the daughterboard and the FPGA converting the signal to digital, if its a received signal, or to analog, if its a signal to be transmitted. Figure 3.24 shows a high-level diagram of the system.

The WBX USRP daughterboard was designed to fit on the Ettus N210. It is a transceiver, it both transmits and receives signals, with a bandwidth capability of 40 MHz, and a frequency range of 50 MHz to 2.2 GHz. The WBX uses the ADF4350 wideband synthesizer with integrated voltage controlled oscillator (VCO). It can generate a range of frequencies from 2200 MHz to 4400 MHz having the additional capability to divide these frequencies by 1, 2, 4, 8 and 16 allowing to output an RF frequency as low as 137.5 MHz [29]. Therefore the operational frequency range is 68.75 MHz to 2.2 GHz, which is half of the upper and lower limits of the synthesizer due to another division by 2 performed by the quadrature demodulation chip on the WBX board. It is important to distinguish the difference between the reported frequency range and the operational frequency

range. Since the desired received signal is transmitted at 50.127 MHz which falls within the reported range but outside the operational range and therefore cannot be processed.
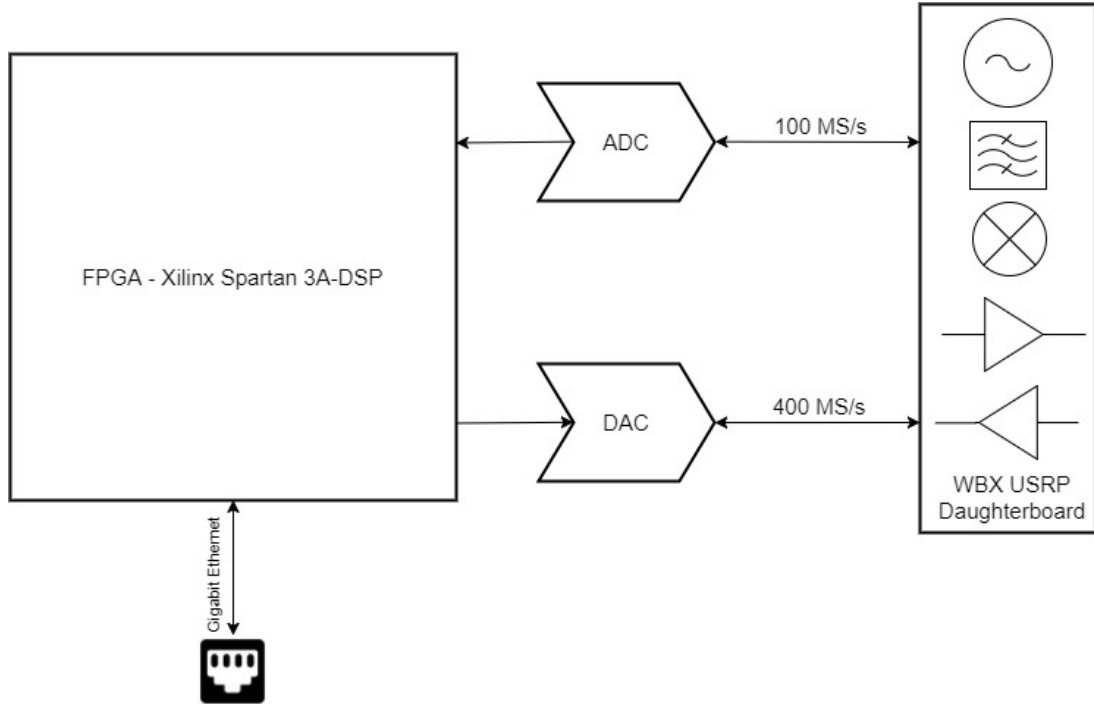

Figure 3.24: High-level diagram of the Ettus N210 and WBX daughterboard connection

3.4.2   GNU Radio Software

GNU Radio is a open-source and free software that provides a variety of SDRs with different signal processing capabilities. It is native to Linux operating system distributions. It is often used by academia, hobbyists, and some commercial products. As of the time of this writing the last stable release was version *3.7.10.2*.

The use of GNU Radio in this project facilitated with signal acquisition, signal analysis, and data recording. The use of pre-packaged blocks help speed up the acquisition process which would otherwise require a considerable amount of time to develop. The data recording and signal analysis process was accomplished using

GNU Radio Companion (GRC), a graphical user interface (GUI) that uses blocks to process a wide variety of signals. GPS signal acquisition was performed using GNSS-SDR software, which was built using the GNU Radio framework.

The installation process of GNU Radio was done by following various online guides. Due to a considerable number of failed installation attempts, a more inclusive script was written that applied specifically for the hardware and operating system (OS) used for this project. PyBombs and UHD were two add-on installations required in conjunction with GNU Radio installation. PyBombs (Python Build Overlay Managed Bundle System) is a dependency that manages all of the "out-of-tree" modules which are signal processing blocks created by GNU Radio users. UHD (USRP Hardware Drivers) are drivers required by GNU Radio if any of the Ettus USRP SDRs are to be used. The scrip shown in source code 2 was written to successfully install GNU Radio, PyBombs, and UHD. The installation procedures were tested using Ubuntu 14.04 LTS.

```bash
#!/bin/bash
currentuser=$USER
# Change to sudo user
sudo su
# Make sure you have sudo access
if [ "$(whoami)" != "root" ]; then
    echo "You are not root."
    exit 1
fi
# Change to root directory
cd ~
# Update OS packages and upgrade OS
apt-get update && time apt-get dist-upgrade
# Install GNU Radio
apt-get install gnuradio
# Open GNU Radio companion
gnuradio-companion
# Install git package
apt-get install git
# Change directory to Downloads
cd ~/home/$currentuser/Downloads
# Clone and install PyBombs
git clone git://github.com/pybombs/pybombs
cd pybombs/
# Open PyBombs GUI. Prepare to install UHD using GUI
./app_store.py
```

Source Code 2: Installation script for GNU Radio and dependencies

Note that on line 26 of the script the user must take over installation and use the PyBombs GUI to install UHD. To install the UHD, navigate to *Available hardware Apps* tab on the GUI. Select the *gr-ettus* blue circular icon to install UHD. Running the bash script and following the UHD steps listed above will have successfully installed all required software to run GNU Radio with the Ettus N210 SDR.

Data recording and signal analysis was done by creating a simple flow graph using GNU Radio blocks. Figure 3.25 shows the flow graph used. The flow graph is

composed of a USRP Source block, a file sink block, and a QT GUI sink block.
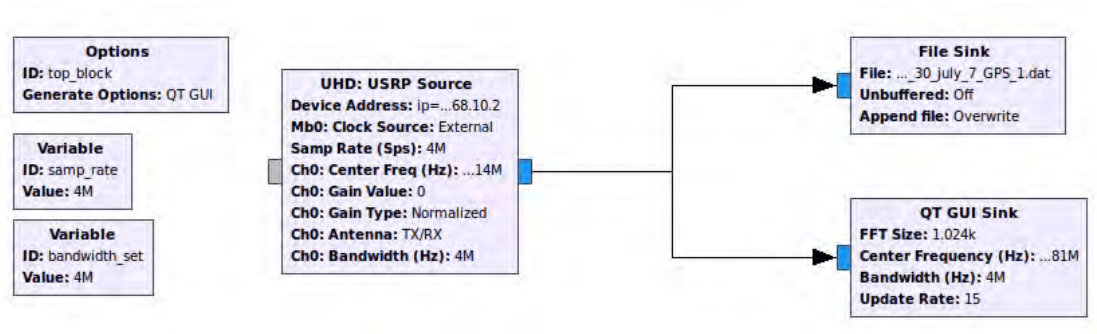

Figure 3.25: Data recording and signal analysis GNU radio flow diagram

Each block in GRC is designed to show the flow and type of data as it is laid out in the flow graph. The flow of data is represented by arrows that connect one block to the next and point to the direction of the data flow. The data type is represented by the color of the small square-like ports on the sides of the block. The USRP Source block displays a gray square port on its left side indicating that it does not contain any incoming data and a blue square port on its right side indicating that a complex data type (the color blue indicates complex type) is transmitted from it. The parameters of the USRP Source block, shown in figures 3.26a and 3.26b, allow the user to define the IP address, clock source, sample rate, and many other parameters. Note parameters *sample rate* and *bandwidth* are assigned to variables which are defined by the *Variable* blocks located on the left side of the flow graph. These variable blocks are used only to define variables and do not generate or receive any data. The *center frequency* parameter was set to 150.381 MHz, which is the third harmonic of the transmitted carrier frequency, 50.127 MHz. The use of the third harmonic instead of the carrier frequency is due to the limitation of the Ettus N210 which is only able to receive frequencies above 68.75 MHz.
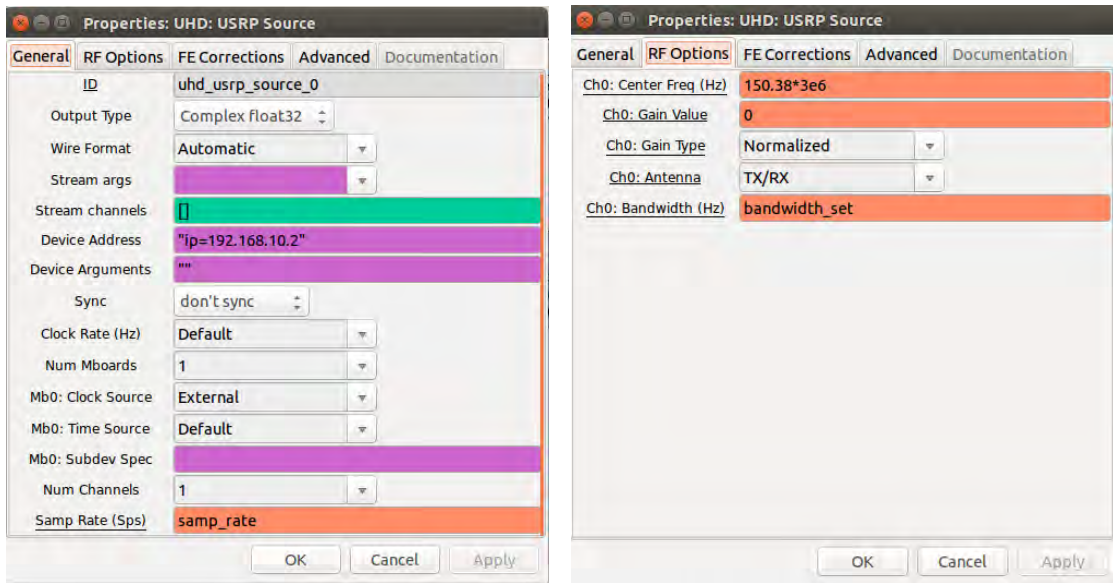
(a) USRP Source block general properties

(b) USRP Source block RF properties

Figure 3.26: USRP Source block parameters

The File Sink block receives complex data type and saves it to a specified *.dat* file name. It gives the option in its properties to either overwrite the file or append, and to record the data unbuffered or buffered. The unbuffered option increases the chance to lose data but increases the write speed while the buffered option is slower but decreases the chance to lose the data before writing it to file. Properties for the file sink block are shown in figure 3.27a. The QT GUI Sink block is used to analyze the signal. It receives a complex data type and displays its content in a series of plots. The plots used the most were a time domain plot and frequency domain plot, however, it could also display waterfall and constellation plots. The properties for the QT GUI Sink block are shown in figure 3.27b.
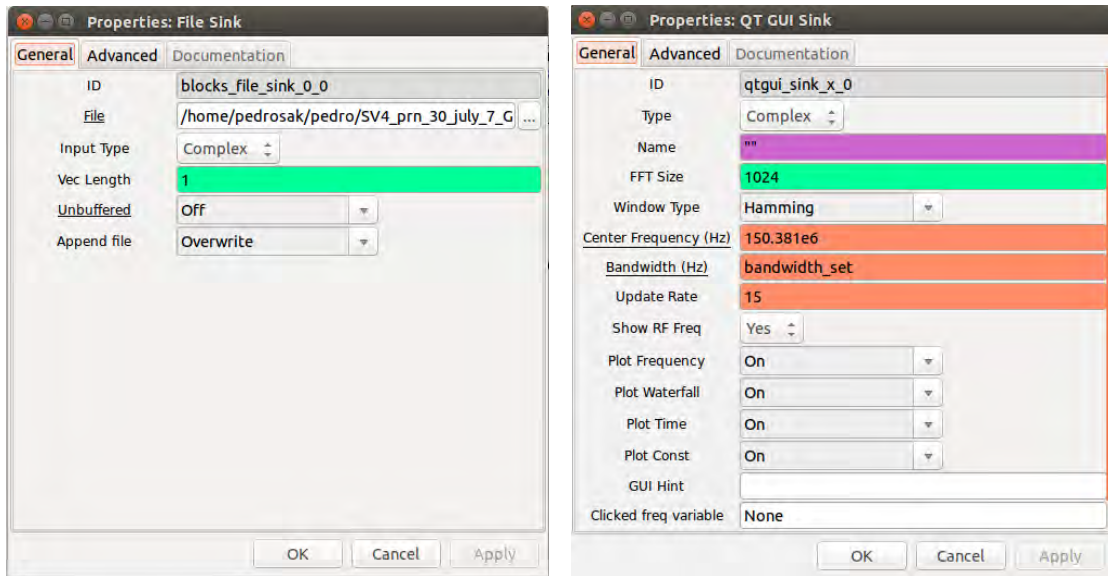
(a) File Sink block general properties   (b) QT GUI Sink block general properties

Figure 3.27: General properties for the File Sink and QT GUI Sink blocks

Running the above flow graph in GRC records any data being received by Ettus N210. Before recording it is a good idea to first check the signal using the different GUI plots provided by the QT GUI sink block. It is important to remember that the flow graph controls the hardware and any limitation on signal processing come from the hardware being used and the connection between the hardware and the target PC. In this project, signal limitations were due to the Ettus N210 daughterboard which provides a signal band between 50 MHz and 2.2 GHz, however as explained in section 3.4.1 the Ettus N210 internal signal synthesizer limits the lower signal band to 68.75 MHz.

The connection between the Ettus N210 and the target PC running GNU Radio used a direct gigabit Ethernet connection. While no loss of data was detected while using GNU Radio there could have been undetected data loss during the file write process. Because all of the recording used for post-processing produced good recordings ensuring no data loss occurred was not needed.

3.4.2.1 Lessons Learned

Errors were encounter when installing UHD. These errors were solved by research-ing forums and asking Ettus lists serves. For sake of completion these solutions are compiled below. Let it be known that this compilation of solutions are in no way conclusive. Other errors may arise with solutions not covered by the ones below.

At times, when installing UHD a warning may appear that states the inability to locate the USRP1 firmware. Running the terminal commands listed in source code 3 will solve this error by downloading a new UHD image. Before execution of the code below the Ettus N210 is to be connected to the target PC and powered on.

```
1    cd ~/Downloads/target/lib
2    sudo mv uhd/ /usr/share/
3    cd /usr/share/uhd/utils
4    sudo ./uhd_images_downloader.py
5    cd ~/Downloads/target/share/uhd
6    sudo mv images/ /usr/share/uhd/
```

Source Code 3: Solution for a warning that occurs during UHD installation

Another error encountered was due to user permissions in Ubuntu. Certain con-figurations done by the UHD installation required specific permissions that need to be manually configured. The terminal commands listed in source code 4 de-fine new user permissions and restarts the target PC. Like the previews solution this solutions required that the Ettus N210 was powered on and connected to the target PC.

```
1        sudo su
2        groupadd usrp
3        usermod -G usrp -a $USER
4        echo 'ACTION=="add", BUS=="use"SYSFS{idVendor}="fffe",
5            SYSFS{idProduct}=="0002", GROUP:="usrp", MODE:="0660"'
6            > tmpfile
7        chown root.root tmpfile
8        mv tmpfile /etc/udev/rules.d/10-usrp.rules
9        reboot
```

Source Code 4: Solution for an error that occurs during UHD installation

The installation procedures and some of the solutions to the errors encountered were found in [30] and [31]. A considerable amount of time learning GRC was spent utilizing the useful tutorials found in [32]. GNU Radio and GRC in particular has a vast amount of features not explored in this project. The GNU Radio framework is a powerful signal processing software that helped further development of this project.

3.4.3   GNSS-SDR Project

The GNSS-SDR project was started by a non-profit research foundation called Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) [33]. It is a free, open-source software under the General Public License V3 [34]. It was built as a GNSS research effort to provide a software receiver that has the capability to target multiple signal bands and multiple GNSS [35]. It was chosen as the GPS software receiver because it allowed many kinds of changes to be made, such as changing the source of the signal (either hardware of file based), and access to all the intermediate signals, parameters and variables [35].

Using the Ettus N210 as the RF front end, the GNSS-SDR software received the signal captured by the RF front end, processed it in accordance to a configura-

tion file, and returned output parameters. The configuration of the software is managed by the Control Plane, a software subsystem charged with creating a flow graph according to the configuration of the processing blocks, while all the signal processing is controlled by the software Signal Processing Blocks. The Signal Processing Plane is built upon the GNU Radio framework which is why it requires a full installation of GNU Radio and its dependencies.

The ROACH FPGA generates the desired GPS signal which is transmitted at the carrier frequency of 50.127 MHz via a physical cable to the Ettus N210. The target PC, which communicates to the Ettus N210 via a gigabit Ethernet port, runs the GNSS-SDR during the transmission process. Over time the GNSS-SDR software processes the received signal providing information about the signal's composition which include the number of SV's acquired, their PRN sequence, and message signal data.

The installation process for the GNSS-SDR is well documented in [36]. It is important to ensure that GNU Radio and its dependencies are installed prior to attempt GNSS-SDR installation. In order to execute the software, ensure proper connection the the RF front-end, in this case the Ettus N210, by the following commands on a terminal window: **uhd_usrp_probe** or **uhd_find_devices**. The first command returns the information about the Ettus N210 and detailed information about the daugtherboard currently installed. The latter returns summarized information about the Ettus N210, however both commands verify proper connection the RF front-end. When proper connection to the RF front-end has been verified and the GNSS-SDR *.conf* configuration file has been configures the following command will execute the GNSS-SDR software and being processing the received signal: **gnss-sdr - -config_file=$LOCATION_OF_CONFIG_FILE$.conf**.

A successful acquisition of the GPS signal will result in a stream of data flow in the terminal window reporting that individual subframes have been received.

An indication of which SV (described by its PRN sequence number) transmitted the received subframe is also indicated. At the termination of the GNSS-SDR software, files will be generated and saved to the directory which the software was invoked from. The position, velocity, and time (PVT) block can be configured to produce a file using The National Marine Electronics Association (NMEA) 0183 format which gives direct access to the PVT data. Two other files, a KML file with geographic data and a Receiver Independent Exchange Format (RINEX) file providing raw satellite navigation data, are produced automatically by the acquisition and tracking blocks. These files are great for post-processing and analysis of the received signal.

3.4.3.1 Lessons Learned

Installation process for the GNSS-SDR can be fragmented depending on the versions of the many packages that can be already installed in the OS. It is important to follow the instruction in [36] regarding installation, and to be familiar with the packages currently installed in the Ubuntu version in use. The configuration file is the interface between the user and the software. Understanding each block configuration, and the process used will give better signal analysis control. A mistake was made during this project by using the GNSS-SDR software as a "black box", only interested on its input and outputs and not diving deeper into the inner workings of the software. More time needs to be devoted to understanding each process, from acquisition to tracking.

Chapter 4

Results

To validate the construction of the GPS signal, a number of tests were conducted on the firmware, software, and the successful acquisition of the GPS signal. The firmware tests were focused on ensuring correct data transfers, timing, and various unit testing. The software tests were mainly done to verify logic and process used. The signal acquisition test was performed using a GPS software decoder and a SDR GPS receiver.

All tests were conducted using the apparatus presented in Section 4.1. It is important to note that no over-the-air transmission was conducted during any of the testing process.

4.1 Hardware Setup

All tests conducted were done with the apparatus listed below and the same configurations on each of the instruments. All of the hardware and instruments used were owned by the Radar & Microwaves Laboratory therefore no other equipment was needed. The hardware setup consisted of two personal computers (PCs), ROACH board, Ettus N210 SDR, signal generator, oscilloscope, and a spectrum analyzer.

Figure 4.1 shows the hardware setup, note that only one PC is shown in the figure. The second PC, use for data and signal acquisition, was setup in a different part of the laboratory.
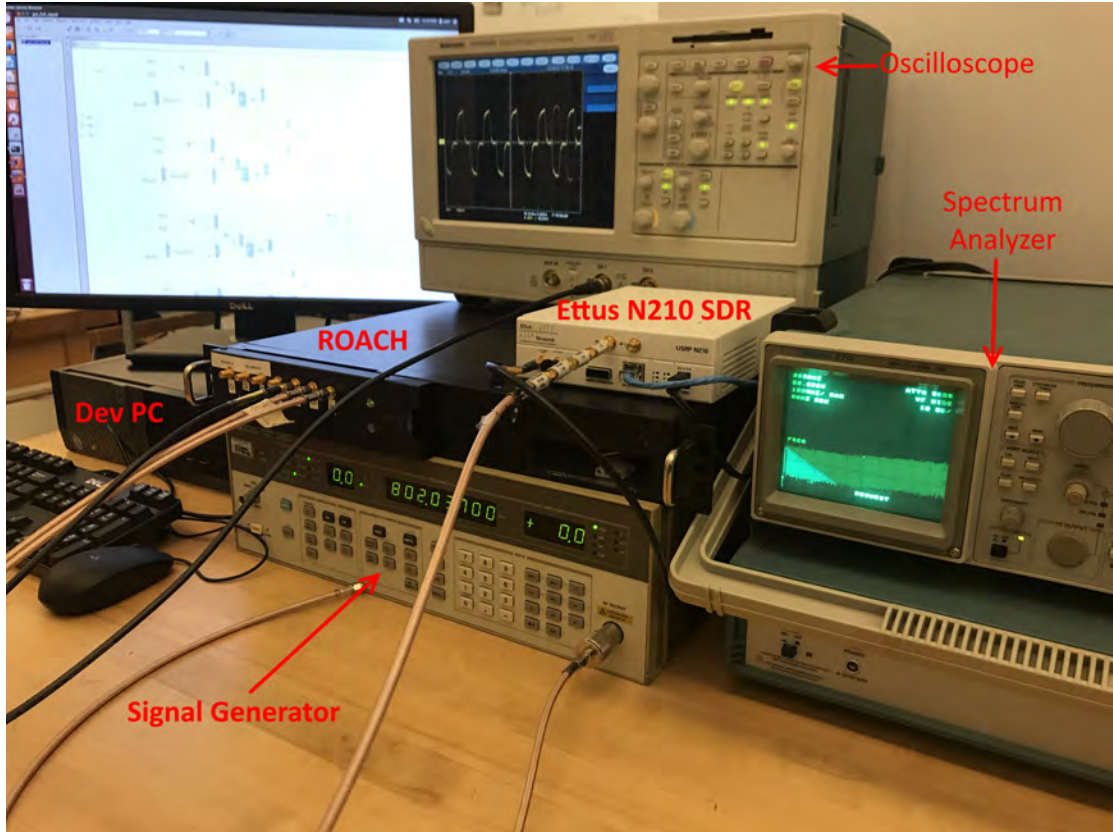

Figure 4.1: Hardware setup

The development PC, labeled *Dev PC* was used to develop the firmware and software deployed in this project. It was connected to a local area network (LAN) where the ROACH board was also connected. The ROACH board had one input, the system clock, and two outputs, I and Q. The system clock input received a clock signal from the signal generator. The I and Q outputs were connected to the oscilloscope, spectrum analyzer, or the Ettus N210 depending on the test being performed. Both ROACH outputs were the same, meaning it transmitted the same signal on both outputs, therefore it was possible to scope the same signal using the oscilloscope and the spectrum analyzer and see the signal in both the time domain

and frequency domain. A better view of the ROACH's input/output ports and
the signal generator is show in figure 4.2. Note the labels on the input/output
ports. The system clock labeled C_o is the clock input to the output board.

Figure 4.2: Signal Generator configurations and ROACH input/output ports.

Figure 4.2 also shows the configuration of the signal generator. The ROACH
system clock was set to a frequency of 802.032 MHz, and an amplitude of 0 dBm.
Changing the frequency would change the system clock and in turn change the
overall characteristics of the transmitted signal, and possibly cause timing errors
in the firmware.

The Ettus N210 SDR was used for data and signal acquisition. One of the
ROACH's outputs, either I or Q, was connected to the the *RF1* input on the Et-
tus. The Ettus was connected directly to the acquisition PC via Ethernet cable.
The time base output provided in the back of the signal generator was connected
to the *REF IN* input on the Ettus N210.

The Ettus N210 external reference clock is used to synchronize the master oscillator and must be a 10 MHz signal with a power level between 0 dBm to 15 dBm. Figure 4.3 shows the time base output on the signal generator, 4.3a, and the front of the Ettus N210, 4.3b, with the *REF IN* port on the bottom left corner.

(a) Back of signal generator showing the time base output connection

(b) Front of the Ettus N210 showing the reference in port

Figure 4.3: Reference Clock connection between the Ettus and the signal generator

4.2 GPS Signal Validation

Prior to testing a full GPS signal, PRN and message data were transmitted and acquired by a GPS receiver. Each individual signal was verified for timing accuracy and correctness of data generation. A switch on the firmware allowed control over transmission of both signals, the PRN signal could be switched off allowing only the transmission of the message signal or vice-versa. This allowed for the independent test of each signal.

4.2.1 PRN Signal Validation

The test validation of the PRN signal attempted to answer two questions. First, was the PRN signal transmitted correctly for all of the possible PRN sequences. Second, when multiple PRN sequences are transmitted at once was the signal still

correct and detectable.

To answer both questions a test was conducted by generating 4 random PRN sequence signals at once. This was done by assigning one random PRN sequence signal to each of the four PRN generators in the firmware. As explained in section 3.1.2, the firmware was designed to allow the transmission of four independent signals. These four signals mimic a signal configuration of four different SVs. Each of these four independent signals generated unique PRN sequences from their PRN generator section in the firmware. The four PRN sequences chosen were 9, 15, 23, and 30 assigned to signal of SV 1, SV 2, SV 3, and SV 4, respectively. Neither order assignment of the PRN sequence nor the chosen SV affected the signal, they were chosen arbitrarily. Any PRN sequence on any SV would produce the same results.

Each signal was independently transmitted starting with SV 1 (PRN 9). The signal was acquired and the data saved as explained in section 3.4. The PRN signal test consisted of correlating the raw signal data with a synthesized PRN signal of the same sequence. A synthesized PRN signal for each of the chosen PRN sequences was created, then correlated against the raw signal data of the same PRN sequence. The expected result of the correlation was a high correlation peak if both the synthesized signal and the raw were equal and low correlation peak if they were not.
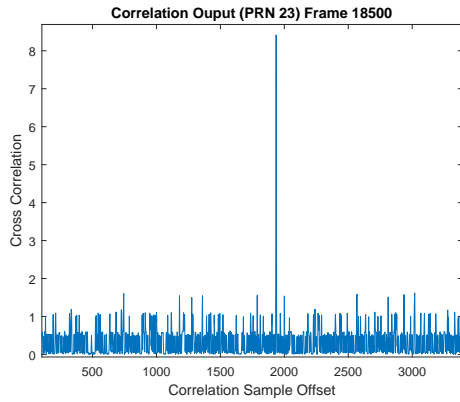
Figure 4.4 shows the correlation results of a single frame, at an arbitrary time in the correlation, for all four PRN signals. The high correlation peak is evidence that a known signal, the synthesized PRN signal, in fact exists within the transmitted PRN signal.
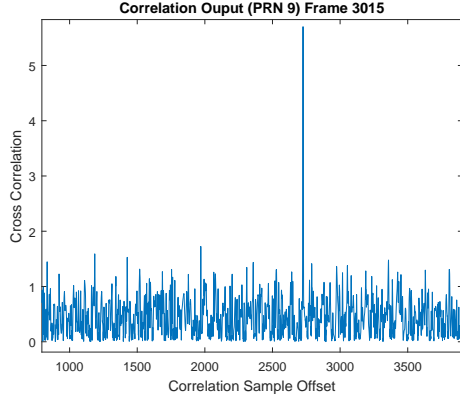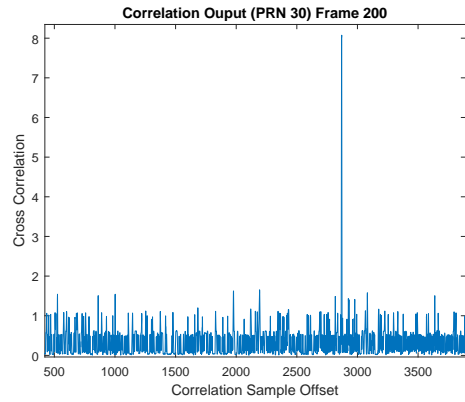
(a) Correlation Output of PRN 9
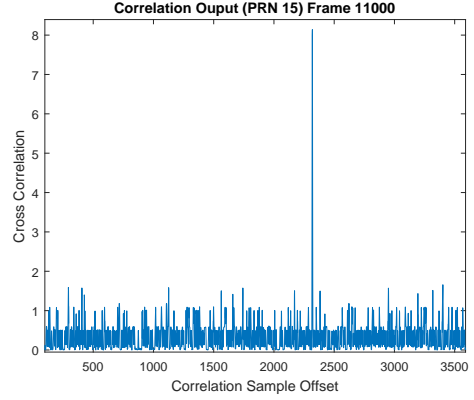
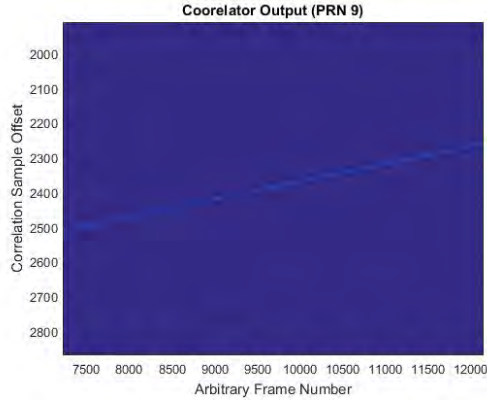(b) Correlation Output of PRN 15

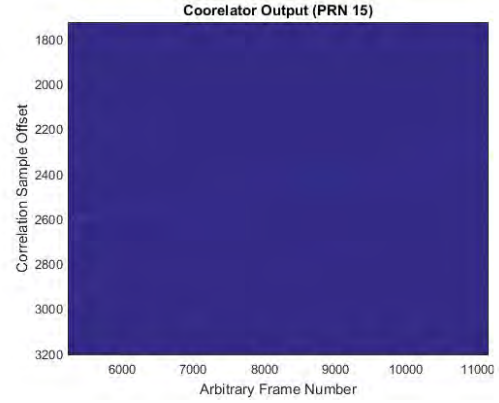(c) Correlation Output of PRN 23

(d) Correlation Output of PRN 30

Figure 4.4: Correlation output (single frame) of all four PRN signals

Because the transmitter and receiver were not phase locked there was some residual drift in the signal. This drift is visible when plotting the correlation over all frames as well as plotting the correlation output of different individual frames. Figure 4.5 shows the drift of the same correlated signals mentioned above. The slightly slanted line is the correlation peak over all frames. Note the position of the correlation line as it shifts due to drift. This result is a clear indication of a successful correlation and evidence of a drift. A correlation output without any drift would show a completely horizontal line indicating that no correlation offset from frame to frame. Section 3.3.2 describes the post-processing performed on the transmitted signal to reduce the drift before correlation was done.
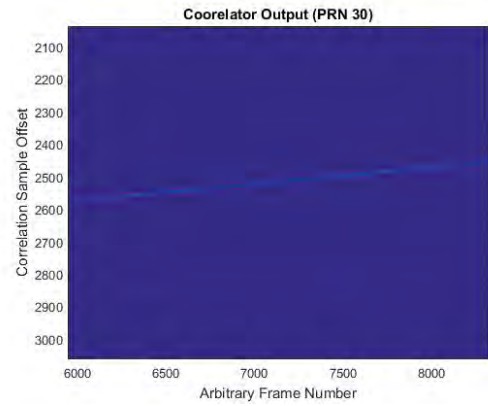
(a) Correlation output of PRN 9

(b) Correlation output of PRN 15
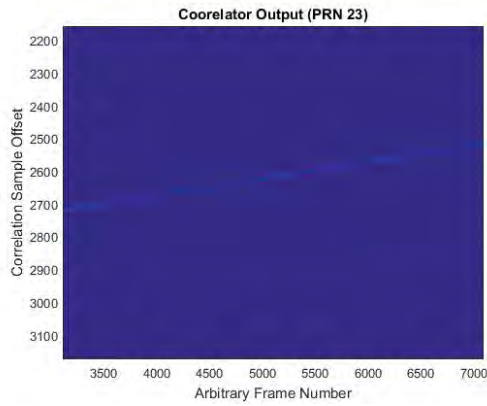
(c) Correlation output of PRN 23

(d) Correlation output of PRN 30

Figure 4.5: Correlation output (all frames) of individually transmitted PRN signals

The above correlation was conducted for all possible PRN sequences. All tests yield the same results certifying proper detection of any desired PRN sequence.

A second test was performed to validate the correlation process. This second test used a transmitted signal composed of all four SVs. Each of the four SVs contained a different PRN sequence. Again, the chosen PRN sequences were the same as the previous test, the only difference was instead of individually transmitting each PRN signal all signals were transmitted at once: the transmitted signal was the sum of all four PRN signals. Much like the previous test a synthesized PRN signal was created but this time the PRN sequence was one known to not be one of the four PRN sequences contained in the transmitted signal. The expected result was

a low correlation between the PRN sequence not present in the transmitted signal.

Figure 4.6 shows the correlation between PRN sequence 28, which is not expected to be in the transmitted signal, and the transmitted signal which is a combination of the PRN signals, 9, 15, 23, and 30. Low correlation is presented in all frames, figure 4.6b, as well as in a single frame, figure 4.6a.

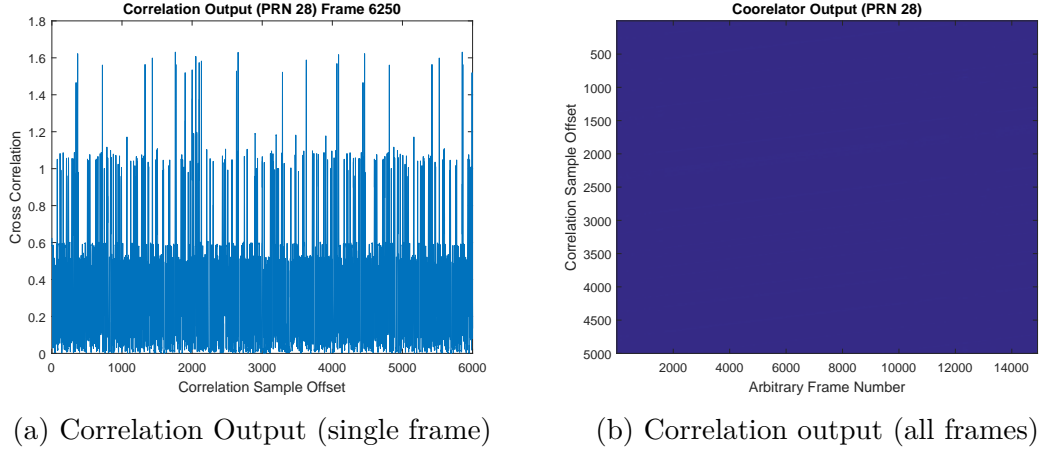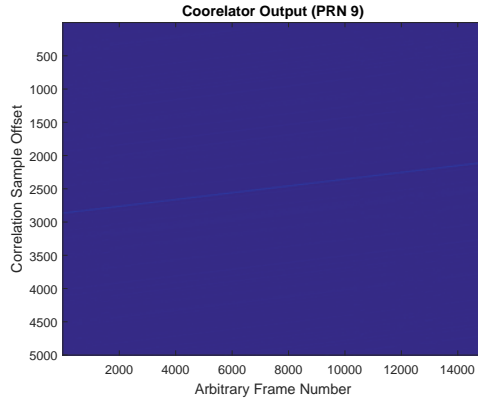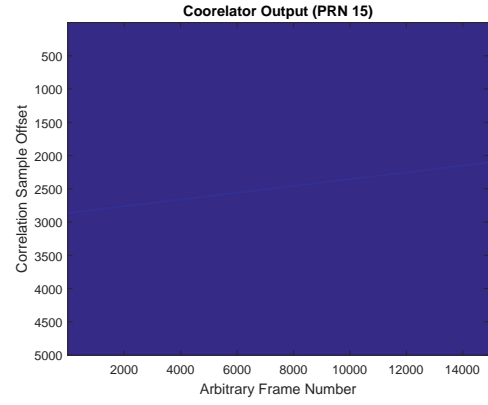(a) Correlation Output (single frame) (b) Correlation output (all frames)

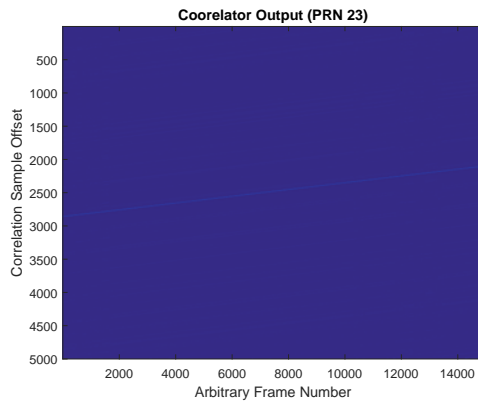Figure 4.6: Low correlation output of PRN 28

Finally, the detection of each individual PRN sequence in the composite transmitted signal was performed. The purpose of this last test was to detect one of the four PRN sequences in the transmitted signal which was composed of all four PRN sequences. As expected, the test resulted in high correlation of all four PRN sequences present in the transmitted signal. The multiple frame correlation output plots are shown in figure 4.7. A single, random frame output correlation is shown in figure 4.8. This test was conducted for all PRN sequences not present in the transmitted signal yielding the same results.
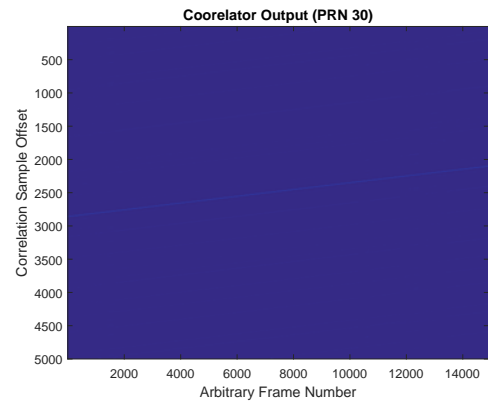
(a) Correlation output of PRN 9

(b) Correlation output of PRN 15

(c) Correlation output of PRN 23

(d) Correlation output of PRN 30

Figure 4.7: Correlation output (all frames) for composed transmitted signal

(a) Correlation Output of PRN 9   (b) Correlation Output of PRN 15

(c) Correlation Output of PRN 23   (d) Correlation Output of PRN 30
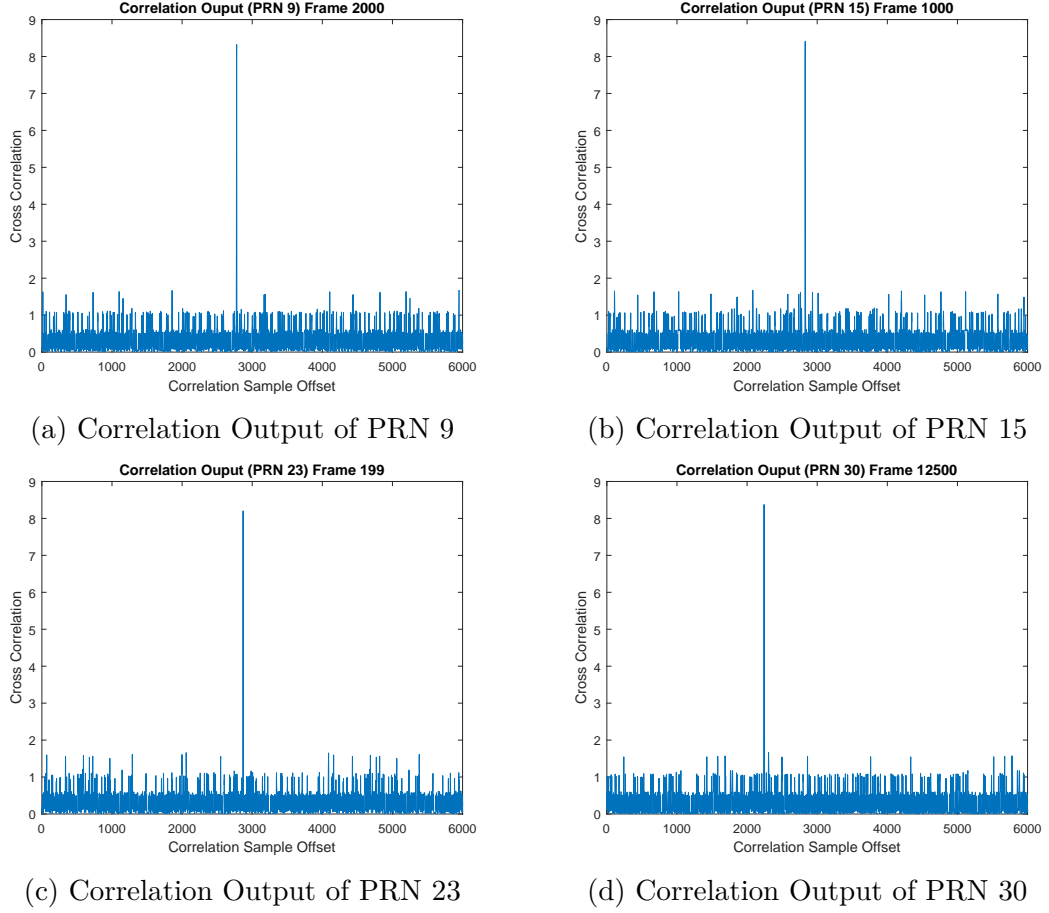
Figure 4.8: Correlation output (single frame) for composit signal

The resulting PRN signal transmitted by the ROACH FPGA is a signal with a wide bandwidth where the nulls are at every +/-1.023 MHz from the center frequency. Figure 4.9 shows the resulting spectrum plot of the four PRN signals transmitted at once. The carrier frequency of the GPS signal was tuned to 50.127 MHz due to the DAC running at 200.508 MHz and the external clock at 802.032 MHz. Measurements done using the spectrum analyzer placed the nulls at the expected positions. A visual inspection of the plot shows the nulls being symmetrically distributed by about 1 MHz separation within each 2 MHz boxes.

Figure 4.9: Power spectrum of PRN signal

4.2.2 Test Vector Data Results

Four test signals were constructed to ensure proper data storage by the internal BRAM. As mentioned in section 3.1.2, BRAM blocks were designed into the firmware to store the data used to modulate the carrier frequency. In the GPS signal, the data stored in the BRAM is the navigation data; ephemeris and almanac data. To test, the BRAMs were loaded with test vector data, each properly constructed to test different possible scenarios. The software, detailed in section 3.2, loaded the BRAM with either the test vector or message data depending on a test flag called *TESTING_IN_PROGRESS*. If the test flag was set to '1' the test vector data would be loaded onto the BRAM.

During testing of the four test vector each vector was loaded onto the BRAM of one of the four SVs. Four data vectors for four SVs. One SV signal was transmitted at a time via a control switch in the firmware allowing for independent testing of each data vector. However, since the overall transmitting signal structure had not changed, the test data was still modulated onto the carrier with a PRN signal. The chosen PRN signal sequence was PRN 9. Since PRN signal testing had already been conducted, choice of PRN sequence was completely arbitrary and any other PRN sequence could have been chosen.

The first SV transmitted a repeating pattern of zeros and ones. This vector pattern tests the BRAM address counter and validates that the format used to write data to the BRAM is correct. Figure 4.10 shows a representation of how the data vector is supposed to be stored in the BRAM. The left column, in blue, shows the address of the BRAM and the right column, in yellow, shows the data stored in the given address. Each address in the BRAM can store 32-bits, however a word is defined to be 30-bits. The test vectors are generated in groups of 30-bits therefore the last two bits, bit 31 and 32, of the BRAM in any given address are terminated in the firmware. In other words, the BRAM can store 32-bits in each address but in this project only the first 30 bits are being used and the last 2 bits are ignored. This is represented by the red zero digits in the diagram. This vector produces 270 ones followed by 330 zeros. This test proves that the counter is sequentially counting through each of the 32 bits in each of the addresses before switching to a new address. Due to the long repetition of this sequence, if the BRAM address counter switched to a new address too early the sequence would be shorted than expected. The recorded test vector is first decoded than the raw bit stream data is normalized using an arbitrary threshold value to produce the figure 4.11. This figure shows the expected sequence of ones and zeros. A count of bits was performed to ensure that in fact there were 270 ones followed by 330

zeros in turn proving that the format used to write data to the BRAM was correct and that address counter was counting properly through each bit before going to the following address.
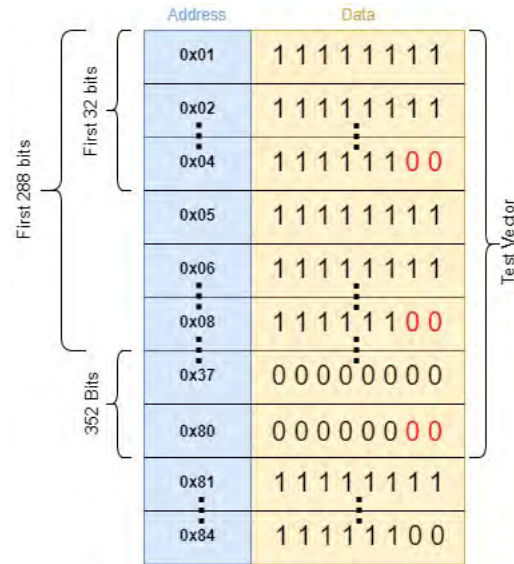

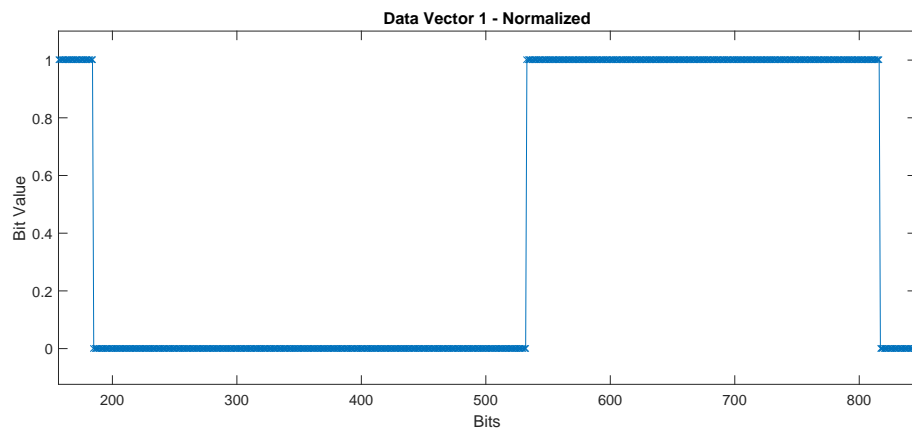Figure 4.10: Diagram representing test vector data 1 stored in BRAM

Figure 4.11: Normalized bit stream test vector 1 data received

The second test vector is a sequence of 30 ones followed by 30 zeros. This attempted to test bit loss, to test bit shifting, and to tune the post-analysis script. The expected results were; first, no bit value would show out of place, meaning that a 1 bit or more would not show between a sequence of zeros. Second, no sequence would be shorter or longer than 30 bits of the same value. Third, the

95

post-analysis script normalized the bits accurately by not assigned a value of one to a raw bit that should have been zero. The third test was more of a validation, it just helped with configuration of the threshold for the bit normalization. By tuning the threshold the script can better determine the normalized value, either one or zero, of every given raw bit sample. This cuts down on how many times the raw bits have to be normalized due to change in threshold. This is important because the vector test results depend on the normalization of the raw data. In-correct assertions of bits will result in faulty data and test failures. An example of incorrect bit normalization is shown in figure 4.12.
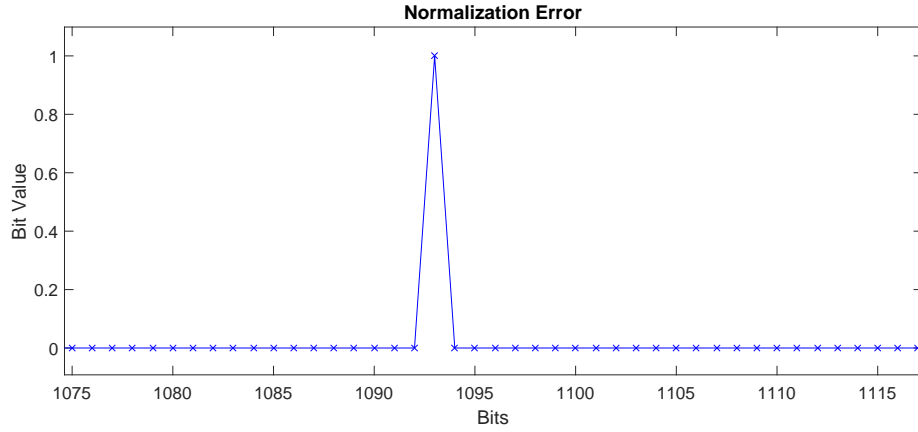

Figure 4.12: Bit normalization error

Figure 4.13 shows a representation of how test vector data was to be stored in the BRAM. The normalized bit stream result in figure 4.14 shows successfully transmission of test vector with the expected sequence of bits.
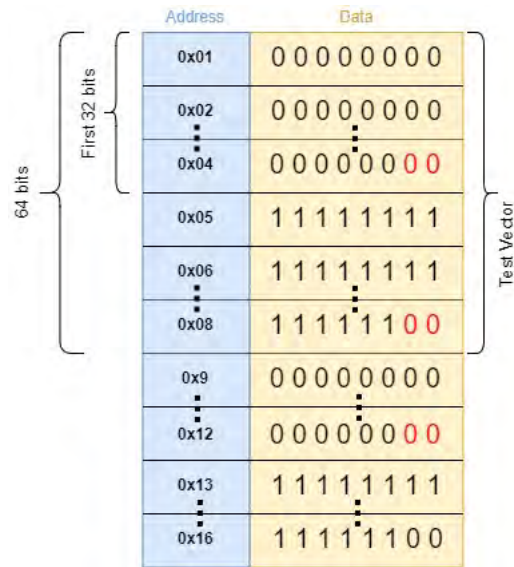
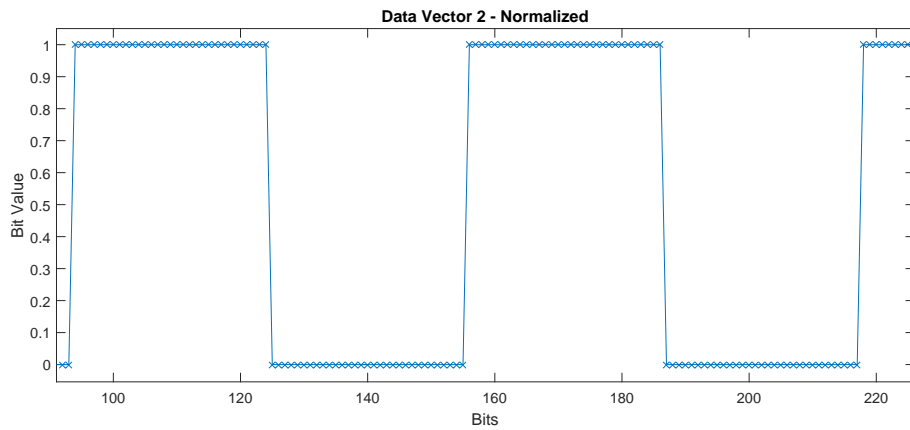Figure 4.13: Diagram representing test vector data 2 stored in BRAM

Figure 4.14: Normalized bit stream test vector 2 data received

Test vector three was better designed sequence to further validate the proper write process used. It tested bit precision within each address in the BRAM. Unlike previous test which tested blocks of address at a time, this test was designed to analyze the 30 bits within each address on the BRAM. This was done by positioning sequences of 3-bits, 1-bit, 9-bits, and 14-bits. Figure 4.15 shows a representation of test vector three data in the BRAM. The first 6-bits are three zero bits followed by three one bits. This identifies the most significant bit (MSB) in the BRAM which should be the first bit to be transmitted. A zero bit follows

this sequence which then proceeded by 9 one bits. The single zero bit tests any drifting in the bit counter, meaning that the bit counter shift by a non-zero value during each iteration. The following 9 one bits and the 14 zero bits to follow test the ending and beginning of each address. If there were any count error these 23 bits would catch it by showing up in the beginning on the next address or by being a different number of ones and zeros at the end of a given address. A second sequence of 30 bits continues the address count test, seeing if any of the 16 one bits or 14 zero bits are shifted into a different address. Overall, test vector three is the most robust test conducted and a passing result, shown in figure 4.16, ensured proper use of the BRAM.

Figure 4.15: Diagram representing test vector data 3 stored in BRAM
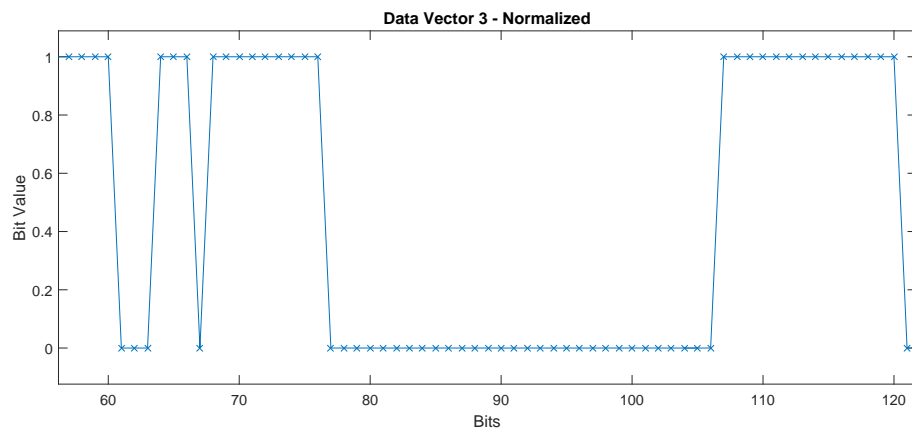

Figure 4.16: Normalized bit stream test vector 3 data received

Lastly, the fourth test vector was a sanity check test to make sure no obvious errors were missed. This test vector left shifted 1 bit each 30 bits. Starting with a sequence of 29 zero bits followed by 1 one bit. The next 30 bits sequence was 28 zero bits followed by 1 one bit followed by 1 zero bit. This repeats until the MSB of 30th sequential address is a one bit followed by 29 zero bits. Figure 4.17 shows a representation of the test vector 4 data in the BRAM. This is the longest test vector composed of 960 bits. Data recording and analysis power was unable to handle a complete test vector data capture. Recording of the full test vector would require processing of a large file (around 4 to 5 GB) and no computer available could process such large file. However, parts of the data vector were analyzed for correctness. Figure 4.18 shows part of the test vector after the raw bit stream was normalized. Note the reverse polarity of the bits cause by improper normalization of the raw bit stream. Regardless of the reverse polarity the bit shift is clearly shown.
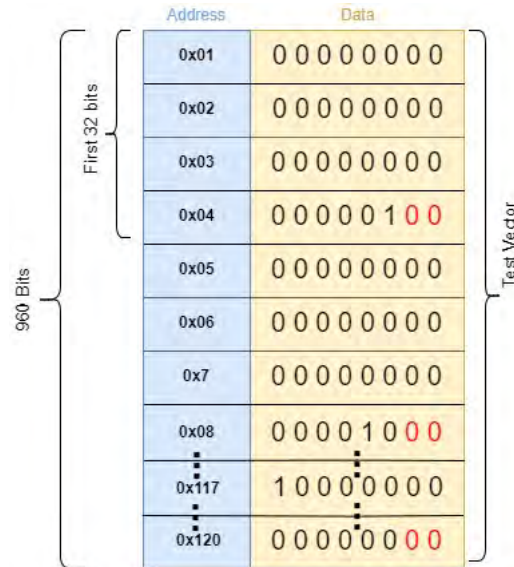

Figure 4.17: Diagram representing test vector data 4 stored in BRAM
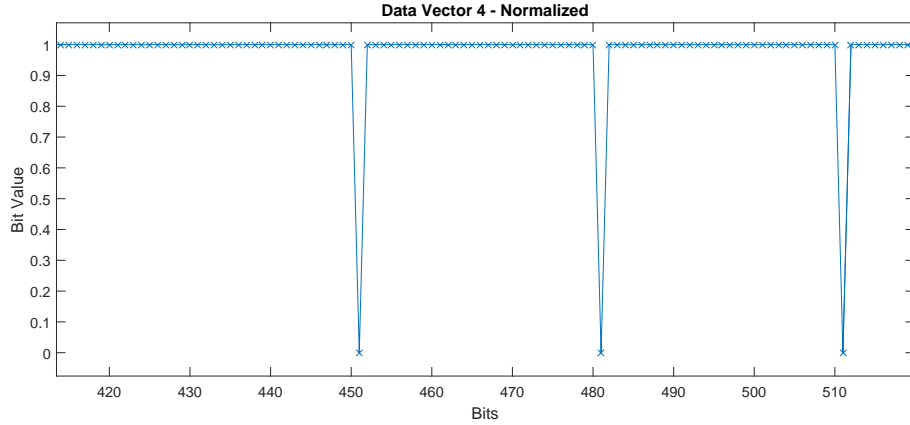
99

Figure 4.18: Normalized bit stream test vector 4 data received

4.2.3 Message Signal Validation

To validate the message signal the initial test was to ensure that the message signal was transmitting at 50 Hz. It is important to note that the validation of the message signal attempted to test only the correctness of the signal, meaning that tests were performed to ensure that the rate at which the data was being transmitted was in fact 50 bits per second. The validation of the data transmitted by the message signal was tested later by the use of a software decoder.

The initial test verified the transmission of a single message signal from one SV. This was possible by controlling a switch on the firmware that allowed turning off the PRN signal and transmitting only the message signal. Another switch in the firmware allowed control over how many SVs were transmitting at once. Once the firmware was configured to transmit only the message signal from one SV the transmitted signal was analyzed using an oscilloscope. The signal transmitted was expected to show a change in bit, which in the time domain of a BPSK signal would consist of a change in phase of the signal, no faster than every 20 ms.

Checking the bit change at 50 Hz was difficult to be analyzed because it is hard to see a 20 ms bit stream modulated on a 50 MHz wave. With the oscilloscope

100

was set to 40 ns unit scale it was possible to catch a bit transition on the screen. The idea behind this test was to visually inspect the scope and see how often a bit change was detected. Detecting more than one bit change in a single slice of 40 ns was an indication that the message signal rate was not correct. Figure 4.19 shows one bit change is detected. It took about 8 seconds to capture that figure, meaning that around 400 bits had already been transmitted if the message signal was at a rate of 50 Hz. Not seeing more than one bit change at a time and taking about 8 seconds to capture a bit change were both good indications of a proper message signal rate. Furthermore, a 10 second visual inspection yielded a constant sinusoidal signal without any bit changes. Part of this visual inspection was capture in figure 4.20.
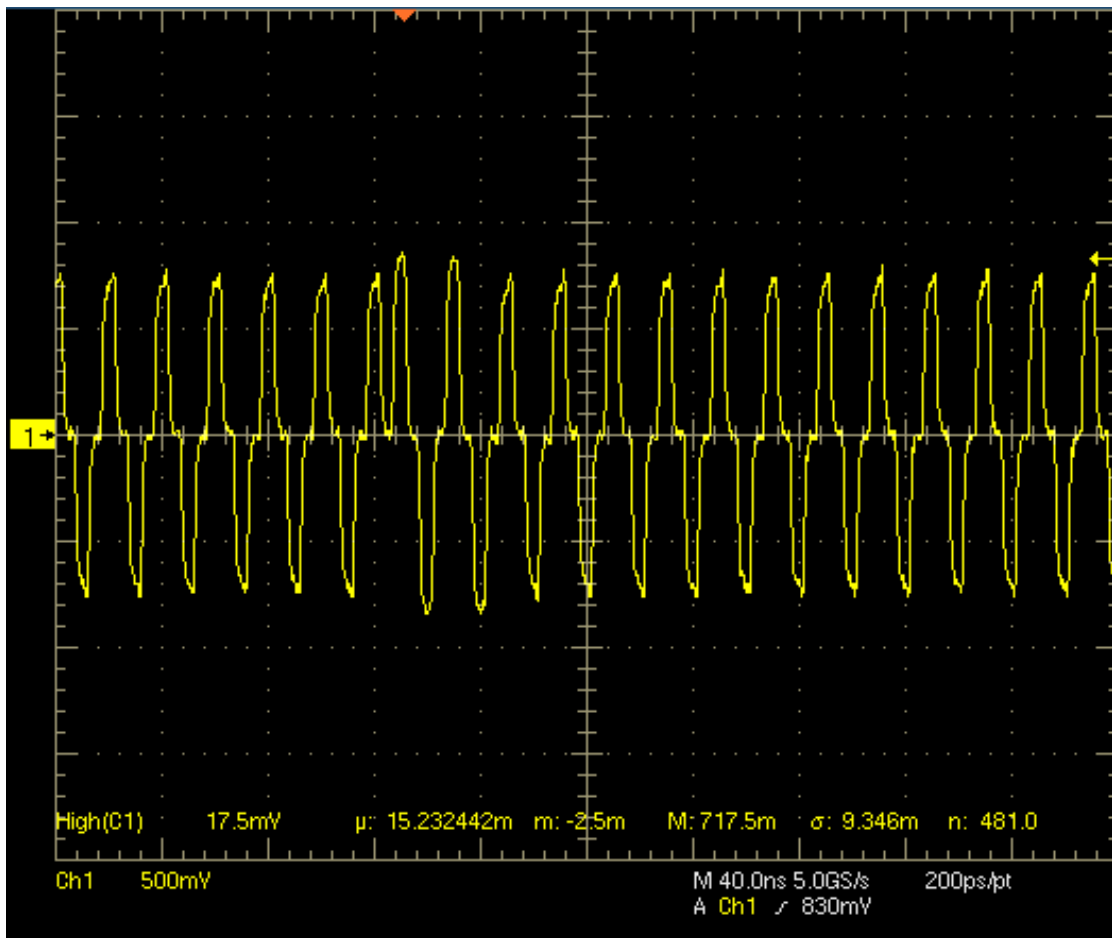
Figure 4.19: Message signal bit change at 40 ns
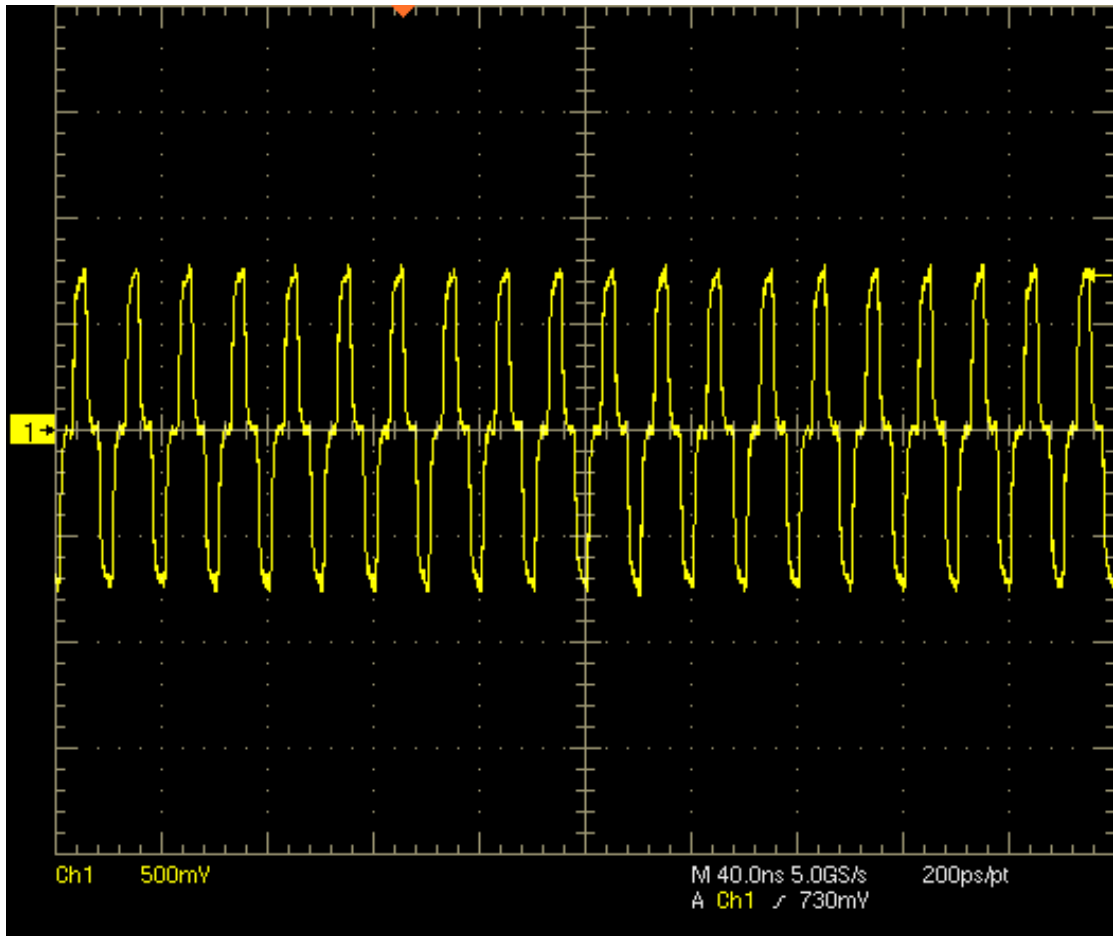
Figure 4.20: Visual inspection of message signal with no bit change at 40 ns

To ensure that bit changes were actually occurring and that the visual inspections resulting in no bit change were not just periods where the data remained the same, the message signal was scoped at 200 μs. Figure 4.21 shows multiple bit changes which was the expected result. Ideally there would have been about 100 bit changes in each of the 200 μs boxes on the scope, but do to limitations on the oscilloscope there was no way to confirm this.
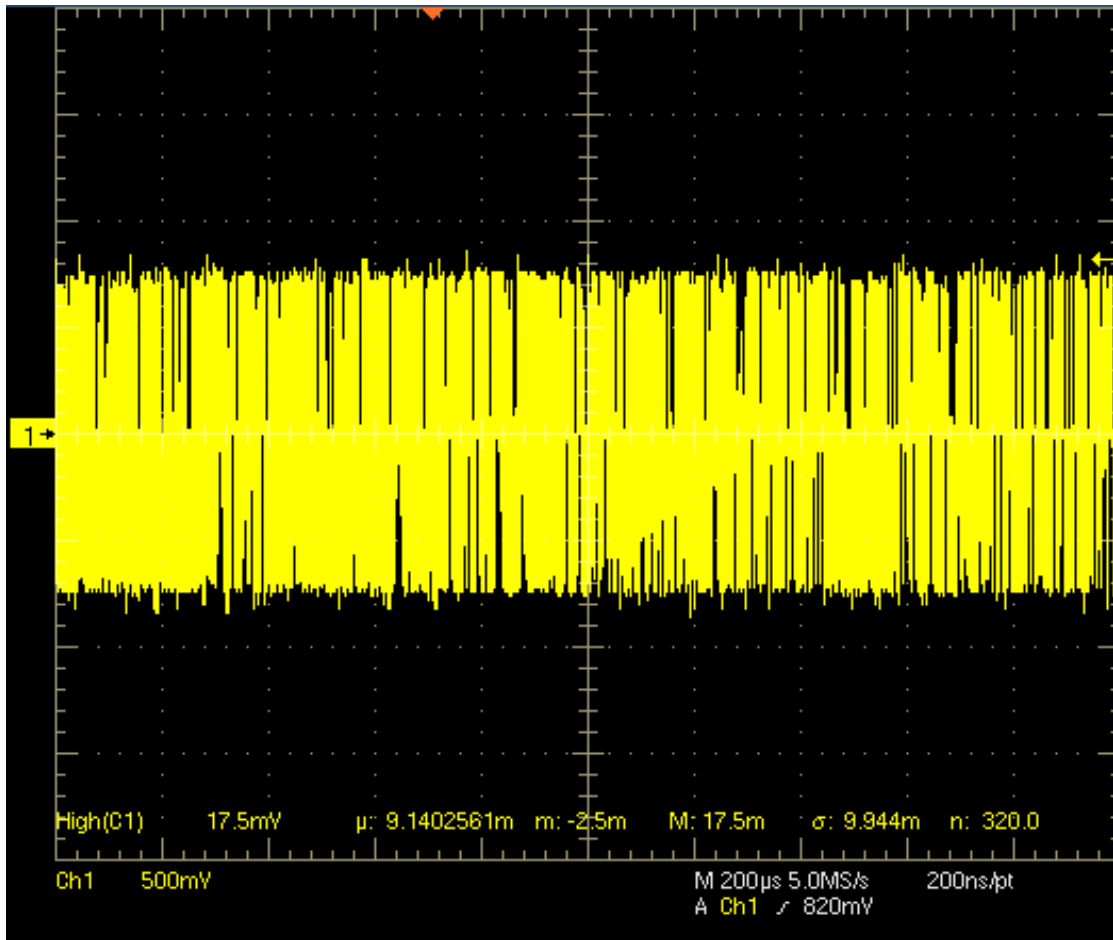
Figure 4.21: Message signal showing multiple bit changes at 200 μs

The message signal was also inspected in the frequency domain to ensure that the 50 Hz message signal modulated with the carrier at 50.127 MHz was in fact centered on the carrier frequency. Figure 4.22 shows the power spectrum of the transmitted message signal with the peak centered at the carrier frequency.
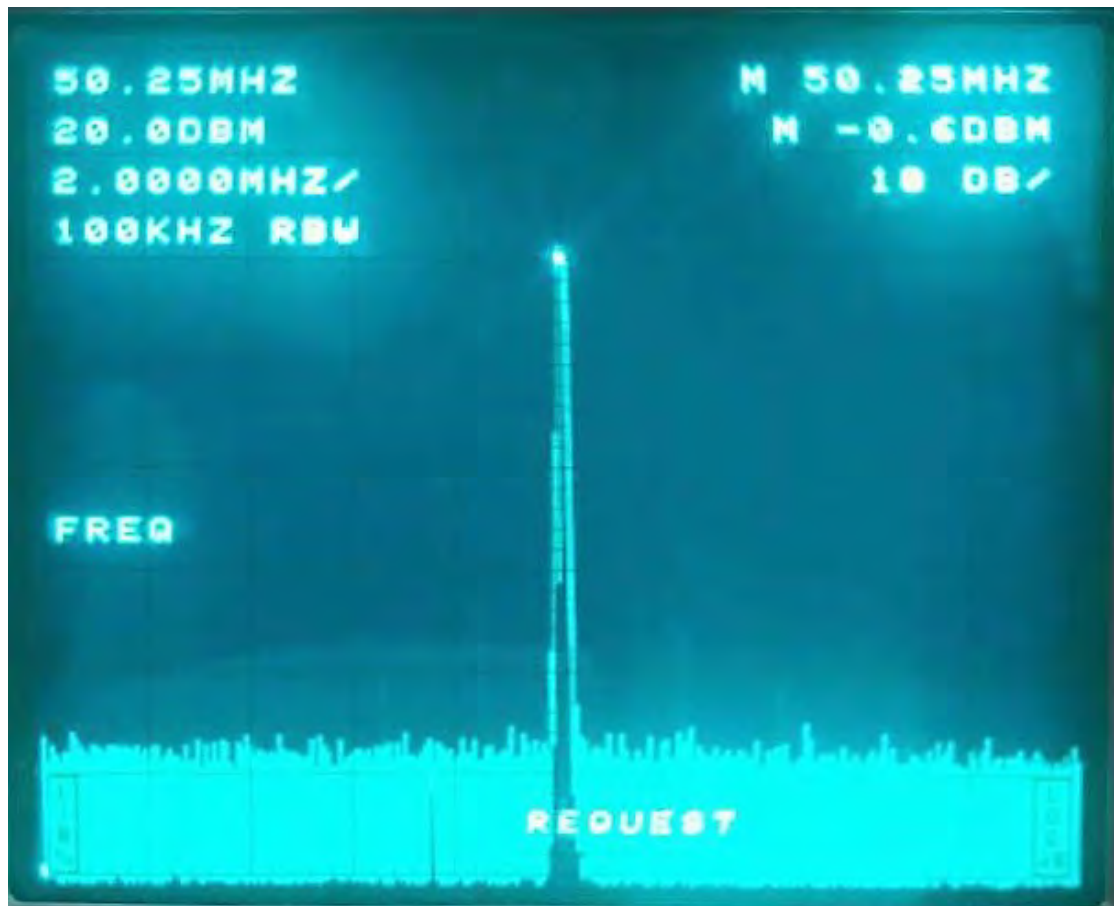
Figure 4.22: Power spectrum of the message signal centered at 50.127 MHz

Progressing from a single message signal in one carrier to four message signals in one carrier was as simple as controlling two switches in the firmware. This allowed for the transmission of four message signals. Each of the four message signals shared a common clock but generated different data. The transmitted signal was a sum of four message signals. This sum would either output zero, half, or full amplitude. This change in amplitude that depends on the sum of each of the message signal bits is shown in figure 4.23.
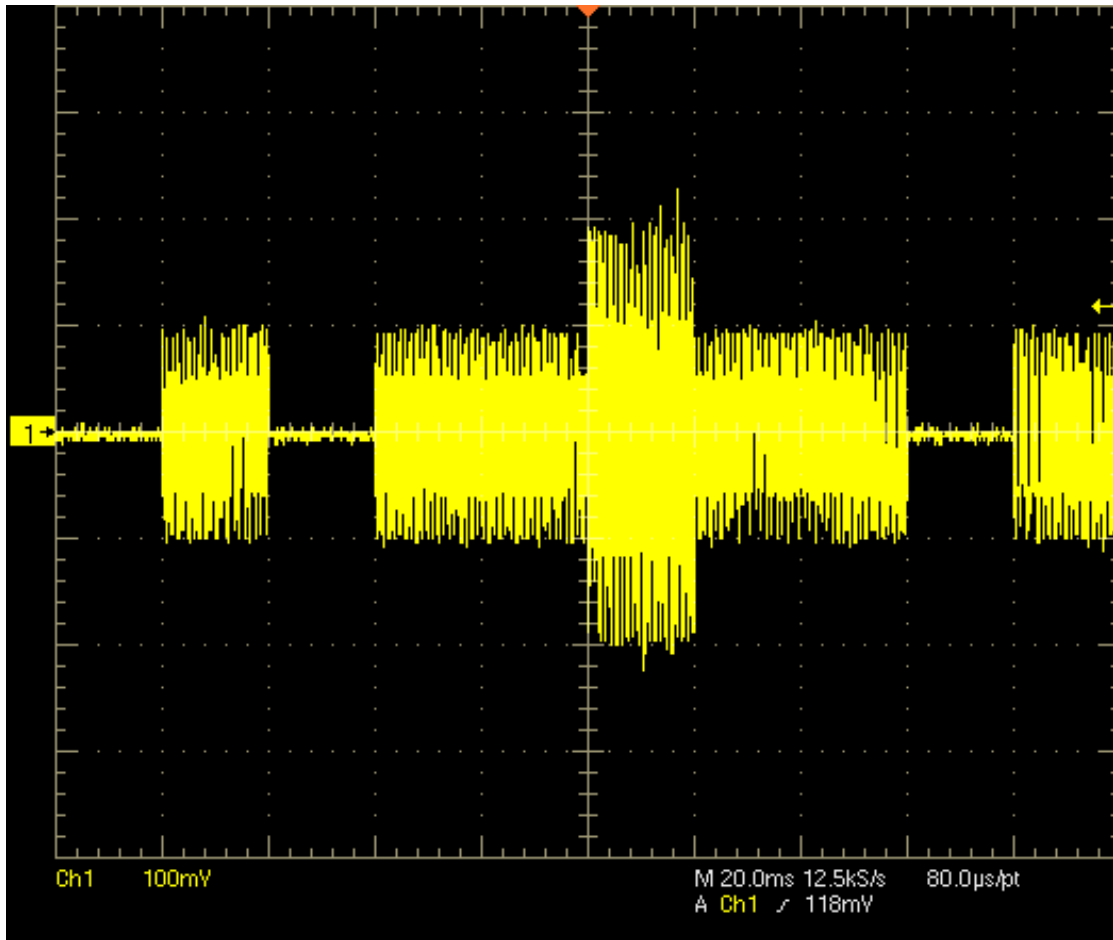
Figure 4.23: Sum of four message signals resulting in amplitude change

The resulting change in amplitude captured by the oscilloscope absolutely shows proper bit time due to the duration of each 20 ms bit streams with equal amplitude. Due to instrument limitations inspecting a signal at such low frequency was not easy. These initial tests using the oscilloscope were done mainly to ensure that no major problems with the construction of the message signals were present. A more complete test was transmitting the message signal to a decoder to ensure the data had been created at the appropriate rate. However, it was important to analyze the message signal both from a single SV and all four SVs to not compound large errors before testing transmission and acquisition of the full GPS signal.

4.2.3.1   Lessons Learned

During testing of the message signal a mistake was found in the message signal clock schema of the firmware. The message signal clock is synchronized with the PRN signal clock which is at a 1.023 MHz. The PRN clock is divided down to 50 Hz through a series of clock dividers. First 1.023 MHz was divided by 1023 producing a 1000 Hz clock, which was further divided by 20 producing the 50 Hz message clock. The process used to detect each clock pulse was done by implementing a rising edge detector. This rising edge detector checked the previous clock count and the current clock count then determined if the current clock count was a rising edge. However, a mistake was made on the logic of the edge detector resulting in an incorrect clock count for the message signal.

The error caused the message generated block to produce all bits stored in the BRAM at the rate of system clock, 200.508 MHz, every 50 Hz. Figure 4.24 shows the burst of data being transmitted every 20 ms. Each box on the scope screen is 10 ms. The reoccurring peaks present every 20 ms are the burst of data being dumped out of the BRAM.
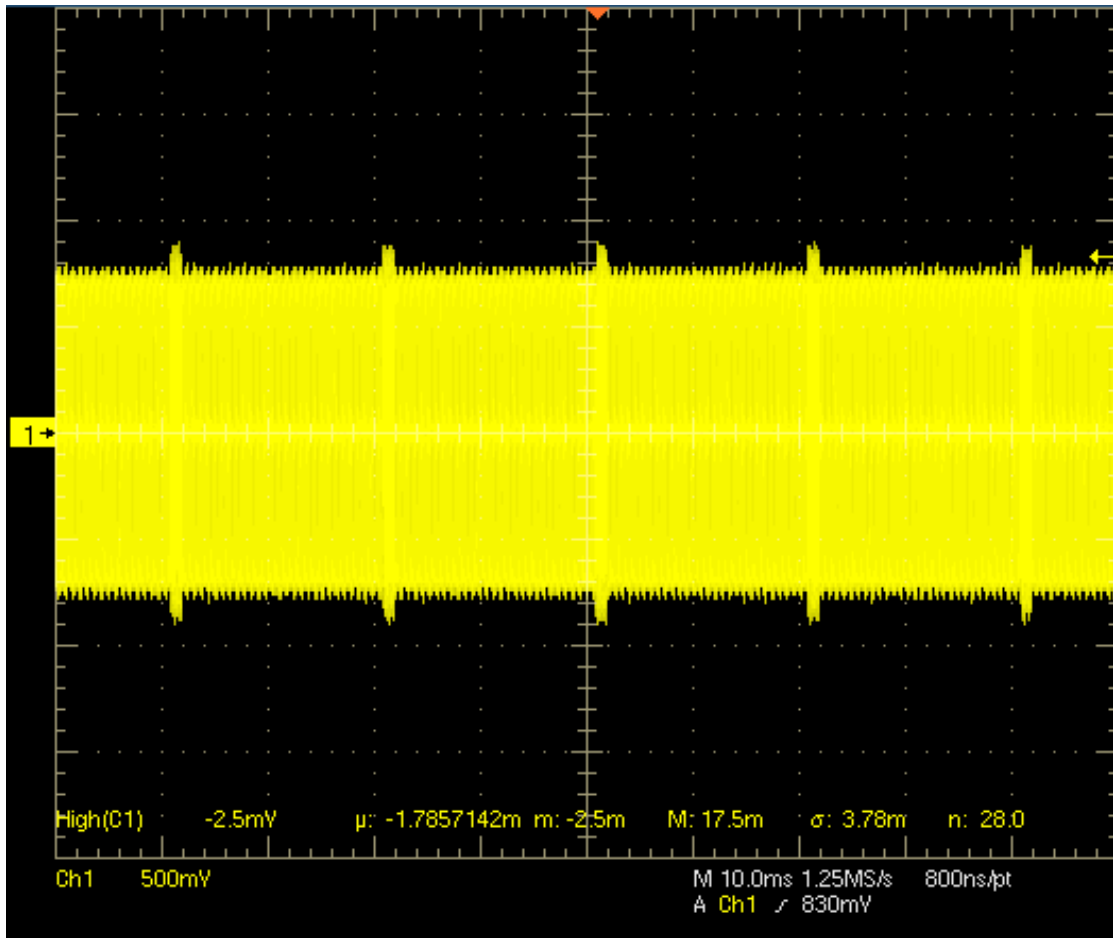
Figure 4.24: Message clock error showing burst of data every 10 ms

Further analysis showed the rapid bit change happening every 50 Hz. Figure 4.25a shows the rapid bit change at 80 ns. Note no bit changes occur prior to the center line of the graph followed by multiple changes in the signal phase. This rapid change pointed to a message clock error which was keeping the clock from transmitting at 50 bits per second. Figure 4.25b shows the same rapid bit change at 20 μs, further indication of the message clock error.
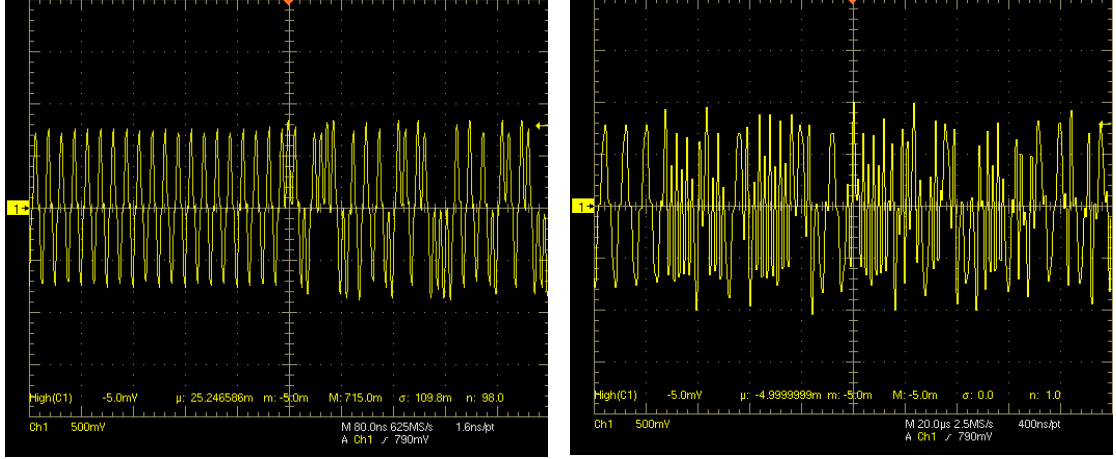
(a) Message clock error at 80 ns   (b) Message clock error at 20 μs

Figure 4.25: Message clock error showing rapid change of bits

The early discovery of this error assisted on the testing of the message signal. Analyzing the message signal at 20 ms without amplitude change was not feasible due to difficulties in analyzing a 20 ms bit rate in a 50 Hz signal. Removing the clock error then checking for rapid bit changes was the best way to test for proper clock rate of 50 Hz. Fixing the error then looking for the absence of the error helped not only correct the message signal but served as a way to further test it.

4.3 Ettus SDR Results

A desired outcome of this thesis project was to receive the generated GPS signal using an off-the-shelf GPS receiver. Instead, a SDR GPS receiver was chosen to provide the control needed not available with an off-the-shelf option. Previous experience with the Ettus N210 made the use of it in this project a natural choice. Connectivity between the Ettus N210 and the GNURadio software had already been established, all parts of the system had been tested, and it was ready to be used. Furthermore, the use of GNSS-SDR software, a software built on top of the GNURadio framework, was installed and utilized to receive the generated GPS

signals.

A diagram shown in figure 4.26 describes the connection set up between the
ROACH and the Ettus N210. The ROACH generated signal was transmitted
via a cable to the *RF 1* input port of the Ettus N210 after going through 45 dB
of attenuation. The attenuation provided a power deduction for better simulation
of a GPS signal. It was a concerned that the GNSS-SDR software would reject
any signal above an expected power level therefore the attenuation helped reduced
the power of incoming signal. The Ettus N210 communicated with a target PC
running the GNSS-SDR software. The target PC controlled the parameters of the
Ettus N210 in order to receive the signal.
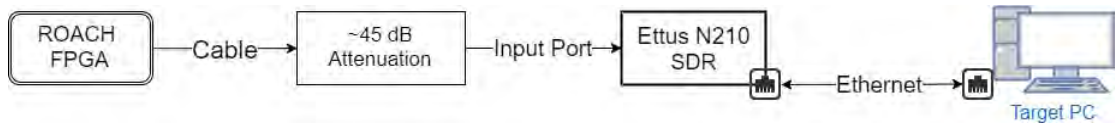
Figure 4.26: Connection between the ROACH and the Ettus N210

In order to acquire the GPS signal a GNSS-SDR configuration file had to be modi-
fied to the specifics of the desired signal to acquire. GNSS-SDR installs with many
different configuration files. These configuration files target many different GNSS
and hardware configurations. The *gnss-sdr_GPS_L1_USRP_realtime.conf* con-
figuration file was modified so it could be used by the Ettus N210. Code 5 shows
the changes made to the original file in order to be used by the Ettus N210. Lines
1-10 defined the hardware configuration parameters including the Ettus N210 IP
address, the clock source it uses, and what frequency to tune it to. The subdevice
parameter, line 9, specifies the *RF 1* input port on the Ettus N210. The samples
parameter, line 10, is set to zero meaning infinite samples. Lines 12-15 define the
channel parameters. Line 13 assigns 8 channels for signal acquisition but line 14
allows only one of those 8 channels to acquire a signal at a time.

```
1      ;######### SIGNAL_SOURCE CONFIG ############
2      SignalSource.implementation=UHD_Signal_Source
3      SignalSource.device_address=192.168.10.2
4      SignalSource.clock_source=external
5      SignalSource.item_type=gr_complex
6      SignalSource.sampling_frequency=2000000
7      SignalSource.freq=150380000.005776
8      SignalSource.gain=0
9      SignalSource.subdevice=A:0
10     SignalSource.samples=0
11
12     ;######### CHANNELS GLOBAL CONFIG ############
13     Channels_1C.count=8
14     Channels_1C.in_acquisition=1
```

Source Code 5: Changes made to GNSS-SDR configuration file

The configuration defines a vast amount of parameters. For a detailed explanation of the above parameters as well as all of the configuration parameters, and a step-by-step guide on how to set up GNSS-SDR to receive C/A L1 GPS signal refer to [36]. The main test conducted using GNSS-SDR was to acquire the generated GPS signal therefor parameter configurations for tracking, and PVT (user position, velocity, and time) were unchanged.

Initial acquisition results failed during testing of a single SV full GPS signal (PRN and message signals modulated with the carrier) being transmitted to the Ettus N210. The results viewed on a terminal window on the target PC are shown in figure 4.27. The GNSS-SDR would identify that a signal was being received but no identification of the signal was made. The time of acquisition did not matter, the GNSS-SDR never acquired the signal when only one SV was being transmitted.

Figure 4.27: Failed acquisition results from GNSS-SDR software

Failed acquisition results do not indicate error with the received GPS signal, is just shows that the current signal being received has no characteristics of GPS signal detectable by the GNSS-SDR software in its current configurations. For example, if the Ettus N210 was currently received an authentic GLONASS signal but configured to receive a GPS signal instead the GNSS-SDR software would yield the same result.

By modifying the ROACH firmware to include four SVs with a full GPS message the acquisition test was conducted again using the same parameters. This time the GNSS-SDR was able to successfully detect a GPS signal. As shown in figure 4.28, GNSS-SDR correctly identified subframes 3 and 5 from PRN 15, 1, 6, and 4. However, as it continued to acquire the signal it began to identify subframes from random PRN sequences not being transmitted at that time. Although unable to correctly acquire the correct subframes the test results proved that a GPS signal was being transmitted. Either some characteristics of the GPS signal were incorrectly generated or the GNSS-SDR software was improperly configured hindering the correct identification of subframes.

```
Current input signal time = 146 [s]
Current input signal time = 147 [s]
NAV Message: received subframe 3 from satellite GPS PRN 15 (Block IIR-M)
NAV Message: received subframe 3 from satellite GPS PRN 01 (Block IIF)
NAV Message: received subframe 3 from satellite GPS PRN 06 (Block IIF)
NAV Message: received subframe 3 from satellite GPS PRN 04 (Block Unknown)
Current input signal time = 148 [s]
Current input signal time = 149 [s]
Current input signal time = 150 [s]
Current input signal time = 151 [s]
Current input signal time = 152 [s]
Current input signal time = 153 [s]
Current input signal time = 154 [s]
Current input signal time = 155 [s]
Current input signal time = 156 [s]
Current input signal time = 157 [s]
Current input signal time = 158 [s]
Current input signal time = 159 [s]
Current input signal time = 160 [s]
Current input signal time = 161 [s]
Current input signal time = 162 [s]
Current input signal time = 163 [s]
Current input signal time = 164 [s]
Current input signal time = 165 [s]
Current input signal time = 166 [s]
Current input signal time = 167 [s]
Current input signal time = 168 [s]
Current input signal time = 169 [s]
Current input signal time = 170 [s]
Current input signal time = 171 [s]
Current input signal time = 172 [s]
Current input signal time = 173 [s]
Current input signal time = 174 [s]
NAV Message: received subframe 5 from satellite GPS PRN 15 (Block IIR-M)
NAV Message: received subframe 5 from satellite GPS PRN 06 (Block IIF)
NAV Message: received subframe 5 from satellite GPS PRN 01 (Block IIF)
NAV Message: received subframe 5 from satellite GPS PRN 04 (Block Unknown)
Current input signal time = 175 [s]
Current input signal time = 176 [s]
```

Figure 4.28: Partially successful acquisition results from GNSS-SDR software

Despite partial test failures, mainly the inability to correctly identify the PRN signals and extract the message data, the use of GNSS-SDR helped ensure confidence on the integrity of the generated GPS signal. The signal generated only included the PRN signal and message signal and lacked the proper Doppler delays as well as the G2 register delays. Even though the message data transmitted was authentic almanac data it is believed that the GNSS-SDR software required all parts for a typical, "real-world", GPS signal. Any GPS receiver calculates the user position by using ephemeris data, contained in the message signal, and time elapsed since transmission. The generated signal transmission time changed subframe to subframe, however with no delay in the received signal the GNSS-SDR

112

software will not be able to calculate position.

Further time needed to be devoted to fully understanding the GNSS-SDR software. Gathering more knowledge of the acquisition process used by the software would assist in altering the generated signal to be successfully acquired by the software.

4.3.1 Lessons Learned

A considerable amount of time was spent attempting to acquire the generated GPS signal. At the time, the generated GPS signal consisted of only one SV signal transmitting both the PRN and an incomplete message signal. It later evolved to a true message signal with a tested PRN signal but no acquisition by the Ettus N210 was made.

After reading through the GNSS-SDR online documentation it was discovered that the software only acquires when at least four SV signals are detected, meaning four different PRN sequences. It was also discovered that, besides needing four SV signals, it also needed to receive at least the first three subframes of the message signal.

Once the ROACH firmware was updated to include four different SVs with complete message signals, five frames and 25 pages, the Ettus N210 and the GNSS-SDR software started yielding results.

Another observation made was with the input signal attenuation. The ROACH signal was fed into the Ettus N210 after going through 45 dB of attenuation. After increasing attenuation to 130 dB the GNSS-SDR software was no longer able to detect any incoming signal. However, decreasing the attenuation from 130 dB down to 0 dB did not change how the GNSS-SDR software acquired the signal. It was concluded that the attenuation at the input of the Ettus N210 did not impact

the acquisition of the transmitted signal.

4.4   Decoder Results

A software decoder was developed as an alternative to signal acquisition by a GPS receiver. The results presented in this section are decoded data from a composite signal demodulated to recover the carrier signal transmitting by the SV using PRN 30 spreading sequence.

The message data, as defined by the NAVSTAR specifications [11], is a stream of bits with specific sequences. A stream of 300 bits contains data from one of five possible subframes. The decoder identifies each subframe and parses it as defined by the NAVSTAR specifications and in the method described in section 3.3.2. The subframes represented by the results presented in this section are indicative of the success of the decoder and not a collection of all possible results attainable by the it.

A stream of roughly 3000 bits was analyzed by the decoder. In these 3000 bits all 5 subframes were sequentially (every 300 bits) identified and parsed. The TLM word is kept out of the parsed results because it is repeated over every subframe. The contents of the TLM word contain the preamble ('10001011'), telemetry message (a sequence of 14 zeros), a reserved bit (always of value '0'), a status flag (always of value '0'), and parity bits. Although parity bits were parsed they were never used to check the validity of the parsed message.

All of the almanac parameters contain a scale factor. These scale factors, specific to each parameter, scales the value to a specific range such that it can be represented by a given number of bits. The scale factors are listed on table 4.1 along with

number of bits for each almanac parameter. All decoded almanac parameters must be multiplied by the proper scale factor to compute the true almanac parameter value.

Parameter	No. of bits	Scale Factor
e	16	2^{-21}
t_{oa}	8	2^{12}
δ_i	16	2^{-19}
$\dot{\Omega}$	16	2^{-38}
$\sqrt{A}$	24	2^{-11}
Ω_0	24	2^{-23}
ω	24	2^{-23}
M_0	24	2^{-23}
a_{f0}	11	2^{-20}
a_{f1}	11	2^{-38}

Table 4.1: Almanac parameters with scale factors and number of bits

Subframe 4, page 16, was the first subframe and page identified by the decoder within the stream of 3000 bits. The parsed data are shown on figure 4.29 in the same format used by the NAVSTAR specifications shown in figure 4.30. The decoder determined this subframe to be subframe 4 by looking at bits 50 through 52. The page number was identified by the page ID, a 6-bit number located starting at bit 63. In subframe 4 the page ID is defined value (see Table 20-V in [19]) that depends on the page number. For page 16 the defined value is 57. All of the reserved bits were sequences of alternating '1s' and '0s'. All subframe 4 data decoded produced similar results (with expected changes in page and subframe ID).
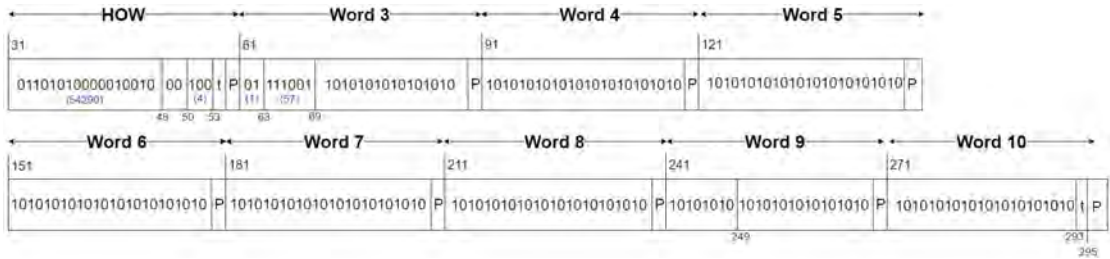

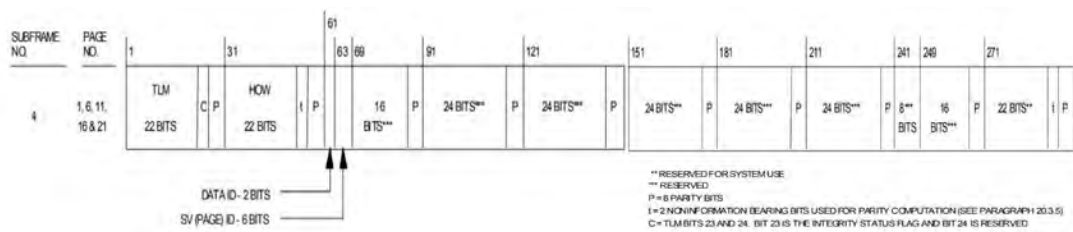
Figure 4.29: Subframe 4 decoded data

Figure 4.30: Subframe 4 data format as defined by NAVSTAR specifications

Following subframe 4, the decoder identified subframe 5, page 16. Figure 4.31 shows the parsed date for subframe 5, page 16. Figure 4.32 shows the NAVSTAR format for subframe 5, pages 1-24. The decoded TOW value (bits 31 through 47) of 54291 was expected given that the previews TOW value (from subframe 4) was one less than this value. This means that the previews subframe happened exactly 6 seconds before subframe 5 (each count of the TWO equals to 6 seconds), which follows the expected timing of the subframe transmission. The decoder checked the subframe ID bits (bits 50 through 52) to determine how to parse this bit. Given that the value of the subframe ID was found to be 5 the data was parsed as such subframe. Validation of the correct identification of the subframe was done by analyzing the data present in words 3 thought 10 and ensuring they coincided with the NAVSTAR specification and almanac data.

The data ID (bits 61 and 62) value of '1' is the correct values for all pages in subframe 5. The SV ID (bits 63 through 68) of 16 means that the almanac data presented within this subframe belongs to SV 16. In order to check the almanac data for SV 16 subframe 1 must be parsed because it contains the week number, which is a number of weeks since the GPS epoch (night of January 5, 1980/morning of January 6, 1980). Using the week number and the time of applicability t_{oa}, contained in subframe 5, the almanac used for this data set could be identified. A visual inspection of the bit placements suggest proper parsing of subframe 5. Further analysis of the bit values was done after parsing of subframe 1.

The t_{oa} had to be scaled so the proper value could be determined. This was done by using the decimal equivalent of t_{oa} (shown in blue) and multiplying by its scale factor shown in table 4.1. The scaled value of $t_{oa} = 99 \cdot 2^{12} = 405504$ was the true parameter transmitted.
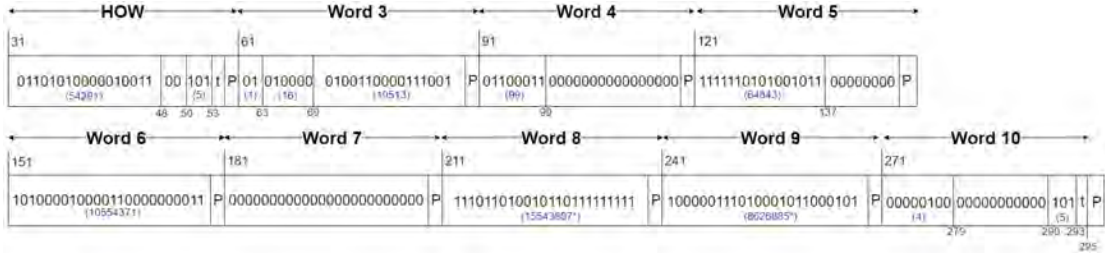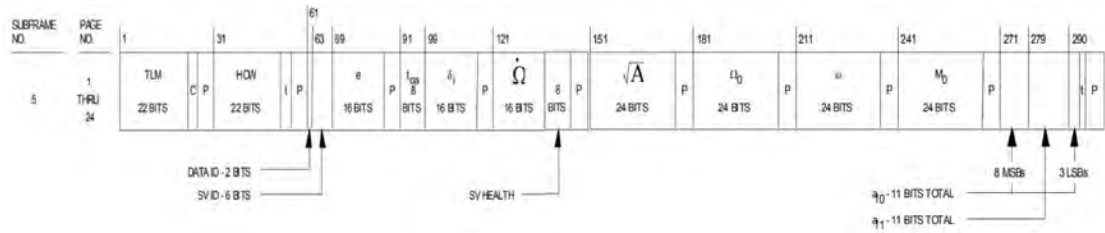

Figure 4.31: Subframe 5 decoded data

Figure 4.32: Subframe 5 data format as defined by NAVSTAR specifications

Further parsing of the bit stream identified subframe 1 succeeding subframe 5, as expected. Figure 4.33 shows the parsed data for subframe 1. Figure 4.34 shows the NAVSTAR format for subframe 1. Right away reserve bits (in word 4, 5, 6 and 7) can be visually identified by the sequence of alternating '1s' and '0s'. The important data parsed from subframe 1 are the TWO (bits 31 through 47) and the week number (bits 61 through 70). As expected, the TOW value is one count more than the previews TOW indicating that 6 seconds had passed since the last subframe was transmitted. The week number 932 indicate how many weeks had passed since the GPS epoch and is the second data needed to identify the almanac used for this data sat. Notice the red bits in word 9 and 10. During inspection of proper bit values it was found that the values used for a_{f0}, a_{f1} and a_{f2} were incorrect. Instead of generating those data using almanac data they should have

117

been calculated by the given equations in paragraph 20.3.3.3.1 of [19]. Because these values were clock correction terms it was not believed to have caused any test failures.
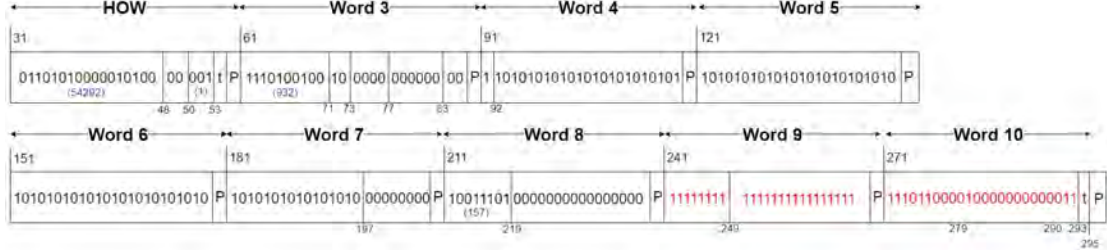
Figure 4.33: Subframe 1 decoded data
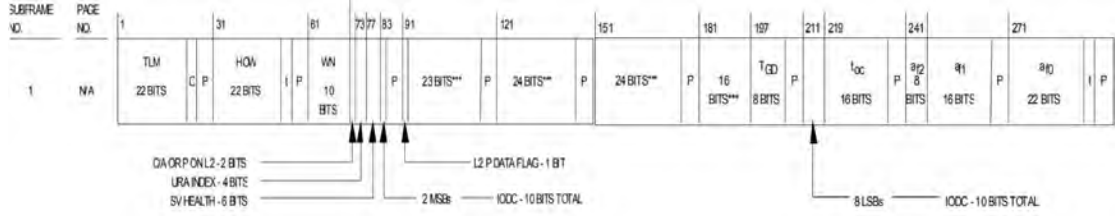
Figure 4.34: Subframe 1 data format as defined by NAVSTAR specifications

Given the week number 932, the almanac must have been generated during the week of July 2, 2017. That week was the second 932 week number since GPS epoch with the first 932 week number occurring the week of November 19, 1997. Tracing the almanac back to the database and cross checking all the generated almanac for the week number 932 the one generated during $t_{oa} = 405504$ was the almanac used for this data set. Table 4.2 shows almanac data for SV 16; the same data decoded from subframe 5.

Parameter	Value	Units
ID	16	
Health	000	
Eccentricity (e)	0.9305000305E-002	
Time of Application (t_{oa})	405504.0000	(s)
Orbital Inclination (i)	0.9893660760	(rad)
Rate of Right Ascension ($\dot{\Omega}$)	0.7931758961E-008	(rad/s)
$\sqrt{A}$	5153.677246	(m 1/2)
Right Ascension at Week (Ω_0)	0.1116336917E+001	(rad)
Argument of Perigee (ω)	0.461919965	(rad)
Mean Anomaly (M_0)	0.3052356242E+001	(rad)
a_{f0}	0.3528594971E-004	(s)
a_{f1}	0.0000000000E+000	(s/s)
Week Number	932	

Table 4.2: Almanac data for SV 16 generated on July 6, 2017

Comparison between subframe 5 and almanac data was done in a number of steps. First, by converting the binary value to its decimal equivalent (shown in blue in the parsed data diagrams). Second, by multiplying the decimal value to its scaling factor. Decimal values marked with '*' indicate that the two's complement of the value must be taken to determine the true almanac value. The two's complement of a binary number is used to represent negative decimal numbers. However an error in the handling of a negative number by the message data generator required that the negative sign be dropped and calculations to restore the true almanac value be done using the absolute value of the two's complement decimal equivalent.

Starting the comparisons with eccentricity defined in word 3 (bits 69 through 84) as $e = 19513$. The almanac data value for e was 0.009305000305. Therefore, $e = 19513 = 19513 \cdot 2^{-21} = 0.0093 \approx 0.009305000305$. The loss in resolution is due to the number of bits used to represent eccentricity. Next was bits 121 through 136 that defined the rate of right ascension $\dot{\Omega}$. Notice that $\dot{\Omega}$ is marked with a '*' indicating that a two's complement conversion needs to be done in order to restore the true almanac value. True almanac value is in units of radians per

second (rad/s) so a conversion to *semi-circles*, units used by the GPS message data, must be done. The complete calculation to restore $\dot{\Omega}$ to its true almanac value is $\dot{\Omega} = 64843 \mapsto \ddot{\Omega} = |-693| = 693 \quad \therefore \quad \ddot{\Omega} \cdot \pi = 2177.1 = 2177.1 \cdot 2^{-38} = 0.7924e^{-8} \approx 0.7931758961e^{-8}$. The process to restore the parsed that to the true almanac data, although convoluted, proves that the correct data was transmitted.

Applying the same process to the remaining almanac parameters in subframe 5 results in the correct restoration of true almanac values. For $\sqrt{A}$ (bits 151 through 175) the restored almanac value was $\sqrt{A} = 10554371 = 10554371 \cdot 2^{-11} = 5153.5 \approx 5153.677246$. For argument of perigee ω (bits 211 through 235) the restored almanac value was $\omega = 15543807 \mapsto \doteq |-1233409| = 1233409 \quad \therefore$ $\dot{\omega} \cdot \pi = 3874900 = 3874900 \cdot 2^{-23} = 0.4619 \approx 0.461919965$. For mean anomaly M_0 (bits 241 through 265) the restored almanac value was $M_0 = 15543807 \mapsto \dot{M}_0 = |-8150331| = 8150331 \quad \therefore \quad \dot{M}_0 \cdot \pi = 2.5605e + 07 = 2.5605e + 07 \cdot 2^{-23} = 3.0524 \approx 3.052356242$. Lastly, a_{f0} defined by combining bits 271 through 278 and bits 290 through 292. The restored almanac value for a_{f0} was $a_{f0} = 37 = 37 \cdot 2^{-20} = 0.35286e^{-4} \approx 0.3528594971e^{-4}$

The decoder parsing results for subframe 2 and 3 are shown in figure 4.35 and 4.36. Applying the same value restoration process to subframe 2 and 3 will yield similar results. Note, almanac data from the transmitting SV was used in subframe 2 and 3 instead of SV 16. Values not defined by almanac were hard coded into the subframe generator functions.
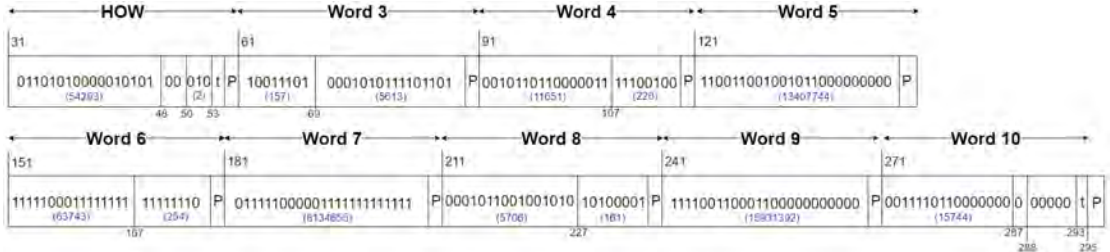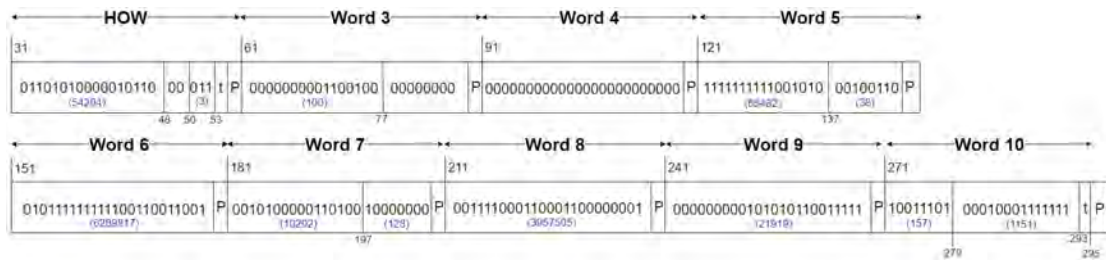

Figure 4.35: Subframe 2 decoded data

Figure 4.36: Subframe 3 decoded data

Chapter 5

Research Conclusion

The results presented in this thesis document shows a successful recreation of the civilian GPS signal using the ROACH processing board and a few off-the-shelf software tools. The recreation of the GPS signal follows the signal characteristics set forth by [19]. It has been shown that the C/A L1 GPS signal was been constructed with all of its parts. When using a software decoder programmed to process signal data using the format prescribed the generated GPS signal contains a correctly generated PRN sequence and full message data. Difficulties arose during acquisition using the GNSS-SDR GPS receiver due unfamiliarity with the GNSS-SDR software and possibly lack of Doppler shift and time delay on the reconstructed signal. Although thoroughly tested, signal acquisition failure may have been due to SNR levels, acquisition power levels, or fluke signal reconstruction errors not discovered during testing.

The process of reconstructing the civilian GPS signal has proven difficult. Even with access to the hardware and software used in this thesis work, producing a true GPS signal worthy of being mistaken as an authentic signal is a task hard to reach. It is the hope of this author that this thesis work has shown how many moving parts must fit together with microsecond precision in order to produce the most rudimentary version of the civilian GPS signal. While achievable, to recreate

a signal accurate enough to be mistaken as an authentic signal is a monumental task requiring years of work, knowledge, and money. While the work focused on the reconstruction of the GPS signal as a side effect it has produced valuable experience with the ROACH processing board and the software tool chain that can be applied to any future signal processing research.

Chapter 6

Future Work

Doppler shift and time delay are used by the GPS receiver to calculate its position. The implementation of Doppler shift and time delay in the reconstructed signal is a natural next step to improve reconstruction accuracy and signal robustness. A simple process for implementing the Doppler shift and time delay is using four BRAMs that are configure with different delay values. A counter would progress through the BRAM address and pass the delay value in that current address to logic that implements the delay onto the signal. The logic would be implemented before it is modulated onto the carrier so that each individual SV could have its own delay encoded into the signal before being added together to create the transmitted composite signal.

Given the proper testing environment, future up conversion of the signal from 50.127 MHz to 1575.42 MHz and transmitting it over-the-air would not only facilitate testing, allowing testing of the generated signal by any GPS receiver, but would also add an important characteristic of the traditional GPS signal. Transmission of the signal at its intended frequency would be valuable as a testing instrument. A robust GPS signal at the L1 frequency would provide any GPS systems in need of testing with a tool that does not required the system to undergo any modifications. It would also allow for robust testing of anti-spoofing

and anti-jamming techniques. Up conversion would require an RF mixer and a signal generator. Preliminary tests have been performed using a Mini-Circuits ZX05-43MH-S+ RF mixer which resulted in a up conversion to 1573.13 MHz. This test was done by connecting the output of ROACH, at 50.127 MHz, to the IF port on the mixer and a signal generator tuned to 1525.293 MHz connected to the LO port. Although not exactly the desired frequency, with some tuning of the local oscillator the desired frequency can be achieved. An emphasis must be made to the fact that no over-the-air transmission was done during this thesis project. With proper signal isolation and frequency up conversion an over-the-air transmission should be tested in the future.

Bibliography

[1] UT News, "UT Austin Researchers Successfully Spoof an \$80 million Yacht at Sea," Jul. 2013. [Online]. Available: `https://news.utexas.edu/2013/07/29/ut-austin-researchers-successfully-spoof-an-80-million-yacht-at-sea`.

[2] E. D. Kaplan and C. J. Hegarty, *Understanding gps: Principles and applications*, 2nd ed. Norwood, MA: Artech House, 2006.

[3] International Civil Aviation Organization, "ICAO Completes Fact-Finding Investigation," *ICAO News Release*, vol. PIO, no. 8/93, 1993.

[4] B. W. Parkinson and S. W. Gilbert, "NAVSTAR: Global Positioning System – Ten Years Later," *Proc. IEEE*, vol. 71, no. 10, Oct. 1983.

[5] C. J. Hegarty and E. Chatre, "Evolution of the Global Navigation Satellite System (GNSS)," *Proc. IEEE*, vol. 96, no. 12, Dec. 2008.

[6] M. A. Earl. (2017). CASTOR - Canadian Astronomy, Satellite Tracking and Optical Research, [Online]. Available: `http://www.castor2.ca/`.

[7] Navigation National Coordination Office for Space-Based Positioning and Timing. (2017). The Global Positioning System, [Online]. Available: `http://www.GPS.gov`.

[8] GPS WORLD, "Almanac: Orbit Data and Resources on Active GNSS Satellites," pp. 1–5, 2017.

[9] 50th Space Wing Public Affairs, "50 SW completes GPS Constellation Expansion," Schriever Air Force Base, Colo., Jun. 2011. [Online]. Available: `http://www.schriever.af.mil/News/Article-Display/Article/277054/50-sw-completes-gps-constellation-expansion/`.

[10] P. Misra and P. Enge, *Global Positioning System - Signals, Measurements, and Performance*, 2nd ed. Lincoln, MA: Ganga-Jamuna Press, 2012.

[11] U.S. Cost Guard, "NAVSTAR GPS: User Equipment Introduction," Tech. Rep., Sep. 1996. [Online]. Available: `https://www.navcen.uscg.gov/pubs/gps/gpsuser/gpsuser.pdf`.

[12] K. Borre, D. M. Ajos, N. Bertelsen, P. Rinder, and S. H. Jensen, *A Software-Defined GSP and Galileo Receiver. A Single-Frequency Approach*. New York, NY: Birkhauser Boston, 2007.

[13] S. Butman and U. Timor, "Interplex-An Efficient Mulichannel PSK/PM Telemetry System," *IEEE Trans. on Communication Technology*, vol. COM-20, no. 3, Jun. 1972.

[14] J. B.-Y. Tsui, *Fundamentals of Global Positioning System Receivers: A Software Approach*. New York, NY: John Wiley & Sons, 2000.

[15] Department of Defense, *Global positioning System Precise Positioning Service Performance Standard*, 4th ed., Feb. 2007.

[16] ——, *Global positioning System Precise Positioning Service Performance Standard*, 4th ed., Feb. 2007.

[17] H. Fukumasa and R. Kohno, "Design of Psudonoise Sequence with Good Odd and Even Correlation Properties for DS/CDMA," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 5, pp. 828–836, 1994.

[18] R. Gold, "Optimal Binary Sequence for Spread spectrum Multiplexing," *IEEE Transaction on Information Theory*, vol. 13, no. 4, pp. 619–621, 1967.

[19] Navigation National Coordination Office for Space-Based Positioning and Timing, *Global Positioning System Directorate Systems Engineering & Integration Interface Specification IS-GPS-200*, Rev. H, Dec. 2015.

[20] Department of Defense and Department of Transportation, "2001 Feredal Radionavigation Plan," Dec. 2001. DOI: `https://www.navcen.uscg.gov/pdf/frp/frp2001/FRP2001.pdf`.

[21] J. McNamara, *GPS for Dummies*, 2nd ed. Noboken, NJ: Wiley Publishing.

[22] Federal Aviation Administration. (2014). Satellite Navigation - Wide Area Augmentation System (WAAS), [Online]. Available: `https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/waas/`.

[23] ——, (2017). Welcome to the William J. Hughes Technical Center WAAS Test Team, [Online]. Available: `http://www.nstb.tc.faa.gov/`.

[24] R. Tubbesing IV, "Commissioning Field Programmable Gate Arrays for Passive Radar Signal Processing," Dec. 2015.

[25] Collaboration for Astronomy Signal Processing and Electronics Research. (Apr. 2013). ROACH, [Online]. Available: `https://casper.berkeley.edu/wiki/ROACH`.

[26] J. Johansen. (Dec. 1999). How To Find a Formula for a Set of Numbers, [Online]. Available: `http://www.johansens.us/sane/education/formula.htm`.

[27] D. Boschen, "GPS C/A Code Generator," version 1.1, *MATLAB*, 2010. [Online]. Available: `https://www.mathworks.com/matlabcentral/fileexchange/14670-gps-c-a-code-generator?s_tid=prof_contriblnk`.

[28] Ettus Research. (2017). USRP N210 Software Defined Radio (SDR), [Online]. Available: `https://www.ettus.com/product/details/UN210-KIT`.

[29] Analog Devices, *Wideband Synthesizer with Integrated VCO*, ADF4350 datasheet, Rev. B, Feb. 2017.

[30] Ettus Research. (2015). USRP Hardware Driver and USRP Manual: USRP2 and N2x0 Series, [Online]. Available: `http://files.ettus.com/manual/page_usrp2.html`.

[31] GNU Radio. (2015). GNU Radio: InstallingGR, [Online]. Available: `http://files.ettus.com/manual/page_usrp2.html`.

[32] ——, (Jul. 2017). Guided Tutorial GRC, [Online]. Available: `https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC`.

[33] Carles Fernandez-Prades, CTTC. (2017). Configurations - About Us, [Online]. Available: `http://gnss-sdr.org/about/`.

[34] Free Software Foundation. (Jun. 2007). The GNU General Public License v3.0, [Online]. Available: `https://www.gnu.org/licenses/gpl-3.0.html`.

[35] C. Fernandez-Parades, J. Arribas, P. Closas, C. Aviles, and L. Esteve, "GNSS-SDR: An Open Source Tool For Researchers and Developers," *Proceedings of the 24th International Technical Meeting of t he Satellite Division of the Institute of Navigation (ION GNSS 2011)*, pp. 780–794, 2011.

[36] Carles Fernandez-Prades, CTTC. (Apr. 2016). Configurations - GNSS-SDR, [Online]. Available: `http://gnss-sdr.org/conf/`.