Design and Implementation of a SoC-Based Real-Time Vector Tracking GPS Receiver

by

Brian A. Keyser

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 10, 2015

Keywords: GPS, software receiver, vector tracking, FPGA

Copyright 2015 by Brian A. Keyser

Approved by:

David Bevly, Chair, Albert J. Smith, Jr. Professor
Peter Jones, Woltosz War Eagle Motor Sports Professor
Victor Nelson, Professor of Electrical Engineering

Abstract

This thesis provides the design and implementation of a GPS receiver which utilizes advanced tracking algorithms on a small, low cost platform. The tracking algorithms used are of a class of algorithms known as vector tracking. Vector tracking receivers have been known to have an increased immunity to jamming and maintain signal lock on weaker signals. These benefits often come at the price of computation time, as the algorithms can require extensive matrix inversion and impose critical timing requirements on the receiver. To handle the computational burdens, a system-on-chip implementation was chosen using the Xilinx Zynq architecture. This architecture couples an FPGA with a dual-core ARM processor in a small package and can be acquired on development boards at a low cost. This thesis demonstrates how this architecture is utilized to overcome the strict timing requirements of a real-time vector tracking GPS receiver. The design and implementation of the receiver is described such that it can be used as an aide in the development of other advanced acquisition, tracking, or navigation algorithms. Performance results are given in regards to tracking, positions, and processing times.

Acknowledgments

First and foremost, I give every bit of credit to Jesus Christ, my Lord and Savior. Without the overwhelming impact He has had in my life, I could not be where I am or accomplished the things I have done. I am immensely thankful for His grace, of which I have done nothing to deserve. My wife Sara has been fantastic throughout this whole process; she continues to love and support me and is the reason I was able to continue my education. I would also like to thank my friends and family for all of the words of encouragement and support in various forms.

I owe a great debt of gratitude to Dr. David Bevly for the opportunity to pursue a master's degree. The financial support I received through Dr. Bevly helped make it possible, but his time and mentoring have made it a truly enriching experience. Dr. Jones and Dr. Nelson have both made themselves available well outside their duties, and I am very grateful for the time and patience they have shown me. I want to thank the other GAVLAB members for all of the guidance and assistance I received. In particular, Scott Martin has been pivotal in my accomplishments in the GAVLAB. His wisdom and encouragement were truly a blessing. I'd also like to extend my gratitude to Integrated Solutions for Systems for their support of this research. Specifically, David Hodo has worked through my learning process with great patience and help along the way.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Since the Global Positioning System (GPS) was first declared operational in 1995, significant amounts of research have gone into improving the accuracy and reliability of GPS [1]. With the removal of Selective Availability in 2000, civilian users were able to see drastic increases in the accuracy of GPS receivers. The integration of more constellations (GLONASS, Galileo, Bei-Duo, etc.) has helped to improve the overall accuracy and reliability of satellite navigation systems as well [2]. Still, there are situations where satellite navigation is hindered by the availability and strength of the satellite signals. These situations can be in large cities (urban canyon) or heavily wooded areas (dense foliage). While operating in these situations, a GPS receiver may be unable to track satellites due to blocked line-of-sight availability or weakened signals. While other navigation methods such as odometry or inertial systems can be used to aid GPS in these situations, it is imperative that GPS remain accurate and reliable throughout these situations.

Consider navigating through an urban canyon such as in Figure 1-1 using GPS or some other type of satellite navigation. Because GPS requires clear line-of-sight from the user to the satellite, often buildings can obstruct the user and cover most of the area where satellites would

be seen. Not only can this adversely affect the accuracy of the user's position and velocity, but if enough satellites are blocked then this situation can prevent satellite navigation altogether.

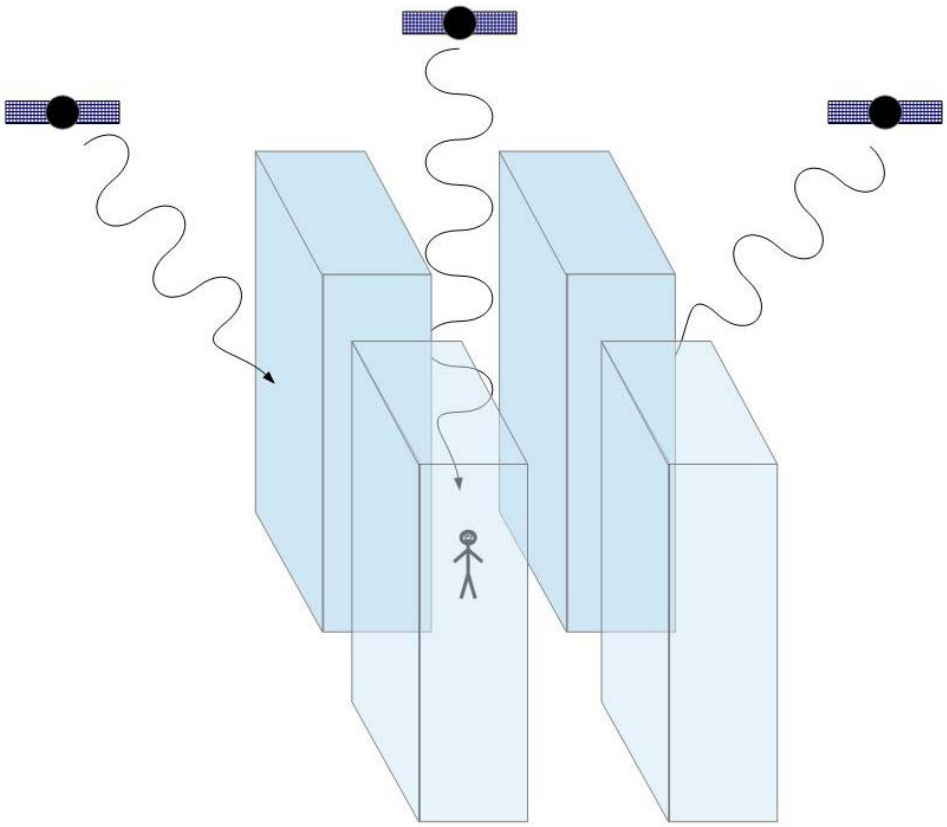

Figure 1-1 : Urban Canyon

Likewise, in areas with heavy foliage, the GPS signals can be attenuated or even completely blocked. Since the GPS signals are already extremely weak once reaching the surface of the Earth, any additional attenuation can cause severe negative impacts on navigation solutions.

In recent decades, numerous efforts have been made to mitigate the negative performance that satellite navigation methods incur from these situations. Vector tracking has been one effort

where significant strides have been made towards such improvements [3] [4] [5] [6] [7]. This method involves modifying the signal tracking process to utilize an optimal estimation technique, which enables satellites with stronger received signals to aid in the tracking of weaker signals. Several benefits can result from aiding the tracking loops, such as a more robust navigation solution in adverse environments and instant reacquisition of satellites after short losses of line-of-sight.

However these benefits come at a cost. Because vector tracking is involved at the signal processing level of the receiver, it cannot be implemented on commercial receivers that do not allow modifications to the tracking algorithms (which includes most, if not all Commercial-Off-The-Shelf receivers). Furthermore, the integration of the optimal estimation techniques requires significant processing abilities. When cost and size are not a concern, computational intensity isn't an issue. However there is a large demand for reliable navigation utilities with small, low cost, low-power implementations.

The work described in this thesis seeks to resolve these issues. Others have documented some of the issues involved in this implementation, however a vector tracking receiver in a small and inexpensive package has yet to be completed [8] [5].

A small, low-cost hardware platform is chosen that is able to handle the computational burden of vector tracking. A GPS receiver which implements a type of vector tracking algorithms is designed and constructed on a Zynq XC7Z020. The platform used is highly programmable, and the implementation details are given in the thesis. This enables readers to pursue expansion of the receiver developed in this work so that perhaps more advanced algorithms can be easily implemented or other constellations can be integrated.

1.2 Software Receivers

Software receivers have emerged in recent years as a major tool in the development of wireless communications [9] [10]. Software receivers are easily programmed and reprogrammed and thus have become a great development tool for researching new communication systems and signal processing algorithms. Because processing speeds continue to increase, software receivers can be first quickly tested using post-processed data before being implemented in a real-time design.

Developing GPS receivers from a software receiver serves many purposes. First, there is immense flexibility in the design process, as well as numerous means by which to analyze performance [2]. Some software receivers offer real-time operations, others are used as a post-processing tool to aid in the evaluation of new techniques and algorithms. Often times, these can lend themselves to be developed into hardware, as a full-hardware implementation or a mixture of hardware and software.

1.3 System-On-Chip

A system-on-chip (SoC) design is one in which an entire computing system (including processing, clocking, memory, etc.) is implemented on a single chip. This allows for significant improvements in terms of:

- Ease of implementation
- Signal quality
- Power consumption

- Size

Many new designs are moving toward SoC implementations because advances in technology are allowing more components to be put on smaller platforms and become integrated into smaller packages [11]. The SoC used in this work comprises two main components: an FPGA and a microprocessor.

1.3.1 Field-Programmable Gate Array

A Field-Programmable Gate Array (FPGA) is essentially programmable hardware. They are integrated circuits that are made up of programmable logic blocks (PLB), input/output (I/O) cells, interconnect, and configuration memory, as in Figure 1-2 [12]. The actual function of an FPGA can be programmed into it, such that the same hardware can be utilized in an unlimited number of ways. By configuring the PLBs, I/Os, and interconnects, the function of the FPGA can be set, reset, or even modified during operation.

FPGAs are often used in the process of designing application specific integrated circuits (ASIC). ASICs are designed for a single hardware implementation; however, they can perform the functions in that implementation with great speed and efficiency. ASICs tend to be smaller, faster, less expensive, and use less energy than their FPGA counterparts. But this comes at a very high initial cost to produce an ASIC. An FPGA can be used as a means of prototyping a particular hardware implementation prior to making ASICs, so that multiple iterations of the ASIC do not need to be produced as the hardware implementation is refined.

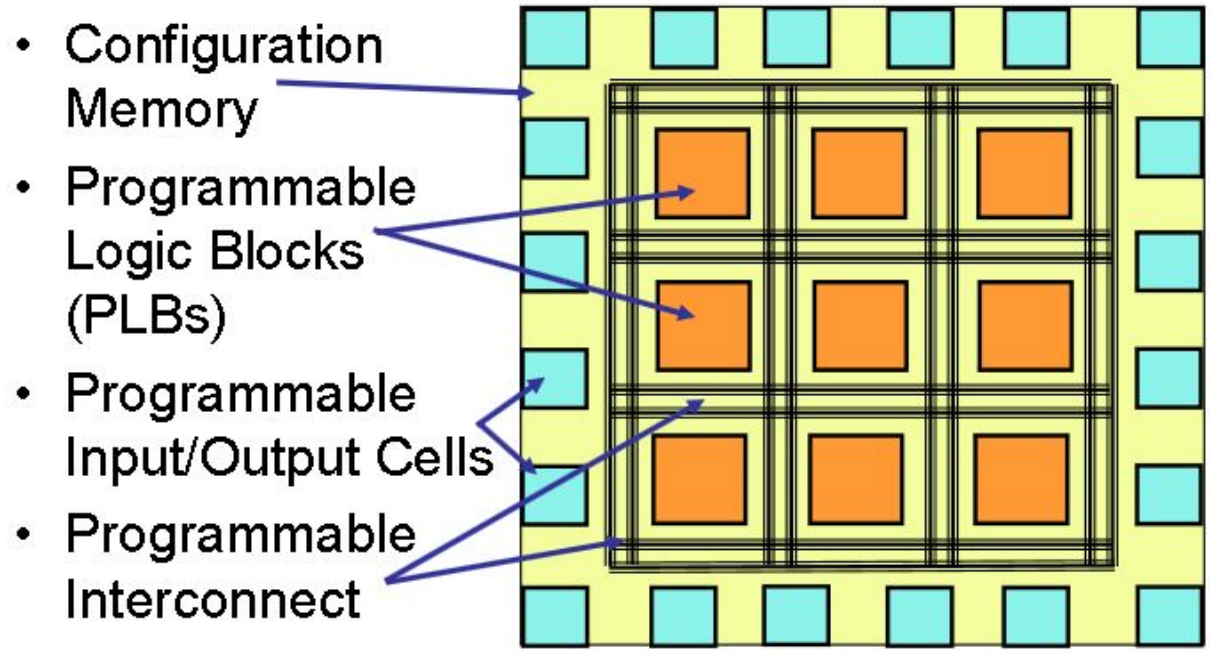

Figure 1-2 : FPGA Layout [12]

FPGAs have many qualities that make them great platforms for GPS receivers. As mentioned above, the ability to be reprogrammed allows for flexibility and forgiveness during the design process. Even during operation, FPGAs can often operate near the speeds of comparable ASICs, although at a small fraction of the development costs. FPGAs are also highly parallelizable; functions that need to occur at the same time can be programmed onto separate parts of the FPGA and allowed to run simultaneously. This is particularly helpful in the design of GPS receivers, as tracking channels need to operate quickly and simultaneously [13] [14].

There are other aspects of FPGAs, however, that are less desirable for this application. Power consumption has long been an area of concern for more capable FPGAs. Long strides have been made in this area as manufacturers have begun providing low power options, and often designs can be modified to require less power. Also, sometimes simple mathematic

calculations can be expensive in terms of FPGA resources. This is particularly difficult when an operation only needs to be completed once – the FPGA resources must be allocated for this operation. There are options to reconfigure FPGAs during operation (Run-Time Reconfiguration); however this is one of the more complicated topics in regards to FPGA design [12]. Along those lines, FPGA definitions are produced from Verilog/VHDL models (VHSIC [Very-High-Speed Integrated Circuit] Hardware Description Language), neither of which is as common among young engineers as C or C++. Again, strides have been made to allow programming of FPGAs through other programming environments, such as MATLAB or LabVIEW, but these are still only growing in familiarity and popularity.

1.3.2 Specialized Cores

As a result of seeing customers repeatedly use FPGAs for similar functions, FPGAs also now integrate more specialized functions into the fabric of the chips. Block RAMs (BRAM), Digital Signal Processing (DSP) slices, and microprocessors are just some of the specialized cores now available within FPGAs. The integration of these devices is what makes the SoC structure able to accomplish so much.

1.4 Contributions and Outline

On the topic of vector tracking algorithms, much work has been done in the development and validation of such algorithms. Some work has also been done in developing a vector tracking receiver for use on a home PC [15]. Previous efforts at developing a small, low cost solution to a real-time vector tracking receiver have come up short due to a lack of sufficient hardware at the time [8]. This thesis provides a highly programmable platform for use in the design, testing, and

implementation of advanced algorithms for software receivers. A real-time vector tracking GPS receiver has yet to be implemented on a small, low cost platform and thus this thesis provides details to the design and implementation of such a receiver.

A background of the Global Positioning System and the signal structure of the satellite signals will be provided in Chapter 2. This lays the foundation for how the processing of the GPS signals will be accomplished on the SoC. Chapter 3 lays out the typical structure of a GPS receiver, explaining the processes that a receiver uses to produce a navigation solution. The algorithms used in the vector tracking receiver will be developed and the use of such algorithms in the vector receiver will be discussed. Implementation details are given in Chapter 4 where the allocation of processing tasks, hardware design of the FPGA, and software design of the ARM programs are explained. Chapter 5 highlights the results of the work, showing navigation performance as well as processing performance. Because the performance in terms of the accuracy of the navigation solution has been well documented for these algorithms, more effort is given to the processing performance and the effect this has on the receiver's operation. Finally, conclusions from the work are presented in Chapter 6 along with possible expansions and improvements for the receiver.

Chapter 2

Overview of GPS Signal Structure

2.1 Background

This chapter discusses the structure of GPS signals and the properties of the codes transmitted. The unique properties of these codes will be the basis of how the receiver acquires and tracks these signals. The data carried by the signals will also be introduced.

GPS users rely on electromagnetic waves sent from the satellites orbiting Earth and received by their antenna. In the operation of a GPS receiver, the receiver does not send any information or signals back to the satellites; it only receives signals and determines user position, velocity, and time based on the processing of the received signals.

In general, a radio signal can be described as:

$$s = Asin(2\pi ft + \theta) \tag{2.1}$$

where A is the amplitude, f is the frequency in Hz, and θ is the initial phase [16]. Information can be carried by the signal by modulating any of these three parameters. Amplitude modulation (AM) and frequency modulation (FM) are commonly used today. As the names imply, AM modulates the amplitude of the signal and FM modulates the frequency. When the phase is

modulated, it is called phase modulation. An example of these modulation methods can be seen Figure 2-1.

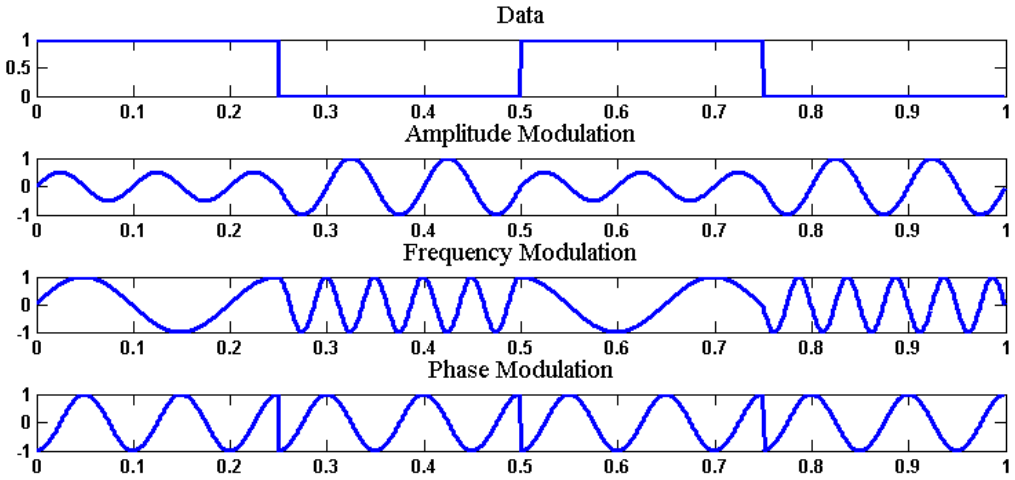

Figure 2-1 : Modulation Techniques

GPS is a phase modulated signal with phases of 0 and π; because it modulates between two phases, it is called a binary phase-shift keying (BPSK) signal [17]. The phase modulation signal in Figure 2-1 shows a BPSK signal. A single frequency is used and a shift in phase is used to indicate a change in data. This can be related as multiplying the signal by a ±1 to shift the phase (notice that for a shift in data, the BPSK signal of Figure 2-1 changes from a value of +1 to -1). The rate in which the data is transmitted plays a key role in the frequency spectrum which the BPSK signal occupies, which is explained more in the description of the signal from the Frequency Domain.

2.2 Time Domain Description

GPS satellites broadcast three signals simultaneously. Each signal comprises a carrier, code, and navigation data as seen in Figure 2-2. The code and navigation data are represented as square waves shifting between ±1, while the carrier is a sinusoidal wave. The transmitted signal is the product of all three components. Note that in Figure 2-2 the x-axis is not to scale: each component of the final signal has a vastly different frequency. This serves as an illustrative example of how the three components are multiplied together to produce the transmitted signal.

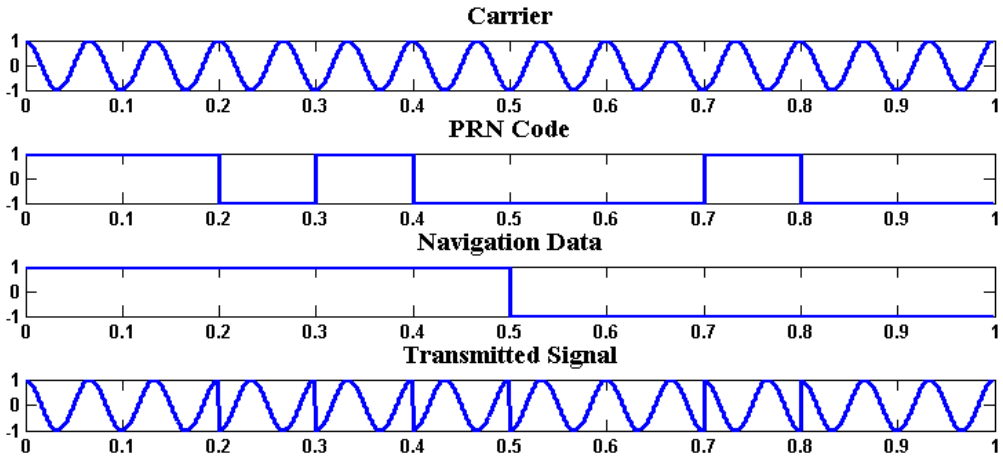

Figure 2-2 : GPS Signal Structure

The three signals can be described as below [1]

$$s_{L1}(t) = \sqrt{2P_C}D(t)x(t)\cos(2\pi f_{L1}t + \theta_{L1}) + \sqrt{2P_{Y1}}D(t)y(t)\sin(2\pi f_{L1}t + \theta_{L1}) \quad (2.2)$$

$$s_{L2}(t) = \sqrt{2P_{Y2}}D(t)y(t)\cos(2\pi f_{L2}t + \theta_{L2}) \tag{2.3}$$

These three signals are comprised of two different frequencies and two different codes. The civilian code is broadcast on the L1 frequency ($f_{L1} = 1575.42\,MHz$). The military code is broadcast on both the L1 frequency and the L2 frequency ($f_{L2} = 1227.60\,MHz$). Because the military code is not available to civilian users, this thesis will focus solely on the civilian code (the top line of (2.2)).

The civilian code is made up of four components: amplitude ($\sqrt{2P_C}$); navigation data ($D(t)$); spread spectrum code ($x(t)$) and a high frequency carrier ($\cos(2\pi f_{L1}t + \theta_{L1})$). The amplitude of the signal is related to the average power in the transmitted signal [1]. Once the signal is received at a GPS receiver's antenna, this amplitude has decreased due to the distance the signal has traveled as well as other environmental effects. The navigation data, which will be described in more detail later, is transmitted at 50 bits per second and contains information about the satellites. The spread spectrum code is a unique code to each satellite. It is a pseudo-random sequence that is 1023 chips long and transmitted at a chipping rate of 1.023 MHz. Chips and chipping rate are generally used when referencing the GPS spread spectrum code because the "chips" are pseudo-random and do not actually carry any information like "bits" do [18]. The high frequency (radio frequency, RF) carrier is a sinusoidal wave with a given frequency and initial phase. The carrier is realized as a cosine wave at the L1 frequency; the initial phase is considered to be zero. However it should be noted that there is likely to be a phase difference between the transmitted signal and the receiver, even if the frequency is properly determined. All four components are multiplied together to result in the transmitted signal.

2.3 Frequency Domain Description

Some signal processing techniques are greatly utilized by means of a good understanding of the frequency domain representation of the GPS signal. This can be accomplished by taking the Fourier Transform of the time domain signal. Beginning with Equation (2.4)

$$s_{C/A}(t) = \sqrt{2P_C}x(t)\cos(2\pi f_{L1}t) \tag{2.4}$$

note that the navigation data is missing from the equation. The spread spectrum code is used in place of both the spread spectrum code and the navigation data because it simplifies our analysis without modifying the outcome [1]. The period of a bit in the navigation data is 20 milliseconds, during which the spread spectrum code will have gone through 20x1023=20460 chips. The effect of the navigation data is essentially to flip the sign of all 20460 chips of the spread spectrum code. The net effect of this on the Fourier Transform is negligible. In (2.4), we also ignore the initial phase, as it can be considered zero for this analysis.

The Modulation property of Fourier Transforms states that a signal multiplied by a sinusoid simply shifts the Fourier Transform of the original signal to the positive and negative of the modulating frequency as in Equation (2.5).

$$\mathcal{F}\{a(t)\cos(2\pi f_0 t)\} = \frac{1}{2}A(f - f_0) + \frac{1}{2}A(f + f_0) \tag{2.5}$$

Applying (2.5) to (2.4) results in

$$F\{s_{C/A}(t)\} = \frac{\sqrt{2P_C}}{2}X(f - f_{L1}) + \frac{\sqrt{2P_C}}{2}X(f + f_{L1}) \tag{2.6}$$

All that is needed is the Fourier Transform of the spread spectrum code. The majority of the impact of the spread spectrum code on the Fourier Transform can be seen by simplifying the code into a single pulse that is the width of a chip. The Fourier Transform of a pulse with amplitude A and duration T is given by (2.7).

$$F\left\{Ap\left(\frac{t}{T}\right)\right\} = ATP(fT) = AT\frac{\sin(\pi Tf)}{\pi Tf} = ATsinc(\pi Tf) \tag{2.7}$$

The amplitude of the spread spectrum code is unity, and the period is approximately 1 microsecond. The time domain and frequency domain representation of this can be seen in Figure 2-3. Notice that the frequency domain representation is shown as the magnitude, such that the sinc function is always positive.

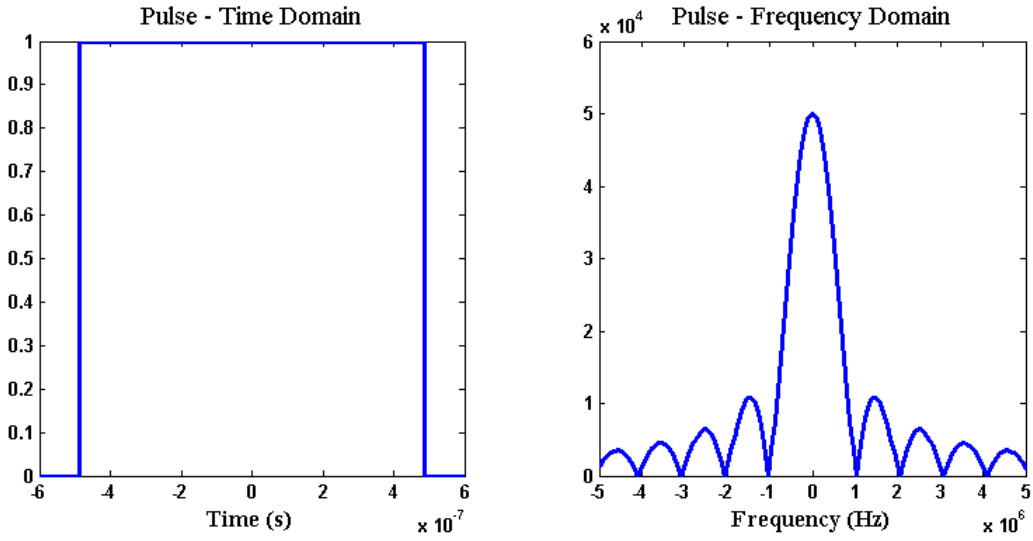

Figure 2-3 : Pulse - Time Domain and Frequency Domain Representation

When the pulse in Figure 2-3 is modulated using the cosine carrier at the L1 frequency, the Fourier Transform is simply the sinc function in Figure 2-3, shifted to $+f_{L1}$ and $-f_{L1}$. The frequency spectrum of the civil signal will be dominated by this frequency-shifted sinc function. However, because the signal has been simplified, some other characteristics of the spectrum have

left out. The navigation data and the carrier phase offset will both have very small effects on the final spectrum, as well as the influence of the code as an entire sequence instead of a single pulse [1]. Additionally, there is the consideration of the repetition of the code. With all things considered, the final spectrum would be dominated by the frequency-shifted sinc function multiplied by a series of delta functions in spacings of 1 kHz.

This analysis aids in the understanding of how the signal shifts in frequency due to up or down converting, as well as gives a good understanding of the bandwidth that the signal occupies. The GPS signals can all be contained within the 2 MHz width of the main lobe in Figure 2-3. Thus, whenever the signal is filtered, a bandpass filter of 2 MHz bandwidth centered on the carrier frequency can be utilized.

2.4 Received Signal Power

When GPS signals reach the Earth's surface, they are very weak. The input power level for the antenna on the satellites is only about 50 Watts, and only half of that is devoted to the civilian signal [1]. Consequently, the specification for the minimum received power level for civilian users on the Earth's surface (given no other hindrances to the signal) is -160 dBW. In general, signal power is slightly stronger and a received power of about 10^{-16} Watts is normally observed in clear skies [1]. This, unfortunately, is well below the level that is detected by an antenna. Consequently, the signals described in the Time and Frequency Domain sections above cannot be seen by hooking an oscilloscope or a spectrum analyzer up to an antenna. Signal power level is the limiting factor to the use of GPS, as that makes it very easy to overwhelm with other signals (intentionally and unintentionally). There is, however, a structure within the GPS signals that allow it to be detected amongst greater powered signals. The carefully constructed

spread spectrum codes (Gold Codes) have unique correlation properties that give the signals an extra boost to be detected amongst noise [17]. These codes, as well as the utilization of their correlation properties, are the topic of the following section.

2.5 Gold Codes

The GPS civilian signal utilizes codes that are termed Gold codes or Gold sequences (also referred to as Coarse Acquisition Codes or C/A Codes) [19]. They are pseudorandom binary sequences that have unique correlation properties that benefit the GPS signals in multiple ways. Not only does this give the processing gain necessary to allow the signals to be detected amongst noise, but it also enables all of the satellites to broadcast on the same frequency and occupy the same frequency spectrum.

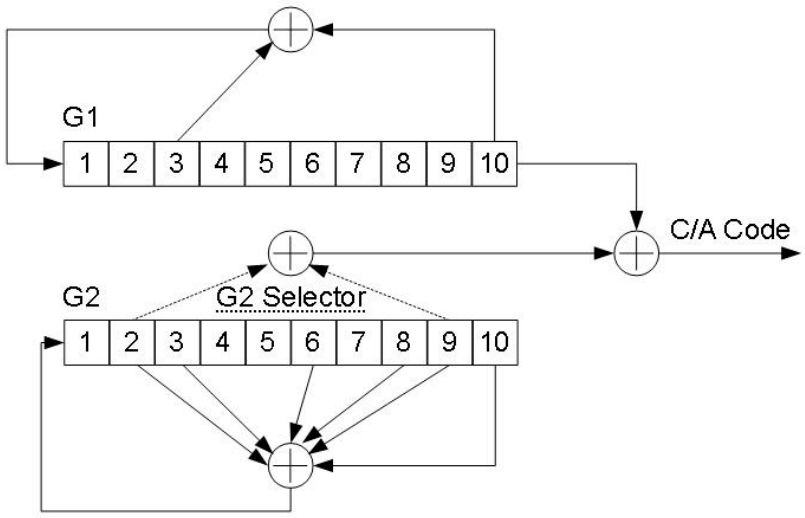

Figure 2-4 : C/A Code Generation

The Gold Codes are generated using two maximal length linear shift register sequences [20]. The generation of the signals follows the diagram in Figure 2-4. The diagram consists of two shift registers, G1 and G2, and the output is a combination of certain combinations of the elements in the registers. The generation begins with both registers filled with ones. Two elements from G2 are passed through an exclusive-or (XOR), the G2 Selector output is XOR'd with the last element in G1, and the result is the output. The element returned to the beginning of each register is also a combination of multiple elements. Each GPS satellite uses a different pair of elements from G2 in the G2 Selector to generate the output; this is how the different codes are determined for all of the satellites. There are 37 unique combinations for the G2 elements, of which 32 are used for the C/A Codes [17]. Both shift registers will return to being filled with ones after producing 1023 outputs, after which they will repeat. Thus each of the Gold Codes is 1023 elements long.

The GPS C/A codes are also called pseudorandom noise (PRN) sequences. The sequences may appear random just due to the fact that they do not repeat for over 1000 samples; however, the name implies more and serves as a good introduction to the correlation properties of the codes. Consider a sequence generated by tossing a coin and writing +1 for heads and -1 for tails. If one were to guess what the value of any one toss was, it would be a guess or random. Furthermore, the expected value of the outcome would be zero; even though a zero cannot be generated, there should be approximately equal amounts of +1s and -1s. Now, if two sequences are generated in this fashion, there would be no correlation between the two, meaning that knowledge of the values in one sequence will not enable you to predict the values in the other sequence. Thus it would be said that these sequences are not correlated. Also, when in the middle of generating one sequence, knowledge of all of the past values of the coin toss cannot help

predict the value of the next coin toss (remember that it is “expected” to be zero, but that won’t happen). Thus we would say that each sequence is not correlated with itself in time [21]. These properties which make up our fictitious random sequences are found in the Gold Codes.

Correlation is a measurement of the similarity of two waveforms: auto-correlation measures the similarity between a waveform and a time-shifted version of the same waveform, and cross-correlation measures the similarity between a waveform and all time-shifted versions of another waveform [1]. Naturally, the auto-correlation of a sequence will have a maximum value with a time shift of zero. Because the Gold Codes are not correlated in time, the auto-correlation is nearly zero with every other time shift. This can be seen in Figure 2-5, where the replica was shifted to give a correlation peak at 823 chips. In Figure 2-5, the normalized auto-correlation is plotted versus a shift in chips. Because the code length is 1023, the maximum value would be 1023 for a time shift of zero. In Figure 2-5, the values are normalized to this maximum value. There are some interesting things to note about this idealized auto-correlation that will aid in some of the signal processing in the receiver:

- The normalized value at the idealized peak is 1
- The normalized value at ±1/2 chips is 0.5
- The normalized value at ±1 chip is ≈0
- At whole chip delays, the normalized auto-correlation will take on values of [-1/1023, -65/1023, 63/1023] precisely
- The side lobes (whole chip delays other than the peak) will be at least 24 dB weaker than the peak.

In particular, take notice of the first three bullets. The autocorrelation will take on the shape of a triangle around the peak, where half of the amplitude will be at a half chip distance from the peak and the amplitude will be ≈0 at 1 chip distance and greater.

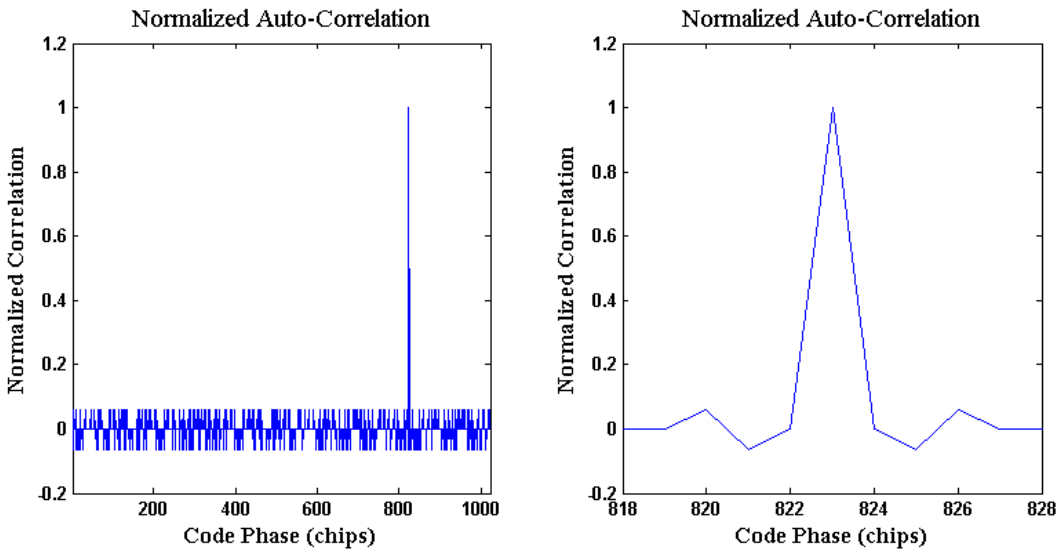

Figure 2-5 : Normalized Auto-Correlation of GPS Gold Codes

Similar to the auto-correlation, the cross-correlation of the Gold Codes can be well predicted. The cross-correlation of two GPS Gold Codes is shown in Figure 2-6. For all time shifts, the correlation value is near zero. In fact, just as with the auto-correlation, these values can be predicted. The normalized cross-correlation of two GPS Gold Codes (at whole chip delays) will take on the values [-1/1023, -65/1023, 63/1023] precisely. This ensures that the cross-correlation will be at least 24 dB weaker than an auto-correlation peak.

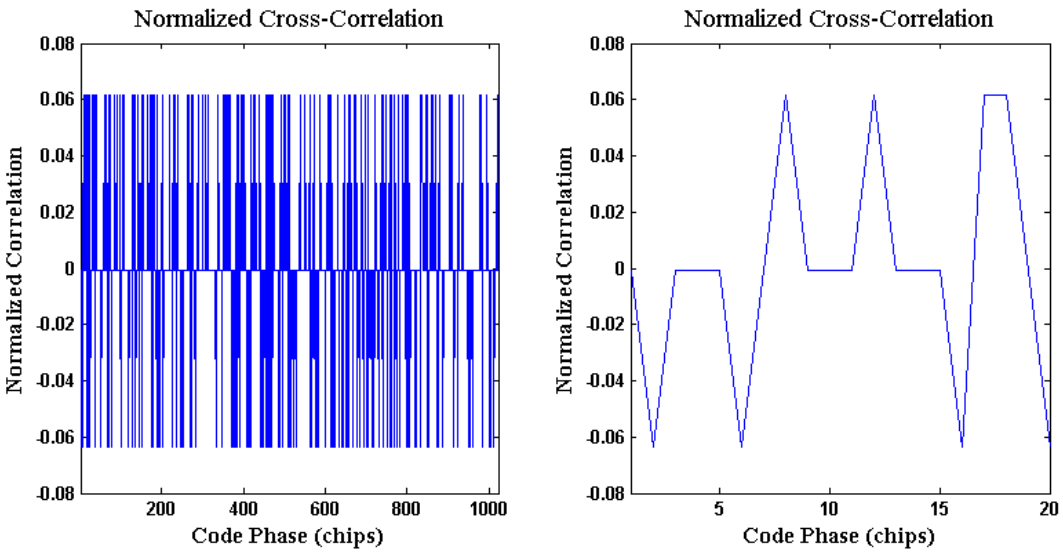

Figure 2-6 : Cross-Correlation of GPS Gold Codes

These correlation properties allow the GPS constellation (which consists of up to 32 satellites) to continuously transmit different data from each satellite at the same time. This scenario can be likened to AM or FM radio stations. All radio stations in a particular area transmit different information at the same time. However these are all transmitted on different frequencies, which allow the radio in your car to hone in on one station. GPS on the other hand, transmits from every satellite on the same frequency. The information can be decoded from each satellite by knowledge of the Gold Code being transmitted from that satellite. This concept is called code division multiple access (CDMA).

2.6 Navigation Data

As mentioned before, in addition to the Gold Code and carrier, the GPS satellites also broadcast navigation data at 50 bits per second. This data creates another modulation to the phase of the carrier, again between 0 and π. The navigation data consists of timing information as well

as satellite orbital parameters. There are five subframes in the GPS Navigation Data; each subframe contains 10 words of 30 bits each (300 bits per subframe, 1500 bits total), and the sequence of subframes repeats indefinitely. At the beginning of each subframe are a Telemetry word (used for signaling the beginning of a subframe) and a Handover word (used for determining Time of Week, configuration flags, and subframe ID). Every word following these two contain information about the current satellite's orbital parameters (called Ephemeris and is found in the first three subframes) or information about all of the satellites in orbit (called Almanac and is found in the last two subframes). The navigation data is essential to determining the user position, velocity, and time. For more information about the structure of the navigation data subframes and calculating satellite positions from their parameters, see the GPS ICD [22].

Chapter 3

GPS Receiver Structure

3.1 Front End

GPS front-ends follow the same general structure of other RF receivers. The primary purpose of the front-end is to bring the RF signal down to a frequency that can be easily sampled by an Analog to Digital Converter (ADC). Front-ends also have secondary purposes of filtering and amplifying the signal, generally in the analog domain, so that signal processing in the digital domain can be done more easily [23]. A typical structure of a front-end is shown in Figure 3-1.

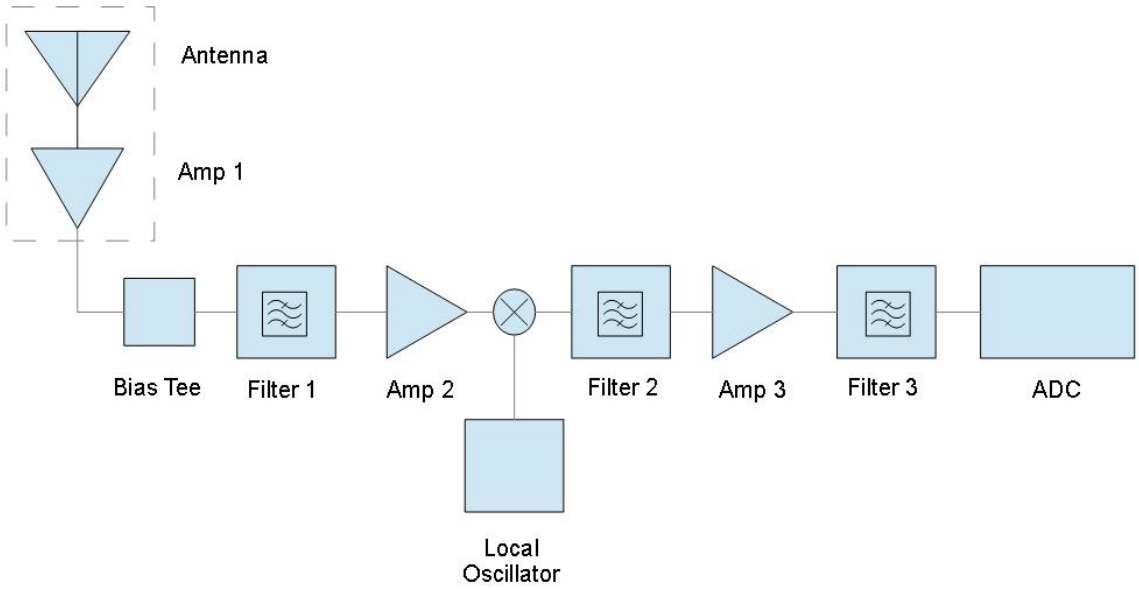

Figure 3-1 : Front-End Block Diagram

The signal path begins with the antenna receiving the RF signal. Antennas are constructed to be receptive of particular frequencies, thus a GPS antenna will be designed for either the L1 or L2 frequency (or sometimes both). The signal is very weak at this point, and must be amplified. Ideally, this amplification will occur very close to the antenna so that the weak RF signal is not subjected to more noise. In an active antenna, power is fed to the antenna so that an amplifier is placed inside the antenna. In a passive antenna, this amplification must be done outside of the antenna. An active antenna is shown in Figure 3-1. A bias-tee is shown following the antenna; the bias-tee is used to power the antenna through the same cable that the RF signal is passed through. A schematic of a bias-tee is shown in Figure 3-2. DC power is applied to Port 1, which is passed to the antenna through Port 2. Port 3 is the RF output that goes to the front-end.

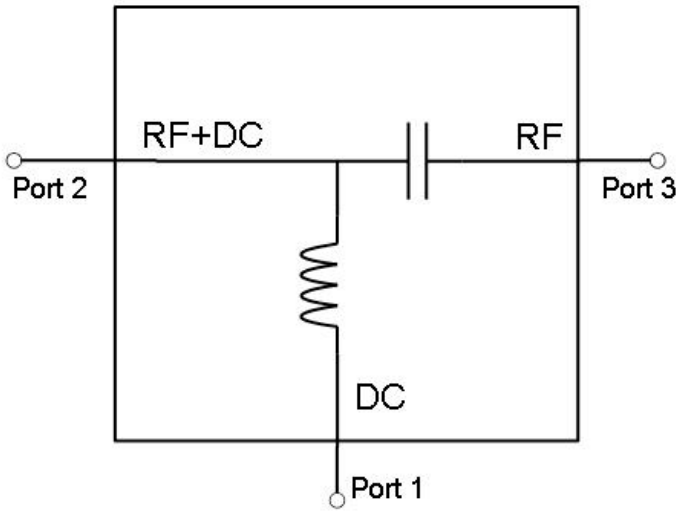

Figure 3-2 : Bias Tee Schematic

After the bias-tee, the RF signal is filtered and amplified again. The purpose of filtering is to eliminate some of the frequency content of the received RF signal. As mentioned in Chapter 2, the frequency spectrum of the GPS signals is well known and only occupies a particular frequency band. With this knowledge, a band-pass filter can be used to attenuate signals in frequencies outside of the GPS spectrum.

Following this stage of filtering and amplification, a local oscillator is used to down convert the RF signal to a frequency that is more easily sampled by the ADC. Recall from trigonometric identities, that the product of two sinusoids is the sum and difference of the frequencies, as in (3.1).

$$\begin{aligned} &\mathrm{A}_1 \sin(2\pi f_1) * A_2 \sin(2\pi f_2) \\ &\qquad = \frac{A_1 A_2}{2}\sin\big(2\pi(f_1 - f_2)\big) + \frac{A_1 A_2}{2}\sin\big(2\pi(f_1 + f_2)\big) \end{aligned} \tag{3.1}$$

Thus if f_1 is the L1 GPS frequency, a local oscillator (LO) can be chosen to bring the difference of the two frequencies to a desired value. The mixing process is called either down-

conversion or up-conversion, depending on whether the desire is to keep the difference or the sum (in this case, it is called down-conversion as the desire is to keep the lower frequency). Because the down-conversion process has the effect of decreasing the signal amplitude and adding unwanted frequency content, the down-converted signal is both amplified and filtered again.

In Figure 3-1, three separate filtering and amplification stages exist in the signal path. This may seem slightly redundant, however there is special purpose to it (often times, front-ends may have more than 3 filtering and amplification stages). The amplification in the antenna is used because the antenna and receiver are generally not in the exact same position, and thus the signal must traverse some distance; if the signal is not amplified, then the signal may become overwhelmed by noise in the local environment. Generally speaking, IF amplifiers have lower cost than RF amplifiers, thus many front-ends are heavy in IF amplifiers [17]. However, it is also desirable to have amplifiers in different frequency ranges; thus in the front-end, multiple amplifiers are used to increase the signal power throughout the conversion process [23]. Filter 1 is used to limit the input bandwidth of the signal; Filter 2 is used to suppress unwanted frequencies generated by the mixer; Filter 3 is used to limit noise generated by IF amplifiers [17].

The final stage of the front-end is the ADC. Here, the IF signal is sampled into discrete values for digital signal processing. There are various ADCs that can be used, and they are generally described by sampling frequency and number of bits in the output. Most GPS receivers use at least 2 bits, while the sampling frequency is dependent on the IF and processing constraints [2].

3.2 Traditional Receiver

As described above, the RF front-end handles the analog signal processing prior to the digital signal processing that occurs in the remainder of the receiver. The operations of a typical GPS receiver will be described here as they occur following the processing from the front-end. In general, the satellite signals are first detected in the Acquisition process, followed by the Tracking process, where the signals will be tracked with local replicas such that navigation data can be retrieved and ranging information can be extracted. With navigation data and ranging information, the receiver can produce a navigation solution.

3.2.1 Acquisition

Acquisition serves two main functions in a GPS receiver. First, acquisition reveals to the receiver which satellites are present in the incoming signal. This can be estimated given other information such as time and approximate location, but a stand-alone receiver with no such information must run through an acquisition sequence to determine which satellites are available. Second, acquisition gives the receiver approximate values for the code phase and carrier frequency for a particular satellite. These approximations will be passed along to the tracking loops to enable them to begin tracking the incoming satellite's signal. These two functions are accomplished simultaneously.

Two acquisition schemes will be discussed in this thesis: serial search and parallel code phase search. In both schemes, the correlation properties of the PRN codes will be exploited; however each one operates in a different domain. Each scheme generates a local estimate of a particular satellite's incoming signal (PRN sequence modulated on an intermediate frequency

carrier) and correlates that estimate with the data received from the RF front-end. If there is a "spike" in the correlation, then the satellite is present and the parameters used to generate the local estimate can be used by a tracking loop. A spike is a single value that raises a significant value above the other values as in the plots in Figure 2-5. If there is not a spike in the correlation, then the receiver can either search for a different satellite or change the local estimate and correlate again. Unfortunately, the presence of a spike is a bit of a relative definition. Signal strength, among other factors, can determine how large a spike is; this could produce inaccurate results if the receiver is only searching for a certain threshold. Instead, the receiver will generate a table of correlation values with one axis as code phase and the other axis as carrier frequency. Once the table has been filled, a search of all the correlation values can be done and the presence or absence of a relative peak can be determined. While both schemes follow this generic sequence, they accomplish the task in very different ways.

3.2.1.1 Serial Search Acquisition

The serial search acquisition method is the simpler of the two and easier to implement. It is the more intuitive of the two because it implements the correlation process in the time domain. Figure 3-3 shows a block diagram of the correlation process in the serial search algorithm.

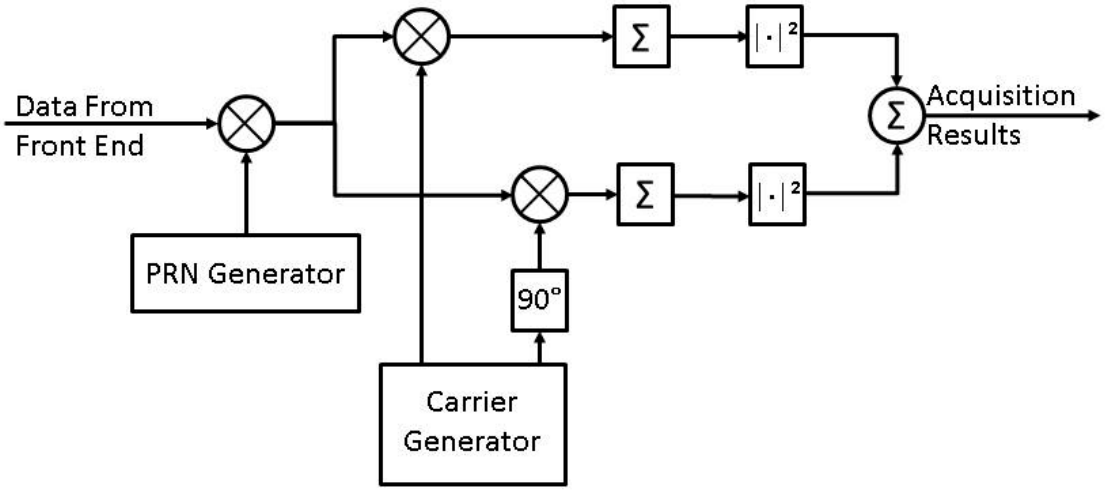

Figure 3-3 : Serial Search Acquisition Block Diagram

The incoming signal in Figure 3-3 is mixed with the output from a PRN Generator. The PRN Generator produces a replica of the satellite's Gold Code, represented as +1s and -1s. The result is then split into two branches: an In-phase branch and a Quadrature branch. The In-phase branch (the top branch in Figure 3-3) is mixed with the Carrier Generator's output. The Carrier Generator's output is a sine wave at the frequency of the IF plus an estimated Doppler frequency. The result of this mixing is summed and squared. In the Quadrature branch, a similar process occurs, except the Carrier Generator's output is shifted by 90° (resulting in a cosine wave). Once both branches are summed and squared, the branches are summed together to give the correlation output. The In-phase and Quadrature branches are both required to account for an error in the carrier phase between the incoming signal and the local replica. If only one branch is used, then there can exist a maximum carrier phase error of $\frac{\pi}{2}$ (intuitively it would be π, however a phase difference of π is the same waveform multiplied by -1 and the correlation value will be the same

magnitude). This carrier phase error can significantly reduce the correlation value magnitude. By adding the 2nd branch, this carrier phase error is reduced to a maximum of $\frac{\pi}{4}$.

The correlation output is stored in a table like the one shown in Figure 3-4. The table consists of an axis of code phases and an axis of Doppler frequencies. The PRN Generator will shift its output to give a different time delay in the generated Gold Code; the Carrier Generator will use different Doppler frequencies in generating the local carrier. Once this table is full, a local spike can be detected and the code phase and Doppler frequency associated with that spike will be used in the tracking channel.

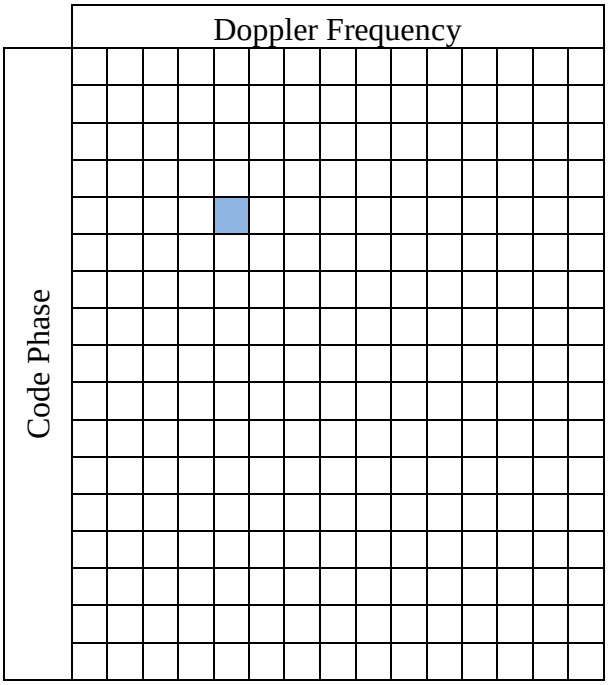

Figure 3-4 : Acquisition Search Table

Due to the autocorrelation properties of the Gold Code, the size of the code phase bins in Figure 3-4 must be at least ½ a chip wide. This ensures that at worst, the peak of the correlation is offset by ¼ of a chip from the theoretical peak, meaning that half the correlation magnitude

still remains. The size of the Doppler bins is driven by the pull-in region of the tracking loops; this is discussed more during the hardware implementation section of this paper.

3.2.1.2 Parallel Code Phase Search Algorithm

Although simple, the Serial Search Algorithm can take a very long time. Consider using a Doppler search span of ±5 kHz, in bins of about 200 Hz. This requires $\frac{10000}{200} + 1 = 51$ Doppler bins to search. Include that with 2046 code phase bins (for ½ chip width bins) and the total size of the Acquisition Search Table is 104346 elements. Some architectures such as FPGAs allow for some parallelization of this process, where multiple correlators can be set up as in Figure 3-3 to process multiple code phases or Doppler frequencies at once. However, there are digital signal processing techniques that can be utilized to provide even greater processing gains.

It is well known that the Discrete Fourier Transform (DFT) can be used to compute circular convolution of two signals [24]. This is a very useful tool, however in GPS the goal is for circular cross-correlation rather than convolution [25]. In [2], it is shown that to perform a circular cross-correlation of two signals, one must simply take the complex conjugation of one of the DFT outputs prior to the multiplication in the frequency domain.

The benefit to the Parallel Code Phase Algorithm is that each iteration of the algorithm is able to produce the correlation value for an entire set of code phases. This allows one complete column of Figure 3-4 to be searched with one step; in addition, the bin sizes will be based on the size of the DFTs, which could accomplish high accuracy results without extensive processing. The overall structure for the Parallel Code Phase Algorithm is shown in Figure 3-5. This method must still search over all of the Doppler bins as described in the Serial Search, so the process begins with mixing the incoming signal with an In-phase and Quadrature estimation of the

carrier. The mixing results are combined and used in a DFT. The replica C/A code is also processed through a DFT and then the complex conjugate of the result is multiplied with the frequency components of the carrier portion. The inverse DFT is taken of the product, and the magnitude is squared to generate the output.

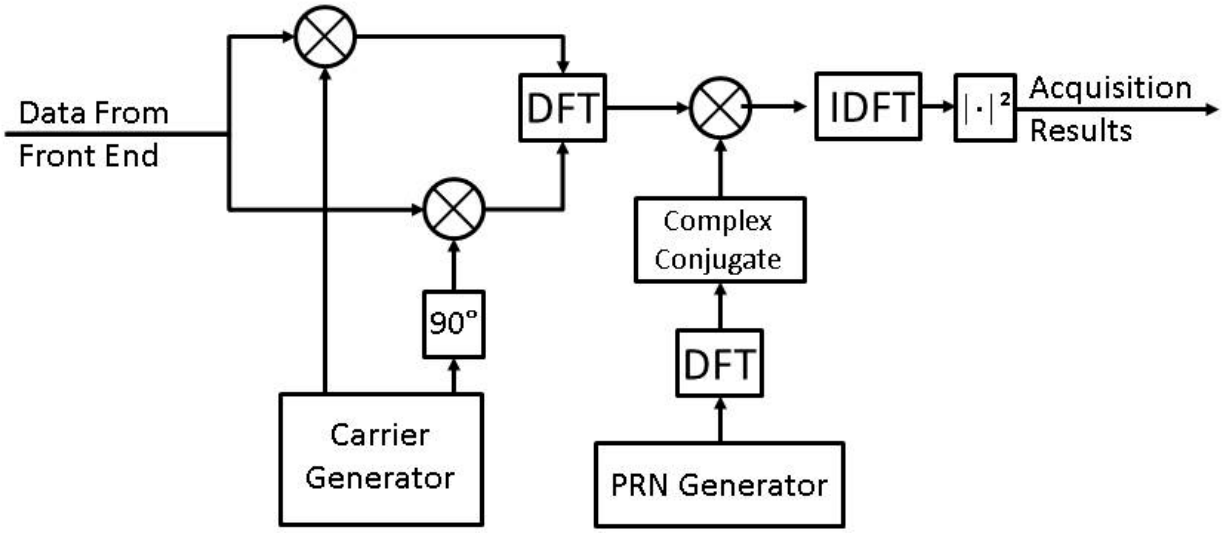

Figure 3-5 : Parallel Code Phase Search

The algorithm for the Parallel Code Phase Search lends itself to be processed much more quickly than the Serial Search, if simply for the fact that the algorithm only needs to repeat over the Doppler search bins. This reduces the number of searches from 104346 to 51. Moreover, the development of Fast Fourier Transforms (FFT) as a means of quickly producing DFT results enables this process to be completed even faster. Fortunately, FFTs come easily packaged for FPGAs and can even be clocked fast enough to produce very low latencies. This topic is explored more in Section 4.3.1.

3.2.2 Tracking

Once Acquisition has finished the search table for a satellite, it will pass the satellite number, code phase estimate, and Doppler frequency estimate to a tracking loop. The tracking process will bring these into more accurate estimates and continue to update these estimates by correlating the incoming signal with a locally generated replica. The correlation outputs are used in feedback loops to determine any adjustments that need to be made to the replica signal to maintain lock on the incoming signal. In each tracking loop, the code phase and Doppler frequency are tracked in two separate feedback loops that occur in parallel. The code phase feedback loop is called a delay-lock-loop and the carrier feedback loop is either a frequency-lock-loop or phase-lock-loop.

3.2.2.1 Delay-Lock-Loop

In its basic form, a DLL is simply a feedback control loop: the controller is the Numerically Controlled Oscillator (NCO), the plant is a bank of correlators, and the sensor is the discriminator. A block diagram of a DLL is shown in Figure 3-6.

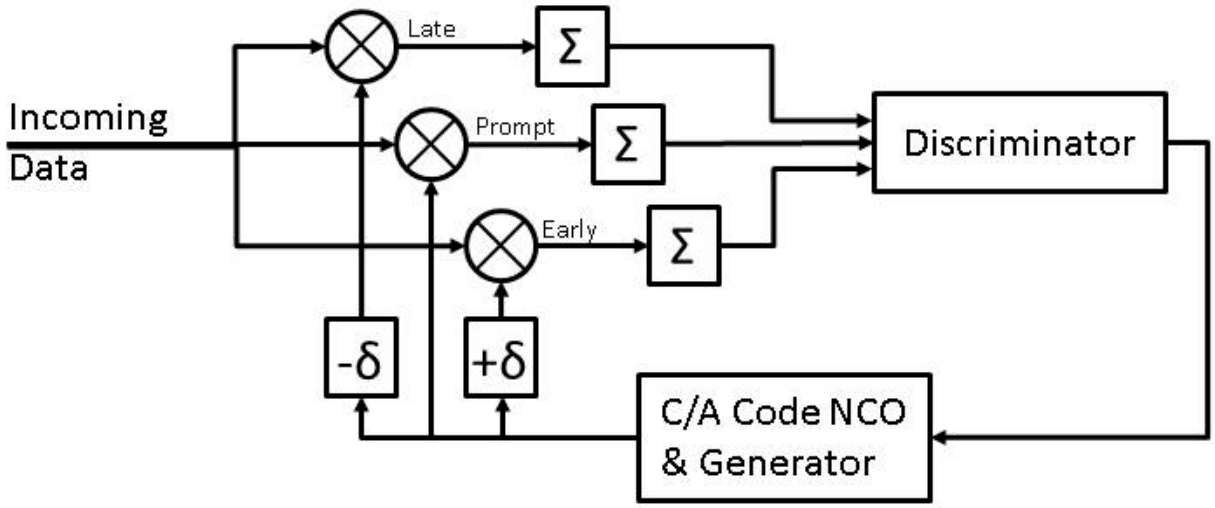

Figure 3-6 : Delay-Lock-Loop Block Diagram

The incoming signal is split into three branches: Early, Prompt, and Late. Each branch multiplies the incoming signal with a time-shifted version of the C/A code. The Prompt branch is being tracked such that it is the most closely aligned with the incoming signal. The Early branch is shifted forward in time by δ chips while the Late branch is shifted in the opposite direction by δ chips. A typical value for δ is ½ chips, but this can be adjusted to either increase the accuracy (decrease δ) or increase the bandwidth (increase δ). A time domain explanation of the signal in the three branches is shown in Figure 3-7.

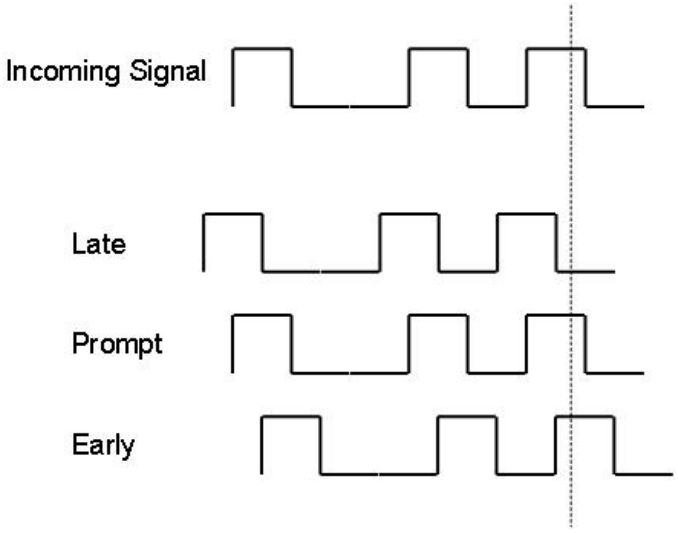

Figure 3-7 : Early-Prompt-Late Alignment

After the incoming signal is multiplied by the Early, Prompt, and Late replicas, each branch accumulates samples for a given amount of time. Typically, 1 millisecond is used for the accumulation period (Integration period) because the C/A Code repeats every millisecond. The Integration period, however, can be increased for greater accuracy.

The Discriminator in Figure 3-6 is used as a means of converting the three correlator outputs into an error between the Prompt replica and incoming signal. There are several different code phase discriminators, which fall into two categories: Coherent and Non-Coherent. A list of common discriminators is shown in Table 3-1. Each discriminator has different pull-in characteristics, meaning that they behave differently to errors between the replicas and the incoming signal. This work implements a Normalized Early Minus Late Power discriminator because it has favorable response to code phase errors greater than ½ chips [2].

Table 3-1 : Common DLL Discriminators

Type	Discriminator Equation	Name
Coherent	$I_E - I_L$	Early Minus Late
Non-Coherent	$(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)$	Early Minus Late Power
Non-Coherent	$\frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{(I_E^2 + Q_E^2) + (I_L^2 + Q_L^2)}$	Normalized Early Minus Late Power
Non-Coherent	$I_P(I_E - I_L) + Q_P(Q_E - Q_L)$	Dot Product

The output of the discriminator is often filtered using a 2nd order low-pass filter. If the discriminator is normalized, then this output becomes an adjustment to the NCO driving the generation of the C/A Code. In its simplest form, the discriminator will tend to slow down the NCO if the Early branch is greater than the Late branch as in Figure 3-8.

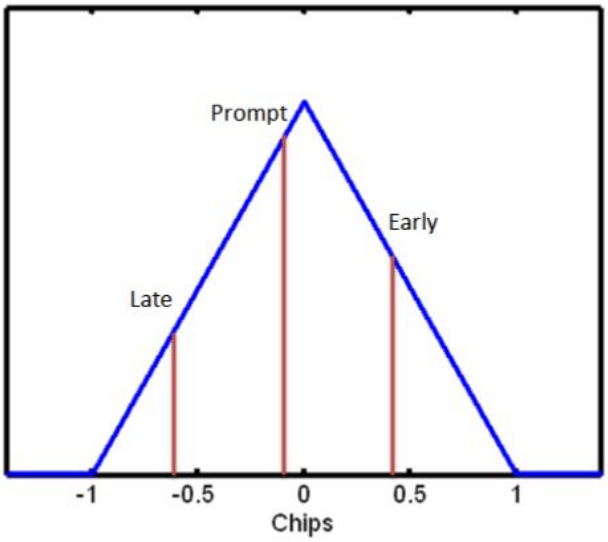

Figure 3-8 : Ideal Autocorrelation Function with Early, Prompt, and Late Outputs [8]

3.2.2.2 Carrier Tracking Loop

Just as a DLL fine tunes the code phase of the incoming signal, the Carrier Tracking Loop (CTL) fine tunes the carrier replica to match the incoming carrier in both phase and frequency. The CTL follows a similar structure to that of the DLL, in particular to its resemblance of a feedback control loop. For the moment, it will be assumed that the C/A code is perfectly matched and has already been wiped from the incoming signal, leaving only the IF carrier wave. This is the point where Figure 3-9 begins.

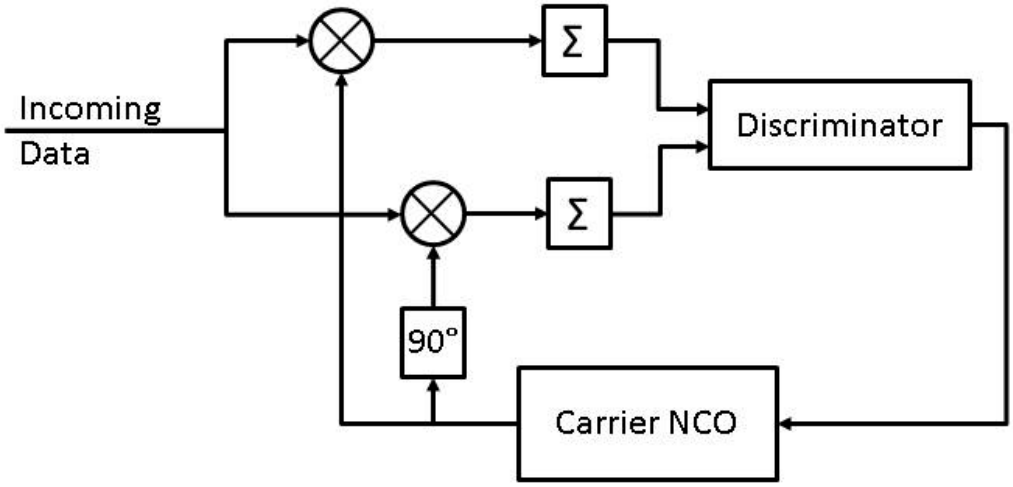

Figure 3-9 : Carrier Tracking Loop

The reference carrier is split into two branches: In-phase (I) and Quadrature (Q). The I-branch is mixed with the sine of the output from the NCO while the Q-branch is mixed with the cosine of the output. Recall from (3.1) that when two sinusoids are mixed, the output is the sine of the sum and difference of their phase arguments. Each branch accumulates over an integration period (typically 1 millisecond), which effectively eliminates the high frequency component of

the mixing process. This output is then passed through a discriminator; the discriminator value is normally filtered and applied as an adjustment to the Carrier NCO.

Typically, the CTL will be described as either a Frequency-Lock-Loop (FLL) or a Phase-Lock-Loop (PLL). As the names imply, an FLL locks to the incoming signal's frequency and a PLL locks to the incoming signal's phase. FLLs tend to have a greater pull-in range, thus allowing the Doppler estimate from the acquisition process to be less accurate [26]. However, since FLLs don't match phase with the incoming signal, the I- and Q-branches do not become distinguishable and the navigation data bits cannot be recovered from an FLL. In this work, the tracking loops begin with an FLL and once locked in frequency will shift to a PLL. Aside from the difference in discriminators, the two loops have the same structure. The discriminators for the FLL and PLL are shown in Table 3-2. The subscript p indicates that the value is taken from the Prompt branch correlator output, and the subscript $p1$ indicates the first of two successive integration periods.

Table 3-2 : CTL Discriminators

Name	Discriminator Equation
Phase Discriminator (PLL)	$\text{atan}\left(\frac{Q_P}{I_P}\right)$
Phase-Difference Discriminator (FLL)	$\frac{\text{atan}\left(\frac{Q_{P1}}{I_{P1}}\right) - \text{atan}\left(\frac{Q_{P2}}{I_{P2}}\right)}{Integration\ Period\ (s)}$

3.2.2.3 Complete Tracking Loop

The complete tracking loop combines both the DLL and the CTL as shown in Figure 3-10. The Prompt branch of the DLL encompasses the CTL, allowing both tracking loops to

operate simultaneously. Also notice that the DLL now utilizes two sets of the Early, Prompt, and Late branches – one for the I-branch and one for the Q-branch.

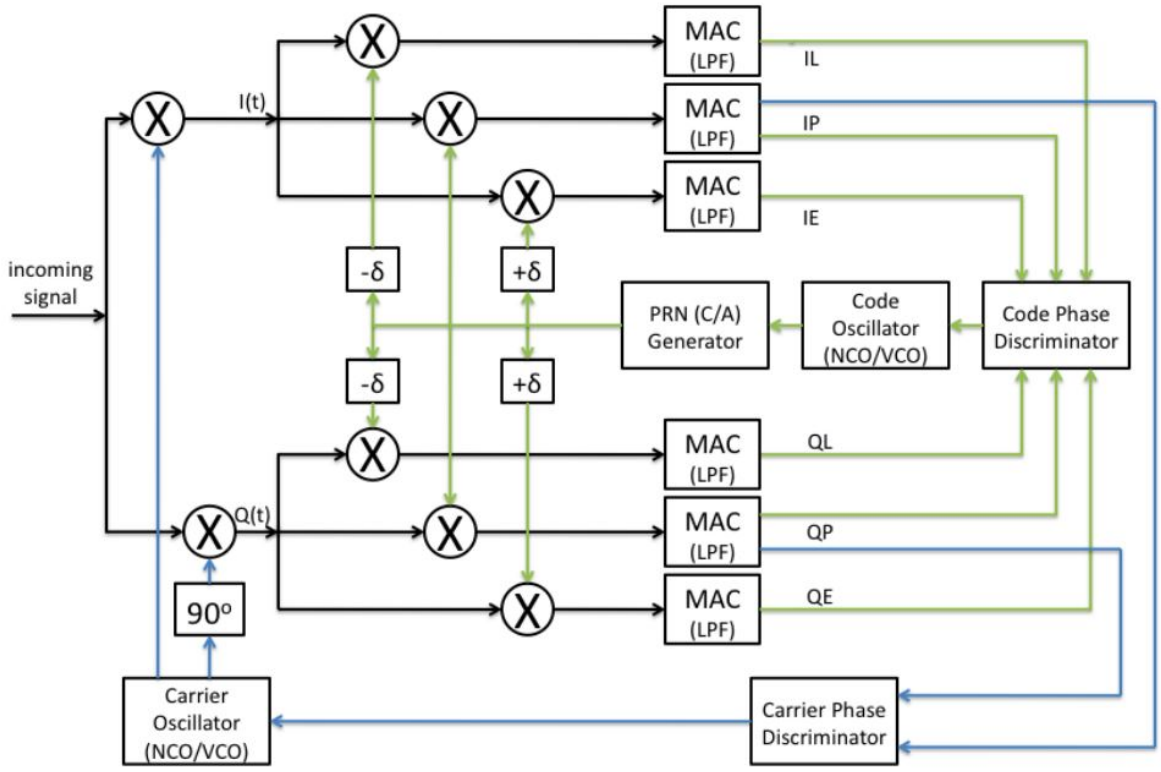

Figure 3-10 : Complete GPS Tracking Loop [8]

3.2.2.4 Ranging Information

One of the main purposes of the tracking loops is to obtain ranging information. From each tracking loop that is locked in both code phase and carrier phase, the receiver can obtain estimates of range to the satellite and relative velocity with respect to the satellite. The relative velocity between the satellite and user is directly related to the Doppler frequency of the incoming signal. This measurement is easily obtained by using the frequency command of the Carrier NCO and relating it to the change in range by (3.2).

$$f_{DOPPLER} = (f_R - f_T) = -\frac{\dot{r}}{\lambda} \tag{3.2}$$

Where f_R and f_T are the received and transmitted frequencies, respectively, λ is the wavelength, and $\dot{r}$ is the change in range (range rate). Because the measurement (Doppler frequency) is actually biased by the receiver clock drift, the measured value is termed the pseudorange rate [1].

The range measurement is effectively as simple: measure the time between when a signal is sent and when it is received, then multiply it by the speed of light as in (3.3).

$$\rho = (t_R - t_T)c \tag{3.3}$$

Where t_R and t_T are the received and transmitted times, respectively, c is the speed of light, and ρ is the pseudorange. As with the range rate, because the receiver time is biased the actual measurement is termed the pseudorange. Because of the highly accurate and synchronized characteristics of the GPS signals, every chip that is transmitted contains precise information about the time it was sent. Until the navigation data bits have been decoded and GPS Time established, this timing information is ambiguous. However, once the receiver has knowledge of GPS time, it needs only to keep track of where in the C/A Code the tracking loop is to know what the transmitted time is (and can use the GPS time for the receive time).

3.2.3 Navigation Solution

The Navigation Solution is the main purpose of the GPS receiver; here, the user's position and velocity are determined, as well as GPS Time. In order for this to occur, the receiver

needs to know the orbital parameters of the satellites being tracked in addition to the pseudorange measurements (for position determination) and the pseudorange rate measurements (for velocity determination).

The orbital parameters for the satellites come from the Ephemeris of the navigation data bits. The navigation data is received from the In-phase branch of the tracking loops and decoded in the receiver to give position and velocity of the satellite. The navigation data also provides GPS Time at the beginning of each subframe. Thus from the navigation data and the tracking loop's knowledge of the C/A Code phase, the satellite's position and velocity are known as well as the time at which the signal was transmitted.

In order to get pseudorange measurements, the receiver must know the time of the received signal. Until a navigation solution is computed, however, the receiver does not yet know GPS Time (unless that information is provided from an outside source; a stand-alone receiver is assumed here). What is known, though, is the relative receive time of multiple tracking loops. When the receiver detects the beginning of a subframe from one of the tracking loops, it can keep track of how many clock cycles occur between then and when it detects the beginning of the same subframe on the other tracking loops. Because the satellites are synchronized and the beginning of the subframe was sent at the same time from all of the satellites, a relative pseudorange can be found for all of the satellites. Given this, the pseudoranges can be approximated by assigning them values between 65 and 83 milliseconds (standard bounds for pseudoranges, in time, for a receiver on the Earth's surface) while maintaining their relative differences [2]. Using these estimates of the pseudoranges and the satellite orbital parameters, a least squares estimate of the user position can be calculated. As

described in the following section, this will also correct the receiver's time, such that the next pseudorange measurements will be correct.

3.2.3.1 Position Estimation

A description of the method used for estimation of the receiver's position begins with the definition of the pseudoranges given in (3.4).

$$\rho_c^{(k)} = r^{(k)} + c * \delta t_u + \tilde{\varepsilon}_\rho^{(k)} \tag{3.4}$$

This simplification assumes corrections due to atmospheric errors and satellite clock errors have already been applied. The atmospheric errors are due to changes in the signal speed as it passes through the ionosphere and troposphere; these errors can be estimated with simple models or received from an outside source. The satellite clock errors are very small timing errors in the satellite clocks; the parameters describing these errors are broadcast in the navigation data. $\rho_c^{(k)}$ is this "corrected" pseudorange; $r^{(k)}$ is the true range; c is the speed of light; δt_u is the receiver's clock bias; and $\tilde{\varepsilon}_\rho^{(k)}$ is the combined effect of any unaccounted for residual errors. The superscript (k) denotes that the value corresponds to the k^{th} satellite.

Let the receiver's position be denoted as $\boldsymbol{x} = (x_u, y_u, z_u)$ and the position of the k^{th} satellite as $\boldsymbol{x^{(k)}} = (x^{(k)}, y^{(k)}, z^{(k)})$. The true range can be described as:

$$r^{(k)} = \sqrt{(x^{(k)} - x_u)^2 + (y^{(k)} - y_u)^2 + (z^{(k)} - z_u)^2} = \left\| \boldsymbol{x^{(k)}} - \boldsymbol{x} \right\| \tag{3.5}$$

Substituting b for $c * \delta_u$, (3.4) can be rewritten as:

$$\rho_c^{(k)} = \left\| \boldsymbol{x}^{(k)} - \boldsymbol{x} \right\| + b + \tilde{\varepsilon}_\rho^{(k)} \tag{3.6}$$

In the pseudorange measurement equation, the satellite positions are known from the ephemeris data, the pseudorange measurement is known from the tracking loop (or for the first calculation, estimated), and the error term is assumed to be zero-mean. That leaves four unknowns: the receiver clock bias b, and the three coordinate values describing the receiver position $\boldsymbol{x}$. These four unknowns are the same in all of the pseudorange measurement equations, regardless of which satellite (notice that in (3.6) none of the unknowns have a superscript). Thus, the receiver's position can be determined if at least four satellites are available.

If only four pseudorange measurements are available, then the four unknowns can be solved for easily. However, more satellites may be available for measurements (in fact, it is desired that there are more than four). If more than four pseudorange measurements are available, then the system of pseudorange measurement equations is over determined. In a situation such as this, a popular estimation technique called Least Squares can be used to solve for the receiver's position and clock bias [1].

Generally, this process begins with an initial guess for the receiver position and clock bias: $\boldsymbol{x_0} = (x_0, y_0, z_0)$ and b_0. Given the initial guess, an approximation for the pseudorange ($\rho_0^{(k)}$) can be derived

$$\rho_0^{(k)} = \left\| \boldsymbol{x}^{(k)} - \boldsymbol{x_0} \right\| + b_0 \tag{3.7}$$

Let the true receiver position and clock bias be $\boldsymbol{x} = \boldsymbol{x_0} + \boldsymbol{\delta x}$ and $b = b_0 + \delta b$, where $\boldsymbol{\delta x}$ and δb are unknown corrections that need to be applied to the initial estimates to get the true values. A system of linear equations can be developed which allow $\boldsymbol{\delta x}$ and δb to be solved for

$$\begin{aligned}\delta\rho^{(k)} &= \rho_c^{(k)} - \rho_0^{(k)} \\ &= \left\|\boldsymbol{x}^{(\boldsymbol{k})} - \boldsymbol{x_0} - \boldsymbol{\delta x}\right\| - \left\|\boldsymbol{x}^{(\boldsymbol{k})} - \boldsymbol{x_0}\right\| + (b - b_0) \\ &\approx \frac{(\boldsymbol{x}^{(\boldsymbol{k})} - \boldsymbol{x_0})}{\|\boldsymbol{x}^{(\boldsymbol{k})} - \boldsymbol{x_0}\|} \cdot \boldsymbol{\delta x} + \delta b \\ &= -\mathbf{1}^{(\boldsymbol{k})} \cdot \boldsymbol{\delta x} + \delta b\end{aligned} \tag{3.8}$$

In (3.8), $-\mathbf{1}^{(\boldsymbol{k})}$ represents a line of sight vector from the initial estimate of the receiver's position to the k^{th} satellite, and $\boldsymbol{a} \cdot \boldsymbol{b}$ represents the dot product of two vectors.

Given a set of pseudorange measurements from K satellites, the set of linear equations becomes

$$\boldsymbol{\delta\rho} = \begin{bmatrix}\delta\rho^{(1)} \\ \delta\rho^{(2)} \\ \vdots \\ \delta\rho^{(K)}\end{bmatrix} = \begin{bmatrix}\left(-\mathbf{1}^{(1)}\right)^T & 1 \\ \left(-\mathbf{1}^{(2)}\right)^T & 1 \\ \vdots & \\ \left(-\mathbf{1}^{(K)}\right)^T & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{\delta x} \\ \delta b\end{bmatrix} \tag{3.9}$$

The matrix containing the line-of-sight vectors is termed the geometry matrix and if often simply noted as $\boldsymbol{G}$. The solution to the system of equations is given in

$$\begin{bmatrix}\boldsymbol{\delta x} \\ \delta b\end{bmatrix} = (\boldsymbol{G}^T\boldsymbol{G})^{-1}\boldsymbol{G}^T\boldsymbol{\delta\rho} \tag{3.10}$$

The corrections can then be added to the initial estimates

$$\begin{aligned}\boldsymbol{x_1} &= \boldsymbol{x_0} + \boldsymbol{\delta x} \\ b_1 &= b_0 + \delta b\end{aligned} \tag{3.11}$$

This can be repeated until the corrections that are calculated become very small. This method of calculating corrections based on previous estimates is known as the Newton-Raphson Method. One small draw back to this method is that it has the possibility of converging on an

incorrect solution. This can be handled well if the initial estimate is fairly well known. Fortunately, for a GPS receiver located on the Earth's surface, an initial estimate at the center of the Earth will allow the Newton-Raphson Method to converge on the correct solution.

3.3 Vector Receiver

Vector tracking receivers have been studied in the GNSS community for decades. Research has shown that receivers have improved immunity to noise and jamming, as well as the benefit of instantaneous reacquisition of satellites after temporary loss of line-of-sight [3] [27] [28] [29]. At its heart, a vector tracking receiver utilizes a Kalman filter to estimate the navigation solution. This navigation solution then becomes the basis for updating the tracking loops. This method allows the inherent cross correlation between tracking loops to be exploited. Because the Kalman filter can adapt the influence each tracking loop has on the navigation solution, stronger satellite signals can aid in the tracking of weaker satellite signals.

As described in the previous section, traditional receivers (termed scalar tracking receivers from hence forth) have a tracking loop that updates a local signal replica in a single feedback loop (sometime termed channel). Each channel tracks one satellite and only interacts with the navigation solution process by supplying pseudorange and pseudorange rate measurements. This structure is shown in Figure 3-11.

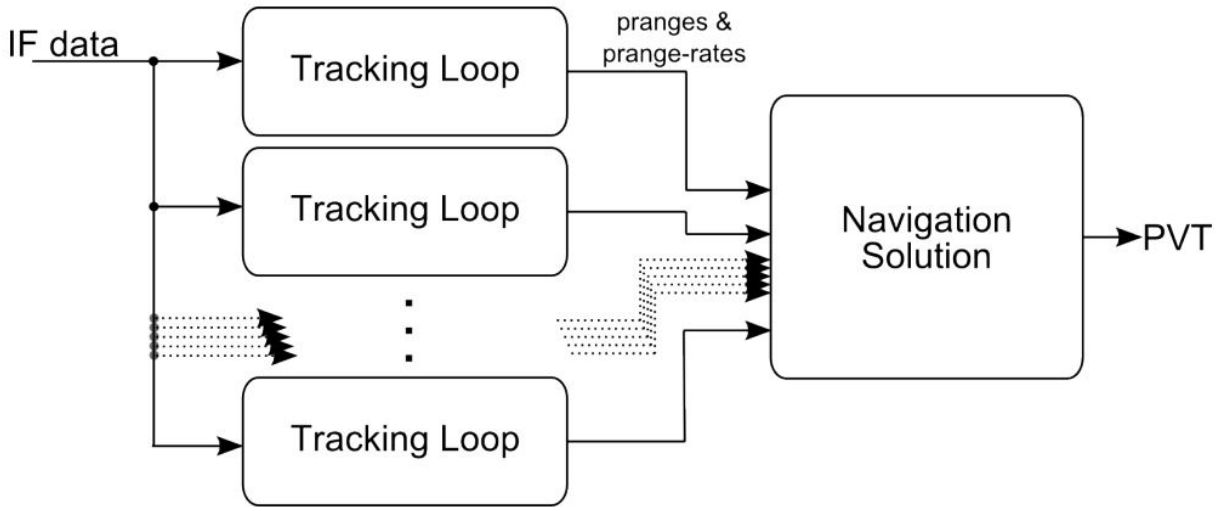

Figure 3-11 : Scalar Tracking Block Diagram

Vector tracking receivers take the diagram in Figure 3-11(where each Tracking Loop block has an individual feedback loop) and convert it into a “shared” feedback loop as in Figure 3-12. The output of the navigation solution is fed back into all of the tracking loops to aid in the estimation of the code and carrier NCO frequencies. Note that now the “tracking loops” are no longer using their internal feedback loops and now only consist of the signal correlation and discriminators. Also, the discriminators (which are described in more detail in Section 4.5) are slightly different such that the outputs of the channels are now errors in the pseudoranges and pseudorange rates.

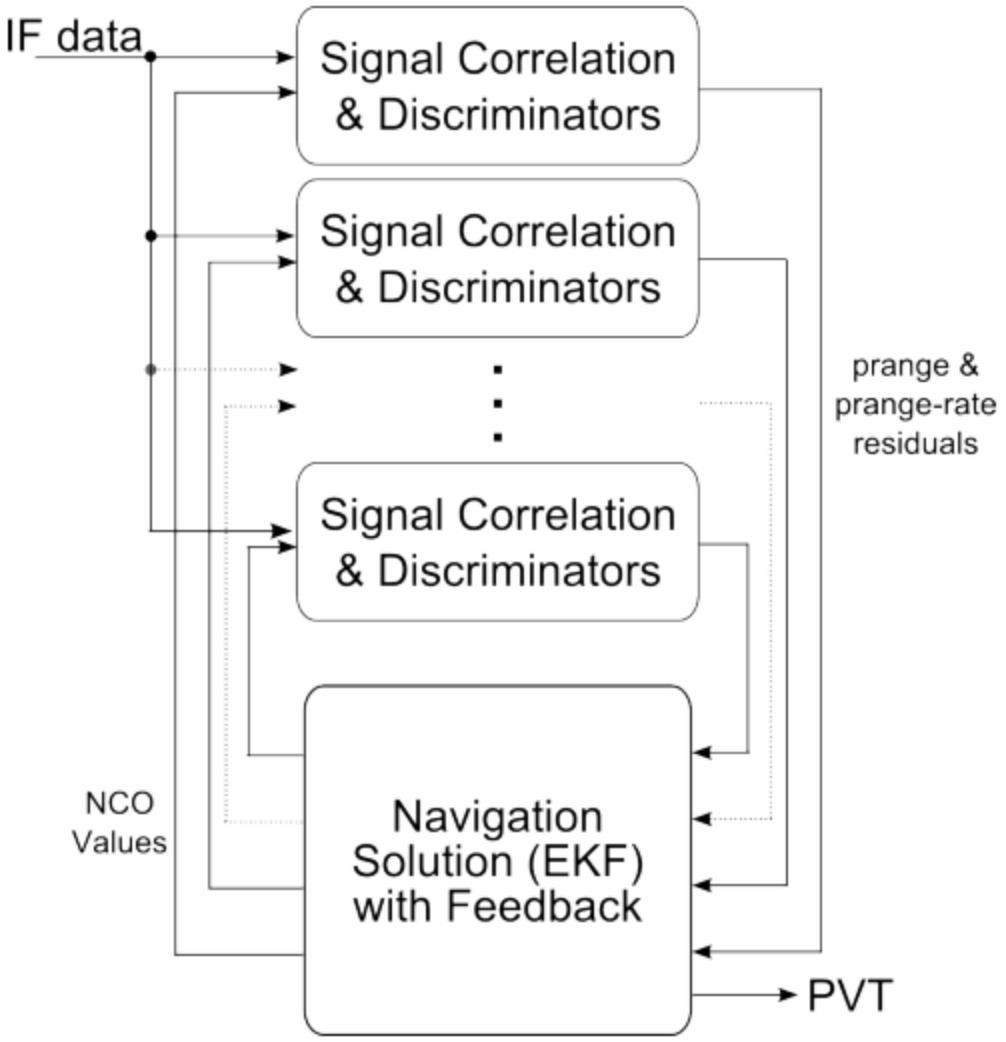

Figure 3-12 : Vector Tracking Block Diagram

3.3.1 Algorithms

The vector tracking approach used in this research is known as a vector delay frequency lock loop (VDFLL). The navigation solution uses a single, centralized Extended Kalman filter to estimate the code phase and carrier frequency for each satellite being tracked. In this work, the states of the Kalman filter are the user's position, velocity, clock bias, and clock bias rate (clock drift)

$$\overline{\boldsymbol{x}} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \\ z \\ \dot{z} \\ b \\ \dot{b} \end{bmatrix} \tag{3.12}$$

The states are referenced in the Earth-Centered Earth-Fixed (ECEF) coordinate frame. The discretized time updates for the Kalman filter are described in detail in [30] and presented in (3.13).

$$\overline{\boldsymbol{x}}_{k+1} = A_d \overline{\boldsymbol{x}}_{\boldsymbol{k}} + Q_d \tag{3.13}$$

Where

$$A_d = \begin{bmatrix} \boldsymbol{\alpha} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} \\ \mathbf{0_{2x2}} & \boldsymbol{\alpha} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} \\ \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \boldsymbol{\alpha} & \mathbf{0_{2x2}} \\ \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \boldsymbol{\alpha} \end{bmatrix} \tag{3.13a}$$

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \tag{3.13b}$$

$$Q_d = \begin{bmatrix} \boldsymbol{Q_x} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} \\ \mathbf{0_{2x2}} & \boldsymbol{Q_y} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} \\ \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \boldsymbol{Q_z} & \mathbf{0_{2x2}} \\ \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \mathbf{0_{2x2}} & \boldsymbol{Q_c} \end{bmatrix} \tag{3.13c}$$

$$Q_x = \begin{bmatrix} \sigma_x^2 \frac{T^3}{3} & \sigma_x^2 \frac{T^2}{2} \\ \sigma_x^2 \frac{T^2}{2} & \sigma_x^2 T \end{bmatrix} \tag{3.13d}$$

$$Q_y = \begin{bmatrix} \sigma_y^2 \frac{T^3}{3} & \sigma_y^2 \frac{T^2}{2} \\ \sigma_y^2 \frac{T^2}{2} & \sigma_y^2 T \end{bmatrix} \tag{3.13e}$$

$$Q_z = \begin{bmatrix} \sigma_z^2 \frac{T^3}{3} & \sigma_z^2 \frac{T^2}{2} \\ \sigma_z^2 \frac{T^2}{2} & \sigma_z^2 T \end{bmatrix} \tag{3.13f}$$

$$Q_c = \begin{bmatrix} \sigma_b^2 T + \sigma_r^2 \frac{T^3}{3} & \sigma_r^2 \frac{T^2}{2} \\ \sigma_r^2 \frac{T^2}{2} & \sigma_r^2 T \end{bmatrix} \tag{3.13g}$$

In (3.13), σ_x^2 is the variance of the state x. This value, along with the variance of y and z, can be tuned based on the expected dynamics of the receiver. σ_b^2 is the variance of the receiver clock bias and σ_r^2 is the variance of the receiver clock drift. In the absence of statistical analysis on the receiver clock, rule-of-thumb values of $c^2 \times 10^{-19}$ m for clock bias variance and $4\pi c^2 \times 10^{-20}$ m/s can be used from [31].

The output equation of the system relates the states to the outputs by a matrix containing vectors of line-of-sight unit vectors to each of the satellites being tracked. Both positions and derivatives (velocities) are related using line-of-sight unit vectors as shown in (3.14) and (3.15).

$$\bar{y}_k = C\bar{\boldsymbol{x}}_{\boldsymbol{k}} + \bar{v} \tag{3.14}$$

$$C = \begin{bmatrix} a_{i,1} & 0 & a_{j,1} & 0 & a_{k,1} & 0 & 1 & 0 \\ 0 & a_{i,1} & 0 & a_{j,1} & 0 & a_{k,1} & 0 & 1 \\ & & & \vdots & & & & \\ a_{i,N} & 0 & a_{j,N} & 0 & a_{k,N} & 0 & 1 & 0 \\ 0 & a_{i,N} & 0 & a_{j,N} & 0 & a_{k,N} & 0 & 1 \end{bmatrix} \tag{3.15}$$

The measurement update in the Kalman filter uses residuals as described in (3.16).

$$\bar{\boldsymbol{x}}_{\boldsymbol{k}}^{(+)} = \bar{\boldsymbol{x}}_{\boldsymbol{k}}^{(-)} + K_k(\bar{y}_k - C\bar{\boldsymbol{x}}_{\boldsymbol{k}}^{(-)}) \tag{3.16}$$

Recall that in Figure 3-12, the inputs to the navigation solution (EKF) are the residuals of the system. Thus (3.16) becomes (3.17) and the measurement update becomes a simple correction to the previous state estimate based on the outputs of the tracking channels.

$$\bar{\boldsymbol{x}}_{\boldsymbol{k}}^{(+)} = \bar{\boldsymbol{x}}_{\boldsymbol{k}}^{(-)} + K_k\varepsilon_k \tag{3.17}$$

3.3.2 Asynchronous vs. Synchronous Measurement Updates

A major issue in the implementation of these algorithms is the timing of the measurement updates [32]. With vector tracking, the measurements come from the correlator outputs in the tracking loops. For a measurement update to contain measurements from all of the channels, the filter must wait for all of the channels to finish an integration period. This must also be the same integration period. Because all of the channels will finish an integration period at different time, the filter may update too late for some channels.

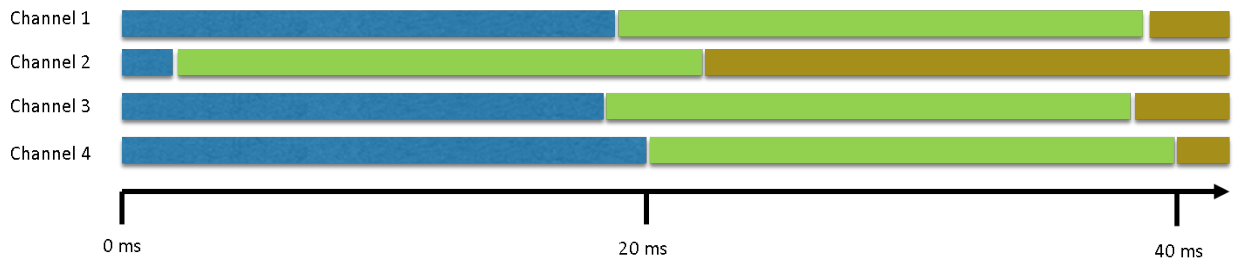

Figure 3-13 : Vector Update Timing Diagram

Figure 3-13 is a timing diagram used to highlight the timing issue associated with the vector measurement updates. Shown are four tracking channels across three different 20 millisecond integration periods. The time axis corresponds to the receiver time; the colors correspond to identical transmit times, such that the beginning of the green section on Channel 1 was transmitted at the exact same time as the beginning of the green section on Channel 2.

There is a very drastic difference between Channel 1 and Channel 2, although this situation is completely possible. It is known that transit time for satellite signals range from 65 to 83 milliseconds. Because this difference is less than 20 milliseconds, it can be assumed that received signals will not overlap by a whole data bit. However, as illustrated above, as little as 2 milliseconds can be left for overlap. Notice how the blue integration period of Channel 4 ends very near to the beginning of the gold integration period of Channel 2. Ideally, the NCO adjustments based on the blue integration period should be made to the green integration period. If the Kalman filter equations can be updated quickly enough, then these updates may be applicable for the green integration period for Channels 1, 3, and 4. However, once the updates have been calculated, the green integration period for Channel 2 will be nearly over. To overcome this issue, the receiver can use the Correlator outputs from the blue integration period and predict the NCO adjustments required for the gold integration period. Thus the NCO

adjustments will be applied one integration period late; however they will be calculated based on a prediction of the values of the Kalman filter states at the point that the brown integration period begins. This measurement update method for the filter is termed Synchronous updating [8].

Another method, called Asynchronous updating, was described in [33] and helps to overcome some of the shortcomings of Synchronous updates. Figure 3-14 will be used as a reference to the timing of these asynchronous updates and how the measurement update needs to be modified to account for this new method. Figure 3-14 is similar to Figure 3-13: the time axis shows the receiver time and four channels are shown in regards to their receive time for the beginning and middle of common integration periods. Note that the receive times for the start/end of integration periods (termed Integrate and Dump in the figure) are staggered due to differing distances to the satellites.

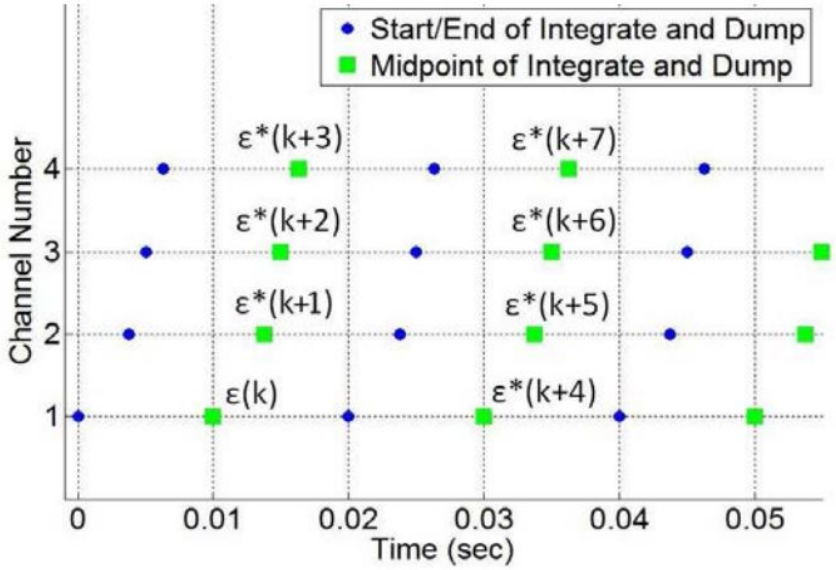

Figure 3-14 : Diagram of Staggered Data Bit Arrival Times [33]

The measurements from the tracking channels, while calculated at the end of an integration period, most closely resemble the state of the vector at the middle of an integration period. For Channel 1, presume that the filter states have previously been propagated to the middle of the integration period of Channel 1 ($\hat{x}^-(k)$) via a time update of the Kalman filter. The measurements collected at the end of this integration period ($\epsilon(k)$) represent the difference between the true state ($x(k)$) and this previously propagated estimate as in (3.18).

$$\epsilon(k) = C(k)[x(k) - \hat{x}^-(k)] + \nu(k) \tag{3.18}$$

Here, no adjustments will be made and the measurement update can proceed as in (3.19).

$$\hat{x}^+(k) = \hat{x}^-(k) + K(k)\epsilon(k) \tag{3.19}$$

For the subsequent channels, the measurement represents the difference between the true state and the estimate of the state from when the NCOs were last updated for that channel; this is different than the last state estimate. For example, the residuals for the second channel need to be:

$$\epsilon(k+1) = C(k+1)[x(k+1) - \hat{x}^-(k+1)] + \nu(k+1) \tag{3.20}$$

However, the state estimate $\hat{x}^-(k+1)$ is the forward propagation of the states after the measurement update from the first channel. When the NCOs for Channel 2 were updated, the state estimate was still $\hat{x}^-(k)$ rather than $\hat{x}^-(k+1)$. The residuals from the second channel are actually:

$$\epsilon^*(k+1) = C(k+1)[x(k+1) - \hat{x}^{*-}(k+1)] + \nu(k+1) \tag{3.21}$$

Here, $\hat{x}^{*-}(k+1)$ is the state estimate when this channel began the last integration period. It can be related to $\hat{x}^-(k+1)$ by (3.22).

$$\hat{x}^{*-}(k+1) = \hat{x}^{-}(k+1) - A(k,k+1)K(k)\epsilon(k) \tag{3.22}$$

Thus, because the states were updated between when the Channel 2 NCOs were updated and the measurements were generated for Channel 2, the raw measurements can be corrected for the Channel 1 measurement update by (3.23)

$$\epsilon(k+1) = \epsilon^{*}(k+1) - C(k+1)A(k,k+1)K(k)\epsilon(k) \tag{3.23}$$

A more general formulation for the adjustment of the residuals is given in [33] and restated below:

$$\epsilon(k+N) = \epsilon^{*}(k+N) - C(k+N)\sum_{m=k}^{k+N-1} A(m,k+N)K(m)\epsilon(m) \tag{3.24}$$

For all updates after the first, the $C(k+N)$ matrix will be needed. This matrix is related to the unit vector from the user to the satellite on channel N. In the asynchronous updates of the vector receiver, this matrix only differs slightly from that described in (3.15)

$$C(k+N) = \begin{bmatrix} u_x & 0 & u_y & 0 & u_z & 0 & 1 & 0 \\ 0 & u_x & 0 & u_y & 0 & u_z & 0 & 1 \end{bmatrix} \tag{3.25}$$

The receiver described in this thesis implements the asynchronous measurement method. This simplifies the timing of the measurement updates and enables the updates to occur more quickly. The goal in using the asynchronous method is to limit the time that the receiver is operating with “old” NCO frequency commands.

Chapter 4

Receiver Implementation

4.1 Receiver Overview

The receiver hardware described previously in Chapter 3 can be simplified as in the diagram in Figure 4-1. The RF Front-End receives an RF signal from the antenna. This signal is amplified, filtered, and shifted in frequency before it is passed to an analog-to-digital converter. These samples, along with the sample clock, are passed to the FPGA for signal processing. After the RF Front-End down-converts and samples the incoming signal, the receiver can be broken down into two processing areas: IF processing and baseband processing.

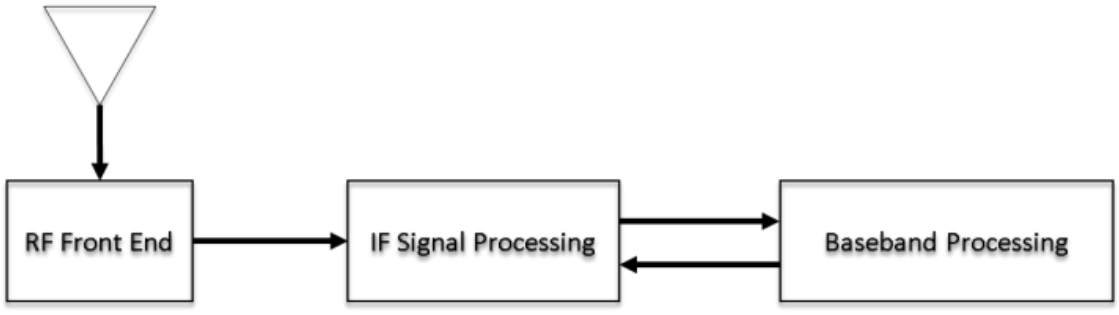

Figure 4-1 : Receiver Hardware Overview

The IF processing includes: generating the local signal replica and correlating it with the incoming signal. Any counters or other modules used for book-keeping are also considered part of the IF processing. This includes: a global counter used to align the incoming data with the receiver's internal time; a code accumulator; and a carrier cycle accumulator in each tracking loop to keep record of carrier cycles. All IF processing must be performed at or above the sampling frequency of the ADC in the front-end to meet real-time requirements. The sampling frequency for the receiver constructed here is 16.368 MHz. In addition to a high data rate for the incoming data, all modules must run in parallel. In commercial receivers, this is often accomplished on application specific integrated circuits (ASICs). The development of ASICs can be expensive and unforgiving, despite showing great performance. For research projects or prototypes however, FPGAs can be used to efficiently perform these high frequency parallel tasks on a reprogrammable platform.

The Baseband processing includes the updates for the tracking loops, as well as the navigation solution calculations. The tracking loop updates operate at frequencies from 50 Hz to 1 kHz. While the baseband processing happens at slower rates than the IF processing, many of the computations done in the baseband processing can be very time intensive and complex. In the vector receiver implemented in this research, the filter updates require matrix multiplications of varying sizes as well as matrix inversions. To handle these requirements in a very flexible fashion, a micro-processor or similar structure can be used.

4.2 Hardware Selection

Other vector tracking receivers have been attempted on FPGAs before [8]. A major drawback to an all-FPGA approach is the need for a processor capable of completing baseband

tasks quickly. Soft-core processors can be implemented on FPGAs which can perform many of these tasks. This can be very helpful as the processor and the FPGA can be incorporated together in a single chip. However, soft-core processors tend to run slower than comparable hard-core processors and also lack acceleration architecture that can increase performance.

In order to continue the use of an FPGA, yet overcome the drawbacks to a soft-core processor, the receiver developed in this thesis is implemented on a Zynq All-Programmable platform. The Zynq is a System-on-Chip (SoC) that incorporates an FPGA with a dual-core ARM processor on a single chip. This allows the FPGA and processor to maintain the same tight coupling seen with soft-core processors while gaining the performance from including a hard-core processor with two cores. As seen in Figure 4-2, the Zynq architecture allows for use with a variety of interfaces and customization.

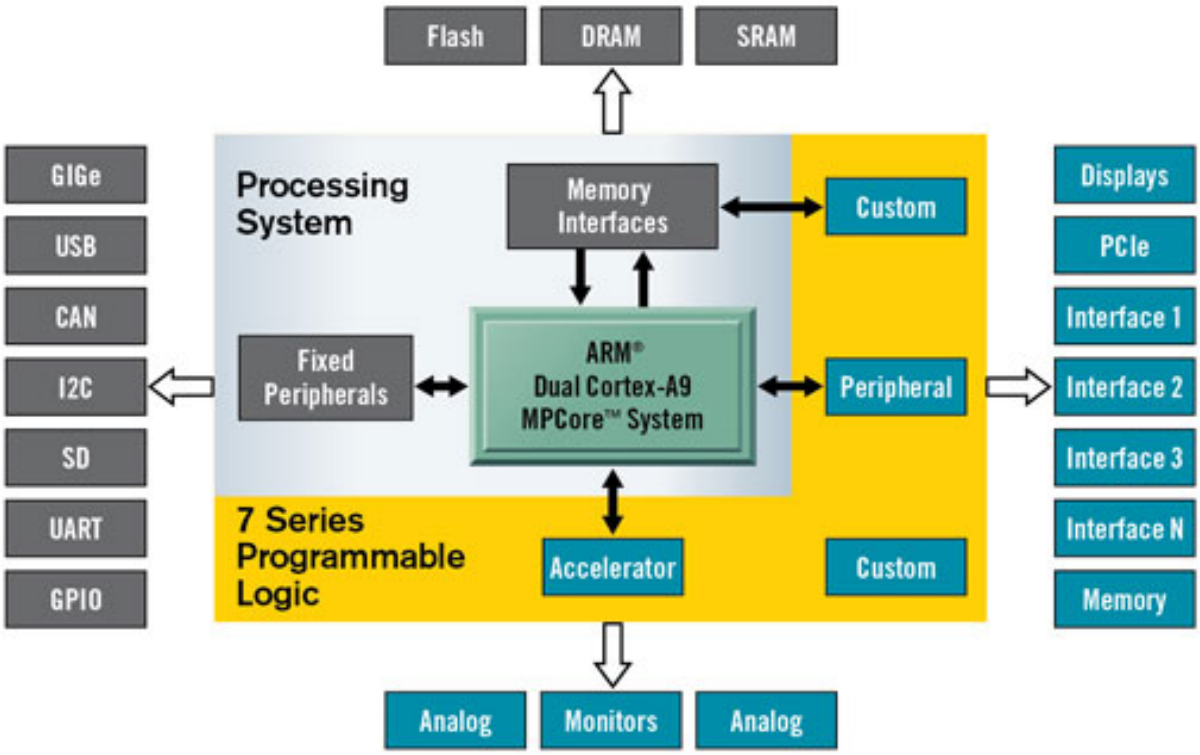

Figure 4-2 : Zynq Architecture Overview [11]

The development board used for this receiver is the Zedboard from AVNET. The Zedboard incorporates a Zynq -7000 SoC XC7Z020-CLG484-1 as well as a variety of peripheral interfaces and memory options. The chip used on the Zedboard comprises an Artix-7 FPGA with 85k logic cells and 220 digital signal processing (DSP) slices. The processor is a dual ARM Cortex –A9 core clocked at 667 MHz and integrating a NEON SIMD engine. At the writing of this thesis, the Zedboard can be purchased for under $400.

While this receiver is fully implementable on the Zedboard, the Zynq platform allows for certain scalability. For more inexpensive options, Zynq chips can be purchased that have only 28k logic cells and 80 DSP slices. On the other hand, for more capabilities (and higher cost)

Zynq chips can be purchased that have 444k logic slices and 2020 DSP slices, as well as processor cores that are clocked at up to 1 GHz.

The RF front end for this receiver is a MAX2769 from Maxim Integrated Circuits. It is a complete front-end and has the capability of being used for GPS, GLONASS, or GALILEO. A block diagram of the front-end can be seen in Figure 4-3. There are also many configurable parameters such as intermediate frequency (IF), filter type and order, automatic gain control (AGC) settings, and sampling frequency [34]. Some of the settings used for this receiver are shown in Table 4-1.

Table 4-1 : RF Front-End Settings

Sampling Frequency	16.368 MHz
Intermediate Frequency	4.092 MHz
IF Filter Bandwidth	2.5 MHz
IF Filter Order	5th

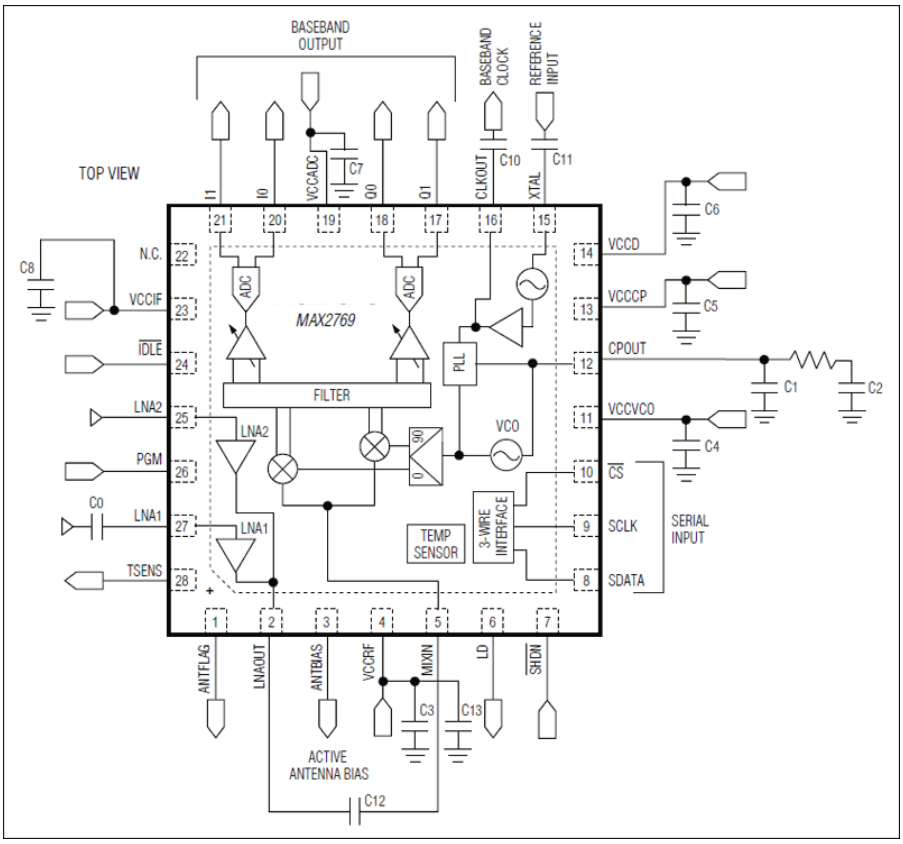

Figure 4-3 : MAX2769 Block Diagram [34]

4.3 Receiver Structure

With an understanding of the processing requirements of the receiver and the hardware used in constructing the receiver, the operations of the receiver can also be broken off into different parts of the hardware. The overall organization of the Zynq is described in Figure 4-4 and will be described in detail throughout this section.

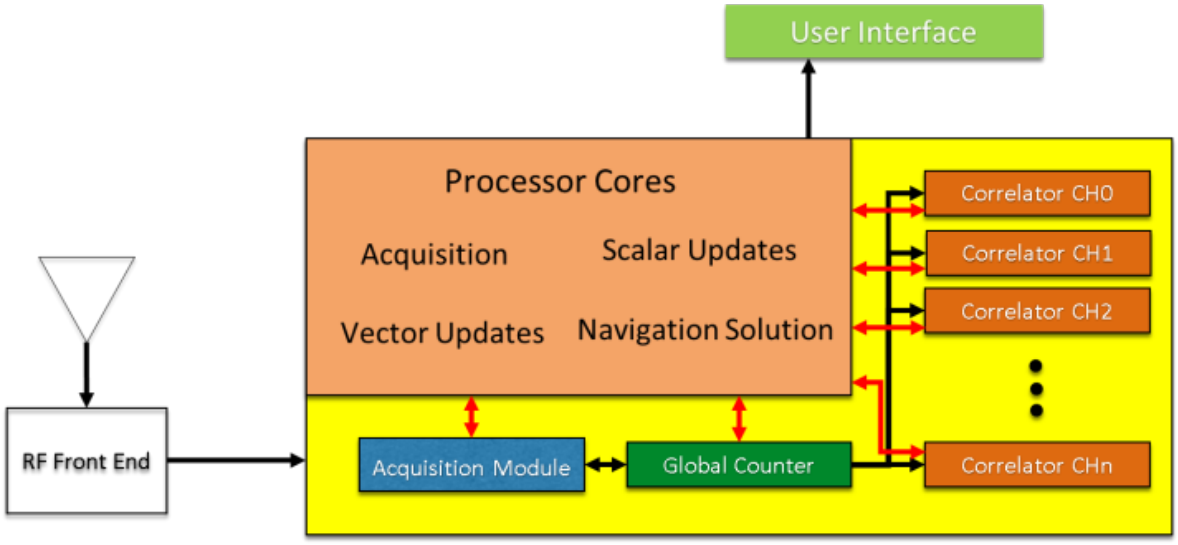

Figure 4-4 : Receiver Structure [32]

The yellow portion of Figure 4-4 indicates the FPGA and the orange indicates the processor. In the FPGA, there are different modules for the different operations of the receiver. The Acquisition module handles the tasks associated with the acquisition algorithm; the Global Counter handles scheduling and synchronization; and the Correlator channels operate as tracking loops. The black arrows in the FPGA indicate signals that pass between modules. Because FPGAs allow for easy routing of signals between these modules, information can be shared between the sections of the FPGA with relative ease. The red arrows indicate communication from each of the modules to the processor. This is an example of one of the major benefits of the system-on-chip architectures: each module on the FPGA holds register space in the memory of the processor. This allows for easy access to these register spaces from either the FPGA or the processor.

The structure shown in Figure 4-4 is made available by incorporating the modules into custom peripherals with processor bus access. The processor bus is adopted from the Advanced eXtensible Interface (AXI) protocol, which is part of a popular family of microcontroller buses called ARM AMBA. By wrapping these modules in VHDL-defined AXI peripherals, the modules can be used on a variety of platforms.

4.3.1 Acquisition Structure

The acquisition structure follows the basis of an acquisition method called Averaging Correlation. Averaging Correlation is a deviation from the Parallel Code Phase Search described in Section 3.2.1.2 that seeks to shrink the size of the FFT without adversely affecting the correlation output [35] [36] [37]. Because FFTs require lengths of power-of-two to operate, several different methods exist to help data accommodate this requirement; zero-padding, pseudorandom-subsampling, and averaging correlation are just three examples of some commonly used methods for FPGA implementation [38]. Averaging correlation is used in this work due to its simplicity and effectiveness in reducing the size of the FFT and data storage needs without compromising detection.

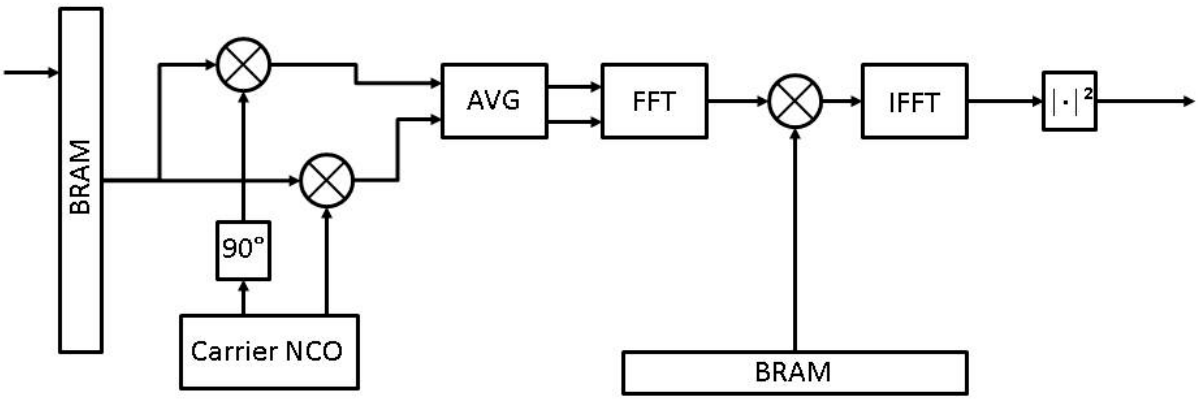

Figure 4-5 : Averaging Correlation Acquisition Block Diagram

A block diagram of the Acquisition Structure is shown in Figure 4-5. The averaging correlation implementation in this work records 16384 samples of incoming data into BRAM and operates on this data throughout the search for a given satellite. This data is mixed with a local carrier replica then down-sampled through averaging to a length of 1024. This averaged data is then fed as the input to one of two FFTs (one is a forward FFT, the other an inverse FFT). The C/A Code in this implementation has been preprocessed and stored in BRAM. Because the FFT length is only 1024, if both the real and imaginary components of the C/A Code's frequency content are held in 4 bit values then all 32 satellites can be stored in one of the 36 kB BRAM. The output of this BRAM is the complex conjugate of the FFT of the C/A Code and can be directly multiplied with the FFT of the incoming signal. This product is then fed to the IFFT and the magnitude of the result is squared and passed to the processor for storage.

The major caveat to the averaging correlation is deciding which data samples to average over. For 16.368 MHz, approximately every 16 samples will be averaged into 1 sample. If the true beginning of the PRN sequence in the received signal happens to coincide with the 15^{th} sample stored, then the averaging effect can actually prevent acquisition of the signal. To

alleviate this issue, the process is repeated for different starting points for the averaging. Notice that 16 extra samples are stored in BRAM (16384 as opposed to 16368). This allows the process to begin averaging as late as the 15^{th} sample and still have enough data to finish averaging. To be as accurate as possible, 16 different starting points should be used. Even if this process is repeated 16 times, it will still be faster than computing an FFT that is 16 times longer (in terms of data length). However, considering that the Acquisition process only needs to be as accurate as half a chip, this implementation only repeats for every other starting point such that the averaging correlation process is only computed 8 times.

4.3.2 Tracking Structure

The overall structure of the tracking channels is shown in Figure 4-6, which is similar to Figure 3-10 shown previously. However in this section the tracking loop is described in its relevance to implementation in the AXI Bus on the FPGA. Note that everything shown in Figure 4-6 is contained in the FPGA; this structure interacts with the microprocessor through the AXI Interconnect on the right side of the figure.

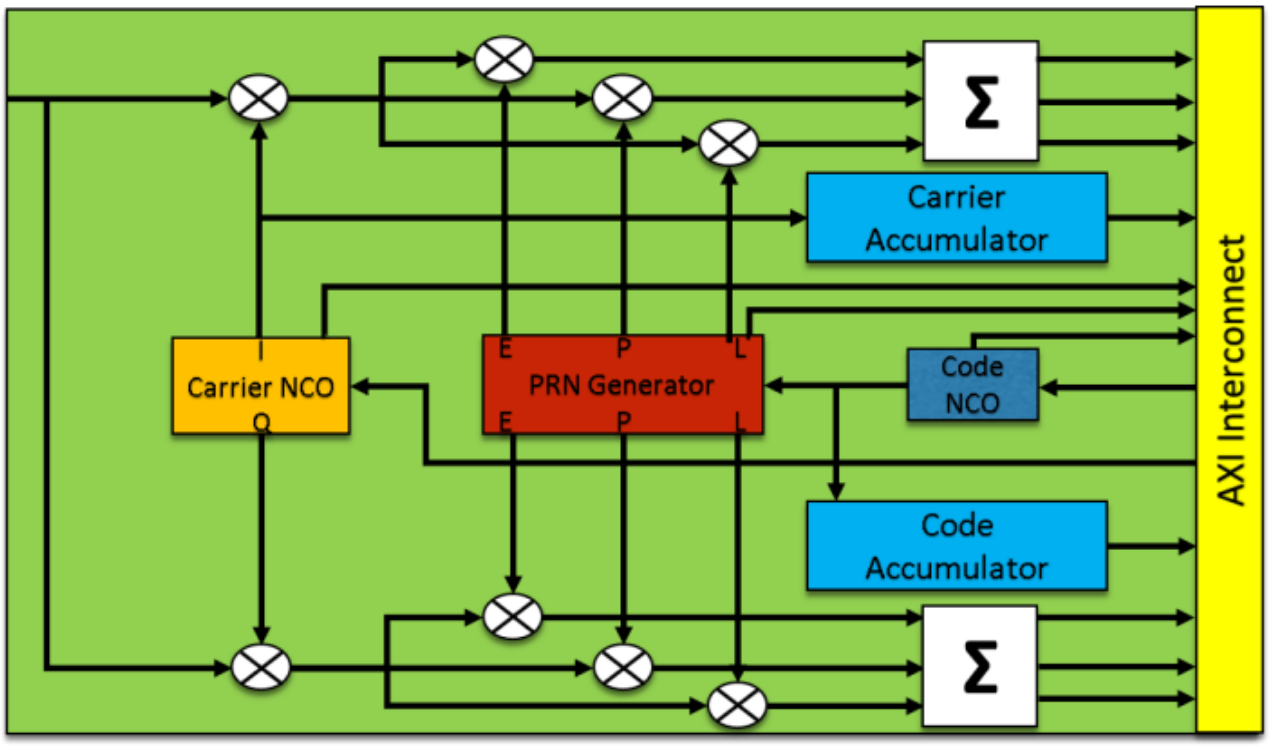

Figure 4-6 : Tracking Channel Module [32]

The incoming signal (arrows on the left-most side) is mixed with a local carrier and code replica. The correlation is completed by summing the output of the mixed signals for a specified integration period. Early, Prompt, and Late code replicas are generated and In-phase and Quadra-phase carrier replicas are generated. The outputs of the module now become the correlation results and accumulator values (along with code phase and carrier phase). These outputs are mapped directly to registers in the processor via the AXI Interconnect. Furthermore, whenever the end of an integration period occurs (for this receiver, 1 millisecond integration periods are used and thus the end of an integration period is defined as the end of a PRN sequence), the output values are latched into the registers and an interrupt signal is sent to the processor's interrupt controller. When the loop updates are completed, the processor writes updated values to

the NCO command registers. These are mapped directly to the code and carrier NCOs such that there is low latency between the processor writing updated values and the NCOs modifying their phase increments.

In addition to tracking the incoming signal, the receiver will include other tracking loops similar to Figure 4-6 but at greater spacings than the whole chip width from Early to Prompt. Recall that when the replica is offset by more than a chip from the incoming signal that the correlation value will be approximately zero. In fact, this value can be taken to be noise in the loop since there should be no correlation between the signals. These tracking loops with greater spacings are called noise correlators since they will be used to get an estimate of the noise in the receiver. For each tracking channel (where there is a structure like Figure 4-6), there will be three more structures similar to Figure 4-6 to "track noise."

4.3.2.1 PRN Generator

The PRN Generator is the key element driving the timing in each tracking channel. Recall from Section 2.5 that the satellites generate the C/A Code from unique combinations of two shift registers. The PRN Generator used in the hardware of this receiver mimics the generation of the C/A Code in the satellites. Figure 4-7 repeats the structure of the PRN Generator. The values in the shift registers are shifted on the rising edge of the Code NCO's output so that the generation of the local replica of the C/A Code is clocked by the Code NCO. A very important aspect of the PRN Generator is that it defines the beginning/end of integration periods in the Tracking Channels. It accomplishes this by triggering a signal when both shift registers are filled with 1's (this only occurs at the beginning of the C/A Code). Thus each

tracking loop has a rising-edge reference to the beginning/end of a PRN sequence to use for processor interrupts (and resets for the accumulators as described below).

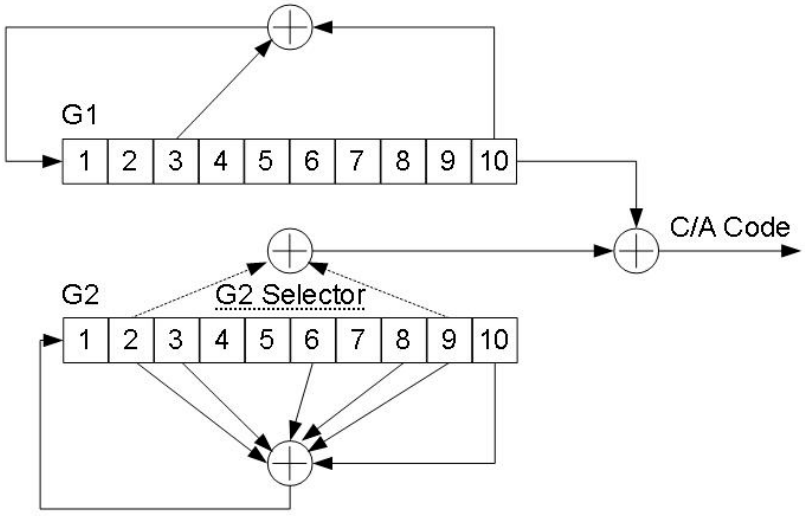

Figure 4-7 : PRN Generator

4.3.2.2 NCOs and Accumulators

It is apparent that the NCOs are pivotal in the generation of the local replica of the satellite signal. Each NCO is effectively given a frequency command; this value is stored in a register and can be changed at any point in time. Once it is changed, the change in frequency can be noticed on the output on the following clock cycle (this enables the processor to update the NCO commands quickly and effectively as soon as it finishes the tracking updates). The NCOs used in this work have three outputs: a sine wave, a cosine wave, and a phase. For the Carrier NCO, the sine wave and cosine wave are used to mix with the incoming signal. For the Code NCO, just the sign of the sine wave is used to drive the PRN Generator. The phase output for each NCO is also used to aid in more accurate ranging measurements.

The Code Accumulator from Figure 4-6 can be viewed as a "chip counter." The processor sets the Code Accumulator to begin synchronized with the beginning of a GPS Subframe. From that point, whenever the Code NCO triggers the PRN Generator the Code Accumulator increments by 1. The Code Accumulator continues to count up until the end of a Subframe (6 seconds = [1023 chips/sec * 1000 msec/sec * 6 sec] = 6,138,000 chips) and then resets. The benefit to this is to enable the receiver to have direct access to the transmit time of the signal at any point in time. As mentioned before, each chip in the received signal contains precise timing information. In fact when the tracking loop is properly aligned with the incoming signal, the local replica is the receiver's best estimate of the transmit time. Therefore, the Code Accumulator value (or the number of chips since the beginning of the last subframe), when combined with the Time of Week (found from the Z-Count in the Ephemeris Data) as in (4.1), gives the transmit time of that particular chip.

$$TransmitTime = TimeOfWeek + CodeAccumulator * \frac{1}{1{,}023{,}000} \tag{4.1}$$

It is possible that the receiver may need to know the transmit time more accurately than the period of a chip. This accuracy can be acheived by reading the phase value from the Code NCO and converting it to seconds using (4.2).

$$Seconds = CodePhase * \frac{1}{2^{NCO\ Accumulator\ Bits}} * \frac{1}{1{,}023{,}000} \tag{4.2}$$

The receiver requests pseudorange measurements from the tracking channels simultaneously. Thus all of the tracking channels are at a different point in their C/A Code due to

different distances traveled for each signal. With the Code Accumulator, the receiver can request these measurements at any point in time. When a measurement request is made, the receiver latches both the Code Accumulator value and the Code NCO phase value from each channel, and writes them to the processor registers. Then, using the receiver's internally kept time, the pseudoranges can calculated using (4.3).

$$\rho^{(k)} = \left(ReceiverTime - TransmitTime^{(k)}\right) * c \tag{4.3}$$

Where $ReceiveTime$ is the receiver's time, $TransmitTime^{(k)}$ is the transmit time of the signal for the k^{th} satellite (calculated using (4.1) and (4.2)), and c is the speed of light.

4.4 Scalar Updates

When the receiver is operating in scalar receiver mode, each tracking channel operates independently of the other tracking channels. At the end of a correlation period for each channel, the processor will use the correlation outputs to calculate new commands for the NCOs. While the end of correlation periods occur relatively slowly, the processor must respond to each channel as quickly as possible to ensure that data is not missed.

For this research, a 1 millisecond integration time is used when the receiver is in scalar mode. The end of an integration period, which is determined by the end of a PRN sequence in the PRN generator, triggers an interrupt in the processor and provides a trigger to latch the values of the correlation outputs. As described above, the processor will access the correlation values from registers that are controlled by the AXI Bus. When the interrupt is triggered in the processor, the update sequence is as shown in Figure 4-8.

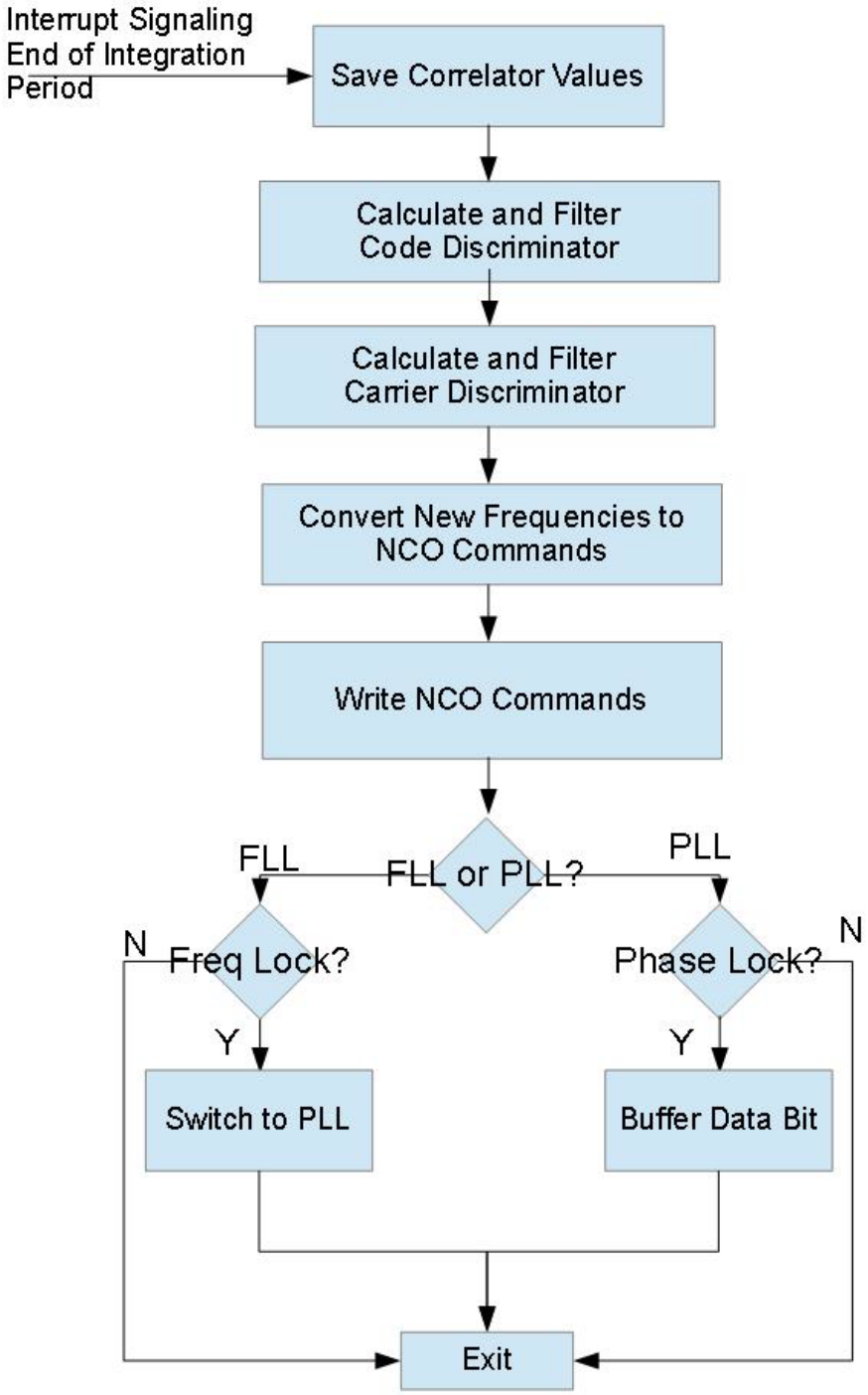

Figure 4-8 : Scalar Updates Flow Graph

The correlator values are retrieved from the AXI Bus and saved in local memory. Only the six primary correlator values are necessary for scalar updates; however the noise correlators are also stored for carrier-to-noise ratio estimation. The code discriminator is then calculated and filtered. A normalized early-minus-late power discriminator is used for the code. The discriminator value is filtered by a 2^{nd} order low pass filter; this value is now a correction to the code NCO frequency. The carrier discriminator is calculated and filtered in a similar way. Initially, the carrier discriminator is a frequency discriminator which causes the carrier tracking loop to operate as an FLL. The new NCO commands are generated from the filtered discriminator outputs and written to the NCO command registers. The update then checks if the FLL is in frequency lock by using a binary up-down counter. Once the FLL has reached frequency lock, the channel will switch to a PLL in order to accurately pull off navigation data bits. The check for phase lock determines when the channel can begin extracting navigation data bits.

As mentioned before, this update must be done as quickly as possible to ensure that no data is lost. It is not generally good practice to do any computations within an interrupt, however in this case the computations are considered to be the greatest priority at the time. Several timing requirements must be taken into consideration when doing this. Complications can arise when several channels have integration periods that end at or near the same time. In this processor, interrupt signals are buffered such that no interrupts are missed; however, a cascading effect can occur. Consequently, for scalar processing updates, a maximum processing time can be determined by the necessity that all channels must be able to be updated within the time of one integration period. For this work, a 1 millisecond integration time and 12 channel receiver are considered to give a scalar update limit of 83 microseconds.

$$Update\ Limit = \frac{Integration\ Period}{\#\ of\ Channels} = \frac{1\ ms}{12\ ch.} = 83\ \mu sec \tag{4.4}$$

While this gives an upper limit to reach real-time operation, other considerations exist. As more updates occur at or around the same time, the update for the last channel can begin a significant amount of time past the end of the integration last period. Because the correlator values are latched at the end of an integration period, this data will not be lost until the end of the next integration period. However, the longer the channel must wait to update the NCO commands, the longer the tracking channel is operating on "old" information. This can lead to negative performance characteristics, particularly in high dynamic operations.

4.5 Vector Updates

When the receiver operates as a vector tracking receiver, the updates take on a very different process. The interrupts that occur from the PRN Generator continue to interrupt the processor; however because the integration period for the vector receiver is increased to 20 milliseconds, the correlation outputs are summed in the processor until the integration period is complete. Thus, for 19 out of every 20 interrupts, the processor simply adds the correlator values to accumulated values and allows the tracking loop to continue operating with the same NCO values. The processor's operation for the 20th interrupt is described in this section. A summary of the steps used in the vector updates is given in Figure 4-9.

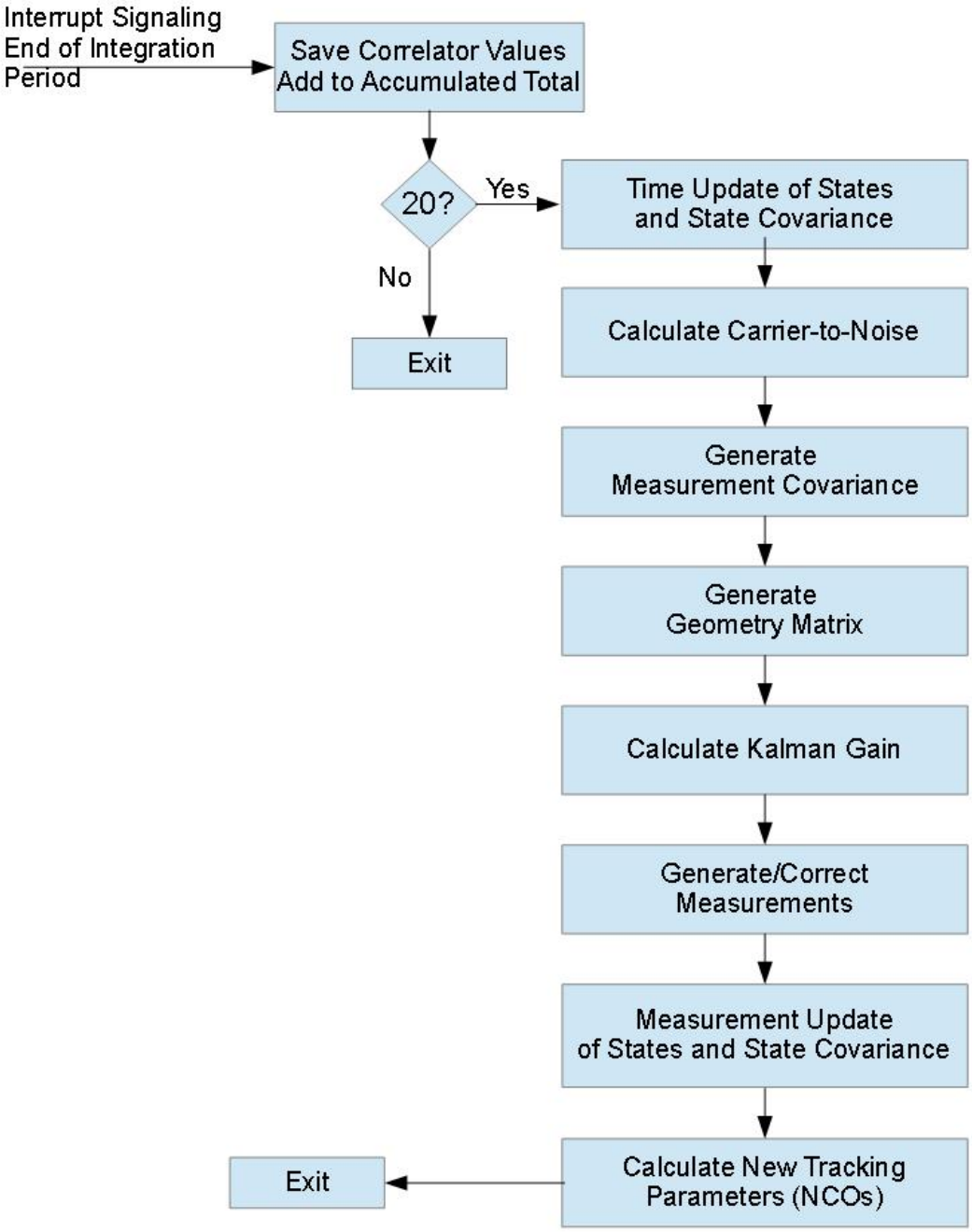

Figure 4-9 : Vector Update Flow Graph

Time Update

The receiver uses the Code NCO frequency to determine how much time has passed since that channel began its integration period. Because the NCO frequency will not change throughout the integration period, this can also be used to easily calculate the middle of the integration period. Recall from Section 3.3.2 that the state estimates are calculated for the middle of the integration periods, as opposed to the end. The time update for the states and the state covariance matrix will be calculated using the time difference between the last measurement update and the middle of the recently completed integration period.

$$\bar{\boldsymbol{x}}_{\boldsymbol{k+1}}^{(-)} = A_{k,k+1}\bar{\boldsymbol{x}}_{\boldsymbol{k}}^{(+)} \tag{4.5}$$

$$P_{k+1}^{(-)} = A_{k,k+1}P_k^{(+)}A_{k,k+1}^T + Q_d \tag{4.6}$$

Calculating Carrier-to-Noise Ratio

An accurate estimation of the carrier-to-noise ratio is key for the operation of the vector receiver; without it, the receiver has no reference to determine the quality of measurements taken from the tracking loops. First, the receiver calculates the variance of all of the noise correlator outputs. The noise correlator outputs are assumed to be thermal noise and thus the variance of the thermal noise changes slowly over time [33]. To achieve an accurate estimate of the noise variance, the variance is filtered through a moving average filter using (4.7)

$$\tilde{\sigma}_\eta^2 = (1-\alpha_\eta)\tilde{\sigma}_\eta^2 + \alpha_\eta\sigma_\eta^2 \tag{4.7}$$

where $\tilde{\sigma}_\eta^2$ is the running average of the noise variance, σ_η^2 is the instantaneous noise variance from the current noise correlator outputs, and α_η is the filtering value.

The signal amplitude is estimated in a similar manner. The instantaneous amplitude is calculated based on the current correlator outputs, then the amplitude is smoothed using a moving average filter. The instantaneous amplitude is calculated using the Early and Late correlators to help account for any error that may exist in the Prompt correlator.

$$A = (IE + IL)^2 + (QE + QL)^2 \tag{4.8}$$

In (4.8), A is the instantaneous signal amplitude, and IE, IL, QE, QL are correlator outputs. The moving average filter for the amplitude is implemented in the same manner as (4.7), resulting in a running average for the signal amplitude $\tilde{A}$. With the noise variance and signal amplitude, the carrier-to-noise ratio can be calculated using (4.9).

$$C/N_o = 10\log_{10}\frac{\tilde{A} - 4\tilde{\sigma}_\eta^2}{2T\tilde{\sigma}_\eta^2} \tag{4.9}$$

Where T is the integration period (20 milliseconds).

Generating Measurement Covariance

After calculating the Carrier-to-Noise ratio, the estimate of the signal amplitude and noise variance can be used to generate the measurement covariance matrix for the given satellite. Taken from [30],

$$E\{v_R^2\} = \frac{8\delta\tilde{\sigma}_\eta^4 + 4\tilde{A}^2\delta\tilde{\sigma}_\eta^2 f(\rho_e)}{2\tilde{A}^2}$$
$$f(\rho_e) = 2\frac{\rho_e^2}{\delta^2} + \frac{1}{2} \tag{4.10}$$

And

$$E\{v_{RR}^2\} = \frac{\tilde{\sigma}_\eta^4}{2\tilde{A}^2} + \frac{1}{4}R^2(\rho_e)\tilde{\sigma}_\eta^2 \tag{4.11}$$

Where ρ_e is the range error, δ is the width of a chip, and $R(\rho)$ is the correlation function. Both equations assume that the range error is less than ½ of a chip width.

Generate the Geometry Matrix

As mentioned in Section 3.3.2, the Geometry matrix used in this method of measurement updates is only related to one satellite. For completeness, (3.25) is repeated here.

$$C(k+N) = \begin{bmatrix} u_x & 0 & u_y & 0 & u_z & 0 & 1 & 0 \\ 0 & u_x & 0 & u_y & 0 & u_z & 0 & 1 \end{bmatrix} \tag{4.12}$$

Calculating the Kalman Gain

The Kalman gain is calculated using (4.13).

$$K(k) = P^-(k)C^T(k)[C(k)P^-(k)C^T(k) + R]^{-1} \tag{4.13}$$

This step is where some of the computational burden is lifted from the processor by implementing the Asynchronous method described in Section 3.3.2. Here, only two measurements are used for each update, range error and range-rate error for one satellite. In the Synchronous method, range and range-rate error for each satellite being tracked would be used (2*N measurements, where N is the number of active channels used in the update). By using only 2 measurements to update in the Asynchronous method, the matrix inversion used to calculate the Kalman gain is only a 2x2, which is trivial. In the Synchronous method, the inversion would be 2Nx2N.

Generating Measurements

In the receiver scalar mode, the receiver generated pseudorange and pseudorange rate measurements from the tracking loops as described in Section 3.2.2.4. Measurements for vector tracking mode are slightly different and thus are generated in a different manner. The measurements are pseudorange and pseudorange rate errors. Fortunately, these can be calculated directly from the correlator outputs as in (4.14) and (4.15).

$$Y_R = IE^2 + QE^2 - IL^2 - QL^2 \tag{4.14}$$

$$Y_{RR} = tan^{-1}\left(\frac{Y_{cross}}{Y_{dot}}\right) \tag{4.15}$$

Where:

$$Y_{cross} = IP_1 * QP_2 - IP_2 * QP_1 \tag{4.15a}$$

$$Y_{dot} = IP_1 * IP_2 + QP_1 * QP_2 \tag{4.15b}$$

Recall from Section 3.3.2 that the measurements generated at the end of an integration period need to be adjusted for any updates to the states that have occurred since the beginning of the integration period. The correction is repeated here for reference

$$\epsilon(k+N) = \epsilon^*(k+N) - C(k+N)\sum_{m=k}^{k+N-1} A(m, k+N)K(m)\epsilon(m) \tag{4.16}$$

The summation in (4.16) is stored as a vector of corrections for all of the channels. When a channel corrects its measurements, it then computes the projection of its corrections to the

other channels. For example, in Figure 4-10, after Channel 1 computes its corrected measurements, it projects this correction forward for Channel 2 by adding $A(k, k+1)K(k)\epsilon(k)$ to the Channel 2 correction value in the vector of channel corrections. $A(k, k+1)$ is estimated as the difference between the beginning of the completed integration period on Channel 1 (located by point A in Figure 4-10) and the beginning of the current integration period on Channel 2 (located by point B in Figure 4-10). Note that the receiver time in which this update is occurring is at point C, while the states are being updated for the time at point k.

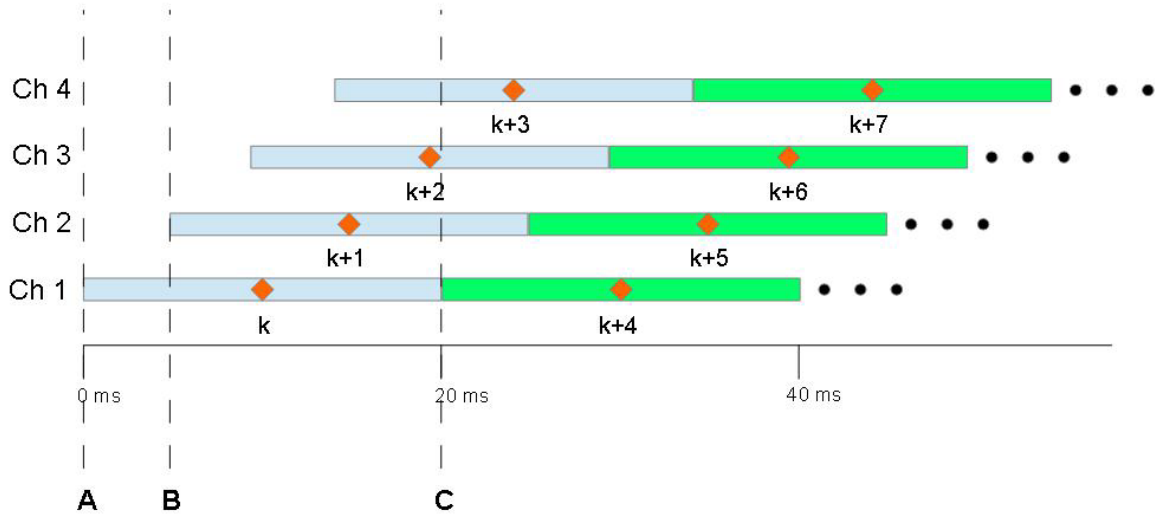

Figure 4-10 : Asynchronous State Estimate Updates

The corrected measurements will be propagated forward for each channel and applied to that channel's correction in the correction vector. Once completed, the current channel will clear its value in the correction vector so that it can be updated when the other channels complete their measurement updates.

Measurement Update of States/State Covariance

The state estimates are updated with the corrected measurements as well as the state covariance matrix as shown below.

$$\hat{x}^{+}(k) = \hat{x}^{-}(k) + K(k)\epsilon(k) \tag{4.17}$$

$$P^{+}(k) = [I_{8x8} - K(k)C(k)]P^{-}(k) \tag{4.18}$$

Calculating New Tracking Parameters

With the new estimates of the states, the receiver can now generate new frequency commands for the NCOs. To generate the Code NCO value, the pseudorange must be calculated. This is accomplished in a slightly different manner than during the scalar receiver mode. First, an index is kept which tracks the receiver's time when the beginning of the data bit being integrated over was transmitted from the satellite. This time is compared to the current receiver's time to get the pseudorange (note that the time base for these calculations remains in receiver time – the clock drift is accounted for in the Kalman filter as one of the states). An estimated pseudorange is then generated based on the user position, satellite position, atmospheric effects, and receiver clock dynamics. The difference in the measured pseudorange and the estimated pseudorange represents an error in the code phase. Dividing this by the integration period gives an error in the code frequency, which is applied to the current NCO frequency.

Just as the Code NCO is updated based on the user and satellite positions, the Carrier NCO is updated based on the user and satellite velocities. More specifically, the line-of-sight velocity between the user and satellite can, by definition of the Doppler effect, determine the offset in frequency [30].

$$f_{Carr\ NCO} = \left(1 - \frac{Vel_{LOS}}{c}\right)\frac{f_{L1}}{1 - \frac{\dot{b}}{c}} - f_{L1} + f_{IF} \tag{4.19}$$

Where $f_{Carr\ NCO}$ is the new NCO frequency, Vel_{LOS} is the line-of-sight velocity, f_{L1} is the L1 frequency, f_{IF} is the intermediate frequency of the receiver, and $\dot{b}$ is the receiver clock drift.

4.6 Transition from Scalar to Vector

The transition of GPS receiver from scalar mode to vector mode can be very difficult and must happen very quickly. The receiver first runs until a scalar position solution is calculated and then transitions into the vector mode. A scalar navigation solution is necessary to "kick-off" vector tracking because the generation of the NCO commands requires an initial estimate of the states. Recall that, when this transition occurs, each of the channels is at a different point in the received signal. In vector tracking, the integration periods increase to 20 milliseconds, which will occur during a navigation data bit. It is important to have all of the channels integrating over the same data bit, and thus after the scalar position has been determined the receiver will project when these longer integration periods will begin.

From the time that the scalar navigation solution was determined to the beginning of the first 20 millisecond integration period, the receiver continues to run scalar updates on all of the channels. This keeps the phases of all of the channels locked with the incoming signal. During this time, the receiver also transitions the scalar navigation solution into the states of the vector tracking receiver. This represents the first state estimate of the receiver and is valid for the receiver time associated with the scalar navigation solution. In order to ensure that the residuals calculated from the correlator outputs are representative of the error on the state estimate, the states are projected forward to the beginning of the first 20 millisecond integration period on

each channel. These projected state estimates are used in calculating the NCO commands for each channel to begin its integration period.

4.7 Navigation Solution

The overall purpose of a GPS receiver is to determine the user's position, velocity and time. All of the work of acquiring the satellite signals, tracking the signals, and decoding navigation data bits is for the purpose of generating a navigation solution. In Section 3.2.3, the Least Squares method was explained which is used when the receiver operates in scalar mode. In Section 3.3.1, the states of the Kalman filter used in vector mode described states that are in fact the navigation solution. This section will briefly describe how these methods are called in the receiver such that a Navigation Solution can be generated.

In general, the receiver needs to have some specified output rate at which the navigation information is output. In the FPGA, the receiver implements a global counter that runs off the sample clock to provide the processor with 1 second interrupt signals. Periodically, this counter must be corrected for drift in the sample clock. When the processor acknowledges the interrupt signal, it will generate a navigation solution based on whether it is in scalar or vector mode.

In scalar mode, the Least Squares method must be used. The interrupt signal, when triggered, also latches the values for the Code and Carrier Accumulators as well as the Code and Carrier NCO phases on each channel. This is to secure the information needed for the pseudorange measurements, and to ensure that the pseudorange measurements for each channel are synchronized to the call for a navigation solution. The receiver can then calculate the satellite positions and perform the Least Squares solution for the navigation solution.

In vector mode, no values need to be latched from the FPGA. In fact, because the states of the filter are updated whenever a measurement is collected, the navigation solution can be accessed with ease at any time. There is likely to be some time difference between when the last measurement update occurred and the navigation solution is called. To adjust for this, a time update is performed to bring the states up to the receiver time when the interrupt triggered.

Chapter 5

Performance Results

The receiver described in the previous chapter has been initially implemented in C++ for preliminary analysis and testing for functionality. The results from the C++ receiver showed success in the receiver's ability to continue tracking satellites in the vector mode. The FPGA design was then completed and implemented on the ZedBoard. Initially in lieu of a live-sky test, data was recorded from the RF front-end, then played back into the receiver at slightly faster than real-time operation. The results from the ZedBoard implementation are presented here to show a continuation of signal tracking when the receiver is in vector mode. The receiver was then tested with a live-sky; the position results of the live-sky test are presented.

5.1 Tracking Results

Correlator outputs from Channel 0 in the receiver are shown in Figure 5-1. To ease the processor's burden, the correlator outputs were only saved when a navigation solution was called (once every second). For approximately 2/3 of the data run shown (~0-60 seconds), the receiver was tracking the satellite using scalar updates. Once enough satellites were available that a navigation solution could be calculated, the receiver generated scalar navigation solutions for two seconds before switching to vector mode. After switching to vector mode, the receiver ran for about 30 seconds (~60-90 seconds).

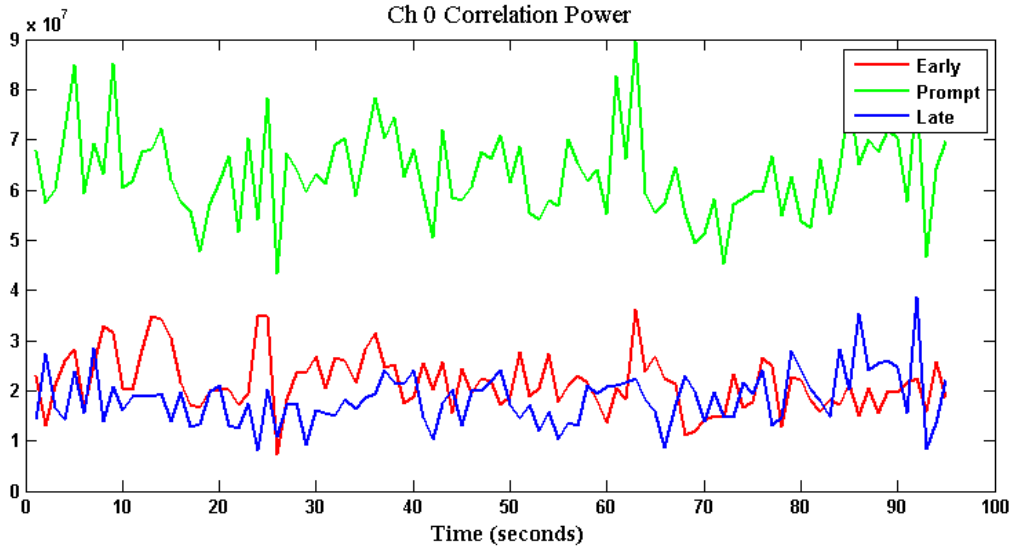

Figure 5-1 : Channel 0 Correlator Outputs

The receiver maintained lock on the satellite and transitioned smoothly from scalar to vector mode. In Figure 5-1, the Early/Prompt/Late power is shown to indicate that the signal power remained in the Prompt replica; however the individual I and Q legs are not distinguishable from this plot. Figure 5-2 shows the distinction between the different branches of the receiver.

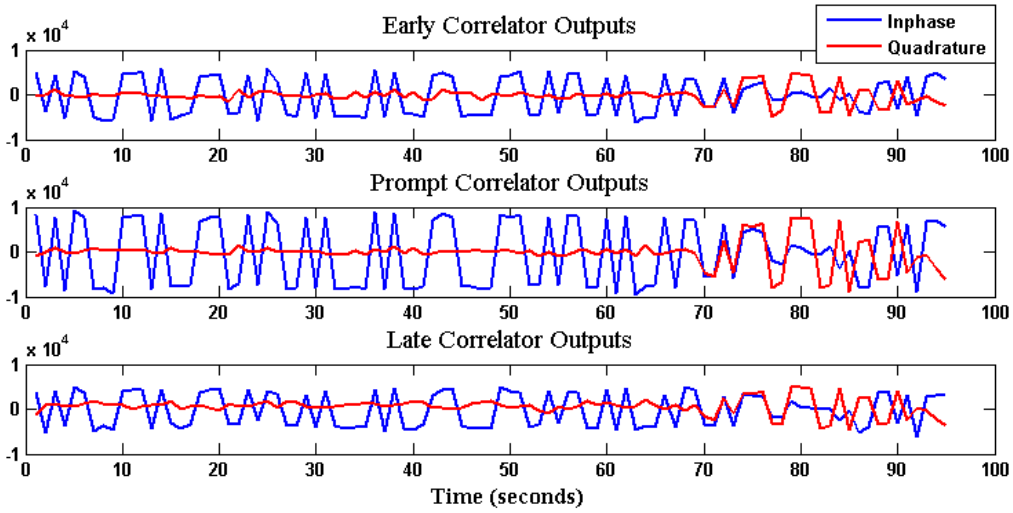

Figure 5-2 : Ch. 0 Correlator Outputs (I & Q)

Even though Figure 5-1 shows that the majority of the correlation power remains in the Prompt branch (showing lock on the C/A code), Figure 5-2 shows that the In-phase branch loses correlation power to the Quadrature branch after the receiver switches to vector mode. This is a result of the receiver only tracking frequency lock in vector mode. The filter implementation only predicts the Carrier frequency and doesn't attempt to track the Carrier phase. Accurate ranging information is still available when tracking frequency lock; however navigation data cannot be recovered unless the receiver has carrier phase lock.

Figure 5-3 shows the correlator outputs for the other channels used in the Navigation Solution. Note that the time at which Channel 3 begins reporting values is approximately the time when the transition to vector mode begins.

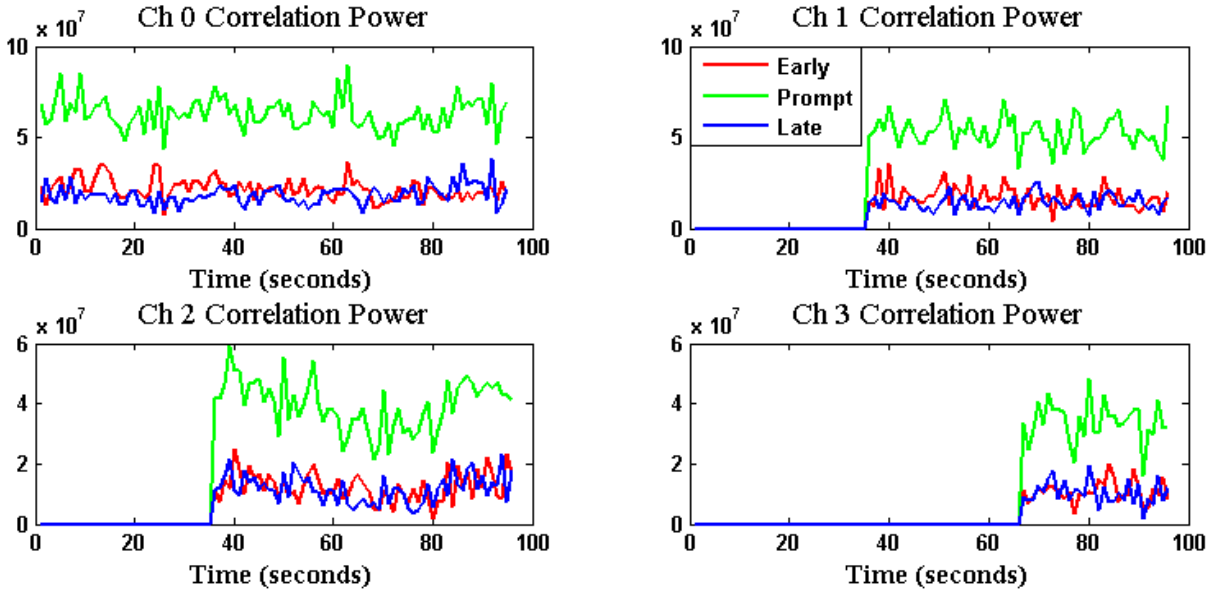

Figure 5-3 : Correlator Outputs for All Channels

5.2 Position Results

As mentioned before, the results in Section 5.1 were gathered without the use of an antenna. After confirmation that the receiver continued to track satellites while in vector mode, the receiver was connected to an antenna for a live-sky test. As before, once the receiver was able to calculate a navigation solution in scalar mode for two seconds, it switched to vector mode and continued reporting positions for approximately 30 seconds.

Figure 5-4 : Position Results

The dots in Figure 5-4, show the position output of the receiver across the 30 second period. The blue oval denotes the two positions reported while the receiver was in scalar mode, while the orange oval denotes the positions reported while the receiver was in vector mode. The "true position" of the receiver is marked by the gold star (the "true position" is estimated by observation of the map).

It is clear that the receiver experiences a large shift in position when the receiver switches to vector mode. Once this shift has occurred, the receiver becomes more steady; however it is clear that there is a bias in the positions reported in vector mode. There are several ways in which the accuracy of the receiver can be improved and this bias removed. Improvements to the pseudorange estimate equation (in particular the atmospheric models) can be made to help eliminate this bias.

5.3 Hardware Results

While the scalability of the Zynq architecture gives freedom to select larger hardware if needed, it was desired to implement this receiver on the Zynq 7020 found on the Zedboard. Thus, the resource utilization of a 12 and 16 channel vector tracking receiver was studied; the results are shown in Table 5-1.

Table 5-1: Resource Utilization

	12 Channels	16 Channels
Registers	31%	42%
LUTs	58%	77%
Occupied Slices	84%	96%
DSP Slices	0%	0%
36KB BRAM	1%	1%
18KB BRAM	4%	6%
Processor Cores	50%	50%

The 16 channel approaches utilization of all available logic slices, however without occupying almost 25% of look-up tables and almost 60% of the registers. The implementation leaves free many of the available block RAMs as well as all of the DSP slices. While the DSP slices would be very effective if used for the Correlator modules, their lack of use here simply shows room for further optimization. Also, to complete the tracking loop updates and navigation solution, only one of the processor cores has been utilized, freeing up the other processor for other tasks. Note that the results in Table 5-1 do not show the Acquisition structure described in Section 4.3.1, although that is nearly completely contained in DSP slices and BRAM.

The results shown in Figure 5-1, Figure 5-2, and Figure 5-3 are with the receiver only applying vector tracking updates to four channels. With so few channels, real-time operation is much easier because the more channels that are added the more measurement updates the receiver must finish in a given amount of time. Thus, an investigation into the timing results of the receiver was done in order to understand how many channels the receiver could operate with.

Recall that the measurement updates only occur every 20 milliseconds on each channel. However, the current configuration of the receiver has each channel reporting its correlator outputs to the processor every millisecond. This can easily be adjusted, however we shall keep this restraint for now. To test the amount of time each measurement update took, an output pin was flagged high at the beginning of the update and pulled back low at the end. This pin was observed on an oscilloscope and the duration of a measurement update is 77 microseconds. In a worst case scenario, all of the channels would need to complete a measurement update at the same time. If this were to occur, the processor would need to handle all of these updates sequentially and in less than a millisecond (such that the tracking channels could report their correlator values to the processor and no data is lost). This implies that a 12 channel receiver can be implemented in the current form.

$$\#\ of\ Channels = \frac{Integration\ Period}{Update\ Duration} = \frac{1\ ms}{77\ \mu sec} = 12.98\ Channels \qquad (5.1)$$

Again, this scenario is highly unlikely and the 1 millisecond time restraint is rather arbitrary. For a true restriction on the real-time operability of the receiver as it handles measurement updates would consider a 20 millisecond integration period. This would allow for a 259 Channel receiver.

There is more to consider than simply fitting all of the required processing into the real-time constraints. When a measurement update begins, the tracking channel will have just completed an integration period and will continue tracking into the next one. While the measurement update is being completed in the processor, the tracking channel is correlating new data with NCO values from the previous integration period. During the 77 microseconds of the measurement update, 1260 new data samples are correlated in the new integration period. While

this is less than 0.4% of the total samples in the integration period, the effect of this can be worsened by the scenario described above (i.e. two or more channels requiring measurement updates at the same or near the same time). During static situations, this is unlikely to have a drastic effect, as the NCO values will not change greatly from one integration period to the next. However, the effect this may have on high dynamic situations has yet to be investigated.

Chapter 6

Conclusions and Future Work

6.1 Conclusions and Contributions

The work described in this thesis has addressed the problem of implementing computationally burdensome GPS algorithms in a small package on a low-cost platform. A highly programmable architecture was used that enables redefinition, modification, and expansion. Implementation details have been provided which detail not only how vector tracking algorithms can be implemented on an SoC, but also serve as a guide for how to employ other advanced acquisition, tracking, or navigation algorithms on such a platform.

There are certain environments where GPS receivers can have difficulty operating; in particular urban canyons and dense foliage can block or weaken the already very weak GPS satellite signals. Research has been done in mitigating the negative effects of such adverse conditions, and a class of tracking algorithms called vector tracking has been proven in simulation to provide improvements for these environments. With vector tracking, weak signals can be tracked effectively with aiding from stronger signals. Additionally, following brief outages receivers can instantaneously reacquire the lost satellite signals. While the benefits of vector tracking have been noted, the difficulties in implementing a receiver that utilizes such algorithms have also been noted. In particular there is a large computational burden associated

with noise estimation in the tracking loops as well as Kalman filter operations to update the tracking loops. With the current state of processing technologies, there is little that cannot be computed on the best of today's computers. However, many of the applications that wish to use GPS in these adverse conditions require that the solution be small and low cost.

6.2 Future Work

While the vector tracking algorithms have been successfully implemented, there is still more that needs to be investigated regarding this implementation. As mention in Section 5.3, the latency in the measurement updates may have an effect on high dynamic situations; as more satellites are being tracked, the processing latency will have a greater effect. To investigate this, the receiver implemented in this thesis should be compared to a post-process software receiver in a dynamic situation. The number of satellites being tracked should be noted and increased as much as possible. The correlator outputs will be a good indication of whether the predictions for the new NCO values are applied soon enough in the integration period to enable effective tracking.

A natural step in future progress of the receiver is to incorporate an IMU into the Kalman filter. There are several ways that have been explored that range from different levels of coupling between the IMU data and the tracking loops. A goal of this work is to incorporate a Loosely Coupled implementation first, then progress into Deep Integration if the chosen hardware is permitting. The IMU integration will add another computational burden to the filter updates and thus care must be taken to ensure that the processor can handle the added measurements.

Because this work only focused on the use of GPS signals for the receiver, the receiver would benefit greatly from the inclusion of other satellite constellations. As mentioned at the

beginning of this thesis, there are several other constellations being launched which can be utilized to increase the accuracy of the receiver. However, to accommodate more constellations there will need to be more channels. Because the percentage of occupied slices in the FPGA is approaching 100% for 16 channels, some measures will need to be taken to allow for this expansion. Different optimization schemes could be used in the synthesis of the VHDL, and the DSP slices have still yet to be used. Also, as mentioned before, there are other Zynq options that would allow for the use of a larger FPGA.

Bibliography

[1] P. Misra and P. Enge, Global Positioning System: Signals, Measurements, and Performance, Lincoln, MA: Ganga-Jamuna Press, 2001.

[2] K. Borre, D. Akos, N. Bertelsen, P. Rinder and S. H. Jensen, A Software-Defined GPS and Galileo Receiver: Single-Frequency Approach, 2006.

[3] D. Benson, "Interference Benefits of a Vector Delay Lock Loop," in *Proceedings of the 63rd Annual Meeting of The Institute of Navigation* , Cambridge, MA, 2007.

[4] R. Crane, "A Simplified Method for Deep Coupling of GPS and Intertial Data," in *Proceedings of the National Technical Meeting of the Institute of Navigation*, San Diego, CA, 2007.

[5] W. L. Edwards, B. Clark and D. Bevly, "Implementation Details of an Embedded Deeply Integrated GPS/INS Navigation System," in *IEEE/ION Position, Location, and Navigation Symposium*, Palm Springs, CA, 2010.

[6] T. Pany, R. Kaniuth and B. Eissfeller, "Deep Integration of Navigation Solution and Signal Processing," in *Proceedings of the International Technical Meeting of the Institute of Navigation*, Savannah, GA, 2005.

[7] M. Petovello and G. Lachapelle, "Comparison of Vector-Based Software Receiver Implementations with Applications to Ultra-Tight GPS/INS Integration," in *Proceedings of the International Technical Meeting of the Institute of Navigation*, 2006.

[8] W. L. Edwards, *Development of a GPS Software Receiver on an FPGA for Testing Advanced Tracking Algorithms,* Master's Thesis, Auburn University, 2010.

[9] D. M. Akos, *A Software Radio Approach to GNSS Receiver Design,* Ph.d Dissertation, Ohio University, 1997.

[10] B. Ledvina, S. Powell, P. Kintner and M. Psiaki, "A 12-Channel Real-Time GPS L1 Software Receiver," in *Proceedings of the 2004 National Technical Meeting of the Institute of Navigation,* Anaheim, CA, 2003.

[11] Zedboard, "Zedboard Brouchure," 2014. [Online]. Available: http://zedboard.org/sites/default/files/product_briefs/ZedBoard%20Brochure%20%28English%29.pdf.

[12] C. Stroud, "Overview of FPGAs," [Online]. Available: http://www.eng.auburn.edu/~strouce/class/elec4200/FPGAoverview.pdf. [Accessed 2014].

[13] S. Cobb, "FPGA-Optimized GNSS Receiver Implmentation Techniques," in *Proceedings of the ION GNSS*, Savannah, GA, 2008.

[14] D. K. Reddy and T. S. Savithri, "Area Efficient Rapid Signal Acquisition Scheme for High Doppler DSSS Signals," *International Journal of Computer Networks & Communications,* vol. 5, no. 2, pp. 71-82, 2013.

[15] S. Kiesel, A. Maier, U. Seiller and G. Trommer, *Preliminary Simulation Results of a Deeply Coupled GPS/INS System for High Dynamics,* 2009.

[16] S. Haykin and M. Mohler, Communication Systems, Wiley, 2009.

[17] J. B. Y. Tsui, Fundamentals of Global Position System Receivers: A Software Approach,

Wiley, 2005.

[18] B. Parkinson and J. Spilker, Global Positioning System: Theory and Applications, Washington, DC: AIAA, 1996.

[19] R. Gold, "Optimal Binary Sequences for Spread Spectrum Multiplexing," *IEEE Transactions on Information Theory,* Vols. IT-13, pp. 619-621, 1967.

[20] E. D. Kaplan and C. J. e. Hegarty, Understanding GPS: Principles and Applications, Artech House, 2006.

[21] G. R. Cooper and C. D. McGillem, Probabilistic Methods of Signal and System Analysis, New York, NY: Oxford University Press, Inc., 1999.

[22] GPS.gov, "Interface Control Documents," [Online]. Available: http://www.gps.gov/technical/icwg/. [Accessed 2014].

[23] D. M. Pozer, Microwave Engineering, John Wiley & Sons, Inc, 2013.

[24] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, Pearson, 2010.

[25] J. Leclere, C. Botterson and P.-A. Farine, "Improving the Performance of the FFT-based Parallel Code-phase Search Acquisition of GNSS Signals by Decomposition of the Circular Correlation," in *Proceedings of ION GNSS*, Nashville, TN, 2012.

[26] J. T. Curran, G. Lachapelle and C. C. Murphy, "Improving the Design of Frequnecy Lock Loops for GNSS Receivers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 850-868, 2012.

[27] M. Lashley, D. M. Bevly and J. Y. Hung, "Performance Analysis of Vector Tracking Algorithms for Weak GPS Signals in High Dynamics," *IEEE Journal of Selected Topics in*

Signal Processing, vol. 3, no. 4, pp. 661-673, 2009.

[28] M. Lashley and D. M. Bevly, "Vector Delay/Frequency Lock Loop Implementation and Analysis," in *Proceedings of the Institute of Navigation*, Anaheim, CA, 2009.

[29] M. Lashley and D. M. Bevly, "Analysis of Discriminator Based Vector Tracking Algorithm," in *Proceedings of the National Technical Meeting of the Institute of Navigation*, San Diego, CA, 2007.

[30] B. Clark, *Fault Detection and Exclusion in Deeply Integrated GPS/INS,* Ph.d Dissertation, Auburn University, 2012.

[31] R. G. Brown and P. Y. C. Hwang, Introduction to Random Signals and Applied Kalman Filtering, New York, NY: Wiley, 1997.

[32] B. Keyser, D. Hodo, S. Martin and D. Bevly, "Implementation Details of a Real-Time SoC-Based Vector Tracking Receiver," in *Proceedings of ION GNSS*, Tampa, FL, 2014.

[33] M. Lashley and D. M. Bevly, "Comparison in the Performance of the Vector Delay/Frequency Lock Loop and Equivalent Scalar Tracking Loops in Dense Foliage and Urban Canyon," in *Proceedings from the 24th International Technical Meeting of the Satellite Division of the Institute of Navigation*, Portland, OR, 2011.

[34] M. Integrated, "MAX2769 Universal GPS Receiver," [Online]. Available: http://www.maximintegrated.com/en/products/co. [Accessed 2014].

[35] J. A. Starzyk and Z. Zhu, "Averaging Correlation for C/A Code Acquisition and Tracking in Frequency Domain," in *Proceedings of the 44th IEEE Midwest Symposium on Circuits and Systems*, Dayton, OH, 2001.

[36] A. Alaqeeli, J. Starzyk and F. van Graas, "Real-Time Acquisition and Tracking for GPS Receivers," in *Proceedings of the 2003 International Symposium on Circuits and Systems*, 2003.

[37] C. Sajabi, C.-I. H. Chen, D. M. Lin and J. B. Y. Tsui, "FPGA Frequency Domain Based GPS Coarse Acquisition Processor Using FFT," in *Proceedings of the Instrumentation and Measurement Technology Conference*, Sorrento, Italy, 2006.

[38] D. M. Lin and J. B. Y. Tsui, "Acquisition Schemes for Software GPS Receiver," in *Proceedings of the 11th International Technical Meeting of the Institute of Navigation*, Nashville, TN, 1998.

Appendix

IF Data Recorder

There are several tools and techniques available for test and verification of embedded systems. GPS receivers pose a special constraint in that the data to be processed in the system spans several different operating frequencies: radio frequency, intermediate frequency, and baseband. Fortunately in this thesis, because the RF signal processing is done on a commercial product (the GPS front-end), the design of the receiver is only concerned with test and verification within the IF and baseband. This can be particularly difficult for the IF, as there are more limited tools available that allow for analysis at such high speeds. To help overcome this difficulty, a high-speed digital I/O card from National Instruments was utilized as an IF data recorder. The data recorded from the device was used in both a C++ software receiver for testing baseband operations and played back into the FPGA for testing IF operations. The set-up and use of the IF data recorder is outlined in this appendix.

The IF data recorder used for this work was developed around a National Instruments PCIe card, the NI-6537B. The device has 32 digital I/O ports, all of which can be operated at frequencies as high as 50 MHz. There are also other application specific pins such as an input clock (for use with an external reference oscillator) or an output for the onboard sample clock. The device is easily programmed in LabVIEW, and can be housed in most desktop PCs. In this work, it was housed in a small ruggedized PC so as to allow for its use in mobile environments.

The IF data recorder served two functions in this work: to record sampled data from the GPS front-end, and to play that same sampled data back into the receiver.

Figure A.1 : NI 6537B

Figure A.2 : Ruggedized PC

Recording IF data for testing was instrumental in the development of this receiver. The data recorded from the front-end is sampled at 16.368 MHz and has 2 bit quantization (sign and magnitude). The front-end outputs the ADC sample clock along with the data samples, and thus the IF data recorder can use the ADC clock as an external reference to collect the data. The LabVIEW program simply pulls samples into a buffer then writes them to a binary file. One of the great benefits of having a file with sampled data from the front-end is that this data can be used in the development of the software of the receiver while the hardware of the receiver is being developed in parallel. For this work, a software receiver was developed in C++ to aid in the development of the processor tasks without the use of the FPGA. To use this C++ receiver, a data file was needed.

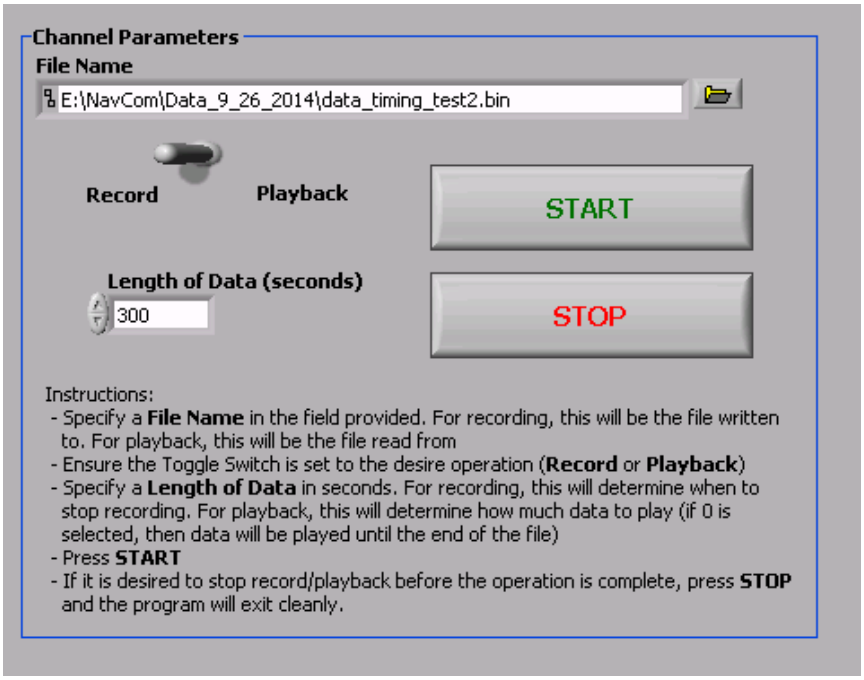

Figure A.3 : LabVIEW Program User Interface

The ability to play the recorded data back at real-time speeds was also pivotal in the development of the receiver. When the IF data recorder was used as a digital output, it would

read values from a binary file into a buffer, then write the buffer to the output pins at a specified rate. The device would also output the internal sample clock for reference. This effectively allowed the IF data recorder to replace the front-end while playing data that is already well known (from being used in the development of the software receiver as mentioned above). This functionality was utilized in the test and verification of the FPGA design. After a portion of the FPGA design was verified in simulations, the FPGA could be programmed and tested against predicted values from the software receiver.

Part of what made the IF data recorder useful for the test and verification of the FPGA designs was the use of the ChipScope IP cores from Xilinx. ChipScope allows signals in the FPGA to be stored and then output through the JTAG port for inspection. In utilizing ChipScope, input data could be compared to processed data at the sampling frequency. This was particularly useful to check the components of the tracking loops such as the NCOs and multiply-accumulates. ChipScope can store large enough sets of data that entire integration periods could be observed. The signals saved during the integration period can then be compared to expected values from both FPGA simulation and the software receiver.

Despite the high frequencies of the sampled data, the IF data recorder enabled analysis of the FPGA operations on a clock cycle level. The IF data recorder also enabled the simultaneous development of the software receiver and the FPGA, and allowed a common verification link between the two.