UCGE Reports
Number 20253

Department of Geomatics Engineering

X86-Based Real Time L1 GPS Software Receiver

(URL: http://www.geomatics.ucalgary.ca/research/publications/GradTheses.html)

by

Shahin Charkhandeh

Date

April 2007

Abstract

Given the demanding computational requirements of software-based GPS receivers, high data processing efficiency is required to obtain real-time performance. There are two basic approaches to accomplish this: reducing the number of computations required, or improving the efficiency with which the computations are carried out. This work takes the latter approach, primarily by using the MMX technology available on x86-compatible processors to more rapidly perform the Doppler removal and code correlation computations. Other computational saving methods are also described. Using this approach, computational improvements of greater than 70% are realized over the standard (integer math) implementation. Test results indicate that tracking performance of the software receiver is reasonable and that position and velocity accuracies are at the metre and decimetre per second level, respectively. Chapter one covers an introduction into GPS receiver architecture, software receivers and the needs for them. Chapter two describes in detail the theoretical aspects of different components and algorithms used in the receiver. Chapter three covers the real time operation of the GPS software receiver. It looks into the challenges and issues which needed to be addressed to achieve the real time operation in the receiver. Chapter four presents results of static and dynamic tests performed by the receiver.

Acknowledgement

Dr. Gérard Lachapelle:

There is no word that can describe my indebtedness to you. Your kindness and support has been exemplary and more than I could ever imagine. You are my mentor, my role model and a man whom I strive to be in my professional life. You have impacted my life in many ways and I am a better person because of being your student for two years.

Thank you for everything.

Dr. Mark Petovello

This work would have not been possible without your help. You were there every time that I needed help and faced a road block. I am in debt to you for ever.

Mom and Dad

You are the pure symbols of honesty, love and sacrifice. You are the stars of my life and I am proud to be your son. I dedicate this work to you as small token to show you my appreciation of what you gave me during the last 30 years.

Table of Contents

List of Tables

List of Figures

Notation

Abbreviation and Acronyms

ABAS	Airborne – Based Augmentation System
ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
ANSI	American National Standard Institute
AD	Analog to Digital Converter
BPSK	Bi-Phase Shift Keying
BOC	Binary Offset Carrier
CA-Code	Coarse Acquisition Code
C/N0	Carrier to Noise Ratio
CPU	Central Processing Unit
DFT	Discrete Fourier Transform
DLL	Delay Lock Loop
DMA	Dynamic Memory Access
DoD	U.S. Department of Defence
DOP	Dilution of Precision
DoT	Department of Transportation
EGNOS	European Navigational Geostationary Overlay Service
EU	European Union
ESA	European Space Agency
FFT	Fast Fourier Transform

FLL	Frequency Lock Loop
FPGA	Filed Programming Gate Array
GBAS	Ground-Based Augmentation Systems
GPS	Global Positioning Systems
GNSS	Global Navigation Satellite Services
HOW	Hand Over Word
IF	Intermediate Frequency
IPEXSR	Institute of Geodesy and Navigation PC-based Experimental Software Receiver
LAAS	Local Area Augmentation System
LO	Local Oscillator
MLS	Maximum-Length Sequence
MM0 - MM7	MMX registers
NI	National Instrument
NCO	Numerically Control Oscillator
PLAN	Position, Location And Navigation
PC	Personal Computer
P-Code	Precise Code
PLL	Phase Lock Loop
PRN	Pseudo Random Noise
PSR	Purdue Software Receiver
Q	Quality factor of band pass or notch filter
RAM	Random Access Memory

RF	Radio Frequency
RMS	Root Mean Square
SBAS	Satellite - Based Augmentation Systems
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SSE	Streaming SIMD Extension
S/N_0	Signal to Noise Ratio
TLM	TELEMETRY
TRIGR	Transform-Domain Instrumentation Global Positioning System (GPS) Receiver
UHF	Ultra High Frequency
UTC	Universal Time Coordinated
WAAS	Wide Area Augmentation System

Chapter 1

Introduction

The evolution of technology has increased the demand for higher accuracy, availability and reliability in positioning services. The new requirements increased the need for modernization of current GNSS and also developing new systems that will complement GPS I. Satellite Based Augmentation Systems (SBAS), Ground Based Augmentation Systems (GBAS), Airborne Based Augmentation System (ABAS), US Wide Area Augmentation System (WAAS), Local Area Augmentation System (LAAS) and European Navigational Geostationary Overlay Services (ENGOS) are some of the popular systems that complement GPS to improve performance. However, all the above systems' performances are still limited due to original design of GPS.

Consequently, the experiences gained in the design of GPS and high market demands have initiated a desire for new signals and constellations. The US Department of Defence (DoD) and Department of Transportation (DoT) have started a GPS modernization process, called GPS II and GPS III. The European Union (EU) and the European Space Agency (ESA) decided to launch their own GNSS constellation known as Galileo. Also, the Russian space program has decided to enhance its GLONASS.

Modern GPS receivers usually use Application Specific Integrated Circuit (ASIC) for signal processing and a high speed microprocessor for application calculations. The main

advantage of using an ASIC for signal processing is high speed and low power consumption. ASIC chips can not be recompiled like general purpose processors, therefore they are expensive to redesign and modify.

This highlights the main advantage of software receivers. It allows the user to redesign the system and to test new algorithms. It is fairly inexpensive and easy to modify a software receiver that is running on a programmable microprocessor. Also, as mentioned above, the arrival of new signals requires new algorithms to be developed and tested. Software receivers allow the developer to work on these algorithms and have more control on them.

Research on GPS software started about ten years ago but because of limitations in the computational power that was available at that time, the initial systems have been slow and inefficient. However, as computational power has increased, there has been renewed attention in this field. A lot of research has been done on different optimization techniques to make the receiver as fast as possible and eventually reach real time performance.

This work focuses on the difficulties that exist which limit the real time operation of a GPS receiver and how to over come them. Chapter 2 of this thesis covers the fundamental theory related to GPS receivers. Specifically, it covers concepts from the front-end, acquisition, tracking and solution calculations. Chapter 3 discuses the optimization techniques used in the software receiver developed herein. Chapter 4 demonstrates the results achieved and finally, in chapter 5, conclusions are presented along with a description of useful future work that could be carried out.

There are typically two kinds of software receivers, PC-based and FPGA based. In this section, we briefly describe the work which has been done in each of these areas and the comparisons between the two approaches.

1.1 FPGA-Based Vs PC-Based software receiver

There are typically two different approaches toward software receiver design. The first approach is a complete PC based "software receiver" which processes the digitized intermediate frequency within the CPU of the personal computer. All the signal processing is done inside the PC processor.

In the other approach, the complete base-band processing is performed within the field programmable gate array (FPGA) thus reducing the load on the PC processor. In this method, all the high speed signal processing tasks are completed in FPGA while the lower rate tasks such as navigation calculations are performed in the PC. Both approaches offer a good degree of control over the base-band processing and the capability of testing algorithms side by side.

One of the major requirements for this work was to develop software that can be used as a research tool. Therefore, the PC-Based approach was chosen since it will allow researchers that may not be familiar with FPGA to run and use the tool on their PCs. Selected previous work in this field is summarized below.

IPEXSR (Institute of Geodesy and Navigation PC-based E**x**perimental Software Receiver) is a PC based software GNSS receiver that is developed in University FAF Munich (Pany et al 2004). This receiver is realized in C++ as a Microsoft Windows

program. Real signals are digitalized by an NI 5112 analog to digital conversion (ADC) card which is connected to a low bandwidth (2.5 MHz) GP 2010 L1 front-end. This receiver has the capability to work in real time mode (connected to the front-end) and high speed off line (reading from the data file). It uses FFT/DFT techniques to perform code phase and Doppler search. FFT/DFT techniques for acquisition will be described in detail in next chapter of this thesis. The Tracking component of the receiver works in two modes, normal and under sampling.

In under sampling mode, only every x'th sample is processed. The main purpose of this method is to reduce the processing demands by a factor of x as discussed in the paper. Under sampling decreases the signal to noise ratio by 10 log(x), but does not have any other effect on the tracking performance, even though the Nyquist criterion may not be fulfilled. In Normal Mode of operation, the tracking module makes use of highly optimized code using the SSE2/3 instruction sets. The receiver can track nine channels in real-time with a sampling rate of 4.096 MHz on a single Pentium IV running at 3 GHz. The receiver was able to achieve a positioning accuracy of -0.5 m in longitude, -0.7 m in latitude error and -1.5 m in height.

GPSrx is another real-time GPS software receiver which has been developed in Stanford University (Akos et al 2001). The entire receiver algorithms are written using ANSI C. The receiver provides sub-second acquisition and solves for the position in real-time using four channels on a 650 MHZ x86 compatible processor. The front-end is a single stage frequency down conversion with band pass sampling. The IF 3 dB bandwidth is 2 MHz and the sampling frequency can be in the range of 4-6 MHz. There was a direct intent to keep the receiver "platform independent" therefore the receiver does not benefit

from using the MMX/SSE instruction sets. All the algorithms are highly optimized C routines. An FFT-based circular convolution technique is used for acquisition. The PRN code tracking is done using a traditional non-coherent second order delay lock loop (DLL). Tracking the carrier is solved using a dynamic combination of frequency and phase lock loops (FLL/PLL). The result presented showed a position solution with errors significantly less than 100 m. One of the reasons for the higher error magnitude, although acceptable, is the limitation due to the narrow bandwidth front end design.

A Real time GPS civilian L1 software receiver was also developed by Cornell University (Ledvina et al 2003). The receiver consists of an RF front end, a system of shift registers and DAQ card and a PC running an Intel Pentium 4 at 1.7 GHz. The RF front-end down converts the signal into a 2-bit digital data stream at 5.714 MHz. The Base band mixing of the local code and incoming data is accomplished through a bit-wise parallel operation. Bit-wise parallel operations work with representations of data that store successive samples in successive bits of a word. For example, 32 samples of the RF front-end output are stored in two 32-bit words. One word stores the 32 sign bits of the 32 samples, and the other word stores the 32 magnitude bits. The stored tables of the base-band mixing cosine and sine waves have their sign and magnitude bits stored in separate words, with each 32-bit word storing 32 sign or magnitude bits that tabulate to 32 successive samples of the corresponding cosine or sine waves. Similarly, the stored tables of the prompt and early-minus-late codes store sign or sign and zero-mask bits in words with each word storing 32 samples worth of data. With this scheme, the EXCLUSIVE OR operations that are involved in mixing operate on 32 samples at a time as the processor has a bit-wise

EXCLUSIVE OR command and other bit-wise commands that operate in parallel on each of two input arguments' 32-bit pairs (Ledvina et al 2003). The receiver is able to run 10 channels in parallel and in real time while providing 10-15 m position navigation accuracy for standalone applications (roof top antenna).

The receiver was later modified to process Galileo signals (Ledvina et al 2006). The work showed the interoperability between a GPS and Galileo real-time software receiver. The real-time capability of this receiver is based mainly on two technologies.

The first technology, called bit-wise parallel signal processing (described above) allows the receiver to process 32-64 RF samples in parallel. The second technology is a method for efficient real-time generation of over-sampled bit-packed PRN codes. This technology is helpful since Galileo L1 signal have longer periods than the L1 C/A code signals (Ledvina et al 2006). It is usually a common approach in a software receiver design to store replica PRN codes to reduce the computational real-time needs. However, the longer length of Galileo L1-B and L1-C BOC (1, 1) PRN codes makes this method less attractive due to the additional memory requirements for storing the longer codes. This can be a problem especially in embedded platforms. This second technology addressed this issue while adding some computational cost to the receiver design (Ledvina et al 2006).

The Purdue Software Receiver (PSR) is a real-time software receiver developed at Purdue University for research and teaching purposes (Heckler & Garrison 2004). Real-time operation is achieved by single instruction multiple data (SIMD) instructions found

in modern x86 and Power PC processors. The Software is coded in C++ making use of threaded objects to encapsulate functions and related data together, and to reduce unnecessary copying of data (Heckler & Garrison 2004).

The PSR utilizes the 16 bit signed integer data type as its basic data element. Sixteen signed bits of precision guarantees enough dynamic range to prevent overflow during the multiplication stage of the correlation algorithm using the common 1 to 2 bit quantized GPS signal, 1 bit raw PRN code, and several bits for a digital sinusoid. In addition, the 16 bit signed integer corresponds to the *word* type associated with x86 SIMD operations. Using the original 64-bit MMX registers, four samples are operated upon in parallel. With the expanded 128 bit SSE registers found in more recent CPUs, eight samples can be processed with a single operation. Real time operation of the receiver was demonstrated by forcing the receiver to process 100 seconds of data. In order to remove latency involved with disk access, the 100 seconds of raw GPS data was buffered completely into RAM (not reading from a file) before processing. The raw GPS data was sampled at 2.3333 MHz, leading to a correlation length of 2333 samples. The receiver was able to process this data in less than 100 seconds while running 12 channels. However, there was no information in the paper about the accuracy of the navigation solution computed by the receiver.

A Transform-Domain Instrumentation Global Positioning System (GPS) Receiver (TRIGR) was developed at Ohio University (Soloviev et al 2005). The concept behind this development was to build an instrument that one can simply connect to a GPS signal

and have it reveal ‘everything’ that one needs to know about the GPS signal. This instrument does not operate in real-time but uses a GPS software receiver at its heart. Signal processing of the TRIGR is a combination of batch and sequential processing techniques. The receiver uses a FFT approach in the acquisition component. CA code and carrier tracking measurements are applied to aid the sequential correlators for the tracking of the P code GPS signal. CA-code measurements set up parameters of the replica carrier and replica P-code to wipe-off the carrier and P-code signal components from the incoming L1 or L2 encrypted P(Y) code signals.

The work presented in this thesis attempts to combine different techniques presented in some of the above work to design, develop and test a real-time GPS software receiver. The focus of the work was on real-time operation of the receiver and employing algorithms that can help one achieve that level of performance. As discussed later, there are some trades off in using some of these algorithms (small hit on the accuracy of the receiver). This work is unique in the sense that it takes the use of SIMD instructions to the next level and shows how useful this features can be in GPS software receiver design. As has been presented earlier, software receivers are the heart of many different applications such as TRIGER (Soloviev et al 2005). The work contained in this thesis can also be a building block for such instruments.

Chapter 2

GPS Receiver Theory

This chapter describes the properties of the GPS signal and operations performed inside the GPS receiver to process this data and compute a navigation solution.

2.1 GPS Signal Structure

2.1.1 GPS Carrier

GPS signals are transmitted on one of two radio frequencies in the UHF band (Tsui & James 2000). The two frequencies (L1 and L2) are derived from the fundamental frequency f_0 = 10.23 MHz :

$$f_{L1} = 1575.42 \text{ MHz} = 154\, f_0 \quad (2.1)$$

$$f_{L2} = 1227.6 \text{ MHz} = 120\, f_0 \quad (2.2)$$

The wavelengths of the carriers are:

$$\lambda_{L1} = 19.03 \text{ cm} \quad (2.3)$$

$$\lambda_{L2} = 24.42 \text{ cm} \quad (2.4)$$

2.1.2 Coarse-Acquisition (C/A) and Precise (P) codes

GPS Pseudo-Random Noise (PRN) code is a binary sequence that appears to be random in nature. GPS uses two classes of codes listed in the table below (Tsui & James 2000).

Table 1: Characteristics of GPS codes

Parameter	C/A-Code	P-Code
Chipping Rate (chips/s)	1.023 e6	10.23 e6
Chipping Period (ns)	977.5	97.75
Range of One Chip (m)	293.0	29.30
Period of the Code	1msec	1week

The GPS signal is a phase-modulated signal with an initial phase of zero or π ; this type of phase modulation is referred to as bi-phase shift keying (BPSK). The spectrum shape can be described by the sinc function (sinx/x) with a spectrum width proportional to the chip rate. As an example, if the chip rate is 1 MHz, the main lobe of the spectrum has a null-to-null width of 2 MHz.

The P-code, when encrypted, is termed Y-code. Y-code is classified and unauthorized users cannot access it. (Tsui & James 2000).

.

2.1.3 Generation of C/A code

The GPS signal is from a family of pseudorandom noise (PRN) codes known as Gold codes (Tsui & James 2000). Signals are created as the product of two 1023-bit PRN sequences G1 and G2. A maximum-length linear shift 10 stage register driven by a 1.023 MHz clock generates the G1 and G2 signals. A maximum-length sequence (MLS) generator can be formed from shift registers with proper feedback. The feedback of G1 is from bits 3 and 10 as shown in figure one and corresponds to the polynomial G1: $1 + X^3+X^{10}$. The feedback of G2 is from bits 2, 3, 6, 8, 9 & 10 as shown in figure 1.

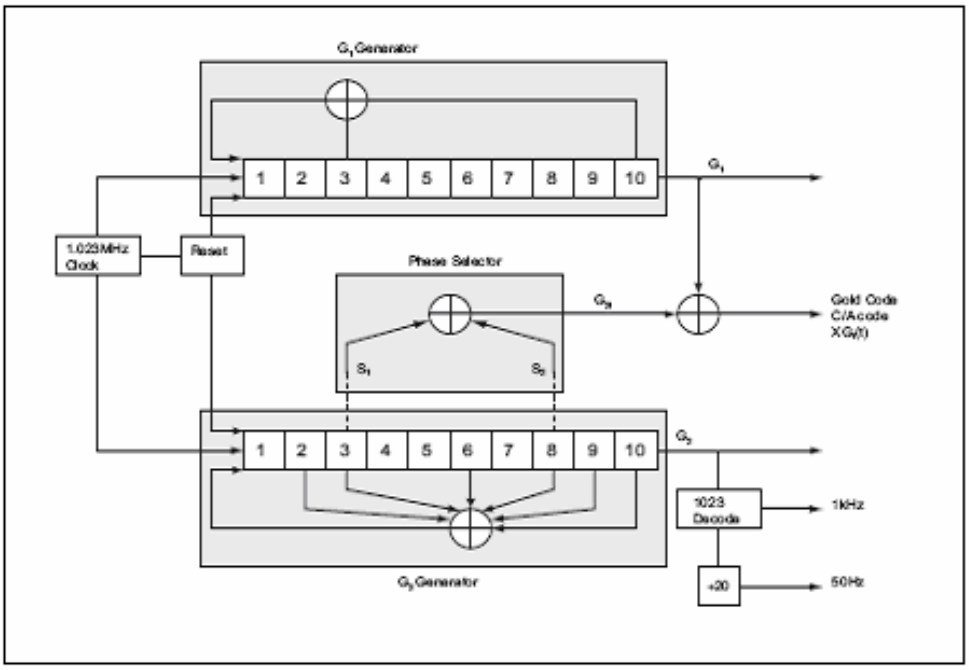

Figure 1: Block diagram of C/A code generation

The corresponding polynomial is G2: $1+X^2+X^3+X^6+X^8+X^9+X^{10}$. To make different C/A codes for satellites, the outputs of the two shift registers are combined in a special

manner. *G*1 always supplies its output, but *G*2 supplies two of its states to a modulo-2 adder to generate its output. The selection of states for the modulo-2 adder is called phase selection. Table 2 shows the combination of phase selections for each C/A code. It also shows the first 10 chips of each code in octal representation. Note that only the first 32 of the 37 possible codes are used as Satellite C/A codes. The remaining five codes are reserved for other uses e.g. ground transmitters (pseudolites).

Table 2: Combination of phase selection for C/A code

Satellite ID Number	GPS PRN Signal Number	Code Phase Selection	Code Delay Chips	First 10 Chips C/A Octal
1	1	2 ⊕ 6	5	1440
2	2	3 ⊕ 7	6	1620
3	3	4 ⊕ 8	7	1710
4	4	5 ⊕ 9	8	1744
5	5	2 ⊕ 10	17	1133
6	6	1 ⊕ 8	18	1455
7	7	2 ⊕ 9	139	1131
8	8	3 ⊕ 10	140	1454
9	9	2 ⊕ 3	141	1626
10	10	3 ⊕ 4	251	1504
11	11	5 ⊕ 6	252	1642
12	12	6 ⊕ 7	254	1650
13	13	7 ⊕ 8	255	1764
14	14	8 ⊕ 9	256	1772
15	15	9 ⊕ 6	257	1775
16	16	2 ⊕ 10	258	1776
17	17	1 ⊕ 4	469	1156
18	18	2 ⊕ 5	470	1467

19	19	3 ⊕ 6	471	1633
20	20	4 ⊕ 7	472	1715
21	21	5 ⊕ 8	473	1746
22	22	6 ⊕ 9	474	1763
23	23	1 ⊕ 3	509	1063
24	24	4 ⊕ 6	512	1706
25	25	5 ⊕ 7	513	1743
26	26	6 ⊕ 8	514	1761
27	27	7 ⊕ 9	515	1770
28	28	8 ⊕ 10	516	1774
29	29	1 ⊕ 6	859	1127
30	30	2 ⊕ 7	860	1453
31	31	3 ⊕ 8	861	1625
32	32	4 ⊕ 9	862	1712
**	33	5 ⊕ 10	863	1745
**	34*	4 ⊕ 10	950	1713
**	35	1 ⊕ 7	947	1134

2.1.3.1 C/A code correlation properties

C/A codes have high autocorrelation peaks and low cross-correlation peaks. Different C/A codes have a cross correlation of -65/1023 (occurrence 12.5 %), -1/1023 (75%) and 63/1023 (12.5%) (Tsui & James 2000). Figure 2 and Figure 3 show the autocorrelation of Satellite 1 and cross correlation of Satellite 15 and 1 respectively.

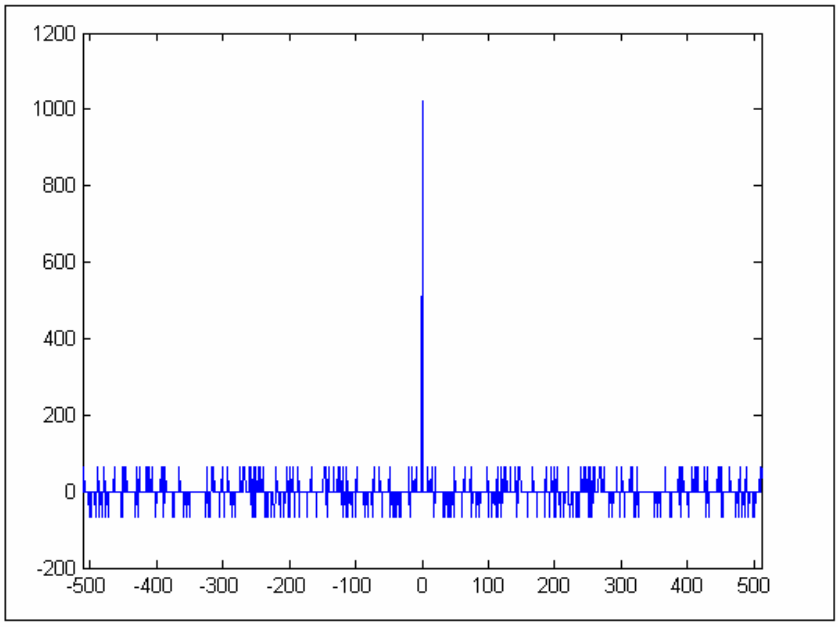

Figure 2 : Autocorrelation of Satellite 1

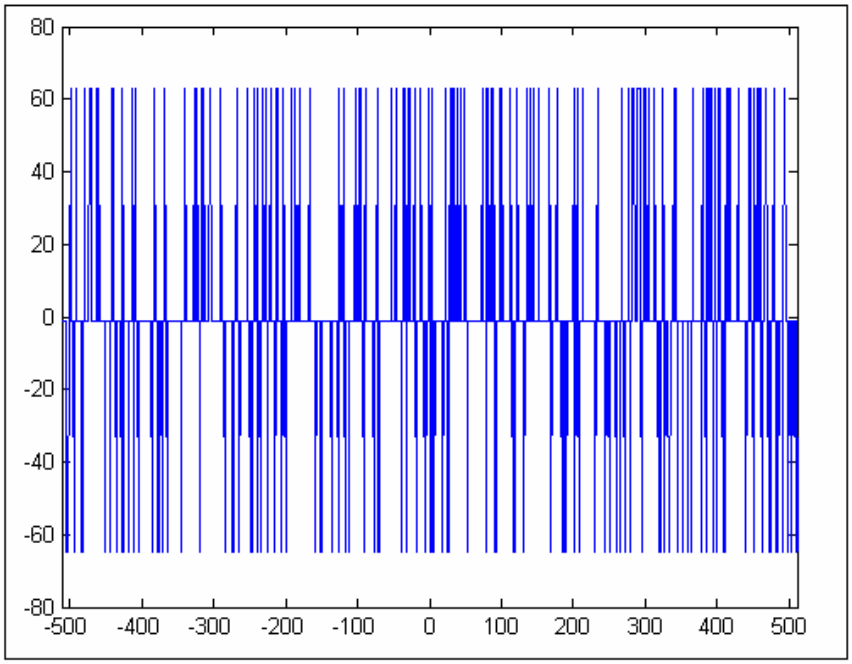

Figure 3 : Cross correlation of Satellite 1 and 15

The above figures show that the maximum autocorrelation peak is 1023, equal to the C/A code length. The other correlation values are 63,-1 and -65 as mentioned above.

2.1.4 Navigation data bits

The C/A code is a bi-phase coded signal, which changes the carrier phase between Zero and π at a rate of 1.023 MHz (Tsui & James 2000). The navigation data bit is a bi-phase code as well at a rate of only 50 Hz, which translates to each data bit being 20 ms long. Since the C/A code is 1 ms, there are 20 C/A codes in one data bit. Therefore, in one data bit all 20 C/A codes have the same phase. A phase transition due to the data bit causes a phases difference of $\pm\pi$ in two adjacent C/A codes.

The navigation data message is divided into a 1500-bit long frame containing 5 sub-frames, each 300 bits long. Each sub-frame contains ten words, each word being 30 bits long. Sub-frames 1, 2, and 3 are repeated in each frame. Sub-frames 4 and 5 have 25 different versions referred to as pages 1 to 25. Considering a bit rate of 50 bps, the transmission of a sub-frame lasts 6 seconds, a frame lasts 30 seconds and an entire navigation message lasts 12.5 minutes (ICD-GPS-200 2003).

2.1.4.1 TELEMETRY (TLM) and HAND OVER WORD (HOW)

The first two words of all the sub frames are the telemetry (TLM) word and hand over word (HOW) (Tsui & James 2000). Each has a length of 30 bits. Figure 4 shows the structure of these words. The TLM word starts with an 8-bit preamble, followed by 16 reserved bits and 6 parity bits.

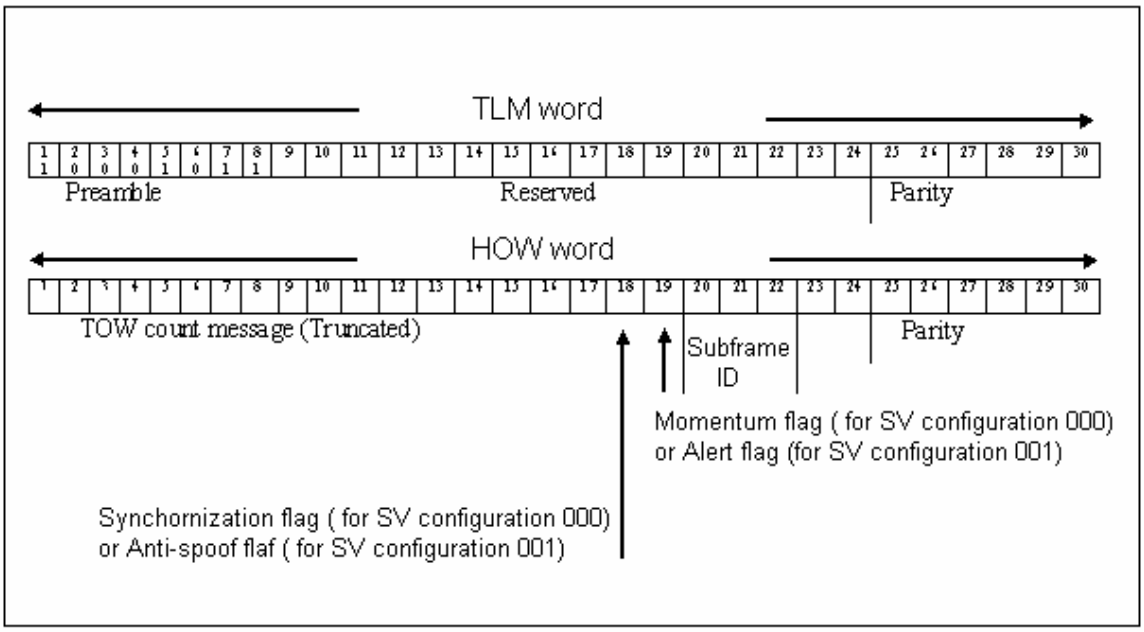

Figure 4 : TLM and HOW words

In addition to the TLM and HOW words, there are eight other words in each sub frame. Below is a brief description of data in these words. For a complete description, one can refer to the ICD-GPS-200 document. Sub-frame 1 contains some clock information. That is information necessary to calculate the time of transmission for the navigation message from the satellite. In addition, sub-frame 1 contains health data indicating whether the data is corrupted. Sub-frames 2 and 3 contain the satellite ephemeris data. The ephemeris data provides the satellite orbits, needed to calculate accurate satellite positions.

As mentioned earlier, the last two sub-frames repeat every 12.5 minutes giving 50 sub-frames. Sub-frame 4 and 5 contain almanac data. The almanac data is the ephemeris and clock data with reduced precision. Additionally, each satellite transmits almanac data for all GPS satellites while only transmitting ephemeris data for itself. The remainder of sub-

frames 4 and 5 contain various data such UTC time parameters, health indicators, and ionosphere parameters.

2.2 GPS Receiver Architecture

2.2.1 RF Front-End

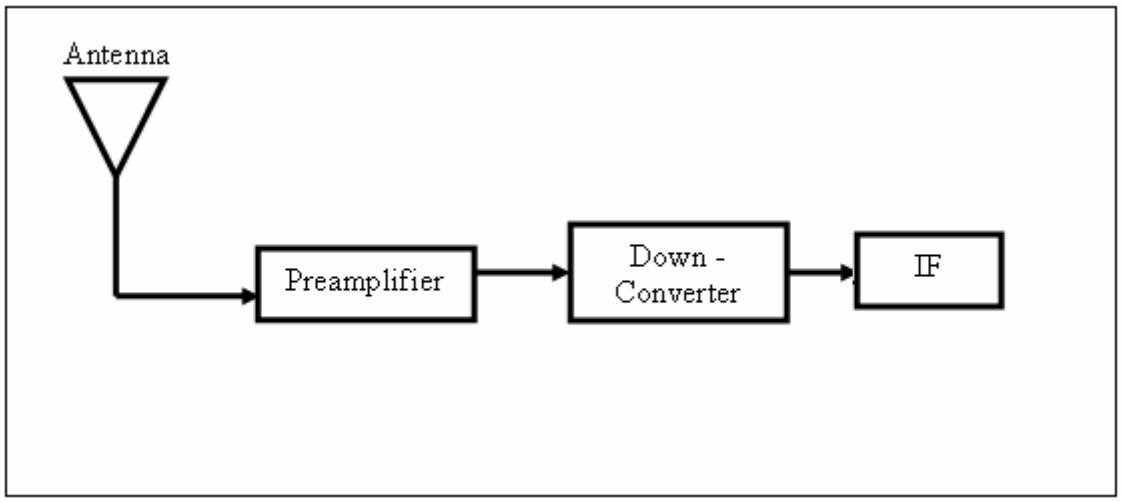

Figure 5: Block diagram of GPS Front-End

The first component after the antenna is usually an amplifier or filter.

The noise figure of a receiver is (Tsui & James 2000):

$$F = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \ldots + \frac{F_N - 1}{G_1 G_2 \ldots G_N} \tag{2.5}$$

where F_i and G_i (i = 1, 2 . . . N) are the noise figure and gain, in linear terms, of each individual component in the RF chain. If the amplifier is the first component, the noise figure of the receiver is low and is approximately equal to the noise figure of the first amplifier, which can be less than 2 dB.

Therefore, the noise contribution caused by the second component e.g. a filter, is reduced by the gain of the preceding amplifier. However, strong signals within the bandwidth of the amplifier may drive it into saturation and generate spurious frequencies.

On the other hand, if the first component is a filter it can stop out-of-band signals from entering the input of the amplifier. A filter with 2 MHz bandwidth (to pass only C/A code) with a center frequency at 1575.42 MHz is considered high Q (Quality Factor) Usually, the insertion loss of such a filter is relatively high, about 2 to 3 dB, and the filter is bulky. The receiver noise figure with the filter as the first component is about 2–3 dB higher than in the previous arrangement (Tsui & James 2000).

2.2.1.1 Mixing Operation and Intermediate Frequency Filtering

Down conversion from RF to IF is accomplished by mixing the incoming signal and noise with a local oscillator signal (*LO*). This process is illustrated in figure 6.

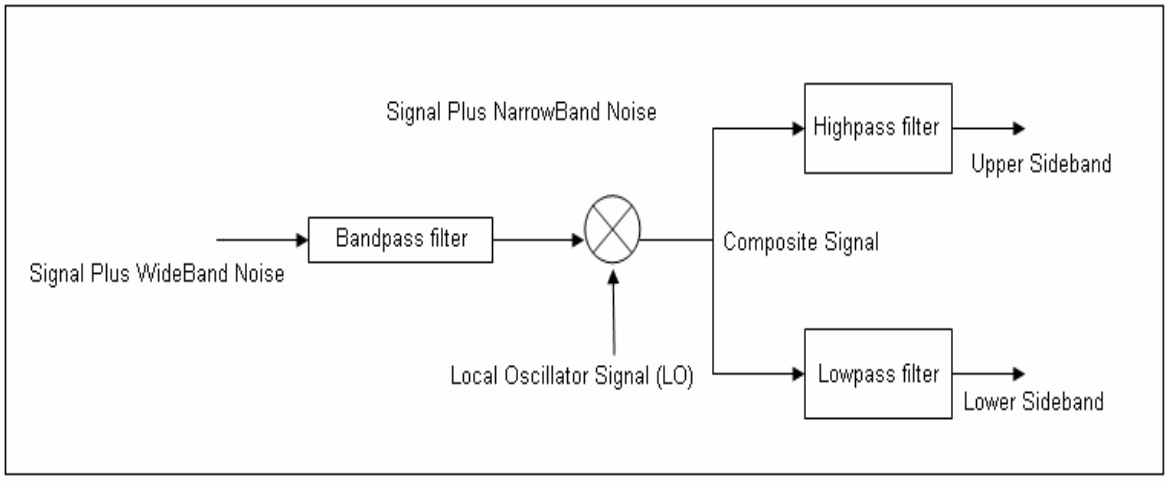

Figure 6 : Mixing operation

A GPS signal may be represented by:

$$s(t) = AC(t)D(t)\cos[(\omega_0 + \Delta\omega) + \Phi_0] \quad (2.6)$$

where

A = signal amplitude

$C(t)$ = PRN code modulation

$D(t)$ = 50bps data modulation

$\omega_0 = 2\pi f_0 =$ Carrier frequency (L1 or L2)

$\Delta\omega = 2\pi\Delta f =$ Frequency offset (Doppler, etc)

$\Phi_0 =$ Nominal but ambiguous carrier phase.

Therefore, the result of the mixing (ignoring the harmonics, LO feed through and image noise) of GPS signal and LO signal ($LO_1(t) = 2\cos\omega_1 t$) is:

$$s_{IF}(t) = AC(t)D(t)\{\cos[(\omega_0 + \omega_1 + \Delta\omega)t + \Phi_0] + \cos[(\omega_0 - \omega_1 + \Delta\omega)t + \Phi_0]\} \quad (2.7)$$

The upper side band is not of any interest and can be eliminated via a low-pass filter. This will result in an IF frequency of $\omega_{IF} = \omega_0 - \omega_1$.

2.2.1.2 Conversion to Baseband

Conversion to baseband means synthesizing the IF signal to that of in-phase and quadra-phase components of the signal envelope. This can be accomplished by analog mixing or by a technique known as IF (or pass-band) sampling (Van Dierendonck 1996).

Analog mixing, illustrated in figure 7, is realized by mixing the IF signal with two local carriers, one of which is phase shifted by 90° with respect to the other (in quadrature). The resulting in-phase and quadra-phase signals are as follow:

$$LO_{2I}(t) = \sqrt{2}\cos\omega_2 t \tag{2.8}$$

$$LO_{2Q} = \sqrt{2}\cos(\omega_2 t + \frac{\pi}{2}) = -\sqrt{2}\sin(\omega_2 t) \tag{2.9}$$

$$I_s(t) = \frac{A}{\sqrt{2}} C(t) D(t) \cos(\Delta\omega_B t + \Phi_0) \tag{2.10}$$

$$Q_s(t) = \frac{A}{\sqrt{2}} C(t) D(t) \sin(\Delta\omega_B t + \Phi_0) \tag{2.11}$$

where the residual frequency offset is:

$$\Delta\omega_B = \omega_{IF} - \omega_2 + \Delta\omega \tag{2.12}$$

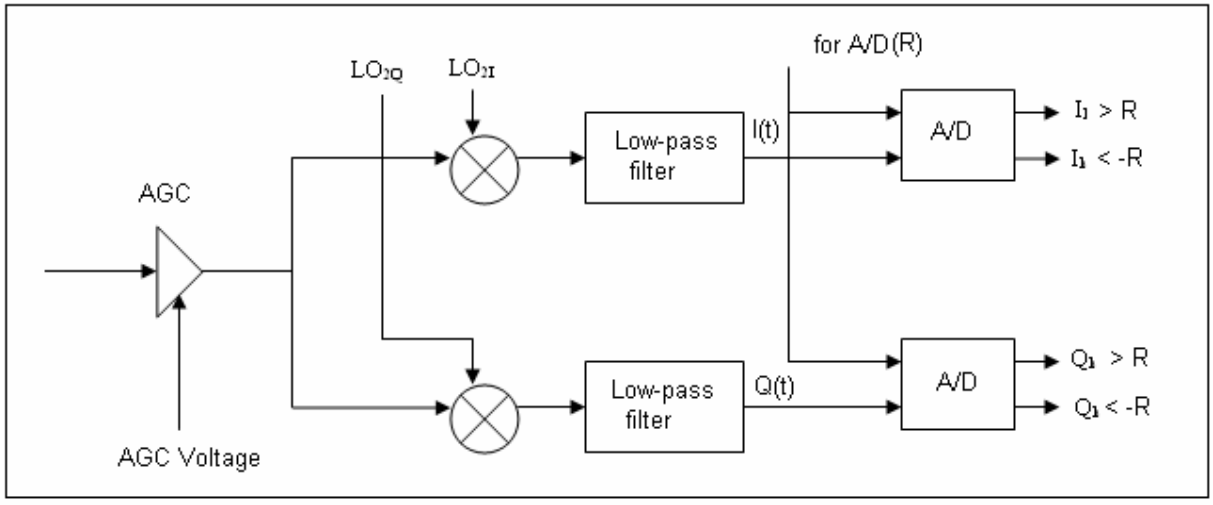

Figure 7 : Conversion to base-band (analog mixing)

Intermediate sampling (IF sampling) shown in Figure 8 (Van Dierendonck 1996) is based on the concept of sampling the IF signal at rate which the Q samples and I are obtained directly.

Let us assume the sample rate of SR as follow:

$$SR = \frac{4 f_{IF}}{N} \tag{2.13}$$

where f_{IF} is the IF frequency being sampled and N is an odd number. Therefore, the signal will be sampled at intervals t_k where:

$$t_k = \frac{kN}{4 f_{IF}} \text{sec}; k = 0,1,..... \tag{2.14}$$

Sampling the signal at this rate will result in the following:

$$\begin{aligned} s_k &= s_{IF}(t_k) = AC(t_k)D(t_k)\cos\left[2\pi(f_{IF} + \Delta f)\frac{kN}{4 f_{IF}} + \Phi_0\right] \\ &= AC_k D_k \cos\left[\frac{\pi kN}{2}(1 + \frac{\Delta f}{f_{IF}}) + \Phi_0\right] = AC_k D_k \cos\left[\frac{\pi kN}{2} + \Phi_k\right] \end{aligned} \tag{2.15}$$

where the Δf is the offset due to Doppler, C_k and D_k are the code and data at time t_k, and

$$\Phi_k = \Phi_0 + \frac{\pi kN\Delta f}{2 f_{IF}} = \Phi_0 + \Delta\omega t_k = \Phi_0 + \Delta\Phi_k \tag{2.16}$$

is the baseband phase of the sample attributable to the nominal phase and frequency offset at time t_k. It can be noted from equation 2.15 (ignoring Δf) that IF signals sampled at exactly 90° phase result in the following sequence of samples based on values of N:

$$\sqrt{2}[I_{sk}, Q_{sk}, -I_{sk}, -Q_{sk}, I_{sk}, Q_{sk}, ...]$$

or

$$\sqrt{2}[I_{sk}, -Q_{sk}, -I_{sk}, Q_{sk}, I_{sk}, -Q_{sk}, ...]$$

Δf causes a phase rotation of the samples that is removed after the sampling process.

It is interesting to notice the negative samples and the meaning of it. To explain this further, one need to drive the formulas as follow:

$$s_k = I_k$$

$$s_{k+1} = \cos(\frac{\pi N}{2}(k+1) + \Phi_0) = \cos(\frac{\pi k}{2}k + \Phi_0 + (\frac{\pi N}{2})) =$$

$$Q_k = \sin(\frac{\pi N}{2} + \Phi_0)$$

$$s_{k+2} = \cos(\frac{\pi N}{2}(k+2) + \Phi_0) = \cos(\frac{\pi N}{2} + \Phi_0 + \frac{\pi 2N}{2}) =$$

$$s_{k+2} = -\cos(\frac{\pi N}{2} + \Phi_0) = -I_k$$

This method is also called pseudo sampling, since the I and Q samples do not occur at the same time. There could be potential problems with this method however. If dealing with large frequency offsets with respect to the sampling frequency, this method can create an extra phase shift in the quadra-phase samples, causing aliasing to a negative frequency offset. However, this is not a problem for a GPS signal with reasonable Doppler.

One of the advantages of this method is that, because *Q* and *I* samples are generated in the same circuitry there are no gain or phase imbalances between them as compared to the analog base-band conversion process. In addition, there is only one A/D converter needed in this method. However, there are certain disadvantages that come with using this approach. One is that the aperture time of sampling process must be small with respect to the period of the IF frequency.Given, Sin(x)/x, where x is proportional to the product of the frequency and the aperture time, attenuation occurs if the aperture time is too long (Van Dierendonck 1996).

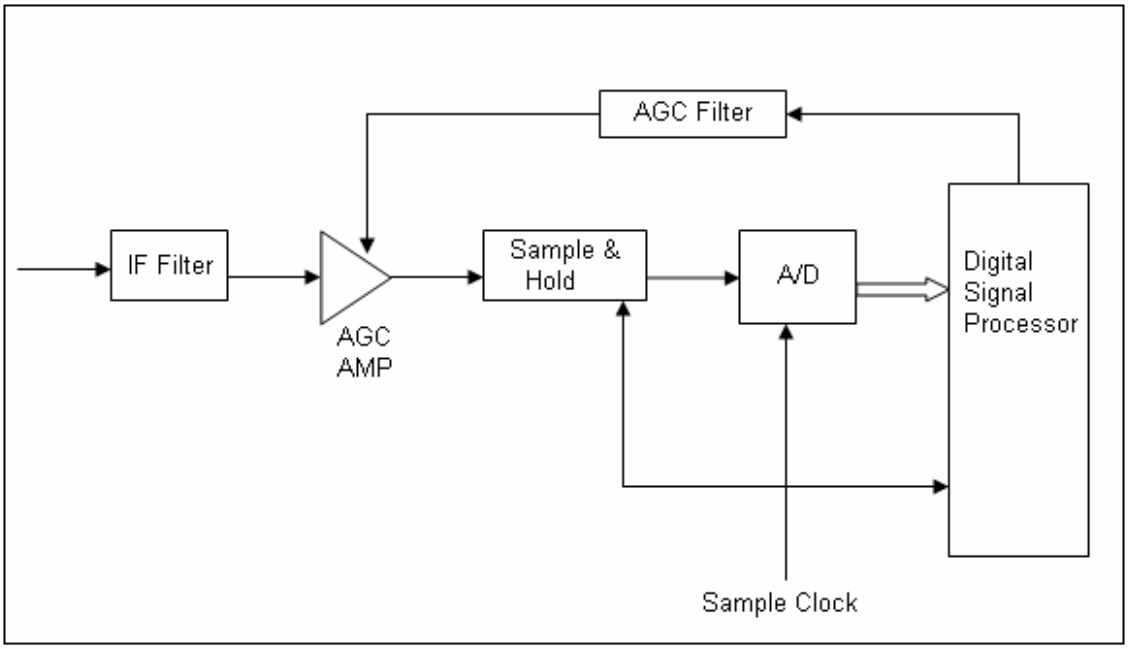

Figure 8: Intermediate frequency sampling process

2.2.2 GPS C/A Code Acquisition

The first step in the operation of GPS receiver is to find a rough estimate of the PRN code offset and carrier Doppler. Acquisition is a coarse synchronization process which determines the estimate of PRN code and Doppler. This information is used to initialize tracking loops for signal tracking and navigation data demodulation. GPS signal acquisition is essentially a two-dimensional search process in which a replica code and a replica carrier are aligned with the received signal.

The correct alignment is identified by measurement of the output power of the correlators. When both the code and the carrier Doppler match the incoming signal, the signal can be correctly de-spread and the data can be demodulated. The results of the

two-dimensional search consist of an estimate of the code offset to within half a chip and a Doppler estimate to within the lock range of the tracking loops. This process is shown in figure 9.

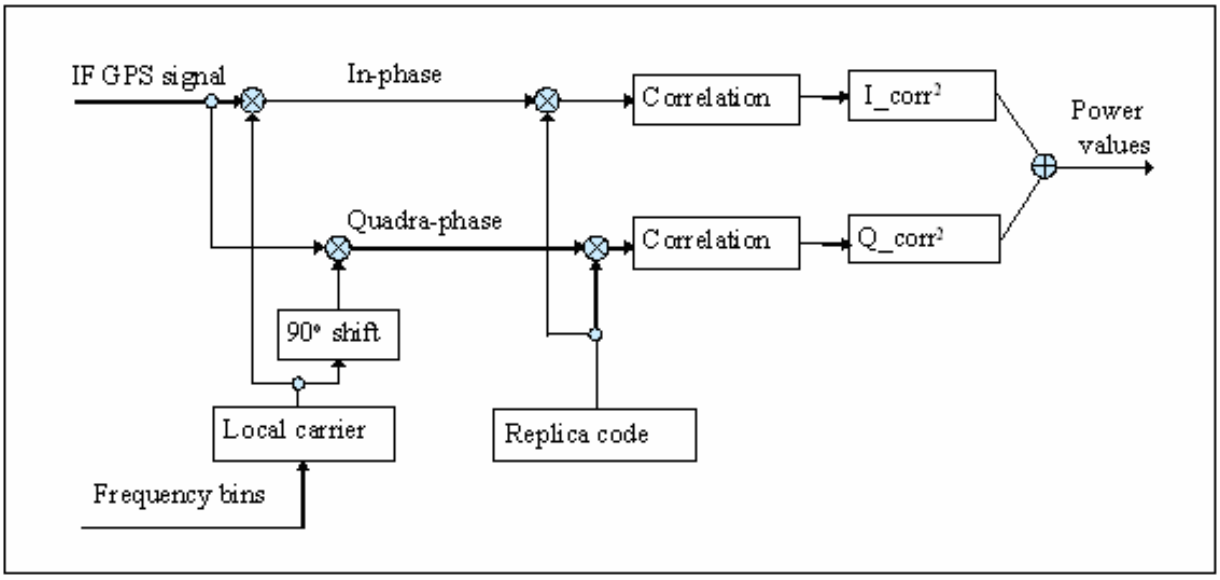

Figure 9 : Block diagram of signal acquisition

The two-dimensional search space covering the full range of the ambiguity of code and Doppler needs to be defined before acquisition can be performed. The Doppler search space can be reduced if the initial estimate of the Doppler is available. If such an estimate is not available, a search space of -5 kHz to +5 kHz is appropriate for GPS signals. The frequency bin size is calculated as follow (Van Dierendonck 1996):

$$\Delta f = \frac{2}{3T} \tag{2.17}$$

where Δf is the size of a frequency bin, in Hz, and T is the coherent integration time (or dwell time) in seconds. As this formula illustrates, there is a trade off between the pre-detection integration time and the speed of acquisition. Longer integration results in

better frequency resolution and higher sensitivity, however it increases the number of frequency bins that needs to be searched. The code search space usually includes all possible code offsets. The resolution of the code search needs to be smaller than half a chip. The sampling frequency is usually used to specify this resolution.

2.2.2.1 Detection Criteria

As was briefly mentioned earlier, acquisition is based on the measurement of correlator output (Krumvieda et al 2002). The correlators provide a measure of the total *I* and *Q* signal value over the coherent integration time (dwell time).

The envelope of the acquisition result $\sqrt{I^2 + Q^2}$ is a measure of the amplitude of the signal. When the local code and incoming signals are aligned, the amplitude of the recovered signal is at a maximum. The In-phase and Quadra-phase components *I* and *Q* are calculated by stripping off the reference code and the carrier from the received signal as shown in figure 10. The envelope is then computed and compared with a threshold. Since the GPS signal is buried in noise, a threshold must correspond to an acceptable probability of false alarm. A false alarm is the probability of detecting a signal from a noisy measurement when in fact such a signal does not exist.

The threshold can be set as follow:

$$V_t = \sigma_n \sqrt{-2 \ln P_{fa}} \tag{2.18}$$

where P_{fa} is the single trial probability of false alarm, and σ_n is the 1-sigma noise amplitude. σ_n is frequently obtained by using a reference PRN which is known to be absent such as PRN 37.

Any cell amplitude that is at or above the threshold is considered to have a signal present. The detection of the signal is a statistical process because each cell contains either noise with the signal present or noise with the signal absent (Krumvieda et al 2002).

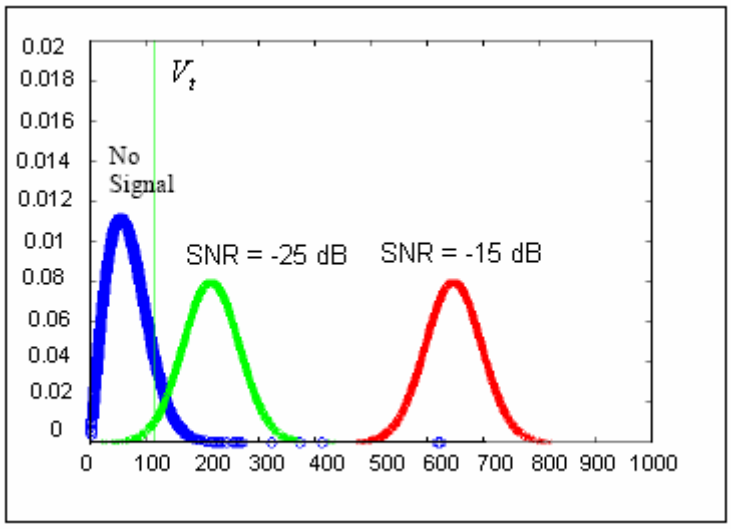

Figure 10: Probability density function for different SNR

Figure 10 illustrates that if the SNR is high, it is easy to set a threshold that provides both low probability of a false alarm and low risk for missed detection. However, for a lower SNR, it is no longer possible because of the significant overlap of the signal distributions.

Looking at Figure 10, the possibility of missed detection is the area under the green curve from the threshold line to left. This is the probability that there is a signal but the threshold is set too high. Probability of false detection is the area under the blue curve from the threshold line to the right. This indicates that there is no signal but since the threshold was set too low, we mistakenly declared the signal present. There is a trade off when setting the threshold to avoid false alarms. If the threshold is set very high to avoid false detections, then there is a high probability that weak signals will not be detected. In GPS this is the typical situation and single trial detection is not effective. It is important to note that when a signal is absent, the envelope has a Rayleigh distribution. Otherwise the envelope has a Ricean distribution (I and Q signals are normally distributed if the noise is Gaussian).

There are several different search methods for the acquisition of GPS signals presented in the literature. Two common approaches are described here.

Tong detector method:

The Tong detector has a reasonable computational burden and is excellent for detecting signals with an expected C/N_o of 25 dB-Hz or higher. Figure 11 presents the Tong search detector algorithm. There are a few parameters to be initialized when using the Tong method based on a receiver's preference of acquisition speed versus probability of detection and false alarm. The variable counter K will give the fastest acquisition speed when set to 1 (Krumvieda et al 2002).

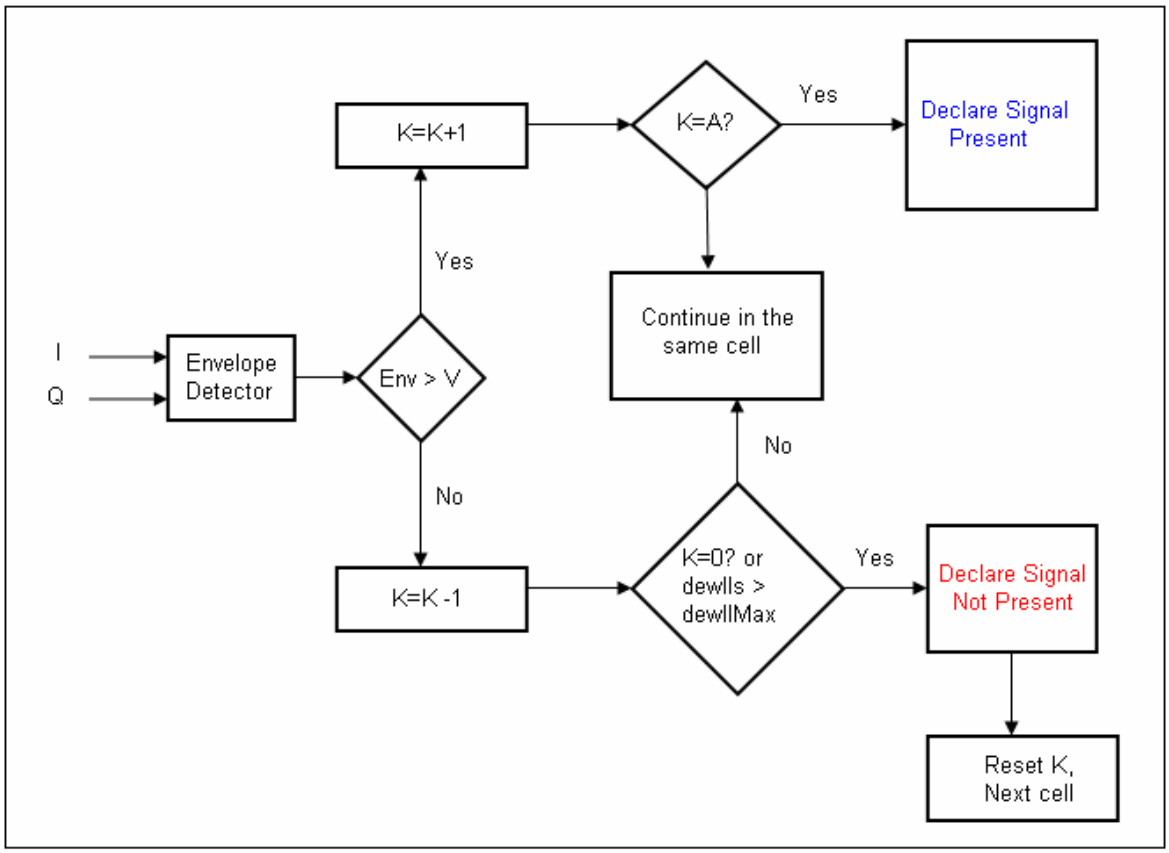

Figure 11: Tong search detector algorithm

The counter variable *K* and confirmation threshold *A* are initialized based on the operational environment. A maximum acquisition speed will be achieved by setting *K* = 1.

If a higher probability of detection and a lower probability of false alarm is required, then *K* may be set to 2 or higher, however, this will increase the time of acquisition. When *K* reaches A the signal is declared present. A typical range for *A* is between 8 and 12, which corresponds to high to low C/N_o values respectively.

The Tong algorithm operates as follows:

- Form a correlation envelope for a given cell every *T* seconds.
- If the correlation envelope exceeds the threshold V_t, then increment *K* by one.
- Else, decrease *K* by one.
- If *K* equals *A* then the signal is declared present and the search is over.
- If *K* equals zero, the signal is not to be present.
- Start the entire process over in the next search-grid-cell.

This method is quite suitable for a hardware receiver where the processing resources are not scarce. DFT acquisition is another method, which is computationally more efficient and faster, therefore more commonly used in software receivers.

DFT acquisition:

The main advantage of the DFT approach is that it calculates the correlation for an entire range dimension (selected Doppler) in a single step (Krumvieda et al 2006). The disadvantage is that when the Doppler is non-zero the reference signal when convolved produces some errors.

The Discrete Fourier Transform (*DFT)* algorithm operates as follows:

- Take the DFT of incoming Signal
- Take the DFT of the reference signal
- Multiply the incoming signal's DFT with the conjugate of the reference signal's DFT.
- Take the inverse DFT of the product, which gives the correlation result in the time domain for all the 1023 code phase offsets.

The DFT of a sampled signal *x(n)* is calculated as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi n \frac{k}{N}) \tag{2.19}$$

Theoretically, multiplying the DFT of two signals and taking the inverse DFT of the product corresponds to a convolution in the time domain. However, since we are interested on the correlation of the incoming GPS signal with the reference signal in the time domain, then this translate to multiplying the conjugate DFT of one signal with the DFT of the other and then taking the inverse DFT of the product.

For the N point DFT of an N-point sequence, N^2 additions and multiplications are required, which is the same as the time domain. However, if the sequence length is limited to a power of two, then using FFT (Fast Fourier transform) $N \log N$ additions and $\frac{N}{2} \log N$ multiplications would be required. This results in a reduction in computational time, at the cost of accuracy. There are several techniques that can be used to change the number of samples to reach this condition, which will be discussed in later chapters (Krumvieda et al 2006).

2.2.3 GPS Signal Tracking

Acquisition produces a coarse estimate of the carrier Doppler and the code offset of the incoming signals. Next, tracking loops start to track variations in the carrier Doppler and code offset due to line-of-sight dynamics between the satellites and the receiver. The tracking process follows the signal and obtains information of the navigation data. When there are phase shifts in the carrier due to the code, such as in the GPS signal, the code must be stripped off. Therefore, to track an incoming GPS signal, both the carrier phase and code offset need to be matched by the locally generated carrier and code. Thus, the lock status of both the carrier lock loop (FLL or PLL) and delay lock loop are required for signal tracking; they must be coupled together as shown in Figure 12 (Tsui & James 2000).

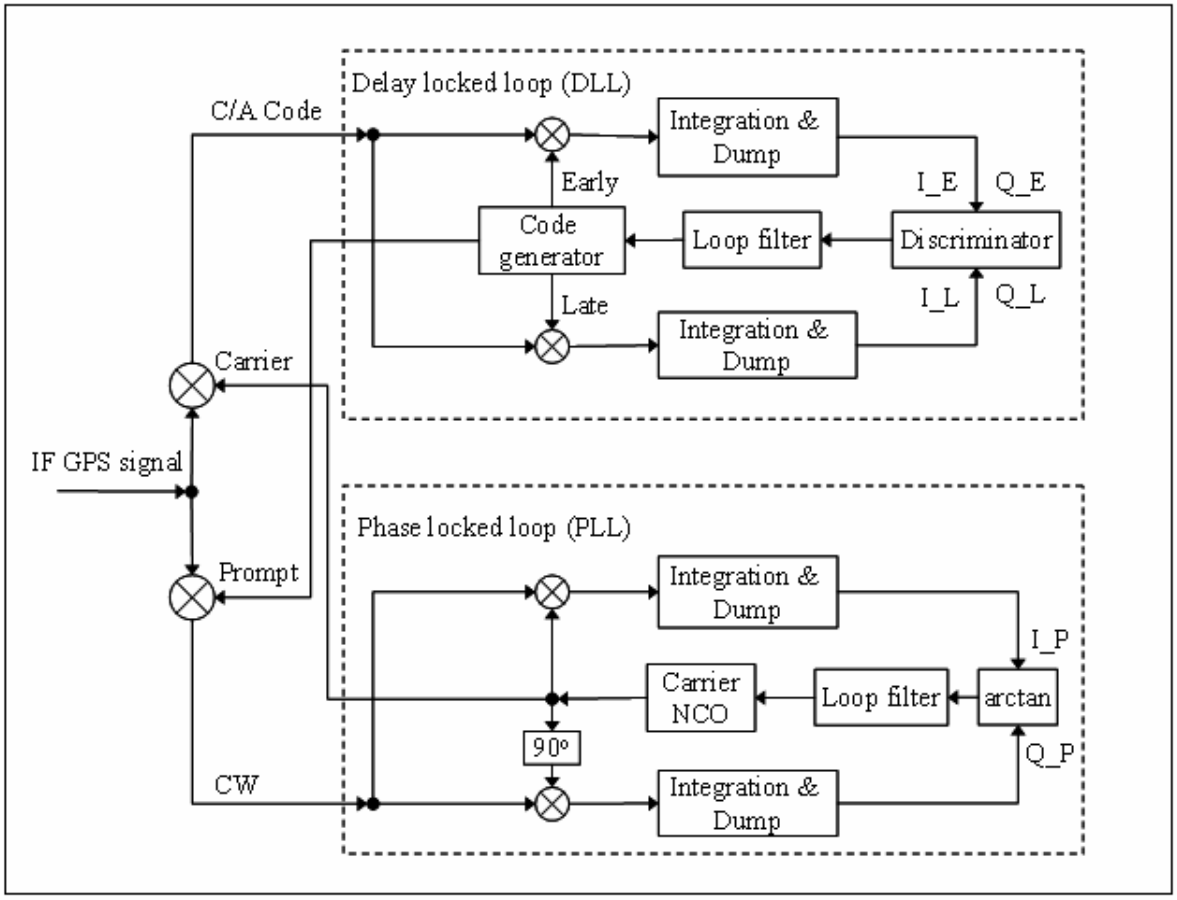

Figure 12: Code and carrier tracking loops

2.2.3.1 Carrier Tracking

Carrier tracking loops are characterized based on the carrier pre-detection integrators, the carrier loop discriminators and the carrier loop filters. These three functions determine the most important performance characteristics of the receiver carrier loop design: the carrier loop thermal noise error and the maximum line-of-sight dynamic stress threshold (Ward 1996).

The carrier loop discriminator defines the type of tracking loop as a Phase Lock Loop (PLL), a Costas PLL (which has tolerance to data modulation on the signal), or a

Frequency Lock Loop (FLL). PLL and Costas loops are more accurate, with the cost being increased sensitivity to dynamics. As the names suggests, PLL and Costas loops generate the phase error while FLL produces the frequency error.

Phase Lock Loops:

If there were no 50-Hz data modulation on the GPS signal, the carrier tracking loop discriminator could use a pure PLL discriminator. However, it is still possible to use the PLL for short integration times through a process of code wipe off. It takes a GPS receiver approximately 12.5 minutes to down load the full navigation message. After this, the receiver can wipe off the navigation message and use the old one as long as the control segment does not upload a new navigation message. Eliminating the navigation bits is done by reversing the sign of integrated In-phase components and Quadra-phase components (Ward 1996).

Costas Loop:

The Costas loop is insensitive to the 50-Hz data modulation in GPS signal therefore it is commonly used in GPS receivers. Table 3 summarizes several GPS receiver Costas PLL discriminators, their output phase errors and their characteristics.

Table 3: Common Costas Loop discriminator used in GPS receivers (Ward 1996)

Discriminator Algorithm	Output Phase Error	Characteristics
$Sign(I_{PS}).Q_{PS}$	$\sin(\Phi)$	Near optimal at high SNR Slope proportional to Signal Amplitude Least computational burden
$I_{PS}.Q_{PS}$	$\sin(2\Phi)$	Near optimal at low SNR Slope proportional to signal Amplitude squared Moderate computational burden
I_{PS} / Q_{PS}	$\tan(\Phi)$	Suboptimal, but good at high and low SNR Slope not signal amplitude dependent Higher computational burden
$ATAN(Q_{PS} / I_{PS})$	Φ	Optimal at high and low SNR Slope not signal amplitude dependent Highest computational burden

Costas PLL loops can be used to detect the bits in satellite data message stream. The I_{PS} samples can be accumulated for the duration of one data bit (20 ms) and the sign of the result is the data bit. However, there will be a 180° phase ambiguity with Costas PLL which will be corrected for during the frame synchronization process. This is done by comparing the known preamble at the beginning of the each sub frame with the data bit stream. If a match with the preamble is found, the bit stream will not be changed. If not, an inverted pattern of preamble is check and if there is match, then the bit stream is inverted as well.

Costas loops are sensitive to dynamic stress; however, they produce the most accurate velocity measurements. Also, for a given signal power level, Costas PLL loops provide the most error-free data demodulation in comparison with FLL. These characteristics

make Costas PLL a very good candidate for GPS receivers. However, a well designed GPS receiver starts tracking with the more robust FLL operated wide band, since it is less sensitive to errors and noise. Then, gradually, it switches to a wideband PLL and finally when the tracking is stable, to a narrow band PLL (Ward 1996).

Frequency Lock Loops:

A FLL is used to estimate the approximate frequency of the GPS signal. It is important the FLL used in a GPS receiver be insensitive to 180° reversals in the *I* and *Q* signals. Ward (1996) states it is usually easier to maintain frequency lock than phase lock when data transition boundaries are unknown (example, during the initial signal acquisition). This is due to the FLL discriminators being less sensitive to situations where some of the *I* and *Q* signals do straddle the data bit transitions (Ward, 1996).

Table 4: Common GPS receiver FLL discriminators (Ward 1996)

Discriminator Algorithm	Output Phase Error	Characteristics
$\frac{sign(dot)cross}{t_2 - t_1}$ Where: Dot = $I_{PS1}.I_{PS2}+Q_{PS1}.Q_{PS2}$ Cross= $I_{PS1}.Q_{PS2} - I_{PS2}.Q_{PS1}$	$\frac{\sin[2(\Phi_2 - \Phi_1)]}{t_2 - t_1}$	Near optimal at high SNR. Slope proportional to signal amplitude Moderate computational burden.
$\frac{cross}{t_2 - t_1}$	$\frac{\sin[(\Phi_2 - \Phi_1)]}{t_2 - t_1}$	Near optimal at low SNR. Slope proportional to signal amplitude squared. Least computational burden.
$\frac{ATAN2(cross, dot)}{(t_2 - t_1)360}$	$\frac{[(\Phi_2 - \Phi_1)]}{(t_2 - t_1)360}$	Four-quadrant arctangent. Maximum likelihood estimator. Optimal at high and low SNR Slope not signal amplitude dependent Highest computational burden

2.2.3.2 Code Tracking

The code tracking loop in the receiver is used to despread the incoming signal from the data bits which are used to provide time of transmission measurements critical for range measurements and subsequently a position solution. Code tracking loops are generally characterized based on the pre-detection integrators, the code loop discriminators and code loop filter. Table 5 summarizes some of the most common non-coherent delay lock loops (DLL) and their characteristics.

Table 5 : Common Delay lock loop discriminators (Ward 1996)

Discriminator Algorithm	Characteristic
$\sum (I_{ES} - I_{LS}) I_{PS} + \sum (Q_{ES} - Q_{LS}) Q_{PS}$	Dot product discriminator. Uses all 3 correlators. Lowest computational burden. For 0.5 chip spacing, produces nearly True error output within ± 0.5 chips.
$\sum (I_{ES}^2 + Q_{ES}^2) - \sum (I_{LS}^2 + Q_{LS}^2)$	Early minus late power Moderate computational burden For 0.5 chip spacing, produces good tacking within ± 0.5 chip of input error.
$\sum \sqrt{(I_{ES}^2 + Q_{ES}^2)} - \sum \sqrt{(I_{LS}^2 + Q_{LS}^2)}$	Early minus late envelope Higher computational burden For 0.5 chip spacing, produces good tracking within ± 0.5 chip of input error.
$\frac{\sum \sqrt{I_{ES}^2 + Q_{ES}^2} - \sum \sqrt{I_{LS}^2 + Q_{LS}^2}}{\sum \sqrt{I_{ES}^2 + Q_{ES}^2} + \sum \sqrt{I_{LS}^2 + Q_{LS}^2}}$	Early minus late, normalized by early plus late envelope Highest computational load

The correlation process mixes this input with the local early (E), punctual (P), and late (L) code replicas in both the In-phase and Quadra-phase arms to get the IE, QE, IP, QP, IL, and QL. Figure 13 shows the correlation power or the correlation envelope of the early component and the late component contains the code mismatch information $\Delta\tau$;

thus, the IE, QE, IL, and QL are usually used in the DLL discriminator to obtain the estimate of $\Delta\tau$.

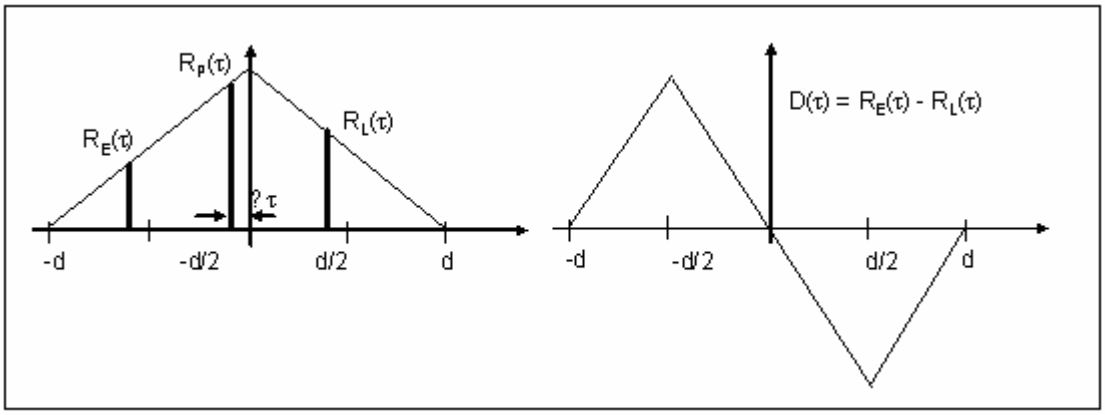

Figure 13: Code mismatch early, prompt and late components

The result of the discriminators, energy in the early and late branches, which has been differenced, is filtered, used to drive the NCO, which in turn clocks the PRN code generator. The results indicate to the NCO that which branch, early or late, has more power and whether the NCO needs to speed up or slow down the locally generated PRN code (Ward 1996).

2.2.3.3 Loop Filters

The objective of loop filters is to reduce the noise so the receiver can better estimate the desired signal. In this section, a very brief description of the filters used in typical GPS receivers is given. For more information, refer to Ward 1996.

Table 6 summarizes the characteristics of some available loop filters. The order of the filter used is based on the requirements of the receiver. For example, for a receiver which

is subject to acceleration dynamics, a third order loop for the PLL is selected since it is insensitive to acceleration.

Table 6: Loop filter characteristics (Ward 1996)

Loop Order	Noise Bandwidth	Typical filter value	Steady-state Error	Characteristics
First	$\frac{\omega_0}{4}$	ω_0 $B_n = 0.25\omega_0$	$\frac{(\frac{dR}{dt})}{\omega_0}$	Sensitive to velocity Used in aided code loops/aided carrier loops Unconditionally stable at all noise bandwidth
Second	$\frac{\omega_0(1+a_2^2)}{4a_2}$	ω_0^2 $a_2\omega_0 = 1.414\omega_0$ $B_n = 0.53\omega_0$	$\frac{(\frac{dR^2}{dt^2})}{\omega_0^2}$	Sensitive to acceleration Used in aided/unaided carrier loops Unconditionally stable at all noise bandwidth
Third	$\frac{\omega_0(a_3b_3^2 + a_3^2 - b_3)}{4(a_3b_3 - 1)}$	ω_0^3 $a_3\omega_0^3 = 1.1\omega_0^2$ $b_3\omega_0 = 2.4\omega_0$ $B_n = 0.7845\omega_0$	$\frac{(\frac{dR^3}{dt^3})}{\omega_0^3}$	Sensitive to jerk stress Used in unaided carrier loops Remains stable at $B_n^- \leq 18Hz$

2.2.4 Pseudo-range derivation

An ambiguity remains in the offset of the data bits after signal acquisition is carried out. This is because the receiver has no timing information on the offset of the transmitted data bits. The beginning of the C/A code is known but there is no timing information on where the beginning of the data bit is (it has a length of 20 C/A code cycles). Failure in detecting the right data bit will lead to failure in bit frame synchronization and failure in recovering the navigation message from the signal.

There are several methods that are used for data bit synchronization. One of the most common approaches is the histogram method. In this method, a data bit is divided into 20

ms segments corresponding to the 20 C/A-code cycles. The receiver then tries to sense a sign change between each of these C/A codes. If a sign change is sensed, a corresponding histogram cell count is incremented until a count in one specific cell exceeds the other 19 bins by a pre-specified amount (Van Dierendonck 1996).

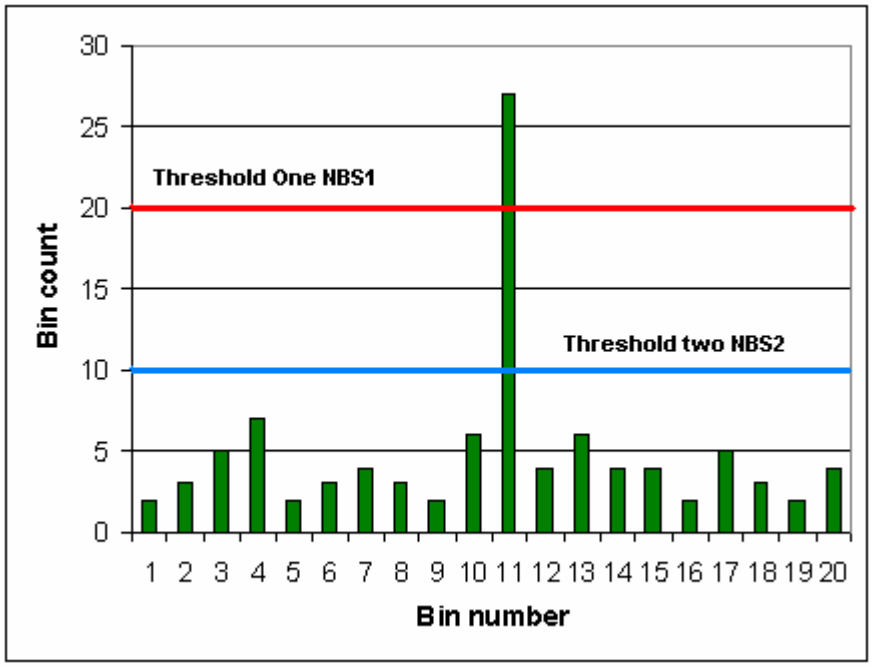

Figure 14: Diagram of a result of Bit synchronization

The algorithm operates as follows:

1. Create 20 cells and set a counter for each cell.
2. Initialize all counters to zero
3. Increment the appropriate cell counter when a sign change is observed.
4. Continue until one of the following conditions is met:
    a. Two cell counts exceed threshold two
    b. Loss of lock
    c. One cell count exceeds threshold one
5. If
    a. Condition (a) is met; conclude that the bit synchronization has failed. Go back to step two.
    b. Condition (b) is met; try to re-establish lock. And start over again.
    c. (c) occurs, declare bit synchronization successful, and the C/A code epoch count is reset to the correct value

The calculation of the thresholds is summarized briefly below. The more extensive explanation can be found in (Van Dierendonck 1996).

The probability of making an error in determining a sign change at a desired signal of the noise ratio (SNR) s as follows:

$$P_{esc} = 2P_e\left(1 - P_e\right) \tag{2.20}$$

where

$$P_e = erfc'\left\lfloor\sqrt{2(S/N_0)T}\right\rfloor \tag{2.21}$$

and

$$erfc'(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-y^2/2} dy \tag{2.22}$$

The number of entries, N_{bs}, in a cell has a binominal distribution. Over T_{bs} seconds, the average number of sign changes (bit transitions) is $25T_{bs}$ so that in a correct cell

$$NBS_1 = 25T_{bs} \tag{2.23}$$

and, in other cells

$$\overline{N}_{bs} = 50T_{bs}P_{esc}\,. \tag{2.24}$$

The thresholds, as well as the time interval T_{bs}, are chosen to provide a good spread between them at a desired SNR. Therefore, given NBS_1, one needs to select NBS_2 and T_{bs} for a desired SNR so that the following inequality holds:

$$25T_{bs} - 3\sqrt{50T_{bs}P_{esc}(1-P_{esc})} \geq NBS_2 \geq 50T_{bs}P_{esc} \tag{2.25}$$

The next step is frame synchronization. This is required in order to process the GPS data and recover the navigation message. In the following few paragraphs, this process will be briefly described.

As was discussed earlier in section 2.4, the GPS navigation message frame structure is partitioned into five sub-frames. Each sub-frame is further subdivided into ten 30-bit words with the two leading words being the telemetry (TLM) word and the handover (HOW) word.

Several algorithms may be used for frame synchronization. Below, is the algorithm employed in the software receiver used herein.

[1] Search for preamble or its inverted bit pattern.

[2] If found, a check is required to verify that the pattern is actually the preamble and it is the beginning of a 30-bit word.

[3] Collect the following 22 bits and checking parity. If parity check does not pass, the candidate preamble is discarded.

[4] There are also other legitimate patterns at the beginning of other words, so additional checks are required. If the correct TLM word exists, the following word must be a HOW word that contains a truncated Z-count. The first eight bits of this truncated Z-count can also resemble a preamble.

[4] Check for parity on the HOW word. If it did not pass, restart the whole process again.

Test to verify the HOW word:

[5] If the HOW test passed, demodulation of the other words can start, and be stored in memory.

[6]A final reliability check on the next preamble and the next Z-count confirms the frame synchronization. This is to check if the preamble is in the right place and the Z-count increments by one.

The next step is to calculate the pseudoranges. A pseudorange measurement is derived based on the following equation

$$\rho(t) = c\left[t_u(t) - t^{(s)}(t-\tau)\right] \tag{2.26}$$

where

$t_u(t)$ is the time of the signal arrival measured by the receiver clock.

$t^{(s)}(t-\tau)$ is the time of signal transmission by the satellite clock where it can be measured based on the Z count.

$$t^{(s)}(t-\tau) = \text{Z - count} + \text{number of navigation bits} + \text{number of C/A codes} + \text{number of whole C/A code chips} + \text{fraction of C/A code chip} \quad (2.27)$$

The process of pseudorange derivation is shown in figure 15 (Misra & Enge 2001). The main goal is to establish the transmission time at the time of measurement. As shown, the arrival time $t_u(t)$ kept by an inner clock is defined by a transition of the receiver clock. In general, these transitions occur sometime in the middle of a C/A code chip. The Z-count, which is also included in the navigation message is a measure of Satellite time. Z-count increments every 1.5 seconds, and the navigation message starts a new sub-frame every 6 s.

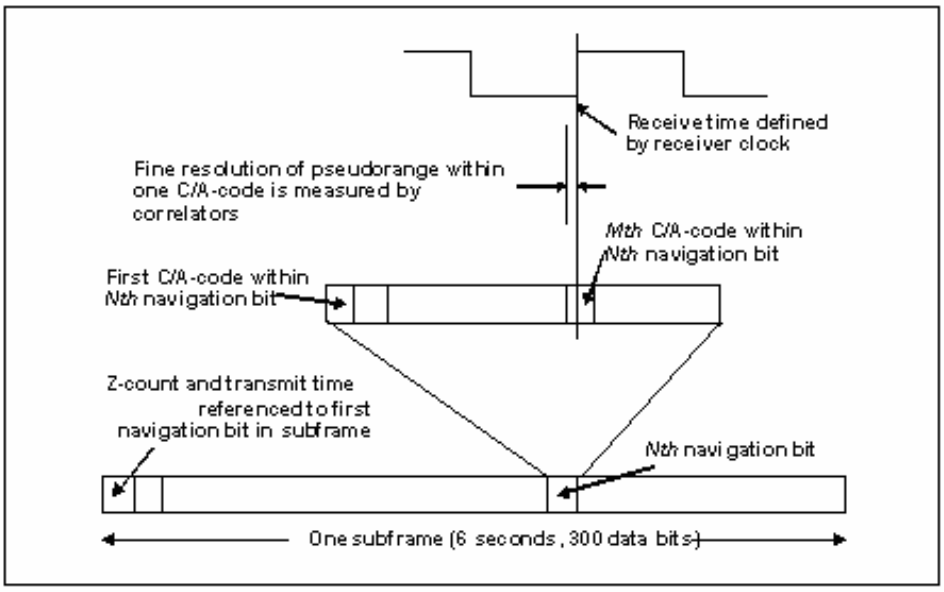

Figure 15: Pseudorange derivation

Since the Z-count establishes satellite time at the beginning of each sub-frame, the transmission time is the Z-count plus the whole number of C/A-code chips since the beginning of the sub-frame. The elapsed time as stated in equation 2.27 is measured by using the whole number of navigation bits, the whole number of C/A-codes since the beginning of the current navigation bits, the number of whole C/A-code chips since the beginning of the current code, the fraction of the current chip. The last two are measured by the DLL and the rest are measured by counters in the bit synchronization and sub-frame synchronization module in the receiver.

Later, all these pseudoranges are combined through the least-squares process to form a navigation solution. An explanation of the theory of least-squares has been discussed in many references and is outside of the scope of this work.

Chapter 3

Real-time GPS Receiver Design

3.1 Computational Bottlenecks

The most computationally expensive task for a GPS receiver is the IF signal processing, including Doppler removal and correlation with the local code. A receiver typically needs to process GPS data acquired at a sampling rate of 2.5 MHz to 5 MHz for C/A code. Additionally, the receiver must perform other operations such as tracking and solution calculation in parallel with the signal processing operations. Although the latter tasks typically run at a slow rate (0.1 to 10 Hz), the computational requirements must still be considered, as they consume computational resources within the receiver. Furthermore, the additional processing requirements for interfacing with an RF front-end in real time must be considered. To satisfy these requirements, it is assumed for purposes of this work that only 50% of the computer's resources are available to the signal processing component of the receiver. This is likely a conservative benchmark. To illustrate the number of operations required, Table 7 shows the number of computations needed in the receiver to process 1 ms of data with different sampling rates. The results given are for a receiver tracking six satellites, the computation of three correlation values (early, prompt and late) with 2.5, 5 and 10 MHz sampling rates.

Table 7: Computational load of the receiver to track six satellites

Sampling Rate	Real Samples			Complex Samples		
	Sin and Cos generation	Multiplication	Addition	Sin and Cos generation	Multiplication	Addition
2.5 MHz	30,000	120,000	90,000	30,000	150,000	120,000
5 MHz	60,000	240,000	180,000	60,000	300,000	240,000
10 MHz	120,000	480,000	480,000	120,000	600,000	480,000

The results of Table 7 demonstrate the increase of the computational load as the sampling rate increases. Therefore, it is recommended to choose the minimum sampling rate appropriate for the application of the software receiver.

Table 8 illustrates that there is a direct relationship between the number of satellites being tracked and the computational load in the receiver. This is clearly expected and it nevertheless shows the difficulty the software receiver faces.

Table 8: Computational load of the receiver for tracking 6, 8 and 12 satellites

Sampling Rate	Real Samples			Complex Samples		
	Sin and Cos generation	Multiplication	Addition	Sin and Cos generation	Multiplication	Addition
6	30,000	120,000	90,000	30,000	150,000	120,000
8	60,000	240,000	180,000	60,000	300,000	240,000
12	120,000	480,000	480,000	120,000	600,000	480,000

Heckler & Garrison (2006) showed that, with normal integer arithmetic, a GPS software receiver cannot operate in real time. As we have discussed earlier, the main computational burden is due to the correlation that has to be performed. A receiver

typically has three correlators. Therefore, for a 12 channels receiver, there is need for a total of 36 correlators. Since the period of the C/A code is 1 ms, there needs to be 36,000 correlations per second performed in order for the receiver to operate in real-time. Based on the assumption of only 50% of available CPU power to perform these correlations, one has to assume that a receiver needs to perform 72,000 correlations per second. A desktop computer purchased in 2006 might have a processor running at 2.0 GHz, a rate of 2×10^9 clock cycles per second. Such a processor could allot 27,777 clock cycles to achieve a single correlation (Heckler & Garrison 2006).

On a Pentium 4 processor, an integer addition takes one clock cycle. A multiplication takes 14 clock cycles (Intel Corporation 2006). A cycle count for a GPS software receiver that uses a 4.092 MHz sampling rate is approximately 130,000 clock cycles versus the limit of 27,777 reported earlier (Heckler & Garrison 2006). This clearly shows that correlation algorithms based on integer arithmetic cannot work for a real-time GPS software receiver.

3.2 Doppler Removal

As mentioned above, during the tracking of one satellite and assuming a sampling rate of 5 MHz and complex GPS data, a receiver needs to generate 5000 (1 ms, length of C/A code) sine and cosine values and multiply these by 5000 samples of the incoming data. Computing all of these complex values in real time is not possible with the current processors in the software receiver. There are two major issues that need to be addressed, first is generation of the sine and cosine values at the desired frequency, the second the mixing of the GPS data with sine and cosine values. These two steps will be covered in more detail in next few paragraphs.

3.2.1 Generation of sine and cosine values

Computing a large number of sine and cosine values in real time is computationally too expensive for a GPS software receiver. This is a challenging task in software since it does not have the same level of parallelism as in hardware. A solution to this problem is to first calculate these values and to store them in memory. Later, they can be used in real time to perform the Doppler removal. There are two methods that have been developed to do this more efficiently.

A table-look-up method reduces this time significantly. In this method, sine and cosine values for all possible phases are calculated and stored in an array. The software needs to use the estimated frequency of the GPS signal to index into this array and obtain the right sine and cosine value.

However, the indexing requires that the data be processed on a sample by sample basis and the memory look up in each step is time consuming. So despite the fact that this

method enhances performance, the gain is not enough for the receiver to perform in real time. This is the method that was used in the prior version of the software receiver used herein (Ma et al 2004) Better performance (30% improvement) can be accomplished with the second method, however.

The second method is called the table grid method. In this method sine and cosine values are computed based on some grid frequencies and saved in a table (Ledvina et al 2003) .With the table grid method, a C/N_0 decrease is expected from using an inexact frequency. However, this has no effect on the accuracy of the solution, nor does it significantly reduce the tracking sensitivity. Ledvina et al (2003) shows that the worst case SNR loss in the case of T=0.001 s is 0.44 dB where T is the coherent integration time.

Since table frequencies are not as precise as the frequency of the incoming signal, there will be a need to do an extra phase rotation at the end of the correlation to compensate for the frequency error. The theory of this rotation is described below.

In the ideal case, I and Q represent the complex GPS signal received from a single satellite, having the form

$$I = AC\cos(2\pi f_r T_s k + \Phi_0) \tag{3.1}$$

$$Q = AC\sin(2\pi f_r T_s k + \Phi_0) \tag{3.2}$$

where A is the average amplitude of the carriers, C is the PRN of the satellite, f_r is the observed carrier frequency, T_s is the sampling frequency, k is an integer sampling unit, and Φ_0 is the initial carrier phase.

Mixing with an ideal local carrier, the quantities *I'*and *Q'* (In-phase and Quadra-phase after ideal Doppler removal) are given as follows (brief derivation given only for *I',* it is analogous for *Q'*):

$$\begin{aligned} I' &= I\cos(2\pi\hat{f}T_s k + \hat{\Phi}_0) + Q\sin(2\pi\hat{f}T_s k + \hat{\Phi}_0) \\ I' &= AC\cos(2\pi(f_r - \hat{f})T_s k + \Phi_0 - \hat{\Phi}_0) \\ I' &= AC\cos(2\pi\partial f T_s k + \partial\Phi) \end{aligned} \tag{3.3}$$

$$\begin{aligned} Q' &= -I\sin(2\pi\hat{f}T_s k + \hat{\Phi}_0) + Q\cos(2\pi\hat{f}T_s k + \hat{\Phi}_0) \\ Q' &= AC\sin e(2\pi(f_r - \hat{f})T_s k + \Phi_0 - \hat{\Phi}_0) \\ Q' &= AC\sin(2\pi\partial f T_s k + \partial\Phi) \end{aligned} \tag{3.4}$$

In the above, $\hat{f}$ is the frequency of the local carrier being used for demodulation, and $\hat{\Phi}_0$ is the initial phase offset of that local carrier.

After correlating *I'* and *Q'* with the local PRN code and integrating for some period NT_s, *I''* and *Q''* are the final correlation values for the In-phase and Quadra-phase channels:

$$\begin{aligned} I'' &= AR(\tau)\frac{\sin(\pi\partial f T_s N)}{\pi\partial f T_s}\cos(\pi\partial f T_s N + \Phi_0 - \hat{\Phi}_0) \\ Q'' &= AR(\tau)\frac{\sin(\pi\partial f T_s N)}{\pi\partial f T_s}\sin(\pi\partial f T_s N + \Phi_0 - \hat{\Phi}_0) \end{aligned} \tag{3.5}$$

These values are the ideal post-correlation values, and are together referred to as Equation 3.5.

In contrast to the ideal frequency removal given above, the results of using the frequency grid table method are given below. Using this method, only an approximate frequency reference f_T is available, where $\hat{f} = f_T + \partial f_T$ and ∂f_T is the difference between the actual

frequency and the table frequency. Using this approximate frequency, the correlation values are calculated in a similar manner to that used before:

$$
\begin{aligned}
I'' &= AR(\tau)\frac{\sin(\pi(f-\hat{f})T_s N)}{\pi(f-\hat{f})T_s}\times \\
&\cos(\pi\partial f_T T_s N + \pi\partial f_0 T_s N + \Phi_0 - \hat{\Phi}_0) \\
Q'' &= AR(\tau)\frac{\sin(\pi(f-\hat{f})T_s N)}{\pi(f-\hat{f})T_s}\times \\
&\sin(\pi\partial f_T T_s N + \pi\partial f_0 T_s N + \Phi_0 - \hat{\Phi}_0)
\end{aligned}
\tag{3.6}
$$

In order to match the ideal form of Equation 3.5, one needs to rotate the resultant correlation values according to the following relations, based on the frequency error between the table and the desired local frequency:

$$
\begin{aligned}
I'' &= I''.\cos(\pi\partial f_T N + \hat{\Phi}_0) + Q''.\sin(\pi\partial f_T N + \hat{\Phi}_0) \\
Q'' &= Q''.\cos(\pi\partial f_T N + \hat{\Phi}_0) - I''.\sin(\pi\partial f_T N + \hat{\Phi}_0) \\
\hat{\Phi}_0 &= \Phi_p + 2\pi f_p T_p
\end{aligned}
\tag{3.7}
$$

where Φ_p and f_P are the phase and the frequency of the previous correlation interval. This method was proposed by Ledvina et al (2003).

In addition to forming the sine and cosine values, mixing the signals is a time consuming task as well. It requires 20,000 multiplications and 10,000 additions of the incoming data against the sine and cosine values for one satellite in 1 ms. MMX instructions are used to perform this operation. The resultant improvement in performance is shown later.

The next step after Doppler removal is the correlation of GPS data with the local code. Assuming a 5 MHz sampling rate, this is equal to performing 30,000 multiplications and

30,000 additions per 1 ms for one satellite (early, prompt and late correlation). Results of the performance tests show that by using standard (integer math) C operators, this task can not be completed in real time for more than one satellite. MMX instructions can help the software to speed up these operations. This will be explained in more detail in the next section.

3.2.2 MMX Technology

SIMD (Single Instruction, Multiple Data) is a technique used to give a higher level of parallelism to the software. It was first used in the large scale super computers, however, with the increase in demand for higher speed processing, mostly driven by gaming and multimedia programming smaller-scale SIMD operations have become available in personal computers.

Figure 16 gives a visual and simple description of the difference between SIMD versus the more typical SISD device (Single Instruction, Single Data Stream).

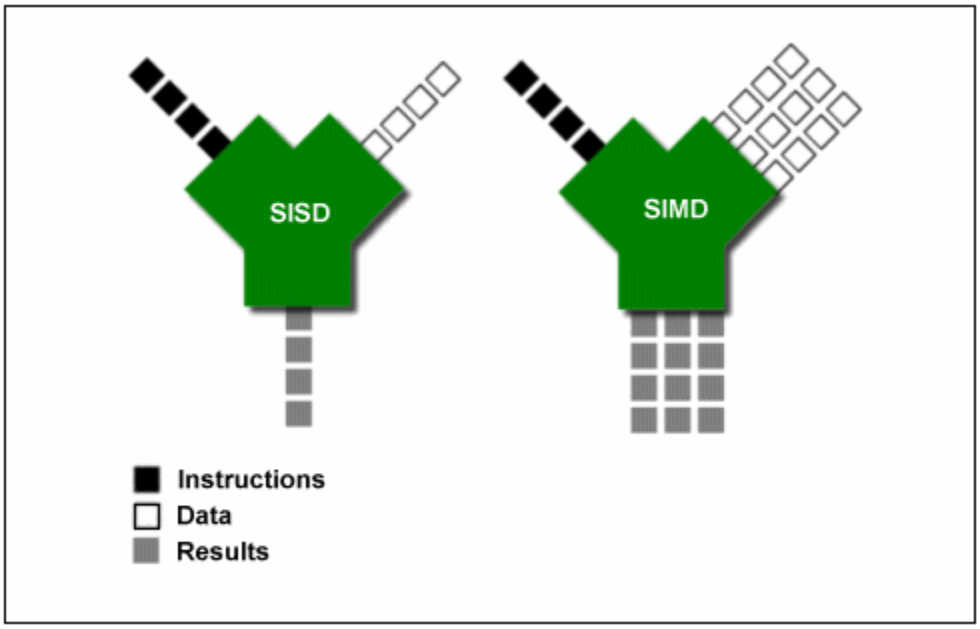

Figure 16: SIMD instruction versus SISD

As Figure 16 clearly illustrates, SIMD exploits a property of the data stream called data parallelism. Data parallelism can be used when a large mass of data of a uniform type needs the same instruction performed on it (as is the case in GPS). SIMD computing is also called vector processing. The reason is that the basic unit that SIMD operates on and shows its power with is a vector.

A normal CPU operates on scalar data, one sample at a time. However, vector processing techniques treat an array of data as one and then process them in parallel. The other term that needs to be discussed is “packed data” format. This references the vectors that used by SIMD operations.

It is important to use the correct data type based on the application of the software since it will have the direct effect on the level of parallelism that is performed when using SIMD instructions.

MMX technology is a set of SIMD (Single Instruction, Multiple Data) instructions available on the Intel platform and on x86-compatible processors. MMX has eight 64-bit registers (MM0-MM7) that can be used to load data and adds 57 new instructions to the x86 processor. It supports 3 basic data types as follow:

1) Packed byte (8 bit), which allows eight samples to be processed at the same time.
2) Packed word (16 bit), allowing four samples to be processed simultaneously.
3) Packed double word (32 bit), which reduces the parallelism to only two samples.

If GPS data is represented with eight bit (char) variables then MMX can load eight samples in parallel from memory and perform multiplications and additions in parallel on this data. Heckler & Garrison (2003) was the first to propose the use of MMX technology in GPS software receivers.

There are two other extensions to MMX that is offered by Intel in their Pentium 3 and 4 platforms, namely SSE (Streaming SIMD Extension) and SSE2 add more instructions, 70 new ones, and also increases the size of the registers to 128 bits. These new instruction sets and larger registers automatically, and with the least amount of code modification, double the size of parallelism in the application. In the software receiver used herein, both MMX and SSE/SSE2 instructions are used to enhance the speed of code. The results of these improvements are presented in the next chapter.

3.2.3 Code Correlation using SIMD

Correlation across the C/A code boundaries may introduce processing loss due to data bit boundaries. The reduced magnitude correlation values negatively influence PLL tracking and have a major impact on obtaining bit-lock. The incorrect correlation values lead to a

change in the discriminator used to determine the data bit transition leading to a periodic loss of bit-lock. The regular loss of bit-lock leads to an inability to properly decode the GPS data message and perform a navigation solution. Heckler & Garrison (2003) proposed a solution to this problem by achieving correlation in three steps:

1) Correlate the GPS data with the local code until the code roll-over point.
2) Perform tracking and use new Doppler to correlate the remaining data. Add this value to the value of the first step in the next epoch to achieve a 1 ms correlation value.
3) Perform the tracking on this 1 ms of data.

Another important technique used in the receiver to enhance its performance is the use of the pre-computed C/A code table. Two periods of C/A code were computed one after the other in memory at the appropriate sampling frequency. Therefore, there is no need to do a wrap around while performing a 1 ms correlation. It is noted that the C/A code Doppler was not considered.

Inside a GPS receiver, a second order DLL (Delay Lock Loop) is used to track the C/A code of the signals. Although all the discriminators discussed by Ward (1996) can be used in the receiver, the normalized dot product discriminator has been chosen. This discriminator, for 1/2 chip correlator spacings, produces nearly true error outputs within $\pm 1/2$ chip of input error. The discriminator output is calculated as

$$(\sum (I_E - I_L) \times I_P + \sum (Q_E - Q_L) \times Q_P) / 2N \tag{3.8}$$

where N is a normalization factor, given as

$$N = {I_P}^2 + {Q_P}^2 . \tag{3.9}$$

This is described by Julien (2005). I_P and Q_P are the In-phase and Quadra-phase values of correlation.

In order to save time and increase performance, the early minus late code can be calculated off line, stored and then used during the correlation process. This reduces the number of correlators per satellite from six to four, since one requires one correlator for both I and Q, and increases the computational performance of the receiver. The result is pure code tracking. Carrier aiding, to date, has not been implemented in the real-time version of the software.

3.2.4 Software Architecture

This section looks at the overall architecture of the software and the work that has been done in the receiver. It reviews the data flow of each component of the software in order to give a better understanding of the work that has been performed for this thesis.

Receiver control is performed within the main function. The software can be configured to work both in real-time and post-processing mode as shown in Figure 17. In the real-time mode, the software needs to initialize the data handling routines for interfacing with the DAQ card. Each step is commented in the code for easier future modifications. The receiver control then moves the status of the software to the acquisition. The acquisition component can perform blind search or can perform a more selective search if prior knowledge of the availability of satellites is available. Next, the receiver status is changed to the tracking. The receiver stays in this mode unless there are not enough satellites to track; in this case, the receiver reverts to the acquisition mode. Both acquisition and tracking use the same code for the Doppler removal and correlation. The

software consists approximately of 7000 lines of code of which almost 50% is new code or modified code. The original software receiver was designed to run in post processing mode (Ma et al 2004). Therefore the data handling was not done in a manner suitable for real time operation. In the real time version, a complete rewrite of the data and channel handling has been implemented that makes the receiver better suited for real-time performance. Also, the previous version did not have any component to interface with the hardware. This code was added for the real-time version of the receiver. Another major difference between the previous version and the new version is the Acquisition component. The new version uses completely different algorithms for performing the search for satellites. This means a better channel management and more efficient use of FFT algorithms.

A major change was made to the old version for the Doppler removal and correlation component of the receiver. The receiver was modified to process a block of data instead of sample by sample. This translated into calling the tracking loop functions and solution calculation algorithms in a completely different way. The receiver was redesigned to use its main function as a receiver control task and manage its channels, acquisition, tacking and other component from the main part.

Another major contribution to the receiver design was modifying the basic data elements (sample sizes) and data manipulation algorithms that were used in the Doppler removal and correlation component. This led to a redesign of the data gathering code that was written for the FPGA. The design of the filter and data handling was all written in C and tested for performance and then implemented for the FPGA. As it was discussed earlier, the previous version used a simple integer arithmetic (which is sufficient for the post

processing mode), However, the new version takes advantage of the MMX algorithms. This change adds much complexity to the code.

The tracking loops were also rewritten to use more efficient discriminators and also to be able to handle processing of block data instead of the sample-by-sample technique that was used in the previous version. Figure 17, 18 and 19 show block diagrams of the components that were discussed above.

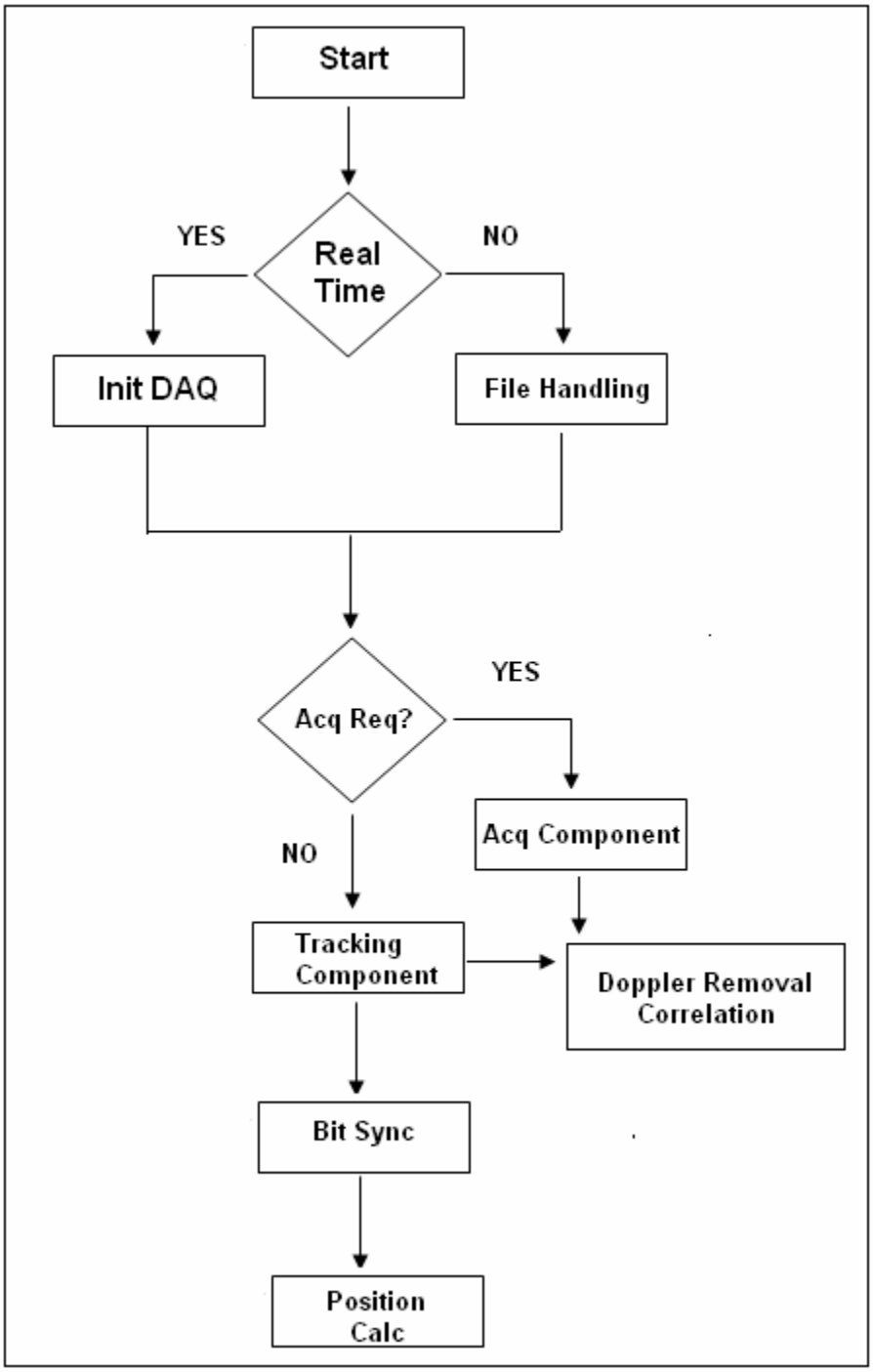

Figure 17: Over all Architecture of the software

The acquisition component was rewritten to take advantage of the FFT algorithm for this receiver. The algorithm is described in a previous section. Figure 18 illustrates the acquisition component in the receiver. Before acquisition, an FFT of local code is created and stored. The first step of the acquisition is to initialize the data structures which represent each channel. The next step is to calculate the noise floor which will be used to detect the presence of the signal. Next, a search in each Doppler bin for the satellites is performed. The search result is then stored in the channel data structure. Thus by comparing the result with the threshold (as described in a previous section) a satellite declared found and the corresponding channel data structure gets updated with related information, for example the Doppler and code phase measurements.

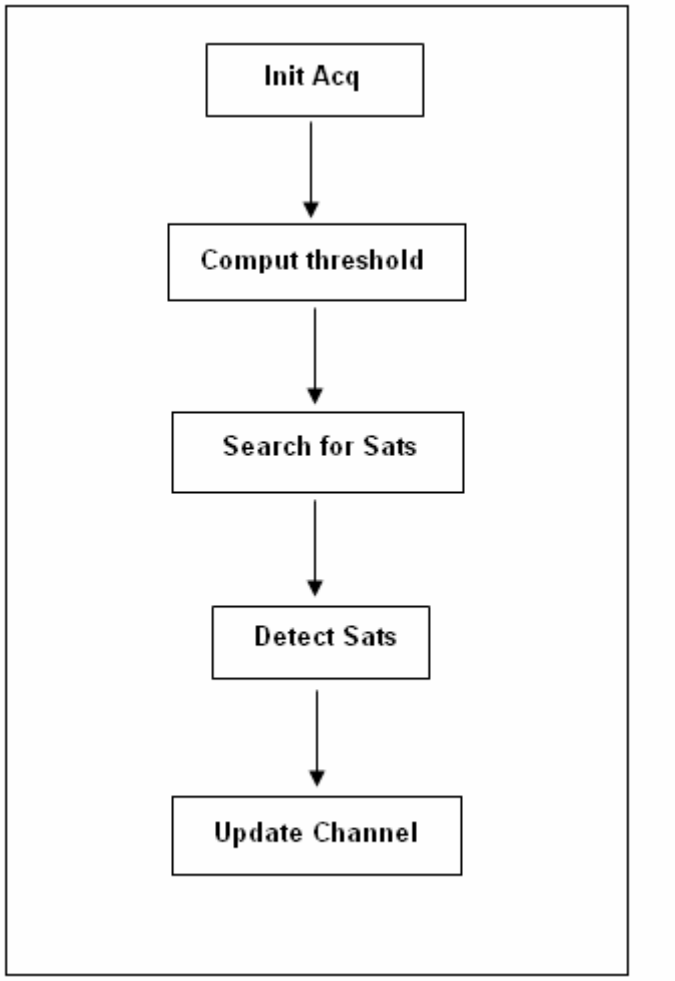

Figure 18: Acquisition flow chart

Figure 19 illustrates the Doppler removal, correlation and tracking components of the receiver. As discussed earlier, the Doppler removal and code correlation are performed up to the code roll-over point. Then the correlation results are passed to the tracking component. Next, the new Doppler and code phase are used to correlate the remaining data and this value is added to the value of the first step in the next epoch to achieve a 1 ms correlation value.

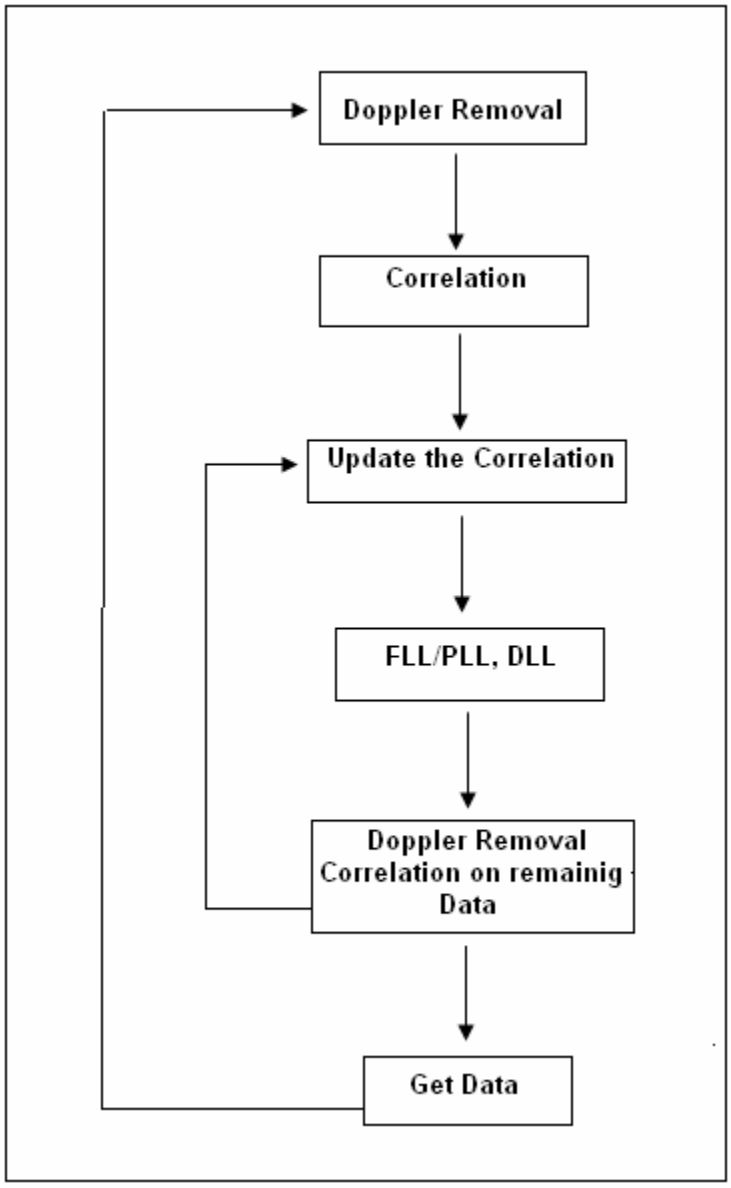

Figure 19: Doppler removal, Correlation and tracking flow chart

Chapter 4

Test Set Results

4.1 Test Set Up

One of the main factors that affect the performance of GPS software receivers is the sampling rate of the GPS signal. The sampling rate contributes to the computational load of the overall system. The higher the sampling rate, the more data there is to be processed and consequently the need for computer resources increases. Therefore choosing the right sampling rate is a necessary initial step in developing a software receiver. As stated in previous chapters, the GPS L1 signal is transmitted at 1575.42 MHz and has a null-to-null bandwidth of 2 MHz. This signal is phase modulated with a Pseudo-Random Noise (PRN) code with a 1.023 MHz chipping rate. The sampling rate should not be synchronous to the code rate as this makes it difficult to obtain fine distance resolution (Tsui 2000). The other factor in choosing the sampling rate is the Nyquist rate. The Nyquist sampling theorem requires that the minimum sampling bandwidth be twice the information bandwidth. Therefore, in case of a GPS L1 C/A code signal, a minimum 2 MHz sampling rate is required. However, since there are no ideal filters it is usual to recommend a sampling rate that is 2.5 times the information bandwidth in order to mitigate the effects of filter roll-off. This means that a 5 MHz sampling rate is a preferred

sampling frequency for the GPS C/A code signal. However, it is possible for the GPS C/A code signal to be sampled at rates lower than 5 MHz.

Pany et al (2003) shows that it is not necessary to retrieve the whole information (the code sequence) of the signal to reconstruct the auto correlation function. Therefore, the GPS signal may be tracked with a lower sampling rate to improve processing efficiency at the expense of increased tracking error. This has allowed many research groups to develop software GPS receivers that operate on sampling frequency of less than 5 MHz (e.g. Akos et al 2001, Heckler & Garrison 2004, Pany et al 2002).

However, the software receiver used herein operates on a 5 MHz (real data) sampling rate. There are two main reasons for this. Reaching real-time performance with a higher rate will give one the flexibility of moving down in sampling rate without worrying about the speed and the performance of the receiver. In addition, a higher sampling rate allows one to use advanced correlator techniques that cannot be implemented when using lower sampling rates, and are therefore desirable.

A National Instruments (NI) Data Acquisition Card (NIDAQ5335) transfers data from the frontend to the software. NI-DAQ offers a library supporting traditional single buffer data transfer while also offering the new technique of double buffering. Double buffering gives the ability for uninterrupted and continuous transfer of large blocks of data to the software.

Figure 20 shows the test and data gathering setup used to evaluate the receiver performance. A Spirent 7700 simulator generates the GPS Signal. The simulator allows to create controlled and repeatable scenarios to test the receiver. Both static and dynamic test scenarios were created. In the static case, an initial position was chosen to be at the

same coordinates as that of the University of Calgary. A valid ephemeris was provided to the simulator for the chosen day. In the dynamic scenario, the test scenario had a vehicle going in straight line (toward east) with a constant speed of 10 km/h. The trajectory of vehicle motion was then recorded through the simulator and later used as the merit for the true trajectory track.

The GPS RF signals were input to a NovAtel Euro-3M™ card, which served as frontend. The Euro-3M™ card has an intermediate frequency (IF) of 70.42 MHz and a front–end bandwidth of 16 MHz, as shown in Figure 20. The output from the card is 6-bit samples (3-bit L1 and 3-bit L2), synchronized with a 40 MHz sampling clock.

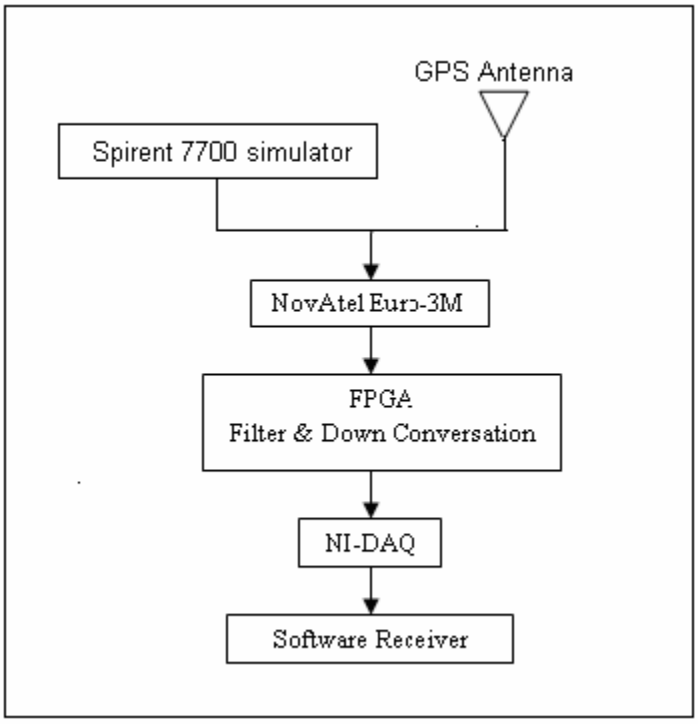

Figure 20: Frontend and test set up

An FPGA implements both the digital down converter and a 6-tap band-pass filter used to reduce the effects of aliasing. Data is the down converted by choosing every eighth sample yielding a 5 MHz output rate. A, typical, 6 dB degradation in SNR is introduced through the digital down conversation process. Unfortunately, the low-order band-pass filter also introduces additional 1-2 dB degradation in the SNR. Better performance is possible with higher-order filtering, however, such a filter could not be implemented herein because of the speed limitations of the FPGA.

In order to illustrate the better performance of higher order filters, the data was post processed and down converted using different filter orders. Next, a search performed for a known signal on the data sets. Lastly, a 1 ms coherent integration performed. As figures 21, 22, 23 and 24 illustrate, the higher order filters eliminate more noise leading to SNR ratio increases. This result shows the importance of choosing a good and suitable frontend for real-time GPS software receivers. This is discussed further in the next chapter as part of future work that needs to be done.

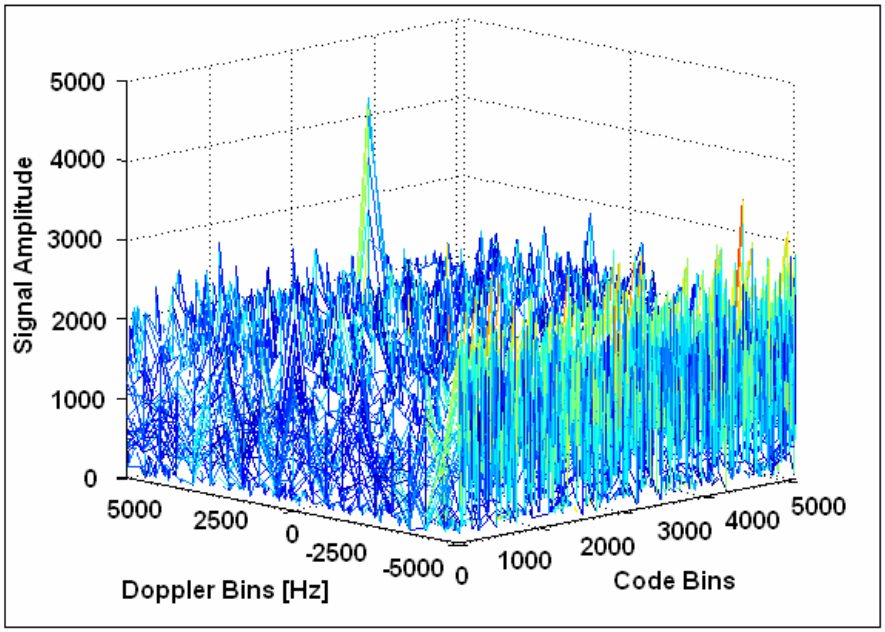

Figure 21 : 6-tap band-pass filter

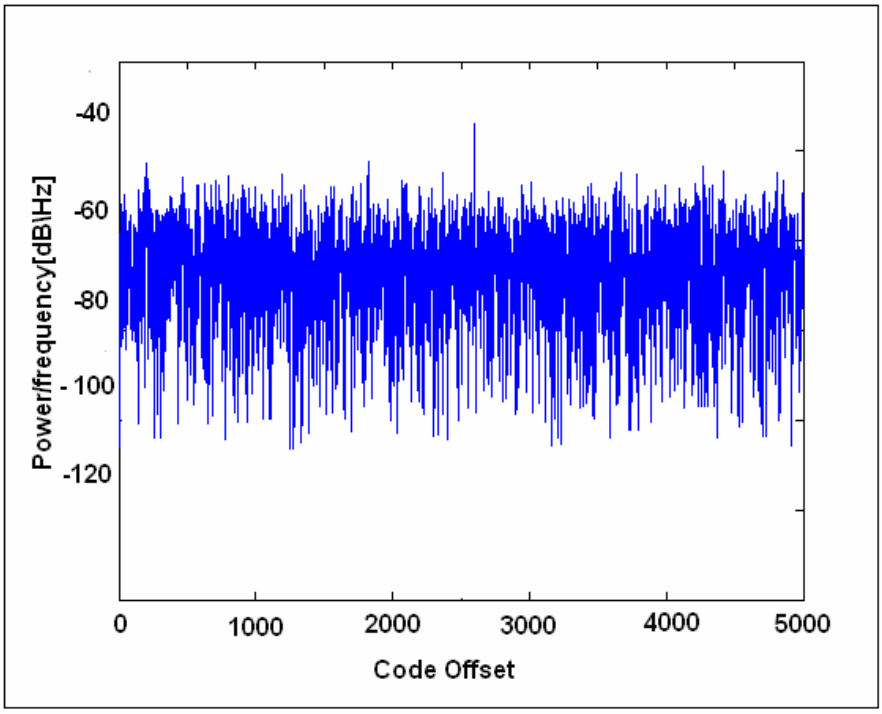

Figure 22: PSD of the incoming signal while using a 6-tap filter

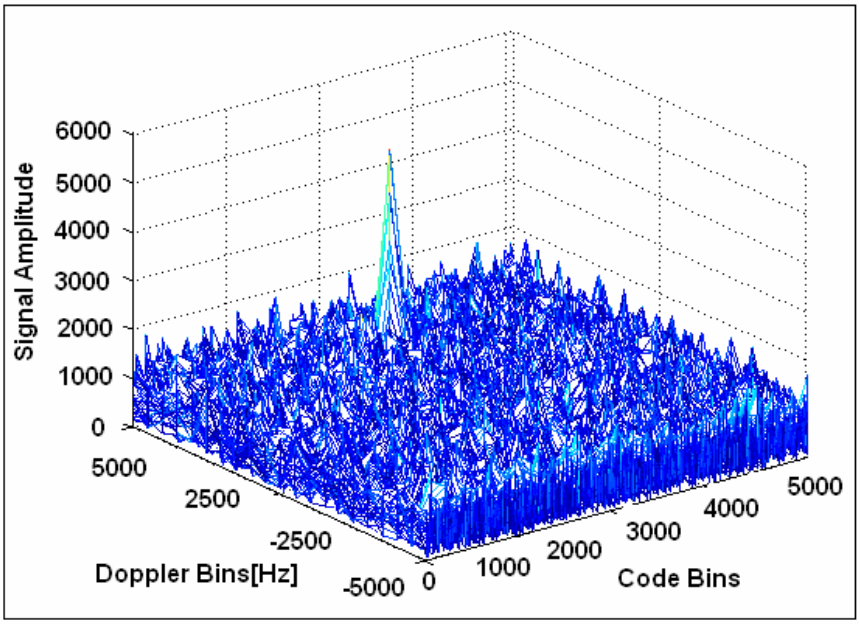

Figure 23: 20-tap band pass filter

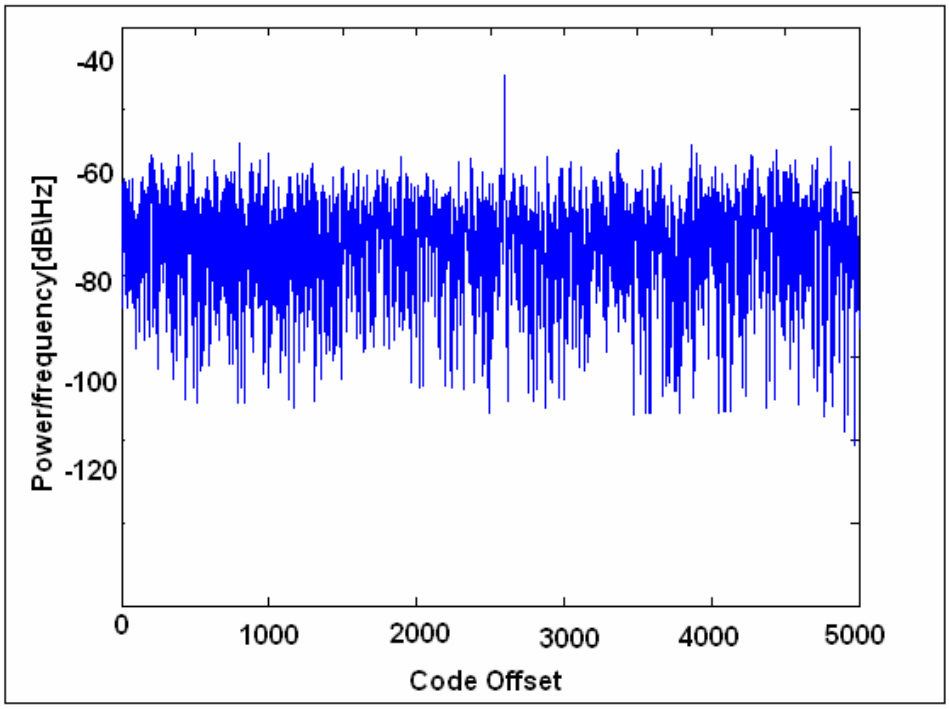

Figure 24: PSD of incoming signal using a 20-tap filter

Figure 25 illustrates the operations performed in the FPGA.

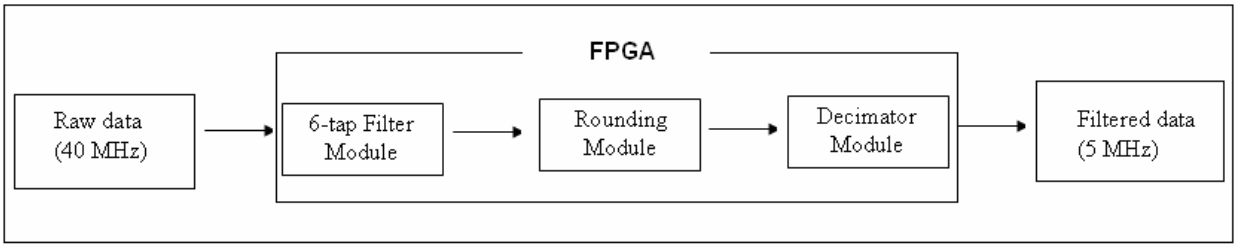

Figure 25: Digital down conversion in FPGA

The NI data acquisition card (5335) has DMA (Dynamic Memory Access) capability that increased the speed of transferring data to the PC. As mentioned earlier, a double buffer scheme is used to transfer blocks of data in a continuous manner. In double-buffered input operations, the data buffer is configured as a circular buffer. The DAQ device fills the circular buffer with data. When the end of the buffer is reached, the device returns to the beginning of the buffer and fills it with data again. This process continues indefinitely until it is interrupted by a hardware error or cleared by a function call (DAQ 2000). However, during double buffering, the NI-DAQ card internally divides the buffer into two equal parts. This allows the NI-DAQ to coordinate user access to the data buffer with the DAQ device. The coordination scheme is conceptually simple; NI-DAQ copies data from the circular buffer in sequential halves to a transfer buffer that the user creates. The user can then process this data while the DAQ device transfers the incoming data into the

other half of the circular buffer. The key to real-time implementation therefore, is to process all of the data in the transfer buffer before it is over-written.

For the following tests, the receiver was configured with the following parameters:

- 1 ms coherent integration
- Second order FLL with 10 Hz bandwidth and decision directed cross product discriminator
- Third order PLL with 18 Hz bandwidth and ATAN discriminator (aided by FLL).
- 20 Hz Pseudo-range measurement output rate.
- No Carrier Aiding
- Second order DLL with 2 Hz bandwidth, Non-Coherent Normalized Early-minus-Late envelope (0.5 chip spacing).

4.2 Real-time Performance

Before developing the interface between the hardware and software, a series of test were designed and conducted to evaluate the speed of the receiver and the algorithms employed in the receiver. GPS data was gathered through the set up illustrated in Figure 26. The goal is to show that the receiver can process 1 s of GPS data in less than 1 s.

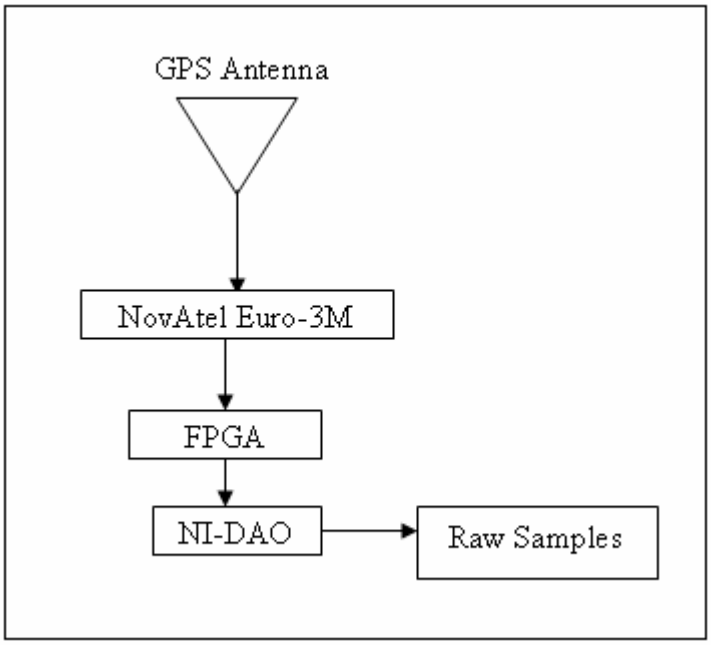

Figure 26: Data Collection set up for Performance Testing

Table 9 summarizes the results of the post processing test. For each version of code or platform, 30 independent runs were made. The table results are the average time to process 30 seconds of GPS data while tracking 6 satellites.

Table 9: Performance Comparison

Processor	Algorithm	Processing time (s)	Improvement with MMX
Pentium (1.7 GHz) 256 MB RAM	No MMX	70.1	-
	With MMX	19.0	72.9%
Pentium 4 (2.43GHz) 512 MB RAM	No MMX	63.0	-
	With MMX	18.5	70.6%

The above results show a significant improvement in speed with MMX technology. In the "No MMX" version, the processing algorithm is exactly the same except that the MMX instructions are not used, but rather normal integer mathematics. It can be observed that reaching the real time speed with simple C operators is impossible with current high-end general purpose processors.

The performance measures given in Table 9 include all core signal processing, navigation processing and receiver monitoring. The processing times do not however include additional processing that will be necessary to enable real-time delivery of data samples from the RF front end to the software receiver, as the tested version reads data from previously recorded files.

After developing the interface between the front end and the software, a second set of tests were performed to analyse the real time capability of the receiver. In this test, the maximum numbers of satellites the receiver could track were input as an argument to the receiver. NI-DAQ card and the interface software were tuned so that if there is an over

write of the input data, an exception would be shown. In this way, the receiver's performance can be monitored closely to ensure no input data gets discarded.

It was observed that the receiver is capable of tracking eight satellites in real-time on a Pentium 4 processor 3.2 GHz and 1 GB of RAM and hyper-threading enabled. In this configuration, the receiver used only 55-60 % of the available CPU resources. Hyper-threading had no effect on the MMX performance. The receiver is also single threaded which is not affected by the hyper-threading feature of the window. However, by simulating the second logical CPU, hyper-threading enables other applications to run more easily while the software receiver program is being executed. The results that follow were all gathered while the receiver was operating in real-time and tracking eight satellites.

4.3 Acquisition Performance

Table 10 shows the time it takes the receiver to perform a full sky search in a cold start mode under line-of-sight conditions. Results presented are for 1 and 2 ms coherent integration. The time is the total time needed to search for all possible satellites.

Table 10: Acquisition Speed

Coherent Integration	Time (s)
1ms	2.4
2ms	6.1

Figure 27 and Figure 28 show the results of an FFT search for all possible code and Doppler bins for PRN 17 using 1 and 2 ms of coherent integration respectively.

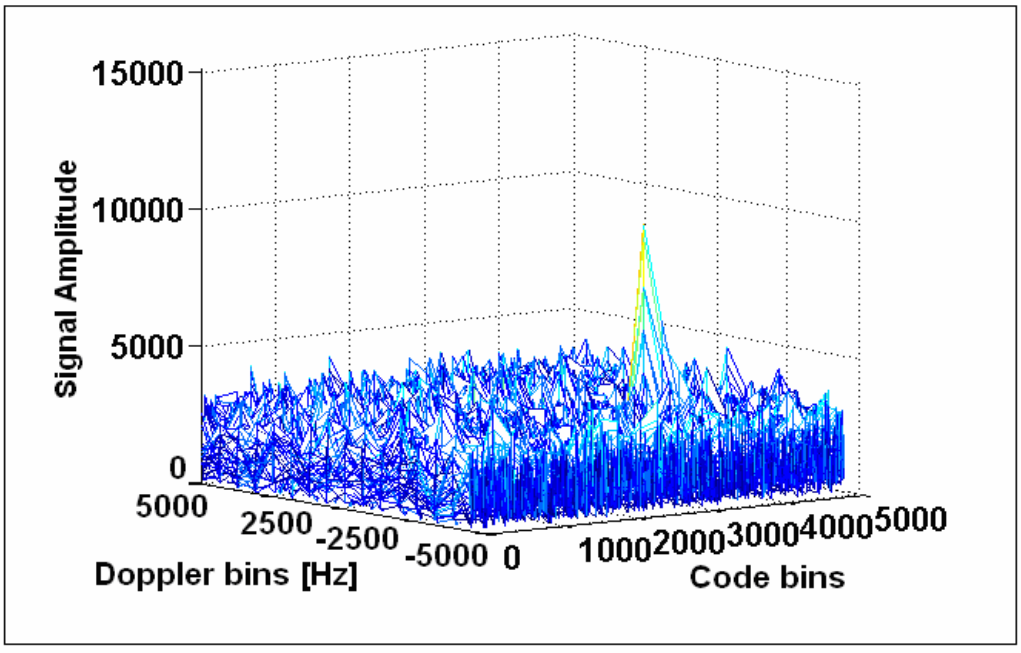

Figure 27: 1 ms coherent integration

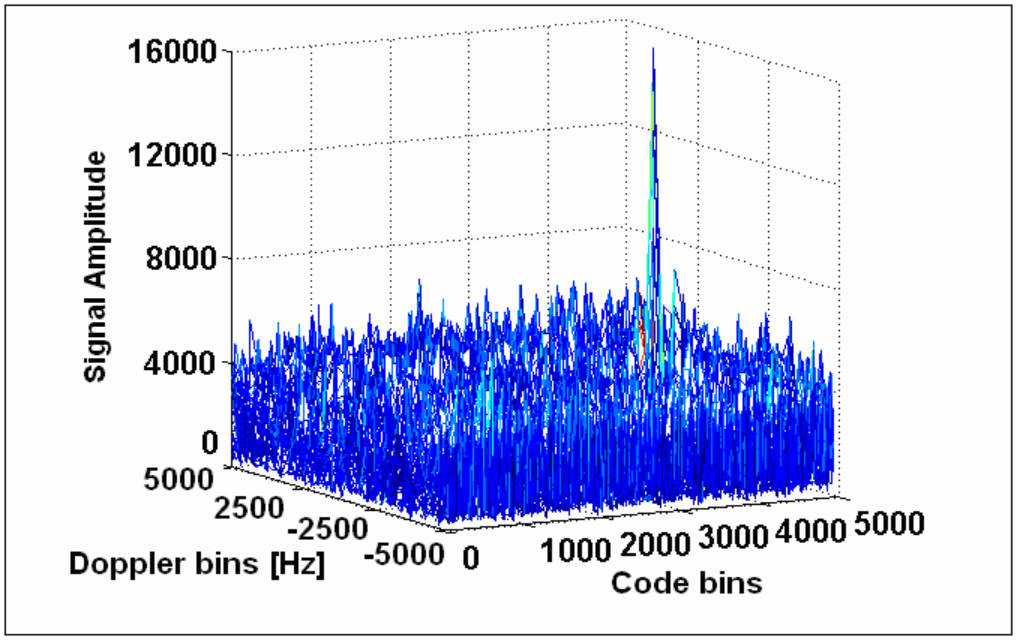

Figure 28: 2 ms Coherent Integration

As expected, increasing the time of coherent integration increases the size of the peak, therefore making the task of detection easier and more robust.

It has been discussed earlier that the NI-DAQ card uses the double buffer technique to transfer data from the FPGA to the software. This means a block of data is gathered first and then passed to the software for processing. Following the acquisition process, samples that were not processed during acquisition need to be discarded (in order to stay in real-time) and therefore the Doppler and code phase need to be predicted for the future. The second approach used in the receiver is that all data gets processed. This means the receiver is not working in real-time. The size of the buffer basically determines the size of delay that the receiver operates with. In this work, the size of the buffer was chosen so that the software can perform a 1-ms coherent integration (blind search). This essentially

means that the receiver is 2.4 seconds behind real-time when it processes the data. This can be a very critical issue in high dynamic environments. An ideal scenario is for the receiver to transfer 1 ms of data from the DAQ card and perform the acquisition on that 1-ms of data in 1 m or less. Then transfer the next ms of data from the DAQ card and continue searching. This way, the receiver is always working on the most recent data available from the frontend. However, this approach was not possible with the current architecture of the receiver. First, there should be an interrupt routine implemented in software to measure time and implement transfer control from the software after 1 ms to the DAQ module that transfers data from the frontend. Also, the software needs to keep a record of where in the process of acquisition it is when the interrupt is issued (which PRN, which Doppler and which code bin).

In addition, the DAQ card should not have too much overhead (in double buffer mode) when it switches back and forth between the two buffers at such a high rate. The above approach, proved to be too computationally expensive with the current architecture, platform and hardware. Therefore, the simple double buffer approach was chosen while noting the potential problems associated with it.

4.4 Tracking Performance

This section presents the tracking results. All of the results were obtained using PRN 23 and are indicative of other satellites. Figure 29 shows the C/N_0 calculated by the receiver for all the PRNs that are being tracked by the receiver in real-time. As expected, the signals are strong since they were generated by the simulator and do not contain effects of antenna gain pattern roll off. The sampling rate of 5 MHz (real data) is used for Figure 30. In this section, results from using a sampling rate of 5 MHz (complex data) are also presented. The goal of this section is to show that the receiver can operate with both real and complex data.

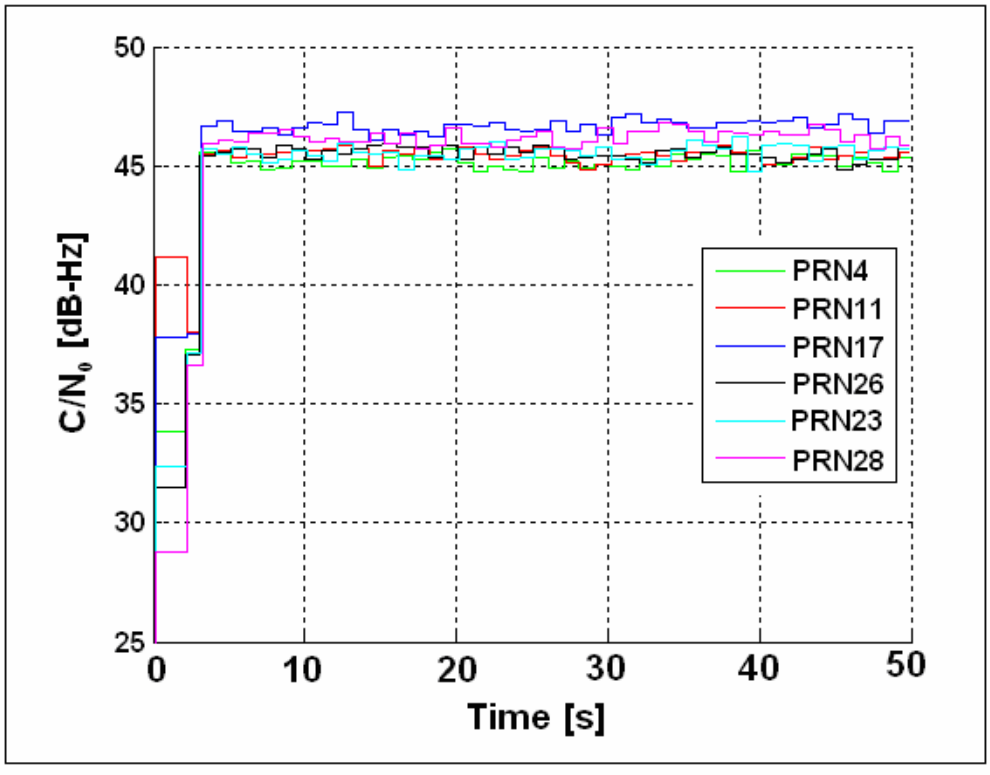

Figure 29: Estimated C/N_0 for all PRNs

Figure 30 and Figure 31 show the Doppler for PRN 23 and PRN 22 real and complex data (5 MHz sampling rate) respectively and indicate good carrier tracking by the receiver. In order to verify that the Doppler is correct and the receiver is tracking the correct PRN, a NovAtel OEM4 receiver was connected and ran at the same time as the software receiver. The Doppler was logged from the OEM4 and later compared to the software receiver's reported Doppler measurements. The small difference that is seen in the behaviour of the Doppler tracked by OEM4 and the software receiver is due to various reasons. The two receivers use different oscillators and the effect of the oscillator has a direct effect on the Doppler. Also, the parameters used in the tracking module of the OEM4 are different than that of the software receiver, leading to the small differences observed in the graphs.

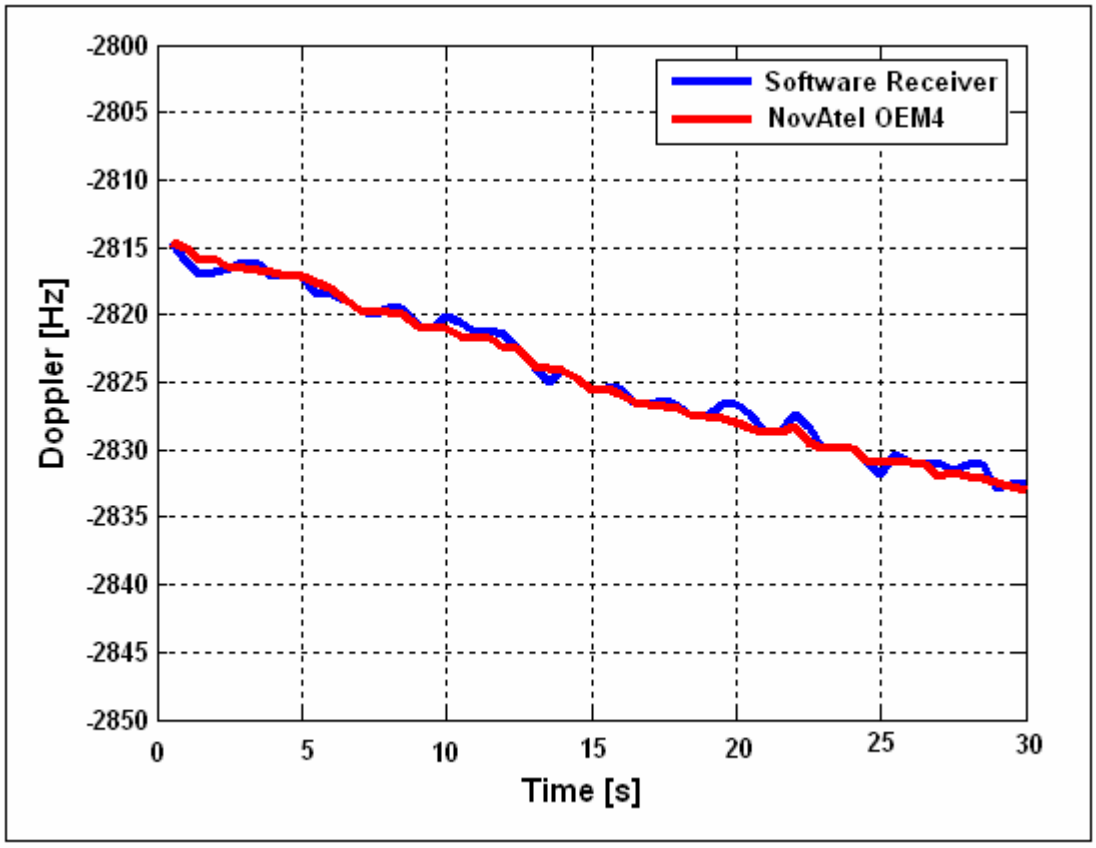

Figure 30: Doppler value for PRN 23

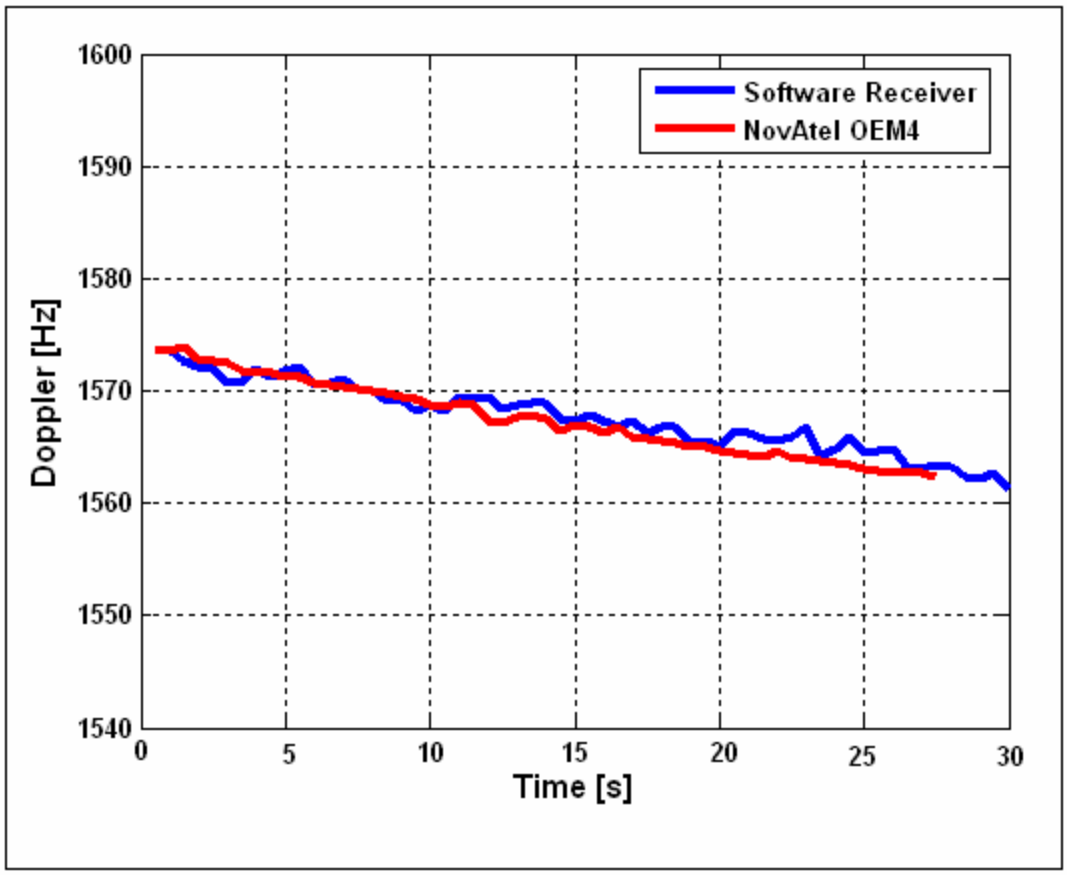

Figure 31: Doppler value for PRN 22

Figure 32 is a smoothed version of the output of the PLL lock detector (real data). As expected, the value converges to one which shows good phase lock during tracking. The same results are observed for the complex data.

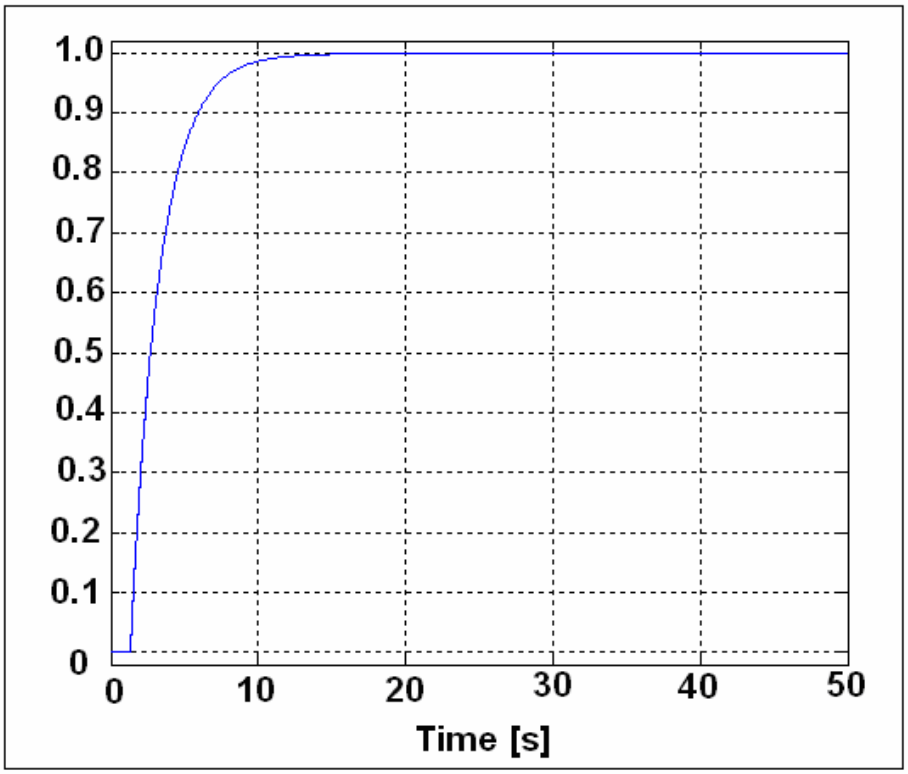

Figure 32: PLL Lock detector output

4.5 Position Accuracy

A single-point position and velocity solution was computed using the PLAN group's C^{3}NAVG$^{2\text{TM}}$ software. C^{3}NAVG$^{2\text{TM}}$ uses an epoch-by-epoch least-squares algorithm. The results shown in Figure 33 and Figure 34 were obtained using the software receiver pseudorange measurements only. Table 11 shows the RMS errors for position (north, east and up) and their corresponding DOP. The errors are within a reasonable range (metre level) for single-point operation. As it was discussed previously, a Spirent 7700 simulator was used for the real-time tests. Static and dynamic scenarios were set up to use the default settings of the simulator. This means that no troposphere and multipath errors

were simulated in these scenarios. The ionosphere error was also set to zero (only difference between the default scenario and that used for this work). Table 11 results also show a direct relationship between the DOP and the error magnitude.

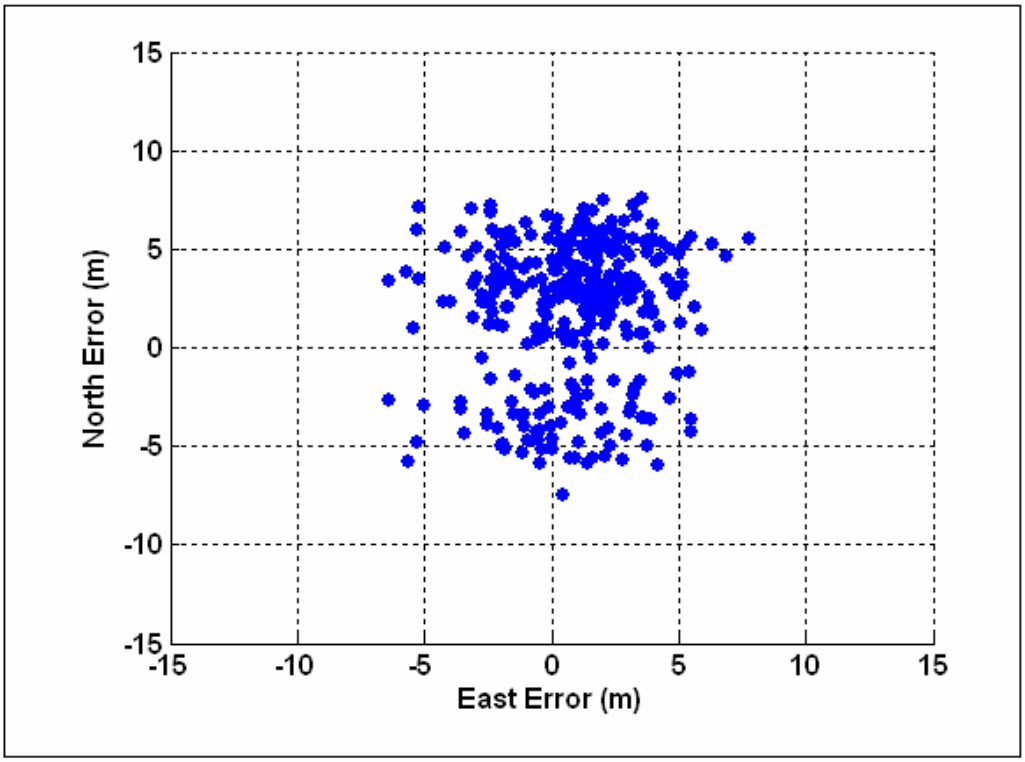

Figure 33: Scatter plot of North and East errors

Table 11: Position error statistic

Parameter	**North**	**East**	**Up**
DOP	0.9	0.9	2.1
Mean Error	3.7 m	2.2 m	10.5 m
RMS Error	4.1 m	3.1 m	12.2 m

Figure 34 shows the difference between the solution that was calculated by a NovAtel OEM4 receiver and the software receiver. As Figure 34 and Table 12 demonstrate, the NovAtel unit has better solution accuracy. This is likely due to more sophisticated signal processing algorithms.

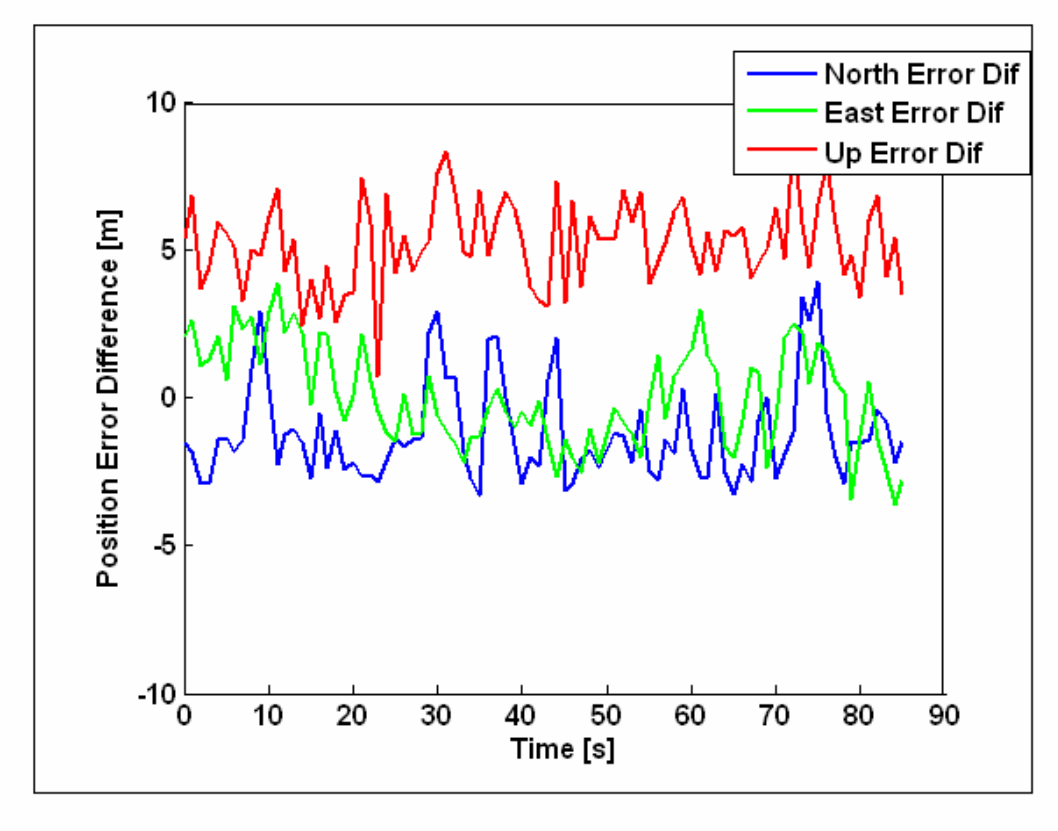

Figure 34: Solution error difference with OEM4

Table 12 shows some statistical values related to figure 34 for better understanding the performance of the receiver versus the OEM4.

Table 12: Position error difference with OEM4

Parameter	North	East	Up
Mean Error	-1.2 m	0.8 m	5.3 m
RMS Error	1.9 m	1.5 m	7.2 m

Figure 35, shows the errors in the pseudorange that are calculated by the receiver. For this test, a simulator was used and the pseudoranges were logged by both the simulator and the receiver. As the figure shows, the noise in the pseudoranges reaches three metres.

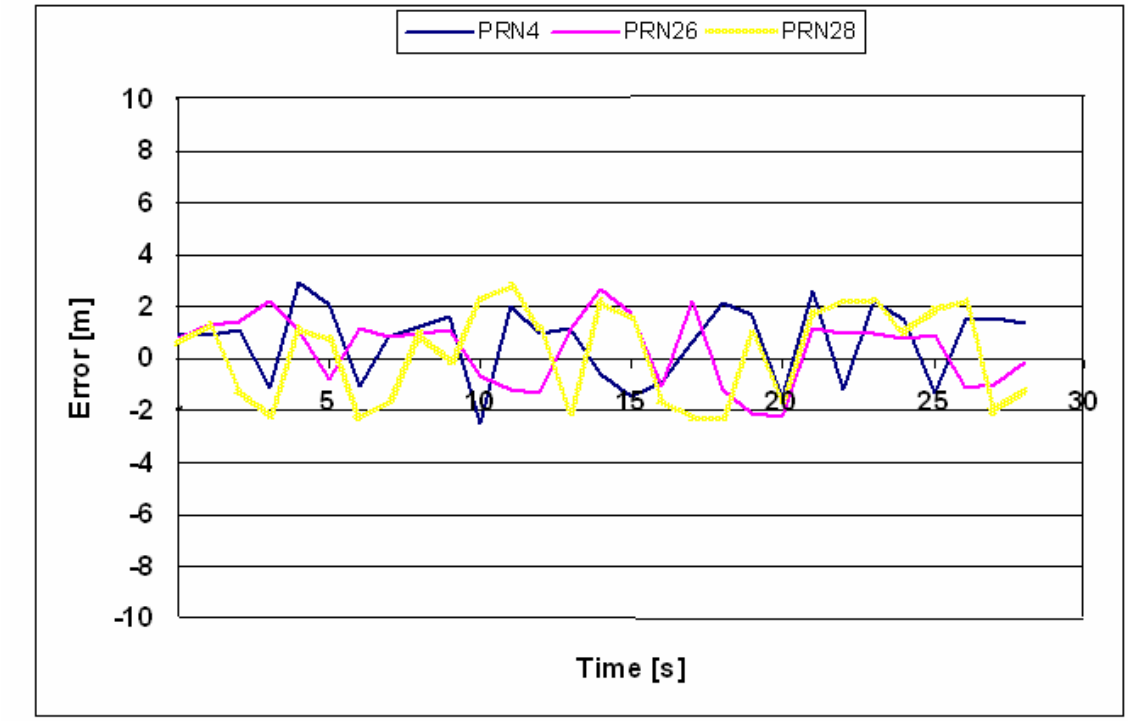

Figure 35: Pseudorange errors

As mentioned above, a static antenna was used in the experiment. Figure 36 shows the velocity errors, which have an RMS value of 0.09 m/s for north, 0.08 m/s for east and 0.15 m/s for the vertical component, respectively.

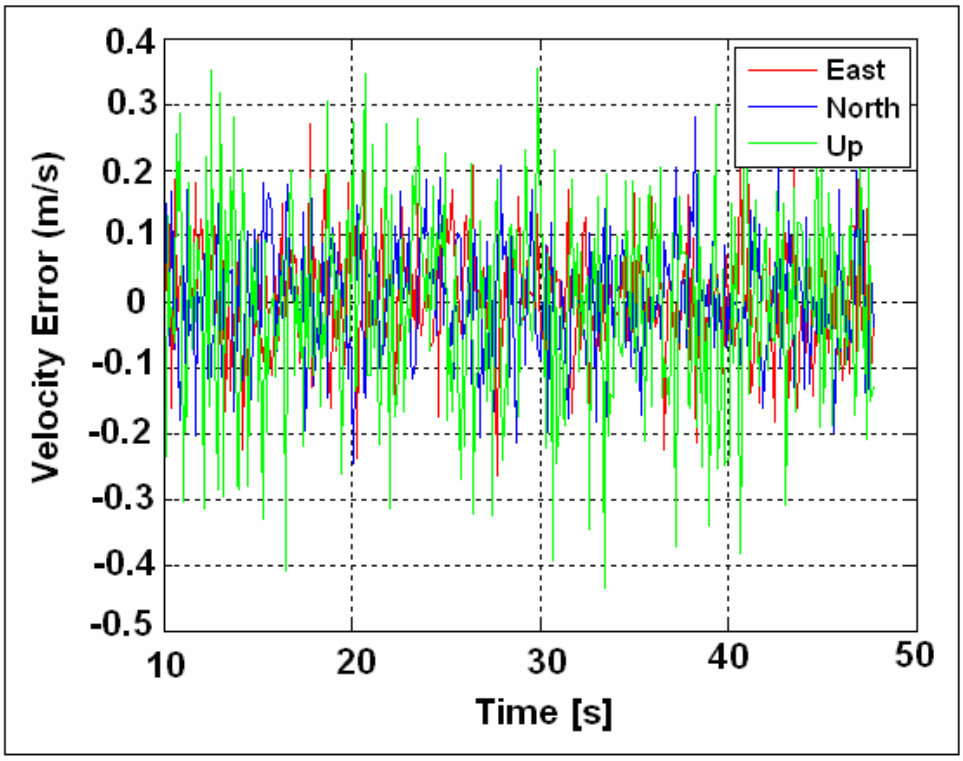

Figure 36: Software receiver-Derived velocity errors

The next set of graphs shows the position accuracy when the receiver is operating on complex data. It should be noted that this is real data that was gathered on the roof top antenna range at the University of Calgary. The data was later down converted to 5 MHz and post processed. Since the FPGA code is written to only work with real data, the post processing method was chosen for complex data analysis.

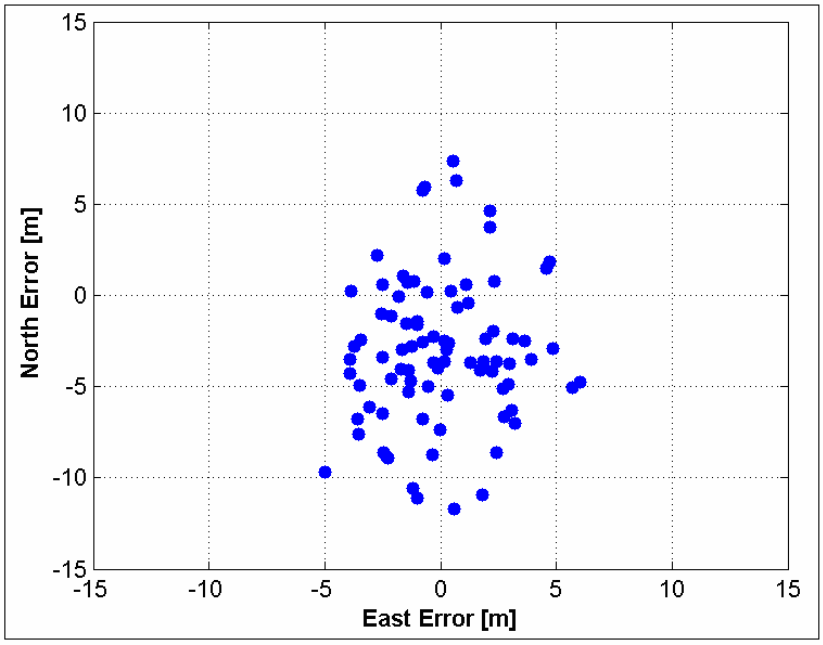

Figure 37: Scatter plot of North and East error

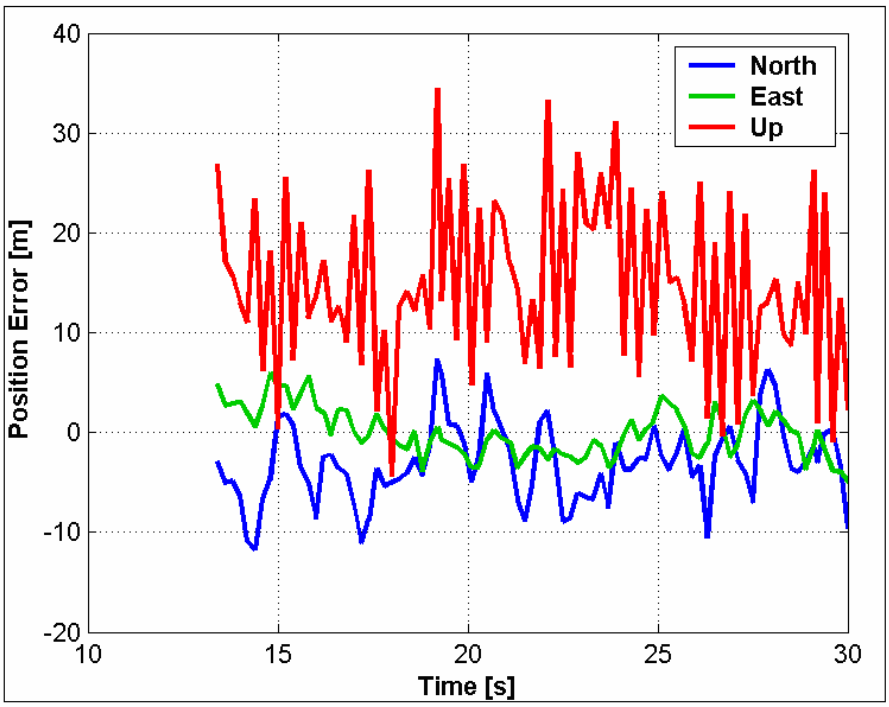

Figure 38: Position error versus time

Table 13 shows the RMS errors for position (north, east and up) and their corresponding DOP. The errors are within the reasonable range (metre level). Table 13 results also show a direct relation between the DOP and the size of errors.

Table 13: Position error statistics

Parameter	**North**	**East**	**Up**
DOP	1.3	0.7	2.2
RMS Errors	5.1 m	2.5 m	17.0 m

As mentioned above a static receiver was used in the experiment. Figure 39 show the velocity errors, which have an RMS value of 0.14 m/s for north, 0.11 m/s for east and 0.29 m/s up. Hardware receivers typically have a corresponding 0.05 m/s RMS. This is because that most hardware receiver use delta carrier phase measurements to determine the velocity component, however, in this analysis, delta positions have been used to determine the velocity. Since pseudo-range measurements are nosier than carrier phase measurements and also no carrier smoothing was employed, larger RMS values are observed for velocities.

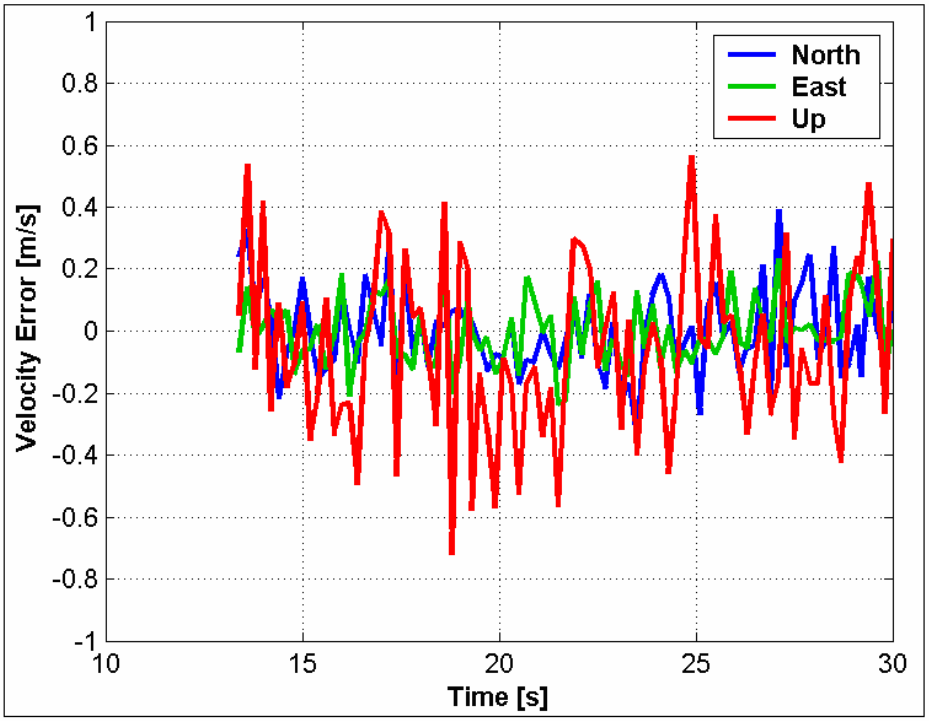

Figure 39: Velocity error versus time

Next, the result of a simple dynamic test is presented. A straight line, constant speed, 10 km/h, going from west to east was designed in the simulator. The latitude, longitude and height were logged by the simulator and used as the true trajectory for the motion of the vehicle. As the Figure 40 and Table 14 illustrate, the positions are within a reasonable range for such a receiver.

Table 14: Position error dynamic test

Parameter	**North**	**East**	**Up**
DOP	0.9	0.7	2.4
Mean Errors	3.9 m	2.0 m	11.0 m
RMS Errors	4.8 m	3.0 m	16.5 m

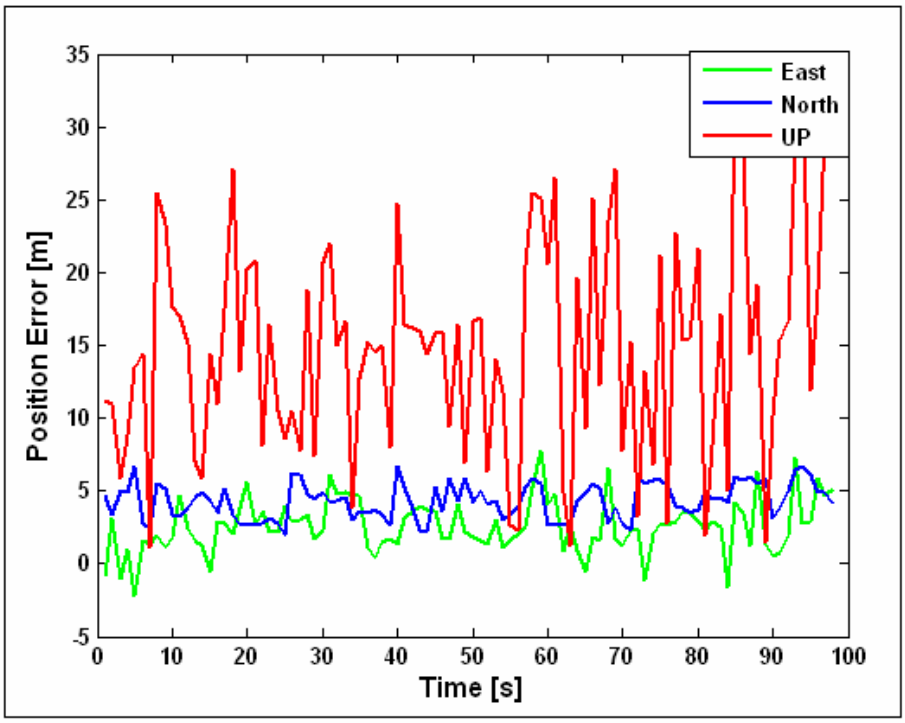

Figure 40: Position errors in dynamic mode

Chapter 5

Conclusions and Recommendations

This work focused on the development of a real-time software GPS receiver. It showed that, based on the demanding computational requirements of a software-based GPS receiver, high data processing efficiency is required in order to obtain real-time performance. There are two basic approaches to accomplish this: reducing the number of computations required, or improving the efficiency with which the computations are carried out. This work took the latter approach, primarily by using MMX technology available on x86-compatible processors to more rapidly perform the Doppler removal and code correlation computations. Other computational saving methods and algorithms were also described in the earlier chapters. The work demonstrates that computational improvements of greater than 70% are realized over the standard (integer math) implementation which allows the receiver to operate in real-time while tracking up to eight satellites. Test results indicate that tracking performance of the software receiver is reasonable and that position and velocity accuracies are at the metre and decimetre per second level, respectively. Also, there were some short comings observed in this work which need to be addressed and can be used as the basis for future development work.

5.1 Front-end and Acquisition component

It was discussed in chapter 4 that GPS signals can be sampled with a 2.5 to 5 MHz sampling rate. In this work, the receiver was designed to work with a 5-MHz sampling rate. A combination of a Euro-3M card, FPGA for down conversation and NI-DAQ 5335 was used for the frontend. However, it was noticed that the digital down conversion introduces a 6 dB loss which reduces the C/N_o ratio. Also, the down conversion adds an additional level of complexity to the frontend. The importance of a good frontend and low sampling rate was addressed. For future work, it is recommended to choose a more suitable frontend for this receiver thus eliminating the need for the down conversion process. One of the advantages of using a software receiver is that all of the baseband signal processing algorithms can be easily modified to work with new sampling rates and new frontend designs.

5.2 Performance

The most computationally expensive task of a GPS receiver is IF signal processing, which includes Doppler removal and correlation with the local code. A receiver typically needs to process GPS data acquired at a sampling rate of 2.5 MHz to 5 MHz for C/A code. Table 7 showed the numbers of operations required to track 6 six satellites based on the above sampling frequencies.

It was also shown\ through a simple example that real-time operation of a software receiver cannot be accomplished through simple integer arithmetic operations (Heckler & Garrison 2006). It was also shown that when using SIMD instructions such as MMX, a 70% improvement in processing time could be realized. SSE and SSE2 instructions can

also lead to even better performance. Codes that are written with this algorithm can be packaged and linked as a library and used in any other software receiver easily. Also, because of the flexibility that software receivers offer, it can be used both as a post-processing tool and a real-time receiver. It would be very useful for future work to add a graphical interface to the tool. The post processing software could also still benefit from SIMD instructions to increase processing speed.

A more optimized method for Doppler removal was proposed in this work and implemented in the receiver. This method is called the Table grid method in which both sine and cosine values are computed based on some grid frequencies and saved in a table (Ledvina et al 2003). With the Table grid method, a decrease of C/N_0 is expected due to using an inexact frequency. However, this has no effect on the accuracy of the solution, nor does it significantly reduce the tracking sensitivity. Test results for tracking were compared to NovAtel OEM4 and they were considered to be satisfactory. This was proof that the Table grid method can be safely employed in the software receiver to significantly reduce the computational burden.

For position accuracy performance, a signal was generated with the Spirent 7700 simulator. A position with RMS errors of about 5 m for North and East and about 15 me for Up was observed for both static and dynamic cases. As discussed earlier, these results are well within the range expected from this receiver.

References

Akos, D.M., P.L. Normak, A. Hansson, A. Rosenlind, Enge.P,(2001), Real-Time GPS Software Radio Receiver, Proc, of institute of Navigation 2001 National Technical Meeting (January 22-24, 2001,Long beach, CA) 809-816.

Borre. K, D. Akos, N.Bertelsen, P. Rinder, A. Jensen (2006), A software-Defined GPS and Galileo Receiver , A Single-Frequency Approach, Birkhauser.

Heckler,G.W & James L.Garrison (2006), SIMD correlator library for GNSS software receivers, GPS solout .

Heckler,G.W & James L.Garrison (2004), Architecture of a Reconfigurable Software Receiver, ION GNSS 17th International technical Meeting of Satellite Division(September 21-24, 2004, Long Beach, CA) 947-955

Krumvieda, Ken, P. Madhani, C. Cloman, E. Olson,; J. Thomas, P. Axelrad, W. Kober, A Complete IF Software GPS Receiver: A Tutorial about the details

ICD-GPS-200 (2003), Interface Control Document, Navstar GPS Space Segment and Navigation User Interface, ARINC Research Corporation, El Segundo, CA, January 14

Intel Copration (2006) Basic architecture, IA-32 Intel architecture software developer's manual, vol 3B

Julien, O. (2005) Design of Galileo L1F Receiver Tracking Loops. PhD Thesis, published as Report No. 20227, Department of Geomatics Engineering, The University of Calgary.

Ledvina, B.M., M.L. Psiaki, S.P. Powel, and P.M. Kintner (2003), A12-Channel Real-Time GPS L1 Software Receiver, Proc. of institute of Navigation National Technical Meeting (January 22-24, 2003 Anaheim, CA) 767- 783

Ledvina, B.M., M.L. Psiaki, S.P. Powel, and P.M. Kintner (2006), A Real-time software receiver for the GPS and Galileo L1 Signal, Proc. of institute of Navigation National Technical Meeting (January 18-20, 2006 Monterey, CA)

Ma, C, G. Lachapelle & M.E. Cannon (2004), Implementation of a Software Receiver, ION GNSS 17th International technical Meeting of Satellite Division (September 21-24, 2004, Long Beach, CA) 882-893.

Pany,T. , B.Eissfeller, G.Hein, S.W. Moon and D.Sanroma (2004) IPEXSR: APC Based Software GNSS Receiver Completely Developed in Europe. ION GNSS 17th

International technical Meeting of Satellite Division (September 21-24, 2004, Long Beach, CA).

Skone, S., G. Lachapelle, D. Yao, W. Yu and R. Watson (2005) Investigating the Impact of Ionospheric Scintillation using a GPS Software Receiver. Proceedings of GNSS 2005 (Session C3, Long Beach, CA, 13-16 September).

Soloviev, A., S.Gunawardena, F. Van Grass, Development of High Performance High Update Rate Reference GPS Receiver. ION GNSS 18th International technical meeting of the satellite division, 13-16 september 2005, Long Beach,CA.

Tsui, James B-Y. (2000), Fundamentals of Global Positioning System Receivers: A Software Approach, John Wiley & Sons Inc.

Van Dierendonck, A.J. (1996), Global Positioning System: Theory and Applications, Volume I, Chapter 8: GPS Receivers, AJ Systems, Los Altos, CA 94024. Inc.

Ward, P (1996), Understanding GPS, Principles and Applications, Boston: Artech House, Inc.

Zheng, B. and G. Lachapelle (2005) GPS Software Enhancements for Indoor Use. Proceedings of GNSS 2005 (Session C3, Long Beach, CA, 13-16 September).

.