IMPLEMENTATION AND OPTIMIZATION OF A
GLOBAL NAVIGATION SATELLITE SYSTEM SOFTWARE RADIO

A Thesis Presented to

The Faculty of the

Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirement for the Degree
Master of Science

by

Sunil Bhanot

November, 1998
ACKNOWLEDGEMENTS

The author expresses his gratitude to Dr. Michael S. Braasch, Assistant Professor of Electrical Engineering and Computer Science, for all his encouragement and support throughout this project. The author is indebted to him for his frequent inspirations and suggestions in molding the author into a researcher. Appreciation is also extended to Dennis Akos, Chris Snyder, research engineers at the AEC for their timely help, support and encouragement in all aspects of the research. Appreciation is also extended to Dr. Frank van Graas, Associate Professor of Electrical Engineering and Computer Science for his support.

A significant amount of appreciation needs to be bestowed upon the people who funded this work. They include The Air Force Office Of Scientific Research and the Wright Patterson Air Force Base. The Joint University Program for Air Transportation Research sponsored by The Federal Aviation Administration and the National Aeronautics and Space Administration also provided a tremendous environment for research. Appreciation also needs to be extended to the people at the Honeywell Technology Center for their financial support.

All the people at the Avionics Engineering Center are thanked for their friendship and motivation they provided the author. A special thanks goes out to all the friends who extended their support and cooperation and to the one upstairs without whose help this whole project would not have been possible.
TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

TABLE OF CONTENTS iv

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF ACRONYMS ix

1. INTRODUCTION 1

2. GLOBAL NAVIGATION SATELLITE SYSTEMS 4
   2.1 GLOBAL POSITIONING SYSTEMS 4
   2.2 GLOBAL ORBITAL NAVIGATION SATELLITE SYSTEM 7

3. SOFTWARE RADIO 10
   3.1 FRONT END CONFIGURATION 10
   3.2 SOFTWARE SIGNAL PROCESSING 17

4. THEORETICAL BACKGROUND 19
   4.1 SPREAD SPECTRUM SIGNAL ACQUISITION 20
      4.1.1 Serial Search Acquisition 22
   4.2 SIGNAL TRACKING 24
      4.2.1 Costas Carrier Tracking Loop 25
      4.1.2 Early/Late Non Coherent Code Tracking 27

5. REAL TIME GPS SYSTEM DESIGN 30
   5.1 PROPOSED IMPLEMENTATION 30
LIST OF TABLES

TABLE 6.1 Satellites Identified In The Collected Data Through The Acquisition Process
LIST OF FIGURES

FIGURE 3.1 Typical Front End Design With Multiple Stages Of downconversion

FIGURE 3.2 Software Radio Front End Direct Digitization Front End Design

FIGURE 3.3 The Gps-Sps Software Radio Front End

FIGURE 4.1 Serial Search Acquisition Algorithm Block Diagram

FIGURE 4.2 Generalised Feedback Tracking Loop

FIGURE 4.3 Costas Tracking Loop Block Diagram

FIGURE 4.4 Early-Late Non Coherent Delay Lock Loop Block Diagram

FIGURE 5.1 Proposed Implementation Of The GPS-SPS Software Radio

Figure 5.2 Call Chart For The Proposed Software Radio Implementation

Figure 6.1 Block Diagram Of The C/A Code Geneartor

Figure 6.2 Shifting Of The G2 Code Based On The Satellite Vehicle Number

Figure 6.3 Plot Of The Unprocessed Gps Data

Figure 6.4 Plot Of The Sine Wave For The Sin(),Cos() Lookup Table Approach

Figure 6.5 Plot Of The Acquisition Process For The Satellite Vehicle Number 17

Figure 6.6 Plot Of The Acquisition Process For The Satellite Vehicle Number 6

Figure 6.7 Early/Late Samples With Respect To The Prompt Code

Figure 6.8 The Early/Late Generation

Figure 6.9 Plot Of Carrier/Code Tracking For Satellite Vehicle Number 17

Figure 6.10 Plot Of The Navigation Data For Satellite Vehicle Number 17
Figure 6.11 Plot Of The Synchronized Data For Satellite Vehicle Number 17 75
Figure 6.12 Comparison Between The Synchronized And Unsynchronized Data For Satellite 17 76
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>analog-to-digital converter</td>
</tr>
<tr>
<td>ASIC</td>
<td>application-specific integrated circuit</td>
</tr>
<tr>
<td>bps</td>
<td>bits per second</td>
</tr>
<tr>
<td>BW_i</td>
<td>information bandwidth</td>
</tr>
<tr>
<td>C/A</td>
<td>coarse acquisition</td>
</tr>
<tr>
<td>CDMA</td>
<td>code division multiple access</td>
</tr>
<tr>
<td>f_c</td>
<td>carrier frequency</td>
</tr>
<tr>
<td>f_s</td>
<td>sampling frequency</td>
</tr>
<tr>
<td>FDMA</td>
<td>frequency division multiple access</td>
</tr>
<tr>
<td>GLONASS</td>
<td>Global Navigation Satellite System</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GPS-PPS</td>
<td>Global Positioning System – Precise Positioning Service</td>
</tr>
<tr>
<td>GPS-SPS</td>
<td>Global Positioning System – Standard Positioning Service</td>
</tr>
<tr>
<td>m</td>
<td>meters</td>
</tr>
<tr>
<td>Mbps</td>
<td>mega bits per second</td>
</tr>
<tr>
<td>rad</td>
<td>radians</td>
</tr>
<tr>
<td>RF</td>
<td>radio frequency</td>
</tr>
<tr>
<td>SV</td>
<td>satellite vehicle</td>
</tr>
</tbody>
</table>
**Introduction:**

The NAVSTAR Global Positioning System (GPS) commissioned by the Department of Defense is one of the primary radionavigation systems currently in use. GPS and the Russian Global Orbiting Navigation Satellite System (GLONASS) are elements of the Global Navigation Satellite Systems (GNSS). Currently these two systems are a part of GNSS but it can also include other navigation systems in the future. The Global Positioning System plays a very important role in the determination of absolute and relative position, velocity, and time for both civilian and military users. GPS is a complex radionavigation signal employing code division multiple access (CDMA) spread spectrum modulation to provide a horizontal position estimate to within 100 meters for civilian users. Considerable research[9] is being directed towards ensuring that the GPS receivers are robust enough to provide the extremely high reliability necessary for use in aviation applications.

The software radio is a conceptual receiver design process in which the primary aim is to place the analog to digital converter (ADC) as close as is possible to the antenna and process the resulting samples using a programmable microprocessor. The implementation has many advantages over the traditional method in that it offers the ability to process a wide range of radio frequency (RF) transmissions using a single front end. In this design philosophy the ADC operates at a very high rate, thereby capturing a wide range of the frequency spectrum. The programmable microprocessor then digitally filters a specific transmission, decimates it, and further processes the resulting samples to extract the desired modulated information.
A GNSS software radio provides a good simulation/testing environment. In order to evaluate a specific algorithm, there is no need to replace hardware components or develop additional simulations – simply make the modifications in the software, compile the code, and download to the processor for real time execution and evaluation of performance. The software radio can be used as the ideal test bed for evaluation of various signal-processing algorithms. The front-end hardware will be minimized and the design will be able to process multiple GNSS transmissions. With a GNSS software radio it would be possible to make intelligent and educated decisions regarding the performance. The ideal of the GNSS software radio is to provide this kind of flexibility and ease of performance to the designers of various signal-processing algorithms. To achieve the desired flexibility and performance the two components of the software radio, the front-end implementation and the software programming, must be fully investigated.

The ideal front-end design involves the capture and processing of multiple GNSS transmissions. This provides additional measurements, which can be exploited to ensure the integrity of the position solution. The first objective of the software radio is to place the ADC as close as is possible to the antenna. The ADC cannot be placed next to the antenna as every RF transmission will require some degree of amplification and filtering. At present, processing signals at different frequencies traditionally involves multiple front-end designs, one for each signal of interest. There may be errors due to the analog nature of the components involved with this type of configuration. Another concern in the front-end design is the sampling frequency. It is important to minimize not only the front-
end hardware but also the required sampling frequency, as this directly determines the required processing power.

The second part of the GNSS software radio is concerned with the software based signal processing using a programmable microprocessor. The goal here is to achieve real time performance and hence the signal processing will be based on proven algorithms, which aid in real time implementation. This will establish a framework for testing advanced algorithms. The signal processing will concern itself with issues such as acquisition and code/carrier tracking and these issues will be investigated thoroughly. The processing routines will also provide a position estimate, which normally is the output of a GNSS receiver. This serves a dual purpose in that it helps in not only providing the position estimate but it also substantiates the accuracy of the signal processing algorithms. An important aspect of the software radio design is the ability to run in real time mode. With a view to achieving this goal, various platforms as well as various implementation strategies will be investigated.

Finally, the work on the software receiver is aimed at minimizing the run time while at the same time providing the desired accuracy. This work is generic in the sense that it can be applied to any RF transmission. Also, the inclusion of new techniques and optimization techniques can be facilitated due to its highly modular design. GNSS provides a platform in which the accuracy of the signal processing is critical and the performance demands are very stringent; thus a software radio implementation will offer significant advantages over a traditional front end design.
2. Global Navigation Satellite Systems

Satellite-based navigation systems are the latest development in an effort to compute the exact position. All satellite navigation systems consist of one or several satellites, a network of tracking or receiving stations which record the data, a computing center which takes in the data from the tracking stations and calculates the orbital parameters; a data link between the computing center and the navigator, usually via the satellite; and finally the navigators receiving and computing equipment. Satellite-based transmissions operate on the time-of-transmission or the trilateration[1] concept. The time delay of a RF transmission, scaled by the speed of light provides the distance between the receiver and the transmitter. Using multiple measurements to different transmitters, it is possible to obtain the position by means of the intersection of spheres, which provides the domain within which the signal might have been transmitted.

Currently the two primary GNSSs in operation are the NAVSTAR Global Positioning System and the Russian GLONASS[2]. The intent of this chapter is to provide system parameters, as well as the theoretical background, necessary for a software radio implementation.

2.1 Global Positioning System

GPS is an all-weather navigation tool for marine, land and airborne users throughout the world. GPS offers two levels of service: the military-specific Precise Positioning System(GPS-PPS) and the Standard Positioning System(GPS-SPS) for the civilian users. The target software radio implementation is a GPS-SPS receiver utilizing
the civilian component of GPS. GPS satellites relay direct-sequence spread spectrum signals including a navigation message with the satellite’s current position. The NAVSTAR GPS is made up of (1) The Space segment[3], which consists of 24 satellites operating in six orbital planes and provides, with a high probability, at least four visible satellites, which are required to compute the user’s position. (2) The GPS control segment consisting of ground stations whose function is to monitor and maintain the integrity of the satellites and (3) The GPS user segment consists of the base of receivers, which extract position and timing information from the broadcast signal.

The GPS ranging signals are transmitted at two frequency bands designated as L1 and L2 centered at 1575.42 MHz and 1227.6 MHz. The GPS-SPS component has the following signal structure.

\[ S_i(t) = \sqrt{2P} CA_i D_i(t) \cos(w_i t + \phi) \]

Where

- \( S_i(t) \) GPS-SPS broadcast from the \( i^{th} \) satellite
- \( i \) indicates the satellite number
- \( P \) signal Power
- \( CA_i \) C/A or PRN, code for the \( i^{th} \) satellite
- \( D_i \) Navigation data for the \( i^{th} \) satellite
- \( \phi \) Phase offset

GPS-SPS as stated earlier uses a CDMA spread spectrum modulation technique and the spreading code or the PRN code for the GPS-SPS signal is known as the Coarse/Acquisition while for the GPS-PPS signal it is known as the P-code.

The C/A codes are a subset of the Gold Code family, a collection of 1023 PRN codes which provide good multiple access capabilities and low cross correlation. The C/A code has a chipping rate of 1.023 Mbps and a period of 1023 chips. Each satellite has a unique C/A code; produced from the modulo-2 sum of two 1023-chip PRN codes.
G1 and G2. Two maximal-length linear shift register of 10 stages, generates the G1 and G2 codes. The tap positions are specified by the generator polynomial for the two codes as follows:

\[
\begin{align*}
G1: & \quad 1 + X^3 + X^{10} \\
G2: & \quad 1 + X^2 + X^6 + X^8 + X^9 + X^{10}
\end{align*}
\]

The G2 code is delayed by a certain amount of time before performing the modulo-2 summation to give the C/A code. The code period of 1023 bits is intentionally selected so as to permit rapid acquisition.

The navigation data \(D_i(t)\) provides the user with additional data required to solve for position, velocity and time. The GPS control segment uploads the navigation data to each satellite once per day or more often. The navigation data generated at 50bps provides information about the satellite position at time of transmission, satellite time, time transfer information, ionospheric corrections and the like. The navigation data is synchronized with the C/A code and are formatted into frames of 1500 bits and 30s in duration, and are subdivided into 5 subframes each of 300 bits and 6s in duration. Subframe 1 contains clock and health data specific to the broadcasting satellite. Subframes 2 and 3 hold the ephemeris parameters. Subframes 4 and 5 contain additional support information such as the almanac data for atmospheric corrections. The Navigation data is repeated every 1500 bits or every 30 seconds, so a window of GPS-SPS data of 30s will provide all the parameters necessary for a position solution.

The GPS-SPS signal received by the user has a received power spectral density below that of the thermal noise of the receiver. The minimum received power into a 3dB gain linear polarized antenna is \(-160\text{dBW}\) for satellites with an elevation angle greater
than five degrees. The received signal is very weak as compared to the noise and hence demodulation and processing requires using spread spectrum techniques.

2.2 Global Orbital Navigation Satellite System

GLONASS is implemented and maintained by the Russian ministry of defense. GLONASS[2] uses the FDMA approach assigning each satellite a unique frequency. The GLONASS space segment consists of 24 satellites operating in three orbital planes and are designed so as to provide at least four visible satellites on any location on the earth with a high probability. The GLONASS control segment and the user segment are similar to the GPS system with the control segment being in charge of monitoring the satellites as well as orbital corrections while the user segment is just a set of receivers for computing the position estimate.

The GLONASS signal is given by the following

\[ S_i(t) = \sqrt{2P} R(t) M(t) D_i(t) \cos(2\pi(f_c + i \cdot 0.5625 \cdot 10^6)t + \phi) \]

Where

- \( S_i \) : GLONASS broadcast for the \( i \)th frequency channel
- \( i \) : The frequency channel
- \( P \) : Signal Power
- \( R \) : The PRN Code
- \( M \) : Meander Sequence
- \( D_i \) : Navigation data for the \( i \)th channel
- \( f_c \) : 1602MHz(base frequency)
- \( \phi \) : Phase offset

GLONASS uses the FDMA[4] approach by assigning each satellite a unique frequency. Initially GLONASS started off with 24 channels; currently the implementation process is undergoing certain changes. Currently channels 1 to 15 and 21 to 24 are in use. Later modifications to the design results in the GLONASS satellites
being assigned frequency channels –7 through +6, which indicates that the GLONASS band will be shifted down from the current frequency band of 1602MHz. The GLONASS is similar to the GPS in the sense that it also incorporates a PRN code for ranging purposes. The GLONASS PRN has a chipping rate of 511Kbps and a period of 511 chips which is the same for all satellites because of FDMA multiplexing. The PRN code is a maximal length sequence generated by using a nine-position shift register. The generator polynomial for the same is given by

\[ G: 1 + X^5 + X^9 \]

The Navigation data \( D_i \) for the GLONASS satellites are generated at 50bps and combined with a meander sequence generated at 100bps to form the navigation message. The Navigation data, which provides additional parameters necessary for the computation of velocity, time and position, PRN code and meander sequence generation are synchronized within the transmitted signal. The GLONASS data consists of a 7500 bit superframe, which in turn is divided into five 1500 bit frames wherein each frame is further subdivided into 15 strings of 100 bits each. Each Navigation string, which is of 2 second duration is divided into two parts, the first 1.7 seconds is the navigation message, while the last 0.3 seconds consists of a 30 bit time mark.

The exact format of the GLONASS data and the algorithms needed to solve them are well documented[2]. The navigation data required for a position solution is broadcast every 30 seconds. The GPS and GLONASS systems differ in the modulation techniques used; GPS uses the CDMA[4] modulation scheme while GLONASS uses FDMA[4] modulation. Both the systems make use of the PRN code for ranging purposes. A 30-
second window of data for both GPS and GLONASS provides sufficient parameters required for the computation of position velocity and time. The GNSS software radio implementation has been modeled with a view to incorporate the GLONASS signal as well without having to change many system parameters.
3. Software Radio

The objective in a software radio[5] design methodology is to sample the signal and process the resulting samples using a programmable processor. The design issues associated with the software radio are reducing the number of non-linear analog components as well as minimizing the sampling frequency. The sampling frequency required directly determines the total required processing power. The advantages offered by the software radio design philosophy are plenty as it provides a good test bed for the simulation of various techniques. It will no longer be necessary to build costly prototypes for testing new or advanced signal processing algorithms. Testing can be done by simply coding the required signal processing and downloading to the target processor. The software radio implementation also helps in the removal of age-based non-linear components.

3.1 Front-end Configuration:

The front-end is the heart of the software radio architecture[5]. An attempt is made to present a comparison between the traditional front-end design and the software radio design. A figure of the conventional front-end design for the software radio is shown in figure 3.2. The front-end design has the ADC placed as close as is possible to the antenna as compared to the traditional front-end design shown in figure 3.1 with multiple stages of down-conversion and amplification. The problem with multiple stages of down-conversion (as opposed to single stage) is that it introduces additional analog
Conventional Front End Design

First Stage

Nth Stage

Figure 3.1 Typical Front End With Multiple Stages Of Down Conversion

Software Radio Front End Design

Figure 3.2 Software Radio Front End Design
components whose performance is a function of age and temperature. However it is also impossible to place the ADC right next to the antenna as any RF transmission requires some degree of amplification and filtering. The goal of the software radio architecture is to reduce the number of analog components and still manage to sample at RF.

Another significant advantage of the Software radio design as compared to the traditional front-end design is the ability to process the wide range of RF transmissions using a single front-end, which would not, be possible using the traditional front-end design. In this method the ADC operates at a very high rate capturing a wide range of the frequency spectrum. The programmable processor digitally filters and decimates the resulting samples to extract the desired modulating information. This provides for a tighter interaction between the simulation and the actual testing environment and thus eliminates the need for costly prototypes required in the case of a traditional front-end.

The front-end design depicted in figure 3.2 requires the sampling of the signal directly at RF. There are two approaches to consider. One is based on the information bandwidth and the other is based on the center frequency. A blind application of the Nyquist[6] theorem in the information bandwidth requires sampling at a rate twice the highest frequency. This is quite impractical since a sampling rate of well over $3\text{GHz}$ would be required.

Sampling centered around the center frequency is based on the principle of bandpass sampling[7]. As per the Nyquist sampling criteria the absolute minimum sampling frequency must be greater than twice the information bandwidth. This
technique known as bandpass sampling is the technique of undersampling a modulated signal to achieve frequency translation via intentional aliasing. The various stages of local oscillators and mixers are no longer required and aliasing is the main principle behind achieving frequency translation. The advantage of using this method is that the sampling frequency and the consequent processing rate are proportional to the information bandwidth rather than the carrier frequency.

The GPS-SPS signal operates on a carrier frequency of 1575.42MHz and a first null bandwidth of approximately 2 MHz. These parameters provide the desired information to determine the sampling frequency requirement. The minimum sampling frequency is based on the RF carrier frequency, $F_c$, and information bandwidth, $BW_1$ and the minimum allowable sampling frequency $F_s$ in this case is given by

$$F_s > 2 \times \left(\frac{F_c + BW_1}{2}\right)$$

The information within the range $[0, F_s/2]$ is uniquely defined. The choice of this sampling frequency provides the potential to recover any RF transmissions in that frequency range. A problem related to this design methodology is the processing power required for processing the resulting samples. No processor is available which is capable of processing the data at the required speeds.

The other technique, that of using the information bandwidth of the resulting signal to derive the sampling frequency is considered next. The minimum sampling frequency of the information bandwidth is given by

$$F_s > 2 \times BW_1$$
Now in the case of the GPS-SPS signal the information bandwidth can be approximated by the first null bandwidth which is approximately 2MHz. Thus for the present case a sampling frequency requirement of 4MHz is imposed. This is a fairly novel approach in the sense that a signal centered at 1575.42MHz is sampled at 4MHz or above in order to extract the information available. This technique is known as the bandpass sampling\cite{8} in which the signal is intentionally undersampled so as to achieve frequency translation via intentional aliasing. The inherent advantage of using this technique are the availability of a suitable ADC. Sampling is occurring at a much lower rate and hence processing the resulting samples becomes feasible with the available computing power. The trade-offs between using the bandpass sampling and traditional sampling are many. The primary advantage is that the sampling frequency and the consequent processing are a function of the information bandwidth and not the carrier frequency thereby eliminating the constraints imposed due to the ADC and the available computing power. The main disadvantage of using the bandpass sampling technique is the high Q band pass filter that will be required for filtering the signal.

The design used for the GPS-SPS software radio design is shown in figure 3.3. Ideally this should have been similar to figure 3.2 but it is a compromise between the conventional design and the ideal design. The design uses a single stage of downconversion and bandpass sampling. This single stage of frequency translation adds a minimal number of components and significantly reduces the analog input bandwidth requirements on the ADC. A compromise can be achieved without the loss of generality.
GPS-SPS Software Radio Front End Design

Figure 3.3 The GPS-SPS Software Radio Front End Design
The design of the front end used is shown in figure 3.3. The LNA after the antenna amplifies the signal upon reception thereby providing 26dB of gain and is then split into two paths. The first one goes into a commercial GPS receiver manufactured by Plessey Inc, which is just used to track the visible GPS satellites. The second goes to the direct digitization front end design[9]. The signal is connected to a filter and further amplified prior to being mixed with a 1554.74MHz LO. The LO centers the desired frequency component at 20.68MHz. The down converted frequency is further amplified and passed through a narrow bandpass filter directly prior to sampling with an ADC. This design adheres to the software radio philosophy of minimizing the number of components of the front end. Truly speaking the design is a compromise between the traditional front-end design and the ideal front-end design.

The direct digitization[9] bandpass sampling for the case of the GPS-SPS signal has been presented. It is essential to realize that the above theory is applicable to any RF transmission. Multiple GNSS transmissions are well suited for concurrent processing because of the similarity in signal structure. The techniques involved for the front end implementation for the case of a single GNSS transmission have been presented. The first requirement of a software radio that of the front end design has been presented. The next requirement, that of processing the sample on a programmable processor is investigated in detail in the next section.

3.2 Software Signal Processing

The next major aspect of the software radio is the signal processing. Ideally the samples from the ADC are to be processed using a programmable microprocessor. This
provides the ultimate environment in which the costly hardware prototyping will be eliminated. Newer and advanced algorithms can be simply downloaded to the target processor and their performance evaluated. The software radio architecture will replace the application specific IC's (ASIC) used in normal GPS receivers. The ASIC's are utilized for initial processing, digital down-conversion, correlation and accumulation of the GPS signal. The type of acquisition that can be performed with ASIC's is limited. This limitation is taken care of by the programmable processor in the software radio design.

Software algorithms for the software radio can be broadly divided into three main parts; signal acquisition, code tracking and carrier tracking. The first step to be taken in software processing is acquisition. Acquisition is used to determine the specific PRN code, the phase of the PRN code and an estimate of the carrier frequency shift. This is a three-parameter search required prior to the inception of code and carrier tracking. The search space for acquisition, more specifically the PRN code are any of the 32 GPS-SPS PRN codes. The search space is limited to 20840 combinations. The extensive search space involved is a major hindrance in achieving real time performance.

Acquisition provides an initial accurate estimate of the code phase but the line of sight dynamics will perturb the code rate and thus tracking the incoming code is required. This is done in the code-tracking phase. The programmable receiver has to provide the entire signal processing which includes acquisition, carrier tracking and code tracking on samples streaming at 5MHz rate. Given the availability of processor power and the cost involved, this operation will require the use of some optimization techniques, which are
dealt with in detail. Many processor details like the mode of data transfer, techniques like bus mastering[10], which help in transferring data on the PCI bus at the rate of 5MHz have been investigated. The language used for processing is C; a high level language with the required robustness and portability desired from an application of this type. The effects of using C and Assembly language on a DSP[11] chip have also been investigated. The viability of using different platforms as well as different programming platforms with a view to achieving real time performance have also been investigated.
4. Theoretical Background

Processing the signal is the second most important aspect of the GNSS software radio design. Processing of the GNSS signal can be broadly divided into signal acquisition which is used to determine the specific PRN code, the phase of the PRN code and an estimate of the carrier frequency. The second part is tracking which is a carrier tracking loop coupled with the code tracking loop; and its purpose is to track either the phase or frequency of the incoming carrier, depending on the implementation. The code tracking loop is used to generate a synchronized version of the PRN code on the signal used primarily to remove the spread spectrum modulation. These two loops are coupled in that the code tracking loops requires an accurate phase or frequency estimate of the carrier.

The software processing begins with the acquisition process which is the initial step in processing any CDMA transmission. Once acquisition has successfully identified a signal and its associated parameters in the data, control is transferred to the coupled tracking loops which are used to track the incoming phase or frequency and to remove the spread spectrum modulation. The algorithms chosen for software implementation are fundamentally proven algorithms and their performance is well predicted using linear models. The underlying theory behind each of the algorithms used is investigated in detail in this chapter while the experimental results and real time implementation details are presented in the next chapter.
4.1 Spread Spectrum Signal Acquisition

Signal acquisition[13] is the first step in processing the GPS-SPS and the GLONASS signals. Acquisition here is a three-parameter search required prior to the inception of the code and carrier tracking loops. The three parameters for which the search is performed are the specific PRN code, the code phase of the PRN code which is typically within ±1/2 of a chip and an estimate of the carrier frequency shift. The PRN code is the spreading code that is used for a particular satellite. For the case of GPS-SPS the PRN code is the specific C/A code. The search space is limited to any of the possible 32 PRN codes known as the gold codes[3]. The second parameter to be searched for is the code phase which is a search for the specific code alignment in the received data. The purpose of this is to be able to generate a locally synchronized version of the PRN code which will then be correlated with the incoming signal to remove the spread spectrum modulation. The determination of the PRN code phase is usually within ±1/2 of a chip. The C/A code for the civilian user is a relatively short code with a period of \(2^{10} - 1\) which is equal to 1023 bits or a 1ms duration at 1.023Mbps. Determination of the received code phase estimate is used to set the reference code phase. Normally, for initial acquisition when the time is unknown the uncertainty is bound by the entire C/A code. A code search is always performed from the early side of the estimated location of the correlation peak to prevent the acquisition of the multipath signal.

The final parameter in the search space is the carrier frequency, which can be considered to be the intermediate frequency if downconversion or bandpass sampling is used. For the case of the GPS-SPS signal it is equivalent to 1575.42 MHz or the resulting
IF frequency which appears to be deterministic given the way the front end is implemented. Now there are many factors which may cause a deviation and one of them is the line of sight dynamics. The line of sight dynamics creates what is known as a doppler shift in the received carrier frequency. In addition, the Local Oscillator (LO), used to downconvert the incoming signal has a significant effect in the sense that the frequency drift is proportional to the type of oscillator used and this effects the effective frequency. This is taken care of by running a search of ±10 KHz, in 500 Hz steps, about the nominal IF.

Provided no information is available to the user then all parameters in the search space must be fully explored. This is known as the cold start in acquiring and tracking the GNSS signals. Alternately if some information can be had beforehand then the search space can be greatly reduced, say for instance for the case of the GPS-SPS configuration the satellites transmit what is known as the almanac data which provides approximate positions of all the satellites as a function of time along with each satellite’s C/A code. If the almanac data is known then the receiver stores this information when it powers down and when it gets powered up again then it uses the almanac data, its internal clock for a reference time and its last location to identify which satellites are visible and the corresponding C/A code. This technique is known as warm start[14] and is very effective in reducing the search space for the acquisition process. The technique of warm starting the receiver cannot be used in case the receiver has been moved by a significant distance or if it has not been used for a significantly long time such that the almanac data has become outdated. The technique used to perform acquisition is the serial search algorithm.
the nuances of which have been documented very effectively[12]. The serial search algorithm uses the accumulate and dump type of filters to detect the power peaks.

4.1.1 Serial Search Acquisition

The serial search algorithm was one of the first techniques used for a CDMA signal and it uses active correlation until the correct code epoch is determined. In this approach each unknown parameter is evaluated serially until all possibilities are exhausted or the correct combinations of the parameters have been detected. The principle behind the serial search is that the timing epoch of the local PRN code is set and the locally generated PRN code is correlated with the incoming PRN code. The envelope is compared to the threshold value at intervals which are long as compared to the chip duration. If the threshold is not exceeded, then the local code phase slips to the next cell usually one half of the chip. Now when the incoming and the local PRN codes are in coarse alignment the threshold will be exceeded with a high probability and the local PRN code will not be slipped and it will remain in the same phase with respect to the incoming signal and it is then that tracking is initiated.

A block diagram of the serial search algorithm is shown in figure 4.1. The signal is initially multiplied by a PRN code with a specific code phase. The result is then mixed with a local oscillator to get the inphase as well as the quadrature phase components. This composite result is typically integrated over a PRN code period which is 1ms in the case
Figure 4.1 Serial Search Acquisition Algorithm Block Diagram
of GPS-SPS or GLONASS and can be integrated over a complete data bit. The type of integration used is the accumulate and dump type of integration. A longer integration period results in a lower probability of a missed or false detection. The output of the integrator is then tested for the presence of the signal in the manner described earlier. In case all parameters are correct then the result will be a maximum and should exceed a predefined threshold in which case, tracking will be initiated, otherwise a new combination of parameters will be tested. The search process also takes care of the line of sight dynamics which create a Doppler shift in the received carrier frequency by extending the search space to ±10 kHz in 500Hz steps about the nominal IF.

4.2 Spread Spectrum Signal Tracking

Once acquisition has successfully identified the signal and its parameters in the data, the tracking phase is begun. Control is transferred to the tracking loops which are the code tracking loop and the carrier tracking loop. The two loops are coupled together as the code tracking loop requires an accurate phase or frequency estimate of the carrier depending on the implementation. The carrier tracking[3] loop is the first of the two. The purpose of the carrier tracking loop is to track the phase or frequency of the incoming carrier. The second of the two is the code tracking loop and is used to generate a synchronized version of the PRN code and helps primarily to remove the spread spectrum modulation. It is also used to provide timing information useful in the position solution. The tracking loops used with GNSS processing are well understood. The performance of these loops are studied in depth and are predictable and their software implementation
allows for maximum flexibility in implementation as well as in incorporating advanced
signal processing techniques into the GNSS receiver.

4.2.1 Costas Carrier Tracking Loop

Carrier tracking is used to track the phase or frequency of the incoming signal. The basic block diagram for the carrier tracking[3] loop is shown in figure 4.2. The main components of the loop are the phase detector, a loop filter and a voltage or numerically controlled oscillator. The input signal is compared with a locally generated reference to obtain the phase discrepancy between the two inputs, the difference of which is filtered and applied as input to the voltage or numerically controlled oscillator. Now based on this filtered phase error signal, the oscillator adjusts the frequency of the reference signal in an attempt to match the phase of the incoming signal.

The block diagram for the costas loop is shown in figure 4.3. The costas loop is also a feedback type structure and is similar in principle to the figure 4.2 except that the costas loop uses the phase detector function, which is arctangent discriminator function. The main components are the voltage controlled oscillators, the arctangent discriminator function, the loop filter and the low pass filters. The operation of the costas loop can be described as follows. The input signal is the despread carrier, modulated only with the navigation data. The data modulated carrier is mixed with a reference oscillator, generating the in-phase and the quadrature phase components of the signal. The in-phase and quadrature phase components are then lowpass filtered. The arctangent function takes
Generalised Feedback Tracking Loop

Figure 4.2 Generalised Feedback Tracking Loop

Costas Tracking Loop

Figure 4.3 Costas Tracking Loop Block Diagram
the quotient of the baseband in-phase and quadrature components to compute the error signal. This error is then filtered and input to the oscillator to adjust the frequency of the reference signal to match that of the input signal. Now assuming that the loop is operating in the lock operation, which is the state when the frequency of the reference signal matches that of the input signal, then the modulated bits are present on the in-phase arm of the Costas loop, directly after the lowpass filter.

4.3 Early/Late Noncoherent Code Tracking

The Code tracking loop is used to generate a synchronized version of the PRN code on the signal used primarily to remove the spread spectrum modulation. The code tracking loop requires an accurate phase estimate of the carrier. Acquisition provides an initial estimate of the code phase but the line of sight dynamics will perturb the code rate and hence tracking the incoming code becomes essential. The code tracking loop is important as it is used to despread the signal as well as to provide the time-of-transmission measurements critical for range measurements and subsequently a position solution.

The code tracking loop used in the GNSS software radio is the early/late noncoherent delay lock loop. The block diagram for the same is shown in figure 4.4. The block diagram is very much similar in principle to the general block diagram shown in figure 4.2. The input to the loop is the carrier signal modulated with the navigation data and the spreading code. The input signal is split into two paths and is correlated with two
Figure 4.4 Early/Late Non Coherent Delay Lock Loop Block Diagram
versions of the locally generated code; one an early code and the other a late code. The early and the late code are spaced evenly about the perfectly synchronized or the prompt code. Each of these paths including the prompt code are mixed to baseband thereby generating the in-phase and the quadrature phase components. The energy in the late and the early codes are differenced and the result is filtered and input to the voltage or numerically controlled oscillator which clocks the PRN code generator. The whole idea behind doing this is that the error or bias with respect to the prompt signal indicates which path, early or late, contains more energy and thus whether the oscillator needs to speed up or slow down the locally generated PRN code. In the ideal case the two paths are balanced and the resulting difference is zero. This very effectively takes care of the line of sight perturbations in the sense that it ends up aligning the locally generated code with the incoming signal to around ±0.5 chips.

The block diagram in figure 4.4 can be understood to be a lower level description of the generic feedback structure. The majority of the components in this diagram represent the phase detector. The difference of the early and late paths represents the error signal which goes through the traditional loop filter and is used as input to the oscillator. Mathematically the linear phase lock loop of figure 4.4 has been shown to be a theoretical abstraction of the early/late noncoherent delay lock loop[15]. The result is important as it shows that the equations presented in the earlier sections are directly applicable to the design and software implementation of the code tracking loop and the same equations can be used to calculate the appropriate coefficients for the loop filter once the parameters for the loop operation have been specified.
5. Real Time GPS System Design

Real time GPS software radio design involves the ability to continuously store the data from the ADC card and process the resulting information in order to achieve a position solution. The system design should be able to offer the reliability necessary for such high performance computations. In addition to reliability, modifiability and maintenance of the design are also important factors to be taken into consideration. The signal processing for the radio has been designed in a modular manner thereby making the addition of new and faster routines easier. Keeping in mind the above objectives the following experimental GPS system design has been proposed.

5.1 Proposed Implementation

The software radio implementation can broadly be classified under three main categories the front end design, the Data Acquisition Board[16] and Signal Processing. The front end configuration of the design has been discussed in chapter 3 and the actual front end design is shown in figure 3.4. The front end design is based on the information bandwidth of the GPS-SPS signal for which the first null to null bandwidth is 2 MHz resulting in a use of a 5MHz sampling frequency, which will alias the GPS-SPS center frequency correspondingly.

The implementation strategy and the block diagram of the proposed implementation are shown. The implementation is in two stages 1) the first is the data collection board and 2) a programmable microprocessor. The data will be transmitted from the ADC card to the Processor using the PCI bus. As is evident the ADC is an
Proposed Implementation

Figure 5.1 Proposed Implementation of the GPS-SPS software Radio
integral part of the development process. The data streaming out of the ADC card is to be transmitted over the PCI bus to the Processor, which then processes the data. The rate at which the data is streaming out is 5MHz and an appropriate ADC is required.

The data acquisition board in this case is the COMPUSCOPE 6012/PCI from Gage Applied Sciences Inc[17]. The samples from the ADC are to be processed on a programmable microprocessor. The disadvantage of processing the resulting samples exclusively in software is the availability and cost of the required programmable computational power. A real time GPS software radio will require a microprocessor capable of providing the required signal processing capabilities on samples streaming at 5MHz rate, which as per the available computational resources is quite a challenging task.

The most important consideration, which was taken into account while deciding on an ADC, was real time operation. An ADC was required which works on the PCI bus architecture; which is capable of sampling at a rate of 5 MHz; which will be capable of transmitting the resulting samples most efficiently; and one which works in a real time operating system. The COMPUSCOPE 6012/PCI[17] was the board which came the closest to meeting our requirements. The CompuScope 6012/PCI can sample analog signals at speeds up to 60 MSPS with 12-bit resolution and either store the data in the memory on-board the 6012/PCI board or read it directly into the PC's memory. The COMPUSCOPE 6012/PCI provides excellent support for real time operating systems such as QNX while at the same time providing driver support and capability for 32 bit operating systems like Windows-95 and WINDOWS NT.
The GaGe product in its present form is not very efficient in terms of the data transfer rates. The 6012/PCI board provides for 12 bit samples and packs them in 2 bytes and transfers them over the PCI bus onto the PC’s memory. The disadvantage of using this transfer rate is that the effective bandwidth that can be transmitted gets reduced. The PCI bus is capable of data rates up to 120mbps. Since the samples are packed and then transmitted over the PCI bus 80mbps of the bandwidth[18] is taken up by the transfer leaving 40mbps of the bandwidth unused. This 40mbps of bandwidth is just not enough to send the data back to the processor hard disk. The data acquisition board should have the capability to provide an 8 bit data rate or a 4 bit data rate. The 4 bit data rate is very effective in the sense that the bandwidth occupied is just 20mbps[18] thereby giving the designer of the software extra clock cycles to play with. Due to the high data rate of 12 bits the bandwidth left to do other processing like writing data to the hard drive becomes a difficult task. Even though the speeds involved in reading/writing[19] data to/from the hard drive are faster as compared to writing/reading data to/from the external device the remaining bandwidth does not permit a high speed read/write operation. On the contrary with a date rate of 4 bits the bandwidth constraint is only 40mbps and the remaining 80mbps of bandwidth is freed up to do other operations. This gives a greater flexibility to the design process. The dynamic range is to be taken in consideration while deciding on the number of bits that can be used. The more number of bits the more the dynamic range. The advantage of using more dynamic range or in other words using more bits is that it facilitates the tracking of the signals which are very deeply buried in noise. The
ideal case is a PCI based acquisition board with a sampling frequency of 5MHz and one which packs bits in 8 bit samples or 4 bit samples rather than 12 bit samples.

The computing processor is a host PC capable of buffering the data streaming out of the ADC card and then processing the data. The host PCI bus is used as the medium of communication between the acquisition card as well as the host PC. As was stated earlier the sampling rate of the signal after downconversion is 5MHz. The five million samples from the data acquisition card are to be transferred over the PCI bus onto the onboard memory of the host PC which then begins to process the data.

Signal processing comes into picture once the data is transmitted to the onboard memory of the host PC. The signal processing is basically a 3-parameter search consisting of the PRN code, the Code phase of the PRN code and an estimate of the carrier frequency shift prior to the inception of the code and carrier tracking which mainly concerns itself with tracking the phase or frequency of the incoming signal as well as to generate a local version of the PRN code used primarily to remove the spread spectrum modulation on the incoming signal. Processing the signal is costly in terms of the clock cycles expended. The first problem is the buffering of the 5 million samples in one second and providing the capability to process the signal and compute a position and velocity solution. The first part of signal processing which is acquisition is in itself a costly operation as it runs a serial search for the 3 parameters required wherein it requires a search of 20480 values. The generation of the Inphase and Quadrature phase components of the incoming signal requires the use of the math library for the generation of the sine() and cosine()[20] functions which in itself are a costly operation. Keeping
these considerations in mind the PC chosen was the state of the art Pentium Processor operating at 300Mhz. The software was tested on a QNX, Windows 95 and Windows NT environments using the Watcom C/C++ compiler and MS Visual C++ compilers. The call tree diagram for the execution of the program has been described in figure 5.2. The diagram steps through the various important phases in the development process. The whole process begins with the main program in which the data is initialized and the various system parameters needed for system performance are initialized. Also the files needed for input and output are declared. The next step is the generation of the Coarse/Acquisition code. Once the C/A code has been generated the next step is the computation of the various system parameters like the number of samples and the code base rate. The look up table approach is used to generate the values of sine and cosine for the corresponding frequency offset. The next step is the generation of the inphase and quadrature phase components which are nothing but the multiplication with sine and cosine terms respectively. The next step is to use the look up table approach and calculate the values of the early/late and prompt codes respectively. Once the codes have been generated we can either search for the code phase and make adjustments as well as initiate Code tracking. The difference in energies for the early and late signals is calculated and the signal is appropriately adjusted. The next step is to begin the carrier tracking phase. The carrier tracking phase has to provide a correct estimate of the frequency estimate as this is a very integral part of the entire process of tracking. If the frequency estimate is inaccurate the generation of sine and cosine values at that
Main()

Initialize Data
Declare Files, Variables
Define Parameters for Acquisition, Tracking

Generate C/A Code
Store C/A Code For SV

Read Concerned Files For The Generation Of Sine() & Cosine() Variables

Compute The Number Of Samples, Code Periods Based On Code Base Rate.
Define The Point At Which To Begin Reading Input Data Using Seekset & Read Data
Perform the Following For The Total Number Of Code Periods

Generate Sin() & Cosine() Values Based on the Sampling Rate
Use The Lookup Table Approach To Read In The Correct Values Corresponding To The Correct Frequency Estimate

Figure 5.2a Call Chart For The GPS-SPS Software Radio Signal Processing
Use Look Up Table Approach To Generate The Early/Late/Prompt C/A Codes Multiply The Input Data File & Sine, Cosines To Generate inphase & quadrature Phase Components

Mix The inphase And quadrature Phase Components With The Early Late C/A Codes To Generate Early/Late/Prompt inphase & quadrature Phase Signals

- EARLY inphase & quadrature Signal
- LATE inphase & Quadrature Signal
- Prompt inphase & quadrature Signal

Initiate Code Tracking

Using The Above Values Generate \( \text{Late}(I)^2 + \text{Late}(Q)^2 \) And \( \text{Early}(I)^2 + \text{Early}(Q)^2 \) Compute Discriminator Function

Test If Signal Is Present Determine Code Phase

Output Code Phase

Figure 5.2b Call Chart For The GPS-SPS Software Radio Signal Processing
Calculate Error Coefficients
Compute New Code Rate

Initiate Carrier Tracking
Compute $\text{atan}(\text{prompt}(i))$
$\text{prmnt}(q))$

Compute Error Coefficients
Adjust The Frequency Value

Figure 5.2c Call Chart For The GPS-SPS Software Radio Signal Processing
carrier frequency is not accurate and this may lead to errors. Now this entire process is performed for the duration of the code periods. There is a feedback loop which runs for the total number of code periods. Now the values for the early, late, prompt codes which are generated during the process are written out and this whole procedure is repeated for the specified number of code periods. Once the data is written out then it can be processed appropriately. Bit synchronization[3], Frame synchronization and Parity decoding can then be attempted.

5.2 Current Bottlenecks

The current analog to digital converter from GaGe systems has the capability to operate at the desired sampling rate which is around 5MHz and use the PCI bus to transfer the data from the on card memory to the host PC’s memory. The current acquisition board has a number of disadvantages some of which are discussed in the later paragraphs.

The acquisition board works in three modes 1)Trigger view 2) Memory mode and 3) Real time mode[17]. The Acquisition board samples the data; buffers the data and then transfers the data onto the host PC’s memory. The working of the data acquisition board has been documented well.[17]. The acquisition board is controlled by the driver and as soon as the driver starts out a specified memory block of 64kbits is specified[21]. The memory allocated is a continuous piece of memory and it is the contents of this block of memory that gets written to the hard drive. The GaGe card has a on board FIFO which is very small; almost the size of 32kbits. The way the GaGe card works is that at start up
the driver initializes a contiguous piece of memory, data is written to the on board FIFO and as soon as the FIFO gets full the PCI bus is given priority and the data is transferred. The driver has infinite priority and the acquisition board has been set up in such a way that the PCI bus is given control the minute the on board RAM gets full.

The next aspect of operation is the memory chunks that are written. The contiguous blocks of memory that are allotted are 64k[21] bits in size. They contain header information which remains the same. The first block of data is written into the space immediately after the header and the board gets back to sampling the data once again. Once the FIFO is full again the data is written onto the block of memory immediately after the first one. This process is continued until the whole block of memory is filled with data. Now once the block is full the next stream of data is written back on the space immediately next to the header. The memory block can be thought of as a circular buffer. The acquisition board has been set up in such a way that the driver has infinite priority as compared to other devices and the driver grants control to the PCI bus the minute the on board FIFO gets full.

The data acquisition board in the memory mode as well as the trigger view mode packs samples to ease the constraints imposed on the bandwidth. In the case of the real time operation the samples are not packed, they are instead padded. The samples are padded and 2 samples are sent across the bus in 3 bytes. Padding is necessary as the FIFO can’t be empty at any time. The main drawback of this is again the bandwidth lost in transferring the data. With every 3 bytes of data transferred there is one byte of padded
data which gets sent across. As stated earlier this does not help in freeing up some of the bandwidth required to do the processing.

The sampling rate constraints imposed on the data acquisition board are a matter of concern while taking into account the performance characteristics of such a board. Currently no board exists in the market which can sample at the rate of 5MHz, have a data rate of 8 bits or less, provide bus mastering capability, use the PCI bus for data transfer and finally provide the capability to buffer and sample a continuous stream of data over an extended period of time. Ideally a board is required which can provide the capability to sample at rates similar to or slightly higher that the one provided for the GNSS software radio. The board should provide capability to sample and buffer contiguous streams of data. The whole idea of providing the data transfer capability is to avoid having the processor incur heavy costs in terms of clock cycles. Currently significant development is taking place in the Avionics Engineering Center labs to develop such a board which will meet the requirements imposed by the software radio design philosophy.
6 GPS – SPS Software Radio Implementation

The theory and principles behind the software radio implementation have been presented. The implementation strategy with regards to the front end design, the software processing as well as the bottle necks involved with the project have been discussed. The next step is the complete design and implementation of the GNSS Software Radio.

The software algorithms have been coded in C[22] and the platform used for signal processing is a Pentium processor. For the purpose of verifying the authenticity of the software algorithms a 30 second window of data, or 150MB of raw GPS data was collected and used as the test bed. The unprocessed data appears as noise in the CDMA format the plots of which have been shown in the latter discussions.

The implementation of the Software Radio Signal Processing as stated earlier is broadly divided into Acquisition, Code and Carrier tracking. The GPS signal is a CDMA spread spectrum modulated signal centered at 1575.42MHz; the sampling frequency used in this case is 5MHz and this is used to center the carrier signal to 1.25MHz. The first step in the processing of the signal is the generation of the local version of the C/A code which is discussed next.

6.1 Coarse/Acquisition Code Generation

The spreading or PRN code for the GPS-SPS signal is known as the Coarse/Acquisition(C/A) code. The C/A code has a chipping rate of 1023 chips and a period of 1ms. The C/A codes are a subset of the Gold code[3] family, a collection of PRN codes which provide rapid acquisition. The C/A codes are selected to provide good
multiple access properties for their period. The C/A code is formed by the product of 2 equal period 1023 bit PRN codes G1(t) and G2(t) and is represented as follows

\[ XG(t) = G1(t) \times G2[t+N_i(10T_c)] \]

Where \( N \) determines the Phase offset in chips between G1 and G2. The C/A code chip has a duration of 10 \( T_c \) where \( T_c \) is the P code chip interval. Each code G1 and G2 are generated by a maximal length linear shift register of 10 stages. Initially the two registers are all set to 1’s. The tap positions are specified by the generator polynomial for the two codes as follows

- \( G1: 1 + X^3 + X^{10} \) (6.1)
- \( G2: 1 + X^2 + X^6 + X^8 + X^9 + X^{10} \) (6.2)

the block diagram implementation of the same is shown in figure 6.1.

The block diagram shows two 10 stage feedback shift registers clocked at 1.023MBps having feedback taps at stages 1 and 10 for G1 and at 2,3,6,8,10 for G2. The various delay offsets are generated by tapping off at appropriate points on the G2 register and modulo-2 adding the two sequences together to get the desired delayed version of the G2 sequence. The code tap positions[3] for the various codes are specified by the DOD. In the current implementation they are stored in a 32 array element. The code tap position for example for satellite vehicle 6 is 17 and these values are appropriately stored.

One important point to be discussed first is the data types used to store the various variable involved. The idea is to minimize the size of the generated code by reducing the number of bits in which the data is stored. Special care has been taken to store all the
C/A CODE GENERATOR

Figure 6.1 Block Diagram Of The C/A Code Generator
variables in the form of integers and short integers as they take up 4 and 2 bytes of memory respectively. This helps in the sense that the memory overhead used in performing the operation is less and it also aids in reducing the code size of the executable.

The C/A code generation proceed by first generating the G1 code, the G2 code, shifting the G2 code appropriately using the tap positions and finally generating the modulo-2 sum of the two. The G1 and G2 codes are generated by using a shift register, which in this case happens to be a array of 10 elements and using the tap positions as stated above for both G1 and G2. For example for the generation of G1 code the array elements 3 and 10 are multiplied and the value is stored in a separate variable. Next the contents of the register are moved one bit to the left and the contents of the last register which in this case is the 10\textsuperscript{th} element are stored for both the registers thereby giving the G1 and G2 codes respectively. This whole operation is performed for the entire C/A code duration which in this case in 1023 bits.

The next operation to be performed is the shifting of the G2 code based on the satellite being tracked. The code tap positions are unique for each satellite and are stored in a 32 bit array as stated earlier. Based on the satellite that is being tracked the code tap position is appropriately picked. For example, if satellite vehicle number 6 is being tracked then the 6\textsuperscript{th} element of the array is selected which in this case happens to be 17. Now once the code tap position is selected based on the satellite number the next step is to shift the G2 code appropriately. Continuing with the analogy of the satellite vehicle 6 for which the tap position is 17, the shifting of the G2 code for the same is done in the
following manner. The last 17 elements of the G2 array are stored as the first 17 elements of a temporary array gtemp. Next the first 1007 elements are stored beginning from position 18. This can be diagrammatically depicted as shown in figure 6.2

![Diagram of G2 Code Shifting](image)

Figure 6.2 Shifting Of The G2 Code Based On the Satellite Vehicle Number

The above figure is based on the results obtained for satellite vehicle Number 6. As is shown in the figure the last 17 elements are moved to the first 17 elements of the temporary array while the first 1006 elements are moved to positions beginning from 18 and ending at 1023 thereby giving the G2 code consisting of an array of 1023 elements.

Once the G2 code has been generated and appropriately shifted to account for the particular satellite which is being tracked the next step is the generation of the actual C/A code which is simply the product of the G1 and the G2 codes; which in turn yields a C/A code of 1023 bits. Now that the C/A code has been generated there is one issue to be accounted for and this is the adjustment for the desired sampling frequency. The C/A
The code is 1023 bits in length and the sampling frequency used in this case is 5MHz which leads to the information that the total number of samples in 1 ms of data are around 5000. This 1023 bit C/A code has to be spread out over the whole 5000 samples i.e. the code rate of 1.023MHz has to be spread out over the entire sampling frequency of 5MHz. This is done using the following relationship

\[
\text{Code rate} / \text{Sampling Frequency} \times (1 : \text{Number Of Samples}).
\]

The results for this are 0.2046, 0.4092, 0.6138, 0.8184, 1.023 …… This shows that for the first 4 samples the C/A code element 0 is selected and for the next 5 elements the C/A code element 1 is selected and so on thereby evenly spreading out the entire 1023 bit C/A code over 5000 samples. The computation of the above has been done in a generic form and no change to the program code needs to be made to accommodate different sampling frequencies.

The generation of the C/A code is an important aspect as the C/A codes vary depending on the satellite vehicle that is being tracked. Also the C/A code is necessary as the locally generated C/A code is used to align with the incoming signal and calculate the PRN code phase. The C/A code for this application had been written in the form of a function call. With real time performance in mind and the need for additional clock cycles to process the data it was decided to implement the C/A code generation in the form of a lookup table approach. In this approach the C/A codes for all the satellites are generated and stored in files with the appropriate names. The file with the corresponding satellite number is read in and the data is stored appropriately. The advantage of this is
the clock cycles that can be freed up for use. By not using the C/A code in the form of a function call we are effectively reducing the overhead associated with such a call, like storing the contents of the registers onto the stack, storing the return address. The flow of execution is disrupted thereby reducing the very purpose of using system-generated optimization techniques like software pipelining. Another advantage is that the whole operation of generating the C/A code is limited to just one READ operation which is considerably less expensive in terms of clock cycles[22] as compared to generating the C/A code on the run.

This method of implementation has a significant disadvantage in the sense that the code size increases appropriately as a new buffer needs to be allocated to store the values of the C/A code. An optimum balance is desired between the actual executable size and the speed of execution. Since real time performance was desired it was decided to go ahead with the second implementation technique at the expense of code size.

6.2 Signal Acquisition

The next step to be performed is Acquisition. Acquisition is the process of determining the PRN code, the PRN code phase and the carrier frequency. The PRN code is the spreading code which in this case is any of the possible 32 GPS-SPS PRN codes. The PRN code phase, within ±1/2 of a chip allows a locally synchronized version of the PRN code to be generated which is then correlated with the incoming signal to remove spread spectrum modulation. The final parameter in the search is the carrier
frequency, which can be considered the IF in case downconversion or bandpass sampling is used.

The detailed block diagram of figure 4.1 explains the process involved in performing serial search acquisition. The signal is multiplied by the PRN code with a specific code phase. The result is then mixed with a local oscillator to get the in-phase and quadrature phase components. This composite result is integrated over a complete code period which is 1ms in the case of GPS-SPS. The output of the integrator is tested for the presence of the signal.

The data collected with the front end design is shown in figure 6.3. The noise levels as shown in the figure are very high and the actual signal is buried deep within the noise. The purpose of acquisition is to test for the presence of the signal and list out the specific PRN code phase. The various parameters that are used in the acquisition process are listed. The sampling frequency used is 5MHz, the expected carrier frequency, which is the IF, is defined to be 1.250250MHz and the LO frequency drift is taken to be 500MHz.

A locally generated version of the PRN code can be generated by using a function call. The advantages and disadvantages of doing the same have been discussed earlier. Keeping in mind that real time performance is required, the overhead of generating the C/A code is replaced by a read and store operation. The C/A codes pertaining to each satellite have been stored in files with the name casat*.dat where “*” represents the satellite for which the code is required. The program is made robust enough that the user need only enter the satellite vehicle number which is being tracked.
Figure 6.3 Plot Of The Unprocessed GPS Data
and the program will open the file corresponding to that satellite number. For example for satellite vehicle number 17 the file casat17.dat is opened and the data read.

Once the locally generated version of the C/A code has been generated the next step is the generation of the in-phase and quadrature phase components at baseband. The process can be accomplished by using the built in standard math library functions sin() and cos(). The disadvantage of using the built in functions is that they are costly in terms of clock cycles consumed[20]. The standard functions take floating point arguments which are stored in 8 bytes thereby increasing the code size. The library functions sin() and cos() require 55 clock cycles on a pentium processor[20] to compute one value. The total number of samples in one code period are 5000 based on the sampling frequency. The total overhead in generating 5000 sine and cosine values using the function calls was too high. An alternative inexpensive method for generating the sine and cosine values was required.

An alternate method of generating the sine and cosine values was considered. This method is based on the cyclic properties of the sine wave. A sine wave is shown in figure 6.4. One of the properties of the sine wave is that the values are cyclic and repeat themselves at certain intervals. The sine wave is periodic in nature in the sense that sin(θ) is equal to sin(N2π + θ) ie the values of the sine wave are identical at every 2π intervals and it is this property of the sine waves that can be used to good advantage. The generation of the sine and cosine values is based on the lookup table approach. The values of sine and cosine are generated from 0 to 2π and are stored in the appropriate files which will then be read in at run time. As is shown in the figure 6.4 the entire sine
Figure 6.4 Plot Of The Sine Wave For The Sin()/Cos() Lookup Approach

\[ \text{Pstep} = 2^{32} \div 4 = 2^{30} \]

Start = P0 = 0

\[ \text{P1} = 2^{30} \text{(Pstep)} \]

\[ \text{P2} = 2^{30} + 2^{30} = \text{P1} + 2^{30} = \text{P1} + \text{Pstep} \]

\[ \text{P3} = 2^{30} + 2^{30} + 2^{30} = \text{P1} + \text{Pstep} + \text{Pstep} = \text{P2} + \text{Pstep} \]

\[ \text{P4} = 2^{30} + 2^{30} + 2^{30} + 2^{30} = \text{P1} + 3 \times \text{Pstep} = \text{P3} + \text{Pstep} \]

\[ \text{P5} = \text{P1} + 4 \times \text{Pstep} = \text{P4} + \text{Pstep} = \text{P1} \]

Similarly we have

\[ \text{P6} = \text{P1} + \text{Pstep} \text{ or simply P2} \]

\[ \text{P7} = \text{P2} + \text{Pstep} \text{ or simply P3 and so on.} \]
wave can be divided into four quadrants simply as P0, P1, P2, P3 and finally P4 to correspond to 0, \( \pi/4 \), \( \pi/2 \), \( 2 \times \pi/3 \) and finally \( 2\pi \). These values of P0, P1, P2 etc are repetitive and occur again at P5, P6, P7, P8, P9 and P10. The idea here is to store only the values P0, P1, P2, P3 and P4 and use these values to construct the rest of the sine wave and that is essentially the whole principle behind the lookup table approach.

The expected carrier frequency as derived from the front end design is \( f_c = 1.25\text{MHz} \) while the sampling frequency is \( f_s = 5\text{MHz} \). The value of \( P_{step} \) is defined using the following equation

\[
P_{step} = (f_c + f_s) \times 2^{32}
\]

Where \( f_c \) the expected carrier frequency

\( f_s \) the sampling frequency.

The sine and cosine values for a signal with a center frequency of 1.25MHz and sampling frequency of 5MHz are generated and stored in separate files which are then read in at run time. The initial value is taken to be 0. The top 24 bits of the vector \( pvalue \) are masked and shifted 24 places to the right to form an 8 bit number between 0 and 255. The 8 bit number is used as an index to pick out the value from the file. The advantages of using the lookup table approach as compared to the traditional approach are plenty. The whole process of generating sine and cosine values has been replaced by a read, a mask and shift operation which are considerably less expensive than using the standard library functions.

The next step in the process is the generation of the C/A code. As was discussed earlier the C/A code can be generated by using a single function call. The same can be
modelled using the same lookup technique[23] as was used for the generation of the sine() and cosines(). As was stated earlier the C/A codes pertaining to all the satellites are generated separately and stored in files and are read into an array catable. Here “pincr2” is used and this is the increment value by which the C/A code values are read. The value of pincr2 is defined using the following equation

\[ Pincr2 = (\text{coderate} \times 2^{32}) \div (\text{Numsamp}) \div (\text{Numbits}) \]

Where the code rate is 1.023MHz

\[ \text{Numsamp} \] is the number of samples = \(1.02e-3 \times f_s\)

\[ \text{Numbits} \] is the number of bits to be masked

The \text{Numbits} is based on the total number of bits that are to be masked which in this case is 10 and hence the value of \text{Numbits} is \(2^{10}\) which is equal to 1024. The idea here is to basically take the top 10 bits of the number and use them. We proceed by masking the value of the number which has 1’s in the top 10 places. This number is taken and a bitwise AND is performed so that the lower bits go to 0 and the upper 10 bits remain the same ie a logical AND operation is performed so that the top 10 bits remain the same and the rest go to 0. Now shifting the value thus obtained by 22 places gives us a number which has 1’s in the lower 10 bit positions. This number \text{index2} is used to pick the C/A code values. The C/A code table as we know is not an even power of 2; it has 1023 elements rather than 1024 elements and this property of the C/A code forces us to use a IF statement to round off values going above 1023. Now whenever a value goes above
1023 then it is reset to 0 and a number 419304 is added to \( p\text{value} \). Here we are basically adding a 1 in the 10 bit position so that the top 10 bits go to 0 and the rest remain the same. Whenever the value of \( \text{index2} \) is within the range then is incremented by \( \text{pincr2} \) and the whole process is repeated for the total number of samples present.

An alternative to this approach is to try to use the top 1 bit rather than the top 10 bits. Here we perform a logical AND to set the lower 31 bits to 0 and the top bit is shifted by 31 places to the right or 1 place to the left. Now a check is ran on this value and whenever this value is greater than 0 then it is incremented by 1 and \( p\text{value} \) incremented by \( \text{pincr2} \). Now as the value increases it is again tested to see if it reaches the upper bound which in this case is 1023 and another check is performed. If the top bit value has reached 1023 then it is set to 0 in the manner similar to the one described earlier and the whole process is repeated for the total number of samples present.

The next important aspect of the acquisition process is the signal itself. The data acquisition board samples at the rate of 5 million samples per second. It is this data which is to be read again in the program. The valid data bits here are the top 8 bits. So an important implementation strategy is to try to reduce the time and clock cycles involved with reading the data in. It is advantageous to have the data acquisition board simply pack the samples and send them over to the host PC in the form of 8 bit samples. For the purpose of testing the software the 30 second window of data collected with the front end design is formatted and saved as 8 bit characters. Testing has shown that it is faster to read in 8 bit signed characters as compared to reading in 32 bit integers[24]. The data file is stored under the name \( \text{srvychar.bin} \) to signify the fact that it is data in the signed
character format. Another advantage of using the signed characters as compared to integers is that the multiplication operation is faster with signed characters than with integers and this saves some valuable clock cycles.

An issue that requires mention here is that of the Doppler. The Doppler as was stated earlier is the frequency drift value and a value of 500 is selected. What this means is that the search for the various parameters is conducted within a search space bounded by \((f_c \pm \text{maxdop})\). The advantage of using a \text{maxdop} value will become clear as we move onto tracking wherein the line of sight perturbations play an important role. The next step is the actual mixing of the signal which is achieved by multiplying the C/A code with the \text{sine()} and \text{cosine()} signals to achieve the in phase and quadrature phase components. The in phase and quadrature phase components are to be mixed with the actual signal and integrated over a single code period which in this case is 1ms. The integration is performed by first determining the total number of samples in 1ms of data. The total number of samples is computed by using the following equation

\[
\text{No of samples} = \text{code period(1ms)} \times f_s (5e6) = 5000
\]

The composite signal is then integrated over 5000 samples. Integration here is achieved by performing an addition operation over the complete code period. The sum over an entire code period is stored in convenient locations. Once summation over the entire code period is performed the test for the presence of the signal is performed. If all the parameters used in the program are correct then the result should be maximum and the should exceed a predefined threshold. This is done by using an IF statement and the maximum value is determined. The search ends with the result which is the best carrier
frequency estimate to within ± 500Hz and the best estimate of the code phase which is specifically the sample number in the raw data where the code periods actually starts. The program was tested for different satellites as is shown by table 6.1. The table below lists the satellites identified in the collected data as well as the specific code phase and the carrier frequency. For example for the satellite vehicle number 17 the code phase estimate is 470 while the frequency estimate is 1.25e6. For the purpose of conducting the tests the satellite vehicle 17 will be used.

As was discussed earlier the search for the code phase took into account a term called \textit{maxdop}. This \textit{maxdop} is to account for the line of sight perturbations as well as the oscillator frequency uncertainties. The use of \textit{maxdop} is to take care of the frequency drift. The search process is limited to a range of center frequency ± \textit{maxdop} which effectively means that the search in this case is conducted in 3 bins. The determination of the number of frequency bins to search for has been dynamically implemented by using the following relationship

\[ 2 \times \text{maxdop} / \text{freqstep} + 1 \]

where \textit{maxdop} is the maximum doppler to be accounted for

\textit{freqstep} is the no of steps.

The frequency bins through which the program cycles are dependant on the value of \textit{maxdop}. For example changing the value of \textit{maxdop} to 8000 will result in conducting the search within 33 bins above and below the center frequency.
Table 6.1 Satellites identified in the collected data through the Acquisition Process

<table>
<thead>
<tr>
<th>Satellite PRN#</th>
<th>Code Phase (samples)</th>
<th>Frequency Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2885</td>
<td>1246000</td>
</tr>
<tr>
<td>10</td>
<td>3815</td>
<td>1245000</td>
</tr>
<tr>
<td>17</td>
<td>470</td>
<td>1250500</td>
</tr>
<tr>
<td>23</td>
<td>2201</td>
<td>1250000</td>
</tr>
<tr>
<td>26</td>
<td>2665</td>
<td>1247500</td>
</tr>
</tbody>
</table>

Figure 6.5 shows the results obtained for the satellite vehicle 17. As is clear from the graph the search is conducted in three frequency bins. The first figure in the series
shows the ambient noise associated with the signal but it is clear from the figure that the signal is present but the effect of noise is very high and for some cases the signal may not be clearly visible. The next plot shows the signal with the appropriate peak value and the level of noise is very low. Same is the case with plot 3 on the same figure. The obvious question arises where exactly is the signal centered. The answer is that plot 2 is a more accurate estimate of the acquisition process as compared to plot 3. A close look at the two plots reveals the intensity levels for both and as is clear the second plot is at a higher level than the first and that is the more accurate of the two. In order to substantiate the results additional tests were conducted on various satellites and the results plotted. Figure 6.6 shows the result for satellite vehicle number 6. The results of the plots match the expectations. It is worth mentioning at this point that the accuracy requirements from the acquisition process are very high. Acquisition has to provide a very good estimate of the code phase rate. This code phase rate is used in demodulating the information and the carrier frequency estimate that it provides is very helpful during the tracking process.

The acquisition process helped in calculating the code phase estimate as well as the carrier phase estimate to within a particular Doppler range. Once the acquisition process has successfully identified the signal as well as the associated parameters the control is transferred to the tracking loops.
Figure 6.5 Plot of the Acquisition Process for satellite vehicle number 17
Acquisition Process For SV 6

Figure 6.6 Plot of the Acquisition Process for satellite vehicle number 6
6.3 Signal Tracking

Tracking phase is initiated once the signal parameters have been successfully identified. The purpose of the carrier tracking loop is to track the phase or the frequency of the incoming signal. It is also responsible for the demodulation of the data encoded on the carrier. The code tracking loop generates a synchronized version of the PRN code and is used primarily to remove the spread spectrum modulation. The principle behind each of these two tracking loops have been discussed in detail in chapter 4. The code tracking loop will be attempted before we attempt carrier tracking.

6.3.1 Early/Late NonCoherent Code Tracking

The line of sight perturbations will perturb the code rate and hence tracking the incoming signal becomes important. Tracking the incoming signal is important in the case of any spread spectrum receiver as the locally generated version of the PRN code is used to despread the signal for further processing. The tracking loop used here is the early/late noncoherent delay lock loop. The input signal is correlated with two versions, an early and a late, of the locally generated PRN code. The two versions are equally spaced, typically ±0.5 chip, about the synchronized or prompt code. Each of these two paths are mixed generating the inphase and quadrature phase and the energy in the early and the late paths is differenced and the result is filtered and input to the voltage controlled oscillator which either speeds up or slows down the locally generated PRN code.
The lookup table approach used for the calculation of the C/A table values can be
applied here with some modifications. The idea here is that the early C/A code is going to
be 2 samples ahead and the late C/A code is going to be 2 samples late. As is shown in
the figure 6.8 the prompt code is shown with each block representing a bit. The C/A code
is 1023 bits long. The late code is two samples late which means that the C/A code comes
in two samples late with respect to the prompt code. As shown in the figure the late
samples differ from the prompt samples and are slower as compared to the prompt code.
Similarly we have the early code which is two sample ahead of the prompt code.

The generation of the early code and the late code is based on spacing the prompt
code appropriately. One of the implementation strategy was to try to generate three

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>5</th>
<th>1022</th>
<th>1023</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>1023</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>.........</td>
<td>1022</td>
<td>1023</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Prompt Code**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1021</th>
</tr>
</thead>
<tbody>
<tr>
<td>1022</td>
<td>1023</td>
<td>1022</td>
<td>1023</td>
</tr>
</tbody>
</table>

Figure 6.7 Early/Late Samples With Respect To The Prompt Code
different C/A codes. The disadvantage of using this was that this was a redundant process in the sense that the same values will have to be generated thrice. This obviously resulted in expending more clock cycles than was necessary. The ideal situation is one in which the early and late codes are generated from the prompt code as the early and late codes are the C/A codes itself though delayed by a certain amount of chip spacing. It was realized that it is more effective to generate the late and early C/A codes from the prompt code itself. The lookup technique discussed earlier for generation of the C/A code values has been extended to design a new technique to compute the C/A codes at a considerably less cost.

The prompt C/A code was generated using the lookup table using the mask and shift operation[23]. The same approach is used here and we define a new variable known as *earlylate*. This value is twice the value of the variable *pincr*. The value of *pincr* is defined to be $2^{31}$. The value of *earlylate* is dependent on the number of samples that need to be delayed. In the present case the chip spacing is $\pm 1/2$ and so the number of samples to be delayed is 2. The value of *earlylate* twice the *pincr* value as the *pincr* value is used to move on from one sample to the other. The contents of *pvalue* point to the current sample number in the CA code. Now if *pvalue* points to the current value then it follows that *pvalue + earlylate* should point to the element that is 2 places away from the current value and *pvalue - earlylate* should point to the element that is 2 places before the current value. This is depicted more graphically in figure 6.8.
STATE A: This state depicts the actual C/A code generation which we shall call the prompt code. The \textit{pvalue} parameter at this stage is \(2*pincr\). Now to move onto the next element the \textit{pvalue} term has to be incremented by \textit{pincr}. So at stage C the value of \textit{pvalue} is \(pvalue + 2*pincr\) which is basically like 2 samples apart. Now this value \(2*pincr\) is stored as another variable \textit{earlylate}. Now since our CA code is stored as a circular buffer of 1023 elements this technique can be extended to include the early values.

Now continuing from the C/A code generation using the top bit to map as an index into the C/A table array a comparison is made to determine if the value of \textit{pvalue} + \textit{earlylate} is greater than or equal to what the value should be at sample no \(i + 2\). If the condition is true then the index value is checked to see if it is greater than 1023. If the index value turns out to be greater than or equal to 1023 then it is set back to zero. This as was stated earlier is the method of implementing a circular buffer[23] of 1023 elements. The C/A code is 1023 bits long which is not an even power of 2 as a result of which it is not possible to implement a circular buffer. The circular buffer is implemented using an IF statement at the cost of breaking the software pipeline. Getting back to the
implementation if the index value is not greater than or equal to 1023 then it is incremented by 1 and used as the index in the array `catable` thereby giving the late C/A code.

The implementation of the early C/A code is also similar to the technique used to generate the prompt codes. The important point to remember here is that the C/A code is being implemented in the form of a circular buffer of 1023 elements. From the theoretical description as well as the block diagram it is very much clear that for the early code the value of `earlylate` has to be less than the current value of `pvalue`. A test is ran to see if the result is true. If the value of `earlylate` is definitely less by twice the `pincr` value then the current index value is used to index into the array `catable`. Now what about the situation when the current value of the index is equal to 0. Theoretically when the prompt code at 0 is being generated means that the early code is at 1023 and that is taken care of by using another IF statement and if the result is true setting the value of `index` to 1023 and beginning the whole process once again.

The main drawback of using the technique listed above is the use of multiple IF statements. IF statements tend to disrupt the flow of execution and an extra test is to be performed every time an IF statement is performed. This leads to an increase in the number of clock cycles needed to perform the operation. Another implementation methodology considered was to make use of function calls for the generation of early as well as the late but this was discounted as it was deemed to be a very costly operation. Another tactic used was to generate 5025 samples of the prompt code and then shift them appropriately and store them into different array. This technique also proved to be costlier
than the current technique discussed as it required 5000 *2 array storing operation as well as 5000 * 2 array read operations. Also appropriate memory is required for the same which means that the code size increases correspondingly. Keeping in view the following it was decided to go ahead with the current implementation which in fact turned out to be faster than the other techniques discussed inspite of the fact that multiple IF’S disrupt[22] the flow of execution.

Once the early and the late A codes are generated the next step is to compute the Inphase and quadrature phase components. The term $I^2 + Q^2$ for the early, late and the prompt code are generated. The need to generate $I^2 + Q^2$ for the early and late is largely due to the fact that a determination is to be made whether the incoming code is lined with the locally generated code and depending on the energy levels present the code is either delayed or speeded up. The early late discriminator function calculation is done once the code phase rate is determined using the acquisition process

The complete process as described above is used to determine the appropriate energy level so that the code can be appropriately slowed down or speeded up. The difference in energy is given by

$$E_D = (I_e^2 + Q_e^2) - (I_l^2 + Q_l^2)$$

Where $E_D$ the discriminator function

$I_e$ the energy in the inphase component of the early signal

$Q_e$ the energy in the quadphase component of the early signal

$I_l$ the energy in the inphase component of the late signal
Q1 the energy in the quadphase component of the late signal

This difference of energy is used to control the speeding up or slowing up of the CA code.

The block diagram of figure 4.5 shows the general structure of the tracking loop. The loop is programmatically implemented by having a feedback structure in which the old values of the error signal are updated and fed back. The filter coefficients for the code and carrier tracking loops are defined. The signal is filtered using the coefficients defined earlier. The older values are updated by the newer versions. At this point another thing to be mentioned is the code rate of the signal. As was stated earlier the code rate changes because of which the total number of sample over which acquisition is performed also changes. With the early signal the total number of samples will be less as compared to the case when the late signal is being tracked. This is depicted in the following relationship

\[
\text{No of samples} = \text{code base rate} \times \text{sampling frequency}
\]

This is accommodated in the program dynamically and is updated by using the signal coming out of the loop filter, the value of which may be either positive or negative depending on whether the signal is early or late. Now this value is used to determine the code rate. The new code rate is then determined which in turn helps determine the number of samples over which acquisition is to be performed. The whole process is repeated until a lock is detected and the incoming signal is lined up with the locally generated version of the PRN code.
6.3.2 Costas Carrier Tracking Loop

The code and carrier tracking processes are coupled with each other. The filter coefficients for the tracking loop have been discussed in detail in chapter 4. They are defined here based on the definitions provided earlier. The carrier tracking loop is used to keep track of the phase or frequency of the incoming signal. The block diagram of the carrier tracking loop has been defined in figure 4.4. As shown in the block diagram the discrepancy in phase is computed and is applied as voltage to a numerically controlled oscillator. Based on the filtered error phase signal the oscillator adjusts the frequency of its reference signal in an attempt to match the phase of the incoming signal.

The phase discrepancy is given by the following

\[ P.D = \text{ATAN}(I_p + I_q) \]

Where PD denotes the phase difference or the error signal in this case

- \( I_p \) denotes the inphase component of the prompt signal
- \( I_q \) denotes the quadphase component of the prompt signal

The phase difference is used to control the phase of the locally generated signal. A feedback structure similar to the one used in code tracking is implemented wherein the older values are updated by the newer values. The \textit{pvalue} term is updated to accommodate for the appropriate change and the corresponding value is selected.

Figure 6.9 shows the plot of the early, late and prompt codes for satellite vehicle number 17 at code phase 469. The total number of samples for which tracking is attempted is 2000 which is the equivalent of a 2 second window of data. The values of
the early, late and prompt codes are written out to the corresponding file names; the data is imported to the Matlab programming environment where it is plotted and the results analyzed. The data is written using the printf statement and is written in ascii format. Binary writing of the data is comparatively faster than writing in ascii and the difference is a 10% speedup but due to the fact that a binary write requires all the data to be stored in the form of arrays it was decided to go ahead with the ascii write as compared to the binary write[24]. Binary write requires all variables to be stored in the form of arrays and it has the capability to write chunks of data at a time. Now for the above case binary write requires 6 arrays of 5000 elements each. The clock cycles involved with reading and writing 6 5000 element arrays is considerable high. The memory requirements also increase correspondingly. One of the goals was to make an attempt to keep the code size to a minimal length which can be stored on the RAM. Keeping these considerations in mind it was decided to go ahead with the ascii write as the advantages incurred by using the binary writes negates by the heavy clock cycles used in processing array data.

The acquisition parameters used to initialize the tracking loops need to be accurate. In case the parameters are not accurate then there is a slight pull in process during which the tracking loops will acquire and lock onto the signal. The tracking loops do lock onto the signal and correctly demodulates the data bits but there is some transient as a result of the erroneous frequency initialization. It is very essential that the acquisition process correctly determine the code phase of the signal as well as the carrier frequency. Once the tracking phase has locked onto the signal and important signal parameters identified the next step is the demodulation of the data bit.
Figure 6.9 Plot Of Carrier/Code Tracking For SV 17
6.4 Bit Synchronization

The inphase arm of the Costas loop has the Navigation bits present. Upon initial reception of the C/A code acquisition the C/A code epoch ambiguity results in lack of knowledge of the databit timing. Thus, bit synchronization is required. There are a number of techniques available to achieve bit synchronization. The technique used here is similar to the histogram approach. The approach breaks an assumed data bit period(20ms) into 20 CA code epoch periods(1ms) and senses the sign changes between successive epochs. For a sensed sign change a corresponding cell count is incremented until a count in one specific cell exceeds the other by 19 bins by a pre specified amount. The procedure for the same can be defined theoretically as follows

1) A counter is initialized and runs from 0 to 19

2) Each sensed sign change is recorded

3) The process is continued until one of the following occurs
   a) Two cell counts exceed a predefined threshold
   b) Loss of lock
   c) One cell count exceeds a predefined threshold

4) If a) occurs, bit synchronization fails because of low C/N_o or lack of bit synchronization. If b) occurs, lock is reestablished. If c occurs, bit synchronization is successful and the CA code epoch count is set to the correct value.

Figure 6.10 shows the data present in the inphase arm of the Costas loop. The data is that of satellite vehicle number 17. The plot is subdivided into two parts, the first of
which is the navigation data over 5000 samples. To get a better understanding of the bit synchronization process a zoomed view of the same plot is shown in the second part. This plot is of the first 1000 elements and as is clear from the graph the data bits are not exactly lined up. There is sometimes a lag due to the pull in process because of which there might be some transient present which could be erroneous.

Bit synchronization[3] is begun by first finding the largest transition amplitude between two consecutive samples. This transition is assumed to be an edge in the input data vector. An important point to be mentioned here is that the first 200 samples are neglected as they can be unreliable. This unreliability stems from the pull in process before the signal can be locked and the pull in process can be minimized by using the minimum tracking loop bandwidths for the dynamics expected. Once the largest transition is detected the bits before the transition need to be identified. Now at this point a situation might occur in which the bits can be before the transition or they could be after the transition. The bit identification process proceed by assuming the bits are 19 or 20 samples in duration. The 19 samples are integrated before the transition occurs. Now if the integrated value is greater than 0 then the samples are assigned to be 1. The 20th sample is assigned a 1 if it is greater than 0 otherwise it is considered to be the first sample of the next bit. Now each time a bit is identified the value of the variable $tran$; which is the largest transition is decremented by 19 or 20 samples. The process for detecting the bits which occur after the sample number where the largest transition occurs is very much similar to the one discussed above.
Figure 6.10 Plot Of Navigation Data For SV 17
Figure 6.11 Plot Of The Synchronized Data For SV 17
Figure 6.12 Comparison B/W Synchronized & UnSynchronized Data For SV 17
The process is implemented repeatedly over 400 samples ahead in order to identify the largest transition while the identification process is just the same once the largest transition is identified.

Figure 6.11 shows the output which is a vector whose amplitude transition are synchronized with the local data. The plots are shown for satellite vehicle 17 and in a manner similar to the previous figure the transitions are plotted for 5000 samples and 1000 samples respectively. In order to reiterate the results obtained and to verify the correctness of the program figure 6.12 is plotted. The figure is a comparison of the synchronized and the unsynchronized data. As is shown in the figure the transitions are accurately detected which thereby leads to the conclusion that the bit synchronization algorithm works as expected.

6.5 Frame Synchronization & Parity Decoding

The demodulated navigation data combined with the code tracking loop measurements provide the necessary components to compute a position solution. The decoded data bits must be searched for a possible preamble and if successful a parity check is performed and the words in the data are decoded. The 8 bit preamble of 10001011 is at the beginning of each 6-s subframe. The 8 bit preamble could also be inverted and be present as 01110100 because of sign ambiguity. For the data obtained from the bit synchronization routine the preamble to be searched is 1 -1 -1 1 -1 1 1 1 which due to sign ambiguity could be inverted as -1 1 1 1 -1 1 -1 -1. The procedure for correctly finding the preamble is as follows
1) Search for the upright or inverted preamble which in this case could be 1 -1 -1 -1 1 1 1 1 1 or -1 1 1 1 -1 1 -1 -1

2) When one of the two patterns is found it is checked to see if it is the beginning of a 30 bit word. The next 22 bits are stored and a parity check is performed on them. If the parity check fails then the pattern is discarded and a new search for the pattern is initiated.

3) If parity check passes then it proves that the preamble existed at the beginning of a 30 bit word. In such a situation additional searches must be performed and if it is the correct TLM word then the following word must be a handover word that contains a truncated Z-count.

4) Parity should pass the HOW word. If not the frame synchronization process should be started

5) If the HOW word seems legitimate then the demodulation of the other data bits must be begun and the data is stored in memory.

The data obtained from decoding the frames combined with the information available from the tracking loops provide the necessary components required to compute a position solution. Parity decoding cannot be attempted before data frame synchronization because of unknown word boundaries where the timing uncertainties are large. For parity decoding it is sufficient to assume that the last 6 bits of the 30 bit word are the parity bits for decoding purposes. Parity decoding is essentially a exclusive-or operation

\[ D = H \oplus d \]
Where $D$ is the 6 bit parity vector and $H$ is the $6 \times 26$ mask matrix with 1's in the elements corresponding to the exclusive-or's of the 6 parity equations of the GPS signal specifications [3]. This operation is performed one column at a time for each incoming bit, the parity vector equals the last 6 bits of the 30 bit word and if that is the case then parity passes and the last 24 bits of $d$ are the decoded bits with the sign ambiguity resolved.

Once the data bits are decoded the information available from the frames coupled with the code tracking loop measurements provide the necessary components to compute a position solution. The algorithms stated above are proven algorithms and are expected to perform accurately. The idea behind implementing the above was to achieve real time performance. The resulting code was desired to be optimized to an extent that processing the GNSS signal in real time mode becomes a reality. The algorithms were coded on a Watcom compiler and tested on different platforms. Initially the tests were conducted on a 166MHz Pentium processor and as significant improvement in timing was achieved, the test bed was shifted to a 266MHz processor. First acquisition was performed and was followed up by tracking the signal at the specified code phase. The timing measurement for the acquisition process on a 200MHz Pentium processor was of the order of 1.8 seconds. The time taken to track one second of data on a 266MHz processor was around 1.4 seconds while with the incoming data being stored on the cache a timing of 1.2 seconds was achieved. We have also tested it on a dedicated 300MHz processor and a timing of 0.94 seconds was obtained. With the availability of the 366MHz and 400MHz Pentium processor the ability to process the GNSSS signals in real time should not be in
doubt and hence we can say that this code significantly reduces the amount of time needed for computation.

Though the computational power is available the project is still a long way from achieving real time performance. The novel front end design places certain stringent restrictions on the ADC board[16]. The difficulties associated with the data acquisition board have been dealt with in detail in chapter 5 as well as the requirements which the data acquisition board is expected to satisfy. With the ongoing research in the Avionics Engineering Center labs the dream of an ideal data acquisition board could become a reality soon and with it comes the ability to process any kind of GNSS signal in real time mode using the software radio.
7. Summary And Conclusions

This project has focussed on a real time implementation of the GNSS software radio. The GNSS concepts as well as the software radio concepts were discussed in detail underlying the benefits of each. The advantages and disadvantages related to the traditional approach as well as those related to the software radio design philosophy were also presented. GNSS is an ideal choice for the software radio implementation as a result of complex signal processing and for the increased use on transmission as a navigation aid.

The GNSS software radio design was discussed. The two main parts of the software radio are the front end design and the associated signal processing. The signal processing with a view to achieving real time performance was the highlight of this research. In the software radio the ADC is placed as close to the antenna as possible thereby removing the intermediate stages of down conversion and reducing the components related to frequency down conversion. The placement of the ADC close to the antenna gives rise to a few difficulties pertaining to sampling rate but this was taken care of by aliasing the given signal. Bandpass sampling was used to provide frequency translation. The advantages and the constraints imposed due to Bandpass Sampling were discussed.

The second part of the software radio is the signal processing involved. The goal related to signal processing is to attempt to achieve real time performance. The signal processing involved is quite a complex task as the GNSS signals use CDMA spread spectrum modulation. Signal acquisition, PRN code and carrier tracking, data
demodulation and processing are all involved. A variety of test beds and platforms were evaluated with the view to attaining real time performance. Various methodologies, optimization strategies and implementation techniques were considered and implemented. An effort was made to achieve the desired level of performance without making the entire project platform specific. The rules of software portability, robustness and adaptability were kept in mind while programming the software. A variety of algorithms were analyzed, coded and tested with a view to attaining the most optimum performance with regards to computational speed and processing power.

Another important part of signal processing was the data acquisition board. The intrinsics related to the data acquisition board and the underlying sampling rate have been discussed threadbare. The data acquisition board is very important as it is used to transfer the results of high rate sampling onto the host PC's memory for processing. The concepts of data transfer, PCI bus mastering and sampling rates were analyzed and investigated. The deficiencies related to the data acquisition board and the constraints imposed on the board due to bandpass sampling are also discussed.

Finally a 30 second window of GPS-SPS data was used to verify and validate the algorithms and to get a estimate on the various performance parameters needed for real time implementation. A basis for a real time software radio has been established. With future research being concentrated on the development of a data acquisition board the goal of processing multiple GNSS signals in real time seems certain.
REFERENCES:


[18] Correspondence With Snyder, Chris, Spring 1998.


APPENDIX A: C/A Code Generation

/*Code to Generate the COARSE/ACQUISITION Code For Various Satellites And Store Them In Appropriate Files. This program has been implemented in the form of a function call and the appropriate files written in binary format*/

#include<stdio.h>
#include<math.h>
#define coderate 1.023e6

void cacode(short int *ca, int svnum, int fs, int numsamp)
{
    int temp, caindex, ca1;
    int cainseq, indb, inda;
    int caholder[10000];
    int j, i, g2[1023], g1[1023], save1, save2;
    int reg[10] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1};
    int reg2[10] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1};
    int g2tmp[1023], ss_ca[1023], k, g2shift;

    /*
    C
    C The Gold Codes Specified By The Department Of Defence
    C
    */

    int g2s[33] = {0, 5, 6, 7, 8, 17, 18, 139, 140, 141, 251, 252, 254, 255, 256, 257, 258, 469, 470, 471, 472, 473, 474, 509, 512, 513, 514, 515, 516, 859, 860, 861, 862};

    g2shift = g2s[svnum];

    /*
    C
    C Generation Of The G1 Code With The Specified Tap Positions
    C
    */
    for(i = 0; i <= 1022; i++)
    {
        g1[i] = reg[9];
save1 = reg[2]*reg[9];

for(j = 9; j >= 1; j--)
    { reg[j] = reg[j-1]; }  
    reg[0] = save1;

/*
C
C Generation Of The G2 Code With The Specified Tap Positions
C
*/
g2[i] = reg2[9];
for(j = 9; j >= 1; j--)
    { reg2[j] = reg2[j - 1]; }  
    reg2[0] = save2;

/*
C
C Shifting The G2 Code Appropriately Based On The Gold Code Values Specified By
C The DOD For Specific Satellites
C
*/
i =0;
    for(j = 1022 - g2shift + 1; j <= 1022; j++)
        { g2tmp[i] = g2[j];  
          i = i + 1; }  
k = i;
    for(j = 0; j <= 1022 - g2shift; j++)
        { g2tmp[k] = g2[j];  
          k = k + 1; }  
    for (i = 0; i <= 1022; i++)
        { g2[i] = g2tmp[i];  
          }
/* 
*C Forming The Simple Sample C/A Code 
*C */

for(i = 0; i <= 1022; i++)
{
    ss_ca[i] = g1[i] * g2[i];
}

/*/ 
*C Adjusting For The Desired Sampling Frequency 
*C */
cainseq = ceil( numsamp / ( ( fs / coderate) * 1023));
indb =0;
for (inda = 0; inda < cainseq; inda++)
{
    for(i = 0; i <= 1022; i++)
    {
        caholder[indb] = ss_ca[i];
        indb = indb + 1;
    }
}
j = 0;

for (i = 1; i < numsamp; i++)
{
    caindex = ceil(coderate * i / fs);
temp = caindex - 1;
cal = caholder[temp];
ca[j] =ca1;
j = j+1;
}

return;
} /*End Of The Program*/
APPENDIX B: Code To Perform Acquisition And Tracking

#include<math.h>
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<sys/timeb.h>

#define SEEK_SET 0

/*
C C Defining System Parameters Like The Code Base Rate, Satellite Vehicle To Be C Tracked, The frequency offset value.
C */

#define svnum 17
#define codebaserate 1.023e6
#define numchips 1023
#define freq 1.25025e6
#define initval 468

/*
C C Defining The Sampling Frequency And The Number Of Code Periods To Evaluate C */

#define fs 5.0e6
#define codeperiods 10000

/*
C C Defining The Code Tracking Parameters C */

#define k_dll 50.0
#define coeff1dll 5.332793746814834e-5
#define coeff2dll -5.32569139501465e-5
#define coeff3dll -1.0
`/*
C
C Defining The Carrier Tracking Parameters
C
*/
#define k_tll 1.256637061435917e3
#define coefn1tll 5.301831553599558e-5
#define coefn2tll -5.130771400150319e-5
#define coefd2tll -1.0

`/*
C
C Specifying The Raw Data File For The Input Read Operation
C Defining The Specific C/A Code File To Be Read
C Defining The Output Files To Be Written
C
*/
#define dname "srvy1sfl.dat"
#define name "casat"
#define ext "dat"
#define sinfile "sinint.dat"
#define cosfile "cosint.dat"
#define earlywritei "ertiary.bin"
#define earlywriteq "erlyvalq.bin"
#define latewritei "lative.bin"
#define latewriteq "latevalq.bin"
#define prmpwritei "pmptvali.bin"
#define prmpwriteq "pmptvalq.bin"

`/*
C
C The Beginning Of The Main Program
C
*/
main()
{
char index,filename[100];
struct timeb start,finish;
FILE *datptr,*readptr,*earlyptr,i*earlyptrq,*lateptr,*lateptrq;
FILE *prmpptr,i*prmpptrq;
int *sin5array,*cos5array,*catable;
unsigned int pvalue4,index4,pincr4,pctl;
unsigned int early_late,tpbit,pvalue,i,l,pstep;
int promptca,earlyca,lateca,tempin,tempqq,ns,npad,nchipminus1;`
int asumearlyi, asumlatei, asumpmrpti, asumearlyq, asumlateq, asumpmrptq;
float diffintime, msecs;
double coderate, filterdll, filterdllold, errordll, errordllold;
double twopi, filtertll, filtertllold, errortll, errortllold, tlltemp, numchipsdbl;
signed char *rawdat;

/*
 C
 C Allocating Memory For The Various Array Elements
 C */
sin5array = calloc((int)(256), sizeof(int));
cos5array = calloc((int)(256), sizeof(int));
catable = calloc((int)(1023), sizeof(int)); /*specify largest possible*/

/*
 C
 C Reading In The Specific C/A Code Data File Say CASAT*.DAT Where * is the
 C Satellite Vehicle Number, Opening The Files In Which The SINE() & COSINE()
 C Values Have Been Stored.
 C */
sprintf(filename, "%s%02d.%s", name, svnum, ext);
readptr = fopen(filename, "rb");
fread(catable, sizeof(int), numchips, readptr);
fclose(readptr);
readptr = fopen(sinfile, "rb");
fread(sin5array, sizeof(int), 256, readptr);
fclose(readptr);
readptr = fopen(cosfile, "rb");
fread(cos5array, sizeof(int), 256, readptr);
fclose(readptr);
datptr = fopen(dname, "rb"); /*open raw data file*/

/*
 C
 C Specifying The Output Files To Write The Data To
 C */
earlyptri = fopen(earlywritei, "w");
earlyptrq = fopen(earlywriteq, "w");
lateptri = fopen(latewritei, "w");
lateptrq = fopen(latewriteq, "w");
prmptptri = fopen(prmptwritei, "w");
twopi = 2.0 * acos(-1.0);  
nchipminus1 = numchips - 1;  
numchipsdbl = ((double) (numchips));

/*
 C
 C Specifying And Initializing The Variables Needed For The Lookup Approach
 C
 */
pvalue = 0;
index4 = 0;
pvalue4 = 0;

/*
 C
 C Defining The Feedback Parameters For The Tracking Loops
 C
 */
errordllold = 0.0;
filterdllold = 0.0;
errortllold = 0.0;
filtertllold = 0.0;

/*
 C
 C Defining The Values Used For Computing The sine(), cosine(), early/late values
 C
 */
pstep = (unsigned int) (((freq/fs)*pow(2,32))+0.5);
pincr4 = (unsigned int) (((codebaserate/fs)*pow(2,31))+0.5);

/*
 C
 C Defining The Space Required Dynamically Based On The Number Of Chips
 C
 */
ns = (int) (ceil((numchipsdbl / (((double)(pincr4))/pow(2,31)))));
nspad = ns + (int) (((double) (ns)) * 0.25)); /* make array 25% bigger */

/*
 C
 C Adjusting The Code Rate After The Tracking Loops
 C
*/
C Defining Parameters For The Generation Of Early/Late Code From The Prompt Code
C
*/
early_late = pincr4*2;       /*two sample early/late spacing*/

/*
C Specifying The Correct Value To Read In Based On The Code Phase Rate
C*/
rawdat = calloc(nspad,sizeof(signed char));
fseek(datptr,initval*sizeof(signed char),SEEK_SET);
/*
C Timing Information
C*/
time(&start);
/*
C Beginging The Actual Loop Based On The Code Periods And Accounting For The
C Change In Doppler.
C*/
for(l=0;l<codeperiods;l++)
{
    pincr4= (unsigned int) (((coderate/fs)*pow(2,31))+0.5);
    ns = (int) (ceil((numchipsdbl / (((double)(pincr4)) /pow(2,31))));
    fread(rawdat,sizeof(char),ns,datptr);

    /*
C Setting Up Variables For The Accumulate And Dump Filter
C*/
    asumearlyi =0;
asumlatei = 0;
asumprmpti = 0;
asumearlyq =0;
asumlateq = 0;
asumprmptq = 0;
I
C Running The Loop For The Total Number Of Samples
C
*/
   for(i=0;i<ns;i++)
   {
       index=((pvalue & 4278190079)>>24);
       tpbit = ((pvalue4 & 2147483648)>>31);
   }

I
C Implementing a 1023 bit Circular Buffer For Computing The Prompt Code
C
*/
   if(tpbit > 0)
   {
       pvalue4 = pvalue4 + 2147483648;
       index4 = index4 + 1;
       if(index4 > nchipminus1)
       {
           index4 = 0;
       }
   }
   promptca = catable[index4];

I
C Using The Prompt Code To Generate The late Code
C
*/
   if((pvalue4+early_late) >= 2147483648) /*late code*/
   {
       if(index4>=nchipminus1)
       {
           lateca = catable[0];
       }
       else
       {
           lateca = catable[index4+1];
       }
   }
   else
lateca = catable[index4];

if(early_late <= pvalue4)  /*early code*/  
{
    earlyca = catable[index4];
}
else  
{
    if(index4 == 0)
    {
        earlyca = catable[nchipminus1];
    }
    else  
    {
        earlyca = catable[index4-1];
    }
}

tempin = sin5array[index] * rawdat[i];
tempqq = cos5array[index] * rawdat[i];

asumearlyi += earlyca * tempin;
asumearlyq += earlyca * tempqq;
asumprmpiti += promptca * tempin;
asumprmpptq += promptca * tempqq;
asumlatei += lateca * tempin;
asumlateq += lateca * tempqq;
pvalue = pvalue + pstep;
pvalue4 = pvalue4 + pincr4;
}           /*end of the ns loop*/

/*
C
C Code Tracking
C
*/
errordll=(pow(((double)(asumlatei))/635.0),2.0) + /* 127*5=635.0 */
      pow(((double)(asumlateq))/635.0),2.0)) -
      (pow(((double)(asumearlyi))/635.0),2.0) +
      pow(((double)(asumearlyq))/635.0),2.0));
filterdll=coefn1dll * errordll + coefn2dll * errordllold -
             coefd2dll * filterdllold;
errordllold=errordll;
filterdllold=filterdll;

/*
C
C Calculating The Change In Code Base Rate Based On The Energy Levels In The
C Early/Late Signals
C
*/
coderate=codebaserate + k_dll * filterdll;

/*
C
C Carrier Tracking
C
*/
errortll= atan(((double)(asumprmpti))/((double)(asumprmptq)));  
      filtertll=coefn1tll * errortll + coefn2tll * errortllold -
             coefd2tll * filtertllold;
      tlltemp=(k_tll * filtertll);
      ptll = (int) (tlltemp * pow(2,32) + 0.5);

/*
C
C Adjusting The Frequency Offset Value Based On The Tracking Loop
C
*/
pvalue = pvalue + ptll;

/*
 C
 C The Feedback Loop
 C *

   errorllold=errorll;
   filterllold=filterll;

/*
 C
 C Writing Out The Values
 C *

 fprintf(earlyptri,"%d\n",asumearlyi);
 fprintf(earlyptrq,"%d\n",asumearlyq);
 fprintf(lateptri, "%d\n",asumlatei);
 fprintf(lateptrq, "%d\n",asumlateq);
 fprintf(prmptptri,"%d\n",asumprmpti);
 fprintf(prmptptrq,"%d\n",asumprmptq);
 }

    /*end of the loop*/

/*
 C
 C Performing Timing Measurements Down To The Nearest Millimeter
 C *

 ftime(&finish);
 diffintime=(difftime(finish.time,start.time));
 msecs=((float)finish.millitm)-((float)start.millitm);
 diffintime=diffintime+msecs/1000;
 printf("the time taken is %f\n",diffintime);
 fclose(earlyptri);
 fclose(earlyptrq);
 fclose(lateptri);
 fclose(lateptrq);
 fclose(prmptptri);
 fclose(prmptptrq);
 fclose(datptr);
/*
C
C Freeing Up The Memory Space
C
*/

free(rawdat);
free(sin5array);
free(cos5array);
free(catable);
}

/******/
C
C End Of The Program
C
*/