11-1-2004

Behavioral VHDL Implementation of Coherent Digital GPS Signal Receiver

Viswanath Daita
University of South Florida

Behavioral VHDL Implementation of Coherent Digital GPS Signal Receiver

by

Viswanath Daita

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering
Department of Electrical Engineering
College of Engineering
University of South Florida

Co-Major Professor: Srinivas Katkoori, Ph.D.
Co-Major Professor: Moreno Wilfrido, Ph.D.
Sanjukta Bhanja, Ph.D.

Date of Approval:
November 01, 2004

Keywords: Time domain, C/A code, Sliding correlator, Acquistion, Tracking

DEDICATION

To my family

ACKNOWLEDGEMENTS

I would like to take this opportunity to express thanks and gratitude to Dr. Katkoori for his support, encouragement and patience during the course of this research. I would also like to thank Dr. Moreno for his support and patience and Dr. Bhanja for being on my committee. I would also like to thank all the faculy who guided me during my Masters.

I would like to acknowledge all the help and support provided by family who stood by me at all times and have been a source of encouragement . I am grateful to my father - D.Satyanarayana, sister - Vijaya Gouri, brother-in-law - T.G.K. Murthy and my brother - Raghunath, who always believed in me.

Finally, I would acknowledge all the help and support given by members of VCAPP group, especially Viswanath, Narender, Ranganath, and Sriram. I would like to thank all my friends and roomates at USF for their constant support and encouragement. I would also like to acknowledge the help provided by the Computer Science Tech support team led by Daniel Prieto.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

AGC	Automatic Gain Control
ASIC	Application Specific Integrated Circuit
BPSK	Binary Phase Shift Keying
C/A	Coarse/Acquisition
CMOS	Complementary Metal Oxide Semiconductor Field Effect Transistor
DGPS	Differential Global Positioning System
DLL	Delay Locked Loop
DSP	Digital Signal Processor
DSSS	Direct Sequence Spread Spectrum
FFT	Fast Fourier Transform
FHSS	Frequency Hopped Spread Spectrum
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System for GPS and GLONASS
IF	Intermediate Frequency of order 100s of KHz - few MHz
L1	Civilian GPS signal of frequency 1.575 GHz
L2	Civilian GPS signal of frequency 1.227 GHz
MEMS	Micro-Electro-Mechanical Systems
NMEA	National Marine Electronics Association
P	Precision Code
PLL	Phase Locked Loop
PRN	Pseudo Random Noise Code
RF	Radio Frequency of order 100s of MHz- few GHz
RHCP	Right Hand Circularly Polarized

RINEX	Receiver Independent Exchange
SoC	System-on-a-Chip
SS	Spread Spectrum
VCO	Voltage Controlled Oscillator
VHDL	VHSIC Hardware Description Language
WAAS	Wide Area Augmentation System

BEHAVIORAL VHDL IMPLEMENTATION OF COHERENT DIGITAL GPS SIGNAL RECEIVER

Viswanath Daita

ABSTRACT

Global Positioning System is a technology which is gaining acceptance. Originally developed for military purposes, it is being used in civilian applications such as navigation, emergency services, etc. A system-on-a-chip application merges different functions and applications on a single substrate. This project models a GPS receiver for a system on chip application. The GPS receiver, developed as a core, is intended to be a part of a navigation tour guide being developed. The scope of this work is the GPS C/A code on the L1 carrier. The digital signal processing back-end in a GPS receiver is modeled in this work. VHDL modeling of various communication sub-blocks, detection and demodulation schemes is done. A coherent demodulation of the GPS signal is implemented. GPS receiver calculates the position based on the data collected from four satellites. Given four satellites, acquisition of the data from the signals is performed and data demodulated from the same. Synthetic data is generated for validation purposes. Code acquisition and tracking of the GPS C/A signal is implemented. Cadence NC-Launch VHDL simulator is used to validate the behavioral VHDL model.

CHAPTER 1
INTRODUCTION

Global positioning system is an advanced navigation and positioning system used today for various applications. These vary from GPS guided missiles for precision bombing in the military, to peace time and civilian uses, such as navigation, treasure hunt hobbies, and agriculture.

Exploration has been important to mankind which has resulted in discovery of continents and new worlds. In ancient times, navigation was based on the planetary and stellar positions. This changed to the use of magnetic compass in the medieval times up to very recent in the past. Such alternatives always had their disadvantages and misgivings when dealing with hostile weather conditions, for example, foggy conditions, reduced visibility, thereby hindering navigation. With the development of satellites and improvements in radio signal transmission and reception, these were used for the navigation purposes as well as for positioning. The advantage of using radio signals is that they are immune to the weather effects. Earlier systems included LORAN (*Lo*ng-*Ra*nge *N*avigation), OMEGA to guide aircraft and ships [1]. LORAN was restricted to the United States and Britain. OMEGA was a truly globally available positioning system. The use of satellites in positioning and navigation was first applied in TRANSIT (Navy Navigation Satellite System), a project developed at the Applied Physics Laboratory at Johns Hopkins University. The Doppler frequency shifts of the signals transmitted by satellites were used to determine the satellite orbit. The receiver on the earth could determine its position from the knowledge of the satellite orbit and the Doppler shift measurement of the frequency. Global Positioning System, in short GPS, is a product of the United States Department of Defense. Intended for military purposes, especially precise positioning for ammunition, it has been demonstrated that it could be used for civilian purposes as well. Its utilization has been demonstrated correctly during the two Gulf wars where precision guided missiles have found target with a high probability destroying enemy positions. The target's co-ordinates are loaded in the computer of the missile which is guided by the satellites. Apart from these military advantages civilian applications too such as navigation and surveying have found use for GPS. *Connected Car* [2] is a more recent example of how GPS can be used as a navigation aid in co-ordination with other applications and frameworks such as Microsoft .Net Framework, Bluetooth etc. It can be used as a guide in new places. GPS has been used also to land an airplane in adverse weather conditions. A GPS measurement can have an error of 5-10 m (uncorrected) or up to 1m discrepancy (using WAAS and DGPS). Agriculture also has found use of the GPS - to control the distribution of the chemicals and fertilizers. In conjuction with Geographic Information System (G.I.S), GPS has found more use in tracking animals, humans, and knowing the seismology of the earth at a given place. Further advances in GPS signal reception could lead to indoor coverage, in downtown areas, and under trees etc., where the reception is low. This is what helps GPS to be a part of the emergency services. In this class of applications, another popular one is pervasive computing-location awareness. GPS presents a solution to this end in mobile communication electronics. A lot of research goes into how to make the GPS signal more reliable, visible, and accurate. This system of navigation uses omnipresent radio waves and relative time of arrival of signals to determine positions. The two common frequencies used today by GPS satellites to broadcast are L1 (1575.42 MHz) and L2 (1227 MHz). L1 is primarily a civilian signal while L2 is used for military purposes. L1 is also used by the military and L2 by civilians though the civilians donot have a knowledge of the codes modulating the L2 frequency. From 2005 onwards, GPS satellites will be broadcasting new signals which could help eliminate positioning errors due to Ionospheric effects. The current civilian signals will be boosted by the addition of another civilian signal on L2. From 2008, a new frequency band called L5 will be emitted at 1176.45 MHz which is also a civilian signal. L3 and L4 will carry non-navigation information for the military [3].

1.1 GPS Integration and Issues

GPS receivers incorporate Direct Sequence Spread Spectrum (DSSS) Techniques in their analysis. Traditionally, GPS receiver has been a chip set, consisting of two or more chips. With the advances in Integrated Circuit technology there is a trend towards a single chip solution, which is advantageous in many ways. Such a chip will help integration of a variety of applications from cell phones to wrist watches. It involves a high level of design integration. A single chip GPS solution developed by Valence Semiconductors [4], Motorola [5] and Sony recently has shown that such a solution is possible. The single chip solution looks at integrating the RF/IF block with the digital signal processing block on a single chip. Sony's RF CMOS technology [6] was used to develop such a single chip solution. It was shown that a separate acquisition and tracking blocks could improve the performance of the GPS chip as it gave the independence to develop better algorithms. With the new algorithms it was shown that there was a drastic reduction in the acquisition times for a hot start.

Effects of integration and configuration possibilities of single chip GPS implementation are next described [7]. Chip scale integration results in foremost reduction in size, improves reliability, reduces cost, and reduces power consumption. Typical two-chip solutions, one for the RF/IF section and the other for the DSP/ASIC account for more power consumption as there is amplification needed for signals that exit a chip. The single chip solution removes such a need as these signals become internal to the chip. A single chip GPS also results in higher performance by allowing higher speeds. Integration towards a single chip involves integrating different likely components such as LNA, RF section, digital processing, microprocessor, peripherals, and I/O drivers. For forming a single GPS chip to be applied in various environments an ideal integration would be the LNA, RF section and digital processing into one chip such that the peripherals and microprocessor can be shared with the application. The major issues to be addressed in a single chip solution are the cross-talk between the digital signals and weak analog signals.

1.2 Receiver Processing History

Digital Signal Processing techniques have been used to process the GPS signals. Such digital receivers perform code correlation, which is the main signal processing requirement, on digital data rather than analog data. Digital signal processing of the digital data is performed in a DSP chip or an ASIC. A software based design approach of a GNSS receiver has been demonstrated in [8]. This develops a technique to sample the incoming signal very near to the antenna and at RF range. Such placement removes the need for analog signal processing components. Research into software defined radios based digital/software GPS receiver provides algorithmic specification of the process. Any changes required would lead to change in code. The GPS receiver consists of many components such as PLLs, DLLs, etc., for processing the signal. A digital implementation of the same has resulted in greater fidelity. Software radio based GPS receivers try to digitize the signal very near to the antenna. Conventionally, Analog to Digital Converter is employed as in a Super-heterodyne receiver after few stages of down-conversion. Software receivers try to sample at GHz frequency range. At the same time, they process signals without any analog components which result in nonlinearities.

1.3 System-on-a-Chip Implementation (SoC)

System-on-a-Chip design is defined as a complex Integrated Circuit that integrates the major functional elements of a complete end-product into a single chip or chip set [9]. An SoC design includes the peripheral components and the motivation for such a design stems from advances in technological perspectives. Sometimes the system-on-a-chip design could incorporate the analog, MEMS components inputs. A system on chip uses intellectual property cores in its design which form the basic reusable blocks. These cores could be any complex function which is used in an application. Such core based design is termed Block-based design. Another design aspect is the platform based design approach which uses a family of Hardware-Software architectures to implement the required architecture.

In Location based computing where GPS could be used to determine the location, a SoC approach could be used to implement the design. In such situations a platform based approach is preferred.

The basic intention behind developing this digital GPS prototype is two pronged. First it is designed as part of a single chip receiver. This receiver is intended to be used to develop a System-on-a-Chip design, with this acting as a core within a mobile location computing tour guide developed by [10]. This work tries to develop the back end in behavioral VHDL. This work includes design and implementation of the back end/baseband Digital Signal Processing Techniques. Zhuang [11] was one of the original works which illustrated the advantages of designing a software receiver over hardware based receiver in monitoring effects of the system parameters. Braasch [12] deals with the different receiver architectures and the performances of these with respect to acquiring and tracking a signal. Both these processes are dependent on correlation. Computing the correlation has been achieved in several ways. It can be implemented in time domain or in frequency domain. Apart from this, the use of a DSP or using a software based approach to implement the same also determines how correlation is performed. Hardware based N-point sequences to correlate in a DSP was demonstrated by Van Nee [13]. Faster acquisition time could be obtained by using a software approach or by using efficient algorithms such as FFT to calculate the correlation , eg., Averaging correlator [14], Modified Averaging correlator [15], block processing techniques [16], [8]. An attempt towards implementing a block processing based GPS receiver was done in [15] using FPGA which uses parallel processing, thereby increasing the speed. Besides using an ASIC/FPGA which was the case above, one could also develop software defined GPS receivers (SGR) [17], [18] where the processing is implemented on a computer as shown in Figure 1.1. A hardware receiver block diagram and a different software receiver block diagram are as shown in the Figure 1.2 below [19]. Software receivers are flexible in operation and can deviate from conventional hardware approach. A user could take a snapshot of the data instead of continuously tracking it. Also new algorithms could be developed without changing the hardware, which is not possible in the hardware GPS

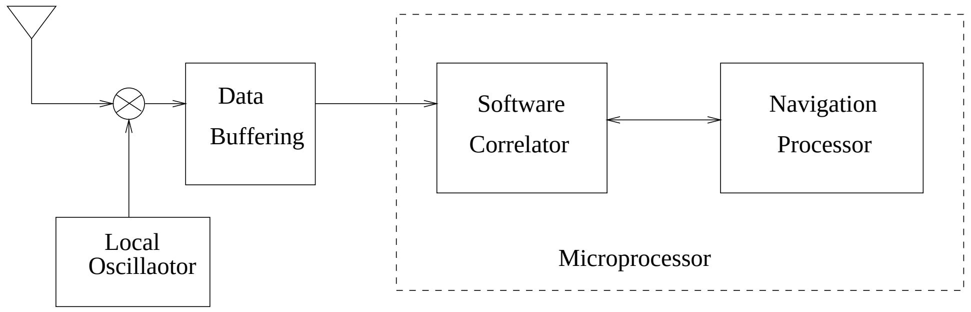


Figure 1.1. Software GPS Receiver

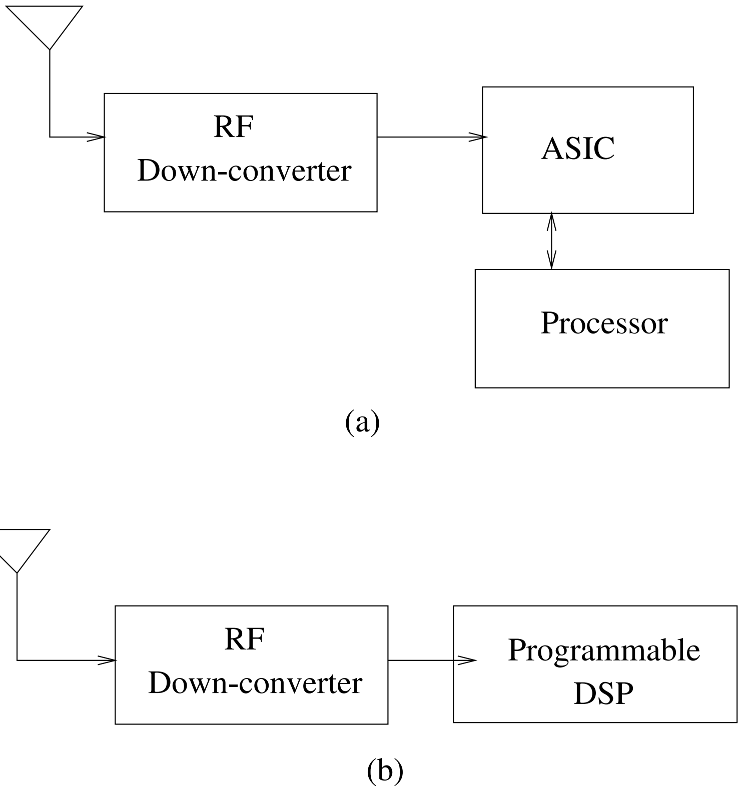


Figure 1.2. Receiver Block Diagrams: (a) Hardware (b) Software

receiver design [20]. One can write programs to process the signals in a way independent of the underlying hardware.

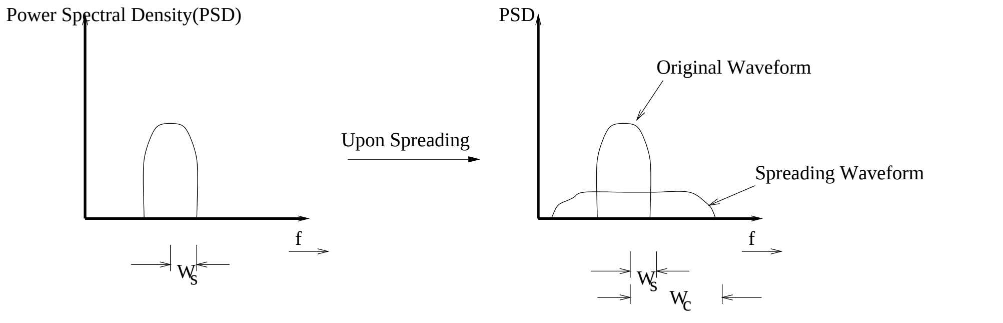
This work involves VHDL implementation of the code tracking and acquisition loops in the GPS receiver. Chapter 2 introduces Spread Spectrum Communications and GPS Signals. Chapter 3 describes the GPS receiver both hardware based and software-based, and describes the pros and cons of each. Chapter 4 presents an overview of the VHDL Implementation of time domain serial acquisition of signals. Chapter 5 presents the experimental results. In conclusion Chapter 6 summarizes the work and outlines future directions.

CHAPTER 2
SPREAD SPECTRUM SIGNALS

This chapter introduces the concept of the Spread Spectrum communications which is the basis of the GPS. Briefly described is the nature of spreading and the effects of spreading the baseband data. This chapter outlines the different modulation techniques used in SS communications and also the different spreading codes. The important concept of correlation is also dealt, with reference to synchronization. Also introduced are both the correlation methods serial and parallel, which help in code acquisition and tracking. An introduction to the primary problem of Spread spectrum communications, synchronization, is given and emphasis given to both code acquisition and code tracking. Demodulation could be either coherent or non-coherent and the differences between the two have also been outlined. Finally an introduction to GPS signals is given. Details of the different segments of GPS system, the GPS signal structure, generation of GPS C/A codes have been listed.

2.1 Introduction

Spread Spectrum signaling was first used in World War II [21] to communicate by shifting the control frequencies at a very fast rate. It is used in areas where essential communications can be jammed on intention [22]. Anti-jamming and low probability of intercept (LPI) are two important advantages of Spread Spectrum Communications in multi-user communication environments where each user could be assigned a unique code (Code Division Multiple Access). Spread spectrum communications is used when the bandwidth of the baseband modulating waveform (W_s) is spread to a wider bandwidth (W_c) (by a spreading sequence) as shown in the Figure 2.1. The Process Gain or Spread Spectrum Gain for the



Baseband BW W_s spreaded by a sequence of BW W_c. The Power Spectral Density (PSD) is plotted versus frequency.

Figure 2.1. Spreading Bandwidth using High Frequency Code

above system is given by:

$$G = 10 \log W_c/W_s \tag{2.1}$$

Spread spectrum is also useful when the data rate is low and the distances to be transmitted are long. For such transmission, we would otherwise need an antenna of very large diameter. This kind of transmission, however, limits the antenna size to normal standards. The usefulness of a spreading code is that it spreads the data over a large bandwidth [23]. A spread spectrum system is illustrated in Figure [24] below.

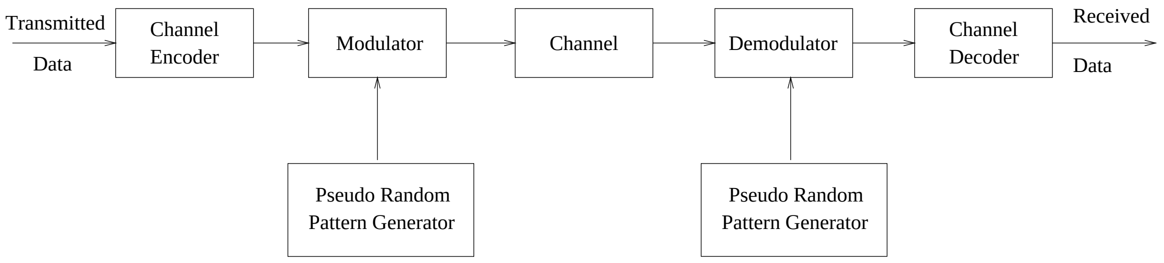


Figure 2.2. Spread Spectrum Model

As shown in the Figure 2.2, the information sequence is modulated with sequence generated by a pseudo-random pattern generator at the transmitter which is then removed at the receiver by using an exact replica. To get the information sequence at the output of the receiver one has to have a synchronous copy of the spreading sequence. PSK (phase

Table 2.1. Spreading Codes - A Comparision

Sequence	*Auto − Correlation*	*Cross − Correlation*
m-sequence	Ideal	Poor
Gold Codes	Poor	Ideal
Walsh Transforms	Poor	Zero(if synchronized)
Kasami Codes	Poor	Poor

shift keying) and FSK (frequency shift keying) are two common modulation techniques to implement spread spectrum. PSK uses a phase shift of $\pm\pi$ for a chip change. Such a modulated signal is termed Direct Sequence Spread Spectrum signal. If the code is used to modulate an M-ary FSK then it results in Frequency Hopping Spread Spectrum (FHSS). The carrier is modulated by the data. Two important processes have to take place at the receiver end to get the final data. At first, the carrier along with any Doppler effects has to be removed and then the modulating code has to be stripped off before one can start detecting the data at the receiver end. There are different kinds of codes that are used to spread the data in spread spectrum communications. The codes need to have certain properties to qualify as spreading codes. They should have good cross-correlation (ideally zero), ideal autocorrelation (should be zero if offset is greater than one chip) and also they have to be random in nature. As it is difficult to work with truly random codes, there are some codes which are periodic in nature, but random and at the same time satisfy the two criteria listed above known as *pseudo-random sequences*. Examples include Gold codes, maximal length sequences, Walsh codes, and Kasami codes. Table 2.1 illustrates the auto and cross-correlation properties of these codes[22].

2.1.1 Direct Sequence Spread Spectrum (DSSS)

A brief introduction is given here to Direct Sequence Spread Spectrum (DSSS) which is used in GPS. DSSS is more robust than FHSS (Frequency Hopped Spread Spectrum). Frequency and phase synchronization is easier in DSSS than the latter, because only one frequency acts as the carrier and doesn't change. The difficult part of synchronization in a DSSS Receiver is that it has to acquire both the chip time as well as the symbol timing while the FHSS Receiver has to acquire only the symbol time. An equivalent mathematical representation of BPSK modulation is a multiplication with +1 or -1 corresponding to a 0 or a 1 bit respectively. A BPSK DSSS signal can be represented as:

$$s(t) = Ac(t)d(t)sin(wt + \phi) \tag{2.2}$$

where

w: the carrier frequency at which the signal is being transmitted

c(t): modulating pseudo random code

d(t): data bits

Such a phase modulated signal is demodulated by correlating the received signal with a synchronized replica of the spreading signal. If

$$r(t) = Ad(t - T_d)c(t - T_d)cos(w_0(t - T_d)) \tag{2.3}$$

is the received signal then a replica of the code $c(t - \hat{T}_d)$ is used to synchronize. The correlator output in the Figure is:

$$r(t) = Ad(t - T_d)c(t - T_d)c(t - \hat{T}_d)cos(w_0(t - T_d)) \tag{2.4}$$

If $T_d = \hat{T}_d$ the received code is synchronized with the incoming code. The output of the receiver at this instant is the transmitted data with the delay T_d. Thus data signal is recovered in two steps: detection and demodulation as has been represented in Figure 2.3.

The above discussion assumes that the local and received signals are synchronized. In reality one has to synchronize them before detection and demodulation can start.

2.1.2 Synchronization

It is the process of matching the locally generated spreading signal with the incoming spread spectrum signal. Synchronization is a two step procedure:

1. Acquisition and
2. Tracking.

In acquisition phase the two signals are brought into coarse alignment with each other. Once the signal has been acquired, tracking phase starts. In this phase, the closest possible waveform is tracked and a fine alignment is maintained using a feedback loop.

2.1.2.1 Acquisition

In this phase the replica signal is brought within one code chip time of the incoming signal. The receiver continuously searches for the pseudo-random pattern generated by the transmitter during acquisition. Given the chip duration to be T_c, the initial delay to be T_d, and N, the number of code phases which are to be searched for acquisition, the initial time for acquiring a signal in time steps of $T_c/2$ is:

$$T_{acq} = 2NT_d \tag{2.5}$$

For correlating signals either signal matching can be done as in a *matched filter* or time averaged cross correlation as in a *correlator*. The *matched filter*, matched to a known pseudo random code, looks for whether a threshold has exceeded the preset value. On such an occurrence the data is said to have been acquired and the receiver starts demodulating

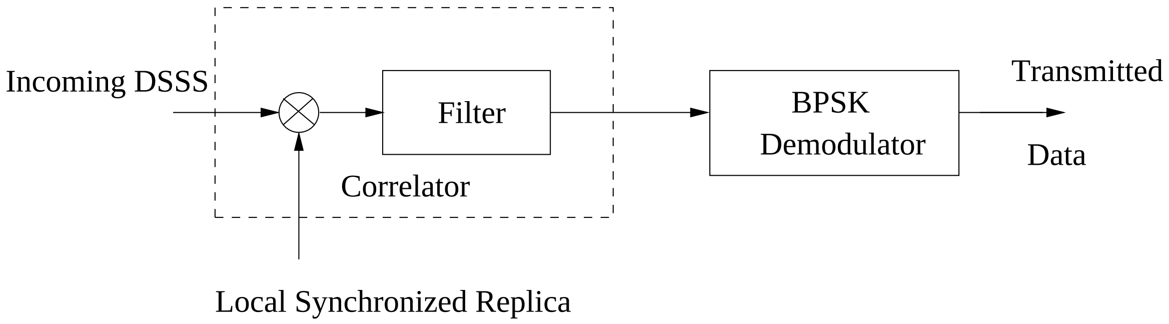


Figure 2.3. Direct Sequence Demodulator

the data. On the other hand, the *correlator*, cycles through different phases in discrete time steps, to match the incoming code. The cross-correlation is performed over N chips and the correlator output is compared to a threshold. If the threshold has not exceeded the preset value, the phase is advanced by a discrete time interval and the process repeated until the signal is detected. If the signal is not detected in a given time (dwell time), the search process is repeated again. This code acquisition problem is illustrated in Figure 2.4.

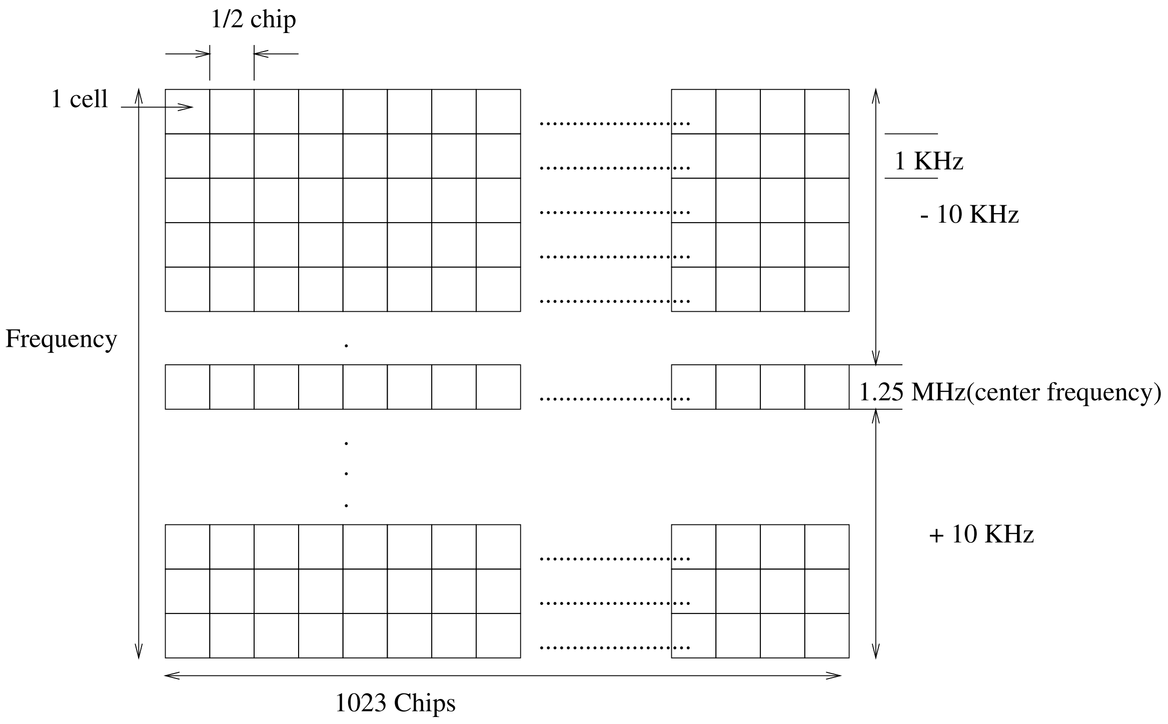
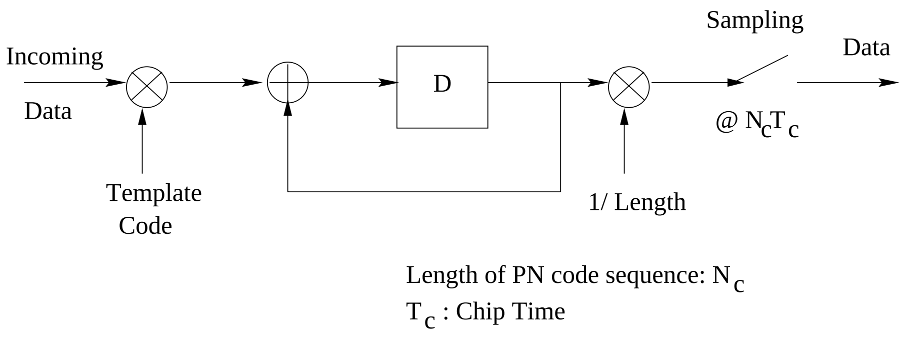


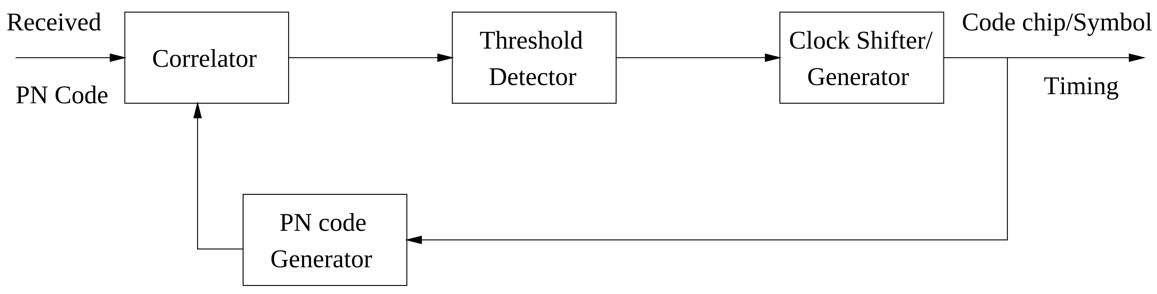
Figure 2.4. Two Dimensional Acquisition Search Space

It assumes a 2D search space to find code phase and carrier Doppler frequency without knowing the carrier phase. This is an example for GPS search space where the code phase is spread among 2046 chips and the carrier frequency spread out around 1.25MHz ±20kHz. Signal acquisition can be categorized as either:

1. serial or
2. parallel acquisition.



(a)



(b)

Figure 2.5. Sliding Correlator (a) Serial Correlator (b) Serial Acquisition

Figure 2.5 illustrates a serial correlator and a general acquisition loop.

The serial acquisition is time consuming, but has a simple implementation. On the other hand, the parallel acquisition is very difficult to implement, the complexity varying in direct proportion to the length of the code, but acquires the data quickly compared to serial technique. In the parallel correlator, the incoming pseudo-noise code is correlated with locally generated code with different code phase delays. The correlator outputs are compared to a preset threshold. The largest of the correlator outputs is the correct code phase delay, that we are interested in. As illustrated in Figure 2.6, the maximum time for acquisition in parallel approach depends on the dwell time (the Integrate and Dump time) and there is no shifting of the code phase in discrete steps.

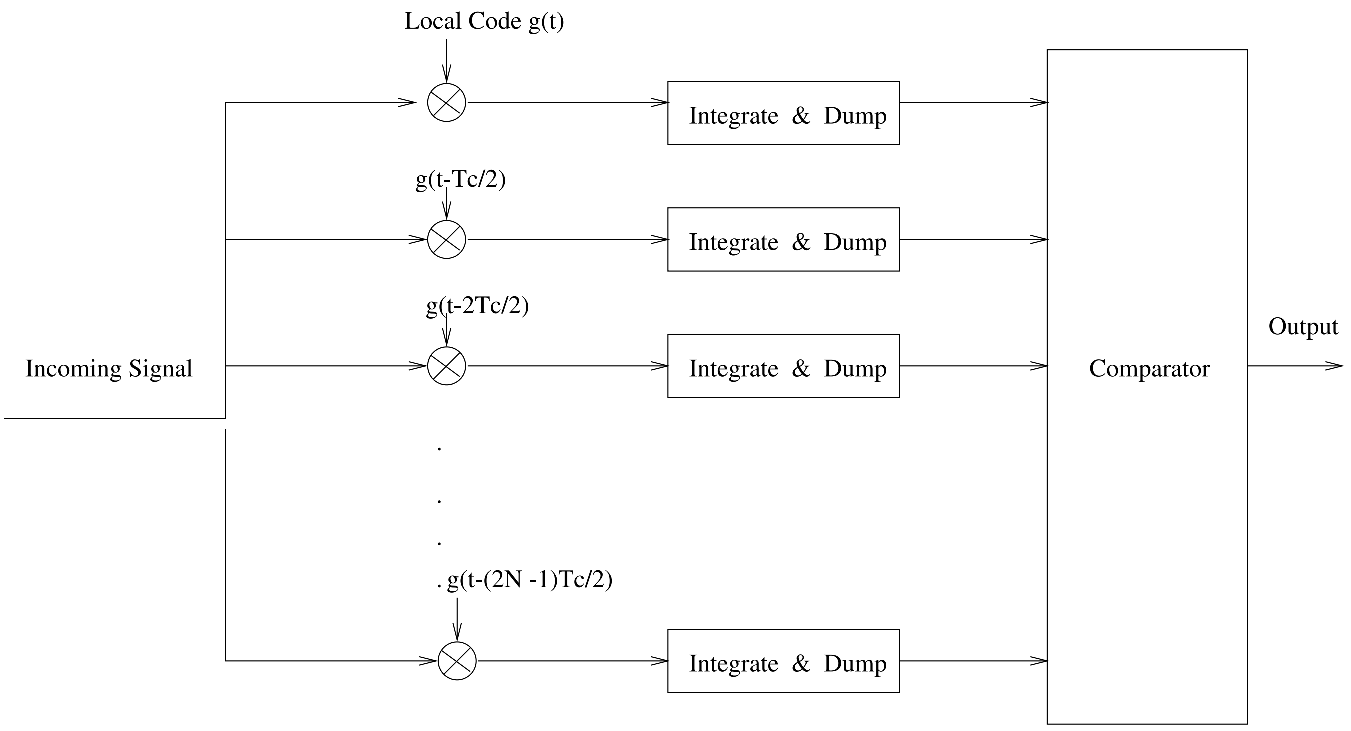


Figure 2.6. Parallel Correlator

2.1.2.2 Code Tracking

After the initial acquisition, which is a coarse search process, a fine synchronization starts known as *tracking*. The tracking maintains the local PN code in synchrony with the incoming signal to within half a chip time. There are two methods to implement code tracking in DSSS systems:

1. *Delay-Locked Loop* and

2. *Tau-dither Loop.*

In the *delay locked loop*, shown in Figure 2.7, the incoming signal is multiplied with two outputs of the local PN code generator which are delayed mutually by t $\leq T_c/2$, where T_c is the chip time. The cross correlated values are filtered and envelope detected and subtracted. This error signal is passed through a loop filter which drives a voltage controlled oscillator. VCO's output drives the PN generator. Thus, the error signal changes the phase of the local replica code so as to synchronize with the acquired signal. If the cross correlator output from one correlator is greater than the other then the VCO clock frequency is either advanced or retarded. This change tries to generate a PN code which produces a zero error signal as input to the VCO.

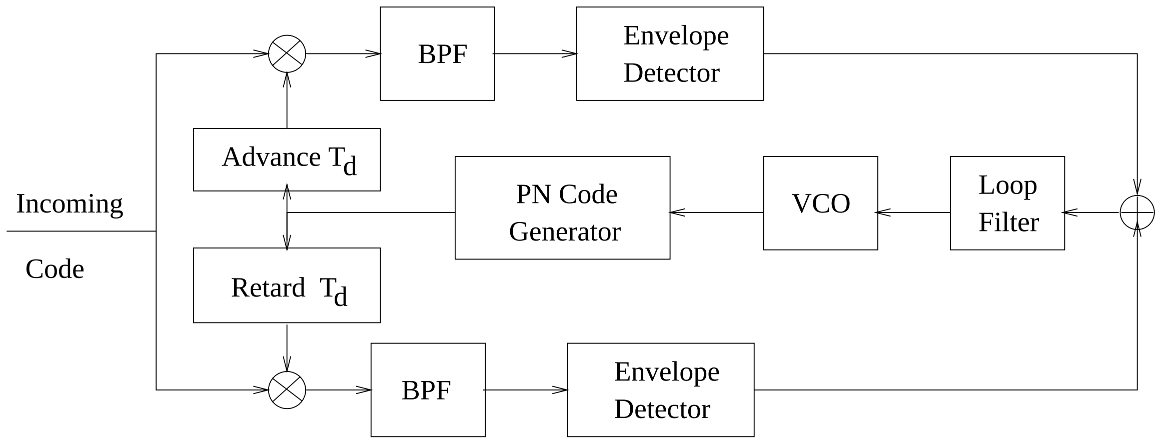


Figure 2.7. Delay Locked Loop

An *Early-Late correlator* is an example of a delay locked loop which is used in tracking GPS signals. It uses $t_\delta = T_c/2$. The codes generated are: c_E, c_P, c_L referred to the Early, Prompt, and Late codes have a relationship as shown below:

$$c_E(t) = c(t + \triangle t + 0.5T_c) \tag{2.6}$$

$$c_P(t) = c(t + \triangle t) \tag{2.7}$$

$$c_L(t) = c(t + \triangle t - 0.5T_c) \tag{2.8}$$

The correlation values and an error signal can be plotted as an S-curve as illustrated in Figure 2.8. The S-curve plots the the early and late auto-correlation. As shown these differ by T_c. The $R_e(t) - R_l(t)$ produces an error signal plotted as $e(t)$ in the S-curve [25].

As in code acquisition, one can think of coherent and non-coherent modes for code tracking. Figure 2.7 shows a coherent delay tracking loop, where the carrier phase is known. If the carrier phase is unknown, then envelope detection is used to interpret the data bit.

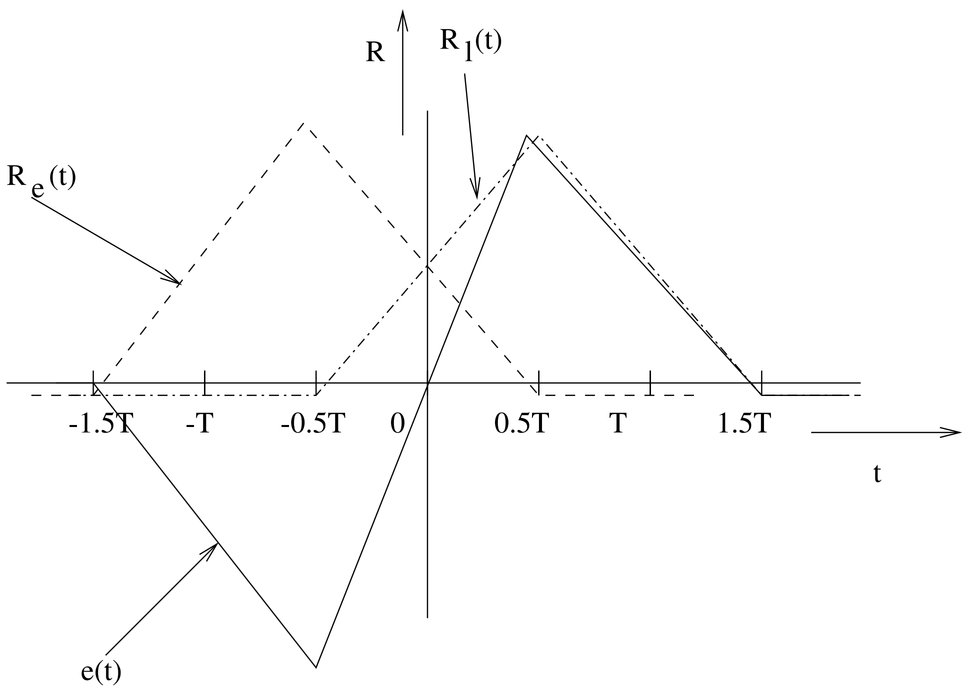


Figure 2.8. S-Curve

In the next section, a brief description of coherent and non-coherent signal processing is presented.

2.1.3 Coherent and Non-coherent Signal Processing

Signal processing in a receiver can be done with or without the knowledge of the carrier phase information. Since, it is difficult to keep track of carrier phase continuously real world applications are *non-coherent* in nature. For signals, an envelope detection is used to detect message bits. In the other case, where we have information about the phase, the demodulation process becomes simpler without the need for any non-linear operation such as envelope detection. For coherent acquisition, the sliding correlator would work perfect. Because of unknown phase, there are some changes in the acquisition technique as described below.

In non-coherent baseband processing, one uses a quadrature waveforms in addition to the envelope detection to detect the incoming signal. A non-coherent demodulating circuit is shown in Figure 2.9. If $r(t) = \sqrt{(}2/T)c(t)cos(2\pi wt + \phi)$ is the incoming signal, then the

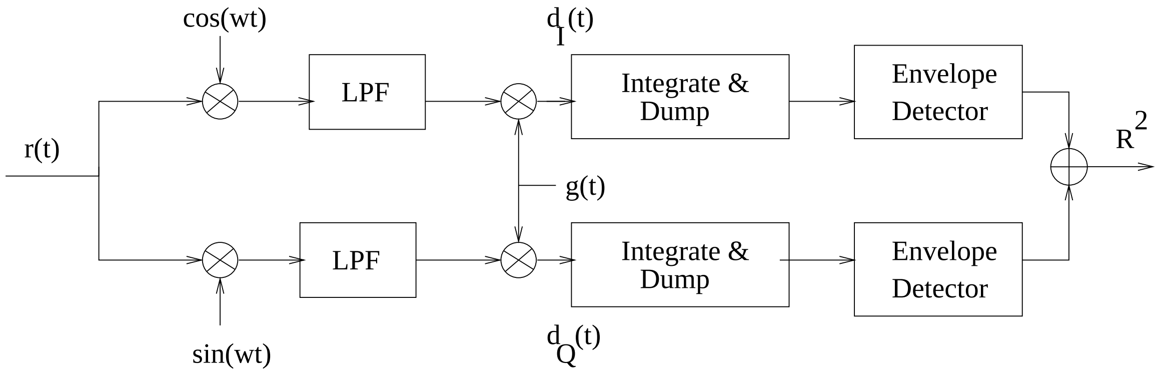


Figure 2.9. Non-coherent Demodulation

quadrature components are given as:

$$r_I(t) = \sqrt{(}2/T)c(t)cos(2\pi wt + \phi)\sqrt{(}2/T)cos(2\pi wt) \tag{2.9}$$

$$r_Q(t) = \sqrt{(}2/T)c(t)cos(2\pi wt + \phi)\sqrt{(}2/T)sin(2\pi wt) \tag{2.10}$$

Passing through the low pass filter the higher frequencies are eliminated and the resulting filter output is just dependent on the unknown phase:

$$d_I(t) = \sqrt{(}2/T)c(t)cos(\phi) \tag{2.11}$$

$$d_Q(t) = \sqrt{(}2/T)c(t)sin(\phi) \tag{2.12}$$

This signal is correlated with the local signal $\hat{c}(\tau)$. The correlator output is squared and added to get rid of the phase information. Thus, the term $R_c^2 c(\tau)$ is now compared to a threshold, to make a decision on the synchronization or data bit. As the correlating signal has a phase component, just correlating it with local code will not yield in the correct value of correlation to measure against the threshold.

For coherent signal processing, the following occurs: first we acquire the phase before any other processing occurs. Once this happens, the incoming signal is multiplied with in phase and quadrature components with proper phase relationships. The components become:

$$r_I(t) = \sqrt(2/T)c(t)cos(2\pi wt + \phi)\sqrt(2/T)cos(2\pi wt + \phi) \tag{2.13}$$

$$r_Q(t) = \sqrt(2/T)c(t)cos(2\pi wt + \phi)\sqrt(2/T)sin(2\pi wt + \phi) \tag{2.14}$$

These signals, when passed through a low pass filter to remove the higher frequency components, result in just the code which is used to modulate the carrier.

$$d_I(t) = 1/Tc(t) = d_Q(t) \tag{2.15}$$

This is the incoming signal to the correlator. As this is independent of phase there is no need for envelope detection and the c(t) is known.

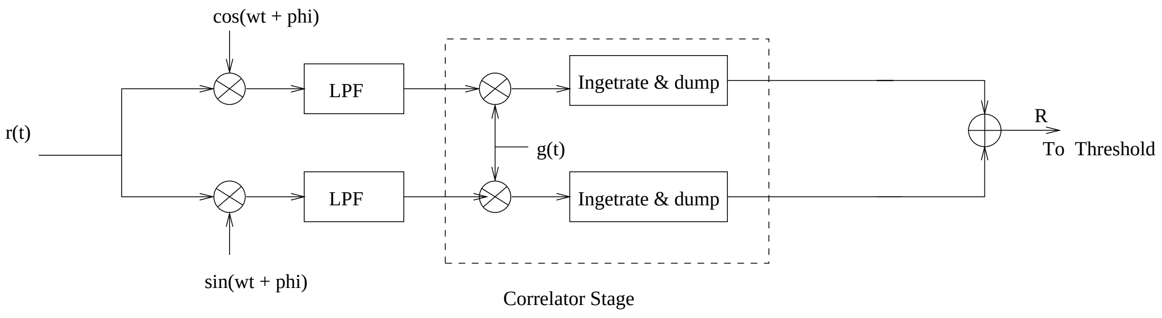


Figure 2.10. Coherent Demodulation

Figure 2.10 illustration depicts a coherent signal demodulation process. This section discussed about the Spread spectrum communications which form the basis of Global Positioning system signaling. The next section deals with Global Positioning System Signals and characteristics.

2.2 Global Positioning System

Global Positioning system makes use of celestial satellites in determining the position of a user. This is based on the principle of Triangulation. The GPS signal from a satellite is radiated at 500 Watts. By the time this signal comes to Earth after traveling about 20000 km it will be of the order of 10^{-13} watts [3]. From such signal we have to extract the navigation message and timing information to determine the location. GPS signals arriving have to deal with lot of errors such as environmental, errors due to shift in the position of the satellite and the receiver etc. The position of a user using radio signal is determined using either hyperbolation or triangulation. GPS uses the latter. Triangulation is the process in which we determine a position based on the location of three fixed points in space. Any point on the circumference of a circle one is equidistant from the radius of that circle as shown in Figure 2.11. In 2D, if a user is at a distance of a from point A and b from point B as shown he would be at locations 1 or 2. If he were to be in addition at a distance c from C then he would be at point 3 as shown. In 3D when two spheres (ranges of satellite signals) intersect we have a common plane of intersection which is a circle. This circle when intersected with a third sphere results in two points common to all three satellites, indicating the position of the user. One of these positions is in space and the other a terrestrial point. This can be determined by measurement from a fourth satellite. As shown in the Figure 2.11 the distances may not be accurate and there could be an error in the transmission time. Hence the ranges we calculate assuming perfect transmission are called pseudo-ranges.

2.2.1 GPS Segments

For the GPS to function smoothly there are three important constituents. They are: User Segment (receiving segment), the Satellite constellation and the Control Segment. The Satellite Constellation constitutes of 24 satellites at an altitude of about 20,000 km above the Earth's surface. These satellites are arranged in sets of 4 satellites in 6 orbital planes. These orbital planes are inclined to the Earth's equatorial plane at an angle of 55°. The

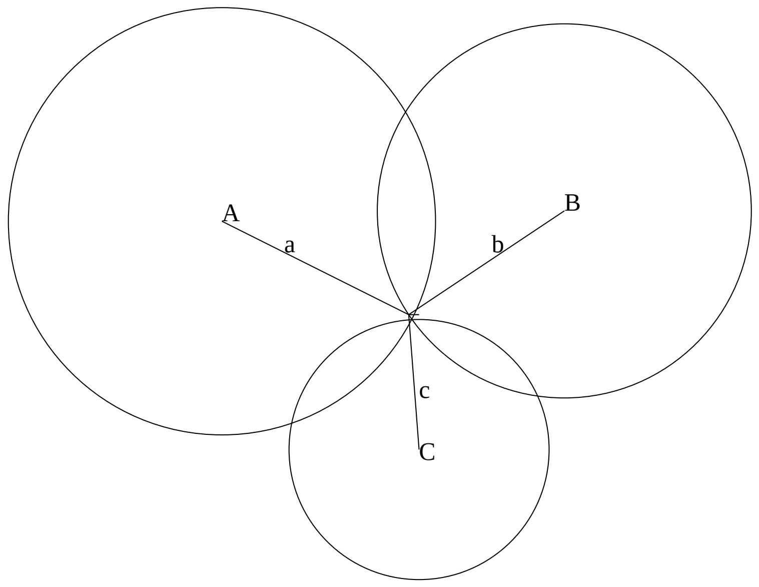


Figure 2.11. Triangulation

orbital plane location are defined by the longitude of the ascending node while the satellite location by the mean anomaly [26]. These satellites are at such a height and in such orbits such that there are at-least four satellites visible to a user at any location and at any given time. At a time one can however receive signals from 7 to 9 satellites [27]. Figure 2.12 illustrates the GPS constellation.

The Control segment consists of Master Control Station at Colorado Springs, 5 Monitor Stations located around the world to ensure maximum satellite coverage and ground antennas. The functions of the Operations Control Segment include maintaining the satellite orbital position, and monitoring the health of the satellite constellation. The health include parameters like the power, fuel levels among others. The ground stations make pseudo range measurements by passively tracking the satellites. This data is used by the Master Control Station to update the navigation message with ephemeris data, corrections and almanac data. This updated information called TT&C (Telemetry, Tracking and Command) data. This information for each satellite is uploaded by a ground up link antenna when that particular satellite is in view of the antenna.

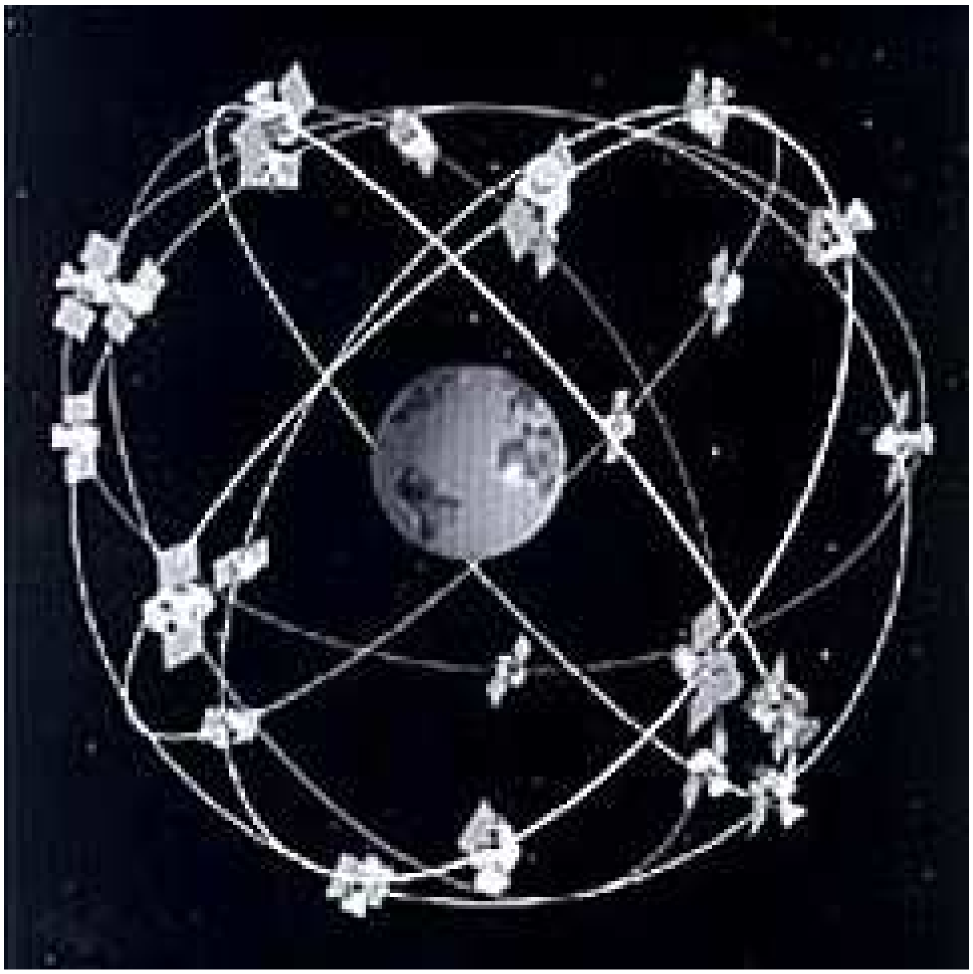


Figure 2.12. GPS Satellite Constellation

The User segment consists of GPS receivers trying to find out the user's location on the surface of the Earth. The GPS Receivers employ a hemispherical coverage antenna which has a Right Hand Circular Polarization (RHCP). The polarization ensures the differentiation between multi-path and direct path signals. The GPS receiver measures pseudo-ranges from three different satellites to compute the user's location. It does this by calculating the code and carrier phases and also demodulating the navigation message data. Apart from the position the receiver may also perform the PVT measurements. The GPS receivers are of two kinds: (i) Tracking a P(Y) code and C/A code on dual frequencies L1 and L2 or (ii) Tracking C/A code alone on L1 frequency. Apart from this categorization, the receivers can be grouped on the combination of which data they are processing (carrier phases or pseudo-ranges) and which code is available (C/A, P, or Y code). They are [28]:

1. C/A code pseudo-range,
2. C/A carrier phase,
3. P-code carrier phase, and
4. Y-code carrier phase receivers.

The carrier phase receivers result in a greater accuracy than pseudo-range measurements. Also the P and Y codes have better accuracy (up to 1 m discrepancy) compared to the C/A codes (which have an accuracy of up to 10 m). The three code segments are shown in Figure 2.13.

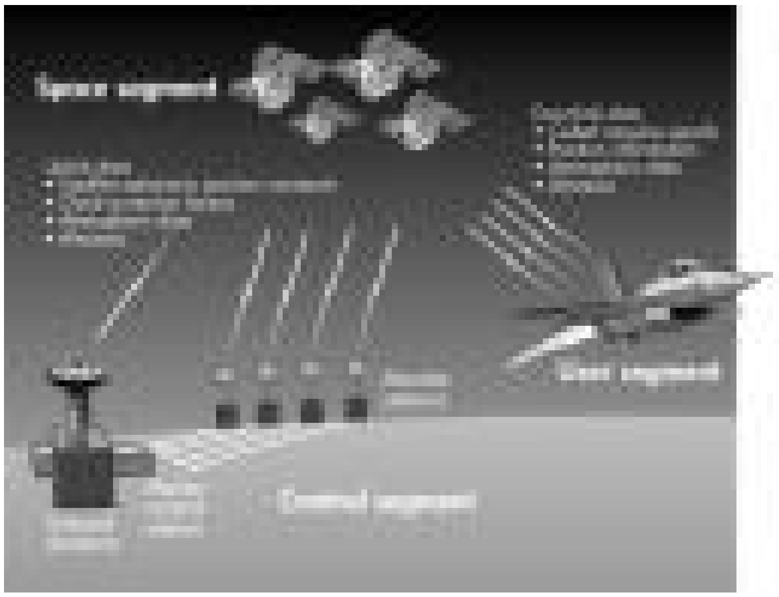


Figure 2.13. GPS Segments

Next we deal about the signals which are manipulated in this system.

2.2.2 Signals

The GPS signal is a Direct Sequence BPSK spread spectrum signal represented as:

$$s(t) = A(t)c(t)d(t)sin(2\pi(f_o + \triangle f)t + \phi + \triangle\phi) \tag{2.16}$$

where

A(t): amplitude of the transmitted signal

c(t): pseudo-random code (C/A code of the satellite 1.023 MHz)

d(t): navigation data stream (50 Hz)

f_0: carrier frequency of the transmitted frequency (1.575 GHz)

$\triangle f$: frequency offset due to relative position change

ϕ: original carrier phase

$\triangle\phi$: carrier phase offset

Since this is a BPSK signal corresponding to c(t) there is a phase change of $\pm\pi$. The rate at which the code phase changes is termed as the chip rate. The GPS signals are radio frequency signals and of the order of GHz. They have been used for both civilian and military purposes. The two commonly used frequencies are the L1 and L2 frequencies which are 1.575 and 1.227 GHz respectively. The GPS signal is modulated by two kinds of code P (Precision) and C/A (Coarse/Acquisition code) code. The C/A code is 1023 chip sequence generated at 1.023 MHz and repeats at every 1ms while the Precision code is at 10.23 MHz and repeats for every week. Though the frequency is same there is little interference between the signals from different satellites because the modulating codes (part of Gold codes) are near orthogonal, i.e. the cross-correlation is zero or small.

2.2.2.1 PRN Codes

Various codes can be used as Pseudo Random Sequences which are used to modulate the carrier signal. The major ones are: Gold, Kasami, Walsh Transforms, m-sequence. These codes have different auto-correlation and cross correlation properties which determine their utility. The autocorrelation which measures the amount of similarity between two waves is used in GPS Signal demodulation to determine whether the local replica code is matching the incoming the received signal. The codes described above are generated using Linear Feedback Shift Registers. The C/A code, which belongs to the family of Gold Codes is a 1023 chip sequence and is generated as illustrated in the Figure 2.14.

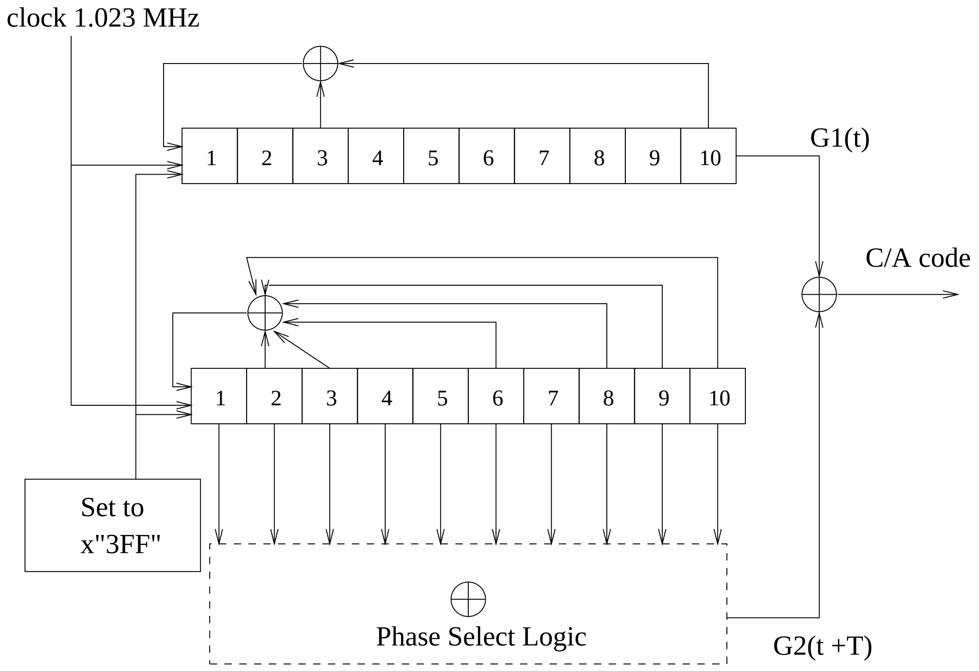


Figure 2.14. Code Generator

Two polynomials are used to describe the shift registers. For C/A case

$$G1 = 1 + x^3 + x^{10}, G2 = 1 + x^2 + x^3 + x^6 + x^8 + x^9 + x^{10} \tag{2.17}$$

are the two polynomials. The unique C/A code for each satellite is obtained from the modulo 2 sum of the delayed version of G2 register and G1 register. By adding two phases of a PN code we get another phase but not another code. A different tap combination results in a different code. There are 32 PRN numbers each associated with a satellite and

Table 2.2. Comparision of C/A and P Codes

Signal Property	C/A	Precision
Chip Rate	1.023×10^6	10.23×10^6
Code Length	1023	6.1871×10^{12}
AutoCorrelation Period	1 ms	1 week
Chip Time	977.5 ns	97.8 ns

another 5 associated with ground transmitters. C/A code is of length $2^n - 1$ where n is the length of the shift register.

The P code is generated using four linear feed back shift registers called X1A, X2A, X1B, and X2B. The repetition of such a code is 1 week and it is of length 6.1871×10^{12} chips. For more information on P code one can refer [29]. A brief comparision between the two codes is given in Table 2.2.

The autocorrelation process is fundamental to signal demodulation of the GPS signal. The autocorrelation function of a random binary code is similar to the pulse waveform. The GPS codes are however periodic hence these codes are called Pseudo Random Codes. The autocorrelation of a GPS C/A code is given as:

$$R_c(\tau) = 1/1023 * \int_0^T c(t) * c(t+\tau) dt \tag{2.18}$$

The autocorrelation function of a C/A code is shown in Figure 2.15. It is a repeating sequence with period 1 ms. It takes different side lobe values including as 63/1023, -65/1023, -1/1023, 1023/1023 [1].

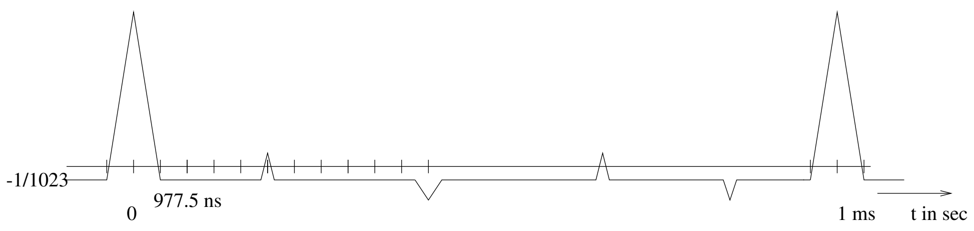


Figure 2.15. C/A Code Autocorrelation

Assuming that both the transmitter and the receiver are synchronized, we can first get the delay and hence the time of transmission of the signal from a particular satellite. Knowing this the pseudo-ranges can be calculated as :

$$R = c(T_2 - T_1) \tag{2.19}$$

where

T_1: time of Transmission at the satellite with respect to GPS Time

T_2: time of Reception at the receiver

The illustration in Figure 2.16 represents the delay along with transmitted modulated data. The received code is compared with a replica. The difference in time gives us the amount of delay (also the T_d).

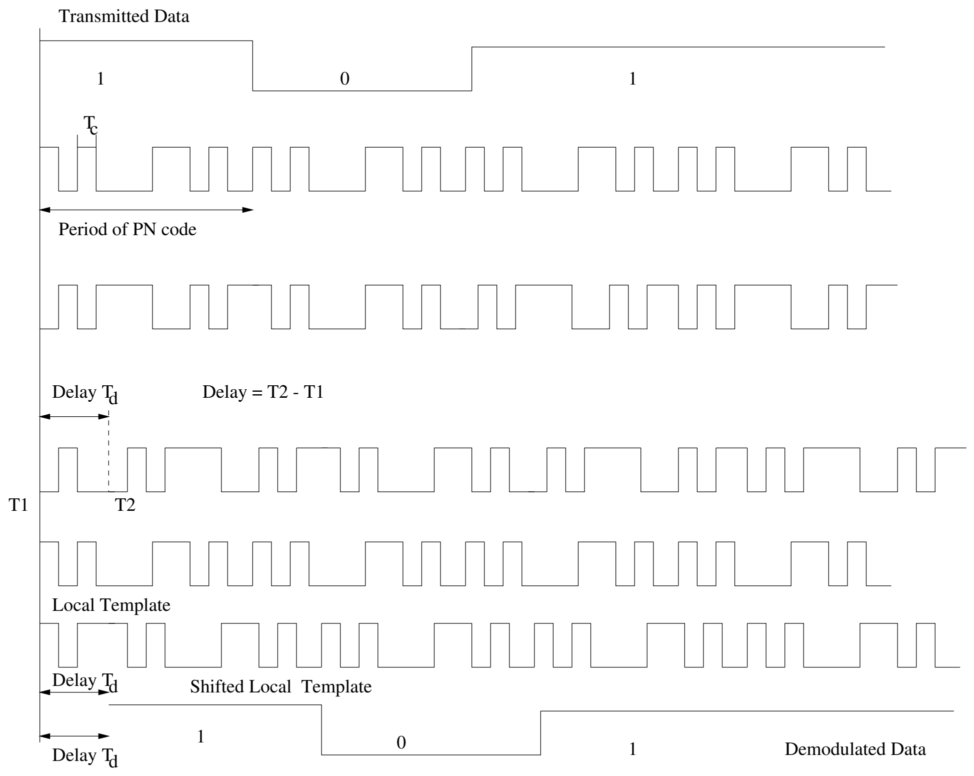


Figure 2.16. Position Determination using Pseudo-codes

However the clocks are not synchronized and there is an error which has to be taken into account. Then the true geometric range becomes

$$R = c[(T_u - t_u) - (T_s - t_s)] = c[T_u - T_s] + c[t_s - t_u] = r + c \triangle t \tag{2.20}$$

where

$\triangle t = t_s - t_u$

T_u: time of reception of the signal at the receiver

T_s: time of transmission of the signal from satellite

t_s: offset of the satellite clock from system time

t_u: offset of the receiver clock from system time.

c: velocity of light in vacuum ($3 \times 10^8 ms^{-1}$).

One can calculate the position of the user in (x,y,z) from a knowledge of the initial positions of satellites (x_i, y_i, z_i) and also the pseudo-ranges which can be calculated from above. If (x_i, y_i, z_i) are the position of satellite i and if we know such positions for four satellites one can solve 4 equations to get the unknown receiver position as:

$$PR_i = \sqrt(((x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2) + \triangle e \tag{2.21}$$

where

$\triangle e$: represents errors.

These errors and biases could result from various sources. These include [30]:

1. Ephemeris Errors - relating to non-accurate available almanac data after a period of 4 hours

2. Satellite and Receiver clock errors - resulting from non synchronization of the satellite and receiver

3. Multi-path error - dealing with signals arriving after reflections instead of arriving directly. It distorts the incoming signal and affects carrier phase and pseudo-range measurements

4. Receiver Measurement Noise - related to the limitations of the receiver electronics

5. Ionospheric delay - The ionospheric layer in the atmosphere acts as a dispersive medium, bending the signal and changing its speed. The change in speed causes an error in measurement

6. Tropospheric Delay - This causes a delay in the signal thereby we measure a different distance which is longer than the actual distance.

The above system of four non-linear equations can be solved using Kalman filtering or closed form techniques or iterative techniques based on linearization [26].

2.2.2.2 Signal Data Structure

The data format for a GPS signal is shown in Figure 2.17 below. The basic element is 1 ms duration C/A code of 1023 chips. The navigation data bit has a data rate of 50 Hz and it is 20 ms long and contains 20 C/A codes. Thirty such data bits constitute a word of 600 ms long. Ten words make up a sub-frame 6 sec long. Five such sub-frames make a frame which is 30 sec long. Twenty five such pages constitute a complete data set equal to 12.5 minutes long. To determine the position of a user one needs to know the position of the satellites in addition to the distances from the satellites to the user as we have seen above. The first three sub-frames contain the required data. In effect 18 sec of data is necessary to calculate a user position. To report the data for computation of position the GPS uses NMEA or RINEX formats. NMEA (National Marine Electronics Association) is a communication standard developed for marine instruments. Another industry standard is the RINEX (Receiver Independent Exchange).This is an ASCII format and is used for exchange of data between different receivers.

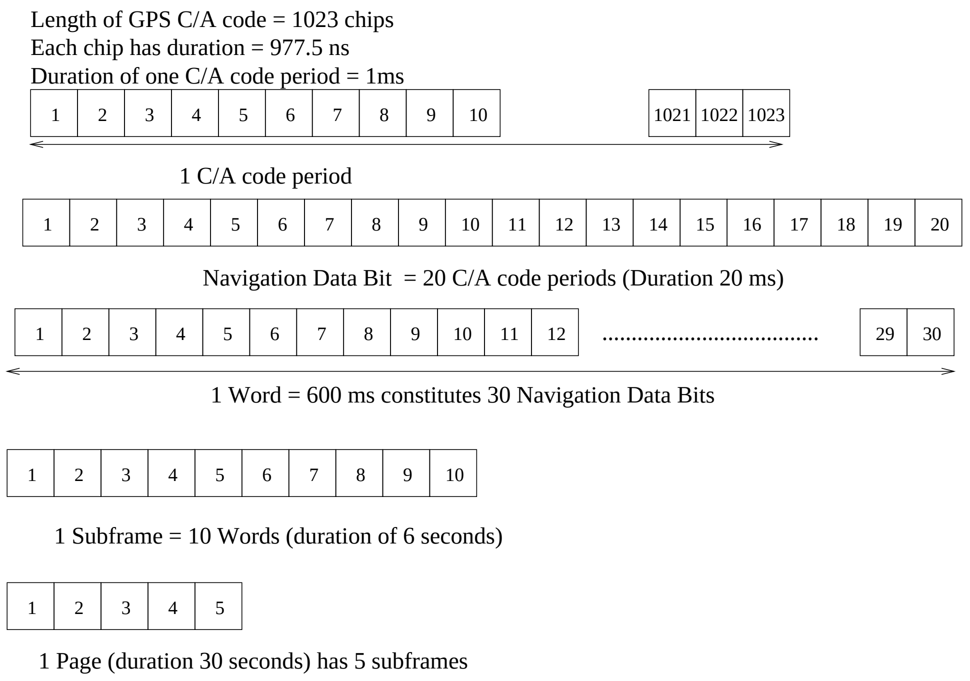

Length of GPS C/A code = 1023 chips
Each chip has duration = 977.5 ns
Duration of one C/A code period = 1ms
1 2 3 4 5 6 7 8 9 10 1021 1022 1023
1 C/A code period
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Navigation Data Bit = 20 C/A code periods (Duration 20 ms)
1 2 3 4 5 6 7 8 9 10 11 12 .... 29 30
1 Word = 600 ms constitutes 30 Navigation Data Bits
1 2 3 4 5 6 7 8 9 10
1 Subframe = 10 Words (duration of 6 seconds)
1 2 3 4 5
1 Page (duration 30 seconds) has 5 subframes

Figure 2.17. GPS Chip and Data Bit Structure

2.3 Summary

This chapter outlined the basic concepts of spread spectrum communications, the model, modulation and demodulation schemes. Various detectors were introduced. It also described the GPS signals and segments, useful for GPS signal propagation and detection. It discussed about the correlation properties of the GPS signals which are instrumental in signal demodulation and detection.

CHAPTER 3

GLOBAL POSITIONING SYSTEM RECEIVER

The GPS user segment as described in Chapter 2 consists of GPS receivers. The control segments uploads the ephemeris data and the GPS receiver uses this data to calculate the position. The receiver is described here, with emphasis on the digital backend processor. A brief description of the various components is outlined.

3.1 Receiver Configurations

State-of-the-art receivers have multi-channels for signal reception. Advantages of multi-channel over single channel are described below and types of receiver starts are introduced.

3.1.1 Single and Multi-channel Receivers

A GPS signal as mentioned before is a Spread Spectrum based signal. Using a GPS receiver, we can determine our location at any point on the earth's surface. GPS receivers are categorized broadly based on whether they have a single channel for each satellite or multiplex different satellite signals with one channel. They can be single-channel sequential, single-channel multiplexed, or single channel per satellite. In single channel sequential receiver, each of the satellites are tracked continuously, one at a time for few seconds, before tracking another satellite. In single channel multiplexing, the sequencing rate is high so that the data from four satellites are viewed simultaneously. In multiple channel receivers, which is the norm today each of the satellite is assigned a single channel. Today's receivers come with twelve parallel channels and operate on both the L1 and L2 frequencies. As mentioned earlier four satellites are needed to know the position, and at any point in time there are at least 7-9 satellites visible to the receiver. Hence, special precision and backup capability are the advantages of having more than 5 channels [31].

For the receiver to operate, some information must be known up front. The visible satellite information stored in the almanac helps in the signal acquisition. This information also helps in speeding up the search process instead of a cold start, when no information about the satellites in range is known. Typically, a GPS receiver takes about 30 seconds to read the ephemeris data for a satellite after signal acquisition. A generic GPS receiver is presented here. We also briefly describe the signal flow inside the receiver. A traditional GPS receiver, shown in Figure 3.1 consists of two stages: the analog front end and the digital back end. The analog front end described below is used to condition the signal and generate the input for the digital back end. It takes in incoming satellite signals with powers of the order 10^{-16} Watts and identifies them in the presence of noise at its input 4000 times stronger than the satellite signal. The digital back end, which is usually a digital signal processor, takes the input and computes the position and location information. Though it is usually a chip set, only recently have there been attempts to have a single chip GPS receiver. Irrespective of the implementation, the generic GPS receiver functional block diagram is as shown in the Figure 3.1.

As shown, the main components/blocks of the GPS receiver include the Antenna, Pre-Amplifier, Frequency synthesizer, Frequency down-conversion to IF, Analog to Digital conversion, and signal processing. The antenna is used to collect the GPS signals. Various parameters such as the satellite visibility profile, polarization of the incoming signals, elevation, etc., determine the design of the antenna. The antenna is a Right Hand Circular Polarization antenna with hemispherical visibility. Signals which have been received using the antenna are very weak in strength. This is due to many factors such as the distances traveled by the GPS signal, the multi path interference by other frequencies, and the multiuser environment etc. For further processing, the received signals are amplified and conditioned by the Pre-amplifier. The reference oscillator provides the time and frequency reference for the receiver. This has to be stable, because the important measurements in GPS are the

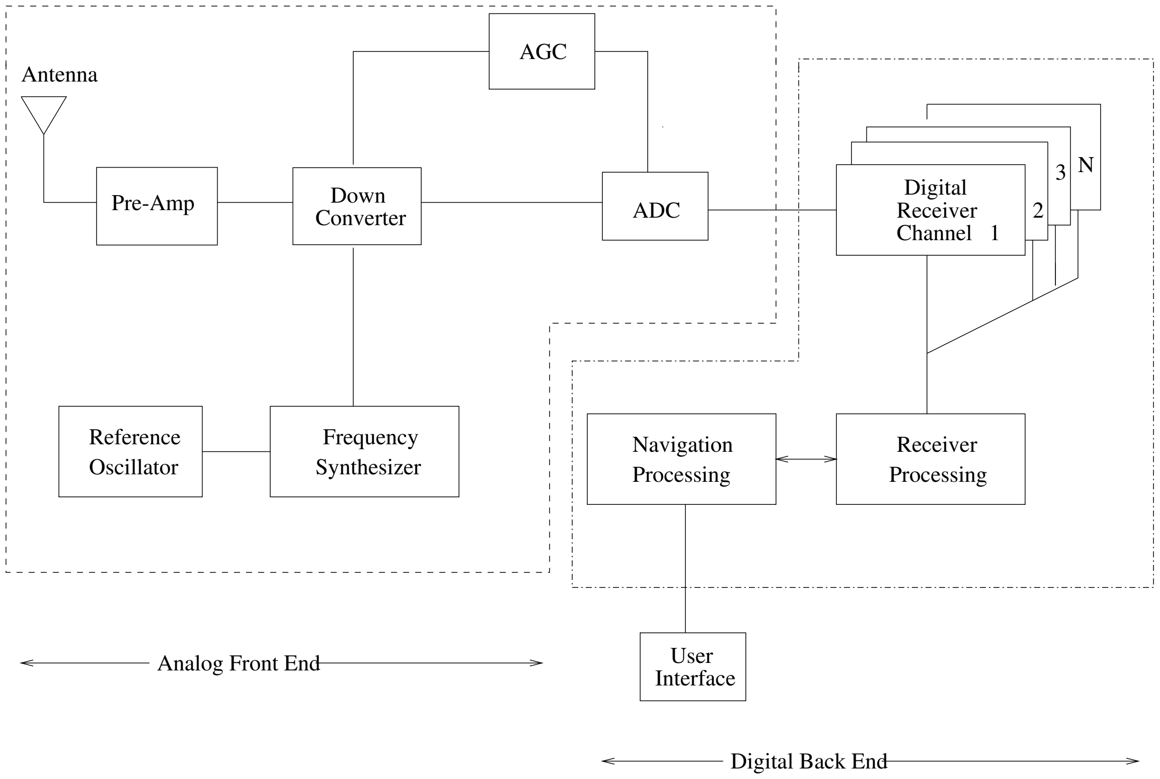


Figure 3.1. GPS Receiver

time of arrival of the signal and the carrier phase which are used to calculate the pseudo-range of the satellite. Either rubidium, or oven controlled crystal oscillators, or temperature controlled crystal oscillators are in use. The frequency synthesizer is required to generate the reference sampling clocks, local IF to which the input signal has to be down-converted as well as signal processing clocks. The down-converter mixes the local oscillations generated by the frequency synthesizer with the incoming signal to generate the IF frequency signal for further processing. The down-conversion is usually in two stages and preserves the PRN codes and the Doppler effects. [32] however refers to design of a direct conversion GPS receiver. In direct conversion, the 1.5 GHz signal is sampled without the intermediate IF stages, thereby eliminating the need for local oscillators and PLLs. The ADC and AGC are useful for rejecting unwanted sidebands after mixing and to maintain a constant amplitude for signal processing. The AGC ensures that the signal amplitude is spread amongst the quantization levels of ADC. GPS receivers use a 1-bit or 2-bit data and hence have two or four quantization levels. Two approaches are possible for digitizing the incoming signal

1. direct digitization of the L1 signal; or

2. down-conversion of the input signal to IF followed by digitization.

The former removes the need for mixer and other analog components but the ADC must operate at high frequency. The down-converted approach uses mixers and analog components to down convert the RF signal to IF range and then manipulates this with practical ADCs. The sampling frequency selection is related to C/A code chip rate. It should be not a multiple of the C/A chip rate, since in that case the synchronization is not achieved [20]. The relation between the incoming IF signal, sampling frequency, and the output frequency can be derived from:

$$f_o = f_i - nf_s/2 \tag{3.1}$$

where

$f_o \leq f_i/2$

f_o: output frequency

f_s: sampling frequency

f_i: input frequency

n: integer

Also if $\triangle f$ is the bandwidth of the input signal, then according to Nyquist requirement the sampling frequency f_s, should be greater than $2 \times \triangle f$ (practically $2.5 \times \triangle f$). Given the bandwidth of C/A code signal to be 2MHz null-to-null, we can safely choose 5 MHz to be the sampling frequency. The relation between the output signal, sampling frequency, and the bandwidth of the incoming signal can be expressed as:

$$f_o \approx f_s/4 \tag{3.2}$$

where

$f_s \geq 2 \times \triangle f$

The digital receiver channel generates the local Pseudo-Noise codes and changes in phase and operates on the incoming signal samples. It is used to acquire the satellite signals, tracking the code and carrier signals etc. [33]. The pseudo-ranges and time tags and GPS system data which are used in Navigation processing are outputs of signal processing. The input IF signal can be represented as:

$$s_i(nT_s) = A\hat{c}_i(nT_s)\hat{d}(nT_s)sin(2\pi(f_{IF} + \triangle f)nT_s + \phi) \tag{3.3}$$

where

s_i : i^{th} satellite information

nT_s: sampling interval T= $1/f_s$

A: Amplitude

$\hat{c}_i(nT_s)$: sampled and delayed C/A code

$\hat{d}(nT_s)$: delayed navigation data bit

$\triangle f$: frequency offset

Parallel processing as shown in Figure 3.1 is performed on the IF to track visible satellites simultaneously by the individual receiver channels. The GPS receiver measures the code phase for pseudo-range measurements from the satellite signals. It also extracts the carrier frequency and if it is in phase lock with the incoming signal then it could calculate the delta pseudo-range measurements [33].

Figure 3.2 below shows the receiver channel where the back end digital processing is performed.

Demodulation takes place within the receiver channel. As described previously, it constitutes two phases: Acquisition and Tracking. Signal Acquisition in a GPS receiver refers to problem of searching different satellite signals. Depending upon the kind of receiver hardware or software used there are different techniques. The hardware receiver uses continuous time domain correlation to acquire while the software receiver uses blocks of data

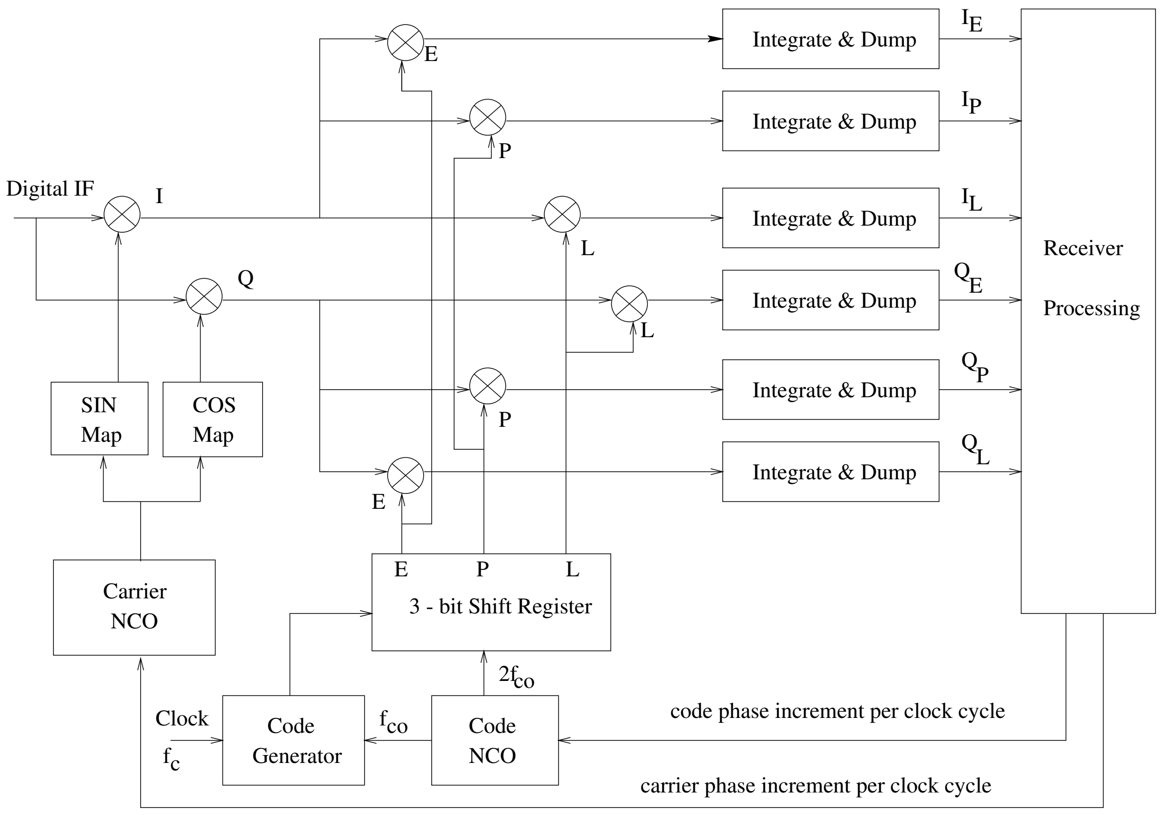


Figure 3.2. Digital Receiver Channel

to perform the acquisition. The design goal of any GPS receiver should be the ability to track the data after it has been acquired without any delay. The amount of data used for acquisition purposes is important. There is a navigation data bit change every 20 ms. Therefore, if we consider two consecutive 10 ms intervals, then we can determine if there was a transition. However, since a C/A code is 1 ms long the phase changes every 1 ms, and one can consider 1 ms duration for strong signals for signal acquisition.

Based on the availability of prior information of the presence of a satellite, there are three starts:

1. Hot start: When a GPS receiver has been switched off for less than 4 hours, it still has the almanac data valid and hence knows which satellites are visible. Based on the last

known position, current time from local clock, and satellite visibility, the acquisition is quick.

2. Warm Start: The receiver has been switched off for more than 4 hours resulting in less than accurate ephemeris data. This determines a rough list of satellites which are in range.

3. Cold Start: This takes the longest time to acquire. In this case the receiver has no prior information and the receiver has to randomly search for arbitrary satellites.

In the beginning, eight channels are used to acquire signals from any eight satellites for some time. If any channel does not acquire a satellite, then a different satellite is searched. The time, when the receiver reports the position first after powering up, is called the *Time to First Fix*. Depending upon the start, this can vary from under 18 seconds (Hot Start), or under 45 seconds (Warm Start) to 3 minutes (Cold Start). GPS signal acquisition follows the same lines as the spread spectrum signal acquisition described in Chapter 2. Figure 2.4, describes the search space of GPS signal acquisition problem in 2D, i.e., we are interested in the code phase and carrier frequency, given which satellites we are looking at. The GPS C/A code is searched in steps of $T_c/2$ where T_c represents the chip time ie., it searches over 2046 bins. Also, the Doppler frequency offset can be approximately $\pm$10kHz. Hence, the receiver search space has 20 bins each of 1kHz, around the center frequency of the IF carrier signal. The total number of acquisition cells would then be approximately 40,000. The dwell time for each cell can vary from less than 1 ms for strong signals to 20 ms for weak signals. The acquisition configuration within a receiver channel is shown in Figure 3.3.

During acquisition, the correlation peak is detected and the receiver calculates an envelope to determine if the correlation peak has crossed the threshold. In each bin, the envelope is estimated and compared to a threshold to determine the presence or absence of a signal. The detection is described by a PDF (probability density function) for each cell. Figure 3.4 (shaded portion) shows the PDF of detection.

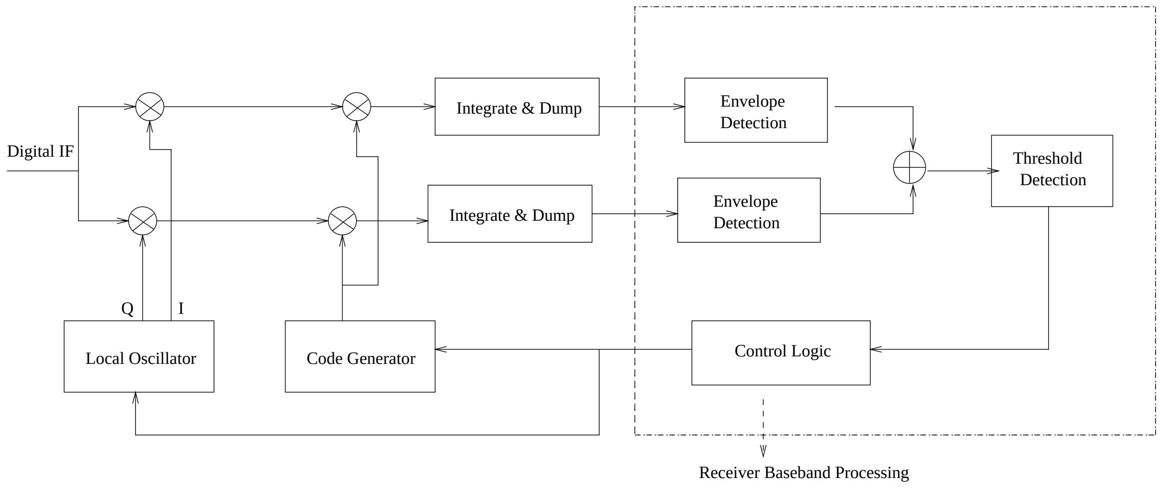


Figure 3.3. Channel Serial Acquisition

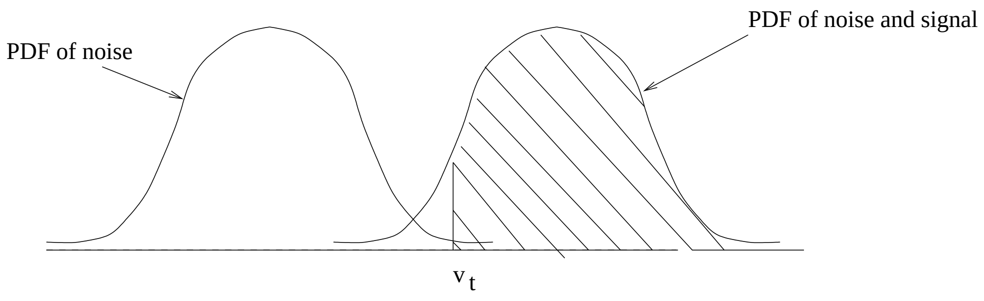


Figure 3.4. PDF of Detection

If we assume a dwell time of 1 ms, then the minimum time to acquire a signal would be 40 seconds. Once the signal has been acquired, the next phase of tracking starts. Figure 3.5 shows the tracking of both carrier and code in a channel.

Both the code and carrier are continuously tracked, using the receiver channel in Figure 3.5. Inherently, there is a phase locked loop to track the carrier and a delay locked loop to track the code phase, respectively. During tracking, if the code changes suddenly or drifts beyond tracking range, then the receiver has to re-acquire and the tracking operation comes to a halt. This is known as *loss-of-lock*. The carrier tracking is performed to lock the incoming signal phase with the locally generated carrier signal phase. This is performed

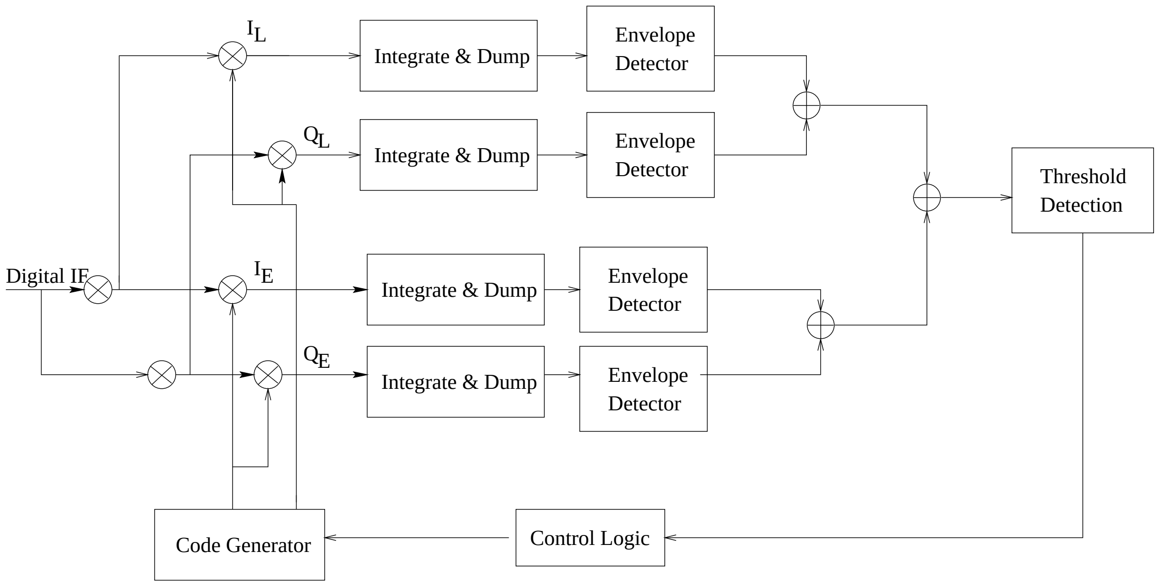


Figure 3.5. Tracking Loops

using Phase Locked Loops to lock the phase of the signals. The incoming signal is mixed with the local signal in quadrature to convert it to baseband. This step is known as *carrier wipe off* since the resulting signal doesnot contain any carrier component. The signal is then correlated with prompt C/A code (this results in a *code wipe off*) and integrated as shown in Figure 3.5. The Accumulate and Dump acts as a low pass filter and filters the double frequency term in the quadrature mixer output leaving only the correlation value. The carrier loop discriminator gives an output phase difference which is fed to the synthesizer to generate appropriate phase to keep the signal in synchronization.

The code tracking is based on a Early-late correlation value. When a signal is acquired, the local replica is within a chip time of the incoming signal. The delay locked loop tracks the signal by generating an early and a late signal. These are generated using a 3-bit shift register in Figure 3.2. The shift register is clocked at $2f_c$, while the code generator is clocked at f_c. The incoming code is correlated with both the early and late code samples. The correlation envelopes are subtracted to get an error signal. This determines whether the clock has to be advanced or retarded. Both the early and late samples are 1/2 chip time delayed with respect to prompt code.

The receiver tries to match the replica code with the incoming code along with any change due to Doppler and also the replica carrier frequency with the incoming frequency adjusted with Doppler. The receiver measures the time delay and hence can calculate the pseudo-ranges. The ephemeris data which are obtained from decoding the sub-frames are also used in conjunction with the pseudo-ranges by the navigation processor to compute the position of the user and also the velocity.

3.2 Summary

This chapter discussed about a GPS receiver and the digital back end functions. It discussed how a receiver could be operated and what signals it manipulates upon. The different spread spectrum demodulation schemes applied to a GPS receiver were described. The next chapter discusses the VHDL Implementation of the digital backend.

CHAPTER 4

VHDL DESIGN AND IMPLEMENTATION

This chapter presents the implementation details of the *GPS Digital Receiver* explained thus far. First, is outlined the standard procedure in the design of a digital ASIC design.

The GPS receiver, as has been described before, consists of two parts: analog front end and digital processor. Here is described briefly, the design flow for the analog front end but the scope of this work is only the digital processor. The design of the RF system of the GPS receiver involves RF circuit design flow. The design flow first requires the system specifications which are used to determine the RF circuits. These RF circuits are simulated using software such as *Matlab* or *ADS*. Apart from the RF system in a receiver there are also the digital circuits which process the RF system output.circuit system design flow is shown in figure. The digital circuit system is first designed at an algorithmic level. This describes the implementation in terms of basic modules and the interconnection between them. The modules can be described at the behavioral or structural level of design abstraction [22]. The behavioral description represents the function. This behavioral module is then simulated and validated. The validated design is next synthesized into a netlist, which can be realized on hardware (either ASIC or FPGA). A place and route tool takes the gate-level netlist and generates a layout based on the component library of the target technology. This design is re-simulated to take care of the timing constraints and delays until the timing conditions are met. The final layout is realized onto a silicon substrate. The design of a digital receiver involves both digital and RF circuitry. Problems of coupling effects of the analog RF signal on the digital circuitry, have to be taken care of during the design.

4.1 Digital System Design and Implementation

Behavioral level VHDL code is written to implement the digital processing of the signal. The input to the receiver is digital IF; we assume sampled digital data for simulation purposes. As described in the previous chapter, the incoming signal is first down converted to IF range which has a center frequency of $f_c = 21.25$ MHz and then sampled at 5 MHz to give a 1.25MHz signal input to the baseband processing. Since the code and carrier phase information is preserved during down-conversion, the incoming signal of 1.25MHz which is modulated by a given PN code as well as data, is assumed for simulation purposes. The digital backend, illustrated in Figure 3.1, includes demodulator, GPS C/A code generator, and code and carrier synchronization loops. Behavioral VHDL descriptions have been written and verified to have the desired functionality of the following components:

1. Accumulate and Dump,
2. Linear Feedback Shift Registers,
3. Direct Digital frequency synthesizer,
4. Correlator and matched filters,
5. Digital synthetic data generation,
6. BPSK IF signal generation, etc.

4.1.1 Accumulate and Dump

An *Accumulate and Dump* is used in a receiver as shown in Figure 3.1 as an *Integrate and dump*. It implements the function of a low pass filter. This communication block adds samples over a time interval and at the end of this interval resets, hence the name Integrate and Dump. It can be implemented as shown in the Figure 4.1. The integrate and dump implements the function shown below:

$$y_k = \sum_{i=(k-1)N}^{k} N x_i \tag{4.1}$$

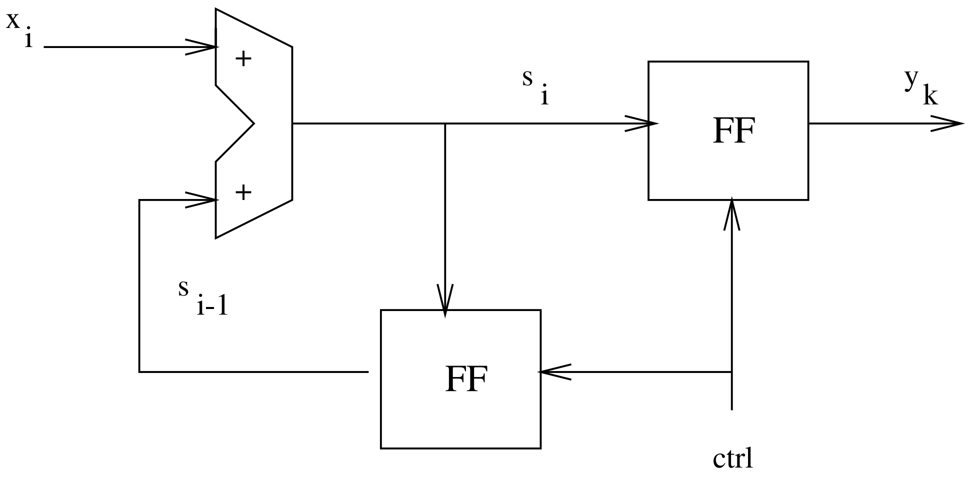


Figure 4.1. Integrate and Dump

The integrate and dump becomes a binary matched filter in the event of the incoming signal being rectangular signal pulses [34]. The following algorithm describes the function of the Integrate and dump.

Algorithm: Integrate and Dump

```
if $count < N$ then
    $count \leftarrow count + 1$
    $sum \leftarrow sum + data$
else
    $output \leftarrow sum$
    $count \leftarrow 0$
end if
```

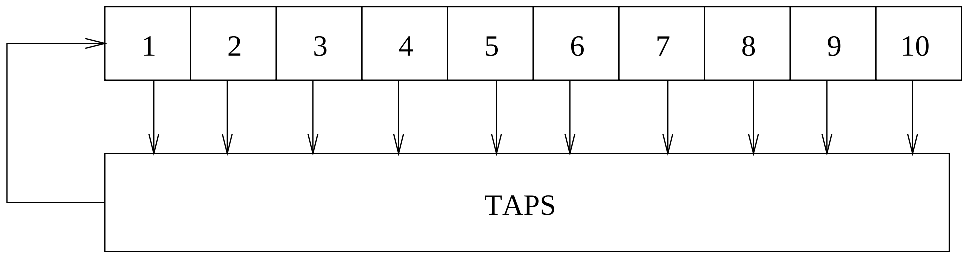


Figure 4.2. Linear Feedback Shift Register

4.1.2 Linear Feedback Shift Register (LFSR)

A feedback shift register is a register which has the input to be a modulo-2 sum of its outputs. The LFSR is used in generation of the pseudo-random codes in the system implemented. Based on the taps used, one can get different codes from a LFSR. For the GPS system, the C/A code uses two 10-bit LFSBs as given in equation 2.17. Mathematically, a LFSR polynomial can be represented as:

$$C(s) = z^k + c_{k-1}z^{k-1} + ... + c_1 z + 1 \tag{4.2}$$

where c_i are known as the taps which determine what bits have to be modulo-2 added. If $c_i = 1$, the bit is modulo-2 added else the corresponding bit doesn't add to the output sum. The multiplication and addition are done using AND and XOR gates.

4.1.3 Numerical Controlled Oscillator (NCO)

A numerical controlled oscillator is the digital counterpart of an analog voltage controlled oscillator. Based on the error voltage, the output frequency is changed in an analog VCO. Similarly, in a digital NCO, based on the error input word the output frequency is altered. An NCO consists of an accumulator, to which an incoming error signal is added. This error signal decides the output frequency of the NCO. If f_c is the clock frequency, ϕ, the incoming error signal magnitude, n, the number of bits of the accumulator, the free running output frequency of the NCO is given as:

$$f_{out} = f_c \phi / 2^n \tag{4.3}$$

The NCO generates a square wave whose frequency is controlled by the error. The NCO forms an important part of digital receiver as it is used in the timing recovery. As described above, it is used to change the frequency of the clock, thereby, the clock timing.

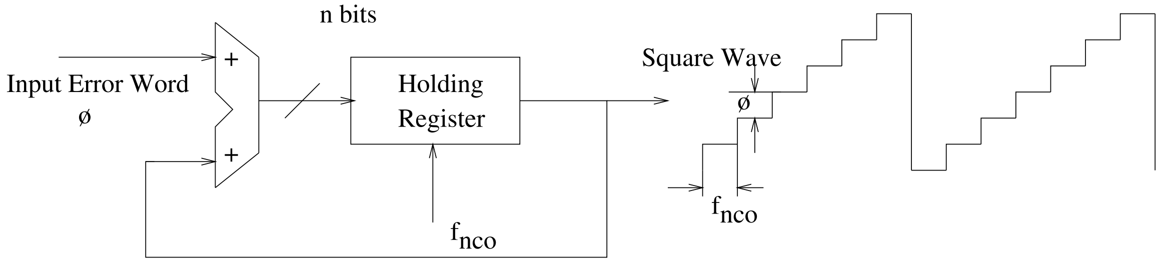


Figure 4.3. Numerical Controlled Oscillator

```
Algorithm NCO
  while sum < threshold do
    sum $\Leftarrow$ sum + $\phi$
    if sum > threshold then
      change input error word $\hat{\phi}$
    else
      sum $\leftarrow$ sum + $\hat{\phi}$
    end if
  end while
```

4.1.3.1 Code NCO

A code NCO is used in the code tracking loop. It is used to generate a clock signal, as well as the prompt signal, based upon the Early minus Late signal obtained by correlating the early and late codes.

4.1.3.2 Carrier NCO

The carrier NCO is used in the carrier tracking loop, to track the phase of the incoming carrier, by changing the phase of the local signal. The carrier tracking is done using a Costas loop. Based on the carrier discriminator, the error signal is generated [26]. This error signal is used to increase or decrease the phase of the locally generated signal, so that it matches with the incoming signal.

4.1.4 Direct Digital Frequency Synthesizer

To produce sine and cosine waves digitally one of the approaches used is the look up table method. It is an extension of the NCO. The NCO generates a square waveform. The amplitude is converted to a corresponding sine wave. A DDFS is shown in Figure 4.4. The k MSBs of the accumulator register of the NCO are used to address a lookup table which contains an n bit precision sine and cosine values. One can generate a sine wave by storing values in the lookup table for one complete cycle. This requires large ROM. Hence, another technique used is to store only one quarter of the cycle and since the other cycles are mirror images, one can appropriately obtain those values. The incoming digital data is modulated by the carrier and code. The demodulation process at the receiver involves carrier stripping initially. For this, one needs the sine and cosine values which can be generated using this module.

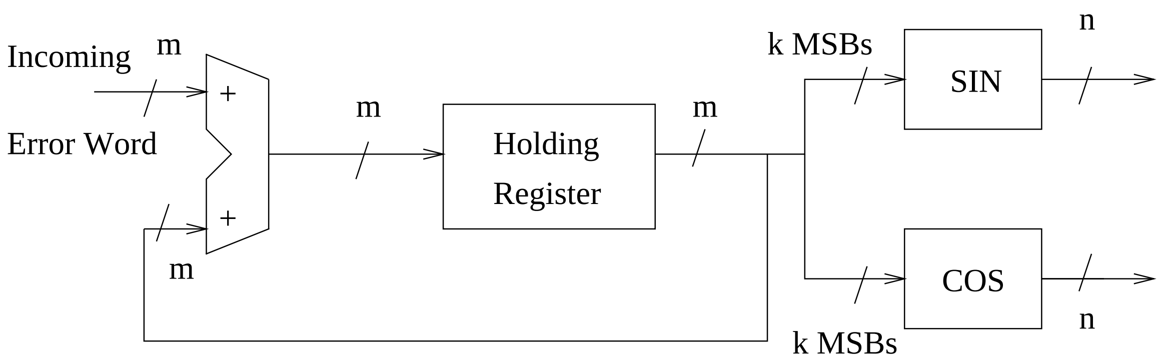


Figure 4.4. Direct Digital Frequency Synthesizer

The VHDL code for the digital frequency synthesizer to generate a 1.25 MHz signal is presented.

4.1.5 Digital IF Generation

For simulation purposes, we generated an IF carrier using a DDFS described in section on DDFS. The pseudo-noise code, generated, modulates this IF carrier. The pseudo-noise code is generated using two LFSBs. The block diagram of the transmitter used is shown in Figure 4.5.

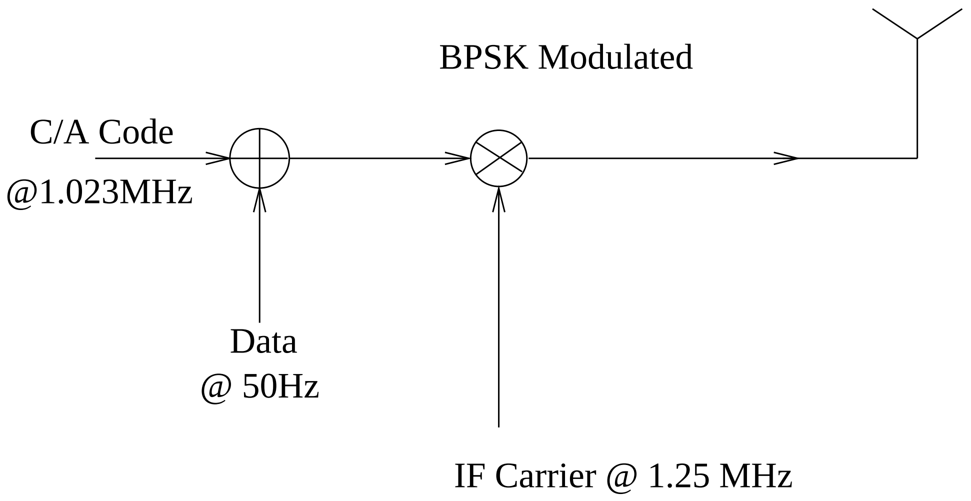


Figure 4.5. IF GPS Signal Generation

4.1.6 Correlators and Matched Filters

These constitute the most important function of the demodulation and detection process in a receiver. A serial correlator is shown in Figure 2.5(a). The digital correlator tries to match the codes by multiplying them and summing the result by and accumulate and dump. The block diagram is shown in Figure 4.6. The following algorithm describes the functionality of the correlator implemented.

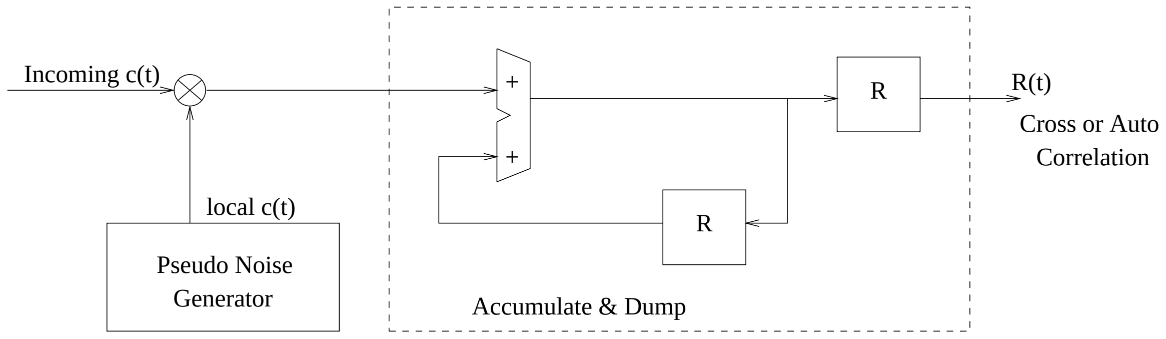


Figure 4.6. Digital Implementation of Serial Correlator

```
Algorithm: Correlation
  if count $<$ N then
    sum $\leftarrow$ sum + local-data $\times$ incoming
  else
    corr-value $\leftarrow$ sum
  end if
```

4.1.7 Control Signal Generation

The control signals have to be generated to control the acquisition, change from acquisition phase to tracking phase, etc. The VHDL design of the acquisition assumes a known carrier frequency and phase. Hence, a coherent signal demodulation was designed. Clock timing is also assumed. This assumption leads to a knowledge of the bits at every clock. The code acquisition problem then translates to a one-dimensional problem of searching the phase given the carrier frequency and which satellite code to search for. Correlation is the process behind acquisition of the signal. The acquisition process can be stated as an algorithm as follows:

```
Algorithm: Phase Changes
  if reset = 0 then
    state $\leftarrow$Reset
```

```
else
   state ← Acquisition
end if
while state ← Acquisition do
   if sum ≤ threshold then
      state ← Acquisition
   else
      state ← Tracking
   end if
end while
```

As described in the algorithm above the state changes after the receiver has been switched on. If the receiver is in acquisition phase, then it goes to a tracking phase only after it has determined the code phase to within one chip timing. The serial correlation acquisition algorithm implemented, changes delays code in steps of half chip, until the locally generated code matches with the incoming code.

Algorithm: Acquisition

```
while STATE state ← begin acquiring do
   sum = sum + localdata × din
   if sum < threshold then
      delay clock by $T_c/2$
      state ← continue acquiring
   else
      state ← begin tracking
   end if
end while
```

Code tracking involves the use of a delay locked loop. In the behavioral design of the delay locked loop, early and late codes are generated based on a clock signal which generates the prompt code. This phase as described earlier, starts once we have the prompt code aligned to within a chip time of the incoming code. The early and late codes are correlated with the incoming signal and the correlation values squared to get the absolute values. From these, an error signal is generated. Based on the error signal, we have to advance the clock such that the new early, and late correlation values produce a prompt code which is within half-a-chip time alignment with the incoming code.

Algorithm: Tracking

```
while STATE state $\Leftarrow$ continue tracking do
  correlate early and late codes
  square the correlation values
  generate early - late correlation values
  if Early< Late then
    shift clock by $T_c/2$
    state $\leftarrow$ continue tracking
  else
    prompt code is near incoming code
  end if
end while
```

4.1.8 Multiple Satellite Tracking

A receiver typically receives more than one satellite signal at a time. Though these signals have same frequency, they have different codes. These codes are orthogonal and the cross correlation between the codes themselves is very small. When acquiring different codes, each channel in a receiver looks for a particular code from the signal transmission. If there are four satellites transmitting data, then the problem of acquisition is that each channel has to detect the carrier frequency and code phase. Assuming that the carrier frequency is known, it translates to code phase determination of multiple codes in each channel.

Algorithm: Multiple Code Acquisition

```
if reset = 0 then
  state $\leftarrow$ begin acquisition
end if
while STATE state $\leftarrow$ begin acquisition do
  for channel in 0 to n do
    for satellite = 0 to K do
      state $\leftarrow$ Acquire
      if $sum_{channel,satellite}$ > threshold  then
        $satellite_{satellite}$ is acquired
      else
        change to different satellite
      end if
    end for
  end for
end while
```

4.1.9 GPS Data Generator

As described in the chapter on GPS signals, the carrier is modulated by both the pseudo-random code, as well as, the data. This data is generated at 50 Hz. Figure 4.7 shows how the data is generated.

4.2 Summary

The present chapter presented the behaviour VHDL implementations of various communication components used in a GPS receiver. Valid assumptions were described which were used in the design. The next chapter presents the experimental results obtained by simulating the above modules.

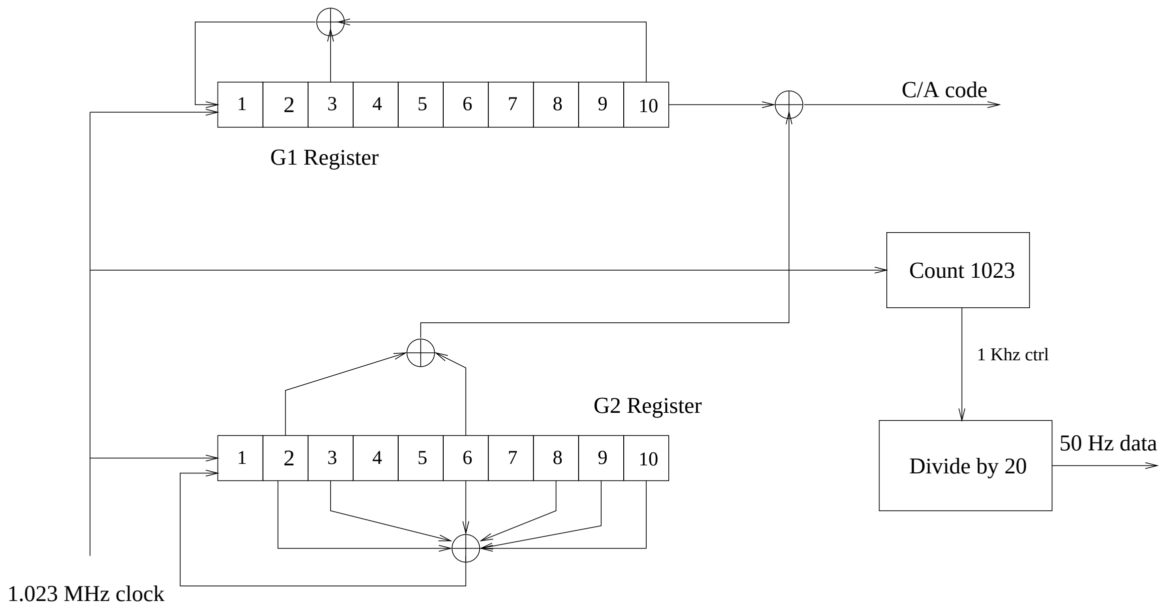


Figure 4.7. Data Generation

CHAPTER 5

EXPERIMENTAL RESULTS

Various communication blocks have been implemented as cores which could be used in the design of different communication systems. The design of the digital signal processing in a GPS receiver is implemented. The design included the different communication modules listed in chapter 4. Behavioral code for these modules was written and explained in chapter 4 and the corresponding waveforms are presented in this chapter. A coherent signal demodulation was assumed during the processing of the signal. By this we understand that the code is known and the carrier phase is known.

5.1 Communication Blocks Simulations

Figure 5.1 shows the sine and cosine waveforms obtained from the simulation of the DDFS. The DDFS was designed using a lookup table technique. The corresponding numerically controlled oscillator output is also represented by signal *holdsum*. We can observe that the NCO output is a square wave, the step size being the incoming phase error. As shown in Figure 5.1, once the threshold is crossed in the accumulator, it is reset and the sine wave repeats. The frequency of the signal generated is 1.25MHz which is the IF for the system under study. The carrier is a binary phase shift keyed signal modulated by the pseudo-noise code and a data signal. Figure 5.2 shows the sine wave modulated by the C/A code. As we can observe in Figure 5.2, a bit 0 has zero phase, while for a bit 1, there is a change of phase. The amount of phase change is π radians, for a data corresponding to bit 1, and $-\pi$ radians for a bit 0. As shown, the waveform matches the original sine wave when there is a bit 0 being transmitted. The Integrate and dump as described before is used in correlation. The correlation process, used in the coherent demodulation, is a serial

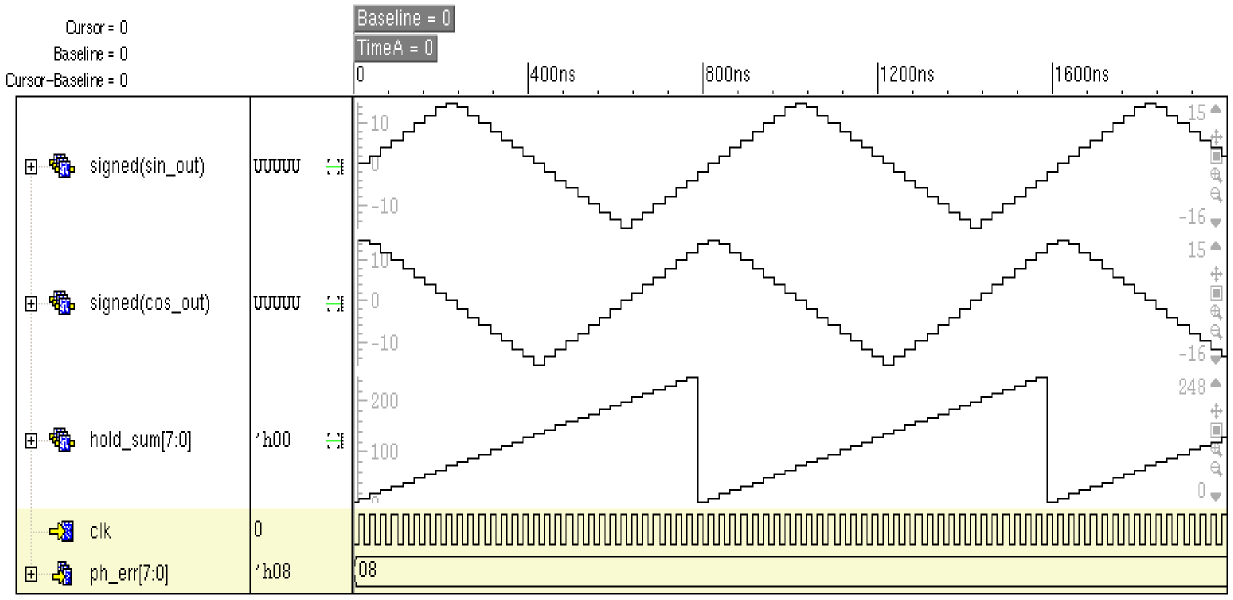


Figure 5.1. Sine Wave Generation

method. Figure 5.3 shows the auto correlation of satellite 1, *c_a*, with a local signal, *recca*. The signal *intsum*, represents the correlation sum for the integration period of 1 ms. The integration time for a strong satellite signal can be assumed to be 1 ms [35]. If both the c_a and recca are equal, the resulting sum is maximum. If the data bit is '0', the sum is positive maximum, while for a '1', the sum is a negative minimum. The intsum is accumulated for this period and dumped at the end of it resulting in the saw waveform as shown.

5.2 GPS Signal Simulation

As discussed in the chapter on GPS signals, the data is a 50 Hz signal, synchronous with the C/A code. Synthetic data is generated for testing the model designed. The waveform in Figure 5.4 shows a data signal(*din*) which is synchronous with the code (signal *c_a*). Figures 5.5, 5.6, 5.7, 5.8, 5.9, and 5.10 represent the codes for different satellites. These are obtained by using different delays from the G2 register.

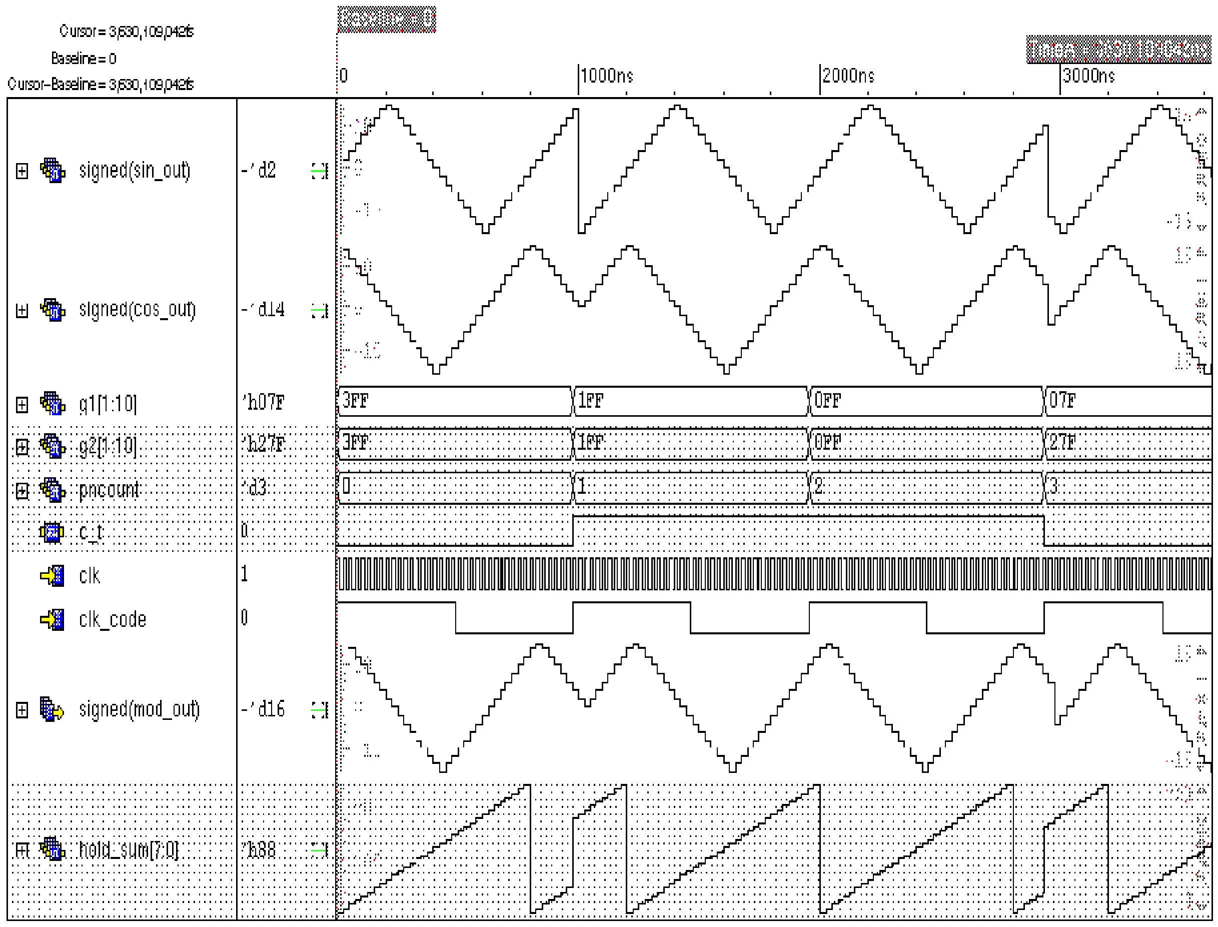


Figure 5.2. Binary Phase Shift Keying Signal

The major process involved in demodulation is the correlation of the incoming signal with the local signal. A sliding correlator has been implemented in this design for correlation purposes. It consists of a multiplier and an accumulate and dump as described in chapter 4. A generated signal is said to be correlated if the threshold of the accumulate and dump exceeds a maximum value. In this design, a correlation integration time of 1 ms was used and the maximum sum of ± 1022 was checked for detecting the correlation. The signal *int_sum* represents the correlation sum which is checked, if threshold is exceeded or not. This process is the basis of the two problems solved next: acquisition and tracking. Knowing

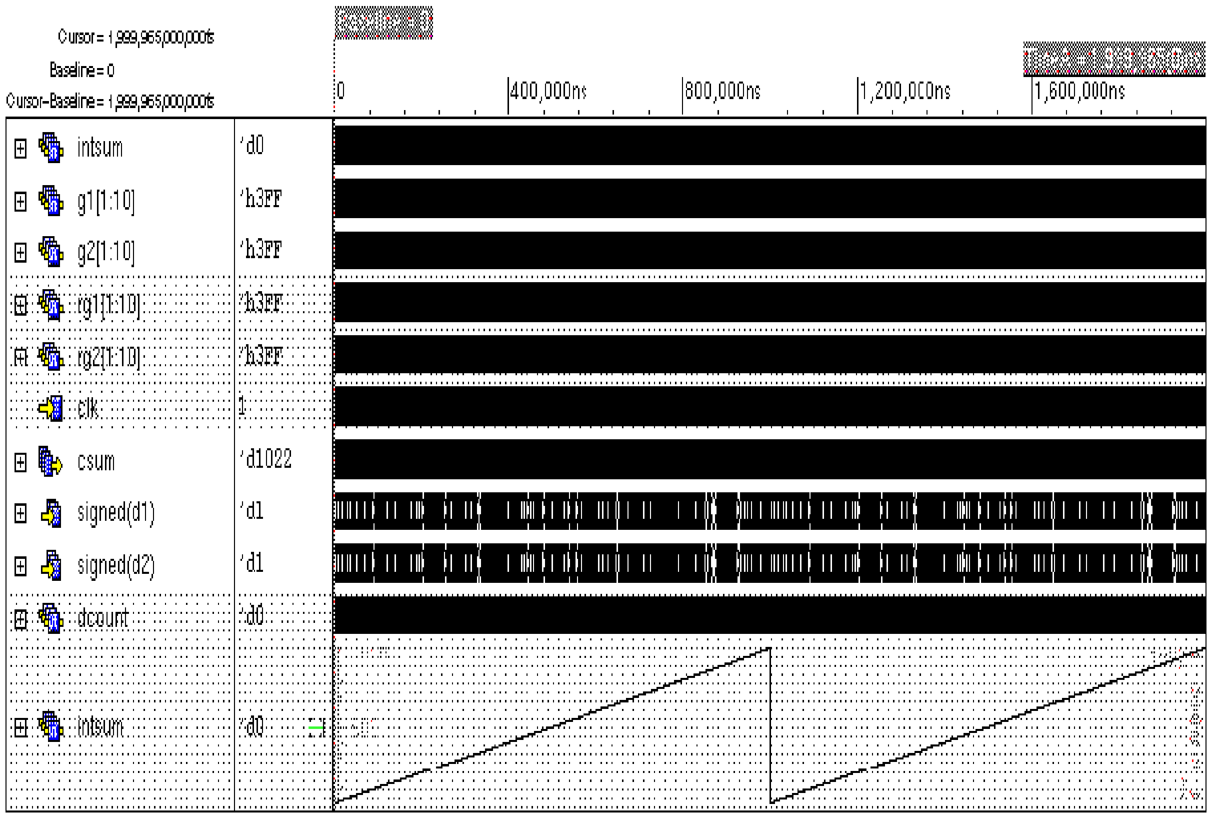


Figure 5.3. Auto Correlation using Accumulate and Dump

that data from a satellite is being received, it has to be synchronized with a local signal to demodulate the incoming data. Figure 5.11 represents acquisition of a satellite with ID 1.

The acquisition is achieved by shifting the code generation every half chip time as shown. The control signal *shiftclk* determines, if there is any phase change from acquisition to tracking or if the state remains constant depending upon the value of the correlation sum. If the *int_sum* value exceeds the threshold value, the code is generated in the same frequency and phase, otherwise the phase of the code is delayed by half chip time.

If *shiftclk is equal to lt*, it implies that the signal has not been acquired and is acquired if the *shiftclk is equal to eq*. Once the signal has been acquired, the incoming signal is within one chip time of the local signal as has been shown in Figure 5.11. Once the signal has been

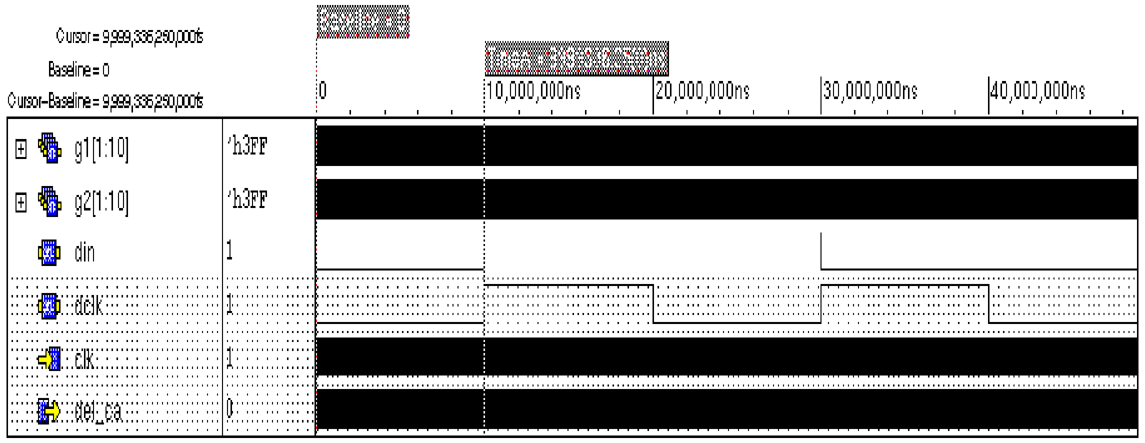


Figure 5.4. Data Generation

acquired, to detect the data signal, we need to know precisely when the signal changes. This is achieved by designing an Early-Late tracking loop. The tracking loop samples the incoming signal at or very close to the original incoming signal by correlating the early and late codes. The *early_code* and *late_code* in Figure 5.12 represent the early and late codes. The tracking phase is entered when *shiftclk is equal to eq* and the *recca* signal corresponds

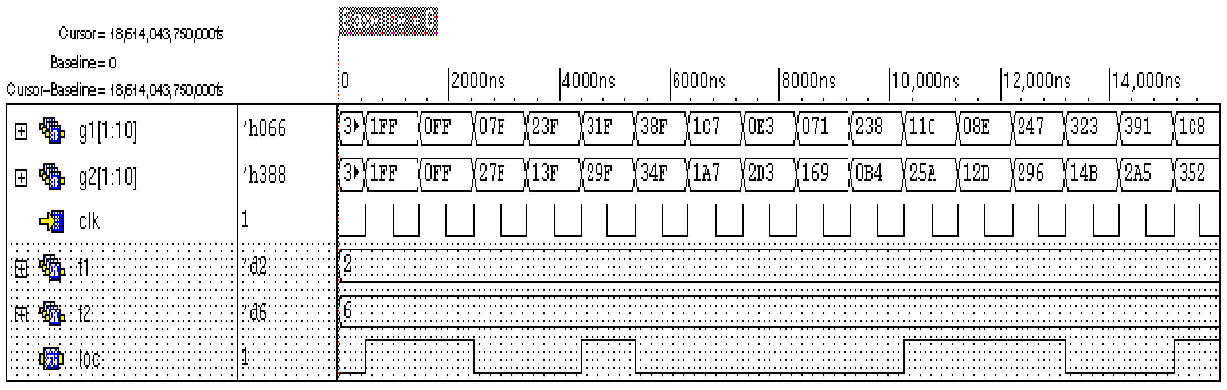
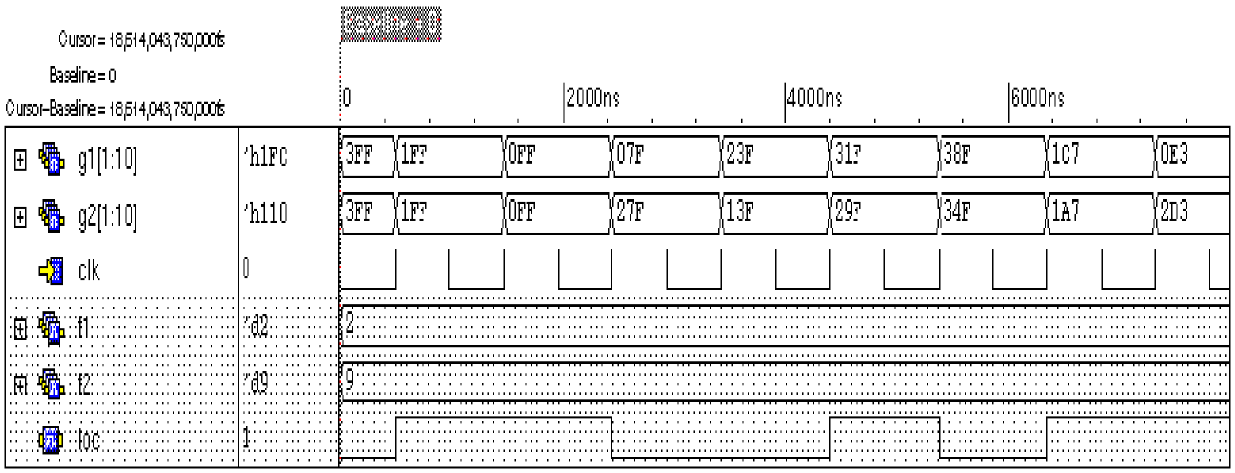

Cursor = 18,614,043,760,000fs
Baseline = 0
Cursor-Baseline = 18,614,043,760,000fs
0
2000ns
4000ns
6000ns
8000ns
10,000ns
12,000ns
14,000ns
g1[1:10]
'h066
3 1FF 0FF 07F 23F 31F 38F 1C7 0E3 071 238 11C 08E 247 323 391 1C8
g2[1:10]
'h388
3 1FF 0FF 27F 13F 29F 34F 1A7 2D3 169 0B4 25A 12D 296 14B 2A5 352
clk
1
f1
'd2
2
f2
'd6
6
loc
1

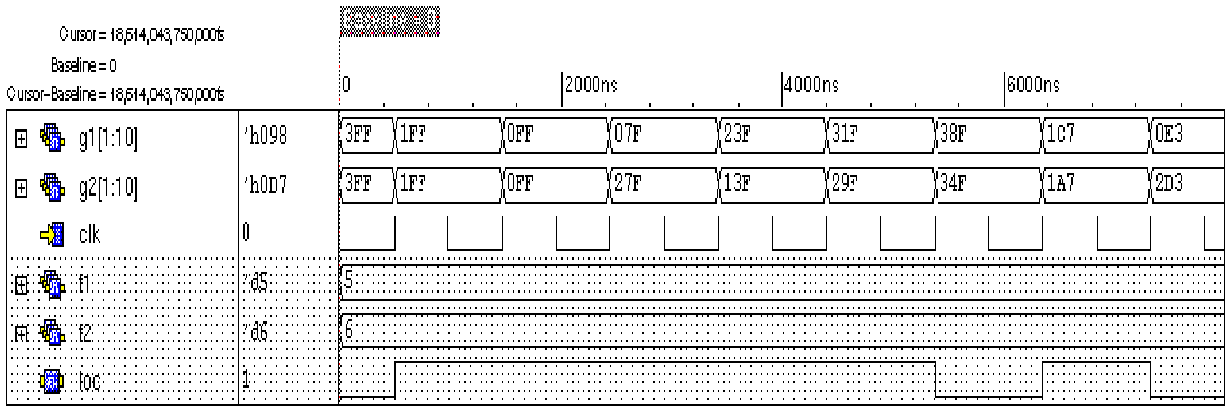
Figure 5.5. C/A Code for Satellite 1



Cursor = 18,514,043,750,000fs
Baseline = 0
Cursor-Baseline = 18,514,043,750,000fs
0
2000ns
4000ns
6000ns
g1[1:10]
'h1FC
3FF 1FF 0FF 07F 23F 31F 38F 1C7 0E3
g2[1:10]
'h110
3FF 1FF 0FF 27F 13F 29F 34F 1A7 2D3
clk
0
f1
'd2
2
f2
'd9
9
loc
1

Figure 5.6. C/A Code for Satellite 8

to the local receiver prompt code. By shifting the phase of the clock by half a chip, a new early and late code phases are generated and correlated with the incoming signal. To generate an error signal, the absolute correlation values of the early and late codes must be known. The correlation values are squared to obtain this and an error signal is generated based on whether the early correlation value is greater than the late correlation value. If



Cursor = 18,514,043,750,000fs
Baseline = 0
Cursor-Baseline = 18,514,043,750,000fs
0
2000ns
4000ns
6000ns
g1[1:10]
'h098
3FF 1FF 0FF 07F 23F 31F 38F 1C7 0E3
g2[1:10]
'h0D7
3FF 1FF 0FF 27F 13F 29F 34F 1A7 2D3
clk
0
f1
'd5
5
f2
'd6
6
loc
1

Figure 5.7. C/A Code for Satellite 12

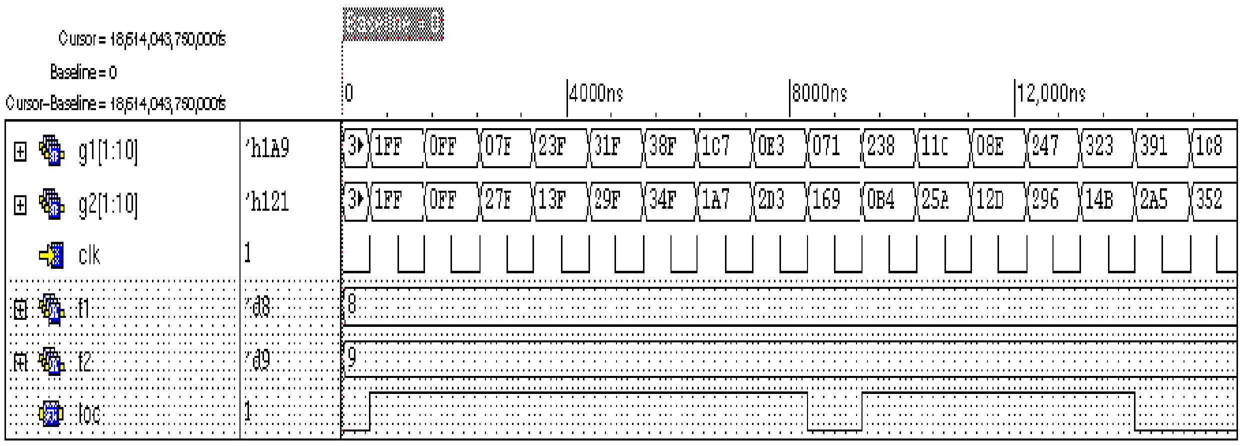

Cursor = 18,614,043,750,000fs
Baseline = 0
Cursor-Baseline = 18,614,043,750,000fs
0
4000ns
8000ns
12,000ns
g1[1:10]
'h1A9
3 1FF 0FF 07F 23F 31F 38F 1C7 0E3 071 238 11C 08E 247 323 391 1C8
g2[1:10]
'h121
3 1FF 0FF 27F 13F 29F 34F 1A7 2D3 169 0B4 25A 12D 296 14B 2A5 352
clk
1
f1
'd8
8
f2
'd9
9
toc
1

Figure 5.8. C/A Code for Satellite 15

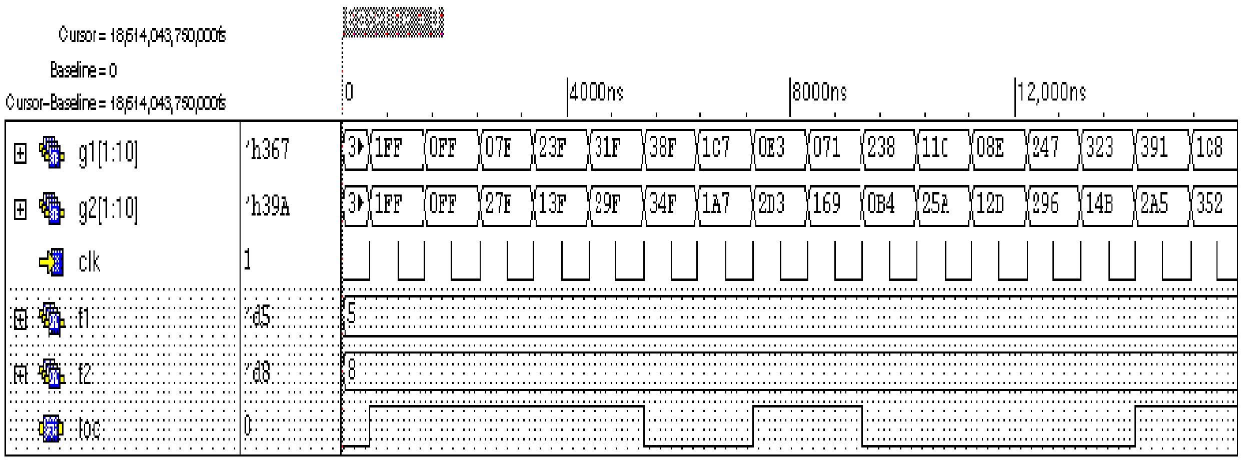

Cursor = 18,614,043,750,000fs
Baseline = 0
Cursor-Baseline = 18,614,043,750,000fs
0
4000ns
8000ns
12,000ns
g1[1:10]
'h367
3 1FF 0FF 07F 23F 31F 38F 1C7 0E3 071 238 11C 08E 247 323 391 1C8
g2[1:10]
'h39A
3 1FF 0FF 27F 13F 29F 34F 1A7 2D3 169 0B4 25A 12D 296 14B 2A5 352
clk
1
f1
'd5
5
f2
'd8
8
toc
0

Figure 5.9. C/A Code for Satellite 21

the correlation sum of the early code is smaller than the correlation sum of the late code, then the code is delayed. Otherwise, the code is said to be tracking the incoming code. As shown, *din* is within half chip time of *recca*.

The *recca* signal is the nearest replica of the incoming signal. The data is detected from this signal. This has been illustrated by *data_out*. If the sum is greater than 1020, the data is detected as 0 and if the sum is less than -1020, the data is detected as a 1. Modern GPS

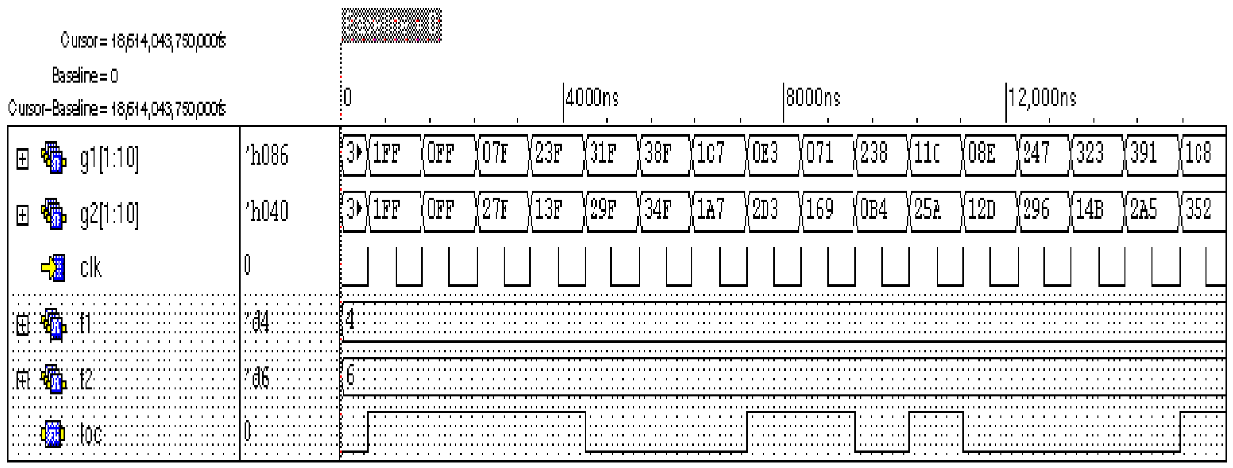


Figure 5.10. C/A Code for Satellite 24

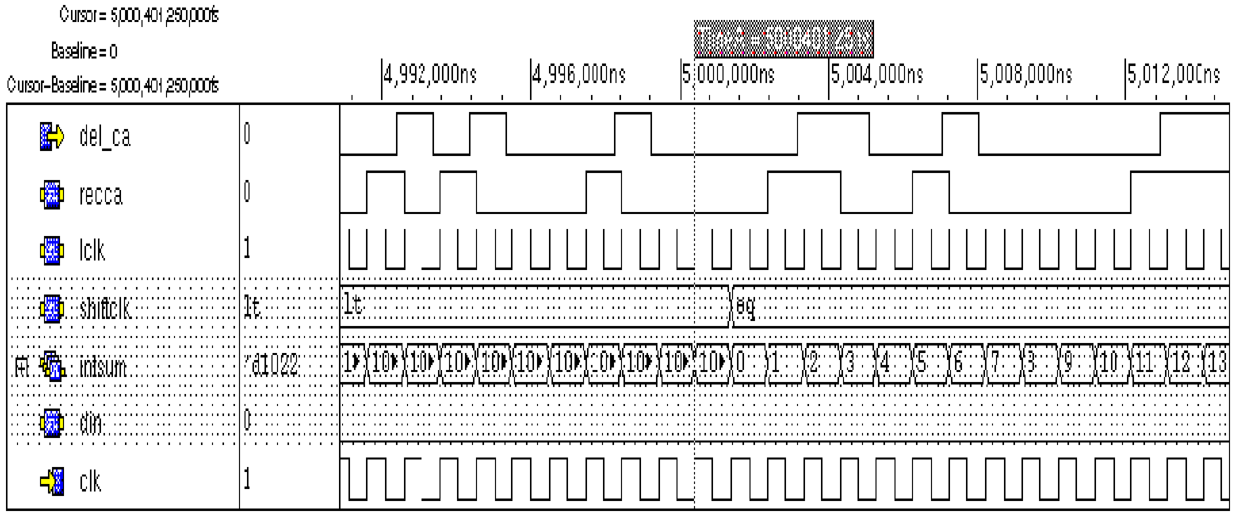


Figure 5.11. Acquisition of a Single Satellite by Incrementing Code Phase in Half Chip Increments

receivers process data from multiple satellites in parallel. This can be achieved by either having a single channel multiplexing the satellites or having multiple channels with each of them detecting only one satellite at a time. The incoming signal at a GPS receiver is made

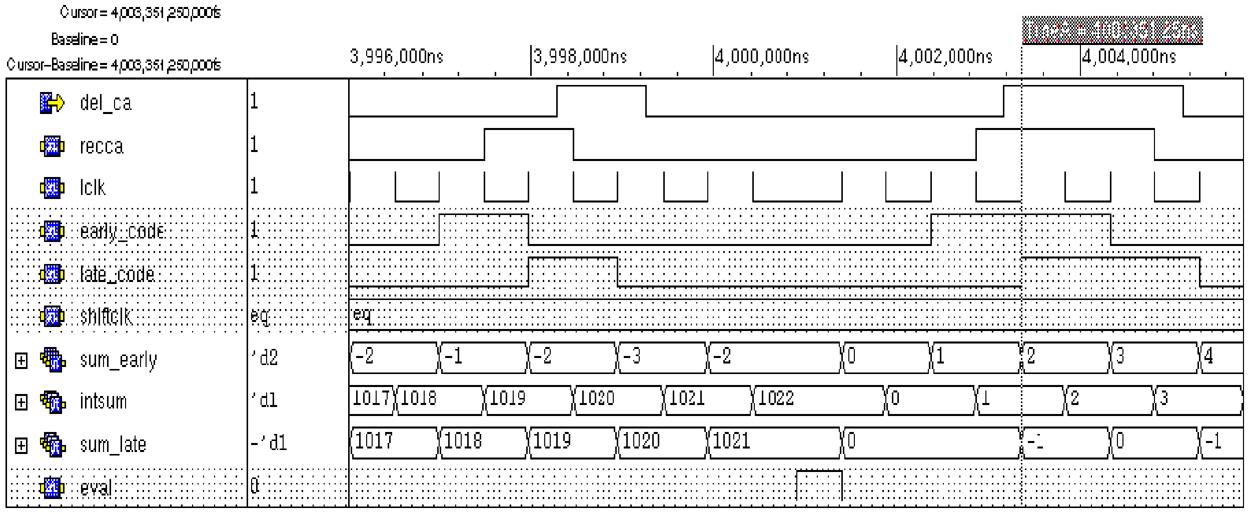


Figure 5.12. Tracking After Signal has been Acquired

up of codes from multiple satellites, transmitting at different frequencies. As the cross-correlation properties of the C/A codes are such that, they have litter interference, one can assume that each channel can correlate with multiple GPS satellite signals. The correlation which results in the greatest correlation sum is the satellite which is being locally generated. Figures 5.13, 5.14, 5.15, 5.16, 5.17, and 5.18 illustrate the case where four satellite signals are being acquired by a four channel GPS receiver. Each channel has four correlators one for each satellite. When *shiftclk is equal to eq*, it indicates that the signal has been acquired and the next phase of tracking could start.

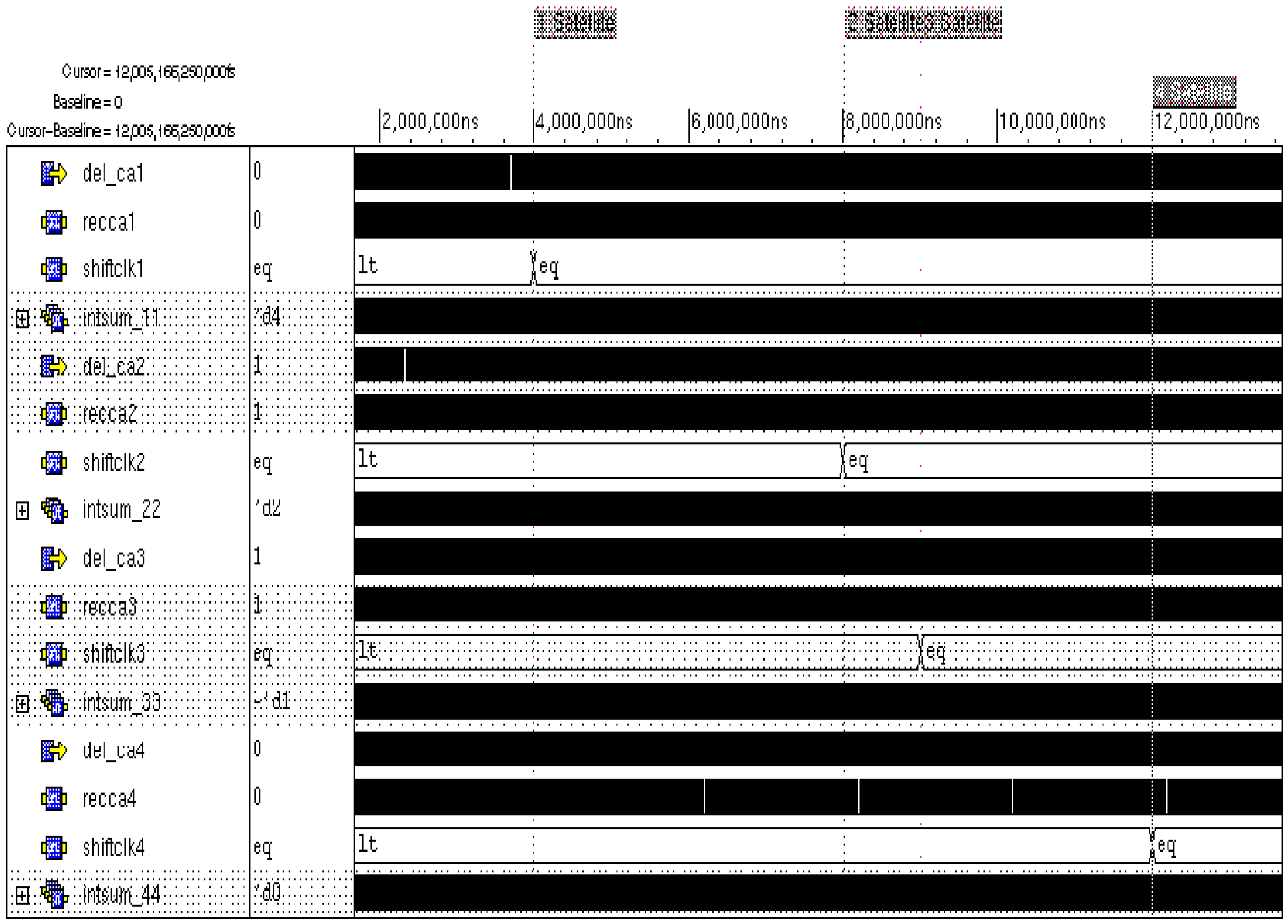


Figure 5.13. Acquiring Multiple Satellites Set 1

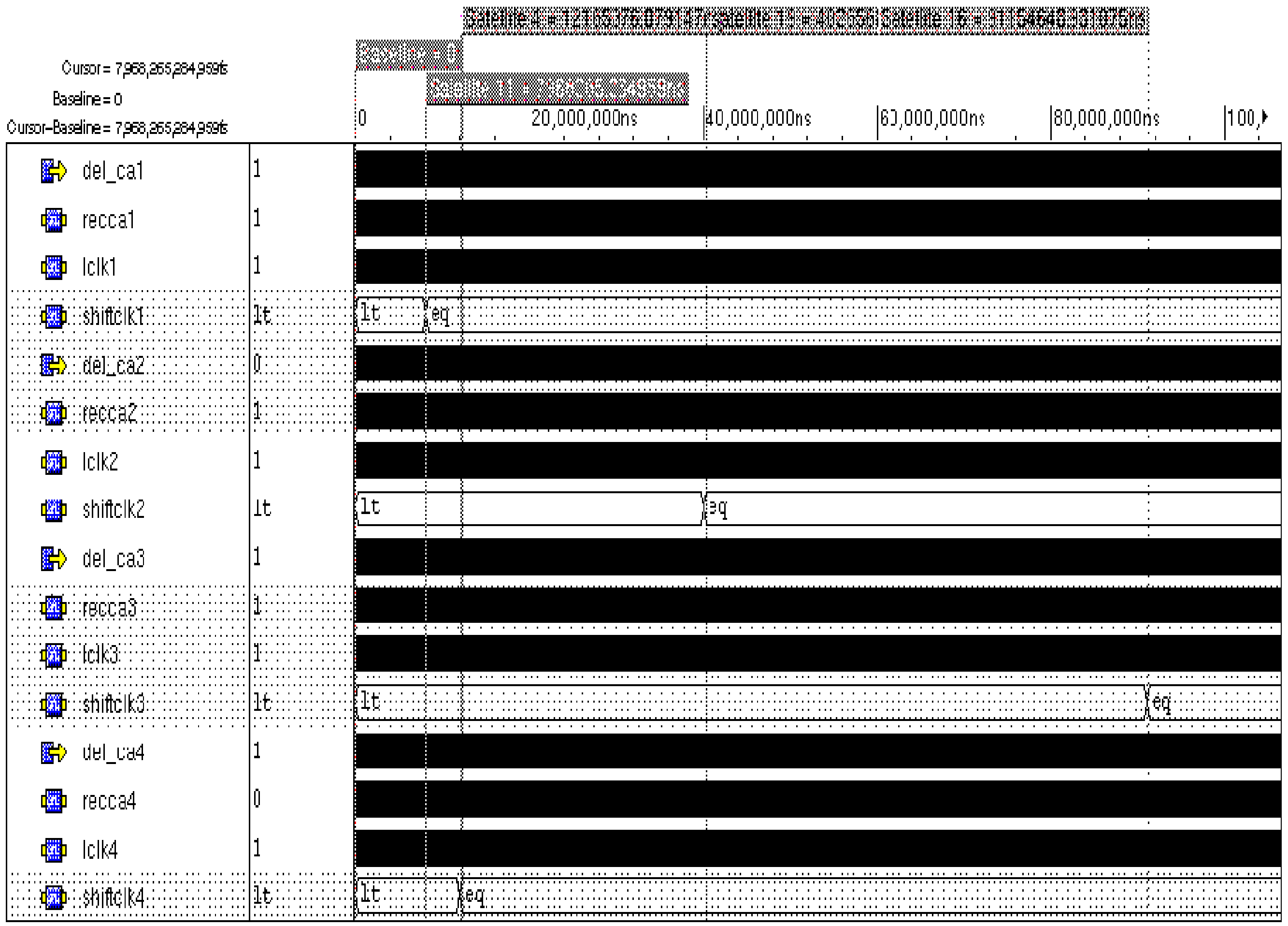


Figure 5.14. Acquiring Multiple Satellites Set 2

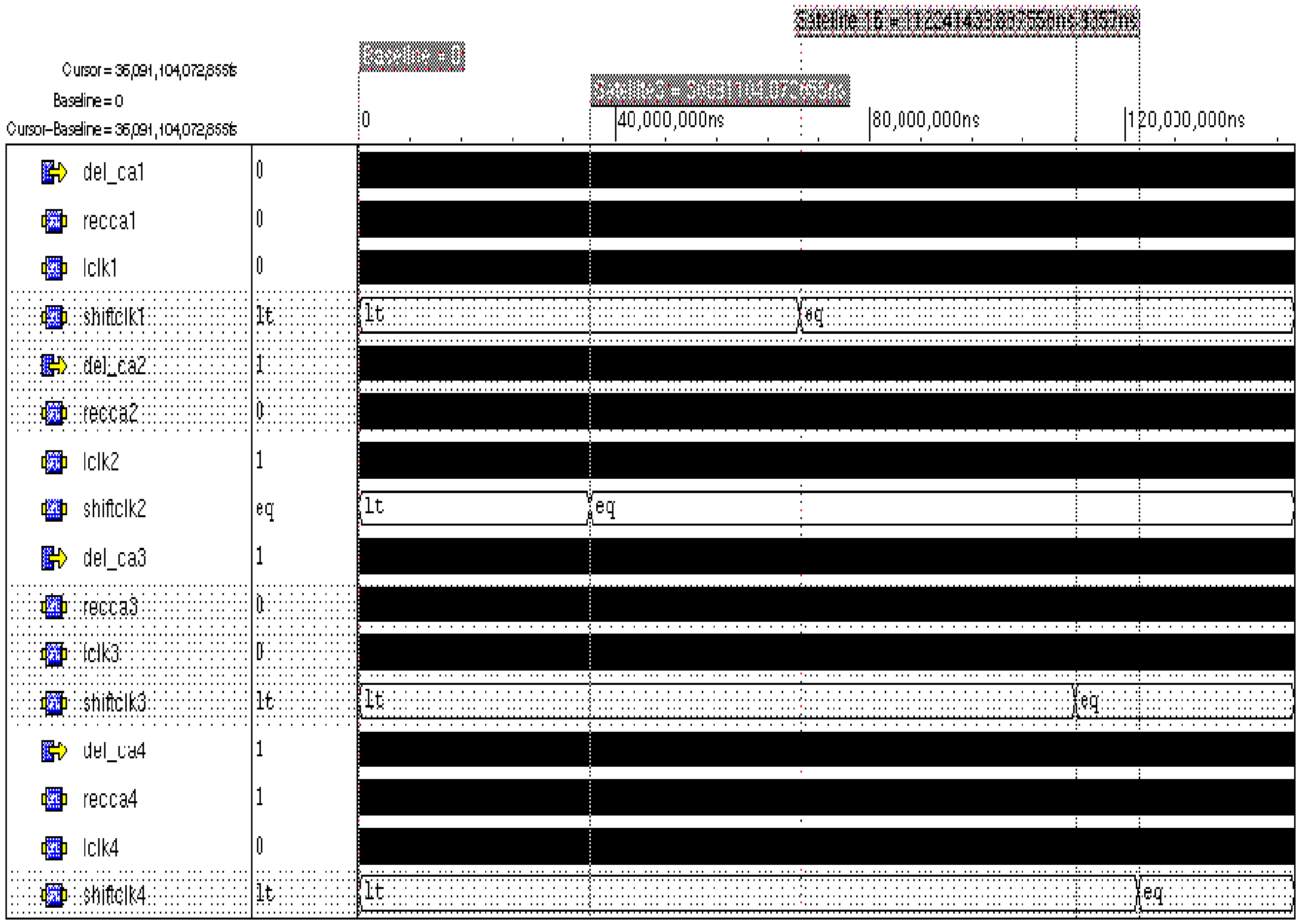


Figure 5.15. Acquiring Multiple Satellites Set 3

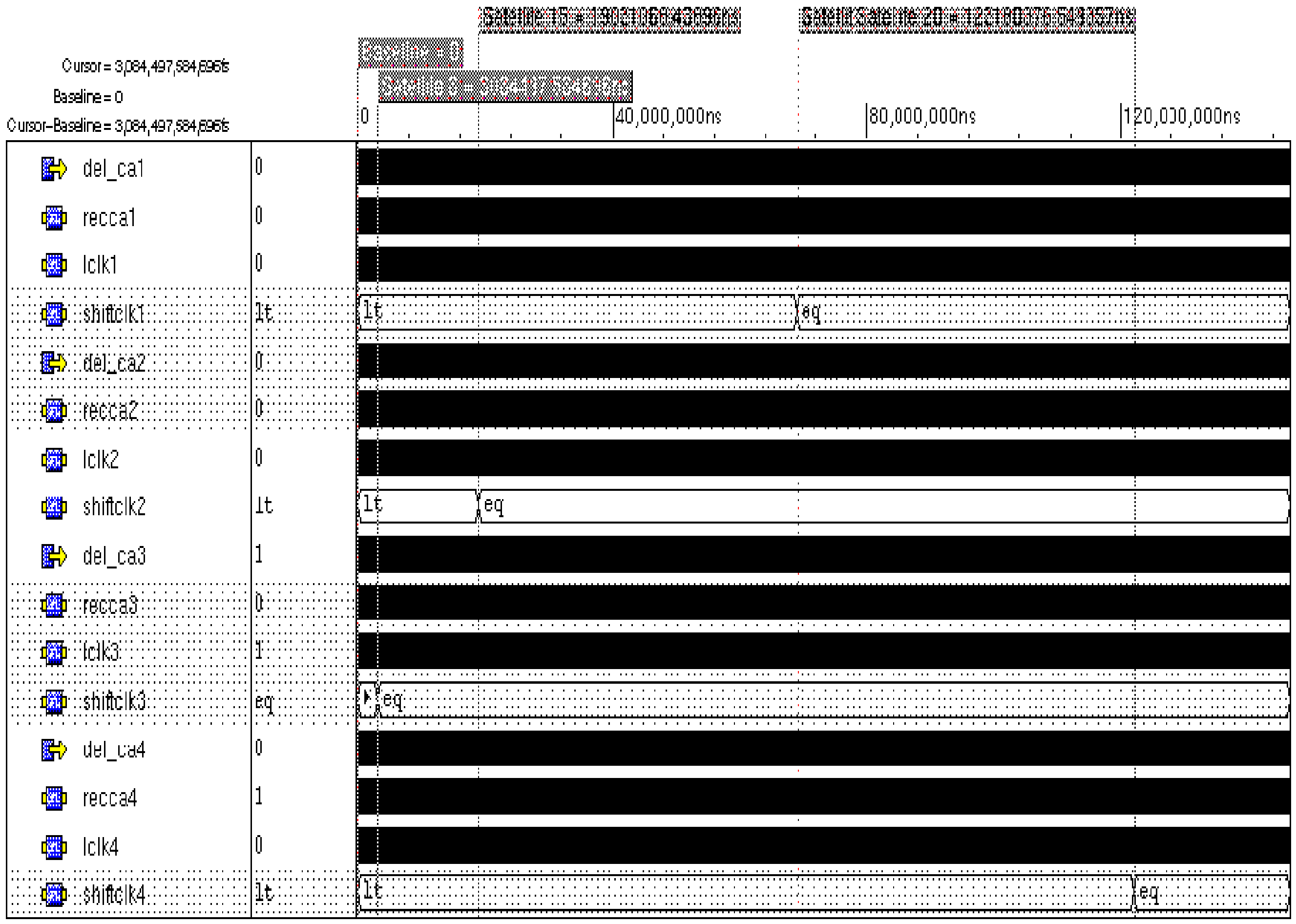


Figure 5.16. Acquiring Multiple Satellites Set 4

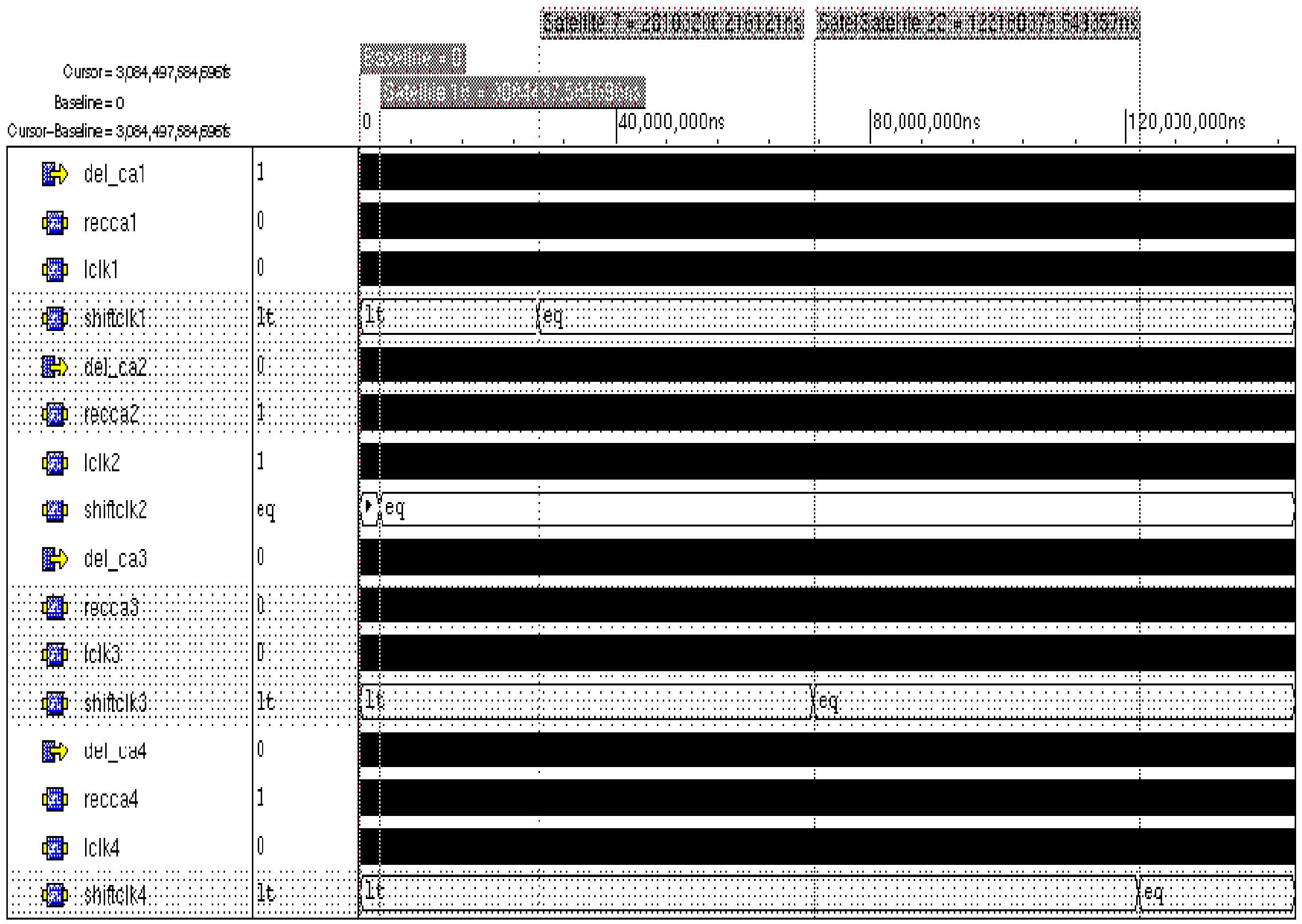


Figure 5.17. Acquiring Multiple Satellites Set 5

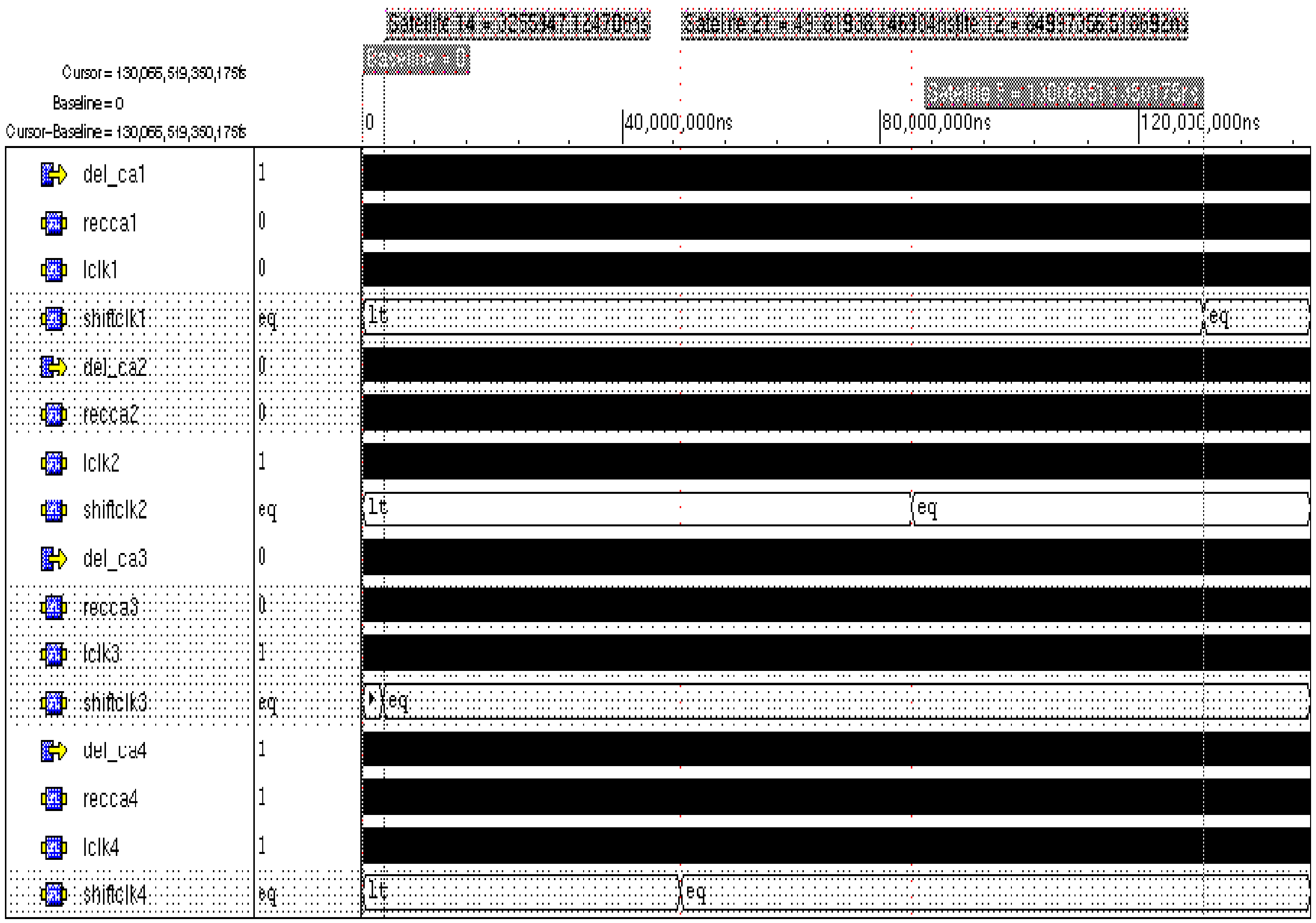


Figure 5.18. Acquiring Multiple Satellites Set 6

5.3 Summary

Behavioral VHDL codes for communication blocks were written. The experimental waveforms are shown for each of the components and functional blocks of the digital back-end GPS receiver. To account for atmospheric effects and other cumulative effects the generated data was delayed. It was seen that the desired data is obtained at the receiver after demodulation and detection. Results for acquiring data from multiple satellites was also presented. A four channel parallel receiver was simulated.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This work outlined the implementation of a GPS receiver in time domain. It dealt with VHDL implementation of the digital backend of a GPS receiver. Different functional blocks and communication blocks were implemented as part of this work. The scope of this work, was to develop a working code acquiring and tracking module, capable of acquiring a GPS signal and tracking it. Synthetic data was generated at the required rate and modulated the PRN sequence. This transmitted data was demodulated and detected and the expected data was recovered. Thus, a DS/SS receiver was implemented, in time domain, capable of acquiring and tracking a GPS C/A code signal. The receiver implementation assumed a coherent signal acquisition and tracking. This work also dealt with acquiring codes from multiple satellites. It used a dedicated channel for each of the satellites being tracked. Four satellites were continuously being acquired.

For this to be integrated as an independent module, the carrier acquisition has to be performed alongwith the code acquisition. This module has to be tested on original GPS data to validate it. The entire model has to be synthesized, to be used in conjunction with the tour guide being developed. Low power modes and functionalities have to be incorporated. Newer algorithms to speed up the acquisition times in the time domain could be worked upon. Acquiring data from a greater number of satellites and tracking them simulatneously is another aspect for future research. Integrating this module with the analog front end to achieve proper GPS functionality is a future work. Finally, developing algorithms for using the GPS receiver indoors is an aspect of future research.

REFERENCES

[1] P. Misra and P. Enge. *"Global Positioning System: Signals, Measurements and Performance"*. Ganga-Jamuna Publishers, 2001.

[2] Micosoft Press Pass. Microsoft unveils first connvected concept cars. http://www.microsoft.com/presspass.

[3] P. Enge. Next Generation GPS Receiver. http://www.sciam.com/.

[4] Press Release. Single chip GPS Module. http://www.valence.com/.

[5] Instant GPS Product Documentation. Single chip GPS Receiver. http://www.motorola.com/ies/GPS/products.html.

[6] WirelessDevNet.com Press Release. SONY Introduces Industry's First 1-Chip CMOS GPS. http://www.wirelessdevnet.com/news/2003/dec/15/news1.html.

[7] J. Ashjaee. GPS: Challenge of Single Chip. http://www.gpsworld.com.

[8] D.M. Akos. *A Software Radio Approach to Global Navigation Satellite System Receiver Design*. PhD thesis, Ohio University, 1997.

[9] G. Martin and H. Chang. "Tutorial 2 System-on-Chip Design". In *Processings of the 4th International Conference on ASIC*, pages 12–17, Oct 2001.

[10] U. Kailasam. High Level VHDL Modeling of Low Power ASIC for a Tour Guide. Master's thesis, University of South Florida, 2004.

[11] W. Zhuang and J. Tranquilla. "Digital Baseband Processor for the GPS Receiver Modeling and Simulations". *IEEE Transactions on Aerospace and Electronic Systems*, pages 1343–1349, Oct 1993.

[12] M.S. Braasch and A.J. van Dierendonck. "GPS Receiver Architectures and Measurements". In *Proceedings of the IEEE*, pages 48–64, Jan 1999.

[13] D.J.R. van Nee and R.J.R.M. Conen. "New Fast GPS Acquisition Techniques using FFT". *Electronic Letters*, pages 158–160, Jan 1991.

[14] Z. Zhen. Averaging Correlation for Weak Signal Global Positioning System Signal Processing. Master's thesis, Ohio University, 2002.

[15] A.A. Alaqeeli. *Global Positioning System Signal Acquisition and Tracking using Field Programmable Gate Arrays*. PhD thesis, Ohio University, 2002.

[16] U. deHaag. *Block Processing of GPS Signals*. PhD thesis, Ohio University, 1999.

[17] A. Fridman and S. Semenov. "Architectures of Software GPS Receivers". *GPS Solutions*, pages 58–64, 2000.

[18] S.P. Powell B.M. Ledvina and P.M. Kintner. "A 12 channel Real Time GPS L1 Software Receiver". 2000.

[19] S.M. Krishna and B.R. Madhukar. "Digital Signal Processors in GPS Receivers". *GPS Solutions*, pages 67–71, 2000.

[20] J. B-Yen Tsui. *"Fundamentals of Global Positioning System Receivers - A Software Approach"*. Wiley-Interscience, 2000.

[21] R.C. Dixon. *"Spread Spectrum Systems"*. John Wiley and Sons Inc, 1994.

[22] C. Chien. *"Digital Radio Systems On A Chip"*. Kluwer Academic Publishers, 2001.

[23] B. Sklar. *"Digital Communications- Fundamentals and Applications"*. Pearson Education, 2001.

[24] J.G. Proakis. *"Digital Communications"*. McGraw Hill, 2000.

[25] J.S. Lee, and L.E. Miller. *"CDMA Systems Engineering HandBook"*. Artech House Publishers, 1998.

[26] E.D. Kaplan. *"Understanding GPS Principles and Applications"*. Artech House Publishers, 1996.

[27] G. Sanjeev. Feasibility Study For Implementation of Global Positioning System Processing Techniques in Field Programmable Gate Arrays. Master's thesis, Ohio University, 2000.

[28] B.H. Wellenhof, H.Lichtenegger, and J. Collins. *"GPS Theory and Practice"*. Springer-Verlag, 2001.

[29] Interface Contrl Document published by US DoD. "GPS ICD 2000 Document".

[30] A. El-Rabbany. *"Introduction to The Global Positioning System"*. Artech House Publishers, 2002.

[31] D. Herskovitz. "Global Positioning System Receivers". *Microwave Journal*, pages 66–69, September 1994.

[32] J. Ceccherelli. Revolutionary GPS Receiver Chipset. http://www-3.ibm.com/chips/micronews/vol6no/ceccherelli.html.

[33] B.W. Parkinson, and J.J. Spilker Jr. *"Global Positioning System : Theory and Applications Volume 1"*. American Institute of Aeronautics and Astronautics, Inc., 1996.

[34] M.S. Roden. *"Digital Communication Systems Design"*. Prentice Hall, 1988.

[35] Y. Suh D. Manandhar and R. Shibasaki. "GPS Signal Acquisition and Tracking An Approach towards Development of Software based GPS Receivers". *Technical Report of IEICE*, 2004.

APPENDICES

Appendix A VHDL Code

Correlator

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_bit.all;


entity corr is

    port (d1 : in signed(2 downto 0);

          d2 : in signed(2 downto 0);

          clk : in std_logic ;

          rst : in std_logic;

          flag: out std_logic;

          csum : out integer);

end corr;

architecture corr_beh of corr is

   signal dcount, intsum : integer := 0;

begin  -- corr_beh

   process (clk, rst,d1, d2)

   begin  -- process

        -- activities triggered by asynchronous reset (active low)

        if rst = '0'then

            dcount <= 0;

            intsum <= 0;

        -- activities triggered by rising edge of clock

        elsif clk'event and clk = '1' then

                dcount <= dcount + 1;

                intsum <= intsum + to_integer(d1 * d2);
                if dcount = 1022 then
                    dcount <= 0;
                    intsum <= 0;
                    csum <= intsum;
                end if;
        end if;
    end process;
end corr_beh;
```

Code and Data Generation

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity del_codewdata is
     port (clk : in std_logic;
           rst : in std_logic;
           del_ca : out std_logic);
end del_codewdata;
architecture data_beh of del_codewdata is
    signal g1, g2 : std_logic_vector(1 to 10) := (others => '1');
    signal dcount,timecount : integer := 0;
    signal flag,din,dclk : std_logic := '0';
    signal count : integer := 0;
    constant t1 : integer := 2;
    constant t2 : integer := 6;
begin  -- del_beh
    process
        variable c_a : std_logic := '0';
    begin  -- process
        -- activities triggered by asynchronous reset (active low)
        if rst = '0' then
            count <= 0;
            c_a := '0';
            g1 <= (others => '1');
            g2 <= (others => '1');
            dcount <= 0;
            timecount <= 0;
            dclk <= '0';
            flag <= '0';
        end if;
        wait until clk'event and clk= '1';
        -- activities triggered by rising edge of clock
            g1(2 to 10) <= g1(1 to 9);
            g2(2 to 10) <= g2(1 to 9);
            g1(1) <= g1(3) xor g1(10);
            g2(1) <= g2(2) xor g2(3) xor g2(6) xor g2(8) xor g2(9) xor g2(10);
            c_a := g1(10) xor (g2(t1) xor g2(t2));
        -- introducting a delay in the signal to account for any effects
        if timecount < 10230 then
            flag <= '0';
        else
            flag <= '1';
        end if;
        if flag = '0' then
            wait for 0 ns;
        else
            wait for 400 ns;
        end if;
            del_ca <=  transport din xor c_a after 2.75 us;
            count <= count + 1;
```

```
            timecount <= timecount + 1;
        if count = 1022 then
            count <= 0;
            c_a := '0';
            g1 <= (others => '1');
            g2 <= (others => '1');
            dcount <= dcount + 1;
            if dcount = 9 then
                dclk <= not dclk;
                dcount <= 0;
            end if;
       end if;
    end process;
    -- data generation data is inverted every 20 ms
    process (dclk, rst)
    begin  -- process
    -- activities triggered by asynchronous reset (active low)
        if rst = '0' then
           din <= '0';
    -- activities triggered by rising edge of clock
           elsif dclk'event and dclk = '1' then
           din <= not din;
        end if;
    end process;
end data_beh;
```

DDFS Sine Generation and IF Generation

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity dig_synth2 is
     port (clk : in std_logic;
           clk_code : in std_logic;
           reset : in std_logic;
           ph_err : in unsigned(7 downto 0);
           mod_out : out signed(4 downto 0));
end dig_synth2;
architecture beh_synth2 of dig_synth2 is
    signal sin_out : signed(4 downto 0);
    signal cos_out : signed(4 downto 0);
    signal g1,g2 : std_logic_vector(1 to 10) := (others => '1');
    constant T1 : integer := 2;
    constant T2 : integer := 6;
    signal pncount : integer := 0;
    signal flag : std_logic := '0';
    signal c_t : std_logic := '0';
begin  -- beh_synth
    process (clk, reset,c_t)
        variable hold_sum : unsigned(7 downto 0) := (others => '0');
    begin  -- process
        -- activities triggered by asynchronous reset (active low)
        if reset = '0' then
            hold_sum := (others => '0');
        -- activities triggered by rising edge of clock
        else
           if  c_t'event and c_t = '1'then
               hold_sum := hold_sum - "10000000";
           elsif c_t'event and c_t = '0' then
               hold_sum :=  "10000000" + hold_sum;
           end if;
           if clk'event and clk = '1' then
              hold_sum := hold_sum + ph_err ;  -- when there is no change in data.
           case hold_sum(7 downto 3) is
             when "00000" => sin_out <= "00000";
             when "00001" => sin_out <= "00010";
             when "00010" => sin_out <= "00100";
             when "00011" => sin_out <= "00110";
             when "00100" => sin_out <= "01000";
             when "00101" => sin_out <= "01010";
             when "00110" => sin_out <= "01100";
             when "00111" => sin_out <= "01110";
             when "01000" => sin_out <= "01111";
             when "01001" => sin_out <= "01110";
             when "01010" => sin_out <= "01100";
             when "01011" => sin_out <= "01010";
             when "01100" => sin_out <= "01000";
             when "01101" => sin_out <= "00110";
             when "01110" => sin_out <= "00100";
             when "01111" => sin_out <= "00010";
             when "10000" => sin_out <= "00000";
             when "10001" => sin_out <= "11110";
             when "10010" => sin_out <= "11100";
             when "10011" => sin_out <= "11010";
             when "10100" => sin_out <= "11000";
             when "10101" => sin_out <= "10110";
             when "10110" => sin_out <= "10100";
             when "10111" => sin_out <= "10010";
             when "11000" => sin_out <= "10000";
             when "11001" => sin_out <= "10010";
             when "11010" => sin_out <= "10100";
             when "11011" => sin_out <= "10110";
             when "11100" => sin_out <= "11000";
             when "11101" => sin_out <= "11010";
             when "11110" => sin_out <= "11100";
             when "11111" => sin_out <= "11110";
             when others => null;
         end case;
         --generating cosine values
         case hold_sum(7 downto 3) is
             when "00000" => cos_out <= "01111";
             when "00001" => cos_out <= "01110";
             when "00010" => cos_out <= "01100";
             when "00011" => cos_out <= "01010";
```

```
                when "00100" => cos_out <= "01000";
                when "00101" => cos_out <= "00110";
                when "00110" => cos_out <= "00100";
                when "00111" => cos_out <= "00010";
                when "01000" => cos_out <= "00000";
                when "01001" => cos_out <= "11110";
                when "01010" => cos_out <= "11100";
                when "01011" => cos_out <= "11010";
                when "01100" => cos_out <= "11000";
                when "01101" => cos_out <= "10110";
                when "01110" => cos_out <= "10100";
                when "01111" => cos_out <= "10010";
                when "10000" => cos_out <= "10000";
                when "10001" => cos_out <= "10010";
                when "10010" => cos_out <= "10100";
                when "10011" => cos_out <= "10110";
                when "10100" => cos_out <= "11000";
                when "10101" => cos_out <= "11010";
                when "10110" => cos_out <= "11100";
                when "10111" => cos_out <= "11110";
                when "11000" => cos_out <= "00000";
                when "11001" => cos_out <= "00010";
                when "11010" => cos_out <= "00100";
                when "11011" => cos_out <= "00110";
                when "11100" => cos_out <= "01000";
                when "11101" => cos_out <= "01010";
```

```
             when "11110" => cos_out <= "01100";
             when "11111" => cos_out <= "01110";
             when others => null;
 end case;
         mod_out <= cos_out;
         end if;
       end if;
end process;
    process (clk_code, reset)
        begin   --process
         -- activities triggered by asynchronous reset (active low)
        if reset = '0' then
               g1 <= (others => '1');
               g2 <= (others => '1');
               pncount <= 0;
               c_t <= '0';
        -- activities triggered by rising edge of clock
        elsif clk_code'event and clk_code = '1' then
            g1(2 to 10) <= g1(1 to 9);
            g2(2 to 10) <= g2(1 to 9);
            g1(1) <= g1(3) xor g1(10);
            g2(1) <= g2(2) xor g2(3) xor g2(6) xor g2(7) xor g2(9) xor g2(10);
            c_t <= g1(10) xor g2(T1) xor g2(T2);
            pncount <= pncount + 1;
            if pncount = 1022 then
                g1 <= (others => '1');
                g2 <= (others => '1');
                pncount <= 0;
                c_t <= '0';
            end if;
        end if;
    end process;
end beh_synth2;
```

Acquisition

```
--acquires the signal to within a chip accuracy
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_bit.all;
entity reccorr1 is
    port (din : in std_logic;
          rclk : in  std_logic;
          recrst : in std_logic);
end reccorr1;
architecture rec_correlation of reccorr1 is
    signal rg1, rg2 : std_logic_vector(1 to 10) := (others => '1');
    signal recca : std_logic := '0';
    signal tclk,lclk : std_logic := '0';   -- local clock to manipulate
    signal csum : integer := 0;   -- outputs the correlation sum
    type shift_state is (eq, lt);
    signal shiftclk : shift_state := eq;   -- to shift the clock
    signal rcount : integer := 0;
    signal eval : std_logic := '0';
    constant t1 : integer := 2;
    constant t2 : integer := 6;
begin  -- corr_beh
    process (lclk,din)
        variable rint, tint, intsum, dcount : integer := 0;
        variable tmp : shift_state:= eq;
    begin  -- process
```

```
-- activities triggered by asynchronous reset (active low)
if recrst = '0' then
    rg1(1 to 10) <= (others => '1');
    rg2(1 to 10) <= (others => '1');
    recca <= '0';
    rcount <= 0;
    intsum := 0;      --the integrator sum to determine threshold
    dcount := 0;       -- the count of the number of data samples
    shiftclk <= eq;
-- activities triggered by rising edge of clock
elsif lclk'event and lclk = '1' then
    rg1(2 to 10) <= rg1(1 to 9);
    rg2(2 to 10) <= rg2(1 to 9);
    rg1(1) <= rg1(3) xor rg1(10);
    rg2(1) <= rg2(2) xor rg2(3) xor rg2(6) xor rg2(8) xor rg2(9) xor rg2(10);
    rcount <= rcount + 1;
    recca <= rg1(10) xor (rg2(t1) xor rg2(t2));
    if recca = '0' then
        rint := 1;
    elsif recca = '1' then
        rint := -1;
    end if;
    if din = '0' then
        tint := 1;
    elsif din = '1' then
        tint := -1;
```

```
            end if;

            intsum := intsum + (rint * tint);

            if rcount = 1022 then

                if intsum > 1020 or intsum < -1020 then

                    shiftclk <= eq;

                    csum <= intsum;

                else

                    shiftclk <= lt;

                end if;

                rg1(1 to 10) <= (others => '1');

                rg2(1 to 10) <= (others => '1');

                recca <= '0';

                rcount <= 0;

                intsum := 0;      --the integrator sum to determine threshold

                dcount := 0;       -- the count of the number of data samples

            end if;

         end if;

    end process;

    process (recrst,rclk)

    begin  -- process

        if  recrst = '0' then

            lclk <= rclk;

            eval <= '0';

        elsif rclk'event then

            if rcount = 0 and lclk = '0' then

                if eval = '0' then
```

```
                    eval <= ’1’;
                    if shiftclk = eq then
                        lclk <= not lclk;
                    elsif shiftclk = lt then
                        lclk <= lclk; -- transport rclk after 488.75 ns;
                    end if;
                else
                    eval <= ’0’;
                    lclk <= not lclk;
                end if;
            else
                if eval = ’1’ then
                    eval <= ’0’;
                end if;
                   lclk <= not lclk;
        end if;
      end if;
    end process;
end rec_correlation;
```

Tracking

```
--created by ViswanathDaita
--on 8/16/04
--tracks the signal for a single channel
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_bit.all;
entity dll3synth is
     port (d_in : in std_logic;
           rclk : in  std_logic;
           recrst : in std_logic;
           dout : out std_logic);
end dll3synth;
architecture dll3synth_behave of dll3synth is
    signal rg1,eg1, rg2,eg2 : std_logic_vector(1 to 10) := (others => '1');
    signal recca,early_code,late_code : std_logic := '0';
    signal lclk : std_logic := '0';   -- local clock to manipulate
    type shift_state is (eq, lt);
    signal shiftacq,shifttrack : shift_state := lt;   -- to shift the clock
    signal rcount,sum_early,sum_late,ecount,lcount,early_env,late_env: integer := 0;
    signal acquire,track : std_logic := '0';
    constant t1 : integer := 2;
    constant t2 : integer := 6;
    begin  -- corr_beh
    process (lclk,d_in)
        variable rint, tint, intsum, dcount : integer := 0;
```

```
      variable tmp : shift_state:= eq;
   begin  -- process
      if recrst = '0' then
          rg1(1 to 10) <= (others => '1');
          rg2(1 to 10) <= (others => '1');
          recca <= '0';
          rcount <= 0;
          intsum := 0;      --the integrator sum to determine threshold
          dcount := 0;       -- the count of the number of data samples
          shiftacq <= lt;
      elsif lclk'event and lclk = '0' then
          rg1(2 to 10) <= rg1(1 to 9);
          rg2(2 to 10) <= rg2(1 to 9);
          rg1(1) <= rg1(3) xor rg1(10);
          rg2(1) <= rg2(2) xor rg2(3) xor rg2(6) xor rg2(8) xor rg2(9) xor rg2(10);
          rcount <= rcount + 1;
          recca <= rg1(10) xor (rg2(t1) xor rg2(t2));
          if recca = '0' then
              rint := 1;
          elsif recca = '1' then
              rint := -1;
          end if;
          if d_in = '0' then
              tint := 1;
          elsif d_in = '1' then
              tint := -1;
```

```
            end if;

            intsum := intsum + (rint * tint);

            if rcount = 1022 then

                if intsum > 1020 then

                     shiftacq <= eq;

                       if shifttrack = eq then

                           dout <= '0';

                       end if;

                elsif intsum < -1020 then

                     shiftacq <= eq;

                       if shifttrack = eq then

                           dout <= '1';

                       end if;

                else

                     shiftacq <= lt;

                end if;

                rg1(1 to 10) <= (others => '1');

                rg2(1 to 10) <= (others => '1');

                recca <= '0';

                rcount <= 0;

                intsum := 0;      --the integrator sum to determine threshold

                dcount := 0;       -- the count of the number of data samples

           end if;

         end if;

    end process;

    process (lclk,d_in,recrst)
```

```
        variable inc,e_in,l_in : integer := 0;
        variable delay : std_logic := '0';
    begin  -- process
        if recrst = '0' then
            ecount <= 0;
            lcount <= -1;
            eg1 <= (others => '1');
            eg2 <= (others => '1');
            late_code <= '0';
            early_code <= '0';
            inc := 0;
            e_in := 0;
            l_in := 0;
            sum_early <= 0;
            sum_late <= 0;
            delay := '0';
            early_env <= 0;
            late_env <= 0;
        elsif shiftacq = eq then
            if lclk'event and lclk = '1' then
                eg1(2 to 10) <= eg1(1 to 9);
                eg2(2 to 10) <= eg2(1 to 9);
                eg1(1) <= eg1(3) xor eg1(10);
                eg2(1) <= eg2(2) xor eg2(3) xor eg2(6) xor eg2(8) xor eg2(9) xor eg2(10)
                early_code <= eg1(10) xor eg2(t1) xor eg2(t2);
                if d_in = '1' then
                    inc := -1;
                else
                    inc := 1;
                end if;
                if early_code = ’1’ then
                    e_in := -1;
                else
                    e_in := 1;
                end if;
                late_code <= early_code;
                if late_code = ’1’ then
                    l_in := -1;
                else
                    l_in := 1;
                end if;
                sum_early <= sum_early + inc * e_in;
                if ecount = 0 then
                    sum_late <= 0;
                else
                    sum_late <= sum_late + inc * l_in;
                end if;
                ecount <= ecount + 1;
                early_env <= sum_early * sum_early;
                late_env <= sum_late * sum_late;
                if ecount = 1022 then
                    eg1 <= (others => ’1’);
```

```
                    eg2 <= (others => '1');
                    ecount <= 0;
                    lcount <= -1;
                    sum_late <= 0;
                    sum_early <= 0;
                end if;
            end if;
         end if;
  end process;
    -- to generate the local clock from the receiver clock
  process (recrst,rclk)
  begin  -- process
        if  recrst = '0' then
             lclk <= rclk;
             acquire <= '0';
             track <= '0';
        elsif rclk'event then
            if shiftacq = eq then
               if rcount = 1022 and lclk = '0' then
                    if early_env < late_env then
                        if track = '0' then
                            track <= '1';
                            lclk <= lclk; -- transport rclk after 488.75 ns;
                        else
                            track <= '0';
                            lclk <= not lclk;
                        end if;
                    else
                           lclk <= not lclk;
                    end if;
               else
                    if track = '1' then
                        track <= '0';
                    end if;
                    lclk <= not lclk;
               end if;
               shifttrack <= eq;   -- entered tracking mode
         elsif shiftacq = lt then
           shifttrack <= lt;   -- not entered tracking
             if rclk'event then
               if rcount = 0 and lclk = '0' then
                    if acquire = '0' then
                        acquire <= '1';
                        lclk <= lclk;
                    else
acquire <= '0';
lclk <= not lclk;
                    end if;
               else
                    if acquire = '1' then
                        acquire <= '0';
                    end if;
                    lclk <= not lclk;
               end if;
            end if;
        end if;
    end if;
    end process;
end dll3synth_behave;
```

Acquiring Multiple Satellites

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_bit.all;

entity mul_channel_rx1 is

    port (sat_1 : in std_logic;   -- data from satellite id 1

          sat_2 : in std_logic;   -- data from satellite id 2

          sat_3 : in std_logic;   -- data from satellite id 6

          sat_4 : in std_logic;   -- data from satellite id 9

          rclk : in  std_logic;

          recrst : in std_logic);

end mul_channel_rx1;

architecture mult_corr1_behave of mul_channel_rx1 is

    signal rg1, rg2,rg3,rg4,rg5,rg6,rg7,rg8 : std_logic_vector(1 to 10) := (others => '1

    signal recca1,recca2,recca3,recca4 : std_logic := '0';

    signal lclk1,lclk2,lclk3,lclk4 : std_logic := '0';  -- local clocks

    signal csum1,csum2,csum3,csum4 : integer := 0;   --  correlation sum

    type shift_state is (eq, lt);

    signal shiftclk1,shiftclk2,shiftclk3,shiftclk4 : shift_state := eq;

    signal rcount1,rcount2,rcount3,rcount4 : integer := 0;

    signal eval1,eval2,eval3,eval4 : std_logic := '0';

    constant t1 : integer := 2;

    constant t2 : integer := 6;

    constant t3 : integer := 3;

    constant t4 : integer := 7;

    constant t5 : integer := 10;
    constant t6 : integer := 1;

    constant t7 : integer := 4;

    constant t8 : integer := 5;

    constant t9 : integer := 8;

    constant t10 : integer := 9;

begin  -- corr_beh

    process (lclk1,sat_1,sat_2,sat_3,sat_4)

        variable rint1 : integer := 0;

        variable tint1,tint2,tint3,tint4 : integer := 0;

        variable intsum_11,intsum_12,intsum_13,intsum_14 : integer := 0;

    begin  -- process

        if recrst = '0' then

            rg1(1 to 10) <= (others => '1');

            rg2(1 to 10) <= (others => '1');

            recca1 <= '0';

            rcount1 <= 0;

            intsum_11 := 0;      --the integrator sum to determine threshold

            intsum_12 := 0;

            intsum_13 := 0;

            intsum_14 := 0;

            shiftclk1 <= lt;

        elsif lclk1'event and lclk1 = '1' then

            rg1(2 to 10) <= rg1(1 to 9);

            rg2(2 to 10) <= rg2(1 to 9);

            rg1(1) <= rg1(3) xor rg1(10);

            rg2(1) <= rg2(2) xor rg2(3) xor rg2(6) xor rg2(8) xor rg2(9) xor rg2(10);
            rcount1 <= rcount1 + 1;

            recca1 <= rg1(10) xor (rg2(t1) xor rg2(t2));  -- satellite id 1

            -- determine the receiver code

            if recca1 = ’0’ then

                rint1 := 1;

            elsif recca1 = ’1’ then

                rint1 := -1;

            end if;

            -- determining the received code

            if  sat_1= ’0’ then

                tint1 := 1;

            elsif sat_1 = ’1’ then

                tint1 := -1;

    end if;

            if sat_2 = ’0’ then

                tint2 := 1;

            elsif sat_2 = ’1’ then

                tint2 := -1;

            end if;

            if sat_3= ’0’ then

                tint3 := 1;

            elsif sat_3 = ’1’ then

                tint3 := -1;

            end if;

            if sat_4 = ’0’ then

                tint4 := 1;
            elsif sat_4 = '1' then
                tint4 := -1;
            end if;
             intsum_11 := intsum_11 + (rint1 * tint1);
             intsum_12 := intsum_12 + (rint1 * tint2);
             intsum_13 := intsum_13 + (rint1 * tint3);
             intsum_14 := intsum_14 + (rint1 * tint4);
             if rcount1 = 1022 then
                if (intsum_11 > 1020 or intsum_11 < -1020) or
                   (intsum_12 > 1020 or intsum_12 < -1020) or
                   (intsum_13 > 1020 or intsum_13 < -1020) or
                   (intsum_14 > 1020 or intsum_14 < -1020) then
                    shiftclk1 <= eq;
                else
                    shiftclk1 <= lt;
                end if;
                rg1(1 to 10) <= (others => '1');
                rg2(1 to 10) <= (others => '1');
                recca1 <= '0';
                rcount1 <= 0;
                intsum_11 := 0;  --the integrator sum to determine threshold
                intsum_12 := 0;
                intsum_13 := 0;
                intsum_14 := 0;
            end if;
         end if;
```

```
    end process;

    process (recrst,rclk)

    begin  -- process

        if  recrst = '0' then

            lclk1 <= rclk;

            eval1 <= '0';

        elsif rclk'event then

            if rcount1 = 0 and lclk1 = '0' then

               if eval1 = '0' then

                    eval1 <= '1';

                    if shiftclk1 = eq then

                        lclk1 <= not lclk1;

                    elsif shiftclk1 = lt then

                        lclk1 <= lclk1; -- transport rclk after 488.75 ns;

                    end if;

               else

                    eval1 <= '0';

                    lclk1 <= not lclk1;

               end if;

            else

               if eval1 = '1' then

                    eval1 <= '0';

               end if;

               lclk1 <= not lclk1;

        end if;

      end if;
    end process;

    process (lclk2,sat_1,sat_2,sat_3,sat_4)

        variable rint2: integer := 0;

        variable tint5,tint6,tint7,tint8 : integer := 0;

        variable intsum_21,intsum_22,intsum_23,intsum_24 : integer := 0;

    begin  -- process

        if recrst = ’0’ then

            rg3(1 to 10) <= (others => ’1’);

            rg4(1 to 10) <= (others => ’1’);

            recca2 <= ’0’;

            rcount2 <= 0;

            intsum_21 := 0;      --the integrator sum to determine threshold

            intsum_22 := 0;

            intsum_23 := 0;

            intsum_24 := 0;

            shiftclk2 <= lt;

        elsif lclk2’event and lclk2 = ’1’ then

            rg3(2 to 10) <= rg3(1 to 9);

            rg4(2 to 10) <= rg4(1 to 9);

            rg3(1) <= rg3(3) xor rg3(10);

            rg4(1) <= rg4(2) xor rg4(3) xor rg4(6) xor rg4(8) xor rg4(9) xor rg4(10);

            rcount2 <= rcount2 + 1;

            recca2 <= rg3(10) xor (rg4(t3) xor rg4(t4));  -- satellite id 2

            if recca2 = ’0’ then

                rint2 := 1;

            elsif recca2 = ’1’ then
                rint2 := -1;
            end if;
            if  sat_1= '0' then
                tint5 := 1;
            elsif sat_1 = '1' then
                tint5 := -1;
            end if;
            if sat_2 = '0' then
                tint6 := 1;
            elsif sat_2 = '1' then
                tint6 := -1;
            end if;
            if sat_3= '0' then
                tint7 := 1;
            elsif sat_3 = '1' then
                tint7 := -1;
            end if;
            if sat_4 = '0' then
                tint8 := 1;
            elsif sat_4 = '1' then
                tint8 := -1;
            end if;
            intsum_21 := intsum_21 + (rint2 * tint5);
            intsum_22 := intsum_22 + (rint2 * tint6);
            intsum_23 := intsum_23 + (rint2 * tint7);
            intsum_24 := intsum_24 + (rint2 * tint8);
```

```
            if rcount2 = 1022 then
                if (intsum_21 > 1020 or intsum_21 < -1020) or
                   (intsum_22 > 1020 or intsum_22 < -1020) or
                   (intsum_23 > 1020 or intsum_23 < -1020) or
                   (intsum_24 > 1020 or intsum_24 < -1020) then
                    shiftclk2 <= eq;
                else
                    shiftclk2 <= lt;
                end if;
                rg3(1 to 10) <= (others => '1');
                rg4(1 to 10) <= (others => '1');
                recca2 <= '0';
                rcount2 <= 0;
                intsum_21 := 0;  --the integrator sum to determine threshold
                intsum_22 := 0;
                intsum_23 := 0;
                intsum_24 := 0;
            end if;
         end if;
    end process;
    process (recrst,rclk)
    begin  -- process
        if  recrst = '0' then
            lclk2 <= rclk;
            eval2 <= '0';
        elsif rclk'event then
```

```
            if rcount2 = 0 and lclk2 = '0' then
                if eval2 = '0' then
                    eval2 <= '1';
                    if shiftclk2 = eq then
                        lclk2 <= not lclk2;
                    elsif shiftclk2 = lt then
                        lclk2 <= lclk2; -- transport rclk after 488.75 ns;
                    end if;
                else
                    eval2 <= '0';
                    lclk2 <= not lclk2;
                end if;
            else
                if eval2 = '1' then
                    eval2 <= '0';
                end if;
                lclk2 <= not lclk2;
            end if;
    end if;
    end process;
    process (lclk3,sat_1,sat_2,sat_3,sat_4)
        variable rint3 : integer := 0;
        variable tint9,tint10,tint11,tint12 : integer := 0;
        variable intsum_31,intsum_32,intsum_33,intsum_34 : integer := 0;
    begin  -- process
        if recrst = '0' then
```

```
            rg5(1 to 10) <= (others => '1');
            rg6(1 to 10) <= (others => '1');
            recca3 <= '0';
            rcount3 <= 0;
            intsum_31 := 0;      --the integrator sum to determine threshold
            intsum_32 := 0;
            intsum_33 := 0;
            intsum_34 := 0;
            shiftclk3 <= lt;
        elsif lclk3'event and lclk3 = '1' then
            rg5(2 to 10) <= rg5(1 to 9);
            rg6(2 to 10) <= rg6(1 to 9);
            rg5(1) <= rg5(3) xor rg5(10);
            rg6(1) <= rg6(2) xor rg6(3) xor rg6(6) xor rg6(8) xor rg6(9) xor rg6(10);
            rcount3 <= rcount3 + 1;
            recca3 <= rg5(10) xor (rg6(t1) xor rg6(t5));  -- satellite id 6
            if recca3 = '0' then
                rint3 := 1;
            elsif recca3 = '1' then
                rint3 := -1;
            end if;
            if  sat_1= '0' then
                tint9 := 1;
            elsif sat_1 = '1' then
                tint9 := -1;
            end if;
            if sat_2 = '0' then
                tint10 := 1;
            elsif sat_2 = '1' then
                tint10 := -1;
            end if;
            if sat_3= '0' then
                tint11 := 1;
            elsif sat_3 = '1' then
                tint11 := -1;
            end if;
            if sat_4 = '0' then
                tint12 := 1;
            elsif sat_4 = '1' then
                tint12 := -1;
            end if;
            intsum_31 := intsum_31 + (rint3 * tint9);
            intsum_32 := intsum_32 + (rint3 * tint10);
            intsum_33 := intsum_33 + (rint3 * tint11);
            intsum_34 := intsum_34 + (rint3 * tint12);
            if rcount3 = 1022 then
                if (intsum_31 > 1020 or intsum_31 < -1020) or
                   (intsum_32 > 1020 or intsum_32 < -1020) or
                   (intsum_33 > 1020 or intsum_33 < -1020) or
                   (intsum_34 > 1020 or intsum_34 < -1020) then
                    shiftclk3 <= eq;
                else
```

```
                    shiftclk3 <= lt;
                end if;
                rg5(1 to 10) <= (others => '1');
                rg6(1 to 10) <= (others => '1');
                recca3 <= '0';
                rcount3 <= 0;
                intsum_31 := 0;  --the integrator sum to determine threshold
                intsum_32 := 0;
                intsum_33 := 0;
                intsum_34 := 0;
            end if;
         end if;
    end process;
    process (recrst,rclk)
    begin  -- process
        if  recrst = '0' then
            lclk3 <= rclk;
            eval3 <= '0';
        elsif rclk'event then
            if rcount3 = 0 and lclk3 = '0' then
                if eval3 = '0' then
                    eval3 <= '1';
                    if shiftclk3 = eq then
                        lclk3 <= not lclk3;
                    elsif shiftclk3 = lt then
                        lclk3 <= lclk3; -- transport rclk after 488.75 ns;
```

```
                  end if;
              else
                  eval3 <= '0';
                  lclk3 <= not lclk3;
              end if;
           else
              if eval3 = '1' then
                  eval3 <= '0';
              end if;
              lclk3 <= not lclk3;
       end if;
    end if;
 end process;
 process (lclk4,sat_1,sat_2,sat_3,sat_4)
      variable rint4 : integer := 0;
      variable tint13,tint14,tint15,tint16 : integer := 0;
      variable intsum_41,intsum_42,intsum_43,intsum_44: integer := 0;
 begin  -- process
      if recrst = '0' then
          rg7(1 to 10) <= (others => '1');
          rg8(1 to 10) <= (others => '1');
          recca4 <= '0';
          rcount4 <= 0;
          intsum_41 := 0;      --the integrator sum to determine threshold
          intsum_42 := 0;
          intsum_43 := 0;
```

```
            intsum_44 := 0;
            shiftclk4 <= lt;
        elsif lclk4'event and lclk4 = '1' then
            rg7(2 to 10) <= rg7(1 to 9);
            rg8(2 to 10) <= rg8(1 to 9);
            rg7(1) <= rg7(3) xor rg7(10);
            rg8(1) <= rg8(2) xor rg8(3) xor rg8(6) xor rg8(8) xor rg8(9) xor rg8(10);
            rcount4 <= rcount4 + 1;
            recca4 <= rg7(10) xor (rg8(t3) xor rg8(t5));  -- satellite id 9
            if recca4 = '0' then
                rint4 := 1;
            elsif recca4 = '1' then
                rint4 := -1;
            end if;
            if  sat_1= '0' then
                tint13 := 1;
            elsif sat_1 = '1' then
                tint13 := -1;
            end if;
            if sat_2 = '0' then
                tint14 := 1;
            elsif sat_2 = '1' then
                tint14 := -1;
            end if;
            if sat_3= '0' then
                tint15 := 1;
            elsif sat_3 = '1' then
                tint15 := -1;
            end if;
            if sat_4 = '0' then
                tint16 := 1;
            elsif sat_4 = '1' then
                tint16 := -1;
            end if;
            intsum_41 := intsum_41 + (rint4 * tint13);
            intsum_42 := intsum_42 + (rint4 * tint14);
            intsum_43 := intsum_43 + (rint4 * tint15);
            intsum_44 := intsum_44 + (rint4 * tint16);
            if rcount4 = 1022 then
                if (intsum_41 > 1020 or intsum_41 < -1020) or
                   (intsum_42 > 1020 or intsum_42 < -1020) or
                   (intsum_43 > 1020 or intsum_43 < -1020) or
                   (intsum_44 > 1020 or intsum_44 < -1020) then
                    shiftclk4 <= eq;
                else
                    shiftclk4 <= lt;
                end if;
                rg7(1 to 10) <= (others => '1');
                rg8(1 to 10) <= (others => '1');
                recca4 <= '0';
                rcount4 <= 0;
                intsum_41 := 0;    --the integrator sum to determine threshold
                intsum_42 := 0;
                intsum_43 := 0;
                intsum_44 := 0;
            end if;
        end if;
    end process;
    process (recrst,rclk)
    begin  -- process
        if  recrst = ’0’ then
            lclk4 <= rclk;
            eval4 <= ’0’;
        elsif rclk’event then
            if rcount4 = 0 and lclk4 = ’0’ then
                if eval4 = ’0’ then
                    eval4 <= ’1’;
                    if shiftclk4 = eq then
                        lclk4 <= not lclk4;
                    elsif shiftclk4 = lt then
                        lclk4 <= lclk4; -- transport rclk after 488.75 ns;
                    end if;
                else
                    eval4 <= ’0’;
                    lclk4 <= not lclk4;
                end if;
            else
                if eval4 = ’1’ then
```

```
                    eval4 <= '0';
                end if;
                lclk4 <= not lclk4;
            end if;
        end if;
    end process;
end mult_corr1_behave;
```