UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

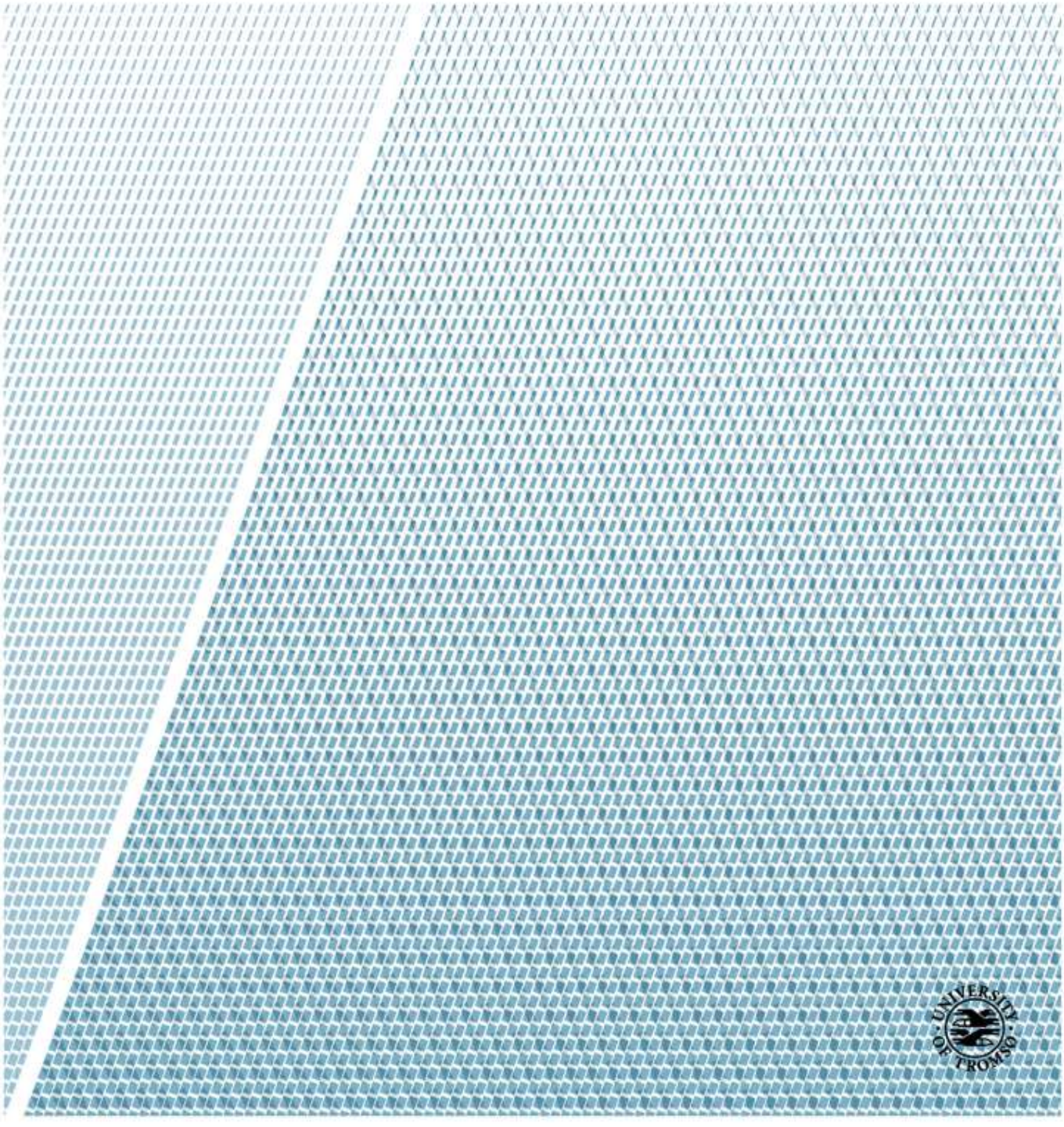
Department of Satellite Engineering

FPGA-based Tracking System for GNSS Receivers

—

Anil Manandhar

Master thesis in Satellite Engineering, June 2017



Title:	**FPGA-based Tracking System for GNSS Receivers**	*Date:* June 6, 2017 *Classification:* Open *Pages*: 66
Author:	**Anil Manandhar**	*Attachments:* 5
Department:	Faculty of Engineering Science and Technology	
Course:	SHO6300, Master thesis -M-ST	
Supervisor:	**Associate Professor Dr. Tuan-Vu Cao**	
Principal:	UiT The Artic University of Norway, Campus Narvik	
Principal Contact:	**Associate Professor Dr. Tuan-Vu Cao**	

Keywords: Tracking system, GNSS receivers, code tracking loop, carrier tracking loop, code alignment, carrier signal alignment, FPGA-based tracking algorithm

Acknowledgements

Firstly, I would like to express my special thanks of gratitude to my supervisor, Associate Professor Dr. Tuan-Vu Cao for providing me a golden opportunity to do this project **"FPGA-based tracking system for GNSS receivers"**. His lectures on embedded systems paved the pathway for my research and VHDL programming. It is because of his continuous guidance and supervision that this project could operate smoothly. I am immensely grateful for his availability despite his time constraints. Undoubtedly, this project would have been incomplete without his support.

I would also like to thank my advisor Mr. Tor-Aleksander Johansen for his wonderful feedback and comments, which only improved my writing in thesis.

Lastly, I would like to thank my friends and family for supporting me spiritually throughout writing this thesis. They were the source of inspiration and motivation.

Anil Manandhar

Abstract

This report presents the design and simulation of code and carrier tracking system for a GNSS receiver. The GNSS receiver processes the signal sent by satellites in space. These signal contain carrier wave signal, ranging code and navigation data in encrypted form. To demodulate the navigation data, the processing block should accurately track the phase of incoming code and the frequency of incoming carrier wave signal.

In code tracking loop, a DLL is used where three replicas of incoming PRN code namely early PRN, prompt PRN and late PRN are generated and correlated with the incoming signal. The result of these correlators is a numerical value that determines how the replica codes correlate with the incoming PRN code. Based on correlation value, a code loop discriminator decides in which direction the phase of PRN code is to be shifted. Then a perfectly aligned PRN code is generated by the local code generator.

In carrier tracking loop, a PLL is used where a local carrier wave signal is multiplied with the incoming signal to wipe off carrier signal and PRN code of the incoming signal. The output after multiplication is sent to the carrier loop discriminator to determine the carrier phase error which is filtered out by a carrier loop filter. Then the output from the filter is used as feedback to a NCO that generates a perfectly aligned carrier wave signal.

The other half of this report deals with the VHDL programming of the tracking subsystems that can be synthesized in a FPGA kit. It should be noted that all subsystems of a tracking block cannot be hardware synthesized. A VHDL program and a testbench program for the subsystem that can be hardware synthesized is coded and tested in ISIM. The implementation of FPGA-based tracking algorithm is verified with use of an oscilloscope. Finally, a design for FPGA-based tracking system for GNSS receivers is proposed.

List of abbreviations

GNSS	Global Navigation Satellite System
GPS	Global Positioning System
SIS	Signal In Space
ASIC	Application Specific Integrated Circuits
PVT	Position, Velocity and Time
FPGA	Field Programmable Gate Array
LEO	Low Earth Orbit
UHF	Ultra High Frequency
ITU	International Telecommunication Union
C/A	Coarse Acquisition
P-Code	Precision Code
PPS	Picture Parameter Set
SPS	Sequence Parameter Set
BPSK	Binary Phase Shift Keying
LFSR	Linear Feedback Shift Register
NCO	Numerically Controlled Oscillator
I2C	Inter-IC
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background study

GNSS (Global Navigation Satellite System) has increased in popularity over the past decade, perhaps due to the rapid development and wide use of consumer products based on GNSS. In the present context, we cannot even imagine an electronic device (such as cell phone, laptop and tablet) or an automobile not being equipped with a GPS (Global Positioning System). Currently, GPS is the GNSS, primarily used by both military and civilians. However, the services that are provided to the civilian users are limited compared to the military users. Some of the areas of GNSS application are:

- Personal navigation
- Aviation applications
- Automotive applications
- Marine applications
- Geodesy and surveying
- Space applications

In this project, the main concern of GNSS is in the field of space applications. The GNSS receiver in the terrestrial environment is sophisticated with unlimited resources such that hardware and software components can be altered whenever it is required. However, the resources in space environment is limited because of which alteration in hardware and software is challenging. Power consumption, performance, reliability, size, weight and radiation in space are some of the constraints needed to be considered while designing a GNSS receiver.

1.1.1 GNSS receiver

A GNSS receiver is a system that processes the signal sent by the satellites in space or simply, it processes the Signal In Space (SIS) to determine user's position, velocity and time.

The satellites are in continuous motion. So the receiver has to make sure that the signals

emitting from the satellites are continuously tracked and monitored. The receiver monitors the propagation time of the incoming signals traveling through space. A technique called "pseudorange" is used to make a rough estimation of true range between a satellite and a user. This computed value has to go through a number of phenomena before it can be interpreted as a precise measurement of the true distance.

The basic block diagram of a typical GNSS receiver is shown in figure 1.1 extracted from (Re & Ruggieri 2007). A typical GNSS receiver consists of an antenna fed into a series of ASIC (Application Specific Integrated Circuits) components controlled by a processor. The received signal has to go through a number of processes such as signal acquisition, signal tracking, synchronizing navigation data and decoding navigation data before delivering the final results - PVT (Position, Velocity and Time).

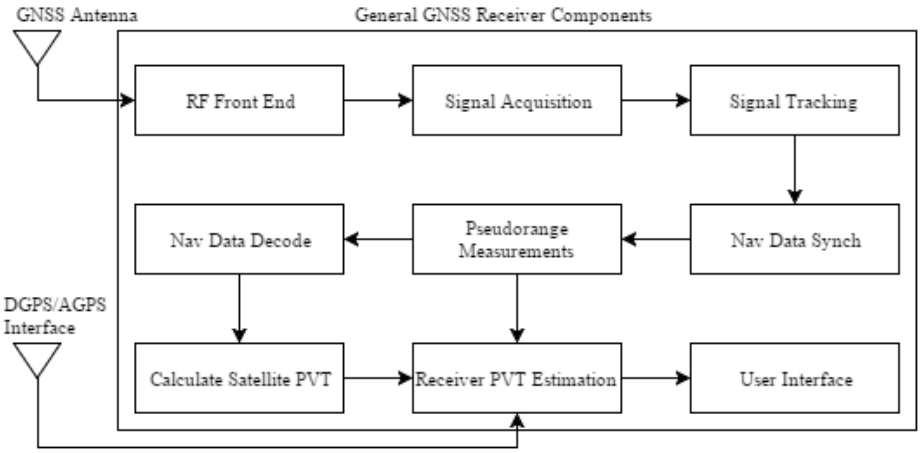


Figure 1.1: General block digram of a typical GNSS Rreceiver (Re & Ruggieri 2007)

The GNSS receiver can be classified into hardware based receiver, software based receiver and FPGA (Field Programmable Gate Array) based receiver.

Hardware based receiver

Traditionally, the GNSS receivers are ASIC receivers or simply hardware receivers. These are designed to perform certain categories of applications. They have high functionality in terms of performance and power consumption (Guruprasad 2015). However, the major drawback of this system is that the design cannot be modified easily because ASIC is a specialized chip designed for specific purposes. When new algorithms are needed to be implemented, the modifications would require a re-fabrication of the receiver which eventually increases cost and time. This has led to the development of software based receiver.

Software based receiver

A software based receiver uses a central processing unit that processes the signal coming from the satellites. The goal of a software receiver is to make the design as simple as possible by

using softwares instead of hardware components for digital signal processing. This has resulted in decreased size, low power consumption, low cost and high flexibility. The downside of this technology is that a central processor still has to perform all digital signal processing tasks usually performed by digital correlators (Gleason & Gebre-Egziabher 2009). This increases processing load in the processor which eventually lowers the performance of the receiver. This has led to the development of FPGA based receiver.

FPGA based receiver

In a FPGA based receiver, a highly flexible FPGA kit is used. The FPGA based receivers have grown popularity over the last decade, primarily due to the high circuit density achievable on a relatively small programmable chip. Furthermore, this system can be developed at a very low cost. Here, the power consumption is very low and the performance is very high unlike software based receiver which is slow. However, designing a receiver based on FPGA is very complex.

1.1.2 Tracking system in GNSS receivers

The basic block diagram of a tracking system in GNSS receiver (Johansson, et al. 1998) is shown in figure 1.2. A tracking system in GNSS receiver plays a very important role in demodulating a navigation data from an incoming signal. An incoming signal can be a GPS (L1, L2, L5), GLONASS (L1, L2), GALILEO (E1A-E1C, E5a, E5b, AltBOC, E6) or BeiDou (B1, B2, B3). A tracking system consists of two main parts - code tracking loop and carrier tracking loop. To successfully track the navigation data coming from the satellites, a tracking system has to properly track the code phase and the carrier phase.

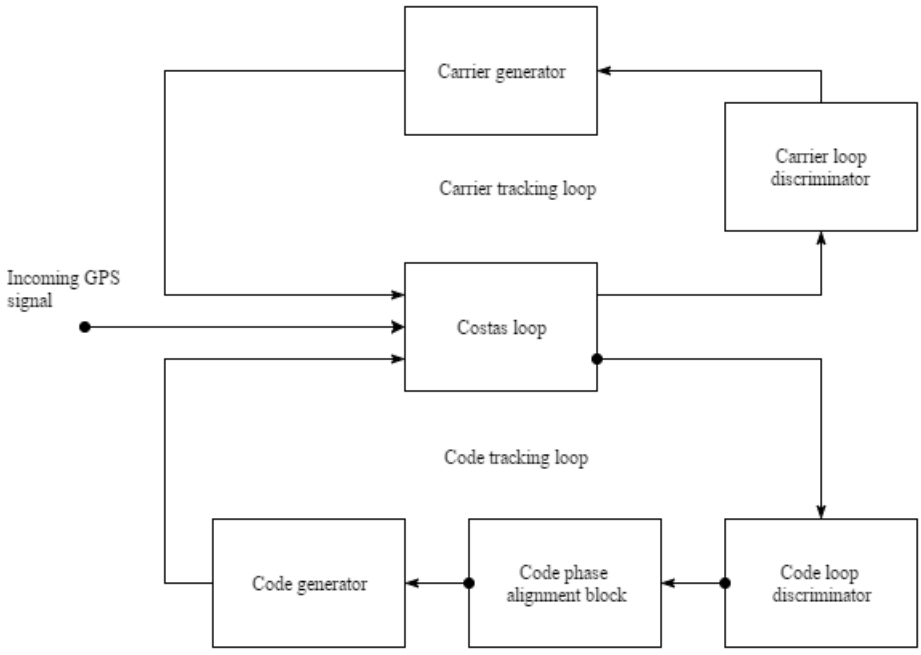


Figure 1.2: General block digram of a tracking system in GNSS receiver (Johansson et al. 1998)

A Doppler frequency shift is a one of the reason that the code phase and the carrier phase get misaligned. So, a code tracking loop functions in such a way that the incoming code is properly aligned with the local code. Similarly, a carrier tracking loop functions in such a way that the incoming carrier signal is properly aligned with the local carrier signal. Once the code and carrier signals are properly tracked, the demodulation of navigation data can be performed.

1.2 Previous works in tracking system for GNSS receivers

Many researches were carried out related to code and carrier tracking system for GNSS receivers. Only major works in tracking system for GNSS receivers are discussed in this section. The list begins with a traditional tracking system for GNSS receivers and ends with a modern FPGA-based tracking system for GNSS receivers.

1.2.1 Coherent spread spectrum systems

This book by (Holmes 1982), mainly focused on coherent carrier demodulation technique for synchronization of direct sequence spread spectrum. This method could only be implied when the local carrier wave is perfectly aligned with the incoming carrier wave. Thus, the performance of this system was very poor whenever the carrier signals were misaligned. This led to the development of new methods and techniques for code and carrier tracking.

1.2.2 GNSS code and carrier tracking in the presence of multipath

Multipath is one of the error source in differential GNSS positioning. According to (Brodin & Daly 1997), for short delay multipath signals, carrier multipath error was a major problem. The author made a thorough investigation on the performance of several coherent and non-coherent discriminators concerning the multipath, where he found that mean code errors produced by "non-coherent early minus late power discriminator" were greater than those of "coherent discriminator" and "dot product discriminator". Thus, the author concluded that "the early minus late power discriminator" should be practiced for better performance.

1.2.3 High performance code and carrier tracking architecture

The work presented in the paper (Weill 2010) resulted in a new architecture with improved performance over conventional tracking methods. His work was based on (Brodin & Daly 1997) and used a central processing unit that had to perform all the digital signal processing tasks. A reduced position and velocity errors, reduced tracking thresholds, reduced search space size and reduced number of satellites required were some benefits from his work. A use of processing unit would make the design simple but it would also increase load in the processor which would eventually affect the performance of receiver.

1.2.4 Implementation of code and carrier tracking stage on a FPGA

FPGA provides flexibility for both developer and designer to make changes in the system without configuring the hardware blocks. This paper (Kappen & Noll 2006) focused on a reconfigurable GNSS receiver. It meant that making changes in the software could easily

alter the performance of the receiver. Furthermore, this system could be developed at a much lower cost. The author also claimed that designing the system in a FPGA was one the toughest task as all of the subsystems concerning the GNSS receiver could not be hardware synthesized.

1.3 Thesis outline

1.3.1 Problem statement and motivation

GNSS receivers hold many promising applications for LEO (Low Earth Orbit) satellites. GNSS receivers are mostly focused in determining users position, velocity and time. This goal can only be achieved if the carrier wave signal frequency and code phase are accurately tracked.

A hardware based tracking system for GNSS receiver has a better performance but it lacks flexibility. Modification in algorithms require re-fabrication of the receiver resulting in extra cost and time. A software based tracking system for GNSS receiver is flexible but the use of processor in digital signal processing tasks increase processing load which eventually lowers the performance of the receiver. Thus, a FPGA based tracking system for GNSS receiver is required. A FPGA based tracking system is highly flexible such that new algorithms can be easily implemented by making changes in the program. Some other advantages of FPGA based tracking system is that the manufacturing cost is very low with high performance.

1.3.2 Thesis objectives

The main objective of this thesis is to design and simulate a tracking system for GNSS receiver in a simulation environment using tools such as simulink. There are many algorithms to be followed for the design. The best and the most effective of them must be chosen. These algorithms are discussed in chapter 2.

The other task of this thesis is to redesign the subsystems of the tracking system for GNSS receiver such that they can be synthesized in a FPGA kit. It should also be noted that all of the subsystems involved in the receiver cannot be synthesized in a FPGA. Only the ones that can be hardware synthesized shall be implemented. Finally, the design needs to be tested and verified.

Some other additional tasks are to understand the GNSS receivers, understand signal tracking algorithm for GNSS receivers and propose a design of tracking system for GNSS receiver in satellite applications.

Chapter 2

Code and Carrier Tracking

Based on figure 1.2 from chapter 1, a thorough discussion on GPS signal characteristics, Doppler frequency shift, code tracking algorithms and carrier tracking algorithms are made in this chapter.

2.1 GPS signal characteristics

GPS satellites transmit microwave signals. The GPS receiver antenna either on or near the Earth's surface respond to these received signals by determining users position, velocity and time. The GPS signals are broadcasted on three frequencies - L1, L2 and L5.

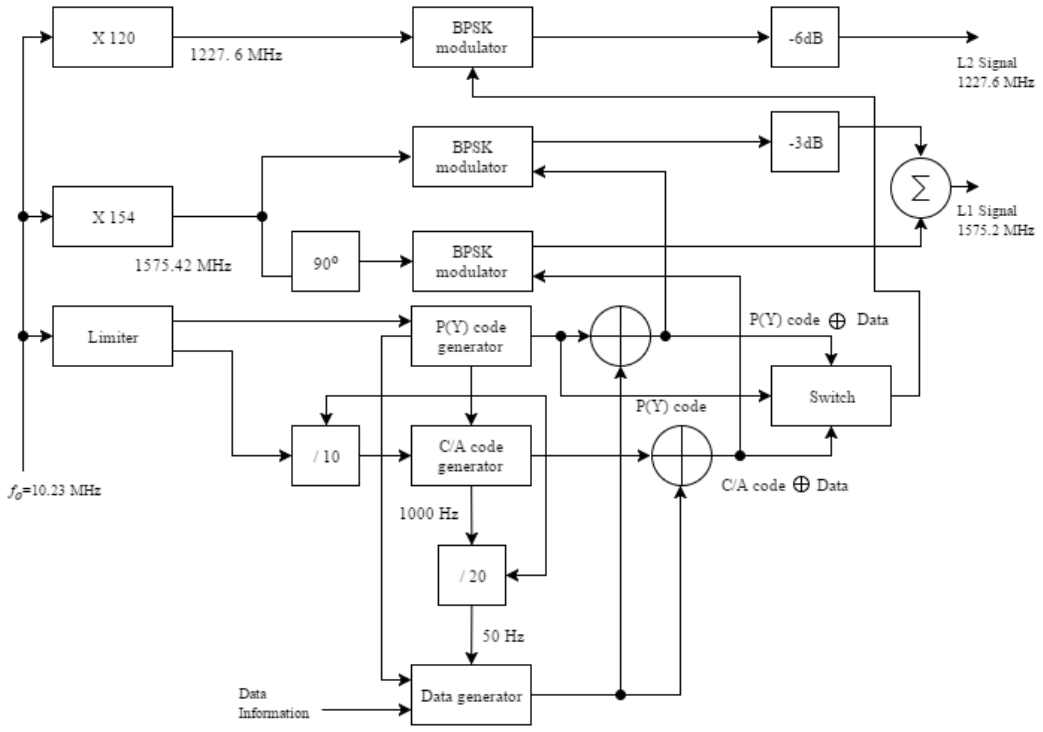


Figure 2.1: Block diagram to generate GPS signals - L1 and L2 (Borre, et al. 2007)

An illustration of GPS signal generation for L1 signal and L2 signal are shown in figure 2.1.

To generate a L1 signal, P-code (Precision code) and navigation data are multiplied together, followed by a BPSK (Binary Phase Shift Keying) modulation with incoming carrier signal. Then a second multiplication is performed between C/A (Coarse/Acquisition) and navigation data, followed by a BPSK modulation with incoming 90^o phase-shifted carrier signal. Finally, the two results from BPSK modulators are summed up to obtain the L1 signal of frequency 1575.42 MHz.

Now, to generate a L2 signal, a multiplication is carried out between P-code and navigation data. The result of multiplication and carrier wave signal are modulated by BPSK technique to get a L2 signal of frequency 1227.6 MHz.

This project focuses on L1 signal tracking for GNSS receivers. Thus, figure 2.1 is simplified into figure 2.2 (Johansson et al. 1998). A common frequency, f_o=10.23 MHz signal is used to generate a L1 carrier signal of f_{L1}=1575.42 MHz. The modulo-2 sum in the figure represents the exclusive-or operation, denoted by $\oplus$. The modulo-2 sum outputs a signal with values {0,1}. Then this result (from modulo-2 sum) and the carrier signal is modulated by Binary Phase Shift Keying (BPSK) to get a L1 signal with values {-1,1}. In other words, in BPSK modulation, signal with values {0,1} are mapped into {-1,1} (Borre et al. 2007). One good reason to choose BPSK over other modulation techniques is that retrieving the original bits in receiver is much easier. The chipping rate of the C/A code is 1.023 MHz while the bit rate for the navigation data is at a much slower rate of 50 Hz.

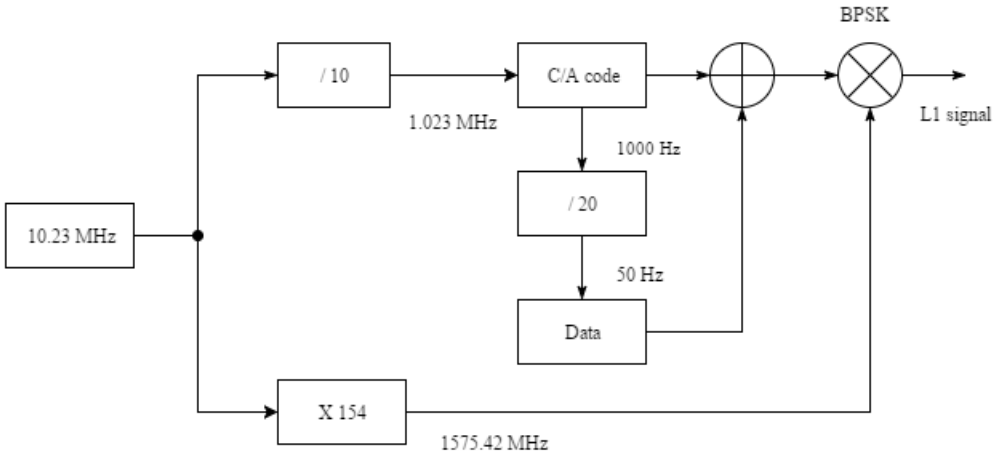


Figure 2.2: A simplified GPS L1 modulator configuration (Johansson et al. 1998)

The L1 signal is defined by

$$L_i(\omega_1 t) = A[CA_i(t) \oplus D_i(t)]sin(\omega_1 t) \tag{2.1}$$

where,

L_i = signal from i^{th} satellite
ω_1 = frequency for L1, in radians

i = satellite id number
A = amplitude of the signal
CA_i = C/A code for i^{th} satellite
D = navigation message/data for i^{th} satellite

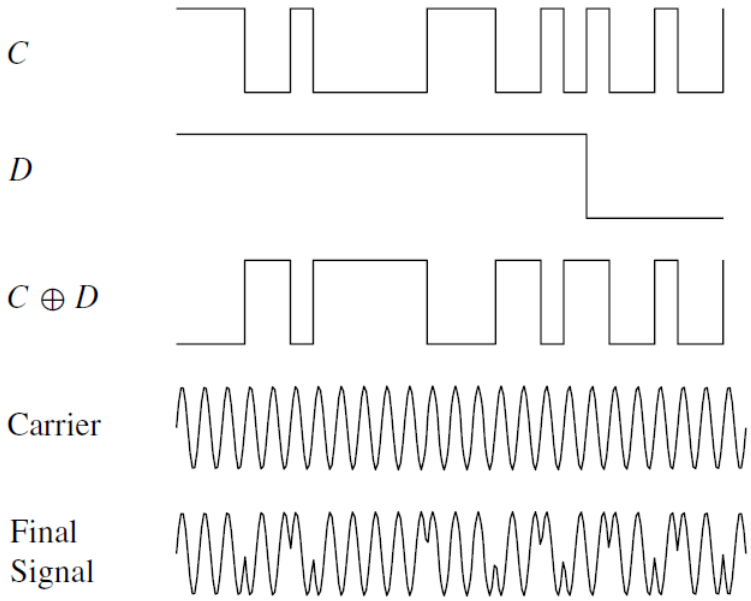


Figure 2.3: C/A code, navigation data, output from modulo-2 sum, carrier signal and final GPS signal (Borre et al. 2007)

The GPS signal carries a navigation data or a message signal in an encrypted format. It also consists of a carrier wave signal and a ranging code. The signal behavior of C/A code, navigation data, output from modulo-2 sum, carrier signal and GPS signal can be seen in figure 2.3.

2.1.1 Carrier wave signal

The summary for signal L1 and signal L2 frequencies are listed in table 2.1. This table also

Table 2.1: Frequency summary for signals L1 and L2

Signal designation	**L1**	**L2**
carrier frequency, f_o = 10.23 MHz	$154f_o$ = 1575.42 MHz	$120f_o$ = 1227.60 MHz
PRN code chipping rates, R_o = 10.23 MHz	P-code = R_o = 10.23 MHz and C/A = $R_o/10$ = 1.023 MHz	P-code = R_o = 10.23 MHz and C/A = $R_o/10$ = 1.023 MHz
Navigation data rate	50 Hz	50 Hz

summarizes frequency of carrier signal, PRN code chipping rate and navigation data rate for signals L1 and L2 respectively.

2.1.2 Navigation data

The navigation data is a 50 Hz signal which is the final output of GNSS receiver. According to (Tysowski 2009), the navigation data has three major parts - A, B and C. Part A holds GPS date, time and satellite's health condition. Part B holds orbital information, also called ephemeris data that provides the means for monitoring the satellite's position. Part C holds all other vital indications regarding the satellite, also called almanac.

2.1.3 Ranging code - P-code and C/A code

The P-code has a length of $2.35*10^{14}$ chip. The L2 carrier is frequency modulated by P-code while the L1 carrier is modulated with both P-code and C/A code. Unfortunately, P-code is accessible only for PPS (Picture Parameter Set) users whereas C/A code is available for SPS (Sequence Parameter Set) users (Johansson et al. 1998). Therefore, we will only focus on C/A code throughout this report.

C/A codes for each satellite is unique. These codes are also referred to as Gold codes or PRN sequences as described by (Gold 1967).

The architecture of a C/A code generator is shown in figure 2.4 as from (Gold & Dixon 1998). Two 10-bit Linear Feedback Shift Registers (LFSR) - G1 and G2 generate a sequence of maximum length of $N = (2^n - 1) = (2^{10} - 1) = (1024 - 1) = 1023$ bits. This sequence repeats every 1 ms (period, $T = 1$ ms), contributing to a nominal chipping rate of 1.023 MHz. Then the two resulting 1023 bits long codes are fed into modulo-2 sum to generate 1023 bits long C/A code. Initially, both G1 and G2 are all set to ones as all-zero state is illegal. The polynomial that describes the G1 and the G2 registers are usually in the form of

$$G = (1 + x^i) \tag{2.2}$$

where x^i represents the output from the i^{th} cell of the corresponding shift register.

The polynomial describing the G1 register is given by

$$G1 = 1 + x^3 + x^{10} \tag{2.3}$$

which means the output from cell 3 and cell 10 are fed into a modulo-2 sum. Then the output from this modulo-2 sum is sent as a feedback into the cell 1.

Similarly, the polynomial that describes G2 register is given by

$$G2 = 1 + x^2 + x^3 + x^6 + x^8 + x^9 + x^{10} \tag{2.4}$$

Here, outputs from cell 2, cell 3, cell 6, cell 8, cell 9 and cell 10 are fed back into a modulo-2 sum. Then the output from modulo-2 sum is sent as a feedback into the cell 1.

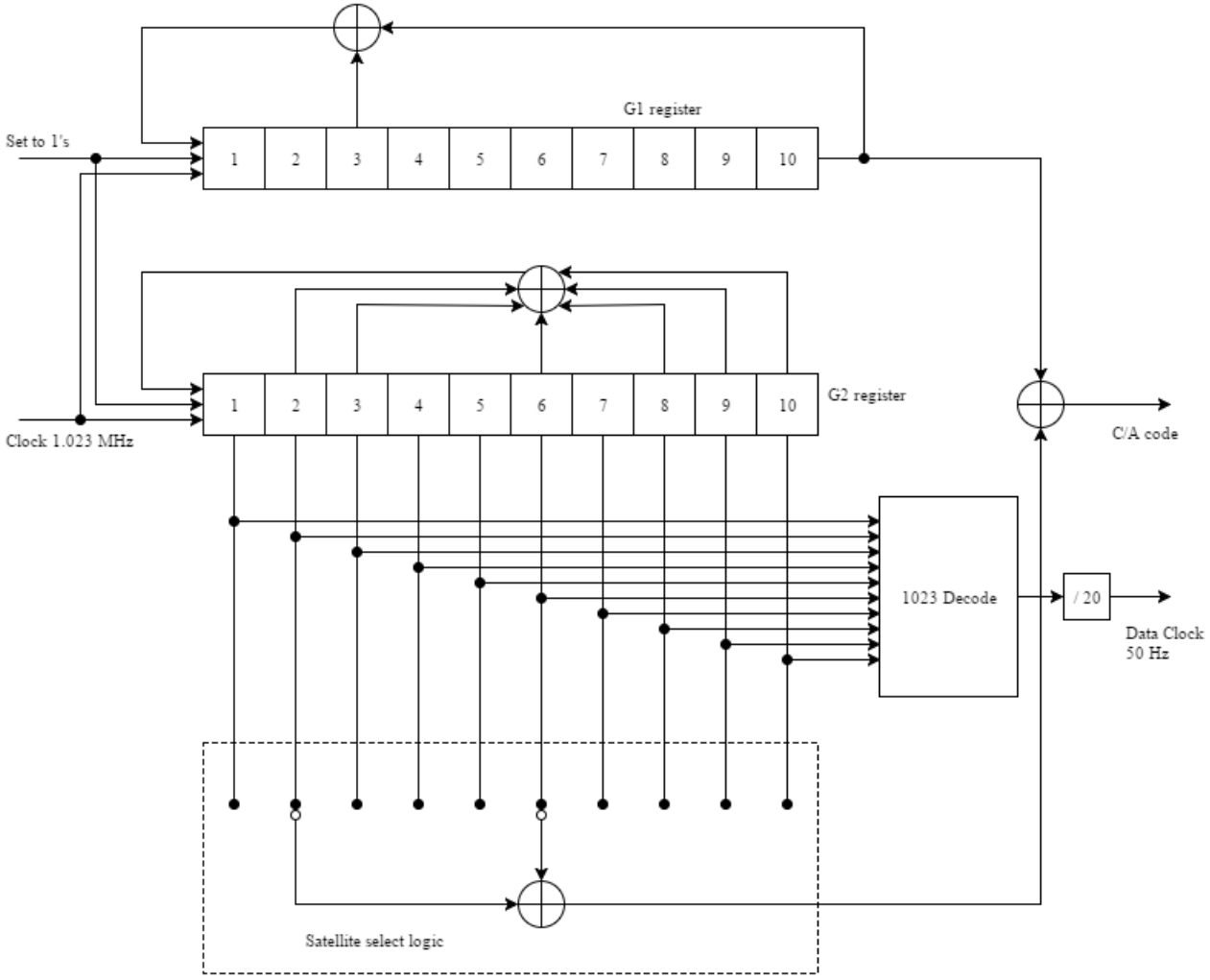


Figure 2.4: C/A code architecture (Gold & Dixon 1998)

Now, to generate the satellite specific C/A code, an exclusive-or is applied between the output from G1 register and the delayed version of output from G2 register. For delayed version of G2 register, an exclusive-or is applied between the two selected states, also called the selected phases. The phase selection in G2 for each satellite is illustrated in table 2.2, derived from (Klobuchar 1987).

Table 2.2: C/A code phase assignments for respective satellite (Klobuchar 1987)

Satellite ID	**GPS PRN signal number**	**Code phase selection G2**	**Code delay chips**
1	1	$2 \oplus 6$	5
2	2	$3 \oplus 7$	6
3	3	$4 \oplus 8$	7
4	4	$5 \oplus 9$	8
5	5	$1 \oplus 9$	17
6	6	$2 \oplus 10$	18
7	7	$1 \oplus 8$	139
8	8	$2 \oplus 9$	140
9	9	$3 \oplus 10$	141
10	10	$2 \oplus 3$	251
11	11	$3 \oplus 4$	252
12	12	$5 \oplus 6$	254
13	13	$6 \oplus 7$	255
14	14	$7 \oplus 8$	256
15	15	$8 \oplus 9$	257
16	16	$9 \oplus 10$	258
17	17	$1 \oplus 4$	469
18	18	$2 \oplus 5$	470
19	19	$3 \oplus 6$	471
20	20	$4 \oplus 7$	472
21	21	$5 \oplus 8$	473
22	22	$6 \oplus 9$	474
23	23	$1 \oplus 3$	509
24	24	$4 \oplus 6$	512
25	25	$5 \oplus 7$	513
26	26	$6 \oplus 8$	514
27	27	$7 \oplus 9$	515
28	28	$8 \oplus 10$	516
29	29	$1 \oplus 6$	859
30	30	$2 \oplus 7$	860
31	31	$3 \oplus 8$	861
32	32	$4 \oplus 9$	862
33*	33	$5 \oplus 10$	863
34*	34	$4 \oplus 10$**	950**
35*	35	$1 \oplus 7$	947
36*	36	$2 \oplus 8$	948
37*	37	$4 \oplus 10$**	950**
*Through satellite 33 to 37, the PRN codes are reserved for other uses as ground transmitter **C/A codes for 34 and 37 are identical			

Correlation properties of C/A code

For modulation, gold codes are used because of their correlation properties (Parkinson 1996). The two main important correlation properties of the gold codes are

- Nearly no cross correlation:
  For C/A codes C^i and C^k for satellites i and k, the cross correlation is defined by

$$r_{ik}(m) = \sum_{l=0}^{1022} C^i(l)C^k(l+m) \approx 0, \text{for all } m \tag{2.5}$$

- Nearly no correlation except for zero lag:
  All C/A codes are nearly uncorrelated with each other, except for zero lag. So it makes easy to find out even when similar codes are perfectly aligned. For C/A codes C^k for satellite k, the autocorrelation property is defined by

$$r_{kk}(m) = \sum_{l=0}^{1022} C^k(l)C^k(l+m) \approx 0, \text{for } |m| \geq 1 \tag{2.6}$$

  For more details on correlation properties, read through (Parkinson 1996)

2.2 Doppler frequency shift

The satellites and the receivers are in continuous motion. This causes a shift in Doppler frequency. It affects both acquisition and tracking of the GPS signal. The maximum Doppler frequency shift for a stationary receiver antenna is at around ±5 KHz whereas for a moving receiver antenna, the maximum Doppler frequency shift is ±10 KHz (Johansson et al. 1998). If deviation in frequency is not corrected, GPS receiver will eventually lose track of the satellite and no data will be received.

The misalignment between the incoming codes and the locally generated codes are caused due to the Doppler frequency shift. The frequency of the carrier signal for L1 is $f_c = 1575.42$ MHz while the nominal chipping rate of C/A code is only $f_{CA} = 1.023$ MHz. This means f_{CA} is $f_c/f_{CA} = 1540$ times smaller than f_c. Therefore, Doppler frequency shift has a small affect on the C/A code which is about ±10 KHz/1540 = ±6.5 Hz for a moving GPS receiver. Likewise, for a stationary GPS receiver, the Doppler frequency shift is only about ±5 KHz/1540 = ±3.2 Hz (Borre et al. 2007).

2.3 Code tracking

One problem in the receiver end while GPS signal tracking is to maintain the phase of local PRN aligned with the phase of incoming PRN. Because of the Doppler frequency shift, the phase of PRN received and the phase of locally generated PRN gets misaligned throughout the receiving process. The solution to this problem is to use a DLL (Delay Locked Loop) as seen from chapter 1, figure 1.2. In this method, three replicas of PRN codes - early PRN, prompt PRN and late PRN codes are generated and correlated with the incoming signal.

The basic block diagram of code tracking loop can be seen in figure 2.5. In this figure, the local oscillator generates a perfectly aligned replica of carrier signal. Here, the incoming signal is multiplied with the local carrier signal and then with replicas of PRN codes. The results of multiplication are integrated and dumped to obtain the correlation values. The result of these correlators is a numerical value which tells how the replica codes correlate with the incoming code.

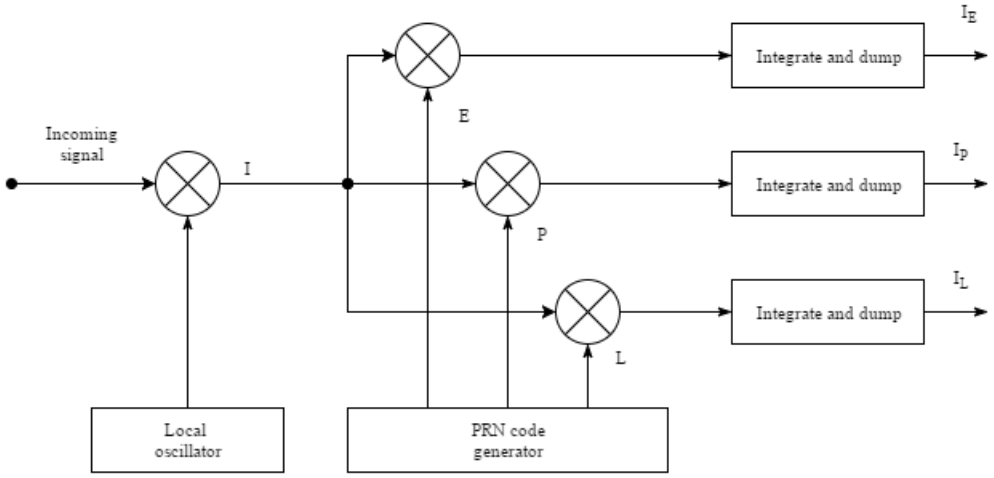


Figure 2.5: Basic block diagram of code tracking loop (Borre et al. 2007)

Based on the outputs of correlators - I_E, I_P and I_L, we determine if the phase of PRN code is needed to be shifted to right or left (Peterson & Ziemer 1985). The example A in figure 2.6 and the example B in figure 2.7 give a clear understanding on how this method works. In example A, the late PRN code has the highest correlation with the PRN code in the incoming signal. Therefore, the code phase must be decreased, in other words, the code must be shifted to the right. Similarly, in example B, the prompt PRN code has the highest correlation with

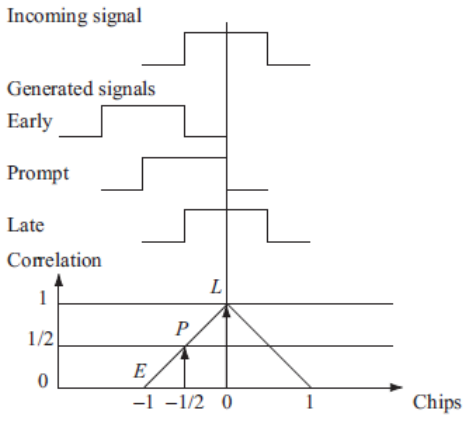


Figure 2.6: Comparison between the three outputs from the correlators, example A (Borre et al. 2007)

the PRN code in the incoming signal. Here, the local code is perfectly aligned with the incoming code i.e. shifting is not required.

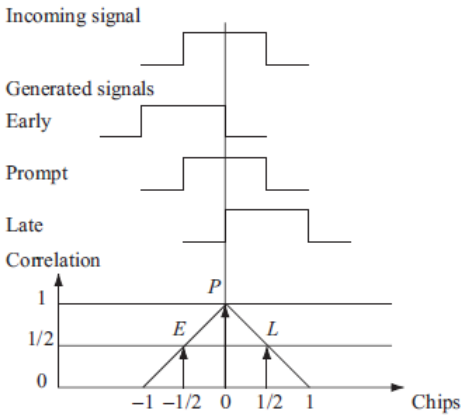


Figure 2.7: Comparison between the three outputs from the correlators, example B (Borre et al. 2007)

The block diagram in figure 2.5 is useful only in the optimal case i.e. when frequency and phase of the carrier signal is properly locked. However, this is never true in the real condition. Due to the Doppler frequency shift and other noises in the channel, a code phase error is

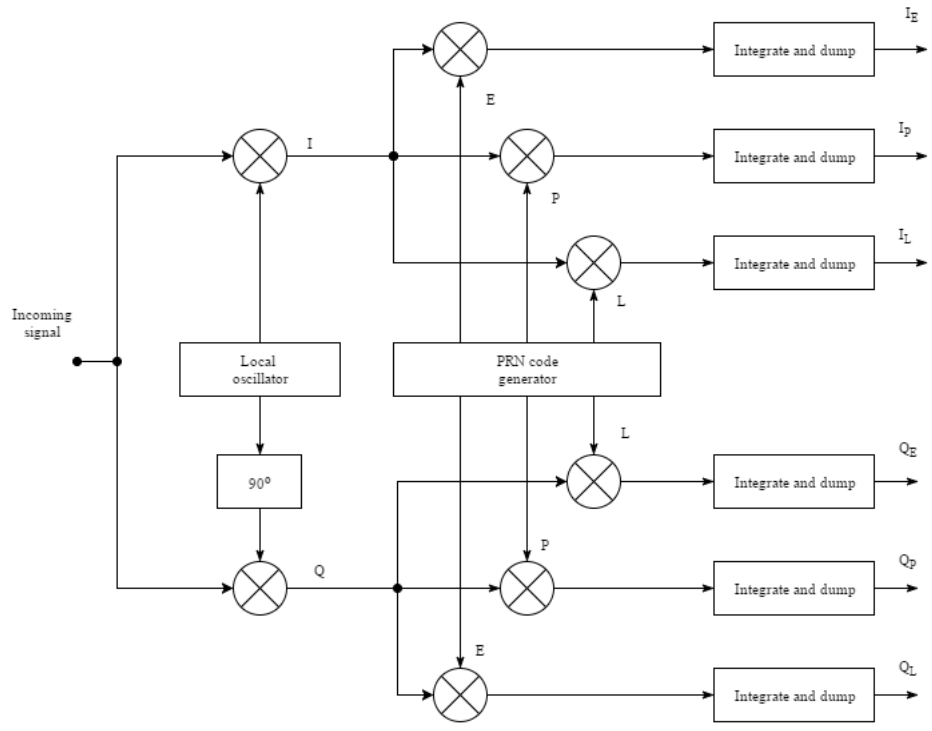


Figure 2.8: DLL code tracking block diagram with six correlators (Johansson et al. 1998)

experienced in the incoming code. Hence, a more sophisticated design with six correlators like in figure 2.8 (Johansson et al. 1998) is required. This system is independent of the phase of local code. If the incoming code and the local code are perfectly aligned, all energy will be transferred in the in-phase arm. But if these signals are not aligned, then energy gets distributed between the in-phase arm and the quadrature arm. The upper half of figure 2.8 is called the in-phase arm while the bottom half is called the quadrature arm. The signal I is also known as in-phase signal whereas the signal Q is known as quadrature signal. Here, the signal Q is multiplied with three code replicas which are integrated and dumped. The final output of six correlators are I_E, I_P, I_L, Q_E, Q_P and Q_L.

Once the outputs from six correlators are obtained, they are sent to code loop discriminator. The code loop discriminator is based on some algorithm as explained in table 2.3. The output of this discriminator is a control signal that allows the PRN code generator to either shift the phase to right or left. If the codes are perfectly aligned, the discriminator feedbacks a control signal that allows the PRN code generator not to shift the phase of PRN code.

The table 2.3 consists of one coherent discriminator while the rest are non-coherent discriminators.

Table 2.3: Different types of discriminator (Borre et al. 2007)

Type	**Discriminator, *D***
Coherent	$D = I_E - I_L$
Early minus late power	$D = (I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)$
Normalized early minus late power	$D = \frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{(I_E^2 + Q_E^2) + (I_L^2 + Q_L^2)}$
Dot products	$D = I_P(I_E - I_L) + Q_P(Q_E - Q_L)$

- Coherent discriminator:
  The coherent discriminator is the simplest of all discriminators. This method does not require the quadrature arm. This method can be implied only when the local carrier signal is perfectly aligned with the incoming carrier signal.
- Early minus late power (Non coherent discriminator):
  The response of this type of discriminator is almost the same as that of coherent discriminator within $\pm\frac{1}{2}$ nominal chipping rate of C/A code.
- Normalized early minus late power (Non coherent discriminator):
  This method is the best of the rest of discriminators. The response of this discriminator

is independent of the performance of PLL (Phase Locked Loop) as it invokes both in-phase arm and quadrature arm. Furthermore, this helps the DLL to maintain track of the signal even when the chip error is larger than $\pm\frac{1}{2}$ nominal chipping rate of C/A code. Thus, for simulation, the normalized early minus late power - a non coherent discriminator is considered.

- Dot product (Non coherent discriminator):
  This method is the only discriminator that uses all outputs from the six correlators.

The further extension of figure 2.8 with an inclusion of code loop discriminator is illustrated in figure 2.9. This figure represents a complete block diagram for a code tracking loop.

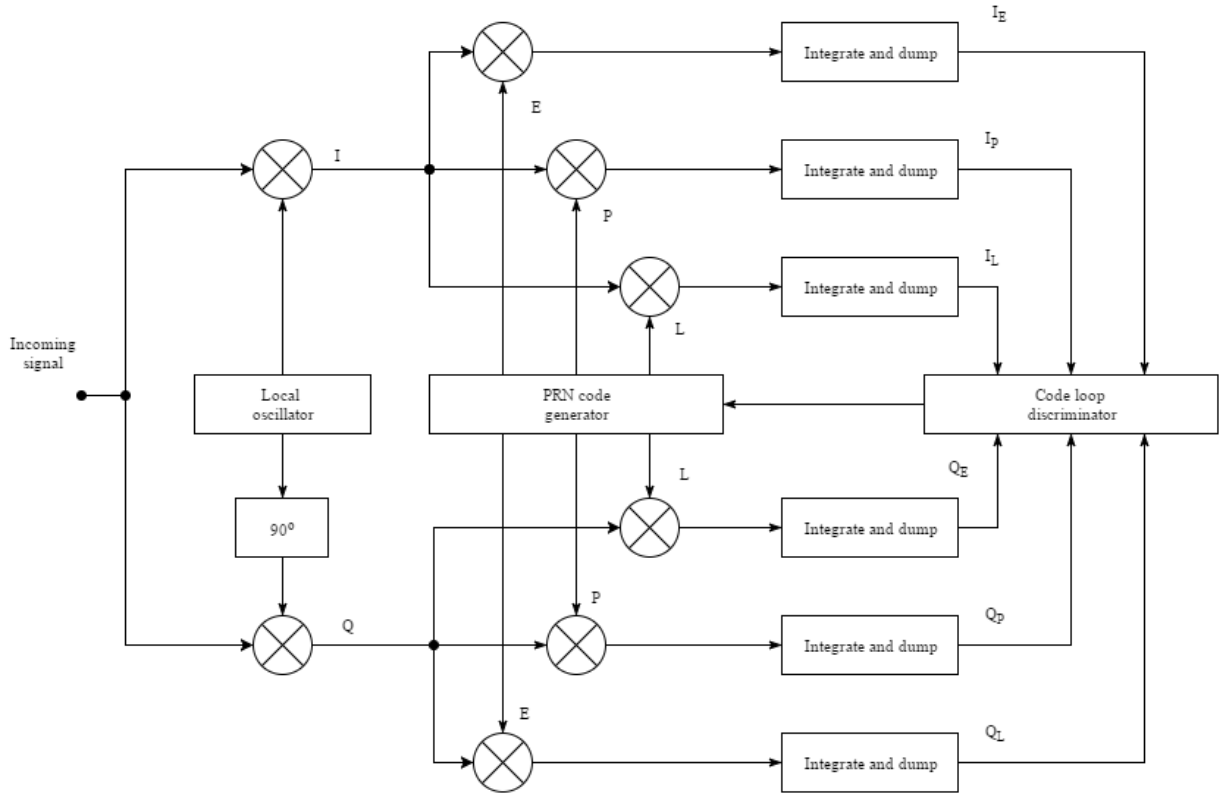


Figure 2.9: A complete block diagram of a code tracking loop (Johansson et al. 1998)

2.4 Carrier tracking

The other problem in tracking is to keep track phase and frequency of carrier signal. The change in carrier frequency makes the receiver to lose track of the satellites. Based on figure 1.2 from chapter 1, a PLL or FLL (Frequency Locked Loop) is used to track a carrier signal.

The basic form of PLL carrier tracking loop is shown in figure 2.10 as presented in (Borre et al. 2007). The incoming signal is first multiplied with the replica of incoming carrier signal

and then with the local PRN code. This process is performed to wipe off carrier signal and PRN code of the incoming signal. The output after the multiplication is sent to carrier loop discriminator to determine the carrier phase error. This phase error is filtered out by carrier loop filter. Finally, the output from the carrier loop filter is used as a feedback to a carrier generator or NCO (Numerically Controlled Oscillator) that produces a perfectly aligned carrier wave signal as compared with the incoming carrier wave signal.

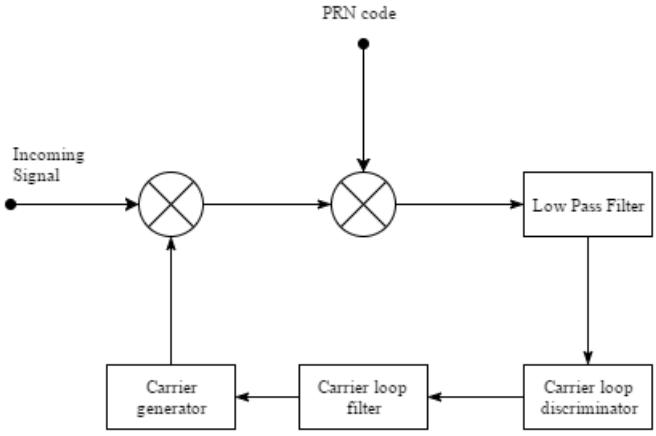


Figure 2.10: Block diagram of basic PLL carrier tracking loop (Borre et al. 2007)

Figure 2.10 represents a basic form for tracking the carrier wave signal. This system is sensitive to 180° phase shifts. The solution to this problem is an extension version of figure 2.10 which is illustrated in figure 2.11. This figure uses Costas loop (Borre et al. 2007).

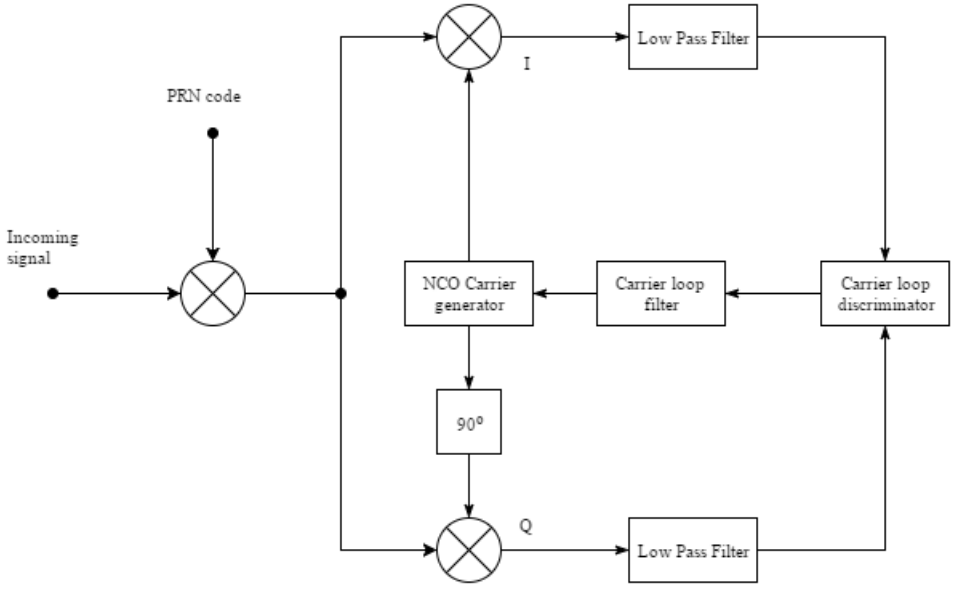


Figure 2.11: Block diagram of carrier tracking loop

The multiplication in the in-phase arm returns the following equation

$$D^k(n)cos(\omega_{IF}n)cos(\omega_{IF}n+\varphi) = \frac{1}{2}D^k(n)cos(\varphi) + \frac{1}{2}D^k(n)cos(2\omega_{IF}n+\varphi) \tag{2.7}$$

where,

D^k	Data from k^{th} satellite
ω_{IF}	Incoming signal Frequency (IF) in radians
φ	Phase difference between the phase of incoming signal and phase of local replica

Similarly, the result after product in quadrature arm returns

$$D^k(n)sin(\omega_{IF}n)sin(\omega_{IF}n+\varphi) = \frac{1}{2}D^k(n)sin(\varphi) + \frac{1}{2}D^k(n)sin(2\omega_{IF}n+\varphi) \tag{2.8}$$

These resulted signals from in-phase arm and quadrature arm are filtered out by a LPF (Low Pass Filter) to get the following two signals.

$$I^k = \frac{1}{2}D^k cos(\varphi) \tag{2.9}$$

$$Q^k = \frac{1}{2}D^k sin(\varphi) \tag{2.10}$$

Further calculation to get φ can be done as

$$\frac{Q^k}{I^k} = \frac{\frac{1}{2}D^k sin(\varphi)}{\frac{1}{2}D^k cos(\varphi)} = tan(\varphi) \tag{2.11}$$

$$\varphi = \arctan\left(\frac{Q^k}{I^k}\right) \tag{2.12}$$

The equation 2.12 gives a clear idea that the phase error can be minimized if the correlation in the quadrature arm is minimized. Some other discriminator can be seen in table 2.4. The output for type 1 discriminator is proportional to sin(φ) while the output for type 2 discriminator is proportional to sin(2φ).

Table 2.4: Costas loop discriminator types (Kaplan & Hegarty 2005)

Type	**Discriminator**
1.	$D = sign(I^k)Q^k$
2.	$D = I^k Q^k$
3.	$D = \arctan\left(\frac{Q^k}{I^k}\right)$

The performance of the three discriminators mentioned in table 2.4 is presented in figure 2.12. From this figure, it can be clearly understood that discriminator outputs are zero whenever the phase error is $-180^o, 0^o$ and $+180^o$. Thus, the Costas loop is insensitive to the $\pm 180^o$ phase shifts in case of a navigation bit transition (Kaplan & Hegarty 2005). The most preferable method for computing is the third method mentioned in table 2.4 which is also applied in simulation. Unfortunately, this method consumes the most time.

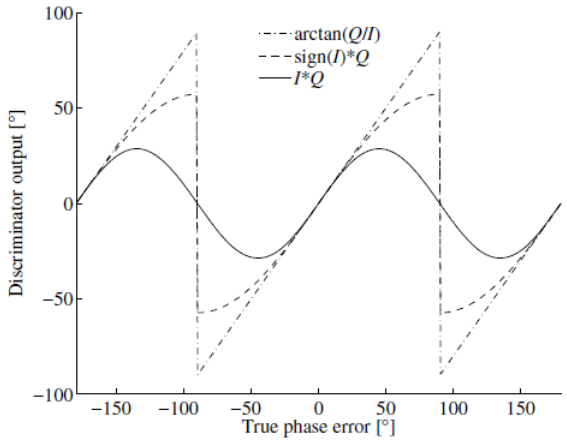


Figure 2.12: Performance of different Costas loop discriminators (Kaplan & Hegarty 2005)

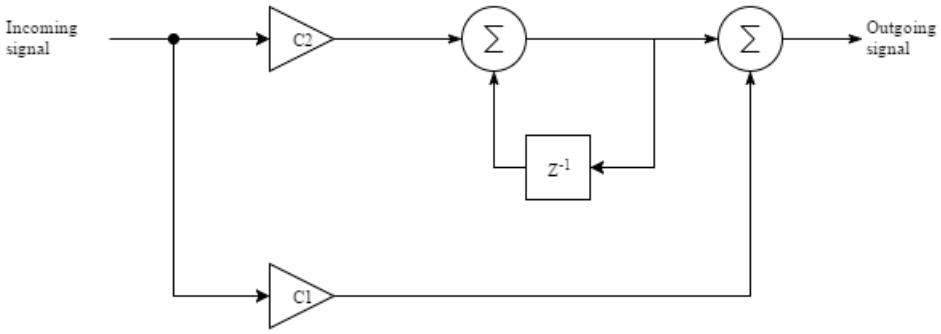


Figure 2.13: Second Order PLL (Chung, et al. 1993)

After computing the phase error using the discriminator, it is filtered out by a carrier loop filter. To design a carrier loop filter, a second order PLL is used as in figure 2.13. According to (Chung et al. 1993), the transfer function of a linearized analog PLL is given by

$$H(s) = \frac{K_d F(s) N(s)}{1 + K_d F(s) N(s)} \tag{2.13}$$

where,

K_d gain of phase discriminator

$F(s)$ transfer function of filter defined as $\frac{1}{s}\frac{\tau_2 s+1}{\tau_1}$

$N(s)$ transfer function of NCO defined as $\frac{K_o}{s}$

Substituting the values of $F(s)$ and $N(s)$ in equation 2.13 and considering natural frequency $\omega_n = \sqrt{\frac{K_o K_d}{\tau_1}}$ and damping ratio $\zeta = \frac{\tau_2 \omega_n}{2}$, we get the following equation.

$$H(s) = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + s\zeta\omega_n s + \omega_n^2} \tag{2.14}$$

The equation 2.14 is still in analog form. To convert this into digital form, bilinear transformation is applied which yields another equation (Borre et al. 2007)

$$H_1(z) = \frac{(4\zeta\omega_n T + (\omega_n T)^2) + 2(\omega_n T)^2 z^{-1} + ((\omega_n T)^2 - 4\zeta\omega_n T)z^{-2}}{(4 + 4\zeta\omega_n T + (\omega_n T)^2) + (2(\omega_n T)^2 - 8)z^{-1} + (4 - 4\zeta\omega_n T + (\omega_n T)^2)z^{-2}} \tag{2.15}$$

Now, the digital form of transfer function of filter and NCO are given by

$$F(z) = \frac{(C_1 + C_2) - C_1 z^{-1}}{1 - z^{-1}} \tag{2.16}$$

$$N(z) = \frac{K_o z^{-1}}{1 - z^{-1}} \tag{2.17}$$

The digital form of equation 2.13 can be presented as

$$H(z) = \frac{K_d F(z) N(z)}{1 + K_d F(z) N(z)} \tag{2.18}$$

Substituting equation 2.16 and equation 2.17 in equation 2.18, we get

$$H_2(z) = \frac{K_o K_d (C_1 + C_2) z^{-1} - K_o K_d C_1 z^{-2}}{1 + (K_o K_d (C_1 + C_2) - 2)z^{-1} + (1 - K_o K_d C_1)z^{-2}} \tag{2.19}$$

From equation 2.15 and 2.19, we obtain the values of coefficients - C_1 and C_2 as

$$C_1 = \frac{1}{K_o K_d}\frac{8\zeta\omega_n T}{4 + 4\zeta\omega_n T + (\omega_n T)^2} \tag{2.20}$$

$$C_2 = \frac{1}{K_o K_d}\frac{4(\omega_n T)^2}{4 + 4\zeta\omega_n + (\omega_n T)^2} \tag{2.21}$$

where natural frequency ω_n is given by

$$\omega_n = \frac{8\zeta B_L}{4\zeta^2 + 1} \tag{2.22}$$

where B_L is the noise bandwidth in the loop. Altering the values of damping ratio ζ and

natural frequency ω_n also alter the response time of digital carrier loop filter.

Finally, a complete block diagram of code and carrier tracking loop for GNSS receiver is illustrated in figure 2.14. Here, the integrator acts like a LPF with a stop band frequency of 1 KHz and the high frequencies are filtered out. Therefore, the LPF can be left out without much trouble. This final figure will be the basis for all designs in simulation.

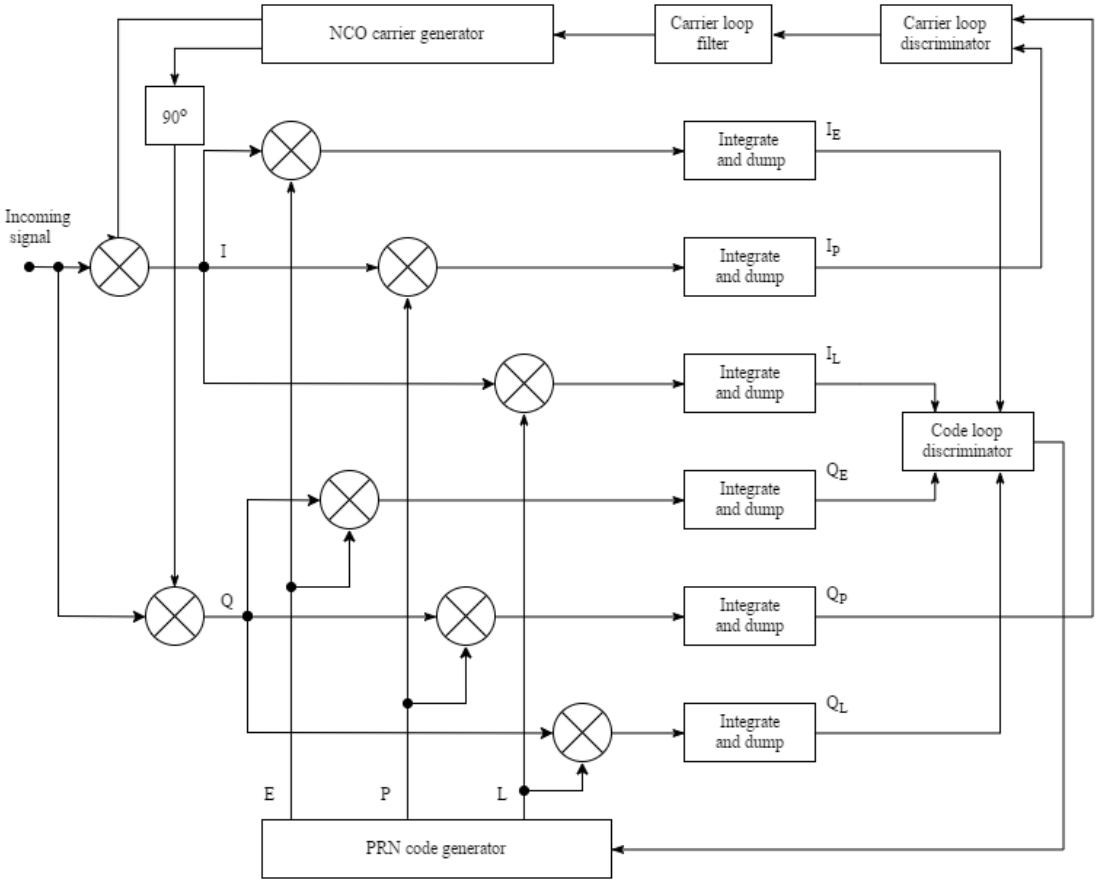


Figure 2.14: Block diagram of code and carrier tracking loop (Borre et al. 2007)

Chapter 3

Tracking Algorithm Implementation and Simulation Using Simulink Toolbox

The tracking algorithm discussed in chapter 2 is implemented and simulated using simulink toolbox. The simulink models designed in MATLAB are attached in appendix A.

The first section focuses on simulation of GPS L1 signal based on figure 2.2 from chapter 2. It includes simulation of PRN sequence for satellite id #1, navigation data, carrier wave signal, results of modulo-2 sum between PRN sequence and navigation data. Then the simulation result for GPS L1 signal is presented with some discussion.

The second section focuses on implementation and simulation of code tracking algorithm based on figure 2.14. First, we generate three replicas of incoming PRN code. Then a code phase discriminator is designed followed by a code phase alignment program. Finally, we verify the code tracking algorithm by comparing the phase of replica PRN with the phase of incoming PRN.

The third section focuses on implementation and simulation of carrier tracking algorithm based on figure 2.14. First, a carrier loop discriminator is designed followed by a design of carrier loop filter. Then we look at the locally generated carrier wave signal and verify the carrier tracking algorithm by comparing the phase of local carrier signal with the phase of incoming carrier signal.

3.1 Simulation of GPS signal

In this project, a real GPS data was not available. So, a random signal with a sequence of 1's and 0's is considered to be a navigation data and is used for generating a L1 signal, sampled at 5 MHz. Based on figure 2.2 from chapter 2, a simulation model was designed. The simulink design can be referred in appendix A, figure A.1. To generate a L1 signal, the following steps are followed:

- Generate PRN code, generate navigation data and generate carrier wave signal.
- Perform an exclusive-or between PRN sequence and navigation data.
- Perform BPSK modulation between the carrier wave signal and the result from exclusive-or between PRN sequence and navigation data.
- Generate L1 signal.

3.1.1 Simulation of PRN sequence for satellite id #1

For PRN sequence simulation, the polynomials G1 and G2 as described in chapter 2, section 2.1.3 are applied. Neither a real GPS signal nor a satellite id are available. Thus, we choose satellite id #1 (as from chapter 2, table 2.2) such that an exclusive-or is performed between cell 2 and cell 6 in register G2. The code phase selection for other satellites are also described in chapter 2, table 2.2. The configuration parameter blocks for registers G1 and G2 are presented in appendix A, figure A.3 and appendix A, figure A.4 respectively.

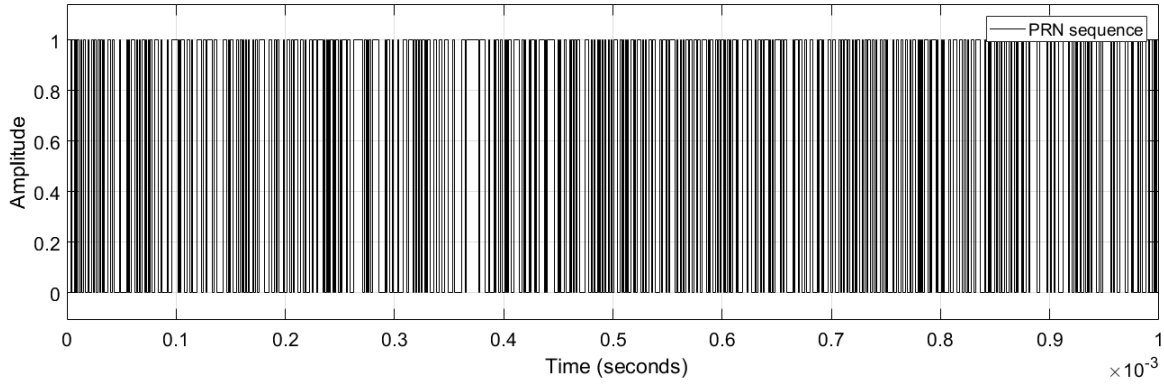


Figure 3.1: Simulation result of PRN sequence in 1 ms

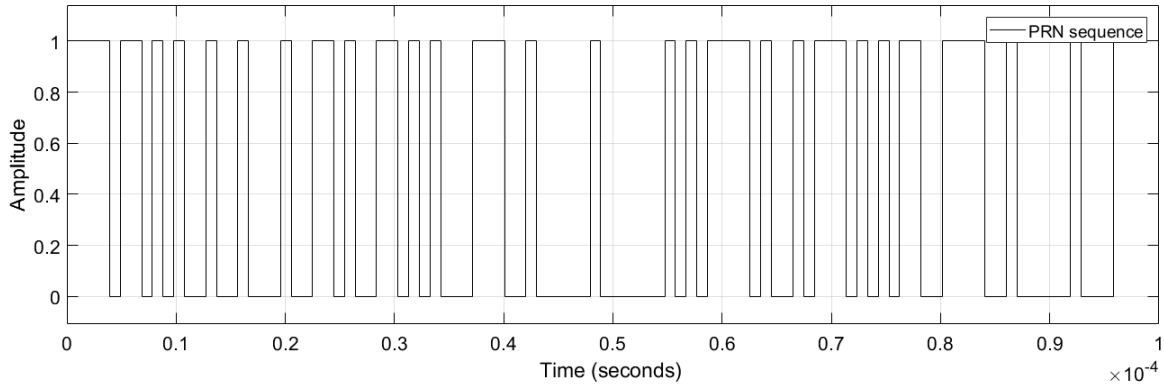


Figure 3.2: Simulation result of PRN sequence showing first 100 bits

Now, we model a PRN sequence generator. The PRN sequence generator found in simulink library is used to generate the sequence. This model is presented in appendix A, figure A.2.

The results of simulation for 1 ms can be seen in figure 3.1. This figure consists of a sequence of 1023 bits in 1 ms such that its nominal chipping rate equals 1.023 MHz. This sequence of 1's and 0's is repeated every 1 ms. The first 100 bits of a PRN sequence for satellite id #1 can be seen in figure 3.2.

3.1.2 Simulation of navigation data

To generate a navigation data of 50 Hz, we use the PRN sequence generator. This time, the polynomials G1 and G2 are randomly defined as a navigation data with a random sequence of 1's and 0's is required. The result from this model (presented in appendix A, figure A.5) generates a sequence of 1's and 0's as shown in figure 3.3. For every 0.1 second, 5 bits of data is generated. Thus, for every 1 second, 5 bits $*$ 10 = 50 bits are generated resulting in 50 Hz frequency for the navigation data.

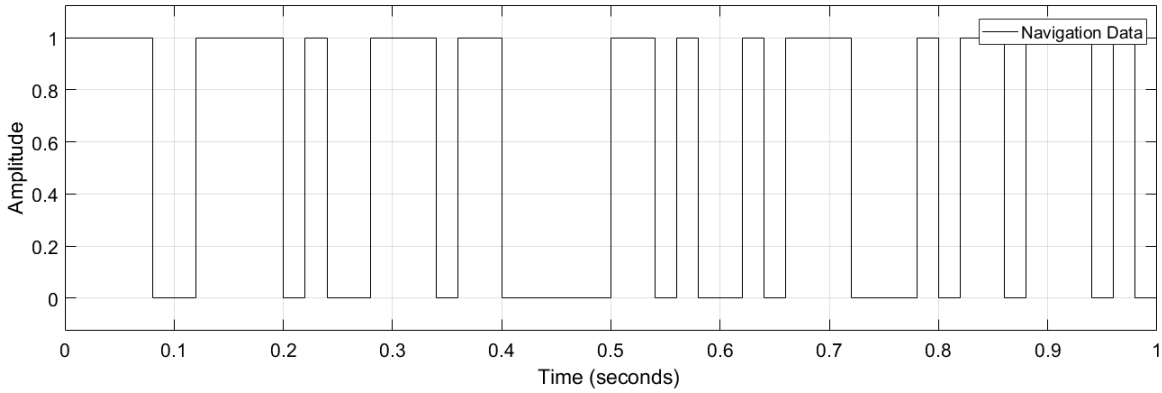


Figure 3.3: Simulation result of navigation data (5 bits of data is generated in every 0.1 seconds leading to generation of 50 bits in every 1 second)

3.1.3 Simulation of carrier wave signal

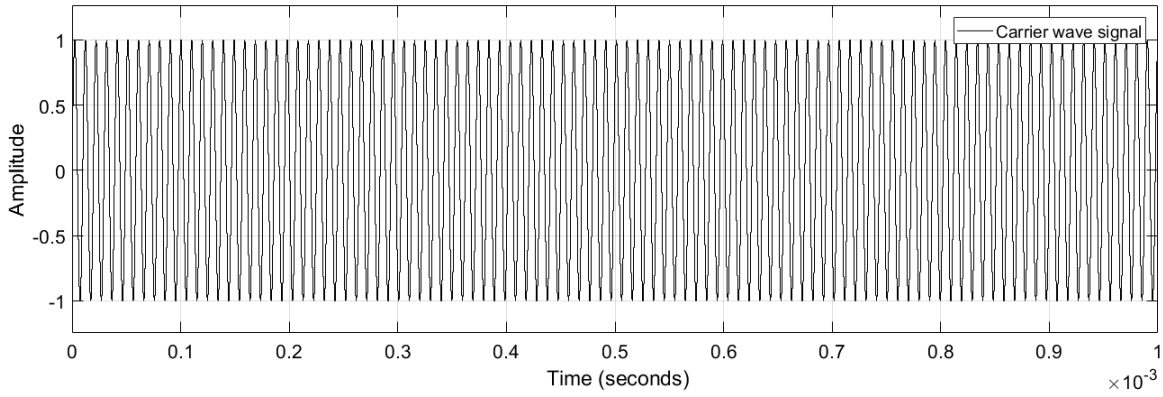


Figure 3.4: Simulation result of carrier wave signal output in 1 ms

The carrier wave of L1 signal has a frequency of 1575.42 MHz (chapter 2, table 2.1). This signal is generated using a NCO block. The simulink model to generate the carrier wave is presented in appendix A, figure A.6. Using this model, a carrier wave signal is generated. The simulation result is illustrated in figure 3.4. Even though, a simulation is run for only 1 ms, a clear picture of a carrier wave cannot be seen. Thus, for a better picture, a magnified version of figure 3.4 is presented in figure 3.5. From this picture, we clearly see that the carrier wave signal resembles a sine wave.

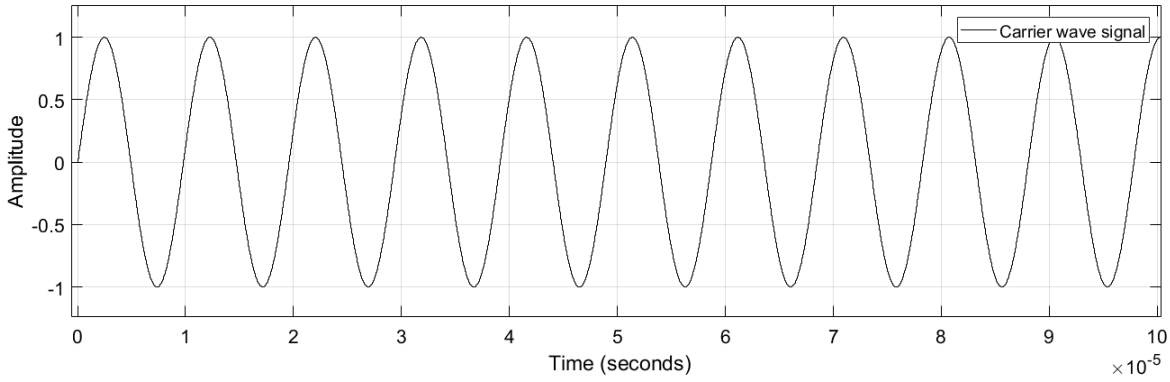


Figure 3.5: Simulation result of carrier wave signal output - magnified

3.1.4 Simulation of output from modulo-2 sum

Until now, we have generated a PRN sequence, a navigation data and a carrier wave signal in simulink toolbox. An exclusive-or is applied between a PRN sequence and a navigation data. The simulation result of modulo-2 sum or exclusive-or is illustrated in figure 3.6. When the values of PRN sequence and navigation data are same, the exclusive-or outputs "0" and when the values of PRN sequence and navigation data are different, the exclusive-or outputs "1". The final simulation result from a modulo-2 sum is a sequence of signal with values ranging between 0 and 1.

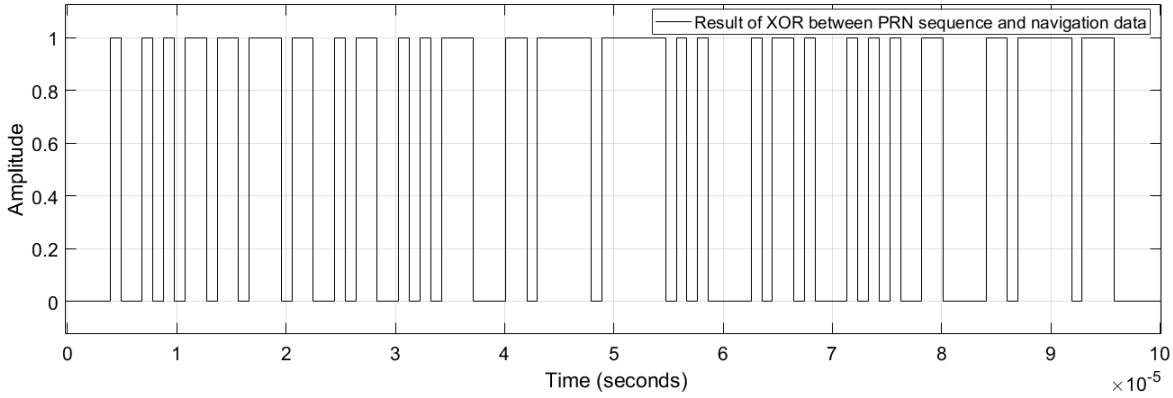


Figure 3.6: Simulation result of exclusive-or between PRN sequence and navigation data

3.1.5 Simulation of L1 signal

Now, a BPSK modulation is applied between the carrier wave signal and the results of modulo-2 sum to obtain the L1 signal. The simulink model to generate L1 signal is presented in appendix A, figure A.1. This model is completely based on the block diagram presented in chapter 2, figure 2.2. From figure 3.6, we see that the result of exclusive-or are mapped into values {0,1}. After BPSK modulation with carrier wave signal, a L1 signal is generated with values {-1,1}. Thus, we conclude that a BPSK modulation maps a signal with values {0,1} into {-1,1}. The final simulation result of L1 signal can be seen in figure 3.7 which is similar to the one in figure 2.3 from chapter 2. A more detailed figure of L1 signal can be seen in figure 3.8 where the signal is sampled at 5 MHz.

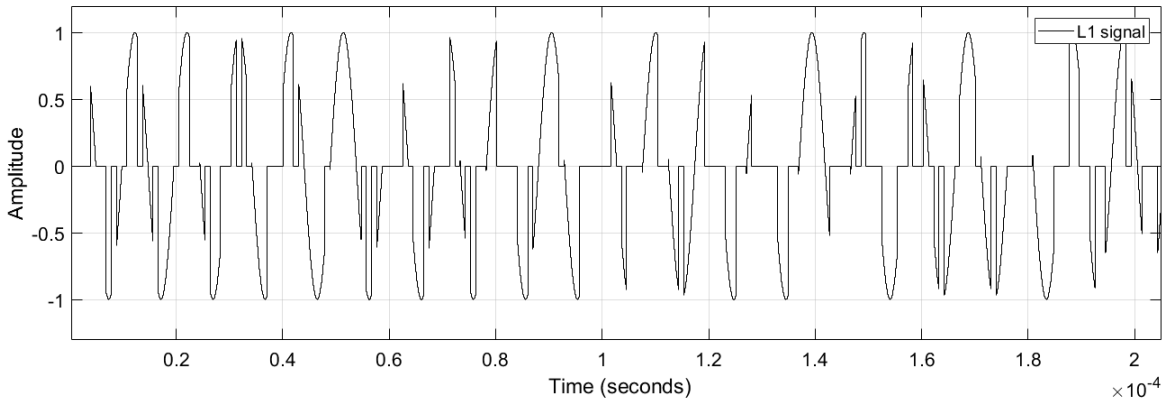


Figure 3.7: Simulation result of L1 signal

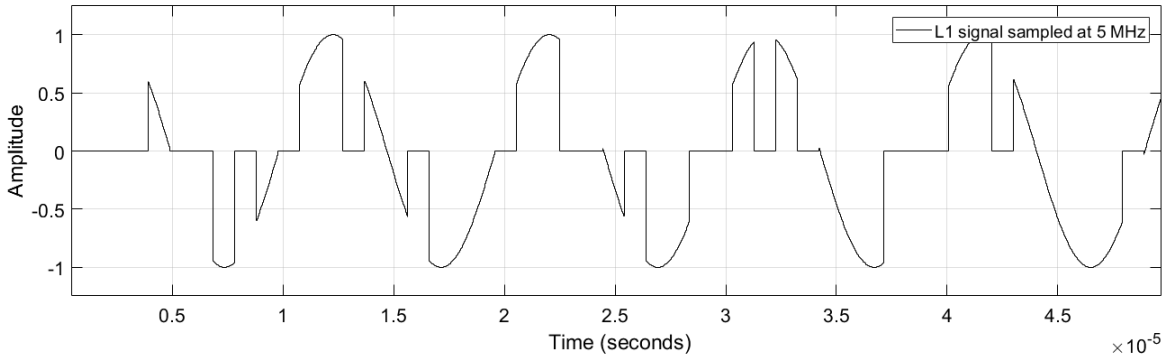


Figure 3.8: Simulation result of L1 signal at 5 MHz

3.2 Code tracking

The simulation design of code tracking loop is based on figure 2.14 from chapter 2. The phase of incoming PRN code and the phase of locally generated PRN code gets misaligned because of the Doppler frequency shift. A DLL is used to solve this problem.

In this method, the following steps are followed:

- Multiply incoming signal and local carrier wave. This result is again multiplied with three replicas of PRN code - early, prompt and late PRN codes. Then a correlation value is obtained.
- Based on correlation values, a decision is made whether to shift the PRN code to right or left. With proper phase shifting in the code generator, a perfectly aligned PRN sequence is generated.

3.2.1 Simulation of early, prompt and late PRN code

The nominal chipping rate of a PRN sequence is 1023 bits per milliseconds i.e. it has a frequency of 1.023 MHz. The three replicas of incoming PRN codes are generated locally. These three replicas are equally spaced by $\pm\frac{1}{2}$ a nominal chipping rate of a PRN code. It means that each PRN replica is spaced by ± 511 bits. A simulink model (attached in appendix A, figure A.7) is designed based on figure 2.14 from chapter 2. The early PRN is delayed by 1022 bits and thus, it starts after 0.999 ms $\approx$ 1 ms. The simulation result of an early PRN

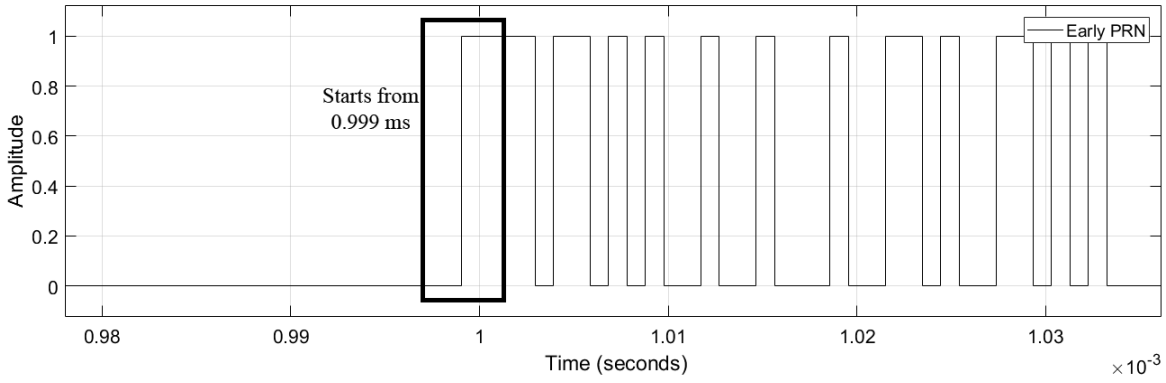


Figure 3.9: Simulation result of early PRN code from a local code generator

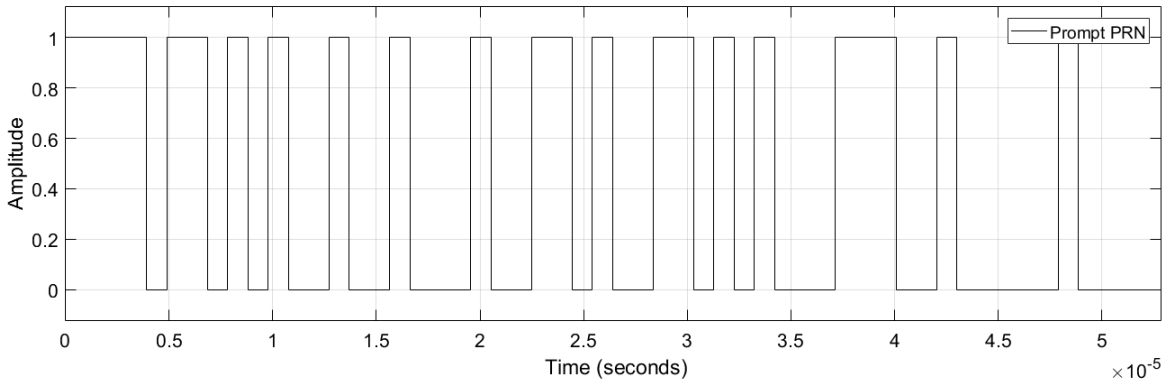


Figure 3.10: Simulation result of prompt PRN code from a local code generator

code is shown in figure 3.9. The prompt PRN is generated without any delays and thus, it starts from 0 s. The simulation result of a locally generated prompt PRN is shown in figure 3.10. The late PRN is delayed by 511 bits and thus, it starts after 499.5 ms $\approx$ 0.5 ms. The simulation result of a locally generated late PRN is shown in figure 3.11. Hence, an equally spaced three replicas of PRN code are generated with a frequency of 1.023 MHz.

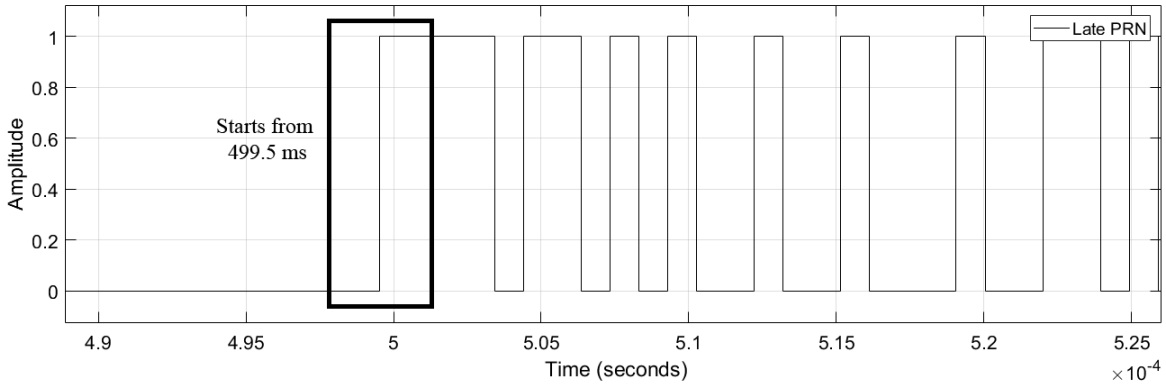


Figure 3.11: Simulation result of late PRN code from a local code generator

3.2.2 Design of a code loop discriminator

As explained in chapter 2, section 2.3 - the outputs from the six correlators are sent to the code loop discriminator to calculate the code phase error. The algorithm for code loop discriminator is based on "normalized early minus late power discriminator", also explained in chapter 2, table 2.3. This algorithm is shown in equation 3.1.

$$D = \frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{(I_E^2 + Q_E^2) + (I_L^2 + Q_L^2)} \tag{3.1}$$

An alternative to "early minus late power discriminator" is shown in equation 3.2 defined by (Johansson et al. 1998). Both algorithms expressed in equation 3.1 and equation 3.2 have a similar performance.

$$D = \sqrt{\frac{I_E^2 + Q_E^2}{I_L^2 + Q_L^2}} \tag{3.2}$$

The phase error obtained by using both algorithms is presented in figure 3.12. The code phase error waveform clearly shows that the performance from both algorithms are exactly the same. The only difference is the guideline. For equation 3.1, "amplitude = 0" on the Y-axis is the guideline to maintain the phase of PRN code. For equation 3.2, "amplitude = 1" on the Y-axis is the guideline to maintain the phase of PRN code. So either of the algorithm can be used for simulation. In this project, equation 3.2 is followed to calculate the code phase error. With every iteration, a code phase error is calculated. This error is used as a feedback to the "code phase alignment correction block" (as shown in appendix A, figure A.8) to make necessary phase shift in PRN codes.

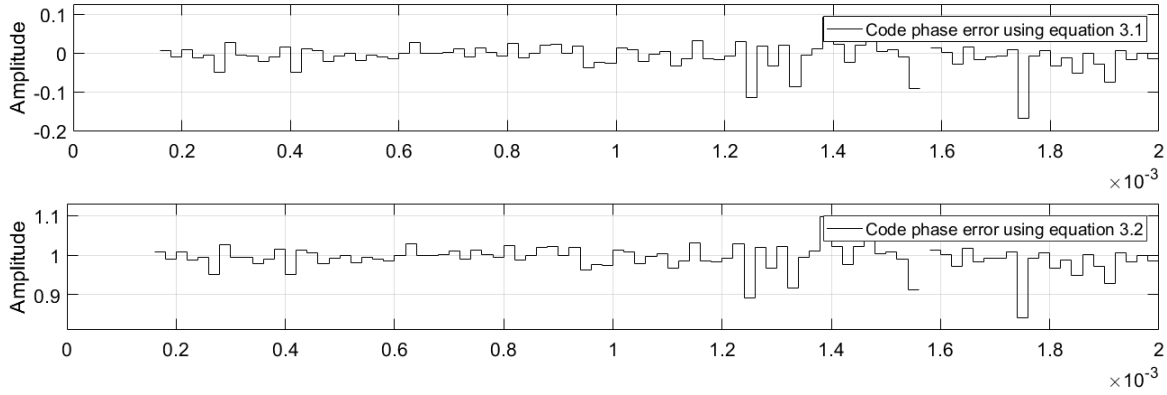


Figure 3.12: Simulation results of code phase errors obtained by using algorithms explained in equation 3.1 and equation 3.2

3.2.3 Programming a code phase alignment block that makes decision on code phase shift

Previously, the code phase error was calculated using the code loop discriminator. This code phase error is used as a feedback to the code phase alignment correction block. This block is

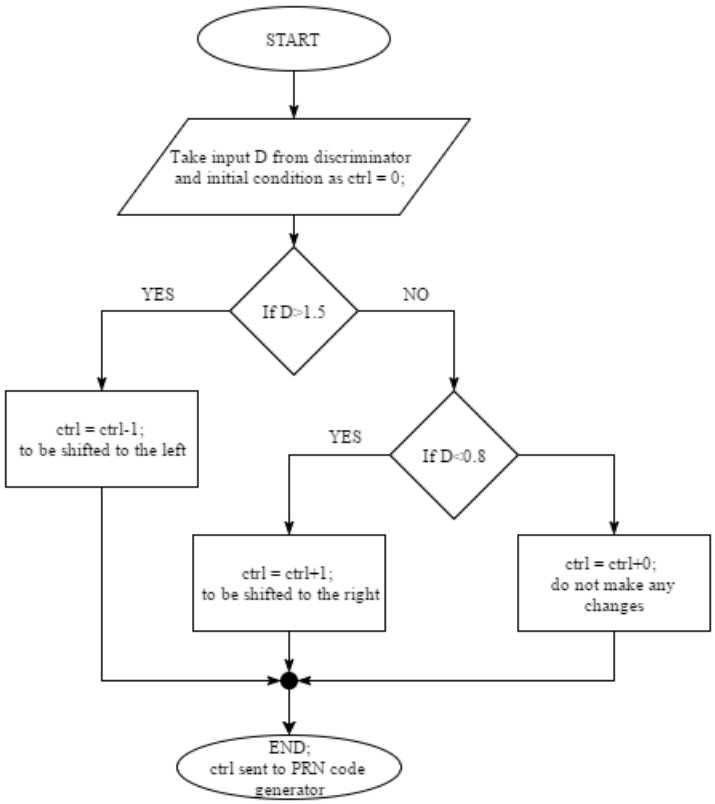


Figure 3.13: Flowchart of code phase alignment correction (Johansson et al. 1998)

designed using user-defined MATLAB function. This function is based on a flowchart shown in figure 3.13 in the previous page. The output of this function is a control line that allows the PRN code generator to either make a phase shift or not make a phase shift. If the value of code phase error is greater than 1.5, a control line is decreased by 1 and allows the local code generator to make a phase shift to the left. If the value of code phase error is lesser than 0.8, a control line is increased by 1 and allows the local code generator to make a phase shift to the right. If none of these conditions meet, then the local code generator does not make any changes in the phase of PRN code. The MATLAB code for this user-defined function is attached in appendix B, section B.1.

3.2.4 Simulation of code tracking system for GNSS receiver in simulink toolbox

To verify a code tracking system, we compare an incoming PRN code with a locally generated PRN and see if these are perfectly aligned or not. Furthermore, we look at the code phase error (output from code loop discriminator) at a specific time elapsed and then compare incoming PRN code with locally generated PRN code.

Since, the PRN code has a very high frequency of 1.023 MHz. Comparing it with our naked eyes is a difficult task. So, a better solution is to take a difference between an incoming PRN code and a local PRN signal. When the result of subtraction becomes zero, we assume that the incoming PRN code and local PRN code are perfectly aligned, otherwise they are not.

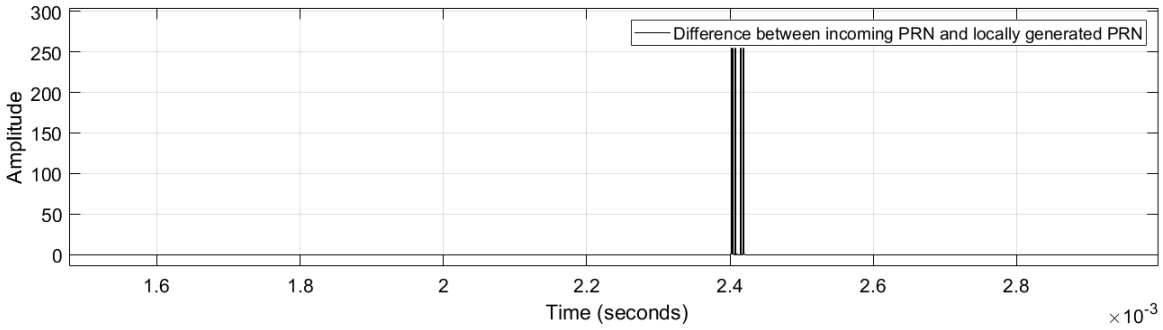


Figure 3.14: Simulation result of difference between incoming PRN code and locally generated PRN code (The codes are almost perfectly aligned except at $t = 2.4$ ms)

The simulation result of subtraction between an incoming PRN sequence and a local PRN sequence is shown in figure 3.14. This figure suggests that the incoming PRN code and the local PRN code are almost aligned perfectly except at time elapsed, $t = 2.4$ ms. The simulation result of incoming PRN and local PRN is shown in figure 3.15. The big box in this figure at $t = 2.4$ ms focuses where the phases are misaligned. This happens when the DLL cannot lock the phase of incoming PRN signal. The DLL technique that we use in this project is one of the best but also one of the slowest. Only after $t = 2.423$ ms, the DLL locks the phase of the incoming PRN signal. The complete simulation model for code tracking is presented in appendix A, figure A.10.

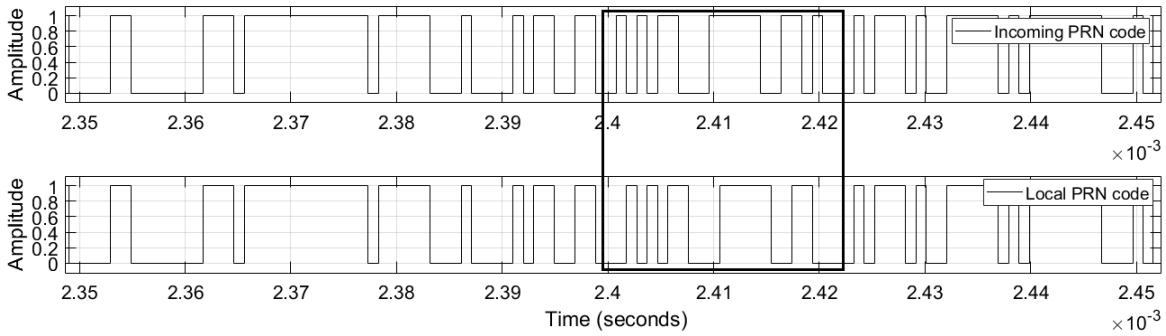


Figure 3.15: Simulation comparison between incoming PRN code and locally generated PRN code

3.3 Carrier tracking

This is the final task of tracking system where the incoming carrier signal is tracked such that phase and frequency of the local carrier wave is same as that of the incoming carrier signal. For tracking the phase of incoming carrier wave signal, a PLL is applied (explained in chapter 2, section 2.4).

In carrier wave tracking, we follow the following steps:

- First, the incoming signal is multiplied with the local carrier wave and then with the replica of incoming PRN code. This is performed to wipe off carrier signal and PRN code of the incoming signal. The result is a correlation value.
- The output of the correlators is sent to the carrier loop discriminator to determine the carrier phase error which is filtered out by the carrier loop filter.
- Finally, the the output from carrier filter is used as a feedback to the NCO that generates a perfectly aligned carrier wave signal as compared with the incoming carrier wave signal.

3.3.1 Design of a carrier loop discriminator

The simulink model for the Costas loop (based on chapter 2, figure 2.14) can be seen in appendix A, figure A.10. In the Costas loop, an integrator and dump is used instead of a LPF. The integrator integrates over one PRN code period and performs just like a LPF with a stop band frequency of 1 KHz. The detailed model for integrator and dump used in simulink can be seen in appendix A, figure A.9.

To calculate the correlation, we only require the results of multiplication from prompt PRN code which are I_P and Q_P. Once the correlation values are obtained, they are sent to the carrier loop discriminator to determine the carrier phase error.

As from chapter 2, section 2.4 - the carrier phase error is given by

$$\varphi = \arctan\left(\frac{Q_P}{I_P}\right) \tag{3.3}$$

Based on this equation, a carrier phase discriminator is designed which outputs the carrier phase error. The output from the carrier phase discriminator is shown in figure 3.16. With every iteration, a carrier phase error is obtained. Then it is filtered out by a carrier loop filter.

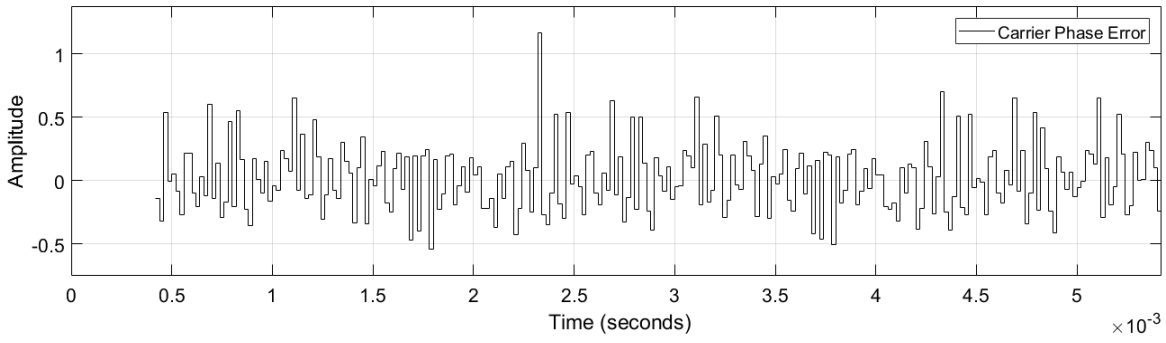


Figure 3.16: Simulation result of carrier phase error from carrier loop discriminator

3.3.2 Design of a carrier loop filter

The proper design of a carrier loop filter is very important. If a carrier loop filter is not properly designed, the PLL will not lock the phase of the incoming carrier wave. This results in misalignment between incoming carrier wave and local carrier wave. Thus, a carrier loop filter needs to be designed in a proper way.

As from chapter 2, section 2.4 - we have the following list of equations (from equation 3.4 to equation 3.8). The digital form of transfer function of a carrier loop filter and a NCO is given by

$$F(z) = \frac{(C_1 + C_2) - C_1 z^{-1}}{1 - z^{-1}} \tag{3.4}$$

$$N(z) = \frac{K_o z^{-1}}{1 - z^{-1}} \tag{3.5}$$

where K_o is a gain of NCO. The coefficients - C_1 and C_2 are given by

$$C_1 = \frac{1}{K_o K_d} \frac{8\zeta\omega_n T}{4 + 4\zeta\omega_n T + (\omega_n T)^2} \tag{3.6}$$

$$C_2 = \frac{1}{K_o K_d} \frac{4(\omega_n T)^2}{4 + 4\zeta\omega_n + (\omega_n T)^2} \tag{3.7}$$

where K_d is a carrier discriminator gain and ω_n is a natural frequency in radians

Again, natural frequency ω_n is given by

$$\omega_n = \frac{8\zeta B_L}{4\zeta^2 + 1} \tag{3.8}$$

where ζ is the damping ratio and B_L is the noise bandwidth in the loop.

To design a carrier loop filter, we require coefficients - C_1 and C_2. A carrier loop filter is designed in such a way that, only the damping ratio, ζ and the noise bandwidth, B_L are taken as input parameters. The program automatically calculates the coefficients - C_1 and C_2. The table 3.1 shows the values for the input parameters used in the simulation. The values for B_L and ζ are based on "hit and trial" method. With the values mentioned in table 3.1, the PLL works out just fine. The MATLAB code to calculate coefficients - C_1 and C_2 is attached in appendix B, section B.2.

Table 3.1: Input parameters to the carrier loop filter

Parameters	**Description**
$\zeta = 0.7$	Damping ratio
$B_L = 50$ Hz	Bandwidth
$K_o K_d = 400\pi$	K_o is NCO gain K_d is discriminator gain
$T = \frac{1}{1000}$s	Sampling time

3.3.3 Simulation of local carrier wave

The output of carrier loop filter is used as a feedback to adjust the parameters for the NCO such that it finally, generates the exact replica of an incoming carrier signal. The NCO can generate both sine and cosine wave. The simulation outputs can be seen in figure 3.17. The carrier wave generated by local NCO (sine) is fed into the in-phase arm and the 90^o phase shifted carrier wave (cosine) is fed into the quadrature arm. These can be seen in appendix A, figure A.10.

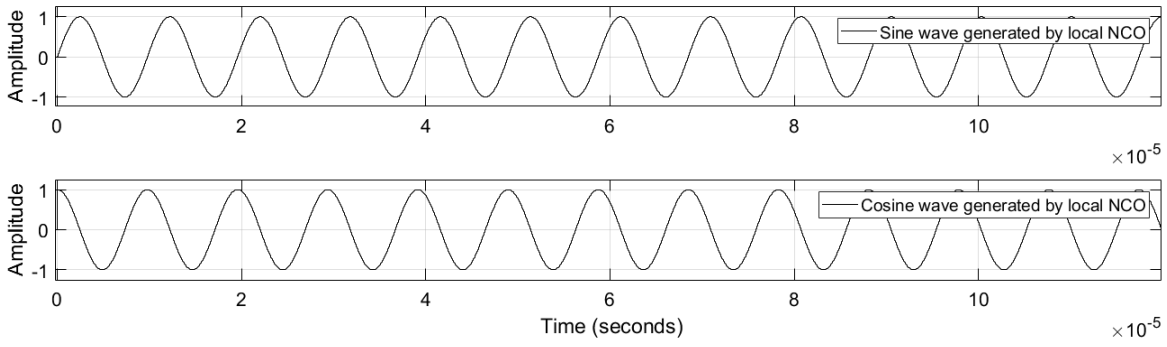


Figure 3.17: Simulation outputs from NCO showing both sine and cosine waves

3.3.4 Simulation of carrier tracking system for GNSS receiver in simulink toolbox

To verify the carrier tracking system, we compare the incoming carrier wave signal with locally generated carrier wave signal. The simulation result of the incoming carrier wave and the locally generated carrier wave is illustrated in figure 3.18. From this figure, we can clearly see that the phase of incoming carrier wave and the phase of local carrier wave generated by a NCO is same. It means that the carrier wave filter is designed properly and the PLL is able to lock the phase of incoming carrier wave. Thus, we conclude that the local carrier wave is perfectly aligned with the incoming carrier wave . The simulink model for the carrier tracking loop is presented in appendix A, figure A.10.

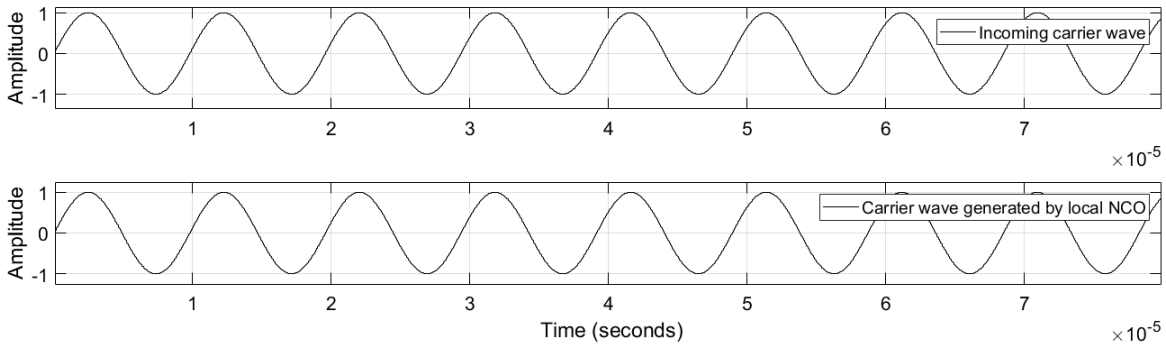


Figure 3.18: Simulation results of comparison between incoming carrier signal and local carrier signal

Chapter 4

Implementation of FPGA-based Tracking Algorithm

In this chapter, we look at the design of tracking system in a hardware level. The top level view of hardware design of a tracking system can be seen in figure 4.1 which is based on figure 2.14 from chapter 2.

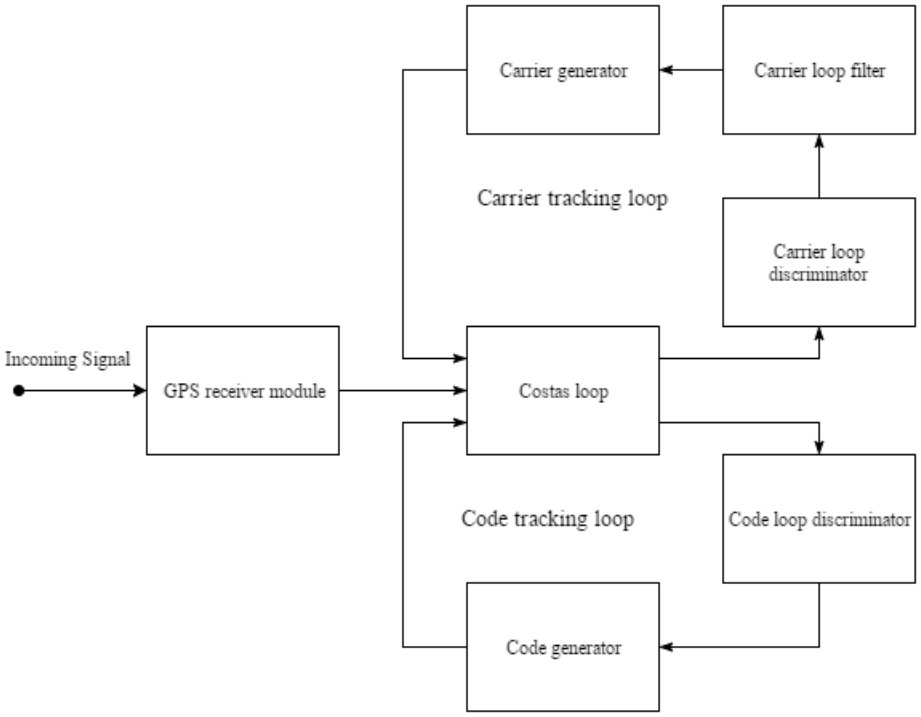


Figure 4.1: Top level view of hardware design of a tracking system

The project is limited with hardware resources. The only available piece of hardware is a FPGA kit, Spartan 3E-100 CP132. So, we look at the subsystems of tracking system that can be implemented on this kit. A tracking system consists of two major subsystems: carrier tracking loop and code tracking loop.

The carrier tracking loop is designed using PLL. It consists of Costas loop, carrier loop discriminator, carrier loop filter and local carrier generator. These can be seen in figure 4.1. The carrier loop discriminator is a user-defined function which cannot be synthesized on a provided FPGA kit. Similarly, Costas loop, local carrier generator and carrier loop filter cannot be synthesized.

The code tracking loop is designed using DLL. It consists of Costas loop, code loop discriminator and code generator. These can be seen in figure 4.1. The Costas loop is designed using integrator and dump which performs like a LPF with a band stop frequency of 1 KHz. This filter cannot be synthesized on a provided FPGA kit. The code loop discriminator is a user-defined function that decides when and in which direction to shift the phase of code. So, the only remaining part is a PRN code generator which can be programmed in VHDL and implemented on a provided FPGA kit.

In short, the only major part that can be programmed in VHDL and implemented on a FPGA kit, Spartan 3E-100 CP132 is a PRN code generator.

To implement a PRN code generator on a FPGA, the following steps are followed:

- Design a PRN code generator in Register-Transfer Level (RTL).
- Code a VHDL program for a code generator.
- Code a testbench program for a code generator.
- Compare simulation results in ISIM toolbox and simulink toolbox to verify the VHDL program.
- Program FPGA and check results in oscilloscope.

4.1 RTL schematic design of a PRN code generator

Based on theory from chapter 2, section 2.1.3, a Register-Transfer Level (RTL) schematic design of prompt PRN is designed which can be seen in figure 4.2. Both G1 and G2 registers use 10 flipflops and generate a sequence of $2^{10} - 1 = 1023$ bits. For G1 register, the outputs from cell 3 and cell 10 are fed into an exclusive-or. Then the output from exclusive-or is used as a feedback into the cell 1. For G2, the outputs from cell 2, cell 3, cell 6, cell 8, cell 9 and cell 10 are fed into an exclusive-or. Then the output from exclusive-or is used as a feedback into the cell 1. Finally, an exclusive-or is applied between the output from G1 register and a delayed version of output from G2 register. Since, we choose satellite id #1 (as from table 2.2, chapter 2), an exclusive-or is applied between cell 2 and cell 6. Based on the RTL schematic design, a VHDL program is coded for a code generator.

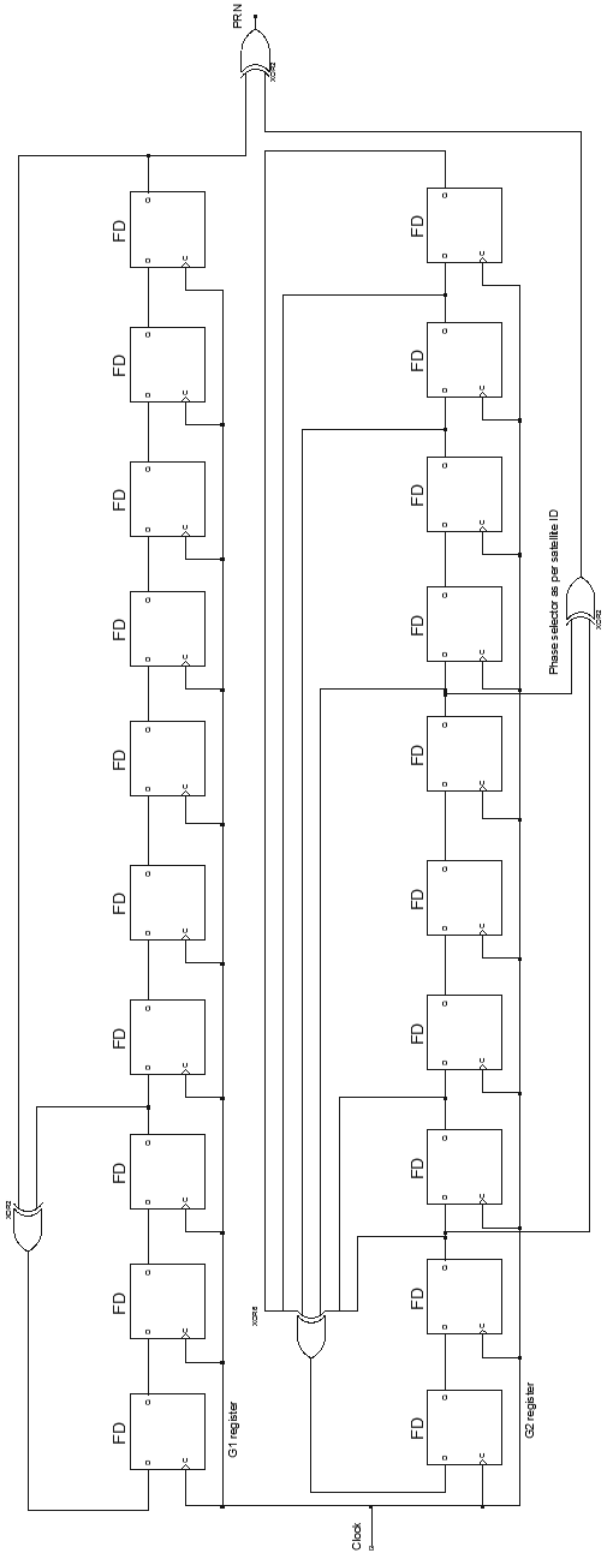


Figure 4.2: RTL schematic design of a PRN code generator

4.2 Coding and verification of VHDL program

The PRN code generator in a code tracking loop has three outputs - early PRN, prompt PRN and late PRN. Before simulation, a VHDL programs for early, prompt and late PRN are coded. These VHDL programs are attached in appendix B - section B.3, section B.4 and section B.5 respectively. Then a testbench program is coded followed by a simulation in ISIM. The testbench program is well documented in appendix B, section B.6. The same testbench program is used to simulate early, prompt and late PRN.

4.2.1 Verification of VHDL program for early PRN code

A simulation is carried out in ISIM toolbox to verify the VHDL program for early PRN. The VHDL program for the early PRN is attached in appendix B, section B.3. The results of simulation in ISIM toolbox and results of simulation in simulink toolbox for early PRN are shown in figure 4.3 and figure 4.4 respectively. The *final_pn* signal in figure 4.3 defines the early PRN. The results from figure 4.3 and figure 4.4 clearly show that the early PRN takes off after 0.999 ms $\approx$ 1 ms which verify that the VHDL program and the testbench program (attached in appendix B, section B.6) for the early PRN are working.

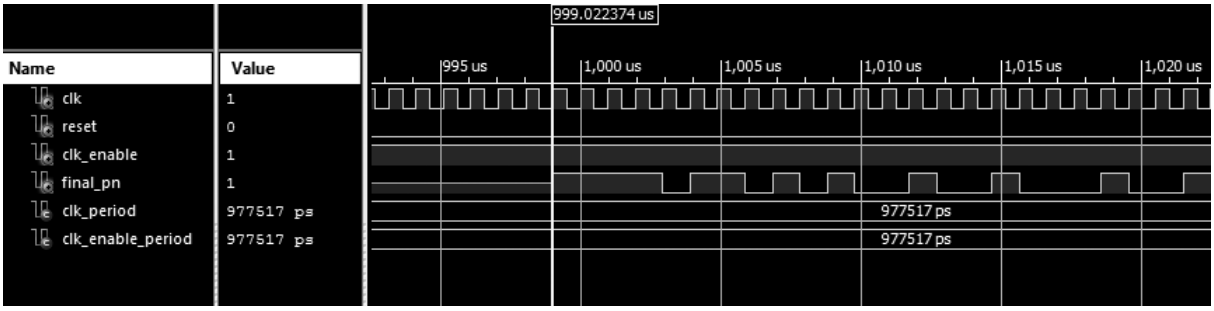


Figure 4.3: Results of simulation of early PRN in ISIM toolbox

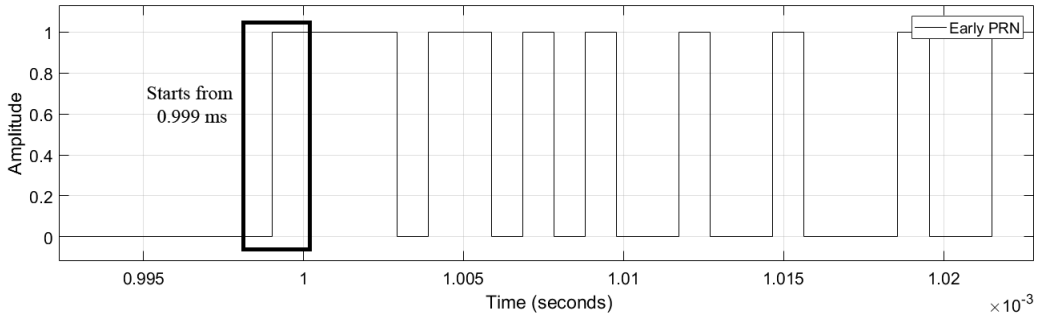


Figure 4.4: Results of simulation of early PRN in simulink toolbox

4.2.2 Verification of VHDL program for prompt PRN code

A simulation is carried out in ISIM toolbox to verify the VHDL program for prompt PRN. The VHDL program for the early PRN is attached in appendix B, section B.4. The results

of simulation of prompt PRN in ISIM toolbox can be seen in the figure 4.5. Now, the result of simulation in ISIM toolbox is compared with the results of simulation in simulink toolbox (in figure 4.6). The *final_pn* signal in figure 4.5 defines the prompt PRN. This *final_pn* signal and the signal shown in figure 4.6 are exactly the same. Both sequences start from 0 s. These results verify that the program coded in VHDL for prompt PRN code is correct. It also proves that the testbench program (appendix B, section B.6) for the prompt PRN is correct.

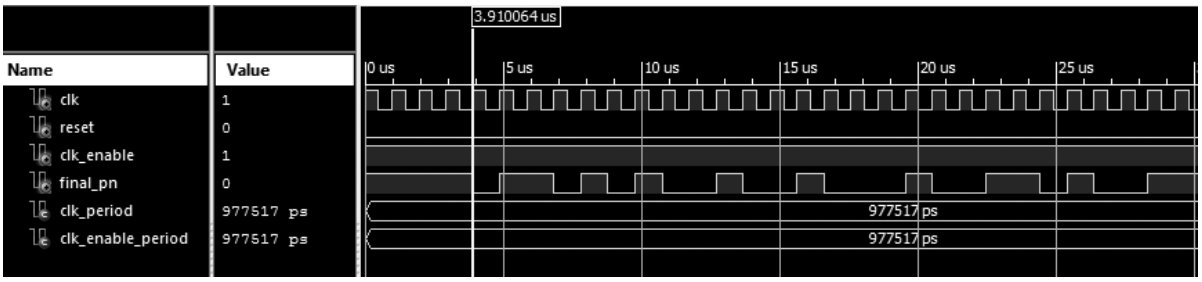


Figure 4.5: Results of simulation of prompt PRN in ISIM toolbox

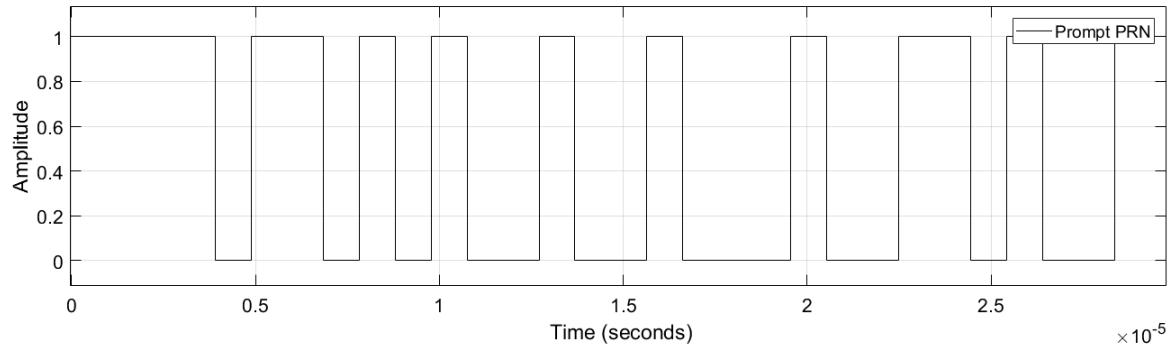


Figure 4.6: Results of simulation of prompt PRN in simulink toolbox

4.2.3 Verification of VHDL program for late PRN code

A simulation is carried out in ISIM toolbox to verify the VHDL program for late PRN. The

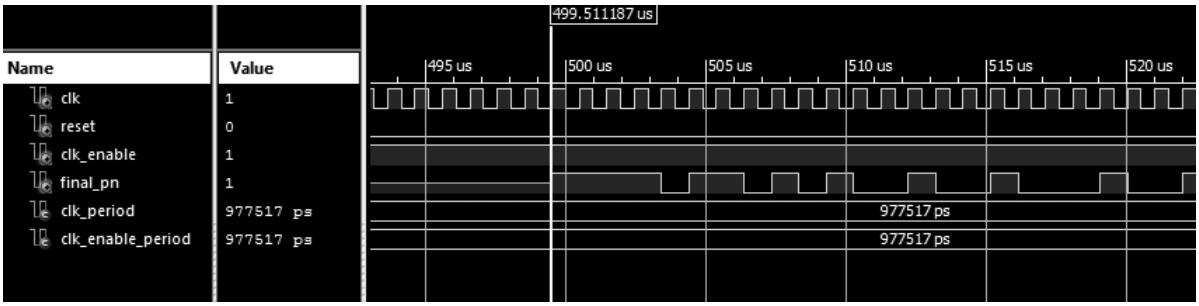


Figure 4.7: Results of simulation of late PRN in ISIM toolbox

VHDL program for the early PRN is attached in appendix B, section B.5. Now, a simulation is carried out in ISIM toolbox to test the late PRN. The results of simulation in ISIM toolbox and simulink toolbox for late PRN are illustrated in figure 4.7 and figure 4.8 respectively. The *final_pn* signal in figure 4.7 defines the late PRN sequence. It is clear that the late PRN begins after $\frac{1}{2}$ a nominal chipping rate of prompt PRN which is 0.499 ms $\approx$ 0.5 ms. Both these simulation results in ISIM toolbox and simulink toolbox are similar. Thus, we conclude that the program coded in VHDL and the testbench program for late PRN are correct.

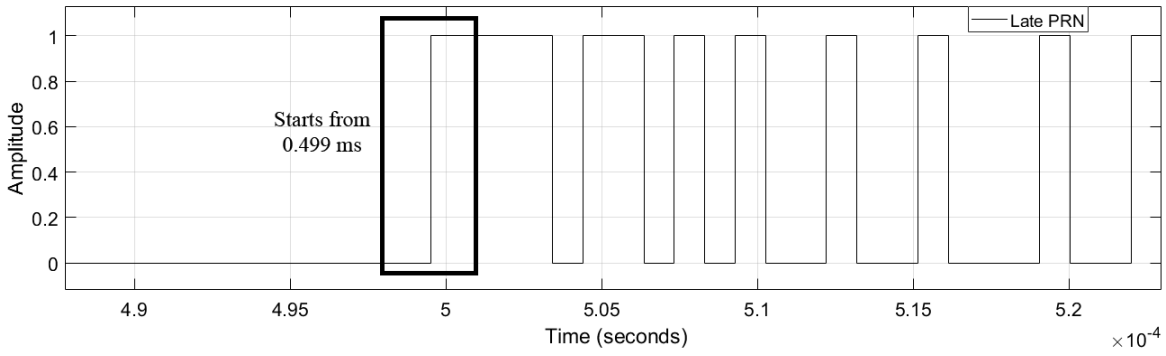


Figure 4.8: Results of simulation of late PRN in simulink toolbox

In summary, the VHDL program to generate early, prompt and late PRN codes were successfully verified by simulating in ISIM toolbox and comparing those results with simulation results in simulink toolbox.

4.3 FPGA implementation and verification

The FPGA kit, Spartan 3E-100 CP132 is used in this project to verify implementation of FPGA-based tracking algorithm. The pin details of this kit is attached in appendix C.

To verify VHDL programs that generate early, prompt and late PRN codes; VHDL programs without a use of clock divider function were coded. However, a clock divider must be used for implementation on FPGA because the clock frequency (= 25 MHz) in FPGA kit is different than that of the project requirement (= 1.023 MHz).

The Spartan 3E-100 CP132 FPGA kit is facilitated with oscillators of frequency 25 MHz, 50 MHz and 100 MHz. For implementation, we choose 25 MHz oscillator pinned at "C8". The PRN code is driven by a clock frequency of 1.023 MHz. To achieve a nearest clock frequency of 1.023 MHz, a clock divider function with N = 5 is selected such that the new clock frequency becomes 25 MHz/2^4= 1.5625 MHz. The VHDL code for prompt PRN with the use of clock divider function is attached in appendix B, section B.7.

Now, to verify the implementation of FPGA-based tracking algorithm, a bit file is generated which is attached in appendix B, section B.8. The FPGA kit is programmed with this bit file. The clock signal is locked at pin "B8", the reset signal is locked at pin "G12", the

clock_enable signal is locked at pin "L3" and the final_pn signal is locked at pin "M5". To achieve a waveform in oscilloscope, a probe is connected to pin "M5" and "GND". The results as seen in oscilloscope is shown in figure 4.9 and 4.10.

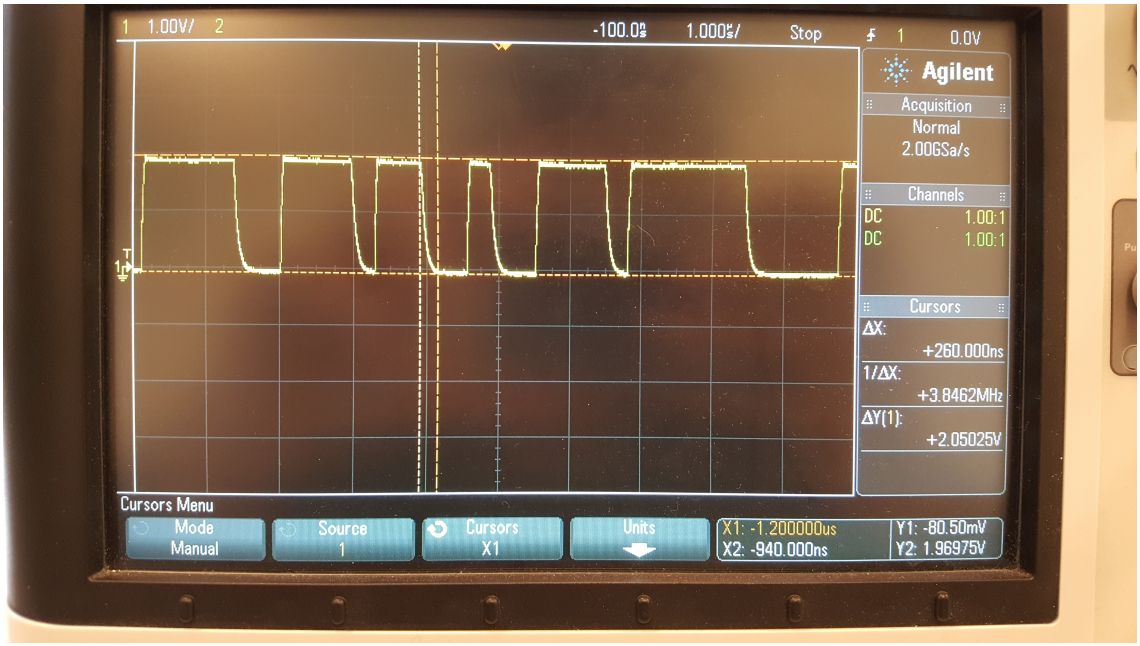


Figure 4.9: Implementation of code generator in FPGA focusing clock frequency and peak-to-peak voltage

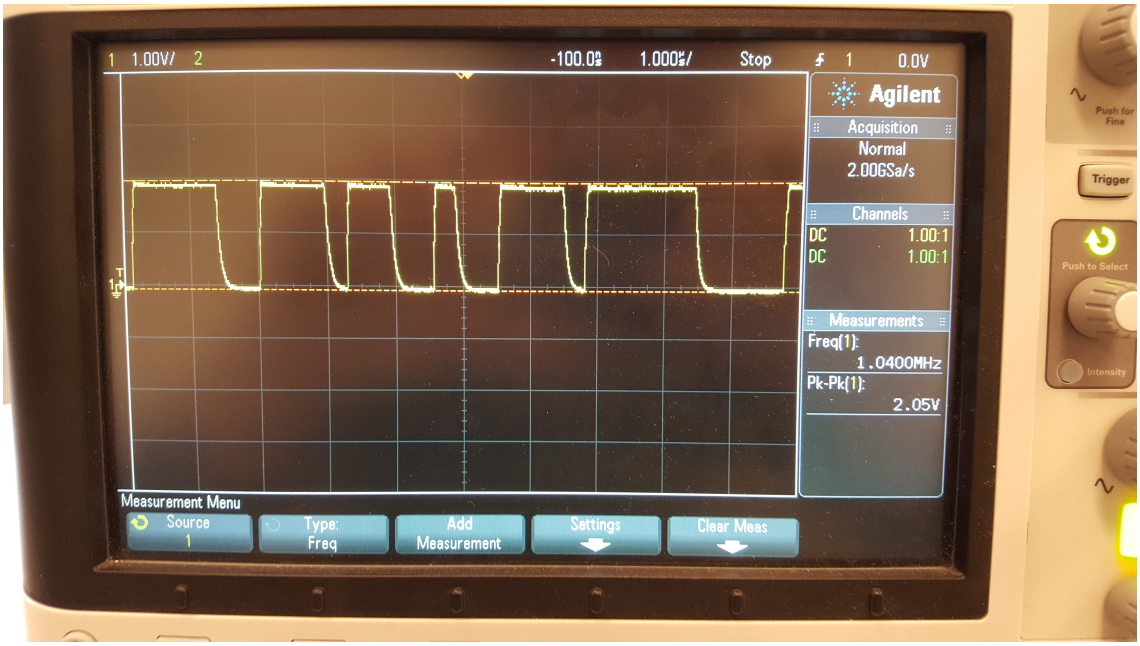


Figure 4.10: Implementation of code generator in FPGA focusing delay, ΔX

Figure 4.9 shows a PRN code being generated by a FPGA kit. This figure focuses mainly on the delays in the signal. In hardware, an exact PRN signal cannot be achieved as compared to the signal achieved in simulation. Here, the value of $\Delta X = 260$ ns i.e. the bit transition time from 1 to 0 is 260 ns. This figure also gives information on amplitude which is defined by $\Delta Y = 2.050$ V.

Figure 4.10 focuses on the frequency of signal. In theory, a clock divider function with N = 4 should be used to achieve a new clock frequency of 1.5625 MHz. But in practical case, the frequency of the clock fluctuates with time. Therefore, the signal frequency as shown in the figure is only 1.04 MHz at the time when we freeze the oscilloscope.

Furthermore, the results in oscilloscope resembles the simulation results (figure 4.5) in ISIM toolbox and simulation results (figure 4.6) in simulink toolbox. This resemblance verifies the implementation of tracking system for GNSS receiver on FPGA kit.

In summary, the PRN code generator was successfully simulated in ISIM toolbox. The results of comparison between simulation results in ISIM toolbox and simulink toolbox for early, prompt and late PRN verified the VHDL programs that generate early, prompt and late PRN. Then, the code generator was implemented on FPGA kit namely Spartan 3E-100 CP132. Finally, the implementation on FPGA was verified using an oscilloscope. Unfortunately, with lack of hardware resources, the only tracking algorithm that could be implemented on FPGA kit was the PRN code generator. Implementation of other tracking subsystems was left for future works (chapter 5).

Chapter 5

Conclusion and Future Works

5.1 Discussion and conclusion

The tracking system for GNSS receiver was designed and simulated successfully in the simulink toolbox. It consists of two major subsystems: code tracking loop and carrier tracking loop.

The code tracking loop was designed using DLL. The DLL was designed using Costas loop, code loop discriminator and local code generator. A non-coherent, early minus late power discriminator was applied that allowed a local code generator to produce a perfectly aligned code. This theory was applied in simulink toolbox and was successfully verified.

The carrier tracking loop was designed using PLL. The PLL was designed using Costas loop, carrier loop discriminator, carrier loop filter and local carrier wave generator. The discriminator outputs a carrier phase error filtered by a carrier loop filter. The feedback from filter to NCO allows the NCO to produce a perfectly aligned carrier wave signal. This theory was applied in simulink toolbox and was successfully verified.

The other half of the report dealt with VHDL programming and testing of code generator on a FPGA kit (Spartan 3E-100 CP132). The project lacked hardware resources. So, the only option for testing was simulating the VHDL program in ISIM toolbox. The simulation results in simulink and ISIM were compared. To verify the implementation of tracking algorithm on FPGA, an oscilloscope was used. The final results were very satisfying.

5.2 Recommendations for future works

This project can still be taken into another level. The project was limited with hardware resources. So, I would recommend specific hardwares that can be used for future work.

- For a GPS receiver module, MAESTRO A2235H can be used. The operating frequency of this receiver module is 1.575 GHz with an accuracy of 2.5 m. This module can be interfaced by I2C (Inter-IC), SPI (Serial Peripheral Interface) or UART (Universal Asynchronous Receiver/Transmitter). The technical details of A2235H is attached in appendix C, section C.5.

- The Costas loop can be designed and implemented on FPGA if a more powerful FPGA kit was available. A XAPP 132 FPGA kit can be used to implement a DLL. A standard implementation of DLL in XAPP 132 can be seen in appendix C, section **??**.
- For code loop discriminator and carrier loop discriminator, I would recommend a microcontroller STM32F030F4P6TR. It is a 32 bit microcontroller with 20 pins. This module is embedded with I2C , SPI and USART (Universal Synchronous/Asynchronous Receiver/Transmitter) interfaces. The technical details of STM32F030F4P6TR is attached in appendix C.
- A carrier loop filter is a LPF and can be designed using passive components.
- For a local carrier generator, RF signal generator HM8135 can be used. This device is very expensive and can be found at a price of NOK 34257.00 as from (farnell, 2017). The device can generate a signal of frequencies ranging from 1 Hz to 3 GHz. The technical details of HM8135 is attached in appendix C.

Based on the modules/hardwares mentioned above, I would like to propose a top level design of a FPGA-based tracking system for GNSS receivers with specific hardwares mentioned. The design proposal can be seen in figure 5.1 which can be a base for future works in FPGA-based tracking system for GNSS receivers.

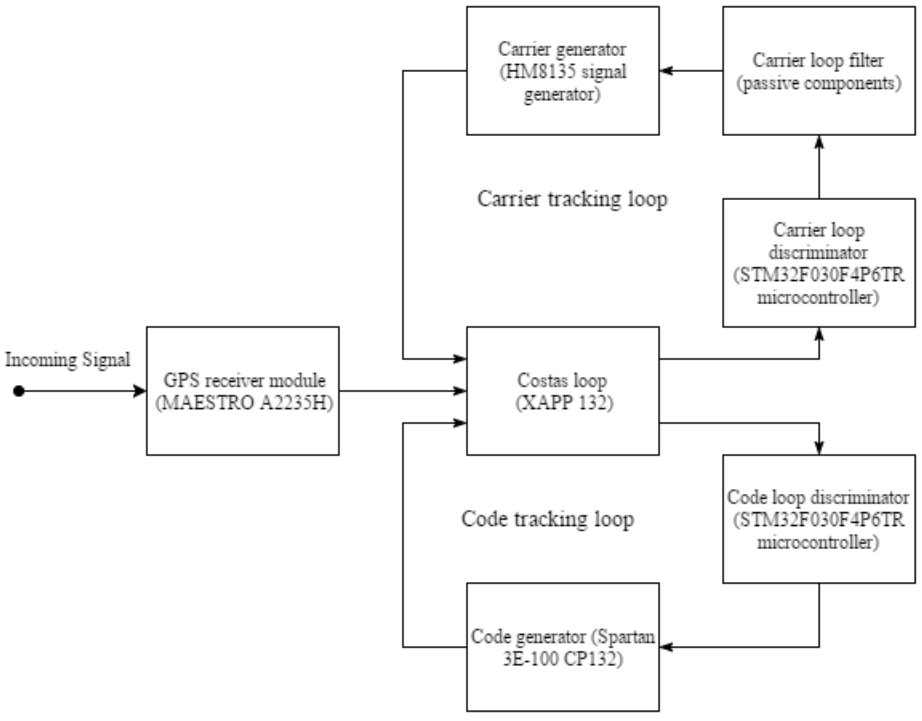


Figure 5.1: Top level design of a FPGA-based tracking system for GNSS receivers with specific hardwares mentioned

Bibliography

K. Borre, et al. (2007). *A software-defined GPS and Galileo receiver: a single-frequency approach*. Springer Science & Business Media.

G. Brodin & P. Daly (1997). 'GNSS code and carrier tracking in the presence of multipath'. *International journal of satellite communications* **15**(1):25–34.

B.-Y. Chung, et al. (1993). 'Performance analysis of an all-digital BPSK direct-sequence spread-spectrum IF receiver architecture'. *IEEE Journal on Selected Areas in Communications* **11**(7):1096–1107.

S. Gleason & D. Gebre-Egziabher (2009). 'Gnss navigation: Estimating position, velocity, and time'. *GNSS Applications and Methods (Scot Gleason and Demoz Gebre-Egziabher, eds.), Artech House, Norwood, MA* pp. 55–86.

R. Gold (1967). 'Optimal binary sequences for spread spectrum multiplexing (Corresp.)'. *IEEE Transactions on Information Theory* **13**(4):619–621.

R. Gold & R. C. Dixon (1998). 'Method for generating and encoding signals for spread spectrum communication'. US Patent 5,724,383.

S. Guruprasad (2015). *FPGA-Based Software GNSS Receiver Design for Satellite Applications*. Ph.D. thesis, YORK UNIVERSITY TORONTO.

J. K. Holmes (1982). 'Coherent spread spectrum systems'. *New York, Wiley-Interscience, 1982. 636 p.* **1**.

F. Johansson, et al. (1998). 'GPS satellite signal acquisition and tracking'. *Undergraduate projects* .

E. Kaplan & C. Hegarty (2005). *Understanding GPS: principles and applications*. Artech house.

G. Kappen & T. G. Noll (2006). 'Application specific instruction processor based implementation of a GNSS receiver on an FPGA'. In *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*, pp. 58–63. European Design and Automation Association.

J. A. Klobuchar (1987). 'Ionospheric time-delay algorithm for single-frequency GPS users'. *IEEE Transactions on aerospace and electronic systems* (3):325–331.

B. W. Parkinson (1996). *Progress in astronautics and aeronautics: Global positioning system: Theory and applications*, vol. 2. Aiaa.

R. L. Peterson & R. E. Ziemer (1985). 'Digital Communications and Spread Spectrum System' .

E. Re & M. Ruggieri (2007). *Satellite communications and navigation systems*. Springer Science & Business Media.

P. K. Tysowski (2009). 'Method of downloading ephemeris data based on user activity'. US Patent 7,633,438.

L. R. Weill (2010). 'A high performance code and carrier tracking architecture for ground-based mobile GNSS receivers'. In *23rd International Technical Meeting of the Satellite Division of the Institute of Navigation 2010 (ION GNSS 2010)*.

Appendix A

Simulation Models as Designed in Simulink

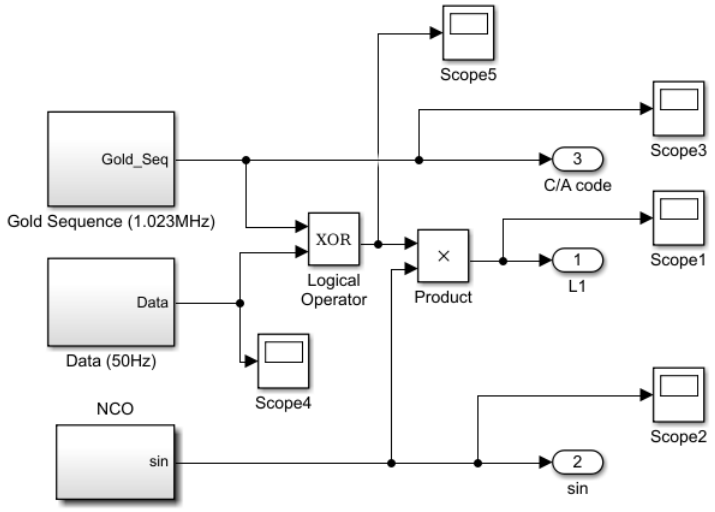


Figure A.1: A simplified GPS model

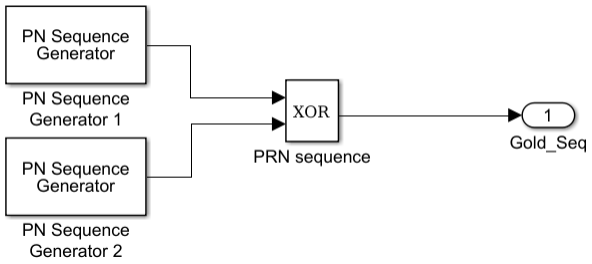


Figure A.2: A PRN sequence generator model

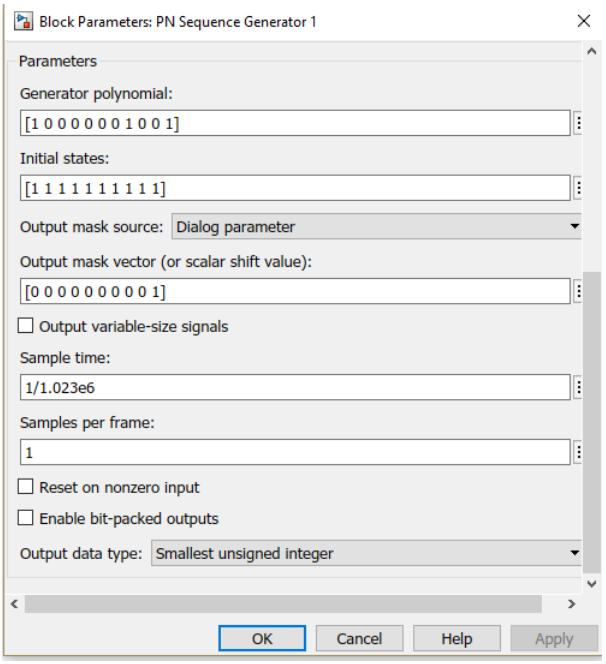


Figure A.3: Configuration parameter for G1 register

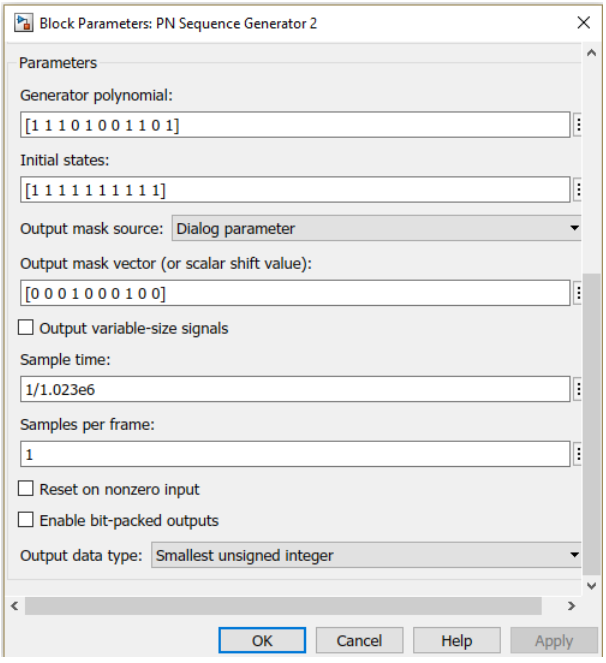


Figure A.4: Configuration parameter for G2 register

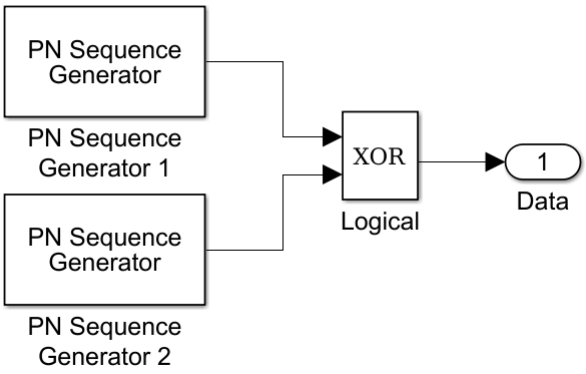


Figure A.5: Model to generate navigation data

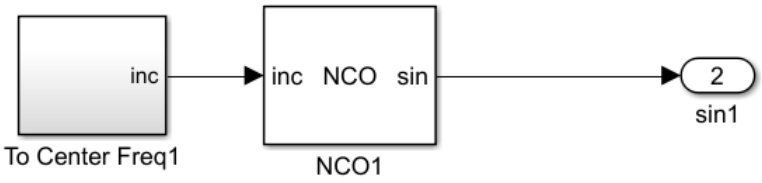


Figure A.6: Model to generate carrier wave

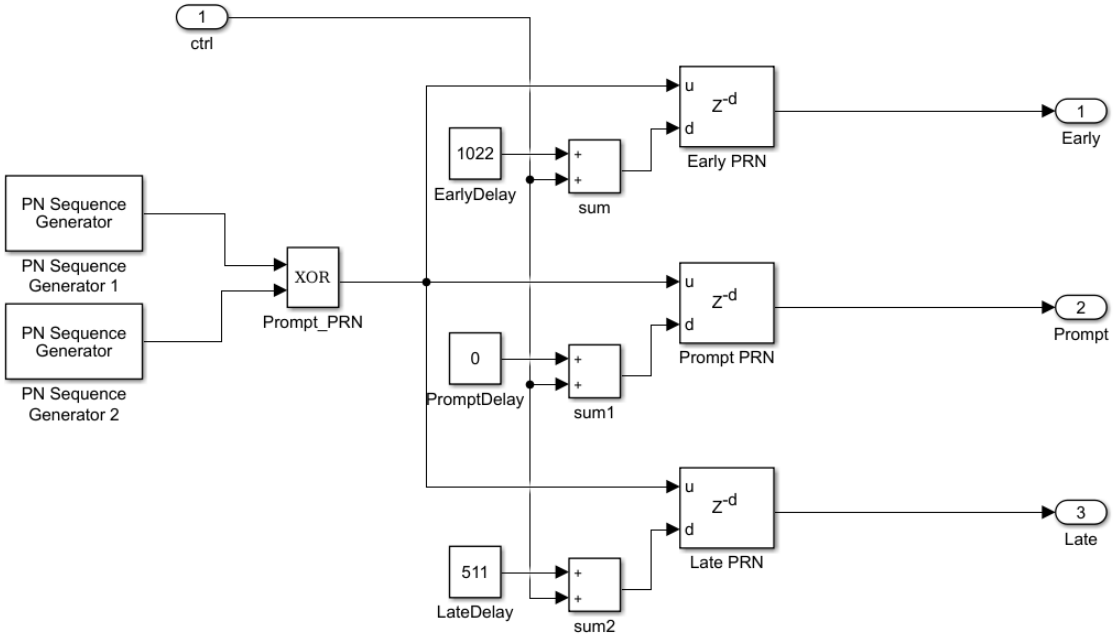


Figure A.7: Model to generate early, prompt and late PRN codes

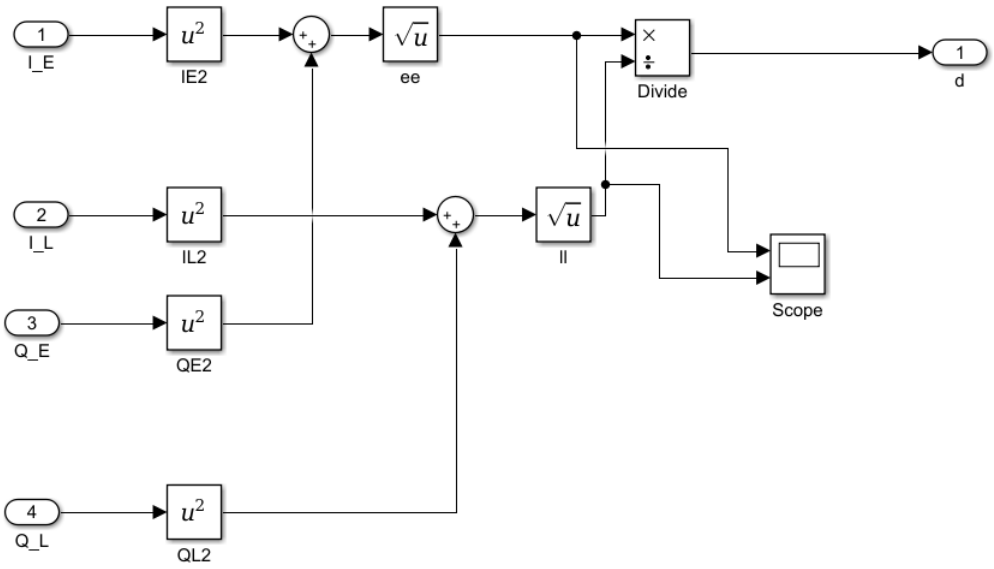


Figure A.8: Implementation of code phase discriminator for equation 3.2

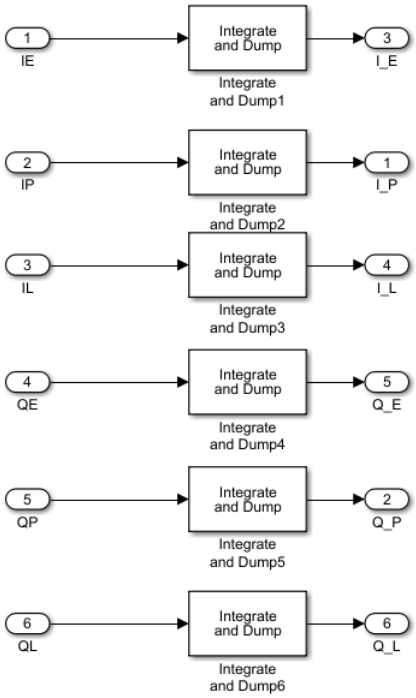


Figure A.9: Details of integrator and dump

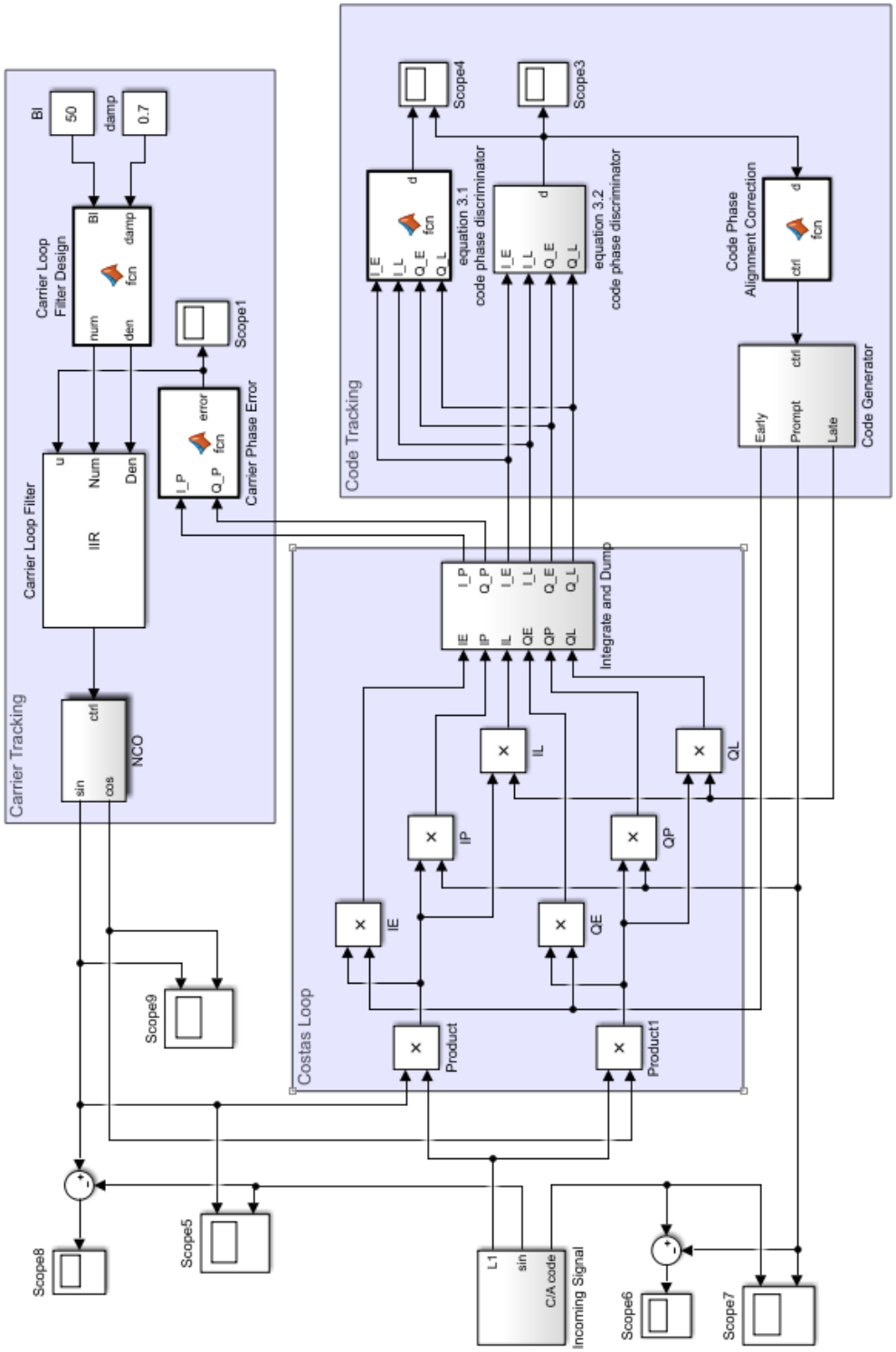


Figure A.10: A complete model for code and carrier tracking

Appendix B

MATLAB and VHDL codes

B.1 MATLAB code for correcting code phase alignment

```
function ctrl = fcn(d)

    ctrl = 0;
if d > 1.5
    ctrl = ctrl - 1; % to be shifted to the left
elseif d < 0.8
    ctrl = ctrl + 1; % to be shifted to the right
else
    ctrl = ctrl + 0; % do not make any change
end
```

B.2 MATLAB code for calculating coefficients C_1 and C_2

```
function [num, den]= fcn(Bl, damp)

wn = 8*Bl*damp/(4*damp*damp+1);
dT = 1e-3;
k = 400*pi;
c1 = 1/k*8*damp*wn*dT/(4+4*damp*wn*dT+wn*wn*dT*dT);
c2 = 1/k*4*wn*wn*dT*dT/(4+4*damp*wn*dT+wn*wn*dT*dT);

num = [c1+c2 -c1];
den = [1 -1];
```

B.3 VHDL code for early PRN

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE WORK.HDL_PRN_001_PKG.ALL;

ENTITY HDL_prn_001 IS
    PORT( clk : IN std_logic;
    reset : IN std_logic;
    clk_enable : IN std_logic;
    final_pn : OUT std_logic
    );
END HDL_prn_001;

ARCHITECTURE rtl OF HDL_prn_001 IS
- - Signals
    SIGNAL enb : std_logic;
    PN_Sequence_Generator_1_out1 : OUT unsigned(7 DOWNTO 0); - - uint8
    PN_Sequence_Generator_2_out1 : OUT unsigned(7 DOWNTO 0); - - uint8
    SIGNAL pn_reg : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out : std_logic;
    SIGNAL pn_xorout : std_logic;
    SIGNAL pn_newvalue : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted : unsigned(8 DOWNTO 0); - - ufix9_E1
    SIGNAL pn_reg_1 : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out_1 : std_logic;
    SIGNAL pn_xorout_1 : std_logic;
    SIGNAL pn_newvalue_1 : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted_1 : unsigned(8 DOWNTO 0); - - ufix9_E1
    SIGNAL A : std_logic;

BEGIN
    enb <= clk_enable;
    pn_newvalue(0) <= pn_reg;
    pn_xorout <= pn_newvalue(0)(0) XOR pn_newvalue(0)(3);
    pn_value_shifted <= pn_newvalue(0)(9 DOWNTO 1);
    pn_newvalue(1) <= pn_xorout & pn_value_shifted;
    pn_out <= pn_newvalue(0)(0);

PN_generation_temp_process1 : PROCESS (clk, reset)
BEGIN
    IF reset = '1' THEN
        pn_reg <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
        IF enb = '1' THEN
```

```
            pn_reg <= pn_newvalue(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process1;

PN_Sequence_Generator_1_out1 <= resize("0" & pn_out, 8);

    pn_newvalue_1(0) <= pn_reg_1;
    pn_xorout_1 <= pn_newvalue_1(0)(0) XOR pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(3)
XOR pn_newvalue_1(0)(8);
    pn_value_shifted_1 <= pn_newvalue_1(0)(9 DOWNTO 1);
    pn_newvalue_1(1) <= pn_xorout_1 & pn_value_shifted_1;
    pn_out_1 <= pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(6);

PN_generation_temp_process2 : PROCESS (clk, reset)
BEGIN
    IF reset = '1' THEN
        pn_reg_1 <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
        IF enb = '1' THEN
            pn_reg_1 <= pn_newvalue_1(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process2;

PN_Sequence_Generator_2_out1 <= resize("0" & pn_out_1, 8);
A <= pn_out XOR pn_out_1;
final_pn <= transport A after 2*511*977.517 ns;
END rtl;
```

B.4 VHDL code for prompt PRN

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE WORK.HDL_PRN_001_PKG.ALL;

ENTITY HDL_prn_001 IS
    PORT( clk : IN std_logic;
    reset : IN std_logic;
    clk_enable : IN std_logic;
    final_pn : OUT std_logic
    );
END HDL_prn_001;
```

```
ARCHITECTURE rtl OF HDL_prn_001 IS
- - Signals
    SIGNAL enb : std_logic;
    PN_Sequence_Generator_1_out1 : OUT unsigned(7 DOWNTO 0); - - uint8
    PN_Sequence_Generator_2_out1 : OUT unsigned(7 DOWNTO 0); - - uint8
    SIGNAL pn_reg : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out : std_logic;
    SIGNAL pn_xorout : std_logic;
    SIGNAL pn_newvalue : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted : unsigned(8 DOWNTO 0); - - ufix9_E1
    SIGNAL pn_reg_1 : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out_1 : std_logic;
    SIGNAL pn_xorout_1 : std_logic;
    SIGNAL pn_newvalue_1 : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted_1 : unsigned(8 DOWNTO 0); - - ufix9_E1
    SIGNAL A : std_logic;

BEGIN
    enb <= clk_enable;
    pn_newvalue(0) <= pn_reg;
    pn_xorout <= pn_newvalue(0)(0) XOR pn_newvalue(0)(3);
    pn_value_shifted <= pn_newvalue(0)(9 DOWNTO 1);
    pn_newvalue(1) <= pn_xorout & pn_value_shifted;
    pn_out <= pn_newvalue(0)(0);

PN_generation_temp_process1 : PROCESS (clk, reset)
BEGIN
    IF reset = '1' THEN
        pn_reg <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
        IF enb = '1' THEN
            pn_reg <= pn_newvalue(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process1;

PN_Sequence_Generator_1_out1 <= resize("0" & pn_out, 8);

    pn_newvalue_1(0) <= pn_reg_1;
    pn_xorout_1 <= pn_newvalue_1(0)(0) XOR pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(3)
XOR pn_newvalue_1(0)(8);
    pn_value_shifted_1 <= pn_newvalue_1(0)(9 DOWNTO 1);
    pn_newvalue_1(1) <= pn_xorout_1 & pn_value_shifted_1;
    pn_out_1 <= pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(6);

PN_generation_temp_process2 : PROCESS (clk, reset)
```

```
BEGIN
    IF reset = '1' THEN
        pn_reg_1 <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
        IF enb = '1' THEN
            pn_reg_1 <= pn_newvalue_1(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process2;

PN_Sequence_Generator_2_out1 <= resize("0" & pn_out_1, 8);
A <= pn_out XOR pn_out_1;
final_pn <= A;
END rtl;
```

B.5 VHDL code for late PRN

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE WORK.HDL_PRN_001_PKG.ALL;

ENTITY HDL_prn_001 IS
    PORT( clk : IN std_logic;
    reset : IN std_logic;
    clk_enable : IN std_logic;
    final_pn : OUT std_logic
    );
END HDL_prn_001;

ARCHITECTURE rtl OF HDL_prn_001 IS
- - Signals
    SIGNAL enb : std_logic;
    PN_Sequence_Generator_1_out1 : OUT unsigned(7 DOWNTO 0); - - uint8
    PN_Sequence_Generator_2_out1 : OUT unsigned(7 DOWNTO 0); - - uint8
    SIGNAL pn_reg : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out : std_logic;
    SIGNAL pn_xorout : std_logic;
    SIGNAL pn_newvalue : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted : unsigned(8 DOWNTO 0); - - ufix9_E1
    SIGNAL pn_reg_1 : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out_1 : std_logic;
    SIGNAL pn_xorout_1 : std_logic;
    SIGNAL pn_newvalue_1 : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted_1 : unsigned(8 DOWNTO 0); - - ufix9_E1
```

```
    SIGNAL A : std_logic;

BEGIN
    enb <= clk_enable;
    pn_newvalue(0) <= pn_reg;
    pn_xorout <= pn_newvalue(0)(0) XOR pn_newvalue(0)(3);
    pn_value_shifted <= pn_newvalue(0)(9 DOWNTO 1);
    pn_newvalue(1) <= pn_xorout & pn_value_shifted;
    pn_out <= pn_newvalue(0)(0);

PN_generation_temp_process1 : PROCESS (clk, reset)
BEGIN
    IF reset = '1' THEN
        pn_reg <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
        IF enb = '1' THEN
            pn_reg <= pn_newvalue(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process1;

PN_Sequence_Generator_1_out1 <= resize("0" & pn_out, 8);

    pn_newvalue_1(0) <= pn_reg_1;
    pn_xorout_1 <= pn_newvalue_1(0)(0) XOR pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(3)
XOR pn_newvalue_1(0)(8);
    pn_value_shifted_1 <= pn_newvalue_1(0)(9 DOWNTO 1);
    pn_newvalue_1(1) <= pn_xorout_1 & pn_value_shifted_1;
    pn_out_1 <= pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(6);

PN_generation_temp_process2 : PROCESS (clk, reset)
BEGIN
    IF reset = '1' THEN
        pn_reg_1 <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
        IF enb = '1' THEN
            pn_reg_1 <= pn_newvalue_1(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process2;

PN_Sequence_Generator_2_out1 <= resize("0" & pn_out_1, 8);
A <= pn_out XOR pn_out_1;
final_pn <= transport A after 511*977.517 ns;
END rtl;
```

B.6 Testbench for PRN

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY HDL_prn_001_tb IS
END HDL_prn_001_tb;

ARCHITECTURE behavior OF HDL_prn_001_tb IS
    - - Component Declaration for the Unit Under Test (UUT)
    COMPONENT HDL_prn_001
    PORT(
        clk : IN std_logic;
        reset : IN std_logic;
        clk_enable : IN std_logic;
        final_pn : OUT std_logic
        );
    END COMPONENT;

    - - Inputs
    signal clk : std_logic := '0';
    signal reset : std_logic := '1';
    signal clk_enable : std_logic := '0';
    signal final_pn : std_logic := '0';

    - - Clock period definitions
    constant clk_period : time := 977.517 ns;
    constant clk_enable_period : time := 977.517 ns;

BEGIN
    - - Instantiate the Unit Under Test (UUT)
    uut: HDL_prn_001 PORT MAP (
        clk => clk,
        reset => reset,
        clk_enable => clk_enable,
        final_pn => final_pn
        );

    - - Clock process definitions
    clk_process :process
    begin
        clk <= '1';
        wait for clk_period/2;
        clk <= '0';
        wait for clk_period/2;
```

```
    end process;

    clk_enable_process :process
    begin
    clk_enable <= '1';
    wait;
    end process;

    - - Stimulus process
    stim_proc: process
    begin
    reset <= not(reset);
    wait;
    end process;
    END;
```

B.7 VHDL code for prompt PRN with use of clock divider

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.HDL_PRN_001_PKG.ALL;

ENTITY HDL_prn_001 IS
    PORT( clk : IN std_logic;
          reset : IN std_logic;
          clk_enable : IN std_logic;
          final_pn : OUT std_logic
          );
END HDL_prn_001;

ARCHITECTURE rtl OF HDL_prn_001 IS
    - - Signals
    SIGNAL enb : std_logic;
    SIGNAL PN_Sequence_Generator_1_out1 : unsigned(7 DOWNTO 0); - - uint8
    SIGNAL PN_Sequence_Generator_2_out1 : unsigned(7 DOWNTO 0); - - uint8
    SIGNAL pn_reg : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out : std_logic;
    SIGNAL pn_xorout : std_logic;
    SIGNAL pn_newvalue : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted : unsigned(8 DOWNTO 0); - - ufix9_E1
    SIGNAL pn_reg_1 : unsigned(9 DOWNTO 0); - - ufix10
    SIGNAL pn_out_1 : std_logic;
    SIGNAL pn_xorout_1 : std_logic;
```

```
    SIGNAL pn_newvalue_1 : vector_of_unsigned10(0 TO 1); - - ufix10 [2]
    SIGNAL pn_value_shifted_1 : unsigned(8 DOWNTO 0); - - ufix9_E1
    SIGNAL A : std_logic;
    SIGNAL slow_clk : std_logic;
    SIGNAL clk_divider : std_logic_vector(3 DOWNTO 0) := (OTHERS => '0');

BEGIN
clk_division : process (clk, clk_divider)
begin
    if (clk = '1' and clk'event) then
        clk_divider <= clk_divider +1;
    end if;
    slow_clk <= clk_divider(3);
end process clk_division;

enb <= clk_enable;
pn_newvalue(0) <= pn_reg;
pn_xorout <= pn_newvalue(0)(0) XOR pn_newvalue(0)(3);
pn_value_shifted <= pn_newvalue(0)(9 DOWNTO 1);
pn_newvalue(1) <= pn_xorout & pn_value_shifted;
pn_out <= pn_newvalue(0)(0);

PN_generation_temp_process1 : PROCESS (slow_clk, reset)
BEGIN
    IF reset = '1' THEN
        pn_reg <= to_unsigned(1023, 10);
    ELSIF slow_clk'event AND slow_clk = '1' THEN
        IF enb = '1' THEN
            pn_reg <= pn_newvalue(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process1;
PN_Sequence_Generator_1_out1 <= resize("0" & pn_out, 8);

pn_newvalue_1(0) <= pn_reg_1;
pn_xorout_1 <= pn_newvalue_1(0)(0) XOR pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(3)
XOR pn_newvalue_1(0)(6) XOR pn_newvalue_1(0)(8) XOR pn_newvalue_1(0)(9);
pn_value_shifted_1 <= pn_newvalue_1(0)(9 DOWNTO 1);
pn_newvalue_1(1) <= pn_xorout_1 & pn_value_shifted_1;
pn_out_1 <= pn_newvalue_1(0)(2) XOR pn_newvalue_1(0)(6);

PN_generation_temp_process2 : PROCESS (slow_clk, reset)
BEGIN
    IF reset = '1' THEN
        pn_reg_1 <= to_unsigned(1023, 10);
    ELSIF slow_clk'event AND slow_clk = '1' THEN
```

```
        IF enb = '1' THEN
            pn_reg_1 <= pn_newvalue_1(1);
        END IF;
    END IF;
END PROCESS PN_generation_temp_process2;

PN_Sequence_Generator_2_out1 <= resize("0" & pn_out_1, 8);
A <= pn_out XOR pn_out_1;
final_pn <= A;

END rtl;
```

B.8 Implementation constraints file for PRN

```
NET "clk" LOC = "B8";
NET "reset" LOC = "G12";
NET "clk_enable" LOC = "L3";
NET "final_pn" LOC = "M5";
```

Appendix C

Datasheet and Technical Details of Hardwares

C.1 Pin details of FPGA kit, Spartan 3E-100 CP132

FPGA pin definition table color key		
Grey		Not available to user
Green		User I/O devices
Yellow		Data ports
Tan		Pmod port signals
Blue		USB signals

Basys 2 Spartan-3E pin definitions											
Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
C12	JD1	P11	SW0	N14	CC	B2	JA1	P8	MODE0	M7	GND
A13	JD2	M2	USB-DB1	N13	DP	C2	USB-WRITE	N7	MODE1	P5	GND
A12	NC	N2	USB-DB0	M13	AN2	C3	PS2D	N6	MODE2	P10	GND
B12	NC	M9	NC	M12	CG	D1	NC	N12	CCLK	P14	GND
B11	NC	N9	NC	L14	CA	D2	USB-WAIT	P13	DONE	A6	VDDO-3
C11	BTN1	M10	NC	L13	CF	L2	USB-DB4	A1	PROG	B10	VDDO-3
C6	JB1	N10	NC	F13	RED2	L1	USB-DB3	N8	DIN	E13	VDDO-3
B6	JB2	M11	LD1	F14	GRN0	M1	USB-DB2	N1	INIT	M14	VDDO-3
C5	JB3	N11	CD	D12	JD4	L3	SW1	P1	NC	P3	VDDO-3
B5	JA4	P12	CE	D13	RED1	E2	SW6	B3	GND	M8	VDDO-3
C4	NC	N3	SW7	C13	JD3	F3	SW5	A4	GND	E1	VDDO-3
B4	SW3	M6	UCLK	C14	RED0	F2	USB-ASTB	A8	GND	J2	VDDO-3
A3	JA2	P6	LD3	G12	BTN0	F1	USB-DSTB	C1	GND	A5	VDDO-2
A10	JC3	P7	LD2	K14	AN3	G1	LD7	C7	GND	E12	VDDO-2
C9	JC4	M4	BTN2	J12	AN1	G3	SW4	C10	GND	K1	VDDO-2
B9	JC2	N4	LD5	J13	BLU2	H1	USB-DB6	E3	GND	P9	VDDO-2
A9	JC1	M5	LD0	J14	HSYNC	H2	USB-DB5	E14	GND	A11	VDDO-1
B8	MCLK	N5	LD4	H13	BLU1	H3	USB-DB7	G2	GND	D3	VDDO-1
C8	RCCLK	G14	GRN2	H12	CB	B14	TMS	H14	GND	D14	VDDO-1
A7	BTN3	G13	GRN1	J3	JA3	B13	TCK-FPGA	J1	GND	K2	VDDO-1
B7	JB4	F12	AN0	K3	SW2	A2	TDO-USB	K12	GND	L12	VDDO-1
P4	LD6	K13	VSYNC	B1	PS2C	A14	TDO-S3	M3	GND	P2	VDDO-1

C.2 Technical details of GSM receiver module, A2235-H

PERFORMANCE

Channels	48 parallel tracking
Correlators	400,000 plus
Frequency	L1 - 1,575 MHz
Sensitivity	
Tracking Navigation Acquisition (cold start)	- 163 dBm - 160 dBm - 148 dBm
Position Accuracy (horizontal)	< 2.5 m CEP (autonomous) < 2.0 m CEP SBAS
Time To First Fix	
Hot Start[1]	< 1 s
Warm Start[2]	< 35 s
Cold Start[3]	< 38 s
Navigation	
Update Rate	1 Hz/ 5 Hz Supported

COMMUNICATION

UART - NMEA (Default)	
NMEA message Switchable	GGA, RMC, GSA, GSV, VTG, GLL, ZDA
Baud rate Switchable	4,800 (default) 1,200 to 115.2k
Ports	Tx (NMEA output) Rx (NMEA input)
UART - SiRF Specific SSB/OSP	
SiRFbinary protocol	Protocol for SiRFstar product family up to SSIV
One Socket Protocol	Protocol extension for SiRFstarIV
Baud rate Switchable	57.6k (default) 1,200 to 115.2k
Ports	Tx (Binary output) Rx (Binary input)
SPI - NMEA/SiRF Specific	
Clock	Up to 6.8 MHz
Ports	DO (NMEA / Binary output) DI (NMEA / Binary input) SPI CLK (clock - input) SPI CS (chip select - input)
I2C - NMEA/SiRF Specific	
Clock	Up to 400 kbps
Ports	I2C DIO (NMEA / Binary input / output) I2C CLK (clock - input)

HIGHLIGHTS

SiRFnav™	High availability and coverage; improved TTFF in weak signal environments
SiRFaware™	Keeps module in a state of readiness for rapid navigation (hot start)
Jammer remover technology	Detects and removes up to 8 in-band jammers with minimal loss of sensitivity
A-GPS	Embedded Extended Ephemeris (SiRFInstantFix1) and Ephemeris Push support
MEMS I2C interface	Prepared to use additional sensor information for improved navigation
ROM-based design	Prepared to store configuration and calibration data and to allow firmware updates
Internal antenna	Best matched build-in antenna for easy integration

ENVIRONMENT

Temperature	
Operating	-40°C to +85°C
Storage	-40°C to +85°C
Humidity	Non condensing

POWER

Input voltage	3.0 to 3.6 VDC Nominal 3.3 VDC
Average current draw	
Full power mode (searching)	35 mA
Full power mode (tracking)	21 mA
Trickle Power Mode	9.2 mA
PTF mode	236 µA
MPM / SiRFaware	115 µA
Hibernate	25 µA

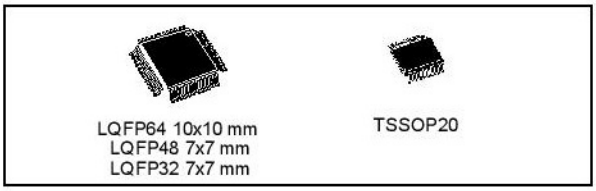
MECHANICAL

Dimensions	
L x W x H	17.8 x 16.5 x 7.1 mm
L x W x H	0.7" x 0.65" x 0.28"
Weight	4.0 g / 0.14 oz.

C.3 Technical details of microcontroller, STM32F030F4P6

Features

- Core: ARM® 32-bit Cortex®-M0 CPU, frequency up to 48 MHz
- Memories
  - 16 to 256 Kbytes of Flash memory
  - 4 to 32 Kbytes of SRAM with HW parity
- CRC calculation unit
- Reset and power management
  - Digital & I/Os supply: V_{DD} = 2.4 V to 3.6 V
  - Analog supply: V_{DDA} = V_{DD} to 3.6 V
  - Power-on/Power down reset (POR/PDR)
  - Low power modes: Sleep, Stop, Standby
- Clock management
  - 4 to 32 MHz crystal oscillator
  - 32 kHz oscillator for RTC with calibration
  - Internal 8 MHz RC with x6 PLL option
  - Internal 40 kHz RC oscillator
- Up to 55 fast I/Os
  - All mappable on external interrupt vectors
  - Up to 55 I/Os with 5V tolerant capability
- 5-channel DMA controller
- One 12-bit, 1.0 µs ADC (up to 16 channels)
  - Conversion range: 0 to 3.6 V
  - Separate analog supply: 2.4 V to 3.6 V
- Calendar RTC with alarm and periodic wakeup from Stop/Standby
- 11 timers
  - One 16-bit advanced-control timer for six-channel PWM output
  - Up to seven 16-bit timers, with up to four IC/OC, OCN, usable for IR control decoding
  - Independent and system watchdog timers
  - SysTick timer



- Communication interfaces
  - Up to two I^2C interfaces
    - one supporting Fast Mode Plus (1 Mbit/s) with 20 mA current sink,
    - one supporting SMBus/PMBus.
  - Up to six USARTs supporting master synchronous SPI and modem control; one with auto baud rate detection
  - Up to two SPIs (18 Mbit/s) with 4 to 16 programmable bit frames
- Serial wire debug (SWD)
- All packages ECOPACK®2

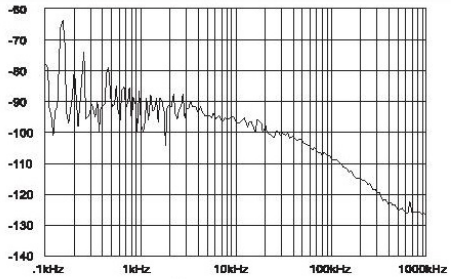
Table 1. Device summary

Reference	Part number
STM32F030x4	STM32F030F4
STM32F030x6	STM32F030C6, STM32F030K6
STM32F030x8	STM32F030C8, STM32F030R8
STM32F030xC	STM32F030CC, STM32F030RC

C.4 Technical details of signal generator, HM8135

3 GHz RF-Synthesizer HM8135
Valid at 23 °C after a 30 minute warm-up period

Frequency	
Range:	1 Hz to 3 GHz
Resolution:	1 Hz
Settling time:	‹ 10 ms
Frequency Reference 10 MHz	
Standard: TCXO	
Stability (0 to 50°C):	≤ ± 0.5 ppm
Aging:	≤ ± 1 ppm/year
Option: OCXO (HO85)	
Stability:	≤ ± $1x10^{-8}$
Aging:	≤ ± $5x10^{-9}$/year
Internal reference output:	(rear panel)
Level:	TTL
External reference input:	(rear panel)
Level:	› 0 dBm
Frequency:	10 MHz ± 20 ppm
Spectral purity (without modulation)	
Harmonics:	≤ - 35 dBc (typ.)
Non-harmonics:	≤ - 50 dBc (› 15 kHz from carrier)
Sub-harmonics:	≤ - 50 dBc (typ.)
Phase noise:	(at 20 kHz from carrier)
f ‹ 16 MHz:	≤ - 120 dBc/Hz
16 MHz ≤ f ‹ 250 MHz:	≤ - 95 dBc/Hz
250 MHz ≤ f ‹ 500 MHz:	≤ - 105 dBc/Hz
500 MHz ≤ f ‹ 1000 MHz:	≤ - 100 dBc/Hz
1 GHz ≤ f ‹ 2 GHz:	≤ - 95 dBc/Hz
2 GHz ≤ f ‹ 3 GHz:	≤ - 90 dBc/Hz
Residual FM:	typ. ‹ 4 Hz; ≤ 6.5 Hz (in 0.3 - 3 kHz bandwidth)
Residual AM:	typ. ‹ 0.06 % (in 0.03 - 20 kHz bandwidth)



(Typical phase noise at 1 GHz)

Output level	
Range:	-135 to +13 dBm
Resolution:	0.1 dB
Precision f ‹ 1.5 GHz; level › - 120 dBm	
for level › - 57 dBm:	≤ ± 0.5 dB
for level ‹ - 57 dBm:	≤ ± (0.5 dB + (0.2 x (-57 dBm – level))/10)
Precision f › 1.5 GHz; level › - 120 dBm	
for level › - 57 dBm:	≤ ± 0.7 dB
for level ‹ - 57 dBm:	≤ ± (0.7 dB + (0.5 x (-57 dBm – level))/10)
Impedance:	50 Ω
V.S.W.R.:	
f ≤ 1 GHz:	≤ 1.5
f › 1 GHz:	≤ 2.5
Modulation sources	
Internal:	10 Hz - 200 kHz sine wave 10 Hz - 20 kHz square wave, triangle, sawtooth
Resolution:	10 Hz
External:	(input on front panel)
Impedance:	10 kΩ II 50 pf
Input level:	2 V_{pp} for full scale
Coupling:	AC or DC
Output:	front panel
Level:	2 V_{PP}
Impedance:	1 kΩ
Ext. frequency resp. (to - 1dB):	10 Hz to 100 kHz for AC
Distortion:	‹ 2 % (AM-depth ≤ 60 %, fmod ≤ 1 kHz) ‹ 6 % (AM-depth ≤ 80 %, fmod ‹ 20 kHz)
Frequency modulation	
Source:	internal or external
Deviation:	± 200 Hz to 400 kHz (depending on frequency band)
Resolution:	100Hz
Accuracy:	± 3% + residual FM (fmod ≤ 5 kHz) ± 7% + residual FM (5 kHz ‹ fmod ‹ 100 kHz)
Ext. frequency response: (to – 1dB):	
DC coupling:	0 to 100 kHz
AC coupling:	100 Hz to 100 kHz
Distortion:	‹ 1 % for deviation ≥ 50 kHz at 1 kHz ‹ 3 % for deviation ≥ 10 kHz
Phase modulation	
Source:	internal or external
Deviation:	
‹ 16 MHz:	0 to 3.14 rad
› 16 MHz:	0 to 10 rad
Resolution:	0.01 rad
Accuracy:	± 5 % to 1 kHz + residual PM
Ext. frequency response (to – 1dB):	
DC coupling:	0 to 100 kHz
AC coupling:	100 Hz to 100 kHz
Distortion:	‹ 3 % for fmod = 1 kHz and deviation = 10 rad
FSK modulation	
Range (F0 – F1):	16 to 3000 MHz
Mode:	2 FSK levels
Data source:	external
Max. rate:	10 kbit/s
Shift (F1 – F0):	0 to 10 MHz
Resolution:	100 Hz
Accuracy:	see under FM
PSK modulation	
Mode:	2 PSK levels
Data source:	external
Max. rate:	10 kbit/s
Shift (Ph1 – Ph0):	
‹ 16 MHz:	0 to ± 3.14 rad
› 16 MHz:	0 to ± 10 rad
Resolution:	0.01 rad
Accuracy:	see under PM
Pulse modulation	
Source:	external (rear panel)
Dynamic range:	
f ‹ 2 GHz:	› 80dB
f › 2 GHz:	› 55dB
Rise/fall times:	‹ 50 ns (typ. ‹ 10 ns)
Delay:	‹ 100 ns
Max. frequency:	2.5 MHz (typ. 5 MHz)
Input level:	TTL
Sweep mode	
Range:	1 MHz to 3000 MHz
Depth:	500 Hz to 2999 MHz
Sweep time:	20 ms to 5 s
Trigger:	internal

Protective functions

The synthesizer is protected against reverse power applied on RF output up to 1 W for a 50 Ω source and against any DC source up to ± 7 V. The protection disconnects the output until manually rearmed by operator.

Miscellaneous	
Interfaces:	RS-232 (standard), IEEE-488 (optional), USB (optional)
Configuration memories:	10
Safety class:	Safety Class I (EN61010-1)
Power supply:	115/230V ± 10%, 50/60Hz
Power consumption:	approx. 40 VA
Operating temperature:	+ 10 to + 40 °C
Max. relative humidity:	10 to 90 % (without condensation)
Dimensions (W x H x D):	285 x 75 x 365 mm
Weight:	approx. 5 kg

C.5 Implementaion of DLL on FPGA kit, XAPP 132

The circuit shown in Figure 9 resembles the BUFGDLL macro implemented in such a way as to provide access to the RST and LOCKED pins of the CLKDLL.

The dll_standard files in `xapp132.zip` show the VHDL and Verilog implementation of this circuit.

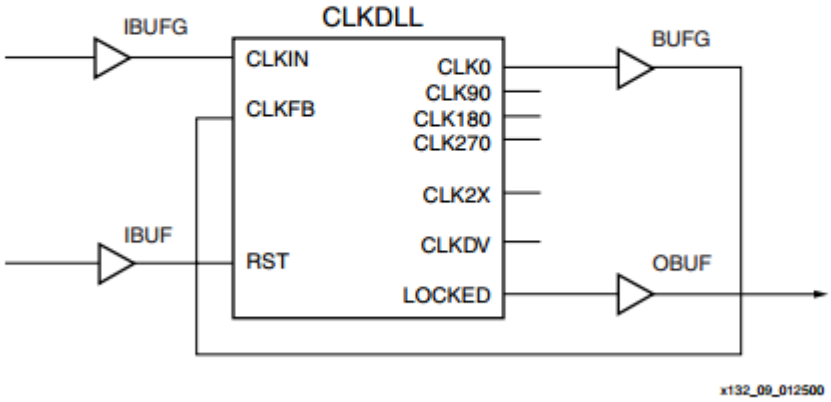


Figure 9: **Standard DLL Implementation**