University of Nottingham
UK | CHINA | MALAYSIA

Department of Electrical and Electronic Engineering
University of Nottingham

Develop a GPS/GNSS receiver: Design of Correlators for GNSS Receivers Based on VHDL

Author **Yaowen Hu(20495331)**
Supervisor **Dr Paul Blunt**
Moderator **Dr Sendy Phang**
Date **September 2023**

Project thesis submitted in part fulfilment of the requirements for the degree of Master of Science **Electronic Communications and Computer Engineering**, The University of Nottingham.

Abstract

GPS/GNSS has an important position and role in daily life production as well as in military field. A set of good performance of GPS receiver can be half the effort. This paper uses VHDL language to design the tracking module of GPS/GNSS receiver, which is one of the core modules of the receiver. I will introduce the principle and design method of each submodule respectively, and finally complete the design of the correlator.

Acknowledgements

I wish to express my sincere appreciation to the following individuals who played vital roles in the completion of my dissertation. Dr. Paul Blunt, my supervisor, for generously sharing his extensive knowledge and answering my questions. Dr. Sendy Phang, my moderator, for her meticulous guidance and insightful questions that helped shape my work. My team members, Emmanuel Ortsin and Xingjian Fu, for their discussions and unwavering encouragement.

I also extend my gratitude to my family and girlfriend for their constant support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In an era where precise positioning, navigation, and timing have become fundamental requirements for numerous applications, the *Global Navigation Satellite System* (GNSS) has emerged as a transformative technology. *Global Positioning System*, commonly known as GPS, has revolutionized the way we navigate, communicate, and interact with our surroundings. As the demand for accurate and reliable GNSS positioning continues to grow, there is an increasing need to explore innovative methods to develop advanced GNSS receivers capable of handling diverse and challenging environments.

The aim of this project is to design and develop a GPS/GNSS receiver, focusing on the critical component of correlators, which form the backbone of any GNSS receiver. Correlators play a vital role in the tracking part of the GNSS positioning process, making them a key element in the overall receiver architecture.

1.1 Background

The GNSS has become an integral part of modern life, transforming how we navigate, communicate, and interact with the world around us. As the demand for accurate and reliable positioning information continues to grow, there is a constant need for advancements in GNSS receiver technology. The receiver's tracking module, responsible for extracting precise positioning data from satellite signals, plays a critical role in ensuring the accuracy and robustness of the overall GNSS positioning process.

- Increasing Reliance on GNSS: In recent years, there has been a substantial increase in the reliance on GNSS technology across various sectors and industries. From commercial transportation and aviation to precision agriculture, emergency response, and scientific research, GNSS has become a ubiquitous tool in countless applications. The accurate positioning and timing information provided by GNSS have not only improved operational efficiency but have also enabled the development of innovative services and solutions.

- Importance of the Tracking Module: The tracking module is a vital component of any GNSS receiver, responsible for acquiring and locking onto the weak satellite signals and maintaining their synchronization. Efficient tracking algorithms can increase SNR (*Signal-to-Noise-Ratio*) which are essential for accurate and real-time positioning, especially in dynamic environments where satellite signals may be temporarily obscured or weakened. Optimizing the tracking module's performance is crucial to enhancing the overall receiver's sensitivity, accuracy, and responsiveness to varying environmental conditions.

- Opportunities for Hardware Optimization: As the demand for high-performance GNSS receivers increases, there are opportunities to explore hardware optimization techniques to improve tracking capabilities. Implementing the tracking module using VHDL (*Very High-Speed Integrated Circuit Hardware Description Language*) offers advantages such as parallel processing and hardware acceleration, allowing for real-time tracking and reduced power consumption. The utilization of VHDL enables the development of customized *Application-Specific Integrated Circuits* (ASICs) or *Field-Programmable Gate Arrays* (FPGAs) tailored to the tracking function, leading to more efficient and specialized GNSS receivers.

1.2 Aims and Objectives

1.2.1 Aims

The project has three aims: first, to design a tracking processor for GNSS receivers using VHDL; second, to verify the results using MATLAB; and third, to learn about GNSS signal acquisition and tracking by reading papers, books, etc.

1.2.2 Objectives

- Literature survey on GNSS signal processing and tracking
- Be familiar with the principles of correlation functions, correlators, VHDL and Vivado
- Write a literature review
- Create and verify a set of correlators model in MATLAB
- Write the RTL (*Register Transfer Level*) code of the correlator in VHDL
- Write the testbench code of the correlator in VHDL
- Instance several times (e.g. three times) of the correlator VHDL code to implement the tracking part of the GNSS receiver
- Write the testbench code of the project
- Compare the results from VHDL with those from MATLAB
- Record and analyse the VHDL results against the MATLAB results
- Complete the final dissertation

1.3 Description of the Work

I will design the correlators of the receiver using VHDL.I plan to design a set of correlators in an FPGA. The correlators can be used to capture GNSS signals to allow subsequent

devices to perform ranging and positioning. My front-end device will acquire the GNSS signal and transmit it to me. During the acquisition process I can receive at least one GNSS signal. I will extract the ranging code from it and compare it with the ranging code generated by the receiver, i.e. using a correlator.

Chapter 2

Literature Review

The literature review chapter serves as a critical foundation for this project, offering an in-depth exploration of existing research, developments, and advances in the field of GNSS receivers, with a particular focus on the tracking module. This chapter aims to identify the key theories, methodologies, and technologies that have shaped the evolution of GNSS tracking and provide valuable information for the design and development of the FPGA-based GNSS tracking module.

2.1 GPS/GNSS

GPS/GNSS consists of a constellation of satellites orbiting the Earth, transmitting continuous signals containing precise timing and positioning information. Each GNSS satellite is equipped with atomic clocks, ensuring high accuracy in the signals it emits. The GNSS constellation comprises multiple satellites, typically in *Medium Earth Orbit* (MEO) [1], providing global coverage to ensure that a sufficient number of satellites are visible from any point on Earth at any given time.

2.1.1 Functioning of GNSS

A GNSS receiver on the ground intercepts signals from multiple satellites in the constellation. By analysing the timing and phase information in these signals, the receiver can calculate the distance between itself and each satellite. By triangulating the dis-

tances from multiple satellites, the receiver can determine its precise three-dimensional position (latitude, longitude, and altitude). Additionally, the receiver can synchronize its internal clock with the highly accurate satellite atomic clocks, providing precise timing information.

2.1.2 Key Components of GNSS

GNSS commonly consists of these four components: satellites, GNSS receivers, control segment, and user segment [2].

Satellites: The heart of GNSS is the constellation of satellites orbiting the Earth. Each satellite broadcasts signals carrying unique identification information and precise timing data.

GNSS Receivers: GNSS receivers are devices that intercept and process the satellite signals to calculate the user's position, velocity, and time. These receivers can be integrated into various devices, such as smartphones, car navigation systems, aviation equipment, and scientific instruments.

Control Segment: The control segment consists of ground-based monitoring stations and control centres responsible for monitoring the health of the satellites, maintaining their orbits, and ensuring accurate timing information.

User Segment: The user segment encompasses the GNSS receivers used by individuals, industries, and organizations to access positioning, navigation, and timing services.

2.1.3 Applications of GNSS

The applications of GNSS are diverse and far-reaching, permeating nearly every aspect of modern life. Some key applications include [3]:

- Navigation: GNSS enables precise and real-time navigation for land, sea, and air transportation, making it a critical component of navigation systems in vehicles, ships, and aircraft.

- Surveying and Mapping: GNSS is widely used in geodetic surveying, mapping, and

cartography to obtain accurate geographic data for urban planning, construction, and land management.

- Precision Agriculture: GNSS-based systems optimize agricultural processes by providing precise positioning for automated machinery, crop monitoring, and targeted application of resources like fertilizers and pesticides.

- Emergency and Disaster Response: GNSS aids emergency services in locating and coordinating responses during disasters, enabling efficient search-and-rescue operations.

- Timing and Synchronization: GNSS provides highly accurate timing information, essential for the synchronization of critical infrastructure, such as power grids, telecommunication networks, and financial systems.

- Scientific Research: GNSS data is used in scientific research, including the study of tectonic movements, sea level changes, atmospheric monitoring, and climate research.

In conclusion, GPS/GNSS has transformed the way we navigate and interact with our environment. By leveraging signals from a constellation of satellites, GNSS provides precise positioning, navigation, and timing services, powering applications across diverse sectors and enriching various aspects of modern life. As technology continues to evolve, GNSS is expected to play an increasingly critical role in shaping our interconnected world.

2.1.4 Why Choose GPS/GNSS

On 4^{th} September 2019, a lecture[4] was given at the "*International Colloquium on Scientific and Fundamental Aspects of GNSS*". Spacecraft mission design, astrodynamics, space navigation, software development, and space politics are among the areas of competence of the author, Joel Parker, a flight dynamics engineer at NASA Goddard Space Flight Center. The "*Transiting Exoplanet Survey Satellite*" (TESS) project, which started in 2017, seeks to find planets in the habitable zones of other stars. He actively participates in the flight dynamics team for this mission.

The presentation provides a comprehensive overview of the history and scope of GPS/GNSS, as well as its potential future applications in lunar exploration and beyond.

Numerous flight examples are presented to illustrate the indispensable role of GPS in the space sector. This underscores the critical importance of GNSS technology in space missions. The author then delves into new areas of GPS application, specifically focussing on lunar exploration. The concept of a GNSS receiver designed for the Artemis Project is introduced.

The report concludes with a summary of key development directions:

- Studying GNSS improvements and capabilities for use on the moon
- Internal and external cooperation with the user and supplier communities via the ICG (*International Committee on GNSS*), to guarantee signal quality and data availability
- Making use of tried-and-true antenna and receiver technology to solve technical problems
- Making flight demonstrations as moon exploration efforts are stepped up internationally
- Using operational programs to optimize the advantages of exploration and science

This presentation illustrates the success of GNSS applications and demonstrates their importance. This is one of the reasons why I chose this topic. It shows s a high level of professionalism and will undoubtedly prove valuable for your MSc. project, particularly in providing a thorough introduction to the background of GPS/GNSS and its ongoing development.

2.1.5 GNSS Signal Plan

In complex engineering projects and system development, clear and effective communication between various subsystems and components is essential for successful integration

and operation. The *Interface Control Document* (ICD) of GNSS provides the basic information of the system, such as *Intermediate Frequency* (IF), modulation scheme, code frequency, etc.

After reviewing the interface control documents [5–10] of some systems, we have summarized the parameters of each system in Table 2.1.

Table 2.1: Specifications of Systems

Specifications	GPS	GLONASS	Galileo	BeiDou
Frequency band	L1: 1575.42 MHz L2: 1227.6 MHz L5: 1176.45 MHz	L1: 1602MHz L2: 1246MHz (14 channels)	E1: 1575.420MHz E6: 1278.750MHz E5a: 1176.450MHz E5b: 1207.140MHz	B1c : 1575.42MHz B2a: 1176.45MHz B2b: 1207.14MHz B1I: 1561.098MHz B3I: 1268.52MHz
Band width	Block IIR, IIRM, and IIF: 20.46 MHz GPS III, GPS IIIF, and subsequent: 30.69 MHz	L1: 7.875MHz(562.5 kHz each) L2: 6.125MHz(437.5 kHz each)	E1: 24.552MHz E6: 40.920MHz E5a: 20.460MHz E5b: 20.460MHz	B1c: 32.736MHz B2a: 20.46MHz B2b: 20.46MHz B1I: 4.092MHz B3I: 20.46MHz
Modulation scheme	BPSK	Modulo-2 addition CDMA	E1: CBOC E5: AltBOC E6: BOC	B1c: QMBOC(6, 1, 4/33) Others: BPSK
Antenna polarization	RHCP*			
Chip rate	L1 C/A & P: 1.023MHz L2 CL & CM: 511.5 kHz L5 data & channel: 10.23 MHz	L1 C/A: 0.511MHz L1 P: 5.11MHz L2 C/A: 0.511MHz L2 P: 5.11MHz	E1 ranging Code: 1.023MHz E6: 5.115MHz E5: 10.230MHz	B1c ranging code: 1.023MHz B2a ranging code: 10.23MHz B2b ranging code: 10.23MHz B1l ranging code: 2.046MHz B3l ranging code: 10.23MHz

Note: *RHCP: *Right Hand Circularly Polarized*

Given that the GPS L1 system was the first to be deployed[11], it possesses a relatively straightforward structure. Based on the GPS L1 ICD[5], we can summarize the signal plan of GPS L1 in table 2.2.

Table 2.2: Specifications of GPS L1

Specifications	GPS	
Service name	C/A	LIC
Centre frequency	1575.42MHz	1575.42 MHz
Frequency band	L1	L1
Access technique	CDMA	CDMA
Signal component	Data	Data Pilot
Modulation	BPSK	TMBOC(6,1,1/11)
Code frequency	1.023 MHz	1.023 MHz
Primary PRN code length	1,023	10,230
Code family	Gold Codes	Weil Codes
Data rate	50 bps/50 sps	50 bps/100 sps N/A

As a result, the GPS L1 signal has been selected as the primary target signal for this project. Furthermore, it is important to note that the L5 signal has ten times the bandwidth of L1. Thus, in the event that the utilization of the L5 signal becomes necessary, the adjustments required for its implementation would be minimal.

2.2 Tracking Process

2.2.1 Book from P. D. Groves[12]

This book offers a thorough textbook on navigation systems, which make use of GNSS, *Intelligent Navigation Systems* (INS), and other sensors to deliver precise *positioning, navigation, and timing* (PNT) data. Paul D. Groves and Kayton M. Ned, two renowned authorities in the subject of navigation systems, are the esteemed authors behind this work.

Professor of satellite navigation at University College London, Paul D. Groves' research focuses on navigation algorithms and technology. Kayton M. Ned, on the other hand, was an expert in navigation and control systems for aerospace applications and was a former Stanford University professor of aeronautics and astronautics. With their wealth of knowledge and experience, they are qualified to write a thorough and reliable textbook on navigation systems.

The basics of GNSS systems are described in Section 8.1, which will provide us a foundational understanding. It explains how to find the receiver's distance from the satellite.

The user equipment in Chapter 9 refers to the receiver. It describes how GNSS receivers are made. An antenna, reception hardware, range processor, and navigation processor are all components of a GNSS receiver. The tracking component of the ranging processor is the main focus of my work. This chapter makes note of the six correlators that make up a basic receiver design and the frequent use of additional correlators to quicken signal collection. There is also a comprehensive flowchart of the capture procedure available.

In general, as this book is a textbook, it provides a very authoritative and accurate introduction to GNSS-related technologies. It is reliable and thought-provoking.

2.2.2 Book from E. D. Kaplan and C. Hegarty[3]

Elliott D. Kaplan has made significant contributions to the development and understanding of GNSS technology. Christopher J. Hegarty is also a prominent figure in the GNSS community. He has extensive experience with GPS and other satellite navigation systems. They have been associated with the MITRE Corporation and contributed to the advancement of GNSS technology and its applications.

The book delves into the fundamental principles underlying GNSS technology, including GPS, GLONASS, Galileo, BeiDou, and others. It explores the concepts of satellite orbits, signal structure, receiver design, and various positioning techniques used in GNSS applications.

Chapters 3 to 7 provide a systematic introduction to the various navigation systems, including GPS, GLONASS, Galileo, BeiDou, etc. It is of great help for the background part of the project. Chapter 8 details the design of the receiver and introduces the antenna, RF(*Radio Frequency*) front-end, acquisition and tracking respectively, as well as the design of the loop filter. This part was extremely helpful to my understanding. It describes the working principle and the GNSS process through numerous mathematical equations.

2.2.3 Paper from J. C. Juang, Y. H. Chen et al.[13]

The paper titled "*Design and implementation of an adaptive code discriminator in a DSP/FPGA-based Galileo receiver*" is authored by Jyh Ching Juang, Yu Hsuan Chen, Tsai Ling Kao, and Yung Fu Tsai from National Cheng Kung University.

The coded tracking loop and its related discriminator play a significant role in the tracking performance of their work on the GNSS receiver design. The authors suggested a plan that was put into practice on a DSP/FPGA board to improve tracking performance. The GIOVE-A signal was used for testing in experiments, and the outcomes showed the benefits of their suggested code tracking architecture and discriminator design.

The paper presents a well-defined design methodology for an adaptive coding discriminator, which was subsequently implemented and thoroughly tested. This design approach is framed as an optimization problem leading to the development of two discriminators. Notably, the adaptive *noncoherent multi-correlator* (NMC) discriminator showcased improvements in transient response and tracking error.

Throughout the article, detailed mathematical principles of capture, tracking, and other processes are elaborated, along with a description of the correlator's design ideas. Numerous test results for correlators based on the new approach are provided, making this paper a valuable reference for my MSc project.

2.2.4 Paper from O. Jakubov, P. Kovar et al.[14]

To develop advanced GNSS signal processing algorithms, such as multi-constellation, multi-frequency, and multi-antenna navigation, a flexible and re-programmable *software-defined radio* (SDR) solution is essential. To achieve this goal, the researchers have introduced various receiver architectures. Their chosen approach involves constructing an RF front-end with an FPGA universal correlator, mounted on an ExpressCard directly connected to a PC. This setup allows GNSS researchers and engineers to write signal processing algorithms (e.g., tracking, acquisition, and localization) in a Linux application programming interface. The unique hardware configuration enables easy modification of the RF front-end via a PC program, providing the flexibility to increase the number of RF channels, correlators, or antennas by attaching more ExpressCards to the PC, thus enhancing computational capability.

Utilizing the SDR platform, they have successfully implemented a GNSS receiver, offering significant advantages, including low cost, high software customization, and the potential for straightforward upgrades in computational power. The article also discusses a generic FPGA-based GNSS receiver, allowing a comparison of the different approaches' advantages and disadvantages.

The outcome of their work is the prototype of the Witch Navigator receiver. They have conducted successful tests on Galileo E5, E1b, and E1c signals with the correlator and the RF front-end. Moreover, they have fully addressed the communication between the FPGA and the PC, developing and testing all the corresponding controllers and drivers.

The article provides a comprehensive analysis of the design principles for receivers based on FPGAs, and the SDR platform enables the validation of the FPGA platform through MATLAB simulations. This combination of approaches facilitates the development of advanced and adaptable GNSS signal processing algorithms for future navigation systems.

2.3 FPGA Technique

FPGA is a type of integrated circuit that allows users to configure its hardware functionality after manufacturing. Unlike ASICs, which are designed for specific tasks and cannot be reconfigured, FPGAs offer flexibility and programmability. This characteristic makes FPGAs suitable for a wide range of applications [15], including digital signal processing, communication systems, image and video processing, and in this case, GNSS receiver design.

Here are the advantages of using FPGA in the project [16]:

- Hardware Customization: FPGAs allow customization of hardware functionality, tailoring specific algorithms and designs for the GNSS tracking module. This flexibility optimizes hardware resources, resulting in a specialized and efficient GNSS receiver.

- Real-Time Processing: FPGAs excel in parallel processing, enabling real-time data processing with low latency. This is crucial for continuous and accurate GNSS positioning, especially in dynamic environments with intermittent satellite signals.

- Power Efficiency: FPGA designs can be optimized for high performance with low power consumption. This is essential for battery-operated GNSS devices, extending battery life and improving overall device usability.

- Rapid Prototyping and Iteration: FPGA development allows quick prototyping and iterative design refinement. Changes can be implemented and tested rapidly, speeding up development and performance improvements.

- Adaptability and Future Upgrades: The programmable nature of FPGAs allows easy updates and future upgrades. GNSS algorithms can be incorporated without hardware changes as technology advances.

- Cost-Effectiveness: FPGAs offer a cost-effective solution for custom hardware design compared to ASICs. They are suitable for smaller-scale projects or research without requiring expensive fabrication.

In summary, the choice of FPGA to design the GPS receiver is a no-brainer.

Chapter 3

Methodology

The methodology chapter is a pivotal component of this research, unveiling the systematic approach taken to address the specific goals and inquiries of the project. Within this chapter, an in-depth description of the research design, data collection methodologies, and analytical techniques deployed is meticulously presented. By offering a transparent and methodical account of the research process, this chapter safeguards the robustness and integrity of the study's outcomes.

3.1 Tracking

A complete GPS/GNSS work process includes acquisition, tracking and navigation. My project is mainly focused on the tracking stage.

After the acquisition phase, where the receiver identifies and locks onto the satellite signals, the tracking phase begins. During this phase, the receiver closely monitors the received signals and tracks their variations to accurately determine the user's position, velocity, and timing information. This involves maintaining a stable lock on the satellite signals despite various challenges, such as signal degradation due to atmospheric conditions, obstructions, and interference.

The following figure 3.1 shows the basic architecture of the tracking module. The signal is received by the RF front-end and is filtered and fed to the ranging processor. At the same time, the NCO (*Numerically Controlled Oscillator*) generates a GPS L1 IF

carrier and a carrier for the C/A (*Coarse Acquisition*) code. The GPS signal is first modulated to IF and then multiplied with the C/A code and finally integrated. Using the principle of the cross-correlation function, the C/A carrier frequency is continuously adjusted to synchronize with the C/A code in the GPS signal. And finally, complete the tracking[17].

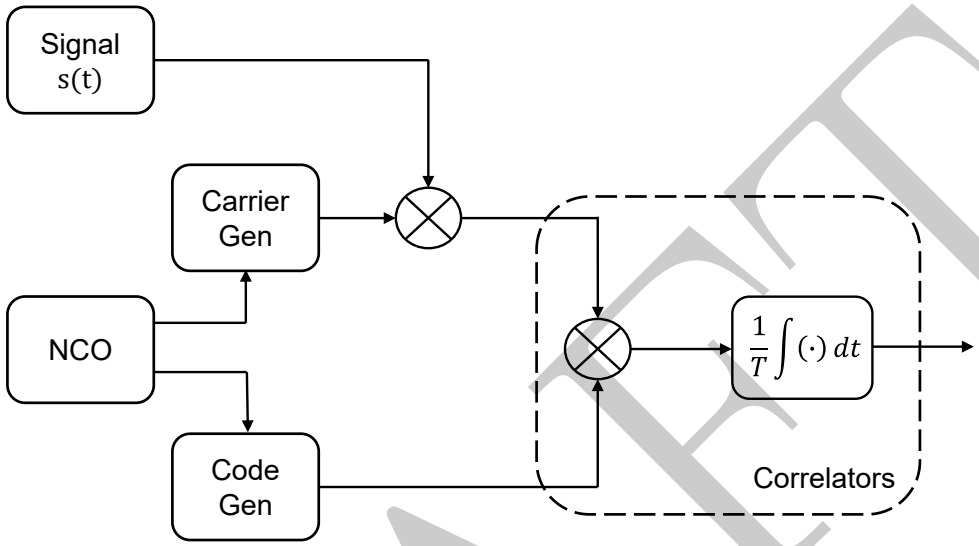


Figure 3.1: Tracking Module Architecture

3.1.1 NCO

In digital communications, it is often necessary to modulate and shift baseband signals (often IF) to high frequencies for transmission because the wavelengths of the high-frequency signals are better matched to the available antenna sizes. In order to modulate, we need to generate a carrier for the high-frequency signal, which is often a sine or cosine signal. Therefore, a module is required to generate the carrier at the desired frequency consistently and accurately.

Using hardware, we have three ways to generate such signals: direct form oscillator, NCO, and CORDIC algorithm. After comparing, we learnt that using the CORDIC takes up the least amount of resources in FPGAs[18], however, using the NCO is the simplest solution. In this project, we have designed an NCO module in FPGA to generate different frequency carriers.

The NCO is a signal generator that produces a specified frequency. It can generate

square wave signals, i.e. PWM signals with a duty cycle of 50%, as well as sinusoidal cosine signals and so on[19]. It is often used in conjunction with a DAC (*Digital-to-analog Converter*) so that an analogue signal of a specified frequency can be output. In general, the NCO consists of two parts, the *Phase Accumulator* (PA) and the *Look-up Table* (LUT)[20]. Their architecture is shown in figure 3.2.

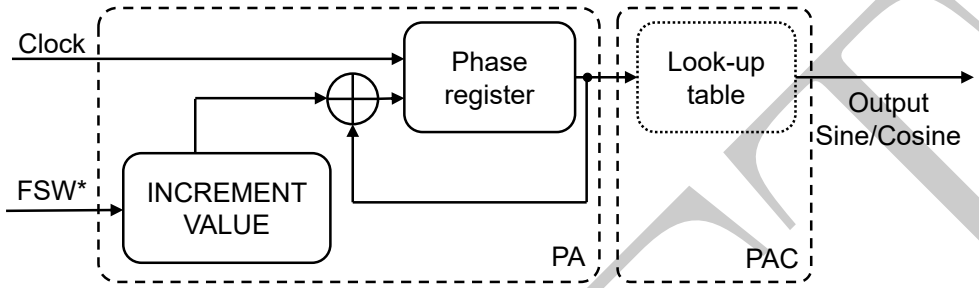


Figure 3.2: NCO Architecture
Note: *FSW: *Frequency Setting Word*

Phase Accumulator

The phase accumulator will complete an accumulation in each clock cycle according to the set increment value. In other words, the accumulator outputs the address of the look-up table so that the look-up table generates the correct sine-cosine signal[21]. By setting the increment value, the look-up table is made to sample in a controlled manner.

Here, we assume that the bit width of the accumulator is 32 bits. The ratio of the incremental 32-bit value to the fixed 4,294,967,296 accumulator overflow value determines how often the overflow occurs. This controls the triggering of the NCO output waveform. The equation is as follows,

$$F_{out} = (increment\ value) \times \frac{F_{clock}}{2^{Accum\ width}} \tag{3.1}$$

For example, if the clock is 99.375MHz and the NCO is required to generate a 1.023MHz signal with a register bit width of 32 bits, then the increment value 44,213,852 needs to be written to FSW.

Phase-to-Amplitude Converter

By using the look-up table, we can convert the phase value into an outputable sine-cosine signal. In other words, the look-up table is actually a *Phase-to-Amplitude Converter* (PAC). The easiest way to build it is to use *Read-only Memory* (ROM)[22], create the amplitude data in advance using software such as MATLAB, and then import it into a memory file or directly into the VHDL code.

This look-up table contains all the magnitude values in a cycle. The magnitude values corresponding to the phases are rounded and stored in the table. This table can be thought of as a matrix. Assume that the bit width of each element is N bits. N is an integer power of 2. The calculation of the indexes in the table will be greatly simplified[23]. In this case, a sine-cosine signal can be output simply by selecting the appropriate m bits in the phase representation.

In fact, we can save resources by "cunningly" designing look-up tables[24]. Such an operation also allows us to easily control the resolution of the NCO output. Below I will describe the method of designing a look-up table.

Firstly, according to the subsection 3.1.1(Phase Accumulator), we are able to plot the phase function. Figure 3.3Below is the phase function that I am plotting using MATLAB.

The phase will be accumulated at each cycle until it overflows and starts again. In fact, this function completes the conversion from the time domain to the phase domain. Next, we will convert the phase domain to the amplitude domain, which is the PAC. The following figure also shows the PAC function generated using MATLAB.

In figure 3.4, one prefers to use the cosine function with high resolution, i.e. the blue line. This then consumes a lot of resources as we need to set the magnitude values for each phase and the size of the look-up table will become huge. And then such a high resolution is redundant for us. So, when we don't need such high resolution, we can "cunningly" design the look-up table to reduce the resolution, e.g. the red lines.

Assuming, the phase overflow value is 1,024, i.e. 10 bits wide, and we need 8 dots per cycle, then one dot can be generated every 128 phases. Specifically, each phase interval holds a different magnitude value. Then, we can calculate that the lookup table or ROM

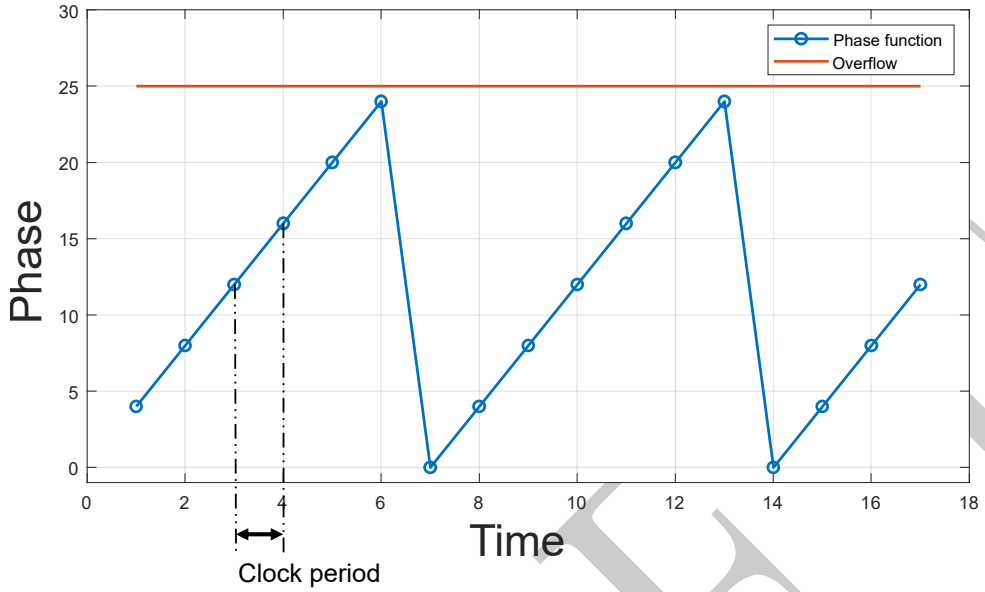


Figure 3.3: Phase Function

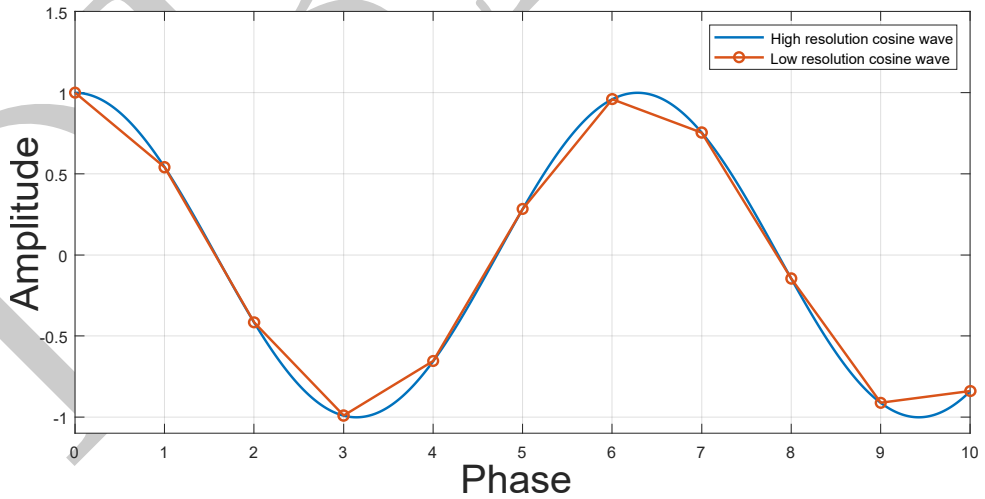


Figure 3.4: PAC Function

has a depth of 8 and a width of 10 bits, occupying 80 bits. In fact, the first interval is $0 \sim 127$, i.e. $00\ 0000\ 0000_{(2)} \sim 00\ 0111\ 1111_{(2)}$, the next interval is $00\ 1000\ 0000_{(2)} \sim 00\ 1111\ 1111_{(2)}$, and so on. We can find that we only need to know the highest three bits to obtain the current amplitude value. Then, we can calculate that the depth of the ROM is 8 and the width is 3 bits, occupying 24 bits. Using this approach can greatly reduce resource usage, as shown in the table below.

Table 3.1: ROM Size for Different Methods

Method	ROM depth	ROM width(bits)	Size(bits)	Size percentage change from previous method(%)
Original method	1024	10	10240	N/A
Method 1	8	10	80	-99.22
Method 2	8	3	24	-70

3.1.2 Code Generator

In GPS L1, there are two types of ranging codes, C/A code and P(*Precise*) code. C/A code sends 1,023 chips per 1ms and P code sends 10,230 chips per 1ms, which has a higher frequency and hence is more precise. Generally, P-codes are provided to military users and are encrypted for transmission. The encrypted P-code is known as the Y-code[25].

Different satellites require different unique C/A codes with good correlation and balancing properties. Therefore, the use of PRN(*Pseudorandom Noise*) codes is a natural fit[26].

Signals encoded using PRN achieve established levels of coding performance with good compatibility. As you can see from its name, the PRN code is not really a random code. It can be pre-calculated. It possesses a very small value of cross-correlation or auto-correlation. However, using it for signal processing becomes more complicated[27].

Here are three ways to generate C/A codes[28], the PRN codes for GPS,

- *Linear Feedback Shift Register* (LFSR)
- Memory codes

- Hash functions

In fact, the most common way is to use LFSR or memory codes.

LFSR

In simple, LFSR is a shift register. In order to generate the C/A code, we need two shift registers, called G1 generator and G2 generator. The result of exclusive-or(i.e., modulo-2 addition) between the output of G1 and the output of G2 is the C/A code. Before the exclusive-or, G2 needs to be delayed. The amount of delay is different for each satellite and this data can be queried in the ICD.

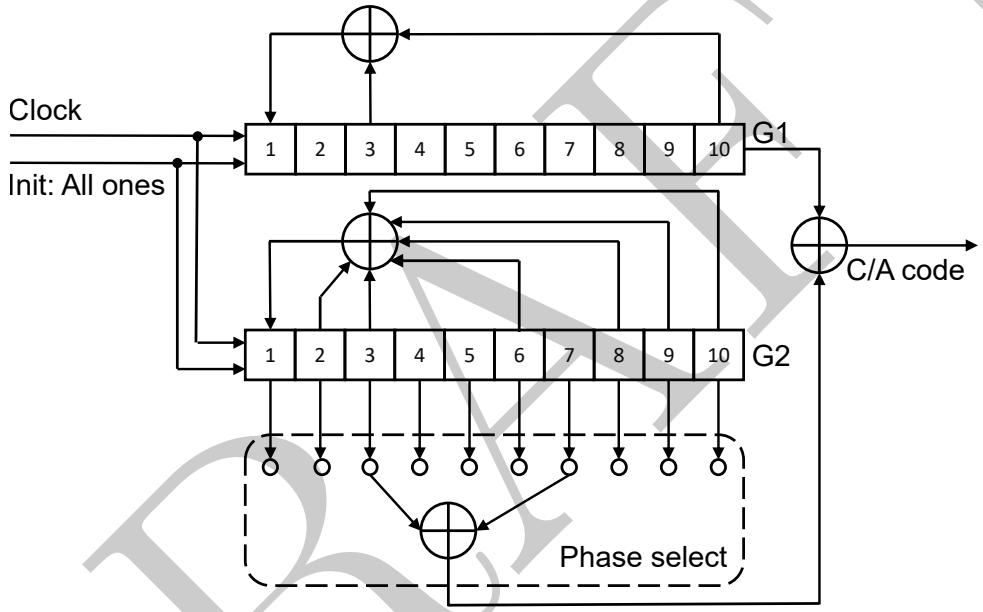


Figure 3.5: C/A Code Generator

Figure 3.5 illustrates the structural scheme of the generator. The RPN code design specification requires that the G1 shift register needs to feed the modulo-2 sums of the third and tenth levels back to the first level. Therefore, the polynomial generator for G1 is $G1 = 1 + X^3 + X^{10}$. In order for the G2 shift register to generate the specified delay, we need to feed the specified levels back to the first level after the modulo-2 addition operation according to the PRN assignments table. PRN No.2 is used here as an example, and after consulting the assignments table[5], we know that the phase selection for C/A is: $3 \oplus 7$. Therefore, we need to modulo-2 sum add the third and seventh stages as the output of G2. The polynomial of G2 is $G2 = 1 + X^2 + X^3 + X^5 + X^8 + X^9 + X^{10}$. Finally,

the outputs of G1 and G2 are modulo-2 added, and the result is the C/A code, or PRN №2 code. The clock port is driven by the 1.023MHz signal.

Memory Codes

Nowadays, in order to save the computational resources of the processor, we can store the pre-generated 1,023-bit PRN code in inexpensive ROM.

Pros and Cons of the LFSR and Memory Codes

LFSR:

- + Known mathematical properties (balance, correlation properties) across families of codes
- + Simple generation - one chip at a time with small resource requirements
- - Limited in number

Memory codes:

- + Potentially optimal
- - Complex selection
- - Requires memory on transmitter and receiver (all codes stored in non-volatile memory, possible latency issues)

In my project, I will use MATLAB to calculate the required PRN code in advance and store it in the ROM data type of VHDL

3.1.3 Correlator

The correlator is the core of the tracking module. It is used to track the received signal and generate the early, prompt, and late codes. The correlator is also called the *Code Tracking Loop* (CTL). The most important thing in a correlator is to perform correlation operations, which in this case involve cross-correlation operations.

Cross-Correlation

Correlation functions are divided into two categories: auto-correlation functions and cross-correlation functions. Correlation is a matching process that gives the similarity of two signals [29], defined by equation 3.2.

$$R_f(\tau) = \int_{-\infty}^{\infty} f(t)f(t+\tau)dt \quad for -\infty < \tau < \infty \tag{3.2}$$

Where $x(t)$ is the signal itself. When the functions in the integrated function are the same, the $R_x(\tau)$ is then called an auto-correlation function which measures how closely a signal matches a copy of itself and the delay between these two signals.

If two functions or signals are not equal, they are said to be cross-correlation functions [30], defined as,

$$R_{f\star g}(\tau) = \int_{-\infty}^{\infty} f(t)g(t+\tau)dt \quad for -\infty < \tau < \infty \tag{3.3}$$

When the signal is discrete, equation 3.3 should be transformed into equation 3.4. In fact, it's the scalar product of the two signals.

$$R_{f\star g}[n] = \sum_{m=-\infty}^{\infty} f(m)g(m+n) \quad for -\infty < m < \infty \tag{3.4}$$

Figure 3.6 shows a visual comparison of auto-correlation and mutual correlation. Where the index of the peak of the cross-correlation is the delay value.

That is, as two signals become more similar their correlation function becomes larger. In GPS/GNSS receivers, we need to use the cross-correlation function to measure the delay between the GPS signal and the local code to synchronise.

Correlator

The correlator is a digital circuit that performs the correlation operation between the received signal and the local code. The correlation operation is an inner-production operation. The correlation operation is performed by multiplying the received signal with

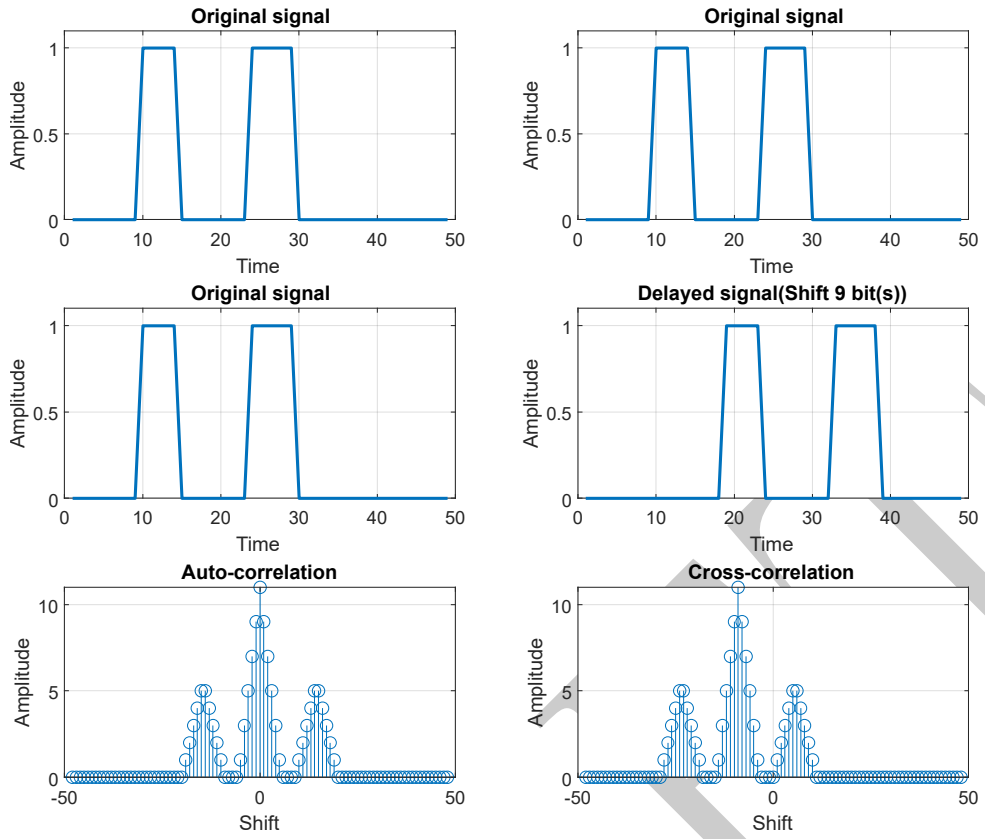


Figure 3.6: Comparison of Auto-correlation and Cross-correlation

the local code and summing them up.

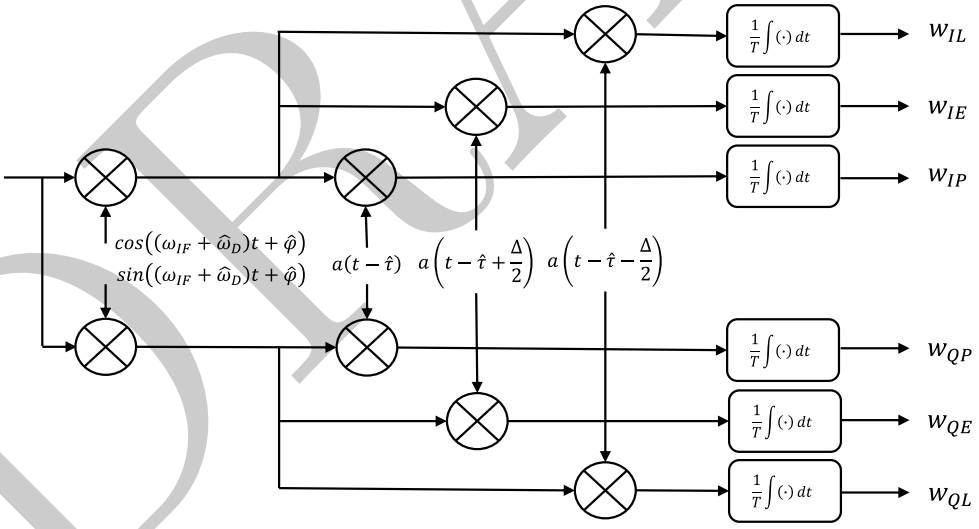


Figure 3.7: Architecture of the Correlator

The correlation operation usually needs 6 correlators to perform the correlation operation at the same time. In this case, the signal is processed in the I/Q phase respectively and each phase is required to multiply by the early, prompt, and late code respectively as well. Therefore, there will be 6 correlators. The massively parallel correlator arrays in

receivers allow for the parallel search of thousands of code bins which will save a lot of time[12]. The structure of the GPS correlator is shown in figure 3.7.

Figure 3.8 illustrates how the early, prompt and late code change as the phases of the C/A code signals are advanced with respect to the input signal. For ease of understanding, only the incoming signal is shown along with the 1-bit C/A code. In reality, the incoming PRN code is drowned in noise, and each correlator must accumulate a large amount, i.e., 1 ms of inner product, so that each envelope amplitude emerges from the noise.

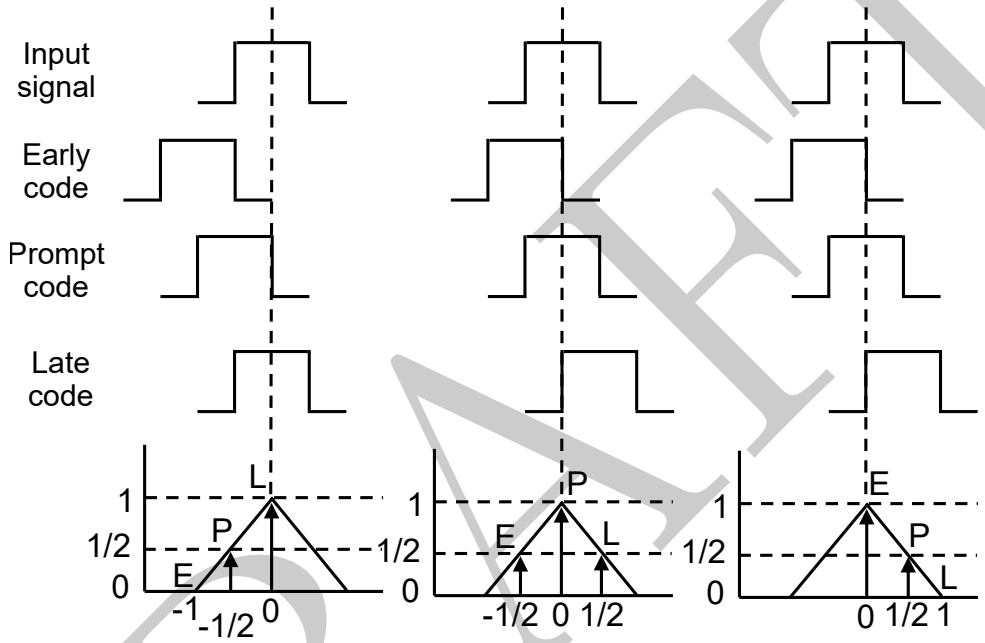


Figure 3.8: Code Correlation Phases

Successful tracking is proved only when the value of P is maximum and the value of E is equal to the value of L.

3.1.4 Doppler Effect

A typical scenario is one in which the receiver antenna is fixed to the earth and constantly receives signals from satellites in orbit. Using the antenna as a reference system, the satellite is constantly moving at high speed. The frequency of the signal from the satellite is fixed, while the frequency received on the ground is constantly changing, a phenomenon called the Doppler effect, which produces a Doppler shift.

The Doppler shift is,

$$\nu = \frac{v}{\lambda} \tag{3.5}$$

Where v is the relative velocity between the transmitter and receiver, which is negative if its distance becomes small and positive if the distance becomes large. λ is the wavelength of the signal [31]. In this example, however, the transmitter and receiver are moving in the same straight line. When this requirement is not met, we need to revise the formula.

Assuming that the receiver is moving at the speed of v, the angle between the moving direction and the wave propagation direction is γ as shown in figure 3.9.

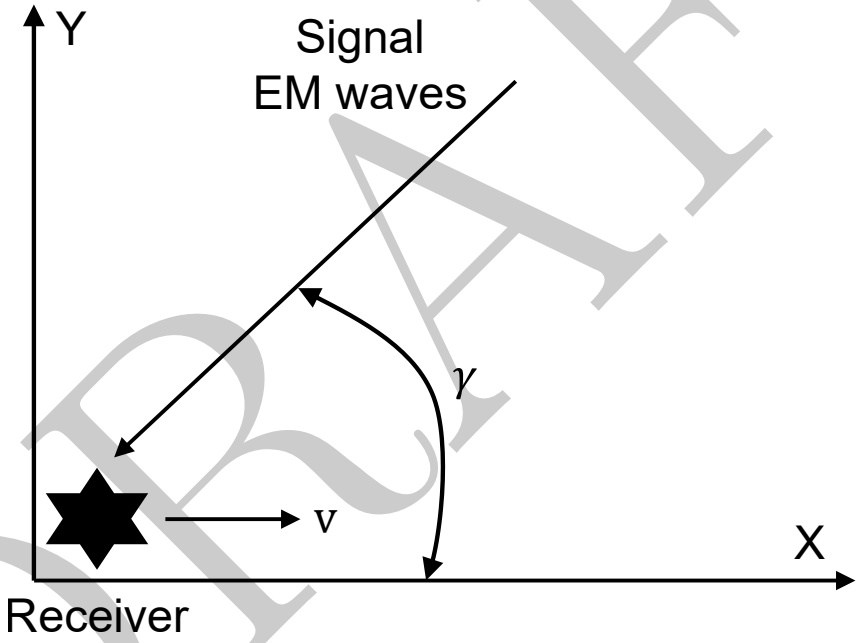


Figure 3.9: Diagram of the Doppler Effect in Two-dimensional

In this case, the speed of movement has changed to the projection of the speed of movement, which is $v\cos(\gamma)$. The revised Doppler shift is then:

$$\nu = \frac{v}{\lambda}\cos(\gamma) \tag{3.6}$$

3.2 Verification

3.2.1 Software Verification

To verify the correctness of the results, this project will use open-source code *SoftGNSS* for Matlab simulation. Also, the results of the simulation will help me to debug the code better.

3.2.2 Hardware Verification

Verification is an important step in FPGA development that saves time by quickly identifying faults in the code. It also prevents faults from being carried over to the next stage of the chip such as ASICs, which saves a lot of cost [32]. There are three main verification methods for developing FPGAs: behavioural simulation, post-synthesis functional simulation and post-implementation functional simulation[33]. Due to the time constraints of the project, only behavioural simulation was used for this project.

- Behavioural Simulation:

  RTL-level simulation allows us to simulate and verify your design before any transformations are performed by synthesis or implementation tools. We can verify your design as a module, block, or system.

  Typically, RTL simulation is performed to verify code syntax and confirm that the code functions as expected. In this step, the design is described primarily in RTL, so no timing information is required.

- Post-Synthesis Functional Simulation:

  It allows simulation of synthesized netlists to verify that the synthesized design meets functional requirements and behaves as expected.

- Post-Implementation Functional Simulation:

  We can perform either functional simulation or timing simulation after implementation. Timing simulation is the closest simulation to actually downloading the design into the device. It allows you to ensure that the implemented design meets

the functional and timing requirements and that the behaviour within the device matches the desired behaviour.

Chapter 4

Implementation

The previous section focused on the theoretical approach to designing a tracking module for a receiver, and this chapter will explain the specific practical process and the problems encountered during the process. This project mainly implements the tracking module on a GPS receiver, please note that this is only a separate module and does not realise the full receiver functionality. It also has RF front-end and acquisition modules in the front module, and ranging modules as well as navigation modules in the back module. The project in fact has a hypothetical target device, but based on the current conditions, the main use of VHDL for code writing, as well as behavioural simulation to verify the results, will not be on-board operations.

4.1 System Architecture

4.1.1 RF Front-end

The RF front-end is a device that collects any signals you expect. In this project, we will use *NT1065_FMC2* as our front-end. This device is designed to receive GPS, GLONASS, Galileo, BeiDou, IRNSS, QZSS and L1, L2, L3, L5, E1, E5a, E5b, E6, B1, B2, B3 bands. It has an FMC(*FPGA Mezzanine Card*) interface, which allows it to have a faster transfer rate to the Xilinx board [34]. Figure 4.1 shows the device's appearance. Here is its key specification table.

Table 4.1: Key Specification of *NT1065_FMC2*

Content	Specifications
Chip	NT1065
Number of channels	4
Reference frequency sources (MHz)	TCXO 10 TCXO 24.84
Bit width of ADC (bits)	2

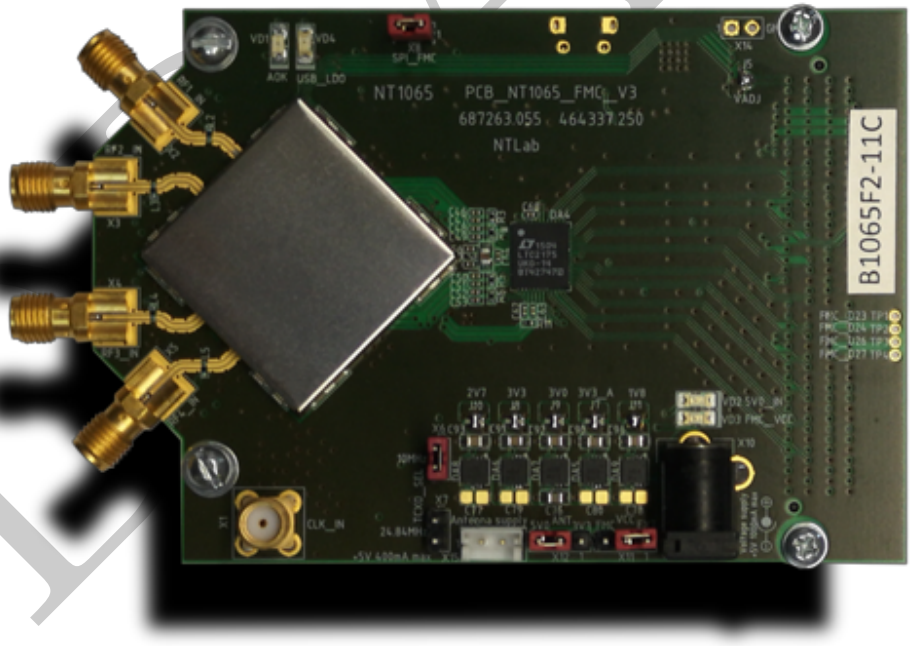


Figure 4.1: *NT1065_FMC2*

4.1.2 FPGA Board

The FPGA board is the core module of this project. It will process the acquisition, tracking and navigation stages. The board we choose is *KCU105 Evaluation Board* from *AMD Xilinx*. It's very powerful. Its specification and appearance will be shown below.

Table 4.2: Key Specification of *KCU105 Evaluation Board*

Content	Specifications
Chip	XCKU040-2FFVA1156E FPGA
System Logic Cells (K)	530
DSP Slices	1,920
Block RAM (Mb)	21.1
16.3Gb/s Transceivers	20
I/O Pins	520
Memory	2GB DDR4 component memory 64MB flash 8Kb EEPROM

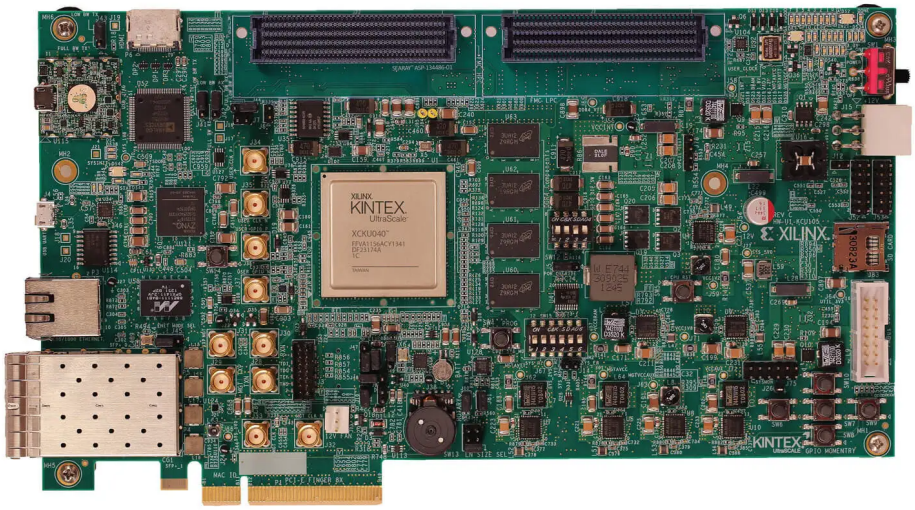


Figure 4.2: *KCU105 Evaluation Board*

4.2 Develop Environment

4.2.1 Vivado

As we are planning to use FPGA from *Xilinx*, we are supposed to code in *Vivado*. It is a powerful and versatile integrated development environment (IDE) created by Xilinx for FPGA and SoC development. It offers a comprehensive suite of tools and features for designing, implementing, and programming FPGAs and SoCs, enabling engineers to create custom digital hardware solutions. With its user-friendly interface and advanced design automation capabilities, *Vivado* simplifies the process of hardware design, synthesis, verification, and debugging. It supports a wide range of Xilinx devices, making it a go-to tool for hardware engineers and developers working on cutting-edge applications in industries like telecommunications, aerospace, and embedded systems. The version we use is **2018.3**. The reason for that is that it is stable but not outdated.

4.2.2 ModelSim

ModelSim is used to verify the design. We will run the simulation and check all the waveforms of the key signal. *ModelSim* is a leading digital simulation and verification tool by Mentor Graphics, now a part of Siemens. It's widely used for designing and testing digital systems, enabling engineers to simulate hardware description languages like VHDL and Verilog. *ModelSim* assists in debugging, verification, and validation of complex digital designs in various industries.

ModelSim is particularly compatible with Vivado, and when used in conjunction with it, you get twice the result with half the effort, and it can quickly locate code errors.

4.2.3 MATLAB

Matlab is a high-level programming environment and language renowned for its versatility in numerical computing, data analysis, and algorithm development. Used across various fields, from engineering to finance, it offers extensive libraries and tools for modelling, simulation, and solving complex mathematical problems, making it indispensable in re-

search and industry. The reason for using it is to run the *SoftGNSS* to verify my design as well.

4.3 Signal Analyse and Pre-process

At the beginning of the project, the only data known to us were the input signals. Therefore it is very crucial to analyse and pre-process the input signals. I will be using MATLAB to analyse the data and the figure below shows the results of my analysis.

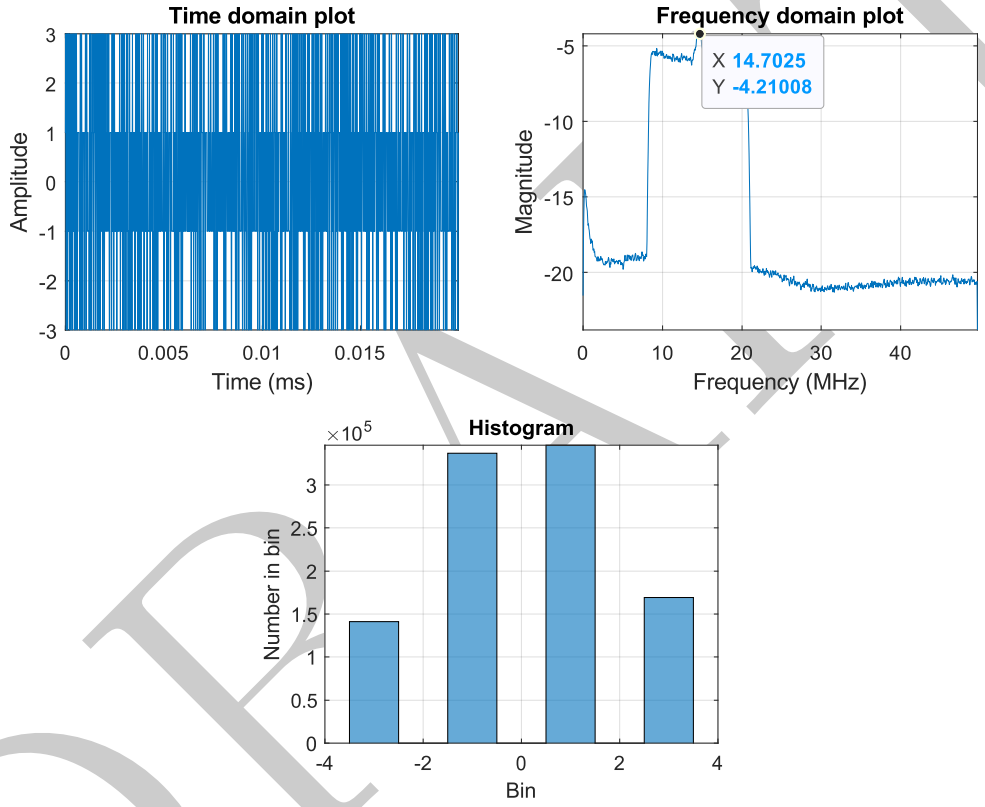


Figure 4.3: Signal Analyse

From the frequency domain in figure 4.3, we can see the IF is about 14.7025MHz. This is in line with our expectations. Also a perfect spectrum of signals can be seen, proving that I'm picking up the correct GPS signals. We should take note of the histogram, which provides us with a lot of basis. We can calculate the noise floor based on it, and we can also verify that we are reading the right data on the testbench based on its distribution. The following table 4.3 shows the distribution information of the input signals.

The following table shows the mean square at different input signal levels.

Table 4.3: Distribution Table of the Input Signal

Level	Counts	Percentage (%)
-3	141,278	14.22
-1	337,040	33.92
1	346,219	34.84
3	169,213	17.03

Table 4.4: Mean Square Accumulation Values

Level	Mean square
-3	22.5
-1	2.5
1	2.5
3	22.5

Combining the mean square accumulation values from table 4.4, with the distribution of the input signal from table 4.3, results in the following theoretical noise floor value for one accumulation period T i.e.,1ms.

$$NF = 2\times((14.22\%+17.03\%)\times22.5+(33.92\%+34.84\%)\times2.5)\times f_s\times T \approx= 1{,}739{,}122 \quad (4.1)$$

Where, NF is the noise floor value, $f_s = 99.375MHz$ and, $T = 1ms = 1\times10^{-3}s$. That is, this system has a floor noise of 1,739,122.

In addition, the basic information will be shown in table 4.5.

Table 4.5: Basic Information of the Signal

Specifications	Values
IF (MHz)	14.58
Sampling frequency (MHz)	99.375
Code frequency (MHz)	1.023
Data type	int 8
Samples/code	99,375
Accumulate time (ms)	1

4.4 Tracking

Before tracking, we need the acquisition result. The part has been done on the *SoftGNSS*. Here are the results.

Table 4.6: Results of the Acquisition

Specifications	Values
Chopped bits	46,250
PRN (№)	2
IF (MHz)	14.5778
Doppler (Hz)	2200
Code offset (Chips)	412

These results are fed directly into the tracking stage. And to ensure that I got the correct input data from the front-end. I will save the input data from the testbench and plot the histogram of it like the figure 4.3. The histogram is shown in figure 4.4.

Obviously the input signal is correctly distributed across the values, and I'm getting the correct input signal. This is an important way to check if the results are correct.

4.4.1 NCO

The NCO block is driven by a 99.375MHz signal. Firstly, we need to obtain the FSW. The bit width of the PA is 32 bits and the search frequency is 500Hz. The required carrier

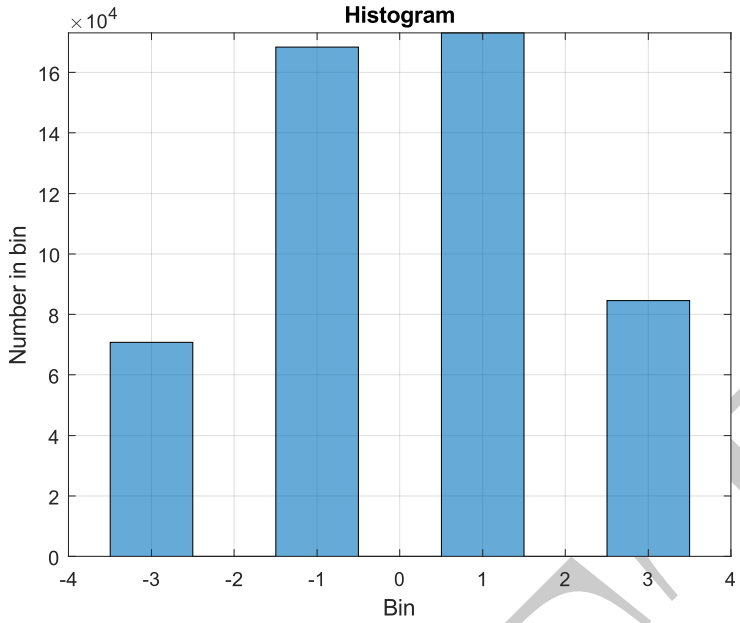


Figure 4.4: Histogram of Input Data from Testbench

frequency is $IF + Doppler$, which is 14.5822MHz.

However, the code frequency is a bit different. To obtain that, we will first introduce the search frequency. The search frequency allows us to let the local code block slide to determine to inner product. We are expecting to have half a chip more per millisecond, i.e., 0.5 chips/ms. Therefore, the search frequency is 500Hz. Here is the diagram of search frequency.

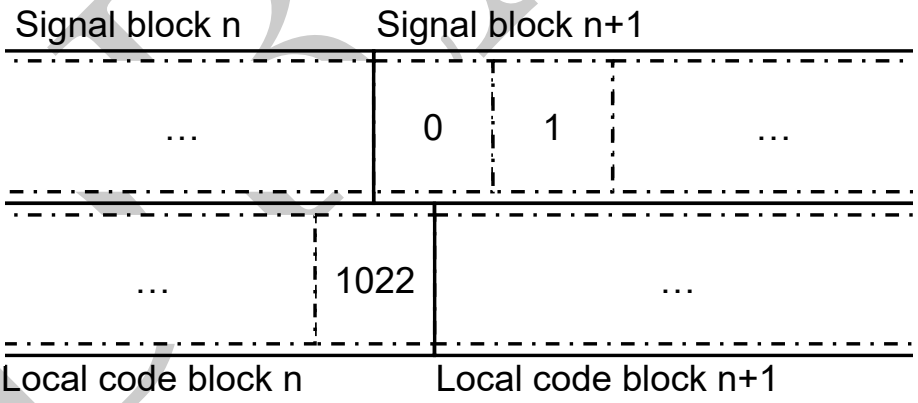


Figure 4.5: Diagram of Search Frequency

The required code frequency is $Code\ frequency - Search\ frequency = 1.0225Mhz$ then. Using the equation 3.1, we can determine the proper increment value or FSW.

$$(increment\ value) = \frac{Requied\ frequency}{Sampling\ frequency} \times 2^{PA\ bit\ width} \tag{4.2}$$

In order to be able to track 10.23MHz C/A codes in the future, the 10.23MHz signal will first be generated using the NCO and a 1/10 divider will be used to obtain a 1.023MHz signal. Therefore, the FSWs for carrier and code are 630,239,720 and 441,922,422.

As for the design of VHDL, essentially, the PA is a counter, except that each clock does not increment by 1 but by a specified value based on FSW. It is also possible to set the overflow value. After the overflow value is reached, the counter will be set to 0.

Since we only need two-bit width carrier signals, the lookup table for the NCO will be very simple, as shown in the table below.

Table 4.7: LUT of NCO for Generating Carrier

Phase (First 3 bits)	Amplitude	
	Cosine	Sine
000	2	-1
001	2	1
010	1	2
011	-1	2
100	-2	1
101	-2	-1
110	-1	-2
111	1	-2

Simulation and Waveform

In this subsection, the NCO-related code is simulated and all signals generated by the NCO are shown.

The following figure shows the PA waveform when the NCO generates the C/A code clock signal. As mentioned before, a $10.23MHz - 5{,}000Hz$ signal is generated first, and then a 1/10 divider is used to get $1.023 - 500Hz$. From the figure, it can be seen that after ten overflows, the C/A code is clocked at 1.024MHz, which is within the acceptable range, although there are errors.

Figure 4.7 shows that using the NCO to generate the carrier signals containing cosine

and sine signals which are used to get the I/Q phase of the input signal.

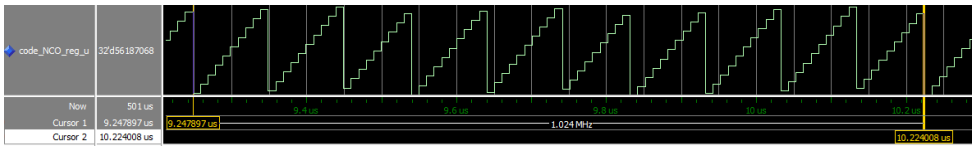


Figure 4.6: Waveform of NCO for Generating C/A Code

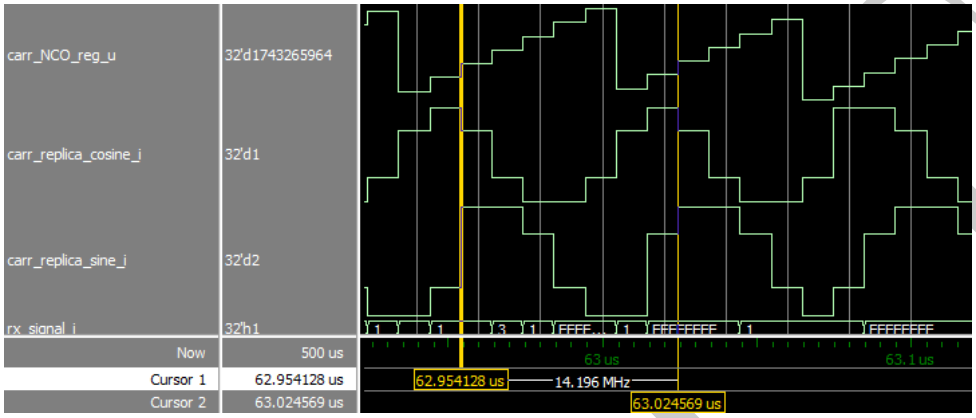
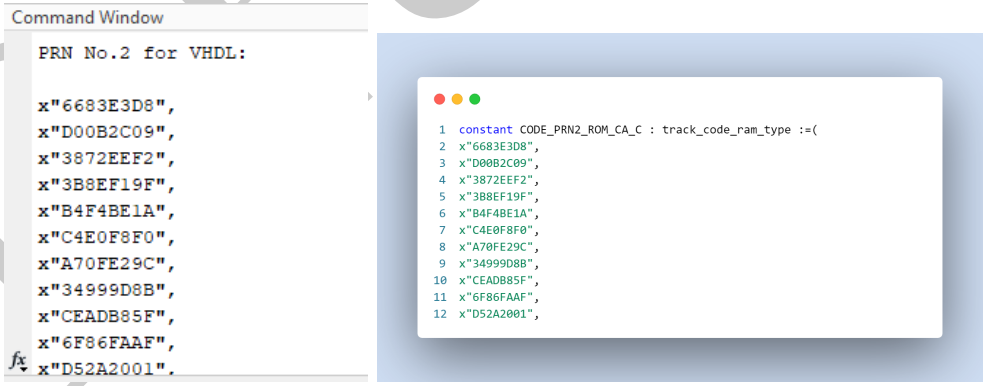


Figure 4.7: Waveform of NCO for Generating Carrier

4.4.2 Code Generator

In order to save computational resources on board, I calculated the required PRN code, i.e. PRN №2 code, in advance. It is generated through Matlab and stored in VHDL code.



(a) MATLAB (b) VHDL

Figure 4.8: Part of the Code of PRN №2

The above figures show the PRN №2 code generated by MATLAB and the values saved in the VHDL code, which have been saved in hexadecimal, respectively.

It should also be noted that a normal PRN code should consist of 0 and 1, whereas here the 0 and 1 in the PRN code are interchanged in order to facilitate the subsequent correlator to complete the inner product calculation.

4.4.3 Correlator

The design of the correlator is at the heart of this project. The multiplier and accumulator are again at the heart of the correlator.

Our design requirement is to find the inner product of the local code and the input signal over a period of 1 ms.

- Firstly a timer of 1 ms is required, here we use a counter to determine the elapsed time by recording the number of PRN codes. When the number of PRN codes reaches 1023, then one millisecond has been reached.

- At the beginning of a 1 ms slot, the inner product of each bit and the current C/A code bit is stored in a register. The result of the inner product of each bit is continually accumulated until 1 ms is reached.

- After every 1 millisecond end, the value of the accumulation register is saved and set to zero before the next millisecond is calculated.

Note that I don't know whether the energy is mainly concentrated in phase I or phase Q, so each product operation needs to be processed separately for I/Q phase. Assuming we simulate for 60ms, we will get 60 results. Plotting these sixty results in Matlab gives results similar to figure 3.6. However, to ensure that we are tracking the correct code delay, we need three correlators that use the early, prompt, and late code to determine their results after correlation.

Early, Prompt, and Late Code Generator

In order to generate the early, prompt, and late code, I design a register *code_subcarr_delay_reg_u*. It is 98 bits wide and filled with PRN №2 code. I take the

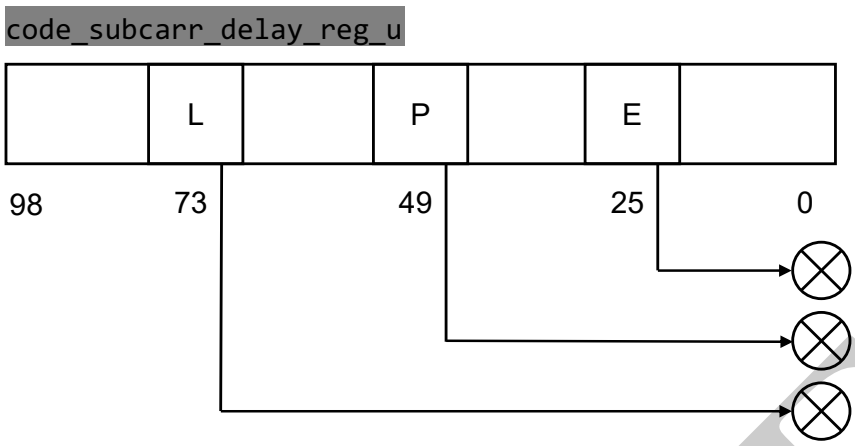


Figure 4.9: Diagram of *code_subcarr_delay_reg_u* Register

25th bit for the early code, the 49th bit for the prompt code, and the 73rd bit for the late code. It is shown in figure 4.9.

Multiplier Optimization

As mentioned earlier, finding the inner product is the core of the correlator, so a lot of multipliers will be used. However, compilers like Vivado will call on the on-board DSP resources to replace the multipliers during synthesis, which I don't want to see because on-board DSP resources are scarce [35, 36], and they should be used elsewhere. So I can optimize the multiplier for the current project.

```
1  -- prompt mixing
2  if (replica_code = '1') then
3      post_carr_code_mix_P_I_i <= post_carr_mix_I_i;
4      post_carr_code_mix_P_Q_i <= post_carr_mix_Q_i;
5  else
6      post_carr_code_mix_P_I_i <= -post_carr_mix_I_i;
7      post_carr_code_mix_P_Q_i <= -post_carr_mix_Q_i;
8  end if;
```

Figure 4.10: Part of the Code of Multiplier

Since each time on the inner product is to let the current signal bit multiplied with the current C/A code bit, and the C/A code is only used in two cases are 0 and 1. When the C/A code bit is 1, the result of multiplying any number with 1 is unchanged, and

when the C/A code bit is 0, I need to convert it to -1 first, and the result of multiplying any number with -1 is its opposite. In this case, rewriting the code saves a lot of on-board resources. The optimized code is shown in figure 4.10.

Simulation and Waveform

Figure 4.11 below is a waveform of the signals mentioned above, I will explain in detail the significance of each signal and how it relates to the theory.

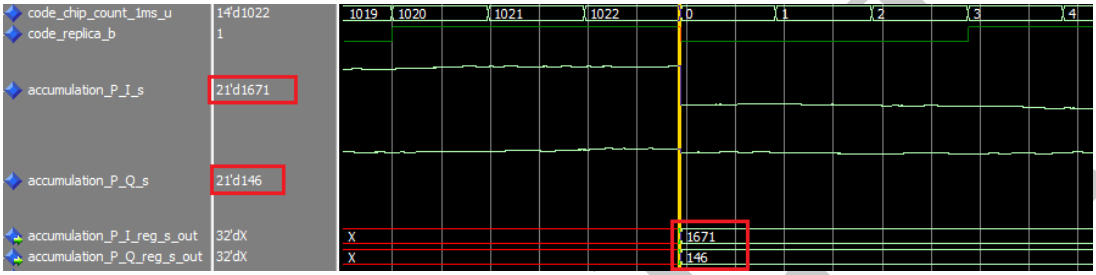


Figure 4.11: Waveform of Key Signals in Correlator

- *code_chip_count_1ms_u*: This signal is used to indicate 1 millisecond of time. It is essentially a counter that instructs the other correlation accumulation registers to store the current accumulation value by reaching a counter value of 1022, i.e. 1 milliseconds
- *code_replica_b*: This signal is the waveform of the PRN №2 code output by bit.
- *accumulation_P_I_s*: This is the accumulation result of the I-phase prompt code, and you can see that the value has ups and downs. The result of this millisecond accumulation is 1,671.
- *accumulation_P_Q_s*: This is the accumulation result of the Q-phase prompt code, and result of this millisecond accumulation is 146.
- *accumulation_P_I_reg_s_out*: This signal saves the result of this millisecond accumulation of the I-phase prompt code.
- *accumulation_P_Q_reg_s_out*: This signal saves the result of this millisecond accumulation of the Q-phase prompt code.

Chapter 5

Results

In this section, I will show my results, i.e., the results of the correlator. It consists of a sequence of data, and when I simulate for 10 milliseconds, it will have ten sets of data.

5.1 Theoretical Results

Before this, I can theoretically analyse the correlation after the peaks

We know that the sampling frequency is $99.375MHz$, so the number of samples per millisecond is $99.375kSamples$. Therefore, each code slice contains,

$$SamplesPerChip = \frac{SamplesPerms}{1023} \approx 97.14\ Samples \tag{5.1}$$

According to table 4.6, it can be seen that the C/A offset number is 412, so the peak should occur around $412/SamplesPerChip = 4.24Chips$. As each chip takes 2ms, the peak should occur around $4.24 \times 2 = 8.48ms$.

5.2 Results and Waveform

Simulate in Modelsim for 50 ms and save the *accumulation_P_I_reg_s_out* and *accumulation_P_Q_reg_s_out* values and input into MATLAB for analysis.

In figure 5.1, we can see the values of two registers that store the values associated with the I/Q phases, respectively, and at the bottom is the sum of their squares. We

can see that the peak reaches its maximum at the eighth value, i.e. at the eighth second, which agrees with my theoretical estimate. However, it is not a significant enough peak, nor is it large enough, so I suspect that the correlator's performance is not yet up to standard.

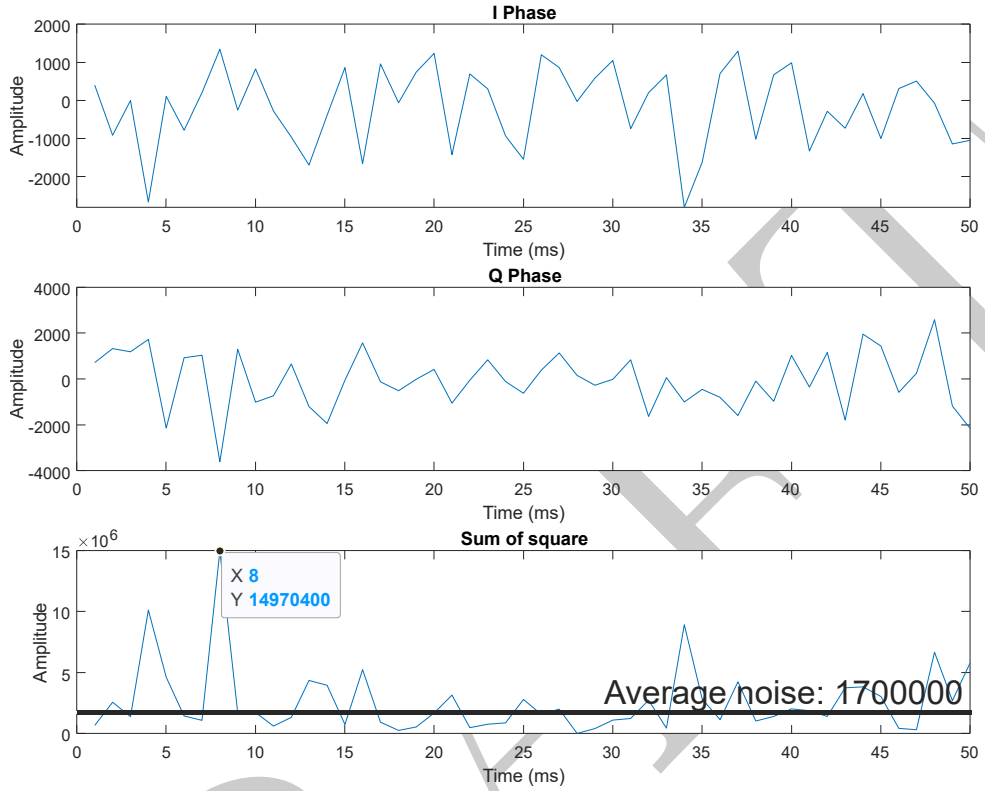


Figure 5.1: Result of Correlator for 50ms

Chapter 6

Summary and Reflections

As the results came in, the project came to a close. There is a lot of work to summarize and reflect on as well as improve. I will describe two aspects below.

6.1 Summary and Reflections

For the thesis: the first chapter focuses on the background as well as the objectives of the project. The second chapter introduces the related work, starting with the signal and analysing the GPS/GNSS signal characteristics, followed by reviewing the previous books related to GPS tracking, and finally introduces the FPGA technology and explains the reason for using FPGA. Chapter 3 analyses theoretically how to perform GPS signal tracking, various modules such as NCO, code generator etc. are studied and analysed. Finally, the verification methodology is explained to ensure that the results can be evaluated objectively. Chapter 4 is a concrete implementation, where much of the theory needs to be engineered due to the gap between theory and practice. Firstly, the equipment used, and the development environment are described. Next the input signals are analysed. Finally, each module was developed, and the core algorithms in it were introduced. In the last chapter, the results are analysed, and the results are predicted from the theory first, and then compared with the actual results, which are basically in line with the expectations.

For the project itself, I think it is somewhat difficult. Especially in terms of code debugging. I need to analyse the errors from the waveforms and this is something I need

to improve. I also realized the difference between theory and practice, and that completing the code writing does not necessarily complete the function in question.

6.2 Future Work

In section 5.2, I mentioned that the correlator's performance is not up to standard. I think the following aspects can be improved:

- NCO accuracy needs to be improved. When reviewing the simulation waveforms, I found that the frequency generated by the NCO is not accurate, I suspect that it is a precision issue, and subsequently the NCO module can be redesigned to improve the precision.
- For the problem that the peak value is not obvious enough. I suspect that the correlator design is unreasonable. Subsequently, I plan to export the middle data to *SoftGNSS* for testing, and compare each step of the correlator to determine where the problem lies.
- More objective evaluation methods are needed. However, there is currently no such method, and the commonly used C/No plot, which is the ratio of carrier signal energy to noise energy over a 1 Hz bandwidth, which is related to multipath effects, receiver antenna gain, attenuation of the antenna cable, the level of the satellite signal emission, and tropospheric delays, can be used, and is usually set to a value of 45 dB-Hz [37].

Bibliography

[1] U. Hugentobler and O. Montenbruck, *Satellite Orbits and Attitude*. Cham: Springer International Publishing, 2017, pp. 59–90. [Online]. Available: https://doi.org/10.1007/978-3-319-42928-1_3

[2] R. B. Langley, P. J. G. Teunissen, and O. Montenbruck, *Introduction to GNSS*. Cham: Springer International Publishing, 2017, pp. 3–23. [Online]. Available: https://doi.org/10.1007/978-3-319-42928-1_1

[3] E. D. Kaplan and C. Hegarty, *Understanding GPS/GNSS: Principles and Applications, Third Edition*, 3rd ed. Place of publication not identified: Place of publication not identified: Artech House, 2017.

[4] J. Parker, "New frontiers in space use of gnss: Moon and beyond," 12th Sep 2019.

[5] U.S. Space Force, "Gps interface specification is-gps-200, revision n," 22, Aug 2022.

[6] ——, "Gps interface specification is-gps-705, revision j," 22, Aug 2022.

[7] ——, "Gps interface specification is-gps-800, revision j," 22, Aug 2022.

[8] Roscosmos, "Interface control document edition 5.1," 2008.

[9] ESA EUSPA, "Galileo - open service - signal in space interface control document v2.0," January 2021.

[10] China Satellite Navigation Office, "Beidou navigation satellite system open service performance standard (version 3.0)," May 2021.

[11] T. Mai, "Global positioning system history," 10 Aug, 2023 2012. [Online]. Available: https://www.nasa.gov/directorates/heo/scan/communications/policy/GPS_History.html

[12] P. D. Groves, *Principles of Gnss, Inertial, and Multisensor Integrated Navigation Systems*. Boston: Boston: Artech House, 2007, available also in a print ed. Mode of access: Internet via World Wide Web. Title from title screen.

[13] J. C. Juang, Y. H. Chen, T. L. Kao, and Y. F. Tsai, "Design and implementation of an adaptive code discriminator in a dsp/fpga-based galileo receiver," *GPS solutions*, vol. 14, no. 3, pp. 255–266, 2010.

[14] O. Jakubov, P. Kovar, P. Kacmarik, and F. Vejrazka, "The witch navigator - a low cost gnss software receiver for advanced processing techniques," *Radioengineering*, vol. 19, no. 4, pp. 536–543, 2010.

[15] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of fpgas," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, 2007.

[16] V. Agarwal, H. Arya, and S. Bhaktavatsala, "Design and development of a real-time dsp and fpga-based integrated gps-ins system for compact and low power applications," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 45, no. 2, pp. 443–454, 2009.

[17] S. Gleason and D. Gebre-Egziabher, *GNSS Applications and Methods*. Norwood: Norwood: Artech House, 2009, available also in a print ed. Mode of access: Internet via World Wide Web. Title from title screen.

[18] S. Kadam, D. Sasidaran, A. Awawdeh, L. Johnson, and M. Soderstrand, "Comparison of various numerically controlled oscillators," in *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, vol. 3, Conference Proceedings, pp. III–III.

[19] N. A. Ranabhatt, S. Agarwal, R. K. Bhattar, and P. P. Gandhi, "Design and implementation of numerical controlled oscillator on fpga," in *2013 Tenth International Conference on Wireless and Optical Communications Networks (WOCN)*, Conference Proceedings, pp. 1–4.

[20] G. Ramana Reddy, C. Perumal, P. Kodali, and V. R. Bodapati, "Design and analysis of dual-mode numerically controlled oscillators based controlled oscillator frequency modulation," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 5, pp. 4935–4943, 2022.

[21] G. D. Ghiwala, P. P. Thaker, and G. D. Amin, "Realization of fpga based numerically controlled oscillator," *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*, vol. 1, no. 5, pp. 7–11, 2013.

[22] K. Suganthi and A. Abinaya, "Design and implementation of numerically controlled oscillator," in *2019 International Conference on Computer Communication and Informatics (ICCCI)*, Conference Proceedings, pp. 1–4.

[23] D. Hummels, "Numerically controlled oscillators," 2019.

[24] D. Gisselquist, "Building a numerically controlled oscillator," 2017. [Online]. Available: https://zipcpu.com/dsp/2017/12/09/nco.html

[25] C. J. Hegarty, "Gnss signals — an overview," in *2012 IEEE International Frequency Control Symposium Proceedings*, Conference Proceedings, pp. 1–7.

[26] D. Dharmappa, A. Gupta, F. B. Singh, T. S. Ganesh, and M. R. Raghavendra, "New lfsr based methodology to generate prn codes with lengths suitable for gnss having low correlation," in *2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, Conference Proceedings, pp. 1–4.

[27] D. Manandhar, "Introduction to global navigation satellite system (gnss) signal structure," 2018. [Online]. Available: https://home.csis.u-tokyo.ac.jp/~dinesh/Dinesh_T_files/GNSS_06_Introduction_SingalStructures.pdf

[28] O. K. Mikhaylova, I. V. Korogodin, and I. V. Lipa, "Universal ranging code generator of gnss signals," in *2020 European Navigation Conference (ENC)*, Conference Proceedings, pp. 1–10.

[29] B. Sklar, *Digital communications : fundamentals and applications / Bernard Sklar, fred harris*, third edition. ed. London : Pearson, 2021, (Fredric J.).

[30] I. Glover, *Digital communications / Ian A. Glover, Peter M. Grant*, 3rd ed. Harlow: Harlow : Prentice Hall, 2010, includes bibliographical references and index.

[31] A. F. Molisch, *Wireless communications / Andreas F. Molisch*, 2nd ed. Chichester, U.K. : IEEE, 2011, includes bibliographical references and index.

[32] U. Farooq and H. Mehrez, "Pre-silicon verification using multi-fpga platforms: A review," *Journal of Electronic Testing*, vol. 37, no. 1, pp. 7–24, 2021. [Online]. Available: https://doi.org/10.1007/s10836-021-05929-1

[33] Vivado, *Vivado Design Suite User Guide: Logic Simulation*, ug900 (v2023.1) ed., 2023.

[34] NTLab, *NT1065_FMC2: Evaluation kit user manual*, November 2018. [Online]. Available: www.ntlab.com

[35] F. d. Dinechin and B. Pasca, "Large multipliers with fewer dsp blocks," in *2009 International Conference on Field Programmable Logic and Applications*, Conference Proceedings, pp. 250–255.

[36] M. Kumm, J. Kappauf, M. Istoan, and P. Zipf, "Resource optimal design of large multipliers for fpgas," in *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*, Conference Proceedings, pp. 131–138.

[37] D. Wujiao, D. Xiaoli, and Z. Jianjun, "Comparing gps stochastic models based on observation quality indices," *Geomatics and Information Science of Wuhan University*, vol. 33, no. 7, p. 718, 2008. [Online]. Available: http://ch.whu.edu.cn/en/article/id/1646