VIETNAM NATIONAL UNIVERISTY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY (HCMUT)
FACULTY OF COMPUTER SCIENCE & TECHNOLOGY

GRADUATION THESIS

GPS RECEIVER ON FPGA

MAJOR: COMPUTER ENGINEERING
COMMITTEE: 1

Supervisors:

Assoc. Prof. Dr. Pham Quoc Cuong - HCMUT

Reviewer:

Assoc. Prof. Dr. Tran Ngoc Thinh - HCMUT

Authors:

Vu Nam Binh - 2152441

Nguyen Minh Khoa - 2152131

Tran Minh Trung - 2153073

HO CHI MINH CITY - May-2025

HĐ1 CLC KTMT – 157

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

KHOA:KH & KT MÁY TÍNH
BỘ MÔN: KỸ THUẬT MÁY TÍNH

NHIỆM VỤ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP

Chú ý: Sinh viên phải dán tờ này vào trang nhất của bản thuyết trình

HỌ VÀ TÊN: Nguyễn Minh Khoa — MSSV: 2152131
HỌ VÀ TÊN: Trần Minh Trung — MSSV: 2153073
HỌ VÀ TÊN: Vũ Nam Bình — MSSV: 2152441
NGÀNH: Kỹ thuật Máy tính — LỚP: MT21KTM

1. Đầu đề luận văn/ đồ án tốt nghiệp:
Bộ thu tín hiệu GPS trên FPGA *(GPS Receiver on FPGA)*

2. Nhiệm vụ (yêu cầu về nội dung và số liệu ban đầu):

- Study the GPS Signal Specification
- Study about implementing Digital Signal Processing components on FPGA (Filter, Fast Fourier Transform, Numerically Controlled Oscillator, ...)
- Propose the architecture of the GPS radio receiver
- Implement the proposed architecture on the Ultrascale 96 v2 FPGA board
- Implement the proposed architecture on the Ultrascale 96 v2 FPGA board
- Conduct testing and comparing with existing work

3. Ngày giao nhiệm vụ: 06/01/2025

4. Ngày hoàn thành nhiệm vụ: 22/5/2025

5. Họ tên giảng viên hướng dẫn: **Phần hướng dẫn:**

1) Phạm Quốc Cường

Nội dung và yêu cầu LVTN/ ĐATN đã được thông qua Bộ môn.

Ngày 06 háng 01 năm 2025

CHỦ NHIỆM BỘ MÔN
(Ký và ghi rõ họ tên)

Phạm Quốc Cường

GIẢNG VIÊN HƯỚNG DẪN CHÍNH
(Ký và ghi rõ họ tên)

Phạm Quốc Cường

PHẦN DÀNH CHO KHOA, BỘ MÔN:
Người duyệt (chấm sơ bộ):
Đơn vị:
Ngày bảo vệ:
Điểm tổng kết:
Nơi lưu trữ LVTN/ĐATN:

HĐ1 CLC KTMT - 157

TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KH & KT MÁY TÍNH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Ngày 15 tháng 5 năm 2025

PHIẾU ĐÁNH GIÁ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP

(Dành cho người hướng dẫn)

1. HỌ VÀ TÊN: Nguyễn Minh Khoa — MSSV: 2152131
Trần Minh Trung — MSSV: 2153073
Vũ Nam Bình — MSSV: 2152441
NGÀNH: Kỹ thuật Máy tính — LỚP: MT21KTM

2. Đề tài:
Bộ thu tín hiệu GPS trên FPGA *(GPS Receiver on FPGA)*

3. Họ tên người hướng dẫn: Phạm Quốc Cường

4. Tổng quát về bản thuyết minh:
Số trang: — Số chương:
Số bảng số liệu — Số hình vẽ:
Số tài liệu tham khảo: — Phần mềm tính toán:
Hiện vật (sản phẩm)

5. Những ưu điểm chính của LV/ ĐATN:
- Sinh viên hoàn thành tốt nhiệm vụ đặt ra
- Hệ thống hoạt động ổn định
- Sinh viên làm việc nghiêm túc, báo cáo đạt yêu cầu
- Kết quả đạt được mặc dù chưa thật sự ấn tượng nhưng chấp nhận được đối với ĐATN của SV dưới điều kiện các trang thiết bị chưa được tốt như các công bố khoa học

6. Những thiếu sót chính của LV/ĐATN:
- Hệ thống thử nghiệm cho kết quả chưa thật sự cao về hiệu suất tính toán và thời gian thực thi.

7. Đề nghị: Được bảo vệ ☒ Bổ sung thêm để bảo vệ ☐ Không được bảo vệ ☐
8. Các câu hỏi SV phải trả lời trước Hội đồng:
a. Đâu là rào cản cho việc gia tăng độ chính xác và hiệu suất của hệ thống?

9. Đánh giá chung (bằng chữ: Xuất sắc, Giỏi, Khá, TB): Điểm: 9.0/10

Ký tên (ghi rõ họ tên)

Phạm Quốc Cường

HD 1CLC KTMT - 157

TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KH & KT MÁY TÍNH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Ngày tháng năm

PHIẾU ĐÁNH GIÁ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP

(Dành cho người ~~hướng dẫn~~/phản biện)

1. Họ và tên SV:

Vu Nam Binh	MSSV: 2152441	Ngành (chuyên ngành):Kỹ thuật Máy tính
Nguyen Minh Khoa	MSSV: 2152131	Ngành (chuyên ngành):Kỹ thuật Máy tính
Tran Minh Trung	MSSV: 2153073	Ngành (chuyên ngành):Kỹ thuật Máy tính

2. Đề tài: **GPS RECEIVER ON FPGA (Bộ thu tín hiệu GPS trên FPGA)**

3. Họ tên người ~~hướng dẫn~~/phản biện: PGS.TS. Trần Ngọc Thịnh

4. Tổng quát về bản thuyết minh:

Số trang: 114 Số chương: 7
Số bảng số liệu: 4 Số hình vẽ: 77
Số tài liệu tham khảo: 27 Phần mềm tính toán:
Hiện vật (sản phẩm)

5. Những ưu điểm chính của LV/ ĐATN:

The thesis focuses on the design and FPGA implementation of a GPS receiver system using the Ultra96-V2 platform. It provides a thorough background on GPS principles, including signal structure, acquisition, and tracking phases. A specific architecture for GPS acquisition using FFT/IFFT (PCS method) is proposed and implemented. The tracking process includes PLL for carrier phase tracking and DLL for code phase tracking. Functional verification is performed using MATLAB-based golden models. Synthesis and implementation results on the Ultra96-V2 board are provided, including resource utilization and timing. Performance evaluation includes acquisition time comparisons with other works. The project discusses two versions of FFT implementation (Scale and Non-scale) considering trade-offs like accuracy and resource utilization.

6. Những thiếu sót chính của LV/ĐATN:

The experiment results are too simple. There is a limited practical demonstration or integration with SDR systems; it lacks a detailed user interface and deployment testing.
The proposed system is unfairly compared with other implementations because those works are outdated, dating back more than 10-15 years, and the evaluation should be compared with newer works.

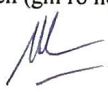
7. Đề nghị: Được bảo vệ ☑ Bổ sung thêm để bảo vệ ○ Không được bảo vệ ○

8. Các câu hỏi SV phải trả lời trước Hội đồng:

a. Can you elaborate on the trade-offs between the 'Scale' and 'Non-scale' FFT versions in terms of performance impact on the overall GPS signal acquisition and tracking accuracy, especially in weak signal conditions?
b. What were the primary challenges in implementing the CORDIC algorithm for carrier wipe-off, and how did its fixed-point implementation precision affect the acquisition results?

9. Đánh giá chung (bằng chữ: Xuất sắc, Giỏi, Khá, TB): *Very Good (giỏi)* Điểm : 9/10

Ký tên (ghi rõ họ tên)



Trần Ngọc Thịnh

This thesis is dedicated for our parents and our instructors at HCMUT.

CONTENTS

List of Figures **xi**

List of Tables **xv**

Acknowledgment **xvii**

Abstract **xix**

1 Introduction **1**
1.1 Introduction 1
1.2 Outline 2

2 Background and Related work **3**
2.1 Introduction to Global Positioning System (GPS) 3
2.1.1 Background and History 3
2.1.2 Basic Principles of Operation 4
2.2 GPS Acquisition Phase 15
2.2.1 Doppler Frequency 15
2.2.2 Code Phase Delay 17
2.2.3 FFT and IFFT application in Acquisition phase 18
2.2.4 GPS Acquisition 23
2.3 GPS Tracking Phase 29
2.3.1 Phase-Locked Loop 29
2.3.2 Delay-Locked Loop 31
2.3.3 Integrate and Dump 32
2.3.4 BPSK Demodulation 33
2.4 Hardware 33
2.4.1 Ultra96-V2 Overview 33
2.4.2 Comparison with Other FPGA Boards 34
2.4.3 Resource Utilization 34
2.4.4 Zynq MPSoC Processing System 34
2.5 Related Work 34
2.5.1 GPS Acquisition Phase 34
2.5.2 GPS Tracking Phase 42

3 Proposed Design Architecture **45**
3.1 Overall System 45
3.1.1 System Architecture 45
3.1.2 GPS Top Architecture 47
3.2 GPS Acquisition Phase 48
3.2.1 ACCUMULATOR and CORDIC blocks 48
3.2.2 FFT and IFFT blocks 49
3.2.3 COMPLEX MULTIPLIER blocks 49
3.2.4 BUFFER blocks 49
3.2.5 TOP 4 PEAK SELECTION block 50
3.3 GPS Tracking Phase 50

4 Design Implementation **53**
4.1 GPS Acquisition Phase 53
4.1.1 ACCUMULATOR and CORDIC blocks 53
4.1.2 FFT and IFFT blocks 57
4.1.3 COMPLEX MULTIPLIER block. 62
4.1.4 BUFFER blocks 63
4.1.5 TOP 4 PEAK SELECTION block 64
4.2 GPS Tracking Phase 69
4.2.1 Carrier Wipe-off 69
4.2.2 Correlation and Code Generators 71
4.2.3 Integrate and Dump 73
4.2.4 Phase Locked Loop and PI filter 74
4.2.5 Delay Locked Loop and PI filter 75
4.2.6 Navigation Message Extraction 77

5 Experimental Setup & Evaluation **79**
5.1 Input Generation 79
5.2 Functional Verification. 80
5.2.1 GPS Acquisition Phase 80
5.2.2 GPS Tracking Phase. 98
5.3 Synthesis & Implementation Results 99
5.3.1 Synthesis & Implementation. 99
5.3.2 Board run result. 101
5.4 Performance Evaluation 102

6 Future Work **105**
6.1 Software-defined Radio 105
6.2 GPS Acquisition 106
6.2.1 Developing non-coherent and coherent integration. 106

6.2.2 New proposed GPS acquisition architectures 106
6.3 Navigation Message Decoding 108
6.4 Adaptation to GPS L2C. 109

7 Conclusion **111**
Bibliography 112

LIST OF FIGURES

2.1 The Constellation Diagram of BPSK Modulation Used in GPS L1 . 6
2.2 Coarse/Acquisition Code Example [1] 6
2.3 Coarse/Acquisition Code Generation Flowchart 7
2.4 G2 Selector Table 1 [2] 8
2.5 G2 Selector Table 2 [2] 9
2.6 Coarse/Acquisition Code Generator Block Diagram 9
2.7 1 Satellite Results in Multiple Valid Points on The Circle 11
2.8 2 Satellite Circles Result in 2 Intersection Points 11
2.9 3 Satellite Circles Result in 1 Intersection Point 12
2.10 1 Satellite Sphere Results in Multiple Valid Positions 13
2.11 2 Satellite Spheres Intersection Results in a Circle 13
2.12 3 Satellite Spheres Result in 2 Intersection Points 14
2.13 Doppler effect example (adopted from [3]) 16
2.14 Butterfly behavior of 8-points FFT 21
2.15 Two-dimensional search space 25
2.16 Three-dimensional space (adapted from [4]) 25
2.17 Parallel Frequency Search [5] 27
2.18 Parallel Code Phase Search block diagram [5] 28
2.19 Early, late, prompt code replica tracking[6] 31
2.20 Sensitivity parameters [7] 37
2.21 Search space parameters (adpated from [7]) 37
2.22 Results and performance of three method implementations [7] . . 38
2.23 Shapiro GPS receiver implementation[8] 42

3.1 Proposed System Architecture 46
3.2 Detailed Block Diagram in Vivado 47
3.3 GPS Top Block Diagram 47
3.4 GPS acquisition design overview diagram 48
3.5 GPS tracker design overview diagram 50

4.1 Accumulator detailed block diagram 54
4.2 CORDIC detailed block diagram 56
4.3 Fast Fourier Transform overview block diagram 57

4.4 FFT stage detailed block diagram 58
4.5 FFT prelast stage and last stage detailed block diagram 59
4.6 Complex multiplier block diagram 63
4.7 Two smaller blocks inside Top 4 Peak Selection block 64
4.8 Detailed block diagram for Satellite Record block 65
4.9 Top 4 peak selection flow chart 66
4.10 Top 4 peak selection flow chart - Branch (1) 67
4.11 Top 4 peak selection flow chart - Branch (2) 67
4.12 Top 4 peak selection flow chart - Branch (3) 67
4.13 Top 4 peak selection flow chart - Branch (4) 68
4.14 Carrier wipe-off implementation 70
4.15 NCO implementation 71
4.16 NCO phase accumulator 71
4.17 Code generators implementation 72
4.18 Code phase accumulator 73
4.19 Dump timing generator 74
4.20 *early_acc_i_reg* Integrate and Dump diagram 74
4.21 PLL discriminator calculation 75
4.22 Carrier NCO phase increment calculation 75
4.23 First in-phase early accumulator power pipeline stage 76
4.24 Second early accumulator power pipeline stage 76
4.25 DLL discriminator calculation 76
4.26 Code NCO phase increment calculation 77
4.27 Navigation message extraction 78

5.1 FFT design functional verification result - Version: NON_SCALE . 81
5.2 FFT design functional verification result - Version: NON_SCALE (cont) 82
5.3 FFT design functional verification result - Version: NON_SCALE (cont) 82
5.4 FFT design functional verification result - Version: NON_SCALE . 82
5.5 FFT design functional verification result - Version: NON_SCALE (cont) 83
5.6 FFT design functional verification result - Version: NON_SCALE (cont) 83
5.7 FFT design functional verification result - Version: SCALE 84
5.8 FFT design functional verification result - Version: SCALE (cont) . 84
5.9 FFT design functional verification result - Version: SCALE (cont) . 84
5.10 FFT design functional verification result - Version: SCALE 85
5.11 FFT design functional verification result - Version: SCALE (cont) . 85

5.12 FFT design functional verification result - Version: SCALE (cont) . 85
5.13 IFFT design functional verification result 86
5.14 IFFT design functional verification result (cont) 86
5.15 IFFT design functional verification result (cont) 87
5.16 IFFT design functional verification result (cont) 87
5.17 IFFT design functional verification result (cont) 87
5.18 IFFT design functional verification result (cont) 88
5.19 Accumulator - Cordic functional verification result 89
5.20 Accumulator - Cordic functional verification result (cont) 89
5.21 Accumulator - Cordic functional verification result (cont) 90
5.22 Accumulator - Cordic functional verification result (cont) 90
5.23 Accumulator - Cordic functional verification result (cont) 90
5.24 Accumulator - Cordic functional verification result (cont) 91
5.25 Accumulator - Cordic functional verification result (cont) 91
5.26 GPS Acquisition result from Matlab model 92
5.27 GPS Acquisition result from Matlab model 92
5.28 Plotted correlation results of Matlab model versus our design (Top 1) 93
5.29 Plotted correlation results of Matlab model versus our design (Top 2) 93
5.30 Plotted correlation results of Matlab model versus our design (Top 3) 94
5.31 Plotted correlation results of Matlab model versus our design (Top 4) 94
5.32 Plotted correlation results of Matlab model versus our design (Top 5) 95
5.33 Correlation results of our design - Version: SCALE (TOP 1) 96
5.34 Correlation results of our design - Version: SCALE (TOP 2) 96
5.35 Correlation results of our design - Version: SCALE (TOP 3) 96
5.36 Correlation results of our design - Version: SCALE (TOP 4) 97
5.37 Correlation results of our design - Version: SCALE (TOP 5) 97
5.38 Example navigation message result with a test file with known incoming signal at PRN ID 11, Doppler shift 5500 Hz and code delay of 350 chips 98
5.39 Example navigation message result with a test file with known incoming signal at PRN ID 20, Doppler shift 8250 Hz and code delay of 1243 chips 99
5.40 Vivado Timing Report Non-scale Version 100
5.41 Vivado Timing Report Scale Version 100
5.42 Vivado Power Report Non-scale Version 101

5.43 Vivado Power Report Scale Version 101
5.44 Board run result of file gps_10s.bin 102
5.45 Board run result of file spain_gps_10s.mem 102

6.1 Proposed SDR Architecture 105
6.2 GPS acquisition design overview diagram - First new proposed architecture 107
6.3 GPS acquisition design overview diagram - Second new proposed architecture 108
6.4 GPS navigation message frame structure[9] 109

LIST OF TABLES

2.1 GPS Band and Description [10] 5
2.2 Comparison of Natural Order and Bit Reversal Order in DIF FFT . 22
2.3 Advantages and disadvantages of three acquisition methods . . . 35

4.1 Angles corresponding to different values of n from 0 to 4095 54
4.2 Example for convergent rounding 62

5.1 Comparison of synthesis and implementation result of scale and non-scale version 99
5.2 Comparison of timing result of scale and non-scale version 100
5.3 Comparison of power report of scale and non-scale version 100
5.4 GPS acquisition performance comparison 103

ACKNOWLEDGMENT

This thesis can be successfully completed thanks to the invaluable guidance and mentorship of Assoc. Prof. Pham Quoc Cuong, from the Faculty of Computer Science and Engineering - Ho Chi Minh City University of Technology. His support and expertise have played a crucial role in helping us not only to determine the goal but also to keep us motivated throughout stages of doing this thesis.

We would like to express our greatest gratitude toward members in the faculty, peers, and senior/fellow students who have shared their knowledge and experiences. Thanks to their support, whether they are discussions, suggestions, or advices, we are able to overcome challenges and obstacles throughout the journey of completing this thesis. The environment that our university, especially our faculty created has encouraged us to be a better version of ourselves every day, helping us to form skills, knowledge, as well as our personalities to be able to fit our society.

Finally, we are extremely grateful to our families for their unconditional support, love, and encouragement. Having their belief in us has always been an unlilmited source of strength and motivation, greatly helping us accomplish this milestone.

Project Authors
Nguyen Minh Khoa
Tran Minh Trung
Vu Nam Binh

ABSTRACT

For decades, Global Positioning System (GPS) has been an inevitable term in many fields such as army, traffic, and daily lives. The need for a fast and accurate GPS process becomes increasingly important and vital for applications in those fields. In the scope of hardware implementation, it is worth considering the performance, power, area, and timing of any GPS design that is put onto a chip/device. This thesis explores the design and implementation of a GPS receiver system operating on the Ultra96-V2 FPGA platform, aiming at real-time signal acquisition and tracking applications. A proposed architecture of GPS acquisition process has been implemented with two FFT instances and one IFFT instance (inspired by the PCS method) and tested using MATLAB-based golden models. A GPS tracking process is also implemented with PLL for carrier phase tracking and DLL for code phase tracking. With the operation frequency at $200MHz$, the acquisition time for a single-period PRN code search in our design is around 53 milliseconds.

ACRONYMS

2D Two Dimensions

AXI Advanced eXtensible Interface

BPSK Binary Phase-shift Keying

BRAM Block Random Access Memory

C/A Coarse/Acquisition

DLL Delay Locked Loop

DMA Direct Memory Access

FEC Forward Error Correction

FFT Fast Fourier Transform

FLL Frequency Locked Loop

FPGA Field Programmable Gate Array

FSM Finite State Machine

GNSS Global Navigation Satellite System

GPIO General Purpose Input/Output

GPS Global Positioning System

IFFT Inverse Fast Fourier Transform

LSB Least Significant Bit

LUT Look-Up Table

MSB Most Significant Bit

NCO Numerically Controlled Oscillator

NRMSE Normalized Root Mean Square Error

PCS Parallel Code Phase Search

PFS Parallel Frequency Search

PI Proportional-Integral

PLL Phase Locked Loop

PRN Pseudo Random Noise

RAM Random Access Memory

RMSE Root Mean Square Error

SDR Software-Defined Radio

SNR Signal-to-Noise Ratio

SS Serial Search

UAV Unmanned Aerial Vehicles

WHS Worst Hold Slack

WNS Worst Negative Slack

WPWS Worst Pulse Width Slack

1

INTRODUCTION

1.1. INTRODUCTION

In this thesis we focus on the design and implementation of a GPS receiver system on the Ultra96-V2 FPGA platform, focusing on achieving real-time signal acquisition and tracking applications. Our thesis aims to process and handle the GPS L1 C/A signals - a GPS band for civil usage that uses Coarse/Acquistion (C/A) code and operates with frequency at 1575.42MHz.

Our system architecture will utilize both the Programmable-Logic (PL) part and Processing-System (PS) part. The PL part is responsible for implementing all of signal processing tasks, including C/A Code generation, carrier wipe-off, FFT/IFFT operations, complex conjugation, DLL/PLL tracking loops, etc. On the other hand, the PS part will handle the input stream to the PL part and export output to UART port by utilizing Direct-Memory-Access (DMA) IP.

One of the most challenging task in this thesis is the integration of FFT and IFFT modules into the acquisition stage. Our design follows parallel code phase search (PCS) method, allowing the simultaneous estimation of all code phase delays at a given Doppler frequency value. To validate the correctness and effectiveness of the acquisition module design, a MATLAB-based simulation model is developed and used as a golden model for verification purposes.

The tracking stage is implemented through robust loops, including a Phase Locked Loop (PLL) for carrier tracking and a Delay Locked Loop (DLL) for code tracking. These two loops help the system to lock on targeted satellite signals for the navigation data demodulation step.

All design components are verified through simulation before synthesis and hardware testing. Through experimental evaluations, our design demonstrate

1

successful detection and tracking of GPS satellite signals, including the decoding of navigation preamble patterns.

1.2. OUTLINE

The rest of this thesis is organized as follows:

- **Chapter 2** provides the background knowledge about GPS systems including the history about GPS, structure of GPS signals, the GPS operation principle, and the detailed operation of GPS acquisition and tracking processes. We also reviews relevant research works related to GPS implementation on FPGA platform in this chapter.
- **Chapter 3** presents our proposed system architecture. It describes the overall design, the combination between the Programmable- Logic (PL) part and the Processing-System (PS) part. It also shows the key components for both the acquisition and tracking phases.
- **Chapter 4** discusses the design implementation of each module. It includes the block design of the acquisition modules and the tracking modules.
- **Chapter 5** presents how we setup experiment and how to evaluate our design. This chapter presents how we generate testing input, do functional verification for GPS acquisition and tracking. Resource usage will also be reported in this chapter.
- **Chapter 6** presents future work, including the integration with Software-Defined Radio (SDR) architecture, 2 extra designs for acquistion process, completely decode navigation message, and plans for expanding the receiver to support other GPS signal format.
- **Chapter 7** concludes the thesis by summarizing the achievements and contributions of the project.

2

BACKGROUND AND RELATED WORK

Before going further into the design architecture and implementation, we will set up the background theory of the Global Positioning System in this chapter. Moreover, we also reference and discuss some of the most relevant work to our GPS receiver design in the last section of this chapter.

2.1. INTRODUCTION TO GLOBAL POSITIONING SYSTEM (GPS)

2.1.1. BACKGROUND AND HISTORY

The Global Positioning System (GPS) is a satellite-based navigation system originally developed by the United States Space Force for military applications such as to determine precise locations, improve missile guidance systems, and support battlefield coordination. The system was designed to provide Position, Velocity, and Timing (PVT) data with high accuracy, enabling military users to operate effectively in any region of the world, day or night, under any weather conditions throughout periods the receiver had a full access to four or more satellites.

GPS operation principle is quite simple, which based on the trilateration concept. A method that measures distances from multiple satellites whose positions and times are known, by calculating the time delay betwee the transmission moment and reception moment of satellites signals, a receiver can accurately determine its poistion in three-dimensional space.

The history of satellite navigating began in 1957 ([11]), when the Sputnik 1 was launched into space to demonstrate how the Doppler effect can be used to track the movement of satellites from the ground. Therefore, scientists at Johns

Hopkins University realized that they can reverse the process, which means to make people on Earth determine their position based on satellite signals.

This idea paved the way for the development of Transit in 1958 - the first global satellite navigation system. Transit provided location updates for submarines and ships every few hours using many satellites. At that time, Transit was slow and also low accuracy, which requires the need for a novel solution.

Even though GPS was initially designed for military purposes, its usage for civilian purposes became increasingly significant. In 1983, President Ronald Reagan allowed people access to a lower version of GPS due to the tragic dowing of Korean Air Lines Flight 007. However, the U.S. government also introduced a policy called Selective Availability (SA) to limit the accuracy of GPS signals that can be seen by anyone, which makes the precision of military GPS unique and superiority.

In 2000, The U.S. government officially disabled the Selective Availability policy, which allows people to have full accuracy access of the Standard Positioning Service (SPS). After that, the combination of GPS into daily lives has grown sharply, leading to navigation systems in smartphones, vehicles, drones, etc.

Today, GPS has become an inevitable system in any fields around the world, leading to Its rapid expansion with GNSS, GLONASS, European Union's Galileo, and China's BeiDou. The impact of GPS is growing more and more important in our daily lives, with many applications: mapping, transportation, marine, aviation, fitness trackers, etc.

2.1.2. BASIC PRINCIPLES OF OPERATION

The Global Positioning System (GPS) worked by using a constellation of satellites in Medium Earch Orbit (MEO) to continuously transmit orbital information, its current information, and its location in Earth Centered Earth Fixed (ECEF) coordinates and its time. From the information provided by the satellites, the receiver on the ground could accurately calculate and update its real-time location on Earth.

GPS system is broadcasted by satellites on several radio frequencies in a radio band called L band, a designation by the Institute of Electrical and Electronics Engineers (IEEE) for frequencies from 1 GHz to 2 GHz or wavelength of 30 cm to 15 cm respectively. The GPS system broadcast L band is further divided into several smaller bands with respective pseudonoise code as follow:

Band and Code	Frequency	Description
L1 C/A	1575.420 MHz	The original GPS band for civil usage, using Coarse/Acquisition (C/A) code.
L1C	1575.420 MHz	Intended to replace L1 C/A, but not fully operational at the time of this report.
L1 P(Y)	1575.420 MHz	Old GPS L1 version used exclusively by the US military.
L1M	1575.420 MHz	Replacement for L1 P(Y) system for US military usage.
L2 CM	1227.600 MHz	Civilian-use GPS band with anti-jamming features.
L2 CL	1227.600 MHz	Enhanced civilian GPS band for challenging signal environments.
L2 P(Y)	1227.600 MHz	Older US military GPS band.
L2M	1227.600 MHz	Newer US military GPS band.
L3	1381.050 MHz	Used by the US military for nuclear explosion detection.
L4	1379.913 MHz	Intended for ionospheric correction studies; not yet operational.
L5I and L5Q	1176.450 MHz	Advanced civilian GPS band used in aviation, maritime, and railway safety.

Table 2.1: GPS Band and Description [10]

In the GPS L1 C/A system, the carrier signal is modulated using the Binary Phase-Shift Keying (BPSK) modulation method in which the phase of the carrier signal is encoded by 2 symbols 0 and 1, or in other word, the phase of the carrier wave for binary value of 1 would be 180° out of phase with phase for binary value of 0. The symbol rate of the BPSK modulation used is 1023 symbols/second while the bit rate of navigation message is 50 bits/second.

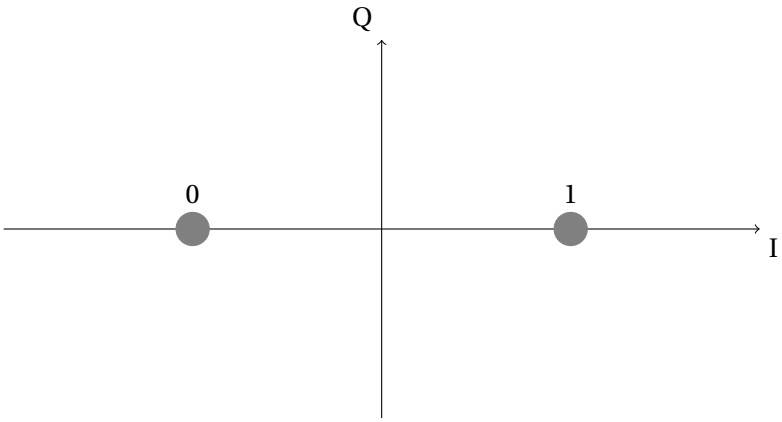


Figure 2.1: The Constellation Diagram of BPSK Modulation Used in GPS L1

COARSE/ACQUISTION CODE

Coarse/Acquisition Code (C/A Code) is a simple but important element in the Global Positioning System. The C/A Code is a pseudo-random noise (PRN) sequence, which is a 1023-bit binary sequence, unique for each satellite and sent repeatedly every 1 millisecond. The word "random" only has the meaning of the random appearances of bit 0 and bit 1 in the sequence. However, this sequence is not as random as its name but instead, deterministically created based on a mathematical algorithm.

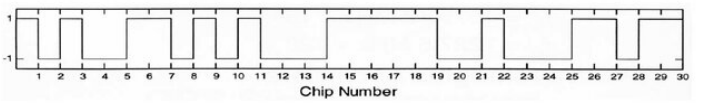


Figure 2.2: Coarse/Acquisition Code Example [1]

The C/A Codes are generated with the aim of identification for each satellite but this does not mean that these codes can be assigned any random numbers. They are created with the requirement of low cross-correlation between every random pair of codes. For instance, the C/A Code of satellite 1 and the C/A Code of satellite 2 should not have a high correlation. If this happens, it is possible for the GPS receiver to make a mistake in distinguishing 2 satellites. Therefore, people developed a mathematical algorithm to generate these 1023-bit binary sequences for all 32 available satellites. Following the clear explanation in the blog [12] and paper [13], the algorithm operates as in Fig 2.3:

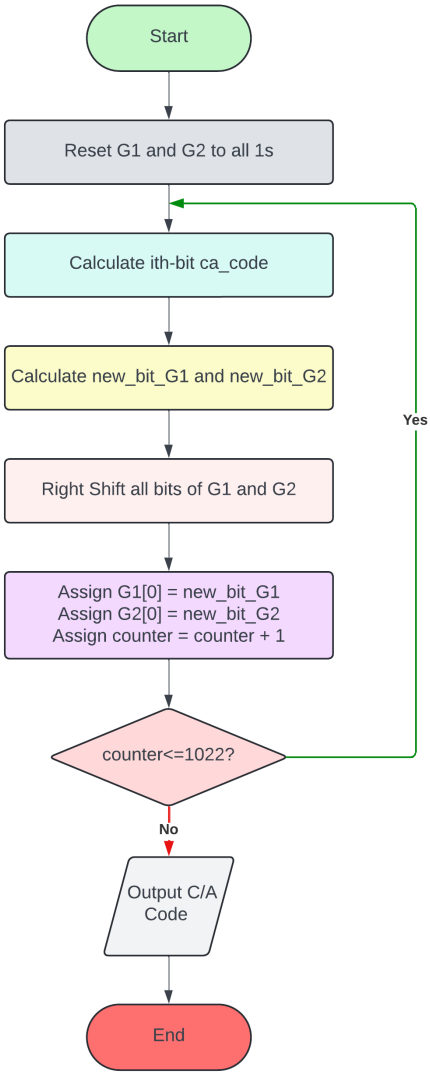


Figure 2.3: Coarse/Acquisition Code Generation Flowchart

The algorithm consists of 3 components:

- G1 generator: G1 is a 10-bit shift register, which is initialized as all bit 1s. At each iteration, the new value of G1 is calculated by a polynomial feedback:

$$f_{G1}(x) = 1 + x^3 + x^{10} \tag{2.1}$$

- G2 generator: G2 is a 10-bit shift register, which is initialized as all bit 1s. At each iteration, the new value of G2 is calculated by a polynomial feedback:

$$f_{G2}(x) = 1 + x^2 + x^3 + x^6 + x^8 + x^9 + x^{10} \tag{2.2}$$

- Phase selector: After calculating G1 and G2 shift registers, we need to perform 1 more step to calculate the corresponding output bit at the i-th iteration based on bits of G1 and G2. With the G1 register, we will take its oldest bit (the most right bit), but with the G2 register, which bits are chosen for calculating depending on each separate GPS satellite. The phase selector as its name is responsible for choosing the correct bits of the G2 register to combine with G1 most right bit to generate the output. The way this phase selector chooses which bits to use for calculation is defined in the GPS interface as in Fig 2.4 and Fig 2.5.

SV ID No.	GPS PRN Signal No.	Code Phase Selection		Code Delay Chips		First 10 Chips Octal*	First 12 Chips Octal
		C/A(G2$_i$)**	(X2$_i$)	C/A	P	C/A	P
1	1	2 ⊕ 6	1	5	1	1440	4444
2	2	3 ⊕ 7	2	6	2	1620	4000
3	3	4 ⊕ 8	3	7	3	1710	4222
4	4	5 ⊕ 9	4	8	4	1744	4333
5	5	1 ⊕ 9	5	17	5	1133	4377
6	6	2 ⊕ 10	6	18	6	1455	4355
7	7	1 ⊕ 8	7	139	7	1131	4344
8	8	2 ⊕ 9	8	140	8	1454	4340
9	9	3 ⊕ 10	9	141	9	1626	4342
10	10	2 ⊕ 3	10	251	10	1504	4343
11	11	3 ⊕ 4	11	252	11	1642	|
12	12	5 ⊕ 6	12	254	12	1750	|
13	13	6 ⊕ 7	13	255	13	1764	|
14	14	7 ⊕ 8	14	256	14	1772	|
15	15	8 ⊕ 9	15	257	15	1775	|
16	16	9 ⊕ 10	16	258	16	1776	|
17	17	1 ⊕ 4	17	469	17	1156	|
18	18	2 ⊕ 5	18	470	18	1467	|
19	19	3 ⊕ 6	19	471	19	1633	4343

* In the octal notation for the first 10 chips of the C/A code as shown in this column, the first digit (1) represents a "1" for the first chip and the last three digits are the conventional octal representation of the remaining 9 chips. (For example, the first 10 chips of the C/A code for PRN Signal Assembly No. 1 are: 1100100000).

** The two-tap coder utilized here is only an example implementation that generates a limited set of valid C/A codes.

⊕ = "exclusive or"

NOTE #1: The code phase assignments constitute inseparable pairs, each consisting of a specific C/A and a specific P-code phase, as shown above.

Figure 2.4: G2 Selector Table 1 [2]

SV ID No.	GPS PRN Signal No.	Code Phase Selection C/A(G2$_j$)****	(X2$_i$)	Code Delay Chips C/A	P	First 10 Chips Octal* C/A	First 12 Chips Octal P
20	20	4 ⊕ 7	20	472	20	1715	4343
21	21	5 ⊕ 8	21	473	21	1746	
22	22	6 ⊕ 9	22	474	22	1763	
23	23	1 ⊕ 3	23	509	23	1063	
24	24	4 ⊕ 6	24	512	24	1706	
25	25	5 ⊕ 7	25	513	25	1743	
26	26	6 ⊕ 8	26	514	26	1761	
27	27	7 ⊕ 9	27	515	27	1770	
28	28	8 ⊕ 10	28	516	28	1774	
29	29	1 ⊕ 6	29	859	29	1127	
30	30	2 ⊕ 7	30	860	30	1453	
31	31	3 ⊕ 8	31	861	31	1625	
32	32	4 ⊕ 9	32	862	32	1712	
65	33***	5 ⊕ 10	33	863	33	1745	
66	34**	4 ⊕ 10	34	950	34	1713	
67	35	1 ⊕ 7	35	947	35	1134	
68	36	2 ⊕ 8	36	948	36	1456	
69	37**	4 ⊕ 10	37	950	37	1713	4343

* In the octal notation for the first 10 chips of the C/A-code as shown in this column, the first digit (1) represents a "1" for the first chip and the last three digits are the conventional octal representation of the remaining 9 chips. (For example, the first 10 chips of the C/A-code for PRN Signal Assembly No. 1 are: 1100100000).

** C/A-codes 34 and 37 are identical.

*** PRN sequence 33 is reserved for other uses (e.g. ground transmitters).

**** The two-tap coder utilized here is only an example implementation that generates a limited set of valid C/A-codes.

⊕ = "exclusive or"

NOTE #1: The code phase assignments constitute inseparable pairs, each consisting of a specific C/A and a specific P-code phase, as shown above.

Figure 2.5: G2 Selector Table 2 [2]

The G1 and G2 generator using polynomial feedback sounds quite complicated but in fact, it is effortless to implement this Coarse/Acquisition Generation block as in Fig 2.6.

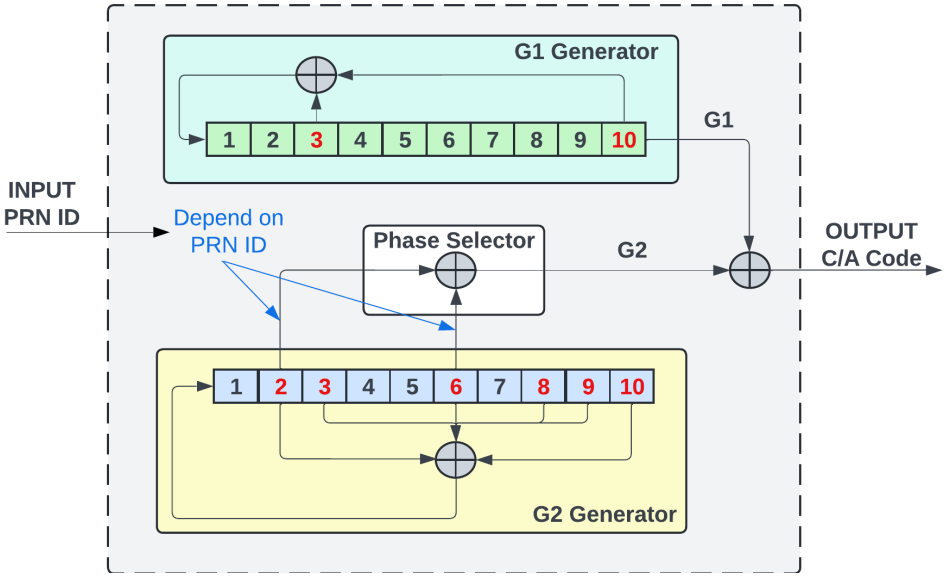


Figure 2.6: Coarse/Acquisition Code Generator Block Diagram

This block diagram is an example of creating the C/A Code of the satellite

1. In Fig 2.4, the code phase selection for G2 is 2 and 6 (SV ID No.1). In this block diagram, the red color positions in the G1 and G2 generators indicate a fixed selection of the locations of bits to calculate new value for these 2 shift registers. The positions 2 and 6 are chosen for phase selection to give the G2 output. Then G2 and G1 will be XORed to give the output C/A Code bit.

After we collect and demodulate the received signal to retrieve navigation messages, the next step is to determine where the GPS receiver is currently located on Earth based on what we have. The idea behind this process relies on the principle of trilateration, which is a method that helps determine the coordinates of an object based on the distance from this object to at least three other ones with known positions in theory where everything is ideal so that no noise exists. This section explores the detailed process of position calculation, starting from the received satellite data to the final computed coordinates.

As mentioned above, we need to have at least three satellites' information to calculate the receiver position but this is only in theory. In fact, to ensure the precise operation of the GPS receiver, we need at least one more satellite's information to handle the problems coming from the clock offset between the receiver and the satellite transmitter. A navigation message sent from each of four satellites consists of many fields but to calculate the user's position, we only need two things:

- The satellite's coordinates in the ECEF coordinate system (x, y, and z).
- The distance from the satellite to our GPS receiver.

2-DIMENSIONS TRILATERATION METHOD

Initially, we receive data from only one available satellite in the sky as shown in Fig 2.7. However, receiving only one satellite's position information can not help us determine where we are. In Fig 2.7, we draw four different positions on the West, East, South, and North of the circle but these are only symbolic positions. In reality, with data obtained from one satellite, the GPS receiver's position can be every point in the circumference.

Figure 2.7: 1 Satellite Results in Multiple Valid Points on The Circle

Now, if we increase receiving one more satellite's data, we will still not obtain the position of a user. At this time, two circumferences of two satellites will intersect at two points, which gives us two possible output user positions but there is no way to remove one of them and accept the other. Fig 2.8 shows us two existing intersections of the blue and green circumferences.

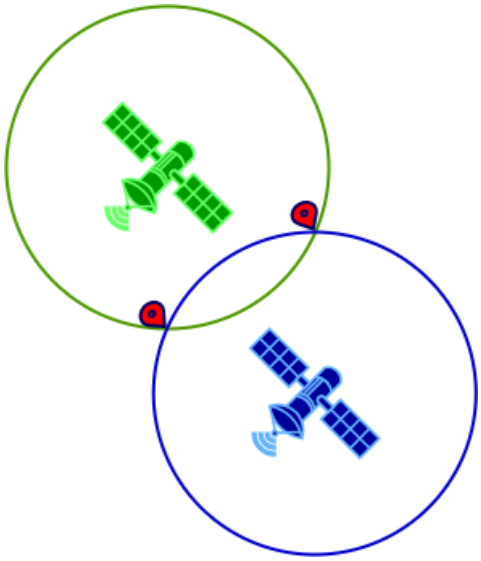


Figure 2.8: 2 Satellite Circles Result in 2 Intersection Points

Theoretically, if we receive enough data from three satellites, we can determine exactly where we are on Earth. In Fig 2.9, there is only one unique inter-

section among three circumferences, which is the expected user's position.

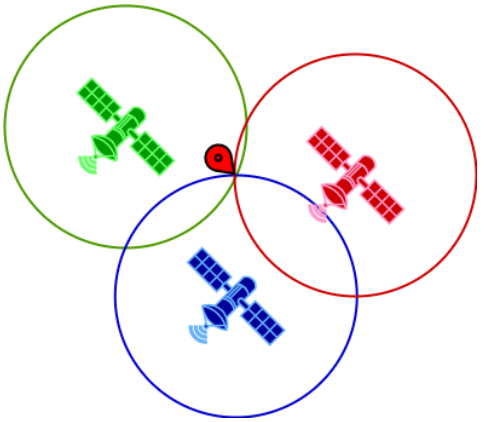


Figure 2.9: 3 Satellite Circles Result in 1 Intersection Point

3-DIMENSIONS TRILATERATION METHOD

We used the 2D trilateration model only to simplify the idea of the GPS receiver's position determination depending on the received data from GPS satellites. Now, we will move to the 3D trilateration model, which is the actual model utilized to determine the GPS receiver's position on Earth. To understand the need to use at least four satellite data to determine precisely the position of the user, we will go step by step:

- **The GPS Receiver receives data from only one satellite:** In Fig 2.10, we only draw simply four points on the sphere, but in reality, given the distance between a satellite to the receiver, our position can be any points as long as they lie on the surface of the sphere covering the satellite.

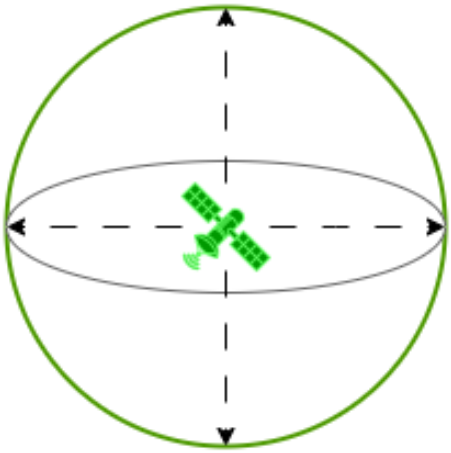


Figure 2.10: 1 Satellite Sphere Results in Multiple Valid Positions

- **The GPS Receiver receives data from two satellites:** In this case, the intersection between two spheres gives us a circle as shown below. This means that the solution (position) of the GPS receiver can be anywhere on this circle. Visually in Fig 2.11, any point on the dashed boundary of the yellow-filled circle is a valid solution. Therefore, the final solution is still hidden and we need to increase receiving more satellites' information.

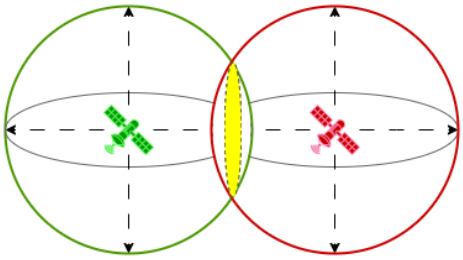


Figure 2.11: 2 Satellite Spheres Intersection Results in a Circle

- **The GPS Receiver receives data from three satellites with two intersections:** In the 2D trilateration, there will be only one possible solution (position) which is the intersection among three circles covering three satellites. However, it is not that easy in the case of a 3D trilateration model because the intersection among three spheres can result in many possible situations mathematically. The intersection among three spheres

usually results in two points represented by two bold dots lying on the dashed boundary of the yellow-filled circle in the given figure below. We can imagine simply how these two points appear. Firstly, we intersect two spheres from two satellites to get a circle. Then, we intersect the third sphere from the third satellite with the circle to get these two points. One of them is reasonable and the other will be removed due to its non-real value. The term "non-real value" here means that this point is not located on Earth. There are two possible non-real values: the point is located somewhere higher than the ground or it is located somewhere underground while the other solution will give us a position located on Earth as shown in 2.12.

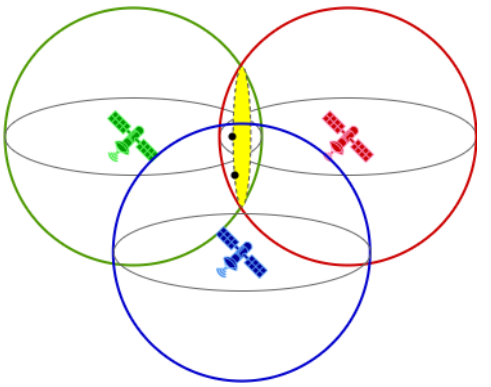


Figure 2.12: 3 Satellite Spheres Result in 2 Intersection Points

- **The GPS receiver receives data from three satellites with multiple intersections:** If you intersect the circle intersection between two satellites' sphere with the third satellite's sphere, the case of multiple valid intersection points can only happen when the circle lies completely on the surface of the third sphere or to simplify, the radius of the circle intersection must be equal to the radius of the third sphere's radius. Consequently, the condition for this situation to happen is that three satellites must be located at the same position, which is absolutely nonsense. Therefore, we can ignore this problem.
- **The GPS receiver receives data from three satellites with no intersection points:** This situation is also ignored. The reason is that the signals from these three satellites must reach the GPS receiver. This means that the location of the GPS receiver is an intersection point of the three spheres. Therefore, in this mathematical model, there must exist at least one intersection point.

- **The GPS receiver receives data from four satellites:** In the situation of three satellite spheres resulting in two possible intersection points, we stated that we can eliminate one of them due to the non-real value. However, we will not deal with this situation like that. Instead, we will receive data from at least four satellites to refine the solution and more importantly handle the error created by clock offset. All the concepts we have talked about until now are based on the assumption of ideal conditions for transmitting signals from the GPS satellite to the receiver. In reality, when a signal is transmitted from the satellite to the receiver, there will be many noises occurring and leading to errors in the signal such as ionospheric error, and tropospheric error, ... but they are almost not so significant for us to consider except for the case of the clock offset, which is the difference in the clock between the transmitter and the receiver. Therefore, we will use at least four satellites' information to handle the problems of two possible intersection points using only three satellites' information and the error caused by clock offset.

2.2. GPS ACQUISITION PHASE

GPS acquisition phase is the first step in the process of determining which satellites are available in view of the GPS receiver's location. Throughout this process, the GPS receiver also needs to roughly estimate two important parameters which are Doppler shift and code phase delay. Before we dive deeply into the operation of this phase, some necessary background theories will be presented.

2.2.1. DOPPLER FREQUENCY

The Doppler frequency, also known as Doppler shift, refers to the variation in frequency or wavelength of a signal that occurs due to the relative motion between the transmitter and the receiver. Doppler effect appears in many fields of our daily life, not only in the GPS system. An interesting and iconic example of this effect is when people hear the sound of the siren of an ambulance. The closer the ambulance moves to you, the faster its siren's pitch sound is.

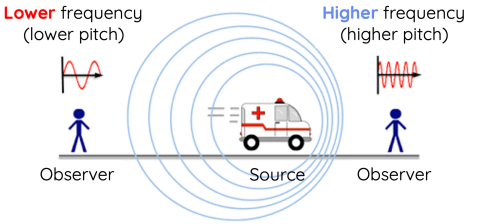


Figure 2.13: Doppler effect example (adopted from [3])

As in Figure 2.13, we can easily observe that the sound wave from the ambulance seems to be "compressed" when it moves closer to the right-side observer (higher pitch) and "stretched" when it moves far away from the left-side observer. This interesting effect also happens in case of GPS signal transmission and it is mainly caused by the high-speed movement of the satellites while the impact of the receiver's movement also creates the Doppler shift, but with a low fraction.

The equation 2.3 shows the shifted version of the signal coming from a GPS satellite [14].

$$x_{\text{IN}}[k] = A(t)\tilde{s}_T(t-\tau(t))e^{j(2\pi f_D(t)t+\phi(t))}\Big|_{t=kT_s} + n(t)\Big|_{t=kT_s} \tag{2.3}$$

where

- $A(t)$ is the signal amplitude.
- $\tilde{s}_T$ is a filtered version of $s(T)$.
- $s_T(t)$ is the digital signal sampled from the original analog GPS signal.
- $\tau(t)$ is a time-varying code delay.
- $f_D(t)$ is a time-varying Doppler shift.
- $\phi(t)$ is a time-varying carrier phase shift.
- $n(t)$ is random noise from the environment.
- T_s is the sampling period.

If we focus on the exponential term that includes the Doppler shift frequency and process further more, we can observe that for a discrete signal, if it is affected by the Doppler shift effect, it will be shifted in time by multiplying with the exponential given in 2.4

$$e^{j2\pi n\frac{f_d}{f_s}+\phi} \tag{2.4}$$

where

- n is the discrete index.
- f_d is the Doppler shift frequency.
- f_s is the sampling frequency.
- ϕ is the initial carrier phase.

Clearly, to wipe off Doppler shift effect, we need to multiply the received signal with an amount of

$$e^{-j2\pi n\frac{f_d}{f_s}} \tag{2.5}$$

The problem is that the GPS receiver never knows the exact value of the Doppler shift frequency when it receives a signal from a specific GPS satellite at any given time. Therefore, we need a solution to find the correct Doppler shift frequency here and this job is the responsibility of the GPS acquisition phase. Clearly, the receiver can never search the value of the Doppler shift on an infinite set, so instead, people observe that with a normal civilian application on Earth, the Doppler shift range will be between -10 and +10 kHz. This means that the first responsibility of the GPS acquisition phase is to estimate the Doppler shift frequency in the range of -10 to +10 kHz.

2.2.2. CODE PHASE DELAY

As we mentioned earlier, every GPS L1 C/A Code has 1023 chips which are sent from the GPS satellite periodically every 1 millisecond (f = 1.023 MHz). Moreover, they are sent continuously, and there are no mechanisms in the GPS system such that the transmitter can inform the receiver when it starts sending out the first chip in its period. To acquire a satellite signal, the GPS receiver needs to create a locally generated C/A code itself and then correlate that code with the received one. Later on, we will present the concept of cross-correlation and autocorrelation between 2 signals.

There are many cases in which the receiver starts receiving signal from the satellite at the moment that the satellite has already sent many chips of the signal instead of the first chip. The offset between the received GPS signal and the locally generated one is called the code phase delay. In case the GPS receiver can not recognize this phase delay, it will never be able to acquire any GPS signals. Therefore, the second responsibility of the GPS acquisition phase is to estimate the code phase delay.

2.2.3. FFT AND IFFT APPLICATION IN ACQUISITION PHASE

This algorithm utilizes a really powerful mathematical tool to help reduce one dimension out of the search space, which is the Fast Fourier Transform. The characteristic of the C/A code is periodic every $1ms$, so we can easily exploit the circular correlation here.

DISCRETE FOURIER TRANSFORM

In the design of the GPS acquisition, we need to calculate the convolution between the incoming signal and the local replica signal. In mathematics, due to the convolution theorem, the convolution between two signals in the time domain is equal to their multiplication in the frequency domain, and this is where the Fourier transform is really applicable. For simplicity, this transform helps us convert a signal in the time domain to the frequency domain. Suppose that we have a continuous function in the time domain $f(x)$, its Fourier transform is as follows.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt \tag{2.6}$$

where

- $f(t)$ is the time-domain function.
- $F(\omega)$ is the frequency-domain representation.
- ω is the angular frequency, which is equal to $2\pi f$ with f being the frequency, not the function.
- $e^{-j\omega t}$ is the complex exponential component.

Fourier transform mathematically works well, but in fact, a computer or a digital signal processor cannot work with a continuous signal. Due to the above-mentioned limitation, the Discrete Time Fourier Transform (DTFT) is created as the following formula:

$$F(e^{j\omega}) = \sum_{-\infty}^{\infty} f[n] e^{-j\omega n} \tag{2.7}$$

However, this type of Fourier transform can not yet be used in practice. The output of the DTFT is a continuous frequency-domain representation of the discrete time signal f[n], and unfortunately, a computer or a digital signal processor cannot work well with infinite data. Otherwise, they operate smoothly with a finite data set. The problem of DTFT can be solved by the Discrete Fourier Transform (DFT), which converts a finite and discrete time signal in the

time domain to its finite and discrete representation in the frequency domain as the following formula:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1 \tag{2.8}$$

Everything is fine right now due to the ability to implement the DFT in practice, but if we take a closer look, we will see a problem with the DFT, which is its expensive computation. Computing quickly, we have:

- N multiplications and N-1 additions for each output of the frequency component $X(k)$, so we have 2N-1 operations, which can be scaled to N operations.
- N frequency components corresponding to N outputs.

Therefore, to fully compute all the outputs of the frequency components, it costs $N \times N = N^2$ operations, which is painfully expensive. Many decades ago, people had to accept that they had to implement this extremely computationally expensive process to achieve their needs, but this was only possible with a small set of data sets. However, in 1965, a powerful algorithm appeared and made a strong outstanding hit to the history of computational mathematics and signal processing, which is the Fast Fourier Transform algorithm (FFT).

FAST FOURIER TRANSFORM

Fast Fourier transform is a very powerful algorithm utilized to implement the Discrete Fourier Transform. The beauty of FFT is showed in its elegance and efficiency. It is a true masterpiece of algorithmic thinking: by exploiting the symmetry and periodicity of the Fourier transform, the FFT optimizes the computation, which reduces unnecessary calculations. It is not just faster but an artistic example of how computational theory can meet practical utility in a harmonious way.

Utilizing the characteristics of symmetry and periodicity of the Fourier transform, the FFT algorithm uses the divide-and-conquer approach. This approach reduces the total number of computations. A key concept in the FFT is the "butterfly" operation, where pairs of data points are combined in a specific way to generate the DFT values for smaller subproblems. This operation is repeated in multiple stages, each time halving the size of the problem. The process involves decomposing the signal into its even and odd components, which are then processed independently, leading to a more efficient solution.

To implement the FFT algorithm, people have many ways, which are decimation in time and decimation in frequency. Depending on which type of

implementation, we will have the bit-reversal data reordering at the input or output of the module.

BUTTERFLY BEHAVIOR IN FFT/IFFT

As we mentioned above, FFT is used in our design to solve the original computational cost problem of the DFT algorithm. Recall that the formula of the Discrete Fourier Transform is:

$$X[\frac{k}{N}] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n} \tag{2.9}$$

The origin problem takes N data samples and calculates one summation across every input for each value of k to produce N data. Moreover, multiplication is also required for every element in the summation, which will eventually cost N^2 multiplications to complete the calculation. Obviously, this primitive approach is not considered to be resource-utilized and need to be updated. This is where the DIF (Decimation-in-Frequency) FFT (we chose to follow this method rather than Decimation-in-Time in this project) comes in handy.

First let's break the original problem into two subproblems:

$$X\left[\frac{k}{N}\right] = \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] e^{-j2\pi \frac{k}{N} n} + x[n+\frac{N}{2}] e^{-j2\pi \frac{k}{N}(n+\frac{N}{2})} \right) \tag{2.10}$$

$$X\left[\frac{k}{N}\right] = \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x\left[n+\frac{N}{2}\right] e^{-j2\pi \frac{k}{2}} \right) e^{-j2\pi \frac{k}{N} n} \tag{2.11}$$

Consider even and odd frequency bins $2k$ and $2k+1$, we get:

$$\begin{bmatrix} X\left[\frac{2k}{N}\right] \\ X\left[\frac{2k+1}{N}\right] \end{bmatrix} = \begin{bmatrix} \sum_{n=0}^{N/2-1} \left(x[n] + x\left[n+\frac{N}{2}\right] e^{-j2\pi \frac{2k}{N} n} \right) e^{-j2\pi \frac{2k}{N} n} \\ \sum_{n=0}^{N/2-1} \left(x[n] + x\left[n+\frac{N}{2}\right] e^{-j2\pi \frac{2k+1}{2}} \right) e^{-j2\pi \frac{2k+1}{N} n} \end{bmatrix} \tag{2.12}$$

We can simplify equation 2.12 with simple facts: $e^{k2\pi} = 1$ and $e^{(2k+1)\pi} = -1$.

$$\begin{bmatrix} X\left[\frac{2k}{N}\right] \\ X\left[\frac{2k+1}{N}\right] \end{bmatrix} = \begin{bmatrix} \sum_{n=0}^{N/2-1} \left(x[n] + x\left[n+\frac{N}{2}\right] \right) e^{-j2\pi \frac{2k}{N} n} \\ \sum_{n=0}^{N/2-1} \left(x[n] - x\left[n+\frac{N}{2}\right] \right) e^{-j2\pi \frac{1}{N} n} e^{-j2\pi \frac{2k}{N} n} \end{bmatrix} \tag{2.13}$$

By doing such operations, we have already split the original DFT problem into two similar DFTs, each having half the size of the original. As a result, the term $e^{-j2\pi \frac{1}{N} n}$ in the second line is commonly called the twiddle factor (W_N^k). This twiddle factor is crucial for this algorithm since we can use a look-up table to store its values based on its properties.

- $W_N^k = W_N^{k+N}$
- $W_N^k \cdot W_N^m = W_N^{k+m}$
- $W_N^{N-k} = (W_N^k)^*$
- $W_N^{-k} = (W_N^k)^*$
- $(W_N^k)^m = W_N^{km}$

As mentioned above, the operation $x[n] + x[n + \frac{N}{2}]$ shows that we only need to store half of the input size in a memory block and perform addition for every input after the input index at $\frac{N}{2}$.

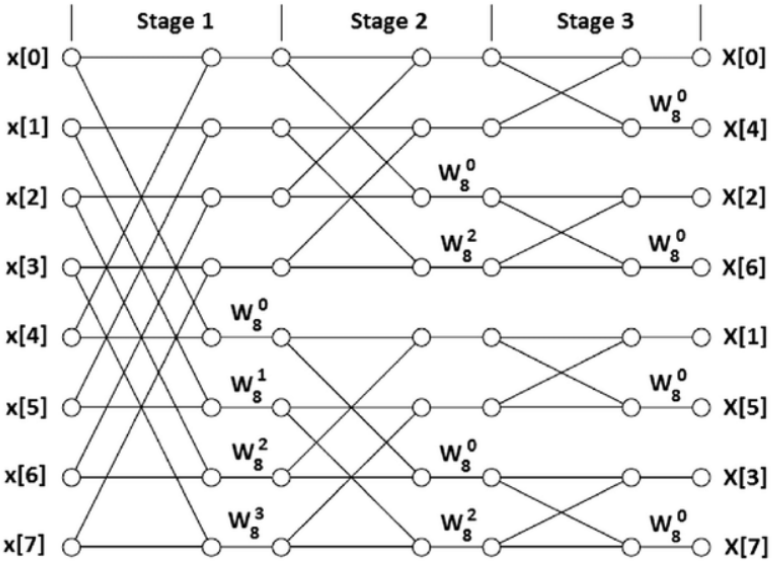


Figure 2.14: Butterfly behavior of 8-points FFT

In Fig. 2.14, it is clear that we only need to store 4 inputs, and from the fifth input onward, we can compute the results in Stage 1. Moreover, after completing all 4 outputs of Stage 1, we can immediately begin processing Stage 2 without waiting for the remaining 4 outputs. One notable fact is that the output is in bit-reversal order:

Natural Index	Binary	Reversed Binary	Reversed Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Table 2.2: Comparison of Natural Order and Bit Reversal Order in DIF FFT

Therefore, we need to rearrange the output to obtain the final result of the FFT operation. Moreover, the technique demonstrated in this example will still work even when the number of inputs N is increased. Furthermore, all the techniques and algorithms described above will also apply in the case of IFFT (Inverse Fast-Fourier Transform) problems. The only difference is the sign of the twiddle factors. The formula of twiddle factor in FFT is:

$$W_N^k = e^{-j\frac{2\pi k}{N}}$$

While the twiddle factor in IFFT is given by:

$$W_N^{-k} = e^{j\frac{2\pi k}{N}}$$

Therefore, the implementation of IFFT can copy the same butterfly structure as in FFT implementation, with just the difference in twiddle factor sign. Besides, it is important to normalize the final IFFT result by N to obtain the correct output.

CROSS-CORRELATION BETWEEN TWO SIGNALS

In GPS Acquisition process, the goal is to determine the correct pair of the Doppler frequency shift and the code phase delay offset. Obtaining this can be done by performing cross-correlation operation between the received PRN code and the local generated PRN code. The correlation formula is given below in 2.14.

$$\text{corr}(x[n], h[n]) = \sum_{k=0}^{N-1} h[k] \cdot x[n+k] \tag{2.14}$$

Here, $x[n]$ represents the received GPS signal, $h[n]$ is the locally generated PRN code, and N is the length of the PRN code. Direct computation of cross-correlation is obviously costly because the complexity of the algorithm is $O(n^2)$.

Therefore, to reduce this computational cost, we apply the Fast Fourier Transform (FFT), which significantly reduces the complexity of whole process to only $O(N\log(N))$.

To resource-saving compute the whole cross-correlation operation, we perform the calculation in the frequency domain using the Fourier Transform. Take the Fourier Transform on both sides of the equation, we have:

$$FFT\{\text{corr}(x[n],h[n])\} = FFT\{h[k]\}\cdot FFT^*\{x[n+k]\} \tag{2.15}$$

Besides, we need to conjugate the FFT result of the other signal before doing multiplication. That is:

$$\text{corr}(x[n],h[n]) \xrightarrow{F} H(f)\cdot X^*(f) \tag{2.16}$$

Therefore:

$$\text{corr}(x[n],h[n]) = \text{IFFT}(X(f)\cdot H^*(f)) \tag{2.17}$$

Recall that in practical GPS acquisition process, we perform the FFT of the received signal and the FFT of the local PRN code. The next step is to multiply the FFT of the received signal with the conjugate of the FFT of the local PRN code:

$$Y(f) = X(f)\cdot H^*(f) \tag{2.18}$$

The correlation result is obtained by applying IFFT:

$$y[n] = \text{IFFT}(Y(f)) \tag{2.19}$$

Since both the received signal and the PRN code may contain complex numbers (especially when considering the Doppler shift and phase variations), it is essential to take the conjugate of the PRN code's FFT. The conjugation ensures that the complex conjugate multiplication correctly aligns the phases during cross-correlation. Without taking the conjugate process, the result can become inverted or shifted, which leads to a fail result of peak detection. The peak in the cross-correlation result output is the correct pair of code phase delay offset and correct Doppler frequency shift.

2.2.4. GPS ACQUISITION

From 2.2.1 and 2.2.2, we can easily observe that the GPS acquisition phase is no more than a two-dimensional search mechanism with:

- One dimension is for the Doppler shift estimation. As we mentioned earlier, the value of this Doppler shift for normal civilian applications can range from -10 KHz to +10 KHz. However, we will not search for all values inside this range; instead, a step of 250 Hz will be used. Therefore, there will be totally 81 Doppler shift frequencies that we need to search for, which are -10000 Hz, -9750 Hz, -9500 Hz, ..., 0 Hz, 250 Hz, 500 Hz, ..., 9500 Hz, 9750 Hz, and 10000 Hz. The lower the value of the step, the more precise the result of the Doppler shift is, and it also helps the tracking phase track the satellites. However, if we use such a small step value, the number of Doppler shift frequencies to search for also increases and leads to higher unnecessary delay of the acquisition phase. That is the reason why we choose the value of 250 Hz for a step here in this project.

- One dimension is for the code phase delay estimation. Theoretically, there are 1023 chips in a single 1-millisecond period of the PRN code, so there will be 1023 code phase bins that the GPS acquisition phase needs to search for, which are from 0 to 1022. However, it can vary depending on the number of samples used. For example, the PRN code is sent out at frequency of 1.023 MHz but the GPS receiver samples it at 4.096 MHz, then in 1 millisecond, there are 4096 samples instead of 1023 samples. Therefore, the dimension for code phase delay now includes 4096 values from 0 to 4095. Instead, they are still those 1023 chips but sampled at a higher frequency and to find the real code phase delay (from 0 to 1022 chips), we can convert back the result of code phase delay we receive from the output of the GPS acquisition phase by using formula 2.20.

$$c_{delay} = \frac{1023000}{f_s} \times c^*_{delay} \tag{2.20}$$

  where c_{delay} is the real code phase delay (in the range of 0 to 1022), f_s is the sampling frequency and c^*_{delay} is the result of the code phase delay of our GPS acquisition design.

Throughout its working period, the GPS acquisition phase will continuously try to correlate the input signal with the locally generated PRN code inside to find some peak values. These values indicate some matched satellites have been found, and, along with it, the acquisition phase also estimates and provides the Doppler shift and code phase delay values to the tracking phase. Figure 2.15 shows an example of the two-dimensional search space for the GPS acquisition phase.

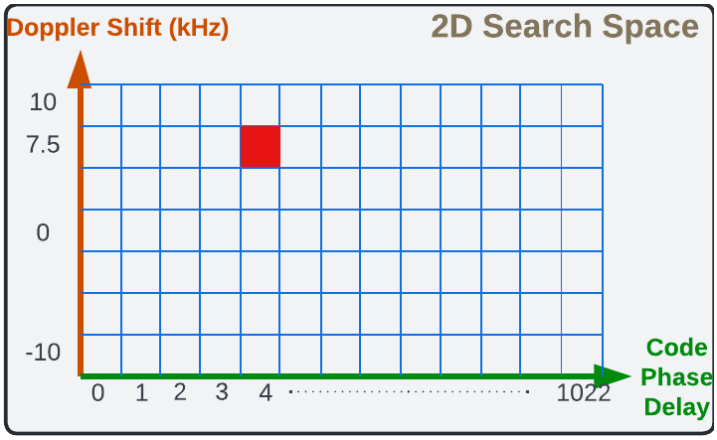


Figure 2.15: Two-dimensional search space

To give a clearer view about the two-dimensional search space, along with the value of the peak correlation result, combine them to create a three-dimensional space with the third dimension being the correlation value as shown in figure 2.16.

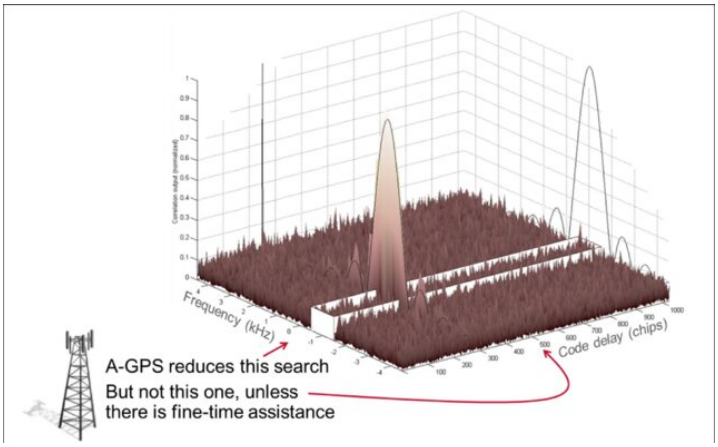


Figure 2.16: Three-dimensional space (adapted from [4])

An immensely sharp peak only occurs at the expected combination of the Doppler shift and the code phase delay. Among other correlation results, they are quite low, or sometimes, in addition to the sharp peak at the correct position, we may observe some more small peaks, especially around the correct position. However, these peaks are still really small in comparison to the highest peak. Practically, the search procedure could be terminated sooner as long

as the correlation result exceeds a predefined threshold value.

Given the operation of the GPS acquisition phase, the easiest algorithm to implement this is to search sequentially all possible combinations in this 2-dimensional search space. However, a big trade-off can be easily observed here, which is the delay in output results:

- Doppler shift ranges from -10 kHz to +10 kHz with step 250 Hz. Therefore, we have $\frac{+10000-(-10000)}{250} + 1 = 81$ Doppler shift values to search.
- Code phase delay ranges from 0 to 1022. Therefore, we have $\frac{+1022-(0)}{1} + 1 = 1023$ code phase delays.
- Totally, we will have $81 \times 1023 = 82863$ combinations to search. In fact, if the correlation result is not so good due to much noise in the signal, we must reduce the step between two continuous bins in the Doppler shift, which leads to a rise in the number of Doppler shift values to search. This also affects the total number of combinations to search over the 2-dimensional search space.

Searching over a two-dimensional search space sequentially is not a reasonable solution, although it is simple and easy to implement on hardware or specifically on FPGA. This method is computationally expensive and also has a long delay. Another solution for this problem is to find a way to reduce one dimension, which means searching for one dimension simultaneously while sequentially searching for the other. While the serial search method is simply what we described above, which is a two-dimensional search, the idea of parallel searching of 1 of 2 dimensions leads us to 2 choices: Parallel Frequency Search (PFS) and Parallel Code Phase Search (PCS).

PARALLEL FREQUENCY SEARCH (PFS)

The name of this method has already shown us that this method parallelizes the search process in the frequency space. Looking at the figure 2.15, we can imagine simply that at a time, PFS will help us to search the entire column in parallel and each of them will be iterated through time by time in serial. For instance, all cells located on the column with code phase delay of 0 with all Doppler shift values from -10 kHz to +10 kHz will be searched simultaneously. Then, the next time, all cells located in the column at code phase delay of 1 with all Doppler shift values from -10 to +10 kHz will be searched simultaneously. This process is repeated until it reaches the final code phase delay bin. Clearly, instead of searching the whole 2-dimensional space as in the serial search method, which searches 82863 combinations, this method helps us to reduce significantly the number of combinations to search to only 1023 (this is

only for 1 PRN code or for 1 satellite). Figure 2.17 shows the block diagram for the PFS algorithm.

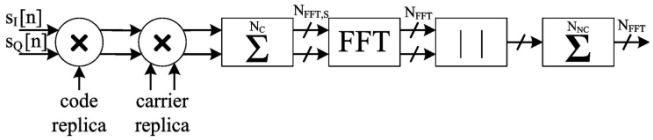


Figure 2.17: Parallel Frequency Search [5]

The operation of this algorithm is described below.

- **Step 1:** The received data is mixed with the locally generated PRN code to remove the PRN code spreading in the received data signal.
- **Step 2:** Data is continued to be mixed with the locally generated carrier to remove the carrier from the signal. **Note:** Only the original carrier frequency, which is 1575.42 MHz in case of L1 C/A signal is removed after this step. To be simple, this step helps to down-convert the signal from a high frequency back to base-band frequency. The Doppler shift frequency has not yet been processed!
- **Step 3:** The result of Step 2 is pushed to the Fast Fourier Transform core to find multiple Doppler frequency bins simultaneously.
- **Step 4:** Further processing of the FFT core result to find the matched Doppler shift frequency.
- **Step 5:** Repeat the algorithm but in Step 1, before mixing 2 signals, shifting the result by 1 code phase delay first.

Note: 5 steps are shown only for 1 PRN ID (or 1 satellite). This process should be executed for all 32 available satellites.

PARALLEL CODE PHASE SEARCH (PCS)

Parallel Code Phase Search is quite similar to Parallel Frequency Search except that it searches the code phase delay in parallel instead of the Doppler shift bins. Looking at the figure 2.15, we can imagine simply that at a time, PFS will help us to search the entire row in parallel and each of them will be iterated through time by time in serial. However, it is only quite similar in concept to the PFS algorithm. The way in which people implement the PCS method is different from the PFS method. This method needs 2 FFT cores and 1 IFFT core

to implement, and the purpose of the FFT core here is not similar to the one used in the PFS algorithm. In this method, people apply the convolution theorem that the convolution in time domain is equivalent to the multiplication in frequency domain. Based on the relationship between cross correlation and convolution, convolution will be utilized to compute the cross correlation between the received PRN code and the locally generated PRN code. Because the code phase delay space is now searched simultaneously, the number of combinations to be searched now reduces significantly to only 81 due to 81 Doppler frequency bins (this result only considers 1 PRN ID). Figure 2.18 shows the block diagram for the PCS algorithm.

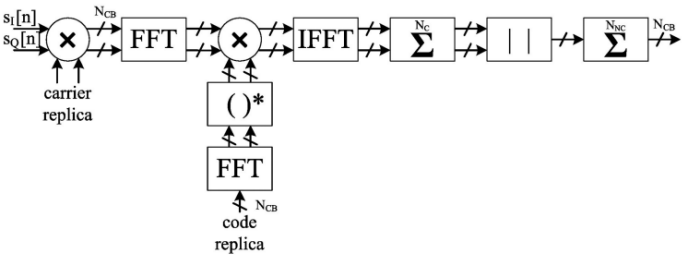


Figure 2.18: Parallel Code Phase Search block diagram [5]

The operation of this algorithm is described below.

- **Step 1:** Initialize the Doppler shift value of $\hat{f_D}$ = +10 kHz.
- **Step 2:** Shift the received signal back $\hat{f_D}$ Hz to remove the Doppler effect on the signal. It is to multiply the input signal with the exponential term:

$$e^{-j2\pi k\frac{\hat{f_D}}{f_s}} \tag{2.21}$$

  where, $\hat{f_D}$ is the estimated Doppler shift, k is the time index of the signal and k runs from 0 to K, K is the number of samples in a code period (in 1 ms), and f_s is the sampling rate.

- **Step 3:** Compute the FFT of the received signal after being wiped off carrier in step 2.
- **Step 4:** Compute the FFT of the locally generated PRN code of the satellite.
- **Step 5:** Compute the complex multiplication between two FFT output result from Step 3 and the complex conjugate of the FFT output result from Step 4.

- **Step 6:** Compute the IFFT of the result from Step 5.
- **Step 7:** Further process to observe the peak value in the FFT output. In 4096 output values from the IFFT core, the position at which the peak value occurs will be the estimated code phase delay.
- **Step 8:** Reduce the estimated Doppler shift value $\hat{f}_D$ by 250 Hz and repeat from step 2 to step 8 until the whole process is complete for all 81 Doppler shift bins from +10 kHz to -10 kHz with step of 250 Hz.

Note: 8 steps are shown only for 1 PRN ID (or 1 satellite). This process should be executed for all 32 available satellites.

2.3. GPS TRACKING PHASE

After getting the best PRN ID of a satellite to track together with the code phase delay and Doppler frequency needed from the acquisition phase, a tracking loop is needed to detect and demodulate the navigation message transmitted by the satellite. Due to the requirement to track 4 satellites at the same time, a typical GPS receiver will implement 4 trackers in parallel.

2.3.1. PHASE-LOCKED LOOP

Based on the Doppler frequency got from the acquisition phase, an oscillator will be set to the Doppler frequency to perform the carrier wipe-off operation. However, as the satellite continues to move, the Doppler frequency will be changed. Therefore, to keep track of the frequency and phase of the signal, a phase-locked loop is used. A phase-locked loop will keep track of the phase error of the signal and try to minimize it through the use of a discriminator, thus reducing the difference between the oscillator frequency (determined by the phase increment) and the Doppler shift.

The discriminator used in this case is called the Costas loop discriminator. Beside its usage as phase error discriminator, the Costas loop also acts as the BPSK demodulator for the GPS signal. The Costas phase discriminator is described by the formula[15]:

$$sin(2\phi_E) = I_P \times Q_P \tag{2.22}$$

where

- ϕ_E is the phase error
- I_P is the in-phase (real) component of the prompt correlator output

- Q_P is the quadrature (imaginary) component of the prompt correlator output

Since the frequency of the NCO is keep closely with the Doppler shift, the phase error would be very small so the equation 2.22 can be approximated as:

$$sin(2\phi_E) \approx 2\phi_E \Rightarrow \phi_E \approx \frac{(I_P \times Q_P)}{2} \tag{2.23}$$

This phase error can be sent into a loop filter, in this case a PI filter to smoothen the phase error output. The PI filter can be described by the formula:

$$\begin{cases} I[n+1] = I[n] + \phi_E[n] \times K_I \\ \phi_{E(\text{filtered})}[n] = \phi_E[n] \times K_P + I[n+1] \end{cases} \tag{2.24}$$

where

- $I[n], I[n+1]$: integral result
- $\phi_{E(filtered)}$: filtered phase error
- K_P: proportional gain
- K_P: integral gain

And thus the carrier NCO phase increment can be calculated by the formula:

$$\Delta\phi = \Delta\phi_0 + \phi_{E(\text{filtered})} \tag{2.25}$$

where

- $\Delta\phi$: phase increment
- $\Delta\phi_0$: initial phase increment

And the initial phase increment can be calculated based on the Doppler frequency acquired from acquisition process:

$$\Delta\phi_0 = f_{Doppler} \times \frac{2^{\text{phase\_width}}}{f_S} \tag{2.26}$$

Finally, for the NCO to output the sine and cosine signal, a LUT that already stored sine and cosine is generated and the final sine and cosine output are extracted by the phase accumulator value. This can be calculated as:

$$\begin{cases} \phi[n+1] = \phi[n] + \Delta\phi[n] \\ nco\_sine = LUT\_sine[\phi] \\ nco\_cosine = LUT\_cosine[\phi] \end{cases} \tag{2.27}$$

2.3.2. DELAY-LOCKED LOOP

Due to the difference between the Gold Code phase generated by the satellite and the local implementation, the code phase need to be tracked to be synchronized between the 2 locations. Therefore, 3 correlators are used with 3 different code generator, the early code runs at 0.5 sample before the prompt code, while the late code runs at 0.5 sample after the prompt code:

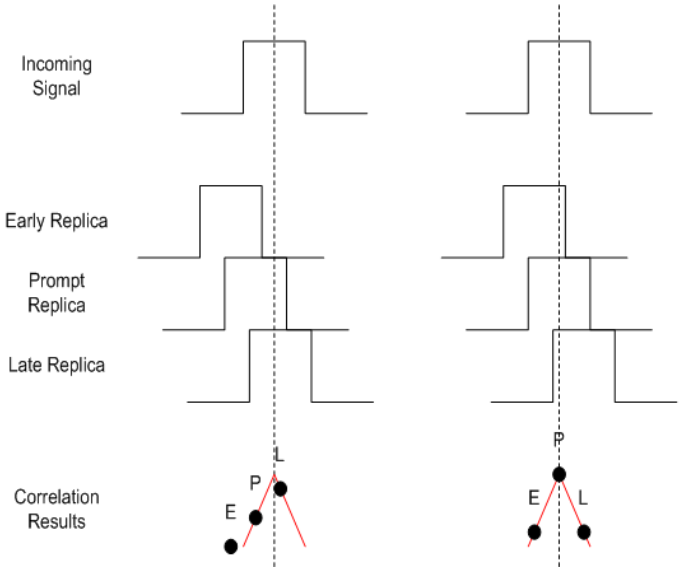


Figure 2.19: Early, late, prompt code replica tracking[6]

When the prompt code matched the code generated by the incoming signal, the power difference will be 0 while the power difference will be greatest when there is a mismatch between the prompt code and the code generated from the satellite. Therefore, our goal is to modify the code phase of the prompt code by tracking the code error between the incoming and local code replica and try to minimize it. To calculate the error of the delay-locked loop, a non coherent early minus late power discriminator is used and it can be defined by the formula[16]:

$$\text{code\_error} = \frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{2} \tag{2.28}$$

where

- I_E is the in-phase (real) component of the early correlator output
- Q_E is the quadrature (imaginary) component of the early correlator output

- I_L is the in-phase (real) component of the late correlator output
- Q_L is the quadrature (imaginary) component of the late correlator output

Like the PLL, the code phase error would also be sent through the PI filter to smoothen the result:

$$\begin{cases} I[n+1] = I[n] + \text{code\_error}[n] \times K_I \\ \text{code\_error}_{(\text{filtered})}[n] = \text{code\_error}[n] \times K_P + I[n+1] \end{cases} \tag{2.29}$$

where

- $\text{code\_error}_{(\text{filtered})}$: filtered phase error

And the code generator phase increment can be calculated by the formula:

$$\Delta\text{code\_phase} = \Delta\text{code\_phase}_0 + \text{code\_error}_{(\text{filtered})} \tag{2.30}$$

where

- Δcode_phase: code phase increment
- $\Delta\text{code\_phase}_0$: inital code phase increment

The initial code phase increment of the code generator can be calculated as:

$$\Delta\text{code\_phase}_0 = f_{code} \times \frac{2^{\text{phase\_width}}}{f_S} = 1023000 \times \frac{2^{\text{phase\_width}}}{f_S} \tag{2.31}$$

And the code phase is calculated by the formula:

$$\text{code\_phase}[n+1] = \text{code\_phase}[n] + \Delta\text{code\_phase}[n] \tag{2.32}$$

Finally, since the sampling rate is at 4096000 Hz compared to the code generator that operates at 1023000 Hz, the 0.5 sample difference at 1023000 Hz turned into $0.5 \times \frac{4096000}{1023000} = 2$ samples difference at 4096000 Hz. The early, prompt and late phase can be calculated as:

$$\begin{cases} \text{prompt\_phase} = \text{code\_phase} \\ \text{early\_phase} = (\text{prompt\_phase} + 4096 - 2) \ \% \ 4096 \\ \text{late\_phase} = (\text{prompt\_phase} + 2) \ \% \ 4096 \end{cases} \tag{2.33}$$

2.3.3. INTEGRATE AND DUMP

To decrease the noise, an integrate and dump system is used. The integration period is 1 ms or equivalent to 4096 samples in case the sampling rate is 4096000 Hz. After the integration is complete, the resulting accumulation of the output of all 3 correlators will be dumped for processing by the PLL and DLL.

2.3.4. BPSK DEMODULATION

The Costas loop used in the PLL will demodulate the BPSK signal used by GPS. Therefore, to extract the navigation message, we simply just need to take the sign of the Costas loop output, which is also the output of prompt correlator:

$$\text{navigation\_message} = sign(I_P) \tag{2.34}$$

2.4. HARDWARE

2.4.1. ULTRA96-V2 OVERVIEW

The Ultra96-V2 is a FPGA Single Board Computer designed for a wide range of embedded applications. With part Xilinx Zynq UltraScale+ MPSoC ZU3EG SBVA484, it integrates both a Processing System (PS) and Programmable Logic (PL), which is suitable for hardware-software co-design and acceleration tasks.

In terms of resources, the Ultra96-V2 offers the following features:

- **Zynq UltraScale+ MPSoC ZU3EG SBVA484**
  - **Processing System (PS):**
    - ◇ Quad-core ARM Cortex-A53 MPCore
    - ◇ Dual-core ARM Cortex-R5 real-time processors
    - ◇ 256KB On-Chip Memory with ECC
    - ◇ Mali-400 MP2 GPU
  - **Programmable Logic (PL):**
    - ◇ 154,350 System Logic Cells
    - ◇ 141,120 Flip-Flops
    - ◇ 70,560 LUTs
    - ◇ 216 Block RAMs (7.6 Mb)
    - ◇ 360 DSP slices
- **Memory and Connectivity:**
  - 2GB LPDDR4 RAM
  - 16GB MicroSD storage
  - Microchip Wi-Fi/Bluetooth module
- **I/O Interfaces:**
  - Mini DisplayPort
  - 1x USB 3.0 Type Micro-B, 2x USB 3.0 Type A

2.4.2. COMPARISON WITH OTHER FPGA BOARDS

Compared to other FPGA boards such as the Xilinx ZCU104 and the Digilent Arty Z7, the Ultra96-V2 is a good option balancing between cost and performance.

The ZCU104 is a costly modern board based on the Zynq UltraScale+ ZU7EV MPSoC, which offers a lot of programmable logic resources (LUTs and FFs), more DSP slices, and high-speed interfaces like PCIe and DisplayPort. As a result, this performance cost a huge amount of money and a larger physical size.

The Arty Z7 is a lower-cost board that uses the Zynq-7000 series (e.g., Z-7020). It is suitable for small FPGA projects with decent resources coming with a decent price. Therefore, it will not be suitable for our medium design.

The Ultra96-V2 is a perfect fit since it features the ZU3EG MPSoC with strong DSP and BRAM counts, integrated wireless connectivity, and support PS - PL programming. These features make it becomes the best solution for us to implement a project that require a large amount of DSP and BRAM needed.

2.4.3. RESOURCE UTILIZATION

As mentioned above, in terms of logic resources, the ZU3EG FPGA on the Ultra96-V2 contains a moderate number of Look-Up Tables (LUTs) and Flip-Flops (FFs). However, it is contains a huge amount of Digital Signal Processing (DSP) slices and Block RAMs (BRAMs), which allows it to effectively perform parallel computations and also manage large data buffers.

2.4.4. ZYNQ MPSOC PROCESSING SYSTEM

Our project requires the interaction between the PS and PL, and with the help of Zynq MPSoC Processing System. We can generate the input stream into our PL modules, also observe the output from our PL modules to UART port through PS.

2.5. RELATED WORK

2.5.1. GPS ACQUISITION PHASE

GPS acquisition phase plays an extremely important role in the working operation of the GPS receiver design. Therefore, there has been an extensive amount of research on this topic since GPS was allowed to be used for civilian purposes. Various important approaches have been taken before to optimize acquisition design in terms of performance, accuracy, and resource utilization. This subsection helps us to review some previous relevant work in the literature.

RESOURCE-EFFICIENT PARALLEL ACQUISITION ARCHITECTURES FOR MODERNIZED GNSS SIGNALS

In the Ph.D. Thesis of the author Jérôme LECLÈRE [7], he discussed the advantages and disadvantages of three types of GPS acquisition algorithm shown in Table 2.3: **Serial Search (SS)**, **Parallel Frequency Search (PFS)**, and **Parallel Code Phase Search (PCS)**.

	Advantages	Disadvantages
SS	• Low complexity • Low resource utilization • Hardware-friendly	• Low performance (long acquisition time)
PFS	• Significantly lower acquisition time in comparison with SS method	• High loss linked to the mismatch between the replica code chipping rate and the received code chipping rate
PCS	• Significantly lower acquisition time in comparison with SS method	• High complexity • Limitation on sampling frequency • High resource utilization

Table 2.3: Advantages and disadvantages of three acquisition methods

Moreover, he also discussed and compared three types of implementation of GNSS acquisition very clearly in detail in Section 3 of chapter 1: **Comparison of GNSS signals acquisition architectures on FPGAs**. Because resource utilization among three types of implementation is different much from each other, the author considered a specific area and then tried to duplicate each type to fit that area to make the comparison fair. By doing that, the three implementations require the same amount of resources, so that we can compare them in a fair manner. The author also provides a formula to compute the total execution time for three acquisition methods with different configurations that is:

$$T_{E,SS} = \frac{d T_T N_{CB} N_{FB}}{G_{FPGA} N_{B,SS}} \tag{2.35}$$

$$T_{E,PFS} = \frac{d T_T N_{CB}}{G_{FPGA} N_{B,PFS}} \tag{2.36}$$

$$T_{E,PCS} = \frac{d T_T N_{FB}}{G_{FPGA} N_{B,PCS}} \tag{2.37}$$

where:

- $T_{E,SS}$, $T_{E,PFS}$, $T_{E,PCS}$ are time to explore the whole two-dimensional search space for one satellite.

- d is the parameter to distinguish between two cases of the data bit transition process, d = 1 means to ignore and d = 2 means to use the alternate half-bit method to process.

- T_T is the total integration time. Usually, PFS and PCS methods only need 1 millisecond of sampling data to process and can give an exact solution. However, this only works if the GPS signal is strong such as in an outdoor environment and does not work in a reverse situation such as indoor environment where there is much noise that makes the signal weaker. Integration only means that the acquisition will try to search over many periods of the PRN code (in multiple milliseconds) instead of only a 1-millisecond period search.

- N_{CB} is the number of code phase delay bins. This value should be 1023 code bins in the original code chipping rate but will vary due to the sampling rate of the GPS receiver. For example, if the sampling rate is 4.096 MHz, then N_{CB} = 4096 code bins.

- N_{FB} is the number of Doppler shift frequency bins. There are no limitations or rules applied on the range of the number of Doppler-shift frequency bins. The Doppler frequency value ranges from +10 kHz to -10 kHz for normal civilian use purposes, so the number of bins is the number of Doppler shift frequencies the GPS receiver will search on this range. Depending on the designer, it can be any value for the search step. The higher the value of N_{FB}, the better the Doppler shift estimation result is. However, a higher value of N_{FB} implies a longer search time and even higher utilization of resources. This is a trade-off that we need to consider later on.

- G_{FPGA} is the ratio between the clock frequency of the FPGA and the sampling frequency: $G_{FPGA} = \frac{f_{FPGA}}{f_s}$.

- $N_{B,SS}$, $N_{B,PFS}$, and $N_{B,PCS}$ are the number of branches in the design. As we mentioned earlier, the design of each method will be duplicated so that it will cover the given amount of resources to make the comparison fair. The branch here means the number of repetitions of each method design to cover the given amount of resources.

Sensitivity	−150 dBm
T_C	10 ms
N_C	40 960
N_{NC}	40
T_T	400 ms

Figure 2.20: Sensitivity parameters [7]

Case	Assisted	Stand-alone
Frequency search space	1360 Hz	11 020 Hz
Frequency step (δ_f)	50 Hz	
N_{FB}	29	221
Code step (δ_C)	1 sample	
N_{CB}	4096 samples	

Figure 2.21: Search space parameters (adpated from [7])

Some configuration parameters are listed in the figures 2.20 and 2.21. Applying these parameters to three acquisition methods, the author observed the results as shown in figure 2.22

		Low-cost FPGA EP3C120		High-end FPGA EP3SE260	
		Assisted case	Stand-alone case	Assisted case	Stand-alone case
Number of branches	$N_{B,SS}$	971		2911	
	$N_{B,PCS}$	2		8	
	N_{B,PCS^*}	-	4	-	11
	$N_{B,PFS}$	1095		3385	
Parallelization	P_{SS}	971		2911	
	P_{PCS}	8192		32 768	
	P_{PCS^*}	-	16 384	-	45 056
	P_{PFS}	12 045		37 235	
Time to explore the search space (ms)	$T_{E,SS}$	4078	31 075	680.1	5183
	$T_{E,PCS}$	483.3	3683	60.42	460.4
	T_{E,PCS^*}	-	1842	-	334.8
	$T_{E,PFS}$	328.7	2505	53.17	405.2

Figure 2.22: Results and performance of three method implementations [7]

The author used the term "assisted case" to describe a "hot start" acquisition, which means that the GPS receiver has prior information about the available satellites in view (from the Internet, previous working time, etc.) so that the receiver in this situation does not need to do a full two-dimensional search. He also experimented with the "cold start" acquisition (stand-alone case), which means the GPS receiver does not have any information about the available satellites so that it needs to do a full two-dimensional search.

Looking at the part of **"Time to explore the search space (ms)"**, we can observe that the SS method has the worst performance. Even in the case of the "assisted case", the time to search the entire two-dimensional space of this method is still more than ten times higher than the PFS, PCS, and PCS* methods (PCS* is a modified version of the PCS method).

Among three left implementations, we can see that the PCS* method has the best performance, then the PFS and PCS methods are respectively in the second and the third place. In fact, the PCS method is frequently used in the GPS acquisition design due to its high performance instead of PFS. The reason

why the PFS algorithm gives better performance in the author's comparison is because he has a large enough amount of resources to be able to duplicate the design many times. Additionally, the design of PFS will have lower resource utilization compared with PCS, so the duplication of PFS will be much higher than that of PCS and, therefore, leading to process more combinations at a time than PCS can. Looking at the first row **"Number of branches"** in the figure 2.22, we can see that $N_{B,PFS}$ is much larger than $N_{B,PCS}$. However, in our case, we are implementing a GPS Receiver on the Ultra96 v2.0 FPGA board, and with the resource configuration of our board, it is not reasonable for us to duplicate the design of the GPS acquisition. We also need to care about the resource for the tracking phase later on.

Therefore, in case of our design, with only one instance of GPS acquisition, the PCS method will have a better performance than the PFS method.

To make everything clearer, we will apply the formula given by the author to apply to our case and compare the performance between PCS and PFS methods. In our capstone project, we only simply search for 1 millisecond of the PRN code, so $T_T = 1$ and d is also considered 1 (ignore the data bit transition). As we mentioned earlier, we do not have enough resources to duplicate the acquisition module instances, so the number of branches in our case is $N_{B,PFS} = N_{B,PCS} = 1$.

Applying the value of T_T, d, $N_{B,PFS}$, and $N_{B,PCS}$ to the equation 2.36 and 2.37, they are simplified to:

$$T_{E,PFS} = \frac{N_{CB}}{G_{FPGA}} \tag{2.38}$$

$$T_{E,PCS} = \frac{N_{FB}}{G_{FPGA}} \tag{2.39}$$

To compare $T_{E,PFS}$ and $T_{E,PCS}$, now we simply need to compare N_{CB} and N_{FB}. Obviously, the smallest possible value of N_{CB} is 1023 code bins, while N_{FB} is not limited but usually ranges from tens to hundreds. In the report [7], the author uses $N_{FB} = 221$ Doppler shift bins, and in our capstone project, we use $N_{FB} = 81$ Doppler shift bins. He and we both use $N_{CB} = 4096$ code bins. In total, the PCS method gives us better performance with a smaller acquisition search time compared to the PFS method in our case:

$$\frac{T_{E,PFS}}{T_{E,PCS}} = \frac{N_{CB}}{N_{FB}} = \frac{4096}{81} \tag{2.40}$$

Moreover, we can easily see that the system of the PFS method will be really difficult to scale up further. It is easy for N_{CB} to increase significantly due to the increase in the sampling rate or the new PRN code types. For instance, the L5 PRN code chipping rate is 10.23 MHz instead of 1.023 MHz like the L1 C/A code in our system. Therefore, if the author wants to change to use the new GPS signal such as L5, the acquisition time of the PFS method will grow ten times compared to that of the L1 GPS signal. Meanwhile, N_{FB} will not be increased to a high value because the responsibility of the GPS acquisition phase is only to estimate the Doppler shift and code phase delay values, not to achieve the accurate value. In case of the Ph.D thesis of the author, he used $N_{FB} = 221$ Doppler shift bins, which can already be considered as a high value for N_{FB}. This also leads to the fact that the scalability of the PFS method is not high compared to that of the PCS method.

Until now, it has been clear that the PCS method outperforms the PFS method in acquisition time in case of not duplicating the design. In terms of the modified version of the PCS method (PCS* method), the reason why it is better than the PCS method is similar to the reason why the PFS method is better than the PCS method in the comparison of the author. In our case, we decide to use the PCS method to implement our GPS acquisition design.

FPGA-BASED REAL-TIME GPS RECEIVER

A GPS receiver fully implemented on FPGA has been done by the author Adam M. Shapiro [8]. However, now we only discuss the GPS acquisition phase design in his system. In his system, he used the conventional GPS acquisition method, which is the Serial Search (SS) method. As we mentioned in the table 2.3, the SS method is very simple to implement and hardware-friendly due to its low resource utilization. However, its performance is really bad with a very long acquisition time.

The SS method has a special mechanism that allows it to complete the acquisition phase sooner than expected. That is, instead of sequentially searching all possible combinations of code bins and Doppler bins, the SS method execution will stop whenever it finds a combination that has a peak correlation value exceeding a pre-defined threshold value. The author uses the term "early termination" to describe this mechanism. However, early termination can only happen in some "lucky situations" when the correct cell to be searched is located somewhere near the beginning of the serial search. The worst case occurs when the correct combination between code bins and Doppler bins is located in the final cell to be searched in the two-dimensional search space.

Adam M. Shapiro estimated in his report that the time for acquisition of a single satellite PRN code can be up to approximately 2.5 minutes when an early termination does not occur. If we have enough resources on FPGA, we can easily duplicate the design of the SS acquisition method to allow searching multiple satellites simultaneously to speed up the system of the SS method, and this is usually the case because the resource utilization of this method is very low. Taking the 2.5 minutes acquisition estimation time from the author, we can estimate the time to complete searching the entire GPS satellite constellation in the general case (suppose that the early termination does not occur). Suppose $N_{B,SS}$ is the number of duplications of the SS method and T_A is the time to complete the acquisition for 32 satellites, we have this formula:

$$T_A = 2.5 \times \frac{32}{N_{B,SS}} \text{ minutes} \tag{2.41}$$

From equation 2.41, it is easy to find out the worst and the best cases of acquisition time of the SS method design.

- **Best case:** occurs when $N_{B,SS}$ is maximum, which is 32. In this case, T_A = 2.5 minutes.
- **Worst case:** occurs when $N_{B,SS}$ is minimum, which is 1. In this case, T_A = 80 minutes.

As you can see, even in the best case, it costs approximately 2 minutes to complete the acquisition search using the SS method. Absolutely, this will not be a good choice if we want to have a high performance GPS receiver.

A HIGH PRECISION ACQUISITION ALGORITHM OF GPS BASED ON PARALLEL FREQUENCY SEARCH

In this article [17], Xiang Gao proposed a new algorithm to improve the accuracy of GPS acquisition based on the PFS method. Moreover, he also proposed a new design that combines the PFS and PCS method together to make the system better instead of only using PFS for the GPS acquisition design. His idea is to let the acquisition design run twice instead of once.

- **First run:** The acquisition design will search for 41 Doppler bins in the range of -10 to + 10 kHz with the search step being 500 Hz to estimate roughly the Doppler frequency.
- **Second run:** After receiving the results of the first run, the author tried to create a set of Doppler frequencies around the estimated Doppler frequency with step of 250 Hz instead of 500 Hz in the first run to achieve

a better accuracy. In particular, he would create a set of frequencies like this $[f_1\ f_2\ f_3\ f_4\ f_5\ f_6\ f_7] = [f-750\ f-500\ f-250\ f\ f+250\ f+500\ f+750]$ and run the acquisition design for the second time using this set.

2

To improve the performance of the design, the author proposes that instead of using the PFS methods twice the operation of acquisition, he uses the PCS method for the first time to speed up the acquisition time and keeps the PFS method for the second time. This proposal is quite reasonable and logical, but it also increases the complexity of the design. We need to be concerned with data control and management for this case. Additionally, the PFS method as we mentioned earlier can not handle the influence of the Doppler effect precisely, which makes it easy to have big errors in the result.

2.5.2. GPS TRACKING PHASE

FPGA-BASED REAL-TIME GPS RECEIVER

In [8], Mr. Adam M. Shapiro implemented a GPS receiver on FPGA, but in this case we will focus on his tracking loop and tracking channel implementation. His design followed standard tracker design, with carrier adding and code correlation. However, his tracking filter included an FLL beside PLL and DLL.

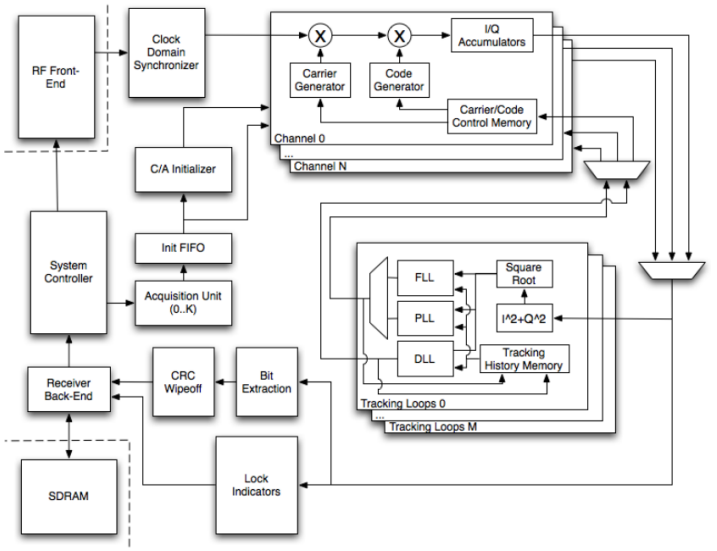


Figure 2.23: Shapiro GPS receiver implementation[8]

In Shapiro's implementation, the DLL phase increment is calculated as[8]:

$$\Delta\phi_{C/A,k+1} = K\left(\frac{(2-\text{chips}_{\text{EML}})2^{N_{C/A}}}{2\,\text{Amplitude}}\right)\frac{\left\|\overline{IQ}_{\text{early}}\right\| - \left\|\overline{IQ}_{\text{late}}\right\|}{\left\|\overline{IQ}_{\text{early}}\right\| + \left\|\overline{IQ}_{\text{late}}\right\|} \tag{2.42}$$

In addition, Mr. Shapiro also implemented a frequency-locked loop with the rotation angle calculated by:

$$\Delta\theta \approx \frac{Q_k I_{k-1} - I_k Q_{k-1}}{\left\|\overline{IQ}_k\right\|\left\|\overline{IQ}_{k-1}\right\|} \tag{2.43}$$

However, Mr. Shapiro only tracks the incoming signal using the DLL and the FLL instead of the PLL, with the PLL planned to be implemented in the future. The purpose of the PLL of tracking the frequency and phase of the incoming signal is already accomplished by the FLL. However, while an FLL can have some advantages over a PLL like better locking time, good again sudden frequency error caused by movement on the receiver side, it is not good enough to accurately track the phase of the incoming signal from the satellite compared to the PLL. So for a highly accurate receiver used on fast moving platform, a tracking loop will actually consist of FLL, PLL and DLL like originally proposed by Mr. Shapiro.

3

PROPOSED DESIGN ARCHITECTURE

Based on the background theory given in chapter 2, we will introduce our proposed GPS receiver system design architecture in this chapter. Besides showing the top design architecture, we also provide the proposed overview block diagram for the GPS acquisition design and GPS tracking design.

3.1. OVERALL SYSTEM

3.1.1. SYSTEM ARCHITECTURE

The overall architecture of our system design is illustrated as shown in Fig 3.1:

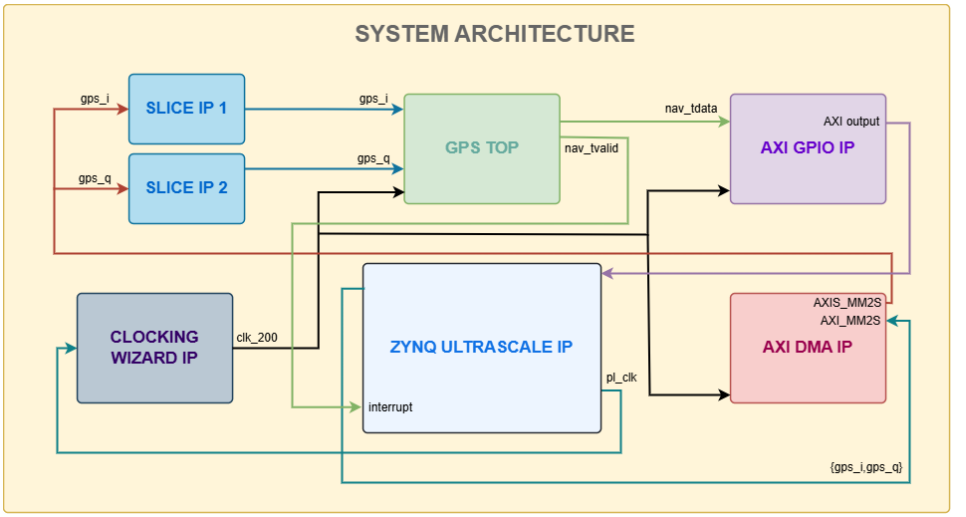


Figure 3.1: Proposed System Architecture

The input data, which is raw GPS signal (can be downloaded online or generated manually by [18]), is first loaded into the microSD card on the Zynq processing system. We then read the data from the SD card and transfer it to our GPS TOP module through the DMA IP. Before the data reaches our PL module, we use two Slice IPs to separate the raw GPS signal into I/Q components respectively (since the original raw GPS data is structured as 16-bit I followed by 16-bit Q repeatedly, we concatenate them into 32-bit words before storing them in our buffer. This is why we need to re-separate each part again using the Slice IPs). Besides, the Clocking Wizard IP is used to generate a frequency of $200MHz$ from the PS Clock.

On the other hand, the output valid signal from our GPS TOP module will also trigger the interrupt that cause the PS to start reading the output data from our GPS TOP module through the AXI GPIO IP and begin the searching preamble operation.

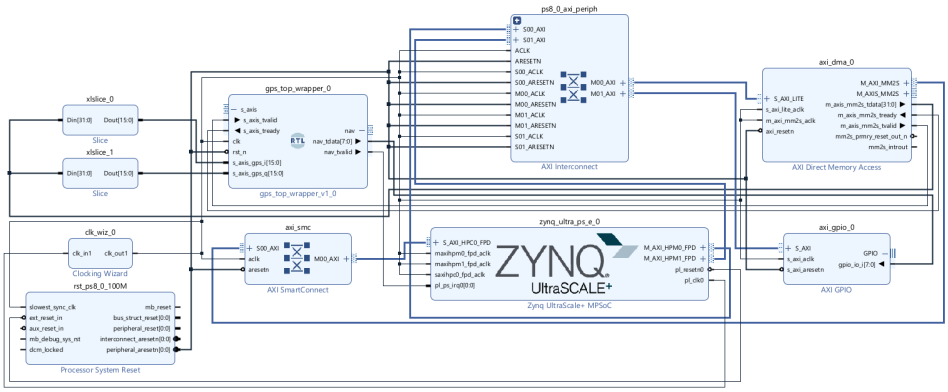


Figure 3.2: Detailed Block Diagram in Vivado

3.1.2. GPS TOP ARCHITECTURE

The overall architecture of our PL design is illustrated as shown in Fig 3.3

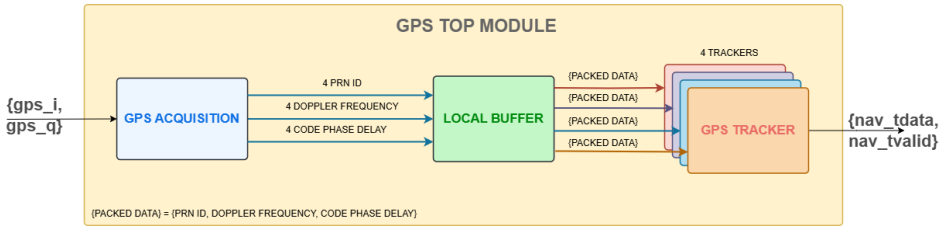


Figure 3.3: GPS Top Block Diagram

As mentioned above, the input raw GPS signal which are 16-bit I/Q quadrature signals are main inputs for the GPS Acquisition block. The output of Acquisition will be based on the top four cross correlation result between local C/A Code and received signal:

- 4 PRN IDs.
- 4 Doppler Shift Frequency.
- 4 Code Phase Delay.

These four values of each kind will be stored in three local registers and each pair of {PRN ID, Doppler Shift, Code Phase Delay} is sent to the corresponding GPS tracker module to initiate the tracking process, which eventually produces the navigation bitstream for further decoding of satellite parameters such as position, velocity, and angle, or for determining the user's position and related data.

3.2. GPS ACQUISITION PHASE

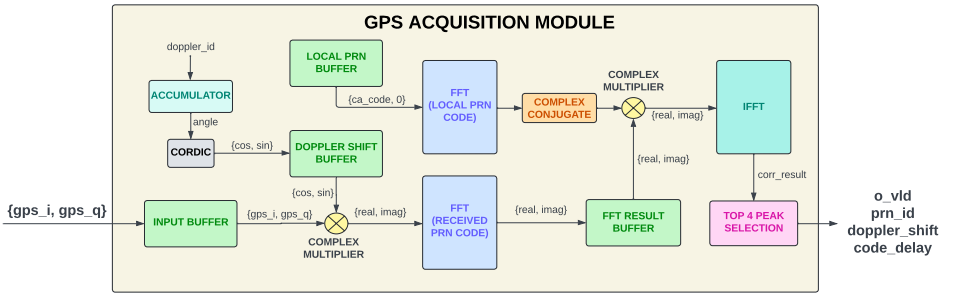


Figure 3.4: GPS acquisition design overview diagram

Figure 3.4 presents an overview block diagram for our proposed GPS acquisition design architecture. As we mentioned in the section **Related Work** 2.5.1, we decided to choose the Parallel Code Phase Search method to build our GPS acquisition module, so the design strictly follows the structure of the PCS method shown in figure 2.18 with three key components: two FFT module instances and one IFFT module instance. We will present the functionality of each block in the overview diagram 3.4. The **COMPLEX CONJUGATE** block is simply to reverse the sign of the FFT result for the locally generated PRN code.

3.2.1. ACCUMULATOR AND CORDIC BLOCKS

The formula 2.5 is in exponential form. If we expand it using Euler's formula, we will have equation 3.1.

$$cos(2\pi n\frac{f_d}{f_s}) - jsin(2\pi n\frac{f_d}{f_s}) \tag{3.1}$$

We need to compute two things: the angle $2\pi n\frac{f_d}{f_s}$ and the value of the cosine and sine functions of the angle. The **ACCUMULATOR** block is responsible for computing the angle and then transferring the result to the **CORDIC** block to compute the cosine and sine functions of the angle.

We researched and developed the CORDIC algorithm based on the knowledge and explanation in the article [19] and the blog [20]. The word "CORDIC" stands for "**C**oordinate **R**otation **D**igital **C**omputer". This algorithm is commonly used in many applications, such as trigonometric functions, multiplication, division, data-type conversion, square root, and logarithms. The strong point of CORDIC is that to implement this algorithm, we only need to use shifts and additions, which makes it really suitable for hardware implementation.

Moreover, the CORDIC algorithm utilizes the iterative method, which refined the solution over iterations, and this improves the result. The iterative rotation is expressed as follows.

$$x_{i+1} = k_i[x_i - d_i y_i 2^{-i}] \tag{3.2}$$

$$y_{i+1} = k_i[y_i + d_i x_i 2^{-i}] \tag{3.3}$$

$$z_{i+1} = z_i - d_i tan^{-1}(2^{-i}) \tag{3.4}$$

3.2.2. FFT AND IFFT BLOCKS

In the PCS method, we utilize the convolution theorem by combining two FFT blocks to convert the received and locally generated PRN codes into the frequency domain. Then, the IFFT block is used to convert back the result to the time domain. We use the sampling frequency $f_s = 4.096$ MHz, which means that we will have 4096 samples to process in a single period of the PRN code. Therefore, two 4096-point FFTs together with one 4096-point IFFT will be used here.

3.2.3. COMPLEX MULTIPLIER BLOCKS

A complex multiplication is more complicated than a real multiplication. Suppose there are two complex numbers $x = a + bj$ and $y = c + dj$, then the complex multiplication between them is:

$$x \times y = (a \times c - b \times d) + j(a \times d + b \times c) \tag{3.5}$$

From equation 3.5, we can see that a complex multiplication consists of four real multiplications and two real additions.

3.2.4. BUFFER BLOCKS

As the name implies, these blocks are actually BRAMs.

- **INPUT BUFFER:** The GPS acquisition needs millions of cycles to complete its working operation, not immediately in one or two cycles. Therefore, the **INPUT BUFFER** is used to buffer the input to be reused for further processes.

- **DOPPLER SHIFT BUFFER:** This buffer is used to store the result of the combination between the **ACCUMULATOR** and the **CORDIC** blocks.

- **LOCAL PRN BUFFER:** In the theoretical working operation, the GPS receiver needs to generate a local replica PRN code for each satellite. Moreover, the replica PRN code needs to be re-sampled at the sampling frequency f_s = 4.096 MHz to match the length of the received PRN code. This will make the design more complicated and more difficult to control or debug later on. Therefore, we choose the option of precomputing the locally generated PRN code at sampling frequency f_S = 4.096 MHz and then storing the result in the **LOCAL PRN BUFFER**.

- **FFT RESULT BUFFER:** the GPS acquisition needs to search for all satellites, which means that the FFT for the local PRN code must run 32 times corresponding to 32 satellites. The **FFT RESULT BUFFER** is used to store the FFT of the received PRN code for reuse.

3.2.5. TOP 4 PEAK SELECTION BLOCK

The target of the acquisition phase is to determine at least four available satellites and their roughly estimated Doppler shift and code phase delay values. This block is used to compare the correlation result sent from the IFFT core. Then, after the entire acquisition search is complete, this block will pick four top satellites that have the highest correlation results. PRN ID, Doppler shift estimation, and code phase delay estimation corresponding to the top four satellites are then sent to the GPS tracking design.

3.3. GPS TRACKING PHASE

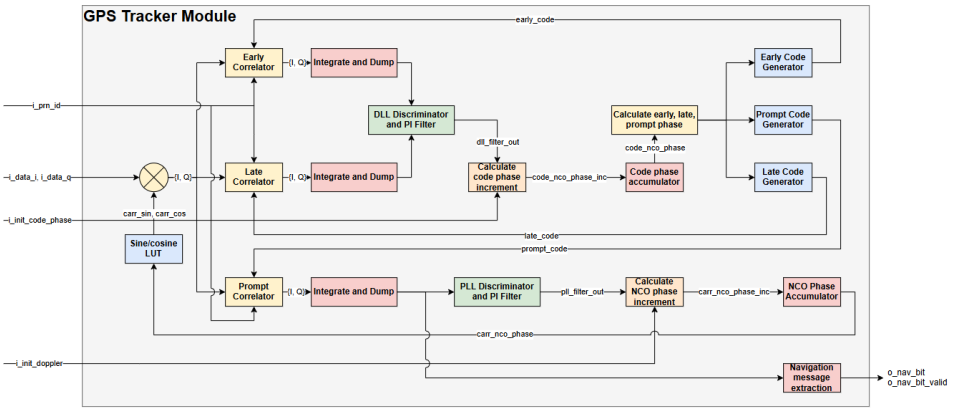


Figure 3.5: GPS tracker design overview diagram

Figure 3.5 presents an overview block diagram for our proposed architecture of a single GPS tracker design. The architecture closely followed the mathematical formula described in chapter 2. The **Sine/cosine LUT** block store the pre-generated sine and cosine value that will be read with a phase input. This output will then multiply with the input signal using complex multiplication to get the resulting complex number. The **Early, Late, Prompt Correlator** will calculate the correlation value of the input signal with the code generated by the **Early, Late, Prompt Generator**. The resulting result will then be accumulated in the **Integrate and Dump** block before all the result is decimated down to 1 kHz sample rate (4096 samples accumulation). This result is then used to calculate code error by the **DLL Discriminator** and the phase error by the **PLL Discriminator** before filtering by **PI Filter**. Then, the phase can be calculated in a separate block, the result is then accumulated and send as feedback to **Sine/cosine LUT** or to calculate the early, late, prompt phase and feed back into **Early, Late, Prompt Code Generators**. Then, the sign of the in-phase component of the prompt correlator integrate and dump output is then sampled once every 20 samples.

4

DESIGN IMPLEMENTATION

Based on the overview block diagram we showed in chapter 3, we will discuss more deeply the detailed implementation of the GPS acquisition and tracking designs in this chapter.

4.1. GPS ACQUISITION PHASE

4.1.1. ACCUMULATOR AND CORDIC BLOCKS

ACCUMULATOR

The **ACCUMULATOR** block is responsible for computing the angle and then transferring the result to the **CORDIC** block. If we compute the angle directly, it costs us additional hardware resources, such as DSP blocks for multiplications. We observe that instead of direct computation, we can apply the accumulator design here. Within the angle $2\pi n\frac{f_d}{f_s}$, there are only 2 variables which are f_d and n:

- f_d is the estimated Doppler shift value and it varies from -10 to +10 kHz with step 250 Hz.
- n is the code phase delay bin. We use the sampling frequency $f_s = 4.096$ MHz, which means that in a single 1-millisecond PRN code period, there are 4096 samples. Therefore, n varies from 0 to 4095.

Our solution here is to precompute the value of $2\pi\frac{f_d}{f_s}$ for each estimated Doppler shift frequency bin f_d. Then, to compute the current angle, we take the $\Delta = 2\pi\frac{f_d}{f_s}$ and add it to the previous angle. Because for every estimated Doppler shift value, the angle $2\pi n\frac{f_d}{f_s}$ only changes due to the change of the variable n,

and this variable changes step by step with step of 1. Let us take an example to make it easy to understand our solution in Table 4.1.

n	0	1	2	3	...	4095
Angle	0	$2\pi\frac{f_d}{f_s}$	$4\pi\frac{f_d}{f_s}$	$6\pi\frac{f_d}{f_s}$	...	$8190\pi\frac{f_d}{f_s}$

Table 4.1: Angles corresponding to different values of n from 0 to 4095

You can see that $angle[n] = angle[n-1] + 2\pi\frac{f_d}{f_s}$. For example, $angle[3] = 6\pi\frac{f_d}{f_s} = 4\pi\frac{f_d}{f_s} + 2\pi\frac{f_d}{f_s} = angle[2] + 2\pi\frac{f_d}{f_s}$, and this is true for every other angle except the first angle. In the first position, it does not have the previous position, so we need to initialize a value for it here. At this position, n is equal to 0, so the angle is 0 no matter what the value of the estimated Doppler shift value f_d is. Based on the idea, the **ACCUMULATOR** block is designed as shown in the detailed block diagram 4.1.

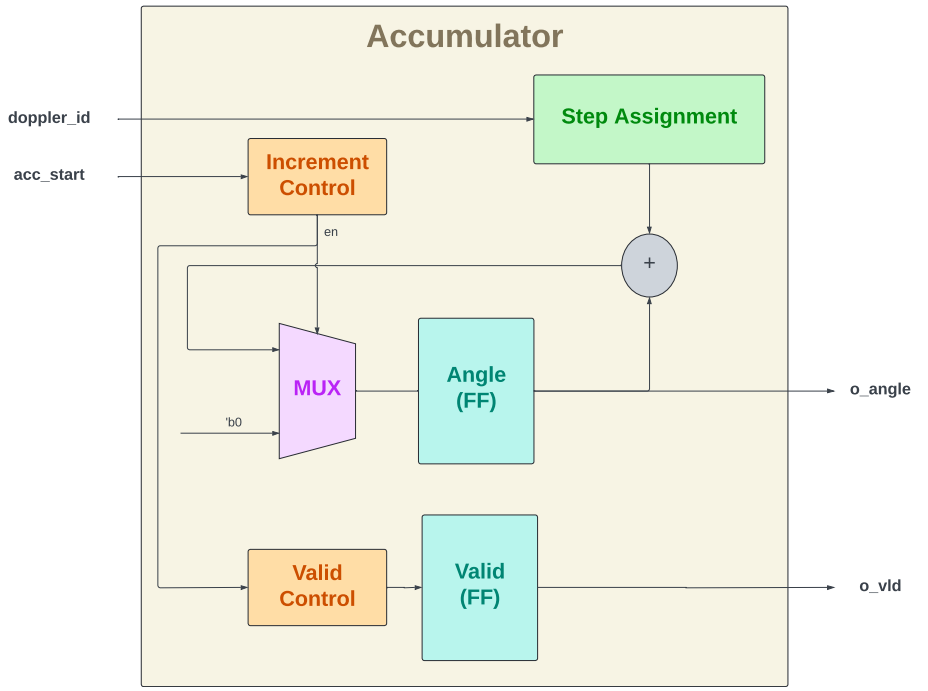


Figure 4.1: Accumulator detailed block diagram

Whenever the GPS acquisition top module needs the accumulator to work, it only needs to send 1 acc_start signal. The accumulator then will operate and

send out continuously computed angles until it reaches the final angle (angle at position 4095). The description for each block in figure 4.1 is as follows.

- **Increment Control:** this block waits for the acc_start input signal to be triggered, then it controls the working operation of the accumulator block by controlling the **"en"** signal to allow the new angle to be updated and to allow the valid control block to start working.

- **Valid Control:** this block waits for the **"en"** signal from the **Increment Control** block to start controlling the **valid** flip flops.

- **Step Assignment:** This block contains the step $\Delta = 2\pi\frac{f_d}{f_s}$ for all values of f_d. We have 81 Doppler bins f_d in the range of -10 to +10 kHz (Doppler bin step $\Delta_{Doppler}$ = 250 Hz), so theoretically there will be 81 Δ values to be stored in this block. However, we utilize the symmetric characteristic of the cosine and sine functions to reduce the amount of Δ values that need to be stored to only 40. We have $cos(\phi) = cos(-\phi)$ and $sin(\phi) = -sin(-\phi)$. Therefore, for a couple of Doppler bins that have opposite signs but the same magnitude, we can compute the Δ value of one using the other. For instance, consider the Doppler bins of +7500 Hz and -7500 Hz, their corresponding values Δ are respectively $2\pi\frac{7500}{f_s}$ and $2\pi\frac{-7500}{f_s}$. Clearly, they are only different from each other in sign but similar to each other in magnitude. Our choice is to store 40 Δ values for Doppler bins from +250 to +10000 Hz. The Doppler bin of 0 Hz is not stored because its Δ value is also 0. Because there are only 40 Δ values to be stored, we do not use BRAM here. Instead, we store them as an array with LUTs to make the design simple and save resources for BRAM. The **Step Assignment** block receives the **doppler_id** from input to send out the corresponding Δ at the position **doppler_id** of the array. The output of this block is then used to add with the value of the **Angle** flip flop to compute the next angle value.

- **Angle and Valid:** They are flip flops to store the resulting angles and the valid signal. The input of the **Angle** flip flop is chosen using a multiplexer. As we mentioned earlier, for $n = 0$ (the first position), we need to initialize the value of 0 for the angle. The multiplexer here is used to initialize the value of 0 or assign a new value for the angle.

CORDIC

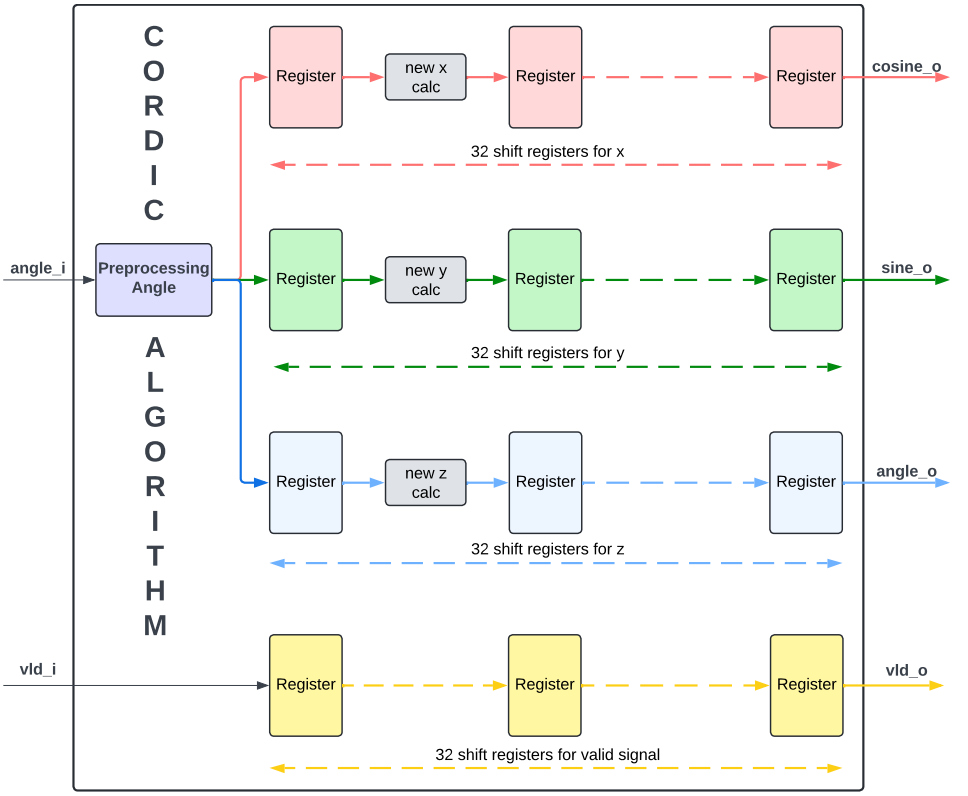


Figure 4.2: CORDIC detailed block diagram

In figure 4.2, except for the first register, the input of all other registers will be calculated by the block "new x/y/z calc", which calculates the formula 3.2, 3.3, and 3.4 as shown above. The first register will store the data received from the **Preprocessing Angle** block. This block is used to preprocess the angle because the CORDIC algorithm can only operate in the range of $\frac{-\pi}{2}$ to $\frac{\pi}{2}$ (in quadrants I and IV). Every angle residing in quadrants II and III must be preprocessed to be in quadrants I and IV before computing its cosine and sine. Because we are using fixed-point representation, the precision of this algorithm gets better and better as long as we use more bits in the input and output signals of this module. We seeked for support from [21] and in fact we also needed to modify the source code at many parts to adapt to our system, we do not just referenece and use that code 100%.

4.1.2. FFT AND IFFT BLOCKS

Initially, we tend to use the available FFT IP core from Xilinx, or the available source on the Internet. However, the capstone project is also a chance for us to learn more about design. Instead of using the available IP core, we want to build everything from scratch to learn and also to achieve better design control. It is also a great opportunity for us to learn about pipeline structures while we try to improve our design's performance. In the process of designing the FFT and IFFT core designs, we also sought some help from the web blog **An Open Source Pipelined FFT Generator** [22] written by Dan Gisselquist. During the design of the FFT core, we faced some challenges with memory management and we tried to find the solution from the blogs and source codes of Dan Gisselquist. Until now, some parts of our FFT core design have been similar to the one from this author, but not everything. Our logic control is almost different from that in his design. Moreover, we also deal with some problems such as bit growth and DSP usage differently from the author.

Figure 4.3 shows the overview block diagram of our FFT core design. We are using a 4096-point FFT core, so there are 12 stages in the FFT design. About the 4096-point IFFT core, the design is the same, but the twiddle factors will be reversed in its imaginary part's sign.

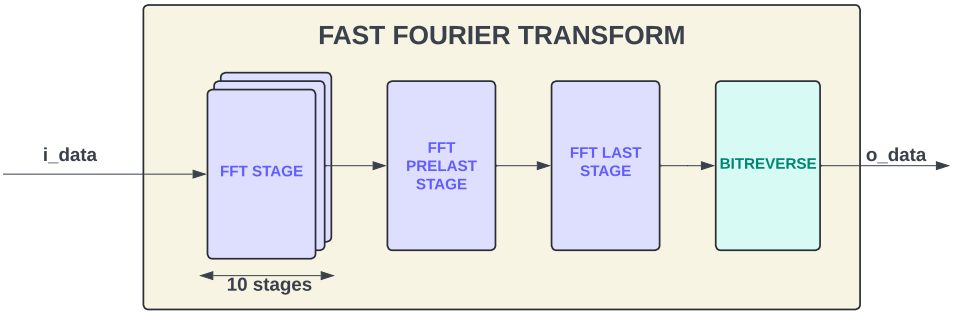


Figure 4.3: Fast Fourier Transform overview block diagram

For every stage of the FFT core, we need to compute the complex additions and especially the complex multiplication that consumes most hardware resources (BRAM and DSP blocks). In the blog [22], the author pointed out that two last stages of the design can be computed simply by additions and subtractions instead of expensive multiplications. We verified the base theory behind this trick that he had written in the blog, and we decide to follow this idea to save hardware resources.

FFT STAGE

This block is for stages from 1 to 10 in our 4096-point FFT core design. The detailed block diagram for the FFT stage is shown in Figure 4.4.

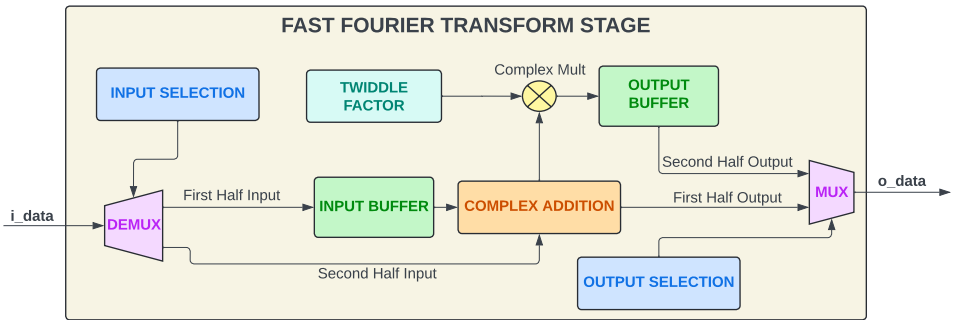


Figure 4.4: FFT stage detailed block diagram

This is the Decimation-in-Frequency (DIF) FFT core design. Because we divide the big FFT core into smaller FFT core (N-point FFT into N/2-point FFT, N/2-point FFT into N/4-point FFT, etc.), the size of every stage will be half of the previous stage. For every stage, we do not wait for the full input data to start working. Instead, we need to wait for a half valid input. You can see that the input **i_data** is not directly used in figure 4.4.

The **INPUT SELECTION** block is used to classify the input. For the first half of the input data, they are stored in the **INPUT BUFFER** to wait for the second half of the input data to arrive to begin computing the output. When the first half input is fully received, the fft stage prepares to compute the output.

For every input data in the second half input, it is added or subtracted with the first half input stored in the **INPUT BUFFER** corresponding to the current index using the complex addition block. For example, the first stage in our FFT core will store 2048 first input data indexing from 0 to 2047. When the input data that has the index of 2048 arrives, it will be added with input data at index 0 stored in the **INPUT BUFFER** and the process repeats.

Looking at the first stage of figure 2.14 for an 8-point FFT, for instance. Stage 1 will store the first half input (x[0], x[1], x[2], and x[3]) into the **INPUT BUFFER**. Then, when x[4] arrives, x[0] will be read from the buffer to compute with x[4]. The combination of x[0] and x[4] gives us two output results as 4.1 and 4.2. However, we do not need to send these two outputs at the same time. Instead, the first input will be sent first and the other will be stored in the **OUT-**

PUT BUFFER. When the FFT stage completes computing and sending enough first half outputs, it changes to send the second half outputs, which are stored in the **OUTPUT BUFFER**. In figure 4.4, the output result of the **COMPLEX ADDITION** block is sent to the output as **"First Half Output"** and is used to multiply with the twiddle factor to create **"Second Half Output"**.

$$out_1 = x[0] + x[4] \tag{4.1}$$

$$out_2 = (x[0] - x[4]) \times W_8^0 \tag{4.2}$$

The **OUTPUT SELECTION** block is similar to the **INPUT SELECTION** block. This block controls whether to send the output directly from the **COMPLEX ADDITION** block (First Half Output) and whether to send the output from the **OUTPUT BUFFER** (Second Half Output).

Note: The **INPUT BUFFER** and **OUTPUT BUFFER** in the FFT stage is implemented with BRAM.

FFT PRELAST STAGE AND LAST STAGE

The prelast stage is the 11^{th} stage and the last stage is the 12^{th} stage of our 4096-point FFT core. The detailed block diagrams of these two stages 4.5 are similar to each other and also quite similar to the normal FFT stage, except that these two stages do not need twiddle factors and complex multiplication, as we mentioned earlier.

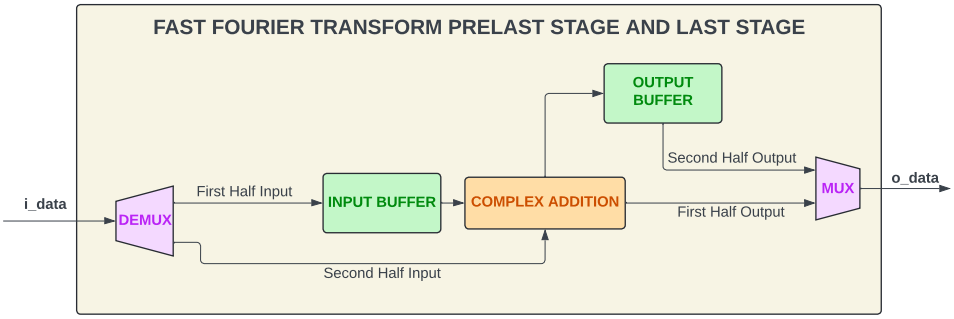


Figure 4.5: FFT prelast stage and last stage detailed block diagram

In these two stages, we utilize only complex additions and subtractions to compute the output result instead of using multiplication with twiddle factors. Moreover, the **INPUT BUFFER** and **OUTPUT BUFFER** in the case of these two stages will not need to be implemented using BRAM as in the FFT stage due to

their small size. We simply use registers instead of BRAM here. In terms of logic control, it is similar to the case of normal FFT stage. We also control the data with the mechanism of **"First Half"** and **"Second Half"**.

BIT GROWTH PROBLEM IN FFT

We have already known that complex additions and multiplications take place at every stage of the FFT core. Consequently, this leads to the problem of bit growth, and if we do not deal with it carefully, overflow can happen, and makes the result incorrect. To understand more clearly about bit growth in FFT, we will do some math here. Taking the example 4.2 and $x[0] = x[0]_r + j \times x[0]_i$, $x[1] = x[1]_r + j \times x[1]_i$, $W = W_r + j \times W_i$, we can expand out_2 to 4.3 and 4.4.

$$out_{2_r} = (x[0]_r - x[4]_r)W_r - (x[0]_i - x[4]_i)W_i \tag{4.3}$$

$$out_{2_i} = (x[0]_i - x[4]_i)W_r + (x[0]_r - x[4]_r)W_i \tag{4.4}$$

Our mission now is to find the maximum value that out_{2_r} and out_{2_i} can increase after stage 1. For out_{2_i}, this will reach the maximum value when $(x[0]_i - x[4]_i)W_r$ and $(x[0]_r - x[4]_r)W_i$ reach the maximum value.

In our GPS case, the input of the FFT core will vary in the range of -1 to +1, so $x[0]_i - x[4]_i$ and $x[0]_r - x[4]_r$ are in the range of -2 to +2.

Until now, we have $out_{2_{i_{max}}} = max(2(W_r + W_i)) = 2 \times max((cos(\phi) + sin(\phi))$. To find $max((cos(\phi) + sin(\phi))$, we solve equation 4.5 and the solution is $\phi = \frac{\pi}{4} + k\frac{\pi}{2}$ with $cos(\phi) = sin(\phi) = \frac{\sqrt{2}}{2}$.

$$\frac{d}{d\phi}(cos(\phi) + sin(\phi)) = 0 \tag{4.5}$$

Therefore, the maximum growth factor is $2 \times (\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}) = 2\sqrt{2} \approx 2.8284$ (2 bits growth). However, this can only happen when all variables in the equation to compute out_{2_i} reach their maximum value and cannot happen all the time.

To overcome this serious problem, we propose 2 options:

- **NON_SCALE FFT:** With this configuration, we propose to add guard bits to the data width. For example, if the input data are in Q1.15 binary representation, we should add an amount of $log_2(N)$ bits to the bit width of the data, which means that the data are now in Q13.15 binary representation (for the case of N = 4096). This will ensure that the intermediate result between stages in the FFT core does not cause overflow. This method

is simple and easy to design, but consumes a lot of resources. Because increasing in bit width means that we need bigger amount of BRAM to store these data and also more DSP blocks needed to compute the multiplications. However, the strongest point of this method is its significantly low error. When working with fixed point representation in FPGA, it is a must to have data loss due to rounding in computations. In **NON_SCALE** configuration, the intermediate data are not scaled, so the error is really small and acceptable. The result seems to match the result from the one executed on software with floating point. We will show this later on in the next chapter 5.

- **SCALE:** With this configuration, we propose that the intermediate results after every stage are shifted to the right by 1 bit (divided by 2). This means that the output results of the FFT core will be shifted to the right 12 bits (divided by 4096) in comparison to its **NON_SCALE** version. This method is also simple to design and also helps us save resources because the bit width of the data is not increased, such as in the configuration **NON_SCALE**. However, the error due to rounding in this method seems to be significant and needs to be taken into account.

ROUNDING IN FFT

While discussing the above bit growth problem, we did mention the error due to rounding in FFT after computations, such as additions, subtractions, and especially multiplications. For additions and subtractions, the bit growth is only 1 bit. For example, adding two 4-bit signals results in a 5-bit signal. However, with multiplication, the bit growth is the sum of the bit widths of two signals. For example, multiplying a 4-bit signal by a 5-bit signal results in a 9-bit signal. If we do not control and remove this bit growth, the bit width of the signal will increase dramatically in the FFT core design. To deal with this problem, we used two options (corresponding to two configurations **NON_SCALE** and **SCALE** of the FFT core design:

- **Truncation:** This method is simply to truncate the LSB growing bits of the results and only take the MSB bits. For example, multiplying two Q1.15 binary representation signals results in a Q2.30 binary representation result. Then, we truncate 15 LSB bits and get a Q2.15 binary representation signal. We continue taking 16 LSB bits as the final result because a signal in the Q1.15 binary representation only has a value in the range of -1 (16'b1000000000000000) to +0.999969482421875 (16'b0111111111111111). If we multiply 2 signals in the range of -1 to approximately +1, the result never exceeds the range of -1 to +1. That is the reason why we can remove

the MSB bit from the Q2.15 binary representation result to obtain the final result after truncation. In our FFT core design, we use this truncation method for the case of the **NON_SCALE** configuration. As we explained earlier, the **NON_SCALE** configuration of the FFT core will keep the accuracy of the result really well, so truncation will not affect much and will also be simple and easy to implement.

- **Convergent rounding:** Convergent rounding (or rounding half to even) is one of the rounding methods introduced by the author Dan Gisselquist in his blog [23] to deal with the rounding problem. The idea is that if the result is exactly the middle value between its lower and upper rounding values, it will be rounded to the even value. Table 4.2 gives us some examples for the convergent rounding method. With some values that are not at the exact half way point, such as 2.1, 3.1, 2.6, and 3.6, their values will be rounded up or down depending on it is near to the upper or lower rounded values. However, take a look at two last rows in the table, we will see a difference. As the name implied, this method will round the data at the exact half way point to its nearest even value. In this case, 2.5 is rounded down to 2 (2 is even) while 3.5 is rounded up to 4 (4 is even). An additional interesting information from Dan Gisselquist is that convergent rounding is the default rounding mode used in IEEE 754 computing functions.

Original value	Convergent rounding
2.1	2
3.1	3
2.6	3
3.6	4
2.5	2
3.5	4

Table 4.2: Example for convergent rounding

4.1.3. COMPLEX MULTIPLIER BLOCK

This is not actually a separated module in our GPS acquisition design. They are implemented directly in the top module of the acquisition because it is quite simple. A complex multiplication consists of 4 real multiplications, 1 real subtraction and 1 real addition as shown in the diagram 4.6 supposing that the first complex number is $complex_1 = a + bj$ and the second complex number is $complex_2 = c + dj$. At least 4 DSP48E2 blocks are used to compute a complex

multiplication. The number of DSP48E2 blocks used can increase depending on the bit width of the input signal. The DSP48E2 block can support up to 27 x 18 bits multiplication.

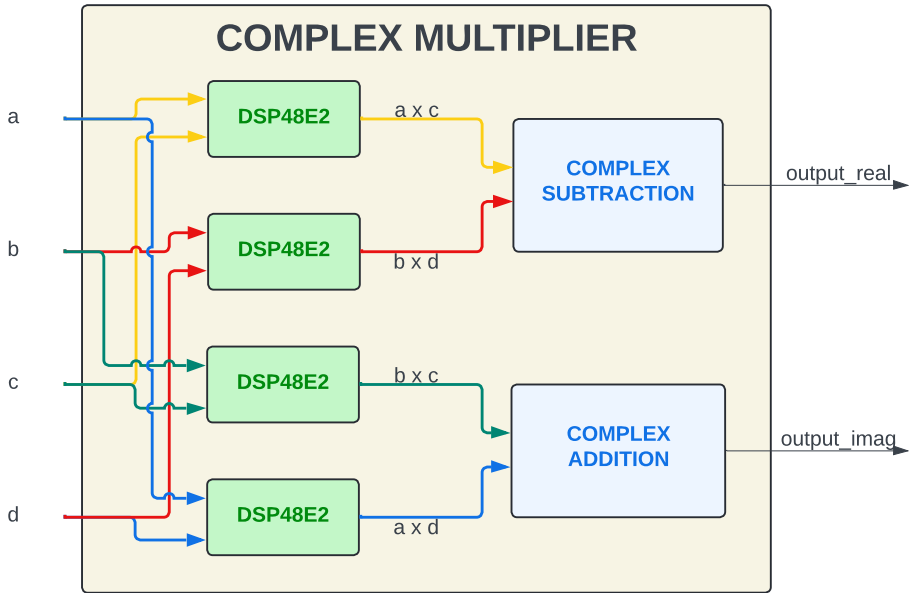


Figure 4.6: Complex multiplier block diagram

4.1.4. BUFFER BLOCKS

All the **BUFFER** blocks shown in Figure 3.4 are implemented using BRAM due to its large size:

- **INPUT BUFFER**: The input of the GPS acquisition design is a complex number with two parts: real and imaginary. Each of the real and imaginary parts is a Q1.15 binary representation (16 bits). Therefore, we need a memory with depth of 4096 elements, and width of each element is 32 bits.
- **DOPPLER SHIFT BUFFER**: The computed results of the **ACCUMULATOR** and the **CORDIC** blocks also have two parts: Q1.15 binary representation of real and imaginary parts. Therefore, we also need a memory with depth of 4096 elements, and width of each element is 32 bits.
- **FFT RESULT BUFFER:** The depth of this buffer is also 4096 but the bit width of each element can vary depending on the configuration **NON_SCALE** or **SCALE**. With **NON_SCALE** configuration, the bit width is 28 bits while it is 16 bits for the **SCALE** configuration.

- **LOCAL PRN BUFFER:** This buffer is used to store precomputed PRN codes for 32 satellites, so its size is really large. We can imagine this buffer as an array $[15:0]mem[0:31][0:4095]$. This means that we will have 32 separated memories, and each memory has a depth of 4096 with the element's bit width of 16 bits. The bit width here is only 16 bits because with the locally generated PRN code, we do not have the imaginary part. The local code contains only real values.

4.1.5. TOP 4 PEAK SELECTION BLOCK

This block is to choose the four satellites that have the highest correlation results to send to the output of the GPS acquisition design. The easiest solution for this block is the sorting algorithm. That is, after the acquisition design has completed its operation, this block will try to sort all the results and then send the data corresponding to the four highest correlation results to the output. However, we observe that, in fact, we only want to pick the four satellites, the others are not necessary to be sorted. Therefore, we propose a solution for this block with two smaller blocks inside as shown in Figure 4.7.

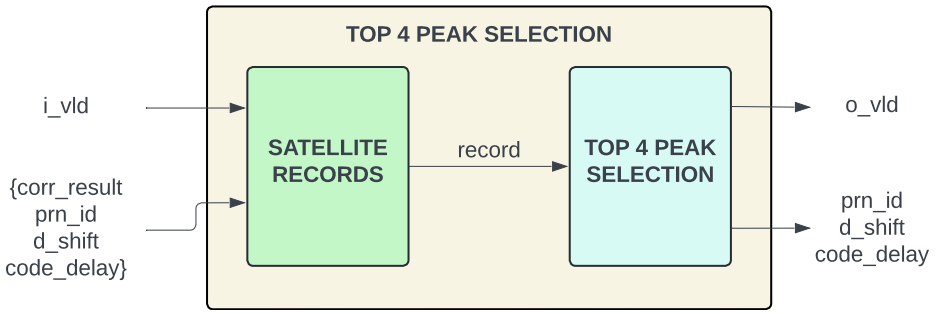


Figure 4.7: Two smaller blocks inside Top 4 Peak Selection block

There are in total $81 \times 32 \times 4096 = 10616832$ correlation results. The **Satellite Records** block will continuously receive all these correlation results. There are 32 registers inside the **Satellite Records** block and each of them has the responsibility of recording the highest correlation result of each satellite together with some data such as prn_id, d_shift, and code_delay. The detailed block diagram for the **Satellite Record** block is shown in Figure 4.8.

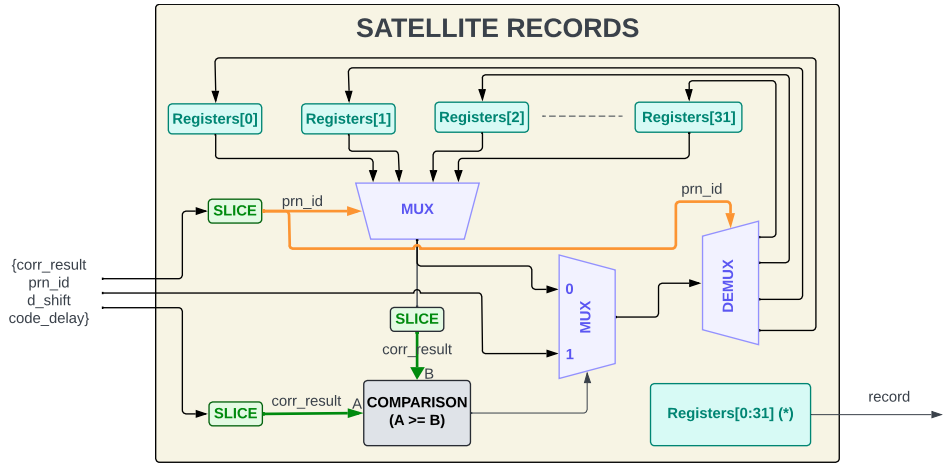


Figure 4.8: Detailed block diagram for Satellite Record block

This is the working operation of the **Satellite Records** block (for each valid input):

- **Step 1:** Compare the input corr_result with the corr_result being stored at index prn_id of the **Registers**. It is to compare corr_result ≥ Registers[prn_id].corr_result. If the result is **TRUE**, the data at the index prn_id of the Registers are updated with the input data. It is to update Registers[prn_id] = input_data. Otherwise, it will keep its previous data.

- **Step 2:** Repeat the process until the GPS acquisition operation is complete.

- **Step 3:** Start sending the Registers array one by one to the **Top 4 peak selection** block after GPS acquisition is complete.

The **Top 4 peak selection** block is simple in idea but quite difficult to draw in the detailed block diagram, so we will instead present the flow chart in Figure 4.9, 4.10, 4.11, 4.12, and 4.13.

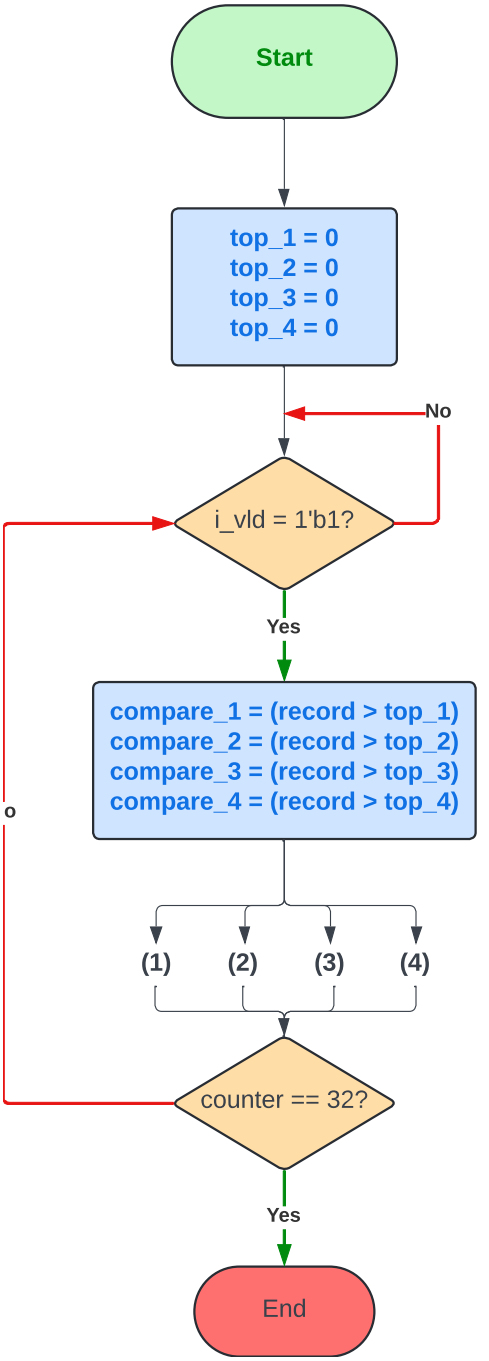


Figure 4.9: Top 4 peak selection flow chart

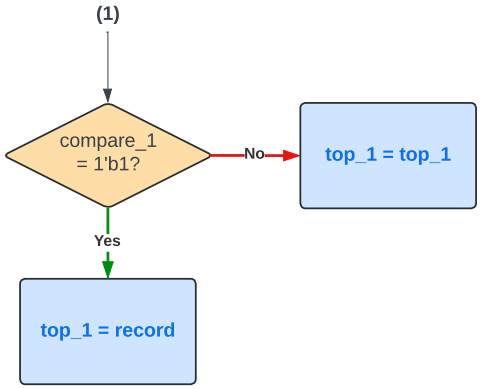


Figure 4.10: Top 4 peak selection flow chart - Branch (1)

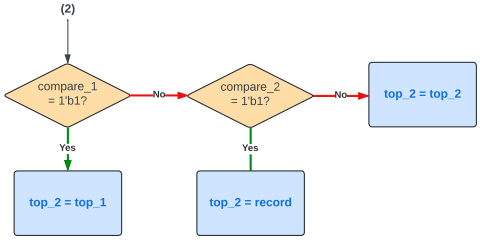


Figure 4.11: Top 4 peak selection flow chart - Branch (2)

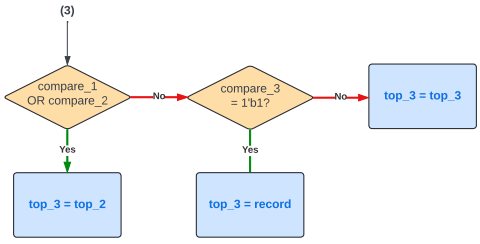


Figure 4.12: Top 4 peak selection flow chart - Branch (3)

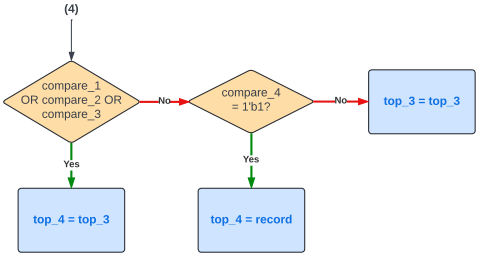


Figure 4.13: Top 4 peak selection flow chart - Branch (4)

The input of this block are 32 highest correlation results of 32 satellites from the **Satellite Records** block. There are 4 registers to store the data corresponding to four satellites that have the highest correlation results. They are **top_1**, **top_2**, **top_3**, and **top_4** in the flow chart. The working operation of this block is:

- **Step 1:** Initialize the value of **top_1**, **top_2**, **top_3**, and **top_4** to be 0s.
- **Step 2:** Whenever there is a valid input data, update the value of 4 registers following the flow chart 4.10, 4.11, 4.12, and 4.13. The general idea to update the data of each register is if the corr_result of the input data is bigger than the corr_result in the data currently stored in the register but smaller than all corr_result values stored in the higher level registers, the input data will be updated to the register. If the corr_result in the input data is bigger than the corr_result in the data stored in higher level registers, the register will be updated the value of the right higher level register. If the corr_result is smaller than the corr_result stored in the register, the register will not be updated. For clearer understanding, please continue reading Steps 2.1, 2.2, 2.3 and 2.4 and also an example we place right below this bulleted list.
- **Step 2.1:** Register top_1 is updated by the input data whenever the corr_result in the input data is bigger than the corr_result in the data currently stored at register top_4.
- **Step 2.2:** For register top_2, if the corr_result in the input data is bigger than the corr_result of the data stored in top_1, top_2 will be updated by the data currently stored in top_1. It is similar to shift registers. Otherwise, we continue checking if the input data is bigger than the corr_result of the data stored in top_2. If so, we update the input data to top_2. If not, it means that the corr_result in the input data is smaller than both the corr_result of top_1 and top_2, so we will not update the data of top_2.

- **Step 2.3:** For register top_3, the process is similar to Step 2.2 except that we will check whether the corr_result in the input data is bigger than the corr_result of the data stored in top_1 and top_2 to update the data of register top_3 by data stored in register top_2.

- **Step 2.4:** For register top_4, the process is similar to Steps 2.2 and 2.3 except that we will check whether the corr_result in the input data is bigger than the corr_result of the data stored in top_1, top_2, or top_3 to update the data of register top_4 by data stored in register top_3.

- **Step 3:** The process repeats until it checks enough 32 valid input data.

- **Step 4:** After completing, send the data stored in 4 registers top_1, top_2, top_3, and top_4 to the output of the GPS acquisition top module.

Let us give an example! Supposing that currently the corr_result stored at top_1, top_2, top_3, and top_4 are respectively 4, 3, 2, and 1. Now there is a new valid input data that has the corr_result of 2.5, we need to update the registers as follows.

- **Register top_1:** $2.5 < 4$ so register top_1 is not updated.

- **Register top_2:** $2.5 < 3$ so register top_2 is not updated.

- **Register top_3:** $2.5 \geq 2$ and $2.5 < 3$ and $2.5 < 4$, so register top_3 is updated by the input data.

- **Register top_4:** $2.5 \geq 1$, but 2.5 is also greater than or equal to the value of 2 of register top_3. Therefore, register top_4 is also updated but by the data of register top_3.

- **Final result:** The value of corr_result of top_1, top_2, top_3, and top_4 are respectively 4, 3, 2.5, and 2. **Note:** the updates of registers are non-blocking assignment, which means they occur simultaneously.

4.2. GPS TRACKING PHASE

4.2.1. CARRIER WIPE-OFF

To eliminate the residual Doppler shift still in the baseband signal, the carrier wipe-off operation is performed. The input signal at baseband used in-phase component as real component and quadrature component as imaginary component. Therefore, the multiplication operation to further shift the signal

out of Doppler shift is a complex multiplication operation with a sine and cosine value created by an NCO where cosine acted as the real component and sine acted as the imaginary component:

$$\begin{aligned} mixer\_i + j \times mixer\_q &= (in\_i + j \times in\_q)(carr\_cos + j \times carr\_sin) \\ &= (in\_i \times carr\_cos - in\_q \times carr\_sin) \\ &+ j \times (in\_i \times carr\_sin + in\_q \times carr\_cos) \end{aligned} \tag{4.6}$$

To implement the mixing result from the equation 4.6, the following design is used:

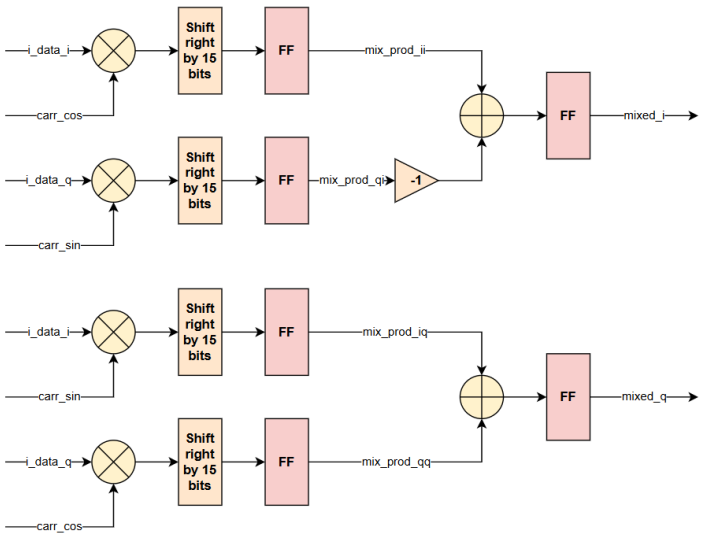


Figure 4.14: Carrier wipe-off implementation

Due to the input signal and NCO signal both being Q1.15 fixed-point number, after the multiplication, the result must be shifted to the right by 15 bit. The resulting multiplication product is buffered by a flip-flop before going through another addition or subtraction stage before being buffered by another flip-flop again. To generate the sine and cosine signal, 2 LUTs storing the sine and cosine value is generated:

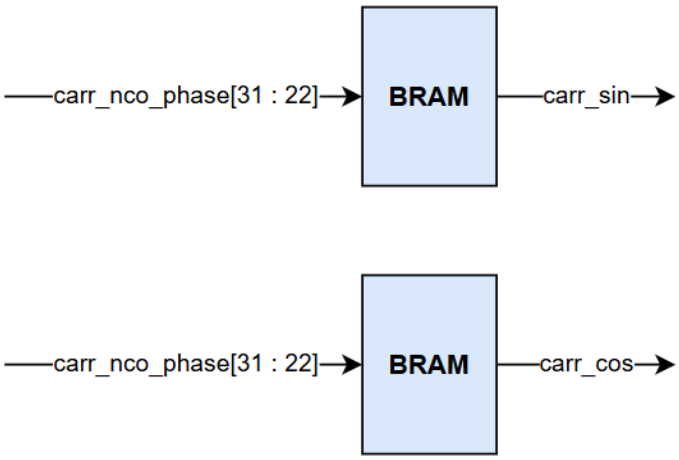


Figure 4.15: NCO implementation

The size of the NCO in this case is chosen to have the depth of 1024 samples so we only take the 10 MSB of the 32 bit wide input phase signal. The phase is calculated by accumulating the phase increment that is calculated by the PLL:

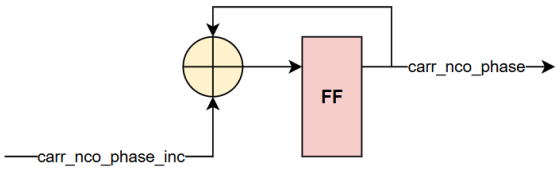


Figure 4.16: NCO phase accumulator

4.2.2. CORRELATION AND CODE GENERATORS

As mentioned, the 4096 code samples is stored into BRAM as 4096 × 32 bits size, with bit 31 down to bit 0 on each line represented PRN ID from ID 1 to ID 32. A total of 3 code generators are generated for each tracker:

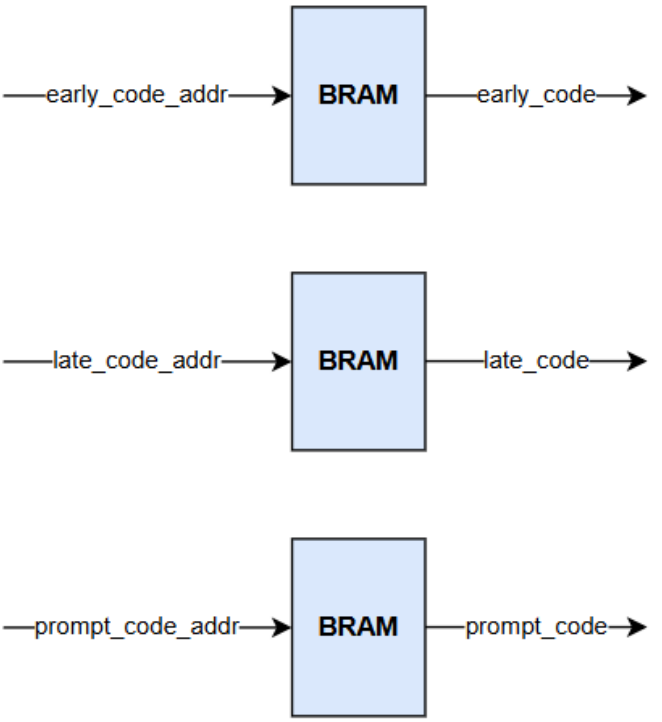


Figure 4.17: Code generators implementation

The addresses needed to get the code value is calculated based on 2 accumulator, the first one accumulates the code phase increment value got from the DLL, the second one accumulates the 12 MSB of the phase value. Then, this accumulated value is used to calculate the 3 addresses needed in a combinational logic block based on equation 2.33:

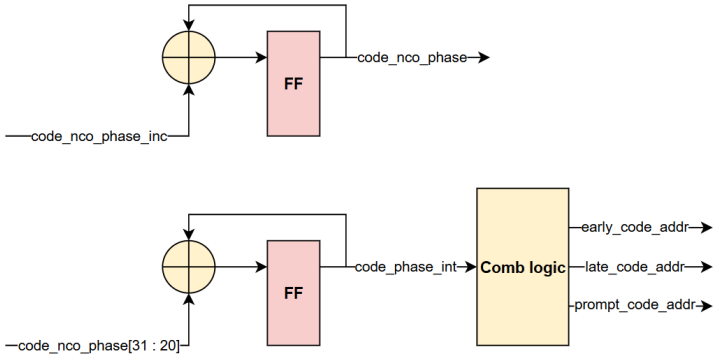


Figure 4.18: Code phase accumulator

Since each code sample is either 1 or 0 to represent the value of 1 or -1, we need to do correlation multiplication by just taking 2's complement of the mixer output result. However, this represented a problem caused by our usage of Q1.15 fixed point format number in the case of -1 × -1. When the mixing output is at -1, the Q1.15 representation will be 0x8000 but the 2's complement will be be $\sim 0x8000 + 1 = 0x7FFF + 1 = 0x8000$ again which is -1 and not 1 like we needed. Therefore, we need to split the correlation result based on 3 cases:

- When the code is -1 (represented by 0) and the mixer output is -1, then the correlation result is 1 or 0x7FFF.
- When the code is -1 and the mixer output is not -1, then the correlation result is the 2's complement of mixer output.
- When the code is 1 (represented by 1), then the correlation result is the mixer output.

4.2.3. INTEGRATE AND DUMP

The dump timing is selected to be at 1 kHz or the dump action happens every 1 ms. Therefore, at the 4096000 SPS sampling rate, the dump action happens once every 4096 clock cycle. Therefore, a counter is implemented that counts up until it reaches 4095 where it will assert the *dump_correlators* signal for 1 clock cycle before deasserting it again. To help with pipelining for later stage, another 4 flip flop stages are included:

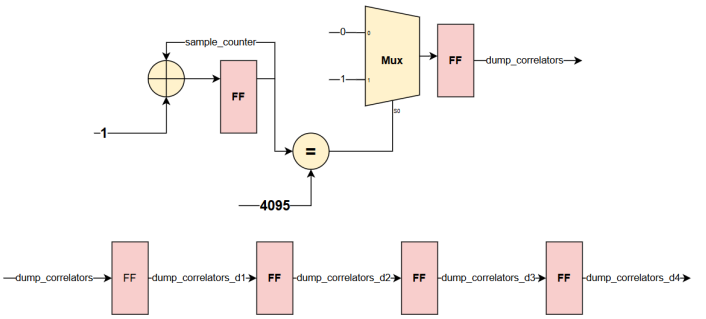


Figure 4.19: Dump timing generator

The *dump_correlators* signal is later used to control the correlator output samples accumulator. The *early_acc_i_reg* signal will receive the accumulator result every time the signal *dump_correlators* is asserted:

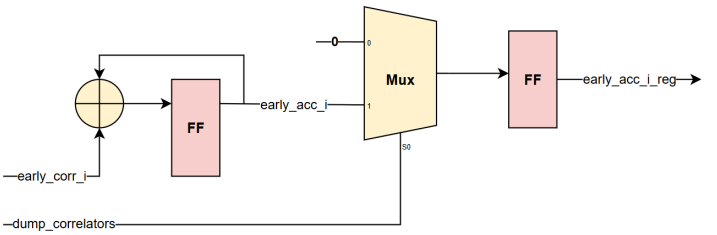


Figure 4.20: *early_acc_i_reg* Integrate and Dump diagram

The same design is replicated for *early_acc_q_reg*, *late_acc_i_reg*, *late_acc_q_reg*, *prompt_acc_i_reg*, *prompt_acc_q_reg* integrate and dump block.

4.2.4. PHASE LOCKED LOOP AND PI FILTER

To help with timing, we divided the PLL and PI filter into 2 pipeline stages, the first stage to calculate the discriminator approximation based on the equation 2.23 while the second stage is based on the equations 2.24, 2.25 and 2.26. The discriminator approximation calculation stage is controlled by the signal *dump_correlators_d1*:

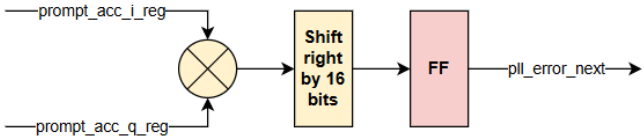


Figure 4.21: PLL discriminator calculation

With the PLL phase error calculation finished, we filtered it with a PI filter based on the equation 2.29 in which *PLL_K1* is the proportional gain and *PLL_K2* is the integral gain, with both of these numbers represented in Q16.16 format for more precision. This entire filter is controlled by the *dump_correlators_d2* signal:

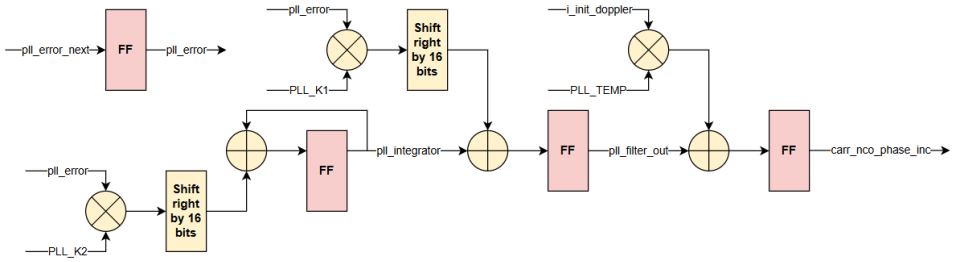


Figure 4.22: Carrier NCO phase increment calculation

Through trial and error, we selected the proportional gain to be 0.03 and the integral gain to be 0.5. After we get the resulting filtered output, the phase increment *carr_nco_phase_inc* is calculated based on 2.26 and 2.25. However, to simplify calculation, we can pre-calculate the $\Delta\phi_0$ as *PLL_TEMP*:

$$PLL\_TEMP = \frac{2^{32}}{4096000} = 1049 \tag{4.7}$$

4.2.5. DELAY LOCKED LOOP AND PI FILTER

Due to the heavy use of math to calculate the DLL discriminator and filter based on equations 2.28, 2.29, 2.30 and 2.31, we divided it into 4 pipeline stages. The first pipelining stage is controlled by *dump_correlators_d1* signal and it calculates the square of *early_acc_i_reg*, *early_acc_q_reg*, *late_acc_i_reg* and *late_acc_q_reg*. An example for *early_i_sq*:

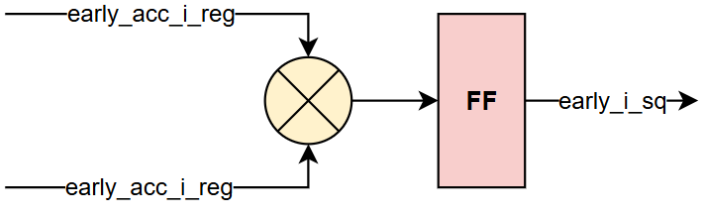


Figure 4.23: First in-phase early accumulator power pipeline stage

Since *early_i_sq* is a Q1.15 format fixed-point number, we must shift back by 16 bit (15 bit fixed point and 1 bit for divide by 2), this stage is controlled by *dump_correlators_d2* signal:

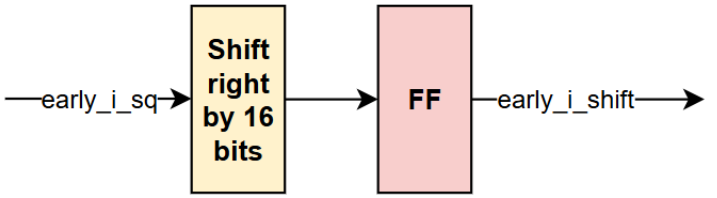


Figure 4.24: Second early accumulator power pipeline stage

Finally, the discriminator controlled by the signal *dump_correlators_d3* is implemented:

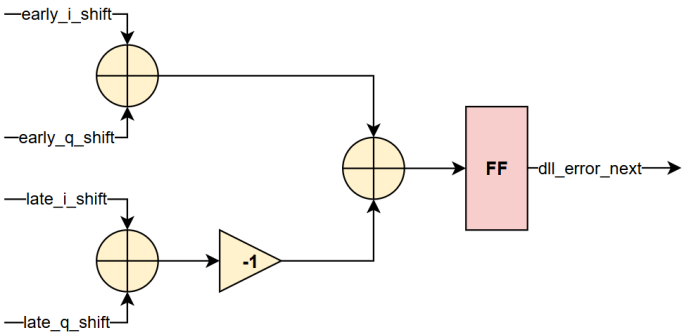


Figure 4.25: DLL discriminator calculation

With the DLL code phase error calculation finished, we filtered it with a PI filter based on the equation 2.30 in which *DLL_K1* is the proportional gain and *DLL_K2* is the integral gain, with both of these numbers represented in Q16.16

format for more precision. This entire filter is controlled by the *dump_correlators_d4* signal:

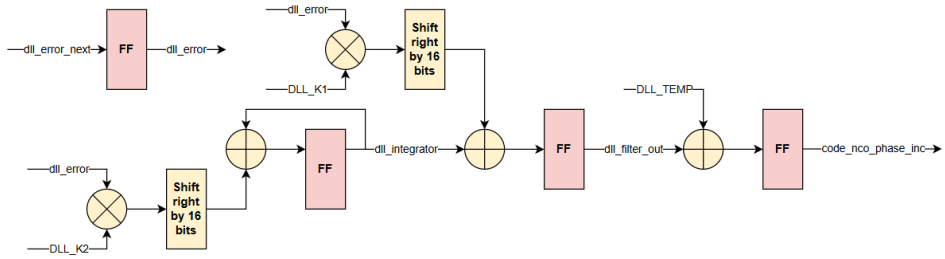


Figure 4.26: Code NCO phase increment calculation

Through trial and error, we selected the proportional gain to be 0 and the integral gain to be 0.001. After we get the resulting filtered output, the phase increment *code_nco_phase_inc* is calculated based on 2.31 and 2.32. However, to simplify calculation, we can pre-calculate the $\Delta\text{code\_phase}_0$ as *DLL_TEMP*:

$$DLL\_TEMP = 1023000 \times \frac{2^{32}}{4096000} = 1072693248 \tag{4.8}$$

4.2.6. NAVIGATION MESSAGE EXTRACTION

To extract the navigation message at 50 Hz or 20 ms period based on the timing of 1 kHz provided by the dump timing signal *dump_correlators_d1*, we sample the sign of the real or in-phase component of the accumulator result of the prompt correlators *prompt_acc_i* once every 20 samples so a bit counter triggered by *dump_correlators_d1* is implemented. Since *prompt_acc_i* is a 32 bit variable used to represent the Q1.15 format number, we can take the MSB 31 bit as the sign of the number since it is sign extended. Since the *o_nav_bit* signal used the value of 0 or 1 to represent the sign value of negative or positive, then we must take the inverse of the sign value (if the sign is positive, then MBS 31 is 0 while 1 when the sign is negative). A signal called *o_nav_bit_valid* is also used to indicate when the navigation message is valid:

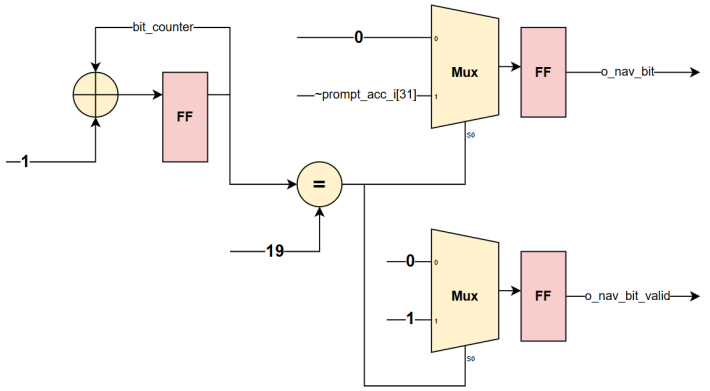


Figure 4.27: Navigation message extraction

5

EXPERIMENTAL SETUP & EVALUATION

In this chapter, we will show you the result of functional verification of:

- GPS acquisition design.
- GPS tracking design.

And the result of synthesis and implementation of the GPS receiver design (integrating GPS acquisition and tracking designs into the top module). In addition, we also give some evaluations of the performance, power, and temperature of the GPS receiver design.

5.1. INPUT GENERATION

As mentioned above, raw GPS signal data can be downloaded online; however, it is often difficult to find suitable files, as most are not compatible with our system design requirements (I/Q 16-bit format, sampling rate of 4096 MHz). Thanks to [18], it is now possible for us to generate raw GPS signal data that is usable for our system design. Besides, several steps are still needed to be done.

Firstly, we need to obtain a daily GPS broadcast ephemeris file (brdc), which can be downloaded from the NASA government website [24]. The correct file format should be ***brdcDDD0.YYn.gz***, where **DDD** represents the three-digit day of the year and **YY** represents the last two digits of the year. It is important to pay attention to the **YYn** part, as downloading a file with **YYg** instead will give us a GLONASS broadcast ephemeris file, which is not suitable for GPS simulation.

Secondly, assume we have a correct brdc file, now it is possible for us to generate raw baseband gps signal file by [18] with some following options:

- -e <brdc file>: RINEX emepheris file downloaded above.
- -c <location>: ECEF X, Y, and Z in meters indicating the user position (can be determined by [25]).
- -d <duration>: simulation duration time.
- -o <output file>: I/Q sampling data file.
- -s <frequency>: Sampling frequency (4096000 Hz in our design).
- -b <bits>: I/Q data format (1/8/16).

For clarity, the following example demonstrates how to generate raw GPS baseband signals in 16-bit I/Q format with a sampling frequency of 4096 MHz:


```
./gps-sdr-sim -e brdc0010.25n -c
   -1802683,6006748,1157978 -s 4096000 -d 30 -o
   input.bin
```

In the example above, the specified ECEF coordinates correspond to Ho Chi Minh City. The command initiates the capture of GPS C/A codes from visible satellites for 30 seconds and outputs the resulting signal to a file named `input.bin`. This is just a simple example in static mode - where the user position does not change, we can also simulate the trajectory of user position throughout the process in dynamic mode by input an excel file recording the user ECEF coordinate positions throughout simulation time.

5.2. FUNCTIONAL VERIFICATION

5.2.1. GPS ACQUISITION PHASE

FAST FOURIER TRANSFORM VERIFICATION

In the working operation of the GPS acquisition phase, we need to compute the FFT of the locally generated PRN codes of all satellites. Therefore, we also tested our FFT design using the input data, which are PRN codes of 32 satellites sampled at 4.096 MHz sampling frequency. We also provide the input data to Matlab and use its built-in FFT function to compute the expected output data. These expected data are then stored in mem files and will be read in the FFT verification testbench. While running the FFT testbench, we continuously followed the valid output data and stored them in an array. After running the FFT for all satellite PRN codes, we compare the actual output stored in the array with the expected output we get from Matlab.

Figures 5.1, 5.2, and 5.3 show us the result of functional verification of the FFT design in NON_SCALE mode. The data in our design are in fixed-point values, so the appearance of errors is a must. Therefore, our target is to compare the actual and expected output results considering a small error. If the error is too large, then there must be some problems that occur inside the design, and we will fix that until the error is low enough. We do not check the error on the absolute value of the FFT output but instead check the error on the value of the real and imaginary parts of the actual output result of our FFT design compared with those of the expected output result. To evaluate the error in the FFT design results, we computed the root mean square error and its normalized version as in Equations 5.1 and 5.2.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.1}$$

$$NRMSE = \frac{NMSE}{\overline{y}} \tag{5.2}$$

where

- $RMSE$ is root mean square error.
- $NRMSE$ is normalized root mean square error.
- n is the number of samples (n = 4096 in our case).
- y is the actual output of our FFT design.
- $\hat{y}$ is the expected output obtained from Matlab.
- $\overline{y}$ is the mean value of the actual output from our FFT design.

```
--------------- Max diff (for each satellite) ---------------

Satellite 0 | Max Real: 0.027527 at index: 1 | Max Imaginary: 0.029999 at index: 3
            | RMSE Real: 0.001430 | RMSE Imaginary: 0.001624 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000089

Satellite 1 | Max Real: 0.028961 at index: 1 | Max Imaginary: 0.032043 at index: 3
            | RMSE Real: 0.001415 | RMSE Imaginary: 0.001672 | NRMSE Real: 0.000079 | NRMSE Imaginary: 0.000090

Satellite 2 | Max Real: 0.030029 at index: 1 | Max Imaginary: 0.030365 at index: 3
            | RMSE Real: 0.001484 | RMSE Imaginary: 0.001602 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000092

Satellite 3 | Max Real: 0.027985 at index: 1 | Max Imaginary: 0.032257 at index: 3
            | RMSE Real: 0.001451 | RMSE Imaginary: 0.001620 | NRMSE Real: 0.000077 | NRMSE Imaginary: 0.000093

Satellite 4 | Max Real: 0.027649 at index: 1 | Max Imaginary: 0.031891 at index: 3
            | RMSE Real: 0.001448 | RMSE Imaginary: 0.001642 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000089

Satellite 5 | Max Real: 0.028656 at index: 1 | Max Imaginary: 0.032257 at index: 3
            | RMSE Real: 0.001486 | RMSE Imaginary: 0.001592 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000091
```

Figure 5.1: FFT design functional verification result - Version: NON_SCALE

```
Satellite 6 | Max Real: 0.028839 at index: 1 | Max Imaginary: 0.031647 at index: 3
            | RMSE Real: 0.001482 | RMSE Imaginary: 0.001652 | NRMSE Real: 0.000082 | NRMSE Imaginary: 0.000090

Satellite 7 | Max Real: 0.028198 at index: 1 | Max Imaginary: 0.031952 at index: 3
            | RMSE Real: 0.001467 | RMSE Imaginary: 0.001627 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000093

Satellite 8 | Max Real: 0.028870 at index: 1 | Max Imaginary: 0.030487 at index: 3
            | RMSE Real: 0.001483 | RMSE Imaginary: 0.001597 | NRMSE Real: 0.000082 | NRMSE Imaginary: 0.000090

Satellite 9 | Max Real: 0.027985 at index: 1 | Max Imaginary: 0.031128 at index: 3
            | RMSE Real: 0.001452 | RMSE Imaginary: 0.001631 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000092

Satellite 10 | Max Real: 0.027771 at index: 1 | Max Imaginary: 0.029541 at index: 3
             | RMSE Real: 0.001447 | RMSE Imaginary: 0.001638 | NRMSE Real: 0.000083 | NRMSE Imaginary: 0.000087

Satellite 11 | Max Real: 0.028687 at index: 1 | Max Imaginary: 0.032227 at index: 3
             | RMSE Real: 0.001481 | RMSE Imaginary: 0.001633 | NRMSE Real: 0.000084 | NRMSE Imaginary: 0.000091
```

Figure 5.2: FFT design functional verification result - Version: NON_SCALE (cont)


```
Satellite 12 | Max Real: 0.028076 at index: 1 | Max Imaginary: 0.032013 at index: 3
             | RMSE Real: 0.001439 | RMSE Imaginary: 0.001630 | NRMSE Real: 0.000078 | NRMSE Imaginary: 0.000091

Satellite 13 | Max Real: 0.029022 at index: 1 | Max Imaginary: 0.029877 at index: 3
             | RMSE Real: 0.001491 | RMSE Imaginary: 0.001643 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000094

Satellite 14 | Max Real: 0.028900 at index: 1 | Max Imaginary: 0.031586 at index: 3
             | RMSE Real: 0.001458 | RMSE Imaginary: 0.001651 | NRMSE Real: 0.000079 | NRMSE Imaginary: 0.000092

Satellite 15 | Max Real: 0.027405 at index: 1 | Max Imaginary: 0.031891 at index: 3
             | RMSE Real: 0.001457 | RMSE Imaginary: 0.001648 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000091

Satellite 16 | Max Real: 0.029083 at index: 1 | Max Imaginary: 0.030334 at index: 3
             | RMSE Real: 0.001488 | RMSE Imaginary: 0.001608 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000089

Satellite 17 | Max Real: 0.029572 at index: 1 | Max Imaginary: 0.031036 at index: 3
             | RMSE Real: 0.001520 | RMSE Imaginary: 0.001603 | NRMSE Real: 0.000083 | NRMSE Imaginary: 0.000091
```

Figure 5.3: FFT design functional verification result - Version: NON_SCALE (cont)

```
Satellite 18 | Max Real: 0.029114 at index: 1 | Max Imaginary: 0.031433 at index: 3
             | RMSE Real: 0.001509 | RMSE Imaginary: 0.001593 | NRMSE Real: 0.000084 | NRMSE Imaginary: 0.000089

Satellite 19 | Max Real: 0.028534 at index: 1 | Max Imaginary: 0.028595 at index: 3
             | RMSE Real: 0.001460 | RMSE Imaginary: 0.001626 | NRMSE Real: 0.000082 | NRMSE Imaginary: 0.000088

Satellite 20 | Max Real: 0.028015 at index: 1 | Max Imaginary: 0.031799 at index: 3
             | RMSE Real: 0.001489 | RMSE Imaginary: 0.001642 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000092

Satellite 21 | Max Real: 0.029755 at index: 1 | Max Imaginary: 0.030243 at index: 3
             | RMSE Real: 0.001528 | RMSE Imaginary: 0.001583 | NRMSE Real: 0.000083 | NRMSE Imaginary: 0.000090

Satellite 22 | Max Real: 0.028534 at index: 1 | Max Imaginary: 0.030945 at index: 3
             | RMSE Real: 0.001503 | RMSE Imaginary: 0.001628 | NRMSE Real: 0.000083 | NRMSE Imaginary: 0.000092

Satellite 23 | Max Real: 0.027893 at index: 1 | Max Imaginary: 0.030396 at index: 3
             | RMSE Real: 0.001461 | RMSE Imaginary: 0.001618 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000090
```

Figure 5.4: FFT design functional verification result - Version: NON_SCALE

```
Satellite 24 | Max Real: 0.028961 at index: 1 | Max Imaginary: 0.031952 at index: 3
             | RMSE Real: 0.001486 | RMSE Imaginary: 0.001625 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000090

Satellite 25 | Max Real: 0.029114 at index: 1 | Max Imaginary: 0.031311 at index: 3
             | RMSE Real: 0.001488 | RMSE Imaginary: 0.001615 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000091

Satellite 26 | Max Real: 0.028076 at index: 1 | Max Imaginary: 0.030518 at index: 3
             | RMSE Real: 0.001388 | RMSE Imaginary: 0.001658 | NRMSE Real: 0.000078 | NRMSE Imaginary: 0.000090

Satellite 27 | Max Real: 0.027954 at index: 1 | Max Imaginary: 0.032166 at index: 3
             | RMSE Real: 0.001482 | RMSE Imaginary: 0.001603 | NRMSE Real: 0.000082 | NRMSE Imaginary: 0.000091

Satellite 28 | Max Real: 0.027771 at index: 1 | Max Imaginary: 0.030487 at index: 3
             | RMSE Real: 0.001448 | RMSE Imaginary: 0.001639 | NRMSE Real: 0.000079 | NRMSE Imaginary: 0.000091

Satellite 29 | Max Real: 0.028961 at index: 1 | Max Imaginary: 0.030090 at index: 3
             | RMSE Real: 0.001421 | RMSE Imaginary: 0.001632 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000089
```

Figure 5.5: FFT design functional verification result - Version: NON_SCALE (cont)


```
Satellite 30 | Max Real: 0.028229 at index: 1 | Max Imaginary: 0.029572 at index: 3
             | RMSE Real: 0.001487 | RMSE Imaginary: 0.001616 | NRMSE Real: 0.000081 | NRMSE Imaginary: 0.000093

Satellite 31 | Max Real: 0.029297 at index: 1 | Max Imaginary: 0.031097 at index: 3
             | RMSE Real: 0.001411 | RMSE Imaginary: 0.001677 | NRMSE Real: 0.000080 | NRMSE Imaginary: 0.000091

-----------------------------------------------------------------

Max diff all time | Real: 0.030029 at index 1 of Satellite 2 | Imaginary: 0.032257 at index 3 of Satellite 5

Simulation Success!

##########################################################
$finish called at time : 1393190 ns : File "D:/Capstone Project/gps_sdr/gps_on_fpga/gps_rtl_tb/gps_acquisition/rtl
```

Figure 5.6: FFT design functional verification result - Version: NON_SCALE (cont)

The root mean square error (RMSE) values for all satellite PRN codes' FFT results are really small in the case of NON_SCALE configuration. To better see the error, we also compute the normalized version of the RMSE, which is NRMSE. In the case of NON_SCALE configuration, the values of NRMSE are also very small. All NRMSE values are only approximately 0.009% of the mean value of the FFT output.

Next, we will check the RMSE and NRMSE values for the case of SCALE configuration.

```
---------------- Max diff (for each satellite) ----------------

Satellite 0 | Max Real: 0.000092 at index: 15 | Max Imaginary: 0.000092 at index: 19
            | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005415 | NRMSE Imaginary: 0.005260

Satellite 1 | Max Real: 0.000092 at index: 27 | Max Imaginary: 0.000092 at index: 24
            | RMSE Real: 0.000023 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005348 | NRMSE Imaginary: 0.005226

Satellite 2 | Max Real: 0.000092 at index: 32 | Max Imaginary: 0.000092 at index: 8
            | RMSE Real: 0.000023 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005198 | NRMSE Imaginary: 0.005569

Satellite 3 | Max Real: 0.000092 at index: 23 | Max Imaginary: 0.000092 at index: 15
            | RMSE Real: 0.000023 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005119 | NRMSE Imaginary: 0.005505

Satellite 4 | Max Real: 0.000092 at index: 22 | Max Imaginary: 0.000092 at index: 15
            | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005295 | NRMSE Imaginary: 0.005228

Satellite 5 | Max Real: 0.000092 at index: 10 | Max Imaginary: 0.000092 at index: 28
            | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005250 | NRMSE Imaginary: 0.005482
```

Figure 5.7: FFT design functional verification result - Version: SCALE

5

```
Satellite 6 | Max Real: 0.000092 at index: 42 | Max Imaginary: 0.000122 at index: 1
            | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005319 | NRMSE Imaginary: 0.005412

Satellite 7 | Max Real: 0.000092 at index: 30 | Max Imaginary: 0.000092 at index: 31
            | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005371 | NRMSE Imaginary: 0.005513

Satellite 8 | Max Real: 0.000092 at index: 23 | Max Imaginary: 0.000092 at index: 22
            | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005413 | NRMSE Imaginary: 0.005373

Satellite 9 | Max Real: 0.000092 at index: 15 | Max Imaginary: 0.000092 at index: 28
            | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005243 | NRMSE Imaginary: 0.005405

Satellite 10 | Max Real: 0.000092 at index: 28 | Max Imaginary: 0.000092 at index: 15
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005566 | NRMSE Imaginary: 0.005023

Satellite 11 | Max Real: 0.000122 at index: 1 | Max Imaginary: 0.000092 at index: 28
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005441 | NRMSE Imaginary: 0.005426
```

Figure 5.8: FFT design functional verification result - Version: SCALE (cont)

```
Satellite 12 | Max Real: 0.000092 at index: 20 | Max Imaginary: 0.000092 at index: 31
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005297 | NRMSE Imaginary: 0.005418

Satellite 13 | Max Real: 0.000122 at index: 0 | Max Imaginary: 0.000092 at index: 45
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005268 | NRMSE Imaginary: 0.005501

Satellite 14 | Max Real: 0.000092 at index: 25 | Max Imaginary: 0.000122 at index: 1
             | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005221 | NRMSE Imaginary: 0.005307

Satellite 15 | Max Real: 0.000122 at index: 0 | Max Imaginary: 0.000092 at index: 45
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005364 | NRMSE Imaginary: 0.005268

Satellite 16 | Max Real: 0.000092 at index: 55 | Max Imaginary: 0.000092 at index: 23
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005325 | NRMSE Imaginary: 0.005229

Satellite 17 | Max Real: 0.000092 at index: 23 | Max Imaginary: 0.000092 at index: 45
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005259 | NRMSE Imaginary: 0.005471
```

Figure 5.9: FFT design functional verification result - Version: SCALE (cont)

```
Satellite 18 | Max Real: 0.000092 at index: 18 | Max Imaginary: 0.000092 at index: 32
             | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005298 | NRMSE Imaginary: 0.005359

Satellite 19 | Max Real: 0.000092 at index: 23 | Max Imaginary: 0.000092 at index: 31
             | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005367 | NRMSE Imaginary: 0.005220

Satellite 20 | Max Real: 0.000092 at index: 30 | Max Imaginary: 0.000092 at index: 31
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005259 | NRMSE Imaginary: 0.005403

Satellite 21 | Max Real: 0.000092 at index: 39 | Max Imaginary: 0.000092 at index: 42
             | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005089 | NRMSE Imaginary: 0.005357

Satellite 22 | Max Real: 0.000092 at index: 27 | Max Imaginary: 0.000092 at index: 20
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005290 | NRMSE Imaginary: 0.005427

Satellite 23 | Max Real: 0.000092 at index: 49 | Max Imaginary: 0.000092 at index: 30
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005412 | NRMSE Imaginary: 0.005453
```

Figure 5.10: FFT design functional verification result - Version: SCALE

```
Satellite 24 | Max Real: 0.000092 at index: 60 | Max Imaginary: 0.000092 at index: 30
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005258 | NRMSE Imaginary: 0.005340

Satellite 25 | Max Real: 0.000092 at index: 46 | Max Imaginary: 0.000092 at index: 10
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005287 | NRMSE Imaginary: 0.005392

Satellite 26 | Max Real: 0.000122 at index: 1 | Max Imaginary: 0.000092 at index: 54
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005474 | NRMSE Imaginary: 0.005256

Satellite 27 | Max Real: 0.000092 at index: 28 | Max Imaginary: 0.000092 at index: 58
             | RMSE Real: 0.000023 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005259 | NRMSE Imaginary: 0.005536

Satellite 28 | Max Real: 0.000092 at index: 27 | Max Imaginary: 0.000092 at index: 46
             | RMSE Real: 0.000023 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005186 | NRMSE Imaginary: 0.005341

Satellite 29 | Max Real: 0.000092 at index: 26 | Max Imaginary: 0.000092 at index: 30
             | RMSE Real: 0.000023 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005423 | NRMSE Imaginary: 0.005275
```

Figure 5.11: FFT design functional verification result - Version: SCALE (cont)

```
Satellite 30 | Max Real: 0.000122 at index: 1 | Max Imaginary: 0.000092 at index: 27
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000024 | NRMSE Real: 0.005259 | NRMSE Imaginary: 0.005556

Satellite 31 | Max Real: 0.000092 at index: 23 | Max Imaginary: 0.000092 at index: 35
             | RMSE Real: 0.000024 | RMSE Imaginary: 0.000023 | NRMSE Real: 0.005497 | NRMSE Imaginary: 0.005128

------------------------------------------------------------

Max diff all time | Real: 0.000122 at index 1 of Satellite 30 | Imaginary: 0.000122 at index 1 of Satellite 14

Simulation Success!

##########################################################
$finish called at time : 1393390 ns : File "D:/Capstone Project/gps_sdr/gps_on_fpga/gps_rtl_tb/gps_acquisition/rtl
```

Figure 5.12: FFT design functional verification result - Version: SCALE (cont)

For the SCALE configuration, we can observe that the NRMSE values are much larger than in the case of NON_SCALE configuration. They are approximately 0.5 to 0.6% of the mean value of the FFT output. However, 0.5 or 0.6 % are still acceptable ratios for the error.

INVERSE FAST FOURIER TRANSFORM VERIFICATION

We also repeated the process of testing the FFT design to test the IFFT design. One thing to note is that we do not have two configurations of the IFFT design as in the FFT design. The default mode for the IFFT design is SCALE due to its formula. Figures 5.13, 5.14, 5.15, 5.16, 5.17, 5.18 show us the result for the functional verification of IFFT design.


```
---------- Max diff (for each satellite) ----------

Satellite 0 | Max Real: 0.000153 at index: 3991 | Max Imaginary: 0.000092 at index: 17
            | RMSE Real: 0.000040 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.987455

Satellite 1 | Max Real: 0.000122 at index: 3931 | Max Imaginary: 0.000092 at index: 29
            | RMSE Real: 0.000039 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000049 | NRMSE Imaginary: 1.990723

Satellite 2 | Max Real: 0.000122 at index: 4047 | Max Imaginary: 0.000092 at index: 20
            | RMSE Real: 0.000041 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000052 | NRMSE Imaginary: 1.987700

Satellite 3 | Max Real: 0.000153 at index: 2541 | Max Imaginary: 0.000122 at index: 1
            | RMSE Real: 0.000040 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000051 | NRMSE Imaginary: 1.948705

Satellite 4 | Max Real: 0.000122 at index: 4001 | Max Imaginary: 0.000092 at index: 41
            | RMSE Real: 0.000040 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000051 | NRMSE Imaginary: 1.966318

Satellite 5 | Max Real: 0.000153 at index: 3040 | Max Imaginary: 0.000122 at index: 2
            | RMSE Real: 0.000041 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000052 | NRMSE Imaginary: 1.980767
```

Figure 5.13: IFFT design functional verification result

```
Satellite 6 | Max Real: 0.000122 at index: 4071 | Max Imaginary: 0.000122 at index: 3
            | RMSE Real: 0.000040 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.923258

Satellite 7 | Max Real: 0.000122 at index: 3832 | Max Imaginary: 0.000092 at index: 52
            | RMSE Real: 0.000038 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000049 | NRMSE Imaginary: 1.974524

Satellite 8 | Max Real: 0.000153 at index: 761 | Max Imaginary: 0.000122 at index: 1
            | RMSE Real: 0.000042 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000054 | NRMSE Imaginary: 2.007532

Satellite 9 | Max Real: 0.000122 at index: 3797 | Max Imaginary: 0.000092 at index: 27
            | RMSE Real: 0.000040 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.998751

Satellite 10 | Max Real: 0.000153 at index: 3071 | Max Imaginary: 0.000092 at index: 31
             | RMSE Real: 0.000040 | RMSE Imaginary: 0.000020 | NRMSE Real: 0.000051 | NRMSE Imaginary: 1.999288

Satellite 11 | Max Real: 0.000122 at index: 4083 | Max Imaginary: 0.000092 at index: 59
             | RMSE Real: 0.000037 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000047 | NRMSE Imaginary: 1.972910
```

Figure 5.14: IFFT design functional verification result (cont)

```
Satellite 12 | Max Real: 0.000153 at index: 767 | Max Imaginary: 0.000092 at index: 62
             | RMSE Real: 0.000039 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000049 | NRMSE Imaginary: 1.993264

Satellite 13 | Max Real: 0.000153 at index: 3069 | Max Imaginary: 0.000092 at index: 27
             | RMSE Real: 0.000041 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000052 | NRMSE Imaginary: 1.939764

Satellite 14 | Max Real: 0.000153 at index: 4076 | Max Imaginary: 0.000092 at index: 63
             | RMSE Real: 0.000041 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000052 | NRMSE Imaginary: 1.987971

Satellite 15 | Max Real: 0.000153 at index: 2895 | Max Imaginary: 0.000092 at index: 29
             | RMSE Real: 0.000040 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000051 | NRMSE Imaginary: 1.914850

Satellite 16 | Max Real: 0.000153 at index: 1973 | Max Imaginary: 0.000092 at index: 27
             | RMSE Real: 0.000041 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000051 | NRMSE Imaginary: 1.923948

Satellite 17 | Max Real: 0.000122 at index: 3859 | Max Imaginary: 0.000092 at index: 17
             | RMSE Real: 0.000040 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.840625
```

Figure 5.15: IFFT design functional verification result (cont)


```
Satellite 18 | Max Real: 0.000122 at index: 4024 | Max Imaginary: 0.000122 at index: 1
             | RMSE Real: 0.000038 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000048 | NRMSE Imaginary: 1.958865

Satellite 19 | Max Real: 0.000122 at index: 3051 | Max Imaginary: 0.000092 at index: 30
             | RMSE Real: 0.000040 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.852721

Satellite 20 | Max Real: 0.000153 at index: 900 | Max Imaginary: 0.000092 at index: 30
             | RMSE Real: 0.000041 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000052 | NRMSE Imaginary: 1.948456

Satellite 21 | Max Real: 0.000122 at index: 4051 | Max Imaginary: 0.000092 at index: 28
             | RMSE Real: 0.000040 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000051 | NRMSE Imaginary: 1.907134

Satellite 22 | Max Real: 0.000122 at index: 3951 | Max Imaginary: 0.000092 at index: 30
             | RMSE Real: 0.000038 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000048 | NRMSE Imaginary: 1.993827

Satellite 23 | Max Real: 0.000153 at index: 4023 | Max Imaginary: 0.000092 at index: 26
             | RMSE Real: 0.000041 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000052 | NRMSE Imaginary: 2.012681
```

Figure 5.16: IFFT design functional verification result (cont)

```
Satellite 24 | Max Real: 0.000183 at index: 924 | Max Imaginary: 0.000092 at index: 28
             | RMSE Real: 0.000043 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000054 | NRMSE Imaginary: 1.889616

Satellite 25 | Max Real: 0.000153 at index: 4073 | Max Imaginary: 0.000092 at index: 40
             | RMSE Real: 0.000039 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.919838

Satellite 26 | Max Real: 0.000122 at index: 4078 | Max Imaginary: 0.000092 at index: 30
             | RMSE Real: 0.000039 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.918343

Satellite 27 | Max Real: 0.000153 at index: 4014 | Max Imaginary: 0.000092 at index: 56
             | RMSE Real: 0.000040 | RMSE Imaginary: 0.000022 | NRMSE Real: 0.000051 | NRMSE Imaginary: 1.946623

Satellite 28 | Max Real: 0.000122 at index: 4053 | Max Imaginary: 0.000092 at index: 56
             | RMSE Real: 0.000039 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.984444

Satellite 29 | Max Real: 0.000153 at index: 986 | Max Imaginary: 0.000092 at index: 31
             | RMSE Real: 0.000040 | RMSE Imaginary: 0.000021 | NRMSE Real: 0.000050 | NRMSE Imaginary: 1.974171
```

Figure 5.17: IFFT design functional verification result (cont)

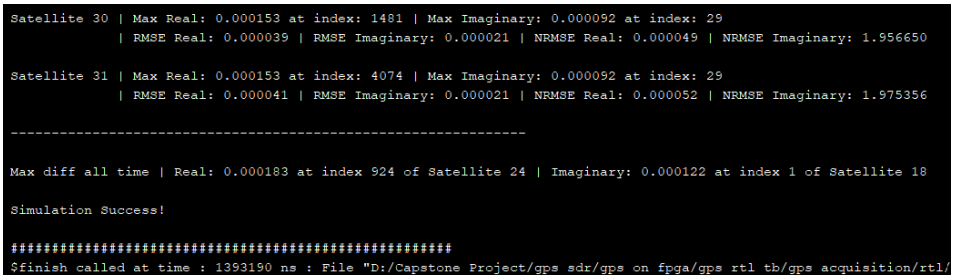


Figure 5.18: IFFT design functional verification result (cont)

In terms of IFFT design, we can easily see that the NRMSE values of the imaginary parts are really high (approximately 1.90 times or 190 % of the mean value of the output result). In the IFFT testing procedure, the input data are obtained from the FFT of PRN codes. PRN codes are purely real-valued signals (their imaginary parts are zero). Therefore, the expected output of the IFFT computed using Matlab's IFFT function is also a real-valued signal. In Matlab, we use this code ifft(fft(prn_code)) and they will definitely return prn_code.

However, because of the usage of fixed-point arithmetic in our IFFT design, it is inevitable that small errors occur during complex computations. Consequently, the imaginary part of our IFFT design's output, though theoretically zero, contains small values that are really close to zero due to rounding effects. Therefore, in this case, considering RMSE and NRMSE does not seem to be a good choice to evaluate the error. Instead, we evaluate the error based on the maximum difference between the expected and actual imaginary parts' result and all of them are close to 0.

ACCUMULATOR AND CORDIC ALGORITHM VERIFICATION

These two blocks will operate together in our design, so we will not verify them separately. Instead, we integrate them into one module and then run the testbench to test their functionalities. The testing procedure is also similar to previous FFT and IFFT verifications. We also computed the expected output results using Matlab and compared them with the actual output from our accumulator - cordic integration design. Figures 5.19, 5.20, 5.21, 5.22, 5.23, 5.24, 5.25 show us the result for the accumulator - cordic design verification.

```
---------------- Max diff (for each satellite) ----------------

Doppler ID 0 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 1 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000021 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 2 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 3 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 4 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 5 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000021 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034
```

Figure 5.19: Accumulator - Cordic functional verification result


```
Doppler ID 6 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 7 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 8 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 9 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
             | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000035

Doppler ID 10 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000021 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 11 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034
```

Figure 5.20: Accumulator - Cordic functional verification result (cont)

```
Doppler ID 12 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 13 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 14 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000021 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 15 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 16 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 17 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034
```

Figure 5.21: Accumulator - Cordic functional verification result (cont)


```
Doppler ID 18 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 19 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 20 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 21 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 22 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 23 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034
```

Figure 5.22: Accumulator - Cordic functional verification result (cont)

```
Doppler ID 24 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 25 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 26 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 27 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000021 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 28 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 29 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034
```

Figure 5.23: Accumulator - Cordic functional verification result (cont)

```
Doppler ID 30 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 31 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 32 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 33 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 34 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 35 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034
```

Figure 5.24: Accumulator - Cordic functional verification result (cont)

```
Doppler ID 36 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 37 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000022 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 38 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000021 | RMSE Sin: 0.000021 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000034

Doppler ID 39 | Max Cos: 0.000031 at index: 1 | Max Sin: 0.000031 at index: 1
              | RMSE Cos: 0.000022 | RMSE Sin: 0.000021 | NRMSE Cos: 0.000034 | NRMSE Sin: 0.000033
```

Figure 5.25: Accumulator - Cordic functional verification result (cont)

One thing to note here is the Doppler ID. The Doppler ID starts from 0 to 39 corresponding to Doppler shift bins from +10000 to +250 Hz. We do not test the Doppler shift bins from -250 to -10000 Hz because they are actually the Doppler shift bins from +250 to +10000 Hz but their sine values are reversed in signs. With the Doppler shift bin at 0 Hz, we also do not test because at this Doppler shift value, the received signal is not shifted, so we use directly the input data in our GPS acquisition top design. The RMSE and NRMSE values are also very small for the accumulator and cordic designs.

GPS ACQUISITION VERIFICATION

Now, it is time for us to come to the main part, which is to integrate all sub-modules into the GPS acquisition top module and add some control logic to control the data flow. Then, we will test the functionality of the GPS acquisition design.

To verify the functionality of the GPS acquisition design, we built a GPS acquisition model in Matlab using built-in functions such as fft and ifft. We pro-

vide the generated data sets (see Section 5.1) to the Matlab model and to our design and check the results to see whether they match. The model in Matlab is also built using the PCS method with two times fft and one time ifft.

We verified our GPS acquisition with three main data sets. However, we will only show the result for one of them here to keep the length of the content of the report short in control. Figure 5.26 presents the expected result of GPS acquisition. Theoretically, a GPS receiver needs to acquire and track at least four satellites, but we also show here the fifth acquired satellites to see if our GPS acquisition design can even acquire it.

```
Top PRN ID
TOP 1: PRN ID 1 - Doppler Shift 7000 - Code Delay 4084
TOP 2: PRN ID 32 - Doppler Shift 6250 - Code Delay 3780
TOP 3: PRN ID 17 - Doppler Shift 9750 - Code Delay 413
TOP 4: PRN ID 11 - Doppler Shift 5500 - Code Delay 364
TOP 5: PRN ID 20 - Doppler Shift 8500 - Code Delay 1282
```

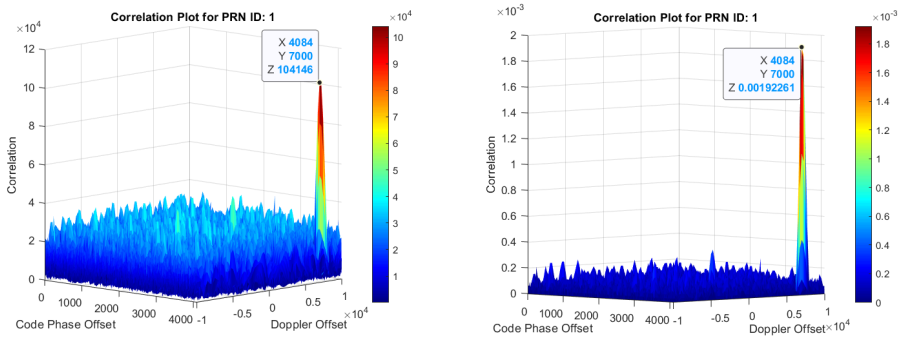
Figure 5.26: GPS Acquisition result from Matlab model

The result of the verification of our GPS acquisition design is shown in Figure 5.27. Our results match correctly the result from the Matlab model. One thing to note here is the values of the code phase delay. You can observe that our results are less than the Matlab model results by 1 unit. This is because in Matlab, indexing starts from 1 instead of 0. This will lead to the difference in the code phase delay range of our design and the Matlab model (0 to 4095 for our design and 1 to 4096 for the Matlab model).

```
############### TOP 5 PEAK ###############

[========== TOP 1 ==========]

PRN ID: 1 - Doppler Shift: 7000 - Code Phase Delay: 4083

[========== TOP 2 ==========]

PRN ID: 17 - Doppler Shift: 9750 - Code Phase Delay: 412

[========== TOP 3 ==========]

PRN ID: 32 - Doppler Shift: 6250 - Code Phase Delay: 3779

[========== TOP 4 ==========]

PRN ID: 11 - Doppler Shift: 5500 - Code Phase Delay: 362

[========== TOP 5 ==========]

PRN ID: 20 - Doppler Shift: 8500 - Code Phase Delay: 1281
```

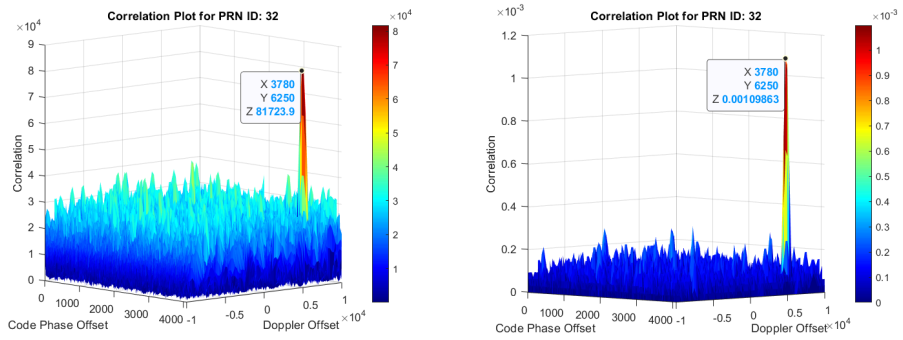
Figure 5.27: GPS Acquisition result from Matlab model

We also developed a short Matlab code to plot the correlation results in the search space for the results of the Matlab model and our GPS acquisition design to compare them. The plotted figures are presented in Figures 5.28, 5.29, 5.30, 5.31, and 5.32.



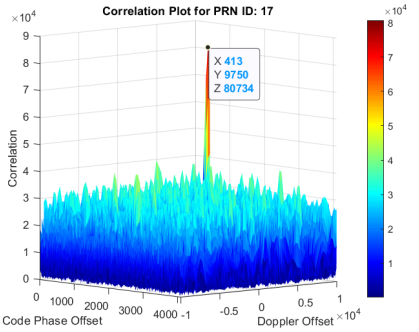
(a) Matlab model (b) Our design

Figure 5.28: Plotted correlation results of Matlab model versus our design (Top 1)

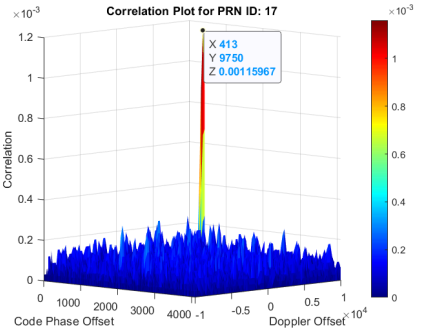


(a) Matlab model (b) Our design

Figure 5.29: Plotted correlation results of Matlab model versus our design (Top 2)



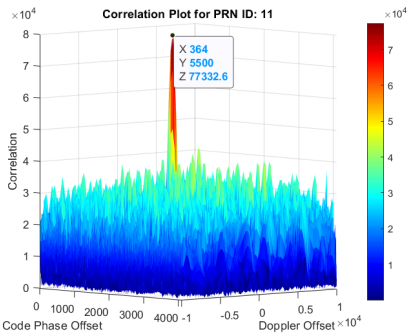
(a) Matlab model



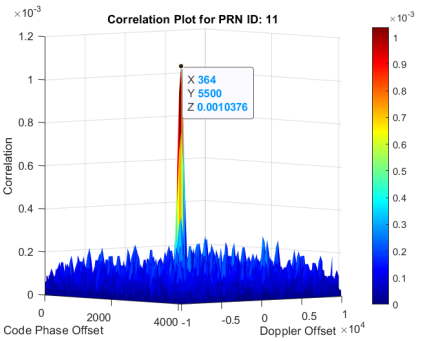
(b) Our design

Figure 5.30: Plotted correlation results of Matlab model versus our design (Top 3)

5

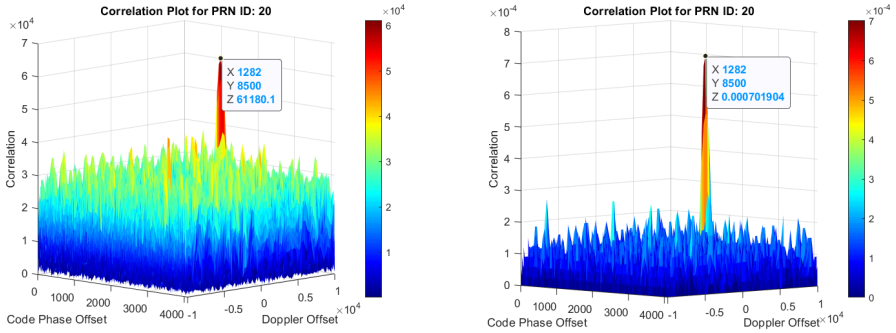


(a) Matlab model



(b) Our design

Figure 5.31: Plotted correlation results of Matlab model versus our design (Top 4)



(a) Matlab model

(b) Our design

Figure 5.32: Plotted correlation results of Matlab model versus our design (Top 5)

The plotted figures have already clearly presented the concept of cross-correlation. It is evident that both figures correctly identify the correlation peak at the same coordinate location in the two-dimensional search space. This indicates that our GPS acquisition design performs accurately and consistently with the theoretical model built in Matlab. If we look at other coordinate locations, we can see that the correlation results are in blue, which means the result is very low. We mean that if the correlation results are computed at incorrect combination between the code phase delay and Doppler shift values, the correlation results will be really low. The correlation results in the area close to the correct position will have higher values (in colors such as yellow, orange, and red). The sharp peak occurs only at the correct position.

However, there is a noticeable difference in the peak correlation amplitudes. The correlation values of the Matlab model is much bigger than our design's ones. This is simply because in the top module integration of the GPS acquisition design, we shifted some intermediate results of the complex multiplication in frequency domain before pushing them to the IFFT core design to guarantee that the problem of overflow will not occur. In general, they do not affect our results, and we can clearly see that the peak still occurs at the correct position, but just in small values.

All the figures and correlation results of GPS acquisition functional verification that we have presented so far are configured with the option NON_SCALE. Figures 5.33, 5.34, 5.35, 5.36, and 5.37 present the plotted correlation results of our design with the version of SCALE FFT.

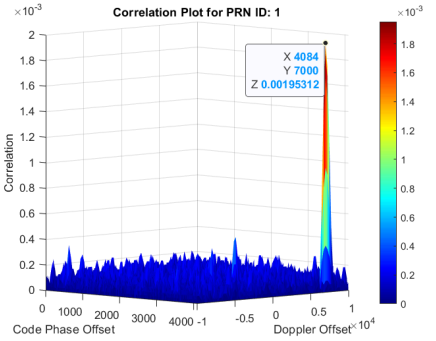


Figure 5.33: Correlation results of our design - Version: SCALE (TOP 1)

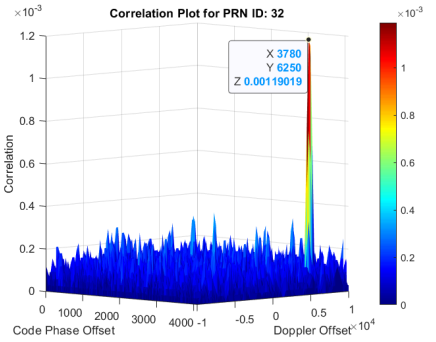


Figure 5.34: Correlation results of our design - Version: SCALE (TOP 2)

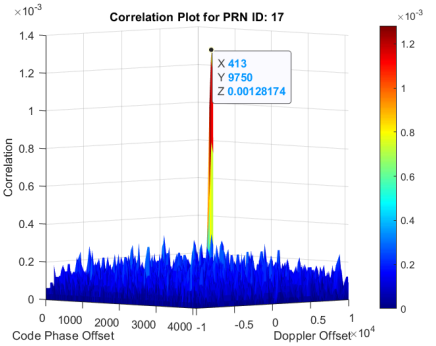


Figure 5.35: Correlation results of our design - Version: SCALE (TOP 3)

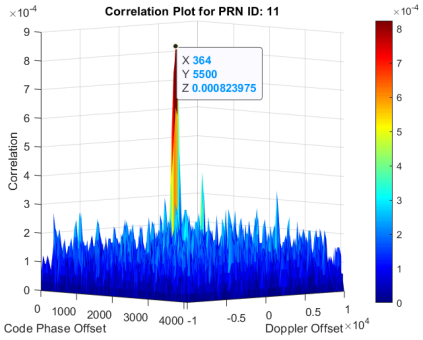


Figure 5.36: Correlation results of our design - Version: SCALE (TOP 4)

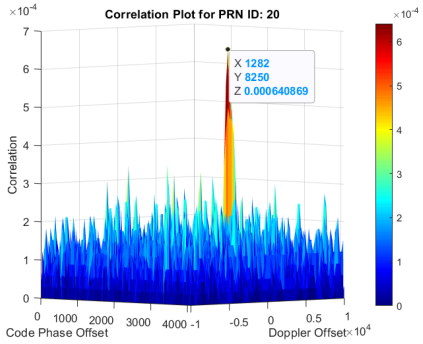


Figure 5.37: Correlation results of our design - Version: SCALE (TOP 5)

Everything is fine except for the top 5 correlation result. From Figure 5.37, we can see that the PRN ID and the code phase delay value match our expected results, which are PRN ID = 20 and code phase delay = 1282. However, the Doppler shift value is 8250 while we expect it to be 8500 as in the results of our Matlab model. There are some reasons for this error listed below, but our target is to acquire at least four satellites, and this error still does not affect our result.

- The correct Doppler shift value is in the middle range between 8500 and 8250 Hz, and this leads to multiple peaks occurrence.

- FFT scaling leads to a reduction in the amplitude resolution. This causes the problem of multiple peaks occurring around the area of the correct peak, and it is more difficult to distinguish peak values.

- Errors caused by quantization, or rounding in FFT scaling is more significant than FFT non-scaling. In general, these errors are still not so important for us to care if the signals are strong enough. However, in case the signals are weak or contain a lot of noise, these errors can lead to wrong Doppler shift estimation.

Until now, we have shown the working operations and correctness of the two configurations SCALE and NON_SCALE for our GPS acquisition design. These two options have a tradeoff and depending on each situation, we can apply different choices.

- **NON_SCALE version:** High accuracy but poorer resource utilization (more BRAM utilization). This version is suitable for GPS applications frequently operating in noisy environments.

- **SCALE version:** Lower accuracy but better resource utilization (lower BRAM utilization). This version is suitable for GPS applications frequently operating in less noisy or outdoor environments.

In terms of how the SCALE version is better in resource utilization than the NON_SCALE version, we will present it in the next section 5.3.

5.2.2. GPS TRACKING PHASE

It is very difficult to verify the GPS tracker with a different models (like one created on MATLAB) due to the differences in filter and tracking loop timing that lead to different navigation message output. However, GPS navigation message has the preamble 10001011 embedded inside so it can be used to verify whether or not the tracker design could properly lock on to the incoming signal that we can use to verify our design.

```
# 10011101100000000110100100010001011011010100110111000000111101000100010011001110010010
############################################################
#
# Simulation success!
#
############################################################
#
# ** Note: $finish    : ./gps_tracker_channel_tb.sv(104)
#    Time: 1708981464840 ps  Iteration: 1  Instance: /gps_tracker_channel_tb
```

Figure 5.38: Example navigation message result with a test file with known incoming signal at PRN ID 11, Doppler shift 5500 Hz and code delay of 350 chips

```
# 1100110100010111101101110100010100010110110111011000110000110110001000011000011000100
############################################################
#
# Simulation success!
#
############################################################
#
# ** Note: $finish    : ./gps_tracker_channel_tb.sv(104)
#    Time: 1708981464840 ps  Iteration: 1  Instance: /gps_tracker_channel_tb
```

Figure 5.39: Example navigation message result with a test file with known incoming signal at PRN ID 20, Doppler shift 8250 Hz and code delay of 1243 chips

However, the resulting navigation message can be detected in reality later than on the test bench because of the difference in phase of the signal, as the trackers need to wait for the result from the acquisition and sorting process.

5.3. SYNTHESIS & IMPLEMENTATION RESULTS

5.3.1. SYNTHESIS & IMPLEMENTATION

The implemented design can be configured to use 2 different versions of FFT: scaled and non-scaled version with different resource usage. A comparison for our synthesis and implementation run with the clocking wizard set to generate 200 MHz and targeting the Ultra96-v2 board:

	Scale	Percentage	Non-scale	Percentage
LUT	31556	44.72%	28718	40.7%
LUTRAM	2375	8.25%	2133	7.41%
FF	28649	20.3%	27960	19.81%
BRAM	167.5	77.55%	182.5	84.49%
DSP48	254	70.56%	254	70.56%
BUFG	2	1.02%	2	1.02%
MMCM	1	33.33%	1	33.33%

Table 5.1: Comparison of synthesis and implementation result of scale and non-scale version

The scaled FFT version used a lot less BRAM (167.5 compared to 182.5 of non-scaled version) at the cost of a lot more LUT, LUTRAM and FF use (31556 LUT, 2375 LUTRAM and 28649 FF for the scaled version compared to 28718 LUT, 2133 LUTRAM and 27960 FF for non-scaled version). The usage of other important resources like DSP48, BUFG and MMCM is the same for both versions. It is noticable that our design uses a huge amount of BRAM, the reason is that we need to store 32 C/A Code mem files for 32 satellites with each file has a size of 64KB. Moreover, the amound of DSP used in our design is also quite high, this is because we need to do a lot of multiplication operator when dealing with

FFT/IFFT problems.

In addition, the timing results between the 2 are different, with the Scale version having a lot better timing than Non-scale version, having higher WNS:

Version	WNS (ns)	WHS (ns)	WPWS (ns)
Scale	0.037	0.010	1.000
Non-scale	0.001	0.009	1.000

Table 5.2: Comparison of timing result of scale and non-scale version

In the end, since the value of WNS is relatively very small (as our system is currently designed to run with the frequency of $200Mhz$, it indicates that our system is operating close to its timing limit. Therefore, it is not recommended to increase the operation frequency for our system at the moment (to further increase the clock frequency, the system would need to be redesigned or upgraded to reduce critical path delays and maintain setup timing).

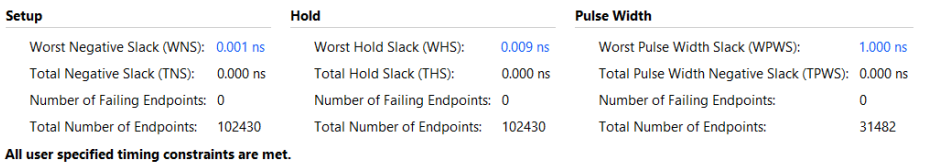


Figure 5.40: Vivado Timing Report Non-scale Version

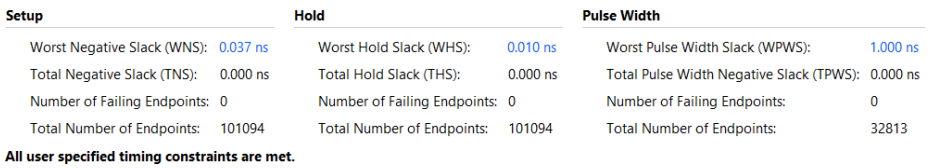


Figure 5.41: Vivado Timing Report Scale Version

However, for power report, both implementations have nearly similar power usage and junction temperature:

	Total power (W)	Junction temperature (C)
Scale	3.307	34.0
Non-scale	3.375	34.2

Table 5.3: Comparison of power report of scale and non-scale version

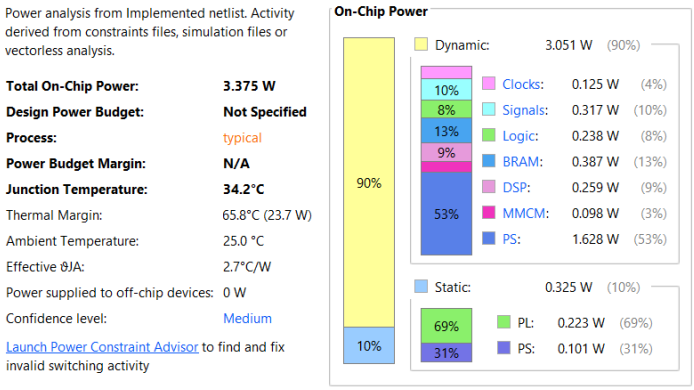


Figure 5.42: Vivado Power Report Non-scale Version

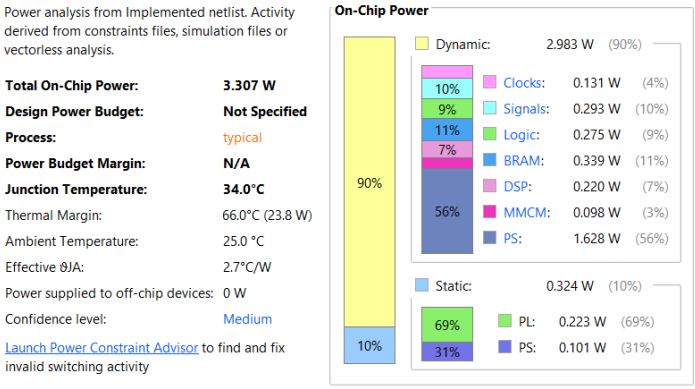
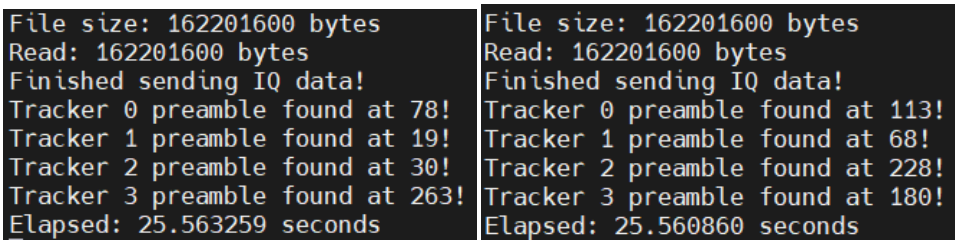


Figure 5.43: Vivado Power Report Scale Version

From Fig 5.42 and Fig 5.43, BRAM power usage is huge, as explained we use a lot of BRAM in our design. One noticable fact is in our design, Dynamic power is huge (90%) compared to only 10% of Static power.

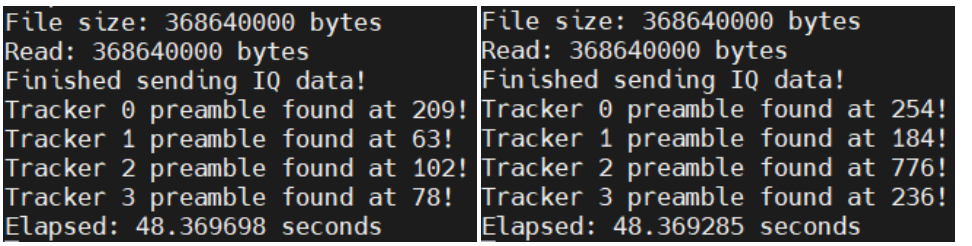
5.3.2. BOARD RUN RESULT

To run the design on the Ultra96-v2 board, 2 test files were prepared: gps_10s.bin and spain_gps_10s.mem. Both of these files are stored on a microSD card, the PS then reads the file and stores it into an array before pushing the data out into the PL through the DMA. The resulting navigation message received from the PL is stored in another array and finally processed to find the preamble to verify the design.



(a) Non-scale (b) Scale

Figure 5.44: Board run result of file gps_10s.bin



(a) Non-scale (b) Scale

Figure 5.45: Board run result of file spain_gps_10s.mem

In the run result of both files, the scale FFT design version always resulted in the preamble to be detected a lot later. This came from the fact that the lower accuracy of the scale FFT resulted in the tracker taking a lot longer to acquire good lock on the signal to get the valid data.

5.4. PERFORMANCE EVALUATION

The performance of a GPS receiver is based on the accuracy of the GPS acquisition phase and also the acquisition time. After researching and designing the GPS acquisition phase for a long time, we conclude that three available GPS acquisition methods have reached their performance limitations. For the SS method, it is really old and is poor in performance, so we will not consider it here. For the PFS and PCS methods, they utilize the FFT and IFFT concepts in their designs, and the running time is mainly from this IFFT core. Undoubtedly, in the scenarios where hardware resources are limited and duplicating the design is not feasible, the PFS and PCS algorithms appear to have reached their performance limit in terms of acquisition time. In our case, the 4096-point IFFT core must operate $81 \times 32 = 2592$ times to compute all correlation results. Therefore, regardless of how efficiently the surrounding GPS acquisition stages are implemented, the IFFT core remains a dominant factor contributing to the

acquisition time. From our point of view, there are two ways to speed up the GPS acquisition running time.

- Duplicate the design to support the acquisition of multiple satellites simultaneously: This method can be applied if the design is implemented on resource-rich FPGA boards.
- Find a totally different method from PFS and PCS or try to find a new algorithm to speed up computing the DFT even better than the FFT algorithm nowadays.

We suppose the configuration of the setup for the GPS acquisition phase as follows.

- Search for 1 period of PRN code only (1 millisecond).
- Doppler shift ranges from -10 to +10 kHz, with a step of 250 Hz (total 81 Doppler shift values to search).
- The code phase delay varies from 0 to 4095 due to the sampling frequency of 4.096 MHz.

Table 5.4 shows the running time of the GPS acquisition phase of our design compared to other designs.

	Clock frequency (MHz)	Acquisition time (ms)	Hardware Platform
Our design	200	53.167	Ultra96-v2 FPGA
Matlab*	-	783	Intel Core i5-11400H
Reference [26]	100	630	Virtex-II Pro FPGA
Reference [27]*	-	32	SFF-SDR Lyrtech
Reference [28]	56.14	93.376	Virtex-II Pro FPGA
Reference [7]	160	66.355	Altera Cyclone III EP3C120

Table 5.4: GPS acquisition performance comparison

We do not have much helpful information from the article Reference [27]*. The author only shows that the time for GPS acquisition using the PCS method is only 1 millisecond. Moreover, 1 millisecond is also not mentioned for all satellites or only one satellite. We assume that the author meant 1 millisecond for one satellite only, so the total acquisition time for the entire satellite constellation is 32 milliseconds. This acquisition time is significantly fast and outperforms all other proposed methods that we referenced, and even our design.

However, as shown, our GPS acquisition design still generally has an acceptable acquisition time if we compared it with other designs.

We also tried to measure the time for our Matlab model acquisition design to complete its working operation using the tic and toc functions. The value of the acquisition time in this case can vary depending on how free the CPU is. We observe that if we open another application while executing the Matlab model, the elapsed time increases. 783 milliseconds is the elapsed time of the Matlab model running on a member's laptop in our group when he only runs Matlab and closes all other applications. His processor is 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70 GHz (12 CPUs) together with 16 GB RAM. This configuration is also strong, but the time for acquisition to complete can be up to hundreds of milliseconds. That is the reason why we need to bring this computationally expensive process to hardware to accelerate it. This is only 1 millisecond search, so 783 milliseconds is not a big problem, but when we expand the search time to multiple periods, the acquisition time on the software will significantly increase.

Although our performance is better than some other proposals in a single PRN code period (1 millisecond) compared to other existing methods as shown in table 5.4, the difference is only on the order of tens of milliseconds, which is not quite significant. In the future, when we extend the search over multiple periods of the PRN code (multiple milliseconds), the total acquisition search time will increase proportionally, but still remains below a few seconds. Supposing that we extend the search over 20 periods of the PRN code, our design's acquisition time will be approximately 1 second while the acquisition times of Reference [28] and Reference [7] are, respectively, approximately 1.87 and 1.33 seconds. The difference between our design and theirs is below 1 second in this case. This difference will be significant in some certain specialized applications where a fast acquisition time is a critical requirement. These applications are usually related to military use, such as GPS in unmanned aerial vehicles (UAVs). With normal civilian use, it is not a critical requirement to have such a GPS acquisition design that can complete the acquisition phase of all satellites over multiple periods with a very high accuracy in only milliseconds.

6

FUTURE WORK

6.1. SOFTWARE-DEFINED RADIO

Since the input for our GPS design is a raw baseband signal, in real life this is not a case when the real GPS signal always contain carrier frequency that is usually very high. Integrating our design with a SDR to downconvert the original raw GPS signal is a next step in our development for this project. In fact, we have already developed and implemented a simple SDR architecture as shown in Fig 6.1 below (successfully complete submodules verification), but we faced a challenge of gathering real raw GPS signal data to test our SDR design (since it usually requires hardware component to capture).

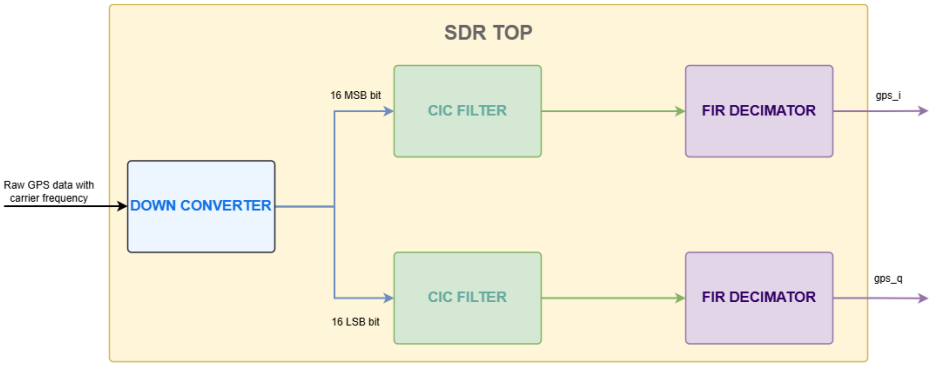


Figure 6.1: Proposed SDR Architecture

6.2. GPS ACQUISITION

6.2.1. DEVELOPING NON-COHERENT AND COHERENT INTEGRATION

One promising future improvement for our GPS acquisition design is to add non-coherent and coherent integration concepts. We mentioned earlier the problem of the NON_SCALE GPS acquisition design, which is the accuracy. Theoretically, we need to receive all samples in only one single period of the received signal and then try to acquire satellites using those samples. However, when the signal is weak or contains a lot of noise, the peak value will be really low and sometimes there will be multiple peaks having similar results. We call this the false-peak selection problem. Our design right now also searches for one single PRN code period only, and this means that to ensure that our design can work further in noisier environments, we need some mechanisms to improve the accuracy of the GPS acquisition results.

Non-coherent and coherent integration mechanisms are the correct answers. Coherent integration allows the correlation process to accumulate the output results over multiple PRN code periods, which leads to the improvement in the signal-to-noise ratio (SNR) in the scenarios with weak or attenuated signals. However, the coherent integration mechanism also has a weak point that is the problem of data bit transition. We will not discuss the issue of data bit transition further, but conceptually, this issue means that if unfortunately during the process of performing coherent integration, the value of a data navigation bit changes, it will lead to the problem that the correlation values when being accumulated will cancel each other out due to their phase reversal, leading to a lower coherent integration result than expected. One solution for coherent integration to work is to limit the search time below 20 milliseconds. This means that the GPS acquisition design can only search for a maximum of 20 continuous PRN code periods with the coherent integration mechanism.

Non-coherent integration mechanism can address the problem of coherent integration. In this mechanism, the magnitude of multiple coherent correlation results will be summed, enabling longer effective integration times without requiring strict carrier phase alignment. Furthermore, the trade-offs between integration time, hardware resource usage, and acquisition latency should be carefully explored in the context of FPGA implementation.

6.2.2. NEW PROPOSED GPS ACQUISITION ARCHITECTURES

We observe that the resource utilization for our GPS acquisition design is still quite high. This is due to the FFT and IFFT cores in our design. Currently, we integrate two FFT instances and one IFFT instance in the GPS acquisition

top module. This architecture strictly follows the block diagram of the PCS acquisition method. However, we want to propose two new architectures for the GPS acquisition design with different trade-offs.

The first new architecture is to integrate one FFT instance and one IFFT instance. This will help us reduce one FFT instance compared to the current design. The resource utilization of the GPS acquisition is mainly from the FFT and IFFT cores, so reducing one FFT core will significantly reduce the resource utilization. This reduction is also logical because the FFT core for the received PRN code only works 81 times corresponding to the FFT of the received PRN code at 81 Doppler shift wipe-off times, while the FFT core for the locally generated PRN codes needs to work $81 \times 32 = 2592$ times. This reduction will also lead to a longer acquisition time, but not much. Currently, we need 10633347 cycles to complete the acquisition phase, and with this new architecture, we will need more $81 \times 4096 = 331776$ cycles or 0.00165888 seconds with our current clock frequency of 200 MHz. More specifically, with this first new architecture, the total acquisition time will be approximately 54.82 milliseconds. We have already prepared the overall block diagram for the new architecture in Figure 6.2. Because there is only one FFT instance and it has to compute the FFT of both received and locally generated PRN codes, we need two finite state machines before and after the FFT instance to manage the data.

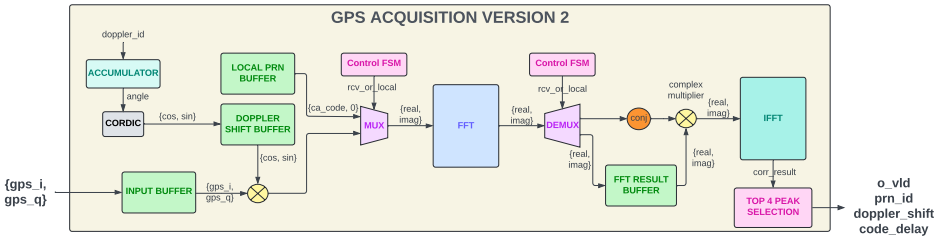


Figure 6.2: GPS acquisition design overview diagram - First new proposed architecture

The second new architecture is to integrate only one FFT instance. Because the designs of FFT and IFFT are only different from each other in the signs of the twiddle factors, if we can carefully manage the data flow with some more logic controls, it is possible that we can reduce to using only one FFT instance in the entire GPS acquisition design. We have also prepared the overall block diagram for the second new architecture in Figure 6.3. If we compared this second new proposed architecture to the first one, we can see that the IFFT instance is removed and there are two additional finite state machines (FSM). Two additional FSMs are responsible for controlling whether the input/output

data come to/from the FFT or IFFT core. With this second new architecture, resource utilization will be significantly reduced compared to the current GPS acquisition design. However, there is a really big trade-off for this architecture that is the acquisition time will be doubled. We need to compute the FFTs of the received and locally generated PRN code, together with the IFFT of the result of the multiplication after two FFTs using only one FFT instance. In particular, the FFT instance in this case needs to work for $32 \times 81 + 81 + 32 \times 81 = 5625$ times, or equivalently, 21565440 cycles plus some delay cycle at the beginning of the FFT. The acquisition time in this case will be around 107 milliseconds, which is double of the current version.

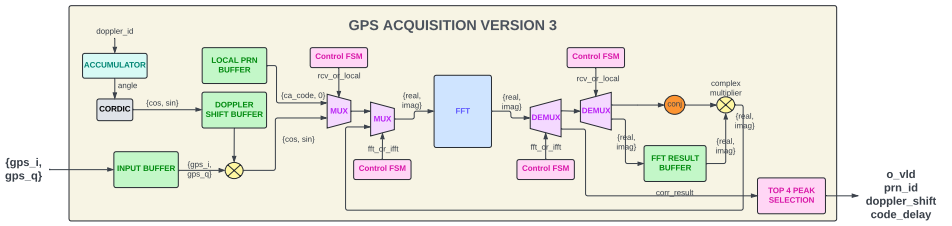


Figure 6.3: GPS acquisition design overview diagram - Second new proposed architecture

In conclusion, the first proposed new architecture would be the best version of GPS acquisition design that can provide the balance between resource utilization and performance. However, the second one also provides a resource saving option for users who have a low-resource FPGA. Although the acquisition time is doubled, it is still acceptable for civilian use.

6.3. NAVIGATION MESSAGE DECODING

With the output from the GPS tracker received, the navigation message can be extracted by synchronizing the message using the preamble; the next step is to decode the navigation message to obtain the required orbital information of each satellite and find our coordinates. In a complete navigation message, there are a total of 25 frames, each frame divided into 5 subframes that consist of 10 words and each word consists of 30 bits:

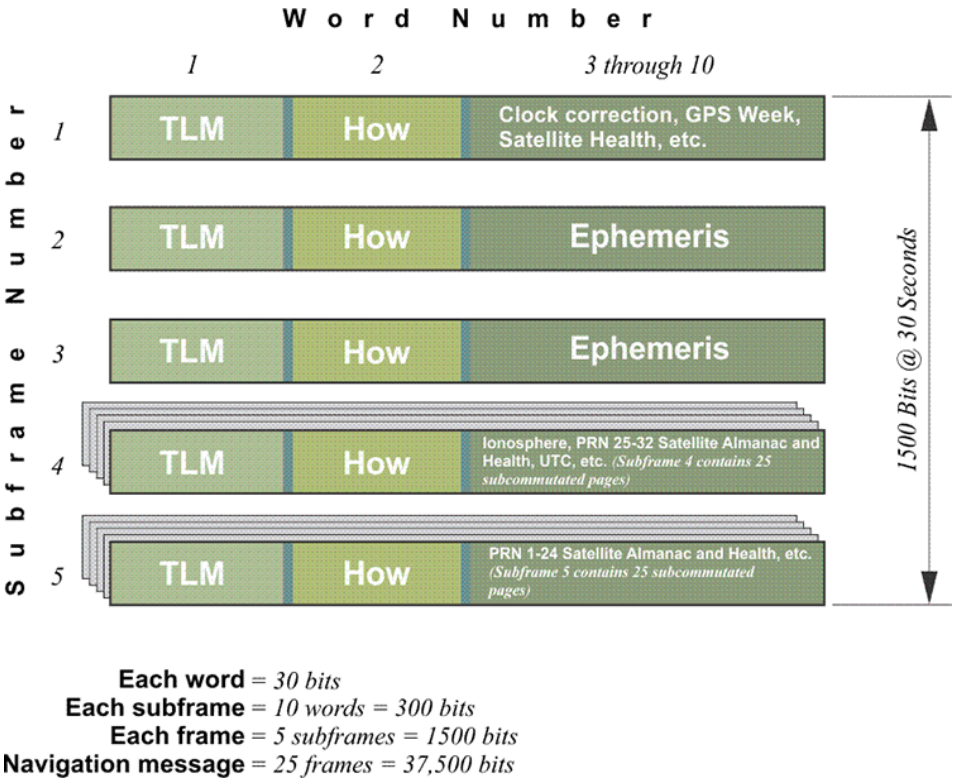


Figure 6.4: GPS navigation message frame structure[9]

All satellites status including velocity, position, health, etc can be extracted completely from navigation message. From there, we can calculate pseudorange as a first step to determine user position in ECEF coordinate system.

6.4. ADAPTATION TO GPS L2C

While GPS L1 C/A has been in use for a long time with numerous documents and reliable satellite connection, it is still prone to various problems, thus GPS L2C was created to improve GPS L1 C/A:

- The addition of a pilot signal. This signal does not carry any data, but it helped the receiver to synchronize with it. This greatly helps in an environment with very low SNR, such as in a forest or in a canyon.
- Additional improvement in navigation message structure, with the use of FEC to improve noise robustness.
- A lot of documentation on CM code used by GPS L2C, greatly simplified

the FPGA implementation.

- L2C receiver can be used together with L1 C/A receiver to increase resistance ionospheric delays and multipath interference, allowing for more precise positioning than L1 C/A alone.

While GPS L2C still retain several features that made it appealing to adapt to from GPS L1 C/A:

- Low sampling rate requirement that is similar to GPS L1 C/A. This means that the front end used to digitize the signal can be shared.
- GPS L2C uses BPSK modulation that is the same as GPS L1 C/A so there is no need to rewrite much of the tracker module.

7

CONCLUSION

In conclusion, this thesis demonstrated the feasibility and advantages of implementing a GPS receiver with an SoC such as the Zynq Ultrascale MPSoC+. By taking advantages of the programmable logic (PL) to implement important components like the GPS acquisition and tracking processes and using the processing system (PS) for the task of detecting the preamble of the navigation message, the system can achieve the desired performance even at 200 MHz while still retain the flexibility to configure the acquisition design to use scale or non-scale FFT. However, there are still many things to improve, like decoding the navigation message, upgrading to support lock loss mechanism in tracking process, or reducing a lot of resource usage from the acquisition stage. In addition, the possibility to adapt the system to modern GPS system like GPS L2C is also considered.

BIBLIOGRAPHY

[1] Seong Hun Seo, Byung Hyun Lee, Sung Hyuck Im, and Gyu In Jee. Effect of Spoofing on Unmanned Aerial Vehivle using Counterfeited GPS Signal. *Journal of Positioning, Navigation, and Timing*, 2015.

[2] U.S. Department of Defense. *Interface Specification: IS-GPS-200M.* Navstar GPS Space Segment, 2020. URL https://www.gps.gov/technical/icwg/IS-GPS-200M.pdf. Accessed: May 1, 2025.

[3] Science Ready HSC Resources. The doppler effect. URL https://scienceready.com.au/pages/dopplers-effect. Accessed: May 1, 2025.

[4] Thomas A Stansell, John W Betz, Frank van Diggelen, and Satoshi Kogure. Proposed evolution of the c/a signal. In *Proceedings of the 28th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2015)*, pages 1807–1825, 2015.

[5] Jérôme Leclère, Cyril Botteron, and Pierre-André Farine. Comparison framework of fpga-based gnss signals acquisition architectures. *IEEE Transactions on Aerospace and Electronic Systems*, 49(3):1497–1518, 2013.

[6] GMV. *Multicorrelator.* European Space Agency, 2011. URL https://gssc.esa.int/navipedia/index.php/Multicorrelator. Accessed: April 26, 2025.

[7] Jérôme Leclère. *Resource-efficient parallel acquisition architectures for modernized GNSS signals.* PhD thesis, EPFL, 2014.

[8] Adam M Shapiro. Fpga-based real-time gps receiver. *Dissertations, Cornell University*, 2010.

[9] Jan Van Sickle. Telemetry and handover words. https://www.e-education.psu.edu/geog862/node/1740. Accessed: May 1, 2025.

[10] J.A Ávila Rodríguez. *GPS Signal Plan.* European Space Agency, 2011. URL https://gssc.esa.int/navipedia/index.php/GPS_Signal_Plan. Accessed: May 1, 2025.

[11] The Aerospace Corporation. A brief history of gps. https://aerospace.org/article/brief-history-gps, n.d. Accessed: April 29, 2025.

[12] Wahyudin Syam. Generating GPS L1 C/A pseudo-random noise (PRN) code with MATLAB and C/C++, 2022. URL https://www.wasyresearch.com/generating-gps-l1-c-a-pseudo-random-noise-prn-code-with-matlab-and-c-c/. Accessed: May 1, 2025.

[13] Boya Prdaeep Kumar and Chandra Sekhar Paidimarry. Development and analysis of C/A code generation of GPS receiver in FPGA and DSP. *Recent Advances in Engineering and Computational Sciences (RAECS) Conference*, 2014.

[14] GNSS SDR. Acquisition. URL https://gnss-sdr.org/docs/sp-blocks/acquisition/#gps-l1-ca-signal-acquisition. Accessed: May 1, 2025.

[15] GMV. *Phase Lock Loop (PLL)*. European Space Agency, 2011. URL https://gssc.esa.int/navipedia/index.php?title=Phase_Lock_Loop_(PLL). Accessed: April 26, 2025.

[16] GMV. *Delay Lock Loop (DLL)*. European Space Agency, 2011. URL https://gssc.esa.int/navipedia/index.php/Delay_Lock_Loop_(DLL). Accessed: April 26, 2025.

[17] Xiang Gao, Li Xi, Rugui Yao, and Yong Li. A high precision acquisition algorithm of gps based on parallel frequency search. In *2016 25th Wireless and Optical Communication Conference (WOCC)*, pages 1–4. IEEE, 2016.

[18] Takuji Ebinuma. gps-sdr-sim: Gps signal simulator. https://github.com/osqzss/gps-sdr-sim, 2024. Accessed: April 25, 2025.

[19] Ranjita Naik and Riyazahammad Nadaf. Sine-cosine computation using cordic algorithm. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(9):44–48, 2015.

[20] Steve Arar. An introduction to the cordic algorithm. *AllAboutCircuits. com*, 2017. Accessed: May 1, 2025.

[21] Cebarnes. An implementation of the cordic algorithm in verilog. https://github.com/cebarnes/cordic, 2013.

[22] D. Gisselquist. An open source pipelined fft generator. https://zipcpu.com/dsp/2018/10/02/fft.html, oct 2018. Accessed: Apr. 28, 2025.

[23] D. Gisselquist. Rounding numbers without adding a bias. https://zipcpu.com/dsp/2017/07/22/rounding.html, jul 2017. Accessed: Apr. 29, 2025.

[24] NASA Crustal Dynamics Data Information System (CDDIS). GNSS Daily Broadcast Ephemeris Files (RINEX Navigation Files), 2025. URL https://cddis.nasa.gov/archive/gnss/data/daily/2025/brdc/. Accessed: April 29, 2025.

[25] Dominoc925. Coordinate systems: Convert geographic to ecef (wgs-84). https://dominoc925-pages.appspot.com/mapplets/cs_ecef.html, n.d. Accessed: April 29, 2025.

[26] Shankararaman Ramakrishnan, Grace Xingxin Gao, David De Lorenzo, Todd Walter, Per Enge, and Dennis Akos. Design and analysis of reconfigurable embedded gnss receivers using model-based design tools. In *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008)*, pages 2293–2303, 2008.

[27] Kota Solomon Raju, Y Pratap, Virendra Patel, SMM Naidu, Amit Patwardhan, and P Bhanu Prasad. Acquisition digital baseband module for multichannel gps receiver. In *2012 International Conference on Industrial Control and Electronics Engineering*, pages 9–13. IEEE, 2012.

[28] Eduardo Romero-Aguirre, Ramón Parra-Michel, O Longoria-Gandara, and M Aguirre-Hernandez. A hardware-efficient frequency domain correlator architecture for acquisition stage in gps. In *2010 International Conference on Reconfigurable Computing and FPGAs*, pages 412–417. IEEE, 2010.