FPGA PROTOTYPING OF GPS L1 SIGNAL ACQUISITION USING HDL CODER™

A Project Report

submitted by

JAHFAR THATTARUTHODI

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
(Electrical Engineering)
and
MASTER OF TECHNOLOGY
(Communication Engineering)

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **FPGA PROTOTYPING OF GPS L1 SIGNAL ACQUISITION USING HDL CODER$^{\text{TM}}$**, submitted by **Mr. Jahfar Thattaruthodi**, to the Indian Institute of Technology, Madras, for the award of **Dual Degree** (Bachelor of Technology & Master of Technology), is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Radha Krishna Ganti
Project Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 27th June 2016

ACKNOWLEDGEMENTS

I have to praise and thank God for all the blessings he has bestowed upon me and after that, I take this opportunity to express my sincere gratitude to my project guide Dr. Radhakrishna Ganti for his valuable guidance and facilitation through out this project. It is a privilege to be a student at IIT Madras. I express my gratitude to all my teachers for all the academic insights I obtained from them. I am thankful to my colleague Dheeraj and my senior Rahul for giving me substantial support for this project. I would like to thank my friends Jamshed Nasri, Inayath, Ajmal, Abdullah, Hisham, Favas, Ajay, Arjun and all my lab mates for maintaining such a cheerful work atmosphere. I would also like to thank my family for always being there for me over the years. Their love and support have played a large part in making me who I am today.

ABSTRACT

KEYWORDS: GPS (Global Positioning System), SDR (Software Defined Radio), FPGA (Field Programmable Gate Array), HDL (Hardware Description Language), HLS (High Level Synthesis)

Software defined radio implementation of GPS receiver is carried out to study the underlying principles and algorithms of the GPS system. SDR implementation requires minimal hardware such as antenna, down converter and sampler along with a host computer. The digital samples are fed to the computer and all the required signal processing operations are performed by a software. SDR implementation is more convenient for setting new parameters and introducing changes in the algorithms. The main aim of this project is to develop an FPGA prototype of GPS L1 receiver using HLS tool such as HDL Coder^{TM}. The SDR implementation is used as a reference model to verify the FPGA prototype of GPS receiver. A GPS receiver has to perform three main steps to estimate the location of the receiver antenna: acquisition, tracking and navigation data decoding. Designing an HDL compatible Simulink model for GPS L1 signal acquisition and its HDL code generation and synthesis work flow are explained in this thesis. The FPGA design flow starts from algorithms. The behavioural model is designed first in floating point and verified using simulators. Then the synthesis constraints are set and the floating point model is converted to fixed point. HDL code generation and synthesis are performed by HDL Coder via HDL Workflow Adviser. The 'bit file' generated is then loaded to FPGA. The area and timing optimisation is done in order to reduce resource utilization and improve the clock speed. Timing optimisation is done by pipe-lining and area optimisation is done by resource sharing and word length optimisation. HDL Coder provides choice to automatically perform many of the optimization features and therefore it saves a lot of time as compared to manually optimizing the design. Functional verification of the FPGA synthesis is performed FPGA-in-the-Loop co-simulation by interfacing Simulink with the target FPGA board with the help of HDL Verifier^{TM}. Rapid FPGA prototyping of GPS receiver using an HLS tool such as HDL Coder^{TM}and the design and verification challenges are the main focus of the thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ASIC	Application Specific Integrated Chip
BPSK	Binary Phase Shift Keying
C/A	Coarse-Acquisition Code
CDMA	Code Division Multiple Access
CORDIC	COordinate Rotation DIgital Computer
FIL	FPGA in the Loop
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HDL	Hardware Description Language
HLS	High Level Synthesis
HW	Hardware
IP	Intellectual Property
IRNSS	Indian Regional Navigation Satellite System
JTAG	Joint Test Action Group
PPS	Precision Positioning Service
PRN	Pseudo Random Noise
RTL	Register Transfer Level
SDR	Software Defined Radio
SoC	System on Chip
SPS	Standard Positioning Service
SW	Software
UHD	USRP Hardware Driver
USRP	Universal Serial Radio Peripheral
VHDL	VHSIC Hardware Description Language

CHAPTER 1

INTRODUCTION

Global Positioning System(GPS) is the world's first Global Navigation Satellite System(GNSS) developed and deployed by the US department of defence. GPS became fully functional in 1995 and it consists of a cluster of 32 satellites orbiting around the earth in the medium earth and geostationary orbits. GPS satellites send data using the CDMA transmission scheme in the L1 and L2 band of frequencies. GPS data is BPSK modulated and it uses Gold codes for spread spectrum modulation. All the GPS satellites have a high precision atomic clock which are internally synchronised between them and it plays a major role in the accuracy of position measurements. These satellites send their position information and time of transmission as GPS data. A GPS receiver on earth requires data from at least four satellites to calculate its position. The underlying principle behind the position calculation is the method of `trilateration`.

A GPS receiver has to perform three major steps in order to compute the position information: Acquisition, tracking and Navigation data decoding. In this project, an FPGA prototype of GPS L1 signal acquisition is built with the aim of implementing Standard Positioning Service(SPS) of GPS on an FPGA platform. The Acquisition algorithm is designed as an HDL compatible Simulink Model and then FPGA synthesis is performed with the help of HDL Coder. HDL Coder^{TM}is an High Level Synthesis (HLS) tool that accelerates the design flow for ASIC/FPGA development of Matlab and Simulink models and it supports complex data types. Classically ASIC designs include the manual coding of the whole system and verifying the design. Introduction of the HLS tools for FPGA prototyping shifts the focus from low end to high end of the de sign flow. From the designer point of view, the designing of the SoC IP becomes less complex and there is provision for the verification of the design in real time.

In this project, Simulink model of the acquisition part of the GPS receiver have been designed in order to generate the HDL code. The Simulink model has to be constructed only using HDL supported blocks present in the Simulink library. The design is tested for its accuracy and speed with reference to a software defined radio(SDR) implementation of the

GPS receiver which only requires an antenna and a USRP as hardware. The USRP digitizes the received GPS signal and the digital samples are processed using the SDR developed in software(MATLAB) in order to extract the receiver location. The software code developed for SDR implementation is used as a reference for the FPGA prototyping.

The ASIC/FPGA design flow using HDL Coder starts from algorithms. A reference model is created from the algorithms using Simulink/Matlab, in this case Simulink. The reference model is a behavioural model that requires no timing, concurrency or target technology information. A reference model is generally represented in floating point arithmetic. The floating point model is further converted to fixed-point arithmetic for synthesis. The fixed-point arithmetic enables usage of optimal word lengths and integral arithmetic on hardware, therefore, is cheaper and smaller in area compared to the floating point arithmetic. Fixed-point model is verified in a simulator to have equal functionality as that of floating-point model. When the model has been verified and is working, synthesis constraints can be set. The implementation of the Acquisition module from algorithm level to FPGA using HDL Coder and the testing of the model using Virtex-5 XC5V-LX110T FPGA board are discussed in detail in the following chapters.

1.1 Organisation of the thesis

Chapter 2 briefly describes the underlying principles of GPS receiver design.
Chapter 3 presents a detailed description of the theories behind GPS signal acquisition.
Chapter 4 briefly explains about the various hardware components used in the front-end of the GPS receiver
Chapter 5 introduces the high level synthesis(HLS) tool HDL Coder^{TM}and the ASIC/FPGA prototyping design flow using the same.
Chapter 6 briefs the functionality of the subsystems used in constructing the Simulink model of acquisition and the floating point to fixed-point conversion of the design.
Chapter 7 shows the HDL code generation via HDL Workflow Adviser and the usage of validation model of the generated HDL code.
Chapter 8 presents various optimization features provided by HDL Coder. FPGA synthesis and functional verification in 'FPGA in the loop(FIL)' configuration are also discussed.
Chapter 9 summarises the works done and the future prospects

CHAPTER 2

HOW DOES GPS WORK?

In GPS, there are 32 satellites deployed around the earth with at least 24 of them are operational at any given time in order to provide navigational solution(location and time information) in all weather conditions, anywhere on or near the earth, where there is an unobstructed line of sight connection to four or more of GPS satellites. The receiver design for standard positioning service(SPS) of GPS consists of mainly three parts: acquisition, tracking and navigation data processing. In acquisition, we process the received digitized signal to extract the signals of each contributing satellite. In tracking, we keep track of the changes in the signal frequency caused due to the relative motion of the satellite with receiver. In navigation data processing, we extract the satellite information from the received signal, find the satellite position, calculate the pseudo distance between the satellites to the receiver and then estimate the receiver position based on technique of trilateration. A brief description to trilateration, acquisition, tracking and navigation data decoding is given below in the interest of highlighting the over all working principles of GPS receiver. GPS signal acquisition algorithms are further explained in detail with FPGA implementation of the same via HDL coder in subsequent chapters.

2.1 Trilateration

The position of a certain point in space can be found from the distance measured from this point to some known positions in space. This concept is known as `trilateration` in geometry and is the fundamental idea behind the working of GPS. Let's illustrate this point using a one-dimensional case. If the satellite position S_1 and the distance to the satellite d_1 from the user are both known, the user position can be at two places, either to the left or right of S_1. In order to determine the user position, the distance d_2 of the user to another satellite S_2 with known position must be known. Thus in a one-dimensional case, we need two satellites to uniquely determine the user location as shown in Fig. 2.1.

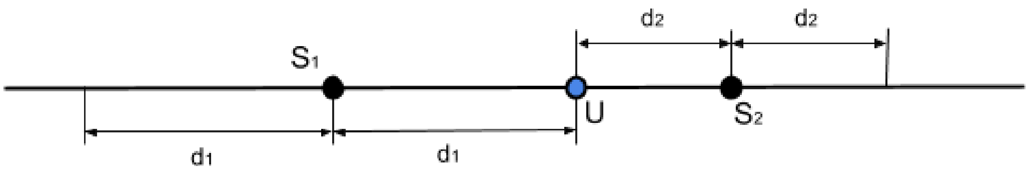


Figure 2.1: Trilateration in One-dimensional case

In a two-dimensional case, in order to determine the user position, three satellites and three distances are required. The locus of a point with constant distance to a known satellite is a circle in the two-dimensional case. Two satellites and two distances give two possible solutions because two circles intersect at two points. A third circle is needed to uniquely determine the user position as shown in Fig. 2.2.

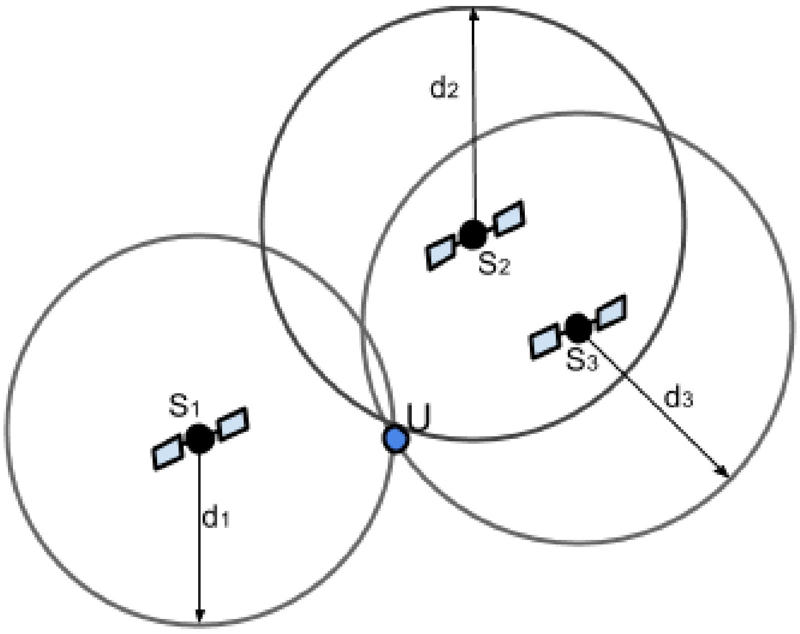


Figure 2.2: Trilateration in Two-dimensional case

.

Similarly, in a three-dimensional case, we require four satellites and four distance measures from the user. The locus of a point with constant distance to a given satellite is a sphere. The two spheres intersect to form a circle, whose intersection with another sphere gives two points. One of the points will be very far from the earth and can be neglected, thereby obtaining position fix using three satellites in three-dimension. Theoretically, we need a fourth satellite to obtain a unique solution as shown in Fig. 2.3. In GPS, the positions of the satellites are obtained from the ephemeris data transmitted by the satellites. The distance from the receiver to the satellites are estimated by finding the time taken for the signals transmitted by the satellites to reach the receiver and then multiplying it with the speed of light. Each GPS satellite is internally synchronized with each other to transmit signals at the same time and each of them stamps the time at which it transmits into the data transmitted. At the receiver we find the time of arrival of each of the signals. The time of arrival measured is not absolute, because the user clock might lead or lag when compared to the GPS clock, since the cheap user clock is not synchronized with the GPS clock. Fortunately, as the signals are transmitted from the satellites at the same time, all the received satellite signals contains the same unknown constant bias in their estimate of time of arrival. Due to this constant unknown bias, the distance estimated between the user and satellites are not absolute and hence are referred to as pseudo-range. Therefore, the GPS system has to estimate (X_u, Y_u, Z_u) and the time offset between the GPS clock and the user clock t u in order to solve for position. Extending the argument for the requirement of three satellites in three-dimension to solve for three unknowns (neglecting the point far from earth in the final solution), we now need four satellites in three-dimension to solve for four unknowns(*ie.* three space coordinates of the receiver antenna and the constant unknown clock bias).

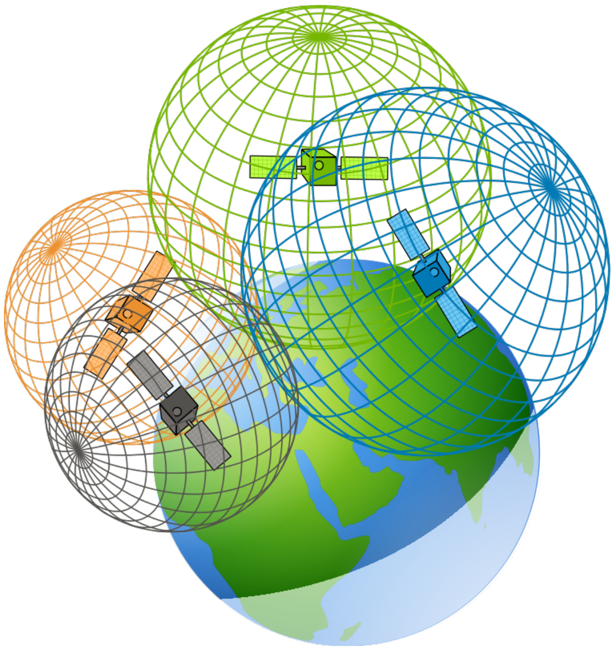


Figure 2.3: Trilateration in Space

.

2.2 Acquisition

The key purposes of acquisition are :

A. To find the satellites which are visible from the user location.

B. To find the time alignment (code phase) of C/A codes in the each satellite signals.

C. To find the Doppler shift in the carrier frequency (Carrier Doppler) of each satellite signals.

Although there are at least 24 operational GPS satellites in space, not all of them will be visible from the user's location on earth. In order to get a fix on the location of the GPS user, we need to know the spatial coordinates of at-least four satellites contributing to the received user signal. Each satellites data is spread with its unique pseudo random noise(PRN) code and at the receiver side, we find out the satellites contributing to the received signal by correlating with locally generated PRN codes. The CDMA codes have high correlation only for zero lag. Therefore, in order to properly decode the received signal, we need to locally generate time aligned PRN codes of each visible satellites. The term 'code phase' here stands for the time alignment of the PRN code in the current block of data. The line-of-sight velocity of the satellite with the receiver causes a Doppler shift in the carrier frequency of at-most $\pm 10kHz$. It is important to know the exact carrier frequency for demodulating the received signal with the locally generated carrier signal. The Doppler shift encountered for each satellites have to be determined with an accuracy of $\pm 500Hz$. Thus the correlation of the received signal with a PRN code and the local carrier frequencies will result in two-dimensional correlation. A satellite that is visible to the receiver will have a significant correlation peak in the acquisition search space, corresponding to its carrier frequency and alignment of it's C/A code in the received data as shown in Fig. 2.4, whereas a satellite that is not visible to the receiver will not have a significant peak in the search space as shown in Fig. 2.5. The ratio of the highest correlation peak to the second highest correlation peak is taken as a peak metric for detecting the visibility of the satellite. If the peak metric is greater than a predetermined threshold, then the satellite is considered to be visible.

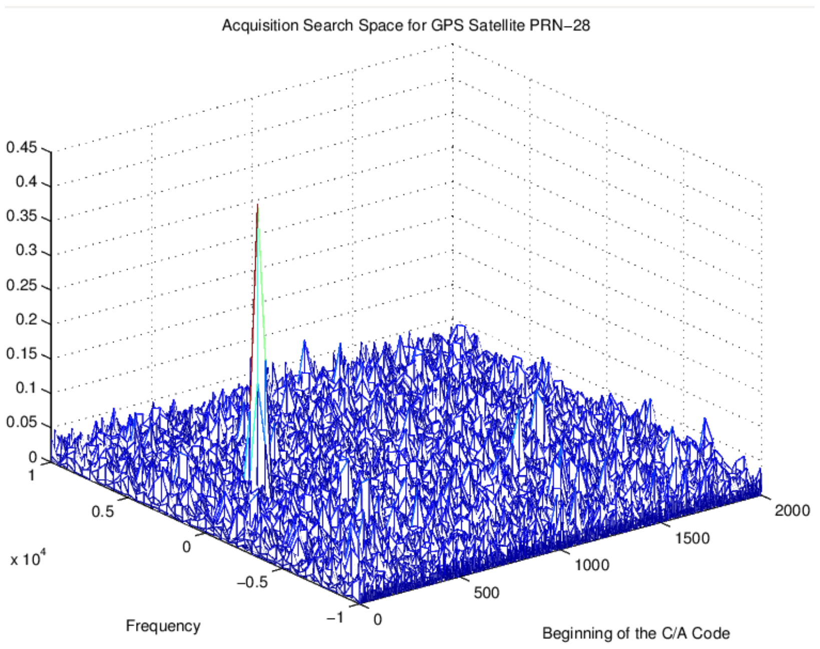


Figure 2.4: Satellite 28 is visible so a significant peak is present

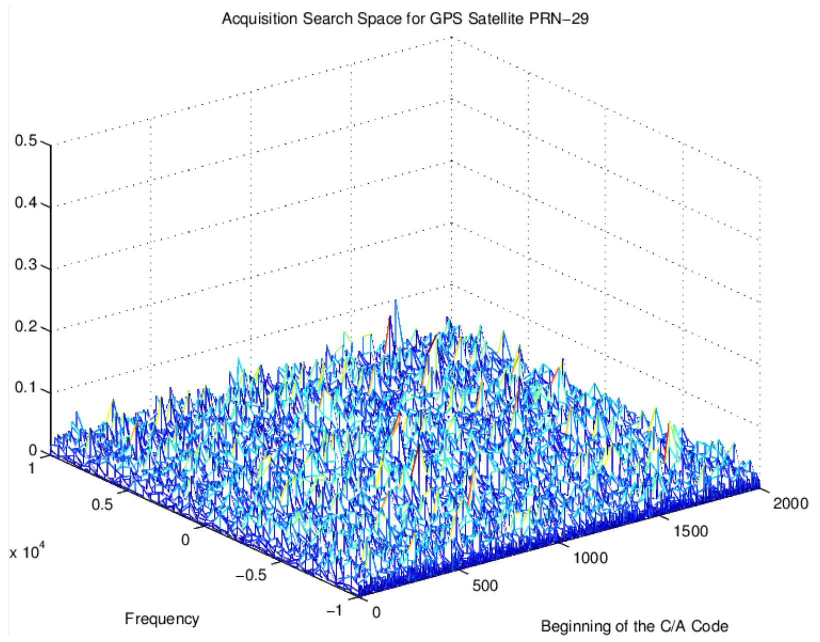


Figure 2.5: Satellite 29 is not visible so no peak is present

2.3 Tracking

After acquisition we only get the initial estimates of carrier frequency and code phase parameters. Due to the relative velocity of the satellite with the receiver, the carrier and code frequency varies slowly over time. In order to correctly demodulate the signal we also need to make sure that the phase of both the incoming carrier signal and the locally generated signal are matched. Therefore the tracking block implements a phase locked loop(PLL) system to make fine corrections to the frequency of carrier and code replica, so as to accurately demodulate the signal. A PLL will have a phase detector and a loop filter in general as shown in Fig. 2.6. If there is phase error between the received signal and the generated signal, then a control mechanism kicks in to adjust the frequency and phase of the generated signal, so as to match it with that of the received GPS signal. Fig. 2.7 depicts simulation of the working of a designed PLL.

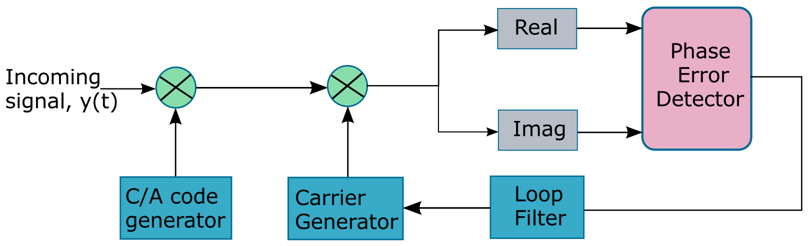


Figure 2.6: Carrier tracking algorithm

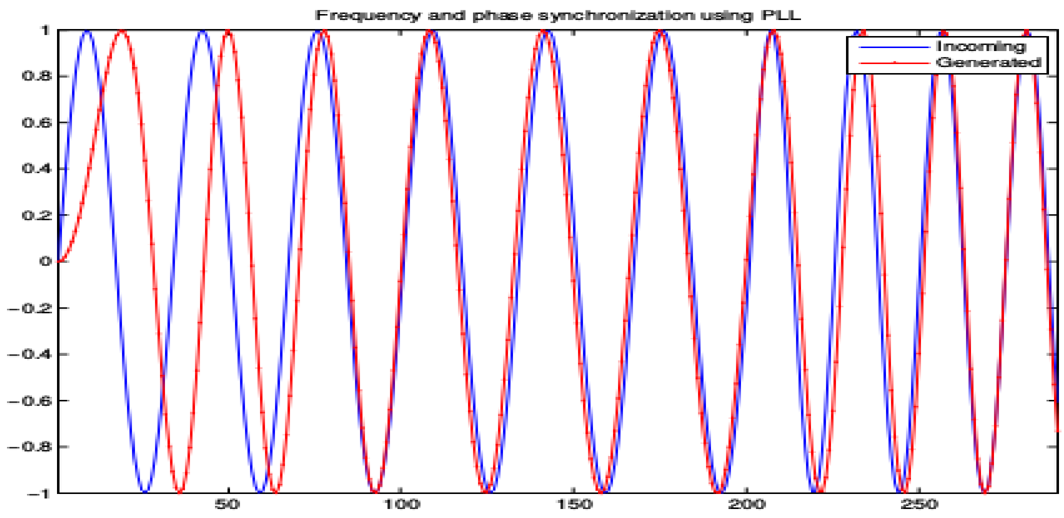


Figure 2.7: Simulating the working of the designed PLL

The C/A code in the incoming signal and the locally generated C/A code can drift apart, as the frequency of the C/A code in the incoming signal changes due to relative motion of the satellite with the receiver. In order to estimate the code phase error, a delay locked loop(DLL) was implemented. The carrier removed signal is correlated with early, prompt and late version of the PRN code, with early and late code having a shift of $\pm 1/2$ chips from the prompt version as shown in Fig. 2.8 , and their extent of correlation with the received signal is determined as (I_E, Q_E), (I_P, Q_P) and (I_L, Q_L). A measure of the code phase error detector can be expressed as :

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

Figure. 2.9 shows an optimized version of the combined tracking loops. Here the I and Q inputs to the phase discriminator are the I_p and Q_p correlation from the code tracking loop.

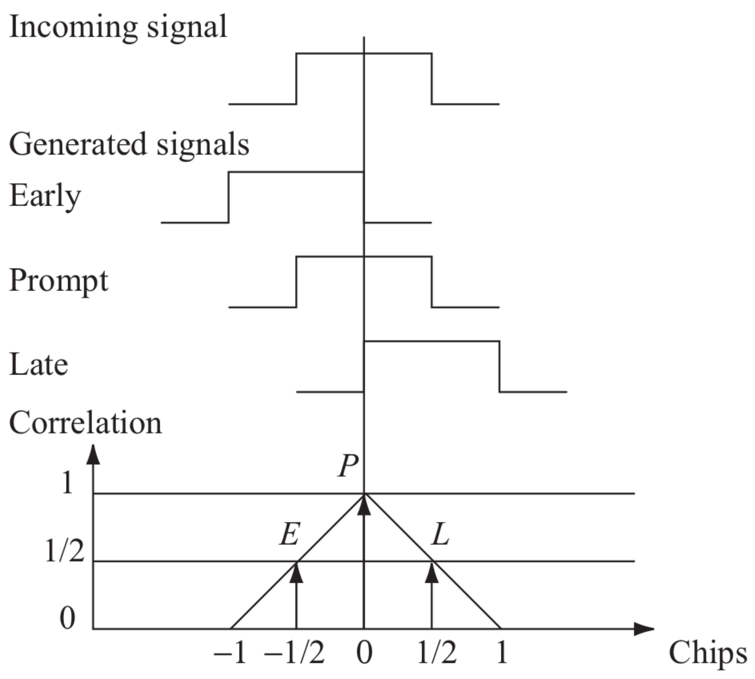


Figure 2.8: Early Prompt Late correlator

The navigation bit sequence is obtained from the in-phase prompt correlator output. The constellation plot of the prompt correlator output shows that the data is concentrated in the in-phase arm as shown in Fig. 2.10 and Fig. 2.11.

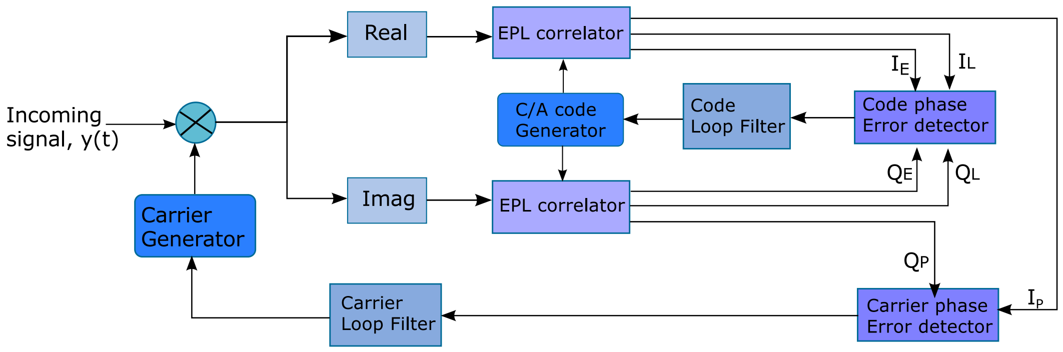


Figure 2.9: Combined Carrier and Code tracking algorithm

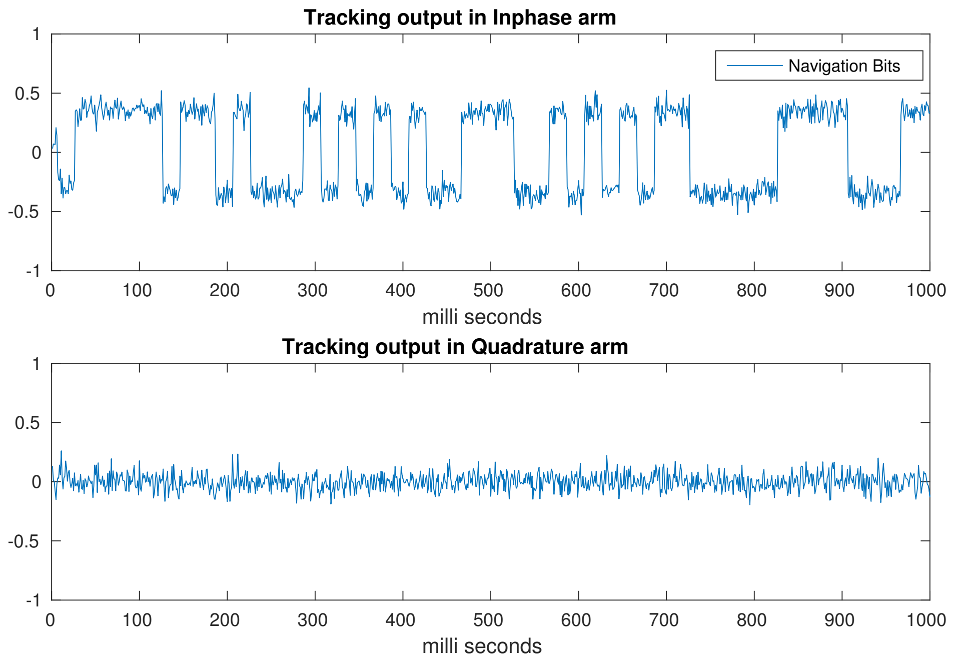


Figure 2.10: Tracking output I_P and Q_P. Navigation bit sequence visible in the inphase arm

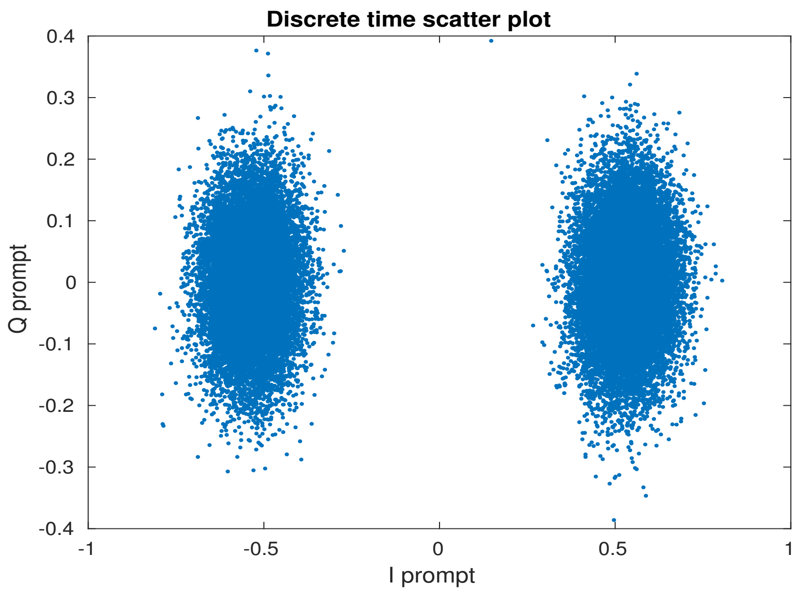


Figure 2.11: BPSK constellation of the GPS navigation data

2.4 Navigation Data Decoding

The navigation bits transmitted on the L1 frequency have a bit rate of 50 *bps*. The basic format of the navigation data is a 1500-bit-long frame containing 5 subframes (SF_1, SF_2, SF_3, SF_4, SF_5), each having length 300 bits as shown in Fig. 2.12 . One subframe contains Ten words ($W_1, W_2, ..., W_{10}$), each word having length 30 bits. Subframes 1, 2, and 3 are repeated in each frame. The last subframes, 4 and 5, have 25 versions (with the same structure, but different data) referred to as page 1 to 25. With the bit rate of 50 bps, the transmission of a subframe lasts 6 s, one frame lasts 30 s, and one entire navigation message lasts 12.5 minutes. Each subframe starts with two special 30 bit long words, the telemetry (TLM) and handover (HOW). TLM contains an 8 bit preamble sequence which is used for identifying the beginning of the subframe (frame synchronization). HOW word contains the information about the subframe number and the GPS time of week at which the signal was transmitted. Subframe 1 contains information needed for the satellite clock correction and the accuracy/health of the data transmitted. Subframe 2 and 3 contains the satellite ephemeris data, which is the information regarding the satellite orbits required for estimating the satellite position. Subframe 4 and 5 contain data which are 12.5 minutes long. They are referred to as almanac data and contains the ephemerides and clock data with reduced precision. Additionally, each satellite transmits almanac data for all GPS satellites while it only transmits ephemeris data for itself.

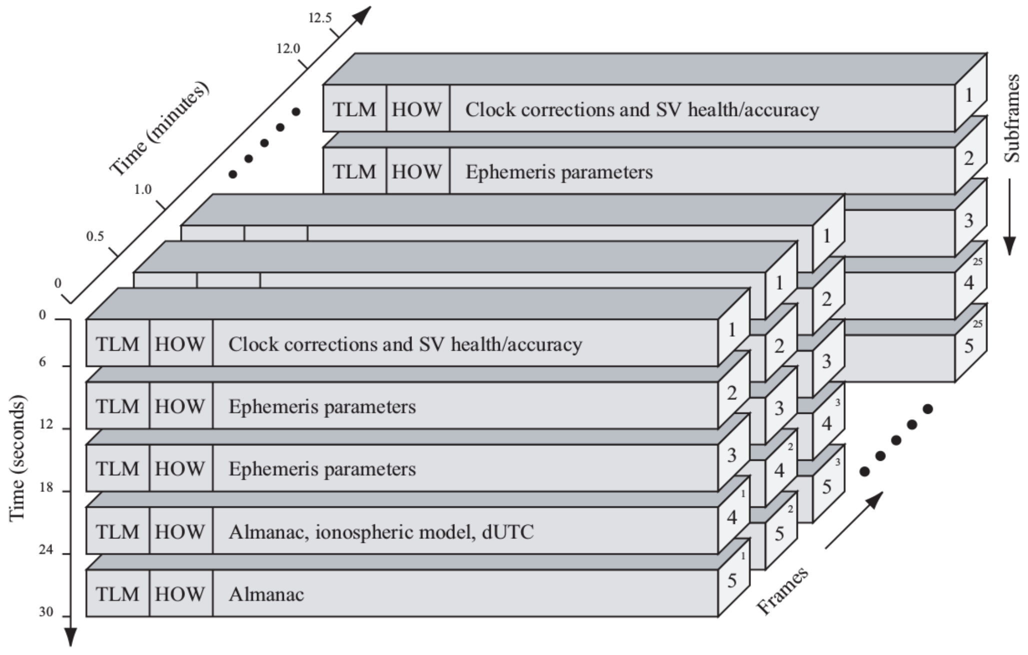


Figure 2.12: GPS Navigation Data Structure

In order to find the beginning of the subframes from the navigation data (obtained from the tracking block), we make use of the preamble sequence found in the word W 1 of each subframes. The data structure in word W 1 is as shown in Fig. 2.13. Besides 24 bits of data, every 30-bit word contains 6-bits for parity checking. $D_1 - D_{24}$ are the 24 data bits, while $D_{25} - D_{30}$ are the 6 parity bits in a word. $D^*{}_{29}$ and $D^*{}_{30}$ represents the last two parity bits from the previous word. From the parity check equations, we calculate the parity bits $D_{25} - D_{30}$ and check if they match with the parity bits of the received navigation bits. Parity Checking also decides if the received navigation bits have to be inverted or not. d_{1-30} here refers to the received navigation data bits. Details of every word can be obtained from the GPS interface control document ICD-GPS-200C.

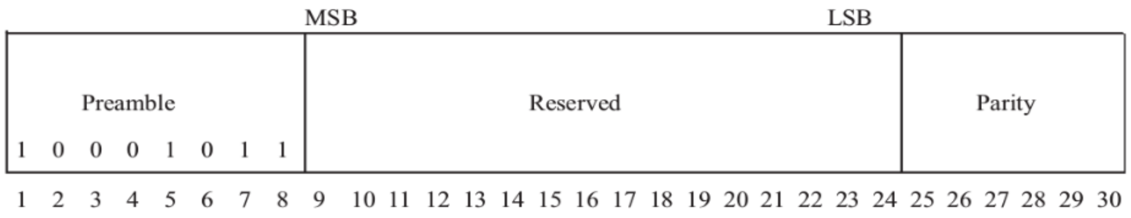


Figure 2.13: Structure of first word in subframe

GPS ephemeris (satellite and orbital information) parameters are extracted from the navigation data after parity checking following the ICD-GPS-200C decoding scheme. The pseudo-range measurement is computed as the travel time from the satellite to the receiver multiplied by the speed of light in vacuum. The receiver has to estimate exactly when the start of a frame arrives at the receiver.Once the pseudo-range is estimated, the GPS position algorithm (based on trilateration) can be used to estimate the location of the receiver.

CHAPTER 3

A DETAILED DESCRIPTION OF GPS SIGNAL ACQUISITION

3.1 GPS Signal Structure

The signal transmitted in the L1 frequency by the GPS satellites are a combination of the standard positioning service (SPS) and the precise positioning service (PS). The signal S_{L1}^k transmitted in the L1 frequency by satellite k can be mathematically expressed as:

$$\begin{aligned} S_{L1}^k(t) &= A_c C^k(t) D^k(t)\cos(2\pi f_{L1}t+\phi) + A_p P^k(t) D^k(t)\sin(2\pi f_{L1}t+\phi) \\ &= m_I^k(t)\cos(2\pi f_{L1}t+\phi) + m_Q^k(t)\sin(2\pi f_{L1}t+\phi) \end{aligned} \tag{3.1}$$

where A_p is the amplitude of the P(Y) code, P(t) is the P(Y) code, A_c is the amplitude of the C/A code, C(t) is the C/A code, D(t) = ±1 is the data, f_{L1} is the L1 frequency, ϕ is the initial phase. The P(Y) part is attenuated by $3dB$ compared to C/A part. The C/A code repeats itself every ms, and one navigation bit lasts 20 ms (Since, data rate is $50bps$) . Hence for each navigation bit, the signal contains 20 complete C/A codes and at the receiver the data bit is found by integrating over the 20 C/A codes.

The received power of the GPS SPS L1 signal is nearly -130dBm. The thermal noise power at the absolute temperature t in $^\circ K$ and for an equivalent bandwidth of B in Hz can be calculated as

$$P_{ThermalNoise} = ktB$$

where k = $1.38 \times 10^{-23} J/^\circ K$. For the GPS SPS L1 signal with a null to null bandwidth of $2MHz$, the thermal noise power is

$$\begin{aligned} &= 1.38 \times 10^{-23} \times 290 \times 2 \times 10^6 \\ &= 8.004 \times 10^{-15} W \\ &= -110.97 dBm \end{aligned} \tag{3.2}$$

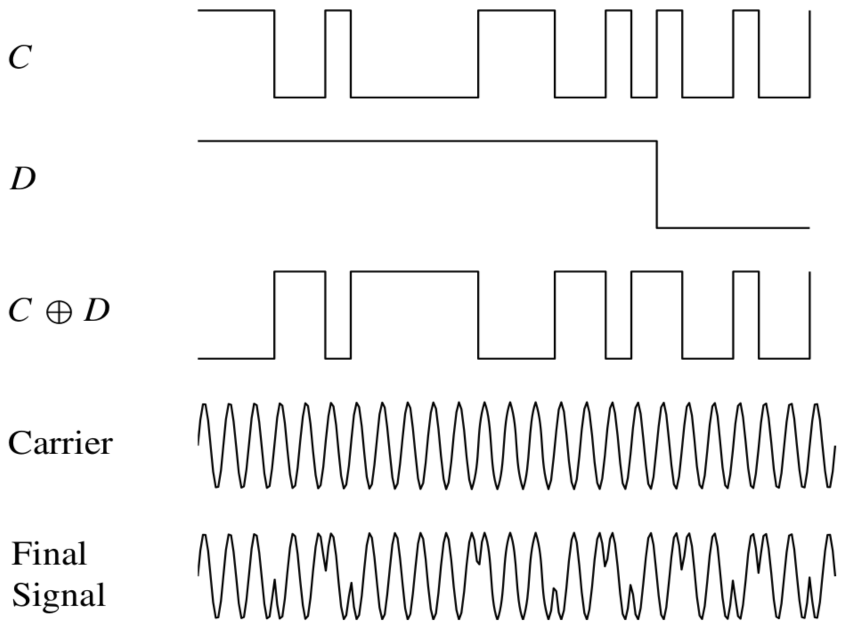


Figure 3.1: The resulting L1 signal obtained by modulating the data signal D with the C/A code C and carrier

As the received signal power is below the noise floor, the signal presence cannot be determined from it's spectrum. This is a feature of the code division multiple access (CDMA) spread spectrum signal and requires the appropriate signal processing to acquire and process the signal. This also implies that the design of the front end is based more on the level of the thermal noise rather than the received L1 band navigation signal. Thus, the voltage induced within the GNSS antenna element results from the thermal noise, which dominates, as well as the GNSS signals from the satellites in view. Theoretically, the spectrum of GPS system with the BPSK modulation for SPS and PS signals should look like in Fig. 3.2

3.2 Downconversion of the Received signal

The signal received by a receiver antenna on earth will be a superposition of the delayed version of the transmitted signal from all the satellites visible from the receiver location. As the satellites are moving, their will be a Doppler effect on its transmitted frequency. It can

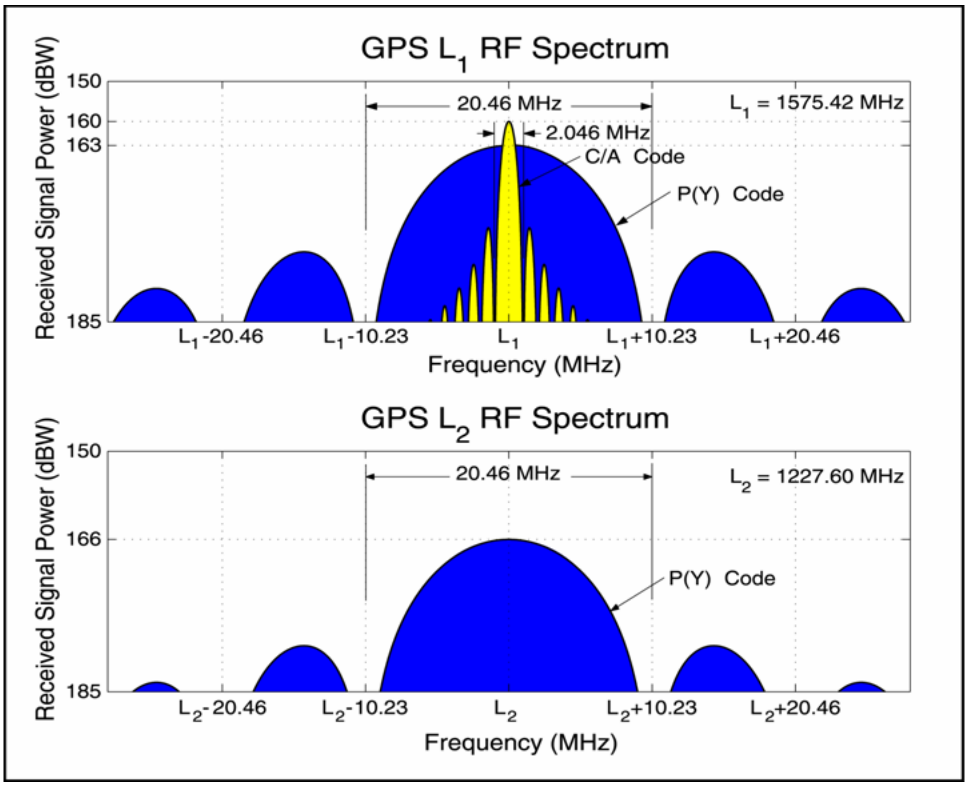


Figure 3.2: Spectrum of GPS SPS + PS in L1 band and PS in L2 band

be expressed as:

$$\begin{aligned} y_{Rx}(t) &= \sum_{k}^{VisibleSat} S^k_{L1}(t-\tau^k) + noise \\ &= \sum_{k}^{VisibleSat} m^k_I(t-\tau^k)\cos(2\pi(f_{L1}-f^k_{dop})t+\phi^k) + m^k_Q(t-\tau^k)\sin(2\pi(f_{L1}-f^k_{dop})t+\phi^k) \\ &= \sum_{k}^{VisibleSat} m^k_I(t-\tau^k)\cos(2\pi f^k_{eff}t+\phi^k) + m^k_Q(t-\tau^k)\sin(2\pi f^k_{eff}t+\phi^k) \end{aligned} \tag{3.3}$$

where $y_{Rx}(t)$ is the received signal by the antenna, τ^k is the time taken for the signal to reach the receiver from satellite k, f^k_{dop} is the Doppler shift in transmitted frequency, f^k_{eff} is the effective transmitted frequency and ϕ^k is the phase of the received signal from satellite k. The signal is then down-converted to baseband using USRP N210 at a local oscillator frequency $f_o = f_{L1}$. The down-conversion of the signal received y_{Rx} from satellite k alone can be expressed as :

Signal in the in-phase arm $y^k_I(t)$ is :

$$\begin{aligned} y_{Rx}(t) * \cos(2 f_o t) = \frac{m_I^k(t)}{2} &\left[\cos\left(2\pi(f_{eff}^k - f_o)t + \phi^k\right) + \cos\left(2\pi(f_{eff}^k + f_o)t + \phi^k\right) \right] \\ + \frac{m_Q^k(t)}{2} &\left[\sin\left(2\pi(f_{eff}^k - f_o)t + \phi^k\right) + \sin\left(2\pi(f_{eff}^k + f_o)t + \phi^k\right) \right] \end{aligned} \tag{3.4}$$

Signal in the Quadrature arm $y_Q^k(t)$ is :

$$\begin{aligned} y_{Rx}(t) * \sin(2 f_o t) = \frac{m_I^k(t)}{2} &\left[\sin\left(2\pi(f_{eff}^k - f_o)t + \phi^k\right) - \sin\left(2\pi(f_{eff}^k + f_o)t + \phi^k\right) \right] \\ + \frac{m_Q^k(t)}{2} &\left[\cos\left(2\pi(f_{eff}^k - f_o)t + \phi^k\right) - \cos\left(2\pi(f_{eff}^k + f_o)t + \phi^k\right) \right] \end{aligned} \tag{3.5}$$

After low pass filtering, we get

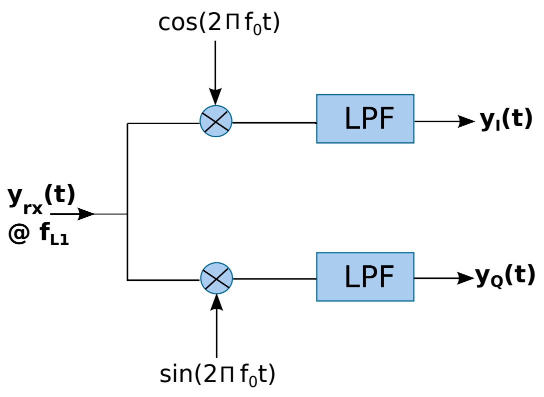


Figure 3.3: Passband to baseband down-conversion in USRP

$$\begin{aligned} y_I(t) &= \frac{m_I^k(t)}{2} * \cos\left(2\pi f_{dop}^k t + \phi^k\right) + \frac{m_Q^k(t)}{2} * \sin\left(2\pi f_{dop}^k t + \phi^k\right) \\ y_Q(t) &= \frac{m_I^k(t)}{2} * \sin\left(2\pi f_{dop}^k t + \phi^k\right) + \frac{m_Q^k(t)}{2} * \cos\left(2\pi f_{dop}^k t + \phi^k\right) \end{aligned} \tag{3.6}$$

The final signal after down-conversion to complex baseband in USRP can be expressed as :

$$y(t) = \sum_{k}^{VisibleSat} \left(m_I^k(t) + j * m_Q^k(t) \right) e^{j\left(2\pi f_{dop}^k t + \phi^k\right)} \tag{3.7}$$

where y(t) is the received signal after down-conversion in USRP, $m_I^k(t) = C^k(t)D^k(t)$, $m_Q^k(t) = P^k(t)D^k(t)$, f_{dop}^k is the Doppler shift for satellite k, ϕ^k is the phase of satellite k.

3.3 Serial Search Acquisition

From the received signal y(t), we need to find which are the visible satellites, their Doppler shift f_{dop}^k and the beginning of the C/A code of each satellite. For demodulating the received signal, we need a locally generated carrier signal as well as a time aligned locally generated PRN code. If both the locally generated signal and code are perfect, then on correlation with the received signal, the output would be very high. A brute force way to do this will be to down convert the received signal with all possible carrier signals and then correlate in time domain with all possible code alignments of the PRN code. We know that $|f_{dop}^k| \leq 10kHz$, so we multiply the received signal $y(t)$ with a locally generated carrier signal $e^{-j2\pi ft}$, where f is varied from -10kHz to 10kHz in steps of 500Hz *i.e* 41 different values of f , one in each iteration. The resultant signal can be expressed as :

$$\begin{aligned} y(t) &= \sum_{k}^{VisibleSat} \left(m_I^k(t) + j * m_Q^k(t)\right) e^{j\left(2\pi\left(f_{dop}^k - f\right)t + \phi^k\right)} \\ &= \sum_{k}^{VisibleSat} \left(C^k(t)D^k(t) + j * P^k(t)D^k(t)\right) e^{j\left(2\pi\left(f_{dop}^k - f\right)t + \phi^k\right)} \end{aligned} \tag{3.8}$$

After wiping off the carrier signal in an iteration, we correlate the signal with C^k , the C/A code of satellite k. The correlation process is repeated with shifted version of C/A code. If the C/A code of 1ms is sampled at 2MHz, then it will have 2000 samples, therefore we have to perform the correlation with 2000 different shifted version of C^k.

$$y_c(t) \otimes C^k = GD^k(t)e^{j\left(2\pi\left(f_{dop}^k - f\right)t + \phi^k\right)} + noise \tag{3.9}$$

where G is the processing gain of C/A correlation. For the C/A code of correct code alignment and correct carrier signal, the correlation output will be high.

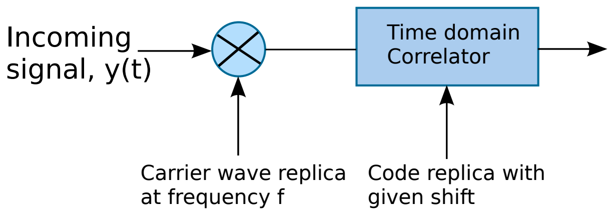

Incoming signal, y(t)
Time domain Correlator
Carrier wave replica at frequency f
Code replica with given shift

Figure 3.4: Serial search acquisition

3.4 Parallel Code Phase Search Acquisition

The serial search acquisition algorithm is too computationally heavy. Modification to the above algorithm to make it computationally efficient can be done by doing the correlation operation in the frequency domain and then take inverse-FFT to find the results of the correlation in time-domain. The code alignment corresponding to the highest correlation is the start of the C/A code in the received signal. A detailed block diagram of the method discussed can be found below:

Relation between circular correlation in time domain and frequency domain:
Let $m(n)$ be the received signal down-converted by the generated carrier frequency at f and $c(n)$ be the locally generated C/A code of a particular satellite. As both these signals are in discrete time domain, they are periodic and hence the correlation is circular.
The correlation between the two signals can be written as:

$$z(n) = \sum_{i=0}^{N-1} m(i)c(i+n) \tag{3.10}$$

The discrete Fourier transform of the two finite length sequences can be computed as:

$$M(k) = \sum_{n=0}^{N-1} m(n)e^{-j2\pi kn/N}$$

and

$$C(k) = \sum_{n=0}^{N-1} c(n)e^{-j2\pi kn/N}$$

The N point Fourier transform of z(n) can be expressed as:

$$\begin{aligned}
Z(k) &= \sum_{n=0}^{N-1}\sum_{i=0}^{N-1} m(i)c(i+n)e^{-j2\pi kn/N} \\
&= \sum_{i=0}^{N-1} m(i)\left[\sum_{i=0}^{N-1} c(i+n)e^{-j2\pi k(n+i)/N}\right]e^{j2\pi ki/N} \\
&= C(k)\sum_{i=0}^{N-1} m(i)e^{j2\pi ki/N} \\
&= C(k)M^{-1}(k)
\end{aligned} \tag{3.11}$$

where $M^{-1}(k)$ is the inverse DFT of $m(n)$.

Alternatively,

$$\begin{aligned}
Z(k) &= \sum_{n=0}^{N-1}\sum_{i=0}^{N-1} c(i)m(i+n)e^{-j2\pi kn/N} \\
&= \sum_{i=0}^{N-1} c(i)\left[\sum_{i=0}^{N-1} m(i+n)e^{-j2\pi k(n+i)/N}\right]e^{j2\pi ki/N} \\
&= M(k)\sum_{i=0}^{N-1} c(i)e^{j2\pi ki/N} \\
&= M(k)C^{*}(k)
\end{aligned} \tag{3.12}$$

Since c(n) is real $C^1(k) = C(k)$

Using this relation, the time domain result can be obtained by taking IFFT of $Z(k)$

$$z(n) = \sum_{k=0}^{N-1} Z(k)e^{j2\pi kn/N}$$

The maximum of $|z(n)|$ in the n^{th} location and respective frequency step gives the beginning of the C/A code and the carrier frequency of the signal.

3.5 How to Determine the Length of Signal Used for Acquisition

The C/A code of satellites C^k are 1ms long, while the navigation data D^k transmitted from the satellites are 20ms long. Typically 1ms of data from the received signal is used for correlating with the locally generated C/A code during acquisition. If the received satellite signal is very weak, then long data length are required for acquisition to detect satellites. The longer the data record used the higher the signal-to-noise ratio that can be achieved. The factors that limit the length of the data record are

- Navigation data transition in the data

The navigation data remains constant for 20ms or in other words, if we take 20 ms of received data, then there can be at-most one data transition. The navigation data transition will spread the spectrum and the output will no longer be a continuous signal. The spectrum spread will degrade the acquisition result. To overcome this, we take two consecutive 1ms received data blocks and perform acquisition algorithm on both. As the data blocks are contiguous, there can be navigation data transition in at-most one of the blocks.

- Doppler effect on the C/A code (Code Doppler)

The Doppler effect on C/A code frequency can be obtained after determining the Doppler effect on the carrier by the following relation:

$$f'_{code} = f_{code} * \frac{f_{L1} + f_d}{f_{L1}}$$

where $f_{code} = 1.023MHz$, $f_{L1} = 1575.42MHz$ and f_d is the carrier Doppler estimated from the acquisition. As $f_d = \pm 10kHz$, the maximum Doppler shift expected on a C/A code is 6.4 Hz. Hence the chip duration in the C/A code varies due to Doppler. If a perfect correlation peak of C/A code with the locally generated C/A code is 1, the correlation peak decreases to 0.5 when a C/A code is off by half a chip. This corresponds to 6 dB decrease in amplitude. If the maximum allowable code misalignment is half a chip (0.489 us) for effective correlation. It takes about 78 ms $(1/(2 \times 6.4))$ for two frequencies different by $6.4Hz$ to change by half a chip. This data length limit is much longer than the duration of 1ms used for acquisition here. Hence, the variation in code Doppler need not be considered during acquisition but they have to be considered during the tracking block.

3.6 How to Determine the Number of Frequency Steps Used in Acquisition

The Doppler frequency range that needs to be searched is $\pm 10kHz$. It is important to determine the frequency steps needed to cover this 20 kHz range. The frequency step is closely related to the length of the data used in the acquisition. When the input signal and the locally generated carrier signal are off by 1 cycle there is no correlation. When the two signals are off by less than 1 cycle there is partial correlation. It is arbitrarily chosen that the maximum frequency separation allowed between the two signals is 0.5 cycle. If the data record is 1 ms, then a 1 kHz signal will change 1 cycle in the 1 ms. In order to keep the maximum frequency separation at 0.5 cycle in 1 ms, the frequency step should be 1 kHz. Under this condition, the furthest frequency separation between the input signal and the correlating signal is 500 Hz or 0.5 Hz/ms and the input signal is just between two frequency bins. If the data record is 10 ms, a searching frequency step of 100 Hz will fulfil this requirement. In our experiments, a frequency step of 500 Hz is used to cover the 20 kHz range, which will amount to 41 frequency steps.

CHAPTER 4

FRONT-END OF GPS RECEIVER

4.1 Block Diagram of Receiver Chain

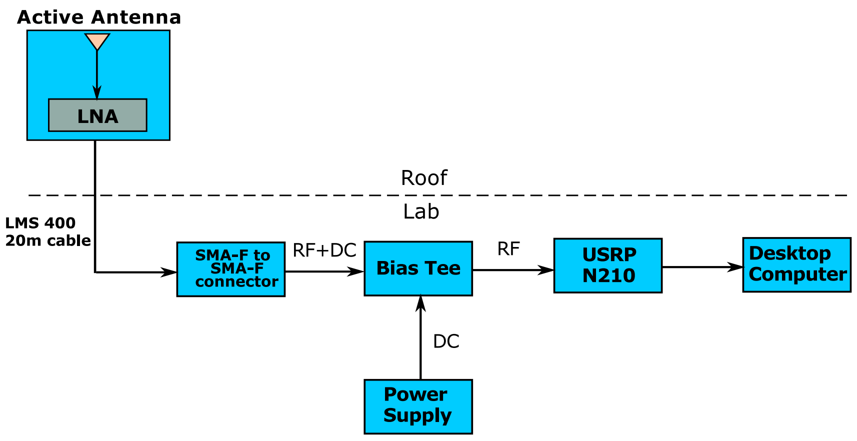


Figure 4.1: Block diagram

The active antenna fitted on the roof of Electrical Science block is able to receive the GPS signal in the L1 band ($f_{L1} = 1575.42MHz$). The antenna has an inbuilt filter with a bandwidth of $f_{L1} \pm 50MHz$ and an amplifier of gain $30dB$. Signal from the antenna is carried to the laboratory through the $20m$ long coaxial cable. A bias tee is used to power up the amplifier in the antenna by supplying $4V$ DC. An SMA-F to SMA-F connector was required to connect the coax cable (SMA-M) with the bias tee's RF+DC port (SMA-M). The bias tee outputs RF signal centred at $1575.42MHz$ to the USRP N210, which down-converts the signal to baseband using its local oscillator at $1575.42MHz$, followed by sampling the signal at $2MHz$.

4.2 Components of the Receiver

4.2.1 Integrated GPS Antenna

The key features required for a GPS antenna are:

Item	Specifications
Model	Synergy Telecom Outdoor Antenna
Center frequency	$1575.42 \pm 3MHz$
LNA Gain	$28 \pm 2dB$
Noise figure	$1.5dB$
Filter Out Band Attenuation	$12dB\ min\ f_0 \pm 50MHz$
DC Voltage	$2.2 - 5V$
DC Current	$5 - 15mA$

Table 4.1: GPS Active Antenna specifications

- Wide spatial angle : The antenna should have wide spatial angle to receive the low elevation satellite signals as well as the high elevation angle signals. The spatial angle should also be narrow enough to block the interfering signal which usually arrive at low elevation angles.
- Small gain variation : The antenna should have small gain variation from zenith to azimuth, so that all the received satellite signal should be of comparable strength. In the CDMA system if the received signals are not of comparable strength then the strong signals may interfere with the detection of the weak signals.
- Right-handed circular polarized (RHCP) : The GPS signals are right-handed circular polarized signals and hence the antenna should also be RHCP. This indirectly helps in mitigating multipath interference as the reflection of an RHCP signal will be an LHCP signal. As the antenna is RHCP, it has lower gain for LHCP signal.

4.2.2 Coax Cable

The $20m$ long coax cable has N-M and SMA-M connectors at it's end. The N-M connector connected to the N-F connector of the GPS Active antenna and the SMA-M connector has to be connected to the Bias Tee's RF+DC port.

Item	Specification
Model No	LMR-400
Length	$20m$
Cable Run Attenuation	$3.9dB$

Table 4.2: Coax cable specifications

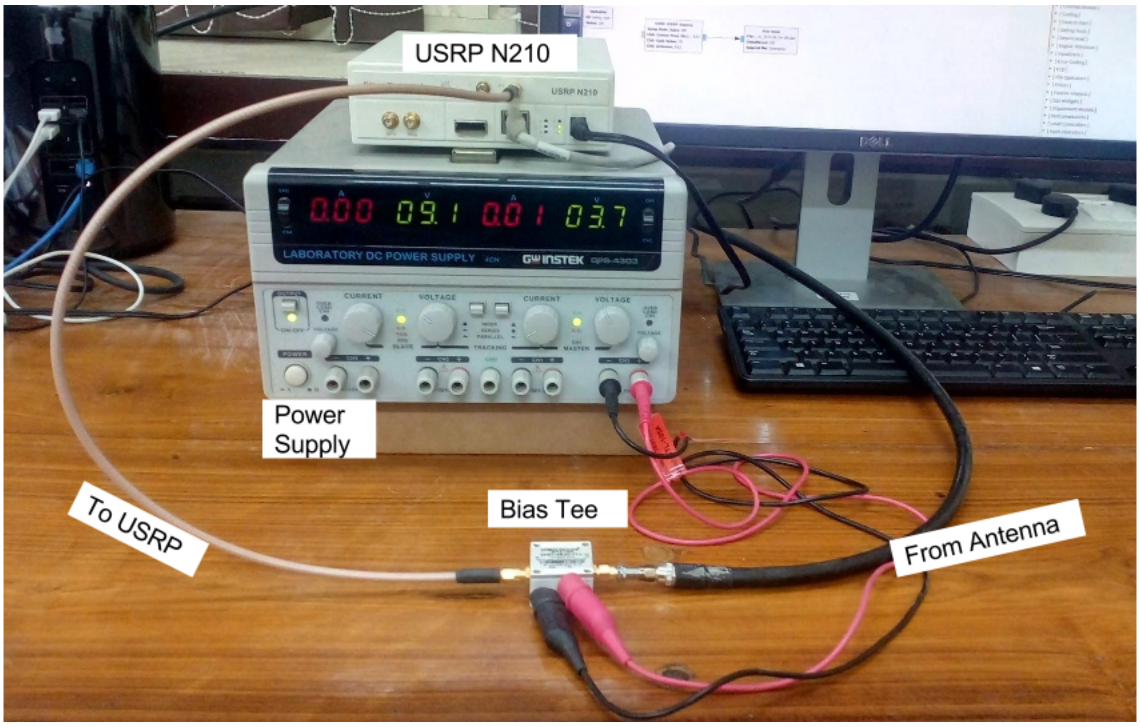


Figure 4.2: Experimental Setup

4.2.3 Connector

The LMR-400 cable running from the roof has SMA-M connection at its end and the Bias Tee also has SMA-M at its RF+DC port, so an SMA-F to SMA-F RF connector was used for connecting the cable with the bias-tee.

Item	Specification
Model No	SF-SF50+
Insertion Loss	$0.05dB$

Table 4.3: Coax cable adapter specifications

4.2.4 Bias Tee

The Bias Tee when powered with power supply at 4.3V is able to power the amplifier/filter in the active antenna without disrupting the GPS signal. The RF+DC port of the bias tee is connected with the GPS antenna and the RF port is connected with USRP N210.

Item	Specification
Model No	ZFBT-4R2G-FT+
Voltage	4.3 V
Insertion Loss	$0.47dB$

Table 4.4: Bias Tee specifications

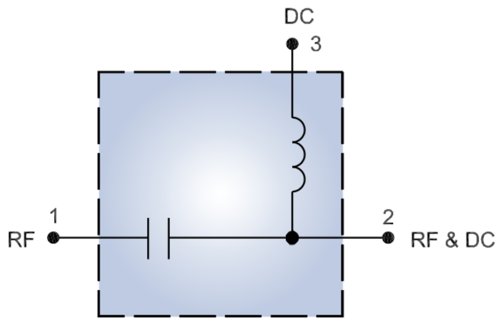
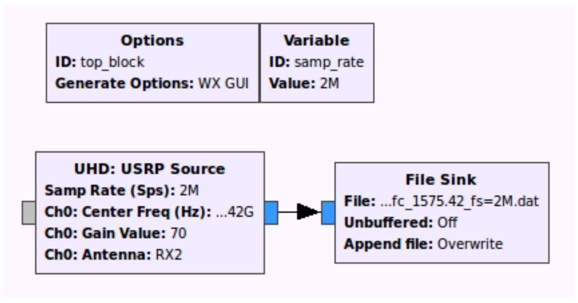


Figure 4.3: Bias Tee Electrical Schematic

4.2.5 USRP N210

The USRP is configured to receive the gps signal by creating a flowgraph in GNURadio-companion. GNURadio is a graphical toolkit used to develop mod- ular SDR systems, which can then be run using USRP devices, using the UHD API. The UHD: USRP Source block is configured to down-convert the signal at $1575.42MHz$ to complex baseband and sample it at $2MHz$. The sampled data contain I and Q signals and are saved to a file in gnuradio-complex datatype.

Figure 4.4: Front view of the USRP N210 transceiver


Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 2M

UHD: USRP Source
Samp Rate (Sps): 2M
Ch0: Center Freq (Hz): ...42G
Ch0: Gain Value: 70
Ch0: Antenna: RX2

File Sink
File: ...fc_1575.42_fs=2M.dat
Unbuffered: Off
Append file: Overwrite

Figure 4.5: GNURadio FlowGraph for configuring USRP to receive GPS signal

4.3 Amplification Consideration

The available thermal noise power P_T at the input of the receiver can be calculated as:

$$P_T = kTB$$

where k is the Boltzmann's constant ($= 1.38 \times 10^{-23} J/K$), $T = 300K$ and $B = 2.046MHz$ is the bandwidth of the receiver in hertz.

$$P_T(dBm) = 10log_{10}(kTB) = -111dBm$$

The maximum voltage required to exercise all the levels of the ADC is about $100mV$ and the corresponding power is nearly $0.1mW$ ($-10dBm$). The C/A code signal power level at the receiver will be approximately $-130dBm$, hence the signal is $19dB(-111 + 130)$ below the noise floor. Therefore, in order to amplify the GPS signal, we need to raise noise floor level to the maximum range of the ADC ($= -10dBm$). If we raise the signal to the level of the ADC, then noise will saturate the ADC output. Hence, we need a net gain of about $101dB(-10+111)$. Since in the RF chain there are filters and cable losses, the insertion loss of these components must be compensated with additional gain. The net gain must be very close to the desired value of $101dB$. Too low a gain value will not activate all the possible levels of the ADC. Too high a gain will saturate some components or the ADC and create an adverse effect.

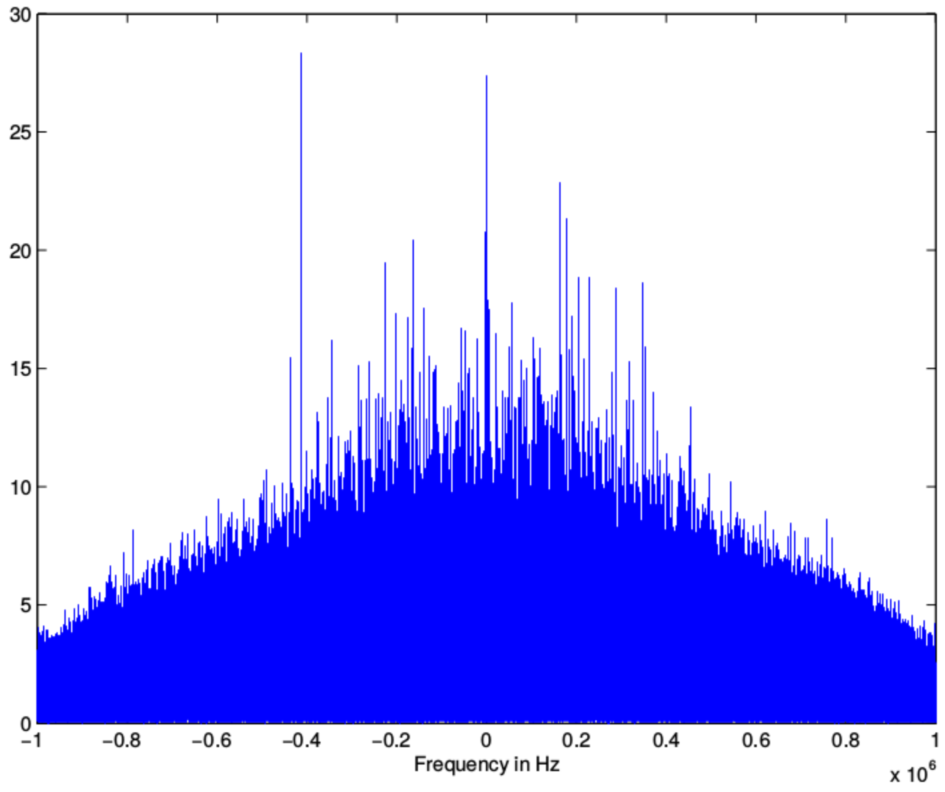


Figure 4.6: Spectrum of the received signal in baseband

CHAPTER 5

INTRODUCTION TO HDL CODER

HDL Coder^{TM} generates portable, synthesizable Verilog and VHDL code from MATLAB functions, Simulink models, and State-flow charts. The generated HDL code can be used for FPGA programming or ASIC prototyping and design.

HDL Coder provides a work-flow advisor that automates the programming of Xilinx and Altera FPGAs. User can control HDL architecture and implementation, highlight critical paths, and generate hardware resource utilization estimates. HDL Coder provides traceability between the Simulink model and the generated Verilog and VHDL code, enabling code verification for high-integrity applications adhering to DO-254 and other standards.

Figure 5.1: HDL code generation from Matlab/Simulink model using HDL Coder

5.1 Algorithm Design for HDL Code Generation

MathWorks HDL coder is a HLS -tool to synthesize MATLAB or Simulink algorithm model to VHDL or SystemVerilog code. This chapter covers the design principles for synthesizable MATLAB or Simulink model. HDL synthesis is performed with an example IP block. HDL coder uses the designed model, including the user-defined settings and the target technology files, as input to generate HDL code for both FPGA and ASIC. The tool has floating point to fixed-point converter built-in so both the floating point and the fixed-point algorithms are supported, which makes it flexible. However, HDL coder has some limitations on design principles to be able to synthesize the design into HDL. These limitations are discussed in the section 4.1 below.

5.1.1 From Matlab Model

MATLAB is generally used for algorithm design of a system for fast simulation and verification purposes of the behavioural model.
Since Soft Ware(SW) technology has more degrees of freedom compared to Hard Ware(HW), HDL coder supports only a subset of MATLAB language that is targeted for HW. For example, synthesis from MATLAB Object-Oriented Programming (OOP) classes is not supported. However, it supports synthesis from MATLAB System Objects that are specialized objects designed for dynamic systems.
To produce rational HDL code, the algorithm should be written from the HW perspective. Algorithm models are often written into simulation optimized vector operations that create parallel structures and copies of combinational logic blocks in HW when processed by HDL coder.An optimized way is to use only the necessary amount of the combinational logic to perform the logic operations within the timing constraint and multiplexing time- variant input signals into the circuit. This reduces the area of the hardware significantly as described in section Optimization.Loop structures can be automatically converted to streaming structures by using loop unrolling. However, parallel structures can be used if the target is to maximize the speed. To optimize the generated model, the algorithm model should be written in a way it is desired to be on HW.
The first thing when starting to design a model for the HDL code generation is to verify that Data Types, Operators and Control Flow Statements to be used are supported by the tool.
To create synthesizable MATLAB code, the structure has to be correct. The design functionality has to be written in functions or MATLAB System Objects that are targeted for dynamic systems. Sub-functions or System Objects are then called within a main function to be included in the synthesis. Handshaking/synchronization between blocks, functions, variable indexing and also signal buffering should be coded in the MATLAB design in a way it is desired to be in RTL.

A feature that limits MATLAB for being used for large designs is that it has no concept of time. Therefore, it is not compatible with multi-rate designs using multiple clock domains.

5.1.2 From Simulink Model

Simulink is a graphical design tool that uses library blocks, MATLAB functions and System Objects to perform certain functionality. Simulink library includes vast selection of hardware optimized blocks and cover some of the functionalities needed in SoC development. User-defined MATLAB function blocks and System Object blocks can be used to create design specific functionalities or to re-utilize existing MATLAB algorithms. Simulink supports the multi-rate designs, therefore, it has an advantage over MATLAB when a design has more than one clock domain.
For HDL code generation, all the blocks available in the Simulink library cannot be used to design the simulink models but only HDL code generation supported blocks are to be used. Fixed-point Converter can be used to convert floating point data types into user-defined fixed-point types. Solver settings can also affect the simulation speed. "MultiTasking" option can be used to speed up the simulation but it's not supported for HDL generation. For HDL code generation "SingleTasking" option has to be used. In this project Simulink is used to implement the GPS signal acquisition algorithms.

5.2 HDL Code Generation From Model

Completed and behaviourally verified algorithm design can be synthesized into RTL. HDL coder takes MATLAB or Simulink design as input and generates either VHDL or Verilog from it. In Simulink case all blocks inside the top-level design are synthesized into separate VHDL-files and are imported as components on the top-level VHDL. An example of the code generation principle is shown in Fig. 5.2. The top- level VHDL-file includes design I/O ports, internal signal declaration, port mapping of the components and assigning internal signals into the I/O ports.

Generating HDL code from a MATLAB design follows the same principle as shown above if sub-functions are written in separate MATLAB function files and are called within a main function. If all the functionality is written in a single MATLAB function or System Object, the whole design is synthesized into a single VHDL-file.
The target specific parameters can be configured by the user in HDL coder. After setting the parameters, HDL coder performs checks for global settings, algebraic loops, compatibility

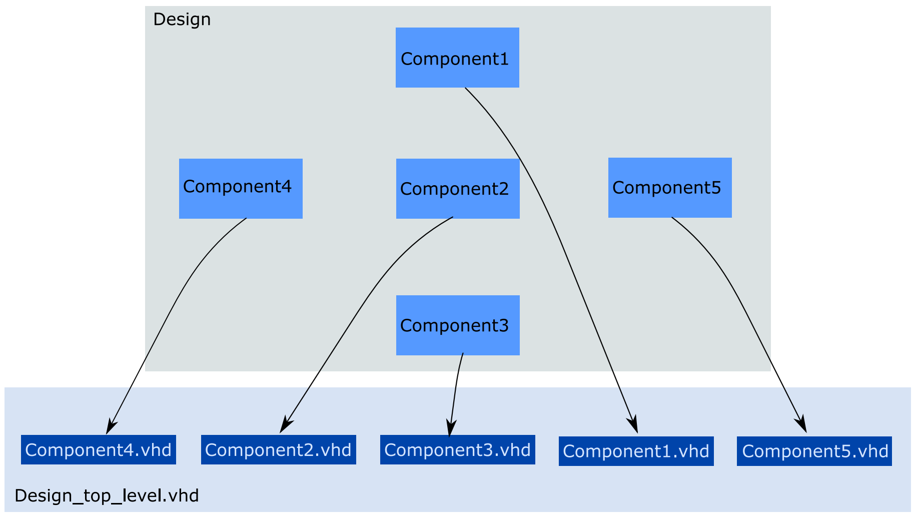


Figure 5.2: HDL code generation principle from Simulink/MATLAB Design

and sample time to verify that the design is synthesizable. Depending on the design size, the RTL synthesis takes from a few seconds to a few minutes. The output is human readable HDL code making it really interesting from the point of RTL coding by automating the RTL code generation. The design flow times are depicted in Fig. 5.3.

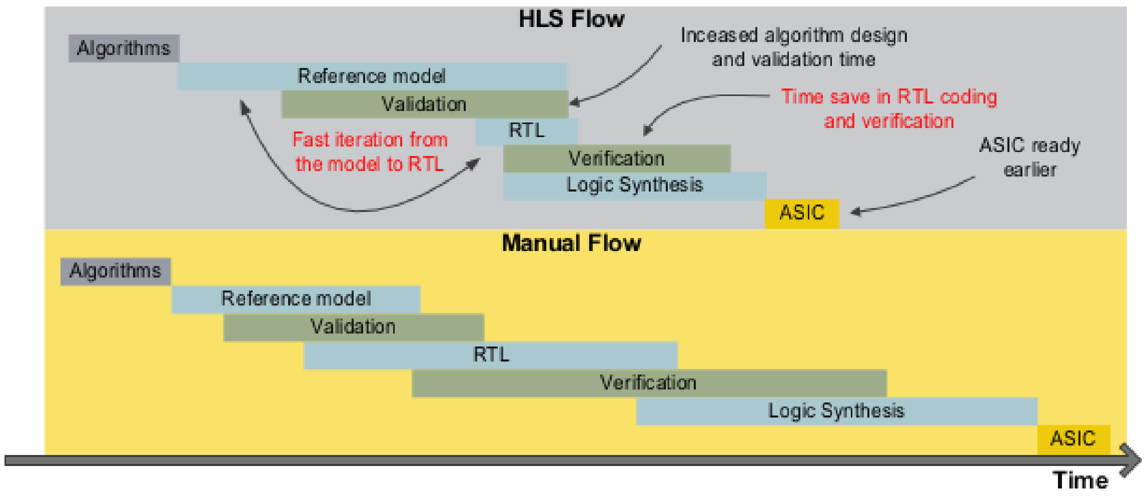


Figure 5.3: Approximate ASIC design flow timeline.

If changes are made on the algorithm design, a new HDL code can be rapidly generated. The generated HDL code also provides visibility backwards to the algorithm model from VHDL-file trough links that take the user to corresponding MATLAB function. It also preserves all the comments of the MATLAB function into VHDL.
HDL coder generates traceability, resource utilization, critical path and optimization reports for the RTL model automatically but user can also disable the report generation. The user

can also view detailed resources block by block. The critical path of the VHDL design can be back annotated in Simulink model.

5.3 HDL Code Verification

In this chapter, HDL code verification possibilities that the MathWorks work-flow provides are discussed.

5.3.1 Verification in RTL simulator

Co-simulation automatically generates stimulus for a HDL model from MATLAB/Simulink test bench and runs a RTL simulator in the background. Co- simulation compares output of the code generation model of the algorithm to the HDL model's output. HDL Verifier is required to be installed. The HDL model is simulated in the background in user-defined RTL Simulator and the output is imported in MATLAB. Co-simulation compares the models bit-accurately and cycle-accurately. Simulation configuration is presented in Fig. 5.4 [27].

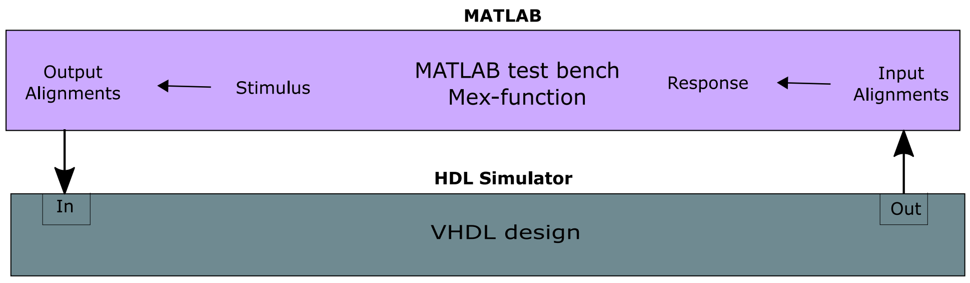


Figure 5.4: HDL co-simulation configuration.

Error is calculated from the differences of code generation model simulation and RTL simulation. The error comes mostly from quantization inaccuracies and if the algorithm model is written with fixed-point data types the error should be zero. The comparison is done on the output ports and it is a rapid way to verify HDL design correctness every time the algorithm is changed. Co-simulation window of the example design is presented in Fig. 5.5.

The other way to simulate RTL is to generate a HDL test bench with HDL coder from MATLAB/Simulink test bench and simulate it manually in a RTL simulator with the generated HDL design files and HDL test bench. Generating the HDL test bench takes roughly

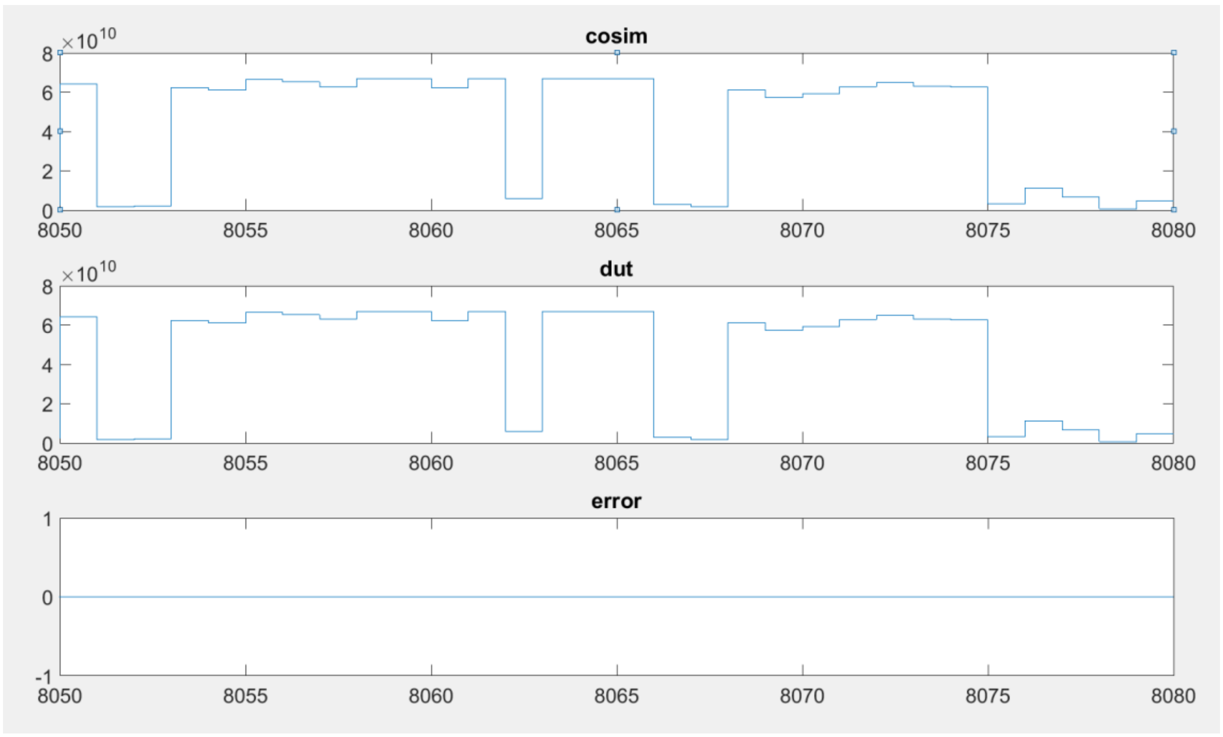


Figure 5.5: Co-simulation window where error between Algorithm model and HDL model is shown

two times longer than the entire co-simulation, thus the user have to manually verify the correctness of design functionality in the RTL simulator. Manual verification further increases the verification time. The automatic HDL code generation from the algorithm model and the RTL functional verification against the algorithm can provide great improvement in prototyping times and efficiency.

5.3.2 Additional RTL verification methods

HDL Coder has support also for other RTL verification methods such as lint checking, code-coverage analysis and verification with validation model.

The code-coverage analysis is done on the algorithm model and since HDL Coder generates the RTL from the algorithm there is no need for RTL code-coverage. The code-coverage checks that all the functions defined are used, all statements are executed, all branches are executed at some condition and all Boolean expressions are evaluated to true and false. This is a fast way to check if there are some functions that are not executed in any conditions.

HDL Coder supports 3^{rd} party lint checking tools. These are Ascent Lint, HDL Designer, Leda and SpyGlass. By enabling the lint checking, the tool generates a script for specified

lint tool and user can also configure lint checking parameters to meet the requirements. The lint checking is used to check suspicious behaviour of the model such as division by zero or assigning values to a variable before variable declaration.[25]
HDL Coder provides validation model verification method to verify functional equivalence of the original algorithm and the code generation model. Difference to the co-simulation is that this compares the original model to the code generation model over comparing the code generation model to the RTL model. Both models are fed with the same stimulus on each time step and output is compared similarly. An example design's validation model simulation output is shown in Fig. 5.6.

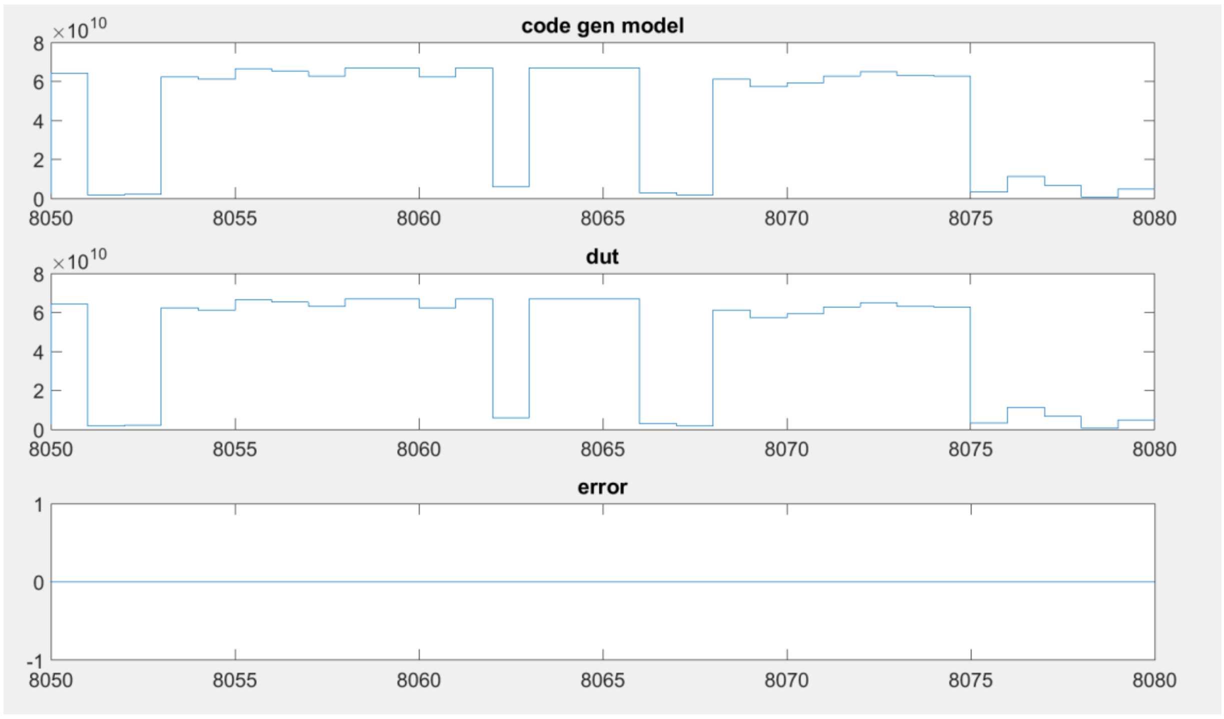


Figure 5.6: Validation model simulation output.

The flow has vast support for different RTL verification methods and all of them can be controlled within one tool. This provides improvement in prototyping flow clarity and may slightly improve the design flow times by automating the 3^{rd} party tool usage. For very detailed verifications with 3^{rd} party programs, it is easier to use the tools manually with Graphical User Interface (GUI) due to better visibility to the configuration parameters.

5.4 FPGA Synthesis and Functional Verification in FPGA Environment

In this section, synthesizing of VHDL/VERILOG model into a gate-level design, programming FPGA and verification of the functionality is discussed.

5.4.1 Logic synthesis

The logic synthesis is performed after the HDL model generated has proper functionality. The flow does not include own synthesis tool but it supports the following tools: Xilinx ISE, Xilinx Vivado, Synopsys Synplify Pro, Altera Quartus II, Mentor Graphics Precision and Microsemi Libero. The user can choose to use any of the listed tools depending on the requirements. HDL coder generates a tool specific script which is used to start the selected tool and synthesize the generated RTL code with the user-defined settings. Synthesis time and result depends on the VHDL/VERILOG model and the chosen synthesis tool. Synthesis tools provide an area report presented in logic cells once synthesis is completed.
The HDL coder generated RTL utilizes less resources than the hand-written model. Hence automated HDL coder work-flow is beneficial for FPGA based rapid prototyping due to faster iteration times compared to hand-writing the model in VHDL/VERILOG . It also creates synthesizable HDL and logic in reasonable size.
The maximum clock frequency can be achieved for the design is

$$f_{max} = \frac{1}{\tau_{critical}}$$

where $\tau_{critical}$ is critical path delay.
By following good algorithm coding rules good design speed can be achieved with the HDL coder workflow.

5.4.2 FPGA environment verification

FPGA verification can be done with FPGA-in-the-loop (FIL) configuration using the generated design and FPGA evaluation board. The FIL flow provides capability of using MATLAB or Simulink for testing the design in a real hardware environment. After HDL code generation it performs logic synthesis and generates a FPGA programming file with target FPGA specific files. FPGA is further programmed through GUI through either Ethernet or JTAG connection. The FPGA programming file can also be generated from the hand-written RTL by using FIL wizard [27]. Programmed FPGA is running in real hardware environment with MATLAB or Simulink stimulus. Data is streamed through FPGA chip and output is compared to the algorithm simulation output. FIL configuration is shown in Fig. 5.7.

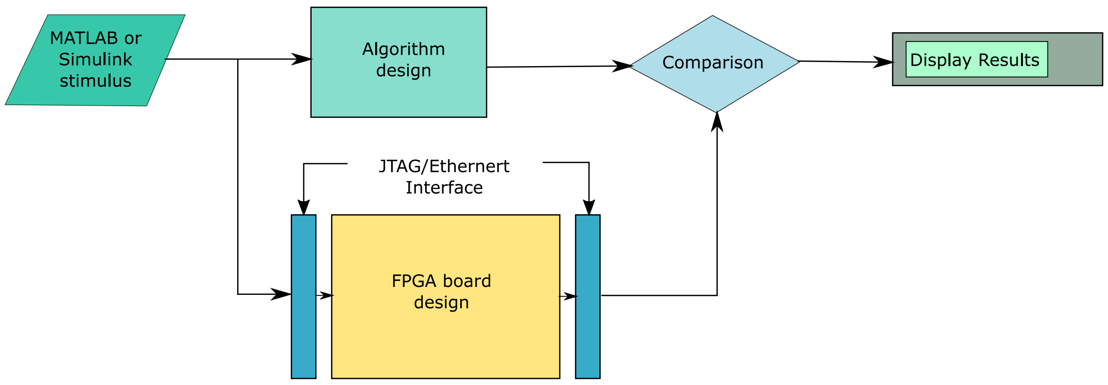


Figure 5.7: FPGA-in-the-loop simulation configuration.

The FIL output is a similar window as in co-simulation (Model Validation). The output data from the FPGA board and the algorithm is presented as waves and the difference between the outputs is compared in an error plot. An FIL simulation window is shown in Fig. 5.8.

From the simulation results in Fig. 5.8, it can be seen that the FPGA model has the same functionality as the algorithm model created in the beginning of the work. The results verify that the FIL flow produces improvement for rapid IP prototyping compared to manual verification in FPGA environment by decreasing the verification times and also making the prototyping flow faster and easier from algorithm into FPGA prototype. The flow provides also additional target work-flows such as Generic ASIC/FPGA, FPGA Turnkey, Simulink Real-Time FPGA I/O and IP Core Generation.

5.5 HDL Code Optimization

HDL Coder provides optimization features that user can apply on a design. Optimization features include adding pipeline registers, resource sharing and loop unrolling.
HDL coder allows user to specify optimization features on top-level or on a single block.

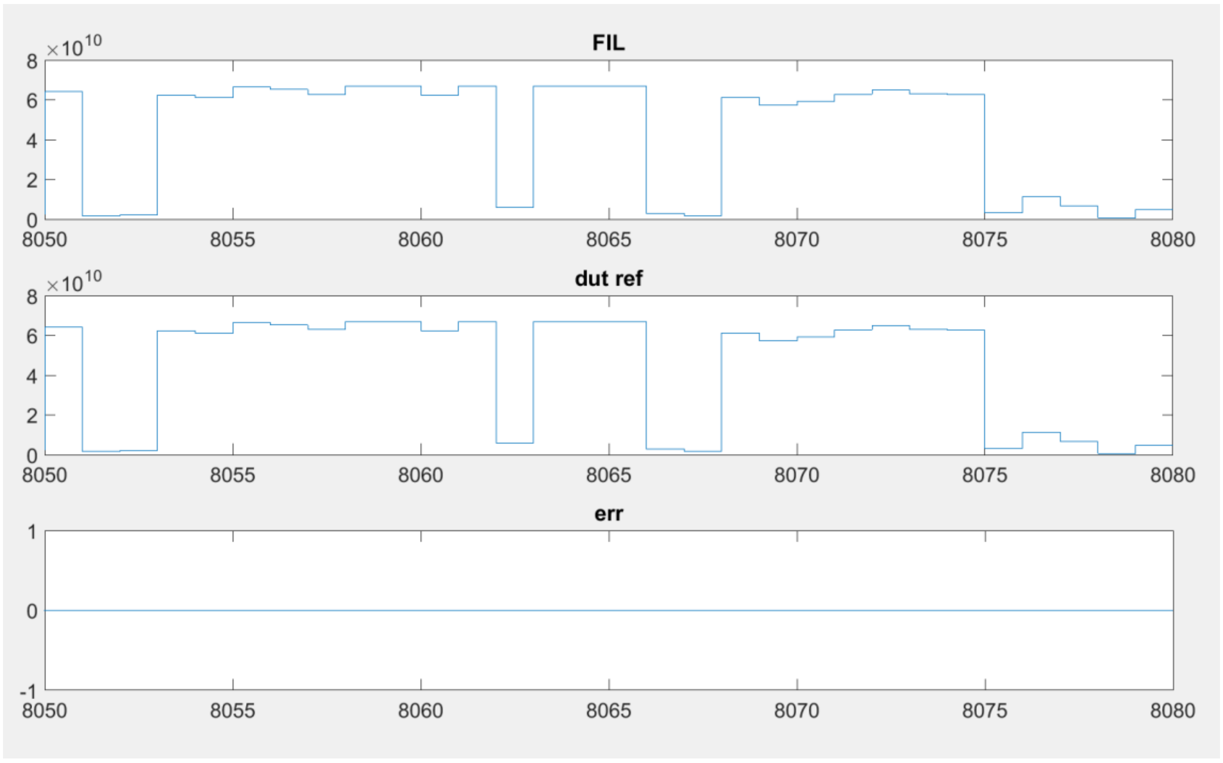
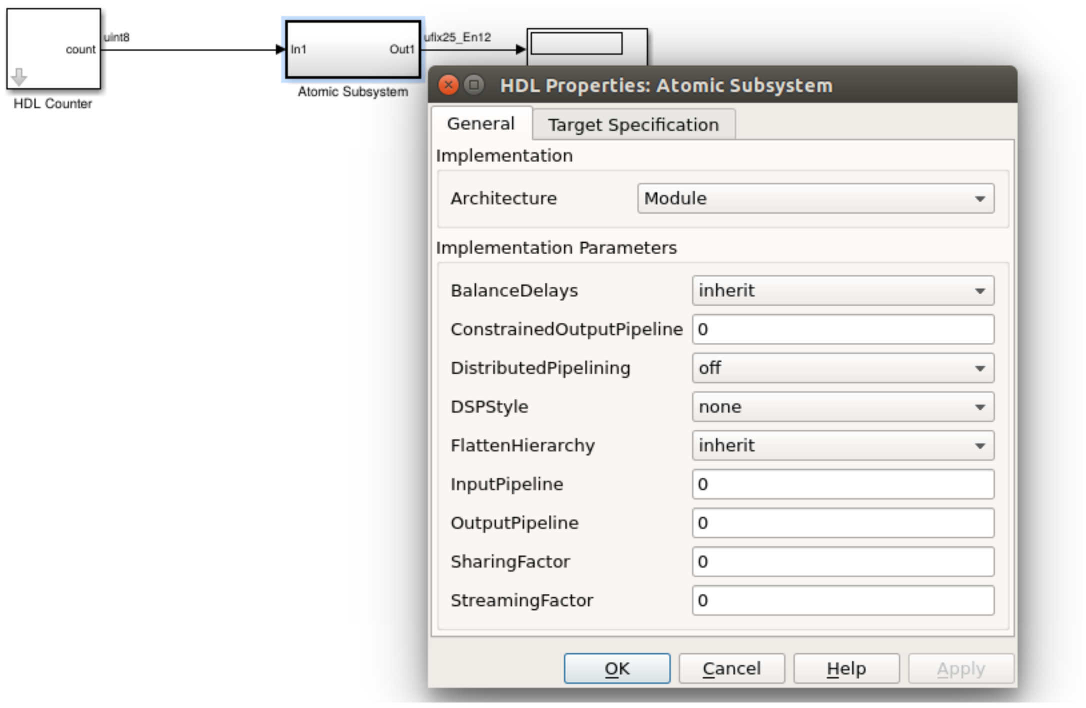


Figure 5.8: FPGA-in-the-loop simulation window presenting error between DUT and FPGA.

HDL properties window allows the user to set input and output pipeline register count, sharing factor and streaming factor. Setting "Distributed Pipelining" option "on" lets HDL Coder to distribute existing or added pipeline registers across the selected block to improve the timing characteristics. "Constrained Output Pipeline" count can be set to redistribute existing delays within your design to meet the constraints. Adding registers in a long logic path is illustrated in Fig. 5.10.Registers specified by "Constrained Output Pipeline" are not affected by "Distributed Pipelining". RAM mapping can be also used to map registers on RAM to save area. It can be specified on every block separately and it only maps those registers to RAM that are larger than the threshold value. HDL properties window is presented in Fig. 5.9

Using built-in optimization features efficiently requires understanding the design and the hardware implementation. Adding unnecessary pipeline registers increase s circuit delay and using sharing or streaming options on already optimal blocks may increase the design complexity and decrease the design quality. Therefore, if algorithm is designed with optimized functions, HDL Coder's optimization features have no effect but may reduce the quality.Sometimes RAM mapping will leads to more resource utilization, for instance enabling RAM mapping in structures which uses more than one clock cycle for configuration process



Implementation

Architecture: Module

Implementation Parameters

BalanceDelays	inherit
ConstrainedOutputPipeline	0
DistributedPipelining	off
DSPStyle	none
FlattenHierarchy	inherit
InputPipeline	0
OutputPipeline	0
SharingFactor	0
StreamingFactor	0

Figure 5.9: HDL Properties window to set optimization parameters.

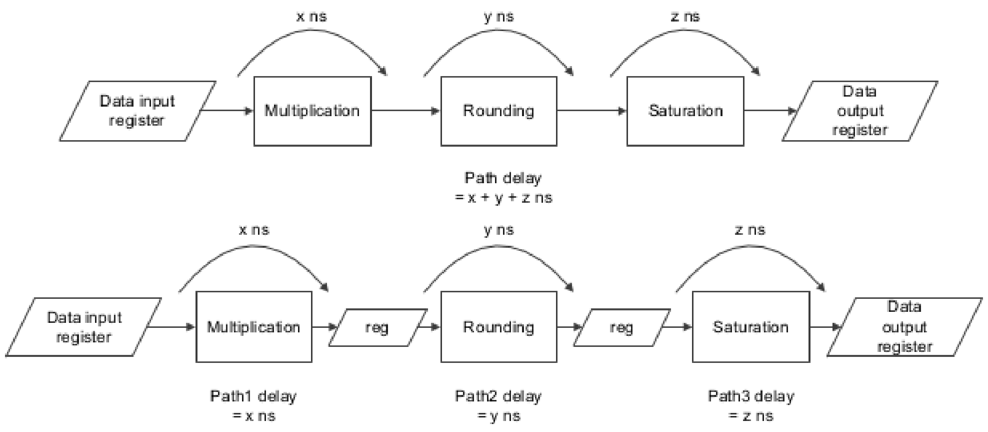


Figure 5.10: Illustration of adding registers in a long logic path.

requires additional pipeline registers and some logic around it to access RAM correctly.

CHAPTER 6

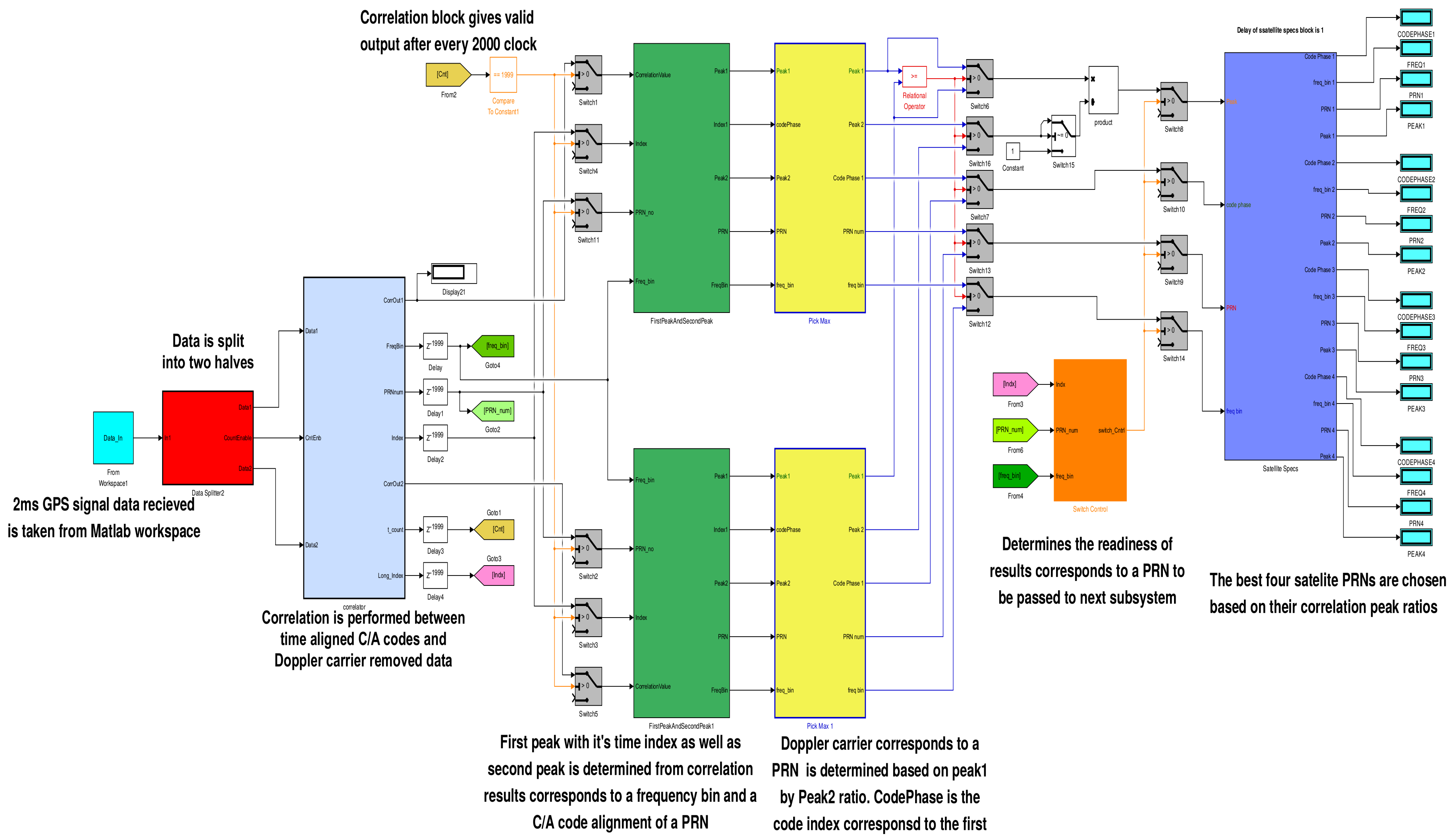
DESIGNING ACQUISITION ALGORITHM IN SIMULINK FOR HDL CODE GENERATION

The acquisition algorithm is designed using HDL coder library blocks in Simulink. The functional description of various blocks and subsystems in the design will be discussed in this chapter. The model is verified by running a simulation on the sample GPS signal data. The design involves look-up tables, delay blocks, memory blocks, logical circuits and arithmetic circuits etc. Mathematical operations like exponential carrier generation, taking the reciprocal etc cannot be implemented directly. However the HDL coder library is equipped with functional blocks which can carry out these operations either by look up tables, CORDIC algorithm or Newton-Raphson etc implementations. Such implementations are not included in the initial design(which is the floating point design). However in the later stages, these issues are taken care of along with the conversion of the floating point model to fixed-point model. The initial design need not be optimised in terms of timing and resources. Optimisation aspects come to picture only when the design is fully HDL compatible. Area and timing optimization can be performed depending upon the construction of the design in order to reduce the resource utilization and increase the clock speed.

6.1 Floating Point Model: Functional Blocks and Subsystems

Data samples for acquisition are stored in a workspace variable as a time series and taken into Simulink model using `FromWorkspace` block in Simulink library. Fig. 6.1 represents the initial floating-point design.

Floating Point Design of GPS Signal Acquistion in Simulink



Correlation block gives valid output after every 2000 clock
Data is split into two halves
2ms GPS signal data recieved is taken from Matlab workspace
Correlation is performed between time aligned C/A codes and Doppler carrier removed data
First peak with it's time index as well as second peak is determined from correlation results corresponds to a frequency bin and a C/A code alignment of a PRN
Doppler carrier corresponds to a PRN is determined based on peak1 by Peak2 ratio. CodePhase is the code index corresponsd to the first
Determines the readiness of results corresponds to a PRN to be passed to next subsystem
The best four satelite PRNs are chosen based on their correlation peak ratios
Delay of ssatellite specs block is 1
From Workspace1
Data Splitter2
correlator
FirstPeakAndSecondPeak
FirstPeakAndSecondPeak1
Pick Max
Pick Max 1
Relational Operator
Switch Control
Satellite Specs
Compare To Constant1

6.1.1 Data Splitter

The purpose of this block is to provide two consecutive $1ms$(in this case 2000 samples since $f_s = 2MHz$) received data as explained in section 3.5 for performing acquisition algorithm. The sample GPS signal data in the workspace contains samples received for a long time($>$35sec). Therefore `Data Splitter` subsystem has designed to take only 2ms of data into the Simulink model in order to perform the acquisition. The data is split into two, each having data of 1ms duration and they are streamed parallely as shown in Fig. 6.2. Since the same signal data has to be correlated with different PRN codes and Doppler carriers, we need to stream the same data repeatedly for each iteration. So `Data Splitter` block traps this 2000 samples in a loop and streamed repeatedly as shown in Fig. 6.3. HDL counter, deserializer , serializer, switch and memory etc are used to implement `Data Splitter` subsystem.

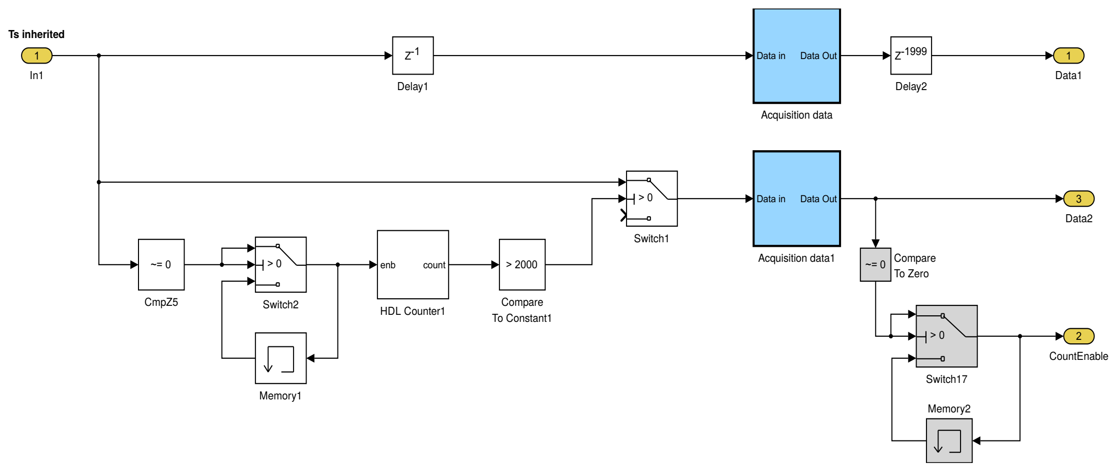


Figure 6.2: Construction of "Data Splitter" subsystem

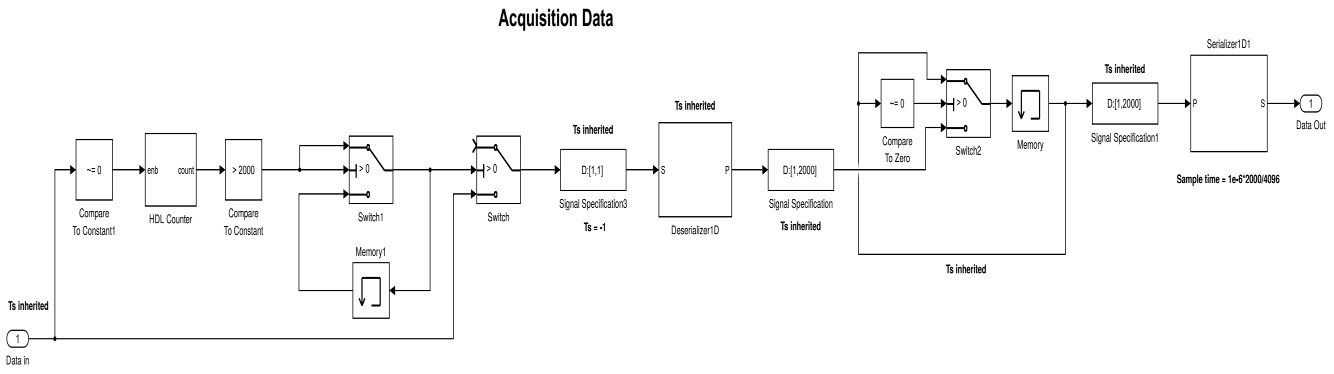


Figure 6.3: Construction of "Acquisition data" subsystem of "Data Splitter"

6.1.2 Correlator

Correlator block executes repeated correlation of various Doppler carrier removed data and the C/A code corresponds to each PRN. Frequency bins of width 500Hz have been arbitrarily chosen for locally generating Doppler carriers. It turns out that the Doppler shift in carrier frequency rarely exceeds 5kHz. Hence total 21 carriers are generated locally in order to cover $\pm 5kHz$ range. Fig. 6.4 shows the entire construction of the correlator block. The subsystem shown in Fig. 6.5 is responsible for periodically generating all the required Doppler carriers. C/A codes corresponds to each PRN is stored in a Look-up-Table in 'PRN code table' subsystem as shown in Fig. 6.7. The subsystem 'PRN index' shown in Fig. 6.6 helps to perform rotational shift of C/A codes and advance the correlation. The subsystem 'Block Sum' as shown in Fig. 6.8 implement the dot product functionality required to conduct correlation.

6.1.3 First Peak and Second Peak Finder

The block 'FirstPeakAndSecondPeak' finds out the first peak and second peak along with their indices(C/A code shifts) from each correlation output vector. It is done in two steps as shown in Fig. 6.9 :

1. Initially, finds out the first six peak points and their C/A code indices by subsystem 'Sorting in code phase' as shown in Fig. 6.10
2. Second peak is found out by avoiding peaks which fall in 'Exclude Range' by subsystem 'Choose Second Peak' as shown in Fig. 6.11.

The 'ExcludeRange' is two samples before and after the C/A code index corresponds to the first peak point. This 'ExcludeRange' has come into picture due to over sampling of C/A code. Therefore the second peak is chosen outside of 'ExcludeRange'.

6.1.4 Pick Max

The 'Pick Max' system helps to pull out the best correlation results corresponds to a PRN. It passes the the informations like code phase, Doppler carrier shift, first peak and second peak etc to the block 'Satellite Specs' corresponds to each PRN successively. The Simulink construction of 'Pick Max' is shown in Fig. 6.12.

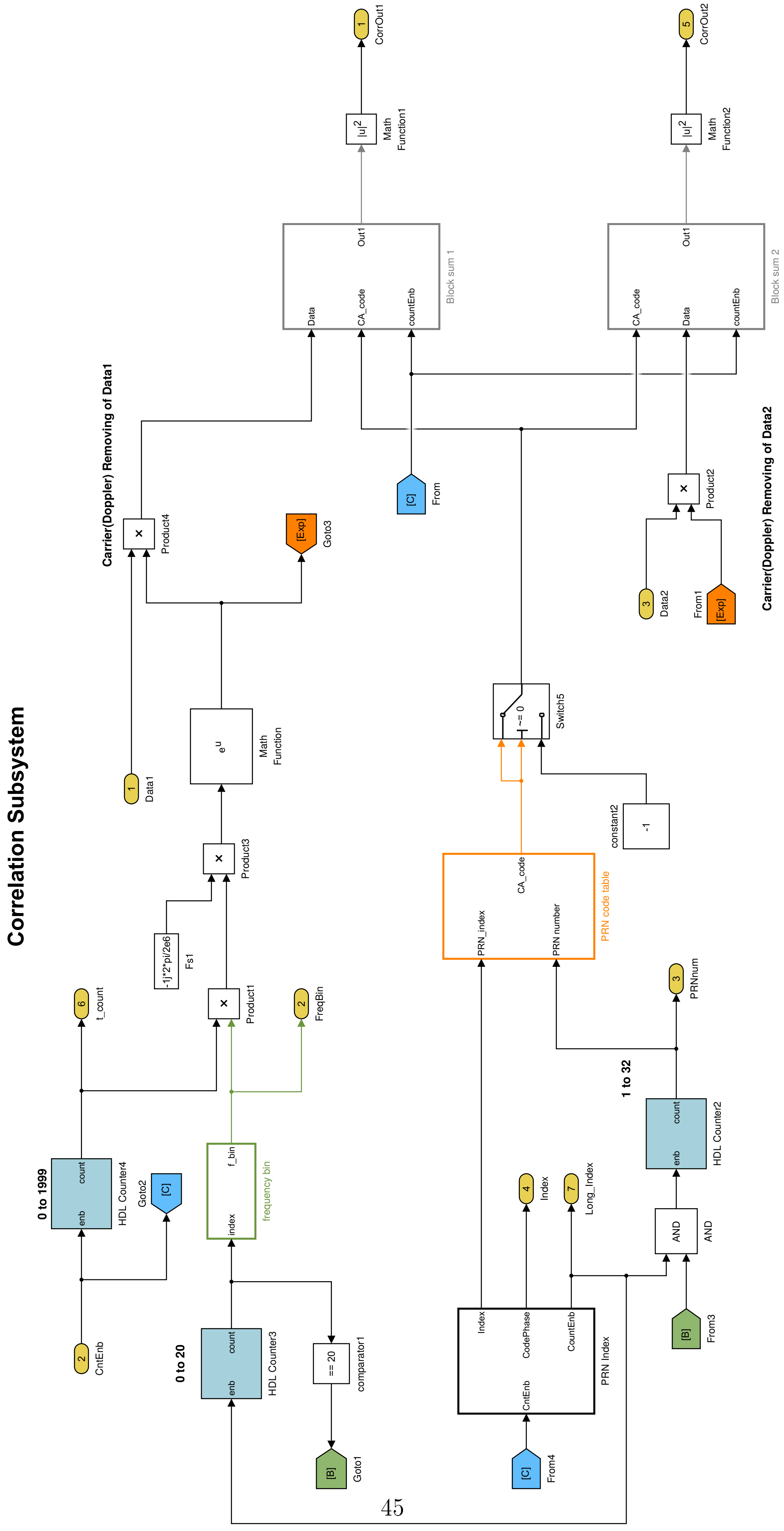


Figure 6.4: Correlator subsystem

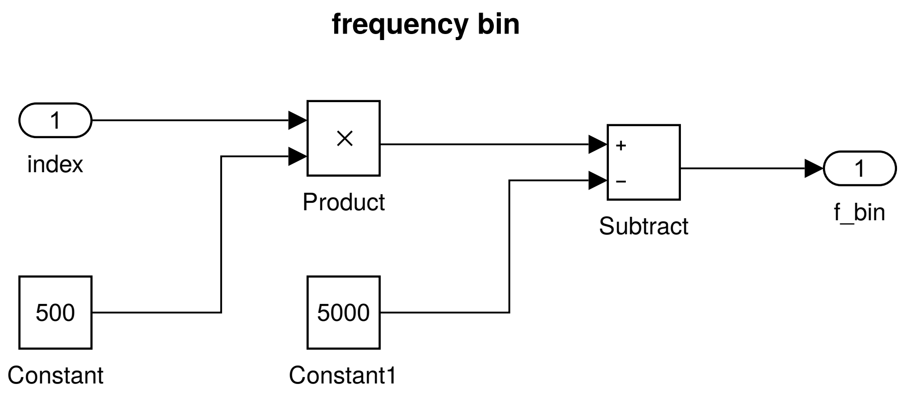


Figure 6.5: 'Frequency bin' subsystem of 'correlator' block

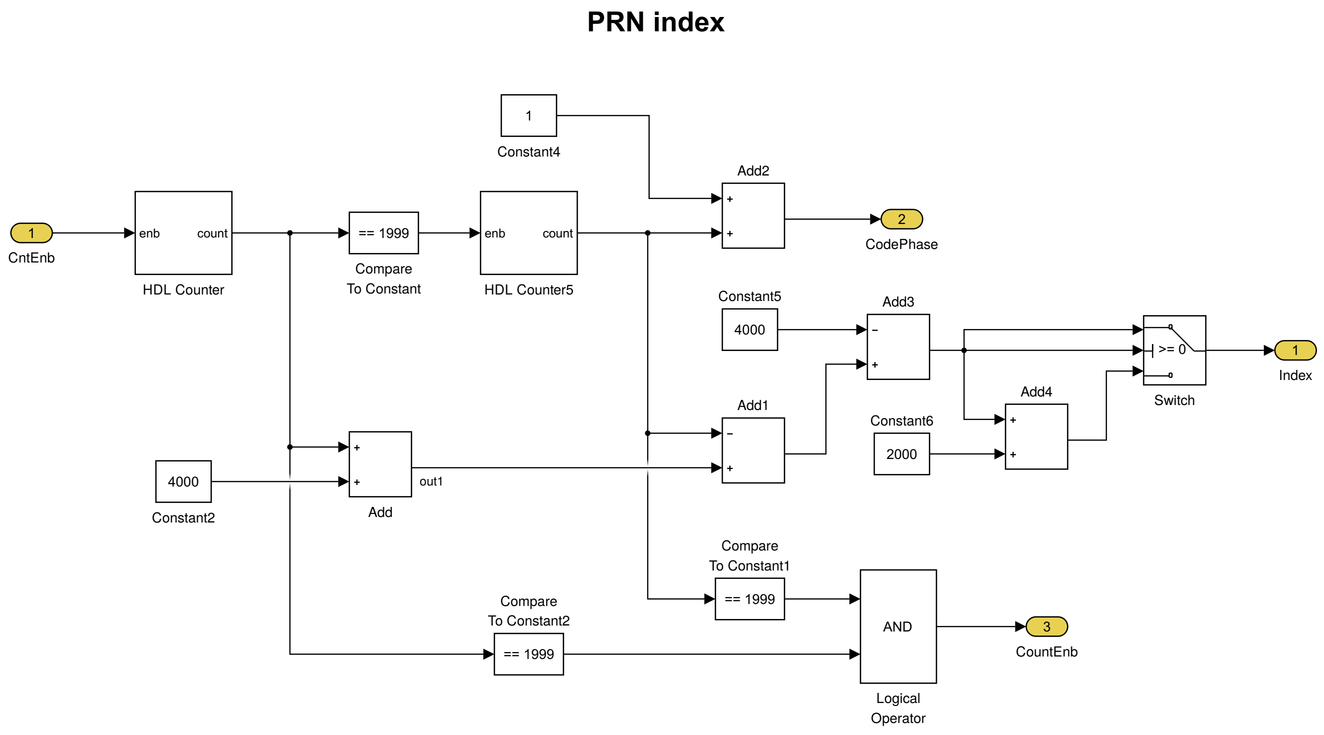


Figure 6.6: 'PRN index' subsystem of 'Correlator' block

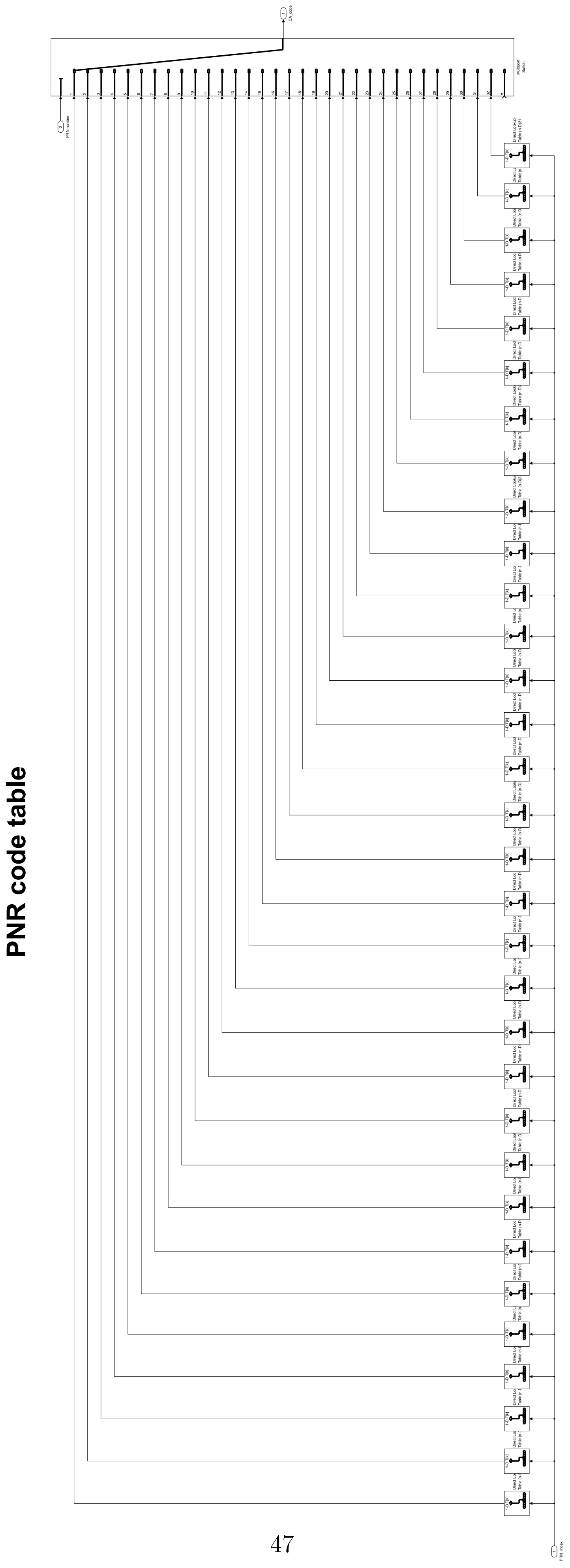


Figure 6.7: 'PRN code table' subsystem of 'Correlator' block

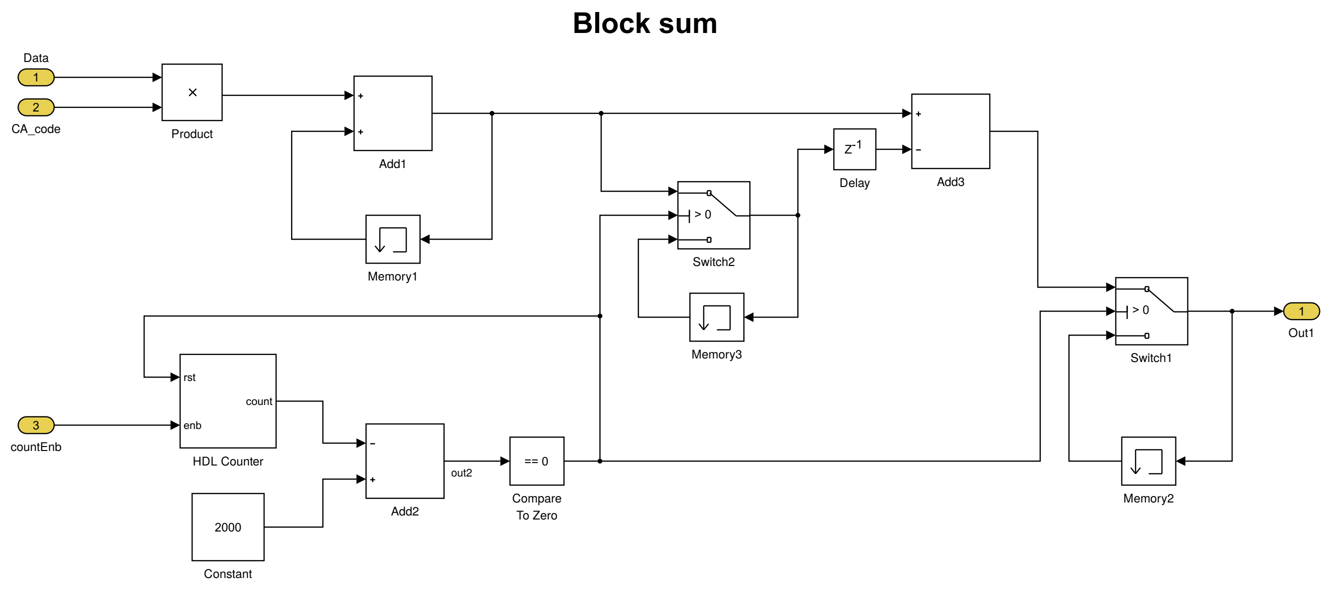


Figure 6.8: 'Block sum' subsystem of 'Correlator' block

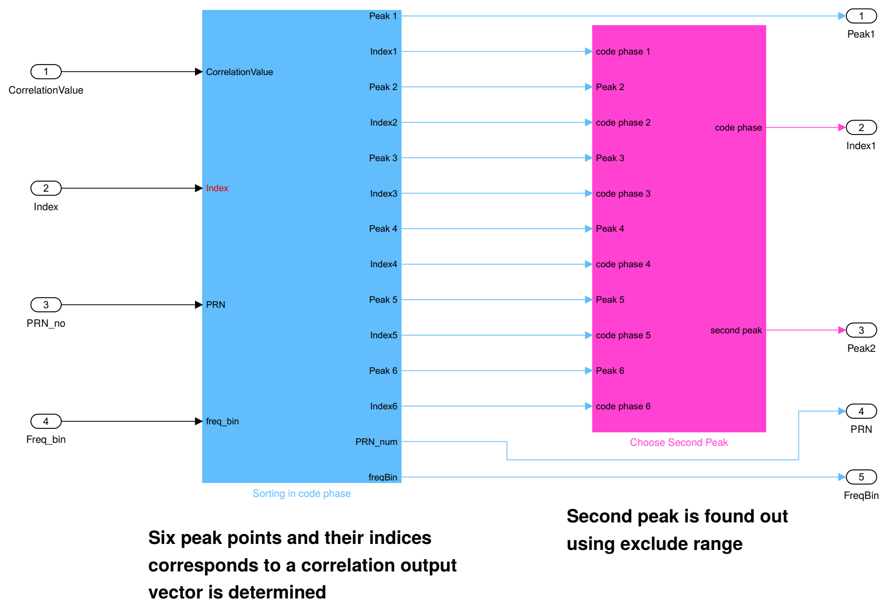


Figure 6.9: Inside the block 'FirstPeakAndSecondPeak'

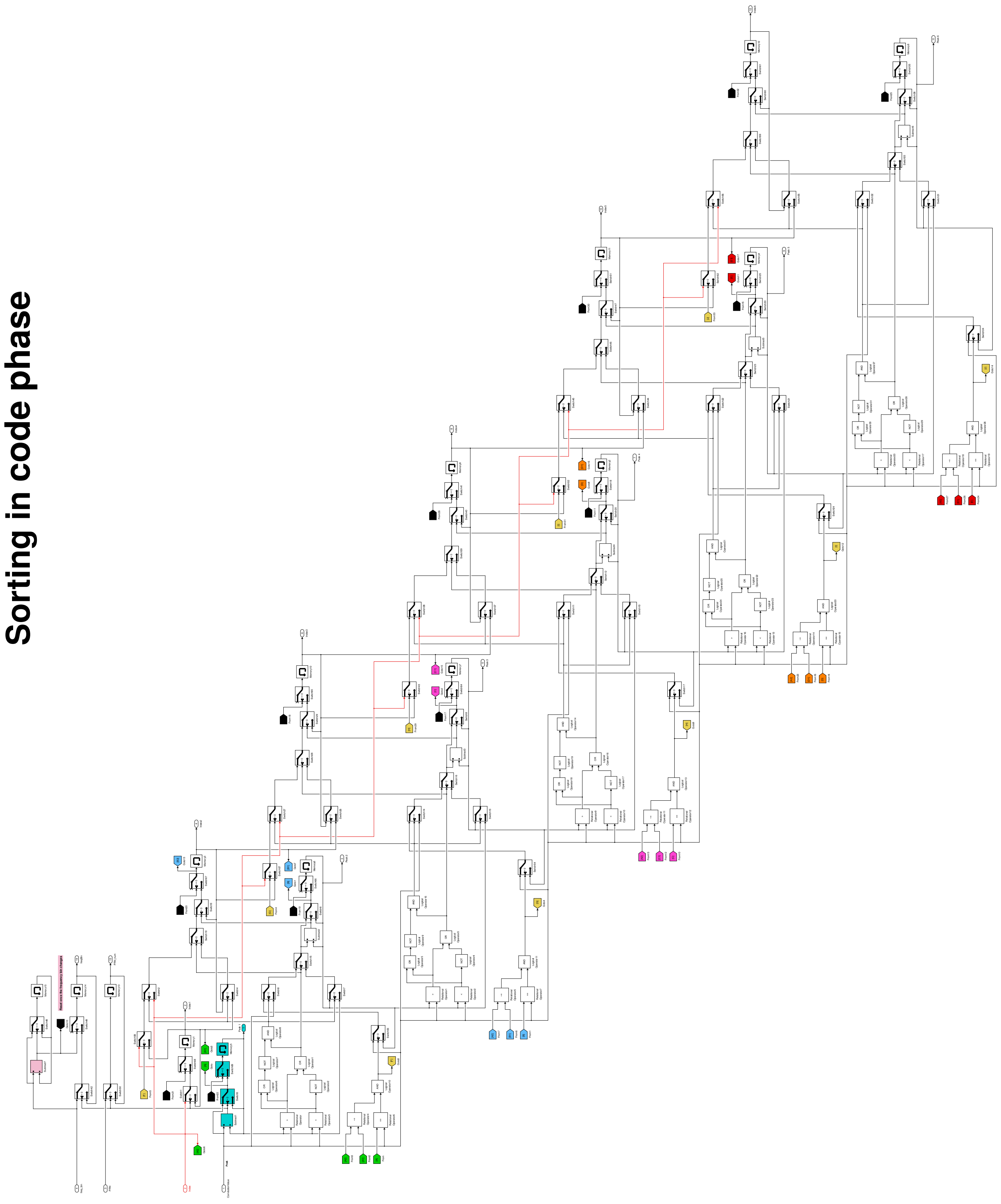


Figure 6.10: Construction of 'Sorting in code phase' subsystem of the block 'FirstPeakAnd-SecondPeak'

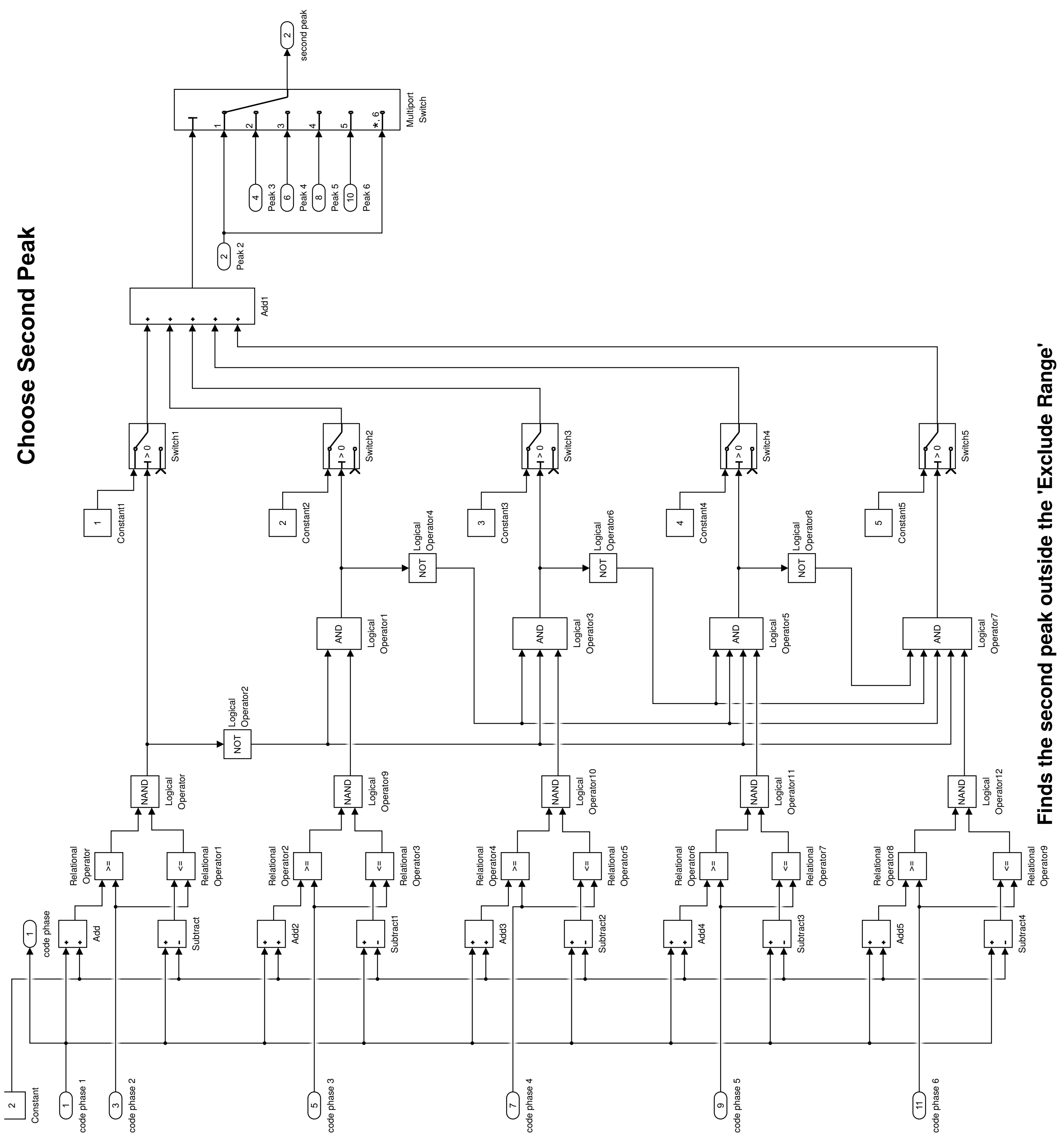


Figure 6.11: Construction of 'Choose Second Peak' subsystem of the block 'FirstPeakAndSecondPeak'

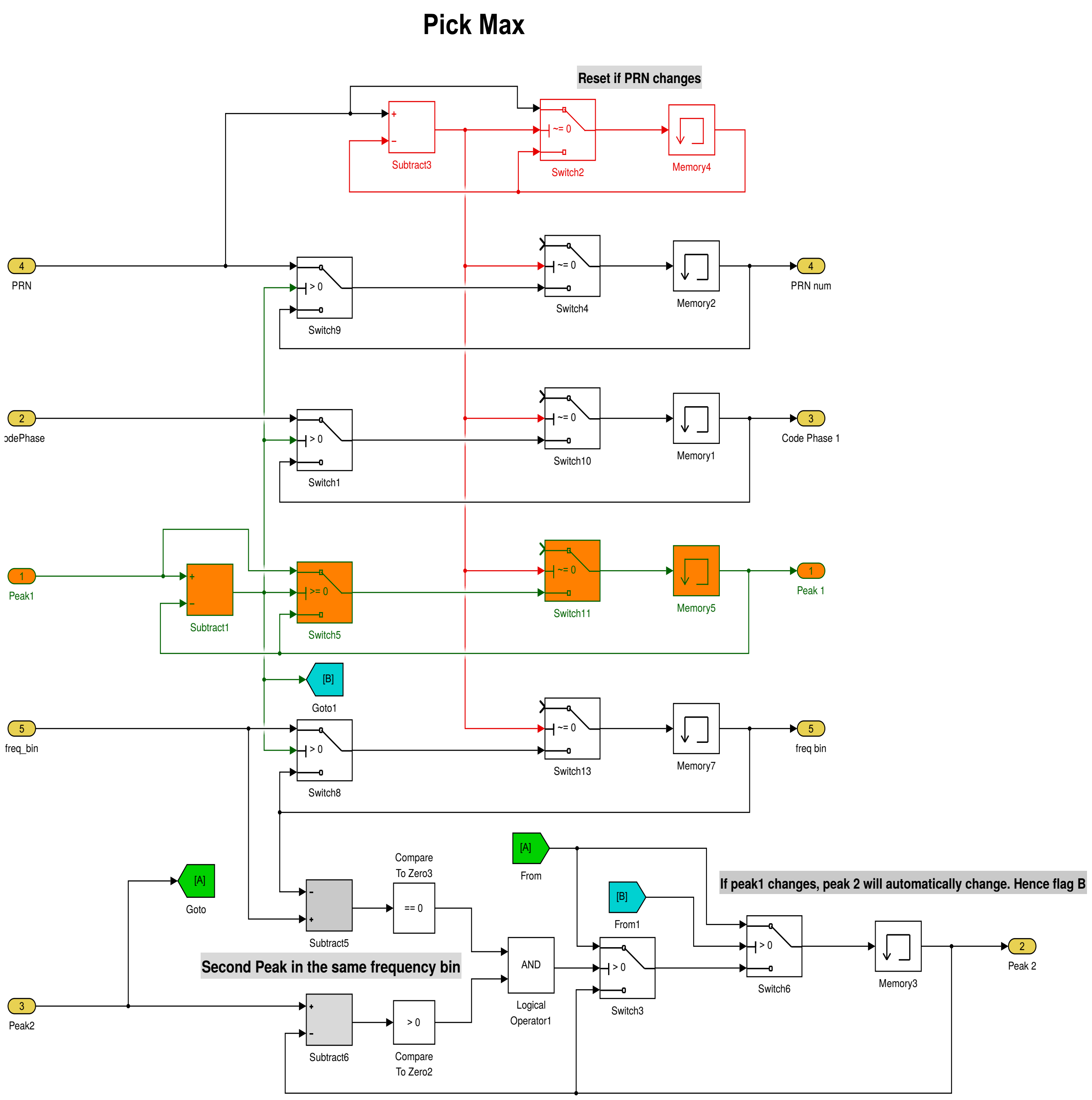


Figure 6.12: Construction of 'Pick Max' subsystem

6.1.5 Switch Control

The subsystem 'Switch Control' helps to identify the end of correlations corresponds to a PRN so that the results from 'Pick Max' subsystem can be passed to 'Satellite Specs'. The construction of 'Switch Control' is shown in Fig. 6.13.

6.1.6 Satellite Specs

The desire of this block is to sort correlation results corresponds to each PRN based on peak1(first peak) by peak2(second peak) ratio. Then the best four PRNs(Satellites) are chosen out of it. If peak1 by peak2 ratio of a satellite is above a predetermined threshold(In this case, it is taken to be 2) then it is declared as a visible satellite to the user. The design of 'Satellite Specs' system is shown in Fig. 6.14.

6.2 Results

A sample GPS signal data is used to test the design and the simulation results are verified with reference to the results of GNSS-SDR. GNSS-SDR is an open-source GNSS software receiver freely available to the research community. More details on GNSS-SDR and its installation is given in Appendix. **??**. The simulation results are documented in Table. 6.1.

Channel	Peak Ratio	PRN	Doppler Frequency Bin	Code Phase
1	28	7.4504	1000	338
2	6	4.2549	500	1751
3	2	4.0474	2500	689
4	17	3.9556	-500	977

Table 6.1: Acquisition simulation results

6.3 Fixed Point Conversion of the Design

Prior to the HDL code generation, the floating point model need to be converted to fixed point. The fixed point model will have fixed word lengths for various blocks included in the design. This will result in quantisation errors and in some cases saturation of included

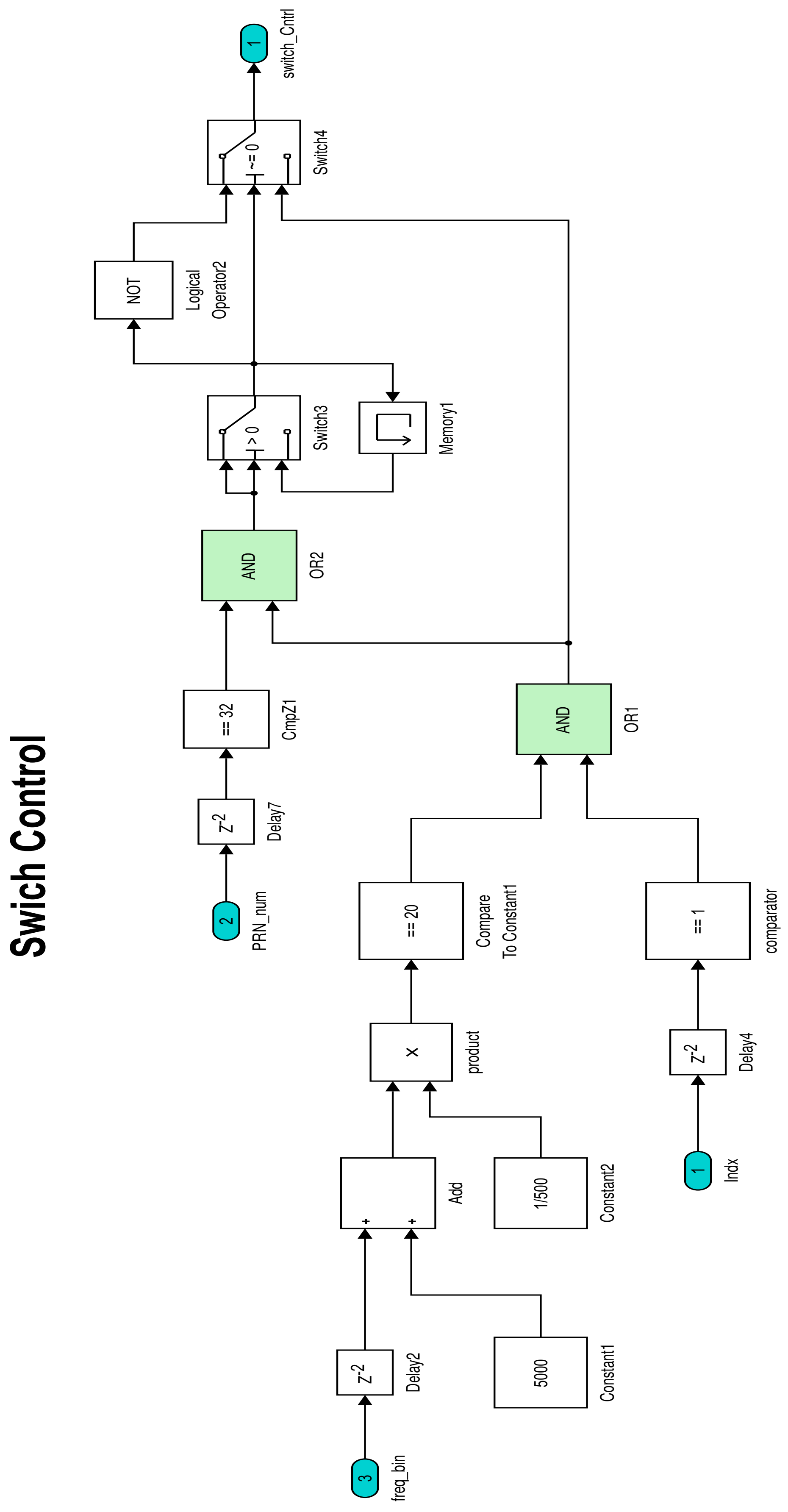


Figure 6.13: Construction of 'Switch Control' subsystem

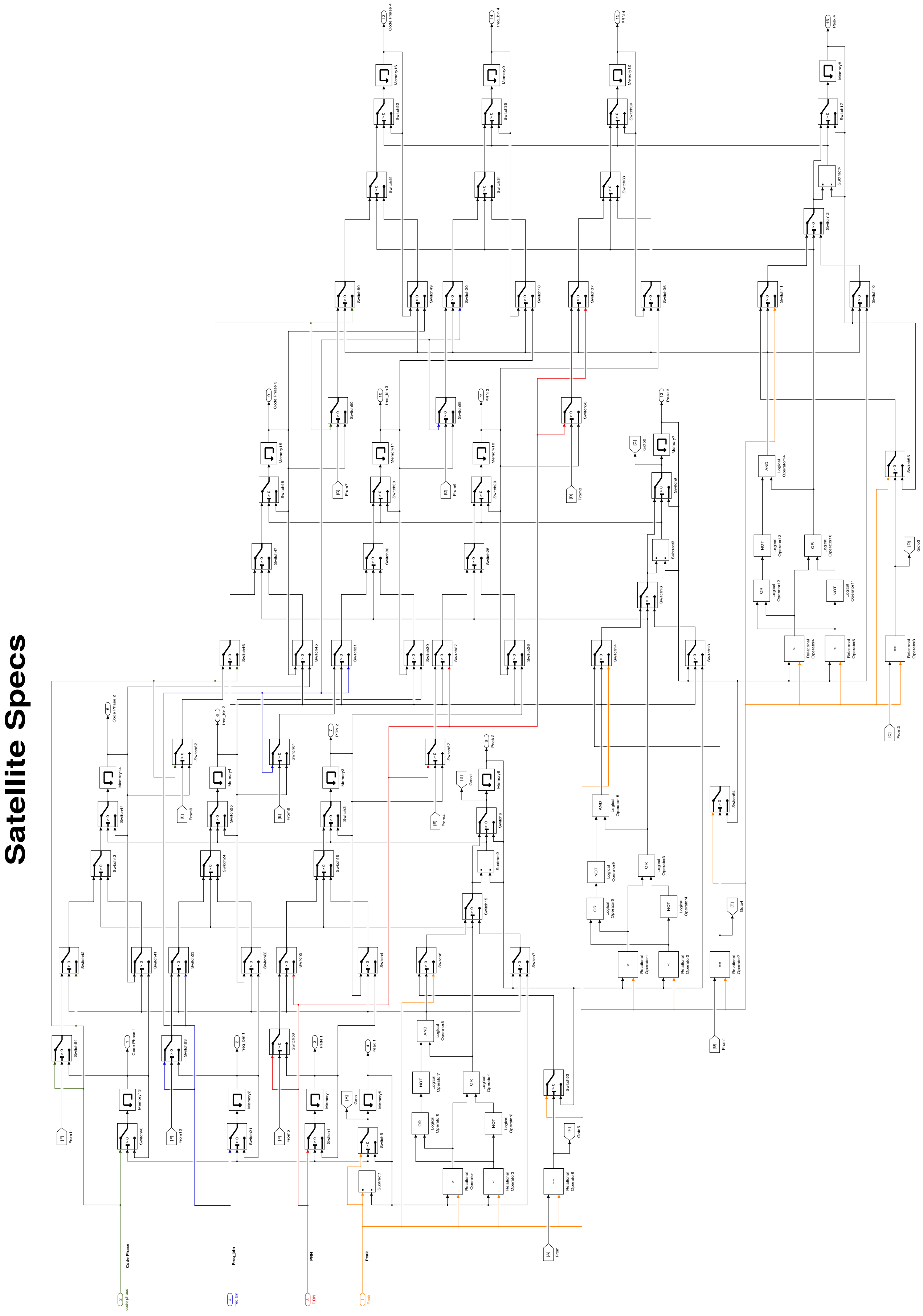


Figure 6.14: Construction of 'Satellite Specs' subsystem

blocks for out of bound inputs. Much care need to be given to fixed point conversion as it plays a major role in resource utilisation and accuracy of the results. HDL code generation from the fixed point model is straight forward using the HDL workflow advisor. Fixed point conversion of the floating point model can be done by the Fixed point designer as well as manually adjusting the data types of Simulink blocks from properties.

6.3.1 Using Fixed Point Designer

Fixed point designer is equipped with the essential steps for converting the floating point model to fixed point. The tool initially derives the range information of the outputs of various blocks by running a simulation over a pre- scribed data set. The range information regarding external inputs need to be set manually. Once the range information of all the blocks in the model are derived, fixed point designer further uses these informations for estimating the required word length at the output of various blocks.
In the Fixed point Designer window pain there is provision to see if saturation occurs at any point in the design due to out of bound inputs. The designer can manually set the required word lengths according to the accuracy requirements. The quantisation errors may accumulate over a certain path and result in erroneous outputs. The word lengths should be increased in such cases to reduce the error. There is always a trade off between word length optimisation and accuracy of the output. The word length of various blocks should be selected such that the quantisation errors lie within a specified range.

The fixed point model of 'Block sum' is given in Fig. 6.15 with all the port data types shown. Fixed point data types can be signed or unsigned. `sfix` refers to signed and `ufix` refers to unsigned data types. `sfix32_En24` means signed data type of word length 32 and 24 bits out of these 32 are used for representing decimal part. So the integer part has 8 bits left and the range will be -128 to 127.
As discussed in chapter 5, certain mathematical operations such as tan^{-1} , division, complex magnitude, exponential wave form generation etc can't be implemented in a single step in FPGA. Even though the HDL coder library is equipped with blocks to carry out such operations, these blocks may not be supported during the HDL code generation. Alternatively these operations can be implemented by Look up tables, CORDIC algorithm or mathematical approximations and Simulink provides such blocks exclusively for HDL code

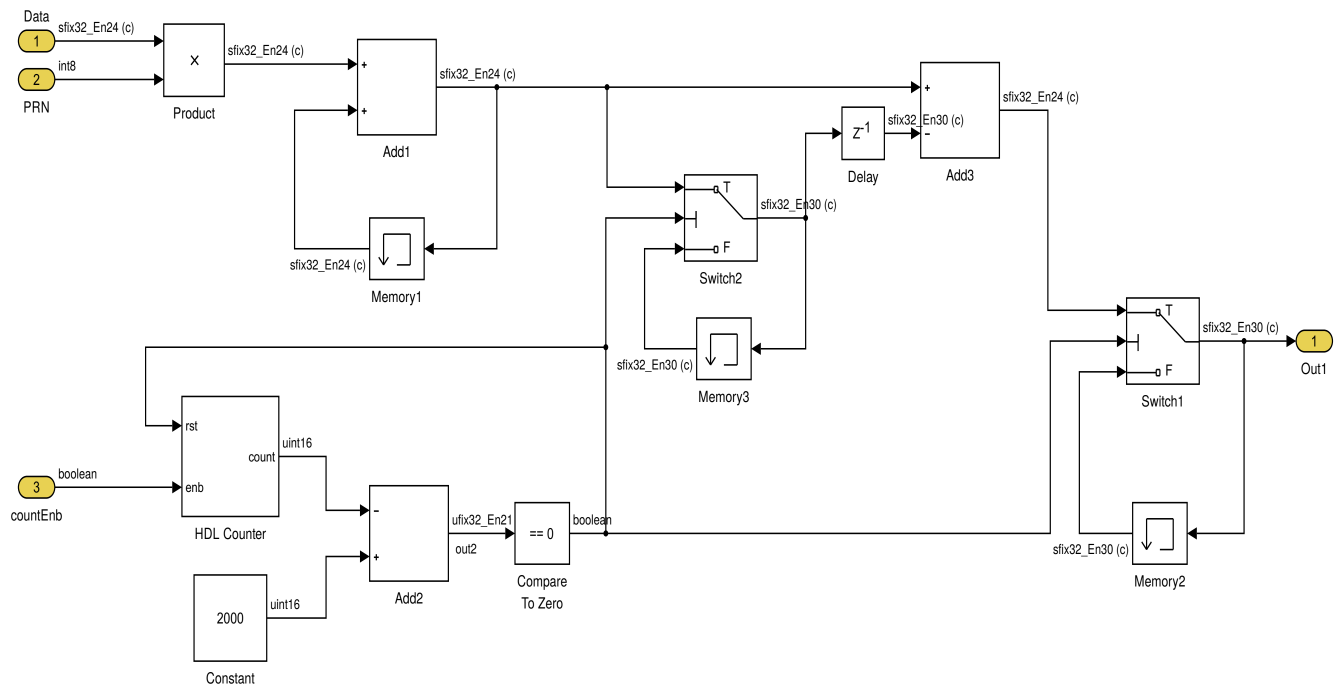


Figure 6.15: Fixed point model of 'Block sum'

generation.

CHAPTER 7

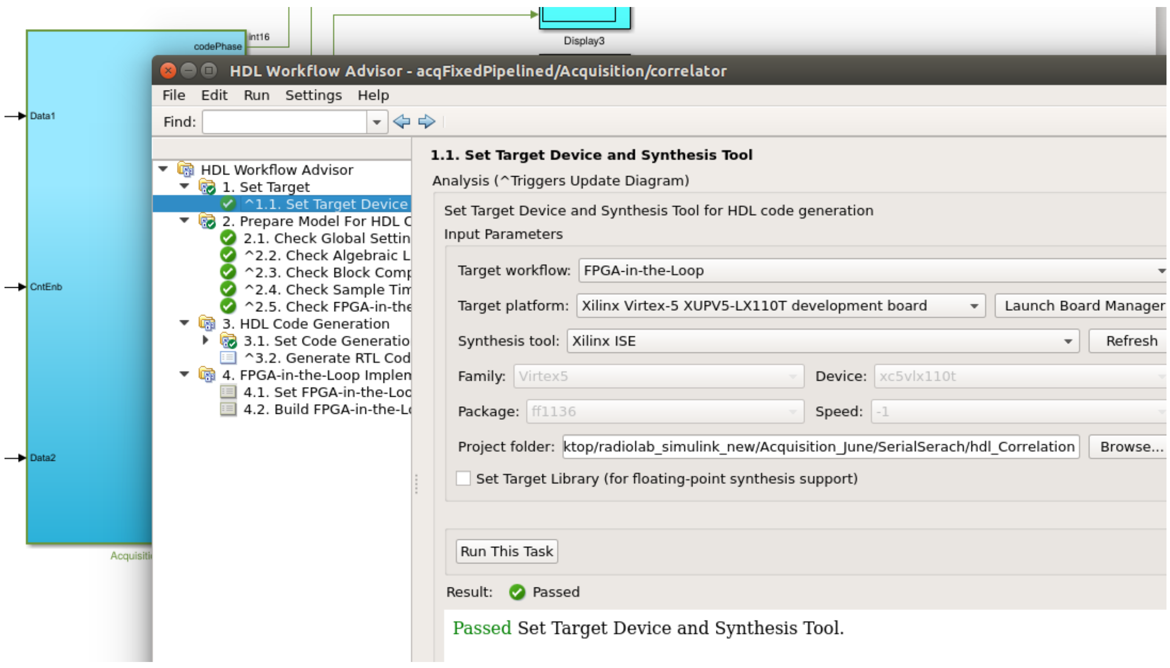
HDL CODE GENERATION AND VALIDATION

HDL code generation can be performed readily by the help of `HDL workflow adviser` once the model is fully converted to fixed-point and all the blocks used in the design is HDL compatible. In this project, the code generation and verification are conducted on a Xilinx Virtex-5 XUPVS-LX110T development board and the synthesis tool used is Xilinx ISE. For advanced FPGA boards such as Virtex-7, Zynq-7000 the synthesis tool required is Xilinx Vivado. By selecting the target workflow as FPGA-in-the-Loop, the HDL verifier allows HDL co-simulation in which the output from the FPGA and the simulation output of corresponding Simulink model can be observed simultaneously. A brief description of principles pertaining to HDL code generation from model is depicted in section. 5.2

7.1 HDL Code Generation via HDL Workflow Adviser

HDL workflow adviser automates various steps required for programming Xilinx and Altera FPGAs from Simulink/Matlab model. User can set the target workflow, target FPGA platform, synthesis tool and project folder etc as shown in Fig. 7.1. After configuring the target specific parameters, HDL coder performs checks for global settings, algebraic loops, compatibility and sample time to verify that the design is synthesizable. Depending on the design size, the RTL synthesis takes from a few seconds to a few minutes. The output is human readable VHDL/VERILOG files and all the blocks inside a top-level design are synthesized into separate VHDL/VERILOG files and are imported as components on the top-level VHDL/VERILOG.

User can opt for generating traceability, resource utilization, critical path, optimization reports and validation model via 'HDL coder properties' of the design under target(DUT) along with RTL synthesis. Traceability report allows navigation between the code and the corresponding Simulink blocks through hyperlinks. The timing and high level resource utilisation reports can be used as a reference to design optimisation. The critical path of the



HDL Workflow Advisor - acqFixedPipelined/Acquisition/correlator

HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device
2. Prepare Model For HDL C
2.1. Check Global Settin
^2.2. Check Algebraic L
^2.3. Check Block Comp
^2.4. Check Sample Tim
^2.5. Check FPGA-in-the
3. HDL Code Generation
3.1. Set Code Generatio
^3.2. Generate RTL Cod
4. FPGA-in-the-Loop Implen
4.1. Set FPGA-in-the-Loo
4.2. Build FPGA-in-the-L

1.1. Set Target Device and Synthesis Tool

Analysis (^Triggers Update Diagram)

Set Target Device and Synthesis Tool for HDL code generation

Input Parameters

Target workflow: FPGA-in-the-Loop

Target platform: Xilinx Virtex-5 XUPV5-LX110T development board

Synthesis tool: Xilinx ISE

Family: Virtex5 Device: xc5vlx110t

Package: ff1136 Speed: -1

Project folder: ktop/radiolab_simulink_new/Acquisition_June/SerialSerach/hdl_Correlation

Set Target Library (for floating-point synthesis support)

Result: Passed

Passed Set Target Device and Synthesis Tool.

Figure 7.1: HDL workflow adviser

design will be depicted using highlighted blocks which is useful for identifying the scope of pipe-lining. Validation model is used for validating the functional behaviour of the generated HDL code with the help of HDL Verifier as explained in Section. 7.2.

HDL workflow adviser proceeds for logic synthesis once the RTL synthesis is completed. However, user can optimize the design in terms of area and speed by examining the RTL synthesis reports, which has been discussed in Chapter. 8.

7.2 RTL Verification Using Validation Model

Validation model enables simultaneous simulation of the Simulink model and corresponding RTL generated. Which also provides a platform to analyse outputs of both model and the error occurred through a scope. In this project, subsystem wise validation is carried out as the simulation of the top-level design is time consuming.

The validation model corresponding to 'correlator' block is shown in Fig. 7.2. The outputs CorrOut1 and CorrOut2 are analysed through scopes shown in Fig. 7.3 and Fig. 7.4 respectively. The RTL model generated is verified for its functional behaviour as the outputs from both models exactly matches and the error continues to be zero. In this fashion RTL model corresponds to every subsystems can be verified for its functional behaviour using validation model.

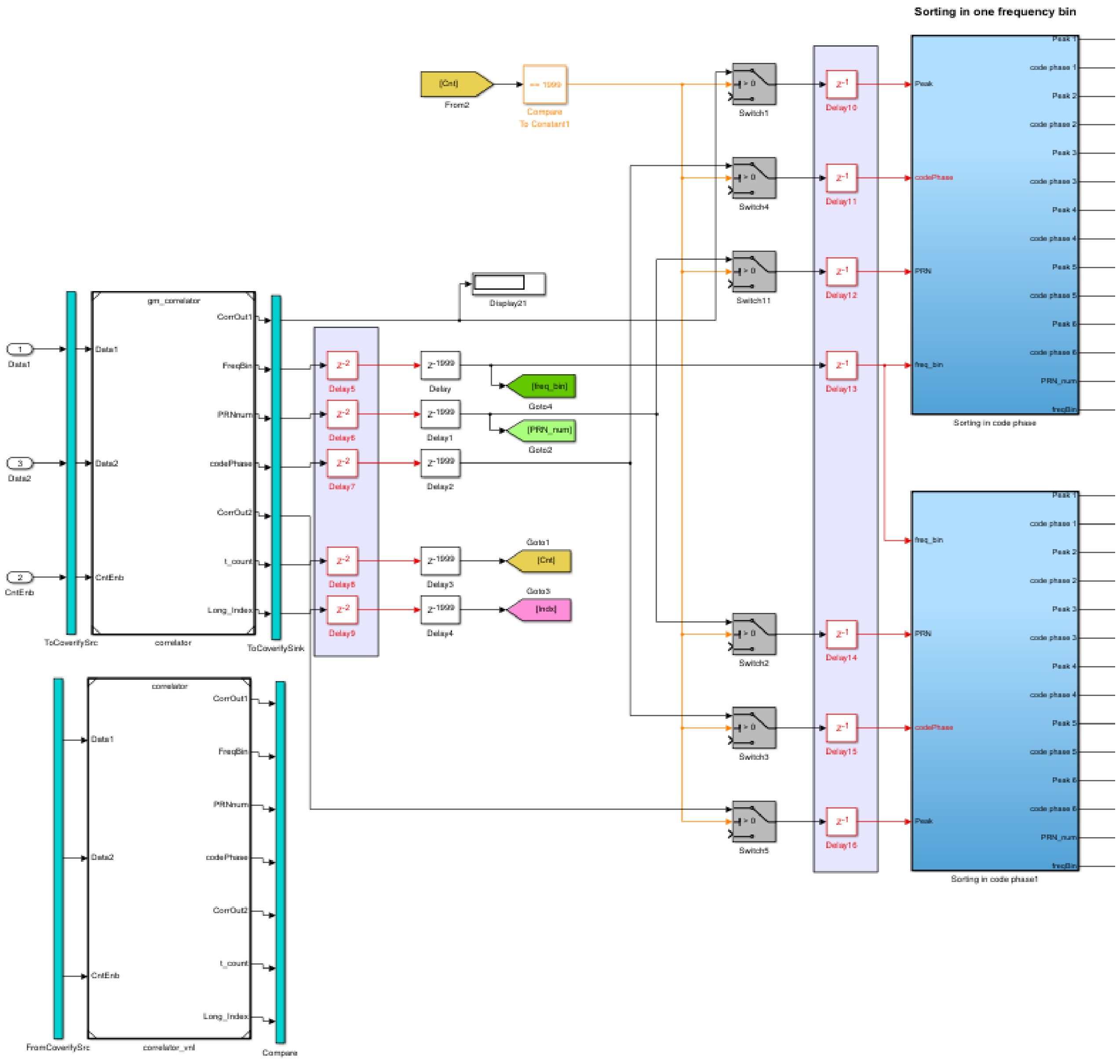


Figure 7.2: Validation model generated corresponds to ’correlator’ subsystem

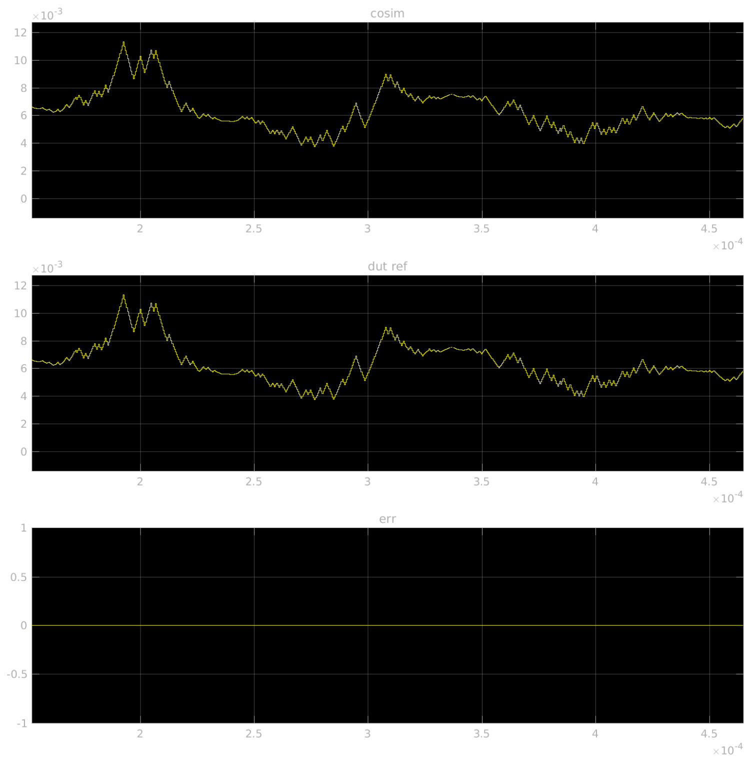


Figure 7.3: The output 'CorrOut1' from both Simulink and RTL model during simulation of the validation model

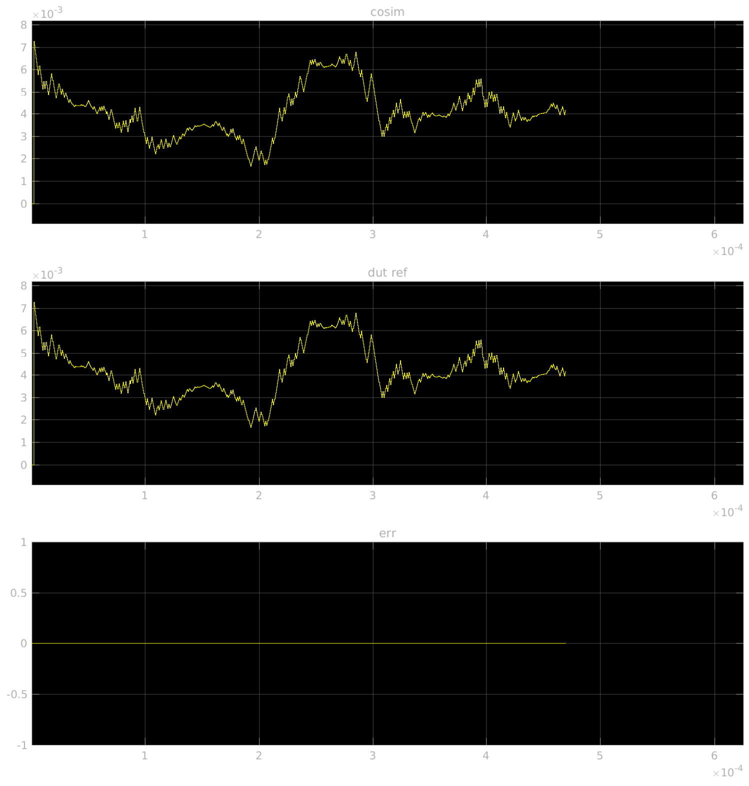


Figure 7.4: The output 'CorrOut2' from both Simulink and RTL model during simulation of the validation model

CHAPTER 8

OPTIMIZATION AND SYNTHESIS OF THE HDL CODE

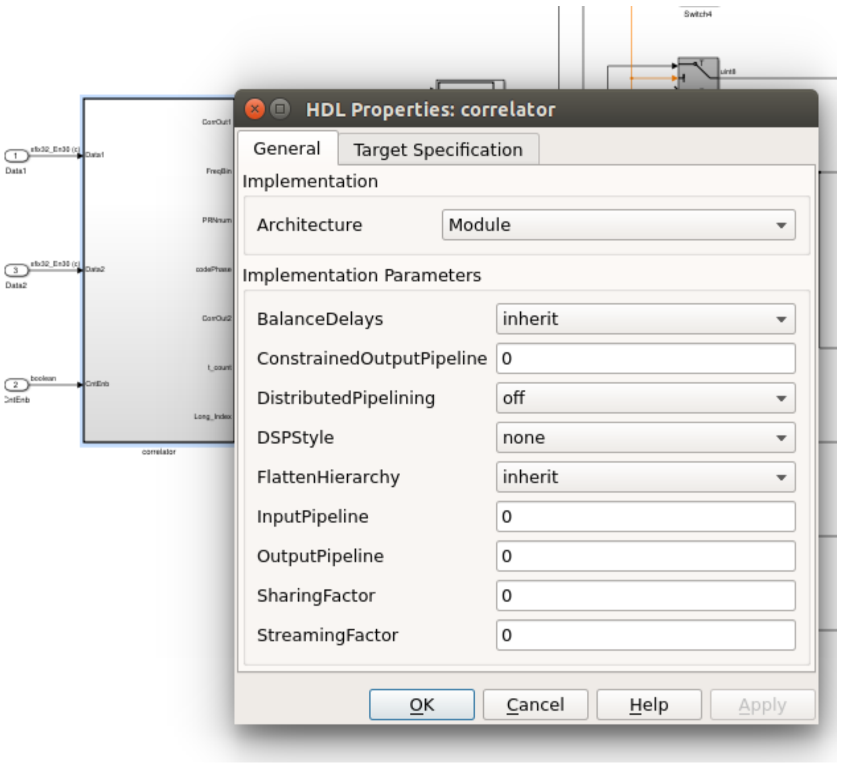
Timing and area optimisation plays a key role in achieving faster clock rates and a concise design that effectively fits into the target FPGA device respectively. Traceability reports can be used to find the high level resource utilisation and critical path in the current design. Area optimisation is mainly done by reducing the word length of various blocks without affecting the accuracy of results. Multiplexing can also be done for functionally-equivalent Simulink blocks by enabling resource sharing so as to further optimize area. Timing optimisation is mainly done by fragmenting the critical path using delays and it is known as pipelining. Via HDL block properties window pane user can set various optimize features like `DistributedPipelining`, `SharingFactor` and `StreamingFactor` etc as shown in Fig. 8.1. HDL coder will then automatically enable these optimization features for the subsystem during the code generation.

Register usage can be reduced tremendously by allowing delays in the design to use RAM by enabling `UseRAM` option in the HDL properties pane of the delay block as shown in Fig. 8.2.

The optimised design of Acquisition Algorithm is shown in Fig. 8.3.

8.1 Critical Path Estimate and Resource Report

The critical path estimate for the unoptimized initial design was 103.978ns which is highlighted in Fig. 8.4. The critical path estimate after optimizing the design by pipe-lining was reduced to 49.801ns which is highlighted in Fig. 8.5. The high level resource utilization report for the unoptimized design and the optimized design is shown in Fig. 8.6 and Fig. 8.7 respectively.



HDL Properties: correlator

General | Target Specification

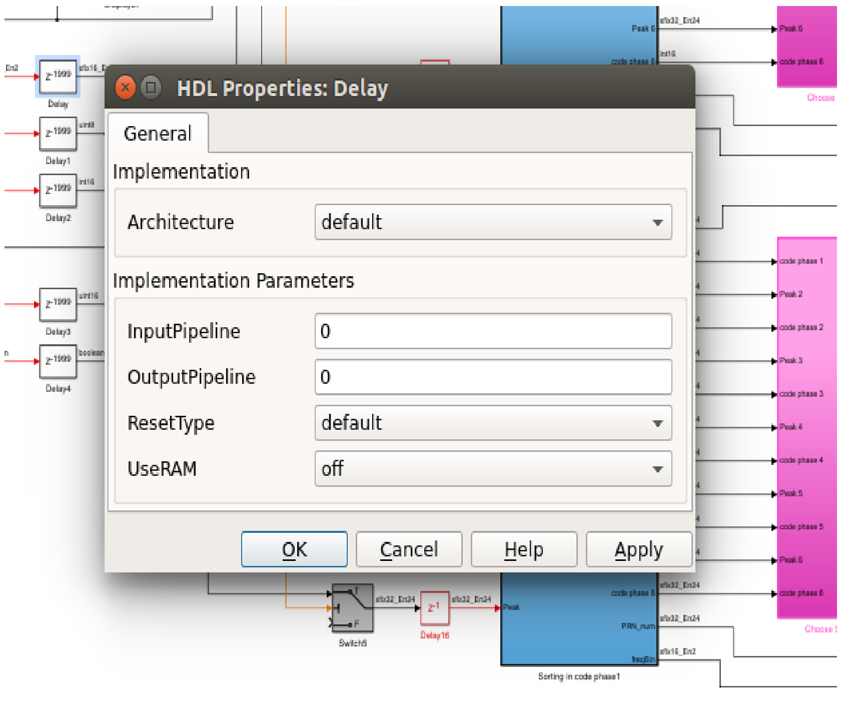
Implementation

Architecture: Module

Implementation Parameters

BalanceDelays: inherit
ConstrainedOutputPipeline: 0
DistributedPipelining: off
DSPStyle: none
FlattenHierarchy: inherit
InputPipeline: 0
OutputPipeline: 0
SharingFactor: 0
StreamingFactor: 0

Figure 8.1: HDL block properties



HDL Properties: Delay

General

Implementation

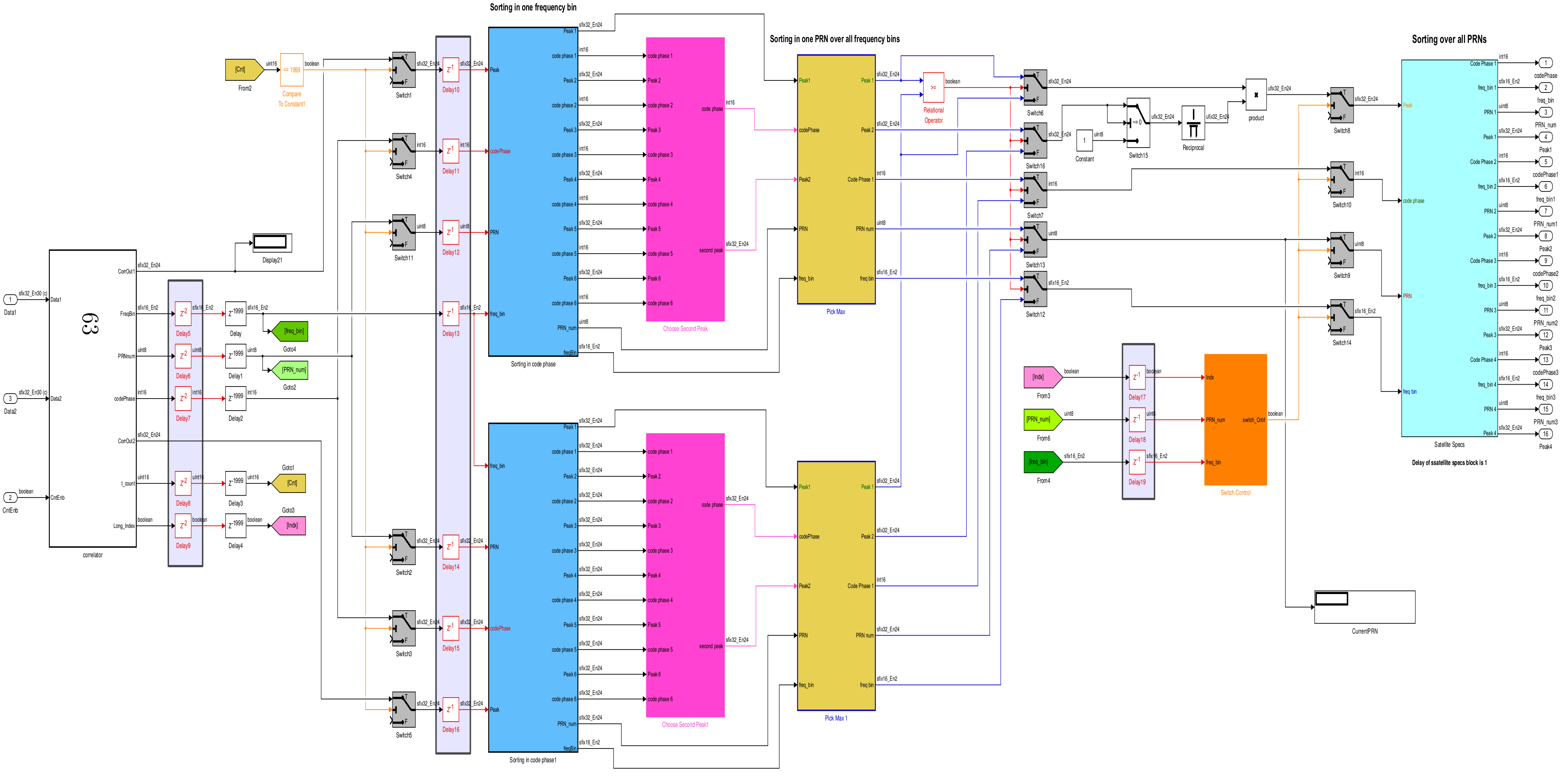
Architecture: default

Implementation Parameters

InputPipeline: 0
OutputPipeline: 0
ResetType: default
UseRAM: off

Figure 8.2: HDL Properties: Delay

Optimized Acquisition Algorithm



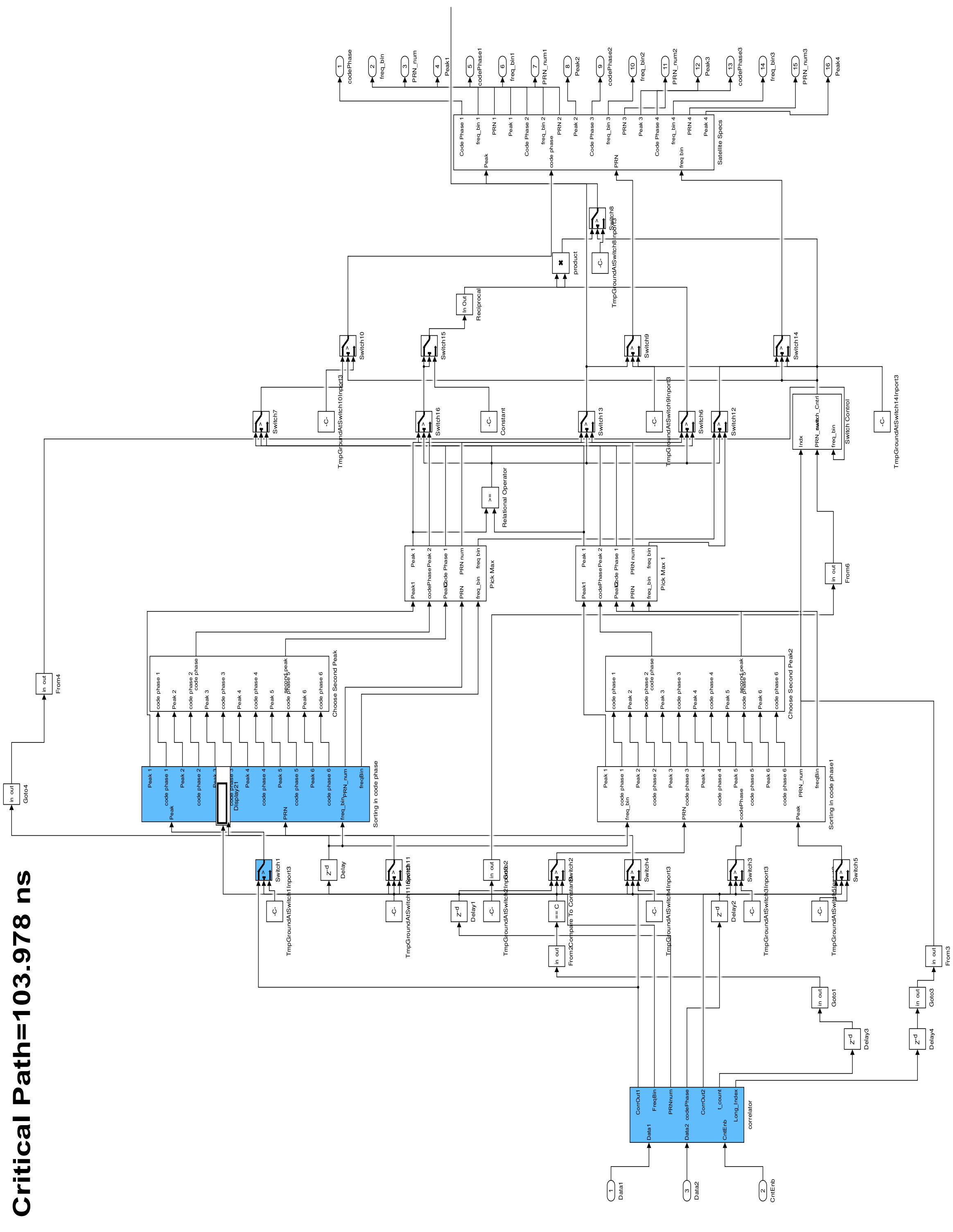


Figure 8.4: The critical path corresponds to unoptimized design

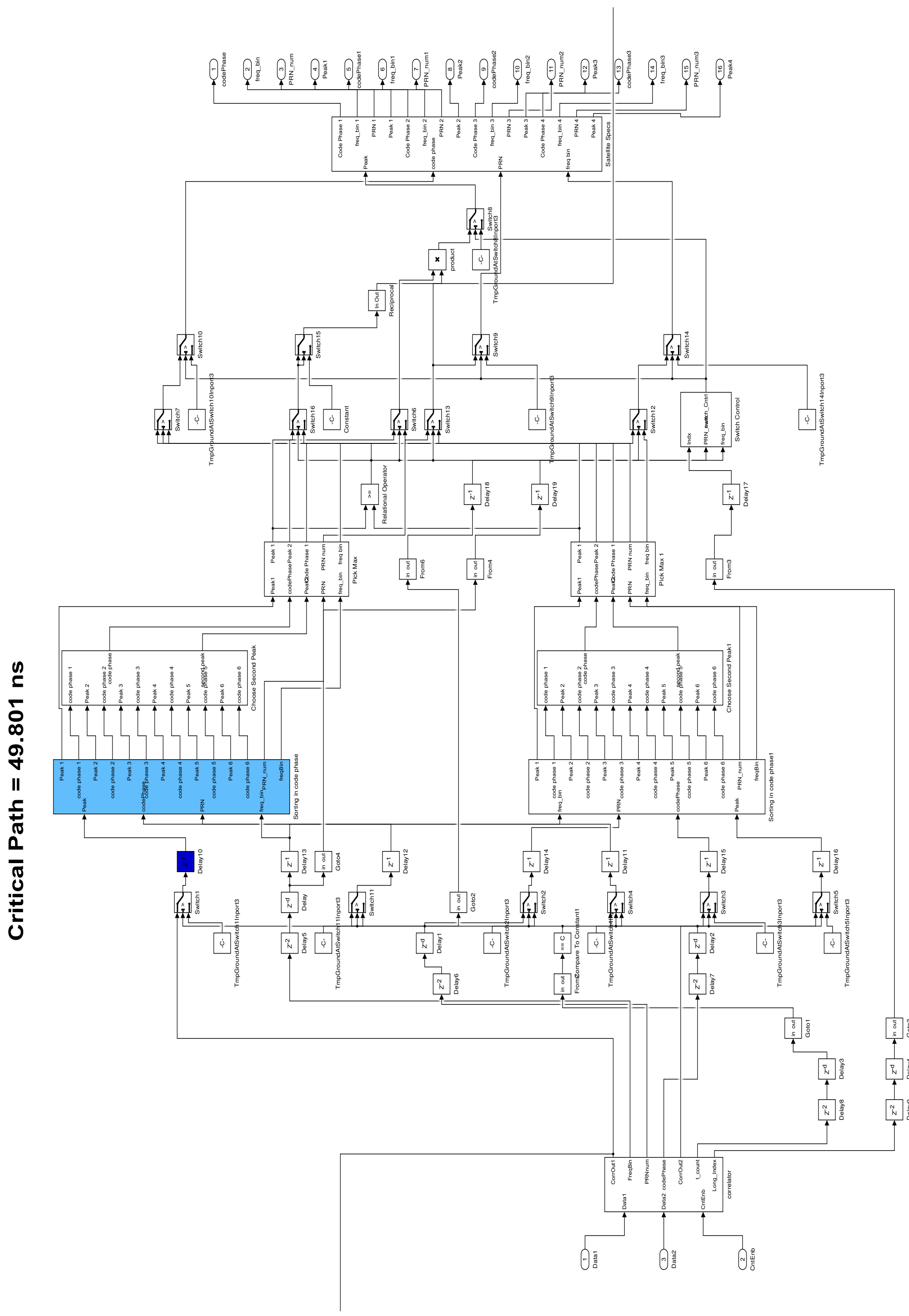
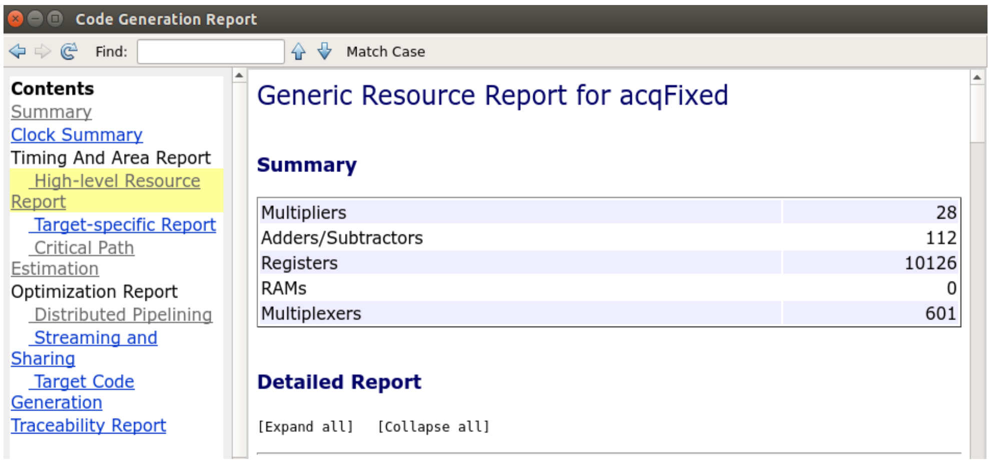


Figure 8.5: The critical path corresponds to optimized design


Code Generation Report

Contents
Summary
Clock Summary
Timing And Area Report
High-level Resource Report
Target-specific Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Target Code Generation
Traceability Report

Generic Resource Report for acqFixed

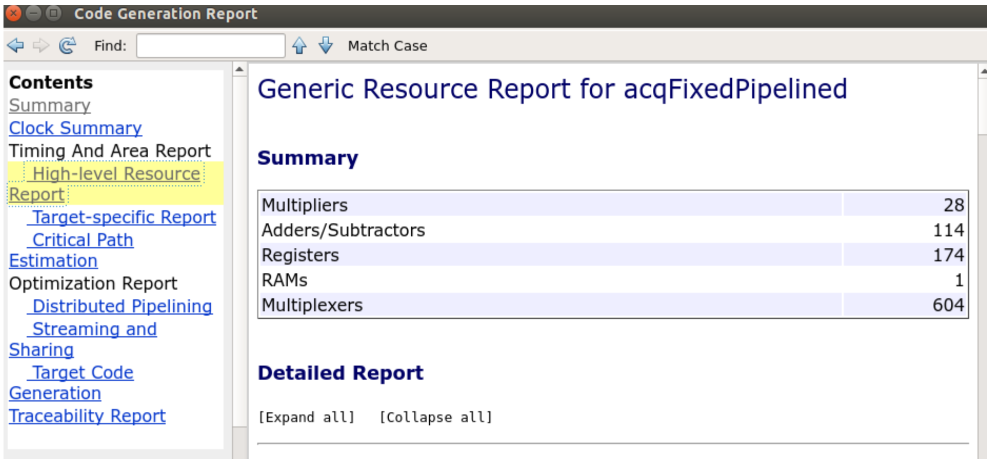
Summary

Multipliers	28
Adders/Subtractors	112
Registers	10126
RAMs	0
Multiplexers	601

Detailed Report

[Expand all] [Collapse all]

Figure 8.6: High level resource utilization report of the unoptimized design


Code Generation Report

Contents
Summary
Clock Summary
Timing And Area Report
High-level Resource Report
Target-specific Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Target Code Generation
Traceability Report

Generic Resource Report for acqFixedPipelined

Summary

Multipliers	28
Adders/Subtractors	114
Registers	174
RAMs	1
Multiplexers	604

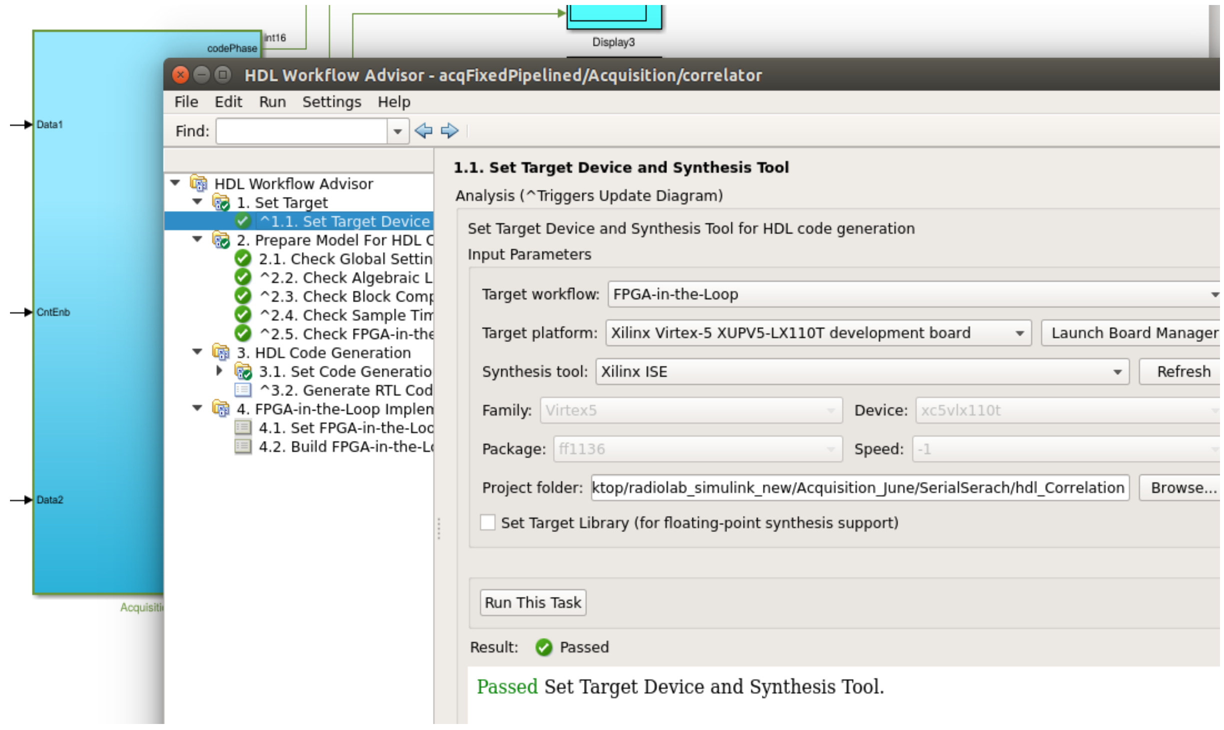
Detailed Report

[Expand all] [Collapse all]

Figure 8.7: High level resource utilization report of the optimized design

8.2 Logic Synthesis and Functional Verification in FPGA Environment

The logic synthesis and functional verification of the design in FPGA in the Loop(FIL) configuration is previously discussed in Section. 5.4. The optimized HDL code is synthesized using Xilinx ISE on Xilinx XUPV5-LX110T Evaluation Platform via HDL Workflow Adviser. See Fig. 8.8. A Simulink model is generated during the synthesis which enables HDL cosimulation in order to verify the functionality in FPGA platform with the help of HDL Verifier^{TM} as shown in Fig. 8.9. The bit file generated after the synthesis can be downloaded to FPGA by double clicking on FIL block and then loading the bit file as shown in Fig. 8.10. Now HDL cosimulation can be performed by running the simulation. The output from the FPGA and the simulation output of Simulink model can be seen simultaneously using scopes provided in Simulink. The error between the two outputs is also shown in the scope and the model can be verified for correctness if the error is always zero. Once the functionality is verified, user can remove the Simulink counterpart and work only on FPGA counterpart in order to have fast communication between FPGA and host computer as shown in Fig. 8.11.



1.1. Set Target Device and Synthesis Tool

Analysis (^Triggers Update Diagram)

Set Target Device and Synthesis Tool for HDL code generation

Input Parameters

Target workflow: FPGA-in-the-Loop

Target platform: Xilinx Virtex-5 XUPV5-LX110T development board

Synthesis tool: Xilinx ISE

Family: Virtex5 Device: xc5vlx110t

Package: ff1136 Speed: -1

Project folder: ktop/radiolab_simulink_new/Acquisition_June/SerialSerach/hdl_Correlation

☐ Set Target Library (for floating-point synthesis support)

Result: ✓ Passed

Passed Set Target Device and Synthesis Tool.

Figure 8.8: HDL workflow adviser

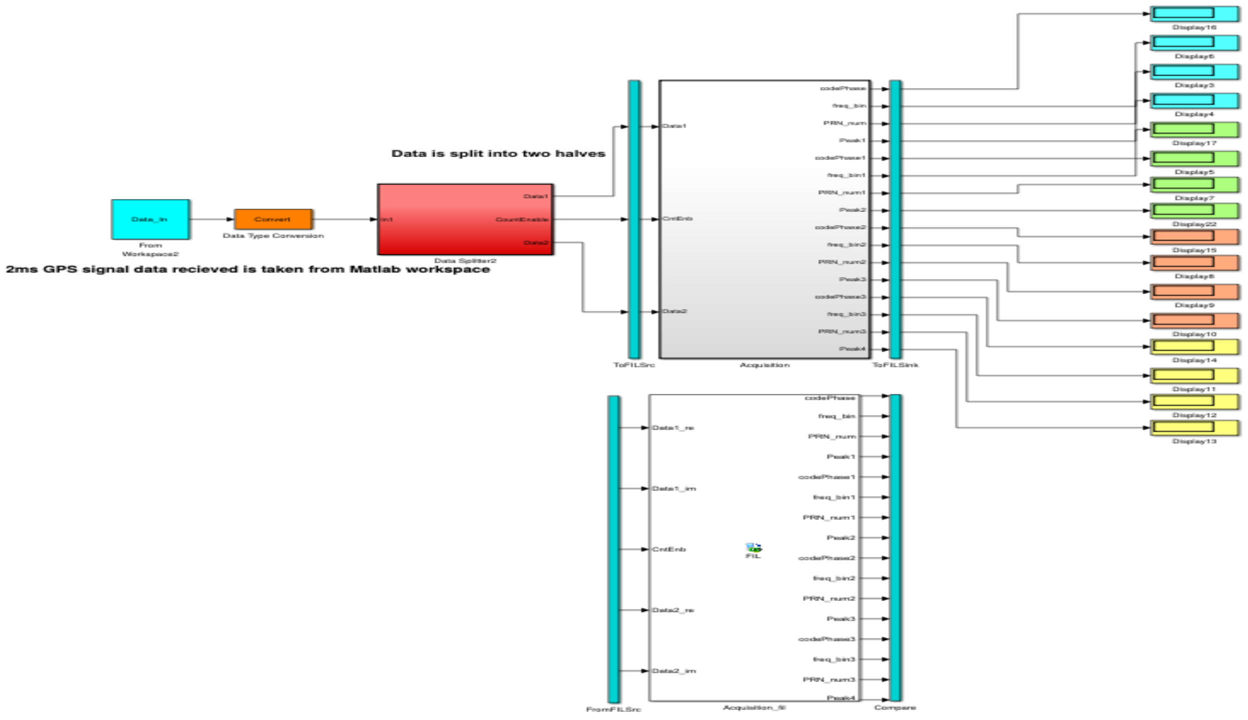
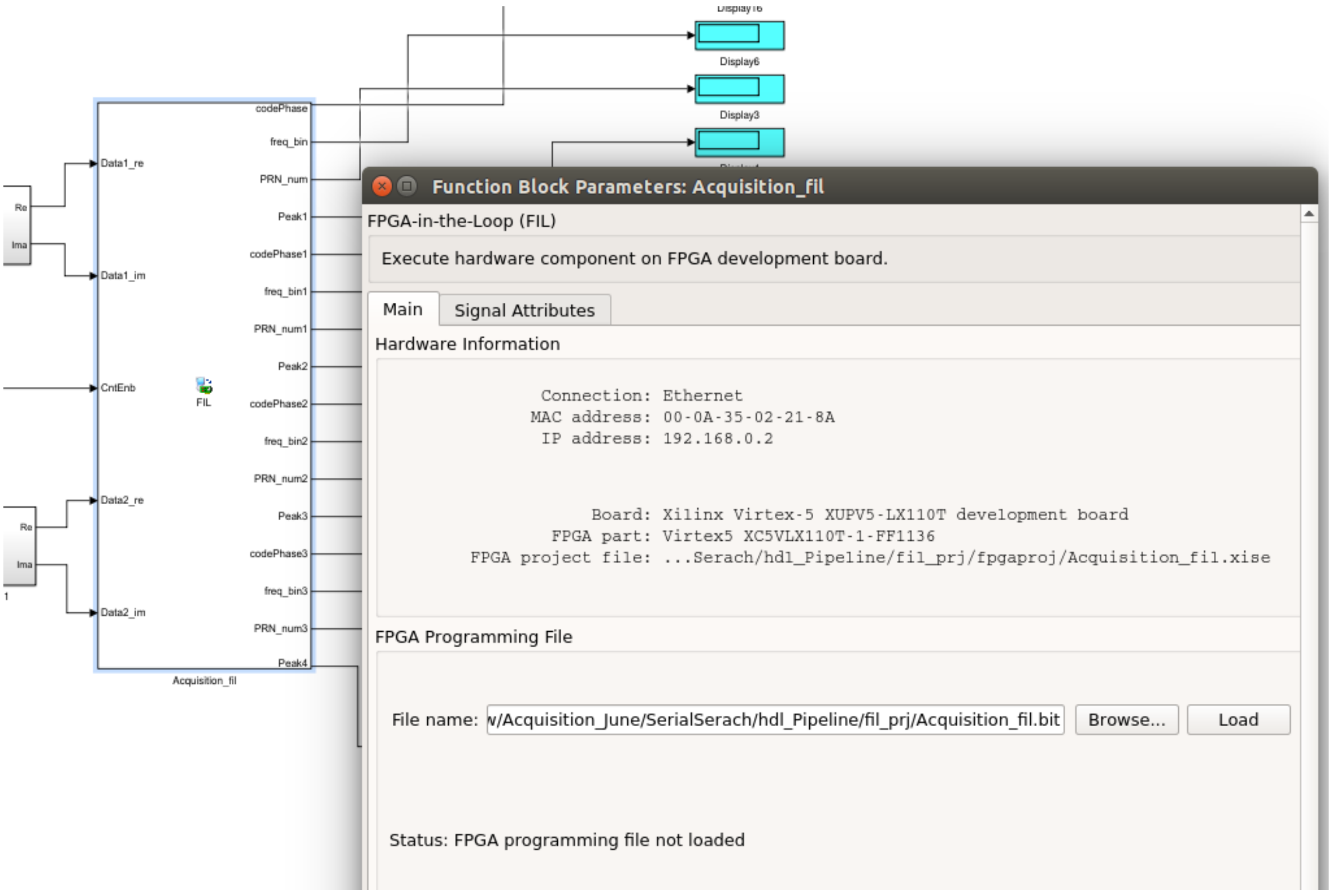


Figure 8.9: HDL cosimulation using FIL configuration by HDL Verifer



Function Block Parameters: Acquisition_fil

FPGA-in-the-Loop (FIL)

Execute hardware component on FPGA development board.

Hardware Information

```
          Connection: Ethernet
         MAC address: 00-0A-35-02-21-8A
          IP address: 192.168.0.2

               Board: Xilinx Virtex-5 XUPV5-LX110T development board
           FPGA part: Virtex5 XC5VLX110T-1-FF1136
   FPGA project file: ...Serach/hdl_Pipeline/fil_prj/fpgaproj/Acquisition_fil.xise
```

FPGA Programming File

File name: w/Acquisition_June/SerialSerach/hdl_Pipeline/fil_prj/Acquisition_fil.bit

Status: FPGA programming file not loaded

Figure 8.10: Programming the FPGA by loading the bit file generated via JTAG concetion

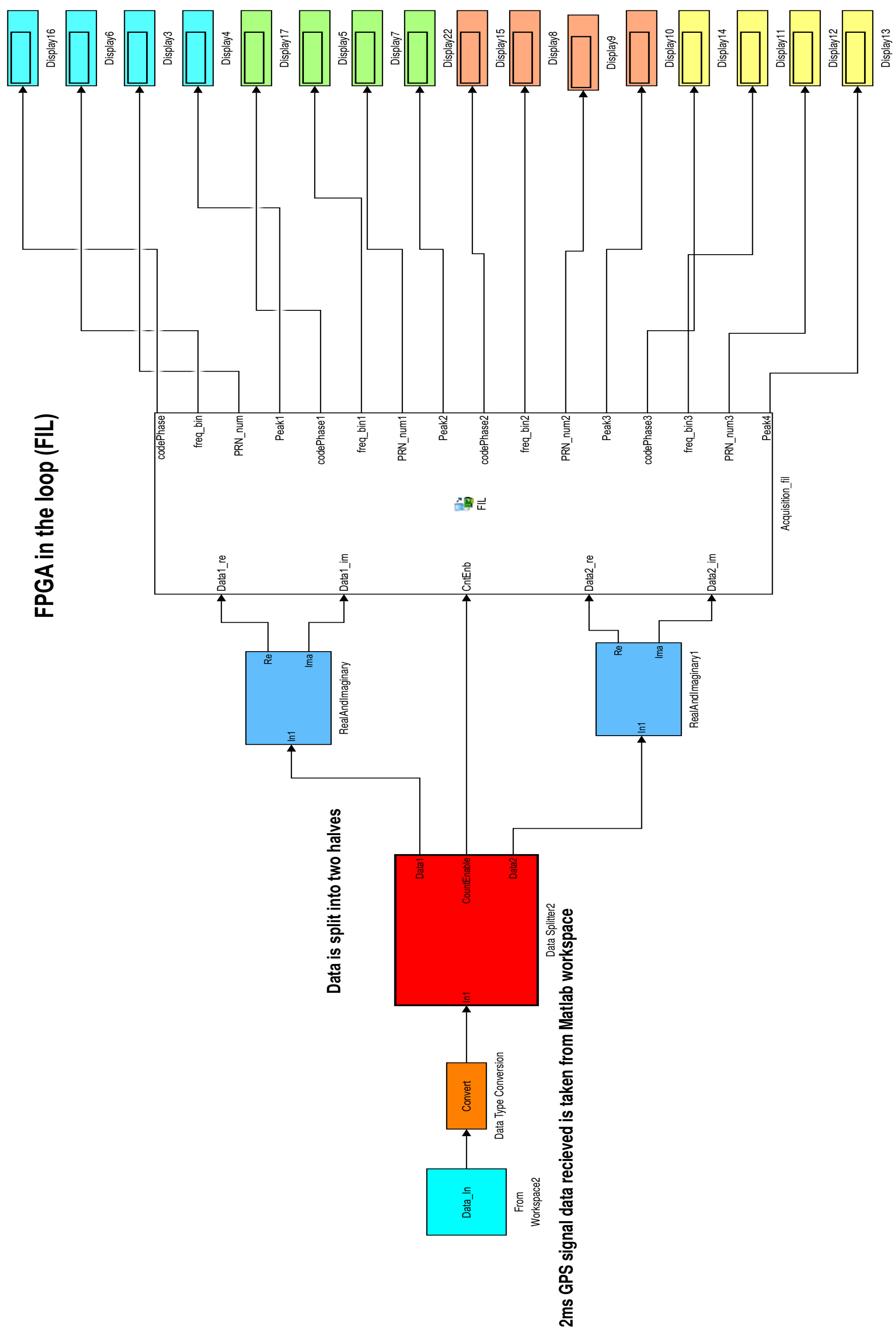


Figure 8.11: Communication between host PC and FPGA in FIL

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 Conclusion

The FPGA prototype of the acquisition algorithm from Simulink model using HDL coder was tested and verified over Xilinx Virtex-5 XC5V-LX110T FPGA evaluation platform. The verification was done by FPGA in the Loop cosimulation model using HDL Verifier. Mathwork's HDL coder is an HLS tool for rapid prototyping of Xilinx and Altera FPGAs from Simulink and MATLAB models. The FPGA design flow starts from algorithms and the SDR implementation of real time GPS L1 receiver was used as a reference to developing the behavioural model of GPS receiver in Simulink by using functional blocks from HDl Coder library alone. The behavioural model was implemented as floating point and was tested for correctness. The floating point model was then converted to fixed point using the Fixed Point Designer. The accuracy of fixed point model was verified using simulations and comparing the result with that of the floating point model.

Synthesis constraints for the FPGA design are not set while developing the initial design. Synthesis constraints depend on the resource limitations of the target FPGA device and clock speed requirements. To make a design that works at high clock rate and fits into the target FPGA, It is required to carry out the timing and area optimisation over the initial design. Timing optimisation is done by distributed pipelining, which is the process of introducing delays appropriately in between the blocks in the critical path. Area optimisation is mainly done by word length minimisation and resource sharing. The traceability reports of high level resource utilisation and critical path information generated by HDL coder during HDL code generation is used as a reference for timing and area optimisation.

The HDL code is generated after the model is converted to fixed point with the help of HDL Workflow Adviser. HDL Workflow Adviser automatically performs a series of steps and generate the HDL code, validation model, high level resource utilization report, critical path estimation , traceability report and optimization report etc. Finally logic synthesis is carried out and the bit file is generated at the end of it. All of them can be accomplished very quickly depending upon the size of the design with the help of HDL Workflow Adviser. Verification of the generated HDL code can be done by HDL cosimulation model or FPGA in the Loop co-simulation model with the help of HDL Verifier.

In this project, FPGA in the Loop configuration is used and it involves interfacing Simulink with the FPGA board and running a simulation in loop by programming the FPGA with the generated HDL code. The verification process is accurate but time consuming due to the delays incurred by interfacing of OS and I/O devices.

9.2 Future Works

In order to complete the implementation of GPS receiver in an FPGA, tracking and navigation data decoding algorithms are also to be implemented along with Acquisition. Tracking algorithm has been implemented by another person in the group, hence the navigation data decoding algorithm has to be implemented in future. Putting all of these three algorithms and generating HDl code will be challenging as far as fitting it into a target FPGA is concerned. It requires a lot of area and timing optimization as the design will be so large in size.

Currently, the algorithm used for acquisition is "Serial Search" and it is very time consuming. So, developing an optimised design for "Parallel Search" will speed up the whole system. But, implementing the parallel search in FPGA requires a lot of resources. Also HDL Optimized FFT block supports only radix-2 algorithm, hence the sampling frequency has to be radix-2(eg:2.048MHz,4096MHz,..). But achieving such a sample frequency using USRP N210 is difficult. Therefore, implementing acquisition in "Parallel Search" algorithm will be challenging but it is essential for speed-up.

The SDR implementation of IRNSS(Indian Regional Navigation Satelliete System) receiver can be done by referring to the algorithms of its GPS counterpart since the basic architecture of both systems are very much similar. Once the SDR of IRNSS is developed, it can be further used a reference to make the Simulink model for HDL code generation.

APPENDIX A

INTRODUCTION TO GNSS-SDR

GNSS-SDR is an open-source GNSS software receiver freely available to the research community. This project provides a common framework for GNSS signal processing which can operate in a variety of computer platforms. This tool is intended to foster collaboration, increase awareness, and reduce development costs in the field of GNSS receiver design and customized use of GNSS signals.

A.1 Installing GNURadio and GNSS-SDR by Pybombs Refer

- Install pybombs (python build overlay managed bundle system)

```
$ git clone git://github.com/pybombs/pybombs
$ cd pybombs
$ sudo ./pybombs install gnuradio
$ sudo ./pybombs install uhd
$ sudo ./pybombs install gnss-sdr
```

- You can also specify the version of gnuradio to install

```
$ git clone git://github.com/pybombs/pybombs
$ cd pybombs
$ git checkout gr-3.6
$ sudo ./pybombs install gnuradio
```

- After the install is finished you can create an environment file (located in $ prefix/setup_env.sh) for your install by running

```
$ ./pybombs env
```

- Source this file (replace $ prefix with the prefix of your recently finished pybombs install) with

```
$ source $prefix/setup_env.sh
```

A.2 Running GNSS-SDR

After install GnuRadio, UHD, GNSS-SDR and other dependencies, you can run GNSS-SDR as follows:

- Check if the USRP is connected to the system by running the following command in the terminal

  ```
  $ uhd_find_devices

  You should get output like this
  ------------------------------------------------------------
  -- UHD Device 0
  ------------------------------------------------------------
  Device Address:
  type: usrp2
  addr: 192.168.10.1
  name: 6
  serial: E8R28M9UP
  ```

- Go to gnss-sdr folder in the pybombs directory

  ```
  $ cd pybombs/src/gnss-sdr/
  ```

- In the `conf` directory, you can find the configuration files for various GNSS satellites and front-end devices. The configuration files contains the settings for parameters like sampling frequency, gain and other settings for processing the signal. For example, *gnss-sdr_GPS_L1_USRP_realtime.conf* file has settings for gnss-sdr to acquire GPS L1 using USRP in realtime. Save the GPS signal received by gnss-sdr automatically by modifying `SignalSource.dump` to true in the conf file.

- Run gnss-sdr by

  ```
   $ cd install
  ./gnss-sdr --config_file = ../conf/gnss-sdr_GPS_L1_USRP_realtime.conf
  ```

- After execution, the GPS signal will be saved in the folder data

- Modify the conf file to print out more information like acquisition,telemetry and pvt data

- To process gps data from a file using gnss-sdr, run

  ```
  $ ./gnss-sdr --config_file = ../conf/mygnss.conf
  ```

.

A.3 Experimenting with GNSS-SDR

- Read `README.md` file in the gnss-sdr directory
- Download a sample signal `2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat` here. The `*.conf` file is outdated, don't use it.
- Download the `*.conf` file from the attachment in this mailing list
- Make the following changes in `2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.conf` file: `SignalSource.filename` to the relative path of the download data
- Datatype of this file is 2byte I/Q interleaved short integer. So size of each sample is 4bytes, 2byte for I + 2byte for Q . (Note if datatype used is gr_complex then we'll have 32bit float I and Q).
- Sampling frequency = 4MHz
- File size = 1.6GB
- recorded time =100sec
- 1.6GB = 100sec x 4MHz x 4 bytes
- Run gnss-sdr by

  ```
  $ cd install
    ./gnss-sdr --config_file = ../conf/
    2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.conf
  ```

- After processing, gnss-sdr outputs data processed at the acquisition, tracking and navigation stage for the user. These outputs can be read using the matlab scripts at `/src/utils` folder in the gnss-sdr directory

APPENDIX B

INTERFACING USRP HARDWWARE WITH MATLAB FOR GPS DATA RECEPTION

The digitized IQ samples of GPS received signal at USRP N210 hardware(which is connected to GPS antenna) can be received by MATLAB as well. The sampling frequency and the center frequency of the USRP need to be set according to the signal specifications. MATLAB has the object *SDRuReceiver* which can configure the USRP and receive data samples from it. MATLAB requires the support package *USRP Radio* to support the aforementioned object and it can be easily installed using the support package installer. The hardware set up required is given in Fig. B.1.

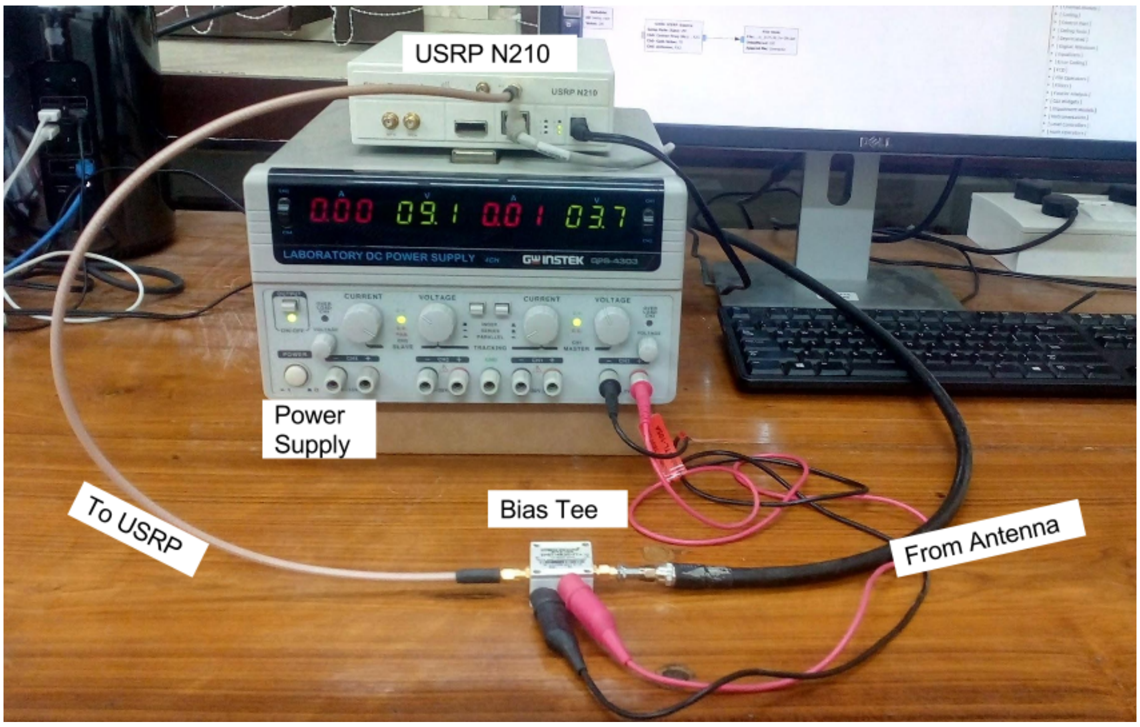


Figure B.1: Hardware set up for GPS signal reception

B.1 Configuring the USRP

In order to communicate with the USRP, an object need to be created in MATLAB using the function *comm.SDruReceiver*. There are many parameters for this object which need to be specified for proper communication with the USRP. In this project, USRP N210 is used and sampling frequency is set as 2 MHz. The center frequency of the received signal is the L1 frequency which is 1575.42 MHz. The following piece of MATLAB code gives the insights about configuring the USRP and taking the data samples to the system.

B.1.1 MATLAB Code

```
global dataOut

dfactor=50;
frame=375000;
time=40; % 40 seconds of data need to be aquired

radio = comm.SDRuReceiver(...
    'Platform',              'N200/N210/USRP2', ...
    'IPAddress',             '192.168.10.1', ...
    'CenterFrequencySource', 'property',...
    'CenterFrequency',       1.57542e9,...          % 380 MHz to 4.42 GHz
    'LocalOscillatorOffset',4000,...
    'Gain',                  38, ...           % 0-38 dB
    'DecimationFactor',      dfactor,...       % sampling frequency of USRP
    'SampleRate',            2e6, ...          % N210 is fixed at 100 MHz.
    'OverrunOutputPort',true,...               % It need to be decimated -
    'FrameLength',           frame,...         % to required value
    'OutputDataType',        'single',...
    'EnableBurstMode',       true,...
    'NumFramesInBurst',      ceil(80*2e6/frame)+6);
```

```
rx_log = dsp.SignalSink;  % step access of object "dsp.SignalSink"
                          % will store the contents of data in a -
                          % buffer. This buffer is of infinite size
                          % and can be accessed as given

%%%%%%%%%%%%%%%%% Initial data acquiring %%%%%%%%%%%%%%%%%%%%

data=step(radio);
step(rx_log, data);

pause(2); % wait for some time to get non zero values
clc;

fprintf('Data acquisition started, wait for 50 seconds ...\n');

%%% Simulation-receive data from usrp and store it in a buffer %%%

count=ceil((100e6/dfactor)*time/frame);
for counter = 1:count
    data=step(radio); % one step execution receives almost 1 ms data
    step(rx_log, data);
end

fprintf('\n\nSaving acquired data ...\n');
output=rx_log.Buffer; % the captured signal 'data' will be stored in
                      % a buffer in the object rx_log Initial values-
                      % in this buffer are transients (1e5 samples).
                      % So skip these much number of samples before -
                      % processing

%%%%%%%%%%%%% Save the acquired Data to a variable %%%%%%%%%%%%%
```

```
skip=1e6;
dataOut=output(skip+1:end).';
dataOut=double(dataOut);

save capture output dataOut data ;

release(radio);
clear radio;
release(rx_log);
clear rx_log;

fprintf('\n\n40 seconds of data has been acquired.\n');
```

USRP is connected to the digital computer using Ethernet cable. The connection to USRP can be checked using the following command.

```
> > >uhd_find_devices
```

You should get output like this

```
– – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –
–– UHD Device 0 – – – – – – – – – – – – – – – – – – – – – – – – –
Device Address:
type: usrp2
addr: 192.168.10.1
name: 6
serial: E8R28M9UP
```

REFERENCES

[1] E. Kaplan and C. Hegarty, *Understanding GPS: principles and applications.* Artech house, 2005.

[2] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. H. Jensen, *A software-defined GPS and Galileo receiver: a single-frequency approach.* Springer Science & Business Media, 2007.

[3] Ivan.G.Petrovski, Toshiaki Tsujii, *Digital Satellite Navigation and Geophysics: A Practical Guide with GNSS Signal Simulator and Receiver Laboratory.* Cambridge University Press, 2012.

[4] U. Madhow, *Fundamentals of digital communication.* Cambridge University Press, 2008.

[5] Rahul.M.Nair, *Software defined radio based implementation of GPS L1 standard positioning service*, Masters thesis, Department of Electrical Engineering, IIT Madras, June 2016.

[6] Adam M. Shapiro, *FPGA-based real-time GPS receiver.* Masters Thesis, Department of Electrical Engineering, Cornell University, January 2010.

[7] Joonas JÃďrviluoma, *Rapid prototyping from algorithm to FPGA prototype.* Masters Thesis, Department of Electrical Engineering, University of Oulu, August 2015.

[8] USRP with MATLAB/Simulink, `https://www.csun.edu/~skatz/katzpage/sdr_project/sdr/uhd_usrp_simulink_doc.pdf`. Using the UHD USRP2 Block in Simulink.

[9] USRP with MATLAB/Simulink, `https://www.csun.edu/~skatz/katzpage/sdr_project/sdr/Instructions_USRP_Simulink.pdf`. Instructions for Using the USRP with MATLAB/Simulink.

[10] MathWorks, `http://in.mathworks.com/products/hdl-coder/`. HDL coder documentation.

[11] MathWorks, `http://in.mathworks.com/products/fixed-point-designer/`. Fixed Point Designer documentation.

[12] Puneet Kumar, *Generating, optimising and verifying HDL code with MATLAB and Simulink*, Documentation, MathWorks, 2012.