FPGA-based data acquisition system for GNSS receiver for LEO-satellites application

Didier Ntibarufata Siboniyo
Master Thesis, Satellite Engineering
Department of Technology
Arctic University of Norway,
Narvik,
Norway,
didisib@outlook.com

June 6, 2017

Abstract

The modern Low Earth Orbit satellites (LEO-satellites) technology is highly complex and reliable than it was yesterday. Nowadays, the LEO-satellites applications extend from civilian to military applications. However, LEO-satellites require a Global Navigation Satellite System (GNSS) service, to achieve various tasks which require positioning and localizing. The challenge is to have a GNSS receiver that is sensitive enough to process weak GNSS signals, as they are buried inside the Doppler frequencies.

Therefore, a specialised data acquisition system for space environment, is utilized in the GNSS receiver to search for visible satellites. However, when it comes to which acquisition algorithm that is suitable for LEO-applications, what matters the most is the Doppler range. That is to say, any algorithm would work, as long as higher Doppler frequency is considered in implementation. But, some algorithms are better and faster than others. On the other hand, the platform in which algorithm is implemented on, does affect the system performance. The FPGA-based data acquisition can only implement a few of the available algorithms. In this project, we focused on serial search algorithms, which seems to be implementable on FPGA.

Preface

"Once we accept our limits, we go beyond them"(Albert Einstein).

I am thankful to my supervisor Dr. Tuan-Vu, and my advisor Tor-Alexander Johansen for their support and guidance through this project.

Nevertheless, I dedicate this thesis to my family.

Contents

Chapter 1

Introduction

1.1 Thesis background

1.1.1 Global navigation satellite system

It is incomprehensible dwelling in a modern world without demanding any navigation technology. Even in the old time, our forefathers had been using one navigation method after another, just to cope with their living. Today, we would certainly associate early methods with irrational technologies or being too primitive, because we are probably using a GNSS which is much faster and precise.

- **GNSS constellation overview**
  The term GNSS refer to a navigation satellite system that has a global or wide coverage on earth. Today, we consider GPS from United States, Galileo from European union, GLONASS from Russia, and BeiDou (COMPASS) from China, as structures within GNSS. However, they all share the same concept. Each GNSS structure can be described based on a specific space constellation of satellites. The constellation, holds informations such as, number of active space vehicles, orbit and the signal structure.

- **GNSS signal characteristics**
  Each satellite transmits a unique ephemeris date, which has low frequency. However, a carrier waves with much high frequency is used to transport this data over enormous distances. In space, carrier wave propagates at nearly the speed of light, and travels through different layers of atmosphere to reach the receiver on or near earth. Unfortunately, these layers reduce the quality of GNSS signals(Gleason & Gebre-Egziabher 2009). Consequently, GNSS signal are weaker, by the time they reach the receiver's antenna. Which means, they need to have a structure that allows the receiver to identify them, otherwise they would be regarded as noise. Therefore, GNSS signal are modulated before leaving the satellite's antenna. It is this modulation which makes it possible to characterise signals according to GNSS's structure. This is because each of the structure has its own preferable method.The GPS, Galileo and Compass use the code division multiple access (CDMA), then the GLONASS utilizes the frequency division multiple access (FDMA) as well as the CDMA. Both CDMA and FDMA belong to what known as direct sequence spread spectrum (DSSS), which is achieved by adding corresponding spreading code in the transmitting line. Normally, ranging code is often used spreading data onto carrier wave. However, the ranging code also has other important functions, such as characterizing a satellite ID, which also avoid interference to happen when all satellite in the constellation are transmitting their ephemeris on the same carrier frequency(Re & Ruggieri 2007). Furthermore, the ranging code is used to measure the transmission time from space vehicle to GNSS receiver.

- **GNSS receiver**
  The GNSS receiver is a device which receives and process the GNSS signals. Such device consists four main functions, as shown on Figure 1.1. The front-end is the first reception function, which down-convert the incoming GNSS signal to the receiver's IF frequency. Additionally, the front-end is

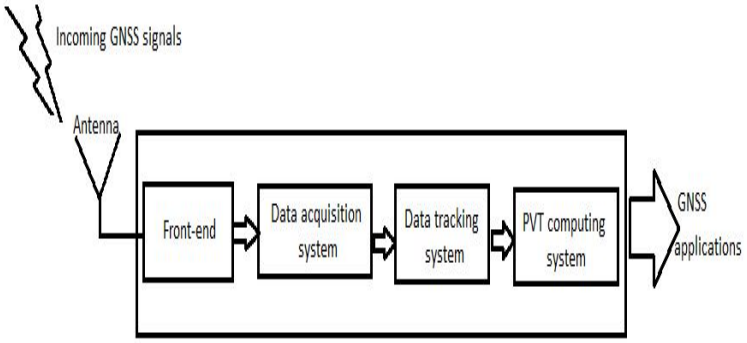

Figure 1.1: General receiver function block

responsible for all digitizing operations, which allows further precessing in the GNSS receiver. The data acquisition searches for available satellites visible to the receiver. The data tracking system removes unnecessary parameter in the navigation data, which allows the computation of PVT. However, navigation data from at least three different satellites to computes the PVT. That is because GNSS utilizes intersection between parabolic curves between multiple references to localize something.This technique is well known as time of arrival (TOA). The more satellites available, the more accuracy positioning becomes.

However, in the GNSS receiver,the DSSS methods as well as the procedures used in the transmission process, becomes advantageous. Only, this time it is used in the reversed order. Starting from the last process, rather than up-conversion, the receiver down convert the GNSS signals first. Then, with help from local signal, the GNSS receiver utilizes correlation techniques to despread energy as well as decodes navigation message.

Furthermore, the spreading code is used to measure the transmission time from space vehicle to GNSS receiver. It requires transmission time from at least three different satellites to compute the PVT. This is because GNSS utilizes intersection between parabolic curves between multiple references to localize something.This technique is well known as time of arrival (TOA). The more satellites available, the more accuracy positioning becomes.

1.1.2 GNSS Signal acquisition system

Each and every GNSS receiver's antenna get hit by many different radio signals,from which genuine GNSS signal is buried. The near transmitter is, the stronger RF is going to be, thus GNSS signals are among the weakest RF at the receiver. Therefore, the first priority in the GNSS receiver is to determine these genuine GNSS signals. To achieve this, the receiver utilizes one of its vital function blocks, the data acquisition system. It acts as a scanner with intention of detecting the GNSS signal. The most crucial operation of any data acquisition system for GNSS receivers, is the correlation.
Constantly, the acquisition performs cross-correlation to lock the correct signal, and auto-correlation to lock the satellite ID in Doppler frequency. This locking is sometimes called aligning of incoming signal to the reference signal, locally generated. If properly aligned, the correlation power becomes high. Furthermore, some detection strategies are utilized to perform statistical comparison from which acknowledgement is made. However, any acquisition type is mathematically defined through algorithms. Then we can split data search algorithm and detection algorithm. Each split has various algorithms that suit for various GNSS receiver's applications. In this project only LEO-satellite application is addressed.

1.1.3 LEO-Satellite overview

Since the launch of Sputnik for nearly 60 years ago, hundreds of satellites have been sent into either deep space or just to orbit the earth. The satellites near earth, can be in orbit the earth in the low earth orbit (LEO), medium earth orbit (MEO), geosynchronous orbit(GEO), highly elliptical orbit (HEO), even other types such as polar orbit, etc. However, in this project, on LEO is addressed. LEO-satellites are low weighted, as a result they maintain the altitudes between 160 kilometres and 2 000 kilometres (Schäfer, et al. 2005). With this altitude, the orbital velocity is about $8Km/s$ which is enough to generate high Doppler frequency (Ali, et al. 1998) with respect to the earth's rotation.
Today, LEO-satellites as well as Nano-satellites applications extend to wider areas both in science, civil and military. That because technology behind them has become more and more complex and better compared to earl ones. However, issues concerning constraints in size and weight still impose lot of challenges as on-board instrumentations must be tiny enough to fit, and must consume very low power to concur the power-source limitation in space.

1.2 Thesis Objectives

As the micro-satellites, increasingly available in the LEO orbit around the earth, satellite technology accelerates in taking over hundreds of applications that were too expensive for few decades ago. However, as stated above, it is incomprehensible to achieve most of these applications without demanding a GNSS receiver. Therefore, we have in this thesis project, a mission of designing an FPGA-based data acquisition for GNSS receiver with respect to LEO-satellite applications. To complete this project, we require to implement and stimulate data acquisition algorithms using Simulink toolbox. It is also required to implement the data acquisition on FPGA.

- **Data acquisition algorithm implementation and simulation using Simulink Toolbox**
  It is easy to understand the behaviour of various algorithms by simulating them. The main goal of this sub task is to develop a model of an acquisition system that function better in the space-born receivers. Since in simulation environment there exists much flexibility and different scenarios can be developed, as well as simulated on various algorithms. Then after all, a suitable design is developed, and carried on to the next process of developing VHDL.

- **Implementation of data acquisition algorithm on FPGA**
  The main goal of this subtask is to implement the design from the previous sub-task, simulation, on the FPGA. If possible, the Simulink model will generate the VHDL code, and the task will be simple. On the other hand, if the model cannot generate the VHDL, then based on the Simulink model, the VHDL shall be developed.

1.3 Previous work

1.3.1 Acquisition algorithms for space-borne receiver

The space technologies have been facing challenges as the space environment demands other standards than on earth. In GNSS applications, the space-borne receiver meets high demands to coincide the weak signals as both receiver and transmitter are in motion. Therefore, many scientific articles, including (Anghileri, et al. 2012) and (Lang, et al. 2016), have presented how weak signals can be acquired. Similarly, for space-borne receiver, the author(s) in (Wang, et al. 2016) has justified that a delay-multiply algorithm suits the LEO-satellites application. However, the drawbacks of this algorithm, is that it can only be implemented on software receivers. Nonetheless, the same author mentioned that by predicting Doppler at the receiver, the system can be designed in such a way that coincides with the effect of motion. Therefore, it seems like any algorithm would work, as long as correct Doppler range is considered.

1.3.2 Acquisition algorithms for optimizing the search time

An acquisition system that requires much time to search for SIS, always results in uneconomical use of receiver's resources. In (Anghileri et al. 2012) and (Leclre, et al. 2013) methods for reducing amount of search number or iteration are described. One technique is to reduce sign transition as described in (Leclre et al. 2013), which reduces number of signal search by 21%, and almost halve the memory resources for implementation. It then concluded by the author in (Patel & Shukla 2011) that the faster the acquisition the lesser power consumption it will require. Making it fast and low cost acquisition method. However, the drawbacks of methods presented in above articles are based on parallel strategy, which utilizes Fourier transformation. So far, our knowledge stretches, the FFT is hard to implement on hardware.

1.3.3 GNSS signal acquisition system on SoC

Theoretically, today there exists countless manner to design and develop a high-tech system. However, as for power saving and fast processing systems design, we may run out of choices. For GNSS receiver in general, technology such as application specified integrated circuit (ASIC), has been used for decades to reduce both processing delay and power consumption. Although ASIC technology has been productive than if discrete components are used, there less flexibility in overall system. Therefore, new platforms have been developed, through different research occasions.

In 1997, the electronic and signal processing laboratory (ESPLAB) of institute of micro-technology (IMT) in collaboration with swatch group research and development (ASULAB) designed the first embedded watch with a GPS receiver. Unfortunately, this watch didn't come to the market(Baracchi-Frei 2010). However, it opened doors for several research projects in the eld of GNSS receivers and implementation technology. Since then, digital signal processing techniques have been used to improve GNSS receivers. As a result, accuracy, short time to the market and more flexibility became obvious and standards in receiver design. Eventually, software defined radio (SDR) became more popular than ASIC. This platform is fast to implement GNSS receiver with, and since it is based on code, it is reusable. The main issue with the SDR is that it uses a central processor for every computation processes.

This is still not the optimal way to reduce the energy consumption and speeding up the processor. Luckily, the system-on-chip (SoC) technology provides a multi-core solution, whereby computation process can have its own processor. In this way, the overall system can shut-down or activate specific processor. Hence, a new trend is about to take over, the field programmable gateway array (FPGA). It is a SoC which can be developed as an SDR yet implement as an ASIC. The fact that it is constituted with many cores, each capable of computing complex task, and that the signal routing intersects these cores, makes the FPGA capable of accessing data in a very short amount of time as well as saving more energy.

1.3.4 FPGA based GNSS signal acquisition

In (Leclère, et al. 2013), "the comparison framework of FPGA-based GNSS signal acquisition architecture" has been presented. However, this comparison does not cover many of algorithms, but it outlines the major aspects in GNSS signal acquisition. Therefore, drawbacks of algorithms in section 1.3.2, are said to be resolved by utilizing the high cost FPGA, that has enough DSP module, of which FFT can be implemented. The implementation of parallel search acquisition on FPGA is also described in (Malik, et al. 2009), however the results in this paper are based on Matlab simulation not the real life implementation.

1.4 Thesis outline

1.4.1 Delimitation

Figure 1.2 reveals the scope of this project. The thesis objectives refer to a GPS receiver for LEO-satellite application, whereas data acquisition system is the topic. The theory covers both detection strategy and acquisition strategies, but the hardware implementation only emphasises on GNSS signal acquisition.

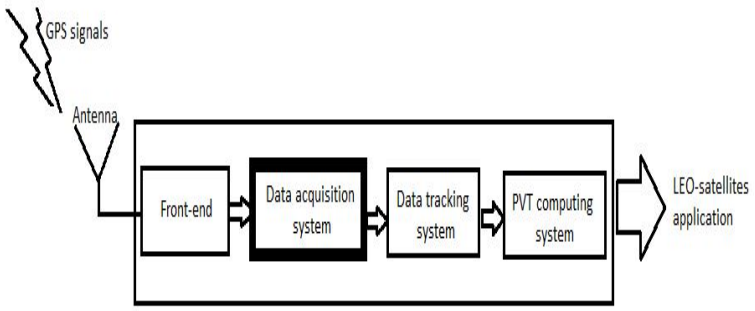

Figure 1.2: The scope of this thesis is the data acquisition system in the GPS receiver

1.4.2 Report outline

Hereafter, this report consists of 5 chapters. In the next chapter, two, necessary theories on how a GNSS receiver works, are covered. This chapter covers topics such as GNSS signal characteristics, general GNSS receiver function block, common GNSS receiver technology and GNSS ranging error.

The third chapter, takes one step further and tackles only about the GNSS signal acquisition system with respect to the LEO-satellite application. However,the main objective of this chapter is to select which algorithm that is suitable for GNSS signal acquisition with respect to LEO-satellite application as well as FPGA implementations. Therefore, detection strategy and search strategy are covered in separate sections, then suitable algorithms on behalf of searching as well as detection is selected.

Based on the result of the third chapter, the Simulink modelling of GNSS signal acquisition is handled in the forth chapter. The main objective is to understand how the system works, and possibly use the Simulink's HDL coder to generate the VHDL to be used in the fifth chapter.

The firth chapter handles necessary procedures to develop VHDL code and synthesizes it on FPGA.

At last, the conclusion remarks are presented in the sixth chapter, where the major trade-off of this thesis is summarized, and possible future work is recommended.

However, the appendix also holds important informations, such as results, codes, etc.

Chapter 2

GPS receiver concept

2.1 GPS signal characteristics

In the GPS constellation there exist 32 satellites. Each satellite broadcasts a unique navigation data on a L carrier band, L1 (1575.42 MHz) or L2 (1227.60 MHz). However, to make each satellite's data unique, a specific ranging code modulates the data. Consequently, each GPS signal consists of ranging code, data and carrier frequency.

2.1.1 Ranging code

The concept of ranging code was invented back in 1967, when Robert Gold attempted to identify each satellite. Thus, ranging code is also known as gold code. Basically, the gold code is a unique set of binary sequence, which is widely used in wireless communication system for identification purposes. In the GPS concept, the gold code is used to identify each satellite in the constellation. That is to say, each satellite transmits a unique radio frequency, which also enables the multiple access transmission to happen. Nevertheless, the use of gold code is somehow similar to the encryption of data, where data is hidden within. Which demands a longer gold code than data. However, for gold code, bit-streams are known chips, which defines a data-less bit-stream. Therefore, adding these chips to the data bits, the result is nothing but noise filed signal. That the reason why ranging code is again known as pseudo-random noise (PRN). In general, there exist many structures for PRN, but for GPS application, there are not so many. The coarse acquisition(C/A) and the precision(P) are commonly used in civilian applications, whereas M-structure is only used for military application.

C/A PRN structure

The most common PRN is the C/A with a sequence of 1023 chips every millisecond. The structure of C/A is generated by the Modulo-2 addition of two polynomials, $G1$ and $G2$.

$$G1 = 1 + z^3 + z^{10} \tag{2.1}$$

and

$$G2 = 1 + z^2 + z^3 + z^6 + z^8 + z^9 + z^{10} \tag{2.2}$$

Where 2.1 and 2.2 indicate the input sequence of $G1$ and $G2$ respectively. The operator + is the same as the operator $\oplus$ on table 2.1, which denote an XOR logical function.

Each polynomial describes input to a 10-bit linear transfer feed-back registers (LTFSR). Which means, the first LTFSR have bits 3 and 10 of its sequence, as the input. Regarding the second LFSR, polynomial $G2$ demands bits 2,3,6,8,9 and 10 of its sequence, to be fed back. In the first LFSR the out put is the last bit, bit 10. Yet, for the second LFSR the output depends on code-phase which corresponds to a satellite ID, as defined on Table 2.1. Then output of the first LFSR is XORed with the second LFSR out to obtain a

Table 2.1: Code phase for C/A and P. The check chips are provided in octal(base 8)

SV ID No.	Code phase selection		Code delay Chips		First 10 C/A Chips	First 12 P Chips
	C/A($G2_i$)	($X2_i$)	CA	P		
1	$2 \oplus 6$	1	5	1	1440	4444
2	$3 \oplus 7$	2	6	2	1620	4000
3	$4 \oplus 8$	3	7	3	1710	4222
4	$5 \oplus 9$	4	8	4	1744	4333
5	$1 \oplus 9$	5	17	5	1133	4377
6	$2 \oplus 10$	6	18	6	1455	4355
7	$1 \oplus 8$	7	139	7	1131	4344
8	$2 \oplus 9$	8	140	8	1454	4340
9	$3 \oplus 10$	9	141	9	1626	4342
10	$2 \oplus 3$	10	251	10	1504	4343
11	$3 \oplus 4$	11	252	11	1642	...
12	$5 \oplus 6$	12	254	12	1750	...
13	$6 \oplus 7$	13	255	13	1764	...
14	$7 \oplus 8$	14	256	214	1772	...
15	$8 \oplus 9$	15	257	15	1775	...
16	$9 \oplus 10$	16	258	16	1776	...
17	$1 \oplus 4$	17	469	17	1156	...
18	$2 \oplus 5$	18	470	18	1467	...
19	$3 \oplus 6$	19	471	19	1633	...
20	$4 \oplus 7$	20	472	20	1715	...
21	$5 \oplus 8$	21	473	21	1746	...
22	$6 \oplus 9$	22	474	22	1763	...
23	$1 \oplus 3$	23	509	23	1063	...
24	$4 \oplus 6$	24	512	24	1706	...
25	$5 \oplus 7$	25	513	25	1743	...
26	$6 \oplus 8$	26	514	26	1761	...
27	$7 \oplus 9$	27	515	27	1770	...
28	$8 \oplus 10$	28	516	28	1774	...
29	$1 \oplus 6$	29	859	29	1127	...
30	$2 \oplus 7$	30	860	30	1453	...
31	$3 \oplus 8$	31	861	31	1625	...
32	$4 \oplus 9$	32	862	32	1712	4343

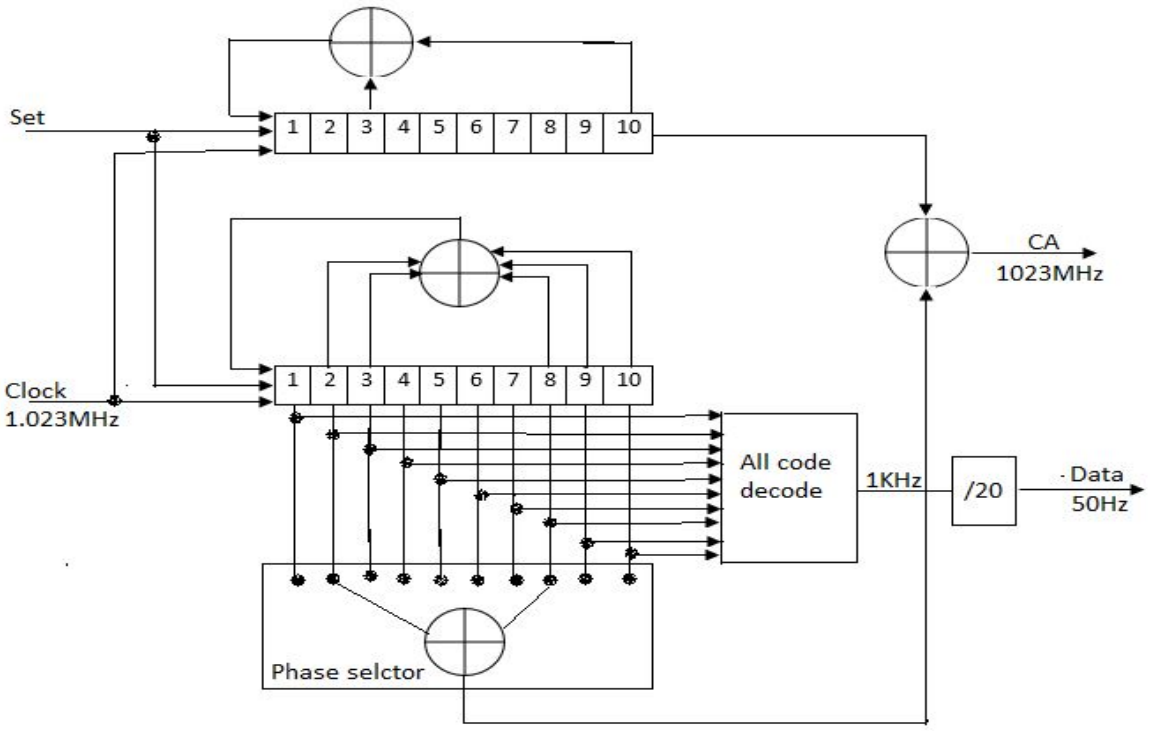

Figure 2.1: C/A code generator

chip of C/A. Figure 2.1 shows a CA generator that runs on 1.023MHz. Which means, each 1 ms, 1023 chips is generated. To check if the sequence is correct, the first 10 chip as described on column 6 of Table2.1, can be used.

P-Code generation

The PRN P-code is a raging code $P_i(t)$ of 7 days in length at the chip rate of 10.23 Mbps. each $P_i(t)$ pattens is generated by the Modulo-2 sum of two sub-sequences denoted as $X1$ and $X2_i$ with length of 15,345,000 chips and 15,345,037 chips, respectively. According to (IS-GPS-200H), $X1$ is also generated in Modulo-2 sum of two sequences (X1A and X1B), with their lengths 4092 and 4093 chips respectively. The $X1$ epoch is generated for each X1A's 3750 counters, allowing the epoch to occur every 1.5 seconds after X1 patten have been generated. X1A and X1B have input to their shift register defined as

$$X1A = 1 + X^6 + X^8 + X^1 1 + X^1 2 \tag{2.3}$$

$$X1B = 1 + X^1 + X^2 + X^5 + X^8 + X^9 + X^{10} + X^{11} X^{12} \tag{2.4}$$

2.3 and 2.4 are initialized to code vector [001001001000] and [010101010100], respectively.

$$X2A = 1 + X^1 + X^3 X^4 + X^5 + +X^7 + X^8 + X^9 + X^{10} + X^{11} X^{12} \tag{2.5}$$

$$X2B = 1 + X^2 + X^3 + X^4 + X^8 + X^9 + X^1 2 \tag{2.6}$$

The $X2_i$ sequences are generated by first producing an X2 sequence and then delaying it by a selected integer number of chips, i, ranging from 1 to 37. Each of the X2i sequences is then modulo-2 added to the X1 sequence thereby producing up to 37 unique P(t) sequences. The X2A and X2B shift registers, used to

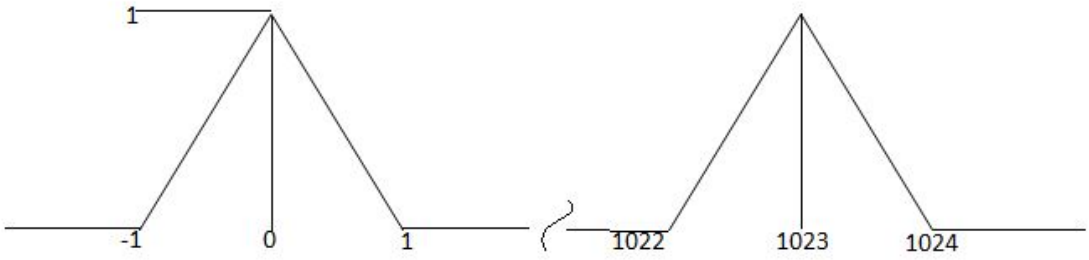

Figure 2.2: Normalized C/A

generate X2, operates in a similar manner to the X1A and X1B shift registers. More about generation of P-code, is found in the technical documentation "IS-GPS-200H".

2.1.2 Navigation data

The navigation message is the central element within the GPS signal. Without it the system would be pointless, as well as impossible to navigate by means of satellite. The message is generated at $50bit/sec$, in the navigational payload, and consists 25 frames. Each sub-frame contains specific informations which are vital for GPS receiver to decode the almanac data, and extract navigation data. It is said in section 2.1.1 that data is added to the ranging code, but how? The method utilizes the modulo 2 addition of data and ranging code. Lets assume the C/A code is used, then the resulting addition is shown on figure 2.3. The C/A generates 1023 chips each millisecond. Yet, a 50 Hz data has a 20 millisecond pulse width. Therefore, for each 20 milliseconds, there are $1023 * 20 = 20460chips$. Since XOR is used to relate data and C/A, thus the resulting addition looks nothing but the invert of original C/A.

2.1.3 BPSK modulated L signal

As mentioned in the introductory section in 2.1, the carrier signal is used to transfer navigation message. Hence, it has much higher frequencies than the actual message. Figure 2.4 shows one of methodologies used for transmitting GPS signals. The modulo 2 sum in figure 2.3 multiplies a carrier wave, pure sinusoid. Having the sum on NRZ format causes the phase of carrier signal to change whenever data sequence changes levels. Hence this method is known as binary phase shift keying (BPSK).

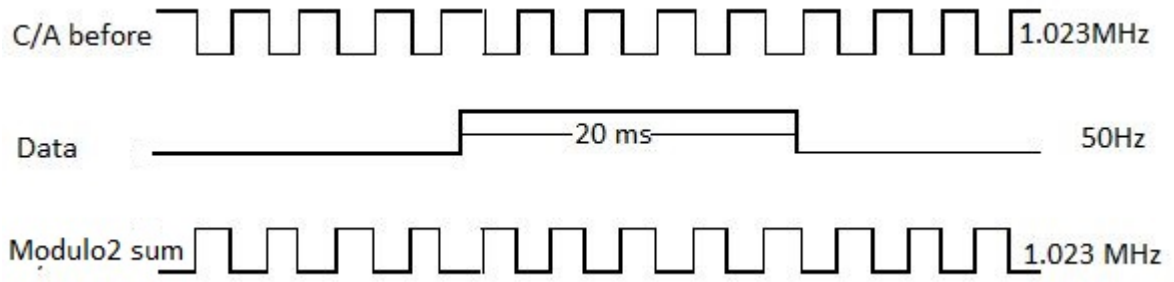

Figure 2.3: Modulo 2 addition of data and C/A

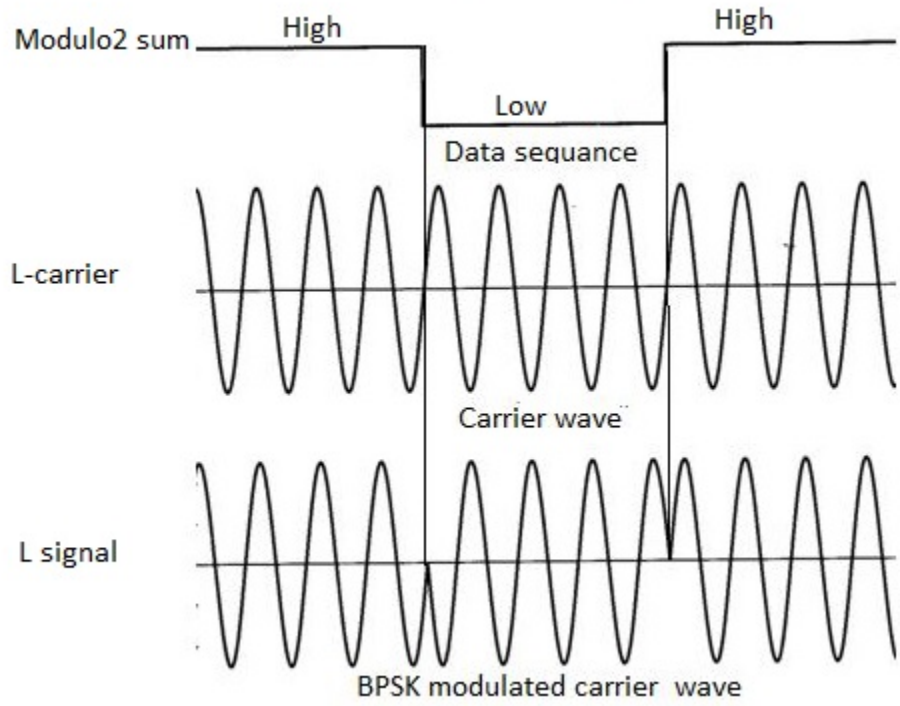

Figure 2.4: Mapping of data and C/A on carrier wave

The complete GPS signal consists of navigation message, raging code and frequency band itself. So that the L signal leaving the antenna of the k^{th} satellite is given as

$$s^k(t) = \sqrt{2P_c}(C^k(t) \oplus D^k t)cos(2\pi f_{L1}t) + \sqrt{2P_pL1}(P^k(t) \oplus D^k t)sin(2\pi f_{L1}t) + \sqrt{2P_PL2}(P^k(t) \oplus D^k t)sin(2\pi f_{L2}t). \quad (2.7)$$

The symbol '$\oplus$' denotes the logical operation "exclusive or". With P_c,P_PL1 and P_PL2 are the powers of signs with C/A or P code. C^k and P^k are the C/A and P(Y) code sequences assigned to satellite number k. D^k is the navigation data sequence. $f1$ and $f2$ are the carrier frequencies of L1 and L2. However,the satellite transmits data on either L1 or L2 at time.Therefore, if only L1 is considered then equation 2.7 becomes:

$$s^k(t) = \sqrt{2P_c}(C^k(t) \oplus D^k t)cos(2\pi f_{L1}t) + \sqrt{2P_pL1}(P^k(t) \oplus D^k t)sin(2\pi f_{L1}t) \quad (2.8)$$

Regardless of where the receiver is, the original signal gathers some noises on its transmission path. So that at the receiver,the [2.8] is mixture of noises and carrier offset. Therefore, at the receiver L1 signal is given as

$$S^k(t) = const. \times sqrt2PD^k(t-\tau)cos(2\pi(f-\Delta)(t-\tau)+\theta) \quad (2.9)$$

With$P$ being the total received power τ being the transmission delay, Δf is the Doppler and θ is the received phase. Mark that the P in equation 2.9 is the common interpretation for power which includes C/A, P(Y) and noises. Whereas the values of P are presented in signal levels, which indicate the strength level of the signal. According to Lei Dong(Dong 2005), the minimum received power is at $-160.0dBW$, $-163.0dBW$ for C/A and P respectively. As for the maximum power, the C/A get $-152.0dBW$ and the for P it is equal to $-155.5dBW$. However, in electronic systems it is commonly to present the performance in terms of carrier-to-noise CNR and signal-to-noise SNR. The CNR defines the raw carrier power to raw noise power in the transmission line, while SNR includes all noise and power measurement from end-to-end. The SNR usually signifies the signal quality seen by the user-end, and is performed mostly on baseband signals.

$$SNR = \frac{S}{N_0 B_n} \quad (2.10)$$

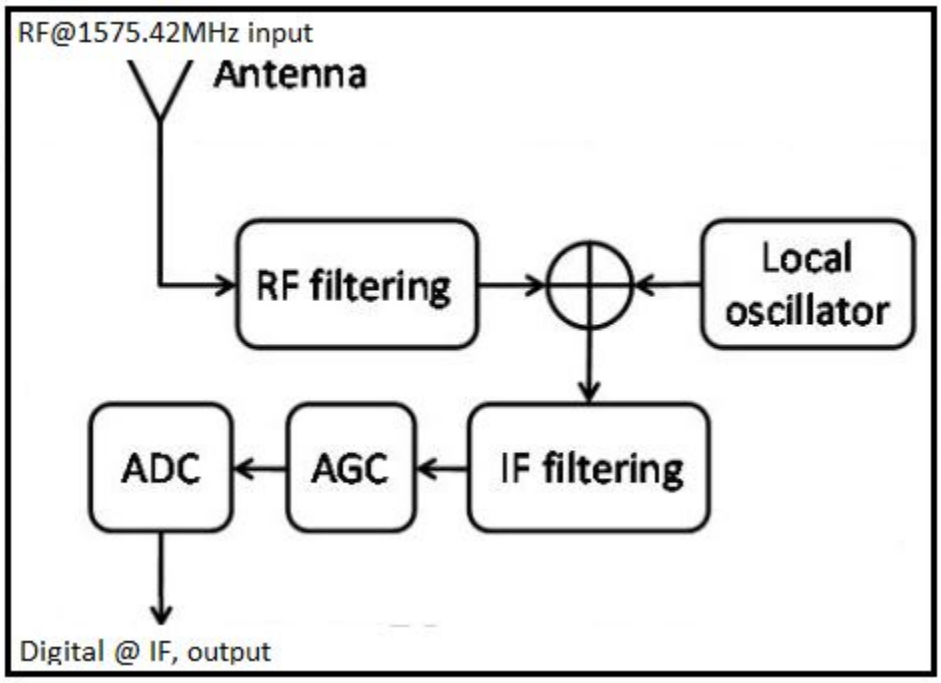

Figure 2.5: General front-end architecture (Originally, by Marco Rao)

Where B_n is the band width of the receiver. S, N denote signal power and noise power within the B_n. The noise is assumed to be a one-sided white noise, and N_0 is the spectral density, usually defined as $N_0 = KT$. According to (Dong 2005), the ratio $C/N_0[dBHz]$ is the most representation of signal power, due to its independence of receiver's bandwidth. Whereas the relationship between the SNR and C/N_0 can be represented as

$$SNR(dB) = \frac{C}{N_0}[dBHz] - B_n(dB) \tag{2.11}$$

with C being the total received signal power.

2.2 GPS receiver's function

2.2.1 The RF front-end system

The first step in the receiving process happens in the front-end section. The main purpose of the front-end is to transform the high RF signal into a necessary format. The format may differ from receiver to receiver, but it must be presented digitally. Basically, the carrier frequency in equation[2.9] is an RF signal of up to 1575.42MHz. However, it is not wise to process signal in such wide bandwidth. Secondly, 1575.42MHz is too much for the most of electronic components to handle. Therefore, incoming signal is down-converted to the receivers IF. Even so, modern digital GPS receivers utilize narrow front-end bandwidth to improve performance (Curran, et al. 2010). That imposes various RF components to achieve a perfect format.The antennas, amplifiers, filters, attenuators and mixers are some of the common. Unfortunately, most of RF component have non-linear characteristics in terms of gains, noise figure and third-order intercept point(Miskiewicz, et al. 2009). As a result, each of the involved component in the front-end does affect the receiver's performance factors, such as sensitivity, dynamic range, etc(Akos & Tsui 1996). Figure 2.5 shows a general front-end architecture.

- **Antennas**
  Basically, the antenna is the first instrument of the receiver, it has the main function of capturing

signals is space of all satellites in the constellation. However, the incoming signals hit the antenna from different directions, which demands a circular polarized antenna. Usually, a GPS receiver's antenna has lower noise amplifier (LNA), initial filter and band limiting (Dong 2005) integrated in one component. Therefore signal after the antenna stage obtains much more manageable bandwidth(Miskiewicz et al. 2009), allowing mixers to proceed the down-conversion.

- **RF mixer**
  Although much of high frequency is attenuated within the antenna stage, but the signal band must be narrower and centre the receivers intermediate frequency (IF). Practically, this is known as down-converting, which is performed by mixing the local oscillator (LO) with the incoming signal. Equations (2.12) through (2.14) describes the involving procedure.

$$S_{IF}(t) + nIF(t) = (S(t) + n(t)) * 2cos(\omega_{Lo}t) + Harmonics + LOfeedthrough + imagenoise \quad (2.12)$$

$$S_{IF}(t) = AC(t)D(t)cos(\omega_{L1} + \omega_{L0} + \Delta\omega)t + \phi_0 + cos(\omega_{L1} + \omega_{L0} + \Delta\omega)t + \phi_0 \quad (2.13)$$

$$n_{IF}(t) = r(t)cos(\omega_{IF}t + \varphi(t)) \quad (2.14)$$

  Where

  - A is the signal amplitude,
  - $SIF(t)$ is the intermediate frequency signal,
  - $nIF(t)$ is the noise after down conversion,
  - $\Delta\omega$ is the frequency offset (ex: Doppler),
  - ϕ_0 is the nominal carrier phase, note that it is also ambiguous.
  - $\omega L0$ is the frequency of local oscillator,
  - ωIF is the intermediate frequency, and
  - $\varphi(t)$ is the noise phase.

- **Filter**
  As the carrier frequency on equation (2.9) passes through antenna stage and especially down-conversion, the frequency limit is now within preferable band. Unfortunately, extra component emerge as consequence of mixer characteristics. In equations 2.12 and 2.13, we can see clearly how original signal catches extra signal. Therefore, the main purpose of this stage filter is to remove these extra signals(Dong 2005).

- **Analogue to digital conversation**
  To complete the signal formatting, the ADC is deployed to both digitize and present the signal on proper format. Basically, quantization methods are applied, whereas a 1 to 5 bit resolution quantizer is used. Practically, 5 is low resolution, and thus introduces a loss of approximately 0.5 dB. To overcome the loss an automatic gain control (AGC) is used. As a result, the overall digital signal is as good as if the higher bit resolution quantizer has been used.

Discritizing the C/A L1 signal

Lets represent equation 2.9 this way

$$r_{RF}(t) = \sum_{i=1}^{L} y_{RF,i}(t) + \eta_{RF}(t) \quad (2.15)$$

Where $\eta_{RF}(t)$ is equivalent to overall noise. The term $y_{RF,i}(t)$ denotes useful L signal, that takes the following structure

$$y_{RF,i}(t) = A_{iCi}(t - \tau^a_{i,0})d_i(t - \tau^a_{i,0})cos[2\pi(f_{RF} + f^i_{d,0})t + \phi_{i,0}] \quad (2.16)$$

where

- A_i being an amplitude of the i^{th} useful signal,
- $\tau^a_{i,0}$ is the delay caused by the communication channel,
- $f^i_{d,0}$) is the Doppler frequency affecting the i^{th} useful signal,
- $\phi_{i,0}$ being a random phase,
- f_{RF} is the carrier frequency which depends on the GPS and on band to be analysed.
- $C_i(t)$ is the spreading sequence assumed to have values in the set $\{-1, 1\}$,
- $d_i(t)$ is the navigation message.

Then the equation 2.15 can be written as:

$$r(t) = \sum_{i=1}^{L} A_{i\tilde{C}i}(t - \tau^a_{i,0})d_i(t - \tau^a_{i,0})cos[2\pi(f_{IF} + f^i_{d,0})t + \phi_{i,0}] + \eta(t) \tag{2.17}$$

With the following condition

$$\tilde{C}_i(t) \approx C_i(t), \tag{2.18}$$

the term $\tilde{C}_i(t - \tau^a_{i,0})$ represents the resulting spreading sequence within filtering process in the receiver's front-end. Signal is then digitized with sampling time T_s. Making Eq 2.15 to become

$$r(nT_s) = \sum_{i=1}^{L} y_i(nT_s) + \eta(nT_s) \tag{2.19}$$

which result in the following version of Eq 2.17 :

$$r(t) = \sum_{i=1}^{L} A_{i\tilde{C}i}(nT_s - \tau^a_{i,0})d_i(nT_s - \tau^a_{i,0})cos[2\pi(f_{IF} + f^i_{d,0})nT_s + \phi_{i,0}] + \eta(nT_s) \tag{2.20}$$

That is the signal model of which impact of both quantization and filter is neglected, meaning that characteristic of the signal is still the same, only on digital format. However, the signal must be represented on discrete domain. That is, a sequence with sampling frequency $f_s = 1/T_s$, denoted as $x[n] = x(T_s)$. Henceforth, $r[n] = r(nT_s) = \sum_{i=1}^{L} y_i[n] + \eta[n]$, then signal model in Eq 2.20 can be rewritten as:

$$r[n] = \sum_{i=1}^{L} A_{i\tilde{C}i}[n - \tau^a_{i,0}/T_s]d_i[n - \tau^a_{i,0}/T_s]cos[2\pi(f_{IF} + f^i_{d,0})nT_s + \phi_{i,0}] + \eta[n] \tag{2.21}$$

which is represented on simpler form as

$$r[n] = \sum_{i=1}^{L} A_{i\tilde{C}i}[n - \tau_{i,0}]d_i[n - \tau_{i,0}]cos[2\pi F^i_{D,0}n + \phi_{i,0}] + \eta[n] \tag{2.22}$$

with $F^i_{D,0} = (f_{IF} + f^i_{d,0})nT_s$ and $\tau_{i,0} = \tau^a_{i,0}/T_s$

2.2.2 The data acquisition system

The second function of the GPS receiver is to search for visible satellites to the receiver, which is performed in the data acquisition. The input of this subsystem is the signal 2.22, that is after the front-end. Notice that this signal propagates with the receivers IF, and still contains all important informations which are unique for any available satellite. The terms $A_{i\tilde{C}i}$ and $F^i_{D,0}$ in the equation 2.22, are essential element for a satellite to be acquired. In fact, the system locally generate a copy of these element from which a rough estimation of Doppler frequency and determination of code phase is made. However, the concept

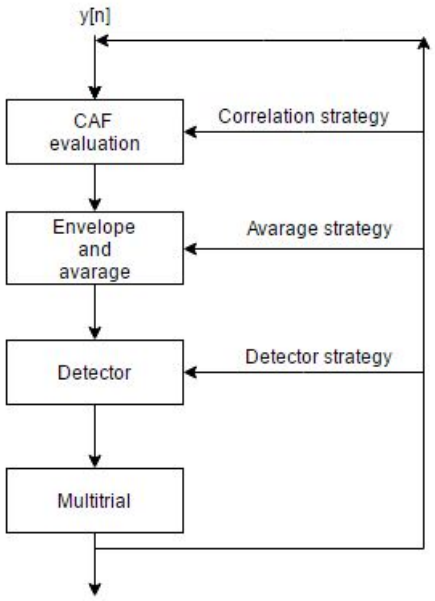

Figure 2.6: General functional blocks of an acquisition system

of acquisition is strongly based on correlation between the incoming signal and locally generated signals. practically, for each satellite ID to acquire, the acquisition system locally generates ranging code replica of the ID. Additionally, the local oscillator with exact F_D as the receiver, generates the in-phase and quadrature signal. Basically, the search is performed in the $\pm fD$, where the satellite ID is said to be acquired only if the ranging code in the incoming signal is properly aligned with the locally generated. Theoretically, the acquisition strategy is done through serial search, where the system steps through every combination . The parallel strategy, where either ranging code or LO is parallelized, optimizes the search. Both strategies are further described in 3.1. However, the framework of the acquisition is shown on Figure 2.6, where each block performs a unique logical operation(Re & Ruggieri 2007).

1. **CAF evaluation**
   All the acquisition systems are based on evaluation and the processing of the cross-ambiguity function (CAF)(Re & Ruggieri 2007). It is at this stage that correlation strategy is deployed, which exhibit a sharp peak in correspondence of code delay and Doppler frequency, as shown on figure 2.7. The approach of evaluating the CAF depends on acquisition strategy. In serial case, multiplication is used, where the local ranging code is multiplied in both I and Q. Each LO component generates a search vector of $\pm fD$ with adequate steps. On the other hand, when parallel strategy is used, then Fourier transformation allows the CAF to be partially or fully performed in frequency domain. Then FFT algorithms are used to achieve correlation. Which is shown in 3.1.2 and 3.1.3.

2. **Envelope and average**
   However, to compute the correlation power from CAF stage, the average strategy is used. In serial, the average strategy is performed by the utilization of integrator and dump, such that allows the acquisition system to evaluate only chip rate (1023 chips) at a time. Then, the result is further processed into a square function. Therefore, if any peak available, will be clearer, otherwise noises will have relatively low amplitudes.

3. **Detector**
   Which amplitude is reasonably to determine whether the average result is the correct signal or noise? The detector compares the derived peak value to threshold. So, the result after comparison determine the presence or the absence of a satellite. However, in both cases a false alarm can happen. As a result, probability based algorithm are rather used to increase the performance of the detector, that

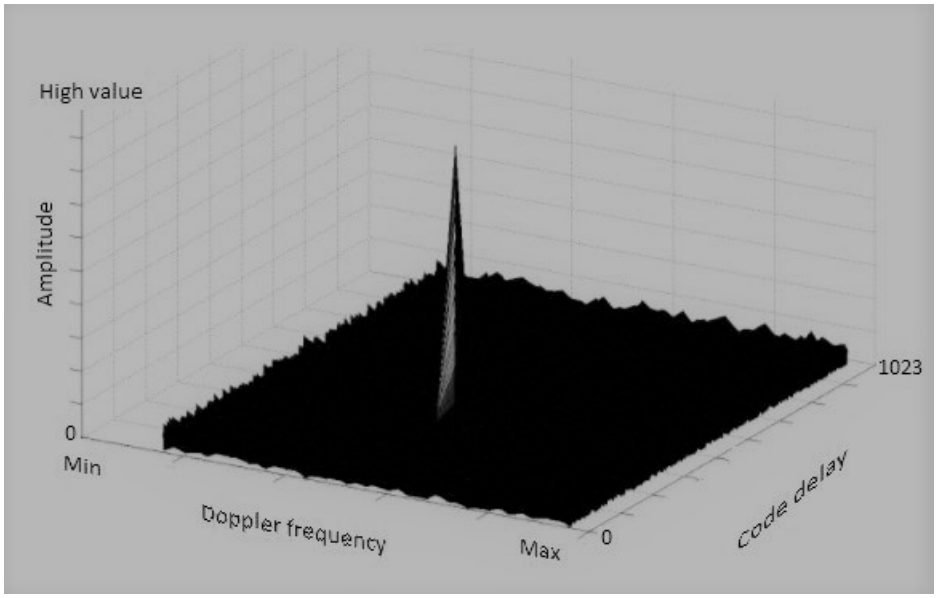

Figure 2.7: General acquisition plot. The peak location is related to code phase and Doppler frequency

is discussed in 3.2.

2.2.3 The data tracking system

The data tracking system has the main task of computing the precise ranging code, measuring the right carrier phase and demodulating the navigation data of the acquired satellite. However, the system is based on signal lock concept, of which carrier Doppler and code offset are tracked. A typical tracking process includes loop filters, integration & dump, discriminator, numerical, discriminator, control oscillator, as well as code generator (Dong 2005).On Figure2.8, it is shown in one of the architectures (Tsui 2005) that implement the above elements.

1. **The delay locked loop (DLL)** utilizes a local code generator that generates three raging code replicas, early(E), prompt(P) and late(L). Then it uses collator techniques from which I_L, Q_L,I_E and Q_E are obtained. Further computation includes discriminator to estimate the $\Delta\tau$ and to match the code phase(Dong 2005).

2. **The phase locked loop (PLL)** has the main function of tracking the phase. Basically, the frequency of LO has to match the frequency of input signal. Hence, NCO is acquired to easily adjust the frequency of both in-phase(I) and quadrature(Q) signal. Then correlation between the LO and prompt code is applied. The resulting parameters I_P and Q_P enters an arctangent discriminator to detect phase mismatch as well as to generate a control signal to the NCO. However, a loop filter is required to reduce noise and determine necessary parameters such as bandwidth, damping coefficient, loop order as well as original frequency(Tsui 2005).

After both DLL and PLL are locked the GPS signal is despread and converted to baseband. Allowing the receiver to recognize the navigation data bits that the in-phase prompt exhibits(Pini, et al. 2012).

2.2.4 PVT computation

After the k navigation data messages are available, the receiver estimates the distances ρk of all k satellites. To compute the position, the velocity and the time, k must be greater than 3. Figure 2.9 (Pini et al. 2012) shows how the receiver with coordinate x_0, y_0, z_0, with respect to the satellites positions in equation 2.23, can be localized.

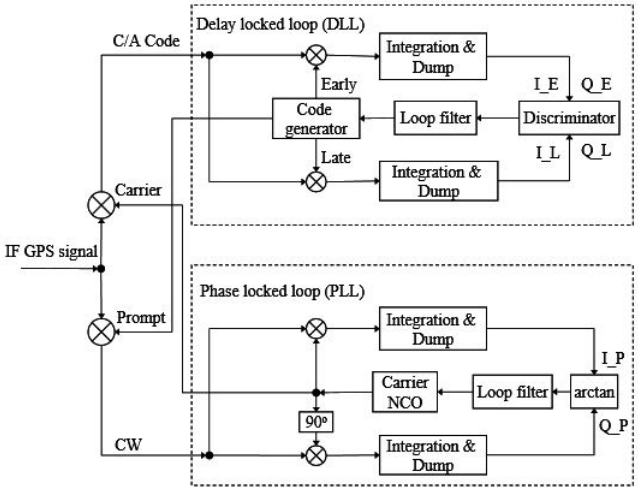

Figure 2.8: General tracking system

$$\rho = \begin{cases} \sqrt{(x_1 - x_u)^2 + (y_1 - y_u)^2 + (z_1 - z_u)^2} = \rho_1 + c.\Delta b \\ \sqrt{(x_2 - x_u)^2 + (y_2 - y_u)^2 + (z_2 - z_u)^2} = \rho_1 + c.\Delta b + c.\Delta_2 \\ \sqrt{(x_3 - x_u)^2 + (y_3 - y_u)^2 + (z_3 - z_u)^2} = \rho_1 + c.\Delta b + c.\Delta_2 \\ \sqrt{(x_4 - x_u)^2 + (y_4 - y_u)^2 + (z_4 - z_u)^2} = \rho_1 + c.\Delta b + c.\Delta_2 \end{cases} \tag{2.23}$$

Where Δb represents the clock bias between the one on board of the satellite and the receiver's, ρ_1 is the reference pseudo-range and c is the speed of light. There are many ways to go from equation 2.9 to get PVT values. However, the common as well as the simplest algorithm is based on least-squares method (Pini et al. 2012), where the strategy of approximation utilizes the linear regression on the direct cosine matrix (DCM) of the ECEF vector towards each satellite.

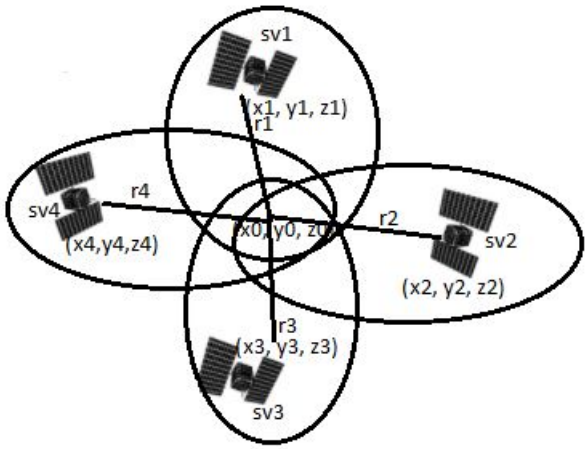

Figure 2.9: General satellite trilatteration

2.3 Common GPS receiver's technology

2.3.1 Hardware-defined

The conversion receiver design method is based on ASIC (Application Specific Integrated Circuit) technology. Technically, this means utilization of dedicated hardware components. Consequently, the architecture is fixed and the system obtains no flexibility (Won, et al. 2006). To modify functionalities or performance, either the ASIC must be re-fabricated or the circuit must be physically rebuild. Since the ASIC is designed with a highly specialized purpose, the hardware receiver may perform faster which can improve the power usage. However some of the drawbacks regarding the less flexibility in the hardware receiver are:

- There is less upgrading opportunity to the newest technology.
- The prototype and development takes longer time as complexity increases.
- The production cost is relative high as a result of time and different types of hardware components used.
- Less effective in terms of reliability and accuracy as a result of noises from discrete components and as their performance degrades over time.
- The power usage may increase as a number of hardware increases, and as degradation takes place.

2.3.2 Software-defined(SDR)

Unlike the ASIC-based design, the Software-based performs all digital processing through general purpose processor (GPP). The motive behind its existence is to overcome the classic limitation due to the ASIC flexibility (Hein, et al. 2006). The main approach is to allocate the GPP close to the antenna as possible, by employing DSP techniques. Such approach results in decrease in size, weight, and minimizes the power consumption as well as product cost. According to (Bayendang 2015), software-defined platform is developed through Matlab, Simulink or the combination of both. As a result the SDR-receiver offer high flexibility in design as various programming languages can be utilized. However, using the GPP as a central unit for all signal processing, minimizes the performance of the receiver.

2.3.3 System on chip (SoC)

A firmware-defined receiver is based on system-on-chip (SoC) engineering, however the design and implementation follows the same procedure as the software-defined. As a result, developers can easily understand both platforms, as the structure remains the same. The bottom line is that the firmware-defined receiver converts the design into embedded language. Such language is then used to implement a SOC such as field programmable array(FPGA).The features of FPGA allows more complex design to be verified, and can also transfer ASIC later, depending on the application. With more than 150 hundreds logical cells, each containing numerous logic gates, the FPGA can load several soft-core processors. The FPGA- based receiver uses this technique, to implement several small processor each designed to perform a specified task. On top of increase in performance, the FPGA-based receivers are more flexible, and easy to modify as they can be considered as software configurable device. There are many ways to develop a firmware-defined receiver. One of the methods is through VHDL coding, however such manner may require several man hours since there are many algorithms to implement. The simplest and fastest is by Simulink utilization.

2.4 GPS ranging errors

As human and technological equipment are imperfect, the possibilities for having errors is always going to happen. On top of that, the signal path itself consists of many error sources within atmospheric layers. In GPS applications the term ranging errors defines all the sources that contribute to error implantation onto

end-to-end channel. Commonly, raging errors are categorized according to where or how they find place in the system. Whereas six categories ephemeris data, satellite clock, ionosphere, troposphere, multipath and receiver are the major. Somehow, each of the six categories, even affects the PVT computation. Lei Dong in (Dong 2005) have modelled some which have direct impact on GPS receiver. Then in (Yang, et al. 2011), how to approach the atmospheric error mitigation is shown.

Chapter 3

GPS signal acquisition for LEO satellite applications

3.1 Acquisition algorithms

3.1.1 Serial search acquisition

One of the most used algorithm in the CDMA systems as well as GPS data acquisition is the serial search(Leclère, et al. 2014). It is one of the classical acquisition method which allows the receiver to search within whole Doppler frequency range. Additionally, it is based on simple configuration techniques, which also makes it popular in GPS receiver design. In the serial search algorithm, the correlators are based on simple mathematics such as accumulators also called integrator and dumper, square function known as envelop. Additionally, the system has some control logic that both determine the threshold and switches the states between acquired, not-acquired and tracking. Figure 3.1, shows an architecture of the serial search for each satellite id.

At the beginning, the local oscillator generates two components, in-phase (I) and quadrature (Q). These component have same FD which is the $(IF + fD_{max})fs$, but they are 90° phase-shifted to each other. Additionally, C/A is also locally generated. Therefore, serial correlation refer to the local signals and compare similarities with incoming signal. The incoming GPS signal is first multiplied by I and Q, the aim is to match the Doppler frequency. Next, the local generated C/A in correspondence of a known satellite ID, is multiplied in both component. This is to match the code delay, which also spreads signal energy over a space of 1023 chips. Therefore correlation energy is obtained by summing up all the spread components. The correlation energy is high if incoming GPS signal matches the local Doppler and code delay. Then absolute value of the correlation energy displays a sharp peak. The serial search searches over all possible combinations within $IF \pm fD_{max}$, in steps of 500 Hz. If we assume fD_{max} equal 10KHz, then the receiver has to search among $1023(2\frac{10KHz}{500Hz} + 1) = 1023 * 41 = 41943 combinations$, which is too much (Boto 2014). One of the common solution to reduce the combinations is to rather use the parallel search strategy.

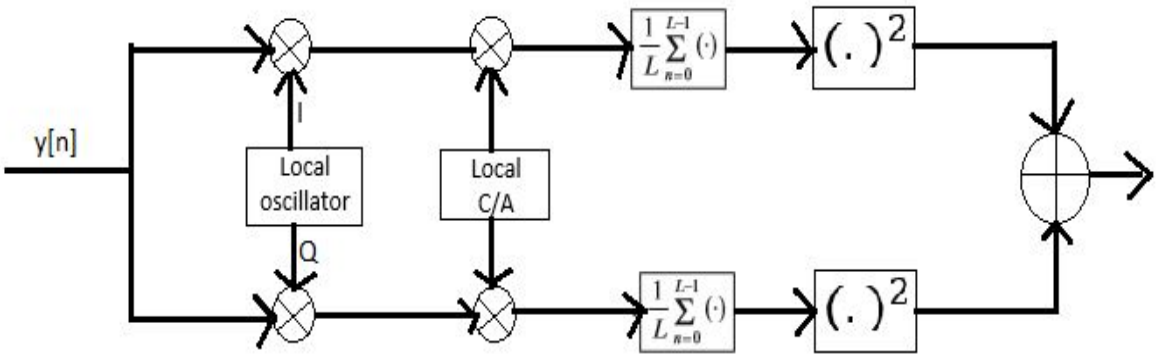

Figure 3.1: Serial acquisition

3.1.2 Parallel frequency space search

One of the method to reduce the search number is the parallel frequency space search algorithm. The key technique is the Fourier transformation which parallelizes the search for one parameter either the Doppler frequency or the code phase (Boto 2014). The locally generated PRN is multiplied by the incoming signal, then Fourier transform transforms the product from time domain into frequency domain, which is shown on figure 3.2. After the transformation the signal becomes continues once again. If the PRN in the incoming signal is well aligned with the local PRN the transformed wave is perfectly a sinusoid, just like on figure 3.3. On the other hand there will be some distortions in the resulting wave. Furthermore, by taking the magnitude of the Fourier transformation results in peak like signals with corresponding index. Each index represent each frequency of the carrier wave signal in frequency domain. So that if a peak in correspondence of the satellite ID is detected, so is the frequency.

However, the accuracy of determining the frequency or performance of the system, depends on the length of the Fourier transformation and the sampling frequency.

$$\Delta f = \frac{fs}{N} \tag{3.1}$$

Equation 3.1 shows the relationship between sampling frequency, number of data and resolution frequency. Usually, by utilizing this algorithm, the resolution frequency is lower compared to that of serial acquisition. That is correct because the parallel search intentionally skips some steps.

3.1.3 Parallel code phase search

Akin to the previous method, the parallel code phase search is based on the parallel strategy that parallelizes one or both parameter in order to reduce search combinations. In this particular algorithm, the code phase

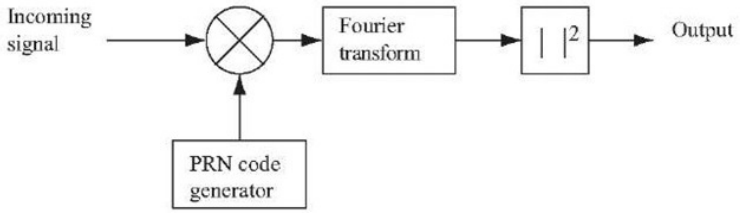

Figure 3.2: Parallel frequency space search

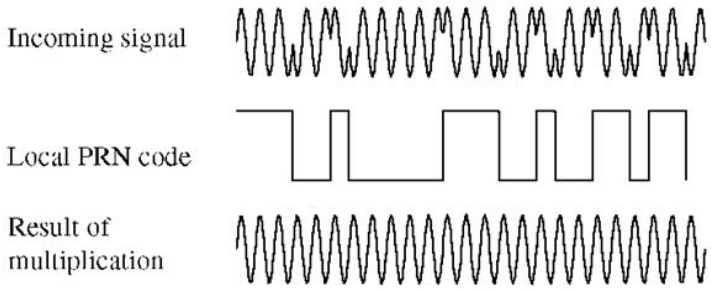

Figure 3.3: The result after multiplication in frequency domain

dimension is parallelized, allowing the system to only perform the search in frequency steps. The key element in this algorithm is the cross correlation which is strongly based on the Fourier transformation(Boto 2014). Just to demonstrate how the cross correlation between two finite sequences works, lets define the two sequences x(n) and y(n) in discrete domain as

$$x(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}} \tag{3.2}$$

and

$$y(k) = \sum_{n=0}^{N-1} y(n) e^{\frac{-j2\pi kn}{N}} \tag{3.3}$$

Then their cross correlation is given as

$$z(n) = \frac{1}{N} \sum_{n=0}^{N-1} x(-m) y(m-n) \tag{3.4}$$

Which can be extended to the fully expression as

$$z(k) = \sum_{n=0}^{N-1} x(m) e^{\frac{-j2\pi kn}{N}} \sum_{n=0}^{N-1} y(m+n) e^{\frac{-j2\pi k(m+n)}{N}} = X^*(k) Y(k) \tag{3.5}$$

Note that both sequences have same length and also that the X^* denote the conjugate operation. So Figure 3.4 shows how it works. At the entry of the system, incoming signal is being multiplied by the I and Q, and then the product is Fourier transformed. Furthermore, the resulting sequence is multiplied by the complex conjugate of the PRN. Mark that the Fourier transformation is first applied to the PRN, before it is complex conjugated. Next, the resulting product is transformed from frequency to the time domain, and finally the enveloping operation is applied onto it. This operation exhibit peak signal in correspondence of the PRN. This algorithm saves much energy consumption, especially when parameters for both FFT and IFT are thoroughly calculated.

3.1.4 Delay-multiply algorithm

The delay-multiply searches the code phase bin one time and does not search for every Doppler frequency. The first procedure is to get rid of the intermediate frequency. This is achieved by multiplying the incoming signal by its delayed version. Then with the help of the correlation, the satellite can be identified through evaluation of the code phase. At the same time, the FFT operation is deployed to evaluate the frequency shift. Figure 3.6 shows the architecture of delay-multiply acquisition, whereas the details about how it works are given in (Wang et al. 2016). However, in (Boto 2014) the similar algorithm is presented in simple architecture as shown on figure 3.5, as well as the process of removing the Doppler frequency in equations 3.6 to 3.8.

$$s_n(t) = s_{incoming}(t) \times s^* delayed(t-\tau) \tag{3.6}$$

$$= A(t)A^*(t-\tau)exp(j2\pi f_D t)exp(-j2\pi f_D(t-\tau)) \tag{3.7}$$

$$= A(t)A^*(t-\tau)exp(j2\pi f_D \tau) \simeq A(t)A^*(t-\tau) \times 1, if \tau \simeq 0 \tag{3.8}$$

According to authors in (Wang et al. 2016) the delay-multiply search method is a seldom used algorithm in ground GPS receivers simply because the Doppler frequency on ground it not high enough. Which would rather result in sensitivity loss due to the several stage multiplications. However this algorithm is number one priority for space-born receiver.

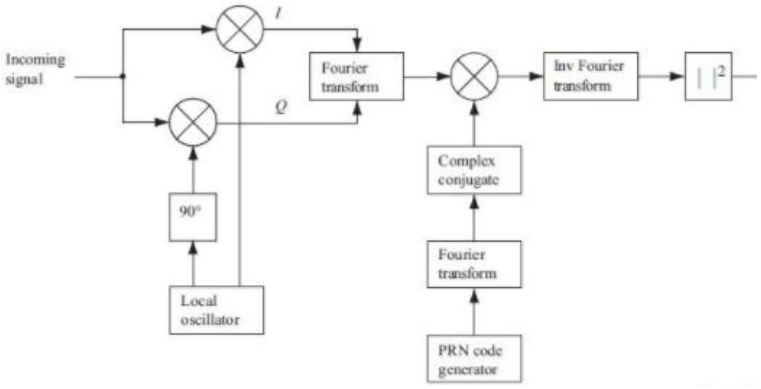

Figure 3.4: Parallel code phase search algorithm

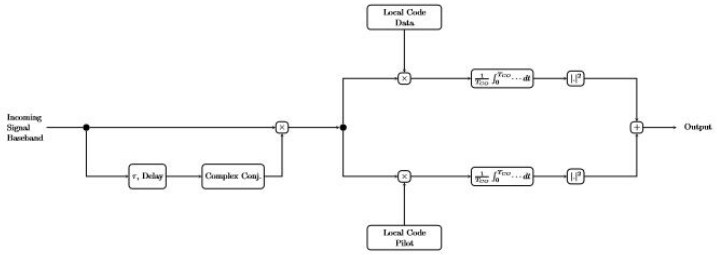

Figure 3.5: Delay and multiply simple architecture version

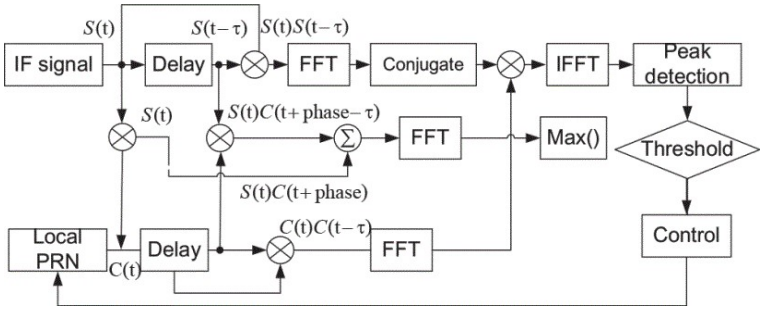

Figure 3.6: The architecture of delay-multiply acquisition

3.2 Detection algorithms

Usually when dealing with GPS receiver, decision making on whether the acquired satellite is true or false comes hand in hand with acquisition concept. Technically, the receiver combines its ability of sampling and the result from the correlator, to form a statistic based alarm. The mechanisms in a such alarm are theoretically defined through some detector algorithms. Three algorithms, coherent detection, non-coherent detection and differentially coherent detection, often appear in many literatures. These algorithms are presented in (Re & Ruggieri 2007) as being depending on how the envelop and averaging is designed. That is, whether it is the envelop which is performed first or the averaging. However, the statistic representation is shown in (BE 2010).

3.2.1 Coherent combining detector

Let the parameter estimate, $\hat{\theta}$ be given as$\{\hat{\zeta}, \hat{omega}_D\}$, then the correlator output be represented as

$$D(\hat{\theta}) = \left| \sum_{m=0}^{M_c N_S - 1} r(mTs) exp(-j\hat{\omega}_D mT_s) c(mT_s - \hat{\zeta} T_{chip}) \right|^2 \tag{3.9}$$

If we assume an input signal having additive white Gaussian noise with the power spectral density(PSD) defined as $N_0/2$ and the variance equal to σ_n^2, then the variance of the complex input $\sigma^2 = 0.5\sigma_n^2$. So that the decision statistic can be represented as distributed random variable with two degrees of freedom and the variance equal to $\sigma_Y^2 = M_c N_s \sigma^2$, which can also be given as

$$\sigma_Y^2 = \frac{M_c N_s B_{if} N_0}{2} \tag{3.10}$$

Equation 3.10 refers to the correlation result, which can give either a false alarm or true alarm. For both cases, two hypotheses, satellite being present H_1 or satellite being absent H_0 can happen. Therefore, we can have probability function of the decision variable for the first hypothesis given as

$$fH_0(x) = \begin{cases} \frac{1}{2\sigma_Y^2} exp(-\frac{x}{2\sigma_Y^2}), & if x \geq 0 \\ 0, & if x < 0 \end{cases} \tag{3.11}$$

When satellite and noise are both present in the correlator result the decision statistic function is no longer central. In this case the second hypothesis H_1 is presented as

$$fH_1(x, \lambda) = \begin{cases} \frac{1}{2\sigma_Y^2} exp\left(-\frac{x+\lambda}{2\sigma_Y^2}\right) I_0\left(\frac{\sqrt{x\lambda}}{\sigma_Y^2}\right), & if x \geq 0 \\ 0, & if x < 0 \end{cases} \tag{3.12}$$

The $I_\nu(.)$ is the Bessel function of order k, defined as

$$I_\nu(z) = \left(\frac{1}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{(\frac{1}{2})^{2k}}{k!\Gamma(\nu + k + 1)} \tag{3.13}$$

Where $\Gamma(.)$ is the Gamma function defined as

$$\Gamma(z) = \int_0^\infty t^{z-1} exp(-t) dt \tag{3.14}$$

That is for real part $\Re\{z\} > 0$. The λ in equation 3.10 denotes the non-centrality of the decision probability, and it is defined in terms of correlator output as

$$\lambda = \left| E\left[\sum_{m=0}^{M_c N_s - 1} r(mTs) exp(-j\hat{\omega}_D mTs) c(mTs - \hat{\zeta} T_{chip}) \right] \right|^2 \tag{3.15}$$

If we ignore the noise contribution, then the above equation can shrink down to

$$\lambda = \left| \sum_{m=0}^{M_c N_s - 1} s(mTs) exp(-j\hat{\omega}_D mTs) c(mTs - \hat{\zeta} T_{chip}) \right|^2 \tag{3.16}$$

Which again can be shrink down to

$$\lambda = \left| M_c N_c \frac{A}{2} \right|^2 = 0.5 (M_c N_S)^2 C \tag{3.17}$$

Only if $\hat{\omega}_D = \omega$ and $\hat{\zeta} = \zeta$.

Therefore, with the coherent detector algorithm, the probability for detection P_d, and of the false alarm P_{fa} can be presented as

$$P_d = \int_{Th}^{\infty} fH_1(x, \lambda) dx = Q1 \left(\frac{\sqrt{\lambda}}{\sigma_Y}, \frac{\sqrt{Th}}{\sigma_Y} \right), \tag{3.18}$$

$$P_{fa} = \int_{Th}^{\infty} fH_0(x) dx = exp\left(-\frac{Th}{2\sigma_Y^2} \right) \tag{3.19}$$

With Th being the threshold constant to compare against. And $Q_k(a, b)$ being a Marcum function defined as

$$Q_k(a, b) = \int_b^{\infty} x \left(\frac{x}{a} \right)^{k-1} exp\left(-\frac{(x^2 + a^2)}{2} \right) I_{k-1}(ax) dx \tag{3.20}$$

3.2.2 Non-coherent detector

The non-coherent estimator utilizes an extra integrator that sums the $D(\hat{\theta})$ of the coherent, by a number K. So, in this case the decision statistic takes this format

$$D(\hat{\theta}) = \sum_{k=0}^{K-1} \left| \sum_{m=kM_c N_s}^{(k+1)M_c N_S - 1} r(mTs) exp(-j\hat{\omega}_D mT_s) c(mT_s - \hat{\zeta} T_{chip}) \right|^2 \tag{3.21}$$

This will have an impact on the rest of the detection parameter. The new dwelling time becomes

$$t_t = K M_c T_{code} \tag{3.22}$$

Then the distribution of decision statistic will have variance equal to σ_Y^2 and $2K$ degrees of freedom, whereby the probability density function of H_0 and H_1 get an additional element due to non-coherent integration.

$$fH_0(x) = \begin{cases} \frac{1}{2\sigma_Y^2} \frac{1}{\Gamma(K)} \left(\frac{x}{2\sigma_Y^2} \right)^{K-1} exp(-\frac{x}{2\sigma_Y^2}), & if x \geq 0 \\ 0, & if x < 0 \end{cases} \tag{3.23}$$

$$fH_1(x, \lambda) = \begin{cases} \frac{1}{2\sigma_Y^2} \left(\frac{x}{\lambda} \right)^{\frac{K-1}{2}} exp(-\frac{x+\lambda}{2\sigma_Y^2}) I_{K-1}(\frac{\sqrt{x\lambda}}{2\sigma_Y^2}), & if x \geq 0 \\ 0, & if x < 0 \end{cases} \tag{3.24}$$

With λ defined as

$$\lambda = \sum_{k=0}^{K-1} \left| E \left[\sum_{m=kM_c N_s}^{(k+1)M_c N_S - 1} r(mTs) exp(-j\hat{\omega}_D mT_s) c(mT_s - \hat{\zeta} T_{chip}) \right] \right|^2 \tag{3.25}$$

Which is equivalent to

$$0.5 K (M_c N s)^2 C \tag{3.26}$$

. Similar to the previous algorithm, the λ is compared to the threshold value, of which detection probability is given as

$$P_d = \int_{Th}^{\infty} fH_1(x,\lambda)dx = Qk\left(\frac{\sqrt{\lambda}}{\sigma_Y}, \frac{\sqrt{Th}}{\sigma_Y}\right), \tag{3.27}$$

and the false alarm probability defined as

$$P_{fa} = \int_{Th}^{\infty} fH_0(x)dx = \frac{\Gamma_k\left(-\frac{Th}{2\sigma_Y^2}\right)}{\Gamma(K)} \tag{3.28}$$

3.2.3 Differentially coherent detector

The differentially coherent estimator calculates the decision statistic from the product complex conjugate of the previous correlator result and the current correlator results. Therefore the decision statistic can be defined as

$$D(\hat{\theta}) = \left|\sum_{r=1}^{R} Y_r Y_{r-1}^*\right|^2 \tag{3.29}$$

Where Y_r and $Y_{r-1}*$ denote the current and the previous correlator results, respectively. And each correlator output $Y_i$given as

$$Y_i = \sum_{m=iM_cN_s}^{(i+1)M_cN_S-1} r(mTs)exp(-j\hat{\omega}_D mT_s)c(mT_s - \hat{\zeta}T_{chip}) \tag{3.30}$$

which is exactly similar as for coherent integrator. If we let R in the equation 3.29 be equal to 1, then we obtain the following probabilities.

$$P_{fa} = \frac{1}{2\sigma_Y^2} exp\left(-\frac{|Th|}{\sigma_Y^2}\right) \tag{3.31}$$

and

$$P_d = \frac{1}{\pi}\int_{-\infty}^{\infty} K_0\left(\frac{Th}{2\sigma_Y^2} - \tau\right)\psi(\tau)d\tau \tag{3.32}$$

With

$$\psi(t) = \frac{1 + erf\left(\frac{m}{\sqrt{2}}\right)}{4} - \frac{1}{\sqrt{8\pi}}\int_0^{\infty} exp\left(-\frac{(y-m)^2}{2}\right) erf\left(\frac{\neg m + \frac{t}{y}}{\sqrt{2}}\right) dy \tag{3.33}$$

where the $erf(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$ denotes an error function. The $K_0(.)$ is the Hankel function. Moreover, when R is grater than 1 the equations above changes, in (O'Driscoll 2007) some approximation has been derived for R goes to infinite.

3.3 Selection of suitable algorithm

The idea behind this project is to research on data acquisition systems for GPS receiver with respect to space applications, specifically for LEO satellites. Hence, among all the algorithms that are presented above, we select the ones that can be utilized in the space-born receivers.

3.3.1 Detector strategy

Among the presented detector algorithm the coherent detector is the simplest but we can not use that as the receiver is not stationary. Furthermore, equations[3.11]to [3.20] will not be true if Doppler frequency offset and modulated data are introduced (BE 2010). The best detector equation would be differentially coherent. It is said to not only decrease the noise as bit transition occurring, but also increase the sensitivity of the receiver, making it perfect for the space-borne receiver. However, the eq[3.29] shows that we require to delay correlator output, and that is not synthesizable on hardware. Although, the non-coherent detector is not as good as the differentially coherent, but it doesn't consist of concept which are impossible to synthesize. Additional, it is a common used detection algorithm for the most GPS receivers.

3.3.2 Serial search data acquisition

Fortunately for acquisition, all of the algorithms above can be utilized, as long as the fD in the $FD = IF + fD$ is sufficient enough to allow the incoming signal stay within $\pm fD$. However, the accuracy of the system varies from algorithm to another. As we have seen, the accuracy is better if we search within all possible combinations, however this is time and resource consuming. So the best algorithm for space application would be the delay-multiply algorithm, or one of the parallel strategy. But this imposes that the FFT and IFT have less length. Nevertheless, besides being appropriate for space-borne receiver and providing good accuracy, the selected algorithm must be able to be implemented on FPGA. Therefore, as in the case for differentially coherent detector, we better not choose the delay-multiply. To easy the work, we will start with the serial data acquisition, because all its mathematical functions can be implemented on hardware.

Chapter 4

Data acquisition algorithm implementation and simulation using Simulink Toolbox

4.1 Design description

As we have seen earlier, the serial search algorithm is strongly based on three major modules, correlator, threshold detection and control logic.The interconnections between the modules are presented on Figure 4.1. However, as the Figure reveals, a module may consist of several subsystems of functions.

1. Serial correlator

   (a) **Local oscillator**. Its purpose is to generate and provide the in-phase(I) and quadrature(Q) signals to the system.There is a 90^0 phase different between the I and the Q. Otherwise they both have the same amplitude, sampling frequency and the frequency $fD = IF + f_D$. Normally, at the receivers antenna the L1 signal have much higher frequency of up to $1575.42MHz$. However, within the front-end, that frequency is controverted to the receiver's intermediate frequency (IF). We assume the receiver's IF of $1.26MHz$ and the sampling frequency of $5MHz$. For Doppler frequency f_D, we must be able to simulate both for stationary receiver which has f_D equal to 5KHz, as well as the space-born receiver with Doppler value up to 35KHz.

   (b) **C/A gold code**. As seen before, the C/A represents the ID of a satellite which is represented on

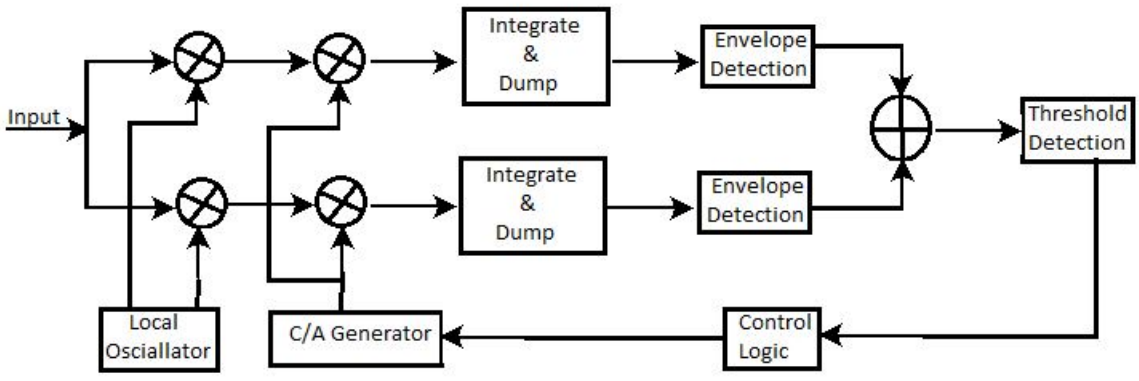

Figure 4.1: Architecture design of serial acquisition

1023 chips. Just as mentioned, this is rather regarded as bit stream than a vector. It means that we have to generate 1023 different bit/chip every certain mount of time, precisely a microsecond. So to achieve that we must have a sample time of exactly $1/1.023MHz$, note that the denominator is our sampling rate. Furthermore, the C/A consist of two 10-bit LFSR, or G1 and G2. Therefore the sampling rate must be the same in both LFSR. The we can follow the description and the phase table provided in the previous chapter to correctly configure the C/A gold code.

(c) **Integrator and dumper**. After I component and Q component are generated and multiplied by the C/A, the system collects a sample which is a equivalent to one chip rate. For that, the integrator and dumper of maximum index equal to 1023 is used. This subsystem is referred to as non-coherent integrator Note that if the first index is 0 then the maximum will be 1022. And mark two separate integrator are required for both I components and Q components.

(d) **Envelop** is actually the square function, and is being used in both I path and Q path. Then after this operation. the two paths are added together before leaving the serial correlator.

2. Threshold detection

(a) **Coherent integrator** is optional, but as long as the receiver is not stationary, it is advised to use it. Here, we do not require to integrate over larger area as for non-coherent integrator. However, we need to bare in mind that the number we choose affects the threshold value. In (Johansson, et al. 1998), it is shown that 10 is enough to be the maximum index of the coherent integrator. And so, we must use 6 as our threshold value, to find the strongest SNR.

(b) **Computing the signal power**. This operation includes reading a set of data and computation of the highest value and the average value within the read data. Then the power signal is obtained by dividing the highest value by the average value. The quotient, which in our case is denoted as the SNR is compared to the threshold value. It is enough to read 500 samples at a time, since the search is normally done in steps of 500 HZ. However, from table 2.1 in chapter 2, we see that ID No 32 has a code delay of 862. Therefore, the system must read at least up to that delay.

3. Control logic

(a) system state control. Usually, the acquisition system has many states. It could be on, off, idle, acquiring, pending, done, etc. However, the most important is to verify if the satellite is acquired or not. Then based on that, the system either have to change to tracking modus and at the same time reinitialize a new search operation for a specific ID, or if time out, it can terminate the search and start all over.

(b) internal signal control can easily be achieved based on current system state and a wish about what next state going to be. Mainly, we have to clock signals, a 1.023 MHz and a 1.26MHz. However, in combination with that states we can easily manipulate the internal subsystem. This could if we want to recalculate the mid-results, such reading or recalculation of the SNR, and so on.

4.2 Block configuration

4.2.1 C/A gold code generation

There are many ways to generate the gold code, either based on Simulink or via Mathscript coding. Either ways, the main key elements in the PRN generation are the linear feedback shift register (LFSR), and the correct code phase as described on table 2.1. First step is to design or configure the LFSRs according to the mathematical expressions in equations 2.1 and 2.2. Next step is the output selection, for G1 it is the last bit/chip. For G2, follow given combinations in table 2.1. Bare in mind that chip delay as well as 10 first chips can be used to verify the configuration, as described in section 2.1.1. Below, we have presented some of methods to configure the gold code on.

- **Unit delays**

  In this configuration, 10 unit delays are connected in series to present one 10-bit shift register. feedbacks can be stretched out according to polynomial explanations. Each delay block involved in this architecture should be initialized to 1, and sampling frequency of 1.023MHz. Drawbacks with this design is that it is space consuming, no reset possibility and is not support for VHDL.

- **D flip-flop**

  On hardware level, a shift register is set of flip-flops (FFs) that install data. Each FF in this set, is equivalent to a single bit in the n-bit register. Therefore, each 10-bit LFSR requires 10 FFs. In our simulation we require a serial register, so that all FFs are connected in serial. Similar to unit delay configuration, the feedbacks to the registers are set with respect to their mathematical expressions and output that satisfy the phase table. The advantages of this configuration is that it gives a feeling of working with hardware. However, the main drawback is that it acts as a hardware in a such way that external clock, reset and ground have to considered. At the same time, it is not in support with HDL coder.

- **Gold sequence generator**

  Simulink provides a faster and easier solution for gold code generation. Two of the present options are shown on figure A.1. With the gold sequence generator, the complete ID sequence can be generated from a single block. However, in this simulation we prefer to utilize two separate LFSR to generate a single ID as shown on this figure

  To utilize PN generator, at least these property must be properly set-up:

  - Generator polynomial, which is similar to the mathematical expression of G.
  - Initial states, similar to the flip-flops. If reset the sequence value goes to 1.
  - Output mask vector. Bear in mind that the output of G1 is always last bit, yet for G2 the output varies with the code phase assigned for each ID.
  - Sample time, is very important especially for generating specific chip length. In our case we have to generate 1023 chips every micro-second. Therefore, our sampling rate is $\frac{1}{1.023e^6}$.

4.2.2 Oscillator

In the GNSS signal acquisition system, oscillation signals are required to set and limit the searching area. Nevertheless, to generate a stimuli signal as well as In-phase and Quadrature signals, we require an oscillator. Simulink offers different blocks that generate oscillating waves. Regarding our targeting mission, the desirable waves must be discrete sinus and cosines. One of the possible approaches is by the help of DSP library in Simulink. We can use a discrete wave generator which is also supported for HDL. With 0 in the phase offset, the output is the sinus, if the phase offset is set to $\pi/2$ cosine is generated. Choosing discrete output, requests a look-up-table computation mode, and that reduces the flexibility in design. Additionally, the amplitude never reaches ± 1. Therefore, we consider other design approaches. The direct digital frequency synthesizer (DDFS) as shown on figure 4.2 is one of the kind mostly used in digital oscillators(Murphy & Slattery 2004). Commonly, the direct look-up table(LUT) with linear interpolation is used to implement the DDFS. However, other method which utilize mathematical series, can also be used to generate a digital sinusoid. In Simulink, the numeric controlled oscillator(NCO) can be advantageous, not only for implementing the DDFS, but also able to simulate different scenarios signal with respect to Doppler frequencies. Figure A.6 shows NCO based DDFS implementation. Notice that the delay acts as the holding register, if removed there will be no signal. However, for counter it is bit tricky. First of all, we are targeting the following signal structures $x1 = A * sin(2\pi f_{IF} + f_D) * T_s$ and $x2 = A * cos(2\pi f_{IF} + f_D) * T_s$, where f_D is the maximum Doppler which we can change. We desire a frequency resolution of 500 Hz, by definition $\Delta f = \frac{1}{T_s * 2^N} Hz$, then computing for N we get 14. Thus, we have such counter on figure A.7, with sampling frequency equal to 5MHz and fD = 10KHz. However, with the suggested implementation on figure A.6, it may result in some warnings. This may occur when quantizer result, from which values of the look-up-table are selected, are not integer. So, the remaining option is to use a simple sine/cosine

source block, that implements mathematical series. However, this may still cause some trouble, as other subsystems may interpret it as a continuous signal. But a "Zero-Order hold" block can be added to resolve the problem.

4.2.3 Integrate and dump

The integrator sums up all the samples in a given range before it outputs the answer, as $y[k] = \sum_{i=(k-1)N}^{k} Nx_i$. The integrate and dump does exist in Simulink and requires less parameter to set up. If for some reasons it doesn't work, an alternative procedure would be a utility of buffer that collects necessary number of data, and then use some of the element to sum them up. However, both integrate & dump and buffer only allow digital or discrete input. Therefore, a "Zero-Order hold" block can come times be used to discretize the input. Just make sure the sampling frequency is correct.

4.3 Digital L1 signal

Recall that the L signal contains gold code, data, and the carrier frequency. The relationship of data and cold code are is the modulo 2 sum as shown figure 4.5, which is further multiplied by the carrier as shown on Figure 4.3. According to the theories in section 2.1, The resulting L should be a BPSK modulated alike. Therefore, to achieve that the Modulo-2 sum must be on NRZ A.1 format, as shown on figure 4.6. Since amplitude for both carrier and Modulo-2 sum switches in ± 1, we get a perfect BPSK as shown on figure 4.9. However, choosing to use such signal as the test signal, the local generated code has to be presented on NRZ format. Otherwise, correlation will face difficulties to determine the incoming signal structure.

4.4 Serial search data acquisition

4.4.1 Correlation in time domain

The key technique behind the serial acquisition is the correlation in time, which compares similarities of incoming signals to the locally generated signals, (I,Q and specific C/A). If signals are similar they will be aligned in time, and so the correlation power becomes high. Thereafter, the average power in Q-arm and I-arm is calculated through integration, whereas envelop is utilized to obtain the absolute power. However, the total correlation power, in serial search data acquisition, is the sum of I power and Q power. Thereafter,

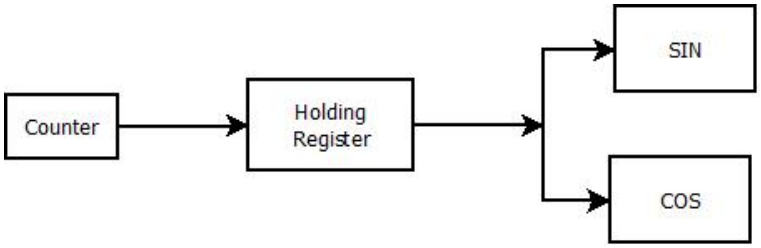

Figure 4.2: Direct digital frequency synthesizer

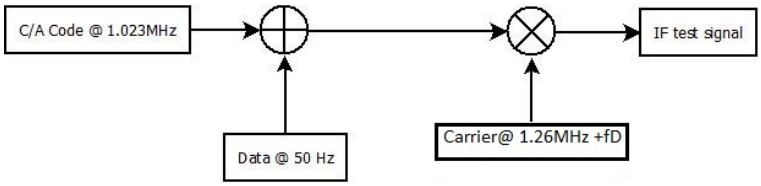

Figure 4.3: L signal generation

the total power is processed in the threshold detector to determine the absence or the presence of a satellite signal.

4.4.2 Peak detection

The first step in peak detection is the non-coherent integrator, which is used because of the increase in Doppler frequency as the receiver is not stationary. Then according to (Johansson et al. 1998) and (Miralles, et al. 2014) the detection strategy begins with reading data that comes out from the non-coherent integrator, followed by computation mechanism of the SNR and comparison with a threshold. In practical, the SNR is the relationship between noise and signal. So let N defines noise in the system, then its expected power can be denoted by $E|N|$. Therefore, if the expected power signal derived by correlator is $E|S|$, then the satellite is statistically acquired if

$$E|SNR| \approx \frac{K * E|N| + K * E|S|}{K * E|N|} = 1 + \frac{E|S|}{E|N|} \tag{4.1}$$

Notice that $E|S|$ must be greater than $E|N|$ in order to generate adequate quotient to exceed the threshold. and if $E|S|$ small, nearly zero the $E|SNR|$ becomes equation 4.2, and of cause satellite is not acquired.

$$E|SNR| \approx \frac{K * E|N|}{K * E|N|} = 1 \tag{4.2}$$

Usually detection procedure follows complex statistics, from which various case scenarios as explained in the detection theory in section 3.2, are considered. However, since we are targeting the hardware implementation of mainly the correlator, we keep this very simple. Figure A.8 shows a simple implementation. The motivation of this implantation is the M-code, which are given in (Johansson et al. 1998). However, we do not use M-codes. Therefore, to read a specific set of data, the buffer is used, from which the search vector is formed. Then, once the vector or data set is created, we find the average value by utilizing the Mean block, and the maximum value by help of the Maximum block. Thereafter, the SNR is computed by dividing the obtained maximum value by the average value. However, to avoid having zero in the denominator which results in infinite quotient, the system forces the average to stay above zero. Which is performed by the "nonzero mean value" subsystem on figure A.8. Finally, the system compares the quotient to the threshold value. Note that even if the non coherent integrator outputs signal that has values of up e^5 or $e^{(x>5)}$, the quotient remains between 0 and 10. Hence 6 as a threshold is reasonable.

4.4.3 Control logic

What makes the control logic interesting is that there is no such fixed way to design it. So in this way creativity may be as important as knowledge of developing hardware. The design on figure A.9 outputs the code phase, if and only if the satellite is acquired, otherwise the code phase remains zero. The tricky part of this module is the generation of control signal. Basically, if a satellite is acquired the PRN corresponding to its ID has to be reset so that the search begins once again from base 1.

4.5 Simulink results

4.5.1 Test signal

Data and PRN C/A

The data and C/A are related with an XOR function. Recall that for each data pulse width there are 20 times 1023 different chips, which almost shade this width as shown on figure 4.4. If zoomed in as on figure 4.5 we can see how fast the C/A oscillates. Note that on figure 4.4, a different ID was used, yet on figure 4.5 satellite ID1 was used.

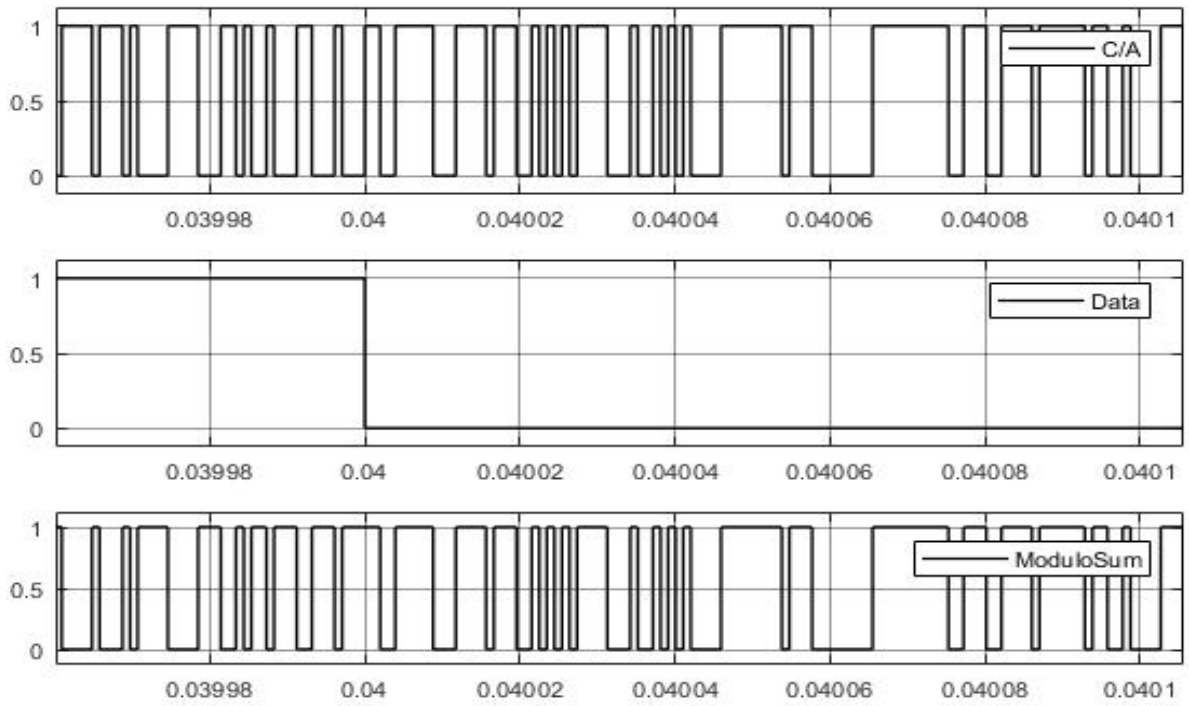

Figure 4.5: Modulo-2 sum of data and C/A for satID1

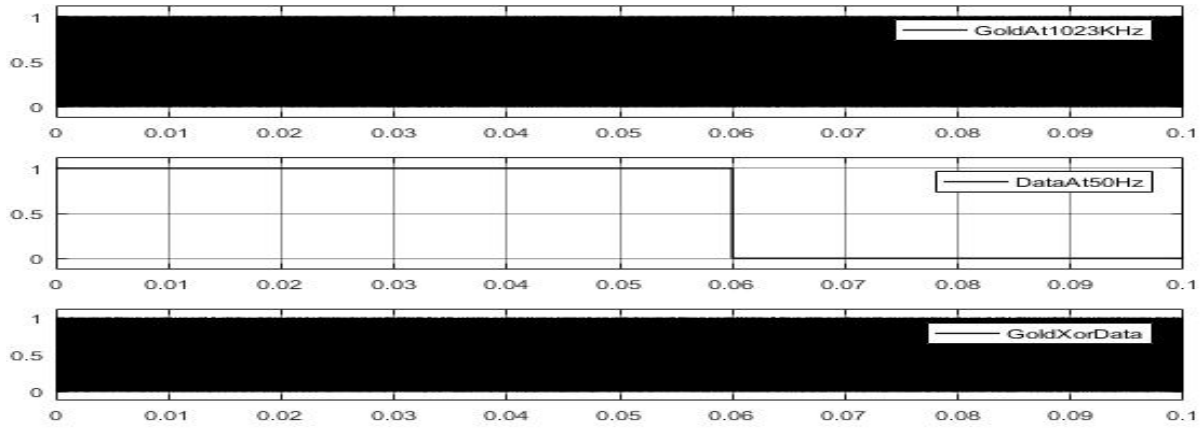

Figure 4.4: 2046 chips for every 20ms

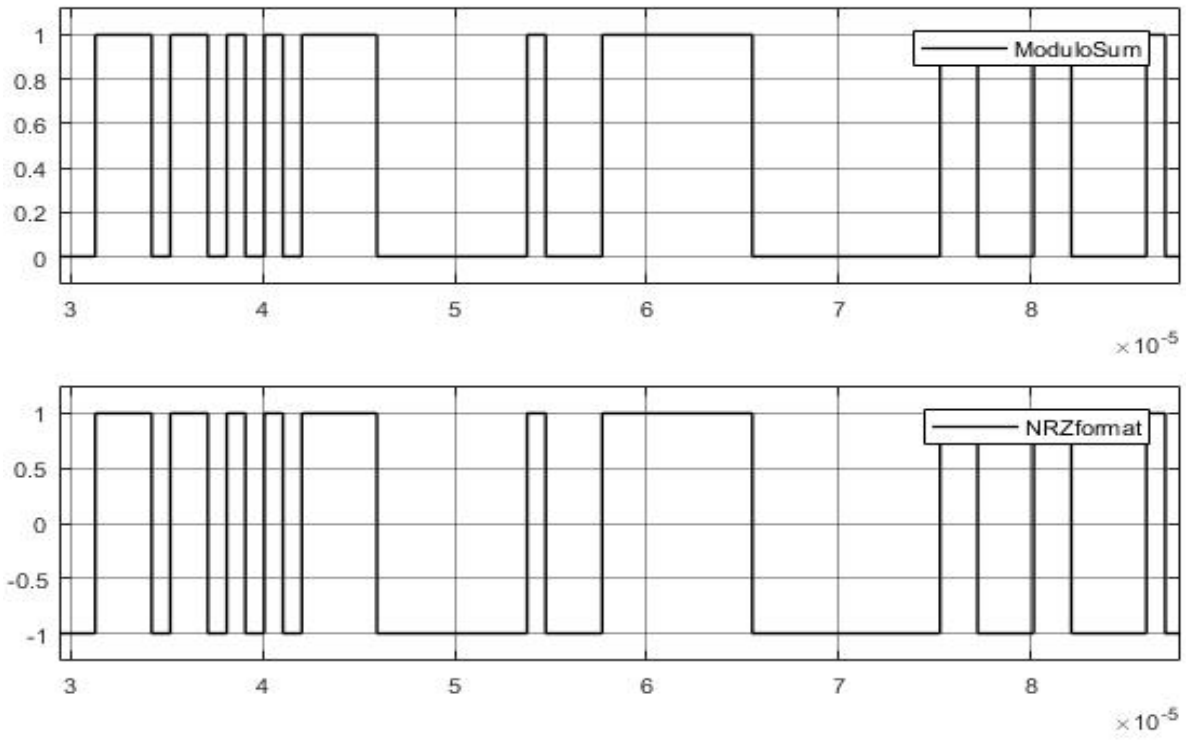

Figure 4.6: Modulo sum on NRZ format

The first ten chips is used to check if the design is correct. Randomly, satellite ID 1, 9, 17 and 32. Hence, results on figure 4.7 acknowledges the 10 first chip given on table 4.1.

Table 4.1: First 10 chip for ID 1, 9, 17, 32

Satellite ID	10 first chip
1	11 0010 0000
9	11 1001 0110
17	10 0110 1110
32	11 1100 1010

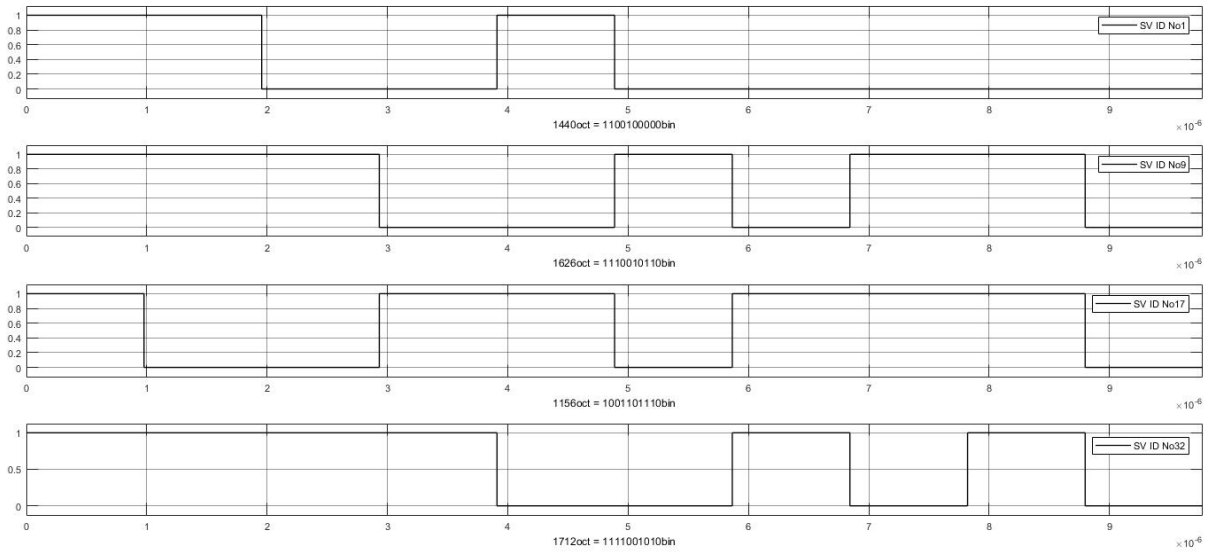

Figure 4.7: 10 first bit for SV ID No 1, 9, 17 and 32

L1 signal

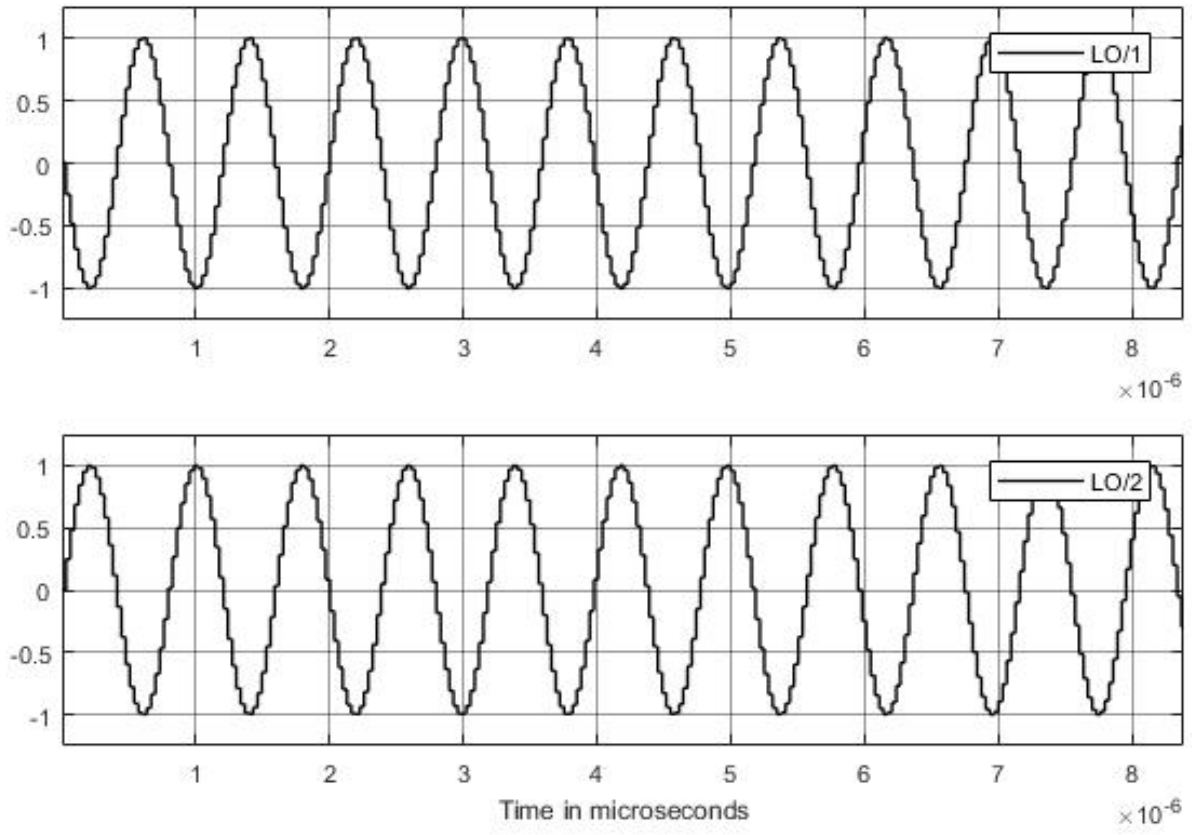

Figure 4.8: In-phase and Quadrature

The first step of our task is to generate a stimulus signal to the acquisition system that mimic the GNSS signal. The stimulus signal contains a satellite ID, data and the carrier wave. The carrier wave is a sinusoid signal frequency of 1.26MHZ and phase frequency that can vary. Note that we assume the L signal after down-converted to the receivers intermediate frequency. That is the reason why we are using 1.26Mhz rather than the frequency of SIS. Although sinusoid wave on Figure on 4.8 are intended to represent LO output, but each of them can be used as a carrier wave. We can see that before adding Doppler, the frequency is exactly 1.26MHz.

As mentioned, the carrier wave carries important information of Data and ranging code, which are set together in Modulo-2 sum as described in the above section. The Modulo-2 sum is then multiplied with the carrier wave. The product is shown on Figure 4.9. We can see that the carrier still propagate with 1.26MHz but shifting the phase as the modulo sum changes the levels. On the other hand, if the modulo sum is between 0 and 1, the resulting product would be as shown on Figure 4.10. Similar, can also happen if local C/A in the next section is on incorrect format.

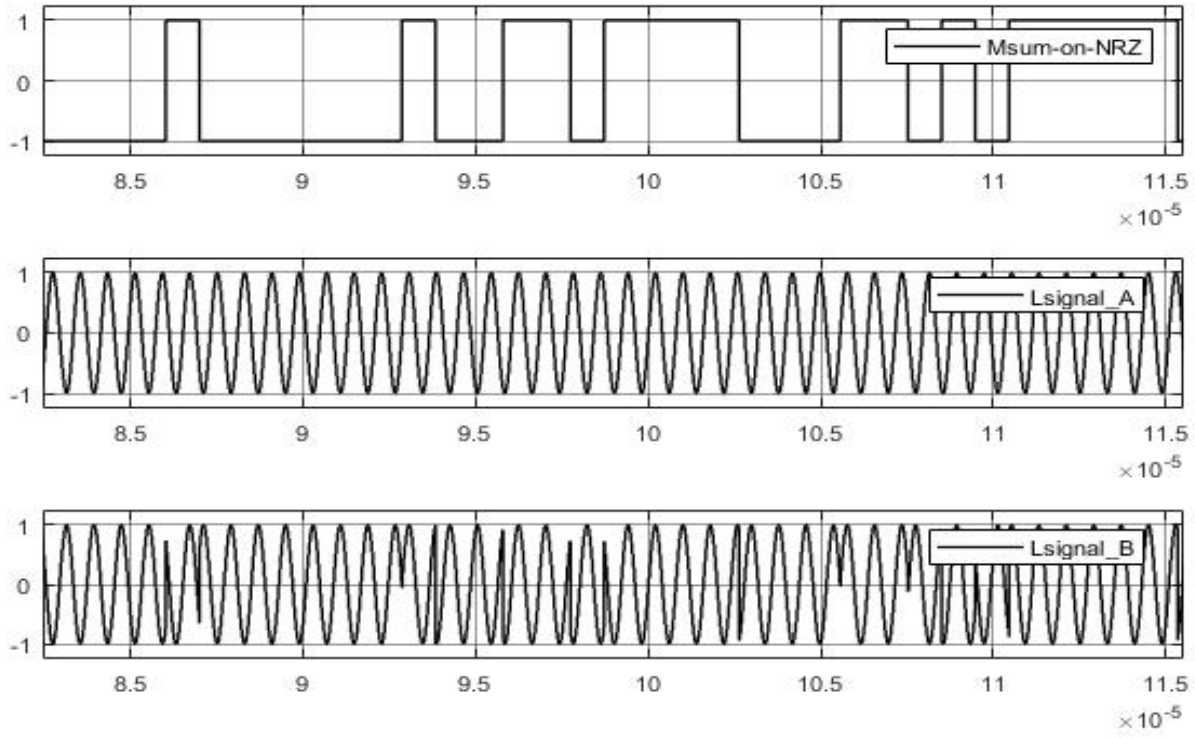

Figure 4.9: L signal being modulated with Modulo-2 sum of data and C/A for ID1

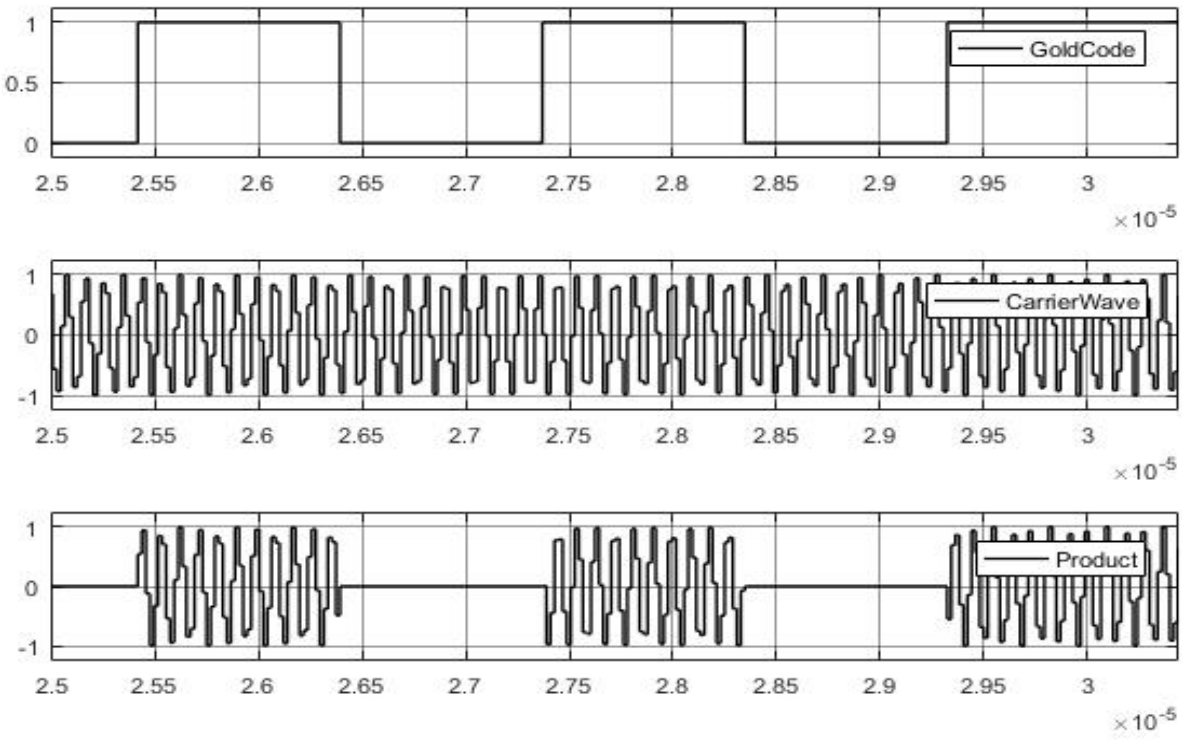

Figure 4.10: Product of carrier and incorrect modulo sum

4.5.2 Serial acquisition

The first process in the serial acquisition is the CAF evaluation which requires a local generation of ranging code, I and Q signals. Basically, we use the same procedure as in previous section to generate the local signal. However, in generation of the local oscillation we must take the the Doppler frequency into account. in general, the Doppler frequency can have any value between the minimum Doppler and the maximum Doppler. However, our design only allows us to operate with constant values. One thing to remember is that Doppler frequency is very small compared to the f_0, so the system is less affected and rather consider the changes as noise. But, this is only true when incoming FD_{in} is less than locally generated FD_{LO} and the difference is not too much. If it happen that FD_{in} exceeds the maximum Doppler in the local signal, correlation power will be affected. Similarity if the local PRN is not the same in the incoming signal the power in the system goes down. Therefore, the serial acquisition utilize both the cross- and autocorrelation to evaluate the CAF for each satellite ID.

Cross correlation

Basically, when serial acquisition is comparing similarities between its locally generated signals and whichever incoming signal, (by whichever we mean signal with different C/A, carrier frequency or Doppler frequency), it can be regarded as cross correlation. However, the cross correlation power will give strong power only if the incoming signal matches the local signal, else power is less. On Figure 4.11, we test the effect of Doppler changes when C/A is correct. The Figure 4.11A is the test signal that has no Doppler frequency, just like the LO on B. Then we compare the result with an LO on Figure 4.11C that has Doppler frequency equal to has 35KHz. After multiplication, we see clearly that 4.11D, which is result of A and B, has much power than on E. By power we mean, the resulting absolute value when all elements are added together. Clearly we see that on E, many of element will cancel each other.

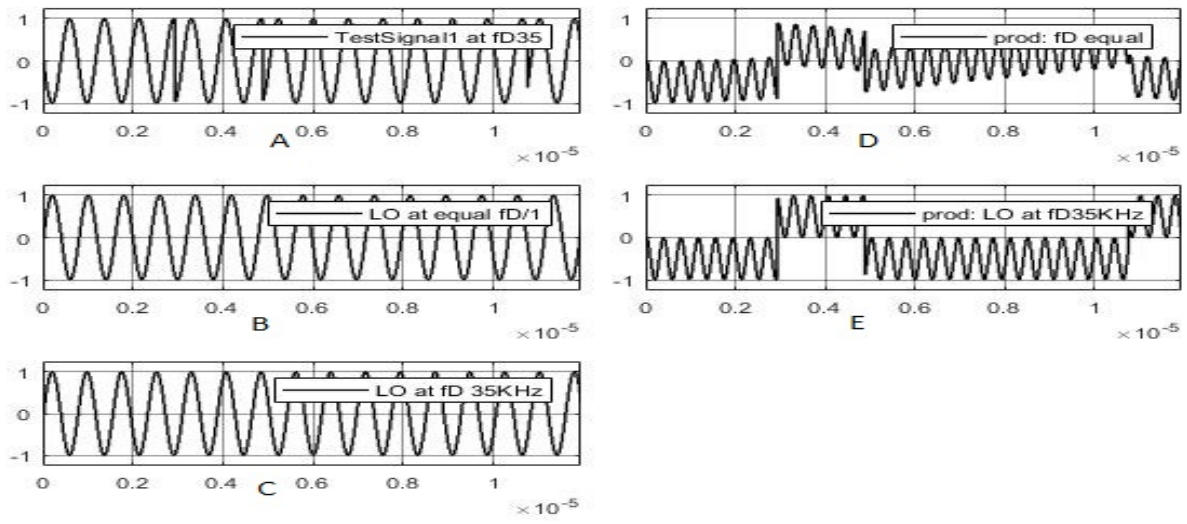

Figure 4.12: Cross correlation, higher Doppler

Incoming signal on Figure 4.12 has 35 KHz is Doppler, same as LO on C. This time we see that the E obtains much of power.

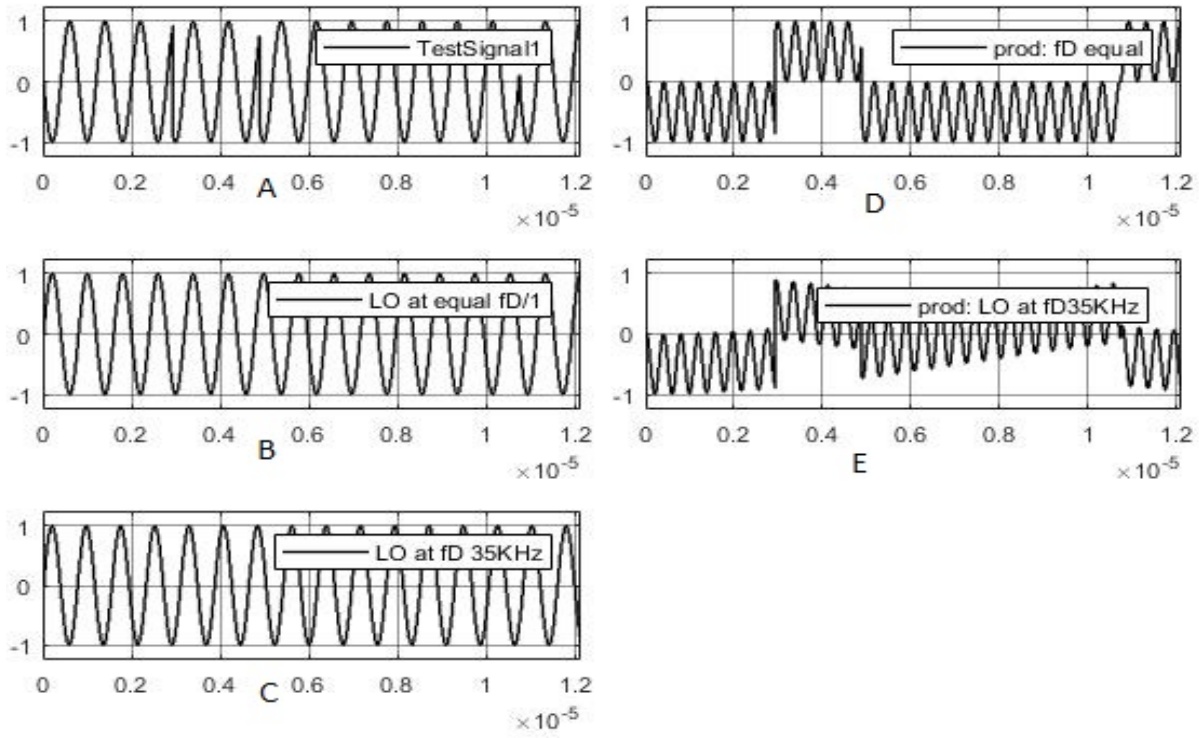

Figure 4.11: Cross-correlation result when incoming signal has less or equal Doppler frequency

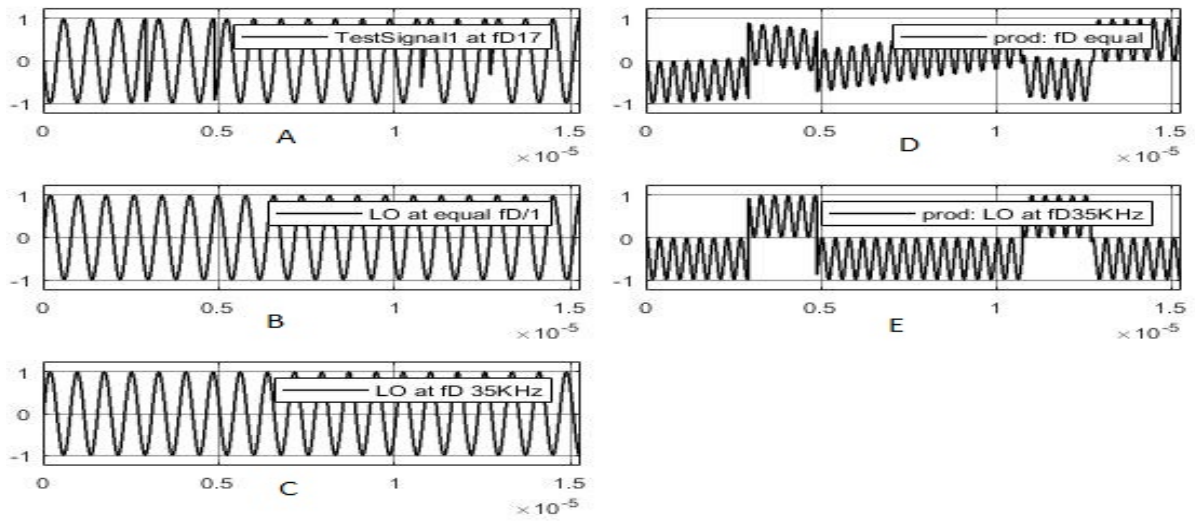

Figure 4.13: When Doppler is just in the middle

If the incoming signal has nearly same Doppler frequency as the maximum Doppler frequency, the energy is still high. As shown On Figure 4.13, the incoming signal on A has Doppler frequency equal to 17 KHz, the LO on B has no Doppler at all, the LO on C has 35 KHz. Then we can see that when Doppler FD_{in} exceed FD_{LO} on D, the energy is distributed unevenly. On the other hand, when incoming signal is within FD_{LO} Max, the energy tends to distribute evenly, and hence E will have much energy. Therefore, the local oscillator must have the maximum Doppler which is expected. Beside the Doppler, C/A is also another parameter that affect the similarities, as well as the correlation power. On Figure 4.14 we have tensionally changed the local PRN from ID1 to ID2. Then we can see that when the incoming signal on A (obviously of ID1), multiplies the PRN of ID2 on B, the energy on A is distributed evenly. However, if we add all those element on C, the sum is going to be small.

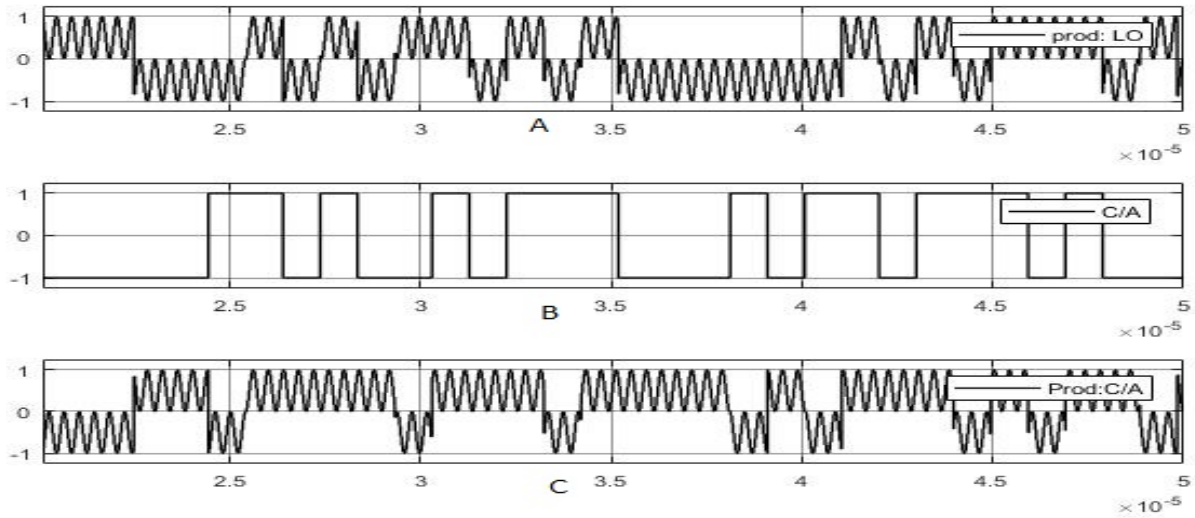

Figure 4.14: cross correlation, equal Doppler but different C/A

Auto correlation

Nonetheless, the serial acquisition evaluate the CAF of a specific satellite ID through utilization of auto correlation. This means that the incoming signal is similar to the local. Therefore, the autocorrelation, is

literally to compare similarity of a signal with itself. In this section, the local C/A and Doppler is the same as in the incoming signal. Then we can see on Figure 4.16 how the auto correlation aligned the incoming signal with both local Doppler and C/A. If the alignment is perfect, the energy becomes high. That is the main principle behind the serial correlation. On Figure 4.17, we can see clearly how consecration is the energy. Note that is we slide and of the signals on A or B, C will suffer from energy loss. Which is exactly what we have seen above.

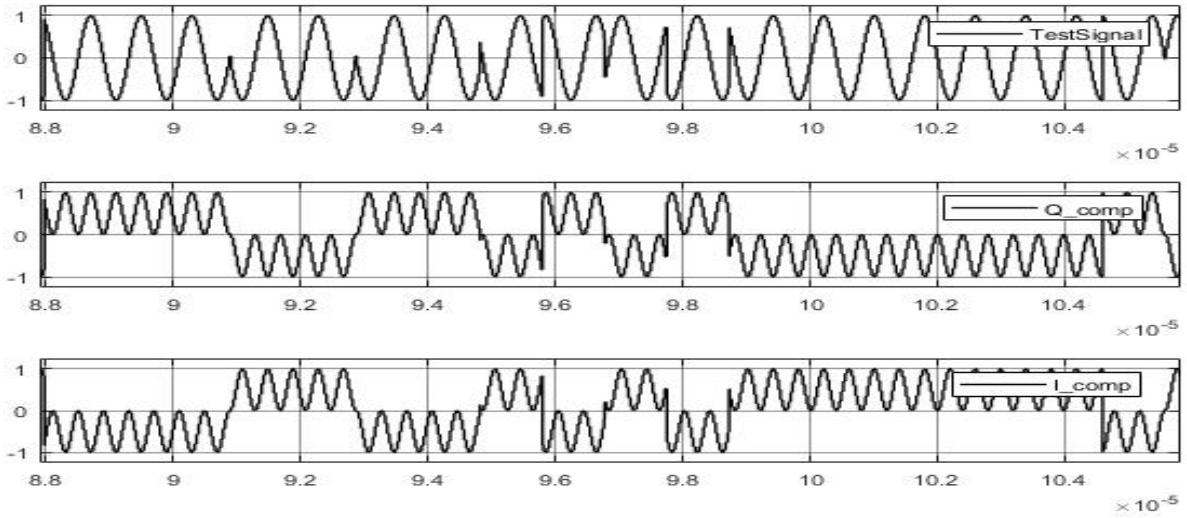

Figure 4.15: I-component and Q-component

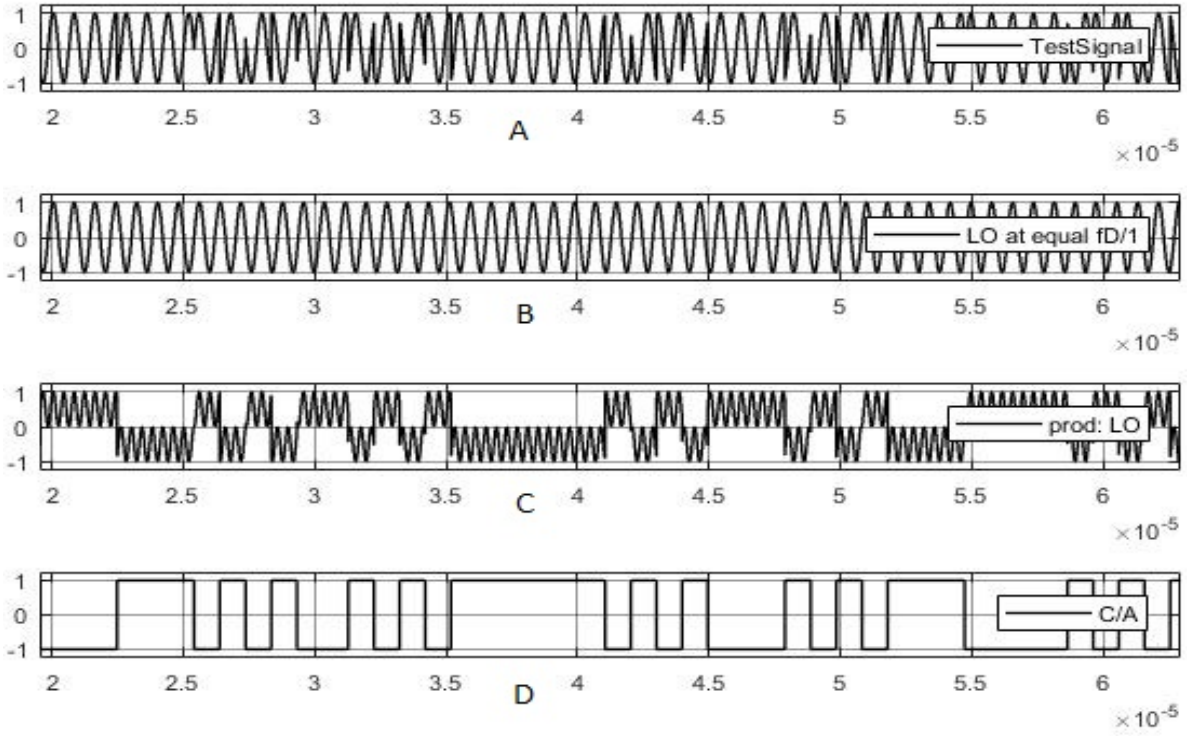

Figure 4.16: Auto correlation, Alignment of C/A in Doppler frequency

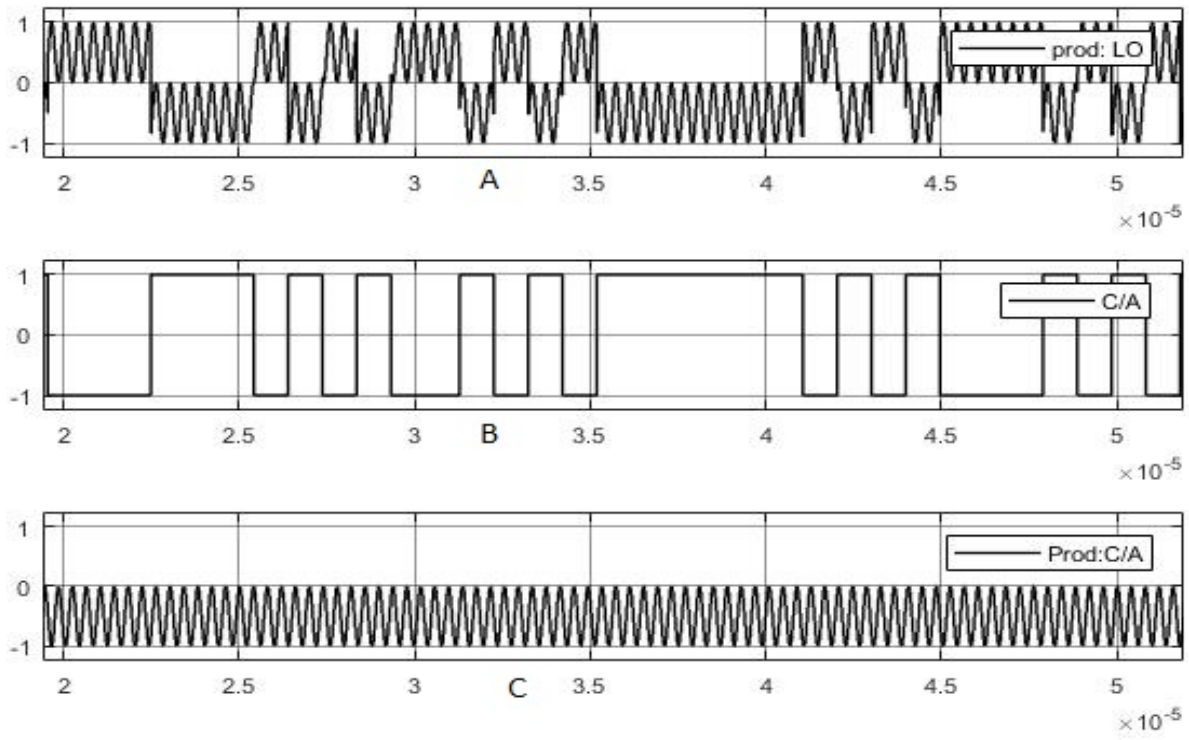

Figure 4.17: Auto correlation, energy distribution

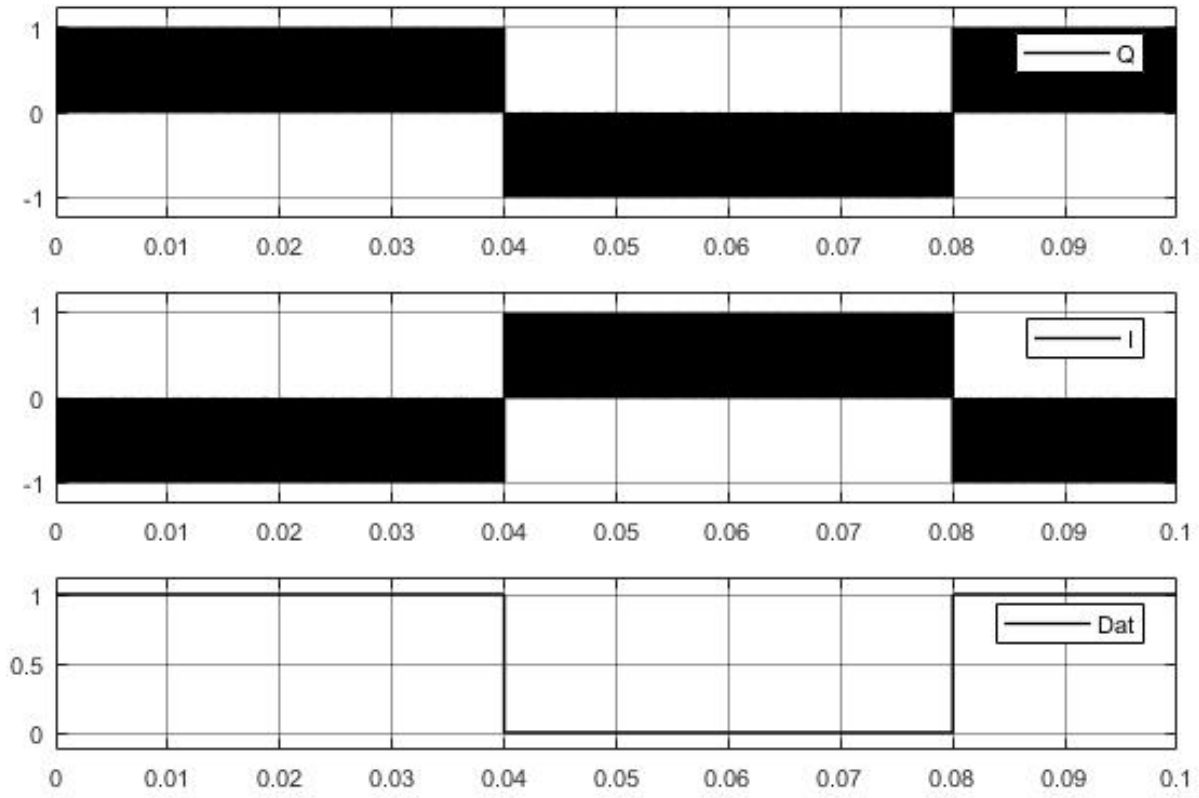

Figure 4.18: Energy distribution along with data PW

Although C/A and the Doppler have direct impact on energy distribution in the serial correlator, but it is also date dependent. Recall that modulo 2 is used to relate data and C/A, whereas, in each data PW holds 20460 chips of CA. nevertheless, the final data bit looks like the inverted version of C/A. Therefore, during correlation energy will be located in the data PW. As shown on Figure 4.18, the I-phase energy is the opposite of the Q-energy. That is true because they are 90^0 phase offset to each other.

Averaging

The auto correlation aligns the incoming signal to the locally generated. However, to compute the correlation power, elements must be summed up. The integrate&Dump takes care of adding exact number of one chip rate, 1023 numbers. Such summation gives the average power over one chip rate. The average can be negative or positive, but it gives a value to a certain number of chips, as shown on Figure 4.19. If we compare Figure 4.18 and Figure 4.19, we can see that the energy structure is still the same for both I and Q. Only that, rather than having energy distributed over a large area, the integrate sums them up. To get the energy in the absolute value, we square it, which is known as enveloping. This time we get only positive value, and they much higher than in the integration. Figure 4.20 shows what happens to the I component, when alignments is correct. On the other hand, Figure 4.21 shows what happens when signals are not properly aligned. The last step of non-coherent serial correlation, is to add the Q-power and the I-power. Figure 4.22 shows the total energy. Notice that the spikes are exactly where the data changes its pulse levels. However, if we normalized the total energy, we would get a nice peak.

If we chose not to use the NRZ format, we would get signal structure as on Figure 4.23. Still, we get some power but it is not properly averaged. And the power will be different in Q and I depending on the type of sinusoid in the test signal.

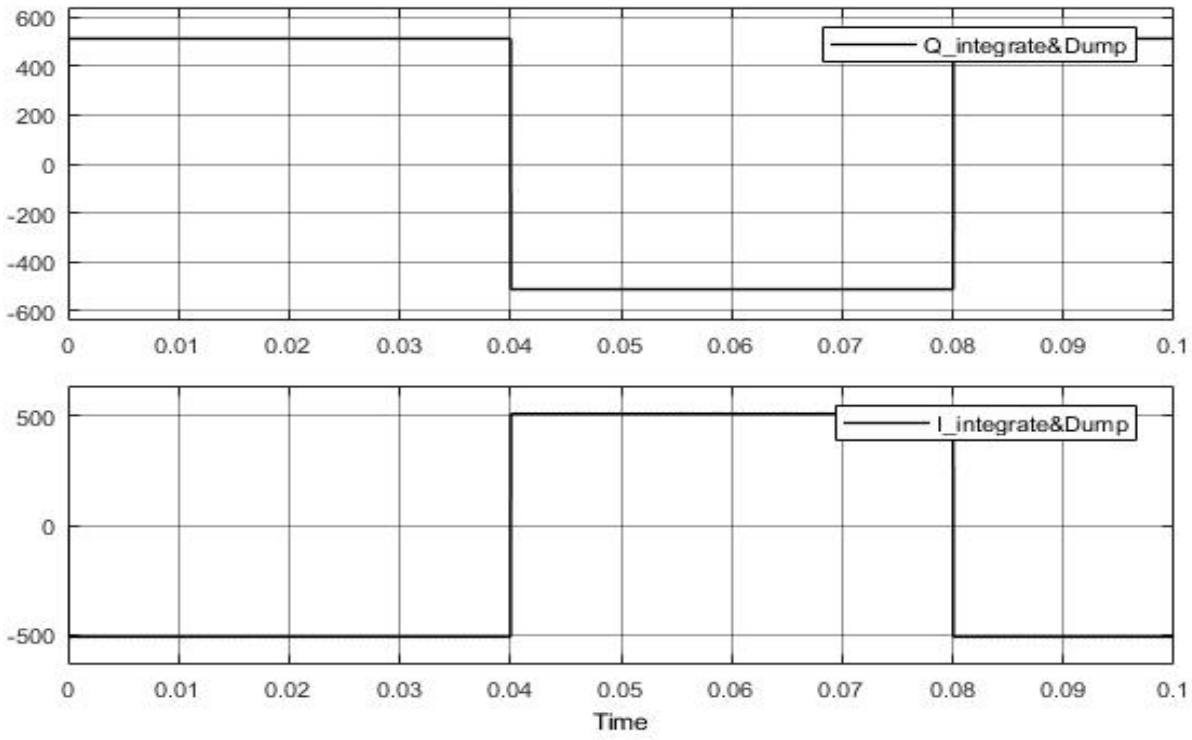

Figure 4.19: Comparison of I arm and Q arm

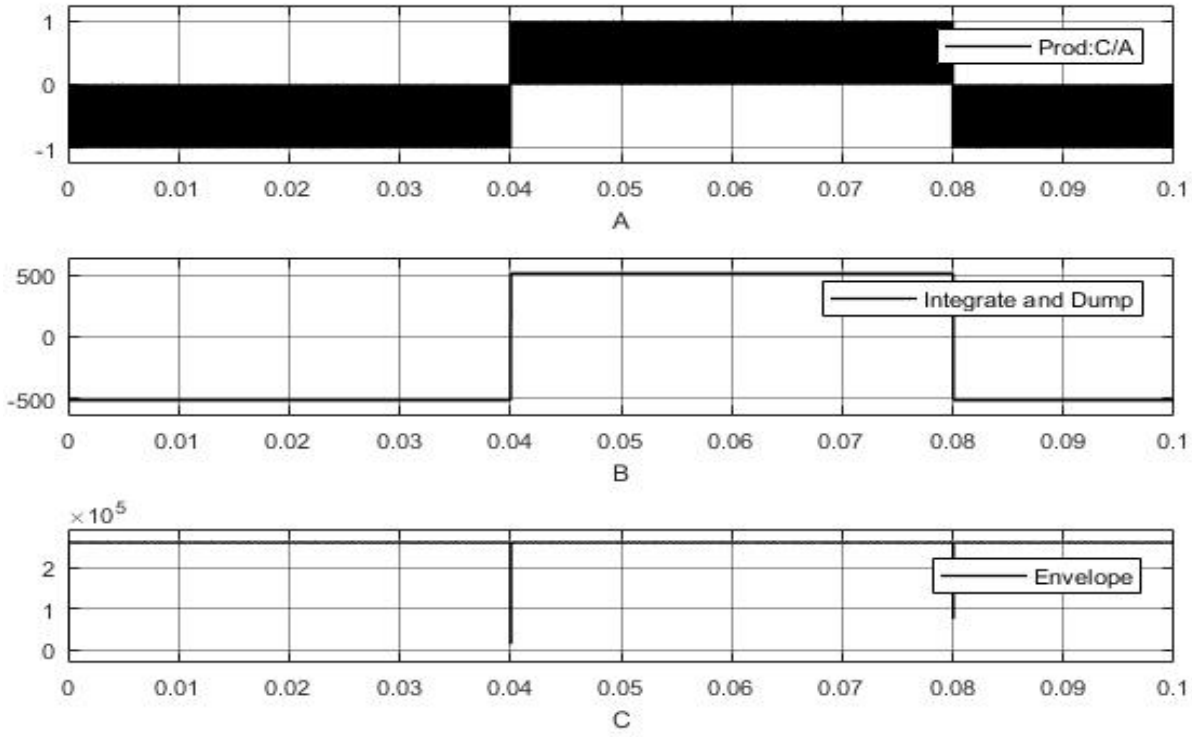

Figure 4.20: Average result when signal are aligned

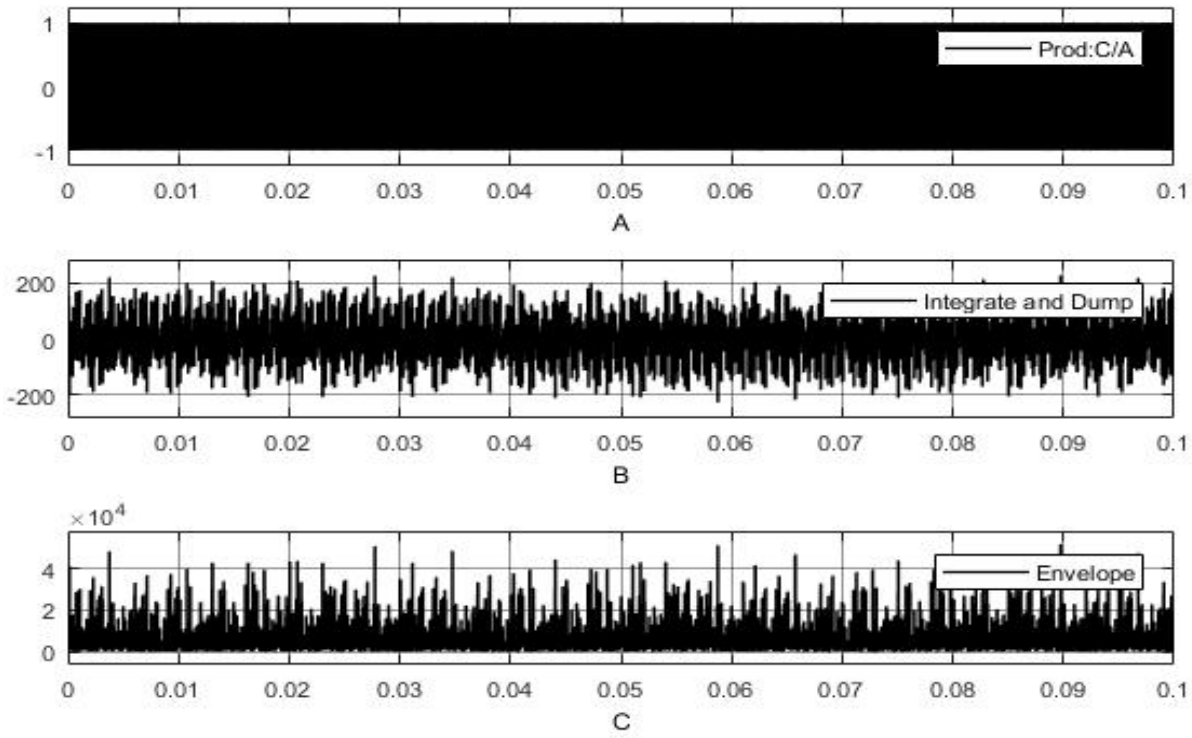

Figure 4.21: Average result when signal are not aligned

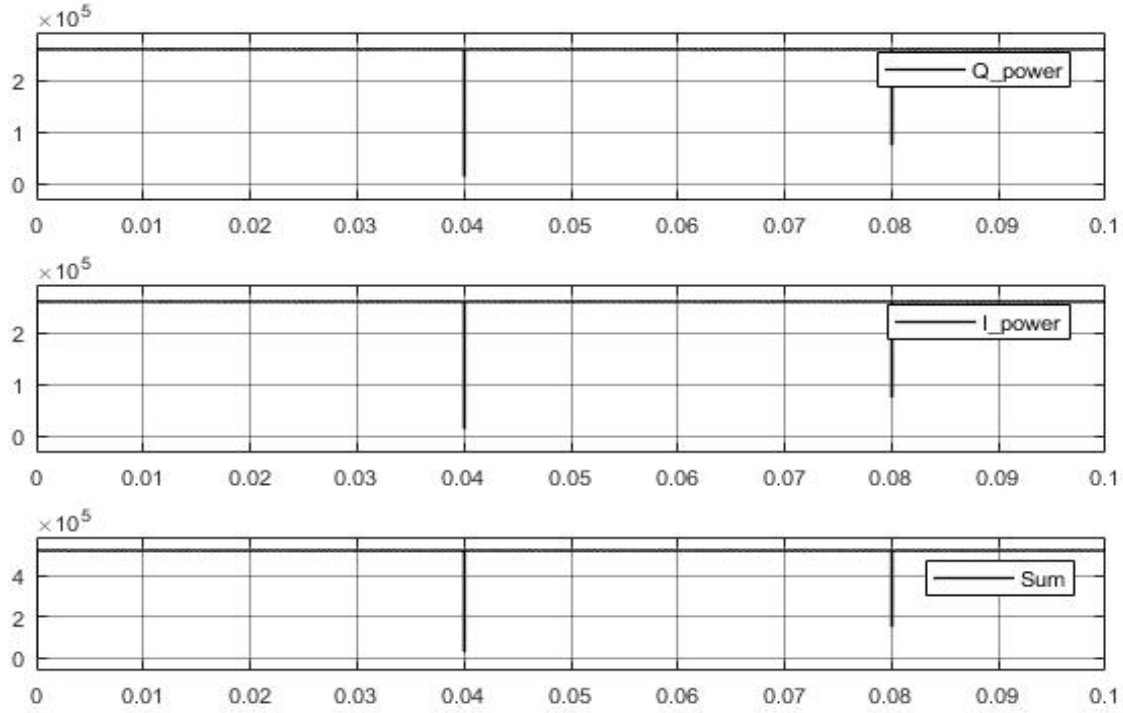

Figure 4.22: Serial correlator power

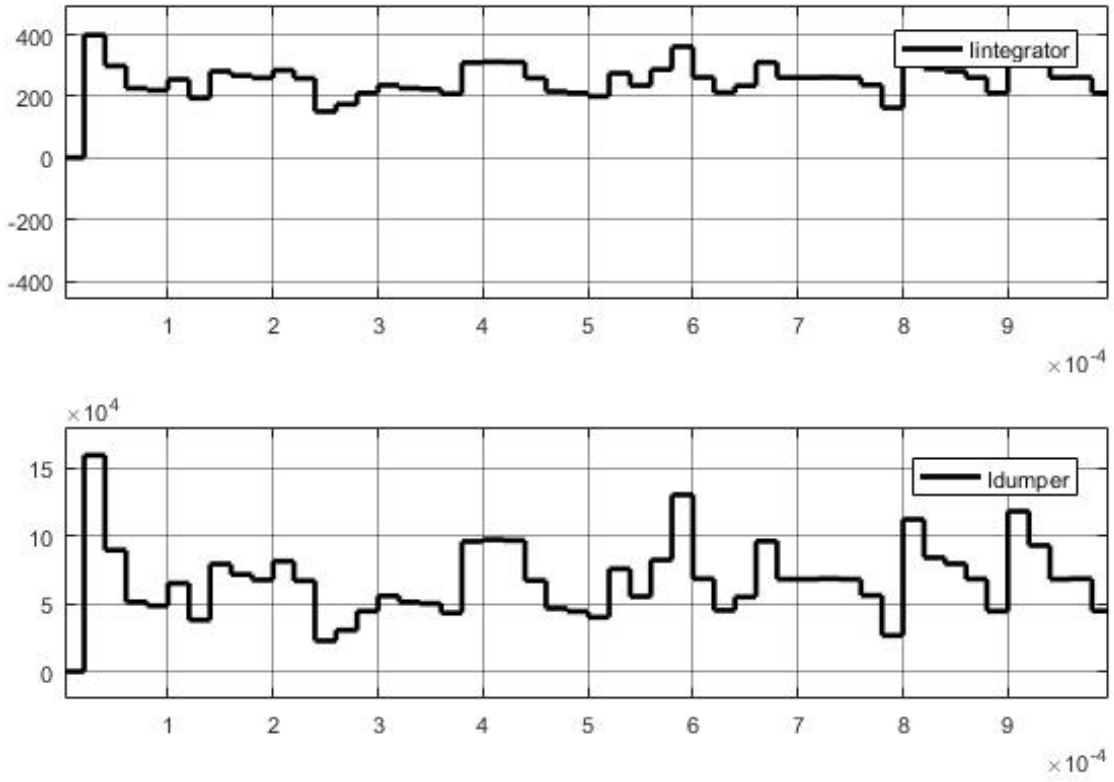

Figure 4.23: Averaging result when NRZ is not used

Detection and control logic

In Appendix section A.2 we attempted to implement equation 4.1 which is modification of equations in section 3.2.2. Our design is presented on Figure A.8. It combines the use of buffer that reads 5000 data and the computation of SNR which is further compared to the threshold. However, such configuration is slower as the SNR computation needs to wait for the buffer to output 5000 data. Therefore, we have modified the design, in a manner that we removed the computation of the NRZ, and rather compare the output directly. First of all we observe the highest power that the serial correlation outputs when signals are not aligned. The values are just less than e^5. And we know that when the signals are aligned, integrator outputs much higher values than e^5. Therefore, our modified design which is presented on Figure A.11, works instantaneously. Observed on Figure 4.24, the acquired signal becomes high once values exceed $5e^5$, else remains zero. At the same time, the local C/A is controlled. If acquired then the C/A resets. Figures 4.25 and 4.26 shows that the C/A is correct, from 10 first chip perspective, after the reset. However, we could not achieve the loop, thus we see that there is no correlation between the resetting and correlation power.

Figure 4.24: Threshold detection and C/A reset

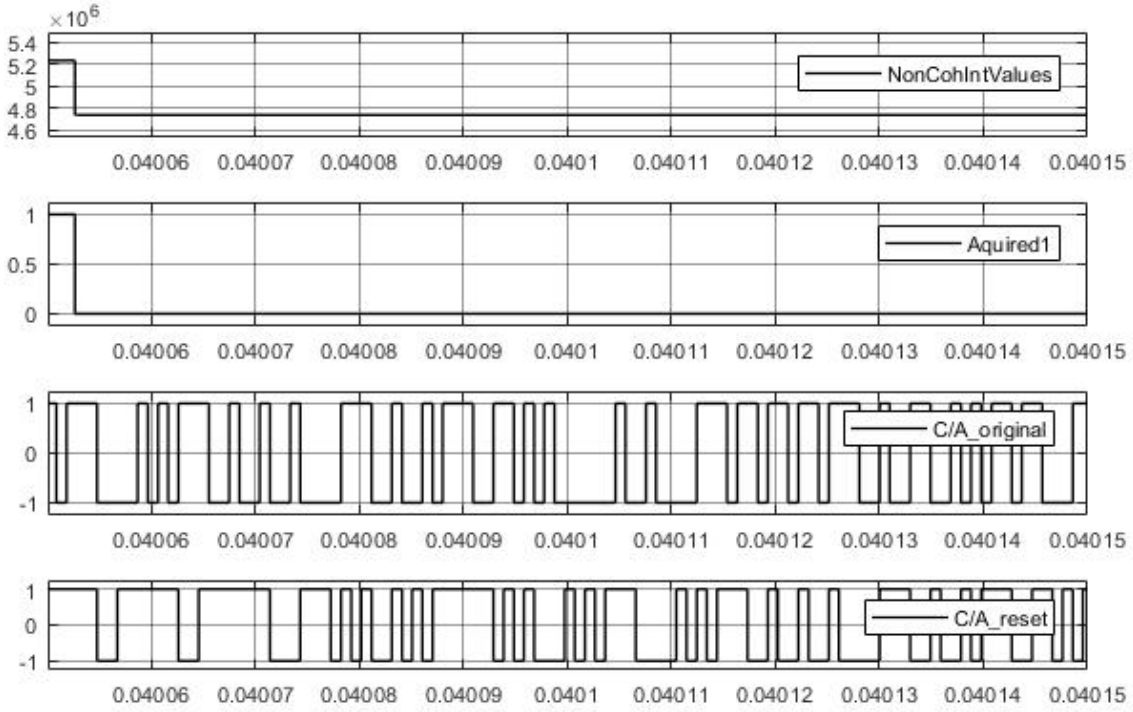

Figure 4.25: Reset and first 10 chip

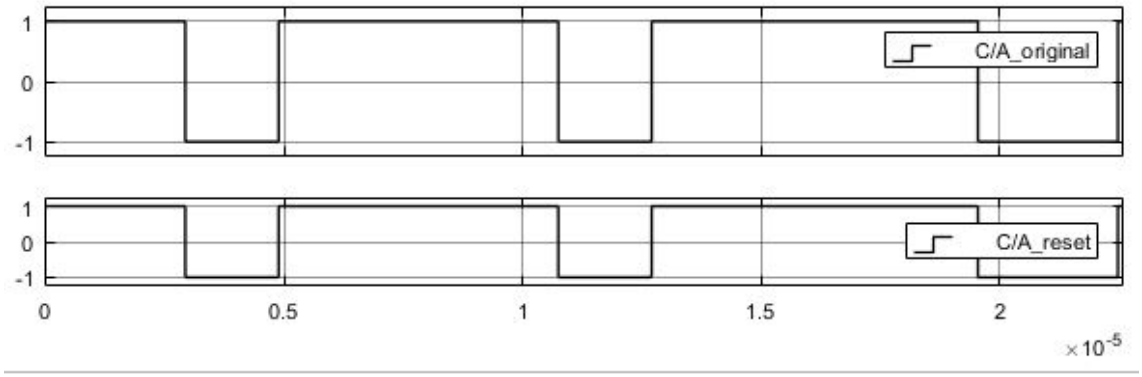

Figure 4.26: C/A before

Chapter 5

Implementation of data acquisition algorithm on FPGA

In this Chapter we develop VHDL code which correspond to Simulink results in previous chapter. However, L1CA is only required for stimulation purposes. That means, for implementation, we only need to design acquisition module which also requires frequency divider. Regarding peak detection, we use simple method that only compares the correlation power to the threshold. Therefore, we do not go deep in detection strategy.

5.1 VHDL design on ISE software design tool

In this section we describe how we intend to develop the VHDL code. Starting from the basic functions such as clock divider, ranging code, integrate&dump and oscillator.

1. **Clock divider**
   It is an important part of design. We require exact frequency for C/A generator as well as for local oscillator. To achieve that, the precise pre-scale value must be calculated, simply by equation 5.1. Thereafter, use the pre-scale in given code on B.1 to get necessary frequencies

$$presc = \frac{systemclockek}{desiredclockek}\frac{1}{2}[Hz] \tag{5.1}$$

2. **Raging code**
   The raging code is directly generated from Simulink HDL generator. On top of the VHDL file, the Simulink generates a package file. This file is required to run the VHDL file in ISE design suit. However, we must make sure that both the clock signal as well as ranging sequence is correct. Both the chip delay and the 10 first chips of a specific satellite ID, can be used to verify this VHDL code.

3. **Integrate and dump**
   This function is required in serial acquisition, to add correlation power. However, we can't generate the VHDL directly from Simulink. Therefore, we must design our own. One solution is to use the IP Core from the ISE software. However, it is relatively easy to design VHDL code for an integrate&dump. As shown on Figure5.1, we require two IF-loops, one for internal sum, another for counter.

4. **Oscillator**
   The oscillator is required to generate both the sinus wave and the cosine wave. One of the approaches to maintain oscillation digitally, is the use of direct digital frequency synthesis (DDFS). In ISE environment, there exist the IP Core for DDFS which can be used for simulation only. Similarly, we can generate VHDL for DDSF from Simulink. If we have to design our own DDFS, then we might follow the architecture presented on figure 5.3.

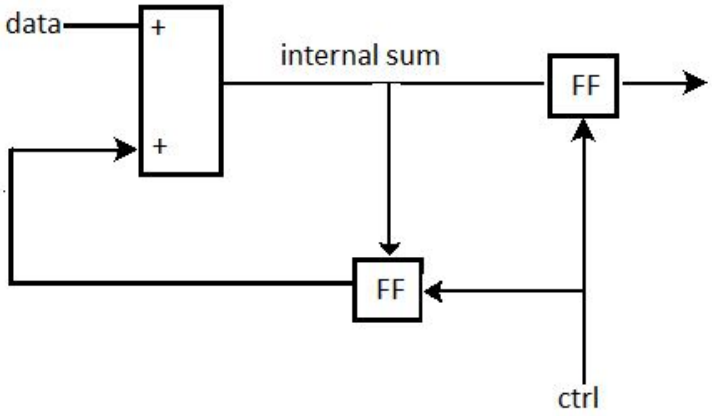

Figure 5.1: Integrator and dump architecture design

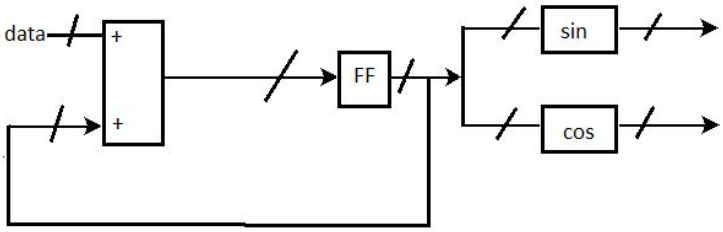

Figure 5.3: DDFS architecture design

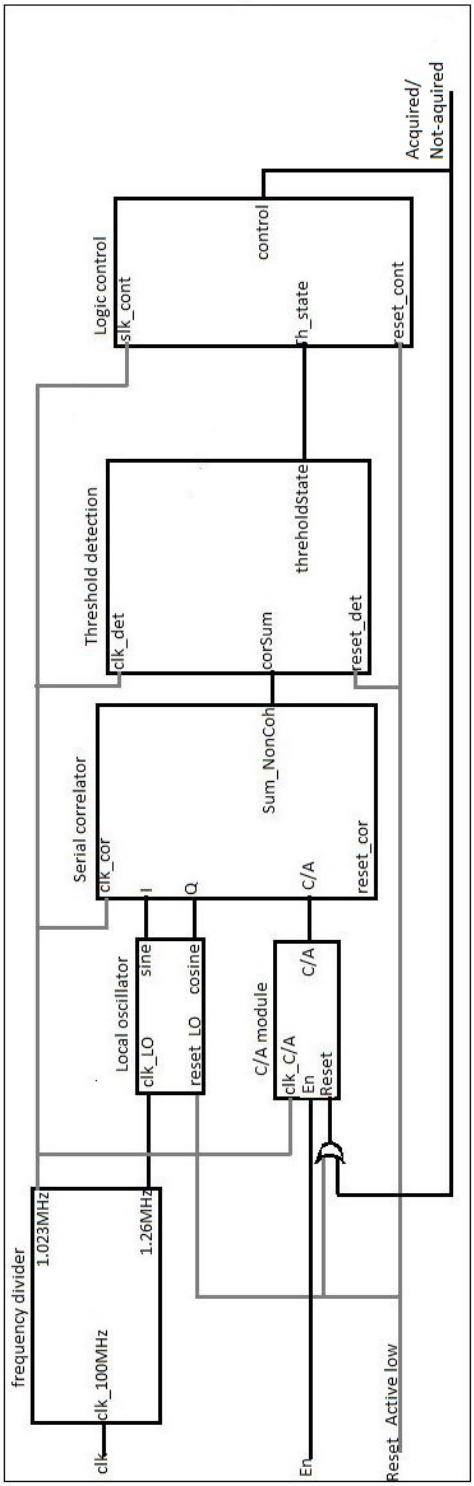

Figure 5.2: Top-view of an intended architecture for stimulation only

5.2 ISim stimulation results

5.2.1 Frequency divider

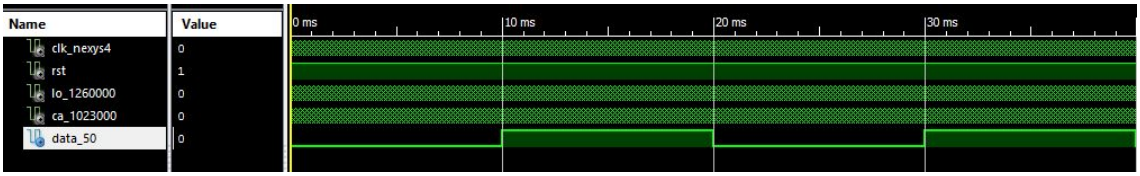

Figure 5.4: Clock frequencies for data, CA and LO

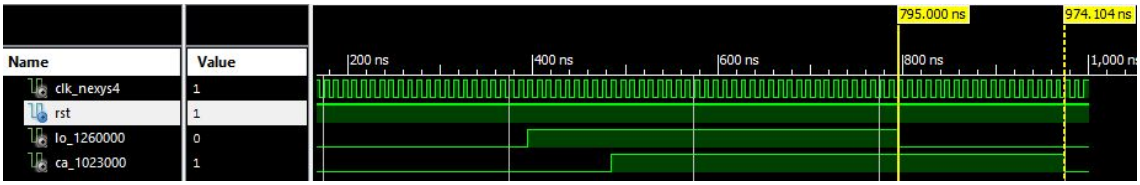

Figure 5.5: Clock frequency for CA and LO

Based on VHDL design in B.1, Figure 5.4 shows clock frequencies for PRN, data and LO. The data frequency is 50 MHz which gives a clock signal that is equal to 20 ms. Then Figure 5.5 presents the frequencies for LO and ranging code. The LO runs on 1.26MHz, that is one oscillation per 795.0 ns. The PRN frequency is 1.023MHz which gives about 974.01 ns.

5.2.2 L1CA for stimulation

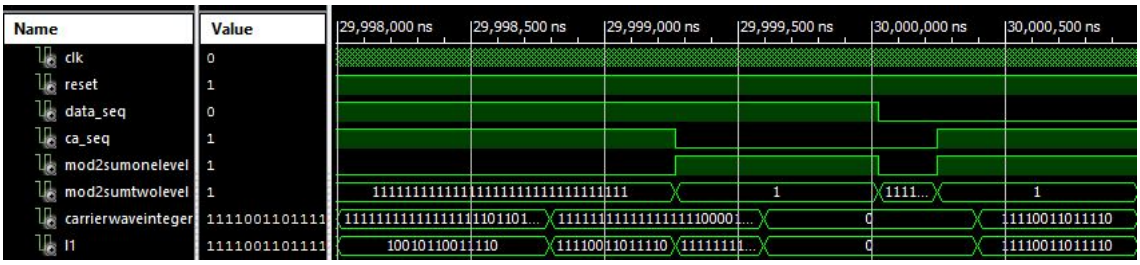

Figure 5.6: Signal to test

Figure 5.6 shows the stimulation result of the L1CA, it follows the VHDL code provided in B.1. We can see the Modulo-2 sum between data and CA of satellite 1, being represented on NRZ format before multiplies the carrier wave. The VHDL code for the NRZ, is presented in B.1. Although, this is not real L1CA, but it is enough to generate stimuli signal to our main system.

5.2.3 Serial acquisition results

The complete VHDL code for serial acquisition, including the test signal is provided in B.1. Whereas, results are presented on Figure5.7 and Figure 5.8. It may look like these figures don't give any information, but they all show that the system works, at least with proper signal type. However, the DDFS which provide sine and cosine, doesn't output sinusoid signal, which makes it odd to verify the internal signals.

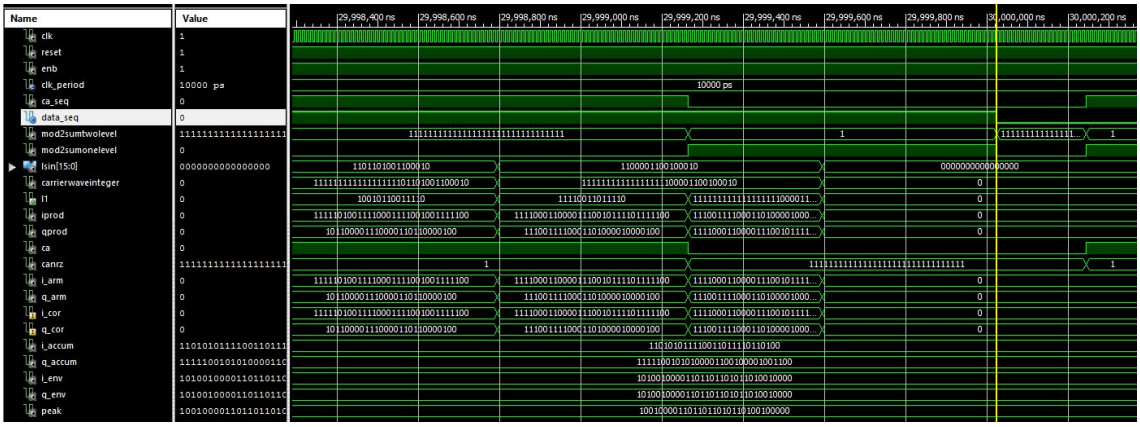

Figure 5.7: Stimulation results

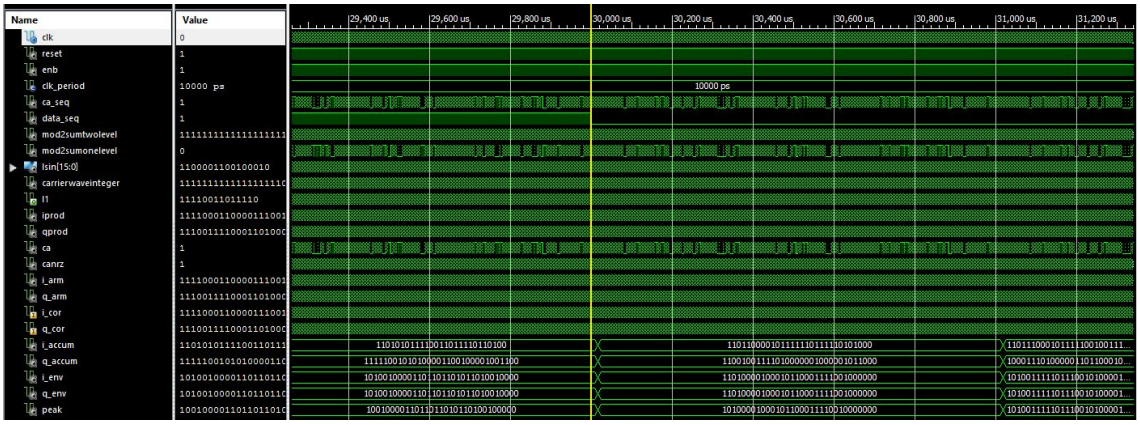

Figure 5.8: Signal to test

5.2.4 Detection and logic control

As mentioned in the introduction of this chapter, we design a simple detection method that only compares the total correlation power to a threshold. Then, according to whether correlation power exceeds the threshold, the Satellite ID is either acquired or not acquired, as shown on Figure 5.9. If acquired, the system has to establish a new search, by resetting the PRN, as shown on Figure 5.10. Moreover, Figure 5.11 verifies the first 10 chips to be correct, and also is the reset.

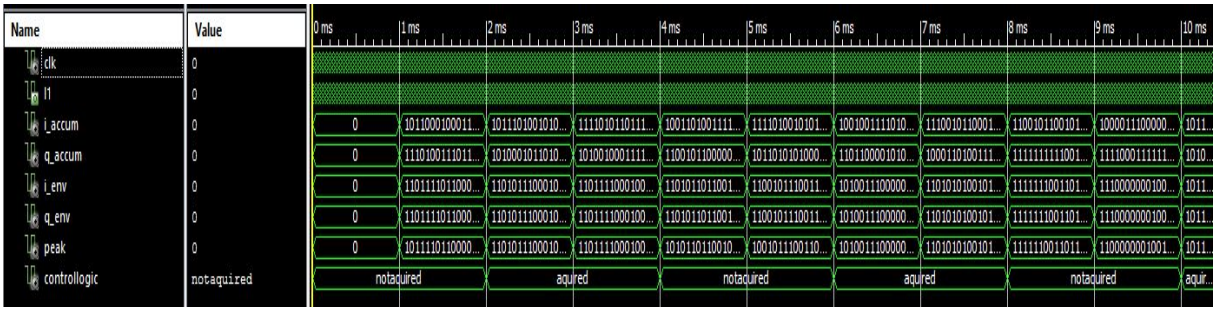

Figure 5.9: Detection

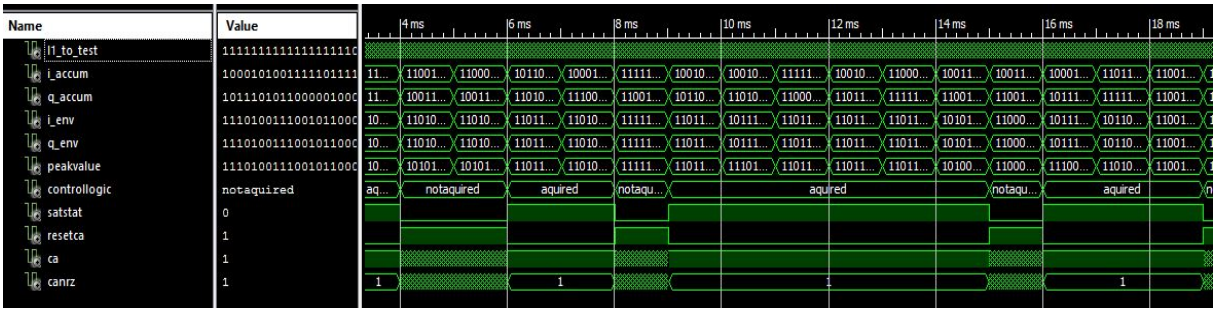

Figure 5.10: Detection showing PRN being reset

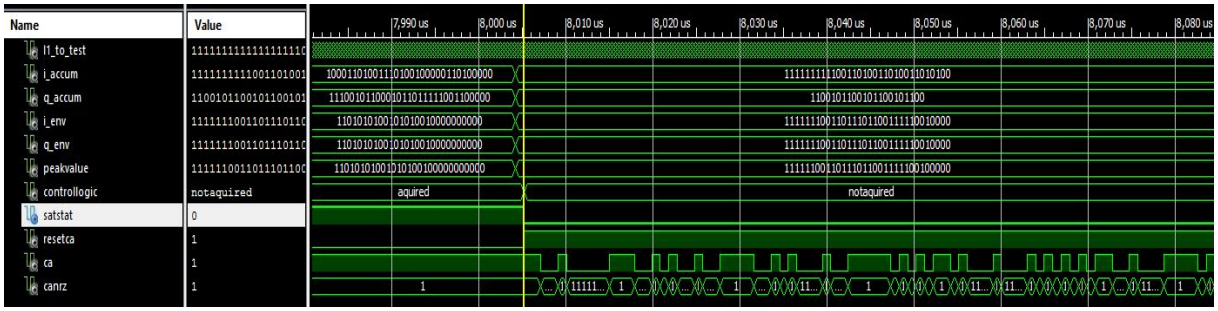

Figure 5.11: Reset and 10 first chip

5.3 Verification on oscilloscope

In this section we verify, with the oscilloscope, that the implementation of frequency divider and CA on FPGA is correct. However, due to some technical issues, there are some ripples formed on the waveform, otherwise every signal shown below is correct.

5.3.1 Frequency divider

Figure 5.12 verifies the LO frequency of 1.26MHz. Each grid on vertical line reads 200 ns, it takes almost 4 grids to complete a single period. which gives nearly 1.26MHz clock frequency.

The CA clock on Figure 5.13 completes a single period in 977.5 ns, which gives a 1.023MHz clock.

Figure 5.14 verifies that the data frequency is 50 Hz. Notice that the each grid is 5ms. However, we do not require data frequency to implement data acquisition on FPGA.

1.26MHz for LO

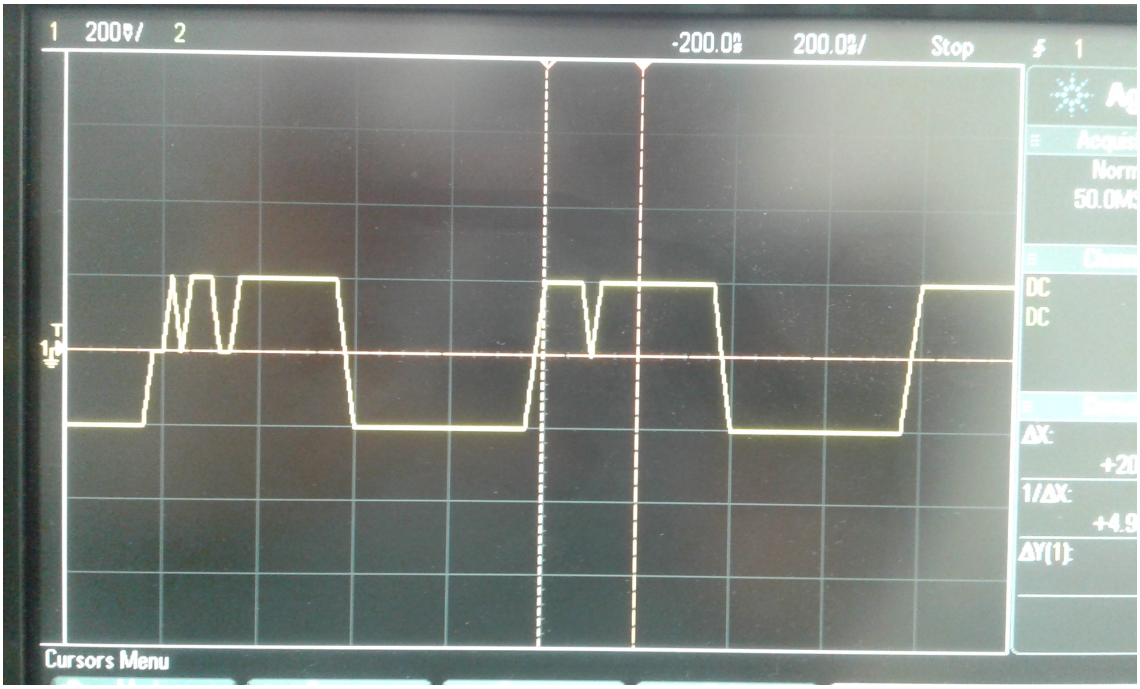

Figure 5.12: LO clock signal of 793.6 ns periodicity

1.023MZ for CA

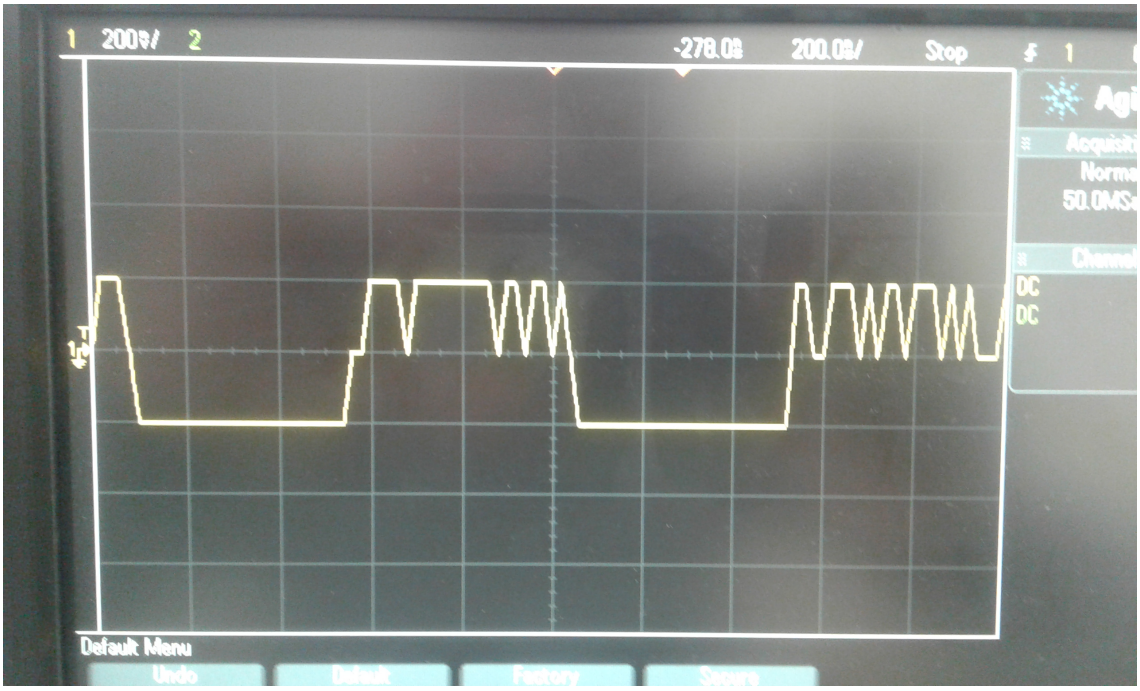

Figure 5.13: CA clock signal of 977.5 ns periodicity

50 HZ for Data

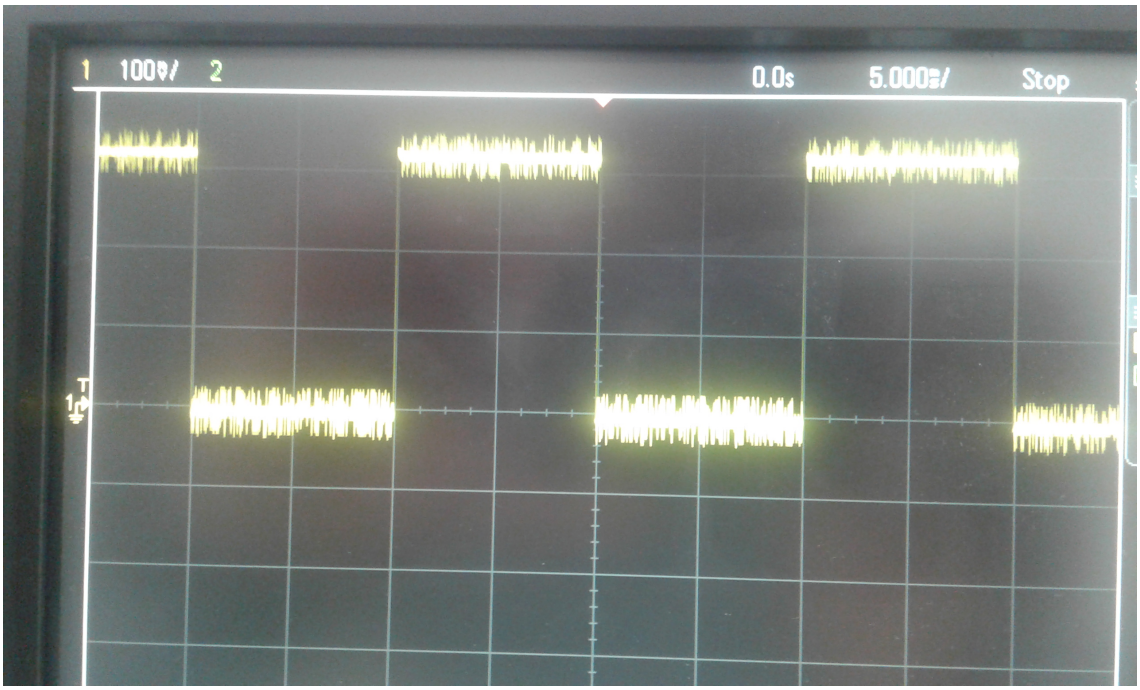

Figure 5.14: CA clock signal of 20 ms periodicity

5.3.2 C/A

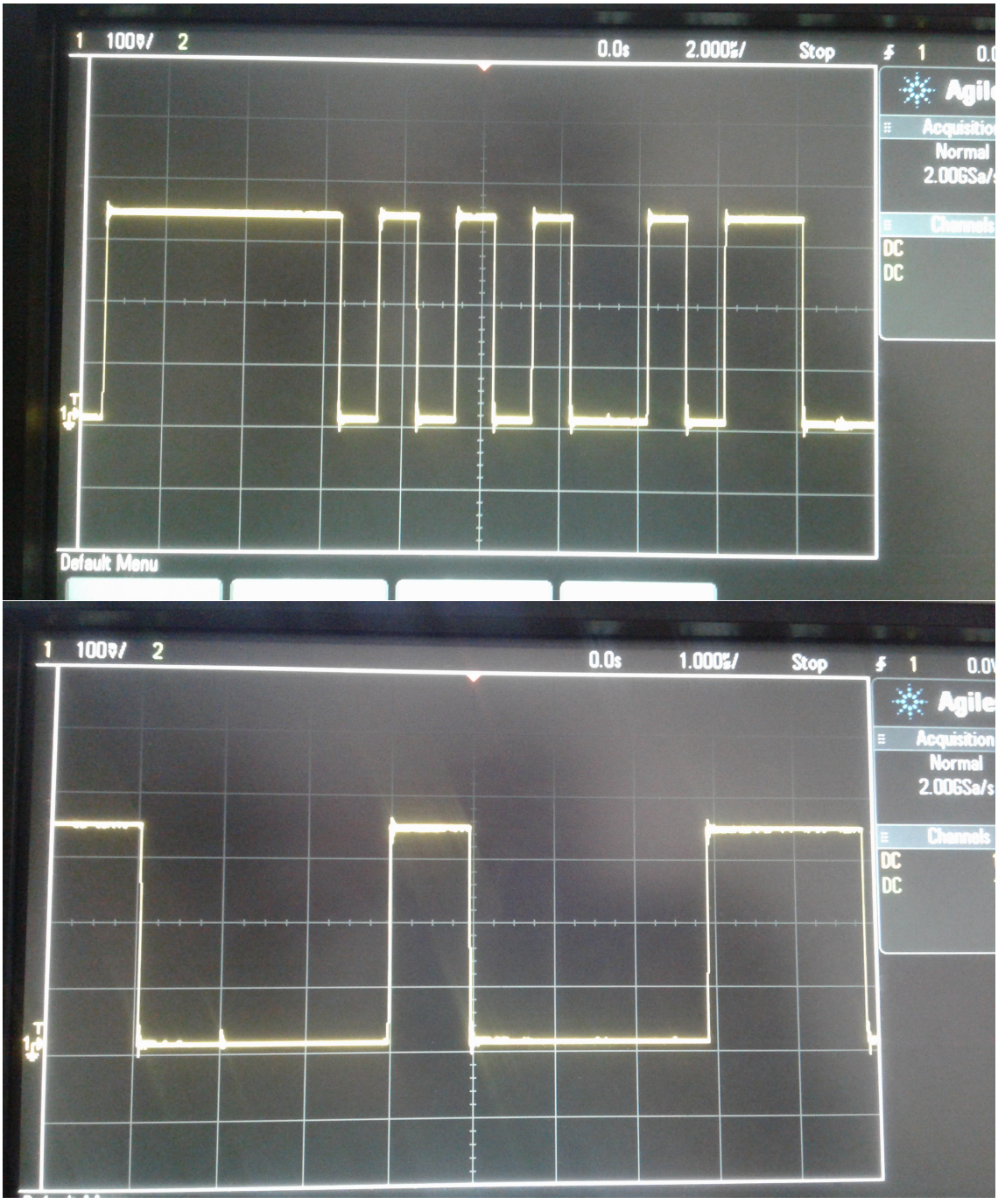

Figure 5.15: C/A for satellite ID1

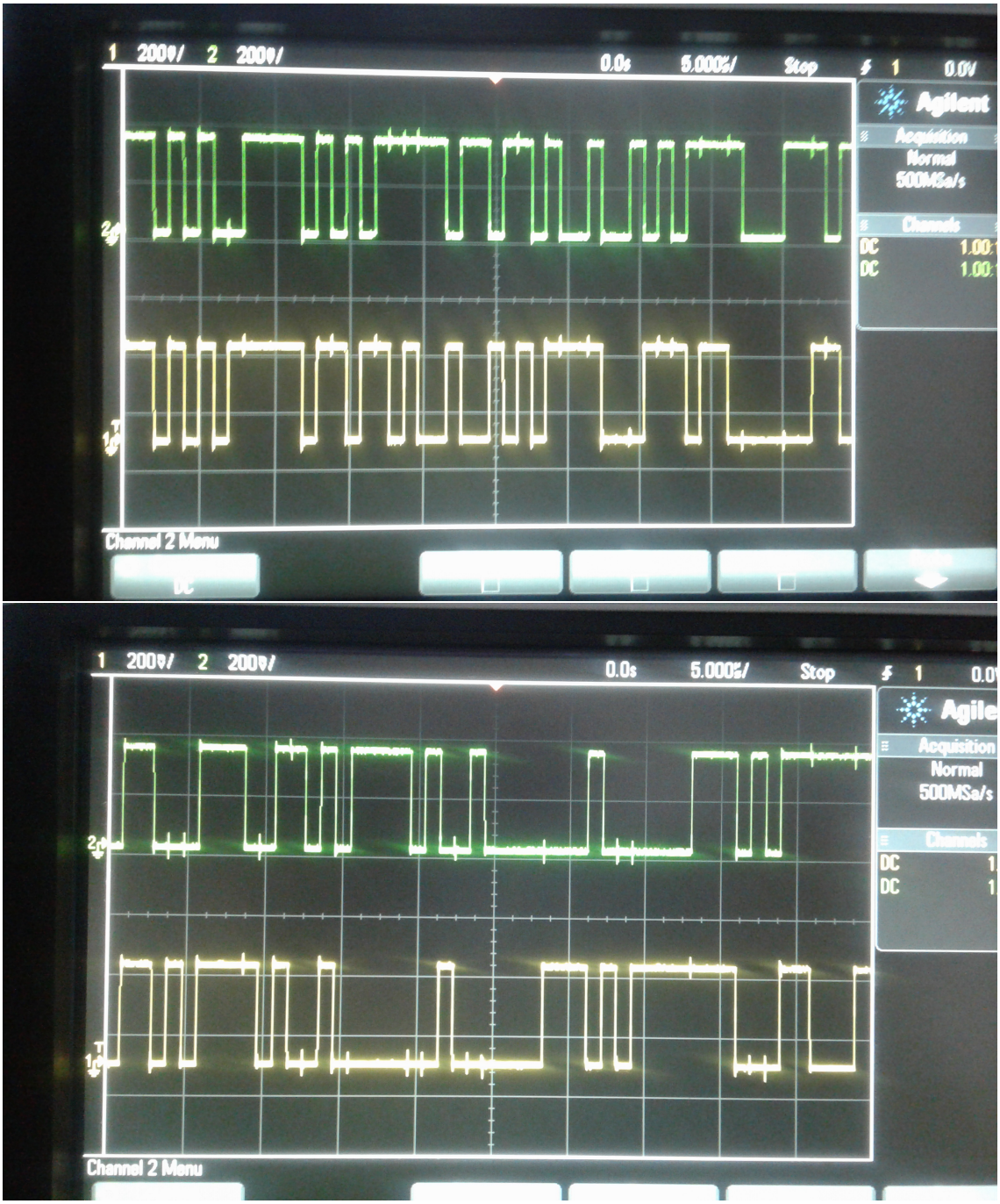

Figure 5.16: Chip delay for satellite ID1

It is given that for satellite-ID, there is 5 chip delay in LFSR-G2. That is between bit 10, and output $2 \oplus 6$. On Figure 5.16, we see that the bottom signal, "$2 \oplus 6$" is delayed by 5 chips compared to the signal on top, G2-bit10. Which verifies that the CA module is correct.

5.4 Suggested design for FPGA implementation

Based on stimulations and implementation results in previous sections, we can modify the codes in such way that allows the hole system to be implemented on FPGA. Figure 5.17 describes what changes we have to make. The local oscillator that generates the I and Q, has to be implemented on soft-core. Similar to the threshold detector. The soft-core care either be within the same FPGA, or as an external micro controller. However, putting soft-core in the FPGA may increase the power consumption, but it is the easiest.

All the signals involved in serial correlation must not be of type integer. We suggest a 16 bit word length to be used. However, this requires signal converting due to two main reasons. First all, the CA is naturally a single bit, so it must somehow be converted. The second reason is that, multiplying to vector with equal length, the result get double length. Therefore, the multiplication result has to be converted. Otherwise, this system seems to be the solution of how we can implement the whole data acquisition on FPGA.

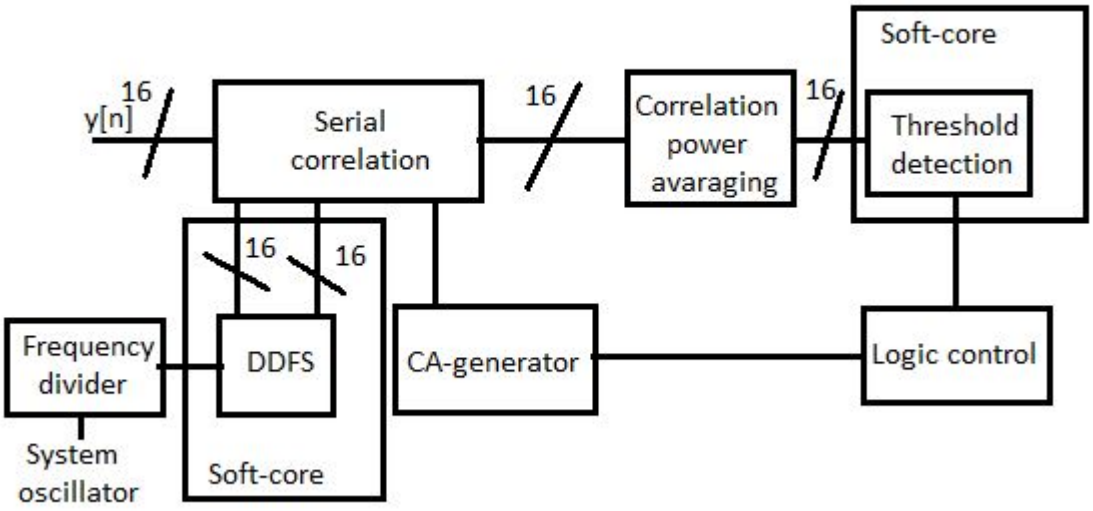

Figure 5.17: Suggested design of serial search acquisition

Chapter 6

Conclusion remarks

6.1 Discussion

The main objective in this project has been to develop an FPGA-based data acquisition system for GNSS receiver that is assumed to service the LEO-satellites. We have seen that the LEO-satellite requires an acquisition system that considers higher Doppler frequency than 25 KHz. Hence, various algorithms can be used. However, some are better than others. The delay-multiply is said to be the best algorithm for space-borne receiver. On the other hand, the fact that we had to implement our design on an FPGA, serial acquisition algorithm is chosen. This is because, it is simple to implement and has neither delays nor FFT which could be problematic to implement on FPGA. Through chapter 4, we designed a Simulink model for serial acquisition. the objective was to automatically generate VHDL code for the design. Unfortunately, only ranging code was generated. The remaining operations to achieve correlation power, are relatively simple for the design of VHDL code. Hence, in chapter 5, we combined VHDL from Simulink and our own code to design the serial acquisition. The detection module is also attempted to be designed, but according to our Simulink model, that involves division operation. However, division is FPGA is not that simple, therefore, due to time constraints, the detection module could not be designed completely.

6.2 Conclusion

The serial search algorithm can be implemented on FPGA. However some modules such as detection and local oscillator are hard to implement on hardcore. Therefore the solution is to use soft-core, which can be designed within FPGA itself or as a separate micro-controller. The Simulink model gives a perfect picture of how the system should work, but it does not necessarily generate VHDL. And if it does, the VHD codes are not automatically error-free, which needs to be verified before any further processing.

6.3 Future work

In this project, integer has been used to design the VHDL for a serial search acquisition. However, bit vector is much better for implementation, and so the complete system can be verified. Besides, design optimization to reduce the resources as well as power consumption has not been done. Therefore, future work with reference to this project is about power optimization. Apart from that, is to complete the the suggested design in section 5.4 can be completed for future work.

Bibliography

D. M. Akos & J. B. Y. Tsui (1996). 'Design and implementation of a direct digitization GPS receiver front end'. *IEEE Transactions on Microwave Theory and Techniques* **44**(12):2334–2339.

I. Ali, et al. (1998). 'Doppler characterization for LEO satellites'. *IEEE Transactions on Communications* **46**(3):309–313.

M. Anghileri, et al. (2012). 'Reduced navigation data for a fast first fix'. In *Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing, (NAVITEC), 2012 6th ESA Workshop on*, pp. 1–7.

M. Baracchi-Frei (2010). *Real-time GNSS software receiver optimized for general purpose microprocessors*. Ph.D. thesis, Université de Neuchâtel.

N. P. Bayendang (2015). *Nano-satellite GPS receiver design and Implementation: a software-to-firmware approach*. Ph.D. thesis, Cape Peninsula University of Technology.

N. M. BE (2010). *Variable dwell time verification strategies for CDMA acquisition with application to GPS signals*. Ph.D. thesis, Department of Electrical and Electronic Engineering, National University of Ireland.

P. Boto (2014). 'Analysis and Development of Algorithms for Fast Acquisition of Modern GNSS Signals'.

J. T. Curran, et al. (2010). 'Reducing Front-End Bandwidth May Improve Digital GNSS Receiver Performance'. *IEEE Transactions on Signal Processing* **58**(4):2399–2404.

L. Dong (2005). *IF GPS signal simulator development and verification*. National Library of Canada= Bibliothèque nationale du Canada.

S. Gleason & D. Gebre-Egziabher (2009). *GNSS applications and methods*. Artech House.

G. Hein, et al. (2006). 'Platforms for a future GNSS Receiver'. *Inside GNSS* **1**(2):56–62.

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

R. Lang, et al. (2016). 'Re-scaling and adaptive stochastic resonance as a tool for weak GNSS signal acquisition'. *Journal of Systems Engineering and Electronics* **27**(2):290–296.

J. Leclère, et al. (2013). 'Comparison framework of FPGA-based GNSS signals acquisition architectures'. *IEEE Transactions on Aerospace and Electronic Systems* **49**(3):1497–1518.

J. Leclère, et al. (2014). 'Acquisition of modern GNSS signals using a modified parallel code-phase search architecture'. *Signal Processing* **95**:177–191.

J. Leclre, et al. (2013). 'Modified parallel code-phase search for acquisition in presence of sign transition'. In *Localization and GNSS (ICL-GNSS), 2013 International Conference on*, pp. 1–6.

S. A. Malik, et al. (2009). 'Search Engine Trade-offs in FPGA-based GNSS Receiver Designs'. In *European Navigation Conference ENC-GNSS*.

D. Miralles, et al. (2014). 'Development of a Simulink Library for the Design, Testing and Simulation of Software Defined GPS Radios'. *Internal PUPR Technichal Report, Polytechnic University of Puerto Rico* pp. 6–7.

A. Miskiewicz, et al. (2009). 'System considerations and RF front-end design for integration of satellite navigation and mobile standards'. *Advances in Radio Science: ARS* **7**:151.

E. Murphy & C. Slattery (2004). 'Ask the application engineer33 all about direct digital synthesis'. *Analog Devices* **38**:1–5.

C. O'Driscoll (2007). 'Performance analysis of the parallel acquisition of weak GPS signals' .

V. Patel & P. Shukla (2011). 'Faster methods for GPS signal acquisition in frequency domain'. In *2011 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC)*, pp. 84–88.

M. Pini, et al. (2012). 'Estimation of Satellite-User Ranges Through GNSS Code Phase Measurements'. *GLOBAL NAVIGATION SATELLITE SYSTEMS* p. 107.

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

F. Schäfer, et al. (2005). 'The inter-agency space debris coordination committee (IADC) protection manual'. In *4th European Conference on Space Debris*, vol. 587, p. 39.

J. B.-Y. Tsui (2005). *Fundamentals of global positioning system receivers: a software approach*, vol. 173. John Wiley & Sons.

J. Wang, et al. (2016). 'Performance Analysis on Delay-Multiply Acquisition for Space-Borne GNSS Receivers'. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pp. 1–5.

J.-H. Won, et al. (2006). 'GNSS software defined radio'. *Inside GNSS* **1**(5):48–56.

L. Yang, et al. (2011). 'An innovative approach for atmospheric error mitigation using new GNSS signals'. *Journal of Navigation* **64**(S1):S211–S232.

Appendix A

Simlink modells

A.1 Serial coherent acquisition

C/A and NRZ formating

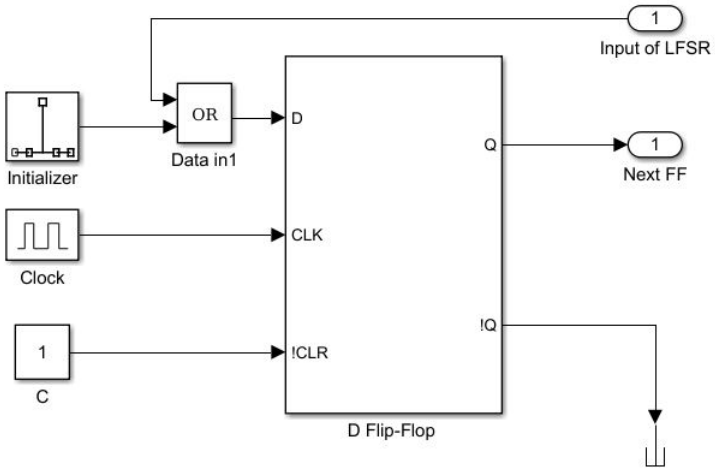

Figure A.1: D flip flop configuration

Figure A.2: Simulink PRN generator

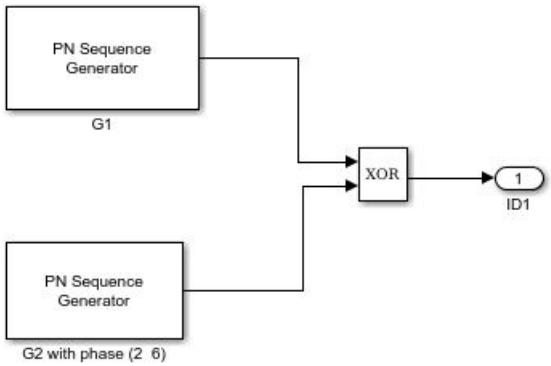

Figure A.3: PRN

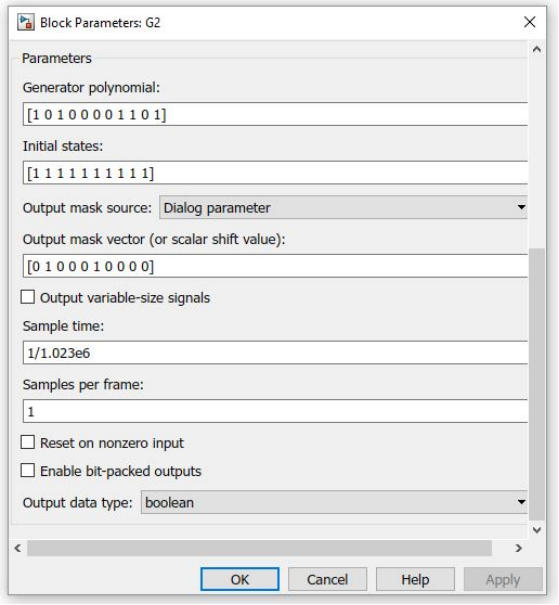

Figure A.4: G2 for ID1 configuration

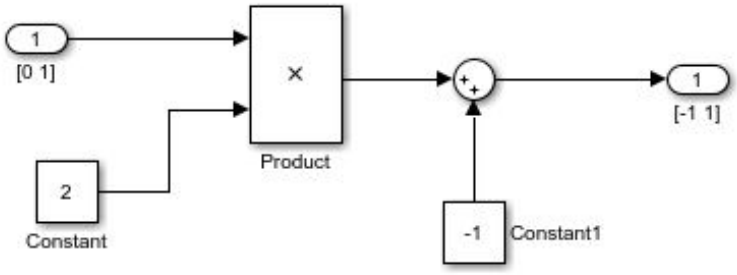

Figure A.5: NRZ

Oscillator

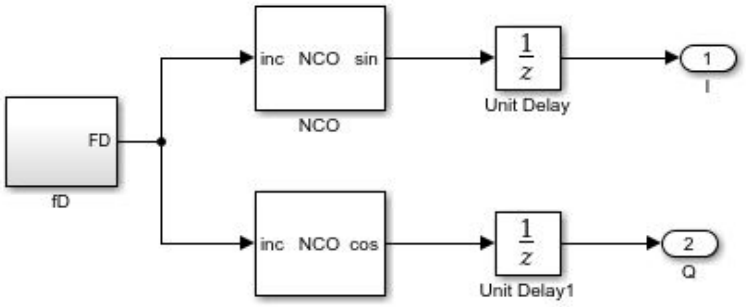

Figure A.6: Configuration of LO

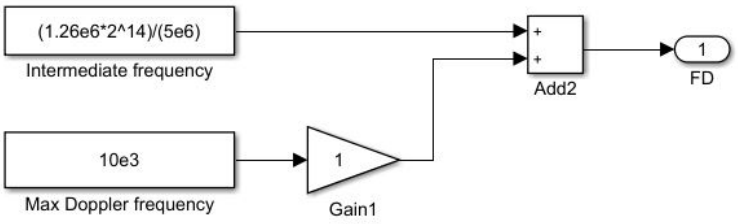

Figure A.7: NCO counter

A.2 Detector & control logic

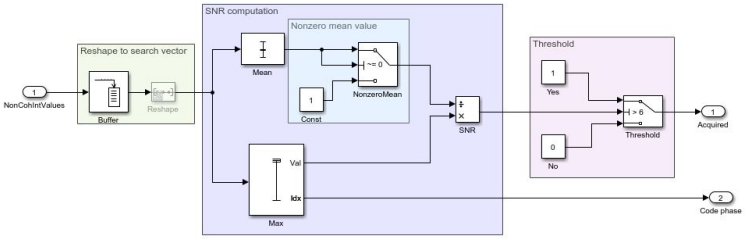

Figure A.8: Threshold detection Simulink model

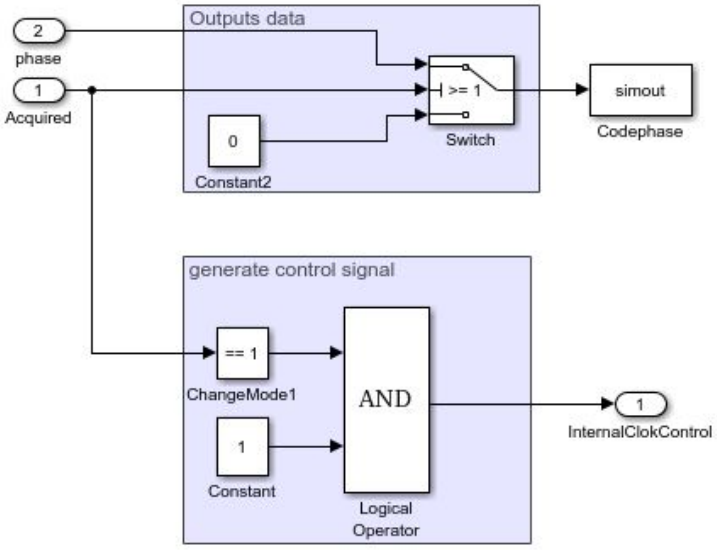

Figure A.9: the Simulink of a simple control logic

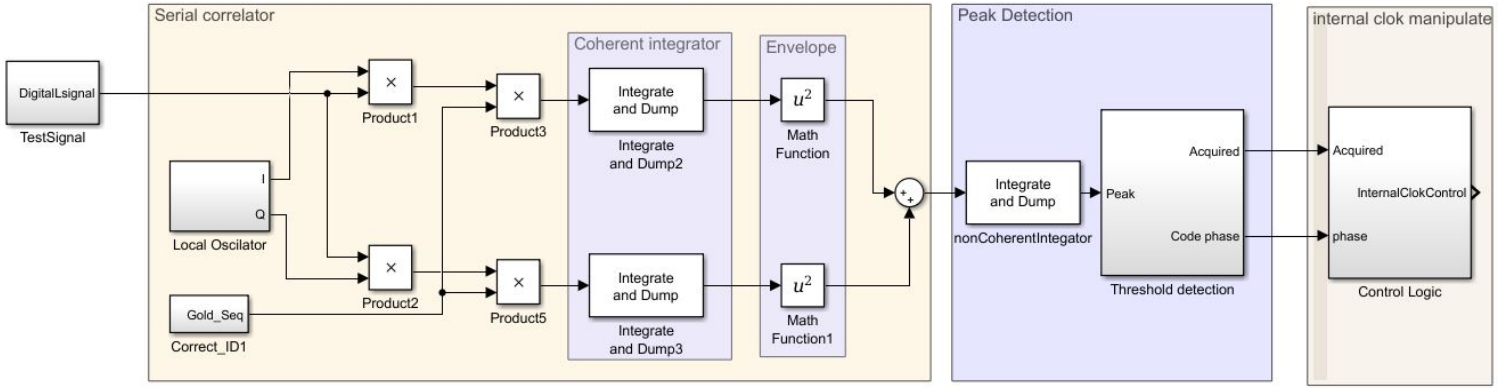

Figure A.10: The Simulink model of a serial search acquisition

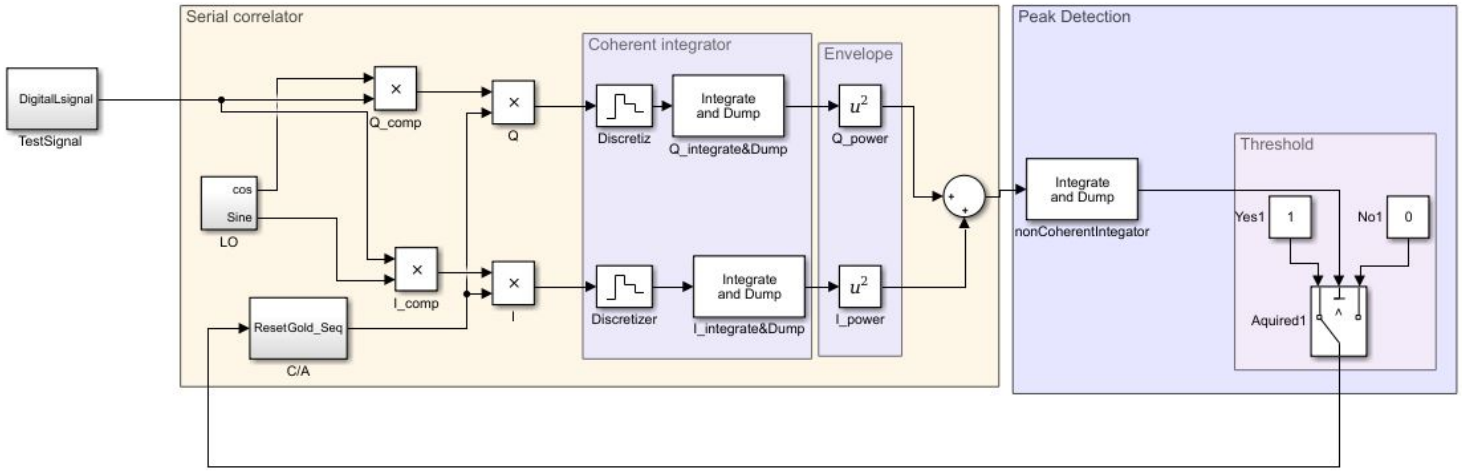

Figure A.11: Serial correlator with simplified detection and logical control

Appendix B

VHDL design

B.1 Full stimulation code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FullscaleStimulation is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enb : in  STD_LOGIC;
           Peak : out  integer
                       );
        end FullscaleStimulation;
architecture Structural of FullscaleStimulation is

component AquisitionSystem is
    Port ( L1CA_sig : in  integer;
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enb : in  STD_LOGIC;
           Peak : out  integer);
end component;

component L1CA is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enb : in  STD_LOGIC;
           L1 : out  integer
                       );
end component;
signal L1_to_test : integer;
begin
Testsignal: L1CA port map(clk => clk, reset => reset,
    enb => enb, L1 => L1_to_test);
Serial_acquisition: AquisitionSystem port map ( clk => clk,
reset => reset, enb => enb, L1CA_sig => L1_to_test, Peak => Peak);

end Structural;
```

L1CA structural code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

entity L1CA is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enb : in  STD_LOGIC;
           L1 : out  integer
                     );
end L1CA;

architecture Structural of L1CA is
component alpha IS
  PORT( clk                                 :   IN    std_logic;
        reset                               :   IN    std_logic;
        enb                                 :   IN    std_logic;
        PN1                                 :   OUT   std_logic             -- ufix1
              );
END component;

component clk_div is
port (
clk_NEXYS4 : in  std_logic;
rst       : in  std_logic;
LO_1260000 : out std_logic;
Data_50 : out std_logic;
CA_1023000   : out std_logic);
end component;

component NRZform is
    Port ( OneLevel : in  STD_LOGIC;
           TwoLevel : out  integer);
end component;

component Oscilator IS
  PORT( clk                                 :   IN    std_logic;
        reset                               :   IN    std_logic;
        clk_enable                          :   IN    std_logic;
       -- ce_out                             :   OUT   std_logic;
        Sin_out                             :   OUT   std_logic_vector(15 DOWNTO 0);
-- sfix16_En14
        Cos_out                             :   OUT   std_logic_vector(15 DOWNTO 0)
-- sfix16_En14
        );
END component;

signal Lcos : std_logic_vector(15 DOWNTO 0);
signal Lsin : std_logic_vector(15 DOWNTO 0);

signal Data_clk : std_logic;
```

```
signal CA_clk : std_logic;
signal LO_clk : std_logic;
signal CA_seq : std_logic;
signal Data_seq : std_logic;
signal Mod2SumOnelevel : std_logic;
signal Mod2sumTwolevel : integer;
signal carrierWaveInteger : integer;

begin
FreqSource:clk_div port map(clk_NEXYS4 => clk,rst=>reset,
Data_50 => Data_clk, CA_1023000 => CA_clk,
LO_1260000 => LO_clk);

Dataa: alpha port map(clk=>Data_clk, reset => reset,enb => enb,
PN1 => Data_seq);

RangingCode: alpha port map(clk=>CA_clk, reset => reset,enb => enb,
PN1 => CA_seq);

Mod2SumOnelevel <= Data_seq xor CA_seq;

NRZformat: NRZform port map (OneLevel => Mod2SumOnelevel,
TwoLevel=>Mod2SumTwolevel);
--Mod2sum<=Mod2SumOnelevel;

carrierwave: Oscilator port map (clk => LO_clk, reset => reset,
clk_enable => enb, Sin_out=> Lsin, Cos_out => Lcos);
carrierWaveInteger <= to_integer(signed(Lsin));
L1 <= carrierWaveInteger * Mod2sumTwolevel;
end Structural;
```

Acquisition system structural code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

entity AquisitionSystem is
    Port ( L1CA_sig : in  integer;
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enb : in  STD_LOGIC;
           Peak : out  integer);
end AquisitionSystem;

architecture Structural of AquisitionSystem is

component clk_div is
port (
clk_NEXYS4 : in  std_logic;
rst       : in  std_logic;
LO_1260000 : out std_logic;
Data_50 : out std_logic;
CA_1023000   : out std_logic);
end component;

component alpha IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        enb                               :   IN    std_logic;
        PN1                               :   OUT   std_logic             -- ufix1

                );
END component;

component Oscilator IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
        Sin_out                           :   OUT   std_logic_vector(15 DOWNTO 0);
-- sfix16_En14
        Cos_out                           :   OUT   std_logic_vector(15 DOWNTO 0)
-- sfix16_En14
        );
END component;

component NRZform is
    Port ( OneLevel : in  STD_LOGIC;
           TwoLevel : out  integer);
end component;

component Avarage is
    Port (
           I_cor : in integer;
```

```
                        Q_cor : integer;
                        Peak : out integer;
          clk : in  STD_LOGIC;
          reset : in  STD_LOGIC);
end component;

Component ThresholdAndControlLogic is
    Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          Peak : in  integer;
          SateliteState : out  STD_LOGIC);
end component;
signal LOcos : std_logic_vector(15 DOWNTO 0);
signal LOsin : std_logic_vector(15 DOWNTO 0);

signal Data_clk : std_logic;
signal CA_clk : std_logic;
signal LO_clk : std_logic;
signal CA       : std_logic;
signal CAnrz : integer;
signal Q_arm : integer;
signal I_arm : integer;
signal Iprod, Qprod : integer;

signal SatStat : std_logic;
signal ResetCA: std_logic;
signal PeakValue : integer;

begin
FreqSource:clk_div port map(clk_NEXYS4 => clk,rst=>reset,
Data_50 => Data_clk, CA_1023000 => CA_clk,
LO_1260000 => LO_clk);

LocalPRN: alpha port map(clk=>CA_clk, reset => ResetCA,enb => enb,
PN1 => CA);

NRZformat: NRZform port map (OneLevel => CA,
TwoLevel=>CAnrz);

LO: Oscilator port map (clk => LO_clk, reset => reset,
clk_enable => enb, Sin_out=> LOsin, Cos_out => LOcos);
corrPower: Avarage port map(clk=>CA_clk, reset => reset,I_cor => I_arm,
Q_cor => Q_arm, Peak => PeakValue);

Detection: ThresholdAndControlLogic port map(clk=>CA_clk, reset => reset,
Peak=>PeakValue,SateliteState => SatStat);
ResetCA <= reset xor SatStat;
peak <= PeakValue;
Iprod <= L1CA_sig * to_integer(signed(LOsin));
Qprod <= L1CA_sig * to_integer(signed(LOcos));
I_arm <= Iprod * CAnrz;
Q_arm <= Qprod * CAnrz;
end Structural;
```

correlator power Avarage

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_bit.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity Avarage is
    Port (
           I_cor : in integer;
                        Q_cor : integer;
                        Peak : out integer;
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC);
end Avarage;

architecture Behavioral of Avarage is
signal Q_accum : integer := 0;
signal Q_Env: integer := 0;
signal I_accum : integer := 0;
signal I_Env : integer := 0;
signal accumMax : integer := 1022;

begin
        CohIntAndDum:Process(clk,reset,accumMax,I_cor,Q_cor)
        variable accumNumber : integer := 0;
        variable I_sum : integer := 0;
        variable Q_sum : integer := 0;
                begin
                if reset = '0' then
                        I_sum := 0;
                        Q_sum := 0;
                        accumNumber := 0;

                elsif rising_edge(clk) then

                        I_sum := I_sum + I_cor;
                        Q_sum := Q_sum + Q_cor;
                        accumNumber := accumNumber + 1;

                        if accumNumber = (accumMax - 1) then
            I_accum <= I_sum;
            Q_accum <= Q_sum;
                I_sum := 0;
                        Q_sum := 0;
            accumNumber :=0;
                    end if;
                end if;
        end process;

        Env: process(I_accum,Q_accum) is
        begin
```

```
        I_Env <= I_accum * I_accum;
        Q_Env <= Q_accum * Q_accum;
        end process;

        Peak <= I_Env + Q_Env;

end Behavioral;
```

Detection and control logic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

entity ThresholdAndControlLogic is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           Peak : in  integer;
           SateliteState : out  STD_LOGIC);
end ThresholdAndControlLogic;

architecture Behavioral of ThresholdAndControlLogic is
type sat_state is (Aquired, NotAquired);
signal ControlLogic : sat_state := NotAquired;
constant Threshold : integer := 5000;
begin
process(clk,reset,Peak,ControlLogic)
begin
if reset = '0' then
ControlLogic <= NotAquired;
elsif rising_edge(clk) then
if Peak >= Threshold then
ControlLogic <= Aquired;
else
ControlLogic <= NotAquired;
end if;
end if;
end process;

process(ControlLogic)
begin
case ControlLogic is
 when Aquired => SateliteState <= '1';
 when others => sateliteState <= '0';
end case;
end process;
end Behavioral;
```

Frequency divider

```
--prescaler = ((clockSpeed/desiredClockSpeed)/2)[Hz]
--NEXYS4 run on 100MHz
--for C/A :1.023MHz => (100/1.023)/2 = 48875855.33Hz --> 48875855=(2E9C94F) in hex
--for Data : 50HZ => (100e6/50)/2    = 1000000 = F4240
--for FD=IF+fD: 1.26MHz+0KHz =>(100e6/(1.26e6+0)) = 39682539.68 = 25D81EB
-- if fD = 10KHZ, then                            = 39370078.74 = 258BD5E
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity clk_div is
port (
clk_NEXYS4 : in  std_logic;
rst        : in  std_logic;
LO_1260000 : out std_logic;
Data_50 : out std_logic;
CA_1023000   : out std_logic);
end clk_div;

architecture Behavioral of clk_div is

  signal presCA : unsigned(23 downto 0):=(others => '0');
  signal clkCA : std_logic:='0';
  signal presLO : unsigned(23 downto 0):=(others => '0');
  signal clkLO : std_logic:='0';
    signal presData : unsigned(23 downto 0):=(others => '0');
  signal clkData : std_logic:='0';
begin

  Div_clk : process (clk_NEXYS4, rst,presCA,presLO,presData)
  begin
    if rst = '0' then
     -- clkCA   <= '0';
      presCA   <= (others => '0');
                presLO <= (others => '0');
                presData <= (others => '0');
    elsif rising_edge(clk_NEXYS4) then
                  presCA <= presCA + "1";
                  presLO <= presLO + "1";
                  presData <= presData + "1";
                       if presCA = X"30" then -- 48.875855 in dec
          clkCA   <= not clkCA;
                    presCA   <= (others => '0');
                       end if;

                       if presLO = X"27" then -- 39.68254 in dec
          clkLO   <= not clkLO;
                    presLO   <= (others => '0');
                       end if;

                       if presData = X"F4240" then -- 1000000 in dec
```

```
       clkData   <= not clkData;
                 presData   <= (others => '0');
                      end if;
  end if;
end process;
 CA_1023000 <= clkCA;
      LO_1260000 <= clkLO;
      Data_50 <= clkData;
end Behavioral;
```

NRZ formarting

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NRZform is
    Port ( OneLevel : in  STD_LOGIC;
           TwoLevel : out  integer);
end NRZform;

architecture Behavioral of NRZform is
signal NRZ_level        : integer;
begin

  NRZ:process(Onelevel,NRZ_level)
  variable level:integer;
  begin
   if Onelevel = '1' then
        level := 1;
        elsif Onelevel = '0' then
        level := -1;
        end if;
        NRZ_level <= level;
        end process;
        TwoLevel <= NRZ_level;
end Behavioral;
```

DDFS

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

ENTITY Oscilator IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
       -- ce_out                           :   OUT   std_logic;
        Sin_out                           :   OUT   std_logic_vector(15 DOWNTO 0);
        Cos_out                           :   OUT   std_logic_vector(15 DOWNTO 0)
        );
END Oscilator;


ARCHITECTURE rtl OF Oscilator IS

  SIGNAL enb                              : std_logic;
  SIGNAL Sin_out1                         : signed(15 DOWNTO 0);
  SIGNAL Cos_out1                         : signed(15 DOWNTO 0);
  SIGNAL address_cnt                      : unsigned(2 DOWNTO 0);
  SIGNAL address_cnt_1                    : unsigned(2 DOWNTO 0);

BEGIN
  enb <= clk_enable;

  Sin_addrcnt_temp_process1 : PROCESS (clk, reset)
  BEGIN
    IF reset = '0' THEN
      address_cnt <= to_unsigned(0, 3);
    ELSIF clk'event AND clk = '1' THEN
      IF enb = '1' THEN
        IF address_cnt = to_unsigned(4, 3) THEN
          address_cnt <= to_unsigned(0, 3);
        ELSE
          address_cnt <= address_cnt + to_unsigned(1, 3);
        END IF;
      END IF;
    END IF;
  END PROCESS Sin_addrcnt_temp_process1;

  PROCESS(address_cnt)
  BEGIN
    CASE address_cnt IS
      WHEN "000" => Sin_out1 <= "0000000000000000";
      WHEN "001" => Sin_out1 <= "0011110011011110";
      WHEN "010" => Sin_out1 <= "0010010110011110";
      WHEN "011" => Sin_out1 <= "1101101001100010";
      WHEN "100" => Sin_out1 <= "1100001100100010";
      WHEN OTHERS => Sin_out1 <= "1100001100100010";
    END CASE;
```

```
  END PROCESS;


  Sin_out <= std_logic_vector(Sin_out1);

  Cos_addrcnt_temp_process2 : PROCESS (clk, reset)
  BEGIN
    IF reset = '0' THEN
      address_cnt_1 <= to_unsigned(0, 3);
    ELSIF clk'event AND clk = '1' THEN
      IF enb = '1' THEN
        IF address_cnt_1 = to_unsigned(4, 3) THEN
          address_cnt_1 <= to_unsigned(0, 3);
        ELSE
          address_cnt_1 <= address_cnt_1 + to_unsigned(1, 3);
        END IF;
      END IF;
    END IF;
  END PROCESS Cos_addrcnt_temp_process2;

  PROCESS(address_cnt_1)
  BEGIN
    CASE address_cnt_1 IS
      WHEN "000" => Cos_out1 <= "0000000000000000";
      WHEN "001" => Cos_out1 <= "1100001100100010";
      WHEN "010" => Cos_out1 <= "1101101001100010";
      WHEN "011" => Cos_out1 <= "0010010110011110";
      WHEN "100" => Cos_out1 <= "0011110011011110";
      WHEN OTHERS => Cos_out1 <= "0011110011011110";
    END CASE;
  END PROCESS;

  Cos_out <= std_logic_vector(Cos_out1);

END rtl;
```

Single bit to integer converter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

entity std_vector_conveter is
port(
                clk : in std_logic;
                bit_sig : in std_logic;
--outsig: out std_logic_vector(3 downto 0);
                integer_sig : out integer
                );
end std_vector_conveter;

architecture Behavioral of std_vector_conveter is
        type X is array(3 downto 0) of STD_LOGIC;
        signal vectorsig : x := (others => '0');
                begin
                        process(clk)
                                variable c : integer := 0;
                                variable siga : x := (others => '0');
                        begin
                                if rising_edge(clk) then
                                        siga(c) := bit_sig;
                                        c := c+1;
                                        if c = 4 then
          -- converting to vector of std_logic
                                                vectorsig <= siga;
                                                c := 0;
                                        end if;
                                end if;
                        end process;
        integer_sig <= to_integer(unsigned(vectorsig));
   --convecting to integer
end Behavioral;
```

C/A

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity CAsimulink is
Port (
reset : in  STD_LOGIC;
clk : in  STD_LOGIC;
en  : in std_logic;
CA : out  STD_LOGIC);
end CAsimulink;

architecture structural of CAsimulink is
component clk_divider is
port(clksys : in std_logic;
--resetsys : in std_logic;
slow_clk : out std_logic);
end component;

component alpha IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        enb                               :   IN    std_logic;
        PN1                               :   OUT   std_logic  -- ufix1
        );
END component;

--signal clk_in : std_logic;
signal clk_ca : std_logic;
--signal reset_sys: std_logic;
--signal enable : std_logic;

begin
slow: clk_divider port map(clksys => clk, slow_clk => clk_ca);
pr: alpha port map(clk => clk_ca,reset => reset, enb => en, PN1 => CA);

end structural;
-------------------------CA/Code with PRN phase for satelite_ID1--(2 xor 6)--
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.untitled_pkg.ALL;

ENTITY alpha IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        enb                               :   IN    std_logic;
        PN1                               :   OUT   std_logic  -- ufix1
        );
END alpha;
ARCHITECTURE rtl OF alpha IS

--phase selector for G2
constant s1s1: integer := 4; --satelitte1 sel1
```

```
constant s1s2: integer := 8; --sate1 sel2
  -- Signals
  SIGNAL G1_out1                          : unsigned(7 DOWNTO 0);  -- uint8
  SIGNAL G1_out1_is_not0                  : std_logic;
  SIGNAL G2_out1                          : unsigned(7 DOWNTO 0);  -- uint8
  SIGNAL G2_out1_is_not0                  : std_logic;
  SIGNAL alpha_out1                       : std_logic;  -- ufix1
  SIGNAL pn_reg                           : unsigned(9 DOWNTO 0);  -- ufix10
  SIGNAL pn_out                           : std_logic;
  SIGNAL pn_xorout                        : std_logic;
  SIGNAL pn_newvalue                      : vector_of_unsigned10(0 TO 1);
-- ufix10 [2]
  SIGNAL pn_value_shifted                 : unsigned(8 DOWNTO 0);  -- ufix9_E1
  SIGNAL pn_reg_1                         : unsigned(9 DOWNTO 0);  -- ufix10
  SIGNAL pn_out_1                         : std_logic;
  SIGNAL pn_xorout_1                      : std_logic;
  SIGNAL pn_newvalue_1                    : vector_of_unsigned10(0 TO 1);
-- ufix10 [2]
  SIGNAL pn_value_shifted_1               : unsigned(8 DOWNTO 0);  -- ufix9_E1

BEGIN

  pn_newvalue(0) <= pn_reg;

  pn_xorout <= pn_newvalue(0)(0) XOR pn_newvalue(0)(3);

  pn_value_shifted <= pn_newvalue(0)(9 DOWNTO 1);

  pn_newvalue(1) <= pn_xorout & pn_value_shifted;

  pn_out <= pn_newvalue(0)(0);


  PN_generation_temp_process1 : PROCESS (clk, reset)
  BEGIN
    IF reset = '0' THEN  -- reset is active high
      pn_reg <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
      IF enb = '1' THEN
        pn_reg <= pn_newvalue(1);
      END IF;
    END IF;
  END PROCESS PN_generation_temp_process1;

  G1_out1 <= resize("0" & pn_out, 8);

  G1_out1_is_not0 <= '1' WHEN G1_out1 /= to_unsigned(16#00#, 8) ELSE
      '0';

  pn_newvalue_1(0) <= pn_reg_1;

  pn_xorout_1 <= pn_newvalue_1(0)(0) XOR pn_newvalue_1(0)(2) XOR
  pn_newvalue_1(0)(3) XOR pn_newvalue_1(0)(8);
```

```
  pn_value_shifted_1 <= pn_newvalue_1(0)(9 DOWNTO 1);

  pn_newvalue_1(1) <= pn_xorout_1 & pn_value_shifted_1;

  --select phase

  pn_out_1 <= pn_newvalue_1(0)(s1s1) XOR pn_newvalue_1(0)(s1s2);

  PN_generation_temp_process2 : PROCESS (clk, reset)
  BEGIN
    IF reset = '0' THEN   --reset is active high
      pn_reg_1 <= to_unsigned(1023, 10);
    ELSIF clk'event AND clk = '1' THEN
      IF enb = '1' THEN
        pn_reg_1 <= pn_newvalue_1(1);
      END IF;
    END IF;
  END PROCESS PN_generation_temp_process2;

  G2_out1 <= resize("0" & pn_out_1, 8);

  G2_out1_is_not0 <= '1' WHEN G2_out1 /= to_unsigned(16#00#, 8) ELSE
      '0';

  alpha_out1 <= G1_out1_is_not0 XOR G2_out1_is_not0;

  PN1 <= alpha_out1;

END rtl;
```

B.2 Synthesizable

Integrator and dumper

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_bit.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity CoherentIntegrator is
Port ( Xin : in  std_logic_vector(4 downto 0);
intsum : inout std_logic_vector(4 downto 0);
Sum : out  std_logic_vector(9 downto 0);
clk : in  STD_LOGIC;
reset : in  STD_LOGIC);
end CoherentIntegrator;

architecture Behavioral of CoherentIntegrator is

constant count: integer := 1023;

begin
Process(clk,reset,Xin,intsum)
variable s:integer;
variable insum : std_logic_vector(4 downto 0);
--variable countv: std_logic_vector(7 downto 0):= (others => '0');
begin
if reset = '0' then
intsum <= (others => '0');
insum := (others => '0');
s:=0;
elsif rising_edge(clk) then
insum := insum + Xin;
s := s+1;
if s = count then
intsum <= insum;
insum := (others => '0');
s:=0;
end if;
end if;

end process;
process(intsum) is
begin
sum <= intsum*intsum;
end process;

end Behavioral;
```