Master Thesis

Modification of a FPGA-based GPS receiver for reflectometry applications (GNSS-R)

Miquel Àngel Ribot
Master of Science in Information and Communication Technologies

October 2011

Advisor:
Prof. Adriano J. Camps

Departament de Teoria del Senyal i Comunicacions
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Prof. Adriano Camps, who has guided, advised and even encouraged me throughout all the development of this thesis. Adriano, thanks for giving me this opportunity to work at your side, with your wisdom and tenacity you are a model for many of us.

I also want to thank Enric Valencia, Isaac Ramos, Xavier Bosch, Nereida Rodriguez and Hyuk Park, all members or former members the Remote Sensing Lab. You have dedicated much of your time helping me and I would like to let you know that I really appreciate that. I also learned a lot from you. To others, who have spent long hours in the lab with me, Pau Haro, Alberto Alsonso, Dani Pascual, thank you very much, you have done this a lot easier.

To all my friends, especially to Frau sisters for hosting me at your home every time that I needed, thank you very much.

Last but not least, thanks to my family who has always believed in me and without whose unconditional support getting here would have been impossible.

To all you, thanks for everything you have done for me and see you soon.


UPC
telecom
BCN

Abstract

Lack of frequent and global global observations from space is currently a limiting factor Earth observation missions. In recent years, as a low-cost alternative, Global Navigation Satellite System's signals Reflectometry (GNSS-R) has stood a potential powerful remote sensing technique. The existing research has shown that GNSS-R has the potential to give environmental scientist a low-cost, wide-coverage measurement network that will allow to derive geophysical parameters such as ocean altimetry, sea state or soil moisture. This data has the potential to greatly increase our knowledge of the Earth's environmental processes.

During the last ten years, the Remote Sensing Laboratory of the Department of Signal Theory and Communications at the Univeristat Politècnica de Catalunya, has worked on the design and implementation of the appropriate receivers in order to track and process this GNSS-R signals in real-time to avoid the storage of huge volumes of raw data. One of its most remarkable efforts is the Passive Advanced Unit for ocean monitoring (PAU) project.

In this work, the possibility of adapting an existing Global Position System (GPS) receiver for GNSS-R applications is explored. This GPS receiver is the Namuru-GPL a open source software receiver implemented for the Namuru development platform developed by the University of New South Wales Satellite Navigation and Positioning Laboratory (SNAP). A modified version of the Namuru-GPL has been implemented. This modified version of the receiver has been able to simultaneous track a C/A L1-band signal and a delayed version of it that simulated a reflected signal with the associated longer propagation path. In addition, the receiver has measured pseudorange differences with a tested resolution up 3 m with a single measurment in a controlled experimental scenario, thus validating the Namuru-GPL capabilities for GNSS-R altimetry applications.

In addition, a new Acquisition Module has been developed. This module dramatically reduces the Namuru-GPL receiver average acquisition time from a few minutes to 2.5 s approximately, thanks to implementing the parallel code acquisition method. Moreover, the Acquisition Module requires low hardware resources and generates Delay Doppler Maps (DDMs).

All the development process stages, including validation through testing of these proposed designs are summarized within this work.


UPC
telecom
BCN

Resumen

La falta de observaciones frecuentes y a escala global desde el espacio es actualmente un factor limitador de las misiones de observación de la Tierra. En los últimos años, y como alternativa al uso de constelaciones de satélites de propósito especifico y alto coste, la reflectometría con señales de oportunidad de los sistemas globales de navegación por satélite (GNSS-R) ha demostrado ser una técnica de teledetección con gran potencial. Las investigaciones realizadas hasta ahora demuestran que las técnicas GNSS-R poseen el potencial para obtener datos ambientales de alto interés científico a bajo coste y con una amplia cobertura de las mediciones realizadas. Dichas mediciones permitirían obtener o mejorar la medida de parámetros geofísicos importantes, como el estado del mar, altimetría, o la humedad del suelo. Una mejor medida de dichos parámetros tienen el potencial de aumentar considerablemente nuestro conocimiento de los procesos ambientales de la Tierra.

Durante los últimos diez años, el Remote Sesing Lab que pertenece al departamento de Teoría de la Señal y Comunicaciones (TSC) de la UPC, ha trabajado en el diseño y la implementación de receptores adecuados para adquirir y procesar señales GNSS-R en tiempo real para evitar el almacenamiento de enormes volúmenes de datos. Uno de los esfuerzos más notables es el proyecto "Passive Advanced Unit for Ocean monitoring" o proyecto PAU.

En este trabajo, se explora la posibilidad de adaptar un receptor GPS para a aplicaciones GNSS-R. Este receptor es el Namuru-GPL, un receptor software open source implementado sobre la plataforma de desarrollo Namuru, desarrollada por el University of New South Wales Satellite Navigation and Positioning Laboratory (SNAP). La versión modificada del receptor Namuru-GPL que se ha implementado, ha sido capaz de seguir una señal GPS C/A en la banda L1 y simultáneamente una versión retardada de la misma que simulaba ser una señal reflejada con un camino de propagación más largo. Además, el receptor ha sido capaz de medir diferencias de pseudorangos con una resolución máxima de hasta 3 m con una única medida, en un escenario experimental controlado, validando así las capacidades del Namuru-GPL para aplicaciones de altimetría mediante GNSS-R.

Además, se ha desarrollado un nuevo Módulo de Adquisición. Este módulo es capaz de reducir drásticamente el tiempo medio de adquisición del Namuru-GPL de unos pocos minutos a 2,5 s aproximadamente, gracias al método de adquisición en paralelo. Además, el Módulo de Adquisición necesita relativamente pocos recursos hardware y es capaz de generar "Delay Doppler Maps" (DDMs).

Todas las etapas del proceso de desarrollo de los diseños propuestos, incluida la validación experimental se encuentran resumidos en este trabajo.

Resum

La manca d'observacions freqüents i a escala global des de l'espai és actualment un factor limitador de les missions d'observació de la Terra. En els últims anys, i com a alternativa a l'ús de constel·lacions de satèl·lits de propòsit específic i alt cost, la reflectometría amb senyals d'oportunitat dels sistemes globals de navegació per satèl·lit (GNSS-R) ha demostrat ser una tècnica de teledetecció amb un alt potencial. Les investigacions realitzades fins ara demostren que les tècniques GNSS-R disposen del potencial per obtenir dades ambientals d'alt interès científic a baix cost i amb una àmplia cobertura de les mesures realitzades. Aquestes mesures permetrien obtenir o millorar la mesura de paràmetres geofísics importants, tals com l'estat de la mar, altimetria, o la humitat del sòl. Una millor mesura d'aquests paràmetres té el potencial d'augmentar considerablement el nostre coneixement dels processos ambientals de la Terra.

Durant els últims deu anys, el Remote Sesing Lab pertanyent al departament de Teoria del Senyal i Comunicacions (TSC) de la UPC, ha treballat en el disseny i la implementació de receptors adequats per rastrejar i processar senyals GNSS-R en temps real i evitar així haver d'emmagatzemar enormes volums de dades. Un dels seus esforços més rellevants és el projecte "Passive Advanced Unit for Ocean monitoring" o projecte PAU.

En aquest treball, s'explora la possibilitat d'adaptar un receptor GPS per aplicacions GNSS-R. Aquest receptor és el Namuru-GPL, un receptor open source implementat sobre la plataforma de desenvolupament Namuru, desenvolupada pel University of New South Wales Satellite Navigation and Positioning Laboratory (SNAP). La versió modificada del receptor Namuru-GPL implementada, ha estat capaç de seguir un senyal GPS C/A a la banda L1 i simultàniament una versió retardada del mateix que simulava ser un senyal reflectit amb un camí de propagació més llarg. A més, el receptor ha estat capaç de mesurar diferències de pseudorangs amb una resolució màxima de fins a 3 m en una única mesura a un escenari experimental controlat, validant així les capacitats del Namuru-GPL per a possibles aplicacions d'altimetria mitjançant GNSS-R.

També s'ha desenvolupat un nou Mòdul d'Adquisició. Aquest mòdul és capaç de reduir dràsticament el temps mitjà d'adquisició del Namuru-GPL d'uns pocs minuts a 2,5 s aproximadament, gràcies al mètode d'adquisició en paral·lel. A més, el Mòdul d'Adquisició consumeix relativament pocs recursos hardware i és capaç de generar "Delay Doppler Maps" (DDMs).

Totes les etapes del procés de desenvolupament dels dissenys proposats, inclosa la seva validació experimental es troben resumits en aquest treball.


UPC
telecom
BCN

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Remote Sensing Using Reflected GNSS Signals

Lack of frequent and global observations from space is currently a limiting factor in many Earth Observation (EO) missions. Two potential remote sensing techniques have been proposed nowadays:

1. The use specific purpose satellite constellations,
2. The use of Global Navigation Satellite Signals (GNSS) as signals of opportunity (where no transmitter is required).

It is well known that GNSS signals scattered from ocean, land and ice are affected by the reflecting surface, and the changes induced by the surface can be observed. Understanding what exactly is being sensed and to what accuracy is driving the nowadays and future applications of the GNSS-R technique. However, the existing research has shown that GNSS remote sensing has the potential to give environmental scientists a low-cost, wide-coverage measurement network that will greatly increase the knowledge of the Earth's environmental processes. For accomplishing those objectives, it is required to achieve a high accuracy measurements, which still is considerably challenging. However, modest, but useful applications such as dangerous sea sensing and coarse surface characterization have already been demonstrated.

Existing remote sensing instruments are often grouped based on their operational frequency rage and whether they utilize passive or active radiation. Passive sensors rely on natural radiance of the Earth, due to an illumination source such as the Sun.

Active microwave and radar instruments as scatterometers, synthetic aperture radars (SAR) and altimeters that work at frequencies between 1 and 30 GHz are all able to effectively penetrate the atmosphere and make measurements in diverse weather conditions. The GNSS signals reside in this frequency range, specially at frequencies commonly called L-Band, which are capable of penetrating cloud cover. Even though GNSS signals are an active source, when used for remote sensing, ubiquitous GNSS transmission are often considered as part of the environment. Thus GNSS remote sensing is often called a passive, bistatic technique.

Using Earth-reflected GNSS signals as an alternative technique for ocean altimetry was first proposed by Martin-Neira (ESA-ESTEC) in 1993 [1]. The technique, designated PARIS (Passive Reflectometry and Interferometry System), has was proven first through experiments over a pond at ESTEC, then from the Zeeland bridge (NL) and later from an aircraft flying over the Mediterranean near Barcelona. Later, more GNSS-R techniques were proposed for wind speed determination and for obtaining sea state corrections in order to recover the sea surface salinity by using L-Band microwaves [5, 6, 3, 4].

Many GNSS systems are available or will be in the near future such as GLONAS, COMPASS, GALILEO, etc. However it is the US Global Positioning System (GPS) the one that today is fully deployed and operational, and thus it has the widest acceptance for GNSS-R applications.

1.1.1 GNSS-R principle and Delay Doppler Map generation

The basic principle behind GNSS-R is simple. When an electromagnetic wave scatters over the sea surface, the scattered signal changes its polarization. For GPS signal, that is transmitted with right hand circular polarization (RHCP), the resulting scattered signal polarization changes to left hand circular polarization (LHCP). Typically, the received signal comes from a single specular reflection point, determined by the shortest path between the transmitting GPS satellite and the receiver. However, when the sea is not in calm, the scattered signals come from a wider region (known as "glistening zone") that enlarges with increasing sea state, in a similar manner as the Sun reflecting over the sea.

When observing the GNSS reflected signals, two points over the sea surface correspond to the same delay and Doppler coordinates, with minimum delay corresponding to the specular reflection point.

These received signals can be processed by correlating it with local replicas of the GPS code generated with different time delays and frequency shifts. The result of this technique is a 2-D function or waveform that relates to the scattered power as a function of the signal time

delay and frequency Doppler shift [2]. This waveform is called Delay-Doppler Map (DDM). Thus, the DDM is the square of the absolute value of the correlations of the reflected GNSS signals (typically GPS) with local replicas of the transmitted signal, but shifted in delay and Doppler, and is given by [3]:

$$|\mathrm{DDM}(\Delta\tau, \Delta f_d)|^2 = T_i^2 \iint \frac{|R|^2 D^2(\vec{\rho})\Lambda^2(\tau)\,|S(\Delta f_d|^2}{4R_0^2(\vec{\rho})R^2(\vec{\rho})} \frac{q^4(\vec{\rho})}{q_z^4(\vec{\rho})} P_v\left(-\frac{\vec{q}_\perp}{q_z}\right) d^2\vec{\rho}, \tag{1.1}$$

where T_i is the integration time, R is the Fresnel reflection coefficient, D is the directivity of the receiving antenna, $\chi(\Delta\tau, \Delta f_d) \approx \Lambda(\Delta\tau)\cdot S(\Delta f_d)$ is the Woodward ambiguity function, that can be approximated by the product of a triangle function $\Lambda(\Delta\tau) = (1\,|\Delta\tau|\,/T_c)$ if $|\Delta\tau| \leq T_c$, being T_c the chip period, and a sinc function:

$$S(\Delta f) = \exp(-j\pi\Delta f\, T_i)\sin(\pi\Delta f\, T_i)/(\pi\Delta f\, T_i), \tag{1.2}$$

$R_0(\vec{\rho})$ and $R(\vec{\rho})$ are the distances between the transmitter and receiver to the scattering point, q is the amplitude of the scattering vector $\vec{q} \triangleq k(\hat{n} - \hat{m})$ where k is the wavenumber, $\hat{n}$ and $\hat{m}$ are the unit vector of the incident and scattered waves, respectively, q_z is the z component of $\vec{q}$, and $\vec{q_\perp}$ is the perpendicular component, and P_v is the sea surface slopes probability density function, and the integral is performed over the whole glistering zone.

In [5, 6] Marchan et. al. found that the normalized DDM's volume (peak amplitude equal to 1) above a given threshold (Figure 1.1) can provide a measurement of the area over which the GNSS signals are scattered. This area can be related to geophysical variables such as the sea surface roughness, with no use of any intermediate model, either numerical to compute the scattering or the sea surface spectrum.

In the rest of the document, the DDMs will be computed by correlating the received scattered signal $s(t)$ with a local replica of the GPS C/A code $a(t)$ for several values of the delay (τ), also often referred as code phase or code phase delay, and frequency Doppler offsets or shifts (f_D):

$$\mathrm{DDM}(\tau, f_D) = \int_0^{T_c} s(t)a(t+\tau)\exp(-j2\pi(f_{L1} + f_D)t)\,dt, \tag{1.3}$$

where (τ, f_D) are delay and Doppler coordinates, T_c is the coherent integration time and $f_{L1} = 1.57542\,\mathrm{GHz}$ is the GPS L1 signal carrier frequency.

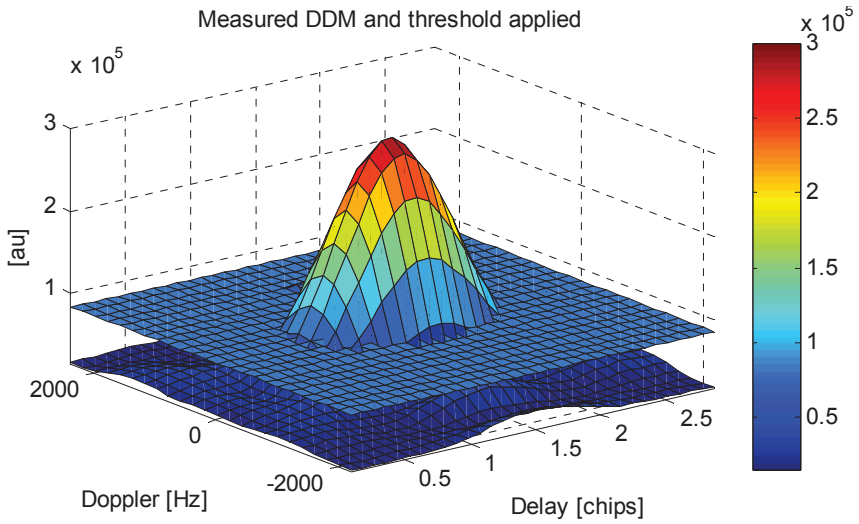


Figure 1.1: DDM example with a threshold applied [2].

1.1.2 GNSS-R at the UPC Remote Sensing Lab: The Passive Advanced Unit (PAU) Project

The Passive Remote Sensing Group belongs to the Remote Sensing Laboratory (RSLab) from the "Departament de Teoria del Senyal i Comunicacions" (TSC), Universitat Politècnica de Catalunya (UPC), in Barcelona, Spain. Its main research lines are related to the European Space Agency (ESA) SMOS mission [8] (such as the MIRAS instrument performance analysis, calibration and reconstruction or field experiments for sea and soil emission determination) and the Passive Advance Unit project.

Based on previous research experiences, the Passive Advanced Unit for ocean monitoring (PAU) project was proposed in 2003. This project goal was to test the feasibility of using GNSS-R over the sea surface to make sea state measurements and, jointly with IR observations to obtain the Sea Surface Temperature (SST), perform the corrections of the L-band brightness temperature [7].

PAU has also been a test bed of new technological demonstrators such as real aperture radiometers with digital beamforming and polarization synthesis, and fully-digital synthetic aperture radiometers etc. In the PAU system a suite of three different instruments operate in a synergetic way:

1. PAU-RAD: a new type of pseudo-correlation L-band radiometer to measure the brightness temperature of the sea surface,

2. PAU-GNSS-R: a GPS-reflectometer to measure the sea state using GPS L1 signal and C/A code, and
3. PAU-IR: an IR radiometer to measure the sea surface temperature.

A set of PAU system prototypes has been developed. These instruments are considered as technology demonstrators where the input signals are the same: the L1-GPS band (PAU/RAD and PAU/GNSS-R share the same RF front-end). This is not a critical issue, since, due to the scattering on the sea surface, the scattered GPS signal is at leas 23 dB below the thermal noise signal. Thanks to the 30.1 dB correlation gain, PAU/GNSS-R can detect the GPS signal when the correct C/A code is applied, and the error introduced in PAU-RAD observables is negligible or it can be kept under control.

Some of the previously mentioned PAU prototypes have been implemented using Full Programmable Gate Array (FPGA) hardware to perform real-time signal processing tasks. A brief description of the PAU prototypes built over the last decade is presented next:

- PAU-Real Aperture instrument:

  With a 4×4 element array with digital beamforming and polarization synthesis, it integrates the PAU/RAD and PAU/GNSS-R receiver types in a single hardware unit. It uses an innovative pseudo-correlation radiometer topology to avoid the classical input switch in a Dicke radiometer [9].

- PAU-Synthetic Aperture instrument:

  With a larger number of receiver elements (25), its main goal was testing new techniques, technologies and algorithms that eventually would be used in future remote sensing space missions, such as future SMOS missions [7].

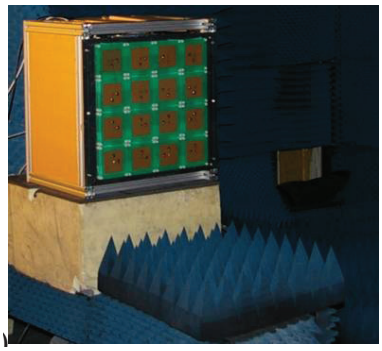
- griPAU:

  Developed for ground tests and algorithm developments during the PAU-Real and Synthetic Aperture instruments development. griPAU was a fully automated evolution of a previous instrument, the PAU-OR. The signal processor implemented within the griPAU instrument in 2007 was the world's first reflectometer that was able to compute DDM in real-time, and at the same time, perform programmable coherent and incoherent integration without the need of post-processing [2].

- PAU-ORA:

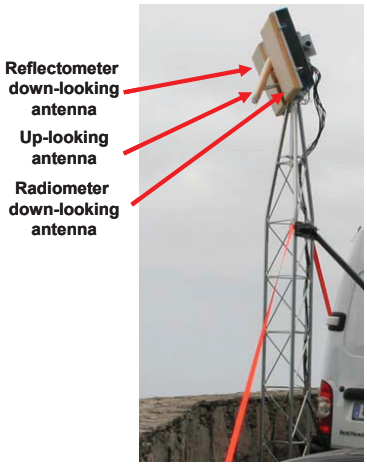
  A lighter version of PAU-OR for aircraft operations from a remote controlled plane.

Some pictures of this instruments are shown in Figure 1.2.

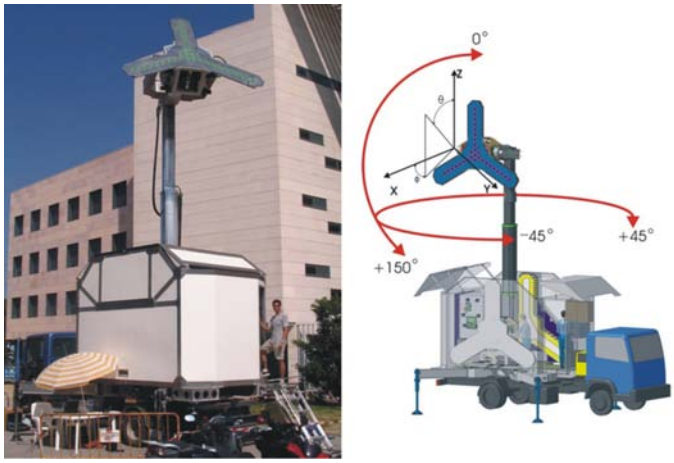


(a) PAU-Real Aperture without radome in UPC anechoic chamber for testing beamforming performance.

(b) PAU–One Receiver Airborne instrument block mounted on the bay of a Remote Control aircraft.



(c) griPAU instrument mounted on a 6 m tower with automatic antenna positioning.



(d) PAU-Synthetic Aperture. View of the whole instrument installed on truck with 8 m robotic mast.

Figure 1.2: PAU project instruments

1.2 Thesis objectives and motivation

The Passive Remote Sensing laboratory acquired a Namuru V2 GNSS development platform [48]. The Namuru is intended to be a fast development platform for GPS L1-band applications. This platform is mainly composed by the Namuru V2 GNSS board and a set of software tools. The board has 2 RF front-ends directly connected to an FPGA. This FPGA is by default programmed with a open source GPS receiver design, the Namuru-GPL, ported to for a FPGA platform from the original A. Greenberg's software [50].

The Namuru development board can be used as a first test bench for developing some parts of GNSS-R applications. The Namuru platform supposes a simple solution especially for the first development phases of new instruments or algorithm testing. Having 2 RF front-ends already integrated with an FPGA opens an interesting set of possibilities. The FPGA integrated within the Namuru board has limited resources, which doesn't make it suitable for applications that require an intensive and complex real-time signal processing. However, the Namuru may be appropriate for developing low-cost GNSS-R altimetry applications.

Thus, in order to explore the Namuru possibilities for GNSS-R two main objectives were defined:

1. The initial objective was **to study the Namuru GNSS development board possibilities for GNSS-R applications**. Starting from a default GPS receiver design, it was necessary **to implement hardware and software modifications to allow receiving signals with both of its L1 GPS RF front-ends**. This will allow to track direct and scattered signals coming from the same GPS satellite avoiding possible synchronism errors that can appear when multiple independent GPS receivers are used. The receiver needs to provide a fine enough estimation of the propagation delay difference between the direct and the scattered signals.

2. However, while developing the required modifications to achieve this first objective, another main objective was detected. The available GPS receiver, implemented within the Namuru development board, was too slow performing the GPS signal acquisition. This drawback limited the range of possible GNSS-R applications and scenarios (for example satellite on-board GNSS-R applications) that usually require low acquisition times.

   Because of that, the second main objective has been **to develop a new Acquisition Module with considerably better performance** that could be integrated within the existing Namuru GPS receiver. This Acquisition Module preferably will use the

parallel code acquisition method. Using this acquisition method the Acquisition Module should also be able to generate DDMs for the satellite signal that it is searching for.

For both objectives, it will be necessary an verification process through simulation and laboratory tests. Designing and developing these required tests are implicit objectives.

1.3 Outline

This document's structure has been chosen to allow a reader to understand as much as it is possible the design and implementation challenges found during the development of this thesis, as well as the challenges faced.

The document first introduces the GPS signal processing concepts necessary to understand the proposed hardware designs needed to fulfill the 2 previously described project main objectives. Then, a detailed description of this hardware designs and implementation is provided. Finally a description of a set of the tests performed to verify and characterize these hardware capabilities is presented. Together with these test descriptions, the results obtained are also presented.

Thus, this master's thesis has been organized in 9 chapters, with this chapter as introduction, as follows:

In Chapter 2, the main GPS signal's components and properties are introduced. It is intended to provide a basic GPS signal overview necessary understand the following chapters. It focuses on the GPS system architecture, the C/A signal in L1 band features, and finishes describing a basic GPS receiver operation. Very related with Chapter 2, Chapter 3 describes the C/A signal acquisition and tracking procedures, especially focusing on the former one. The parallel code acquisition strategy is introduced during this chapter.

Chapter 4 describes the Namuru GNSS development platform. It first opens with a description of a generic GPS RF front-end. Then, both parts of the development platform, an FPGA based hardware and the required software tools, are described in detail.

Chapter 5 presents the Namuru-GPL receiver solution. It is open source software receiver that has been ported to the Namuru GNSS development platform. Thus, some parts, such as the baseband processor unit, are hardware implemented, while other functionalities are performed by an embedded soft-core processor. The second half of this chapter introduces the modifications performed over the Namuru-GPL for making it capable of using both of its 2 front-ends for possible GNSS-R applications. This modification includes changes in

the receiver design and in its software. These modifications are the first main contribution presented in this document.

Chapter 6 describes in very detail the proposed Acquisition Module development, which includes design, implementation and verification stages. This acquisition module uses a parallel code acquisition strategy an is intended to provide a huge reduction in the required acquisition time compared to the Namuru-GPL default serial search acquisition. This Acquisition Module development is the second thesis' main contribution.

Chapter 7 is devoted to describe the implemented test setups to verify and characterize the Namuru-GPL modifications and the proposed Acquisition Module. The results of these tests are also presented in this chapter.

Finally, Chapter 8 summarizes the thesis main ideas and contributions, presents some conclusions on the work done and offers some future work guidelines.

Each chapter has its own introduction and summary and conclusions sections.

Chapter 2

Global Positioning System (GPS) signal overview

This chapter introduces the main components and properties of the GPS signal. The chapter focuses mainly in the civilian signals (C/A) in the L1 band, but it also includes some information about the non-civilian components. The first section offers an overview of the GPS system architecture and their main components. Next section includes a functional description of a basic GPS receiver. The third section presents the principle that makes the global positioning possible. Finally the last section reviews the main GPS signal aspects, paying special attention on the signal C/A code specs, the C/A code generation process and the navigation data included within the signal.

The chapter's goal is to provide a basic GPS signal overview. This overview is necessary to understand the following chapters. Those chapters will focus on more specific aspects of GPS receivers, C/A signal acquisition and their implementation for the thesis objectives.

2.1 GPS Architecture overview

The American Global Positioning System (GPS), also referred to as NAVSTAR GPS - Navigation Signal Timing and Ranging Global Positioning System, by the US Army, was designed to provide a 3D positioning available anywhere and anytime on the Earth surface.

The architecture of GPS system can be divided in three components (see Figure 2.1):

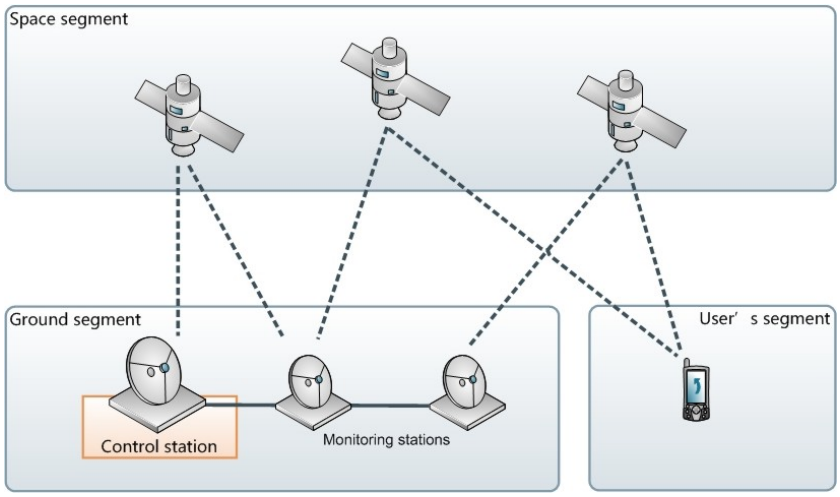


Figure 2.1: GPS architecture segments and their components

1. **The space segment** consists of a set of 32 Medium Earth Orbit (MEO) satellites located in near circular orbits with an average radius of 26560 km, moving at a mean speed of 3.87 km/s, which results in a 12 hours rotation period approximately.

   The satellite constellation is distributed in six orbital planes spaced 60^{o} through the Equator with and inclination of 55^{o}.

   This configuration allows that at least four satellites will be simultaneously observed at a given place and moment. As it is explained later, this is the minimum number of satellites needed by the receiver to calculate a valid position. In order to ensure the service even when one satellite fails, it is necessary to consider a minimum of five visible satellites. Satellite's visibility depends on the latitude, however, there is always a minimum of 5 satellites in view, and for more than 80% of the time this number grows to 7 [10]. The current constellation of satellites is planned to be in operation until 2018-2010 [11]. Each satellite has a very precise on-board atomic clock. The clocks are synchronized between the satellites and they represent one of the most important parts of the system. They are used to derive the GPS system time, as well as to generate the fundamental frequency used to generate the carrier waves of the transmitted signal.

2. **The control segment** consists of a number of monitor stations equipped with precise atomic time standards. Monitor stations measure the position of the visible satellites and relay the information to control stations. Control stations are responsible for sending correction to the GPS satellite when needed. Ground control can upload new data at every moment, but most of the time this is done once every 24 hours [12].

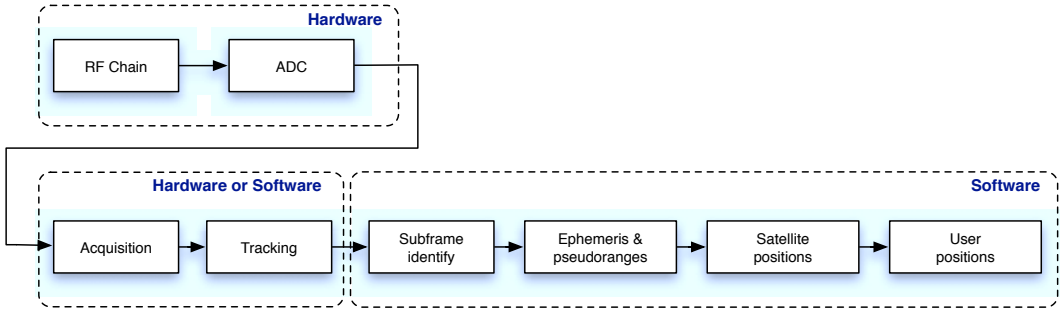


Figure 2.2: A basic GPS receiver block scheme. Note that some receiver parts can be implemented either by hardware or software

3. **The user segment** is divided into two categories; military and civil users. Military users are authorized by the US Department of Defense (DoD) and can access the encrypted GPS signals, which have a significantly improved accuracy. The service provided to civil users is commonly referred as Standard Positioning Service (SPS) and is detailed in [11]. In the current GPS system, civil users are only allowed to access the so-called Coarse Acquisition (C/A) codes.

2.2 A Basic GPS Receiver

Figure 2.2 shows block diagram of a basic GPS receiver. The signals transmitted from the GPS satellites are received from the antenna. Through the radio frequency (RF) chain the input signal is amplified to a proper amplitude and the frequency is down-converted to a desired output frequency.

An analog-to-digital converter (ADC) is used to digitize the output signal. After the signal is digitized, the Acquisition process starts. Acquisition is performed in order to find the signal of a certain satellite. The tracking process coming once acquisition has been successfully performed is used to find the phase transition of the navigation data. In a conventional receiver, the acquisition and tracking are performed by hardware. However, new software approaches for GPS receivers perform this processes by software [18].

From the navigation data phase transition the subframes and navigation data can be obtained. Ephemeris data and pseudoranges can be obtained from the navigation data. The ephemeris data are used to determine the satellite positions. Finally, the receiver position can be computed for the satellite position and the pseudoranges obtained.

2.3 Positioning principle

The positioning concept used by the GPS system is based on the propagation of radio waves through space. The distance ρ traveled by a radio wave is given by the propagation delay Δt multiplied with the speed of light c. For the k^{th} satellite this is

$$\rho^k = c\Delta t^k \text{ [m]}. \tag{2.1}$$

Each satellite sends navigation data containing the exact GPS system time at which the data was transmitted in order to calculate the propagation delay to the receiver. This is then compared with a synchronized reference clock in the receiver, whereby the delay and hence the range to the satellite can be found. The navigation data from satellites also contains necessary information in order to calculate the satellites position.

2.3.1 Pseudoranges measurement

Every satellite sends a signal at a certain time t_{si}. The receiver will receive the signal at a later time t_u. The distance between the user and the satellite i is

$$\rho_{iT} = c(t_u - t_{si}), \tag{2.2}$$

where c is the speed of light, ρ_{iT} is referred to as the true value of the pseudorange from user to satellite i, t_{si} is referred to as the true time transmission from satellite i, t_u is the true time of reception. In practice is almost impossible to obtain the correct time from the satellite or the user. The actual satellite clock time t'_{si} and actual user clock time t'_u are related to the true time as:

$$\begin{aligned} t'_{si} &= t_{si} + \Delta b_i, \\ t'_u &= t_u + b_{ut}, \end{aligned} \tag{2.3}$$

where Δb_i is the satellite clock error and b_{ut} is the user clock bias error. Besides the clock error, there are other factors affecting the pseudorange measurement. The measured pseudorange ρ_i can be written as

$$\rho_i = \rho_{iT} + \Delta D_i - c(\Delta b_i - b_{ut}) + c(\Delta T_i + \Delta I_i + v_i + \Delta v_i), \tag{2.4}$$

where ΔD_i is the satellite position error effect on range, ΔT_i is the tropospheric delay error, ΔI_i is the ionospheric delay error, v_i is the receiver measurement noise error, Δv_i is the relativistic time correction.

While some of these errors can be corrected, the user clock error cannot be corrected through received information. Thus, it will remain as an unknown. As a result we obtain

$$\begin{aligned}
\rho_1 &= \sqrt{(x_1 - x_u)^2 + (y_1 - y_u)^2 + (z_1 - z_u)^2} + b_u, \\
\rho_2 &= \sqrt{(x_2 - x_u)^2 + (y_2 - y_u)^2 + (z_2 - z_u)^2} + b_u, \\
\rho_3 &= \sqrt{(x_3 - x_u)^2 + (y_3 - y_u)^2 + (z_3 - z_u)^2} + b_u, \\
\rho_4 &= \sqrt{(x_4 - x_u)^2 + (y_4 - y_u)^2 + (z_4 - z_u)^2} + b_u,
\end{aligned} \tag{2.5}$$

where b_u is the user clock bias error expressed in distance, which is related to the quantity b_{ut} by $b_u = cb_{ut}$. As it can be observed, four equations are needed to solve for four unknowns (x_u, y_u, z_u and b_u). Thus, in a GPS receiver, a minimum of four satellites is required to solve the user position.

2.4 GPS Signal and Data

The GPS signal structure was designed to allow multiple transmitters using the same frequency and to have a certain tolerance to multipath and jamming. This conception results in a low power spectral density signal in order to avoid mutual interference with other microwave systems, and to allow ionospheric delay estimation for range determination correction. More details of reasons of why these signals were selected during the GPS system design can be found in [10].

There are basically two types of signals: the coarse (or clear)/acquisition (C/A) and the precision (P) codes. The actual P code is not directly transmitted by the satellite, but is modified by a Y code, which is often referred to as the P(Y) code. The P(Y) code is not available to civilian users and represents the before mentioned encrypted signal corresponding to military users segment of the GPS system. Thus, the P(Y) code is classified. Therefore, the discussion will be focused on the C/A code.

Signal Components: The GPS signals are transmitted on two radio frequencies in the UHF band. These frequencies are referred to as L1 and L2 and are derived from a common frequency, $f_0 = 10.23$ MHz:

$$f_{L1} = 154\, f_0 = 1575.42 \text{ MHz}, \tag{2.6}$$

$$f_{L2} = 120\, f_0 = 1227.6 \text{ MHz}. \tag{2.7}$$

These frequencies are very accurate as their reference is an atomic frequency standard. The generated clock is slightly lower than 10.23 MHz to take the relativistic effect into consideration. When a GPS receiver at Earth surface distance receives the signals, they are at the desired frequencies. However, the relative motion between satellites and receiver will produce a Doppler frequency shift, that is approximately ±5 KHz at L1 for a standing receiver.

The signals are composed of the following three parts:

- Carrier: The carrier wave with frequency f_{L1} or f_{L2},
- Navigation data: The navigation data contain information regarding satellite orbits. This information is uploaded to all satellites for the ground stations in the GPS Control Segment. The navigation data have a bit rate of 50 bps.
- Spreading sequence: Each satellite has two unique spreading sequences or codes. The first one is the coarse acquisition code (C/A), and the other is the encrypted precision code (P(Y)). The C/A code is a sequence of 1023 chips. The code is repeated each ms giving a chipping rate of 1.023 MHz. The P code is longer with a chipping rate of 1.023 MHz. Codes repeat themselves each weekly. The C/A code is only modulated onto the L1 carrier while the P(Y) code is modulated onto both the L1 and the L2 carrier.

2.4.1 GPS signal: scheme and generation

Figure 2.3 ([16]) shows a block diagram description of the signal generation process [15]. Block diagram shows how the main clock signal is supplied to the remaining blocks. When multiplied by 154 and 120, it generates the L1 and L2 carrier signals. At the very bottom the data generator generates the navigation data. The code generators and data generator are synchronized through the X_1 signal supplied by the P(Y) code generator.

After code generation, the codes are combined with the navigation data through modulo-2 adders (XOR operation). If the binary sequences are represented by the polar non-return-to-zero representation, i.e., 1's and -1's, ordinary multiplication can be used instead. The C/A code $\oplus$ data and P(Y) code $\oplus$ data signals are supplied to the two modulators for the L1 frequency. Here the signals are modulated onto the carrier signal using the binary phase shift keying (BPSK) method. The two codes are modulated in-phase and quadrature with

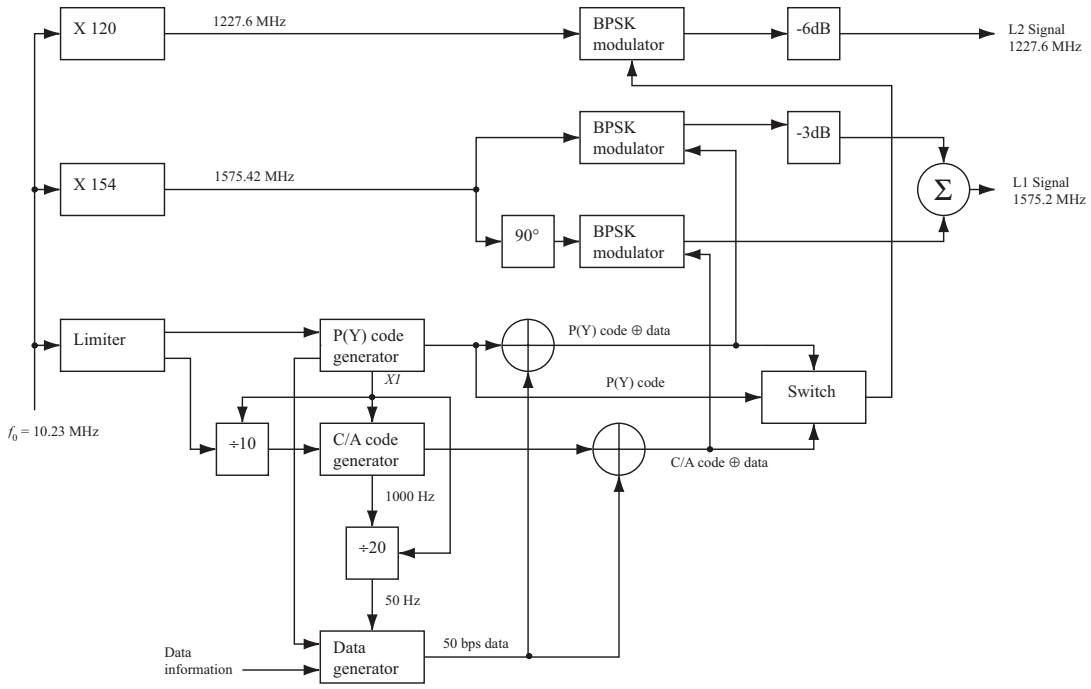


Figure 2.3: Generation of GPS signals at the satellites [16, p. 18]

each other on L1 (90^{o} phase shift between them). After the P(Y) part is attenuated 3dB, these two L1 signals are added to form the resulting L1 signal. The signal transmitted from satellite k in the L1 frequency can be written as:

$$S_{L1}^{k}(t) = A_{P}^{k} P^{k}(t)\, D^{k}(t) \cos\left(2\pi f_{L1} t + \phi\right) + A_{C}^{k} C^{k}(t)\, D^{k}(t) \sin\left(2\pi f_{L1} t + \phi\right), \tag{2.8}$$

where S_{L1}^{k} is the signal at L1 frequency, A_{P}^{k} is the amplitude of the P code, $P^{k}(t) = \pm 1$ represents the phase of the P code, $D^{k}(t) = \pm 1$ represents the data code, f_{L1} is the L1 frequency, ϕ is the initial phase, A_{C}^{k} is the amplitude of the C/A code, $C^{k}(t) = \pm 1$ represents the phase of the C/A code, all of them for the considered k satellite. The P(Y), C/A, and the carrier frequencies are all phase locked together.

Although it largely depends on the receiver, typically the minimum required power levels of the signal be over -130 dBm in L1 band and -136 dBm in L2 for successful C/A signal decoding [18]. These power levels are very weak and the spectrum is spread, therefor they cannot be directly observed. Even when the signal is amplified, the C/A code cannot be observed because it is under the noise power level. As it is explained in [18], the received power levels at various points on the earth are different. The maximum difference is about 2.1 dB between a point under a satellite and a point tangential to the surface of the Earth. In order to generate a uniform power over the surface of the earth, the main beam pattern of the transmitting antenna is slightly weaker at the center. The maximum power is -128 dBm, which occurs at about 40 degrees [18]. The signals generated by the satellite transmitting antenna are right-hand polarized. Thus, in order to achieve the maximum efficiency, the receiver antenna should also be right-hand polarized to receive the satellite direct signal.

Figure 2.4 ([16]) shows the three parts forming the signal on the L1 frequency. The C/A code repeats itself every ms, and one navigation bit lasts 20 ms. Hence for each navigation bit, the signal contains 20 complete C/A codes.

Figure 2.5 ([16]) shows the Gold code C, the navigation data D, the modulo-2 added signal (equivalent to the product for polar non-return-to-zero representation bit representation) and the carrier. The final signal is created by binary phase-shift keying (BPSK) where the carrier is instantaneously phase shifted by 180^{o} at the time of a chip change. When a navigation data bit transition occurs (about third from the right edge), the phase resulting signal is also phase-shifted 180^{o}.

In summary: For GPS the code length is 1023 chips, 1.023 MHz chipping rate (1 ms period time), 50 Hz data rate (20 code periods per data bit), and approximately 90% of signal power within 2 MHz bandwidth.

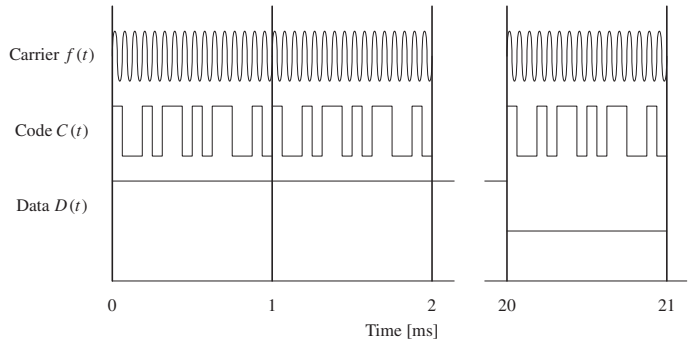


Figure 2.4: L1 signal structure: $f(t)$ is the carrier wave and $C(t)$ is the discrete C/A code sequence. This signal repeats itself every ms. $D(t)$ is the discrete navigation data bit stream. One navigation bit lasts 20 ms. The three parts of the L1 signal are multiplied to form the resulting signal. This Figure is not to scale, it is presented only for illustrative purposes [16, p. 20]

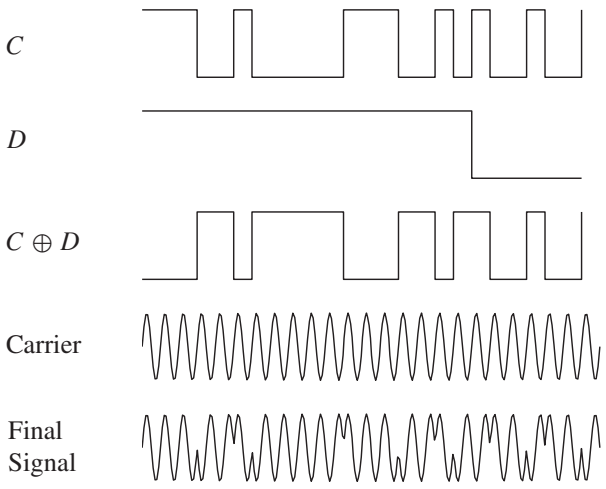


Figure 2.5: First 25 chips of the Gold code for PRN 1. The Figure shows the effect of BPSK modulation of the L1 carrier wave with the C/A code [16, p. 21]

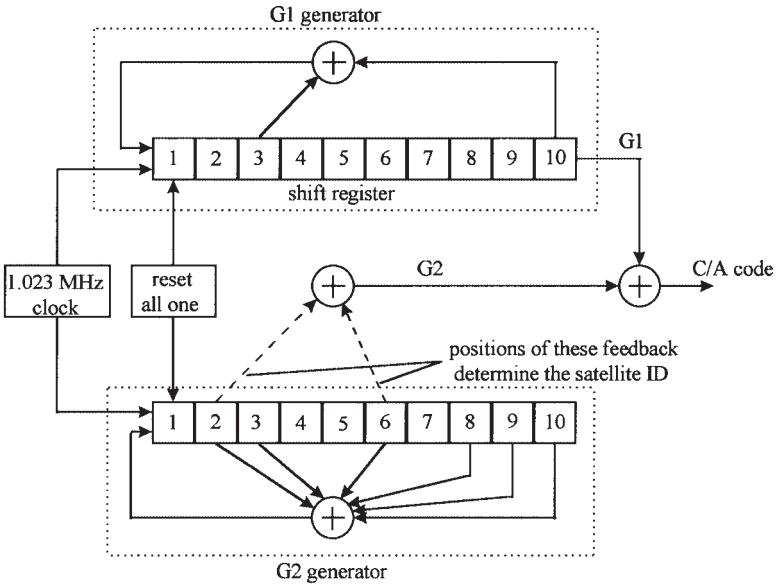


Figure 2.6: C/A code generator [18, p. 81]

2.4.2 The C/A Code

The spreading sequences used as C/A codes in GPS belong to a unique family of sequences. They are often referred to as Gold codes [17]. The are also referred to as pseudo-random noise sequences (PRN sequences), because of their characteristics.

PRN Codes The GPS C/A code is generated using two 10-bit linear feedback shift registers (LFSR) known as G_1 and G_2. These shift registers each have 10 cells generating sequences of length 1023. The two resulting 1023 chip-long sequences are modulo-2 added to generate a 1023 chip-long C/A code. They generate a maximal-length sequence of length $N = 2^n - 1$ elements. To make different C/A codes for the satellites, the output of the two shift registers are combined in a very special manner. The G_1 register always supplies its output, but the G_2 register supplies two of its states to a modulo-2 adder to generate its output. Figure 2.6 shows the block diagram of the C/A codes generation process ([18]). More details on the C/A code generation can be found in [14].

The periodic autocorrelation function for a PRN sequence $a(t)$ of length n chips, with a chip period T_C can be written as follows:

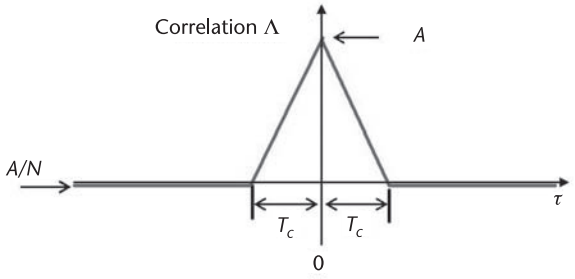


Figure 2.7: Ideal autocorrelation of a PRN sequence [19, p. 44]

$$\Lambda(\tau) = \frac{1}{NT_C} \int_0^{NT_C} a(t)\, a(t+\tau)\, dt$$

The autocorrelation envelope (Figure 2.7) can be approximated by a triangle function, the peak of which (amplitude A) corresponds to the perfect alignment (correlation) between the received code and the locally generated replica [19]. Outside the correlation interval the ideal cross-correlation function is $-A/N$ due to the avoidance of the all-zero (stable) state in the generation of Gold codes.

In practice the PRN sequences exhibit a three-valued cross-correlation function with values $\left\{-\frac{1}{2^{n-1}}, \frac{-t(n)}{2^{n-1}}, \frac{t(n)-2}{2^{n-1}}\right\}$, where

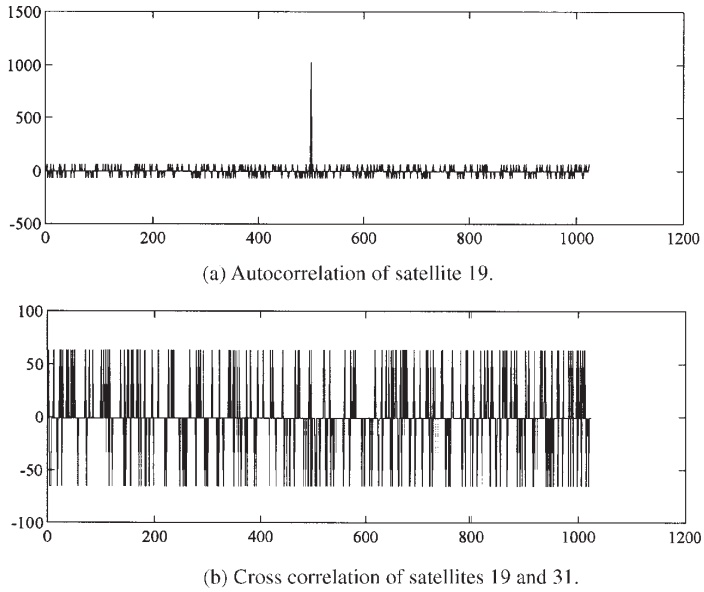
$$t(n) = \begin{cases} 2^{(n+1)/2} + 1 & (\text{odd } n) \\ 2^{(n+2)/2} + 1 & (\text{even } n) \end{cases}$$

Therefore, PRN sequences employed for GPS C/A code have cross-correlation values of $\left\{-\frac{1}{1023}, -\frac{65}{1023}, \frac{63}{1023}\right\}$. The autocorrelation of the C/ A codes of satellite 19 and the cross correlation of satellites 19 and 31, both of them multiplied by the code length (1023), are shown in Figures 2.8a and 2.8b respectively. These satellites were arbitrarily chosen.

It becomes obvious that C/A codes were not arbitrary selected for the GPS system, but they were selected because of their correlation properties:

1. Nearly no cross correlation: All the C/A codes are nearly uncorrelated with each other

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



(a) Autocorrelation of satellite 19.

(b) Cross correlation of satellites 19 and 31.

Figure 2.8: Auto and cross correlation of C/ A code [18, p. 84]

Where C^i and C^k are the codes for satellites i and k.

2. Nearly no correlation except for zero lag: This property makes it easy to find out when two similar codes are perfectly aligned (Correlation Gain)

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

2.4.3 Doppler Frequency Shift

The Doppler frequency shift caused by the motion of the transmitter (satellite) relative to the receiver affects both the acquisition and tracking of the GPS signal. For stationary GPS receiver the maximum Doppler frequency shift for the L1 frequency is around ± 5 kHz. If the receiver is in movement, this maximum shift value increases.

The Doppler frequency shift on the C/A code is small because of the low chip rate of the C/A code. The C/A code has a chip rate of 1.023 MHz, which is 1575.42/1.023 = 1540 times lower than the L1 carrier frequency. It follows that the maximum Doppler frequency on the C/A code is approximately 3.2 Hz for the stationary receiver.

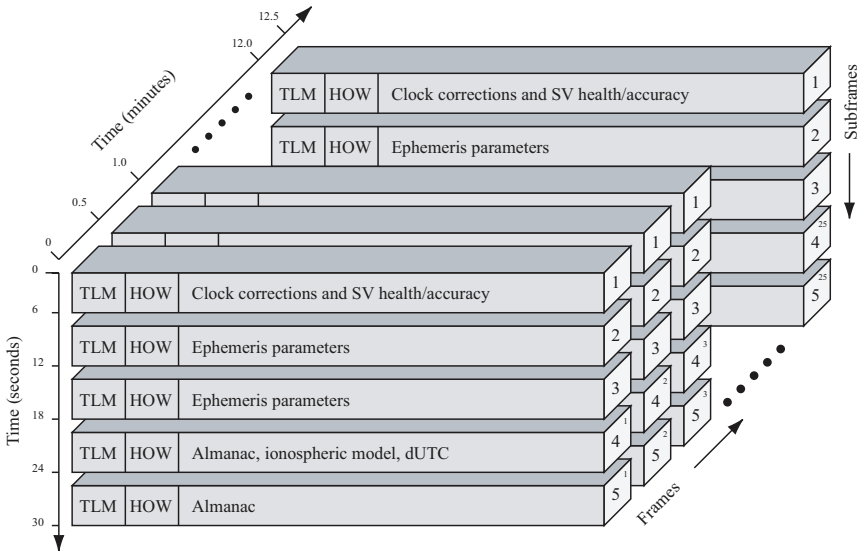


Figure 2.9: GPS signal data overall structure [16, p. 29]

The Doppler frequency on the C/A code can cause misalignment between the received and the locally generated codes and the values of the Doppler frequency are important for tracking the received signal.

2.4.4 Navigation Data

Once the carrier and the code have been wiped off the incoming signal, the remaining information, disregarding any noise, are the encoded navigation data broadcast by the satellite. The navigation data are transmitted on the L1 frequency with a bit rate of 50 bps. Figure 2.9 ([16]) shows the overall structure of an entire navigation message.

The basic format of the navigation data is a 1500-bit-long frame containing 5 subframes, each having length 300 bits. One subframe contains 10 words, each word having length 30 bits. With the bit rate of 50 bps, the transmission of a subframe lasts 6 s, one frame lasts 30 s, and one entire navigation message lasts 12.5 minutes.

Data bits are encoded as ± 1, and bit transitions can thus be recovered by looking for 180 degree phase shifts (negations) of the incoming carrier signal. The subframes of 10 words always begin with two special words, the telemetry (TLM) and handover word (HOW) pair.

TLM is the first word of each subframe and it is thus repeated every 6 s. It contains a preamble that should be used for frame synchronization. HOW contains a 17-bit version

of the time of week (TOW), followed by two flags supplying information to the user of antispoofing, etc.

In addition to the TLM and HOW words, each subframe contains eight words of data.

- Subframe 1 contains first of all clock information. This information will be needed to compute the time navigation message transmitted from the satellite. In addition, this frame also contains health data indicating if data should or should not be trusted.
- Subframes 2 and 3 contain the satellite ephemeris data . The ephemeris data relate to the satellite orbit and are needed to compute a satellite position. Subframes 4 and 5 contain almanac data. The almanac data are the ephemeris and clock data with reduced precision.
- Additionally subframes 4 and 5 also contain UTC parameters, health indicators, and ionospheric parameters. For a more in-depth description of the contents of the navigation data, see [20].

2.5 Chapter summary and conclusions

This chapter has provided an overview of the GPS signal main features, paying special attention to the C/A code and L1 band. The 3 segments in which is the GPS architecture is divided: the space segment, the control segment and the user segment; have been also introduced.

This chapter shows how thanks to the pseudorange measurements and the navigation data contained within the received signal, a GPS receiver is able to compute their position anywhere in Earth's surface at anytime. Although the received signal is very weak, C/A code features provide a considerable processing gain that allows a successful signal acquisition and tracking. Those acquisition an tracking processes performed by the GPS receiver will be described in more detail in the following chapter.

Chapter 3

C/A signal acquisition and tracking

This chapter describes both acquisition and tracking procedures. Signal acquisition is introduced first. Its main features will be described, as well as the main different acquisition methods. After acquisition, signal tracking (both, carrier and code tracking) is described. This chapter content has been carefully selected to understand the work described in the following chapters. Thus, even when there are more possibilities for acquisition and tracking, this chapter focuses on those related with the work done.

The signal processing for GPS is based on a channelized structured. Figure 3.1 shows an overview of a receiver's channel. The first step is the acquisition. The acquisition gives rough estimates of the signal parameters. Next, these parameters are refined by the tracking blocks. Acquisition and tracking are required in order to extract the navigation data from the signal and to compute the pseudoranges.

3.1 C/A signal acquisition

In order to track and decode the information in the GPS signal, an acquisition method must be used. The purpose of acquisition is to identify all satellites visible to the user [16]. If a satellite is visible, the acquisition process must determine the next properties of the received signal:

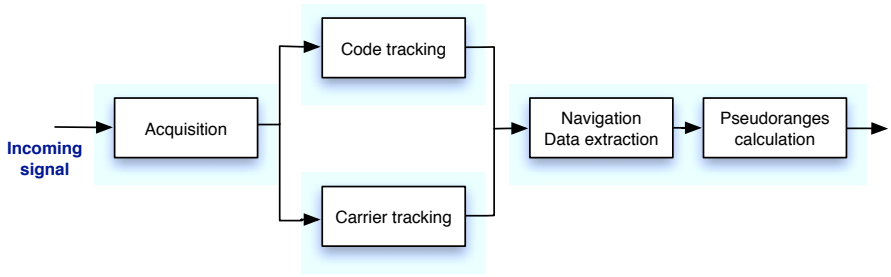


Figure 3.1: GPS receiver channel. Block diagram

- Frequency: The frequency of the signal received from a specific satellite will probably differ to its nominal value. The reason for this difference, as it was previously introduced, is the relative motion between the satellite and the receiver, causing a Doppler effect.
- Code Phase: The code phase indicates the point in the current data block where the C/A code begins. If a data block of 1 ms is examined, the data include an entire C/A code.

Once the signal has been detected, these parameters will be passed to a tracking module. Tracking process will precisely determine those parameters in order to track the received signal and be able to obtain information such as the navigation data.

The acquisition method must search over all the possible expected frequency range to cover the possible Doppler effects. In order to accomplish the search in a short time, the bandwidth of the searching program has to be carefully selected. It can not be very narrow, using a narrow bandwidth for searching means taking many steps to cover the desired frequency range and it could be considerably time consuming. However, searching through with a too wide bandwidth filter will provide a relatively poor sensitivity. On the other hand, the tracking method usually has a much narrower bandwidth [18]; thus it's possible to achieve high sensitivity.

For any GPS receiver, the received signal s is a combination of signals from all n visible satellites:

$$s(t) = s_1(t) + s_2(t) + ... + s_n(t) \tag{3.1}$$

GPS uses a code-division multiple access (CDMA) system scheme. Thus, thanks to the C/A code correlation properties, its possible to acquire the k^{th} satellite signal from the signal s.

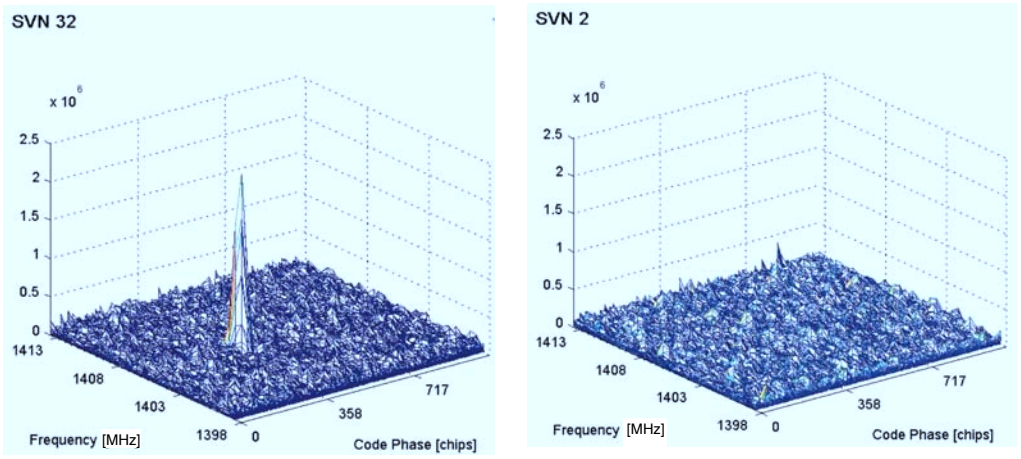


Figure 3.2: Example acquisition results for SVNs 32 and 2 (arbitrary selected). First plot shows a peak corresponding to a positive satellite with PRN 32 detection, while second plot shows the results when satellite with PRN 2 is not visible

This is done by multiplying s with the local replica of the C/A code corresponding to the satellite k. Performing this operation, signals from other other satellites are nearly removed due to the almost 0 cross-correlation between C/A codes. In addition, to avoid removing the desired signal component, the locally generated C/A code must have the correct code phase (aligned in time).

After multiplication with the locally generated code, the signal must be mixed with a local generated carrier wave in order to remove this carrier wave from the received signal. As mentioned before, the frequency will suffer variations from the nominal frequency due to the doppler shift. To identify if a satellite is visible or not, it is sufficient to search the frequency in steps of 500 Hz [16].

Following the carrier wave removing process, all signal components are squared and summed providing a numerical value. The acquisition procedure works as a search procedure over a set of those computed numerical values. Each value belongs to one specific code phase and frequency shift pair.

If the maximum value of this set of values exceeds a determined threshold, the satellite will be considered as acquired with the frequency and phase shift corresponding to that maximum value. Figure 3.2 shows a typical acquisition plot performed for a visible satellite and for a satellite that is not currently visible.

3.1.1 Maximum data length for acquisition

In theory, the longer the data record used in acquisition calculation the higher the signal-to-noise ratio that can be achieved [18]. But, increasing the data record size implies increasing the time of calculation or using a more complex hardware if acquisition process is performed by hardware.

The main factors that can limit the length of the data record are: the bit transitions in navigation data and the doppler effect on the C/A code [16].

The 20 ms bit duration will limit the data record size to 10 ms when there is no navigation data bit synchronization. The reason is that in 20 ms of data at most there can be only one data transition. Thus, if one takes the first 10 ms of data and there is a data transition, the next 10 ms will not have one. If longer data record are used, and it contains a navigation data transition, the transition will spread the spectrum, degrading the acquisition result value.

The second limit of data length is related with the Doppler effect on the C/A code. If a perfect correlation peak is 1, the correction peak decreases to 0.5 when a C/A code is off by half a chip (corresponding to 6 dB decrease in amplitude). In, [18] Tsui demonstrates demonstrates this limitation with the following example case: let's assume that the maximum allowed C/A code misalignment is half a chip (0.489 us) for effective correction. The chip frequency is 1.023 MHz and the maximum Doppler shift expected on the C/A code is 6.4 Hz [18]. It takes about 78 ms (1/(2x6.4)) for two frequencies different by 6.4 Hz to change by half a chip. This data length limit is much longer than the 10 ms; therefore, 10 ms of data should be considered the longest data used for typical acquisition methods.

3.1.2 Acquisition methods

In the following sections two different acquisition methods will be presented. First serial search acquisition method will be introduced. This method is the simplest acquisition approach. In order to overpass the serial search acquisition limitations, code phase parallel acquisition is introduced. This method, although is higher complexity, is able to perform the acquisition in much lower times and it represents a more computationally efficient alternative to serial acquisition.

3.1.2.1 Serial search acquisition

Serial search acquisition the most simple approach in order to perform the C/A signal acquisition. This method is often-used for acquisition in CDMA systems [16] and GPS is a CDMA system. Figure 3.3 shows the block diagram of the serial search algorithm.

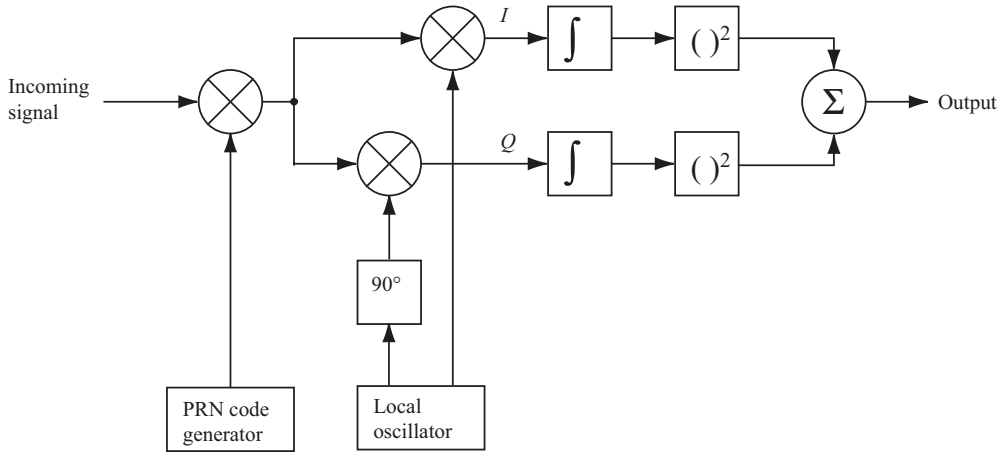


Figure 3.3: Serial search algorithm block diagram [16, p. 76]

Serial search algorithm is based on multiplication of locally generated PRN code sequences and locally generated carrier signals. The receiver has a PRN generation block which generates a PRN sequence corresponding to a specific satellite. This PRN sequence is generated with a certain code phase, from 0 to 1022 chips. The received signal is initially multiplied by this locally generated PRN sequence, and after that is multiplied by a locally generated carrier signal. Multiplication with the locally generated carrier signal generates the in-phase signal I, and multiplication with a 90º phase-shifted version of the locally generated carrier signal generates the quadrature signal Q.

The obtained I and Q signals are integrated over 1 ms, corresponding to the length of one C/A code, and finally squared and added. The output is a value of correlation between the incoming signal and the locally generated signal. If the computed correlation value is over a predefined threshold, the frequency and code phase parameters are considered correct, and they can be passed on to the tracking algorithms.

The use of I and Q signals is because the phase of the received signal is unknown, thus it will be a common feature of all acquisition methods. If the locally generated carrier signal is not completely phase aligned with the received signal, it will be possible that part of the received signal power has moved to the Q component because , even when ideally the signal power should be located entirely in the I part, as the C/A code is only modulated onto that [16]. Thus, to assure a correct signal detection, it is necessary to take into account both the I and the Q signal.

The acquisition search algorithm perfumes two different sweeps: a frequency sweep over an

specified bandwidth, in 500 Hz steps for ±10 kHz in a typical receiver [16]; and a code phase weep over all 1023 different code phases. Then, the total number of possible combinations in this case is

$$1023 \text{ (code phases)} * 41 \text{(frequency steps)} = 41943 \text{ combinations.} \tag{3.2}$$

With such a large number of possible combination, the exhaustive use of this acquisition method tends to be very time consuming. This elevated time consumption can be considered as the main weakness of the serial search acquisition. On the other hand, the implementation of the acquisition method is very straight-forward, which requires the use of less complex hardware or software in the receiver.

3.1.2.2 Parallel code search acquisition

The serial search acquisition method showed that it is a very time-consuming procedure to search sequentially through all possible values of the two parameters frequency and code phase. If any of the two parameters could be eliminated from the search procedure or if possible implemented in parallel, the performance of the procedure would increase significantly.

As the name parallel code phase search acquisition implies, this acquisition method parallelizes the search of the code phase parameter using the Fourier transform. The basic principle of this method was first presented in [21]. Parallelizing acquisition in the code phase dimension implies that only a reduced number of steps should be performed. The number of steps corresponds to the number of possible frequency steps in the selected acquisition bandwidth. Modern receivers often use this search technique, which can detect the signal presence within a matter of milliseconds [22].

The goal of the acquisition is to perform a correlation with the incoming signal and a PRN code. Instead of multiplying the input signal with a PRN code with 1023 different code phases as done in the serial search algorithm method, it is more convenient to make a circular cross correlation between the input and the PRN code without shifted code phase.

3.1.2.3 Circular correlations with Fourier transforms

In the following, a method of performing circular correlation through Fourier transforms will be described, see [16, p. 82], [23, p. 746] and [18, p. 140].

The discrete Fourier transforms of the finite length sequences $x(n)$ and $y(n)$ both with lengths N are computed as

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N} \text{ and } Y(k) = \sum_{n=0}^{N-1} y(n)\, e^{-j2\pi kn/N} \tag{3.3}$$

On the other side, the circular cross-correlation sequence between two finite length sequences x(n) and y(n) both with length N and with periodic repetition is defined as

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

Then the discrete N-point Fourier transform of z(n) can be expressed as (the $\frac{1}{N}$ scaling factor is omitted)

$$Z(k) = \sum \sum_{m=0}^{N-1} x(-m)\, y(m-n)\, e^{-j2\pi kn/N} \tag{3.5}$$

$$= \sum_{m=0}^{N-1} x(m)\, e^{j2\pi km/N} \sum_{n=0}^{N-1} y(m+n)\, e^{-j2\pi k(m+n)/N} = X^*(k)\, Y(k) \tag{3.6}$$

where $X^*(k)$ is the complex conjugate of $X(k)$

3.1.2.4 Parallel acquisition: Method steps description

Figure 3.4 shows a block diagram of the parallel code phase search method. Its basic steps are:

1. First, the incoming signal is multiplied by a locally generated carrier signal. Same as in serial search acquisition method, multiplication with the signal generates the I signal, and multiplication with a 90^{o} phase-shifted version of the signal generates the Q signal.
2. Next, the I and Q signals are combined to form a complex signal $x(n) = I(n) + jQ(n)$.
3. The complex signal $x(n)$ is used as input to the DFT function.
4. The locally generated PRN code is transformed into the frequency domain and the result is complex conjugated.
5. The Fourier transform of the input is then multiplied with the Fourier transform of the PRN code.

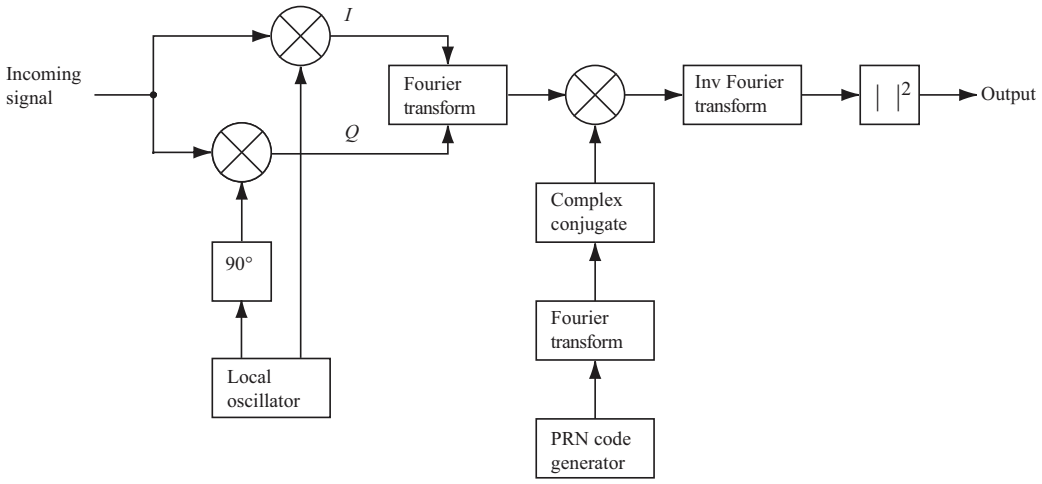


Figure 3.4: Parallel code phase phase acquisition block diagram [16, p. 83]

6. The result of the multiplication is transformed into the time domain by an inverse Fourier transform.

7. The absolute value of the output of the inverse Fourier transform represents the correlation between the input and the PRN code. If a peak that exceeds a predefined threshold is present in the correlation, the index of this peak marks the PRN code phase of the incoming signal.

The previous process must be repeated for each defined carrier frequency step. For each frequency step is necessary to perform one Fourier transform and one inverse Fourier transform, so the computational efficiency of the method strongly depends on the implementation of these functions.

The accuracy of the parameters estimated by this acquisition depends on the sampling frequency. In many cases, the PRN code phase estimation is more accurate compared to serial search acquisition as it gives a correlation value for each sampled code phase. That is, if the sampling frequency is 5 MHz, a sampled PRN code has 5000 samples, so the accuracy of the code phase can have 5000 different values instead of 1023 [16].

An additional advantage of this method is that only one PRN code should be generated for each acquisition. The PRN generator generates a code with no code phase. As in the implementation of the other acquisition algorithms, the code generation can be performed off-line. The next step performs a Fourier transform of the PRN code, and the result is

complex conjugated. This PRN generated code, now in frequency domain, is ready to be multiplied with the incoming signal component in frequency domain, in each frequency step.

3.1.3 Acquisition methods comparison: execution time and system complexity

As previously mentioned, the theoretical performance regarding the computational demands are different between the two acquisition methods introduced [22]. In [16], an execution time comparison between serial search acquisition and code phase parallel acquisition is presented. Both methods are implemented using Mathworks MATLAB and are executed over the same hardware (a PC). Serial search showed to be 87 times slower than parallel code phase search.

The considerable reduction in the acquisition time makes parallel search very convenient in some applications. Parallel search is a must for some applications, like the use GPS in LEO satellites where the satellite visibility time window can last only few minutes [24]. In addition, parallel acquisition shows to be the only feasible option for weak signal acquisition [25]. That's because in order to increase the receiver's sensibility, usually coherent integration time is increased and with it the computation time. For example, if a 10 times improvement in sensitivity is desired, it is possible to increase the coherent integration time to 10 ms, but in addition it is also needed to increase the fineness of the grid in the frequency dimension by another factor of 10. This implies a new Time to First Fix (TTFF) approximately 100 times higher. Considering the TTFF could be on the order of 30 min for 1 ms coherent integration time for signals around 35 dB-Hz, increasing to 10 ms integration time implies a new TTFF on the order of 3000, which in practice is totally useless.

Although the results obtained in [16], is important to notice that the price to pay in code phase parallel acquisition is a significant increase in receiver's complexity. Parallel acquisition implies the use of Fourier transforms, which requires specific hardware (if the acquisition module is hardware implemented) or a fast enough signal processor (for a software acquisition module).

3.2 Tracking C/A signal basics

The acquisition process only provides coarse estimations of the time delay and frequency shift parameters. Thus, it is necessary to implement a process to track and refine these values and demodulate the navigation data from the specific satellite, to finally provide an estimation

of the pseudorange. The accuracy of the final value of the code phase is connected to the accuracy of the pseudorange estimated.

The tracking contains two parts, code tracking and carrier frequency/phase tracking:

- Code tracking: The code tracking is most often implemented as a delay lock loop (DLL) where three local codes (replicas) are generated and correlated with the received signal. These three replicas are referred to as the early, prompt, and late replica, respectively. The three codes are separated by a half-chip length.
- Carrier frequency/phase tracking: The other part of the tracking is the carrier wave tracking. This tracking can be done in two ways: either by tracking the phase of the signal or by tracking the frequency. In this case, it is possible to use a frequency-locked loop (FLL) or a PLL, from which we define an incoherent or coherent system, respectively. A FLL deliver more robust tracking in high-dynamic environments and in very weak signal conditions [26]. On the other hand, in order to enable precise pseudorange rate, integrated Doppler measurements and carrier phase positioning a receiver must track the phase of the incoming carrier. Thus, a coherent PLL system is required.

The tracking is running continuously to follow the changes in frequency as a function of time. If the receiver loses track of a satellite, a new acquisition must be performed for that particular satellite.

3.2.1 Delay-Locked Loop (DLL)

In most common GPS receivers, the PRN code sequence is acquired and tracked using a delay-locked loop (DLL) [**?**]. DLL can be achieved when implementing either incoherent FLL or coherent PLL carrier tracking [16] The idea behind the DLL is to correlate the input signal with three replicas of the code.

The first step of this code tracking method is converting the C/A code to baseband, by multiplying the incoming signal with a perfectly aligned local replica of the carrier wave. After that, the signal is multiplied with the three code replicas. The three replicas are nominally generated with a spacing of $\pm 1/2$ chip. After this second multiplication, the three outputs are integrated and dumped. The output of these integrations is a numerical value indicating how much the specific code replica correlates with the code in the incoming signal.

The three correlation outputs I_E, I_P and I_L are then compared to see which one provides the highest correlation. Figure 3.5 shows and example of code tracking [16]. In Figure 3.5a the

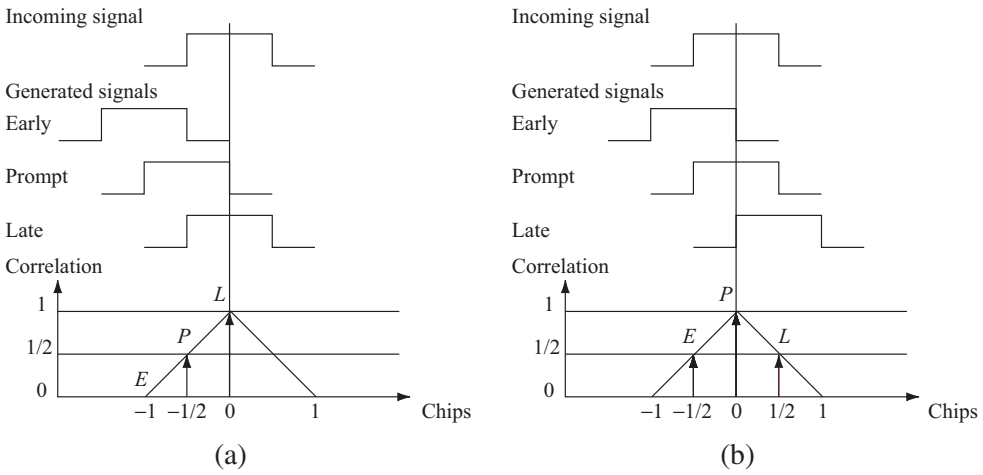


Figure 3.5: Code tracking example [16, p. 97]. Three different local codes are generated and correlated with the incoming signal. (a) The late replica has the highest correlation so the code phase must be decreased, i.e., the code sequence must be delayed. (b) The prompt code has the highest correlation, and the early and late have similar correlation. The loop is perfectly tuned in.

late code has the highest correlation, so the code phase must be decreased. In Figure 3.5b the highest peak is now aligned with the prompt replica, while the early and late replicas have equal correlation. This case is an example of properly tracked code phase.

The described DLL with three correlators is only optimal when the local carrier wave is locked in phase and frequency. But when there is a phase error on the local carrier wave, the signal will be more noisy, making more difficult for the DLL to keep lock on the code. So, the DLL present in a GPS receiver is often designed as in Figure 3.6. This design is independent of the phase on the local carrier wave. If the code tracking loop performance has to be independent of the performance of the phase lock loop, the tracking loop has to use both the in-phase an quadrature arms to track the code [16, **?**].

After getting the error value, the DLL needs to feedback to the PRN code generators if the code phase needs to be adjusted. There are different possible discriminators, coherent and non-coherent [16]. The DLL discriminator selection depends on the type of application and the noise in the signal. One common tracking loop discriminator is the the normalized early minus late power [16, **?**] This discriminator can be described as

$$D = \frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{(I_E^2 + Q_E^2) + (I_L^2 + Q_L^2)} \tag{3.7}$$

where I_E, Q_E, I_L, and Q_L are output from four of the six DLL correlators. Typically this

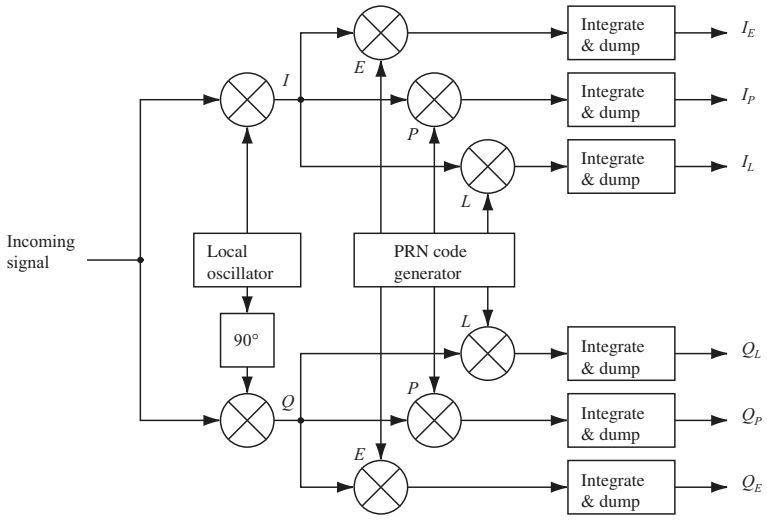


Figure 3.6: Block diagram of the DLL with six correlators [16, p. 97]

discriminator is chosen because it is independent of the performance of the PLL as it uses both the in-phase and quadrature arms. The normalization of the discriminator causes that the discriminator can be used with signals with different signal-to-noise ratios and different signal levels [16, pp. 99-100].

3.3 Chapter summary and conclusions

This chapter has introduced the acquisition and tracking processes performed by every GPS receiver.

First acquisition process has been extensively described, including different approaches to this process. The first approach, the serial search acquisition has showed to be too much time consuming. This handicap represents an important limitation for fast acquisitions required in high dynamics environments. It also has been considered that improving acquisition sensibility with serial search acquisition implies a not feasible increase in acquisition time. On the other hand, parallel code phase acquisition has been presented as an very interesting alternative. Working in frequency domain allows to increase the acquisition speed as well as acquisition sensibility. The price to pay is a more complex GPS receiver acquisition hardware or software.

Finally, the C/A signal tracking has been depicted in the last section. Tracking the C/A signal implies tracking the carrier wave and the code phase. For code phase tracking, the most

common solution is to use a Delay Lock Loop (DLL). The DLL basics have been introduced together with some design considerations applied to a GPS receiver, like the recommended non-coherent discriminator, and the use of 6 correlators (3 for the in-phase branch, and 3 for the quadrature branch).

Chapter 4

Namuru GNSS development platform overview

This chapter goal is to introduce the Namuru GPS receiver that has been used as development platform in this project's development.

First of all, a description of a generic GPS receiver RF front-end is presented. This description emphasizes the main RF front-end and the typical GPS receiver's antenna features. Following this general description, the rest of the chapter content is divided in two parts: the first part describes the different hardware elements that compose the Namuru development board; on the other hand, the second part introduces the software tools that are also part of the development platform.

The explained software tools were already used to develop the existent Namuru GPS receiver and are necessary to implement hardware and software using the Namuru platform. Due to the close relationship with the design tools, the design methodology is also presented together with the software tools description.

4.1 GPS receiver front-end hardware overview

This section's purpose is to provide some insight into how the satellite signals propagating through space result in useful digital data stream. This conversion task is performed by the RF front-end hardware.

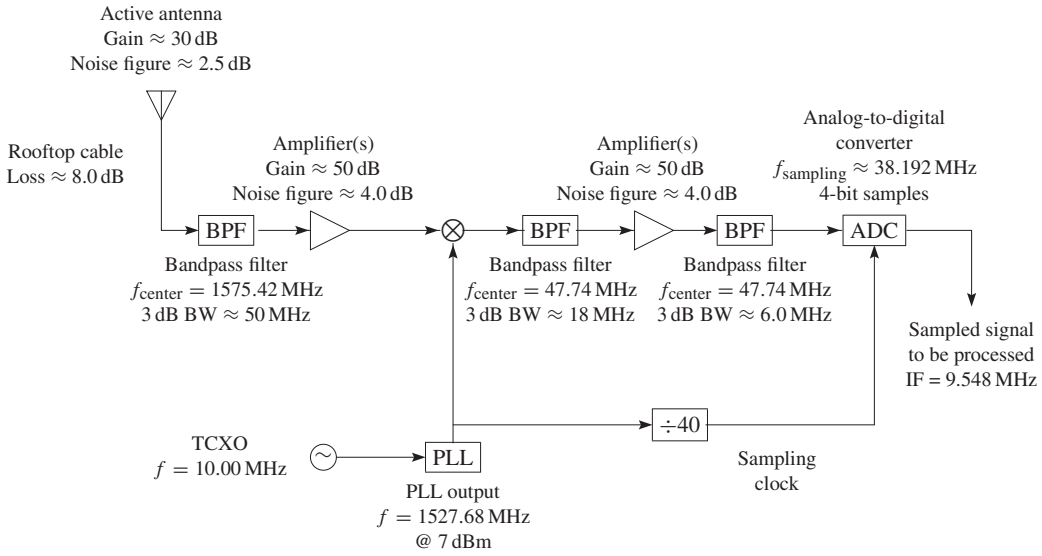


Figure 4.1: GNSS L1 front-end block diagram [16, pp. 55]

The receiver operation process starts with the GNSS signal, propagating through space, which is incident on a GNSS antenna. This, in turn, induces a voltage within the element. That voltage corresponds to a guaranteed signal power of -160 dBW in the case of the GPS (for C/A code) [27] and has a carrier frequency of 1575.42 MHz. Considering a bandwidth of 2 MHz (the approximate null-to-null bandwidth the GPS C/A code signal), the received GPS signal power is actually below that of the thermal noise floor [16, p. 53]. This is a feature of the code division multiple access (CDMA) spread spectrum signal. Thus the design of the front end is based more on the level of the thermal noise rather than the received L1 band navigation signal.

The analog induced voltage that results from the incident GPS signal and thermal noise remains much too weak and at too high a frequency for most analog-to-digital converters (ADCs) to operate [16, p. 54]. In order to overcome this, the front end will utilize a combination of amplifier(s), mixer(s), filter(s), and its own oscillator to condition the incident voltage on the antenna to the resulting digital samples. Figure 4.1 shows a functional GNSS L1 front end block diagram.

The front-end includes an analog-to-digital converter (ADC). Figure 4.1 shows that the ADC is the final component in the front-end path. The CDMA mature of the GNSS signal requires very little dynamic range from the sample signal [16, p. 54]. A conservative approach is to use 2 bits per sample. In this case some form of gain control must be implemented to provide

proper quantization. A common practice is to implement an automatic gain control (AGC). The goal of the front-end is to exercise all the available bits with the ADC. Thus, the gain can be dynamically adjusted to do so.

In summary, the object of the bulk components within the front-end is to condition the voltage incident on the antenna fro sampling by the ADC. In order to accomplish this for the most ADCs, the required functions are: amplification, frequency translation/downconversion, and filtering. These prepare the signal for analog-to-digital conversion, which in the end results in the samples to be processed within the software receiver.

4.1.1 GNSS Antenna

Although typically the antenna is not considered a part of the front-end design, since it is the firs component in the signal path, it dictates elements that follow and it is important to review the main requirements that a L1 GPS receiver antenna has to fulfill.

To select the right antenna is important to consider a set of fundamental parameters to describe its performance. In this case, it is possible to consider three parameters as the most relevant:

- **Frequency and Bandwidth:** The antenna will be designed to induce a voltage form radio waves propagating at GNSS L1frequency or 1.574 GHz. In addition, the design should accommodate the appropriate bandwidth of the the desired signal, that is 2 MHz. This is usually specified using two additional antenna parameters [16, p. 56]: the Voltage Standing Wave Ratio (VSWR) and impedance. Practically all GNSS front-end components utilize an impedance of 50 Ω, which is typical for a majority of radio frequency design. VSWR is a measure of impedance mismatch. The VSWR is typically on the order of 2.0:1, which equates to 90% power absorption across the bandwidth of desired frequencies.

- **Polarization:** Received GNSS signals are right-hand circularly polarized (RHCP), and the antenna should be designed as such. Using RHCP is used to mitigating the multipath effects. As it has been shown before in this document, the polarization of the reflected signal will flip to left-hand circular polarization (LHCP). The polarization of the GNSS antenna provides a significant level of suppression from erroneous multipath reflection.

- **The antenna pattern:** The antenna pattern describes the directivity of the antenna. Isotropic antennas don't make sense for GNSS. This is because the signal source, GNSS

satellites, are overhead for most applications. Thus, an hemispherical antenna pattern is preferred to receive the signal from only positive elevation angles from all azimuth directions. Given the problem of multipath, the antenna pattern could be crafted such that it was designed to reject the received signals below 10°-20° elevation.

Probably two of the most popular GNSS L1 antenna implementations are the patch and helix approaches[16, p. 56]. These approaches can be either active or passive antennas. In the case of active antennas, now the antenna is an active element and requires power for its internal amplifier. Using an active antenna approach is common in cases where the antenna is far from the receiver or when extra sensibility is required. In the rest of situations, such as handheld GNSS receivers, a passive antenna is used.

4.2 The Namuru V2 GNSS receiver board

The Namuru[1] receiver project was started in mid 2004 by the School of Surveying and Spatial Information Systems, at University of New Wales, Australia. The purpose was to develop a FPGA-based GNSS receiver that could become a research and developing platform for the GNSS community.

The result is the Namuru V2 GNSS receiver, a low cost platform ideally suited for research and development of GNSS receivers. The receiver features an Altera Cyclone™ II FPGA with embedded memory and phase locked loop circuitry. The receiver has been designed with a wide range of features to allow a number of different GPS related applications. The Namuru receiver application include many research opportunities such as: developing GNSS designs for integration with other functions in an FPGA on a portable device, a convenient platform for rapid-prototyping new ideas for GNSS receiver designers, and an accessible tool to assist teaching in GNSS [29].

These applications make the Namuru platform very appropriate for this project. The flexibility provided by the FPGA allows to rapidly implement and test changes in the GPS receiver architecture. However, the use of this platform instead of another GPS development platform is because it was already available for its use in the Remote Sensing Lab.

Figure 4.2 shows a block diagram of the main Namuru V2 GNSS receiver board functional blocks. These blocks or elements will be detailed in the following sections. Additionally, Figure 4.3 shows a picture of the board with the location of their major elements. As it

[1]The word Namuru originates from the language of the Iora tribe that occupied parts of south-eastern Sydney (including the University of NSW) before the arrival of the British. This was the Iora word for 'finding the way'.

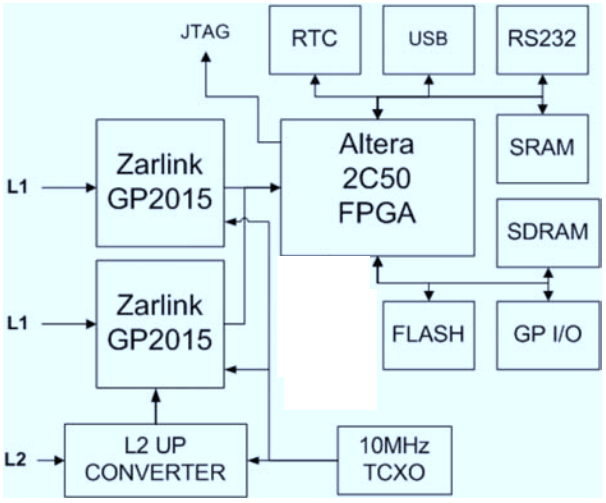


Figure 4.2: Namuru V2 GNSS block diagram [41]

can be seen, the core of the board is the FPGA device providing plenty of capacity for a possible soft-core processor, a GNSS tracking module and other custom logic. Surrounding the FPGA are IO support chips, various memory chips, a configuration controller, the power supply, etc. Under the metal screening cans there are two low noise RF front-ends, build around the Zarlink GP2015 chip [28].

Considerable care was taken in the design of the board to ensure that RF noise generated by the digital circuitry did not appear in the frequency bands used in the signal down conversion process [30].

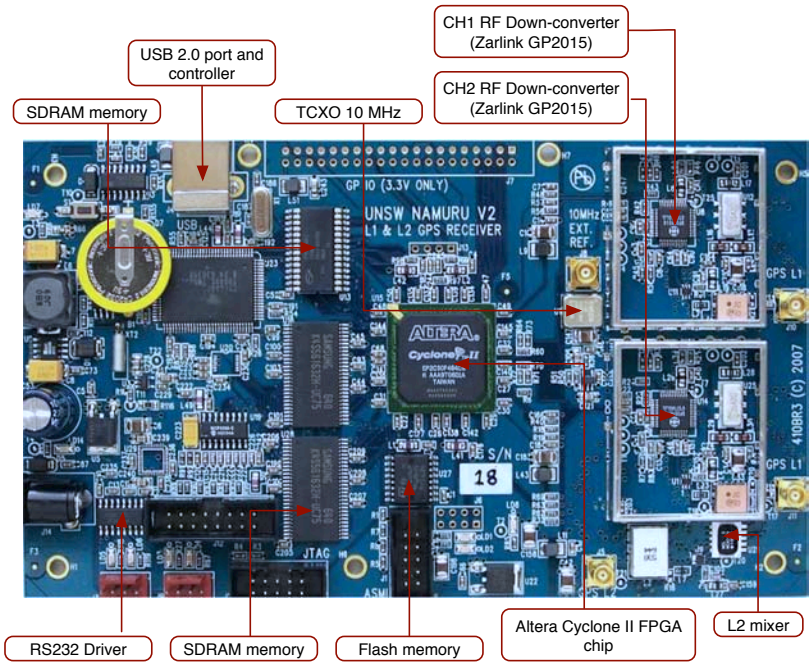


Figure 4.3: Namuru V2 GNSS board major components

4.2.1 L1 RF-Front end: The Zarlink GP2015

The Namuru board includes two Zarlink GP2015 RF down-converters. These chips implement all the RF-front end functionalities. In this case the IF chain is limited to a maximum bandwidth of 2 MHz [41, p. 18]. Both converters are supplied with a 10 MHz reference signal from a common temperature compensated crystal oscillator (TCXO). To reduce signal interference and increase isolation between RF channels, each down-converter is housed in a separate electromagnetic shield.

The Zarlink GP2015 is a small format RF Front-end for GPS receivers [28]. It is suited for size-critical applications and it is designed to operate from either 3 or 5 V supplies.

The GP2015 receives the L1 C/A signal at 1575.42 MHz transmitted by the GPS satellites. The signal is converted to an intermediate frequency (IF) of 4.309 MHz, using a triple down-conversion. The 4.309 MHz IF is sampled using a 2-bit sign/magnitude quantizer. Figure 4.4 shows a detailed block diagram of the GP2015.

The high RF input compression point of the GP2015 means that with subsequent IF filtering it is possible to reject large out of band jamming signals. As it has been mentioned before, the GP2015 divides the down-conversion process in three stages:

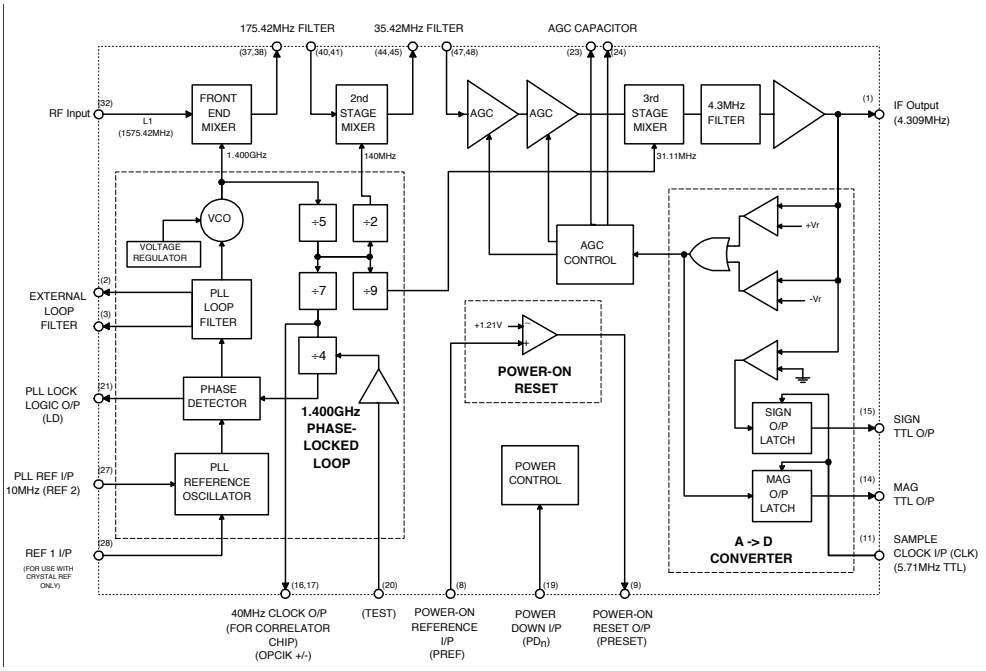


Figure 4.4: Block diagram of the Zarlink GP2015 front-end [28]

On the first stage, the on-chip PLL generates the first local-oscillator frequency at 1400 MHz. The output of the front-end mixer at 175.42 MHz can then be filtered before being applied to the second stage. The second stage contains further gain and a mixer with a local oscillator signal at 140 MHz giving a second IF at 35.42 MHz. The signal from stage 2 is passed through an external filter with a 1dB bandwidth of 1.9 MHz. The performance of this filter is critical to system performance. The output of the filter then feeds the main IF amplifier. This includes 2 AGC amplifiers and a third mixer with a local oscillator signal at 31.111 MHz giving a final IF at 4.309 MHz. There is an on-chip filter after the third mixer that provides filtering centered on 4.309 MHz.

The IF output is fed to a 2-bit quantizer which provides sign and magnitude (MSB and LSB) outputs. The automatic gain control (AGC) loop is controlled by the magnitude data, such that on average the magnitude but is set (high) 30% of the time. The sign and magnitude data bits are latched by the rising edge of an input sample clock. This sample clock needs to be provided from outside of the GPS2015. Thus the sample frequency can be chosen by the designer. However, it is recommended to use 5.714 MHz (=40/7) which results in a output digital signal with an IF centered on 1.405 MHz.

Table 4.1 summarizes the gain values and noise Figure of the GP2015 of the first down-

Table 4.1: Gain and noise factor for the two first down-conversion stages in the GP2015 [28], as well as the equivalent noise temperature.

		Value	Comment
Gain			(typical values)
- 1-stage	G_2	18 dB	
- 2-stage	G_3	27 dB	
NF			
- 1-stage	F_2	9 dB	
- 2-stage	F_3	n/a	
Front-end Temp.			
- 1-stage	T_{IF}	2014 K	$T_{IF} = 290(10^{F_2/10} - 1)$
- 2-stage		n/a	

conversion stages. More details of the GP2015 features can be found in [28].

4.2.2 L2 Up-converter

Additionally, the Namuru board allows the reception of GPS L2 band signals thanks to an included up-converter chain. This up-converter chain takes the L2 signal at 1227.6 MHz and mixes it with a 347.82 MHz signal generated by a frequency synthesizer chip, an ADF4360-7 from Analog Devices Inc. [31]. The reference signal used by this chip is from the common 10 MHz TCXO. The end result is that the L2 signal will now appear at the input of the second RF down-converter channel as an L1 signal at 1575.42 MHz. The L2 reception capabilities of the Namuru board are out of the scope of this document. For more information see [41, p. 18].

4.2.3 FPGA Introduction

In the Namuru board, the data signal processing and control functionalities have been implementing within a field-programmable gate array (FPGA) device. At the highest level, FPGAs are reprogrammable silicon chips. Using prebuilt logic blocks and programmable routing resources, it is possible to configure these chips to implement custom hardware functionality. Digital computing tasks are developed in software and compiled down to a configuration file or bitstream that contains information on how the components should be wired together. The FPGA configuration is generally specified using a hardware description language (HDL). Thus, FPGAs can be easily used to implement any logical function that an application-specific integrated circuit (ASIC) could perform. In addition, FPGAs allow

partial and multiple re-configuration of the portion of the design which offer advantages for research and test new designs and applications.

FPGAs contain programmable logic components called "logic elements", and a hierarchy of reconfigurable interconnects that allow to change the way in which the blocks are wired together. Logic elements can be configured to perform complex combinational functions and, in most FPGAs, these blocks also include memory elements, which may be complete blocks of memory.

The FPGA-based GPS receiver has grown significantly in popularity due primarily to the high circuit density achievable on a relative small programmable chip at a low cost [19, pp. 13-14]. The advantage of FPGAs systems is that the resulting chip runs with the speed of hardware, but it maintains the ability to be programmed like software. On the other side, the hardware can only be efficiently programmed for certain tasks; which supposes a disadvantage since many functions remain difficult to implement at all. For example, performing GPS signal tracking in an FPGA is not difficult as it consist primarily of functions well-suited to generic hardware gates. However, complicated control logic, which is often present in navigation softer control architecture, is not efficient to perform in an FPGA; therefore, a separate control processor is usually used. Another task that is well-suited to FPGAs is performing fast Fourier transforms. Fast and abundant parallel correlators and FFTs benefits need to be considered to increase in power and space that often come with an FPGA. The power issues need to be considered carefully as FPGAs usually tend to draw more power than ASICs.

4.2.4 Altera Cyclone II FPGA

The FPGA included within the Namuru board is a Cyclone™ EP2C50F484C8 device form Altera. The Cyclone™ II FPGAs are built from the ground up for low cost and to provide a 90-nm cost-effective embedded processing solution [43, pp. 5-6]. The FPGA is housed in a 484 pin Ball Grid Array (BGA) package. This device provides 50,528 logic elements, 594,432 bits of RAM, 86 18x18 bit multipliers and 4 phase locked loops. The FPGA is powered with two separate supplies: + 3.3 V for general purpose inputs and outputs and +1.2 V for the FPGA core logic. Table 4.2 summarizes the included Cyclone EP2C50 features.

4.2.5 Namuru on-board memory

The Namuru board includes 3 different memory modules:

SDRAM Along with the FPGA, the Namuru board includes two K4S461632E SDRAM chips from Samsung Electronics [32]. These 16 bits wide single data rate SDRAM

Table 4.2: Altera Cyclone II EP2C50 main features overview

Feature	Figures
LEs	50,528
M4K RAM Blocks (4 kbits + 512 Parity Bits)	129
Embedded Memory (Kbits	581
18-Bit x 18-Bit Embedded Multipliers	86
PLLs	4
Maximum user I/O Pins	450

chips are arrange to allow a possible full 32 bit wide data path[41, p. 19]. The SDRAM controller function needs to be provided. In this case it is provided by a logic block which is located inside the FPGA. One of the phase locked loops in the FPGA is dedicated to providing the SDRAM refresh block with the correct phase shift.

SRAM A single CY62256V25 from Cypress Semiconductors 32K by 8 bit wide SRAM chip is available for slow speed operations such as storing various parameters required fro start up [33]. This SRAM consumes extremely low power when is not in operation and can be optionally powered from a battery backup system to maintain data when main power is off [41, p. 19].

FLASH A single M25P64 64 Mbit serial SPI flash module from ST Semiconductors [34] can be used for storing the FPGA logic configuration or a possible FPGA core IP CPU runtime software.

4.2.6 Namuru board ports: RS232 and USB 2.0, and board clock signals

The Namuru board includes 2 serial ports through a ST3232ED RS232 level translator from ST Semiconductors [35]. This device is equipped with some electrostatic discharge protection. Additionally, one USB port is provided. The Cypress CY7C68013A single USB 2.0 transceiver provides all the functionalities needed for operation [36]. This chip is connected through a 16 bit path to the FPGA and acts as a slave USB devoid to a PC, allowing high speed USB communication (480 Mbit/s) [41, p. 20]. The software for the USB controller is stored in a Microchip 24LC128 I^2C [37] serial flash chip. This chip is connect directly to the CY7C68013A to allow the controller to boot on power up.

As it was mentioned previously when describing the Zarlink GP2015 configuration, the Namuru board includes a single 10 MHz TCXO which is used to feed both RF down-converters and the L2 up-converter synthesizer as shown in Figure 4.5. Notice that this feature is very

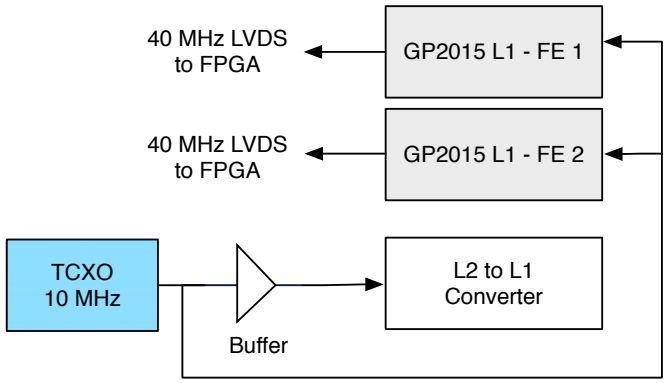


Figure 4.5: Namuru V2 GNSS clocks

significant to mitigate synchronization problems between both front ends, allowing the implementation of applications that compare the received signal in both front ends for a given time instant.

There is also a DS1338 chip from Dallas Semiconductors [38] in order to provide the real-time function clock. The clock chip is connected to the FPGA using a 2 wire I^2C interface. Battery power can be supplied to the clock chip while power is absent.

4.3 Namuru development environment

Along with the Namuru receiver board, the development platform also includes a set of software tools for digital design purposes. The main software tools that are needed to implement new hardware and software designs, as well as to download those designs into the integrated FPGA, will be introduced. Most of these design tools use a description language, VHDL, that will be briefly presented first.

4.3.1 Introduction to VHDL

VHDL is a technology and vendor independent hardware description language. The code describes the behavior or structure of an electronic circuit which a compliant physical circuit can be inferred by a compiler. One of their main applications is to synthesize digital circuits on FPGA chips.

VHDL stands for VHSIC hardware description language. It was the first hardware description language standardized by the IEEE, through the 1076 and 1164 standards [39, p. 492].

VHDL allows circuit synthesis as well as circuit simulation. The former is the translation of a source code into hardware structure that implements the specified functionalities; the latter is a testing procedure to ensure that such functionalities are achieved by the synthesized circuit.

In VHDL, it is possible to define different levels of abstraction depending on the depth of circuit functionality and components available to the designer, and complexity of the circuit so as to make the circuit design manageable [40, pp. 12-13]. There are 3 levels of system abstraction of digital design:

Behavioral/Algorithmic: It is the highest level of abstraction that describes a systems in terms of what it does or how it behaves instead of in terms of its components and the interconnection between them. A behavioral description specifies the relationship between the input and output signals Sequential statements are executed in the sequence in which they are specified. VHDL allows both concurrent as well as sequential signal assignments.

Dataflow: This representation describes how a data moves through the system. This is typically done in terms of data flow between register through buses and interconnections. So this level of abstraction is close to register transfer level (RTL) of system representation. The data flow model makes use of concurrent stamens that are executed in parallel as soon as the data arrives at the input. Dataflow is a concurrent, event-driven approach where logic equation is executed only when any of its input value changes.

Structural/Physical: This level represents the lowest level of abstraction. It describes a system as a collection of gates and components that are interconnected to perform a desired function. It represents a physical description of the circuit. It is possible to compare a structural description to a schematic of interconnected logic gates.

Since there is a more appropriate description level in each case, during the development of this project, the three types of VHDL are used in a mix of different modules.

Next, the Quartus II from Altera is introduced as the electronic design automation (EDA) tool for VHDL synthesis used in this project development. Additionally, ModelSim simulation tool from Mentor Graphics is presented. ModelSim has been selected to perform the simulation and debugging tasks.

4.3.2 Quartus II software

Altera Quartus II is a design software program that allows to create designs for FPGAs, HardCopy Application ASICs and complex programmable logic devices (CPLD). This pro-

gram features a graphic user interface (GUI) to facilitate the design creation [42, pp. 10-11].

Quartus II contains many tools to prepare the VHDL or Verilog[2] code and structures and to program using them the FPGA board. It includes many Intellectual Property (IP) cores[3] implemented by Altera which are directly available for use.

The version used in this project development was the Quartus II 7.2 Student Edition. This software was downloaded from the Altera's web site along with the appropriate license for academical purposes. Both, the software and the required license are available for free. Although the most recent version of this software is the v. 11.0, 7.2 version was selected to ensure compatibility with the existent Namuru VHDL code. This code was implemented using that version, and by using the same version it was more likely to do not find inter-version compatibility issues.

4.3.2.1 FPGA design flow with Altera's Quartus II

Figure 4.6 presents the design flow followed when using Altera's Quartus II. The design entry can be graphical or text-based. Once the design is finished, the first step performed by the Quartus software is a formal check that eliminates syntax errors or graphic design rule errors. The next step is called function extraction. In that step the basic design information is extracted from the design and written in a functional netlist.

The netlist allows a first functional simulation of the circuit and to build an example data set called testbench for later testing of the design with timing information. If the functional test is not passed, starting back with the design entry again in needed. If the functional test is satisfactory, the next step is to proceed with the design implementation, which usually takes several steps and also requires much more compile time than the function extraction.

After finishing the design implementation the circuit is completely routed within the FPGA, which provides precise resource consumption data and allows the designer to perform a simulation with all timing delay information as well as performance measurements. If all these implementation data are as expected, it is possible to proceed with the programming of the actual FPGA; if not it is necessary to start with the design entry again and make appropriate adjustments or changes in the design.

[2] **Verilog** is another hardware description language (HDL). Verilog HDL, usually an alternative to VHDL, is most commonly used in the design, verification, and implementation of digital logic chips at the register transfer level (RTL) of abstraction. It is also used in the verification of analog and mixed-signal circuits.

[3] An **intellectual property core**, IP core, or IP block is a reusable unit of logic, cell, or chip layout design that is the intellectual property of one party. The term is derived from the licensing of the patent and source code copyright intellectual property rights that subsist in the design.

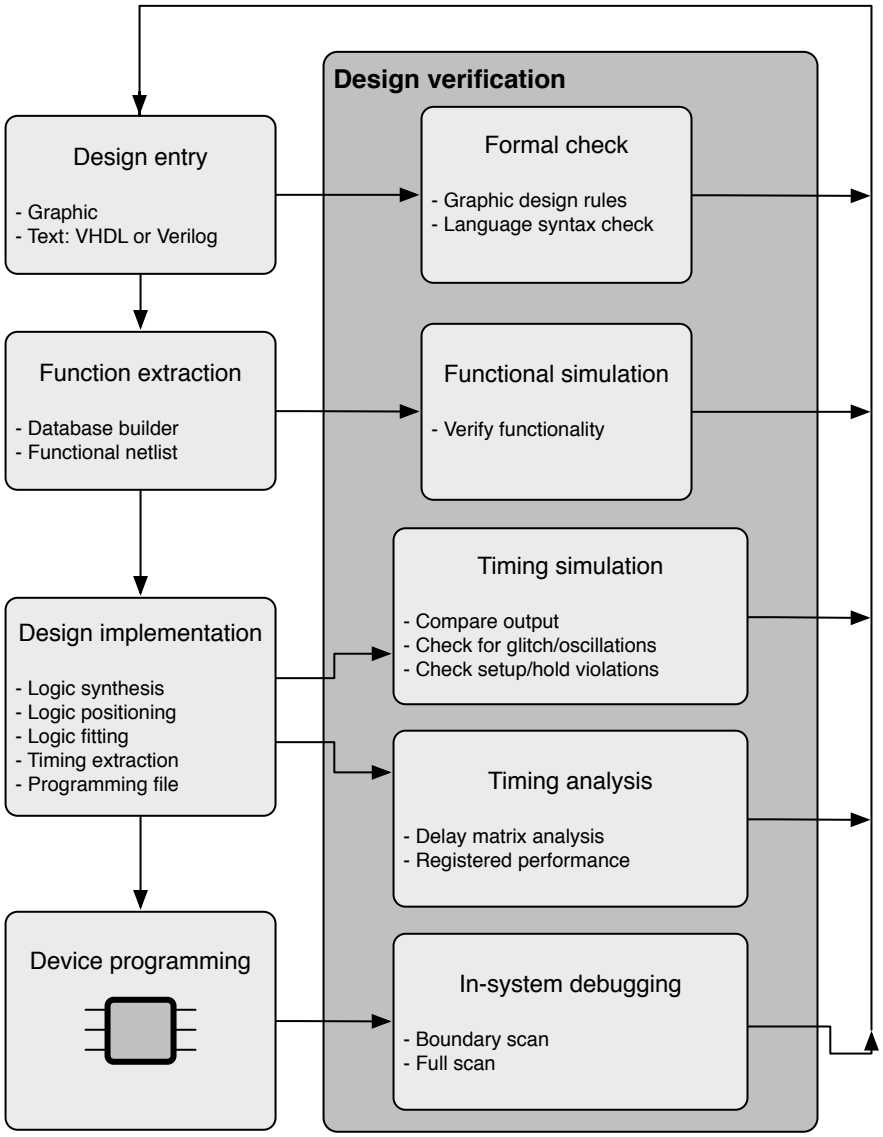


Figure 4.6: Quartus II design flow diagram

Using JTAG interface[4] of modern FPGAs it is possible to directly monitor data processing on the FPGA: it is possible to read out just the I/O cells or the internal flip-flops for debugging purposes. If the in-system debugging fails it is needed to go back, once again, to the design entry.

The decision of whether to work within a graphical or a text design environment is a matter of personal taste prior to experience. A graphical representation can emphasize the highly regular dataflow associated with DSP algorithms. The textual environment, however, is often preferred with regard to algorithm control design and allows a wider range of design styles. Specifically, for Altera's Quartus II, it seemed that with text design more special attributes and more-precise behavior can be assigned in the designs.

4.3.2.2 SOPC Builder

Quartus II includes an associate software called System on a Programmable Chip Builder that automates connected soft-hardware component to create a complete computer system that runs on a FPGA chip [45]. SOPC Builder incorporates a library of pre-made components (including the Nios II soft processor, memory controllers, interfaces, and peripherals) and an interface for incorporating custom ones. Interconnections are made through the Altera's Avalon Bus [44]. Bus arbitration, matching and even clock domain crossing are all transparently handled when SOPC Builder generates the system. A GUI is the only thing used to configure the soft-hardware components (which often have many configuration parameters) and to specify the bus topology. The resulting "virtual" system can be downloaded to the FPGA using the Quartus II corresponding utility.

4.3.2.3 Nios II Processor

SOPC Builder previously introduced can be used to easily integrate IP cores in a design or to generate embedded systems within the FPGA. The Namuru-GPL GPS receiver that will be introduced in the next chapter uses a embedded Nios II processor solution in order to perform control tasks and compute the navigation solution.

A Nios II processor system is equivalent to a microcontroller that includes a CPU and a combination of peripherals and memory on a single chip. The term "Nios II processor system" refers to a Nios II processor core, a set of on-chip peripherals, on-chip memory, and

[4] **Joint Test Action Group (JTAG)** is the common name for what was later standardized as the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. It was initially devised for testing printed circuit boards using boundary scan and is still widely used for this application.

interfaces to off-chip memory, all implemented in a single Altera chip. The Nios II processor is a general-purpose RISC processor core, providing:

- Full 32-bit instruction set, data path, and address space 32 general-purpose registers
- 32 external interrupt sources
- Single-instruction 32x32 multiply and divide producing a 32-bit result
- Dedicated instructions for computing 64-bit and 128-bit products of multiplication
- Floating-point instructions for single-precision floating-point operations
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Hardware-assisted debug module enabling processor start, stop, step and trace under an specific software development environment

Figure 4.7 shows a detailed core block diagram of the Nios II processor architecture. More details on Nios II architecture, features and design procedures can be found in [46] and [47].

Nios II version 7.2 was used in this project. Notice that Nios II is an Altera proprietary IP core, and the license used limited only for academic purposes and one hour of use while the FPGA need to stay connected to the host PC.

4.3.2.4 Nios II Embedded Design Suite (EDS)

If the design uses a Nios II processor, it becomes clear that it should be necessary to develop the software that the processor will execute during its operation. Here is where Altera provides a development environment, the Nios II EDS to easily develop that software.

The Nios II EDS is a consisted software development environment that works for all Nios II processors systems. The Nios II EDS running on a host computer, the FPGA, and a JTAG download cable (such as an Altera USB-Blaster™) allows to write programs for and communicate with any Nios II processor system, such as the one implemented in the Namuru solution. The Nios II processor's JTAG debug module provides a single, consistent method to connect to the processor using a JTAG download cable. Accessing the processor is the same, regardless whether a device implemented only a Nios II processor system, or whiter the Nios II processor deeply in a complex system. Therefore, the interface mechanism for the embedded processor are already provided.

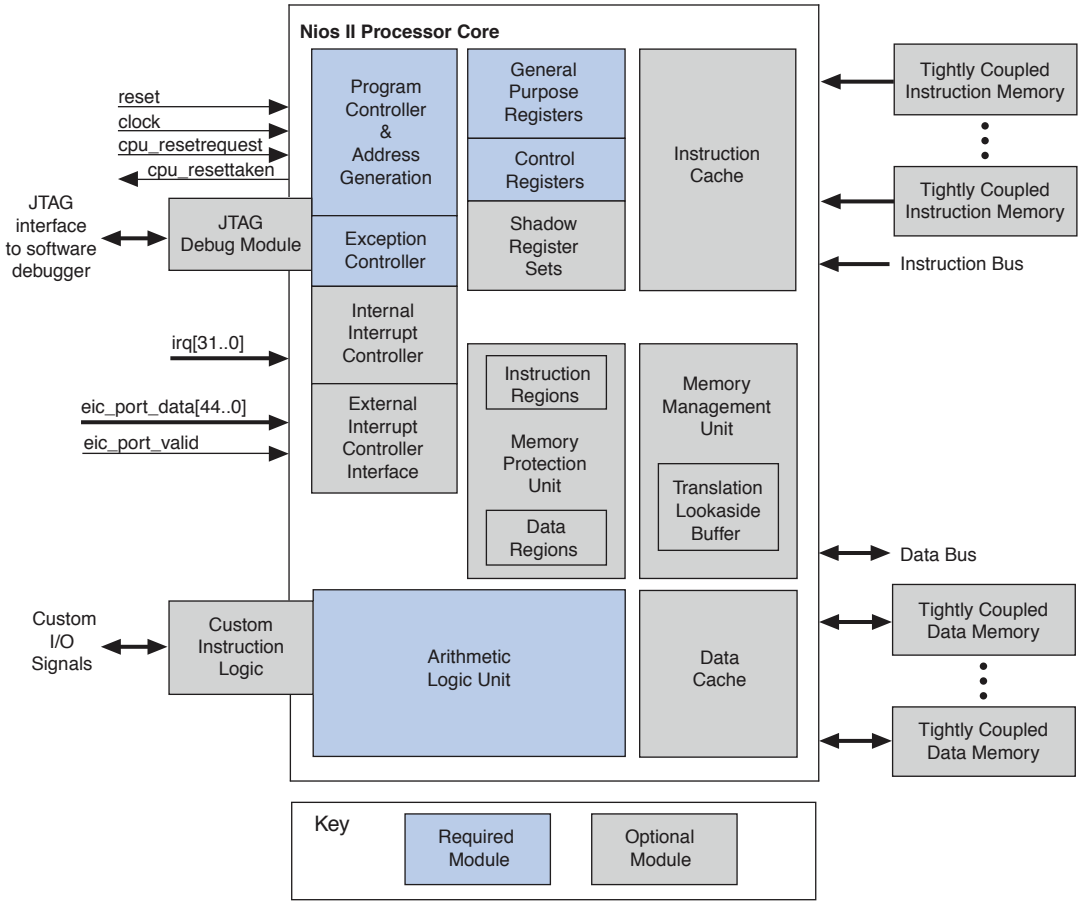


Figure 4.7: Block diagram of the Nios II processor core [46]

The Nios II EDS 7.2 used, includes proprietary and open-source tools (such as the GNU C/C++ tool chain) for creating Nios II programs. The Nios II EDS automates board support package (BSP) creation for Nios II processor-based systems, eliminating the need to spend time manually creating those BSPs. The BSP provides a C/C++ runtime environment, insulating the designer from the hardware in the embedded system. Altera BSPs contain the Altera hardware abstraction layer (HAL) an optional RTOS, and device drivers.

All the C files belonging to the Namuru design have been compiled and downloaded into the Nios II chip using Nios II EDS software. The Nios II EDS presents a new project wizard used to automate the set-up of the C application project and system library projects. This project wizard has been used to setup a new project including all the Namuru files and necessary libraries. More details on the configuration and simulation procedures can be found in [47] and [41].

4.4 Chapter summary and conclusions

This chapter has introduced first, the main tasks performed by a typical GPS receiver front end, and lately a description of the Namuru V2 GNSS development platform.

The Namuru development platform can be divided in two parts: hardware and software tools. The hardware part is the Namuru V2 GNSS board. This board represents a low cost development platform specially interesting for research and development of GNSS receivers. The Namuru board is composed by two RF independent front ends connected to an Altera FPGA with embedded memory and additional peripherals. The main features of the most important board components have been briefly depicted in this chapter. Next, the main software tools have been presented together with a brief introduction to VHDL design. The main software tools used are: Quartus II suite, for digital design, hardware synthesis and device programming of the Namuru board. Within Quartus II, the SOPC builder software has been explained. This software allows to easily interconnect IP cores, such as the Nios II embedded softcore processor, for complex system design.

The Namuru platform description becomes necessary to understand the Namuru-GPL GPS receiver, as well as its proposed modifications, that are presented in the following chapters.

Chapter 5

The Namuru-GPL

Namuru Open Source Software: Namuru-GPL The Namuru Open Source Software project started from a previous open source work, the GPL-GPS software developed by Andrew Greenberg [50, p. 48]. This software was developed to target the Zarlink chipset GP2040, which contains a correlator and an ARM7 processor, and it was designed on the eCos real time operating system. The Namuru Open Source Sotware project ported both the eCos and the GPL-GPS software to the Namuru GNSS GPS receiver platform, so adapting the GPL-GPS to a different but similar correlator, and to a completely different processor. The result of this process was the Namuru-GPL software receiver that is described in the following sections.

The last sections of this chapter describe the modifications done over the original Namuru-GPL design and implementation. These modifications were done in order to adapt the receiver for using both on-board available radio front-ends and being able to successfully detect and track both, direct and reflected, GPS received signals. The use of both front-ends pursues one of this project main goals: to adapt the Namuru GPS receiver for GNSS reflectometry applications.

5.1 Namuru-GPL structure overview

The structure of the Namuru-GPL software-defined receiver can be divided into three units: the front-end, the baseband processor and the general purpose microcontroller. This receiver belongs to the so called software-defined receiver type because most of the operations that are necessary to obtain the user position are executed by a software that runs into the general purpose microcontroller. As it has been introduced in previous chapters, the front-end

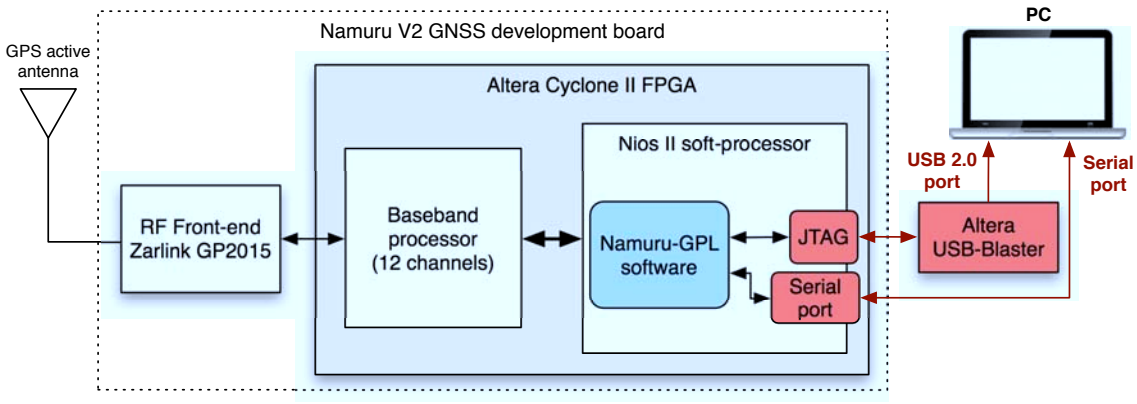


Figure 5.1: Namuru-GPL receiver components and its connections with a controller PC

unit contains a chain of filters, amplifiers, multiple-stage down-converters and at the end, a Analogue to Digital Converter (ADC). The purpose of this unit is to produce a digital stream that represent the sampled GPS signal once it has been filtered, amplified and converted to a lower frequency. This digital stream is then passed to the baseband processor, that mixes it with the local generated replica and accumulate the results into different registers, to make them available to the general purpose microcontroller. The software that runs on the general purpose microcontroller uses these values to understand if a valid GPS signal is present, and if it is so, it controls the baseband processor operation in order to extract the data necessary to compute the user position. Figure 5.1 shows the Namuru-GPL component block scheme, as well as its connections with the PC used to program, debug and receive the obtained data from the Namuru receiver.

As it was discussed in previous chapters, it is possible summarize operations carried out by the Namuru GPS receiver software using the next points:

- Serial search acquisition: the software searches for a valid GPS signal into the data stream; each GPS signal coming from different satellites can be identified because of different spread sequence used (C/A codes), and for each of them it is necessary to try different carrier frequencies, since the relative motions between the receiver and the satellite makes the frequency changes due to the Doppler effect, as it has been presented in previous chapters.

- Tracking: once a valid GPS signal is found, the software has to extract data necessary to compute user position; to accomplish this it needs firstly to clear the signal from the carrier and the spreading sequence, by controlling the correlators that compose the baseband processor so it can generate a precise replica of the incoming carrier and

spreading sequence. The operation of tuning the local generated signals so they can replicate as closely as possible the incoming signals is done by the tracking loops.

- Pseudoranges computation: once the receiver has extracted the data and the tracking loops are in lock, the software can compute the pseudoranges for each GPS acquired signal.
- Receiver position computation: user's position is obtained from pseudoranges the trilateration method as it has been introduced in chapter 2.
- Display information: once the position and pseudo-ranges are obtained, they must be shown to the user; this is done sending the data to a PC using a serial link.

All this tasks are performed or controlled by the general microcontroller implemented inside the FPGA. In the Namuru-GPL this microcontroller is the soft-processor Nios II. These processor's features have been introduced in the previous chapter. Nios II processor runs the eCos real-time operating system[1] (RTOS) to perform, manage and synchronize those mentioned tasks.

5.2 eCos RTOS

eCos is a "runtime system", a static pre-compiled library that application links against during the compile time. The library provides all of the functions of a standard operating system: startup, RTOS kernel, scheduler, etc. [49]. eCos uses a hardware abstraction layer (HAL) to keep its kernel independent of the underlying system architecture, in this case the Nios II RISC architecture. Calls that involve hardware (turning off the interrupts, for example) are defined by the architecture template into direct hardware calls.

5.3 Baseband Processor

A reference design for baseband processing was developed in VHDL [48]. The design follows the conventional GPS receiver signal processor solution. This represents an attempt to make it very accessible to a wide range of users and to provide an easy path for modifications. The tracking module is composed by 12 independent channels.

Figures 5.2 and 5.3 show this 12 channel architecture, as well as a detailed tracking channel representation. Each tracking channel is composed by three complex code correlators (arms), each separated by half a C/A chip.

[1] A **RTOS** is an operating system intended to serve real-time application requests.

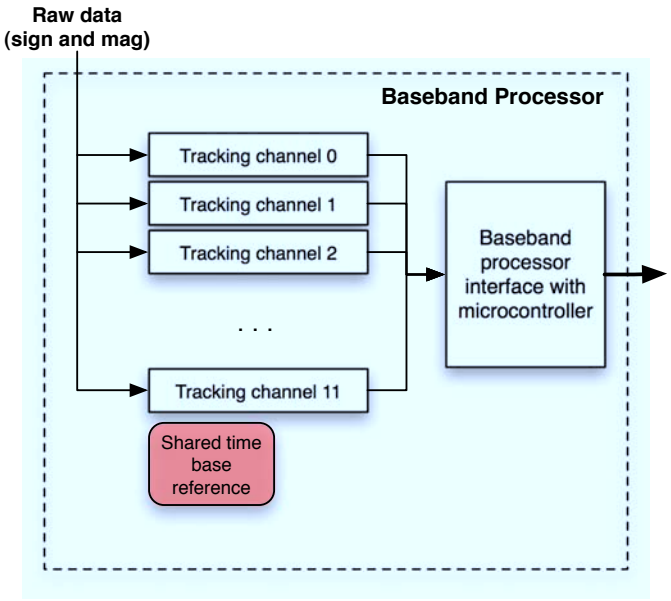


Figure 5.2: Tracking module overview (12 channels)

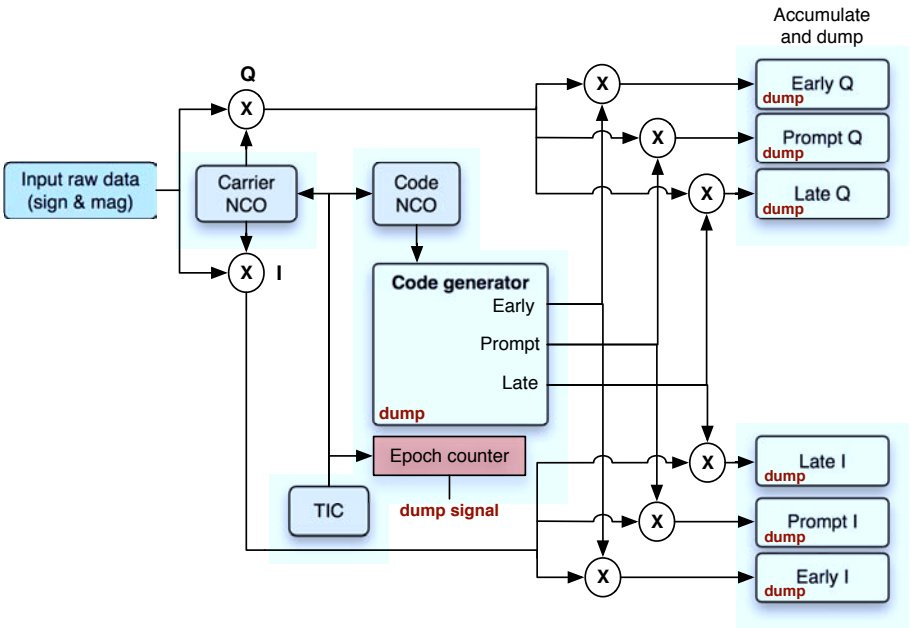


Figure 5.3: Tracking channel details

The tracking module is connected to the Altera Nios II [46] embedded soft-core CPU, as an Altera SPoC Builder peripheral [45] via the Altera Avalon bus [44]. The baseband processor is defined as a memory-mapped peripheral. Setup and control of the module is performed via writes to relevant registers, and data is collected via reads from relevant registers. Mumford provides more details in [48] on register operations and mappings, while a tracking loop description can be found in previous chapter 3.

5.4 Namuru GPS software operation

The Namuru GPS software uses a foreground/background (interrupt/mainline) architecture. The foreground task is driven by a 1 ms timer interrupt on the Nios II microprocessor. All time dependent functions - satellite navigation message decoding, and measurement processing - are associated to different interrupts and performed by different interrupt service routines. Navigation algorithms and display code run as the mainline task with communication between tasks accomplished using global variables.

In order to improve the software performance using the Nios II processor, each task has been modularized into its own code and header file, to clarify and prioritize the tasks. Each task usually has one thread and one data structure associated with it, making it a more modular interface for task additions and/or inserting external communication links. Figure 5.4 shows the different tasks and the existent data flow. A description of each main receiver's software task is presented next:

Channel allocation (allocate.c): The Namuru-GPL baseband processor has 12 channels and so is capable of tracking up to 12 satellites. Namuru-GPL implementation uses a boot strap algorithm for channel allocation. It tries every PRN code in groups of 12 and searches a very wide Doppler region since there is no information of the possible satellites into view of the receiver, neither the clock error, velocity or position.

Tracking (tracking.c): The tracking routine is a state machine that is triggered each millisecond by the baseband processor correlator signals when a new set of data is available for the Nios II processor. This signal causes the interrupt routine to be executed, so that the state machine is updated. Figure 5.5 shows the state machine.

The interrupt routine reads the data from the correlator, and for each channel it executes the actual state:

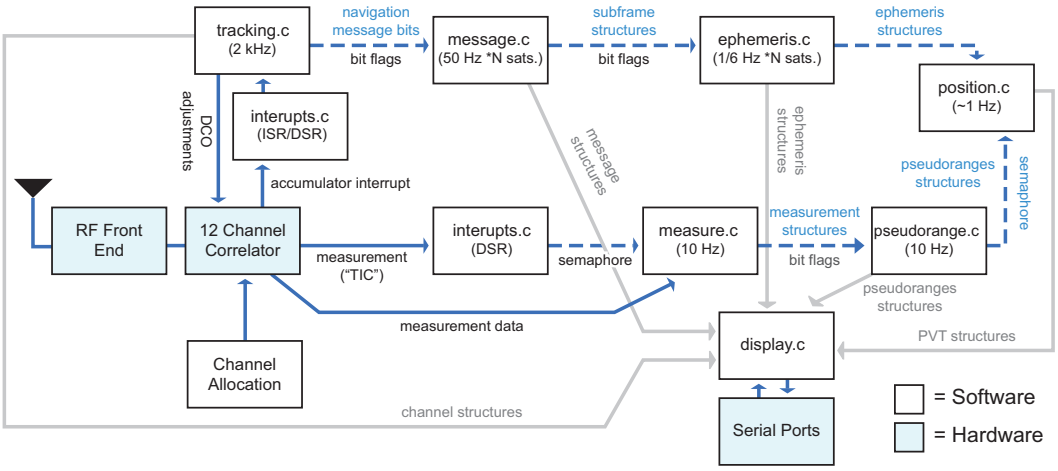


Figure 5.4: Namuru-GPL software flowchart

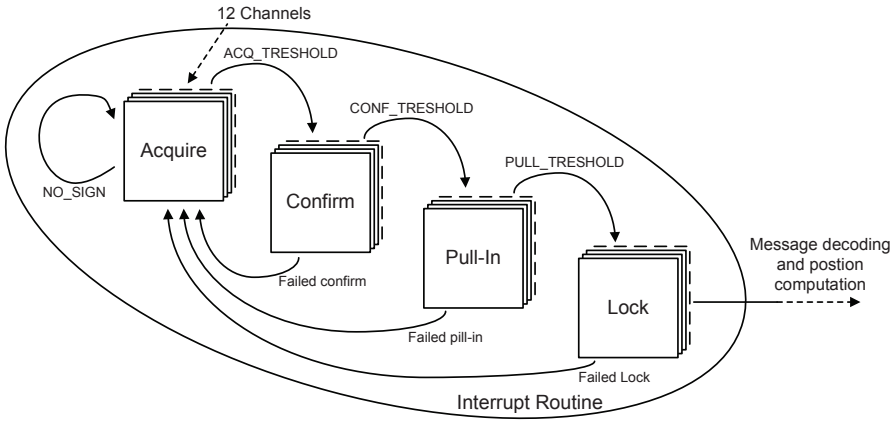


Figure 5.5: Namuru-GPL tracking routine state machine

Acquire: it searches for different code shifts and carrier frequencies within the allocated PRN code; if no signal is found for all code shifts and carrier frequencies the allocated PRN code is change by the allocate thread.

Confirm: if a peak in the correlator data is found, it could indicate the presence of signal; the local replica code shift and carrier frequency used at the moment of the peak are a coarse estimation of the incoming signal code shift and carrier frequency; the presence of signal must be confirmed in order to proceed with tracking.

Pull-in: if the presence of signal is confirmed, the tracking process starts with the pull-in routine, in order to find better estimation of code shift and carrier frequency; this routine contains the implementation of the two tracking loops, one for code and one for carrier, set to be capable to track when the phase error between the local replica and incoming signal is high. The routine also starts to look for message bits to synchronize the receiver with the incoming message.

Lock: when the phase error is reduced the software switches to a PLL and a DLL which have reduced bandwidth; this makes them slow, but greatly improves the loops noise rejection. When the lock is achieved and there is synchronization with message bits, message thread can start to decode information to compute user position.

Message decoding (message.c): The message routine is periodically executed in order to parse the incoming navigation bits from tracking routine into words and subframes in message structure. Thus, this function finds the preamble, the TLM and HOW in the navigation obtained data and synchronize it to the navigation message.

Ephemeris obtention (ephemeris.c): Once the message routine has obtained valid navigation data structured in message frames and subframes, the ephemeris routine is executed in order to read the ephemeris data contained in the received data. These read ephemeris values are converted then to floating point values and stored within the Nios II memory.

Measure routine (measure.c): The measure routine retrieves the measured values from the baseband processor's correlators. These measured values are timing information, such as the measured code phase, needed to compute the pseudorange values.

Table 5.1: Namuru-GPL serial connection configuration parameters

57600 bps
8 bit/sample
no parity check
1 stop bit
no flux control

Table 5.2: Namuru-GPL available display modes their associated keys

t : tracking
m : message
e : ephemerides
r : pseudoranges
p : position
c : screen cleaning
s : stop information updating

Pseudorange obtention and position calculation (pseudorange.c an position.c): Once valid the measure routine has obtained valid timing values, pseudoranges are periodically computed each 0.1 seconds. This is done for each receiver's channel which is successfully tracking a signal (locked state). Once the pseudoranges for 4 different satellites are successfully computed, the position routine starts to compute the receiver's position solution for those given pseudoranges and the ephemeris obtained from navigation data.

Displaying data (display.c): This routine periodically updates the information that is sent to the user using the serial port. Additionally, some debut information is also visualized using the JTAG programmer from the Nios II EDS. The serial port is used in two ways: the first one is connecting a terminal program (such HyperTerminal or equivalent) to the port to visualize the output interface implemented in the the display routine. A valid connection is obtained using the parameters presented in table 5.1.

The output interface gives general information, updated every second, about tracking, computed position, pseudoranges and ephemerides. The visualized information can be changed by pressing the keys presented in table 5.2.

The information obtained can be used to evaluate the performance of the software from a user point of view, for example to measure how long does the software take to give the first valid position. It is also possible to use these data for quantitative analysis, such as capturing the values coming from the six baseband processor's branches, at higher data rate. Data update rates up to 100 Hz has been successfully tested, while trying to acquire faster

than that rate has revealed to interfere with the receiver operations because of the high-rate interruption related with display routine.

5.5 Modifications in Namuru-GPL

The sections describes the modifications done over the original Namuru-GPL design and implementation, in order to adapt the receiver for using both on-board available radio front-ends and being able to successfully detect and track both, direct and reflected, GPS received signals. This includes, first, modifying the baseband processor design and then, modifying the Namuru-GPL software, specifically the channel allocation method and the tracking threshold values.

5.5.1 Architecture design change to use 2 front-ends

The Namuru V2 GNSS board has two independent Zarlink GP2015 front-ends driven by the same system TCXO clock. This feature allows using of one front-end to receive the direct GPS signal, while at the same time, using the second front-end to receive the reflected GPS signal. Since both front-ends are driven by the same clock, and can be equally controlled by the baseband processor, no major synchronization issues should appear.

Since both Zarlink GP2015s are connected to the on-board FPGA pins, only the modification of the baseband processor design is needed. This means modifying the baseband processor VHDL code in order to control and receiver data from 2 front-ends instead of only one.

The Zarlink GP2015 has 3 digital outputs: data sign bit, data magnitude bit and 40 MHz clock signal. The 40 MHz clock signal is used as signal clock for the baseband processor and Nios II microcontroller. On the other side, the sampling instant signal needs to be provided by the baseband processor. This sampling signal has a frequency of 5.714 (40/7) MHz and it is computed by the baseband processor with a simple frequency divider module.

The already used GP2015 will be called front-end 1, while the GP2015 that is going be connected will be referred as front-end 2. The front-end 1 is considered the main front-end. 40 MHz clock signal output from front-end 1 stills being used as system clock. Front-end 2 40 MHz clock signal will not be used. The sampling signal is the same for both front-ends and it is derived from front-end 1 clock, which assures that input signal data is read at the same time instant. This configuration ensures maximum synchronization between the both front-ends that can be obtained with Namuru V2 GNSS board implementation.

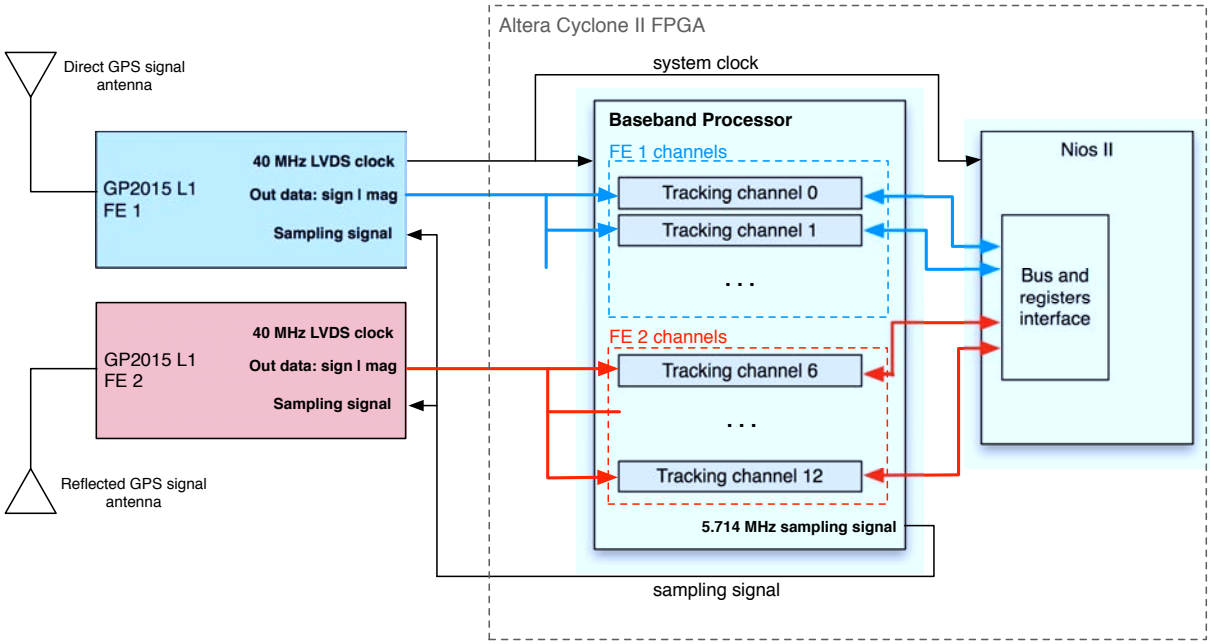


Figure 5.6: Modified Namuru-GPL system connections block diagram using both front-ends and their channel mapping in the baseband processor

The baseband processor has been modified to map the digital output stream obtained from the front-end 1 to the first 6 channels. On the other side, the front-end 2 digital output stream is connected to the other 6 channels. This design was considered in order to reduce as much as possible the changes in the original Namuru-GPL receiver design.

Other design options like using two independent baseband processors were considered, but they were discarded. Increasing the receiver's hardware complexity was not an option because: FPGA resources were limited, changes in hardware also imply changes in Namuru-GPL software, synchronization problems between baseband processors data could appear and, since the system has a considerable degree of complexity it will take a lot of time in design and verification of a new system architecture.

Figure 5.6 shows a block diagram that represents connections between both front-ends and the baseband processor, as well as the described channel mapping.

5.5.2 Changes in Namuru-GPL software: channel allocation

The previously explained changes in the Namuru GPS receiver architecture require modifications in the Namuru-GPL software. In order to fulfill those required modifications, to

tasks were performed: first, adapt the allocation routine conveniently to track the same PRN code in the direct and reflected received signals, and second, implement a manual allocation procedure in order to select the satellite (by selecting its associate PRN code) that has to be tracked in a selected receiver's channel.

5.5.2.1 Default channel allocation method

By default, the allocation routine selects one PRN code to be allocated to one of the baseband processor channels. This allocation procedure is performed in two situations:

When the receiver operation starts, each baseband processors channel needs to be associated with one satellite (a PRN code) before starting the serial acquisition procedure. Once the serial acquisition search procedure performed over one channel has finished without successful detection. In this case the allocation routine assigns a new PRN code to the channel.

In both cases, all channels are associated to different PRN codes.

5.5.2.2 Modified allocation method

The new allocation routine introduces a channel hierarchy. From baseband processor's changes, it becomes clear that now there are two different channel groups:

- First group: channels from 0 to 5 associated with front-end 1, corresponding to the direct GPS signal.
- Second group: channels from 6 to 11 associated with front-end 2, corresponding to the reflected GPS signal.

The first group allocation has not been modified and it is performed as it is explained in the previous section. On the other side, the second group allocation procedure has been radically modified. Now, each channel in the second group is directly associated to one channel of the first group, in the way that the same PRN code is selected. Figure 5.7 shows the channel relationships.

The idea behind this channel association is first detecting and tracking a satellite direct signal using one of the first group channels, and then forcing the second group channel associated with it to only try to detect and track the same satellite signal (in this case, the corresponding reflected signal). This concept reduces the number of free allocation channels to 6, which will typically duplicate the search acquisition time with respect to the original 12

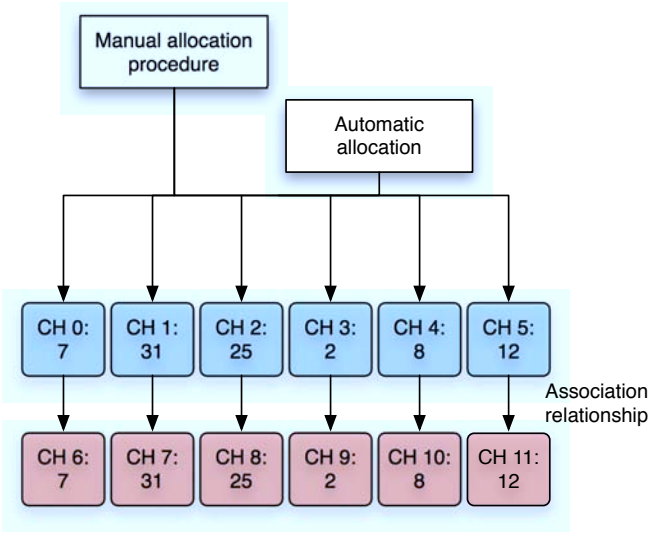


Figure 5.7: Relationships between channels associated with front-end 1 and channels associated with front-end 2

channel configuration. However, this allows monitoring the pseudoranges and signal strength associated to each channel simultaneously. Thus, once the receiver is in the lock state for a channel belonging to group 1 and its associated channel in group 2, the pseudoranges associated to direct and reflected signal could be obtained.

Because the receiver does not know the local clock bias, the Namuru-GPL needs to compute the relative pseudoranges [18, pp. 202-209], which implies that for a correct magnitude pseudorange values, a position fix in the first channel group will be needed. However, difference between the relative pseudoranges is the same that between the pseudoranges, which can be enough for applications that are interested in the difference between pseudoranges (such as altimetry applications).

5.5.2.3 Manual channel allocation

In order to speed up the acquisition process, a manual channel allocation mode has been implemented. It is common in experimental scenarios, to already know which satellites will be in sight. Thus, manually allocating one of those satellites to a channel will reduce the acquisition process time. This manual allocation mode works only for the first channel group, associated with front-end 1, this is the first 6 channels.

The manual channel allocation has been implemented by modifying the display and allocation routines. When the display mode is in tracking mode (user has previously pressed the "t"

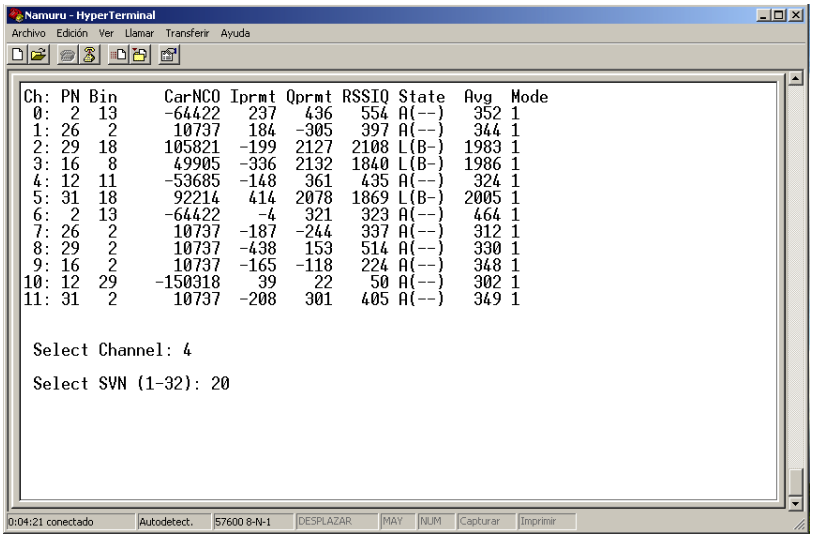


Figure 5.8: Manual channel allocation example

key), pressing enter key the user is asked about which channel allocation wants to change, and next which PRN code wants to associate to that channel. Figure 5.8 shows a screen capture of this process. The channel belonging to the second channel group associated with the modified channel will also be, as a consequence of its association relationship, reallocated.

For implementation details see allocate.c and display.c routine commented source code.

5.5.3 Acquisition and tracking treshold values

The tracking state machine presented in previous sections switches from the acquisition routine to the confirm routine if the sum of early, prompt and late power is above a certain threshold. This threshold can be found with the following considerations: the Namuru implemented front-end Zarlink GP2015 is designed to produce a 2 bit digital stream with the four levels -3, -1, +1 and +3 [28]. The automatic gain control (AGC) adjusts the ADC gain in such way the mean percentage of samples falling into each quantization level is presented in table 5.3.

The digital output stream obtained from the front-end is then mixed by the baseband processor with the locally generated carrier replicas, in-phase and quadrature, before being

Table 5.3: Mean percentage of each quantization level value at the ADC output

Value	Mean percentage
-3	15%
-1	35%
+1	35%
+3	15%

Table 5.4: Sine and cosine approximations used for locally generated carrier in-phase and quadrature replicas

In-phase	+2 +2 +1 -1 -2 -2 -1 +1
Quadrature	+1 -1 -2 -2 -1 +1 +2 +2

integrated for 1 ms. The discrete approximation used as cosine and sine signals is shown in table 5.4.

Considering the case when the input signal is +3 the product with the in-phase local replica over one cycle is

$$+3 \text{ x (In-Phase)} = +6\ +6\ +3\ -3\ -6\ -6\ -3\ +3$$

which has a mean-square value of 22.5; on the other hand, when the input is +1 the product with the in-phase local replica over one cycle is

$$+1 \text{ x (In-Phase)} = +2\ +2\ +1\ -1\ -2\ -2\ -1\ +1$$

which has a mean-square value of 2.5. Assuming a sampling frequency of 5.714 MHz, it is possible to find the mean square value of the in-phase branch integrated over 1 ms as:

$$I^2 = \left[(22.5 \cdot \frac{3}{10}) + (2.5 \cdot \frac{7}{10})\right] \cdot f_{sampling}[MHz] \cdot 1000 \tag{5.1}$$

thus, and considering that the quadrature branch will have an equivalent behavior, total power obtained will be

$$I^2 + Q^2 = 97142 \rightarrow \sqrt{I^2 + Q^2} = 311 \tag{5.2}$$

This is the expect value read from early, prompt and late branches when non GPS signal is present. In order to track signals that are 6 dB over the noise floor, the expected value of

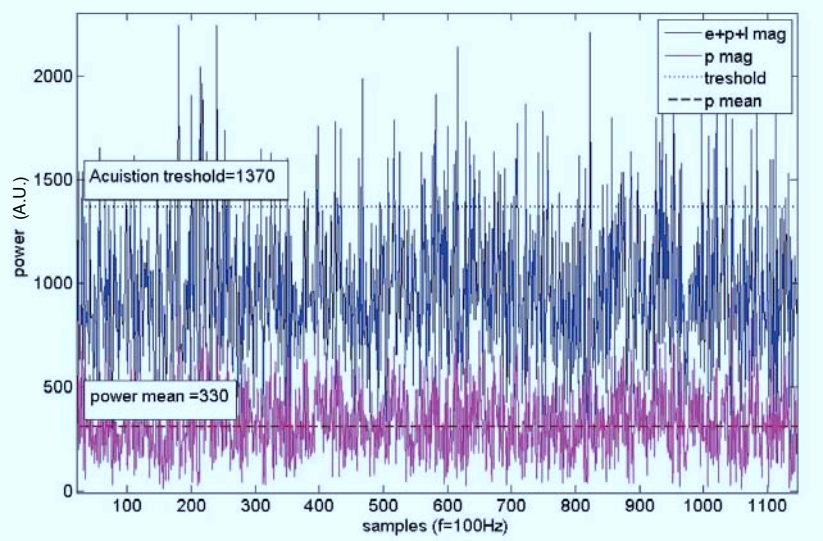


Figure 5.9: Early + Prompt + Late power (blue), prompt power (purple), prompt power mean (brown) and acquisition threshold (black), no GPS signal in-view; sample frequency 100Hz [48]

$I^2 + Q^2$ will be 386729. Considering this, the threshold for one single branch (early, prompt or late) could be established as

$$\sqrt{I^2 + Q^2} = 620 \tag{5.3}$$

and considering the sum of the three branches, in the worst case (6 dB over early, 3 dB over prompt, 0 over late) the threshold value can be selected to

$$620 + 440 + 311 = 1371$$

Notice that this value refers to a mean condition. Figure 5.9 shows the behavior of the sum of early, prompt and late power magnitudes when no signal is present.

Even if the mean value of the prompt power when no GPS signal is present is near to the one predicted (311, 330 measured), the noise causes the sum of prompt, early and late magnitudes to be higher than the predicted threshold too often, considering that to obtain this data a PRN code known to be absent has been used. The solution selected to correct this issue is to apply a simple shift-and-add filter with a cut-off frequency of 124 Hz (shifting by 2). Figure 5.10 shows the result of such filtering over the power magnitudes values. In this case, the power values are much closer to mean expected values, and early + prompt + late power

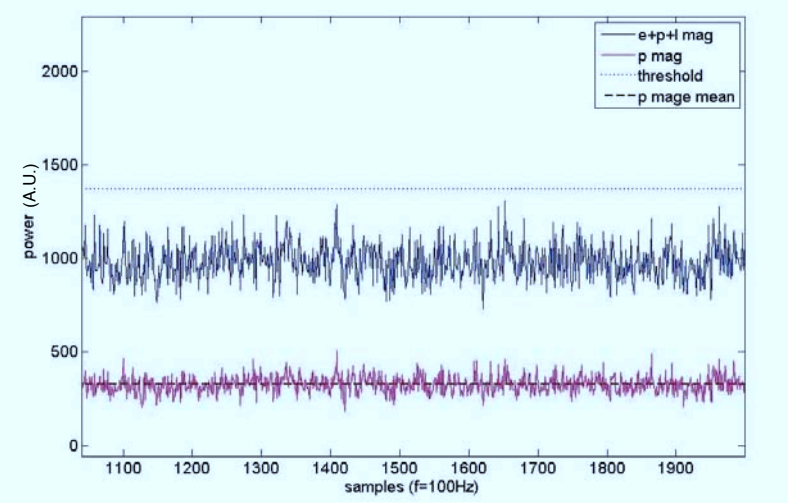


Figure 5.10: Early + Prompt + Late power (blue), prompt power (purple), prompt power mean (brown) and acquisition threshold (black), filtered data no GPS signal in-view; sample frequency 100Hz [48]

signal remains all the time under the selected threshold.

Finally, Figure 5.11 presents an example of the resulting values in case of a successful acquisition, pull-in and lock.

5.5.3.1 Adjusting threshold values

Previous section shows how the different threshold values are obtained for the tracking routine. However, it becomes clear that there is a trade-off relationship when selecting those threshold values, between the Namuru sensibility and the false alarm error rate. The false alarm error occurs when there is no GPS signal, but the noise causes the early+prompt+late power magnitude exceeds the selected threshold. In order to increase the receivers sensibility to be able to acquire and track the reflected GPS signal, these threshold values have been set to new lower values, as showed in table 5.5

As it is showed in table 5.5, only the acquisition threshold value has been decreased. The other threshold values remains the same. The main reason for only changing the acquisition threshold relays on the fact that using the default acquisition threshold value the Namuru-GPL was unable to acquire the reflected signal coming from front-end 2. But, once the signal was successfully acquire, it had no problems during its tracking.

The new acquisition threshold value was set considering the case of 3 dB of signal power over

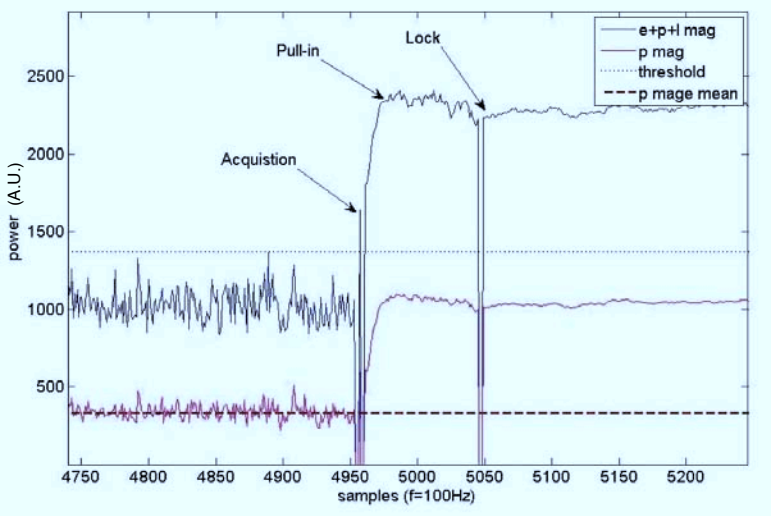


Figure 5.11: Early + Prompt + Late power (blue), prompt power (purple), prompt power mean (brown) and acquisition threshold (black); sample frequency 100Hz [48]

Table 5.5: Namuru-GPL default and new threshold values for tracking routine

Threshold name	Default value	New value
Acquisition (ACQ_THRESHOLD)	1370	1100
Confirmation (CONF_THRESHOLD)	590	590
Pull-in (PULL_THRESHOLD)	580	580
Lock (LOCK_THRESHOLD)	660	660

the considered noise level (311) in early branch and 0 dB over noise level for prompt and late branches. This is:

$$I^2 + Q^2 = 97142 \overset{+3\ \text{dB}}{\rightarrow} 193824 \rightarrow \sqrt{193824} = 440 \tag{5.4}$$

$$\text{Early (3 dB)} + \text{Prompt (0 dB)} + \text{Late (0 dB)} = 440 + 311 + 311 = 1062 \overset{\text{rounded to}}{\rightarrow} \sim 1100 \tag{5.5}$$

The value was rounded up because the measured noise power value was typically higher than 311, between 320 and 330.

After performing some simple tests by decreasing the other tracking routine threshold values, the receiver was repeatedly stuck in false locked states, resulting in incorrect pseudorange and position measurements. Thus those other threshold values, confirmation, pull-in and lock thresholds, were left unchanged.

Notice that increasing the false alarm error rate can cause an increment of the receiver's acquisition time. However, lock signal time is not a constrain in this application if it stays under a reasonable time of minutes. The thresholds are set according to table 5.5 values in Namuru-GPL config file constants.h.

5.6 Chapter summary and conclusions

This chapter has introduced the Namuru-GPL software receiver. As its name suggests, the Namuru-GPL receiver solution has been developed to run over the Namuru V2 GNSS development board described in the previous chapter. The Namuru-GPL solution has been divided into three units: its front-end, implemented using a Zarlink GP2015 chip, and the baseband processor and the Nios II processor, both implemented within the Altera FPGA. This chapter has focused on the baseband processor and the receiver's software that runs on the Nios II.

The baseband processor, developed using VHDL, has been presented as well as its main features. Its design follows the conventional GPS receiver signal processor solution: 6 different tracking arms (early, prompt and late for in-phase and quadrature signals) for each one of the 12 independent channels. Most of the main GPS receiver functions, such as the pseudorange measurement, the navigation data decoding and finally computing the position solution are performed by software. These tasks are executed by the Nios II processor and are synchronized and managed using the available software interruptions and their corresponding

interruption routines. Those tasks and their relationships have been explained during this chapter development.

Then, this project's first main contribution has been introduced. This is the Namuru-GPL modification to allow the receiver to use both on-board front-ends to acquire and track the received signals simultaneously, direct by the first front-end and reflected by the second one. This modification implied to modify the baseband processor VHDL design, as well as introduce changes in the Namuru-GPL software to manage this new design changes.

Finally, in order to increase the receiver's sensitivity to be able to acquire and track weak signals, the thresholds used during the tracking routine have been reviewed and set to lower values.

Chapter 6

Parallel acquisition module

This chapter mainly describes the proposed parallel acquisition module. This module represents this document's second main contribution (Namuru-GPL's adaptation in order to use both available front-ends is the first).

The following chapter is probably the most descriptive part of this document. Its development follows the design process progression. First, it starts explaining the reasons that have motivated the development of this module. Then it follows describing the proposed design and the main decisions taken. At this point, the low-pass filtering and downsampling approach to reduce the size of the required FFT resources is described, as well as the MATLAB tool implemented to simulate the acquisition module's behavior.

The design description is complemented by a complete description of the acquisition module implementation process. Part of this implementation description (the characterization of the different blocks VHDL blocks) has been included in the Appendix A.

This chapter's final sections describe the hardware verification process using VHDL simulation tools, provide a module's performance overview and finally summarize FPGA resources needed by the selected implementation.

6.1 Parallel acquisition module justification

Previous chapter has depicted the Namuru-GPL receiver. This receiver uses a serial search acquisition method in order to detect the presence of GPS signals.

As it was exposed in chapter 2, a frequency domain search is often preferred for a more efficient acquisition. Namuru-GPL testing showed how this acquisition approach can take many minutes to successfully achieve its first position fix. The Namuru-GPL time-to-first-fix (TTFF) is high even when it uses 12 channels. Considering the receiver modifications introduced in the previous chapter, which pursued to adapt the receiver for possible reflectometry applications, the number of channels available for each front-end has been reduced to 6. Thus, if only 6 channels are used, the required time to lock 4 valid satellites will increase, in average by a factor 2. In many GPS applications, like those involving very high receiver speed and altitudes (such as high dynamic regime), the time in which a satellite is in sight can be limited to tens of minutes. This fact clearly limits the Namuru-GPL capabilities for that type of applications.

Chapter 3 introduced the parallel code acquisition method as an alternative to the serial search acquisition. This method has shown to be hundreds of times faster than classical serial search acquisition [53]. Modern GPS receivers typically use parallel acquisition methods, where a frequency domain search is performed, which it is considered to be more computationally efficient. This is because the time domain search represents a serial approach while frequency domain search employs a parallel search strategy. However, frequency domain search is more resource intensive.

Thus, in order to dramatically reduce the required acquisition time and also increase the receiver's sensibility and flexibility, a new parallel acquisition module has been designed and implemented within the FPGA using VHDL. The following sections describe this new proposed parallel acquisition module.

6.2 Proposed Design

The parallel code acquisition has been described in chapter 3. This method requires to transform signals into frequency domain. This is done using an Fast Fourier Transform (FFT) algorithm.

Resources required in a frequency domain search are mainly determined by the FFT algorithm. To exploit the circular convolution, an FFT algorithm must process at least one code period of the input signal. In consequence, for a half-chip spacing correlator, at least $2f_cT$ samples must be processed by the FFT algorithm, where f_c and T are the code chipping rate and code period (in seconds), respectively.

Considering the Zarlink GP2015 front-end sampling frequency, $f_s = 5.714$ MHz, the number of samples for code period ($\approx$1 ms) is 5714. FFT length is constrained to $N = 2^n$ number

of samples. Thus, a 8912 points FFT algorithm is needed. The 5714 input frame length can be zero-padded up to 8192.

Using low hardware resources becomes a main design limitation factor. The acquisition module implementation is desired to fit in the Namuru board's Altera Cyclone II FPGA. This FPGA features have been described in chapter 4. It has only 50,528 logic elements and 581 kbits of on-chip memory. The available 8192 FFT algorithm implementation fits in the Cyclone II FPGA. However, it does not fit together with the Namuru-GPL logic. Additionally, using 8192 samples per each code period in frequency domain notably increments the size of the total memory required by the acquisition module. This required size is over the FPGA available memory resources.

If reducing the number of IF samples to be processed was possible, then, it will be possible to significantly reduce the computationally load of frequency domain acquisition. Thus, reducing the needed hardware resources.

In order to reduce the number of IF samples, two options were studied:

1. Reducing the sampling frequency from 5.714 MHz to a lower value that will need an smaller FFT. However, implementing this option showed to be complicated because of the limited possible sampling frequency range provided by the Zarlink GP2015 front-end. The smaller FFT feasible size with this option was 4096 samples. This FFT size implementation still required too many memory resources.

2. In [53] a filtering and downsampling technique is proposed to reduce the number of FFT samples in frequency domain acquisition. This method can reduce enough the number of IF samples that allows to use an FFT implementation that fits in the Cyclone II FPGA. Thus this method was selected for implementing the new parallel acquisition module, and it is described next.

6.2.1 Design using 1024-points FFT

Thanks to the filtering and downsampling approach based on the work described in [53], a design for the parallel acquisition module, using only 1024-points FFT, has been developed. Figure 6.1 shows the proposed acquisition module data flow.

The incoming IF signal (from RF front-end) is multiplied by a carrier replica generated by a number controller oscillator (NCO). After removing the carrier removal, both signal version (in-phase and quadrature) are filtered through a low-pass anti-aliasing filter with a passband frequency of 512 kHz. After filtering, the signals are down sampled from 5.714 MHz to 1.024

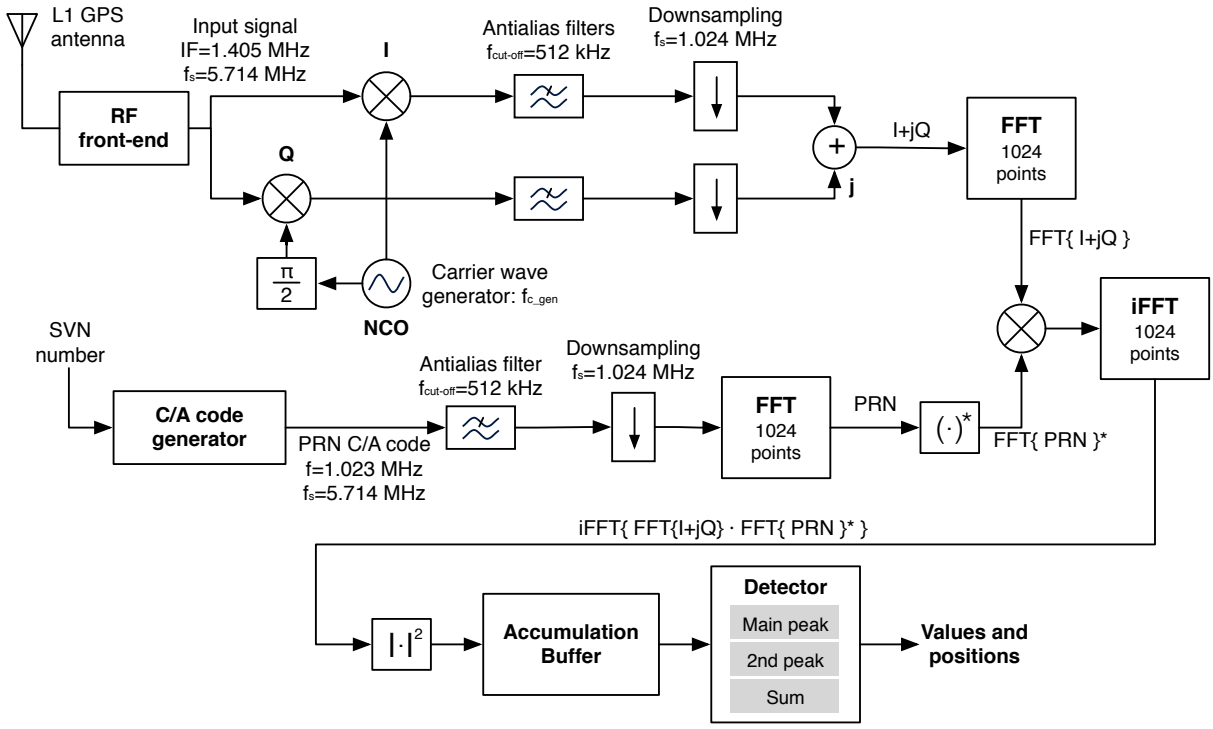


Figure 6.1: Proposed acquisition module general data flow

MHz, such that the resulting number of samples is now 1024, equal to the FFT length. In-phase signal is set as the real FFT input, while quadrature is set as the imaginary input part.

In parallel, the C/A code generator has output the local generated code replica. This C/A code generator can generate the 32 different C/A sequences depending on its SVN[1] input. The C/A code local replica is generated with a 5.714 MHz sampling frequency. Like the incoming received signal, the C/A code is low-pass filtered and down sampled to 1.024 MHz sampling frequency. Then, the resulting 1024 C/A code samples are passed to the frequency domain by the FFT function.

The complex product between the resulting C/A code replica conjugate and the received signal is then computed. The inverse FFT (iFFT) is performed over the resulting product. The squared modulus of iFFT output samples is then computed. At this point the acquisition module has computed delay doppler map (DDM) for 1 ms of coherent integration time and the corresponding SVN.

The resulting 1024-sample frames for a SVN - carrier frequency pair can be incoherently integrated by using an accumulation buffer, that simply adds an stores the incoming frames. The result of the incoherent integration is again another DDM, but know corresponding to an incoherent integration period of N ms, being N the number of frames accumulated by the previous buffer. This DDM is passed as input to the detector module.

The detector module outputs:

1. DDM maximum value and its position
2. DDM second highest value and position. To be sure that this value does not belong to the highest DDM peak, a one-chip separation is needed. This means that this second highest value is only considered as such when its distances is over one chip from the maximum DDM value.
3. Sum of the all DDM position values.

These 5 values can be used to compute evaluation parameters, such as Peak-to-second-peak ratio, and peak to average ratio. These parameters then will be compared to a certain stablished threshold in order to determine if a GPS signal is present or not. Since computing ratio implies divide operations, their computation by dedicated hardware will require many FPGA resources and it is better if it can be performed by software (for example, by the software running on the Namuru-GPL's Nios II soft-processor).

[1]The Space Vehicle Number, uniquely identifies each GPS satellite

6.2.1.1 The anti-aliasing filter

The selected anti-aliasing filter is a simple low pass filter that is used to remove the tail of main spectral lobe. The narrow band lobe is then used for further processing. This filtering causes the signal to lose some of its power. The signal power in a single-sided bandwidth of $'B'$ is expressed as:

$$P = \frac{1}{\pi} \int_0^{2\pi B} S(\omega)\, d\omega \ , \tag{6.1}$$

where

$$S(\omega) = \frac{AT_c^2}{2} \operatorname{sinc}\left(\frac{\omega T_c}{2}\right), \tag{6.2}$$

is the L1 C/A signal Power Spectral Density. A is the signal amplitude and T_c is the C/A code chip period.

Expression 6.1 is a $\text{sinc}^2()$ function, and as such, its main lobe concentrates most of the function area (in this case signal power). If 6.1 function is evaluated, from $B = f_c/4$ to $B = f_c$ it results in a low power loss near the edges of the main spectral lobe. Thus, low-pass filtering can also remove some of the noise from the signal. This feature is exploited to reduce the computational load of frequency domain acquisition.

The bandwidth B of the anti aliasing filter will be selected as 512 kHz in order to resample the signal to a new f_s of 1.024 MHz. Figure 6.2 illustrates the comparison between the filtered and unfiltered C/A code (PRN-31) in the frequency domain. The relationship between the filter bandwidth and the resampling factor is presented next along with the down-sampling process description.

6.2.1.2 The down-sampling process

Following the approach presented in [53], the down-sampled frequency obtained is given as:

$$f_{rs} = \frac{f_s}{\Psi}, \tag{6.3}$$

where Ψ is the downsampling factor and f_s is the original sampling frequency. The low-pass (or anti-aliasing) filter passband B is related with the desired sampling frequency (f_{rs}) as:

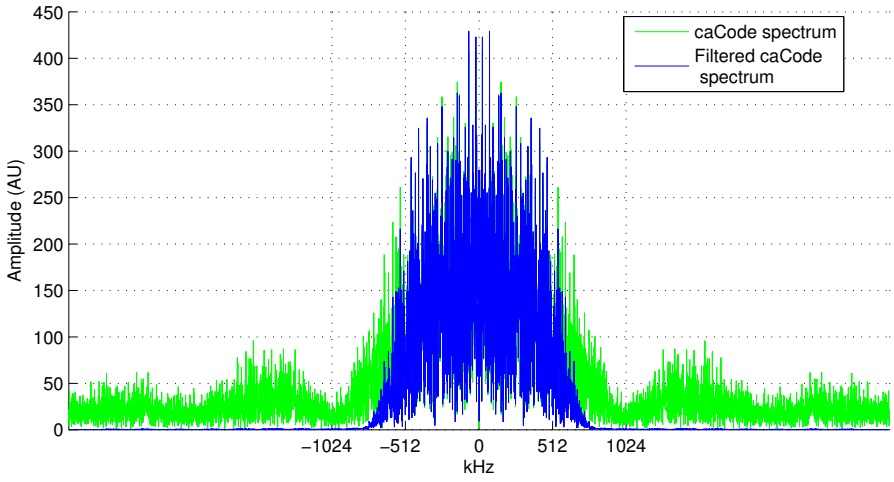


Figure 6.2: C/A code (PRN-31) spectrum after filtering it with the proposed anti-aliasing low-pass filter with a $f_{cut-off} = 0.512$ MHz. The C/A code sampling frequency is $f_s = 5.714$ MHz

$$B = \frac{f_{rs}}{2} = \frac{f_s}{2\Psi} \tag{6.4}$$

The down sampling process can be performed by following the next algorithm. An index vector is generated as:

$$x\,[n] = \left\lfloor \frac{f_s}{f_{rs}}(n-1) \right\rfloor + 1\,, \tag{6.5}$$

where n is an integer, given by $n = 1, 2, \ldots, f_{rs}T$. $\lfloor\rfloor$ represents the integer part. Input IF samples on the indices given by x, are then selected. Thus, reducing the number of samples per code period so that they fit in a 1024-point FFT block.

However, implementing this downsampling algorithm in the FPGA requires to save the index vector in the already limited on-chip memory. To work around the memory limitation issue, a simple phase accumulator can be used to generate a pulse train signal with a frequency equals to f_{rs} and select the filter output samples when this signal is asserted. This method drawback is that it needs f_sT samples, that is 5714 clock cycles for a $f_s = 5.714$ MHz and a $T = 1$ ms, to be obtain the complete the down-sampling process.

The combined filtering and downsampling method can be considered as a resampling to $f_{rs} = \text{FFT-size} \cdot 1000$ Hz approach. Figure 6.3 shows the autocorrelation results for original C/A code (PRN 31) with a $f_s = 5.714$ MHz and the C/A code resampled to $f_{rs} = 1.024$ MHz

using the filtering and downsampling method. This example shows how the autocorrelation results present a similar behavior with normalized correlation values under 0.1 when the code is not aligned.

6.2.2 Design verification methodology

Complex logic, such as the required by the proposed parallel acquisition module, requires a considerable implementation time in VHDL. During the first design stages, the main objective should be to assure that the design concept results are the desired before spending the time required to port it to VHDL.

In order to reduce the design verification time, a model of the proposed acquisition module has been implemented using MATLAB. Using this model, it is possible to introduce and test new changes in the design. For example, it is possible to change the low-pass anti-aliasing filter properties, the downsampling factor or even test other possible FFT sizes. Additionally using a MATLAB model presents the next advantages:

1. MATLAB has already many implemented functions, such as the FFT, speeding up the design verification.

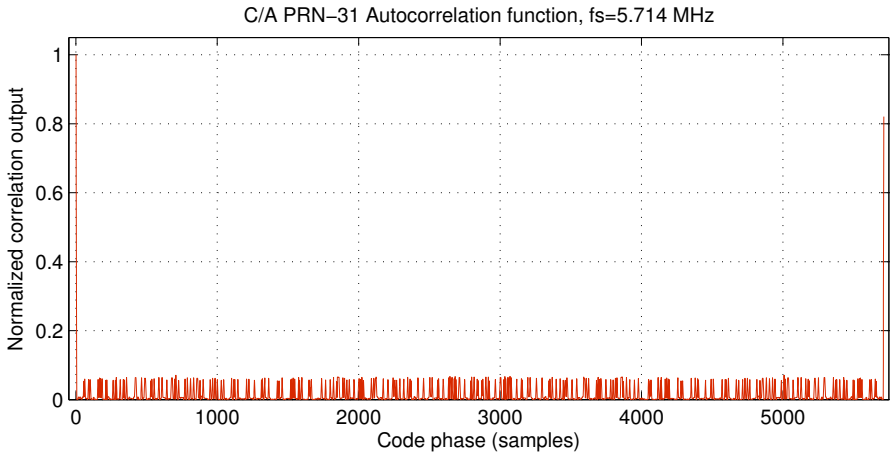
2. With a MATLAB model there's no need to care about advance digital design issues such as control signals and synchronization issues.

3. The model simulation time is greatly reduced to few minutes.

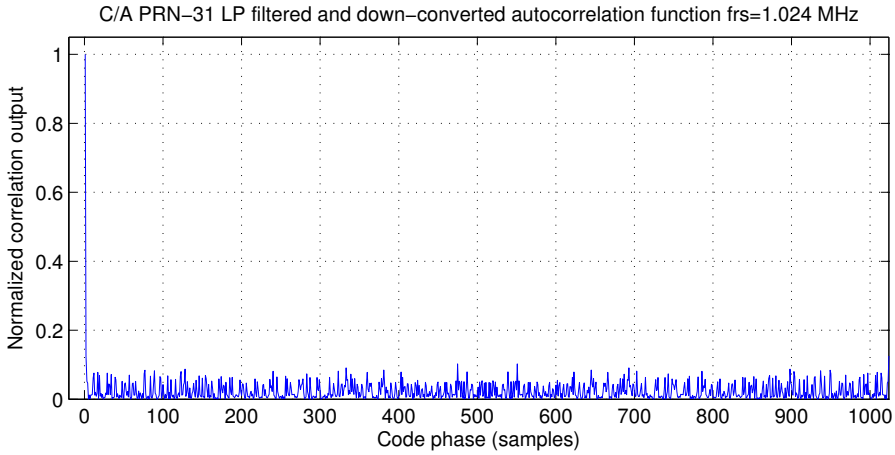
6.2.2.1 MATLAB parallel acquisition module model

The proposed MATLAB model for simulating the proposed parallel acquisition design has been implemented within the Acq_simulator function. This function executes three different acquisition models:

1. Reference model: This acquisition model does not use the proposed filtering and downsampling method. By default, it uses a $f_s = 5714$ MHz, using a 5,714 samples per ms of coherent integration time (set to 1 ms by default). The signals are transformed to frequency domain by a DFT of exactly 5,714 points. This model was implemented in order to have some reference model. This model allows to compare the effects of the filter and down sample design changes with an standard parallel acquisition approach.



(a)



(b)

Figure 6.3: (a) Represents a C/A code T period normalized autocorrelation function result for SVN-31 sampled at $f_s = 5.714$ MHz. The only correlation peak is spitted because with this f_s , a chip is 5.58 samples wide. In (b), the C/A code has been low-pass filtered and down-sampled to $f_{rs} = 1.024$ MHz before computing the its normalized autocorrelation.

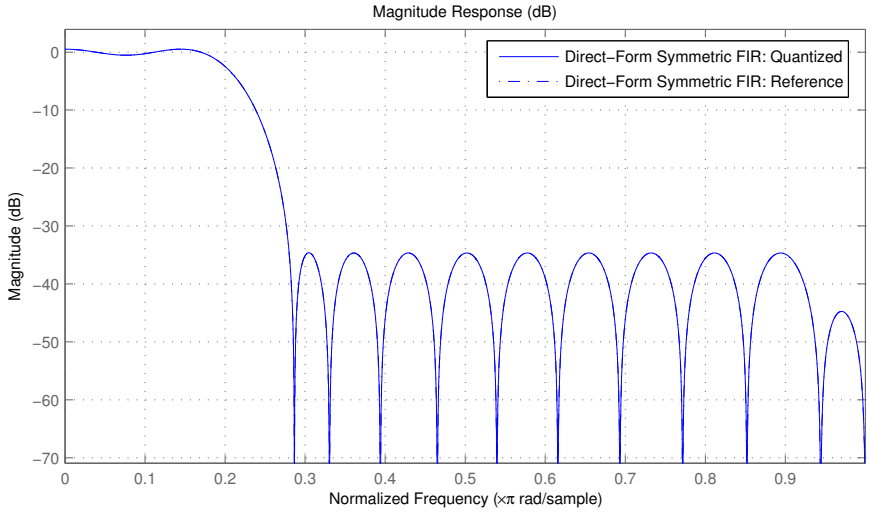


Figure 6.4: Selected Symmetric Low-pass FIR magnitude response

2. Hardware model: This model is able to simulate the acquisition module implementation behavior. It implements the proposed parallel acquisition model described in previous section. This implies low-pass filtering and down-sampling the signal to $f_{rs} = 1.024$ MHz, which results in 1,024 for a C/A code period. Table 6.1 shows the selected low-pass filter characteristics, while Figure 6.4 shows the filter's frequency response. This filter was designed using the MATLAB's Filter Design and Analysis Tool[2] (fdatool). More details about its VHDL implementation are provided later during this chapter.

   In order to reproduce the exact VHDL functional behavior, Altera provides MATLAB functions that exactly simulates the behavior of their hardware IP cores. Some of those IP cores (Altera FFT MegaCore function and Altera NCO) are used in the acquisition module implementation and are described in detail in the following sections of this chapter.

   In addition, this simulation model takes utilizes the implemented data representation (typically signed 10-bit precision per sample) of each data path. This feature allows to simulate also the data quantization losses in the system. Thus, this model allows to accurately simulate the acquisition module implementation within a matlab environment with much shorter simulation times.

[2]The Filter Design and Analysis Tool is a powerful graphical user interface (GUI) in the Signal Processing Toolbox for designing and analyzing filters. It allows to quickly design digital FIR or IIR filters by setting filter performance specifications.

Table 6.1: Selected low-pass anti-aliasing filter details

Parameter	**Value**
Filter structure	Direct-Form Symmetric FIR
Filter order	26
I/O format	fixed-point 10-bit samples
Sampling frequency	5.714 MHz
Passband frequency	0.512 MHz
Stopband frequency	0.8 MHz
Passband ripple	1 dB
Stopband attenuation	35 dB

The Acq_simulator function returns and represents, for all models, reference and hardware, and each SVN:

1. The computed DDM for each SVN.
2. The computed maximum peak-to-average ratio, which is considered as SNR approximation.
3. The computed peak-to-second peak ratio.

The Acq_simulator function allows to select different coherent integration times (only for reference model, in hardware model is fixed to 1 ms) and different non-coherent integration times (multiples of coherent integration period). The Acq_simulator function input parameters and outputs are detailed in table 6.2.

Figure 6.5 shows the results of Acq_simulator execution. The input data file used for this simulation was the "GPS_1.bin" file. This file was provided with the Namuru-GPL software as an additional file for development purposes. It was known in advance that SVNs 1, 11, 20, 31 and 32 signals are present with different power levels.

Figure 6.5 only represents the DDMs of SVNs 20 and 32, but similar DDM profiles were obtained for SVNs 1, 11 and 31. In all cases the correlation peak computed by the Acq_simulator hardware model was clear enough over the noise floor for confirming the presence of the signal, similar to the represented DDMs. On the other hand, no peaks with similar max-to-average ratio were found for the other SVNs.

Based on that results, it is possible to affirm that the Acq_simulator is able to detect the signal GPS C/A signal presence in the L1 band. The Acq_simulator represents a useful tool

```
function [resultsmA resultsmHardware]= Acq_Simulator(fileName, mode,
                    non_coherent_n,ms_coherent)
```

Table 6.2: Acq_simulator function definition and details of its inputs and outputs

Parameter name	**Direction**	**Description**
fileName	In	fileName string specifies the binary data file that Acq_simulator function will use as input for the 3 simulation models. This file must contain the received data stream from the receiver's front-end encoded as $\{-3, -1. + 1, +3\}$
mode	In	mode parameter defines which simulation modules will be executed: 1, only Reference model is executed 2, Reference and Hardware models are executed
non_coherent_n	In	Defines the number of ms_coherent data blocks that will be used for non-coherent integration. The max value for these parameter is limited by the input signal file total size.
ms_coherent	In	Defines the coherent integration time in ms (only integer values are accepted). This parameter only applies for Reference model; Hardware model has fixed coherent integration times of 1 ms. Notice that for a fixed sampling frequency, this parameter directly affects on the FFT function size.
resultsmA/B/Hardware	Out	For each simulation model returns a data structure that contains: 1. DDM(Frequency Bin, code phase , PRN) 2. SNRMetric(PRN): peakMax-to-average ratio 3. peakMetric(PRN): Max-to-second peak ratio

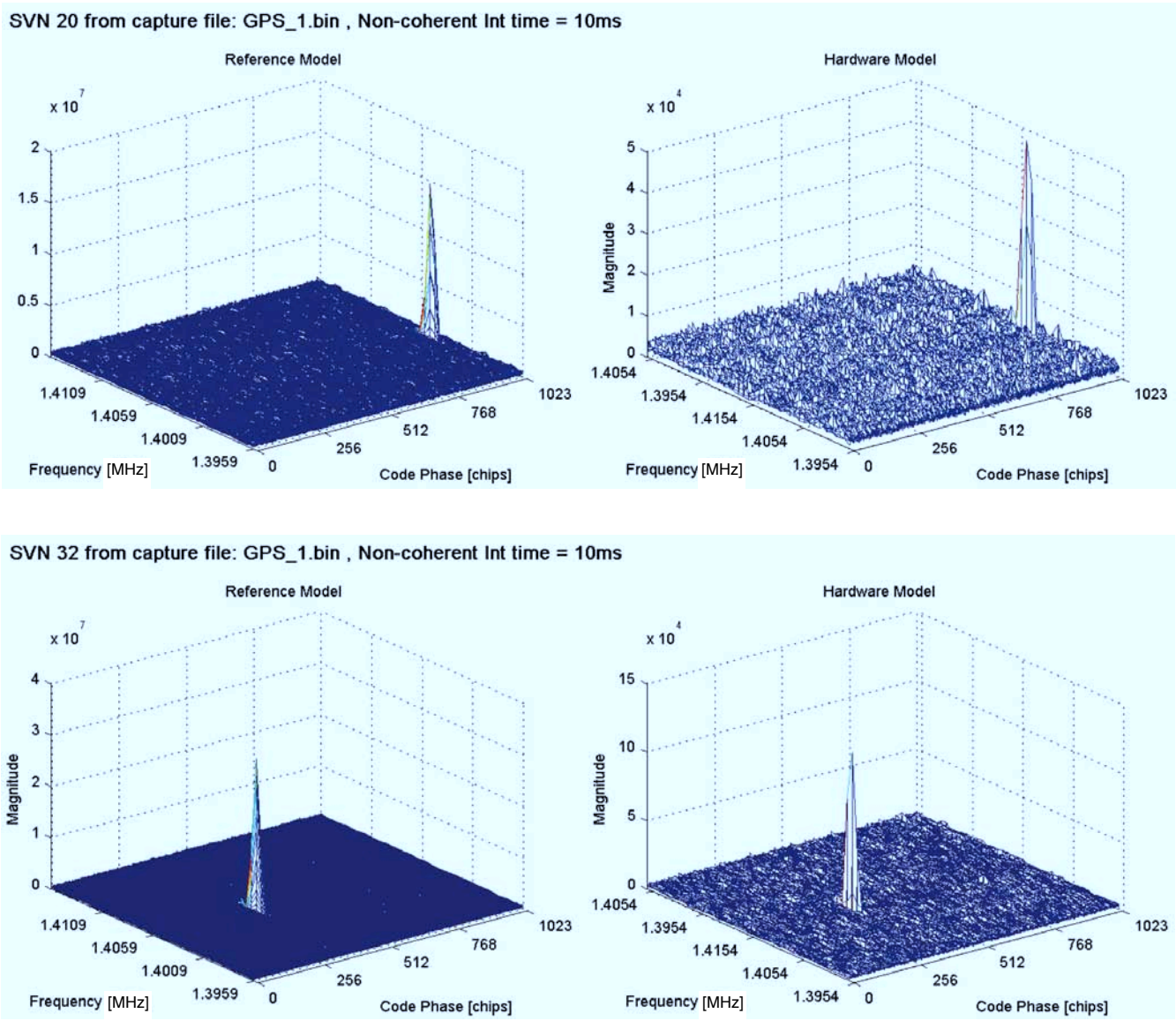


Figure 6.5: Acq_Simulator results for SVNs 20 and 32

Table 6.3: Altera Cyclone II EP2C50F484C6 available resources for acquisition module implementation

Resource	Total	Used by Namuru-GPL	Available
Logic Elements	50,528	21,969	28,559
Memory bits	594,432	87,808	506,624
Embedded Multipliers	172	11	161
PLLs	4	1	3

to validate the proposed acquisition's module hardware behavior, because allows to compare the results obtained with a theoretical reference model. In the same way, the Acq_simulator will be used in following chapters to validate the results obtained in the tests developed to characterize the acquisition module capacities.

6.3 Implementation

This section describes the proposed parallel acquisition module implementation. First the implementation and working requirements are presented. Then, it follows a description of the selected system architecture. Finally, multiple sections are dedicated to describe in detail the most important system blocks.

6.3.1 Implementation requirements

Previous sections only described the operations and data flow performed by the proposed acquisition module. This section will focus on how those operations have been implemented taking into account that:

1. The module needs to fit in the Altera Cyclone II FPGA. The objective is being able to integrate the new proposed parallel acquisition module as the new Namuru-GPL acquisition block. Thus, the free resources available when Namuru-GPL design has been load in the FPGA are the limit. These remaining resources are listed in Table 6.3.

2. The module system clock has been set to 40 MHz. This is the clock frequency used by the Namuru-GPL receiver. Using the same clock frequency will help to avoid synchronization problems between the Namuru-GPL baseband processor and the proposed acquisition block. This clock frequency is very important, because it is directly related

with the total acquisition time. The acquisition process needs to be performed fast enough to provide valid data to the tracking system.

Due to the relative movement between the satellite and the receiver, the code phase changes with time. Thus, the acquisition module should provide valid data about the signal code phase and Doppler shift before code phase changes more than 1 or 2 chips (depending on the tracking system). In worst case, the relative speed between satellite and receiver can reach 7 km/s, that is 7 m/ms.

The C/A code has a chip frequency of 1.023 Mhz, thus, which is a chip duration of 1 ms. With 1 ms the signal (supposing air or empty space) travels $c{\cdot}T_{chip} = 300$ m.

Then, a coarse estimation of the available acquisition time is

$$\frac{2 \text{ chips}{\cdot}300 \text{ m}}{7^{\text{m}}/_{\text{ms}}} \approx 85 \text{ ms} \tag{6.6}$$

Then the total acquisition process time for satellite needs to remain under 85 ms, in order to be able to track its signal.

3. Although the acquisition module is intended to work as the Namuru-GPL new acquisition system, a modular and, as generic as possible, implementation is desired. This will allow the design and implementation reutilization in the future. Additionally, keeping the elements of the system as independent as they can be with each other will facilitate their reuse in other designs and future design changes.

6.3.2 Implementation architecture overview

The acquisition module implementation has been decided in order to be as much independent as it can be of the rest of possible system elements. Thus, each time that module operation is activated, it performs the parallel acquisition process only for the satellite whose PRN number is provided.

The acquisition module search covers a total band of 20 kHz, from $f_{L1}-10$ kHz to $f_{L1}+15$ kHz in steps of 500 Hz, were $f_{L1} = 1575.42$ MHz. This makes a total of **41 possible frequency search steps.**

Once the acquisition process for that PRN is over, the module activates an "acquisition ended" signal. This "acquisition ended" signal could be associated, outside the acquisition module, to a hardware interruption, that will indicate the system that the acquisition results are ready.

When "acquisition ended" signal is asserted, the module outputs the max correlation computed value, together with its corresponding frequency and code phase (that is, its position in the DDM). Additionally, the module outputs the second max correlation value, and the sum of all the computed correlation values.

The acquisition module does not determine if a valid GPS signal is or not present. It only outputs the previously mentioned parameters that will allow the system within the parallel acquisition will be integrated to determine if there is or not such signal.

The acquisition module instead, outputs parameters that can be used to compute different indicators such as max-to-average correlation ratio, or max-to-second max ratio. Outputting those values instead of the already computed indicators allows the system in which the acquisition module is embedded to free select its own "acquisition indicators". Thus, providing more design's options and flexibility.

For example, in the Namuru-GPL case, these indicators would be computed by the Namuru-GPL software running over the Nios II soft-processor. After computing those indicators, would be the software who will compare those indicators with pre-fixed thresholds in order to determine the signal presence. Some of those possible "acquisition indicators" will be described in Chapter 7.

Figure 6.6 shows the acquisition module inputs and outputs, while table 6.4 includes their description.

As an example of possible configuration to integrate the proposed acquisition module with the existing Namuru-GPL receiver is shown in Figure 6.7. This Figure shows a configuration for reflectometry applications in which the same acquisition module could be used to acquire both, direct and reflected signal, by multiplexing the signal coming from the two available front-ends. Notice that the Namuru-GPL software needs to be modified to control the acquisition module.

6.3.3 System's architecture operation overview

Taking into account the system requirements and hardware limitations described in the previous section, Figure 6.8 shows the proposed system architecture. For a better understanding, the architecture will be explained describing the acquisition process realized and which task is performed by each acquisition module block.

By default, the acquisition module remains in an idle state until a rising edge is detected in start input. The system then will perform the next steps:

Table 6.4: Acquisition module inputs and outputs description

Signal Name	**Direction**	**Size (bits)**	**Description**
clk	In	1	Clock signal.
rstn	In	1	Active-low asynchronous reset signal
Main_start	In	1	When a rising edge is detected, the acquisition process starts
Sample_signal	In	1	Provides the sampling frequency reference. During the input signal reading, sign and mag input bit values are be saved by the acquisition module when a rising edge is detected. By default, for a correct system operation this signal's frequency must be fixed to 5,714 MHz. Otherwise, changes within the acquisition module architecture are required.
Input_PRN	In	8	PRN number, from 1 to 32. This parameter indicates which satellites signal will be search for.
sign	In	1	sign bit from RF front-end
mag	In	1	mag bit from RF front-end
Acquisition_ended	Out	1	When acquisition process is ended, this signal is set to '1' during 1 clock cycle. It can be used to generate a possible hardware interruption.
DDM_out	Out	20	It outputs all computed correlation results. This is the DDM. Samples are un updated every clock cycle while they are being passed to the acquisition detector module. Unsigned format.
Max_value	Out	32	Maximum computed correlation value (DDM max). Unsigned format.
Max_freqBin	Out	8	Frequency bin in which the Max_value magnitude was found. Unsigned value from 0 to 61.
Max_Position	Out	16	Code phase corresponding to Max_value. Unsigned value from 0 to 1023.
Second_max	Out	32	Second maximum correlation peak value. Only values with a distance greater than 1 chip from main peak are considered. Unsigned format.
Accum_out	Out	32	Sum of all the computed correlation values. Unsigned format.

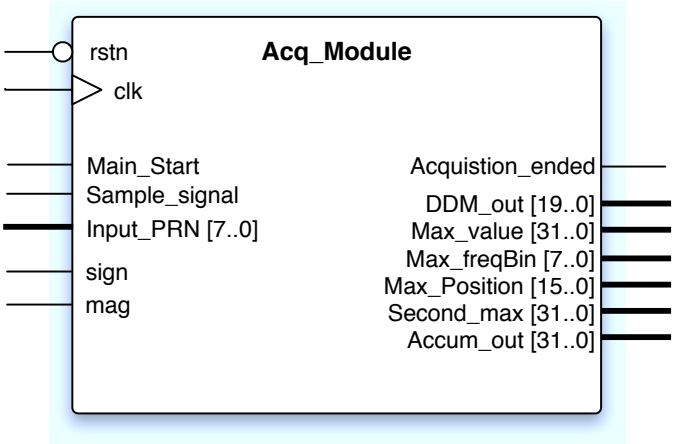


Figure 6.6: Acquisition module inputs and outputs

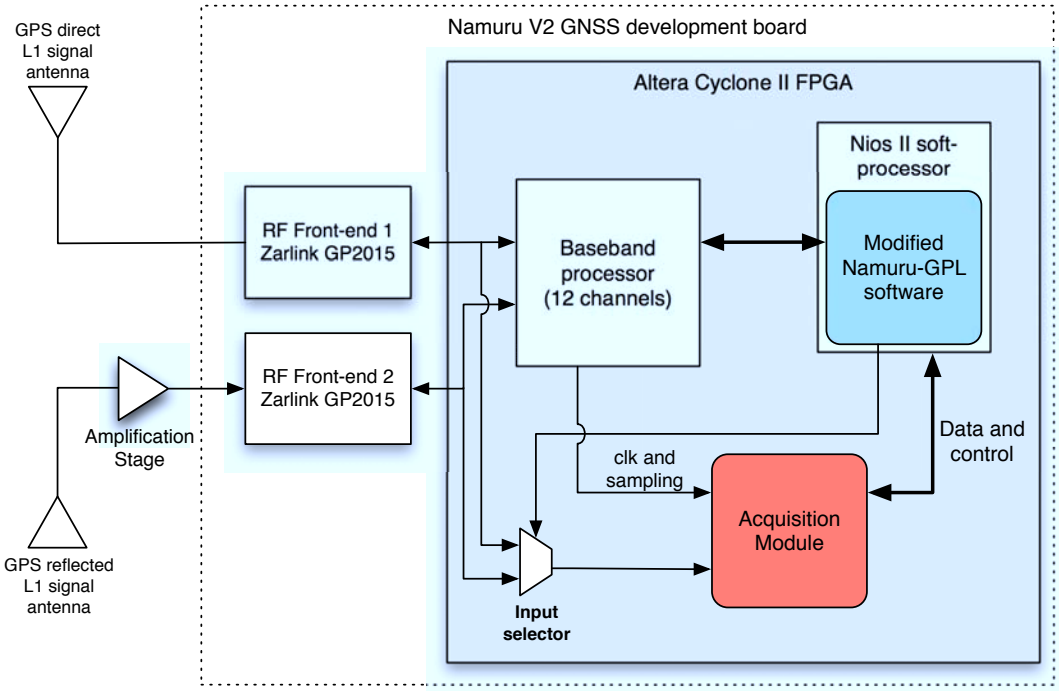


Figure 6.7: Block diagram of a possible acquisition module integration within the Namuru-GPL receiver

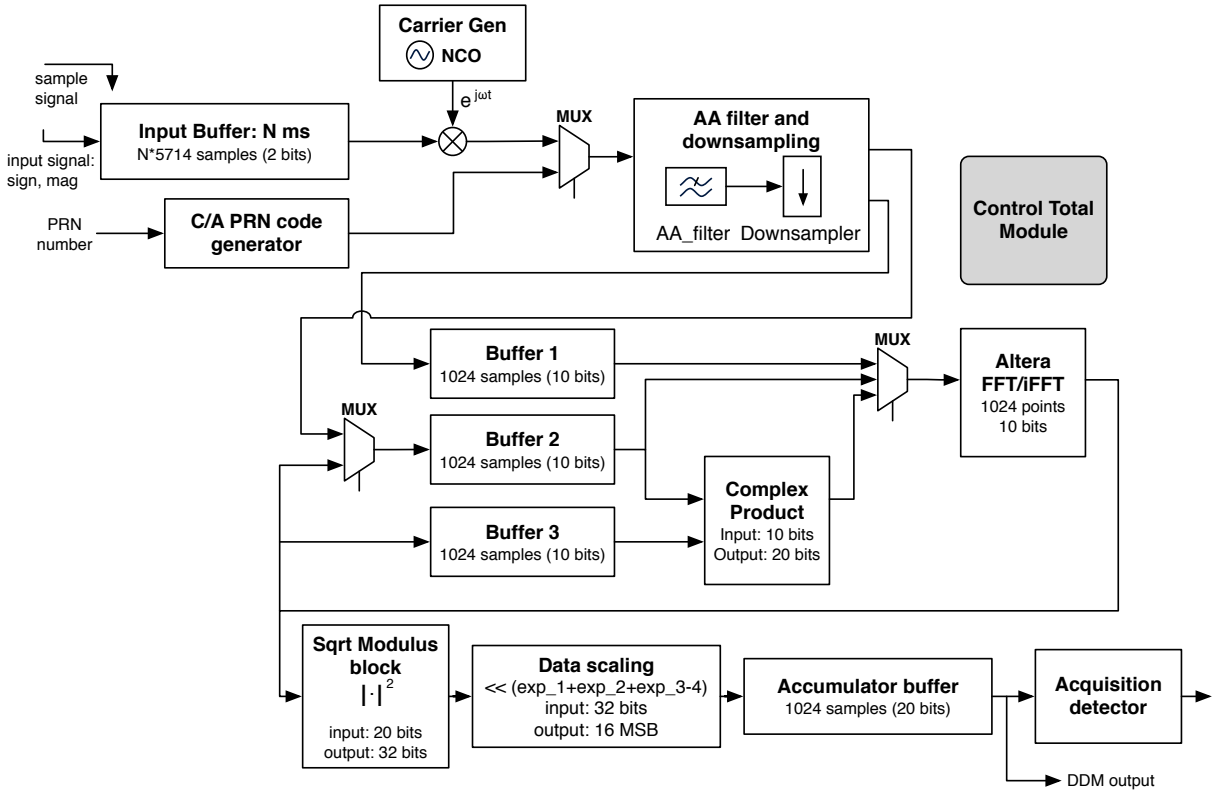


Figure 6.8: Acquisition module design architecture overview

1. The C/A PRN code generator module starts its operation. It uses the PRN number provided as input to generate the corresponding 1023-chip code sampled at f_s = 5.714 MHz. One code period is generated (1 ms), resulting in 5,714 10-bit samples. The sample values are scaled to values $\{-127, 127\}$. This rescaling is necessary to allow a correct their correct filtering in the next step. Rescaling to those values allows a correct filter operation (filter uses fixed-arithmetic with no decimal part) and avoids possible overflows.

2. The C/A code is low-pass filtered and down-sampled by the AA_and_downsampling module. Now the filtered and down-sampled C/A code frame length has been reduced to 1,024 10-bit samples (f_{rs} = 1.024 MHz). This frame size allow its conversion using the implemented 1024-point FFT module.

3. The filtered and down-sampled C/A code is stored in Buffer 2.

4. The filtered and down-sampled C/A code is read from Buffer 2 (set as real part) and transformed into frequency domain by the FFT module. The result is stored again inside the Buffer 2. Additionally Buffer 2 also stores the output FFT exponent (exp_1).

5. In parallel with step 1, the Input Buffer module starts to save the received data stream coming from the front-end. Each time that a rising edge in input "sample signal" is detected, the sign and mag bits values are stored inside the Input Buffer. The Input Buffer size has been defined by a parameter N such as the total buffer size is $N * 5,714$ 2-bit samples. Due to FPGA memory limitations, this N parameter has been fixed to 10, which corresponds to a 10 ms period.

   The selected clock frequency of 40 MHz does not allow to perform the complete acquisition process for 1 ms signal period in less than 1 ms (real-time). However, thanks to this Input Buffer is possible to store 10 ms of contiguous signal. Since this data is stored, it is possible to compute non-coherent integration in 1 ms blocks of those 10 ms without the need to do it in real time. The only time restriction is the total acquisition time limitation described in the previous section.

6. For ($f_{\text{carrier\_gen}}$=$-\ \frac{\text{Search band}}{2}$; $f_{\text{carrier\_gen}}$ + 500 Hz; $f_{\text{carrier\_gen}}$= $\frac{\text{Search band}}{2}$) {

   (a) For (i=1;$i++$; $i \leq 10$) {

      i. Read 5,714 samples from Input Buffer corresponding to the i-th input signal ms and multiply by the sine and cosine generated signals. These signals are generated by the Carrier Generator module. This module implements a numerically controlled oscillator (NCO) that is able to generate different

frequencies depending on an input phase parameter. The sine and cosine signals (in-phase and quadrature) are generated at $f_{\text{carrier\_gen}}$ frequency. Their product with the input signal generates the I and Q signals. I and Q signals are quantized and scaled to $\{-127, -64, -42, -21, 21, 42, 64, 127\}$ values that are 10-bit signed represented.

ii. I and Q signals (of 5,714 samples of size) are passed through the AA_and_downsampling module. The resulting signals I and Q signals have been reduced to 1,024 samples.

iii. I and Q filtered and down-sampled 1024-sample signals are stored in Buffer 1.

iv. The content of Buffer 1 is transformed into the frequency domain by the FFT module. I signal samples are used as FFT real input, while Q samples are used as imaginary input. The obtained complex signal stored in Buffer 3. Additionally Buffer 3 also stores the output FFT exponent (exp_2).

v. The contents from Buffer 2 and Buffer 3 are read and used as input for the Complex Product module. This module outputs the complex product between the complex 1,024-sample complex signals.

vi. The inverse FFT operation is performed over the output of the Complex Product module by the FFT module.

vii. The Squared Modulus block computes the squared modulus operation over the FFT module output. To avoid possible overflow problems, the output are 32-bit samples.

viii. The result is then scaled by the Data_scaling module. This module reads the FFT generated exponents from Buffer 2 (exp_1), Buffer 3 (exp_2) and the last exponent, exp_3, generated during the inverse FFT operation. With these exponents then performs over the 32-bit input samples:

$$\text{Output sample} = \text{Input sample} \cdot 2^{(\text{exp\_1}+\text{exp\_2}+\text{exp\_1})} \tag{6.7}$$

The output samples are truncated to 16 bits (MSB).

ix. If i=1

- Accumulator Buffer saves the 1,024 16-bit samples coming from Data_scaling block.

else

- Accumulator Buffer adds to the previously stored samples the new 1,024 samples (accumulation). In order to avoid overflow, the accumulator buffer is an array of 1024 20-bit elements.

}

(b) Accumulator Buffer content is read by the Acquisition detector module and is set as DDM_out output. This module updates the max value and its position. This is the position within the incoming frame (code phase) and its corresponding $f_{\text{carrier\_gen}}$. Additionally accumulates all the incoming sample values and stores second-max value.

}

7. Acquisition detector outputs the max and second-max values, max value position and the total accumulated value corresponding to the sum of all DDM samples value. At this point a pulse of one clock cycle duration is generated at the Acquisition_ended output.

All the acquisition process is controlled by a control module called Control_Total. This represents an effort to separate the control logic from the operational logic (just as in a microprocessor the ALU logic separated from the control logic). This simplifies the implementation and allows to make changes easily on the existing blocks.

The following section describes this Control_Total block in detail, while a detailed description of the main blocks present in the parallel acquisition implementation can be found in Appendix A.

6.4 Acquisition module control block

All the acquisition module functional blocks operation is controlled by a main control module called "Control_Total". This "Control_Total" module has been implemented following a classical state machine implementation. This approach is very common in the design of control systems.

Figure 6.9 shows the "Control_Total" module Main State machine diagram. Note that this state machine diagram and the ones that are presented next are presented do not follow the strict state machine diagram definition, instead they represent an upper level of abstraction, to avoid unnecessary detail and make the overall system's operation more understandable. More implementation details of "Control_Total" module can be found in the VHDL comments.

The "Control_Total" module implements a main state machine that at same time controls the operation of other 3 state machine. Each one of these 3 other state machines are responsible for different stages of the acquisition process:

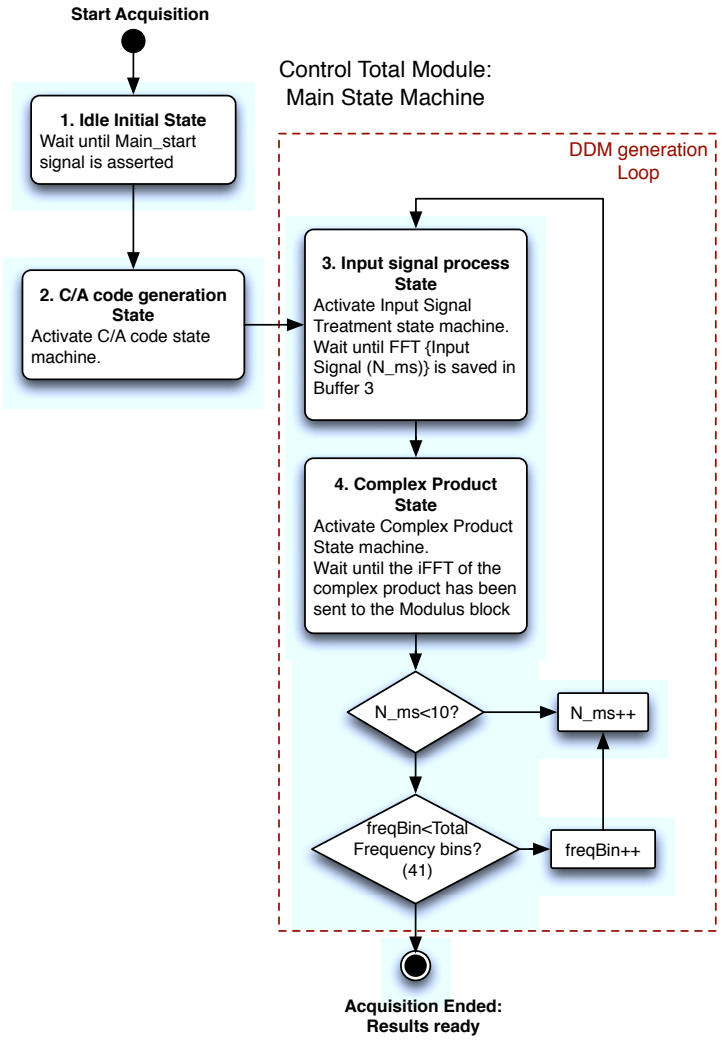


Figure 6.9: "Control_Total" module Main State machine diagram

1. C/A Code generation state machine:

   Figure 6.10 shows the C/A code generation state machine diagram. This stage includes generating the C/A code for the corresponding input PRN-number and its transform into frequency domain.

2. Input Signal Treatment state machine:

   Figure 6.11 shows the Input Signal Treatment state machine diagram. In this stage data stream coming from the front-end is saved into the Input Signal Buffer. This is done only once at the beginning of the acquisition process. This stage reads 1 ms of data from the Input Signal Buffer (5,714 samples) and ends when the carrier mixed, low-pass and down-sampled signal in the frequency domain is saved into Buffer 3.

3. Complex Product state machine:

   Figure 6.12 shows the Complex Product state machine diagram. This stage simply performs the complex product of the input signal and the locally generated C/A code conjugate. It is also the Complex Product state machine who controls the inverse FFT operation over the resulting product. The resulting correlation result is not saved and during the iFFT operation, the output is passed to the Squared Modulus block.

Notice that the last part of the acquisition process, that is from computing the squared modulus of the obtained correlation product until the acquisition is ended, does not need control signaling. The reason is that the modules have been implemented to operate in a cascade configuration, when the end of one module's operation activates the operation of the next module. Thus, the control logic can be reduced in order to increase simplicity and save hardware resources.

This no external control approach is possible only in this acquisition process final part, because the data flow is linear.

6.5 Verification

Implementing a complex design such as the proposed acquisition module requires an intensive verification to ensure the correct behavior of the design. In the case of the proposed acquisition module, this verification has two main purposes:

1. Check that each VHDL block is performing the operations that is suppose to execute, in the correct order, in the expected timing, and perfectly synchronized. Synchronization

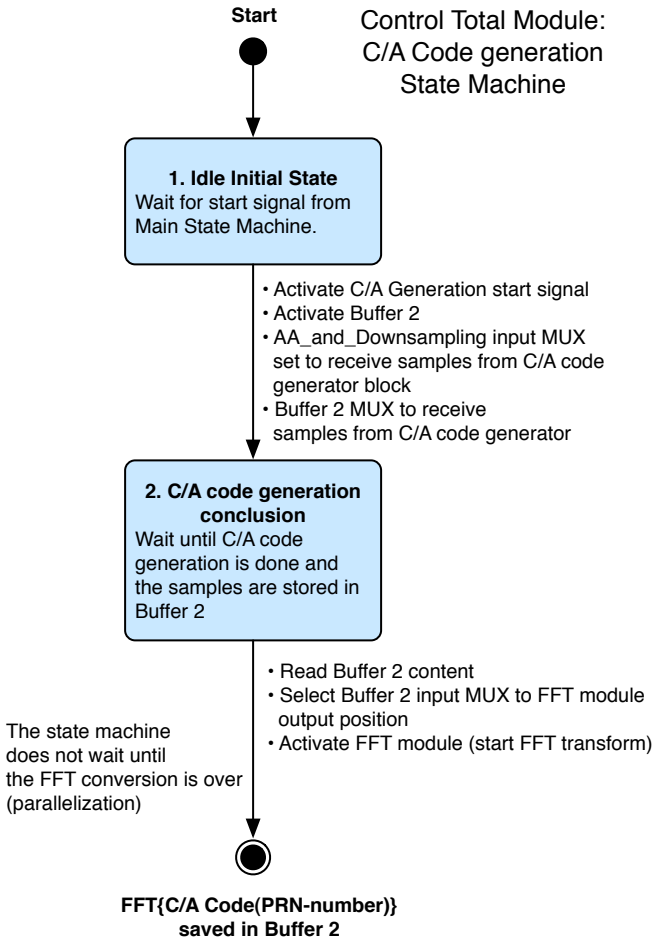


Figure 6.10: "Control_Total" C/A Code generation state machine diagram

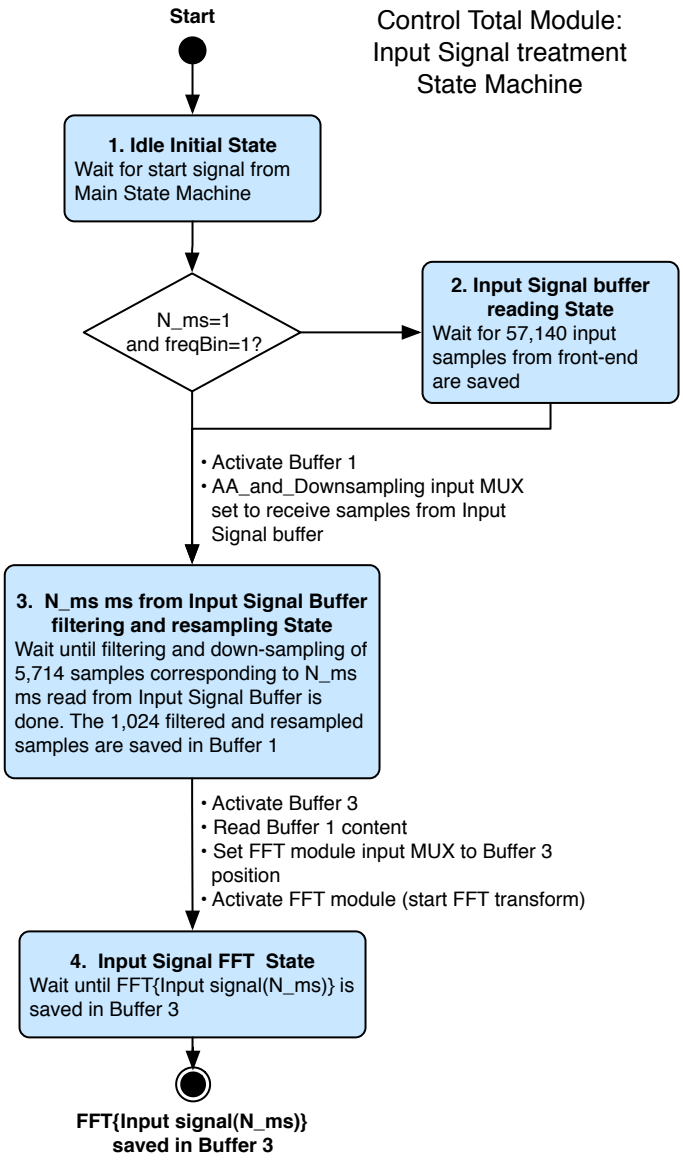


Figure 6.11: "Control_Total" Input Signal Treatment state machine diagram

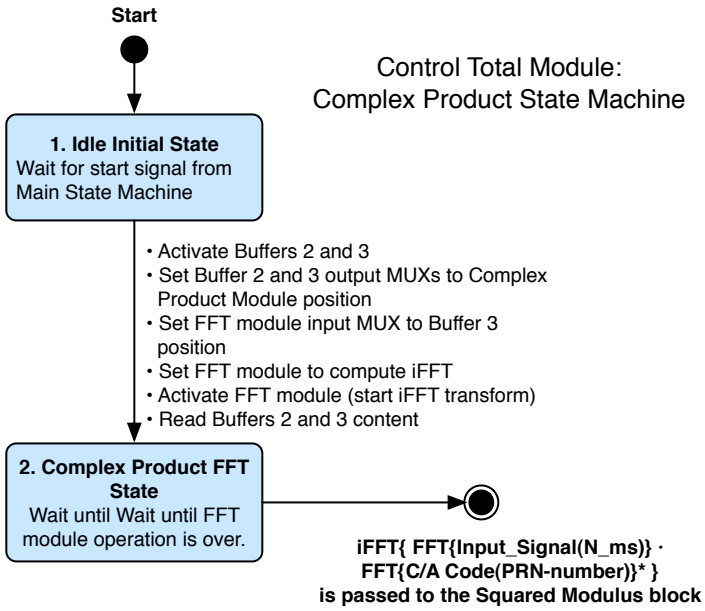


Figure 6.12: "Control_Total" Complex Product state machine diagram

issues are very important within the digital design, and specially in signal processing tasks.

2. Verify that the module outputs are valid and similar to what was expected.

In both cases, the use of hardware simulation tools becomes necessary. While Altera Quartus II includes a simulation tool, this is presents some limitations. While this tool is very precise simulating the hardware behavior at their lower level, the tool has shown to be very slow for behavioral simulation.

In the behavioral simulation, a high level of abstraction to model the design is used. A behavioral design might, for example, contain high-level operations without containing specifics on how the design will be implemented within the FPGA hardware. This allows a fast simulation for complex logic systems such as the proposed acquisition module.

Thus, ModelSim Altera from Mentor Graphics [56] has been used to simulate the behavior of the acquisition module, while gate-level verification and timing simulation have been performed with Quartus II.

Model-Sim features greatly reduce the verification time thanks to an easy-to-use graphical user interface and a really powerful debugging mode. This debugging mode allows to introduce break points in the VHDL code which allows isolating the implementation bugs.

Model-Sim execution produces a waveform representation of the module outputs and internal selected signals during the simulation. This representation allows to verify the correct design's behavior, but it is very limited to check that the obtained results are correct. For a large set of numeric data, visual representation and math tools represent a much more useful alternative to verify the quality of the obtained simulation results. Thus, together with the acquisition module a testbench to verify its execution results was implemented.

6.5.1 Acquisition module testbench

A testbench for the acquisition module has been implemented. Figure 6.13 shows a representation of the testbench structure. The testbench is a VHDL file that includes the acquisition module (Acq_module_v1) as a VHDL component and creates an adequate execution environment. This execution environment includes:

- Clock signal: the testbench generates a clock signal at 40 MHz, same as the produced by the front-end that is used as system clock.
- Sampling frequency signal: a sampling signal is generated at $f_s = 5.714$ MHz by the testbench.
- The input signal: to simulate the acquisition process, input signal data is needed. The testbench reads this data from an previously selected text file and parses it as sign and mag acquisition module inputs. The input data is read at each sampling frequency signal rising edge when the acquisition module is in its input signal reading state.
- Control signals: the testbench deactivates the reset signal and generates a start signal pulse at the beginning of the simulation. The testbench also sets the PRN-number for the simulation.

Additionally, the testbench also can write the internal signal and values into a text file. This is very useful, because once the data is stored in a text file, the content of this text file can be read with MATLAB. Thanks to that, the DDM output can be graphically represented or easily treated. Thus, the testbench saves:

- The DDM output, saved into "DDM_out.txt" file.
- The output of the low-pass and down-sampling filter block. Real and imaginary parts are saved in separate files: "real_aa_down.txt" and "imag_aa_down.txt".
- The output of the FFT block. Again real and imaginary parts are saved in "real_fft_out.txt" and "imag _fft_out.txt" files, respectively.

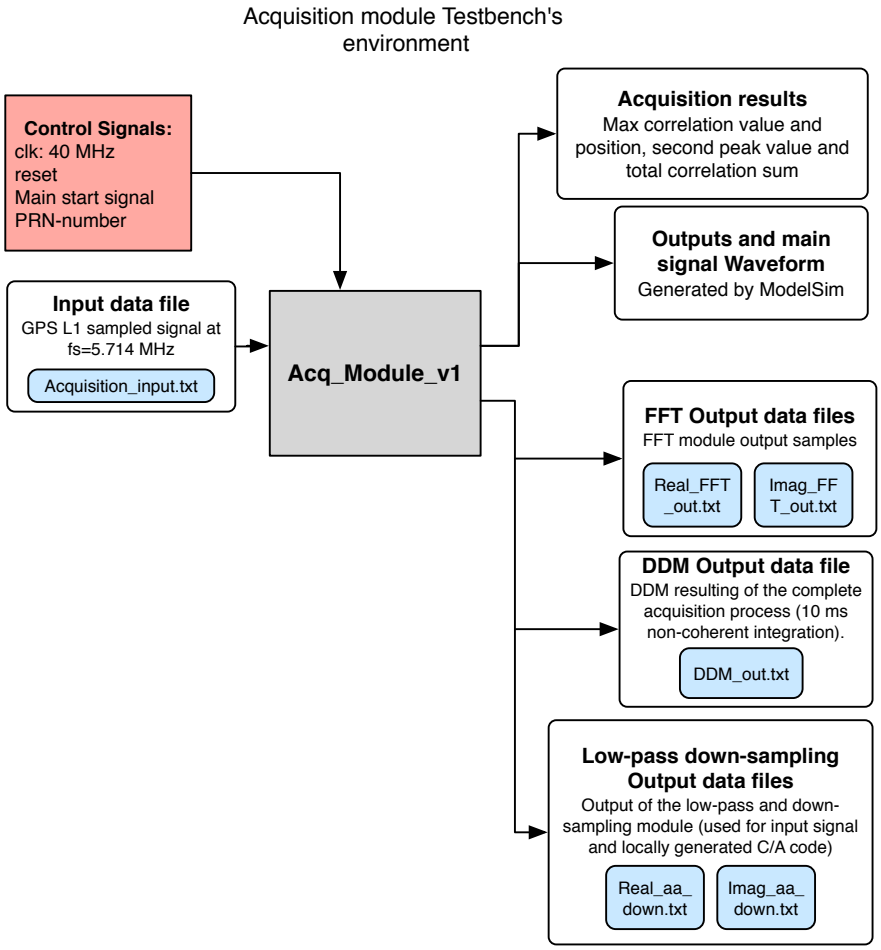


Figure 6.13: Acquisition module testbench description diagram

Table 6.5: Testbench simulation parameters

Parameter	Value	Notes
f_{clk}	40 MHz	-
Sampling frequency:f_s	5.714 MHz	-
Input Signal File	GPS_1.bin	This data corresponds to a GPS signal capture using the Namuru GPS receiver. The signal was sampled at $f_s = 5{,}714$ MHz. The file was provided with the Namuru-GPL software as an additional file for development purposes.
PRN-number (SVN)	32	The satellite 32 was selected because it presents the highest SNR from all the input data file available satellites.

Table 6.6: Example Acquisition Module hardware simulation results for SVN 31 (input signal from capture 'GPS_1.bin' file) with ModelSim compared to the MATLAB's Acq_Simulator Reference model

Output	ModelSim simulation result	Acq_Simulator simulation result (Reference model)
Max peak code phase	865 chips (max found in one of the 5 samples associated to 865 code phase)	865 chips
Max peak frequency	Frequency Bin 22 $\rightarrow f_c = 1.4064$ MHz	Frequency Bin 22 $\rightarrow f_c = 1.4064$ MHz
$\sqrt{\text{Max-to-mean ratio}}$	6.72	9.46

6.5.1.1 Testbench execution results

The proposed `Acq_module_tb` testbench has been executed with ModelSim. The parameters used summarized in table 6.5.

DDM_out data generated during the simulation is represented in Figure 6.14. This data has been obtained importing "DDM_out.txt" data with MATLAB. As it can be seen in the Figure, the module seems to generate a valid DDM. Table 6.6 compares the acquisition result obtained with ModelSim simulation against the result obtained when using the Acq_Simulator Reference model implemented with MATLAB.

To finally validate the acquisition module output, the acquisition module simulator implemented in MATLAB, Acq_Simulator, and described in section 6.2.2.1 has been used. The Acq_Simulator input file and parameters used have been exactly the same as the used in

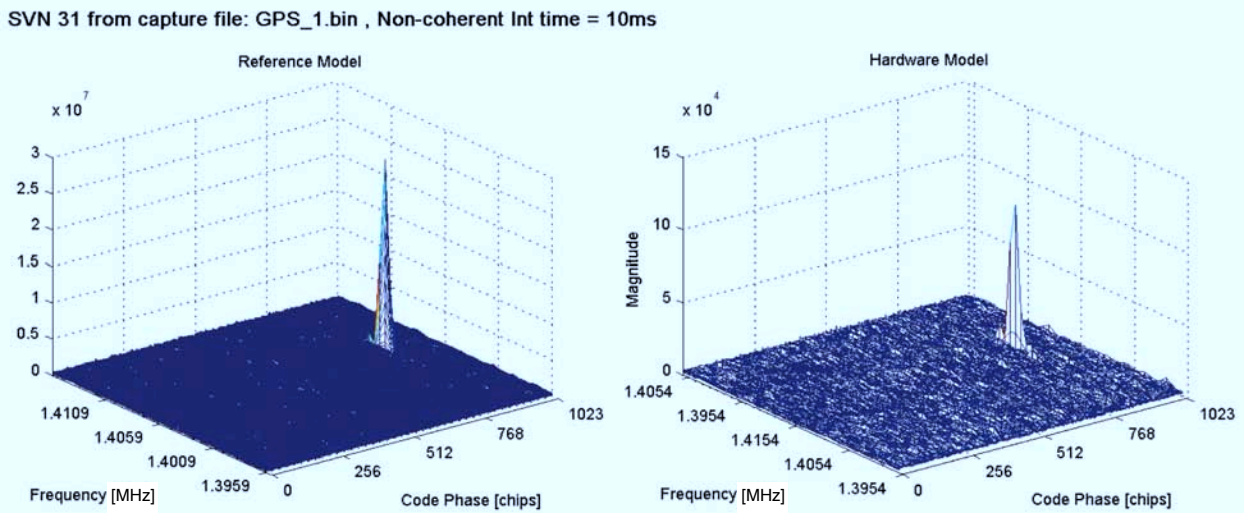


Figure 6.14: Acquisition module simulation result computed with ModelSim (right) compared to Acq_Simulator Reference model result (left) computed with MATLAB

the ModelSim simulation. The result of the Acq_Simulator reference model execution is also represented in Figure 6.14. On the other hand, the Acq_Simulator result obtained for Hardware model exactly was the same[3] as the obtained in the acquisition module simulation using ModelSim, thus they have not been represented again.

Additionally, the same test was reproduced for SVNs 1, 11, 20 and 32, with same positive results.

This result allows to conclude that the proposed acquisition module is able to successfully at detect and acquire an existent GPS signal. Additionally, the Acq_Simulator has proved to be able to exactly reproduce the acquisition module hardware results.

6.6 Performance and resources used

Once the acquisition module behavior has been verified successfully, the last step is check if it can complete the acquisition process within the required time and resources. The maximum time allowed has been explained in section 6.3.1, where the system implementation requirements are described, and has been considered as ~ 85 ms. This time period does not include the time in which the acquisition module is obtaining the signal from the front-end,

[3]The difference between both results was computed and the result was exactly 0.

so called input signal reading. The input signal reading is performed in parallel with the local C/A code generation and it transform to frequency domain, being the code generation considerably shorter than the input signal reading process.

The total processing time for acquisition does not depends on the input signal data value. Thus, for a fixed acquisition parameters of:

- 1 ms of coherent integration time (5,714 for 1 ms period, $f_s = 5.714$ MHz)
- 10 ms of non-coherent integration time (10 periods of 1 ms)
- A total search band of 20 kHz divided in 41 frequency steps of 500 Hz
- System clock at $f_{clk} = 40$ MHz

The total time needed to perform the acquisition process for one SVN, since the start signal is received until the acquisition ended signal is generated, has been measured in ModelSim simulation as

$$\text{Input signal reading and C/A code generation} + \text{Main loop processing time}$$

$$= 10 \text{ ms } + 1.7 \text{ ms/freq. bin*41 bins} = 79.7 \text{ ms},$$

where the main loop processing time needs is 69.7 ms, that is under 85 ms as it is required.

The hardware resources needed by the proposed acquisition module (Acq_Module_v1) have been calculated using the Quartus II Analysis and Synthesis tool. This tool checks for syntax and semantic errors, minimizes design logic and tries to perform the best possible technology mapping for fitting the design into the Altera Cyclone II EP2C50F484C6 FPGA.

The hardware resources needed for implementing the acquisition module are summarized in table A.4. This table field called "Available", refers to the FPGA available resources after implementing the Namuru-GPL GPS receiver design.

6.7 Summary and conclusions

This chapter has covered the design, implementation and verification of the proposed acquisition module. The proposed acquisition arises as a feasible alternative to the serial search acquisition algorithm that the Namuru-GPL receiver uses by default. Different applications,

Table 6.7: FPGA hardware resources required by acquisition module Acq_Module_v1

Resource	**Used by Acq_Module_v1**	**Total**	**Available**	**% of Availabe**	**% of Total**
Logic Elements	7,223	50,528	28,559	25.3 %	14.3 %
Memory bits	236,845	594,432	506,624	46.7 %	39.8 %
Embedded Multipliers	110	172	161	161	63.9 %
PLLs	0	4	3	0 %	0 %

such as GNSS reflectometry, and possible scenarios, like high dynamics conditions can benefit from having much faster acquisition procedure.

The acquisition module implemented allows to acquire GPS L1 C/A signals using the parallel code acquisition method. This method is considerably much more computationally efficient than serial acquisition, although it requires more hardware resources.

The main design decisions have been justified by the low resources, specially in on-chip memory, in the available Altera Cyclone II FPGA and the acquisition process timing limitations using a system clock at $f_{clk} = 40$MHz.

An interesting approach to considerably reduce the sampling frequency, and thus reduce the size of the FFT module needed, has been implemented. This approach, described in [53] has proved to be an interesting alternative for implementing parallel acquisition in FPGA with low hardware resources. Basically, it reduces the number of samples per C/A code period from 5,714 ($f_s = 5.714$ MHz) to 1,024 by low-pass filtering and then directly downsampling the signal, just like a simple resampling procedure. This "resampling" procedure is applied both to the received signal and to the locally generated C/A code replica. The rest of the implemented acquisition procedure just follows the common parallel code acquisition approach.

A MATLAB's acquistion simulator, has been implemented. It provides a reference framework to analyze the effect of the hardware limitations and quickly introduce design changes without having to implement them in VHDL. Acq_Simulator has proved to be able to detect the presence of C/A GPS signal in over a 20 kHz search band (in -10 kHz to +10 kHz from the f_c in L1).

Although a high level of abstraction for describing the acquisition module implementation was desired, many details have been provided in the end. The reason is that it was really difficult to understand the acquisition module's operation without that details, such as the control module characterization. Details about the main VHDL block implemented as subblocks of the acquisition module can be found in Appendix A. For example, this includes

details about the FFT block and the low-pass filtering and down-sampling block.

The implemented module was only tested with simulation tools. The proposed acquisition module has been designed as functional block of a high order entity (such as the Namuru-GPL receiver) and is not able to work with additional logic to control it. Integrating the acquisition module within a GPS receiver, such as Namuru-GPL, requires a considerable development effort and is left as a possible future work.

However, using advance simulation tools, such as Quartus II and ModelSim, the acquisition module implementation can be considered valid, at least in the beginning. Keeping in mind the initial the design requirements, the acquisition module is able perform the acquisition satisfying the imposed timing limitations and only using a 25.3 % of the available hardware logic elements and a 47 % of the available on-chip memory. Thus, the implementation process can be considered as successful. In order to support that claim, a set of tests to characterize the acquisition module sensibility and resolution have been performed in the lab and are described in the following chapter.

Finally, just to mention that this chapter only describes the final solution. Other design and implementation variations were tried at different development stages. But, in all that cases the proposed approach was considered more appropriate than the others, and some times the only possible due to the FPGA's limited available resources.

Chapter 7

Modified Namuru-GPL and proposed Acquisition Module tests

This chapter is intended to describe the tests performed in order to verify the modified version of the Namuru-GPL receiver proposed in chapter 5, as well as the parallel acquisition module described in chapter 6. Together with the tests description, the results obtained while performing those test are presented.

The chapter starts introducing the main objectives of the tests. These objectives are followed by a detailed description of the experimental environment. This section includes a description of the R&D SMU200A vector signal generator and the Namuru's real-time raw data grabber. The signal generator has been used in every single test, while the Namuru's data grabber has been used for capture raw data to test the proposed acquisition design. The rest of the chapter is divided into two blocks: a first block corresponding to the Modified Namuru-GPL receiver tests and the results obtained when trying to measure the pseudorange differences from comparing code phase delayed C/A GPS signals; and a second block where 3 different type of tests for testing the proposed module resolution in frequency, resolution in code phase delay and sensitivity are depicted. Those block's sections also include the tests' results. This chapter is the last chapter before concluding this document, and it is considered one of the most important because it contains the experimental results obtained for validating the work described in previous chapters.

7.1 Tests main purposes

A set of tests were designed and performed in order to fulfill the next objectives:

1. Testing the design changes implemented in the Namuru-GPL GPS receiver. This includes verifying that the receiver is able to acquire and track C/A GPS signals in the L1 band with its two front-end simultaneously. In addition, the test should probe that such modifications allow the Namuru-GPL to track the same satellite signal with both front-ends. The receiver should be able to successfully measure the pseudoranges difference between the signal received with front-end 1 and front-end 2.

2. Characterizing the behavior of the parallel code acquisition module described in the previous chapter. The tests will provide information about its acquisition capacities. These capabilities refer to:

   (a) Code phase resolution: this is the minimum code phase delay difference that the acquisition module is able to measure.

   (b) Doppler shift frequency resolution: same as previous point, but referred to the received signal's doppler frequency doppler frequency, this refers to the minimum frequency difference that the acquisition module is able to measure.

   (c) Maximum sensibility: this is the minimum received signal power that is necessary to successfully detect and acquire a signal.

7.2 Experimental environment description

Establishing a reference GPS experimental environment can be challenging. The main reason is that the GPS satellites are constantly in movement, thus the set of in-sight satellites and their received signal is constantly in change.

In order to provide a fixed experimental frame that will allow to reproduce the same test multiple times in exactly the same conditions, it was decided to perform the tests in the lab using the available artificial GPS C/A signal generator. This signal generator is able to simulate up to 4 satellite GPS C/A signals in L1 band. This generator has been essential to test both, the Namuru-GPL modification and the proposed acquisition module design, in a controlled environment.

In the case of the proposed acquisition module it has not been integrated into the Namuru-GPL, mainly due to time restraints. Integrating the the acquisition module, or other IP

core, within the Namuru-GPL is not straightforward and requires modifying the receiver's hardware and software.

However, it has been possible to test the module with real signal thanks to an real-time IF raw data grabber implemented with the Namuru board. This data grabber, allows to capture the IF raw data stream generated by the Namuru board's front-end at a sampling rate of 5,714 Msamples/s and transmit it through a USB 2.0 port. This captured raw data can be read by ModelSim and used as an input for the simulating the acquisition process.

The C/A GPS signal generator and the data-grabber are key components in all the proposed tests for the acquisition module. How they are configured and used changes depending on the test and thus these descriptions are included within each corresponding test description. However, it is necessary to understand how the signal generator and the data-grabber work to understand the realized tests and to validate the results obtained. Thus, they are briefly described next.

7.2.1 The C/A GPS vector signal generator

The Remote Sensing Laboratory has a Vector Signal Generator, the R&S SMU200A with the internal digital GPS option SMU-K44. This tool was used in all the tests performed.

The R&S SMU200A has been designed to meet all requirements encountered in research and development of modern communication systems. It combines up to two independent signal generators, with an upper frequency limit of 2.2 GHz, in one cabinet of for height units. Figure 7.1 shows a front-view picture of the R&S SMU200A. The RF output of the signal generator is a type N connector is located at its right bottom corner.

The internal digital GPS option SMU-K44 installed in the SMU200A can simulate dynamic L1 band C/A code signals for the GPS with up to four satellites per baseband. In this case, the SMU200A is equipped with two SMU-K44 options, thus up to 8 satellites can be simulated and controlled easily with the SMU200A's GUI. Since the signal generation is done in realtime with actual almanac data, the satellite signals are realistic.

The SMU-K44 allows to different operation modes:

- Generic mode: in this mode, static satellites with constant Doppler shifts are provided for simple receiver tests, like for example sensitivity tests.
- Localization Mode: dynamic satellites according to a real almanac file are simulated. Hence, the connected GPS receiver can obtain a valid 3D position fix at a user-defined location. As a result, the signal generator can perform not only basic RF tests but also

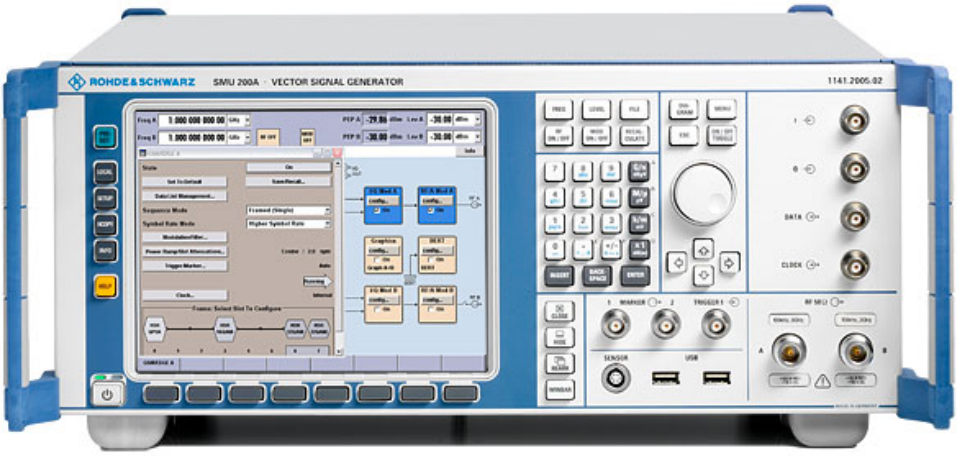


Figure 7.1: ROHDE & SCHARWRTZ SMU-200A Vector Signal Generator front picture

localization tests on GPS receivers. The whole configuration of visible satellites and the generation of the ephemeris out of the uploaded almanac is done automatically by the device. The user only needs to specify time and location. So, a realistic scenario with an optimal satellite constellation with a minimum can be achieved.

In both modes, the GPS signal is generated in realtime (no precalculated waveform) and the output level power ranges from -145 dBm to +20 dBm. More technical details about signal generator and its operational capabilities can be found in [54] and [55].

7.2.1.1 The SMU200A Munich location Scenario

The SMU200A signal generator allows to create a realistic scenario to simulate an specific receiver location. To provide a common framework for all three experiments, the same Munich location scenario was chosen. This scenario is already predefined in the SMU200A. Munich scenario has 4 satellite signals, SVNs 5, 16, 29 and 31. All the scenario's details are summarized in Table 7.1.

7.2.2 The Namuru's real-time IF raw data grabber (RTDG)

Together with the Namuru development tools, a raw IF data logging design, that allows the collection of the GPS data straight from the RF front-end to a file on a PC via the USB 2.0 port, is provided. This is a handy feature for capturing data for software receiver processing

Table 7.1: SMU200A predefined Munich location scenario details

Parameter	**Value**
Reference Frequency [GHz]	1.57542
Total output power [dBm]	-100
C/A code generation filter	Rectangular
Munich location Latitude	48° 9' 0" N
Munich location Longitude	11° 35' 0" E
Munich location Altitude [m]	508
Date [dd.mm.yyyy]	13.07.2008
GPS Mean Time	17:04:00
Satellite elevation mask	7.5°

Parameter	**Satellite 1**	**Satellite 2**	**Satellite 3**	**Satellite 4**
Range Code	C/A	C/A	C/A	C/A
SVN	5	29	2	31
Time Shift [CA-Chips/40]	2,746,618.368	3,054,043.855	3,092,637.722	3,389,813.746
Time Shift [ms]	67.122	74.635	75.578	82.840
Power [dB] respect the total RF output power	-6.2 dB	-6.2 dB	-6.2 dB	-6.2 dB
Doppler Shift [Hz]	-66.79	3,086.33	1,608.68	3,413.53
Duration (Elevation $> 7.5°$) [hh:mm:ss]	03:15:12	05:20:11	02:34:57	04:43:47

and development. In this case, it made possible to simulate and test the proposed acquisition module with real input signal.

The RTDG is divided in two main components: the VHDL logic design that is programmed into the Namuru V2 GNSS board, and a PC software, called Data_grabber console application, that allows to save the data received from the Namuru board through the USB port. Figure 7.2 shows a diagram of the complete RTDG system.

The RTDG logic programmed into the FPGA is responsible of transmitting the front-end generated samples through the USB port. This logic running on the Namuru platform reads in sign and mag raw sampled (2 bit) data at $f_s = 5.714$ MHz. This data has an IF center frequency of 1.405396 MHz. However, this data stream is packed into eight samples into the top of a FIFO as a 16 bit value. The USB chip integrated within the Namuru board (Cypress FX2, see chapter 4) controls the transmission of these samples from the bottom of the FIFO. The samples then travel along the USB cable into the connected PC.

On the other side, the Data_grabber software handles the PC end of the link. This application uses the QuickUSB library [57] provided by Bitwise systems to implement all the USB 2.0 communication functions.

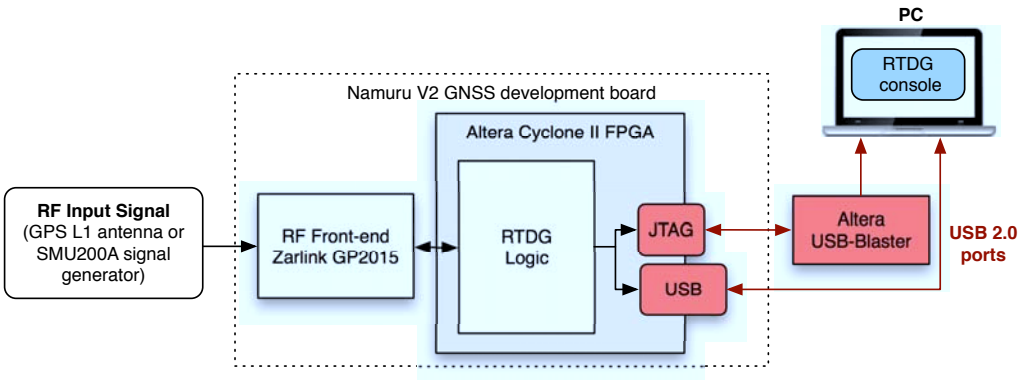


Figure 7.2: RTDG system configuration diagram

Data_grabber software is implemented as small console application that allows raw GPS data to be collected from the Namuru board and logged into a file in real-time. It is very simple and easy to use. The software handles the PC end of the link. It makes use of the QuickUSB library [57] provided by Bitwise systems to implement all the USB 2.0 communication functions.

Once invoked, if all is well with the USB connection, the data console will first ask for a file name ("GPS_data_1.bin" for example), this file will appear in the same directory as the application. Second it will ask how many seconds of data are going to be collected. Note that due to the large buffer sizes used, times less than ten seconds do not work. The application will then communicate with the Namuru's FX2 USB chip and data will flow into the file. Figure 7.3 shows an example of the Data_grabber console use.

While Data_grabber is running, it sees bytes of data arriving in a number of buffers, and unpacks each byte into 4 samples and writes them into a file. The file data format is signed 8 bit values - with 4 possible values: +3, +1, -1, -3. These appear in the file 2's complement.

7.3 Modified Namuru-GPL with two front-ends testing

The Namuru-GPL implemented modification implemented, which allows the receiver to use both available front-ends, needed to be tested. Thus, the objective of this test are:

1. Verifying that the Namuru-GPL 2 front-end receiver is able to acquire and track GPS C/A signals in L1 band in both of its front-ends at the same time. This includes providing pseudorange values for each tracked signal.

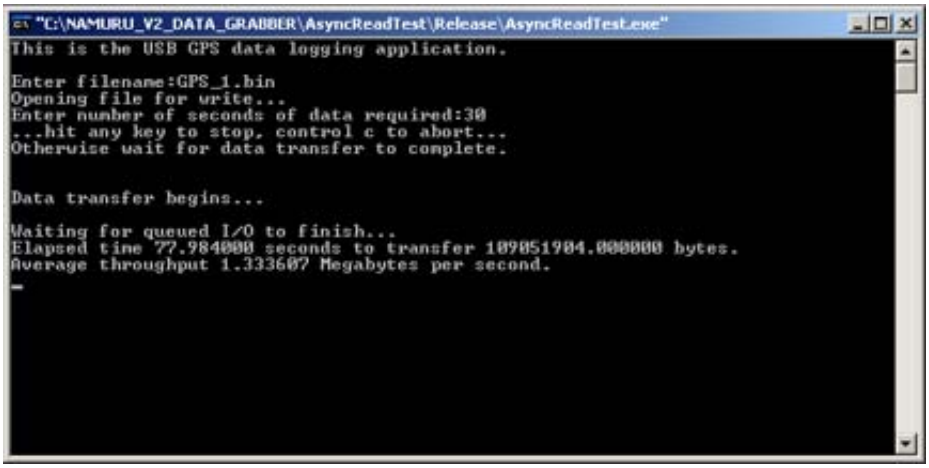


Figure 7.3: Namuru's RTDG console usage example

2. Estimate the minimum time difference between the signal received by front-end 1 (UP or direct signal) and the received by front-end 2 (DOWN or reflected signal) that can be detected as the difference of those signals pseudoranges.

Using the SMU200A signal generator for testing the Namuru-GPL is not straightforward. That is because the SMU200A has only one RF output and this test pretends to test two independent radio front-ends. The front-end 1 is intended to receive only the direct signal, same as the front-end 2 only expects reflected signal. A workaround test solution has been developed to overcome that limitation and it is described in the following section.

7.3.1 Modifications introduced for making the tests possible

One SMU200A generator's RF output should be enough to test both receiver's front-ends if the scheme proposed in Figure7.4 is followed. As it showed in the Figure, this solution combines the use of a particular SMU200A configuration, a power splitter and tuning the Namuru-GPL software. Figure 7.5 shows a picture of the experimental setup during a test run.

7.3.1.1 SMU200A test configuration

Generating to C/A signals corresponding to the same SVNs but delayed in code phase respect each other is possible thanks to the SMU200A signal generator capabilities. The generator

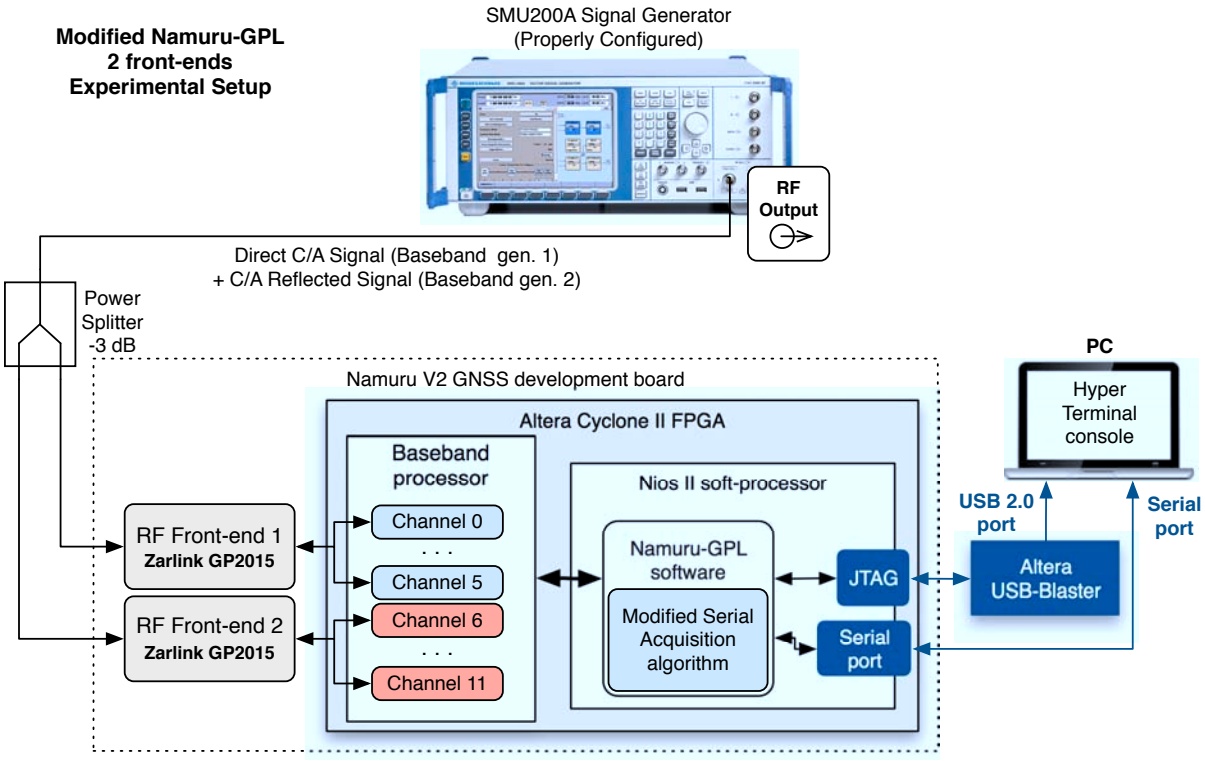


Figure 7.4: Modified Namuru-GPL to work with 2 front-ends: experimental setup scheme

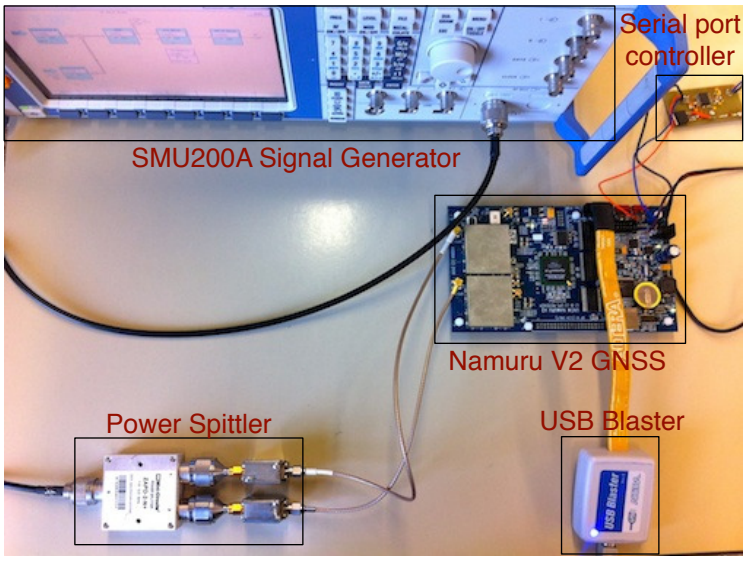


Figure 7.5: Modified Namuru-GPL experimental setup picture

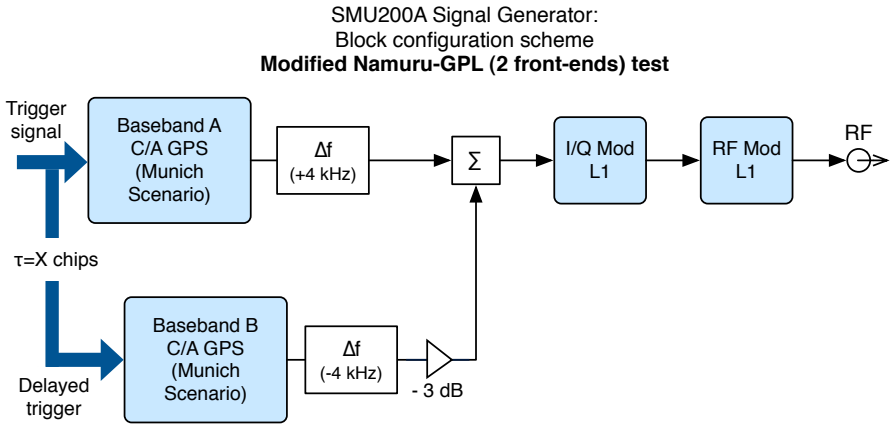


Figure 7.6: SMU200A configuration for measuring testing Modified Namuru-GPL receiver

has two independent baseband C/A code signal generators. Although they work independently, they are driven by the same system clock. The baseband generator 1 will simulate the "direct signal", while "reflected signal" will be simulated by the baseband generator 2.

The configuration established in the SMU200A for the test is the following:

1. The C/A code baseband generator 1 is configured to simulate the default Munich location scenario. C/A baseband generator 2 is also configured to simulate the same scenario. The C/A signal generated by the baseband generator 2, the "reflected" signal, is attenuated 3 dB before being added to the baseband generator 1 signal, at baseband. Figure 7.6 shows a diagram of the SMU200A configuration.

2. The code phase difference between both baseband generators can be controlled. The baseband generator 2 is configured to be triggered by the first one, but with a constant delay. This code phase delay will be set to different values to simulate the different propagation path lengths followed by the "direct" and "reflected" signals.

3. An additional frequency offset is added to each baseband generator output. For baseband generator number 1 a +4 kHz frequency offset is added, while a −4 kHz is added in the case of baseband generator number 2. The reason for adding such frequency is described in the Namuru-GPL software tuning description.

4. The SMU200A total RF power is set to −100 dBm. Note that C/A signals produced by baseband generator 2 are attenuated 3 dB. Thus, the "direct" signals have a −106.2 dBm power, while the "reflected" signals have −109.2 dBm.

7.3.1.2 Mini-circuits ZAPD-2+ Power Splitter

The proposed experimental setup divides the SMU200A output into two signals using an RF power splitter from Mini-Circuits [58](obviously introducing an attenuation of 3 dB + 0.25 dB insertion losses at the output signals). This power splitter is intended to be used for applications such as GPS, satellite distribution and other communication systems. Its 2 way-0° splitter with an operational frequency range of 1 GHz - 2 GHz ($f_L - f_U$). The three splitter ports have a 50 Ω impedance and use N-Type connectors.

7.3.1.3 Namuru-GPL software tuning

With the proposed test setup, simulated "direct" and "reflected" signals are present in both front-ends. The receiver should be able to classify the received signal depending on some of its properties in order to determine if it is dealing with a "direct" or "reflected" signal.

In order to do that, the receiver's acquisition procedure has been modified. The Namuru-GPL receiver implements a serial search acquisition algorithm. This algorithm sequentially searches through a set of different possible frequency offset steps and code phase delays. This search has been modified such as:

1. The acquisition search domain for 6 receiver's channels associated with front-end 1 (channels 0 to 5) only covers positive frequency Doppler shifts.

2. The acquisition search domain for 6 receiver's channels associated with front-end 2 (channels 6 to 11) only covers negative frequency Doppler shifts.

By doing that, it is possible to differentiate between a "direct" signal (positive frequency shift) or a "reflected" signal (negative frequency shift). This division of the acquisition search frequency domain is represented in Figure 7.7.

Thus, it is necessary to configure the SMU200A in order to ensure that all the satellite signals generated by baseband generator 1 have positive Doppler frequency values, while the satellite signals generated by baseband generator 2 have negative Doppler frequency values.

This is accomplished by adding frequency offsets at the output of the both baseband generators (+4 kHz and −4 kHz) as it was already shown in Figure 7.6.

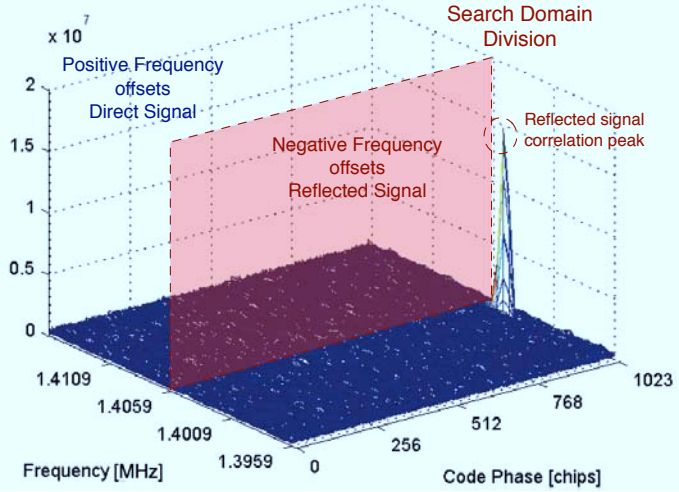


Figure 7.7: Modified Namuru-GPL search acquisition domain for the test proposed

7.3.2 Code phase difference between signals test

7.3.2.1 Description

The purpose of this test is to measure the pseudorange differences ($\triangle\rho_{\tau_i}$) between the "direct" signal captured by the Modified Namuru-GPL front-end 1 and the "reflected" signal captured by front-end 2. As it described in previous sections, the tests are performed in the laboratory using simulated C/A GPS signals in L1 generated by the SMU200A. If the measurement these $\triangle\rho_{\tau_i}$ successfully is possible, it should confirm that the Modified Namuru-GPL concept is valid for applications that require comparing the same signal received with two different propagation delays, such as GNSS-R altimetry applications.

At the beginning of the proposed test, the Modified Namuru-GPL receiver locked the 4 "direct" signals (channels 0 to 3, associated with front-end 1) and the 4 "reflected" signals (channels 5 to 9, associated with front-end 2). After successfully locking the 8 signals, the receiver starts to provide computed pseudorange values for those signals. These pseudorange values are sent via Serial port (see chapter 5) to a PC terminal console and logged into a text file. The pseudorange data was sent to the PC console once per second.

This pseudoranges login process has been performed for 4 different code phase delays (τ_i) between "direct" and "reflected" signals during 300 seconds:

1. $\tau_{4\ \text{chips}} = 4$ chips $= 4.092\ \mu s$
2. $\tau_{2\ \text{chips}} = 2$ chips $=2.046\ \mu s$

3. $\tau_{1 \text{ chip}} = 1 \text{ chip} = 1.023\ \mu s$
4. $\tau_{0.5 \text{ chips}} = 0.5 \text{ chips} = 0.512\ \mu s$

The code phase delay was equally applied to all the satellite "reflected" signals.

In addition to those code phase delays, a code phase delay $\tau_{0.51 \text{ chips}}$ =0.51 chips was also tested. With the SMU200A generator signal selected configuration, 0.01 chips ($\tau_{0.01 \text{ chips}} = 10.23\mu s$) is the delay value admitted to synchronize the 2 internal baseband C/A generators at chip level. This value also tests the Namuru-GPL receiver's capabilities, especially pseudorange measurement's routine. This test is intended to estimate if the Modified Namuru-GPL is able to detect small code phase delay differences ($\sim$ 3 m) between the "direct" and "received" signals, at least in a very controlled experimental environment.

7.3.2.2 Results

Figure 7.8 shows an example of the data received through the Serial port during the test execution. In this capture, the first four channels (0 to 3) have locked the "direct" signals, while the channels 5 to 8 have locked the "reflected" signals. In both cases, the pseudoranges provided were used to compute the measured pseudorange differences ($\triangle\rho_{\tau_i}$).

Figure 7.9 shows an example of these $\triangle\rho_{\tau_i}$. These differences are classified by SVN in 2 plots (SVNs 2 and 5). Each plot depicts the $\triangle\rho_{\tau_i}$ results for each code phase delay difference between "direct" and "reflected" signal.

In some cases, the plots present abrupt edges. These edges are produced when one of the signals that is in locked state is lost while the other, corresponding to same satellite, remains locked. When a signal is in locked state and loses its pseudorange, the value is set to 0. If the "reflected" signal tracking is lost, the $\triangle\rho_{\tau_i}$ value will present a highly negative value ($0 - \rho_{direct}$), on the other hand, if the lost signal is the "direct" signal, this ($\triangle\rho$) will show a very high positive value ($\rho_{reflected} - 0$).

In addition, Figure 7.10 shows an example of the pseudorange values evolution for SVN 2 during a capture. This plot shows how pseudorange values are decreasing with time because of satellite distance is decreasing. Note that a pseudorange estimation error or adjustment produced by the Namuru-GPL received is represented by the small peak at 156 s of capture time. The plot capture time range (x-axis) was properly selected in order to be able to distinguish between both signals pseudorange represented values.

A summary of all the pseudorange measurement results for code phase delays differences from $\tau_{4 \text{ chips}}$ to $\tau_{0.5 \text{ chips}}$ chips can be found in Table 7.2. Note that $\triangle\rho_{\tau_i}$ values are presented in meters for a propagation speed in free space, $c = 2.99792458{\cdot}10^8$ m/s.

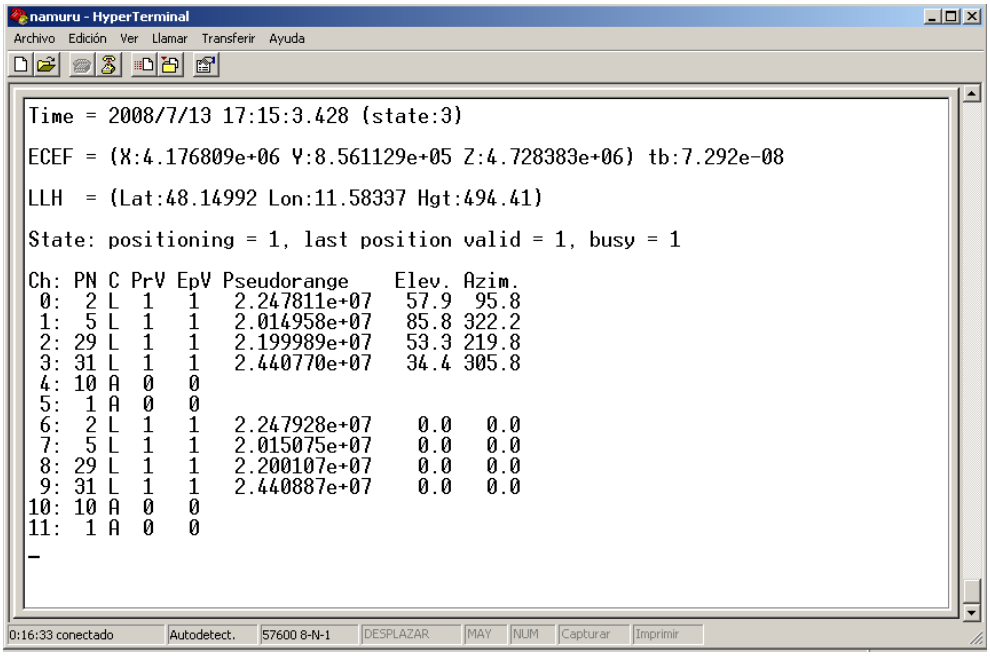


Figure 7.8: Example screen capture while measuring the pseudorange differences test

On the other side, Figure 7.11 shows the results obtained for SVNs 2 and 5 when comparing the code phase delay differences of 0.5 and 0.51 chips between "direct" and "reflected" signals. The average pseudorange difference values (represented in the Figure with dashed line and computed for 150 samples). Table 7.3 summarizes the average and standard deviation of the $\triangle\rho_{\tau_i}$ values for a code phase delay of 0.51 between signals. The table also includes the $\triangle\rho_{\tau_{0.51\text{ chips}}} - \triangle\rho_{\tau_{0.50\text{ chips}}}$ mean values in meters. The calculated values differ $\sim$1 m with the expected value (3 m) in the worst case.

7.4 Acquisition module tests

Three different tests are proposed to characterize the acquisition module capabilities:

1. A sensitivity test, that will provide an estimation of the minimum signal power needed at the acquisition module to distinguish a clear correlation peak.

2. A frequency resolution test, to verify that the acquisition module is able to detect Doppler frequency shifts of 500 Hz.

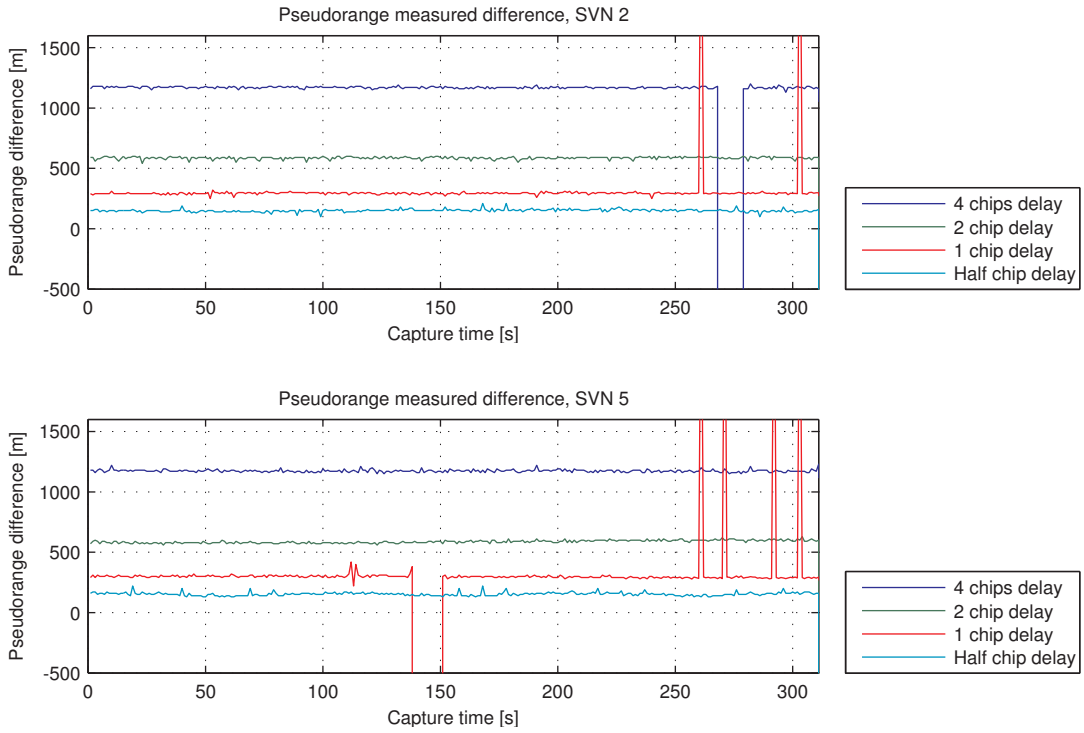


Figure 7.9: Pseudorange differences ($\triangle\rho_{\tau_i}$) for $\tau_{4\text{ chip}}$, $\tau_{2\text{ chip}}$, $\tau_{1\text{ chip}}$ and $\tau_{0.5\text{ chips}}$, code phase delay differences in SVNs 2 and 5 signals.

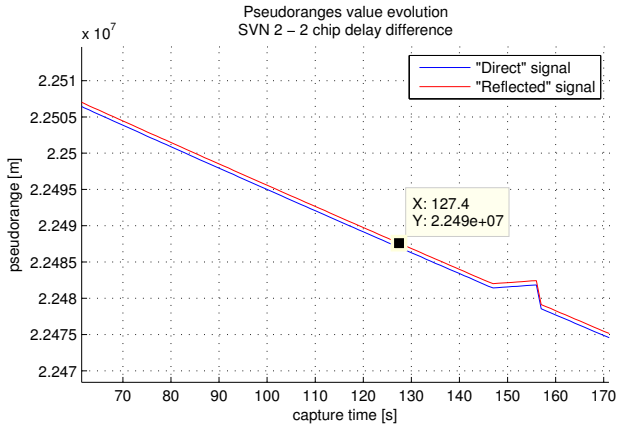


Figure 7.10: Example of measured pseudorange evolution for SVN 2, "reflected" signal delayed 2 chips

Table 7.2: Pseudorange differences ($\triangle\rho_{\tau_i}$) for Munich scenario with 2 C/A independent signals for 4 different code phase delays (τ_i) between them

4-chip code phase delay ($\tau_{4\text{ chips}}$)

	SVN 5	**SVN 29**	**SVN 2**	**SVN 31**
Average value [m]	1173.5	1170.7	1169.9	1172
Std deviation [m]	10.991	22.309	7.809	22.152

2-chip code phase delay ($\tau_{2\text{ chips}}$)

	SVN 5	**SVN 29**	**SVN 2**	**SVN 31**
Average value [m]	587.6	586	586.2	586.2
Std deviation [m]	11.954	13.749	10.175	11.752

1-chip code phase delay ($\tau_{1\text{ chip}}$)

	SVN 5	**SVN 29**	**SVN 2**	**SVN 31**
Average value [m]	297.5	292.3	293.6	291.8
Std deviation [m]	14.160	14.943	8.164	14.191

Half-chip code phase delay ($\tau_{0.5\text{ chips}}$)

	SVN 5	**SVN 29**	**SVN 2**	**SVN 31**
Average value [m]	152.672	150.739	150.733	150.727
Std deviation [m]	16.021	23.708	13.641	13.031

Table 7.3: Pseudorange differences ($\triangle\rho_{\tau_i}$) for a code phase difference $\tau_{0.51\text{ chips}}$ and comparison with $\tau_{0.50\text{ chips}}$

	SVN 5	**SVN 29**	**SVN 2**	**SVN 31**
Average value [m]	155.682	152.418	154.743	153.049
Std deviation [m]	11.8337	21.9906	9.4110	28.2966
Difference with 0.5 chip code phase delay average [m]	155.682 − 152.672 =4.0096	152.418 − 150.739 =3.0107	155.743 − 150.733 = 2.6797	153.049 − 150.727 =2.3227

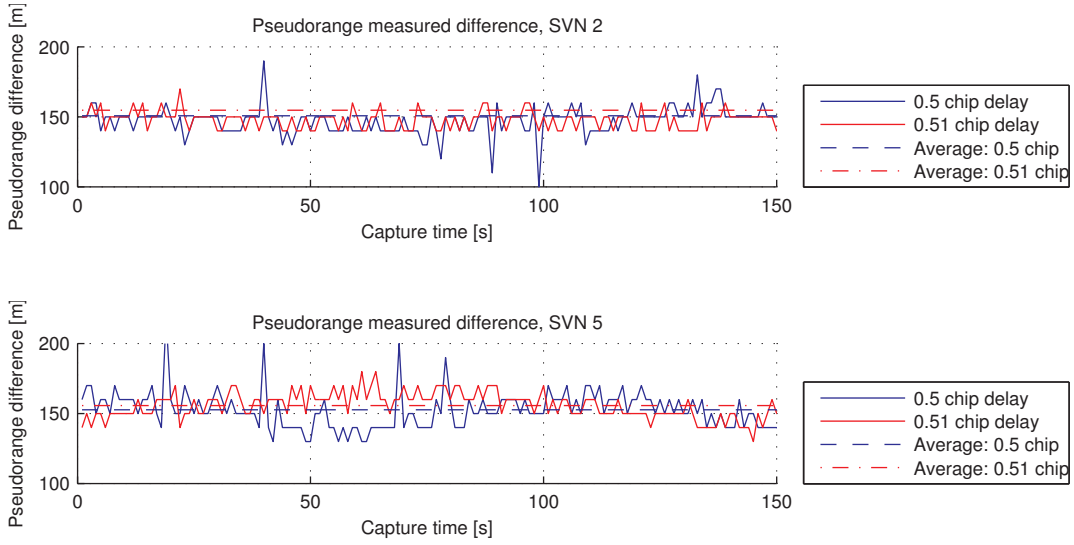


Figure 7.11: Pseudorange differences ($\triangle\rho_{\tau_i}$) for $\tau_{0.51\ \text{chips}}$ and $\tau_{0.50\ \text{chips}}$, in SVNs 2 and 5 signals

3. A code phase resolution test to determine if the acquisition module's resolution while determining the C/A code phase.

The procedure followed for the three tests is the same and it is depicted in Figure 7.12. The first step is to set the components as it is described in Figure 7.2 and load the RTDG design in the Namuru board. Figure 7.13 shows a picture of the experimental setup taken during the tests.

Then, the SMU200A configuration is set according to the test. Right after that start capturing data using the RTDG Data_console application. Once the data has been successfully acquired. This data is loaded as input for the acquisition simulation using the ModelSim VHDL model. At the same time this input data is also loaded into the MATLAB environment to also simulate the acquisition process with the Acq_Simulator MATLAB tool implemented.

Once the ModelSim simulation is over, which can take up to 4 hours, the obtained output data is loaded into MATLAB work space, this is the generated DDM and the acquisition results (max peak value and position, second-peak value and sum of all the DDM's values). Finally, the ModelSim generated DDM data can be processed, if it is needed, and plotted using MATLAB.

In all tests, the MATLAB's Acq_Simulator described in the last chapter was used as Reference Model to validate the ModelSim results comparison. However, these Acq_Simulator

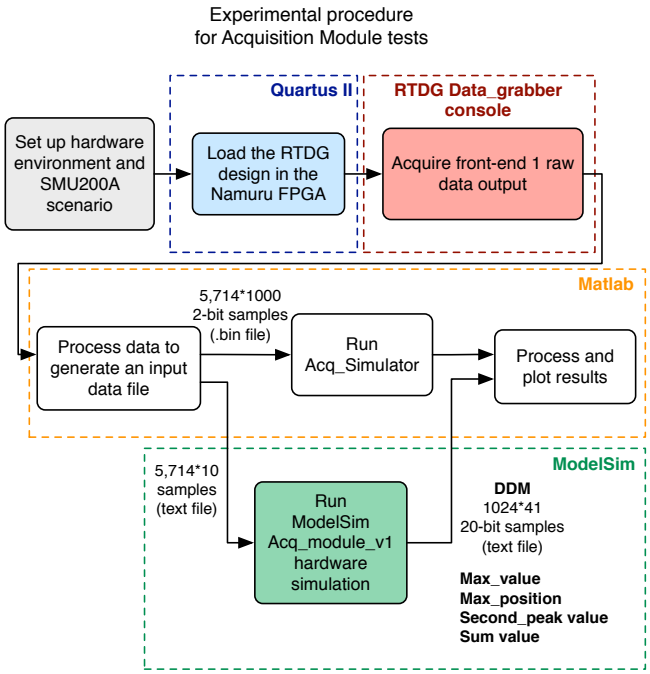


Figure 7.12: Diagram of the experimental procedure for Acquisition Module tests

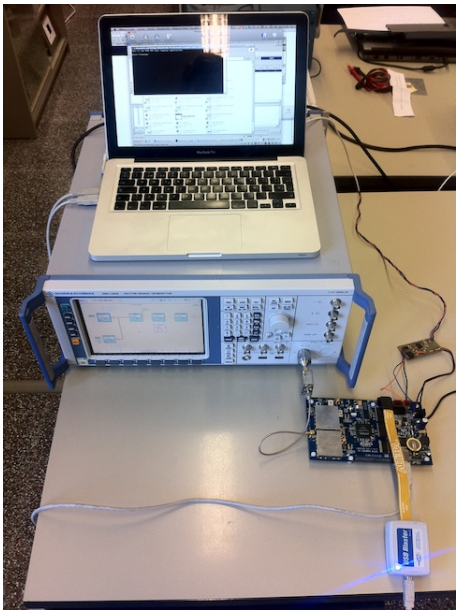


Figure 7.13: Picture of the experimental setup taken during the Acquisition Module tests

reference results are shown only when they have been considered relevant.

7.4.1 Proposed acquisition indicators

In order to set some parameters to quantitatively compare and validate the acquisition results, 2 simple acquisition indicator are proposed. These indicators are calculated using the acquisition module outputs and have been specially relevant for the sensitivity test. These proposed acquisition indicators are:

1. Max-to-average ratio:

   This ratio has defined as:

   $$\text{Max-to-average} = \frac{\text{Max}(DDM)}{\text{Mean}(DDM)}. \tag{7.1}$$

   This ratio can be understood as an approximation of the Signal-to-Noise ratio, since the majority of DDM values that contribute to the mean are noise.

2. Max-to-second peak ratio:

   This ratio has been defined as:

   $$\text{Max-to-2ndpeak} = \frac{\text{Max}(DDM)}{\text{Second\_peak}(DDM)}, \tag{7.2}$$

   where Second_peak(DDM) represents the second highest DDM value. To be sure that this value belongs to a different peak, and is not a part of the main peak, it has to be located at least at 2 chips of distance from the main peak.

Once this acquisition parameters are calculated, they can be compared to predefined thresholds to decide if there is or not a reliable received signal. Setting these thresholds strongly depends on the application. In some cases low thresholds are desired to acquire weak signals, even if that increases the probability of false alarm errors. In other cases, higher thresholds are desired to only acquire signals with strong SNR.

Note that more complex indicators can be defined in order to perform more accurate acquisition decisions. But since this indicators should be calculated by the system within the acquisition module will be integrated, they are not discussed in this document.

7.4.2 Acquisition's sensitivity test

7.4.2.1 Description

The goal of this test is to provide an approximate measurement of the minimum power level required to detect the presence of a signal.

The SMU200A signal generator parameters were set those corresponding to Munich's scenario (see Table 7.1). For the same scenario, 3 different captures were taken. Each capture has been performed selecting a different total RF output power were selected in the SMU200A. The selected total RF output power levels have been: -110 dBm, -113 dBm and -115 dBm.

Since obtaining the ModelSim simulation results was too time consuming, first the acquired data was used as input for the MATLAB Acq_Simulator. Using its Reference Model, computing a signal DDM only requires a few seconds. Thus, if no clear correlation peak is found for a given signal power using the Reference model, it is possible to ensure that no significant correlation peak will be found in the ModelSim simulation. Moreover, the Reference model has shown to generate DDMs with max-to-average ratios always higher than the ModelSim simulation results. This is not surprising since the Acq_module_v1 uses fixed-point data representation (with no decimal part) which introduces data precision losses.

The Acq_Simulator's Reference model shows a barely identifiable max correlation peak for a total output power of -115 dBm. That is why two power levels under -115 dBm were also selected. These power levels were deliberately chosen close to -115 dBm to estimate the acquisition module minimum signal power.

Note that the mentioned power levels correspond to the SMU200A total RF signal output power. Four satellites are present in the Munich scenario and its individual signal power is 6.02 dB below the total output power.

For each power level, the max-to-average and max-to-second peak ratios are provided for each present satellite signal.

7.4.2.2 Results

For each of the 3 selected generator output power levels (-110 dBm, -113 dBm and -115 dBm) the next four plots are included:

1. MATLAB Acq_Simulator - Reference Model: Mesh plot.
2. ModelSim Acq_Module v1 - DDM_out output: Mesh plot. This plot directly represents the values obtained during the ModelSim VHDL simulation (DDM_out).

3. MATLAB Acq_Simulator - Reference Model, X-Z plot.

4. ModelSim Acq_Module v1 - DDM_out output: X-Z plot.

While plots 1 and 2 provide a general view of all the computed correlation values or the DDM, plots 3 and 4 provide a more accurate view of the correlation peak code phase. All plots show the normalized correlation results.

Only the plots obtained for SVN 31 have been represented.

In addition to those plots, for each power level, a table with the computed acquisition indicators summary (for all the present satellites) is included.

Figures 7.14 and 7.15 and Tables 7.4 and 7.5 show the results obtained for a signal power of -116 dBm (total power -110 dBm). A clear correlation peak can be found in both, the Acq_Simulator and ModelSim, results. This power was selected to exemplify a situation in which the received signal SNR is high, such as the direct signal or strong reflected signal in GNSS-R applications.

Figures 7.16 and 7.17 and Tables 7.6 and 7.7 show the results for a signal power of -119 dBm. Although a correlation peak is still visible in both simulation results, the max-to-second peak indicator has been considerably reduced in all cases. The max-to-average is also degraded, but the Acq_Module_v1 has been able to successfully acquire the satellite signal.

Finally, Figures 7.18 and 7.19 show the results for a signal power of -121 dBm. In addition, Tables 7.8 and 7.9 show the the obtained acquisition results. In this case, the Acq_Module_v1 has been able to acquire only 2 of the 4 present satellite signals (SVNs 5 and 29), while for SVNs 2 and 31 the acquisition failed. Looking at Figure 7.19 it becomes clear why this happened. The signal correlation peak is too low and thus it remains hidden in front of higher DDM noisy values. Although the results for SVNs 2 and 31 is not valid, the Acq_Module_v1 results is correct in the sense that it actually provides the maximum DDM value and position, the second-highest value and the sum of all DDM values.

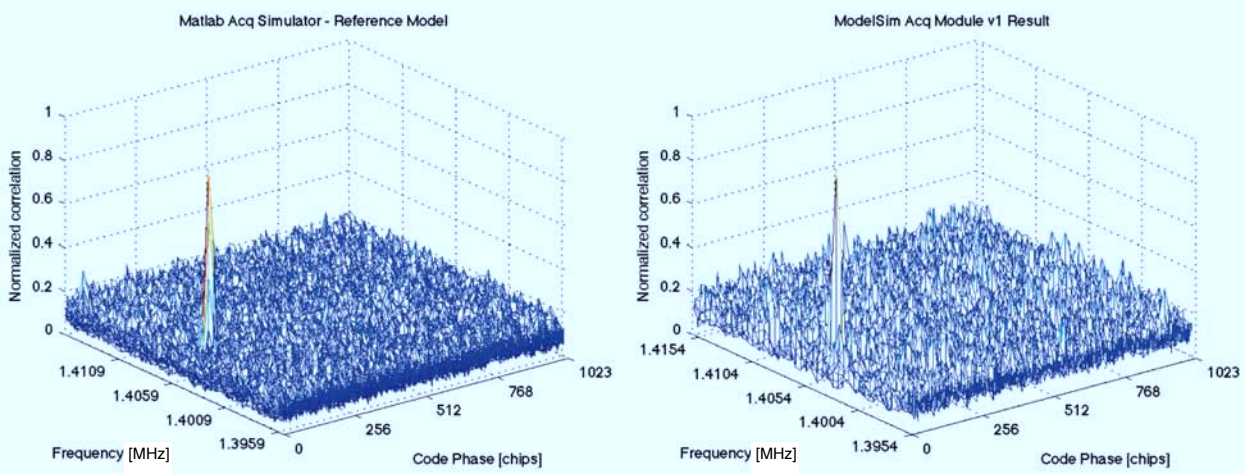


Figure 7.14: SVN 31's DDMs, satellite signal power =−116 dBm

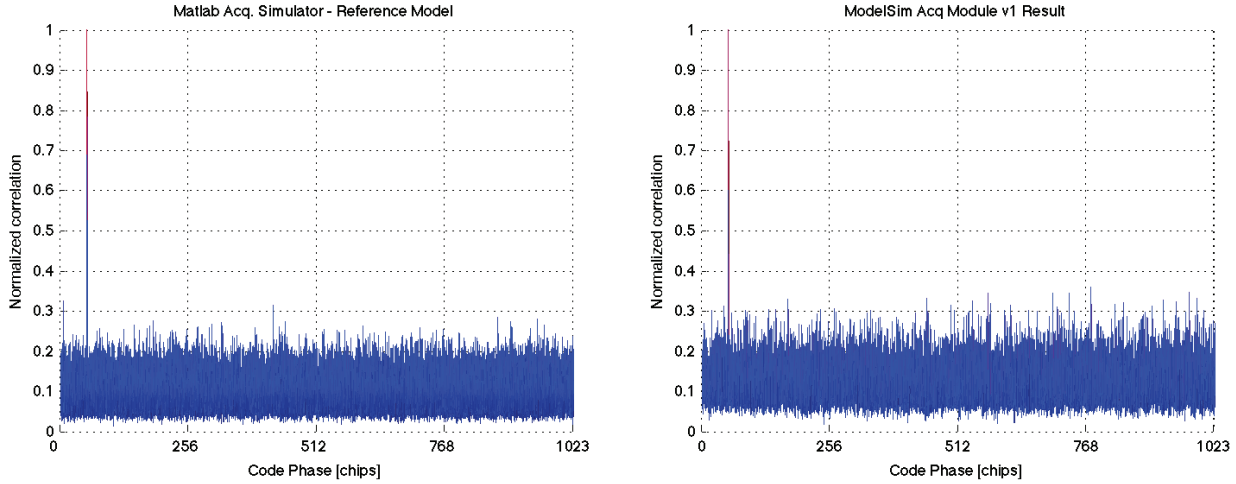


Figure 7.15: SVN 31's DDMs, X-Z view, satellite signal power =−116 dBm

Table 7.4: Sensitivity test results Acq_Simulator - Reference Model: SMU200A Output power −110 dBm (−116 dBm per satellite)

Indicator	**Satellite 1**	**Satellite 2**	**Satellite 3**	**Satellite 4**
SVN	5	29	2	31
Max value position [Code phase; Frequency]	380.4471 chips; 1.4074 MHz	869.5679 chips; 1.4039 MHz	828.0320 chips; 1.4054 MHz	52.9940 chips 1.4039 MHz
Max value position [Samples; Frequency bins]	2125; 24	4857; 17	4625; 20	296; 17
Max-to-average ratio (10 simulations average)	8.2968	9.7733	8.0889	10.1133
Max-to-second peak ratio (10 simulations average)	3.1824	3.1566	2.9194	3.8390

Table 7.5: Sensitivity test results ModelSim (Acq_Module_v1): SMU200A Output power −110 dBm (−116 dBm per satellite)

Indicator	**Satellite 1**	**Satellite 2**	**Satellite 3**	**Satellite 4**
SVN	5	29	2	31
Max value position [Code phase; Frequency]	381 chips; 1.4074 MHz	870 chips; 1.4039 MHz	829 chips; 1.4054 MHz	53 chips 1.4039 MHz
Max value position [Samples; Frequency bins]	381; 24	870; 17	829; 20	53; 17
Max-to-average ratio (10 simulations average)	5.4397	6.6210	6.0638	8.1230
Max-to-second peak ratio (10 simulations average)	1.7823	1.6264	1.4238	1.799

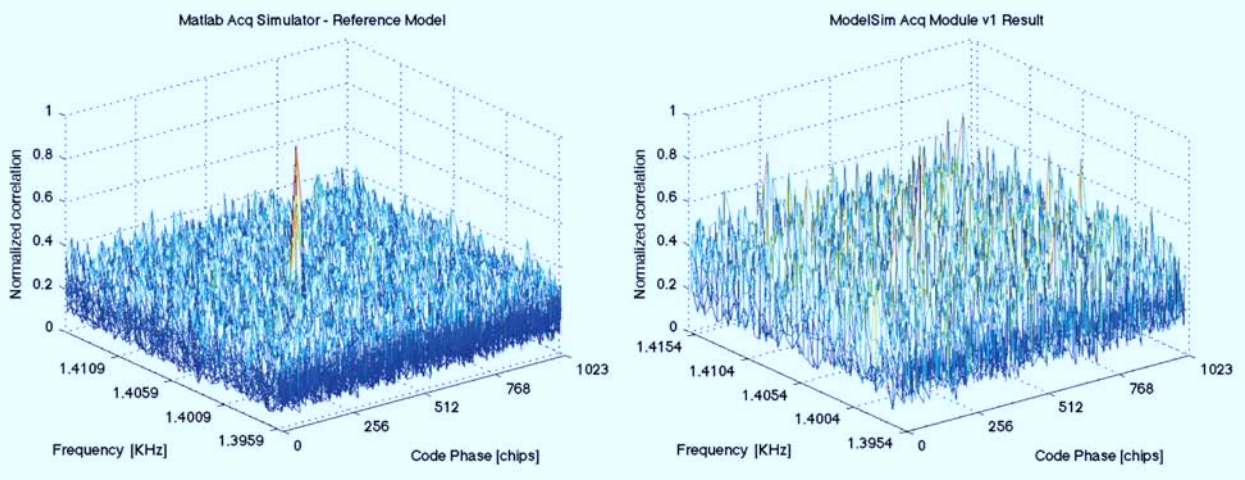


Figure 7.16: SVN 31's DDMs, satellite signal power =−119 dBm.

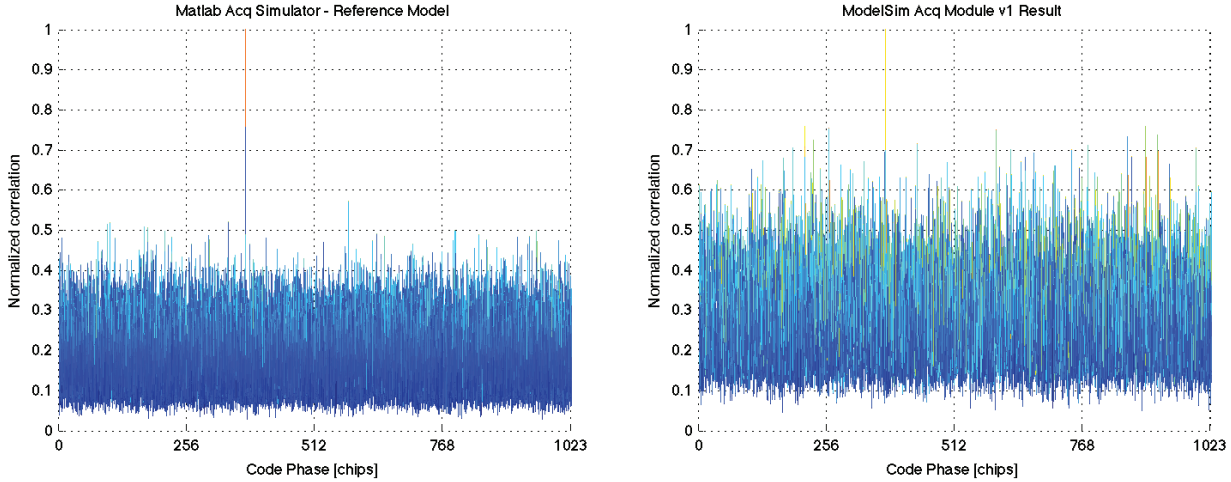


Figure 7.17: SVN 31's DDMs, X-Z view, satellite signal power =−119 dBm

Table 7.6: Sensitivity test results Acq_Simulator - Reference Model: SMU200A Output power −113 dBm (−119 dBm per satellite)

Indicator	**Satellite 1**	**Satellite 2**	**Satellite 3**	**Satellite 4**
SVN	5	29	2	31
Max value position [Code phase; Frequency]	55.32 chips; 1.4074 MHz	203.56 chips; 1.4044 MHz	323.69 chips; 1.40549 MHz	372.03 chips 1.4039 MHz
Max value position [Samples; Frequency bins]	309; 24	1137; 18	1808; 21	2078; 17
Max-to-average ratio (10 simulations average)	5.7030	4.9165	5.2657	5.5203
Max-to-second peak ratio (10 simulations average)	2.0987	1.6617	1.8725	2.0054

Table 7.7: Sensitivity test results ModelSim - Acq_Module_v1: SMU200A Output power −113 dBm (−119 dBm per satellite)

Indicator	**Satellite 1**	**Satellite 2**	**Satellite 3**	**Satellite 4**
SVN	5	29	2	31
Max value position [Code phase; Frequency]	56 chips; 1.4074 MHz	204 chips; 1.4039 MHz	324 chips; 1.4054 MHz	372 chips 1.4039 MHz
Max value position [Samples; Frequency bins]	56; 24	204; 18	324; 21	372; 17
Max-to-average ratio (10 simulations average)	3.4014	3.6742	5.3114	3.6812
Max-to-second peak ratio (10 simulations average)	1.1426	1.0393	1.2343	1.2391

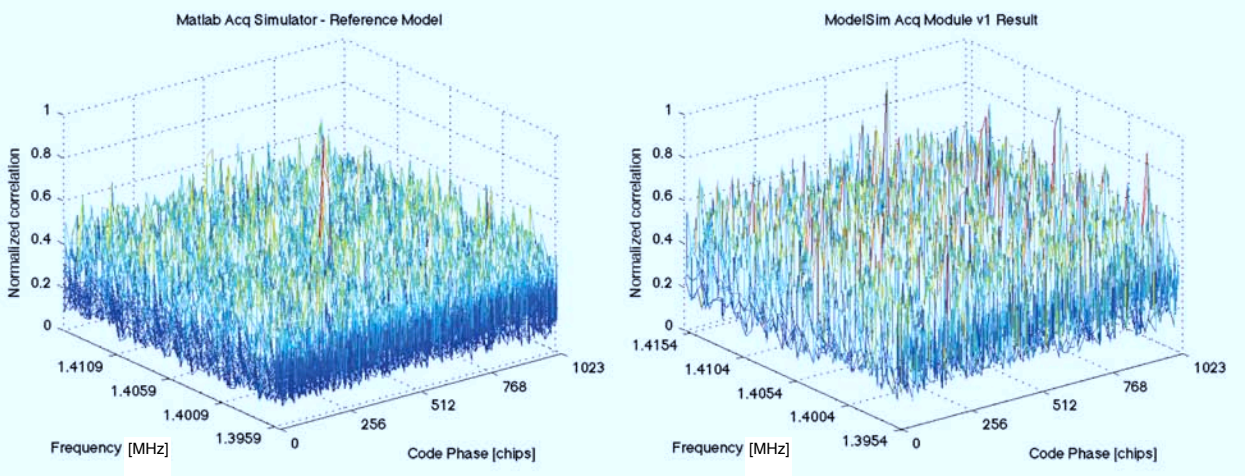


Figure 7.18: SVN 31's DDMs, satellite signal power =−121 dBm.

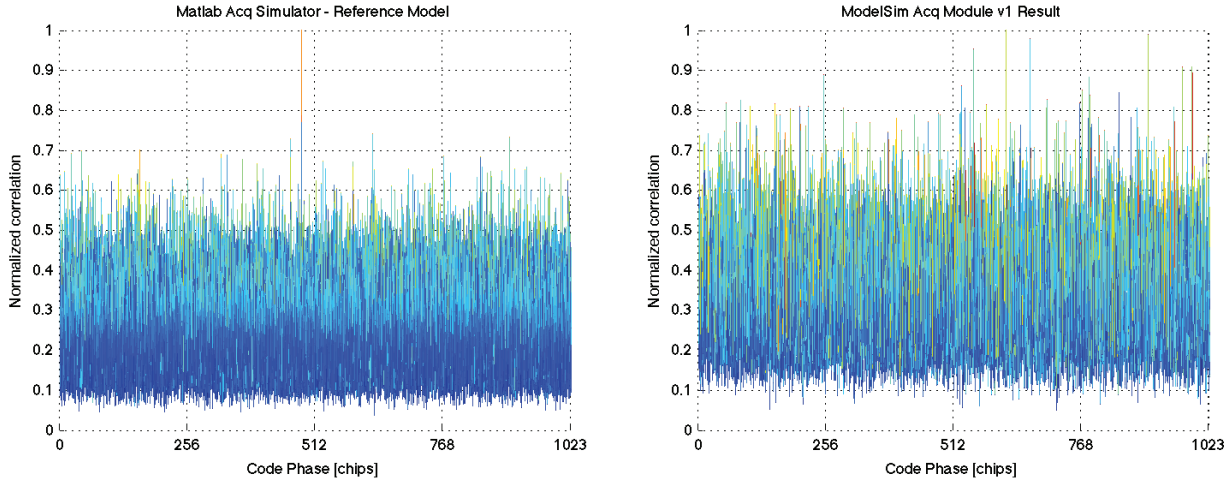


Figure 7.19: SVN 31's DDMs, X-Z view, satellite signal power =−121 dBm

Table 7.8: Sensitivity test results Acq_Simulator - Reference Model: SMU200A Output power −115 dBm (−121 dBm per satellite)

Indicator	**Satellite 1**	**Satellite 2**	**Satellite 3**	**Satellite 4**
SVN	5	29	2	31
Max value position [Code phase; Frequency]	663.14 chips; 1.4074 MHz	370.42 chips; 1.4044 MHz	703.78 chips; 1.40549 MHz	483.21 chips 1.4039 MHz
Max value position [Samples; Frequency bins]	3704; 24	2069; 18	3931; 21	2699; 17
Max-to-average ratio (10 simulations average)	5.1976	3.7354	3.5281	4.0102
Max-to-second peak ratio (10 simulations average)	1.9341	1.3318	1.2393	1.4475

Table 7.9: Sensitivity test results ModelSim - Acq_Module_v1: SMU200A Output power −115 dBm (−121 dBm per satellite)

Indicator	**Satellite 1**	**Satellite 2**	**Satellite 3**	**Satellite 4**
SVN	5	29	2	31
Max value position [Code phase; Frequency]	664 chips; 1.4074 MHz	371 chips; 1.4039 MHz	**740 chips; 1.4054 MHz**	**616 chips 1.4039 MHz**
Max value position [Samples; Frequency bins]	56; 24	371; 18	**740; 19**	**616; 35**
Max-to-average ratio (10 simulations average)	3.2051	3.0224	3.1893	3.0826
Max-to-second peak ratio (10 simulations average)	1.023	1.0117	1.0822	1.1493

7.4.3 Doppler frequency shift resolution test

7.4.3.1 Description

The goal of this test is to verify that the supposed 500 Hz resolution of the proposed acquisition module is reached.

The SMU200A signal generators were again set to those corresponding to Munich's scenario (see Table 7.1). Following a similar procedure, 3 different captures were performed. In this case, the first capture was the original Munich scenario, where in the other 2, an additional Doppler shift was added to each satellite signal. The additional Doppler shift values used have been 500 Hz and 1000 Hz, corresponding to one and two 500 Hz-frequency bins.

The frequency bin size of the acquisition module is 500 Hz. Thus, if the acquisition module works properly, the maximum carrier frequency estimation error should be upper-bounded to 250 Hz.

7.4.3.2 Results

Figure 7.20 shows three SVN 31 Acq_Module_v1 results (computed with ModelSim) corresponding to frequency Doppler shifts of 0 Hz, 500 Hz and 1000 Hz. The Figure represents the low values in blue and moves to red passing through yellow for high values. This is the default MATLAB's "jet" colormap. Taking the first plot as reference, it is clear how the correlation peak correctly moves a frequency bin in each case.

In the other hand, Figure 7.21 shows in this case three SVN 31 Acq_Simulator results (using Reference Model) for frequency Doppler shifts of 0 Hz, 500 Hz and 1000 Hz. The use of a higher sampling frequency, provides a more detailed result in the code phase dimension.

In all the plots, the secondary sync function peaks are visible. Same results where obtained for SVNs 2, 5 and 29, where in all cases the receiver successfully noticed the 500 Hz frequency shift.

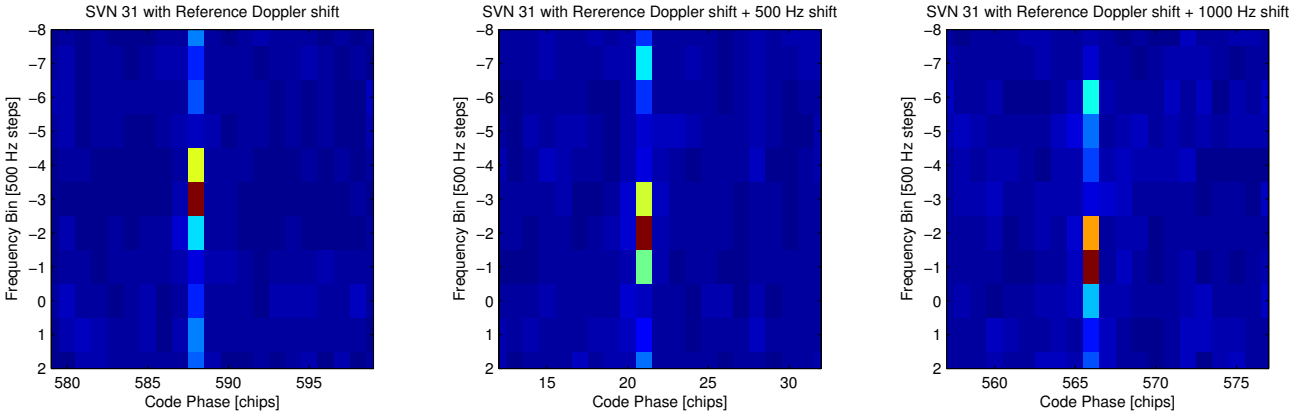


Figure 7.20: Detail correlation results for frequency Doppler shifts of 0 Hz, 500 Hz and 1000 Hz using the Acq_Module_v1

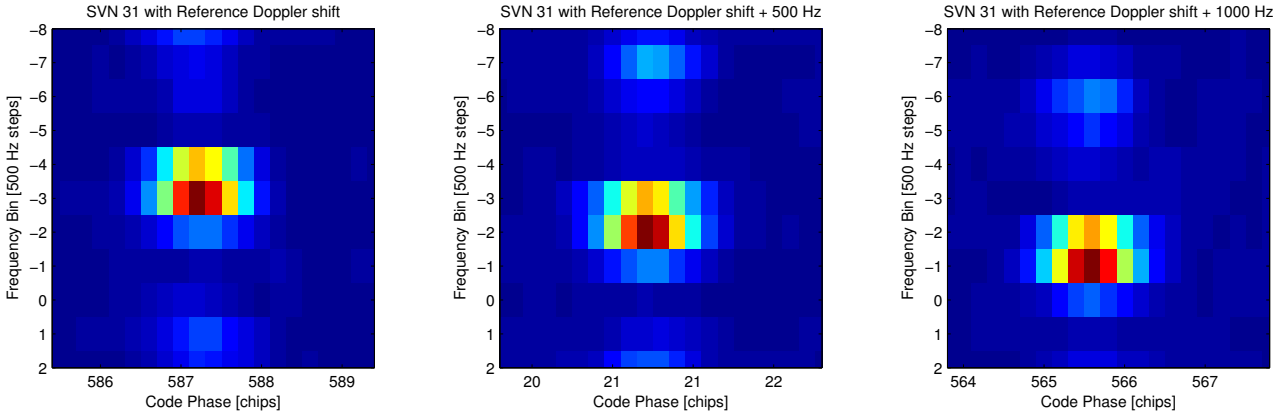


Figure 7.21: Detail correlation results for frequency Doppler shifts of 0 Hz, 500 Hz and 1000 Hz using the Acq_Simulator (Reference Model)

7.4.4 Code phase resolution test

7.4.4.1 Description

The goal of this test is to measure the code phase resolution of the acquisition module.

Each time that a data capture is performed using the RTDG, the obtained value for the code phase delay is different. This has a lot of sense, since there is no synchronization between the C/A code signal generation and data capture process. In order to be able to measure

with precision the code phase resolution, two signals corresponding to the same SVN will be simultaneously generated, but with a previously fixed code phase between them. By doing that, in each data capture the position of the two correlation peaks within the obtained DDM will be different, but their distance to each other in code phase will be constant.

As it was already used for testing Namuru-GPL 2 front-end modification, the SMU200A can generate C/A signals corresponding to the same SVNs but delayed in code phase respect each other. The generator has two independent baseband C/A code signal generators. Although they work independently, they are driven by the same system clock.

The configuration established in the SMU200A for the test is the next:

1. The first C/A code baseband generator is configured to simulate the default Munich location scenario. The second C/A baseband generator is also configured to simulate the same scenario. This implies that the same satellite signal will be present twice. Even when it is not strictly necessary, the C/A signal generated by the baseband generator 2 is attenuated 3 dB before being added to the baseband generator 1 signal, at baseband. If both C/A signals are not perfectly in sync, the resulting DDM for a present satellite will present two correlation peaks at the same frequency but with different code phases. Since baseband generator 2 signal has been attenuated 3 dB, its peak amplitude will be smaller, thus making possible to identify to which baseband generator the signal belongs. Figure 7.22 shows a diagram of the SMU200A configuration.

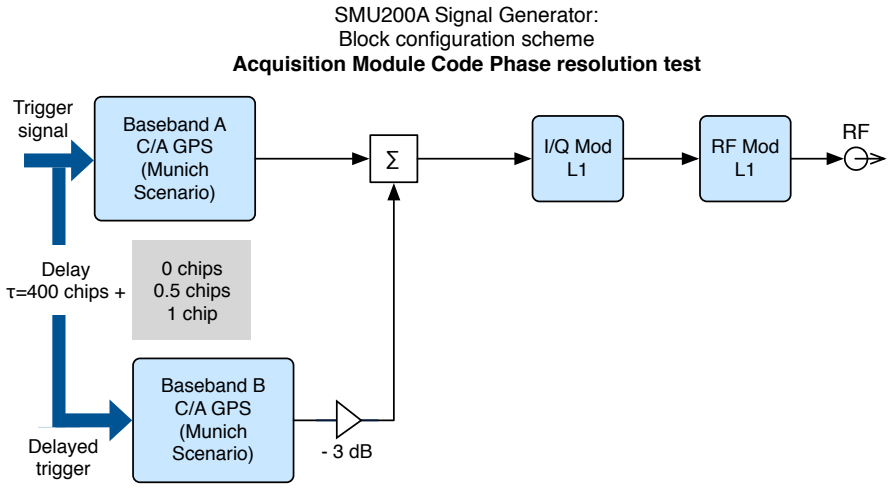


Figure 7.22: SMU200A configuration for measuring the acquisition module code phase resolution

2. To control the code phase difference between both baseband generators, the baseband

generator 2 was configure to be triggered by the first one, but with a constant delay. This was fixed to 400 chips to set a reference measure.

3. To measure the code phase resolution, the code phase difference due to the trigger delay is changed. Two different code phase difference where set in different tests: 1 chip (trigger signal delayed 401 chips), and half chip (trigger delayed 400.5 chips).

7.4.4.2 Results

In Figure 7.23, it is possible clearly see the two correlation peaks for SVN 5. The highest peak corresponds to baseband the generator 1 signal. Note that secondary correlation peaks also appear at different frequencies due to the sync form of the correlation function. At a code phase difference of 400 chips, a lower value correlation peak appears. This peak corresponds to the baseband processor 2. As it was expected, it is easy to determine which baseband generator produces each correlation peak due to the 3 dB power difference.

After confirming the presence of the two correlation peaks at the correct reference distance of 400 chips, this code phase difference was set to 401 chips.

Figure 7.24 shows the correlation result for frequency bin 24 (Doppler frequency shift$\simeq$ 1500 Hz) for both code phase difference cases: 400 and 401 chips. In order to graphically compare the correlation results obtained, the correlation result for the 401-chips code phase difference was circularly-shifted to set its maximum correlation peak at position 254. Thus, both correlation max peaks were aligned.

On the other side, Figure 7.25, shows that the second correlation peak obtained in 400 and 401 chips code phase difference is located at positions 645 and 655, respectively. Thus, this 1-chip code phase position difference proofs that the proposed acquisition module is able to distinguish between code phase differences of 1 chip successfully.

In the test performed to detect half chip code phase differences, the correlation results showed how the second correlation peak in some cases presented a code phase difference of 400 chips, while in other cases this difference was 401 chips. This perfectly makes sense. Since the sampling frequency is set to 1.024 MHz, the resolution is one 1 chip per sample. When trying to detect smaller code phase differences, the acquisition module presents a maximum code phase estimation error of 0.5 chips.

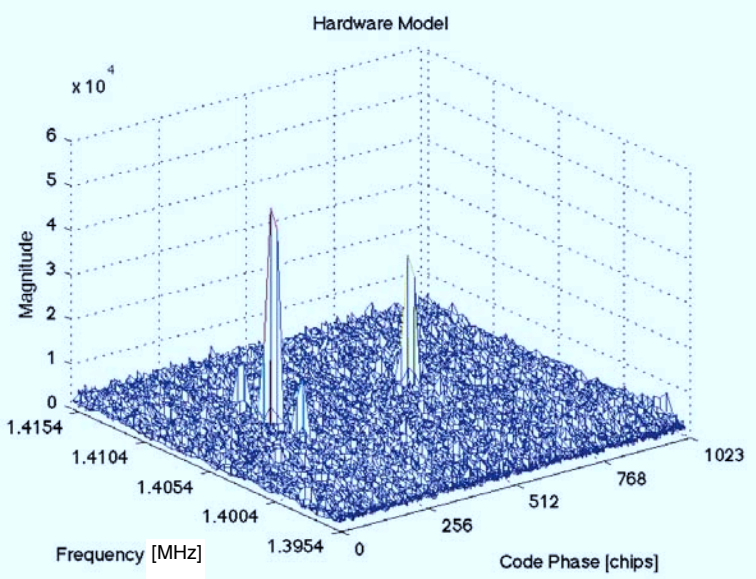


Figure 7.23: Code phase resolution reference test result for SVN 5 and a 400-chip delay

7.5 Chapter summary and conclusions

During this chapter the tests designed to verify the previous chapter's designs have been explained.

All the tests have been possible thanks to the R&D SMU200A vector signal generator. It made possible to settle highly configurational and flexible test frame. The signal generator includes some test scenarios that simulate specific locations at a certain time. The Munich's location simulation scenario has been common for all the tests performed.

The possibilities of the SMU200A perfectly matched with the Namuru's real-time IF raw data grabber (RTDG). This combination of a VHDL design and a PC capture software, has allowed store the SMU200A output, which made possible to test the parallel acquisition module design proposed. This real-time raw data grabber description has been presented within the experimental environment description.

After the experimental environment description, the tests to verify the Namuru-GPL modifications have been described. These tests were a challenge because of the SMU200A single RF output limitation. With only output containing both "direct" and "reflected" signals, and having 2 front-ends at the Namuru board, a workaround for testing was implemented. Opposite sign frequency shifts were added to the "direct" and "reflected" generated signals.

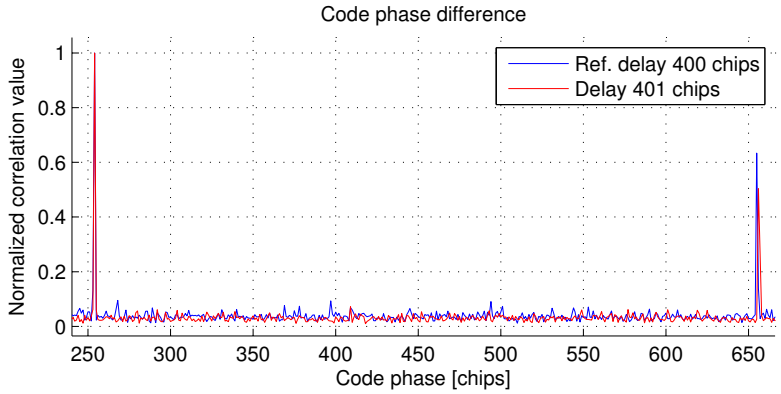


Figure 7.24: IF Frequency 1.4074 MHz (bin 24, Doppler shift $\simeq$ 1500 Hz) representation for a C/A code phase delay difference of 400 and 401 chips

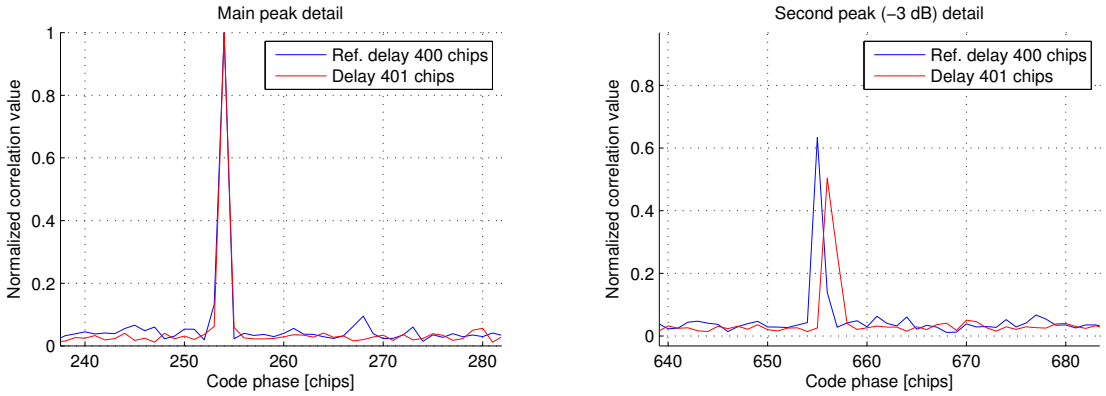


Figure 7.25: Detailed representation of the Figure 7.24 main and secondary peaks

In addition, selectively reducing the search acquisition space in the frequency domain made possible to distinguish between signal generation's origin at the front-end and thus it can search only for the desired signals.

Between "direct" and "reflected" signal a total of 5 code phase delay difference were set for measuring the obtained signal's pseudorange difference in each case. The results presented show how the introduced modifications really allows the Namuru-GPL receiver to acquire and track signals received using both board's front-ends simultaneously. Thus, the Modified Namuru-GPL could be use for applications that would benefit from having a single receiver able to acquire and track both signals avoiding the synchronization problems appeared in applications that use multiple receivers.

In addition, the modified Namuru-GPL has been able to measure pseudorange differences,

that in average allowed a precision up to 3 m, although with an error of 1 m in the worst case.

The rest of the chapter described the three main test performed to validate the parallel acquisition module described in the previous chapter.

The first three tests have proved that the acquisition module should be able to acquire signals with powers as low as -119 dBm at the receiver front-end input in little less than 80 ms. However, more tests should be done establishing different desired false alarm probability values. The sensitivity test results shows how the acquisition module is always able to determine the highest peak correlation position. The main problem is how to determinate if the result can be trusted depending on the acquisition indicator values.

The second acquisition module test, demonstrated that the acquisition module is able to estimate the signal's frequency Doppler with an error of 250 Hz in the worst case (frequency search bins of 500 Hz).

Thanks again to the SMU200A capabilities, in the third acquisition module test, the code phase resolution has been estimated to be 1 chip, by smartly comparing two correlation peaks' distances.

Watching at the results presented during this chapter, it is possible to say that the proposed implementations are able to work as they were expected, within a controlled simulated environment, and that the next should be field testing in real scenarios.

Chapter 8

Summary and conclusions

The goals of this project were: to evaluate the Namuru development board possibilities for GNSS-R applications and to develop a new Acquisition Module in order to reduce the required acquisition time. And these two goals have been successfully achieved. In the following, the results of this effort are summarized.

The first chapters are devoted to provide an overview on GNSS-R concept, the GPS C/A signal and the acquisition and tracking process performed by a GPS receiver. They review the main concepts necessary to provide a solid base in order to understand the proposed solutions.

The main features of the C/A GPS signal in the L1-band have been described and especial attention has been payed to the GPS receiver's acquisition and tracking processes.

By reviewing the main acquisition methods, serial search acquisition has shown to be the simplest alternative, however the most time consuming. This drawback can be an important limitation for environments in where a fast acquisition is a must. The alternative parallel code acquisition method has been described. With this method, the signal correlation is performed in the frequency domain, which is considerably more computationally efficient. In addition, working in the frequency domain allows to increase the acquisition speed as well as the acquisition sensibility, for example, by increasing the integration time.

After providing the necessary GPS background, the main tasks performed by a typical GPS receiver front end, and lately, the Namuru V2 GNSS development platform have been described in Chapter 4. The Namuru development platform can be divided in two parts: the hardware, basically the Namuru development board, and a set of software tools. It has

shown to be a low cost development platform specially interesting for research and development of GNSS receivers. The Namuru board most interesting feature is that it is composed by two RF independent front ends connected to an Altera FPGA with embedded memory and additional peripherals. Such feature provides a considerable design flexibility and the possibility of quickly implement applications that need 2 RF front-ends avoiding the typical synchronization problems that appear when multiple independent receivers are used.

The Namuru-GPL GPS software receiver has been described in Chapter 5. This open source GPS design is intended to run over the Namuru V2 GNSS development board. This single RF front-end design implements a baseband controller within the Namuru board FPGA, which is basically a correlator bank, and a Nios II soft-processor also embedded in the FPGA. Most of the main GPS receiver functions, such as the pseudorange measurement, the navigation data decoding and final computation of the position solution are performed by the software running on the Nios II processor.

The Namuru-GPL design has been used as starting point for evaluating the Namuru possibilities for GNSS-R applications. The Namuru-GPL has been modified to allow the receiver to use both front-ends to acquire and track the received signals simultaneously: the "direct" signal by the front-end 1, and the "reflected" signal by front-end 2. This modification has implied modifying the Namuru-GPL original baseband processor VHDL design, as well as introducing changes in the Namuru-GPL software to manage those changes. In addition, increasing the receiver's sensitivity was desired for tracking weak signals, and thus the thresholds used during the tracking routine were reviewed and reset to lower values.

To overcome the long acquisition time required by the Namuru-GPL acquisition serial search algorithm, a new Acquisiton Module was designed. This Acquisiton Module, extensively described in Chapter 6 and Appendix A and named Acq_Module_v1, implements aforementioned parallel code acquisition algorithm. The obtained Acquisition Module uses a resampling approach for reducing the original input data stream sampling frequency, $f_s = 5.714$ MHz, to $f_{rs} = 1.024$ MHz, in order to reduce the number of samples per code period. This has allowed to reduce the FFT module size, which is typically the most resource consuming part of the design to a 1024-point FFT. This resampling procedure uses the low-pass filtering and down-sampling approach proposed by Qaisar et. al. in [53]. The Acq_Module_v1 implementation required only 7,233 FPGA Logic Elements and 236,845 bits of on-chip memory, which represents a 25% and 46% of the free FPGA resources with the complete Namuru-GPL receiver already loaded.

Together with the Acquisition Module a MATLAB's simulation tool, referred as Acq_Simulator has been implemented to simulate verify the module behavior. This simulation tool has provided a reference framework to analyze the effect of the hardware limitations and quickly

introduce and test design changes without having to implement them in VHDL.

Finally, in Chapter 7, a description of the test set performed as well as the detail results obtained can be found. From these tests, it is possible to draw the following conclusions regarding the project's main objectives:

- The modified Namuru-GPL has been able to measure pseudorange differences, that in average (averaging a period of 150 samples with pseudorange measuring frequency) allowed a tested resolution up to 3 m in a controlled experimental scenario, generated by the SMU200A vector GPS signal generator. The difference measurement error observed when trying to measure 3 m psedorange difference was 1 m in the worst case for a satellite signal power of -116 dBm.

  The signal generator GPS scenario can't be considered as a realistic scenario, for example it has no multi-path propagation or realistic noise conditions. However, the tests show that the modified Namuru-GPL is ready for testing in real scenarios, what represents the following step to be performed in order to characterize its possible GNSS-R application capabilities.

- The proposed Acquisition Module behavior has been validated with three tests. The first test showed how the Acquisition Module has been able to acquire signals with powers as low as -119 dBm at the receiver front-end input, 1 ms of coherent integration time and 10 ms of incoherent integration time, in a total acquisition time of $\simeq 80$ ms for a frequency search band of ± 10 kHz. However, more tests should be done to characterize its false-alarm error probability with such powers and to provide acquisition indicators to determine the reliability of the acquisition results obtained.

  The second and third tests confirmed the expected Acquisition Module resolution in frequency and code phase. The Acquisition Module's error when estimating the signal's frequency Doppler shift is 250 Hz in worst case. Its maximum resolution in code phase is equal to 1 chip.

  Note that the Acquisition Module has been implemented as an IP core, however it has not been integrated yet within the Namuru-GPL receiver. The module's results were obtained by using hardware simulation tools (ModelSim). However, for the same test scenario, the Namuru-GPL serial acquisition method took an average 6 min to successfully acquire 4 satellite signals, while the Acquisition Module can search for all the possible GPS satellite signals in $80 \cdot 32 = 2.56$ s.

Considering the previous points, it is possible to affirm that the initial objectives has been fully accomplished. However, future works should test the modified Namuru-GPL capabilities

in real scenarios and integrate the proposed Acquisition Module within the modified Namuru-GPL design.

Just to finish, mention that the Acquisition Module design architecture could be used for other interesting GNSS-R application that require two signal correlation, with little effort and minor modifications.

Bibliography

[1] Martin-Neira, M., A Passive Reflectometry and Interferometry System (PARIS): Application to Ocean Altimetry, *ESA Journal, Vol. 17*, 1993. U.S. Patent 5 546 087, Aug. 13, 1996.

[2] Valencia, E., et al. Advanced Architectures for Real Time Delay-Doppler Map GNSS-Reflectometers: the GPS Reflectometer Instrument for PAU (griPAU), *Advances in Space Research, 2010, 46(2)*, 196-207.

[3] Zavorotny V.U.; Voronovich, A.G. Scattering of GPS Signals from the Ocean with Wind Remote Sensing Application. *IEEE Trans. Geosci. Remote. Sens. 2000*, 38, 951-964.

[4] Rius, A., Aparicio, J.M., Cardellach, E., Martin-Neira, M., Chapron, B. Sea surface state measured using GPS reflected signals. *Geophys. Res. Lett. 29 (23)*, 2122, 2002.

[5] Marchan; J.F.; Rodríguez, N.; Camps, A.; Bosch, X.; Ramos, I.; Valencia, E. Correction of the Sea State Impact in the L-Band Brightness Temperature by Means of Delay-Doppler Maps of Global Navigation Satellite Signals Reflected Over the Sea Surface. *IEEE Trans. Geosci. Remote. Sens.* 2008, 46, 2914-2923.

[6] Marchan, J.F.; Valencia, E.; Rodriguez-Alvarez, N.; Ramos-Perez, I.; Bosch-Lluis, X.; Camps, A.; Eugenio, F.; Marcello, J. Empirical sea state determination using GNSS-R data. *IEEE Geosci. Remote. Sens. Lett.* submitted for publication, 2009.

[7] Camps; A.; Bosch, X.; Ramos, I.; Marchán, J.F.; Izquierdo, B.; Rodríguez, N. New Instrument Concepts for Ocean Sensing:Analysis of the PAU-Radiometer. *IEEE Trans. Geosci. Remote. Sens. 2007*, 45, 3180-3192

[8] ESA's SMOS. Available online: http://www.esa.int/esaLP/LPsmos.html (accessed on November 12, 2009).

[9] X. Bosch-Lluis, I. Ramos-Perez, A. Camps, N. Rodriguez-Alvarez, E. Valencia, J.F. Marchan-Hernandez, Description and Performance of an L-Band Radiometer with Digital Beamforming, *Remote Sensing, Vol. 3*, pp. 14-40, 2011.

[10] Parkinson, B.; Spilker, J.; *Global Positioning System: Theory and Applications I/II/III*, 1996.

[11] Fisher, S.; Chassemi, K.; *GPS IIF - The Next Generation*. Proceedings of the IEEE, 87(1), 1999.

[12] Conley. R.; *GPS Operational Performance Assessment for 2006. Technical report*, Overlook Systems Technologies, Inc., 2007. http://gps.afspc.af.mil/gpsoc/ documents/2006_GPS_Performance_Report_V3.pdf.

[13] Department of Defense and NAVSTAR GPS, *Global Positioning System Standard Positioning Service Performance Standard, 4th edition*, 2008. http://gps.afspc.af.mil/gpsoc/documents/Signed_SPS_PS_(Sept_30_2008).pdf.

[14] Department of Defense and NAVSTAR GPS, *GPS Navstar interface control document*, 2007. http://www.navcen.uscg.gov/gps/ geninfo/

[15] Kaplan, E. D.; *Understanding GPS Principles and Applications, Second Edition*, Artech House, U.S., 2005.

[16] Borre, K.; Akos, D.; *A Software-Defined GPS and Galileo Receiver: Single-Frequency Approach*. Birkhäuser, Boston, 2006.

[17] Gold, R.; Optimal binary sequences for spread spectrum multiplexing, *IEEE Transactions on Information Theory*, 13(4):619–621, 1967.

[18] Tsui, J.B.; *Fundamentals of Global Positioning System Receivers: A Software Approach*. New York: John Wiley & Sons, 2000.

[19] Gleason, S.; Gebre-Egziabher, D.; *GNSS Applications and Methods*, Artech House, 2009.

[20] U. S. Department of Defense; S*PS Global positioning system standard positioning service signal specification*, 1995.

[21] Kilvington, J.; *Receivers for Navigation Satellite Systems*, U.S. Patent 4601005, July 1986.

[22] Van Nee, D; Coenen, A.; New Fast GPS Code-Acquisition Technique using FFT, *Electronics Letters, Vol.* 27, January 1991.

[23] Oppenheim, A.; Schäfer, R.; *Discrete-Time Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1999.

[24] Birklykke, A. *High Dynamic GPS Signal Acquisition : A Case Study in GPS Receivers on Nano-Satellites in LEO*, Master Thesis. Electronic and Information Technology Dept. Allborg Universitet, 2010.

[25] Winternitz, L.; Boegner, G.; Moreau, M.; Sirotzky, S.; Navigator GPS receiver for fast acquisition and weak signal space applications, *Proc. Inst. Navigation GNSS*, Sep. 2004.

[26] Ward, P.; Betz, J.; Hegarty, C.; *Understanding GPS Principles and Applications (Second Edition)*, E.D. Kaplan and C.J. Hegarty, Norwood, MA: Artech House, 2006.

[27] ICD-GPS-200. Interface control document. ICD-GPS-200, Arinc Research Corporation, Fountain Valley, 1991.

[28] Zarlink Semiconductor, *GP2015. GPS Receiver RF Front End Datasheet*, 2002.

[29] Dempster, A.; Parkinson, K.; Engel, F.; Mumfor P.; Rizos, C.; Heiser, G.; The UNSW/NICTA FPGA-based GPS Receiver: A Tool for GNSS Research, 2005.

[30] Dempster, A.; Parkinson, K.; Engel, F.; Mumfor P.; An Open GNSS Receiver Research Platform, *International Global Navigation Satellite Systems Society IGNSS Symposium,* 2006.

[31] Analog Devices, *ADF4360-7. Integrated Synthesizer and VCO Datasheet*, 2004.

[32] Samsung Electronics, *K4S461632E SDRAM module Datasheet*, 2004.

[33] Cypress Semiconductors , *CY62256V25 32K x 8 2.5 Static RAM Datasheet*, 2002.

[34] ST Semiconductors, *M25P64 64 Mbit, Low Voltage, Serial Flash Memory With 50MHz SPI Bus Interface Datasheet*, 2005.

[35] ST Semiconductors, *ST3232E ±15KV ESD-PROTECTED, 3 TO 5.5V, LOW POWER, UP TO 250KBPS, RS-232 DRIVERS AND RECEIVERS Datasheet*, 2003.

[36] Cypress Semiconductors, *CY7C68013A EZ-USB® FX2LP™ USB Microcontroller High-Speed USB Peripheral Controller Datasheet*, 2011.

[37] Microchip Technology Inc., *24AA128/24LC128/24FC128 128K I2CTM CMOS Serial EEPROM,* 2004.

[38] Dallas Semiconductors Inc. *DS1338 2-Wire RTC with 56-Byte NV RAM Datasheet*, 2003.

[39] Pedroni, V.A., *Circuit Design and Simulation with VHDL, Second Edition*, MIT Press, 2010.

[40] Kaur, G., *VHDL: Basics to programming*, Pearson, 2011.

[41] General Dynamics Corp. Ltd, *Namuru V2 GPS Receiver USER GUIDE*, 2008.

[42] Altera Corp., *Introduction to the Quartus® II Software,* 2010.

[43] Altera Corp., *Cyclone II Device Handbook*, 2008.

[44] Altera Corp., *Avalon Interface Specifications*, 2011.

[45] Altera Corp., *SOPC Builder User Guide*, 2010.

[46] Altera Corp., *Nios II Processor Reference Handbook*, 2011.

[47] Altera Corp., *Nios II Hardware Development Tutorial*, 2011.

[48] Mumford, P.; Parkinson, K.; Dempster, A.; *The Namuru Open GNSS Research Receiver.* University of New South Wales Australia, 2007.

[49] Massa, A.; *Embedded Software Development with eCos*, Prentice Hall, 2002.

[50] Greenberg, A.; *Open source software for commercial off-the-shelf GPS receivers*, Portland State University, 2005.

[51] Altera Corp., *FFT MegaCore Function User Guide v7.2*, 2008

[52] Gunawardena, S., *Feasibility Study for the Implementation of Global Positioning System Block Processing Techniques in Field Programmable Arrays*, Master Thesis. Ohio University, 2000.

[53] Qaisar, S. U., Shivaramaiah, N., Dempster A., Rizos, C., *Filtering IF Samples to Reduce the Computational Load of Frequency Domain Acquisition in GNSS Receivers.* University of New South Wales, Australia, 2008.

[54] Rohde & Schwarz Corp., *Digital Standards for R&S®SMU/SMATE/SMJ/SMBV/AMU/AFQ Datasheet,* 2011.

[55] Rohde & Schwarz Corp., *SMU200A Vector Signal Generator - Datasheet*, 2011.

[56] Mentor Graphics Corp. *ModelSim-Altera brochure*, US, 2010.

[57] Bitwise systems Inc., *QuickUSB Reference Guide v. 2.11.4,* US, 207.

[58] Mini-circuits Inc.; *Coaxial ZAPD-2-N+ Power Splitter/Combiner Datasheet*, US, 2010.

Appendix A

Acquisition Module main blocks characterization

This appendix describes the main blocks that have been implemented in the acquisition module. These main blocks are:

1. The C/A code generator
2. The FFT block

The additional implemented blocks, such as buffers and multiplexers, follow very standard implementations and can easily be understood just by reviewing the VHDL code. On the other hand, other system blocks such as the acquisition and downsampling block or the Control_Total module have been already described during Chapter 6 or its implementation is straightforward.

A.1 The C/A code generator block

This block function is to given a PRN number, from 1 to 32, generate the corresponding 1023 chip sequence sampled at $f_s = 5.714$ MHz resulting in a 5714 samples. Figure A.1 shows the block representation, while block's inputs and outputs are listed in table A.1.

The C/A code generator block is divided in two sub-blocks:

Table A.1: C/A Generator block inputs and outputs description

Signal Name	Direction	Size (bits)	Description
clk	In	1	Clock signal
rstn	In	1	Active-low asynchronous reset signal
PRN_generator_start	In	1	When a rising edge is detected, the C/A code generation starts
prn_number	In	8	The C/A code corresponding to this input prn_number when a rising edge at PRN_generator_start signal will be generated.
output_valid	In	10	Asserted when the C/A code generator is running and thus the output is valid.
sample_out	Out	10	Generated sample C/A code sample scaled by 127. Two possible values $\{-127, 127\}$. Signed representation.
new_sample_outCA	Out	1	Asserted each time that a new valid sample is ready.

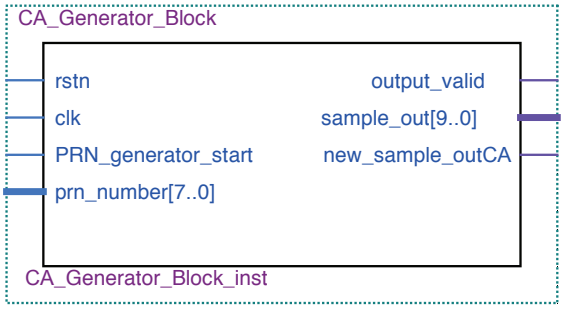


Figure A.1: C/A Generator block representation

1. The PRN generator

   The C/A code is generated according to the model presented in chapter 2. Following this model, this block implements two 10-bit linear feedback shift registers, G1Register and G2Register, were in both cases the input to first register's position is determined by the state of the other cells and the input PRN-number. This block output, called code_phase, is the XOR combination of the G1Register and G2Register.

   This block has two control signals. An enable signal and a "move on" signal. The enable signal determines when the block starts and ends its operation. On the other side, the "move on" signal is controlled by the Code NCO sub-block and allows to adjust the C/A code generation frequency. As it is implemented, this block only generates the next chip value when the "move on" signal is asserted.

2. Code NCO

   This module implements a 32-bit phase accumulator to generate a 1.023 MHz frequency pulse train sampled at $f_s = 5.714$ MHz. The phase increment to generate that frequency is 0x2DD52B47 (hex), computed as:

$$\phi_{inc} = \frac{f_{\text{code nco}} 2^{M \text{ bits}}}{f_s} = \frac{1.023\,[\text{MHz}]\ 2^{32}}{5.714\,[\text{MHz}]} = 0\text{x2DD52B47(hex)} \tag{A.1}$$

   The Code NCO's output, PRN_drive is connected to PRN generator's "move on" signal, thus controlling the frequency at which the C/A code is generated.

A.2 The FFT block

The FFT block has been implemented using the FFT MegaCore function. This is a high performance, highly-parametrizable Fast Fourier transform (FFT) processor. This block implements a complex FFT or inverse FFT (iFFT) for high-performance applications. The version used in this project is the v. 7.2.

The FFT MegaCore function has a fixed transform size architecture. This architecture implements a radix-2/4 decimation-in-frequency (DIF) FFT fixed-transform size algorithm for transform lengths of 2^m where $6 \leq m \leq 16$. This architecture also uses a block-floating point representations to achieve the best trade-off between maximum signal-to-noise ratio (SNR) and minimum size requirements [51].

The fixed transform architecture accepts as an input a two's complement format complex data vector of length N, where N is the desired transform length in natural order: the function

also outputs the transform-domain complex factor in natural order. An accumulated block exponent is output to indicate the corresponding data scaling. This data scaling during the transform allows to maintain precision and maximize the internal signal-to-noise ratio. Transform direction (FFT or iFFT) is specifiable via an input port.

A.2.1 DFT and FFT

The discrete Fourier Transform (DFT), of length N, calculates the sampled Fourier transform of a discrete-time sequence at N evenly distributed points $\omega k = 2\omega k/N$ on the unit circle [18, pp. 137-138].

$$X\left[k\right] = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nk/N}, \tag{A.2}$$

where $k = 0, 1, \ldots, N-1$, shows the length-N forward DFT of a sequence $x(n)$. The length-N inverse DFT is defined as:

$$X\left[k\right] = \frac{1}{N} \sum_{n=0}^{N-1} X\left[k\right]\, e^{-j2\pi nk/N}, \tag{A.3}$$

where $n = 0, 1, \ldots, N-1$.

The complexity of the DFT direct computation is significantly reduced by using the Cooley-Turkey radix-r decimation-in-frequency (DIF) FFT [51, p. 18], which recursively divides the input sequence into N/r sequences of length r and requires $log_r N$ stages of computation. Each of these decomposition stages typically shares the same hardware, with the data read from memory, passed through the FFT processor and written back to memory. Each pass through the FFT processor required to be performed $log_r N$ times. Increasing the radix of the decomposition results number of passes required through the FFT processor reduction. However, increasing the radix also supposes increase the device resources needed. The radix-4 is used in the Altera FFT Megacore function.

A.2.2 Altera FFT MegaCore main specs

The selected version of the FFT MegaCore function is specially suited for the Altera Cyclone II FPGA family. The selected transform length is 1024 points with a data precision (input and output) of 10 bits. The FFT twiddle precision is also 10 bits. Reference [52] performed

a extensive simulation study to determine the optimal bit-depths for implementation of GPS block processing (for FFT-based acquisition) methods in FPGAs. The study concludes that 10 bits is the optimal FFT bit-depth, thus, this data precision was selected.

The FFT MegaCore function allows to select its engine architecture. The architecture selected is the Quad-output FFT architecture with two parallel engines. This architecture is optimal for applications, such as this case, in which transform time is to be minimized. The term quad-output reforest to the throughput of the internal FFT butterfly processor. The engine implementation computes all four radix-4 butterfly complex outputs in a single clock cycle, which are re-ordered at the output. Check [51, pp. 50-65] for more details about the FFT engine architecture implementation.

Although it is possible to implement the FFT MegaCore function using only the FPGA logic elements (LEs), the chosen implementation also uses the DSP blocks embedded in the Cyclone II FPGA to perform the necessary complex products. The use of the DSP blocks allows resource use optimization. However, because the number of DSP blocks is limited, combined implementations are allowed. These implementations use only the available DSP blocks and the rest is implemented using LEs.

A.2.3 Control and integration: FFT_Module_sim_release

Table A.3 presents the FFT MegaCore function inputs and outputs. In order to hide multiple control signals and simplify the FFT MegaCore function control, the FFT has been enveloped in another VHDL block called **FFT_Module_sim_release**. Table A.2 shows its inputs and outputs and how they are used to control the FFT operation.

The selected FFT MegaCore function uses a burst data-flow architecture. In a burst I/O data flow architecture, the core can process a signal input block only. The burst architecture can load the rest of the subsequent FFT input data block only when the previous transform has been fully unloaded. Figure A.2 shows a simulation example to illustrate how the burst architecture works. The signals source_valid and sink_ready indicate when the FFT can accept a new block of data and when a valid output block is available on the FFT output.

A.2.4 Resources and performance

Table A.4 shows the total resources needed by the FFT_Module_sim_release VHDL block, which basically are the resources needed by the Altera FFT MegaCore function plus some LEs for its associated control logic. Same table also contains the number of clock cycles that

Table A.2: FFT_Model_sim_release input and output signals

Signal Name	Direction	Size (bits)	Description
clk	In	1	Clock signal that clocks both the FFT MegaCore function and its control associated logic
reset	In	1	Active-low asynchronous reset signal
start	In	1	When a rising edge is detected, the FFT conversion starts
FFT_inverse	In	1	Inverse FFT calculated if asserted
Data_in_imag	In	10	Imaginary input data, signed format
Data_in_real	In	10	Real input data, signed format
FFT_output_valid	Out	1	Asserted by the FFT when there is valid data to output
output_imag	Out	10	Imaginary output data, signed format
output_real	Out	10	Real output data, signed format
FFT_source_exp	Out	6	Signed block exponent for last frame conversion
FFT_error	Out	2	Indicates an error has occurred and its type

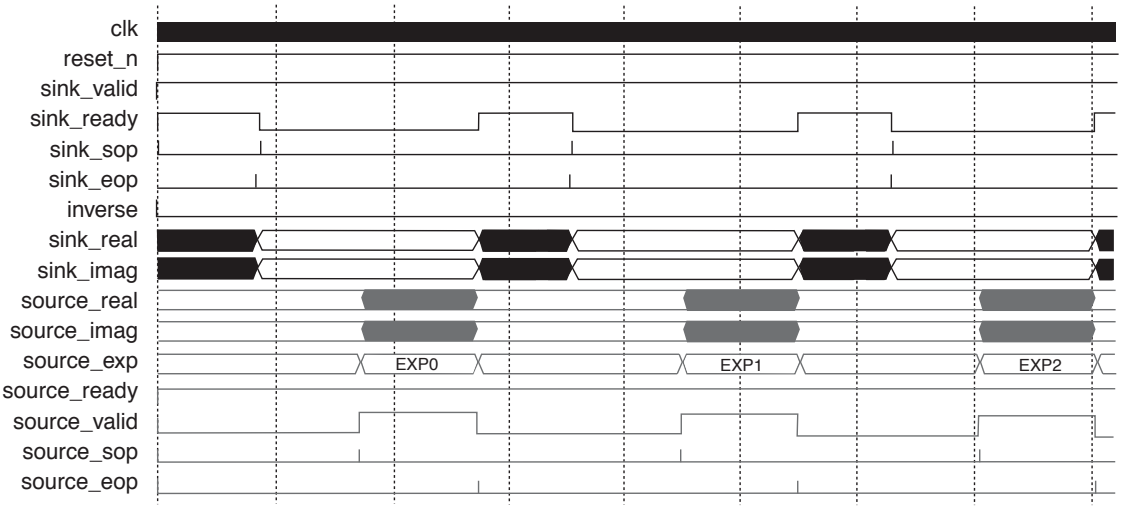


Figure A.2: FFT Burst data flow architecture simulation waveform

Table A.3: Altera FFT MegaCore function block input and output signals

Signal Name	**Direction**	**Size (bits)**	**Description**
clk	In	1	Clock signal that clocks all internal FFT engine components
reset_n	In	1	Active-low asynchronous reset signal
sink_eop (end of packet)	In	1	Indicates the end of the incoming FFT frame
sink_error	In	2	Indicates an error has occurred in an upstream module. In this case this signal is not used and is set to zero
sink_imag	In	10	Imaginary input data, which represents a signed number (two's complement) of 10 bits precision
sink_real	In	10	Real input data, which represents a signed number (two's complement) of 10 bits precision
sink_ready	Out	1	Asserted by the FFT engine when it can accept data
sink_sop (start of packet)	In	1	Indicates the start of the incoming FFT frame
sink_valid	In	1	Asserted when data on the bus is valid. When sink_valid and sink_valid are '1' the data transfer takes place
source_eop (end of packet)	Out	1	Marks the end of the outgoing FFT frame
source_error	Out	2	Indicates an error has occurred
source_exp	Out	6	Signed block exponent: Accounts for scaling of the internal signal values during the FFT computation
source_imag	Out	10	Imaginary output data, signed 10 bits precision
source_real	Out	10	Real output data, signed 10 bits precision
source_ready	In	1	Asserted by the downstream module if it is able to accept data. In this case is always asserted
source_sop (start of packet)	Out	1	Marks the start of the outgoing FFT frame
source_valid	Out	1	Asserted by the FFT when there is valid data to output
inverse	In	1	Inverse FFT calculated if asserted
clk_ena	In	1	Active-high global clock enable input

Resources and module performance	
LEs	4164
Memory bits	57344
M4K RAM blocks	14
DSP Blocks	48
Transform calculation cycles	601
Total block throughput cycles	2650

Table A.4: Resources used by the FFT_Module_sim_release VHDL block and its computational performance

the module needs to convert each input data frame. Both, the transform calculation cycles and the total block throughput cycles are depicted. As it is obvious, the total time needed to transform a complete 1024-sample frame will depend on the system's clock frequency. In this case, with a 40 MHz clock, the total block throughput will be 66.25 μs.

A.2.5 Simulation with MATLAB

Along with the FFT MegaCore function, Altera provides a bit-accurate MATLAB model:

FFT1024_10bits_model.m,

which was used to model the block's behavior using the MATLAB software. This function is specific for the selected implementation features. The model takes a complex vector as input and it outputs the transform domain vector and corresponding block exponent value. The length and direction of the transform (FFT/iFFT) are also passed as inputs to the model.

The input model vector length is an integral multiple of N, the transform length, the length of the output vector(s) is equal to the length of the input vector. However, if the input vector is not an integral multiple of N, it is zero-padded to extend the length to be so.