MICROWAVE-BASED NAVIGATION OF FEMTOSATELLITES USING ON-OFF KEYING

by

NAMRATA JAGDISH KAMTE

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTERS IN AEROSPACE ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2014

Copyright © by Namrata Jagdish Kamte 2014

All Rights Reserved

Dedicated to my parents.

Acknowledgements

I would like to thank Dr. Ben Harris for being a great advisor. In academic research, it is difficult to stay motivated at all times. You have lifted my spirits and restored my belief in myself at every turn. Thank you for believing in my capabilities and helping make this laborious journey enjoyable.

I would like to thank all my fellow researchers at the Satellite Technology Laboratory for their support. I would like to thank Rakesh Pandit for being there for me in my darkest times and for his help in improving my correlation plots. I would like to thank my four–legged companions, Sampson and Scotch for being by my side as I was burning the midnight oil, night after night and putting a smile on my face in the most unfortunate circumstances.

Most importantly, I would like thank my parents for making me a strong–willed and grounded human being. You are the sole reason I have the motivation to wake up every morning and achieve my goals. Thank you for giving me your unconditional support and pushing me to reach for the stars.

April 18, 2014

Abstract

MICROWAVE-BASED NAVIGATION OF FEMTOSATELLITES USING ON-OFF KEYING

Namrata Jagdish Kamte, M.S.

The University of Texas at Arlington, 2014

Supervising Professor: Dr. Robert Ben Harris

The objective of this research is to validate that a custom-built microchip-scale satellite transmitting a signal modulated with a Pseudo Random Noise code using On-Off Keying, can be tracked. The weak GPS satellite signal is modulated with a Pseudo Random Noise (PRN) code that provides a mathematical gain. Our signal is modulated with the same PRN code using On-Off Keying (OOK) unlike Phase Shift Keying used in GPS satellites. Our goal is to obtain timing and positioning information from the microchip-scale satellite via a ground station using the concepts of PRN encoding and the OOK modulation technique.

Decimeter scale satellites, with a mass of 2-6 kilograms, referred to as picosatellites, have been tracked successfully by ground stations. The microchip-scale satellite, called the femtosatellite is smaller with even less mass, at most 100 grams. At this size the satellite can take advantage of small-scale physics to perform maneuver, such as solar pressure, which only slightly perturb large spacecraft. Additionally, the reduced size decreases the cost of launch as compared to the picosatellites. A swarm of such femtostellites can serve as environmental probes, interplanetary chemists or in-orbit

inspectors of the parent spacecraft. In May 2011, NASA's last space shuttle mission STS-134 carried femtosatellites developed by Cornell researchers called "Sprites". The sprites were deployed from the International Space Station but ground stations on Earth failed to track them. In an effort to develop an alternative femtosatellite design, we have built our own femtosatellite prototype.

Our femtosatellite prototype contains the AVR microcontroller on an Arduino board. This assembly is connected to a radio transmitter and a custom antenna transmitting a 433 Mhz radio frequency signal. The prototype transmits a PRN code modulated onto the signal using OOK. Our ground station consists of a Universal Software Radio Peripheral (USRP) with a custom antenna for reception of the 433 MHz signal. The USRP is driven by an open source software-defined radio application called GNU Radio. The required components of the signal are extracted from GNU Radio and processed in order to plot the received data.

Benchtop testing of these OOK signals has yielded a reception sensitivity of upto 1 microsecond, which translates into a ranging capability similar to that of GPS satellites. We have correlated the received and replica PRN sequences and demonstrated that they match. The correlation can be used to obtain the identity and position of the femtosatellite prototype. This demonstrates the ability to track a femtosatellite signal that is lower than ambient noise, just like the signals broadcast from GPS satellites. Further, we have performed a system analysis and recognized key system behavioral problems. Thus we have determinately developed an optimum femtosatellite prototype and designed a novel positioning signal, providing a stepping-stone in the journey of successful femtosatellite communication.

Table of Contents

List of Illustrations

List of Tables

CHAPTER 1

Introduction

1.1 Background

1.1.1 Miniature Satellites

Most satellites in orbit have a mass of well over 1000 kg. For example the American navigation satellite called GPS IIF–4 launched in 2013 has a mass of 1630 kg. In stark contrast, the $50SAT is a satellite launched in 2013 that has a mass of less than 2 kg. There have been numerous efforts to minimize the size and mass of satellites. This has resulted in new classifications of miniature satellites shown in table 1.1

Table 1.1. Miniature Satellite Classification

Class	Mass Range
Microsatellite	1-100 kg
Nanosatellite	1-10 kg
Picosatellite	0.1-1 kg
Femtosatellite	10-100 g

The main advantage of miniaturization is decreasing the overall cost of satellite missions in order to make these missions accessible to the educational and commercial sectors. The availability of low cost micro-electronics and miniature electromechanical system (MEMS) sensor technology, with extremely low power consumption, has fueled the development of such miniature satellites [16]. This research focuses on femtosatellites with a mass of no more than 100 grams.

1.1.2 The CubeSat

The CubeSat is a picosatellite that can be built from commercial off-the-shelf components. The standard $10 \times 10 \times 10$ cm basic CubeSat is often called a one unit or 1U CubeSat and weighs no more than 1.33-kg (Figure 1.1). CubeSats are scalable along only one axis, by 1U increments. CubeSats such as a 2U CubeSat ($20 \times 10 \times 10$ cm) and a 3U CubeSat ($30 \times 10 \times 10$ cm) have been both built and launched.

Figure 1.1. The CalPoly 1 CubeSat.

CubeSats are always secondary payloads. Because they are standardized, they can be safely and inexpensively integrated into the faring of a larger rocket payload. The P–POD is a standardized deployment mechanism for the CubeSat, depicted in Figure 1.2.

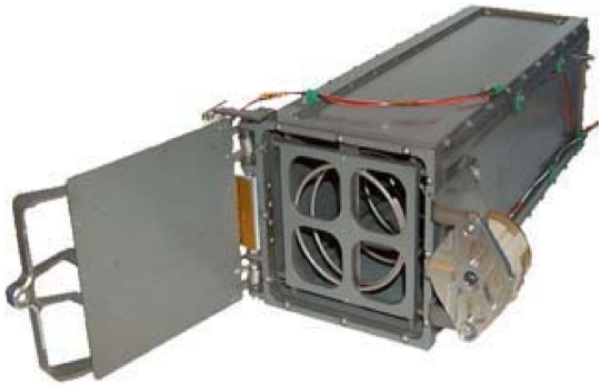


Figure 1.2. CubeSat Poly Orbital Deployer (P-POD).

The fundamental purpose of the CubeSat was to provide a low cost entry point into space for small research groups and universities. Unfortunately, the demand for CubeSat components and launches dramatically increased the cost of launch making difficult for highly cost constrained organizations. Furthermore, many of the efforts to develop a miniaturized payload for the CubeSat missions have not been successful [12]. Some spacecraft sub–systems such as propulsion have not been scaled for CubeSats as well.

Femtosatellites hold the promise to mitigate the problems associated with CubeSats. Swarms of femtosatellites can operate together to achieve goals that could be achieved by a single CubeSat or a standard satellite.. Additionally, femtosatellites significantly decrease the cost of the spacecraft production and assembly as they can be mass produced. Thus femtosatellites could restore research affordability and enable revolutionary mission capabilities [12].

1.2 The Femtosatellite

1.2.1 Definitions and Conceptual Design

A femtosatellite with a flat shape is referred to as a Satellite-on-a-chip or a Chip-Scale Spacecraft. The Satellite-on-a-chip is a completely functional satellite built as a monolithic integrated circuit that can be launched into space to perform a mission while communicating with a ground station [3].

Very small satellites are greatly disadvantaged by physical size limitations for payload accommodation and power generation. However, a detailed trade study including cost has proposed an acceptable size for the femtosatellite to be 10 cm x 10 cm x 2.5 cm and a mass of approximately 300 grams based on a stacked Printed Circuit board (PCB) fabrication [3]. This notional configuration can bee seen in Figure 1.3.

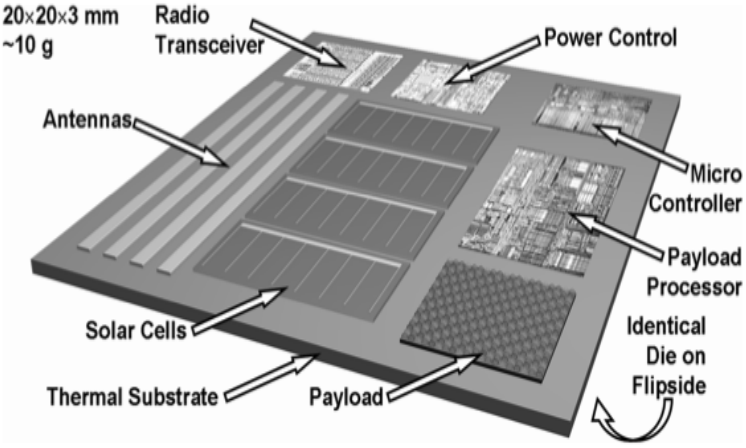


Figure 1.3. Notional femtosatellite configuration.

Due to their small size, a 'mother satellite' is typically required to deploy hundreds of femtosatellites into space. For example, a CubeSat could serve as a mother satellite.

1.2.2 Distributed Satellite Systems

A distributed satellite system is a system of multiple satellites designed to work in a coordinated fashion to perform a mission. One application of femtosatellite research is to enable distributed femtosatellite systems for real time, distributed, multi-point sensing to accomplish remote sensing and science objectives. These missions can be realized by merging the concepts of distributed satellite systems and terrestrial wireless sensor networks.

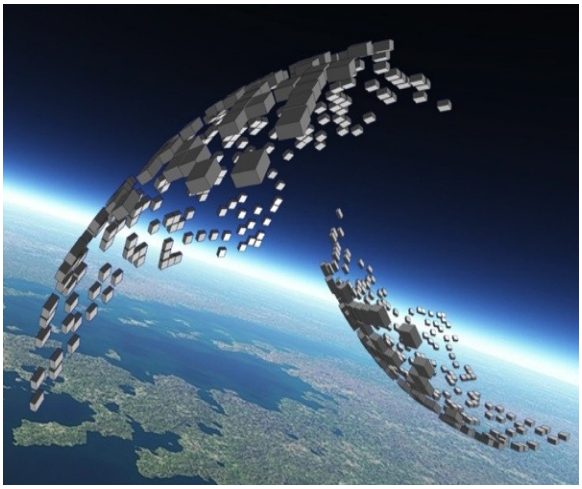


Figure 1.4. Distributed Femtosatellite System..

1.2.3 Advantages of Femtosatellites

1.2.3.1 Reduction in costs

The small size of femtosatellites makes them necessarily less expensive than larger satellites. In addition, the assembly, integration and test process costs are nearly eliminated as it is much cheaper to integrate microelectronics at such a small scale as compared to the larger satellites. Launch costs to Low Earth Orbit (LEO) are also reduced as one launch vehicle could deploy hundreds of these femtosatellites into

their respective orbits. Production schedules are reduced as the number of component interfaces has been reduced to a single level [3].

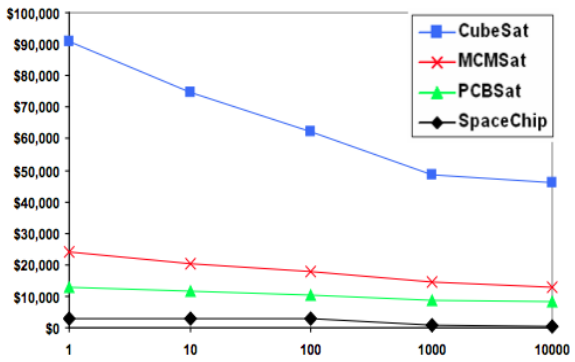


Figure 1.5. Unit Satellite Cost When Mass-Produced..

For femtosatellite missions, the mass production of satellites becomes the driving factor in determining cost. Figure 1.5 shows the cost comparison of four very small satellite technologies for a particular space missions discussed in [12]. The CubeSat is a 1U system and the other three are femtosatellite class systems based on different fabrication approaches. These numbers have been calculated taking the satellite die and its on–board sensors into consideration. It is important to note that the femtosatellite or SpaceChip results are based on a theoretical design unlike the other three systems.

1.2.3.2 Use of small scale physics to perform maneuver.

At the length scale of femtosatellites, their orbit is significantly affected by small scale physics that are typically perturb large spacecraft trajectories. These accelerations can compete with gravity thus providing new mission opportunities. A few examples of small scale physics are: solar radiation pressure that ejects dust

from the solar system; electromagnetic effects that capture and eject dust in planet centered orbits; and aerodynamic drag that captures and lands dust without the high temperature ablation that larger meteors typically experience. Hence femtosatellites can take advantage of these small body dynamics in order to enable new propulsion techniques and missions [4].

1.2.3.3 Potential Missions

Many potential femtosatellite missions have been proposed. Femtosatellites can provide telecommunication. They could also provide remote sensing capabilities. Finally, femtosatellites can monitor other spacecraft. Some examples of potential femtosatellite missions are described below.

In telecommunication, truly global coverage with no user antenna pointing requirements or significant signal delays can be realized with a large number of satellites operating in a LEO constellation. It has been thought that clusters of small satellites operating in LEO will eventually be used to implement "virtual" satellite missions replacing larger monolithic telecommunication satellites [3].

The Smart West Ford project was devised for immediate detection and retransmission of RF signals. With a few thousand femtosatellites distributed in a low orbit directly overhead , users simply tune in to the appropriate transmit and receive frequencies to establish communication. This gives a robust, beyond line of sight capability, potentially worldwide.

Femtosatellite constellations can form a complete space weather sensor network as they can be massively distributed in LEO. Thus they can help predict space weather which affect most of today's space-enabled technologies.

Femtosatellites can be used to measure the density of the upper atmosphere. In LEO, aerodynamic drag in the upper atmosphere is difficult to predict due to

the highly dynamic nature of atmospheric density. This affects the space situational awareness of satellites. Massively distributed femtosatellites in LEO would be a cost effective way of sampling the drag environment.

A large number of spatially separated femtosatellite detectors could improve the overall odds of detecting Terrestrial Gamma Ray Flash es (TGFs), thus improving our scientific understanding of how many TGFs occur per day and where, and provide additional information on energy and flux distributions.

In contrast to the dayside magnetosphere, compressed and confined by the solar wind, the nightside is stretched out into a long "magnetotail". This part of the magnetosphere is quite dynamic, large changes can take place there and ions and electrons are often energized. Femtosatellites can be used to reveal how the magnetotail behaves in response to solar wind variations and measure variations in magnetic fields around a spacecraft and [3].

Constellations of femtosatellites can be used for military and commercial imagery. These constellations can act as visible, UV or infrared imagers and provide images to provide weather forecasting data, agricultural data, disaster monitoring, detailed view of regions of strategic military interest amongst others.

Due to their tiny size and the resultant area to mass ratio, femtosatellites' mass will steadily decline, thus reducing the risk of of collision. With low mass satellite inspectors, more can be carried on any given spacecraft, thus justifying use of short lived disposable inspectors. Femtosatellites can thus be good one–way satellite inspectors.

1.2.4 Femtosatellite Challenges

1.2.4.1 Communication

There are two main communication challenges namely, low power operation and decaying orbits. Femtosatellites operate on extremely low power due the microelectronics used to build them and due to the fact that they use solar energy for power. A notional design proposed in [16] has the power requirement of 291 mWatt when active–all subsystems running, satellite transmitting data–calculated for a mass of about 95 grams). At such low power, communication with the femtosatellites is challenge. Some applications need high data rates which is not possible due to low power availability. However, most missions require a traditional direct downlink to one or more ground stations which can be made possible by selecting transmission frequency with a modest gain. Antenna size and data rates can also be an issue. The proposed missions have low data rates in the low kbps range. This allows the spacecraft transmit power to be low while leveraging a reasonable gain of the ground station antenna [12]. Femtosatellites deployed in LEO will have decaying orbits with a lifespan of a few weeks. This makes orbit prediction difficult and thus complicates the tracking process. Tracking techniques need further development and formalization.

1.2.4.2 Debris

Satellites in LEO are required to comply with the 25-year re-entry plan and this should not be an issue for femtosatellites as long as the starting altitude in below 700-km. Assuming that femtosatellites are deployed in a non-threatening orbit lower than that of the International Space Station (ISS) and other operational systems, their lifetime in LEO will be significantly low, not more than a few weeks. This will ensure quick disposal in LEO. Nonetheless, the femtosatellites can be perceived as

debris while they are in the operational phase of their mission. In that case, some guideline for such massively distributed satellites systems need to be established [12].

1.2.4.3 Design Restrictions

When projects become primarily focused on miniaturizing a payload, they typically fail for two reasons: either the costs become too high or the miniaturization objective cannot be reached. The laws of physics dictate a minimum solar power collection area or a minimum aperture size for the payload. Femtosatellites should be designed within the existing technology in order to be a cost–effective solution [12].

1.2.4.4 Formation Flight

Most of the proposed femtosatellite missions are based on employing a formation flying or swarm concept. Though this concept is popular, complications of formation flying and intra-satellite communication must be solved first [3].

1.2.4.5 Durability of Microelectronics

Microelectronics are prone to the adverse effects of heat and other harsh conditions typical of space weather. Effects of this harsh weather like thermal conditions need to be modeled into the femtosatellite design.

1.2.4.6 Power

The issue of power storage is yet to be resolved. Operating on solar power when exposed to sunlight is the simplest approach, followed by adding a battery [3].

1.2.5 Miniature Satellite Flight History

In 1958, The American Radio Relay League secured a launch opportunity on Discoverer XXXVI from Vandenberg AFB, California for the very first Amateur Radio satellite called OSCAR I. It was successfully launched into a low Earth orbit on the morning of December 12, 1961, barely four years after the launch of Sputnik I. It weighed 10 pounds and was literally built in the garages of the Project OSCAR team. OSCAR I lasted 22 days in orbit before burning up as it re-entered the atmosphere [2].

As a part of Project West Ford, 480 million, 1.78–cm long by 1.8–micron diameter copper wires (or needles) were flown in 1963. They were used to create an artificial ionosphere for the purpose of United States Military communications during the Cold War. The project was successful but was shelved due to debris issues and the development of modern communications satellites.

The Orbital Debris Radar Calibration Spheres-2 were launched in 1995. Multiple 0.5–5kg mass spheres were ejected from STS-63 along with two 13.3–cm long by 102–cm diameter dipoles (1.5 gram mass) and one 4.42-cm long by 102–cm diameter dipole (0.5-gram mass). These experiments were designed to provide small, LEO calibration targets for the ground-based radar and optical systems used for orbital debris measurements. The measurements taken and resulting data processing were a complete success.

The Aerospace PicoSat 1A/1B launched in 2000, was a tethered experiment launched from the Orbiting Picosatellite Automated Launcher (OPAL). They were 250-grams each in mass and 10 x 7.5 x 2.5-cm in size. Two were ejected from MightySat 2.1 a year later, a satellite flown by the Air Force Research Laboratory.

Hundreds of CubeSat missions have been launched since 2000. The first attempt to launch a CubeSat was aboard Russia's Eurorockot in 2003. This mission was unsuccessful. Since then there several successful CubeSat missions. One of them was

the (Space Test Program) STP–S26 launch with 7 satellites aboard, including RAX, a 3U Cubesat developed by University of Michigan. RAX has transmitted beacons that have been successfully decoded with the help of HAMs in Hawaii and Europe shortly after the STP-S26 launch in Kodiak, Alaska. RAX and OREOS have been heard in the US as well, including by Cal Poly.

In May 2011, three of Cornell Universities femtosatellites called 'Sprites' launched with NASA's Space Shuttle Endeavor on its final mission. They were attached to the ISS external platform Materials International Space Station Experiment (MISSE-8) for testing. Unfortunately, ground stations were not successful in tracking the Sprites due to reasons that are not known.

1.2.5.1 Upcoming Femtosatellite Mission (2014)

Cornell's Sprites are scheduled for a second launch in April 2014 (tentatively). 104 Sprites will be deployed from a mother CubeSat called KickSat into LEO. Once deployed, the Sprites' signals will be tracked by a worldwide network of amateur radio ground stations. The Sprites will be tested for their communication capabilities and the durability of microelectronics in the harsh space environment. Our research group at UTA, the Satellite Technology Laboratory will also be tracking the Sprites via our custom built ground station and receiver technology.

1.3 Scope of this Research

The objective of this research is to solve the femtosatellite tracking problem by building a custom femtosatellite prototype which will be tracked by a prototype ground station. Our goal is to obtain a modest reception sensitivity (1-10 μs) and obtain a correlation to confirm the identity of the prototype femtosatellite using

the concepts of PRN encoding and the OOK modulation technique. The goal of this research is to establish Code Division Multiple Access (CDMA) ranging and communication with a simple femtosatellite prototype. This prototype will serve as the foundation for the development of our own femtosatellite in the future. Hence, this research does not deal with the details of femtosatellite design pertaining to subsystems other than the communication subsystem.

The remainder of this thesis document is structured as follows: We will discuss the structure of our femtosatellite signal in Chapter 2. This will be followed by design considerations for our custom femtosatellite and ground station prototypes in Chapter 3. In Chapter 4, we will discuss the design of experiment and the signal processing technique used. In Chapter 5, we will describe in detail the results obtained. This will be followed by an analysis and discussion in Chapter 6. Finally, the conclusions and future work will be discussed in Chapter 7.

CHAPTER 2

Signal Structure

In this chapter,we establish a design for the signal transmitted by our femtosatellite prototype. As we aim at detecting a weakly transmitted signal amidst a large amount of noise, we use the Gold codes to modulate the signal just like the GPS satellites do.

Code division multiple access (CDMA) is a channel access method used by various radio communication technologies. CDMA is an example of multiple access, which is where several transmitters can send information simultaneously over a single communication channel. This allows several users to share a band of frequencies. To permit this to be achieved without undue interference between the users, CDMA employs spread–spectrum technology and a special coding scheme (where each transmitter is assigned a code). GPS satellites use a CDMA scheme called the Gold Code.

Gold codes have bounded small cross-correlations within a set, which is useful when multiple devices are broadcasting in the same frequency range. The Pseudo Random Noise (PRN) sequence is a binary sequence generated by Gold codes. In addition to enabling CDMA, the PRN sequence provides a mathematical gain to the signal. These sequences allow precise range measurements and mitigate the deleterious effects of reflected and interfering signals received by a GPS antenna.

In this chapter, we will first briefly study the PRN sequence generation technique used by GPS satellites and apply it to our prototype signal design. Secondly, we will discuss our selection of the technique used to modulate the PRN sequence onto the signal including a study of the modulation techniques used by GPS and Cornell's

Sprites. Thirdly, we will choose an RF band on which our prototype will transmit data. Finally, we will establish a signal design for our prototype which includes the encoding scheme, modulation technique and transmitting frequency.

2.1 GPS Signal Structure

The GPS signal has three parts: the carrier signal, the ranging code and the navigation data. The ranging code and the navigation data are modulated onto the carrier signal using the Binary Phase Shift Keying (BPSK) modulation technique. We will only go into details about the ranging code as it is the same PRN sequence that we are will use to encode our femtosatellite signal ([17]).

Each GPS satellite transmits a unique PRN sequence. Each PRN sequence is a unique set of 1023 bits, called *chips* which is cycled every millisecond. The duration of each PRN sequence chip is about 1 μs. Equivalently the chip width or wavelength is 300 m. The rate of the C/A code chips called *chipping rate* is 1.023 Mhz or million chips per second (Mcps). The length of a PRN sequence and its chipping rate determine the acquisition and tracking efficiency of the signal in the presence of noise and interference.

2.2 PRN Sequence Generation

The method used to generate a PRN sequence is discussed below. *Our prototype satellite uses this method to generate a PRN sequence.* A PRN sequence is a series of binary chips shown in Figure(2.1) below ([17]).

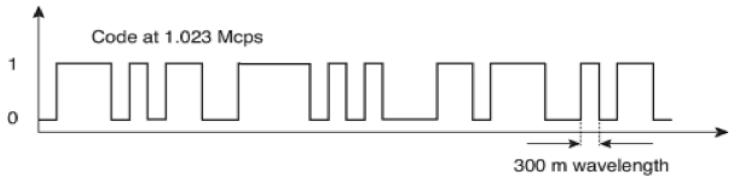


Figure 2.1. The PRN code chips.

The PRN sequence is generated with bit shift registers. Bit shift registers are arrays of bits (1s and 0s) of specified length. At each clock cycle, all of the bits are moved one space to the right in the array. The rightmost bit is the output and a new leftmost bit is generated by an operation on the bits from the previous clock cycle. The following is an illustration of a 5-bit long shift register in action:

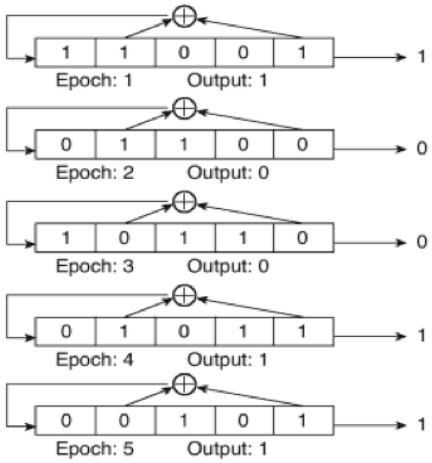


Figure 2.2. Bit Shift Register.

Thus the output sequence from this register, given that this register was initialized with 11001, is [1,0,0,1,1,.....]. Note that the sequence can go on forever, but eventually the output will repeat. The $\oplus$ symbol indicates modulo 2 addition:

$0 \oplus 0 = 0; 0 \oplus 1 = 1; 1 \oplus 0 = 1; 1 \oplus 1 = 0;$. You can do the same with three or more bits.

The satellites are identified by their unique PRN sequence number. Thus 'PRN k' indicates both the PRN code and the satellite number 'k' transmitting it. *In our experiment, the femtosatellite transmits PRN 19 which is simply used to demonstrate the transmission and reception of a widely known PRN code. In future, a custom PRN code can be designed for the femtosatellite prototype.* To create PRN codes, two 10-bit shift registers are used as shown below:

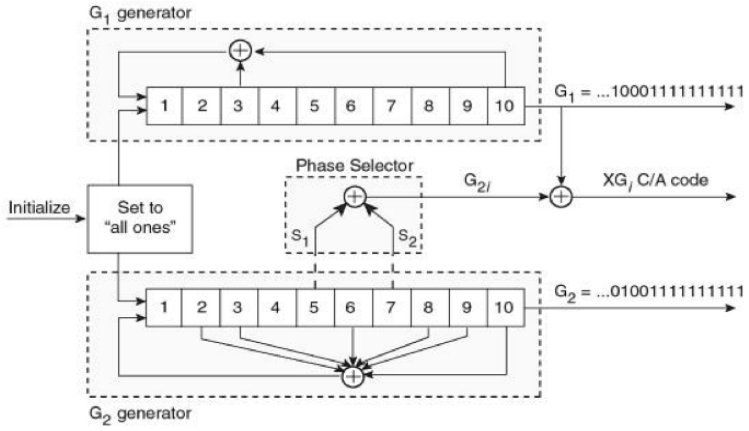


Figure 2.3. PRN Generation.

Note the leftmost bit of the G_1 shift register is generated by performing a modulo 2 addition on bits 3 and 10. Similarly the leftmost bit of the G_2 shift register is generated by performing a modulo 2 addition on bits 2, 3, 6, 8, 9 and 10. Also, note the phase selector in the middle of the Figure(2.3). S_1 and S_2 indicate which bits of the G_2 shift register are added to create the G_{2i} output at each clock cycle.S_1 and S_2 are different for different satellites. For example, PRN 19 is formed by adding bits 3 and 6 of the G_2 shift register to form the G_{2i} bit.

2.3 Modulation Techniques

2.3.1 Binary Phase Shift Keying (BPSK)

As mentioned earlier, GPS satellites use the BPSK technique to modulate the carrier sinusoidal. BPSK can be seen as a form of Amplitude modulation. A 0 bit leaves the carrier unchanged; and a 1 bit multiplies the carrier by -1, which is equivalent to shifting the phase of the sinusoidal by 180 degrees. At bit transitions from 0 to 1, or from 1 to 0, the phase of the carrier is shifted by 180 degrees as shown in Figure (2.4).

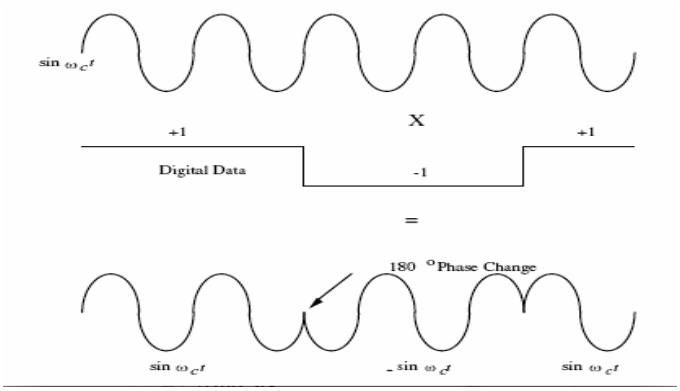


Figure 2.4. Binary Phase Shift Keying (BPSK).

2.3.2 Minimum Shift Keying (MSK)

Cornell University's Sprites use MSK. MSK is a type of continuous phase frequency-shift keying. As the phase transitions are smoothed out, out-of-band interference due to band limiting and amplifier nonlinearity is reduced [25], Figure 2.5.

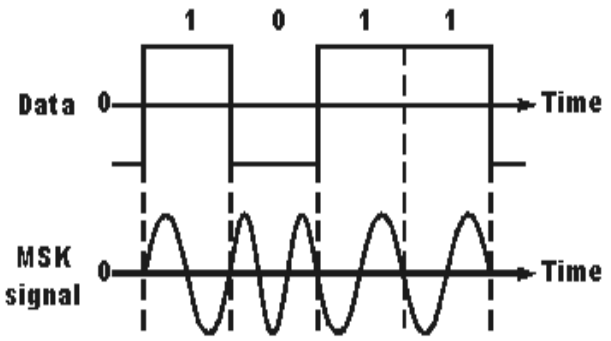


Figure 2.5. Minimum Shift Keying.

2.3.3 On-Off Keying (OOK) :

Amplitude Shift Keying (ASK) is a form of amplitude modulation that represents data as variations of amplitude of a carrier wave. OOK is the binary form of ASK. It represents a series of logical 1s and 0s by simply switching on and off the carrier signal. The representation of OOK is shown in the Figure (2.6). A '0' is represented by having the carrier 'off' (i.e. reducing the amplitude to zero) and a '1' by having the carrier 'on' (i.e. giving it a chosen amplitude A).

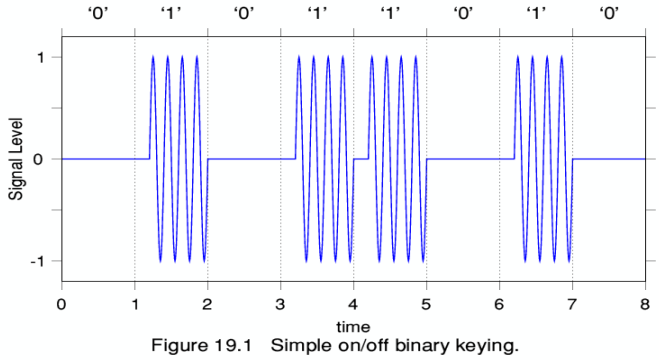


Figure 2.6. On-Off Keying (OOK).

The main disadvantage of OOK is that it does perform well in the presence of undesired signals. As the complexity of the control and data communication apparatus increases, the transmission efficiency decreases because the each additional component factors into the power consumption and could contribute to the presence of undesired signals that interfere with OOK. Nonetheless, we choose OOK as it has several benefits that make it a good fit into our application.

The main advantage of OOK is the simplicity of physical implementation, as one pulse generator is required as opposed to two, as in the case of biphase modulation. It helps create a simple transmitter configuration as a single switch can control the transmitted pulses. Another advantage of OOK is the extremely low cost of implementation. Other than the switch that drives the output high and low, no other external components are necessary. OOK modulation has the advantage of allowing the transmitter to idle during the transmission of a 'zero', thus conserving power and reducing heat generation. As already discussed, one of the main challenges for femtosatellites is the low power that they operate on. Also if we use the configuration that depends on solar energy for power generation, power conservation is of prime importance. OOK is suitable for low data transmission rates which works in our favor as the size of femtosatellites make it difficult to transmit at high data rates. The requirement of estimating athe carrier phase makes coherent (use the knowledge of the phase of the carrier frequency) demodulation of MSK signals complex. OOK has a simpler detection process as the demodulation is non-coherent.

We have selected OOK to be the modulation scheme for our foundational femtosatellite design. Our goal is demonstrate successful communication between our prototype and the ground station using OOK.

2.4 Signal Frequency – 433 MHz

In the process of obtaining a certification in amateur radio operation we have studied the advantages and disadvantages of different radio frequency bands. For example, the 144 MHz band has a very large wavelength and the GHz range bands have rapid spatial attenuation. Our femtosatellite prototype transmits at the 433MHz frequency. The 433 Mhz wave is a *microwave* which is a term applied to radio waves between the frequencies 300 Mhz and 300 Ghz. This range is an Industrial, Scientific and Medical (ISM) radio band. The 432-438 Mhz range is a popular amateur radio satellite band. This is particularly helpful as the goal is that amateur radio ground stations all over the world should be able to track the femtosatellites. 433 Mhz frequency can be produced by a small (size of postage stamp or even smaller) simple chip that has no complicated circuit unlike circuits required for higher ranges. As the 433 Mhz frequency range is used widely for wireless transmission applications, there are numerous different established transmitter circuits that can bought commercially, off the shelf. Most of these 433 Mhz transmitters seamlessly integrate with a variety of microcontrollers.

Thus we have defined the signal structure for our femtosatellite prototype. It has a center frequency of 433 MHz. This 433 MHz carrier signal is modulated with the PRN 19 sequence using the On-Off Keying technique.

CHAPTER 3

Hardware Design and Selection

This chapter has two sections. The first section discusses the transmitter i.e. our femtosatellite prototype design the second discusses the receiver i.e. the ground station. We have performed a thorough study of the hardware components that have been selected for our prototype. We will discuss the details of this study including design considerations and the reasoning behind the selection of specific components for the femtosatellite prototype and the ground station.

3.1 Femtosatellite Prototype Design

The basic components of our femtosatellite are a microcontroller and an RF transmitter that can be bought commercially off-the-shelf. The results of my study of the different configurations in the market follow.

3.1.1 Microcontroller Selection

Our goal is to find a microcontroller that is cheap, readily available off the shelf, works on open source software-compilers and has a good development board. We are opting for a pre–assembled development board for two reasons: The first reason is so we do not have to endure the cumbersome and time consuming process of directly programming every bit of every register on the microcontroller. The second reason is so we do not have to design a femtosatellite printed circuit board (PCB) from scratch with every component as that is a different research problem of detailed system design which is not our focus at the moment. Both the direct microcontroller programming

and detailed femtosatellite design will be implemented in the near future but they are not suited for our current application as our main goal is to tackle the issues of the communication subsystem of the femtosatellite.

Three microcontroller families are most widely used for wireless RF applications, namely Microchip Technology's PICs, Texas Instruments' MSP430s and ATMEL's AVRs. A comparison study between the three families helped make a microcontroller selection.

3.1.1.1 PIC

Microchip's PICs are the oldest family of microcontrollers with many improvements over time. It has the advantage of using a RISC (Reduced Instruction Set Code) design so there are less instructions to remember. RISC is code is efficient, allowing the PIC to run with typically less program memory than its larger competitors. It is cheap and has a high clock speed. On the flip side, you cannot program in real C language as in you cannot use the standard C libraries. Moreover, the PIC's beginner development board called "BASIC Stamp" is not programmable in C (it uses BASIC) and is more expensive than the AVR's development board. All in all, the PIC is by no means an easy platform to work on. It needs a high level of microcontroller programming expertise and even then it is troublesome to work with. Hence, at this point, we have eliminated the PIC as a contender for our prototype's microcontroller. So it now boils down to a comparison analysis between the AVRs and the MSP430s which is discussed in section 5.22.

3.1.1.2 MSP430 and AVR

We will first discuss the AVR followed by the MSP430. With ease-of-use, low power consumption, and high level of integration, Atmel AVR 8- and 32-bit micro-

controllers deliver a unique combination of performance, power efficiency and design flexibility. They are based on the industry's most code-efficient architecture for C and assembly programming. The AVRs can be used with a development board called the Arduino which is an open source electronics prototyping platform based on flexible hardware and software.

Texas Instruments released their flash based, 16-bit microcontroller family called the MSP430. TI also recently released development boards for the MSP430s called the Launchpad.

There are several grounds upon which we can compare the AVR and the MSP430. The main advantage of the MSP430 is that it has the lowest power consumption amongst all microcontroller families. The electric current drawn is less than 1 μA and can be throttled back for even lower power consumption. This feature is very advantageous to femtosatellites that have the fundamental challenge of having to function on extremely low power. This is probably what prompted the Cornell University group to use the MSP430 as the brain of the Sprite. However, just like the PIC, the MSP430 is difficult to program directly. As a result, the Cornell group ported an Arduino development environment to the MSP430 interface for the Sprites which lead to issues as a result of transporting code across different platforms. Cornell's group used the Arduino because the MSP430's development boards have very specialized compilers. They are not as user friendly and flexible as the AVR's development boards. The table below lists some of the advantages of the Atmel AVR over the MSP430:

FEATURE	AVR SOLUTION	MSP430 SOLUTION
Code Size	Architecture optimized for High Level Languages gives **20% smaller code** than MSP430.	Disadvantage in code-size due to 16-bit architecture and less rich instruction set.
Performance	MegaAVR has maximum frequency of **16 MHz**. The extremely good code-density contributes further to the very high performance.	8 MIPS performance. 50% of what the AVR can offer.
Voltage Range	AVR is manufactured in a dense process, but still allowing **2.7-5.5V** operation.	No 5-volt support!
Self Programming	**Full flexibility** in boot-block size. Can use any interface to do self-programming at any speed. Programming 64K of memory can be done in **less than 10 seconds**.	Boot-block algorithm fixed. Needs to use UART running at 9600 Baud giving very slow programming. Needs to toggle external pin to enter programming mode. Programming 64K of memory takes more than 60 seconds.
Read While Write	AVR supports executing interrupts and critical functions while doing memory upgrades.	Interrupts and critical functions are halted while doing memory upgrades.
Segment Size	Memory segments are 128 Bytes giving the ideal trade-off between programming time and flexibility.	Memory segments are typically 512 bytes, giving much longer programming time for updating a small part of the memory.
EEPROM	AVR has internal EEPROM **saving the cost** of an external one, and increasing the **security** for the data.	No internal EEPROM making it necessary to use an external occupying pins and stealing power.
BOD	Internal programmable Brownout protection and power on reset.	**No internal Brownout protection.** Their BOD is similar to the Power On Reset of the AVR. Implementing external BOD adds a $0.30-$1 cost.
Peripheral Mix	8-pin 1K parts with full featured A/D, UART, SPI and Timers.	20/28-pin parts does not have hardware ADC and UART/SPI.
Field Debugging	The megaAVR JTAG interface can be used to debug products installed in the field to enable finding problems caused by the environment.	Field debugging using JTAG is not possible since JTAG disable fuse is irreversible.

Figure 3.1. Advantages of Atmel AVR over TI MSP430.

The MSP430 family has performance limitations and is about 25%-30% slower than the Mega AVR Family. The MSP430 is not code efficient and its application code size is typically 20% larger than code written for the AVR. The MSP430 has limited integration and is missing key peripherals like EEPROM, external SRAM interface and Brown-Out-Protection. Adding this capability externally increases the system cost. The MSP430 offers limited self-programming capability. Their approach is crude compared to the AVR implementation. Moreover, the compiler for MSP430,

Code Composer Studio (CCS) is not free (very expensive, about 900 USD) and it struggles with standard C libraries.

Finally, we picked the AVR microcontroller for our femtosatellite for a variety of reasons. AVRs are known for ease of programmability and the ability to integrate with a wide range of peripheral devices seamlessly. They have been used extensively for the purpose of RF communication thus providing an extremely strong and well-supported platform to build upon. Due to their popularity and availability, they can seamlessly integrated with a large number of commercially available peripherals which is useful in our case as we need to integrate our microcontroller with an RF transmitter peripheral and it's antenna. AVRs use avr-gcc, a targeted version of gcc (GNU Compiler Collections–A compiler system that supports various languages like Objective C++ and Java) for the AVR microcontroller. One, this compiler is completely free and works for Macs, Linux, Windows in a nice distribution. Second it is 'real' C, which is great when you want things like standard libs, includes, linking, unions, structs, and pointers (CCS at least, struggled with these). It also means that porting code is really easy because everyone uses it (all of PIC compilers are not 100% compatible). Third it generates good (fast, small and correct) code: the optimizer is the same as in every other gcc. They can be programmed using the Arduino development boards which are open-source, inexpensive and extremely flexible. They have a wide variety of fully functional libraries for different purposes. Arduino boards are designed around AVRs, which allows the use of the Arduino programming language further simplifying the process of programming the microcontroller and eliminating issues typical of porting code across platforms.

After reviewing the design considerations listed above, we have chosen the Arduino development board with the AVR microcontroller for our femtosatellite prototype.

3.1.2 Selection of AVR-Arduino Board combination

There several types of Arduino development boards and AVRs in the market. For our experiment, we have selected the Arduino *UNO* development board and the *ATMega328* AVR. In this subsection, we will discuss the features of the Arduino UNO and The ATMega328 AVR.

The ATmega series microcontrollers offer substantial program and data memories with performance up to 20 MIPS (Million Instructions Per Second). It has picoPower technology minimizes power consumption. All megaAVRs offer self-programmability for fast, secure, cost-effective in-circuit upgrades. You can even upgrade the flash while running your application (Figure 3.2).

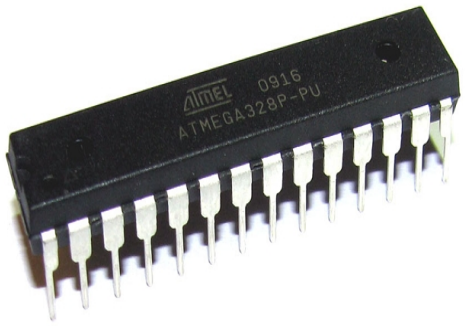
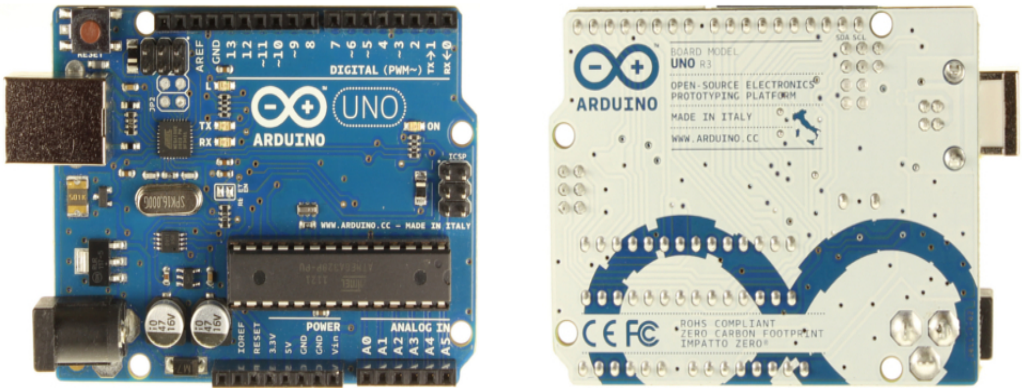


Figure 3.2. The ATmega328 AVR microcontroller.



(a) Arduino UNO R3 Front (b) Arduino UNO R3 Back

Figure 3.3. The Arduino UNO with the ATmega328 AVR.

The Arduino Uno is a microcontroller board based on the ATmega328 (Appendix datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM–Pulse Width Modulation–outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, and a reset button. It contains everything needed to support the microcontroller. We can simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to program it. The length and width of the Uno are 6.858 and 5.334 centimeters respectively, with the USB connector and power jack extending beyond the former dimension. The Arduino Uno can be powered via the USB connection or with an external power supply. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM. Each of the 14 digital pins on the Uno can be used as an input or output.They operate at 5 volts. Each pin can

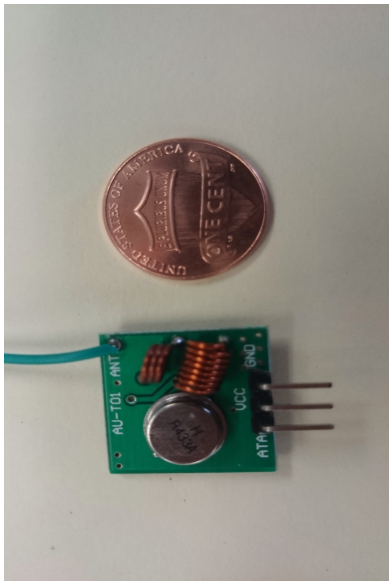
provide or receive a maximum of 40 mA. The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides 5 Volt UART (Universal Asynchronous Receiver/Transmitter) serial communication, which is available on digital pins 0 (RX) and 1 (TX). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). The Arduino Uno can be programmed with the open-source Arduino software. The ATmega328 on the Arduino Uno comes pre–burned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. The table below summarizes the Arduino Uno features:

Table 3.1. Summary of Arduino Uno's features

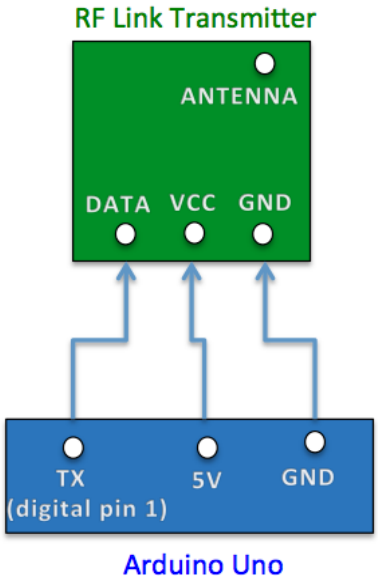
Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

3.1.3 RF Transmitter

After a thorough search for the right RF transmitter for our femtosatellite prototype, we have selected the RF Link Kit transmitter. The transmitter emits on the 433 MHz frequency, which is the frequency we have selected in section 2.4. This transmitter is extremely small in size and mass (size of a penny) which perfectly supports our need to minimize the size and mass of the entire femtosatellite prototype. It is inexpensive - costs not more than 3 USD. This particular rf transmitter has been extensively used with AVRs and Arduinos for RF applications. Thus we know that it seamlessly integrates with our Arduino Uno-AVR combination. The RF Link transmitter and the schematic for connecting the transmitter with the Arduino UNO are shown in Figure (3.1.3).



(a) RF Link Transmitter



(b) Transmitter-Uno Connection Schematic

The detailed specifications of the RF Link transmitter are:

Frequency Range: 433.92 MHz.

Launch distance: 20-200 meters (different voltage, different results).

Operating voltage: 3.5-12V.

Dimensions: 19×19 mm.

Modulation: ASK,OOK.

Transfer rate: 4KB/S.

Transmitting power: 10 mW.

Current Consumption: 8 mA.

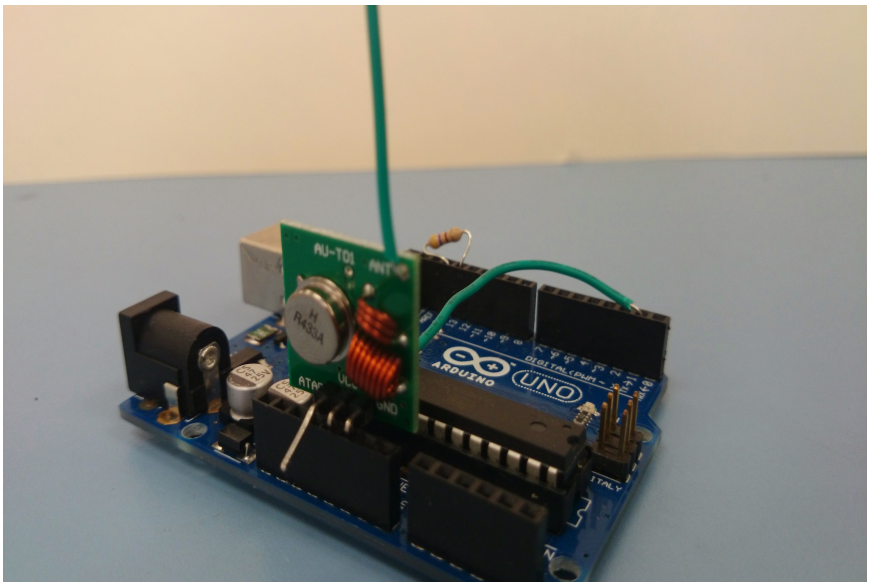
Pinout from left $\rightarrow$ right: (DATA; VCC; GND) and a point to connect the antenna.

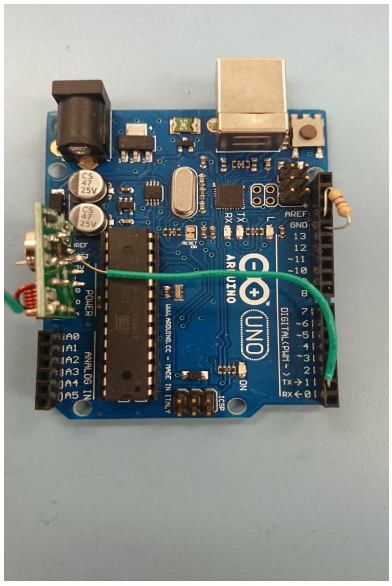
3.1.3.1 The Custom RF transmitter Antenna :

We use a regular 22 gauge wire as our antenna. The length of the antenna is quarter of the signal wavelength. It was calculated as follows:
Period of the wave $= 1/(433.92\,\text{MHz}) = 2.3\,\text{ns}$. Speed of wave $= 3 \times 10^8\,\text{m/s}$. meter to inch conversion $= 3.28\,\text{ft/m} \times 12\,\text{in/ft} = 39.36\,\text{in/m}$. Quarter Wavelength in Inches $= (3 \times 10^8 \times 2.3\text{ns} \times 39.36)/4 = 6.7896$ (approx. 6.8inches).

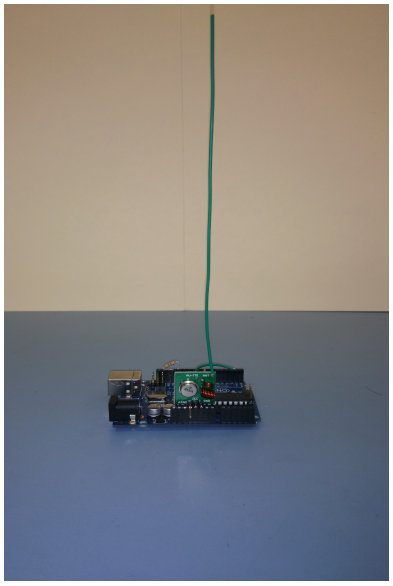
3.1.4 Our Femtosatellite Prototype

Our femtosatellite prototype consists of the ATmega328 AVR microcontroller embedded onto an Arduino UNO development board. This assembly is connected to an RF Link Transmitter using the schematic shown in figure(3.4(b)). The RF transmitter emits on the 433MHz frequency via a custom 6.8 inch wire antenna. The entire assembled femtosatellite prototype can be seen in Figure 3.4.





(c)



(d)

Figure 3.4. Final Design of the Femtosatellite Prototype.

3.2 The Ground Station Prototype

Our ground station prototype consists of an RF front-end device called the Universal Software Radio Peripheral (USRP) driven by a software radio application called GNU Radio. In this section we will briefly study I/Q sampling used by the USRP and then discuss the features and operating principles of the USRP and GNU Radio.

3.2.1 In–Phase (I) and Quadrature–Phase (Q) Sampling Basics

In real or Nyquist sampling the actual signal is divided into a discrete sequence of samples that follow the Nyquist sampling theorem which states that the sampling rate should be greater than twice the bandwidth of the signal.

In contrast, in In–phase and Quadrature–phase (I/Q) sampling we are sampling the actual signal twice at every discrete point. I/Q sampling shows the changes in magnitude (or amplitude) and phase of a sine wave. If amplitude and phase changes occur in an orderly, predetermined fashion, you can use these amplitude and phase changes to encode information upon a sine wave, a process known as modulation. Modulation changes a higher frequency carrier signal in proportion to a lower frequency message, or information, signal. I/Q sampling is highly prevalent in RF communications systems, and more generally in signal modulation [11]. The equation representing a sine wave is:

$$A_c \cos(\, 2\Pi f_c t \, + \, \Phi) \tag{3.1}$$

where A_c is the amplitude, f_c is the frequency and Φ is the phase of the signal.

The frequency and phase of the sine wave equation can be collectively referred to as the phase angle. Therefore, we can represent the instantaneous state of a sine

wave with a vector in the complex plane using amplitude (magnitude) and phase coordinates in a polar coordinate system as follows:

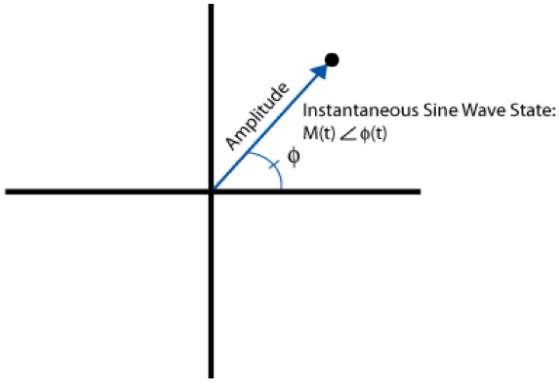


Figure 3.5. Polar Representation of a Sine wave.

I/Q data is merely a translation of amplitude and phase data from a polar coordinate system to a Cartesian (X,Y) coordinate system. Using trigonometry, you can convert the polar coordinate sine wave information into Cartesian I/Q sine wave data. These two representations are equivalent and contain the same information, just in different forms. This equivalence is shown in Figure 3.6.

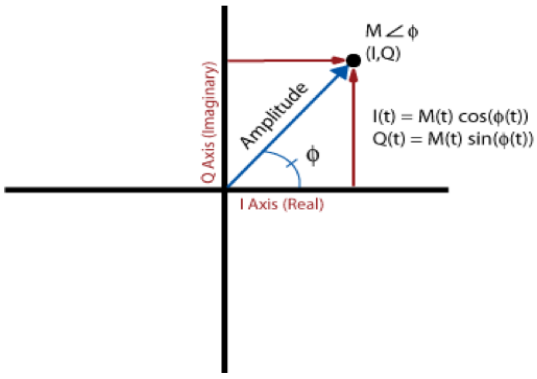


Figure 3.6. I and Q represented in Polar Form.

Because amplitude and phase data seem more intuitive, one might assume you should use polar amplitude and phase data instead of Cartesian I and Q data. However, practical hardware design concerns make I and Q data the better choice. Precisely varying the phase of a high-frequency carrier sine wave in a hardware circuit according to an input message signal is difficult. A hardware signal modulator that manipulates the amplitude and phase of a carrier sine wave would therefore be expensive and difficult to design and build, and, as it turns out, not as flexible as a circuit that uses I and Q waveforms. To understand how to avoid manipulating the phase of an RF carrier directly, refer to the following I/Q modulation equations:

$$\cos(A + B) = \cos(A)\cos(B) - \sin(A)\sin(B) \tag{3.2}$$

$$A\cos(2\Pi f_c t + \Phi) = A\cos(2\Pi f_c t)\cos\Phi - A\sin(2\Pi f_c t)\sin\Phi \tag{3.3}$$

$$From\ Figure\ 3.6,\ I = A\cos(\Phi);\ Q = A\sin(\Phi) \tag{3.4}$$

$$A\cos(2\Pi f_c t + \Phi) = I\cos(2\Pi f_c t) - Q\sin(2\Pi f_c t) \tag{3.5}$$

where I is the amplitude of the in-phase carrier and Q is the amplitude of the quadrature-phase carrier.

The difference between a sine wave and a cosine wave of the same frequency is a 90-degree phase offset between them. Essentially, what this fact means is that you can control the amplitude, frequency, and phase of a modulating carrier sine wave by simply manipulating the amplitudes of separate I and Q input signals. With this method, you do not need to directly vary the phase of an RF carrier sine wave.

Thus in I/Q sampling, each sample of the signal has a corresponding I and Q component. I/Q data can easily be represented as the real and imaginary parts of a complex number. The magnitude and phase of each sample can be calculated using the following relations:

$$Magnitude: \ M = \sqrt{I^2 + Q^2} \tag{3.6}$$

$$Phase: \ \Phi = \arctan(Q/I) \tag{3.7}$$

The use of the relations (3.6) and (3.7) in processing the acquired signal for our experiment will be discussed in Chapter 4.

3.2.2 Important Definitions and Abbreviations

Complex Signal: A complex signal or a quadrature signal is a two-dimensional signal whose value at some instant in time can be specified by a single complex number having two parts: the real and imaginary parts. In the field of communication, they are referred to as in-phase and quadrature phase respectively.

RF signal or RF data: Radio frequency (RF) is a rate of oscillation in the range of around 3 kHz to 300 GHz, which corresponds to the frequency of radio waves, and the alternating currents which carry radio signals. In our experiment, RF signal or RF data is used to denote unprocessed data coming into the receiver from the femtosatellite prototype.

IF signal: In communications and electronic engineering, an intermediate frequency (IF) is a frequency to which a carrier frequency is shifted as an intermediate step in transmission or reception of a signal [14].

Baseband (BB) signal: A baseband signal includes frequencies that are very near zero in comparison to the highest frequency in the signal. In general, signals have a whole range of frequencies added together. Parts of the signal at higher frequencies are con-

verted to lower frequencies for transmission purposes as software receivers (the host PC in our case) cannot handle the high frequencies of the RF signal. The original high frequency components are referred to as the RF signal and the low frequency components are referred to as the Baseband signal.

RF Front-end: In a radio receiver circuit, the RF front end is a generic term for all the circuitry between the antenna and the first intermediate frequency (IF) stage. It consists of all the components in the receiver that process the signal at the original incoming radio frequency (RF), before it is converted to a lower intermediate frequency (IF).

ADC and DAC: The Analog to Digital Converter (ADC) is a device that converts a continuous physical quantity to a digital number that represents the quantity's amplitude. The DAC (Digital to Analog Converter) does the opposite function of the ADC.

FPGA: Field-programmable gate arrays (FPGAs) are re-programmable silicon chips. In contrast to processors that you find in your PC, programming an FPGA rewires the chip itself to implement your functionality rather than run a software application. Advantages of using FPGAs include faster I/O (Input/Output) response times, increased functionality–exceeding the computing power of DSPs (Digital Signal Processors) amongst others [10].

DDC and DUC: A digital down-converter (DDC) converts a digitized real signal centered at an intermediate frequency (IF) to a basebanded complex signal centered at zero frequency. In addition to downconversion, DDC's typically decimate to a lower sampling rate, allowing follow-on signal processing by lower speed processors. The Digital Up-Converter (DUC) does the opposite function of the DDC.

3.2.3 The USRP

Ettus Research's USRP (Universal Software Radio Peripheral) is a computer-hosted software radio peripheral. It uses a host driver called the USRP Hardware Driver software (UHD) that works on all major platforms (Windows, Mac, Linux). The USRP includes an RF front-end device and a DSP which together, can be used to receive and pre–process an incoming RF signal before sending it to the host computer. A USRP has two stages of tuning: The RF front-end translates the incoming RF signal to an IF signal and the DSP translates the IF signal to a BB signal as shown in Figure 3.7 [23].

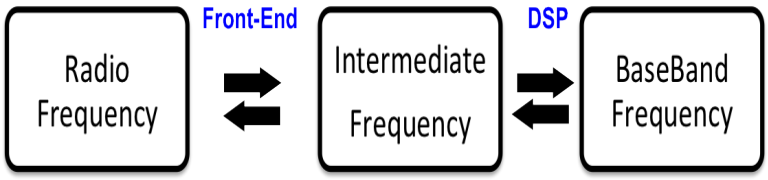


Figure 3.7. Stages of Signal Processing in the USRP.

In our experiment, we use the USRP1 that belongs to the USRP family of products. The USRP1 provides an entry-level platform with built in MIMO (Multiple Input Multiple Output) expansion and a modular design allowing the hardware to operate from DC to 6 GHz. Its architecture includes an Altera Cyclone FPGA, four ADCs, four DACs, four DDCs, two DUCs and USB 2.0 connectivity with the host computer. The ADCs have a 64 megasamples per second (MS/s) sampling rate and a 12–bit resolution; the DACs have a 128 MS/s sampling rate and a 14–bit resolution. The DDCs and DUCs have programmable decimation rates. The USB 2.0 interface (480 Megabits per second) is capable of processing signals up-to 16 MHz

wide. The USRP1 includes connectivity for two daughtercards, enabling two complete transmit/receive chains [21].

The basic block diagram of the USRP operation is shown in Figure 3.8.

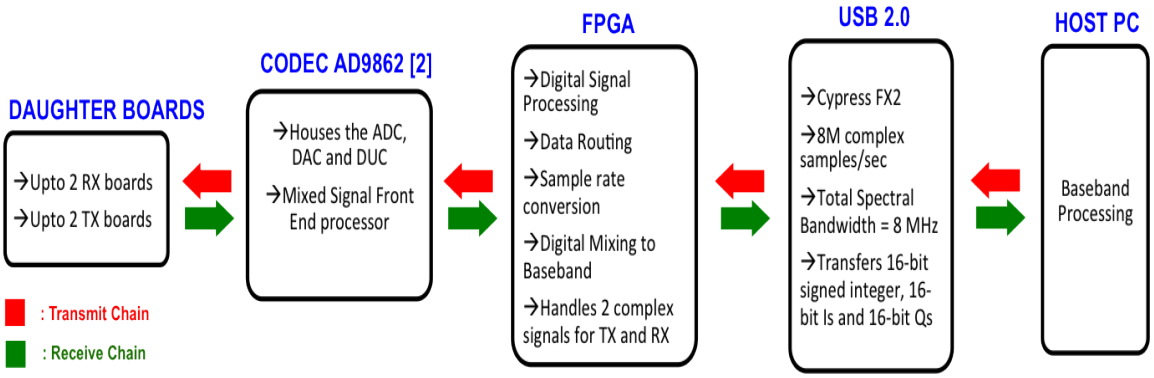


Figure 3.8. Basic Block Diagram of the USRP.

We will now discuss the features and operation of the different components that make the USRP. *Please note that although the USRP has both transmit (TX) and receive (RX) capabilities, we will only discuss the RX chain of the USRP operation as our application only deals with the RX chain. Refer [8–10, 13, 15, 21–24] for details on the transmit chain of the USRP.*

3.2.3.1 Daughterboards

The Ettus Research daughterboards provide the circuitry required for an interface between the ADCs, DACs, and the outside world. Each of these daughterboards includes one or more channels called a frontend [21]. One or more daughterboards can be integrated into a USRP device. The USRP1 includes connectivity for two daughterboards, enabling two complete transmit/receive chains. In our experiment, we use a receive (RX) daughterboard called TVRX2 that receives signals in the range,

42–870 MHz. The TVRX2 board has two real–mode RF frontends, RX1 and RX2. It is operated at low IF. The IF gain range is 0–30 dB.

Automatic Gain Control (AGC) is a signal processing technique where the average or peak output signal is used to adjust the gain to a suitable level, enabling the circuit to work satisfactorily with a greater range of input signals. Without AGC, the user needs to 'ride' (rapidly reduce) a gain control to prevent their ears being blown off should a strong signal or a big noise pulse be received [8]. AGC is used to maximize sensitivity and prevent over–saturation of the ADC. The TVRX2 has an *always-on* Automatic Gain Control (AGC). The software controllable gain is the final stage which controls the AGC set-point for output to ADC [22].

As stated before, the TVRX2 includes two, independent receive chains and outputs two IF signals. The USRP device can sample one of these signals or both. In this case, each IF signal is paired with a 0-filled stream to create an I/Q pair and passed along to an RX DSP Chain. In this case, two separate RX DSP chains are used [21]. Refer Figure 3.9.

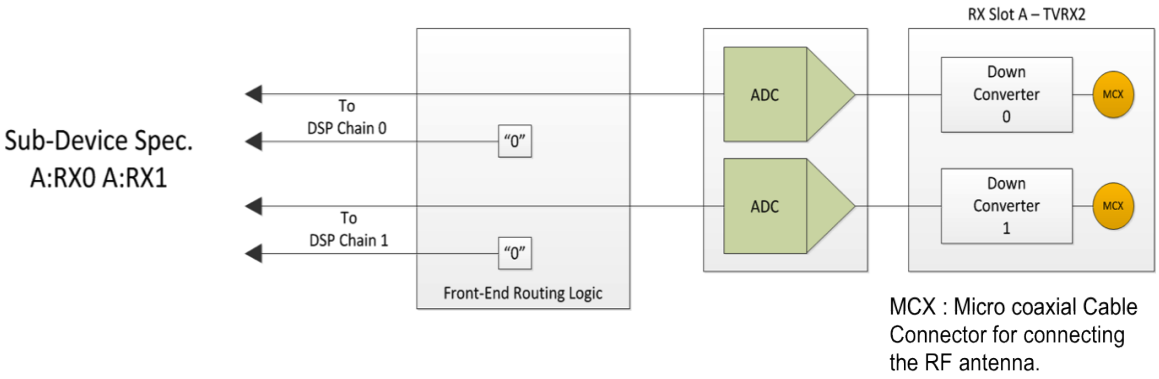


Figure 3.9. TVRX2 Configuration.

3.2.3.2 CODEC

A codec (*co*der–*dec*oder) is a device capable of encoding or decoding a digital data stream or signal. It encodes a data stream or signal for transmission, storage or encryption, or decodes it for playback or editing.

The USRP uses two codec AD9862 mixed-signal front-end processors that are optimized for broadband communication applications. The AD9862 receive path (RX) consists of two channels that each include a high performance, 12-bit, 64 MSPS ADC, input buffer, Programmable Gain Amplifier (PGA), digital Hilbert filter, and decimation filter. The RX can be used to receive real or I/Q data at baseband or low IF. The output data bus can be multiplexed to accommodate a variety of interface types [20].

In theory, the ADC that is housed on the CODEC can digitize a band as wide as 32 MHz. They can bandpass sample signals of upto 150 MHz. The recommended upper limit on frequency is 100 MHz. A PGA is used before ADCs to amplify the input signal and to utilize the entire input range of the ADCs in case the signal is weak. The available sampling rates are submultiples of 128 i.e. 64MS/s, 32MS/s and so on [13].

3.2.3.3 FPGA

The FPGA is responsible for the Digital Signal Processing (DSP) including data routing, sample rate conversion by user–defined factor and digital mixing to baseband in the RX path. It can handle up to two complex signals per RX. It reduces data rates so that can be transferred over the USB 2.0 interface. In the USRP, high sampling rate takes place in the FPGA and low sampling rate takes place in the host PC. The FPGA includes the DDC [13].

The DDCs downconverts the signals coming into the FPGA from IF to BB. It then decimates the signals so that they are acceptable by the USB interface to the host PC [13]. The block diagram of the DDC is shown in Figure 3.10.

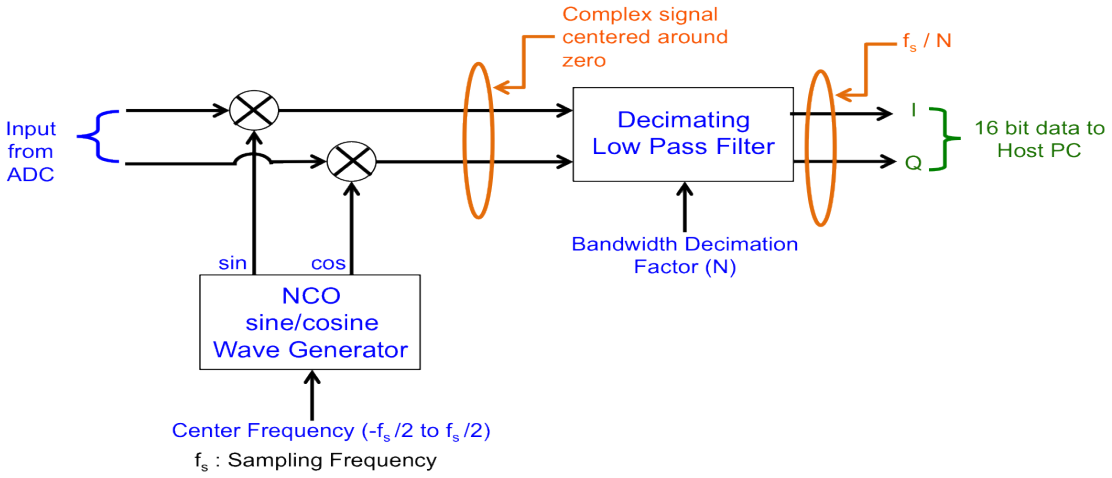


Figure 3.10. Block Diagram of the DDC.

As seen in Figure 3.10, the complex IF signal is multiplied by the constant frequency exponential signal. The resulting signal is also a complex signal centered around zero frequency. Then, this signal is decimated by a factor of N. The decimator can be treated as a low pass filter followed by a downsampler. The resultant samples are sent over the USB 2.0 interface in 16–bit signed integer, I/Q format i.e. 16–bit I and 16–bit Q complex data. This results in 8 million complex samples/second across the USB 2.0 interface. This gives a maximum effective total spectral bandwidth of 8 MHz by the Nyquist sampling criteria. In contrast, the Fun Cube Dongle which is the radio receiver peripheral used by Cornell group for the Sprites has a 96 kHz bandwidth.

Finally when the I/Q complex signal enters the host PC via the Cypress FX2 USB 2.0 interface, the software on the PC takes over the signal processing [15].

3.2.3.4 The RX Path

The RX Path has been summarized in Figure 3.11. The incoming RF signal is routed through the TVRX2 daughterboard into the CODEC chips. Here, signal gets amplified by the PGA followed by a digitization performed by the ADC. This digitized signal gets processed and decimated by the DDC in the FPGA. A 16-bit, I/Q format, BB signal is generated by the FPGA which is sent over the USB 2.0 interface at the rate of 8 MS/s, with a spectral bandwidth of 8 MHz. Finally, the signal from the USB interface enters the host PC which performs further signal processing with a software radio application.

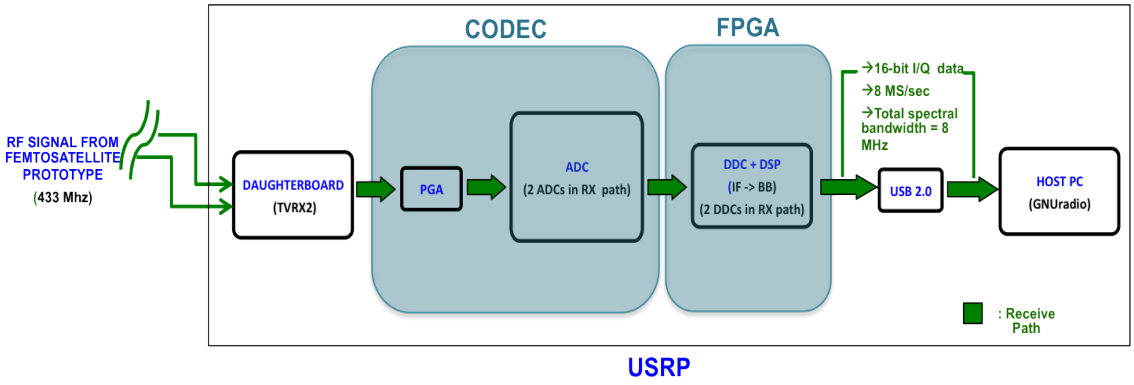


Figure 3.11. The RX Path of the USRP.

3.2.3.5 RF Antenna:

We use a custom RF antenna of the length of 6.8–cm for reception of the 433MHz signal transmitted by our femtosatellite prototype. It connects onto an SMA (SubMiniature version A), coaxial RF connector housed on the USRP.

The USRP1 along with the components described above, as used in our experiment is shown in Figure 3.12.

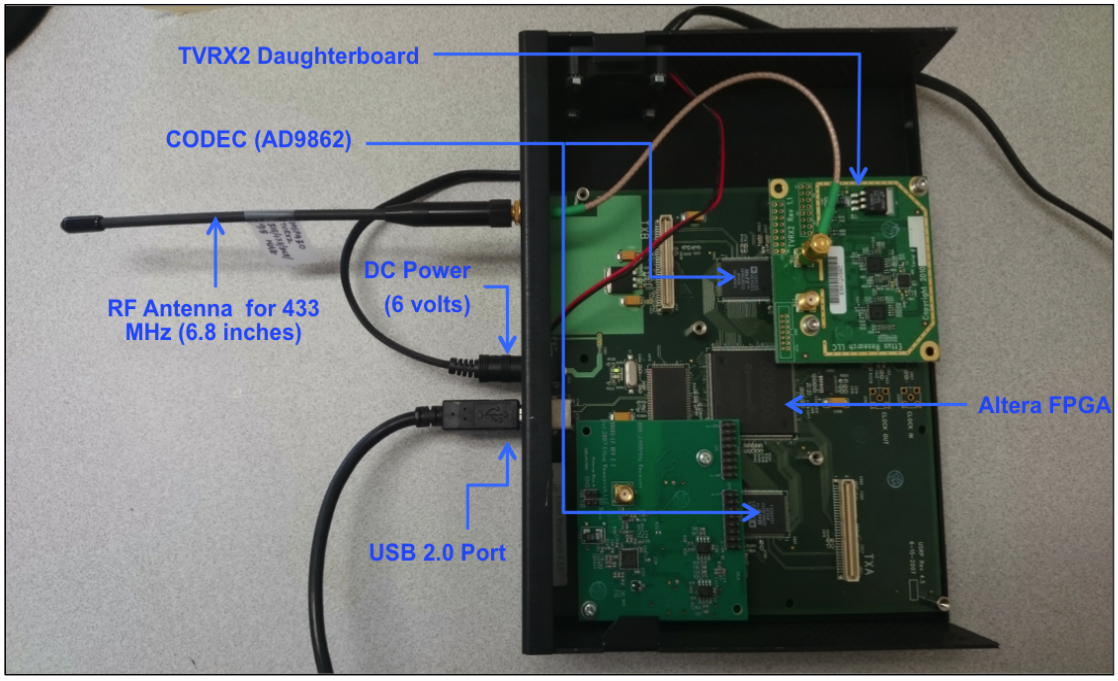


Figure 3.12. The USRP1 as used in the Experimental Setup.

3.2.4 GNU Radio

Software Defined Radios (SDR) are used because hardware radio cannot handle changes in RF services and standards. It will require re-design, re-manufacture, re-marketing and re-development of the hardware radio peripheral. It is much easier to customize a software radio to suit the changing requirements of our research. In software radio, changing RF parameters is as simple as including certain code or even just clicking a button. This flexibility is not offered by hardware radios.

GNU Radio is the software that we use in our experiment to process the data coming in from the USRP. In this section, we will study the features of GNU Radio and its use in our experiment.

3.2.4.1 Introduction

GNU Radio is a free and open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment [7].

Using GNU Radio, a radio can be built by creating a graph where the vertices are signal processing blocks and the edges represent the data flow between them. In our experiment, we use GNU Radio Companion (GRC) which is a graphical tool for creating such signal flow graphs and generating flow-graph source code. The GNU Radio components are connected as shown in Figure 3.13.

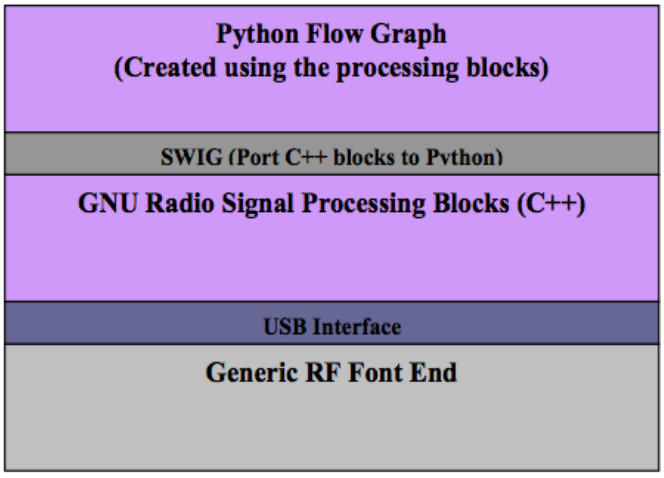


Figure 3.13. GNU Radio Architecture.

The signal processing blocks are implemented in C++ and the graphs are constructed and run in Python. Conceptually, a signal processing block processes an infinite stream of data flowing from its input ports to its output ports. A block's attributes include the number of input and output ports it has as well as the type of data that flows through it. Some blocks have only output ports or input ports. Input and output ports serve as data sources and sinks in the graph. For instance, there are sources that read from a file or ADC, and sinks that write to a file, a DAC or a graphical display. More than 100 blocks are currently implemented in GNU Radio and more are being implemented each day. Using a generic RF front end and few other hardware components like the ADC and DAC (all included in the USRP), GNU Radio code can implement radio functionalities [15].

3.2.4.2 The USRP interface with GNU Radio

The bridge between GNU Radio and the USRP device is a set of blocks in the GNU Radio–UHD component, which includes the USRP source block. The USRP Source Block accepts the complex data coming in from the USRP device and provides this data to downstream processing blocks in GNU Radio.

In our experiment, we use a USRP block in GNU Radio to extract the complex 16-bit data coming in from the USRP 1 device (Refer section 3.2.3.3). We set the significant parameters as per the requirements of our experiment. In particular, we set the sampling rate of the incoming signal at 8 MS/s. We pick this sample rate because the 16–bit I/Q data generated by the DDC is transferred over the USB 2.0 interface at the rate of 8MS/s (Refer section 3.2.3.3). The center frequency of the incoming signal is set at 433 MHz. We also specify the channel on which our RF antenna is connected and the format in which we want the output of the USRP Source Block

i.e. Complex float32. The USRP Source Block configuration used in our experiment is shown in Figures 3.14 and 3.15.

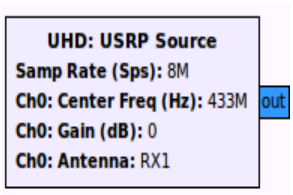


Figure 3.14. The USRP Source Block in the GNU Radio Flowgraph.

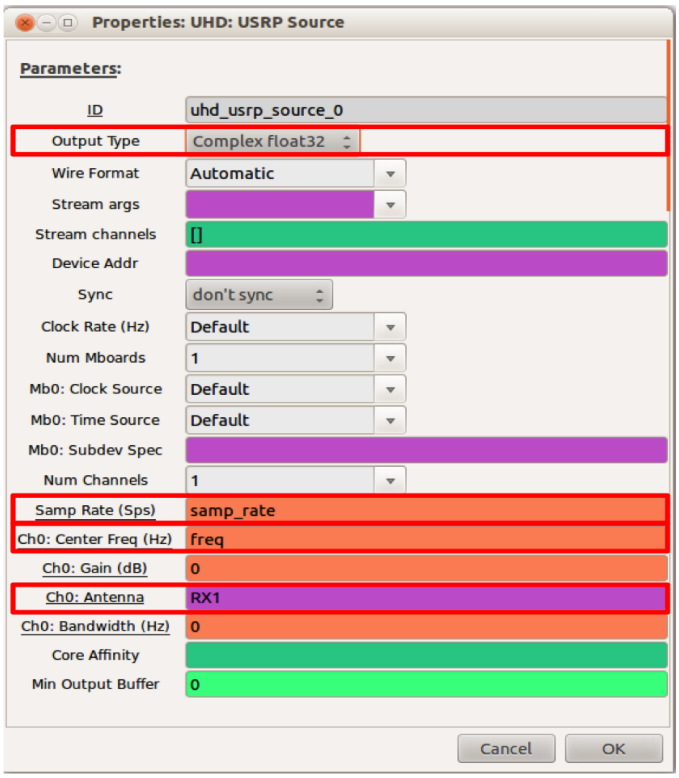


Figure 3.15. The USRP Source Block with Parameter Settings.

The sample rate and frequency prameters of the USRP block are set in separate variable text box blocks in GNU Radio as shown in Figures 3.16 and 3.17.

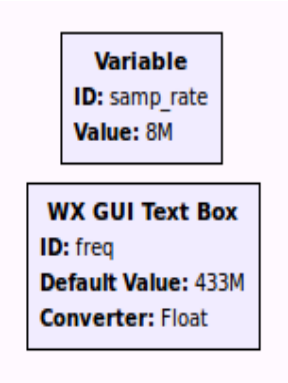
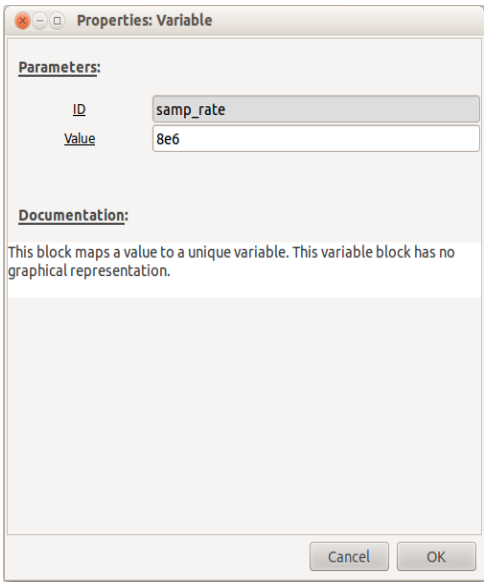
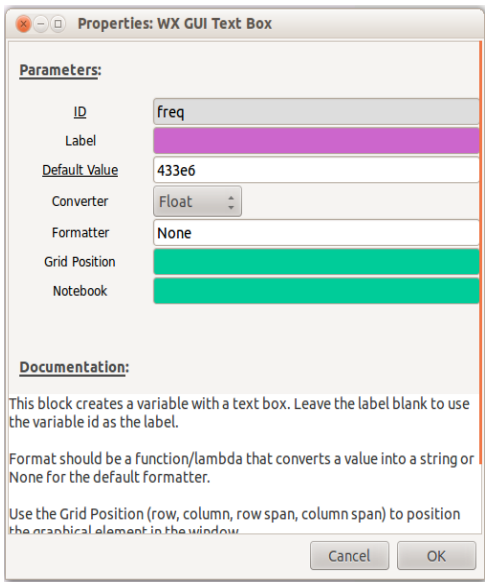


Figure 3.16. The Variable and Text Box in GNU Radio.



(a) Parameter Setting of Variable Block



(b) Parameter Setting of Text Box

Figure 3.17. Variable and Text Box Parameter Settings.

3.2.4.3 Sinks

GNU Radio has various types of sink blocks that can be used to plot or record the incoming data. We use the WX GUI FFT Sink block to plot our data and the File Sink block to record our data.

WX GUI FFT Sink Block:

WX GUI is a widget toolkit for Graphical User Interfaces (GUI). The FFT Sink acts as a Spectrum Analyzer by doing a short time discrete Fourier transform (STFT). This allows us to view the peaks of the signal transmitted by our femtosatellite prototype in real time. It helps in ensuring that pulses of the signal are being received as they are transmitted. The WX GUI FFT Sink Block configuration we use in our experiment is shown in Figure 3.18 and 3.19.

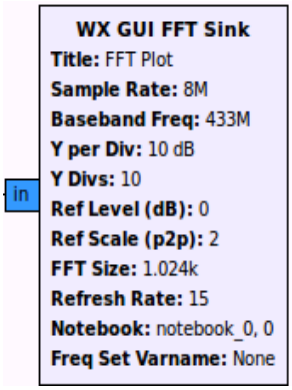


Figure 3.18. The WX GUI FFT Sink Block in the GNU Radio Flowgraph.

Figure 3.19. The WX GUI FFT Sink Block with Parameter Settings.

The File Sink Block:

The File Sink Block writes the samples obtained from the USRP Source Block to a file in binary format (little–endian). The data format depends on the data type used in the File Sink block. We use the complex data type (4 bytes per sample as real and imaginary) as that is format in which we receive data from the USRP. We also specify the file name and the destination where the binary file is to be saved. The File Sink block configuration we use in our experiment is shown in Figures 3.20 and 3.21.

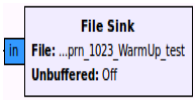


Figure 3.20. The File Sink Block in the GNU Radio Flowgraph.

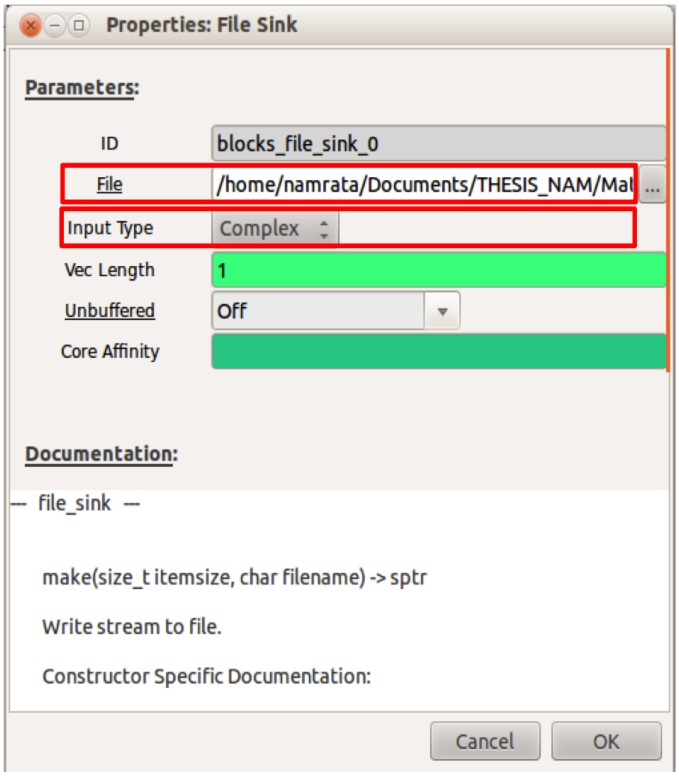


Figure 3.21. The File Sink Block with Parameter Settings.

3.2.4.4 The GNU Radio Flowgraph

The individual GNU Radio blocks described above are put together in GRC in order to access and record data from the USRP1 device. The downstream flowgraph created in GRC for our experiment is shown in Figure 3.22.

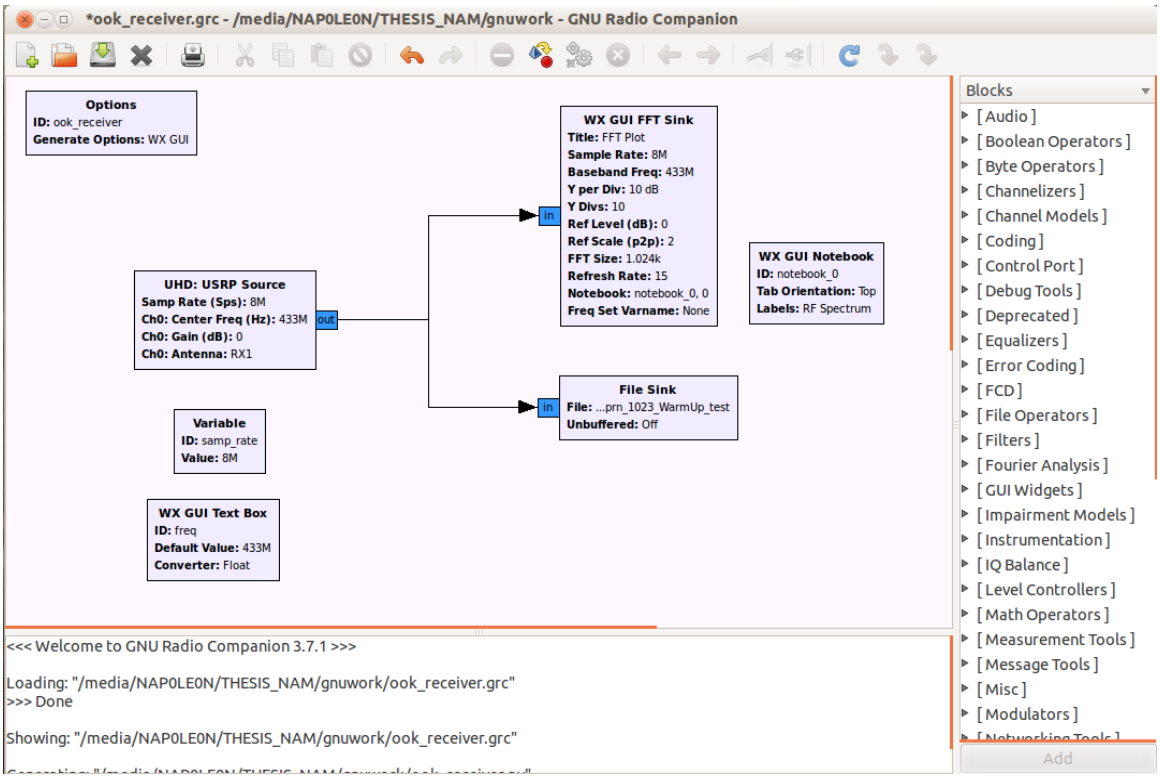


Figure 3.22. The GNU Radio Flowgraph for Data Recording.

As seen in Figure 3.22, The complex data from the USRP1 device is extracted by the USRP Source block. This extracted data is sent into the WX GUI FFT Sink block for real time plotting of the data. At the same time, the extracted data from the USRP Source block is also sent into the File Sink block that records the data into a binary file in the complex format.

Thus we have discussed in detail the reasoning behind our hardware selection and design. In the next chapter, we will discuss how all these components are integrated to design our experiment.

CHAPTER 4

Procedure

In this chapter, we will briefly discuss the auto correlation process. Next, we will describe the signal processing performed in our experiment in detail followed by a summary of the design of experiment.

4.1 Autocorrelation

From section 2.2 we know that a PRN sequence is a series of 0s and 1s generated by a specific algorithm. Once the bits of the PRN sequence are mapped from 0s and 1s to 1s and (-1)s respectively, they can be autocorrelated with similar PRN sequence [17]. Auto–correlation rests upon the fact that the PRN sequence is nearly uncorrelated with itself except for zero shift. For the PRN sequence of satellite k, i.e. $x^{(k)}$,

$$\sum_{i=0}^{1022} x^{(k)}(i) \, . \, x^{(k)}(i+n) \approx 0, \forall \, |n| \geq 1 \tag{4.1}$$

The left–hand side of equation (4.1) defines the *auto-correlation function* of a PRN sequence for shift n. The auto–correlation function of a PRN sequence is nearly zero except for zero shift where it has a sharp peak, also referred to as the *correlation spike*. An example of auto-correlation of the PRN 5 sequence with a bit-shifted version (n= 350) of itself is shown in Figure 4.1 [18]. As the PRN sequences match, we see a clear correlation spike in the correlation function.

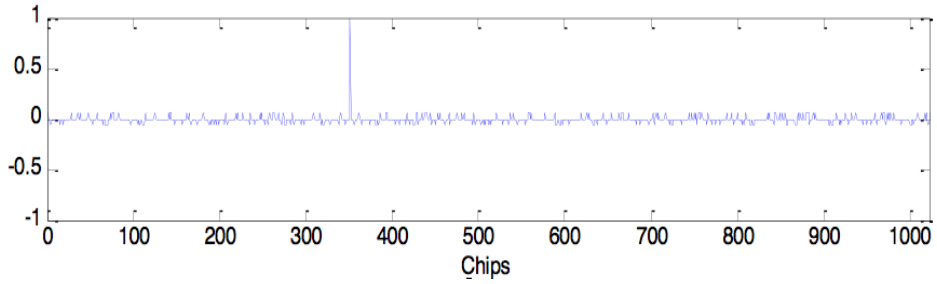


Figure 4.1. Auto-correlation of the PRN 5 Sequence [18].

Figure 4.2 below shows the correlation of the PRN 5 and PRN 2 sequences. As the PRN sequences do not match, there is no correlation spike.

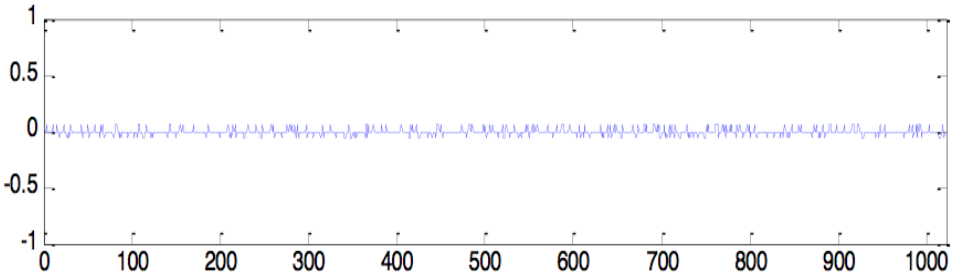
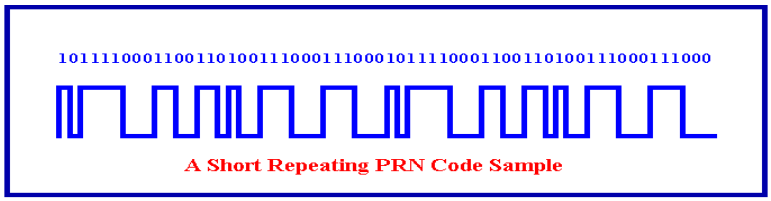


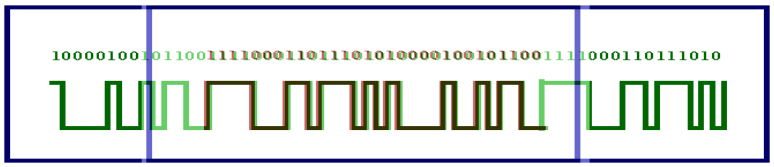
Figure 4.2. Cross-correlation of the PRN 5 Sequence with the PRN 2 Sequence [18].

The receiver or the ground station uses this auto-correlation in the following way. The ground station has a replica of the concerned PRN sequence in its memory that was generated using the algorithm described in section 2.2. The ground station essentially shifts this replica PRN sequence until it matches up with the one it received from the satellite. This is done by using the autocorrelation function of the received signal and the time shifted replica in the ground station. When the two sequences match up, there will be a correlation spike [18] as shown is Figure 4.1. The Figure

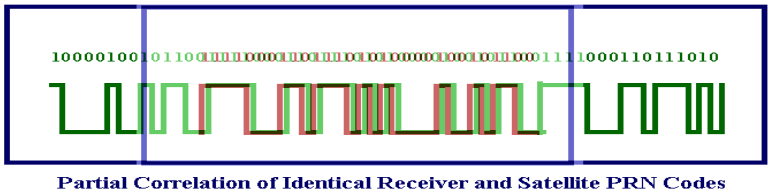
4.3 illustrates full, partial and no correlation. The Figure 4.4 shows the process of matching the replica PRN sequence with the received PRN sequence.



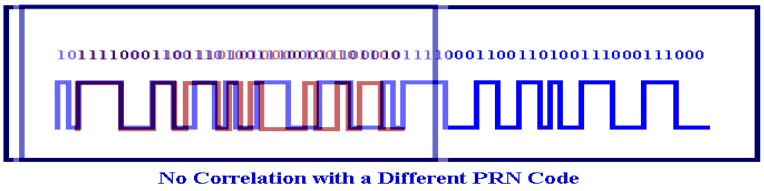
(a)



(b)



(c)



(d)

Figure 4.3. Full, Partial and No Correlation [6].

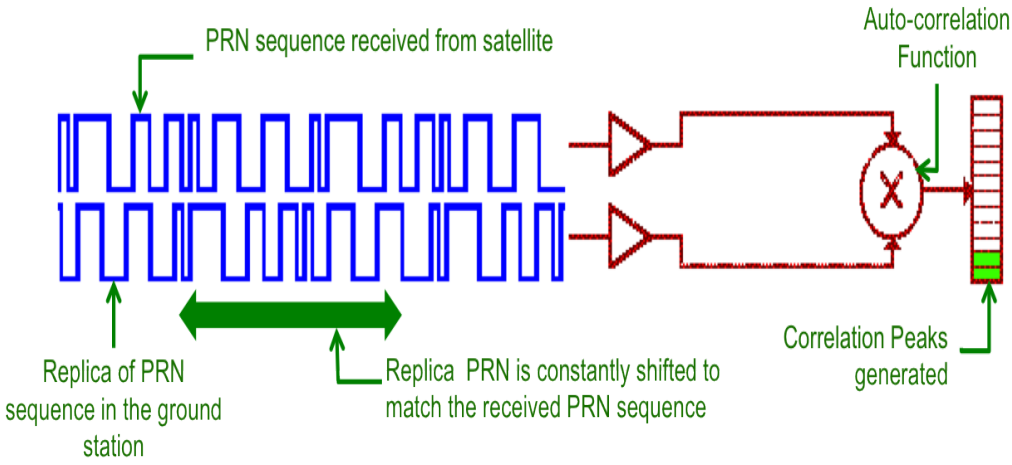


Figure 4.4. Process of Matching the Received and Replica PRN Sequences [6].

The number of chips that were shifted in the replica PRN in order to match the received PRN sequence gives the clock offset between the clocks of the satellite (transmitter) and the ground station (receiver). The location of the correlation spike in the entire spectrum gives the time taken by the PRN sequence to travel from the satellite to the receiver, thus giving us the distance between the two. In reality, this process is biased by clock errors associated both with the satellite and the receiver. In addition, there is interference from background noise that further weakens the already weak satellite signal.

Similarly we have the phenomenon of cross correlation that is used to correlate signals received from two different sources. It is based on the fact that PRN sequences are nearly orthogonal to each other. The sum of the term–by–term products of the two sequences, shifted arbitrarily with respect to each other, is nearly zero. Cross correlation can be used find the location of a femtosatellite with respect to the other. In our research, we will focus on autocorrelation as we deal with only one femtosatellite.

4.2 Signal Processing

From section 3.2.4.3, we know that the data received from the femtosatellite prototype is recorded into a binary file in GNU Radio. In the signal post–processing stage of our experiment, we extract data from these binary files in MATLAB. Further, we extract, compute and plot the significant components of the signals. This process is performed in a sequence of signal processing steps described below.

4.2.1 Extracting Raw Data

In MATLAB, we open the binary file using the commands *fopen* and *fread*. These commands store the raw data from the binary file into a vector. While using these commands, we specify the data format of the incoming data (float32) and that of the data extracted into the vector (single). We also specify the machine format, which specifies the order of reading bits within a byte. In our case, the machine format is IEEE–Little Endian–64–bit (ieee–le.64) (Refer section 3.2.4.3). The magnitude of the elements of the raw data vector are plotted in order to view the signal as a whole. When the raw data is plotted in its entirety, we see that the transmitted pulses are within a very minute part of the entire signal. The rest is noise. This happens because it is not humanly possible to record a sample of data for exactly the same period that the pulses are transmitted as they have a maximum duration of about 2 seconds. In most cases, it is less than a second. Hence, we have to cut out the unnecessary noise captured before and after the transmitted pulses and extract only the required raw data into a vector. We will refer to this extracted vector as the *raw data vector*.

4.2.2 Extracting Is and Qs

From the raw data vector obtained in section 4.2.1, we extract the Is and Qs (Refer section 3.2.1 for details about I/Q data). Alternate elements in the raw data

vector are Is and Qs. So the first element and every second element after in the raw data vector are stored in the I vector. Similarly the second element and every alternate element after in the raw data vector are stored into the Q vector. The magnitude of elements of the vectors I and Q are plotted thus showing the in-phase and quadrature components of the signal.

4.2.3 Computing Magnitude and Phase

We then compute the magnitude and the phase of the signal using equations (3.6) and (3.7), and store them in vectors m and *phi*. We plot the elements of vectors m and *phi*. The magnitude vector plot clearly shows the peaks and valleys of the transmitted pulses. These peaks will be used for further processing as described in the following sub–sections.

4.2.4 The Received PRN Sequence

As discussed in the previous sub–section, the peaks and valleys of the transmitted pulses are clearly visible in the magnitude plot of the signal. Equipped with this capability, we now transmit the entire 1023 chips of the PRN 19 sequence from our femtosatellite prototype. This transmitted sequence is received in the manner described in sections 3.2.3.5 and 3.2.4.4. The binary file is processed as per the procedure described in sub–sections 4.2.1 – 4.2.3. The plot of the magnitude of the received PRN sequence signal clearly shows the peaks and valleys of the 1023 chips transmitted by the femtosatellite prototype.

In order to retrieve the PRN sequence in the form of 1s and 0s, we need to implement a filter. For our application we implement a very basic filter that uses the thresholding technique. In this technique, we physically determine the threshold for elements of the magnitude vector. The elements of the magnitude vector are

mapped into 1s when this threshold is reached and to 0s when the value is less than the threshold. The entire magnitude vector is arbitrarily divided into fractions depending on the peaks of the pulse magnitudes so that we can balance the signal in an attempt to eliminate the effects of noise, heat and AGC on the magnitude of the pulses. A threshold is established for each fraction and the mapping to 1s and 0s is performed. Once we obtain the entire vector based on the fraction-specific thresholds, consisting of 1s and 0s, we further map the elements of the thresholded vector to 1s and (-1)s i.e 0 to 1 ans 1 to (-1)s so that we can apply the property of auto–correlation as described in section 4.1.

So at this point, we have the PRN sequence received by the ground station from the femtosatellite prototype in the form of 1s and (-1)s. From now on, when we refer to the received PRN sequence, we mean the PRN sequence in the form of 1s and (-1)s.

4.2.5 The Replica PRN Sequence

We generate a replica PRN 19 sequence using the algorithm described in section 2.2. Each chip in this 1023–chip sequence is super–sampled. While super–sampling the PRN sequence, we convert each 0 into eight 0s and each 1 into eight 1s. We do this because we sample the received signal at the rate of 8MS/s and each chip has a duration of 1 μs. This implies that each chip of the replica PRN sequence needs to be repeated 8 times. This super–sampled sequence of 0s and 1s is also mapped into 1s and (-1)s just like the received PRN sequence. Thus, now have a replica PRN sequence in the form of 1s and (-1)s.

4.2.6 Autocorrelation Plots

We now perform two auto-correlations. First, we auto-correlate the replica PRN sequence with a bit-shifted version of itself. Secondly, we auto-correlate a bit-shifted

version of the received PRN sequence with the replica PRN sequence. Finally, we compare the two auto-correlations obtained to confirm the location of the correlation spike. The resultant plots obtained in the different stages of signal processing will be discussed in the next chapter.

The following block digram in Figure 4.5 summarizes the signal processing in MATLAB:

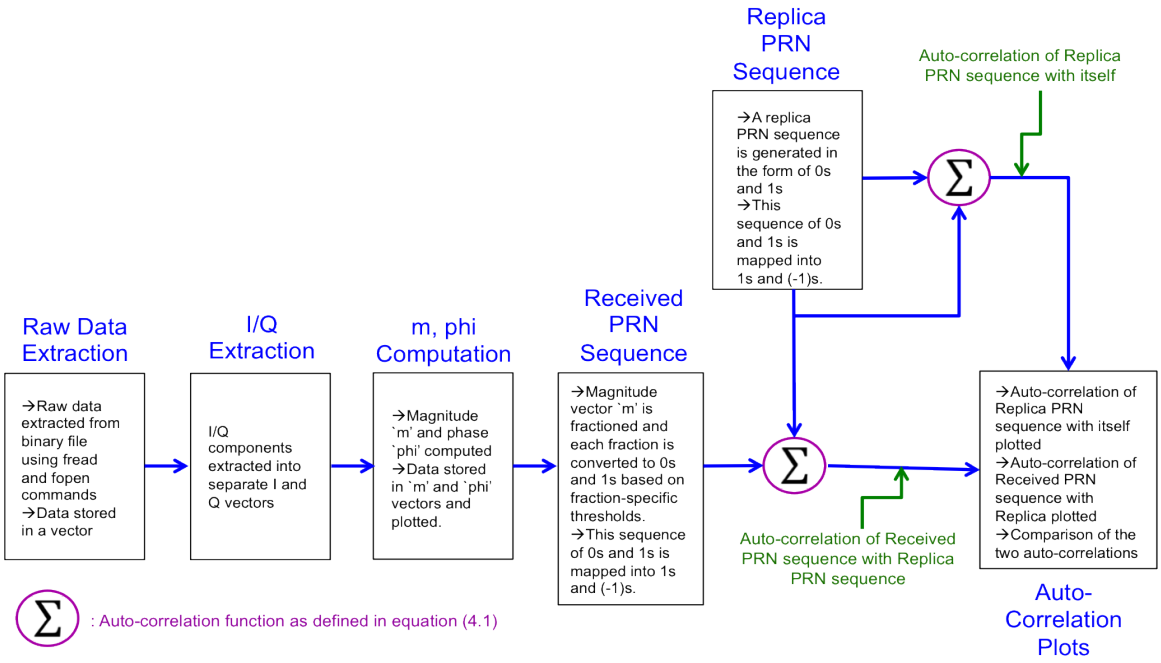


Figure 4.5. Stages of Signal Processing in MATLAB.

4.3 Design of Experiment (DOE)

In this this section we will state our DOE. It will summarize all the equipment and techniques that have been described in this document leading up to this point.

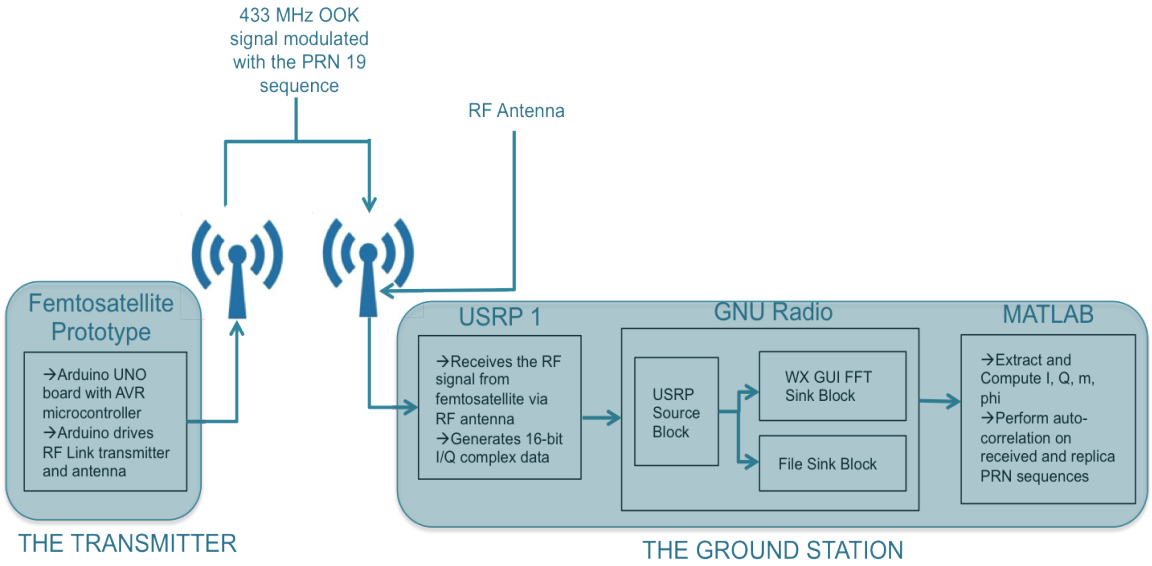


Figure 4.6. Design of Experiment.

As seen in Figure 4.6, we transmit a 433 MHz, OOK signal modulated with the PRN 19 sequence from our femtosatellite prototype. This is done by providing the PRN 19 sequence in the form of 1s and 0s in the Arduino program. Then the program directs the Arduino to ride its digital transmit pin high when a 1 is transmitted and low when a 0 is transmitted for every chip in the PRN sequence. These on–off variations in the digital pin of the arduino drive the RF link transmitter. The transmitter emits a 433 MHz OOK signal, modulated with the PRN 19 sequence by the on–off action of the Arduino.

On the receiving end, the USRP receives this 433 MHz OOK signal and generates 16-bit I/Q complex data. GNU Radio accepts this complex data via the USRP source block. From the USRP source block, the data is plotted in real–time using the WX GUI FFT Sink block. The same data is also recorded into a binary file using a File Sink block. We extract raw data from the binary file in MATLAB and use it to extract, record and plot significant components of the signal. We then perform

auto-correlation of the replica PRN with a bit–shifted version of itself and an auto–correlation of the bit–shifted version of received PRN sequence with the replica PRN sequence. Finally we compare the two auto–correlations to determine the location of the correlation spike.

CHAPTER 5

Results

GPS satellites transmit gold codes which have a length of 1023 chips. Each chip has a duration of 1 μs. This means that the GPS receivers need to have a reception sensitivity of at most 1 μs to correctly receive and decode the signals transmitted by the GPS satellites. This reception sensitivity plays a key role in providing the GPS satellites with ranging capability.

One objective of our experiment is to achieve a reception sensitivity approaching that of GPS satellites. This implies that we need our femtosatellite prototype to transmit a signal capable of being received by our prototype receiver ground station *with a reception sensitivity of 1 μs.* The other objective is to auto-correlate the received PRN–encoded OOK signal with a replica signal and be able to see a correlation spike indicating a match of PRN sequences.

The process of performing the experiment as described in the design of experiment section (4.2), has not been straightforward. We have had to work around several obstacles and discrepancies that emerged along the way. In this Chapter, we discuss the results obtained at every intermediate step leading to the objective and the methods we used to overcome the difficulties. Finally, we discuss the results of our experiment with which we substantiate the ability to achieve the above mentioned goals with our prototype femtosatellite and receiver. We will start by describing the technique used to transmit an OOK signal from the femtosatellite prototype.

5.1 The Transmitted Signal

As mentioned in Section 4.2, our femtosatellite prototype generates a simple OOK signal by driving the digital pin of the Arduino high (5 volts) when a 1 is transmitted and low (0 volts) when a 0 is transmitted. The Arduino can be easily programmed to do so. The digital pin in turn drives the RF link transmitter that emits a 433 MHz signal via the custom–built RF antenna. An example of a transmitted OOK signal is shown in Figure 5.1. The signal in this case has a duty cycle of 50% i.e. it is ON and OFF for the same duration.

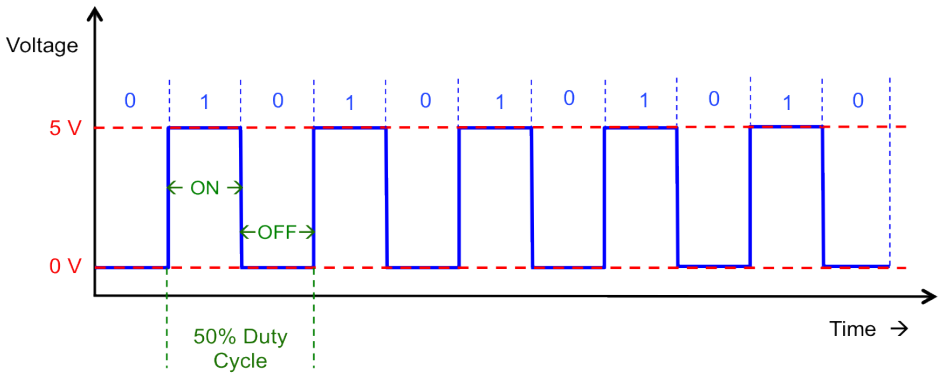
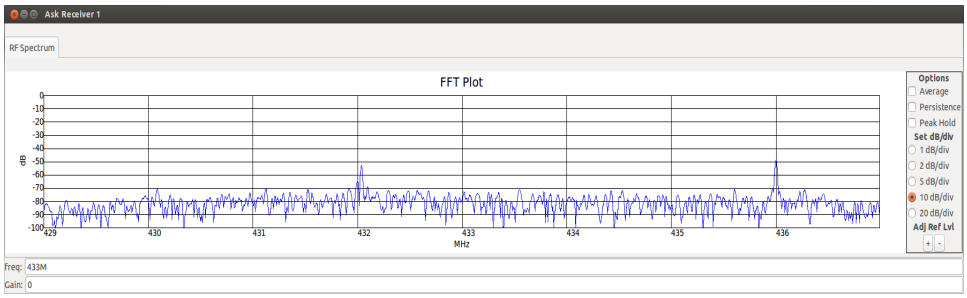


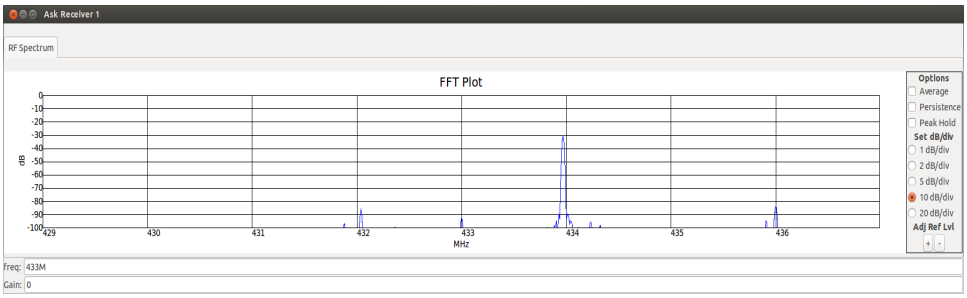
Figure 5.1. The Transmitted OOK Signal.

5.2 Initial Reception Test

Now that we know our input signal and have an experimental setup, we first test to check if our receiver prototype is able to sense a simple OOK signal transmitted by the femtosatellite prototype. We call this the initial reception test. We did this by monitoring the FFT Sink plot in GNU Radio. The following FFT plots were obtained as a result of the initial reception test:



(a) FFT Plot of transmitted 0 (OFF)



(b) FFT Plot of transmitted 1 (ON)

Figure 5.2. FFT Plots of Initial Reception Test.

When a 1 was transmitted from the femtosatellite prototype, the green TX LED on the Arduino blinked and there was a simultaneous peak at 433 MHz as shown in Figure 5.2. Similarly when a 0 was transmitted from the femtosatellite prototype, the FFT plot appeared as shown in Figure 5.2. Note that this plot should ideally be a straight horizontal line. The fluctuations seen in the plot are a result of background noise.

Thus we now know that we are accurately receiving pulses on our receiver ground station prototype as they are transmitted by the femtosatellite prototype. We will now move on to extracting data from the binary file generated by the File Sink block in GNU Radio.

5.3 Data Extraction in MATLAB

We collected several samples of signals received from the femtosatellite prototype. These samples were recorded into binary files by the File Sink block in GNU Radio. We then extracted data from these binary files in MATLAB using the process described in section 4.1. The following plots were obtained from one of the samples received. The sample taken was of an OOK signal which had the following ON–OFF pattern: OFF (1000 ms)–ON (100 ms)–OFF (1000 ms). The duty cycle of one signal period is 9.091%. There is one transmitted ON pulse.

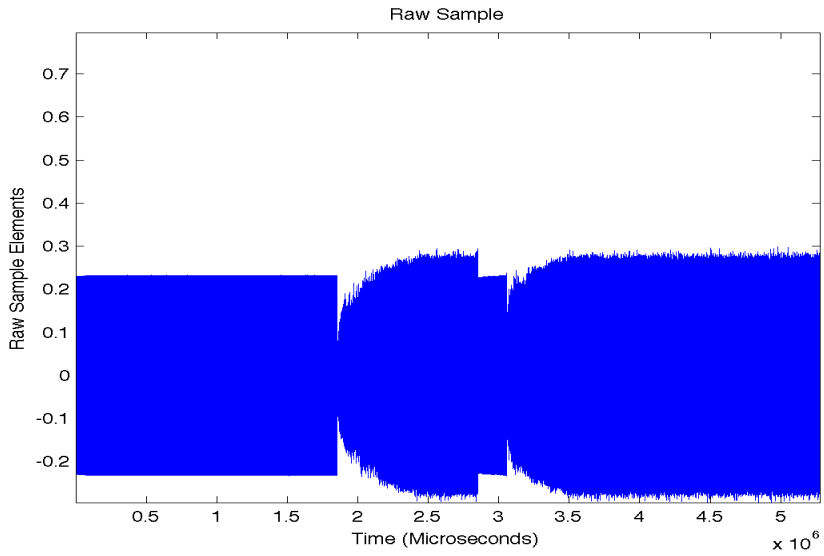


Figure 5.3. Raw Sample.

All elements of the entire raw sample as obtained from GNU Radio's binary file have been plotted against time in Figure 5.3. We will discuss the variations seen in this plot in detail in the magnitude plot 5.7.

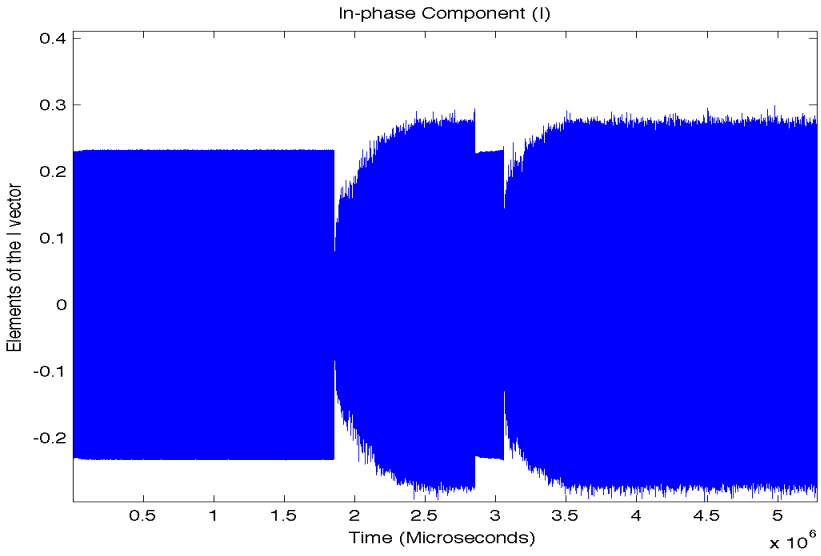


Figure 5.4. In–Phase Component of Signal Sample.

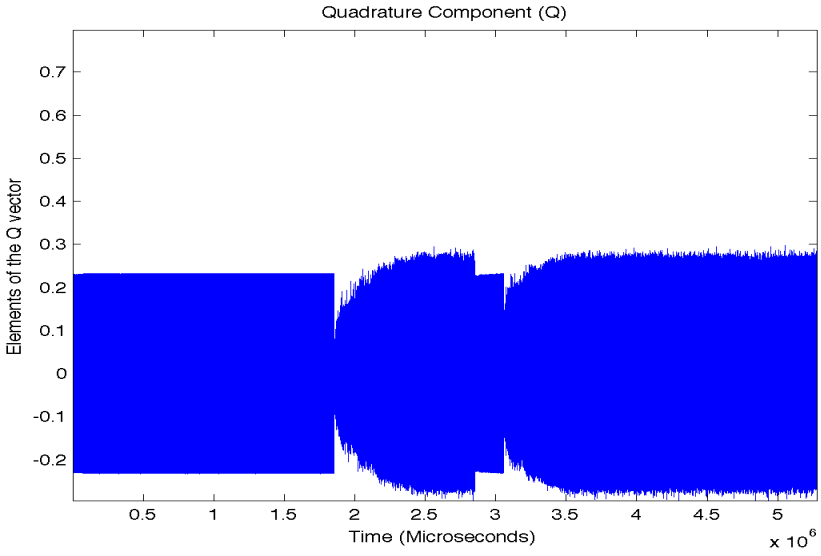


Figure 5.5. Quadrature Component of Signal Sample.

As described in section 4.2.2, Is and Qs are extracted from the raw sample and plotted against time as seen in Figures 5.4 and 5.5 respectively.

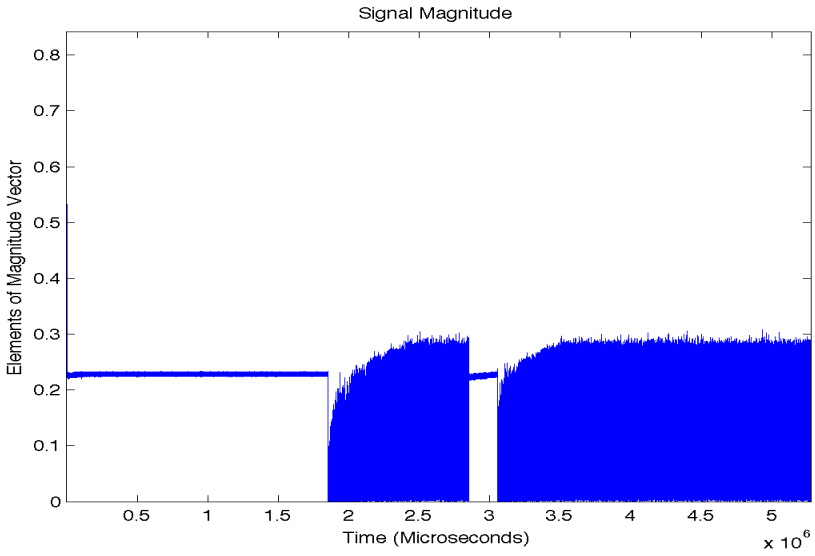


Figure 5.6. Magnitude of Signal Sample.

As described in section 4.2.3, magnitude of the signal sample is computed and plotted against time as seen in Figure 5.6. We have interpreted the details of the variations seen in the magnitude plot. Our interpretations are described below.

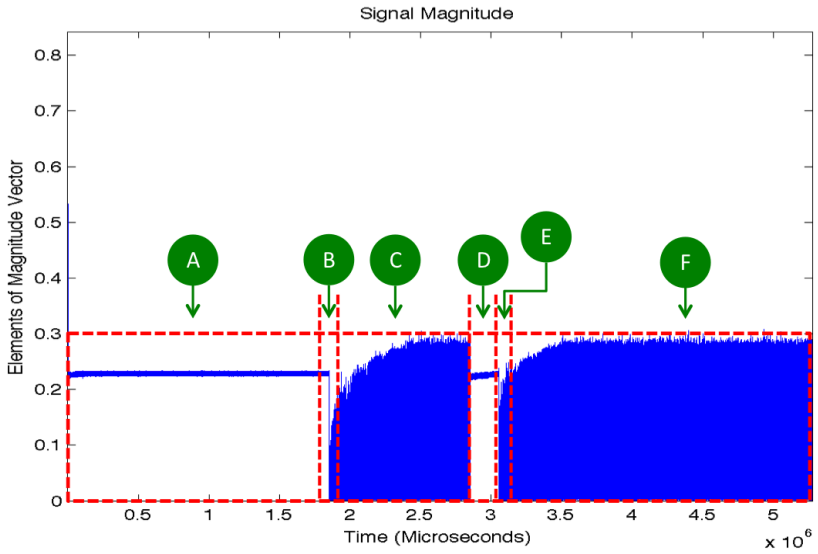


Figure 5.7. Magnitude of Signal Sample with Marked Out Components.

As seen in above Figure 5.7, different parts of the sample have been marked as A, B, C, D, E and F. While taking the samples, the receiver remains active as the femtosatellite prototype is being powered. When the Arduino is powered on, a pulse is transmitted. We will call it the *initial pulse*. The result of the initial pulse transmission is seen in part A of the plot. Part B reflects the point when the signal is driven low for 1000 ms. As seen in part C, when the signal is driven low and kept low for a certain duration (1000 ms in this case), noise begins to grow up to a point where it even exceeds the transmitted high pulse. This is the effect of the AGC as discussed in 3.2.3.1. Part D is the transmitted ON (100 ms) pulse. Part D reinforces the fact that an initial pulse is transmitted when the Arduino is powered on as seen in part A. Part E shows te point when the signal is driven low again for 1000 ms. Once again, the AGC takes over and the noise grows extending beyond the transmitted pulse in part F, similar to part C.

Finally, we compute the phase of the signal sample against time using the technique discussed in section 4.2.3. Due the large number of elements in the entire sample, it is not possible to clearly see the details of the phase plot. Thus we have extracted a small part of the phase vector below.

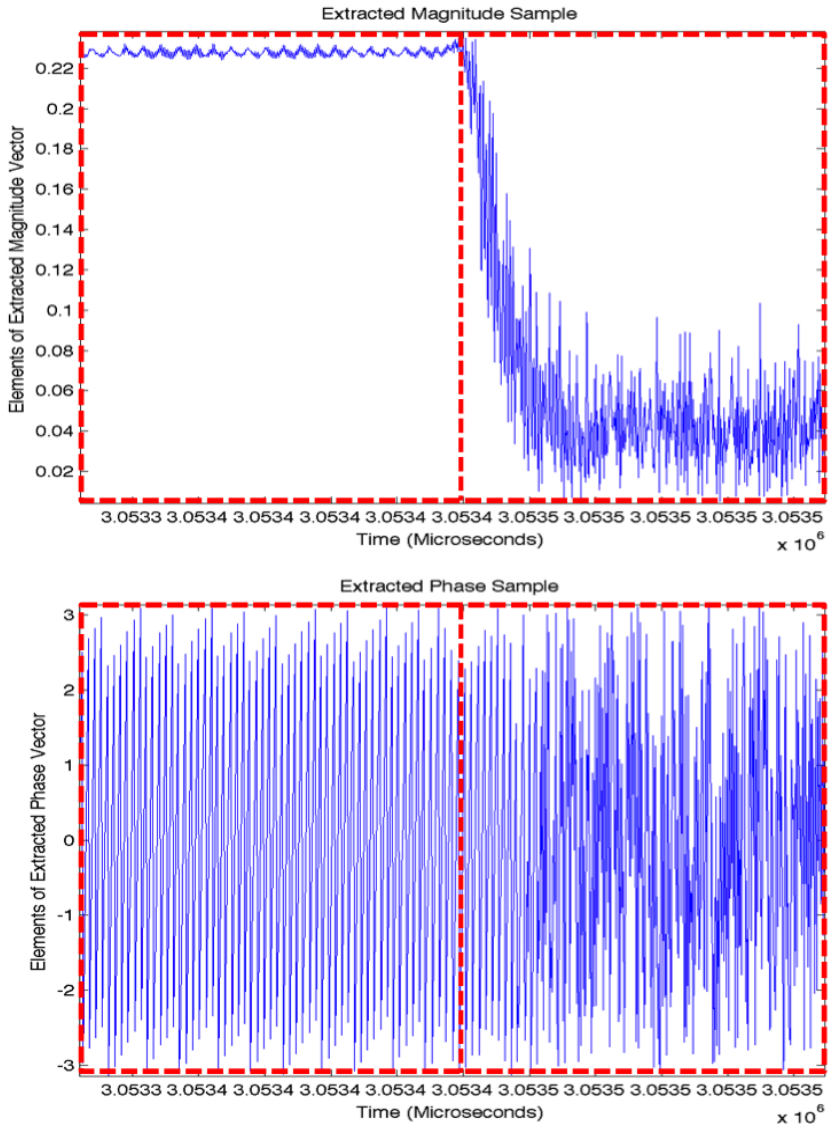


Figure 5.8. Magnitude and Phase Plots of Signal Sample.

The top plot in Figure 5.8 is an extracted magnitude plot focusing on the part where the transmitted signal turns ON followed by an OFF. This is part D and E from Figure 5.7. The bottom plot is the extracted phase vector corresponding to the elements of the extracted magnitude vector. In both Figures, the red blocks to the left are the magnitude and corresponding phase when the transmitted signal is ON and the red blocks to the right are for when the transmitted signal is OFF. Although, not a perfect phase roll, we can clearly see a repeating pattern in the phase plot. This pattern can be further investigated to find the cause behind it and the parameter that makes it repeat every few microseconds. This has been left out of our research from this point on as we need to focus on the magnitude of the transmitted pulses and their exact location in time for the fulfillment of the objectives of this research.

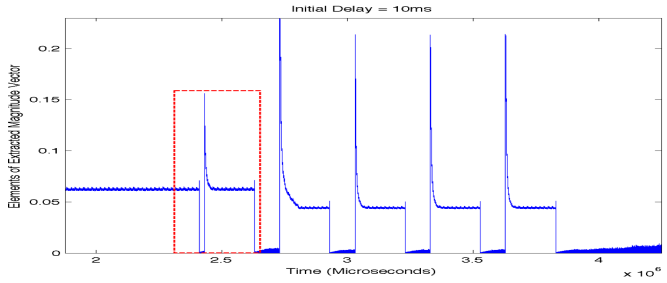
We have learned two important things from the first data extraction test. Firstly, the duration for which the signal is driven low (OFF) should be as low as possible. Secondly, the samples have demonstrated that the initial pulse has an affect on the initial part of the actual transmitted pulse. This could distort our transmitted signal. Hence we need to provide a suitable delay to separate the transmitted signal from the initial pulse. We call this separation the *initial delay*. In the next section, we discuss the process of defining a suitable initial delay.

Please note as mentioned earlier in this section, we are only concerned with the magnitude of the transmitted pulses and their location in time. Even though the raw sample, the Is and the Qs are important in the computation of the magnitude vector, their plots by themselves are not relevant to our experiment. Thus, from here on, we will only discuss plots of the magnitude vector. Additionally, Figure 5.7 shows the magnitude plot of the entire raw sample. As seen in this Figure, the parts of the sample before and after the actual transmitted pulse hold no significance to the rest of our experiment. Thus, from here on, the magnitude plots included in this chapter

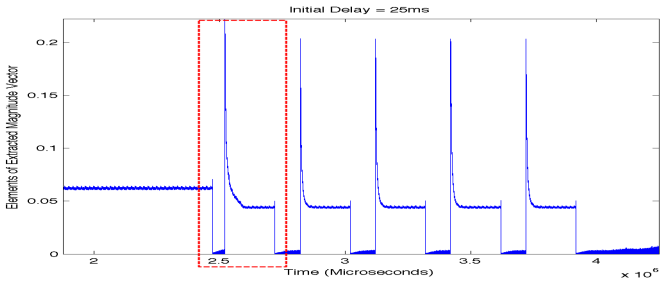
will be plots of the magnitude vector extracted to focus on the actual transmitted pulses. Hence, time on the x axis will be in correspondence with the extracted sample elements.

5.4 Initial Delay Test

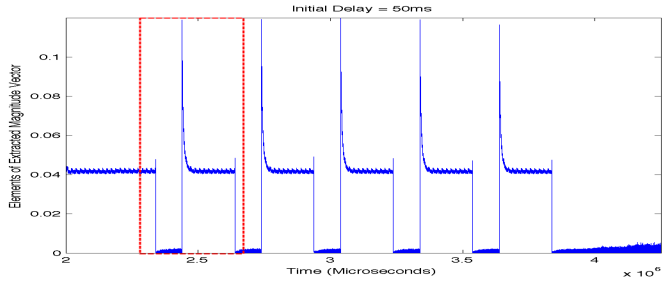
In this test, we have taken samples of the signal with an initial delay of 10 ms, 25 ms, 50 ms and 100 ms and plotted the signal magnitudes against time. The goal is to choose an initial delay that will cause the least distortion to the initial transmitted pulse. At the same time we do not want to use an extremely large initial delay as that would add significantly to the file size which is already very large as 8 million samples are being recorded per second (8MS/s).



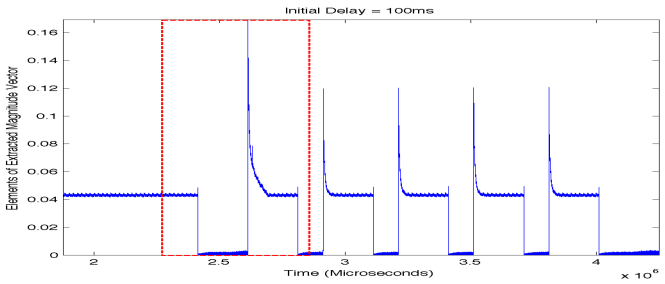
(a)



(b)



(c)



(d)

Figure 5.9. Magnitude Plots of the Initial Delay Test.

The transmitted OOK signal has 5 periods of 100 ms ON and 50 ms OFF i.e. a duty cycle of 66.67%. As seen in the red box in Figure 5.9 (a), at an initial delay of 10 ms, the first transmitted pulse magnitude is affected. Similarly in Figure 5.9 (b) at an initial delay of 25 ms, the first transmitted pulse if affected but not as much as in the case of 10 ms. At an initial delay of 50 ms (Figure 5.9 (c)), the affect of the initial delay on the first transmitted pulse is minimum. We can say this by examining the remaing 4 pulses of that sample. They are almost identical to the first pulse. Finally at an initial delay of 100 ms in Figure 5.9 (d), there is an affect of initial delay on the first transmitted pulse but it is not better than the case of initial delay of 50 ms. As mentioned earlier, we do not want to unnecessarily increase the duration of the signal and thus the file size. Hence, at we have chosen an initial delay of 50 ms to be provided before transmitting the actual OOK signal as it provides minimum distortion to the first transmitted pulse and does not significantly increase file size.

Now that we have established our suitable initial delay before transmitting our OOK signal, the next step is to collect several samples of the transmitted OOK signal and find the best reception sensitivity that our receiver ground station prototype can demonstrate.

5.5 Reception Sensitivity Test

In this test, we took several samples of the transmitted OOK signal having pulse durations in the microsecond range. Starting with a 500 μs ON–100 μs OFF signal, we kept decreasing the pulse duration to achieve our goal of 1 μs reception sensitivity. The following are the plots of the extracted magnitude vector plotted against time for each incremental step towards 1 μs reception sensitivity.

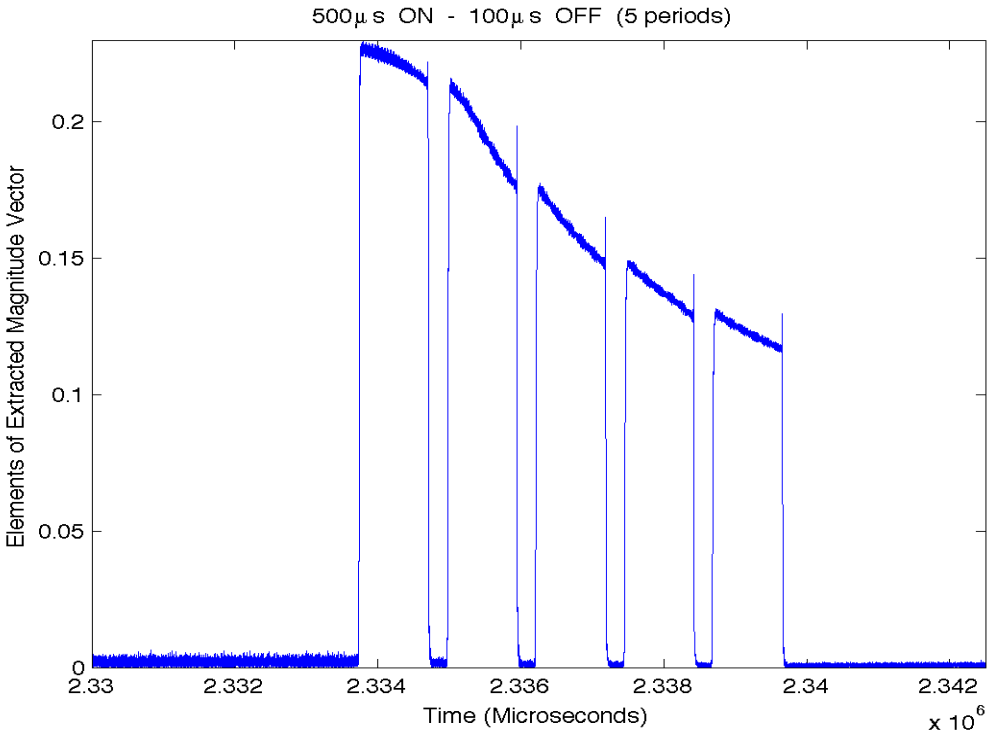


Figure 5.10. Magnitude Plot of The 500 μs ON – 100 μs OFF Signal .

In the case of Figure 5.10, the transmitted OOK signal was a 500 μs ON – 100 μs OFF signal repeating 5 times (5 periods). Thus the duty cycle is 83.33%. The 5 received pulses are clearly visible in the extracted magnitude plot of the signal 5.10. Note that the magnitude decreases with each received pulse which is also an effect of AGC. This effect will be discussed in the next chapter.

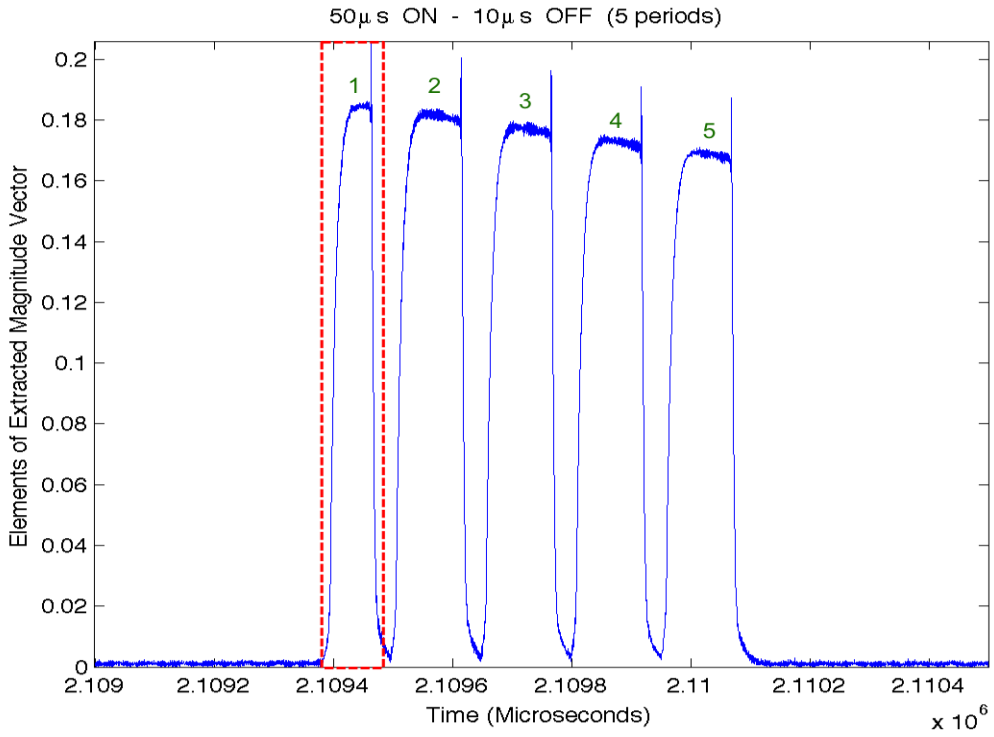


Figure 5.11. Magnitude Plot of The 50 μs ON – 10 μs OFF Signal .

We now decrease our pulse duration to 50 μs. The transmitted signal in case of Figure 5.11 was a 50 μs ON – 10 mus OFF signal having 5 periods. Thus the duty cycle is the same as the previous case i.e. 83.33%. As seen in the dotted red box, at this pulse duration the first received pulse is distorted. The remaining four pulses, although seem to be decreasing (due to AGC) are intact.

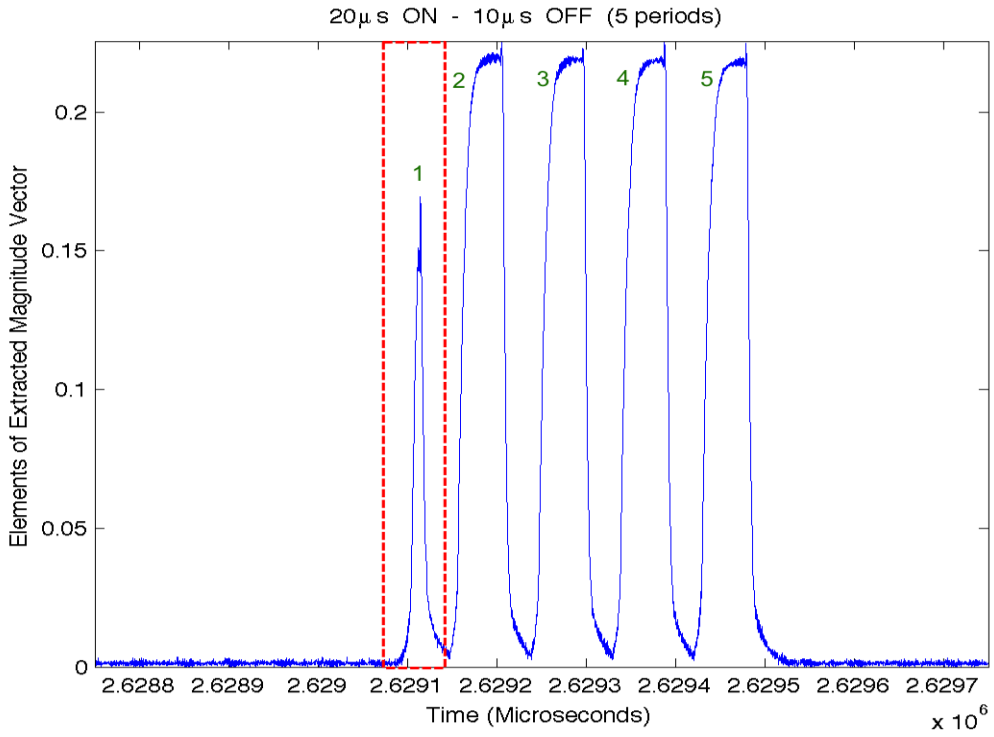


Figure 5.12. Magnitude Plot of The 20 μs ON – 10 μs OFF Signal .

We now decrease the pulse duration to 20 μs. The transmitted signal in case of Figure 5.12 was a 20 μs ON – 10 μs OFF signal having 5 periods. Thus the duty cycle is 66.67%. As seen in the dotted red box, at this pulse duration, the first received pulse is even more distorted compared to the previous 50 μs pulses. The remaining four pulses are still intact.

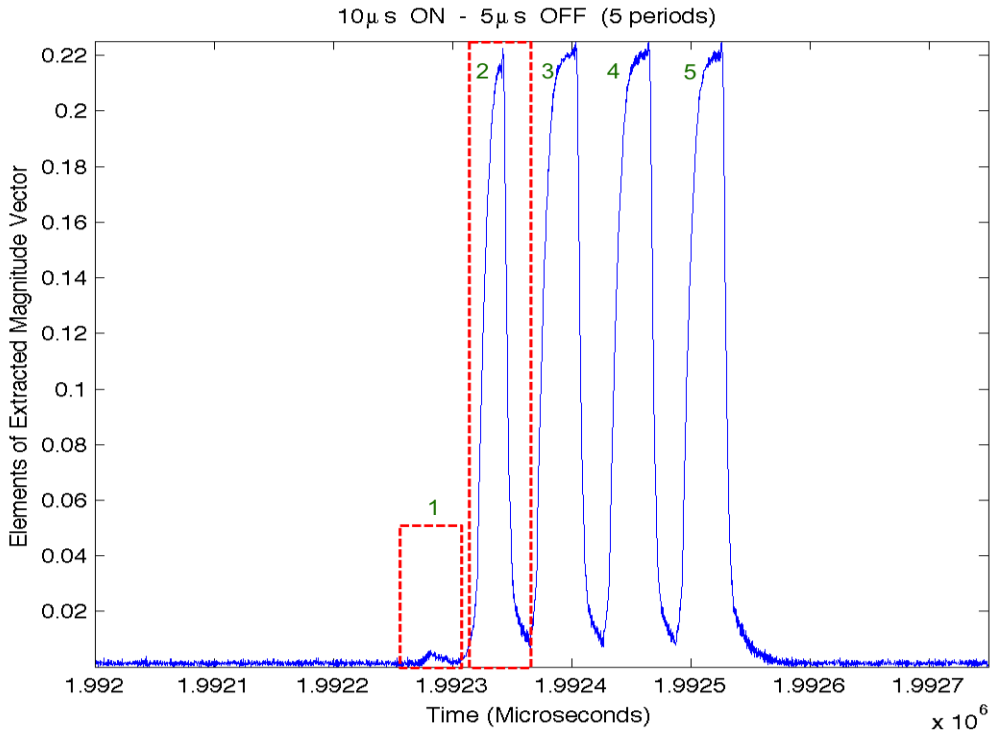


Figure 5.13. Magnitude Plot of The 10 μs ON – 5 μs OFF Signal .

The transmitted signal in case of Figure 5.13 was a 10 μs ON – 5 μs OFF signal having 5 periods. Thus the duty cycle is 66.67%. As seen in the dotted red boxes, at this pulse duration, the first received pulse is almost lost. The second received pulse is distorted. The remaining three received pulses are intact.

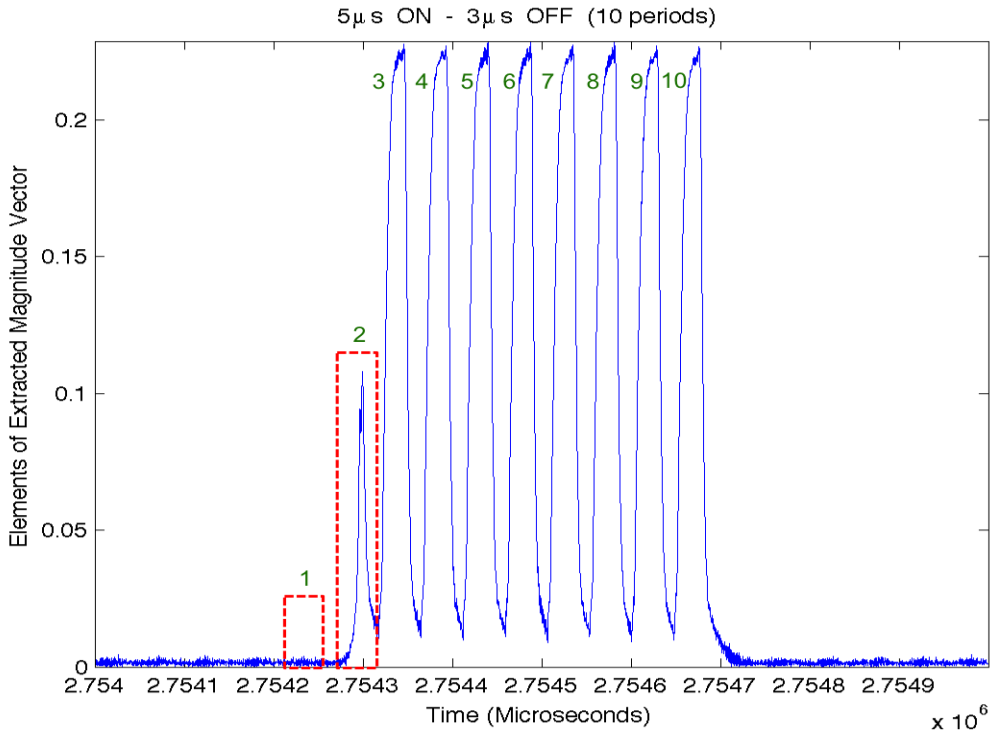


Figure 5.14. Magnitude Plot of The 5 μs ON – 3 μs OFF Signal .

The transmitted signal in case of Figure 5.14 was a 5 μs ON – 3 μs OFF signal having 10 periods. Thus the duty cycle is 62.5%. As seen in the dotted red boxes, at this pulse duration, the first received pulse is completely lost. The second received pulse is more distorted compared to the previous case of 20 μs pulses. The remaining eight received pulses are intact.

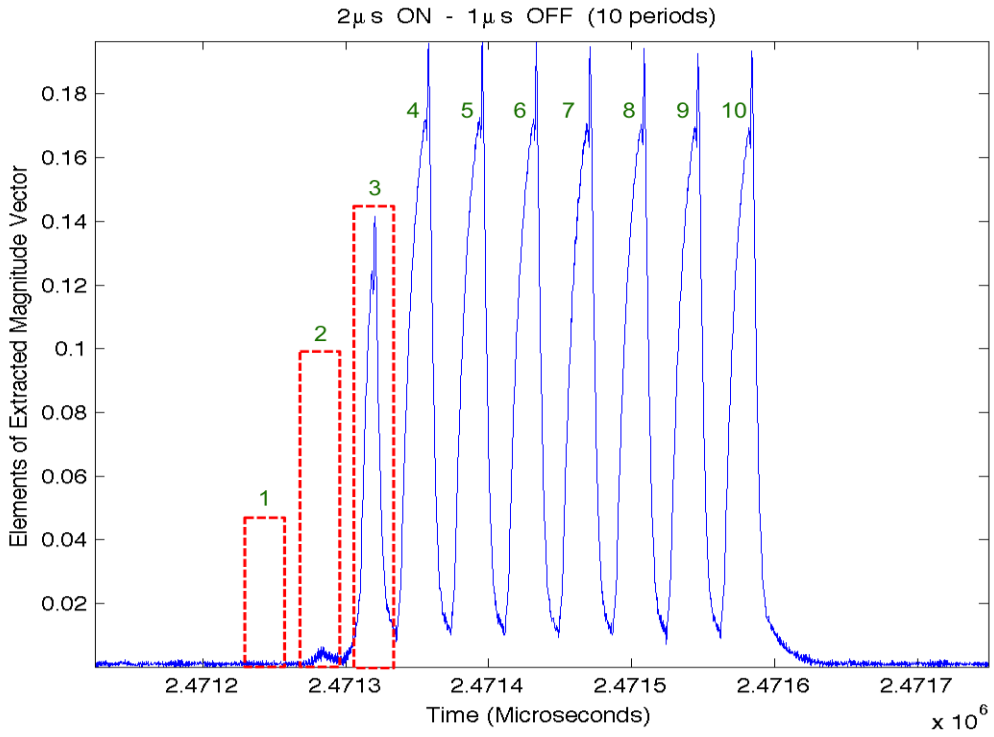


Figure 5.15. Magnitude Plot of The 2 μs ON – 1 μs OFF Signal .

We finally decreased the pulse duration to 2 μs. The transmitted signal in case of Figure 5.15 was a 2 μs ON – 1 μs OFF signal having 10 periods. Thus the duty cycle is 66.67%. As seen in the dotted red boxes, at this pulse duration, the first received pulse is completely lost. The second received pulse is almost lost. The third received pulse is distorted. The remaining seven received pulses are intact.

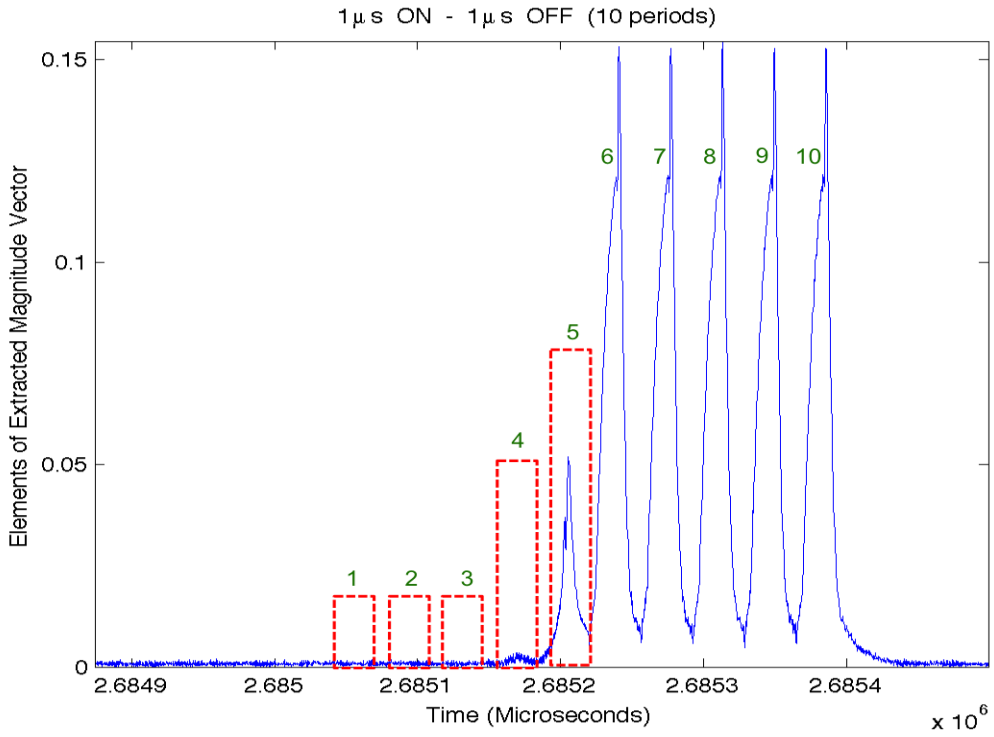


Figure 5.16. Magnitude Plot of The 1 μs ON – 1 μs OFF Signal .

In order to confirm the reception sensitivity of 1 μs, we transmitted a 1 μs ON – 1 μs OFF OOK signal having 10 periods. The duty cycle is 50%. As seen in the dotted red boxes of Figure 5.16, the first three received pulses are completely lost. The fourth received pulse is almost lost and the fifth pulse is distorted. The remaining five received pulses are intact.

As mentioned in the beginning of this chapter, our objective was to achieve a reception sensitivity of 1 μs with our femtosatellite and receiver ground station prototypes. The results obtained in the reception sensitivity test conclusively confirm that we have achieved a reception sensitivity of 1 μs as seen in Figure 5.16. There is a loss of initial received pulses which will be discussed in the next chapter. Equipped

with this reception sensitivity, we will now transmit a PRN–encoded OOK signal via our femtosatellite prototype and process it in our rceiver ground station.

5.6 PRN Sequence Generation and Transmission

We have transmitted the entire 1023–chip PRN 19 sequence from our femtosatellite prototype. This was received on our receiver ground station prototype. The magnitude plot of the entire received 1023–chip PRN sequence in shown in Figure 5.17.

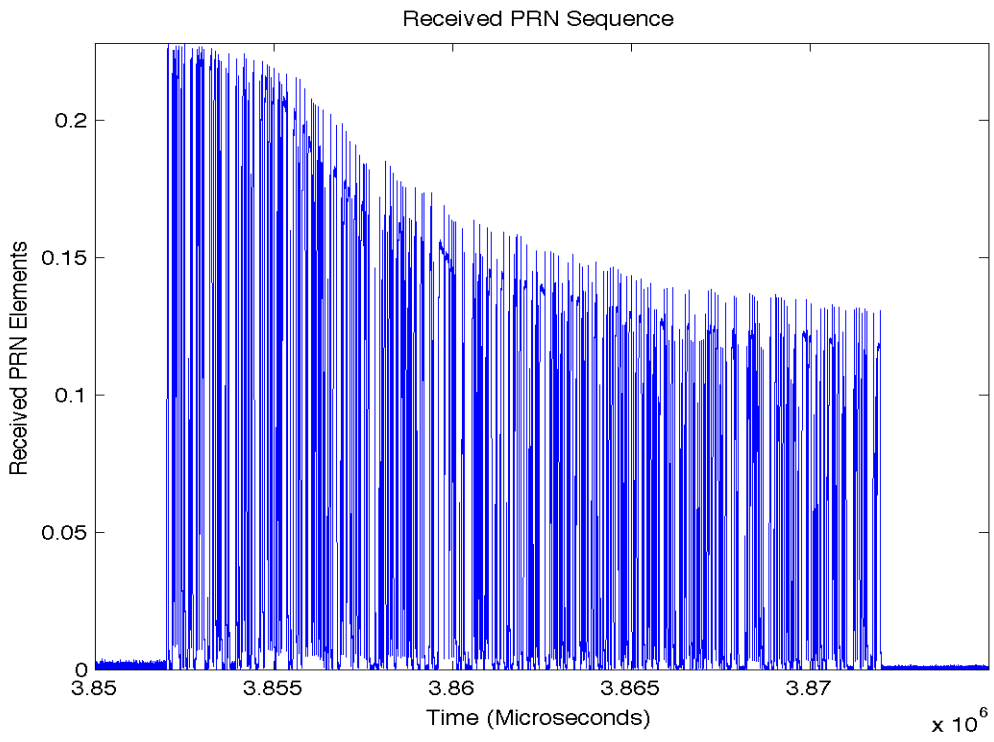


Figure 5.17. Magnitude Plot of the Entire 1023–chips of the PRN 19 Sequence.

As discussed in section 4.2.4, we have used the thresholding technique to extract the PRN sequence from the received data in a sequence of 0s and 1s. We have then

map the extracted PRN sequence from the sequence of 0s and 1s to a sequence of 1s and (-1)s.

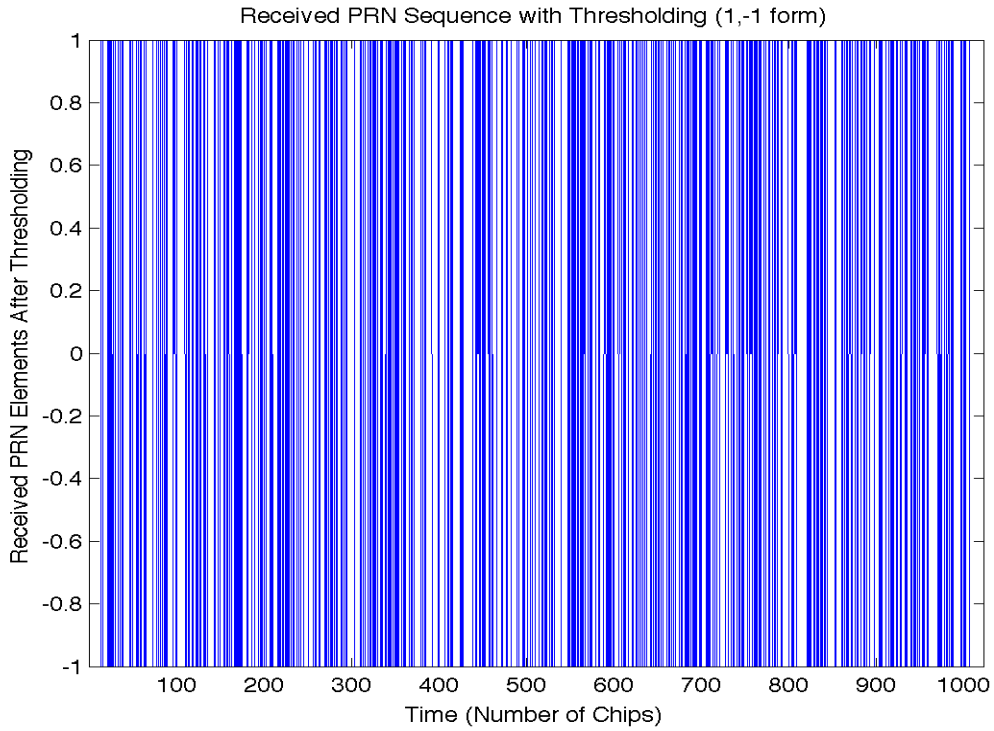


Figure 5.18. Extracted PRN as a sequence of 1s and (-1)s After Thresholding.

As seen in Figure 5.18, we have obtained the received PRN sequence as a sequence of 1s and (-1)s . We can now use this extracted PRN sequence to perform autocorrelation with a replica PRN sequence. The 1023–chip replica PRN sequence of 0s and 1s is generated using the algorithm described in section 2.2. The plot of the 1023–chip replica PRN sequence is shown in Figure 5.19 below.

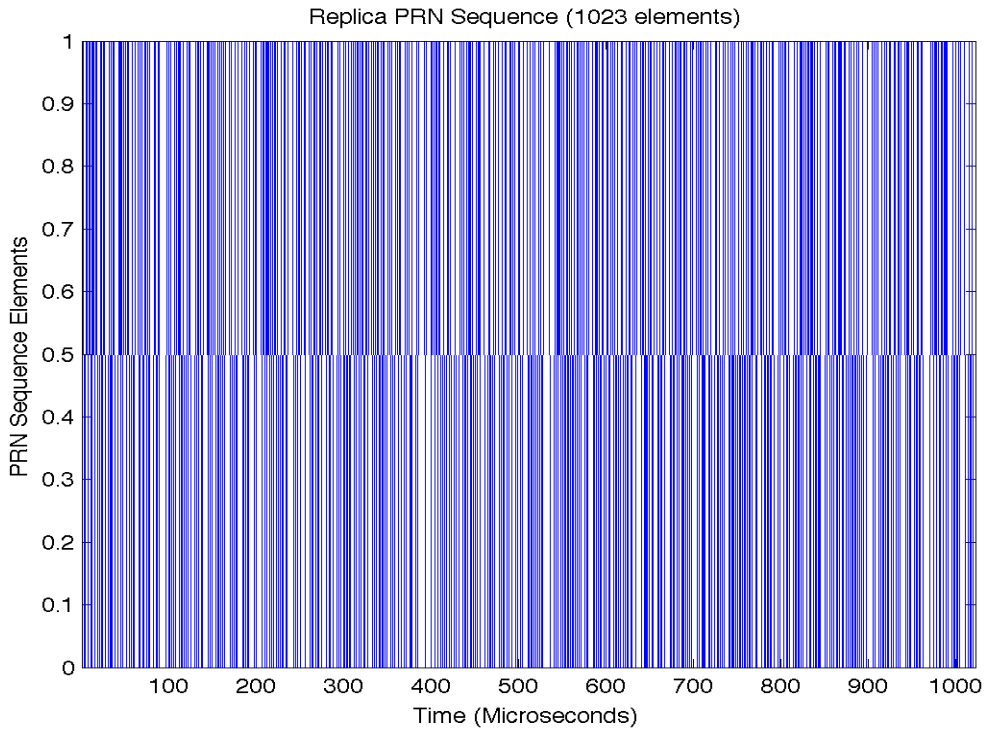


Figure 5.19. 1023–chip Replica PRN Sequence.

The 1023–chip replica PRN sequence is then super–sampled and mapped into a sequence of 1s and (-1)s as discussed in section 4.2.5. Thus we now have both the received PRN sequence and the replica PRN as sequences of 1s and (-1)s. We will now proceed to perform autocorrelation on the two PRN sequences.

5.7 Autocorrelation

First, we have performed autocorrelation of the replica PRN sequence with a bit-shifted version of itself. The bit–shifted version is shifted by 5 μs (4000 samples). This autocorrelation represents the ideal autocorrelation of the received and replica PRN sequences. We will use this ideal autocorrelation as measure to verify our actual

correlation between the received and replica PRN sequences. The autocorrelation of the replica PRN sequence with a bit–shifted version (shifted by 5 μs) is shown below.

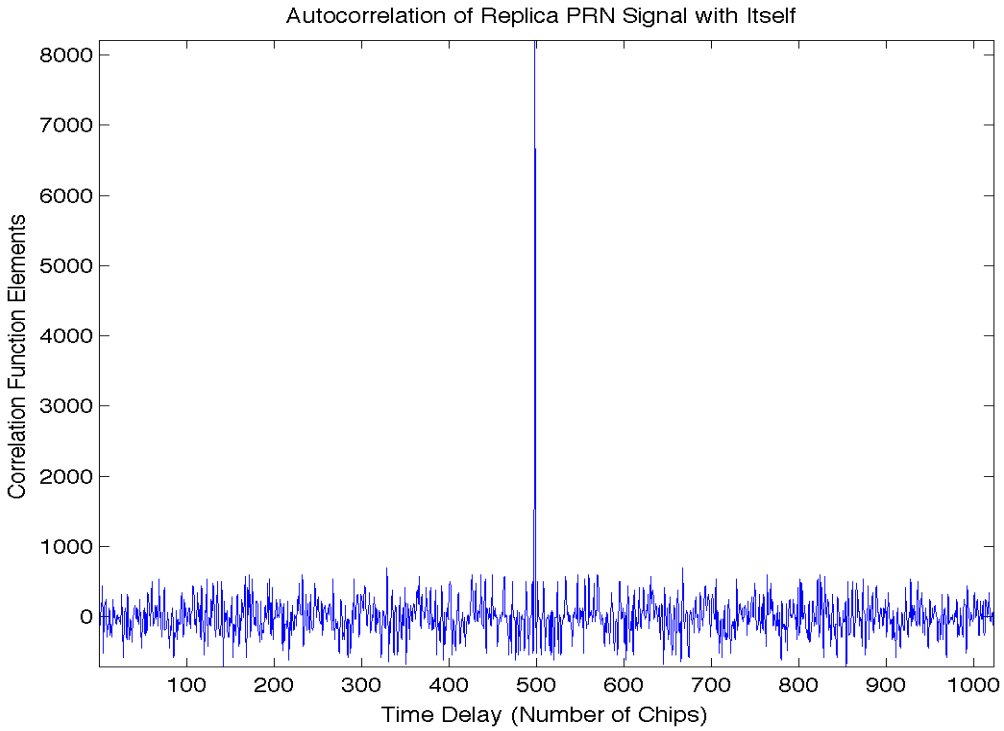


Figure 5.20. Autocorrelation of the Replica PRN Sequence with a Bit–shifted Version of Itself.

As seen in the Figure 5.20, we see a correlation spike as the two autocorrelated PRN sequences are exactly the same. Note that the peaks of the autocorrelation function before and after the correlation spike are not zero but close to zero and have a fixed pattern. Also, note that this autocorrelation was performed in the absence of noise. Next, we have performed autocorrelation of the received PRN sequence with the replica PRN sequence. This autocorrelation is shown in the plot below.

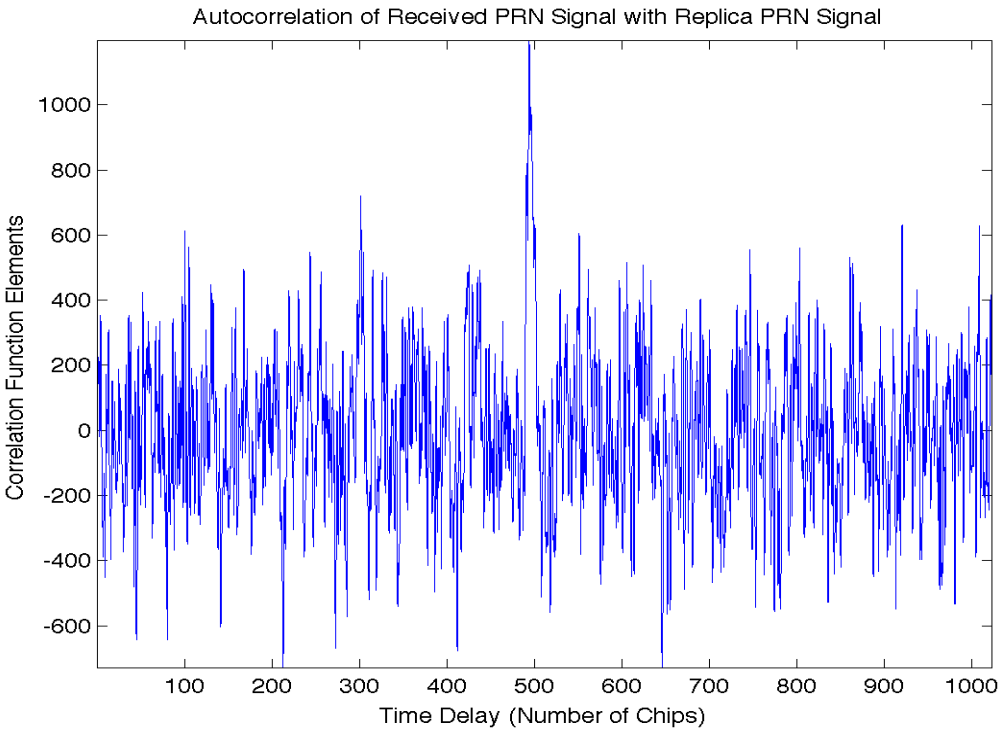


Figure 5.21. Autocorrelation of the Received PRN Sequence with the Replica PRN Sequence.

As seen in Figure 5.21, there is a clear correlation spike as there is a match between the received and replica PRN sequences. Just like the autocorrelation curve of the replica PRN sequence with itself, the peaks of the correlation in Figure 5.23 also have a fixed pattern and are not random. Note that this autocorrelation was performed in the presence of noise. Also note that we have performed autocorrelation of only one set of 1023–chips. We could achieve a correlation spike of a larger magnitude if we correlate multiple 1023–bit samples. We have compared the two autocorrelations plotted in Figures 5.20 and 5.21. The comparison is shown below.

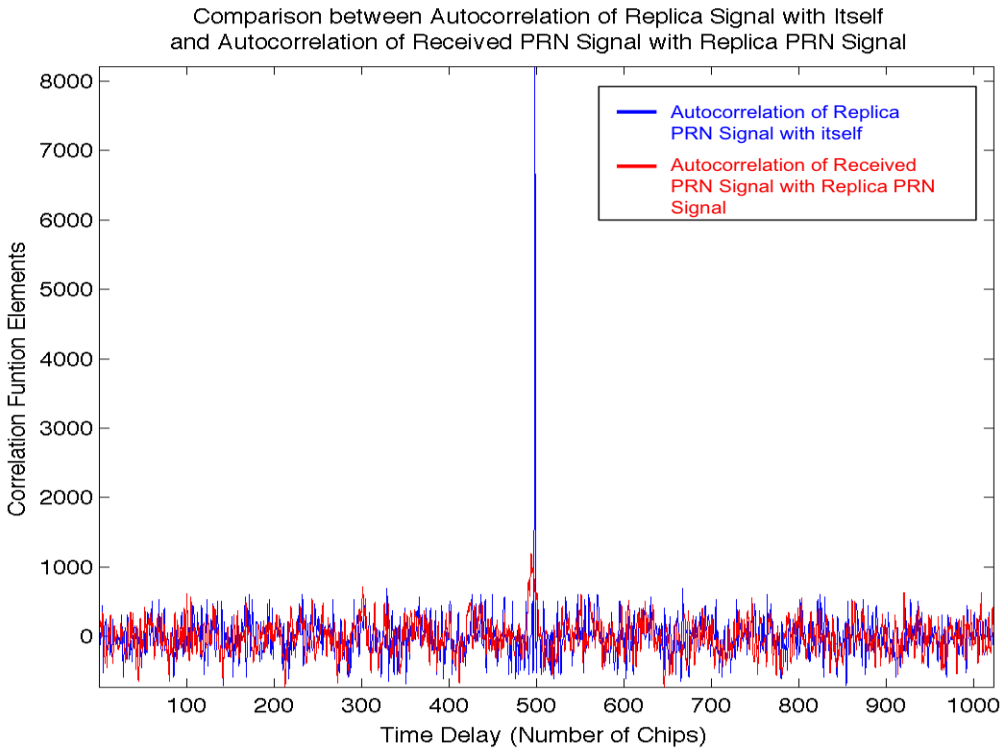


Figure 5.22. Comparison between Autocorrelation of Replica PRN with Itself and Autocorrelation of Received PRN Sequence with Replica PRN Sequence.

Figure 5.22 shows that the two autocorrelations occur at almost the same point in time. There is an error of 1–5 μs between the two correlation spikes which will be discussed in the next chapter. Thus we have achieved our second goal by autocorrelating the received PRN signal with the replica signal and successfully demonstrated a correlation spike indicating a match of PRN sequences.

This chapter has summarized all the intermediate and final results obtained in our experiment. We need to now analyze these results and draw inferences from them in the next chapter.

CHAPTER 6

Analysis and Discussion

In this chapter, we will first perform a system identification on our femtosatellite–ground staion prototype system. We will then discuss system behavior based on the system identification and the results obtained in Chapter 5. Finally, we will conclude with our research findings.

6.1 System Identification

As discussed in the preceding chapters, we have built our own satellite tracking system which consists of the femtosatellite prototype and the receiver ground station prototype. System identification is a process wherein we identify the characteristics of the concerned system and develop a mathematical model based on these characteristics. The goal is to develop a mathematic model that emulates the actual system as closely as possible. The model should be such that given the output of the system, we can estimate the input which in our case is the signal received from the femtostellite. This is done because, in a real situation, we will not know the exact nature of the signal being received from the femtosatellite. The system model can help us estimate the nature of the signal transmitted by the femtosatellite. Note that in the detailed explanation of the system identification process, we will use several time response terms which are defined in Appendix A.

In our analysis, we have modeled our system as a linear system. We then move on to perform a transient response analysis of the system. In the transient response analysis process, we have obtained the step response of our system by transmitting

a pulse (step) from the femtosatellite prototype and plotting its magnitude on the receiving end. The plot of the step response of our system is shown in Figure 6.1.

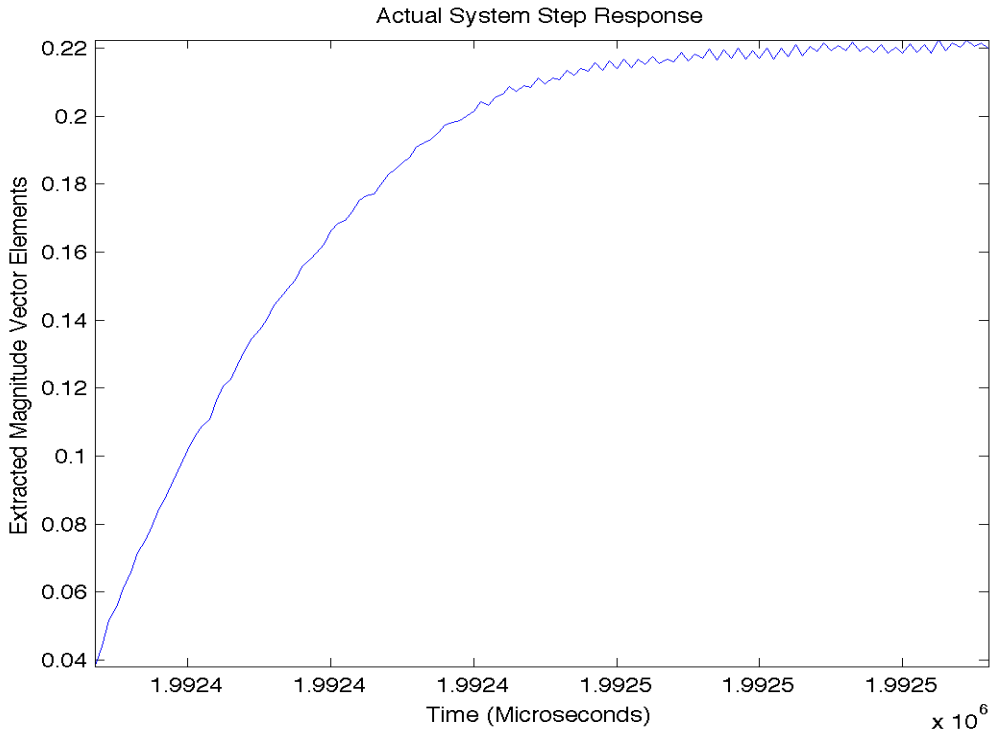


Figure 6.1. Step Response of Actual System.

First, we tested to see if our system is a second order system. We computed the significant transient response characteristics of the system namely the *rise time* t_r, *peak time* t_p, *delay time* t_d *and the percentage overshoot* M_p from the step response curve in Figure 6.1. We repeated this process for several step response samples and obtained average values of t_r, t_p, t_d, M_p over all the samples taken. We then computed the damping ratio ζ and the natural frequency ω_n of the system using their relationship with the transient response characteristics obtained [19]. Using the values of ζ and ω_n, we construct the system's transfer function. A transfer function is the

relationship between the input and output of the system with zero initial conditions. For a second order system, it takes the following form:

$$G(s) = \frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{6.1}$$

where C(s) is the system output and R(s) is the system's reference input.

Based on equation (6.1) and the values of ζ and ω_n, we obtained system transfer functions for the under damped ($\zeta < 1$), critically damped ($\zeta = 1$) and the over damped ($\zeta > 1$) cases. The step response curves obtained in all of these cases *did not* match the actual step response curve as seen in Figure 6.1. Thus we moved on to model the system as a first order system. The transfer function for a first order system is given by equation (6.2).

$$G(s) = \frac{C(s)}{R(s)} = \frac{G_{DC}}{s\tau + 1} \tag{6.2}$$

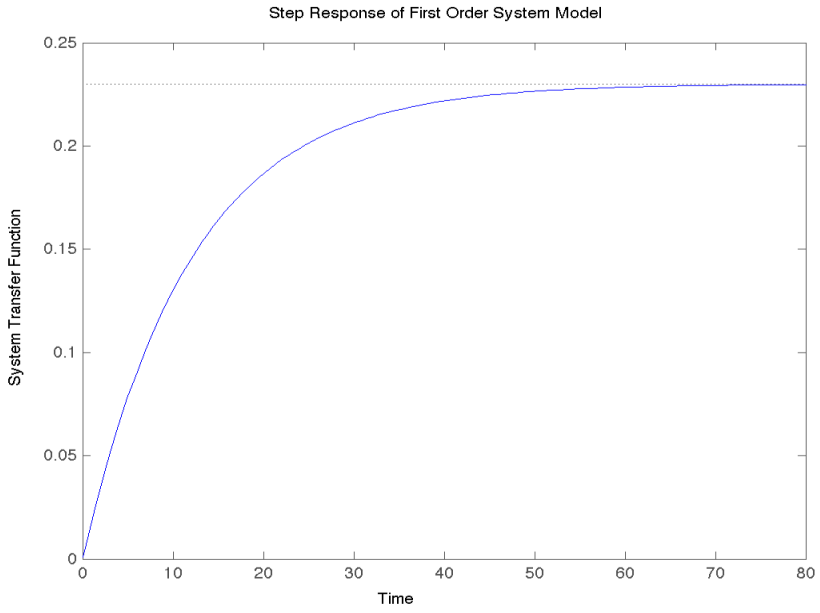
In equation (6.2), G_{DC} is the DC gain of the system and τ is the time constant of the system. G_{DC} is calculated as a ratio of the output voltage to the input voltage. In our case it is calculated as follows:

$$G_{DC} = \frac{0.23 \text{ volts}}{5 \text{ volts}} = 0.046 \tag{6.3}$$

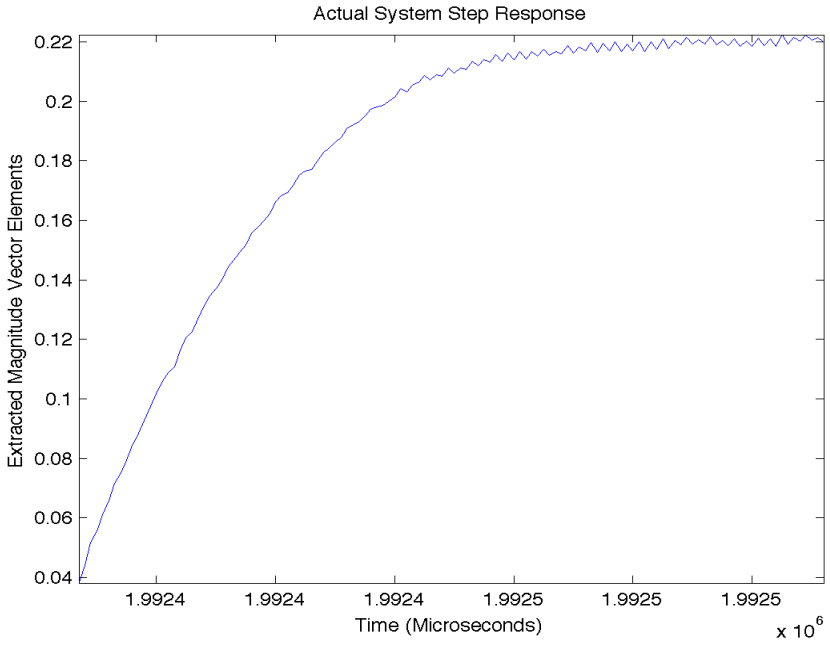
The time constant of the system, τ is the time required for the system response to reach 63% of its final value [19]. For step response curve for a transmitted step of a duration of 10 μs, we have found that the $\tau = 12\,\mu$s. Plugging the values of G_{DC} and τ in equation (6.2), we get the following first order transfer function:

$$G(s) = \frac{C(s)}{R(s)} = \frac{0.046}{12s + 1} \tag{6.4}$$

The step response curve of the first order transfer function in (6.4) has been plotted and compared with the actual system step response below.



(a) Step Response Curve of First Order System Model



(b) Step Response Curve of Actual System

Figure 6.2. Comparison of Step Response Curves of the First Order Model and the Actual System.

As seen in Figure 6.2 the first order model's step response curve is very close to that of the actual system. Thus we have obtained a good estimate model for our system In order to test this model for different pulse durations, we plotted our system's step response curve for pulses of different durations. We then calculated τ (using the 63% technique) for all of these curves to obtain an average τ. The values of τ obtained for different pulse durations are listed in table 6.1.

Table 6.1. τ Values for Different Pulse Durations

Pulse Duration (microseconds)	Time Constant (microseconds)
50	16
20	15
10	12
5	9
2	8

From table 6.1, we calculated the average $\tau = 12$ μs which in in agreement with the τ we used in our transfer function computation in equation (6.3). From table 6.1, we have deduced that the duration of the transmitted pulse has a direct affect on the time constant of the system. This deduction will be discussed in Section 6.3.

Thus from our system identification process, we have concluded that our system can be treated as a first order system for a given, repeating pulse train. In the next section, we will use this system identification and the results from Chapter 5 to assess our system's behavior.

6.2 System Behavior

We have made several observations about our system's behavior which include the effects of heat and the Automatic Gain Control. We will discuss these observations in the subsections that follow.

6.2.1 Automatic Gain Control (AGC)

As discussed in section 3.2.3.1, the USRP Daughterboard we use has an always–on AGC. Irrespective of its advantages, this always-on AGC feature has an adverse effect of the received signal. As we cannot turn off AGC, we forfeit the purity of the received signal. An incorrectly designed AGC system can introduce considerable distortion to an otherwise clean signal [8]. In our experiment, there are visible adverse effects caused by AGC. As seen in Figure 5.3, when we drive the transmitted signal low, it does go to zero but it does not stay at zero. Instead, after the signal reaches zero, unwanted noise amplifications are introduced and they gradually increase to the point that they exceed even a transmitted high or ON pulse of the received signal. This is the effect of the AGC. It is because of this effect that we have tried to keep our transmitted signal low for as small a duration as possible.

Similarly, as seen in Figure 5.10, when we transmit multiple ON–OFF pulses, the magnitude of these pulses decreases over time. This is also an effect of the AGC. It is because of this effect that we have to use a filter (thresholding in our case) to balance the received signal and accurately extract a PRN sequence as seen in Figures 5.17 and 5.18.

Moreover, we have observed that the accuracy of the received pulses increases as the pulse duration or chip size decreases. This is because the AGC is unable to detect pulses in the μs range. This further confirms that our signal is being adversely affected by AGC.

The AGC affects our benchtop experiment because there is a strong carrier signal that can be easily detected by the AGC. A femtosatellite signal transmitted from orbit as sampled by a ground station will not have such as a strong carrier, in fact it will be below the ambient noise. This condition is referred to as subthermal, a condition that femtosatellite signals share with the GPS satellites. This is why the primary PRN codes are used to modulate the GPS signal, as well as the signals from the Sprite and our prototype. Because we predict the femtosatellite carriers received from orbit will be subthermal, we do not believe the AGC will interfere with observations of deployed femtosatellites.

6.2.2 Warm–up Effect

Our femtosatellite prototype design consists of microelectronics. Microelectronics typically require some time to warm–up before they can be fully functional. For this reason, we have lost the initial pulses of our received signal. This effect can clearly be seen in Figures 5.11–5.16. In fact, when we reach our desired pulse duration of 1 μs in Figure 5.16, we have lost almost five pulses. Nonetheless, the remaining five pulses in Figure 5.16 are intact and exactly identical indicating that the microelectronics have warmed up sufficiently by then. We call this the *Warm–up Effect*.

6.2.3 Cool-down Effect

We refer to cool–down time as the time that the microelectronics on the femtosatellite prototype are allowed to cool down. This occurs when a zero or OFF pulse is transmitted. Higher the number of zeros transmitted, longer is the cool-down time provided to the femtosatellite prototype.

In order to demonstrate the effect of different cool-down times, we have plotted the magnitude of the first 16 chips of the received PRN 19 sequence (Refer Section

2.1). The first 16 chips of the PRN 19 sequence are E6D6 in hexadecimal i.e. 1110 0110 1101 0110 in binary. We have plotted these received 16 binary bits as shown below.

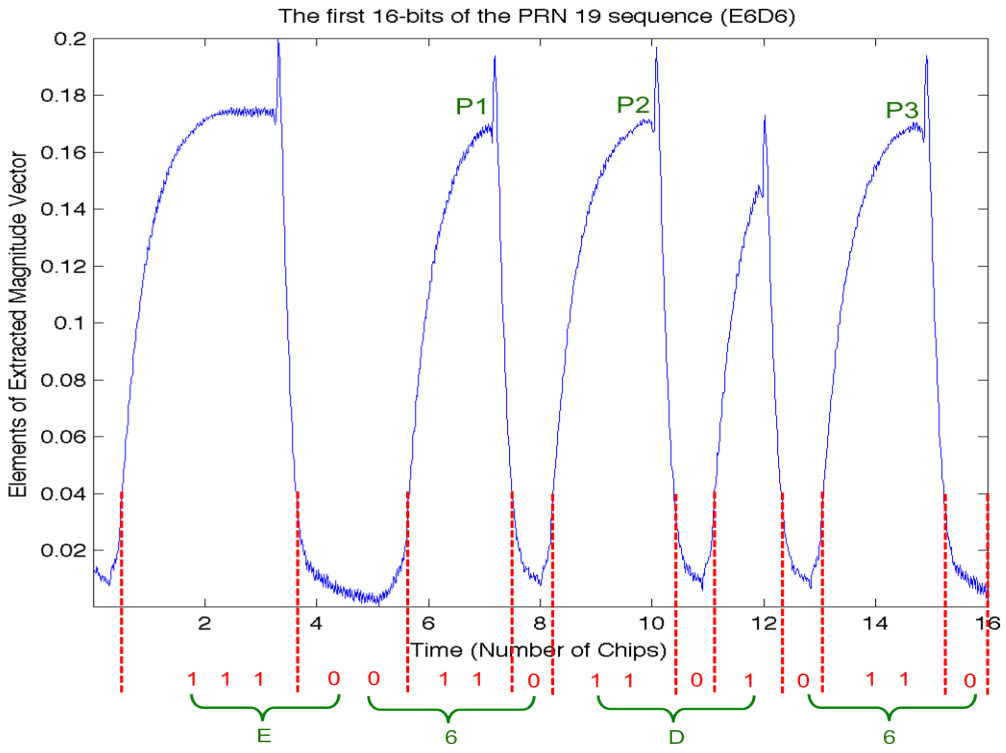


Figure 6.3. First 16 Chips of the Received PRN 19 Sequence.

As seen in Figure 6.3, the first 16 chips of PRN 19 i.e. E6D6 have been accurately received by our ground station prototype. Consider the pulses P1, P2 and P3 in this Figure. They all correspond to the transmission of two ON pulses i.e. 11 in binary. Two zeros (00) were transmitted before P1 whereas one zero (0) was transmitted before both P2 and P3. This means that there was a longer cool-down time before P1 as compared to both P2 and P3. Thus we see clearly that there is difference in the widths of P1 and P2, P3. P2 and P3 have the same width as they

are both preceded by the same cool–down time (0). P3's width is smaller than that of P1 and P2 as it is preceded by a longer cool-down time (00). Hence we have observed that the amount of time that the femtosatellite prototype is allowed to cool down has a direct affect on the width of the transmitted pulses. We call this the *Cool–down Effect*.

6.2.4 Time Constant Characteristics

The time constant (τ) is a measure of the speed of response of a system. We have obtained τ values for different pulse durations in table 6.1. It is evident in this table that the time constant decreases with a decrease in pulse duration. This means that we get a faster system response as we decrease pulse duration or chip size.

We have thus made several observations about our system's behavior that can be used to improve our femtosatellite and ground station prototype designs. Finally we will conclude this chapter by discussing the immediate implications of the results achieved in our experiment.

6.3 Research Findings

Our experiment has been successful in accomplishing both of our set objectives. Firstly, we have achieved a reception sensitivity of 1 μs with our ground station prototype. This translates into a ranging capability approaching that of GPS satellites. This means that we could obtain range measurements from our femtosatellite prototype that can be used to determine positioning information or any application of GPS that involves range measurements.

Secondly, we have successfully demonstrated a match of the received and replica PRN sequences via our autocorrelation plots. As discussed in section 4.1, we can use the location of the autocorrelation spikes in time to measure the distance between

the femtosatellite and the ground station. Also, a match in PRN sequences can be used to confirm the identity of the femtosatellite whose signal is being received. This is possible as each femtosatellite has a unique PRN sequence.

Finally, we have performed a system identification and deduced that our system can be treated as a first order system for a given, repeating pulse train. Further, we have deduced key system behavioral issues like effects of the AGC, the Warm-up Effect, the Cool-down Effect and the influence of pulse durations on the system time constant. This analysis could help improve the design and implementation of our femtosatellite and ground station prototypes.

In the last chapter of our documentation, Chapter 7, we will discuss the conclusions we have made on the basis of our experimental results discussed in Chapter 5 and the analysis made in this chapter.

CHAPTER 7

Conclusions and Future Work

In our final chapter, we will summarize the conclusions of this research and discuss ways to improve the design and methodology of this project.

7.1 Conclusions

In the beginning of this research, we were facing a multitude of challenges. First, we wanted to build our very own femtosatellite prototype with commercially available, inexpensive microelectronics. Second, we wanted to build our own ground station prototype that would be capable of accurately receiving and recording signals from the femtosatellite prototype. Third, we wanted to design a signal that uses the concepts of OOK modulation and PRN encoding. Although, PRN encoding has been extensively used in satellite communication, the OOK modulation technique is not commonly used to modulate satellite signals.

We have made an effort to overcome these challenges with practical and inexpensive solutions. We have built a fully functional femtosatellite prototype and a receiver ground station prototype. Our femtosatellite prototype has transmitted a PRN–encoded, OOK signal which has been received by our ground station prototype with a reception sensitivity of 1 μs which translates into a ranging capability similar to that of GPS satellites. This reception sensitivity has been achieved with inexpensive, commercially available and highly unstable microelectronics. The cost of our entire femtosatellite prototype is under 35 USD. Finally, in order to verify the ability of our ground station prototype to successfully track the femtosatellite prototype and to

obtain positioning and timing information from it, we performed an autocorrelation of the received PRN signal with a replica PRN sequence. The autocorrelation plots thus obtained have demonstrated the capability of our ground station to identify the femtosatellite by its PRN sequence and obtain positioning information from it.

In summary, with this research, my contribution is that I have built a fully functional femtosatellite and ground station prototype. I have designed the structure of the signal that is transmitted by the femtosatellite prototype and received by the ground station prototype with a sensitivity of 1 μs. I have obtained a correlation between the signal received from the femtosatellite prototype and the replica signal. This implies that the identity and position of the femtosatellite could be obtained. Thus I have verified that it is possible to communicate with a femtosatellite made of commercially available, inexpensive and unstable microelectronics using the techniques of PRN sequence encoding and OOK modulation. I have established a reference platform for femtosatellite experimentation. In my analysis, I have performed a system identification and recognized significant system behavioral issues. I have observed that AGC affects the magnitude of the transmitted signal pulses and amplifies noise. The Warm-up effect causes loss of initial received pulses and the Cool-down effect affects the width of the received pulses. Lastly, I have extracted data that proves that the time constant of the system is a function of the received pulse history. As discussed in chapter 6, there are numerous issues that are yet to be addressed. In the next section, we will discuss the scope for improving our experiment and methodology.

7.2 Future Work

In Chapter 6, we have listed some significant behavioral issues of our system. In this section, we will discuss possible ways to deal with these issues and improve our

system's performance. We have discussed the adverse effects of AGC on our received signal in section 6.2.1. This issue would be addressed by using an RF peripheral receiver that has the flexibility to control AGC.

In sections 6.2.2 and 6.2.3, we have seen that the Warm-up effect causes a loss of initial pulses of the received signal and the Cool-down effect has a direct affect on the width of the received pulses. The position of the different components like the resistors, capacitors, LEDs on the femtosatellite circuit board could be contributing to the Warm-up and Cool–down effects. As we use a pre–assembled circuit board for our femtosatellite, we cannot control the placement of the components on the board. One way to mitigate these effects would be to design the femtosatellite printed circuit board from scratch, incorporating the laws of physics that dictate the exact placement of every resistor, capacitor and other component on the board for optimum performance.

Using the Arduino interface to program the microcontroller on our femtosatellite imposes limitations on the kind of encoding and modulation schemes that could be used. Directly programming the microcontroller aboard the femtosatellite could provide the ability to control every component on the femtosatellite board and would also provide more flexibility to design the signal transmitted using any encoding scheme or modulation technique.

We have seen that our autocorrelation plots in Figure 5.23 have a small error. They are not perfectly aligned. We speculate that an improved, dynamic system identification process would improve the the correlation we have have achieved. This implies that a thermal lock loop would be necessary in the ground station trying to achieve GPS positioning accuracy using this simple femtosatellite. A thermal lock loop would be a mechanism to compensate for the effect of heat on the received

pulses such that there is no loss of signal pulses. A thermal lock loop would also help mitigate the Warm–up and Cool–down effects.

We have only discussed some of the solutions to the behavioral issues of our femtosatellite system. Certainly there are many other solutions that can be devised. These solutions with the above mentioned suggestions could help build a ground station which could accurately track femtosatellites and thus be a stepping stone in the process of successful femtosatellite communication.

Appendix A

Abbreviations and Definitions

Definitions of Transient response characteristics:

Transient Response: Transient response of a system is the part of the time response that goes from the initial state to the final state.

Delay Time (t_d): Delay time is the time required for the transient response of a system to reach half of its final value for the first time.

Peak Time (t_p): Peak time is the time required for the transient response to reach the first peak of overshoot.

Maximum Overshoot (Mp): It is the maximum peak value of the response curve measured from unity.

Please refer [19] for the details on the relationships between the different transient response characteristics.

References

[1] J. A. Atchison and M. A. Peck, "A passive, sun-pointing, millimeter-scale solar sail," *Acta Astronautica*, vol. 67, no. 1, pp. 108–121, 2010.

[2] K. Baker and D. Jansson, "Space satellites from the world's garage-the story of amsat," in *Aerospace and Electronics Conference, 1994. NAECON 1994., Proceedings of the IEEE 1994 National*. IEEE, 1994, pp. 1174–1181.

[3] D. J. Barnhart, T. Vladimirova, and M. Sweeting, "Satellite-on-a-chip-a feasibility study," *Proc. 5th Round Table on Micro/Nano Technologies for Space,? Nordwijk, The Netherlands*, 2005.

[4] D. J. Barnhart, T. Vladimirova, A. M. Baker, and M. N. Sweeting, "A low-cost femtosatellite to enable distributed space missions," *Acta Astronautica*, vol. 64, no. 11, pp. 1123–1143, 2009.

[5] D. J. Barnhart, T. Vladimirova, and M. N. Sweeting, "Very-small-satellite design for distributed space missions," *Journal of Spacecraft and Rockets*, vol. 44, no. 6, pp. 1294–1306, 2007.

[6] P. Dana, *Global Positioning System Overveiew.*, available at http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html.

[7] GNURadio, *GNURadio, The Free and Open Software Radio Ecosystem.*, available at http://gnuradio.org/redmine/projects/gnuradio/wiki.

[8] P. Harman, *A Discussion on Automatic Gain Control (AGC) requirements for the SDR1000*, available at http://support.flex-radio.com/Downloads.aspx?id=98.

[9] N. Instruments, *A Digital Downconverter for the NI 5734*, available at http://www.ni.com/example/31525/en/.

[10] ——, *NI FPGA*, available at https://www.ni.com/fpga/.

[11] ——, *What is I/Q Data?*, available at http://www.ni.com/white-paper/4805/en/.

[12] S. Janson and D. Barnhart, "The next little thing: Femtosatellites," 2013.

[13] S. Koslowski, M. Braun, J. P. Elsner, and F. Jondral, "Wireless networks in-the-loop: Emulating an rf front-end in gnu radio," in *SDR Forum 2010 European Reconfigurable Radio Technologies Workshop*, vol. 25, 2010.

[14] F. Lanoford-Smith, "Radiotron designer's handbook," *American Journal of Physical Medicine and Rehabilitation*, vol. 32, no. 5, p. 326, 1953.

[15] N. Manicka, "Gnu radio testbed," Ph.D. dissertation, University of Delaware, 2007.

[16] G. R. McVittie and K. D. Kumar, "Design of a cots femtosatellite and mission," in *AIAA Space 2007 conference and exposition*, 2007.

[17] P. Misra and P. Enge, *Global Positioning System–Signals, Measurements and Performance Second Edition*. Massachusetts: Ganga-Jamuna Press, 2006.

[18] U. of Texas at Austin, *Autocorrelation of GPS Signals.*, available at http://courses.ae.utexas.edu/ase389p7/projects/woodman/Correlation/Autocorrelation.doc.

[19] K. Ogata, "Modern control engineering, 1997," *ISBN: 0-13-227307-1*, pp. 299–231.

[20] E. Research, *AD9860 Datasheet*, available at http://www.analog.com/static/imported-files/data_sheets/AD9860_9862.pdf.

[21] ——, *Application Note - Frontends, Sub-Device Specifications, and Antenna Port Selection*, available at http://www.ettus.com/content/files/kb/application_note_frontends_subdevices_antenna_ports.pdf.

[22] ——, *UHD Daughterboard Application Notes*, available at http://files.ettus.com/uhd_docs/manual/html/dboards.html.

[23] ——, *UHD General Application Notes*, available at http://files.ettus.com/uhd_docs/manual/html/general.html.

[24] ——, *USRP 1 Datasheet*, available at https://www.ettus.com/content/files/07495_Ettus_USRP1_DS_Flyer_HR.pdf.

[25] W. Weng, Y. Liu, H. Hu, and D. Yuan, "An improved ofdm-msk system for wireless communications," in *Communications and Networking in China, 2009. ChinaCOM 2009. Fourth International Conference on.* IEEE, 2009, pp. 1–5.

Biographical Statement

Namrata Jagdish Kamte was born in Mumbai, India in 1987. She has received a Bachelor of Engineering in Instrumentation Engineering from The University of Mumbai in 2010. She has worked as a research assistant in The Satellite Technology Laboratory at The University of Texas at Arlington since 2012. Her research interests include satellite communication, embedded system design and modern control theory.