Parallel GPS Signal Acquisition and Tracking Design and Analysis using an FPGA.

by

Michael Sammartino

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in Engineering

in the

Electrical Engineering

Program

YOUNGSTOWN STATE UNIVERSITY

August, 2015

Parallel GPS Signal Acquisition and Tracking Design and Analysis using an FPGA.

Michael T. Sammartino

Signature:

Michael T. Sammartino, Student Date

Approvals:

Dr. Frank X. Li, Thesis Advisor Date

Dr. Philip C. Munro, Committee Member Date

Dr. Faramarz Mossayebi, Committee Member Date

Dr. Salvatore A. Sanders, Dean of Graduate Studies Date

Abstract

The purpose of this research is to design and model a digital system to acquire, lock, and track a GPS signal in parallel on an FPGA. This project aims to reduce the receiver's time to first fix from a cold start. Applications for this research include high precision targets requiring worldwide coverage where a stored almanac is not acceptable and a fast time to first fix is needed. It can also be utilized in aviation and safety of life applications where it is imperative that faulty data is immediately known and excluded. The uniqueness of this project allows ultra-low energy applications with only volatile memory and sleep currents in the µA range.

Computer simulations have been performed for: (1) replicating GPS satellite C/A code, (2) determining which satellites are visible, (3) coding a hardware description of parallel correlation, locking, and tracking. The system will be distinctive for applications where there is no need or possibility for storage of almanac information to obtain time to first fix (TTFF) with high speed. During these trials an 'out of the box' cold start scenario is performed each time the code is run. The results have shown that successful GPS signal acquisition and decoding time is 1.7 seconds using hardware coding and an FPGA.

Table of Contents

Table of Figures

Table of Tables

List of Abbreviations

A/S	Anti-Spoofing
ASIC	Application Specific Integrated Circuit
bps	Bits Per Second
BPSK	Binary Phase-Shift Keying
C/A	Course/Acquisition
CDMA	Code Division Multiple Access
DoD	Department of Defense
FDE	Fault Detection Exclusion
FPGA	Field Programmable Gate Array
G2	Gold Code Sequence
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HOW	Hand Off Word
I	In-Phase bit
IF	Intermediate Frequency
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
L1	1575.42 MHz Radio Spectrum
L2	1227.60 MHz Radio Spectrum
LE	Logic Element
LSB	Least Significant Bit
MLS	Maximum Length Sequence
MHz	Megahertz
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
PLL	Phase Locked Loop
PRN	Pseudo-Random Noise
Q	Quadrature-Phase bit
RAIM	Receiver Autonomous Integrity Monitoring
SA	Selective Availability
SDRAM	Synchronous Dynamic Random-Access Memory
SVN	Satellite Vehicle Number
TLM	Telemetry Word
TOA	Time Of Arrival
TTFF	Time To First Fix
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Chapter 1 Introduction

The Global Positioning System (GPS) is a satellite-based navigation system developed and maintained by the U.S. Department of Defense [1]. GPS provides worldwide coverage where any point on the earth there will be at least three satellites visible. Satellite coverage over the contiguous U.S. is greater and satellites can be repositioned during war-time for increased accuracy. Citizens are becoming reliant on the successful operation of the GPS system and many daily tasks are performed using the data it provides. This research aims to reduce an embedded devices time to first fix providing less wait time for a consumer and greater performance for an application. An initial test was performed using a Garmin Forerunner 210 GPS watch. This watch is a common device used by fitness hobbyists in order to track run times and routes. The test performed was the time for the watch to receive a valid satellite signal and begin timing and location tracking. The test was performed in two separate geographical locations in order to test the time to start from a cold ‘out of box’ condition. This research will attempt to outperform the Hot-start time of this commercially available watch.

Table 1 Garmin GPS watch startup times

	Little Rock, Arkansas	**Youngstown, Ohio**
Cold-start	95 seconds	106 seconds
Hot-start	8 seconds	7 seconds

In theory, five satellites are necessary for a complete navigation solution (including velocity) which includes x-position, y-position, z-position, time, and clock bias error. This can be reduced to three satellites excluding the z-position and time for reduced accuracy. The complete system consists of three segments which are the control, space, and user segments. The control segment is a network of ground based systems which continually monitor the health and make fine adjustments to the space segment. The master station is located at Schriever Air Force Base in Colorado Springs, Colorado and an overview is shown in Figure 1. The control segment also includes six monitor stations which monitor the exact altitude, position, speed, and overall health of the satellites [2]. Each satellite also communicates with the ground control stations in order to give the user the most precise data available. The satellites do not calculate their own positions; they receive exact positional information from the ground control stations.

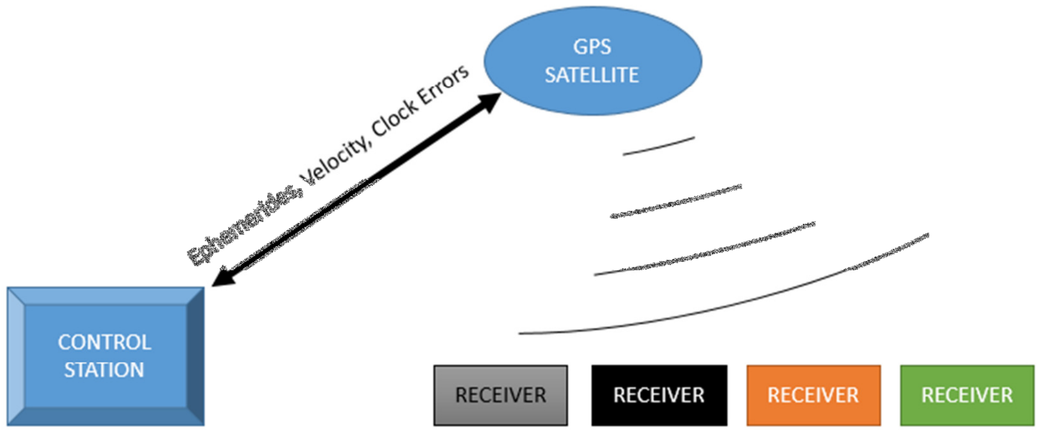


Figure 1 GPS control segment

Safety measures are taken to ensure the constellation will remain functional (with accuracy degradation) even if the master control stations are non-functional. The heart of the space segment is the GPS constellation. It was originally designed to use 24 satellites; however, as of June 2014 there are currently 32 satellites in orbit [3]. The constellation consists of six orbital planes each 55 degrees relative to the equator and 60 degrees relative to each other. The orbits are nongeostationary and orbit at approximately 26,560 km. The approximate cost of this constellation to date is $10 billion [4]; however, its services are free to any user in the world. Antennas on current satellites are non-directional. The newest generation of satellites are being launched with high-gain directional antennas which can be aimed during military conflicts to dramatically increase accuracy and signal coverage. Behind the scenes the GPS constellation is extremely technologically advanced, accurate, and secure. Signals are secure from jamming by the use of pseudorandom noise. Users are denied positional accuracy unless they possess a cryptographic key and can access the encrypted P-code. This is referred to as anti-spoofing (A/S) and it will not allow an unauthorized user to simulate a GPS signal to fool receivers.

GPS has become increasingly popular in the commercial market to include using position information for asset tracking, driving directions, aviation, and time signals. If directions are needed, it is seemingly simple to open a smart phone's 'Maps' app, type in an address and get the best routing to your destination. In aviation, GPS is used to improve the safety and efficiency of flight. GPS based approaches are used to safely navigate below the weather until the pilot can become visual with the runway and make a safe landing. In agriculture, GPS position is used to collect timely geospatial information on soil-plant-animal requirements. This data is used for applying site-specific treatments to increase agricultural production and protect the environment [5]. Each application has specific requirements for accuracy and time to first fix. There are three different methods for starting states of a receiver which are cold-start, warm-start, and hot-start [6] and shown in Figure 2. In a cold-start, the receiver has no location or timing data making it the most time consuming start sequence. During a warm-start the receiver has valid almanac data and assumes the receiver has not moved significantly since its last start. A hot-start is the quickest type of start-up and the receiver still has valid ephemeris data and precise time. Each application of GPS uses different techniques to decide which method will be used.

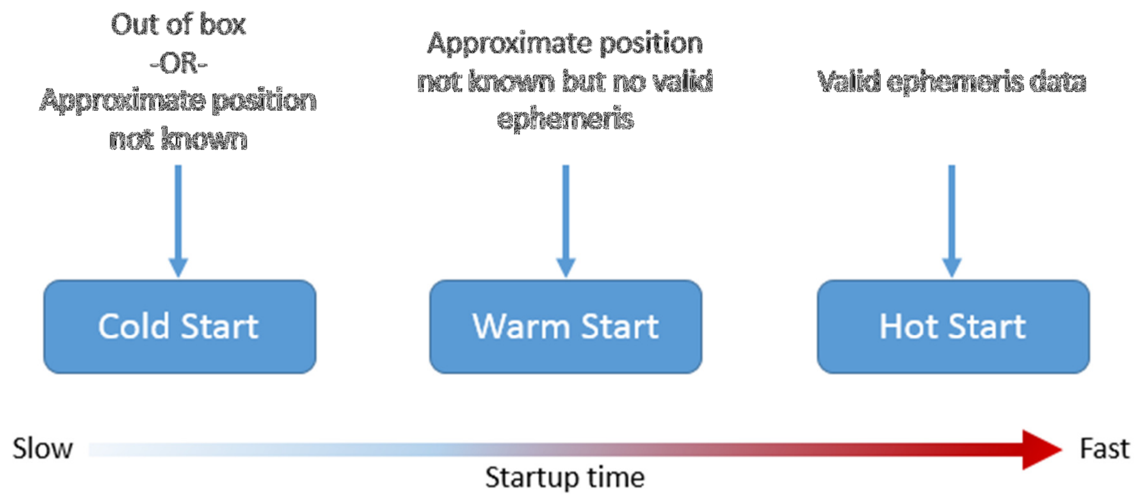

Out of box
-OR-
Approximate position not known
Approximate position not known but no valid ephemeris
Valid ephemeris data
Cold Start
Warm Start
Hot Start
Slow
Fast
Startup time

Figure 2 GPS startup types

Chapter 2 GPS Signal

The backbone of GPS is based off extremely accurate clocks. Each satellite carries multiple on-board cesium and/or rubidium atomic clocks. Signals are carried via two spread spectrum L-band carrier signals transmitting on carrier frequencies 1575.42 MHz and 1227.6 MHz. Both carrier signals are integral multiples of the base frequency f_B= 1.023 MHz.

$$L_1 = f_B \cdot 1540 \tag{1}$$

$$L_2 = f_B \cdot 1200 \tag{2}$$

The L_1 signal utilizes binary phase shift keying (BPSK), modulated by two pseudorandom noise (PRN) codes in phase quadrature, designated as the C/A code and P-code. The L_2 signal is BPSK modulated by only the P-code. This paper will focus on the L_1 signal since the L_2 signal is only usable with a valid crypto code from the U.S. military. The L_1 signal can be mathematically modeled with a 1.023 MHz chipping rate and 1 ms period. Using P_I and P_Q as the respective carrier powers for the I and Q components, *d(t)* as the 50 bps data modulation, and *c(t)* and *p(t)* as the respective C/A and P pseudorandom waveforms we can model the L1 signal in the time domain as

$$L1(t) = 2PId(t)c(t)cos(\omega t + \theta) + 2PQd(t)p(t)sin(\omega t + \theta) \qquad (3)$$

An RF front end, normally an ASIC, is used to extrapolate this analog signal into the digital domain. These chips typically contain an RF amplifier followed by mixing of the signal down to an IF with filtering to eliminate out-of-band signals [7]. The output is normally composed of I and Q components which can range from 2 to 4 bits. The power consumption of RF front ends can be as little as 23 mA [8].

2.1 Pseudoranges

The GPS receiver uses the concept of one-way time of arrival ranging. The range is computed by multiplying the speed of light by the amount of delay that a GPS signal needs to travel from the satellite to the receiver. This quantity is very important as time is one of the base SI quantities which have the highest accuracy of measurement [9]. Pseudorange is the distance to a satellite measured in this fashion, plus or minus some additional error sources [10]. The calculation of a three-dimensional position (not including velocity) requires a minimum of four satellites- three for x, y, z coordinates, and one to solve for clock bias error. Additional satellites are required for certain applications such as aviation which utilize receiver autonomous integrity monitoring (RAIM) and fault detection exclusion (FDE) for additional safety measures. RAIM automatically verifies the validity of data against other satellites, ground stations, and the receivers known position to decide if one of the received signals is invalid. This is important because the navigational message (which includes satellite health data) does not transmit at a fast enough rate. These pseudoranges can be shown mathematically as:

$$PR1 \ldots PR4 = \Delta T1 \ldots \Delta T4 \cdot c \tag{4}$$

$$(x1 - ux)^2 + (x2 - uy)^2 + (x3 - ux)^2 = (PR1 - CE \cdot c)^2 \tag{5}$$

where *c = 299,792,458 [m/s], ux, uy, and uz are user positions, x1, x2, x3 are satellite positions, CE is receiver clock bias error, ΔT1… ΔT4 are difference between time-coded satellite signals and received signal.*

After considering the use of four satellites, squaring both sides, and rewriting in matrix form it yields:

$$\begin{bmatrix} PR1 - (x1^2 + y1^2 + z1^2) - r^2 \\ PR2 - (x2^2 + y2^2 + z2^2) - r^2 \\ PR3 - (x3^2 + y3^2 + z3^2) - r^2 \\ PR4 - (x4^2 + y4^2 + z4^2) - r^2 \end{bmatrix} = \begin{bmatrix} -2x1 - 2y1 - 2z1\ 1 \\ -2x2 - 2y2 - 2z2\ 1 \\ -2x3 - 2y3 - 2z3\ 1 \\ -2x4 - 2y4 - 2z4\ 1 \end{bmatrix} \begin{bmatrix} ux \\ uy \\ uz \\ CE \end{bmatrix} \tag{6}$$

$$Y = MX \tag{7}$$

$$M^{-1}Y = M^{-1}MX = \begin{bmatrix} ux \\ uy \\ uz \\ CE \end{bmatrix} \tag{8}$$

Once the users position is known from the satellites coordinates are transferred to a spherical coordinate system with the earth's core as the center. The distance from the center of earth *r*, latitude *L*, and longitude l can be found as

$$r = \sqrt{x_u^2 + y_u^2 + z_u^2} \tag{9}$$

$$L = tan^{-1}\left(\frac{z_u}{\sqrt{x_u^2 + y_u^2}}\right) \tag{10}$$

$$l = tan^{-1}\left(\frac{y_u}{x_u}\right) \tag{11}$$

These equations do not give exact latitudes and longitudes since earth is not a perfect sphere therefore modification is necessary transferring spherical coordinates to ellipsoidal.

2.2 Navigation Message

The GPS navigation message is transmitted at a data rate of 50 bps. This message is modulated in the L_1 carrier signal. The message structure is composed of a basic format of a 1500 bit long frame made up of five 300 bit sub frames. Each sub frame starts with a pair utilizing a Telemetry (TLM) word and a Handover word (HOW) shown in Figure 3. Each pair contains code checking parity. The first three sub-frames contain the required data to calculate user position; therefore a minimum of 18 seconds of data is necessary to calculate a user position [11]. Each satellite transmits two types of data the Almanac Data and the Ephemeris Data. Almanac data is course orbital parameters for all SVNs and each SVN broadcasts Almanac data for all SVNs. Almanac data is not very precise. Ephemeris data is very precise orbital and clock parameters and each SVN only transmits ephemeris data for itself. Almanac data is considered valid for several months where ephemeris data is transmitted every 30 seconds and valid only for 30 minutes.

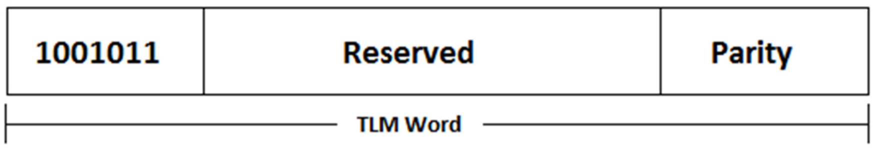


Figure 3 Navigation message TLM format

The 300 bit sub frames of the navigation message contain the following information with the LSB transmitted last.

Sub frame 1 – Satellite clock and health data

- Contains clock and clock corrections parameters
- Satellite accuracy
- Satellite health
- Reserved data

Sub frames 2 and 3 – Satellite ephemeris data

- Orbital parameters
- Harmonic corrections
- Reference time
- Satellite position information

Sub frames 4 and 5 – Support data

- Almanac data
- Special messages
- Ionosphere data
- Spare frames

2.3 Pseudo-Random Noise (PRN)

Two independent signals are transmitted by each satellite called the course acquisition (C/A) code and the precise (P) code. Each satellite has a unique C/A code which can allow the user to determine which satellite the signal comes from. The C/A code is a sequence of 1,023 bi-phase modulations. Each opportunity for a phase-reversal modulation, or switch from a zero to a one, is called a chip (whether or not the phase is actually reversed) [4]. The entire sequence is repeated 1,023 times per second, which results in a chip rate of 1.023 MHz. GPS receivers internally match the unique code of each satellite and compare that to the actual received code. The difference in time can be calculated. This method of measurement is known as code correlation and is shown in Figure 4. Code correlation will only tell the user where they are in relation to the satellite. Alone it cannot calculate the user’s location on earth. In order for the user to know the satellites position, each satellite transmits its position in space on the ephemeris. Another important aspect included in the navigation message is the atomic clock bias errors. The position of the satellites is not calculated by the satellites themselves but uploaded from the control stations on the ground. The satellites merely relay this information to the end user.

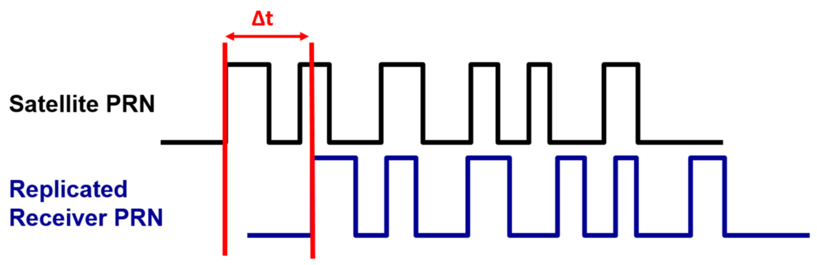


Figure 4 PRN time calculation

2.4 Spread Spectrum and Code Division Multiple Access (CDMA)

GPS signals use a spread spectrum communication system. Spread spectrum uses a larger frequency bandwidth than is needed to transmit information in order to make the transmission more secure. This method has been used by the military for decades in order to make communications more difficult to intercept and jam. Additionally, a spread spectrum delivers multi-user random access and high resolution. Spread spectrum communications were a catalyst for commercial applications including mobile radio applications, satellite communications, and positioning systems as more bandwidth is requested by consumers [9].

2.5 Course Acquisition (C/A) Code

The course acquisition code serves several purposes to include accurate range measurements, resistance to errors caused by multipath, simultaneous range measurement from several satellites, and protection from jamming. Each satellite has a unique C/A code which allows the receiver to communicate with multiple satellites on the same frequency at the same time. All of the codes consist of a repeating sequence of 1023 chips occurring at a rate of 1.023 MHz with a period of 1ms. Each chip is either positive or negative with the same magnitude. The polarities of the 1023 chips appear to be randomly distributed but are in fact generated by a deterministic algorithm implemented by shift registers. The algorithm produces Gold codes, which have the property of low cross-correlation between different codes. C/A code has a unique property called autocorrelation. This method is used in many fields such as communications, statistics, and finance. Autocorrelation is a mathematical tool for finding repeating patterns. The C/A code has an autocorrelation function defined as

$$y(t) = \frac{1}{T}\int_0^T c(t)c(t-T)dt \tag{12}$$

where *c(t)* is the C/A-code waveform, t is the relative delay (in seconds), and T is the code period (1 ms). This autocorrelation function is the basis for code tracking in GPS receivers. As GPS is a spread spectrum signal the power spectrum is distributed in the frequency domain. This can be shown mathematically and is related to the autocorrelation function by:

$$\psi(f) = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T}\psi(\tau)e^{-j2\pi f\tau}d\tau \tag{13}$$

When plotted, this function would consist of spectral lines with 1-kHz spacing due to the 1-ms periodic structure [1].

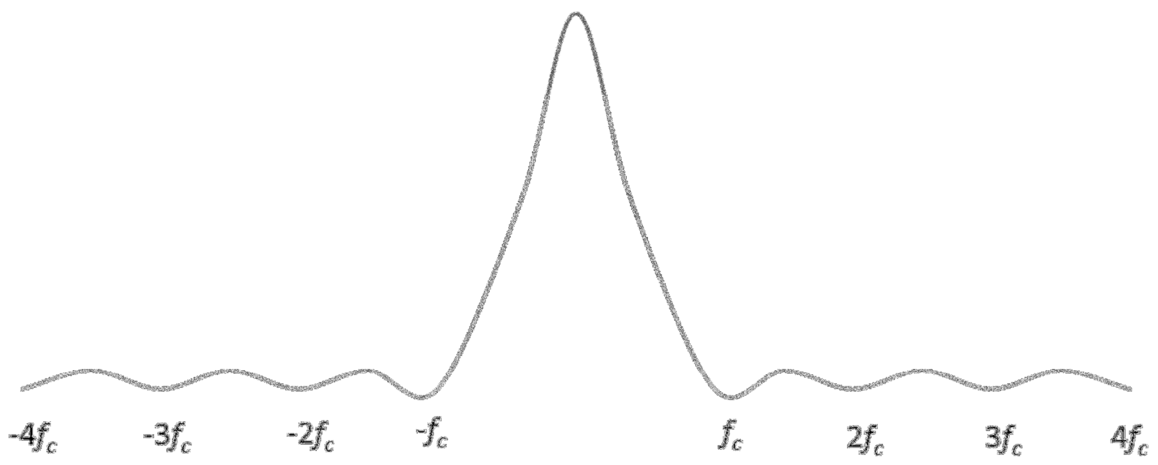


Figure 5 C/A power distribution when fc = 1.023MHz

2.6 Doppler Shift

Doppler shift is the change in frequency of a wave relative to the observer caused by both the source and observer's relative motion. This phenomenon plays an important role in GPS signal integrity. Sources of this error include the rotation of earth, satellite orbit speed, and the receiver's relative motion to earth and the constellation [12]. Doppler Shift error is able to be calculated and is extremely important when the receiver is moving at high speeds in applications such as aircraft or missiles. The maximum rate of change of Doppler shift error can be found by taking the derivative of the doppler shift with respect to time. The solution comes out to approximately 0.178 m/s^2 or 10 kHz/s [13]. This error can be corrected in software or hardware after the navigation message has been extracted. The importance of this error is great as one second of error in measuring the travel time translates into approximately 300 meters of error in position [14].

2.7 Signal Acquisition and Tracking

A GPS receiver has many functions which need complete before it can provide a valid position solution. First, the transmitted signal needs to be filtered and processed into a digital IF signal. ASICs will output in-phase (I) and quadrature (Q) phase components made up of:

$$I = data \cdot \sin(f_c 2\pi t) \tag{14}$$

$$Q = data \cdot \cos(f_c 2\pi t) \tag{15}$$

where f_c is the L_1 carrier frequency. This information will be translated into PRN code which can then be tracked. The necessary steps must occur in a specific order for the data to be valid.

2.7.1 Determine which satellites are visible to the receiver.

In the first step the receiver needs to determine which satellites are available to lock onto. Ideally the receiver will find these satellites quickly, and this performance is measured as time to first fix (TTFF). Depending on receiver characteristics, the TTFF might range from seconds to several minutes. Receivers store almanac data which gives the approximate locations of satellites. If the receiver knows its approximate position (within a few hundred miles) it can search for applicable satellites. The almanac data stored by the receiver is valid for several months. If the receiver is straight out of the box, the search is known as a blind search as it has no almanac data or idea of its position. Cell phones use a technology known as aGPS, in order to decrease the TTFF. This uses almanac data stored in the cell tower to approximate the phones location.

2.7.2 Determine the approximate Doppler of each satellite.

The receiver, knowing its approximate location and velocity, can further reduce the TTFF by estimating the Doppler shift frequency. This reduces the number of searches the receiver needs to perform. A common method to search for doppler shift is to relate the satellite velocity vector (***v***) to and receiver reference clock error (***u***) with the carrier wavelength (λ) known.

$$fd1 = \frac{1}{\lambda}(\boldsymbol{v} \cdot \boldsymbol{u}1 - \boldsymbol{v}1 \cdot \boldsymbol{u}1) + fb \tag{16}$$

2.7.3 Search for the signal.

GPS Signals, being spread spectrum signals, are not as easily tuned as normal radio waves. Since the C/A code’s total signal power is spread over a wide bandwidth, the signal needs to be decoded with a replica code in order to align with the received code. The receiver needs to execute a two-dimensional search in order to find the satellite signal. The dimensions are the C/A code delay and carrier frequency. This search must be conducted across the full delay range of the C/A code for each frequency searched. A generic method to execute the search is to multiply the received waveform by replicas of C/A code, translated by various frequencies, and then passed through a baseband correlator and lowpass filter. Figure 6 shows this generic search. The energy detected will be significant only if the selected code delay and frequency translation match the received signal. When this energy reaches a predetermined threshold, the signal is considered received. Receivers with a relatively high threshold may not be able to detect satellites which are actually visible.

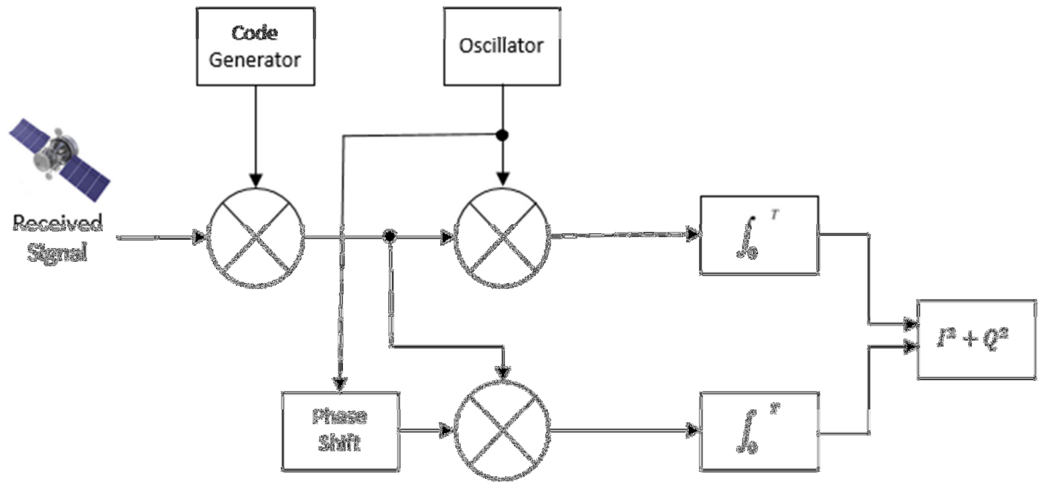


Figure 6 Generic C/A search

2.7.4 Detect and confirm the presence of signal.

The confirmation of a GPS signal can be cumbersome since by the time the GPS signal reaches earth the signal is extremely weak. This can be problematic in receivers as the pre-defined threshold set may not be sufficient to confirm a signal.

2.7.5 Lock onto and track the C/A code.

When the signal is confirmed the reference signal generated by the receiver will be in close alignment with the received signal. This is true at the initial time; however, since the receiver and satellite are constantly moving eventually the doppler shift will bring them out of alignment again. There is a need to continually adjust the timing of the reference code so it maintains accurate alignment with the received code which is known as code tracking. Code tracking should be initiated as soon as signal detection is confirmed. The receiver generated code needs to line up with incoming code as closely as possible. Additionally during this step the code is despread and the precise time measurement or time of arrival (TOA) is taken.

2.7.6 Lock onto and track the carrier.

Next, the carrier phase needs to be tracked to provide velocity measurements and obtain pseudorange measurements for high-accuracy receivers using a phase-lock loop (PLL). A PLL is a control system which generates an output signal with a phase related to the input signal. Another common method of locking onto the carrier is using a Costas loop. In this design the output of the receiver intermediate frequency (IF) amplifier is converted to a complex baseband signal by multiplying by I and Q outputs of a numerically controlled oscillator (NCO) and integrating each product.

2.7.7 Perform data bit synchronization.

In order for bit synchronization to occur it is important for the carrier loop to be locked. When the PLL is locked, the output of the I integrator will be a sequence of values occurring once per millisecond. A common method to perform data bit synchronization is to realize a modulo-20 counter using the receiver-generated C/A code and mark the data boundaries.

2.7.8 Demodulate the navigation data.

Once the data bits are synchronized the navigation data can be demodulated. A common method is to integrate the in-phase component generated by a PLL. Each data bit is generated by integrating the in-phase component over 20ms.

Chapter 3 Hardware Design and Simulation

3.1 Overview

The tasks of this system include (1) replicating GPS satellite C/A code, (2) determining which satellites are visible, (3) coding a hardware description of parallel correlation, locking, and tracking. The verification was done using a variety of software including MATLAB, Python, ModelSIM, and Altera Quartus II. The general program flow is shown in Figure 7. The entirety of the code was custom created.

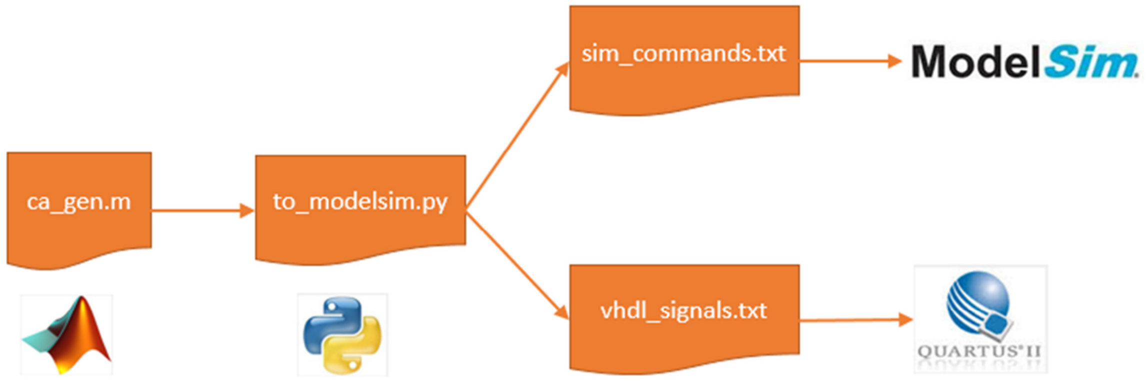


Figure 7 General hardware overview

The simulation assumes the analog processing has been completed by a chip such as Maxim 2769 or Skyworks 4150. Only the digital tasks are included up to decoding the navigation message. The goal is decreasing the search time from a cold start using the concurrent processing properties of an FPGA. The functional diagram in Figure 8 shows analog processing steps prior to digital processing performed in this project. GPS ASIC chips also include a clock output (CLK_OUT) signal which is a 1.023 MHz signal synchronized with the incoming satellite signal.

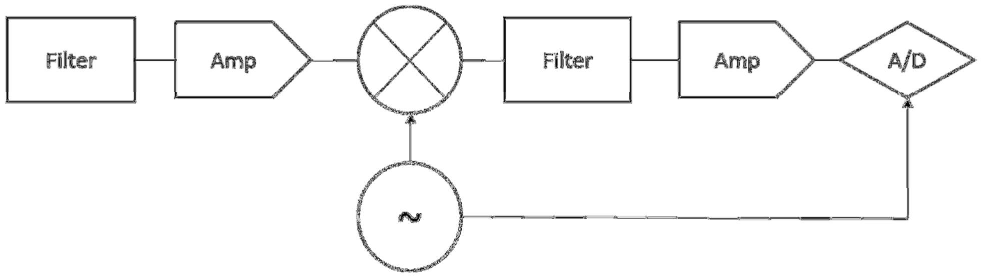


Figure 8 Analog processing functional diagram for a typical ASIC chip

The output of this ASIC is fed into multiple channels of an FPGA which simultaneously track each channel until all of the visible satellites are tracked. An FPGA is capable of running code concurrently as shown in Figure 9, unlike a microcontroller or general purpose processor, which executes code sequentially via a task scheduler. This capability allows all channels to be tracked without delay.

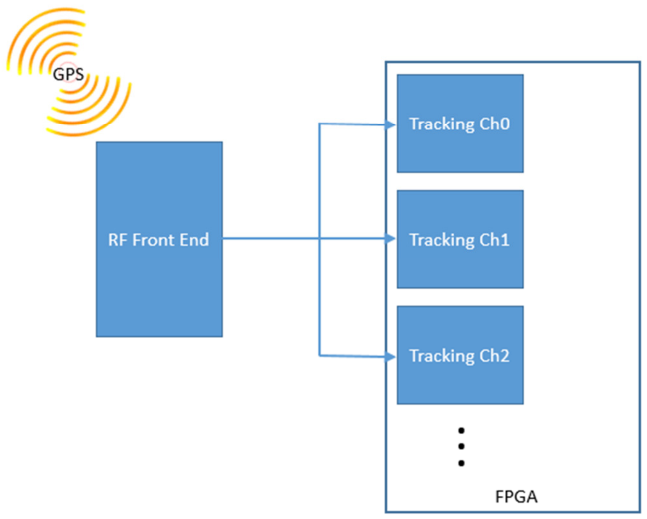


Figure 9 Parallel properties of FPGA

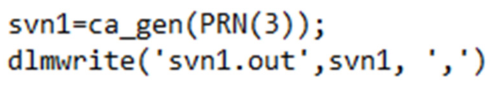
3.2 Generate Satellite C/A Code: MATLAB ca_gen.m

MATLAB (matrix laboratory) is a high-level programming language intended primarily for numerical computing. It shares much of its syntax with the C language, however it includes many built in functions for mathematical computations. MATLAB was used to generate C/A codes of a specific, but random, satellite. The code was built from the bottom up. The code uses a G2 generator along with an MLS which uses a code phase selection to form the output. Each SVN has an assigned PRN number, phase selection, and chip delay. This C/A code uniquely identifies each satellite [30]. The first five satellites per the GPS standards are shown in Table 2.

Table 2 Autocorrelation assignments

Satellite ID	Code Phase Selection	Code Delay Chips
1	$2 \oplus 6$	5
2	$3 \oplus 7$	6
3	$4 \oplus 8$	7
4	$5 \oplus 9$	8
5	$1 \oplus 9$	17

The code generates binary values of one complete C/A cycle, 1023 chips, output in a text file named svn1.out, svn2.out, etc. The function input is the PRN number of which satellite the user wants. For this project the satellite PRN codes used are SVNs 3,7,23,16,4, and 27. These are in no specific order and are used solely to see if the VHDL code will be able to discriminate between satellites. An array is built with delay values and modulo 2 addition is used along with up sampling in order to achieve the 1.023 MHz digital samples necessary. MATLAB's built in 'dlmwrite' function was used in order to pre-process the output for the next step. An example syntax of ca_gen is shown in Figure 10.

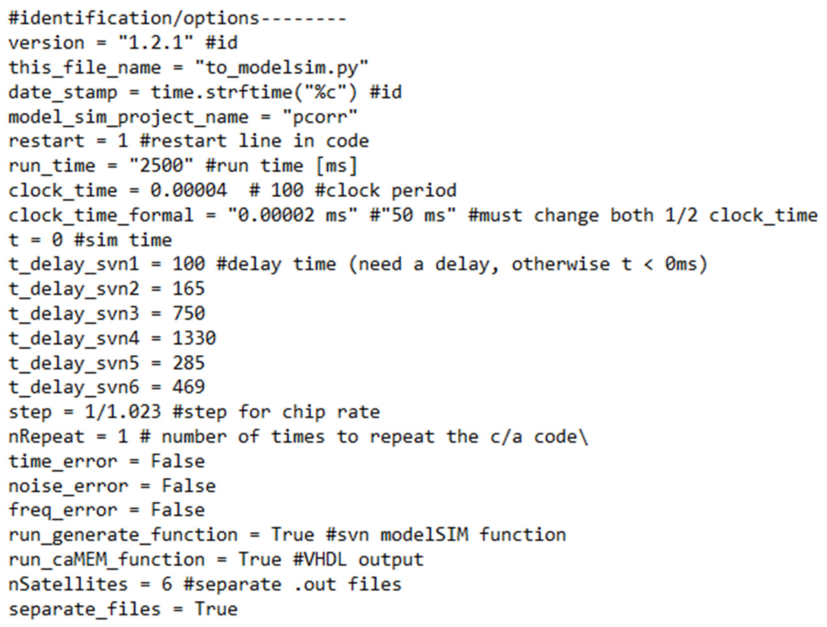


```
svn1=ca_gen(PRN(3));
dlmwrite('svn1.out',svn1, ',')
```

Figure 10 MATLAB command of PRN C/A Code

3.3 Preparing to Simulate: Python to_modelsim.py

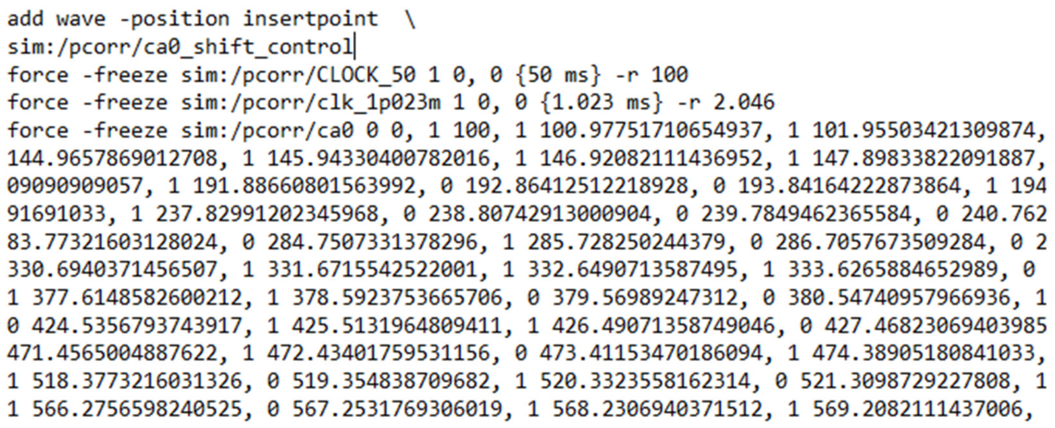
Python is an open source high-level interpreted scripting language that boasts many features which makes code easy to modify and reuse. This was essential for this project in order to try different scenarios ensuring the end product works as expected. An options block was essential for changing simulation scenarios such as satellite numbers, times, and inducing errors. Figure 11 shows the options for the Python outputs.



```
#identification/options--------
version = "1.2.1" #id
this_file_name = "to_modelsim.py"
date_stamp = time.strftime("%c") #id
model_sim_project_name = "pcorr"
restart = 1 #restart line in code
run_time = "2500" #run time [ms]
clock_time = 0.00004  # 100 #clock period
clock_time_formal = "0.00002 ms" #"50 ms" #must change both 1/2 clock_time
t = 0 #sim time
t_delay_svn1 = 100 #delay time (need a delay, otherwise t < 0ms)
t_delay_svn2 = 165
t_delay_svn3 = 750
t_delay_svn4 = 1330
t_delay_svn5 = 285
t_delay_svn6 = 469
step = 1/1.023 #step for chip rate
nRepeat = 1 # number of times to repeat the c/a code\
time_error = False
noise_error = False
freq_error = False
run_generate_function = True #svn modelSIM function
run_caMEM_function = True #VHDL output
nSatellites = 6 #separate .out files
separate_files = True
```

Figure 11 Options for output files

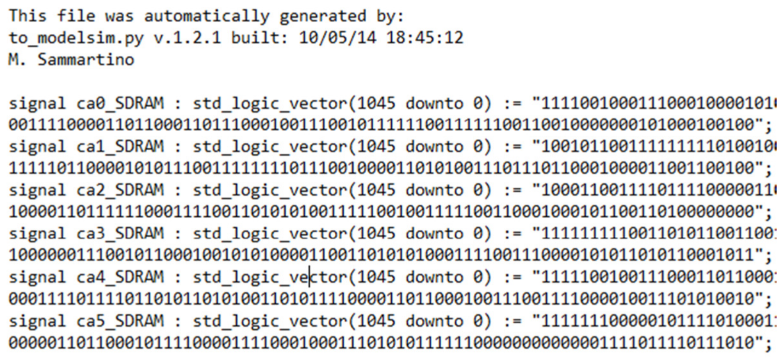
The purpose of the Python script is to output two separate files and easily make changes in the data. The first file outputs actual ModelSIM commands, which makes it possible to copy and paste the output directly to the ModelSIM command line as shown in Figure 12. It is the equivalent of setting up a VHDL test bench except this allows both ModelSIM and VHDL code to be generated at the same time. This dual output reduces simulation setup time.



```
add wave -position insertpoint  \
sim:/pcorr/ca0_shift_control
force -freeze sim:/pcorr/CLOCK_50 1 0, 0 {50 ms} -r 100
force -freeze sim:/pcorr/clk_1p023m 1 0, 0 {1.023 ms} -r 2.046
force -freeze sim:/pcorr/ca0 0 0, 1 100, 1 100.97751710654937, 1 101.95503421309874,
144.9657869012708, 1 145.94330400782016, 1 146.92082111436952, 1 147.89833822091887,
09090909057, 1 191.88660801563992, 0 192.86412512218928, 0 193.84164222873864, 1 194
91691033, 1 237.82991202345968, 0 238.80742913000904, 0 239.7849462365584, 0 240.762
83.77321603128024, 0 284.7507331378296, 1 285.728250244379, 0 286.7057673509284, 0 2
330.6940371456507, 1 331.6715542522001, 1 332.6490713587495, 1 333.6265884652989, 0
1 377.6148582600212, 1 378.5923753665706, 0 379.56989247312, 0 380.5474095796693б, 1
0 424.5356793743917, 1 425.5131964809411, 1 426.49071358749046, 0 427.46823069403985
471.4565004887622, 1 472.43401759531156, 0 473.41153470186094, 1 474.38905180841033,
1 518.3773216031326, 0 519.354838709682, 1 520.3323558162314, 0 521.3098729227808, 1
1 566.2756598240525, 0 567.2531769306019, 1 568.2306940371512, 1 569.2082111437006,
```

Figure 12 ModelSIM command line output from Python code

The second file outputs VHDL signals simulating SDRAM or internal FPGA memory which will be compared to the received C/A code. This allows the VHDL code to create a logical shift right in order to lock onto C/A code. This allows changing satellite PRN codes with ease and the addition of all of the satellites PRN codes in a short amount of time with high accuracy. This code is also capable of directly being copied into the top level VHDL file. Using a scripting language to output these SDRAM values saved a great amount of time as the code needed to be repeated for all SVNs and an error can easily be introduced if hand typed.



```
This file was automatically generated by:
to_modelsim.py v.1.2.1 built: 10/05/14 18:45:12
M. Sammartino

signal ca0_SDRAM : std_logic_vector(1045 downto 0) := "111100100011100010000101
001111000011011000110111000100111001011111100111111001100100000000101000100100";
signal ca1_SDRAM : std_logic_vector(1045 downto 0) := "100101100111111111010010
111110110000101011100111111110111001000011010100111011101100010000110011001 00";
signal ca2_SDRAM : std_logic_vector(1045 downto 0) := "100011001111011110000011
100001101111110001110011010101001111100100111110011000100010110011010000000 00";
signal ca3_SDRAM : std_logic_vector(1045 downto 0) := "111111111001101011001100
10000001110010110001001010100001100110101010001111001110000101011010110001011";
signal ca4_SDRAM : std_logic_vector(1045 downto 0) := "111110010011100011011000
000111101111011010110101001101011110000110110001001110011110000100111010100 10";
signal ca5_SDRAM : std_logic_vector(1045 downto 0) := "111111100000101111010001
000001101100010111100001111000100011101010111111000000000000001111011110111010";
```

Figure 13 VHDL signals output from Python code

It is important to note that the entire VHDL file is not output using Python. The Python output as shown in Figure 13, is a small portion that reduces time and errors in between each setup. The output is simply cut and paste into the top level VHDL file.

3.4 Simulation: VHDL

VHDL is a hardware description language created by the Department of Defense and standardized by IEEE in 1983. VHDL differs from high-level languages since it is a hardware description language. Instead of using a processor and task scheduler to execute tasks, the code is directly executed using gates in hardware. This allows for real-time parallel programming and performance which is not possible using software. However, most FPGA architectures allows CPUs and HDLs to run together maximizing the benefit.

3.5 Parallel Correlator in VHDL

The parallel correlator must be able to execute tasks both sequentially and concurrently. This is not possible with software; therefore a hardware approach needed to be taken. For the scope of this paper any notation of caX will refer to one of the parallel channels ca0 through ca5. This 'X' notation will assume that all six channels are active at the same time and performing the same tasks. The tasks involved are (1) determining which satellites are visible, (2) replicating GPS satellite C/A code, and (3) correlation, locking, and tracking. A functional diagram was created to make program flow easier and shown in Figure 14.

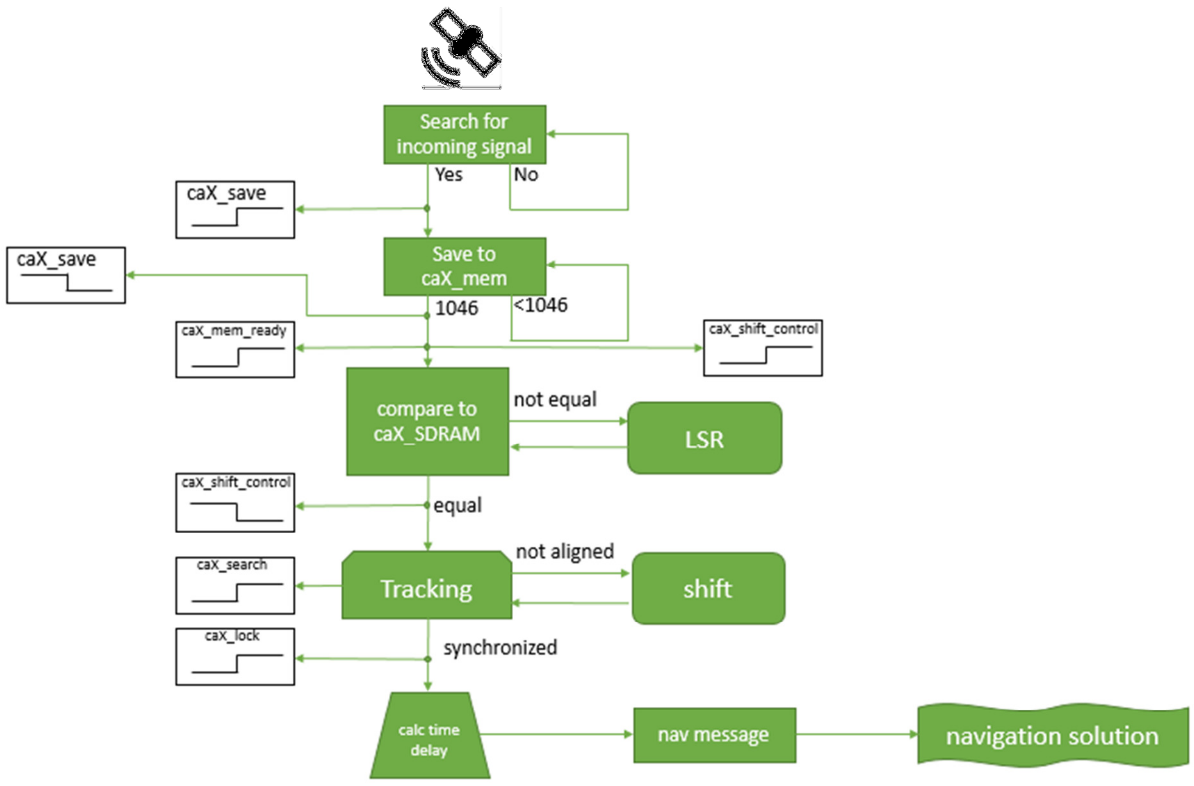


Figure 14 Overview of a single parallel correlator

3.5.1 Determining which satellites are visible

The first task involved is to determine which satellites are visible. Once a data stream is received on ca0, the save signal goes high and the next 1046 bits are stored in ca0_mem. This allows for the entire C/A code to be stored independent of the receive time. This also allows the different C/A codes to start at other than time = 0.

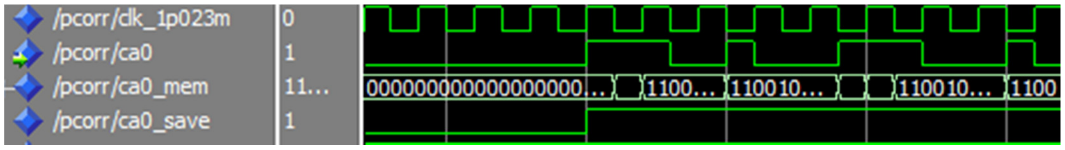


Figure 15 Saving incoming C/A code to FPGA memory

Notice in Figure 15 when the LSB of ca0 is received the ca0_save signal goes high indicating that the entire C/A code will be saved in the ca0_mem memory. Once the entire code is received, ca0_save goes low and no more data will be shifted into ca0_mem. Memory writes are controlled by a 1.023 MHz clock which is forced in ModelSIM and a phase-locked loop when realized on hardware. Note that in reality these clock signals are not synchronized; however, most analog baseband processors have a clock out signal which is synchronized with the incoming satellite which makes this a viable option. Figure 16 shows the parallel process and the different delays simulating weaker and stronger satellite signals. Weaker signals will take longer for the analog processor to discriminate and therefore all signals will become digitized at different rates.

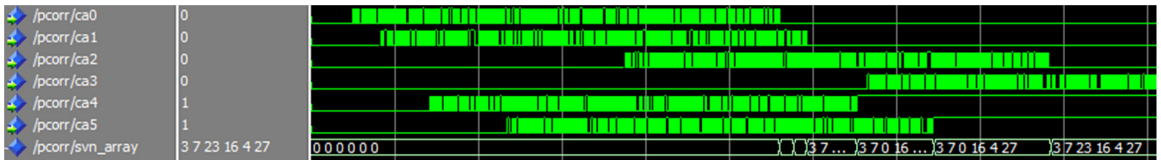


Figure 16 Parallel capturing of visible satellites along with different start times of C/A code

Once C/A code is matched up with one of the caX_SDRAM memories, svn_array will show which satellites are visible to the hardware for further processing. A '0' value in the array means there is no valid satellite visible. In Figure 16, the array initialized with {0 0 0 0 0 0} means there are no valid visible satellites. After all six complete C/A codes are transmitted svn_array is {3 7 23 16 4 27) as the PRN for SVN's 3, 7, 23, 16, 4, 27 were transmitted on ca0 through ca5. A typical SDRAM controller used to store all C/A codes is shown in Figure 17. It is necessary to preload these values so the system can locate the proper satellite. Figure 18 shows the logical shift once the C/A capture is complete. This is necessary to match the C/A code because the LSB will not necessarily be transmitted first as the satellite is continuously transmitting this information. All of the caX_SDRAM's continuously shift while unknown code is on any of the caX_mem channels.

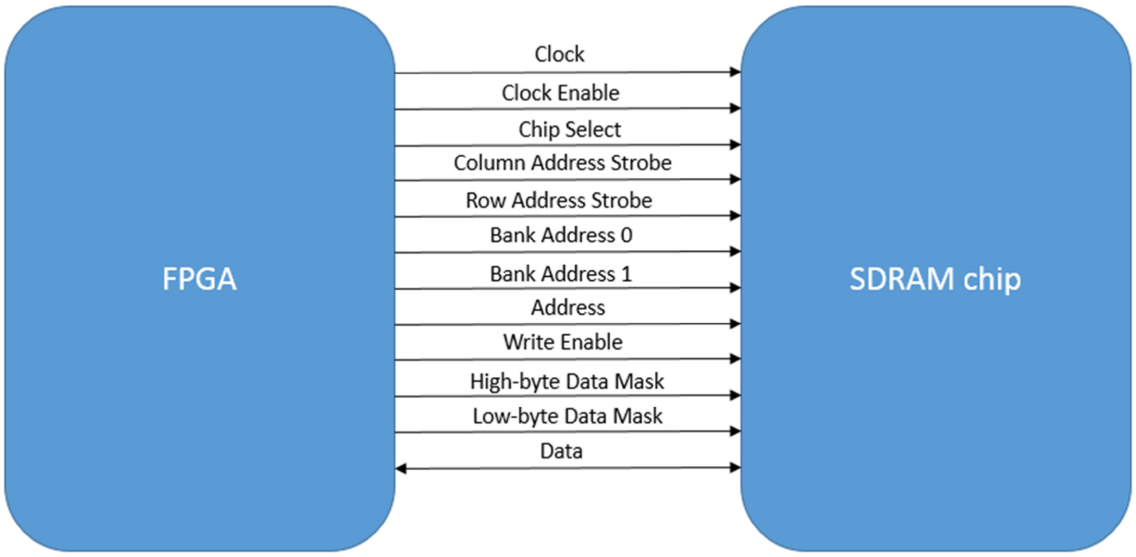
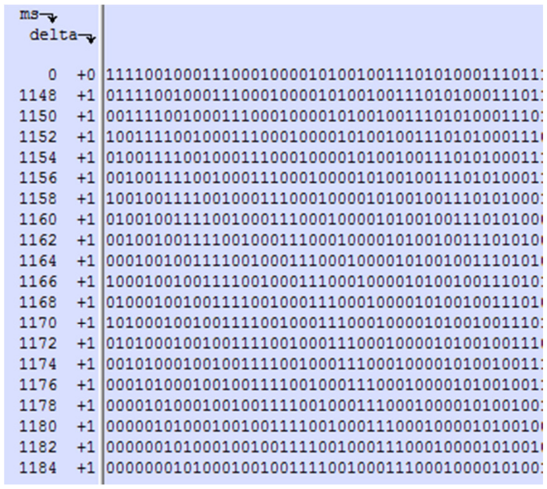


Figure 17 SDRAM controller

Notice in Figure 18 the shift process does not start until 1148 ms into the simulation. This is because the search doesn't start until the caX_shift_control signal goes low. When realized on hardware, the search should be abandoned after a predetermined amount of time and caX_mem dumped when a valid C/A code is not found.



```
ms
 delta

   0  +0 111100100011100010000101001001110101000111011
1148  +1 011110010001110001000010100100111010100011101
1150  +1 001111001000111000100001010010011101010001110
1152  +1 100111100100011100010000101001001110101000111
1154  +1 010011110010001110001000010100100111010100011
1156  +1 001001111001000111000100001010010011101010001
1158  +1 100100111100100011100010000101001001110101000
1160  +1 010010011110010001110001000010100100111010100
1162  +1 001001001111001000111000100001010010011101010
1164  +1 000100100111100100011100010000101001001110101
1166  +1 100010010011110010001110001000010100100111010
1168  +1 010001001001111001000111000100001010010011101
1170  +1 101000100100111100100011100010000101001001110
1172  +1 010100010010011110010001110001000010100100111
1174  +1 001010001001001111001000111000100001010010011
1176  +1 000101000100100111100100011100010000101001001
1178  +1 000010100010010011110010001110001000010100100
1180  +1 000001010001001001111001000111000100001010010
1182  +1 000000101000100100111100100011100010000101001
1184  +1 000000010100010010011110010001110001000010100
```

Figure 18 Logical shift after C/A capture

3.5.2 Replicating GPS satellite C/A code

Once the system knows which satellites are visible it needs to exactly replicate the code in order to calculate timing for the navigation solution. The calculation of timing comes from the difference in time stamps in the navigation message and knowing the constant speed of light (the signals are transmitted at the speed of light). Knowing the timing is important because the amount of time offset made by the receiver's code to make the correlation is directly proportional to the range between the receiver and the satellite. These range measurements are known as pseudoranges. Once the pseudoranges are calculated the navigation message can be extracted for a more precise position. The replicated code is useless unless it is correlated (synchronized) with the incoming c/a signal. The next step in determining position is to synchronize the signals. Figure 19 shows the simulation results of replicated code searching waveforms.

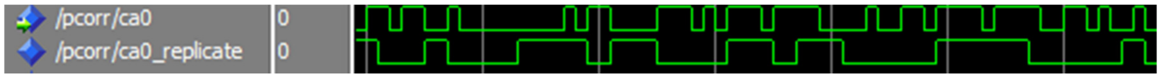


Figure 19 Replicated code searching before correlation

3.5.3 Correlation, locking, and tracking

Once the code is replicated it needs to be correlated, or synchronized with the incoming real-time data from the satellites. Once again the caX_SDRAM memory is used to replicate the C/A code. In order to correlate a snapshot of each vector is taken and compared in VHDL. If the memories do not match then caX_locked is logic low which is logically AND with ca0_replicate and the output will be ‘0’ until correlation is complete. The next step is a logical shift of caX_SDRAM (similar to the algorithm used to search for satellites) and then compared again. Once the codes match, caX_locked will go high and an output will be realized on caX_replicate. The simulation results are shown in Figure 20 with 1.023 MHz clock.

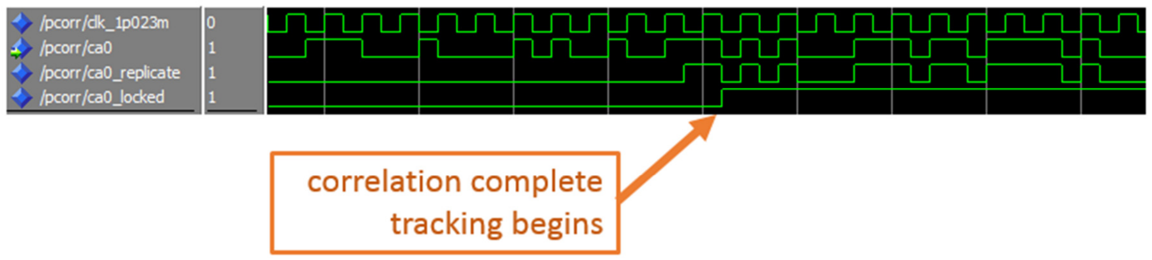


Figure 20 Tracking and correlation of a single satellite

Chapter 4 Hardware Realization

Realizing the simulated code on real hardware is a task for future research; however, the basic correlation code was tested using actual hardware to verify the end result. Altera Quartus II was used to compile the parallel correlator along with a Terasic DE2-115 development and education board as shown in Figure 21, to execute the compiled code. A custom PCB was designed and built in order to test the system after the RF front end.

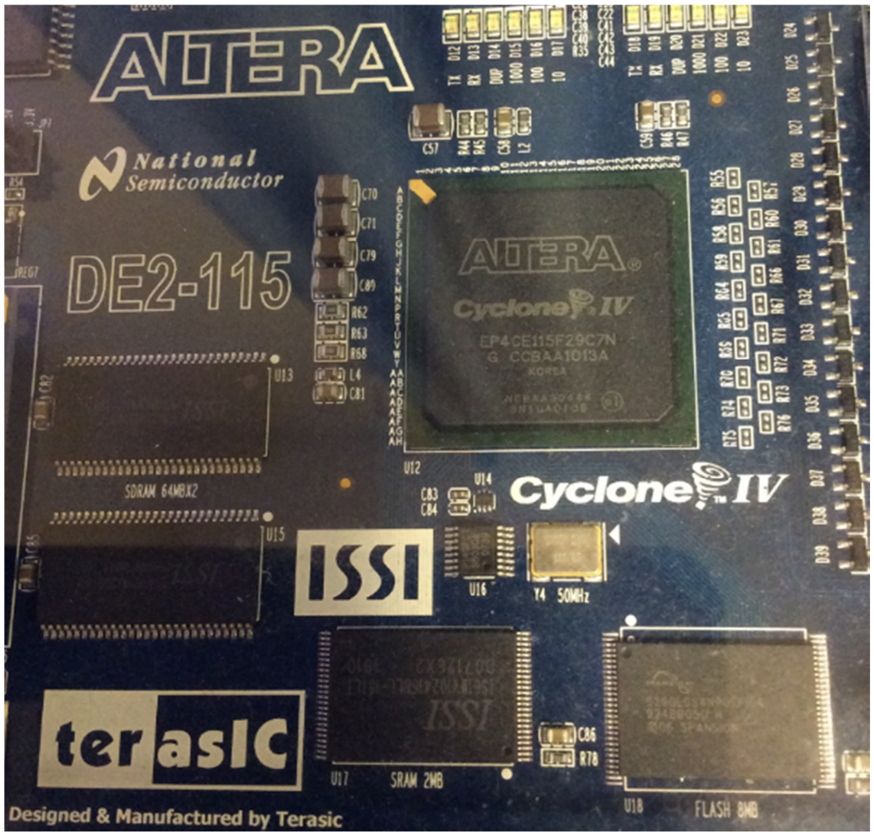


Figure 21 Altera Cyclone IV FPGA on DE2 board

The heart of the DE2 board is an Altera Cyclone IV EP4CE115F29C7N. This FPGA contains 114,480 LEs, 532 embedded 9-bit multipliers, and 4 dedicated PLLs. The main purpose of programming the FPGA was to ensure the simulated code is capable of being executed in real-world scenarios. A state machine in Figure 22 was created for easy interpretation of the current state of hardware. This made troubleshooting a breeze as the DE2 board would tell you which step of the process was trying to be executed on its 7-segment display.

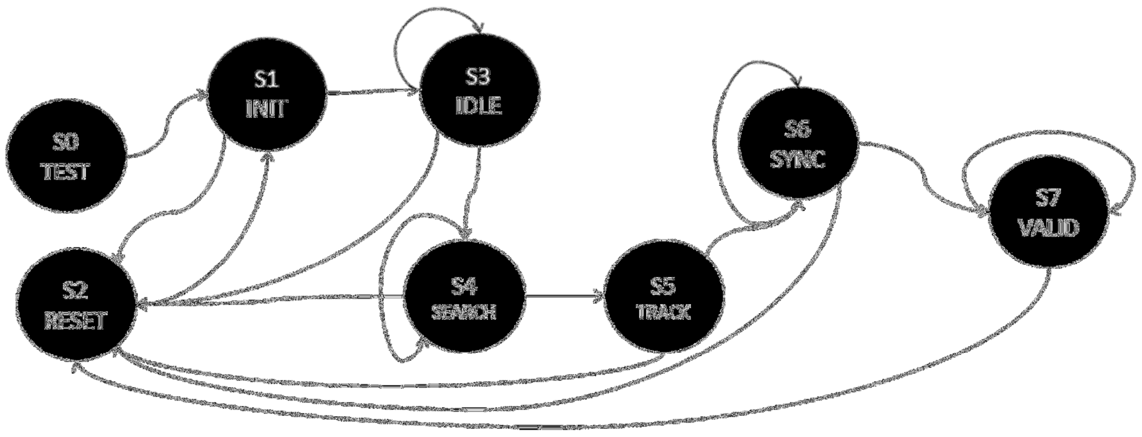


Figure 22 FPGA state machine

The compiler in Quartus II was able to compile the code and downloaded to the DE2 board. The realization of the C/A shift operations is shown in Figure 23.

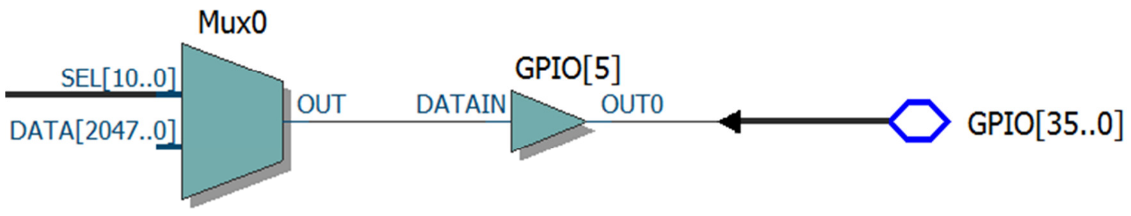


Figure 23 Compilers realization of a C/A shift register

In order to simulate the satellites code, an internal connection was made in VHDL which outputted code from SDRAM at a rate of 1.023 MHz just as the satellite would (after the RF front end). The C/A output was fed into the ca0 signal simulating the 'force' function in ModelSIM. The results of the code execution was exactly as expected. The shift register was able to lock onto the incoming C/A code and replicate the code synchronous with a 1.023 MHz PLL. Figure 24 shows the PLL and ca0_replicate output from an oscilloscope. This test simulated signal transmission time of 0 seconds, which only allowed the replication, tracking, and locking functions be tested. Future work will involve building analog components along with the necessary filters and antennas to receive signals from satellites.

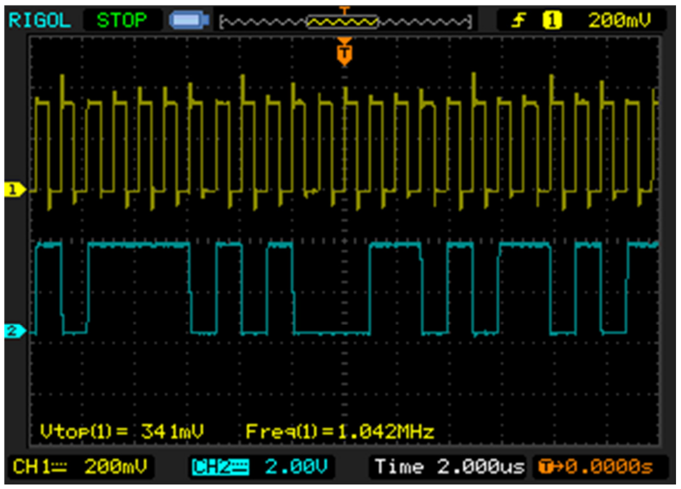


Figure 24 Hardware realization of correlation

A custom prototype board containing a GNSS receiver, antennas, and driver was designed and built in order to test real-world GPS signal and RF front end acquisition as shown in Figure 25.

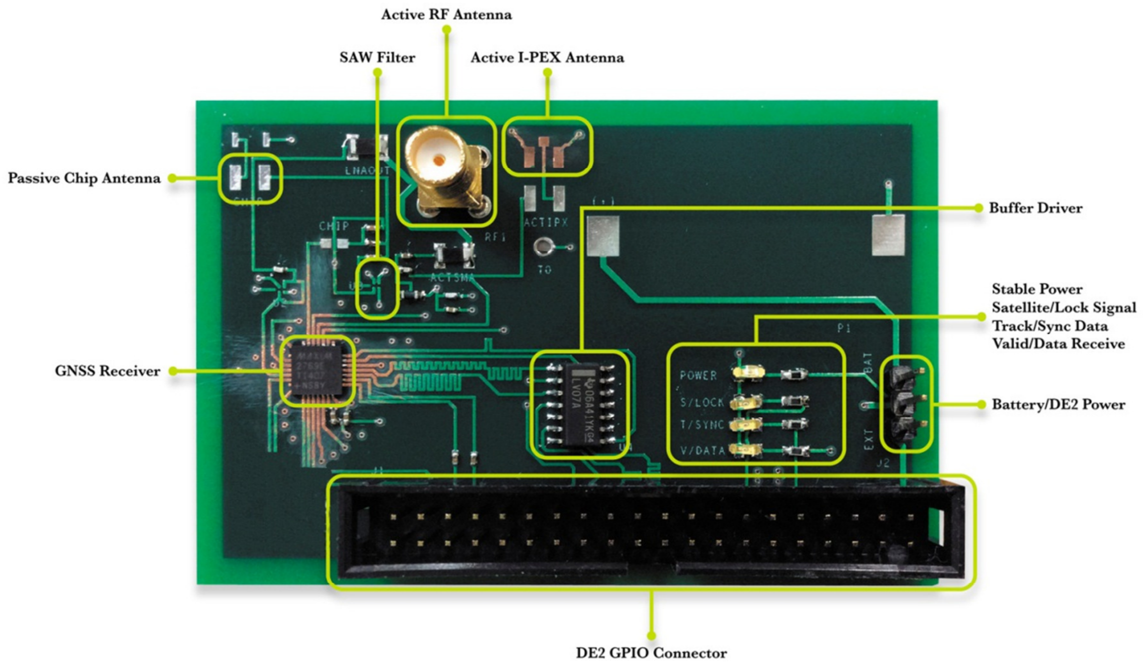


Figure 25 Analog GPS front end board

The complete system test setup is shown in Fig. 26, the DE2 board has 7-segment LED display indicating state 1 (S1), which is waiting for the valid GPS signals.

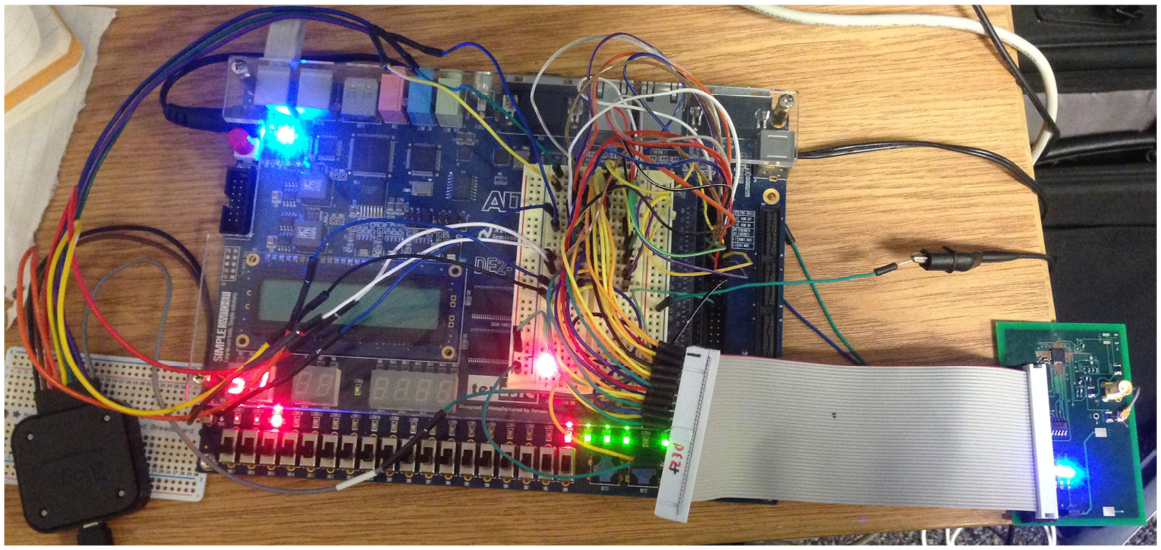


Figure 26 Complete system testing

Chapter 5 Conclusion

The purpose of this research was to explore the possibilities of using an FPGA to simulate a hardware-accelerated GPS parallel correlator and exceed the performance of a commercially available watch. Simulation results in Figure 27 show a valid solution using a hardware-defined approach at 1778 ms.

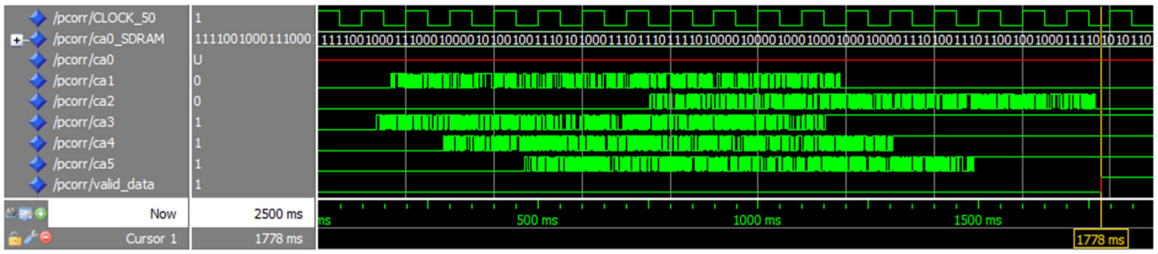


Figure 27 Hardware simulation results

Table 3 shows that hardware defined approach results are over 4 times faster than the commercially available watch.

Table 3 Simulation results comparison

	Garmin Forerunner 210	**Hardware-defined approach**
Startup time	8 seconds	1.7 seconds

This research shows the results of many lines of code spanning multiple languages in order to replicate GPS satellite C/A code, determine the visibility, correlate, and verify the model. The results show that it theoretically is possible to track multiple satellites in parallel using hardware and a single ASIC RF front end. A high precision GPS receiver can be produced with a small footprint and small power consumption. With the increasing complexity of the newest generation GPS satellites this high-performance programmable hardware can be re-programmed on the fly to accept new features of the next generation satellites. Selected code samples are shown in the appendix; however, the full code will not be released as there may be future commercial applications.   Future work will include making the program user friendly, verifying the code using actual hardware to include the analog components, and calculation of the actual navigation solution.

Appendix

Included in this appendix are sample codes from selected parts of the project. Full codes are not included for possible future testing and commercialization. The code samples included in this appendix are:

File Name	Language
ca_gen.m	MATLAB
to_modelsim.py	Python
pcorr.vhd	VHDL
satclk.vhd	VHDL

Table 4 Appendix code descriptions

A.1 ca_gen.m sample code

```
prnArr=[2 6;3 7;4 8;5 9;1 9;2 10;1 8;2 9;3 10;2 3;3 4;5 6;6 7;7 8;8 9;9 10;1 4;2 5;3
6;4 7;5 8;6 9;1 3;4 6;5 7;6 8;7 9;8 10;1 6;2 7;3 8;4 9 5 10 4 10 1 72 8 4 10];

psel = prnArr (sv,:);

for inc=1:L
    g2(:,inc)=mod(sum(q(psel),2),2);
    g(:,inc)=mod(g1(n)+g2(:,inc),2);
   g1=[mod(sum(g1.*s),2) g1(1:n-1)];
   q=[mod(sum(q.*t),2) q(1:n-1)];
end

svn1=cacode([3],1.023);
dlmwrite('svn1.out',svn1, ',')
svn2=cacode([7],1.023);
dlmwrite('svn2.out',svn2, ',')
svn3=cacode([23],1.023);
dlmwrite('svn3.out',svn3, ',')
svn4=cacode([16],1.023);
dlmwrite('svn4.out',svn4, ',')
svn5=cacode([4],1.023);
dlmwrite('svn5.out',svn5, ',')
svn6=cacode([27],1.023);
```

```
dlmwrite('svn6.out',svn6, ',')
```

A.2 to_modelsim.py sample code

```
def generate():
  file_out = open("sim_commands.txt","w")
  svn1 = open('svn1.out', 'r')
  svn2 = open('svn2.out', 'r')
  svn3 = open('svn3.out', 'r')
  svn4 = open('svn4.out', 'r')
  svn5 = open('svn5.out', 'r')
  svn6 = open('svn6.out', 'r')


  svn1_file_contents = svn1.read()
  svn1_delimited_list = svn1_file_contents.split(",")

  svn2_file_contents = svn2.read()
  svn2_delimited_list = svn2_file_contents.split(",")

  svn3_file_contents = svn3.read()
  svn3_delimited_list = svn3_file_contents.split(",")

  svn4_file_contents = svn4.read()
  svn4_delimited_list = svn4_file_contents.split(",")

  svn5_file_contents = svn5.read()
  svn5_delimited_list = svn5_file_contents.split(",")

  svn6_file_contents = svn6.read()
  svn6_delimited_list = svn6_file_contents.split(",")

  print("+delimited lists complete")

  #format:  value time, value time, value time,

  r = []
  r.append(0)

  t = 0 + t_delay_svn1
  step = 1/1.023

  print(len(svn1_delimited_list))
  n = 0

  for k in range(nRepeat):

    for i in range(len(svn1_delimited_list)):
      file_out.write(str(svn1_delimited_list[i]) + " " + str(t))
      n += 1

      if i < len(svn1_delimited_list) - 1:
        file_out.write(", ")

      t = t + step
      r.append(t)
    if k < (nRepeat - 1):
      file_out.write(", ")

  print(n)

  st = 29.3255 # step*30

  t = 0 + t_delay_svn1 + st
  step = 1/1.023

  print(len(svn1_delimited_list))
  n = 0

  for k in range(nRepeat):

    for i in range(len(svn1_delimited_list)):
      file_out.write(str(svn1_delimited_list[i]) + " " + str(t))
      n += 1

      if i < len(svn1_delimited_list) - 1:
        file_out.write(", ")


      if st > 0:
        st = st - step
        t = t
        print(st)
      else:
        t = r[i+1]

    if k < (nRepeat - 1):
      file_out.write(", ")
  print(r)
```

A.3 sim_commands.txt sample output

```
+delimited lists complete

This file was automatically generated by:
to_modelsim.py v.1.2.1 built: 10/06/14 15:03:33
M. Sammartino

+modelsim command:

add wave -position insertpoint  \
sim:/pcorr/CLOCK_50
add wave -position insertpoint  \
sim:/pcorr/clk_1p023m
add wave -position insertpoint  \
sim:/pcorr/ca0
add wave -position insertpoint  \
sim:/pcorr/ca0_mem
add wave -position insertpoint  \
sim:/pcorr/ca0_SDRAM
add wave -position insertpoint  \
sim:/pcorr/ca0_shift_control
```

```
add wave -position insertpoint  \
sim:/pcorr/ca0_replicate
add wave -position insertpoint  \
sim:/pcorr/ca0_replicate_idx
restart

force -freeze sim:/pcorr/CLOCK_50 1 0, 0 {50 ms} -r 100
force -freeze sim:/pcorr/clk_1p023m 1 0, 0 {1.023 ms} -r 2.046
force -freeze sim:/pcorr/ca0 0 0, 1 100, 1 100.97751710654937, 1 101.95503421309874, 1
102.93255131964811, 0 103.91006842619748, ….
```

A.4 vhdl_signals.txt sample output

```
This file was automatically generated by:
to_modelsim.py v.1.2.1 built: 10/06/14 15:03:33
M. Sammartino

signal ca0_SDRAM : std_logic_vector(1045 downto 0) :=
"11110010001110001000010100100111010100011101110111101000010000100010001000100001110100
111011001001000111101010110110011001101100100000000010110000111010111101111000010011011
1101111001101001010010100111010101101110001111001111010001000011111011111100101001011101
000000011010011101101011110010100101110000001000110011101001011110101101011000101011110
1000000110010011010001100011110011101100000000111110011110101011110110000111001010101011
101011110000110100010100110001101111011000110111011010110010111011011111100010000111101
00101000000111011101001101001111010011111110000001011010101100001110010111011010110010110
101011111110011001010110100001100101111100111101000100001111010011110010101101000100110
011111000001110110110011101110101001110000111000100111010100001110011100001101101001000
1110000011100010111110000110101010001110111010011101100001010100011000000101111000111000
01100011001011110000000011101001000010100011011111000111101100010001100000010100001011100
0001011100100001111000011011000110111000100111001011111100111111001100100000000101000100
100
";

+hex format 0x
signal ca0_SDRAM : std_logic_vector(263 downto 0) :=
"0x3c8e2149d4777a10888874ec91ead99b200b0ebde137bcd294eadc79e887df2974069daf2970233a5eb5
8af40c9a31e7603e7abd872abaf0d14c6f6376b2edf887a50774d3d3f82d5872ed65abf995a197cf443d3ca
d133e0ed9dd4e1c4ea1ce1b48e1c5f0d51dd3b0a8c0bc70c65e03a428df1ec460a1705c8786c6e272fcfcc8
0a24";
```

A.5 pcorr.vhd sample code

```
entity pcorr is

port(
          CLOCK_50 : in std_logic; --50 MHz clock
           ca0,ca1,ca2,ca3,ca4,ca5 : in std_logic; --incoming c/a    code
          SW : in std_logic_vector(15 downto 0); --Switch inputs
          KEY : in std_logic_vector(3 downto 0); --Key inputs
          LEDR: buffer std_logic_vector(15 downto 0); -- Red LED's
          LEDG: buffer std_logic_vector(7 downto 0);  -- Green
          SATCLK_OUT: out std_logic;
          GPIO: inout std_logic_vector(35 downto 0)
```

```
       );
end pcorr;

component satclk
       PORT(
              areset         : IN STD_LOGIC  := '0';
              inclk0         : IN STD_LOGIC  := '0';
              c0                   : OUT STD_LOGIC ;
              locked         : OUT STD_LOGIC      );
end component;



           process(clk_1p023m, ca0)
             begin

               if (clk_1p023m'event and (clk_1p023m = '1' or               clk_1p023m =
'0')  and ca0_save = '1' and ca0_write_control /= 0) then --and ca0_write_control <
1045 and ca0_save = '1') then
                 ca0_mem(ca0_write_control) <= ca0;
                 ca0_write_control <= ca0_write_control - 1;
               end if;


      satclk_inst : satclk PORT MAP (
      areset    => areset_sig,
              inclk0  => CLOCK_50,
              c0      => sat_clk,
              locked  => pllsatclk_locked_sig     );
```

A.6 satclk.vhd sample code

```
PORT (
                      areset : IN STD_LOGIC ;
                      clk    : OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
                      inclk  : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
                      locked : OUT STD_LOGIC
       );
       END COMPONENT;

BEGIN
       sub_wire5_bv(0 DOWNTO 0) <= "0";
       sub_wire5    <= To_stdlogicvector(sub_wire5_bv);
       locked    <= sub_wire0;
       sub_wire2    <= sub_wire1(0);
       c0    <= sub_wire2;
       sub_wire3    <= inclk0;
       sub_wire4    <= sub_wire5(0 DOWNTO 0) & sub_wire3;

       altpll_component : altpll
       GENERIC MAP (
              bandwidth_type => "AUTO",
              clk0_divide_by => 50000000,
              clk0_duty_cycle => 50,
              clk0_multiply_by => 1022999,
              clk0_phase_shift => "0",
              compensate_clock => "CLK0",
              inclk0_input_frequency => 20000,
intended_device_family => "Cyclone IV E",
lpm_hint => "CBX_MODULE_PREFIX=satclk",
lpm_type => "altpll",
operation_mode => "NORMAL",
pll_type => "AUTO",
port_activeclock => "PORT_UNUSED",
port_areset => "PORT_USED",
```

References

[1] Grewal, Mohinder S., and Lawrence R. Weill. “Global Positioning Systems, Inertial Navigation, and Integration”. 2nd ed. Hoboken, N.J.: Wiley-Interscience, ISBN: 978-0-470-09971-1, 2007.

[2] "Navigation Programs - Global Positioning System - Control Segment.", (12 Oct. 2014) Retrieved from http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/controlsegments/

[3] "Current GPS Satellite Data." NGA GPS Branch. National Geospatial-Intelligence Agency. (13 Oct. 2014) Retrieved from http://earth-info.nga.mil/GandG/sathtml/satinfo.html.

[4] French, Gregoty T. “Understanding the Gps: An Introduction to the Global Positioning System”, Onword Pr, ISBN: 978-1566902250, 1997.

[5] "GPS Applications." GPS.gov: Applications. National Coordination Office for Space-Based Positioning, (9 Sept. 2013). Retrieved from http://www.gps.gov/applications/

[6] Patil, Bharath, Radhika Patil, and Andre Pittet. "Energy Saving Techniques for GPS Based Tracking Applications,” Integrated Communications, Navigation and Surveilance Conference (ICNS), pp.J8-1,J8-10, 10-12 May 2011, doi: 10.1109/ICNSURV.2011.5935335, 2011

[7] "Global Positioning System Standard Positioning Service Signal Specification” (12 Apr 15) Retrieved from http://www.navcen.uscg.gov/pubs/gps/sigspec/gpssps1.pdf

[8] Groves, Paul D. “Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems.” Boston: Artech House, ISBN: 978-1608070053, 2008.

[9] Mulla, A.; Baviskar, J.; Baviskar, A.; Bhovad, A., "GPS assisted Standard Positioning Service for navigation and tracking: Review & implementation," Pervasive Computing (ICPC), 2015 International Conference on , vol., no., pp.1,6, 8-10 Jan. 2015 doi: 10.1109/PERVASIVE.2015.7087165

[10] Shapiro, Adam. "FPGA-BASED REAL-TIME GPS RECEIVER." (2010). Retrieved from http://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2009to2010/ams348/hw_gps_receiver.pdf

[11] Alaqeeli, Abdulqadir. "Global Positioning System Signal Acquisition and Tracking Using Field Programmable Gate Arrays." (2002). Retrieved from http://www.ohio.edu/people/starzykj/network/Research/Thesis/Abdul_Al_Aqeeli_dissertation.pdf

[12] "Space Segment." *GPS.gov: Space Segment*. National Coordination Office for Space-Based Positioning, 2 Aug. 2014. Web. 8 July 2014. Retrieved from http://www.gps.gov/systems/gps/space/

[13] Tsui, James Bao. “Fundamentals of Global Positioning System Receivers: A Software Approach.” New York: Wiley, ISBN: 978-0471706472, 2000.

[14] French, Gregory T. *Understanding the GPS: An Introduction to the Global Positioning System : What It Is and How It Works*. Bethesda, MD: GeoResearch, ISBN: 0-9655723-0-7, 1996.

[15] Boschen, Dan. "GPS C/A Code Generator - File Exchange - MATLAB Central." *GPS C/A Code Generator*. Mathworks, 1 June 2010. (5 Apr 15) Retrieved from mathworks.com/matlabcentral/fileexchange/14670-gps-c-a-code-generator/content/cacode.m

[16] "Control Segment." *GPS.gov: Control Segment*. National Coordination Office for Space-Based Positioning, 17 Sept. 2014. Web. (8 Apr 14) Retrieved from http://www.gps.gov/systems/gps/control/

[17] "Data Sheet SE4150L." *SE4150L: GPS Receiver IC*. (8 Sept 14) Retrieved from http://www.skyworksinc.com/uploads/documents/202445A.pdf

[18] "Nios (computer Processor)." Altera. (17 Jun 14) Retrieved from ftp://ftp.altera.com/up/pub/Altera_Material/10.1/Tutorials/VHDL/DE2/Using_the_SDRAM.pdf

[19] Samper, Jaizki, and Roc Berenguer. “GPS & Galileo Dual RF Front-end Receiver and Design, Fabrication, and Test” New York: McGraw-Hill, ISBN: 978-0071598699, 2008.

[20] Tsui, James Bao. “Fundamentals of Global Positioning System Receivers: A Software Approach” New York: Wiley, ISBN: 978-0471706472, 2000.

[21] Norton, Peter. “Beginning Python”. Indianapolis, IN: Wiley Pub, ISBN: 978-0764596544, 2005.

[22] Gupta, Rashi. “Making Use of Python” New York: John Wiley & Sons, ISBN: 072-3812219759, 2002.

[23] Lundh, Fredrik. “Python Standard Library” Beijing: O'Reilly, ISBN: 063-6920000969, 2002.

[25] Huang Yan; Yao Hejun; Gao Yuan; Zhang Han; Xu Yuan, "Development of the high real-time GPS time transfer receiver," Precision Electromagnetic Measurements (CPEM 2014), 2014 Conference on , vol., no., pp.150,151, 24-29 Aug. 2014 doi: 10.1109/CPEM.2014.6898303