

Report Title

Development of a Simulink Library for the Design, Testing and Simulation of Software Defined GPS Radios. With Application to Parallel Correlator Structures

ABSTRACT

The GPS, Software Defined Radio, Simulink Library includes at the moment 7 blocks, as shown in Figure 2, which are used to process and simulate raw GPS signals. This library was designed to help in the development of algorithms that will execute faster GPS synchronization. Having a platform where incoming signals parameters can be changed offers a solid base to test new algorithms for GPS SDR-based technology. Parallel correlator structures were developed and tested using modelled data.

Technical Report
W911NF-11-1-0180
58923-CS-REP

Development of a Simulink Library for the Design, Testing and Simulation of Software Defined GPS Radios

With Application to the Development
of Parallel Correlator Structures

By

Damian Miralles
(Undergraduate Research Assistant)

Jennifer Sandoval
(Undergraduate Research Assistant)

Manuel Ortiz
(Undergraduate Research Assistant)

Marvi Teixeira
mteixeir@ieee.org
(Principal Investigator)

W911NF-11-1-0180
58923-CS-REP

Polytechnic University of Puerto Rico

5/1/2014

Contents

1. Introduction	1
2. Global Navigation Satellite System Overview	2
3. Software Defined Radio for Global Navigation Satellite Systems Receivers	4
4. Library Description	6
4.1 –GPS PRN Generator Block.....	7
4.2- Selector.....	11
4.3- Upsampler Block.....	12
4.4- Circular Shift Block	12
4.5- 1, 0 Level Signaling to 1, -1 Level Signaling.....	13
4.6- C/A Code Generator Block	14
4.7- Code Phase Acquisition Block.....	15
4.8 Baseline Parallel Code Search (Doppler Correction Included)	17
4.9- Frame Based Signal Generator (Any Sampling Frequency).....	18
5. Using the library. Installation and useful tips.....	22
5.1- Overview	22
5.2- Adding a Custom Library to the Library Browser	22
5.3- Features of a library	23
5.4- Tips of usage	23
6. Parallel Structures Overview	24
7. Circular Correlation using the FFT	24
8. Metric Definition for Complexity Assessment.....	25
9. Decomposition of Circular Correlation into Smaller Sub-Correlations	25
Using the IDFT we can obtain the even an odd components of the circular convolution in the time domain, as expressed in Equations (5).....	26
10. Parallel Circular Correlation Structures	26
10.1- Baseline Parallel Code Search (PCS).....	27
10.2- Parallel Architecture 1 – Requires the use of a NCO.....	27
10.3- Architecture 2 – Requires a circular shift in the discrete time domain.	29
10.4- Architecture 3 – Requires the use of a NCO.	31
10.5- Architecture 4 – Requires the use of a circular shift in the discrete time domain.....	32
11. References	34
12. Appendix	1

12.1 Simulink Model for Base Line Method	1
12.2 Simulink Model for Architecture 1	2
12.3 Simulink Model for Architecture 2	3
12.4 Simulink Model for Architecture 3	4
12.5 Simulink Model for Architecture 4	5
12.6 Matlab Code for a Standard Baseline Implementation Using Basic Modeled Data.....	6

1. Introduction

Global Navigation Satellite Systems (GNSS) is the standard generic term for satellite navigation systems or “sat nav”. The GNSS allows electronic receivers to determine their location (longitude, latitude and height) with a precision of a few meters using time signals transmitted along a line-of-sight by radio on satellites. The receivers can also determine the precise time.

Most of the already available GNSS receivers utilize the Global Navigation System (GPS) but there other GNSS system in development. Most receivers utilized nowadays are built using hardware to perform most of the tasks necessary to give a position and time to the users. Some of this hardware parts can be replaced using another type of architecture. Recent approaches are more focused on developing a software solution that reduces the hardware implementation by taking code as close to the antenna as possible. The objective of this work is to develop a set of tools that can be used to test and implement more affordable and robust GPS receivers.

One of the main advantages of the software defined receiver is the flexibility of the design, because it allows reconfiguring the software to change parameters that will otherwise incurred in high cost if performed in hardware.

2. Global Navigation Satellite System Overview

In 2010, the only fully operational GNSS is the United States GPS. The Russian (GLONASS) is another GNSS in the process of being restored to full operation. The European Union's Galileo positioning system is a GNSS in initial deployment phase, scheduled to be operational in 2014. China will expand its regional Beidou navigation system into the Compass navigation system by 2020¹. [1].

The GPS (operational fleet of the US) uses the concept of measurement time-of-arrival (TOA) to compute a receiver position [2]. For this method to work, the position of all transmitters needs to be known and the receiver and transmitter's clocks need also to be synchronized. From TOA, the propagation time can be computed. With this time it is possible to get a range estimate to the transmitter, multiplying the propagation time by the speed of light. At this time, assuming the location of transmitters are known, the receiver can compute its position.

The GPS system consists on a constellation of nominally 24 satellites (29 actual satellites) with an orbit radius of 26,650Km, giving the satellites a period of approximately 12 hours. All satellites have highly synchronized Rubidium or Cesium atomic clocks as a frequency reference. The satellites broadcast Code Division Multiple Access (CDMA) ranging codes and navigation data on two frequencies (L1) (1575.42MHz) and (L2) (1227.6 MHz) [1]. All satellites broadcast in the same frequencies but use different codes to differentiate themselves, these codes are described in future sections.

Each satellite broadcasts navigation data that allows the receivers to compute the satellite position and transmission time. The ranging code is used to determine the propagation time of the signal. If the receiver clock was synchronized with the transmitter clock, only three ranging measurements are needed for a 3D position solution, but since the receiver clock is usually drifted with respect to the transmitter clock, four measurements are needed to solve for longitude, latitude, height and receiver clock offset.

The GPS satellite's carrier frequency at L1 and L2 are then modulated with the ranging codes and navigation data using bi-phase shift keying (BPSK). Each satellite generates two different ranging codes, the civilian Coarse Acquisition code (C/A) code and the military P(Y) code. These are modulated onto the carrier's frequencies at 1.023MHz and 10.23MHz respectively. The navigation data is also modulated on the carriers at a 50bps rate [1]. This navigation message contains almanac information that it is later use to determine the satellite position and time of transmission of the signal. The L1 signal carries both civilian and military codes, while the L2 signal is only modulated with the military code. For the purpose of this document only the L1 signal would be discussed and analyzed.

Modern digital receivers are typically made of three parts: a radio frequency (RF) front-end, a digital baseband processor, and a Digital Signal Processing (DSP) module. The function of the RF front-end is to convert the signal frequency to an intermediate frequency (IF) that is much lower so that the signal can be sampled by an analog-to-digital converter (ADC). The digital baseband processor mission is to take care of removing the residual carrier and the PRN code (despreading the signal), this stage is normally implemented by an FPGA for the purpose of gaining processing speed. This stage is, however, tightly coupled with the DSP module. The DSP module examines the output of the baseband processor to determine if the incoming signal is valid for position tracking. If the signal is not being tracked correctly or if it is suitable to perform a position lock, then the DSP module will repeat the process until a valid signal is lock on the digital receiver.

¹ This data may need updating at the time of reading.

Nowadays there is an increasing interest in the development of software defined receivers. In this type of receivers, the objective is to move a generic processor as close to the antenna as possible. This permits most of the components that are traditionally designed as an Application Specific Integrated Circuit (ASIC) to be replaced by signal processing algorithms running in software. The main idea is to a software defined system as close as possible to the antenna.

Traditionally what have been used are hardware receivers, in which the design is specific for the required functionality but involve higher cost of design and a flexibility factor of 0. These receivers are normally faster and happen to be a pretty solid tool in today's market. On the other hand software receivers tend to be slower than their non-reconfigurable hardware counterparts (ASIC) but the ability of being constantly redesigned opens a good channel for optimization and improvement.

The software defined GPS receivers are especially important because they allow the testing and implementation of non-standard algorithms such as weak signal tracking, fast acquisition and others. It is also an important feature because it can reduce development time of new algorithms since the research does not have to wait for a hardware implementation. The type of receiver utilized in this work is the software defined GNSS receiver.

3. Software Defined Radio for Global Navigation Satellite Systems Receivers

“Software-radio is an emerging technology, which seeks to build flexible radio systems, multi-service, multi-band, multi-standard, reconfigurable, and reprogrammable by software.” This definition, borrowed from (Principe, Bacci, Giannetti, & Luise, 2011), describes very well the key idea of the software-defined radio (SDR) design paradigm. More specifically, an ideal SDR transceiver, sketched in Figure 1, is characterized by a commodity off the-shelf (COTS) front-end (FE) that is connected to a programmable processing unit (PU), which processes the baseband (BB) incoming, or transmitted, signal via flexible software. The PU is the core of the system, since it can impart multiple personalities to the transceiver by running specific software, depending on the application context and the required functionalities.

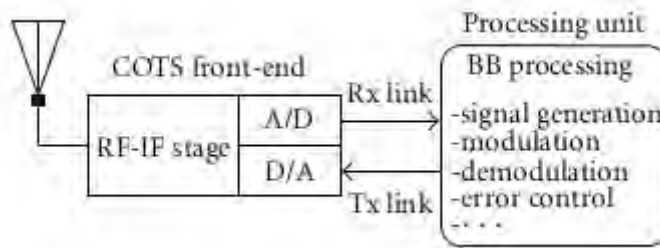


Figure 1 General SDR architecture

The effectiveness and the advantage of a fully SW-defined radio like the one envisaged in Figure 1 with respect to a fully hardware equivalent design can be synthesized in three key points (Principe et al., 2011).

1. **Configurability.** Within a specific communication technology, this means that a unique transceiver for different applications can be factory-configured without any hardware change.
2. **Updatability/Upgradeability.** Since the core is software based, the receiver can be updated and upgraded in case of new and improved algorithms or amendments to a standard, while maintaining the same hardware components.
3. **Flexibility.** The same transceiver, with the same hardware, can be used for different communication technologies, simply by reconfiguring or upgrading the software.

Obviously, these properties are not in general borne to a large extent in fully, non-reconfigurable, HW devices, which are usually constrained and confined to particular applications. The direct consequence of this limitation is, on the one hand, the need to use multiple HW solutions for different technologies and, on the other hand, the possible need of HW replacements at the advent of even modest changes in standards or technologies. Still, we must see how the general notion of an SDR can be beneficial to the field of GNSS.

Satellite navigation has witnessed the advent of a number of new systems and technologies: after the landmark design and development of the GPS. Recently, new features have been studied to improve the performance of the new GNSS, based on the experience of GPS. Multi-personality receivers switch from one satellite system to another and process their signals performing multi-constellation navigation, and at the horizon there is also the integration of satellite navigation with different positioning data coming from different, possibly local, sensors (indoor or inertial navigation). In this framework, an SDR-based receiver has no difficulty in performing the necessary functions of data fusion among different sources of information.

All of the above represents, in the authors' opinion, a very well-founded motivation that led the application of the SDR paradigm to the design of GNSS receivers. The work presented below intends to provide a set of functional tools that would facilitate the design, modeling and simulation of new GNSS techniques within the Simulink environment.

4. Library Description

The GPS, Software Defined Radio, Simulink Library includes at the moment 7 blocks, as shown in Figure 2, which are used to process and simulate raw GPS signals. This library was designed to help in the development of algorithms that will execute faster GPS synchronization. Having a platform where incoming signals parameters can be changed offers a solid base to test new algorithms for GPS SDR-based technology.

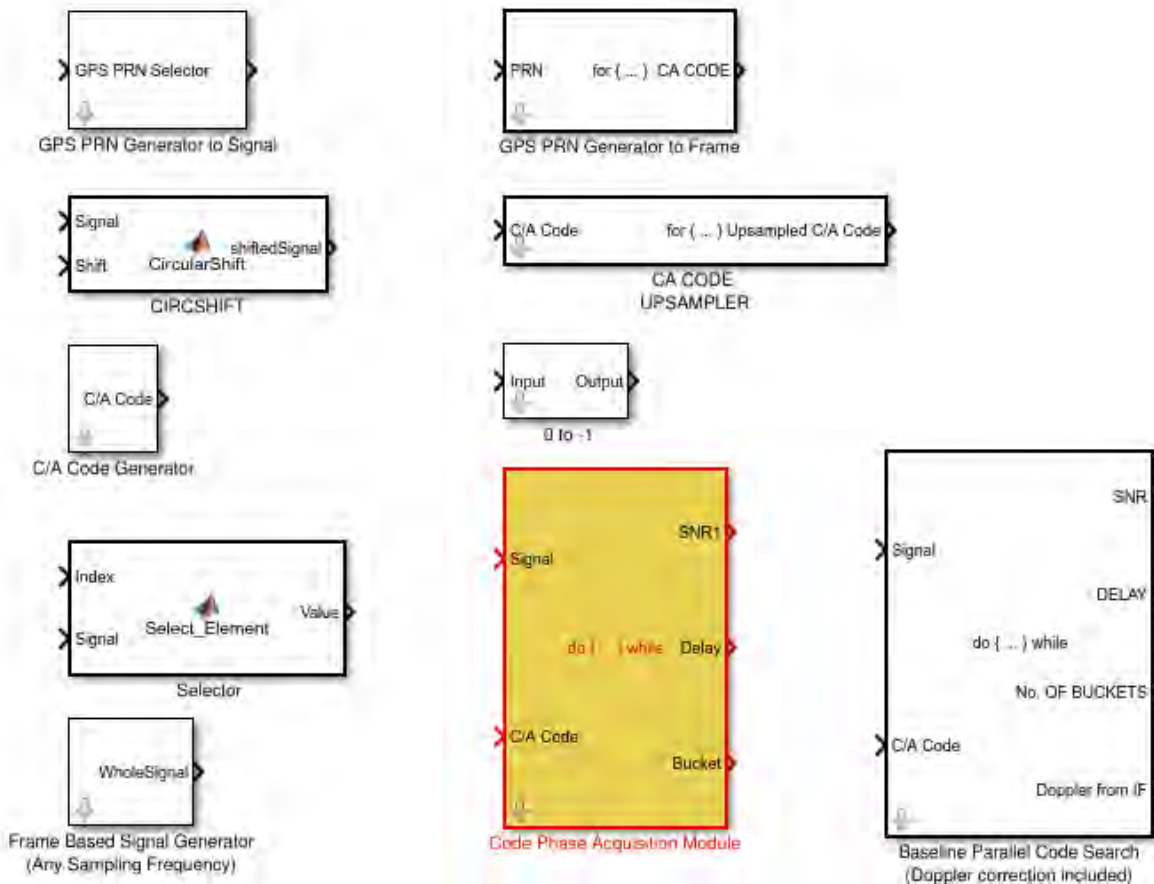


Figure 2 GPS Software Defined Library Modules

Summary of Block Functionality (inline left to right description):

1. GPS PRN Generator to Signal: The block generates the 1023 bits long C/A sequence. The input to the block is a constant value that specifies the desired PRN code. The allowed input values are between 1 and 37. The block functions as a continuous system dependent on the Simulink's simulation time. In order to work properly the system requires 1023 time steps to generate the desired chip for the C/A code.
2. GPS PRN Generator to Frame: The block generates the 1023 bits long C/A sequence. The input to the block is a constant value that specifies the desired PRN code. The allowed input values are between 1 and 37. The block generates a frame based output of 1023 samples not dependent on

the simulation time. It is recommended to set the simulation parameters to execute a single step simulation in order to generate the desired code.

3. **Circular Shift Block:** The block performs a circular shift of the Signal by the amount specified by Shift, the block is equivalent to the "circshift" function of MATLAB.
4. **C/A Code Upsampler Block:** The block performs an up sampling to the input PRN signal. Once the desired sampling frequency is specified the block interpolates the input samples of the PRN code until the desired sampling frequency is achieved. The block needs to receive an input signal of 1023 samples for the desired GPS satellite, and the output signal would have a length of $f_s/1000$ samples corresponding to the up-sampled signal.
5. **C/A Code Generator:** The block was created as a cosmetic tool to create the C/A code of a single PRN given any value of the sampling frequency. This block is ideally used for generation of the local pulse for cross-correlation operations.
6. **Change 0 by -1:** The blocks perform the conversion of the 0 values in the PRN code to -1. This is used to represent the series as in the polar non-return-to-zero format
7. **Selector:** The blocks select an element given an index in any vector of matrix input at Signal port. The block has been tested extensively with one column vector or one row vector
8. **Code Phase Acquisition Module:** The block develops a code phase acquisition algorithm to detect GPS code delay, the block parallelized the code phase search in the frequency domain reducing the number of steps applied to find the desired shift.
9. **Baseline Parallel Code Search (Doppler correction included):** The block implements the Parallel Code Search Acquisition method of a real time raw GPS signal. The block receives two input parameters, where signal is the raw gps signal and C/A code is the local generated code used as a method of comparison.
10. **Frame Based Signal Generator (Any Sampling Frequency):** When implementing the signal processing parts of the GPS receiver, it is necessary to have available some modeled data for testing their functionality under different noise and interference conditions. Even though the block does not include the Navigation data it can be used to test and develop different algorithms to implement the correlator stage in the GPS-SDR receiver. Set the list of satellites under view at the time, indicating with the first index the desired satellite, while the rest are considered as interference satellites. The Satellite Delay parameter would specify the phase delay on the target satellite specified in the first position of the Satellite's List. The Interference Delay Vector specify the phase code for the remaining satellites of the group, notice that the first index of the list is conceived as the multipath delay on the simulation². The gain vector specify a different attenuation for each satellite signal in order to model a more realistic scenario. AWGN is added to the modeled signal by specifying the SNR parameter. Finally, the desired sampling frequency parameter would be used to generate the code given any value of the sampling frequency.

4.1 –GPS PRN Generator Block

The pseudorandom noise (PRN) codes transmitted by the GPS satellites are deterministic sequences with noise-like properties. Each C/A code is generated using a tapped linear feedback shift register (LFSR).

The LFSR is a shift register whose input bit is a linear function of its previous state. It is most often a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift

² The current simulation assumes interference of only 1 multipath signal, parameter may need changes.

register value. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle [4].

The LFSR generates a maximal-length sequence of length $N = 2^n - 1$ elements. A Gold code is the sum of two maximum-length sequences, known as G1 and G2. The GPS C/A code uses $n = 10$. The sequence $p(t)$ repeats every millisecond so the chip length is $1 \text{ ms}/1023 = 977.5 \text{ ns} \approx 1 \mu\text{s}$, which corresponds to a metric length of 300 m when propagating through vacuum or air. The C/A code generator contains two shift registers known as G 1 and G 2. These shift registers each have 10 cells generating sequences of length 1023. The two resulting 1023 chip-long sequences are modulo-2 added to generate a 1023 chip-long C/A code, only if the polynomial is able to generate code of maximum length. Every 1023rd period, the shift registers are reset with all ones, making the code start over. The G 1 register always has a feedback configuration with the polynomial.

$$f(x) = 1 + x^3 + x^{10} \quad (1)$$

This means that state 3 and state 10 are fed back to the input. In the same way, the G 2 register has the polynomial:

$$f(x) = 1 + x^2 + x^3 + x^6 + x^8 + x^{10} \quad (2)$$

To make different C/A codes for the satellites, the output of the two shift registers is combined in a very special manner. The G 1 register always supplies its output, but the G 2 register supplies two of its states to a modulo-2 adder to generate its output. The selection of states for the modulo-2 adder is called the phase selection. Table 1 shows the combination of the phase selections for each C/A Code. It also shows the first 10 chips of each code in octal representation. [5]

Table 1 Code phase assignment

Satellites ID number	GPS PRN signal number	Code phase selection G2	Code delay chips	First 10 chips octal
1	1	2 xor 6	5	1440
2	2	3 xor 7	6	1620
3	3	4 xor 8	7	1710
4	4	5 xor 9	8	1744
5	5	1 xor 9	17	1133
6	6	2 xor 10	18	1455
7	7	1 xor 8	139	1131
8	8	2 xor 9	140	1454
9	9	3 xor 10	141	1626
10	10	2 xor 3	251	1504
11	11	3 xor 4	252	1642
12	12	5 xor 6	254	1750
13	13	6 xor 7	255	1764
14	14	7 xor 8	256	1772
15	15	8 xor 9	257	1775
16	16	9 xor 10	258	1776
17	17	1 xor 4	469	1156
18	18	2 xor 5	470	1467
19	19	3 xor 6	471	1633
20	20	4 xor 7	472	1715
21	21	5 xor 8	473	1746
22	22	6 xor 9	474	1763
23	23	1 xor 3	509	1063
24	24	2xor 4	512	1706
25	25	5 xor 7	513	1743
26	26	6 xor 8	514	1761
27	27	7 xor 9	515	1770
28	28	8 xor 10	516	1774
29	29	2 xor 6	859	1127
30	30	2 xor 7	860	1453
31	31	3 xor 8	861	1625
32	32	4 xor 9	862	1712
-	33	5 xor 10	863	1745
-	34	4 xor 10	950	1713
-	35	1 xor 7	947	1134
-	36	2 xor 8	948	1456
-	37	4 xor 10	950	1713

Figure 3 shows the block diagram for the generation of the PRN

1. Notice that the terms of the G2 polynomial that are set are the ones that determine that next state in the shift register and that the output of the shift register is determined by the particular PRN that is desired to generate. In this case PRN #1 code would be generated after 1023 iterations of the shift register.

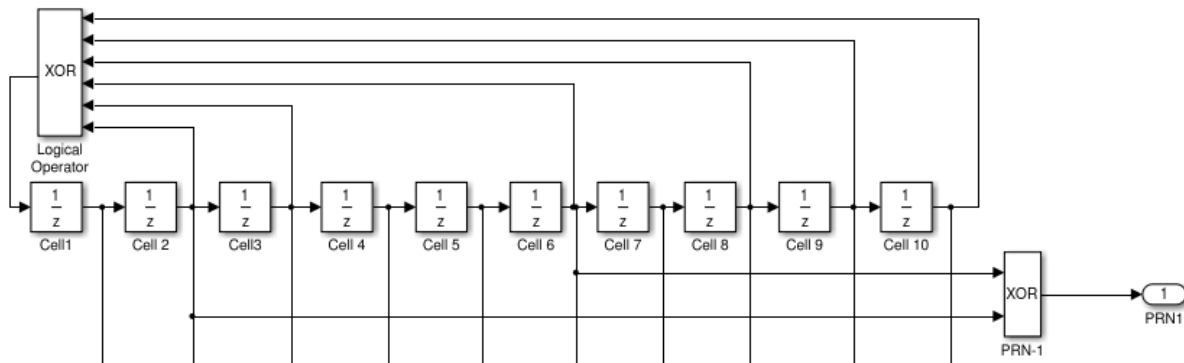


Figure 3 Block Diagram for the G2 Shift Register. The output from G2, that determines different PRN is set PRN 1, since cell 2 and 6 are xored. See table above for details

However Simulink offered a shortcut into this code generation for a much comfortable implementation. By using the PN Sequence Generator Block such a behavior may be achieved. The PN Sequence Generator block generates a sequence of pseudorandom binary numbers using a linear-feedback shift register (LFSR). This block implements LFSR using a simple shift register generator (SSRG, or Fibonacci) configuration. A pseudo noise sequence can be used in a pseudorandom scrambler and descrambler. It can also be used in a direct-sequence spread-spectrum system. The Block has three main features that need to be pointed out.

1. The 'Generator polynomial' parameter values specify the polynomial that determines the shift register. The values should be entered as either a binary vector or a descending-ordered polynomial³.
2. The initial states specifies the values in which each cell should start at the moment of initialization, for the L1 signal these parameters must be initialized to 1.
3. The Output mask source indicates how the output is determined for the shift register, as a Dialog parameter for constant configuration or as input port for modifiable operation during execution.
4. The 'Output mask vector' is a binary vector corresponding to the shift register states that are to be Xored to produce the output sequence values. These parameters must meet the criteria specified in Table 1 to generate the particular satellite code. Notice that for the G1 register the output is the one determined by the left most cell as can be seen in Figure 4, by defining the output mask vector as "0000000001". Something equivalent happens with the G2 register, only that the mask vector is defined as constants variables of 10 bit by setting only the bit positions present in the xored combination defined in Table 1 for each specific PRN.

3 Note that the polynomial entered in Illustration 4 is different from the one defined in Equation 1, this happens because the LFSR used by the block makes the output to the left instead of to the right, for that reason it is only needed to interchange the terms of the polynomial to meet the Simulink specifications. It is however the same polynomial.

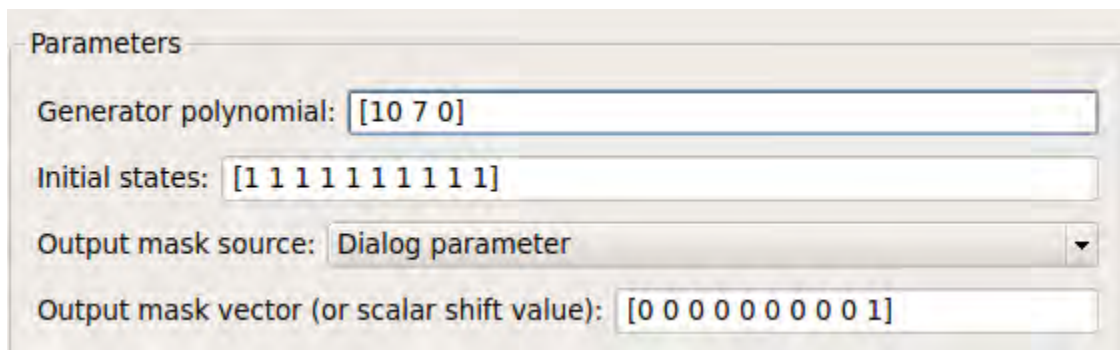


Figure 4 Parameters specification for the G1 Shift Register

Finally it is important to point out that the library contains two flavors of the code generation, named:

1. GPS PRN Generator to Signal. The block is dependable on the simulation time of the code. This code needs N time steps to produce an N long signal, this block is used as a visual aid on the generation and for successful results it is recommended to set a fixed simulation step size of 1023 steps. See Figure 5(left).
2. GPS PRN Generator to Frame. The block is identical to the previous one, with the only difference that the block is not dependent on the simulation time, the block is part of a for-loop subsystems that iterates 1023 times to produce a signal of 1023 samples equivalent to that generated by the GPS fleet. See Figure 5(right).



Figure 5 from left to right, the two different flavors of the C/A Code generator

4.2- Selector

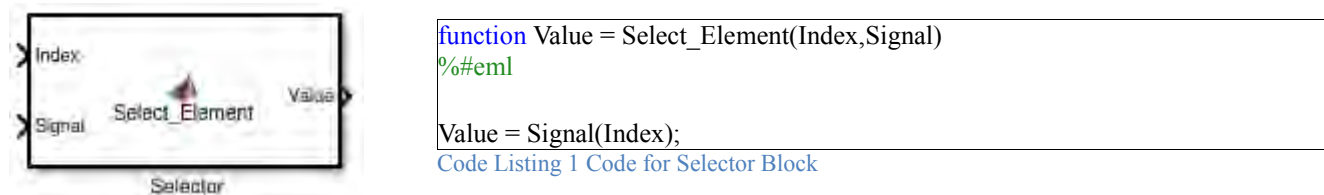


Figure 6 Selector Block Diagram

The Selector block follows the same concepts as the circshift block, it is just a simple tool to integrate into the Simulink environment a useful tool as the selection of an element in column or row based vectors. Input and output data are specified to the MATLAB Function block in the function header as arguments and return values, as in Code Listing 1. The argument and return values of the preceding example function correspond to the inputs and outputs of the block in Figure 6. Data, input triggers, and function call outputs can also be defined by using the Ports and Data Manager, which can be accessed from the MATLAB Function Block Editor by selecting Edit Data. The MATLAB Function block generates efficient embeddable code based on an analysis that determines the size, class, and complexity of each variable.

4.3- Upsampler Block

The up sampler block models a real life situation in which the incoming signal is sampled by an ADC at a much higher rate than the original frequency of 1.023MHz. This creates a digital version of the raw signal. A good approach to model the output of the ADC stage would be to up-sample the original signal by repeating each of its 1023 samples as many times as the ratio between the new sampling frequency and the original frequency of 1.023 MHz. Figure 7 shows how the up sampling was performed in the Simulink environment. Notice that the ratio between the new sampling frequency and the original frequency is multiplied by the iteration variable of the "for-loop subsystem" that goes from 1 to 1023. The result of the multiplication is taken and a ceiling operation is performed to get an index. This index would be used to extract the element of the 1023 chip long incoming signal and output it to the upsampled sequence.

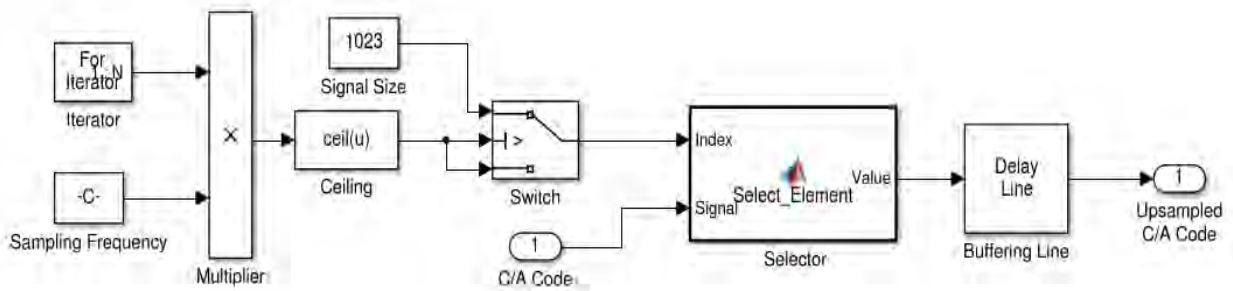


Figure 7 Upsampler Block Diagram

The output of the block would be a signal of size: new sampling frequency/1000 and, therefore, the algorithm within the block emulates the output after an ADC sampling stage in a real time application.

4.4- Circular Shift Block



```
function shiftedSignal = fcn(signal,Shift)
%#eml
shiftedSignal = circshift(signal,Shift);
Code Listing 2 Code for CircShift Block
```

Figure 8 CircShift's Block Diagram

The circular shift block was created to allow the "circshift" operation inside the Simulink environment. $B = \text{circshift}(A, \text{shiftsize})$ circularly shifts the values in the array, A , by shiftsize elements. Shiftsize is a vector of integer scalars where the n -th element specifies the shift amount for the n -th dimension of array A . If an element in shiftsize is positive, the values of A are shifted down (or to the right). If it is negative, the values of A are shifted up (or to the left). If it is 0, the values in that dimension are not shifted.

To export a function into the Simulink environment a MATLAB Function block was used. With this block you can write a MATLAB® function for use in a Simulink® model. The MATLAB function you create, it is executed in simulation as a native block and it automatically generates code for the Simulink

Coder™ target⁴.

The block is used by specifying input and output data to the MATLAB Function block in the function header as arguments and returns values, as shown in Code Listing 2. The argument and return values of the preceding example function correspond to the inputs and outputs of the block in Figure 8. Data, input triggers, and function call outputs can also be defined by using the Ports and Data Manager, which can be accessed from the MATLAB Function Block Editor by selecting Edit Data. The MATLAB Function block generates efficient embeddable code based on an analysis that determines the size, class, and complexity of each variable.

4.5- 1, 0 Level Signaling to 1, -1 Level Signaling.

After code generation, the codes are combined with the navigation data through modulo-2 adders. The exclusive OR operation is used on binary sequences represented by 0's and 1's, and its properties are shown in Table 2.

Table 2 Output of exclusive OR operation

Input	Input	Output
0	0	0
0	1	1
1	0	1
1	1	0

If the binary sequences were represented by the polar non-return-to-zero representation, i.e., 1's and -1's, ordinary multiplication could be used instead. The corresponding properties of the multiplication with two binary non-return-to-zero sequences are shown in Table 3.

Table 3 Output of binary multiplication

Input	Input	Output
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

The internals of the block is presented in the illustration below. A simple operation of multiplication by 2 following by adding -1 will ensure that each 0 in a binary sequence will be changed by -1.

⁴ Review Simulink documentation from the MathWorks website.

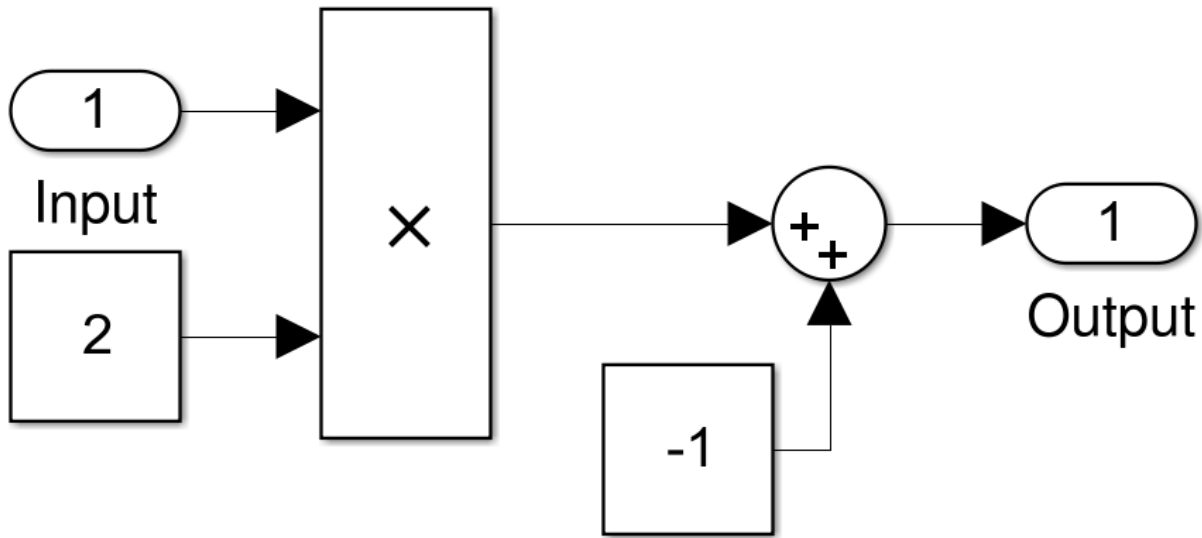


Figure 9 Change 0 by -1 block diagram

4.6- C/A Code Generator Block

The block is just a cosmetic tool to develop a signal acquisition procedure in the frequency domain. This method is based on the traditional approach of circularly convolving the incoming signal with the PRN code of the satellite. However frequency counterpart method, explained in detail later, basically leverages the fact that convolution in the time domain corresponds to multiplication in the frequency domain[6]. The block simply separates the generation of the PRN code for easy setup of simulations.

The block is just as mentioned above a cosmetic tool that was created for purposes of simpler implementation, as Figure 10 shows the block is formed of two previously documented blocks, for more information on this refer to 4.1 –GPS PRN Generator Block and 4.3- Upsampler Block .

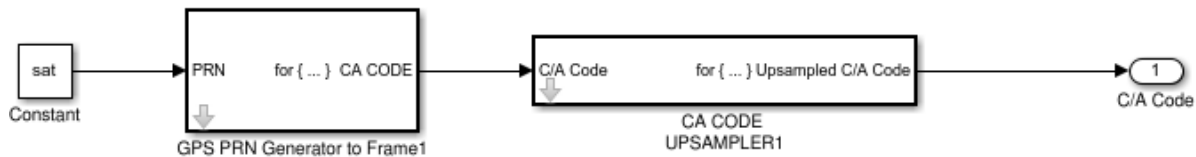


Figure 10 C/A Code's Block internal structure

Figure 11, shows the parameter list of the block, and it requires only specifications of two fields:

1. Sampling Frequency (Hz): The value entered here is just the desired sampling frequency at which the ADC would be working, and so the data rate specification for the model.
2. The satellite's PRN that the user wishes to simulate. Accepted values here range between 1 and 37.

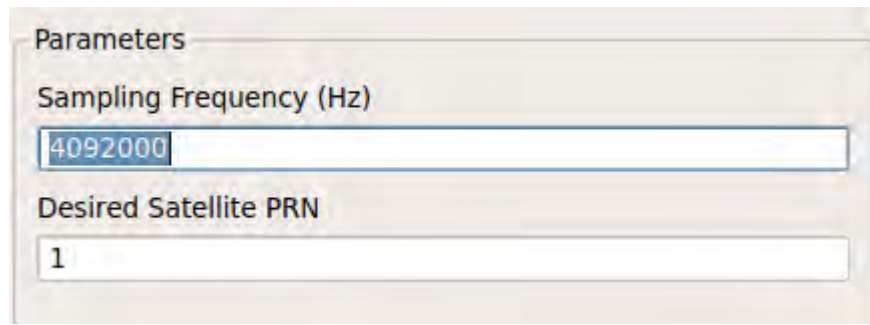


Figure 11 Parameters specification for the C/A code

4.7- Code Phase Acquisition Block

The traditional approach convolves the received signal with the CDMA code of each satellite in the time domain. The correct alignment corresponds to the one that maximizes this convolution. This approach has a computational complexity of $O(n^2)$, where n is the number of samples. More recently GPS receivers lock on the satellite using frequency domain computation. This approach leverages the fact that convolution in the time domain corresponds to multiplication in the frequency domain. It proceeds in the following three steps, shown in:

1. The receiver takes the FFT of the received signal.
2. It multiplies the output of this Fourier transform by the FFT of the CDMA code.
3. It performs the inverse FFT on the resulting signal.

This 3-step process is mathematically equivalent to circularly convolving the signal with the code; thus, the output of the inverse FFT will spike at the correct shift that synchronizes the code with the received signal, [6]. The computational complexity of this approach is $O(n \log n)$. For the past two decades, this has been the algorithm with the lowest computational complexity for synchronizing a GPS receiver.

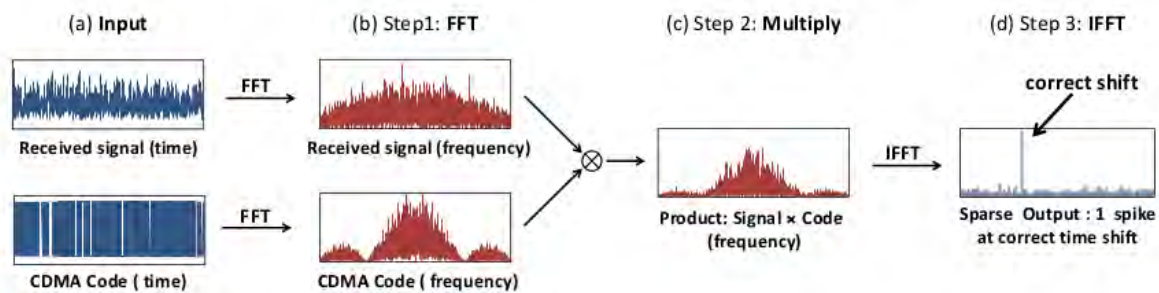


Figure 12 Steps performed by the FFT synchronization algorithm

Going deeper into this subject we have that *The Correlation Theorem* states that the Fourier transform of a correlation integral is equal to the product of the complex conjugate of the Fourier transform of the first function and the Fourier transform of the second function. In our case we perform the complex conjugate of the CDMA code, which is a signal that we already have and that is ready for processing. Performing this operation on the input signal will induced substantially high calculations and more power consumption in embedded applications.

The blocks only have a setting parameter and it is for the sampling frequency of the acquisition method to work, this parameter must be similar to the parameters selected for the 4.6- C/A Code Generator Block and 4.8 Baseline Parallel Code Search (Doppler Correction Included).

The block implements the Parallel Code Search Acquisition method of a real time raw GPS signal. The block receives two input parameters, where signal is the raw GPS signal and C/A code is the local generated code used as a method of comparison.



Figure 14 Baseline Parallel Code Search list of parameters

The block has several parameters to achieve its final functionality, the Signal Size is the size of the signal in samples that would be passed to the code, this value have to be an integer multiple of (sampling frequency/1000). The intermediate frequency, expressed in Hertz, is the frequency to which the carrier frequency is shifted as an intermediate step in transmission or reception.

The sampling frequency defines the number of samples per unit of time (usually seconds) taken from a continuous signal to make a discrete signal. The Desired Satellite parameter defines the satellite that the user wishes to target in the simulation process if it is expected to be on the raw signal.

4.9- Frame Based Signal Generator (Any Sampling Frequency Figure 13 shows the details of the block internal configuration, which performs a circular convolution through a frequency domain implementation.

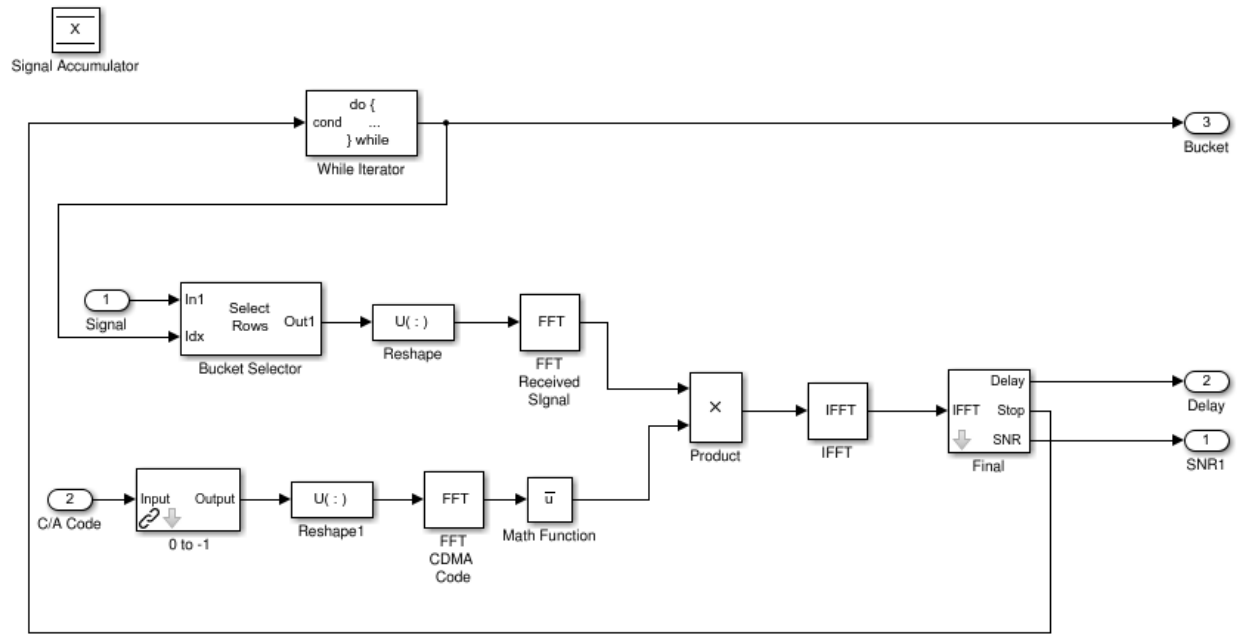


Figure 13 Code Phase Acquisition internal sub-blocks

4.8 Baseline Parallel Code Search (Doppler Correction Included).

The block implements the Parallel Code Search Acquisition method of a real time raw GPS signal. The block receives two input parameters, where signal is the raw GPS signal and C/A code is the local generated code used as a method of comparison.

Parameters	
Signal Size in Samples	572880
Intermediate Frequency(Hz)	9548000
Sampling Frequency(Hz)	38192000
Desired Satellite PRN	21

Figure 14 Baseline Parallel Code Search list of parameters

The block has several parameters to achieve its final functionality, the Signal Size is the size of the signal in samples that would be passed to the code, this value have to be an integer multiple of (sampling frequency/1000). The intermediate frequency, expressed in Hertz, is the frequency to which the carrier frequency is shifted as an intermediate step in transmission or reception.

The sampling frequency defines the number of samples per unit of time (usually seconds) taken from a continuous signal to make a discrete signal. The Desired Satellite parameter defines the satellite that the user wishes to target in the simulation process if it is expected to be on the raw signal.

4.9- Frame Based Signal Generator (Any Sampling Frequency).

This block is one of the most important in the library since it models the noisy, multi-interference signal, which will be later processed by the SDR-receiver. There are several features that make this block a more realistic implementation. The block is frame-based and it can be plugged into the simulation for further correlation operations if the simulation is forced to run only once, this is effectively a simulation requirement when using it. Since the block generates the signal using for iterators subsystems, the time used for iterations is triggered by the internal clock of the “for subsystem” and thus it is independent of the simulation clock time⁵.

The block proceeds in three phases as follows:

Phase 1. See Figure 15 for details

1. Starts by generating the PRN codes of 10 satellites (which is an approximation of the average satellites in view at a time) by using the previously described blocks of PRN Generation. Figure 3, Figure 4 and Figure 5.
2. Perform the upsampling operation by the value specified in the dialog parameter. The block would interpolate the samples and increase the signal size to the desired sampling frequency.
3. Change the sequence to the polar non-return to zero representation by changing each 0 value by -1.
4. Perform a circular delay on the sequence by the value specified for each satellite. Those values are specified in the dialog parameter of the block and are discussed later.
5. Sums up all the signals generated with the steps described before to create a single signal of 1023 chips long.

Phase 2. See Figure 16 for details

1. The block adds white Gaussian noise to the signal by using the Simulink AWGN block. In this stage the input signal is channelized to add the noise and several segments or “buckets” are created from the original input signal. At the moment a total of 40 buckets were created in the signal representation after the noise addition. As a result each bucket presents a different amount of noise.

⁵ For more information on how Simulink simulation works, please refer to the Mathworks documentation on the subject or any Simulink Manual. This manual assumes the reader is familiarized with MATLAB and Simulink environments.

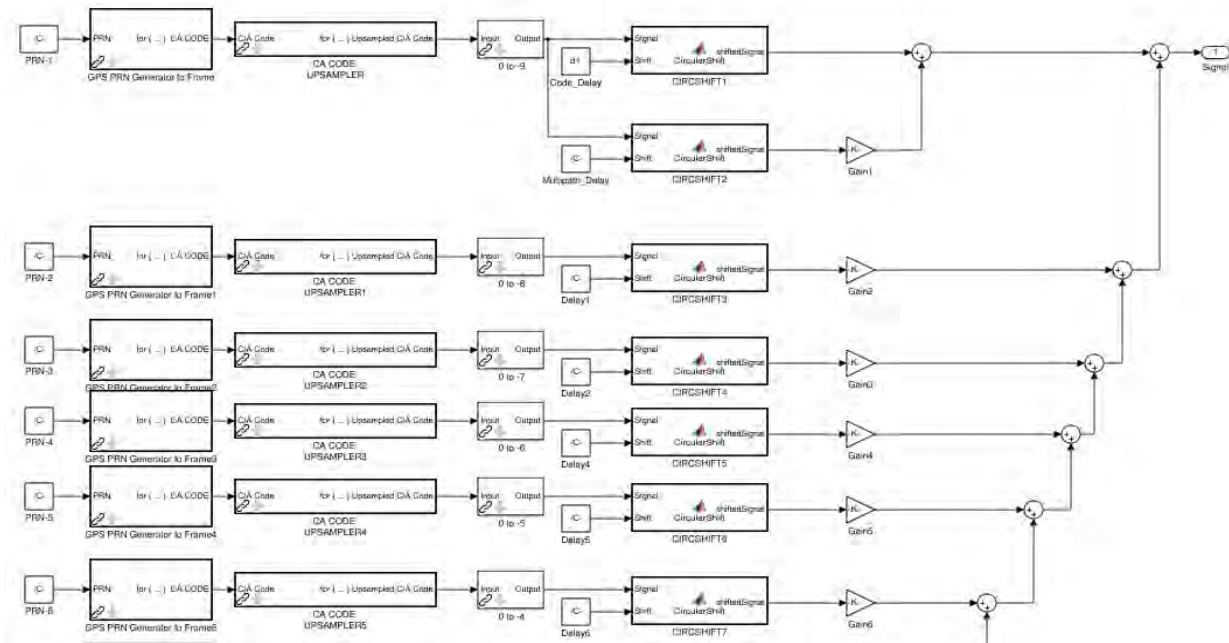


Figure 15 Phase 1 of the signal modeling block

Phase 3. See Figure 17 for details

1. The last phase is just a utility tool to match the block with the previously developed structure. It just reshapes and transposes the signal to get a matrix of 40x1023, which represents 40 buckets of the incoming signal to be processed.

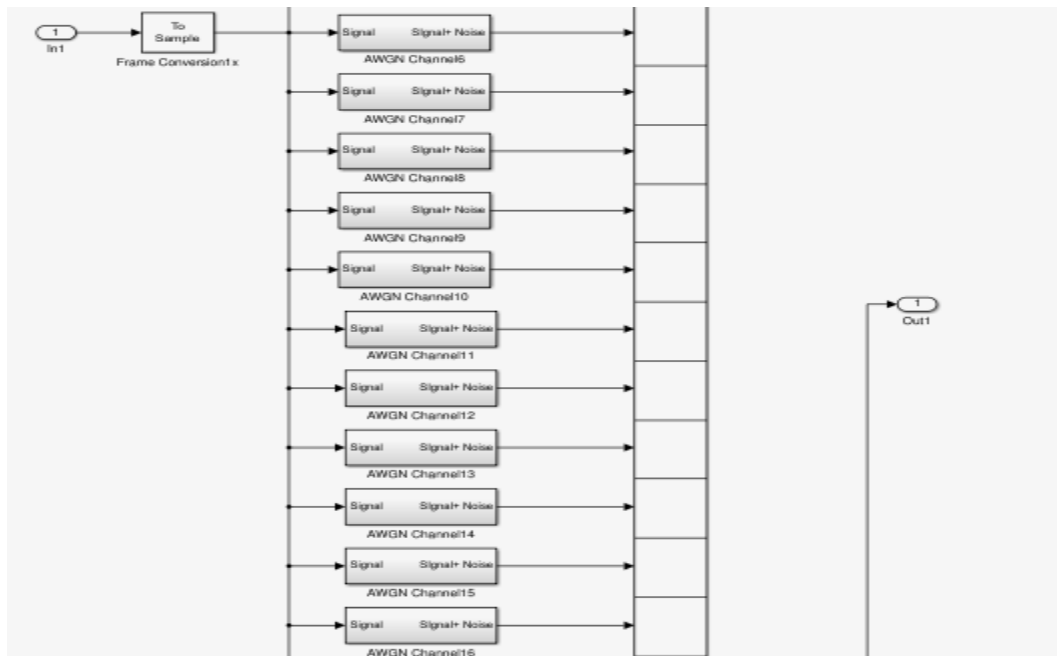


Figure 16 Phase 2. AWGN addition

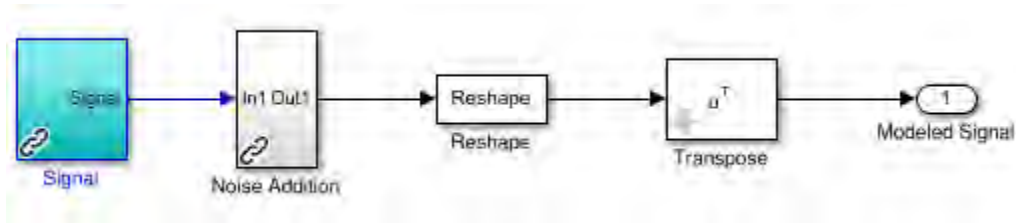


Figure 17 Phase 3. Signal reshaping stage

The block has several parameters modifiable from the dialog parameters as shown in Figure 18, which allows changing the block structure. These parameters modify important aspects of the signal generation and are described below:

1. List of Satellites (1:9): It is a vector based parameter that holds up to 9 satellites PRN numbers. The first number entered would represent the target satellite of the simulation.
2. Desired Satellite Delay: The value holds the satellite delay or phase delay for the PRN specified in the first position of the previous Dialog parameter.
3. Interference Delay Vector (1:9)*First entry is for multipath delay: These values hold the phase delay for the rest of the interfering satellite signals, where the first value indicates the multipath delay for the target satellite.
4. Interference Gain Vector (1:9)*First entry is for multipath attenuation: These values hold a weight vector to multiply each of the generated vectors by an attenuation coefficient.
5. SNR (dB) for AWGN: This value specifies the Signal to Noise Ratio that would be affecting the incoming signal.
6. Desired Sampling Frequency (Hz): This value specifies the sampling frequency for the model.

Parameters

List of Satellites(1:9)

Desired Satellite Delay

Interference Delay Vector(1:9) *First entry is for multipath delay

Interference Gain Vector(1:9) *First entry is for multipath attenuation

SNR (dB) for AWGN

Desired Sampling Frequency(Hz)

Figure 18 List of parameters for the block

5. Using the library. Installation and useful tips

5.1- Overview

A block library is a collection of blocks that serve as prototypes for instances of blocks in a Simulink® model. Simulink comes with two built-in block libraries: the Simulink block library and the Simulink Coder™ block library. The work described in this document is to be included in the Simulink block library, with the single distinction that the library is not built-in inside Simulink. These kind of blocks may be denoted as a user-defined library blocks.[7]

Block libraries, built-in or user-defined are a useful componentization technique for:

1. Providing frequently-used, and seldom changed, modeling utilities.
2. Reusing components repeatedly in a model or in multiple models

A custom library is used identically as any other library. The library needs to be opened in order to be used. This is achieved in the same way as for a model. Once opened, blocks are dragged from a library and placed into a model in the usual way

Simulink provides a Library Browser that you can use to display block libraries, search for blocks by name, and copy library blocks into models. All installed libraries appear in the Library Browser when you open it. The Library Browser is a convenient tool for grouping all available libraries, hence when a custom library is created the developer will usually want to add it to the Library Browser.

5.2- Adding a Custom Library to the Library Browser

Adding a library to the Library Browser is achieved by writing a MATLAB function called `slblocks` (in a file called `slblocks.m`). The function `slblocks` must have a specific format, and `slblock.m` must be in a directory that is on the MATLAB path. The sample code below shows the traditional file in question and how it should be adapted to fit the user needs. It assumes that the library has been saved in a file called `customlib.mdl`. There are two somewhat distinct parts to `slblocks`: the first part specifies what is displayed in the Library Browser; and the second part specifies what is displayed when the library is viewed in an older style Blocksets and Toolboxes view

```
function blkStruct = slblocks
%SLBLOCKS Defines a block library.

% Library's name. The name appears in the Library Browser's
% contents pane.

blkStruct.Name = ['UndergraduateResearch' sprintf('\n') 'Library'];

% The function that will be called when the user double-clicks on
% the library's name.

blkStruct.OpenFcn = 'UndergraduateResearch';
```

```
% The argument to be set as the Mask Display for the subsystem. You
% may comment this line out if no specific mask is desired.
% Example: blkStruct.MaskDisplay = 'plot([0:2*pi],sin([0:2*pi]));';

blkStruct.MaskDisplay = "";
% End of blocks
```

Code Listing 3 Code for the slblock.m file implementing the described library

Once slblocks.m has been written it must be saved and the Library Browser must be closed and restarted for the custom library to be seen in the browser.

5.3- Features of a library

Although libraries have a very similar look and feel to a model they do have some distinct features. These include:

1. Libraries do not get simulated.
2. A library is by default locked. This means that it cannot be unknowingly changed/alterd by a user -- the user must specifically tell Simulink that they know they are making a change. This protects library blocks from mindless or accidental tampering. (When a user tries to make a change to a locked library they receive a message asking them if they want to unlock the library or if they want to discard the changes they are trying to make.)
3. When used in a model a block taken from a library is linked back to the library. This means that all instances of the block used in all models are guaranteed to have exactly the same implementation. In terms of topology they are in fact exactly the same block. Note that each instance of the block may have different parameters. A significant advantage of being linked is that changes made to the library block will propagate to all instance of the block in all models that use the block.

5.4- Tips of usage

1. Some of the code presented in the library may require the installation of a C compiler to work properly in newer version of Mathworks products. Documentation is available on the Internet regarding this issue and it may be easily solved following the directions of the Mathworks Documentation Center.
2. Most blocks on the library are designed to work as frame, for that reason the simulation time parameter when executing code must be set to a fixed time step, with a total of 1 time step executed at any single simulation. Blocks were designed using internal “For” iteration loops that are in charge of all looping requirements.

6. Parallel Structures Overview

This part of the document, and until section 10, proposes alternative architectures to perform a circular correlation using the Fast Fourier Transform (FFT) by decomposing the initial circular correlation into several smaller circular correlations. Such sub-correlations are independent of each other and can be processed in parallel. Even though the results discussed in this document may be applied to any system that performs circular convolution or correlation, we will apply it to the acquisition of Global Navigation Satellite System (GNSS) signals using a, FFT-based, Parallel Code-phase Search (PCS) on the GPS L1 C/A signal. The parallel approach has advantages for hardware-based implementations using Field Programmable Gate Array [8].

The proposed approach is a modification of the main methods of parallelization described in [8], which were previously described in [9] and [10]. Therefore these architectures can be seen as complementary to the ones formulated in [8]. The presented architectures do not need the use of numerically controlled oscillators for multiplication in frequency as described in [8]. The trade-off is the use of a circular shift in time and an extra FFT block, which can be computed in parallel with the others. This could be advantageous depending on the overall implementation strategy regarding speed and accuracy versus resources.

7. Circular Correlation using the FFT

The circular correlation between two finite-length sequences $x[n]$ and $h[n]$ (corresponding to the input signal and code replica in our case, respectively), is defined by Equation (1), where N is the number of samples in one period of the PRN code, \star denotes circular correlation (mod N) and $*$ denotes the conjugate.

$$y(n) = x(n) \star h(n) = \sum_{k=0}^{N-1} h^*(k)x((n+k) \bmod N) \quad (1)$$

In the discrete frequency domain equation (1) can be expressed as:

$$\begin{aligned} DFT[y[n]] &= DFT[x[n]]DFT^*[h[n]] \\ Y(k) &= X(k)H^*(k) \end{aligned} \quad (2)$$

If we use the FFT algorithm to implement the DFT and using the fact that the conjugate of the FFT of a sequence is equal, except for a scaling-by- N factor, to the IFFT of the same sequence [8], then the resulting signal after circular correlation via frequency domain can be expressed as:

$$y[n] = IFFT[FFT[x[n]]IFFT[h[n]]] \quad (3)$$

8. Metric Definition for Complexity Assessment

In order to be able to fully describe the intricacy of the proposed structures a complexity metric notation was defined based on the one proposed in [8].

1. The architecture complexity should be characterized by R-F-N-M-S-NCO as described below.

Table 4 Complexity Metrics

Assigned Code	Meaning
R	Ratio of the processing time of the traditional architecture to the architectures described in later sections
F	Number of FFT operations (or IFFT) required by the architecture.
N	The FFT size. N denotes the size of the full signal in a serial architecture while N/2 is half the original size and it is used in the possible parallel implementations.
M	Number of vector complex multipliers required by the architecture
S	Number of vector adders required by the architecture.
NCO	Number of numerically controlled oscillators.

2. The baseline reference would be the tradition PCS method implemented via a direct serial architecture in the frequency domain, which would measure: 1-3-N-1-0-0, based on the proposed metric.

9. Decomposition of Circular Correlation into Smaller Sub-Correlations

Circular correlation is again defined as in Equation (1) which in the frequency domain, it becomes Equation (2):

$$y(n) = \sum_{k=0}^{N-1} h^*(k)x((n+k) \bmod N) \quad (4)$$

$$Y(k) = X(k)H^*(k) \quad (5)$$

Following any of the equivalent procedures shown in [8], [9] or [10] we obtain a decomposition of circular correlation based on the even and odd components of each sequence in the frequency domain.

The even and odd components can be obtained through a stride-by-2 permutation of the original sequences. The circular convolution in the frequency domain is defined in Equations (4).

$$\begin{aligned}
 Y_0(k) &= X_0(k)H_0^*(k) + X_1(k)H_1^*(k) \\
 Y_1(k) &= e^{j\frac{2\pi}{(N/2)}k} X_0(k)H_1^*(k) + X_1(k)H_0^*(k)
 \end{aligned} \tag{4}$$

Using the IDFT we can obtain the even and odd components of the circular convolution in the time domain, as expressed in Equations (5).

$$\begin{aligned}
 y_0(n) &= IDFT\{X_0(k)H_0^*(k) + X_1(k)H_1^*(k)\} \\
 y_1(n) &= IDFT\{e^{j\frac{2\pi}{(N/2)}k} X_0(k)H_1^*(k) + X_1(k)H_0^*(k)\}
 \end{aligned} \tag{5}$$

It is clear that, because of the multiplication by a complex exponential, this scheme requires the use of a numerical controlled oscillator [8]. Since multiplication by a complex exponential in discrete frequency domain corresponds to a circular shift in the discrete time domain, we will later propose a complementary or alternative architecture based on the fact that the multiplication in frequency by a complex exponential can be translated to a circular shift in the discrete time domain [9].

10. Parallel Circular Correlation Structures

When doing FPGA-based circular correlation the main reason to choose a parallel approach is to have a better throughput, to be able to correlate larger sequences or to address power consumption issues. This is because the parallel signal's length will be half (or shorter) than the original signal length used by a serial architecture. The comparison or the advantages from one structure to another are not measured in performance gains alone but with respect to their computational complexity, use of hardware resources and power consumption.

To have an optimal algorithm it is important to maintain a reduced number of multipliers. It is better to do operations with adders than using multipliers due to their higher computational cost [8]. Nonetheless, algorithms that required an exceedingly large number of adders to offset the savings in the number of multipliers are not favored either. Other issues are related to the number of FFT stages, and whether they are computed in parallel or not, the use of numerical controlled oscillators (NCO) when needed, and the number of serial stages that could impact the architecture latency. See Table 7 for additional information.

10.1- Baseline Parallel Code Search (PCS)

The propagation time is obtained using a synchronization algorithm that allows the device to lock on the received signal for each of the GPS satellites seen by the antenna of the receiver. Each satellite has its own CDMA code, called the C/A code, which consists of 1023 chips that identifies each one [5]. Due to the propagation delay the signal arrives shifted to the receiver, this shift is the exact amount of time that took the signal to arrive at the device. The correlation of the shifted version of the satellite's code with a locally generated code produces a maximum spike in the correlation result whose location gives the propagation delay. Such delay is used to compute the distance of the receiver to each satellite. The geographical location can be then computed by a trilateration process. The receiver does not have a clock as precise as the GPS satellites have, but the accuracy of this calculation can be improved with the 4th satellite used in the trilateration process.

The figure below shows the output results of the method by using the blocks described in the GPS library presented as part of the document. The results obtained may be summarized in the table below:

Table 5 Baseline Architecture Simulation Output Results.

Simulation Output Results from Simulink Implementation		
SNR	Code Phase Delay (samples)	Number of Buckets
124	379	3

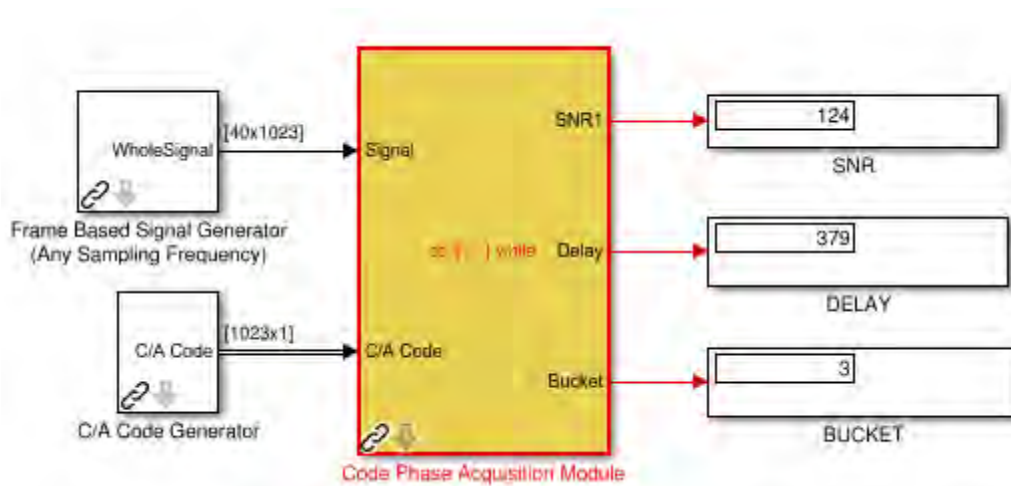


Figure 19 Simulation Output results for Baseline PCS

10.2- Parallel Architecture 1 – Requires the use of a NCO

The first implemented parallel structure follows [8] and is based on the Equations (5) previously described:

$$\begin{aligned}
 y_0(n) &= IDFT\{X_0(k)H_0^*(k) + X_1(k)H_1^*(k)\} \\
 y_1(n) &= IDFT\{e^{j\frac{2\pi}{N}k} X_0(k)H_1^*(k) + X_1(k)H_0^*(k)\}
 \end{aligned}
 \tag{5}$$

Where the multiplication by a complex exponential requires a numerically controlled oscillator (NCO) or a lookup table of some kind, expressing the phases or angles that can be used.

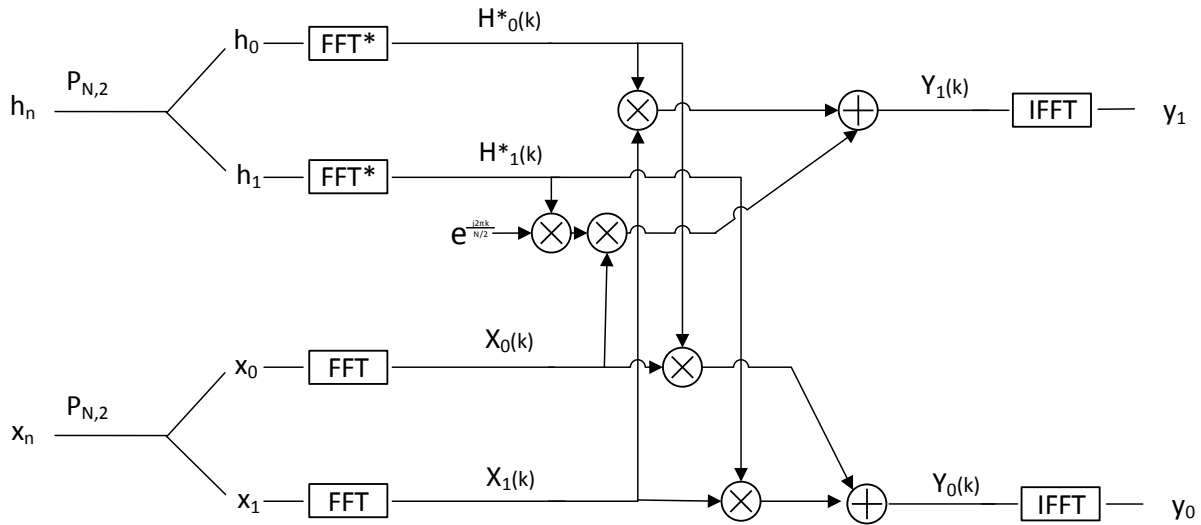


Figure 20 Block diagram of the first parallel architecture implemented.

This first architecture does circular correlation through the frequency domain. The received signal and the generated code are split into two vectors, one containing the entries corresponding to the even indexes of the original vector and the other part containing entries corresponding to the odd indexes. A FFT is performed on these new vectors so that the samples can be expressed in the frequency domain. Note that in order to generate the final correlation result; $y[n]$ from the even and odd components, an inverse stride-by-two permutation has to be performed. This architecture involves 6 FFT operations, 5 multiplications, 2 adders and the use of one NCO. Note that the multiplication by the complex exponential could increase the architecture latency even if it is pre-computed and stored, because it cannot be done in parallel with the other four multiplications.

Table 6. Architecture 1 Simulation Results

Simulation Output Results from Simulink Implementation			
SNR	Code Phase Delay	Number of Buckets	Metric
114.6	379	10	2-6-N/2-5-2-1

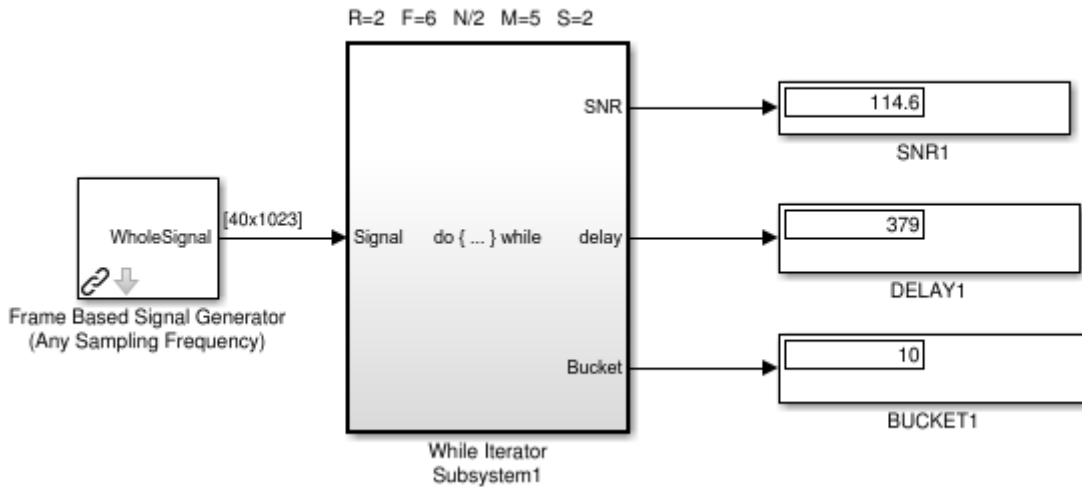


Figure 21 Simulation output results for architecture 2-6-N/2-5-2-1

10.3- Architecture 2 – Requires a circular shift in the discrete time domain.

A well-known property states that multiplication by a complex exponential in the discrete frequency domain corresponds to a circular shift in the discrete time domain. For a length-N sequence it is as follows,

$$\text{DFT}\{h((n - n_0) \bmod N)\} = H(K)e^{j\frac{2\pi}{N}kn_0} \quad (6)$$

Letting $n_0 = 1$ we obtain, $\text{DFT}\{h((n - 1) \bmod N)\} = H(K)e^{j\frac{2\pi}{N}k}$ (7)

This procedure can it is also developed in [9]. Based on this theory we can eliminate the use of a NCO and a multiplier from the first architecture. This variant is achieved when the multiplication by a complex exponential is exchanged for a discrete time circular shift on h_1 followed by a FFT as shown in Figure 22. Unfortunately we still need the FFT of h_1 (without the circular shift) and therefore the tradeoff requires adding an extra FFT, albeit in parallel with the other four FFTs. The remaining 4 multiplication can be done in parallel and the cyclic shift can be routed in hardware without adding to the latency. Its computational complexity would now be 7 FFT, 4 multiplications, 2 adders and 0 NCOs.

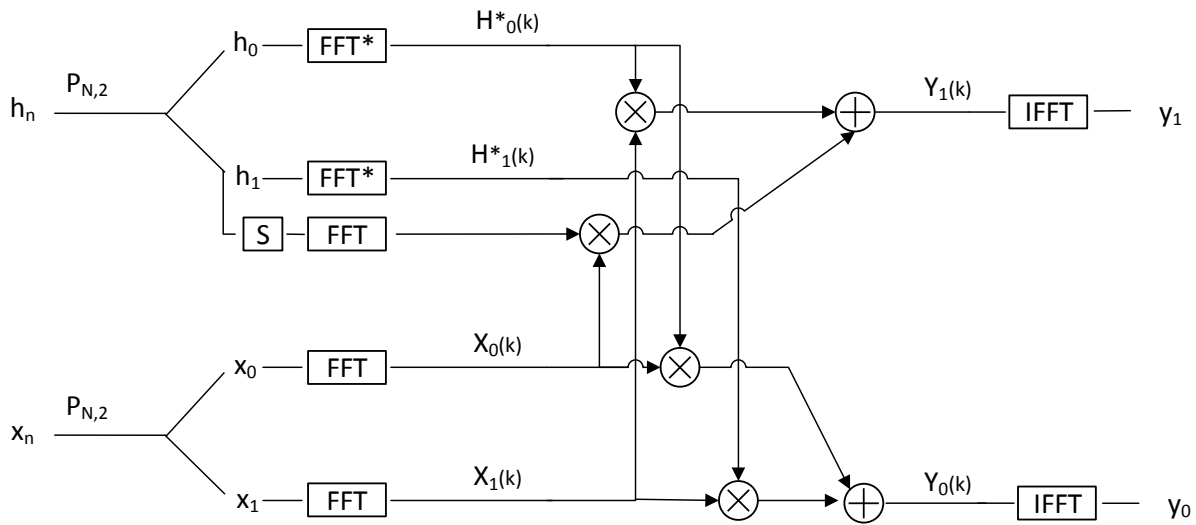


Figure 22 Block diagram of the modified first architecture.

After running this architecture with the same parameters in the Simulink environment we get the following results, summarized in the following table.

Table 7. Architecture 2 Simulation Results

Simulation Output Results from Simulink Implementation			
SNR	Code Phase Delay	Number of Buckets	Metric
124	379	3	2-6-N/2-5-2-0

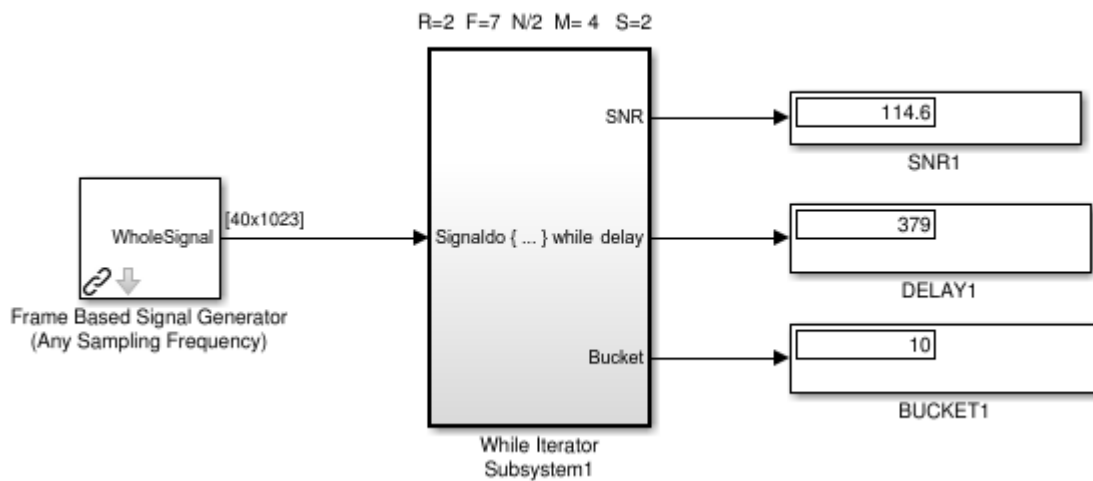


Figure 23 Simulation Output Results for architecture 2-7-N/2-4-2-0

10.4- Architecture 3 – Requires the use of a NCO.

The third architecture is obtained by factorizing as follows the original equations [8]:

$$\begin{aligned}
 Y_0(k) &= [H_1^*(k) - H_0^*(k)] X_1(k) + H_0^*(k)[X_0(k) + X_1(k)] \\
 Y_1(k) &= [e^{j\frac{2\pi}{(N/2)}k} H_1^*(k) - H_0^*(k)] X_0(k) + H_0^*(k)[X_0(k) + X_1(k)]
 \end{aligned}
 \tag{8}$$

A similar factorization can be found in [9] and [10].

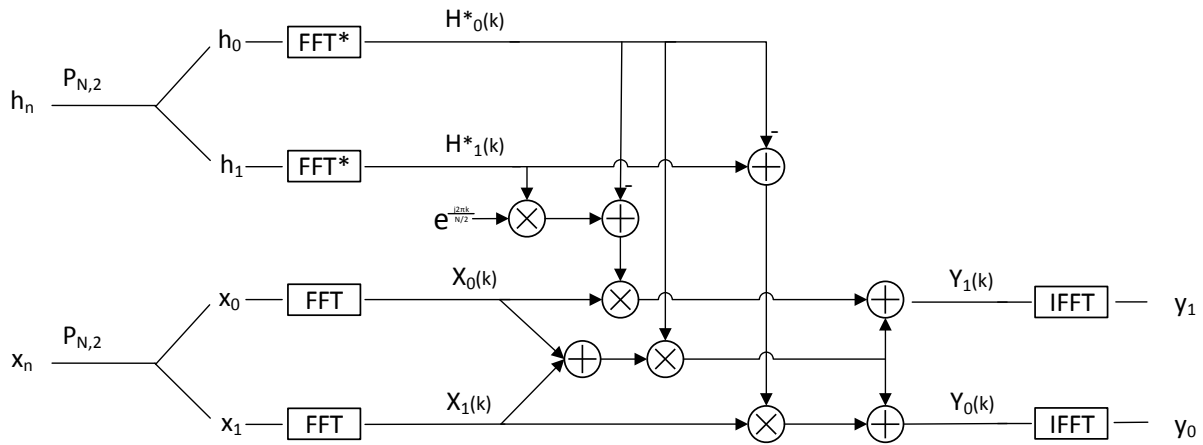


Figure 24 Block diagram of the second implemented architecture

In comparison with the first architecture, this second architecture reduces the numbers of multiplications yet the number of adders used is greater. The structure is particularly different from the first one, as it can be seen in the equations from which the diagram was developed.

Table 8. Architecture-3 Simulink simulation output results

Simulation Output Results from Simulink Implementation			
SNR	Code Phase Delay	Number of Buckets	Metric
114.6	379	10	2-6-N/2-4-5-1

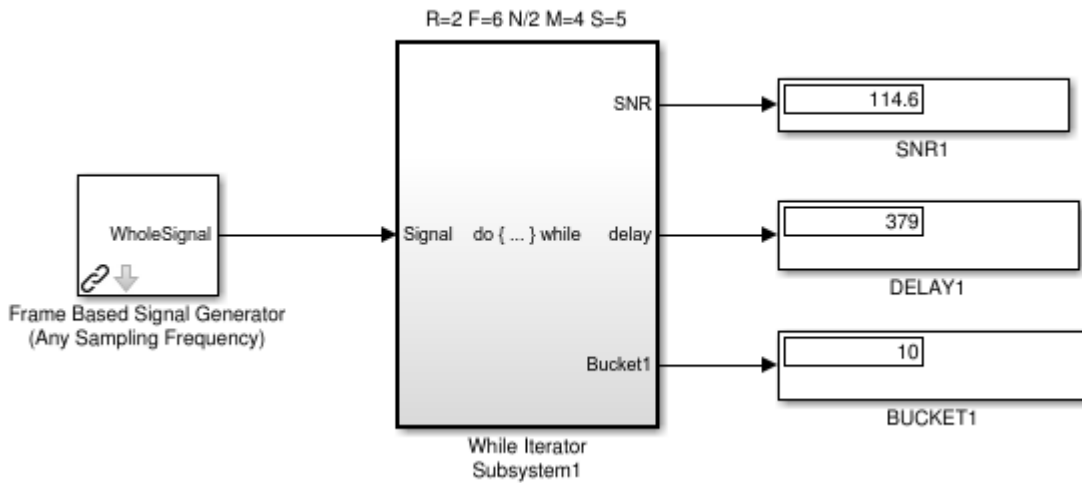


Figure 25 Simulation Output Results for architecture 2-6-N/2-4-5-1

10.5- Architecture 4 – Requires the use of a circular shift in the discrete time domain.

The final architecture is a modified version of Architecture 3. The complex exponential can be omitted to reduce the by one the number, of multiplications and of numerically controlled oscillators.

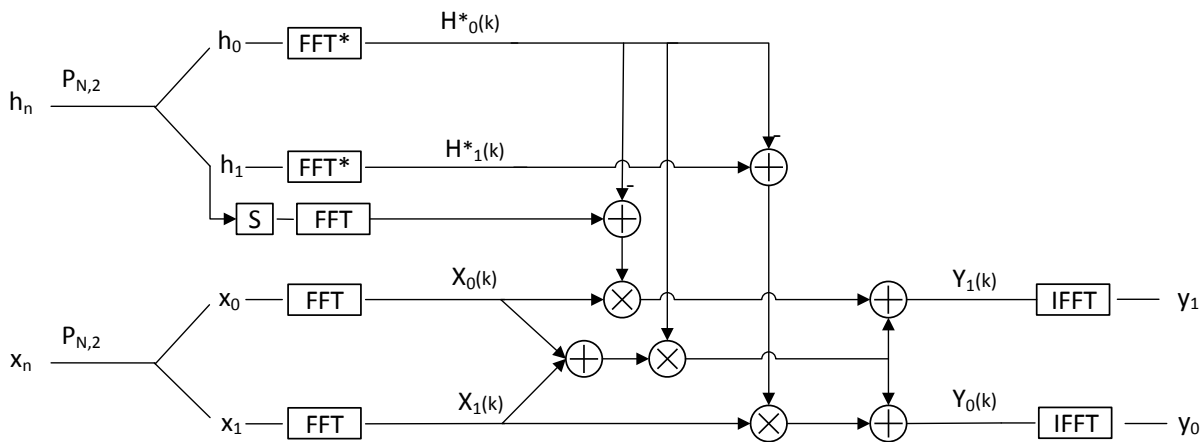


Figure 26 Block diagram of the second modified architecture

Another multiplication, and the use of a NCO, can be omitted if the complex exponential multiplication in frequency is translated to a circular time shift operation on h_1 followed by a FFT. Again we still need the FFT of the un-shifted version of h_1 therefore; the tradeoff requires the use of an extra FFT performed in parallel with the others.

Table 9 Shifted in Time, Architecture-4 Simulink simulation output results

Simulation Output Results from Simulink Implementation			
SNR	Code Phase Delay	Number of Buckets	Metric
114.6	379	10	2-7-N/2-3-5-0

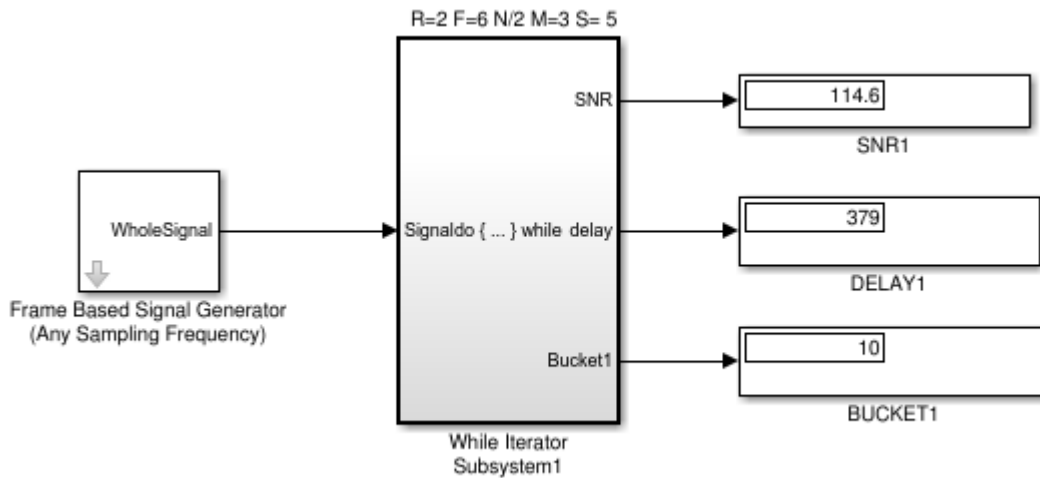


Figure 27 Simulation Output Results for architecture 2-7-N/2-3-5-0

In general terms the proposed architectures were compared for the number of adders, multipliers and FFTs that each one requires in order to parallelize the correlation process.

Table 10 Comparison of the parallel architectures

	First Architecture	First Modified Architecture	Second Architecture	Second Modified Architecture
Radix	2	2	2	2
FFTs	6	7	6	7
Multipliers	5	4	4	3
Adders	2	2	5	5
NCOs	1	0	1	0

Radix-2 architecture will be almost twice as fast as the direct serial architecture [8] if implemented in hardware. The tradeoff is the extended use of hardware resources.

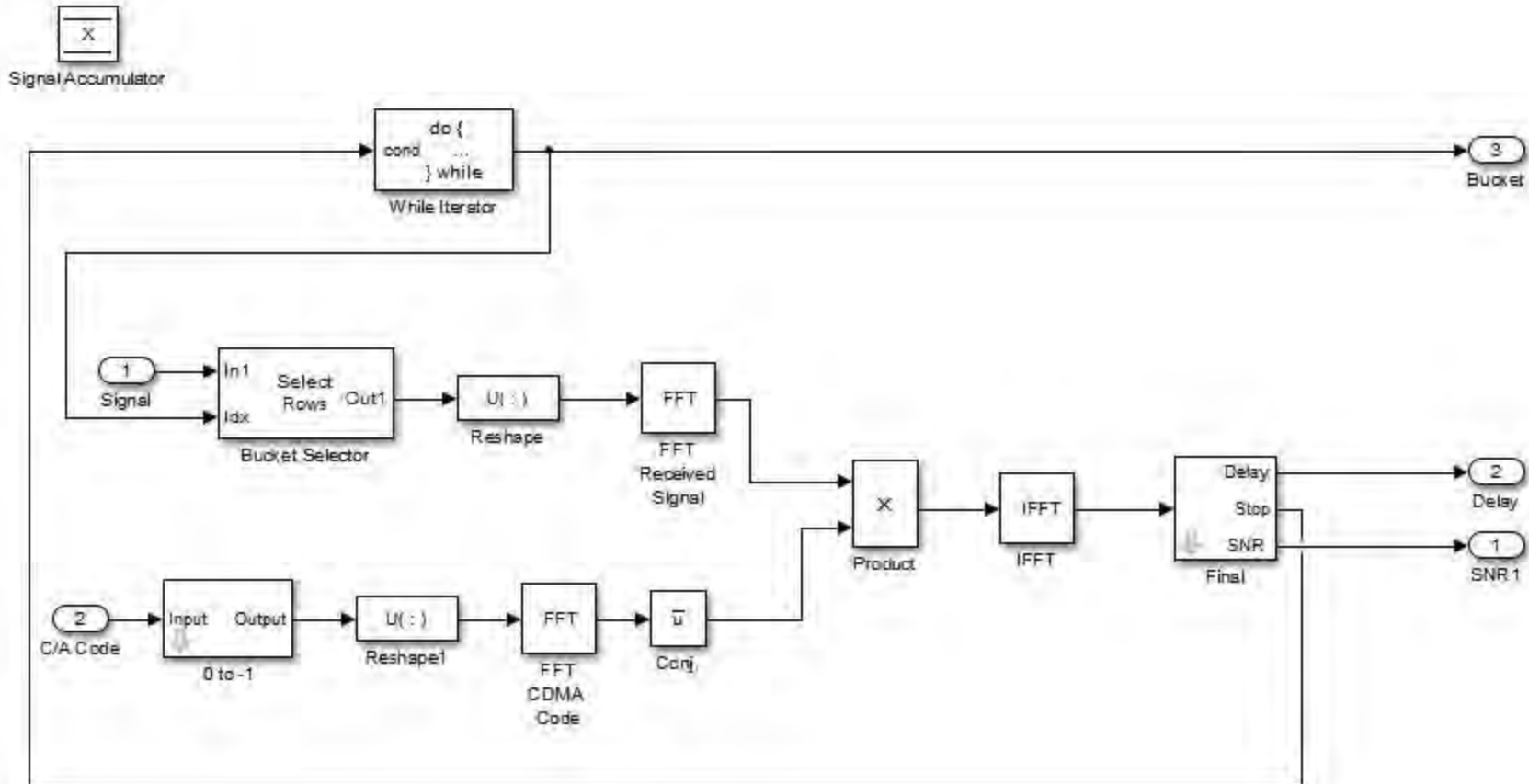
Further architectures based on the Agarwal-Cooley cyclic convolution algorithm are also possible and in the process of being developed.

11. References

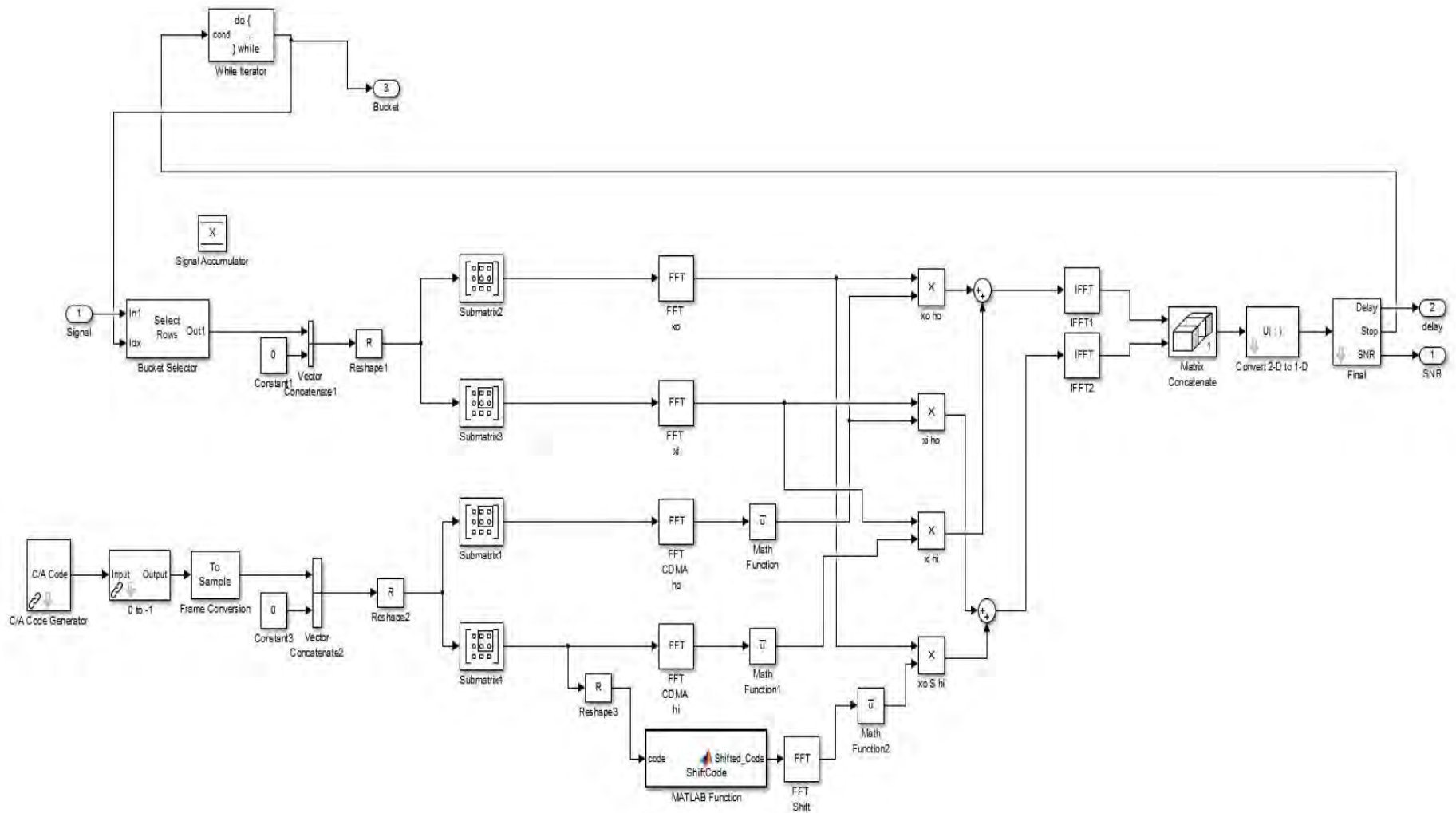
- [1] F. J. Coelho, "Software Defined GPS / Galileo Receiver Master of Science in Electrotechnical Engineering," no. April, 2011.
- [2] C. Jeffrey, An Introduction to GNSS. 2010, p. 96.
- [3] F. Principe, G. Bacci, F. Giannetti, and M. Luise, "Software-Defined Radio Technologies for GNSS Receivers: A Tutorial Approach to a Simple Design and Implementation," *Int. J. Navig. Obs.*, vol. 2011, no. iii, pp. 1–27, 2011.
- [4] New Wave Instruments, "Linear Feedback Shift Registers." [Online]. Available: http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_ifsr.htm. [Accessed: 09-Jan-2014].
- [5] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. Holdt Jensen, A Software-Defined GPS and Galileo Receiver. A Single-Frequency Approach. 2012, p. 175.
- [6] H. Hassanieh, F. Adib, D. Katabi, and P. Indyk, "Faster GPS via the sparse fourier transform," *Proc. 18th Annu. Int. Conf. Mob. Comput. Netw. - Mobicom '12*, p. 353, 2012.
- [7] Mathworks Documentation Center(2013),
- [8] J. Leclère, C. Botteron, and P. Farine, "Improving the Performance of the FFT-based Parallel Code-phase Search Acquisition of GNSS Signals by Decomposition of the Circular Correlation," in *ION GNSS 2012*, 2012, pp. 1–11.
- [9] M. Teixeira and D. Rodriguez, "A new method mathematically links fast Fourier transform algorithms with fast cyclic convolution algorithms," in *Proc. of the IEEE 37th Midwest Symp. on Circuit and Syst.*, Lafayette, LA, Aug. 1994.
- [10] M. Teixeira and D. Rodriguez, "A class of fast cyclic convolution algorithms based on block pseudocirculants," *IEEE Signal Processing Letters*, vol. 2, No. 5, pp. 92-94, May 1995.

12. Appendix

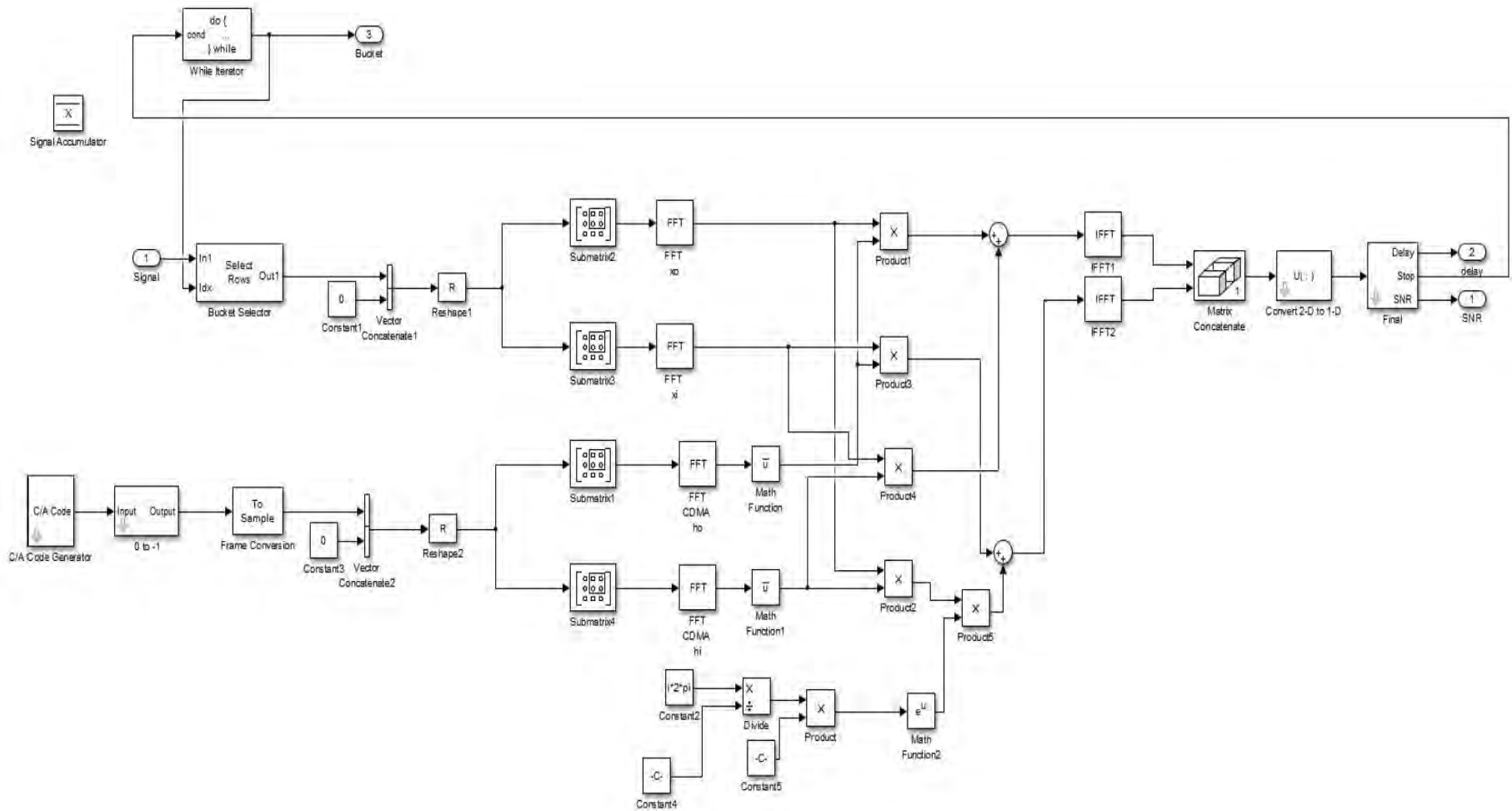
12.1 Simulink Model for Base Line Method



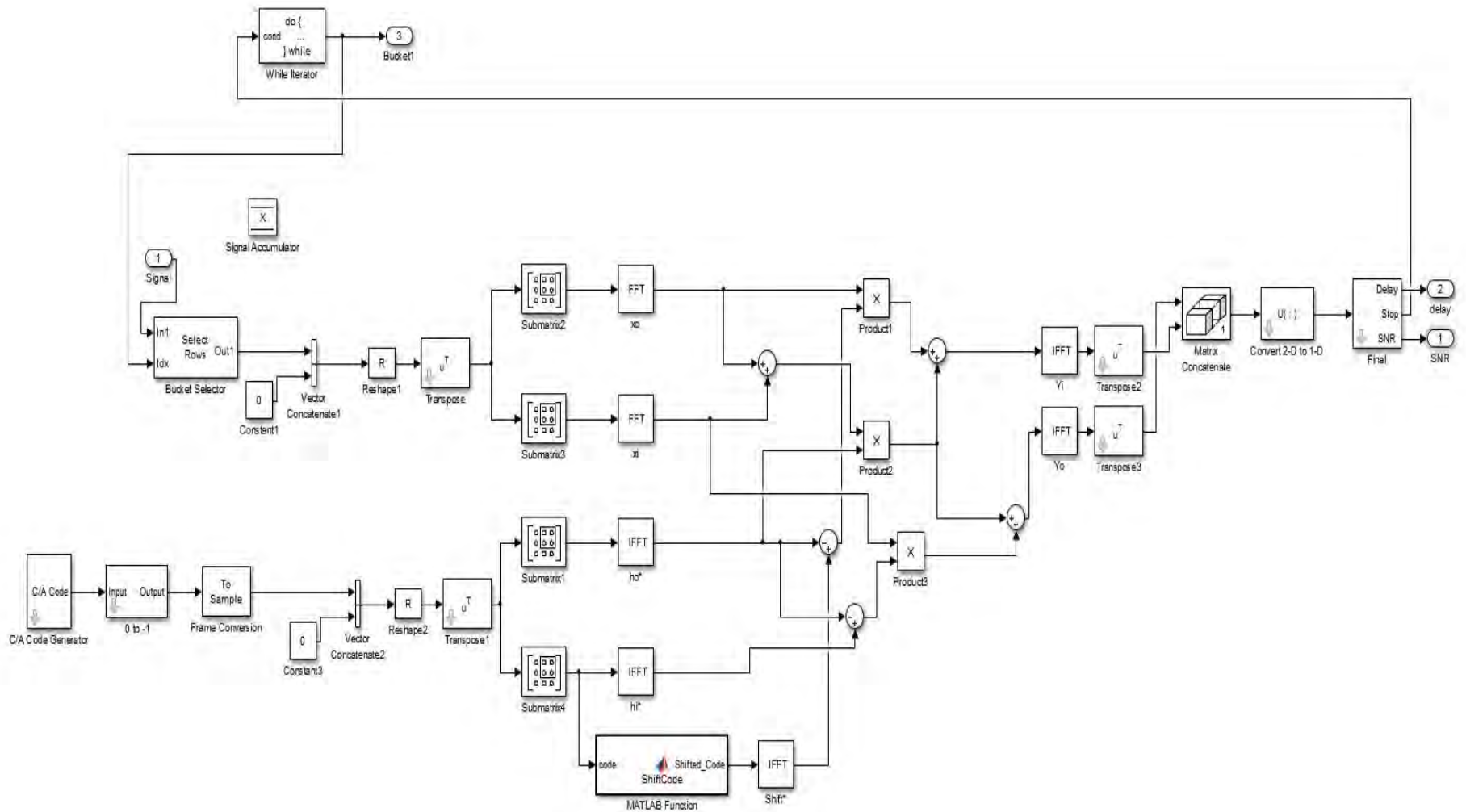
12.2 Simulink Model for Architecture 1



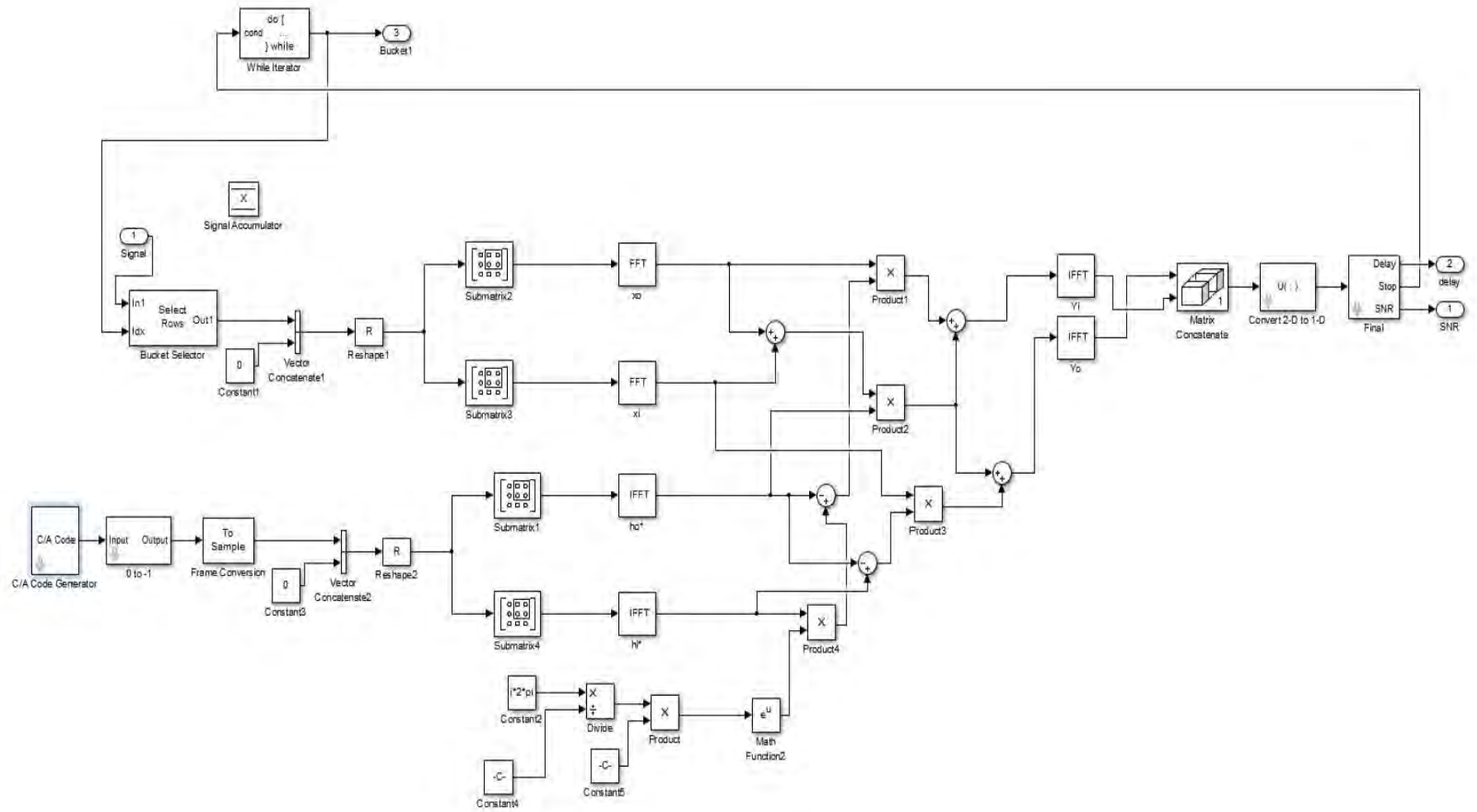
12.3 Simulink Model for Architecture 2



12.4 Simulink Model for Architecture 3



12.5 Simulink Model for Architecture 4



12.6 Matlab Code for a Standard Baseline Implementation Using Basic Modeled Data

```
****GPS PRN Correlator Detector (parallel code phase search)
****doppler search step: 1000 Hz.
% Uses data lms in length (corresponds to 1 PRN) sampled at Fs
*****Originated: Nov 19, 2013 (new name: BL_Model_Data_FH_1.m)
%(former file name: baseline_stand_alone_5j.m and 5)
clear
clc

% Works with modeled data. Code that uses real satellite records is also available.

% ***** DATA SIGNAL MODELING *****

***** SET PULSE, INTERFERENCE AND NOISE PARAMETERS *****
n=1023; %Code Length
k=40 %number of length-n*p buckets that we are listening at
delay=378% synch pulse delay in samples within each bucket
delaymp=80; % multipath pulse delay in samples within the buckets in which it is
embedded
delayint=[80 130 5 280 40 30 100 300]; % interference pulses delay from satellites
interferencegain=[0.7 2 3 0.3 0.9 4 1 3];% gain of interference pulses from
other satellites
multipathgain=0.2 % gain of multipath signal interference
snrinputsignal=-20 %signal to noise ratio in each bucket using AWGN
thresholdsparse=90;
thresholdbaseline=100;

fs = 1023000;
fca = 1023000;

***** PN GOLD SEQUENCE GENERATION for k Satellites *****
satellites=[01 03 07 19 20 22 24 28 31];
% satellites=[6 12 14 4 2 30];
g=cacode(satellites,fs/fca); % generates Gold Code: Function by
pulse=g(1,:); %Define code to be detected
pulse=pulse*2-1;% maps [0,1] values to [-1,1] values
n = length(pulse);

***** GENERATE INTERFERENCE CODES FROM OTHER SATELLITES *****

nint=size(g);
for i=2:nint
pulseint(i-1,:)=g(i,:);
end

% maps [0,1] values to [-1,1] values
for i=1:nint-1
pulseint(i,:)=pulseint(i,)*2-1;
end

% ***** MODELING OF RECEIVED SIGNAL *****
% The desired pulse + noise + interference pulses + multipath will be embedded.

N=(n); %Received signal length in terms k sections of length-n*p each
```

```

signal(N)=0; %length-N signal definition with all 0 entries

signalmain=pulse;
signal=signalmain;
signal=circshift(signal.', delay-1)';

% MULTIPATH INTERFERENCE

signalmultipath1(N)=0; %length-N signal definition with all 0 entries

% startpulsemp=delaymp; %pulse starts with a delay in the bucket where it is
embedded.
% endpulsemp=startpulsemp+n; %pulse ends after starting with certain delay in
bucketnumber
% signalmultipath1(startpulsemp:endpulsemp-1)=pulse; %pulse incorporated within
previous signal range

signalmultipath=signalmain;
signalmultipath=circshift(signalmain.', delaymp-1)';

% INTERFERENCE SIGNALS

%interference signal initialization
for i=1:nint-1
signalint(i,N)=0;
end

%interference pulses
for i=1:nint-1

%Add P=3 interference pulses from each interfering satellite per bucket circularly
shifted
signalmainint=[pulseint(i,:)];
signalint(i,:)=signalmainint;
signalint(i,:)=circshift(signalint(i,:).', delayint(i)-1)';

end

%ADD INTERFERENCE TO SIGNAL

%add interference pulses from other satellites
for i=1:nint-1
signal=signal+interferencegain(i)*signalint(i);
end

%add interference multipath pulse from the satellite we want to synch
signal=signal+ multipathgain*signalmultipath1;

%ADD AWGN TO EACH SIGNAL BUCKET TO FINISH Data Signal Modeling

bucketnp=[];
bucketnp(n,k)=0;

```

```

for i=1:k

    %SELECT MODELED SIGNAL DATA
    bucketnp(:,i)=signal;
    bucketnp(:,i)=awgn(bucketnp(:,i),snrinputsignal);% add AWGN
    wholesignal((i-1)*n+1:i*n)=bucketnp(:,i);
    %wholesignal=wholesignal/max(wholesignal);

end

%This lines are for using the BASELINE method as a STANDALONE method
%Comment otherwise:

%*****
fftpulse=fft(pulse);
conjfftpulse=conj(fft(pulse));

tic
% Method4: BASELINE METHOD using O(nlogn) mults and length-n buckets (TEST CODE)
% Full correlation between pulse and length n-bucket to find actual delay within
bucket
%
corr4(n)=0; % correlation result preallocation
SNRmethod4=1; % SNR method 4 init
x=[];
x(n)=0; %acumulation bucket

for ii=1:floor(length(wholesignal)/n) % divides signal into buckets of length n

% %Correlation using cconv (freq domain)
% corr4=cconv(fliplr(pulse),wholesignal((ii-1)*n+1:ii*n),n);%uses FFT to conv (or
correlate)with each length-n bucket
% x=corr4+x; % accumulates with previous correlation
% [size delaysamples4]=max(x); % find the maximum size and position in accumulated
correlations
% delaymethod4=delaysamples4+1;%adjust delay from FFT cyclic conv result

%Correlation using ifft(fft.fft) (freq domain)
wholesignal = wholesignal((ii-1)*(n)+1:ii*(n));
corr4=ifft(conjfftpulse.*fft(wholesignal));%uses FFT to conv (or correlate)with
each length-n bucket
x=(corr4)+ (x); % accumulates with previous correlation
[size delaysamples4]=max(x); % find the maximum size and position in accumulated
correlations
delaymethod4=delaysamples4;%adjust delay from FFT cyclic conv result

%SIGNAL TO NOISE RATIO Calculation (signal: peak power noise: noise floor power)
xx=x; % auxiliary variable in order to keep the accumulator as it is
xx(delaysamples4)=0; %takes peak out of correlation result
%noisefloorpwr=var(corr); %calculate noise variance %power of noise in output
bucket

```

```
%noisefloorpwr=var(detrend(corr)); %power of noise in output bucket
noisefloorpwr=mean(xx.^2) ;%power of noise bed in output bucket (peak not
included)
SNRmethod4=size^2/noisefloorpwr; % SNR calculation for this method

    if SNRmethod4>thresholdbaseline % threshold comparison

        delaymethod4=delaymethod4

        SNRmethod4=size^2/noisefloorpwr

        buckets_used_baseline=ii

        break
    end

end

baselinetime=toc
```